CS162
Operating Systems and Systems Programming
Lecture 17

Performance
Storage Devices, Queueing Theory

October 24, 2018

Prof. Ion Stoica

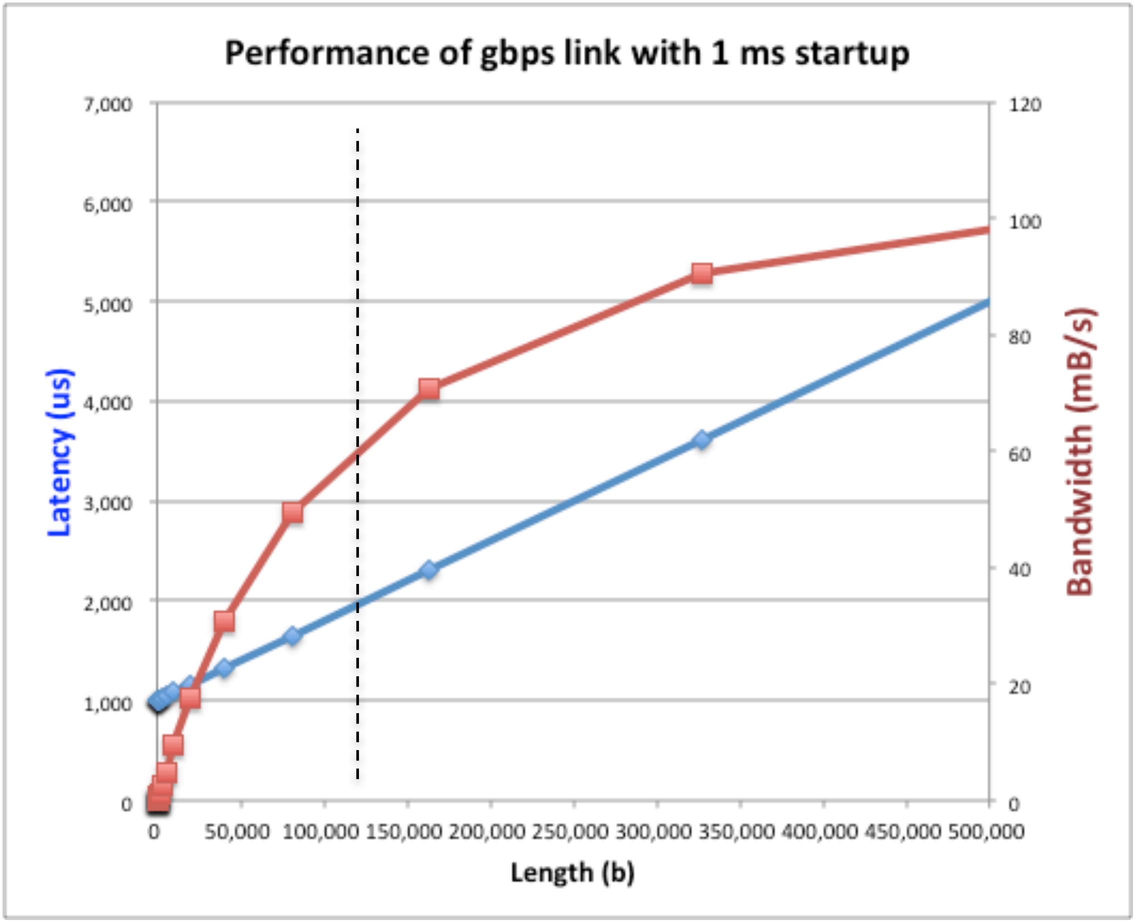
http://cs162.eecs.Berkeley.edu

Review: Basic Performance Concepts

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

Example (Fast Network)

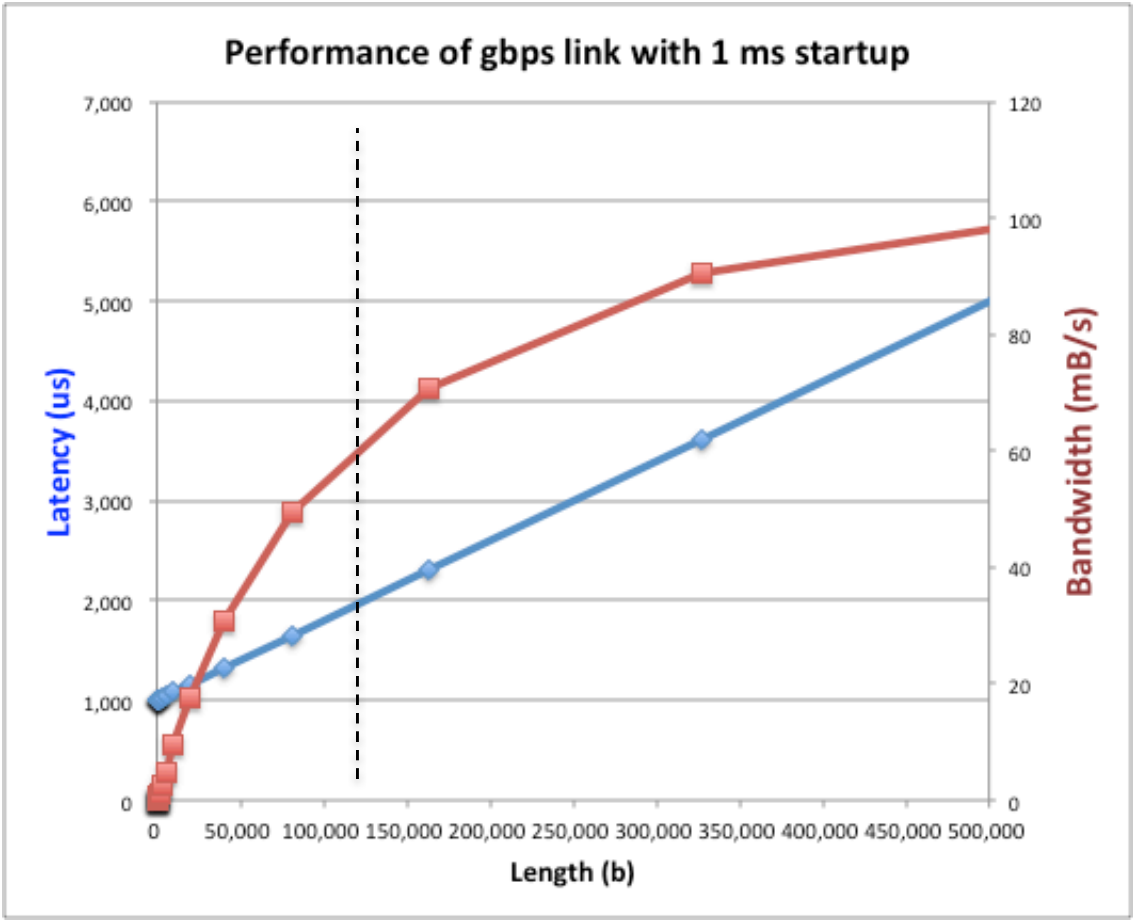
- Consider a 1 Gb/s link (B = 125 MB/s)
  - With a startup cost S = 1 ms



  - Latency(n) = S + n/B
  - Bandwidth = n/(S + n/B) = B*n/(B*S + n) = B/(B*S/n + 1)

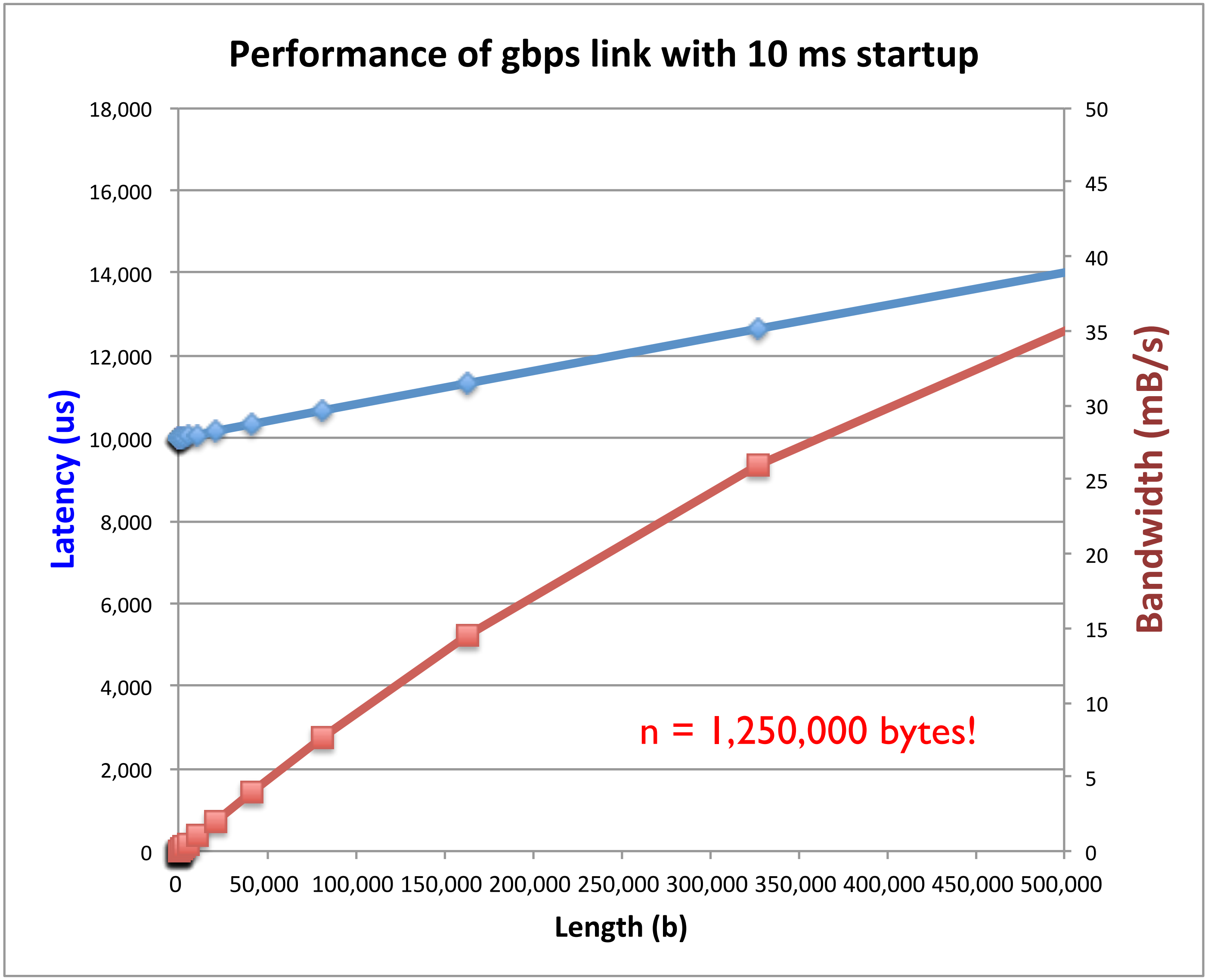
Example (Fast Network)

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



  - Bandwidth = B/(B*S/n + 1)
  - half-power point occurs at n=S*B → Bandwidth = B/2

Example: at 10 ms startup (like Disk)



n = 1,250,000 bytes!

