



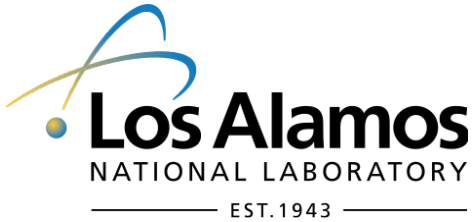
# Managing the Performance of Large, Distributed Systems

**Scott A. Brandt**
**University of California, Santa Cruz**

# Collaborators

- Carlos Maltzahn, Shinpei Kato, Kleoni Iouannidou, Roberto Pineiro, and David Bigelow, Dimitrios Skourtis (University of California Santa Cruz)
- Richard Golding (Kinsey Technical Services (w/DARPA))
- Anna Povzner, Tim Kaldewey, and Ted Wong (IBM Almaden Research Center)
- Andrew Shewmaker (Los Alamos National Laboratory)
- Darren Sawyer (NetApp)






# Who am I?

- **Professor**, Computer Science Department, UC Santa Cruz
- **Director**, UCSC/LANL Institute for Scalable Scientific Data Management (ISSDM)
- **Co-Director**, UCSC Systems Research Laboratory (SRL)
- Background
  - 1999 Ph.D. CS, Colorado
  - 1987/1993 B. Math/MS CS, Minnesota
  - 1982-1994 Programmer/Research Scientist/VP, CPT, B-Tree, Honeywell SRC, Theseus Research, Alliant TechSystems RTS, Secure Computing

- My Research
  - High-performance petascale storage and data management
  - Real-time systems
  - Performance management and virtualization
- Other Research
  - Secure operating systems
  - Asynchronous circuits
  - Real-time image processing
- Recent Highlights
  - Ceph in Linux as of 2.6.34
  - Best Papers at SIGMOD, ECRTS, HPDC

UC SANTA CRUZ

ISSDM

# Today: Distributed system performance guarantees

1. Motivation
2. Some philosophy
3. Our approach: a unifying model
4. Solutions for CPU, storage, network, cache
5. Conclusions

# Distributed systems need performance guarantees

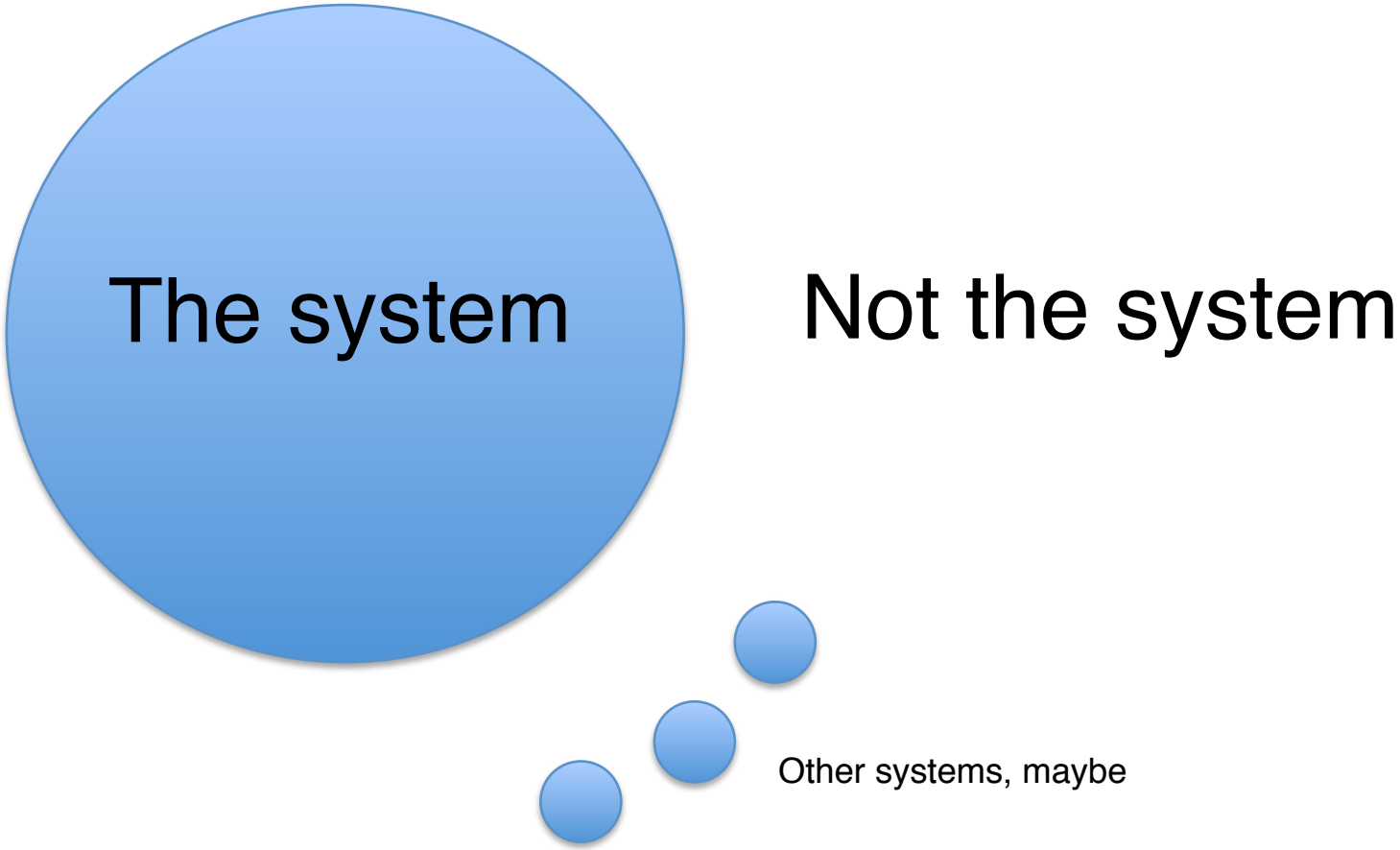


- High-performance scientific simulation and visualization
- Virtual machines
- Cloud services
- Real-time data capture
- Sensor networks
- Smart power grid
- Automotive systems
- Integrated medical systems
- Aerospace systems
- Radio telescopes
- Even so-called best-effort applications

- Distributed systems with many independent, autonomous players

  ... many of which are providing mission- or life-critical operations or control

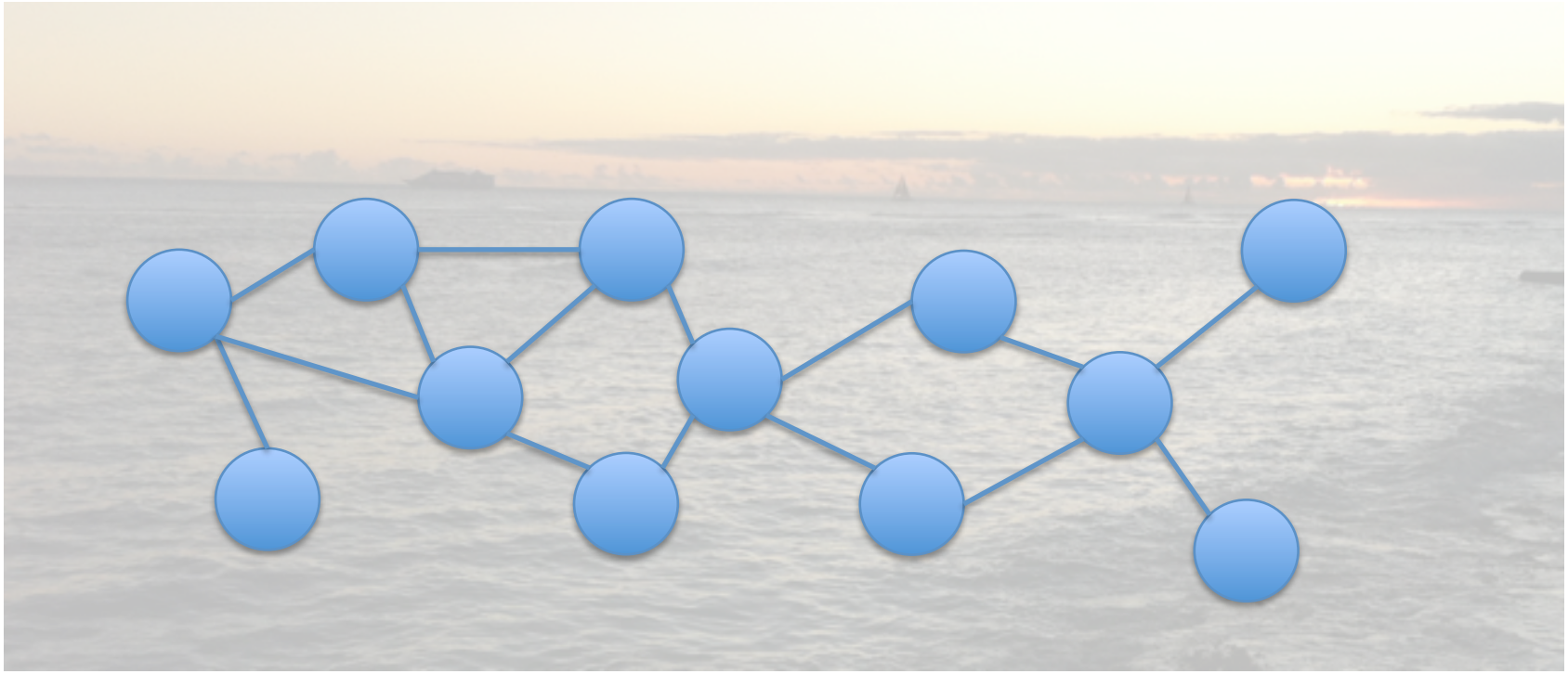
# The model before



# The model now

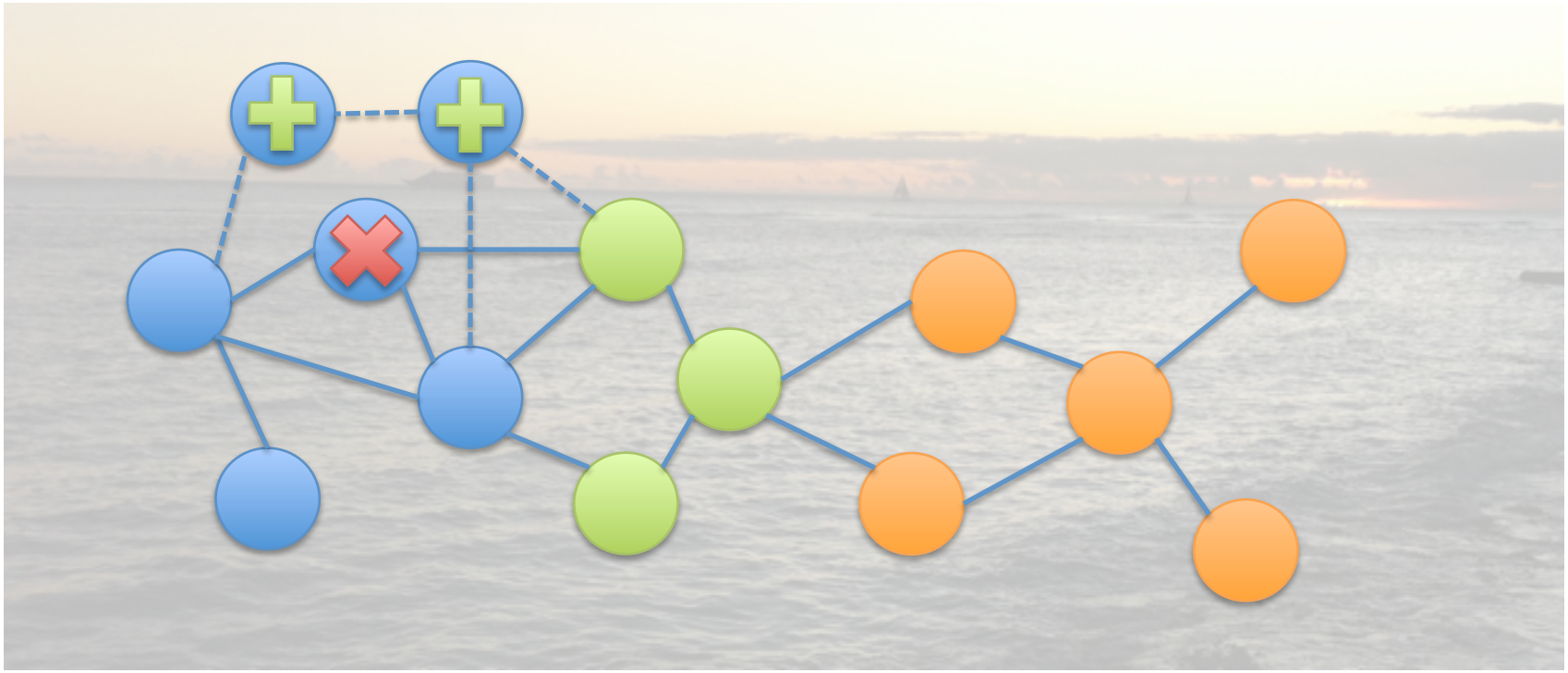

A sea of components—no sharp boundaries
A multipolar/multiuser/coalition world
Long term existence and constant change

# A sea of components



Many pieces, deployed and evolving separately
Where is the boundary of the “application”?

# Long-term existence and constant change



Different pieces get replaced, fail, upgraded independently
Some piece of the system is always in flux

# Example: System F6 (DARPA)

- Modular clustered communication satellite(s)
  - Coordinated navigation
  - Fully cooperative communication functionality

- Need a new capability? Add a new satellite to the cluster.

# Example: Cloud virtualization (Nat'l Labs)



- Provide isolated "virtual" clouds within a larger cloud

# Example: Virtual servers (SAP)

- Provide isolated virtual servers within a cloud

# Example: Supercomputer virtualization (Nat'l Labs)

- Provide guaranteed service within a large-scale distributed supercomputer system

# Example: Virtual machine services (VMware & startup)

- Provide virtual services (e.g., storage) to virtual machines on shared infrastructure

# Challenge: real-time computation



- How do we keep guarantees in such a chaotic world?
- How do we specify what an application needs so that it can be interpreted for different distributed hardware resources?
- How do we know that a mixture of unknown applications from different organizations doing different things is going to work?

# Challenge: real-time communication



- How do we make pieces of the system talk to each other predictably over a chaotic network?
- How do we say what we need in a concise and precise way?
- How do we get it to work on networks that change performance and topology?

# Challenge: Real-time IO/data storage



- How do we make intrinsically unpredictable devices predictable?
- How do we specify requirements in a uniform way?
- How do we maintain requirements on different hardware?

# More challenges

- Interacting resources
- Dynamic workloads
- Interfering workloads (2+2=3)
- Decentralized control
- Non-commensurable metrics

# In a nutshell

- Big distributed systems
  - Serve many unrelated users/jobs
  - Process lots of data
- Current design
  - Use rules of thumb
  - Over-provision
  - Isolate

- Ad hoc performance management leads to *expensive* and *fragile* solutions
- A better system guarantees each application the performance it needs from all components: CPUs, memory, disks, network, ...

# What we want

1. Guaranteed performance

2. Isolation between workloads

3. High performance

# ... in a bit more detail

- Resource management algorithms and architectures capable of providing
  - High performance
  - Arbitrarily hard or soft performance guarantees with
    - Arbitrary resource allocations
    - Arbitrary timing / granularity
  - Complete isolation between workloads
  - All resources: CPU, storage, network, cache

# What we really want

Virtual resources

indistinguishable from "real" resources

with fractional performance

# Isolation is key

- CPU
  - 20% of a 3 Ghz CPU should be indistinguishable from a 600 Mhz CPU
  - Running: compiler, editor, audio, video
- Disk
  - 20% of a disk with 100 MB/second bandwidth should be indistinguishable from a disk with 20 MB/second bandwidth
  - Serving: 1 stream, *n* streams, sequential, random

# Scott's epistemology of virtualization

- Virtual Machines and LUNs provide good HW virtualization
- Question: Given perfect HW virtualization, *how can a process tell the difference between a virtual resource and a real resource?*
- Answer: By not getting its **share** of the resource **when** it needs it

# What we really want (part ii)

- Virtualized performance

- Arbitrarily hard or soft performance guarantees on each resource
  - Hardware-independent
  - Workload-independent
  - Application-independent
  - Commensurable
  - Globally reservable, locally enforceable
  - Changeable

# Our approach

1. Develop a uniform model for performance management

2. Apply it to each resource

3. Integrate the solutions

# Observation

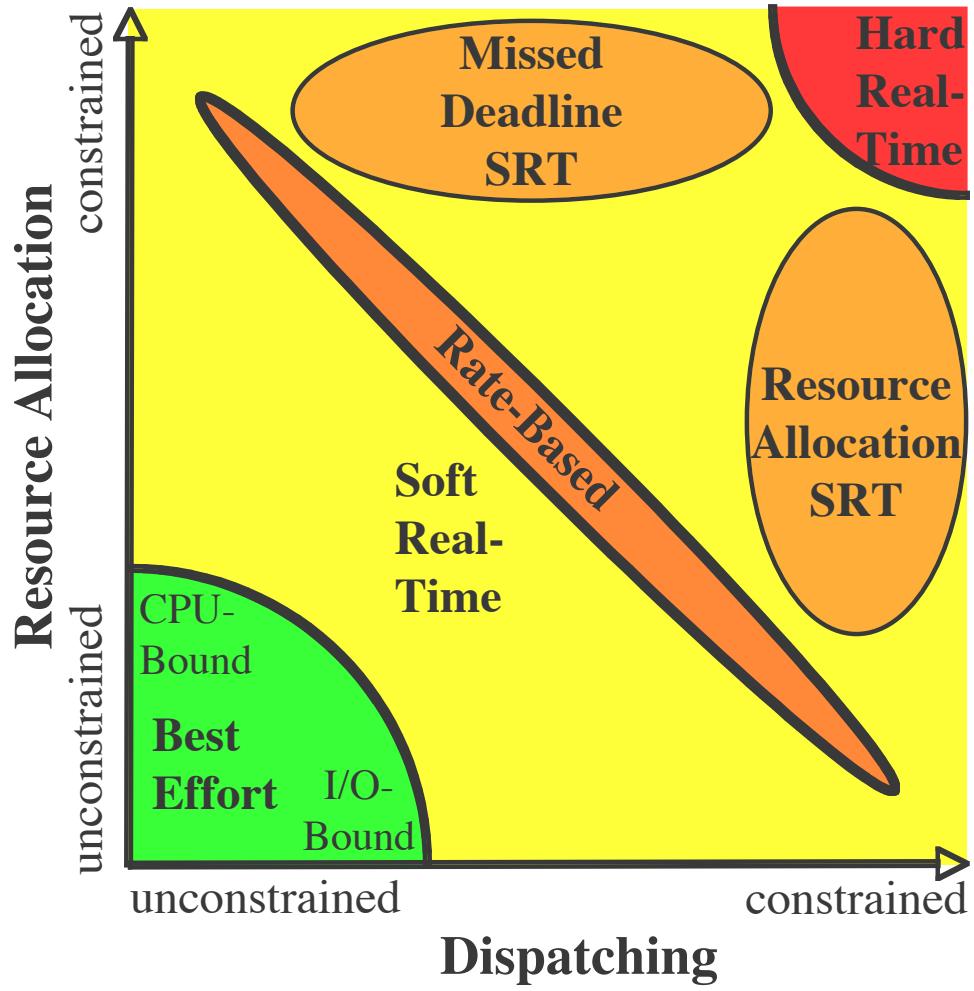
- Resource management consists of two distinct decisions
  - Resource Allocation: *How much* resources to allocate?
  - Dispatching: *When* to provide the allocated resources?

