CS162
Operating Systems and Systems Programming
Lecture 21

Filesystems 1: Performance, Queueing Theory, Filesystem Design (Beginning)

November 14th, 2024

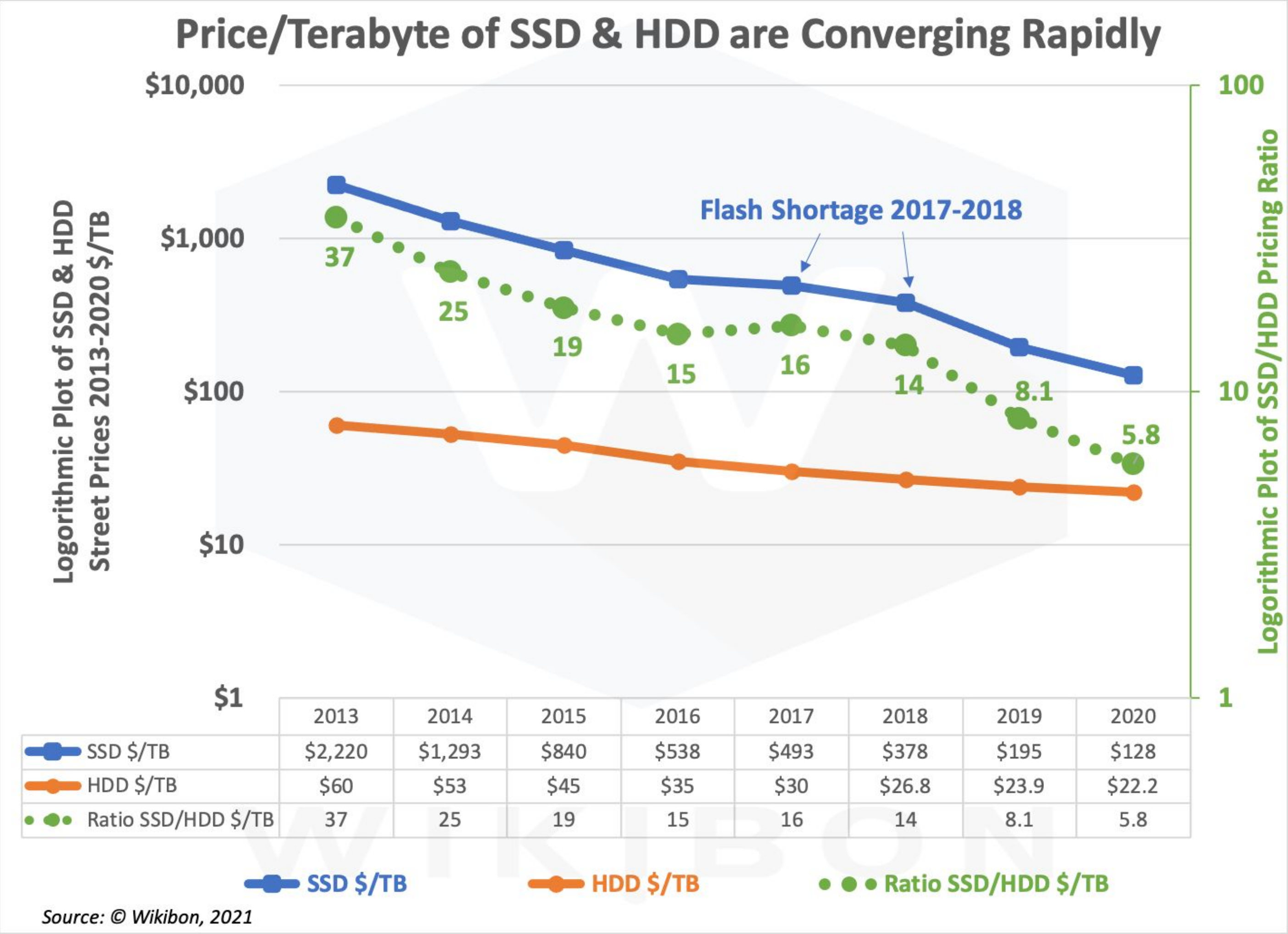
Prof. Ion Stoica

http://cs162.eecs.Berkeley.edu

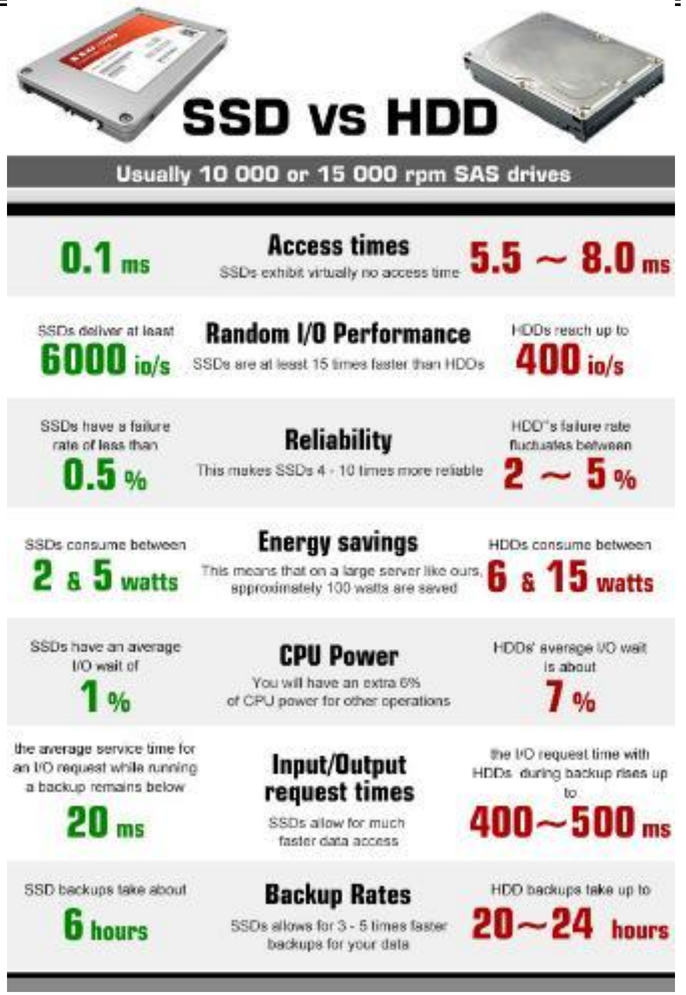
Some “Current” (large) 3.5in SSDs

- Seagate Exos SSD: 15.36TB (2017)
  – Dual 12Gb/s interface
  – Seq reads 860MB/s
  – Seq writes 920MB/s
  – Random Reads (IOPS): 102K
  – Random Writes (IOPS): 15K
  – Price (Amazon): $5495 ($0.36/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: ~ $40K? ($0.4/GB)
    » However, 50TB drive costs $12500 ($0.25/GB)

HDD vs. SSD Comparison



	2013	2014	2015	2016	2017	2018	2019	2020
SSD $/TB	$2,220	$1,293	$840	$538	$493	$378	$195	$128
HDD $/TB	$60	$53	$45	$35	$30	$26.8	$23.9	$22.2
Ratio SSD/HDD $/TB	37	25	19	15	16	14	8.1	5.8



SSD prices drop faster than 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

SSD Summary

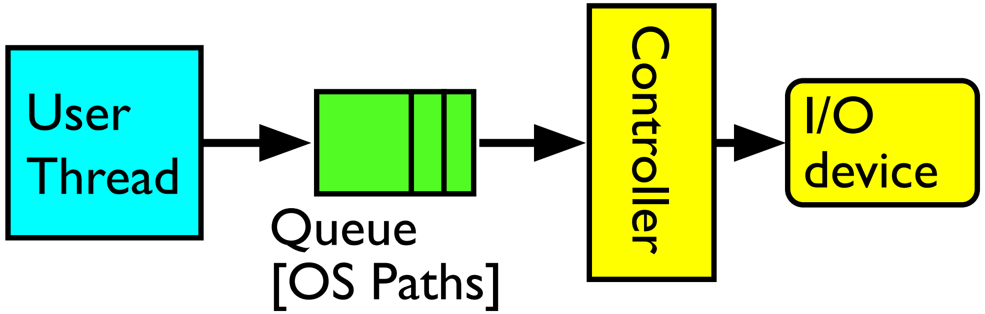
- Pros (vs. hard disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - » Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O bus)
- Cons
  - ~~Small storage (0.1-0.5x disk)~~, expensive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

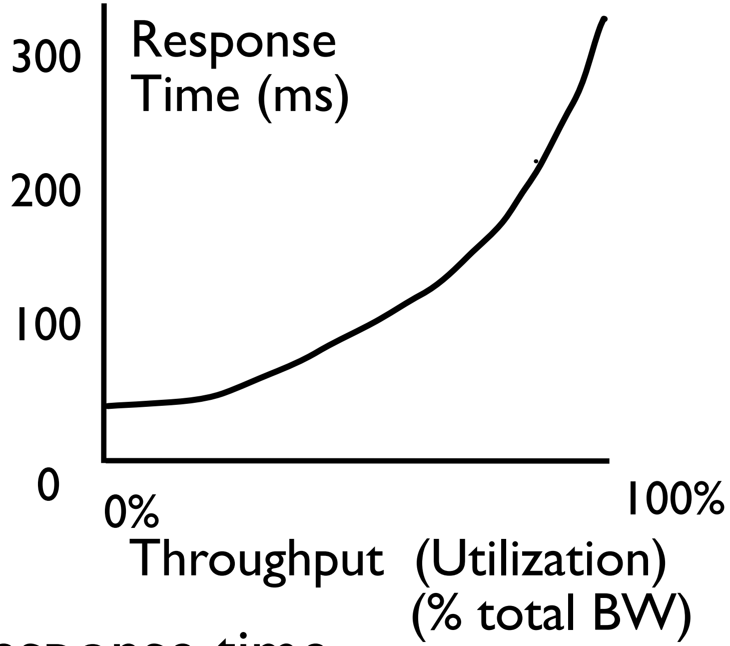
Review: Basic Performance Concepts

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

I/O Performance



Response Time = Queue + I/O device service time



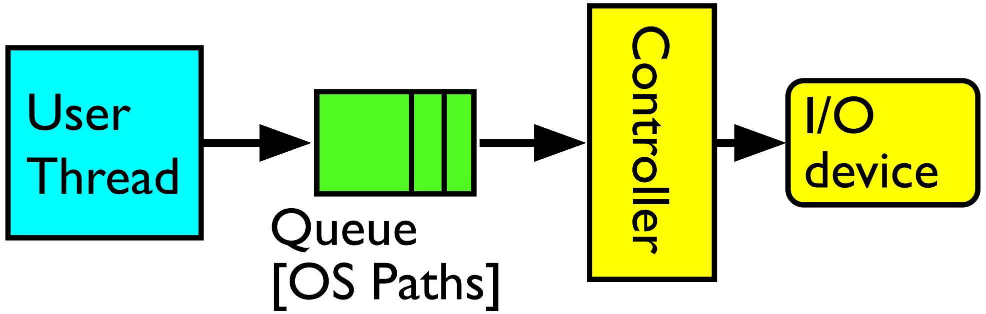
- Performance of I/O subsystem
  – Metrics: Response Time, Throughput
  – Effective BW per op = transfer size / response time
    » EffBW(n) = n / (S + n/B) = B / (1 + SB/n)

