CS162
Operating Systems and Systems Programming
Lecture 17

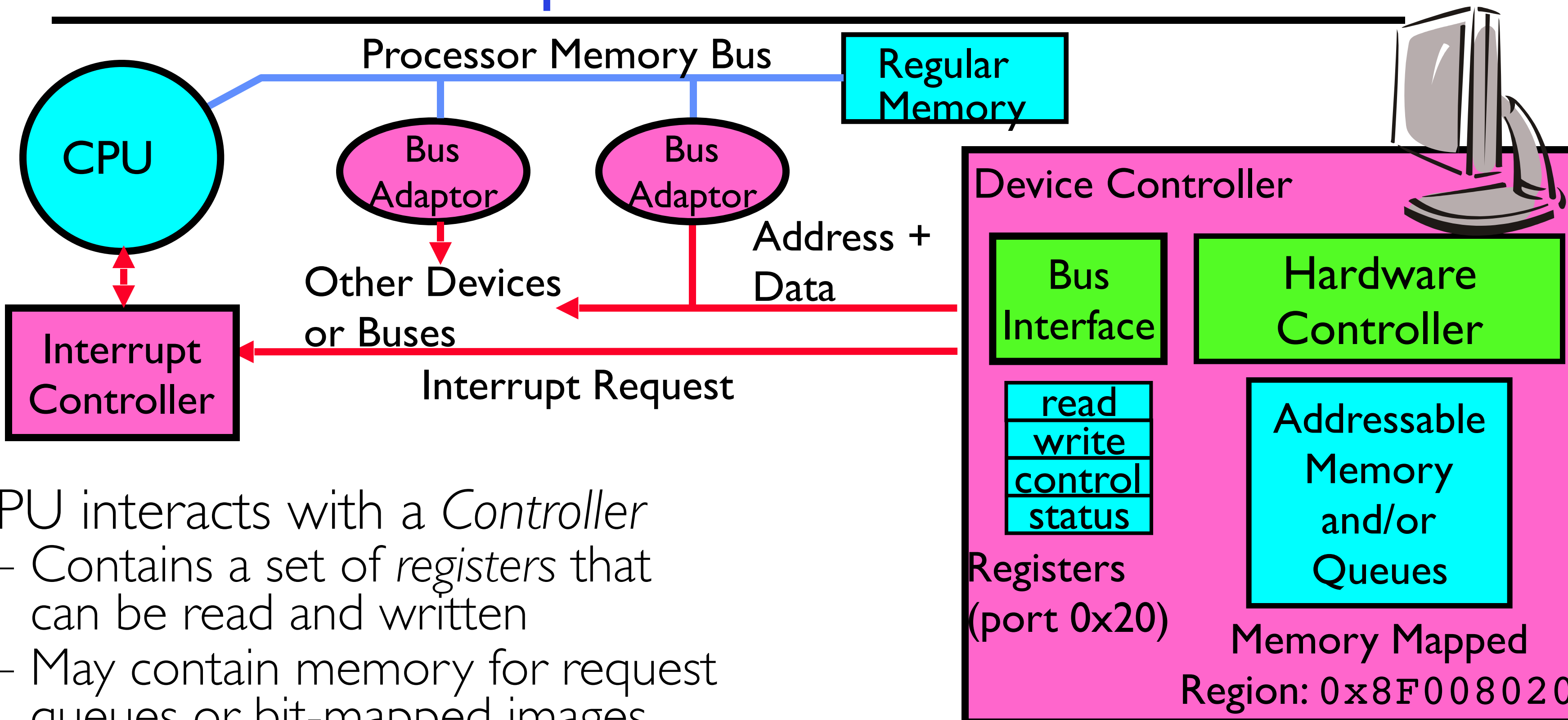
Performance
Storage Devices, Queueing Theory

May 31st, 2020

Prof. John Kubiatowicz
http://cs162.eecs.Berkeley.edu

Acknowledgments: Lecture slides are from the Operating Systems course taught by John Kubiatowicz at Berkeley, with few minor updates/changes*. When slides are obtained from other sources, a a reference will be noted on the bottom of that slide, in which case a full list of references is provided on the last slide.*

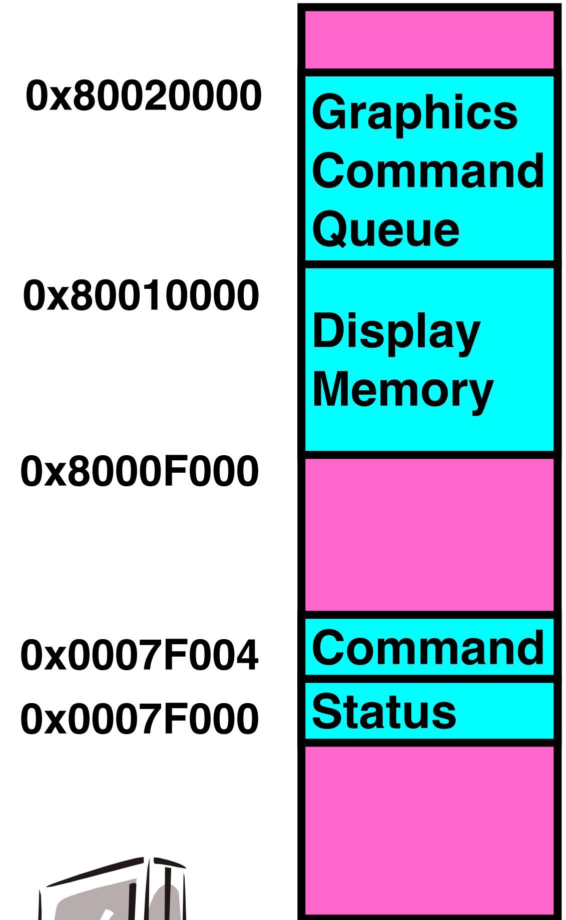
Recall: How the processor talks to the device

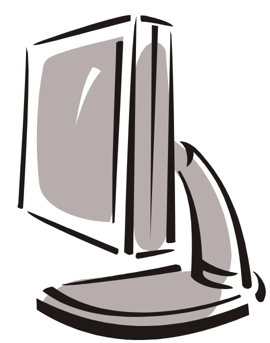


- CPU interacts with a *Controller*
  - Contains a set of *registers* that can be read and written
  - May contain memory for request queues or bit-mapped images
- Regardless of the complexity of the connections and buses, processor accesses registers in two ways:
  - I/O instructions: in/out instructions
    - » Example from the Intel architecture: `out 0x21,AL`
  - Memory mapped I/O: load/store instructions
    - » Registers/memory appear in physical address space
    - » I/O accomplished with load and store instructions

Recall: Memory-Mapped Display Controller

- Memory-Mapped:
  – Hardware maps control registers and display memory into physical address space
    » Addresses set by HW jumpers or at boot time
  – Simply writing to display memory (also called the "frame buffer") changes image on screen
    » Addr: 0x8000F000 — 0x8000FFFF
  – Writing graphics description to cmd queue
    » Say enter a set of triangles describing some scene
    » Addr: 0x80010000 — 0x8001FFFF
  – Writing to the command register may cause on-board graphics hardware to do something
    » Say render the above scene
    » Addr: 0x0007F004
- Can protect with address translation

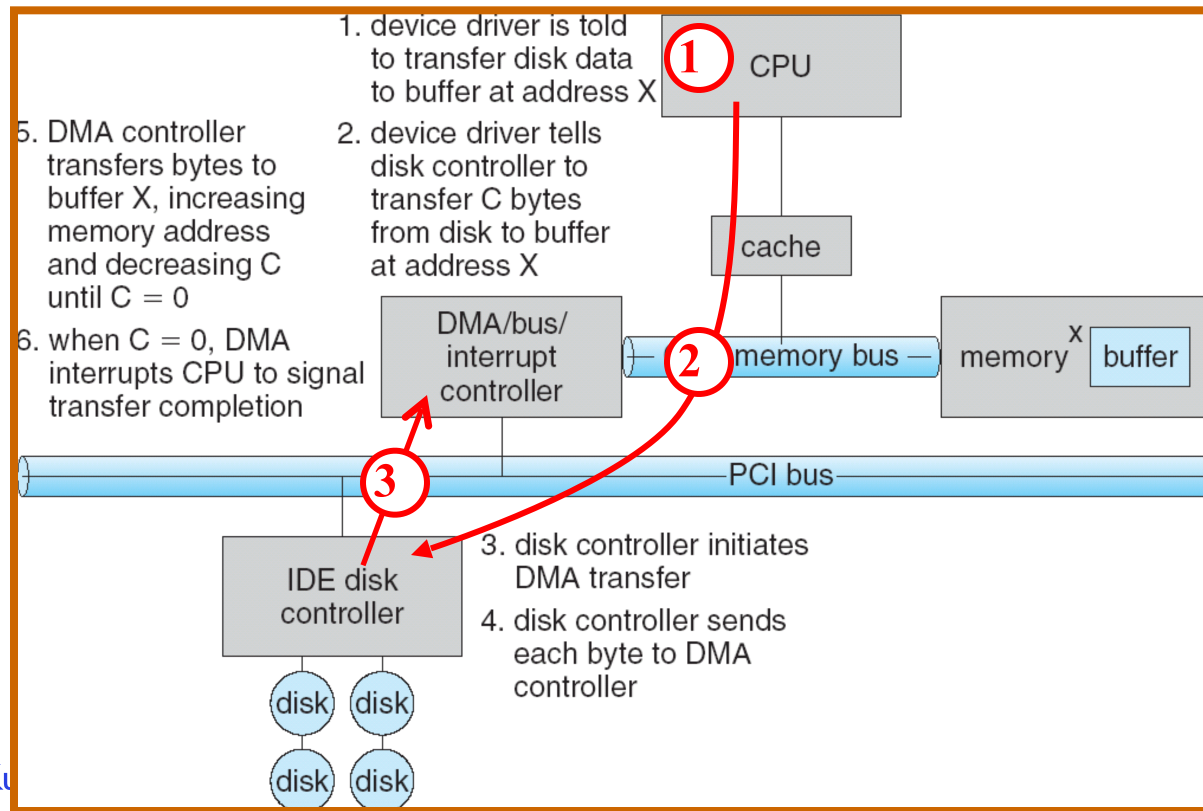




Physical Address Space

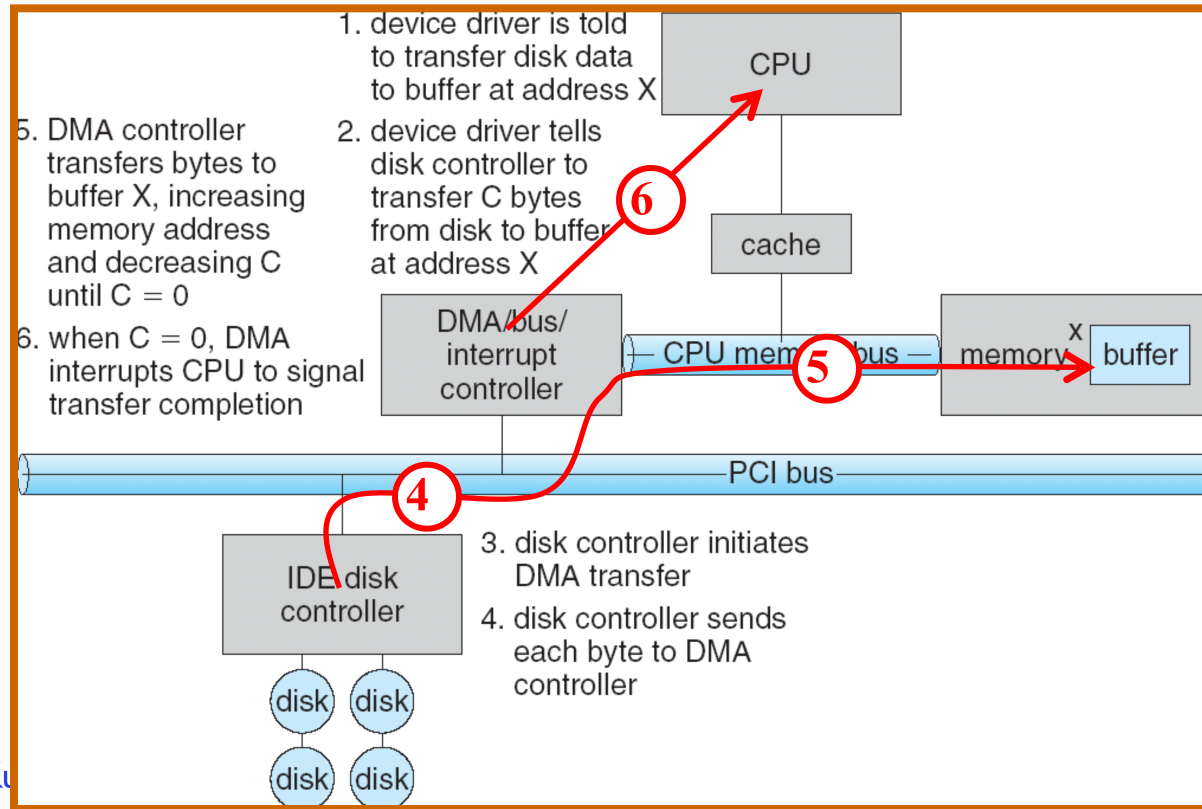
Transferring Data To/From Controller

- Programmed I/O:
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size
- Direct Memory Access:
  - Give controller access to memory bus
  - Ask it to transfer data blocks to/from memory directly
- Sample interaction with DMA controller (from OSC book):



Transferring Data To/From Controller

- Programmed I/O:
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size
- Direct Memory Access:
  - Give controller access to memory bus
  - Ask it to transfer data blocks to/from memory directly
- Sample interaction with DMA controller (from OSC book):



I/O Device Notifying the OS

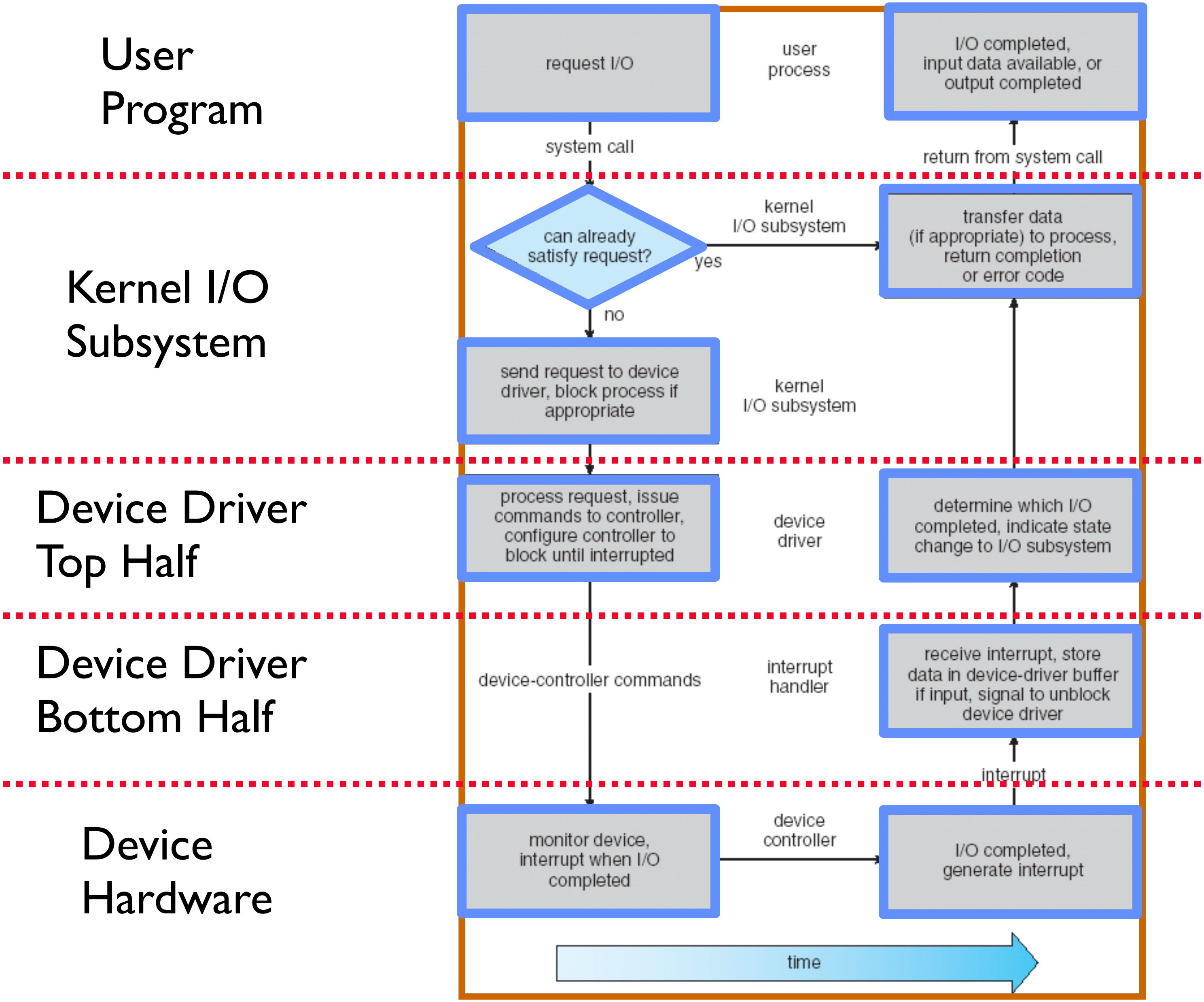
- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- I/O Interrupt:
  - Device generates an interrupt whenever it needs service
  - Pro: handles unpredictable events well
  - Con: interrupts relatively high overhead
- Polling:
  - OS periodically checks a device-specific status register
    - » I/O device puts completion information in status register
  - Pro: low overhead
  - Con: may waste many cycles on polling if infrequent or unpredictable I/O operations
- Actual devices combine both polling and interrupts
  - For instance – High-bandwidth network adapter:
    - » Interrupt for first incoming packet
    - » Poll for following packets until hardware queues are empty

Device Drivers

- Device Driver: Device-specific code in the kernel that interacts directly with the device hardware
  - Supports a standard, internal interface
  - Same kernel I/O system can interact easily with different device drivers
  - 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