- Most resource managers conflate them
  - Best-effort, proportional-share, real-time

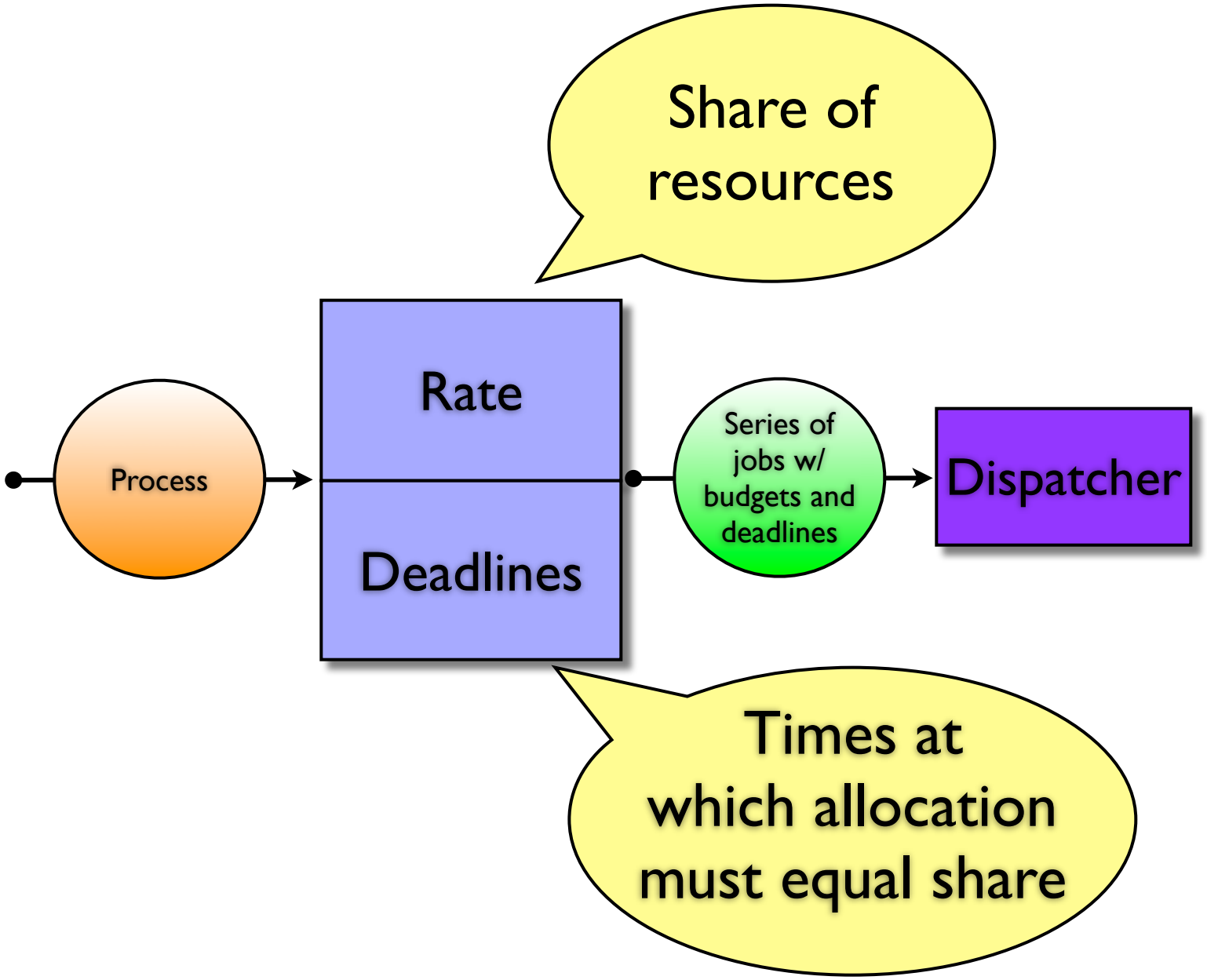
# Separating them is powerful!

- Separately managing resource allocation and dispatching gives direct control over the delivery of resources to tasks

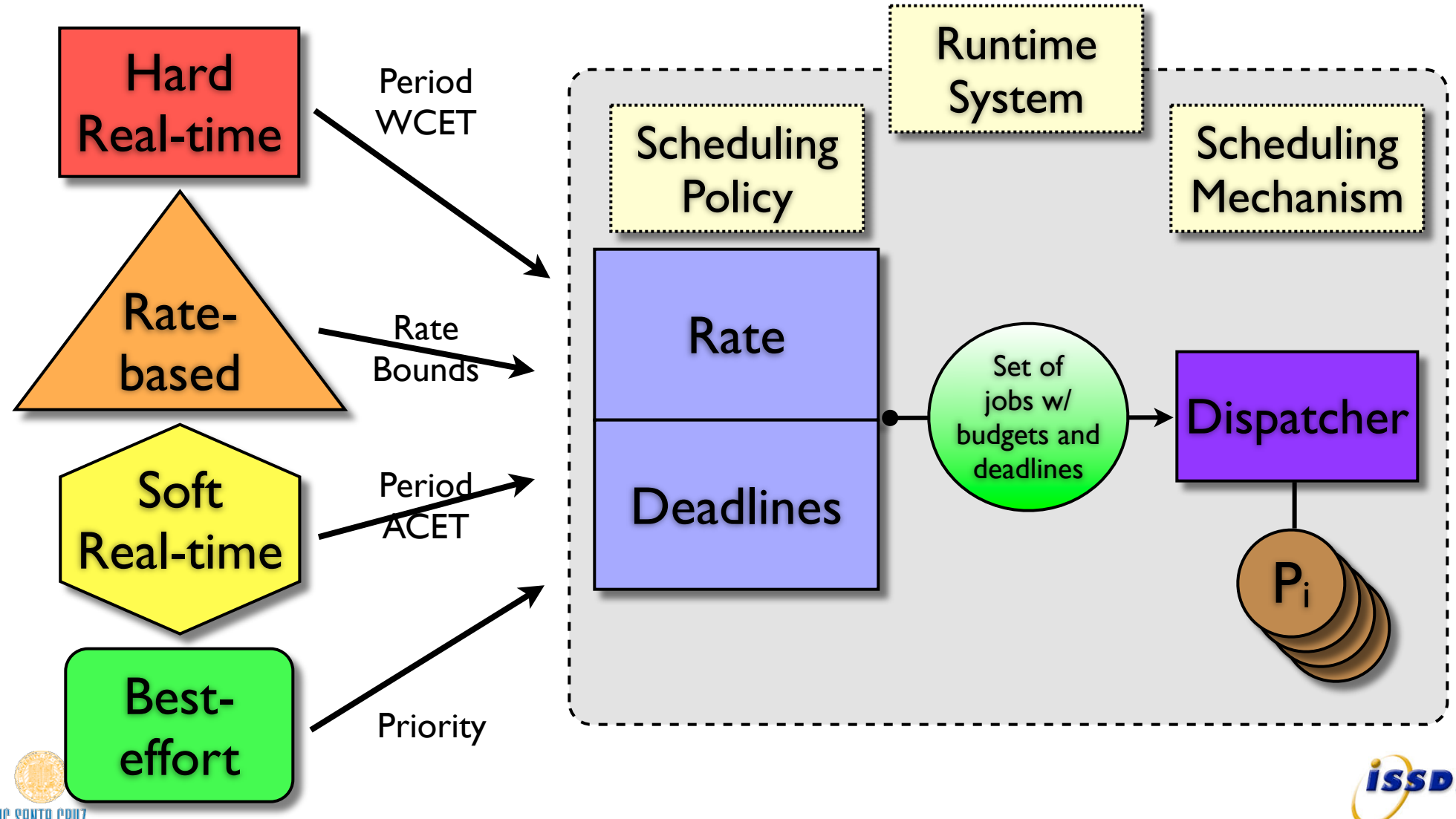
- Enables direct, integrated support of all types of timeliness needs



# The resource allocation/dispatching (RAD) scheduling model

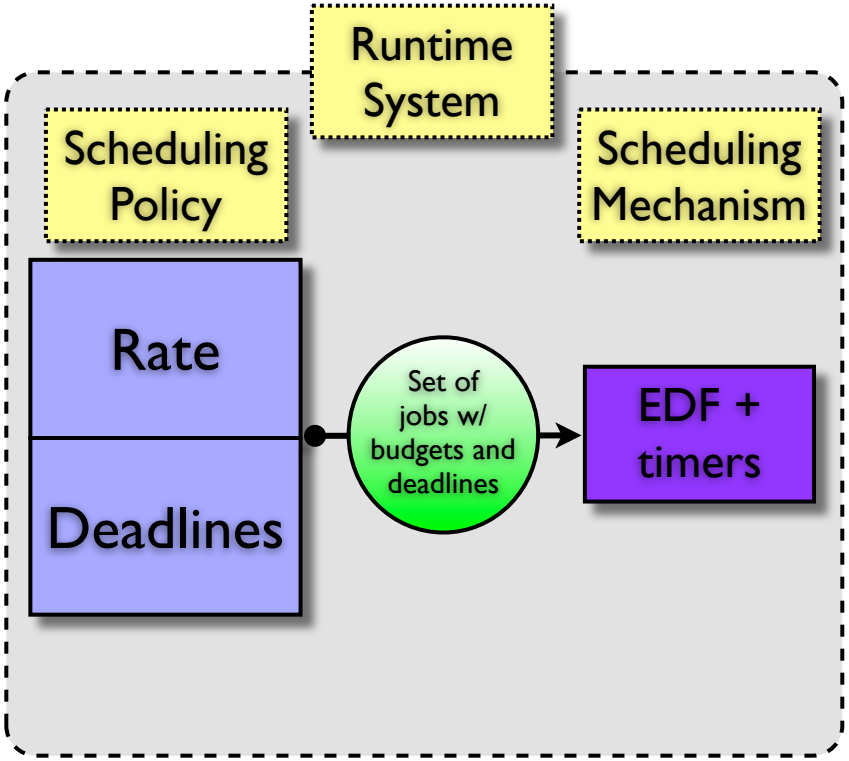


# Supporting different timeliness requirements with RAD



# Rate-Based Earliest Deadline (RBED) CPU scheduler [*RTSS 2003*]

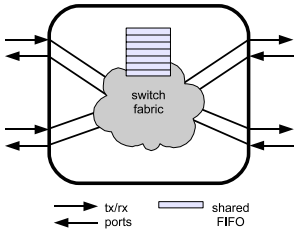
- Processes have rate & period
  - ∑rates ≤ 100%
  - Periods based on processing characteristics, latency needs, etc.
- Jobs have budget & deadline
  - budget = rate * period
  - Deadlines based on period or other characteristics
- Jobs dispatched via Earliest Deadline First (EDF)
  - Budgets enforced with timers
  - Guarantees all budgets & deadlines = all rates & periods



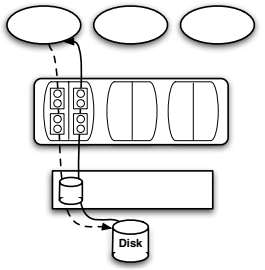
# Adapting RAD to disk, network, and buffer cache

- Fahrrad—Guaranteed disk request scheduling [*Eurosys 2008*]
  Anna Povzner

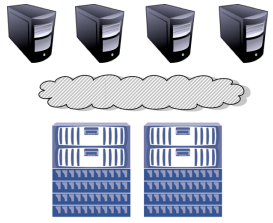
- RADoN—Guaranteeing storage network performance [*FAST08 WiP*]
  Tim Kaldewey & Andrew Shewmaker




- RAD-FLOWS—Buffer management for I/O guarantees [*RTAS 2010*]
  Roberto Pineiro & Kleoni Iouannidou




- Horizon—I/O management for distributed storage systems [*HPDC 2010*]
  Anna Povzner



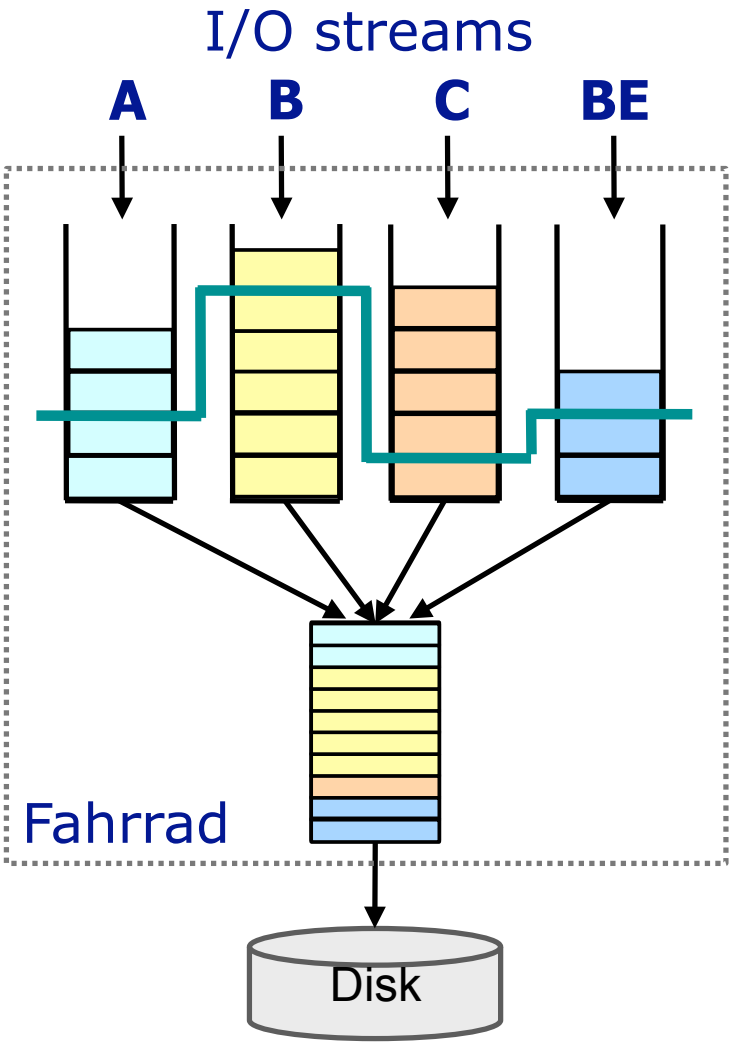
- Plus two more (time permitting)

# Guaranteed disk request scheduling

- Goals
  - Hard and soft performance guarantees
  - Isolation between I/O streams
  - Good I/O performance
- Challenging because disk I/O is:
  - Stateful
  - Non-deterministic
  - Non-preemptable, and
  - Best- and worst-case times vary by 3–4 orders of magnitude

# Fahrrad

- Manages disk *time* instead of disk *throughput*
- Adapts RAD/RBED to disk I/O
- Reorders aggressively to provide good performance, without violating guarantees

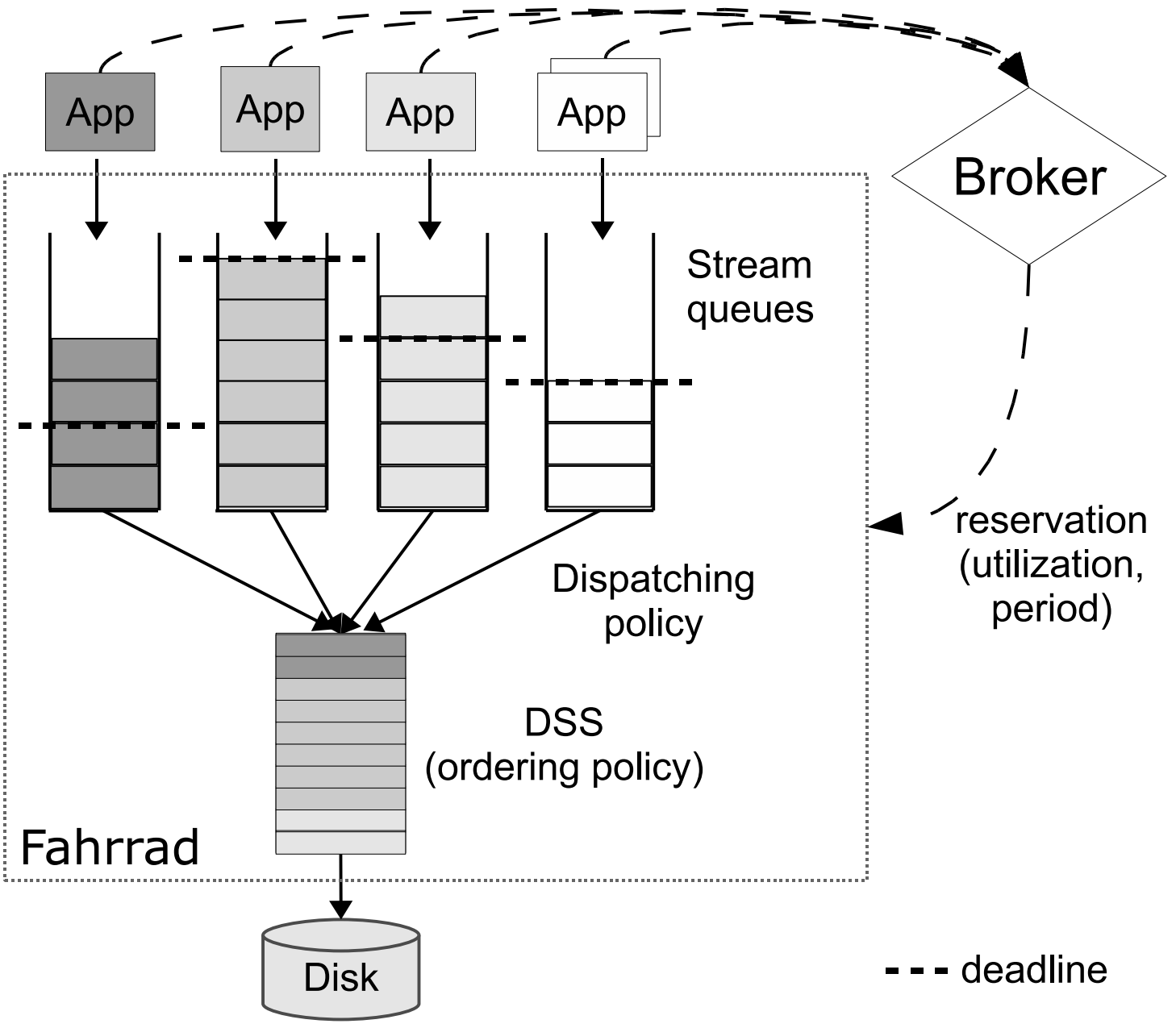



Anna Povzner, Tim Kaldewey, Scott A. Brandt, Richard Golding, Theodore Wong, and Carlos Maltzahn, "*Efficient Guaranteed Disk Request Scheduling with Fahrrad*", Eurosys 2008.

