RADIO: Managing the Performance of Large, Distributed Storage Systems

Scott A. Brandt
and
Carlos Maltzahn, Anna Povzner, Roberto Pineiro,
Andrew Shewmaker, and Tim Kaldewey
Computer Science Department
University of California Santa Cruz
and
Richard Golding and Ted Wong, IBM Almaden Research Center

UPC—July 7, 2009

Who am I?

- **Professor**, Computer Science Department, UC Santa Cruz
- **Director**, UCSC/LANL Institute for Scalable Scientific Data Management (ISSDM)
- **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
  - Real-time systems
  - Performance management and virtualization
  - Active object-based storage
- Other Research
  - Secure operating systems
  - Asynchronous circuits
  - Real-time image processing

Distributed systems need performance guarantees

- *Many* distributed systems and applications need (or want) I/O performance guarantees
  - Multimedia, high-performance simulation, transaction processing, virtual machines, service level agreements, real-time data capture, sensor networks, ...
  - Systems tasks like backup and recovery
  - Even so-called best-effort applications
- Providing such guarantees is difficult because it involves:
  - Multiple interacting resources
  - Dynamic workloads
  - Interference among workloads
  - Non-commensurable metrics: CPU utilization, network throughput, cache space, disk bandwidth

In a nutshell

- Big distributed systems
  - Serve many users/jobs
  - Process petabytes of data
- Data center design
  - Use rules of thumb
  - Over-provision
  - Isolate

- Ad hoc performance management approaches creates marginal storage systems that cost more than necessary
- A better system would guarantee each user the performance they need from the CPUs, memory, disks, and network

Outline

1. *Problem*: Managing the performance of large, distributed storage systems
2. *Approach*: End-to-end performance management
3. *Model*: RAD
4. *Instances*:
   - Disk
   - Network
   - Buffer cache
5. *Application*: Data Center Performance Management and Monitoring

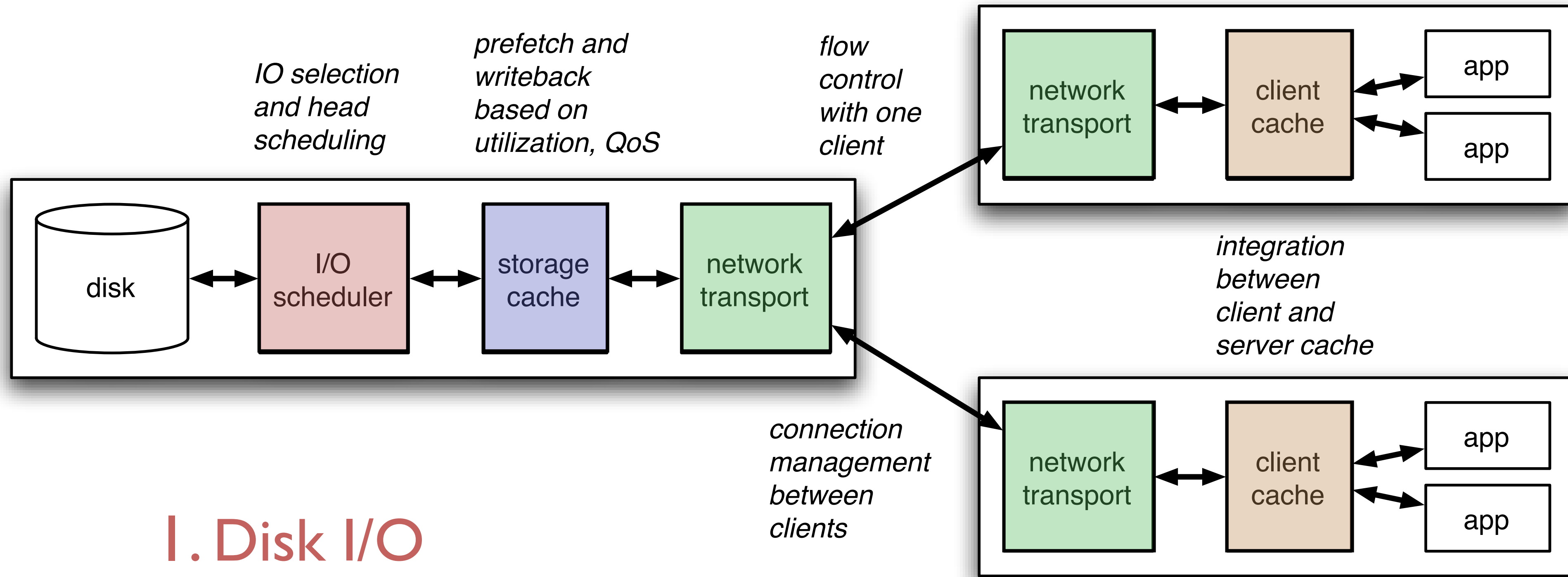
End-to-end I/O performance guarantees

- Goal: Improve end-to-end performance management in large distributed systems
  - Manage performance
  - Isolate traffic
  - Provide high performance
- Targets: High-performance storage (LLNL), data centers (LANL), satellite communications (IBM), virtual machines (VMware), sensor networks, ...
- Approach:
  1. Develop a uniform model for managing performance
  2. Apply it to each resource
  3. Integrate the solutions

Our current target

- High-performance I/O
  - From *client*, across *network*, through *server*, to *disk*
  - Up to hundreds of thousands of processing nodes
  - Up to tens of thousands of I/O nodes
  - Big, fat, network interconnect
  - Up to thousands of storage nodes with cache and disk
- Challenges
  - Interference between I/O streams, variability of workloads, variety of resources, variety of applications, legacy code, system management tasks, scale

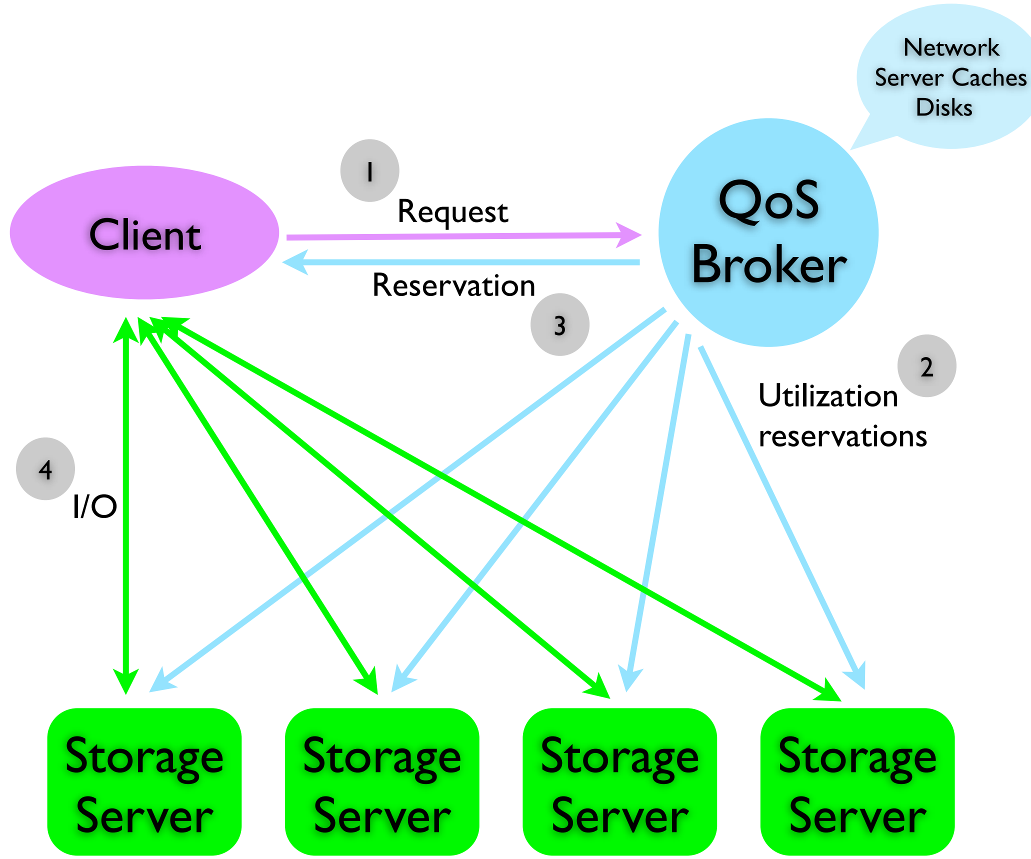
Stages in the I/O path



1. Disk I/O
2. Server cache
3. Flow control across network
    - Within one client's session *and* between clients
4. Client cache

System architecture

- Client: Task, host, distributed application, VM, file, ...
- Reservations made via broker
  - Specify workload: throughput, read/write ratio, burstiness, etc.
- Broker does admission control
  - Requirements + workload are translated to utilization
  - Utilizations are summed to see if they are feasible
  - Once admitted, I/O streams are guaranteed (subject to workload adherence)
- Disk, caches, network controllers maintain guarantees



Achieving robust guaranteeable resources

- Goal: Unified resource management algorithms capable of providing
  - Good performance
  - Arbitrarily hard or soft performance guarantees with
    - Arbitrary resource allocations
    - Arbitrary timing / granularity
  - Complete isolation between workloads
  - All resources: CPU, disk, network, server cache, client cache

➡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

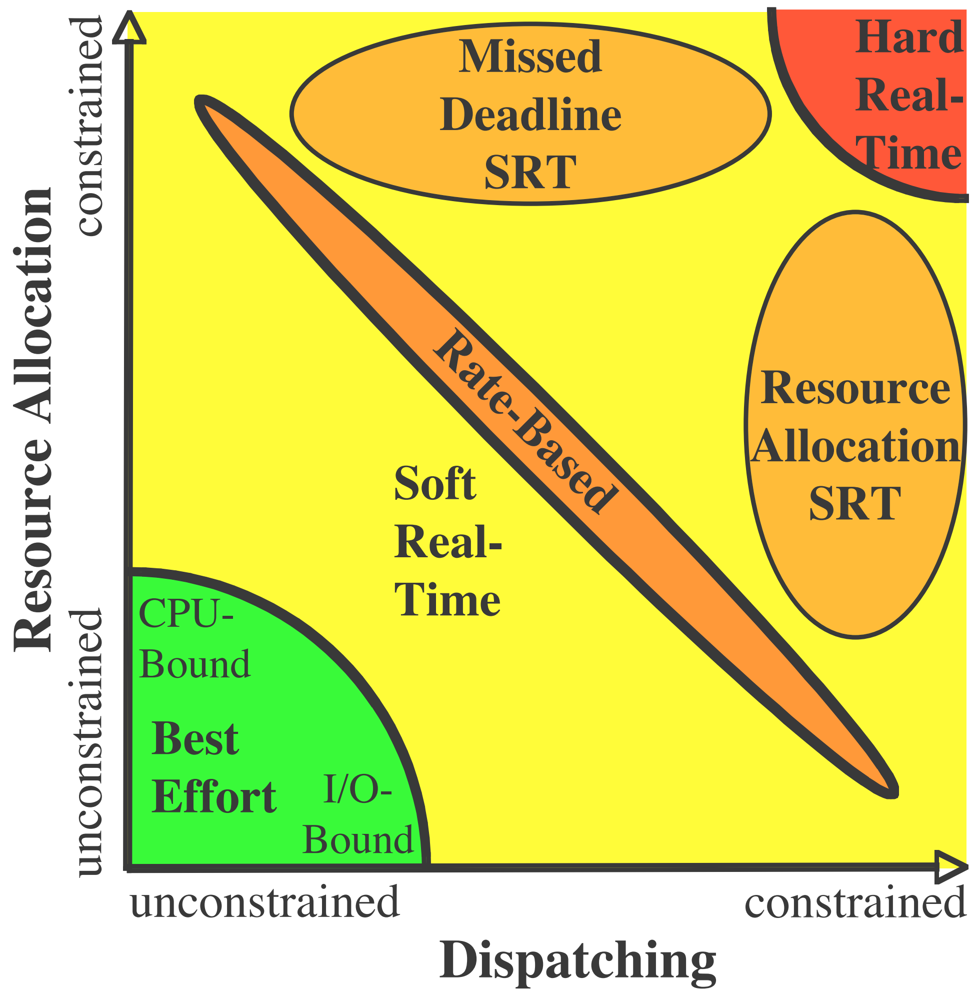
Observation

- 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

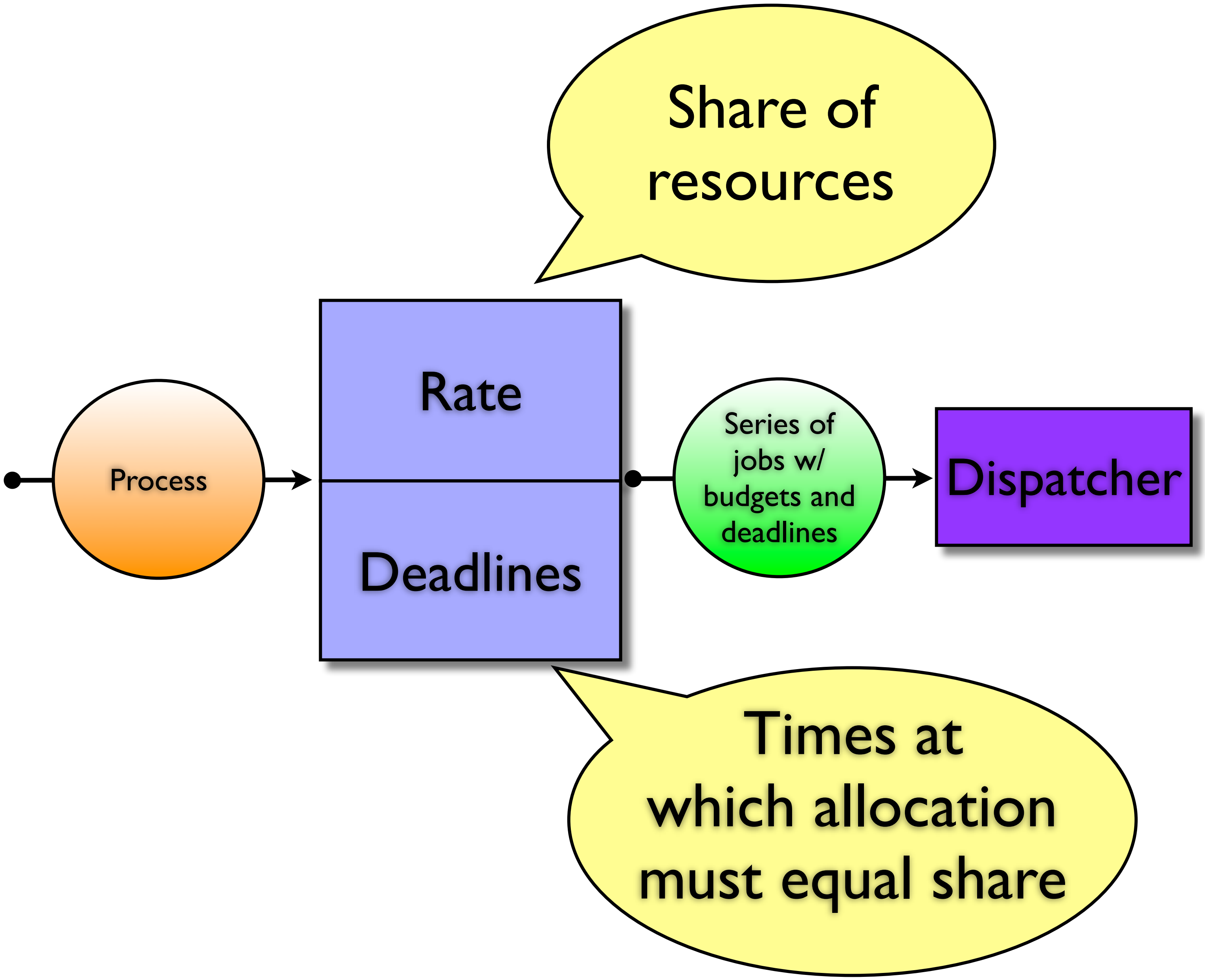
Separating them is powerful!