Life Cycle of An I/O Request

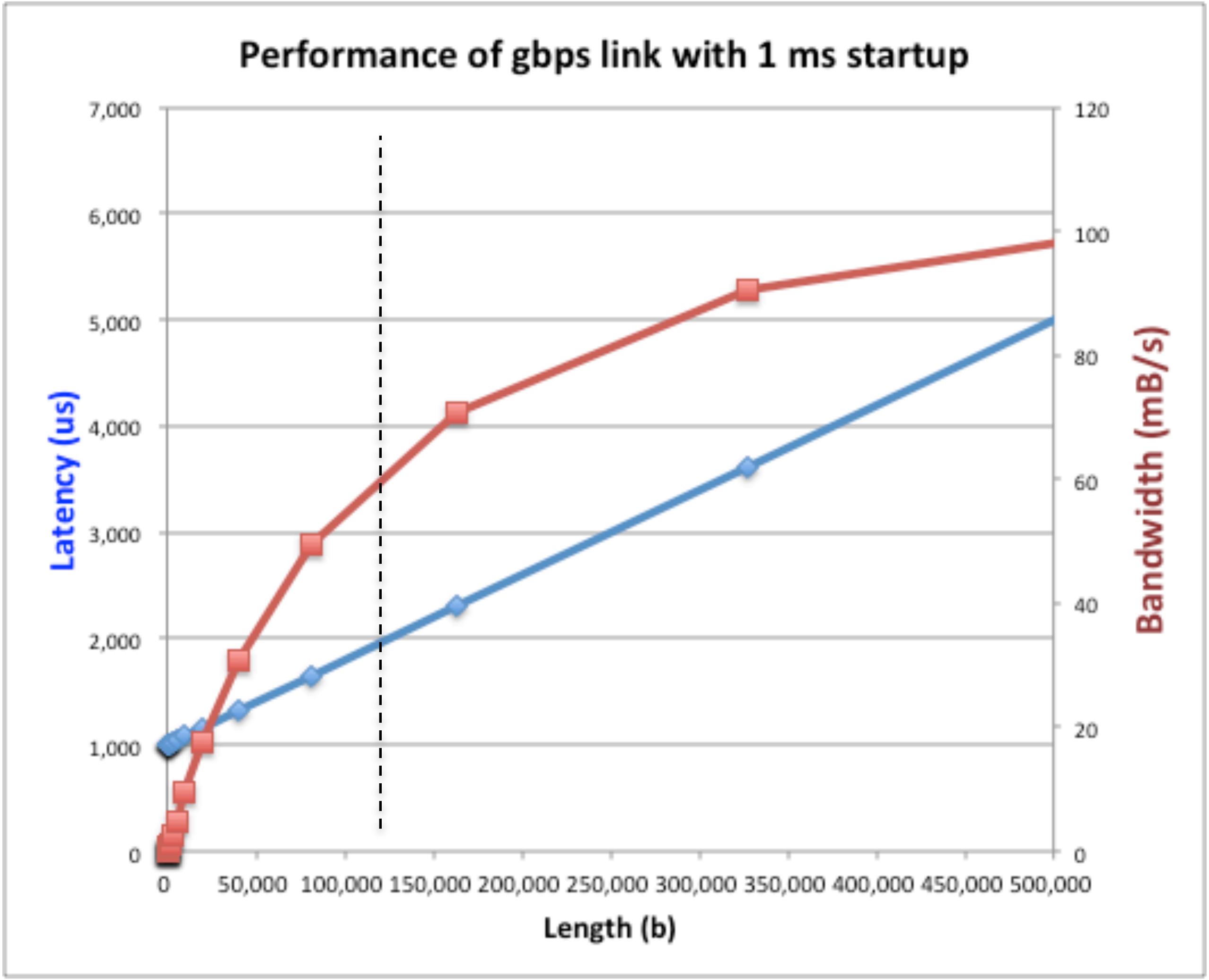


Basic Performance Concepts

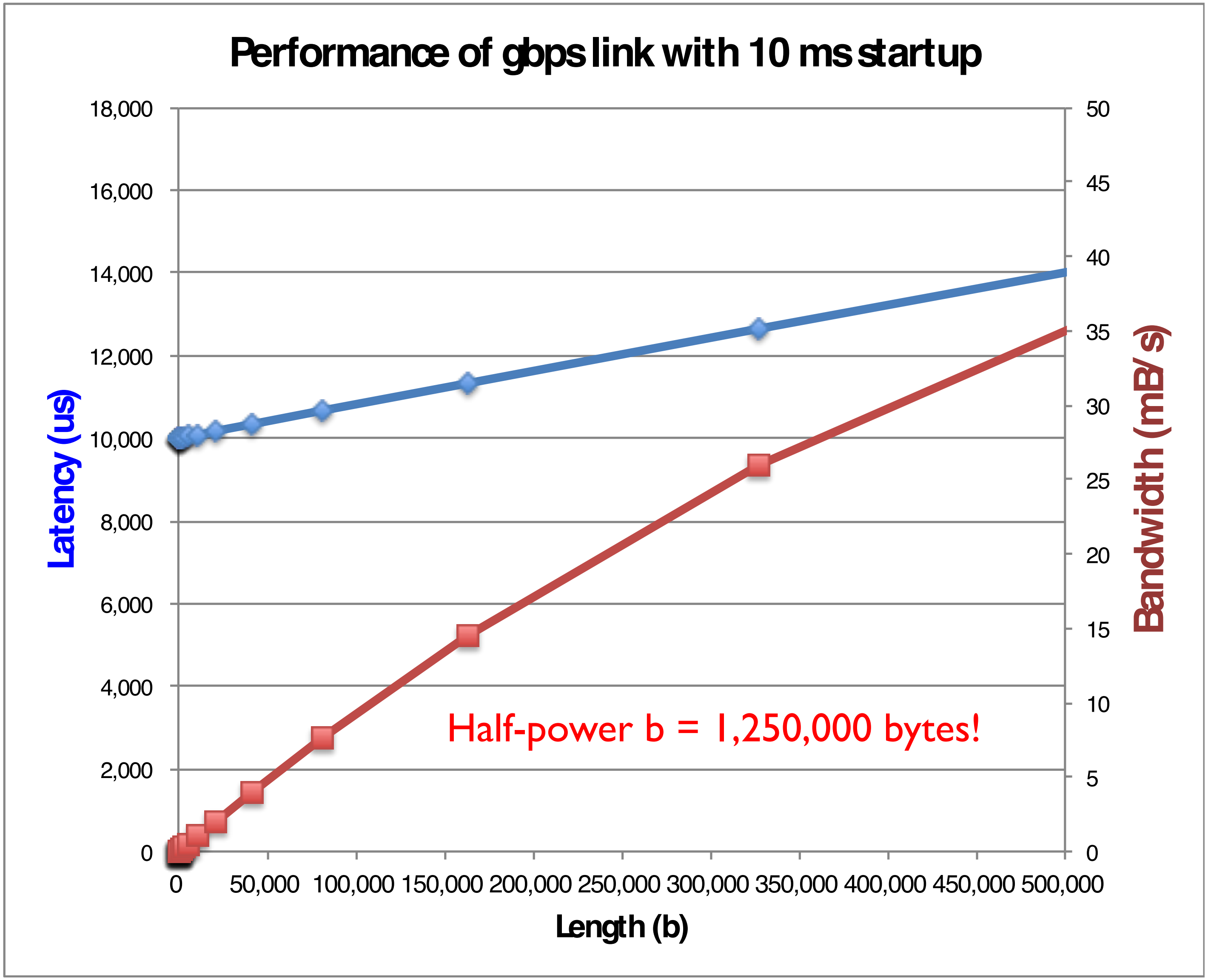
- *Response Time or Latency*: Time to perform an operation(s)
- *Bandwidth or Throughput*: Rate at which operations are performed (op/s)
  – Files: MB/s, Networks: Mb/s, Arithmetic: GFLOP/s
- *Start up* or "Overhead": time to initiate an operation
- Most I/O operations are roughly linear in *b* bytes
  – Latency(*b*) = Overhead + *b*/TransferCapacity

Example (Fast Network)

- Consider a 1 Gb/s link (BW = 125 MB/s)
  - With a startup cost S = 1 ms



Example: at 10 ms startup (like Disk)



What Determines Peak BW for I/O ?

- Bus Speed
  - PCI-X: 1064 MB/s = 133 MHz x 64 bit (per lane)
  - ULTRA WIDE SCSI: 40 MB/s
  - Serial ATA & IEEE 1394 (firewire): 1.6 Gb/s full duplex (200MB/s)
  - SAS-1: 3 Gb/s, SAS-2: 6 Gb/s, SAS-3: 12 Gb/s, SAS-4: 22.5 GB/s
  - USB 3.0 – 5 Gb/s
  - Thunderbolt 3 – 40 Gb/s
- Device Transfer Bandwidth
  - Rotational speed of disk
  - Write / Read rate of NAND flash
  - Signaling rate of network link
- Whatever is the bottleneck in the path…

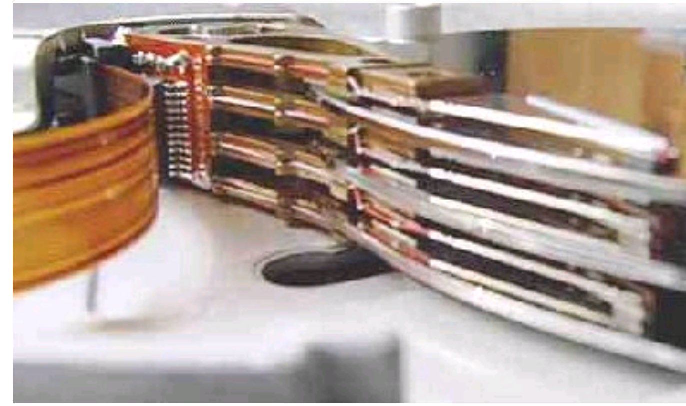
Storage Devices

- 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
  - Erasure requirement in large blocks
  - Wear patterns issue

Hard Disk Drives (HDDs)

Western Digital Drive
http://www.storagereview.com/guide/



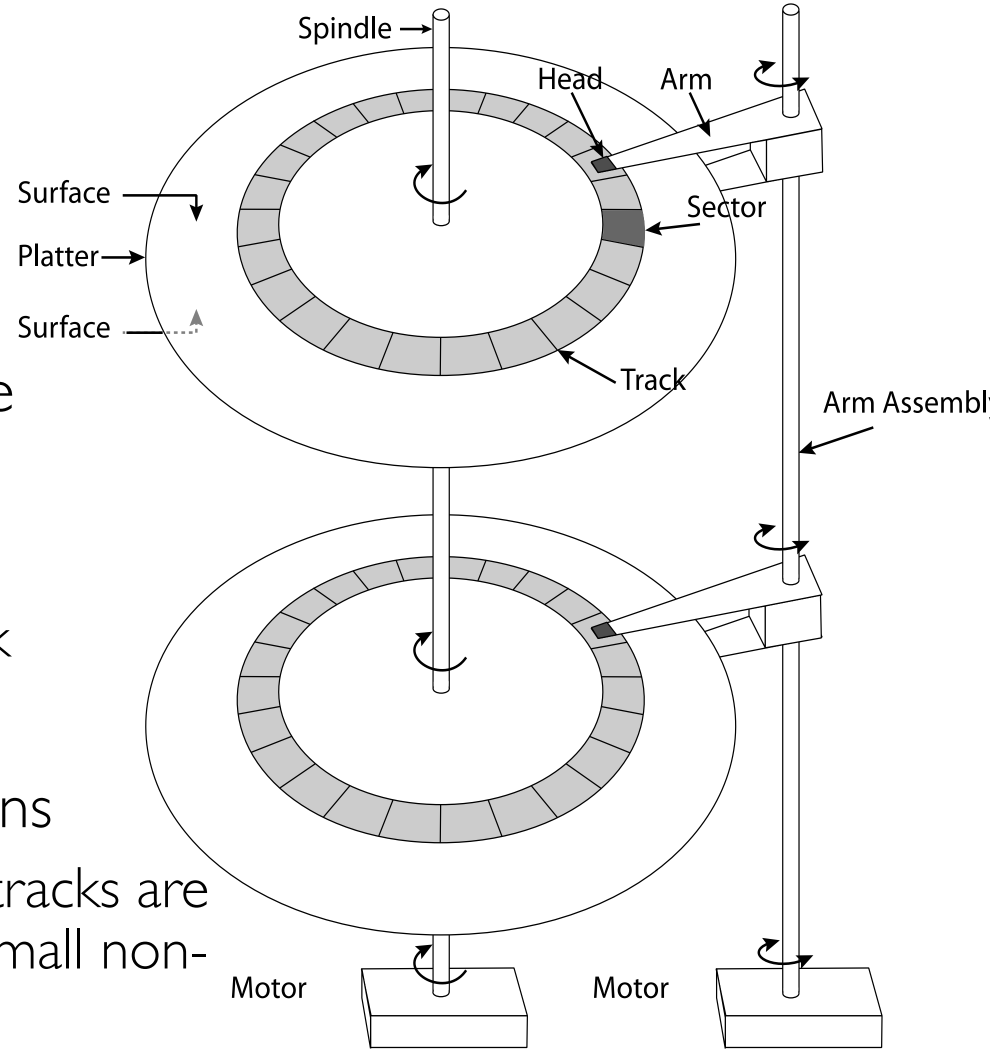
Read/Write Head Side View

IBM/Hitachi Microdrive

IBM Personal Computer/AT (1986)
- 30 MB hard disk - $500
- 30-40ms seek time
- 0.7-1 MB/s (est.)

The Amazing Magnetic Disk

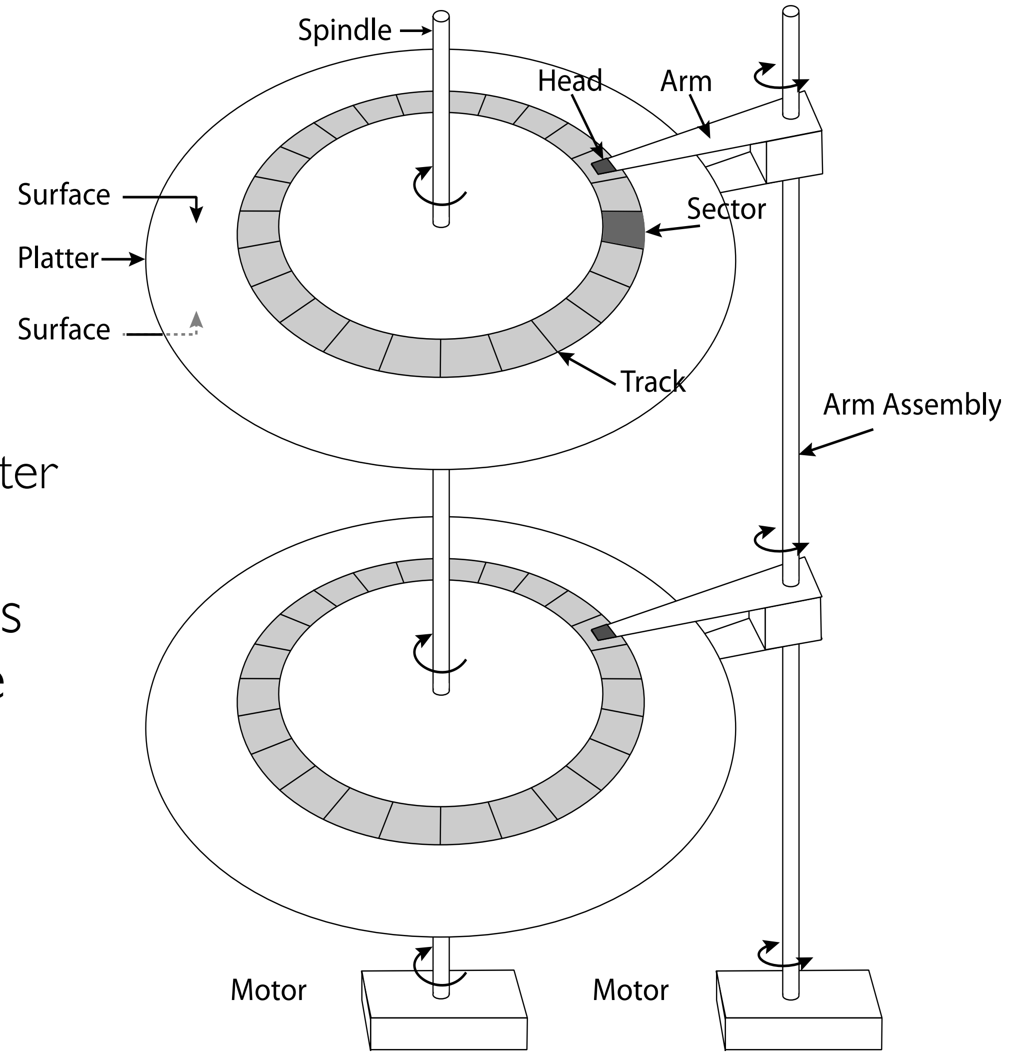
- Unit of Transfer: Sector
  - Ring of sectors form a track
  - Stack of tracks form a cylinder
  - Heads position on cylinders