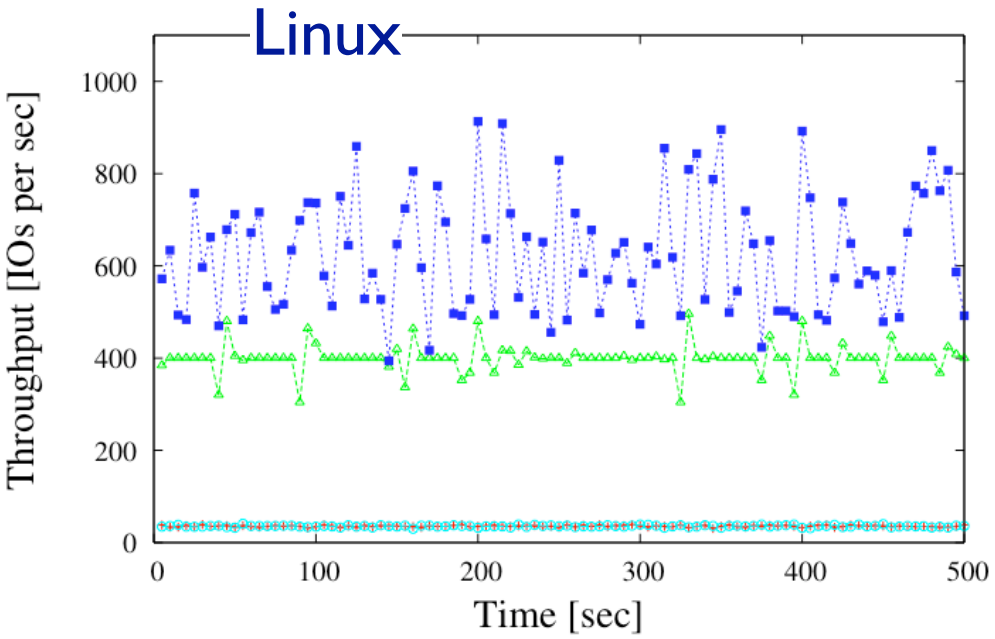
UC SANTA CRUZ

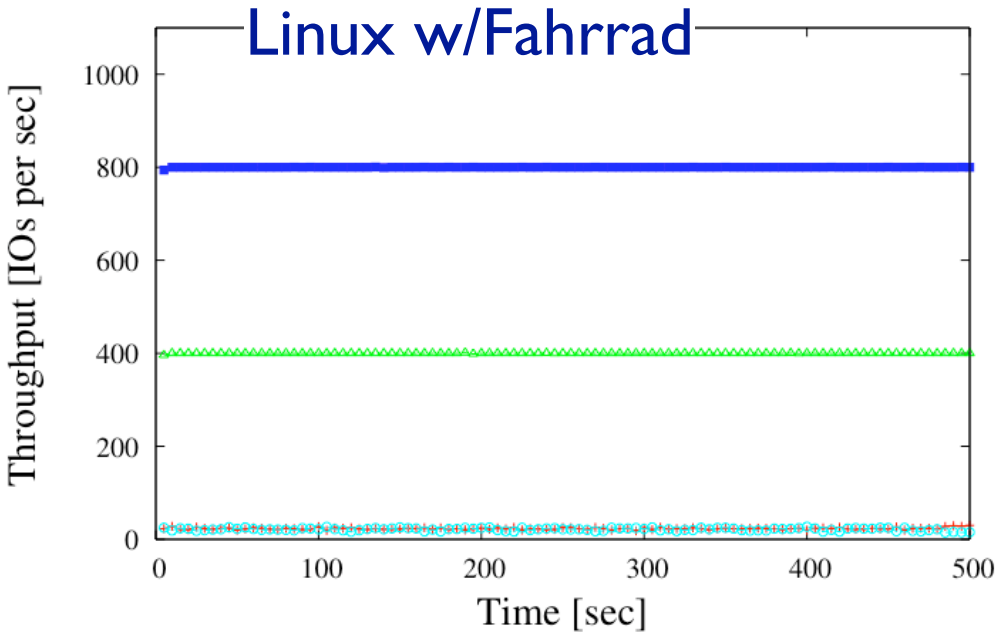
ISSDM

# Fahrrad details



# Fahrrad outperforms Linux FS



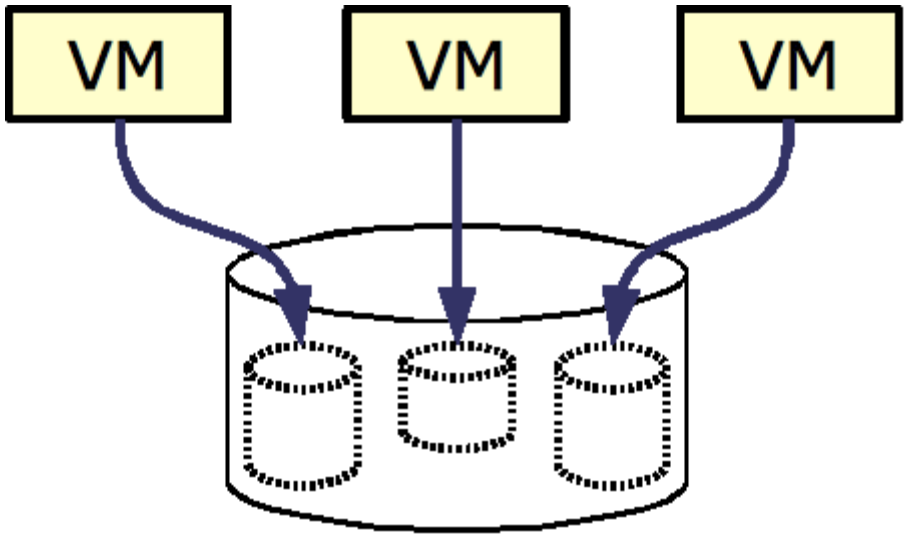

- Workload
  - Media 1: 400 sequential I/Os per second (20%)
  - Media 2: 800 sequential I/Os per second, (40%)
  - Transaction: short bursts of random I/Os at random times (30%)
  - Background: random (10%)
- Result: Better isolation AND better throughput

UC SANTA CRUZ

ISSDM

# Fahrrad virtual disks

- Provide workload-independent performance guarantees
- Isolate from other workloads concurrently accessing the device



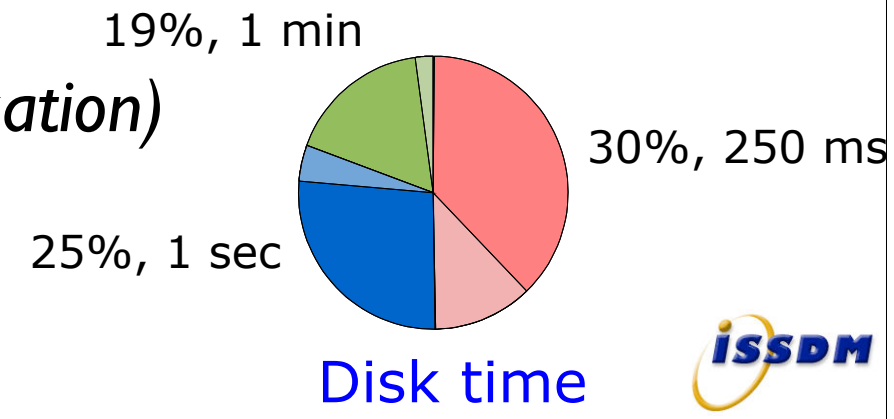

- LUNs virtualize *storage capacity*
- Fahrrad virtualizes *storage performance*

# Fahrrad virtual disks

- Implemented with Fahrrad
- Guarantee share of storage device time
  - Hard guarantees on isolation
  - Throughput = equivalent standalone throughput
- Amount of data transferred:
  - $D_i(x\%, t) = D_i(100\%, x\% \cdot t)$

Share of disk — Time — Share of disk — Time

- Reservation: *disk share (utilization)* and *time granularity (period)* + *overhead*

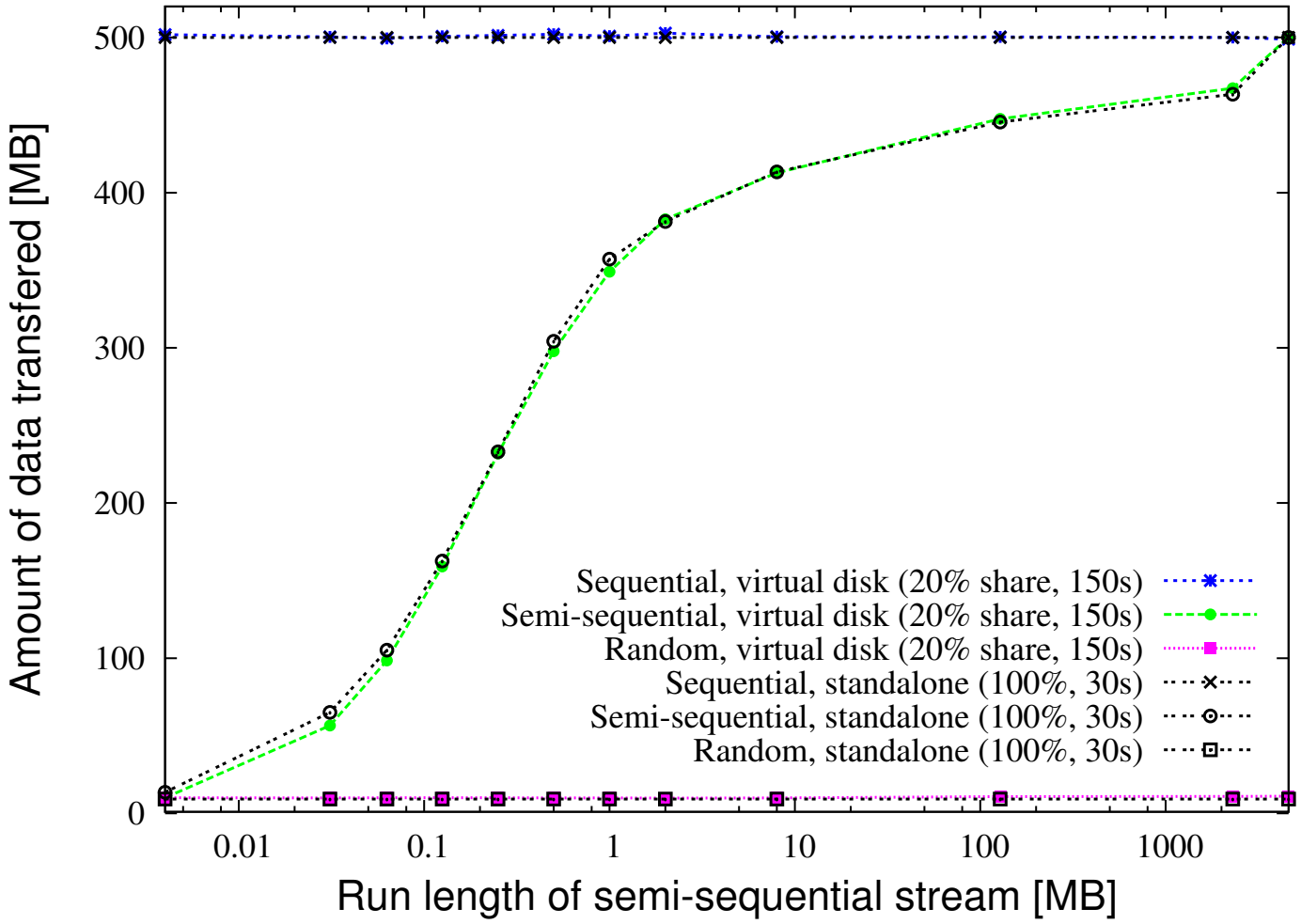



UC SANTA CRUZ

ISSDM

# Fahrrad guarantees throughput

- Throughput is fully determined by reservation & workload
- Virtual disk are completely isolated from each other



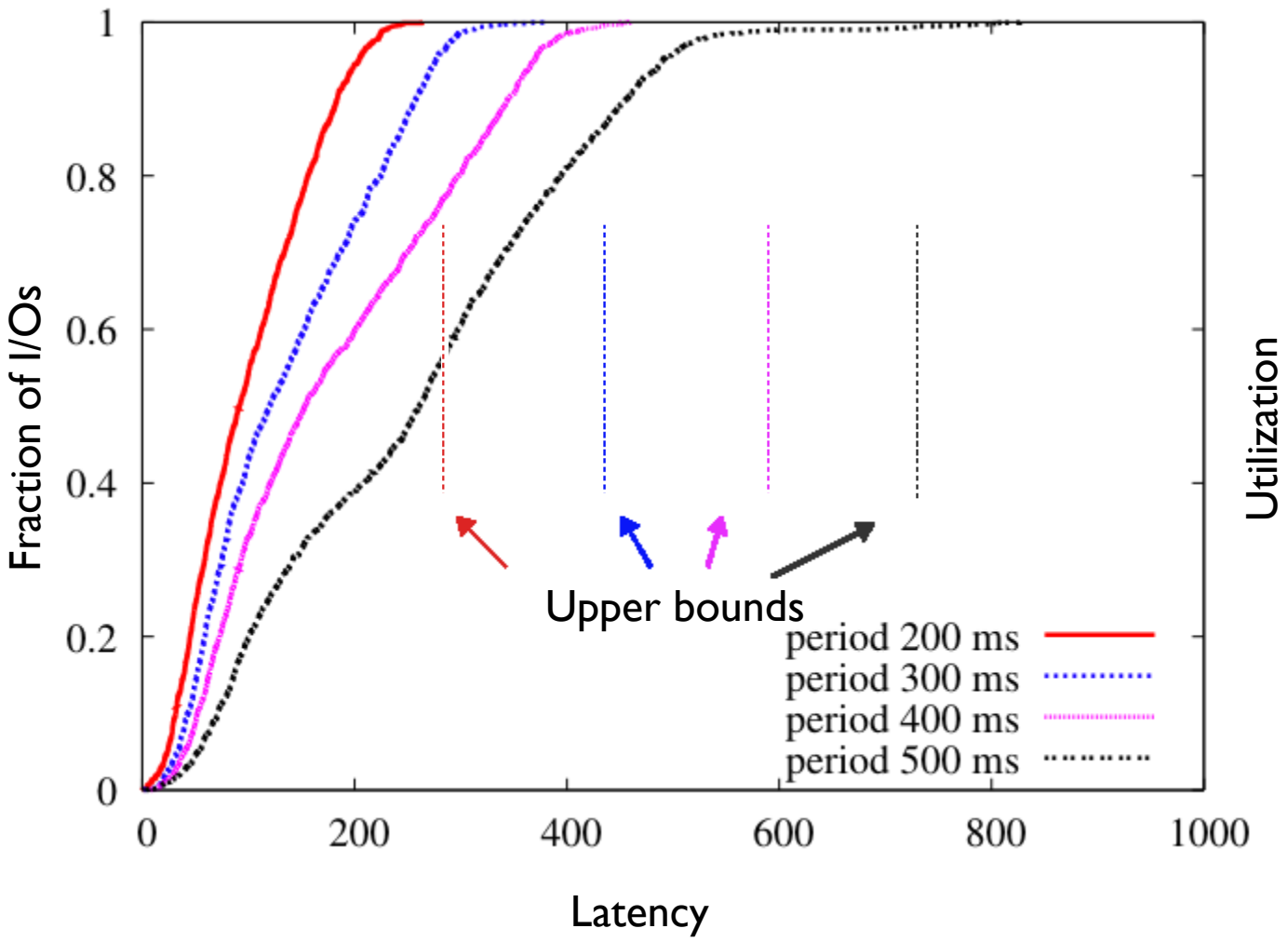

Each virtual disk reserves 20% with 1 second granularity

UC SANTA CRUZ

ISSDM

# Fahrrad guarantees latency

- Latency is bounded by deadlines

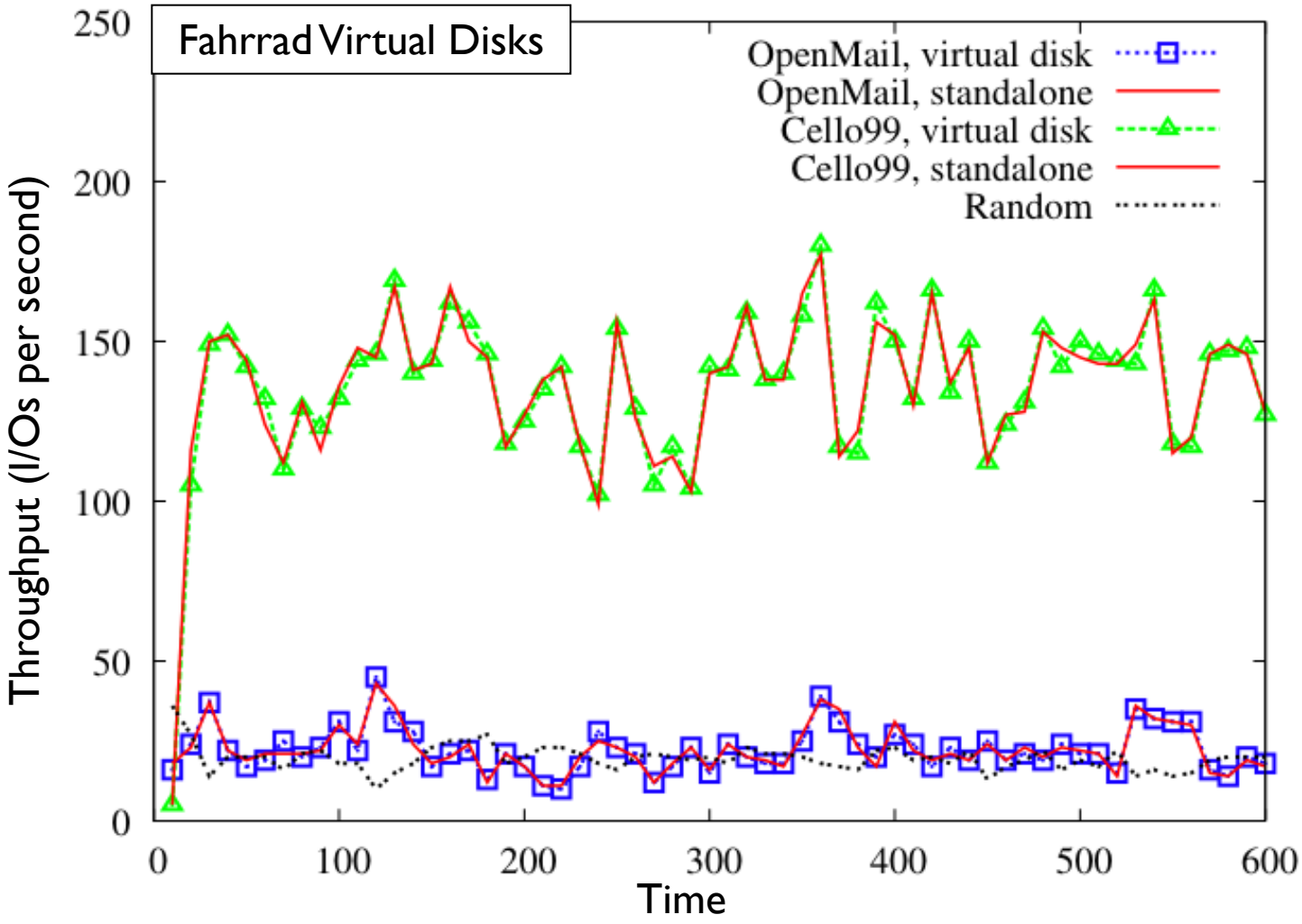



UC SANTA CRUZ

ISSDM

# Fahrrad virtual disks work

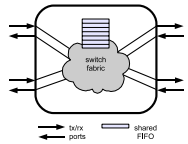
- Fahrrad Virtual Disks provide Cello99 and OpenMail performance very close to standalone
- Cello99 and OpenMail virtual disks share the system with random background stream.




