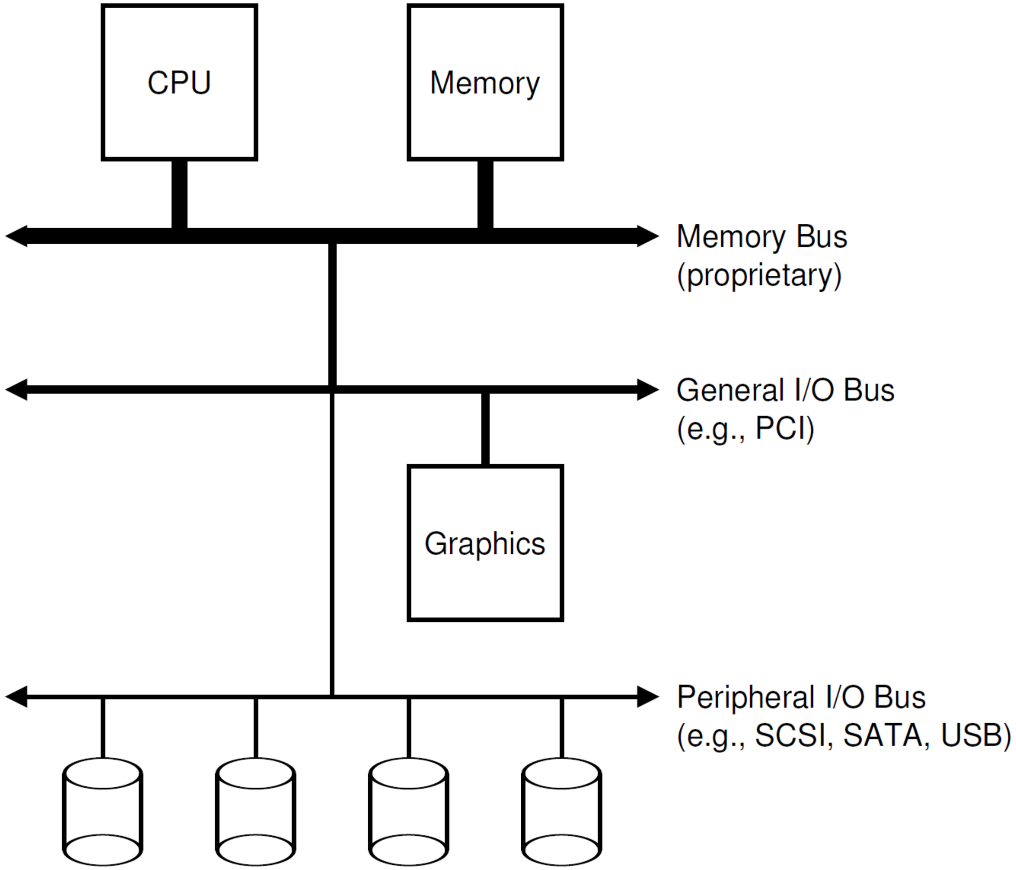
CS162
Operating Systems and Systems Programming
Lecture 18

Storage Devices, Performance, Queuing Theory

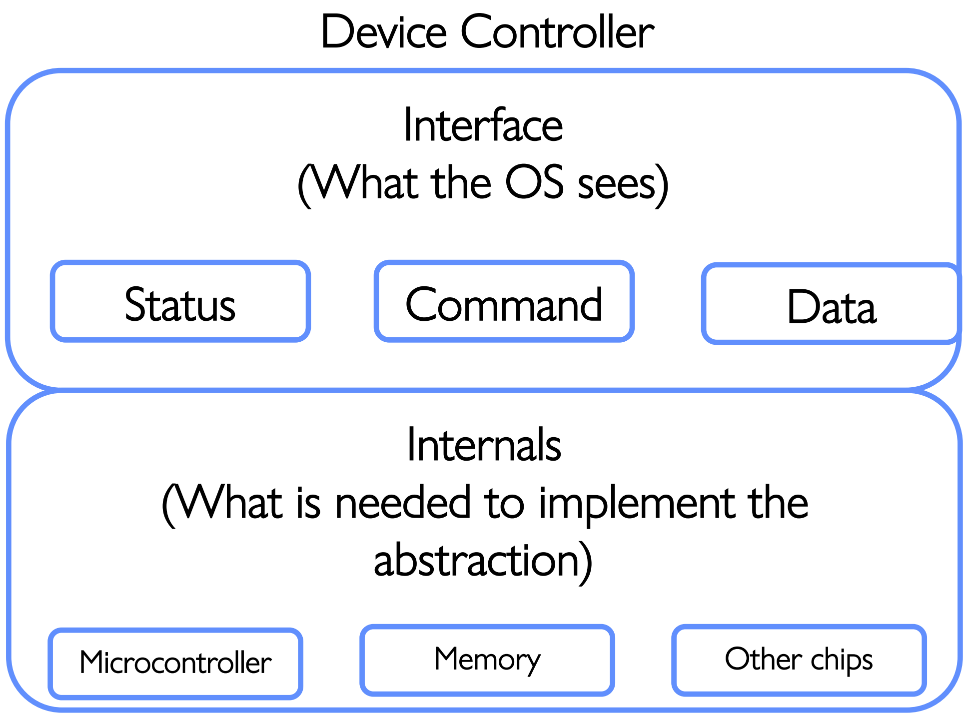
Recall : Simplified IO architecture



Follows a hierarchical structure because of cost: the faster the bus, the more expensive it is

Recall: How does the processor talk to devices?

- Remember, it's all about abstractions!



Hardware interface device presents to OS

Hardware interface device presents to OS

Recall: Device Drivers

- Device Driver: Device-specific code in the kernel that interacts directly with the device hardware
  - Supports a standard, internal interface
  - Special device-specific configuration supported with the **`ioctl()`** system call

- Device Drivers typically divided into two pieces:
  - Top half: accessed in call path from system calls
    - implements a set of standard, cross-device calls like **`open()`**, **`close()`**, **`read()`**, **`write()`**, **`ioctl()`**, **`strategy()`**
    - This is the kernel's interface to the device driver
    - Top half will *start* I/O to device, may put thread to sleep until finished
  - Bottom half: run as interrupt routine
    - Gets input or transfers next block of output
    - May wake sleeping threads if I/O now complete

- Your body is 90% water, the OS is 70% device-drivers

Recall: Life Cycle of An I/O Request

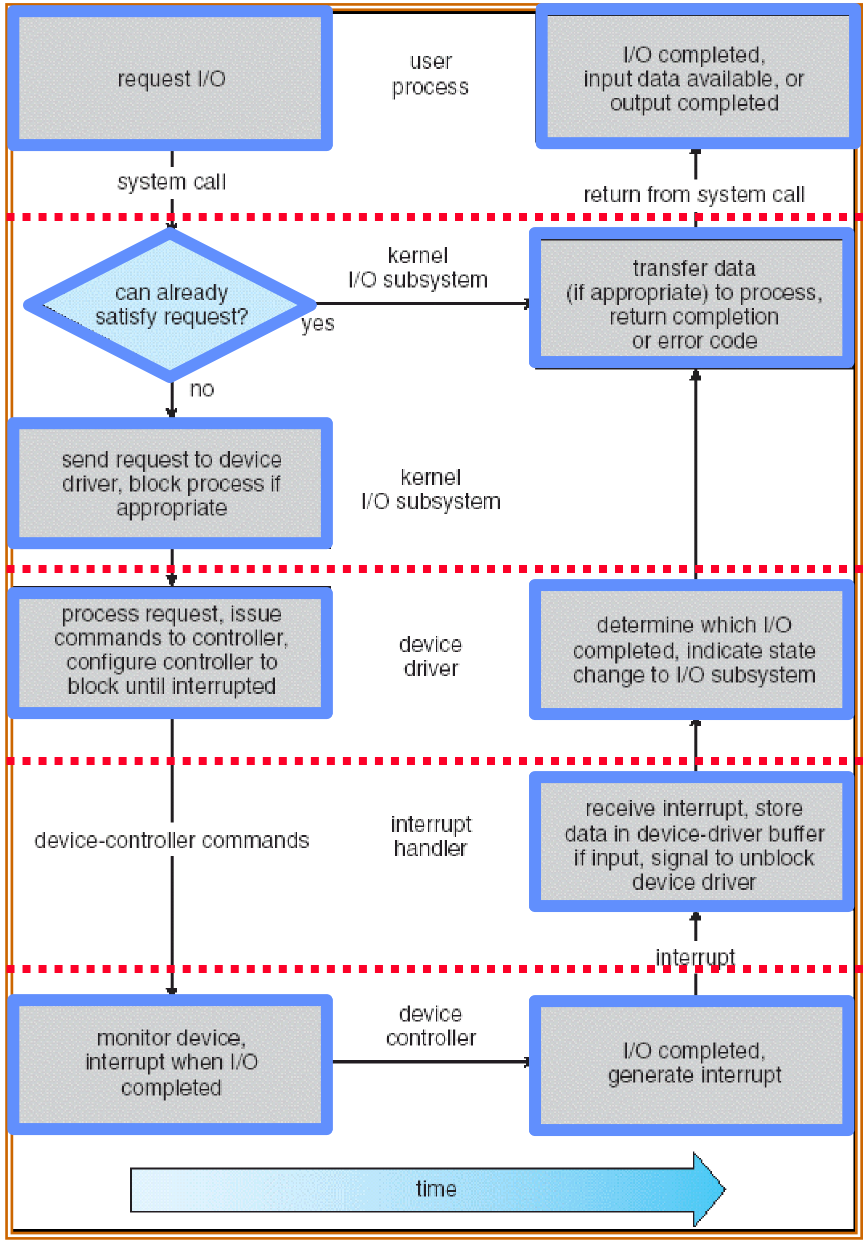
User Program

Kernel I/O Subsystem

Device Driver Top Half

Device Driver Bottom Half

Device Hardware



Ways of Measuring Performance: Times (s) and Rates (op/s)

- *Response Time or Latency* - time to complete a task
  – Measured in units of time (s, ms, us, …, hours, years)

- *Throughput* or *Bandwidth* – rate at which tasks are performed
  – Measured in units of things per unit time (ops/s, GFLOP/s)

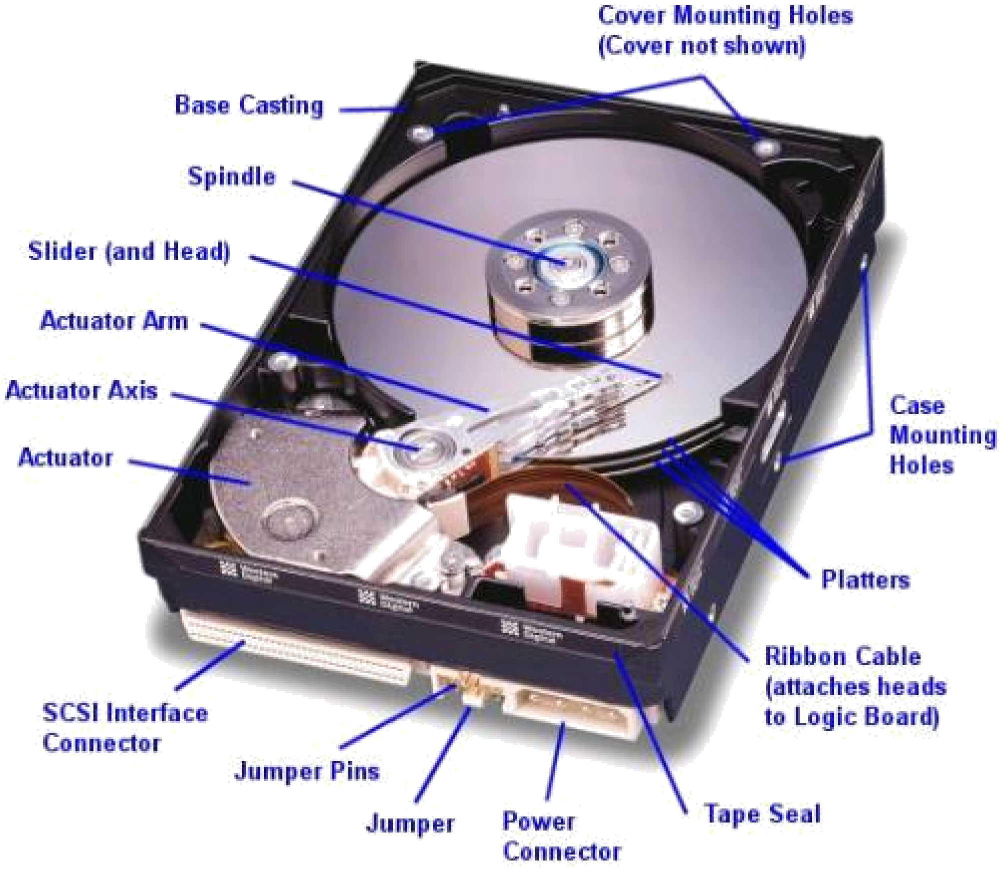
- *Start up or Overhead* – time to initiate an operation

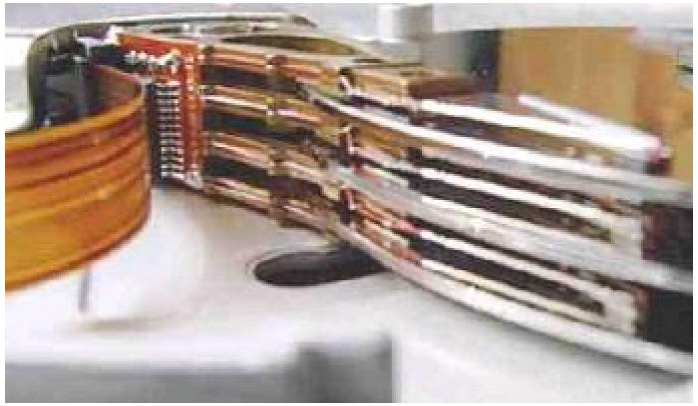
- Most I/O operations are roughly linear in *b* bytes
  – Latency(b) = Overhead + b/TransferCapacity

Storage Devices

- Magnetic disks
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access (except for SMR – later!)
  - Slow performance for random access
  - Better performance for sequential access
- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (5-20x disk)
  - Block level random access
  - Good performance for reads; worse for random writes
  - Wear patterns issue

Hard Disk Drives (HDDs)





