Timing Analysis of Software Executing on Undocumented Multicore Processors

Bjorn Andersson

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM19-0890

Systems Interact with Their Physical Environment

Systems Include Software

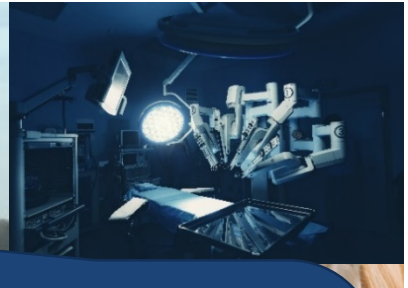
Systems Include Software That Interacts with the Physical Environment

Systems Include Software That Has Real-Time Requirements

Satisfying Real-Time Requirements is a Challenge for These Systems in General

Satisfying Real-Time Requirements is a Challenge for …



"The trick there, when you're processing flight critical information, it has to be a deterministic environment, meaning we know exactly where a piece of data is going to be exactly when we need to — no room for error," Langhout says. "On a multi-core processor there's a lot of sharing going on across the cores, so right now we're not able to do that."

- Jeff Langhout, Acting Director, U.S. Army Aviation and Missile Research Development and Engineering Center (AMRDEC)

Source: "Army still working on multi-core processor for UH-60V," May 2017, Available at https://www.flightglobal.com/news/articles/army-still-working-on-multi-core-processor-for-uh-6-436895/.

Satisfying Real-Time Requirements is a Challenge for …




"A majority of avionics today are running on single-core processors, or multicore processors with all but one core disabled."

"The root of the problem is shared resources, which most of the time creates some kind of interference."

Source: "It's time: Avionics need to move to multicore processors," January 2018, Available at http://www.intelligent-aerospace.com/articles/2018/01/it-s-time-avionics-needs-to-move-to-multicore-processors.html.

Satisfying Real-Time Requirements is a Challenge for …




"In safety-critical domains such as avionics, the multicore "predictability problem" is currently dealt with by turning off all but one core if highly-critical system components exist."

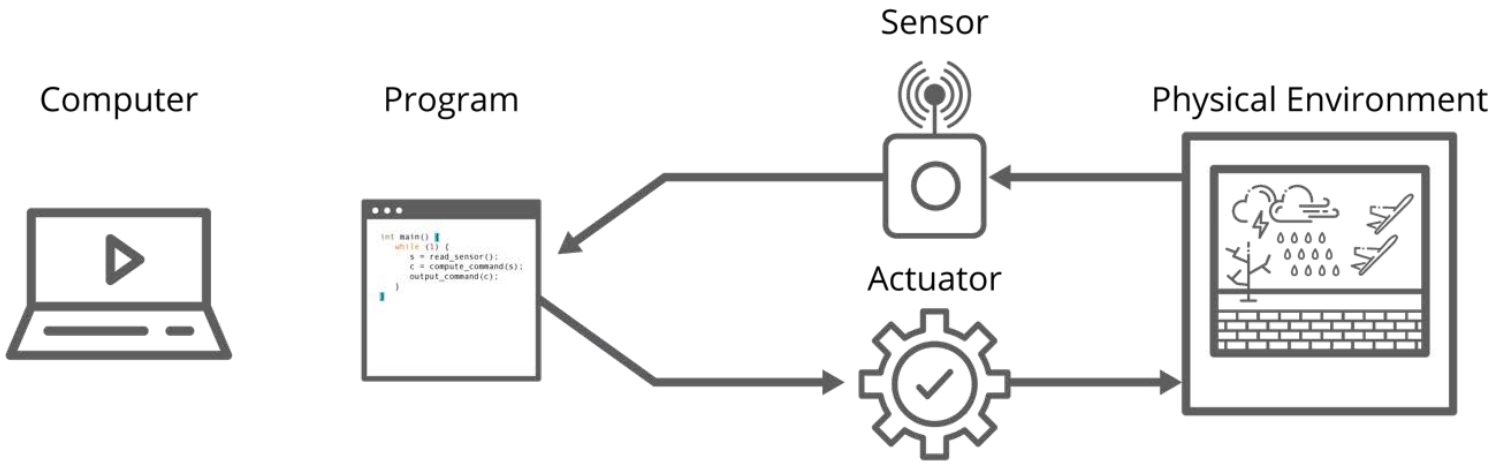
Source: C. J. Kenna et al., "Making Shared Caches More Predictable on Multicore Platforms," RTSS'2012.

Satisfying Real-Time Requirements is a Challenge for …

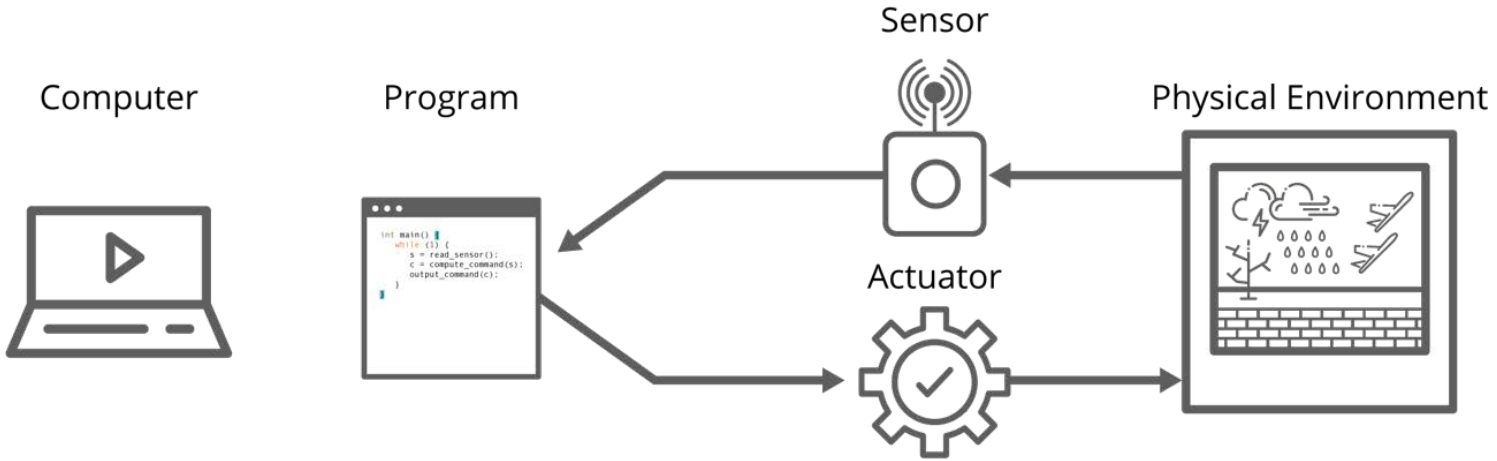
"Currently, avionics manufacturers resolve the multicore "predictability problem" by turning off all but one core if highly critical system components exist."

Source: B. C. Ward et al., "Making Shared Caches More Predictable on Multicore Platforms," ECRTS'2013.

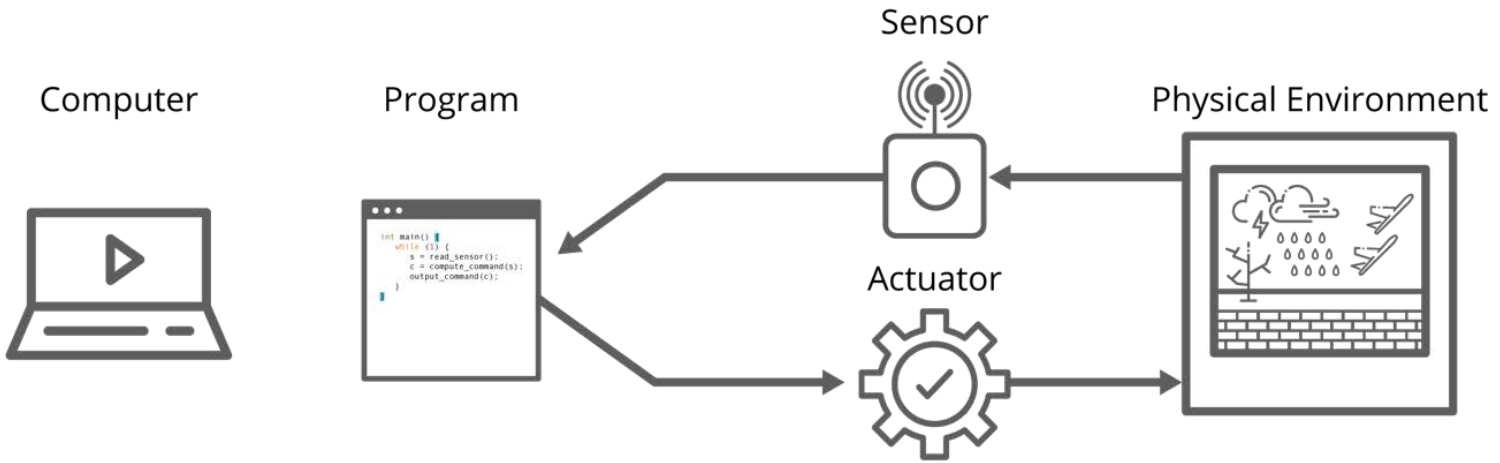
Commonality of these Systems



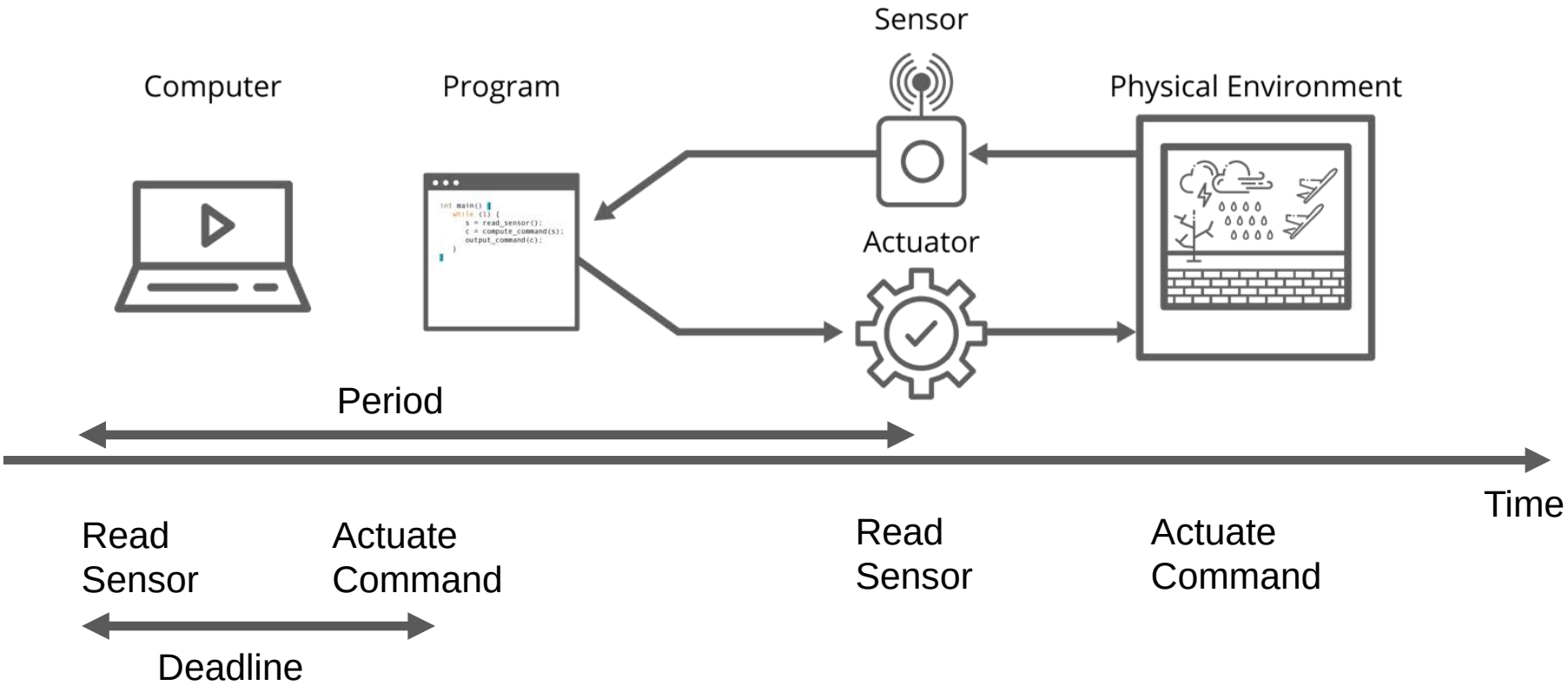
Commonality of these Systems



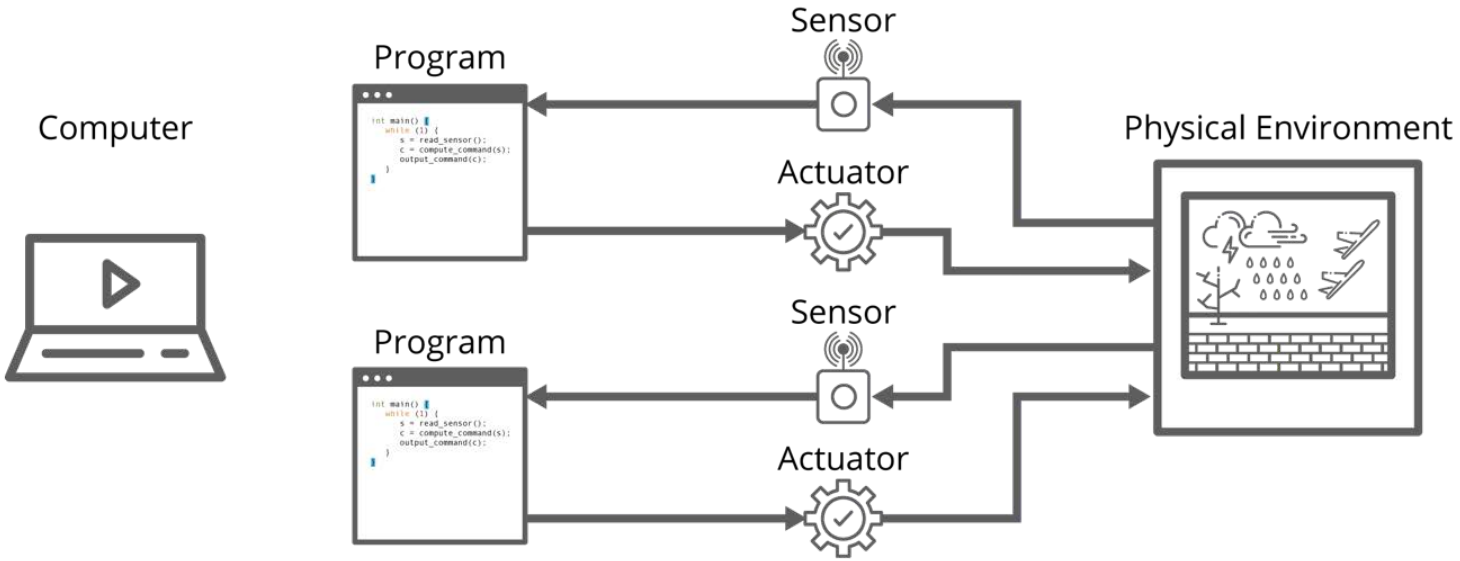
Commonality of these Systems



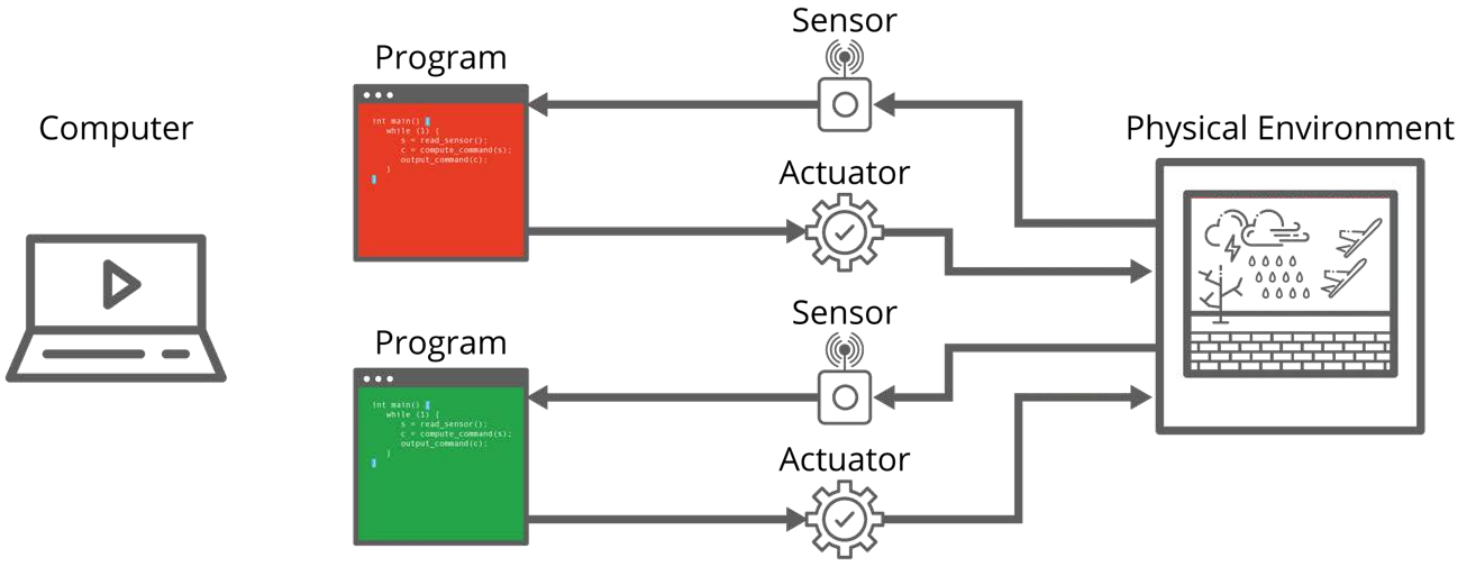
Commonality of these Systems



Commonality of these Systems



Commonality of these Systems



What Makes it Challenging to Satisfy Real-Time Requirements?

What Causes Delay of Software?

What Causes Delay of Software?

What Causes Delay of Software?

Time when one thread
in the software system arrives

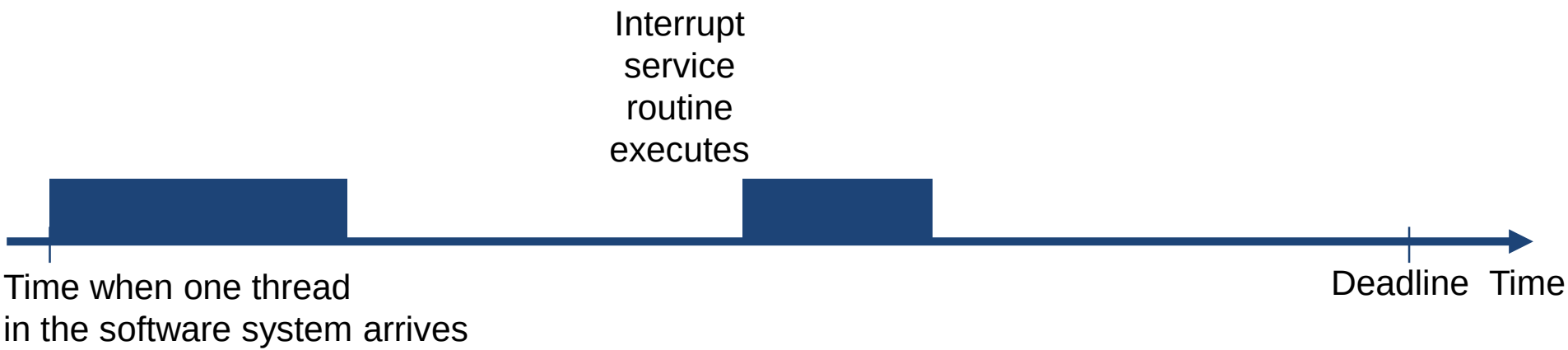
Deadline Time

What Causes Delay of Software?

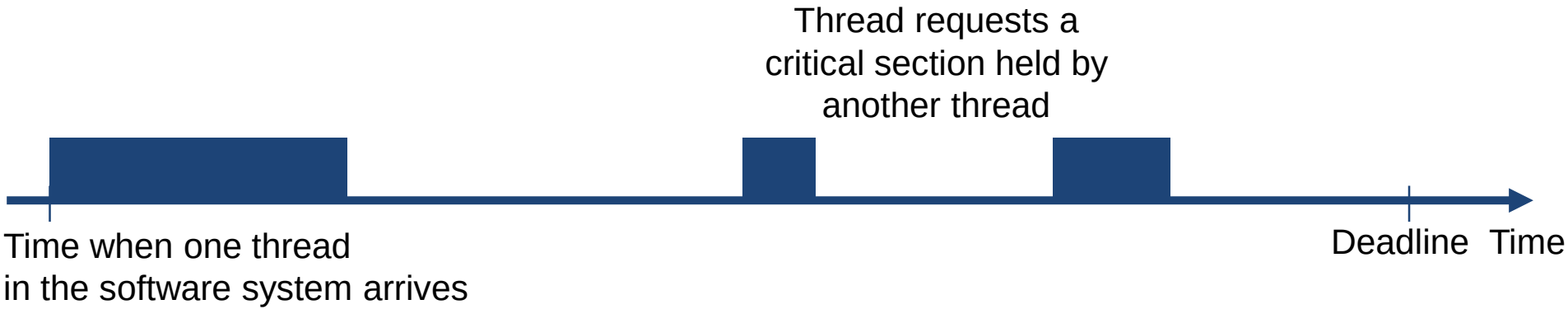
What Causes Delay of Software?

What Causes Delay of Software?

What Causes Delay of Software?

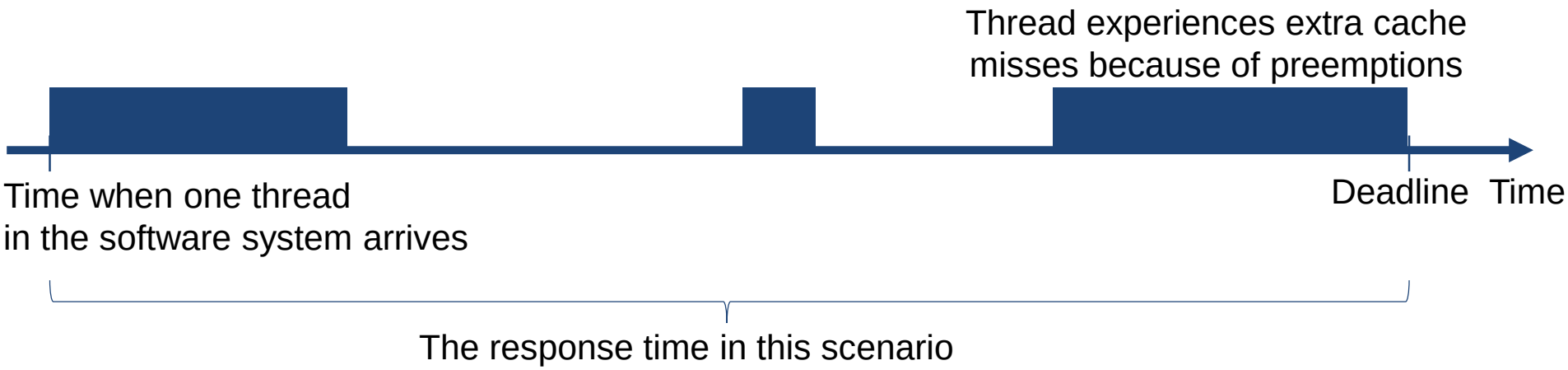


What Causes Delay of Software?



What Causes Delay of Software?

What Causes Delay of Software?

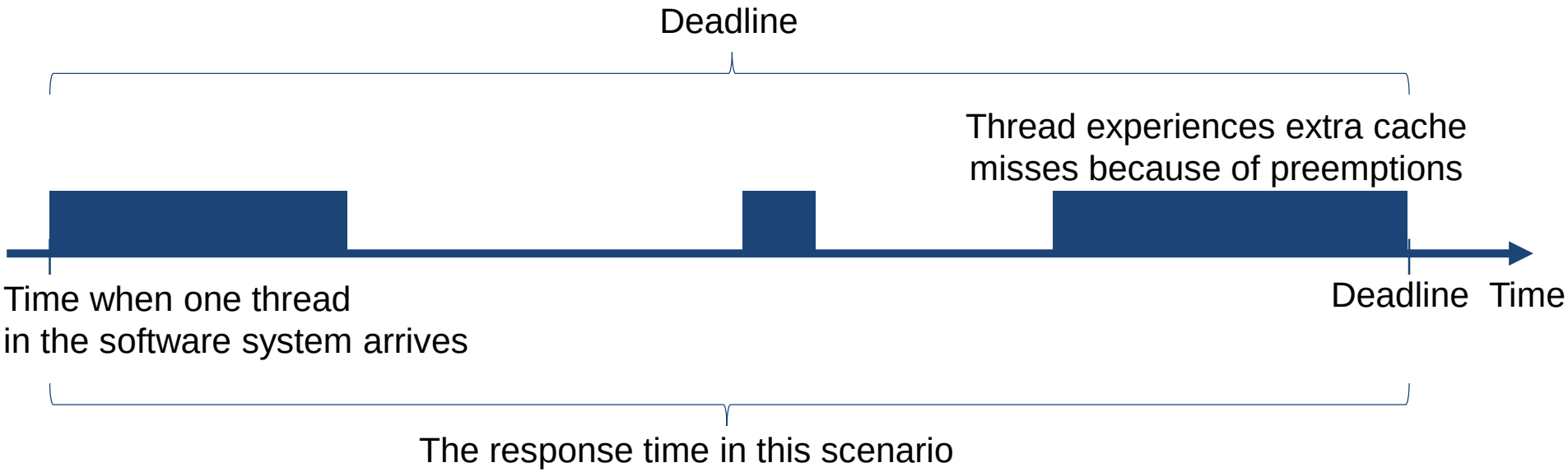


What Causes Delay of Software?

How large can the response time be?

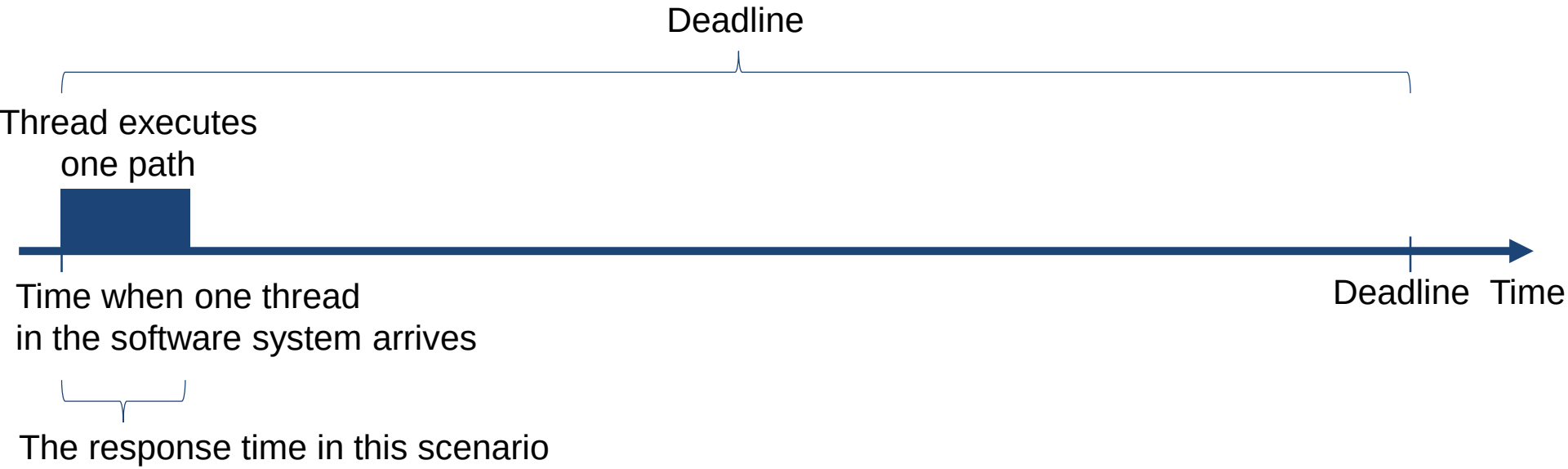
What Causes Delay of Software?

Does it hold for all scenarios that the response time is at most the deadline?



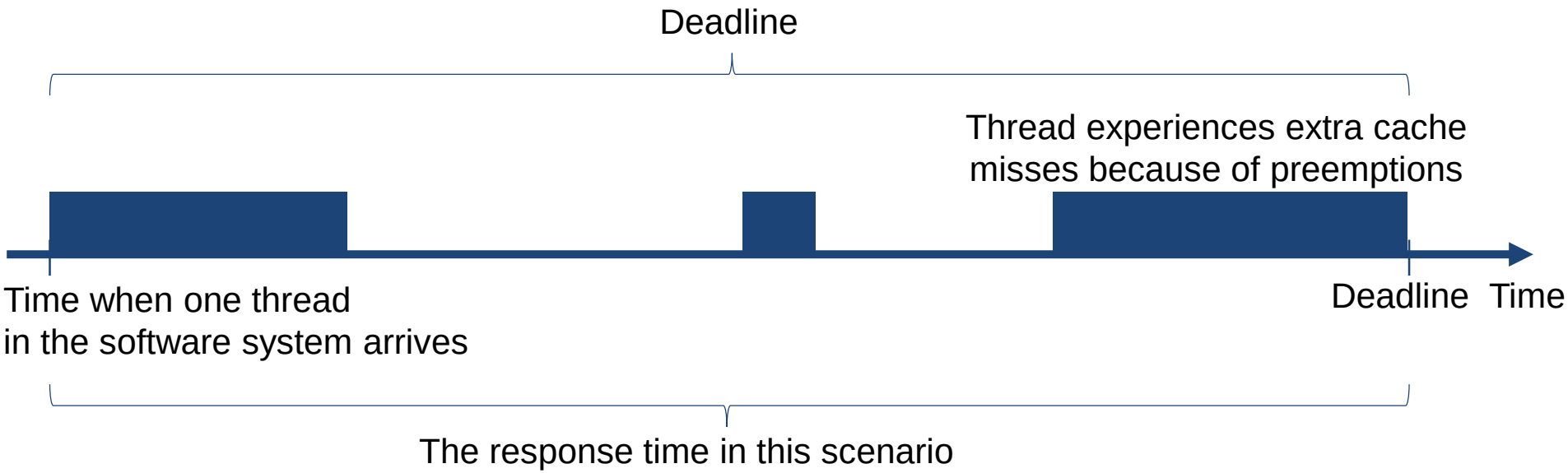
What Causes Delay of Software?

Does it hold for all scenarios that the response time is at most the deadline?



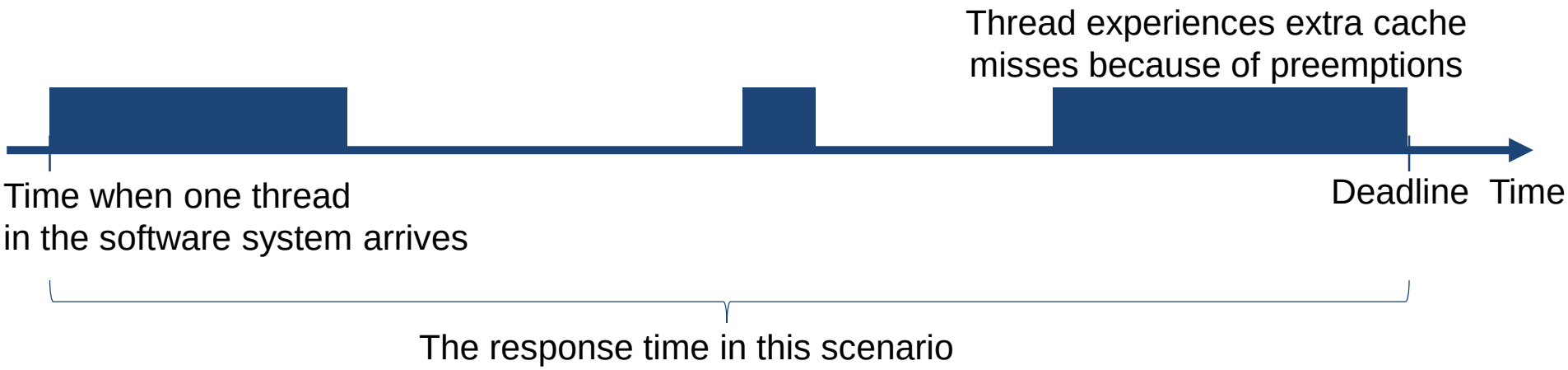
What Causes Delay of Software?

Does it hold for all scenarios that the response time is at most the deadline?



What Causes Delay of Software?

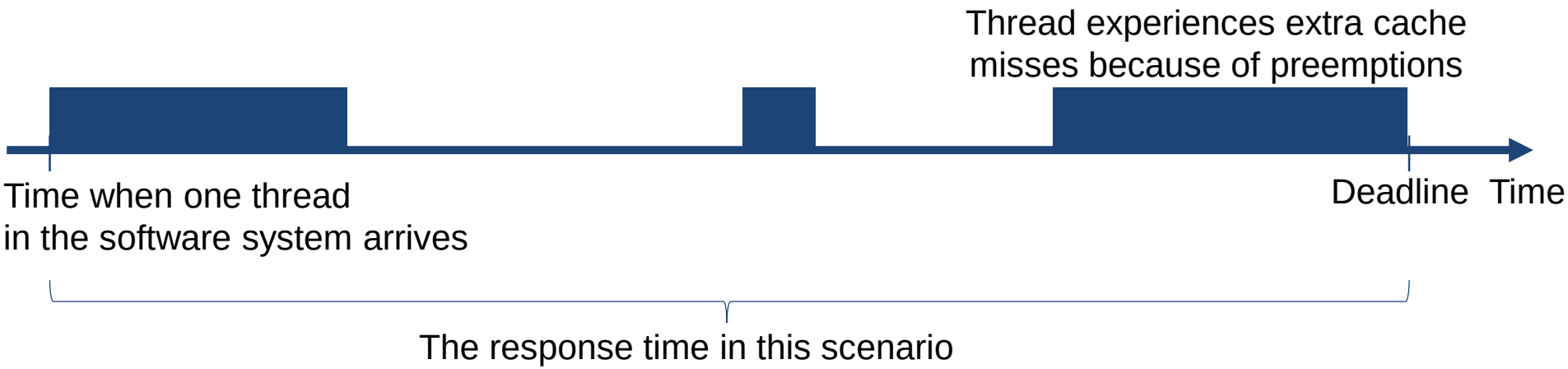
Does it hold for all threads,
that for all arrivals of the thread,
that for all scenarios,
that the finishing time is at most the deadline?



What Causes Delay of Software?

Does it hold for all threads,
that for all arrivals of the thread,
that for all scenarios,
that the finishing time is at most the deadline?

If “yes,” we say the set of threads is *schedulable*.
Otherwise, the set of threads is *unschedulable*.

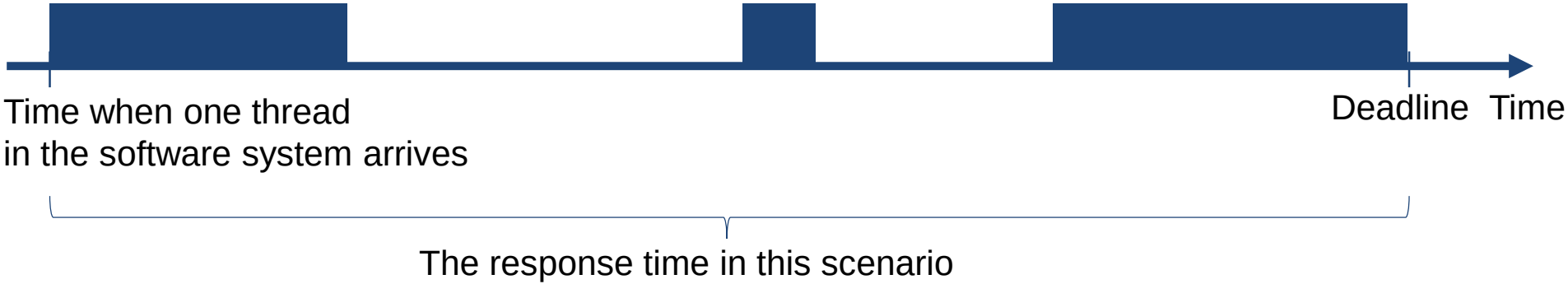


What Causes Delay of Software?

Does it hold for all threads,
that for all arrivals of the thread,
that for all scenarios,
that the finishing time is at most the deadline?

If “yes,” we say the set of threads is *schedulable*.
Otherwise, the set of threads is *unschedulable*.

The process of determining whether a set of threads is schedulable is called *schedulability analysis*.



Conclusions so far

Many systems interact with the physical world

This interaction requires correct timing

Correct timing depends on the whether the delay of the software is at most a certain bound

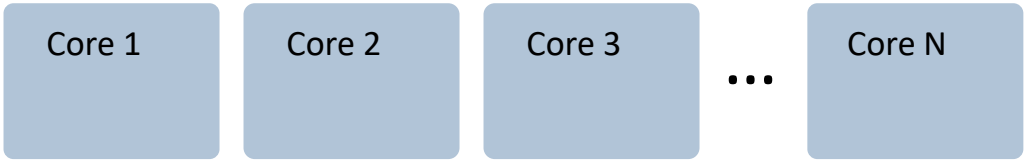
There are many causes of the delay of software (even on a computer with a single core)

Many systems today disable all processor cores except one in order to be confident about timing

Real-Time Requirements of Software Executing on a Multicore Processor

Hardware Trends

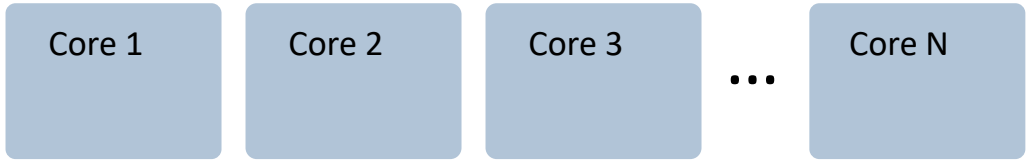
- *All computers are multicores.*



Real-Time Requirements of Software Executing on a Multicore Processor

Hardware Trends

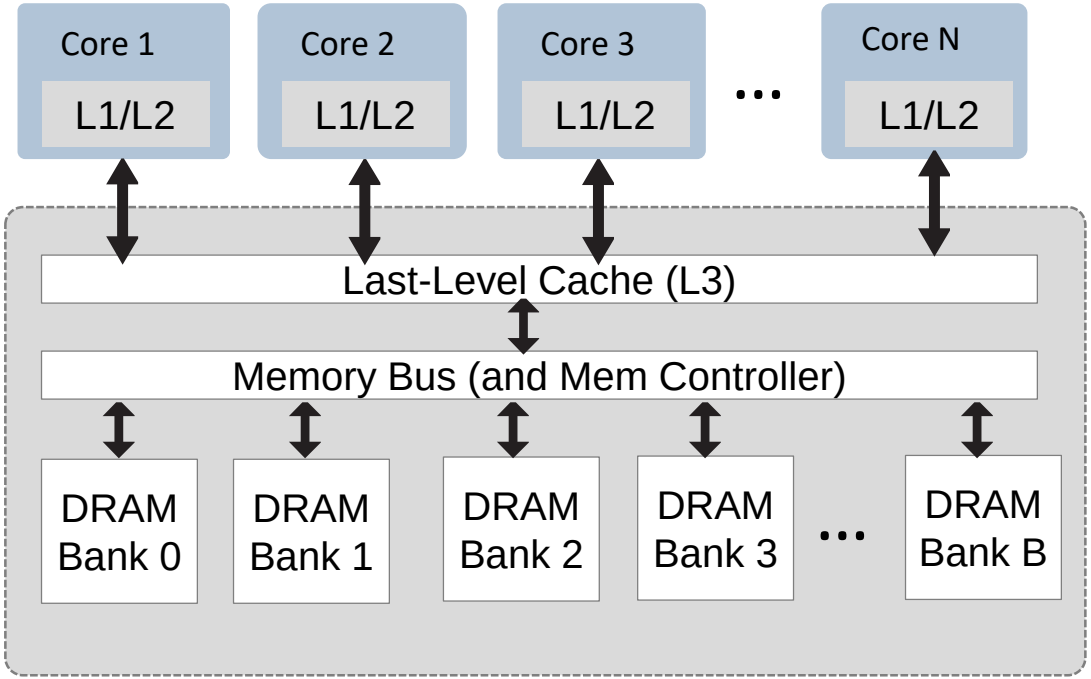
- *All computers are multicores.*
- *Most chip makers do not offer single core.*



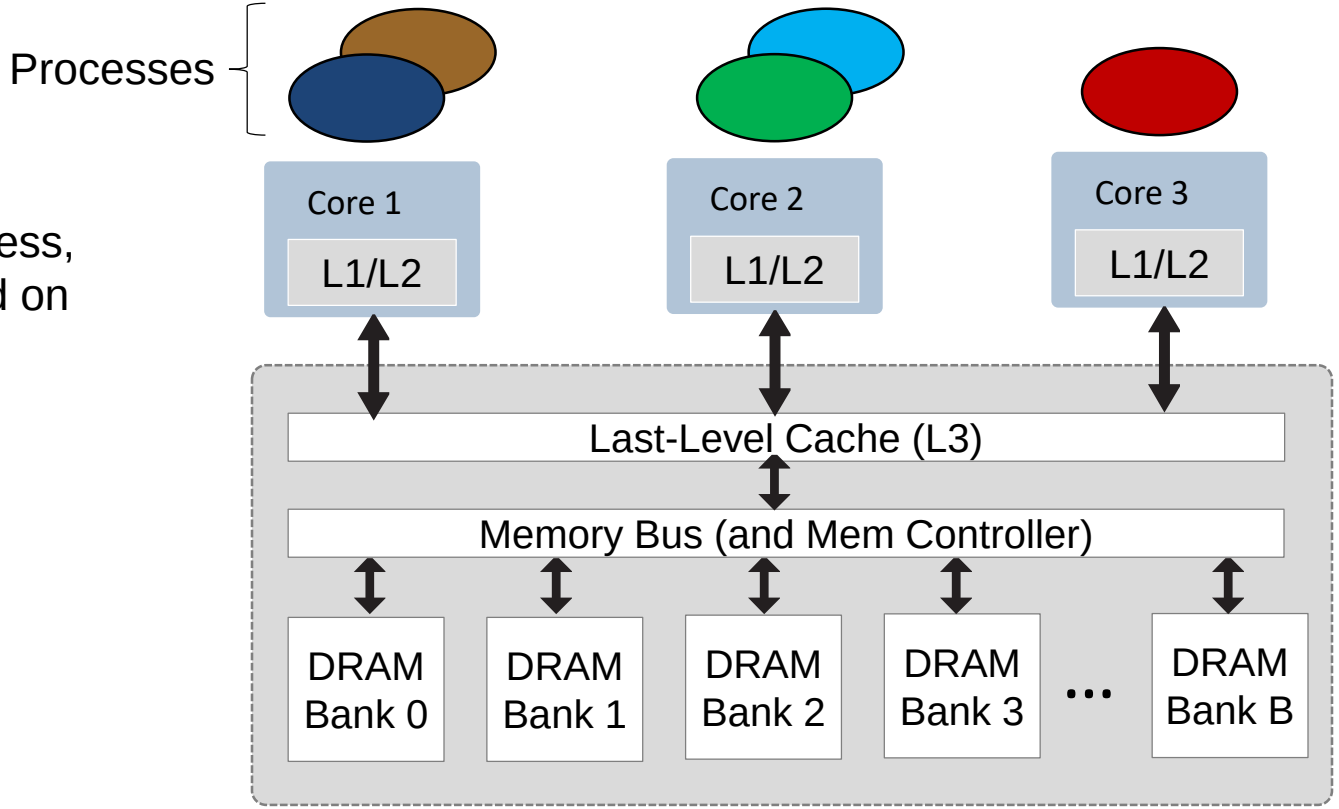
Real-Time Requirements of Software Executing on a Multicore Processor

Hardware Trends

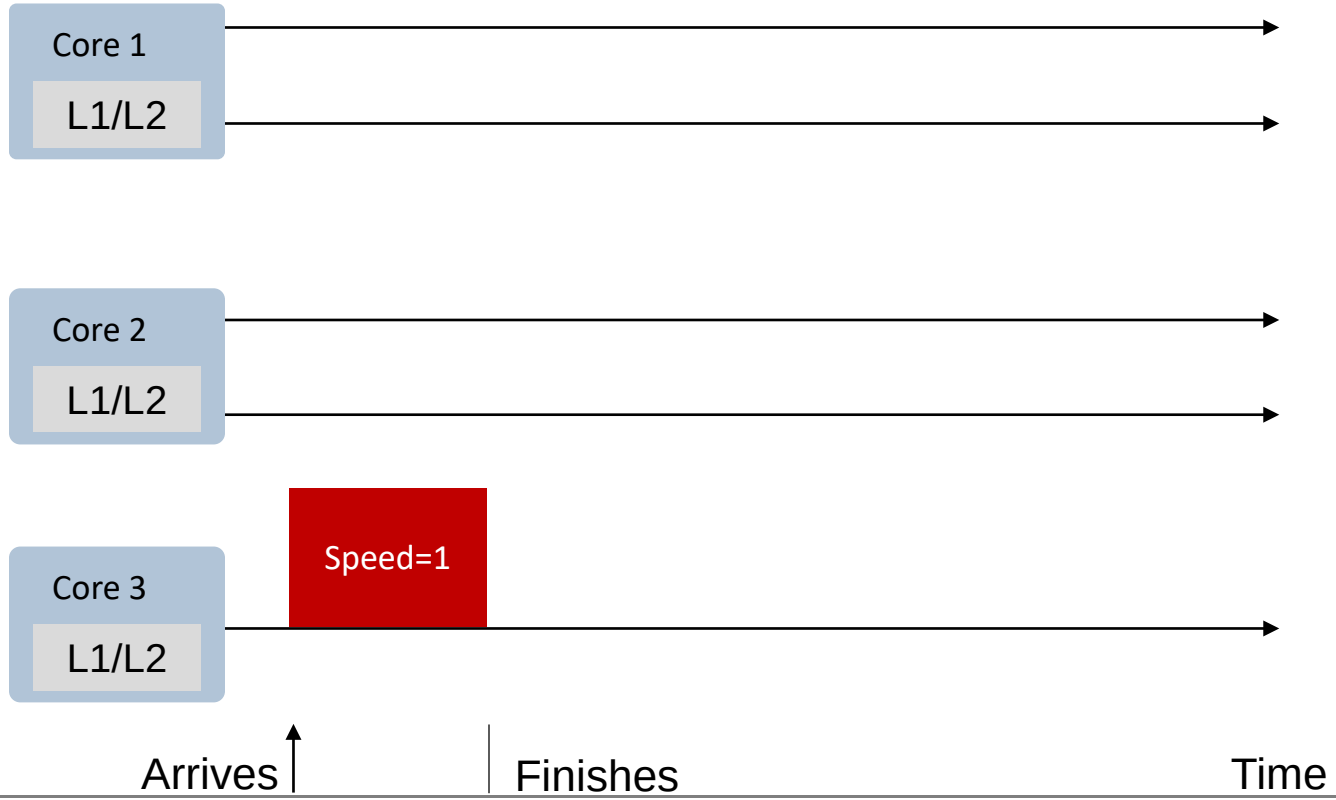
- *All computers are multicores.*
- *Most chip makers do not offer single core.*
- *Most multicores have shared memory.*



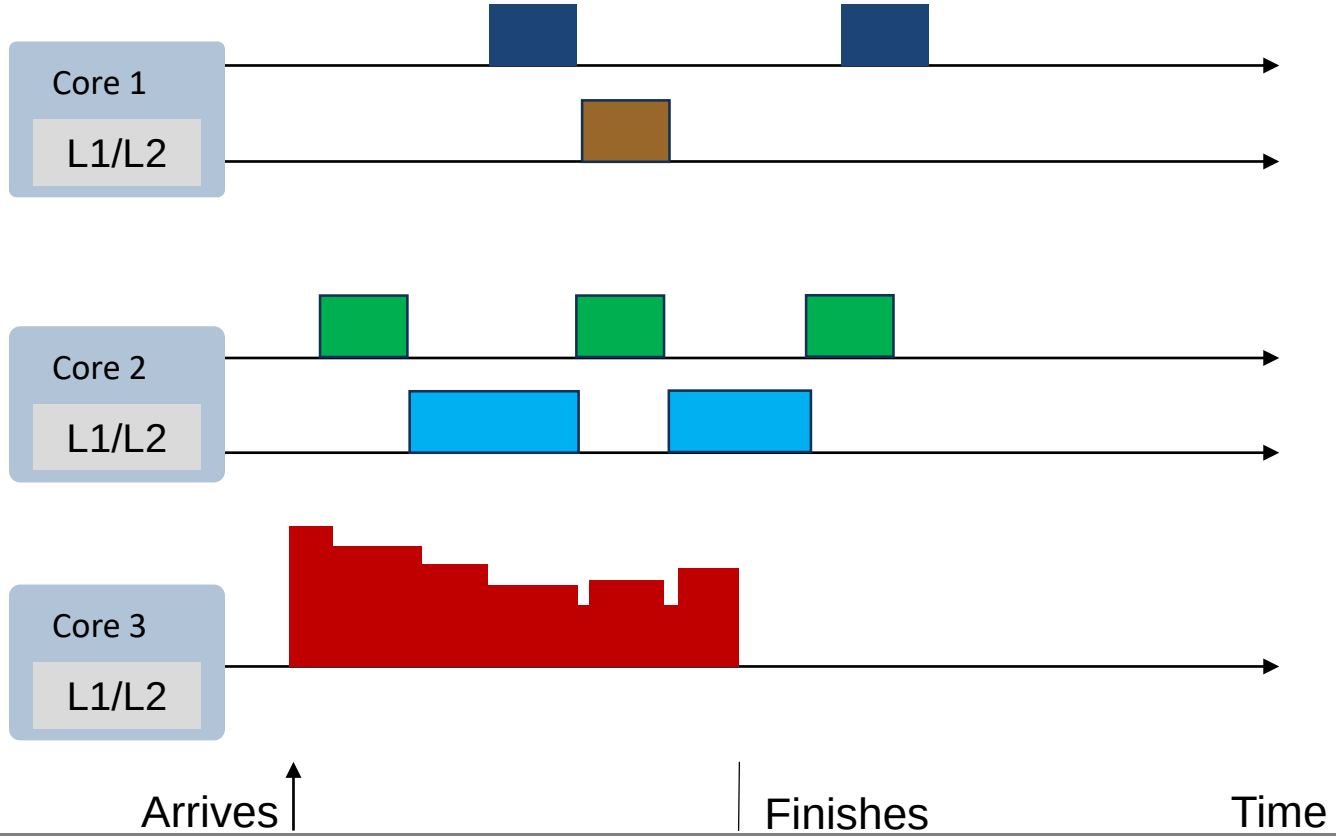
Problem: For each process, compute an upper bound on its response time.



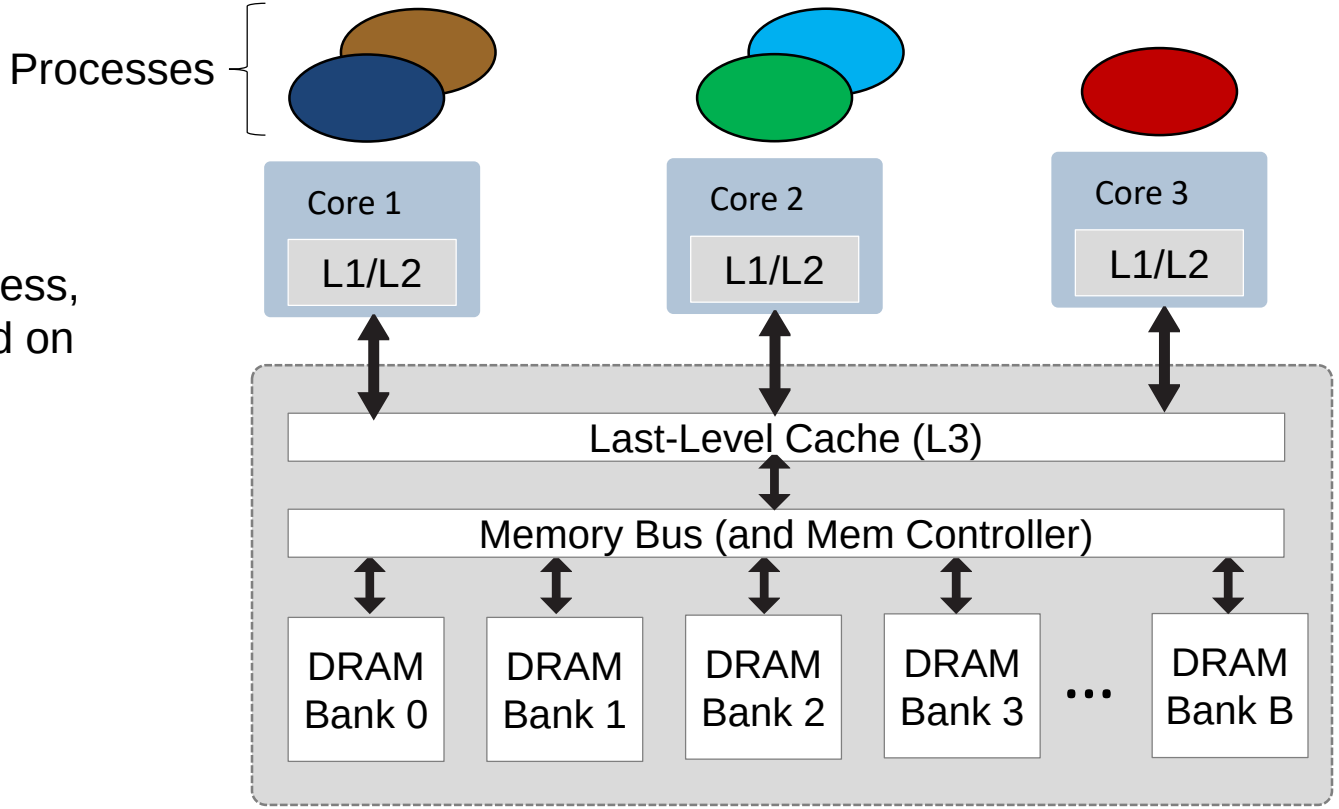
How Co-Runners Impact Speed of Execution



How Co-Runners Impact Speed of Execution

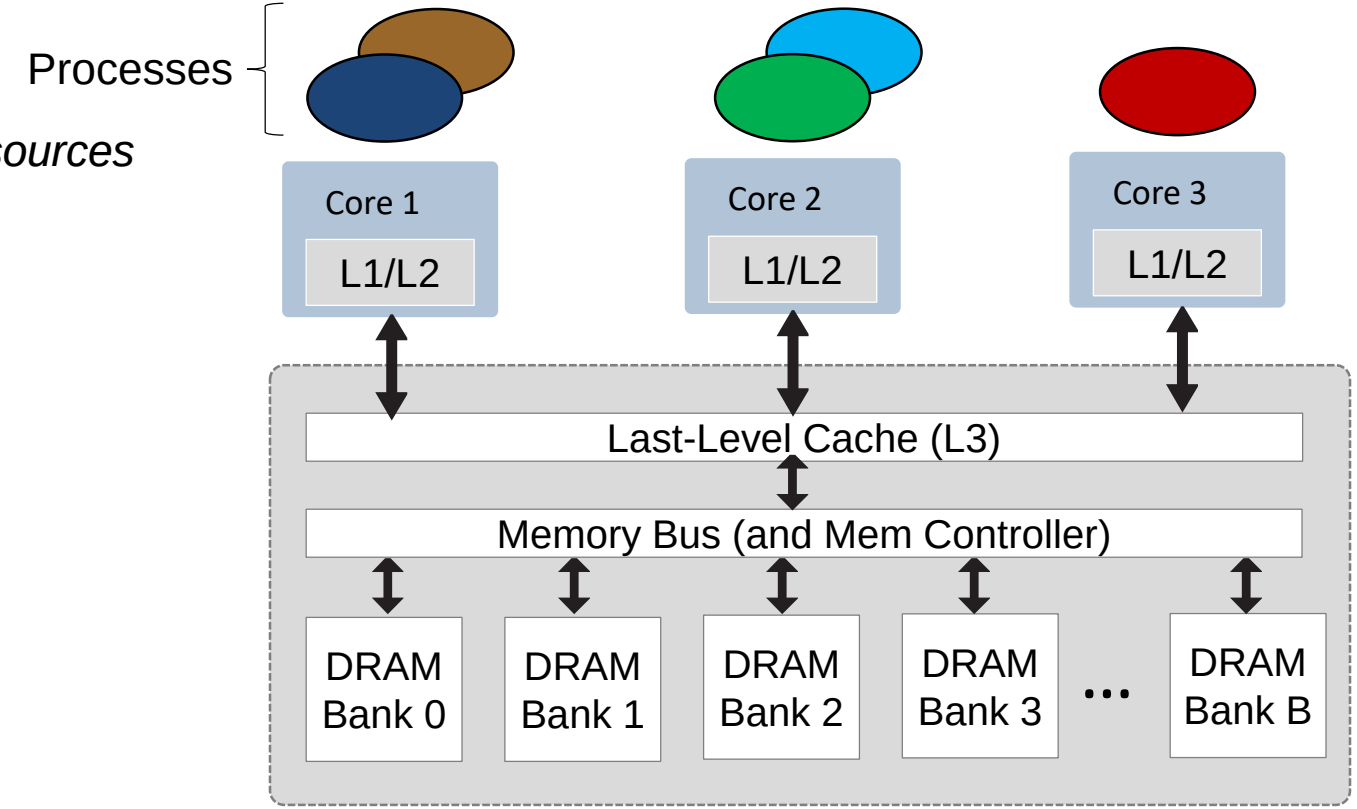


Problem: For each process, compute an upper bound on its response time.

