Background Use of Idle Resource Capacity

Lars Eggert

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781, USA

October 28, 2003

Overview

- **outline**
- introduction
- mechanism
- implementation
- evaluation
- related work
- conclusion

Problem Statement

- computer system = set of resources
- use idle resources “in background”
  - user benefit
  - increase efficiency + performance
- **primary goal: minimize FG delays**
  - or people won’t use it
  - secondary goal: BG throughput
- requires novel scheduling
  - priorities alone insufficient

Contributions

- generic resource processing **model**
- generic idletime **scheduler**
  - resource + workload independent
- FG impact **predictor**
  - error < 15%
- disk + network **implementation**
  - FG > 80%
  - BG < 90%

baseline throughput + latency

Challenges

- **resource characteristics**
  - different: storage, I/O
- arbitrary **workloads**
  - aperiodic, bursty, interactive
- existing **OS, apps, services**
  - minimal API changes
- **system limitations**
  - preemption: cost or absence

Overview

- outline
- **introduction**
- mechanism
- implementation
- evaluation
- related work
- conclusion

Unused Capacity

- many resources frequently unused
  - 50-70% **memory**
  - 70% **CPUs** in LAN
  - **disks + network** (anecdotal)
- future increase likely
  - hardware outgrowing workloads
- **unused capacity = lost opportunity**
- use in background

Idletime Service

- separate, low-priority service class
  - service class **default = FG**
  - inverse of traditional QoS
  - similar: POSIX *idprio* CPU
- **ideal:** zero FG impact
  - BG undetectable → hardware support
- **goal:** minimize FG impact
  - limit performance decrease
  - prevent incorrect processing

Applications + Benefits

- prefetching/caching = reduce access costs
  - prevent delays → **conservative limits**
- migration systems (process/data)
  - *B-Bandit, Condor*, *Sprite*, x*@home*, *Mether*
  - **coarse-grained, single-resource, remote benefit**
- distributed system scheduling
  - *GRID*
  - system-wide idletime
- system optimization + maintenance

Application: Optimizations

- **disk**
  - block replication → storage
  - arm movement → I/O
- **network**
  - IKE exchange, PMTU discovery, DNS lookup
  - warm caches (“prefetch means”)
- **CPU**
  - recompile
  - dynamic optimization
  - warm caches

Application: Maintenance

- idletime cron
  - run **at** time → run **by** time
  - try idletime, fall back to regular
- examples
  - fsck
  - defragment + consolidate
  - virus check
  - updates

Overview

- outline
- introduction
- **mechanism**
- implementation
- evaluation
- related work
- conclusion

Resources

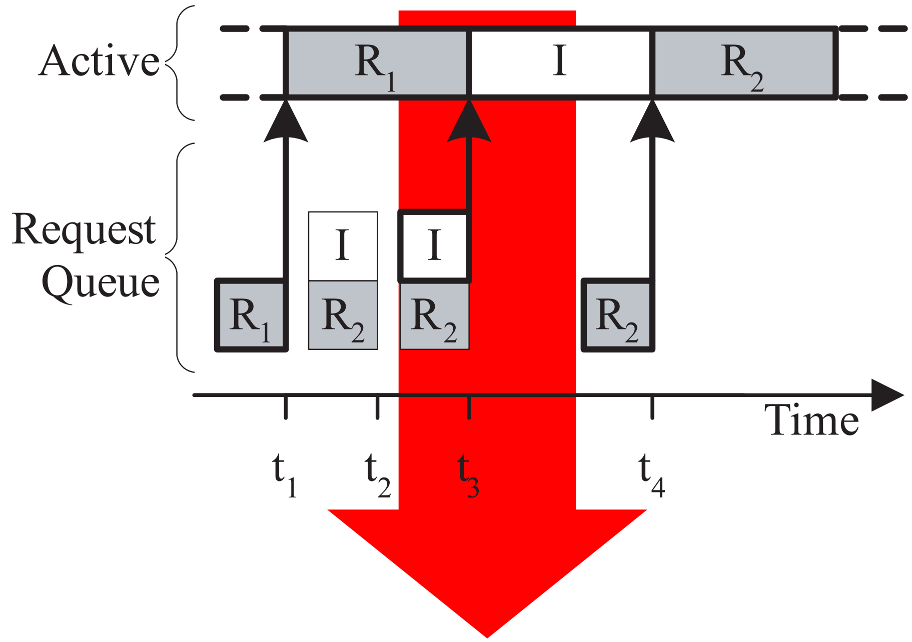
- 2 resource types
  - **spatially** shared: memory, disk
  - **temporally** shared: CPU, network
- support BG use for both
  - talk: focus on temporal
  - dissertation: discusses both
- basic idletime principles
  - isolation, prioritization, preemptability

Isolation

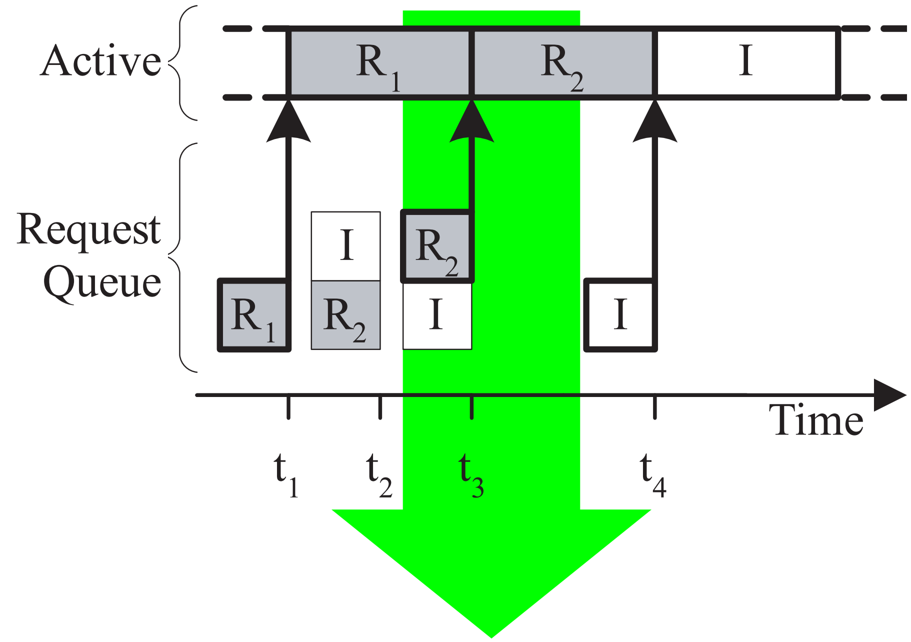
- BG **side effects** interfere with FG
- correctness
  - BG state affects/prevents FG
  - challenge for spatially shared resource
- performance
  - scheduling → idletime scheduler
  - runtime optimizations → disable
- **must disable runtime BG optimizations**
  - OS + hardware: read-ahead, caching, etc.

