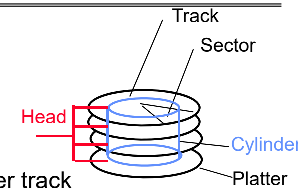
CS162
Operating Systems and Systems Programming
Lecture 19

Filesystems 1: Performance (Con't), Queueing Theory, Filesystem Design

November 2nd, 2020

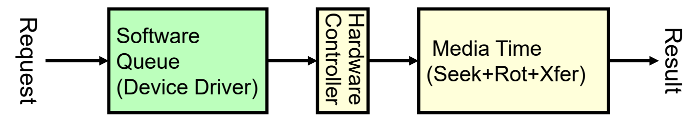
Prof. John Kubiatowicz
http://cs162.eecs.Berkeley.edu

Recall: Magnetic Disks



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

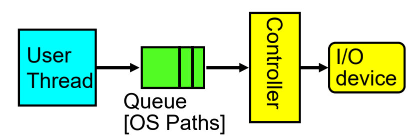
Disk Latency = Queueing Time + Controller time + Seek Time + Rotation Time + Xfer Time



Recall: Typical Numbers for Magnetic Disk

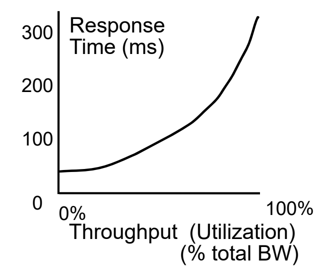
Parameter	Info/Range
Space/Density	Space: 18TB (Seagate), 9 platters, in 3½ inch form factor! **Areal Density: ≥ 1 Terabit/square inch! (PMR, Helium, …)**
Average Seek Time	Typically 4-6 milliseconds
Average Rotational Latency	Most laptop/desktop disks rotate at 3600-7200 RPM (16-8 ms/rotation). Server 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

Recall: Overall Performance for I/O Path

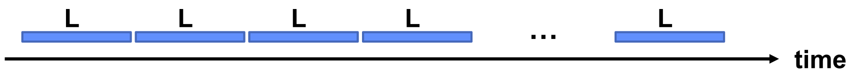


Response Time = Queue + I/O device service time

- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Effective BW = transfer size / response time
  - 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?

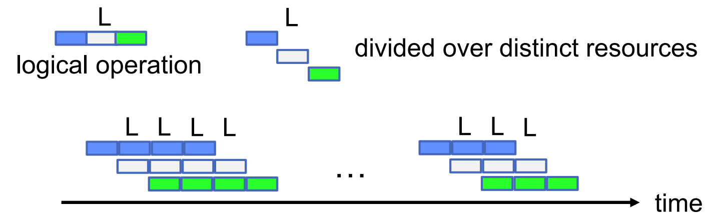


Sequential Server Performance



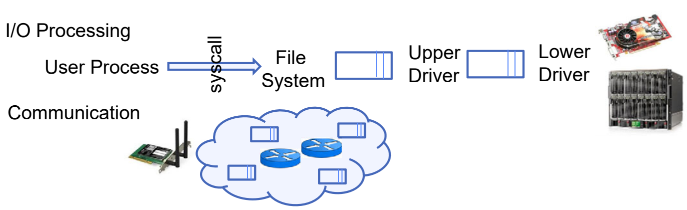
- Single sequential "server" that can complete a task in time L operates at rate $\leq \frac{1}{L}$ (on average, in steady state, …)
  - $L = 10$ ms $\rightarrow B = 100$ $^{\text{op}}/_{\text{s}}$
  - $L = 2$ yr $\rightarrow B = 0.5$ $^{\text{op}}/_{\text{yr}}$
- Applies to a processor, a disk drive, a person, a TA, …

Single Pipelined Server



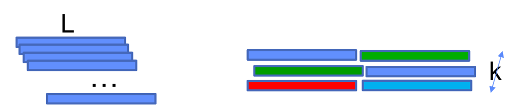
- Single pipelined server of k stages for tasks of length L (i.e., time $^{L}/_{k}$ per stage) delivers at rate $\leq {}^{k}/_{L}$.
  - $L = 10$ ms, $k = 4 \rightarrow B = 400$ $^{\text{op}}/_{\text{s}}$
  - $L = 2$ yr, $k = 2 \rightarrow B = 1$ $^{\text{op}}/_{\text{yr}}$

Example Systems "Pipelines"