# of ops

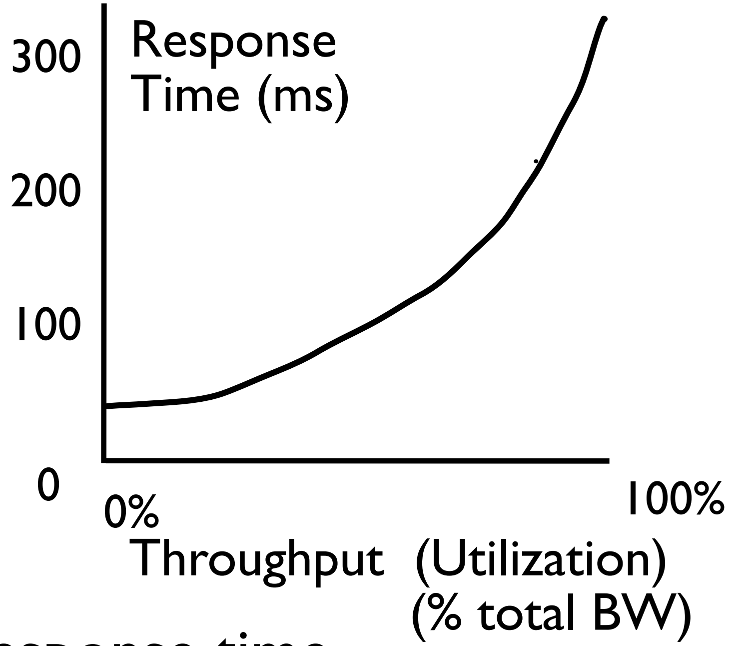
Fixed overhead

time per op

I/O Performance

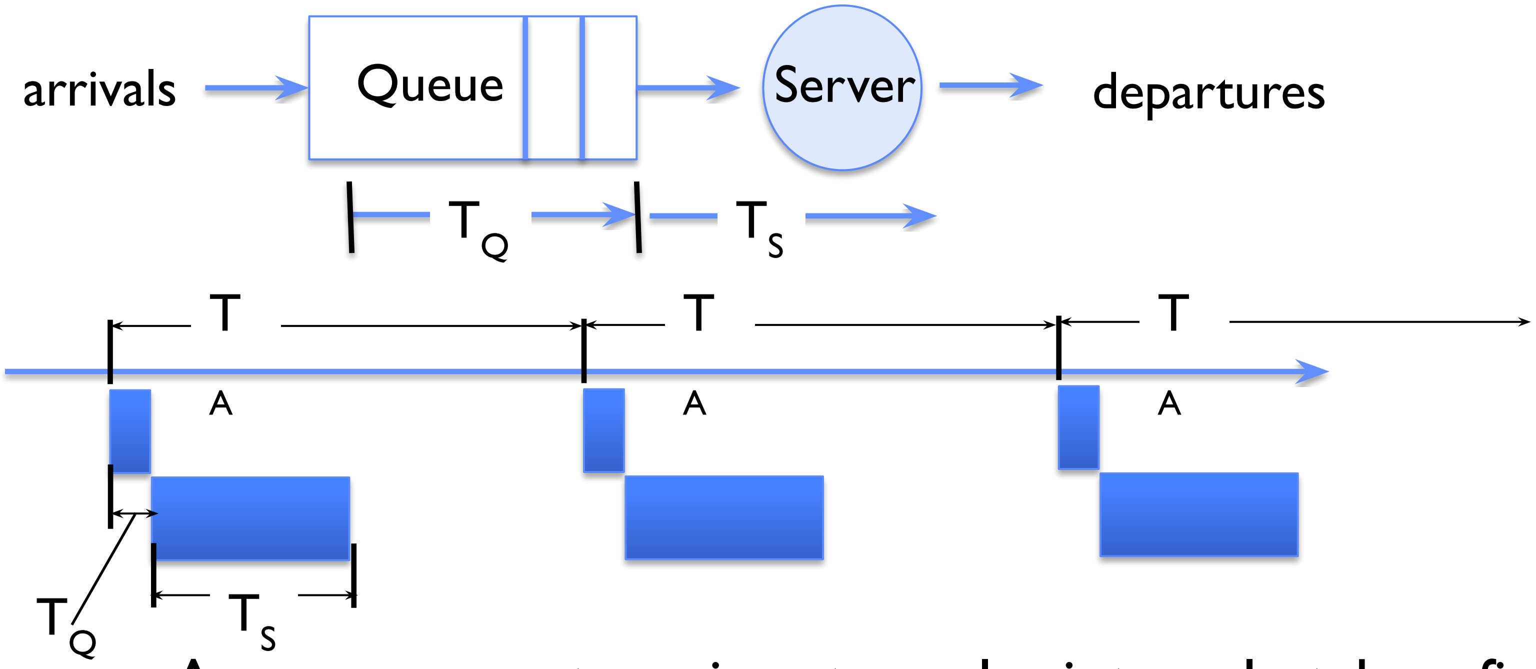


Response Time = Queue + I/O device service time



- 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?