Prioritization

- never serve BG while FG in queue



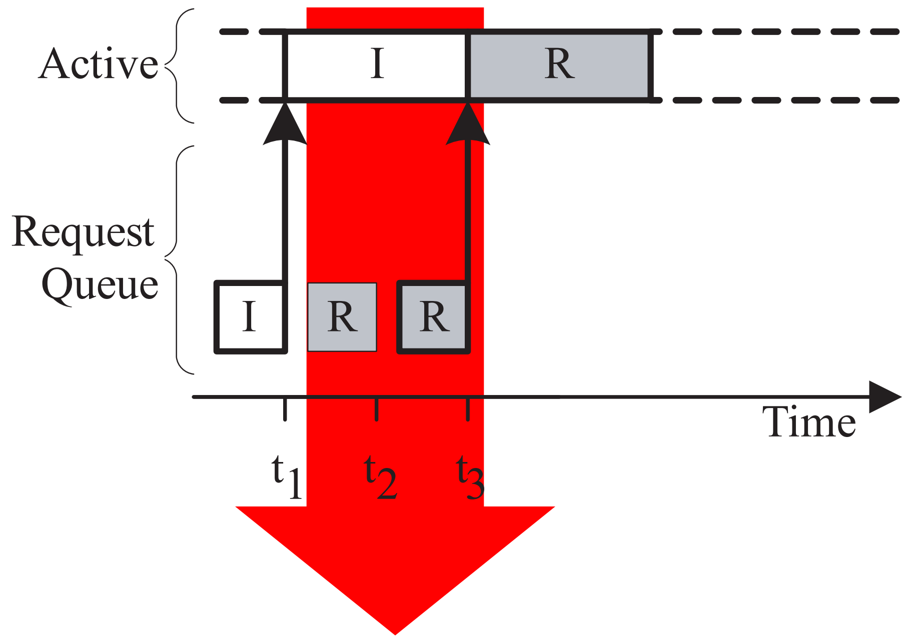
not prioritized



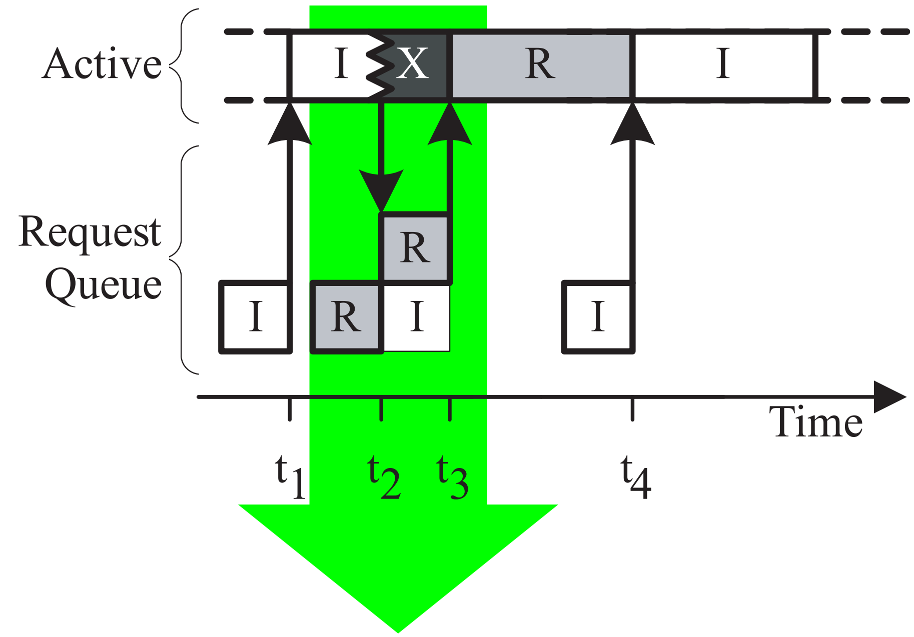
prioritized

Preemptability

- new FG → stop active BG + start FG



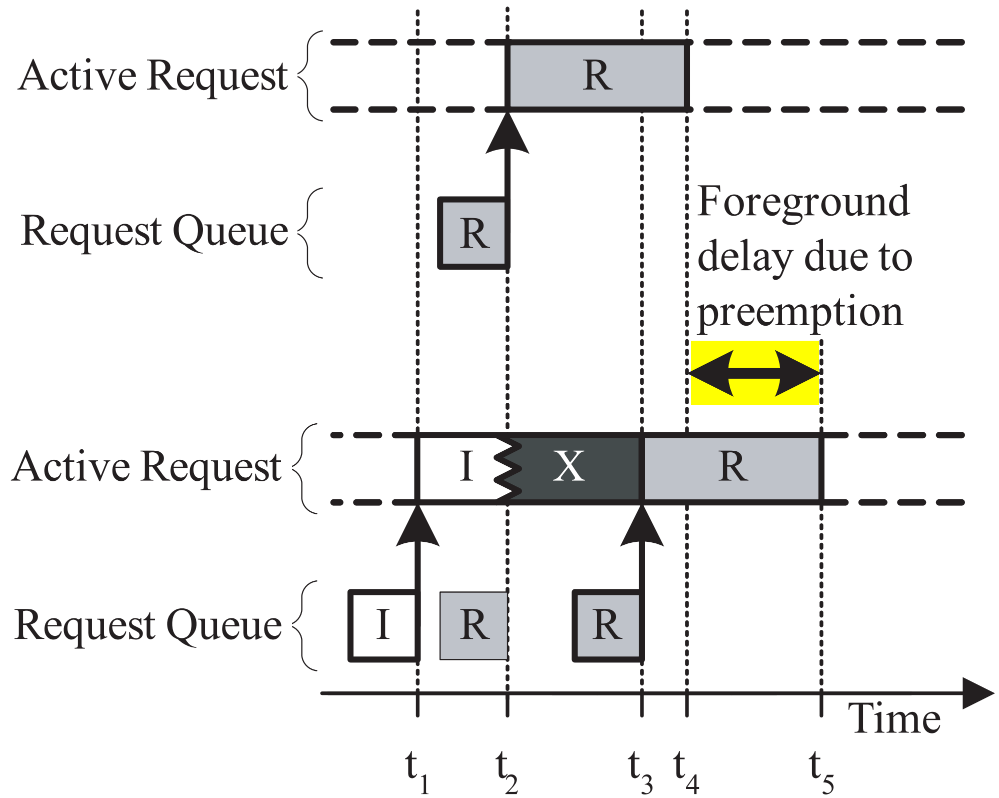
not preempting



preempting

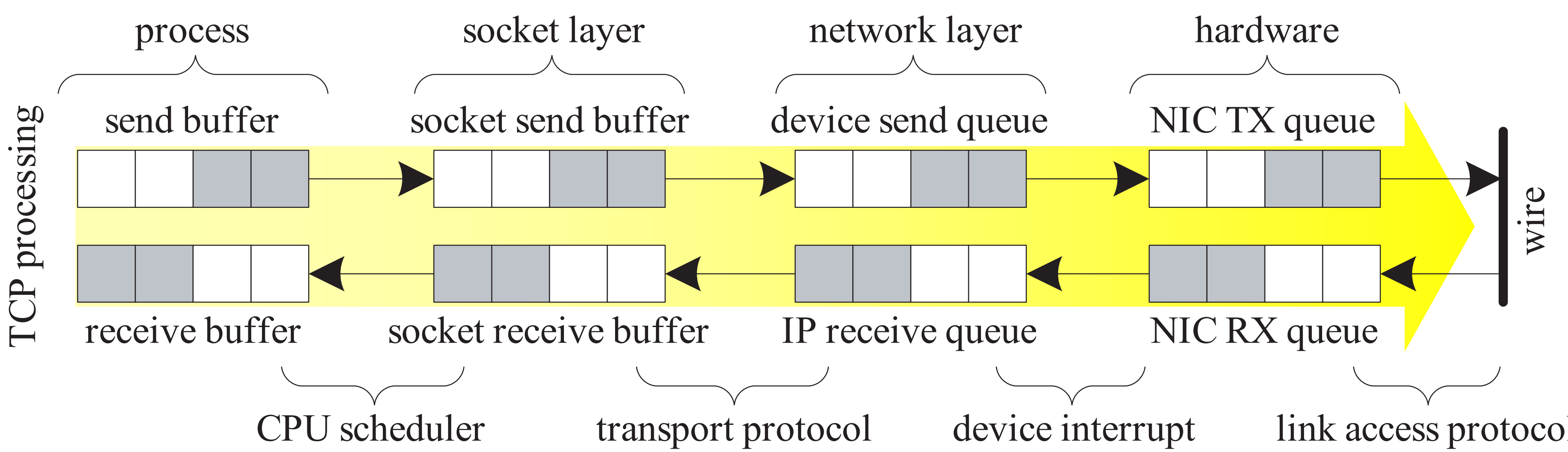
Preemption Cost

- BG $\rightarrow$ FG switch: delay
- main cause of FG performance reduction
- no hardware support: **up to 50% impact**
- active BG must run to completion
- **limit cost = limit FG impact**



Work Conservation

- never remain idle with work queued
  - (never destroy completed work)
- **challenge:** OS = queue hierarchy
  - hierarchy level → priority
  - lower-layer BG → delay higher-layer FG

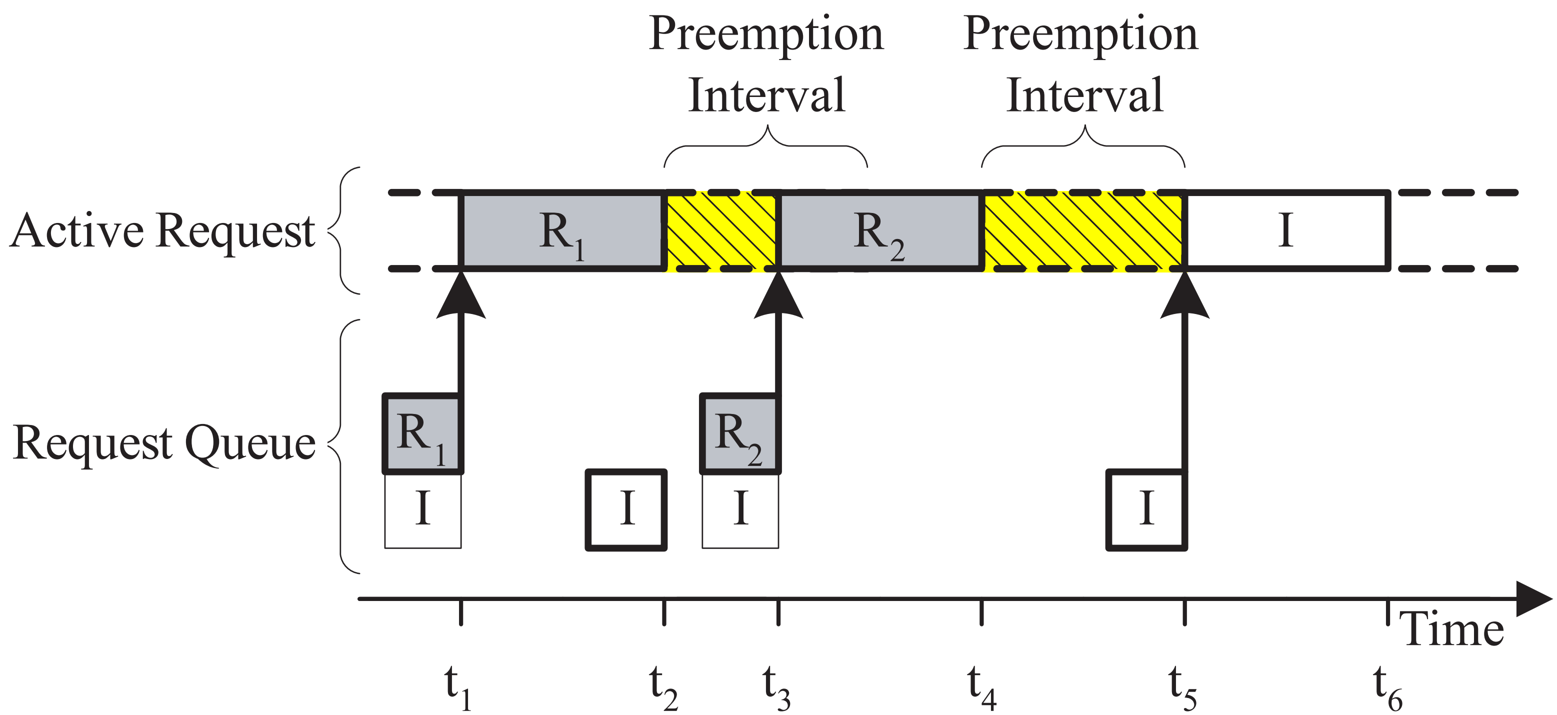


Work Conservation

- **creates** idletime worst-case
  - preemption before each FG request
- **idea:** relax work conservation **for BG only**
  - limit preemptions = limit FG impact
  - extend traditional OS, **localized** modification
- other approach: new OS from scratch
  - *Scout*, etc.
  - existing apps + workloads?

Preemption Interval

- period of relaxed BG work conservation
  - new FG → **start** immediately
  - BG → **halt** until PI ends
- enter PI before FG → BG switch