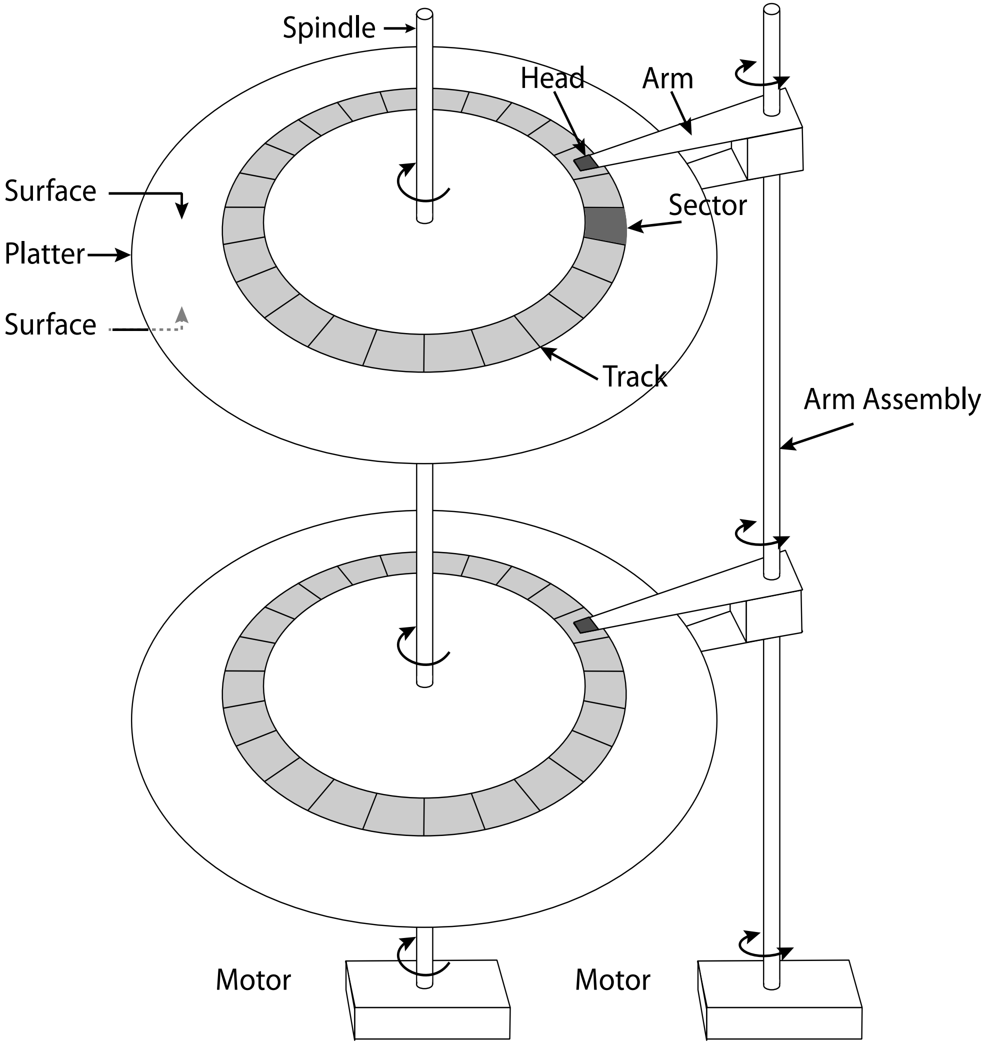
Read/Write Head Side View

IBM Personal Computer 1986

30MB Hard Disk for 500 dollars

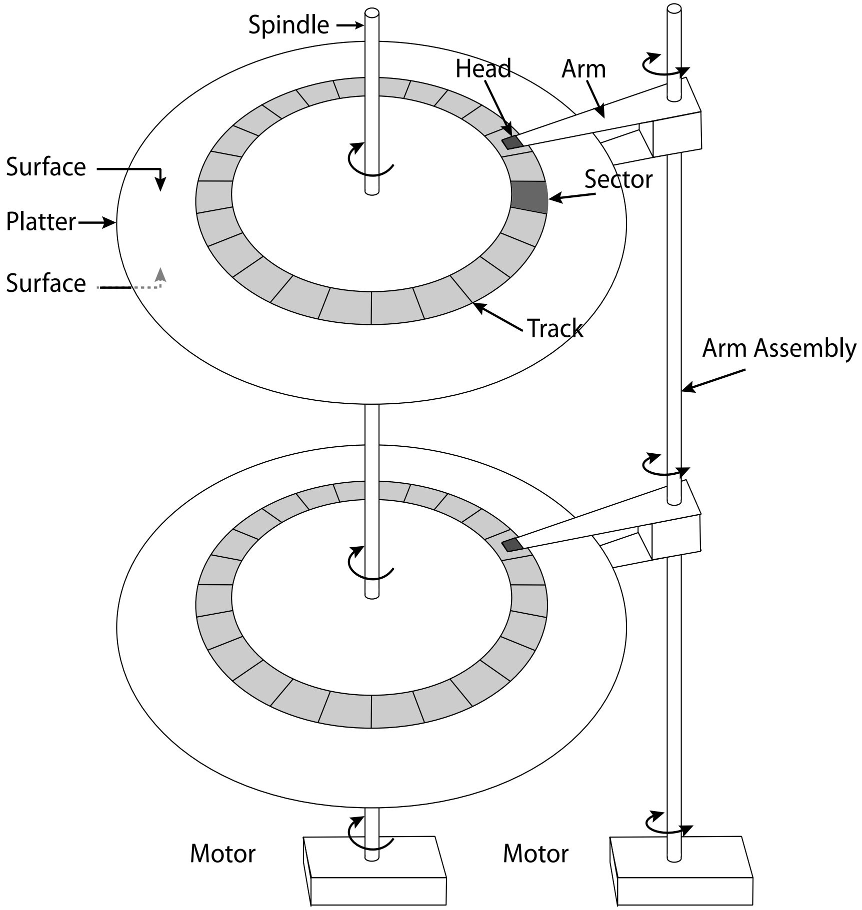
The Amazing Magnetic Disk

- Store data magnetically on thin metallic film bonded to rotating disk of glass, ceramic, or aluminum



The Amazing Magnetic Disk

Store data magnetically on thin metallic film bonded to rotating disk of glass, ceramic, or aluminum



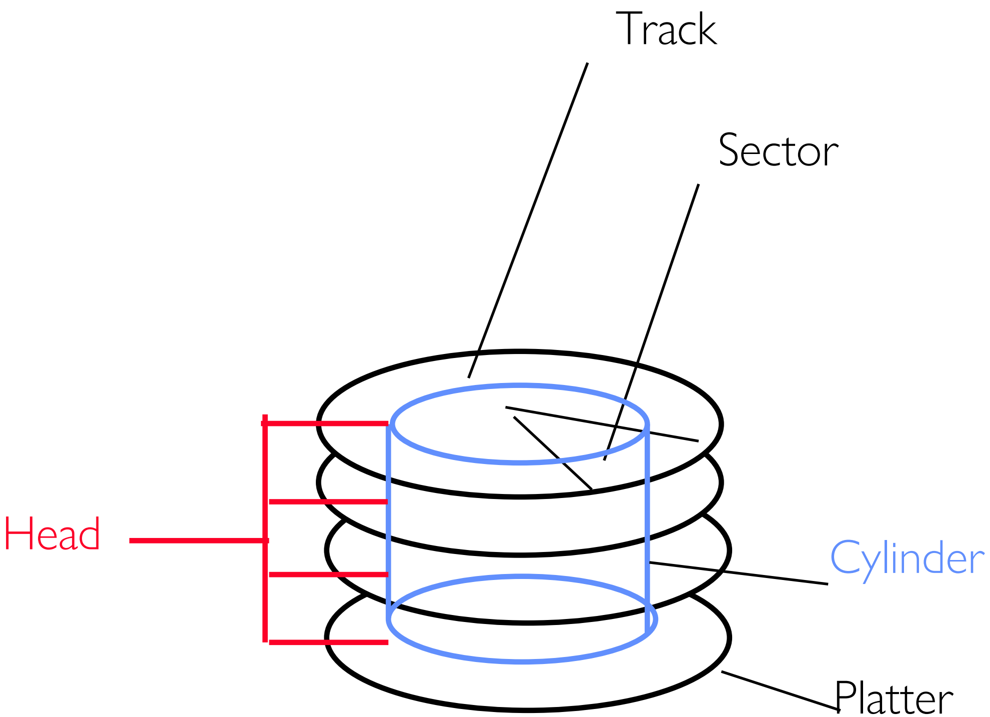
Track: concentric circle on surface

Sectors: slice of a track

- Smallest addressable unit
- Are units of transfers

Cylinder all the tracks under the head at a given point on all surfaces

The Amazing Magnetic Disk



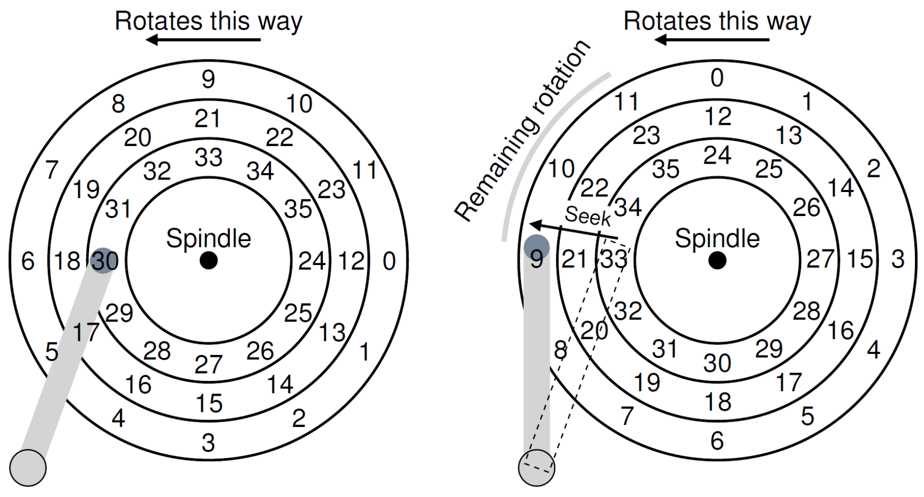
Track lengths vary across disk: outside tracks have more sectors per track, higher bandwidth

Disk is organized into regions of tracks with the same number of sector/tracks

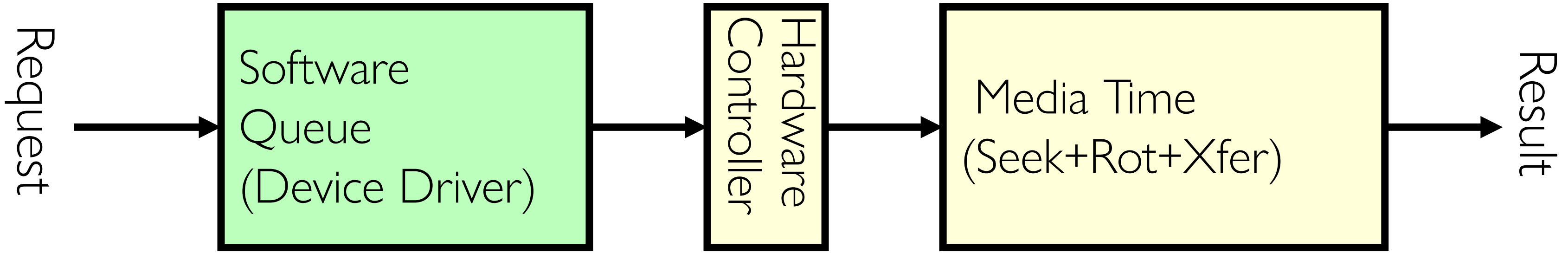
Usually, only outer half of radius is used

The Amazing Magnetic Disk

- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track
  - Rotational latency: wait for desired sector to rotate under r/w head
  - Transfer time: transfer a block of bits (sector) under r/w head



Request Time = Queueing Time + Controller Time + Seek + Rotational + Transfer



Typical Numbers for Magnetic Disk

Parameter	Info/Range
Space/Density	Space: 14TB (Seagate), 8 platters, in 3½ inch form factor! Areal Density: ≥ 1 Terabit/square inch! (PMR, Helium, …)
Average Seek Time	Typically 4-6 milliseconds
Average Rotational Latency	Most laptop/desktop disks rotate at 3600-7200 RPM (16-8 ms/rotation). Server disks up to 15,000 RPM. Average latency is halfway around disk so 4-8 milliseconds
Controller Time	Depends on controller hardware
Transfer Time	Typically 50 to 250 MB/s. Depends on: • Transfer size (usually a sector): 512B – 1KB per sector • Rotation speed: 3600 RPM to 15000 RPM • Recording density: bits per inch on a track • Diameter: ranges from 1 in to 5.25 in
Cost	Used to drop by a factor of two every 1.5 years (or faster), now slowing down

Disk Performance Example

- Key to using disk effectively (especially for file systems) is to minimize seek and rotational delays
- Assumptions:
  – Ignoring queuing and controller times for now
  – Avg seek time of 5ms,
  – 7200RPM ⇒ Time for rotation: 60000 (ms/min) / 7200(rev/min) ~= 8ms
  – Transfer rate of 50MByte/s, block size of 4Kbyte ⇒ 4096 bytes/50×10^6 (bytes/s) = 81.92×10^{-6} sec ≅ 0.082 ms for 1 sector

Disk Performance Example

- Read block from random place on disk (random reads):
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.082ms) = 9.082ms
  - Approx 9ms to fetch/put data: 4096 bytes/9.082×10^{-3} s ≅ 451KB/s

- Read block from random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.082ms) = 4.082ms
  - Approx 4ms to fetch/put data: 4096 bytes/4.082×10^{-3} s ≅ 1.03MB/s

- Read next block on same track (sequential reads):
  - Transfer (0.082ms): 4096 bytes/0.082×10^{-3} s ≅ 50MB/sec