UC SANTA CRUZ

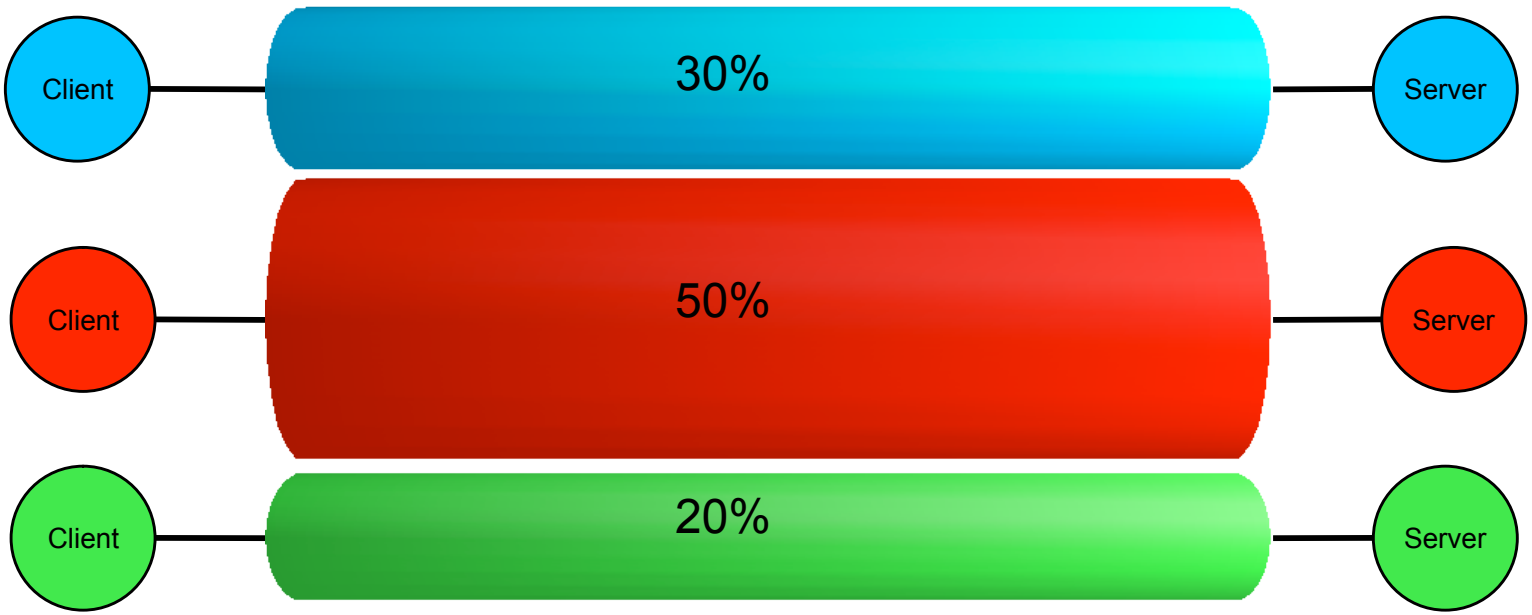
ISSDM

# Guaranteeing storage network performance

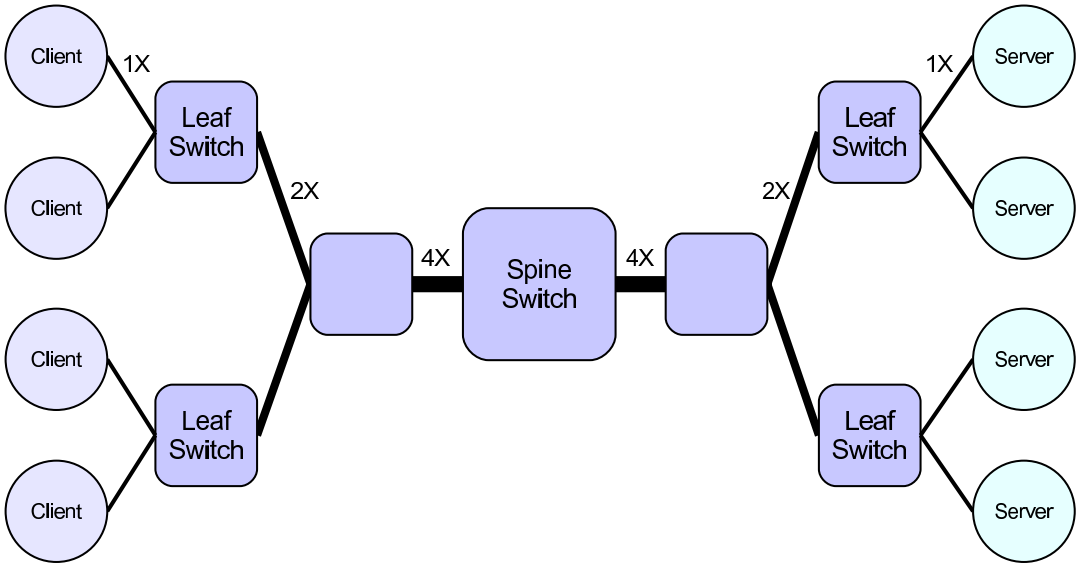


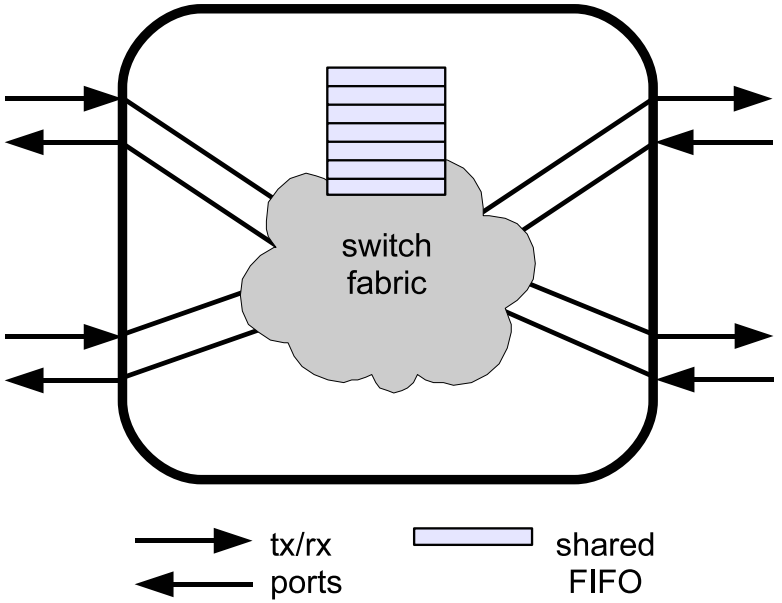

- Goals
  - Hard and soft performance guarantees
  - Isolation between I/O streams
  - Good I/O performance
- Challenging because network I/O is:
  - Distributed
  - Non-deterministic (due to collisions or switch queue overflows)
  - Non-preemptable
- Assumption: closed network

# What we want



# What we have



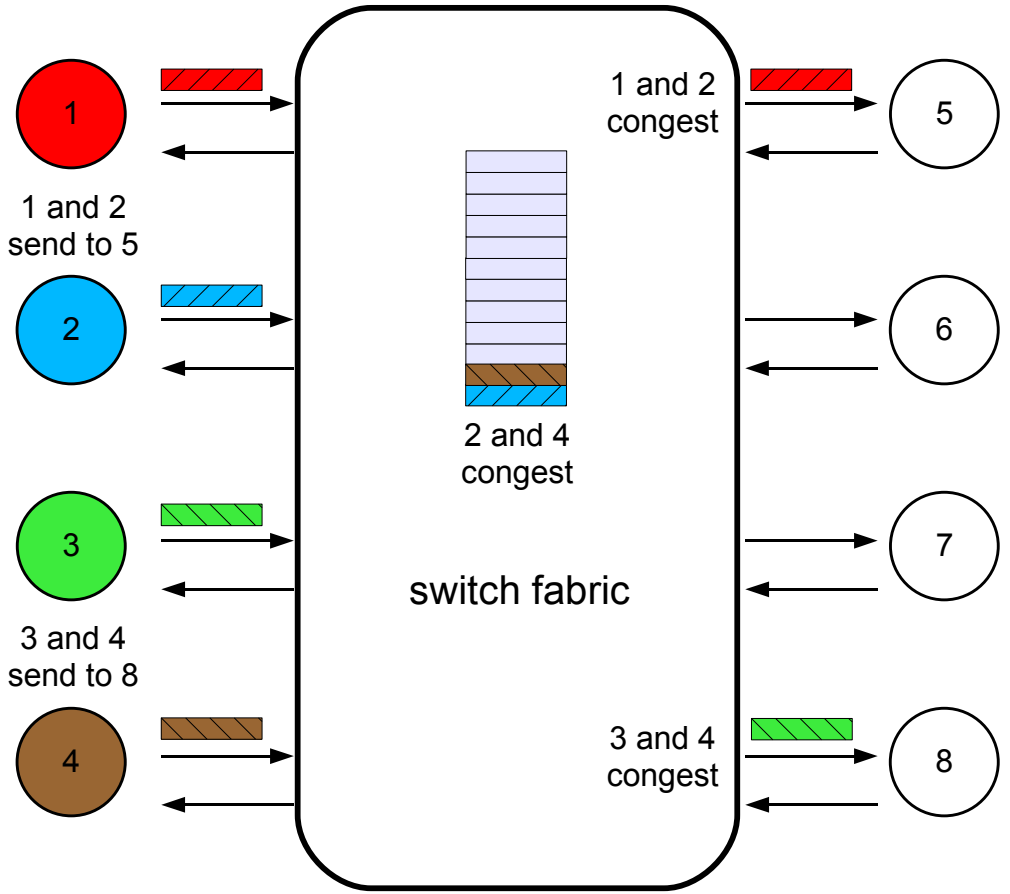

- Switched fat tree w/full bisection bandwidth
- Issue 1: Capacity of shared links
- Issue 2: Switch queue contention

UC SANTA CRUZ

ISSDM

# Congestion in a simple switch model

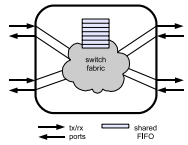
- Delayed packets from unrelated streams affect each other on the queue




UC SANTA CRUZ

ISSDM

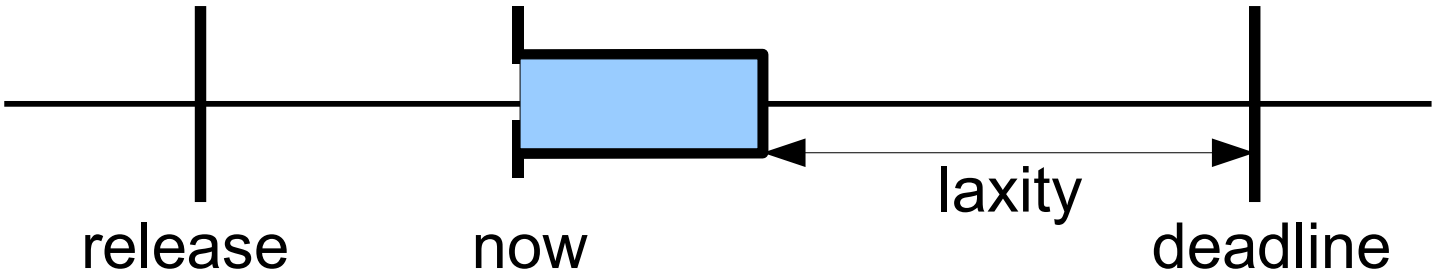
# RADoN




- Each reservation has a *network share* (*utilization*) and a *time granularity (period)*

- Flow control: throttle senders
  - Execution time (per period) e = utilization / period
  - Budget in packets m = e / packets_per_second

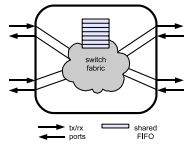
- Congestion control: avoid switch contention by adjusting wait time between packets
  - Percent budget %budget = (1 - %laxity) = e/(d-t)
  - Packet wait time w = $w_{min}$ / %budget
  - Size change w$\Delta$ = -|$w_i$ - $w_{min}$|/2
  - New wait time $w_{i+1}$ = min($w_{max}$, max($w_{min}$, w$\Delta$))




UC SANTA CRUZ

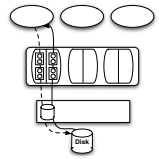
ISSDM

# Userspace RADoN prototype



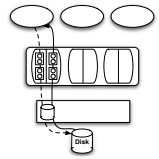

- Detects congestion using Relative Forward Delay
- Responds to congestion using RAD real-time theory
- Decreases packet loss significantly
- Improves goodput
- Requires **no** global knowledge or synchronization
- Ongoing: RADoN kernel implementation

# Buffer management for I/O guarantees



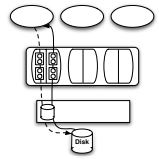
- Goals
  - Hard and soft performance guarantees
  - Isolation between I/O streams
  - Improved I/O performance
- Challenging because:
  - Buffer is space-shared rather than time-shared
    - Space limits time guarantees
  - Best- and worst-case are opposite of disk
  - Buffering affects performance in non-obvious ways

# Buffering roles

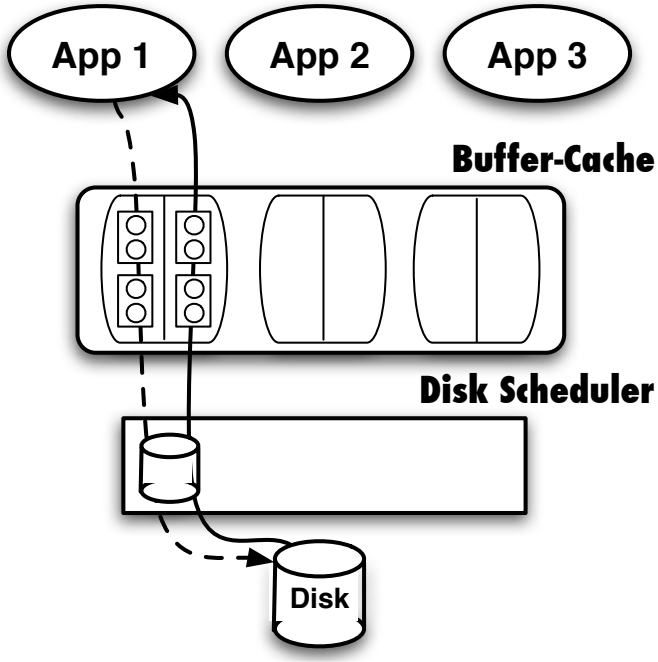



- Decoupling
  - Allows sender and receiver to operate asynchronously
- Speed matching
  - Allows slower and faster devices to communicate
- Traffic shaping
  - Shapes traffic to optimize performance of interfacing devices

# Radium



- I/O into and out of buffer have *rates* and *time granularities (periods)*
- Translate *time requirements* into *space requirements*
- Cache policies enhance performance within constraints determined by I/O requirements
  - Use slack to prefetch reads and delay writes