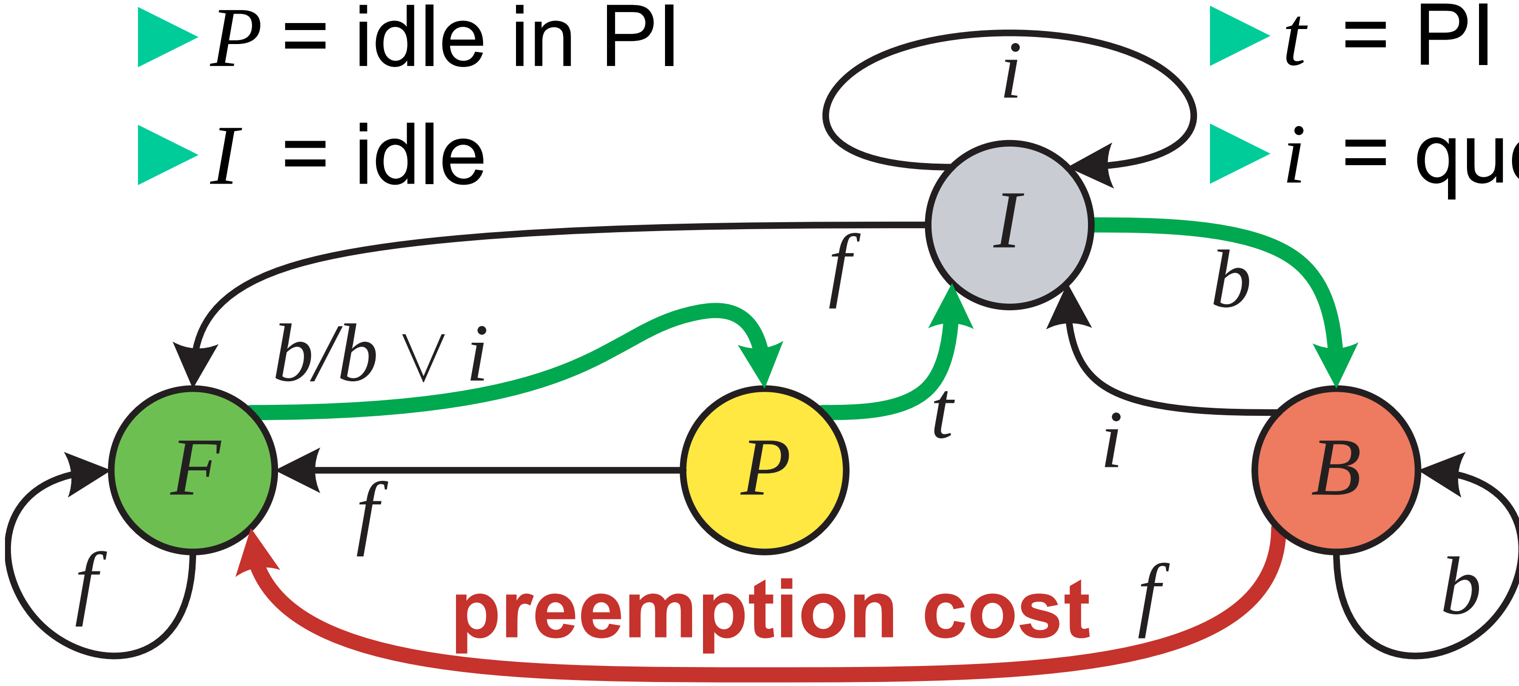


Idletime Scheduler

- priority queue + PI scheduling

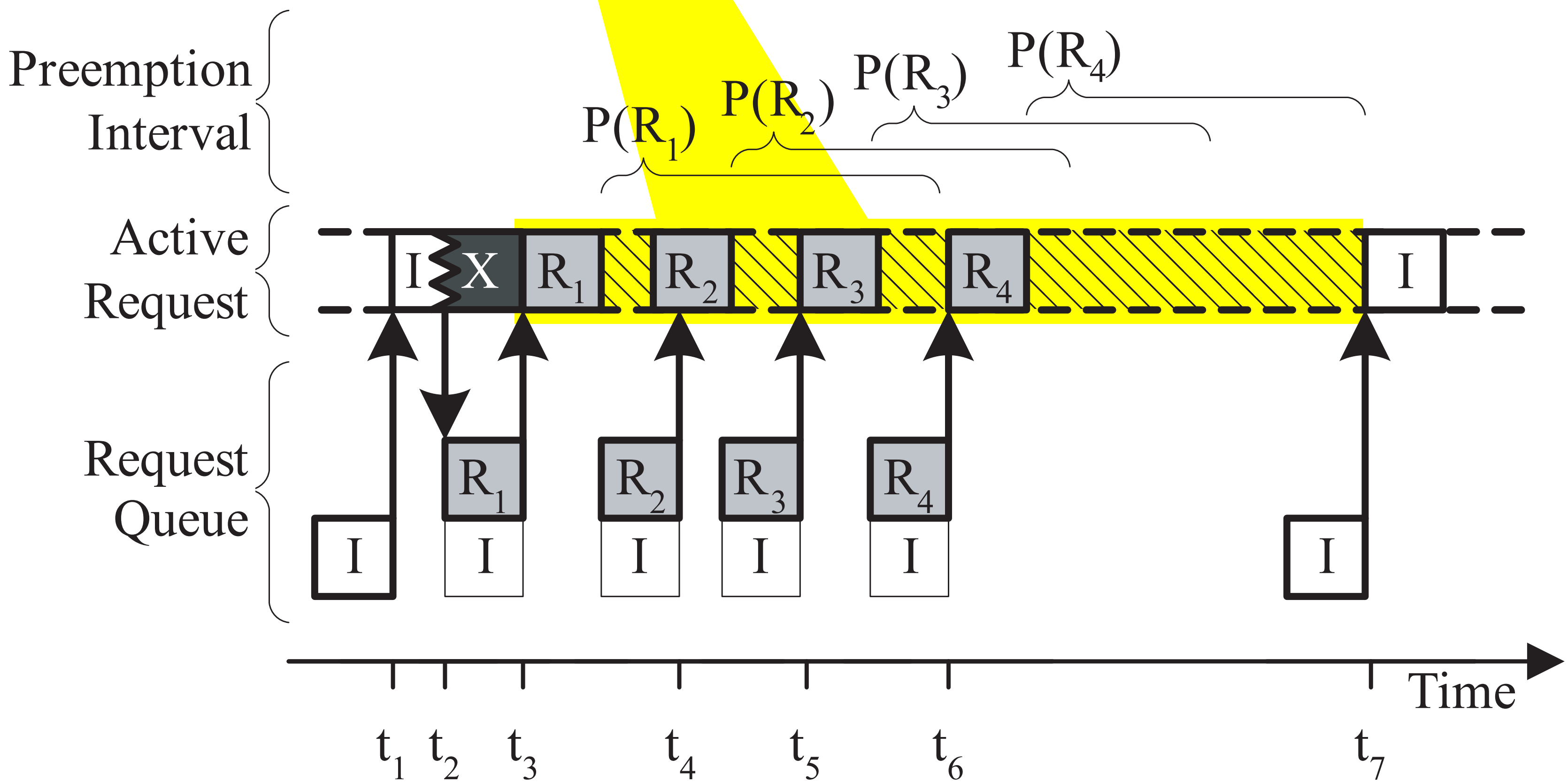
- **states**
  - F = FG active
  - B = BG active
  - P = idle in PI
  - I = idle

- **events**
  - f = FG in queue
  - b = BG in queue
  - t = PI expires
  - i = queue empty



FG Burst Formation

- PI **creates** bursts of FG requests
- max. 1 preemption/burst
- **limits** FG impact

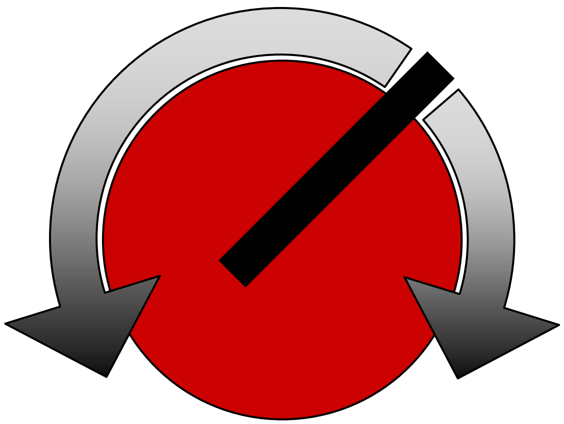


Preemption Interval Length

- **parameter:** controls scheduler
  - FG impact ~ BG performance
  - effective PI length?

shorter PI

- utilize more idle capacity
- higher FG impact
- increase BG performance

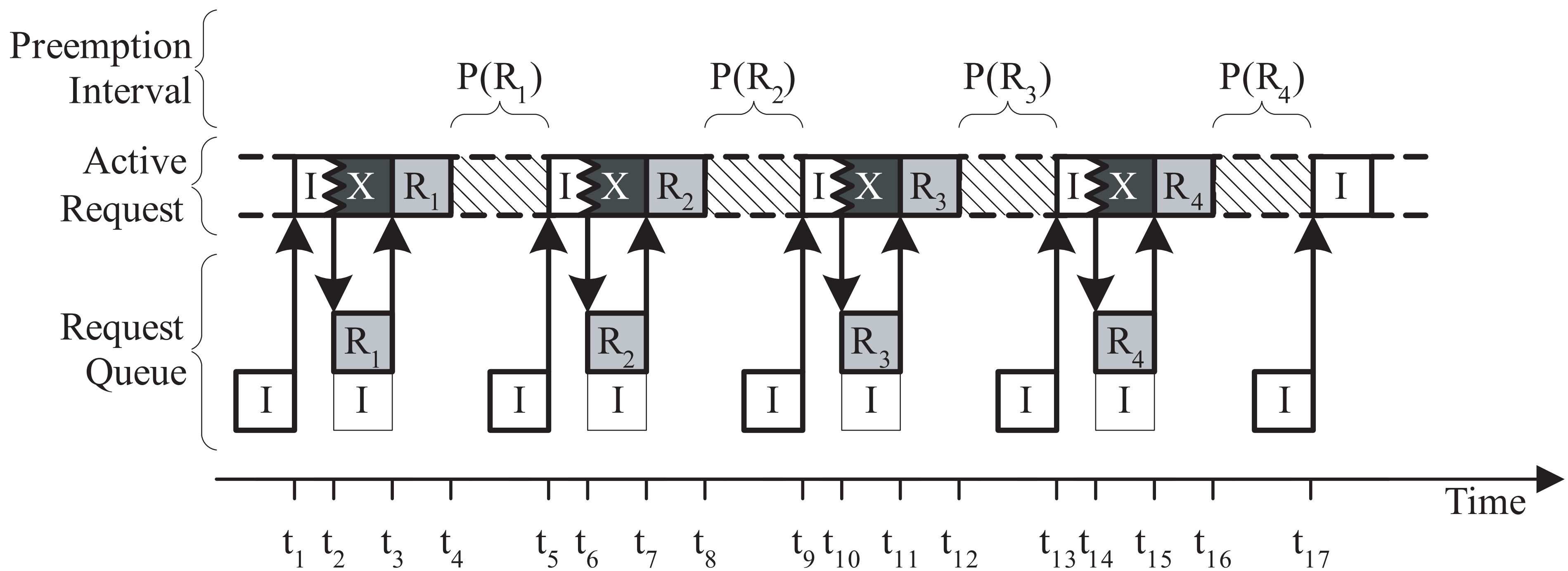


longer PI

- utilize less idle capacity
- reduce FG impact
- decrease BG performance

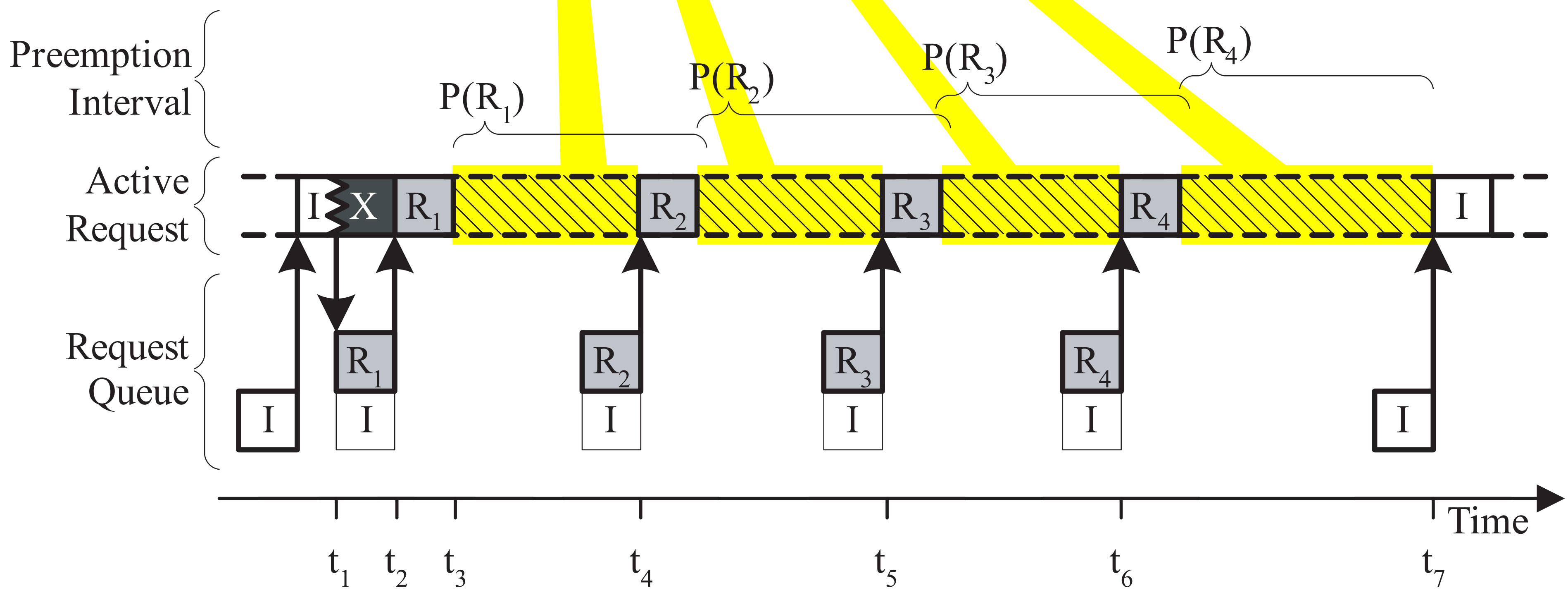
Short Preemption Intervals

- too short = **ineffective**
- mechanism degenerates → priority queue
- no cost limit = no FG impact reduction