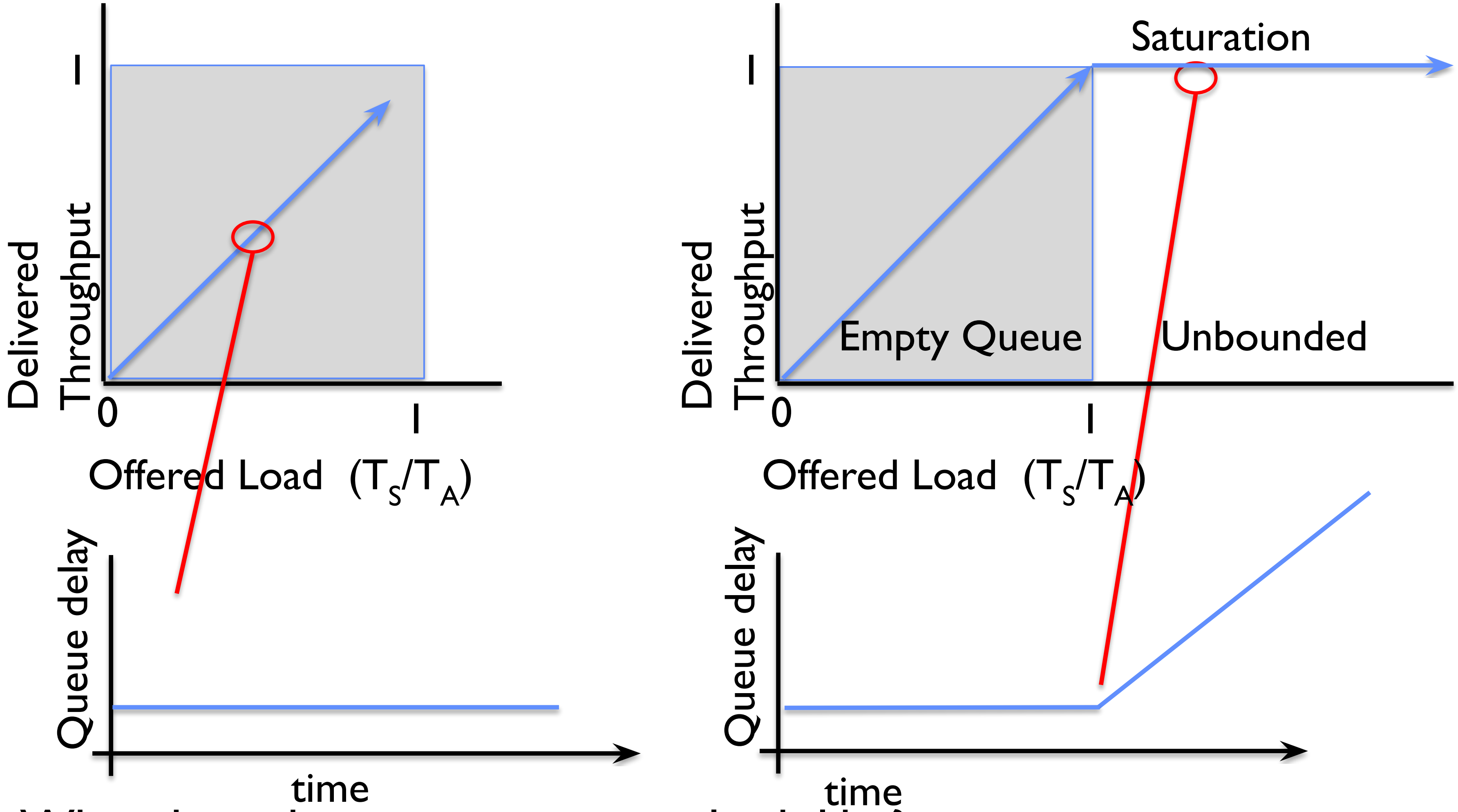
A Simple Deterministic World



T_A = Interarrival time
T_S = Service time
T_Q = Queueing delay

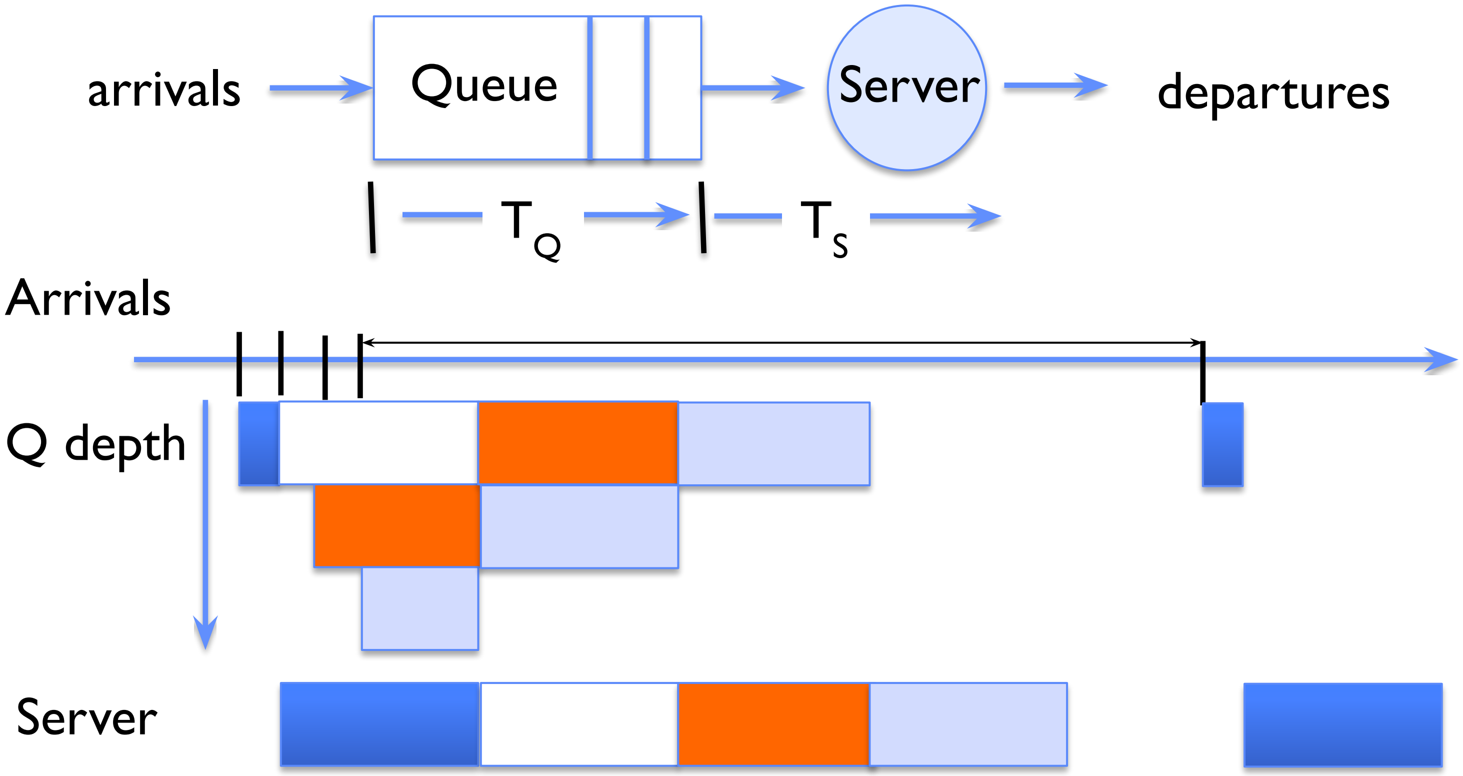
- Assume requests arrive at regular intervals, take a fixed time to process, with plenty of time between …
- Service rate ($\mu = 1/T_S$) - operations per sec
- Arrival rate: ($\lambda = 1/T_A$) - requests per second
- Utilization: $U = \lambda/\mu = T_S/T_A$, where $\lambda < \mu$
- Average rate is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

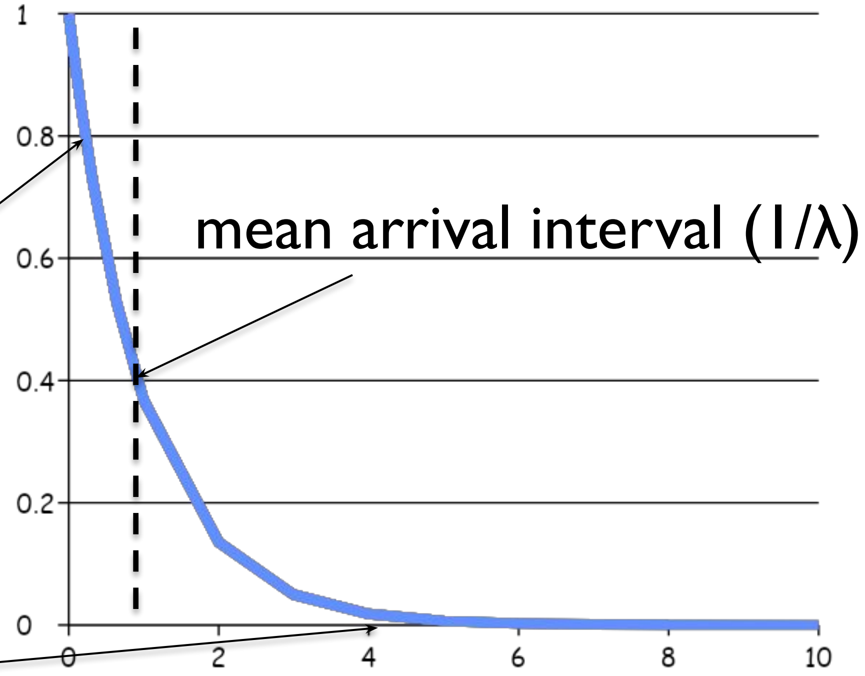
So how do we model the burstiness of arrival?

- 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)

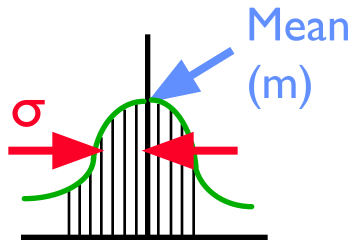
Few long gaps (i.e., low instantaneous rate)



x (λ)

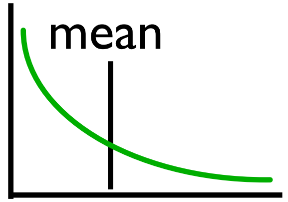
Background: General Use of Random Distributions

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



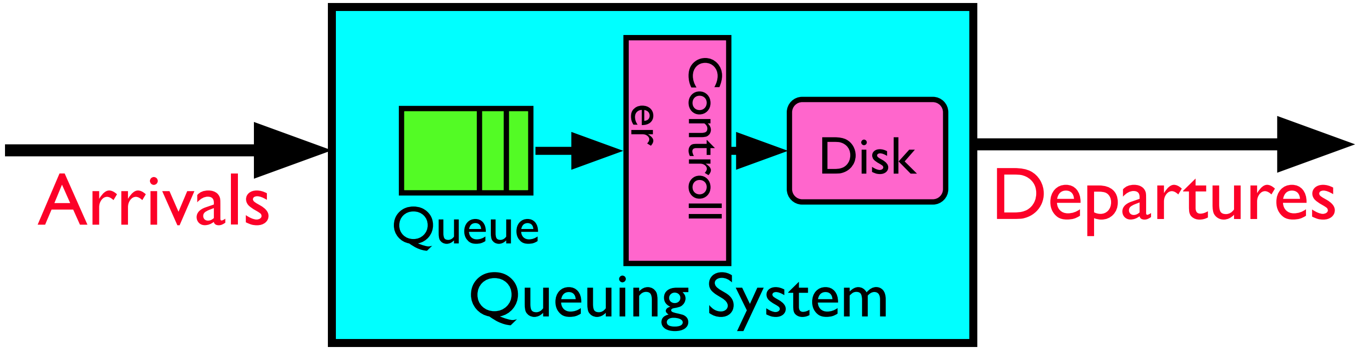
Distribution
of service times

- 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)