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

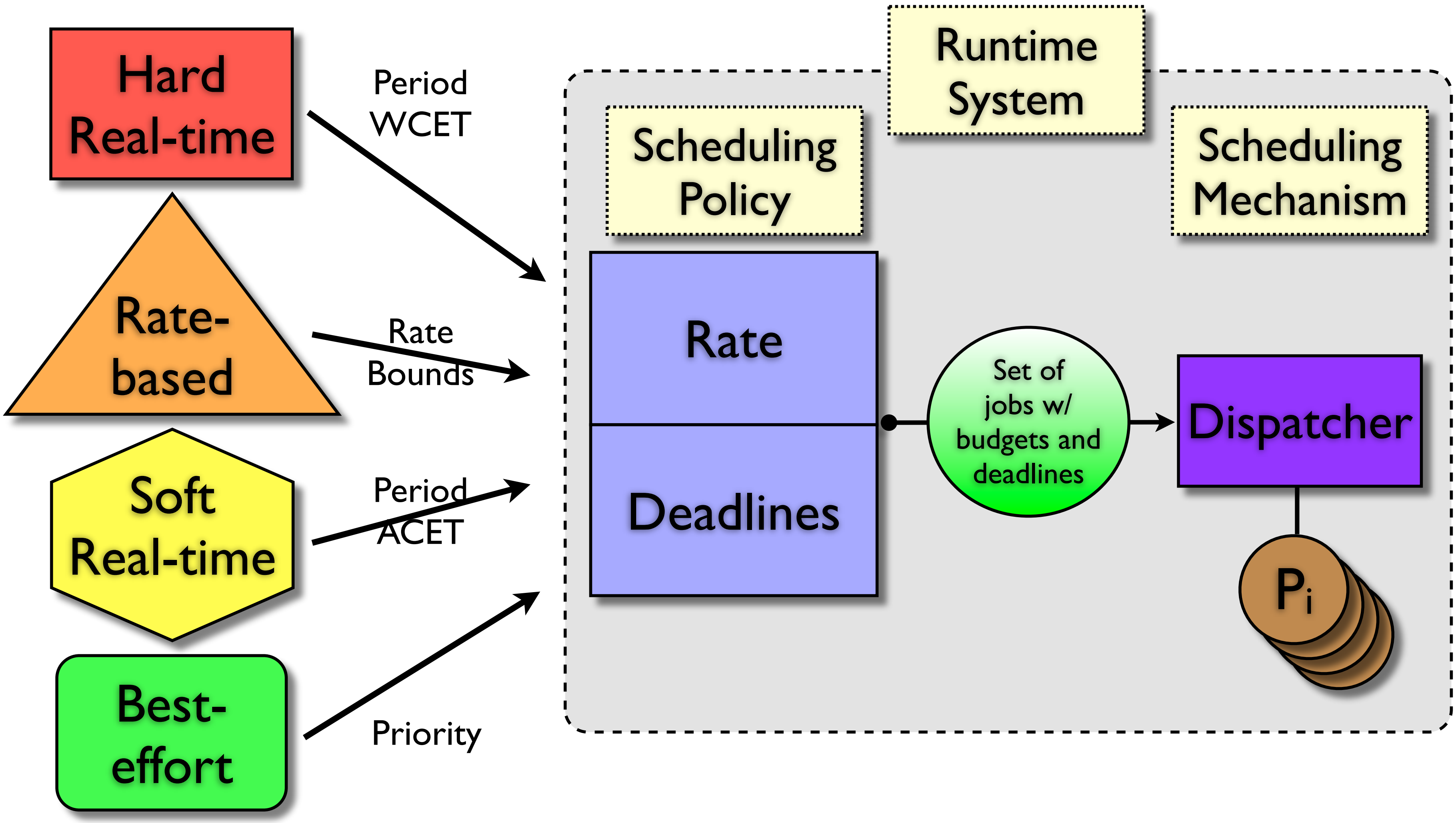
- Enables direct, integrated support of all types of timeliness needs



The resource allocation/dispatching (RAD) scheduling model

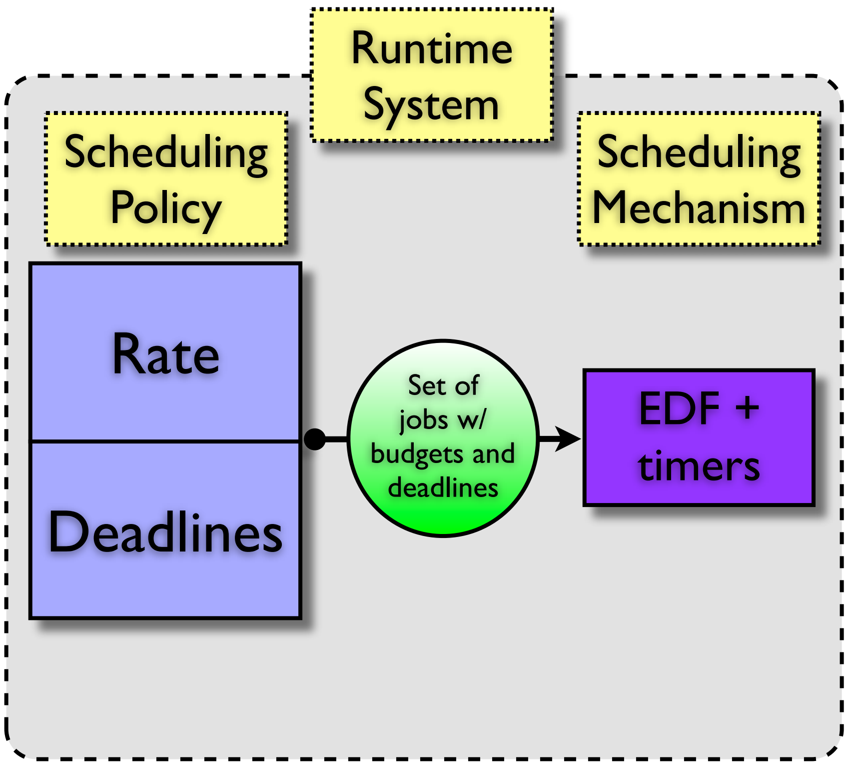


Supporting different timeliness requirements with RAD



Rate-Based Earliest Deadline (RBED) CPU scheduler

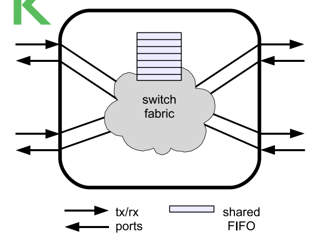
- Processes have rate & period
  - $\sum$rates $\leq$ 100%
  - Periods based on processing characteristics, latency needs, etc.
- Jobs have budget & deadline
  - budget = rate * period
  - Deadlines based on period or other characteristics
- Jobs dispatched via Earliest Deadline First (EDF)
  - Budgets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



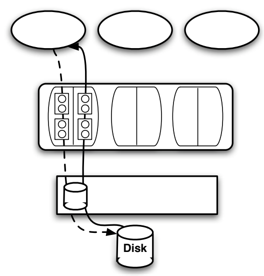
Adapting RAD to disk, network, and buffer cache

- Fahrrad—Guaranteed disk request scheduling
  Anna Povzner (UCSC)

- RADoN—Guaranteeing storage network performance
  Andrew Shewmaker (UCSC and LANL)



- Radium—Buffer management for I/O guarantees
  Roberto Pineiro (UCSC)

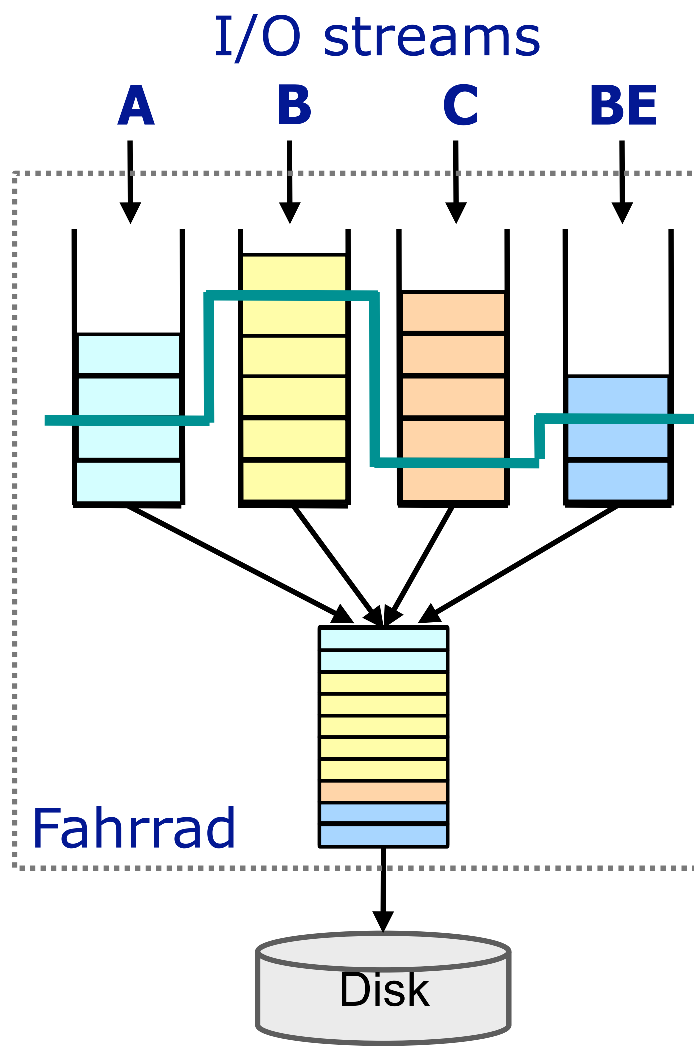


Guaranteed disk request scheduling

- 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

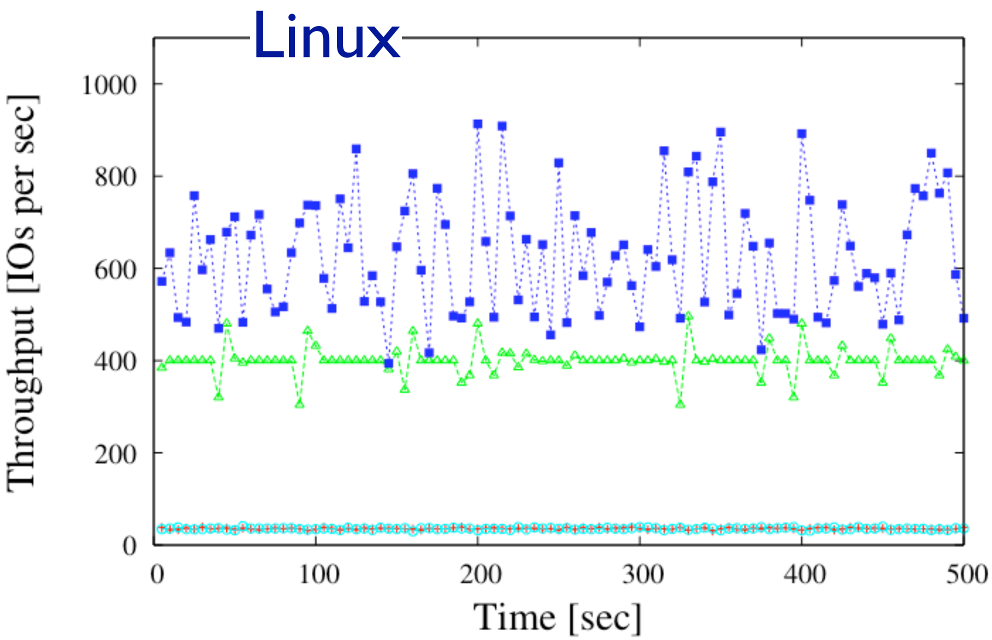
- Manages disk *time* instead of disk *throughput*
- Adapts RAD/RBED to disk I/O
- Reorders aggressively to provide good performance, without violating guarantees

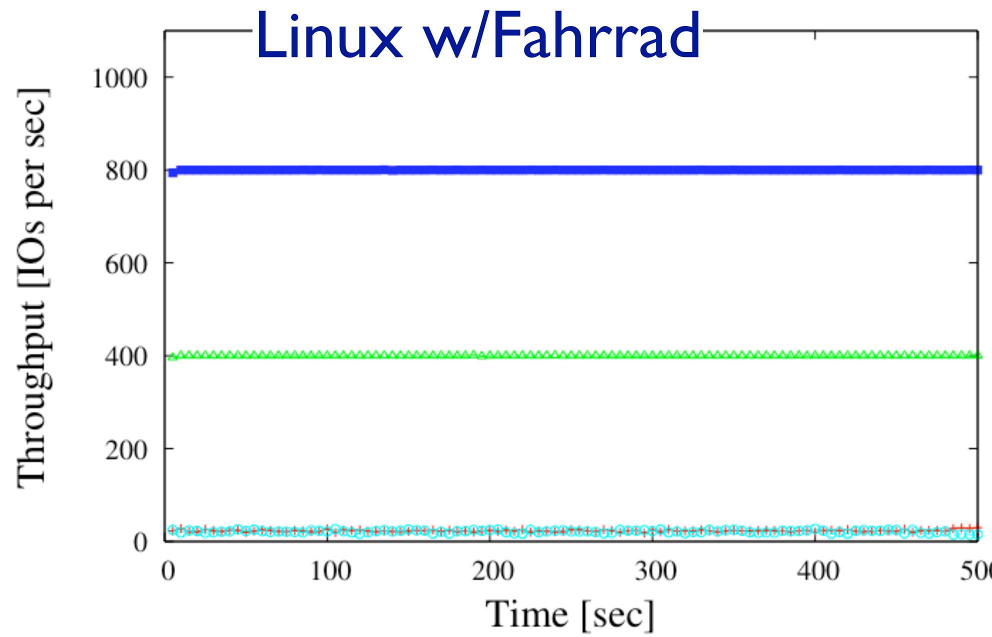


A bit more detail

- Reservations in terms of *disk time utilization* and *period* (granularity)
- All I/Os feasible before the earliest deadline in the system are moved to a Disk Scheduling Set (DSS)
- I/Os in the DSS are issued in the most efficient way
- I/O charging model is critical
- Overhead reservation ensures exact utilization
  - 2 WCRTs per period for "context switches"
  - 1 WCRT per period to ensure last I/Os
  - 1 WCRT for the process with the shortest period due to non-preemptability

Fahrrad outperforms Linux

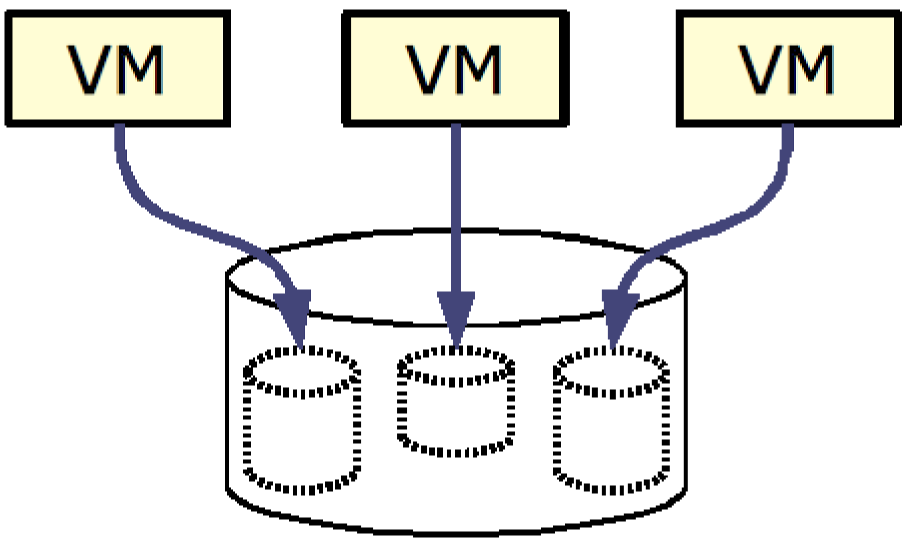




- Workload
  - Media 1: 400 sequential I/Os per second (20%)
  - Media 2: 800 sequential I/Os per second, (40%)
  - Transaction: short bursts of random I/Os at random times (30%)
  - Background: random (10%)
- Result: Better isolation AND better throughput

New work: virtual disks

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



- LUNs virtualize storage capacity
- Fahrrad virtualizes storage performance

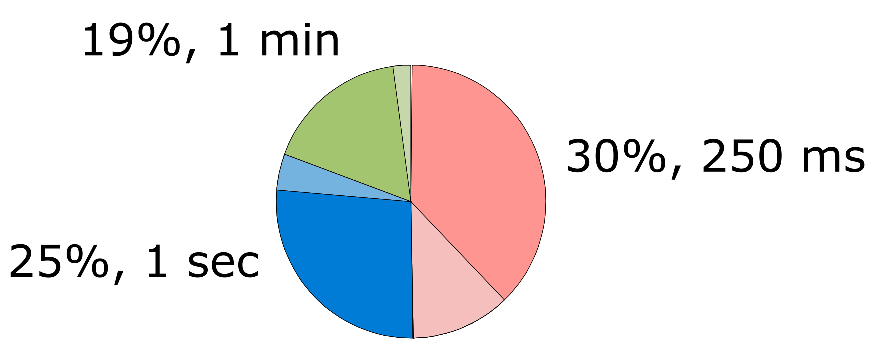
Fahrrad virtual disks

- Implemented with the Fahrrad real-time I/O scheduler
- Guarantee reserved and isolated share of the time on storage device
  - Hard guarantees on performance isolation
  - Virtual disk throughput same as equivalent standalone throughput
- Amount of data transferred:
  - $\forall i, \; D_i(\underline{x\%}, \underline{t}) \; = \; D_i(\underline{100\%}, \underline{x\% \cdot t})$

Share of disk — Time — Share of disk — Time

Guaranteeing performance isolation

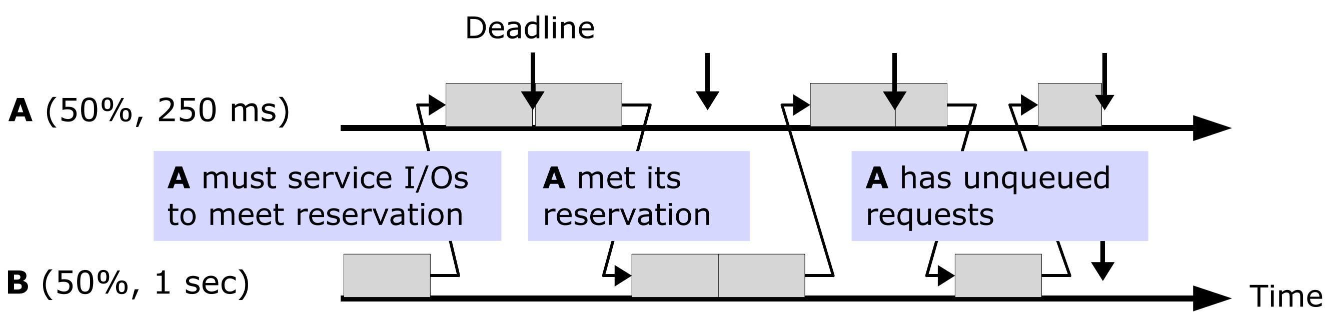
- Virtual disk reservation: *disk share (utilization)* and *time granularity (period)*
  - Account for all extra (inter-stream) seeks
  - Reserve overhead utilization to do them
  - Charge each I/O stream for all of the time it uses, including inter- and intra-stream seeks
  - Reservation = Disk Share + Overhead utilization