Long Preemption Intervals

- too long = waste idle capacity
- poor BG throughput
- limited usefulness

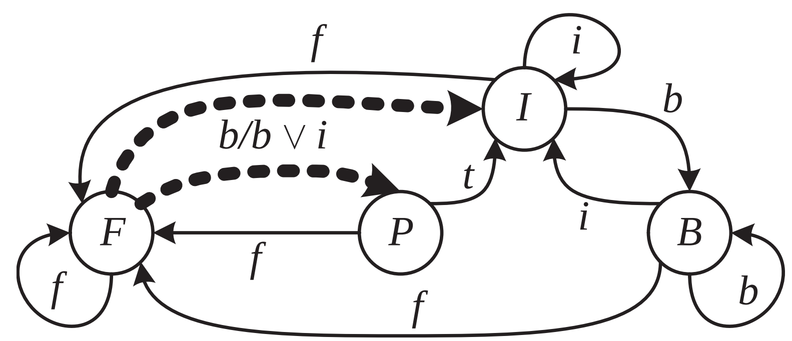


Effective Interval Lengths

- factors
  - resource
  - workload
  - user policy
- **lower bound:** create FG burst length > 1
  - otherwise: no cost amortization
- **upper bound:** FG inter-arrival gap
  - otherwise: BG halt

Future Extensions

- automatic PI length adaptation
  - preemption before FG → **lengthen PI**
  - FG without preemption → **shorten PI**
- **TCP-like:** preemption ~ loss
- allow **fixed** FG impact
  - **skip PI** after FG series
  - increase BG throughput



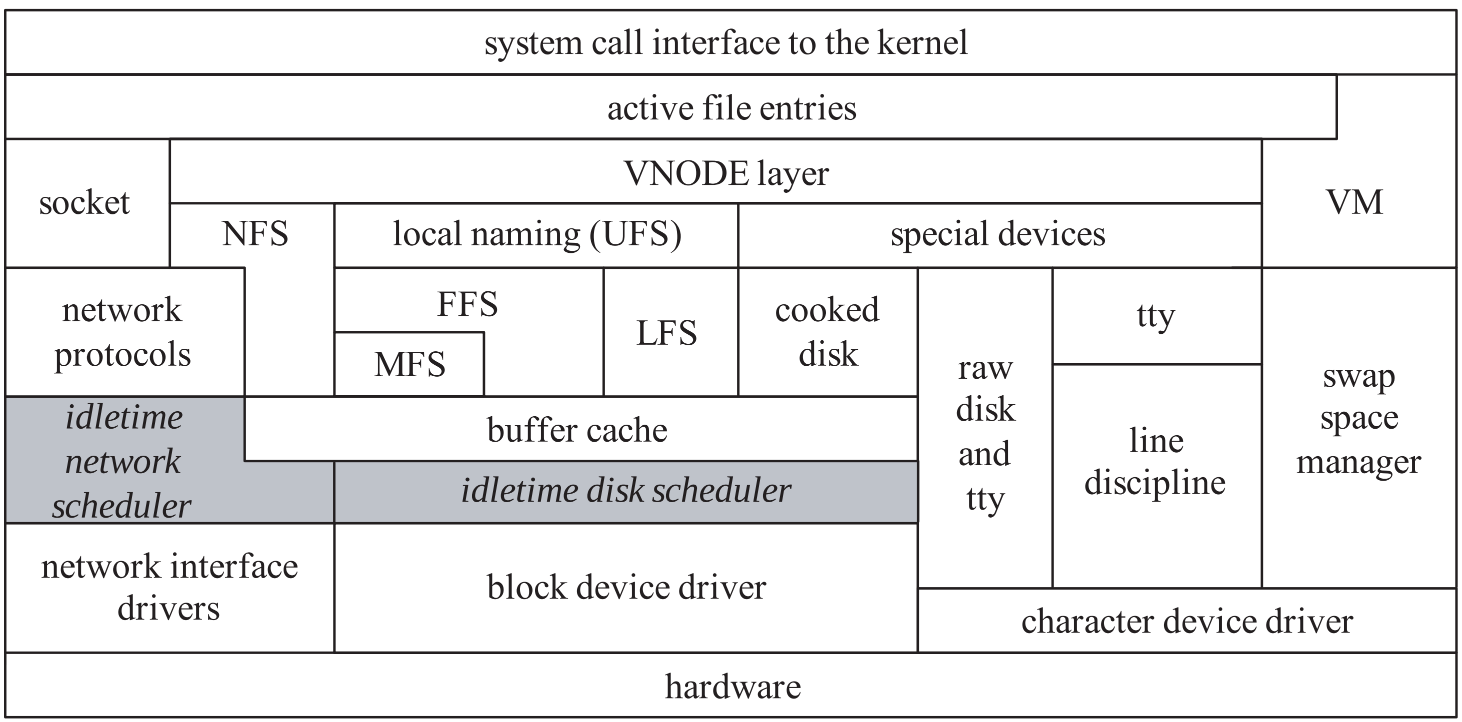
- idletime-aware **intra-class scheduling**
  - idletime = inter-class

Overview

- outline
- introduction
- mechanism
- **implementation**
- evaluation
- related work
- conclusion

FreeBSD 4.7 Implementation

- **localized** modification
  - disk: replace *disksort()*
  - network: new ALTQ discipline
  - (CPU: POSIX *idprio*)



Adapted from [McKusick1996]

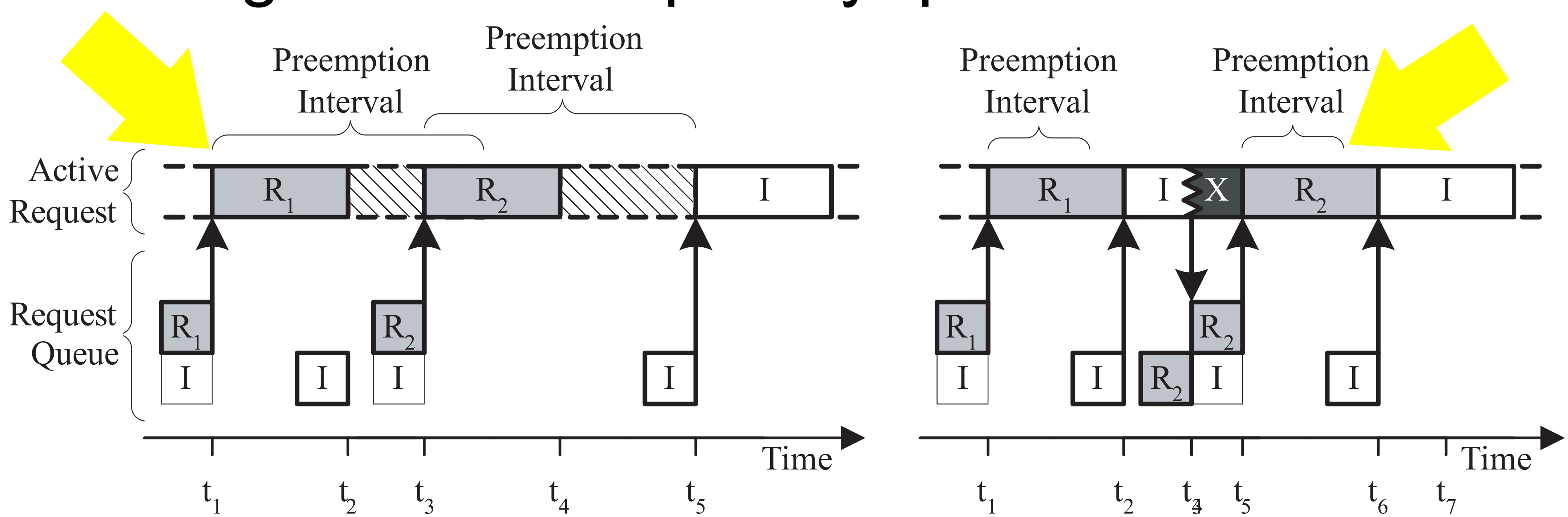
Implementation Details

- **API:** enable BG with descriptor flag
- **disk:** isolation → no runtime optimizations
  - no read-ahead
  - no buffer cache

  for BG only
  - (on-disk optimizations still active)
- **network:** inbound = default FIFO
  - packet marker: reuse diffserv
  - Internet2 QBSS

Implementation Considerations

- **begin** PI with FG request
  - not at end → simplify code
- **expectation:** PI < service time ineffective
  - PI expires while FG active
  - degenerates → priority queue