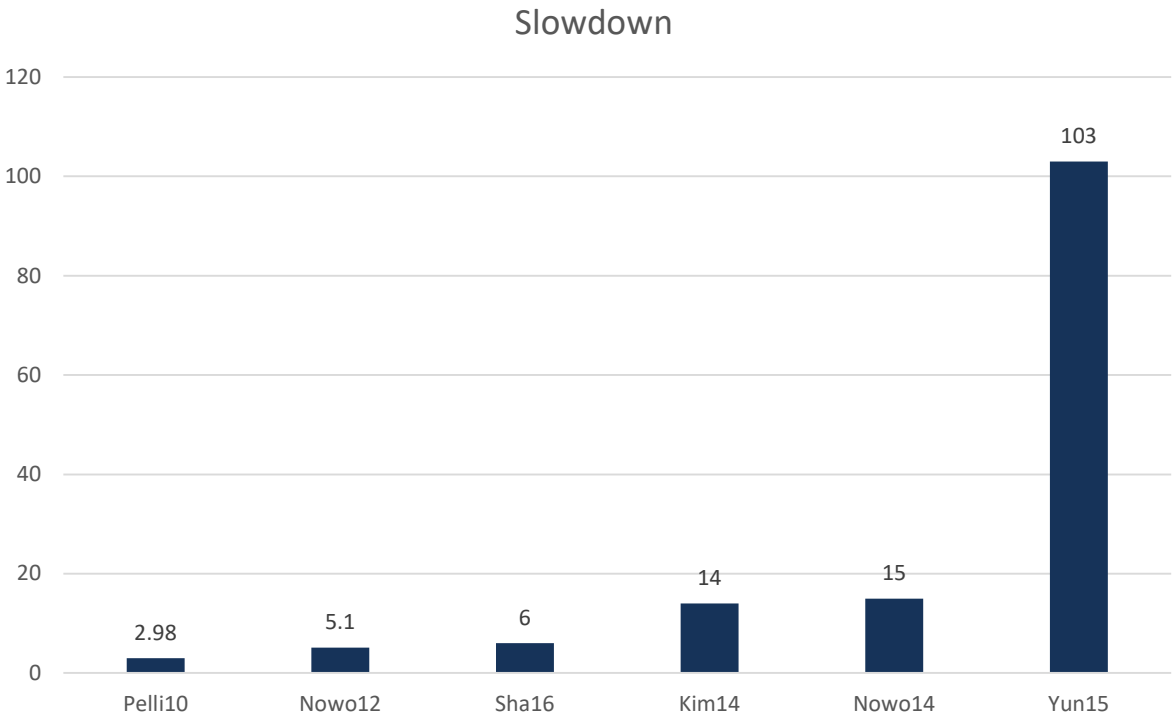


Issues

- *Shared hardware resources impact timing.*

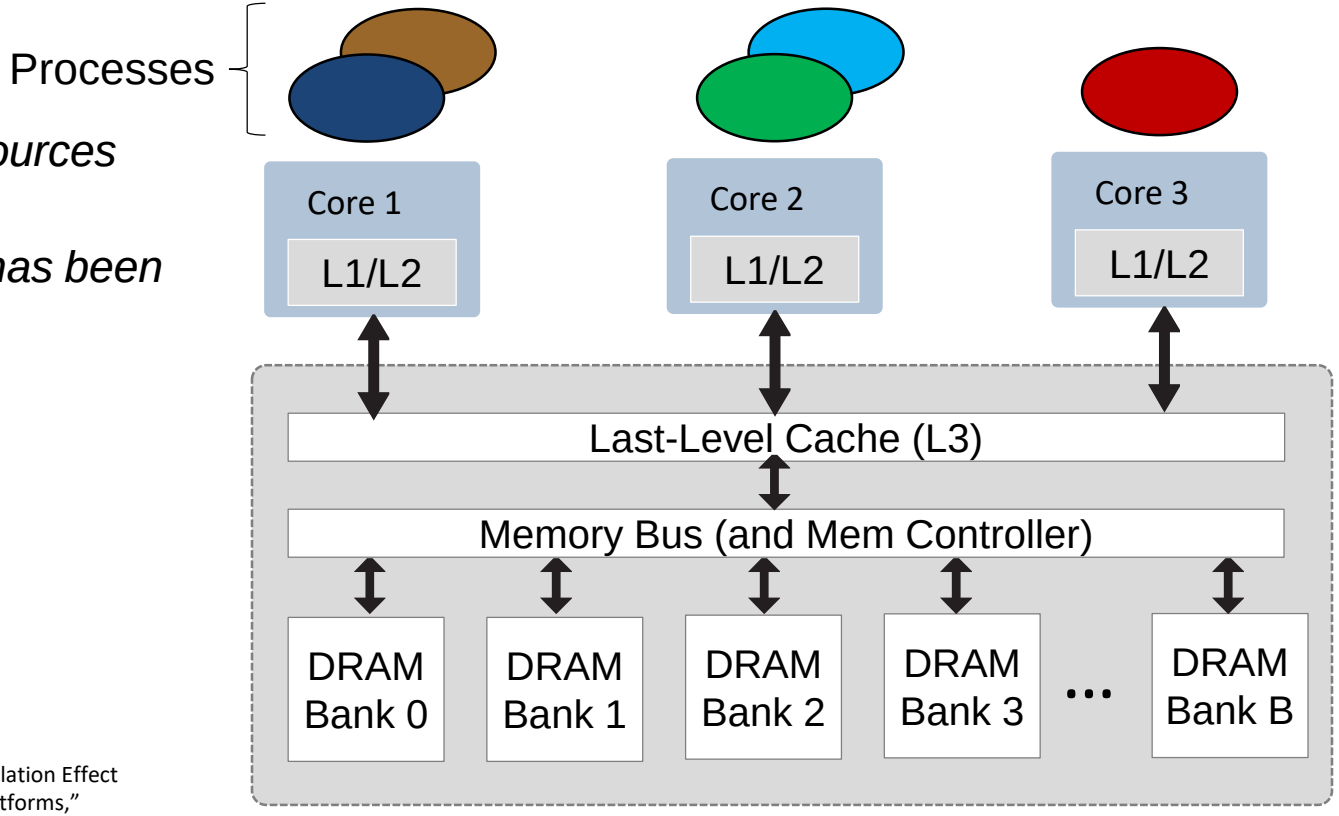


How Bad?



Issues

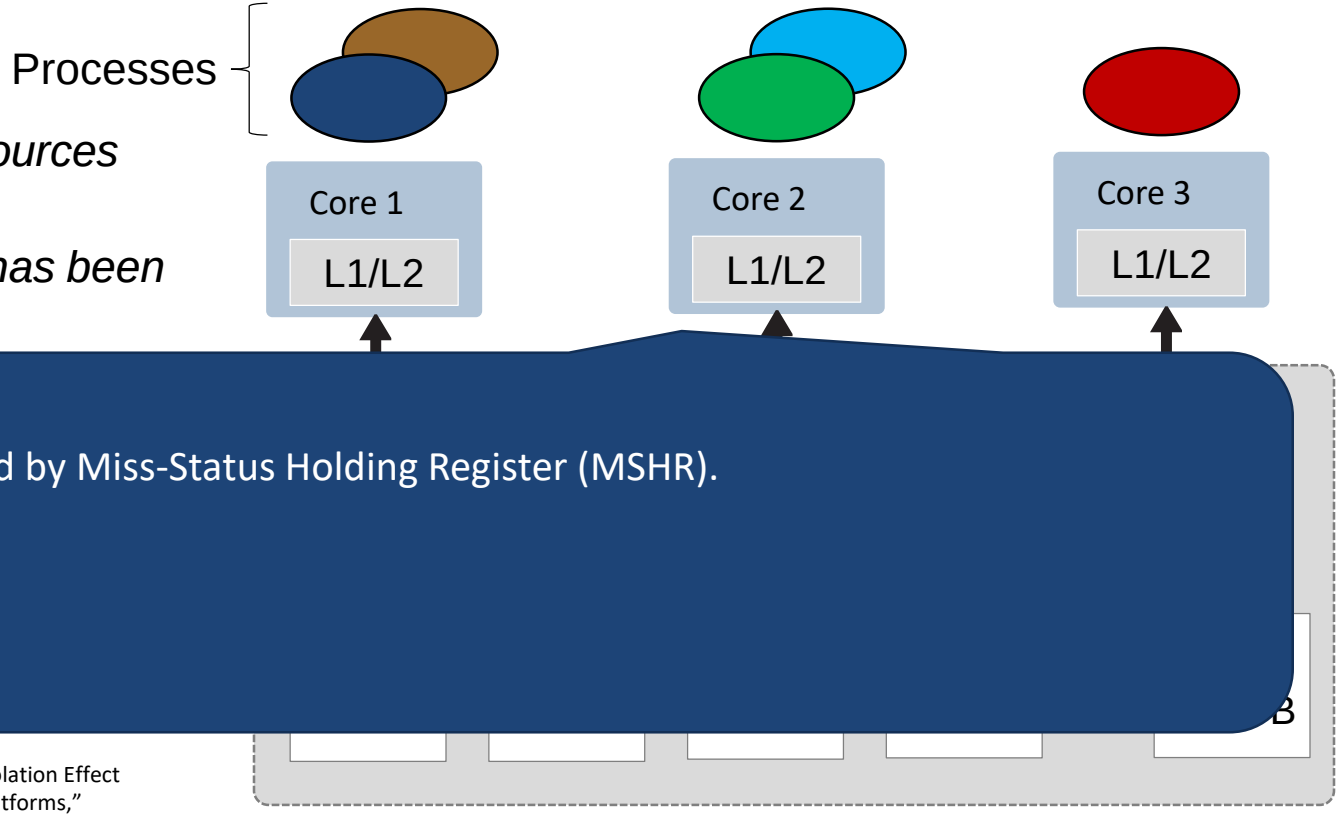
- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**



*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

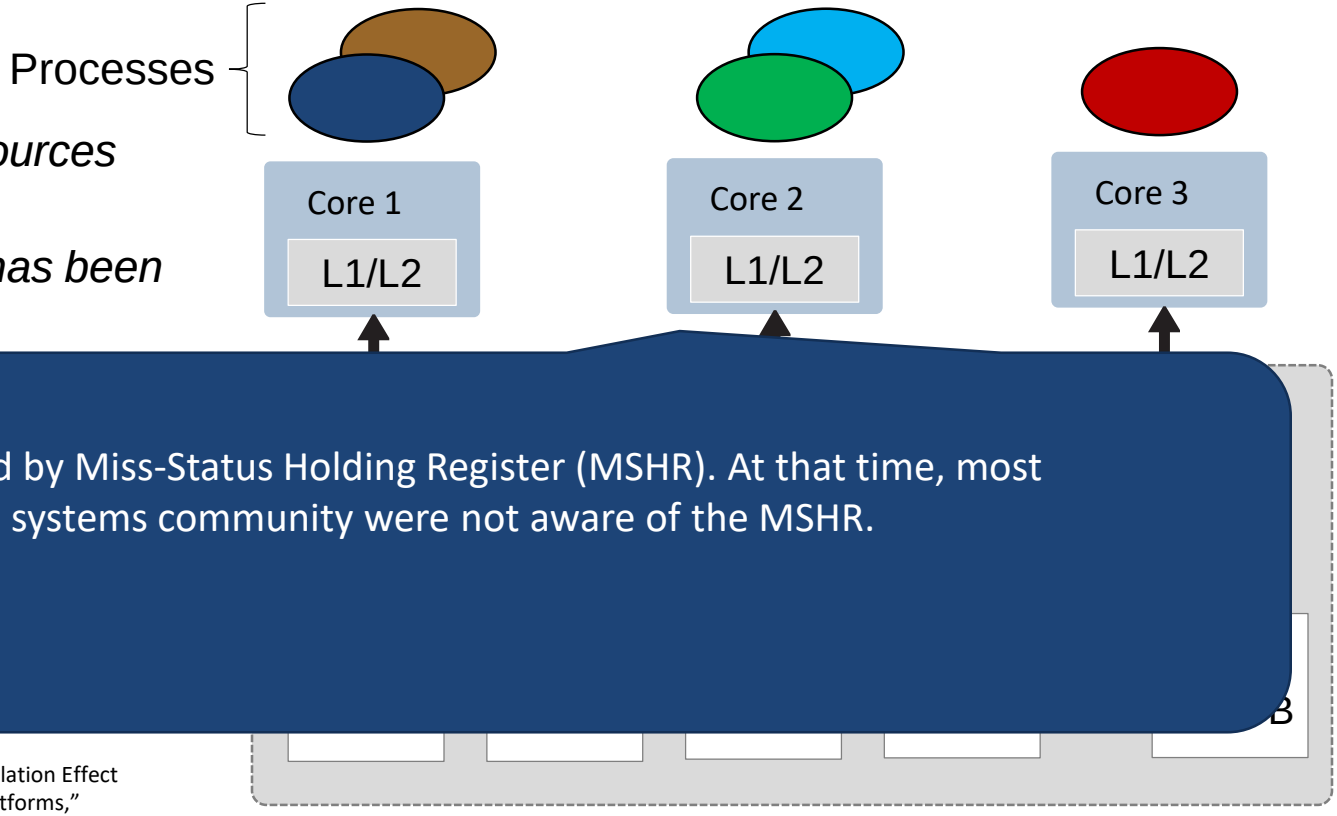
- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**



*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

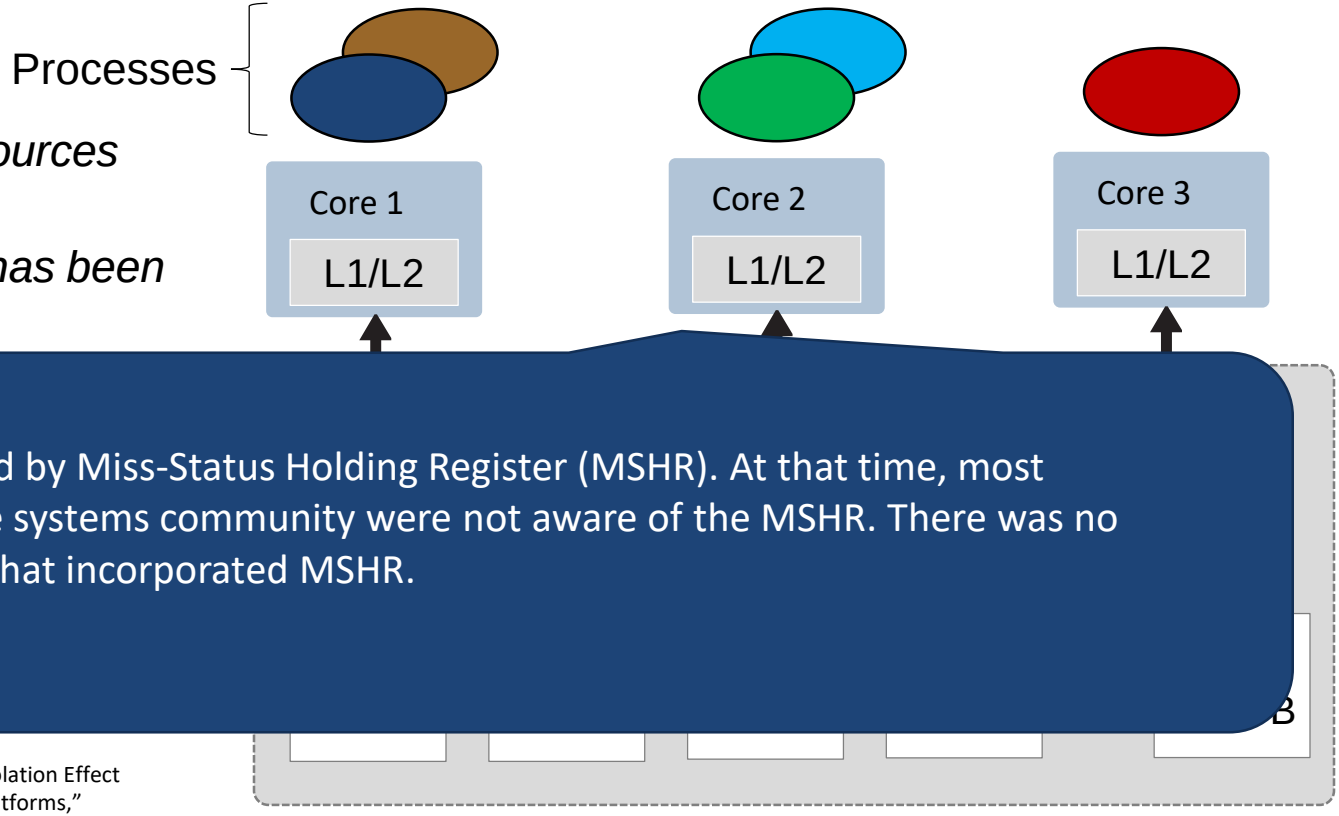


This slowdown is caused by Miss-Status Holding Register (MSHR). At that time, most researchers in real-time systems community were not aware of the MSHR.

*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

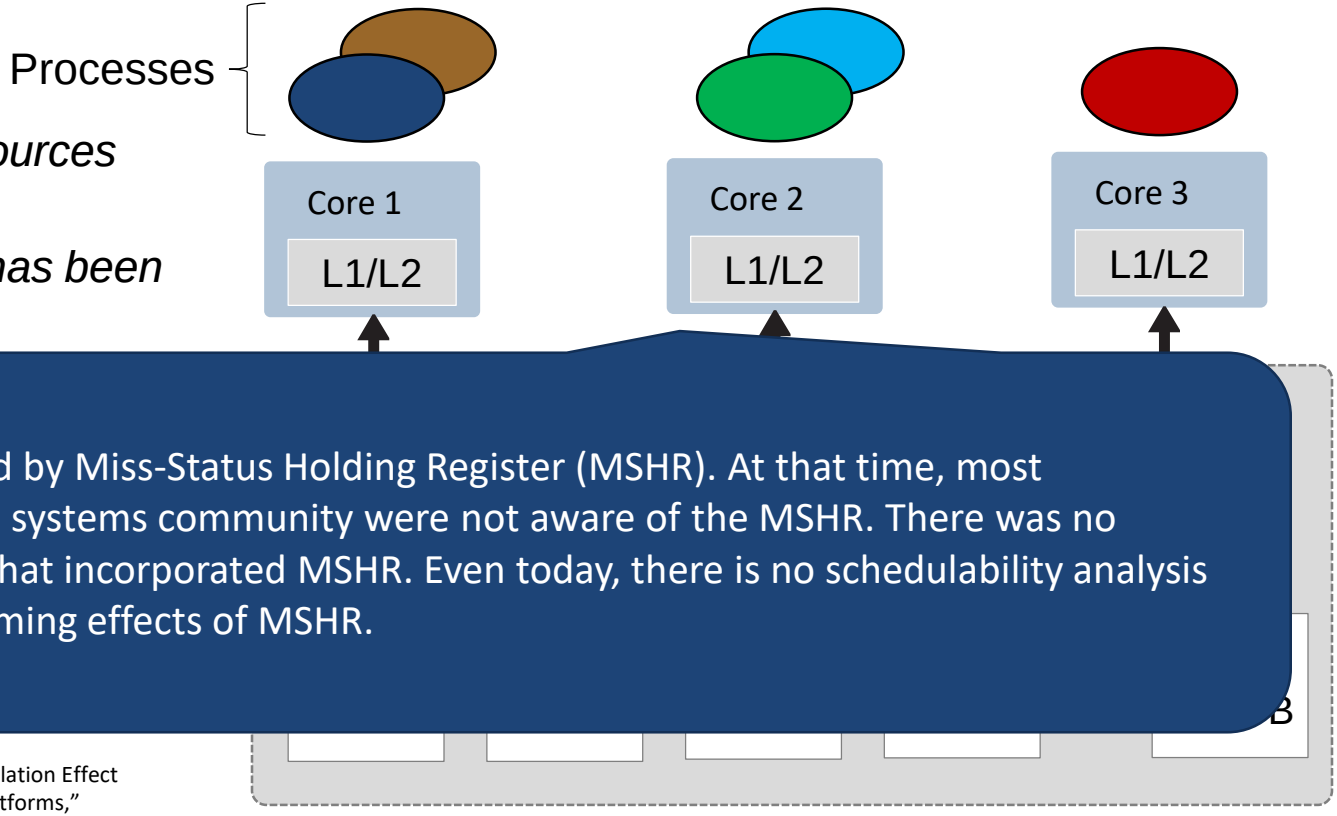


This slowdown is caused by Miss-Status Holding Register (MSHR). At that time, most researchers in real-time systems community were not aware of the MSHR. There was no schedulability analysis that incorporated MSHR.

*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

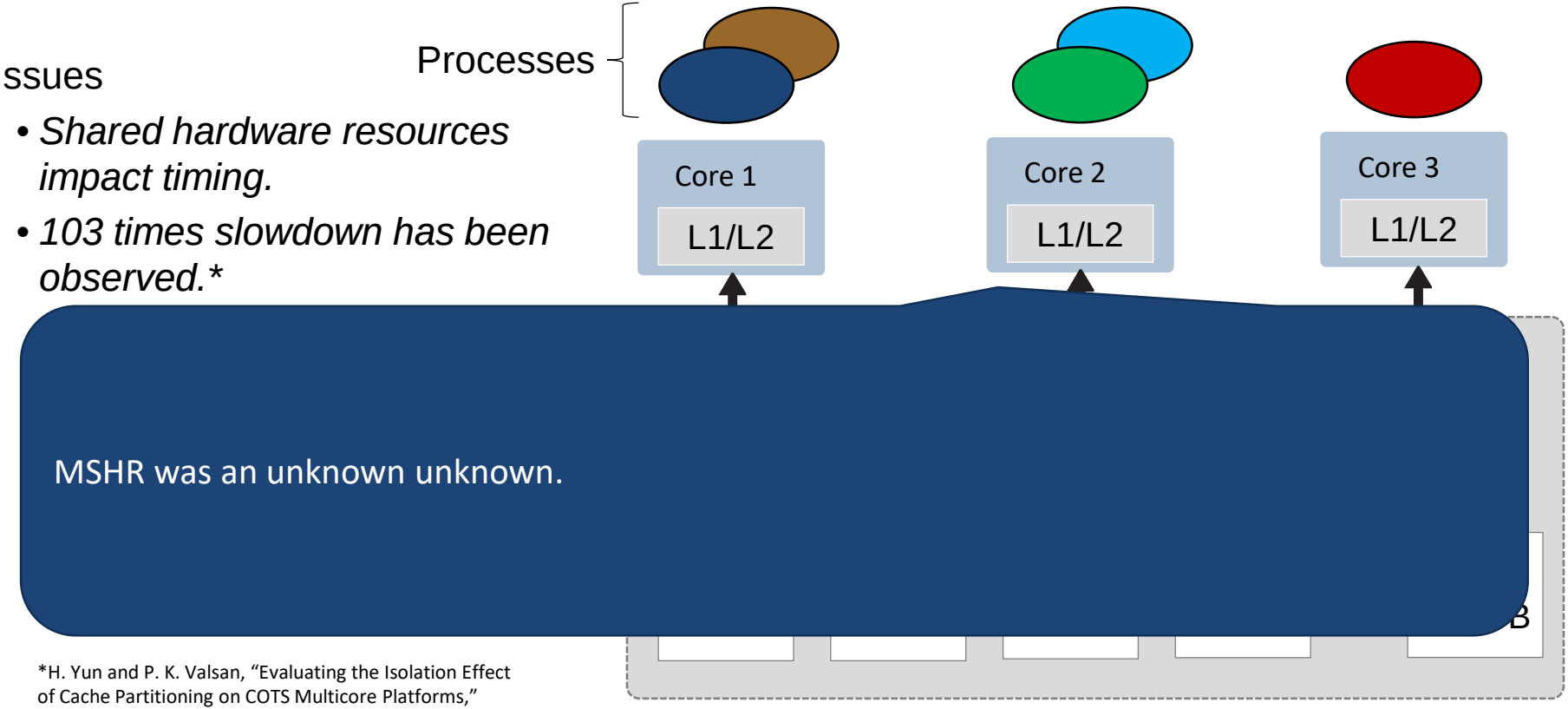


This slowdown is caused by Miss-Status Holding Register (MSHR). At that time, most researchers in real-time systems community were not aware of the MSHR. There was no schedulability analysis that incorporated MSHR. Even today, there is no schedulability analysis that incorporates the timing effects of MSHR.

*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

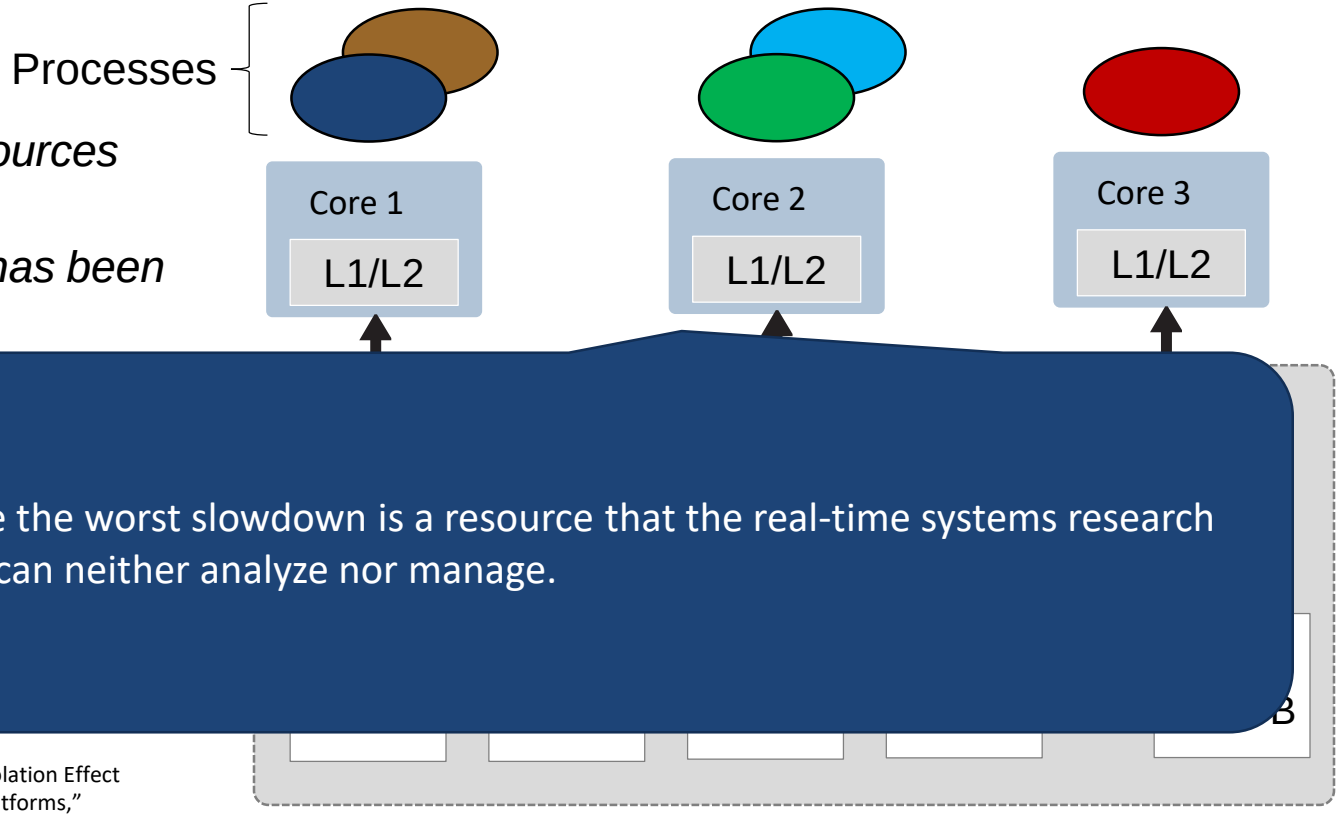
- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**



*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

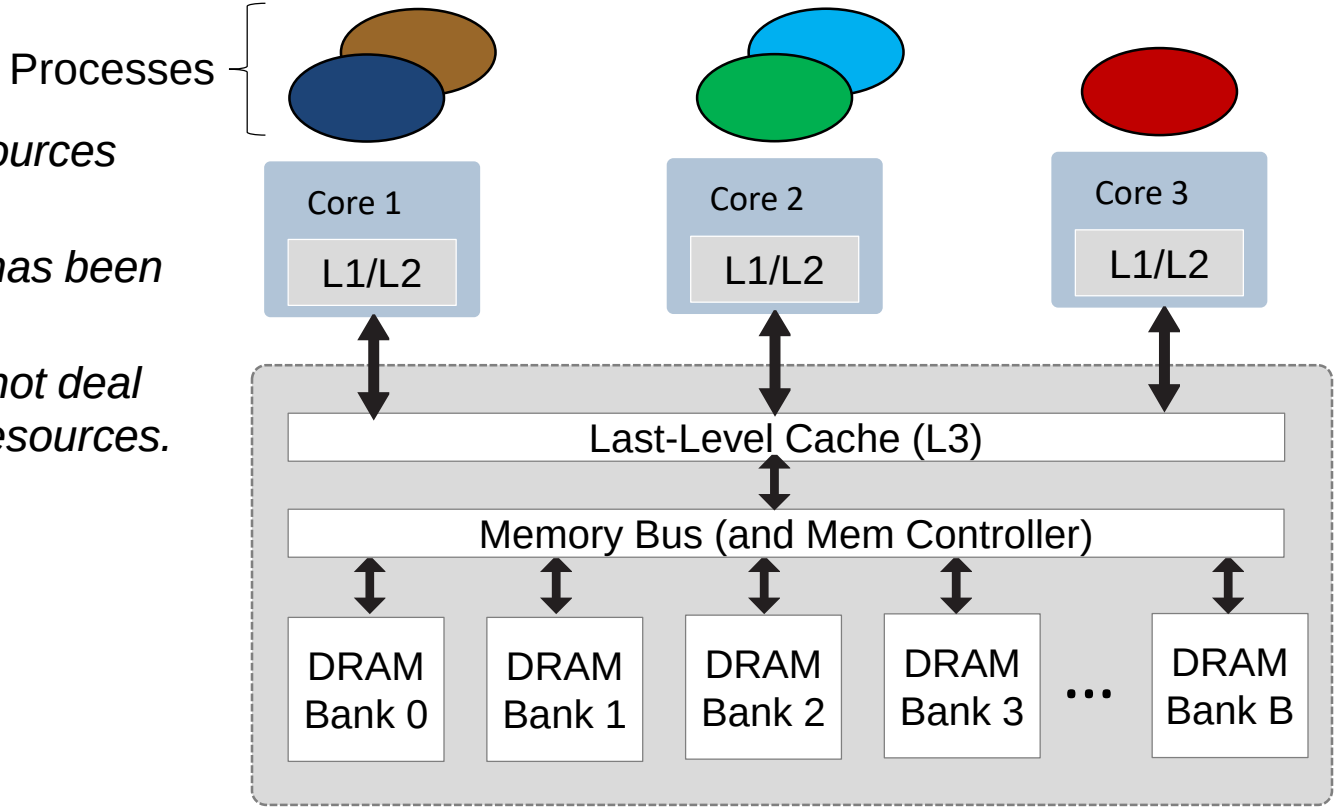


The resource that cause the worst slowdown is a resource that the real-time systems research computing community can neither analyze nor manage.

*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

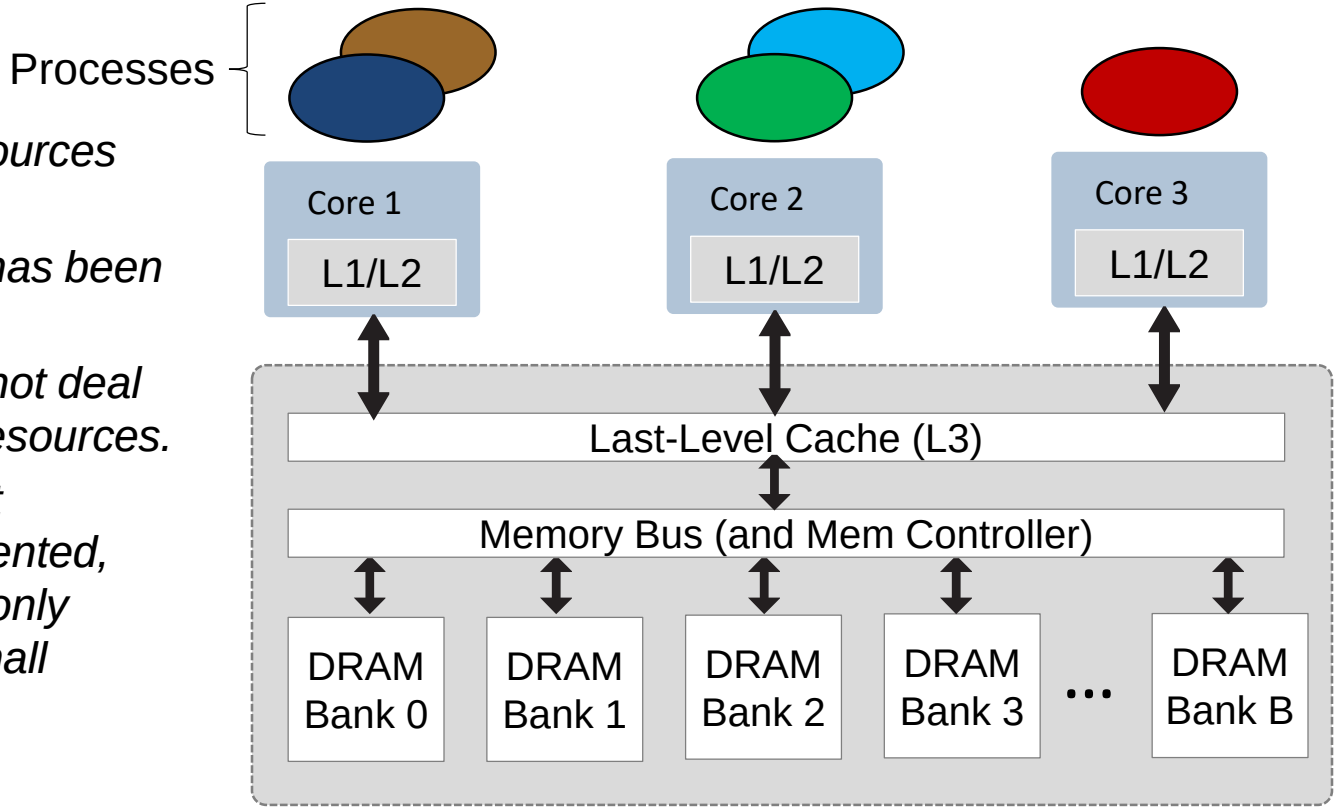
Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed [Yun15].*
- *Current methods cannot deal with undocumented resources.*



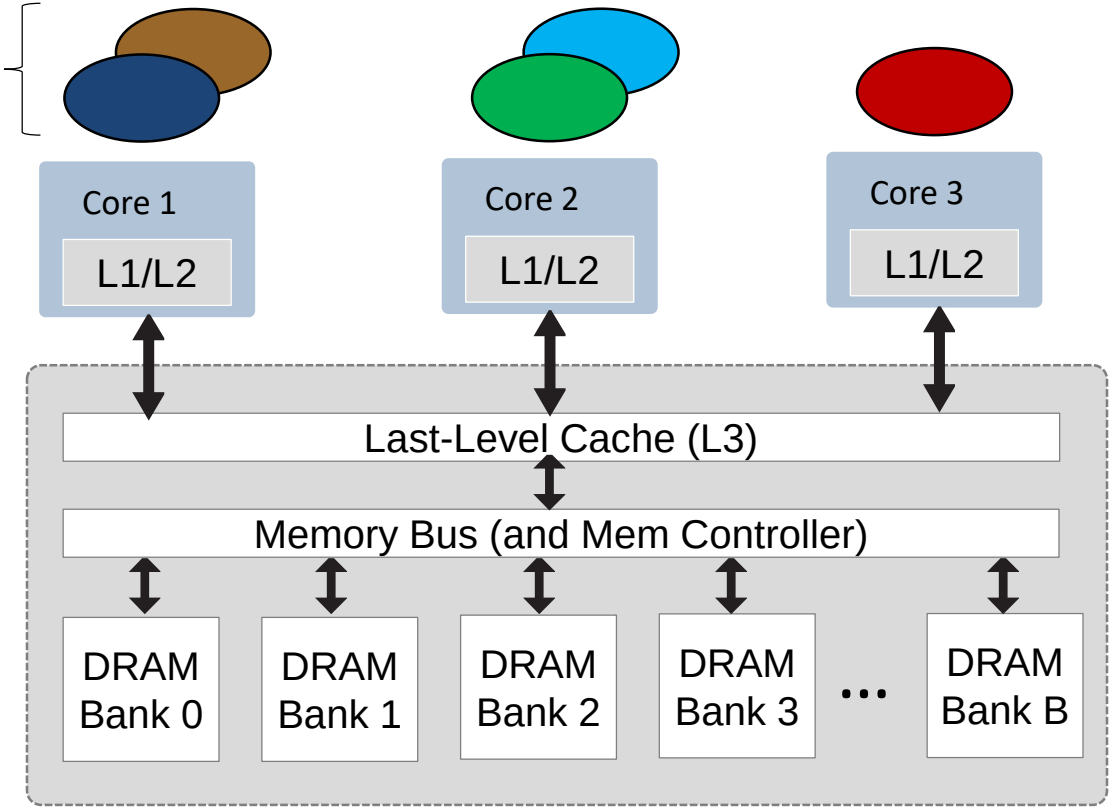
Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed [Yun15].*
- *Current methods cannot deal with undocumented resources.*
- *Even for the case that resources are documented, current methods can only analyze/manage a small set of them.*

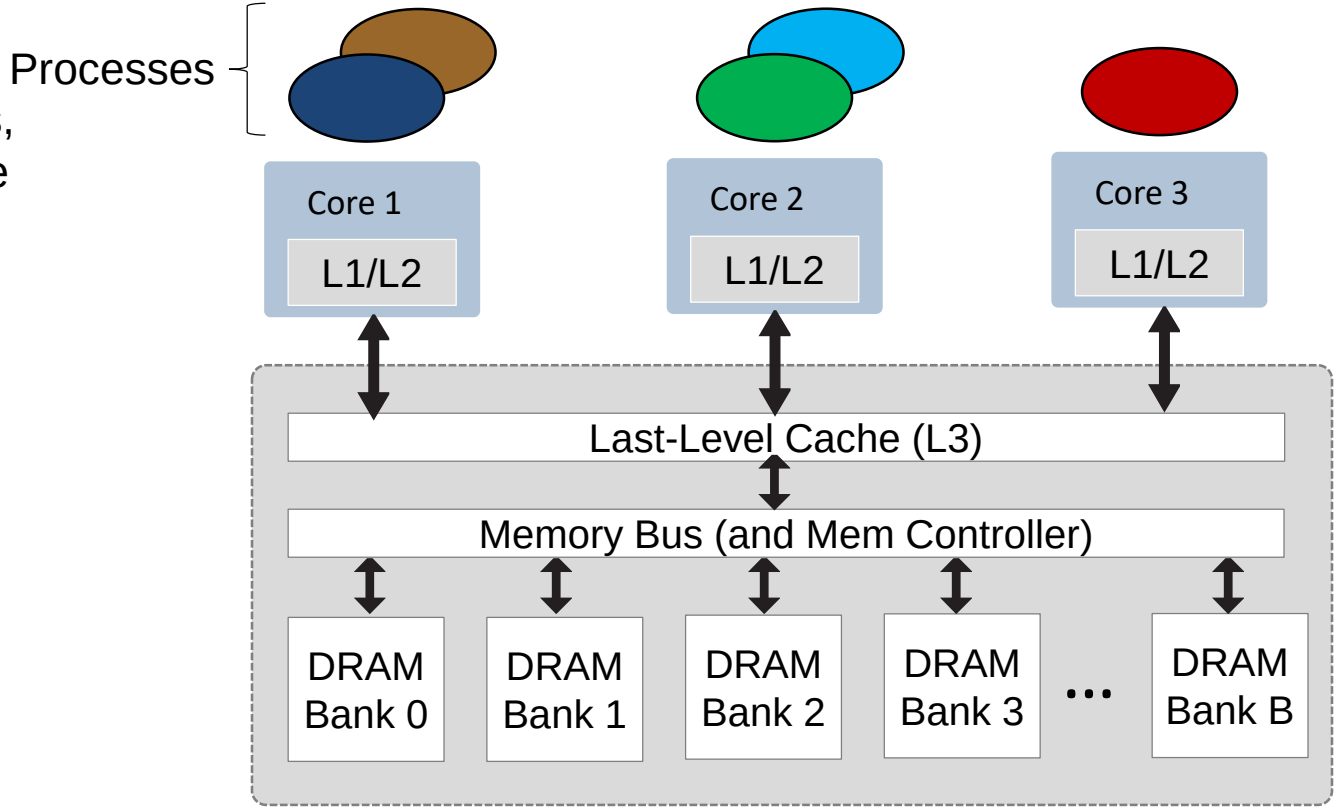


Issues

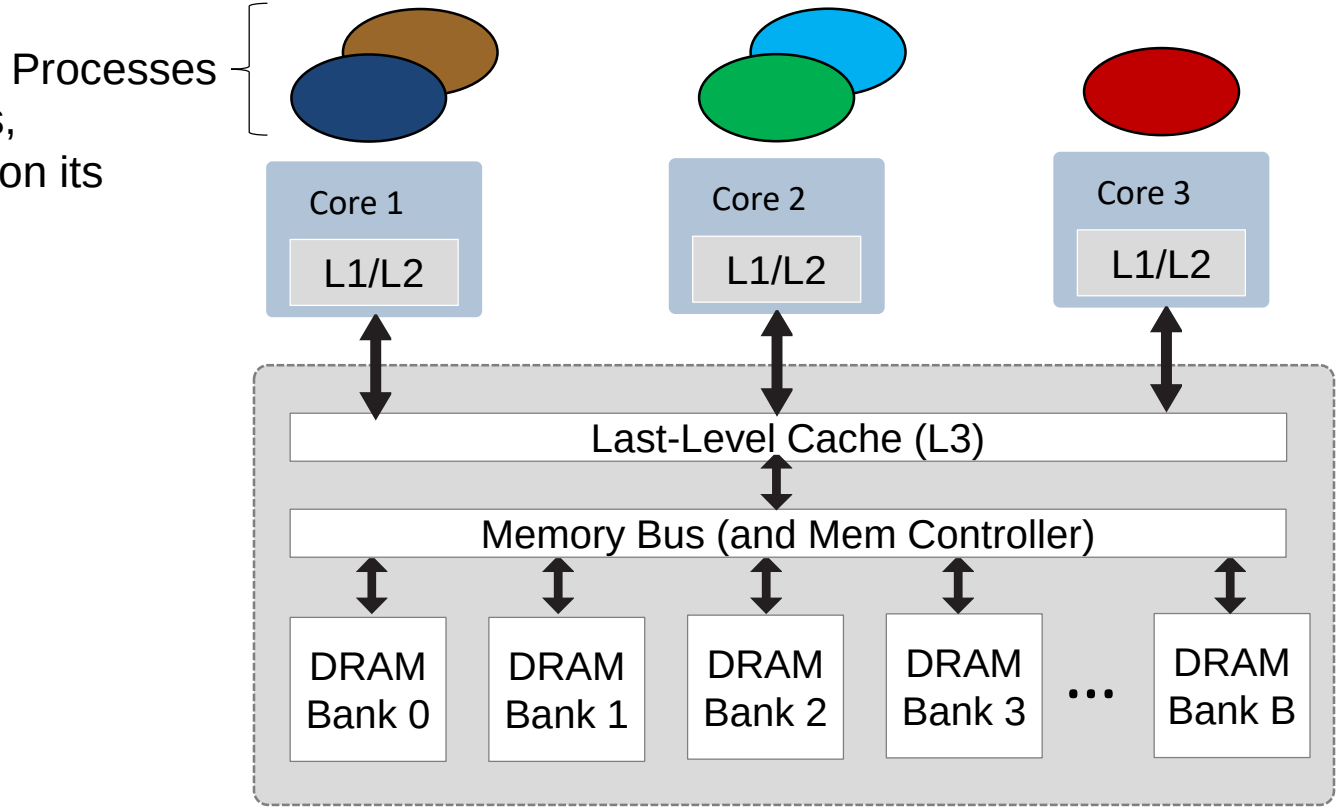
- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed [Yun15].*
- *Current methods cannot deal with undocumented resources.*
- *Even when resources are documented, current methods can only analyze/manage a small set of them.*
- *The problem is getting worse:*
  ** Slowdown increasing*
  ** More undocumented h/w*



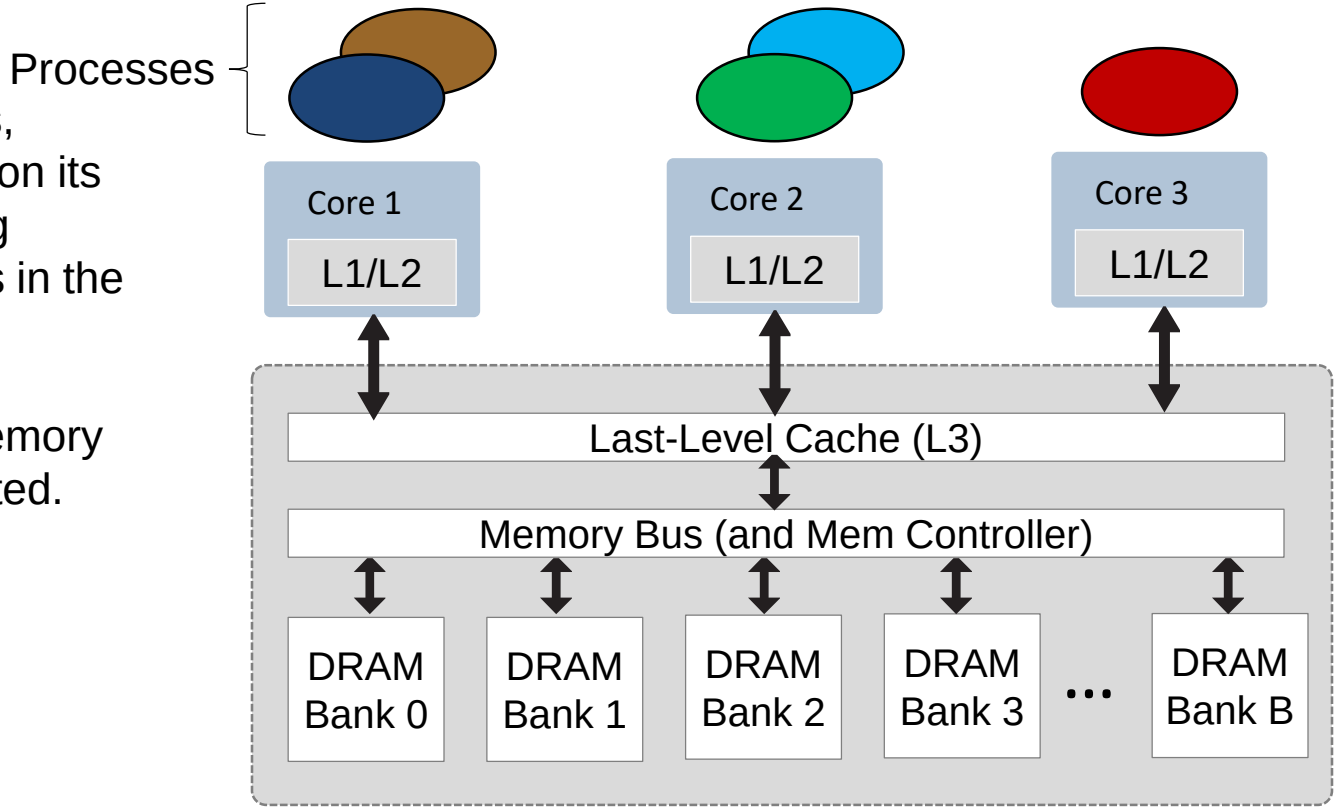
Problem: For each process, compute its response time



Problem: For each process, compute an upper bound on its response time

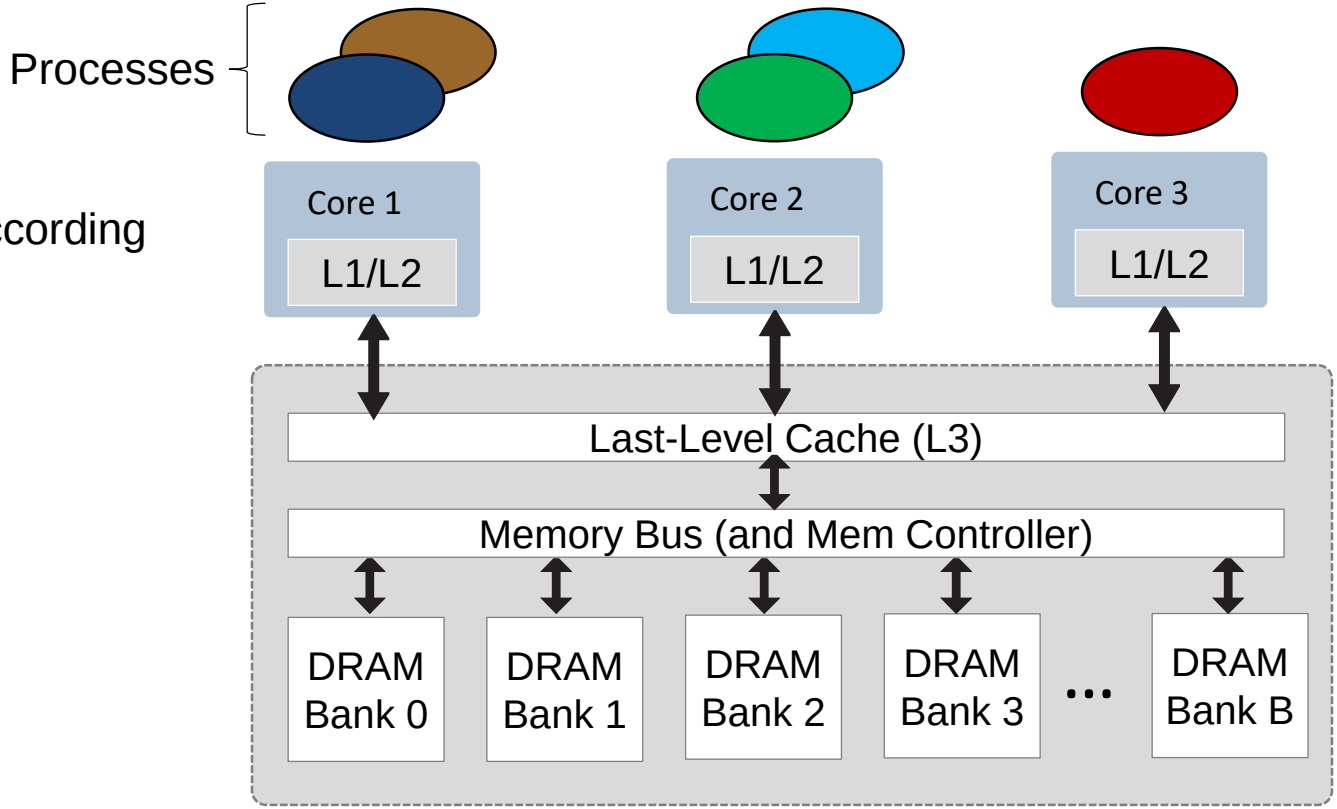


Problem: For each process, compute an upper bound on its response time considering contention for resources in the memory system and that resources in the memory system are undocumented.



Documented resources

- DRAM memory timing tends to be specified according to JDEC standard.

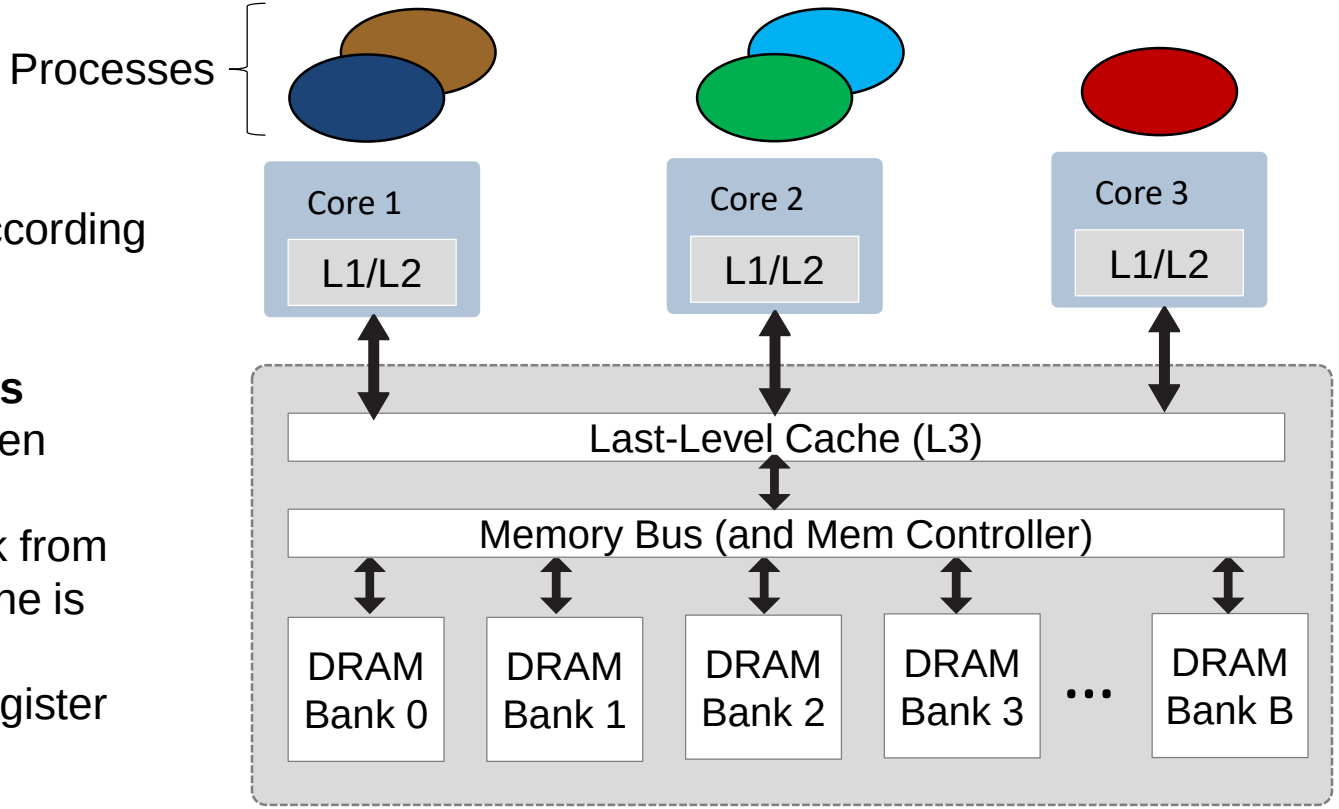


Documented resources

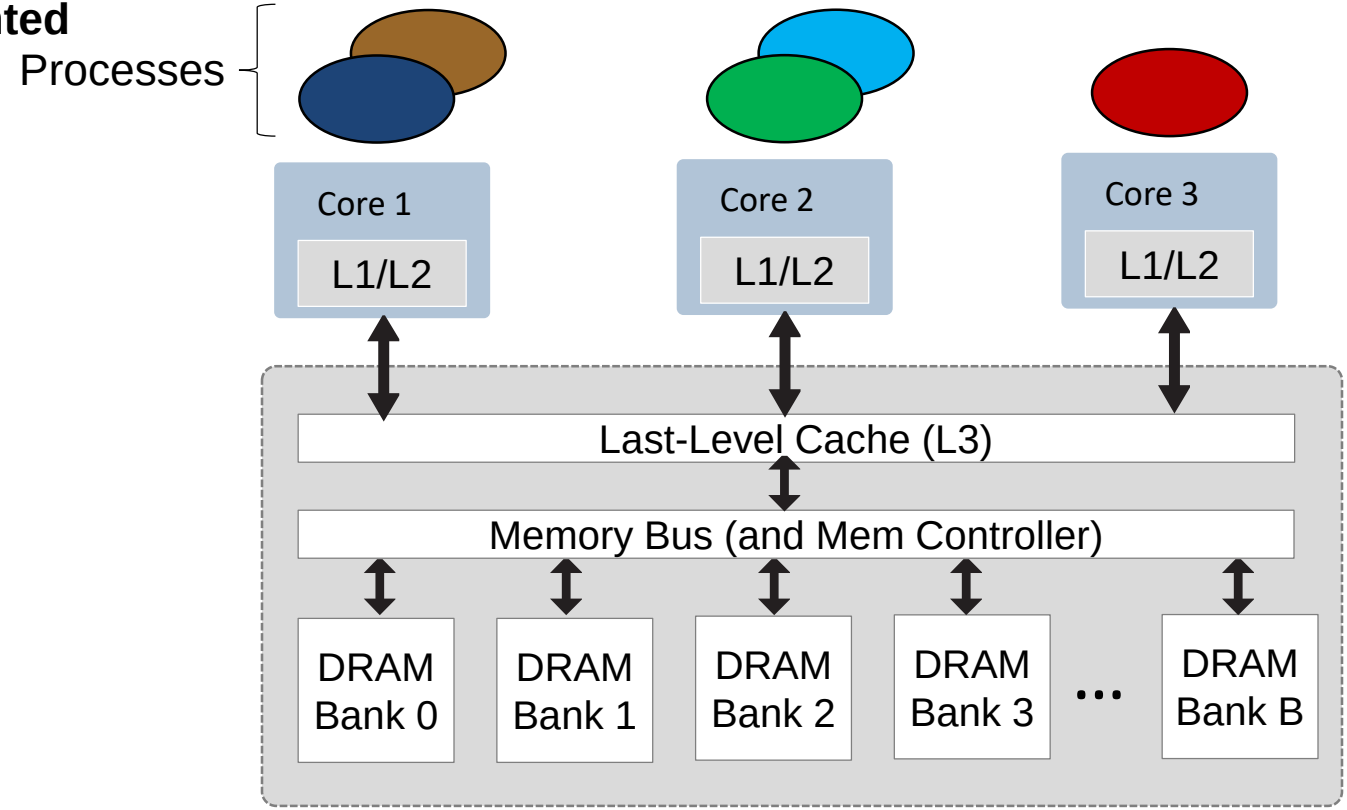
- DRAM memory timing tends to be specified according to JDEC standard.

Undocumented resources

- Memory controller is often undocumented.
- Interconnection network from cores to Last-Level cache is often undocumented.
- Miss-Status Holding Register is often undocumented.