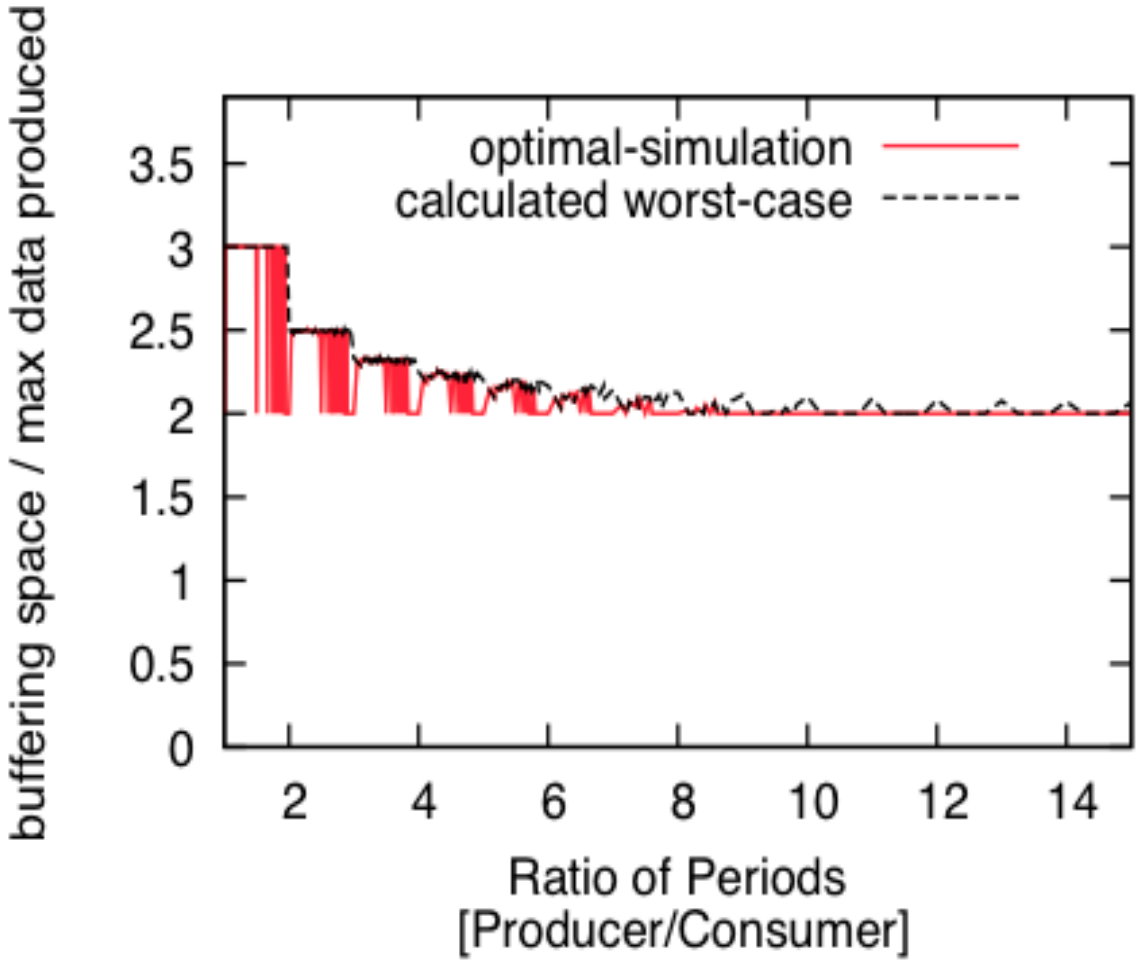



Roberto Pineiro, Kleoni Iouannidou, Carlos Maltzahn, and Scott Brandt, “RAD-FLOWS: Buffering for Predictable Communication,” RTAS 2011

UC SANTA CRUZ

ISSDM

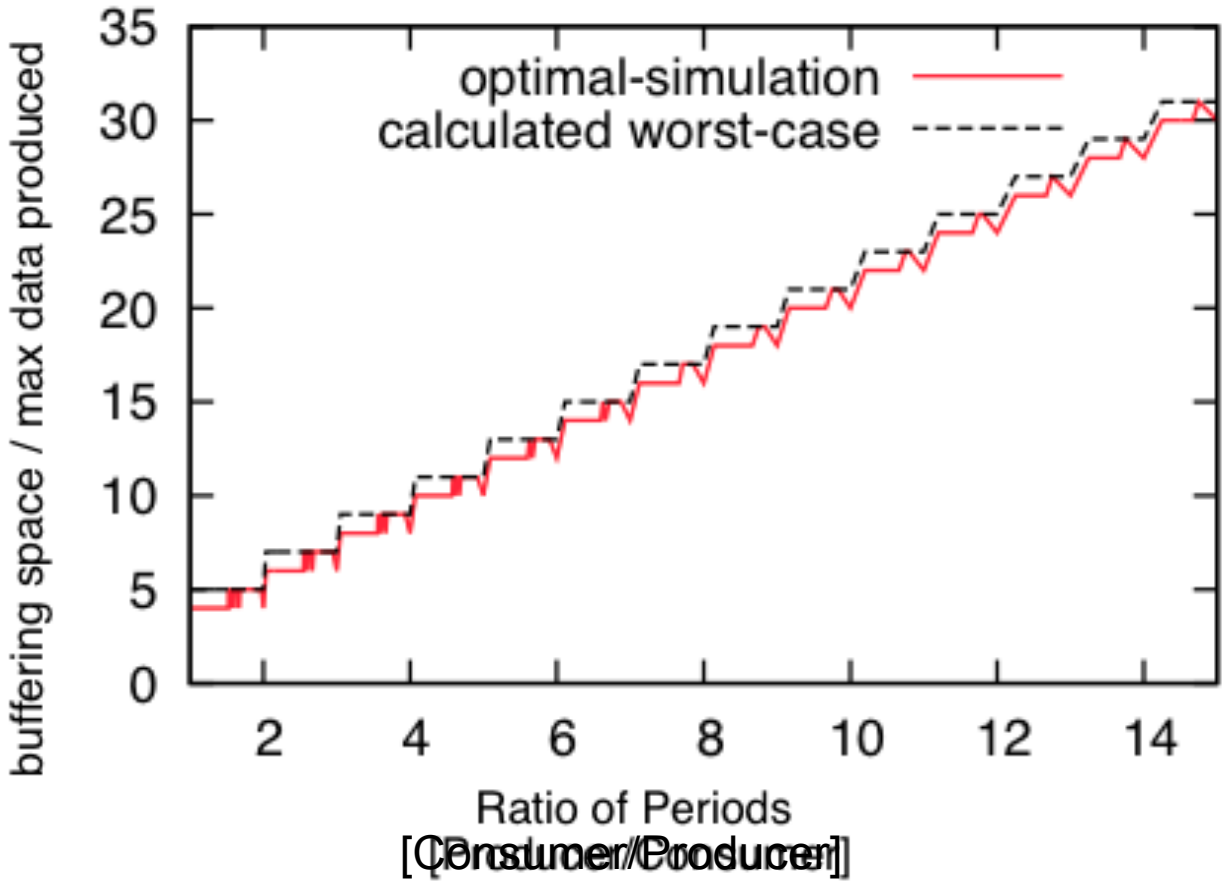
# Analysis when $p_p > p_c$




$$Buffer\ Space\ B \leq 2r_p p_p + \max(0, r_p p_p - (\lfloor \frac{p_p}{p_c} \rfloor - 1) r_c p_c))$$

$$Buffer\ Time\ T \leq 3p_p$$

# Analysis when $p_c > p_p$

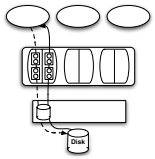



[Consumer/Producer]

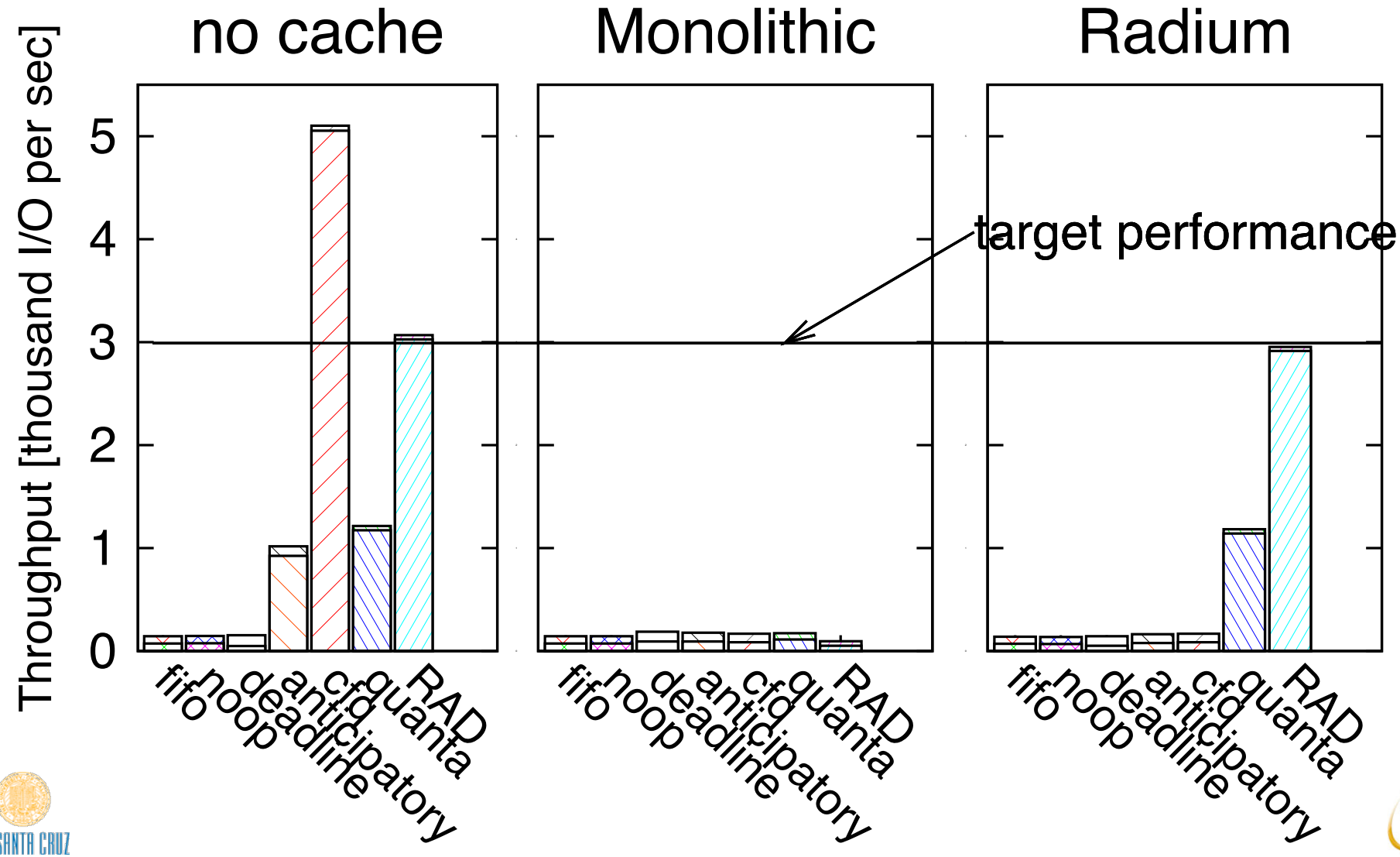
$$Buffer\ Space\ B \leq 2\left(\left\lceil \frac{p_c}{p_p} \right\rceil + 1\right) r_p p_p - r_p p_p$$