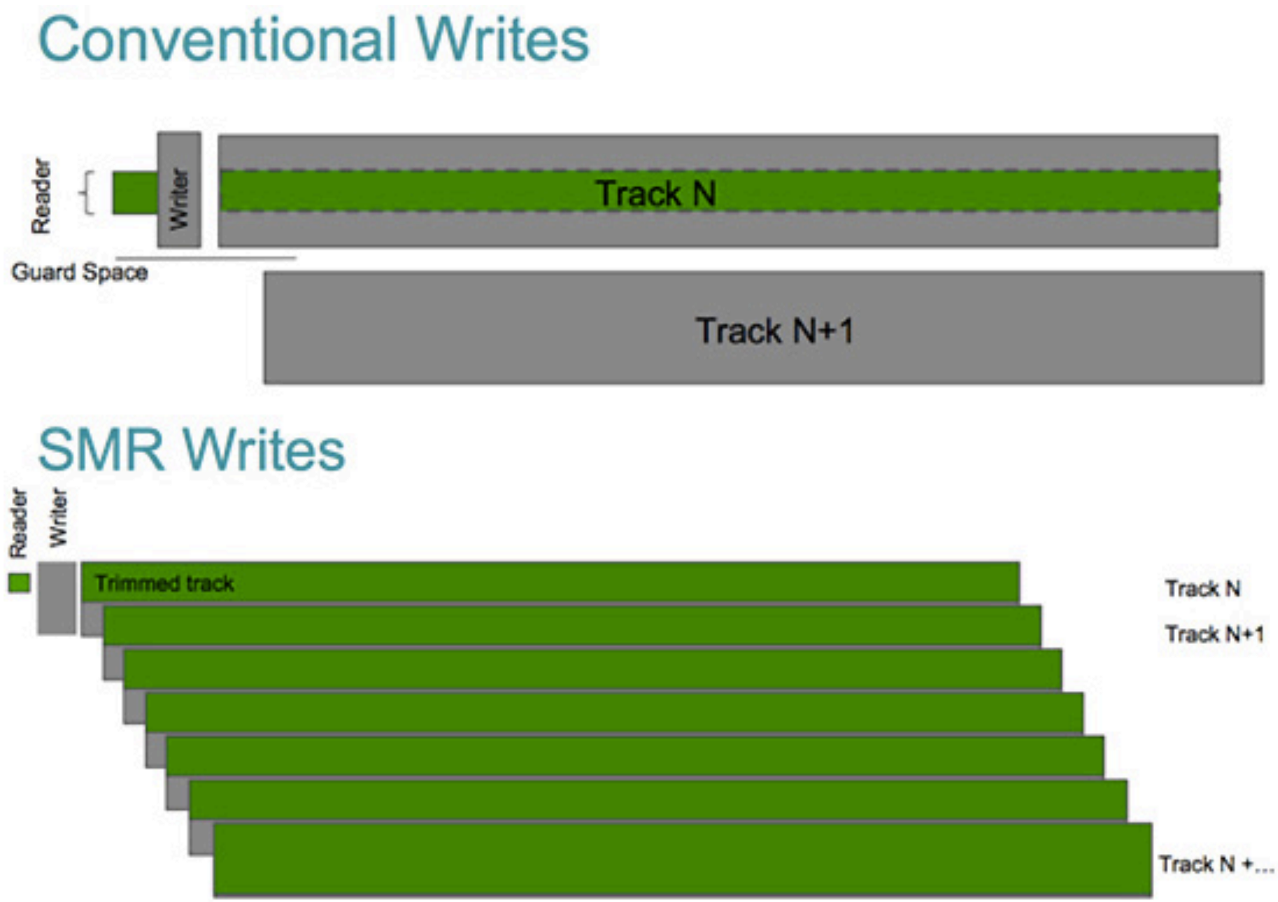
- Disk Tracks ~ 1µm (micron) wide
  - Wavelength of light is ~ 0.5µm
  - Resolution of human eye: 50µm
  - 100K tracks on a typical 2.5" disk
- Separated by unused guard regions
  - Reduces likelihood neighboring tracks are corrupted during writes (still a small non-zero chance)

The Amazing Magnetic Disk

- Track length varies across disk
  - Outside: More sectors per track, higher bandwidth
  - Disk is organized into regions of tracks with same # of sectors/track
  - Only outer half of radius is used
    - » Most of the disk area in the outer regions of the disk
- Disks so big that some companies (like Google) reportedly only use part of disk for active data
  - Rest is archival data



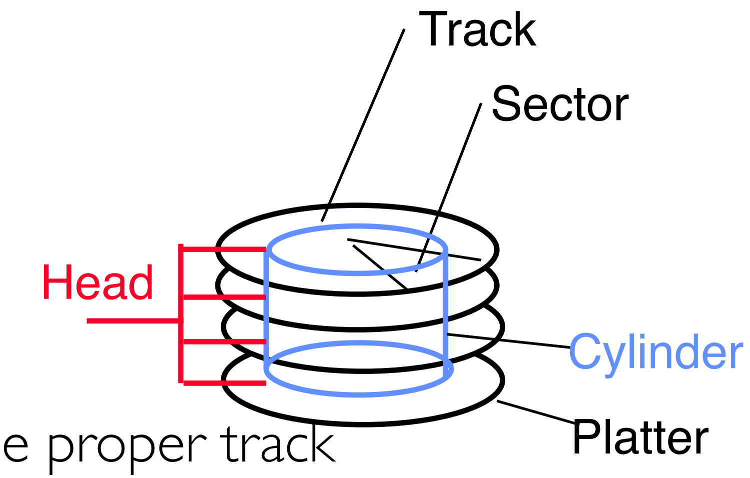
Shingled Magnetic Recording (SMR)



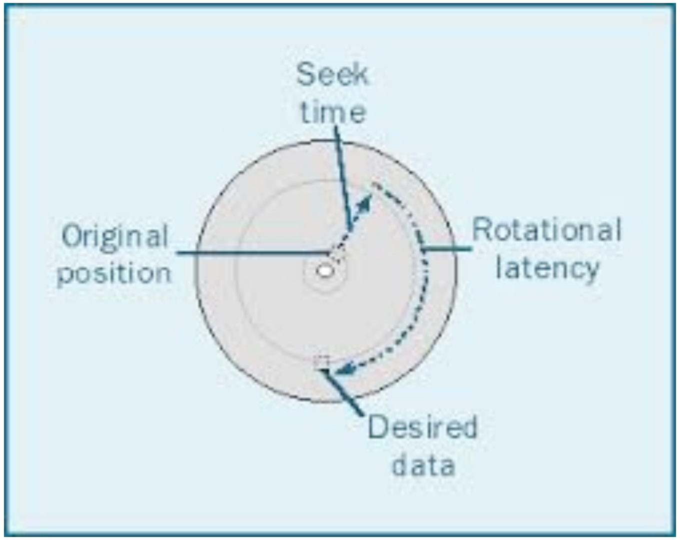
- Overlapping tracks yields greater density, capacity
- Restrictions on writing, complex DSP for reading
- Examples: Seagate (8TB), Hitachi (10TB)

Review: Magnetic Disks

- Cylinders: all the tracks under the head at a given point on all surface

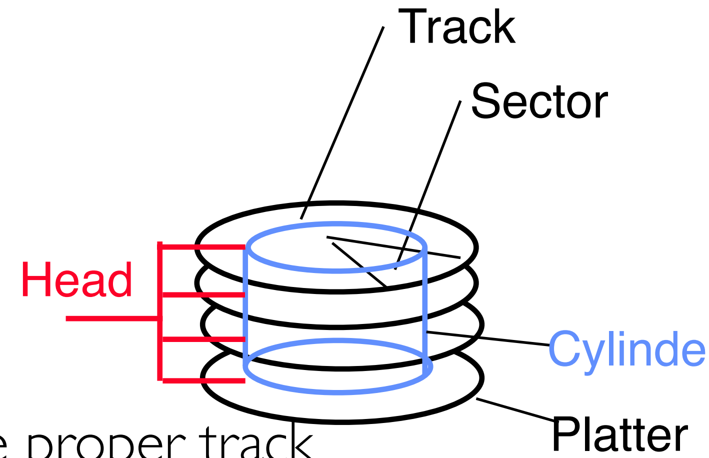


- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track
  - Rotational latency: wait for desired sector to rotate under r/w head
  - Transfer time: transfer a block of bits (sector) under r/w head



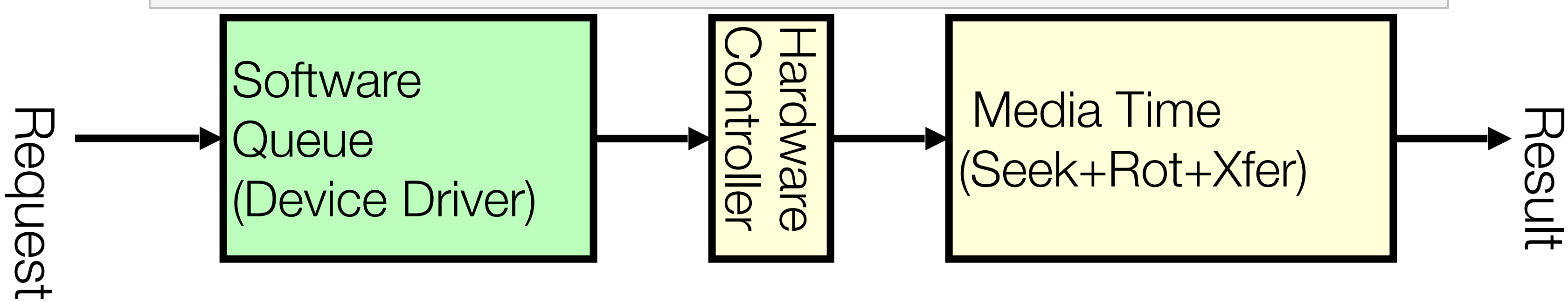
Seek time = 4-8ms
One rotation = 8-16ms
(3600-7200 RPM)

Review: Magnetic Disks



- Cylinders: all the tracks under the head at a given point on all surface
- 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

Disk Latency = Queueing Time + Controller time + Seek Time + Rotation Time + Xfer Time



Typical Numbers for Magnetic Disk

Parameter	Info / Range
Space/Density	Space: 14TB (Seagate), 8 platters, in 3½ inch form factor! Areal Density: ≥ 1Terabit/square inch! (PMR, Helium, …)
Average seek time	Typically 4-6 milliseconds. Depending on reference locality, actual cost may be 25-33% of this number.
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 8-4 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 even faster); now slowing down

Disk Performance Example

- 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
- Read block from random place on disk:
  - 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:
  - Transfer (0.082ms): 4096 bytes/0.082×10^{-3} s ≅ 50MB/sec
- Key to using disk effectively (especially for file systems) is to minimize seek and rotational delays

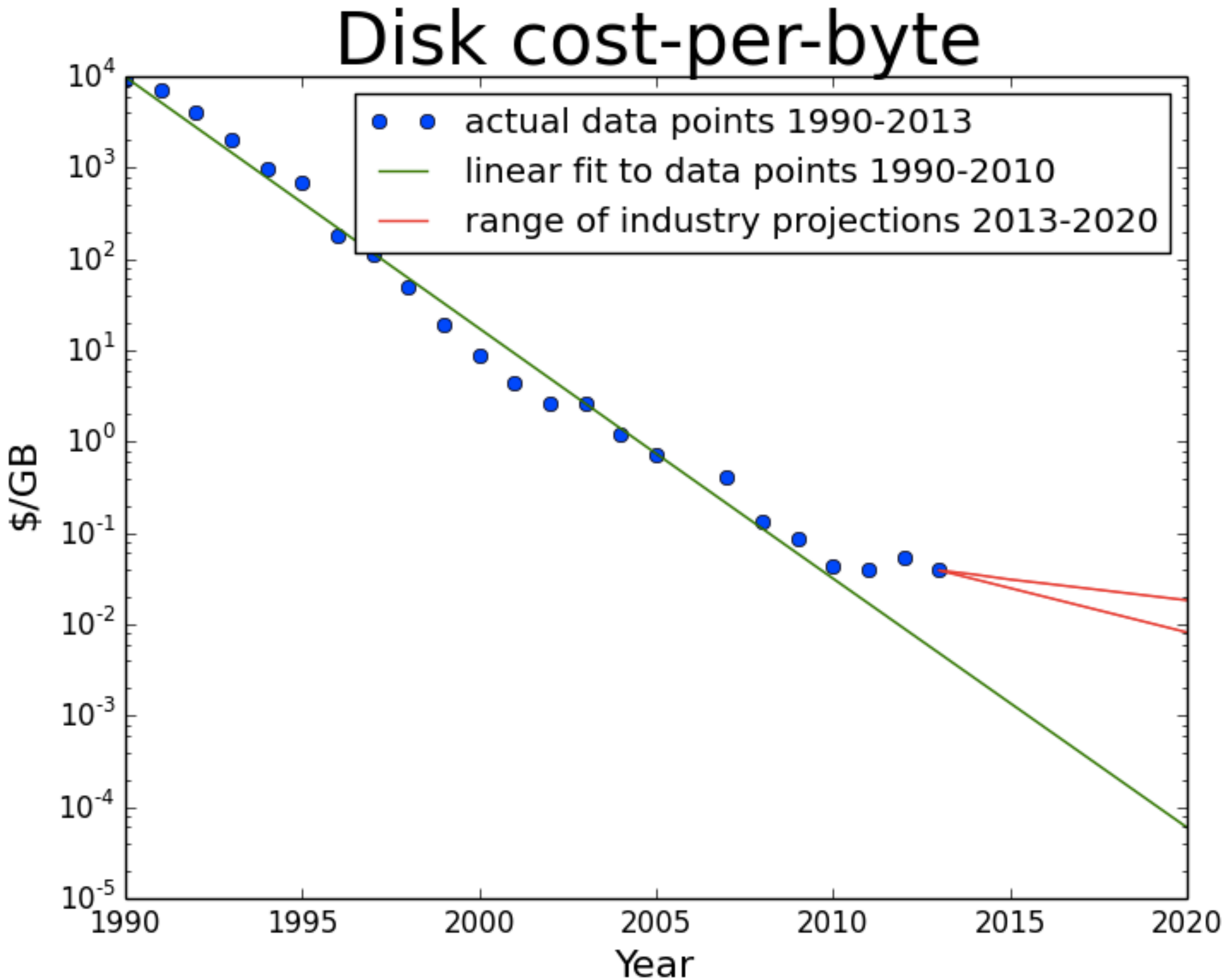
(Lots of) Intelligence in the Controller

- 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
- …

Administrative (midterm)

- You should each select a feature added in kernel 4.* and above (for example you could use: https://www.thomas-krenn.com/en/wiki/Linux_Kernel_Versions)
- email the feature to Mr. Moghaddas and CC me
- if approved, you should read more on the feature and answer the following questions:
  - Why is this feature important? what limitation/problem it resolves?
  - how does it work? including a good level of detail
  - describe everything in a 2 page document (farsi) with pictures and code when necessary (markdown)
  - provide a 5-7 min presentation on the topic
- Feature selection/submission Ordibehesht 3rd
- Report submission Ordibehesht 15th
- Presentation TBD

Hard Drive Prices over Time

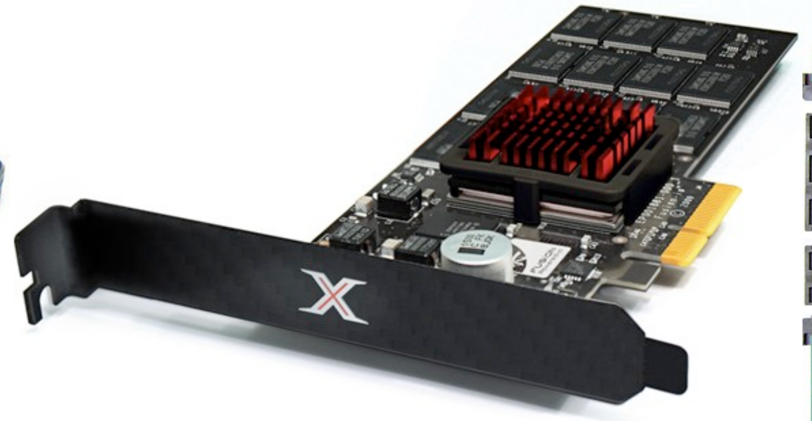


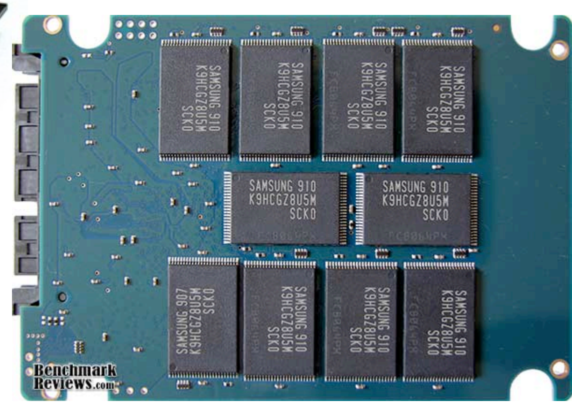
Example of Current HDDs

- Seagate Exos X14 (2018)
  - 14 TB hard disk
    - » 8 platters, 16 heads
    - » Helium filled: reduce friction and power
  - 4.16ms average seek time
  - 4096 byte physical sectors
  - 7200 RPMs
  - 6 Gbps SATA /12Gbps SAS interface
    - » 261MB/s MAX transfer rate
    - » Cache size: 256MB
  - Price: $615 (< $0.05/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 !!)