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 Attached SCSI & Serial ATA & IEEE 1394 (firewire): 1.6 Gb/s full duplex (200 MB/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…

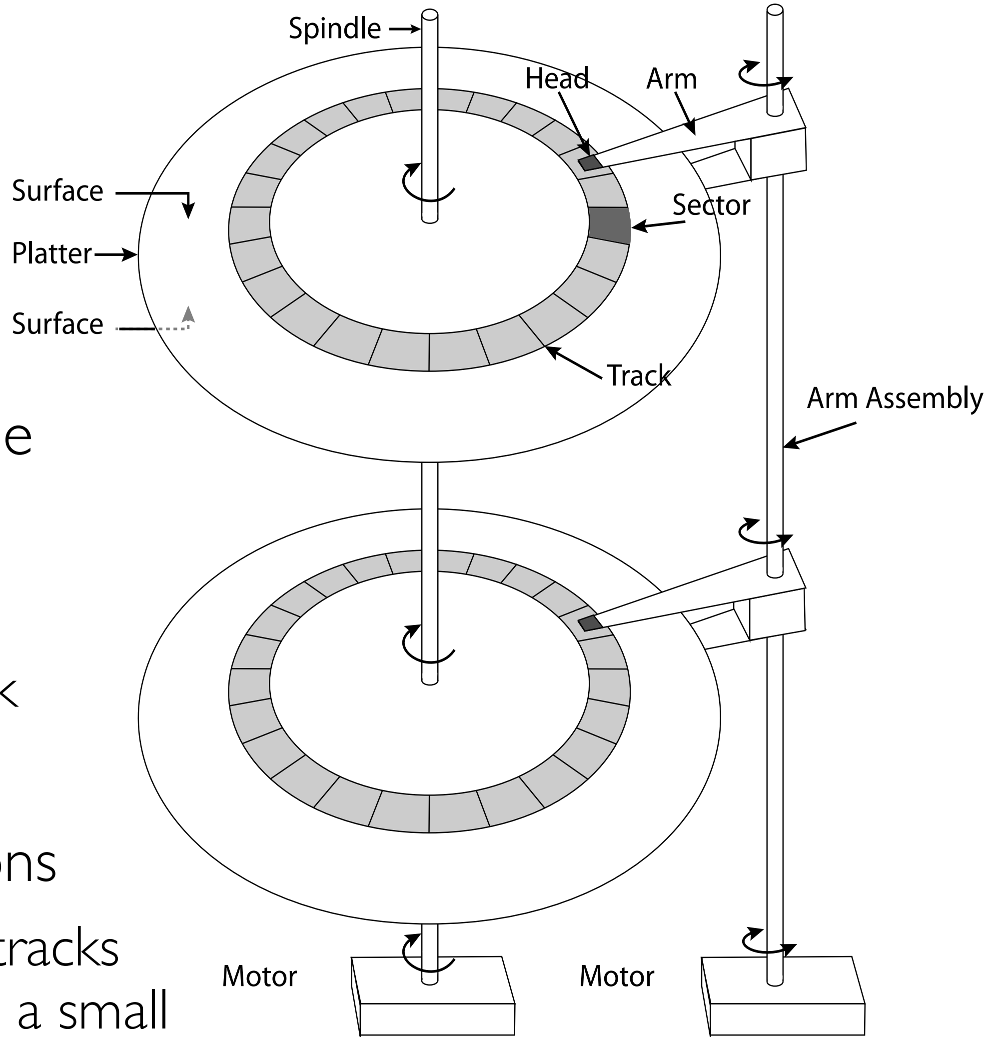
Storage Devices

- 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

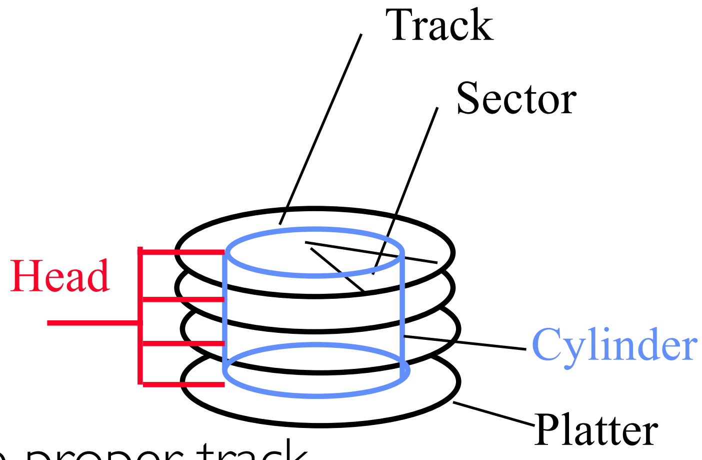
- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (5-20x disk)
  - Block level random access
  - Good performance for reads; worse for random writes
  - Erasure requirement in large blocks
  - Wear patterns issue

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)

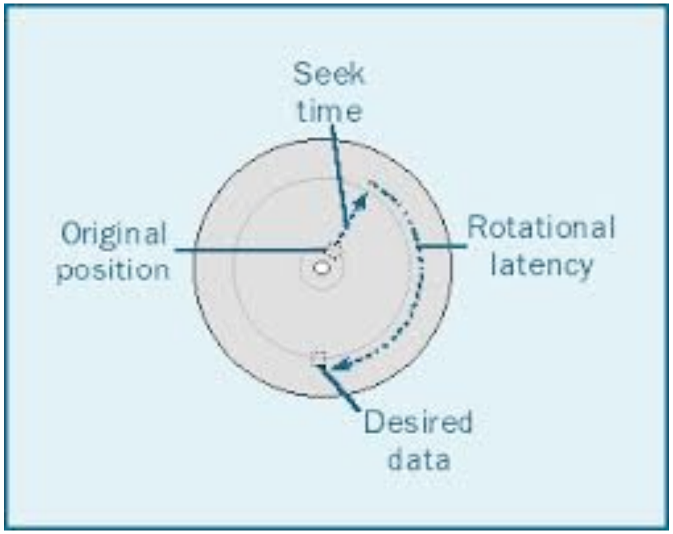


Review: Magnetic Disks



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

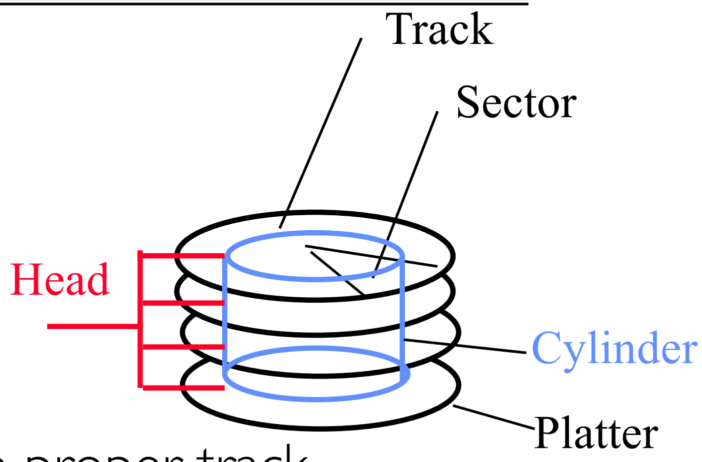
- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track
  - Rotational latency: wait for desired sector to rotate under r/w head
  - Transfer time: transfer a block of bits (sector) under r/w head



Seek time = 4-8ms
One rotation = 1-2ms
(3600-7200 RPM)

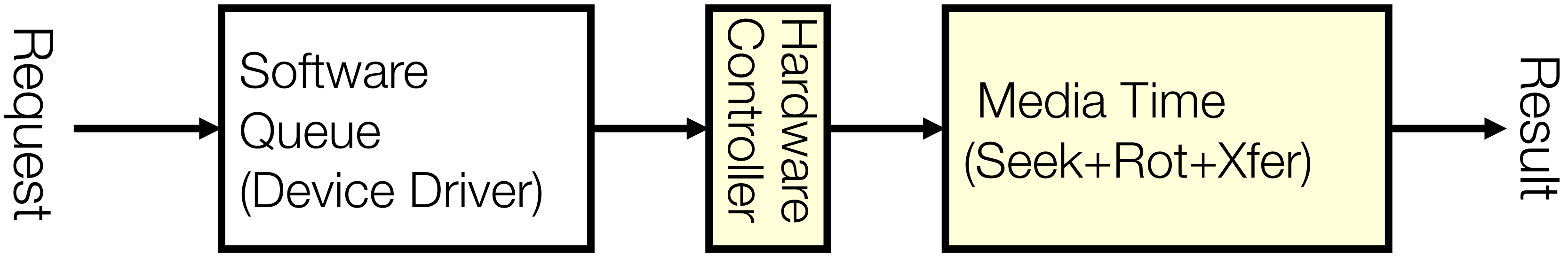
Review: Magnetic Disks

- 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


Request

Software Queue (Device Driver)

Hardware Controller

Media Time (Seek+Rot+Xfer)

Result

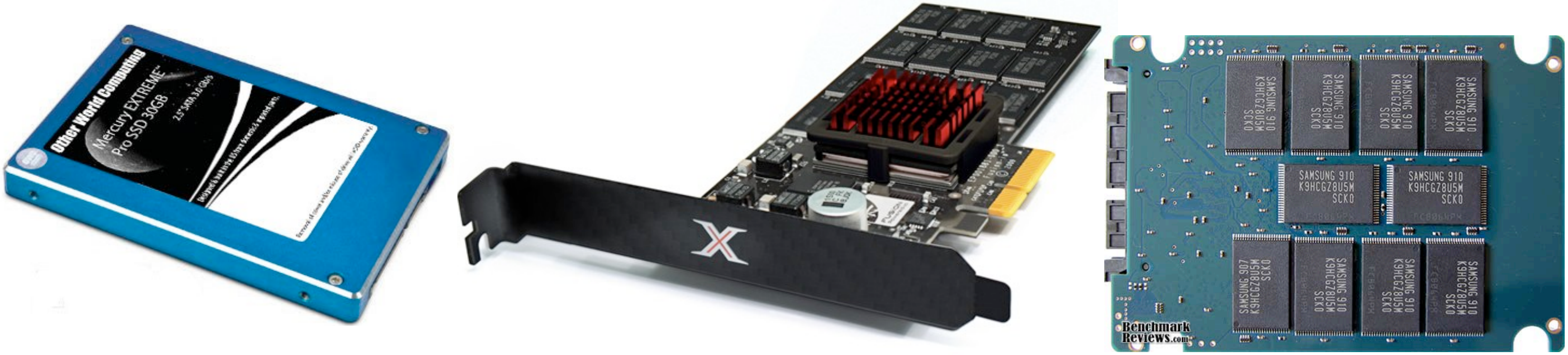
Disk Performance Example

- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms,
  - 7200RPM ⇒ Time for rotation: 60000 (ms/minute) / 7200(rev/min) ~= 8ms
  - Transfer rate of 4MByte/s, sector size of 1 Kbyte ⇒ $1024 \text{ bytes}/4\times10^6 \text{ (bytes/s)} = 256 \times 10^{-6} \text{ sec} \cong .26 \text{ ms}$
- Read sector from random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.26ms)
  - *Approx* 10ms to fetch/put data: **100 KByte/sec**
- Read sector from random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.26ms)
  - *Approx* 5ms to fetch/put data: **200 KByte/sec**
- Read next sector on same track:
  - Transfer (0.26ms): **4 MByte/sec**
- Key to using disk effectively (especially for file systems) is to *minimize seek and rotational delays*

