Virtual Machines Recap

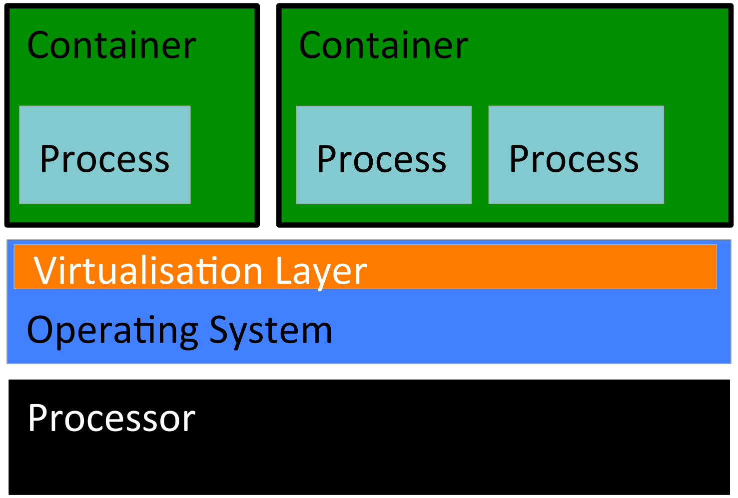
- OS development
- Allow multiple OS'es to run concurrently on same hardware (independent upgrade paths)
- Encapsulate execution environment for application stability
- Resource isolation in multi-tenant data centers
- Data center management: server consolidation, migration, checkpointing

Motivation: overhead of isolation

- VMs are great for isolation, but have significant overheads
  - ***resource*** overheads: disk (GBs) and memory (512 MB+) per VM
  - ***runtime*** overheads: CPU virtualization, I/O virtualization, etc.
  - ***administrative*** overheads: one new OS to manage per VM
  - ***ingress/egress*** overheads: moving large VHDs to/from the cloud
- …but they offer great benefits!
  - Securely isolate guest from host
  - Support live migration
  - Only (?) isolation mechanism strong enough to enable the cloud
- Can we retain their benefits with less overhead?
  - Most apps don't need to see virtualized hardware
  - Most apps don't require their own OS + drivers

OS Containers

- OS kernel modified to virtualise at syscall interface
  - Files
  - Networking
  - PIDs
  - IPC
  - User & group IDs
  - …



- Additional controls on resource allocation
  - Not just best effort
- e.g. Docker, Solaris Zones, …

Container Example: UNIX stat

stat structure, which contains the
/* ID of device containing file */
/* inode number */
/* protection */
/* number of hard links */
/* user ID of owner */
/* group ID of owner */
/* device ID (if special file) */
/* total size, in bytes */
/* blocksize for filesystem I/O */
/* number of 512B blocks allocated */

Linux Containers History

- Chroot
  - Change the root of file system
  - Originally to develop new software releases
- Jail
  - Execute process with restricted set of system calls
  - Ex: postscript viewer in web browser
- Namespaces/cgroups
  - Restrict process visibility and resource usage
  - Per-container network address translation

Containers pros/cons

- Much lower overhead
  - Only one copy of the OS kernel
  - Single level of address translation
  - Drivers not an issue – trusted in the host OS
- Tight(er) coupling between guest/host
  - Can't run different guest OS
  - Harder to encapsulate and migrate state
- … but are they secure?
  - Full OS kernel and drivers in TCB of all containers
  - Syscall interface more complex than VM interface

Threat models for isolation

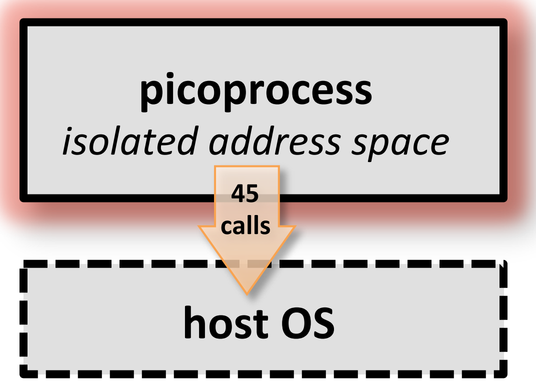
- Traditional enterprise (“friendly multi-tenant”) threat model: *employees run code of their choosing on your system*
- Cloud (multi-tenant) threat model: *anonymous hackers with unlimited access run any code of their choosing on your systems, alongside your most valued customers*
  - Do you trust an OS kernel to isolate them?
  - Do you even trust a hypervisor to isolate them?

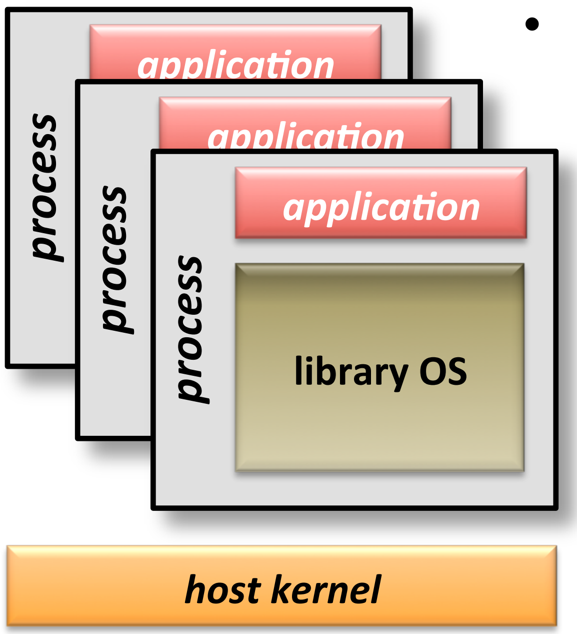
What's the Drawbridge approach?

- Key design philosophy:
  - Start with a tight, secure isolation boundary
  - Add app compatibility ***inside*** isolation container
  - ***Not*** plugging holes in a leaky but compatible interface
- Key components:
  - The *picoprocess*, an isolation mechanism
  - The *library OS*, a compatibility mechanism

Picoprocesses and library OSes

- ***Picoprocess***: concept introduced by MSR's Xax project *(Douceur et al., 2008)*
  - Isolated address space with a *very small*, fixed interface with its host
  - Lightweight, **secure isolation container**

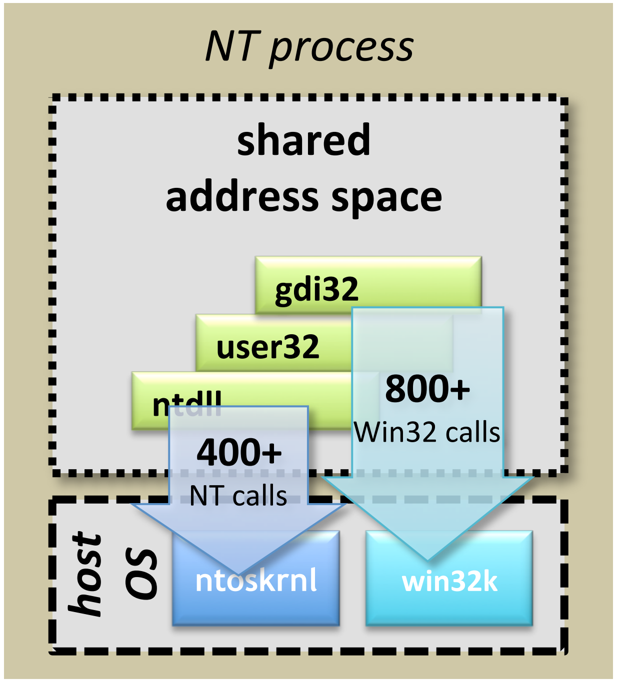


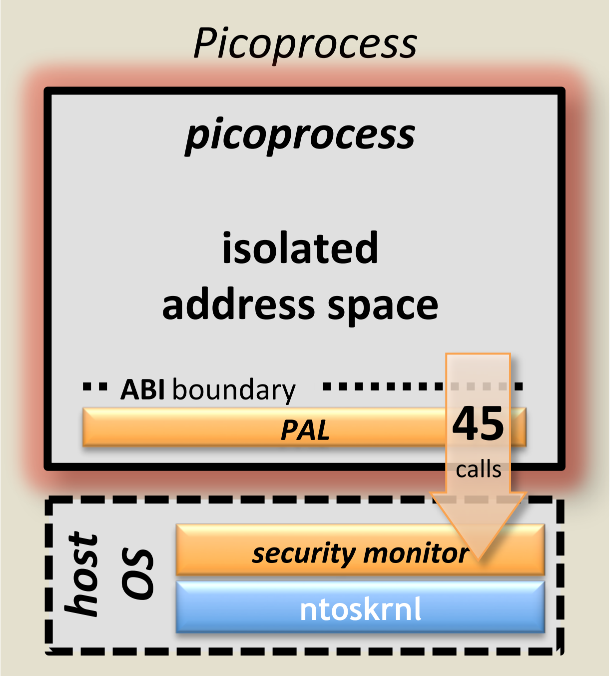


- ***Library OS***: concept championed in CS community in the '90s *(Engler et al., 1995)*
  - Minimal, shared kernel runs in supervisor mode
    - Multiplexes and abstracts hardware resources
    - Enforces cross-application protection
  - Per-app library OS **runs in user mode**
    - Constitutes OS "personality"
    - Provides application services and APIs to application
    - Runs in application's address space (user mode)
    - Each app can choose its own library OS

Drawbridge picoprocess on NT

- NT process with modified service handler
  - **All 1200+ system calls blocked** from user-mode (NTOS and win32k)
  - **45 new system calls added** to process (*Drawbridge system calls*)





The Drawbridge ABI

- ***Drawbridge ABI***: interface between a Drawbridge picoprocess and its host
  - 45 downcalls, 3 upcalls – *everything else is off-limits*
  - Designed from scratch, but heavily inspired by NT
  - APIs have fixed, closed semantics (no IOCTLs)
- Analogous to VM host/guest interface, but with higher-level abstractions
  - **threads** (not virtual CPUs)
  - **virtual memory** (not physical memory)
  - **I/O streams** (not virtual device hardware)
- Design benefits:
  - ***security -*** interface is small enough to undergo manual review / inspection
  - ***portability*** - Windows apps run unmodified on any system that implements 45 functions
  - ***flexibility*** – interface allows app's state to live (almost) entirely in process

Drawbridge ABI

(excerpt)

Threading

```
DkThreadCreate
DkSemaphoreCreate
DkSemaphorePeek
DkSemaphoreRelease
DkObjectsWaitAny
...
```

Memory management

```
DkVirtualMemoryAllocate
DkVirtualMemoryFree
DkVirtualMemoryProtect
```

I/O streams

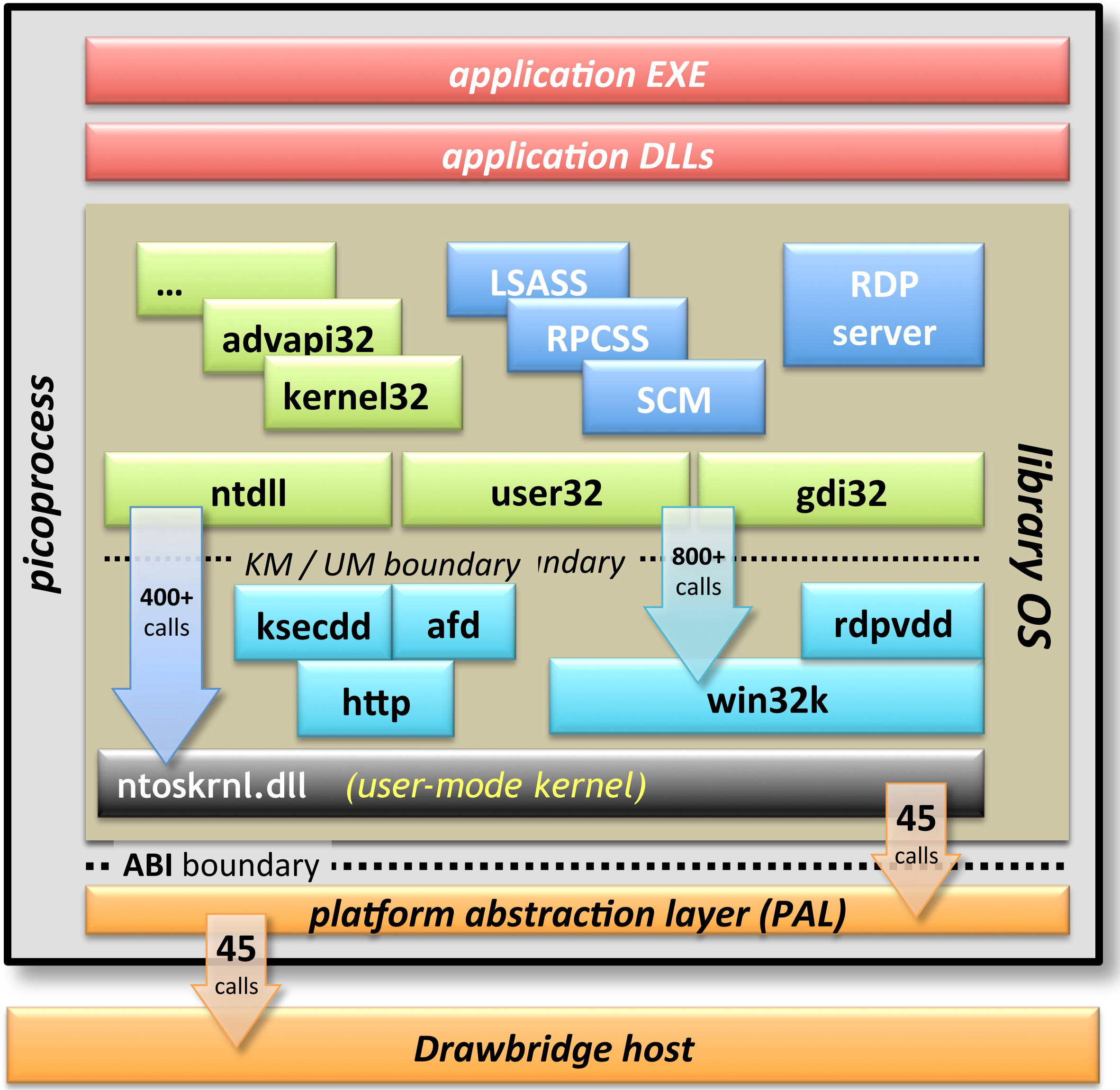
```
DkStreamOpen
DkStreamRead
DkStreamWrite
DkStreamMap
DkStreamFlush
...
```