Lots of Intelligence in the Controller

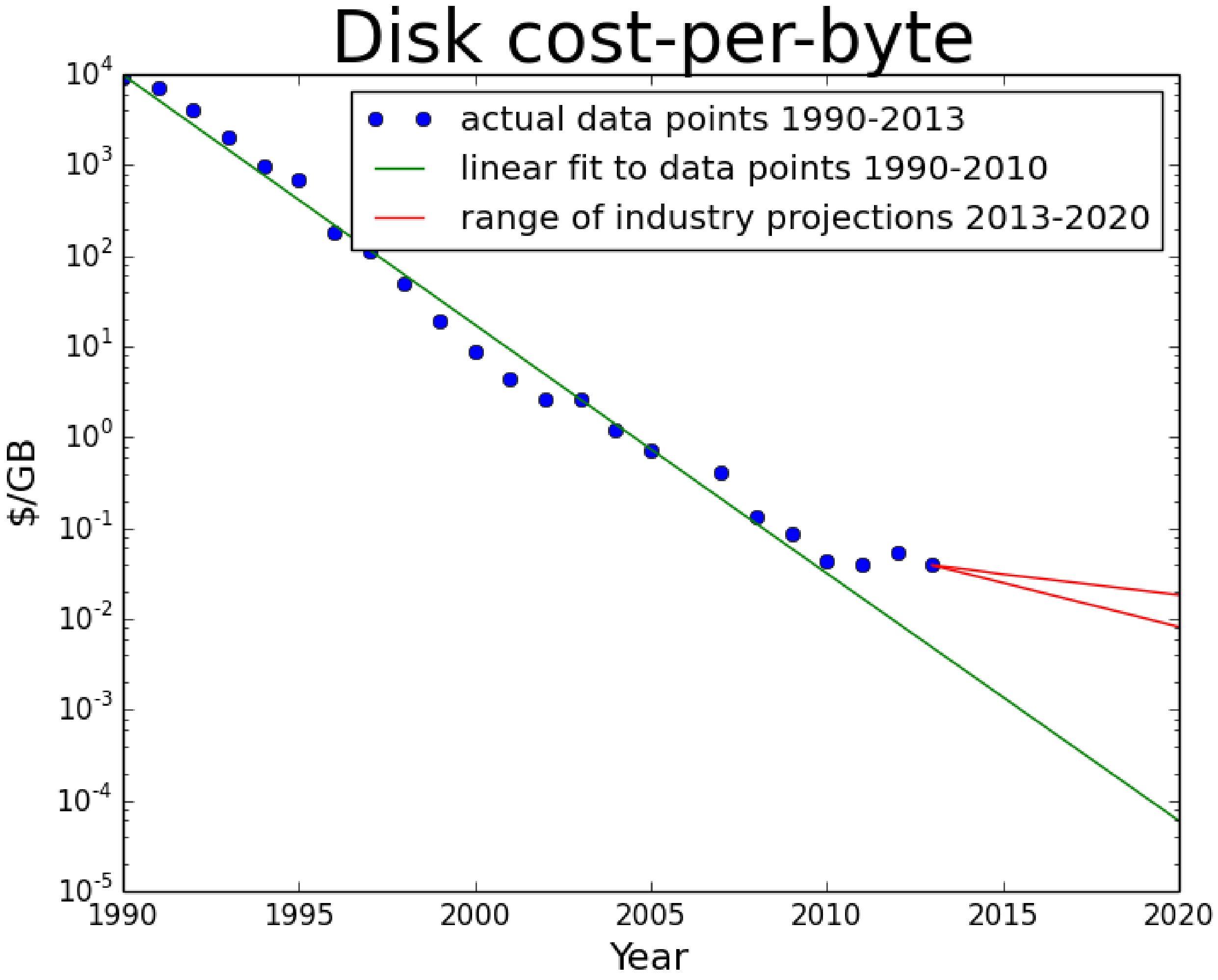
- Sectors contain sophisticated error correcting codes
  – Disk head magnet has a field wider than track
  – Hide corruptions due to neighboring track writes

- Sector sparing
  – Remap bad sectors transparently to spare sectors on the same surface

- Slip sparing
  – Remap all sectors (when there is a bad sector) to preserve sequential behavior

- Track skewing
  – Sector numbers offset from one track to the next, to allow for disk head movement for sequential ops

Hard Drive Prices over Time



Example of Current HDDs

- Seagate Exos X18 (2020)
  - 18 TB hard disk
    - » 9 platters, 18 heads
    - » Helium filled: reduce friction and power
  - 4.16ms average seek time
  - 4096 byte physical sectors
  - 7200 RPMs
  - Dual 6 Gbps SATA /12Gbps SAS interface
    - » 270MB/s MAX transfer rate
    - » Cache size: 256MB
  - Price: $ 562 (~ $0.03/GB)

- IBM Personal Computer/AT (1986)
  - 30 MB hard disk
  - 30-40ms seek time
  - 0.7-1 MB/s (est.)
  - Price: $500 ($17K/GB, 340,000x more expensive !!)

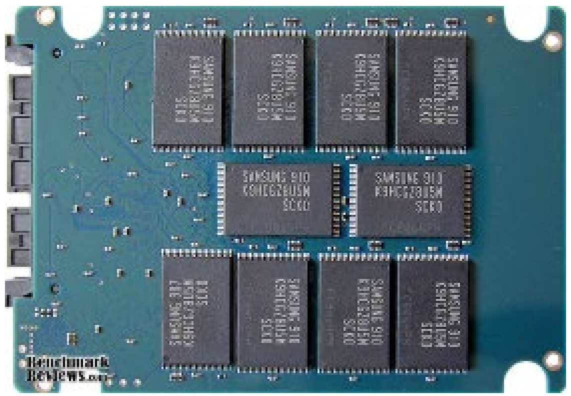
Solid State Drives

- 1995 – Replace rotating magnetic media with non-volatile memory (battery backed DRAM)

- 2009 – Use flash memory
  - Sector (4 KB page) addressable, but stores 4-64 “pages” per memory block
  - Trapped electrons distinguish between 1 and 0

- No moving parts (no rotate/seek motors)
  - Eliminates seek and rotational delay (0.1-0.2ms access time)
  - Very low power and lightweight
  - Limited “write cycles”

- Rapid advances in capacity and cost ever since!



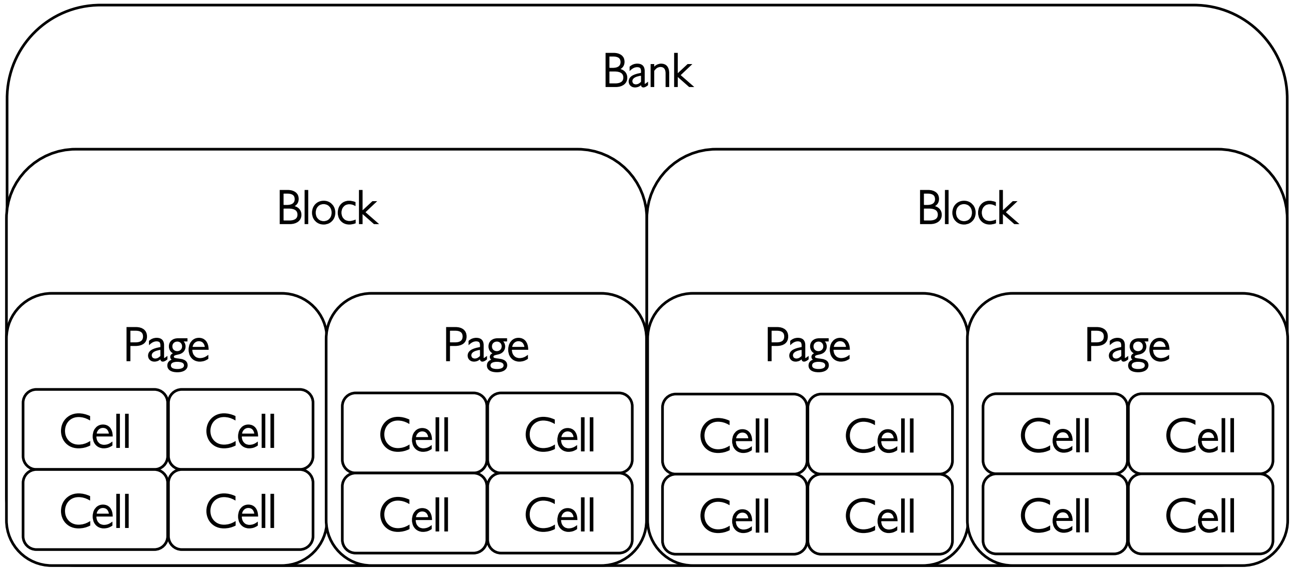
The Flash Cell

- Encode bit by trapping electrons into a cell

- Single-level cell (SLC)
  - Single bit is stored within a transistor
  - Faster, more lasting (50k to 100k writes before wear out)

- Multi-level cell (MLC)
  - Two/three bits are encoded into different levels of charge
  - Wear out much faster (1k to 10k writes)

Of banks, blocks, cells

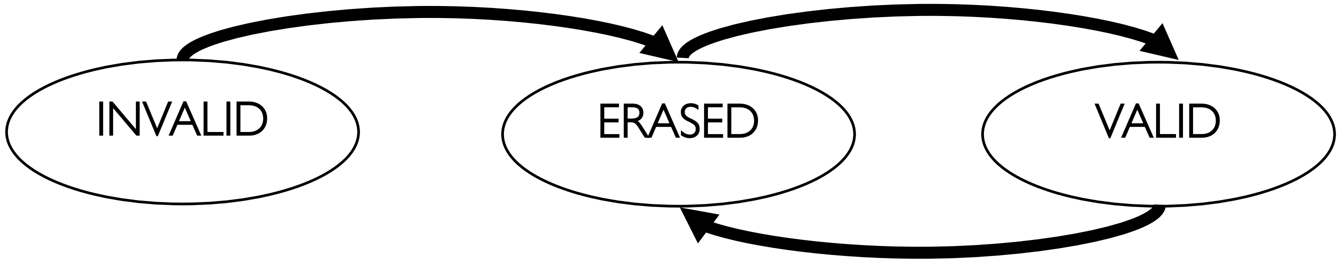


- Flash chips organized in **banks**
  - Banks can be accessed in parallel
- **Blocks**: 128 KB/256KB
  - (64 to 258 pages)
- **Pages**: Few KB
- **Cells:** 1 to 4 bits
- Distinction between blocks and pages important in operations!

Low-level flash operations

- How do you read?
  - Chip supports reading *pages*
  - 10s of microseconds, independently of the previous read page
- What about writing? More complicated!
  - Must first *erase the block*
    - » Erase quite expensive (milliseconds)
  - Once block has been erased, can then *program a page*
    - » Change 1s to 0s within a page.
    - » 100s of microseconds.
  - Blocks can only be erased a limited number of times!

Low-level flash operations



		`iiii`	*Initial: pages in block are invalid* (`i`)
Erase()	→	`EEEE`	*State of pages in block set to erased* (`E`)
Program(0)	→	`VEEE`	*Program page 0; state set to valid* (`V`)
Program(0)	→	**error**	*Cannot re-program page after programming*
Program(1)	→	`VVEE`	*Program page 1*
Erase()	→	`EEEE`	*Contents erased; all pages programmable*

Low-level flash operations

- Assume block of 4 pages. All valid. Want to write Page 0

Page 0	Page 1	Page 2	Page 3
00011000	11001110	00000001	00111111
VALID	VALID	VALID	VALID

Step 1: erase **full** block

Page 0	Page 1	Page 2	Page 3
11111111	11111111	11111111	11111111
ERASED	ERASED	ERASED	ERASED

Step 2: program page 0

Page 0	Page 1	Page 2	Page 3
00000011	11111111	11111111	11111111
VALID	ERASED	ERASED	ERASED

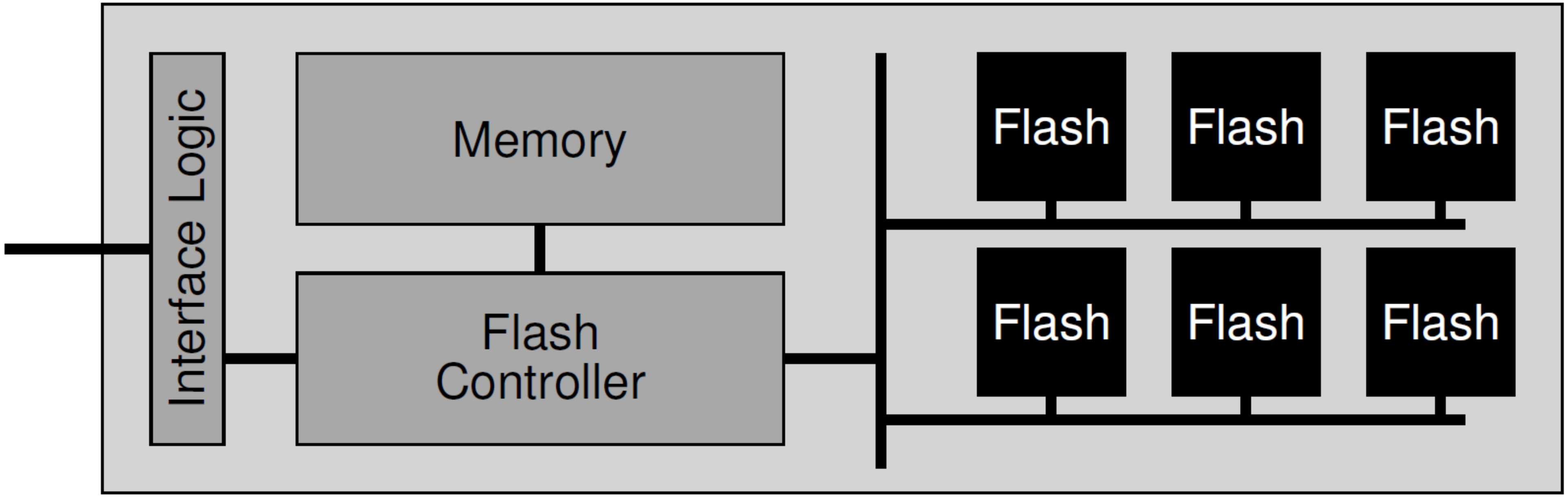
SSD Architecture

- Recall that SSDs uses low-level Flash operations to provide same interface as HDD
  - read and write chunk (4KB) at a time

- Reads are easy, but for writes, can only overwrite data one block (256KB) at a time!