Disk time

Extra seeks

- Intra-stream seeks caused by workload
- Inter-stream seeks caused by reservations
  - Low time granularity causes more frequent seeks
  - At most **two** extra seeks per stream per period
    - **To** and **away** from the stream to meet deadlines



➡Extra seeks caused by un-queued requests

Charging model and reservations

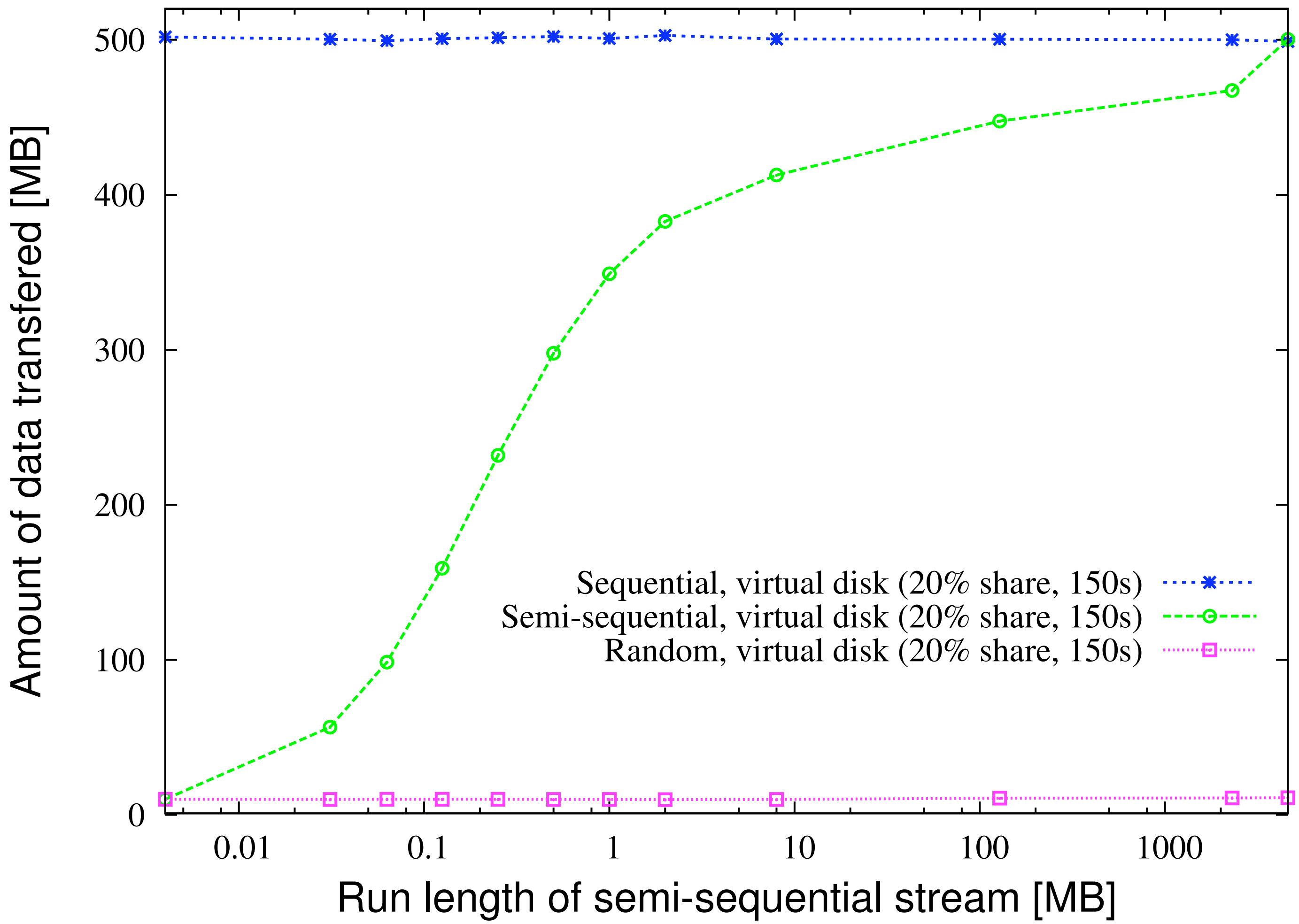
- Charge streams responsible for inter-stream seeking
  - From overhead: for seeks caused by reservations
  - From reservation: for seeks caused by bursty behavior
  - Overhead utilization needed for hard guarantees
  - Overhead utilization = WCRT/p + 2*WCRT/p + WCRT/p

WCRT/p	2*WCRT/p	WCRT/p
Guarantee reserved utilization	Account for inter-stream seeks	Maintain 2 outstanding requests

- We can trade-off hard guarantees for lower overhead by assuming less than worst-case request time

Performance: guaranteeing throughput

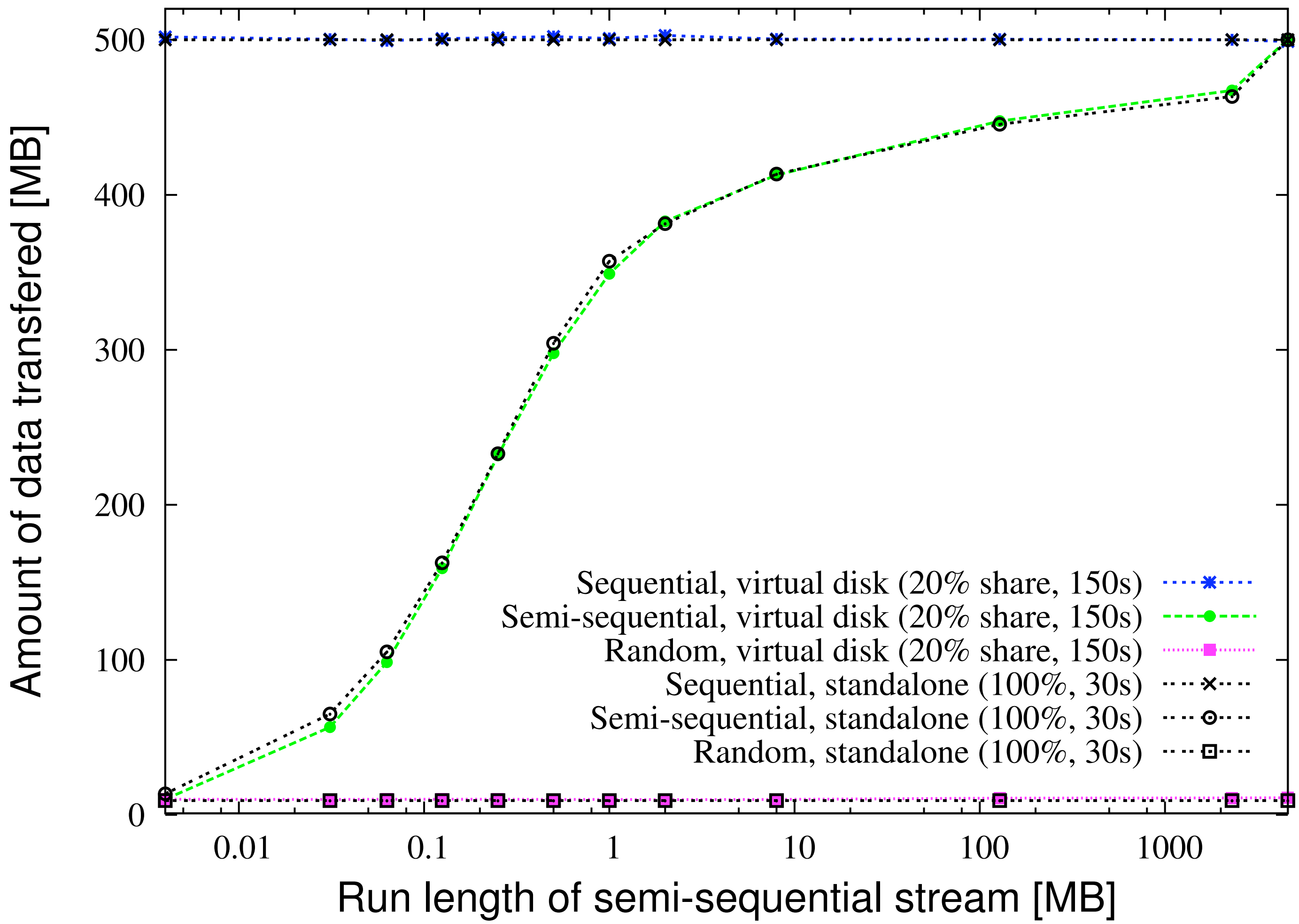
- Throughput is determined by reservation and workload



Each virtual disk reserves 20% with 1 second granularity

Performance: guaranteeing throughput

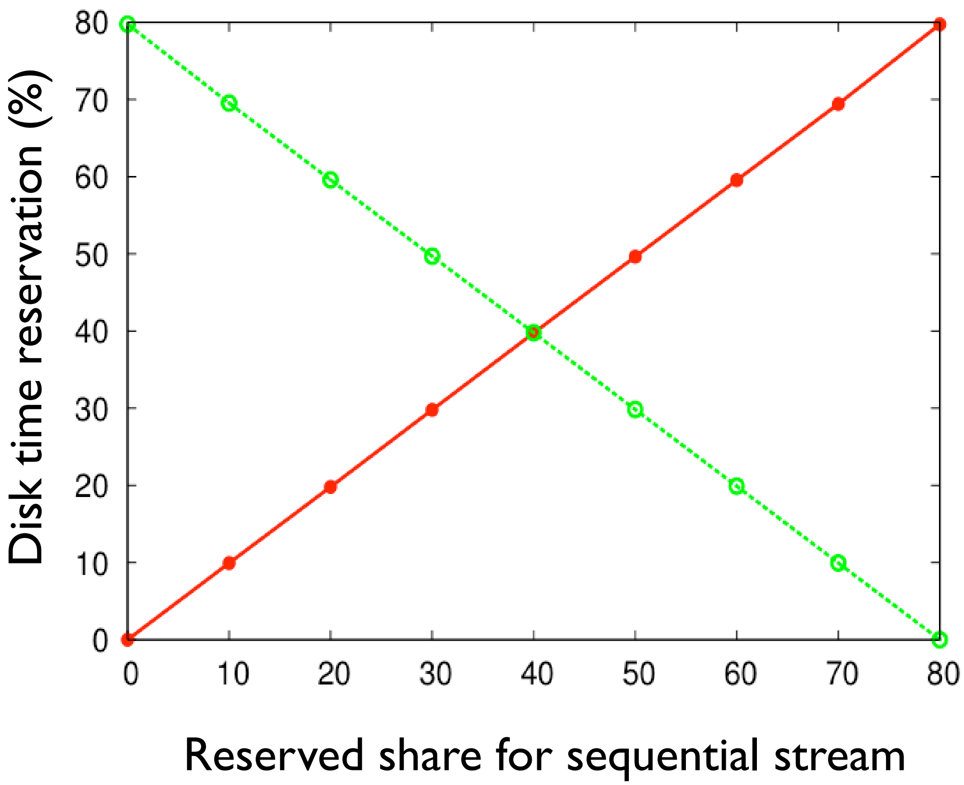
- Throughput is determined by reservation and workload

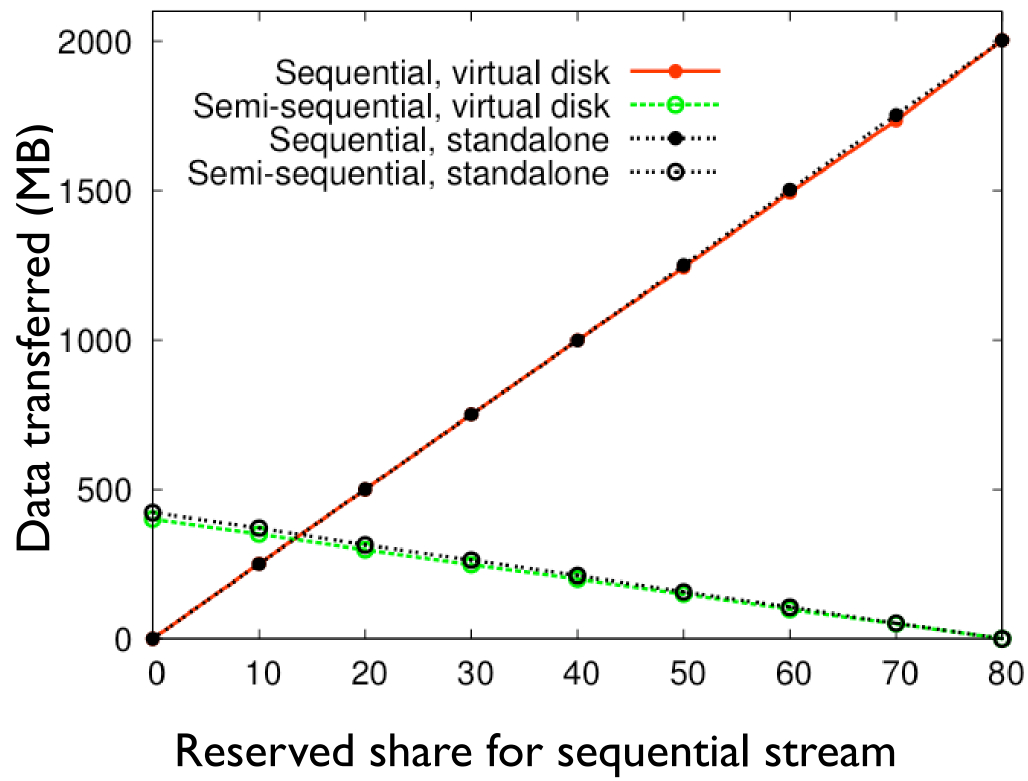


Each virtual disk reserves 20% with 1 second granularity

Performance: Controlling throughput

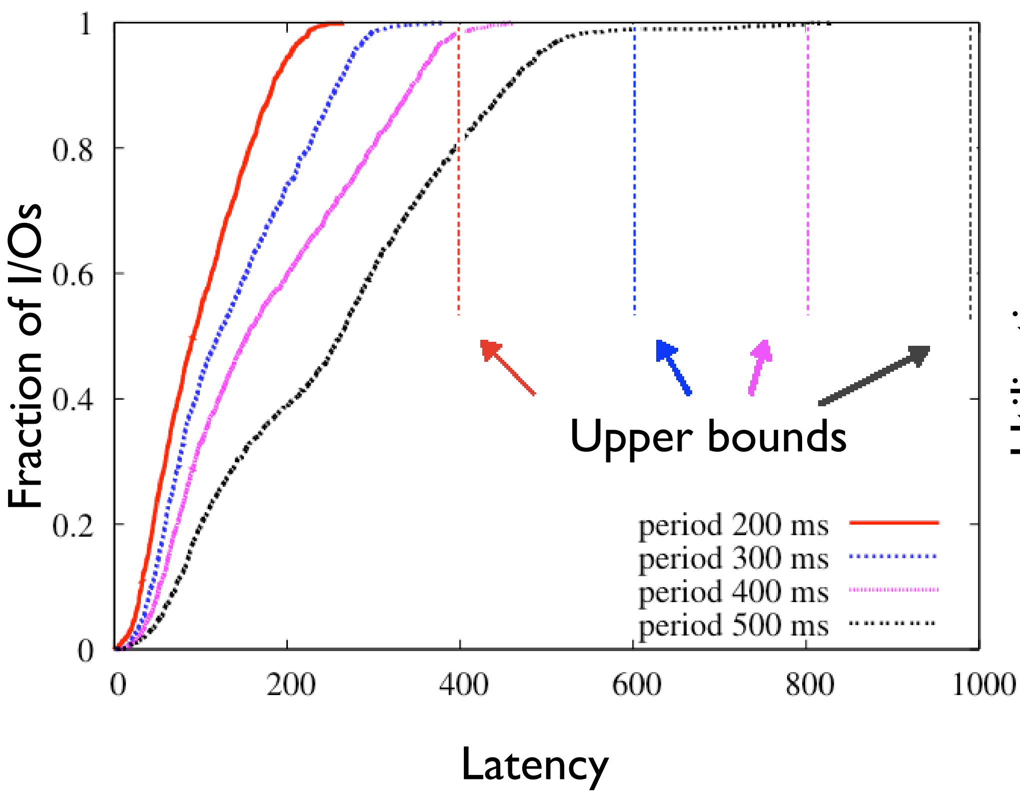
- Each virtual disk is isolated from the other
- Performance is fully determined by the reservation and workload