Upcalls

```
LibOsInitialize
LibOsThreadStart
LibOsExceptionDispatch
```

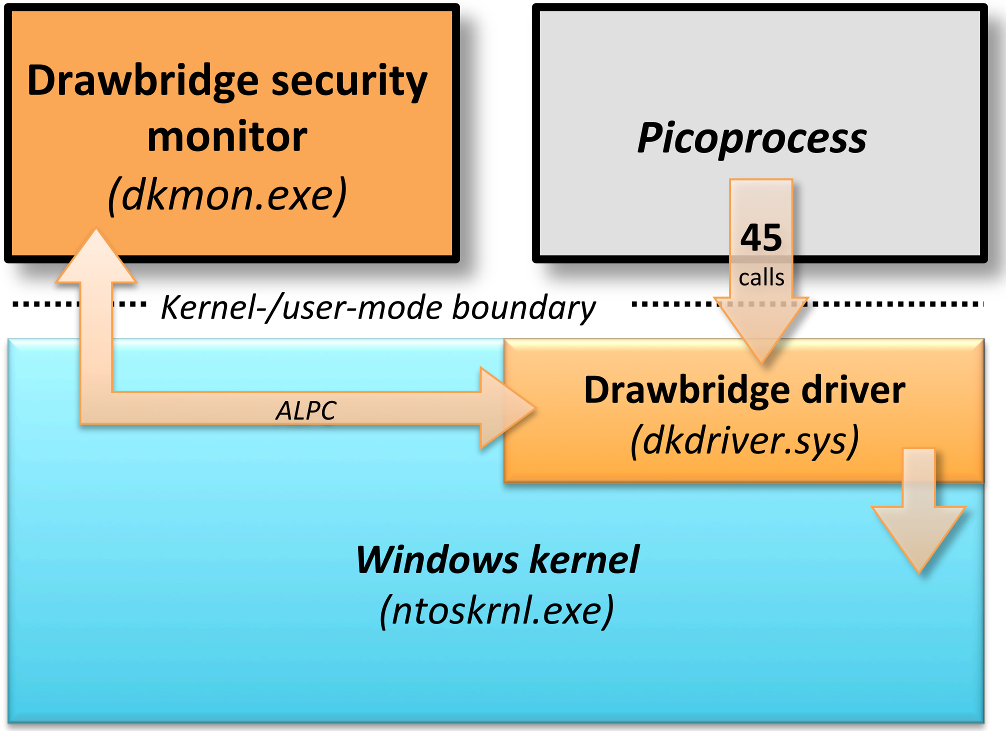
The Windows library OS

- Based on Windows OS
  - Same binaries *(where possible)*
  - Same architecture
- Windows ***enlightened* to run in a picoprocess** with the app
  - lifted into user mode
  - most changes in user-mode kernel
- Example library OS: *Win7 SP1*
  - **100MB on disk** (~150 DLLs)
  - **16MB of working set** + app
  - 5.5+ MLoC for 15,000+ Win32 APIs
- Each picoprocess runs its own library OS
  - app chooses its library OS
  - version need not match across picoprocesses or host



The Drawbridge-on-Windows host

- ***Drawbridge host*** implements 45-function ABI atop Windows
- Analogous to Hyper-V's hypervisor + virtualization stack
- Split between kernel-mode driver and user-mode worker
  - Driver implements ABI
  - Driver consults security monitor for policy decisions



Drawbridge security monitor
(dkmon.exe)

Picoprocess

45
calls

Kernel-/user-mode boundary

ALPC

Drawbridge driver
(dkdriver.sys)

Windows kernel
(ntoskrnl.exe)

The Drawbridge security monitor

- ***Security monitor*** – user-mode half of Drawbridge host
  - launches app in picoprocess
  - makes access policy decisions
  - “normal” NT process
- Policy decisions based on *manifests*
  - All external resources are blocked by default
  - Resources can be white-listed back in by admin
  - Access specified via virtual to physical namespace mappings

Drawbridge security monitor
(dkmon.exe)

Sample Policy

```
[Namespace.Writable]
pipe.server:///RDP=pipe.server:///RDP_Drawbridge         ; expose ‘RDP’ named pipe server out of
                                                         ; process as ‘RDP_Drawbridge’
tcp.server://localhost:3000=tcp.server://localhost:3000  ; allow app to listen (only) on port 3000
tcp.client:=tcp.client:                                  ; allow use of any TCP client socket

[Namespace]
file:///users/jdoe/documents=file:///documents           ; allow R/O access to Documents folder
```

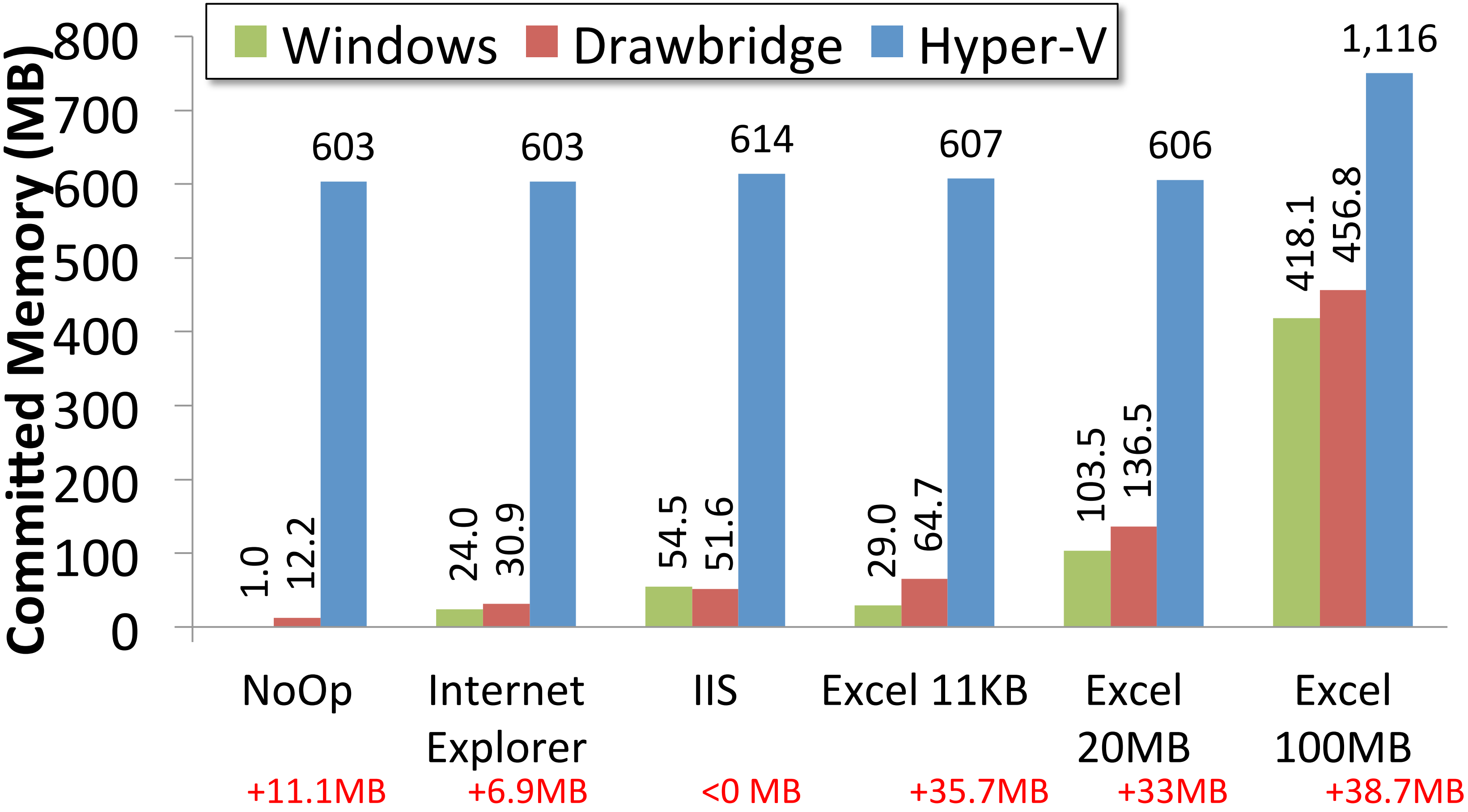
Drawbridge packages

- ***Drawbridge package*** – self-contained, self-describing unit of deployment
- A package contains:
  - Manifest
    - Identity (name, version, options)
    - Dependencies on other packages
    - Access control policy requirements
    - Relative paths to important contained files (e.g. app EXE)
  - Files
  - Registry data (.reg format)
  - Debug resources (e.g. symbols, etc.)
- Everything's a package: app, library, library OS, suspended app
- Security monitor resolves transitive closure of packages and dependencies
  - **File content from packages is unioned** into virtual FS
  - **Registry content from packages is unioned** into virtual registry
  - Packages are read-only, mapped copy-on-write

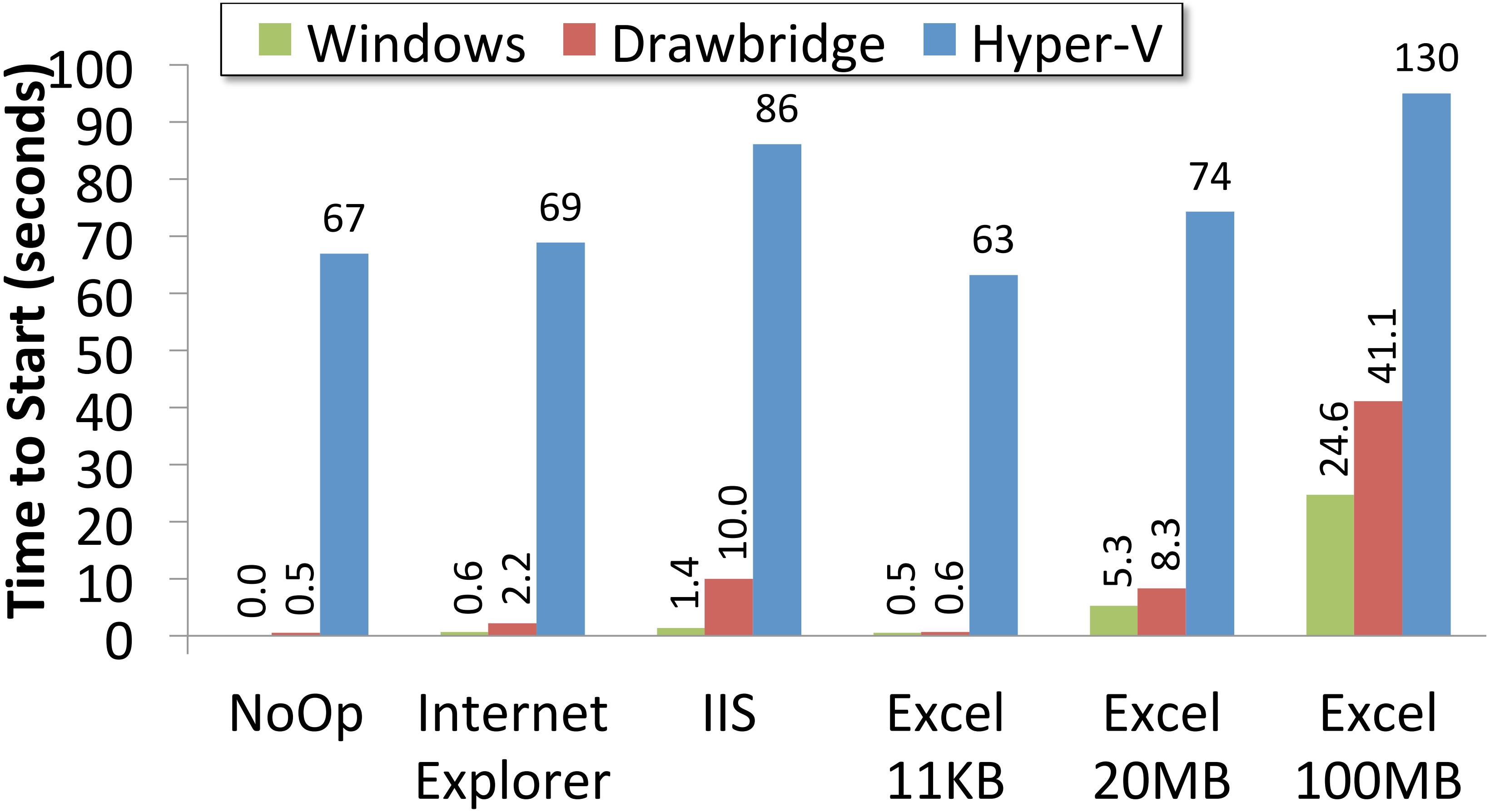
Sample Manifest

```
[Package]
ManifestVersion=1
PackageRevision=4

[Identity]
Name=IISWorker
MajorVersion=7
MinorVersion=5
BuildNumber=7601
Architecture=x64

[Dependency.Win7]
Name=Windows
MajorVersion=6
MinorVersion=1

[Dependency.CLR4]
Name=MicrosoftNET
MajorVersion=4
MinorVersion=0

[Windows.Application]
Exe=package:///windows/
system32/inetsrv/w3wp.exe

[Windows.Registry]
File:///w3wp.exe.dbreg
```

Committed Memory by Apps



Time to Start Application Package



Scheduling

Multilevel Scheduling Examples

- Virtual machine abstraction: no information about underlying resource sharing
- Spark task assignment: how should it partition mapreduce or ML tasks?
  - One per server? What if some servers are busier/slower than others? What if some partitions take more time than others?
  - Many partitions per server? More overhead, more communication
  - How does OS scheduler know which task will be last?