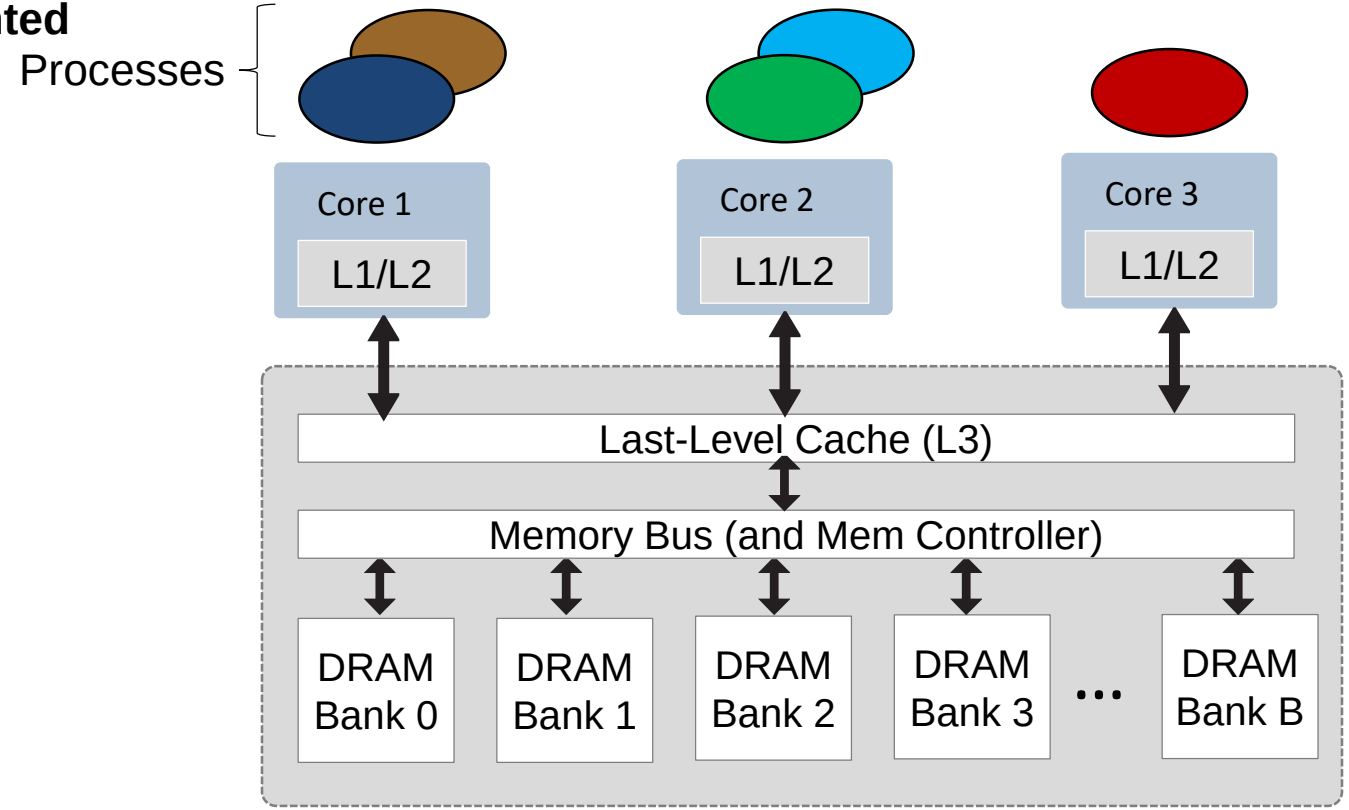


Different reasons for treating resources as undocumented



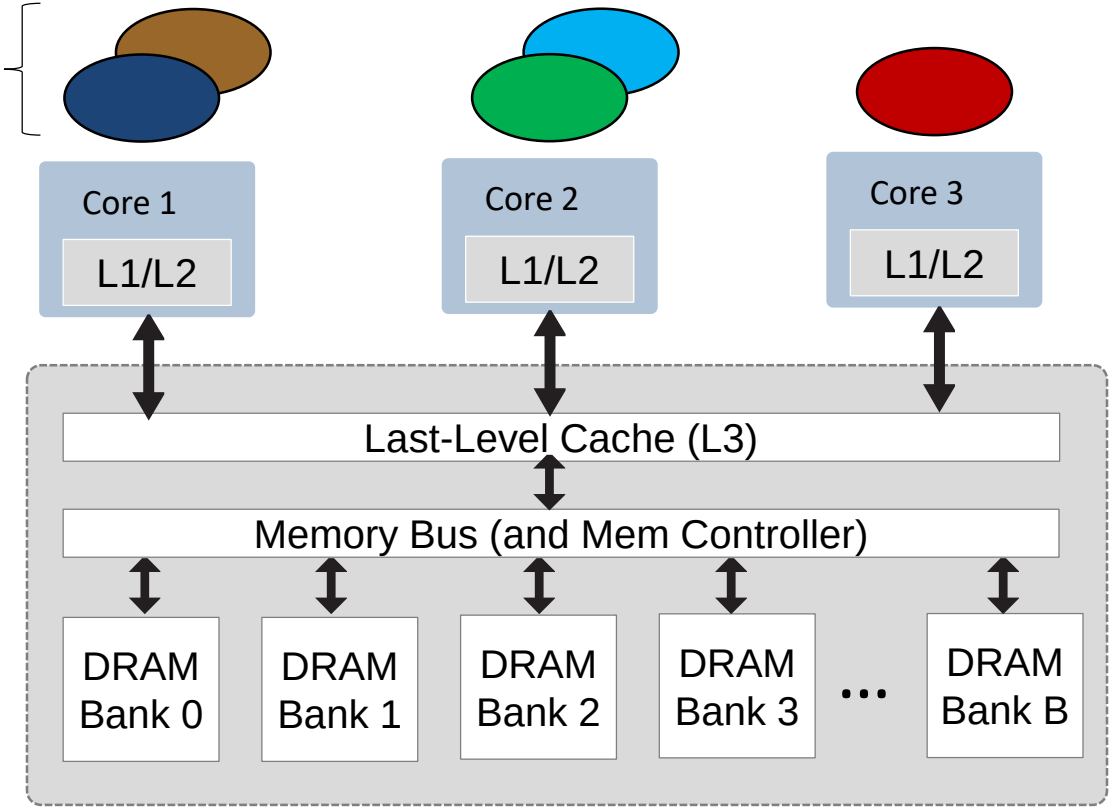
Different reasons for treating resources as undocumented

- Resources are undocumented



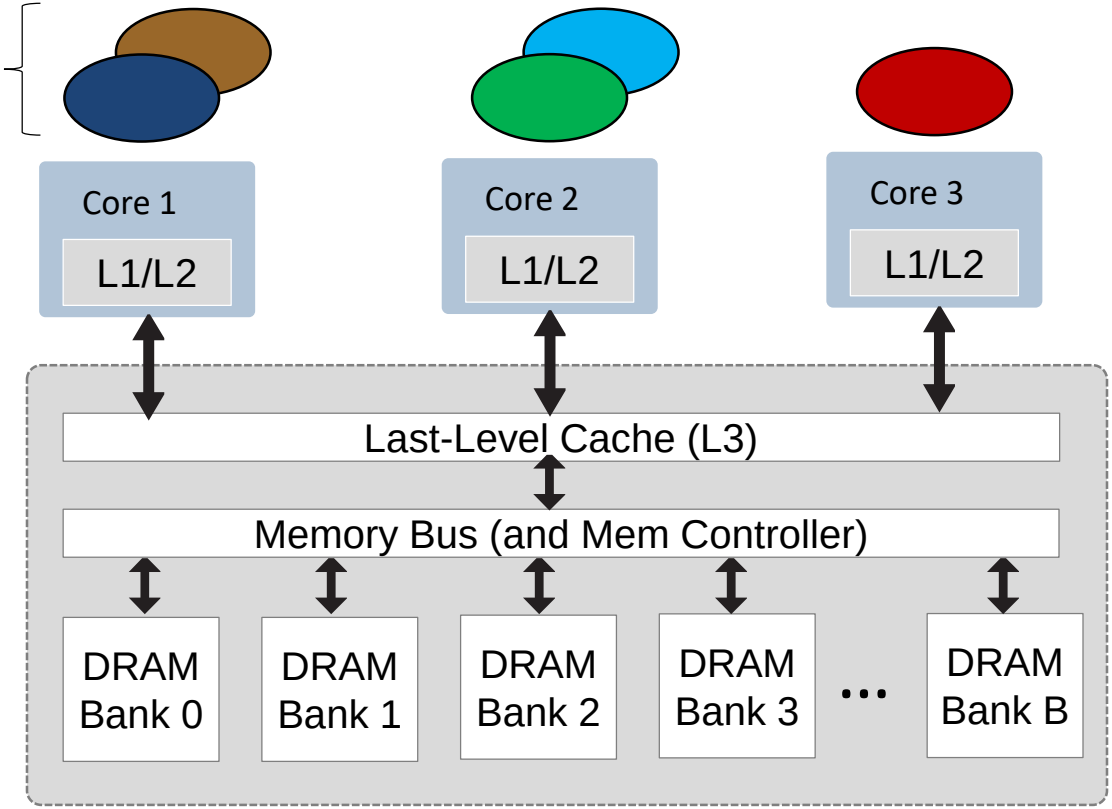
Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect



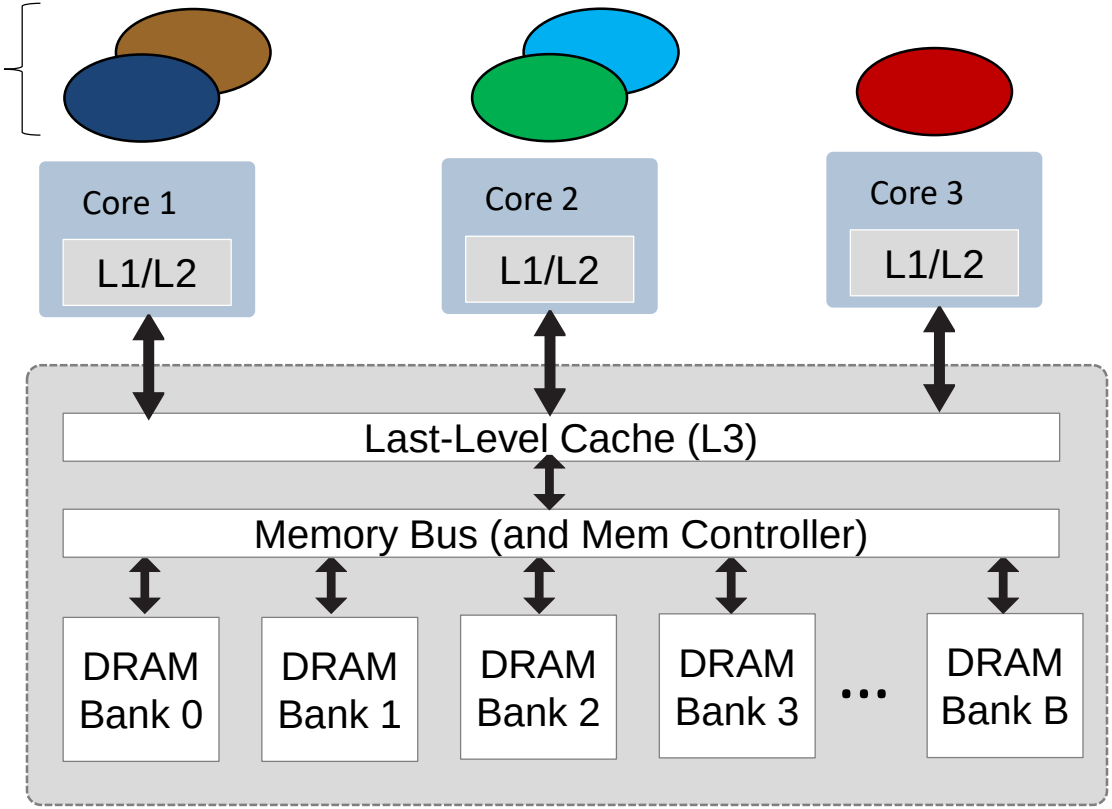
Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
  - Heule:PLDI16:
    50 out of 1795 x86 instructions have incorrect documentation



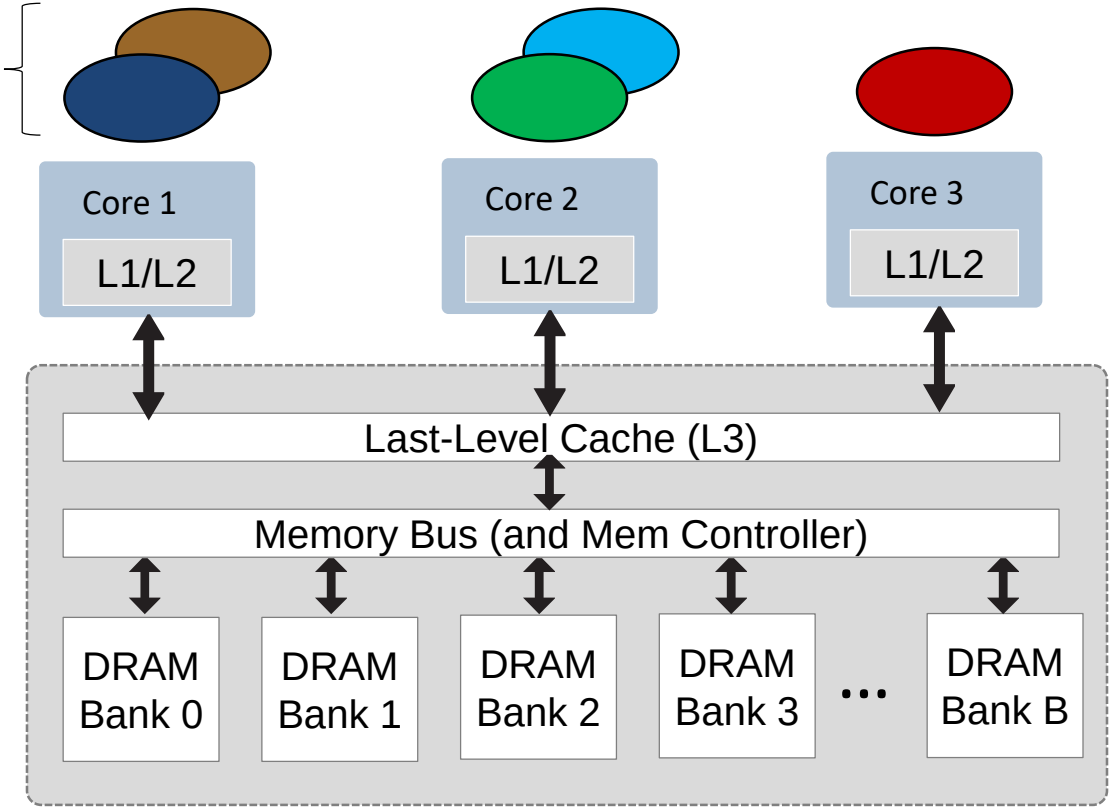
Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
  - Heule:PLDI16:
    50 out of 1795 x86 instructions have incorrect documentation
  - Dasgupta:PLDI19:
    Found incorrect documentation of instructions for x86.



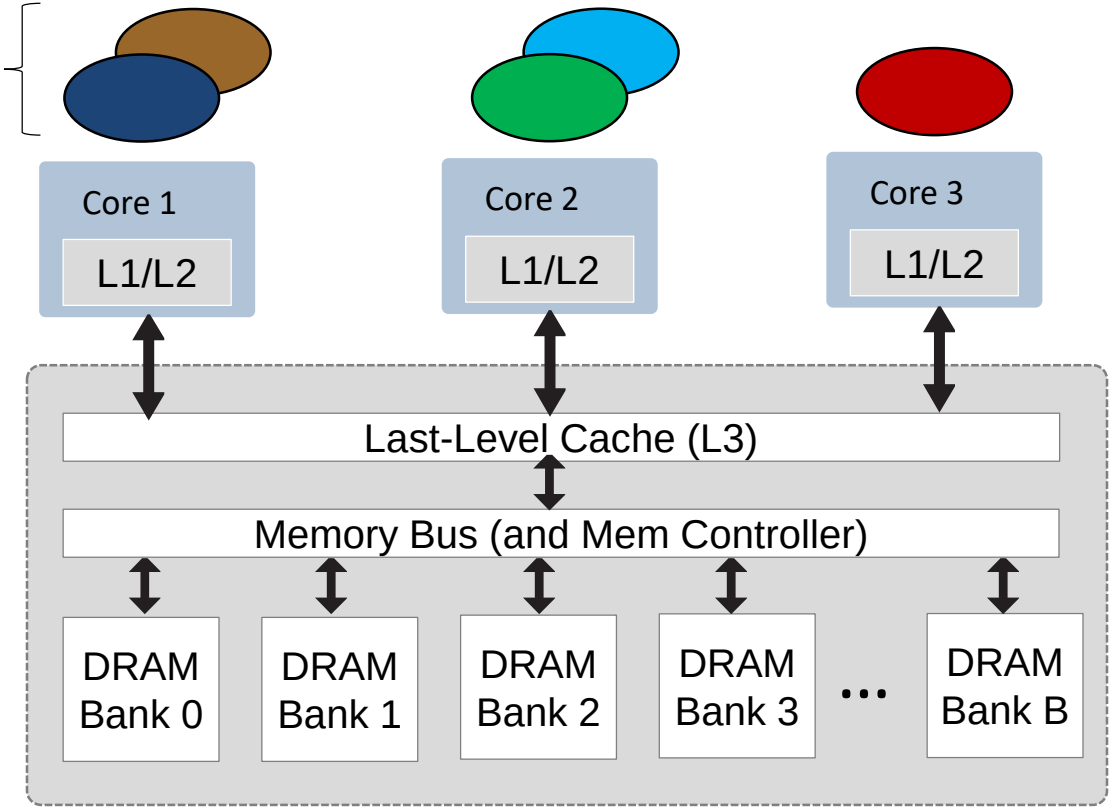
Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
  - Heule:PLDI16:
    50 out of 1795 x86 instructions have incorrect documentation
  - Dasgupta:PLDI19:
    Found incorrect documentation of instructions for x86.
  - Fog19:
    There are discrepancies between measured latencies and latencies in data sheets.



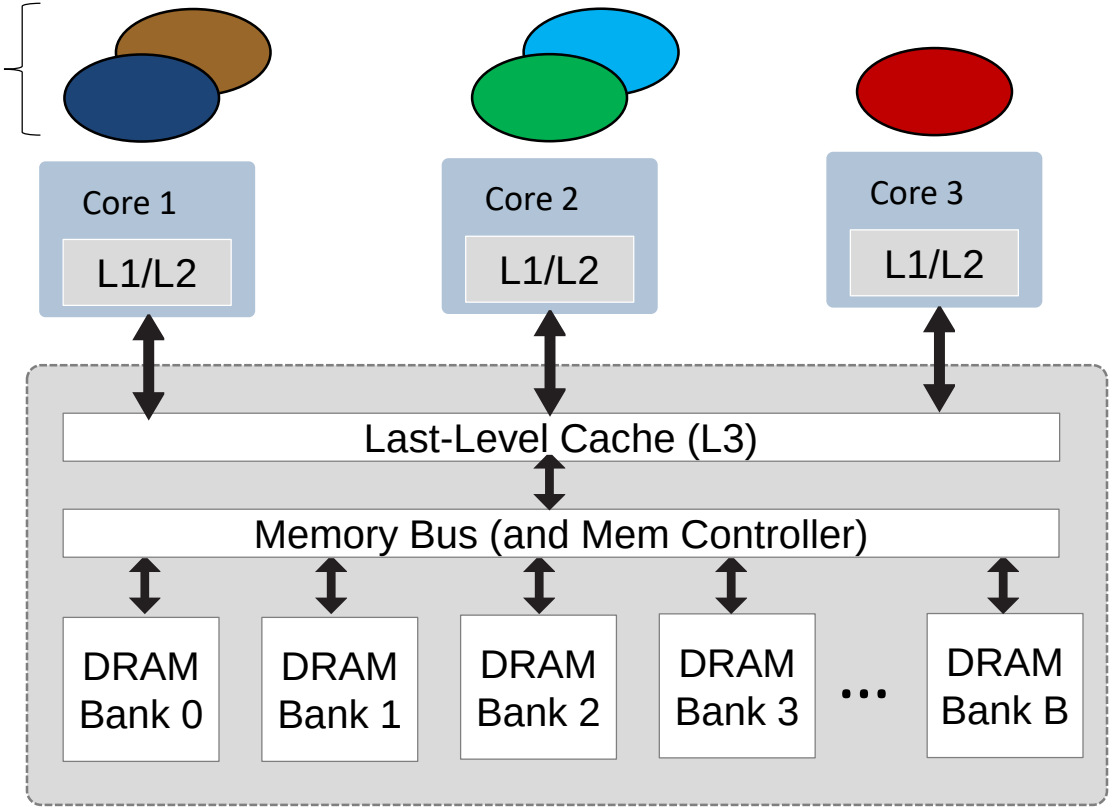
Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
- Resources are documented but one believes the documentation to be incorrect



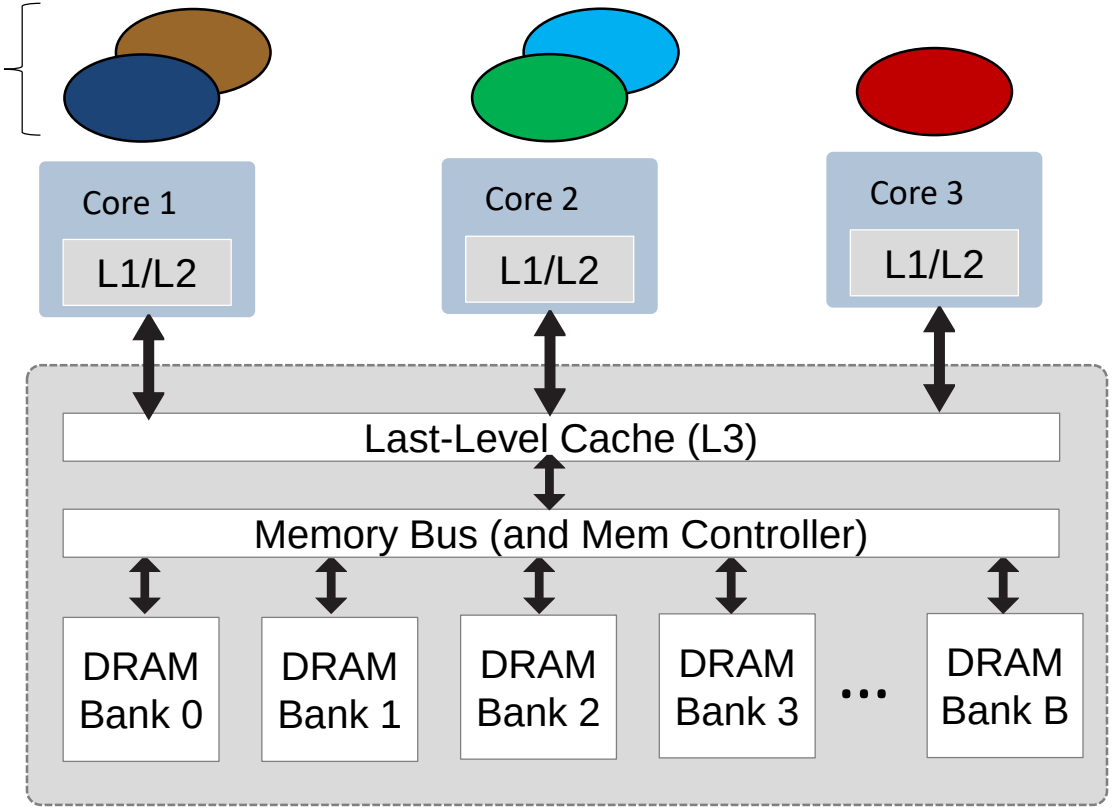
Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
- Resources are documented but one believes the documentation to be incorrect
- Resources are documented and one believe documentation to be correct but it is laborious to create a timing model for schedulability analysis



Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
- Resources are documented but one believes the documentation to be incorrect
- Resources are documented and one believe documentation to be correct but it is laborious to create a timing model for schedulability analysis (and it needs to be changed when one buys a new chip anyway)



The consequences of analyzing timing of software executing on multicores using documentation

The consequences of analyzing timing of software executing on multicores using documentation

The consequences of analyzing timing of software executing on multicores using documentation

Project starts: We want to develop product X

The consequences of analyzing timing of software executing on multicores using documentation

The consequences of analyzing timing of software executing on multicores using documentation

Project starts: We want to develop product X

Buy multi-core proc-essor Y

Read data sheets of multicore processor Y

The consequences of analyzing timing of software executing on multicores using documentation

Project starts: We want to develop product X

Buy multi-core proc-essor Y

Read data sheets of multicore processor Y

Create model of hardware and develop schedulability analysis equations

The consequences of analyzing timing of software executing on multicores using documentation

Project starts: We want to develop product X

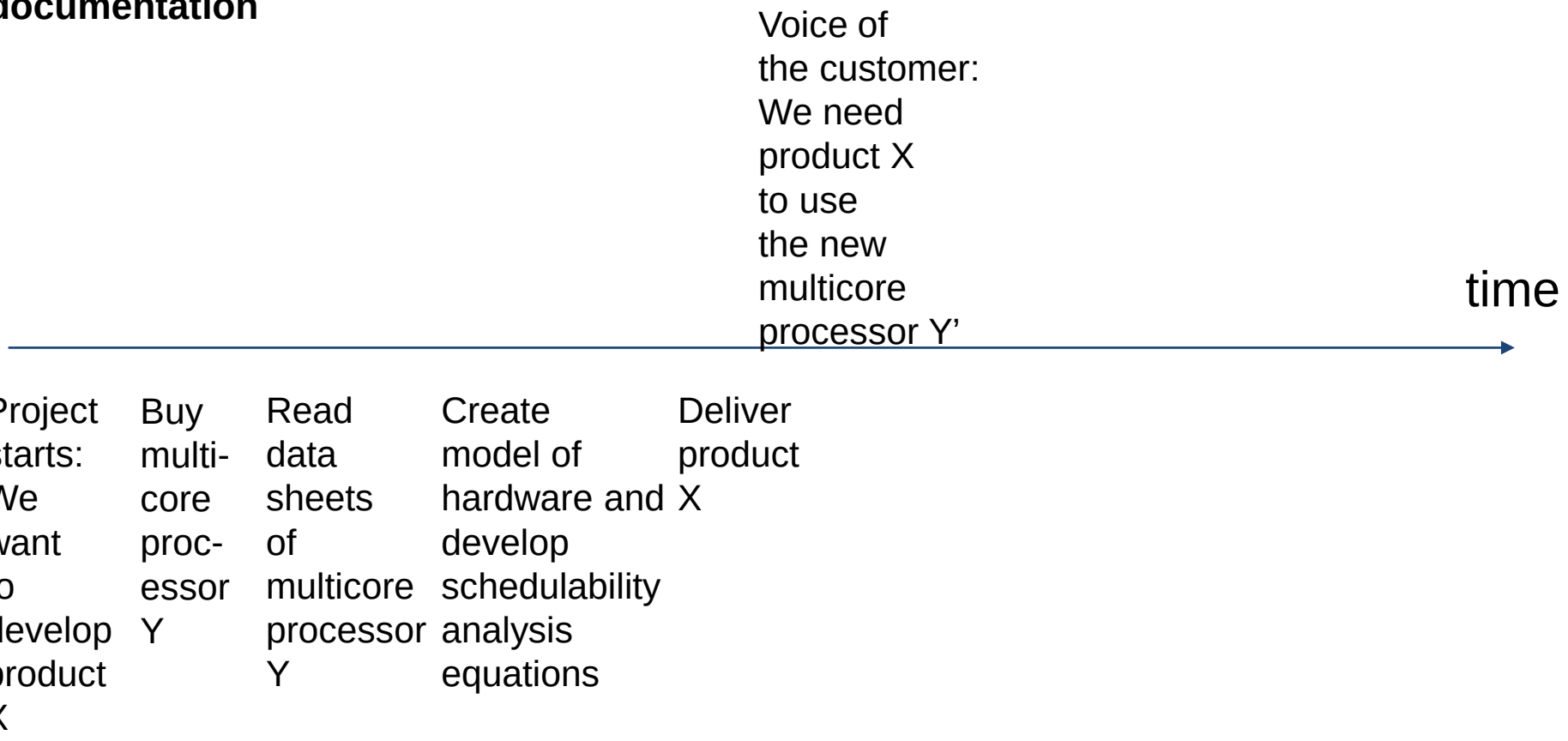
Buy multi-core proc-essor Y

Read data sheets of multicore processor Y

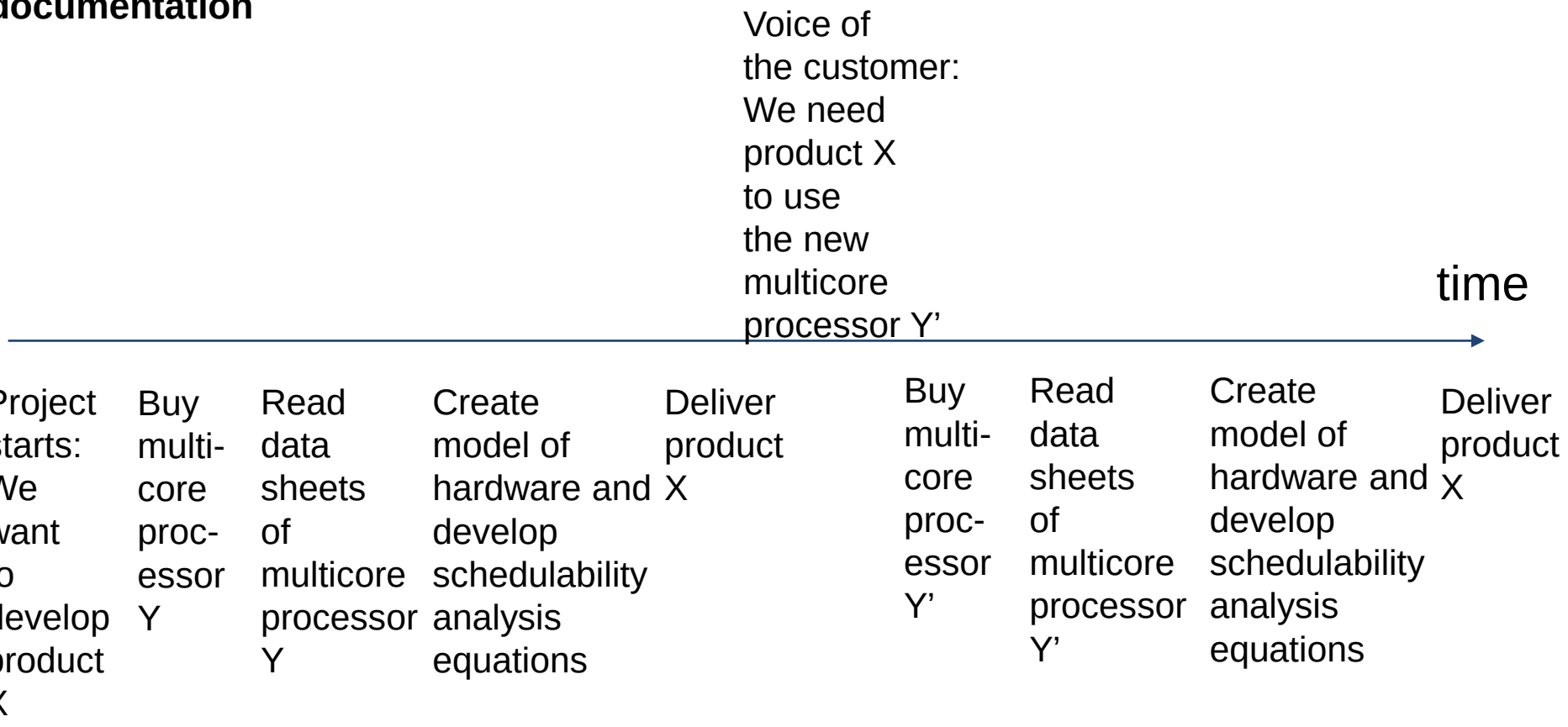
Create model of hardware and develop schedulability analysis equations

Deliver product X

The consequences of analyzing timing of software executing on multicores using documentation



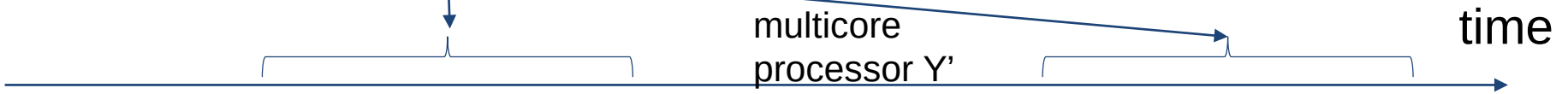
The consequences of analyzing timing of software executing on multicores using documentation



The consequences of analyzing timing of software executing on multicores using documentation

This has to be repeated for each hardware upgrade. Takes a long time. Requires PhD level skills.

Voice of the customer: We need product X to use the new multicore processor Y'



Project starts: We want to develop product X

Buy multi-core proc-essor Y

Read data sheets of multicore processor Y

Create model of hardware and develop schedulability analysis equations

Deliver product X

Buy multi-core proc-essor Y'

Read data sheets of multicore processor Y'

Create model of hardware and develop schedulability analysis equations

Deliver product X

The consequences of analyzing timing of software executing on multicores using documentation

New multicore processor Y becomes available

The consequences of analyzing timing of software executing on multicores using documentation

New multicore processor Y becomes available

time

Academic researcher create model of hardware and develop schedulability analysis equations for Y

The consequences of analyzing timing of software executing on multicores using documentation

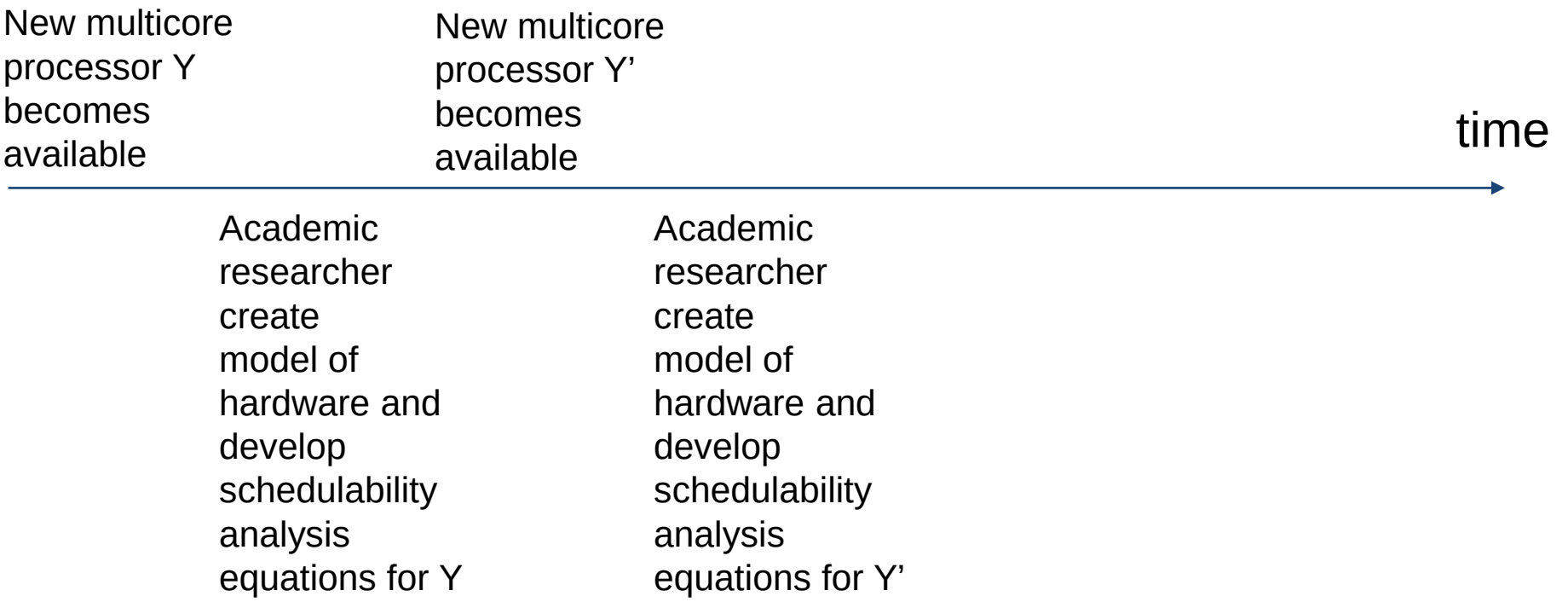
New multicore processor Y becomes available

New multicore processor Y' becomes available

time

Academic researcher create model of hardware and develop schedulability analysis equations for Y

The consequences of analyzing timing of software executing on multicores using documentation



The consequences of analyzing timing of software executing on multicores using documentation

New multicore processor Y becomes available

New multicore processor Y' becomes available

New multicore processor Y'' becomes available

time

Academic researcher create model of hardware and develop schedulability analysis equations for Y

Academic researcher create model of hardware and develop schedulability analysis equations for Y'

The consequences of analyzing timing of software executing on multicores using documentation

New multicore processor Y becomes available

New multicore processor Y' becomes available

New multicore processor Y'' becomes available

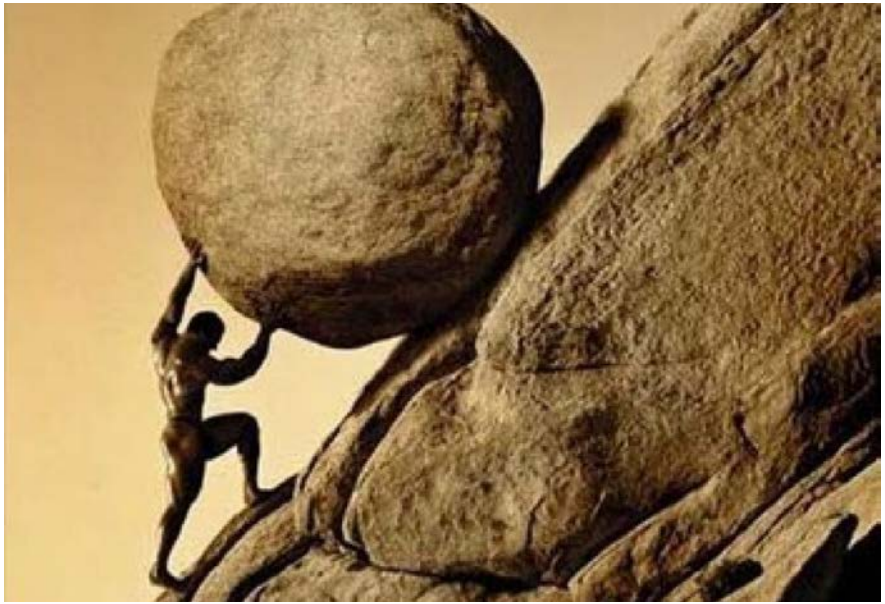
time

Academic researcher create model of hardware and develop schedulability analysis equations for Y

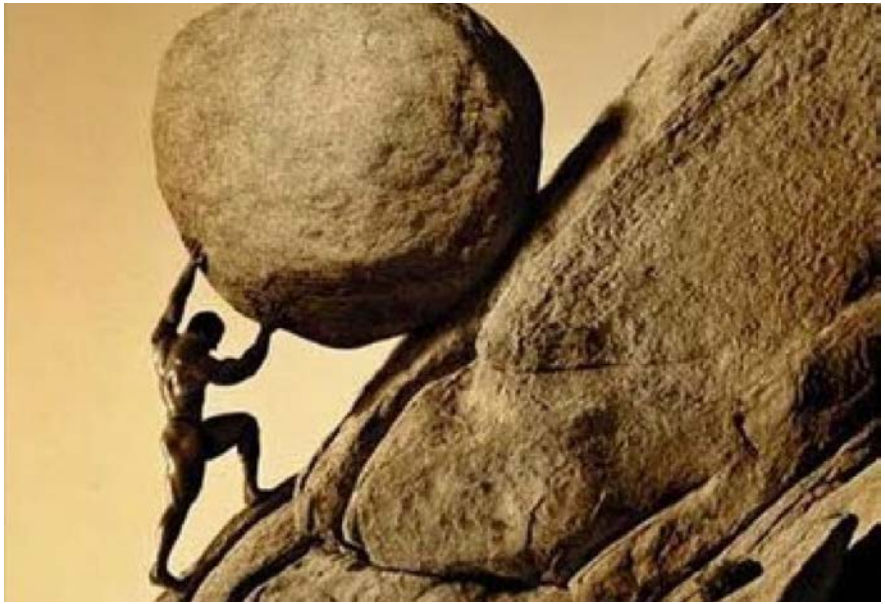
Academic researcher create model of hardware and develop schedulability analysis equations for Y'

Academic researcher create model of hardware and develop schedulability analysis equations for Y''

The consequences of analyzing timing of software executing on multicores using documentation

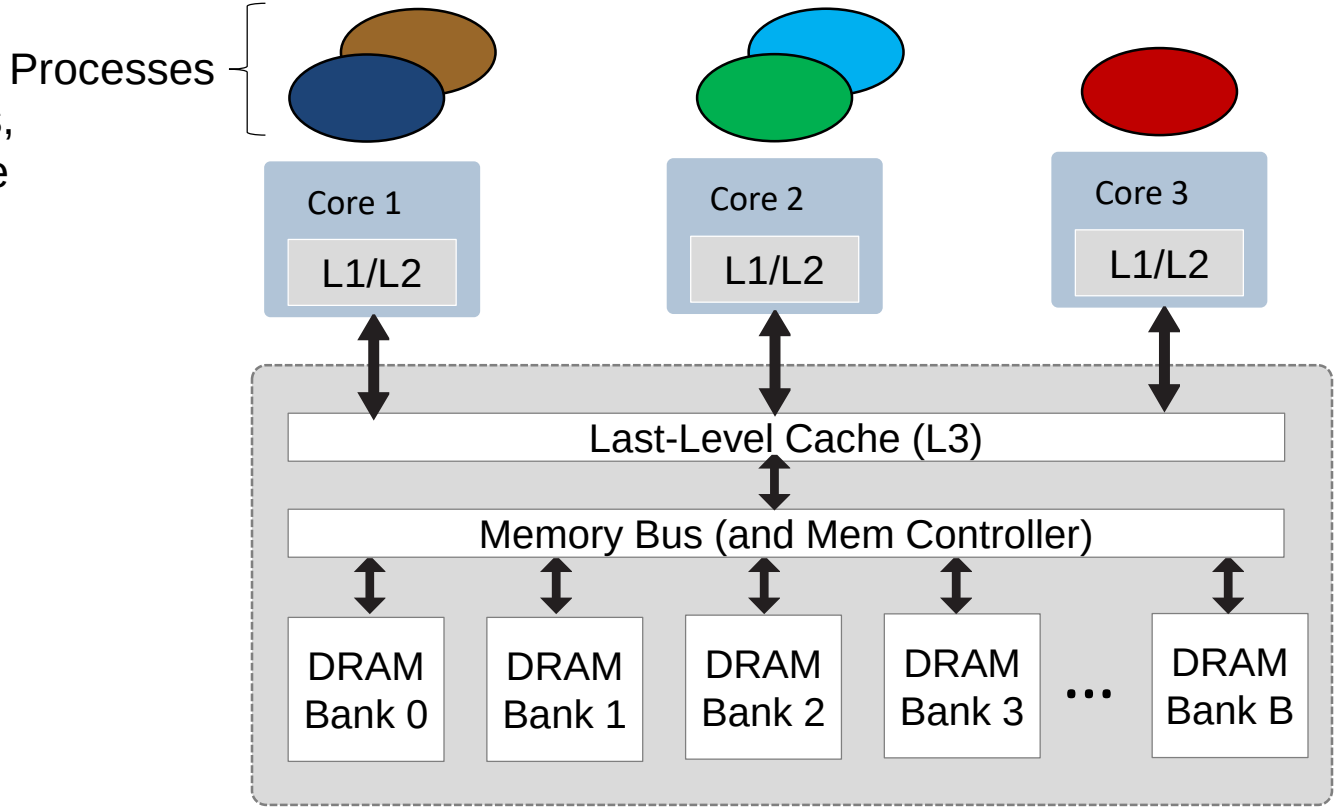


The consequences of analyzing timing of software executing on multicores using documentation

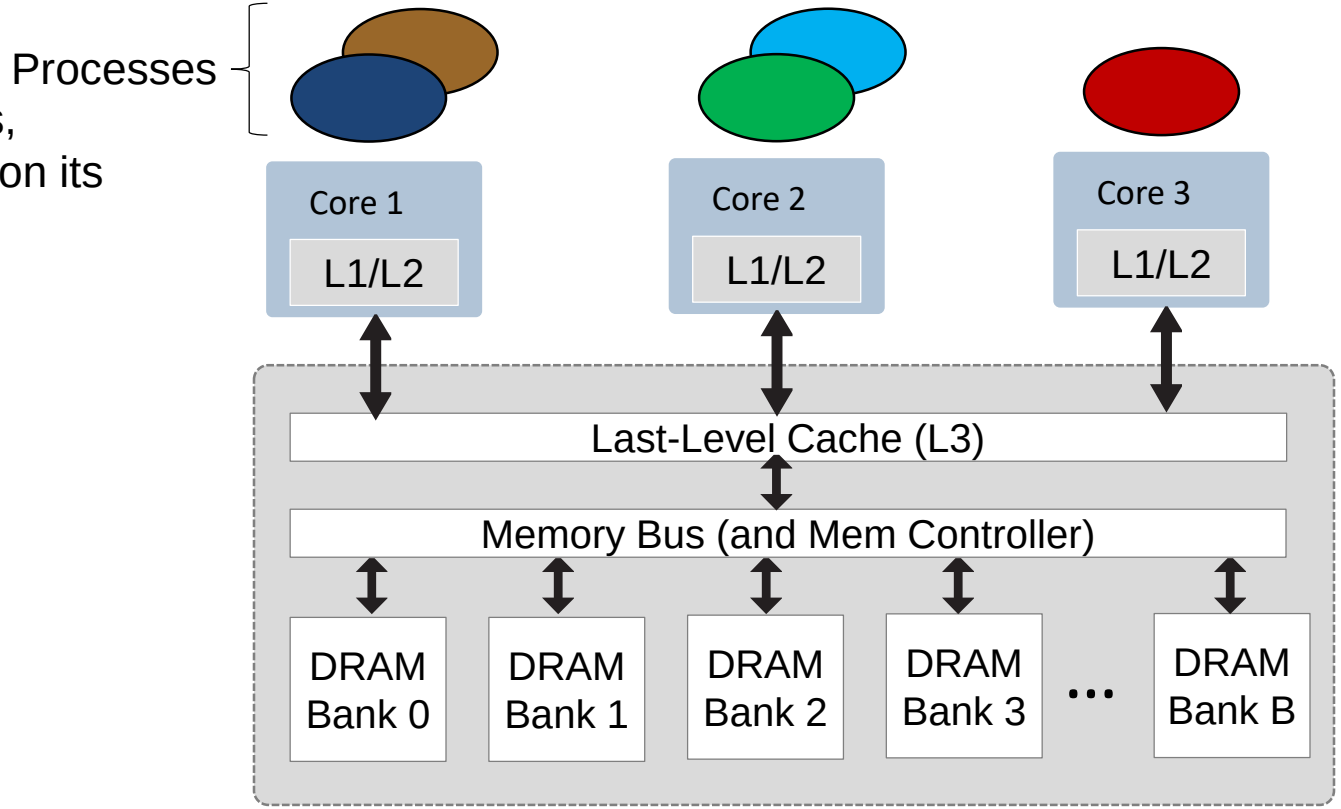


Treating multicore processors as undocumented hardware has the potential to create a model that can be used for processors in the future even for processors that we currently do not know about.

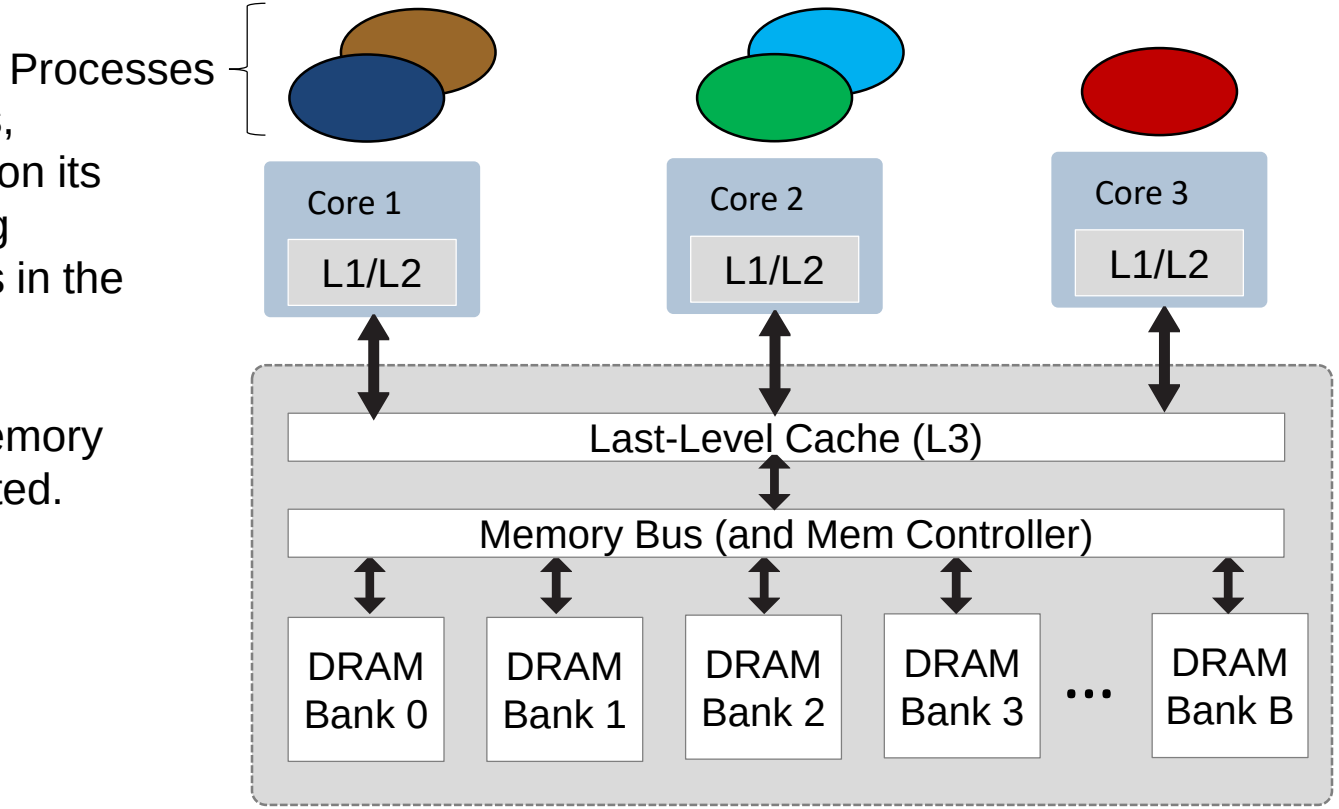
Problem: For each process, compute its response time



Problem: For each process, compute an upper bound on its response time



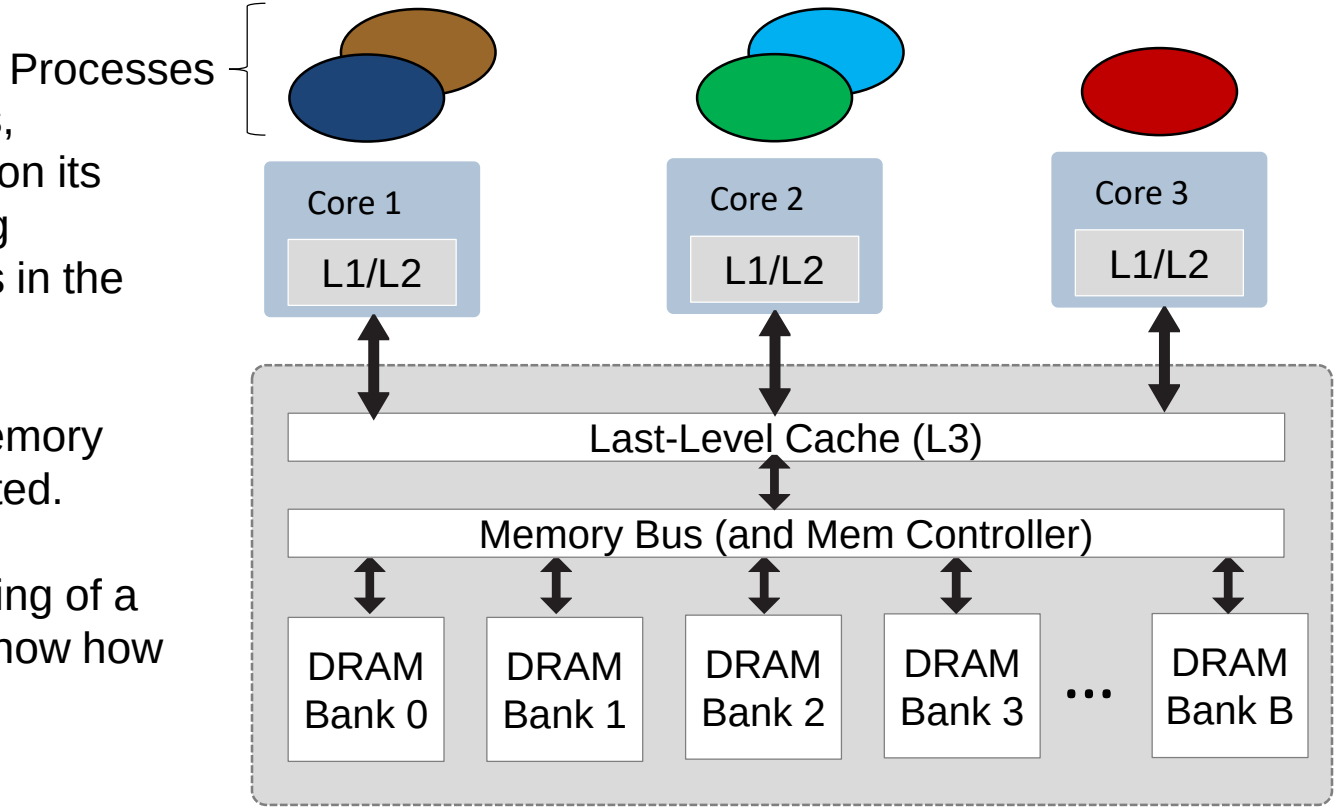
Problem: For each process, compute an upper bound on its response time considering contention for resources in the memory system and that resources in the memory system are undocumented.



Problem: For each process, compute an upper bound on its response time considering
- contention for resources in the memory system

and
- that resources in the memory system are undocumented.

Q: How can we analyze timing of a system when we do not know how the system works?

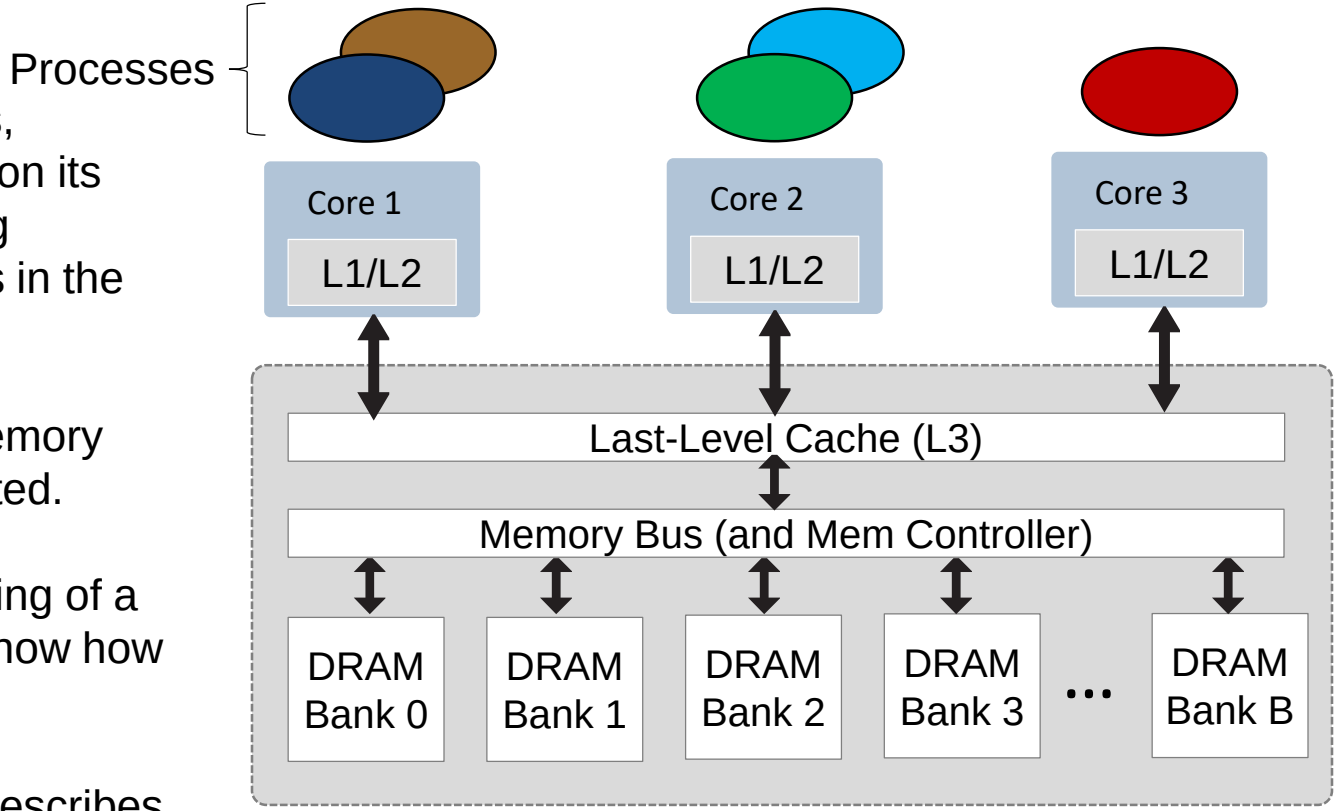


Problem: For each process, compute an upper bound on its response time considering
- contention for resources in the memory system

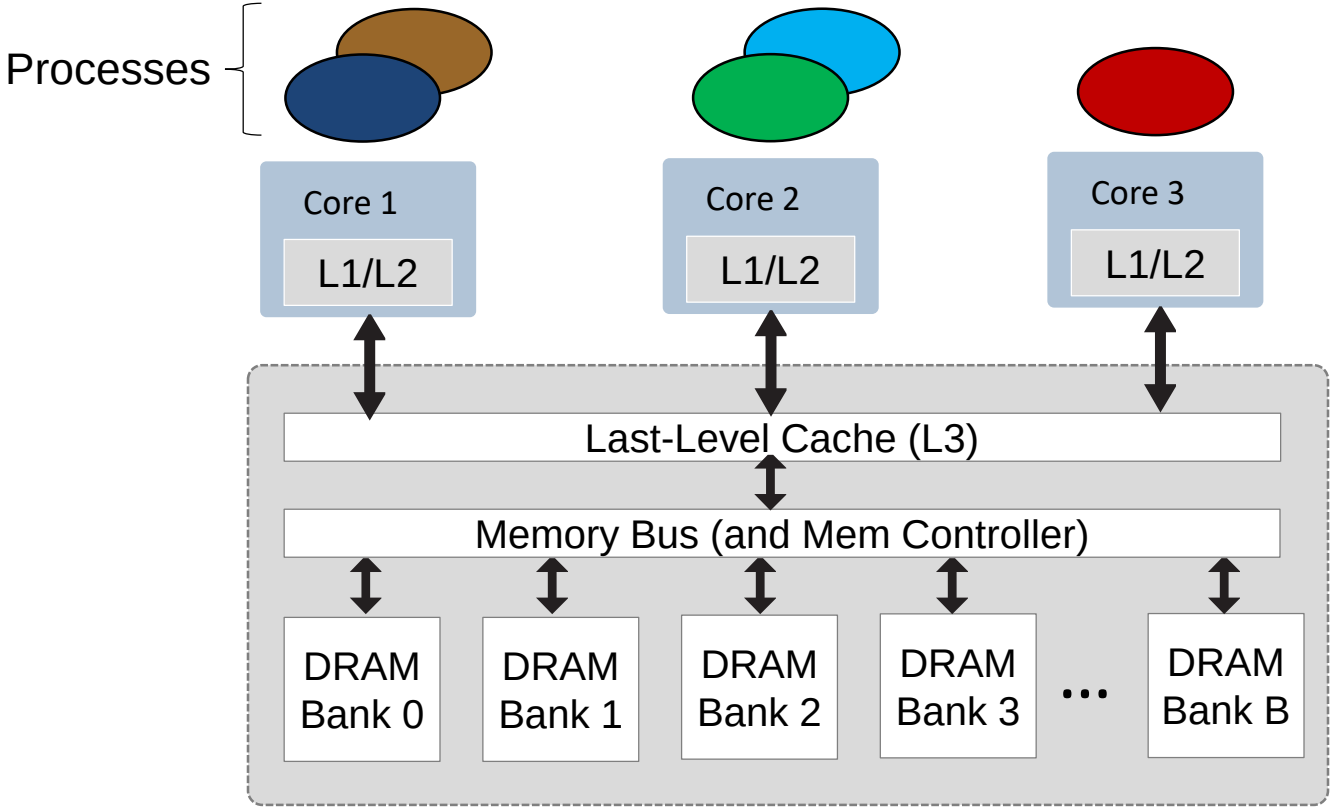
and
- that resources in the memory system are undocumented.

Q: How can we analyze timing of a system when we do not know how the system works?

A: Create abstraction that describes effect of undocumented h/w.



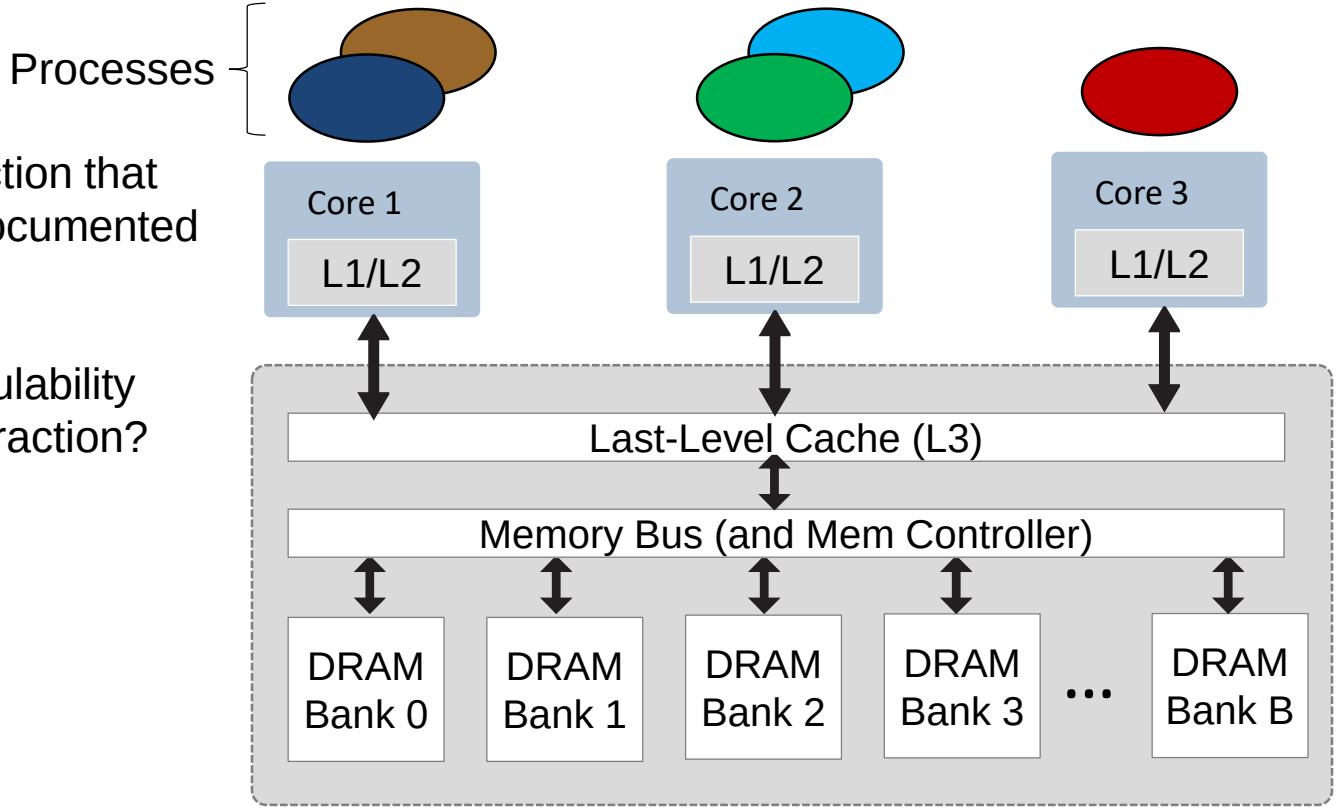
Big Research Questions



Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?