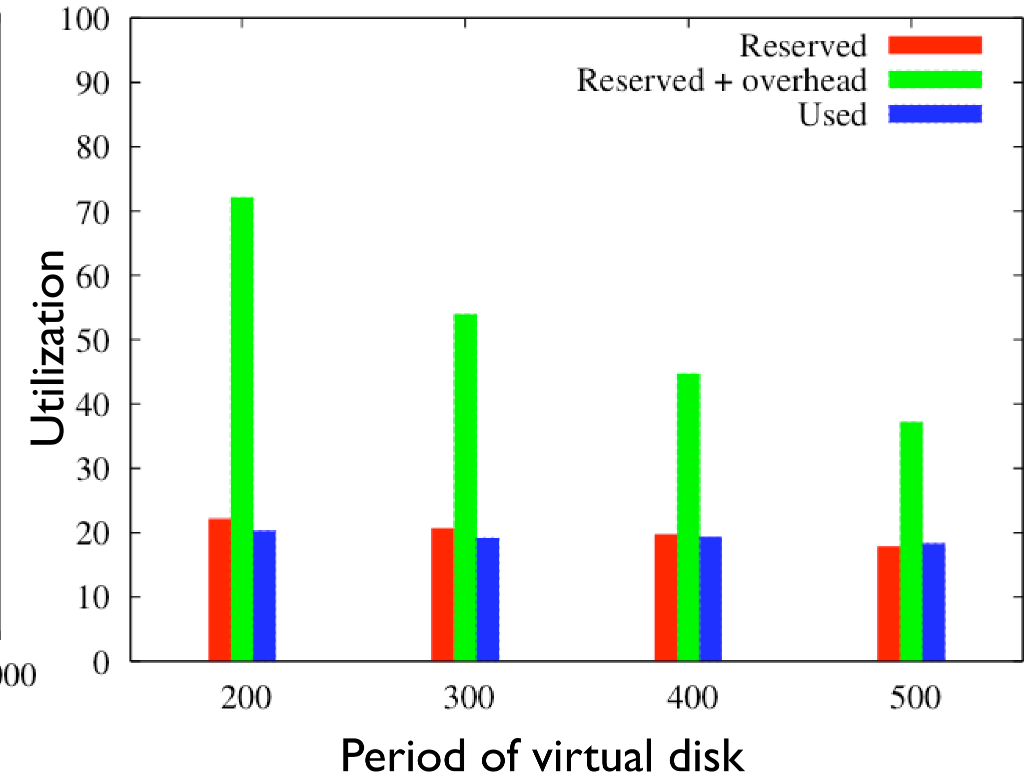




Performance: Controlling latency

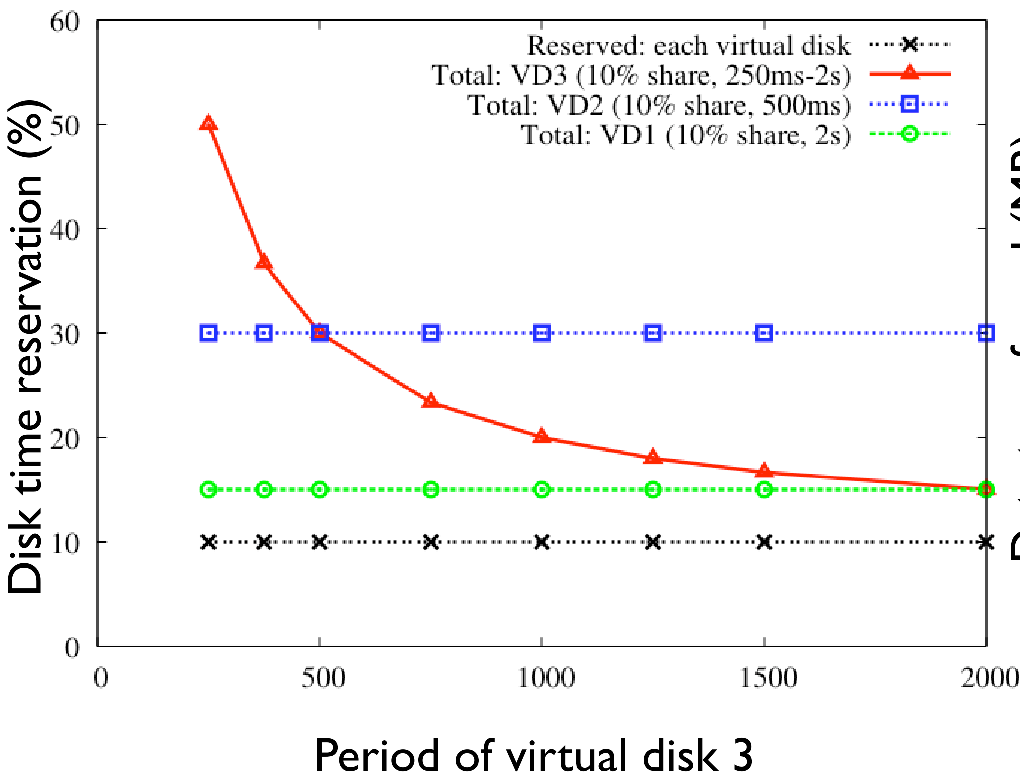
- Reservation granularity bounds latency:
  - period = latency/2
- Virtual device serves periodic semi-sequential stream and shares storage with random background stream. Four experiments for different period reservations.

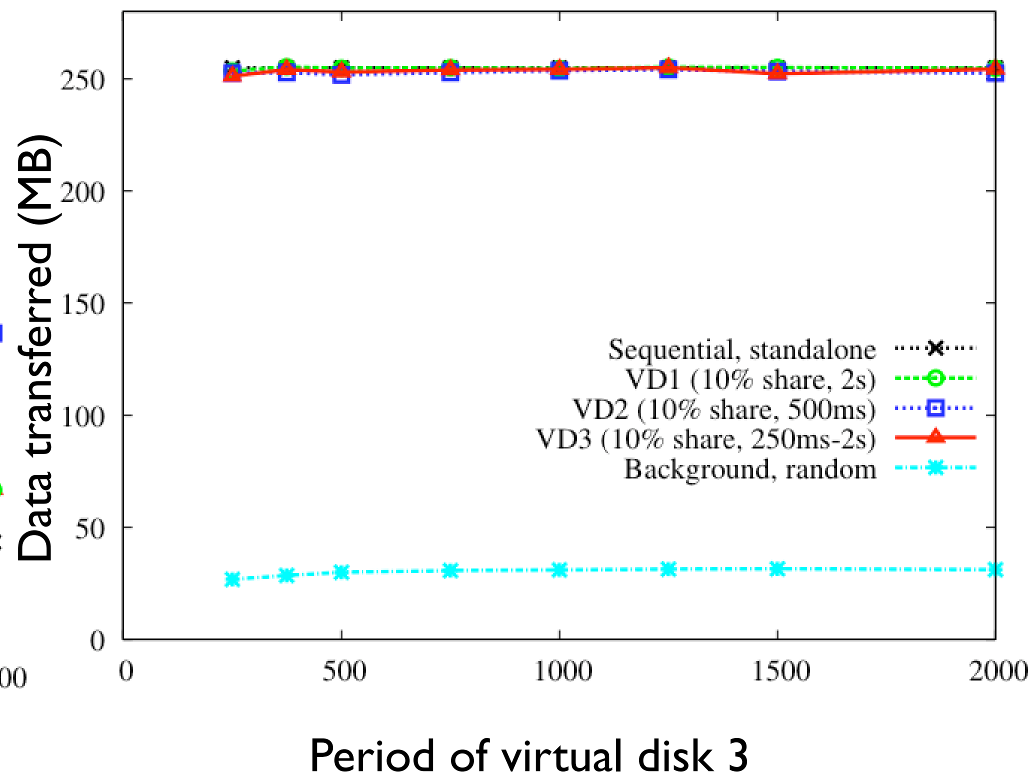




Performance: Isolation guarantees

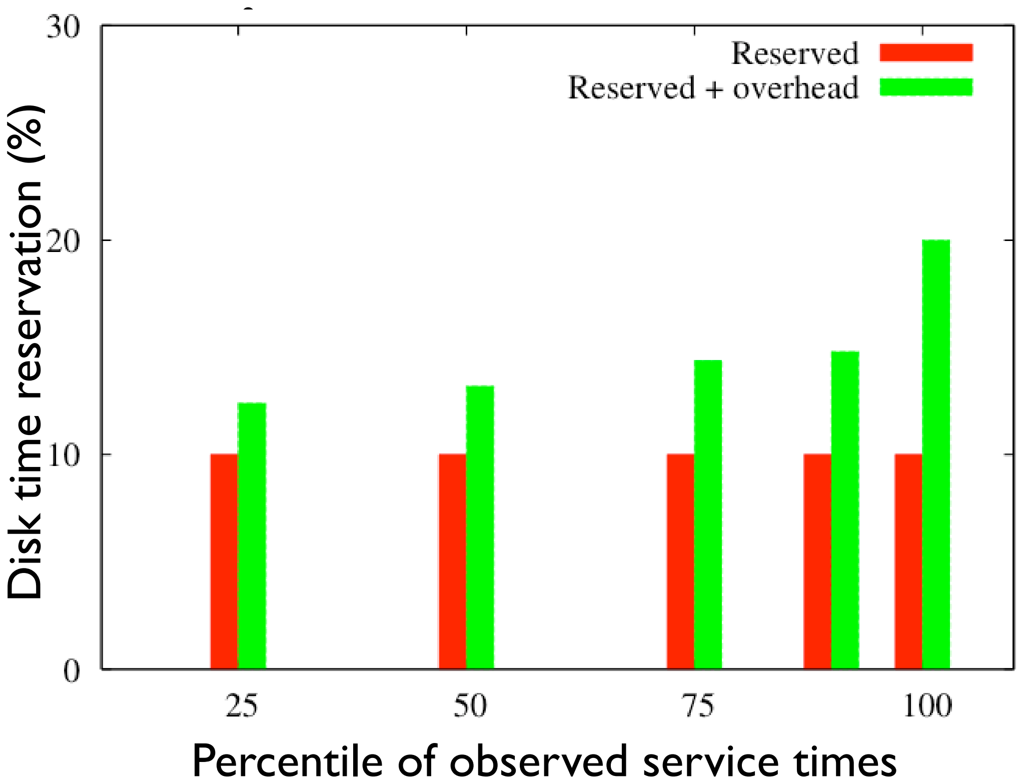
- Hard guarantees require high overhead (proportional to reservation granularity)
  - Three virtual disks each serving one sequential stream with many outstanding I/Os share a storage system with a random background stream.

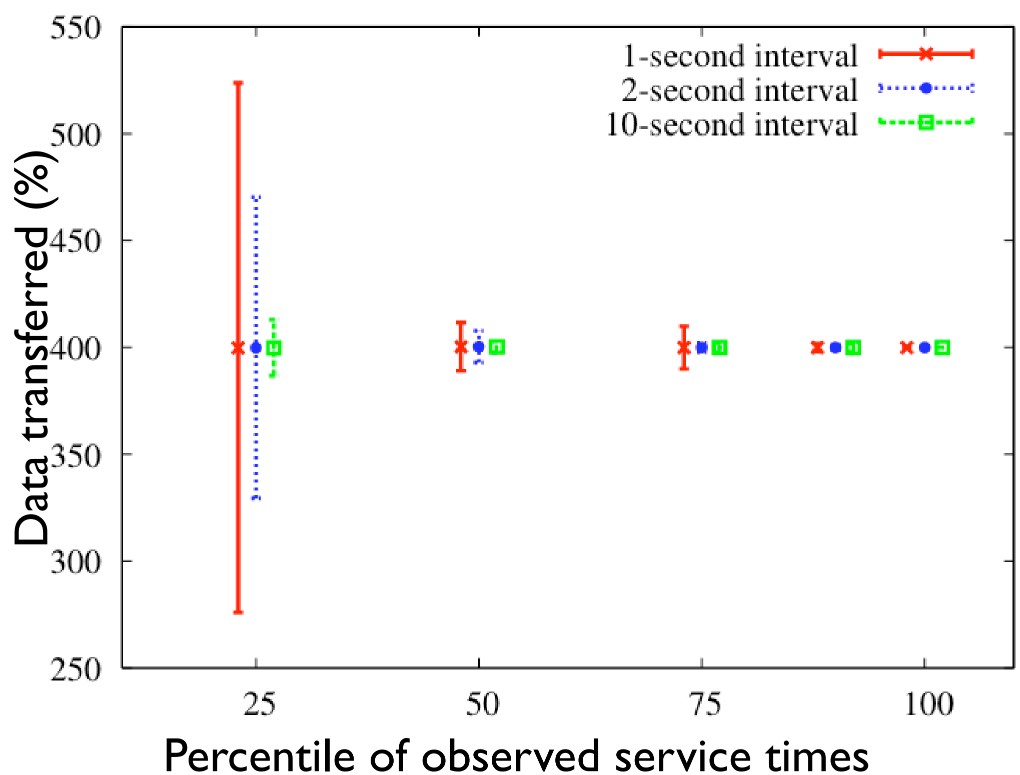




Performance: Soft guarantees w/isolation

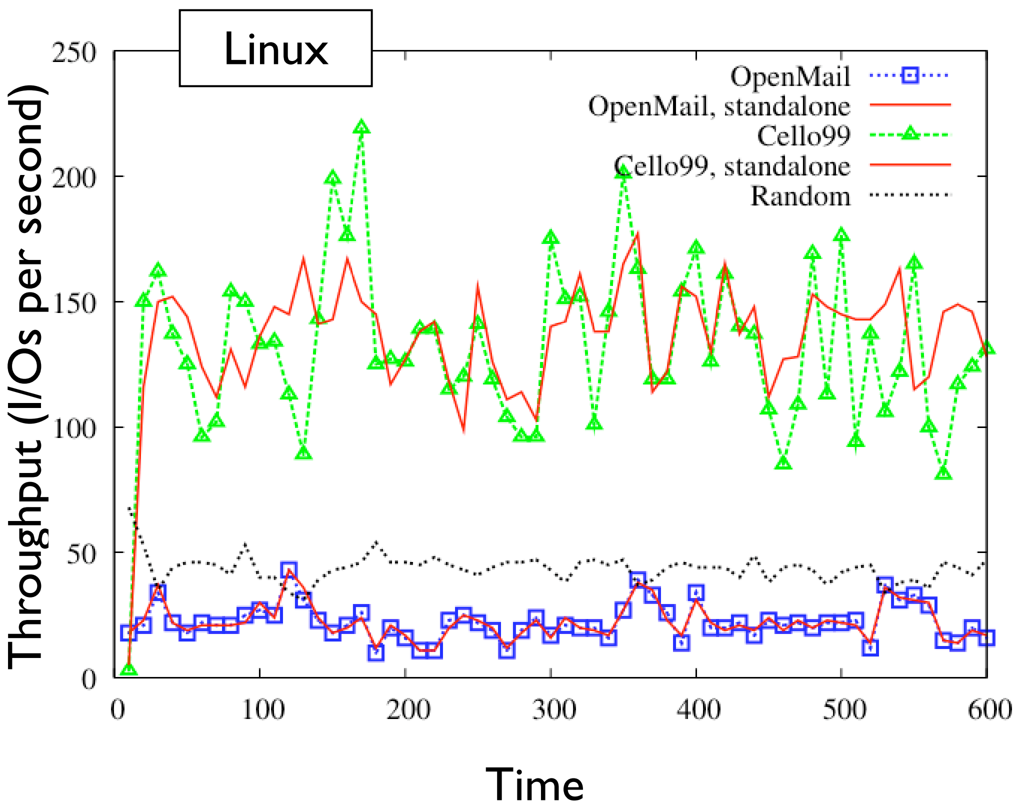
- Overhead based on less than worst-case I/O time
- Increased short term throughput variation
  - Virtual disk (10%, 1 sec) runs one sequential stream with 400 IO/sec arrival rate and shares the system with 5 virtual disks each running one random stream.

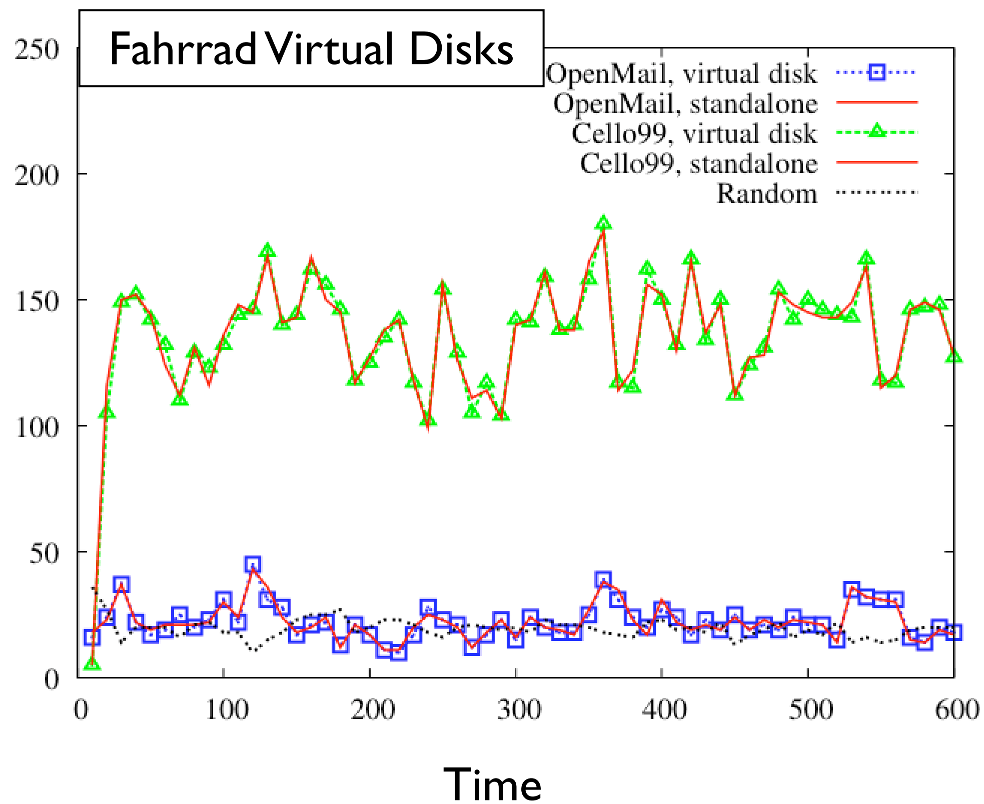




Performance: Soft guarantees w/isolation

- Linux fails to support Cello99 (variation up to 30% from standalone)
- Fahrrad Virtual Disks provide Cello99 and OpenMail performance close to standalone
- Cello99 and OpenMail virtual disks share the system with random background stream.