Multilevel Scheduling

- Process abstraction: no information about physical resources
- Parallel application: how should it split its work?
  - One thread per hyperthread? One thread per core? What if thread takes a page fault?
  - Many threads per hyperthread? More coherence traffic, more overhead. What if many competing tasks?
  - How does application tell kernel which thread to run first? What if task priority is dynamic?

Multilevel Scheduling

- Virtual machine abstraction: guest OS has no information about physical memory
- Host OS chooses a page to evict; writes changes to physical disk
- Guest OS chooses same page to evict; writes changes to virtual disk, faulting in physical page
- VMWare balloon driver communicates resource usage across host/guest OS boundary

Multilevel Scheduling

- Virtual memory: application has no information as to which pages are in physical memory
- OS evicts unused pages, writes changes to disk
- Application uses a garbage collector: some pages are in use, some unused, some garbage
- Application coalesces used data, collects garbage
- Unused garbage pages evicted to disk, brought back in for GC, empty pages re-written to disk

Multilevel Scheduling Revisited

- Many (!) cases where a layer wants to do its own resource management
- But runs on another layer that provides abstraction of virtual resources
- Solutions?
  - Live with it
  - Change the API

Mach External Pager

- When Mach chooses a page to evict, it upcalls to an external pager to do the eviction
  - Original motivation: allow paging over network
- External pager can choose a different page to evict
  - user-level access to page use/modify bits in VTx
  - Kernel only decides how many pages per app
  - Self-paging => better isolation

Scheduler Activations

- Kernel allocates processors to apps
- User-level threads, scheduled at user level
  - Faster! No kernel trap for blocking locks, CVs
  - User-level control over priorities
- Kernel upcalls
  - When new processor is assigned
  - (on different CPU) when processor is taken away
  - Syscall/page fault blocks in kernel

Scheduler Activation Mechanism

- Example: user-level thread does file read, misses in buffer cache, blocks in kernel
- Normal
  - save kernel context, switch to new thread
  - When I/O completes, switch back
- New:
  - Save kernel context, create new thread to do upcall, switch to that thread
  - When I/O completes, complete syscall, then upcall
  - Advanced version: pipeline upcall events

Transparent Asynch I/O

- Many kernels have both synch and asynch I/O
  - Synch: syscall blocks until operation completes
  - Asynch: syscall returns immediately, kernel thread completes operation in background, upcall when done
- Implementation: Synchronous syscall with upcall
  - If blocks, do upcall; user lib schedules new thread
  - When I/O completes, complete syscall
  - When done, "return" by doing another upcall
  - User lib runs the user-level syscall return

Scheduling

- Policy: what to do next, when there are multiple threads ready to run (or packets, or web requests, or …)
- Uniprocessor policies
  - FIFO, round robin, optimal
  - multilevel feedback as approximation of optimal
- Multiprocessor policies
  - Affinity scheduling, gang scheduling
- Queueing theory
  - Can you predict/improve a system's response time?
- Control theory
  - How to achieve response time goals, tail latency, …

Example

- You manage a web site, that suddenly becomes wildly popular. Performance starts to degrade. Do you?
  - Buy more hardware?
  - Implement a different scheduling policy?
  - Turn away some users? Which ones?
- How much worse will performance get if the web site becomes even more popular?

Definitions

- Task/Job
  - User request: e.g., mouse click, web request, shell command, …
- Latency/response time
  - How long does a task take to complete?
- Tail latency
  - How consistent is task response time?
- Throughput
  - How many tasks can be done per unit of time?
- Overhead
  - How much extra work is done by the scheduler?
- Fairness
  - How equal is the performance received by different users?
- Strategy-proof
  - Can a user manipulate the system to gain more than their fair share?

More Definitions

- Workload
  - Set of tasks for system to perform
- Preemptive scheduler
  - If we can take resources away from a running task
- Work-conserving
  - Resource is used whenever there is a task to run
  - For non-preemptive schedulers, work-conserving is not always better
- Scheduling algorithm
  - takes a workload as input
  - decides which tasks to do first
  - Performance metric (throughput, latency) as output
  - Only preemptive, work-conserving schedulers to be considered

First In First Out (FIFO)

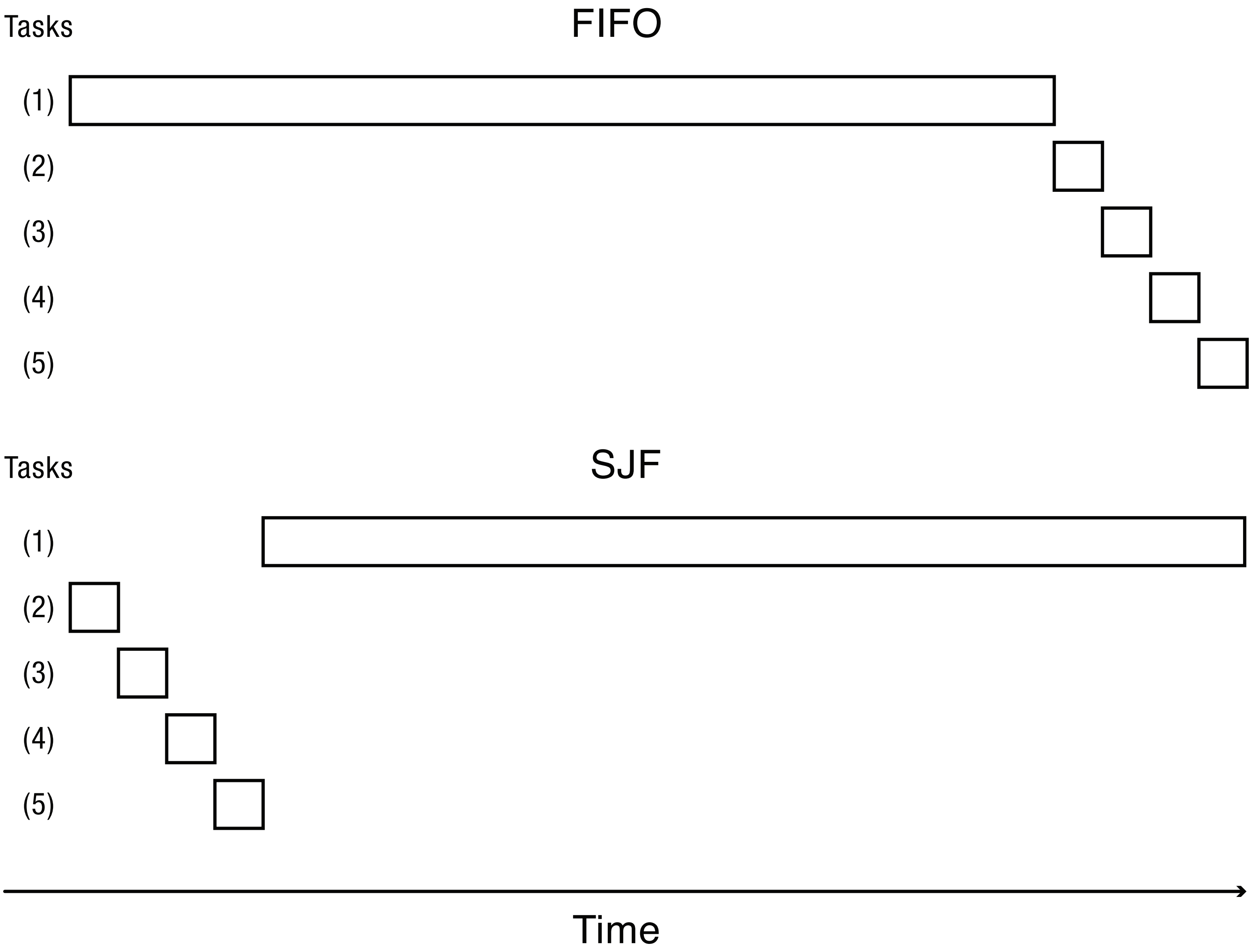
- Schedule tasks in the order they arrive
  - Continue running them until they complete or give up the processor
- Example: memcached
  - Facebook cache of friend lists, …
- On what workloads is FIFO particularly bad?

Shortest Job First (SJF)

- Always do the task that has the shortest remaining amount of work to do
  - Often called Shortest Remaining Time First (SRTF)

- Suppose we have five tasks arrive one right after each other, but the first one is much longer than the others
  - Which completes first in FIFO? Next?
  - Which completes first in SJF? Next?

FIFO vs. SJF



Tasks

FIFO

(1)

(2)

(3)

(4)

(5)

Tasks

SJF

(1)

(2)

(3)

(4)

(5)

Time

Question

- Claim: SJF is optimal for average response time
  - Why?

- Does SJF have any downsides?

Question

- Is FIFO ever optimal?

- Pessimal?

Starvation and Sample Bias

- Suppose you want to compare two scheduling algorithms
  - Create some infinite sequence of arriving tasks
  - Start measuring
  - Stop at some point
  - Compute average response time as the average for completed tasks between start and stop
- Is this valid or invalid?

Sample Bias Solutions

- Measure for long enough that # of completed tasks >> # of uncompleted tasks
  - For both systems!
- Start and stop system in idle periods
  - Idle period: no work to do
  - If algorithms are work-conserving, both will complete the same tasks

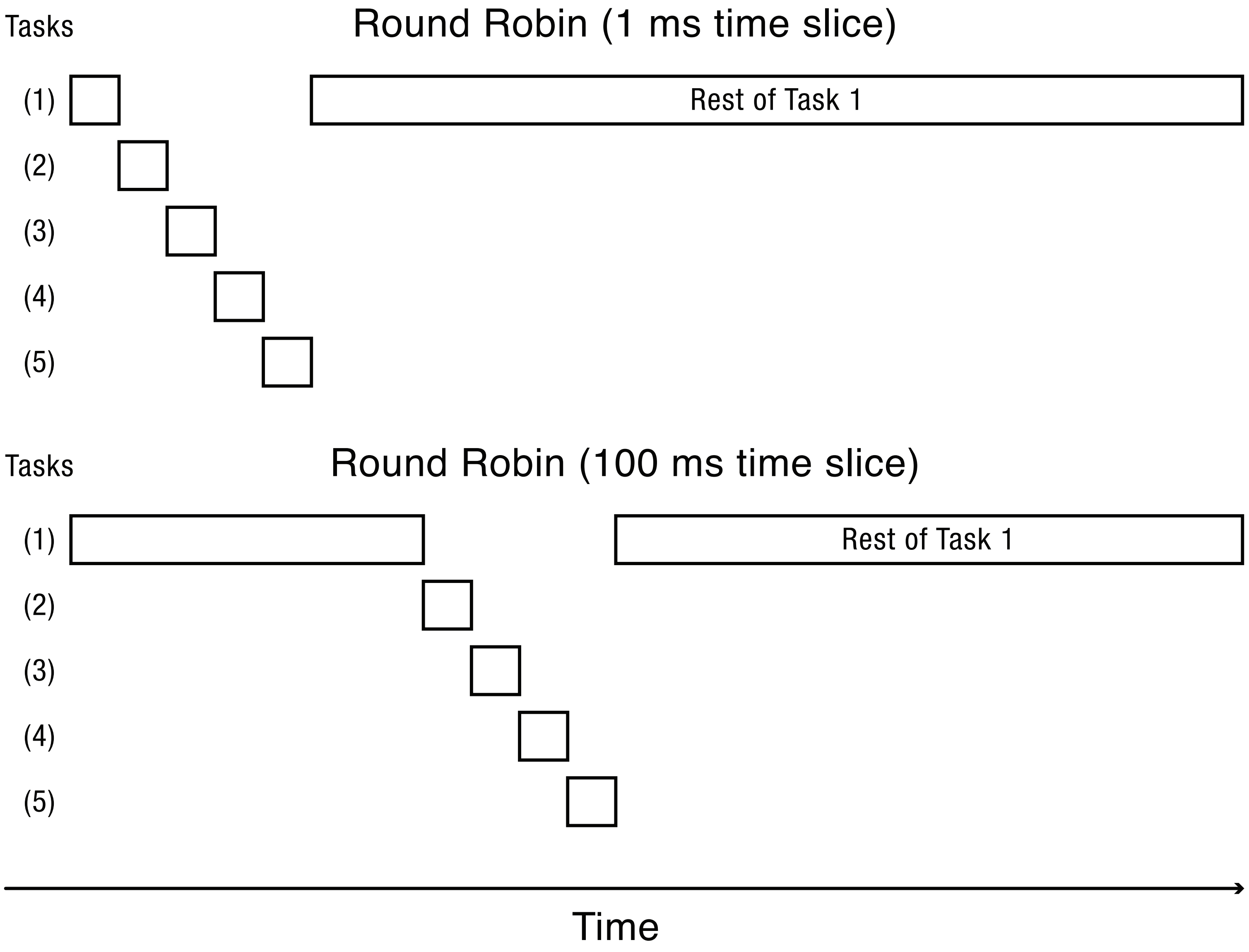
Tail Latency

- What if we are optimizing for tail latency and not average responsiveness?
- Minimize max response time?
  - FIFO? Longest job first?
- SLA: minimize % over max response time?
  - FIFO or SJF with early discard?
- Min-max inflation factor in response time?
  - Round Robin

Round Robin

- Each task gets resource for a fixed period of time (time quantum)
  - If task doesn't complete, it goes back in line
- Need to pick a time quantum
  - What if time quantum is too long?
    - Infinite?
  - What if time quantum is too short?
    - One instruction -> Hyperthreading!