(Lots of) Intelligence in the Controller

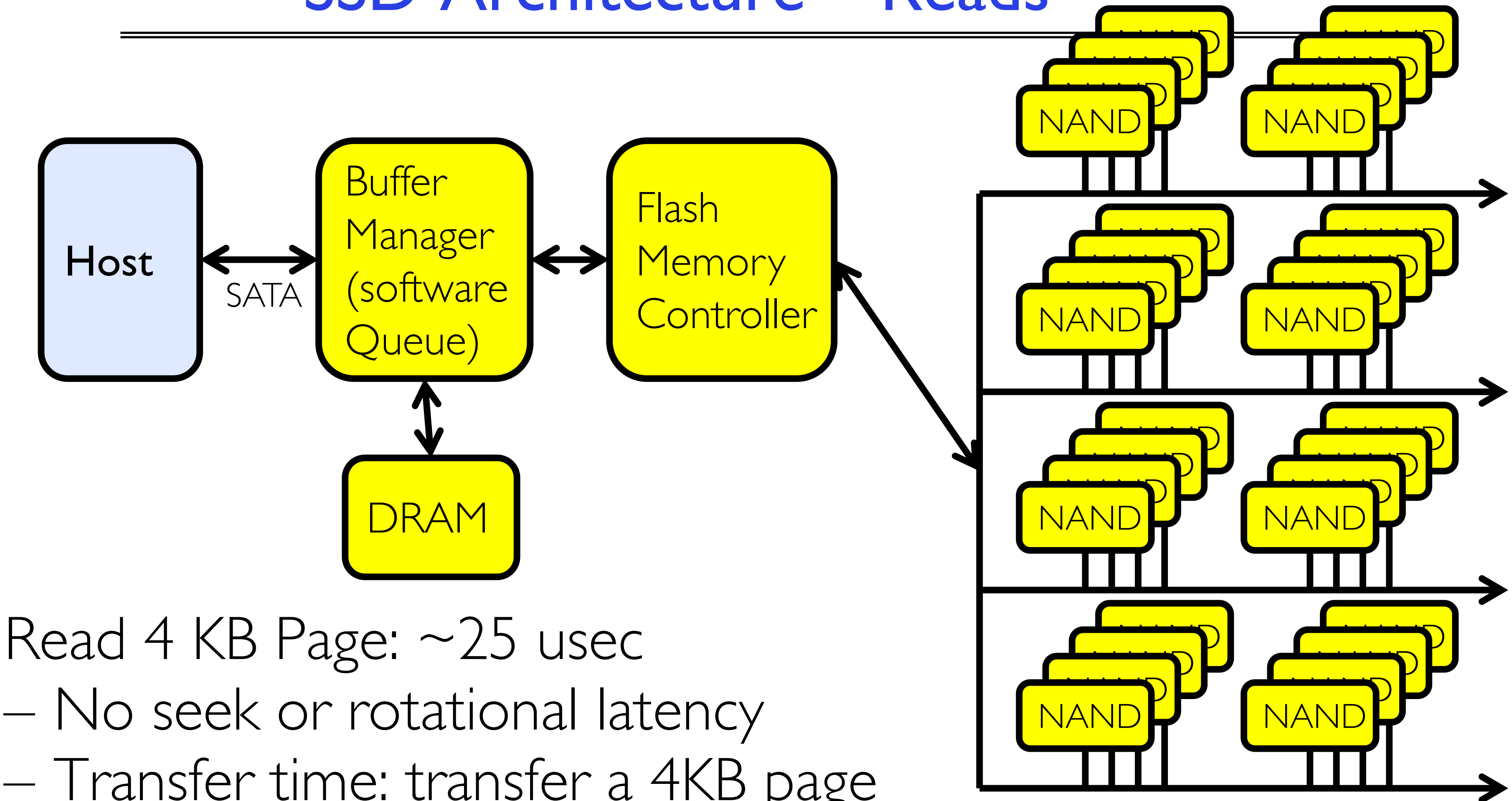
- Sectors contain sophisticated error correcting codes
  - Disk head magnet has a field wider than track
  - Hide corruptions due to neighboring track writes
- Sector sparing
  - Remap bad sectors transparently to spare sectors on the same surface
- Slip sparing
  - Remap all sectors (when there is a 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
- …

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

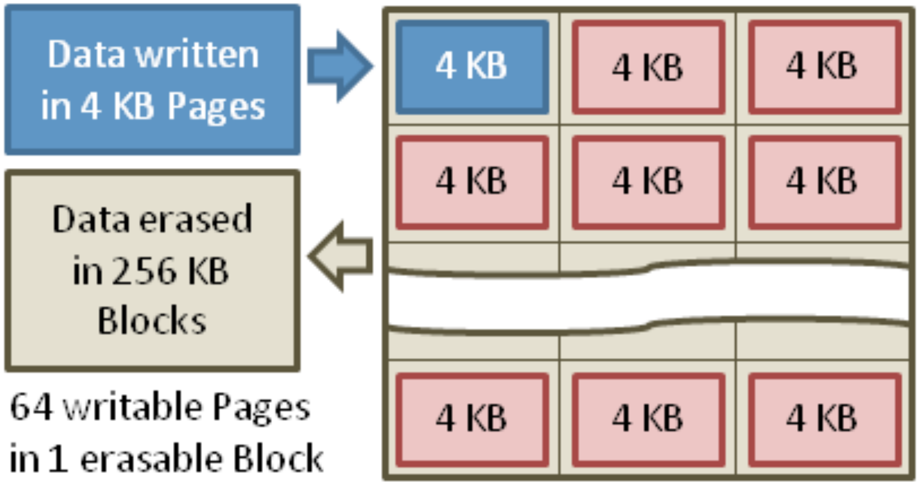


Read 4 KB Page: ~25 usec

- 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

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, ….
- According to 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!

Seagate Enterprise

10 TB (2016)

- 7 platters, 14 heads
- 7200 RPMs
- 6 Gbps SATA /12Gbps SAS interface
- 220MB/s transfer rate, cache size: 256MB
- Helium filled: reduce friction and power usage
- Price: $500 ($0.05/GB)

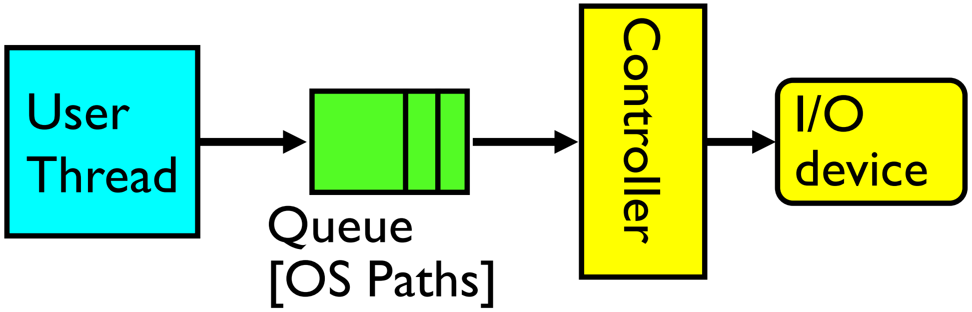
IBM Personal Computer/AT (1986)

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

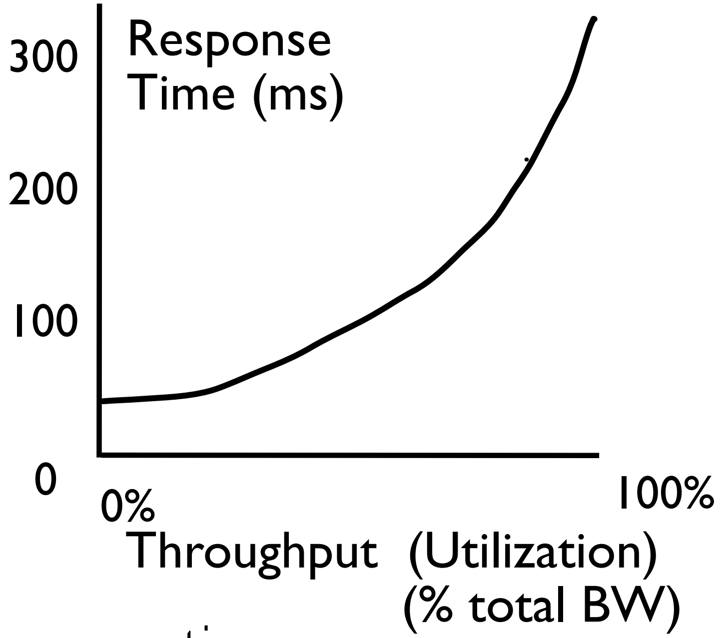
Largest SSDs

- 60TB (2016)
- Dual port: 16Gbs
- Seq reads: 1.5GB/s
- Seq writes: 1GB/s
- Random Read Ops (IOPS): 150K
- Price: ~ $20K ($0.33/GB)

I/O Performance


User Thread
Queue
[OS Paths]
Controller
I/O device

Response Time = Queue + I/O device service time



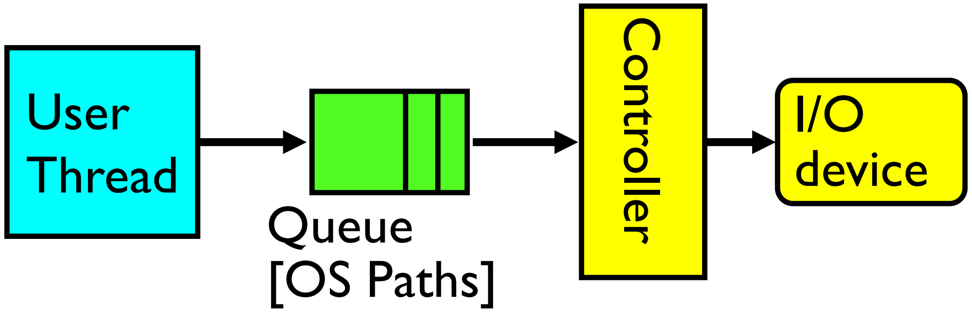
- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Effective BW per op = transfer size / response time
    - » EffBW(n) = n / (S + n/B) = B / (1 + SB/n)