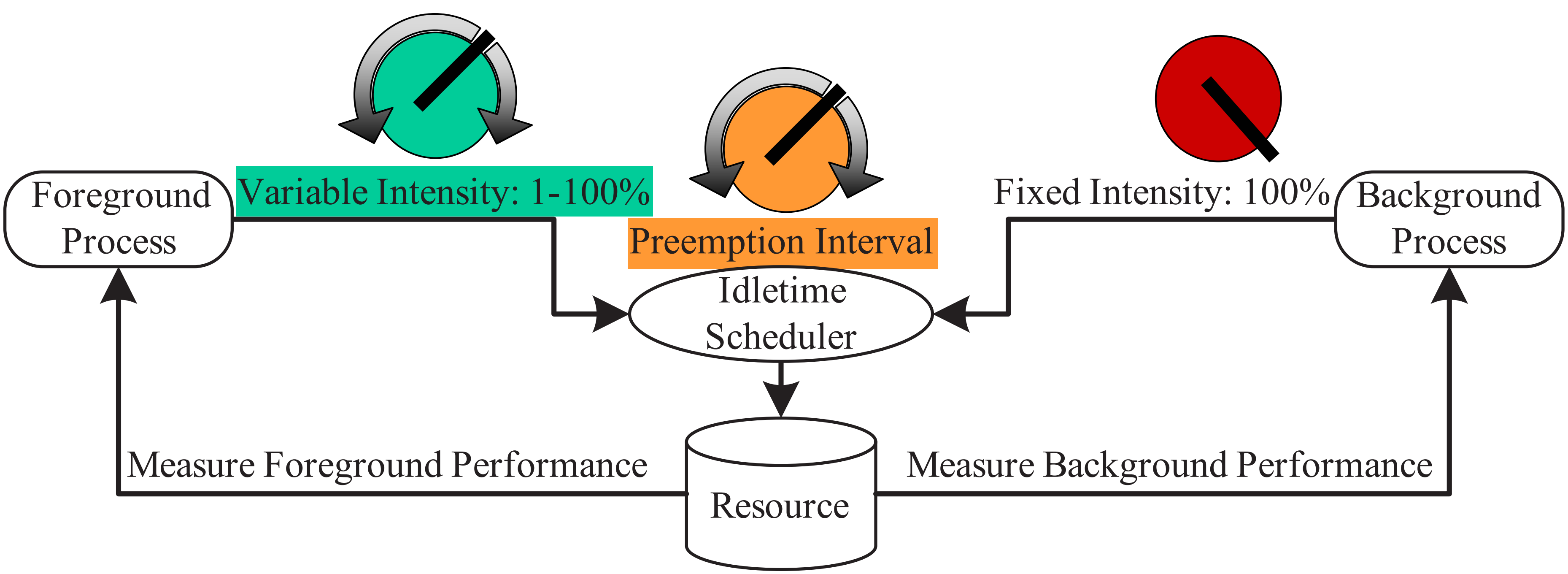


Overview

- outline
- introduction
- mechanism
- implementation
- **evaluation**
- related work
- conclusion

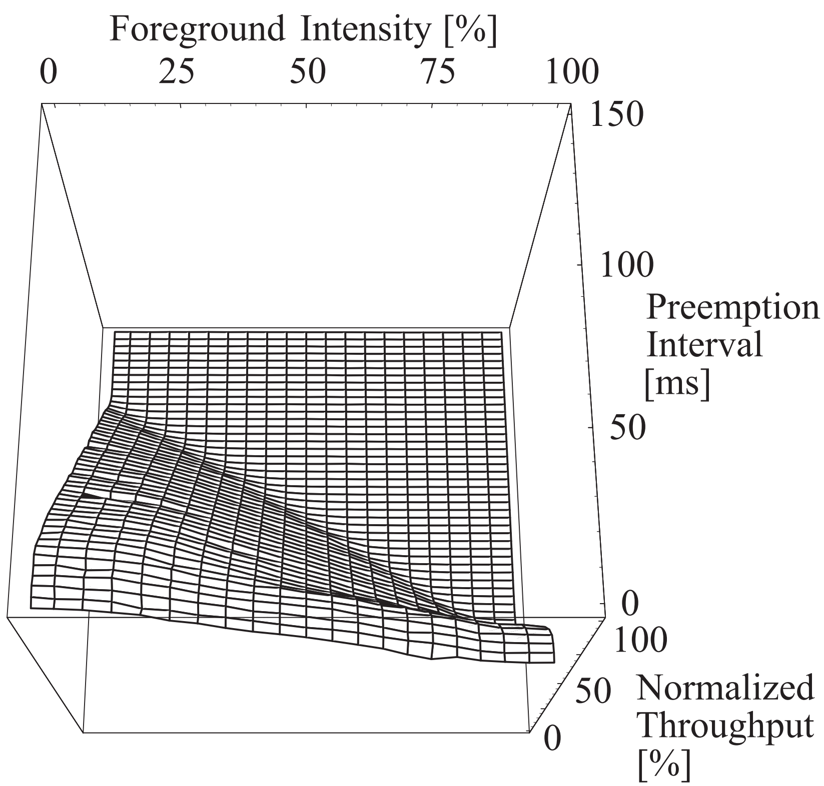
Experimental Setup

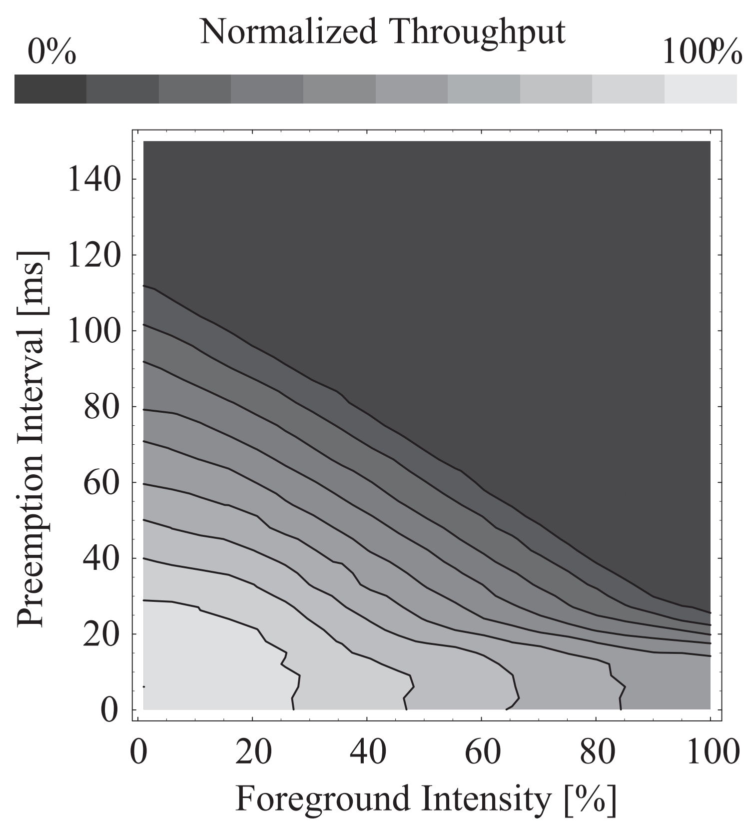
- PI length: variable
- FG intensity: variable
- BG intensity: fixed unlimited **= worst case**



Measurements

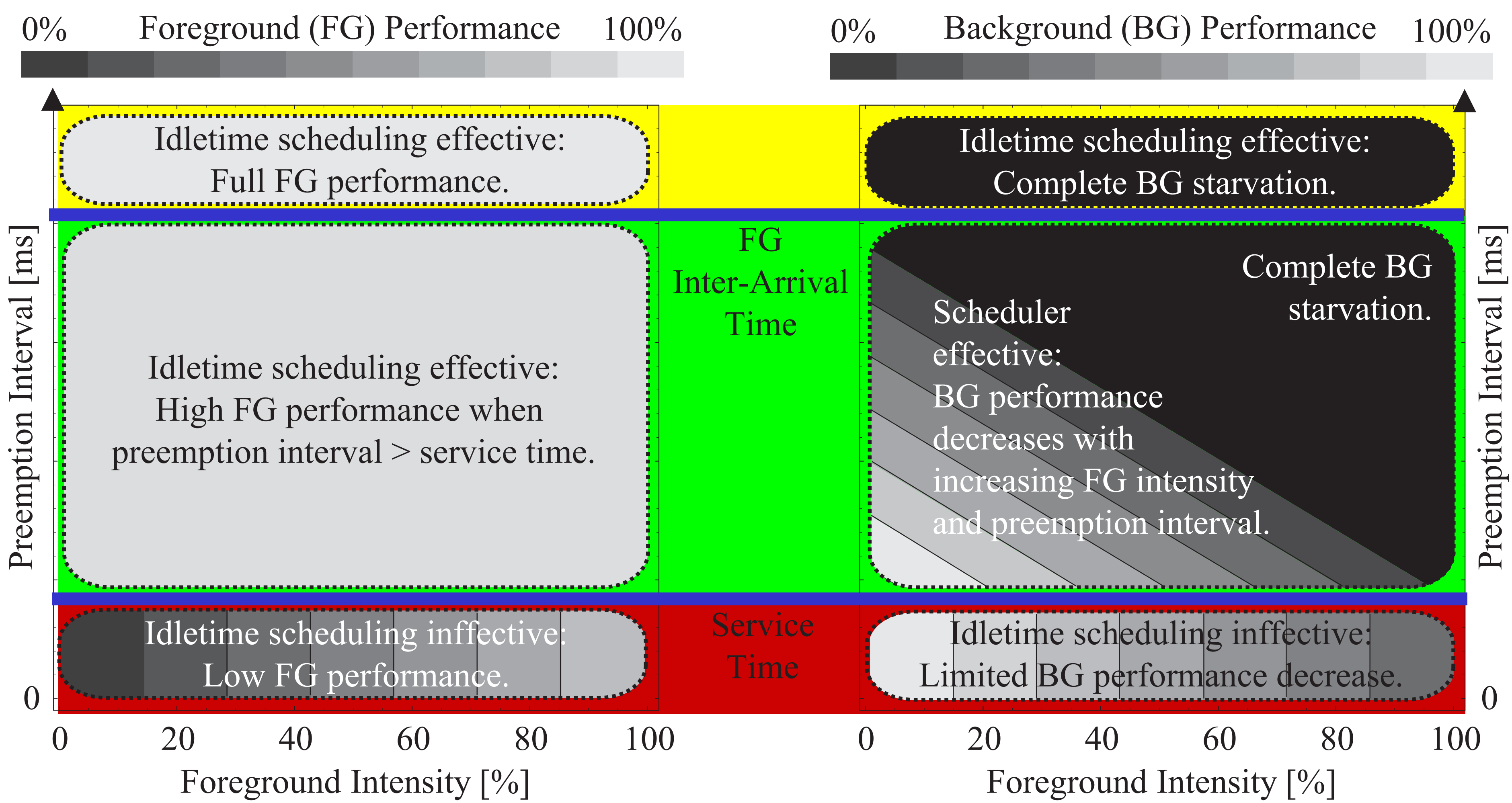
- FG/BG throughput (+ latency)
- normalize against baseline (= no BG)
- contour plot: **lighter = better**





Performance Expectation

- **3 regions** based on PI length
  - (illustration, not measurement)