- Anything with queues between operational process behaves roughly "pipeline like"
- Important difference is that "initiations" are decoupled from processing
  - May have to queue up a burst of operations
  - Not synchronous and deterministic like in 61C

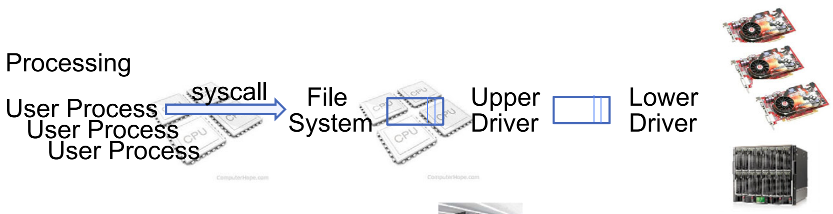
Multiple Servers



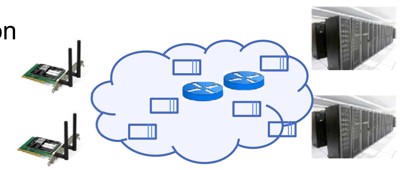
- k servers handling tasks of length L delivers at rate $\leq {}^{k}/_{L}$.
  - $L = 10$ ms, $k = 4 \rightarrow B = 400$ $^{\text{op}}/_{\text{s}}$
  - $L = 2$ yr, $k = 2 \rightarrow B = 1$ $^{\text{op}}/_{\text{yr}}$
- In 61C you saw multiple processors (cores)
  - Systems present lots of multiple parallel servers
  - Often with lots of queues

Example Systems "Parallelism"

I/O Processing



Communication

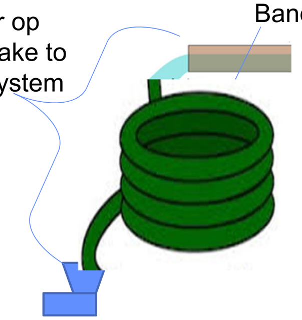


Parallel Computation, Databases, …

A Simple Systems Performance Model

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

"Service Time"



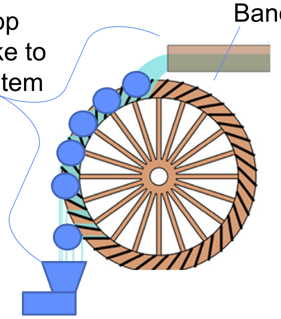
Bandwidth (B): Rate, Op/s
e.g., flow: gal per min

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

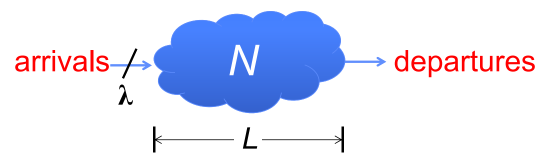
"Service Time"



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

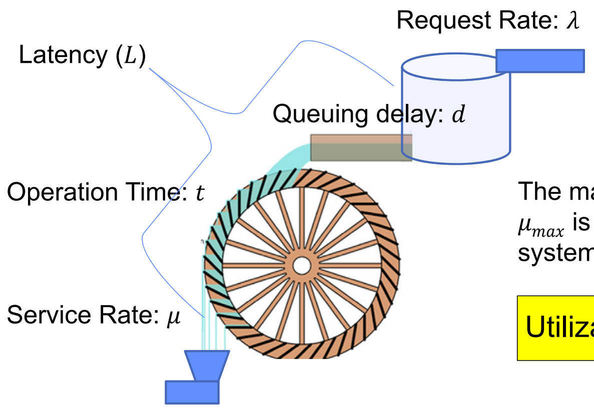
If $B = 2\ ^{op}/_{s}$ and $L = 3$ s
How many ops are "in the system?"