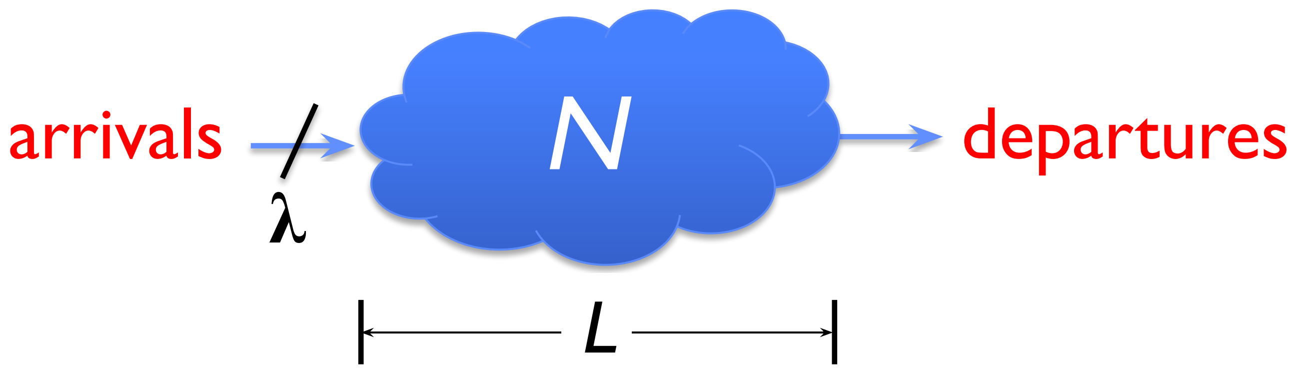
Memoryless

Introduction to Queuing Theory



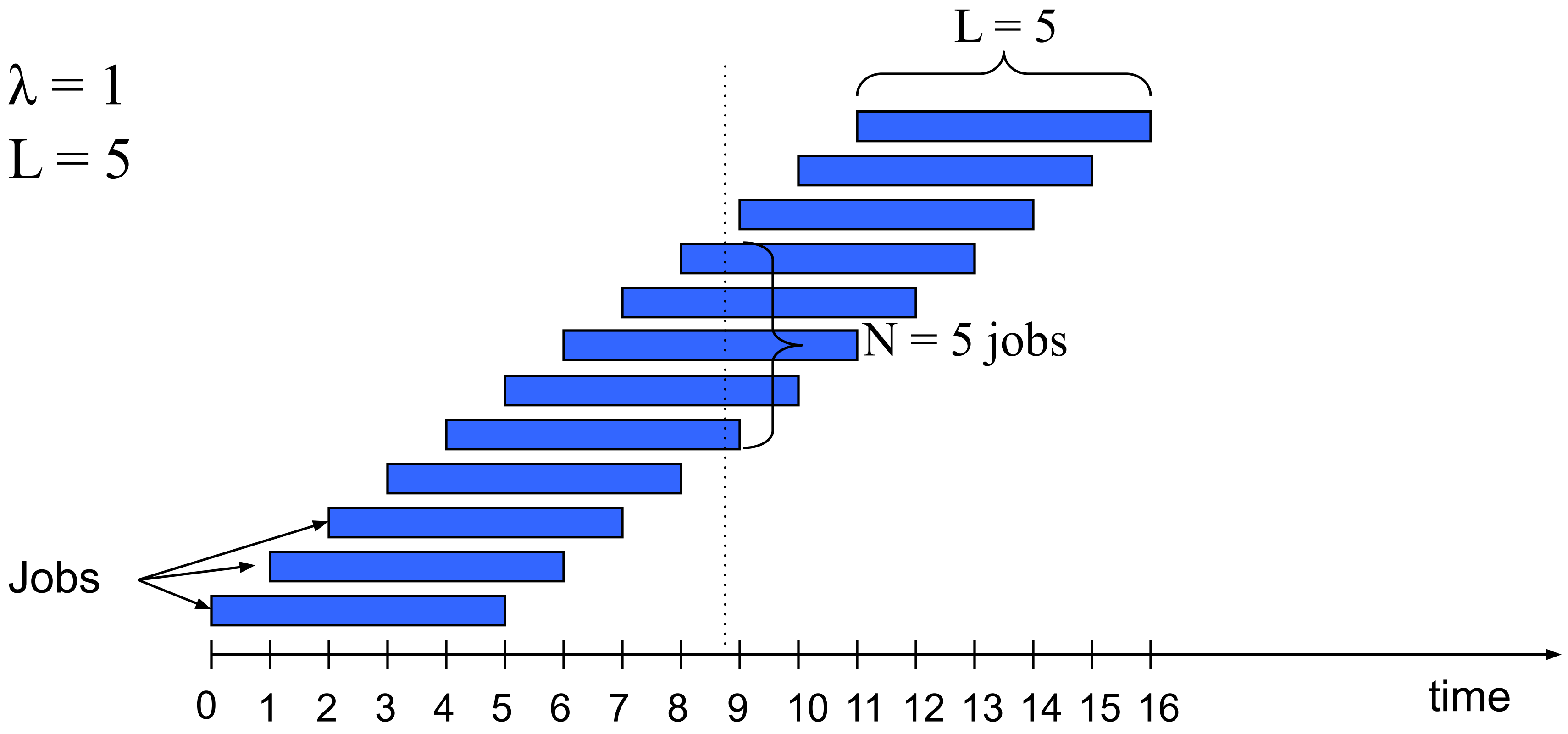
- What about queuing time??
  - Let's apply some queuing theory
  - Queuing Theory applies to long term, steady state behavior ⇒ Arrival rate = Departure rate
- Arrivals characterized by some probabilistic distribution
- Departures characterized by some probabilistic distribution

Little's Law

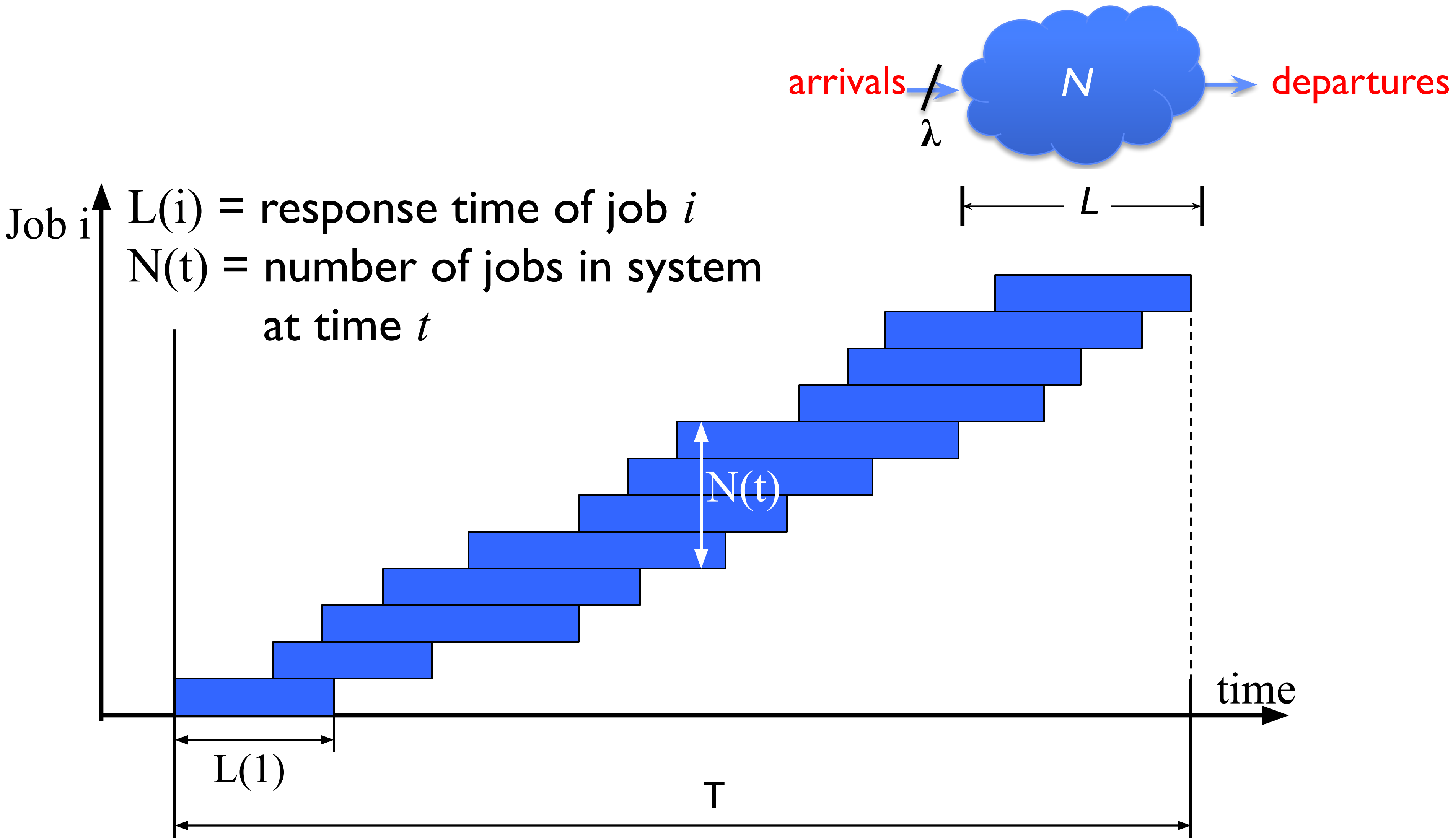


- In any *stable* system
  - Average arrival rate = Average departure rate
- The average number of jobs/tasks in the system (*N*) is equal to mean arrival time (λ) times the response time (*L*)
  - *N (jobs) =* λ *(jobs/s) x L (s)*
- Regardless of structure, bursts of requests, variation in service
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