Round Robin



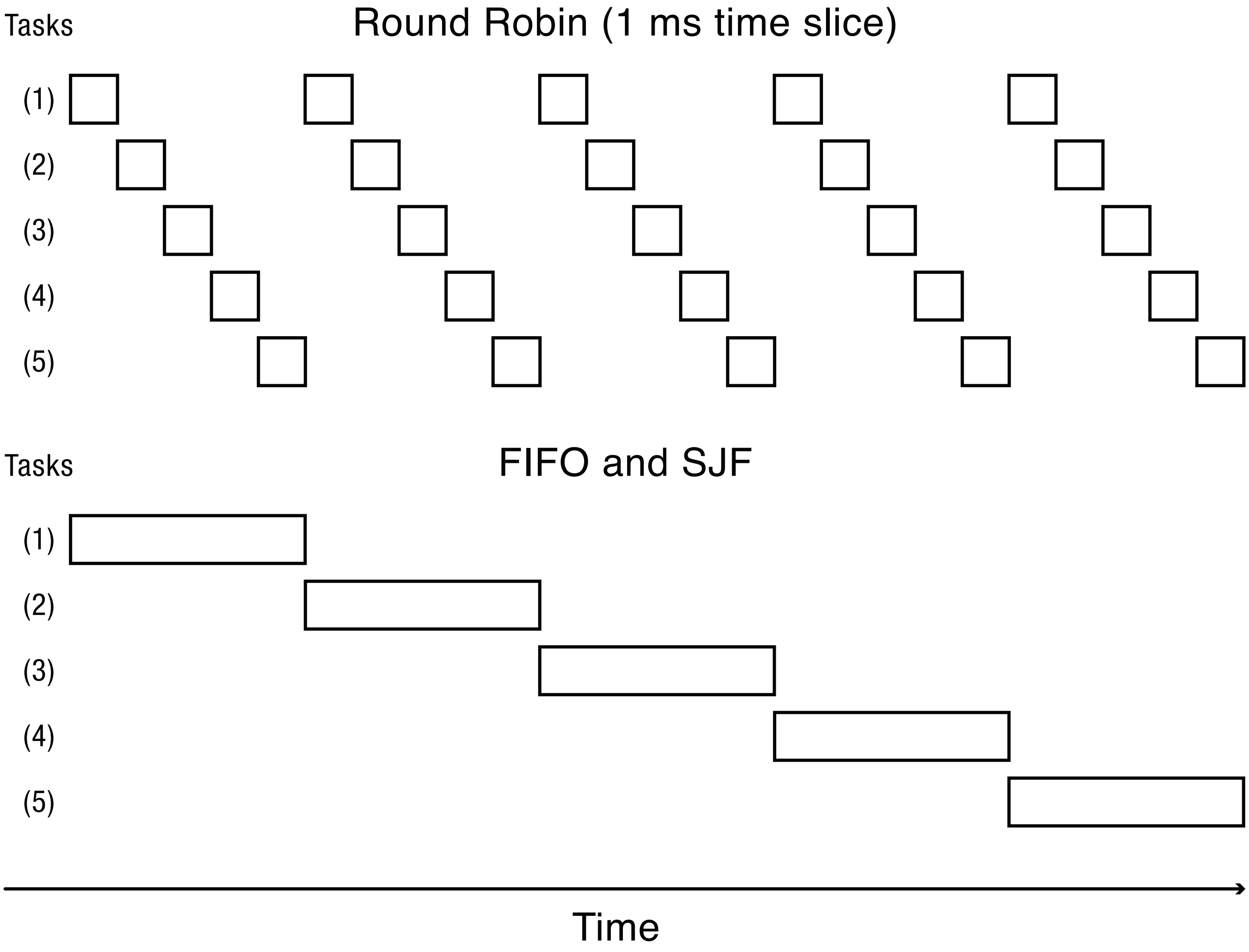
Round Robin (1 ms time slice)

Round Robin (100 ms time slice)

Round Robin vs. FIFO

- Assuming zero-cost time slice, is Round Robin always better than FIFO?

Round Robin vs. FIFO



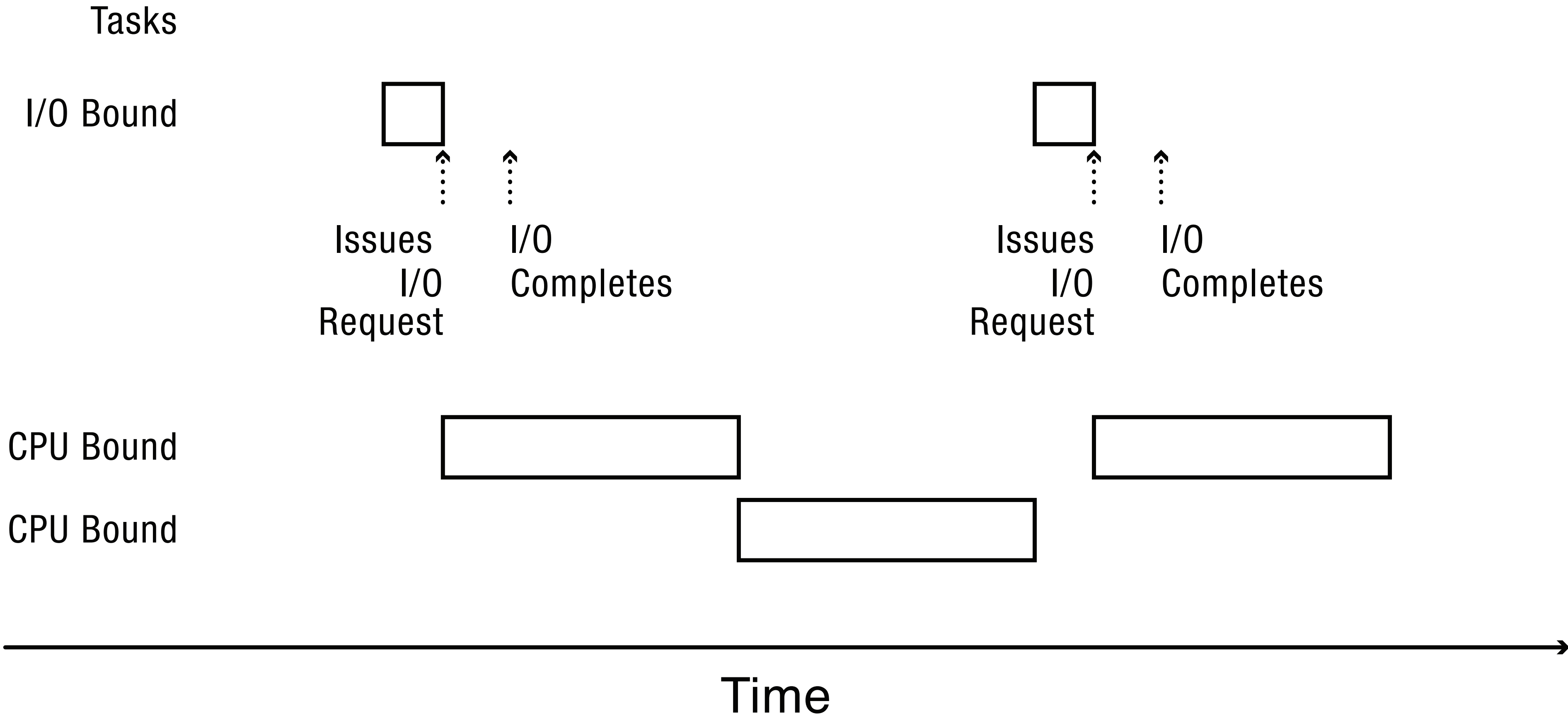
Round Robin (1 ms time slice)

FIFO and SJF

Max-Min Fairness

- Applies to repeating tasks
  - Ex: network bandwidth allocation
- Maximize the min allocation given to a task
  - If any task needs less than an equal share, schedule the smallest of these first
  - Split the remaining time using max-min
  - If all remaining tasks need at least equal share, split evenly
- Implementation
  - Add credits to each task at same rate, debit on use (age)
  - Randomly choose proportional to # of credits

Mixed Workload



Tasks

I/O Bound

Issues I/O Request

I/O Completes

Issues I/O Request

I/O Completes

CPU Bound

CPU Bound

Time

Multi-level Feedback Queue (MFQ)

- Goals:
  - Responsiveness
  - Low overhead
  - Starvation freedom
  - Some tasks are high/low priority
  - Fairness (among equal priority tasks)
- Not perfect at any of them!
  - Used in Linux (and probably Windows, MacOS)

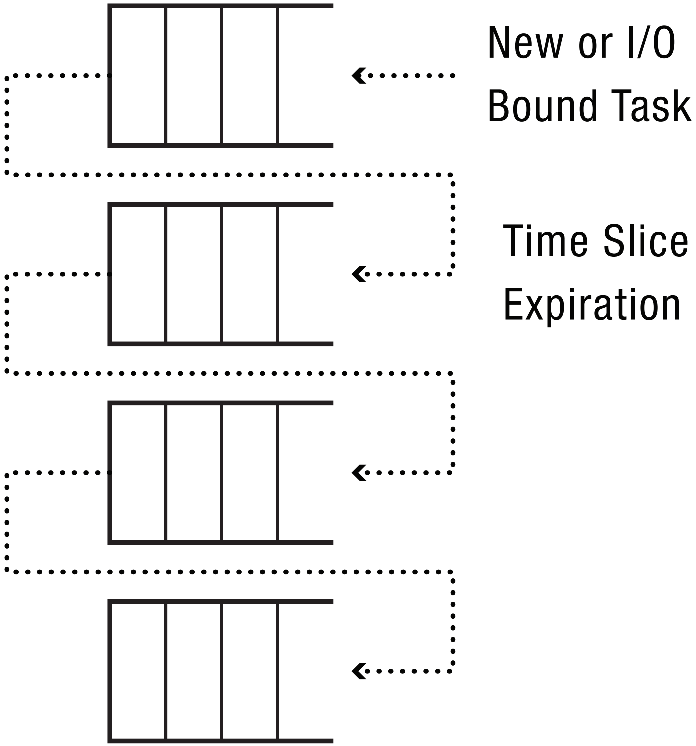
MFQ

- Set of Round Robin queues
  - Each queue has a separate priority
- High priority queues have short time slices
  - Low priority queues have long time slices
- Scheduler picks first thread in highest priority queue
- Tasks start in highest priority queue
  - If time slice expires, task drops one level

MFQ

Priority	Time Slice (ms)
1	10
2	20
3	40
4	80

Round Robin Queues



MFQ and Tail Latency

- How predictable is a task's performance?
  - Can it be affected by other users?

- Linux boosts priority to tasks being starved…

MFQ and Strategy

- Can a user get better performance (response time, throughput) by doing useless work?

Uniprocessor Summary (1)

- FIFO is simple and minimizes overhead.
- If tasks are variable in size, then FIFO can have very poor average response time.
- If tasks are equal in size, FIFO is optimal in terms of average response time.
- Considering only the processor, SJF is optimal in terms of average response time.
- SJF is pessimal in terms of variance in response time.

Uniprocessor Summary (2)

- If tasks are variable in size, Round Robin approximates SJF.
- If tasks are equal in size, Round Robin will have very poor average response time.
- Tasks that intermix processor and I/O can do poorly under Round Robin.