# of ops

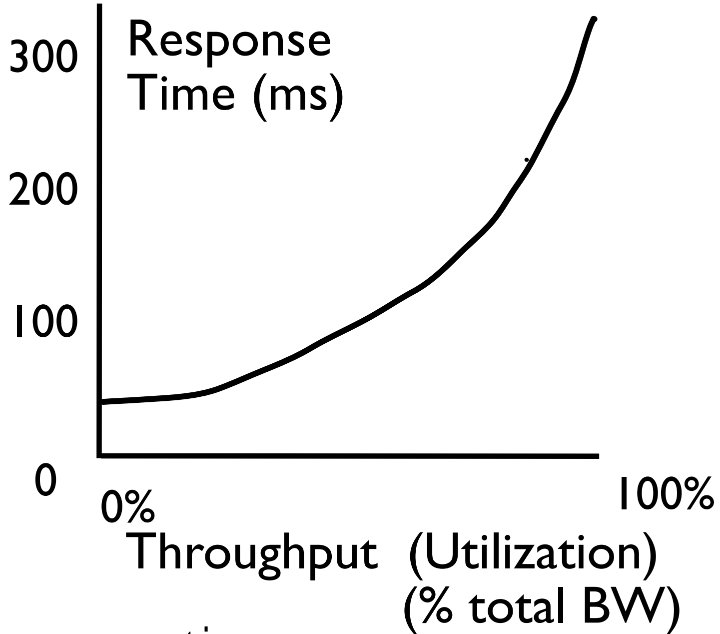
time per op

Fixed overhead

I/O Performance

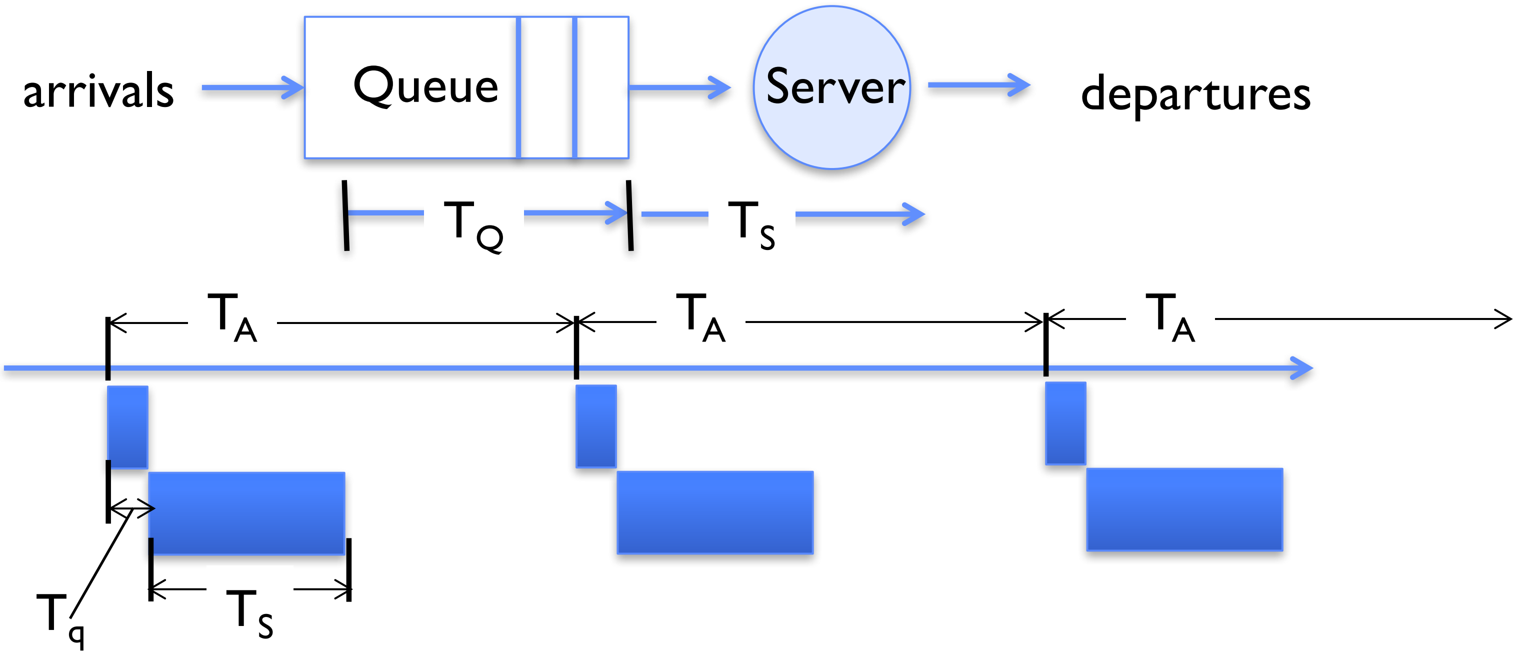


Response Time = Queue + I/O device service time



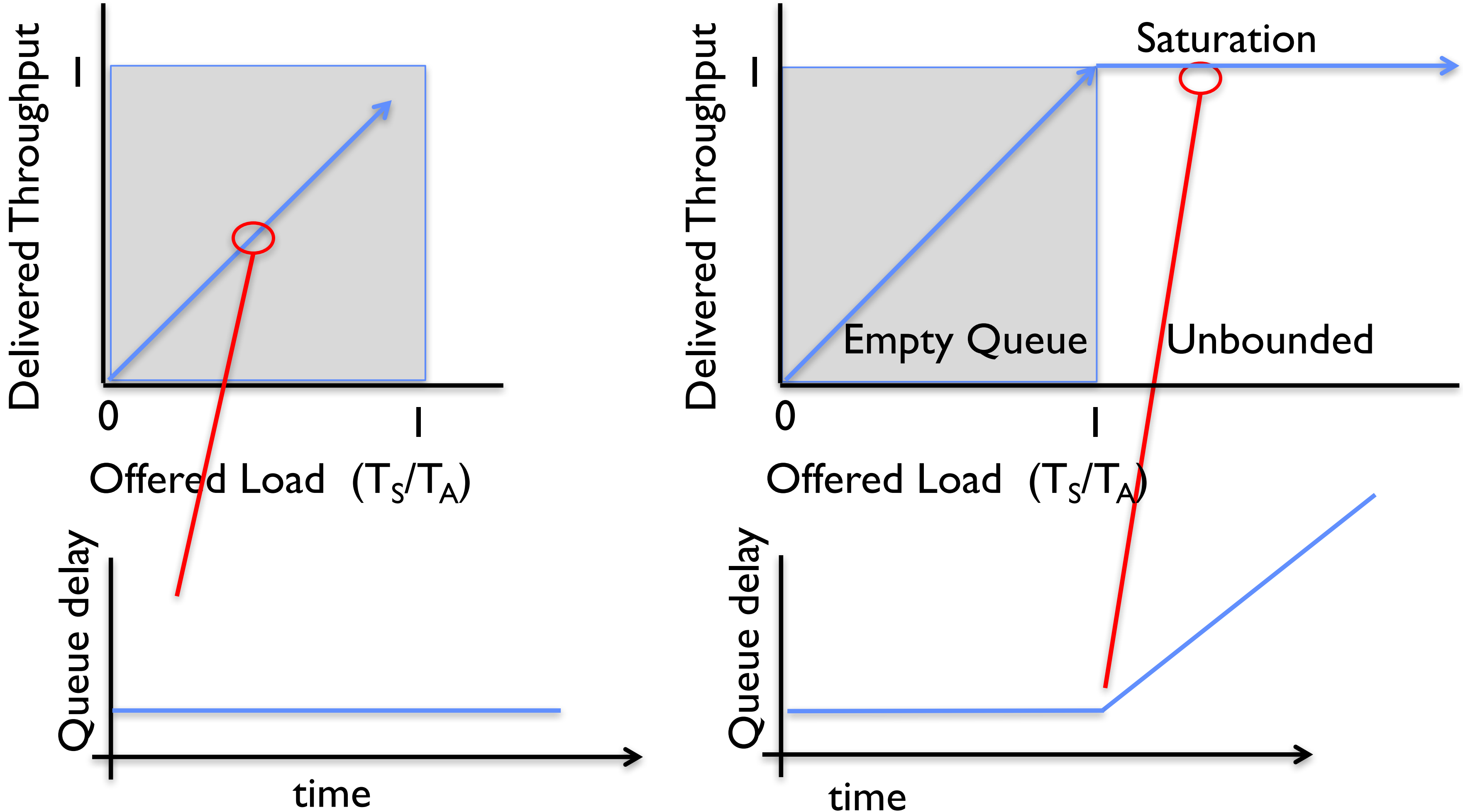
- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Effective BW per op = transfer size / response time
    - » EffBW(n) = n / (S + n/B) = B / (1 + SB/n)
  - Contributing factors to latency:
    - » Software paths (can be loosely modeled by a queue)
    - » Hardware controller
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

A Simple Deterministic World



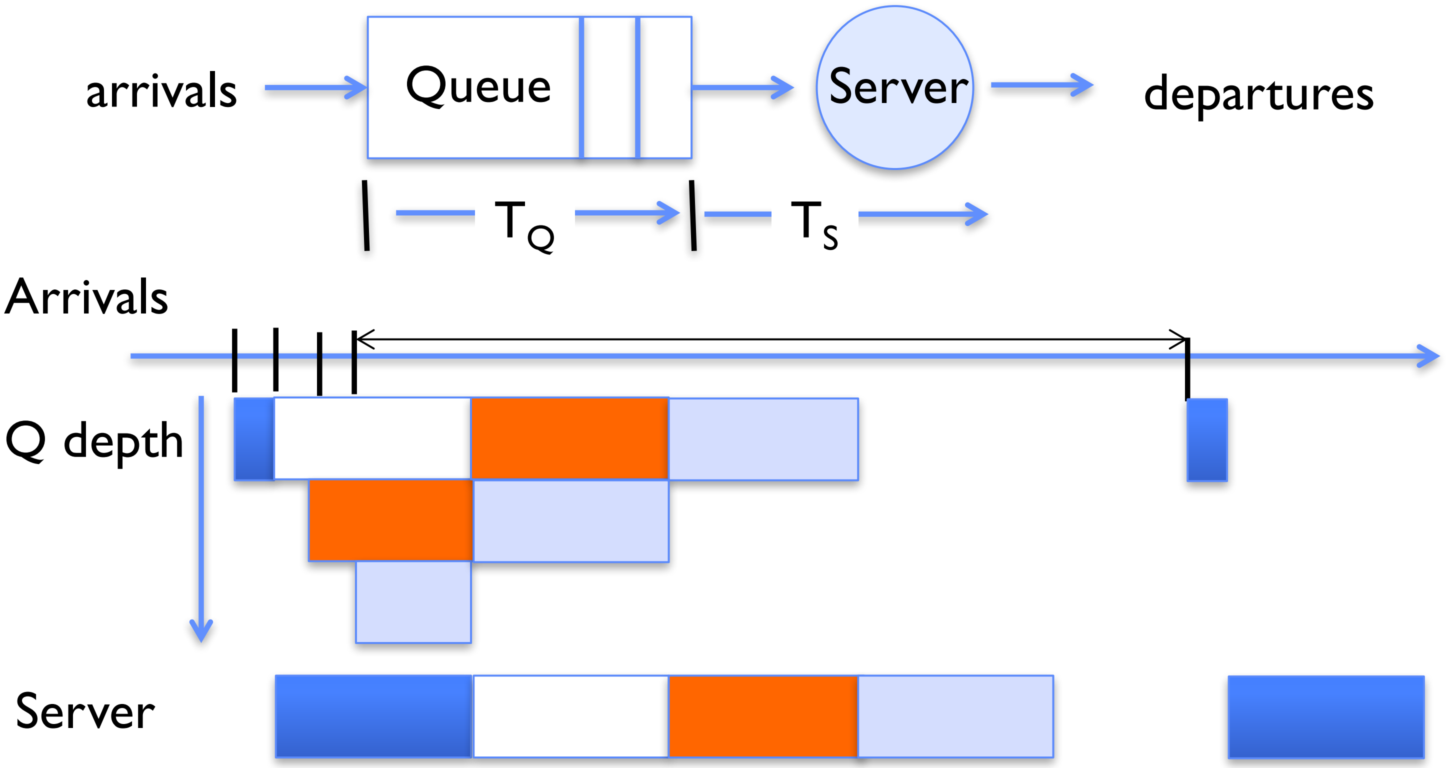
- Assume requests arrive at regular intervals, take a fixed time to process, with plenty of time between …
- Service rate ($\mu = 1/T_S$) - operations per sec
- Arrival rate: ($\lambda = 1/T_A$) - requests per second
- Utilization: $U = \lambda/\mu$, where $\lambda < \mu$
- Average rate is the complete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

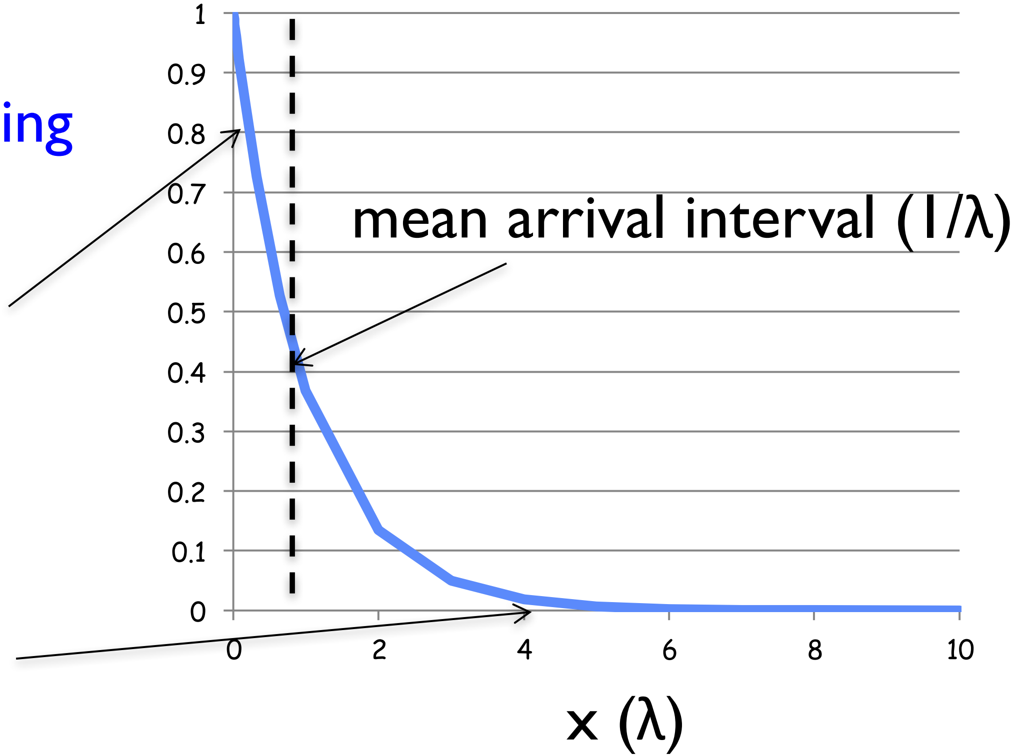
So how do we model the burstiness of arrival?