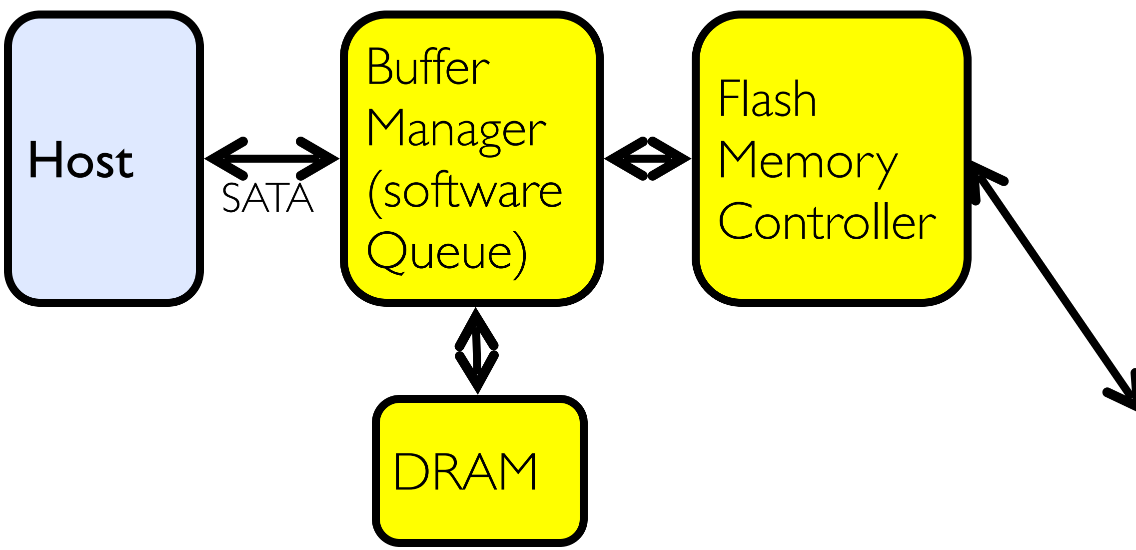
Solid State Disks (SSDs)

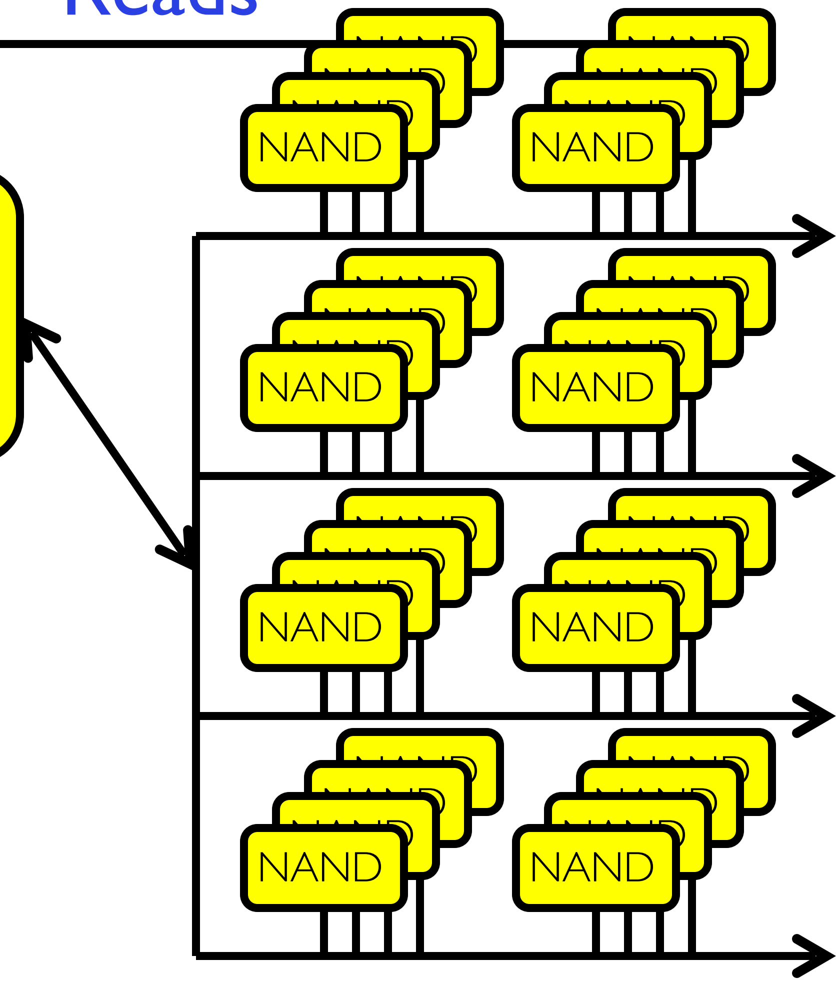




- 1995 – Replace rotating magnetic media with non-volatile memory (battery backed DRAM)
- 2009 – Use NAND Multi-Level Cell (2 or 3-bit/cell) flash memory
  - Sector (4 KB page) addressable, but stores 4-64 "pages" per memory block
  - Trapped electrons distinguish between 1 and 0
- No moving parts (no 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!

SSD Architecture – Reads



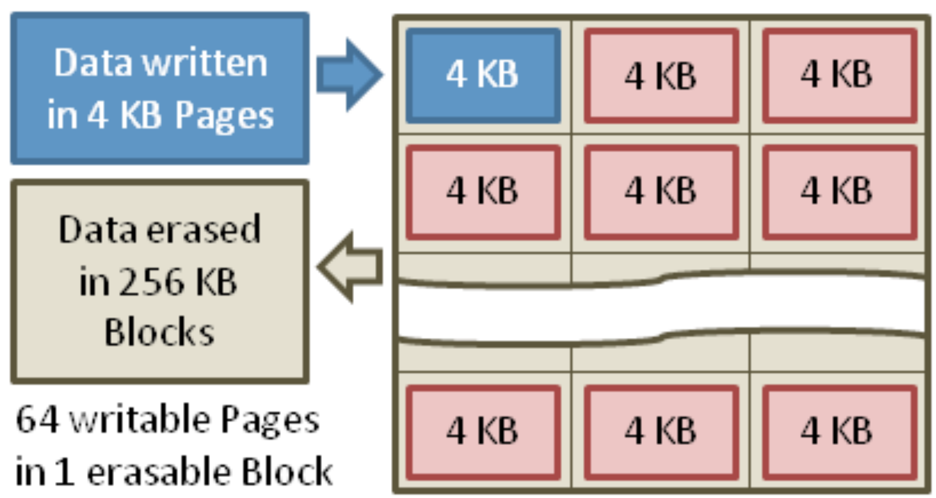


Read 4 KB Page: ~25 usec

- No seek or rotational latency
- Transfer time: transfer a 4KB page
  - SATA: 300-600MB/s => ~4 x10^3 b / 400 x 10^6 bps => 10 us
- Latency = Queuing Time + Controller time + Xfer Time
- Highest Bandwidth: Sequential OR Random reads

SSD Architecture – Writes

- Writing data is complex! (~200μs – 1.7ms)
  – Can only write empty pages in a block
  – Erasing a block takes ~1.5ms
  – Controller maintains pool of empty blocks by coalescing used pages (read, erase, write), also reserves some % of capacity
- Rule of thumb: writes 10x reads, erasure 10x writes



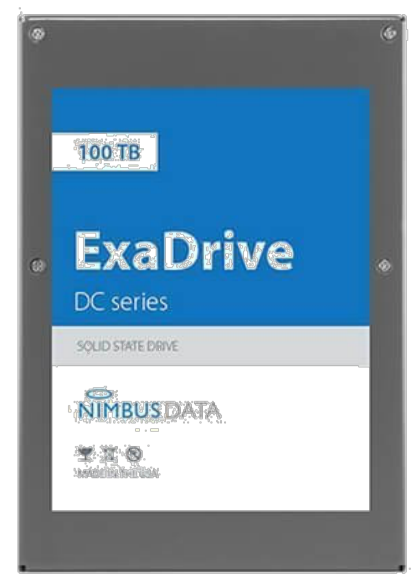
Typical NAND Flash Pages and Blocks

https://en.wikipedia.org/wiki/Solid-state_drive

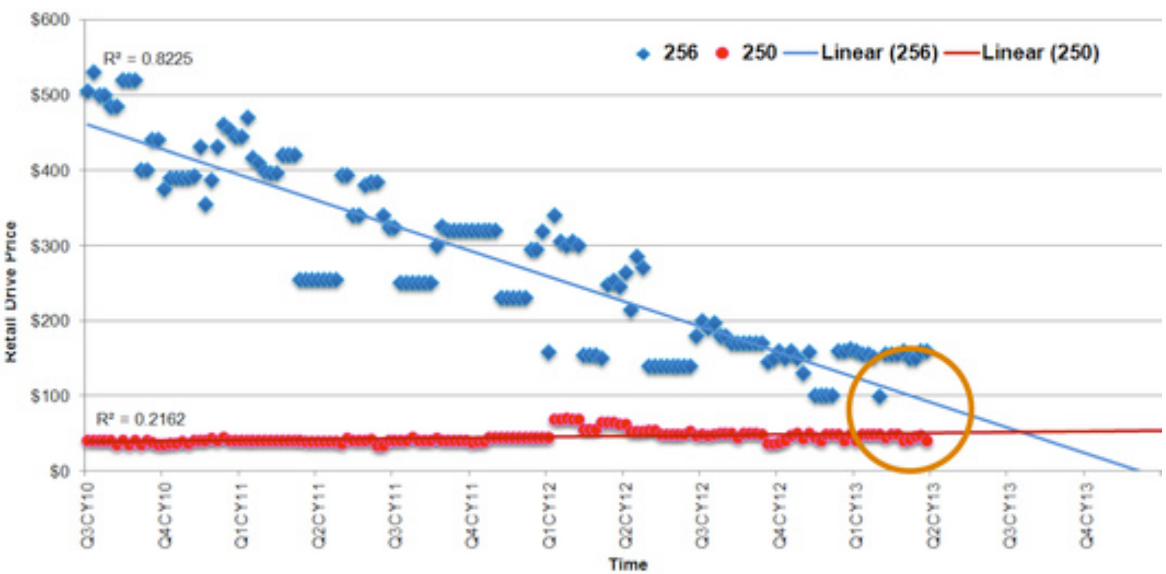
Some "Current" 3.5in SSDs

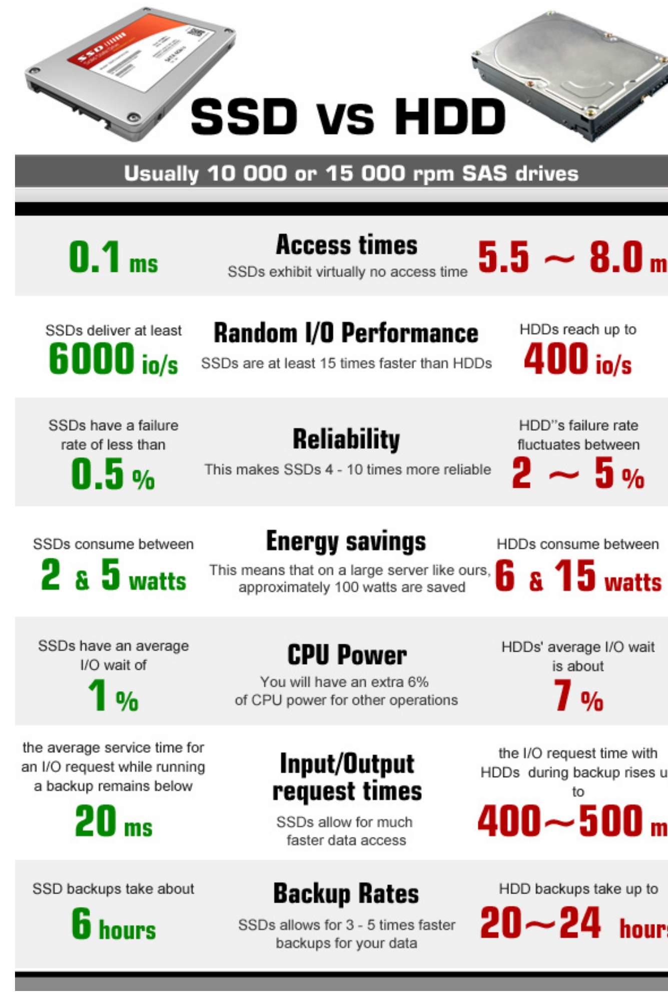
- Seagate Nytro SSD: 15TB (2017)
  - Dual 12Gb/s interface
  - Seq reads 860MB/s
  - Seq writes 920MB/s
  - Random Reads (IOPS): 102K
  - Random Writes (IOPS): 15K
  - Price (Amazon): $6325 ($0.41/GB)

- Nimbus SSD: 100TB (2019)
  - Dual port: 12Gb/s interface
  - Seq reads/writes: 500MB/s
  - Random Read Ops (IOPS): 100K
  - *Unlimited writes for 5 years!*
  - Price: ~ $50K? ($0.50/GB)



HDD vs SSD Comparison





Price Crossover Point for HDD and SSD

	2012	2013	2014	2015E	2016F	2017F
HDD	0.09	0.08	0.07	0.06	0.06	0.06
2.5" SSD	0.99	0.68	0.55	0.39	0.24	0.17

SSD prices drop much faster than HDD

Amusing calculation: Is a full Kindle heavier than an empty one?

- Actually, "Yes", but not by much
- Flash works by trapping electrons:
  – So, erased state lower energy than written state
- Assuming that:
  – Kindle has 4GB flash
  – ½ of all bits in full Kindle are in high-energy state
  – High-energy state about 10^{-15} joules higher
  – Then: Full Kindle is 1 attogram (10^{-18}gram) heavier (Using $E = mc^2$)
- Of course, this is less than most sensitive scale can measure (it can measure 10^{-9} grams)
- Of course, this weight difference overwhelmed by battery discharge, weight from getting warm, ….
- Source: John Kubiatowicz (New York Times, Oct 24, 2011)

SSD Summary

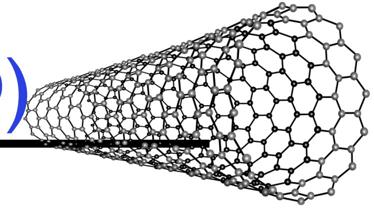
- Pros (vs. hard disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - » Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O bus)
- Cons
  - Small storage (0.1-0.5x disk), expensive (3-20x disk)
    - » Hybrid alternative: combine small SSD with large HDD

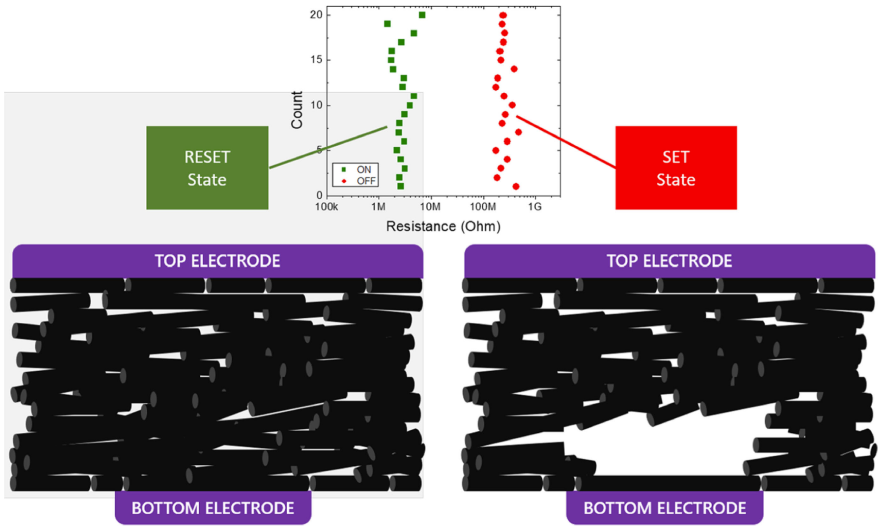
SSD Summary

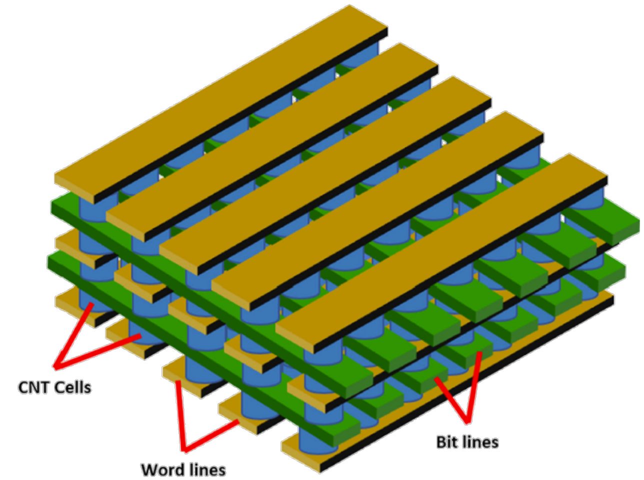
- Pros (vs. hard disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - » Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O bus)
- Cons
  - ~~Small storage (0.1-0.5x disk)~~, expensive (3-20x disk)
    - » Hybrid alternative: combine small SSD with large HDD
  - Asymmetric block write performance: read pg/erase/write pg
    - » Controller garbage collection (GC) algorithms have major effect on performance
  - Limited drive lifetime
    - » 1-10K writes/page for MLC NAND
    - » Avg failure rate is 6 years, life expectancy is 9–11 years
- These are changing rapidly!

No longer true!