Uniprocessor Summary (3)

- Max-Min fairness can improve response time for I/O-bound tasks.
- Round Robin and Max-Min both avoid starvation.
- MFQ approximates SJF
  - High variance for long jobs; vulnerable to strategy

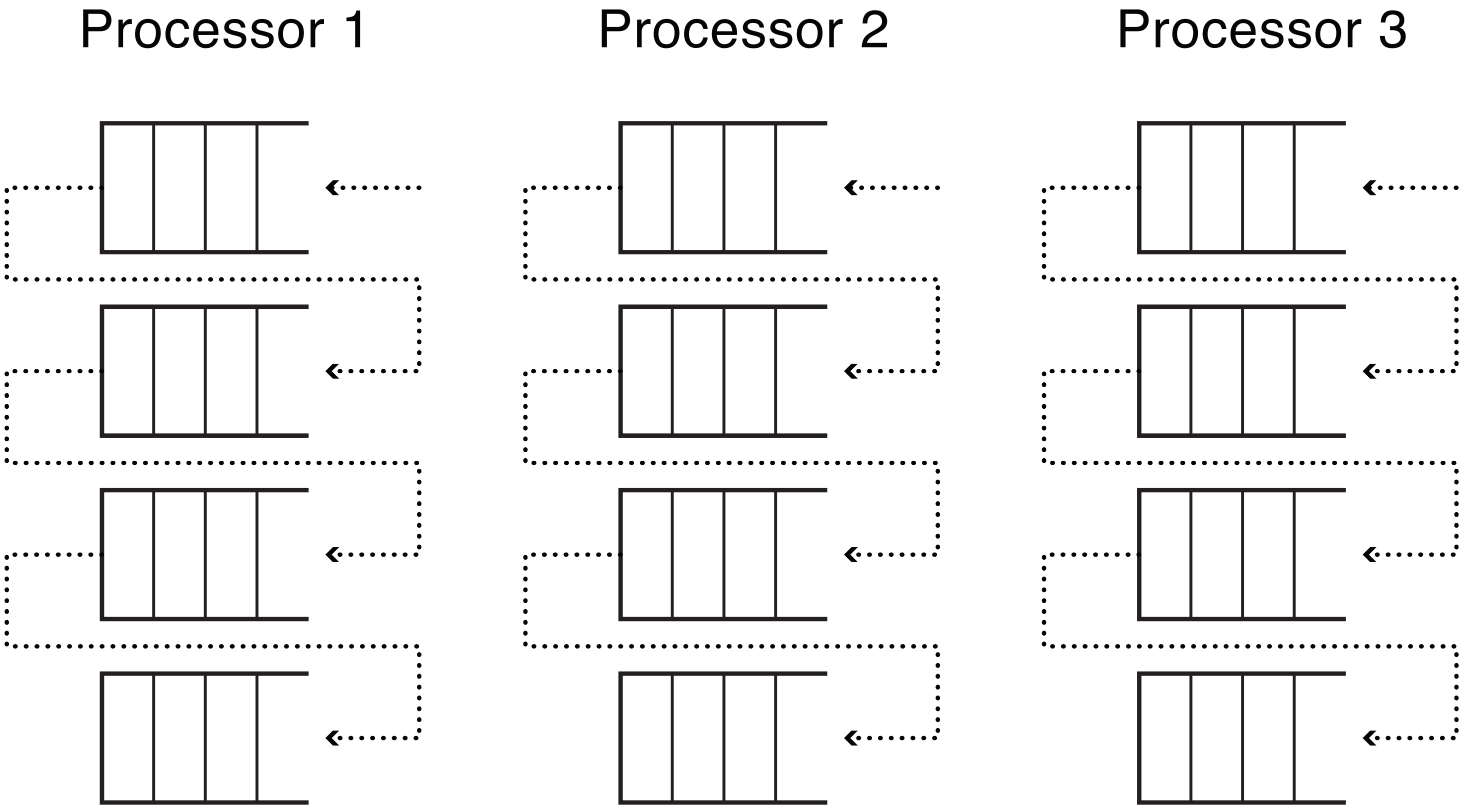
Multiprocessor Scheduling

- What would happen if we used MFQ on a multiprocessor?
  - Contention for scheduler spinlock
  - Cache slowdown due to ready list data structure pinging from one CPU to another
  - Limited cache reuse: thread's data from last time it ran is often still in its old cache

Per-Processor Affinity Scheduling

- Each processor has its own ready list
  - Protected by a per-processor spinlock
- Put threads back on the ready list where it had most recently run
  - Ex: when I/O completes, or on Condition->signal
- Idle processors can steal work from other processors

Per-Processor Multi-level Feedback with Affinity Scheduling



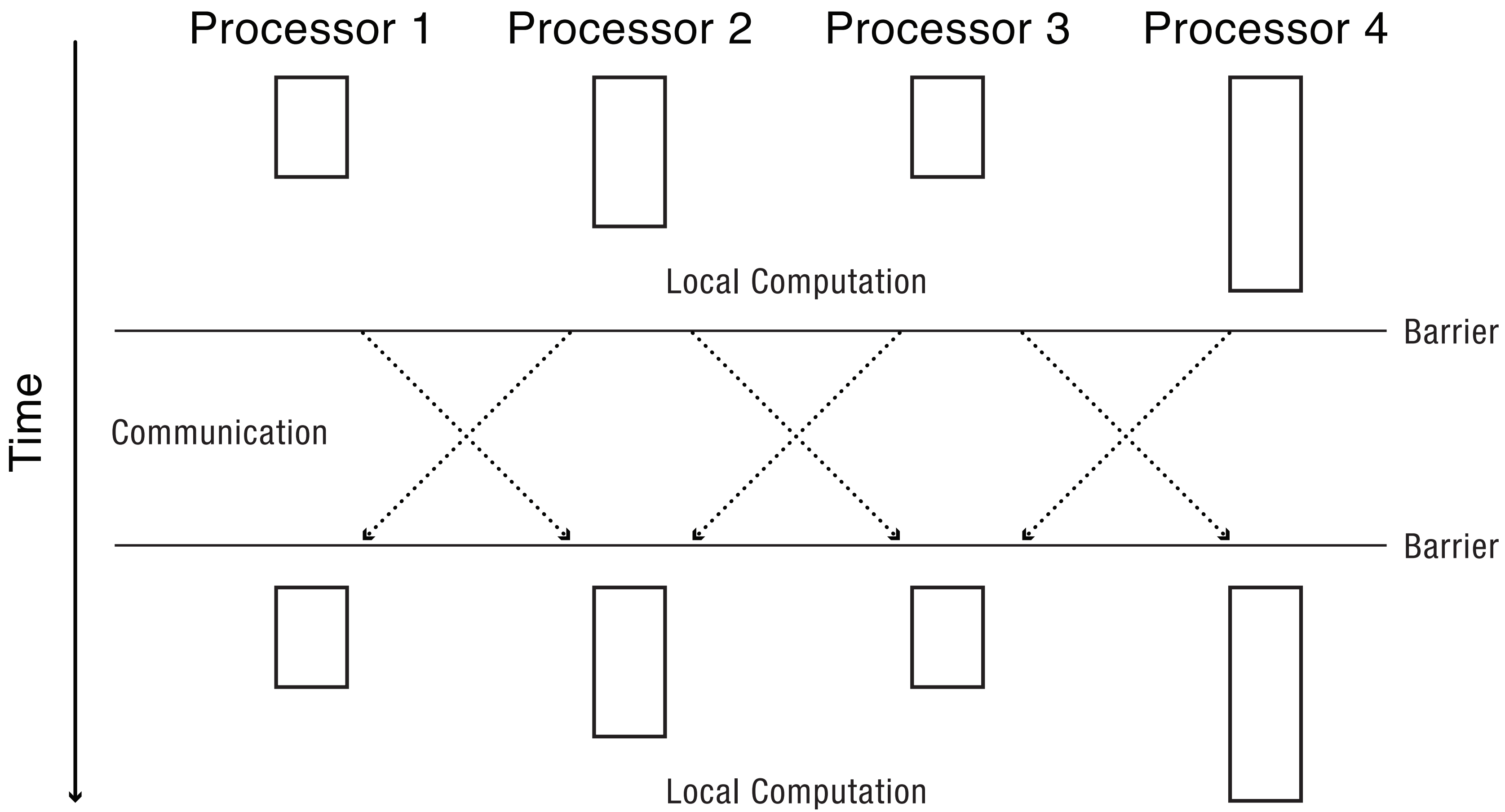
Scheduling Parallel Programs

- What happens if one thread gets time-sliced while other threads from the same program are still running?
  - Assuming program uses locks and condition variables, it will still be correct
  - What about performance?

Bulk Synchronous Parallelism

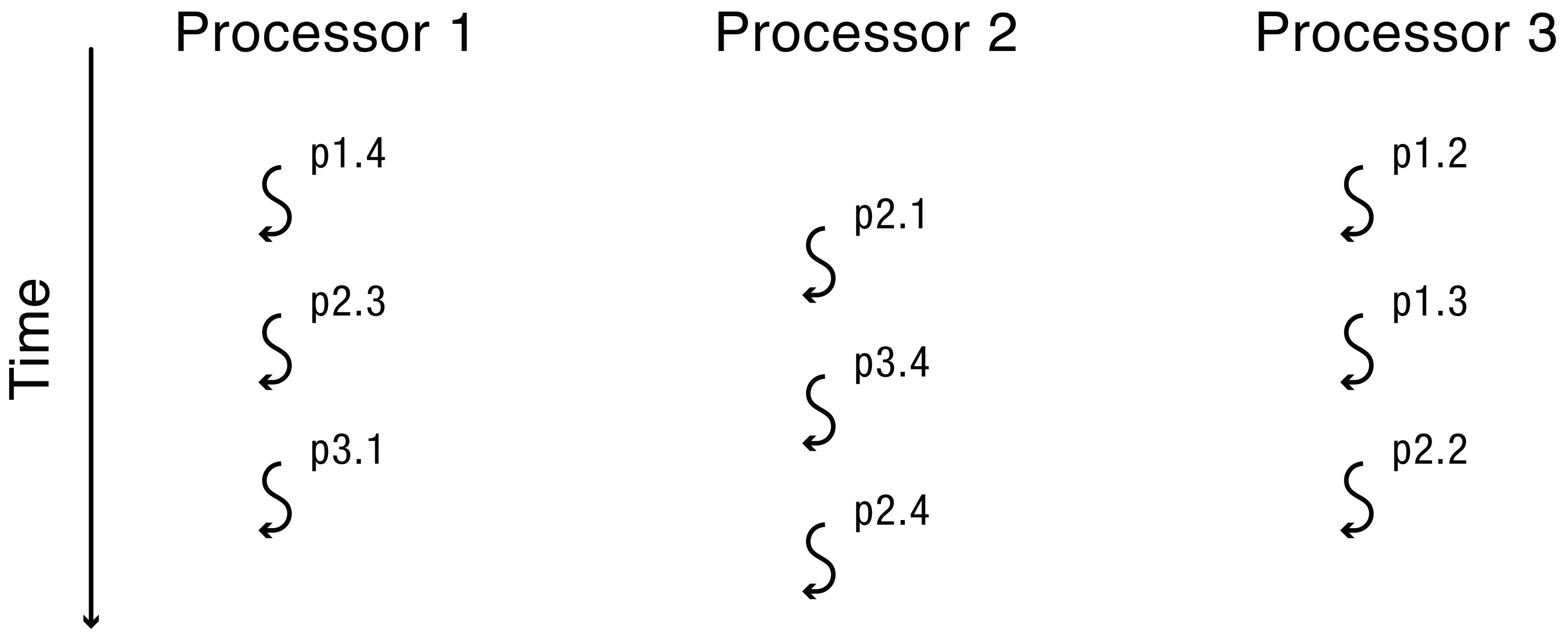
- Loop at each processor:
  - Compute on local data (in parallel)
  - Barrier
  - Send (selected) data to other processors (in parallel)
  - Barrier
- Examples:
  - MapReduce
  - Fluid flow over a wing
  - Most parallel algorithms can be recast in BSP, sacrificing at most a small constant factor in performance

Tail Latency



Scheduling Parallel Programs

Oblivious: each processor time-slices its ready list independently of the other processors