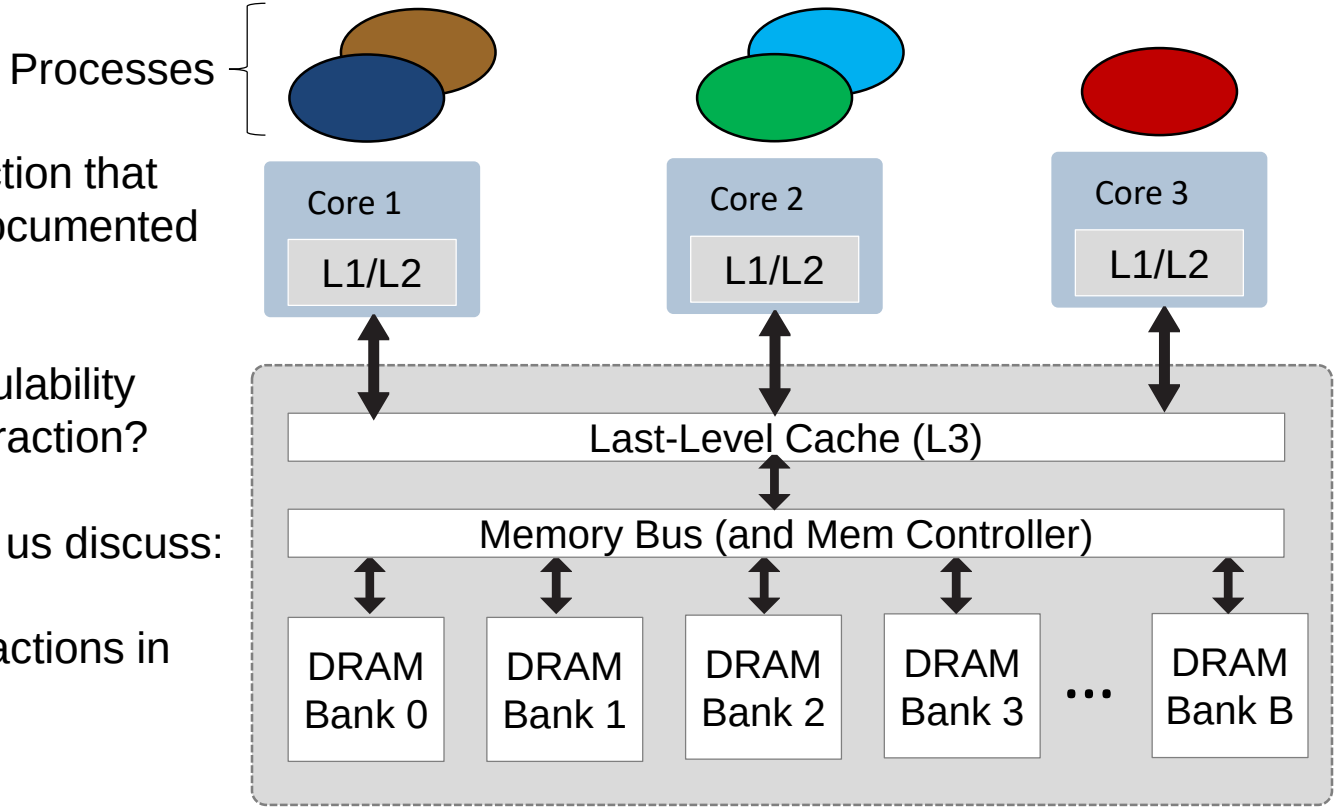
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

Before discussing them, let us discuss:

1. Other approaches
2. General ideas for abstractions in other disciplines



Other approaches

Other approaches

Ignore timing requirements

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Build your own processor (FPGA)

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Build your own processor (FPGA)

Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Build your own processor (FPGA)

Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis

Model some resources and create analysis

Other approaches

Ignore timing requirements	Ignore timing aspects of memory system	Disable all processor cores except one	Use simple processor and run heavy computations on GPU	Build your own processor (ASIC)
Build your own processor (FPGA)	Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis	Model some resources and create analysis	Software-based mechanism to improve predictability/isolation	

Other approaches

Ignore timing requirements	Ignore timing aspects of memory system	Disable all processor cores except one	Use simple processor and run heavy computations on GPU	Build your own processor (ASIC)
Build your own processor (FPGA)	Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis	Model some resources and create analysis	Software-based mechanism to improve predictability/isolation	Reorganize program so that at most one program accesses memory at a time

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

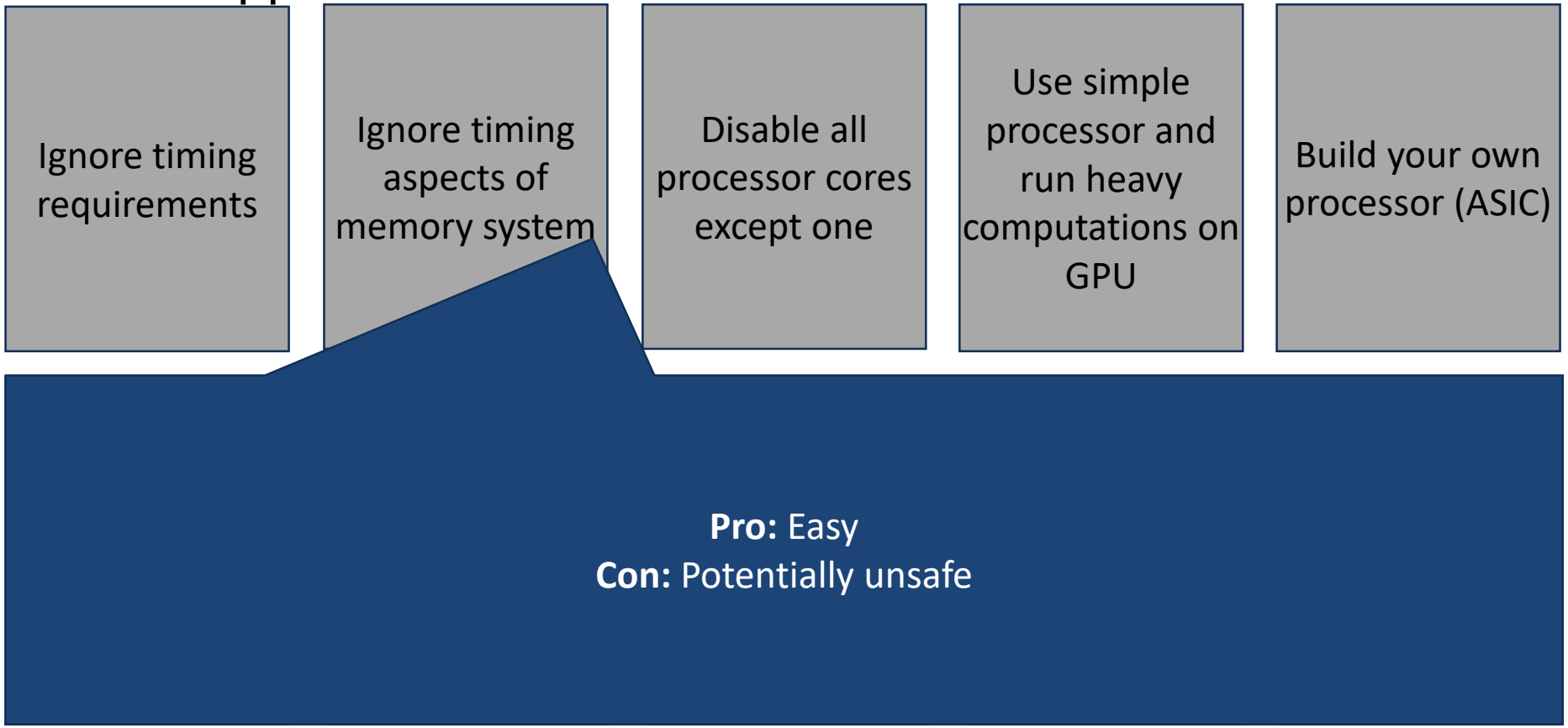
Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Pro: Easy
Con: Potentially unsafe

Other approaches



Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Pro: Easy
Con: Lose lots of processing capacity

Other approaches

Ignore timing requirements	Ignore timing aspects of memory system	Disable all processor cores except one	Use simple processor and run heavy computations on GPU	Build your own processor (ASIC)

Con: Analyzing timing of a single “task” executing on a GPU is still hard.

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Con: High fixed cost.

Other approaches

Con: Low clock frequency

Build your own processor (FPGA)	well-documented hardware (e.g., RISC-V) and model all resources and create analysis	Model some resources and create analysis	Software-based mechanism to improve predictability/isolation	Reorganize program so that at most one program accesses memory at a time

Other approaches

Con: Limits #suppliers. Laborious.

Build your own processor (FPGA)	Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis	Model some resources and create analysis	Software-based mechanism to improve predictability/isolation	Reorganize program so that at most one program accesses memory at a time

Other approaches

Con: Laborious. There may be many resources for which documentation does not exist.

Build your own processor (FPGA)	Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis	Model some resources and create analysis	Software-based mechanism to improve predictability/isolation	Reorganize program so that at most one program accesses memory at a time

Other approaches

Examples: Cache coloring, Cache locking, Bank coloring, MemGuard, TLB coloring.
Con: Requires changes to operating system. Only work for some resources; there are many resources for which there is no isolation mechanism.

Build your own processor (FPGA)	Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis	Model some resources and create analysis	Software-based mechanism to improve predictability/isolation	Reorganize program so that at most one program accesses memory at a time

Other approaches

Examples: PREM, Linkoping@RTSS07.
Con: Laborious. Requires a local memory that is large enough to store working set. Difficult to prove that no memory accesses occurs in certain phases.

Build your own processor (FPGA)	Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis	Model some resources and create analysis	Software-based mechanism to improve predictability/isolation	Reorganize program so that at most one program accesses memory at a time

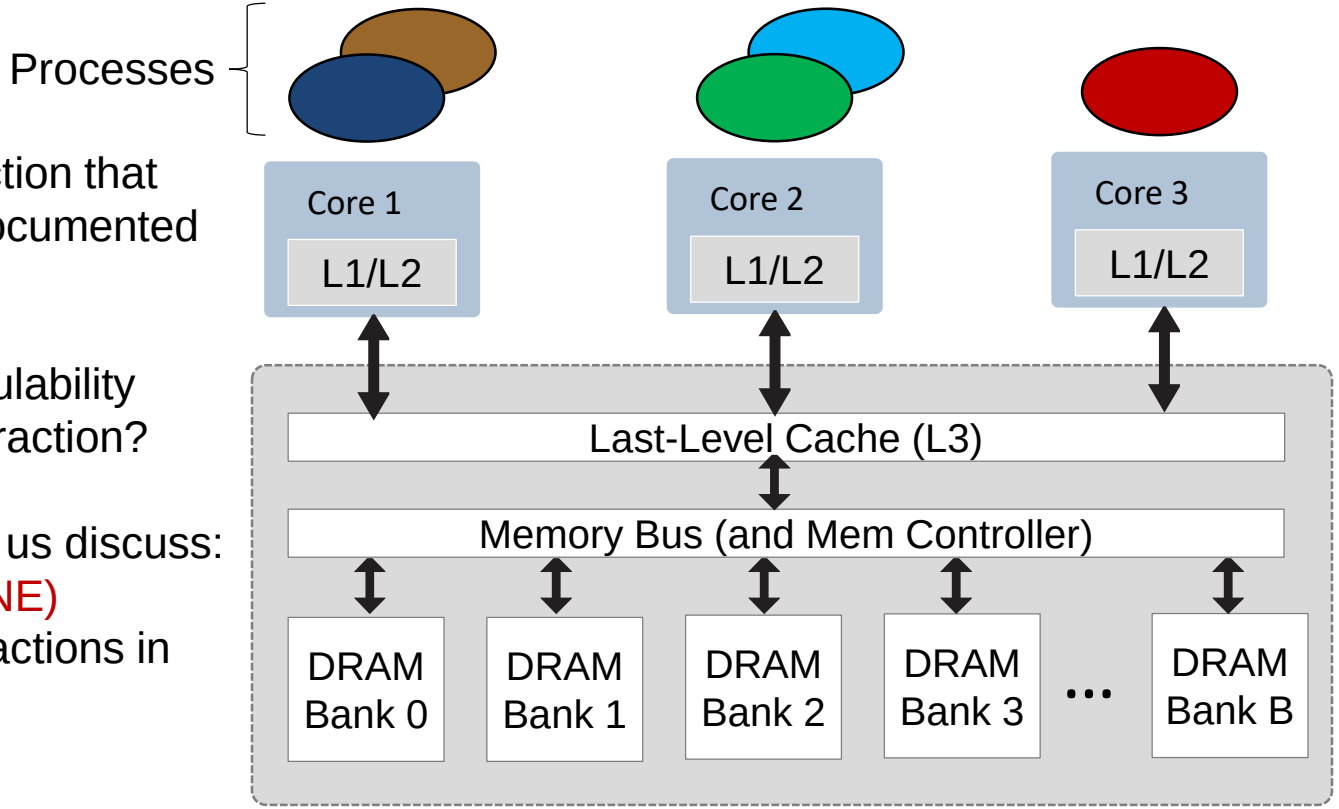
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

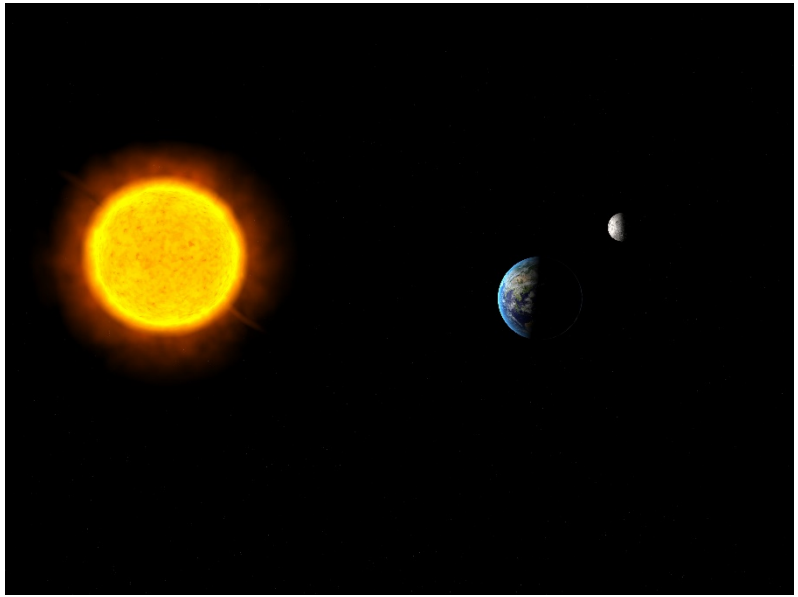
Before discussing them, let us discuss:

1. Other approaches (DONE)
2. General ideas for abstractions in other disciplines

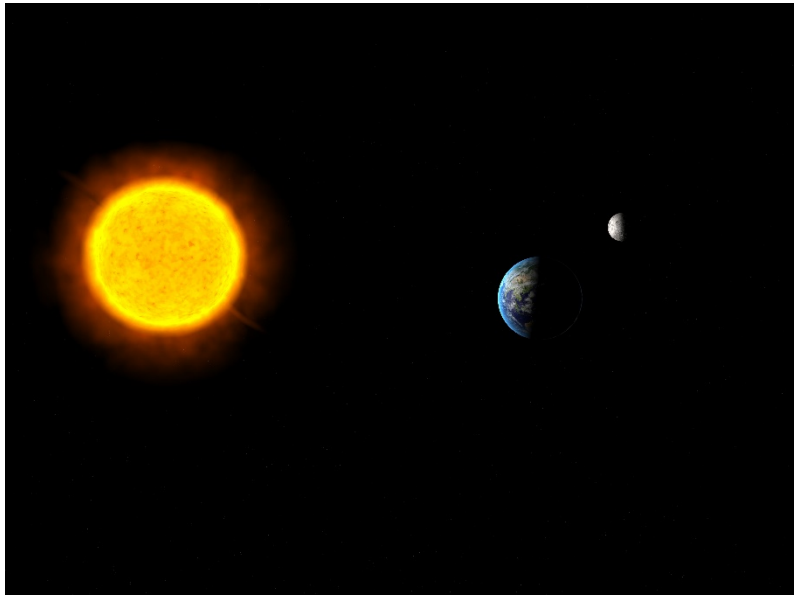


General ideas for abstractions in other disciplines

General ideas for abstractions in other disciplines

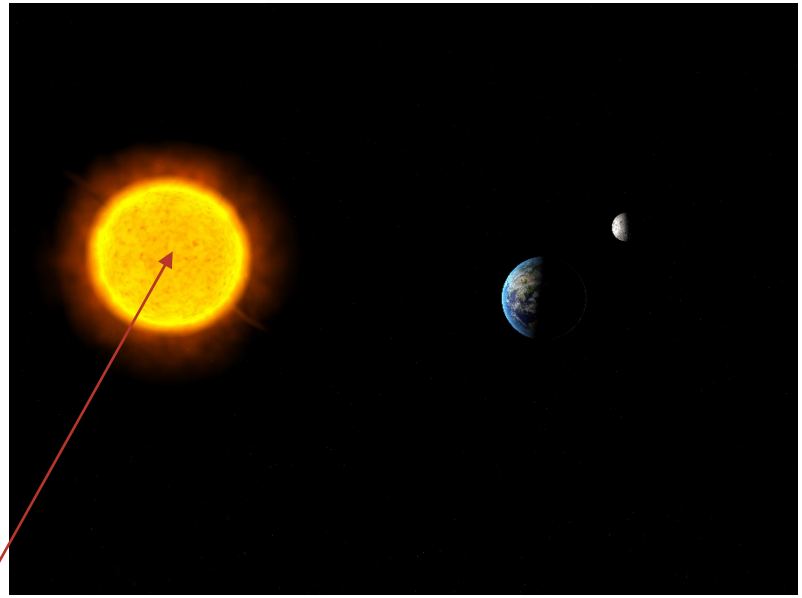


General ideas for abstractions in other disciplines



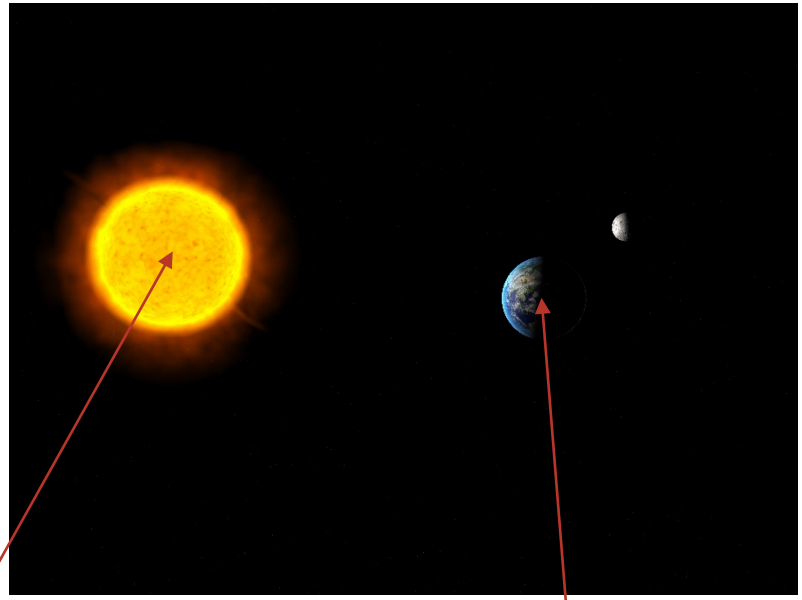
Not drawn to scale

General ideas for abstractions in other disciplines



This body has many atoms

General ideas for abstractions in other disciplines

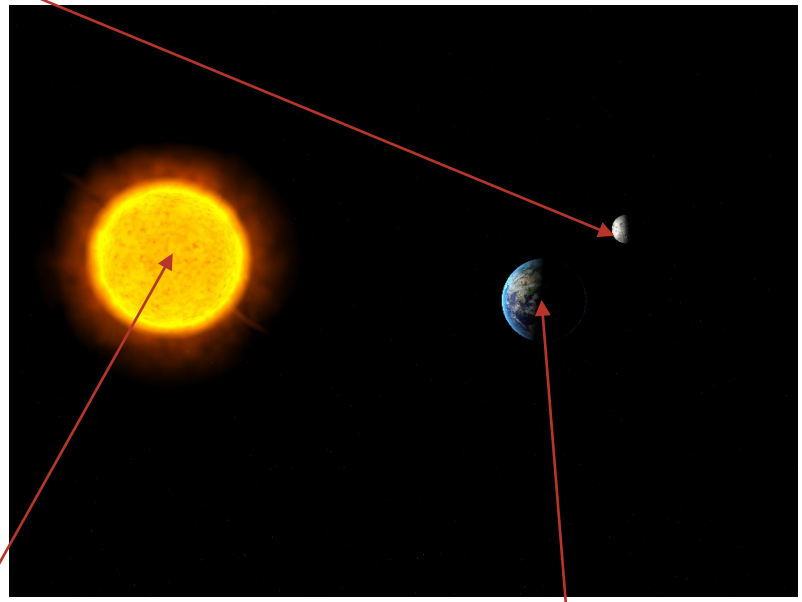


This body has many atoms

This body has many atoms

General ideas for abstractions in other disciplines

This body also has many atoms

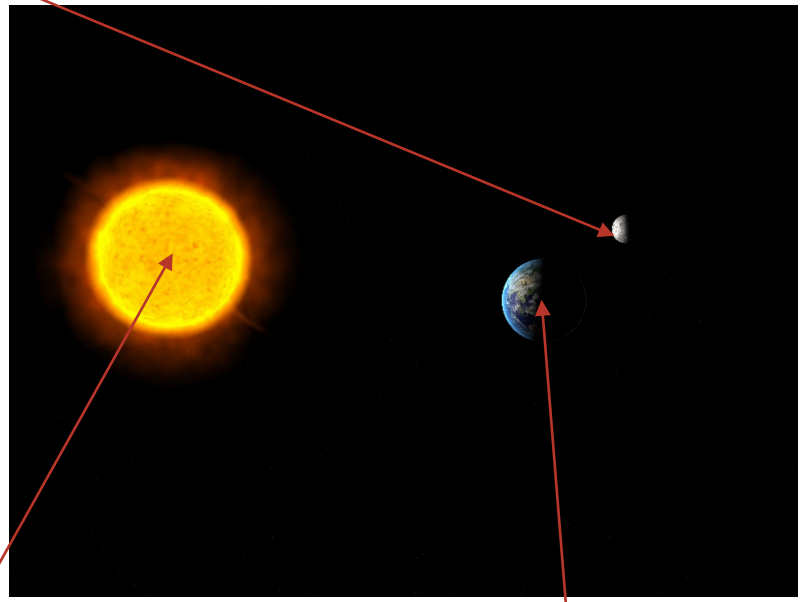


This body has many atoms

This body has many atoms

General ideas for abstractions in other disciplines

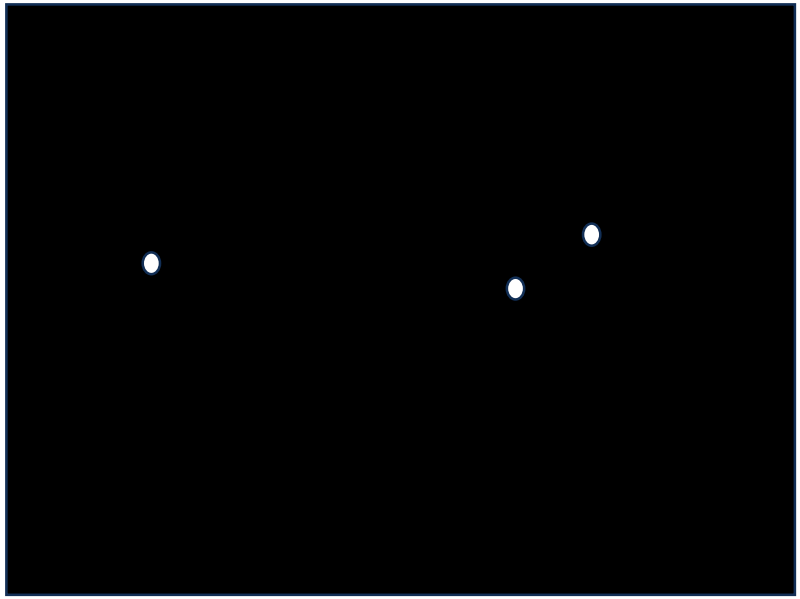
This body also has many atoms



This body has many atoms

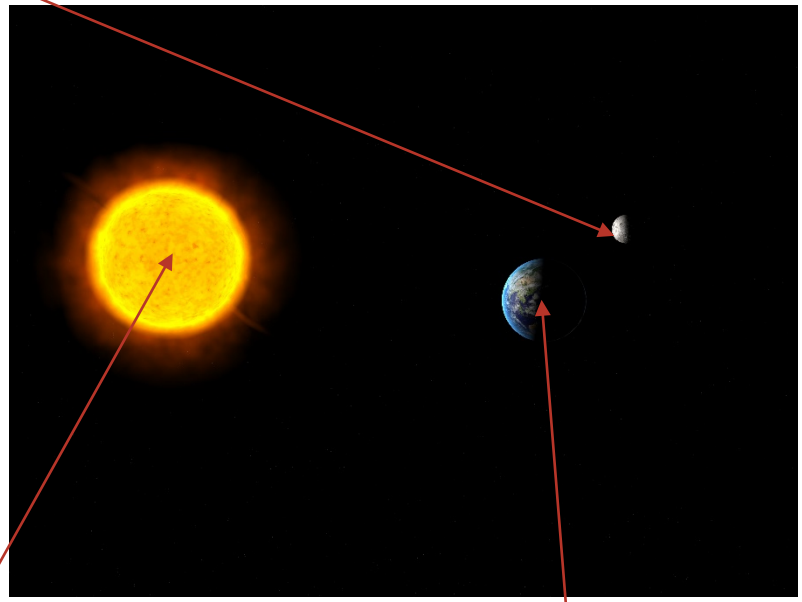
This body has many atoms

We describe each body as a single point mass



General ideas for abstractions in other disciplines

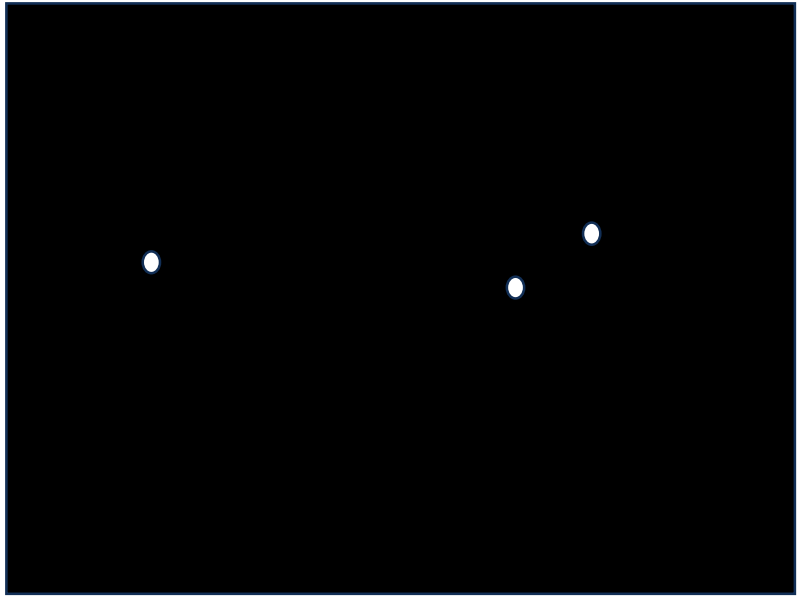
This body also has many atoms



This body has many atoms

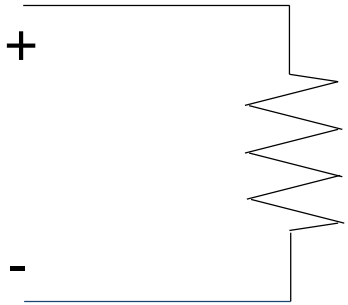
This body has many atoms

We describe each body as a single point mass



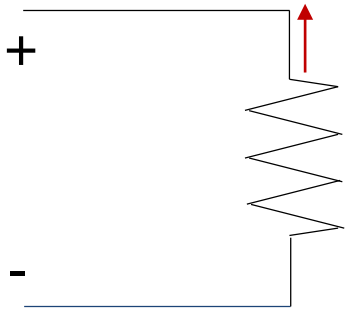
Describe each body with only as many parameters that we need for the analysis that we want to do.

General ideas for abstractions in other disciplines



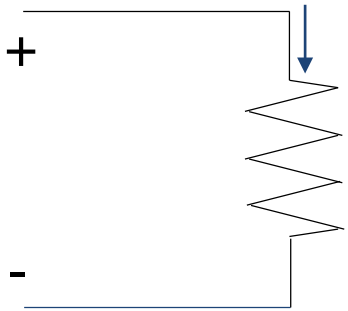
resistor

General ideas for abstractions in other disciplines



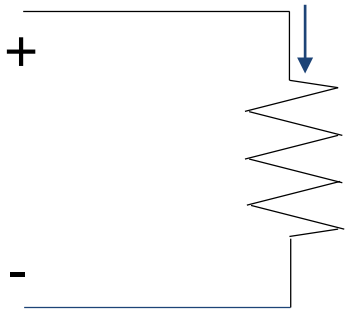
How many electrons per time unit are flowing here?

General ideas for abstractions in other disciplines



How much current is flowing here?

General ideas for abstractions in other disciplines



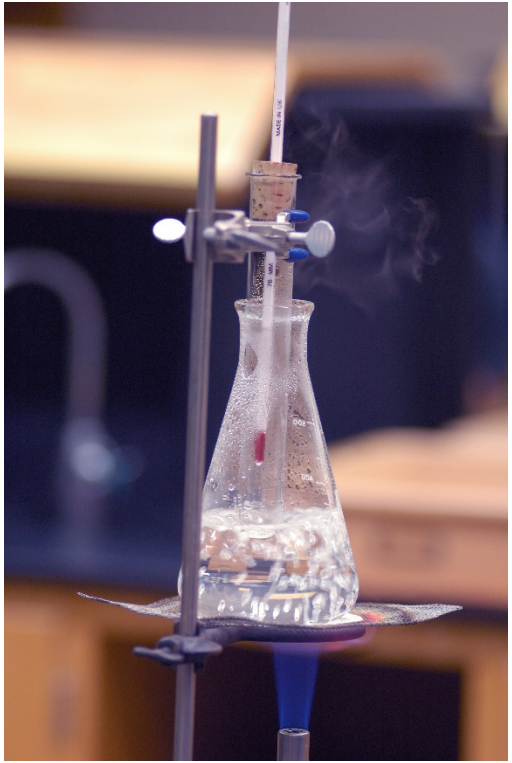
How much current is flowing here?

Ask questions about the aggregate that you care about.

General ideas for abstractions in other disciplines

For **each** of these 10^{24} water molecules,
how fast does this water molecule move?

General ideas for abstractions in other disciplines



What is the water temperature?

Ask questions about the aggregate that you care about.

General ideas for abstractions in other disciplines

What is the water temperature?

If possible: Describe a system with quantities that you can measure.

General ideas for abstractions in other disciplines

1. Describe a system with parts

2. Describe each part with an abstraction

3. Obtain specific values of the abstraction (e.g., using measurements) for each part

4. Ask questions: calculate the answer to the questions

General ideas for abstractions

1. Describe a system with parts

2. Describe each part with an abstraction

3. Obtain specific values of the abstraction (e.g., using measurements) for each part

4. Ask questions: calculate the answer to the questions

General ideas for abstractions

1. Describe a system with parts
   A software system comprises a set of tasks.

2. Describe each part with an abstraction
   Each task is described with a T (period), D (deadline), and C (execution time).

3. Obtain specific values of the abstraction (e.g., using measurements) for each part
   Obtain T from source code. Obtain D from software requirement specification.
   Obtain C by measuring the execution of the program; then add margin.
   (or run WCET tool)

4. Ask questions: calculate the answer to the questions
   Will all deadlines be met if tasks are scheduled by Rate-Monotonic on a single processor?

General ideas for abstractions

1. Describe a system with parts
   A software system comprises a set of tasks.

2. Describe each part with an abstraction
   Each task is described with a T (period), D (deadline), and C (execution time).
   Describe the **effect** of the memory system on execution speed of a task.
3. Obtain specific values of the abstraction (e.g., using measurements) for each part
   Obtain T from source code. Obtain D from software requirement specification.
   Obtain C by measuring the execution of the program; then add margin.
   (or run WCET tool)
   Obtain parameters (e.g., using measurements)

4. Ask questions: calculate the answer to the questions
   Will all deadlines be met if tasks are scheduled by Rate-Monotonic on a single processor?

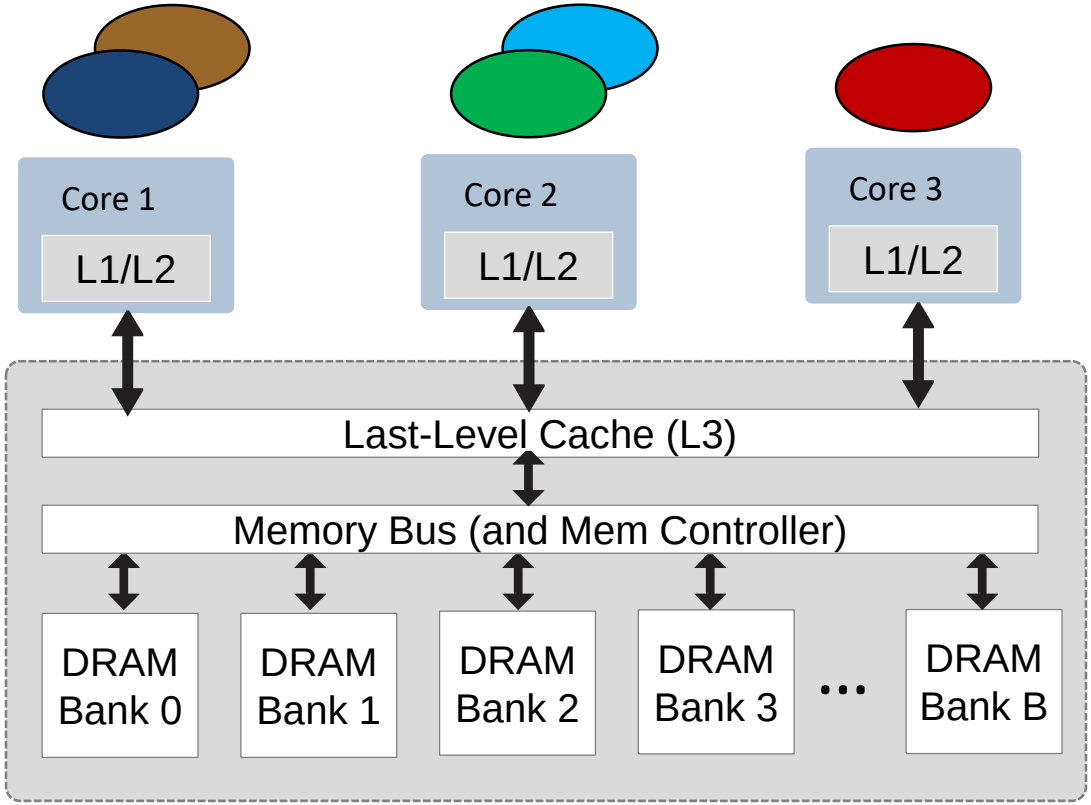
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

Before discussing them, let us discuss:

1. Other approaches (DONE)
2. General ideas for abstractions in other disciplines (DONE)

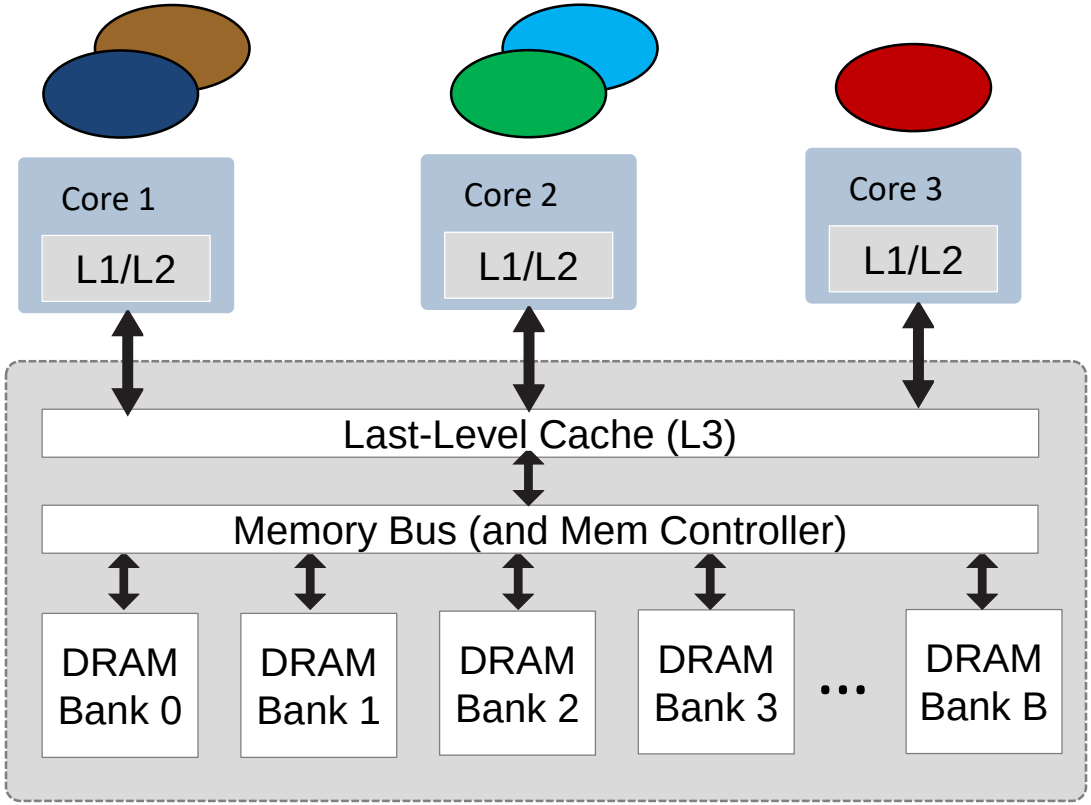


Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

Let us discuss Q1.

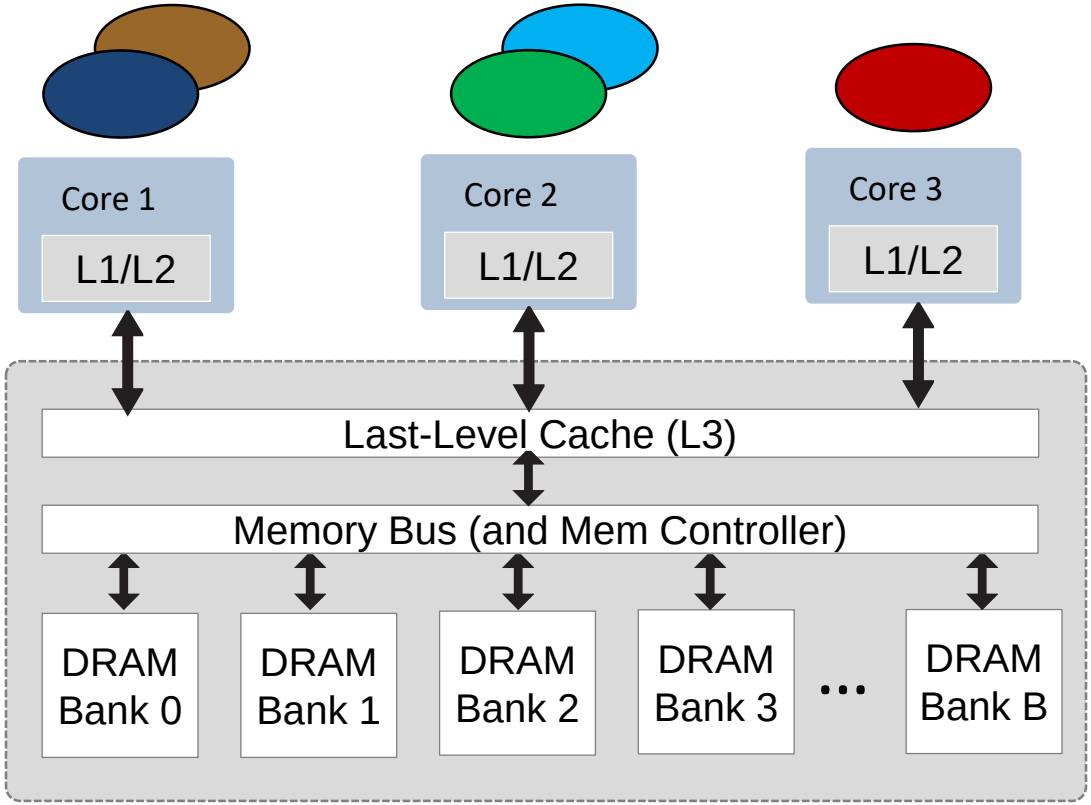


Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

What are the requirements of a good abstraction?



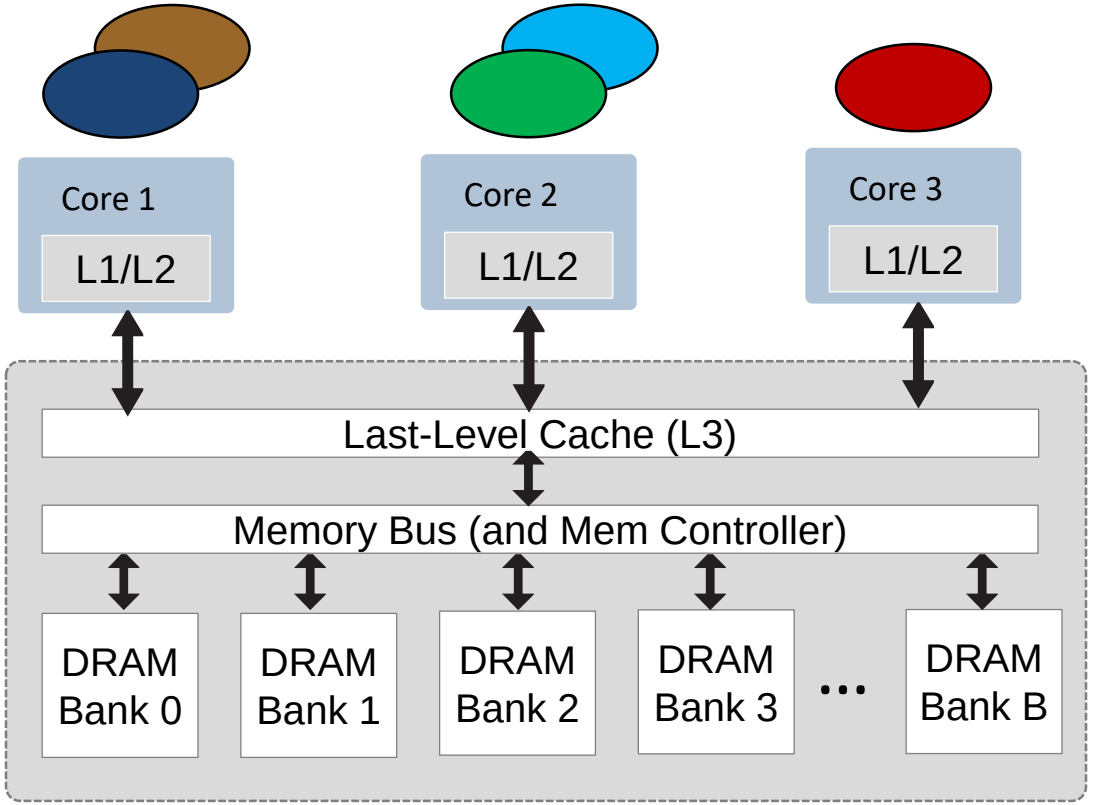
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

What are the requirements of a good abstraction?

1. It should be as small as possible (few numbers; few bits)



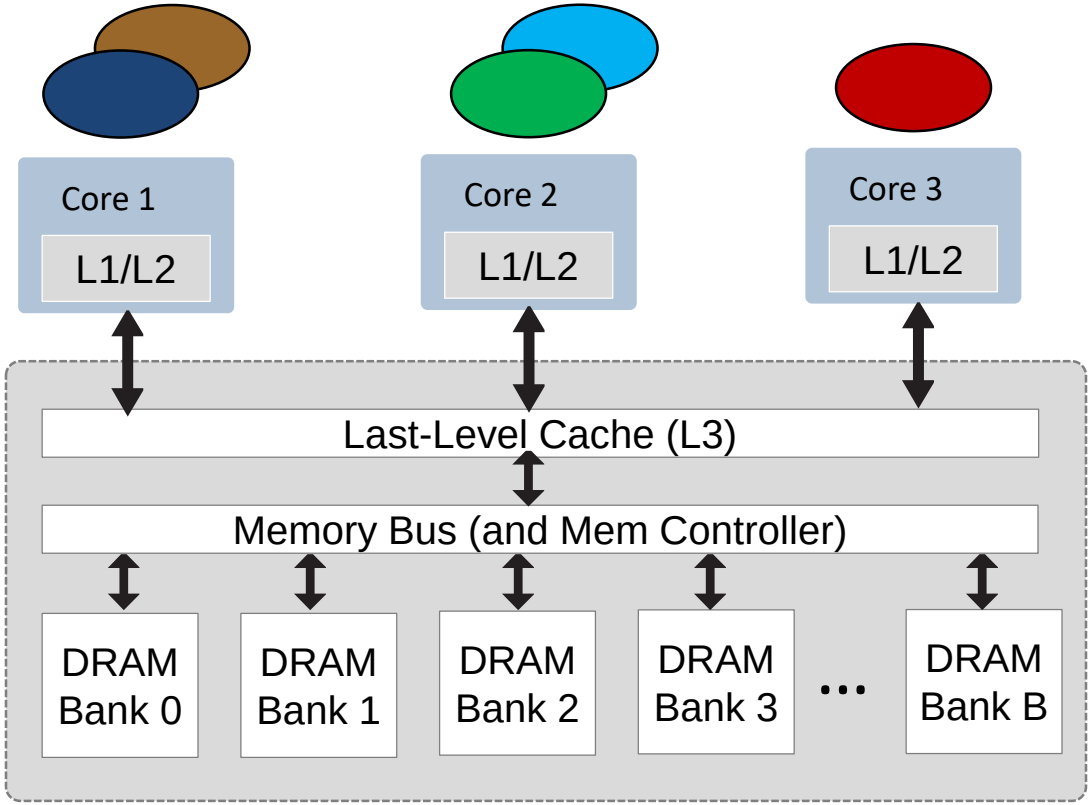
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

What are the requirements of a good abstraction?

1. It should be as small as possible (few numbers; few bits)
2. It should allow us to do prediction/ analysis that we care about



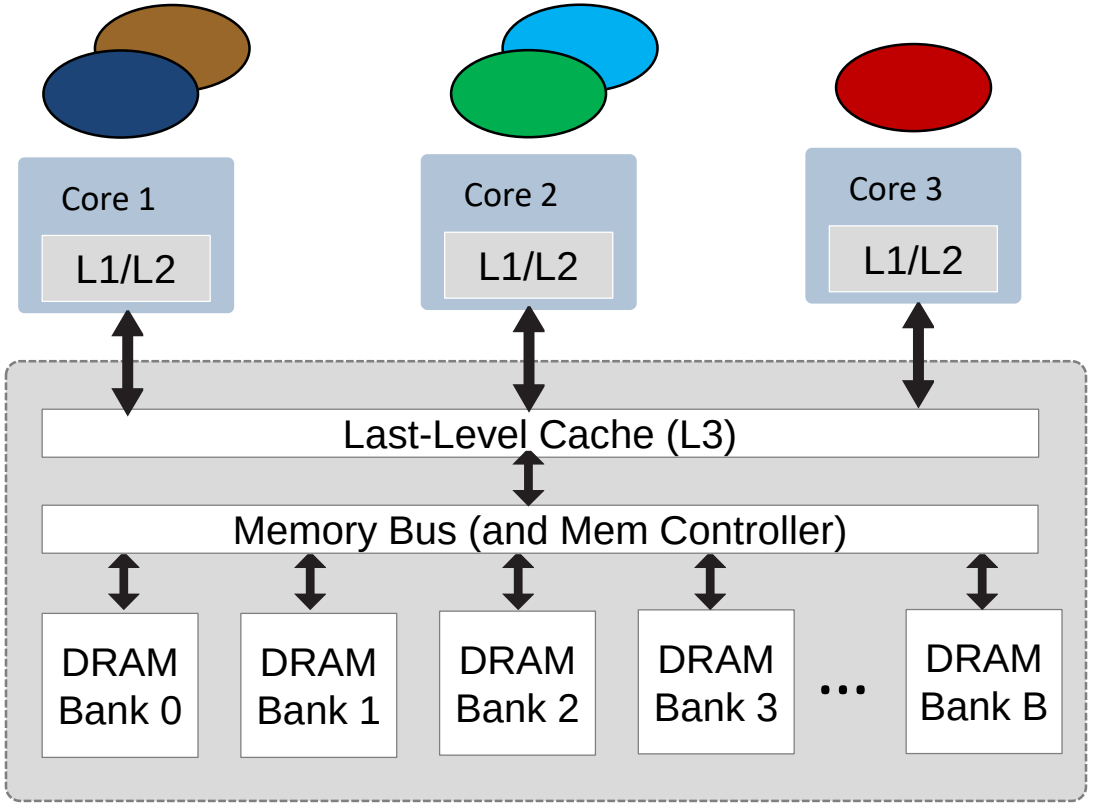
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

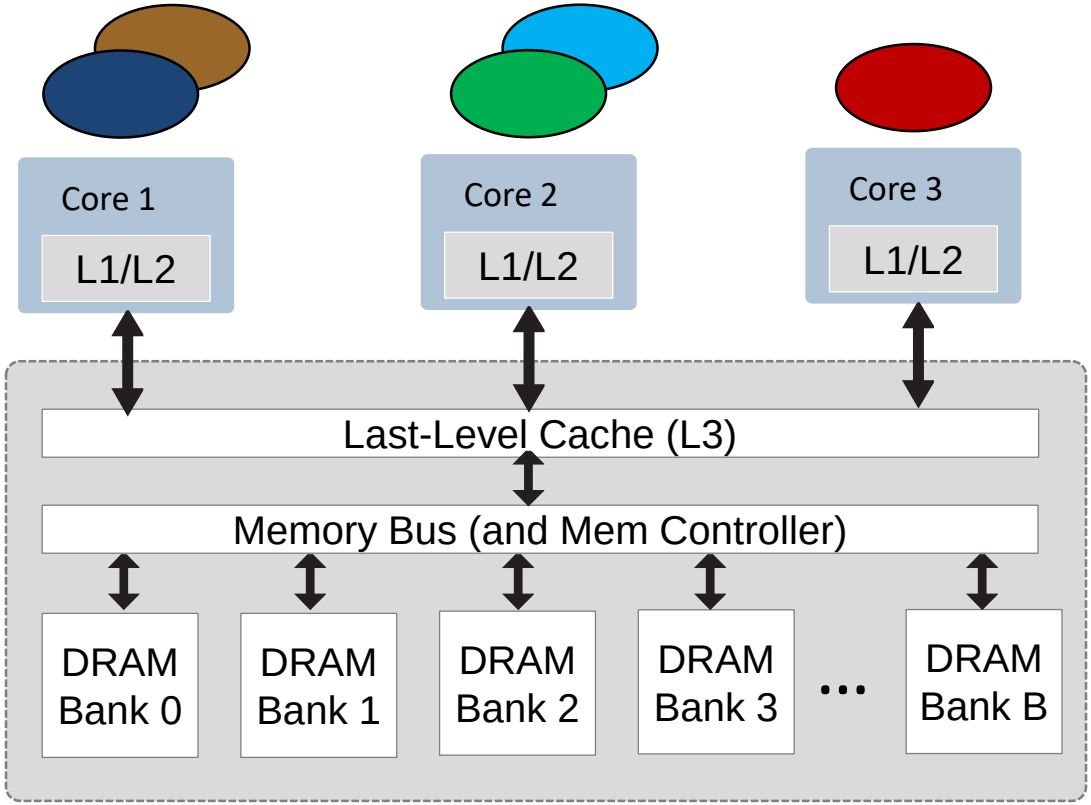
What are the requirements of a good abstraction?

1. It should be as small as possible (few numbers; few bits)
2. It should allow us to do prediction/ analysis that we care about
3. Given a system, it should be possible to find the abstraction (through measurements or lower-level analysis)



Considerations in creating an abstraction for multicore

Let *speed*(*i*,*t*) denote the speed of execution of task *i* at time *t*. Let cor(*i*,*t*) denote the set of tasks, other than task *i*, that executes at time *t*.

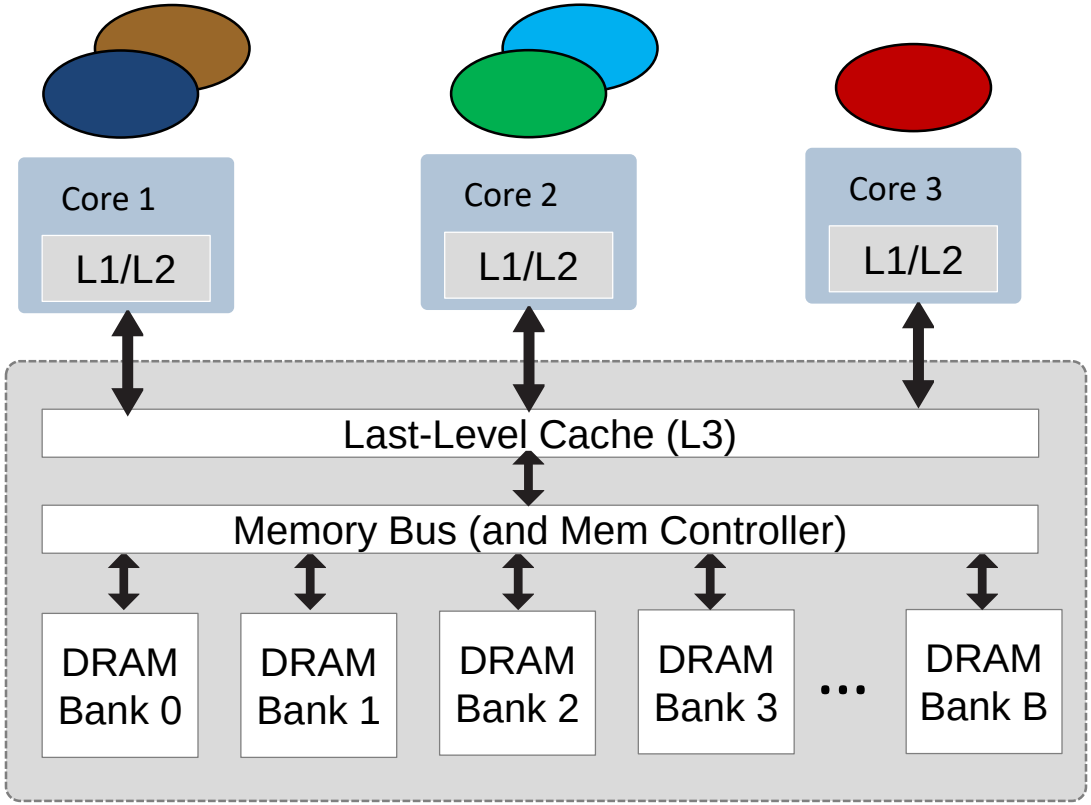


Considerations in creating an abstraction for multicore

Let *speed*(*i*,*t*) denote the speed of execution of task *i* at time *t*. Let cor(*i*,*t*) denote the set of tasks, other than task *i*, that executes at time *t*.

One abstraction could be:

$$\frac{1}{1+|cor(i,t)|} \leq speed(i,t) \leq 1$$



Considerations in creating an abstraction for multicore

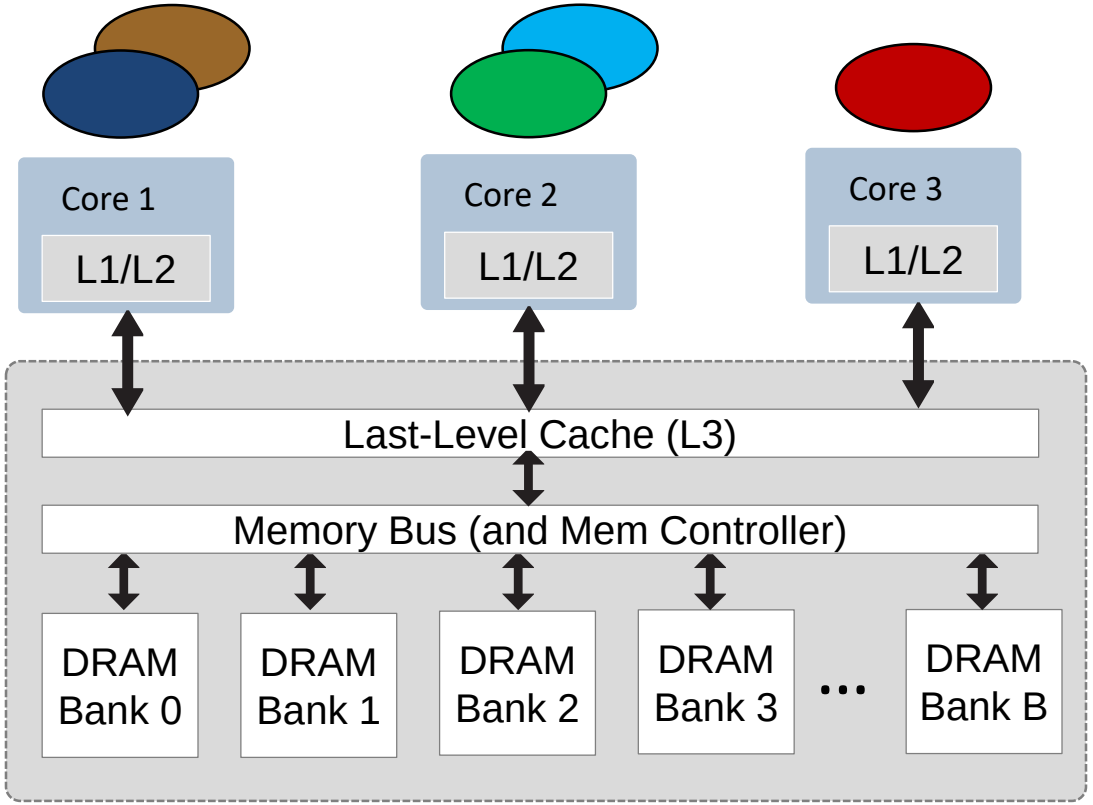
Let *speed*(*i*,*t*) denote the speed of execution of task *i* at time *t*. Let cor(*i*,*t*) denote the set of tasks, other than task *i*, that executes at time *t*.

One abstraction could be:

$$\frac{1}{1+|cor(i,t)|} \leq speed(i,t) \leq 1$$

Drawback of abstraction:

- Does not reflect that different co-runners can have different effect on task *i*.