- Why not just erase and rewrite new version of entire 256KB block?
  - Erasure is very slow (milliseconds)
  - Each block has a finite lifetime, can only be erased and rewritten about 10K times
  - Heavily used blocks likely to wear out quickly

SSD Architecture (Simplified)



Flash Translation Layer (FTL)

- Add a layer of indirection: the flash translation layer
  - Translates request for logical blocks (device interface) to low-level Flash blocks and pages
- Goal: performance and reliability
- Reduce **write amplification**
  - Ratio of the total write traffic in bytes issues by the flash chip by the FTL devided by the total write traffic issued by the OS to the device
- Avoid **wear out**
  - A single block should not be erased too often

FTL – Two Systems Principles

- FTL uses *indirection* and *copy-on-write*

- Maintains mapping tables in DRAM
  - Map virtual block numbers (which OS uses) to physical page numbers (which flash mem. controller uses)
  - Can now freely relocate data w/o OS knowing

- Copy on Write/ Log-structured FTL
  - Don't overwrite a page when OS updates its data
  - Instead, write new version in a free page
  - Update FTL mapping to point to new location

FTL – Two Systems Principles

- FTL uses *indirection* and *copy-on-write*

- Maintains mapping tables in DRAM
  - Map virtual block numbers (which OS uses) to physical page numbers (which flash mem. controller uses)
  - Can now freely relocate data w/o OS knowing

- Copy on Write/ Log-structured FTL
  - Don't overwrite a page when OS updates its data
  - Instead, write new version in a free page
  - Update FTL mapping to point to new location

FTL Example

Initial State

Mapping Table:			
Block 0		Block 1	
E	E	E	E

Write(a0)

Mapping Table a0->0,0			
Block 0		Block 1	
a0			
V	E	E	E

Write(a1)

Mapping Table: a0->0,0/a1->0,1			
Block 0		Block 1	
a0	a1		
V	V	E	E

Write(a1)

Mapping Table: a0->0,0/a1->1,0			
Block 0		Block 1	
a0	a1	a1	
V	V	V	E

Write(a1)

Mapping Table: a0->1,1/a1->1,0			
Block 0		Block 1	
a0	a1	a1	a0
V	V	V	V

Garbage Collect

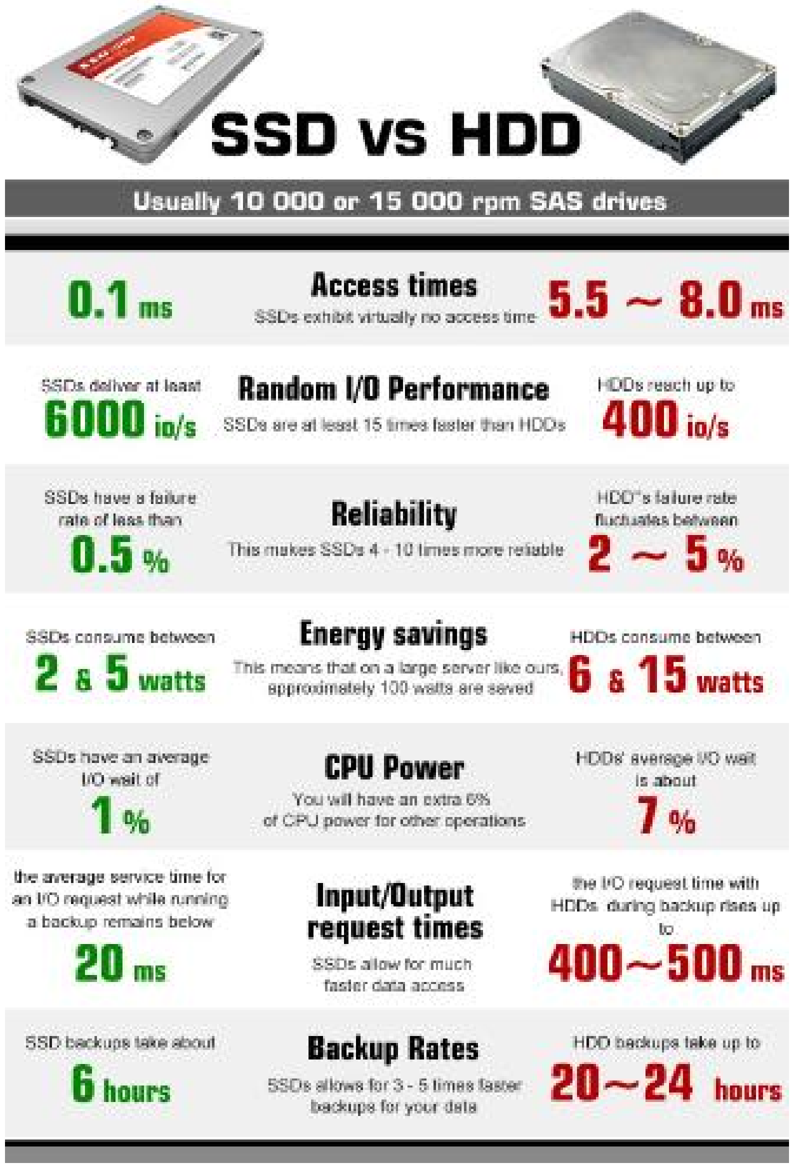
Mapping Table: a0->1,1/a1->1,0			
Block 0		Block 1	
		a1	a0
E	E	V	V

Some “Current” (large) 3.5in SSDs

- Seagate Exos SSD: 15.36TB (2017)
  - Seq reads 860MB/s
  - Seq writes 920MB/s
  - Price (Amazon): $5495 ($0.36/GB)

- Nimbus SSD: 100TB (2019)
  - Seq reads/writes: 500MB/s
  - Random Read Ops (IOPS): 100K
  - *Unlimited writes for 5 years!*
  - Price: ~ $40K? ($0.4/GB)
    - » However, 50TB drive costs $12500 ($0.25/GB)

HDD vs. SSD Comparison



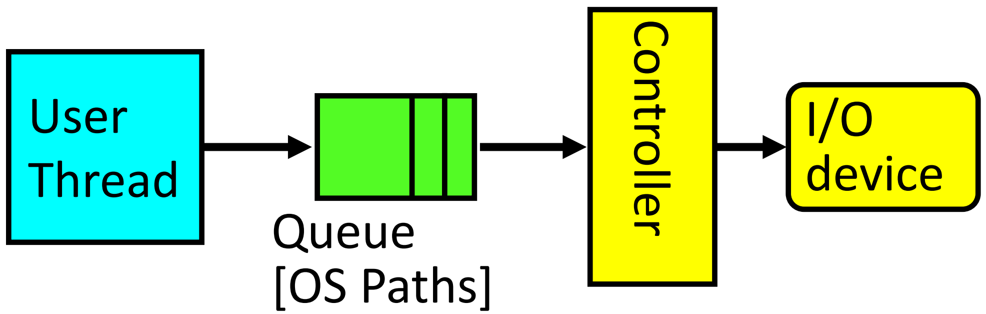
HDD	SDD
Require seek + rotation	No seeks
Not parallel (one head)	Parallel
Brittle (moving parts)	No moving parts
Random reads take 10s milliseconds	Random reads take 10s microseconds
Slow (Mechanical)	Wears out
Cheap/large storage	Expensive/smaller storage

Recall: Overall Performance for I/O Path

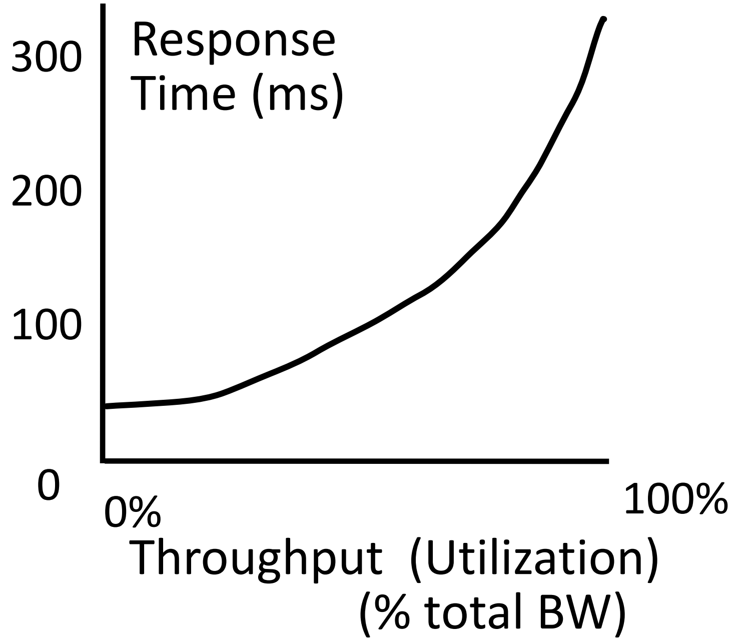
- Performance of I/O subsystem
  – Metrics: Response Time, Throughput

- Contributing factors to latency:
  » Software paths (can be loosely modeled by a queue)
  » Hardware controller
  » I/O device service time

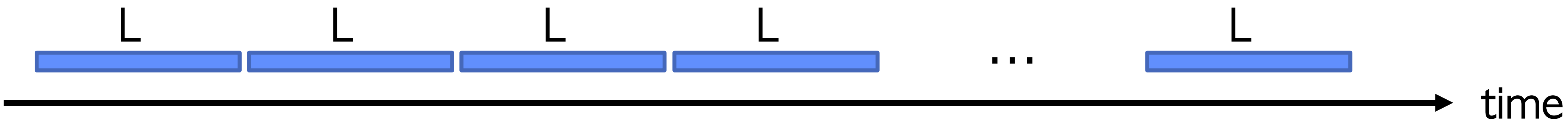
- Queuing behavior:
  – Can lead to big increases of latency as utilization increases
  – Solutions?



Response Time = Queue + I/O device service time

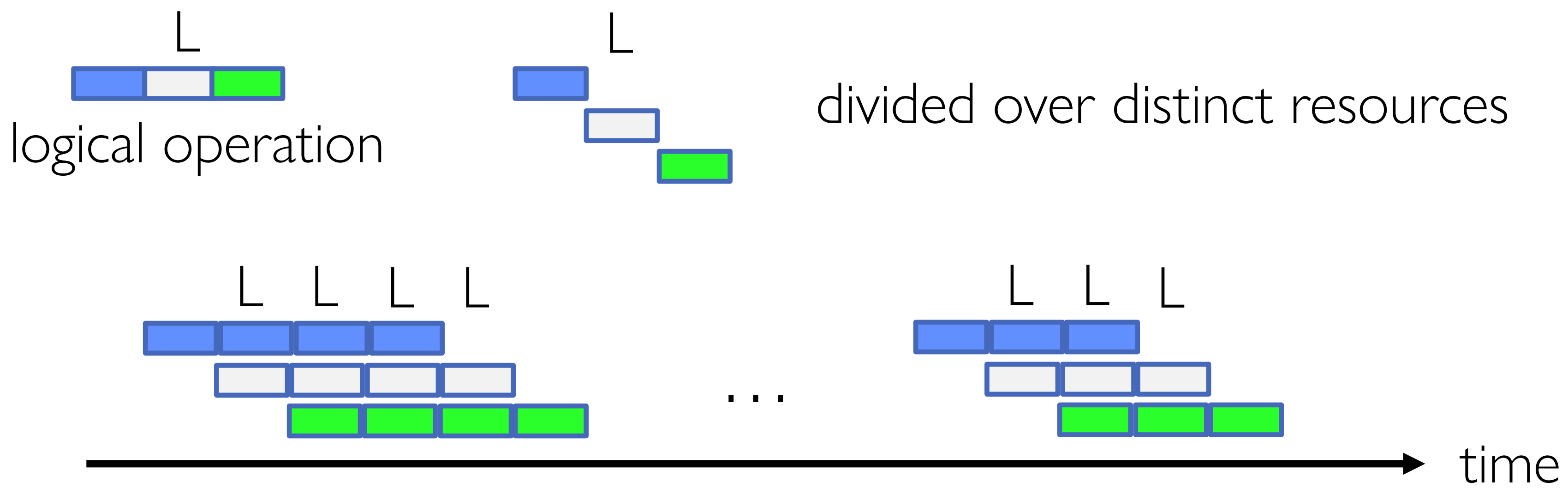


Sequential Server Performance



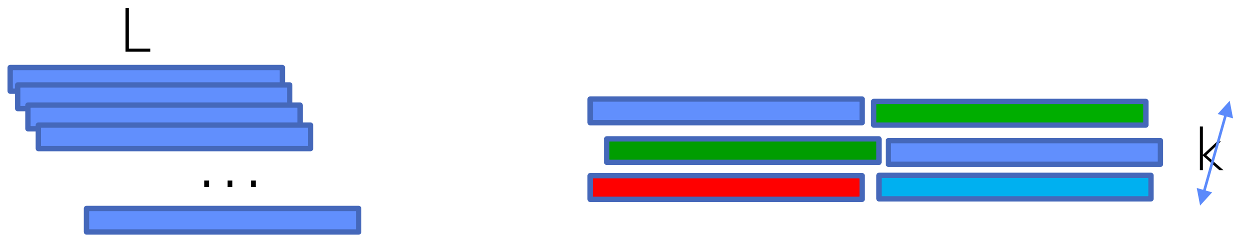
- Single sequential "server" that can complete a task in time L operates at rate $\leq \frac{1}{L}$ (on average, in steady state, …)
  - $L = 10$ ms $\rightarrow B = 100$ $^{op}/_{s}$

Single Pipelined Server



- Single pipelined server of k stages for tasks of length L (i.e., time $^{L}/_{k}$ per stage) delivers at rate $\leq {}^{k}/_{L}$.
  - $L = 10$ ms, $k = 4 \rightarrow B = 400$ $^{\text{op}}/_{\text{s}}$

Multiple Servers

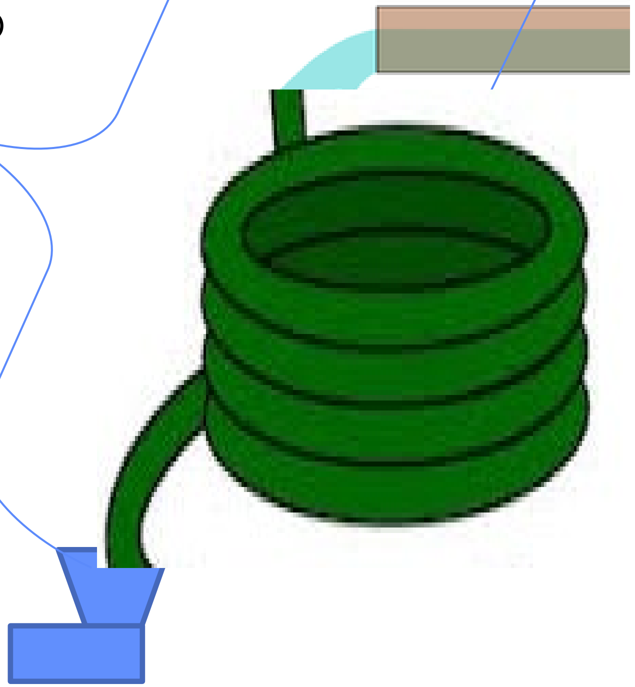


- k servers handling tasks of length L delivers at rate $\leq {}^{k}/_{L}$.
  - $L = 10$ ms, $k = 4 \rightarrow B = 400\ {}^{\text{op}}/_{\text{s}}$

A Simple Systems Performance Model

Latency (L): time per op
- How long does it take to flow through the system

Bandwidth (B): Rate, Op/s
e.g., flow: gal per min



If $B = 2$ $^{\text{ops}}/_{\text{s}}$ and $\mathbf{L} = \mathbf{3}$ s
How much water is "in the system?"