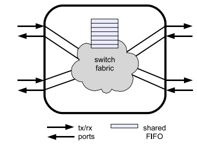




Fahrrad Virtual Disks

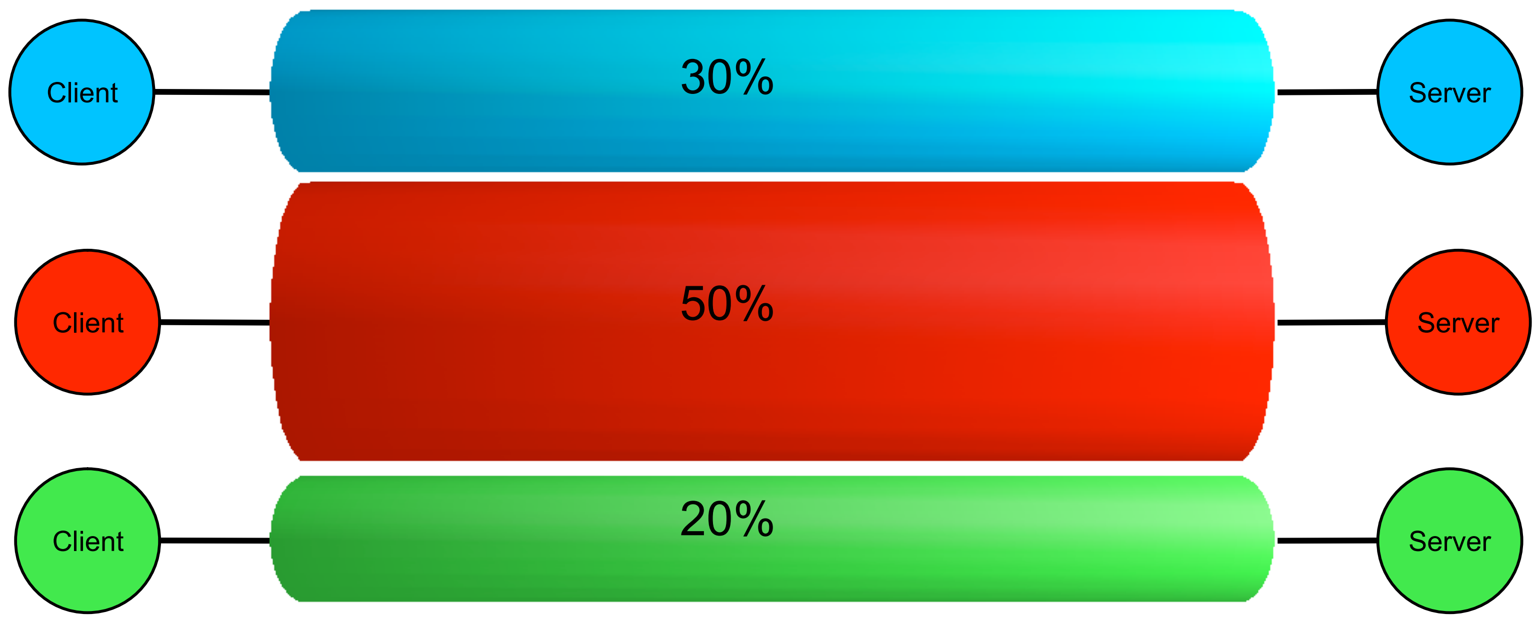
1. Guarantee throughput by accounting for overhead and guaranteeing utilization
2. Guarantee isolation between workloads by accurately accounting for all disk time
3. Provide high throughput (w/guarantees) by minimizing interference between workloads
4. Result: performance of virtual disk depends only on reservation, workload, and performance of device

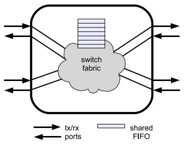
Guaranteeing storage network performance



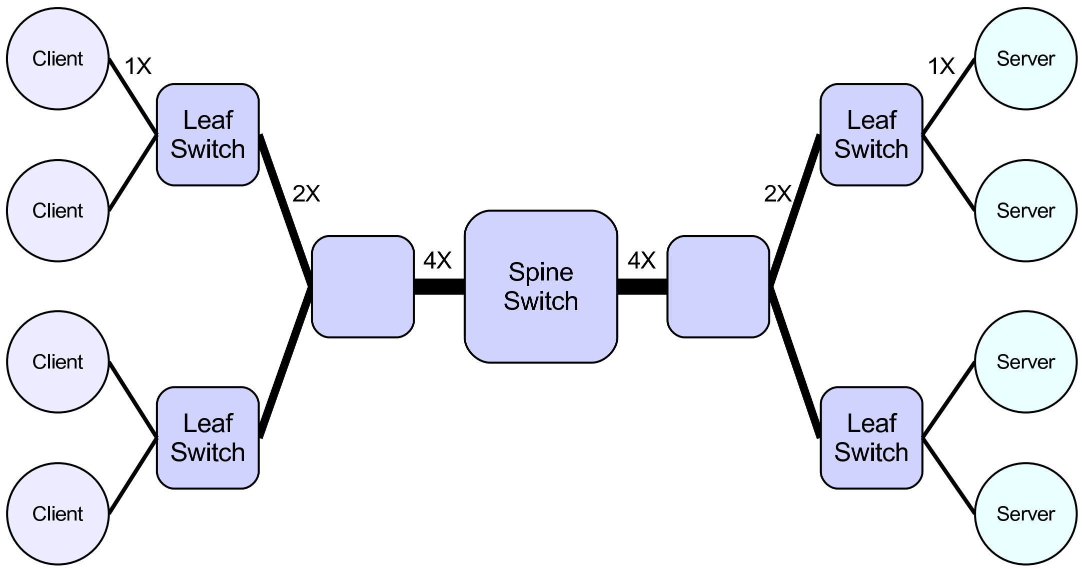
- Goals
  - Hard and soft performance guarantees
  - Isolation between I/O streams
  - Good I/O performance
- Challenging because network I/O is:
  - Distributed
  - Non-deterministic (due to collisions or switch queue overflows)
  - Non-preemptable
- Assumption: closed network

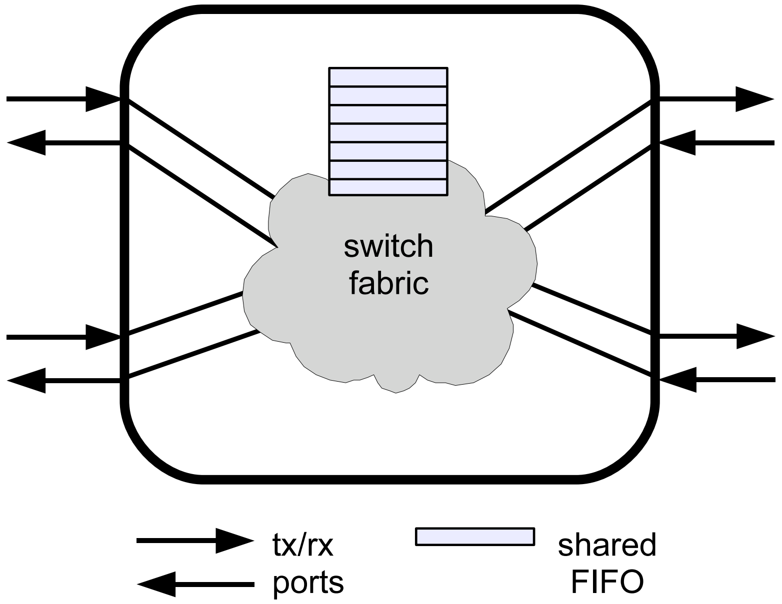
What we want



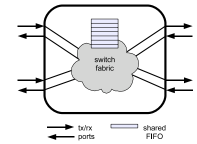


What we have



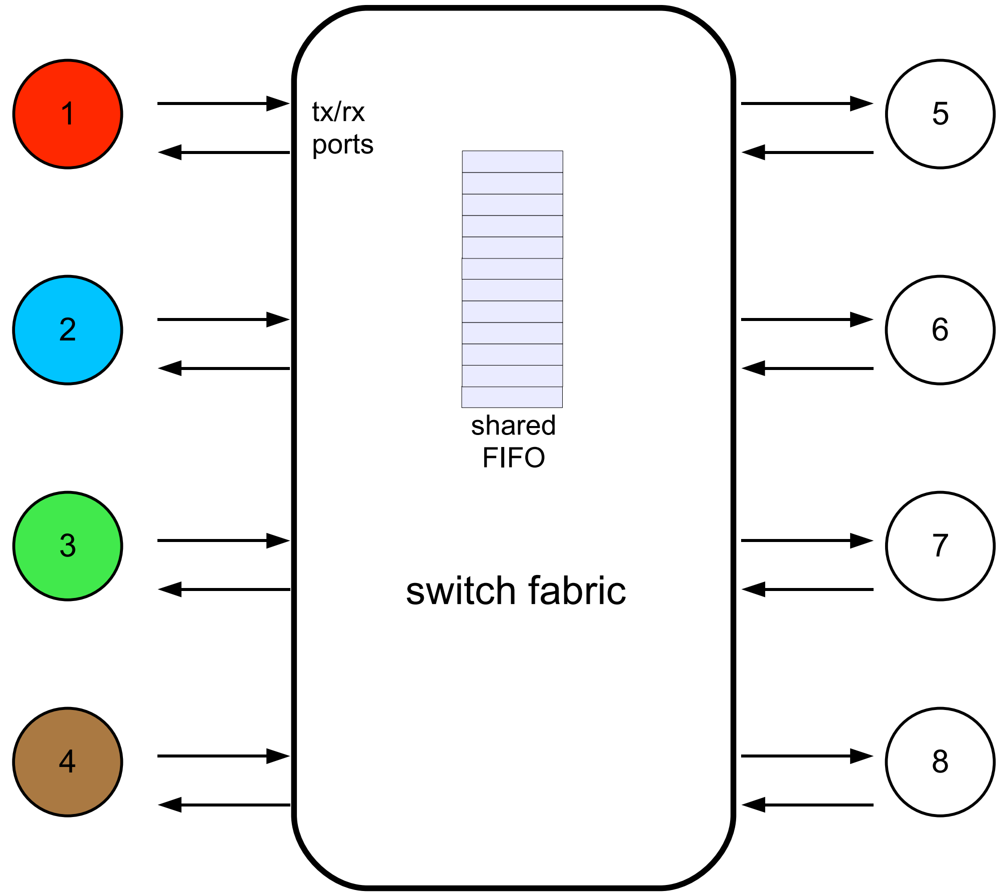


- Switched fat tree w/full bisection bandwidth
- Issue 1: Capacity of shared links
- Issue 2: Switch queue contention

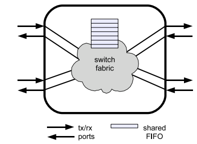


Congestion in a simple switch model

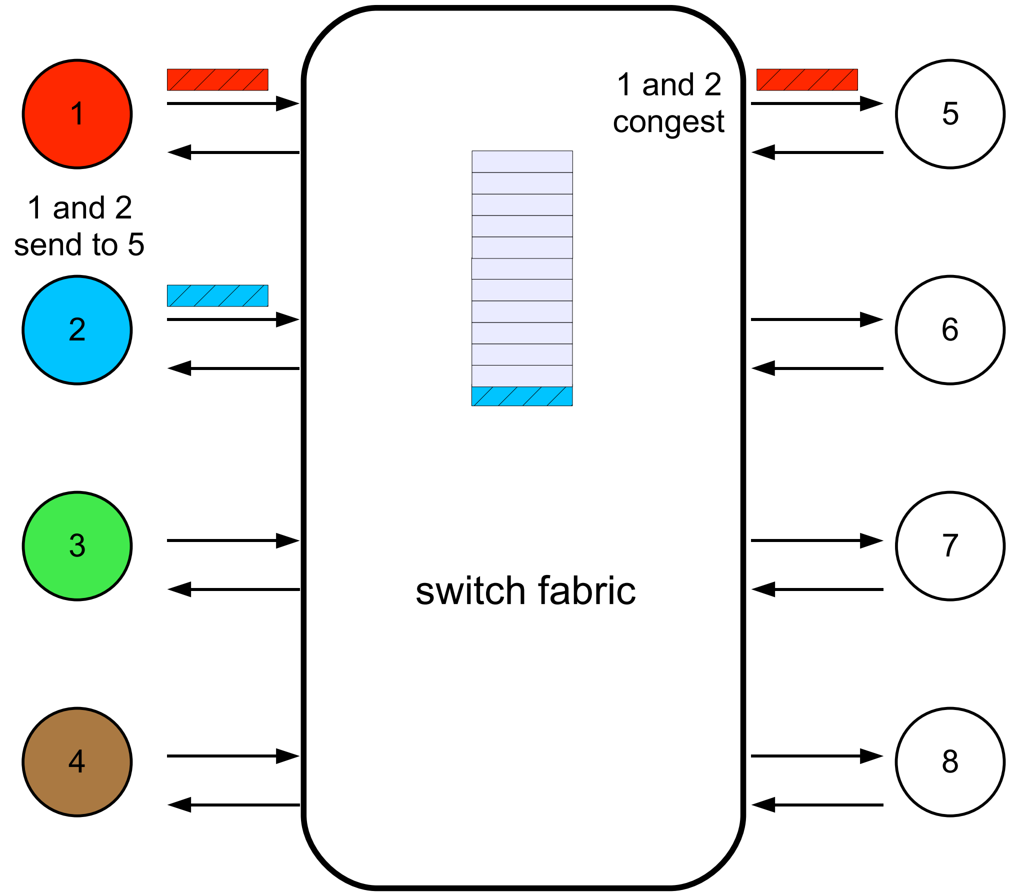
- Each transmit port on the switch is a collision domain

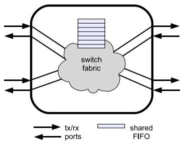


Congestion in a simple switch model



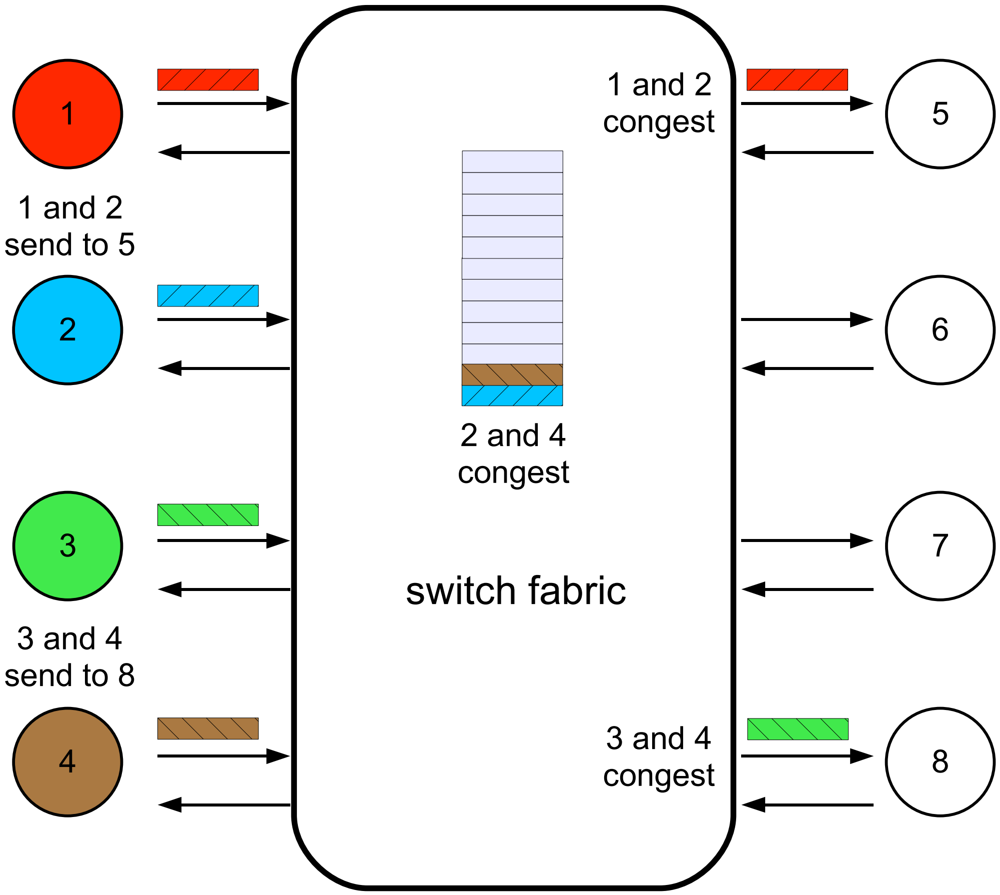
- One of the packets arriving at the same switch transmit port is delayed on the queue