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

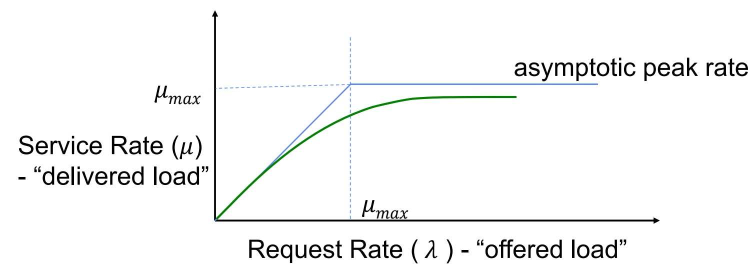


The maximum service rate μ_{max} is a property of the system – the "bottleneck"

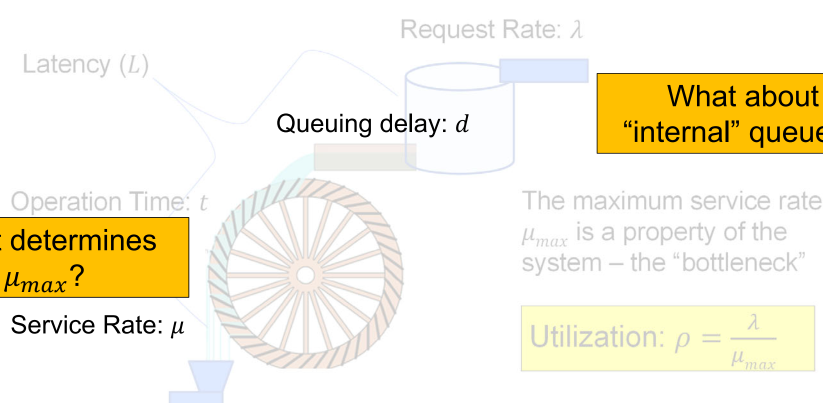
Utilization: $\rho = \frac{\lambda}{\mu_{max}}$

Ideal System Performance

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



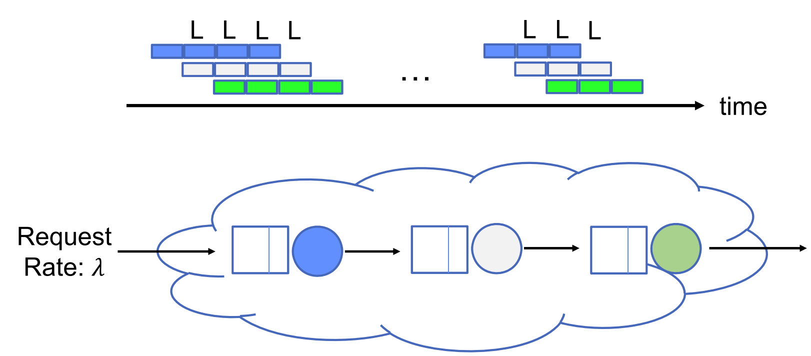
Two Related Questions



What about "internal" queues?

What determines μ_{max}?

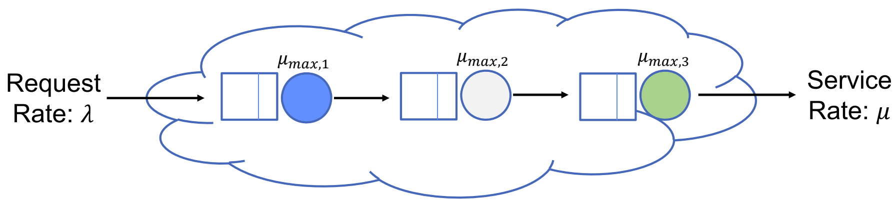
Bottleneck Analysis



Overall System: Series of Stages

Bottleneck Analysis

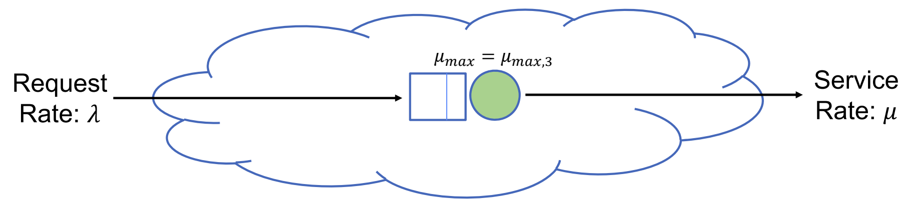
- Each stage has its own queue and maximum service rate
- Suppose the green stage is the bottleneck