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

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

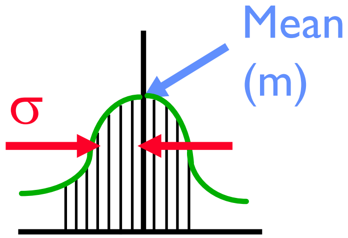
Lots of short arrival intervals (i.e., high instantaneous rate)

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



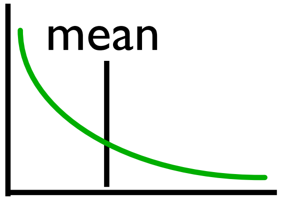
Background: General Use of Random Distributions

- 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 $C \approx 1.5$ (majority seeks < average)



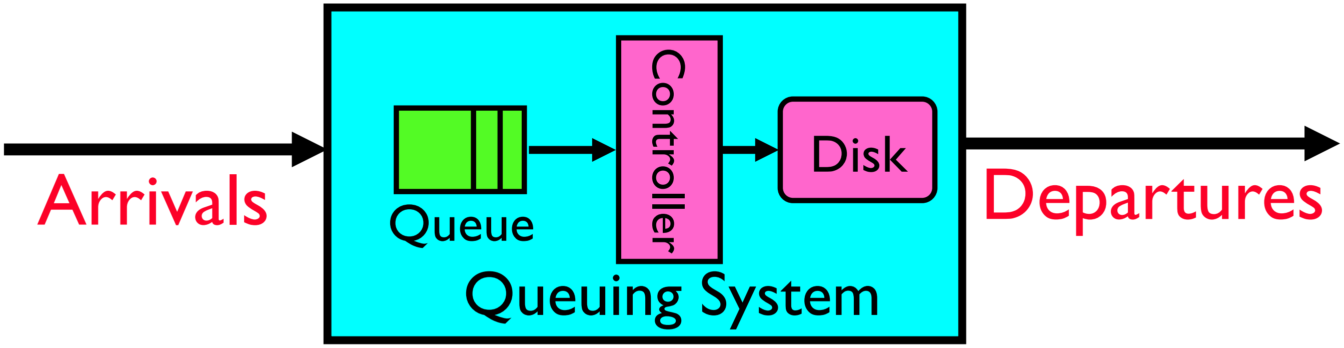
Memoryless

Administrivia

- Midterm 2 coming up on **Mon 10/29 5:00-6:30PM**
  - – All topics up to and including Lecture 17
    - » Focus will be on Lectures 11 – 17 and associated readings
    - » Projects 1 and 2
    - » Homework 0 – 2
  - – Closed book
  - – 2 pages hand-written notes both sides

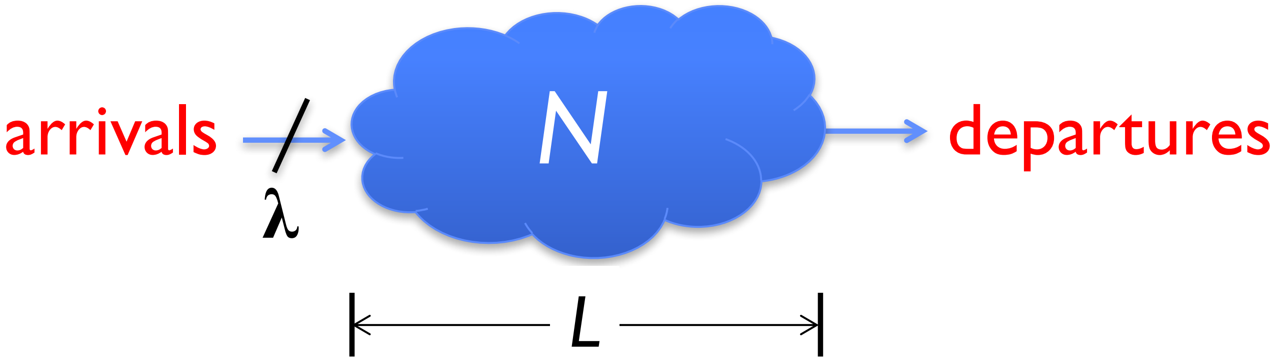
BREAK

Introduction to Queuing Theory



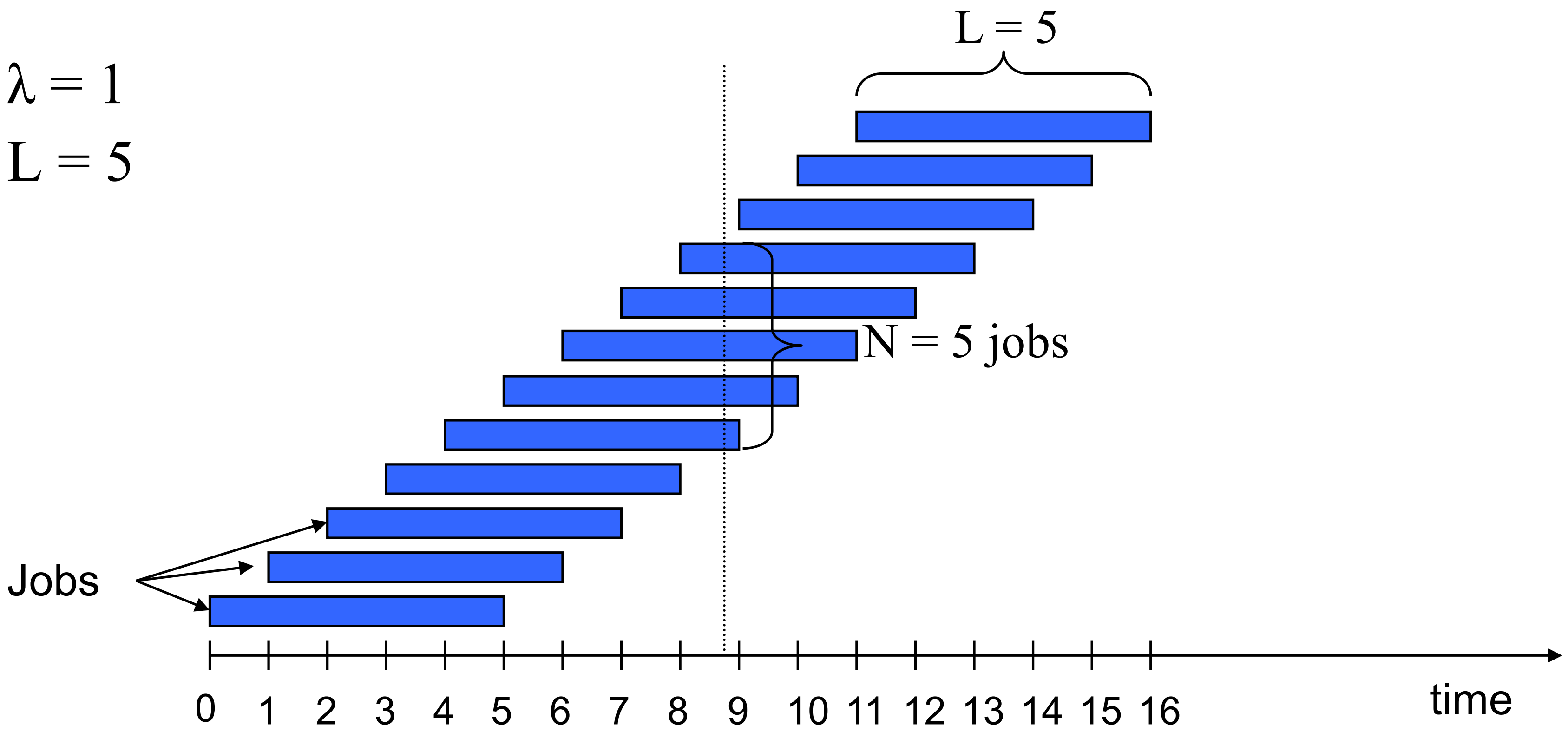
- What about queuing time??
  - Let's apply some queuing theory
  - Queuing Theory applies to long term, steady state behavior ⇒ Arrival rate = Departure rate
- Arrivals characterized by some probabilistic distribution
- Departures characterized by some probabilistic distribution