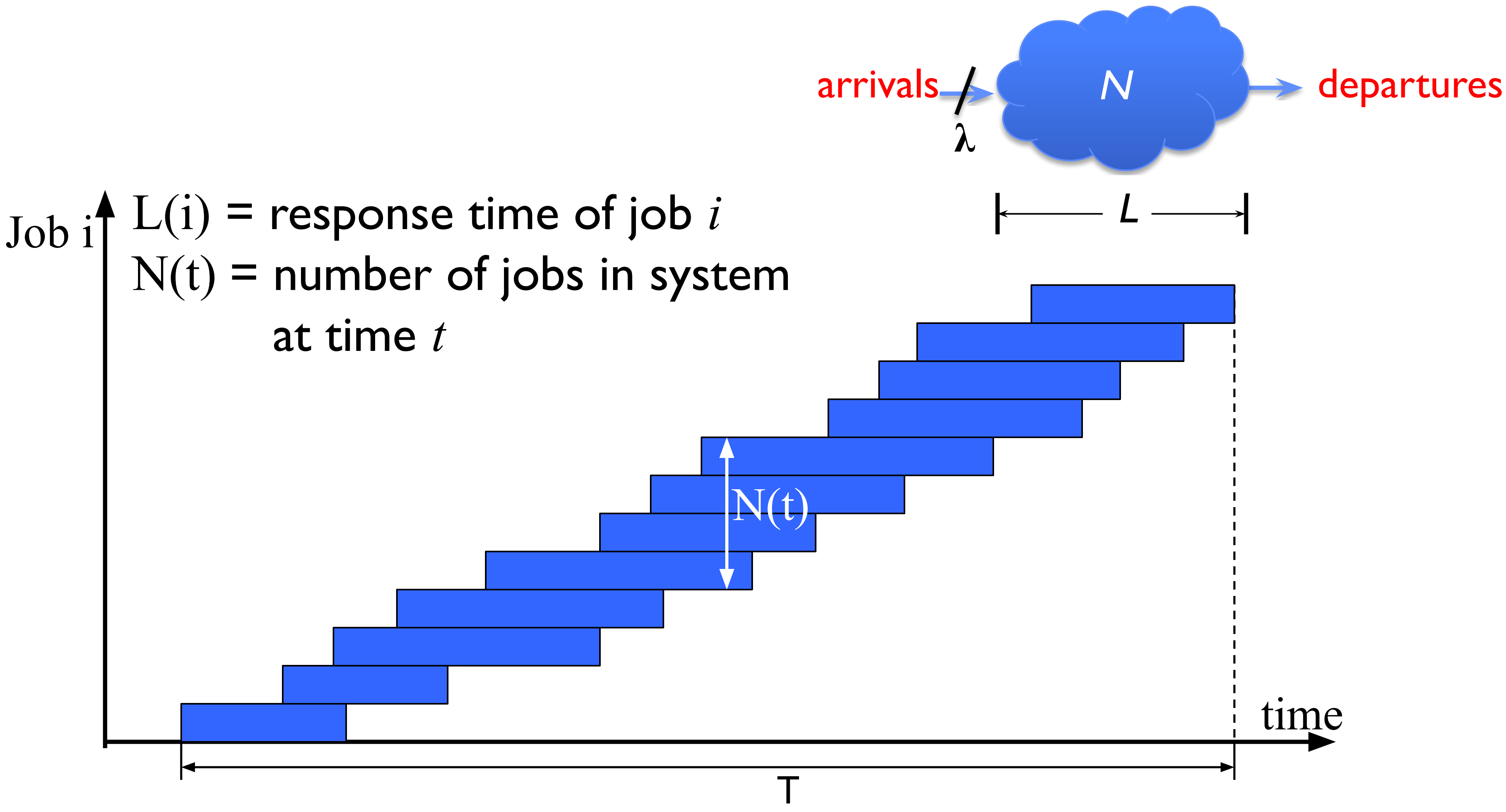
Example



Little's Theorem: Proof Sketch

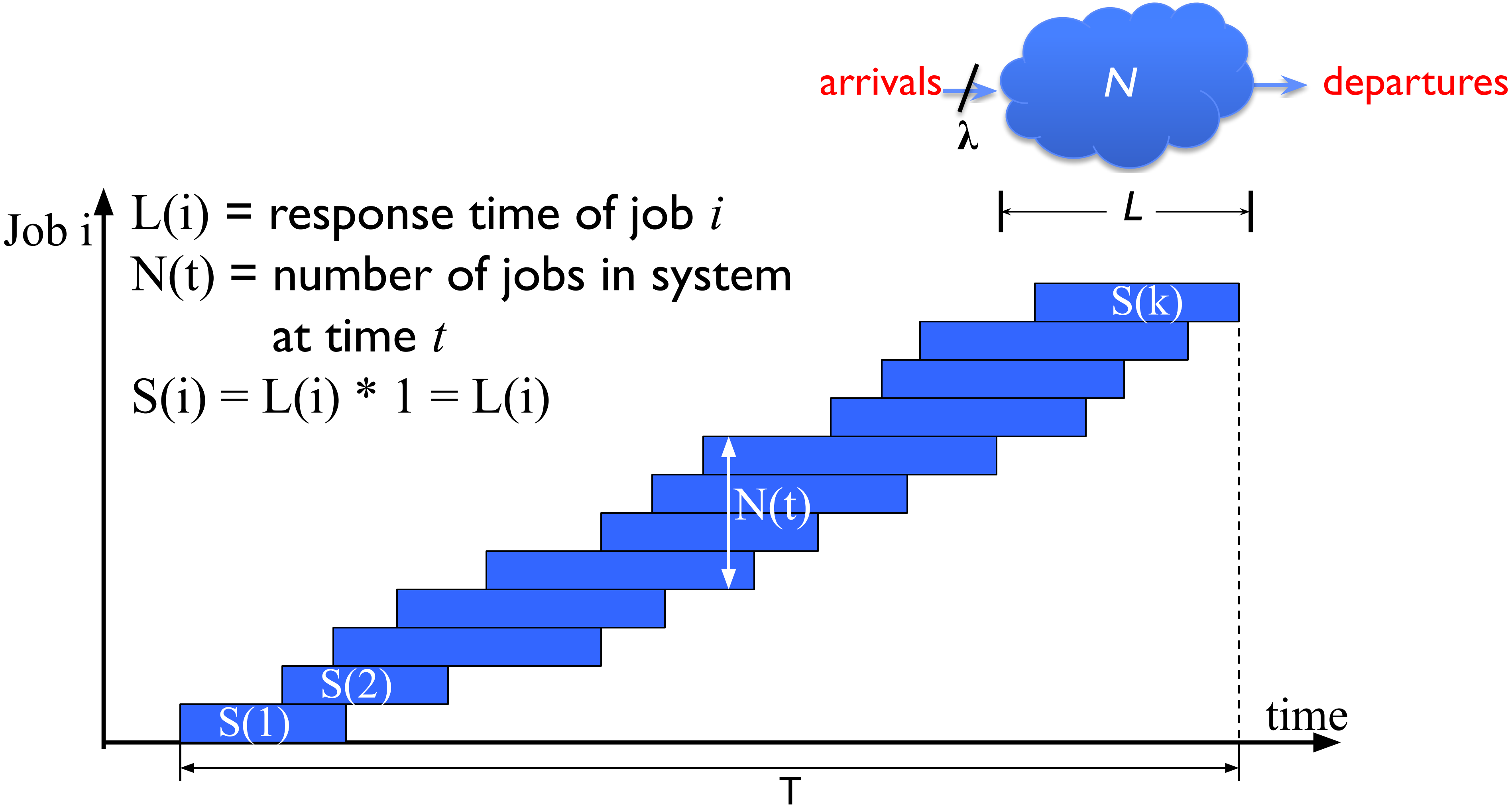


Little's Theorem: Proof Sketch



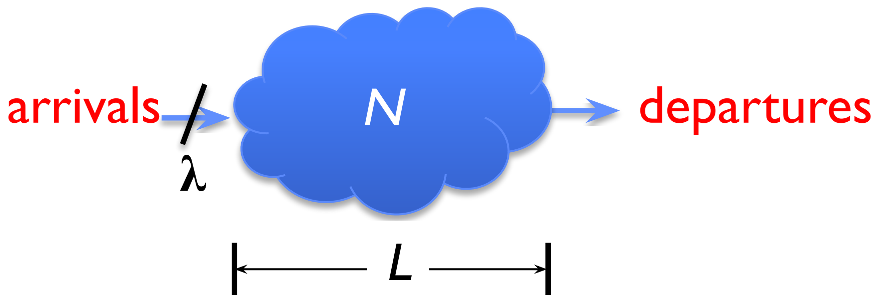
What is the system occupancy, i.e., average
number of jobs in the system?