Nano-Tube Memory (NANTERO)

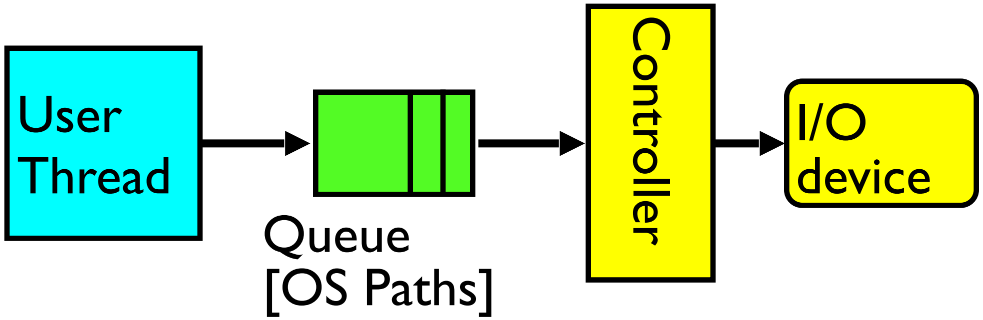


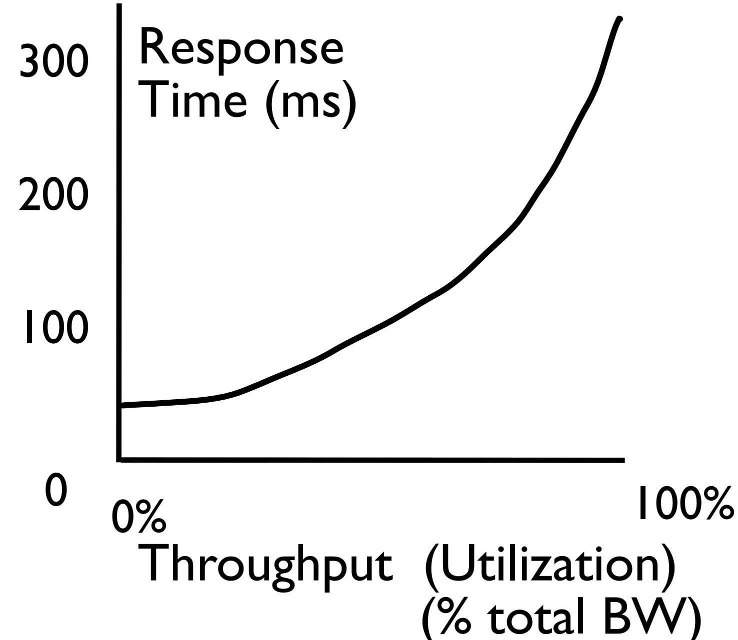




Crosspoint

- Yet another possibility: Nanotube memory
  - NanoTubes between two electrodes, slight conductivity difference between ones and zeros
  - No wearout!
- Better than DRAM?
  - Speed of DRAM, no wearout, non-volatile!
  - Nantero promises 512Gb/die for 8Tb/chip! (with 16 die stacking)





Response Time = Queue + I/O device service time

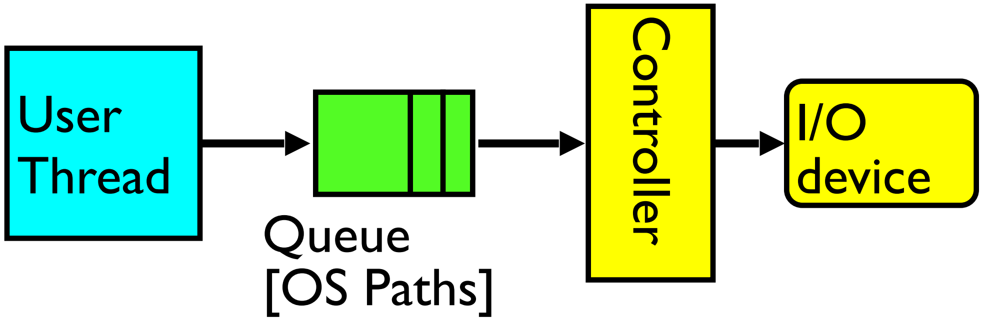
- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Effective BW per op = transfer size / response time
    - » EffBW(n) = n / (S + n/B) = B / (1 + SB/n)

# of ops

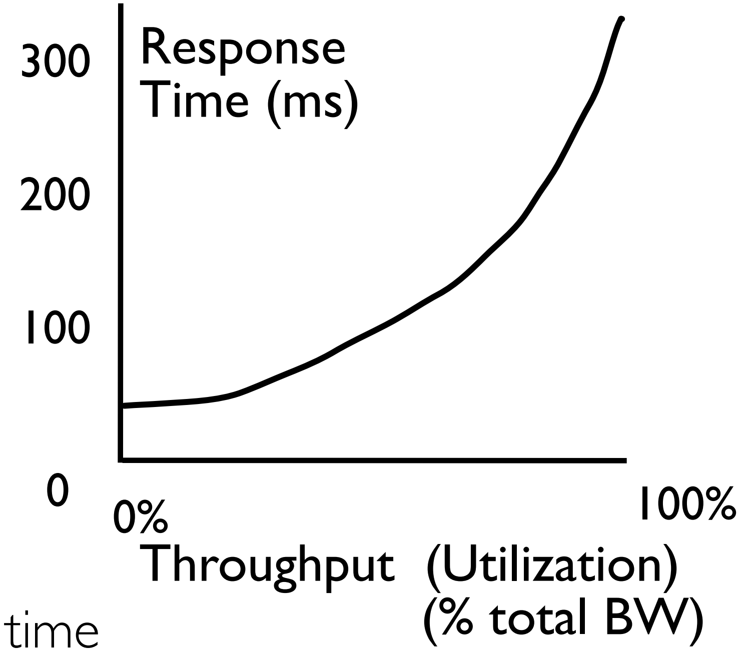
Fixed overhead

time per op

I/O Performance

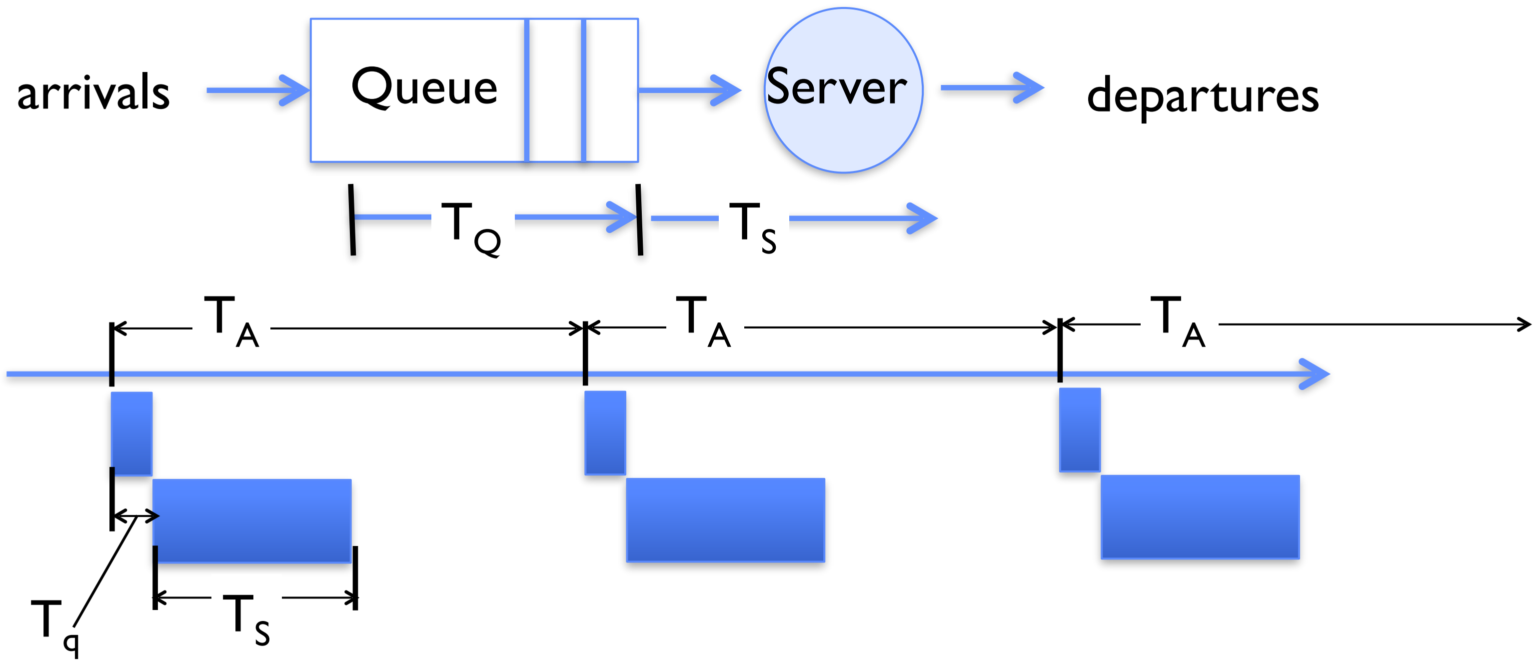


Response Time = Queue + I/O device service time



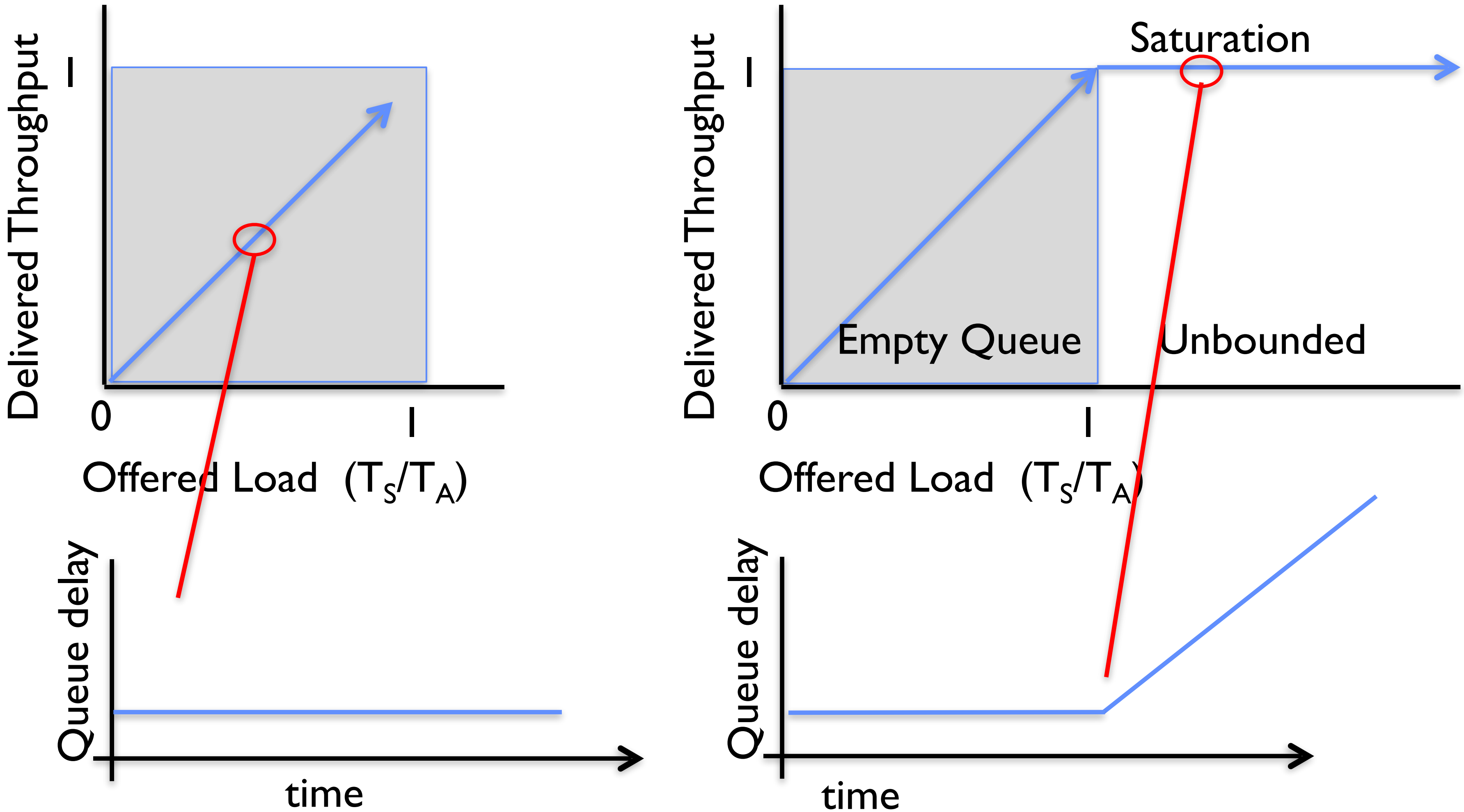
- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Effective BW per op = transfer size / response time
    - » EffBW(n) = n / (S + n/B) = B / (1 + SB/n)
  - Contributing factors to latency:
    - » Software paths (can be loosely modeled by a queue)
    - » Hardware controller
    - » I/O device service time
- Queuing behavior:
  - Can lead to big increases of latency as utilization increases
  - Solutions?

A Simple Deterministic World



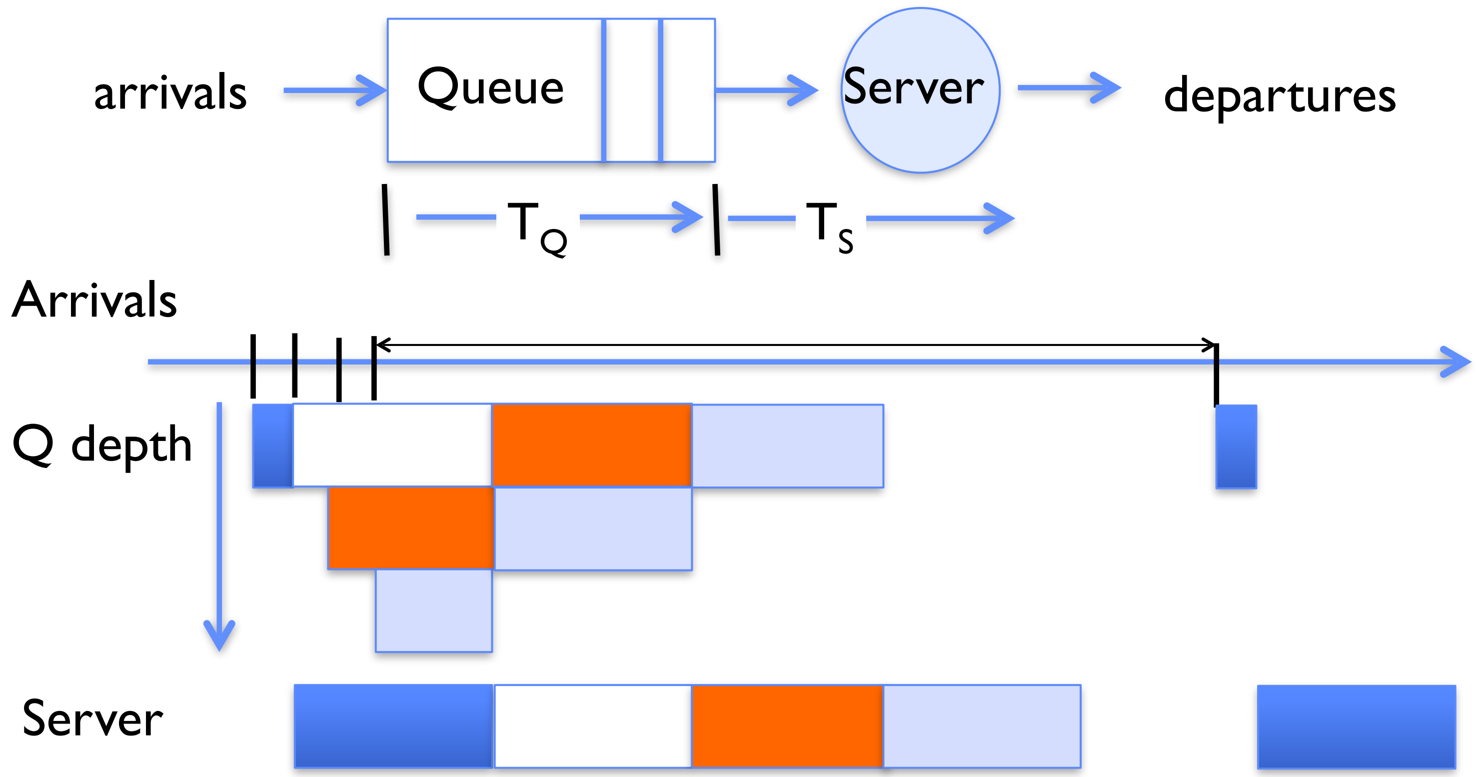
- Assume requests arrive at regular intervals, take a fixed time to process, with plenty of time between …
- Service rate ($\mu = 1/T_S$) - operations per second
- Arrival rate: ($\lambda = 1/T_A$) - requests per second
- Utilization: $U = \lambda/\mu$, where $\lambda < \mu$
- Average rate is the complete story

A Ideal Linear World



- What does the queue wait time look like?
  - Grows unbounded at a rate ~ (T_s/T_A) till request rate subsides

A Bursty World



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

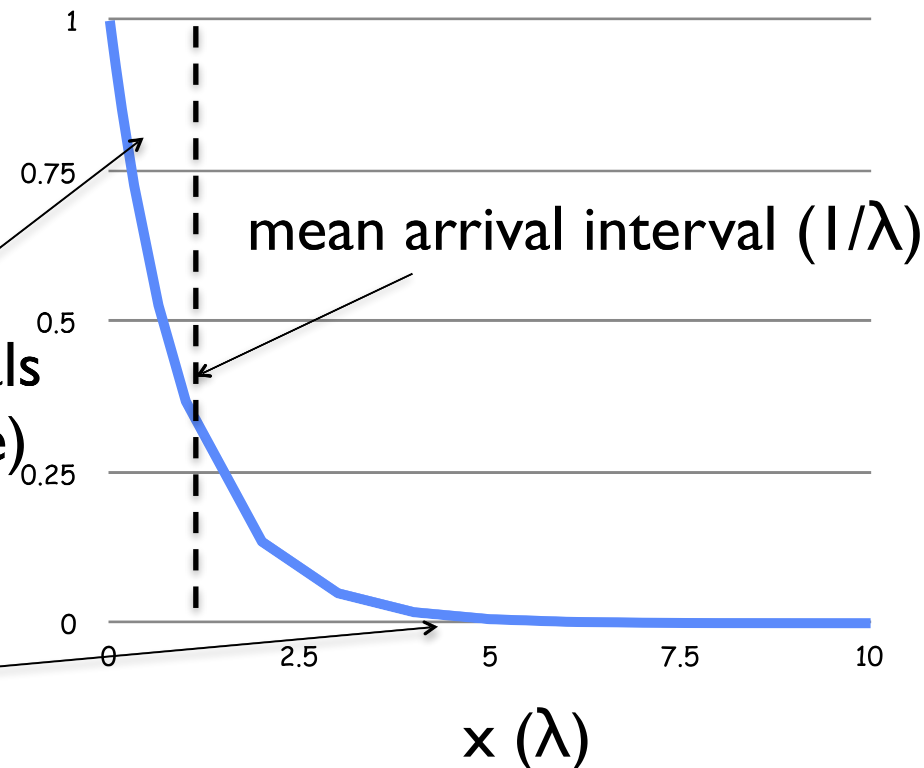
So how do we model the burstiness of arrival?