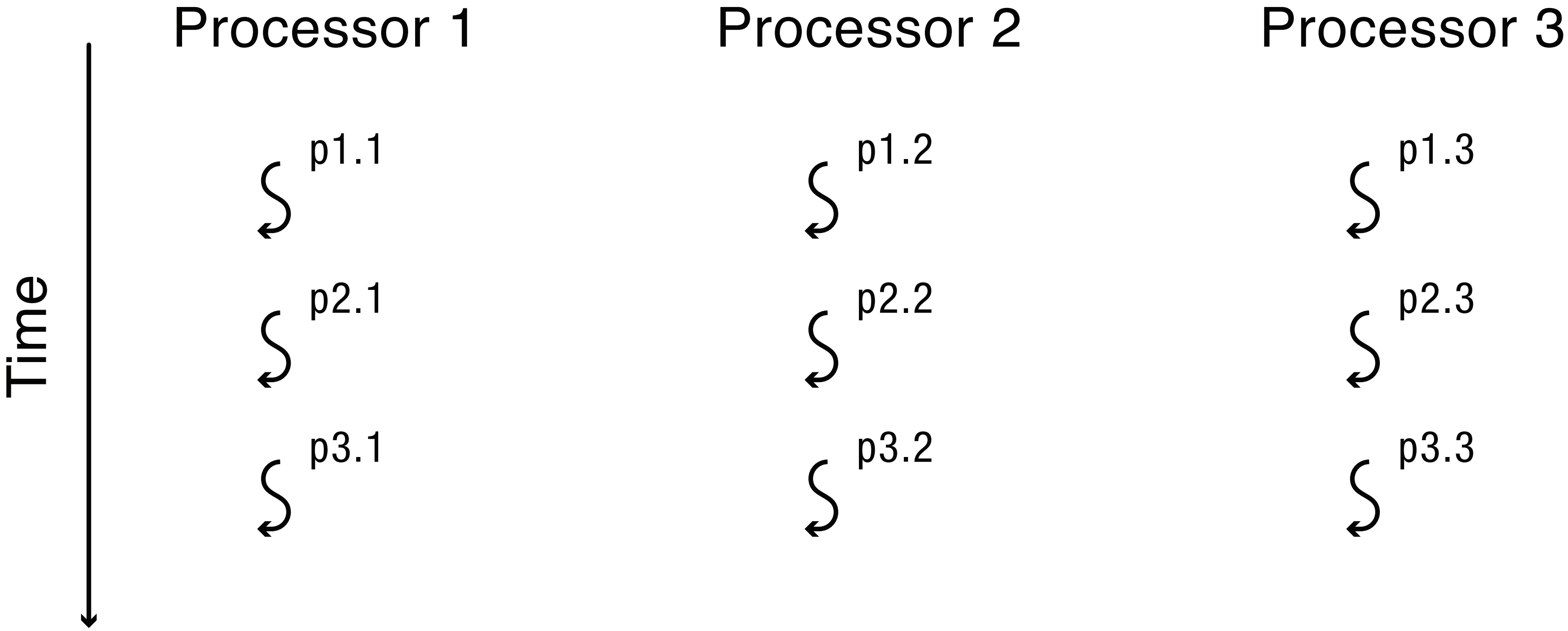


Time

Processor 1	Processor 2	Processor 3
p1.4	p2.1	p1.2
p2.3	p3.4	p1.3
p3.1	p2.4	p2.2

px.y = Thread y in process x

Gang Scheduling

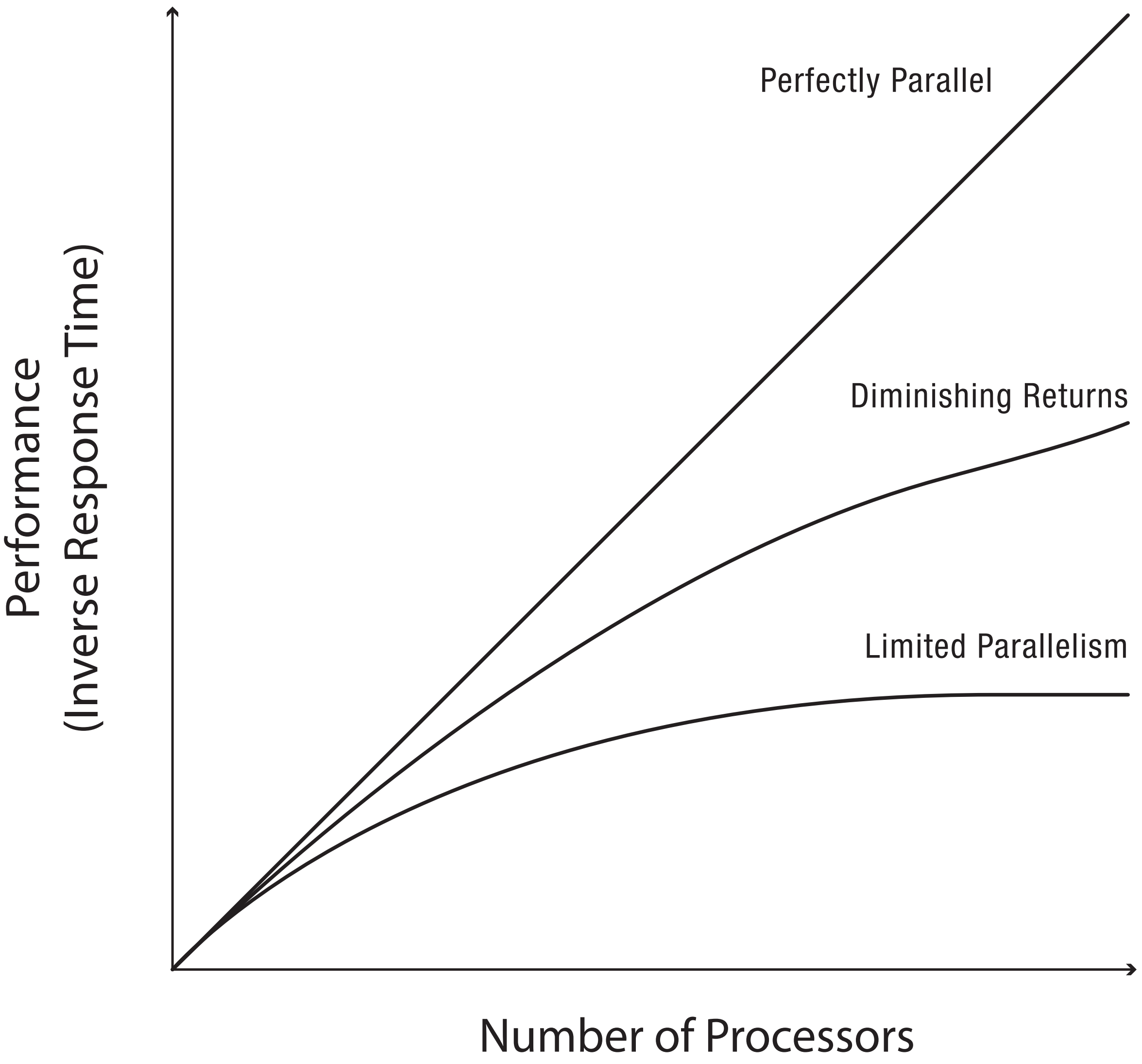


Time

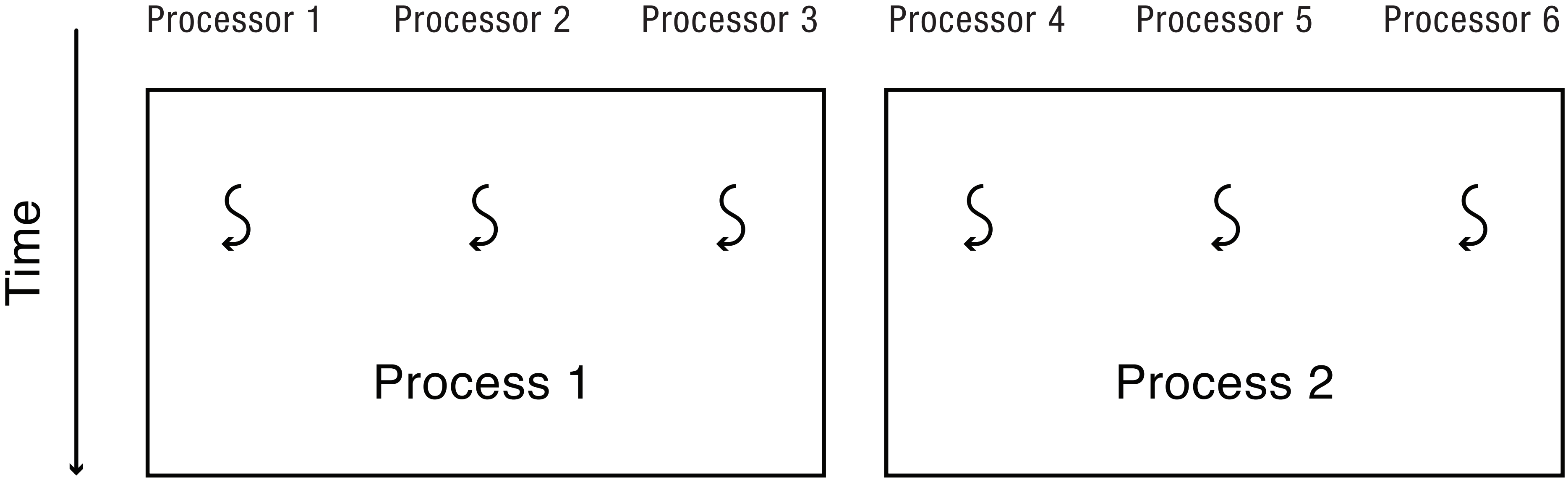
Processor 1	Processor 2	Processor 3
p1.1	p1.2	p1.3
p2.1	p2.2	p2.3
p3.1	p3.2	p3.3

px.y = Thread y in process x

Parallel Program Speedup



Space Sharing

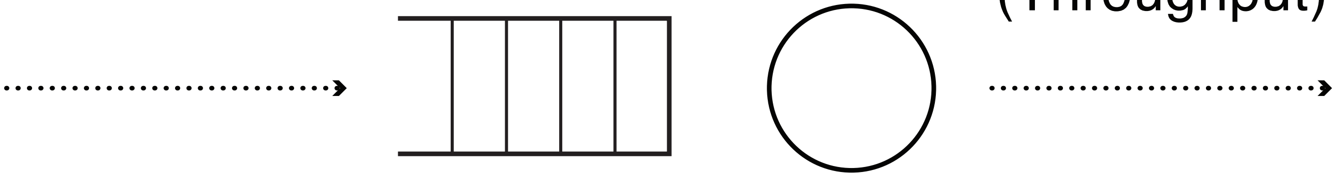


Scheduler activations: kernel tells each application its # of processors with upcalls every time the assignment changes

Queueing Theory

- Can we predict what will happen to user performance:
  - If a service becomes more popular?
  - If we buy more hardware?
  - If we change the implementation to provide more features?

Queueing Model



Arrivals

Departures
(Throughput)

Queue

Server

Assumption: average performance in a stable system, where the arrival rate (λ) matches the departure rate (μ)

Definitions

- Queueing delay (W): wait time
  - Number of tasks queued (Q)
- Service time (S): time to service the request
- Response time (R) = queueing delay + service time
- Utilization (U): fraction of time the server is busy
  - Service time * arrival rate (ƛ)
- Throughput (X): rate of task completions
  - If no overload, throughput = arrival rate

Little's Law

$$N = X * R$$

N: number of tasks in the system

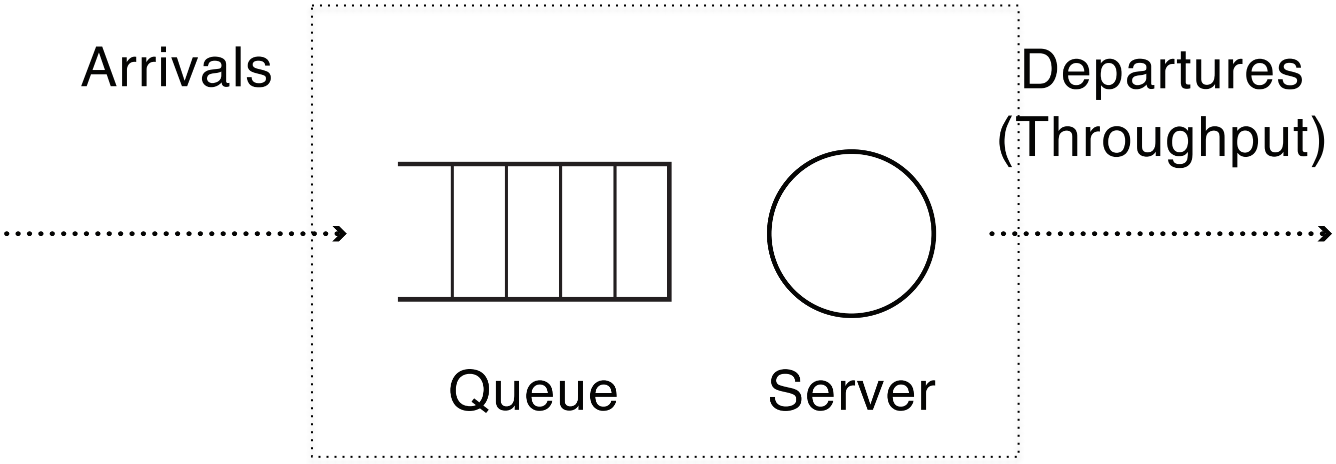
Applies to *any* stable system – where arrivals match departures.