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

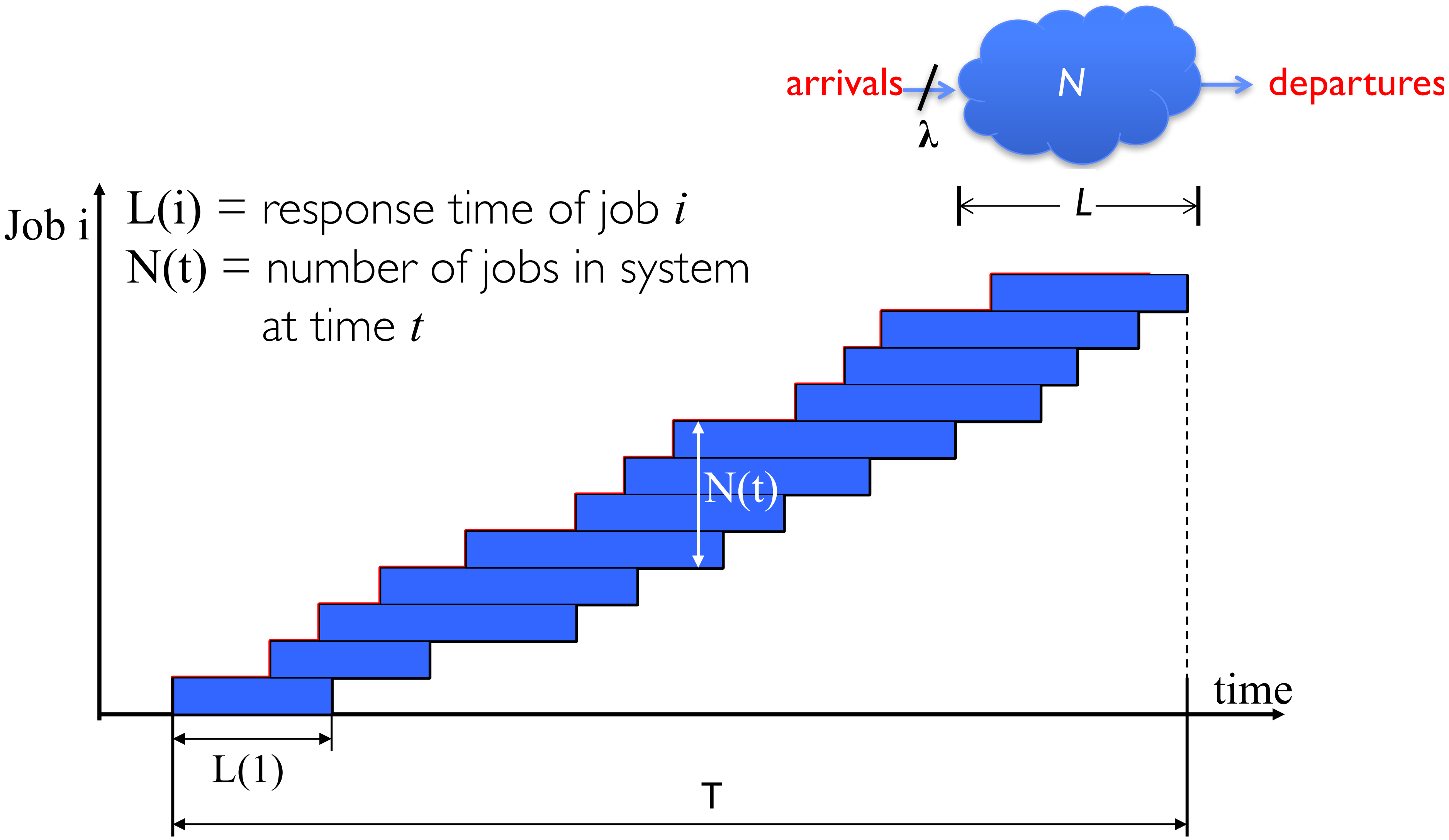
Example



A: $N = \lambda \times L$

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

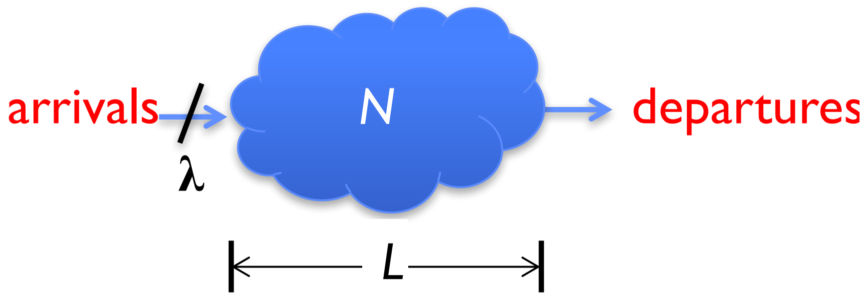
Little's Theorem: Proof Sketch



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

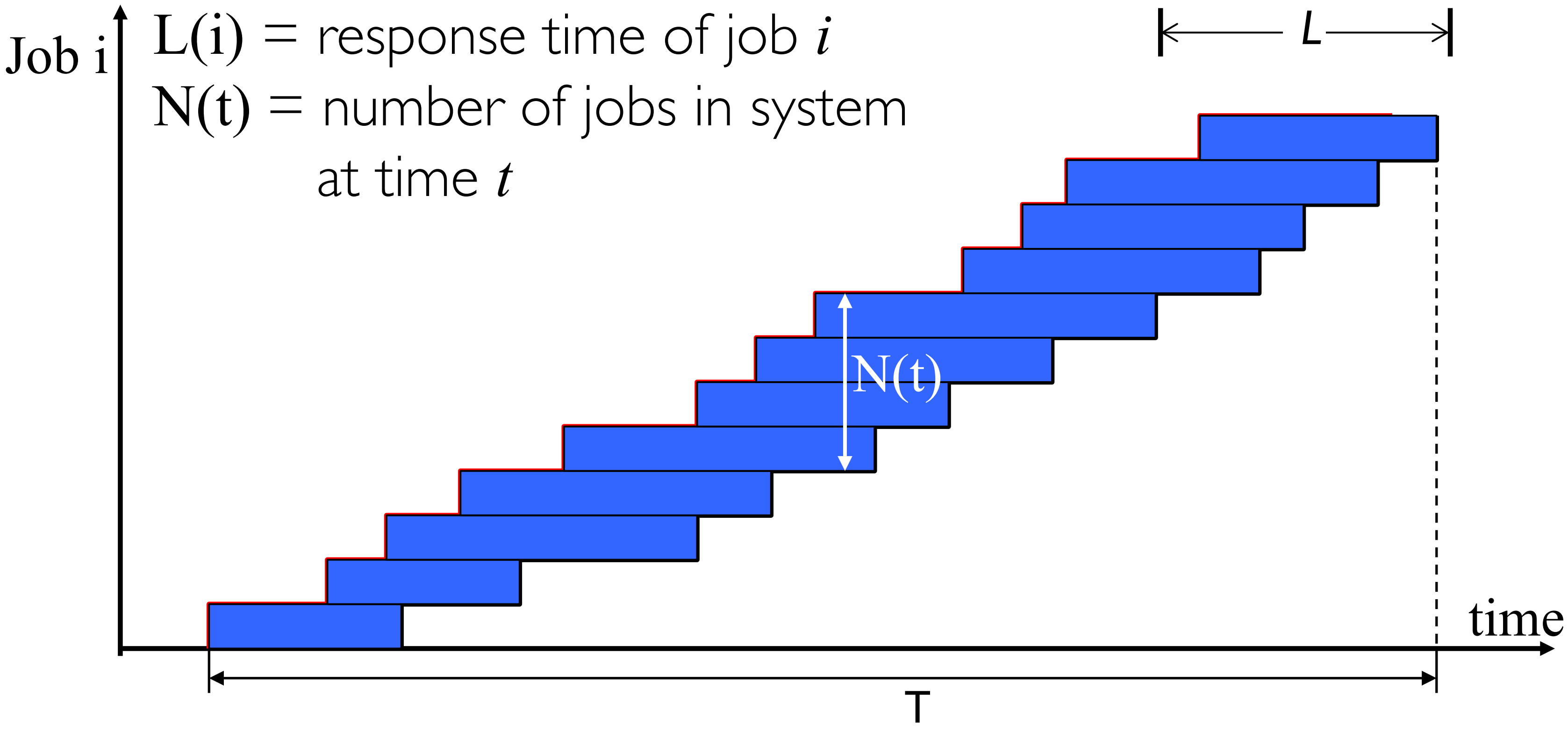
N(t) = number of jobs in system at time *t*