$$Buffer\ Time\ T \leq 2p_c$$

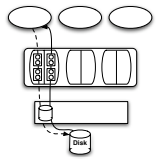
# Managing combined workloads

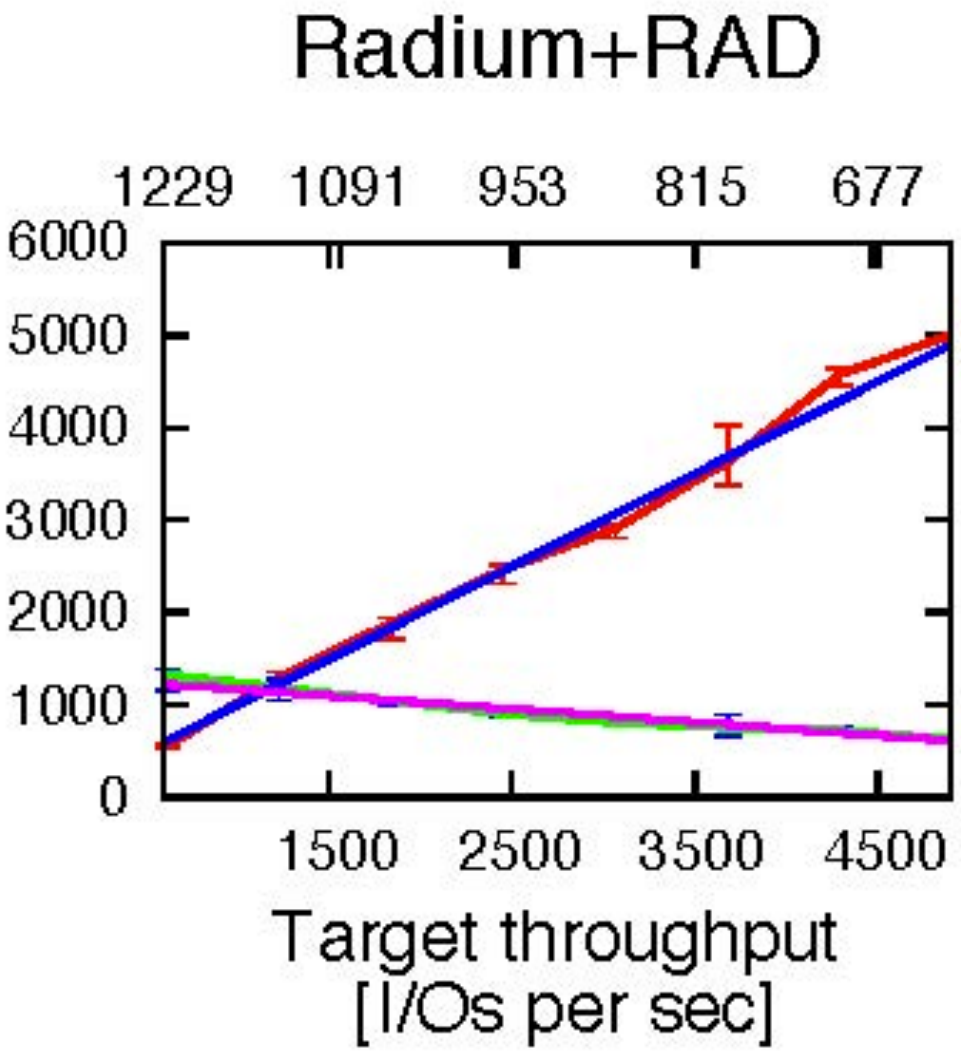


Combined throughput of rand.(top) and seq.(bottom) workloads



# Controlling throughput w/mixed workloads



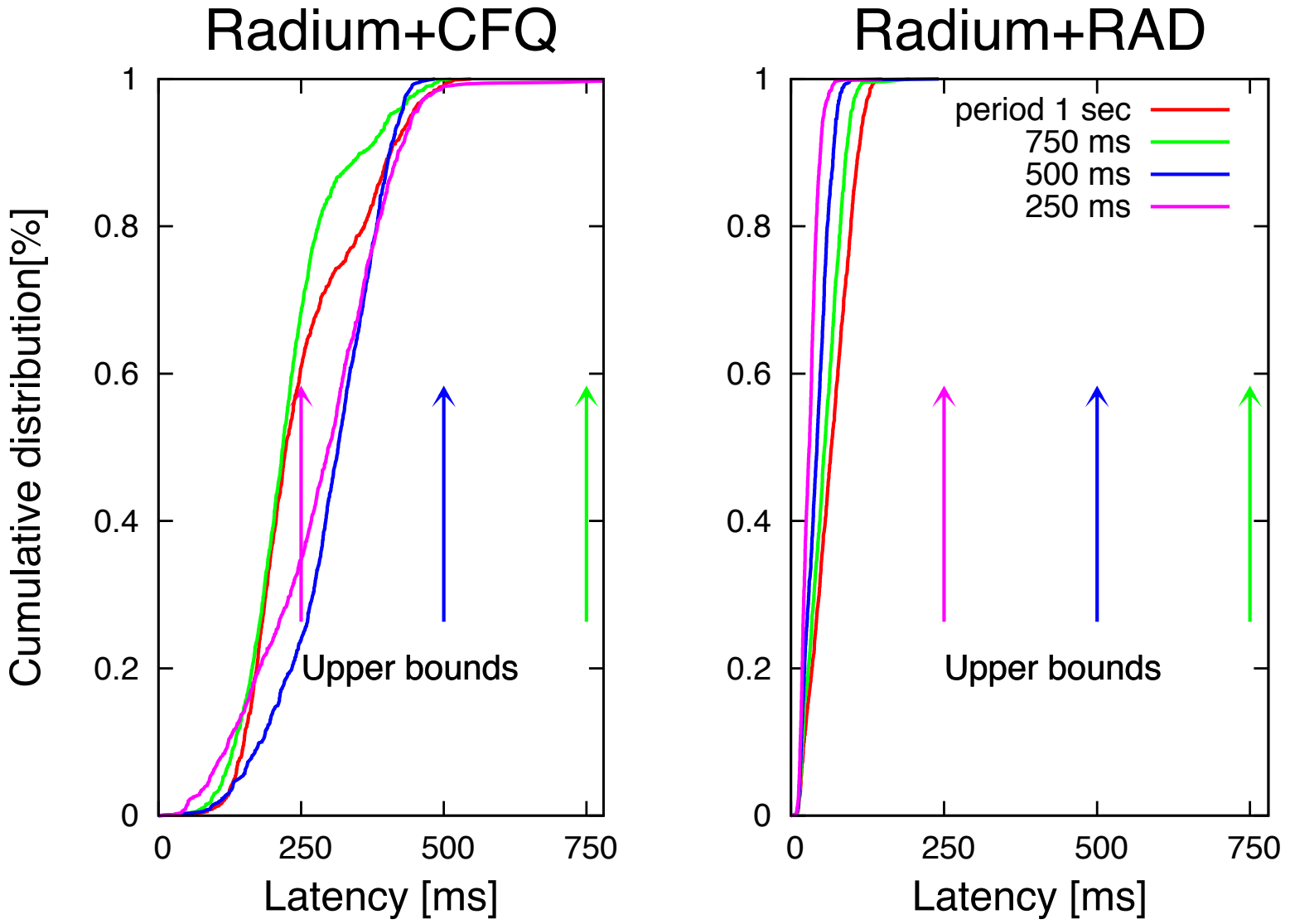



Precise control over the service times of each stream

UC SANTA CRUZ

ISSDM

# Controlling latency w/mixed workloads

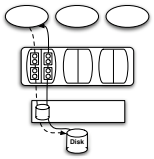


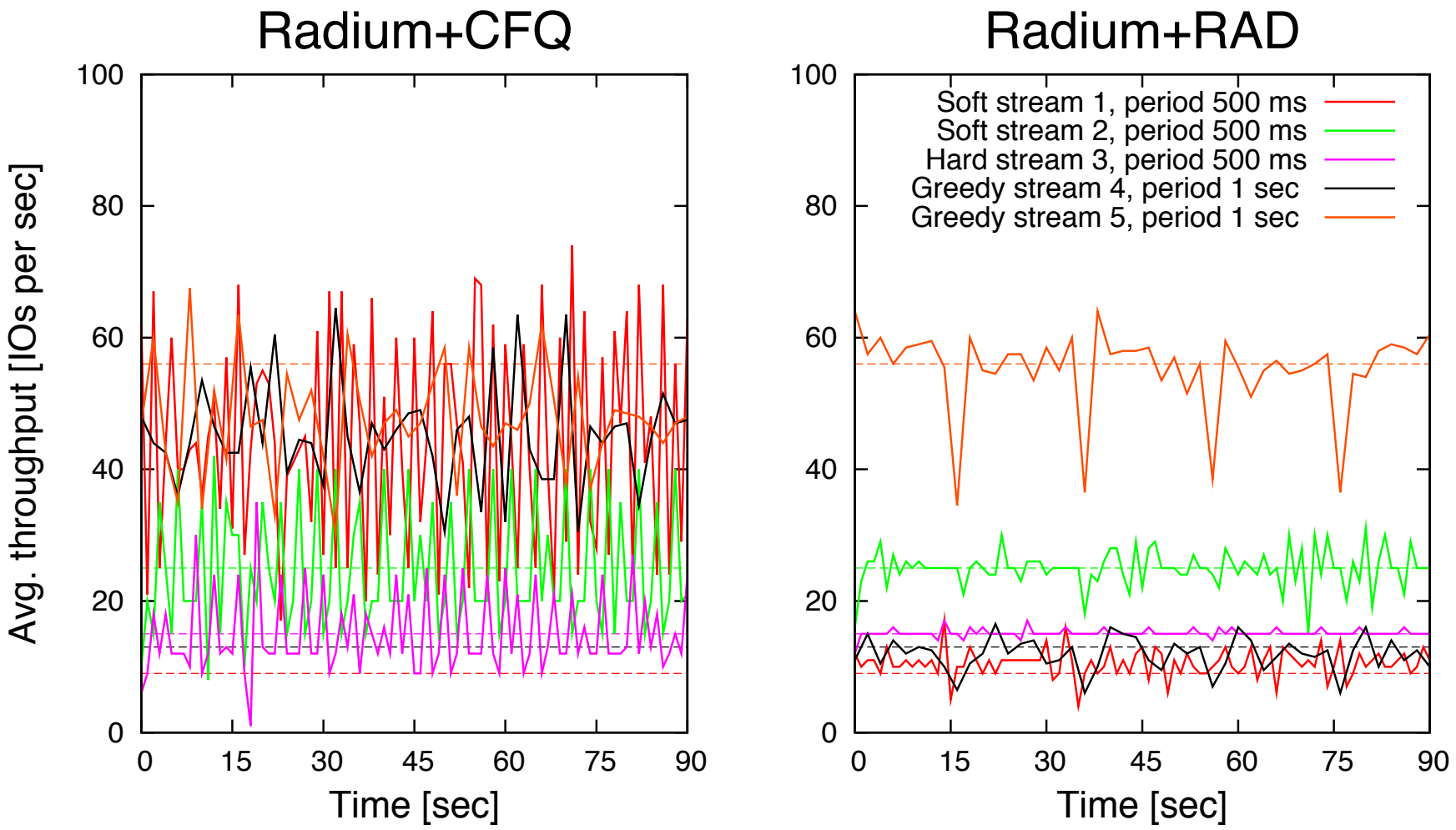

Precise control over the service times of each stream

UC SANTA CRUZ

ISSDM

# Results w/complex workloads





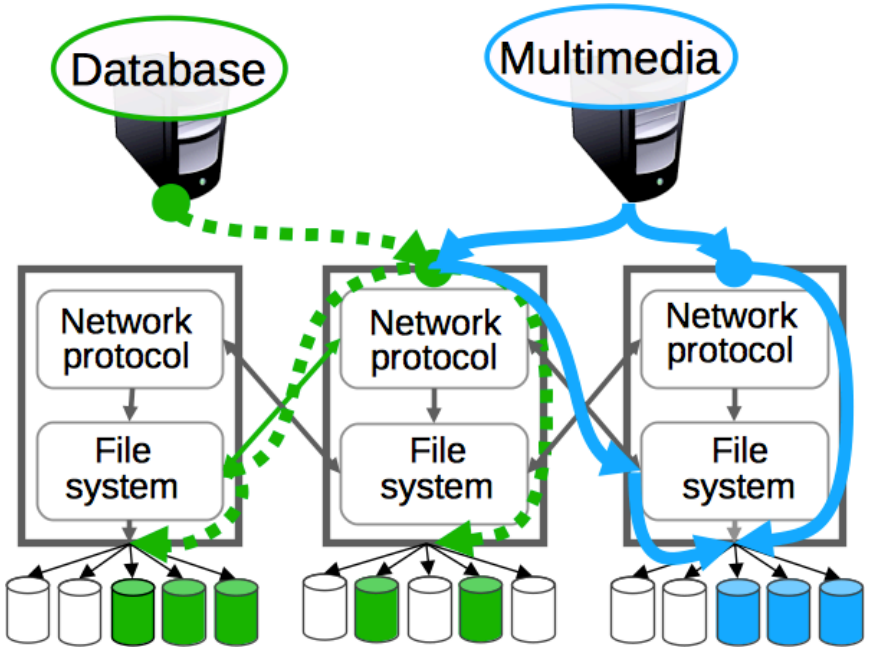

Reasonable control with complex workloads

UC SANTA CRUZ

ISSDM

# Horizon

- Big storage systems are shared, have many disks, and application workloads compete and interfere
- Real distributed systems have
  - Different data layouts
  - Multiple data entry points
  - Different data paths
- Horizon goals
  - Meet performance targets
  - Fully utilize system resources
  - Not rely on reservations
  - Decentralized solution

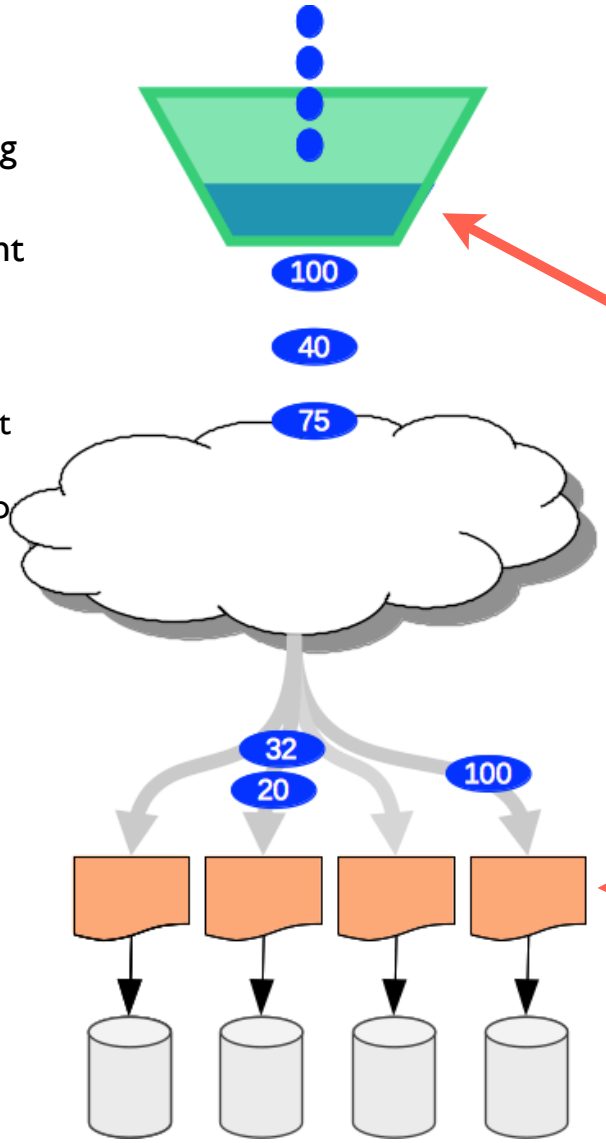



Anna Povzner, Darren Sawyer, Scott A. Brandt, "Horizon: Efficient Deadline-Driven Disk I/O Management for Distributed Storage Systems," HPDC 2010. **Best Paper**

UC SANTA CRUZ

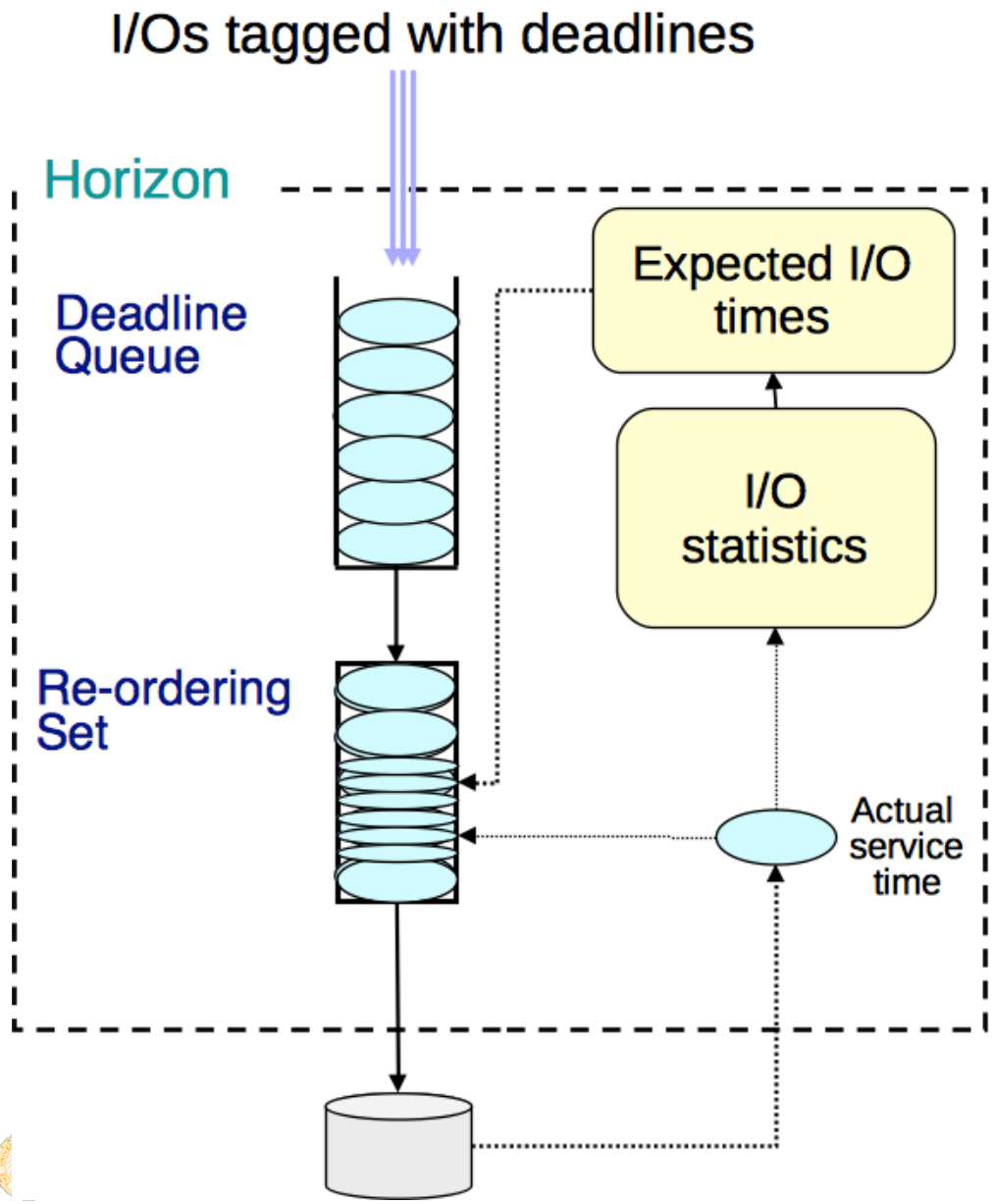
ISSDM

# Multi-layered approach



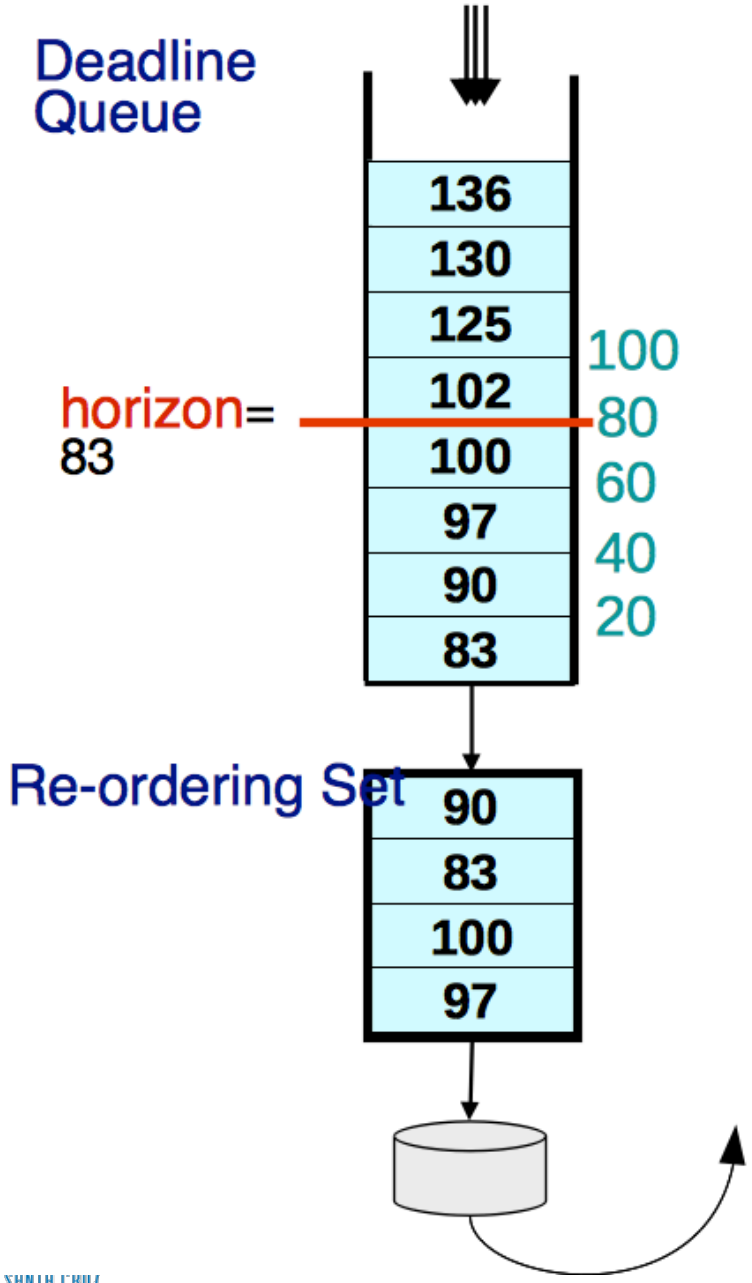

- Workloads specify performance targets
  - Throughput and latency
- Upper layer control mechanism
  - Throughput limiting
  - Deadline assignment based on throughput and latency targets
- Low-level disk schedulers
  - Meet individual request deadlines

# Horizon disk scheduling




- Manage I/O in terms of disk time
- Estimate service times based on service time measurements
- Reorder requests within "slack time" before earliest deadline
- Adjust based on optimizations, overload, latency

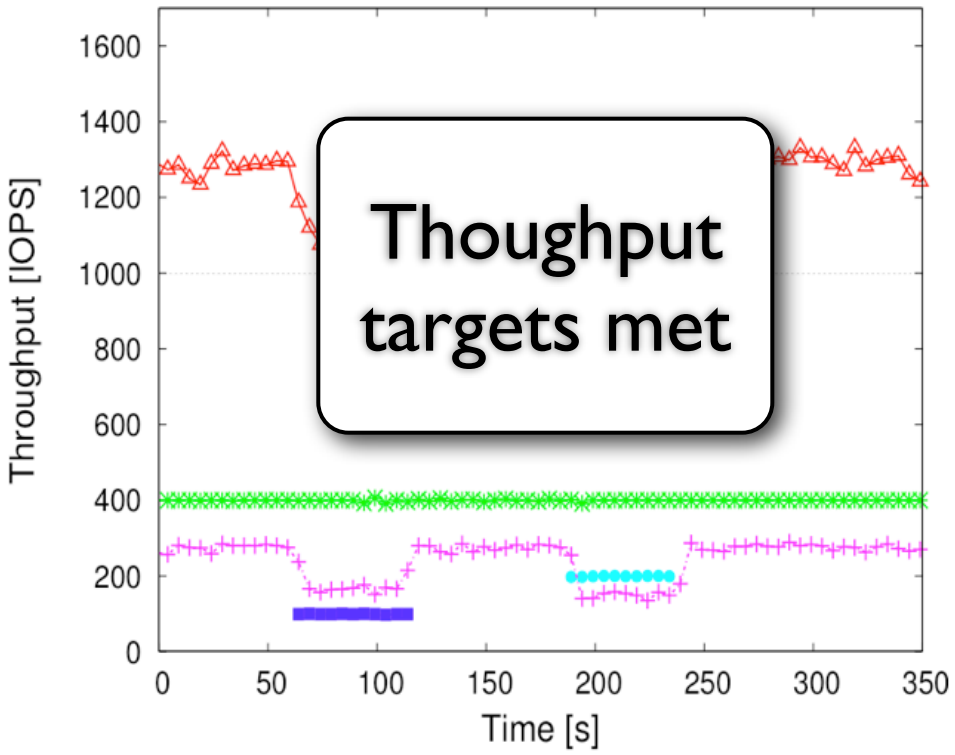
# Horizon disk scheduling

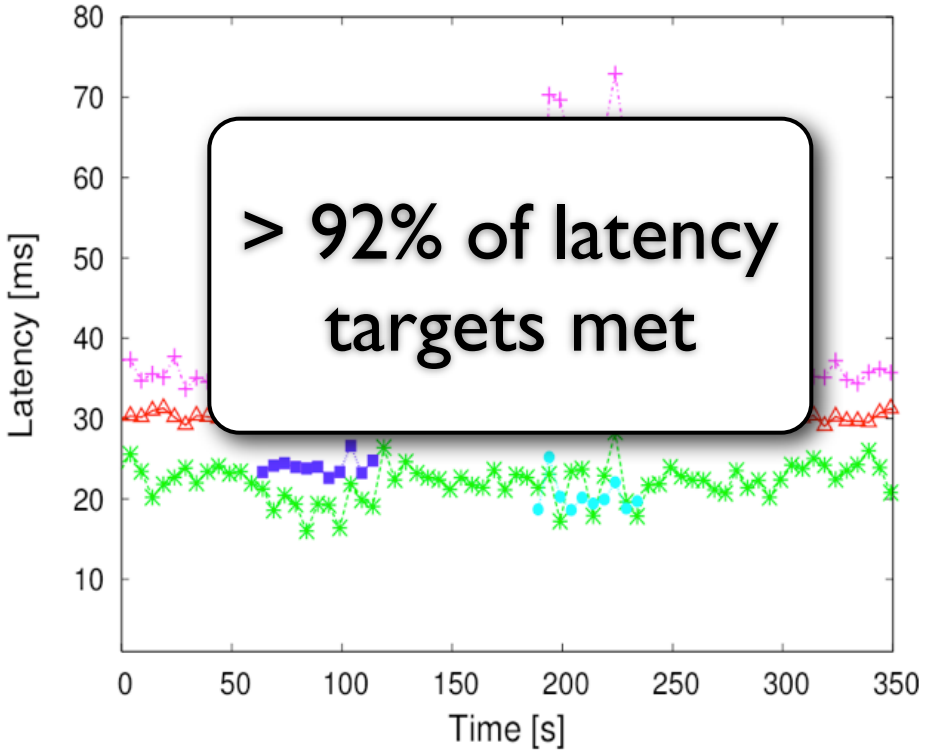



- Horizon set to earliest deadline
- Reordering set = everything that will fit before horizon
- Execution times measured as requests complete
- Optimizations
  - Squeeze in more sequential I/Os
  - Use optimistic estimates
  - Increase reordering set (esp. under overload)
  - Increase device queue
    - Larger = better performance
    - Smaller = tighter deadlines