- Independent of scheduling discipline and burstiness

Question

Suppose a system has throughput (X) = 100 tasks/s, average response time (R) = 50 ms/task

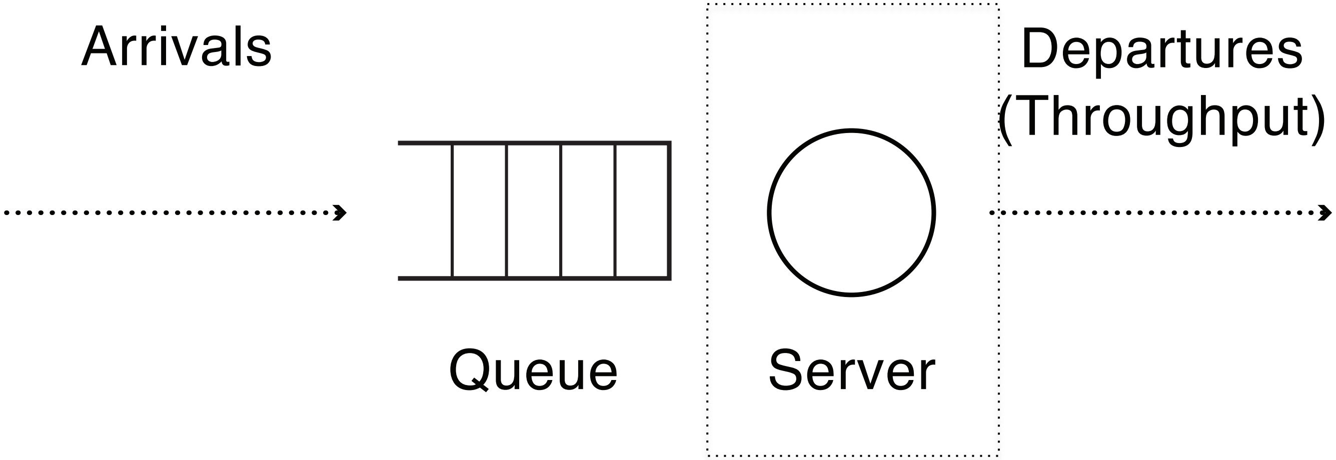
- How many tasks are in the system on average?
  - Hint: Little's Law N = X * R



Question

Suppose a system has throughput (X) = 100 tasks/s, average response time (R) = 50 ms/task

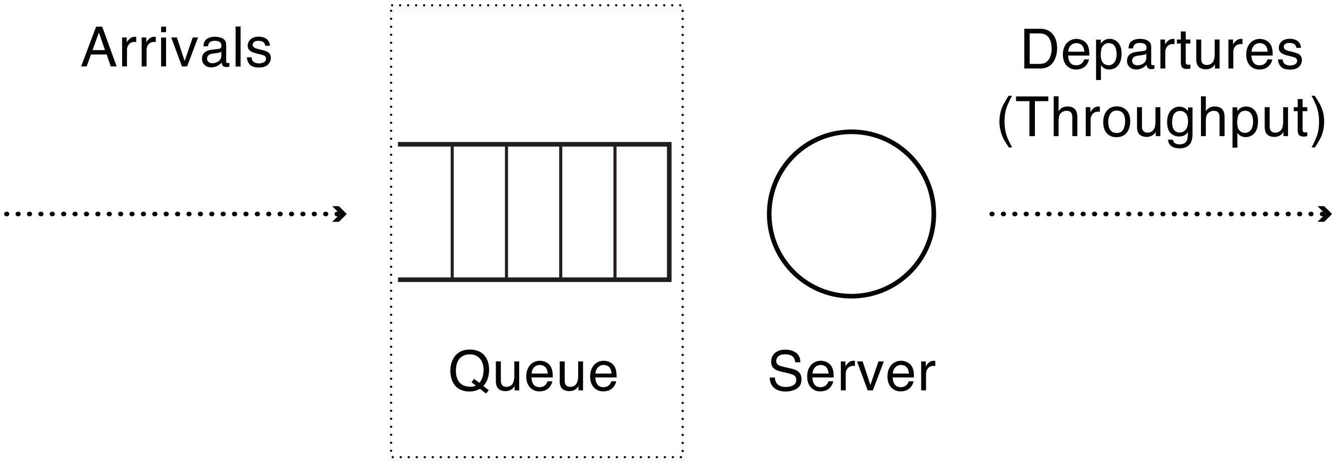
- If the server takes 5 ms/task, what is its utilization? (N = X * R)



Question

Suppose a system has throughput (X) = 100 tasks/s, average response time (R) = 50 ms/task

- What is the average wait time?
- What is the average number of queued tasks?



Question

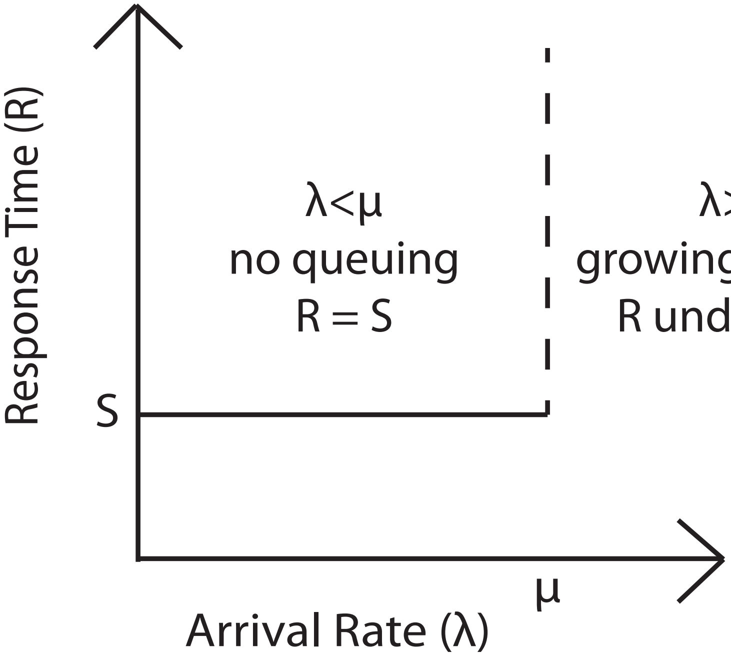
- From example:
  - $X = 100$ task/sec
  - $R = 50$ ms/task
  - $S = 5$ ms/task
  - $W = 45$ ms/task
  - $Q = 4.5$ tasks
- What gives? $W = 45$ ms while $S * Q = 22.5$ ms
  - Hint: what if $S = 10$ms? $S = 1$ms?

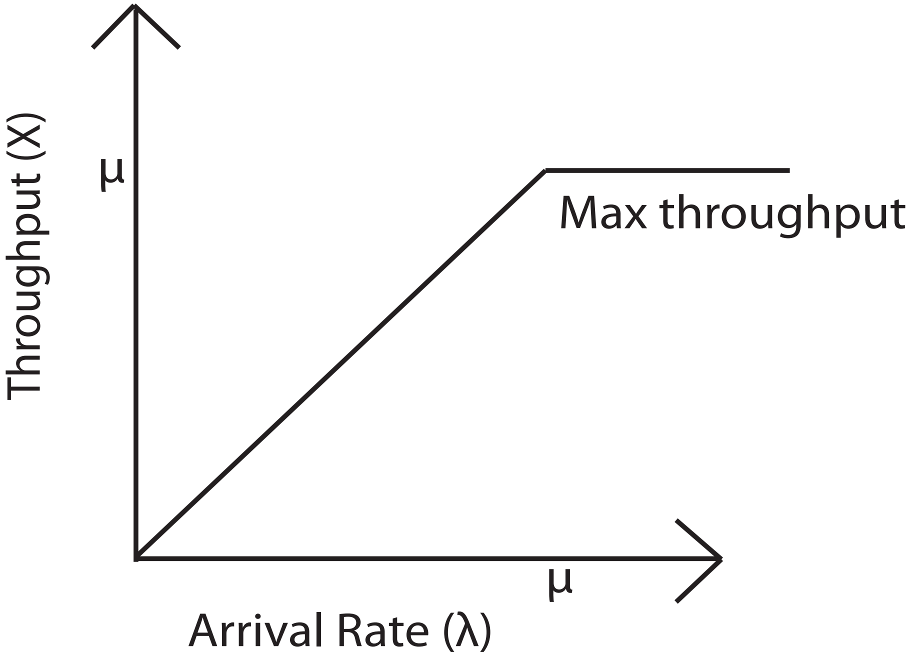
Queueing

- What is the best case scenario for minimizing queueing delay?
  - Keeping arrival rate, service time constant

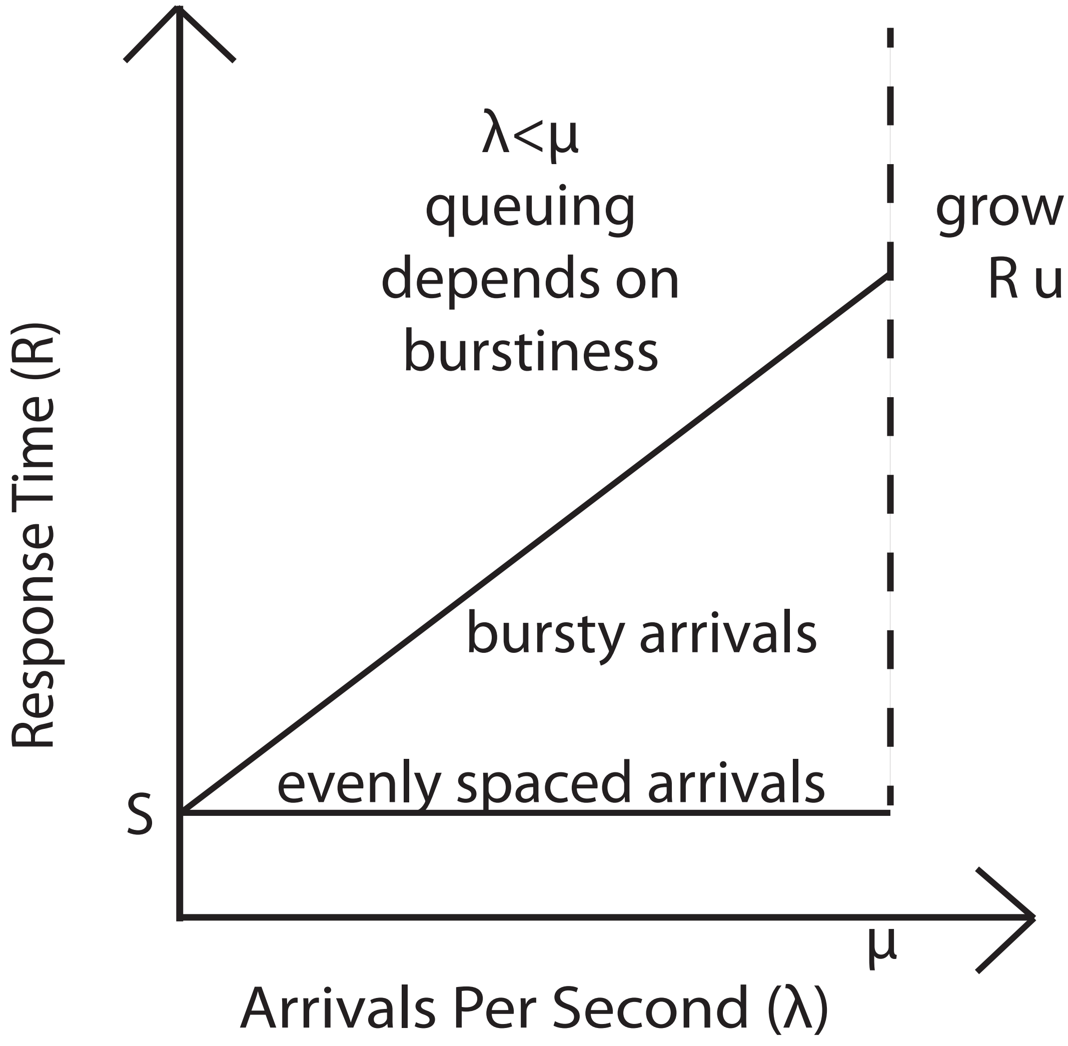
- What is the worst case scenario?

Queueing: Best Case





Response Time: Best vs. Worst Case

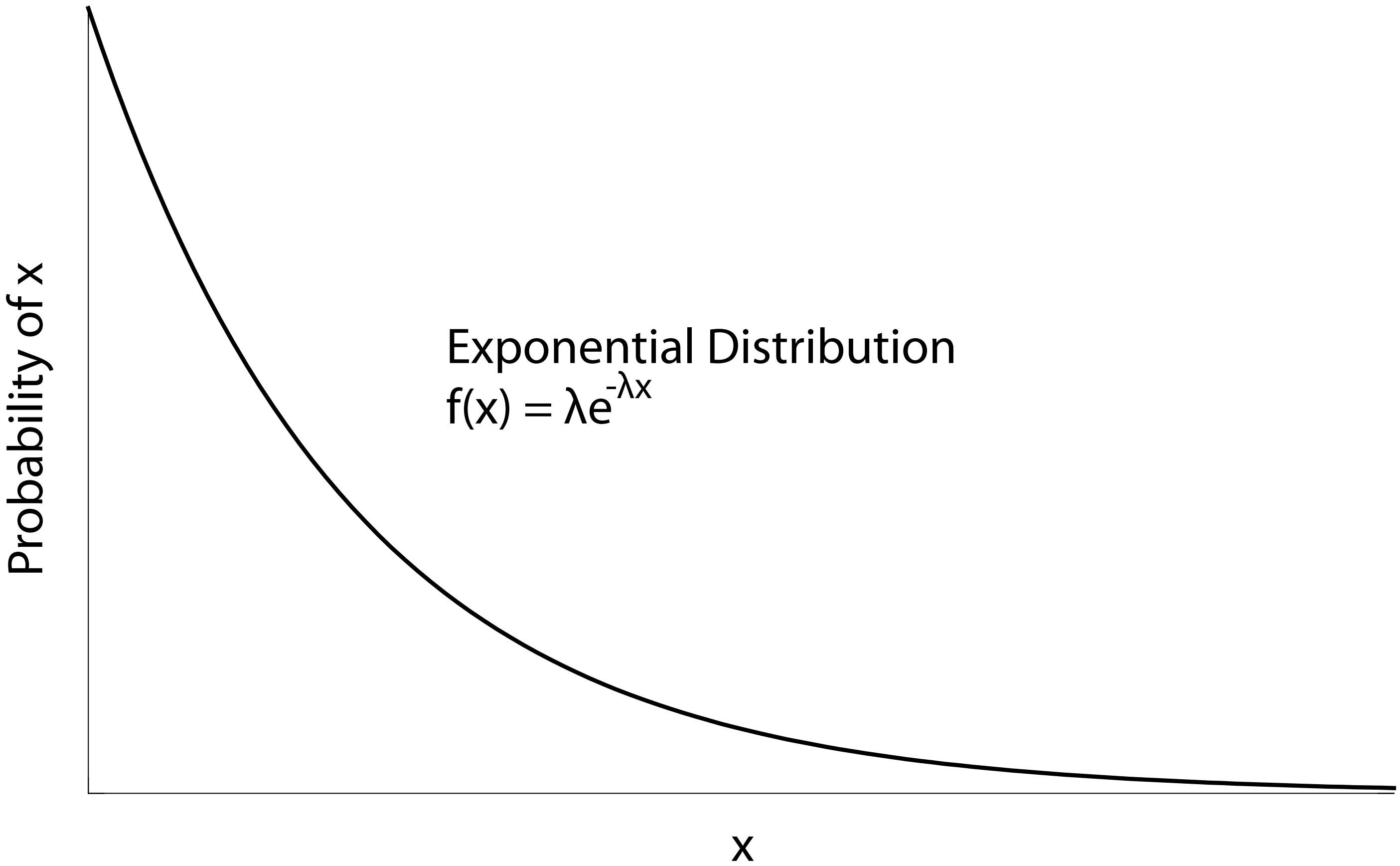


Queueing: Average Case?