- Elegant mathematical framework if you start with *exponential distribution*
  - Probability density function of a continuous random variable with a mean of $1/\lambda$
  - $f(x) = \lambda e^{-\lambda x}$
  - *"Memoryless"*

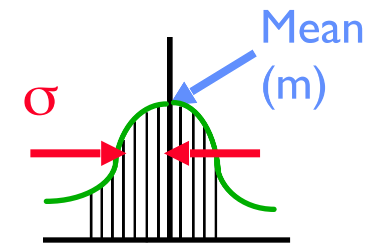
Likelihood of an event occurring is independent of how long we've been waiting

Lots of short arrival intervals (i.e., high instantaneous rate)

Few long gaps (i.e., low instantaneous rate)

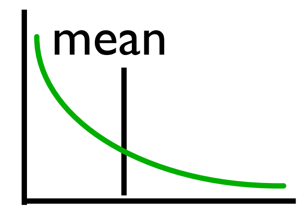


- Server spends variable time (T) with customers
  - Mean (Average) $m = \Sigma p(T) \times T$
  - Variance (stddev²) $\sigma^2 = \Sigma p(T) \times (T-m)^2 = \Sigma p(T) \times T^2 - m^2$
  - Squared coefficient of variance: $C = \sigma^2/m^2$
    Aggregate description of the distribution



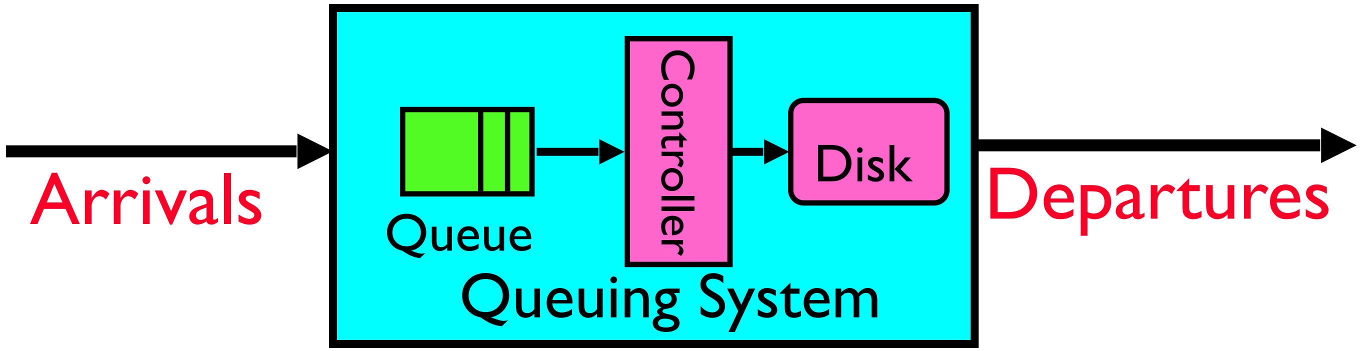
Distribution of service times

- Important values of C:
  - No variance or deterministic ⇒ C=0
  - "Memoryless" or exponential ⇒ C=1
    - » Past tells nothing about future
    - » Poisson process – *purely* or *completely* random process
    - » Many complex systems (or aggregates) are well described as memoryless
  - Disk response times C ≈ 1.5 (majority seeks < average)



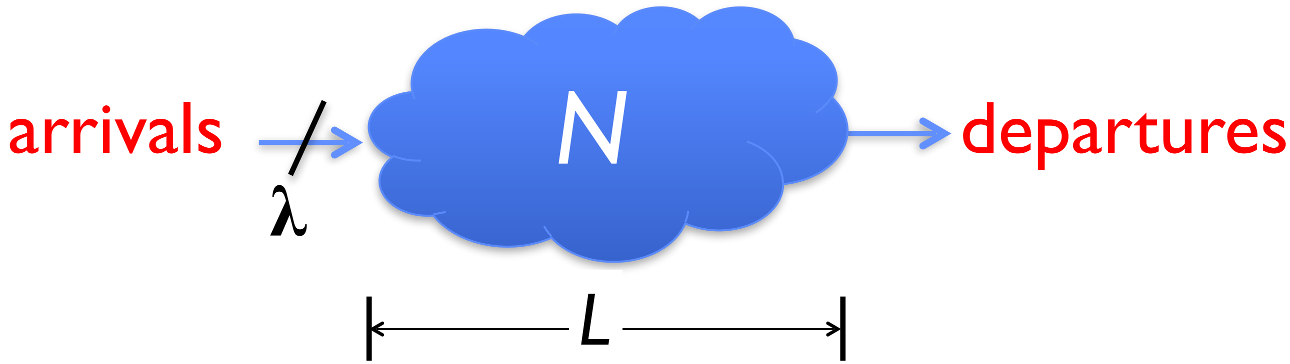
Memoryless

Introduction to Queuing Theory



- What about queuing time??
  – Let's apply some 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

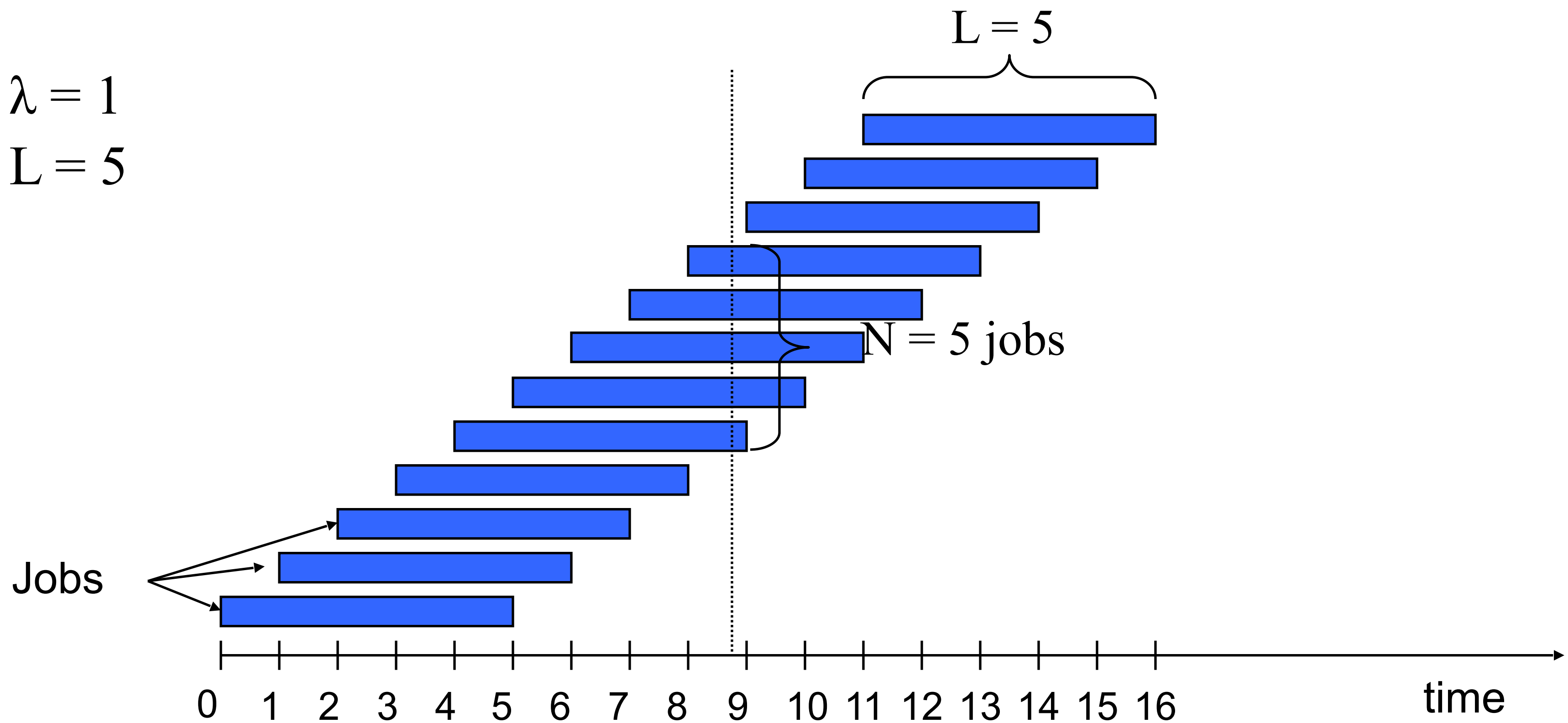


- In any *stable* system
  - Average arrival rate = Average departure rate
- The average number of jobs/tasks in the system (*N*) is equal to arrival time / throughput (λ) times the response time (*L*)
  - *N (jobs) = λ (jobs/s) x L (s)*
- Regardless of structure, bursts of requests, variation in service
  - Instantaneous variations, but it washes out in the average
  - Overall, requests match departures

Example

$\lambda = 1$

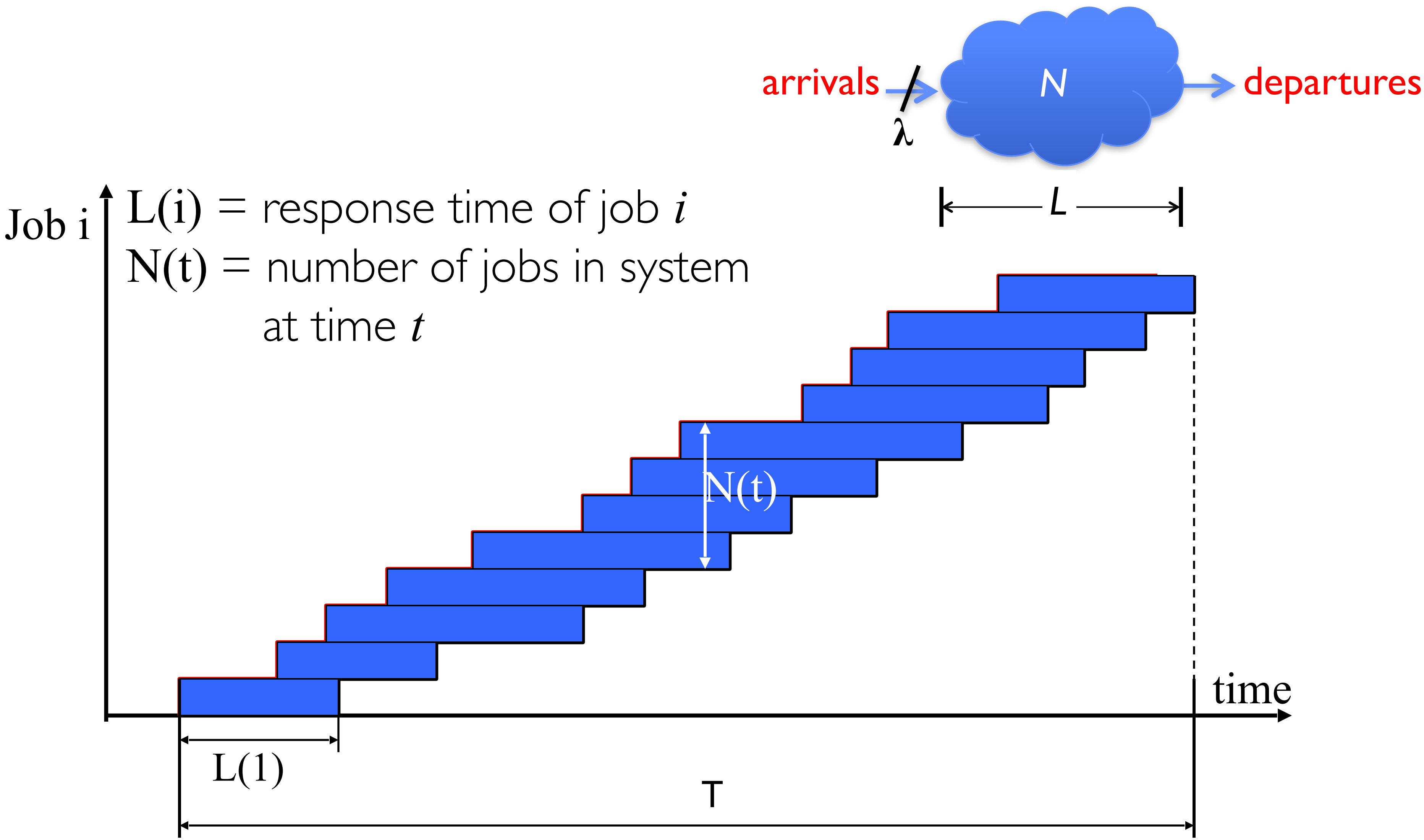
$L = 5$



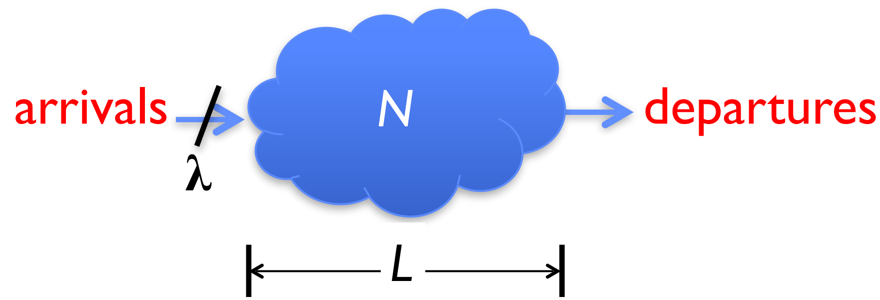
A: $N = \lambda \times L$

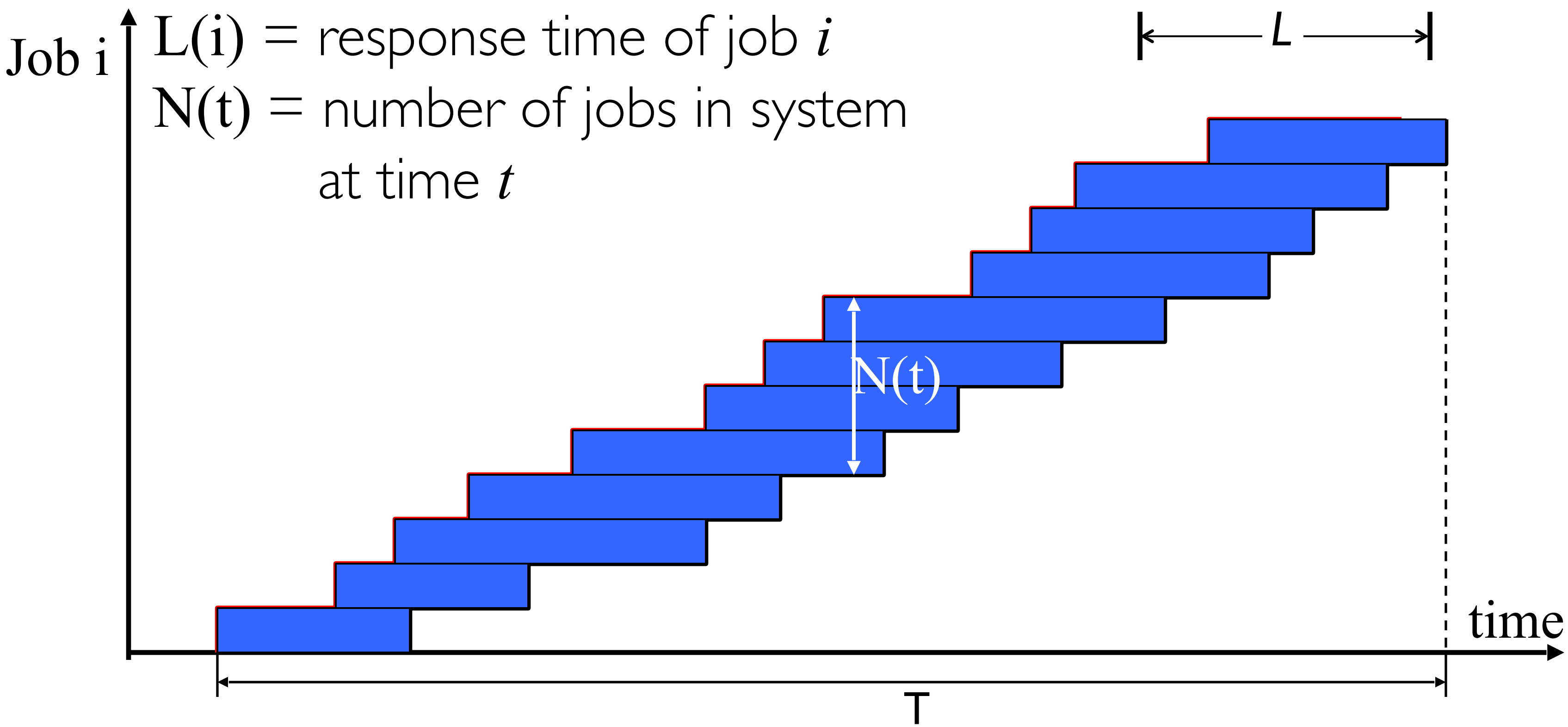
- E.g., $N = \lambda \times L = 5$

Little's Theorem: Proof Sketch



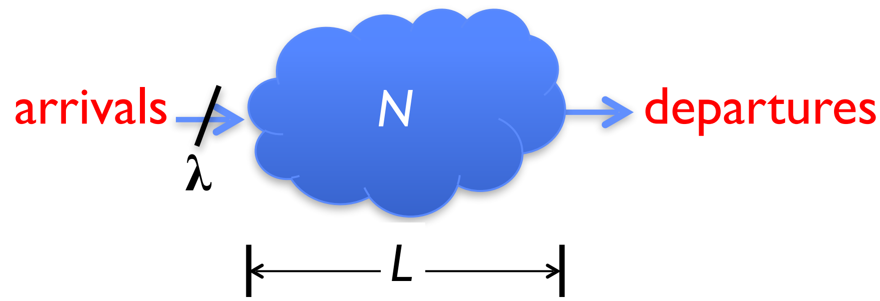
Little's Theorem: Proof Sketch





What is the system occupancy, i.e., average number of jobs in the system?