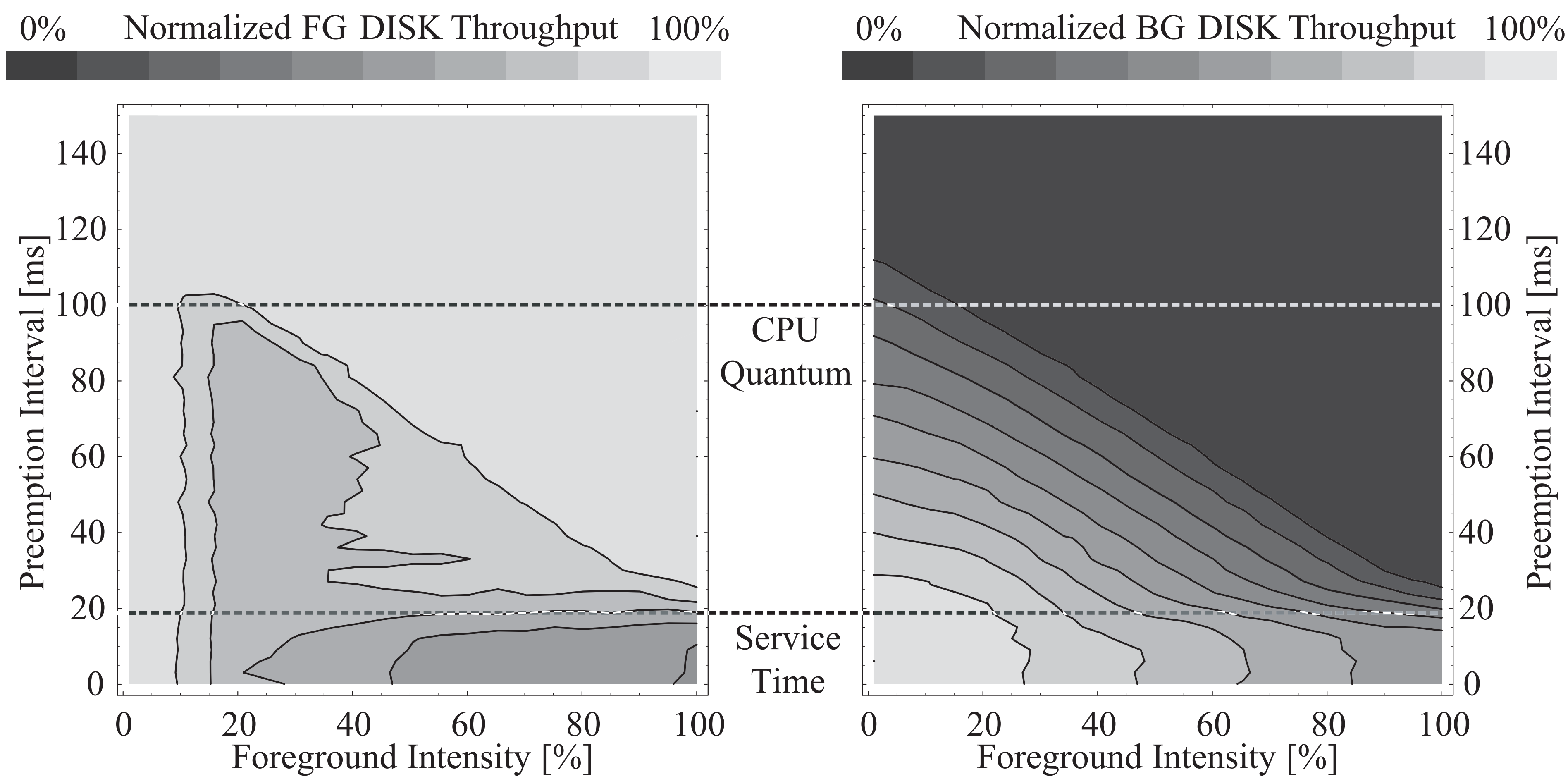
Disk Setup

- UFS file system, random data
- single disk, isolated ATA channel
  - 8.2GB Western Digital Caviar AC28200
  - 15ms maximum seek + 5ms latency (mean)
- FG + BG read 512-byte blocks
  - **2 setups:** random + sequential
- Pentium III SMP, 733Mhz, 512MB RAM
  - SMP only affects user space

Disk: Random Access

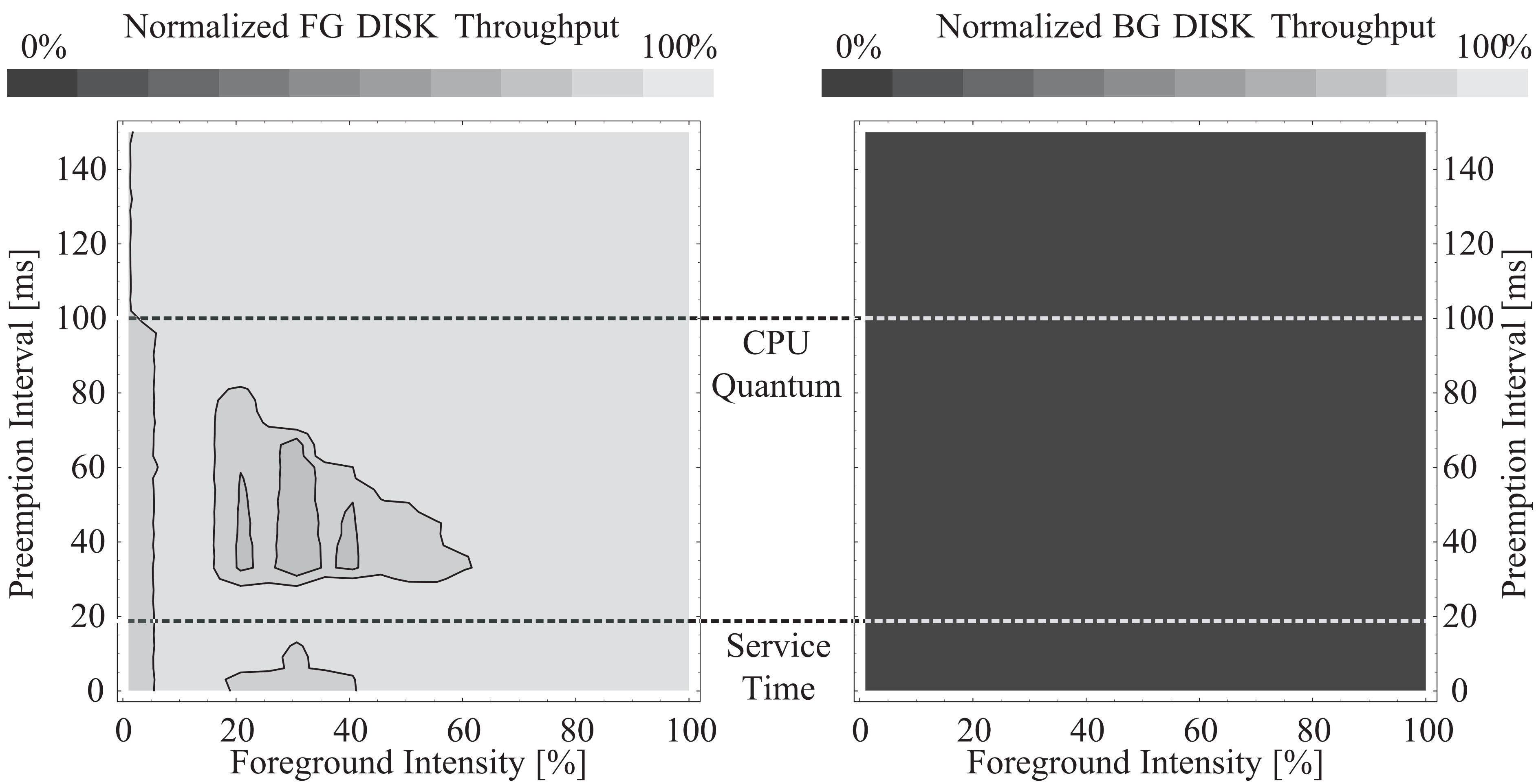
- **FG > 80%**
- **BG ≤ 90%**

of baseline throughput



Disk: Sequential Access

- FG read-ahead **multiplies workload**
  - push read-ahead (= prefetch) into idletime?

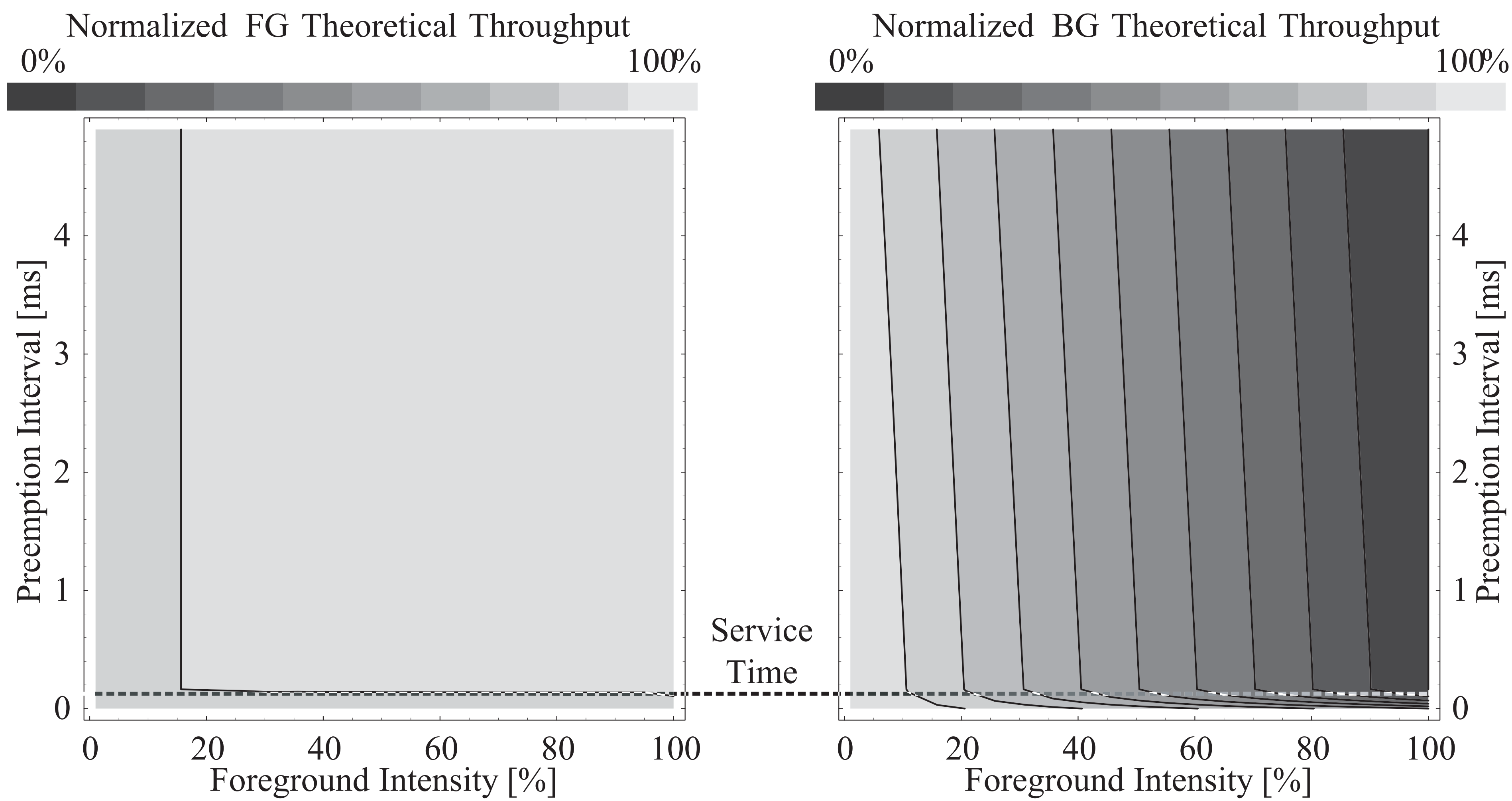


Network Setup

- direct, isolated LAN link (cross-over cord)
  - Intel PRO/1000F 1Gb/s Ethernet fiber
- source + sink hosts
  - Pentium III SMP, 733Mhz, 512MB RAM
- **combinations of UDP + TCP**
- demonstrate predictor for UDP/UDP
  - prediction, measurement, error