Overall System: Series of Stages

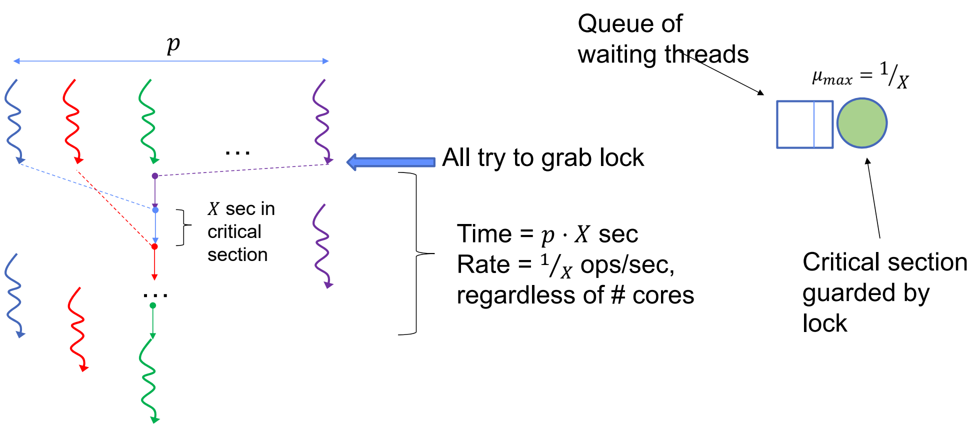
Bottleneck Analysis

- 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}

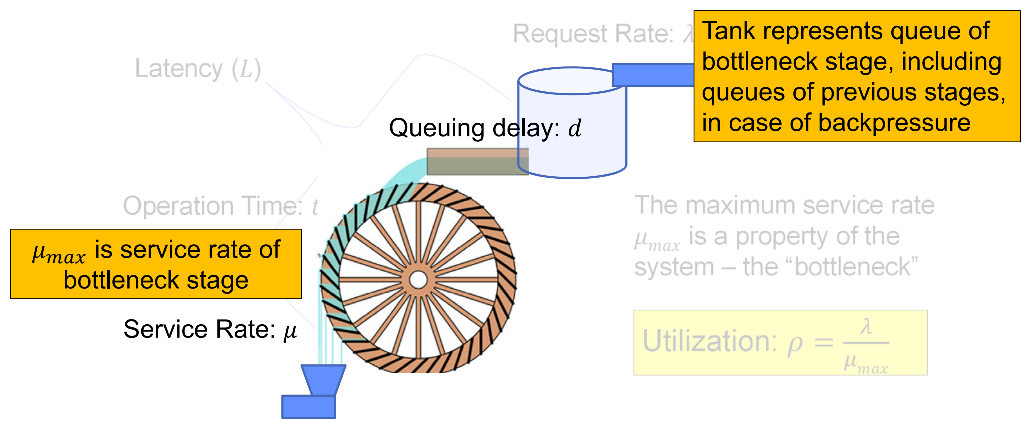


System Model: Bottleneck Stage

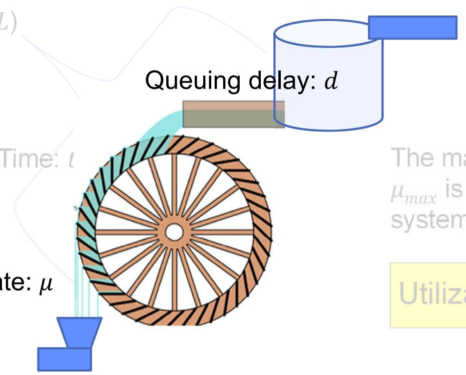
Example: Servicing a Highly Contended Lock



Two Related Questions



A Simple Systems Performance Model

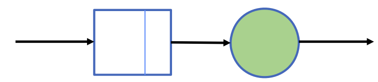


Useful to apply this model to:
- Bottleneck stage
- Entire system up to and including bottleneck stage
- Entire system

The ma[...] μ_{max} is a property of the system – the "bottleneck"

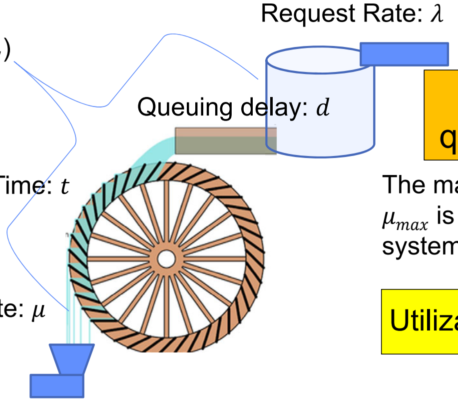
Utilization: $\rho = \frac{\lambda}{\mu_{max}}$

Latency (Response Time)



- Total latency (response time): queuing time + service time
- Service time depends on the underlying operation
  - For CPU stage, how much computation
  - For I/O stage, characteristics of the hardware
- What about the queuing time?