Little's Theorem: Proof Sketch



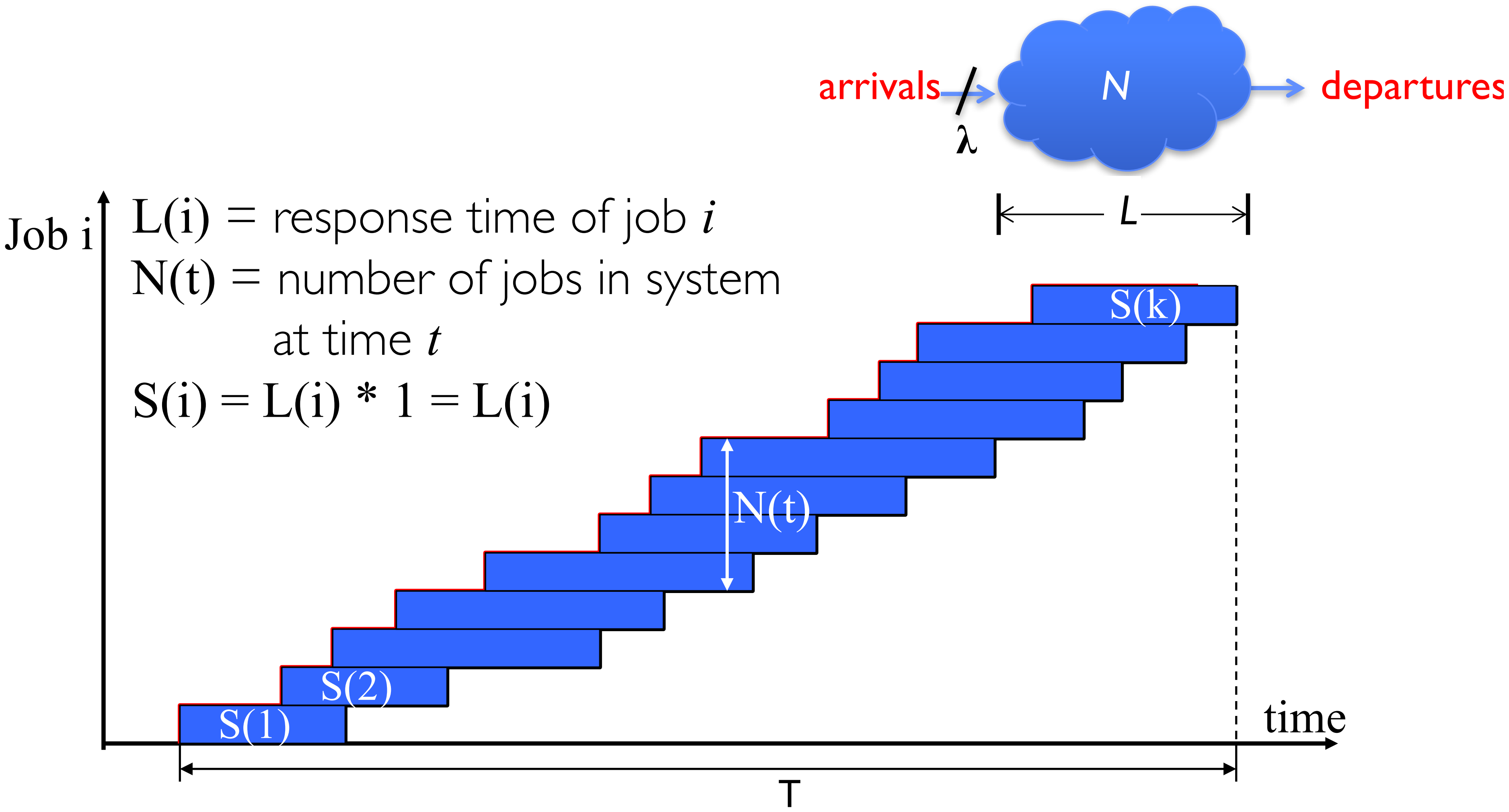
L(i) = response time of job *i*

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



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

Little’s Theorem: Proof Sketch



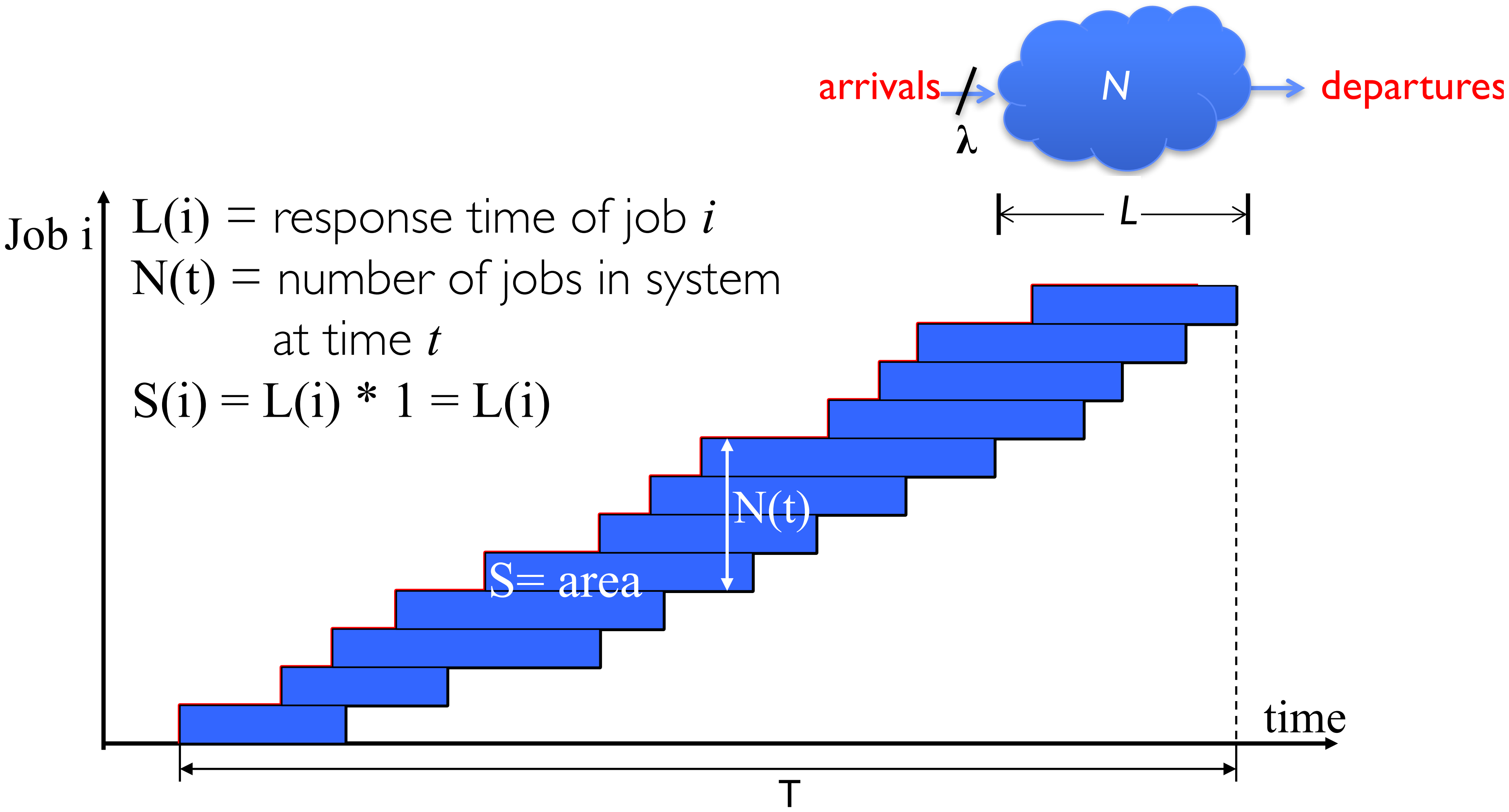
L(i) = response time of job *i*

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



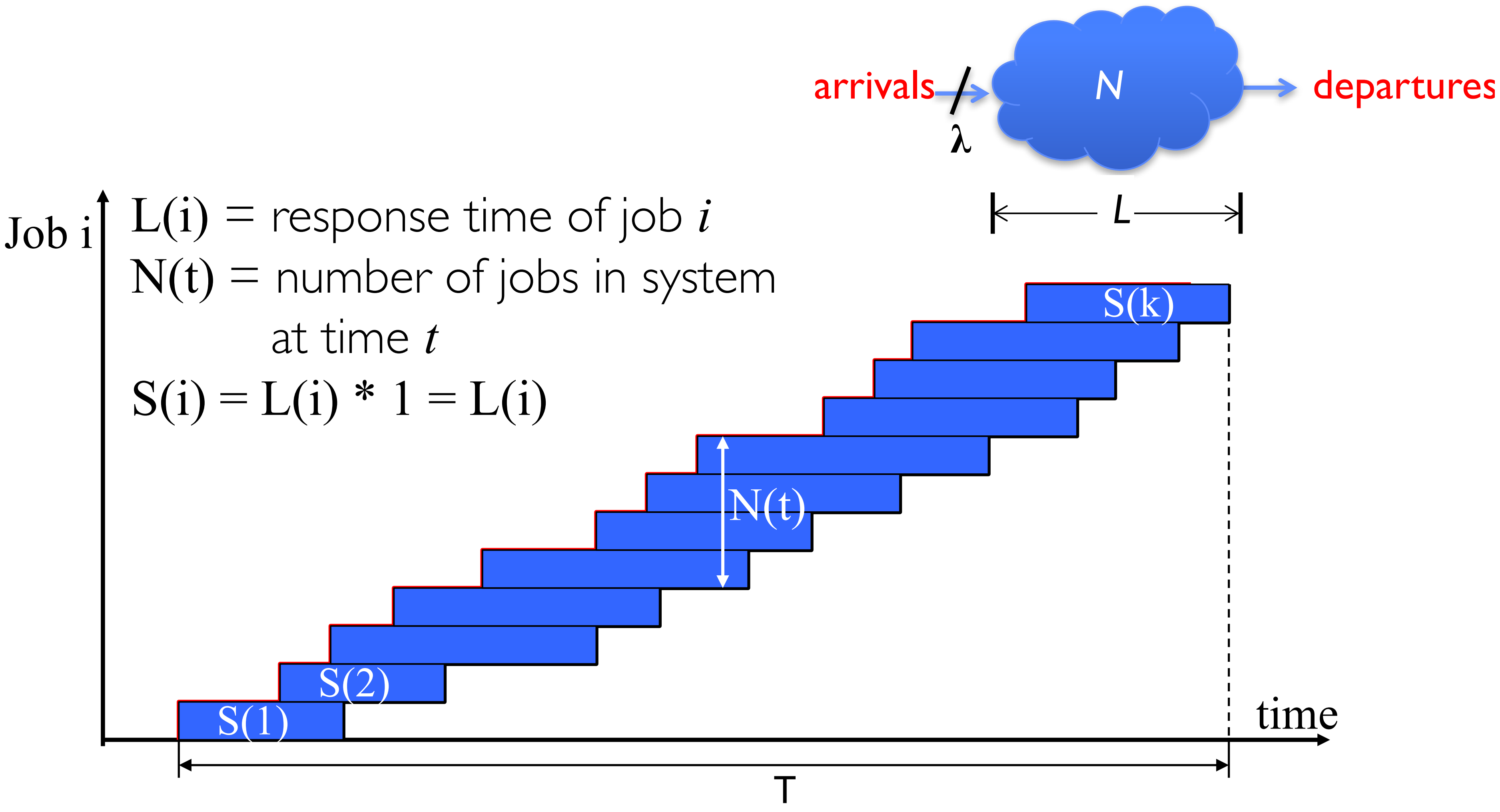
L(i) = response time of job *i*

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)

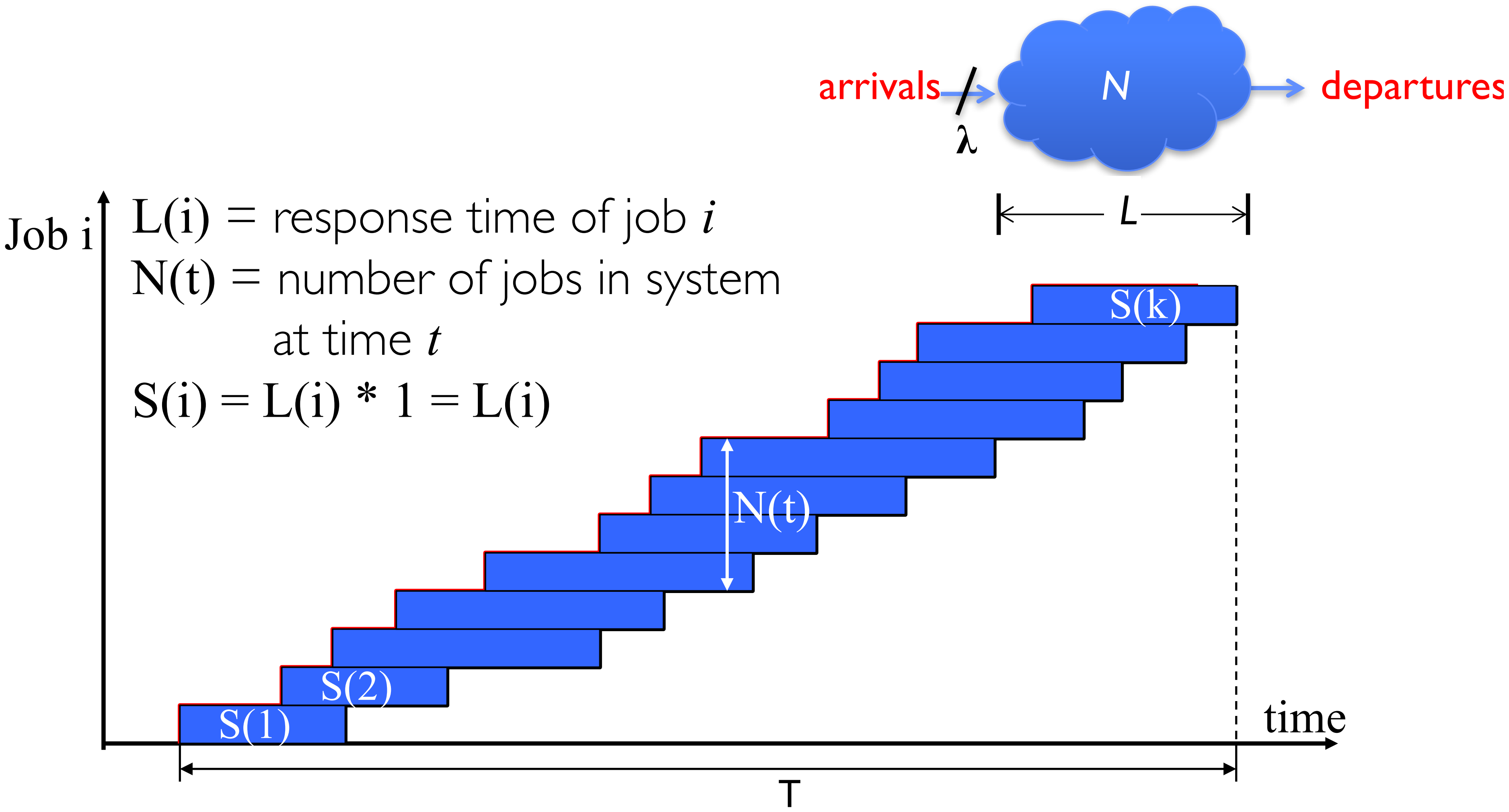
$$Navg = S/T = (L(1) + \ldots + L(k))/T$$

Little's Theorem: Proof Sketch

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

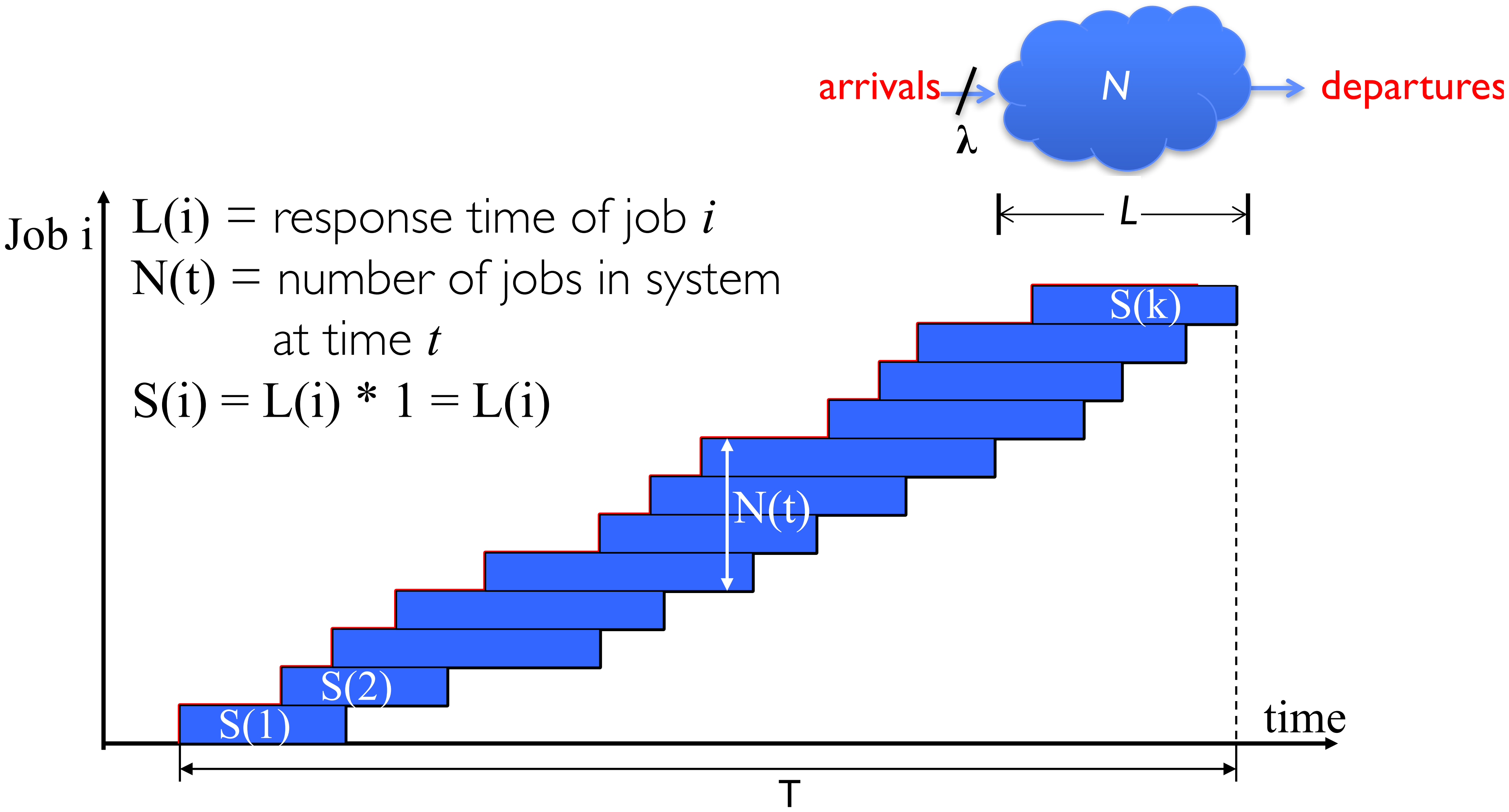
N(t) = number of jobs in system at time *t*