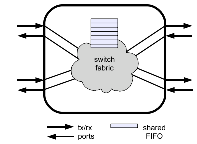




Congestion in a simple switch model

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



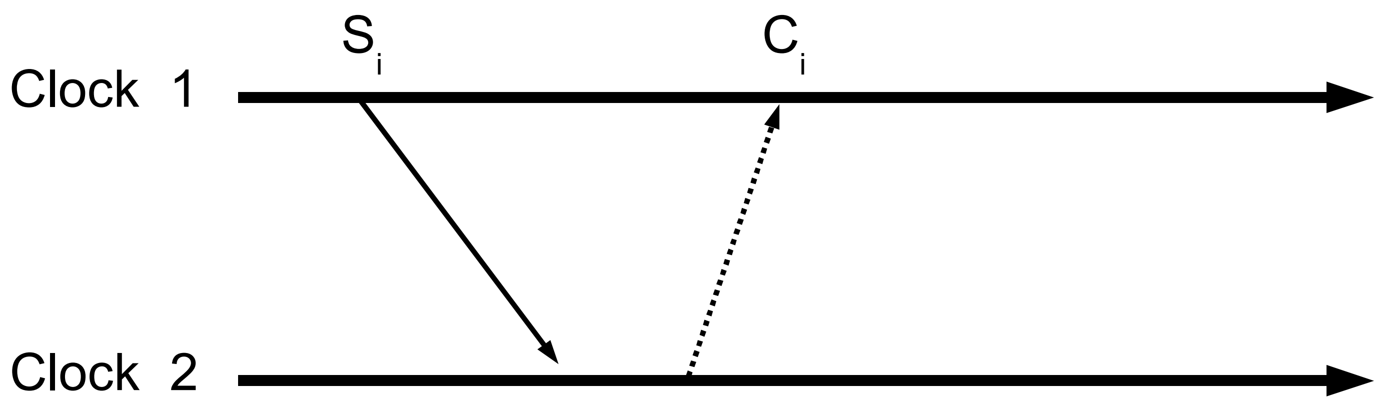


TCP

- Those who do not understand TCP are destined to reimplement it
    - Jon Postel

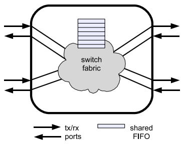
- Ack-clocked flow control
- Packet loss based congestion control
- Sawtooth throughput
- Incast throughput collapse

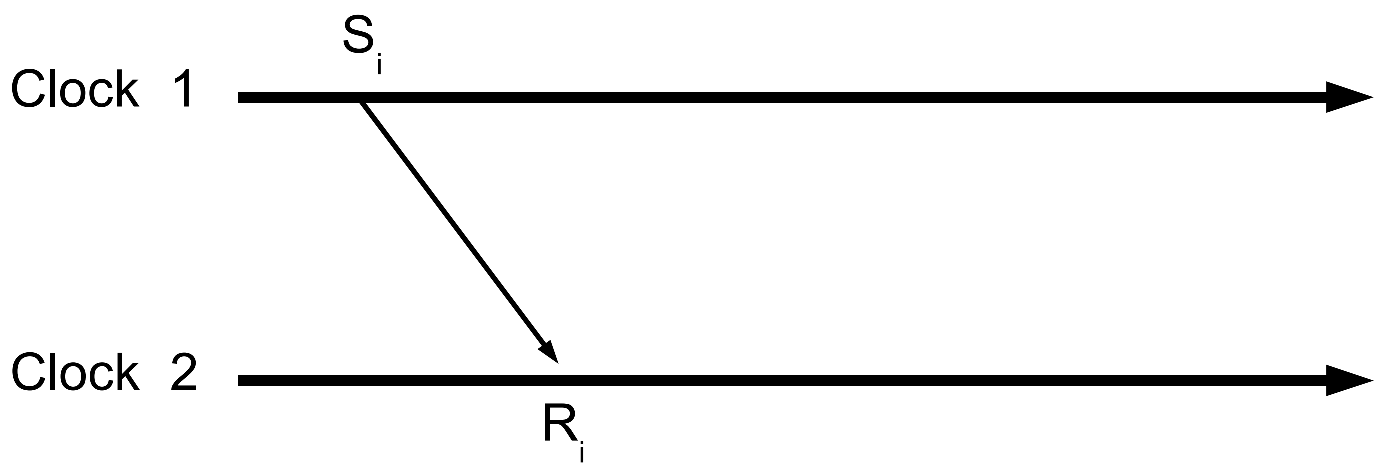
Network resource usage measurements



- Round trip time $RTT_i = C_i - S_i$
- Combines queueing effects on forward and reverse path + response time

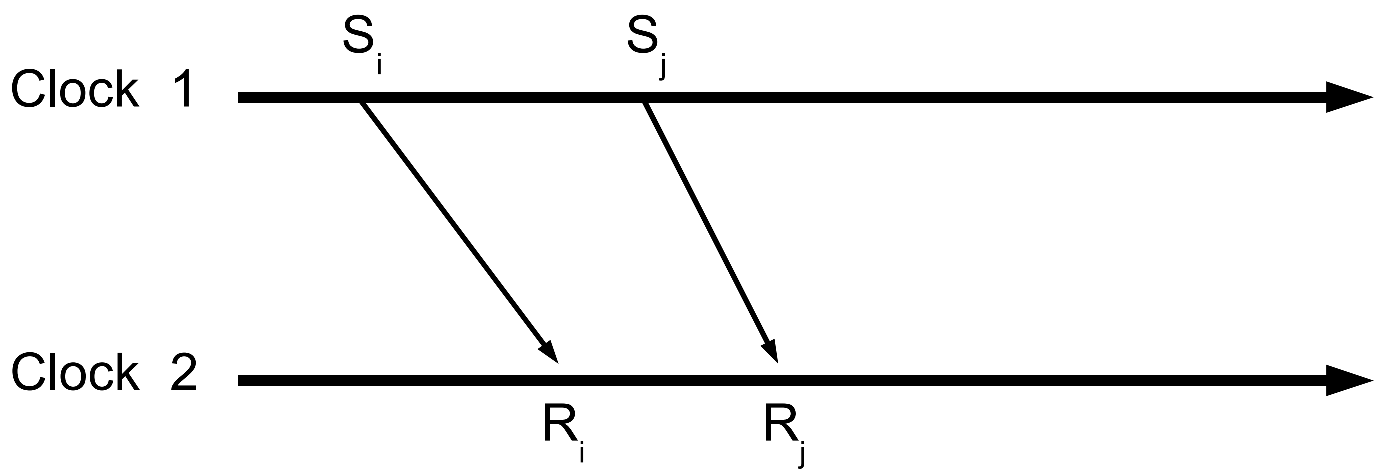
Network resource usage measurements





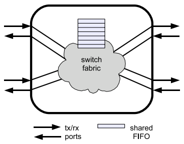
- One-way delay $OWD_i = R_i - S_i$
- Isolates queueing affects on forward path, **but**
- Requires synchronized clocks

Network resource usage measurements



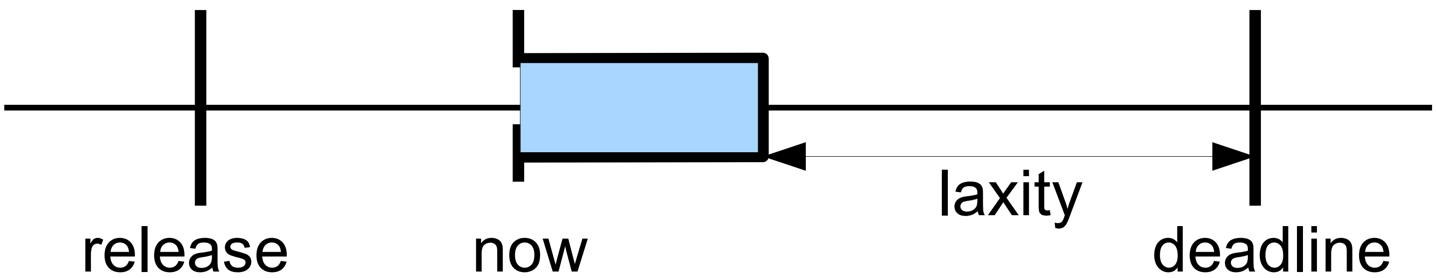
- Relative forward delay $RFD_{i,j} = (R_j - R_i) - (S_j - S_i)$
- Isolates queueing affects on forward path, **and**
- Does **not** require synchronized clocks
  - But they must be relatively stable

RADoN

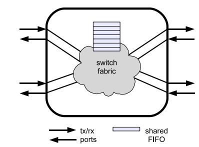


- A reservation has a *network share* (*utilization*) and a *time granularity (period)*

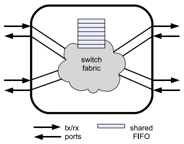
- Two real-time scheduling algorithms
  - Earliest Deadline First (EDF) - absolute deadlines
  - Least Laxity First (LLF) - relative laxities



Approximating optimal scheduling

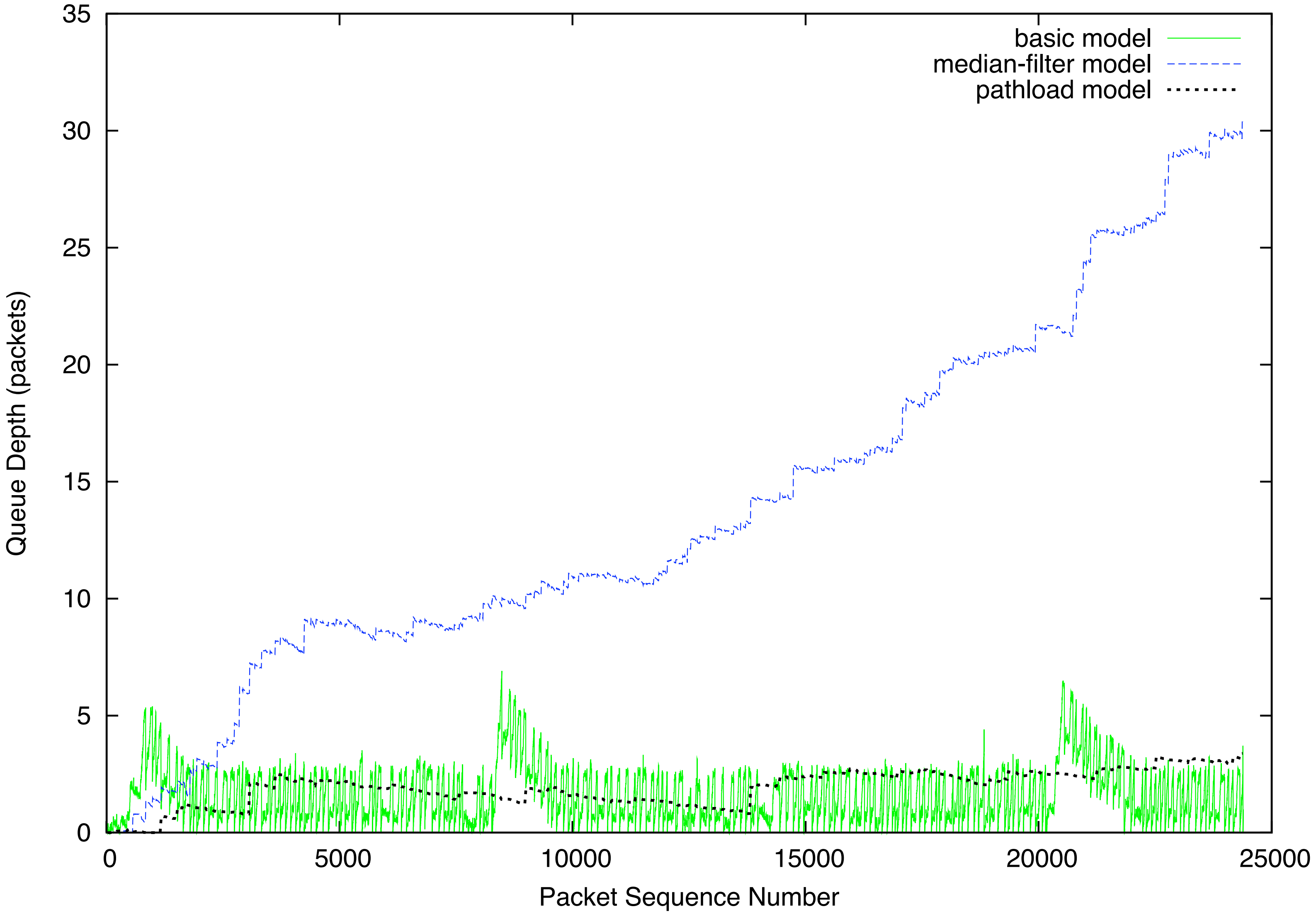


- Flow control - throttling senders
  - Execution time (per period) e = utilization / period
  - Budget in packets m = e / packets_per_second
- Congestion control - avoiding switch contention (adjust wait time between packets)
  - Percent budget %budget = (1 - %laxity) = e/(d-t)
  - Packet wait time $w = w_{min}$ / %budget
  - Size change $w\Delta = -|w_i - w_{min}|/2$
  - New wait time $w_{i+1} = \min(w_{max}, \max(w_{min}, w\Delta))$

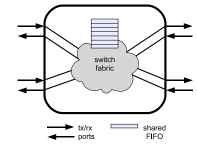


Queue modeling: single network stream

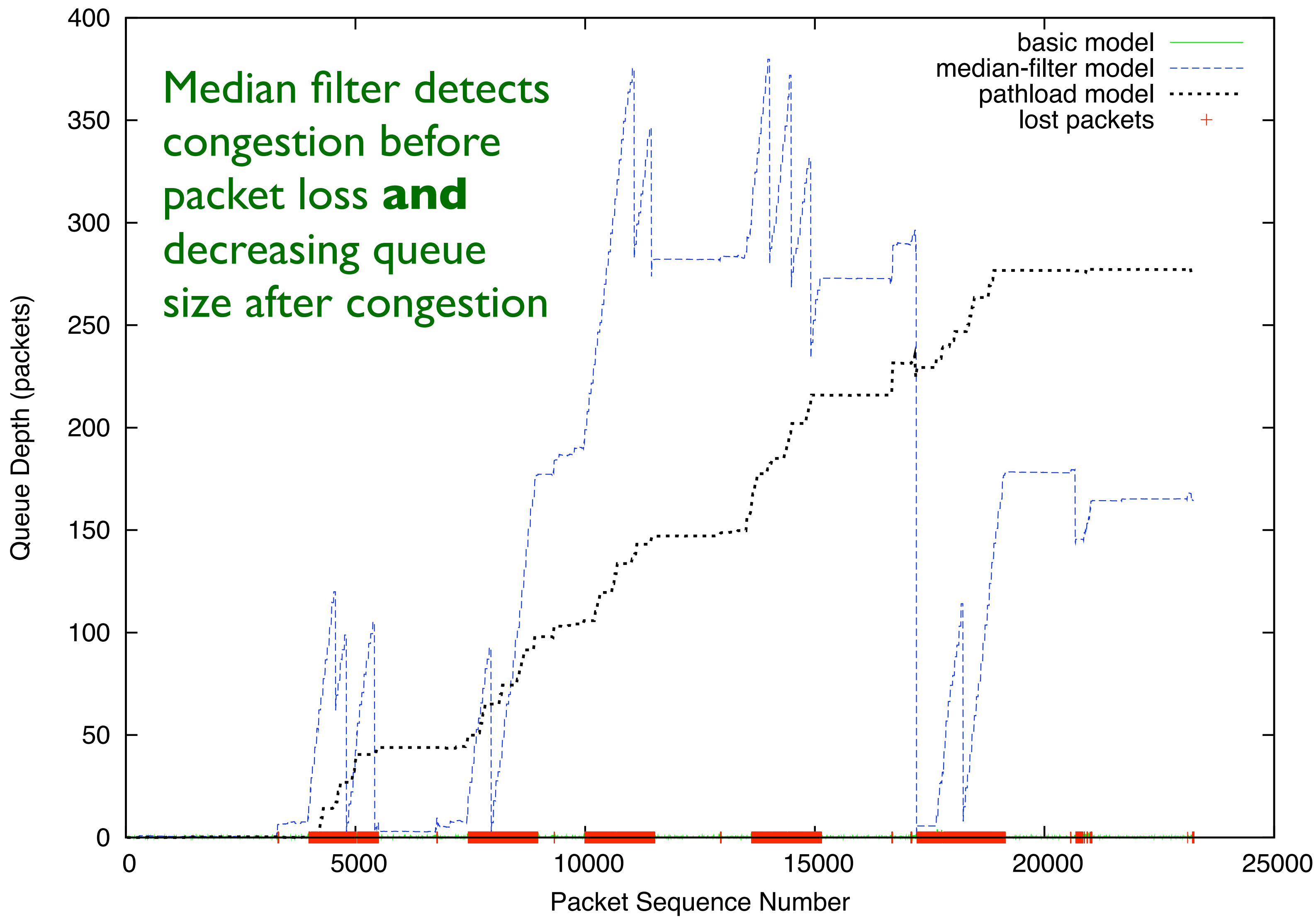
- No contention: 765 Mbps w/no lost packets