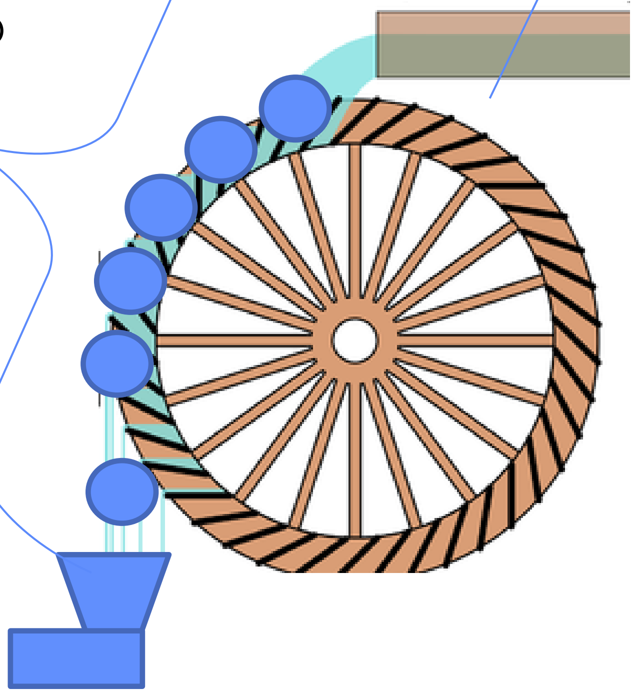
A Simple Systems Performance Model

Latency (L): time per op

- How long does it take to flow through the system

Bandwidth (B): Rate, Op/s

e.g., flow: gal per min



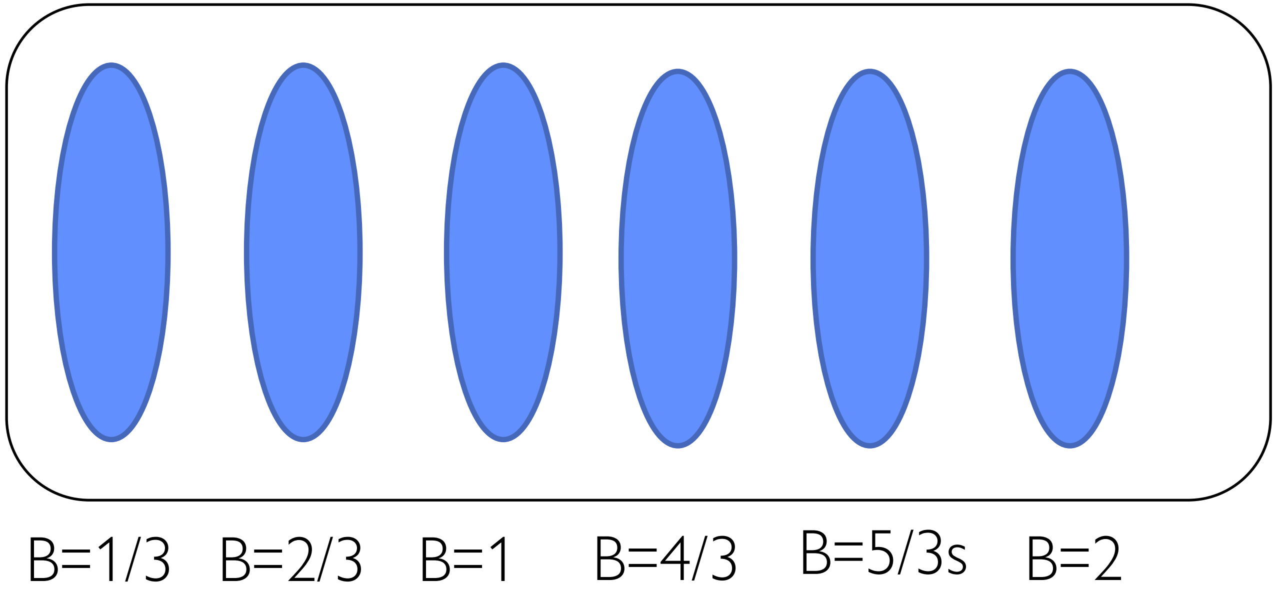
If $B = 2\ ^{op}/_{s}$ and $\mathbf{L} = 3$ s

How many ops are "in the system?"

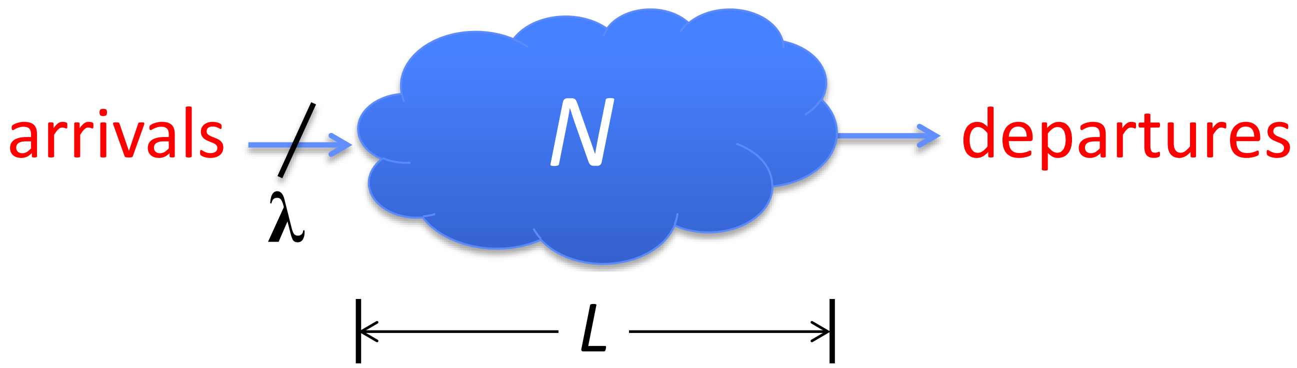
A Simple Systems Performance Model

If $B = 2\ ^{\text{op}}/_{\text{s}}$ and $\mathbf{L} = \mathbf{3}$ s

How many ops are "in the system?"

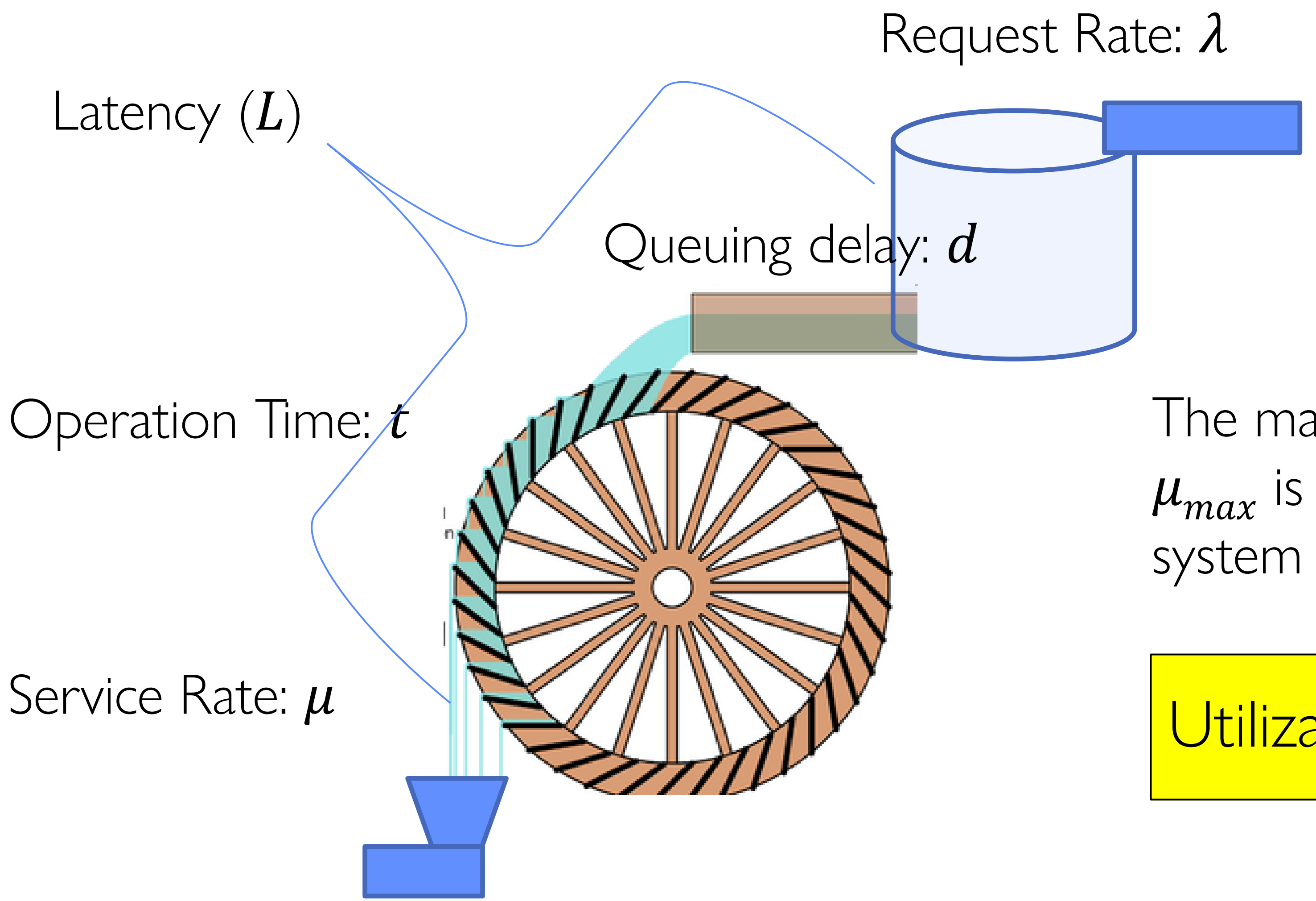


Little's Law ($B \Rightarrow \lambda$)



- In any stable system
  - Average arrival rate = Average departure rate
- The number of "things" in a system is equal to the bandwidth times the latency (on average)
  - *N (jobs) = λ (jobs/s) x L (s)*
  - Applies to any stable system
- Can be applied to an entire system:
  - Including the queues, the processing stages, parallelism, whatever
- Or to just one processing stage:
  - i.e., disk I/O subsystem, queue leading into a CPU or I/O stage, …

A Simple Systems Performance Model

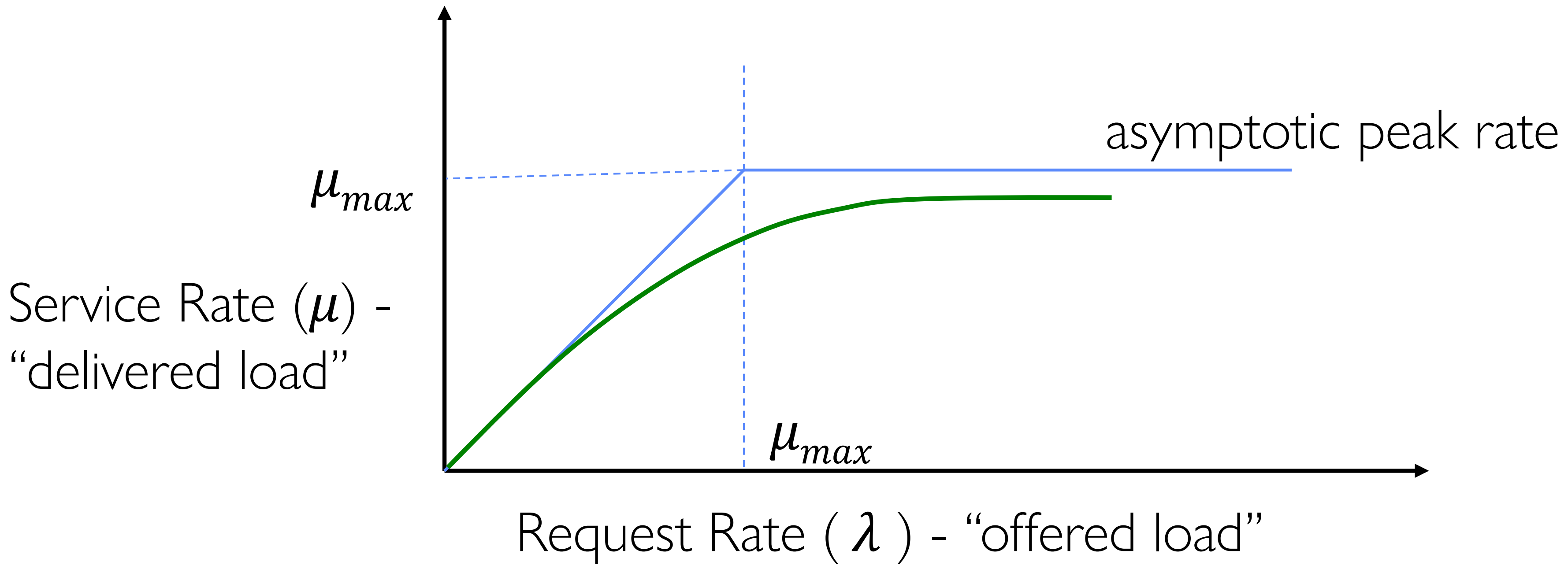


The maximum service rate μ_{max} is a property of the system – the "bottleneck"