Considerations in creating an abstraction for multicore

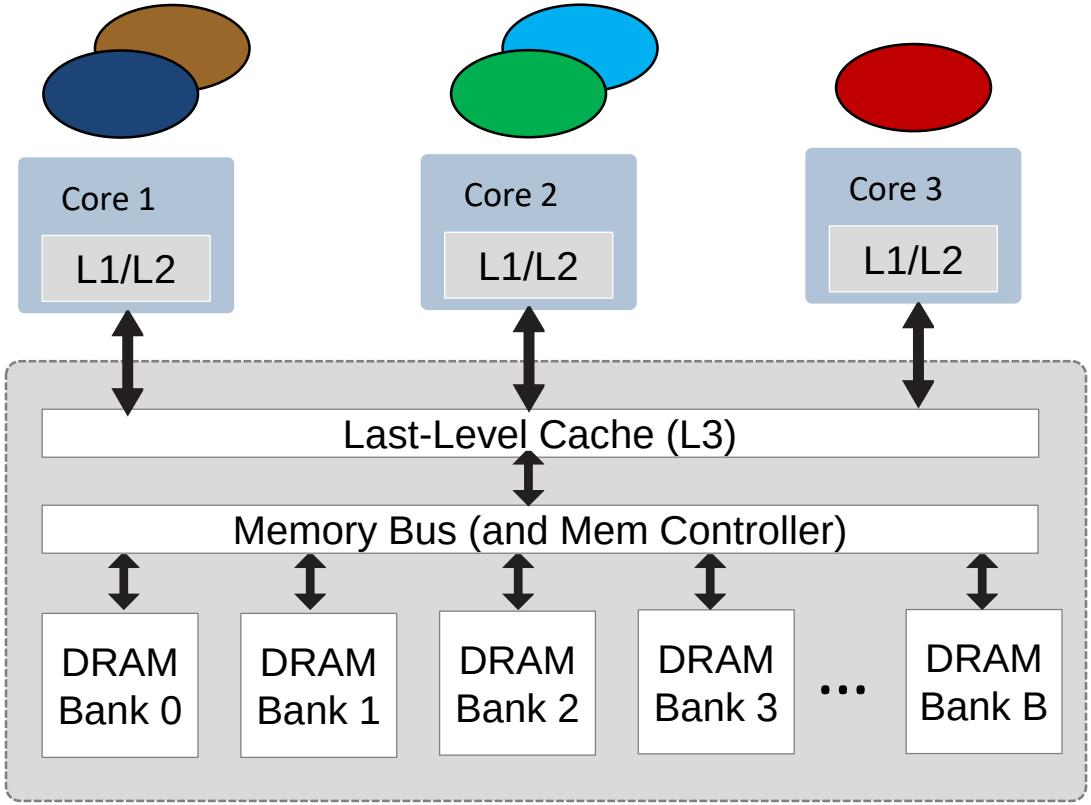
Let *speed*(*i*,*t*) denote the speed of execution of task *i* at time *t*. Let cor(*i*,*t*) denote the set of tasks, other than task *i*, that executes at time *t*.

One abstraction could be:

$$1 - k \times |cor(i,t)| \leq speed(i,t) \leq 1$$

Drawback of abstraction:

- Does not reflect that different co-runners can have different effect on task *i*.



Considerations in creating an abstraction for multicore

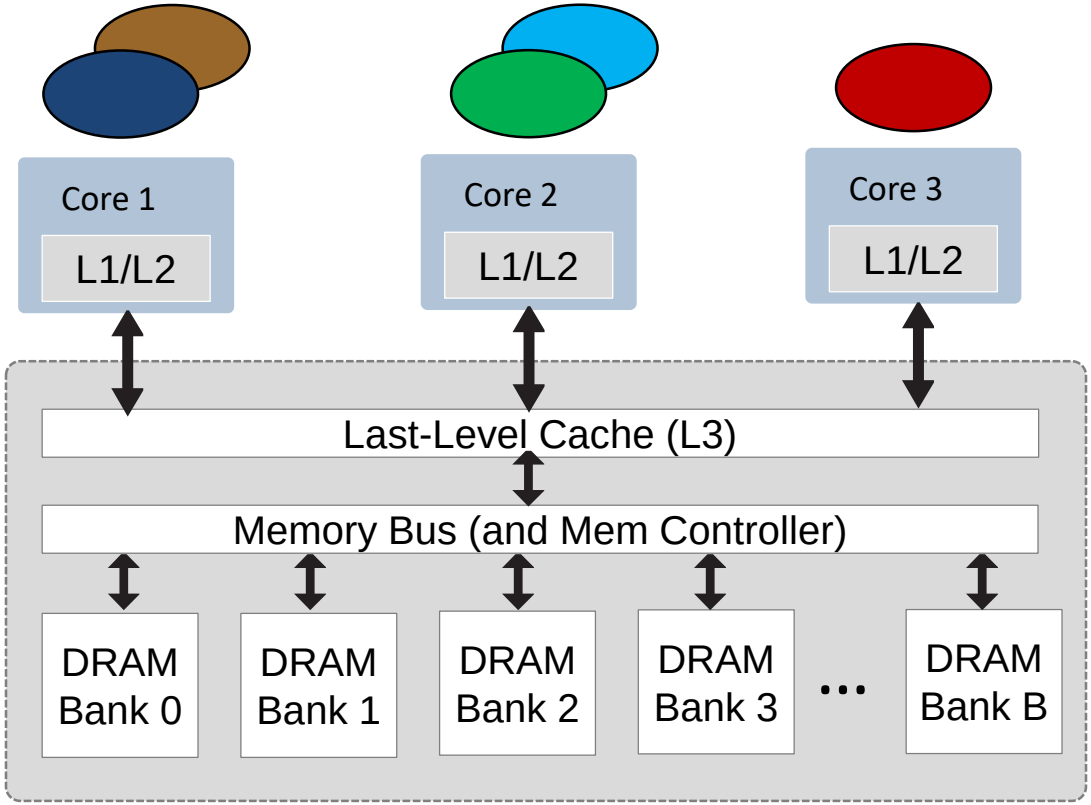
Let *speed*(*i*,*t*) denote the speed of execution of task *i* at time *t*. Let cor(*i*,*t*) denote the set of tasks, other than task *i*, that executes at time *t*.

One abstraction could be:

$$\frac{w_i}{1 + \sum_{j|j \in cor(i,t)} w_{i,j}} \leq speed(i,t) \leq 1$$

Drawback of abstraction:

- In reality, the speed can be super -additive or sub-additive (not reflected in the above).



Considerations in creating an abstraction for multicore

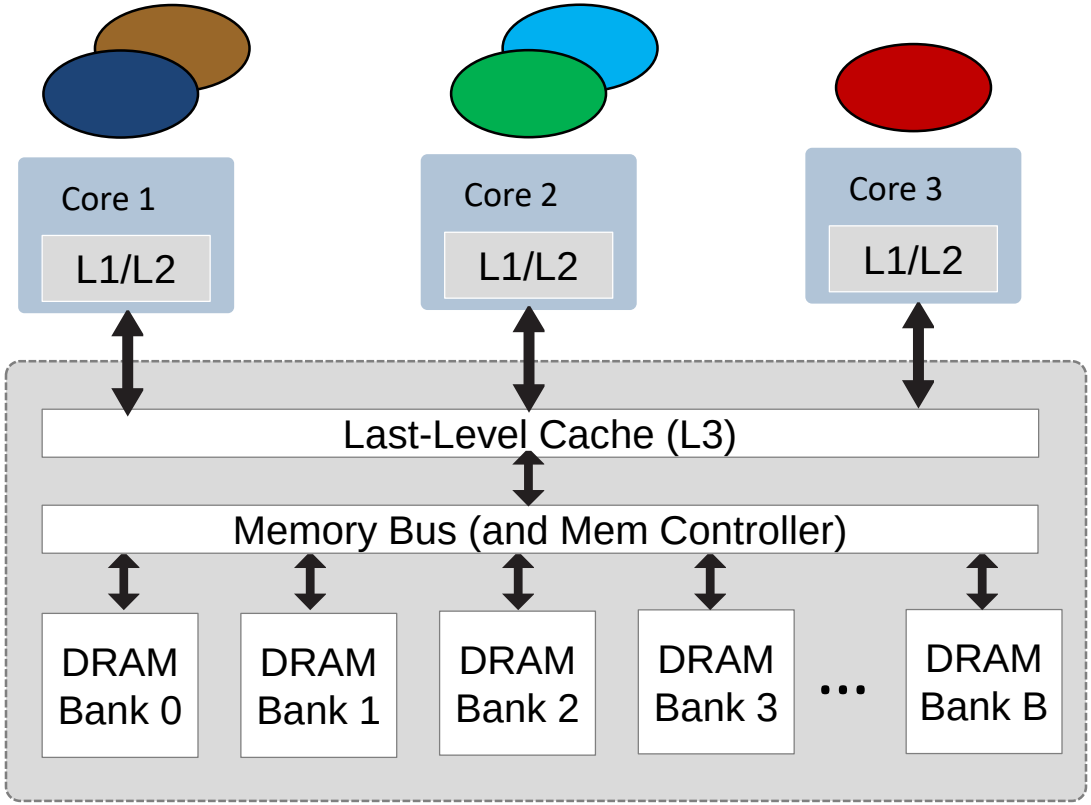
Let *speed*(*i*,*t*) denote the speed of execution of task *i* at time *t*. Let cor(*i*,*t*) denote the set of tasks, other than task *i*, that executes at time *t*.

One abstraction could be:

$$\frac{w_i}{1 + \prod_{j|j \in cor(i,t)} w_{i,j}} \leq speed(i,t) \leq 1$$

Drawback of abstraction:

- In reality, the speed can be super -additive or sub-additive (not reflected in the above).



Considerations in creating an abstraction for multicore

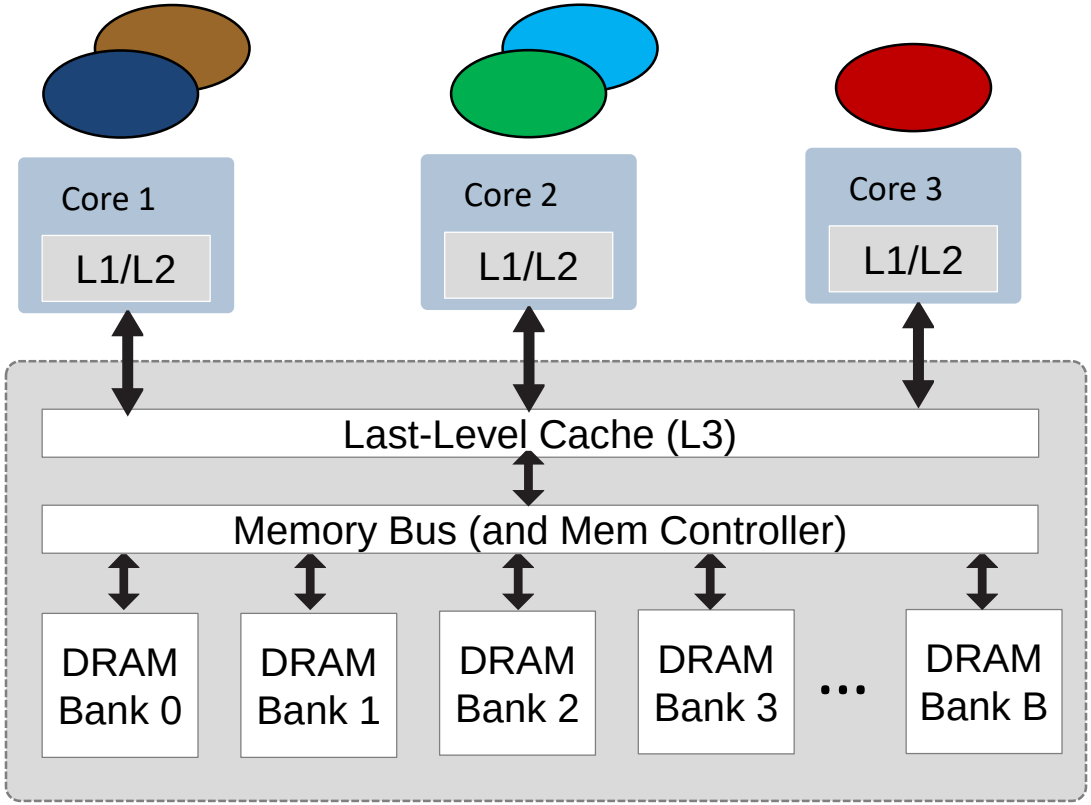
Let *speed*(*i*,*t*) denote the speed of execution of task *i* at time *t*. Let cor(*i*,*t*) denote the set of tasks, other than task *i*, that executes at time *t*.

One abstraction could be:

$$\frac{w_i}{1 + e^{\sum_{j|j \in cor(i,t)} w_{i,j}}} \leq speed(i,t) \leq 1$$

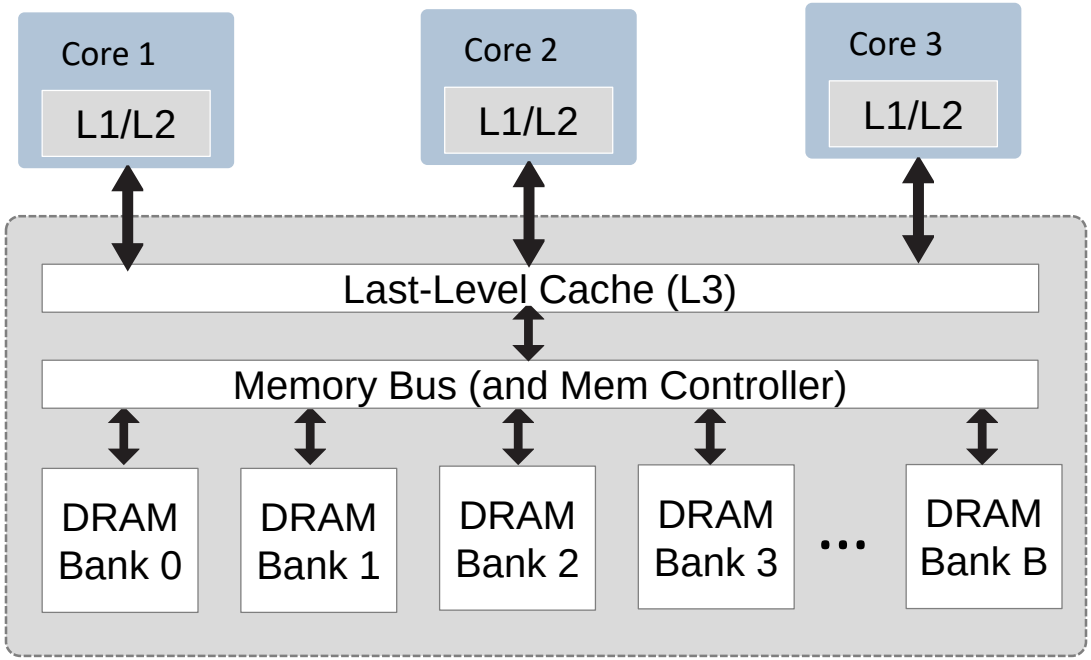
Drawback of abstraction:

- In reality, the speed can be super -additive or sub-additive (not reflected in the above).



Considerations in creating an abstraction for multicore

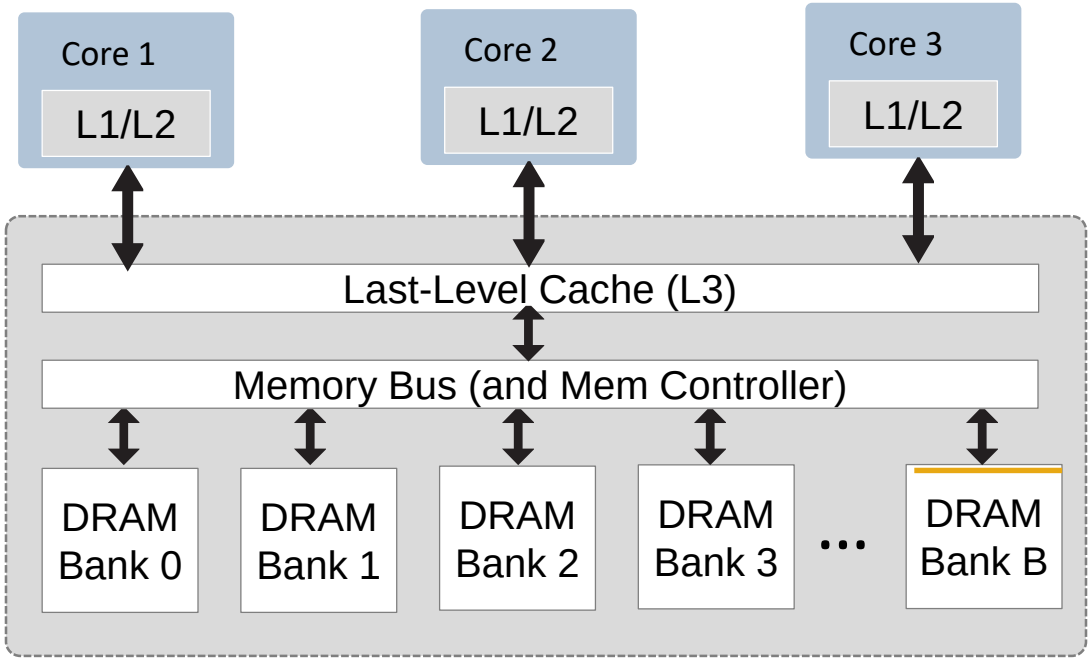
Why is speed sub-additive?



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

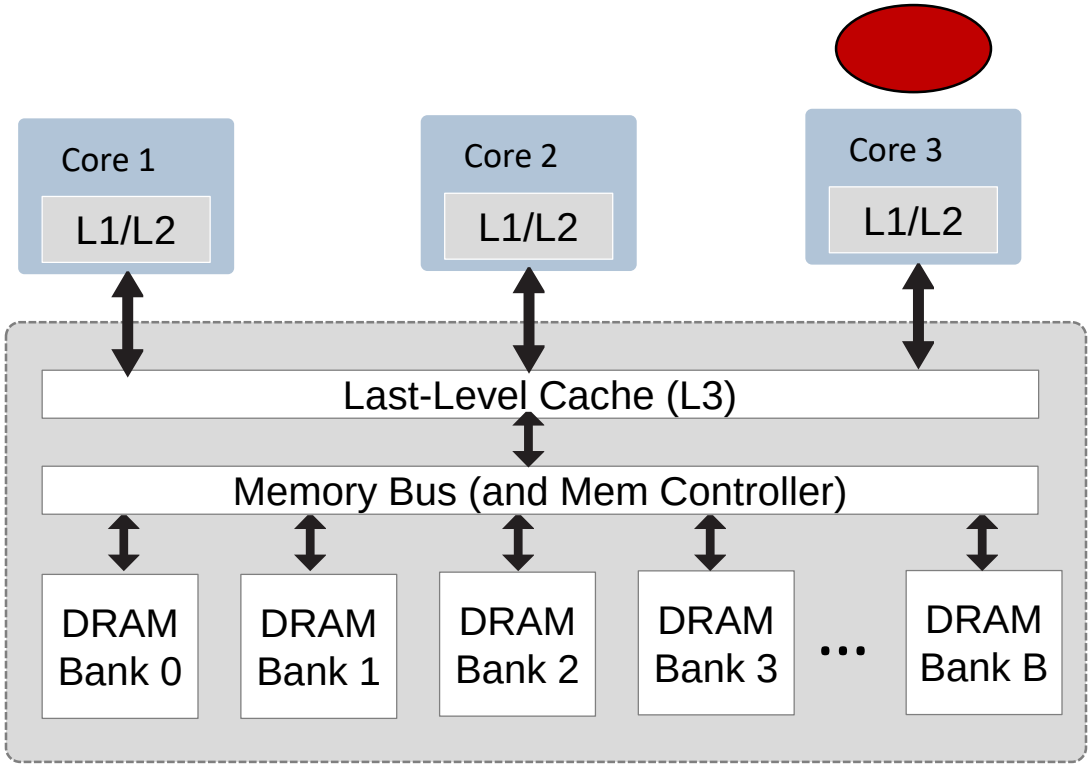
Consider DRAM bank B and its row buffer.



Considerations in creating an abstraction for multicore

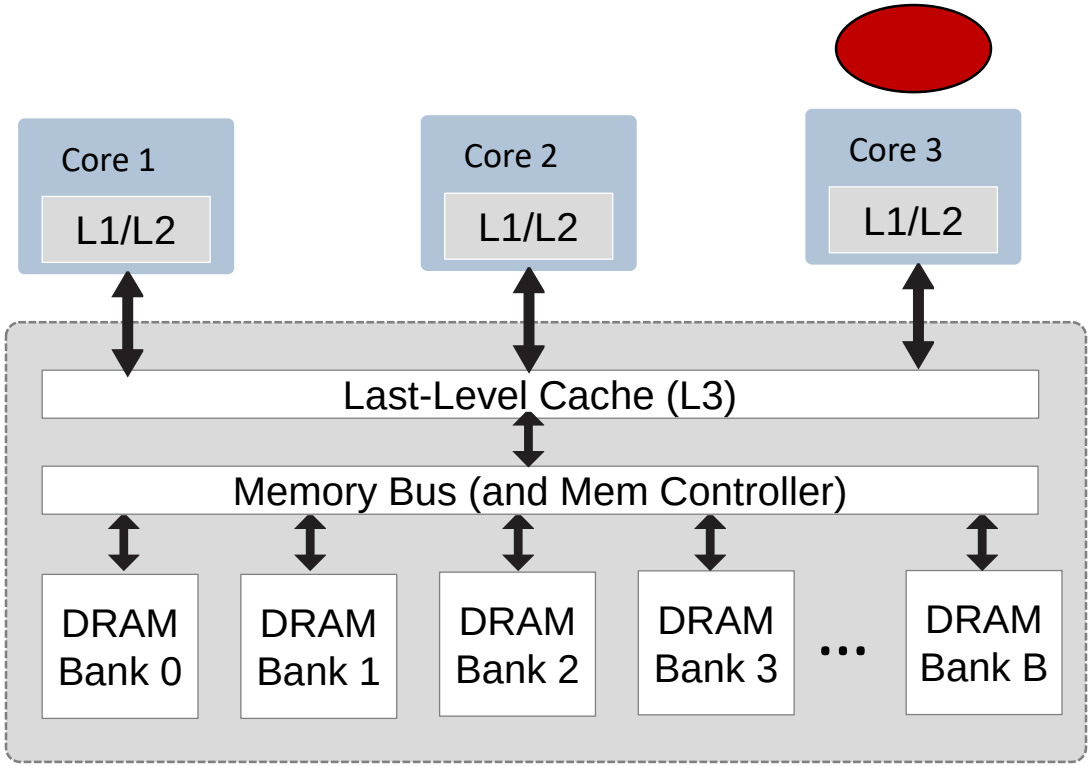
Why is speed sub-additive?
Example:
Red task (victim):
1. load address x to register y
2. load address x' to register z

x and x' are in the same row in DRAM bank B'
But x and x' are in different columns (different addresses)



Considerations in creating an abstraction for multicore

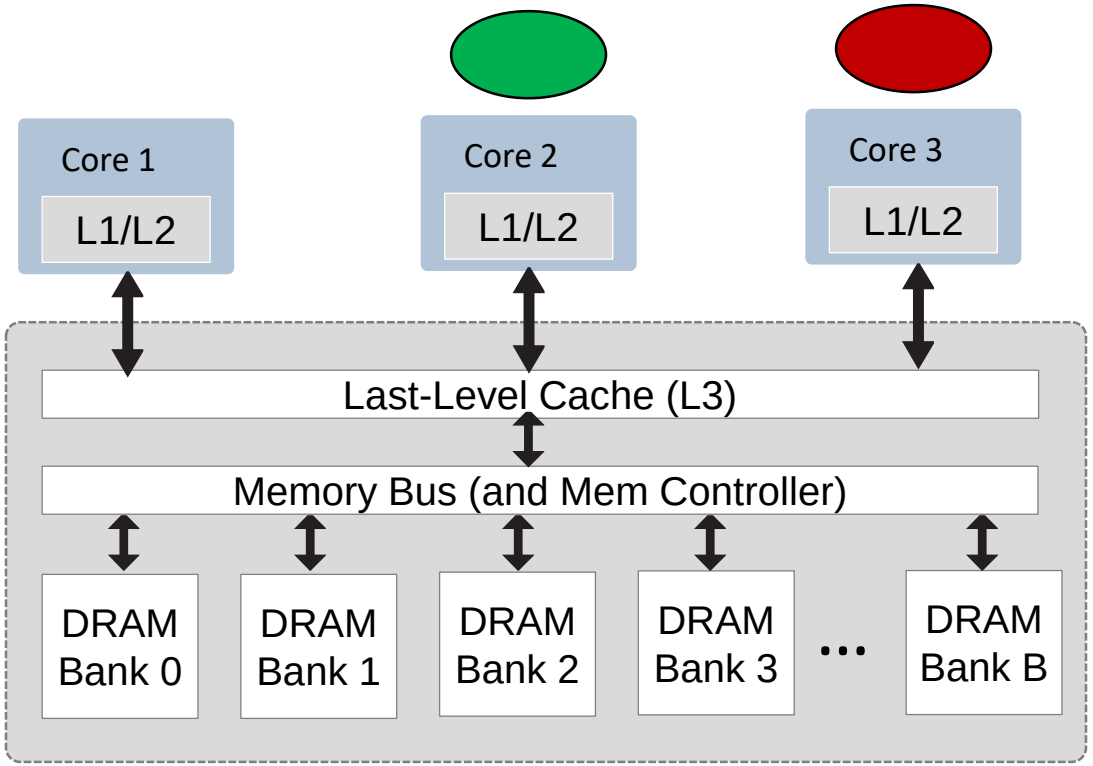
Why is speed sub-additive?
Example:
Red task:
1. load address x to register y
2. load address x' to register z



Considerations in creating an abstraction for multicore

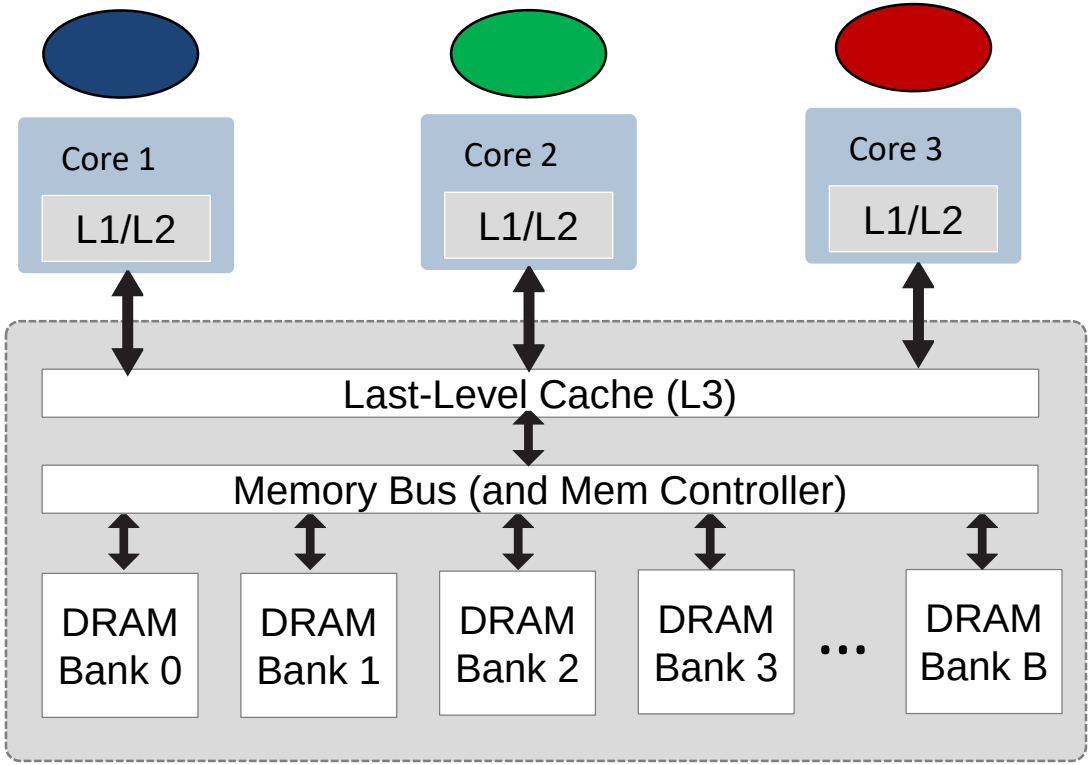
Why is speed sub-additive?
Example:
Red task:
1. load address x to register y
2. load address x' to register z

Green task:
1. load address a (where a is in same bank as x but in different row)



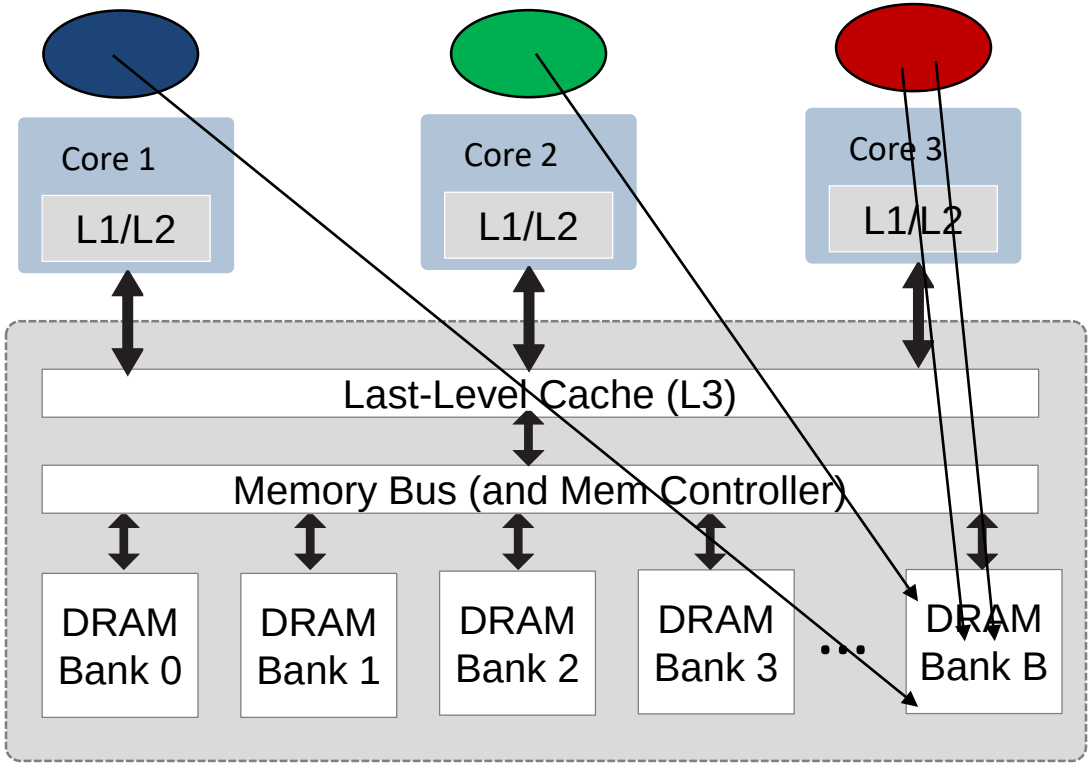
Considerations in creating an abstraction for multicore

Why is speed sub-additive?
Example:
- Red task:
  1. load address x to register y
  2. load address x' to register z
- Green task:
  1. load address a (where a is in same bank as x but in different row)
- Blue task:
  1. load address b (where b is in same bank as x but in different row)



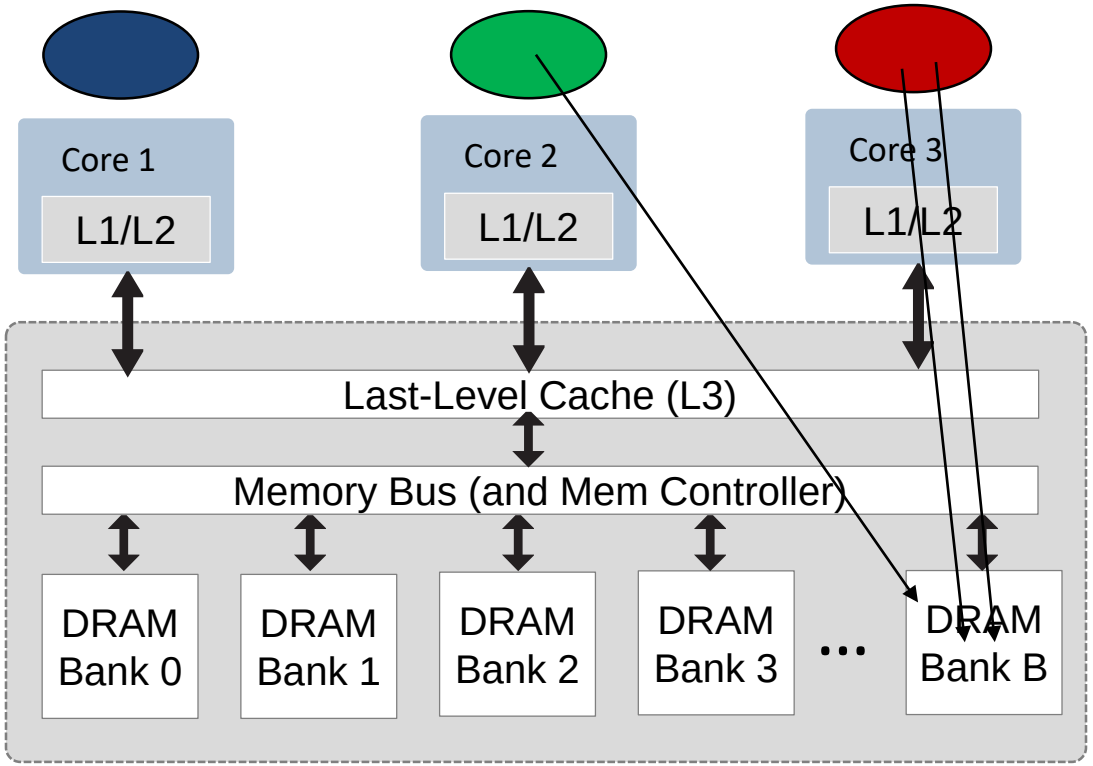
Considerations in creating an abstraction for multicore

Why is speed sub-additive?
Example:
- Red task:
  1. load address x to register y
  2. load address x' to register z
- Green task:
  1. load address a (where a is in same bank as x but in different row)
- Blue task:
  1. load address b (where b is in same bank as x but in different row)



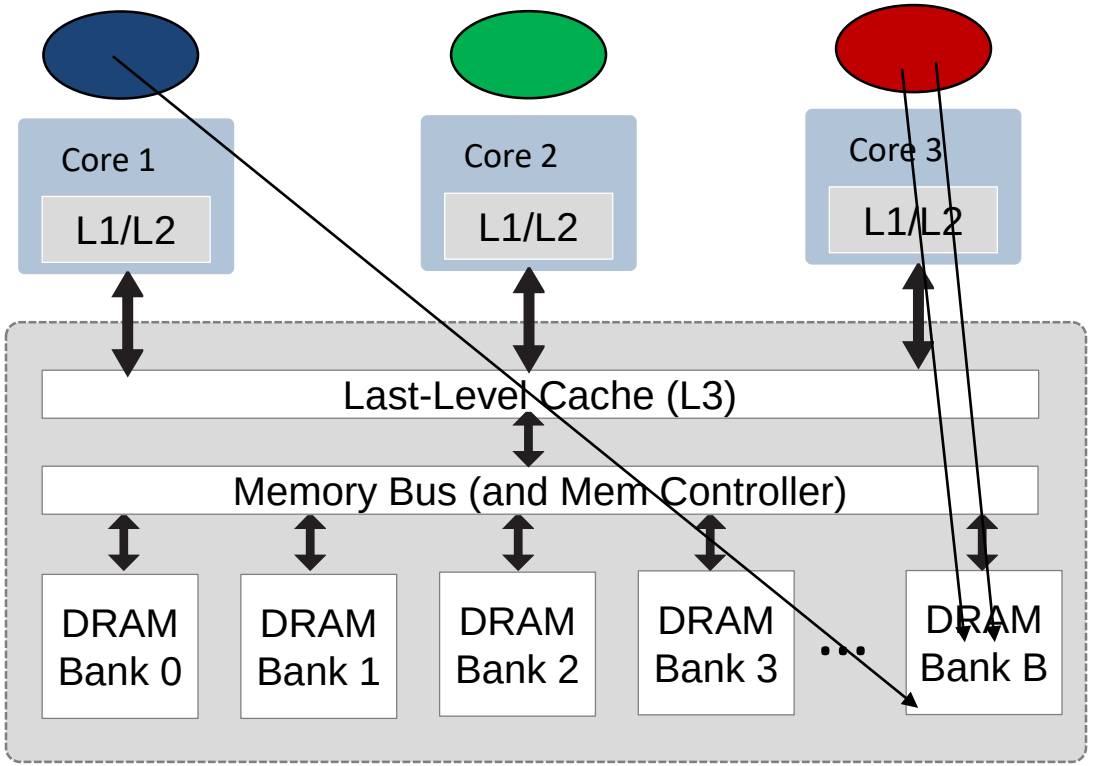
Considerations in creating an abstraction for multicore

Why is speed sub-additive?
Example:
- Red task:
  1. load address x to register y
  2. load address x' to register z
- Green task:
  1. load address a (where a is in same bank as x but in different row)
- Blue task:
  1. load address b (where b is in same bank as x but in different row)
- Observation: Green can evict Red's data in the row buffer.



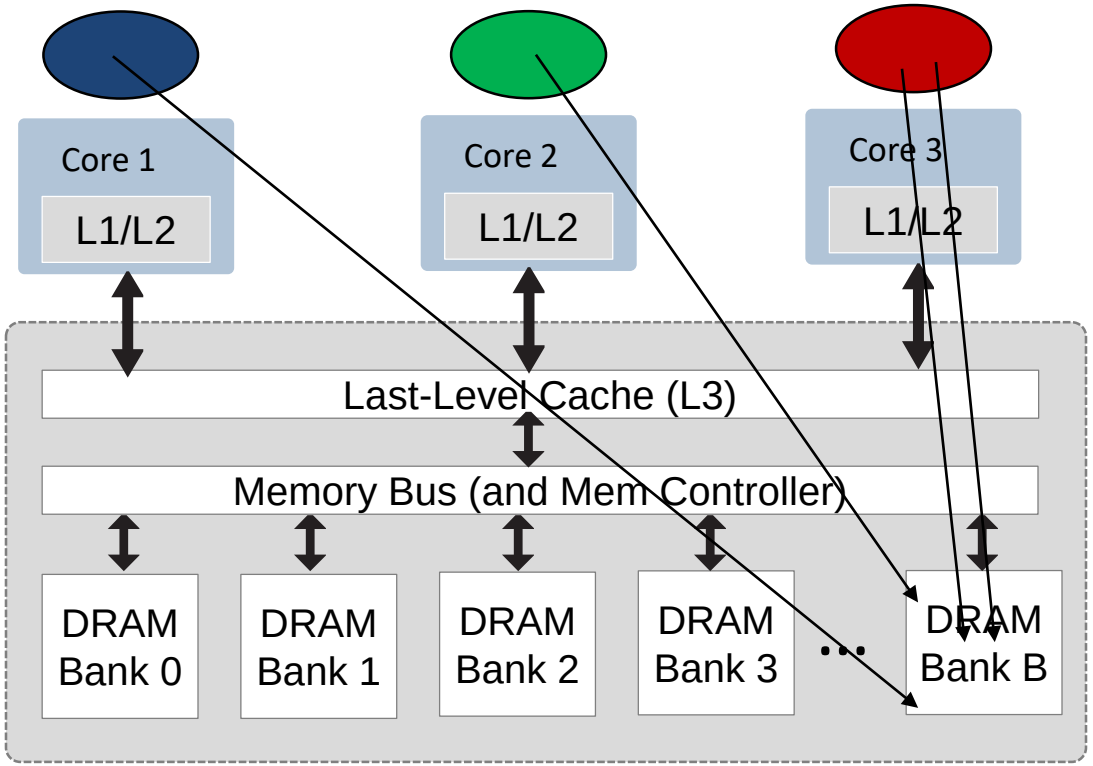
Considerations in creating an abstraction for multicore

Why is speed sub-additive?
Example:
- Red task:
  1. load address x to register y
  2. load address x' to register z
- Green task:
  1. load address a (where a is in same bank as x but in different row)
- Blue task:
  1. load address b (where b is in same bank as x but in different row)
- Observation: Blue can evict Red's data in the row buffer.



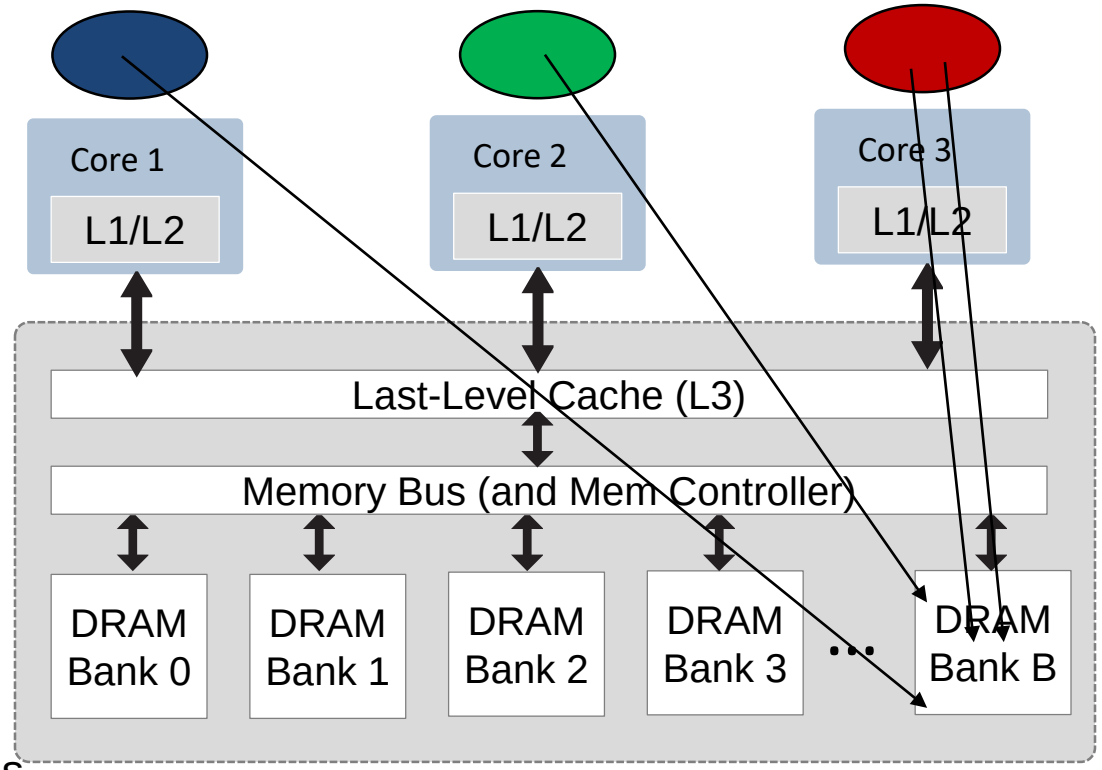
Considerations in creating an abstraction for multicore

Why is speed sub-additive?
Example:
- Red task:
  1. load address x to register y
  2. load address x' to register z
- Green task:
  1. load address a (where a is in same bank as x but in different row)
- Blue task:
  1. load address b (where b is in same bank as x but in different row)
- Observation: If Blue has evicted Red's data in the row buffer, then Green cannot evict it more.



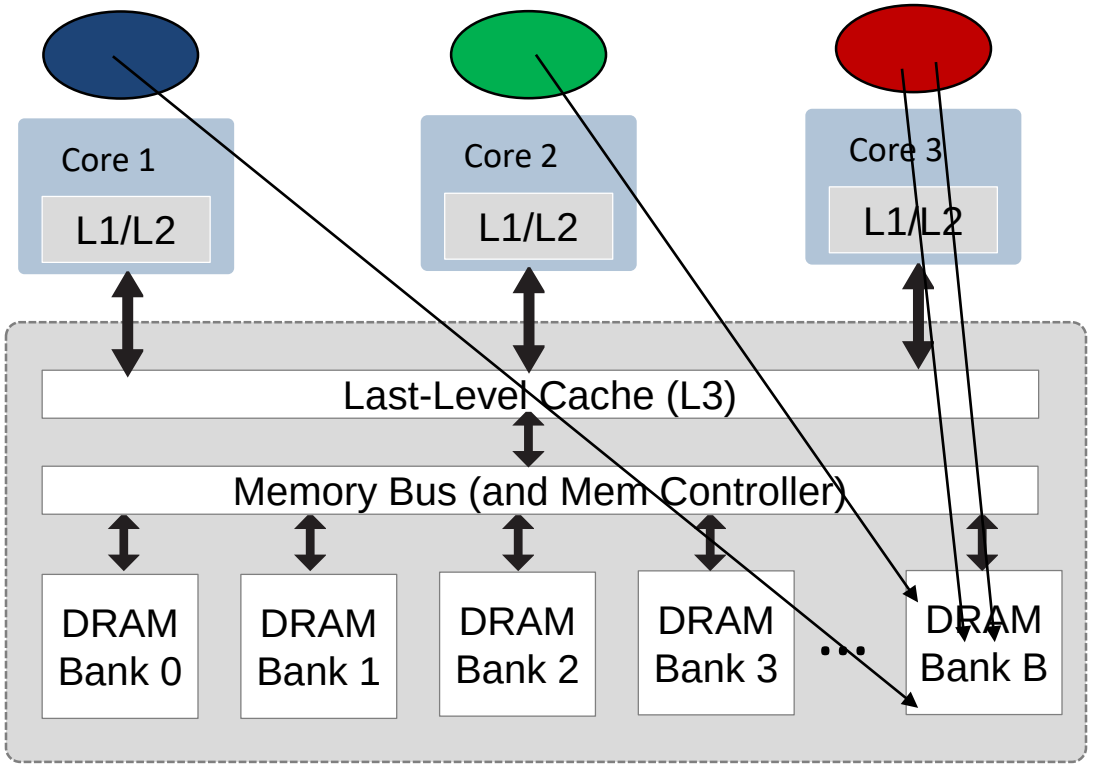
Considerations in creating an abstraction for multicore

Why is speed sub-additive?
Example:
- Red task:
  1. load address x to register y
  2. load address x' to register z
- Green task:
  1. load address a (where a is in same bank as x but in different row)
- Blue task:
  1. load address b (where b is in same bank as x but in different row)
- Observation: If Green has evicted Red's data in the row buffer, then Blue cannot evict it more.



Considerations in creating an abstraction for multicore

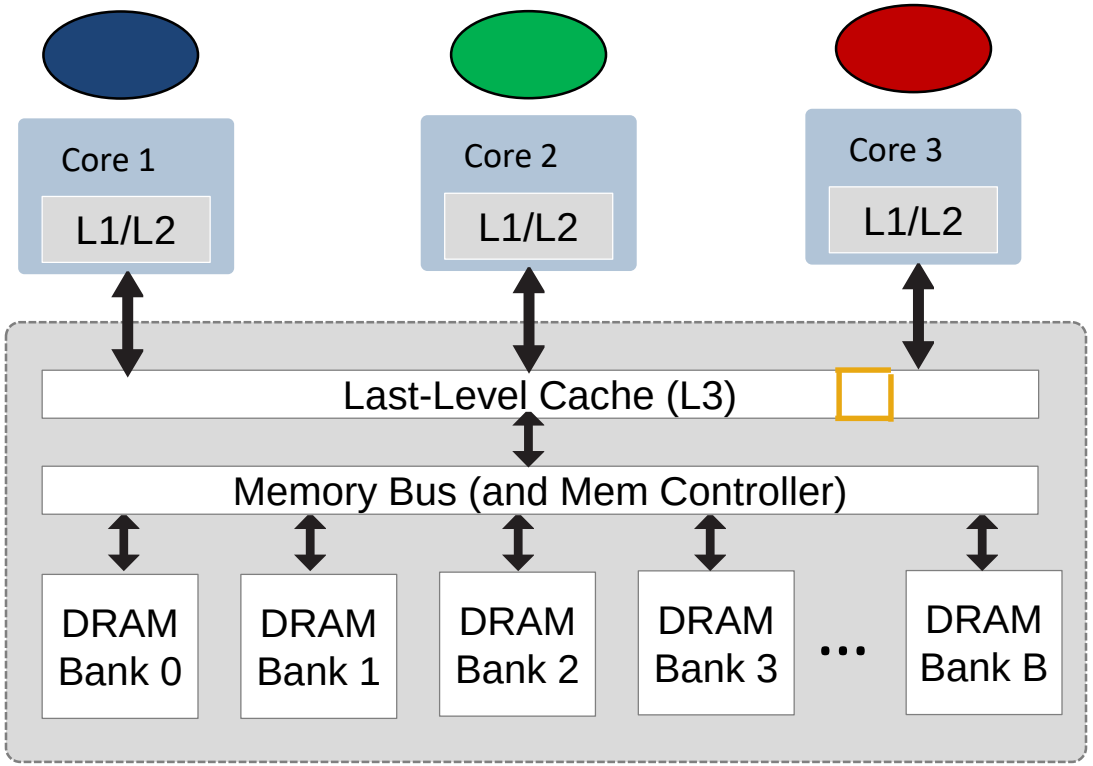
Why is speed sub-additive?
Example:
- Red task:
  1. load address x to register y
  2. load address x' to register z
- Green task:
  1. load address a (where a is in same bank as x but in different row)
- Blue task:
  1. load address b (where b is in same bank as x but in different row)
- Red experiences a slowdown due to Blue or Green but the slowdown is not greater by both.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Consider Last-Level Cache (L3).
Assume associativity = 2.
Consider one specific cache set.



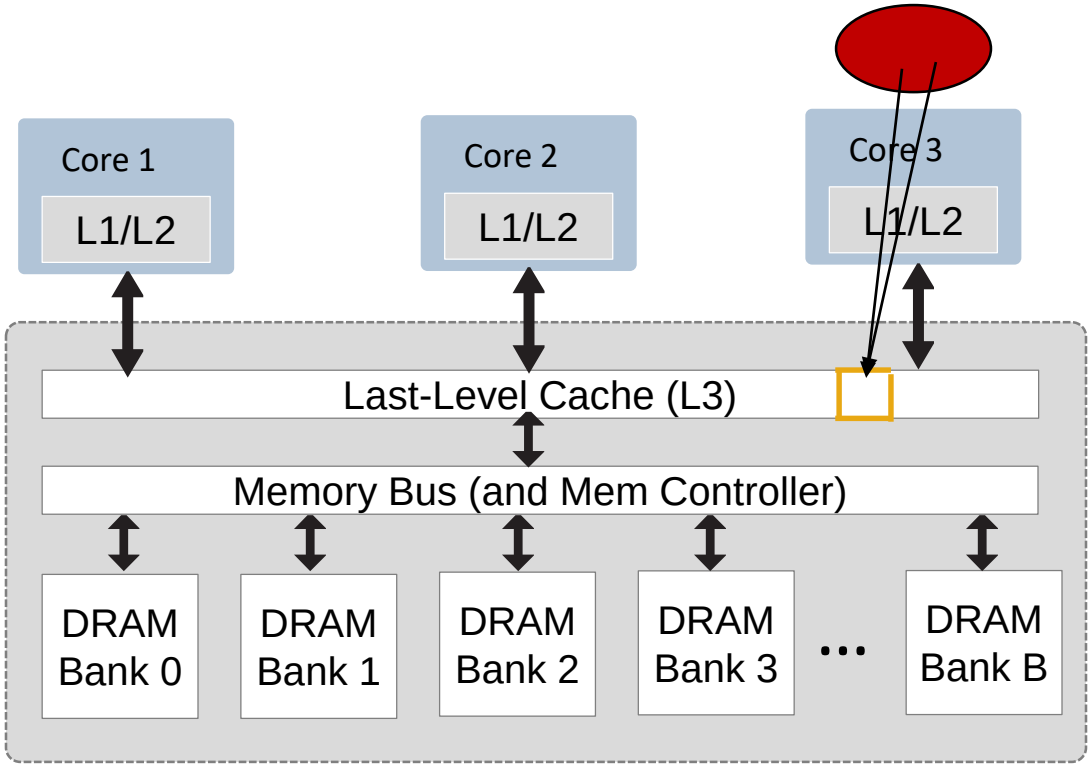
Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z



Considerations in creating an abstraction for multicore

Why is speed super-additive?