Queue modeling: punctuated stream

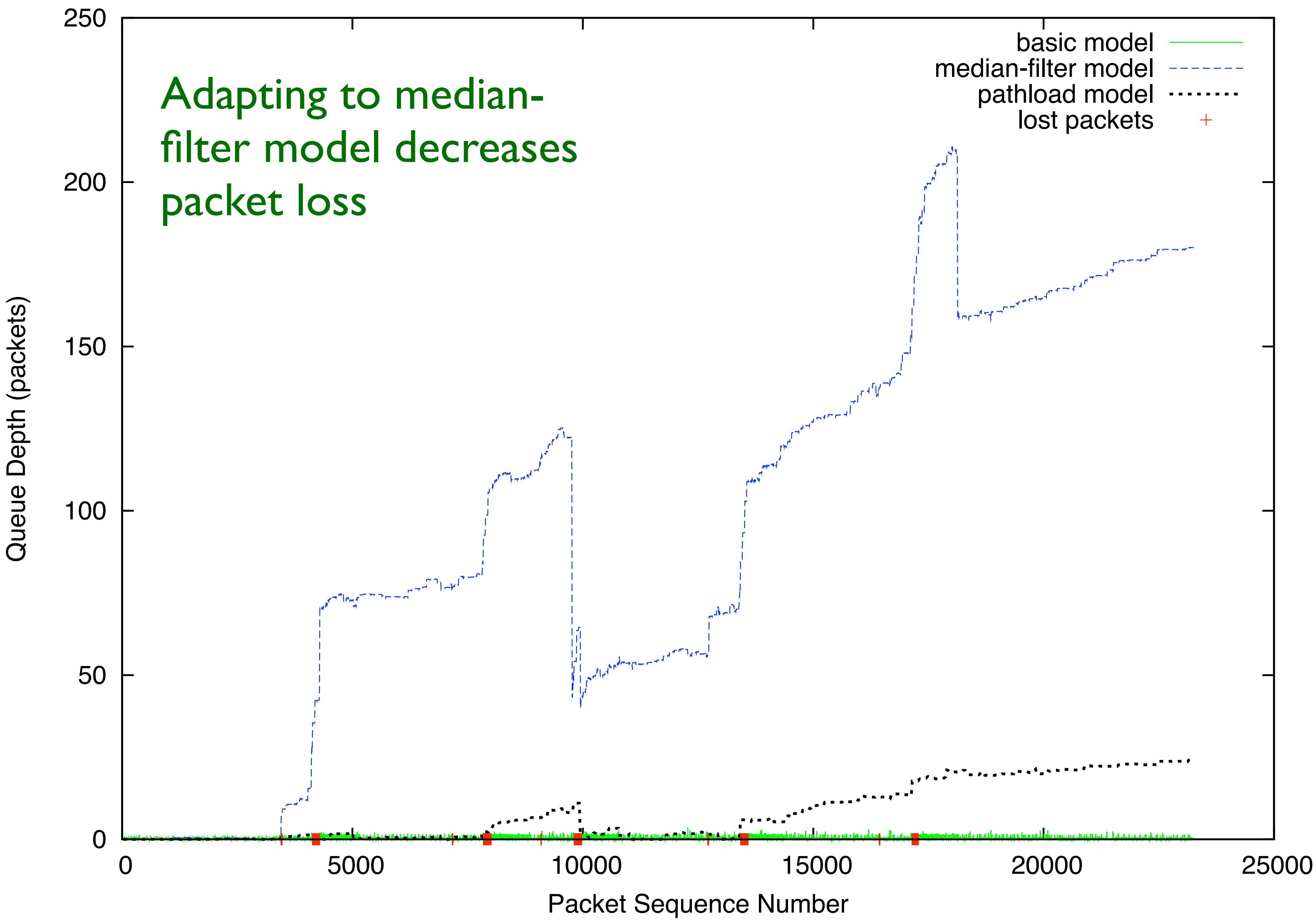


- Contention: 5 bursts of 250 Mbps

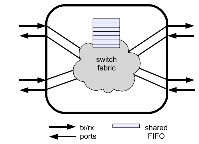


Queue modeling: punctuated adaptive stream

- Contention: 5 bursts of 250 Mbps

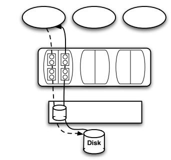


Userspace RADoN prototype



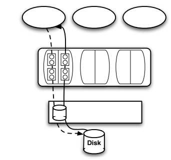
- Detects congestion using Relative Forward Delay
- Responds to congestion using RAD real-time theory
- Decreases packet loss significantly
- Improves goodput
- Requires **no** global knowledge or synchronization
- Ongoing: RADoN kernel implementation

Buffer management for I/O guarantees

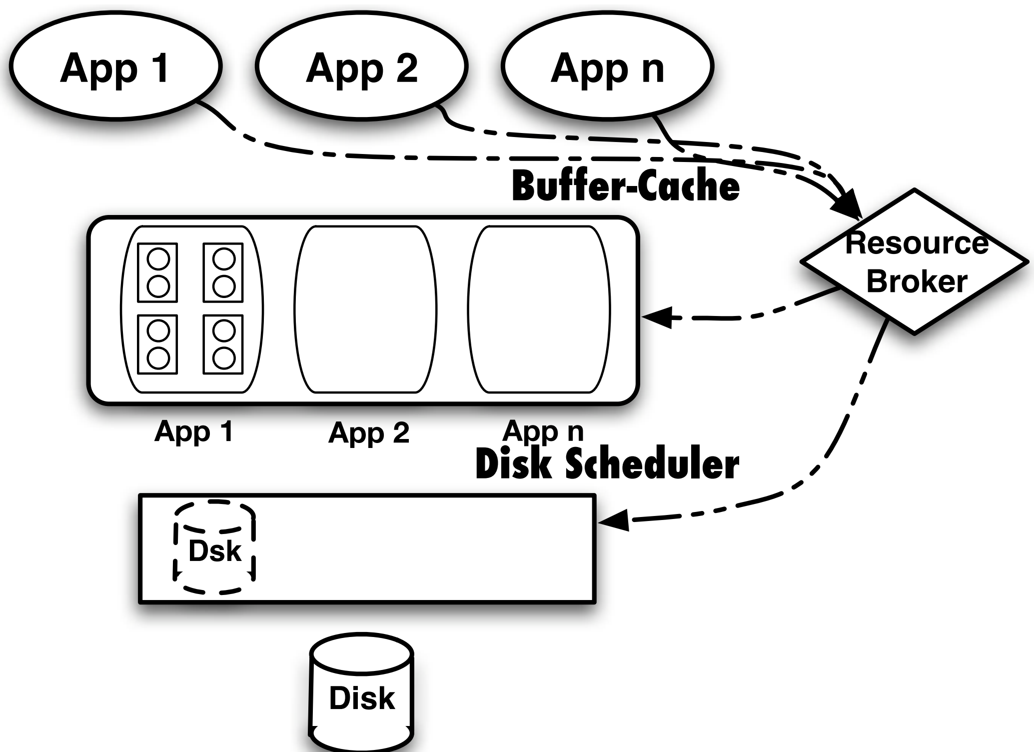


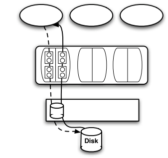
- Goals
  - Hard and soft performance guarantees
  - Isolation between I/O streams
  - Improved I/O performance
- Challenging because:
  - Buffer is space-shared rather than time-shared
    - Space limits time guarantees
  - Best- and worst-case are opposite of disk
  - Buffering affects performance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

Guarantees in the buffer cache



- Role
  - Improve performance
  - Preserve & enhance guarantees
- App-specific guarantees:
  - Hard at core
  - Soft when possible
- Predictable
  - Hard isolation
  - Device time utilization

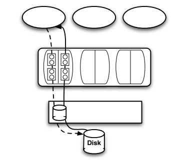




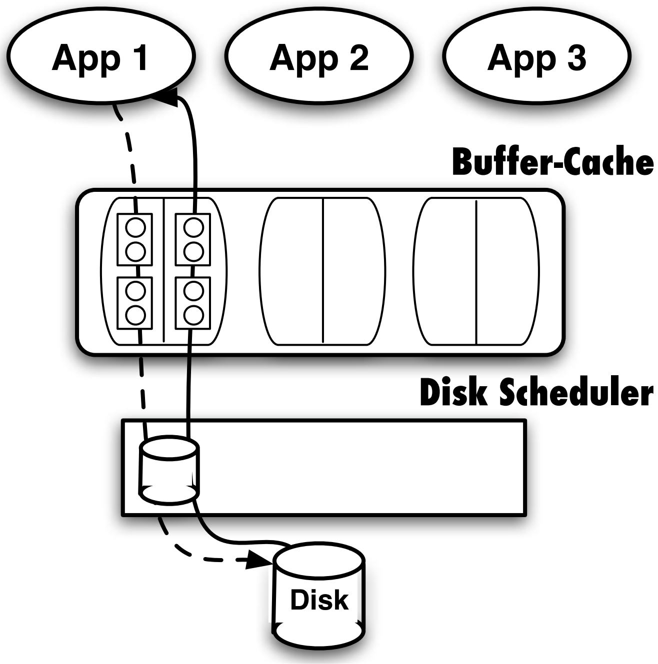
Buffering roles in storage servers

- Staging and de-staging data
  - Decouples sender and receiver
- Speed matching
  - Allows slower and faster devices to communicate
- Traffic shaping
  - Shapes traffic to optimize performance of interfacing devices
- Assumption: reuse primarily occurs at the client

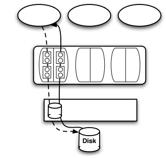
Radium



- I/O into and out of buffer have *rates* and *time granularities (periods)*
  - *Period transformation*: period into cache may be shorter than from cache to disk
  - *Rate transformation*: rate into cache may be higher than disk can support
- Partition cache based on I/O characteristics and performance requirements
- Cache policies enhance performance within constraints determined by I/O requirements
  - Use slack to prefetch reads and delay writes

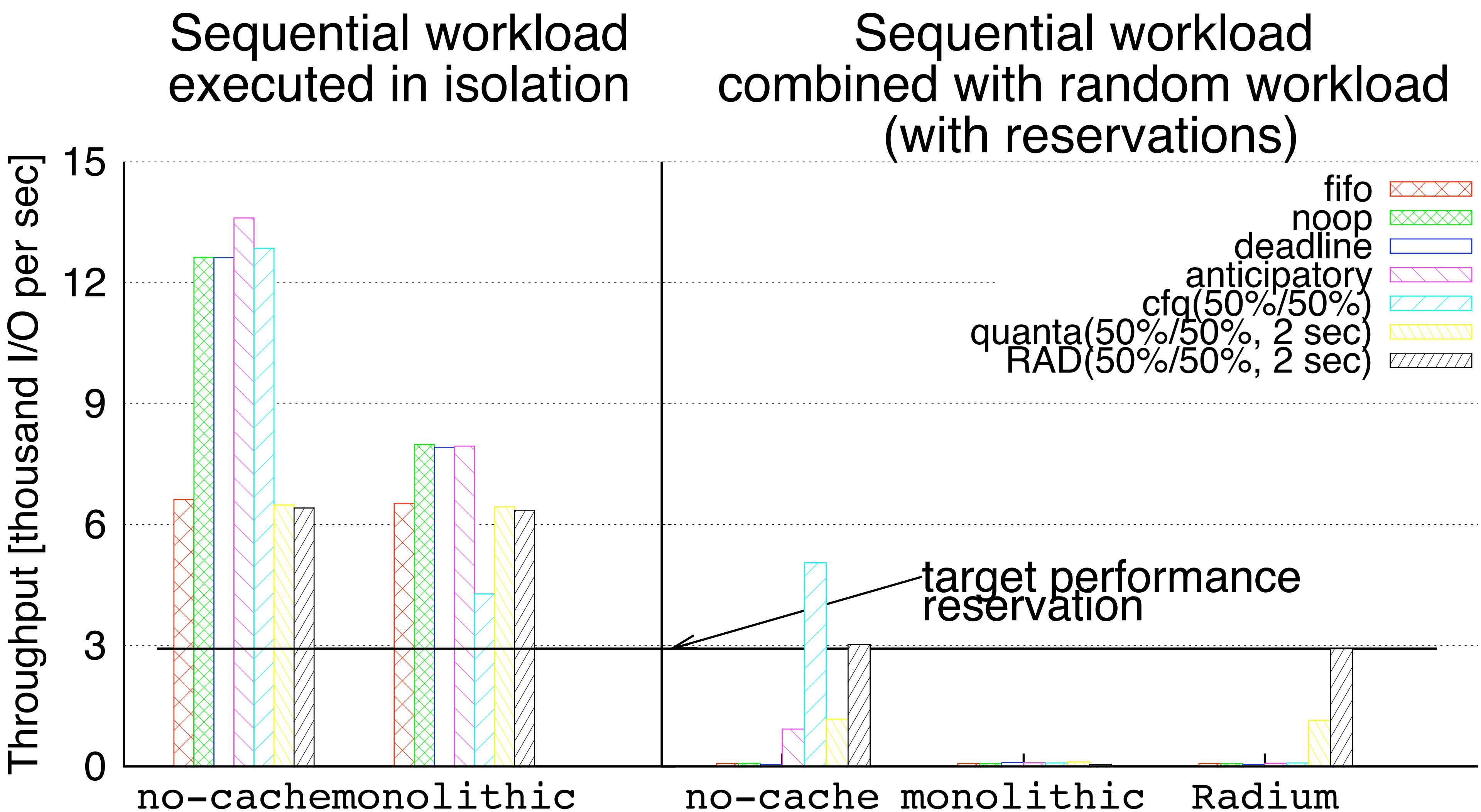


Enhancing guarantees in the buffer cache

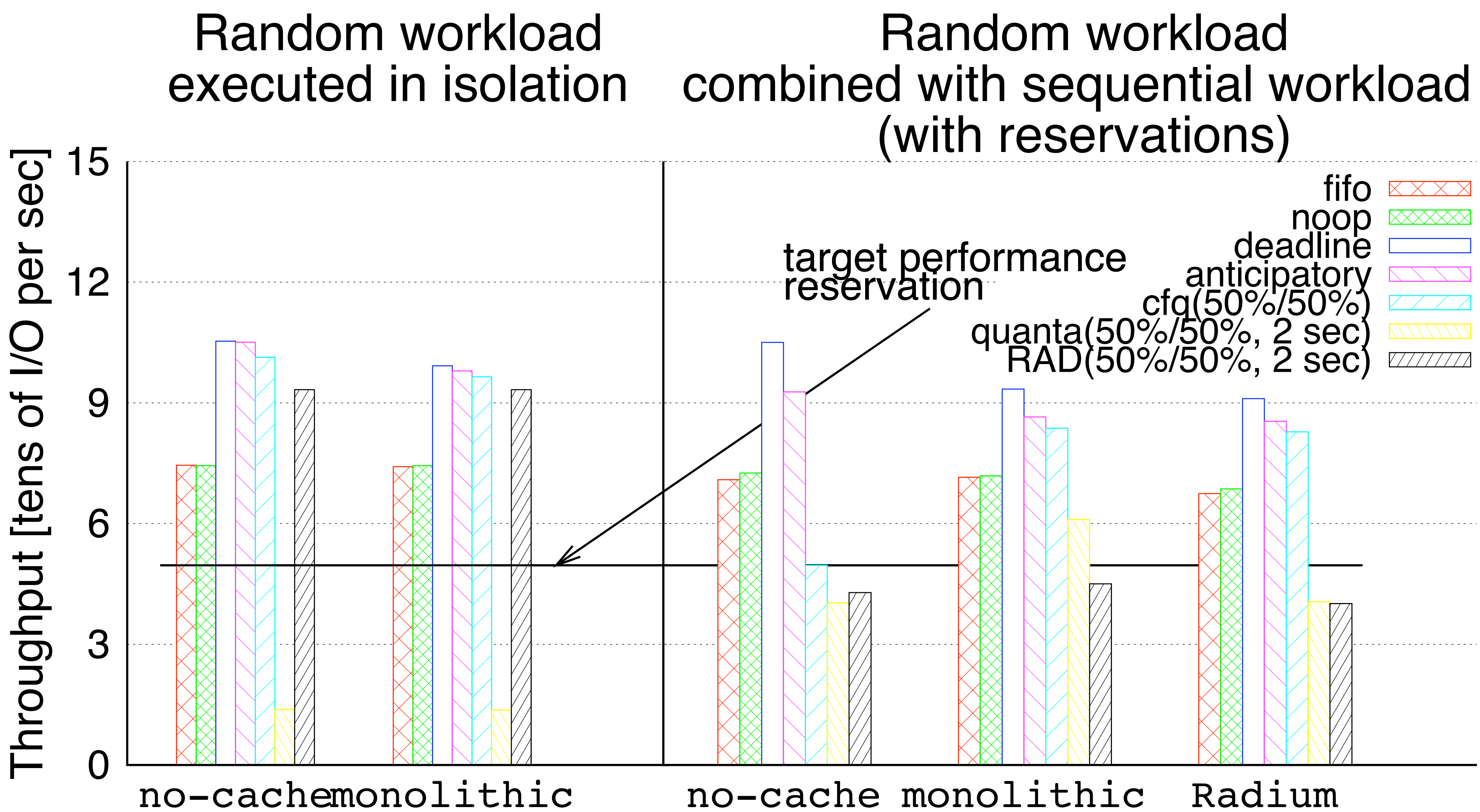


- *Reclaim unused resources* (e.g., unused overhead)
  - Use slack to prefetch reads and delay writes
  - Allow more unguaranteed services
- *Resource redistribution* (buffer swapping) accommodates burstiness
- *Period transformation*: period into cache may be shorter than from cache to disk
- *Rate transformation*: rate into cache may be higher than disk can support