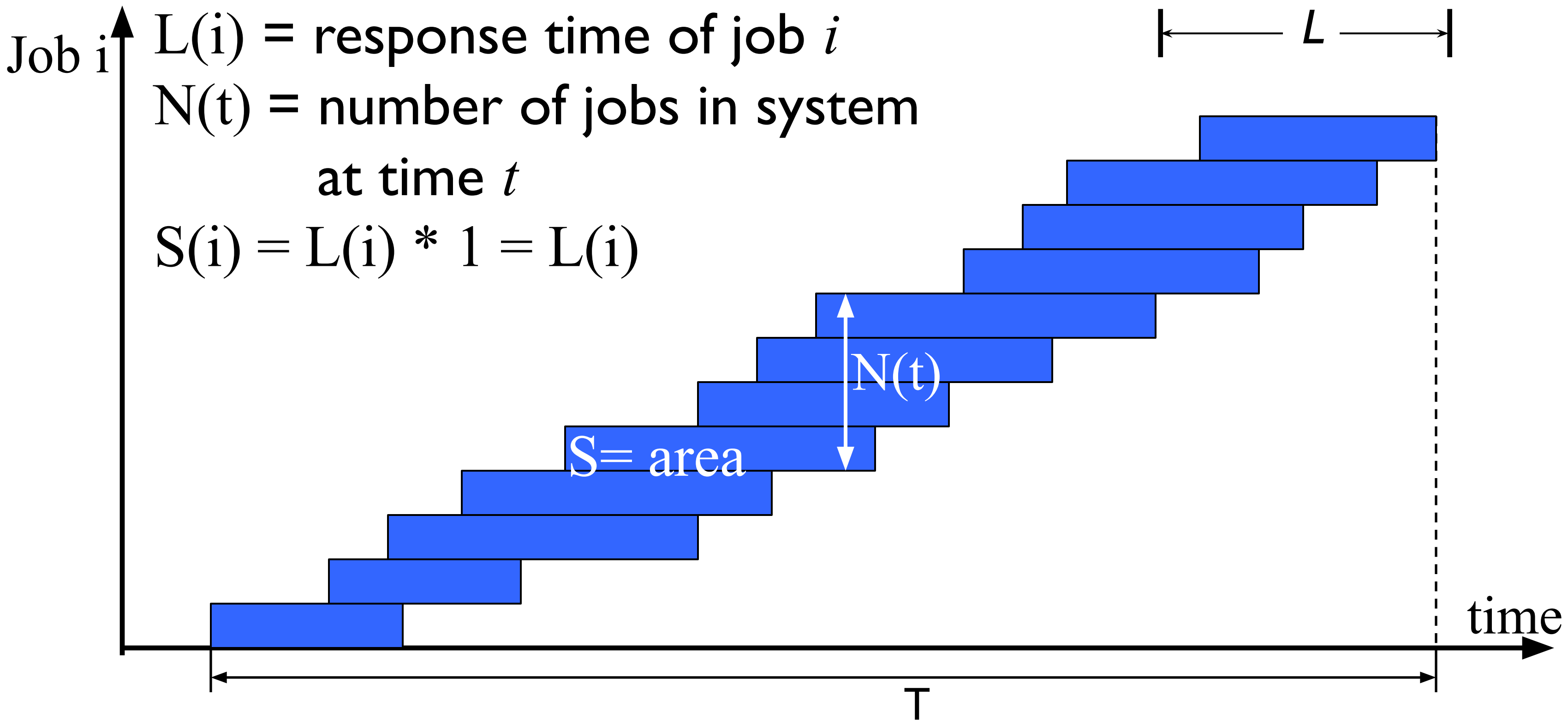
Little's Theorem: Proof Sketch



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

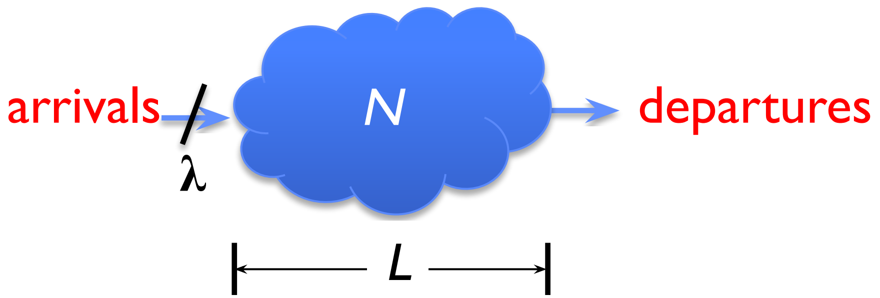
Little's Theorem: Proof Sketch





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

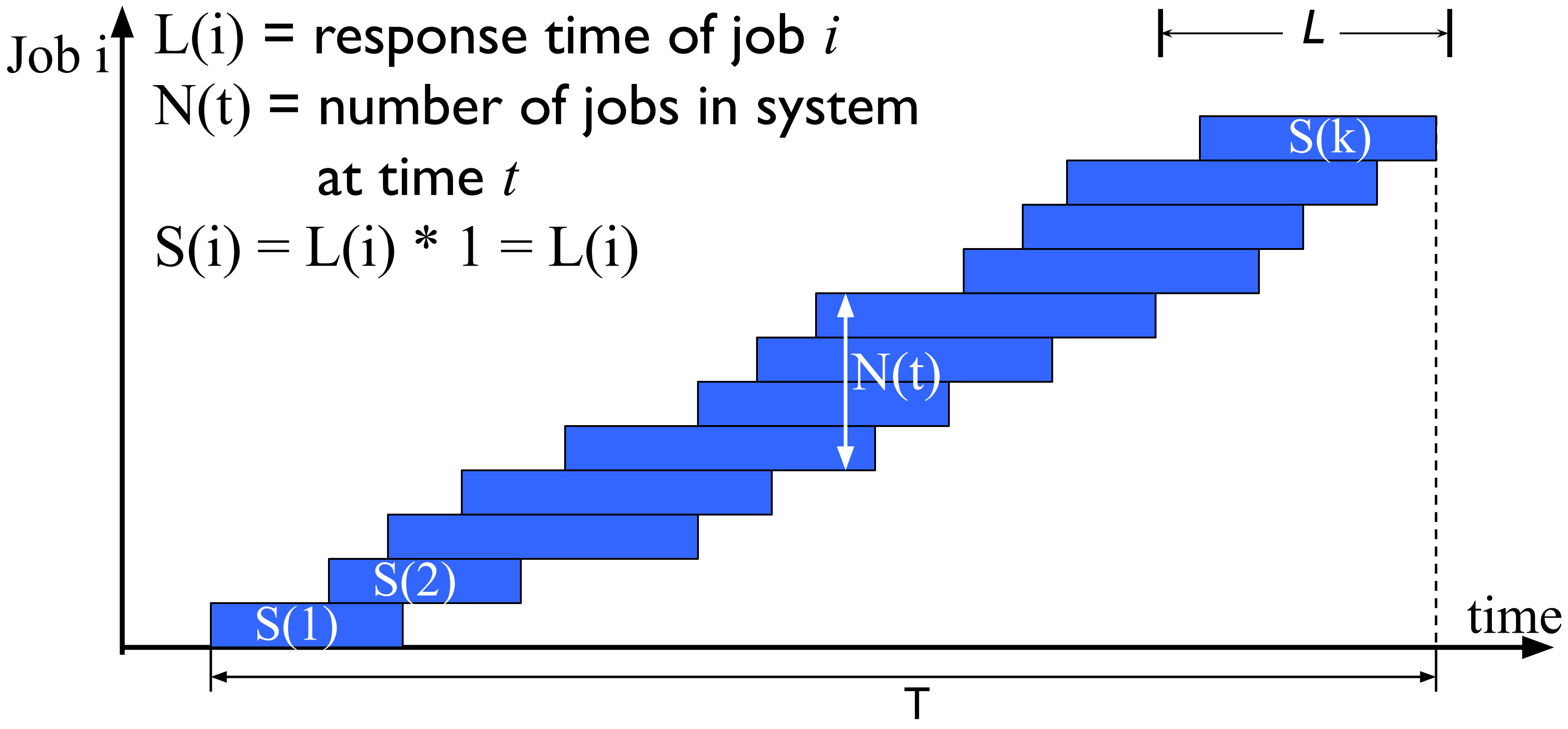
Little's Theorem: Proof Sketch



L(i) = response time of job i

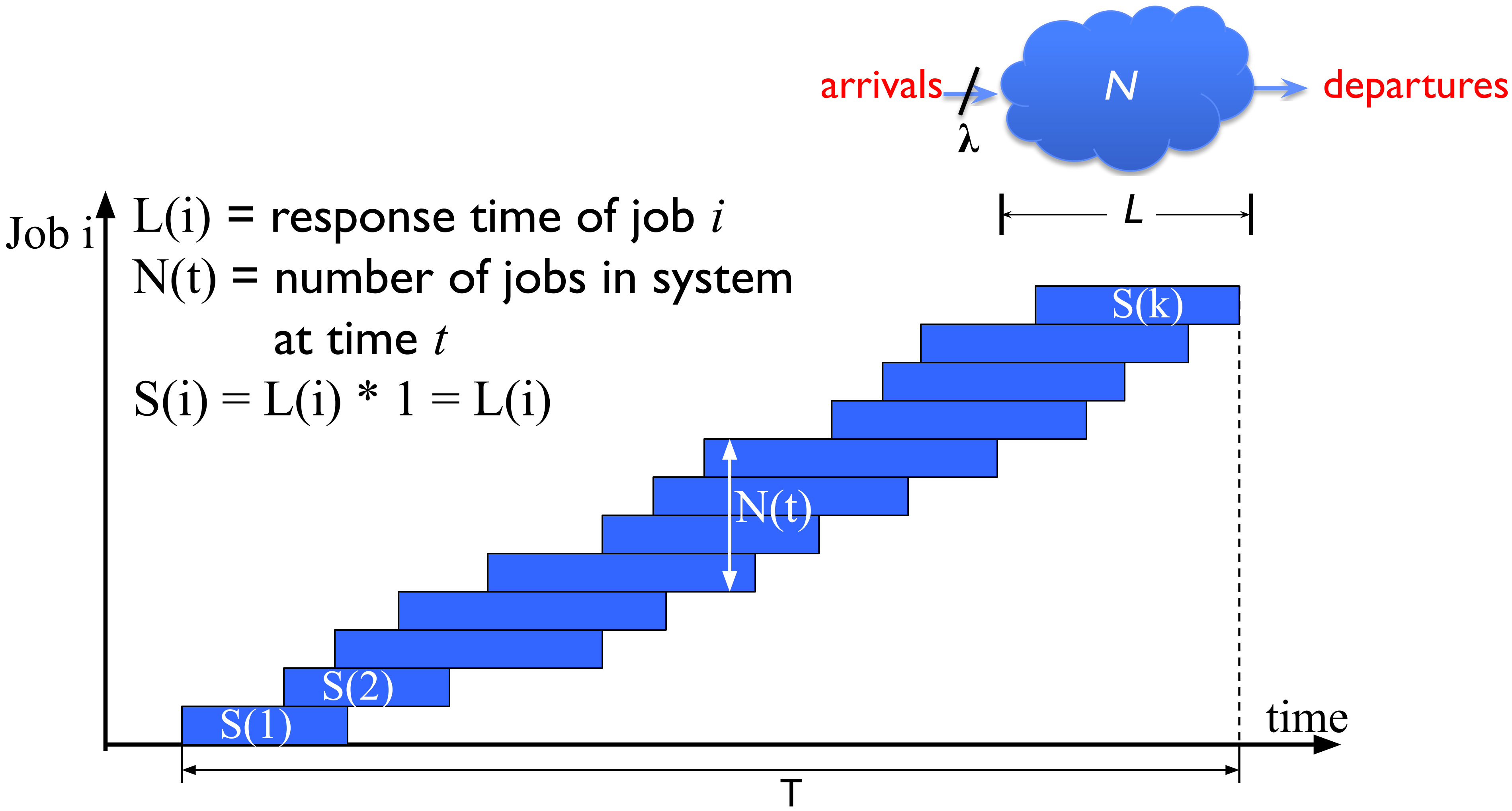
N(t) = number of jobs in system at time t

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



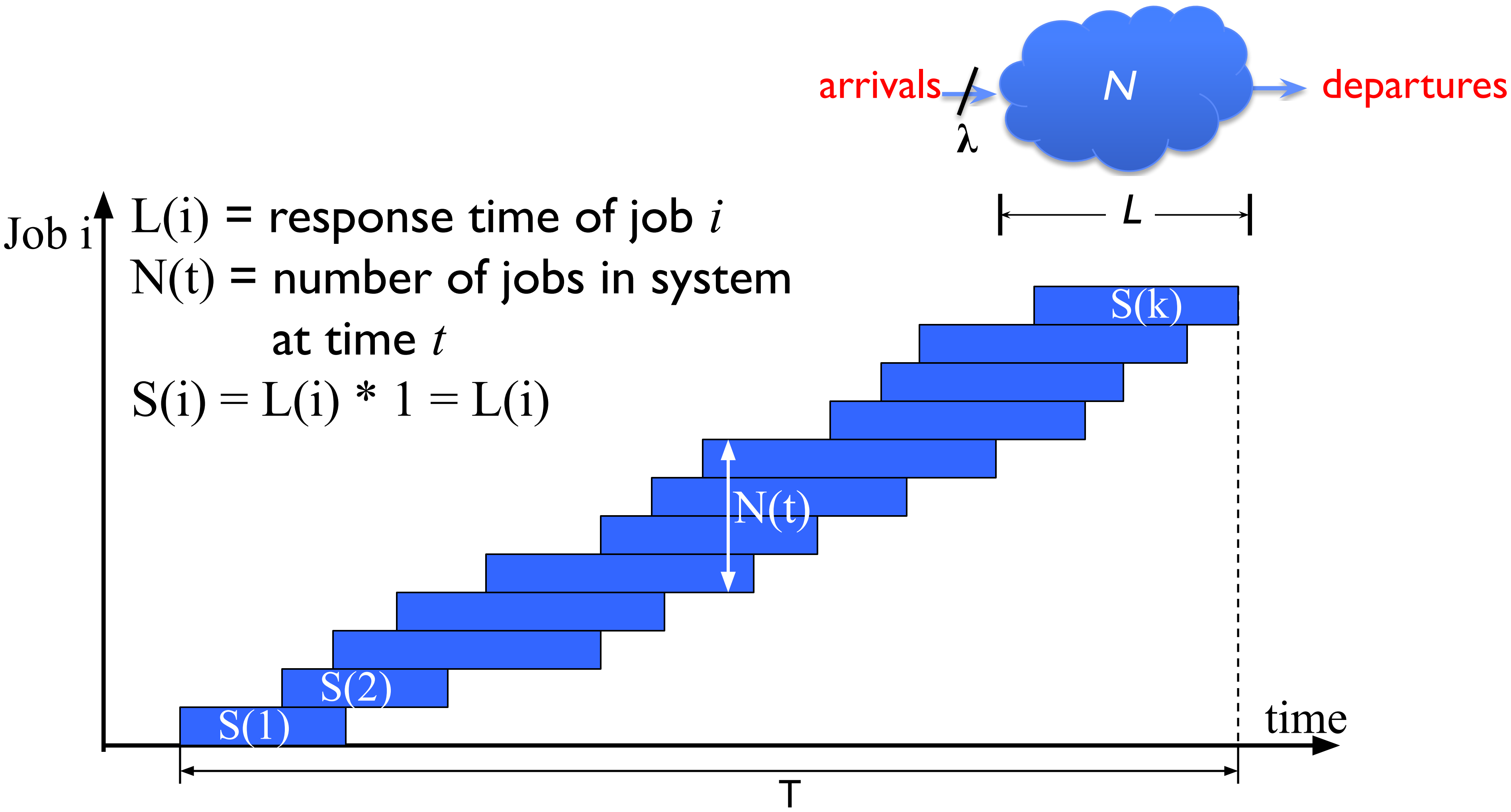
$$N_{avg} = S/T = (L(1) + \ldots + L(k))/T$$