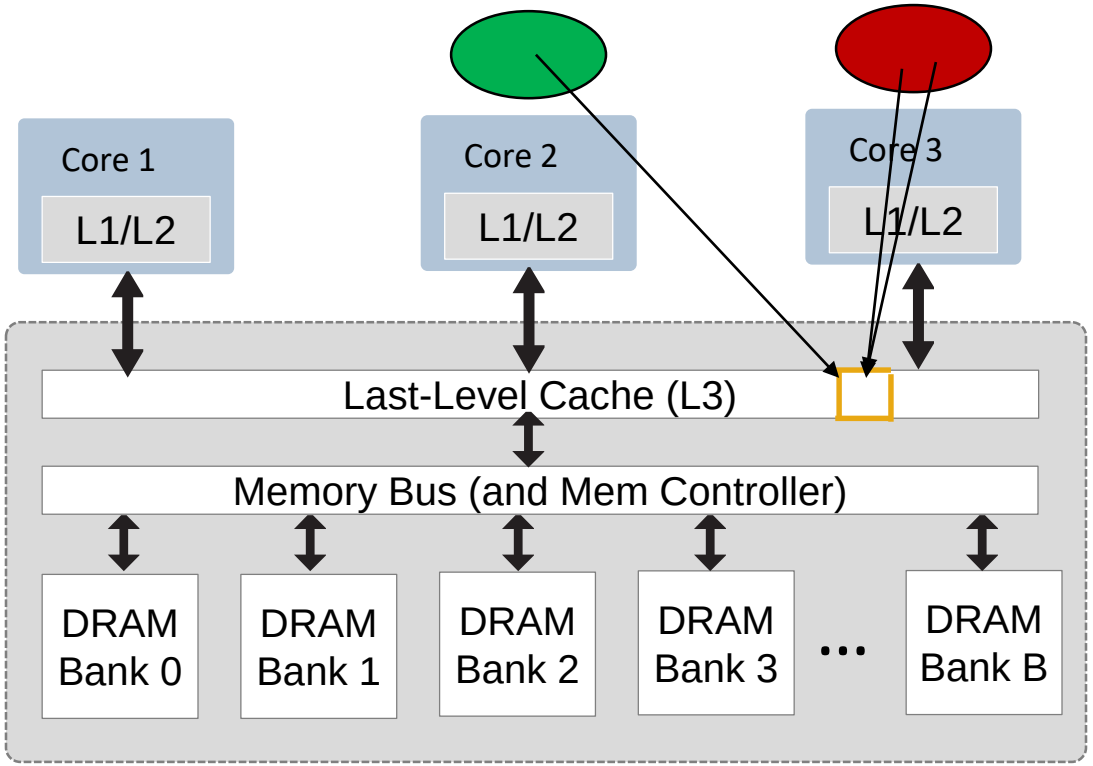
Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

Green task:

1. load address a (where a is in same cache set x)



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

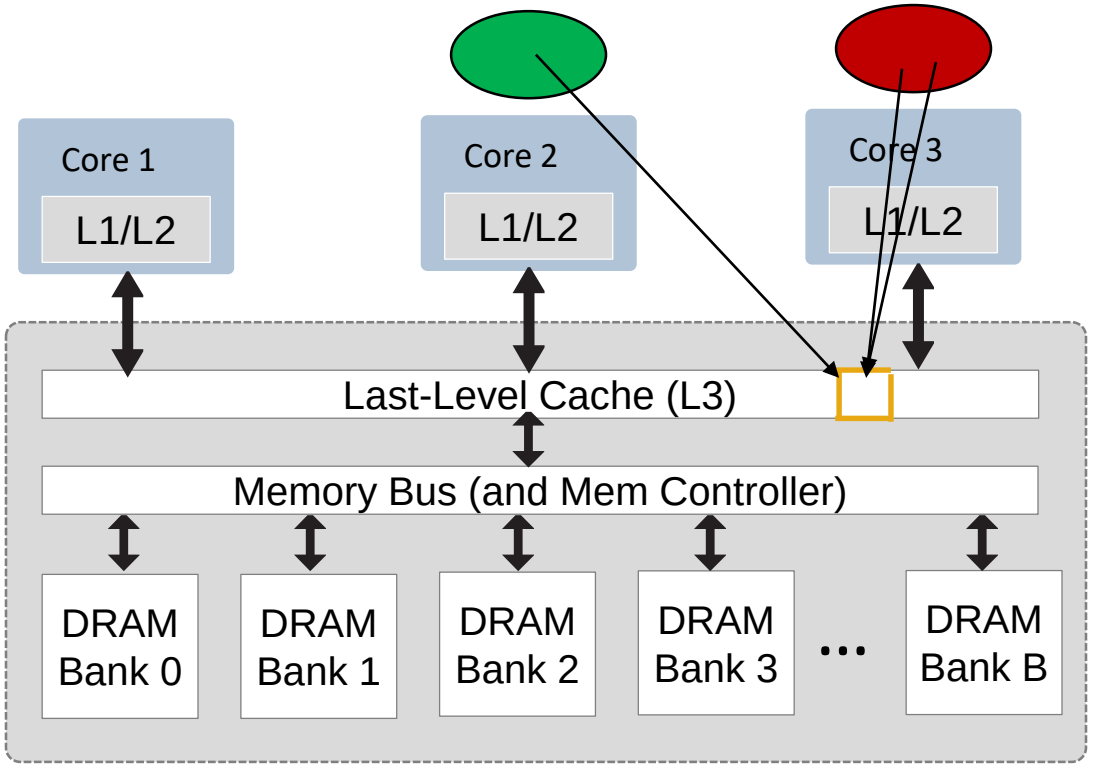
Red task (victim):

1. load address x to register y
2. load address x to register z

Green task:

1. load address a (where a is in same cache set x)

Observation: Green's data and Red's data fit in the cache. No capacity miss.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

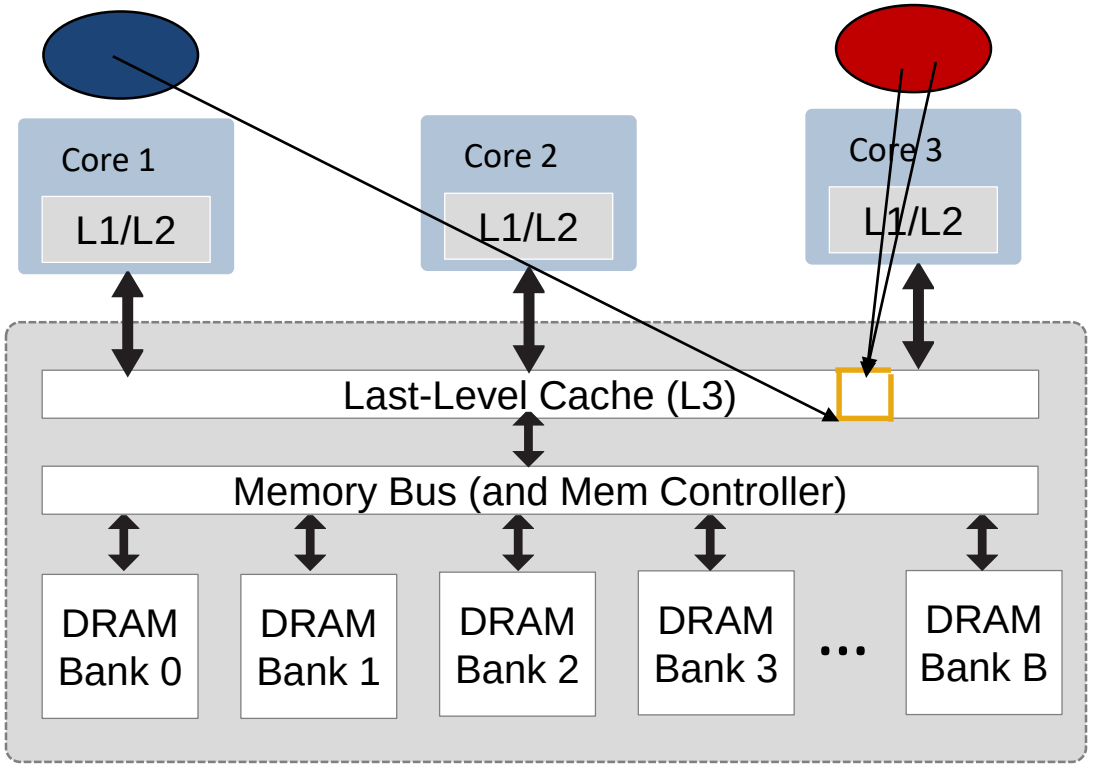
Red task (victim):

1. load address x to register y
2. load address x to register z

Blue task:

1. load address b (where b is in same cache set x)

Observation: Blue's data and Red's data fit in the cache. No capacity miss.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

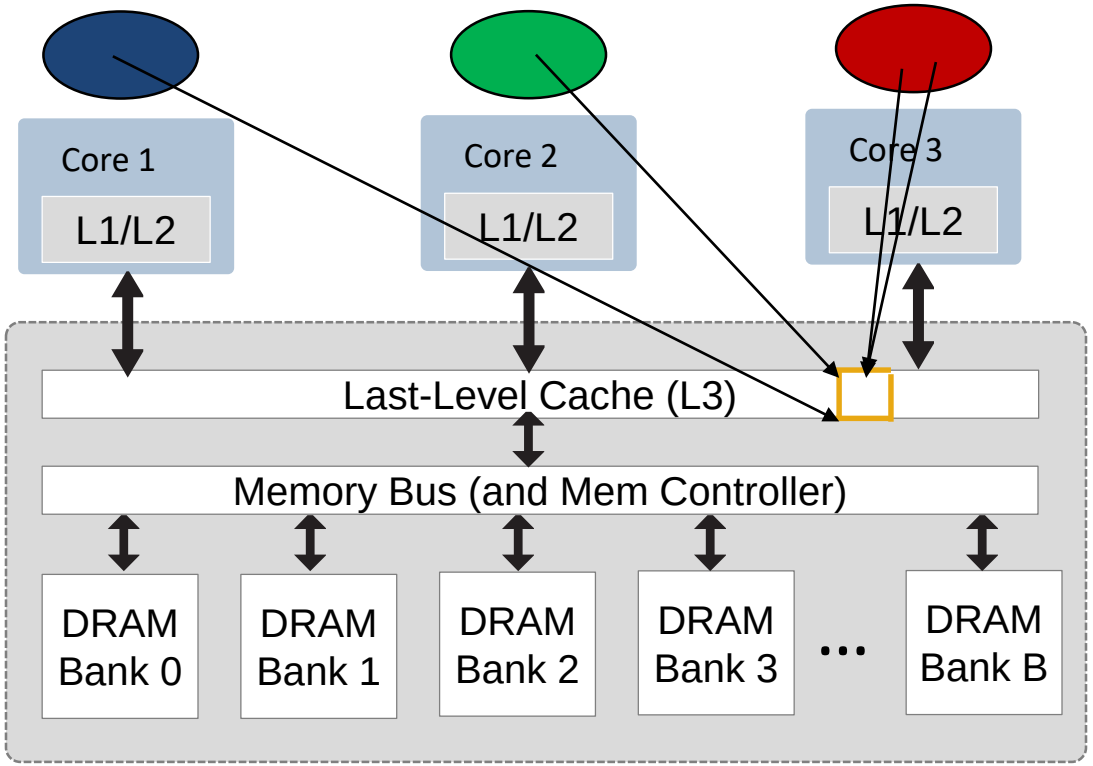
Green task:

1. load address a (where a is in same cache set x)

Blue task:

1. load address b (where b is in same cache set x)

Observation: Blue's data, Green's data, and Red's data do not fit in the cache. Capacity miss.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

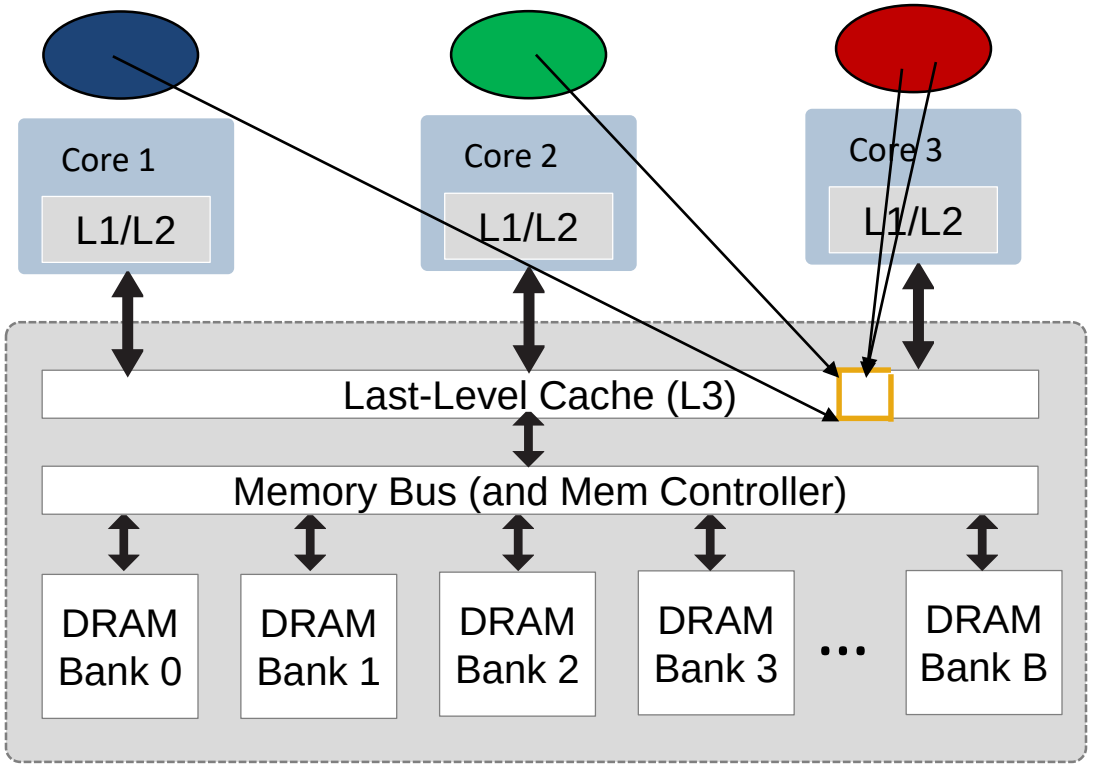
Green task:

1. load address a (where a is in same cache set x)

Blue task:

1. load address b (where b is in same cache set x)

Observation: Green alone does not cause slowdown of Red.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

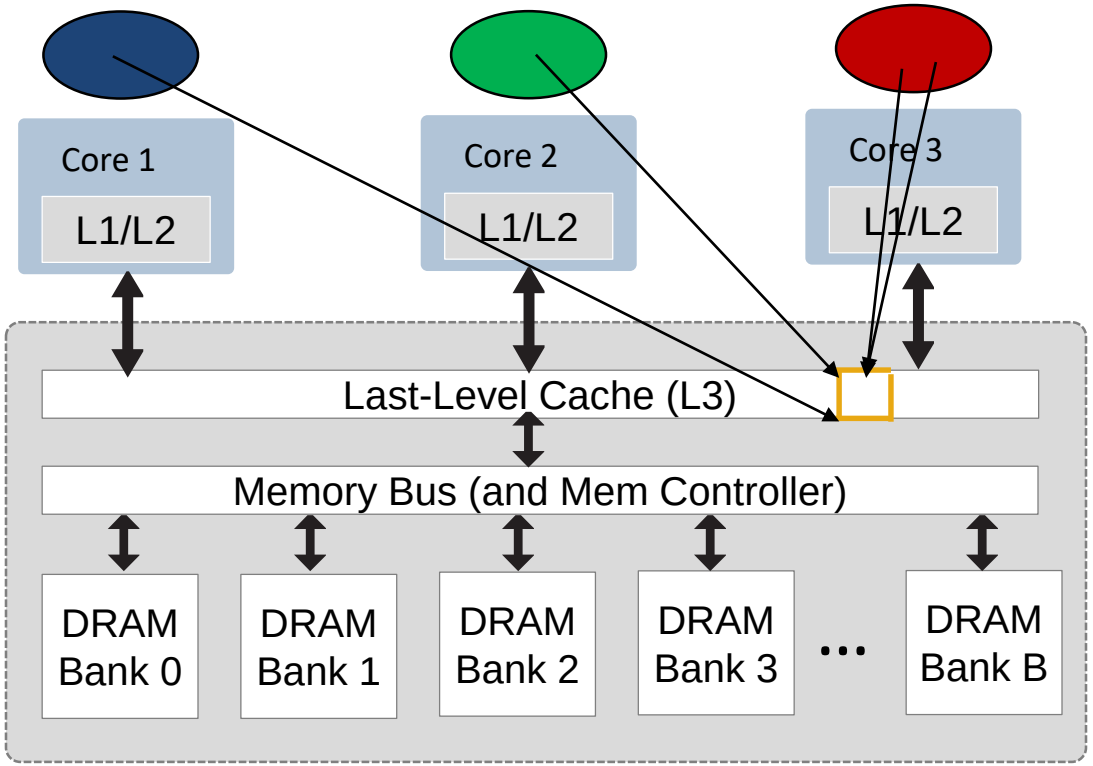
Green task:

1. load address a (where a is in same cache set x)

Blue task:

1. load address b (where b is in same cache set x)

Observation: Blue alone does not cause slowdown of Red.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

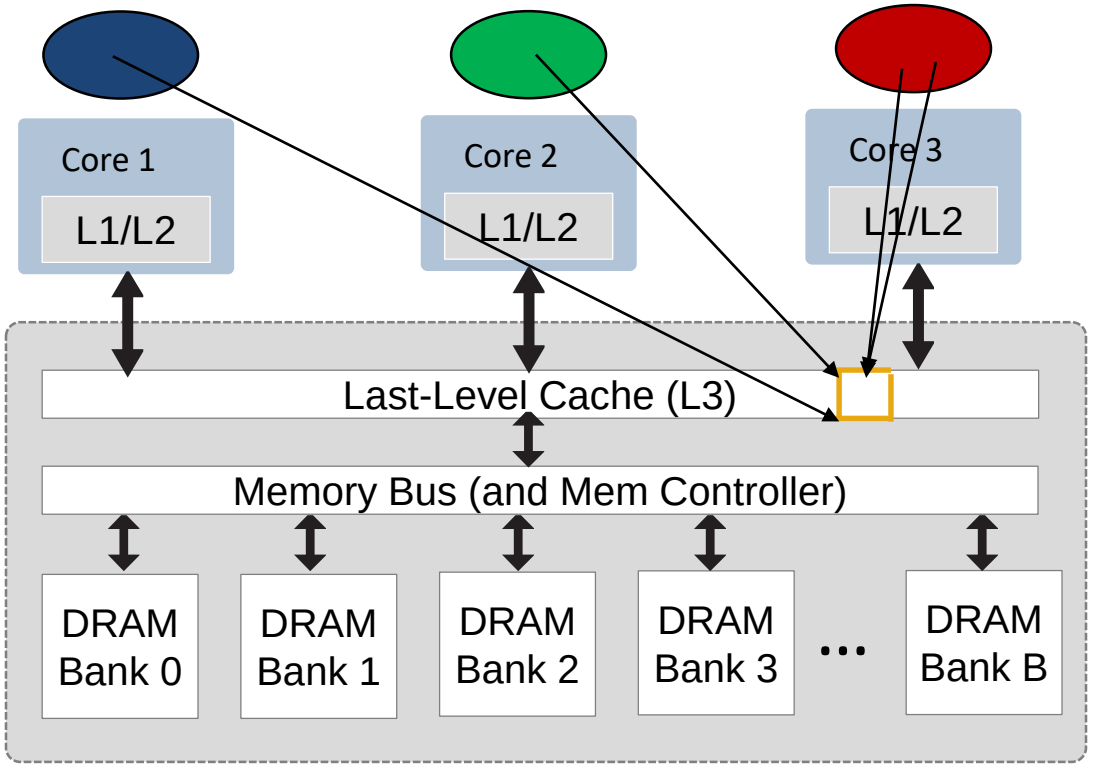
Green task:

1. load address a (where a is in same cache set x)

Blue task:

1. load address b (where b is in same cache set x)

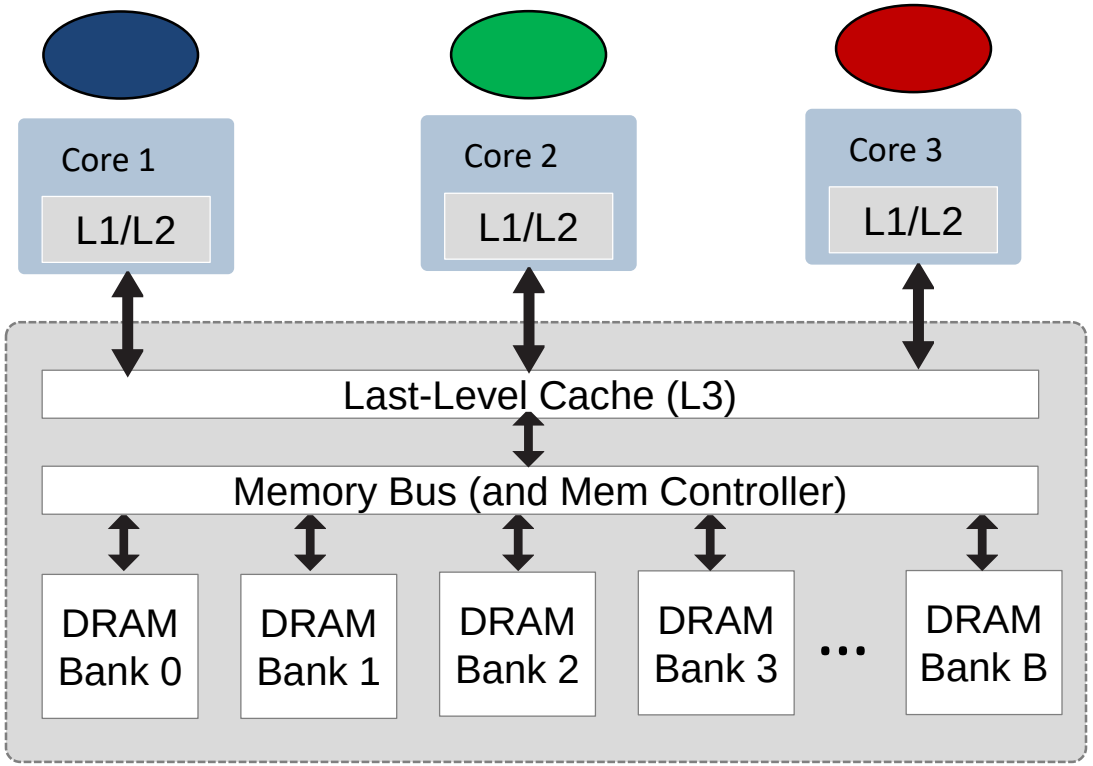
Observation: But Green and Blue cause slowdown of Red.



Considerations in creating an abstraction for multicore

Speed of execution of a task as a function of co-runners can be either

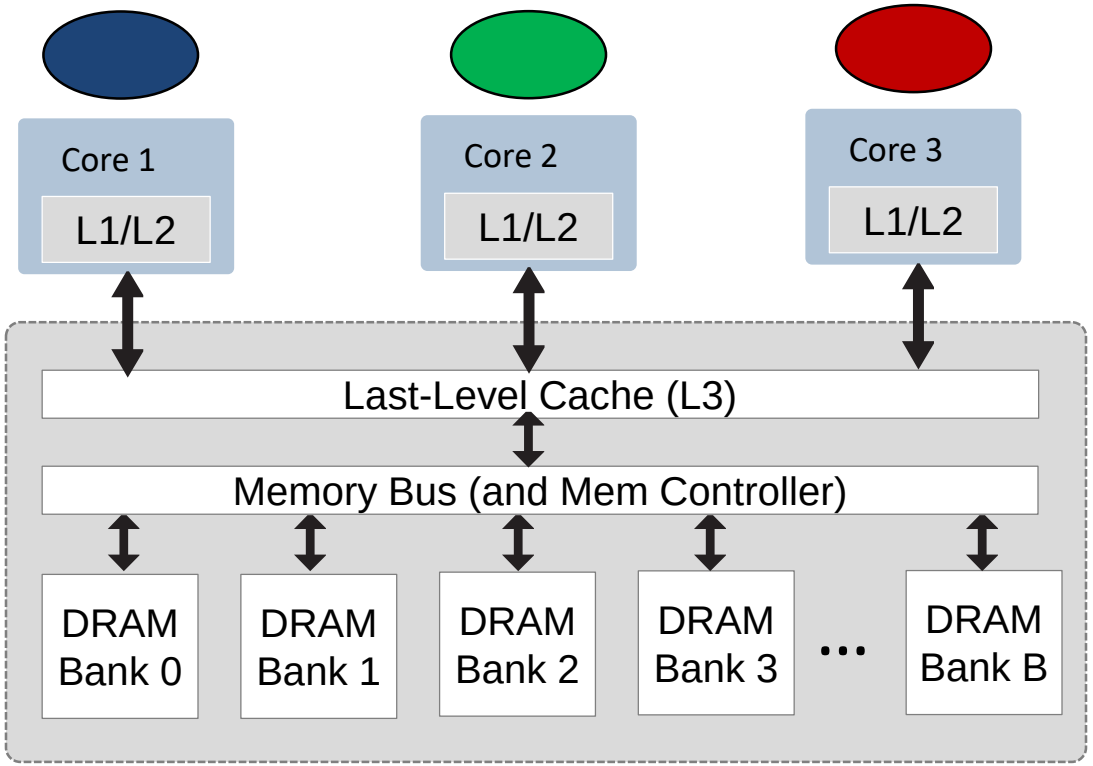
(i) Additive
(ii) Sub-additive
(iii) Super-additive



Considerations in creating an abstraction for multicore

Speed of execution of a task as a function of co-runners can be either

(i) Additive
(ii) Sub-additive
(iii) Super-additive

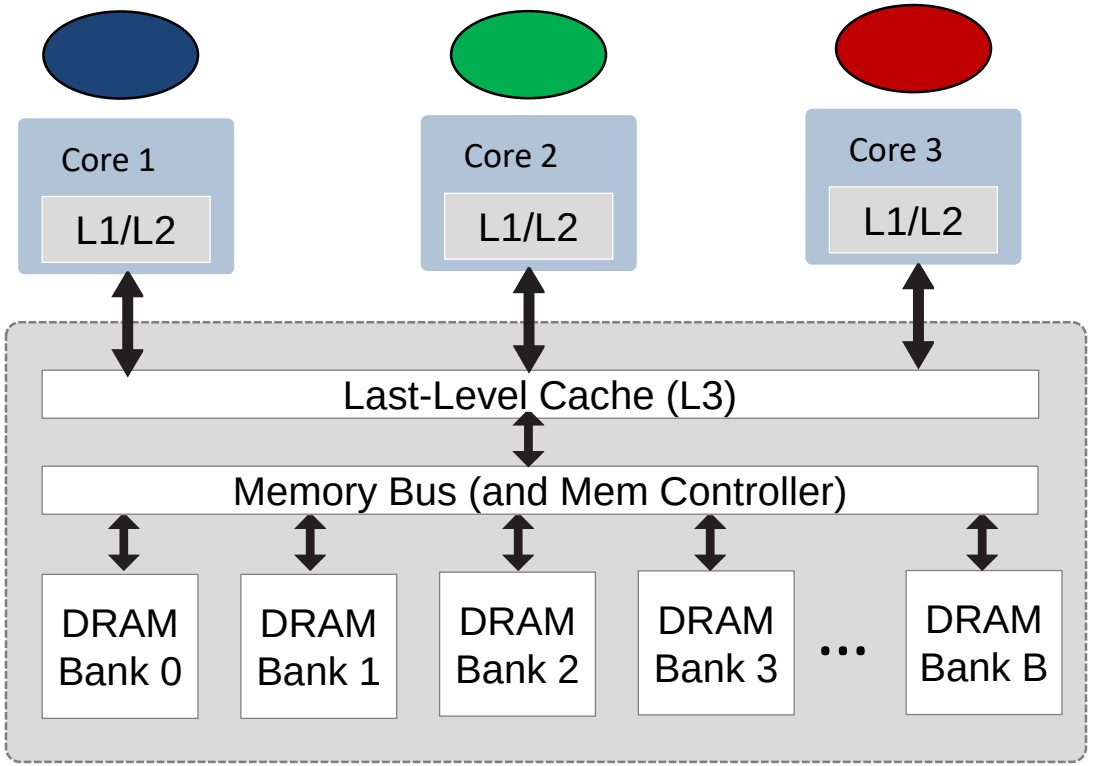


If speed is not additive, how can we describe speed as a function of co-runners?

Considerations in creating an abstraction for multicore

Speed of execution of a task as a function of co-runners can be either

(i) Additive
(ii) Sub-additive
(iii) Super-additive



For each task: Enumerate the set of possible co-runner set.

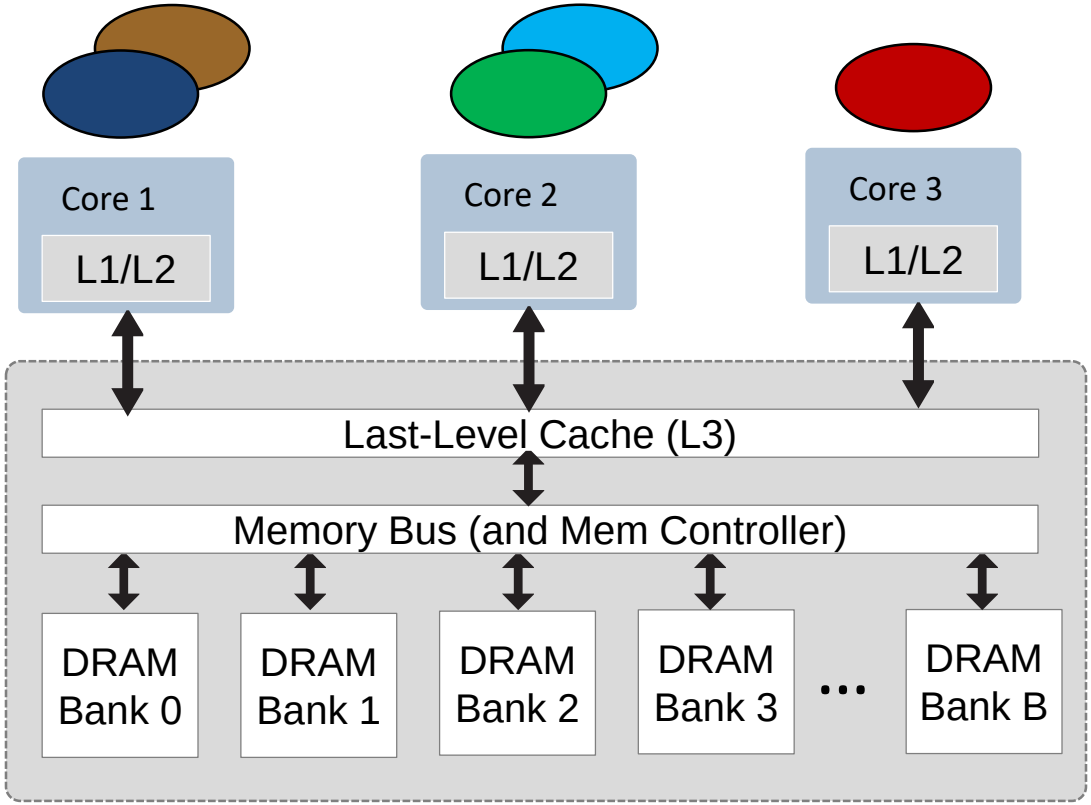
Considerations in creating an abstraction for multicore

Main idea:

For each task *i*,

for each co-runner set of task *i* do

$$lowerboundspeed_i^{cor} \leq speed(i,t) \leq 1$$



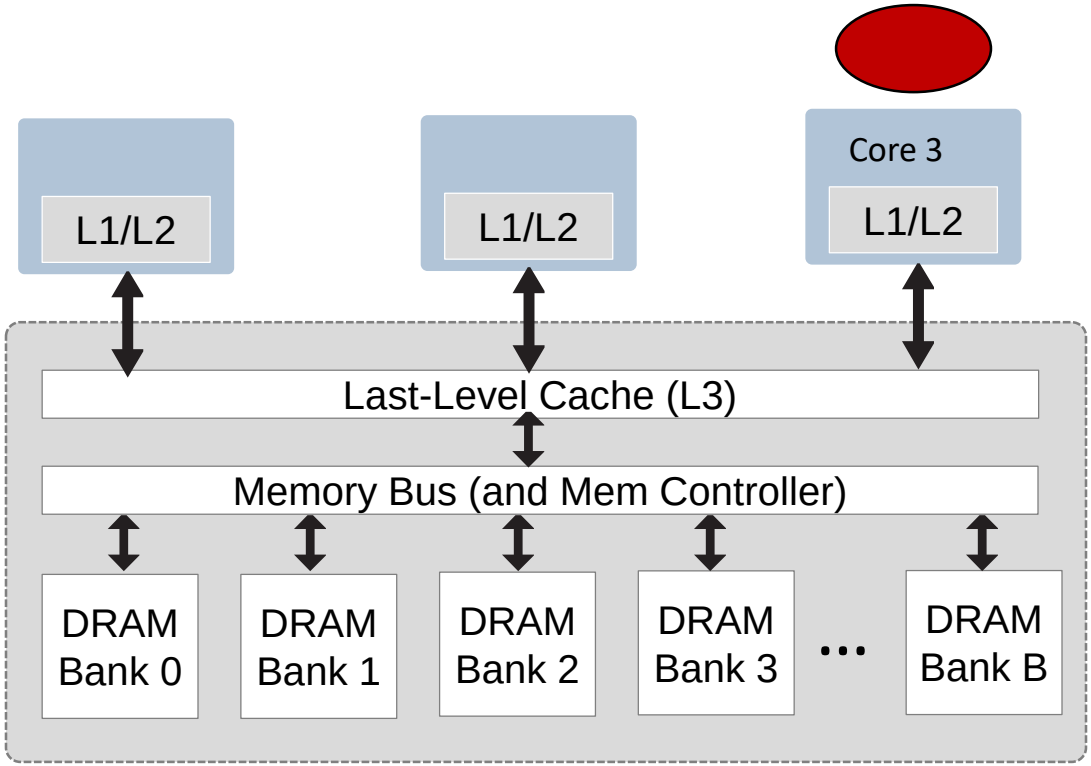
For each task: Enumerate the set of possible co-runner set.

Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1

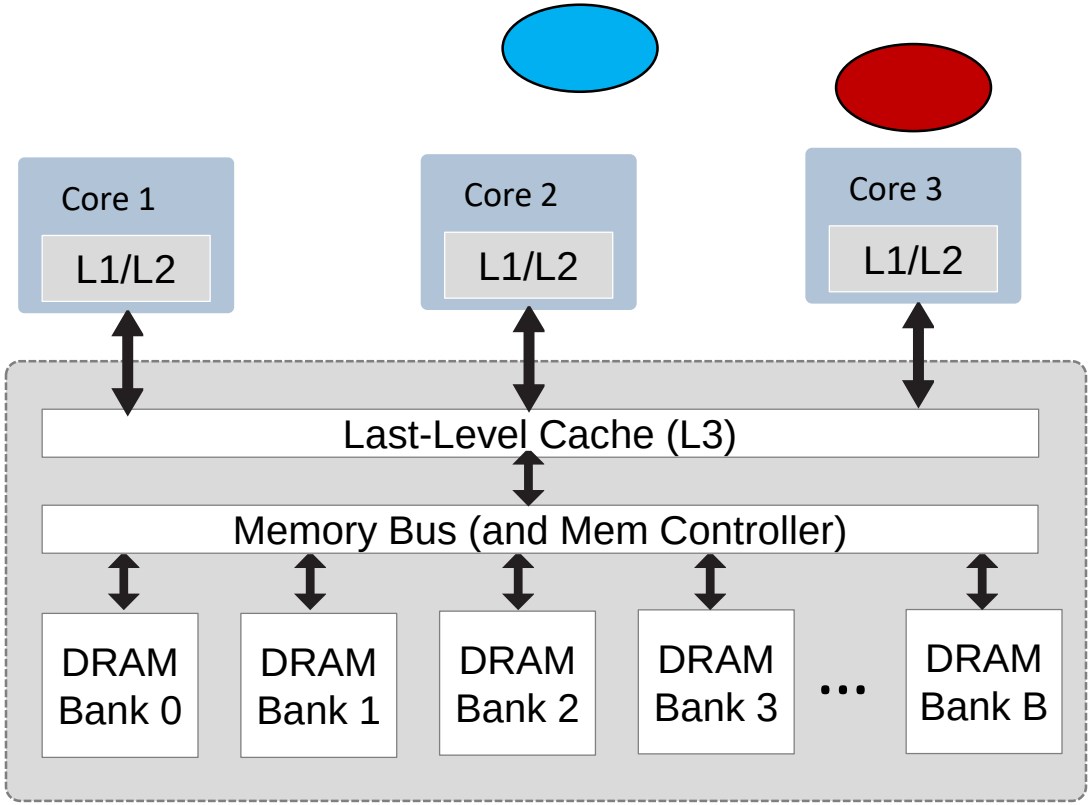


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{cyan}	0.5

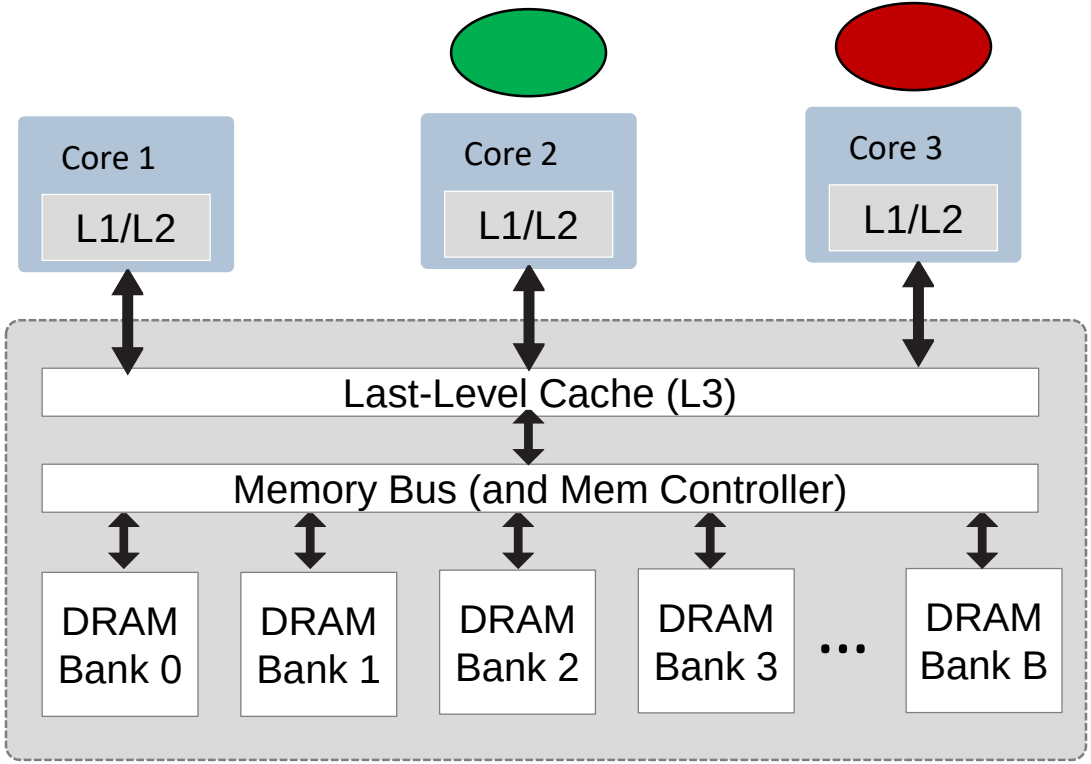


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{green}	0.45

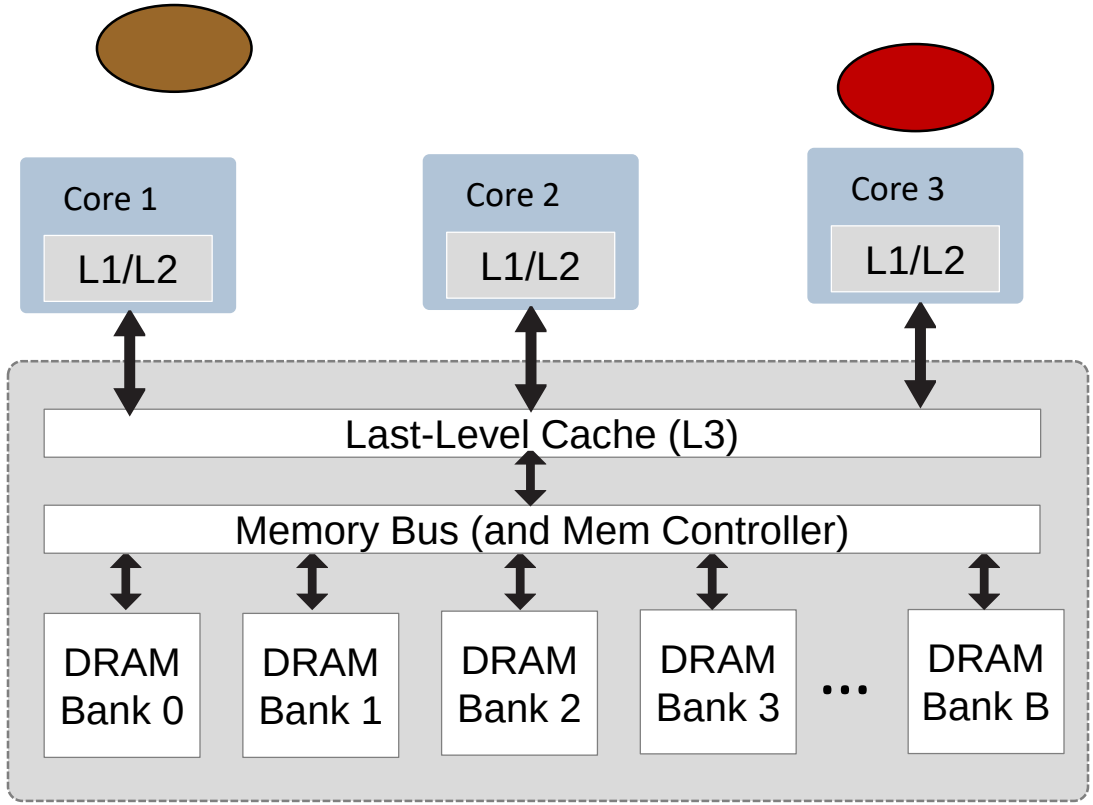


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{brown}	0.45

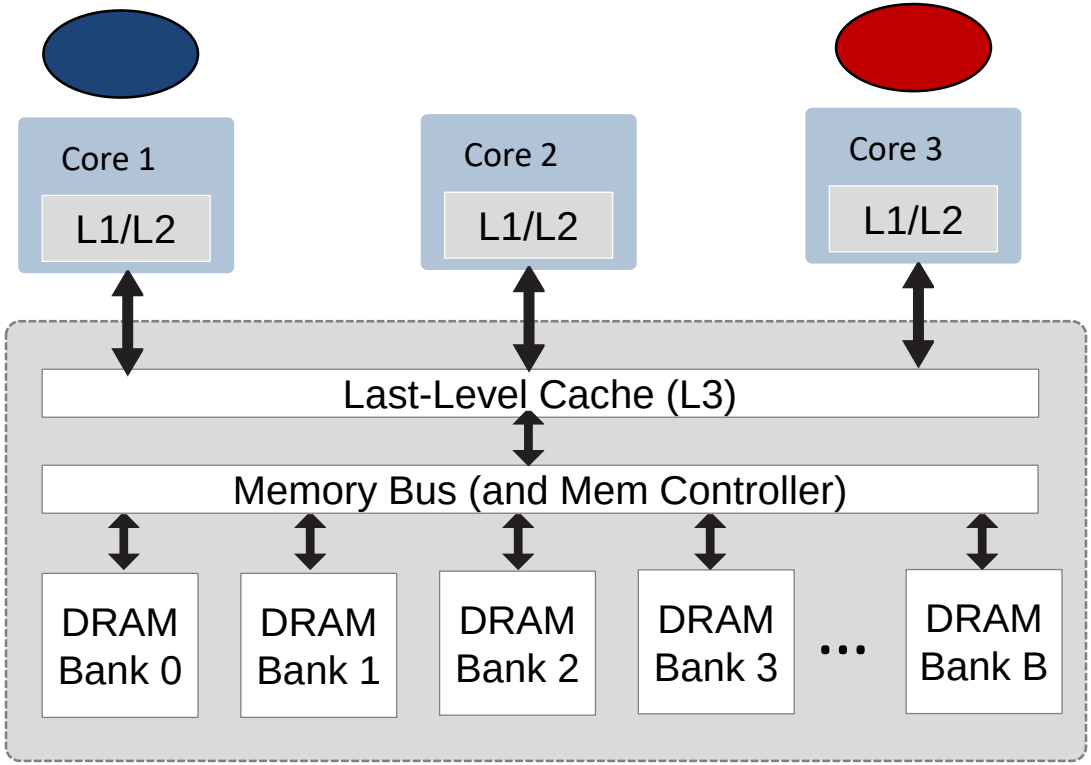


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{blue}	0.25

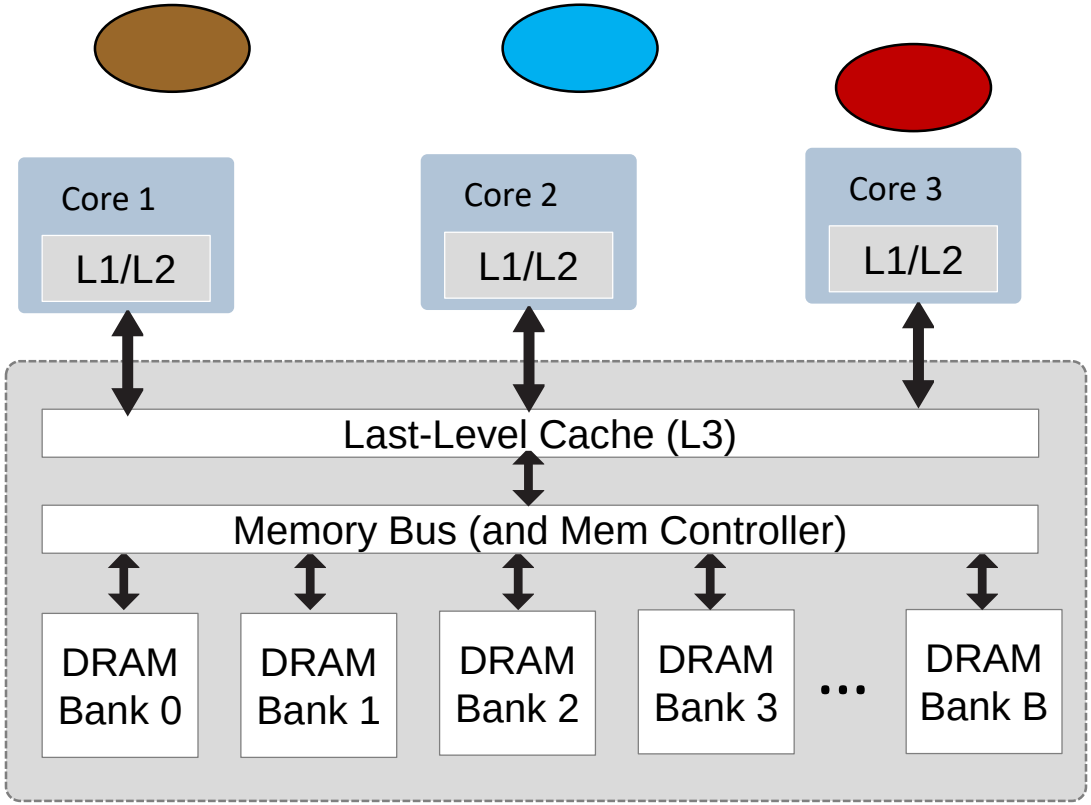


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{cyan, brown}	0.18

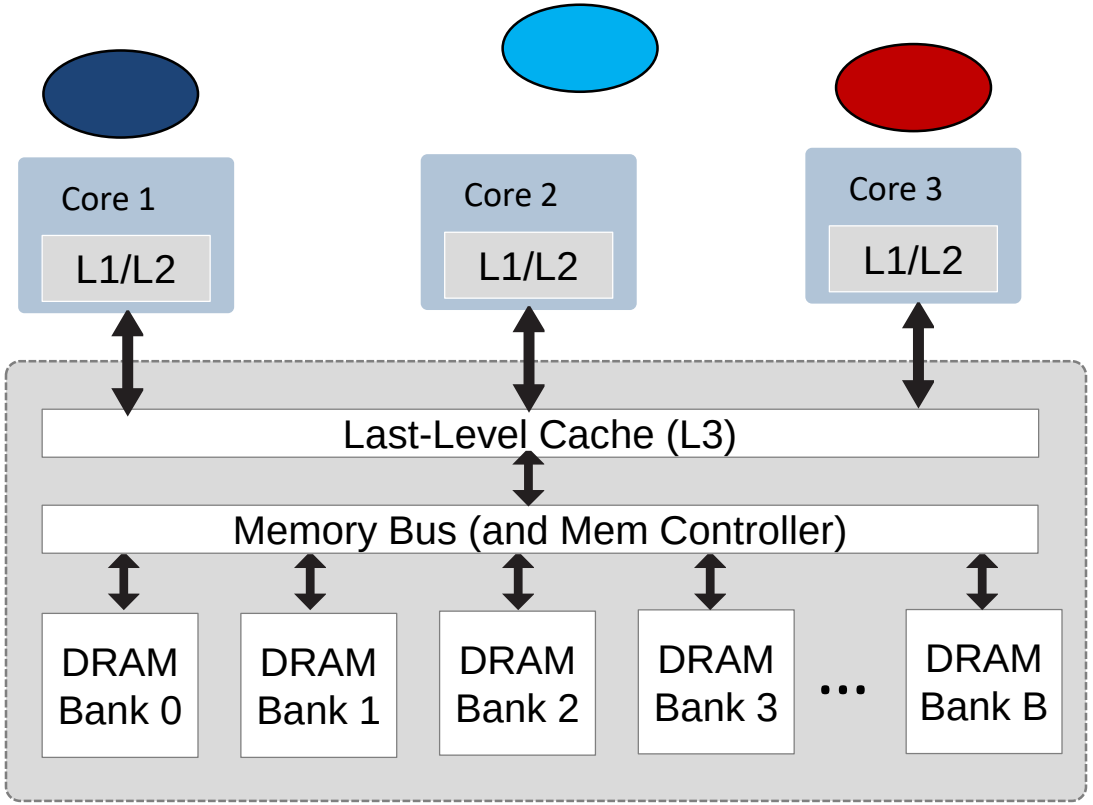


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{cyan, blue}	0.12

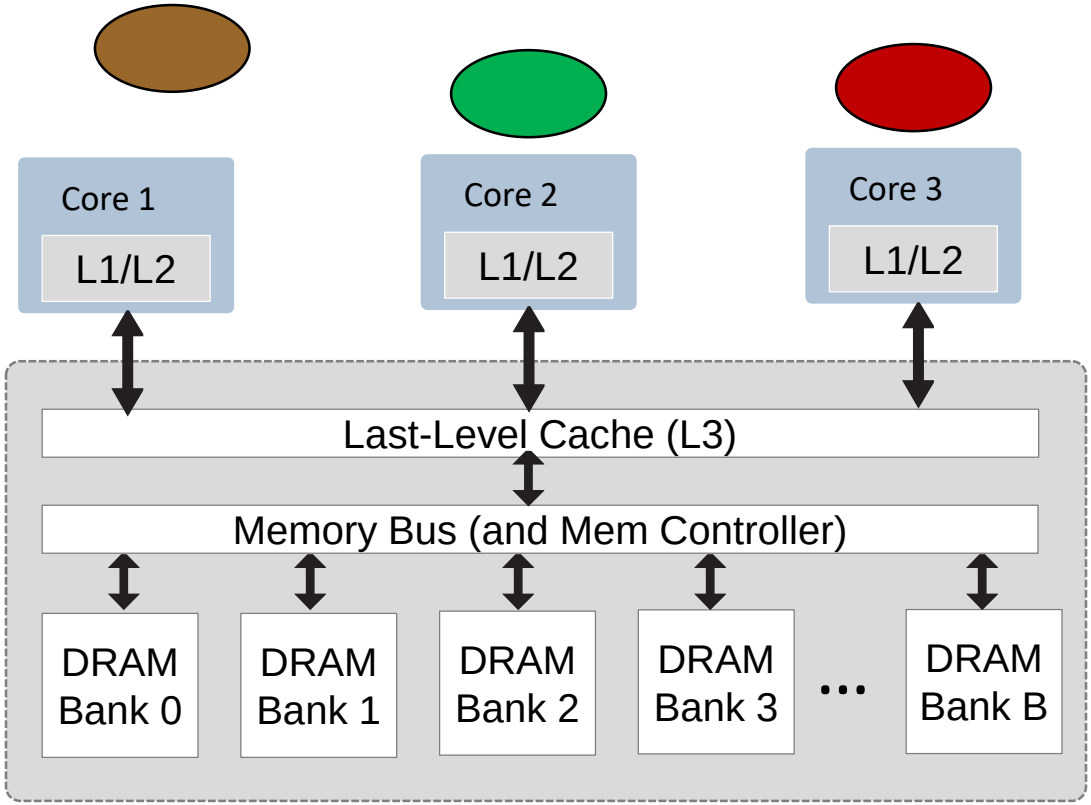


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{green,brown}	0.13

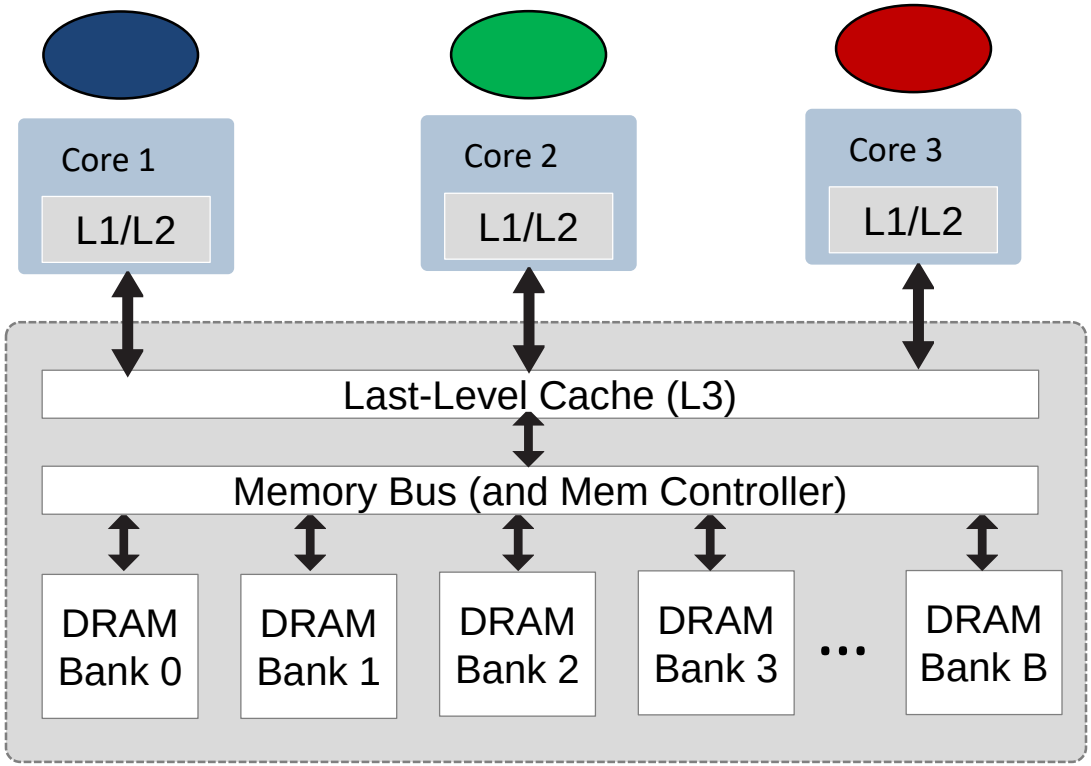


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{green, blue}	0.19

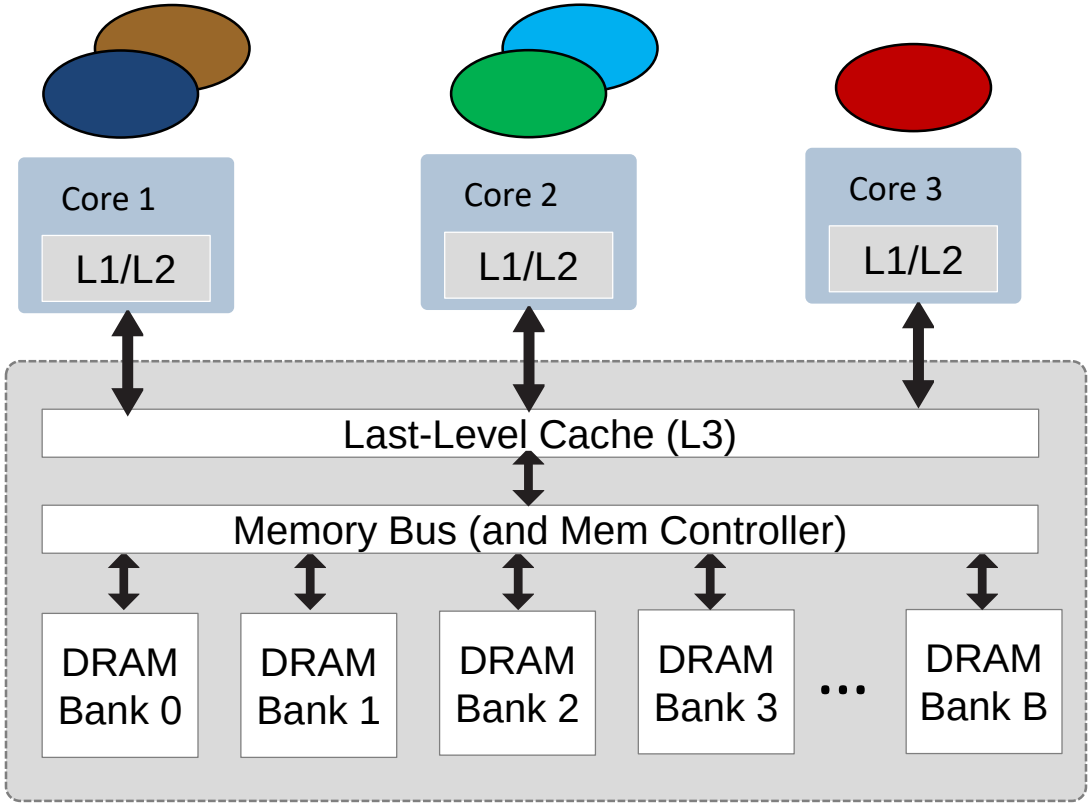


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{cyan}	0.5
{green}	0.45
{brown}	0.45
{blue}	0.25
{cyan, brown}	0.18
{cyan, blue}	0.12
{green,brown}	0.13
{green, blue}	0.19

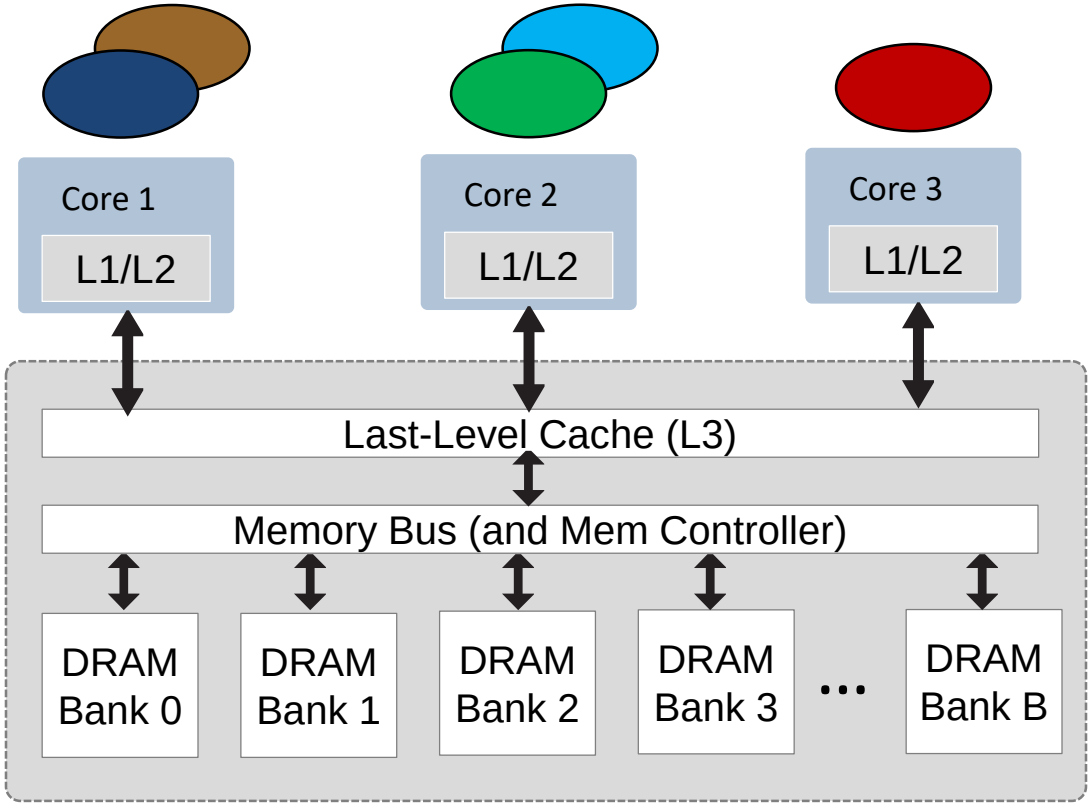


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

C_{red}=4

Co-runner set	Speed
{}	1
{cyan}	0.5
{green}	0.45
{brown}	0.45
{blue}	0.25
{cyan, brown}	0.18
{cyan, blue}	0.12
{green,brown}	0.13
{green, blue}	0.19



Describe resource consumption of other tasks analogously.

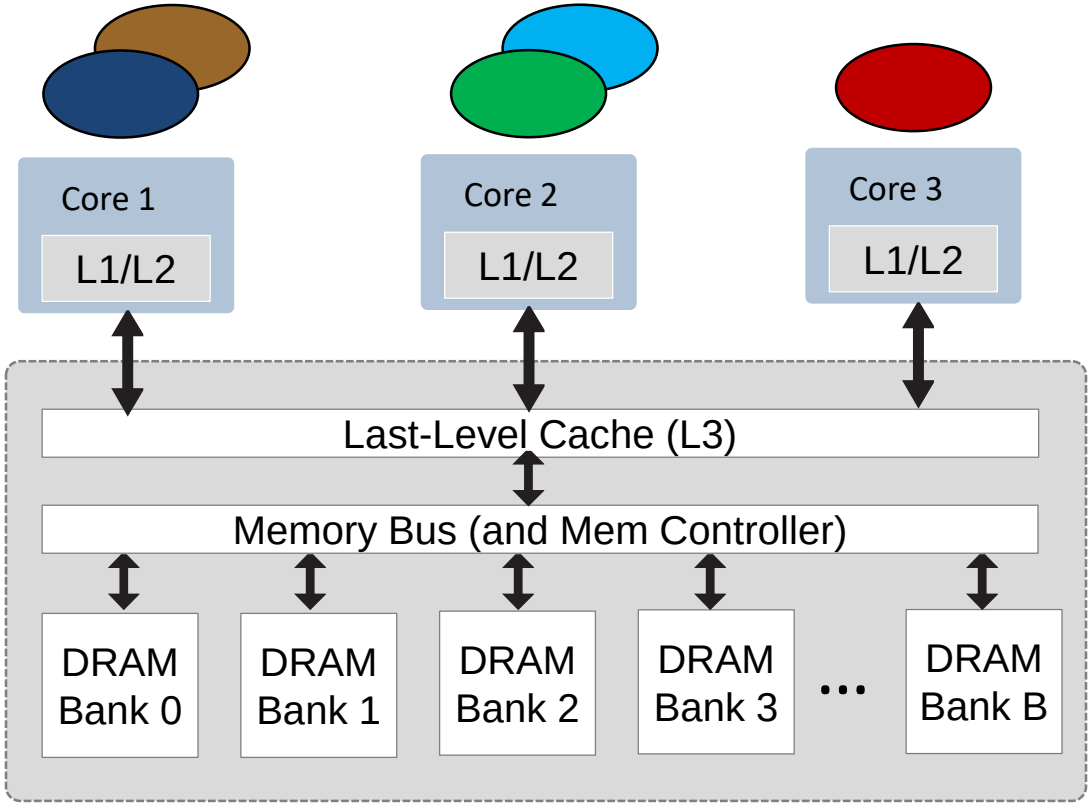
Considerations in creating an abstraction for multicore

Main idea:

For each task *i*,

for each co-runner set of task *i* do

$$lowerboundspeed_i^{cor} \leq speed(i,t) \leq 1$$



For each task: Enumerate the set of possible co-runner set.

Isn't that exponential?
Couldn't this be bad?

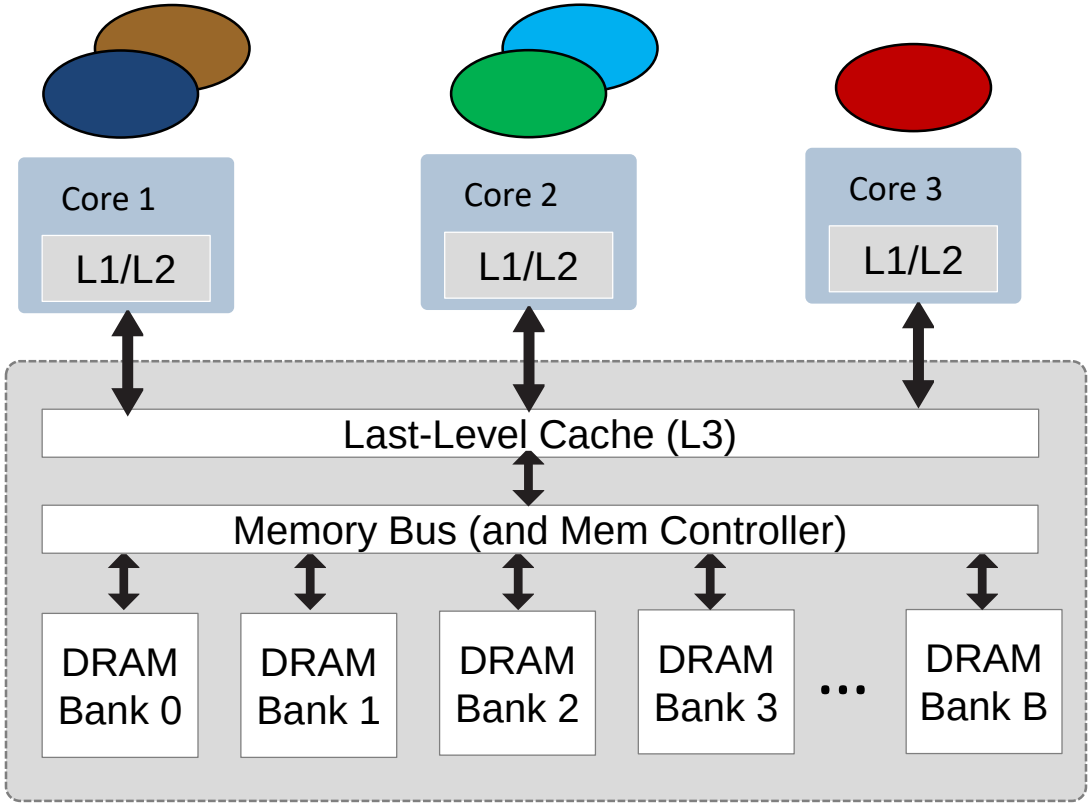
Considerations in creating an abstraction for multicore

Main idea:

For each task *i*,

for each co-runner set of task *i* do

$$lowerboundspeed_i^{cor} \leq speed(i,t) \leq 1$$



Given a task i: #co-runner sets of task i ≤ $\left(\frac{ntasks-1}{nprocessors-1}+1\right)^{nprocessors-1}$

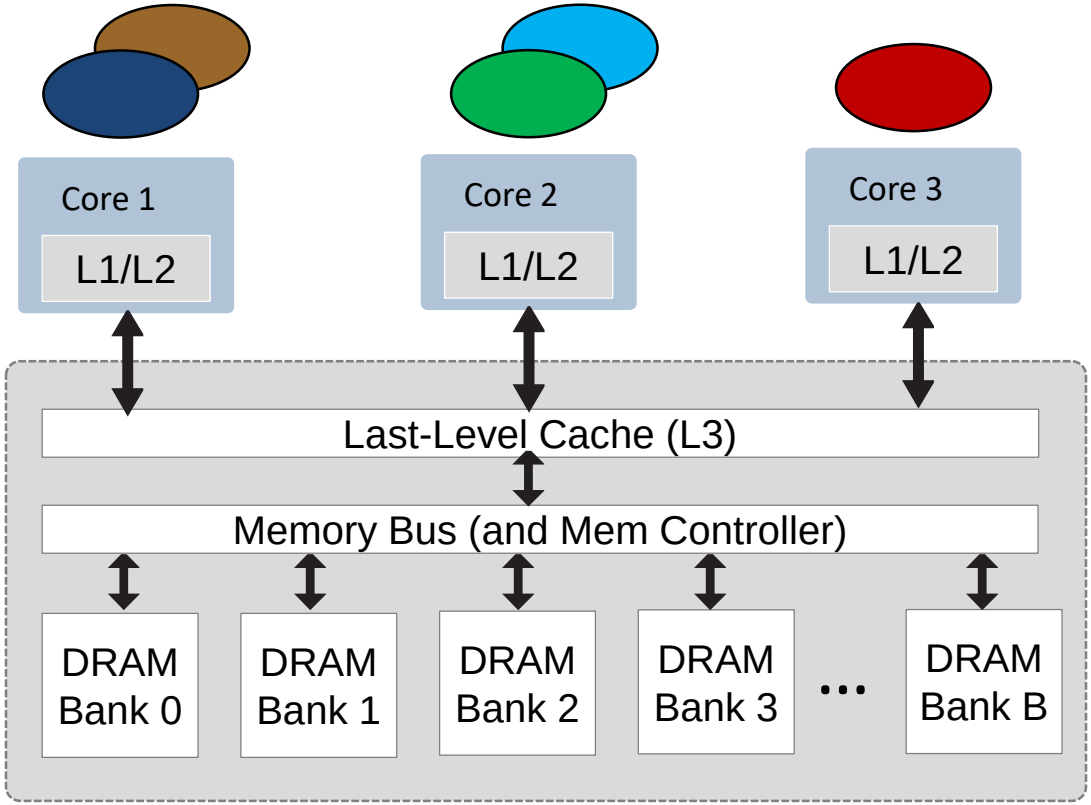
Considerations in creating an abstraction for multicore

Main idea:

For each task *i*,

for each co-runner set of task *i* do

$$lowerboundspeed_i^{cor} \leq speed(i,t) \leq 1$$



Given a task i: #co-runner sets of task i ≤ polynomial

If number of processors is fixed.