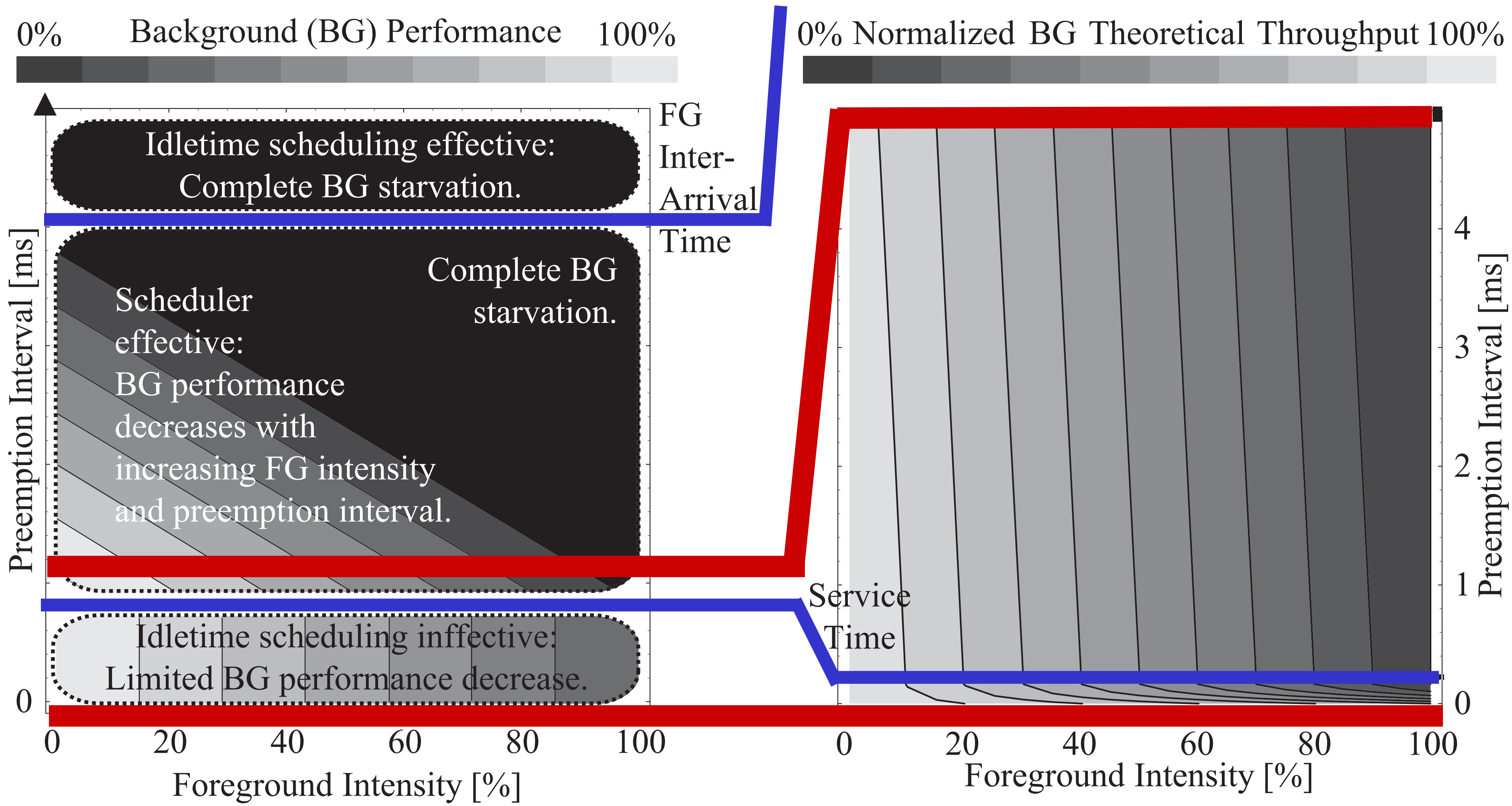
Network Prediction

- simple model analysis for UDP/UDP
  - (math, not measurement)



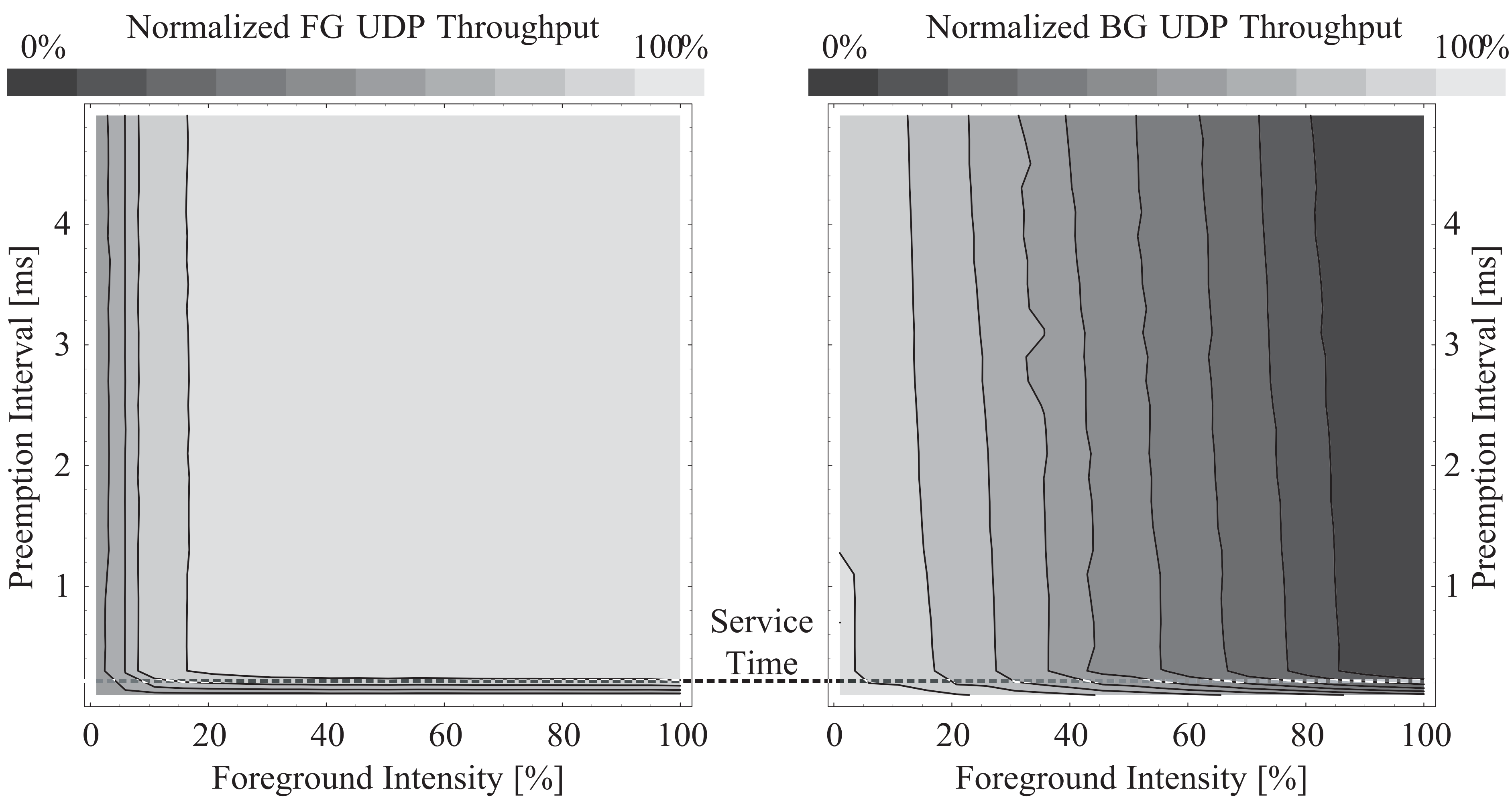
Network: PI Lengths

- PI length: **network << disk**
  - 5ms vs. 150ms



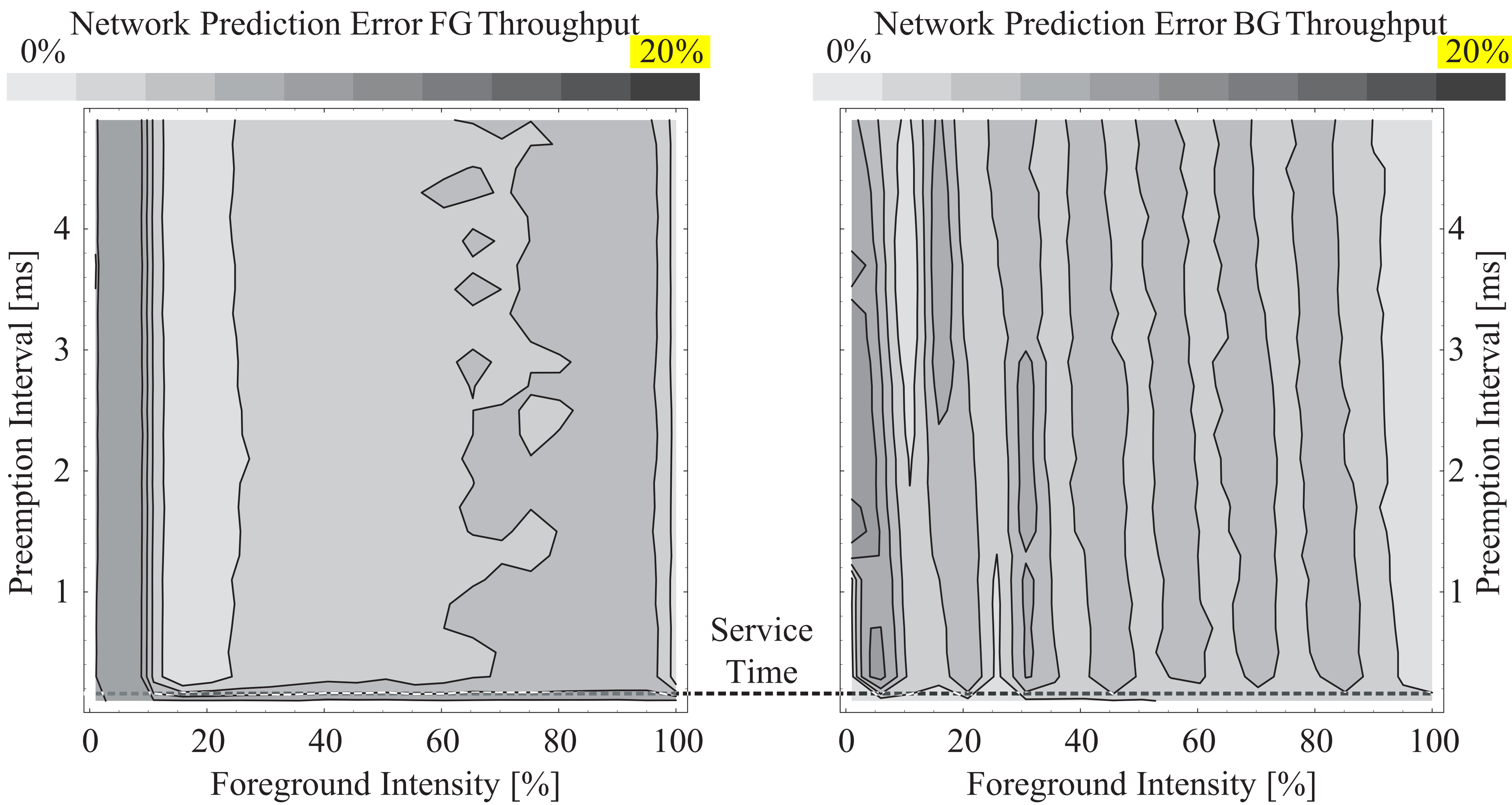
Network: UDP/UDP

- **FG > 90% + BG ≤ 90%** of baseline
  - **except:** low intensity = short FG burst