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



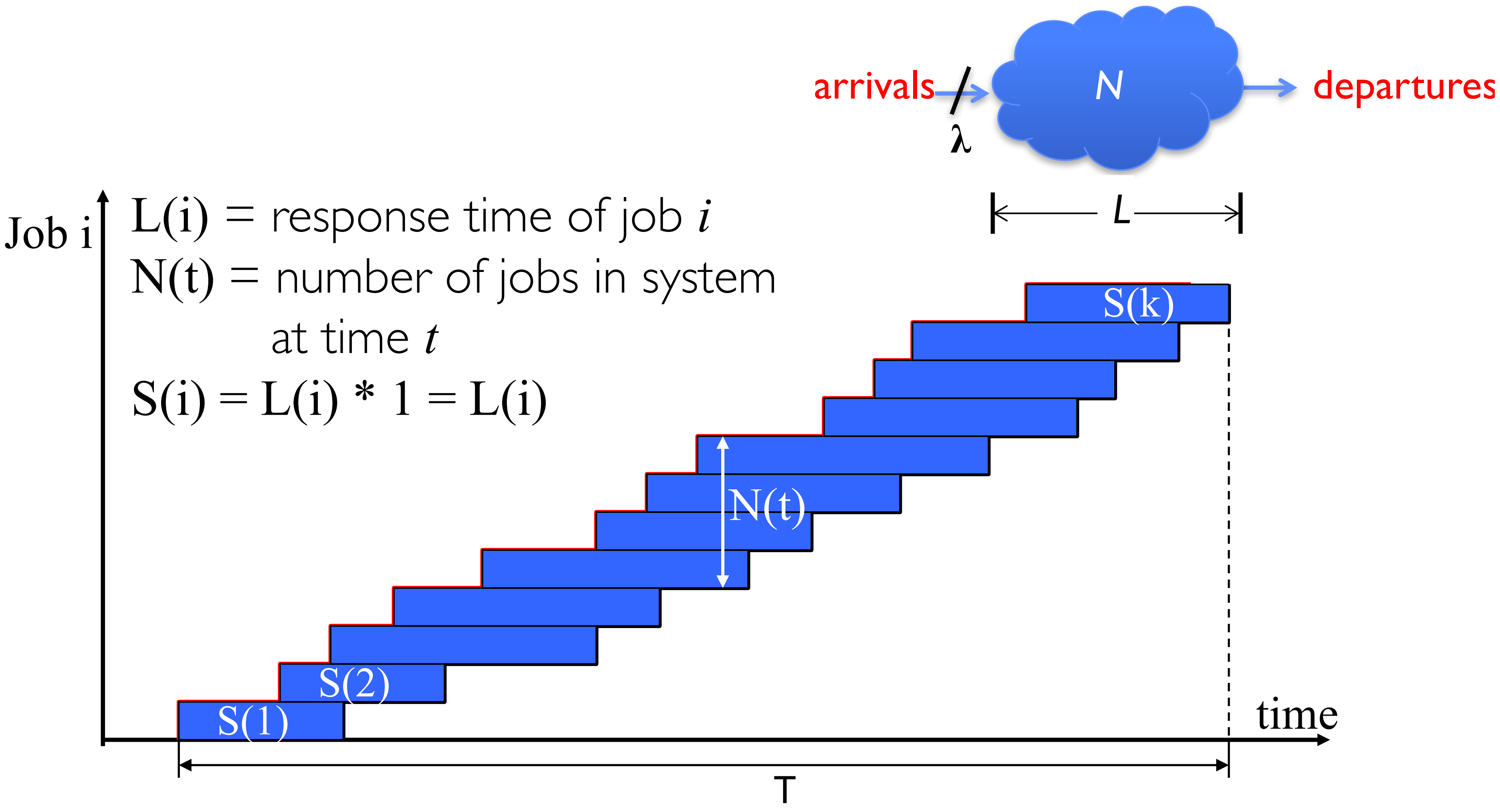
L(i) = response time of job *i*

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*

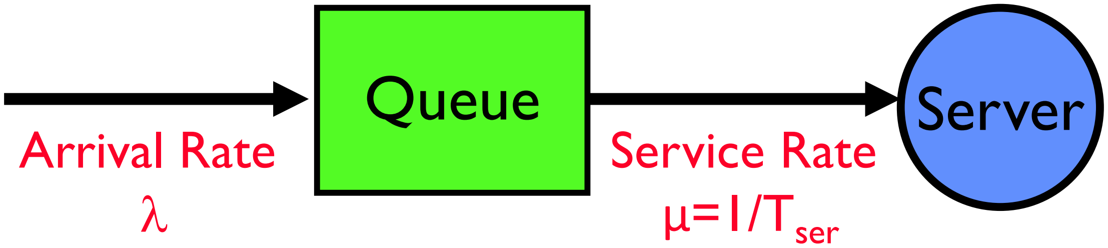
N(t) = number of jobs in system at time *t*

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

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

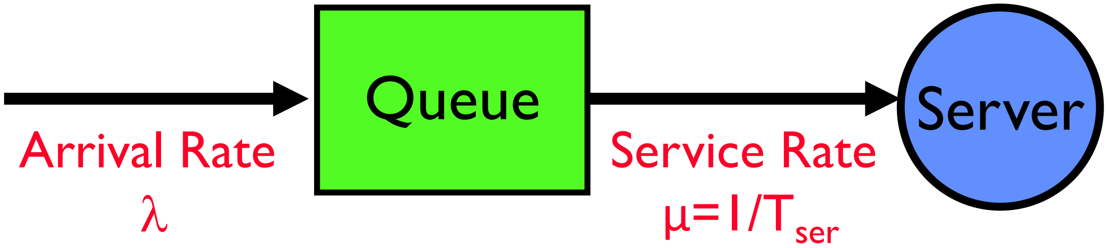
A Little Queuing Theory: Some Results (1/2)

- Assumptions:
  - System in equilibrium; No limit to the queue
  - Time between successive arrivals is random and memoryless



- Parameters that describe our system:
  - λ: mean number of arriving customers/second
  - T_{ser}: mean time to service a customer ("m")
  - C: squared coefficient of variance = σ^2/m^2
  - μ: service rate = $1/T_{ser}$
  - u: server utilization ($0 \le u \le 1$): $u = \lambda/\mu = \lambda \times T_{ser}$
- Parameters we wish to compute:
  - T_q: Time spent in queue
  - L_q: Length of queue = $\lambda \times T_q$ (by Little's law)

A Little Queuing Theory: Some Results (2/2)



Queue

Server

Arrival Rate λ

Service Rate $\mu = 1/T_{ser}$

- Parameters that describe our system:
  - λ: mean number of arriving customers/second $\lambda = 1/T_A$
  - T_{ser}: mean time to service a customer (“m”)
  - C: squared coefficient of variance $= \sigma^2/m^2$
  - μ: service rate $= 1/T_{ser}$
  - u: server utilization ($0 \le u \le 1$): $u = \lambda/\mu = \lambda \times T_{ser}$
- Parameters we wish to compute:
  - T_q: Time spent in queue
  - L_q: Length of queue $= \lambda \times T_q$ (by Little’s law)
- **Results** (**M**: Poisson arrival process, **1** server):
  - **M**emoryless service time distribution (C = 1): Called an **M/M/1** queue
    - » $T_q = T_{ser} \times \boxed{u/(1 - u)}$
  - **G**eneral service time distribution (no restrictions): Called an **M/G/1** queue
    - » $T_q = T_{ser} \times \frac{1}{2}(1+C) \times \boxed{u/(1 - u)})$

A Little Queuing Theory: An Example (1/2)

- 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 + rotation + transfer)

- Questions:
  - How utilized is the disk (server utilization)? Ans:, $u = \lambda T_{ser}$
  - 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: $T_{sys} = T_q + T_{ser}$

A Little Queuing Theory: An Example (2/2)

- Questions:
  - How utilized is the disk (server utilization)? Ans:, $u = \lambda T_{ser}$
  - 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) = $\lambda \times T_{ser} = 10/s \times .02s = 0.2$

T_q (avg time/customer in queue) = $T_{ser} \times u/(1 - u)$

$= 20 \times 0.2/(1-0.2) = 20 \times 0.25 = 5$ ms (0 .005s)

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

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

Queuing Theory Resources

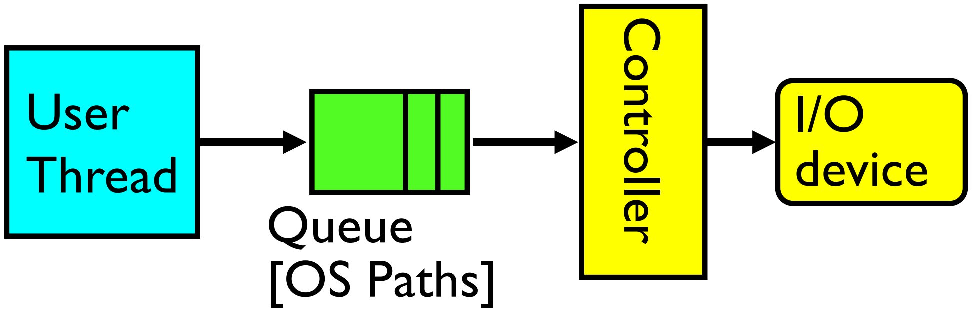
- Resources page contains Queueing Theory Resources (under Readings):
  - Scanned pages from Patterson and Hennessy book that gives further discussion and simple proof for general equation: https://cs162.eecs.berkeley.edu/static/readings/patterson_queue.pdf
  - A complete website full of resources: http://web2.uwindsor.ca/math/hlynka/qonline.html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

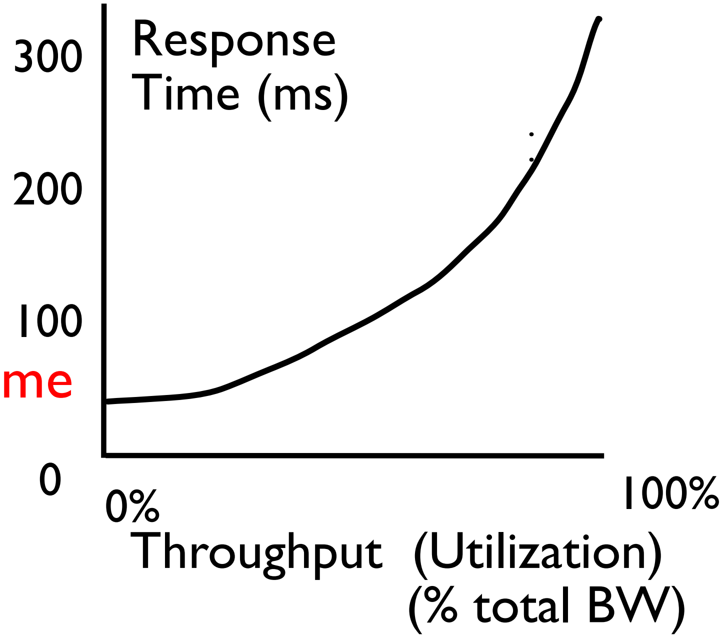
Summary

- 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 → ∞

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

Optimize I/O Performance





Response Time = Queue + I/O device service time

- How to improve performance?
  - Make everything faster ☺
  - More decoupled (Parallelism) systems
  - Do other useful work while waiting
    - » Multiple independent buses or controllers
  - Optimize the bottleneck to increase service rate
    - » Use the queue to optimize the service
- Queues absorb bursts and smooth the flow
- Add admission control (finite queues)
  - Limits delays, but may introduce unfairness and livelock

When is Disk Performance Highest?

- When there are big sequential reads, or
- When there is so much work to do that they can be piggy backed (reordering queues—one moment)

- OK to be inefficient when things are mostly idle
- Bursts are both a threat and an opportunity
- <your idea for optimization goes here>
  - Waste space for speed?

- Other techniques:
  - Reduce overhead through user level drivers
  - Reduce the impact of I/O delays by doing other useful work in the meantime