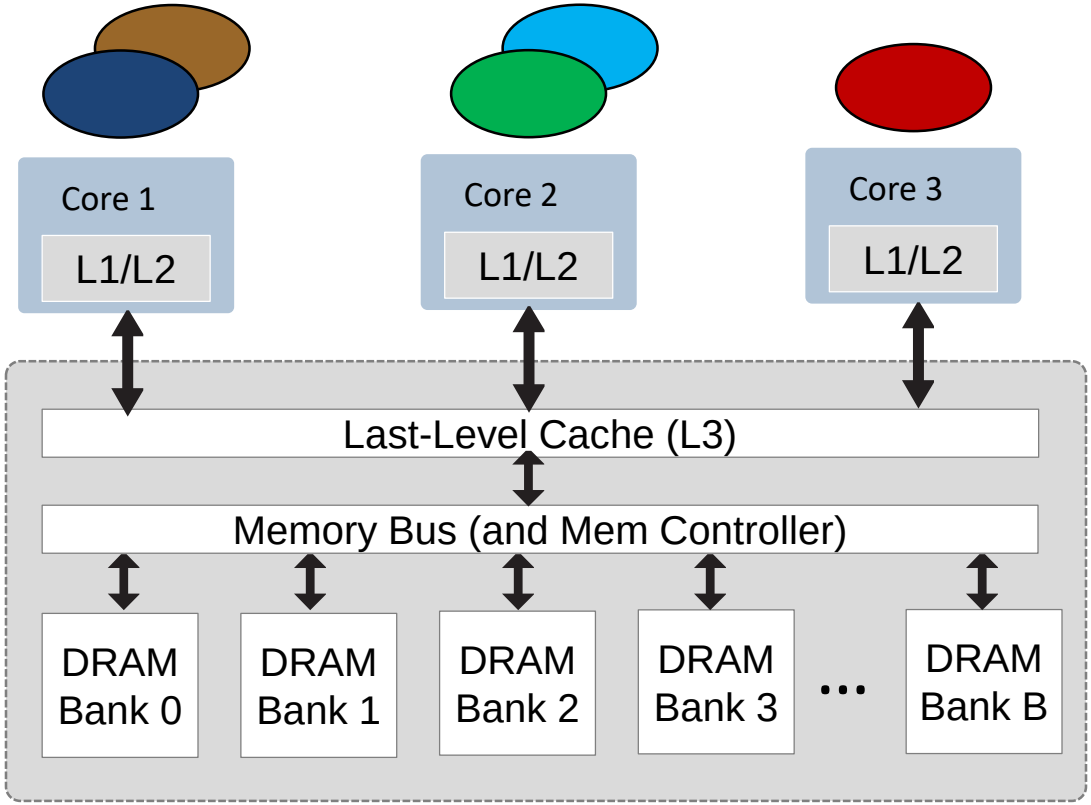
Considerations in creating an abstraction for multicore

Main idea:

For each task *i*,

for each co-runner set of task *i* do

$$lowerboundspeed_i^{cor} \leq speed(i,t) \leq 1$$



Given a task i: #co-runner sets of task i ≤ ntasks

If number of processors = 2.

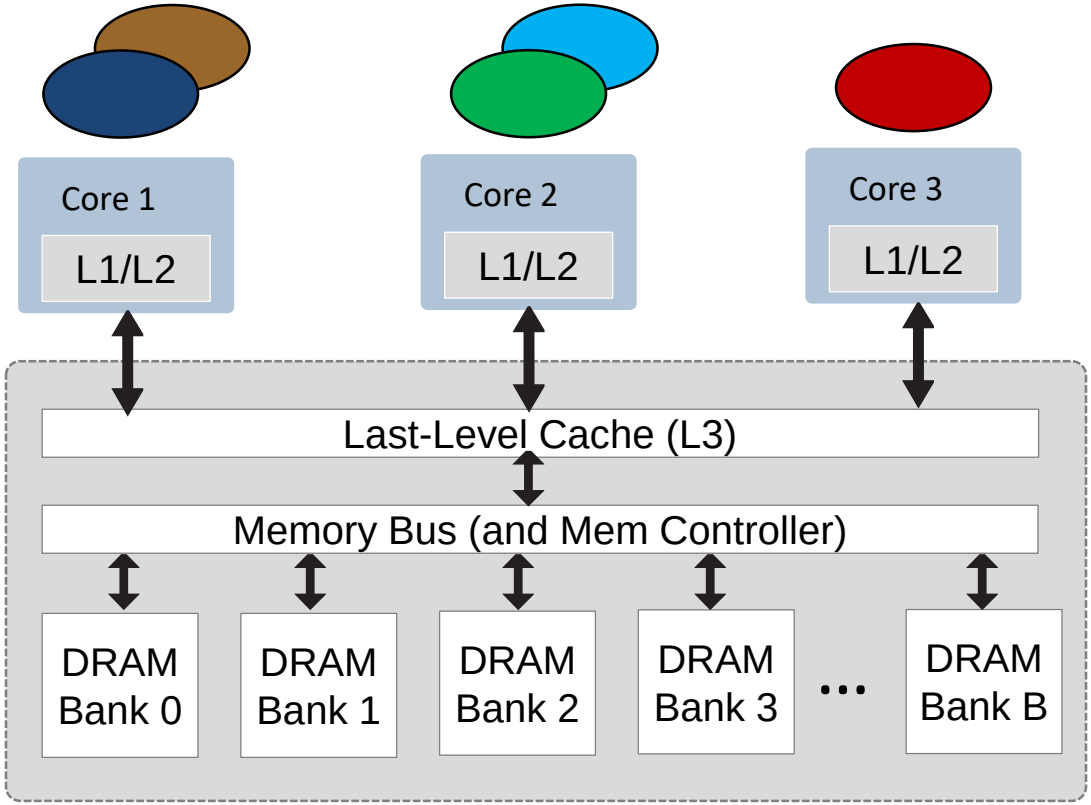
Considerations in creating an abstraction for multicore

Main idea:

For each task *i*,

for each co-runner set of task *i* do

$$lowerboundspeed_i^{cor} \leq speed(i,t) \leq 1$$



Given a task i: #co-runner sets of task i ≤ $\left(\frac{ntasks-1}{2}+1\right)^2$

If number of processors = 3.

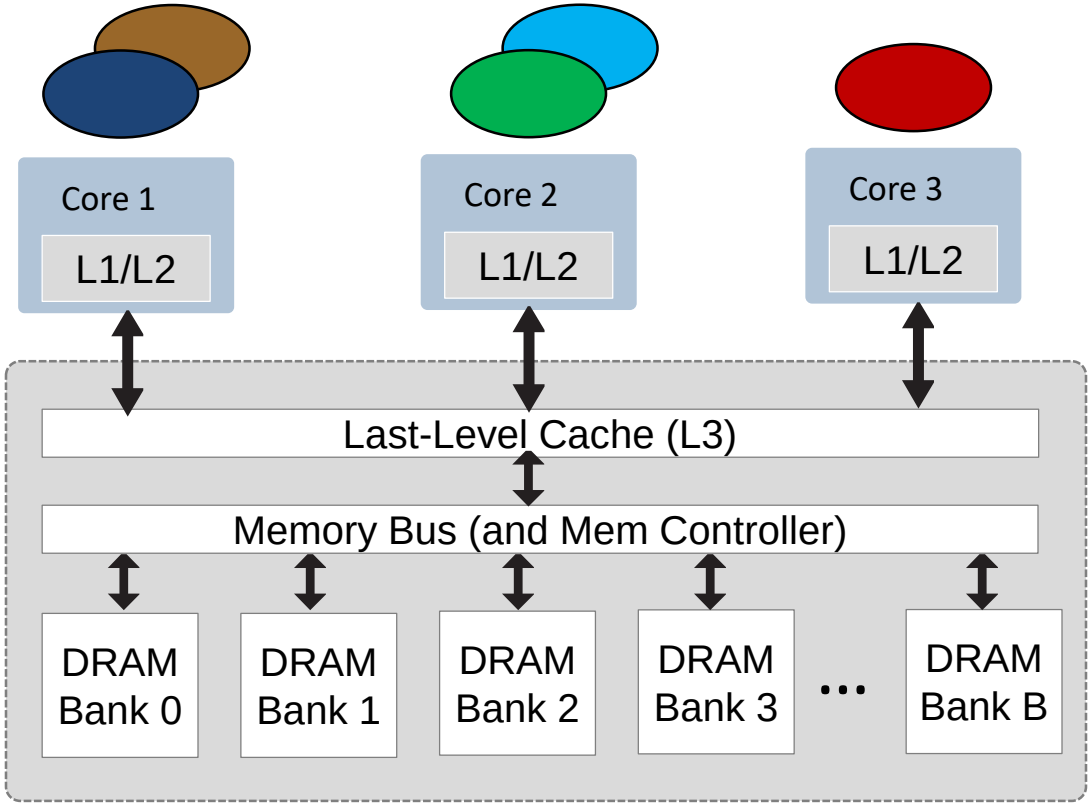
Considerations in creating an abstraction for multicore

Main idea:

For each task *i*,

for each co-runner set of task *i* do

$$lowerboundspeed_i^{cor} \leq speed(i,t) \leq 1$$

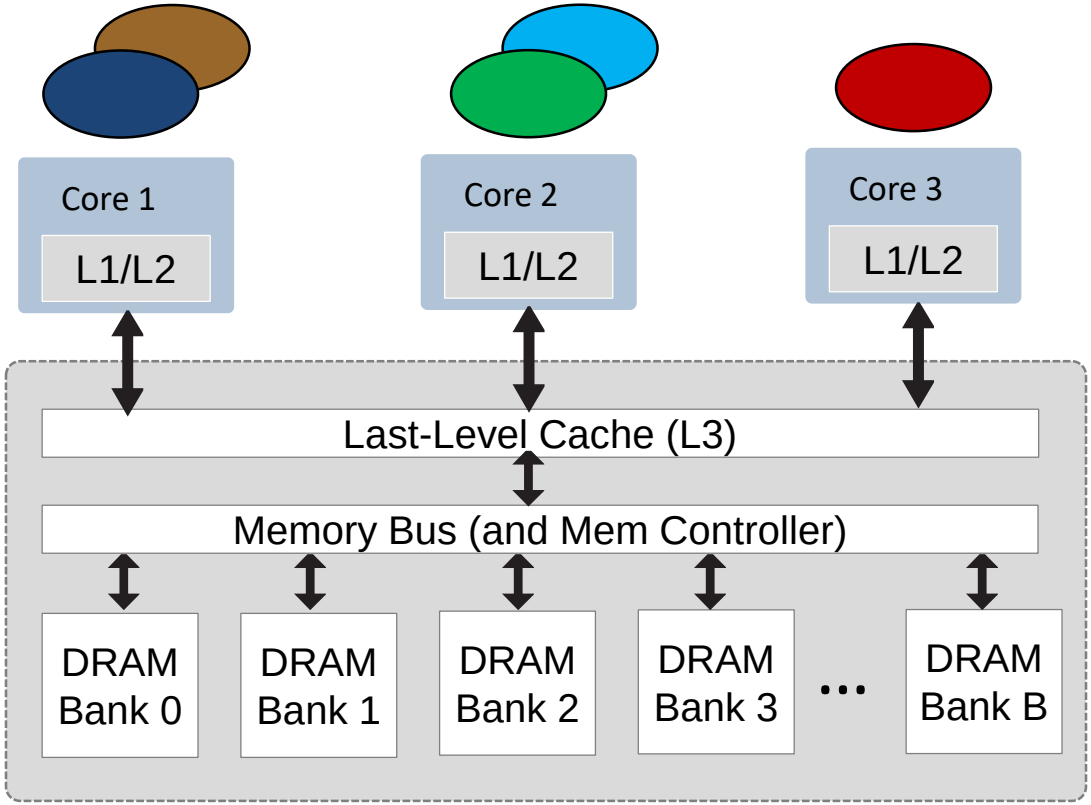


Given a task i: #co-runner sets of task i $\leq \left(\frac{ntasks-1}{3}+1\right)^3$

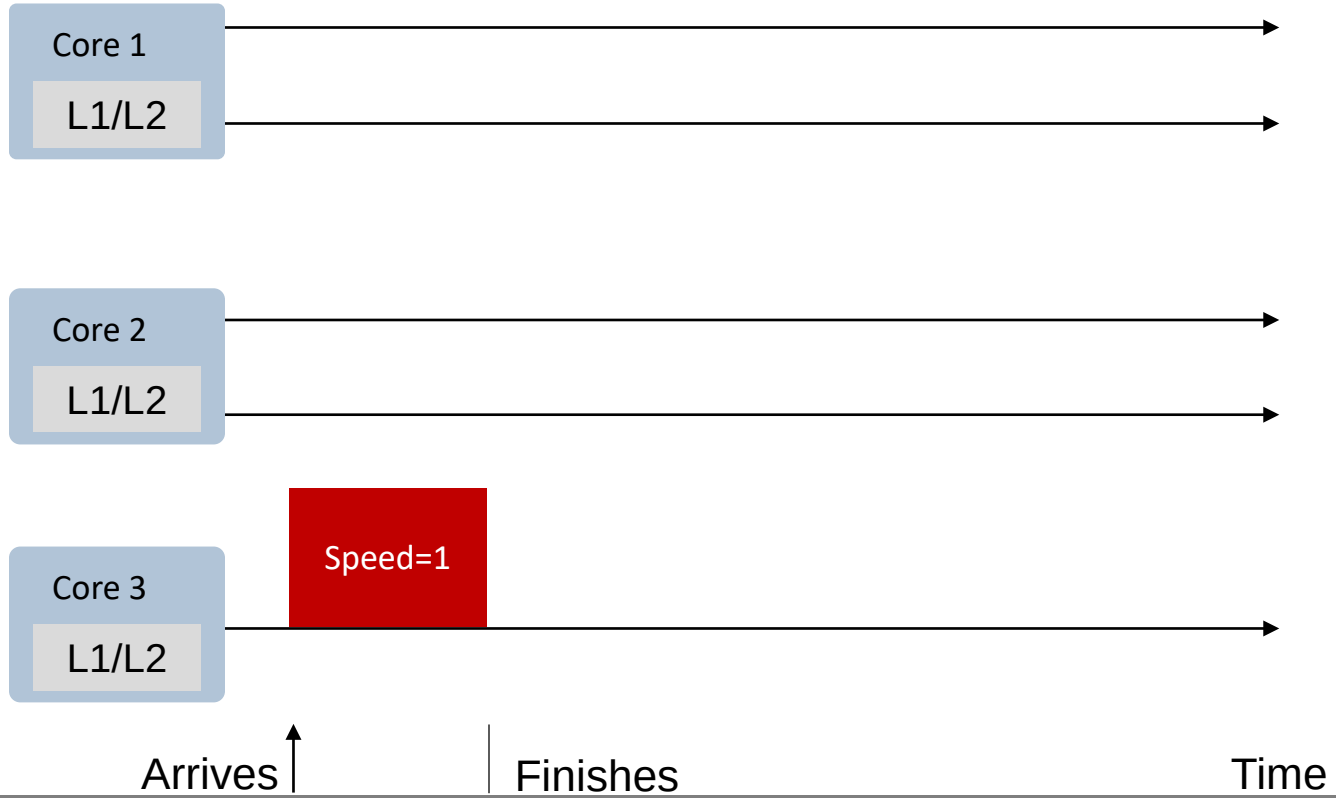
If number of processors = 4.

Considerations in creating an abstraction for multicore

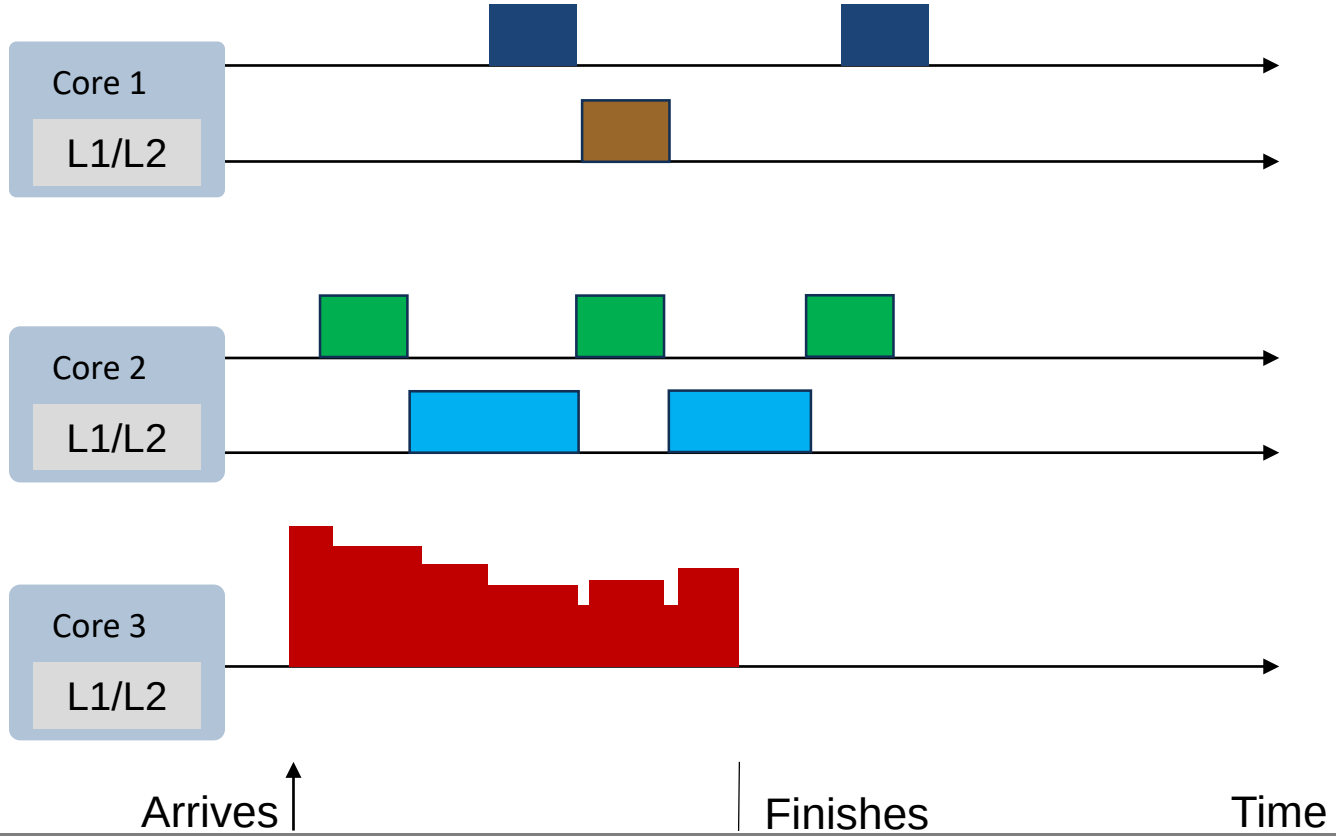
Given a task *i*: A job of task *i* can experience different co-runner set at different times.



How Co-Runners Impact Speed of Execution

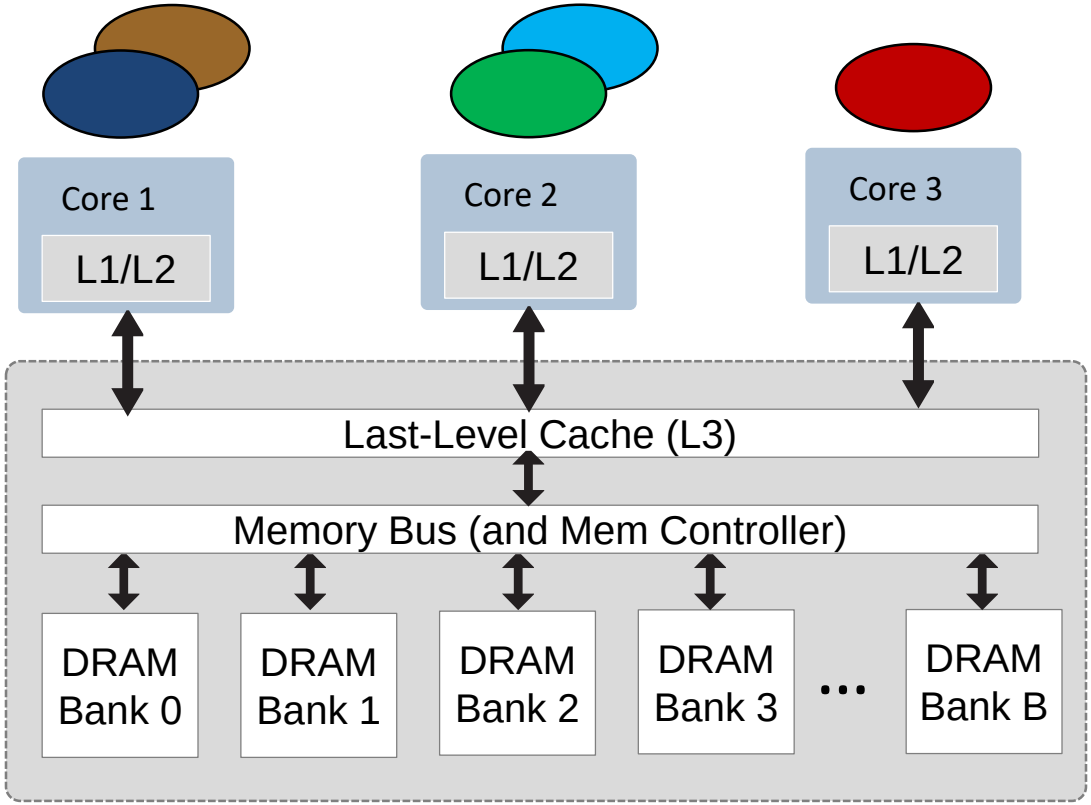


How Co-Runners Impact Speed of Execution



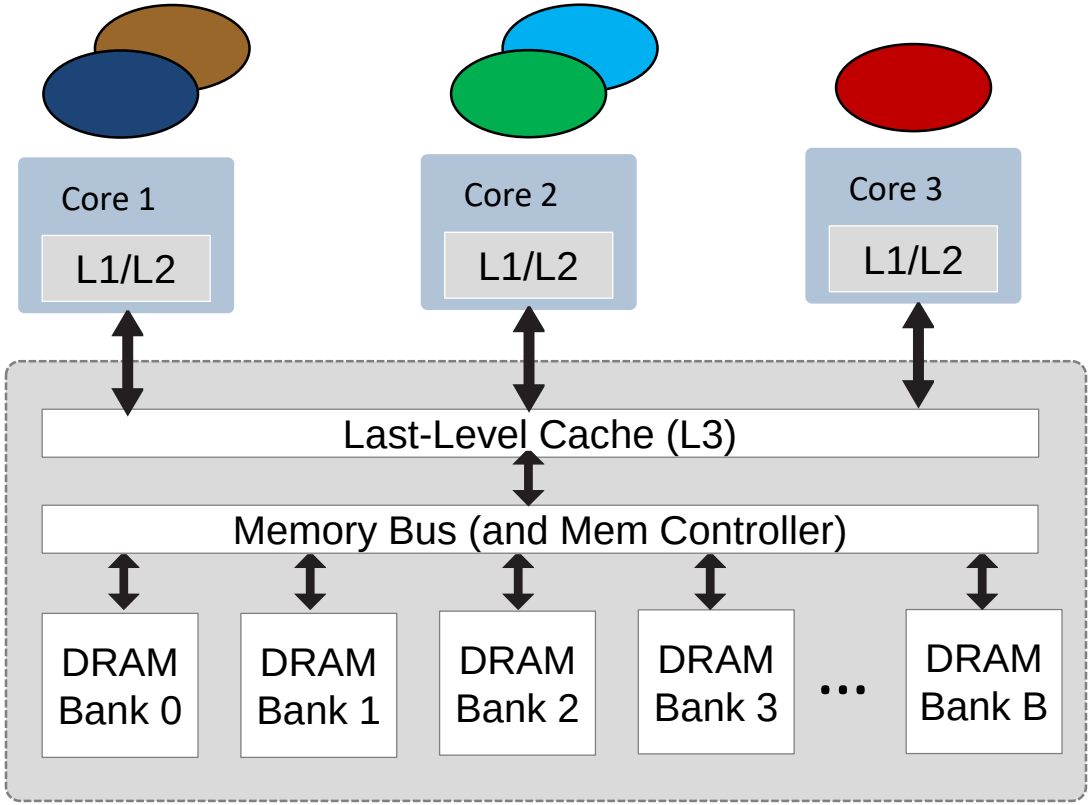
Considerations in creating an abstraction for multicore

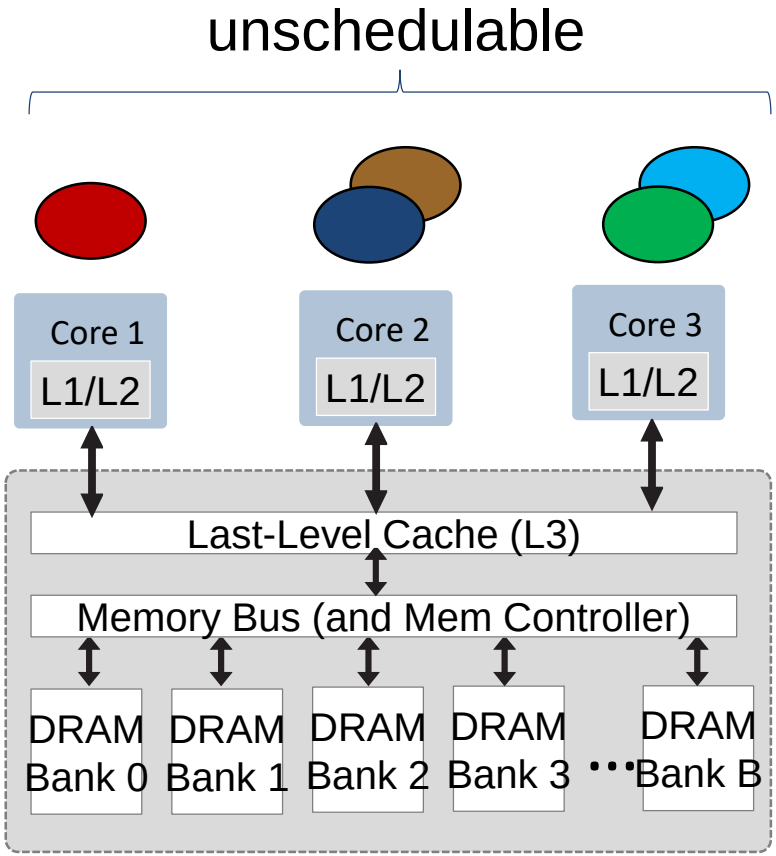
Given a task *i*: A job of task *i* can experience different co-runner set at different times.

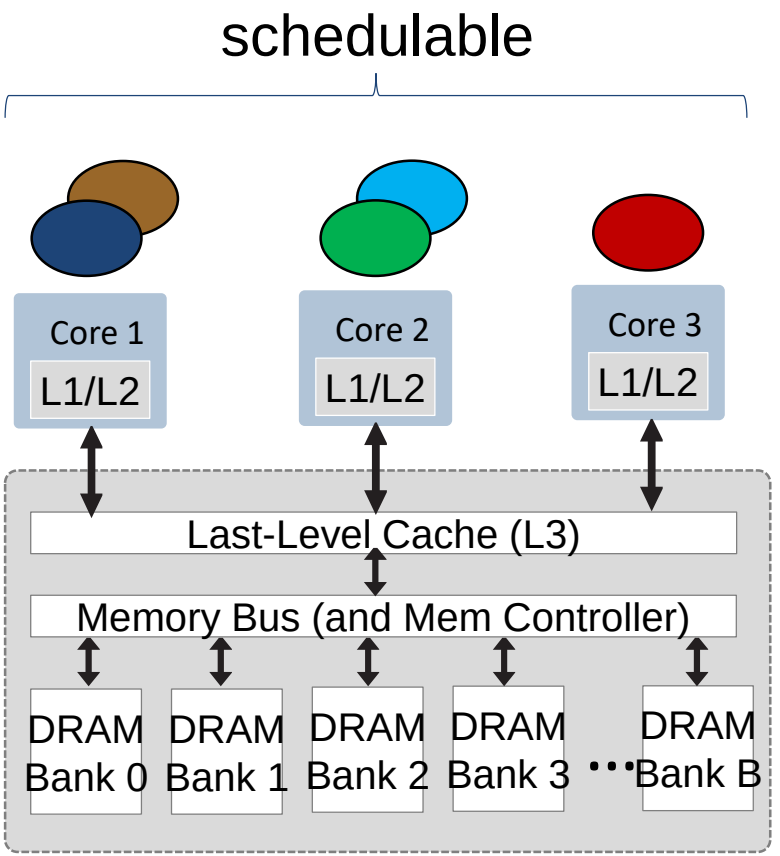


Considerations in creating an abstraction for multicore

In some computer systems, there are some resources (e.g., memory bus) where the arbitration depends on the processor id of the requestor.

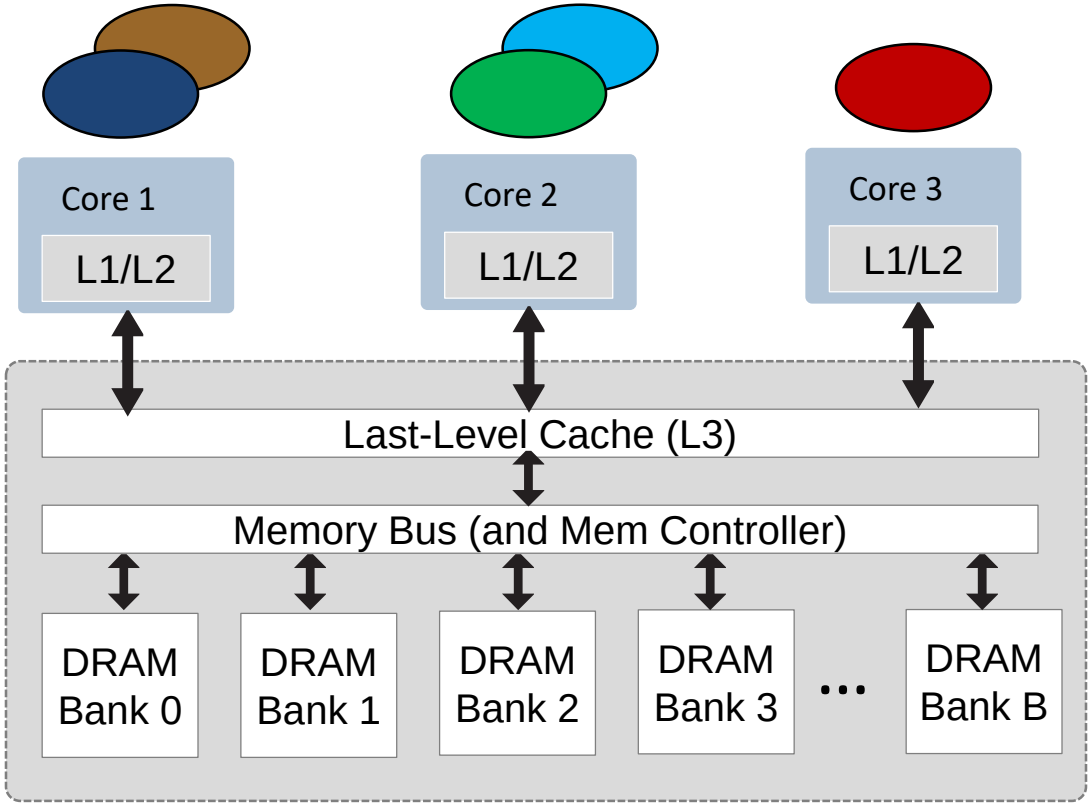






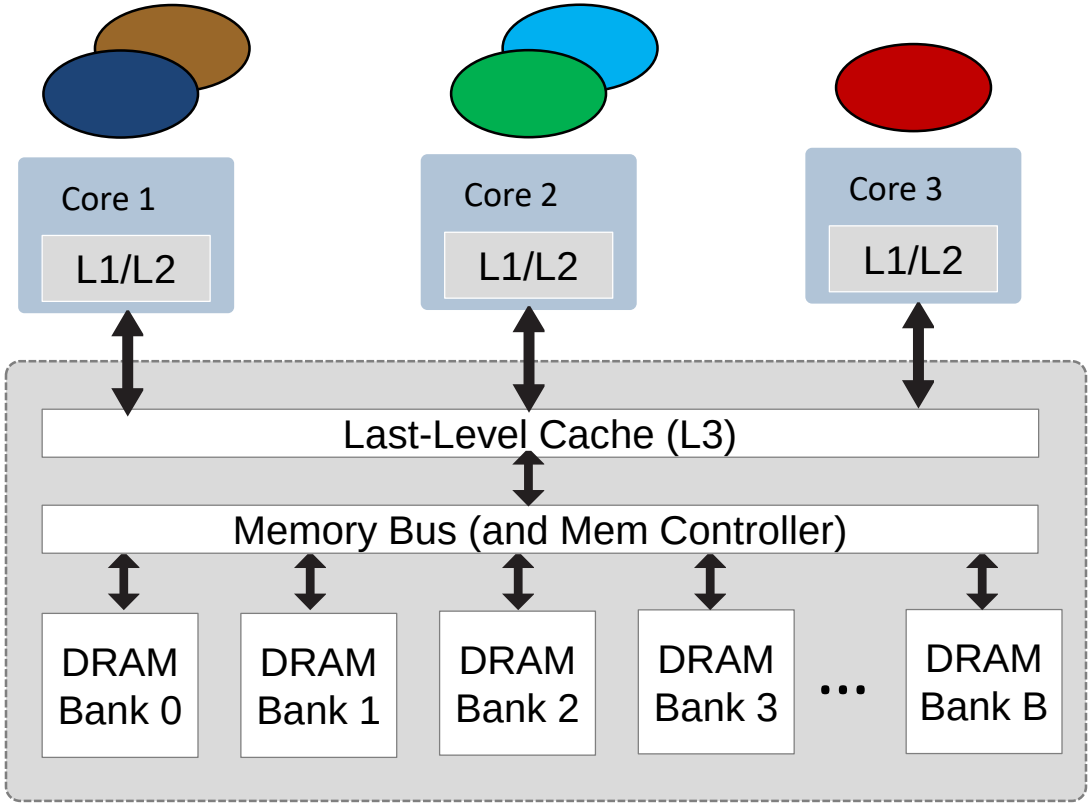
Considerations in creating an abstraction for multicore

In some computer systems, there are some resources (e.g., memory bus) where the arbitration depends on the processor id of the requestor.



Considerations in creating an abstraction for multicore

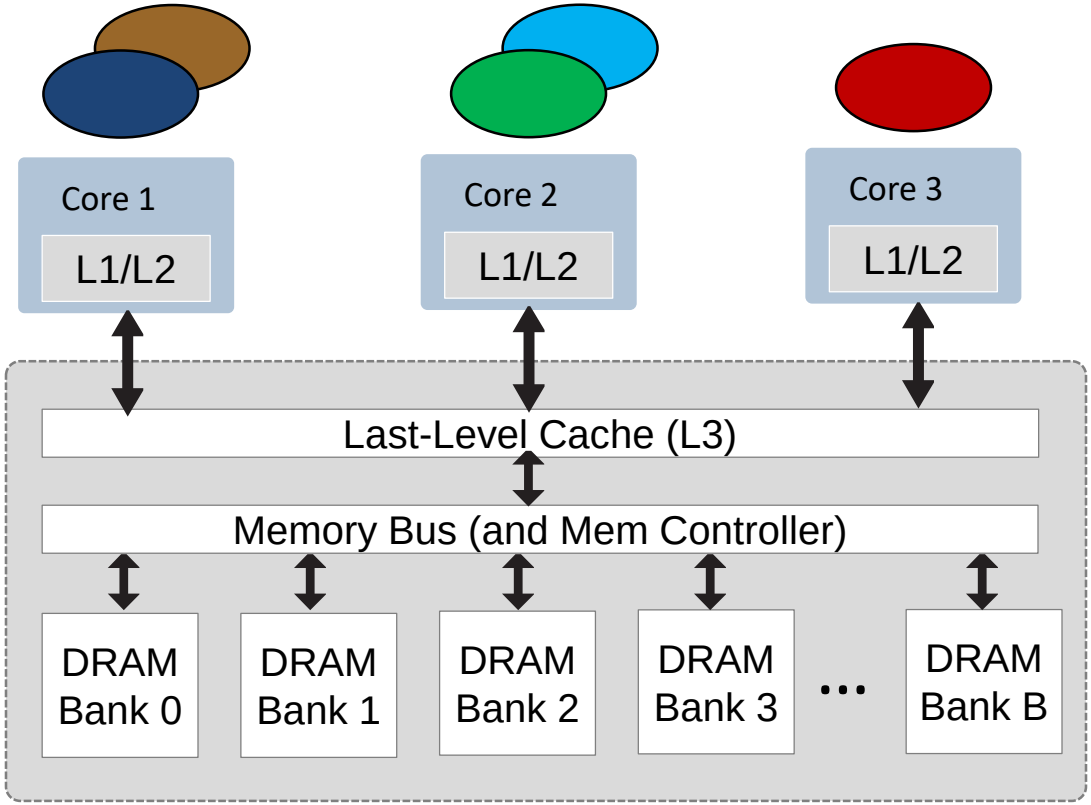
The program behavior may change over time.



Considerations in creating an abstraction for multicore

The program behavior may change over time.

```
while (1) {
  s = wait_until_next_sample()
  update_datastructures(s)
  a = compute_actuation_command()
  actuate_command(a)
}
```

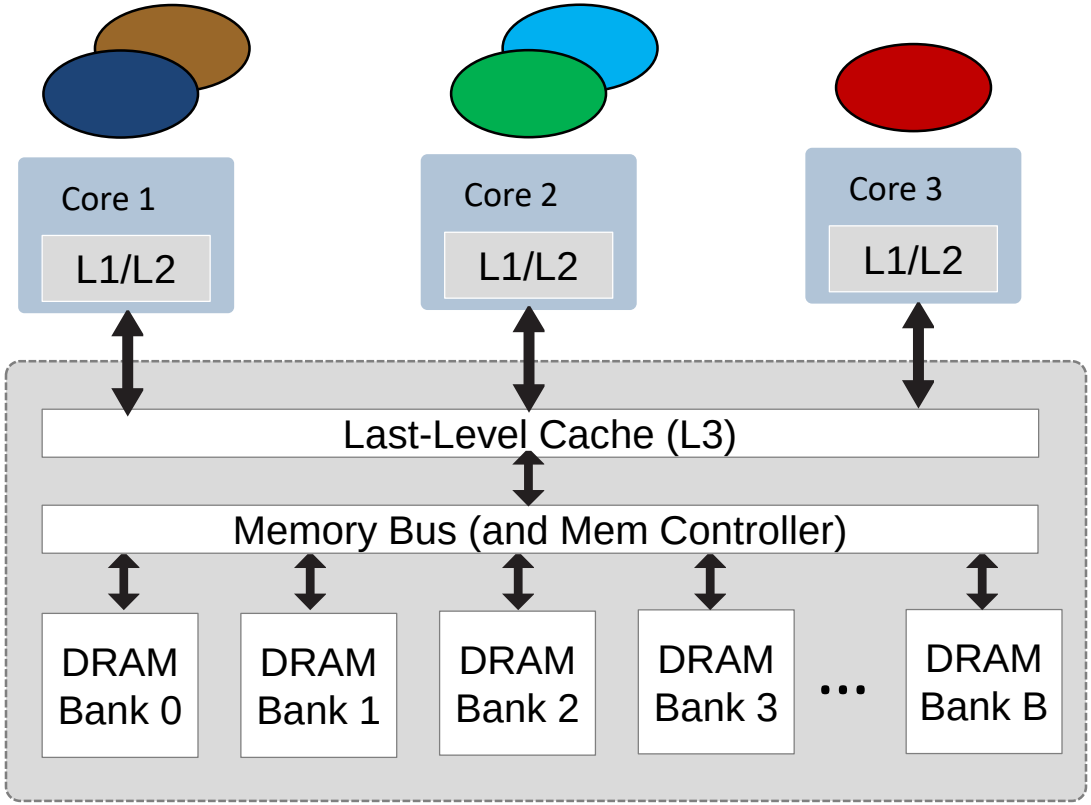


Considerations in creating an abstraction for multicore

The program behavior may change over time.

```
while (1) {
  s = wait_until_next_sample()
  update_datastructures(s)
  a = compute_actuation_command()
  actuate_command(a)
}
```

The program behavior here



Considerations in creating an abstraction for multicore

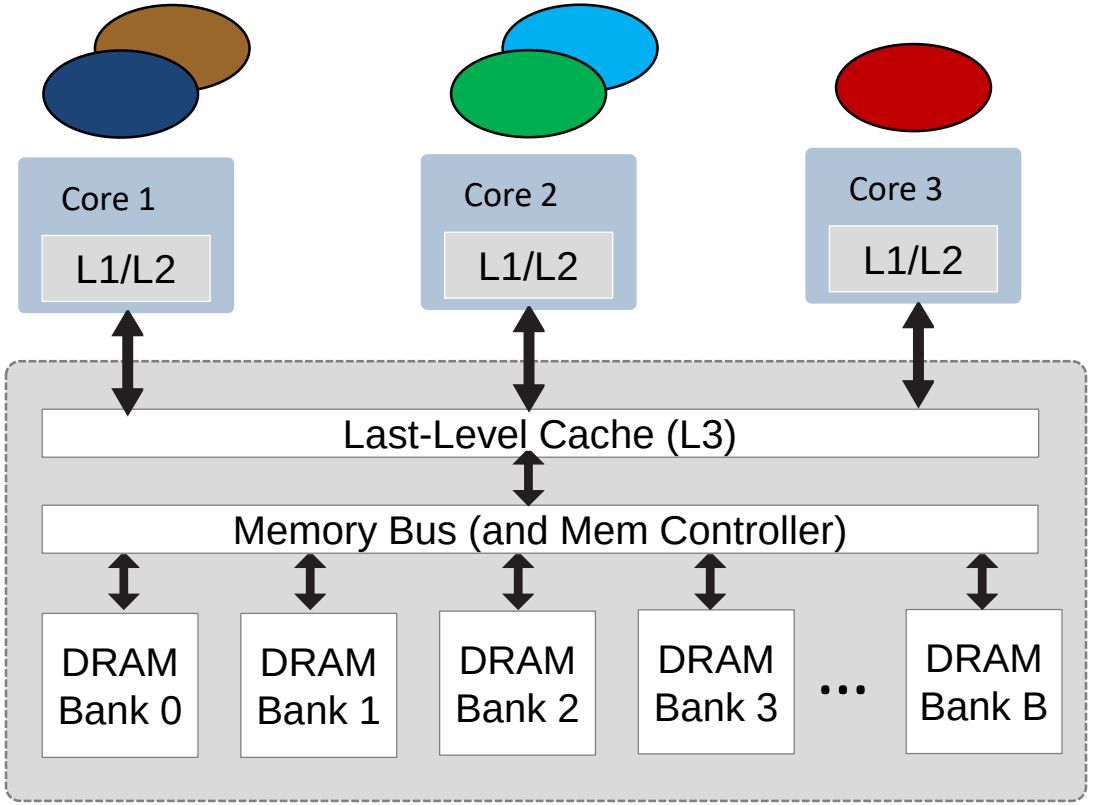
The program behavior may change over time.

```
while (1) {
   s = wait_until_next_sample()
   update_datastructures(s)
   a = compute_actuation_command()
   actuate_command(a)
}
```

The program behavior here

is different from

the program behavior here.

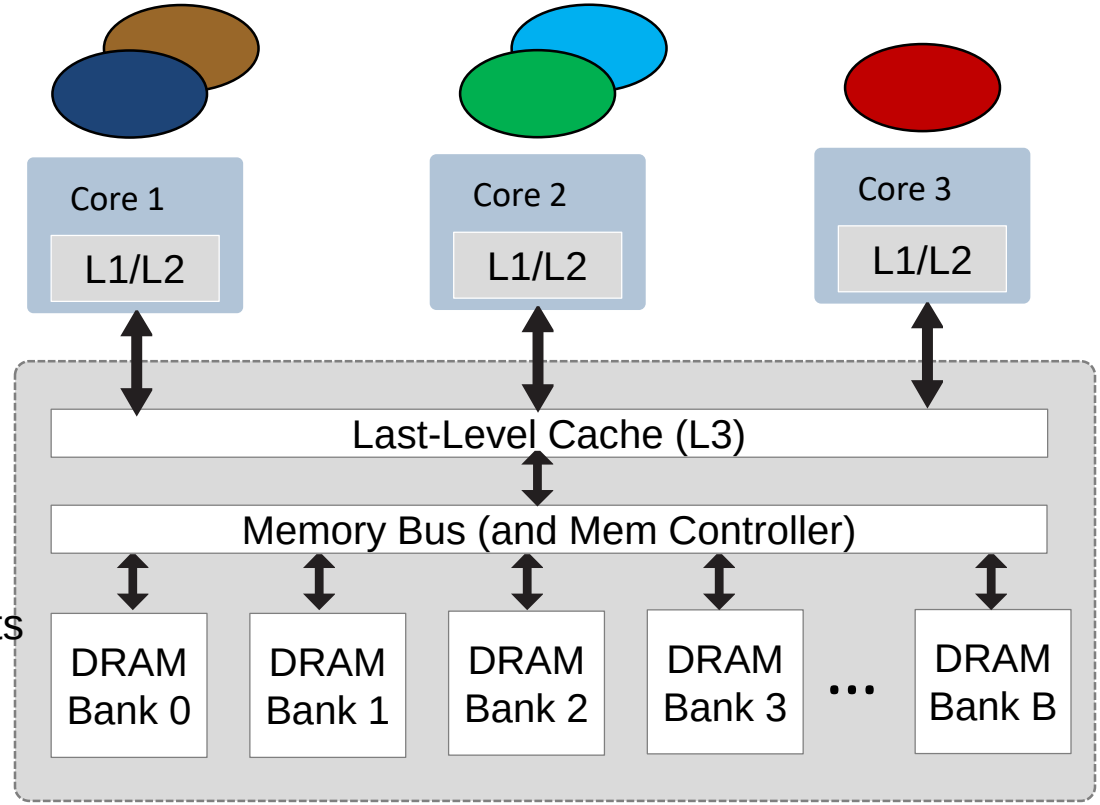


Considerations in creating an abstraction for multicore

The program behavior may change over time.

```
while (1) {
   s = wait_until_next_sample()
   update_datastructures(s)
   a = compute_actuation_command()
   actuate_command(a)
}
```

Describe a task as a sequence of segments where each segment may have different description of lower bound on speed as a function of co-runners.



Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

5 Task

$|\tau| = 5$ $\quad \text{procs}(\Pi) = \{P_1, P_2\}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

2 processors

$|\tau| = 5$ $\quad \boxed{\text{procs}(\Pi) = \{P_1, P_2\}}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

$$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$$
$$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

Task 1

$$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$$
$$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$$
$$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$$

$$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$$
$$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

$$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$$
$$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$$
$$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

Task 2

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

Task 3

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Task 4

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Task 5

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

Minimum inter-arrival times of tasks

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$

$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$

$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$

$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$

$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$

$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$

$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

Deadlines of tasks

$T_1 = 1.500 \quad \boxed{D_1 = 1.500} \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$

$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad \boxed{D_2 = 2.000} \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$

$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad \boxed{D_3 = 2.000} \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$

$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad \boxed{D_4 = 2.000} \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$

$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad \boxed{D_5 = 2.250} \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$

$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$

$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

Sets of segments for each task

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$

$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$

$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$

$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$

$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$

$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$

$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Task 1 has 1 segment.

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad \boxed{V_5 = \{v_5^1, v_5^2\}} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Task 5 has 2 segments.

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

Priorities for each task

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \boxed{\text{prio}_1 = 3} \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \boxed{\text{prio}_2 = 2} \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \boxed{\text{prio}_3 = 3} \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \boxed{\text{prio}_4 = 2} \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \boxed{\text{prio}_5 = 1} \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

Processor to each task is assigned

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \boxed{\text{proc}_1 = 1}$

$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \boxed{\text{proc}_2 = 1}$

$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \boxed{\text{proc}_3 = 2}$

$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \boxed{\text{proc}_4 = 2}$

$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \boxed{\text{proc}_5 = 2}$

$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$

$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

Execution requirement for each segment of each task

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$

$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$

$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$

$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$

$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$

$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$

$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

Speed as function of co-runners

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$

$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$

$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$

$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$

$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$

$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$

$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

Taskset example

If segment 1 of task 1 executes in parallel with segment 1 of task 4, then Segment 1 of task 1 executes with speed 0.5.

$$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$$
$$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$$
$$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$$
$$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$$

$$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$$
$$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

$$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$$
$$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$$
$$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

Taskset example

If segment 1 of task 1 executes in parallel with a segmentset for which information in CO is not available, then segment 1 of task 1 executes with speed 0.5.

$|\tau| = 5 \qquad \text{procs}(\Pi) = \{P_1, P_2\}$

$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$
$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$
$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$

$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$
$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$

$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$
$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$
$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$
$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$

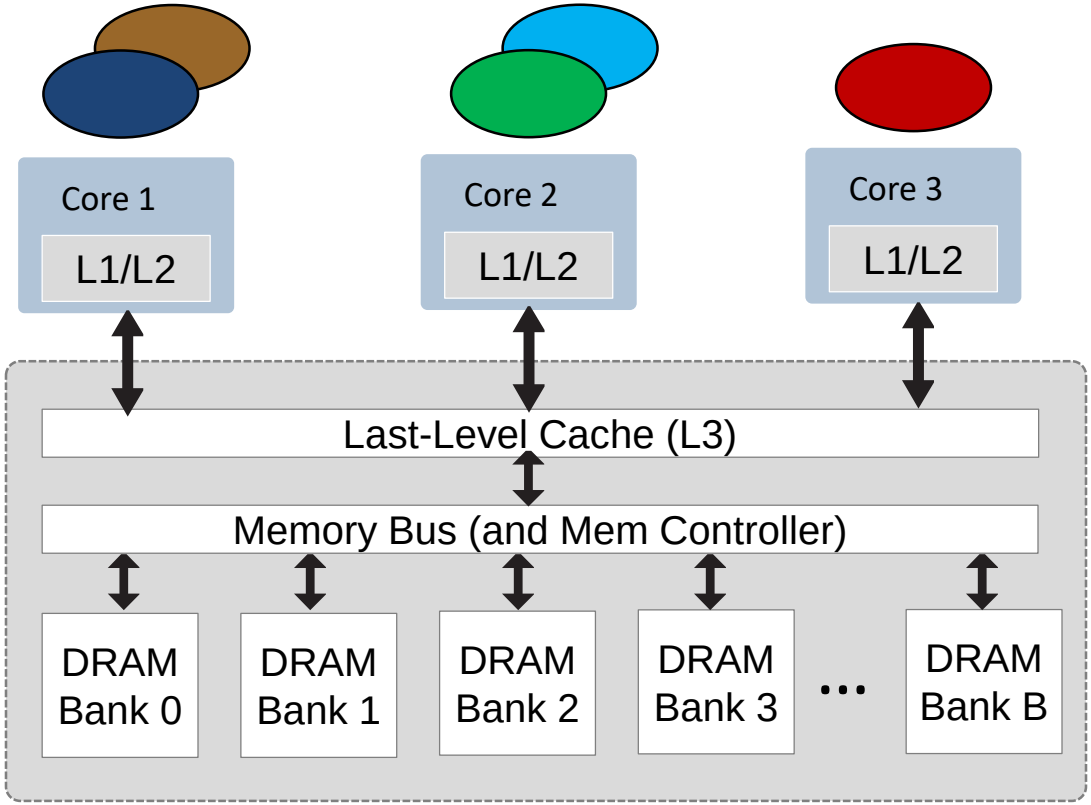
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware? (DONE)

Q2: How to create a schedulability analysis that uses this abstraction?

Before discussing them, let us discuss:

1. Other approaches (DONE)
2. General ideas for abstractions in other disciplines (DONE)

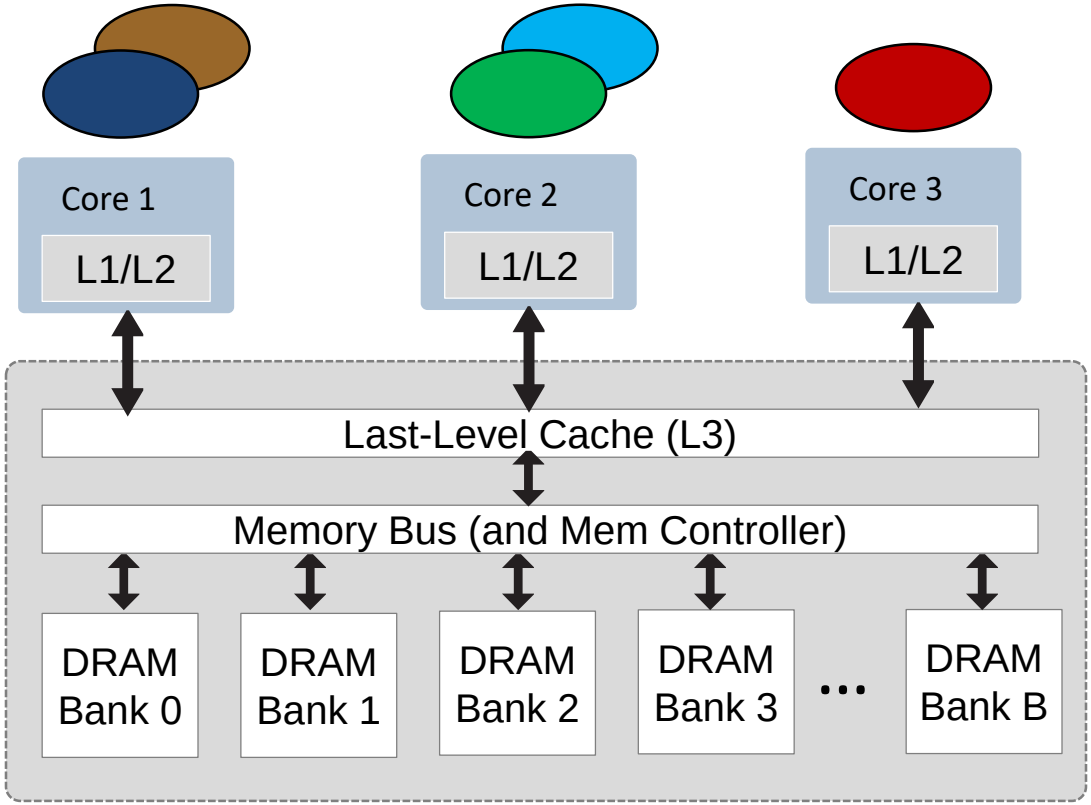


Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

Let us discuss Q2.

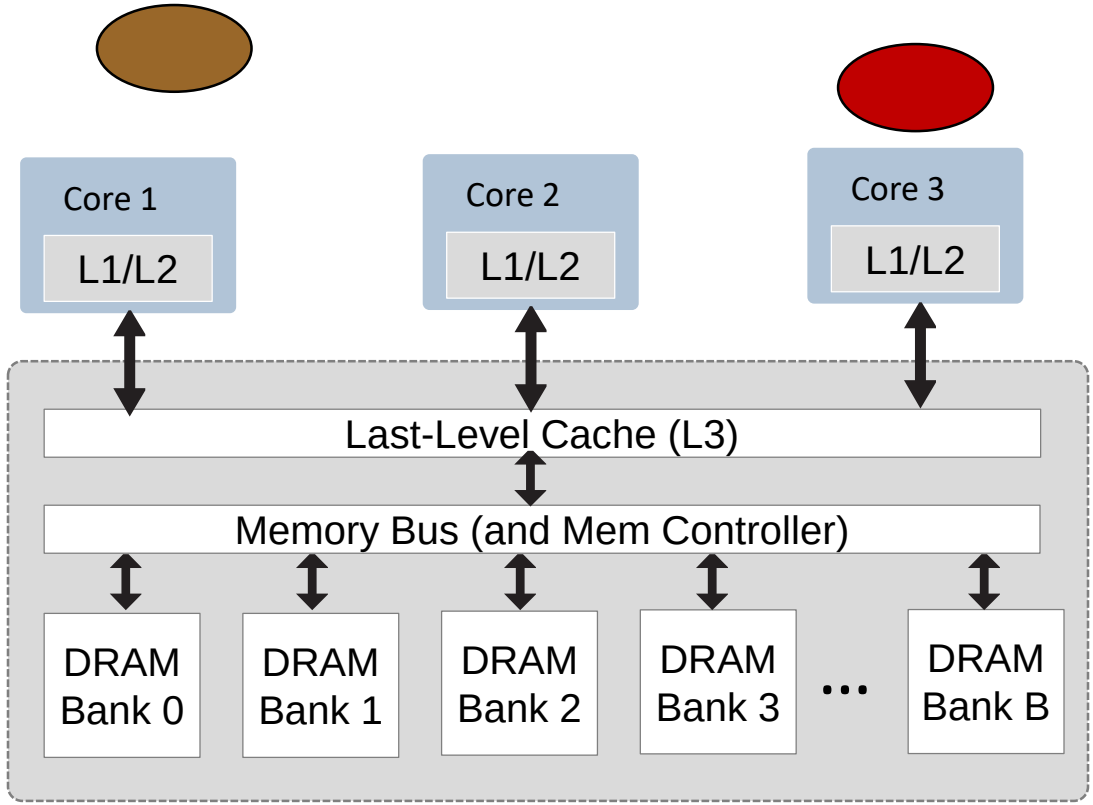


Big Research Questions

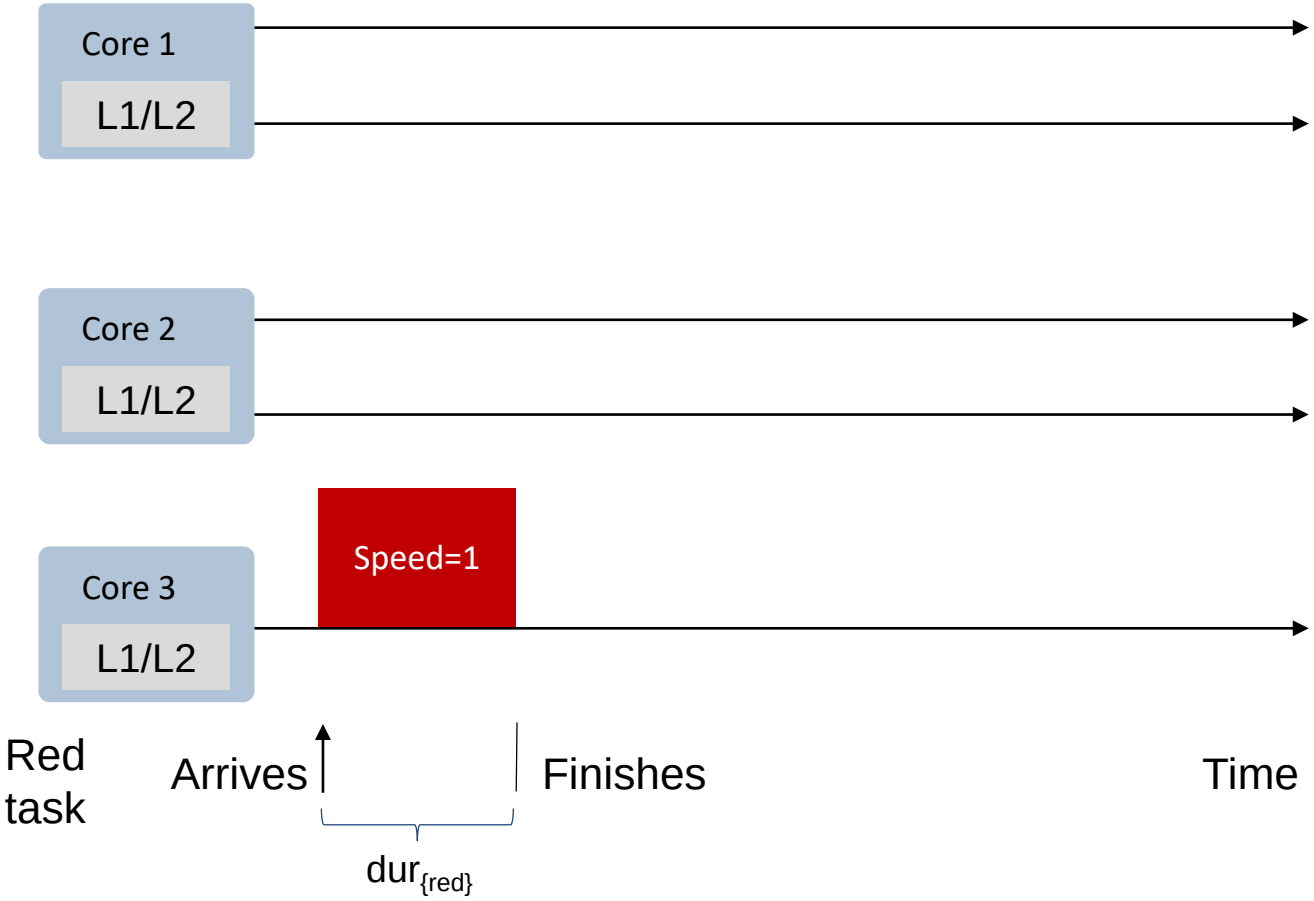
Q1: What is a good abstraction that describes the effect of undocumented hardware?