A Simple Systems Performance Model



When will the queue(s) start to fill?

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

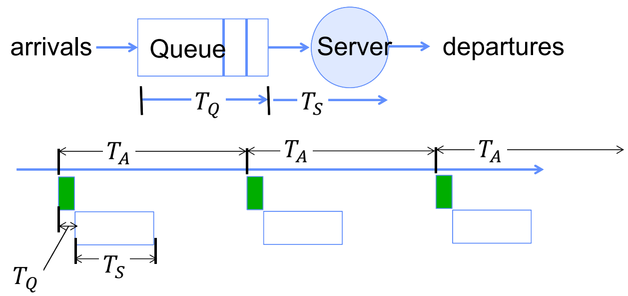
Utilization: $\rho = \frac{\lambda}{\mu_{max}}$

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

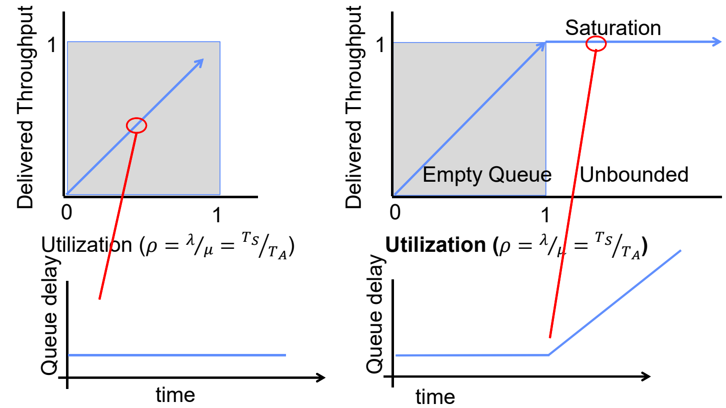
A Simple, Deterministic World

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



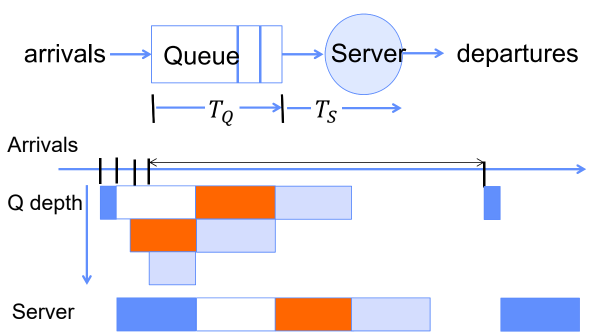
- Assume requests arrive at regular intervals, take a fixed time to process, with plenty of time between …

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
  - $L = T_Q + T_S$

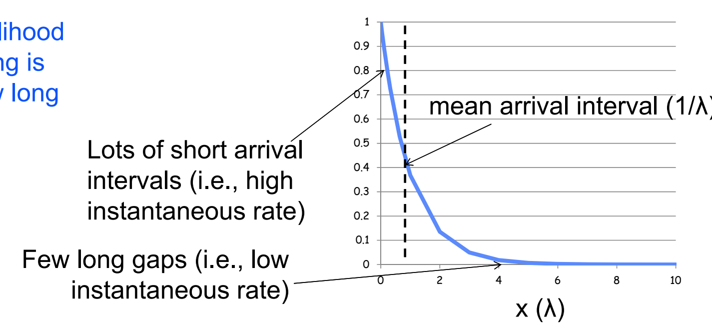


- Requests arrive in a burst, must queue up until served
- Same average arrival time, but almost all of the requests experience large queue delays (even though average utilization is low)

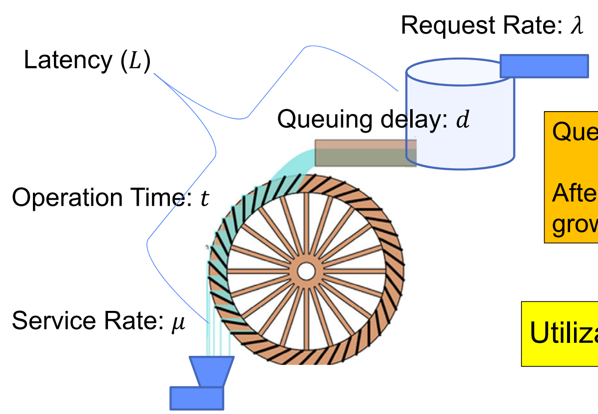
How to model burstiness of arrival?