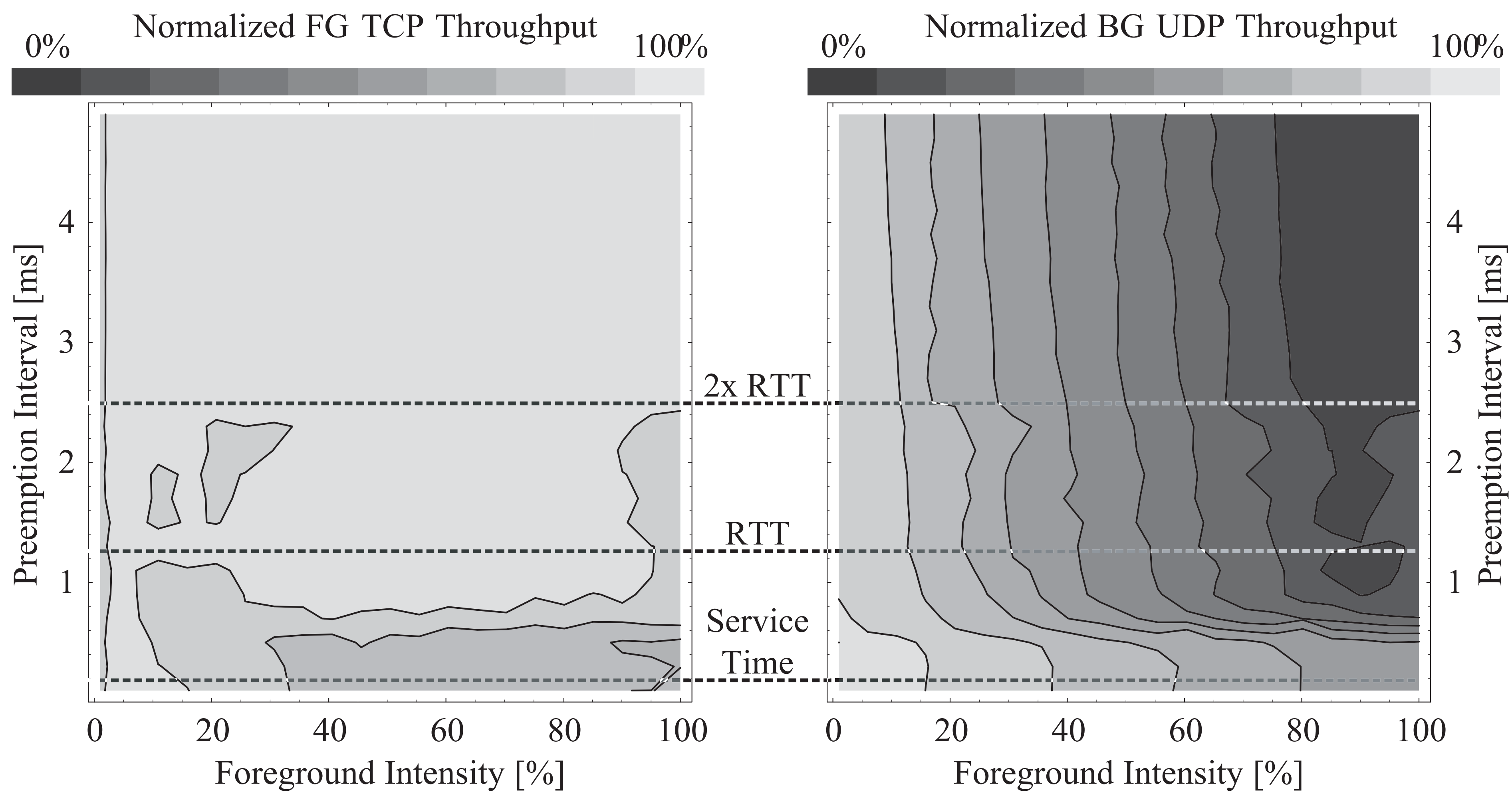
Network Prediction Error

► **error = (prediction – measurement) < 10%**



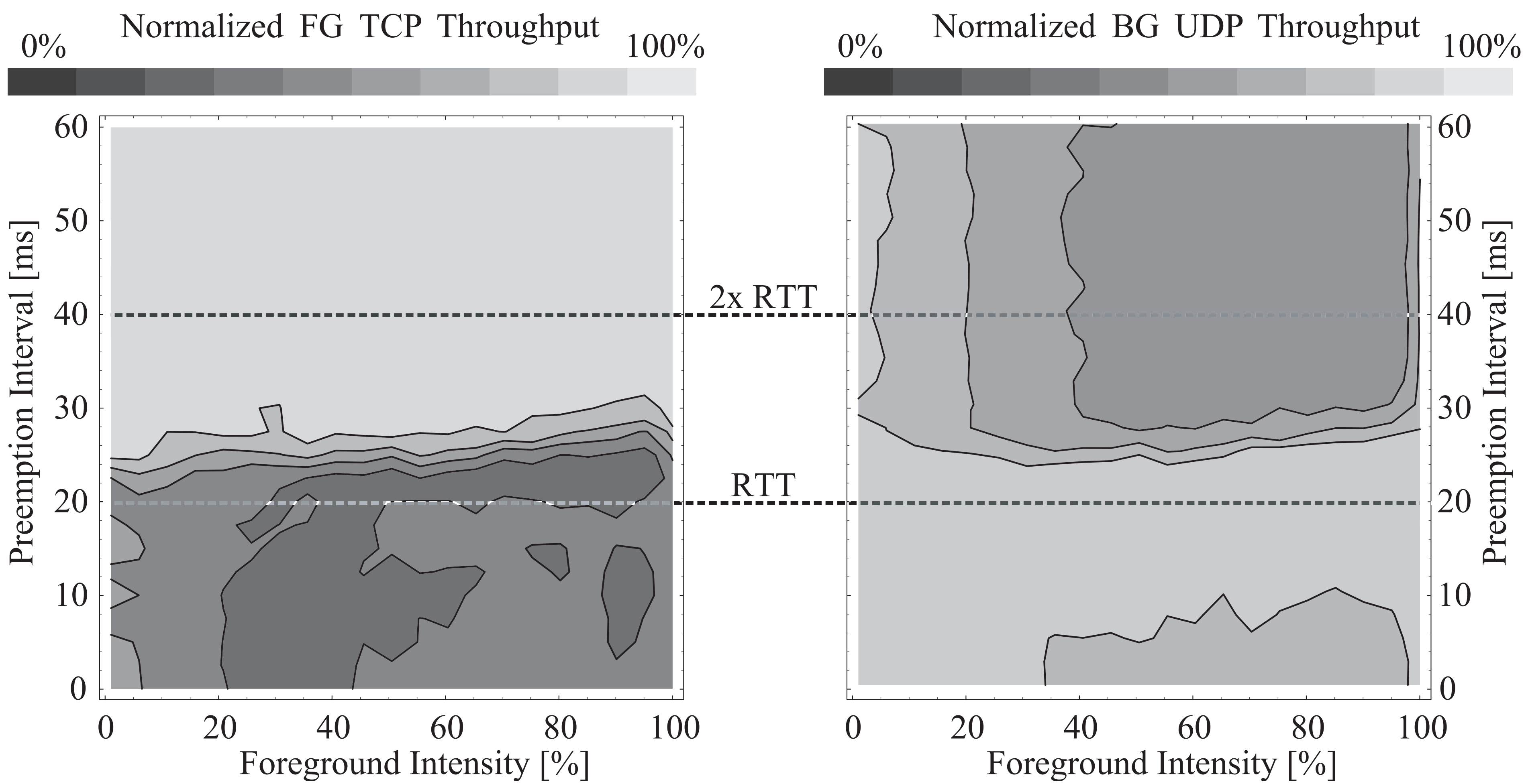
Network: TCP/UDP

- **worst case:** FG TCP vs. greedy BG UDP
  - RTT ~ PI range?



WAN: TCP/UDP

- add 10ms delay + 128KB buffer
  - 100Mb/s link (DummyNet limitation)



Discussion

- PI design workload independent
  - RTT ~ PI length?
- possible DummyNet quantization effect
- experimental goal: overall effectiveness
  - worst-case scenario: unlimited BG
- **appropriate PI → scheduler effective**

Overview

- outline
- introduction
- mechanism
- implementation
- evaluation
- **related work**
- conclusion

Related Work

- realtime systems
- network QoS
- idle capacity consumers
  - process/data migration, GRID
  - prefetching + caching
  - system optimization + maintenance
- other

Realtime Systems

- computation deadlines
  - hard/soft RTS
- 2 service levels
  - **guaranteed** → resource reservations
  - **rest** → unreserved capacity
- reservations = predictable workloads
- ignore FG impact
  - focus: deadlines, not performance
  - no isolation

Network QoS

- approaches by layer
  - L2: drop priority flag (ATM, FR)
  - L3: IP TOS, diffserv, intserv, prop-share
  - L4: *TCP-LP*, *TCP Nice*, *MulTCP*
  - L7: *Mozilla*, *BITS*, *LSAM*, push-polite
- **idletime ~ L3**
  - extend single-bit PHB into OS
- diffserv when **bottleneck = host**

Process/Data Migration

- local work → remote machine
  - process: *Condor*, *Sprite*, *V-System*, *Butler*, *Benevolent Bandit*
  - data: x@*home*, *distributed.net*, *Mether*
- detect/predict idleness: **coarse**
- focus: **1 resource**, mostly CPU
- migration **costly** (process > data)
- local idleness → **remote systems benefit**

Prefetching + Caching

- reduce access cost
  - bandwidth → prefetch
  - storage → cache
- hit rate ~ (cache size + cache content)
- **traditional:** avoid delay = limit benefit
- **idletime:** bound delay = increase benefit
  - increase cache size → idletime storage
  - improve content quality → idletime I/O

Other Related Work

- *anticipatory scheduling*
  - relax work conservation → exploit locality
  - **disk-only, no service classes**
- *MS Manners*
  - monitor BG progress: drop → suspend
  - **reactive, app-level, requires cooperation**

Overview

- outline
- introduction
- mechanism
- implementation
- evaluation
- related work
- **conclusion**

Conclusion

- generic idletime **scheduler**
  - resource + workload independent
- FG impact **predictor**
  - error < 15%
- disk + network **implementation**
  - FG > 80%
  - BG < 90%

baseline throughput + latency