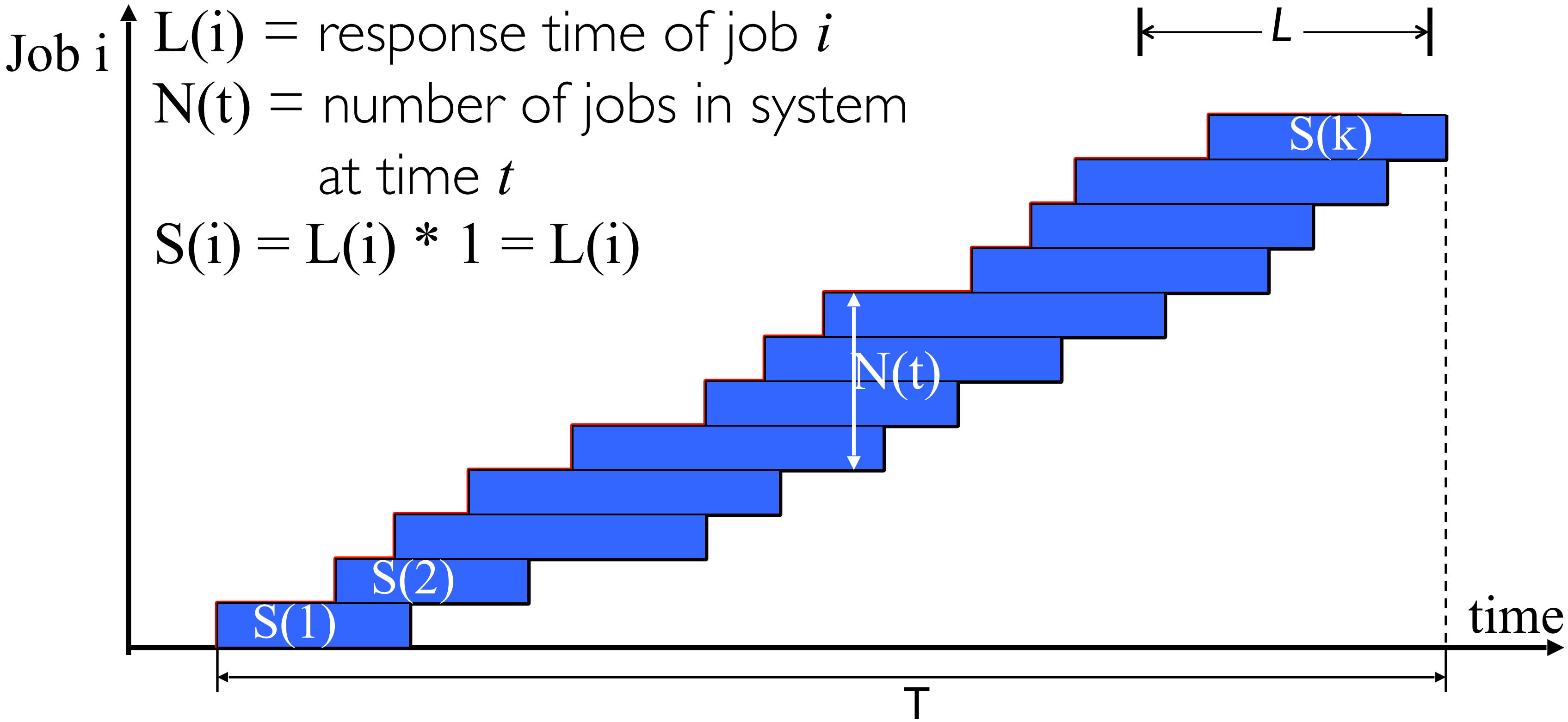
Little's Theorem: Proof Sketch



L(i) = response time of job *i*

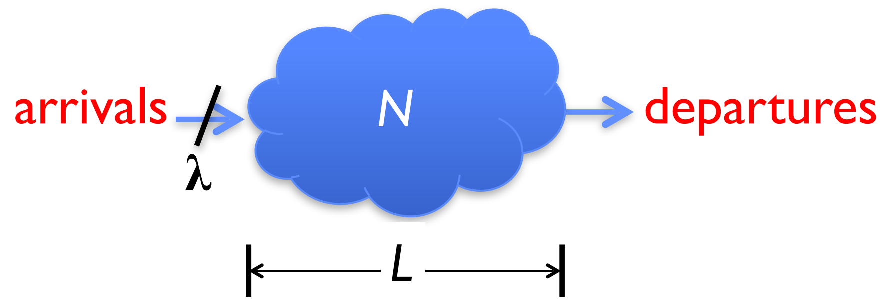
N(t) = number of jobs in system at time *t*

S(i) = L(i) * 1 = L(i)



$$S = S(1) + S(2) + \ldots + S(k) = L(1) + L(2) + \ldots + L(k)$$

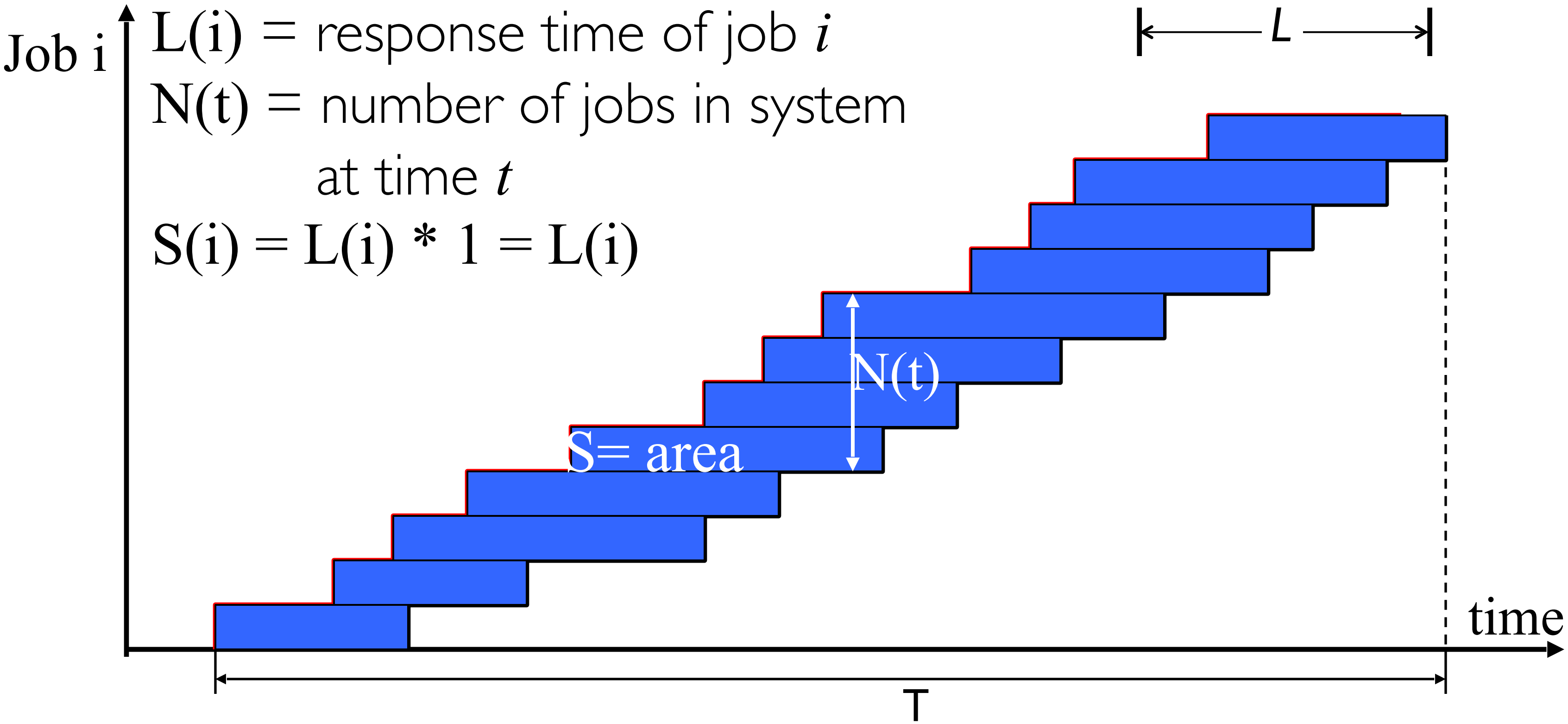
Little's Theorem: Proof Sketch



$L(i)$ = response time of job i

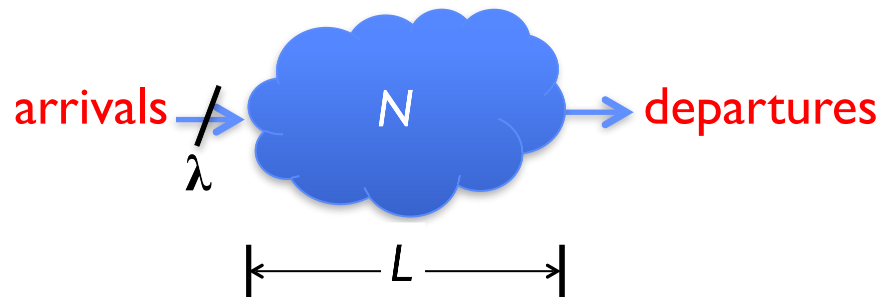
$N(t)$ = number of jobs in system at time t

$S(i) = L(i) * 1 = L(i)$



Average occupancy (N_{avg}) = S/T

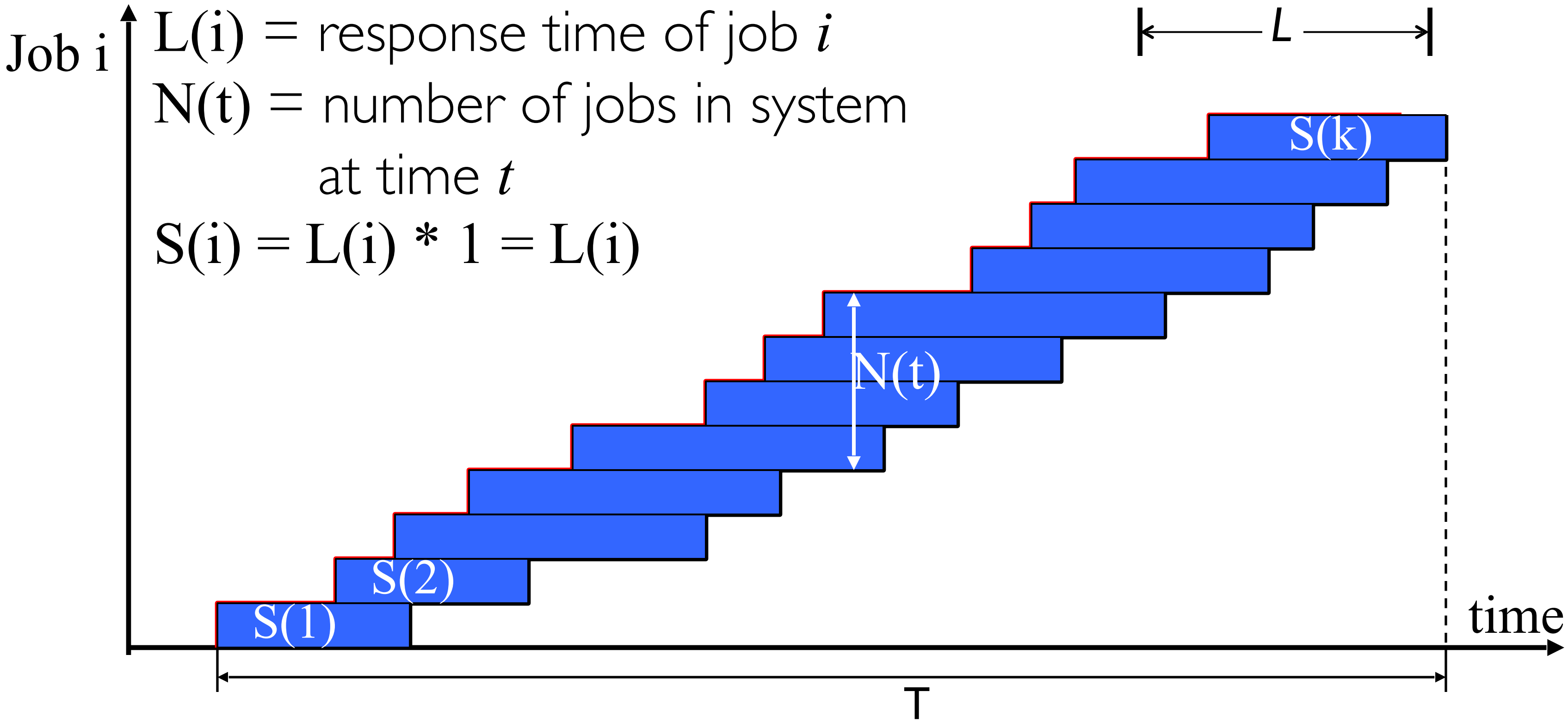
Little's Theorem: Proof Sketch



L(i) = response time of job *i*

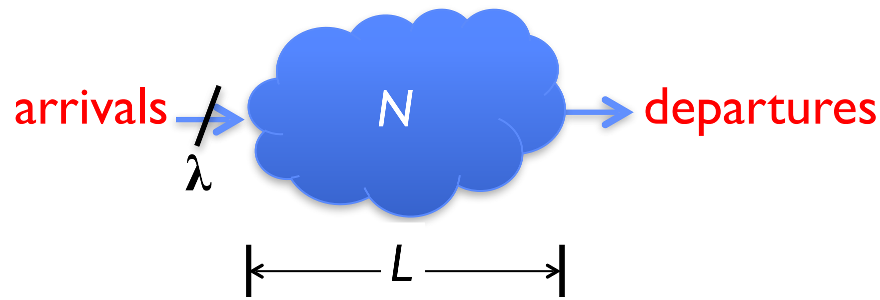
N(t) = number of jobs in system at time *t*

S(i) = L(i) * 1 = L(i)



$$N_{avg} = S/T = (L(1) + \ldots + L(k))/T$$

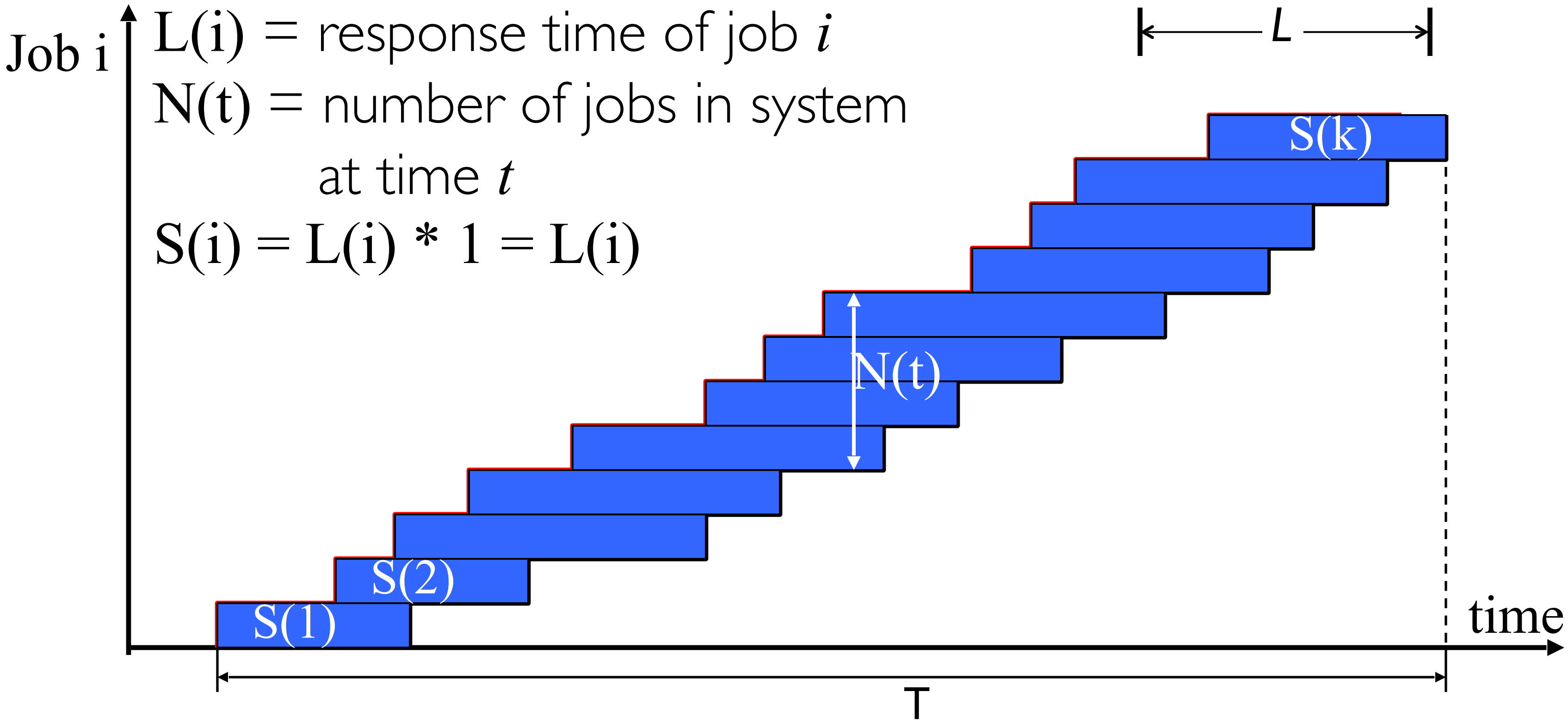
Little's Theorem: Proof Sketch



$L(i)$ = response time of job i

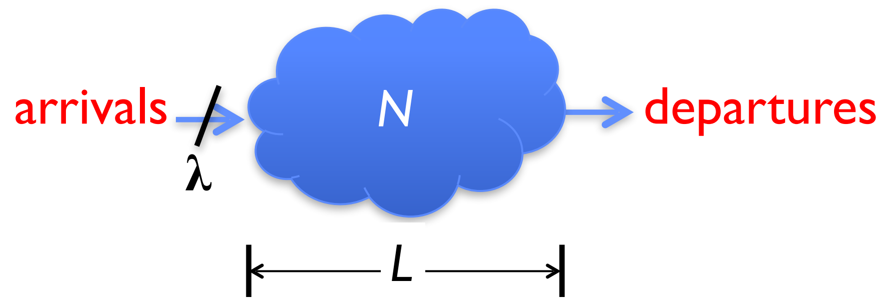
$N(t)$ = number of jobs in system at time t

$S(i) = L(i) * 1 = L(i)$



$$N_{avg} = (L(1) + \ldots + L(k))/T = (N_{total}/T)*(L(1) + \ldots + L(k))/N_{total}$$