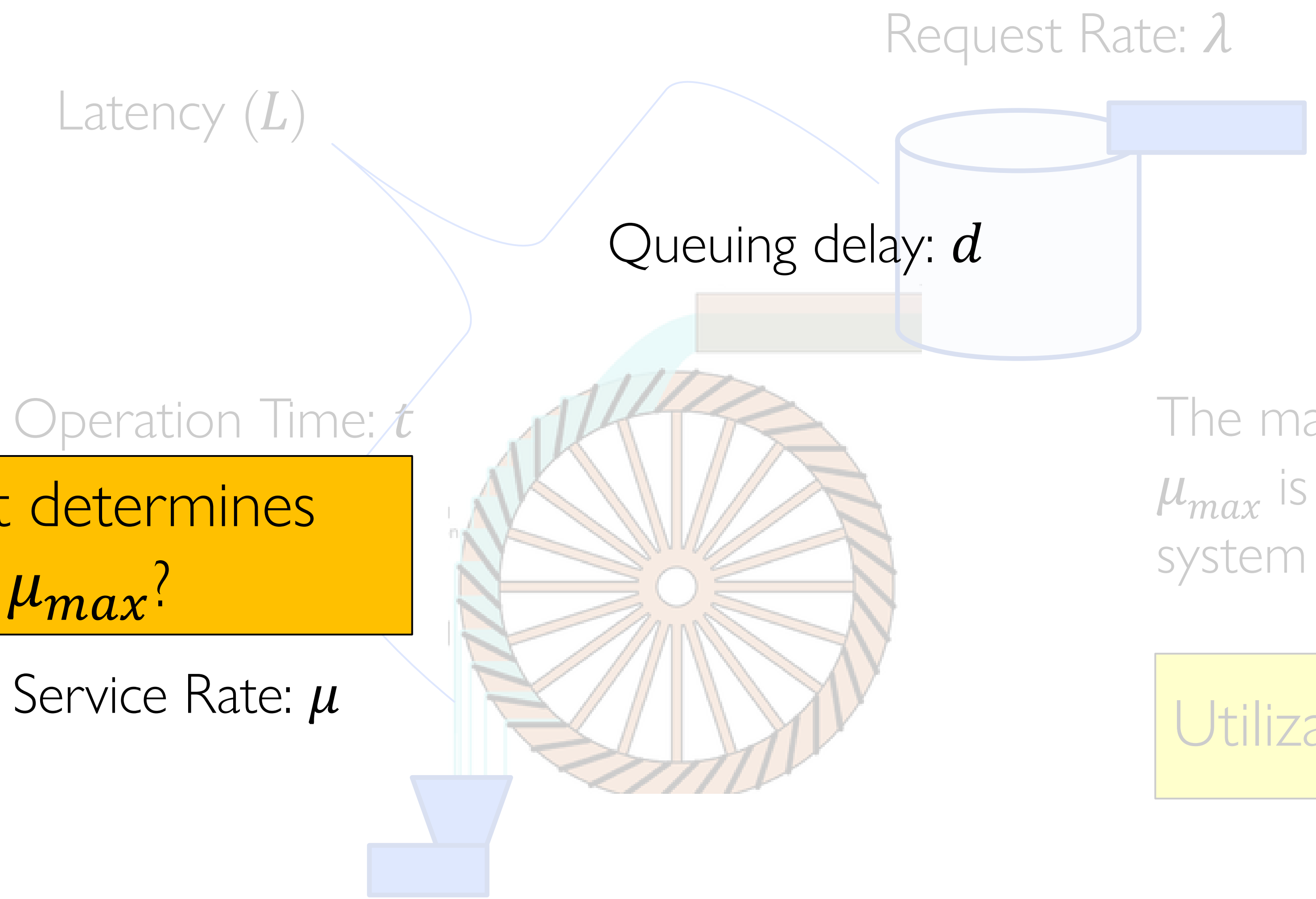
Utilization: $\rho = \frac{\lambda}{\mu_{max}}$

Ideal System Performance

How does μ (service rate) vary with λ (request rate)?



Two Related Questions



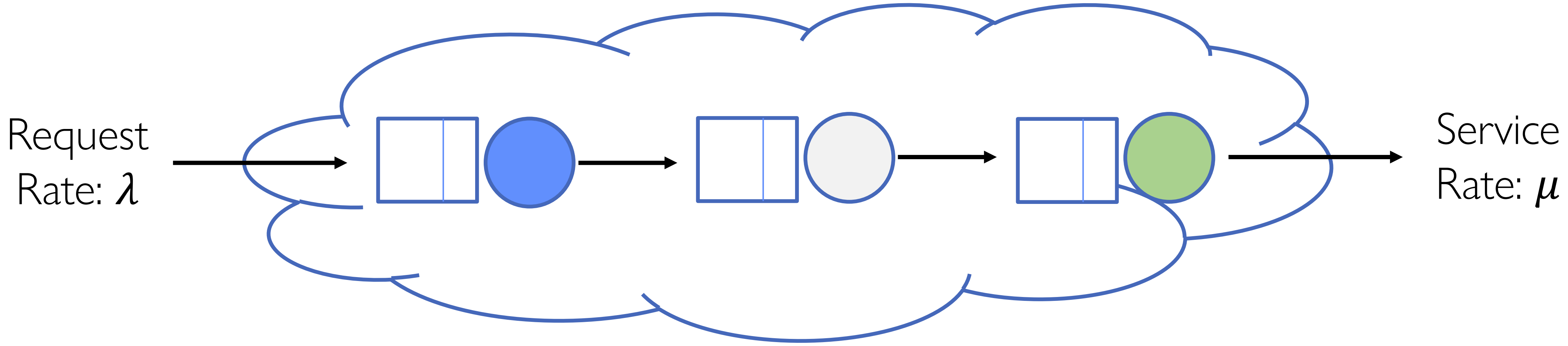
What about "internal" queues?

What determines μ_{max}?

The maximum service rate μ_{max} is a property of the system – the "bottleneck"

Utilization: $\rho = \frac{\lambda}{\mu_{max}}$

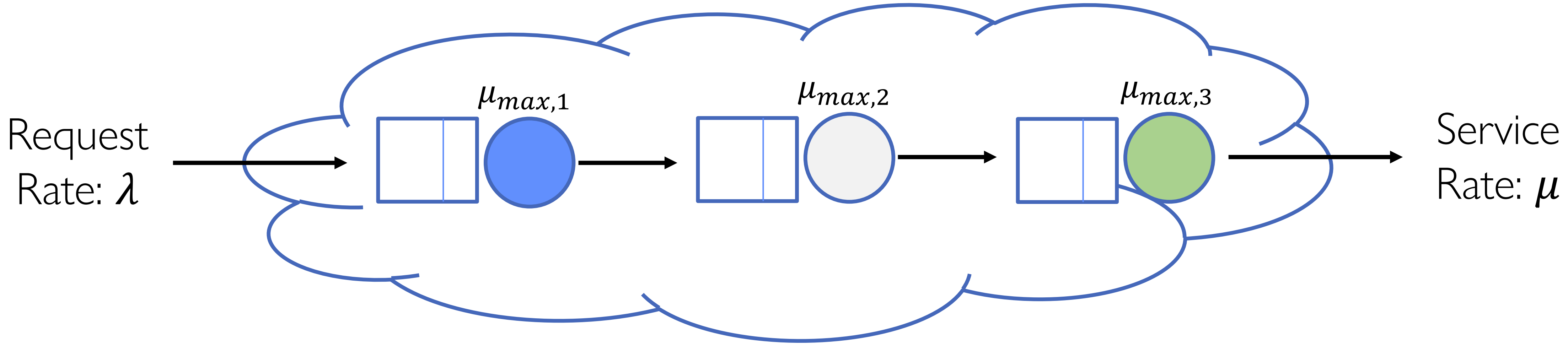
Bottleneck Analysis



Overall System: Series of Stages

Bottleneck Analysis

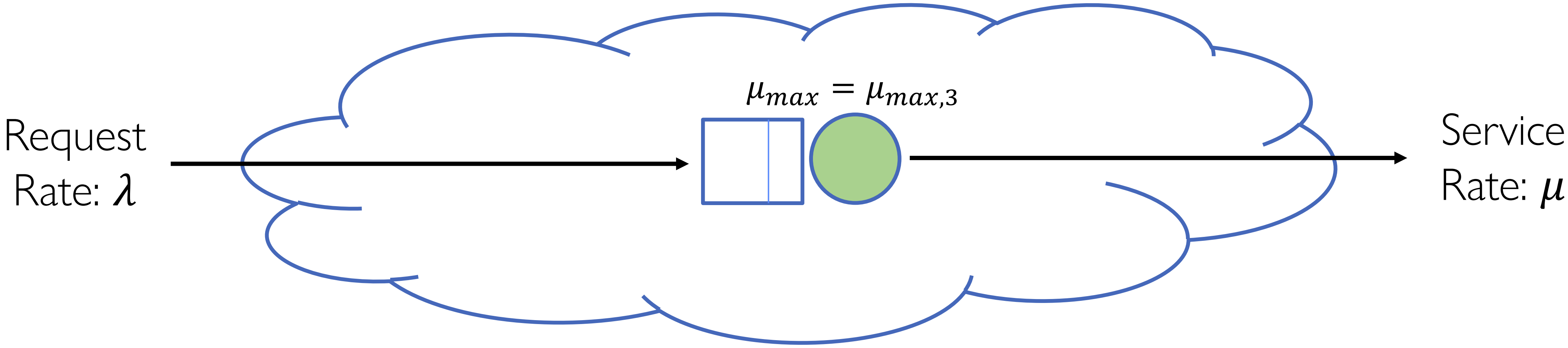
- Each stage has its own queue and maximum service rate
- Suppose the green stage is the bottleneck



Overall System: Series of Stages

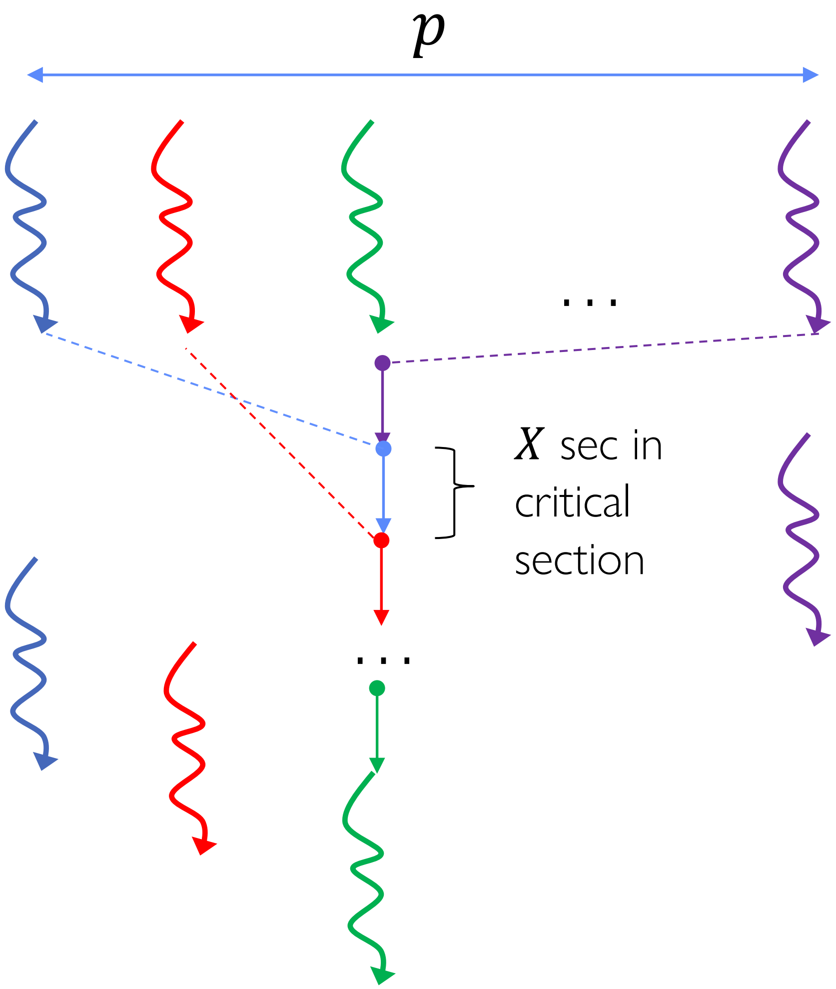
Bottleneck Analysis

- Each stage has its own queue and maximum service rate
- Suppose the green stage is the bottleneck
- The bottleneck stage dictates the maximum service rate μ_{max}