# Horizon in use

- Implemented in NetApp's Data ONTAP (data from FAS3040)
- Performance targets associated with volumes
  - Control mechanism at FS entry point
  - Schedulers between RAID and disks




*40% random, 40 in flight*
*1000 IOPS target*

*media, 40 IOs / 100ms*
*target: 400 IOPS, 80 ms latency*

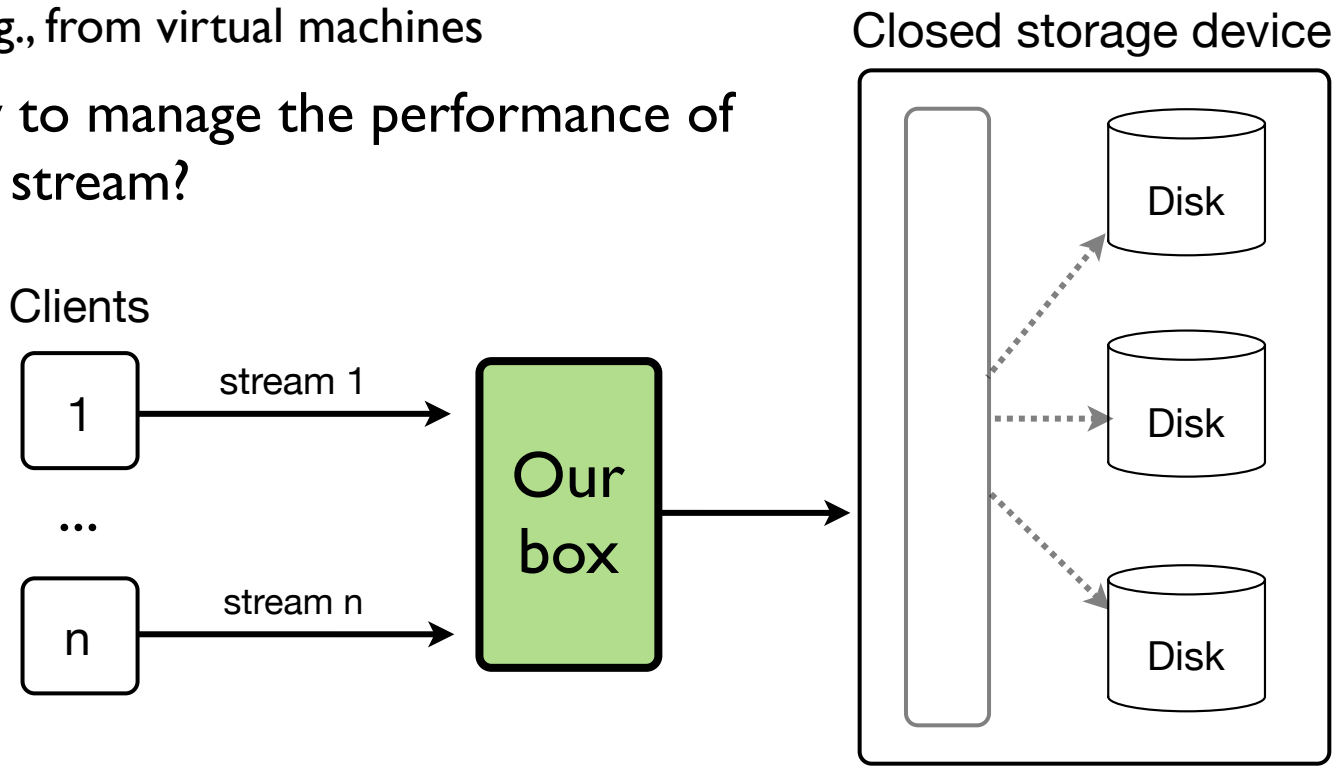
*bursty: 8 IOs / 40 ms*
*40ms latency target*

*random background*

*bursty: 4 IOs / 40 ms*
*40ms latency target*

# Beyond horizon

- Horizon applied to black box storage
- Problem:
  - Storage device may have many disks
    - Example: big shared storage
  - Competing streams of requests
    - e.g., from virtual machines
  - How to manage the performance of each stream?



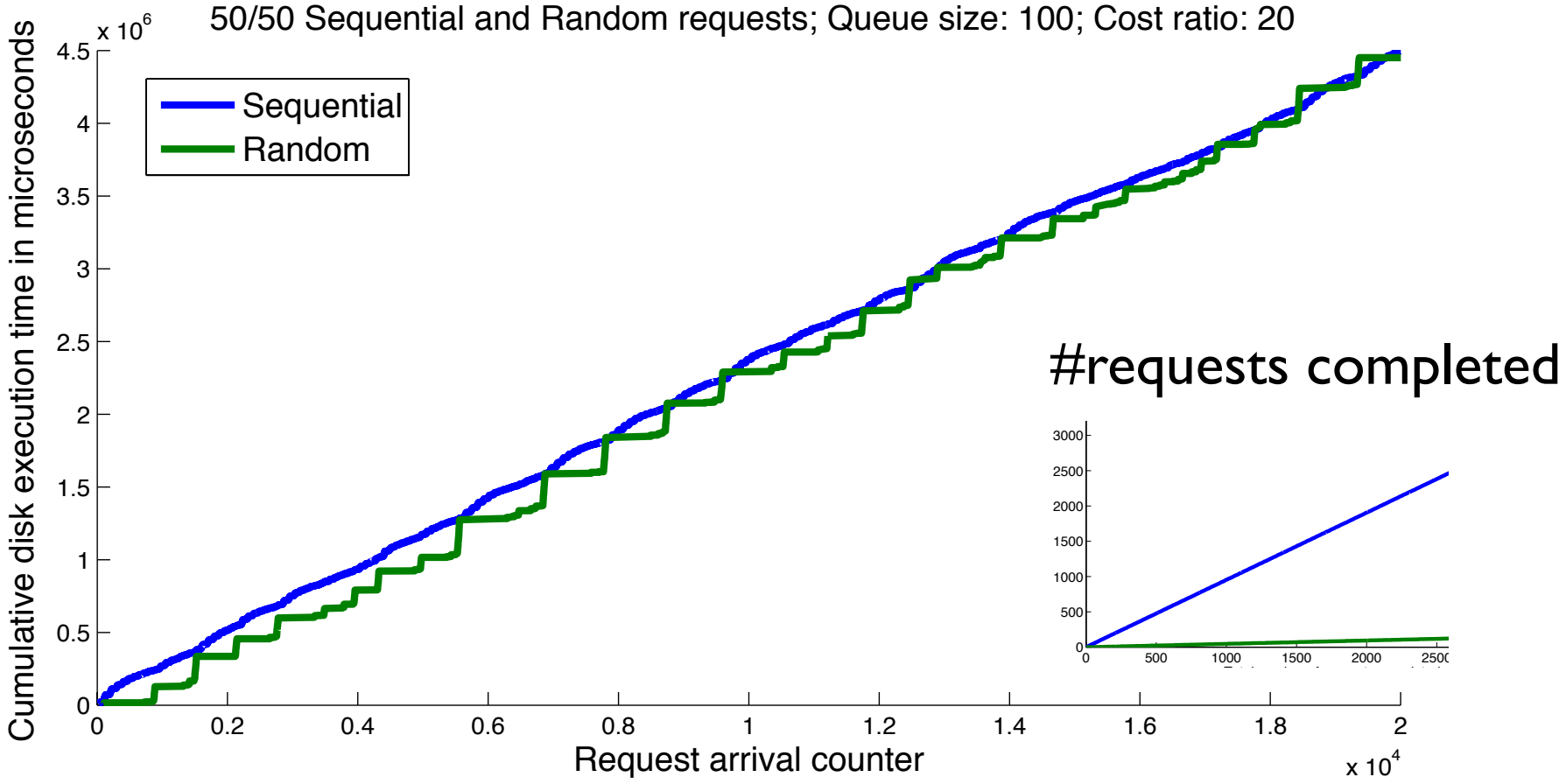
# Solution: Horizon++

- Average over large numbers of requests
- Assume fixed cost ratio between random and sequential requests
- Measure time for 1000 requests
- Compute performance of random and sequential requests
- Charge streams accordingly

UC SANTA CRUZ

ISSDM

# Results w/stable workload

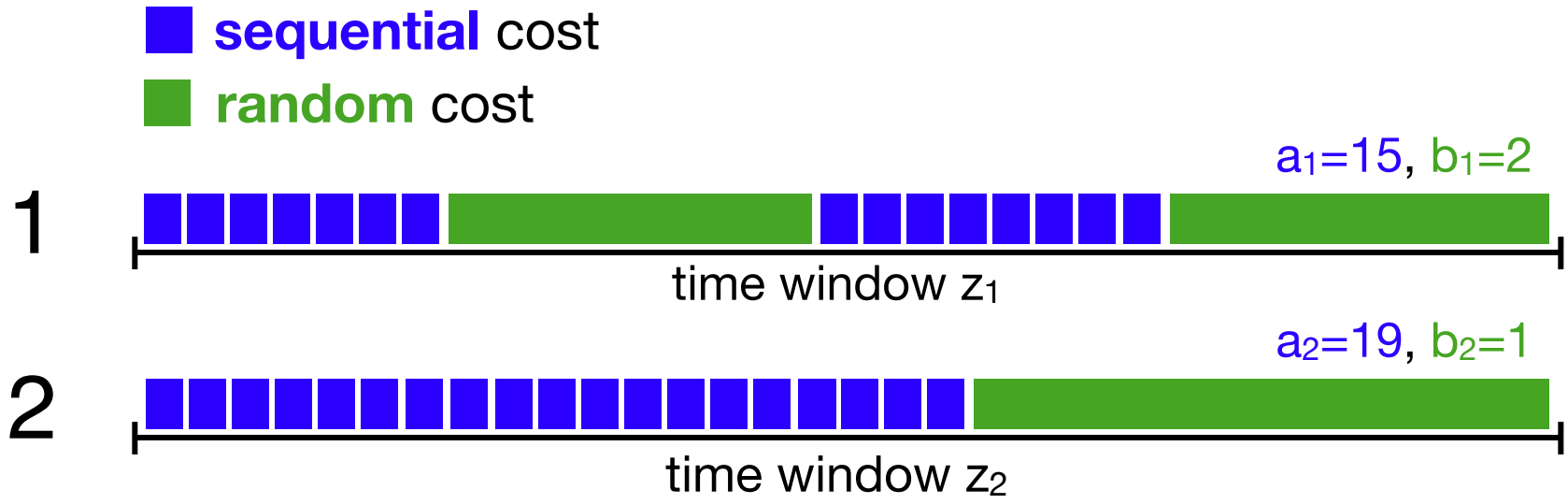



Indication that given a proper update method we can solve the original problem.

UC SANTA CRUZ

ISSDM

# Intersection update



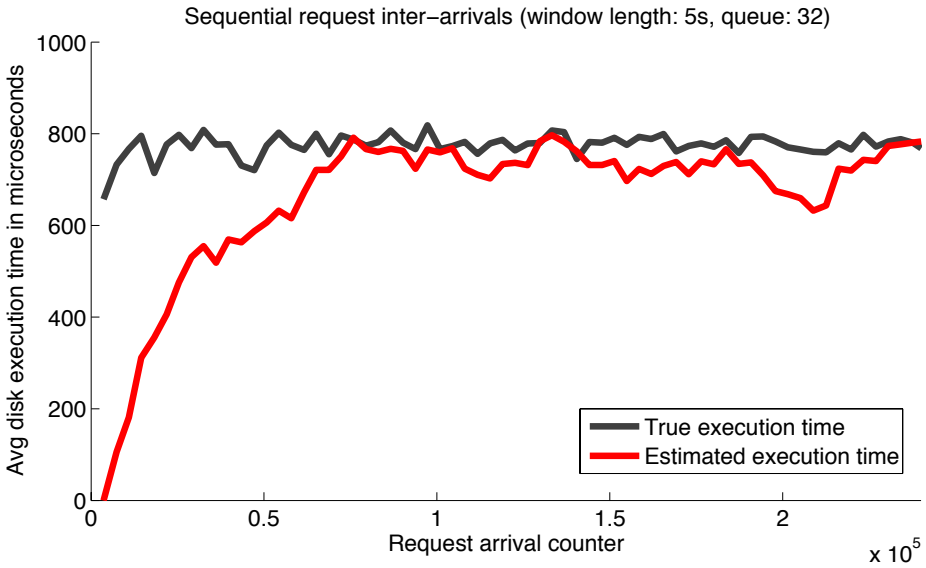

If the execution **costs (x,y)** are the same in $z_1$ and $z_2$ then,

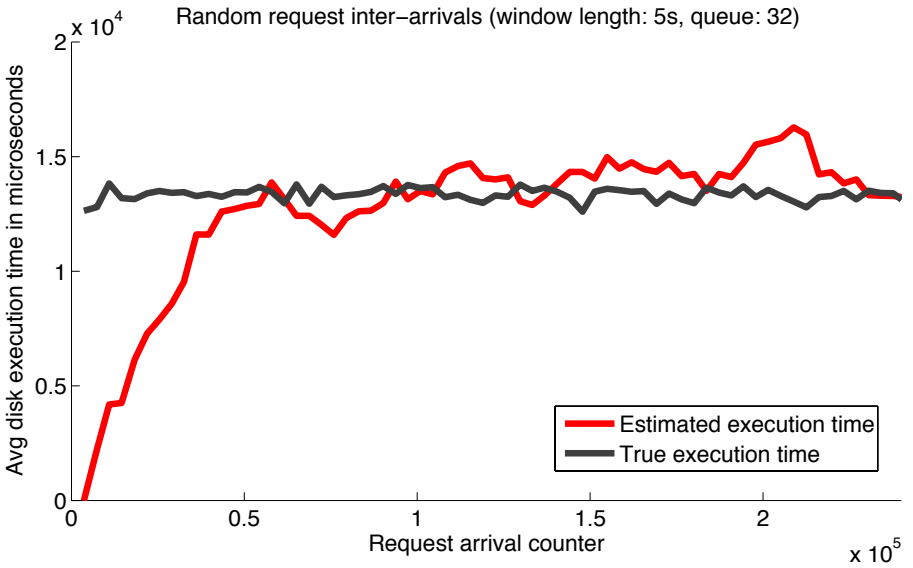
$$a_1x + b_1y = z_1$$
$$a_2x + b_2y = z_2$$

**sequential** cost
**random** cost

we can find the average execution cost of a sequential/random request (unless the lines are parallel.)

# Intersection update works



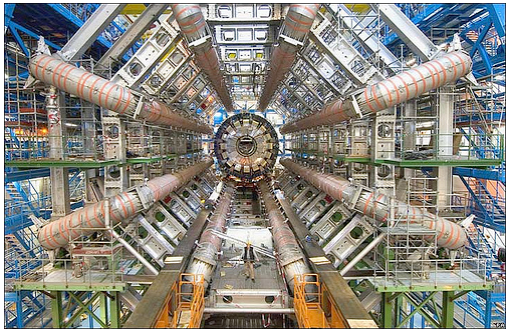

- Costs may not always be stable, so
- Average over sliding-window intervals and update smoothly
- Result: Horizon++ will work

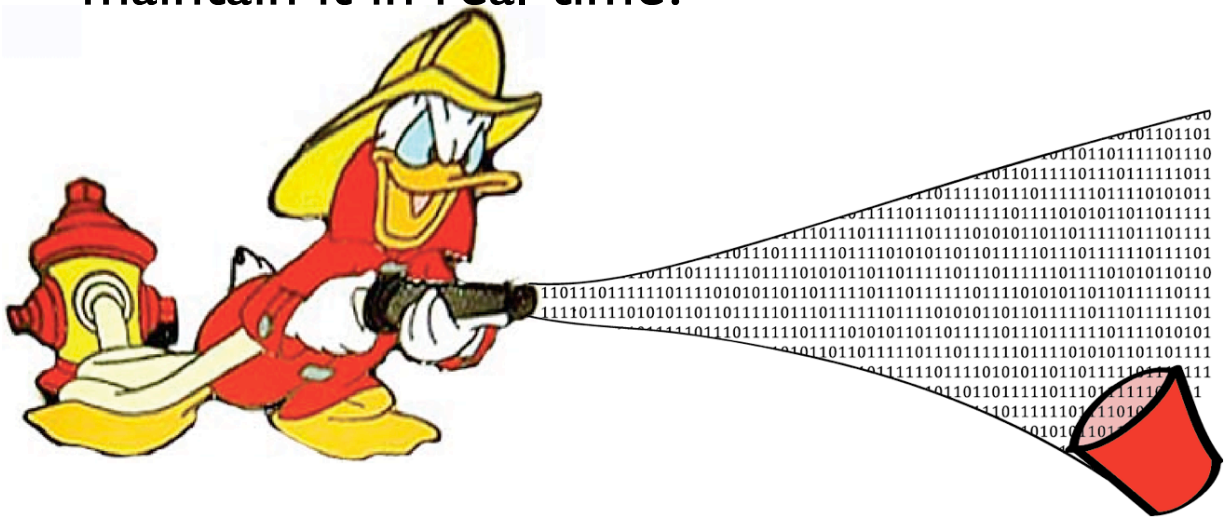
UC SANTA CRUZ

ISSDM

# And one more: TiVo for telescopes [*MSST 2010, RTAS 2012?*]

- Many systems generate huge volumes of data
  - Accelerators, telescopes, simulations, internet traffic generate MB/s, GB/s, or TB/s of data
- Most of the data is uninteresting
  - But some of it is extremely interesting
- How can we capture, index, search, and maintain it in real-time?