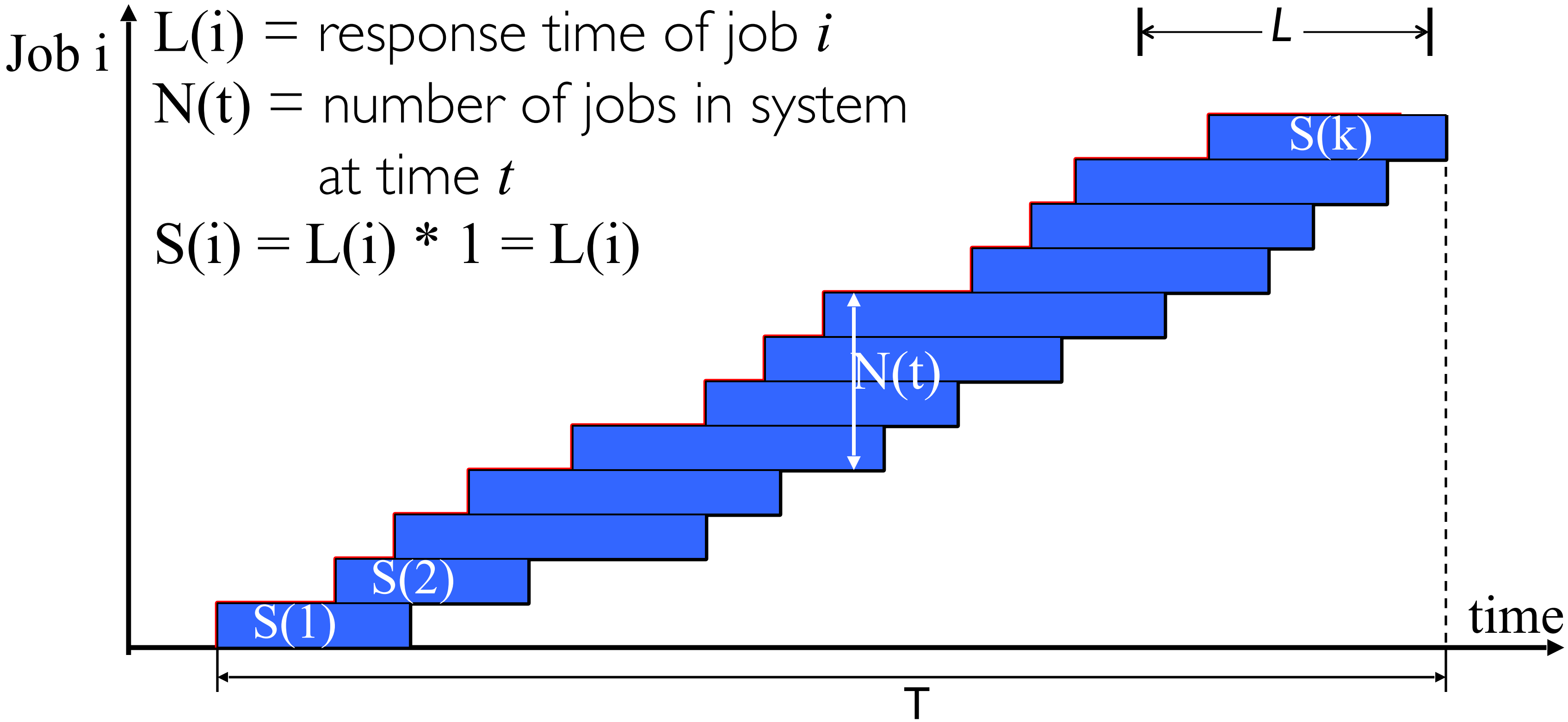
Little's Theorem: Proof Sketch



L(i) = response time of job *i*

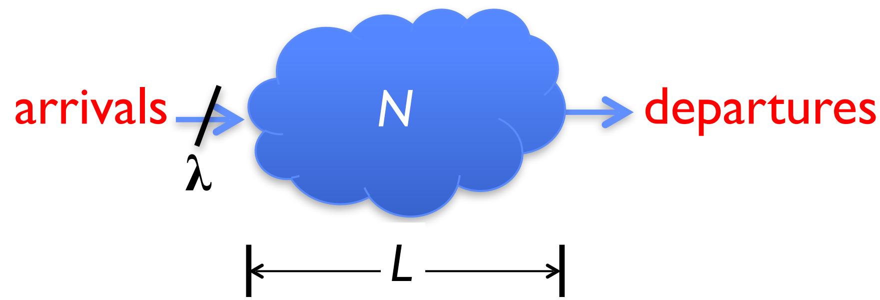
N(t) = number of jobs in system at time *t*

S(i) = L(i) * 1 = L(i)



$$N_{avg} = (N_{total}/T)*(L(1) + \ldots + L(k))/N_{total} = \lambda_{avg} \times L_{avg}$$

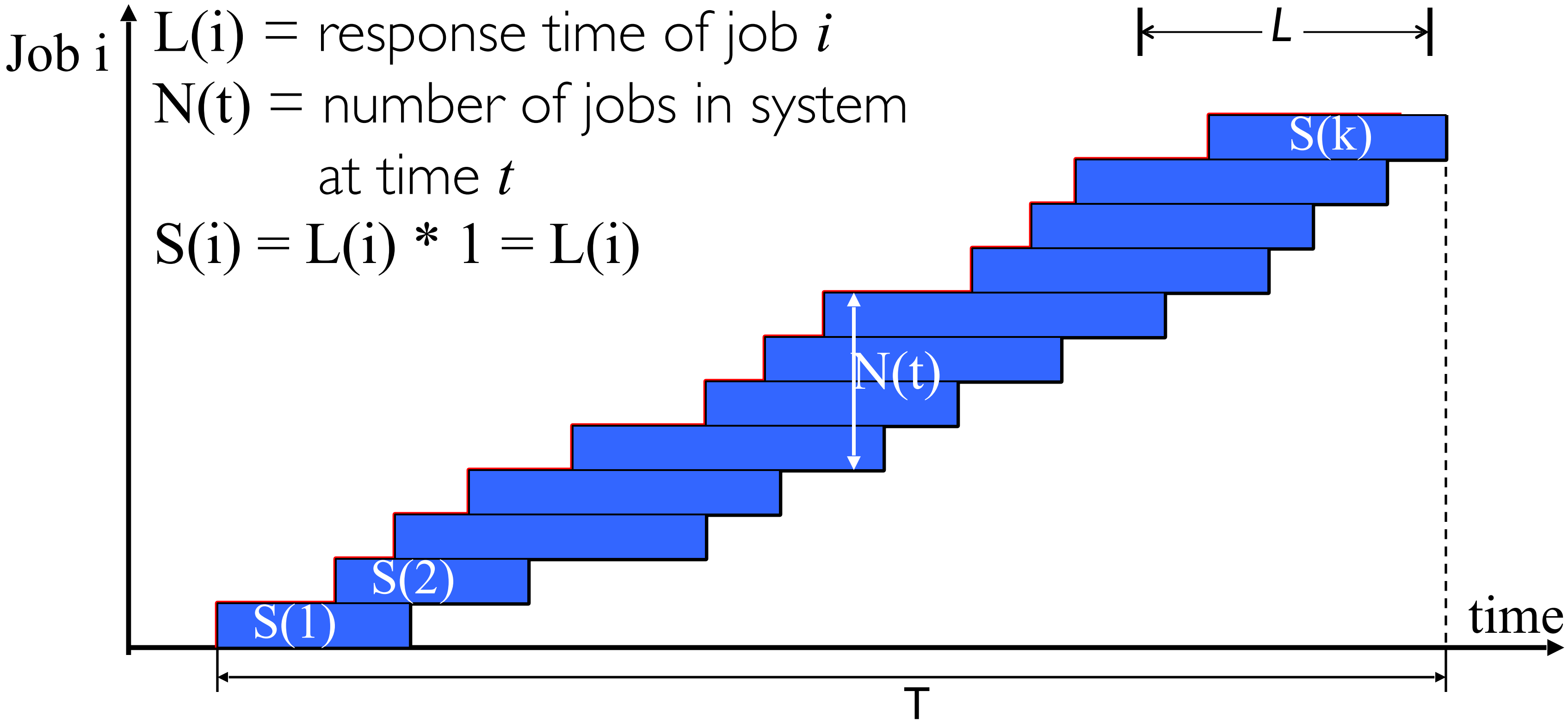
Little’s Theorem: Proof Sketch



L(i) = response time of job *i*

N(t) = number of jobs in system at time *t*

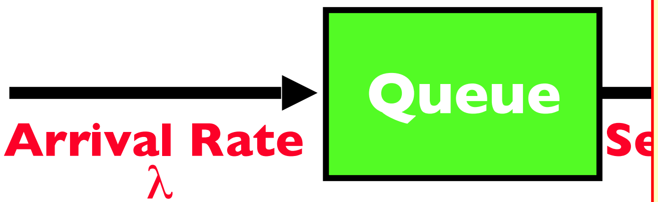
S(i) = L(i) * 1 = L(i)



$$N_{avg} = \lambda_{avg} \times L_{avg}$$

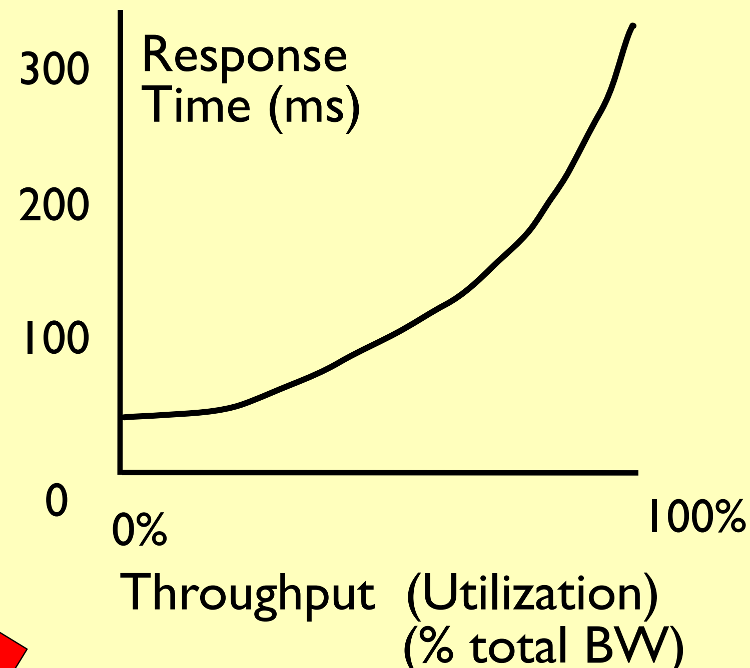
A Little Queuing Theory: Some Results

- Assumptions:
  - System in equilibrium; No limit to the q
  - Time between successive arrivals is ran



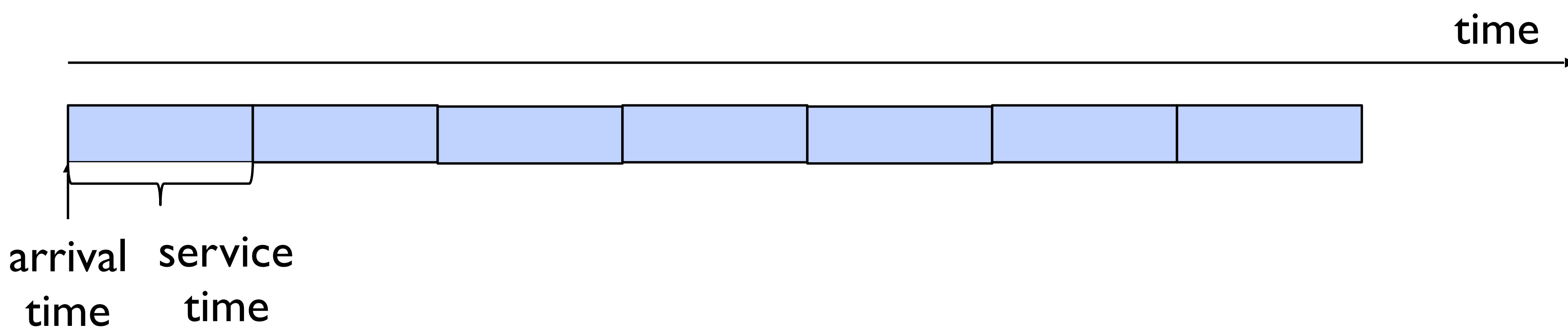
- Parameters that describe our system:
  - λ: mean number of arriving cust
  - T_{ser}: mean time to service a custo
  - C: squared coefficient of variance
  - μ: service rate = $1/T_{ser}$
  - u: server utilization ($0 \le u \le 1$): u =
- Parameters we wish to compute:
  - T_q: Time spent in queue
  - L_q: Length of queue = $\lambda \times T_q$ (by
- Results:
  - Memoryless service distribution (C = 1): (an "M/M/1 queue"):
    - » $T_q = T_{ser} \times u/(1 - u)$
  - General service distribution (no restrictions), 1 server (an "M/G/1 queue"):
    - » $T_q = T_{ser} \times \frac{1}{2}(1+C) \times u/(1 - u)$

Why does response/queueing delay grow unboundedly even though the utilization is < 1 ?



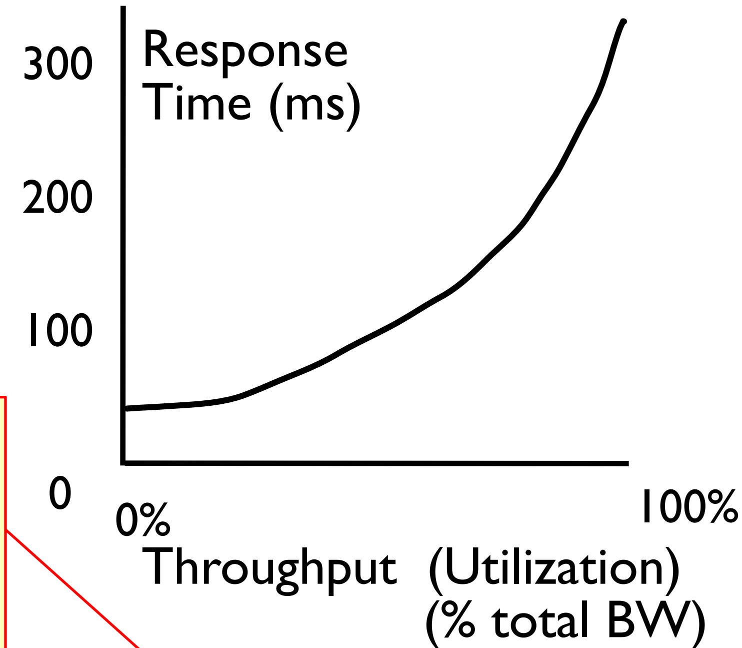
Why unbounded response time?

- Assume deterministic arrival process and service time
  – Possible to sustain utilization = 1 with bounded response time!

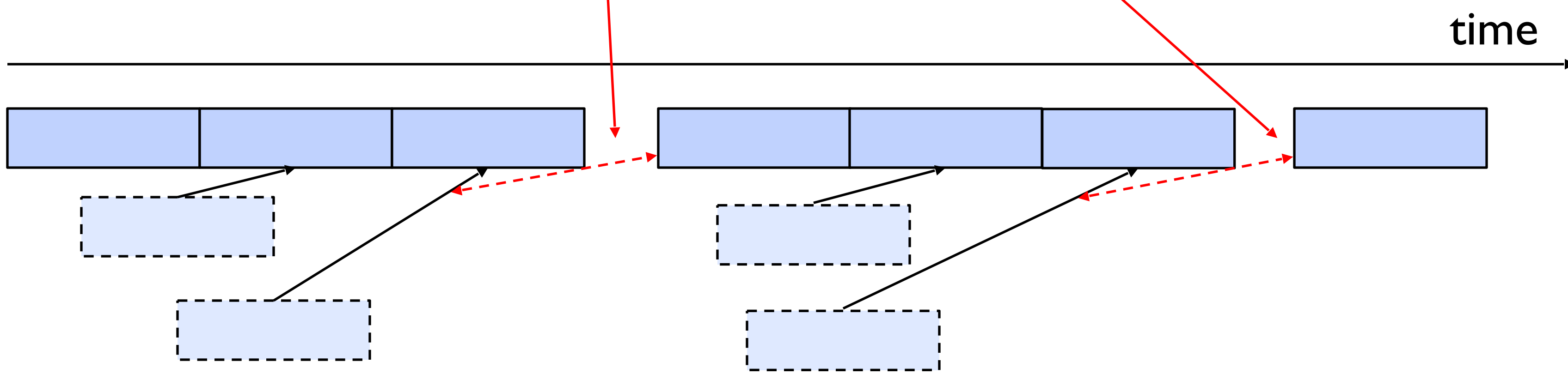


Why unbounded response time?

- Assume stochastic arrival process (and service time)
  – No longer possible to achieve utilization = 1



This wasted time can never be reclaimed!
So cannot achieve u = 1!



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, $u = \lambda T_{ser}$
  - 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)

u (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

Queuing Theory Resources

- Resources page contains Queueing Theory Resources (under Readings):
  - Scanned pages from Patterson and Hennessy book that gives further discussion and simple proof for general equation: https://cs162.eecs.berkeley.edu/static/readings/patterson_queue.pdf
  - A complete website full of resources: http://web2.uwindsor.ca/math/hlynka/qonline.html
- Some previous midterms with queueing theory questions
- Assume that Queueing Theory is fair game for Midterm III!

Summary

- Disk Performance:
  - Queuing time + Controller + Seek + Rotational + Transfer
  - Rotational latency: on average ½ rotation
  - Transfer time: spec of disk depends on rotation speed and bit storage density
- Devices have complex interaction and performance characteristics
  - Response time (Latency) = Queue + Overhead + Transfer
    - » Effective BW = BW * T/(S+T)
  - HDD: Queuing time + controller + seek + rotation + transfer
  - SDD: Queuing time + controller + transfer (erasure & wear)
- Systems (e.g., file system) designed to optimize performance and reliability
  - Relative to performance characteristics of underlying device
- Bursts & High Utilization introduce queuing delays
- Queuing Latency:
  - M/M/1 and M/G/1 queues: simplest to analyze
  - As utilization approaches 100%, latency → ∞

$$T_q = T_{ser} \times \frac{1}{2}(1+C) \times u/(1 - u))$$