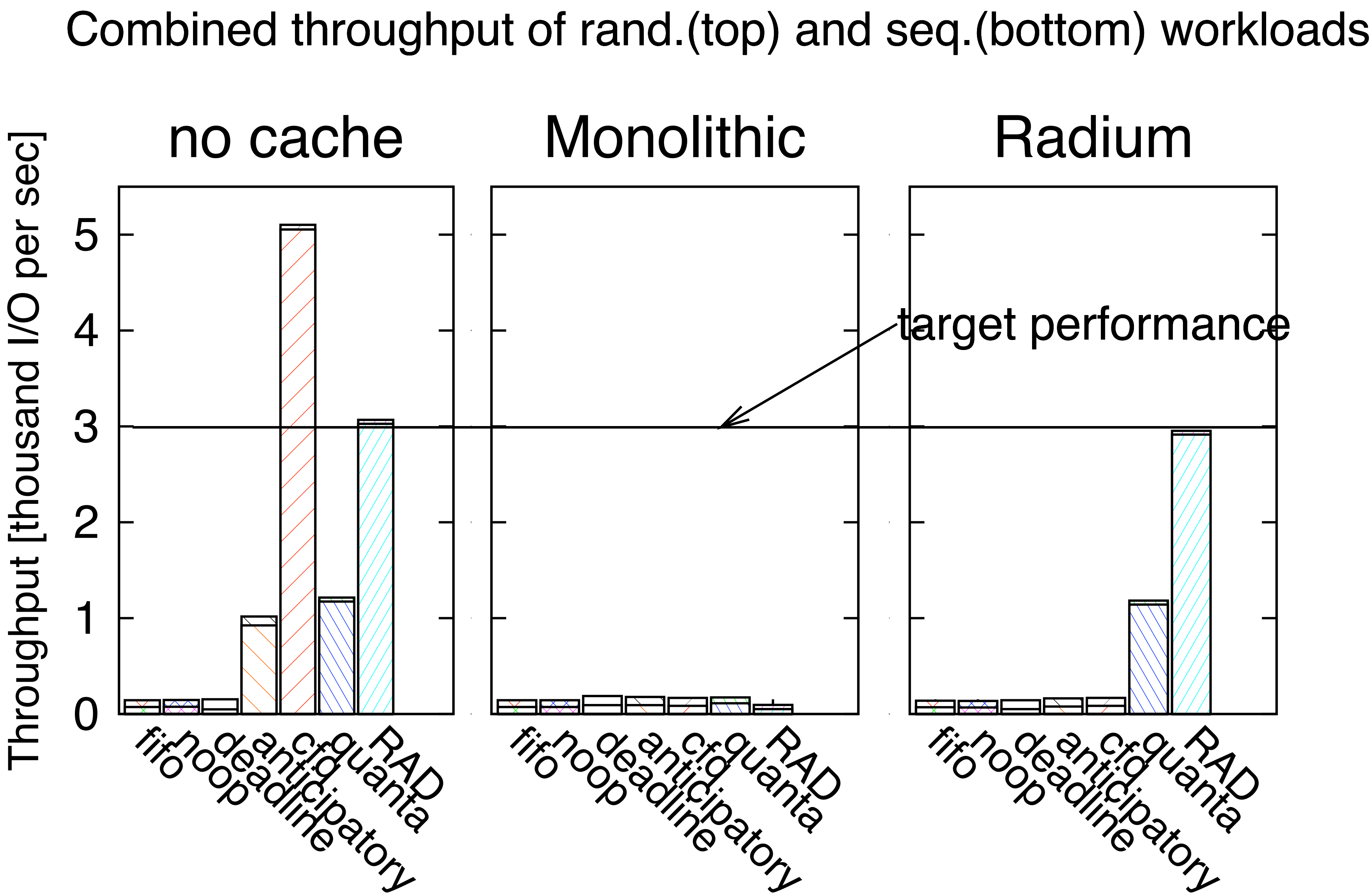
Managing a sequential workload



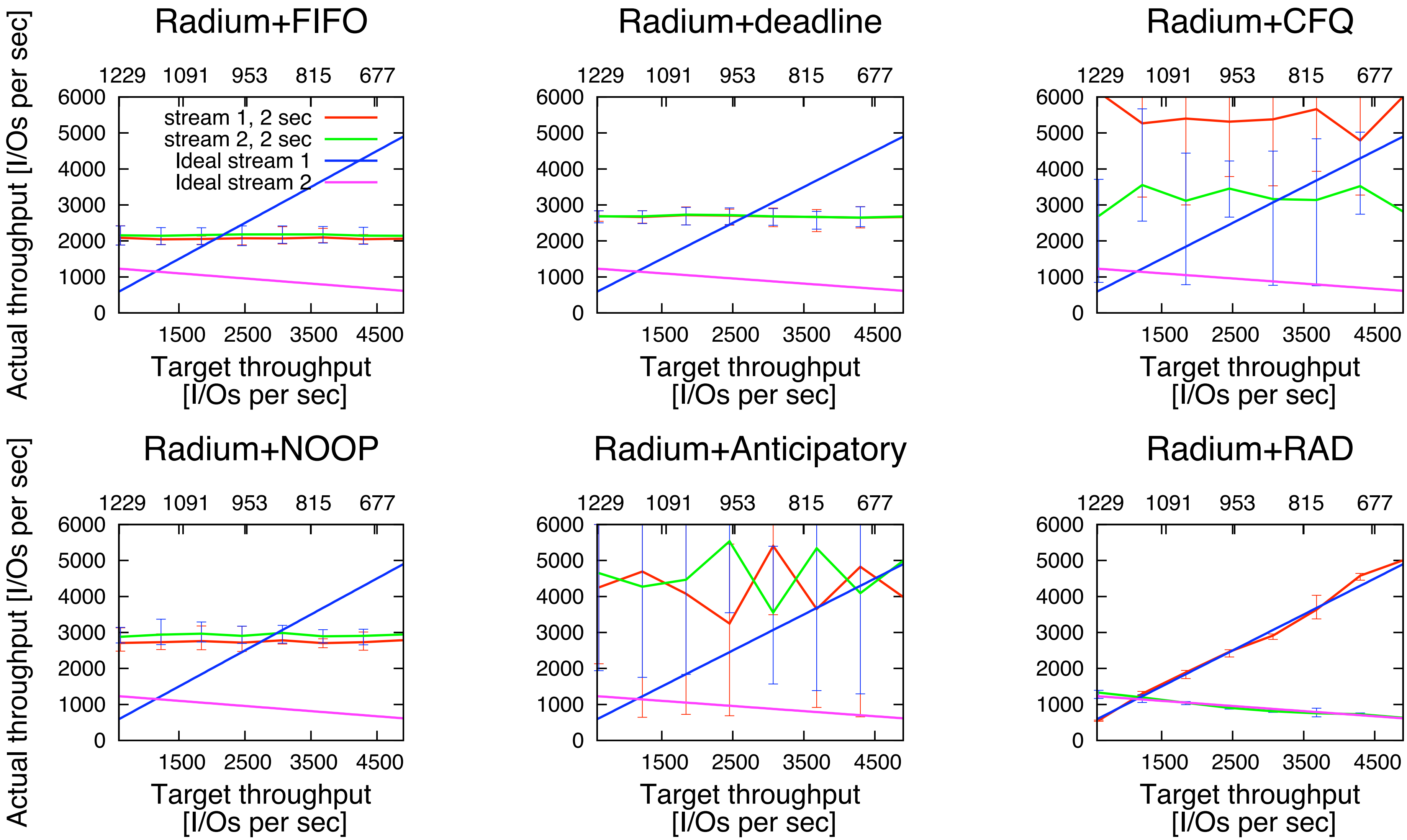
Managing a random workload



Managing combined workloads

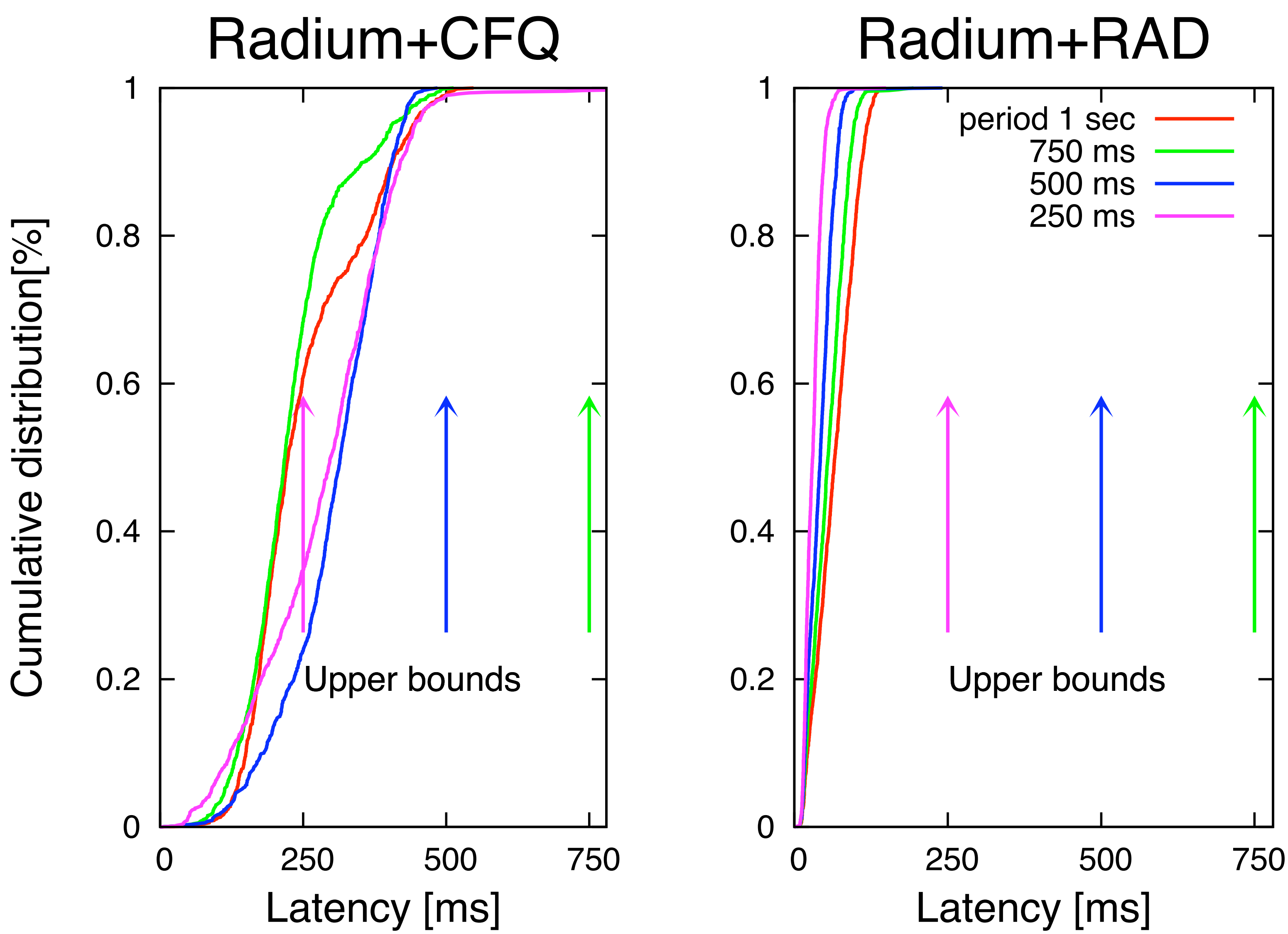


Controlling throughput w/mixed workloads



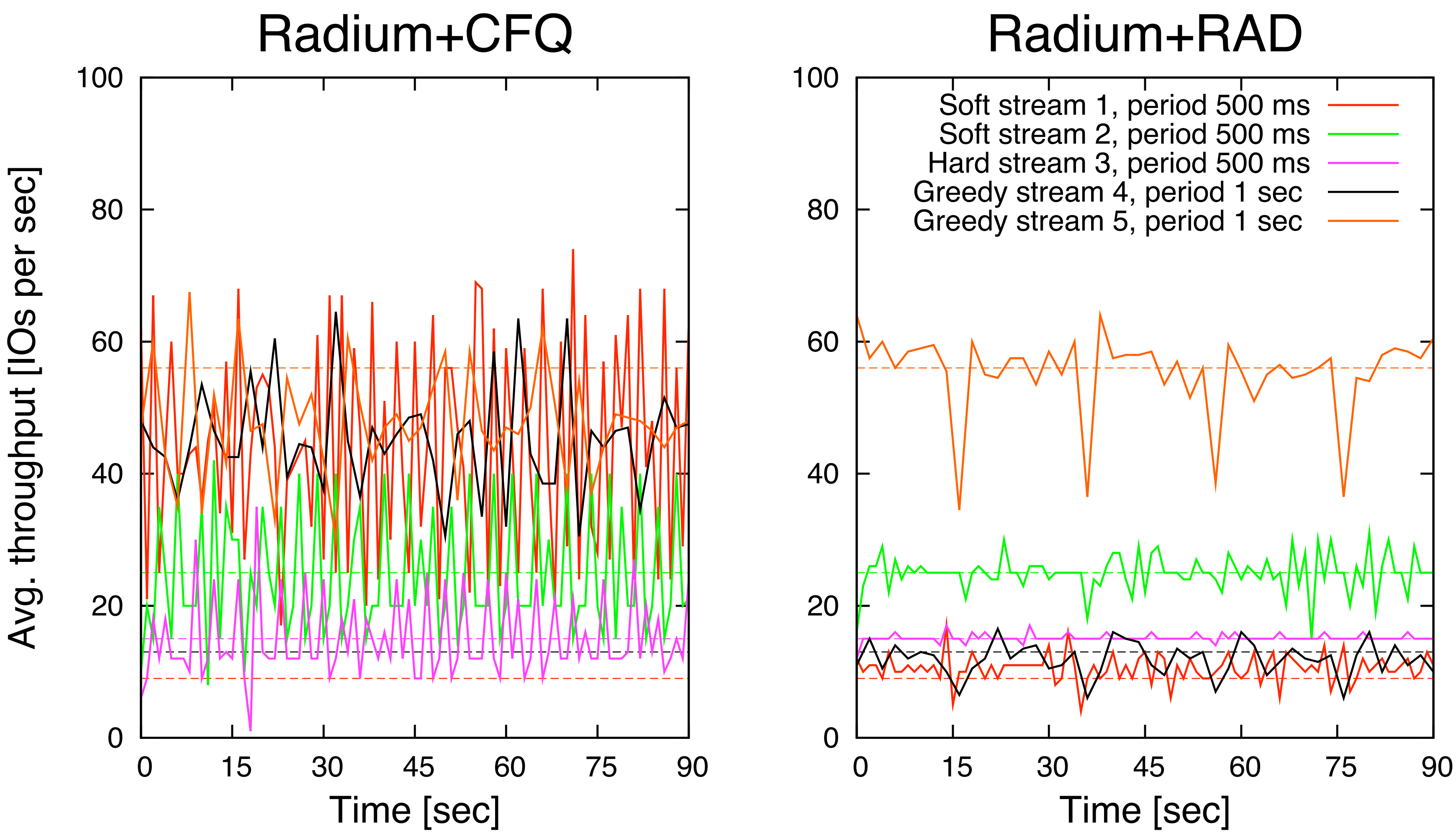
Consistent and predictable throughput for arbitrary reservations

Controlling latency w/mixed workloads



Precise control over the service times of each stream

Results w/complex workloads



Reasonable control with complex workloads

Data center performance management

- Big distributed systems
  - Serve many users/jobs
  - Process petabytes of data
- Data center design
  - Use rules of thumb
  - Over-provision
  - Isolate

- Ad hoc performance management creates marginal storage systems that cost more than necessary
- A better system would guarantee each user the performance they need from the CPUs, memory, disks, and network

Data center performance mgmt. goals

1. A first-principles model for data center perf. mgmt.
2. Full-system performance metrics for client processing nodes, buffer cache, network, server buffer cache, and disk
3. Performance visualization by application, client node, reservation, or device
4. Application workload profiling and modeling
5. Full system performance provisioning and management based on all of the above
6. Online machine-learning based performance monitoring for real-time diagnostics

RADIX

- $1 million from UC Lab Fee program
- Based on schedulers and workload-independent utilization metrics from our E2E QoS research
- Plan
    1. Performance model and metrics
    2. Tools for profiling, prediction, and planning
    3. Operating systems components
    4. Performance monitors and visualization tools
- Case study: LANL data centers

Conclusion

- Distributed I/O performance management requires management of many separate components
  - An integrated approach is needed
- RAD provides the basis for a solution
  - It has been successfully applied to several resources: CPU, disk, network, and buffer cache
- We are on our way to an integrated solution
- There are many useful applications: Data center performance management, full storage virtualization, ...