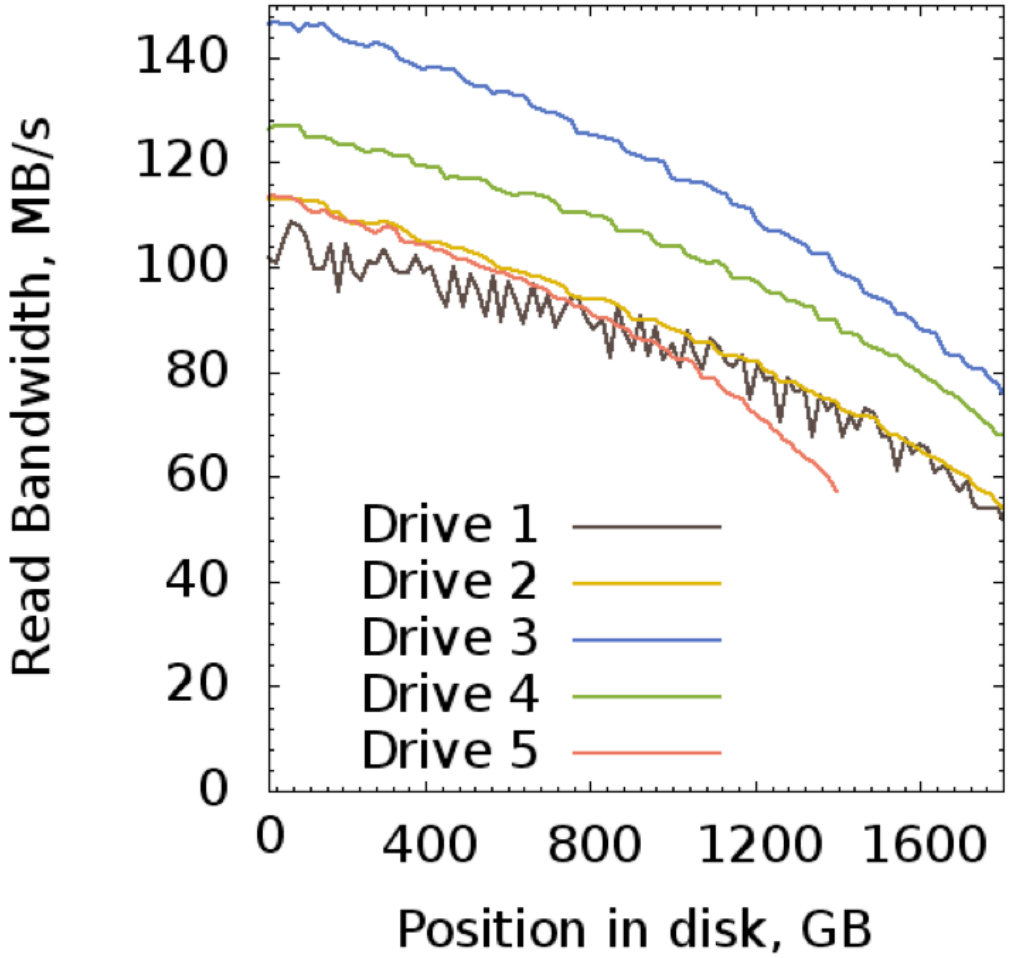


UC SANTA CRUZ

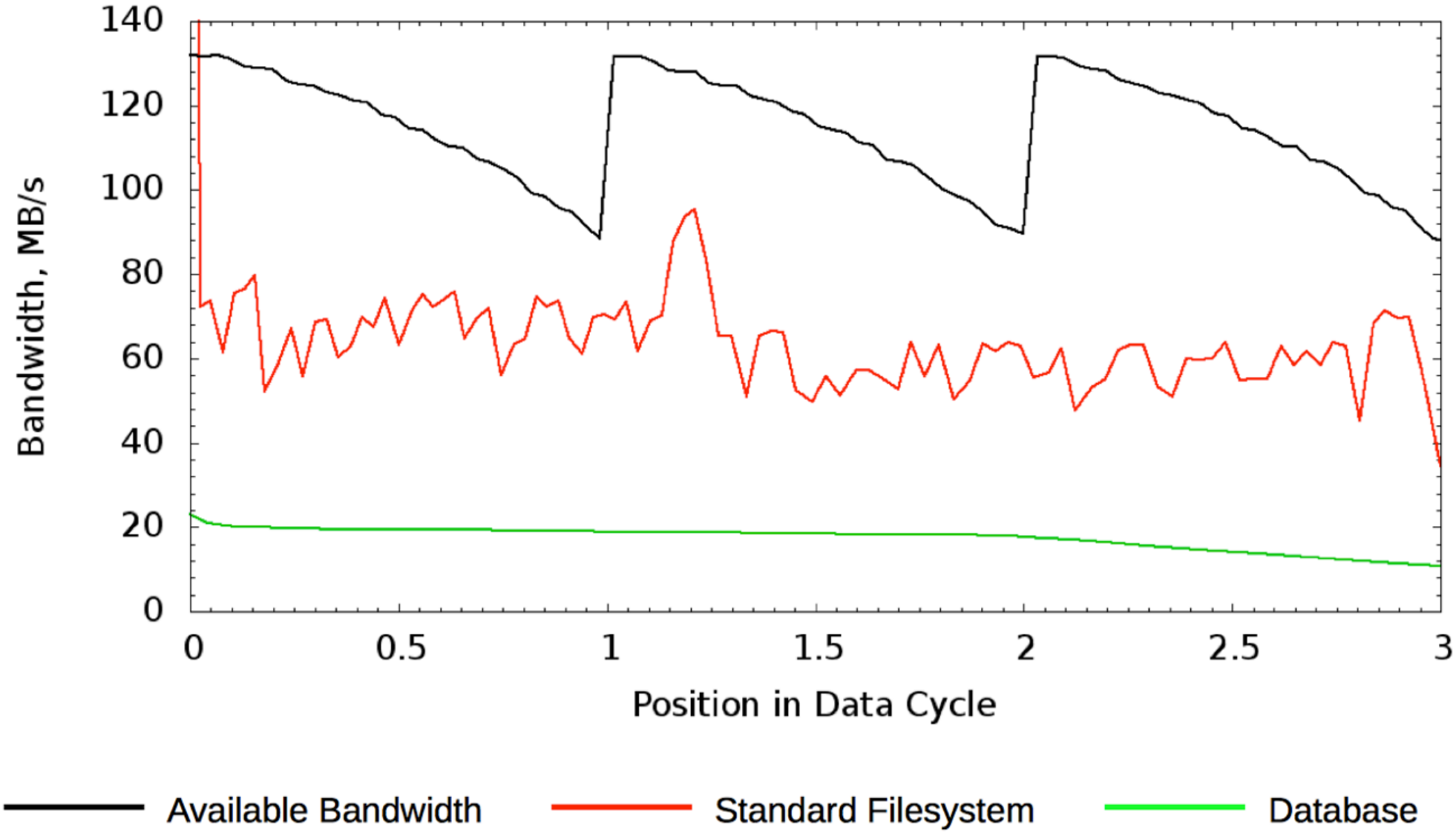
ISSDM

# Understanding disk performance

- Each disk has a unique deterministic performance curve
- Performance degrades over the course of the disk
- Understanding the curves enables performance guarantees

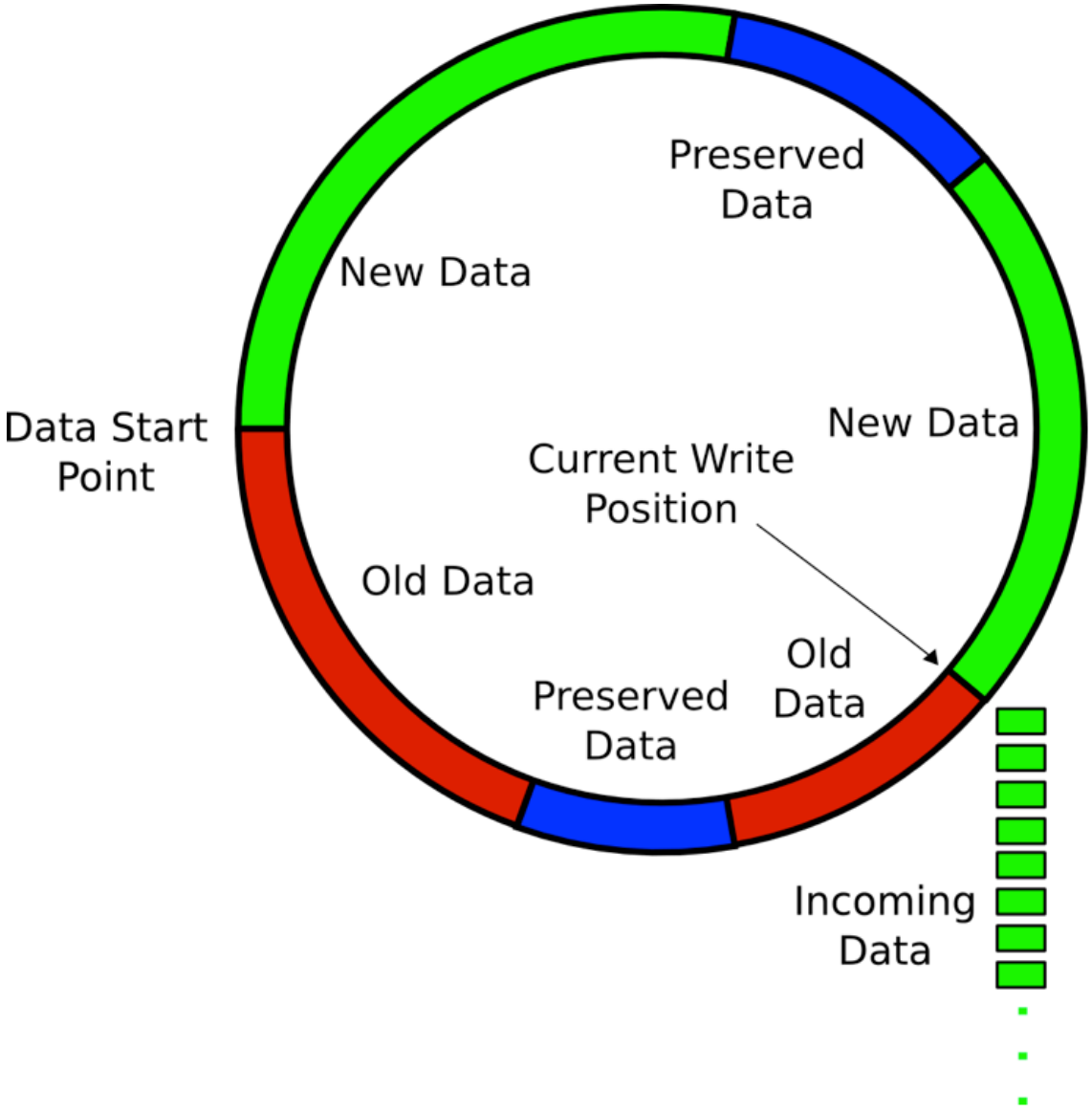


# Existing systems



# Solution - use the disk as a ring buffer

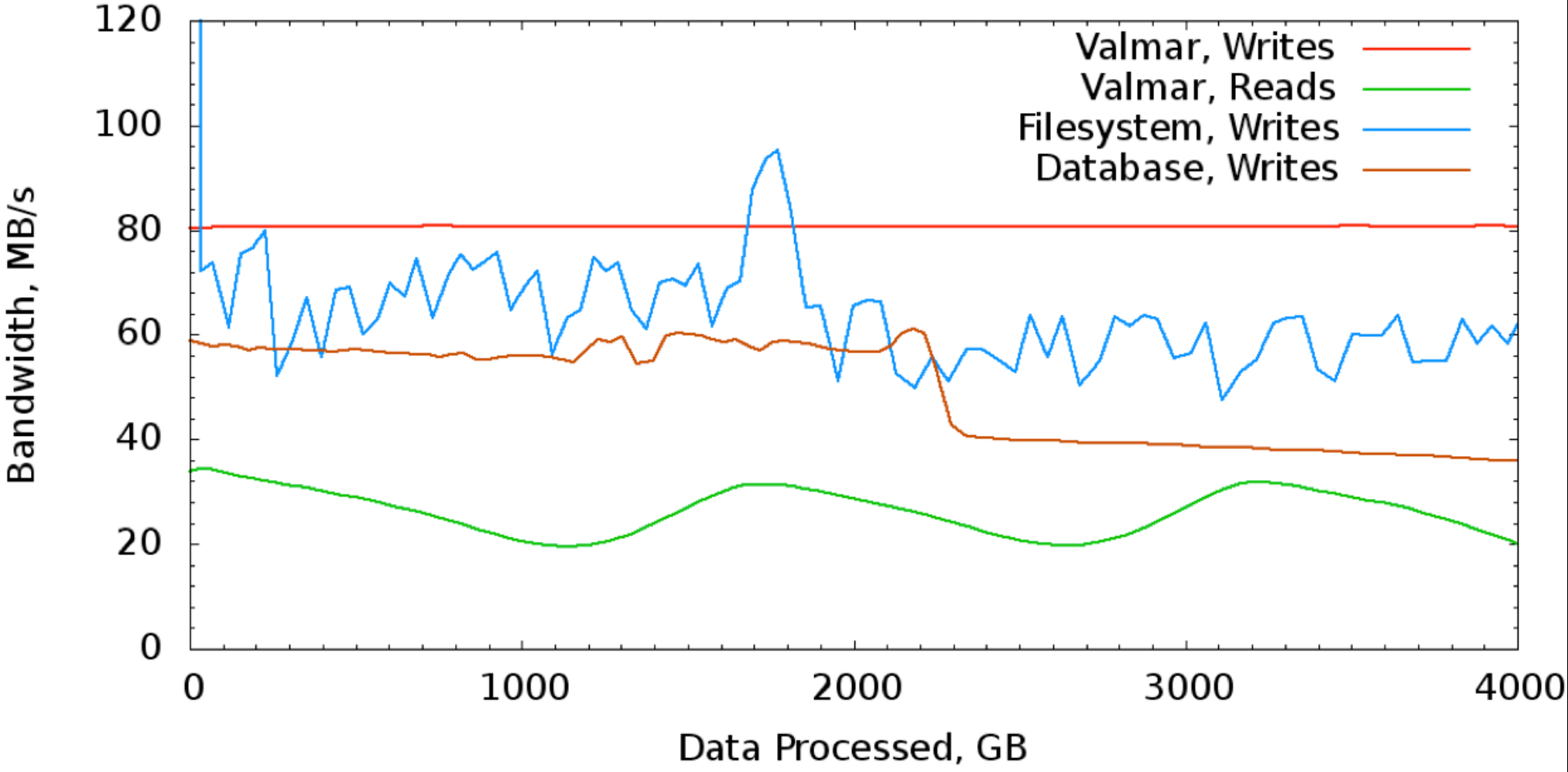
- Fixed disk size
- Fixed data rate and write size
- Limited data lifetime
  - Expires automatically
- Highly predictable
- In-place preservation
- Indexing on the fly
  - Maintained in memory
  - Archived with data
- Gang together as needed



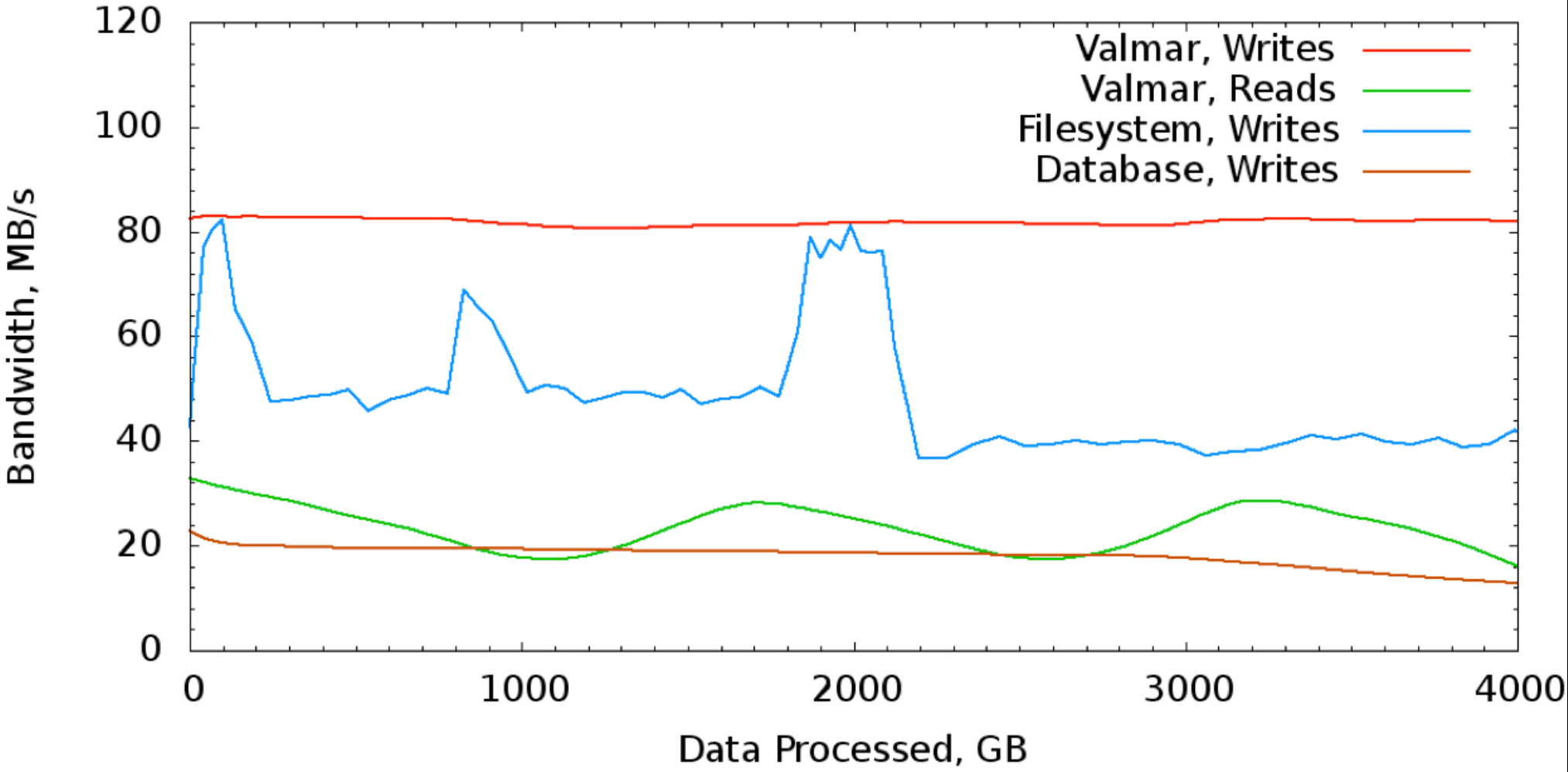

UC SANTA CRUZ

ISSDM

# Results for large elements



# Results for small elements




UC SANTA CRUZ

ISSDM

# Conclusion

- Distributed system performance management is needed
- It is challenging, yet feasible
- A unified approach seems to work well

- RAD is one basis for a unified solution
  - CPU, disk, network, buffer cache, system
- It is in use in real systems
- More in the works

Thank you!