System Model: Bottleneck Stage

Example: Servicing a Highly Contended Lock

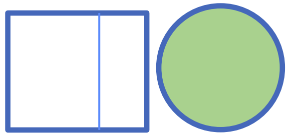


All try to grab lock

Time = $p \cdot X$ sec
Rate = $^1/_X$ ops/sec, regardless of # cores

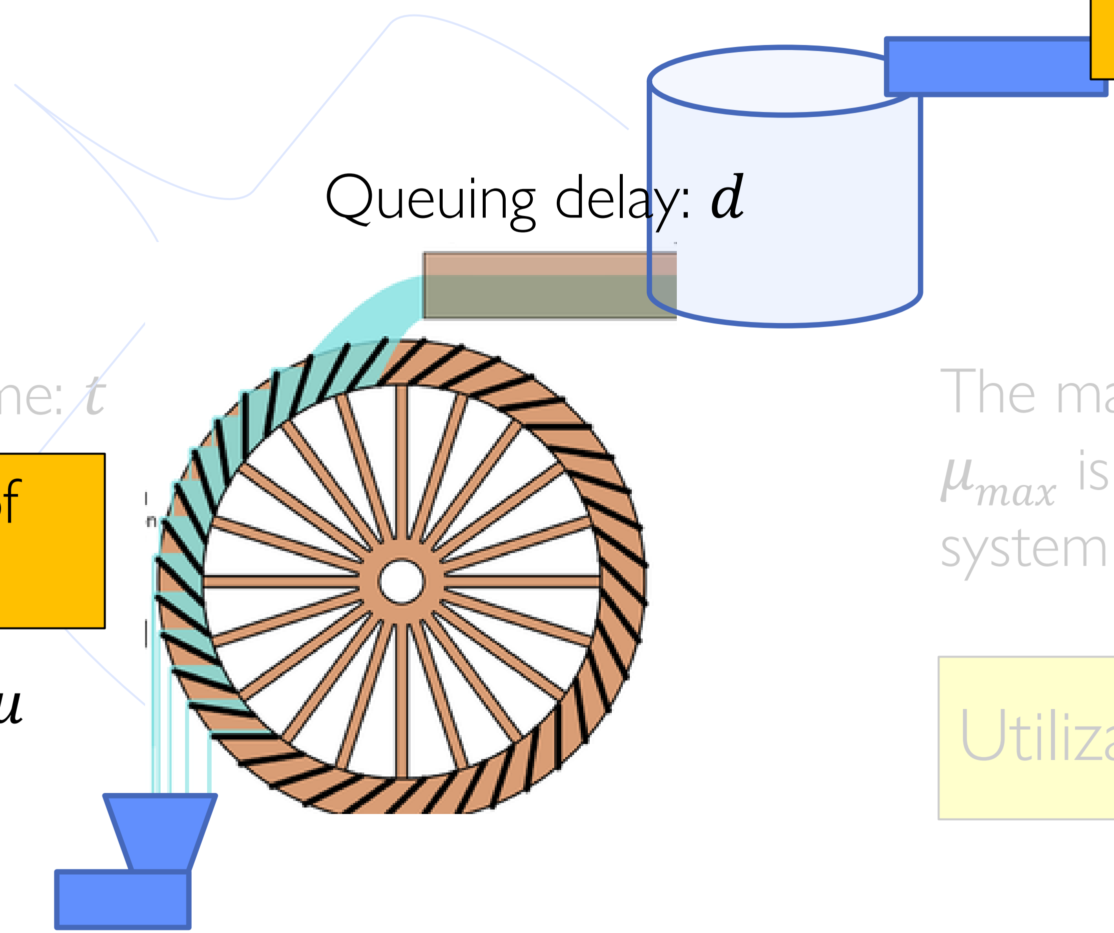
Queue of waiting threads

$\mu_{max} = {^1/_X}$



Critical section guarded by lock

Two Related Questions



Tank represents queue of bottleneck stage

μ_{max} is service rate of bottleneck stage

The maximum service rate μ_{max} is a property of the system – the "bottleneck"

Utilization: $\rho = \frac{\lambda}{\mu_{max}}$

Queuing

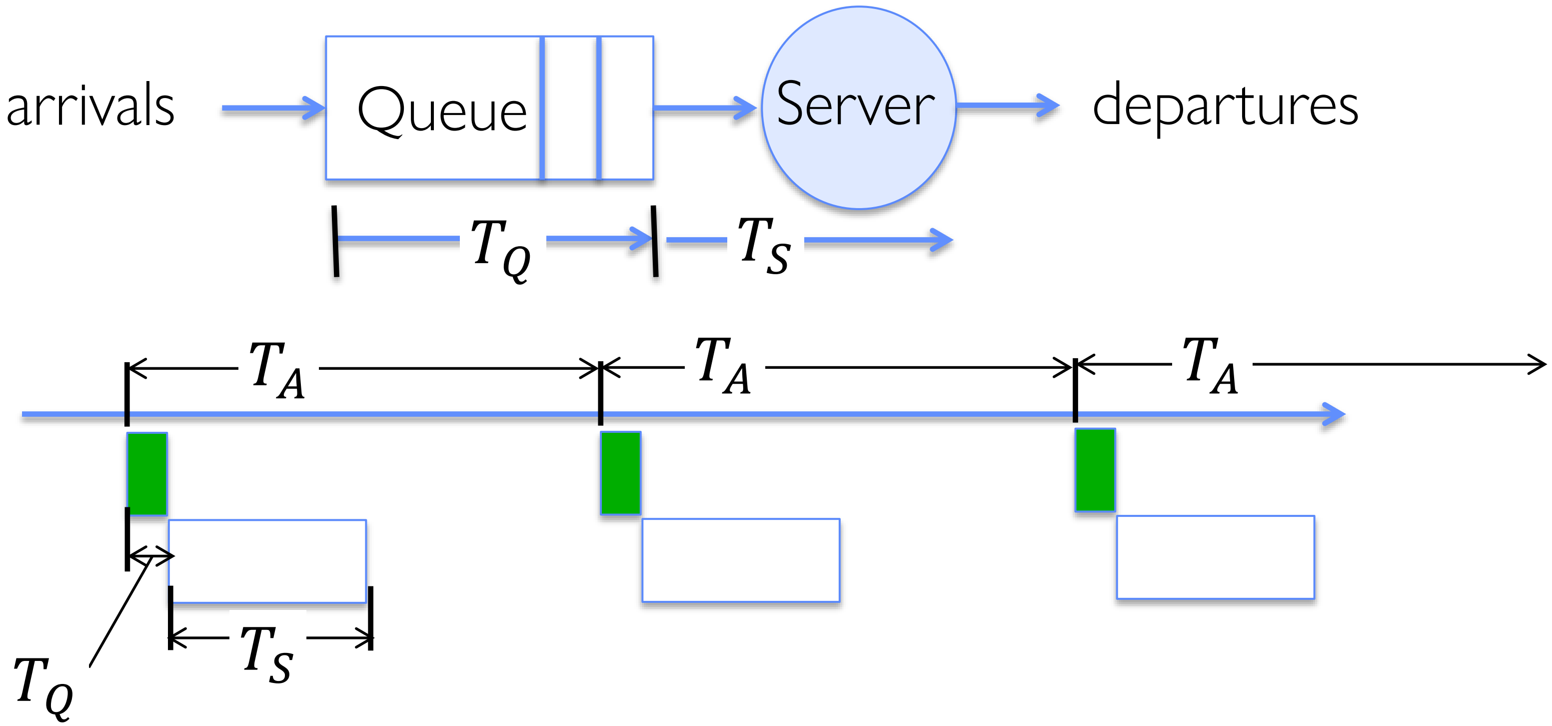
- What happens when request rate (λ) exceeds max service rate (μ_{max})?

- Short bursts can be absorbed by the queue
  - If on average $\lambda < \mu$, it will drain eventually

- Prolonged $\lambda > \mu \rightarrow$ queue will grow without bound

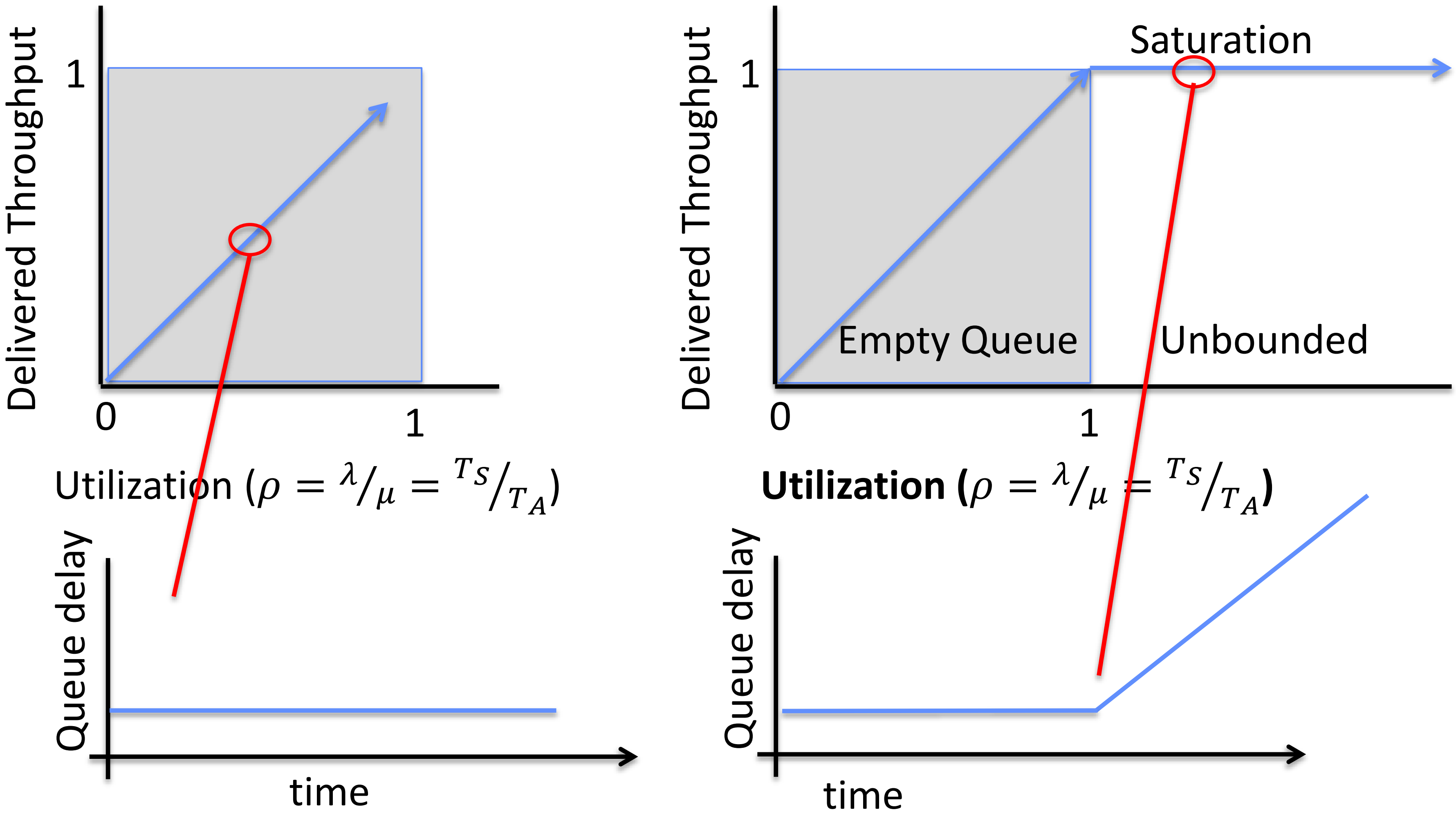
A Simple, Deterministic World

- T_A: time between arrivals
  - $\lambda = 1/T_A$
- T_S: service time
  - $\mu = k/T_S$
- T_Q: queuing time
- $L = T_Q + T_S$



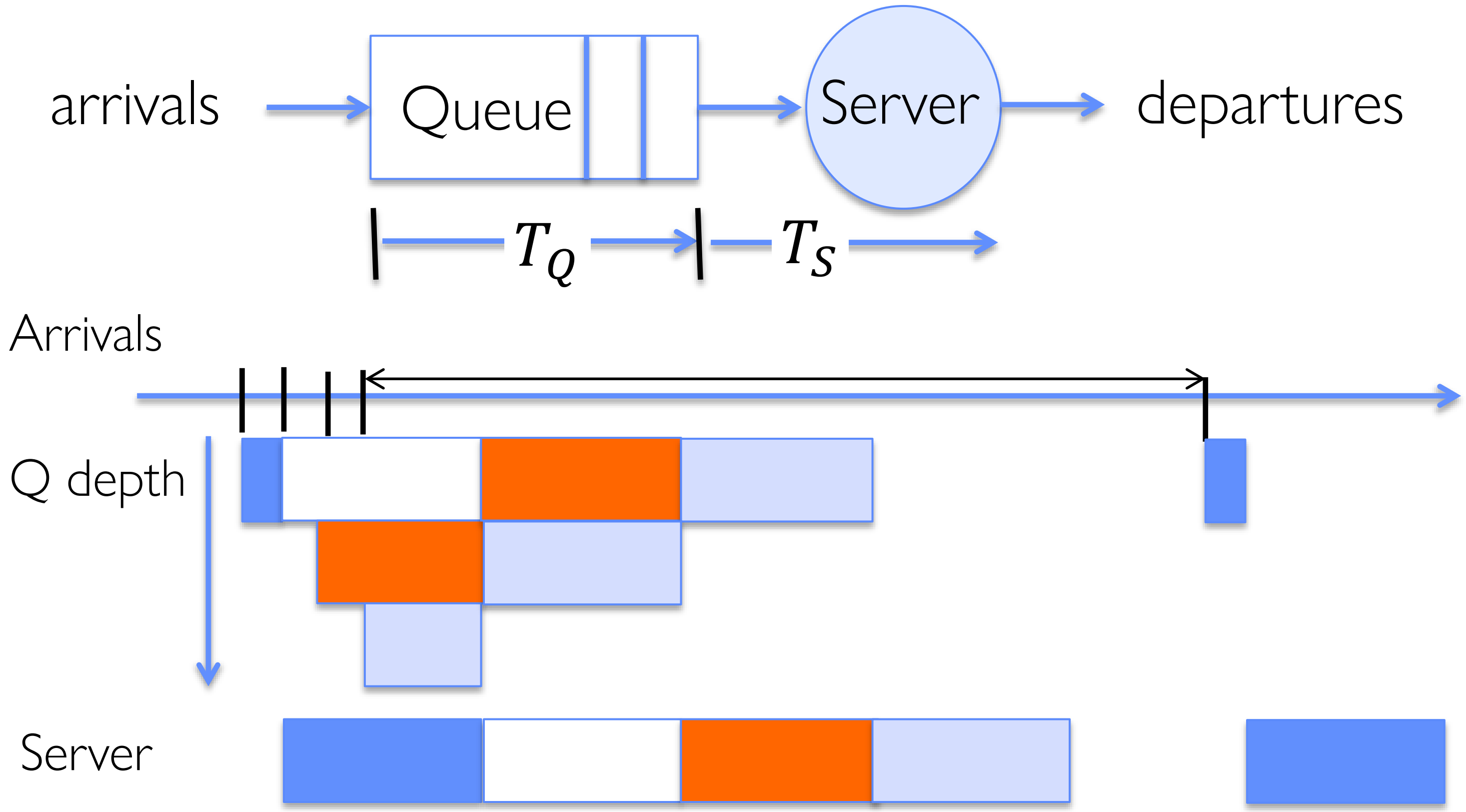
- Assume requests arrive at regular intervals, take a fixed time to process, with plenty of time between …

A Simple, Deterministic World



A Bursty World

- T_A: time between arrivals
  - Now, a random variable
- T_S: service time
  - $\mu = {}^{k}/_{T_S}$
- T_Q: queuing time
  - $L = T_Q + T_S$