Little's Theorem: Proof Sketch



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

Little's Theorem: Proof Sketch



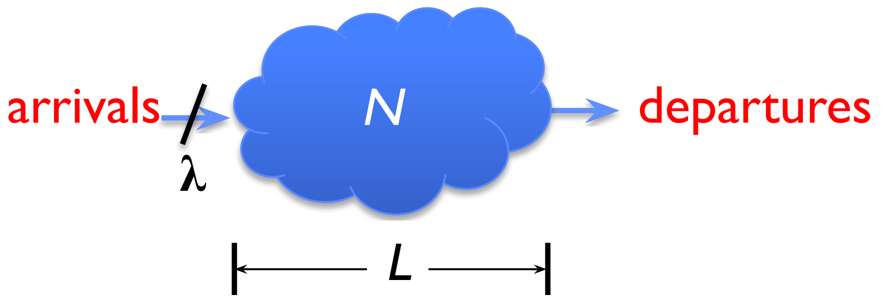
L(i) = response time of job *i*

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

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

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

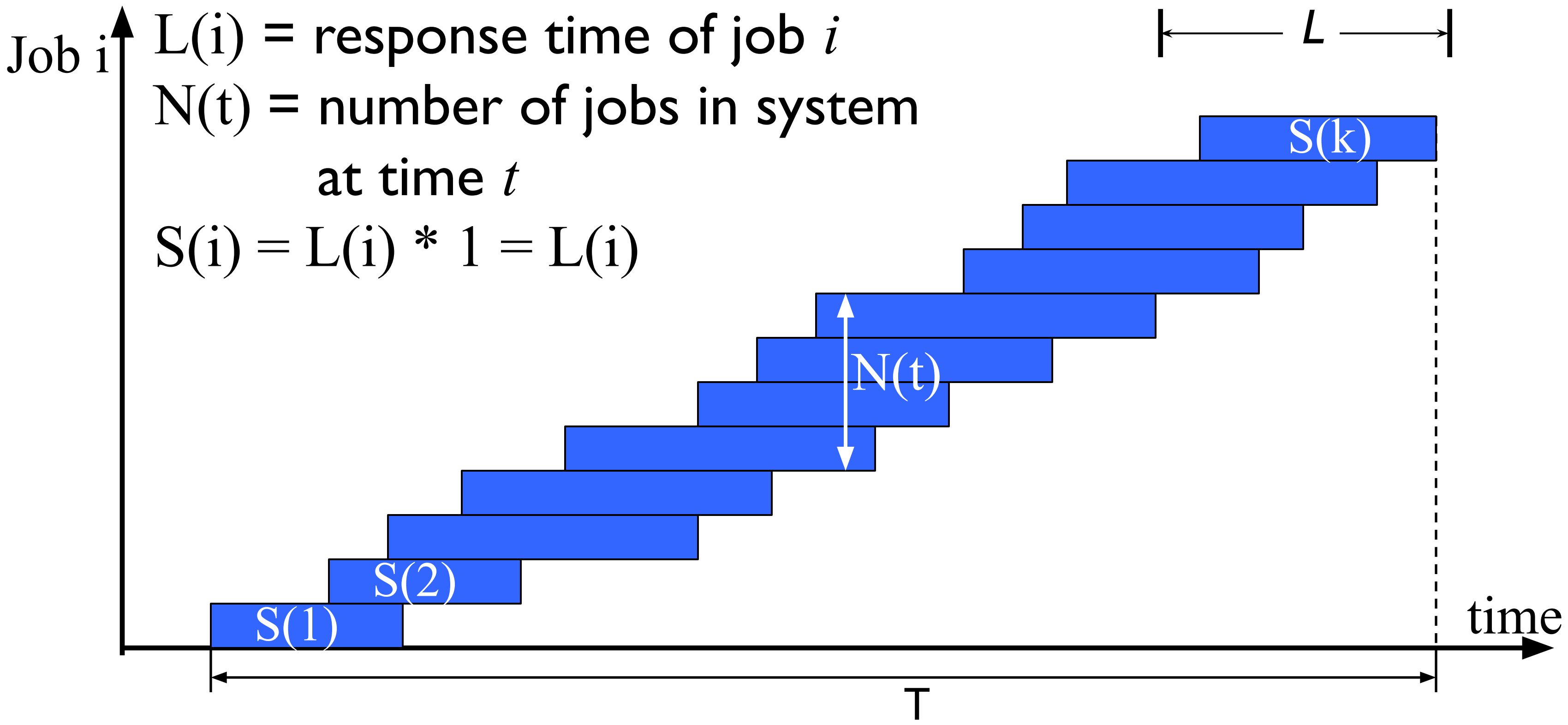
Little's Theorem: Proof Sketch



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

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

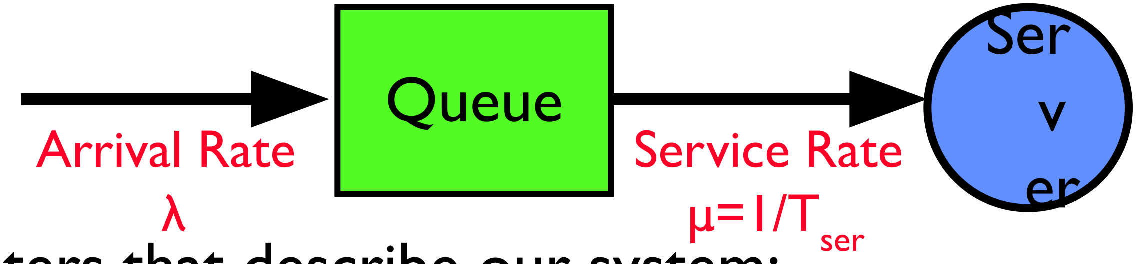
$S(i) = L(i) * 1 = L(i)$



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

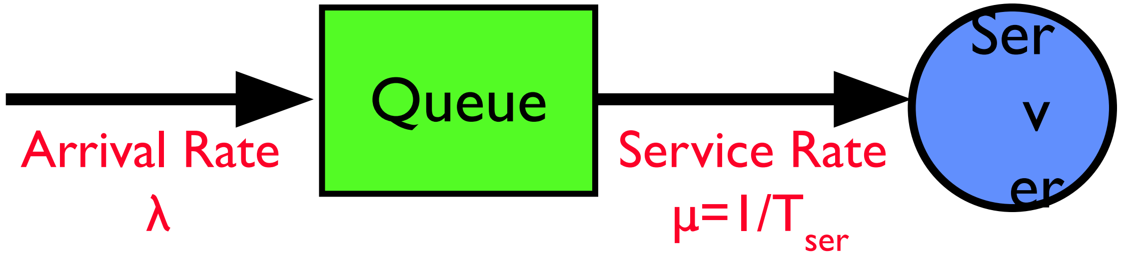
A Little Queuing Theory: Some Results (1/2)

- 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)



- 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 = σ^2/m^2
  - μ: service rate = $1/T_{ser}$
  - u: server utilization ($0 \leq u \leq 1$): $u = \lambda/\mu = \lambda \times T_{ser}$
- 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):
  - Memoryless service time distribution (C = 1): Called an M/M/1 queue
    - » $T_q = T_{ser} \times \boxed{u/(1-u)}$
  - General service time distribution (no restrictions): Called an M/G/1 queue
    - » $T_q = T_{ser} \times ½(1+C) \times \boxed{u/(1-u)}$

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 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 = 0.05s

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

- Queueing Theory is part of Midterm 2

Announcements

- Project 3 released today, Thursday, 14/11
- HW4, RPC Lab Deadline, Monday, 18/11