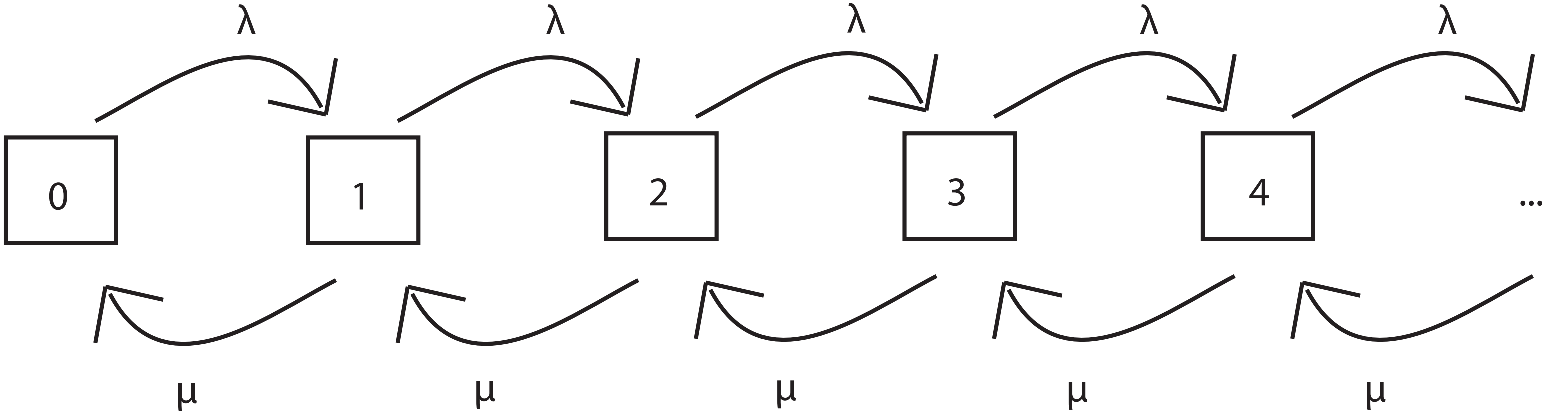
- What is average?
  - Gaussian: Arrivals are spread out, around a mean value
  - Exponential: arrivals are memoryless
  - Heavy-tailed: arrivals are bursty

- Can have randomness in both arrivals and service times

Exponential Distribution

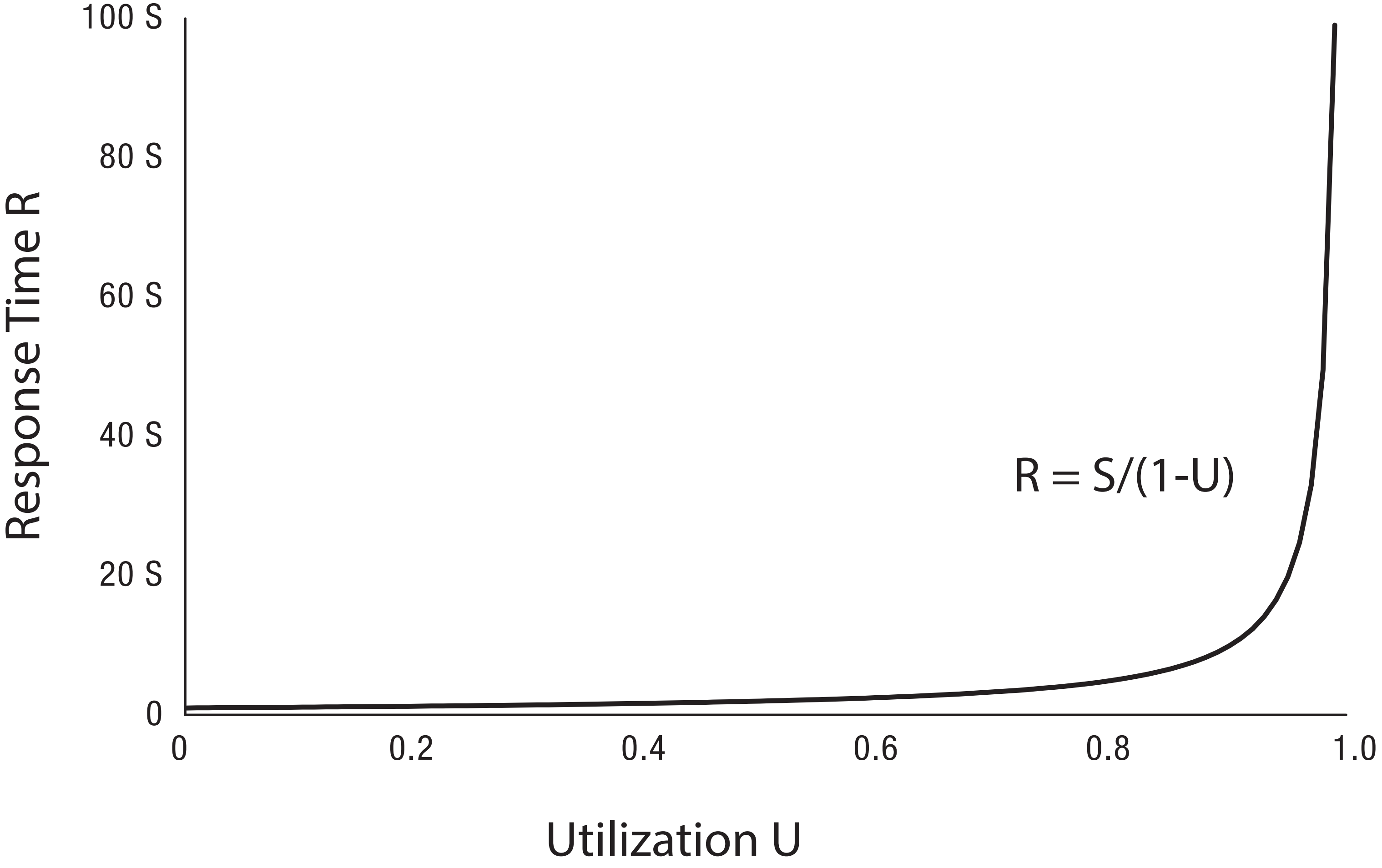


Exponential Distribution



Permits closed form solution to state probabilities,
as function of arrival rate and service rate

Response Time vs. Utilization



Question

- Exponential arrivals: R = S/(1-U)
- If system is 20% utilized, and load increases by 5%, how much does response time increase?

- If system is 90% utilized, and load increases by 5%, how much does response time increase?

Variance in Response Time

- Exponential arrivals
  - Variance in R = S/(1-U)^2

- What if less bursty than exponential?

- What if more bursty than exponential?

What if Multiple Resources?

- Assuming exponential arrival, service times
- Response time =

  Sum over all i

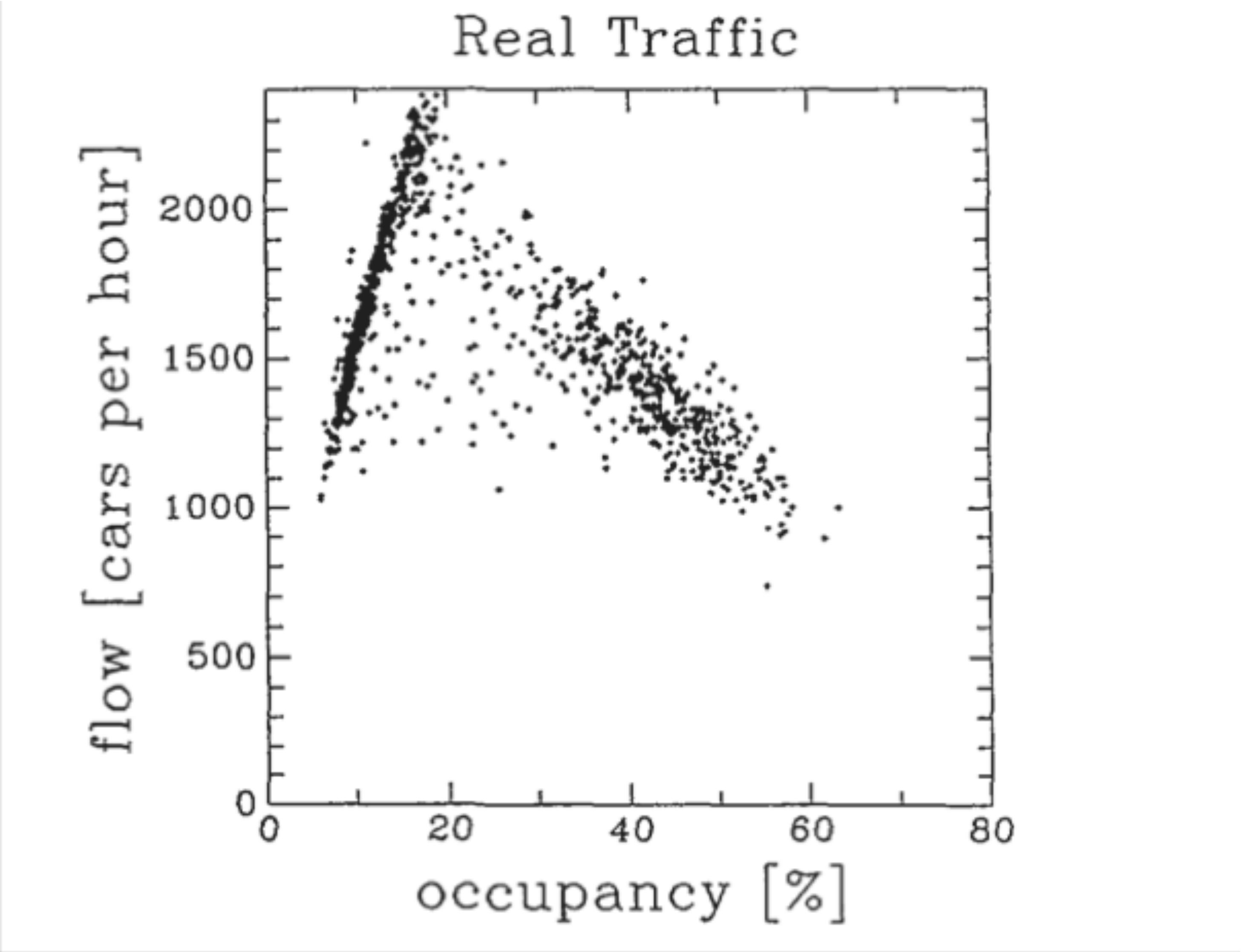
    Service time for resource i /

      (1 – Utilization of resource i)
- Implication
  - If you fix one bottleneck, the next highest utilized resource will limit performance

Overload Management

- What if arrivals occur faster than service can handle them
  - If do nothing, response time will become infinite
- Turn users away?
  - Which ones? Average response time is best if turn away users that have the highest service demand
  - Example: Highway congestion
- Degrade service?
  - Compute result with fewer resources
  - Example: CNN static front page on 9/11

Highway Congestion (measured)



Why Do Metro Buses Cluster?

Suppose two Metro buses start 10 minutes apart.

Why might they arrive at the same time?

Control Theory

- Regulate tasks entering system to meet SLA
  - Or to manage chance of queue overflow
  - Or to optimize for some system objective
- May be complex system
  - May or may not be modelled by queueing theory

Black Box Control Theory

- Assume no internal visibility
  - See input arrivals and task completions
- Regulate at time scale of task response time
  - If too rapid, oscillate
  - If too slow, slow convergence
- Rate(k+1) = a*Rate(k) – b*N(k)