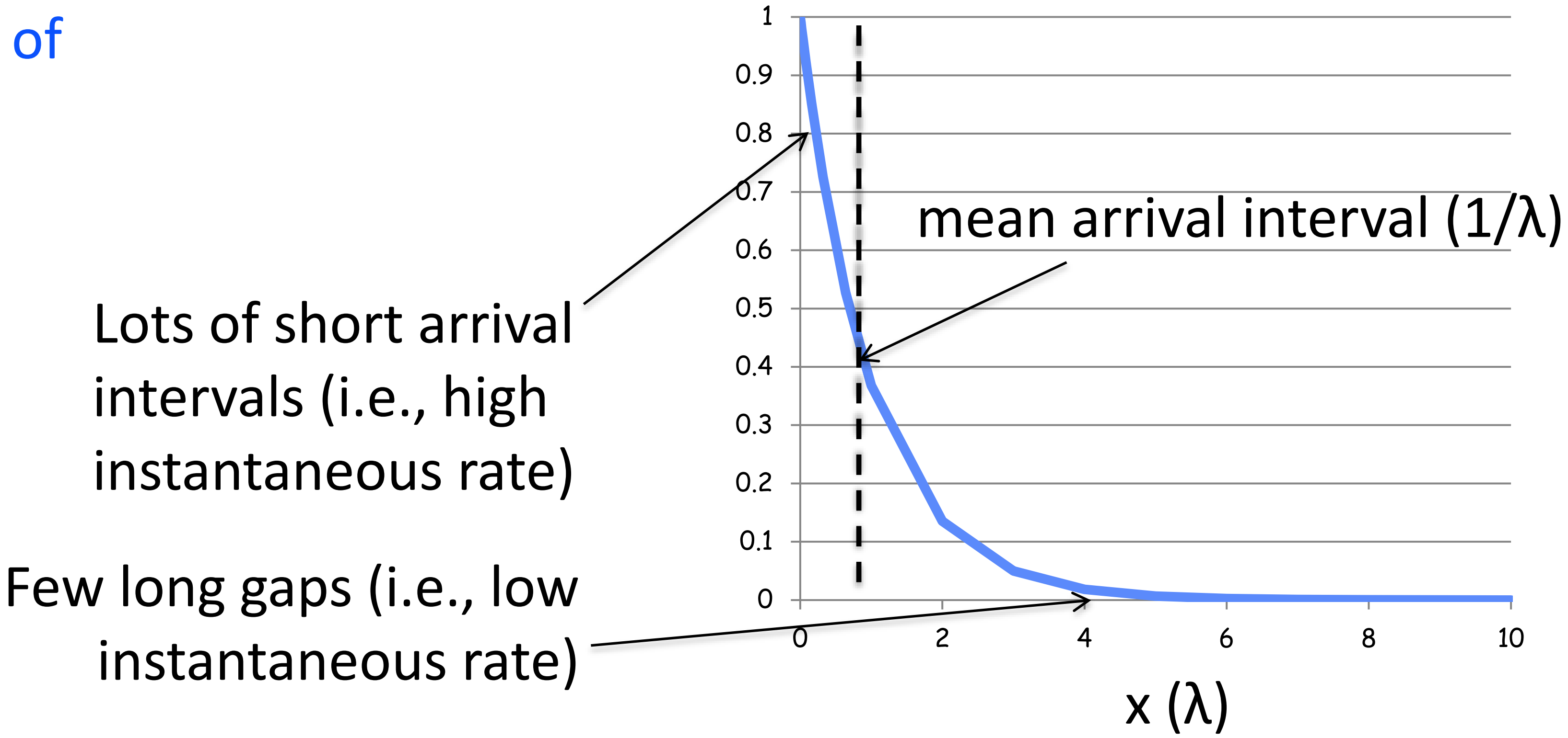


- Requests arrive in a burst, must queue up until served
- Same average arrival time, but almost all of the requests experience large queue delays (even though average utilization is low)

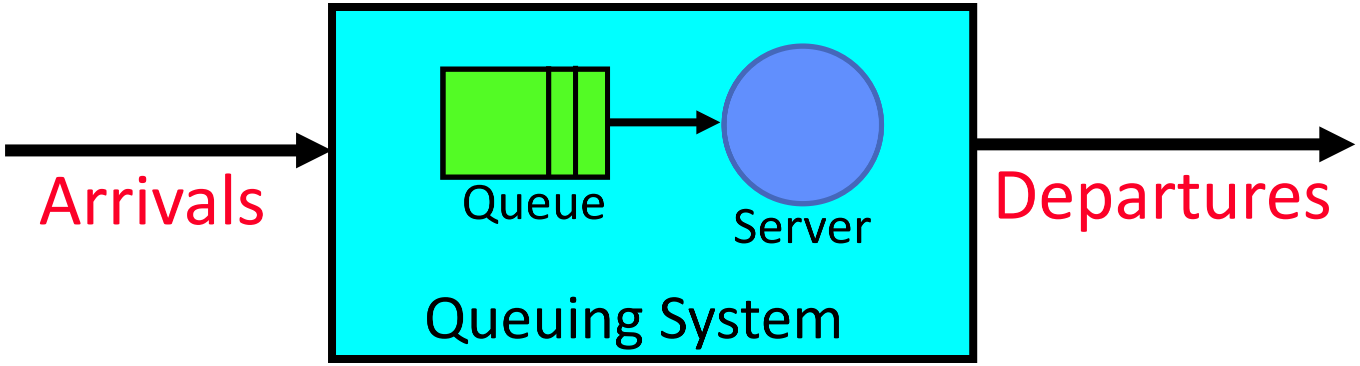
How to model burstiness of arrival?

- T_A, the time between arrivals, is now a random variable
  - Elegant mathematical framework if we model it as an *exponential distribution*
  - Probability distribution function of an exponential distribution with parameter λ is $f(x) = \lambda e^{-\lambda x}$

"Memoryless": Likelihood of an event occurring is independent of how long we've been waiting



Steady State Queuing Theory



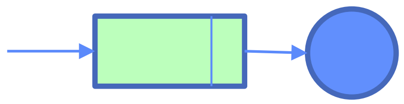
- Queuing Theory applies to long term, steady state behavior
  - Arrival rate = Departure rate

- Arrivals characterized by some probabilistic distribution

- Departures characterized by some probabilistic distribution

Little's Law Applied to a Queue

- When applied to a queue, we get:



$$L_Q = \lambda T_Q$$

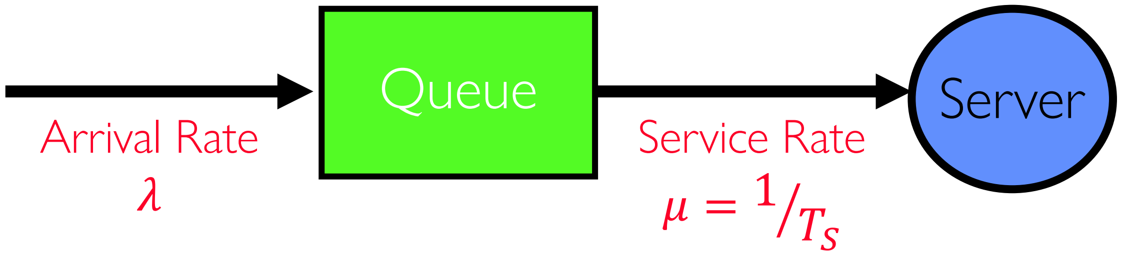
Average Arrival Rate

Average length of the queue

Average time "waiting"

Some Results from Queuing Theory

- Assumptions: system in equilibrium, no limit to the queue, time between successive arrivals is random and memoryless



- λ: arrival rate
- T_S: mean time to service a customer
- C: squared coefficient of variance $({}^{\sigma^2}/_{T_S^2})$
- μ: service rate $({}^{1}/_{T_S})$
- ρ: utilization $({}^{\lambda}/_{\mu})$

Some Results from Queuing Theory

- Memoryless service distribution ($C = 1$)—an "M/M/1 queue":

$$T_Q = \frac{\rho}{1-\rho} \cdot T_S$$

- General service distribution (no restrictions)—an "M/G/1 queue":

$$T_Q = \frac{1+C}{2} \cdot \frac{\rho}{1-\rho} \cdot T_S$$

- λ: arrival rate
- T_S: mean time to service a customer
- C: squared coefficient of variance (σ^2 / T_S^2)
- μ: service rate ($1 / T_S$)
- ρ: utilization (λ / μ)

Some Results from Queuing Theory (con't)

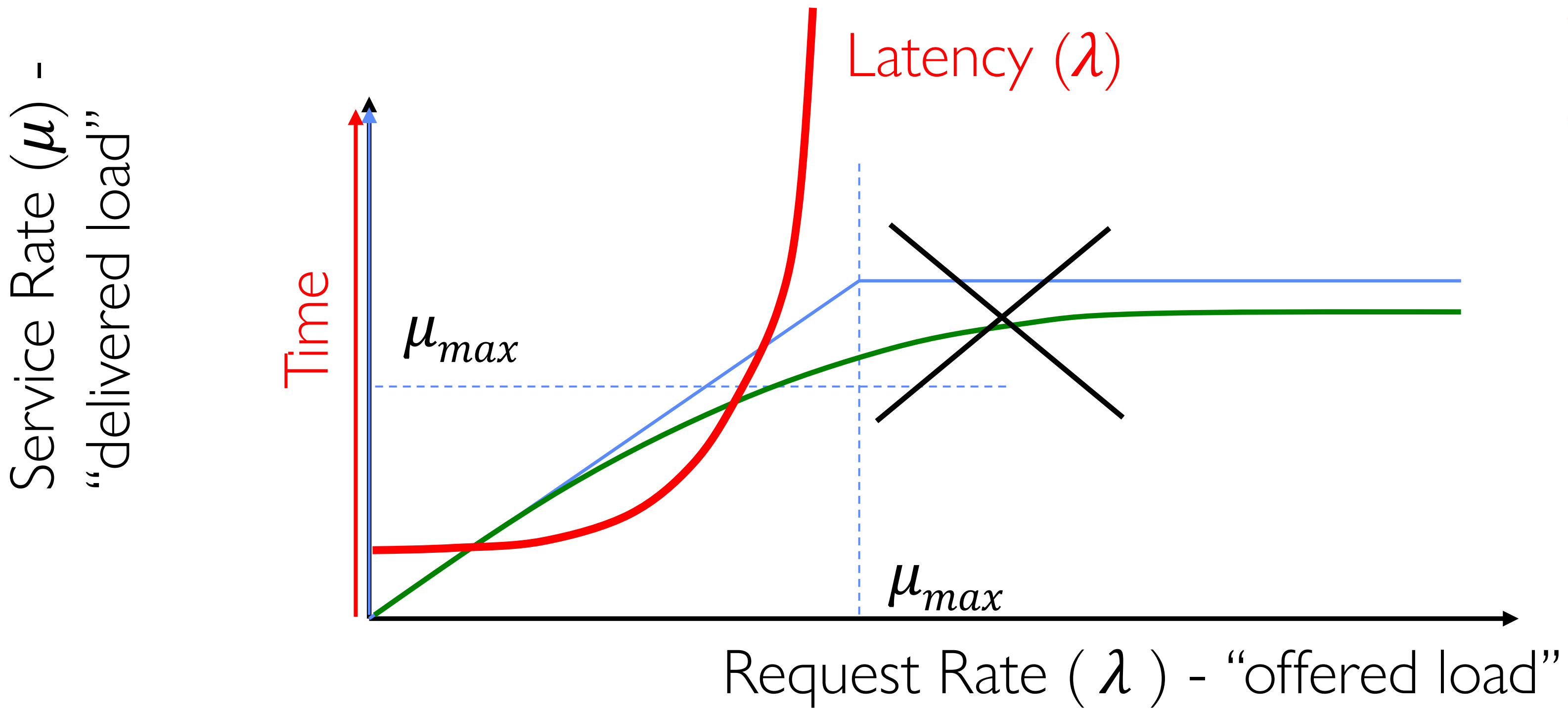
- $T_Q = \frac{\rho}{1-\rho} \cdot T_S$ (memoryless service distribution)

- $L_Q = \lambda T_Q$ (by Little's Law)

Utilization is $\rho = {}^{\lambda}/_{\mu_{max}} = \lambda T_S$, so

- $L_Q = \lambda T_Q = \frac{\rho}{T_S} \cdot T_Q = \frac{\rho^2}{1-\rho}$ (for a single server)

Ideal System Performance



- $T_Q \sim \frac{\rho}{1-\rho}$, $\rho = {}^{\lambda}/_{\mu_{max}}$
- Why does latency blow up as we approach 100% utilization?
  - Queue builds up on each burst
  - But very rarely (or never) gets a chance to drain

A Little Queuing Theory: An Example

- Example Usage Statistics:
  - User requests 10 x 8KB disk I/Os per second
  - Requests & service exponentially distributed (C=1.0)
  - Avg. service = 20 ms (From controller+seek+rot+trans)
- Questions:
  - How utilized is the disk?
    - » Ans: server utilization, $\rho = \lambda T_{ser}$
  - What is the average time spent in the queue?
    - » Ans: T_q
  - What is the number of requests in the queue?
    - » Ans: L_q
  - What is the avg response time for disk request?
    - » Ans: $T_{sys} = T_q + T_{ser}$
- Computation:

λ (avg # arriving customers/s) = 10/s

T_{ser} (avg time to service customer) = 20 ms (0.02s)

ρ (server utilization) = $\lambda \times T_{ser}$ = 10/s x .02s = 0.2

T_q (avg time/customer in queue) = $T_{ser} \times u/(1 - u)$
= 20 x 0.2/(1-0.2) = 20 x 0.25 = 5 ms (0 .005s)

L_q (avg length of queue) = $\lambda \times T_q$=10/s x .005s = 0.05

T_{sys} (avg time/customer in system) =$T_q + T_{ser}$= 25 ms

Conclusion

- Two types of storage devices:
  – HDDs, which are organized as a set of platters split into tracks, which are being spinned by a motor. HDDs have suffer from rotational delay and high seek latency
  – SDDs are built on top of flash technology. Flash offers three operations: read, write, program
  – SDDs do not suffer from seek/rotation delay but suffer from wear out.

- Performance
  – Bottleneck & queueing delay
  – Model arrival/departure rate as probability distributions