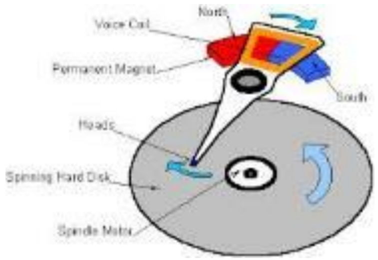
Recall: I/O and Storage Layers

Application / Service

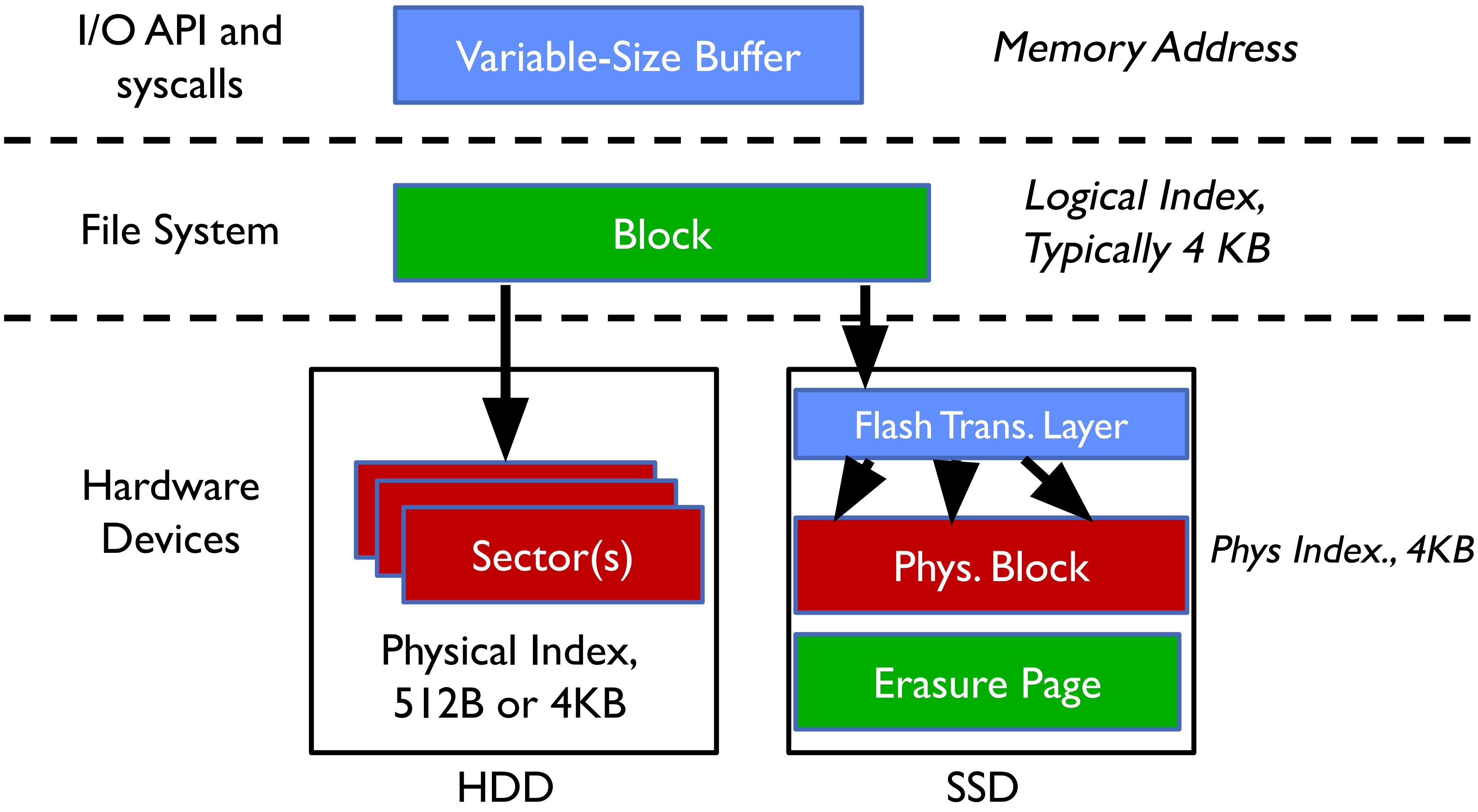
High Level I/O	*Streams*	What we covered in Lecture 4
Low Level I/O	*File Descriptors*	
Syscall	*open(), read(), write(), close(), …* *Open File Descriptions*	
File System	*Files/Directories/Indexes*	What we will cover next…
I/O Driver	*Commands and Data Transfers*	What we just covered…
	Disks, Flash, Controllers, DMA	







From Storage to File Systems



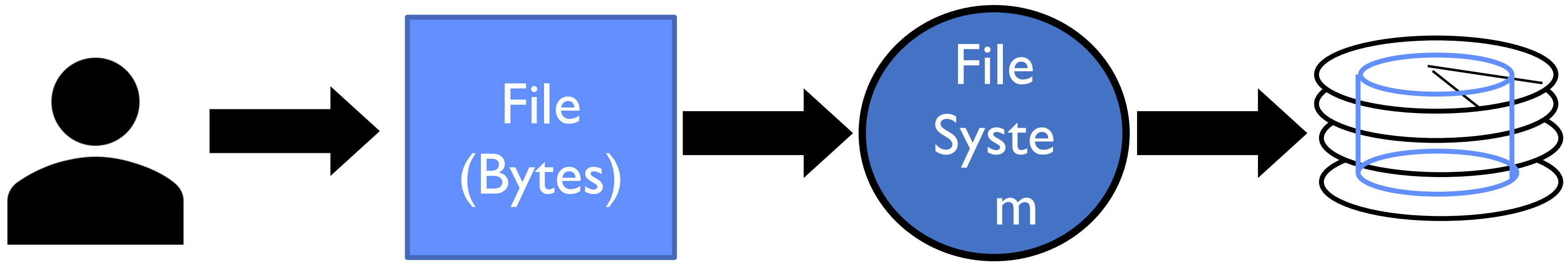
Building a File System

- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.
- Classic OS situation: Take limited hardware interface (array of blocks) and provide a more convenient/useful interface with:
  - Naming: Find file by name, not block numbers
  - Organize file names with directories
  - Organization: Map files to blocks
  - Protection: Enforce access restrictions
  - Reliability: Keep files intact despite crashes, hardware failures, etc.

Recall: User vs. System View of a File

- User's view:
  - Durable Data Structures
- System's view (system call interface):
  - Collection of Bytes (UNIX)
  - Doesn't matter to system what kind of data structures you want to store on disk!
- System's view (inside OS):
  - Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
  - Block size ≥ sector size; in UNIX, block size is 4KB

Translation from User to System View



- What happens if user says: "give me bytes 2 – 12?"
  - Fetch block corresponding to those bytes
  - Return just the correct portion of the block
- What about writing bytes 2 – 12?
  - Fetch block, modify relevant portion, write out block
- Everything inside file system is in terms of whole-size blocks
  - Actual disk I/O happens in blocks
  - read/write smaller than block size needs to translate and buffer

Disk Management

- Basic entities on a disk:
  - File: user-visible group of blocks arranged sequentially in logical space
  - Directory: user-visible index mapping names to files

- The disk is accessed as linear array of sectors
- How to identify a sector?
  - Physical position
    - » Sectors is a vector [cylinder, surface, sector]
    - » Not used anymore
    - » OS/BIOS must deal with bad sectors
  - Logical Block Addressing (LBA)
    - » Every sector has integer address
    - » Controller translates from address $\Rightarrow$ physical position
    - » Shields OS from structure of disk

What Does the File System Need?

- Track free disk blocks
  - Need to know where to put newly written data
- Track which blocks contain data for which files
  - Need to know where to read a file from
- Track files in a directory
  - Find list of file's blocks given its name
- Where do we maintain all of this?
  - Somewhere on disk

Data Structures on Disk

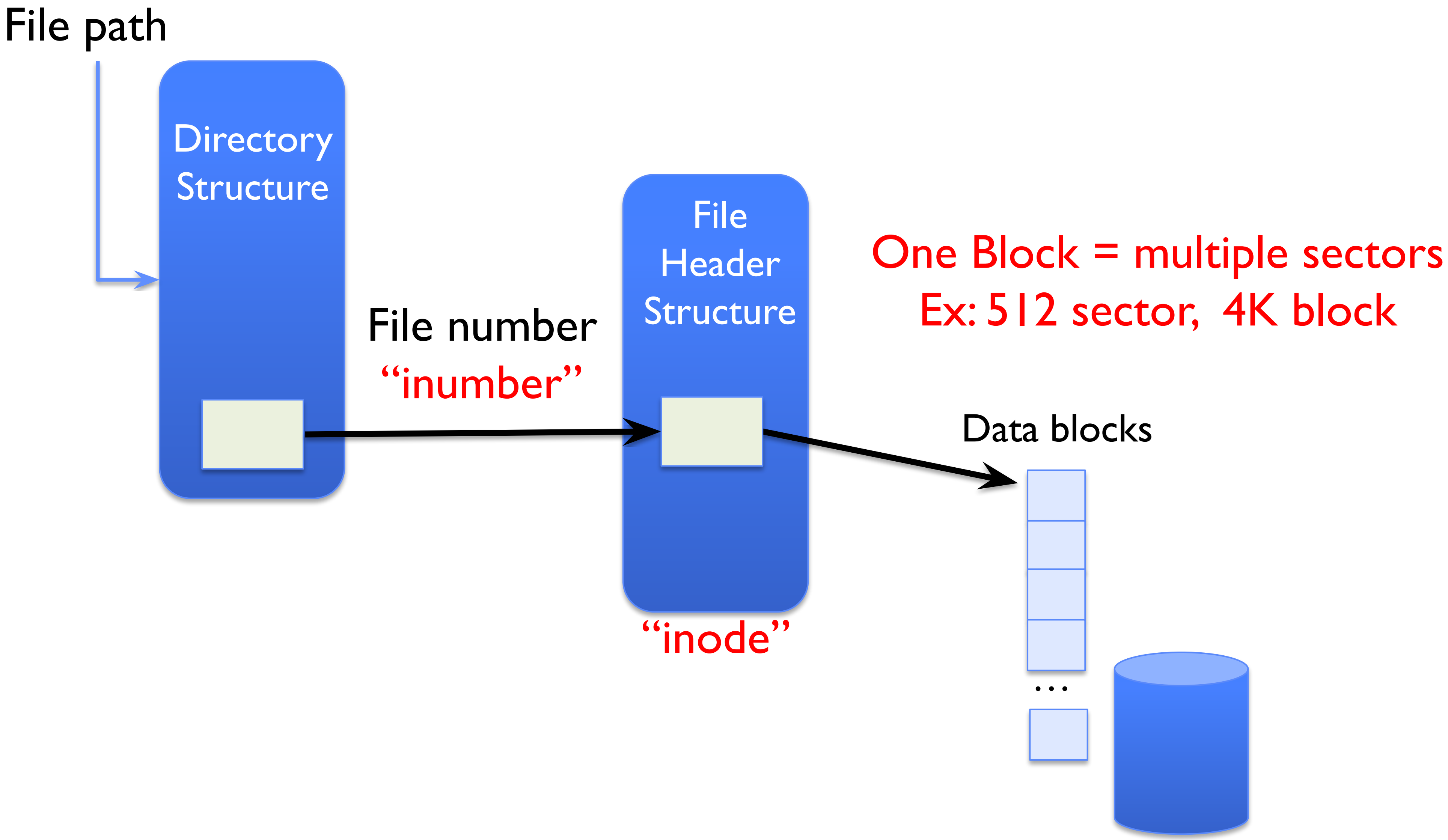
- Somewhat different from data structures in memory
- Access a block at a time
  - Can't efficiently read/write a single word
  - Have to read/write full block containing it
  - Ideally want sequential access patterns

- Durability
  - Ideally, file system is in meaningful state upon shutdown
  - This obviously isn't always the case…

Critical Factors in File System Design

- (Hard) Disks Performance !!!
  - Maximize sequential access, minimize seeks
- Open before Read/Write
  - Can perform protection checks and look up where the actual file resource are, in advance
- Size is determined as they are used !!!
  - Can write (or read zeros) to expand the file
  - Start small and grow, need to make room
- Organized into directories
  - What data structure (on disk) for that?
- Need to carefully allocate / free blocks
  - Such that access remains efficient

Components of a File System



Conclusion

- 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
  - SSD: Queuing time + controller + transfer (erasure & wear)
- Bursts & High Utilization introduce queuing delays
- Queuing Latency:
  - M/M/1 and M/G/1 queues: simplest to analyze
  - As utilization approaches 100%, latency → ∞

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

- File System:
  - Transforms blocks into Files and Directories
  - Optimize for access and usage patterns
  - Maximize sequential access, allow efficient random access