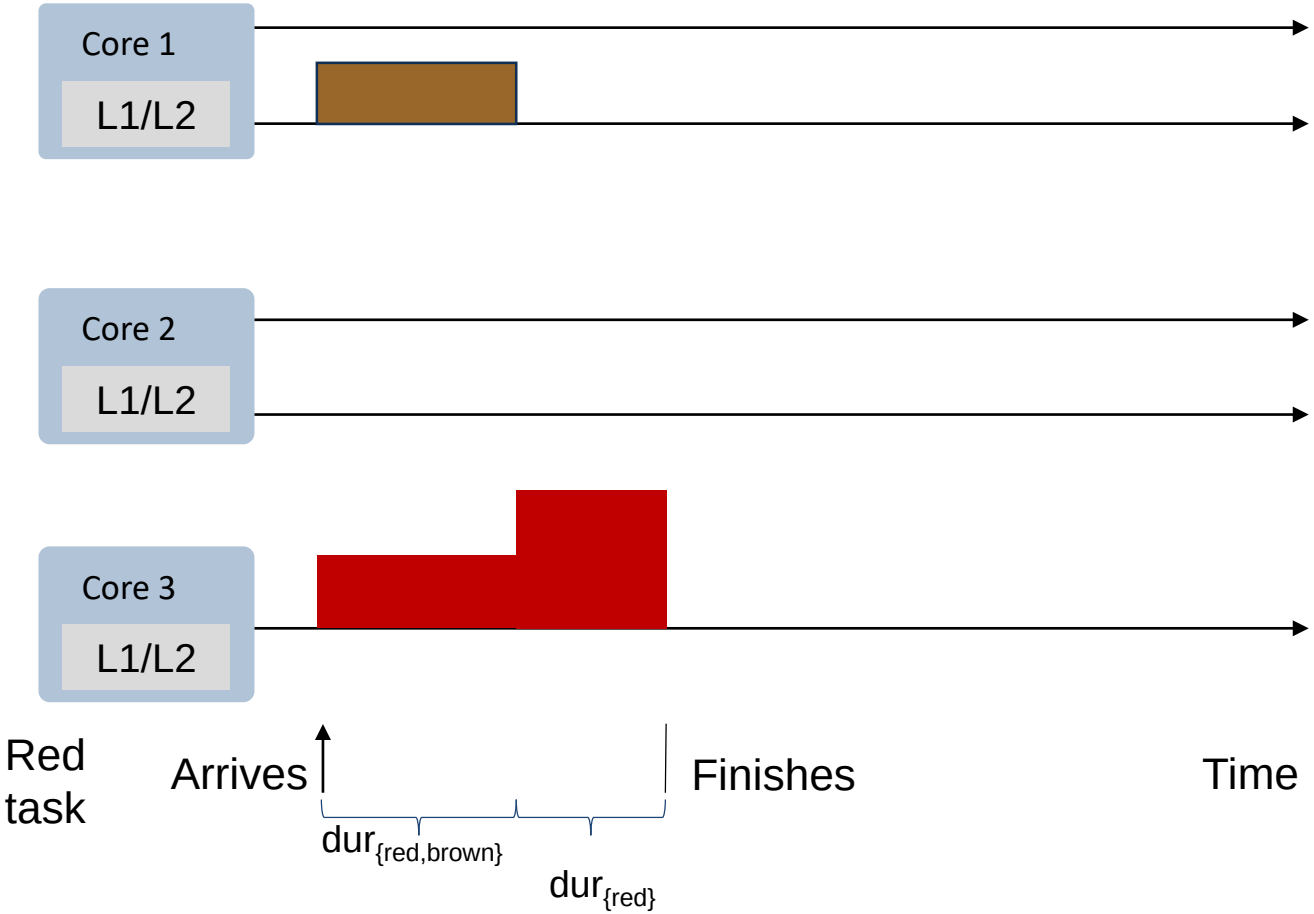
Q2: How to create a schedulability analysis that uses this abstraction?

In order to simplify our discussion initially, let us consider a taskset with only Red and Brown task. Also, let us consider that the minimum inter-arrival time of each of these tasks is infinity (i.e., generates just a single job).



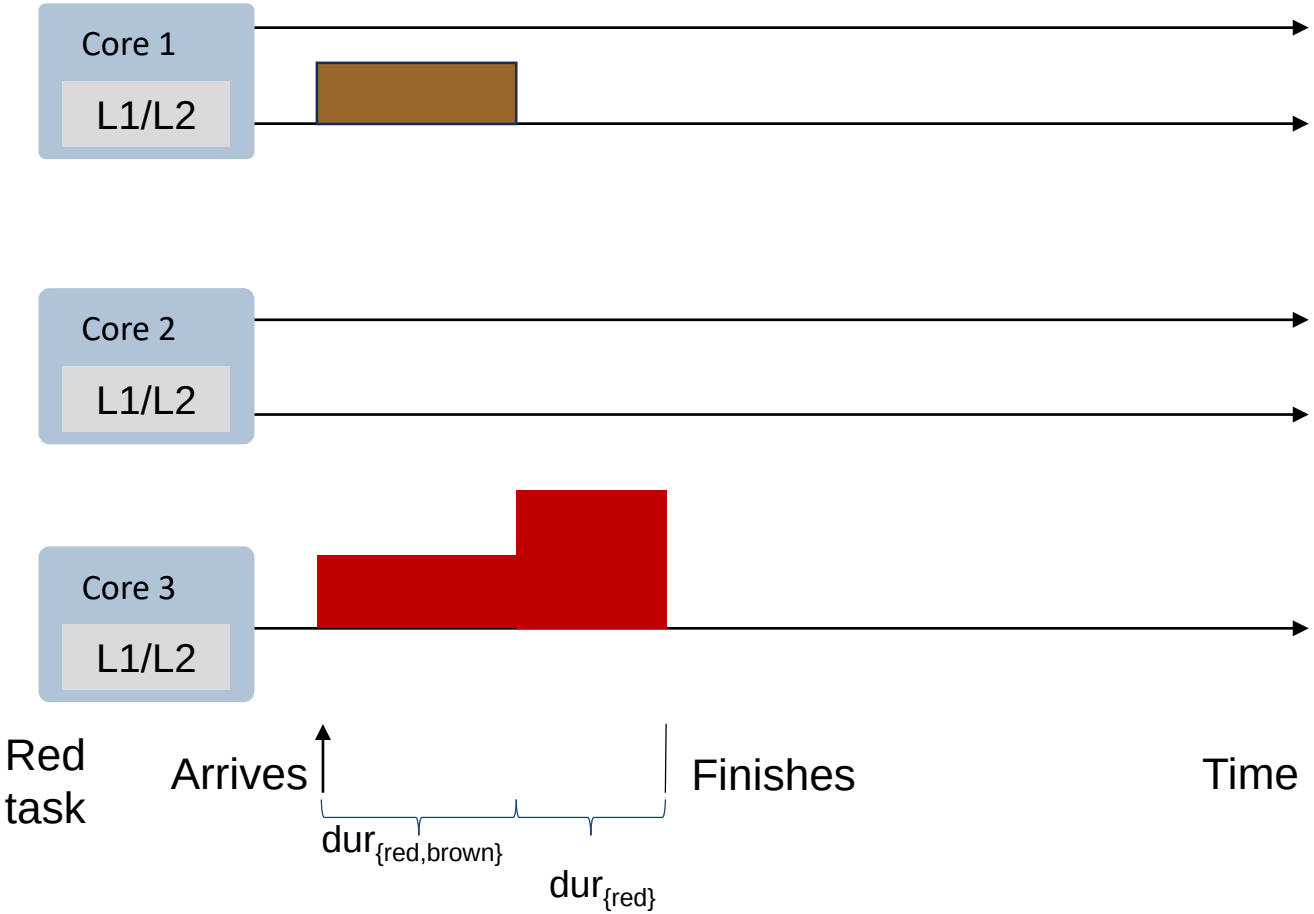
Let $dur_{\{red\}}$ denote the cumulative duration that the set of tasks that execute is the set {red}.





Let $dur_{\{red\}}$ denote the cumulative duration that the set of tasks that execute is the set {red}.

Let $dur_{\{red,brown\}}$ denote the cumulative duration that the set of tasks that execute is the set {red,brown}.



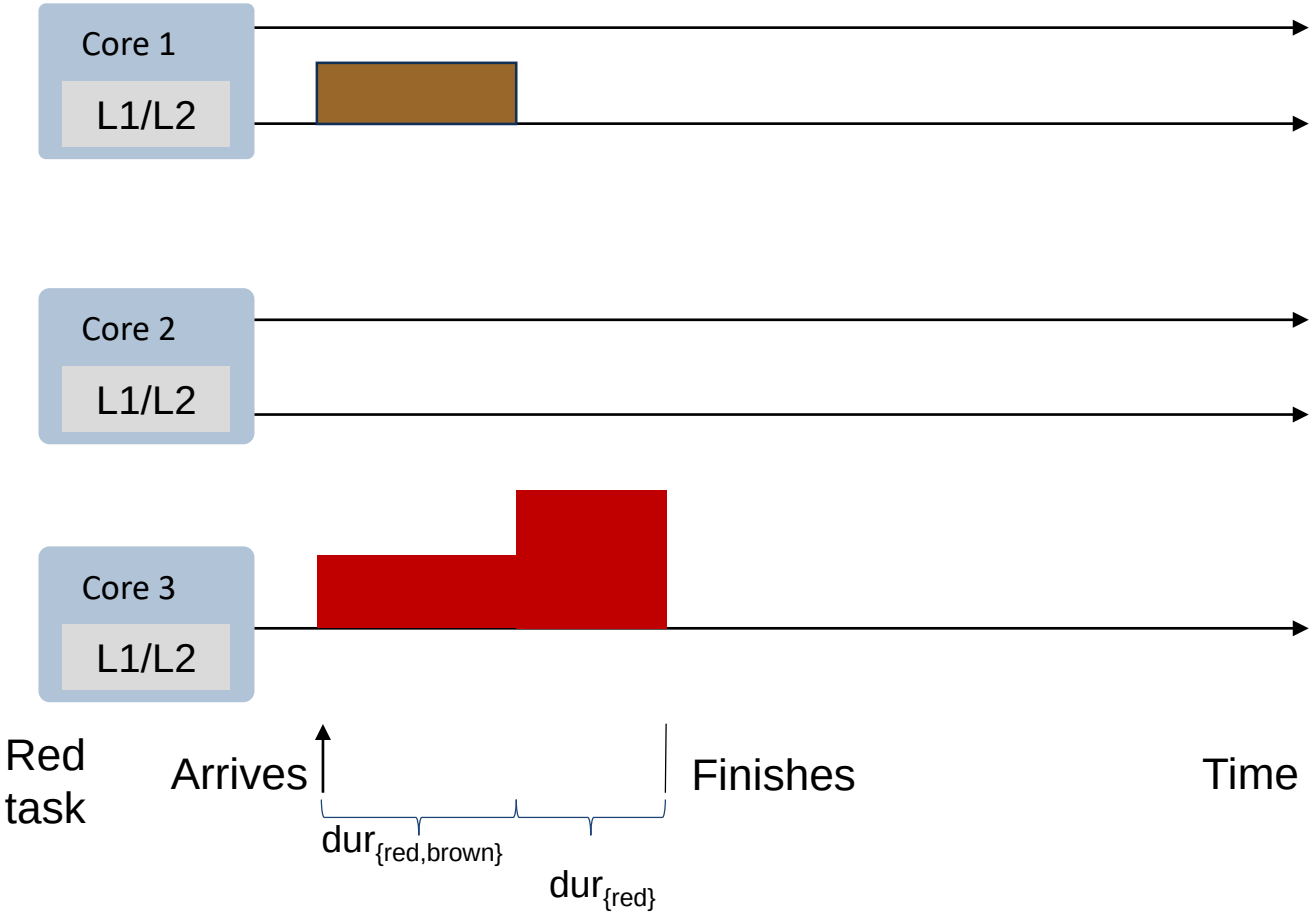
Maximize

$dur_{\{red,brown\}} + dur_{\{red\}}$

subject to

$0.45*dur_{\{red,brown\}} + 1.00*dur_{\{red\}} \leq C_{red}$

$0.6*dur_{\{red,brown\}} + 1.0*dur_{\{brown\}} \leq C_{brown}$



Maximize

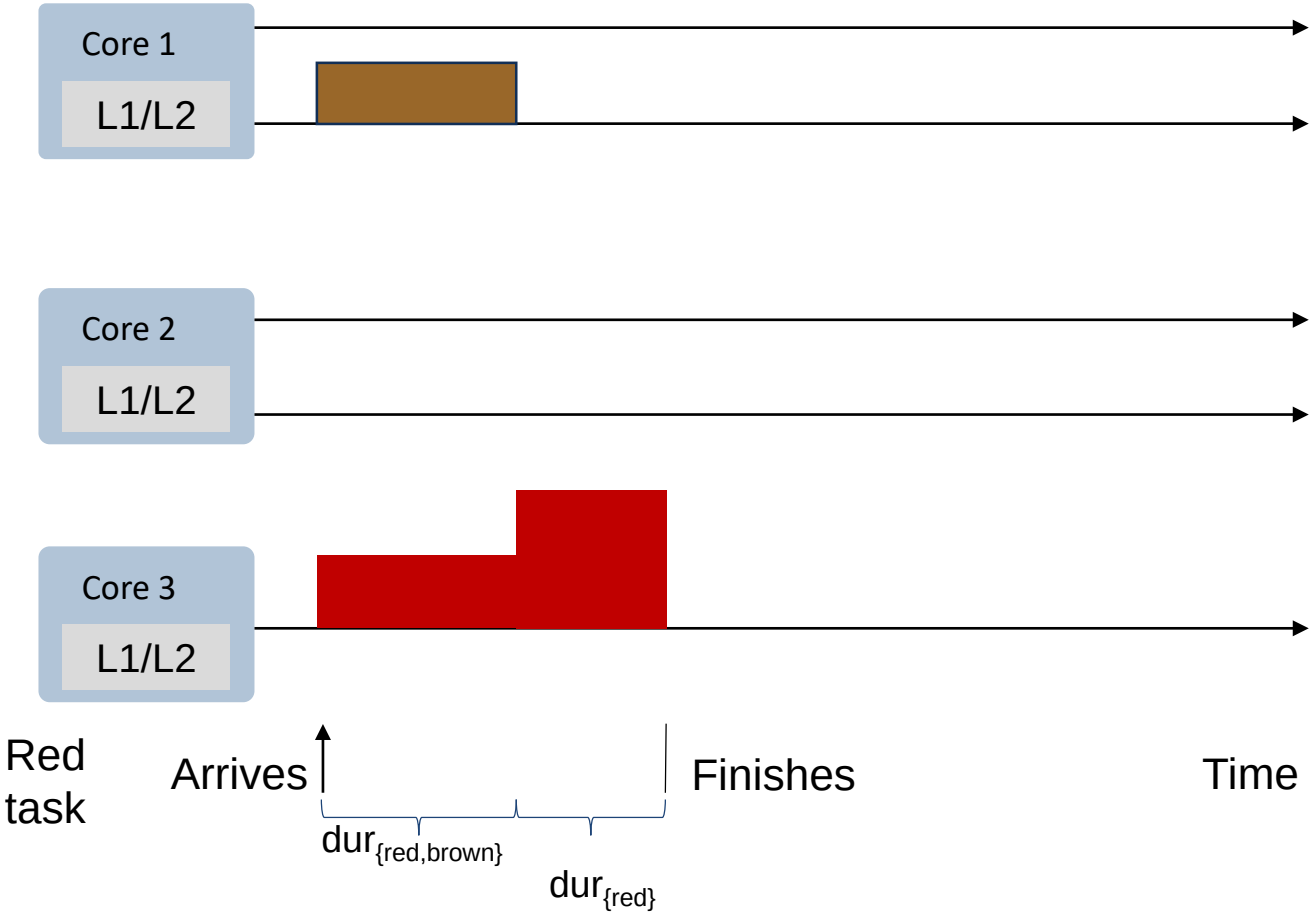
$dur_{\{red,brown\}} + dur_{\{red\}}$

subject to

$0.45*dur_{\{red,brown\}} + 1.00*dur_{\{red\}} \leq C_{red}$

$0.6*dur_{\{red,brown\}} + 1.0*dur_{\{brown\}} \leq C_{brown}$

Lower bound on the speed of execution of Red when it executes in parallel with Brown.



Maximize

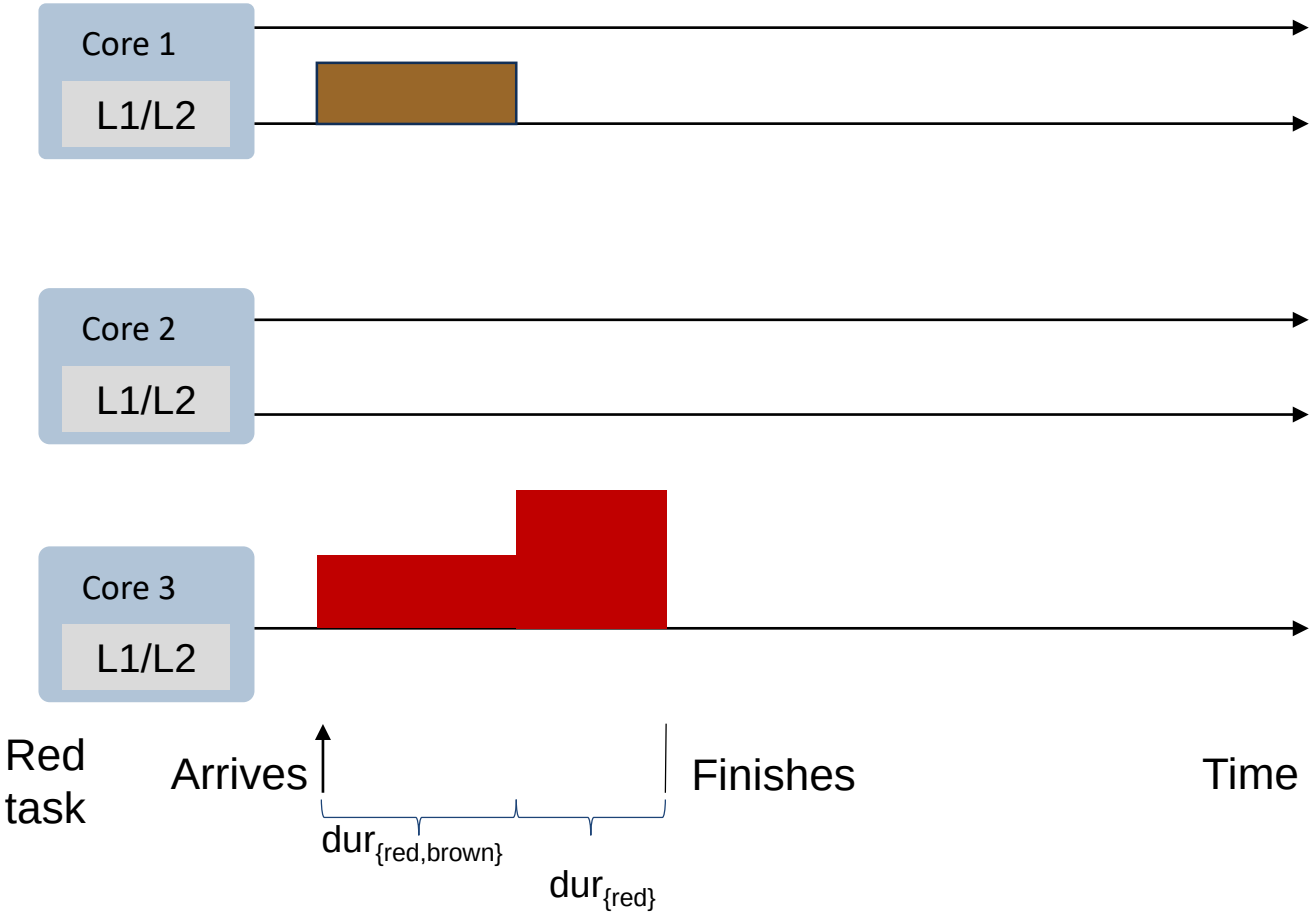
$dur_{\{red,brown\}} + dur_{\{red\}}$

subject to

$0.45*dur_{\{red,brown\}} + 1.00*dur_{\{red\}} \leq C_{red}$

$0.6*dur_{\{red,brown\}} + 1.0*dur_{\{brown\}} \leq C_{brown}$

Lower bound on the speed of execution of Brown when it executes in parallel with Red.



Maximize

$dur_{\{red,brown\}} + dur_{\{red\}}$

subject to

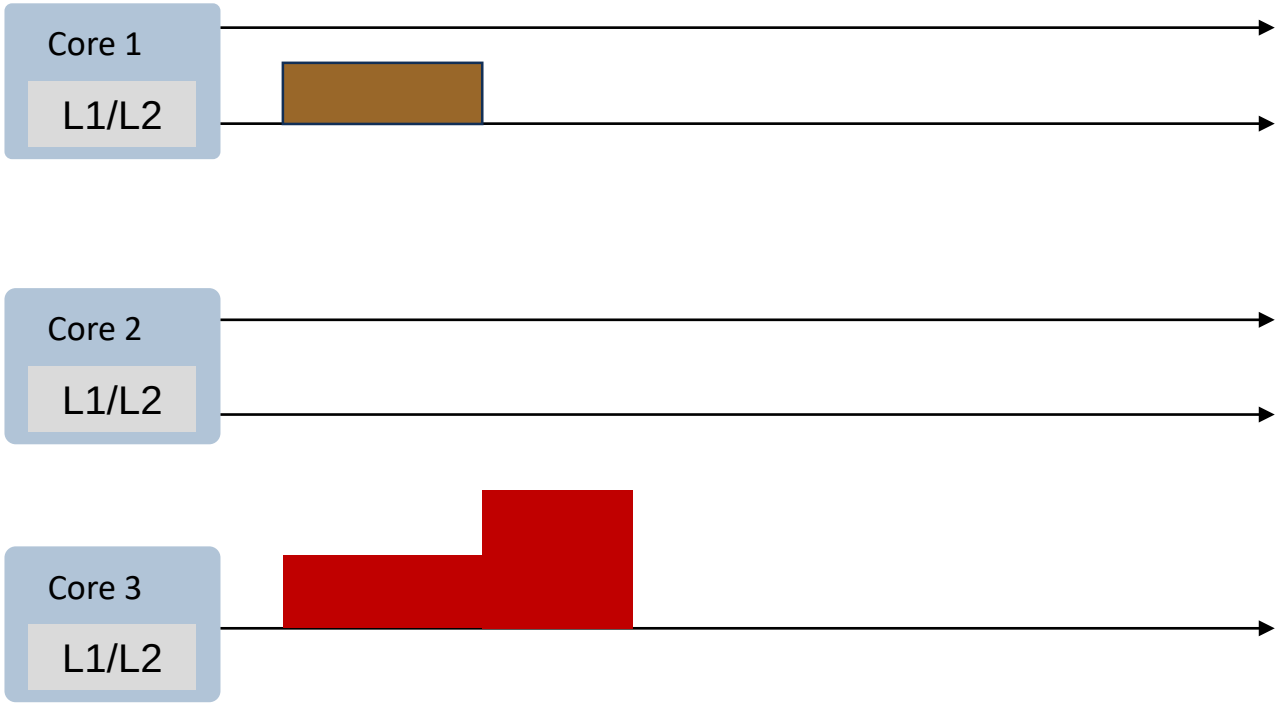
$0.45*dur_{\{red,brown\}} + 1.00*dur_{\{red\}} \leq C_{red}$

$0.6*dur_{\{red,brown\}} + 1.0*dur_{\{brown\}} \leq C_{brown}$

$0 \leq dur_{\{red,brown\}}$

$0 \leq dur_{\{red\}}$

$0 \leq dur_{\{brown\}}$



Maximize

$dur_{\{red,brown\}} + dur_{\{red\}}$

subject to

$0.45*dur_{\{red,brown\}} + 1.00*dur_{\{red\}} \leq C_{red}$

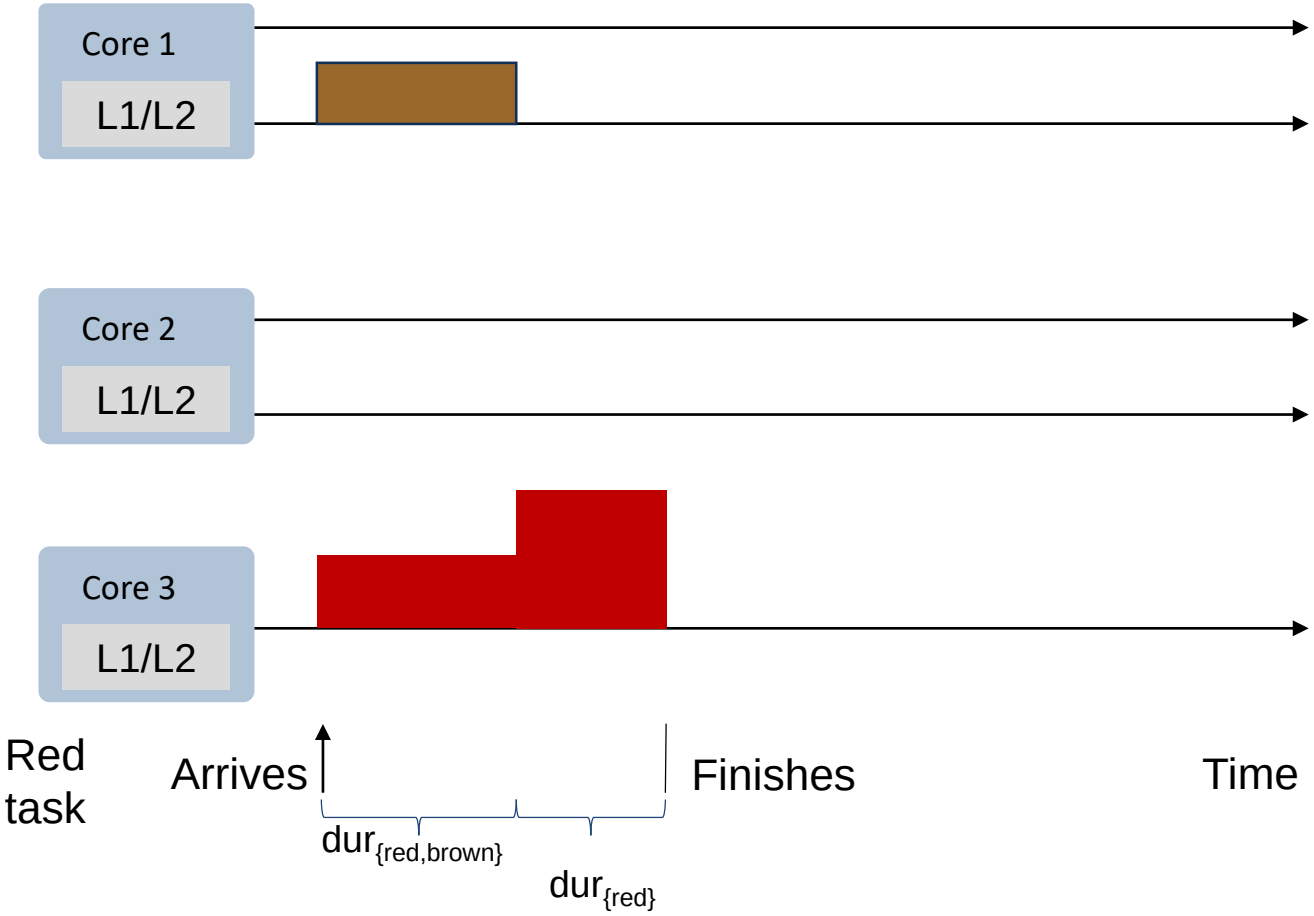
$0.6*dur_{\{red,brown\}} + 1.0*dur_{\{brown\}} \leq C_{brown}$

$0 \leq dur_{\{red,brown\}}$

$0 \leq dur_{\{red\}}$

$0 \leq dur_{\{brown\}}$

Solving this optimization problem yields an upper bound on the response time of Red.



Maximize

$dur_{\{red,brown\}} + dur_{\{red\}}$

subject to

$0.45*dur_{\{red,brown\}} + 1.00*dur_{\{red\}} \leq C_{red}$

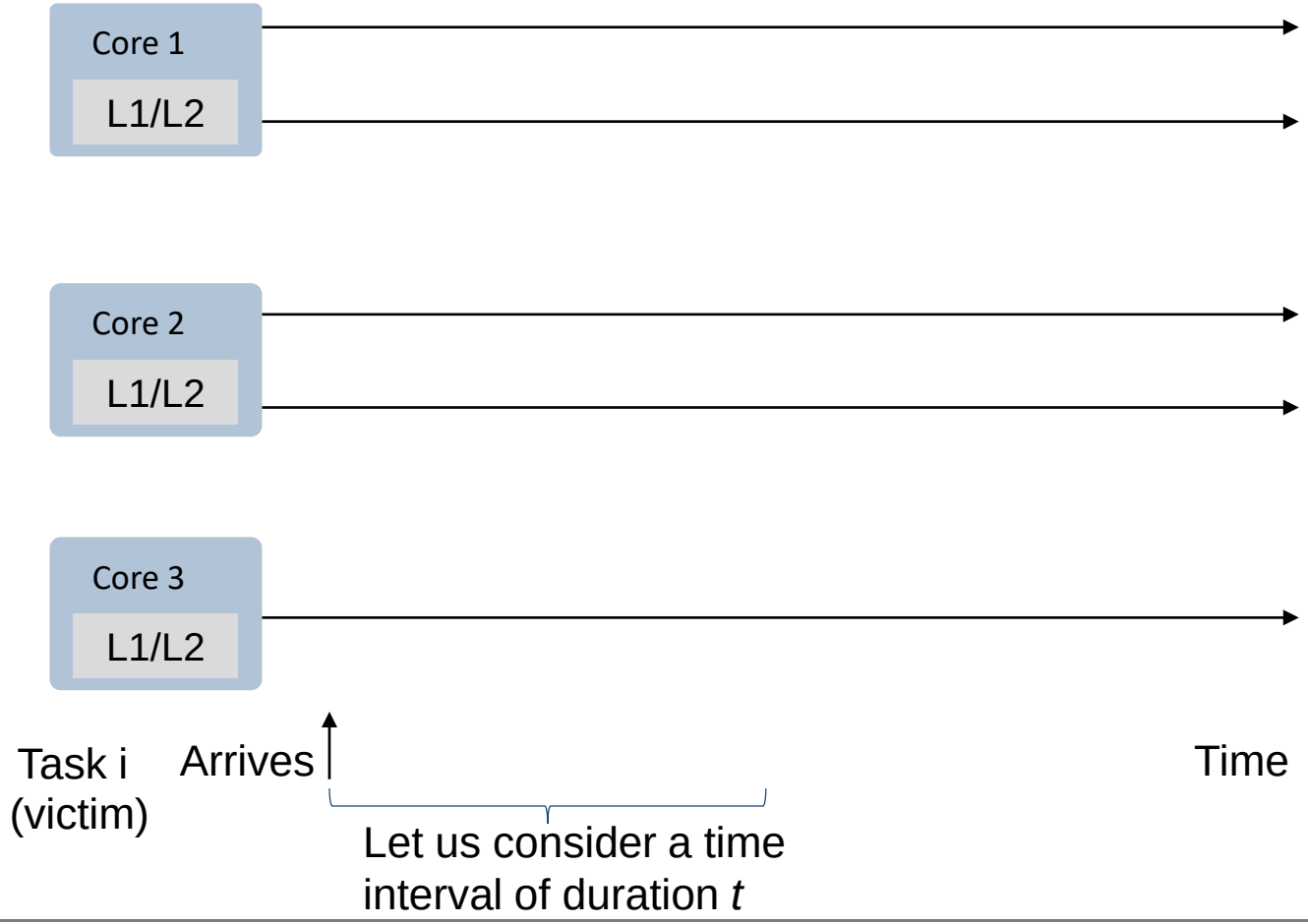
$0.6*dur_{\{red,brown\}} + 1.0*dur_{\{brown\}} \leq C_{brown}$

$0 \leq dur_{\{red,brown\}}$

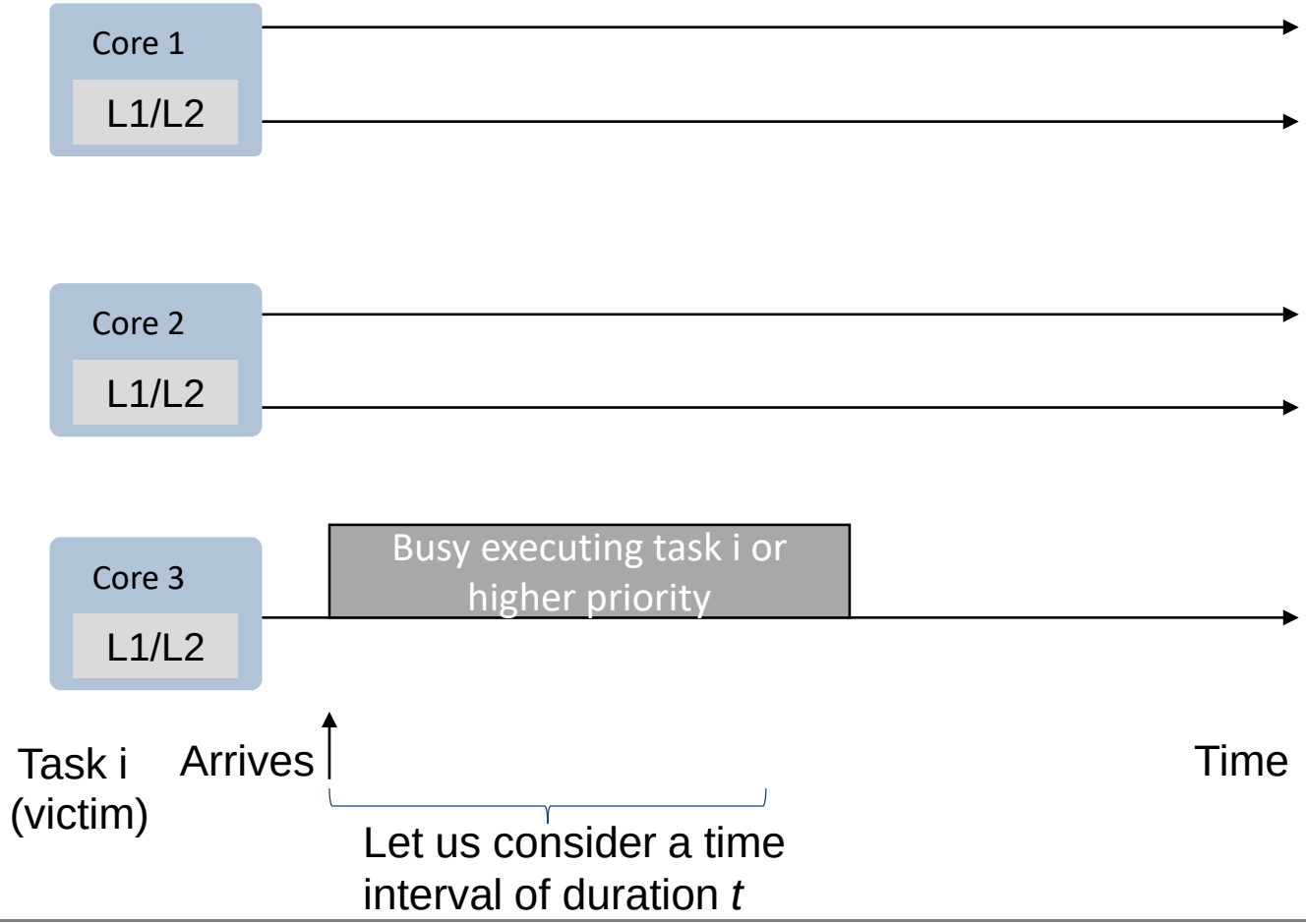
$0 \leq dur_{\{red\}}$

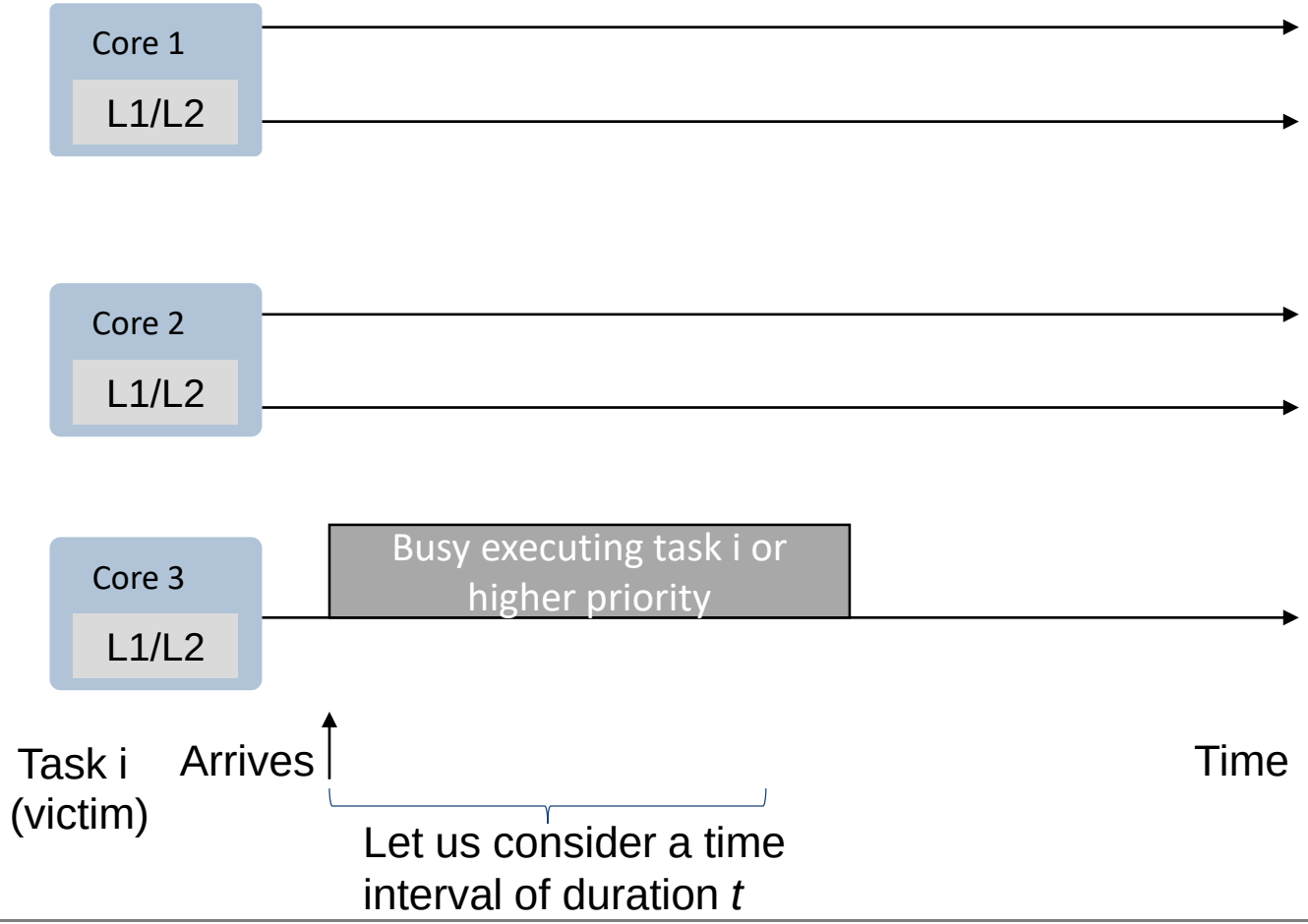
$0 \leq dur_{\{brown\}}$

Let us formulate this in general.



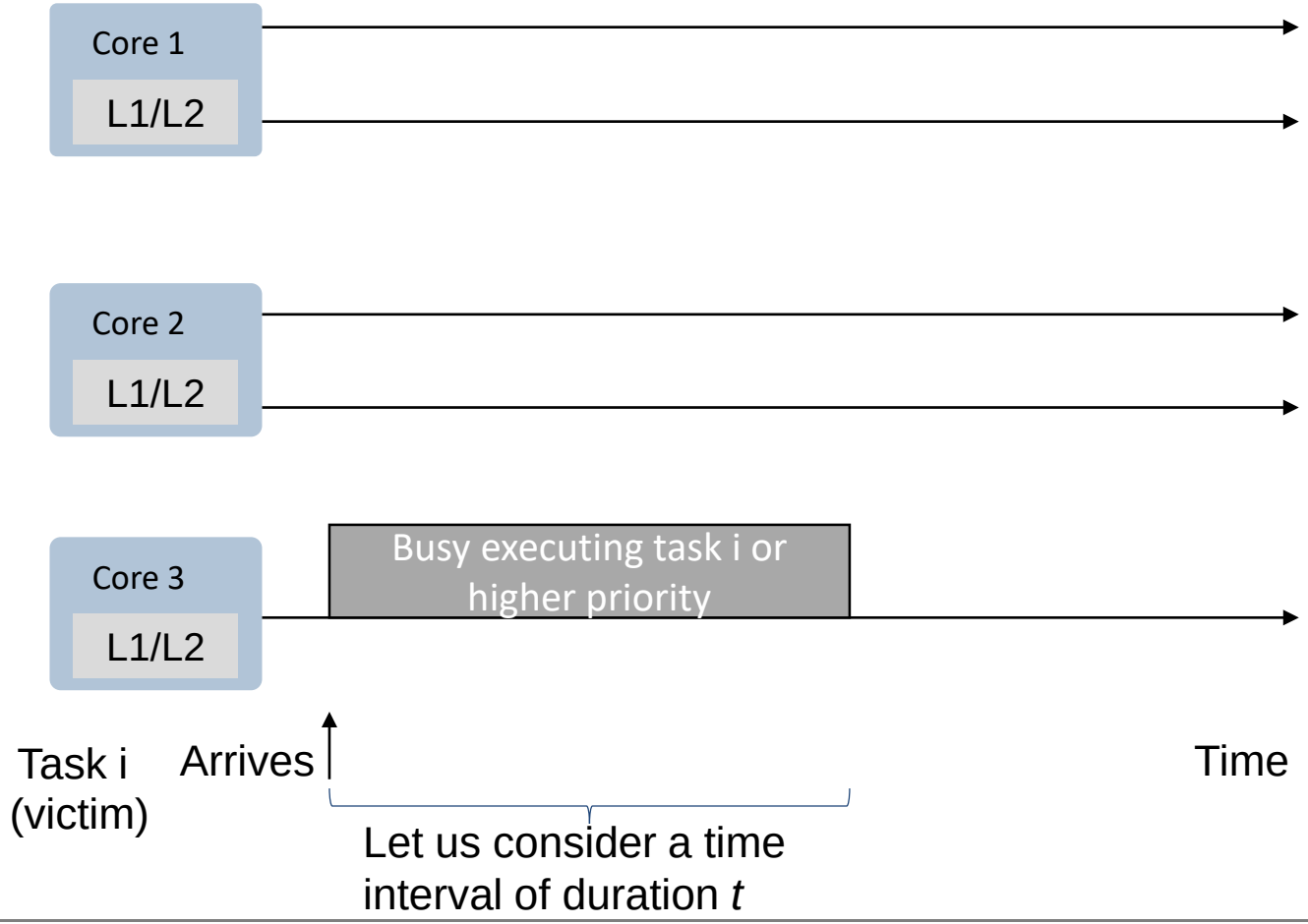
Let τ denote the taskset.
Let $\prod$ denote the computer platform.





Let τ denote the taskset.
Let $\prod$ denote the computer platform.

Given task *i*, find a value of *t* such that an upper bound on the duration of time during which the processor (Processor 3) is busy executing task *i* or higher priority tasks is equal to *t*.



Let τ denote the taskset.
Let $\prod$ denote the computer platform.

Given task *i*, find a value of *t* such that an upper bound on the duration of time during which the processor (Processor 3) is busy executing task *i* or higher priority tasks is equal to *t*.

Define reqlp(τ,Π,i,t) as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau,\Pi,i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau,\Pi,i',k')} \text{du}_s$$

subject to

$$\forall i' \in \text{hep}(\tau,\Pi,i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau,\Pi,i',k')} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \text{xUB}(\tau,\Pi,i',k',t)$$

$$\forall s \in S(\tau,\Pi) \text{ s.t. } (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s)), \text{du}_s \in \mathbb{R}_{\geq 0}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau,\Pi,i,t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define reqlp(τ,Π,i,t) as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau,\Pi,i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau,\Pi,i',k')} \text{du}_s$$

subject to

$$\forall i' \in \text{hep}(\tau,\Pi,i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau,\Pi,i',k')} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \text{xUB}(\tau,\Pi,i',k',t)$$

$$\forall s \in S(\tau,\Pi) \text{ s.t. } (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s)), \text{du}_s \in \mathbb{R}_{\geq 0}$$

For a given taskset, for a given *t*, this is a linear program.

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau,\Pi,i,t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define reqlp(τ,Π,*i*,*t*) as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau,\Pi,i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau,\Pi,i',k')} \text{du}_s$$

subject to

$$\forall i' \in \text{hep}(\tau,\Pi,i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau,\Pi,i',k')} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \left\lceil \frac{t}{T_{i'}} \right\rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \text{xUB}(\tau,\Pi,i',k',t)$$

$$\forall s \in S(\tau,\Pi) \text{ s.t. } (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s)), \text{du}_s \in \mathbb{R}_{\geq 0}$$

For a given taskset, for a given *t*, this is a linear program because these become constants.

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau,\Pi,i,t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define reqlp(τ,Π,i,t) as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau,\Pi,i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau,\Pi,i',k')} \text{du}_s$$

subject to

$$\forall i' \in \text{hep}(\tau,\Pi,i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau,\Pi,i',k')} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \text{xUB}(\tau,\Pi,i',k',t)$$

$$\forall s \in S(\tau,\Pi) \text{ s.t. } (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s)), \text{du}_s \in \mathbb{R}_{\geq 0}$$

A linear program can be solved in polynomial time.

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau,\Pi,i,t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define reqlp(τ,Π,*i*,*t*) as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau,\Pi,i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau,\Pi,i',k')} \text{du}_s$$

subject to

$$\forall i' \in \text{hep}(\tau,\Pi,i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau,\Pi,i',k')} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \text{xUB}(\tau,\Pi,i',k',t)$$

$$\forall s \in S(\tau,\Pi) \text{ s.t. } (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s)), \text{du}_s \in \mathbb{R}_{\geq 0}$$

Given τ, Π, *i*, *t*, evaluating reqlp(τ,Π,*i*,*t*) can be done in polynomial time.

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau,\Pi,i,t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define reqlp(τ,Π,i,t) as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau,\Pi,i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau,\Pi,i',k')} \text{du}_s$$

subject to

$$\forall i' \in \text{hep}(\tau,\Pi,i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau,\Pi,i',k')} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \text{xUB}(\tau,\Pi,i',k',t)$$

$$\forall s \in S(\tau,\Pi) \text{ s.t. } (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s)), \text{du}_s \in \mathbb{R}_{\geq 0}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $\boxed{(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau,\Pi,i,t) \leq t))} \Rightarrow \tau$ is schedulable on Π

Evaluating this can be done in pseudo-polynomial time.

Define reqlp(τ,Π,i,t) as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau,\Pi,i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau,\Pi,i',k')} \text{du}_s$$

subject to

$$\forall i' \in \text{hep}(\tau,\Pi,i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau,\Pi,i',k')} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \text{xUB}(\tau,\Pi,i',k',t)$$

$$\forall s \in S(\tau,\Pi) \text{ s.t. } (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s)), \text{du}_s \in \mathbb{R}_{\geq 0}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau,\Pi,i,t) \leq t)) \Rightarrow \tau$ is schedulable on Π

This is a generalization of the classic response-time analysis for Rate-Monotonic.

Define reqlp(τ,Π,i,t) as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau,\Pi,i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau,\Pi,i',k')} \text{du}_s$$

subject to

$$\forall i' \in \text{hep}(\tau,\Pi,i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau,\Pi,i',k')} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \text{xUB}(\tau,\Pi,i',k',t)$$

$$\forall s \in S(\tau,\Pi) \text{ s.t. } (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s)), \text{du}_s \in \mathbb{R}_{\geq 0}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau,\Pi,i,t) \leq t)) \Rightarrow \tau$ is schedulable on Π

For details (about this theorem and other results about the model), see

B. Andersson et al., ``Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times,'' ACM TECS, 2018.

Define reqlp(τ,Π,i,t) as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau,\Pi,i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau,\Pi,i',k')} \text{du}_s$$

subject to

$$\forall i' \in \text{hep}(\tau,\Pi,i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau,\Pi,i',k')} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'} \times \text{du}_s \leq \text{xUB}(\tau,\Pi,i',k',t)$$

$$\forall s \in S(\tau,\Pi) \text{ s.t. } (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s)), \text{du}_s \in \mathbb{R}_{\geq 0}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau,\Pi,i,t) \leq t)) \Rightarrow \tau$ is schedulable on Π

We have a schedulability analysis for tasks with co-runner dependent execution times.

Comparing this result to classic Rate-Monotonic Analysis

Work	Model (Execution Time Depends on Corunners)	Time Complexity of Exact Schedulability Analysis	Formulation of Schedulability Test	Time Complexity of Schedulability Test
Previous work	No	≤ Pseudo-polynomial	$(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \sum_{j \in \text{hep}(\tau, \Pi, i)} \lceil \frac{t}{T_j} \rceil C_j \leq t))$ $\Leftrightarrow$ schedulable	Pseudo-polynomial
This article	Yes	Co-NP-hard in the strong sense	$(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t))$ $\Rightarrow$ schedulable	Pseudo-polynomial if the number of processors in Π is fixed

This result generalizes classic Rate-Monotonic Analysis to undocumented multicore

Work	Model (Execution Time Depends on Corunners)	Time Complexity of Exact Schedulability Analysis	Formulation of Schedulability Test	Time Complexity of Schedulability Test
Previous work	No	≤ Pseudo-polynomial	$(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \sum_{j \in \text{hep}(\tau, \Pi, i)} \lceil \frac{t}{T_j} \rceil C_j \leq t))$ $\Leftrightarrow$ schedulable	Pseudo-polynomial
This article	Yes	Co-NP-hard in the strong sense	$(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t))$ $\Rightarrow$ schedulable	Pseudo-polynomial if the number of processors in Π is fixed

This looks very theoretical. Does this really work in reality?

I understand that for each task, the set of co-runner sets is polynomial in the number of tasks assuming that the number of processors is fixed.

But I still wonder how one can obtain the lower bound of speed of execution of one task given co-runners.

A measurement-based approach:

5.2 Obtaining Taskset Parameters for a Given Software System

Every scheduling theory that provides 100% guarantees on timing relies on a model. Traditional scheduling theory relies on knowledge of an upper bound on the execution time of a program, and therefore, the research community has developed methods for finding an upper bound on the execution requirement of a program when the program runs in isolation on a single-core processor (Wilhelm et al. 2008). One can distinguish between (1) *static methods* that take the program as input and compute an upper bound without running the program, (2) *hybrid methods* that measure execution times of parts of a program and then use static methods to compute an upper bound for the entire program, and (3) *dynamic methods* where a software practitioner uses domain knowledge to identify a set of worst-case inputs/initial states of the program and then measures the execution time of the program for these inputs/initial states; sometimes the program is run multiple times with variations of inputs/initial state and sometimes a safety margin is added based on engineering judgment. The dynamic method is in common use in industry today.

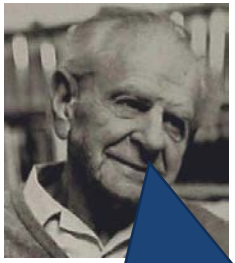
The model used in this article, however, also requires that a task/segment is described with a lower bound on its speed as a function of corunners. Unfortunately, the current state of the art does not offer any method for obtaining such a lower bound while considering all the complexities of modern memory systems. Therefore, in order to use our scheduling theory (in practice and also in our case study/validation in Section 5.3), we need to develop a method for obtaining such a lower bound. We will do so by using method (3) mentioned above but extending it so that it also provides a lower bound on its speed as a function of corunners. The new method is shown below as pseudo-code:

(1) Let SS be an integer indicating the sample size used in our experimental method. Assign a value to SS (for example SS := 100).
(2) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, do
  (a) Run a job of task τ_i on the computer platform s.t. no other tasks execute on this platform.
  (b) Repeat the measurement SS times and let CM_i^k denote the set of these measurements (execution times from measurements of a task executed in isolation).
(3) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, do $C_i^k := \max_{x \in \mathrm{CM}_i^k} x$.
(4) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, do
  (a) Find an upper bound on the number of memory accesses performed by a segment of a job of task τ_i. (This can be obtained using performance monitoring counters.) Run these measurements SS times. Let H_i^k denote the largest measurement.
  (b) Find an upper bound on the time for a single memory access assuming that we do not know the corunners. Let MA denote this (e.g., MA could be 500 nanoseconds).
  (c) $\mathrm{pd}_i^k := \max_{x \in \mathrm{CM}_i^k} \frac{x}{x + H_i^k \times \mathrm{MA}}$.
(5) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, do $\mathrm{CO}_i^k := \emptyset$.
(6) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, for each $s \in S(\tau, \Pi, i, k)$, do
  (a) $\mathrm{co} := s \setminus \{(i,k)\}$.
  (b) For those segments in co, keep running all of them continuously (i.e., if segment $v_{i'}^{k'}$ is in co, then when $v_{i'}^{k'}$ has finished execution, let this segment $v_{i'}^{k'}$ start execution again immediately).
  (c) While the segments in (b) execute continuously, execute a single job of segment v_i^k of task τ_i and measure the execution time of τ_i^k. Repeat this measurement SS times and let $\mathrm{CMC}_{i,\mathrm{co}}^k$ denote the set of measurements (execution times from measurements of a task executed with co-runners).

ACM Transactions on Embedded Computing Systems, Vol. 17, No. 3, Article 71. Publication date: May 2018.

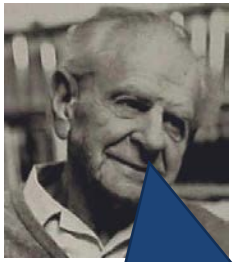
OK, I buy that you can obtain these lower bounds on the speed of execution through measurements. But I wonder how trustworthy these numbers are. Have you done any validation?

OK, I buy that you can obtain these lower bounds on the speed of execution through measurements. But I wonder how trustworthy these numbers are. Have you done any validation?



You could do a case study. Choose a set of tasks and use your measurement-based approach to obtain speed as a function of co-runners. Now, you have a taskset. Then run your schedulability analysis on this taskset.

OK, I buy that you can obtain these lower bounds on the speed of execution through measurements. But I wonder how trustworthy these numbers are. Have you done any validation?



Then run the system and measure response times of each task. If you observe one case where the observed response time > calculated upper bound on response time, then you have falsified your theory; otherwise, you have corroborated your theory.

5.3 Validation of Model and Schedulability Analysis

We want to compare the computed upper bound on the response time of a task against the measured response times of jobs of this task in order to find out if there is any case in practice where the measured response time of this task exceeds the upper bound on the response time of this task. If so, the schedulability analysis would be unsafe. To get insight into this question, we pursue a case study with a specific taskset and a real embedded platform (ARM Cortex-A9 quad-core processor).

To avoid cache-related preemption delay, we used the software cache partitioning implementation of Linux/RK (Kim et al. 2013)[3] and assigned a private cache partition to each task.

We constructed a taskset as follows. First, we created six synthetic programs with different intensities of memory accesses on the target platform. We call each such program a *building block*. There is no shared variable between building blocks. We characterize the execution time and corunner interference using the method in the previous section with SS := 100. The corunner description is complete; that is, the building block is characterized for each possible set of corunners.

Then we create tasks from these building blocks as follows: Task τ_1 consists of executing building block 1. Hence, we model task τ_1 as having a single segment. Task τ_2 consists of executing building block 2 and then executing building block 3. Hence, we model task τ_2 as having two segments. Task τ_3 consists of executing building block 4. Hence, we model task τ_3 as having a single segment. Task τ_4 consists of executing building block 5. Hence, we model task τ_4 as having a single segment. Task τ_5 consists of executing building block 6. Hence, we model task τ_5 as having a single segment. Task τ_6 consists of executing building block 2 and then executing building block 6. Hence, we model task τ_6 as having two segments.

We assign tasks to processors and assign priorities to tasks and assign T and D parameters and obtain the execution time and corunner interference parameters from the building blocks. This gives us the taskset specified in Table 5. The CO-parameters are not shown because they consume lots of space; these parameters are available at the end of the source-code file of the tool that performs schedulability testing.

Our tool yields the following upper bounds on response times of task: 0.113 for τ_1, 0.554 for τ_2, 0.132 for τ_3, 0.153 for τ_4, 0.146 for τ_5, and 0.407 for τ_6 (units in seconds).

We run the system for 1,000 seconds and measure the response times of jobs. For task τ_i, we record the maximum response time of a job of this task and let r_i denote it. From the experiment, we obtain $r_1 = 0.107$, $r_2 = 0.483$, $r_3 = 0.127$, $r_4 = 0.137$, $r_5 = 0.142$, $r_6 = 0.365$ (units in seconds). With these figures, we obtain that the response time overestimates the observed response times by less than 15%. From this experiment, we see that (1) no deadline was missed and (3) the observed response times were close to the computed upper bound.

From B. Andersson et al., ``Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times,'' ACM TECS, 2018:

> With these figures, we obtain that the response time overestimates the observed response times by less than 15%. From this experiment, we see that (1) no deadline was missed and (3) the observed response times were close to the computed upper bound.

Conclusion: It is possible to analyze timing of software executing on undocumented multicore.

Thanks!