- 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



A Simple Systems Performance Model



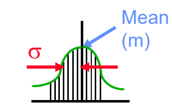
Queue grows at rate $\mu - \lambda$

After time t , it will have grown to length $t(\mu - \lambda)$

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

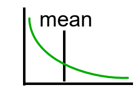
Background: Random Distributions

- Server spends variable time (T) with customers
  - Mean (Average): $m = \sum p(T) \cdot T$
  - Variance (stddev²): $\sigma^2 = \sum p(T) \cdot (T - m)^2 = \sum p(T) \cdot T^2 - m^2$
  - Squared coefficient of variance: $C = \sigma^2 / m^2$



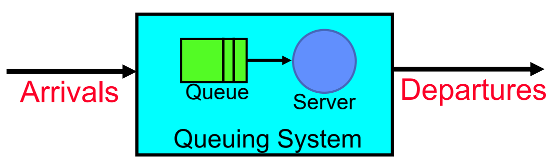
Distribution of service times

- Important values of C:
  - No variance or deterministic $\Rightarrow C = 0$
  - "Memoryless" or exponential $\Rightarrow C = 1$
    - » Past tells nothing about future
    - » Poisson process – *purely* or *completely* random process
    - » Many complex systems (or aggregates) are well described as memoryless



Memoryless

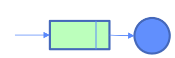
Steady State Queuing Theory



- 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:



Average Arrival Rate

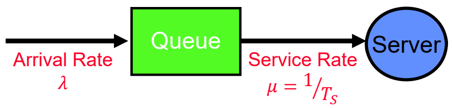
$$L_Q = \lambda T_Q$$

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 (σ^2/T_S^2)
- μ: service rate ($1/T_S$)
- ρ: utilization (λ/μ)

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)

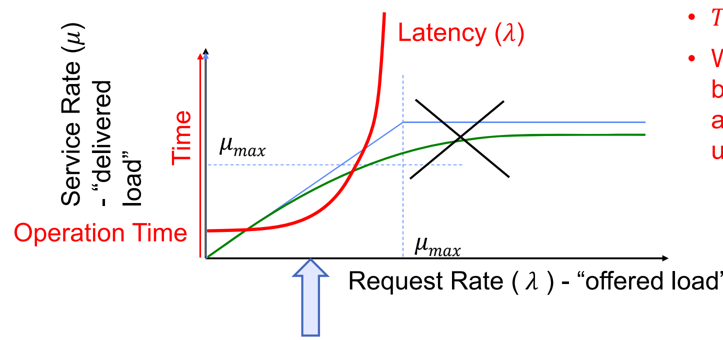
Some Results from Queuing Theory (con't)

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

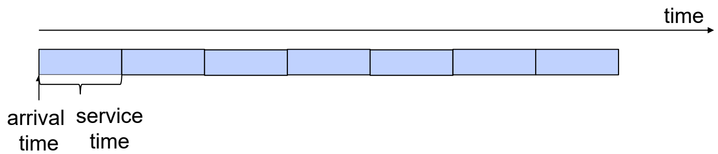


- $T_Q \sim \frac{\rho}{1-\rho}$, $\rho = \lambda/\mu_{max}$
- Why does latency blow up as we approach 100% utilization?
  - Queue builds up on each burst
  - But very rarely (or never) gets a chance to drain

"Half-Power Point" : load at which system delivers half of peak performance
- Design and provision systems to operate roughly in this regime
- Latency low and predictable, utilization good: ~50%

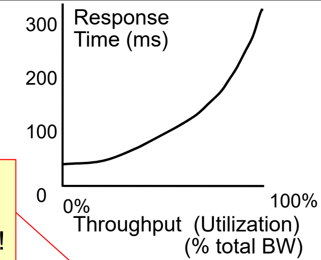
Why unbounded response time?

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

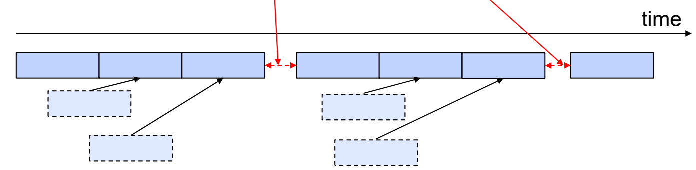


Why unbounded response time?

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



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



A Little Queuing Theory: An Example

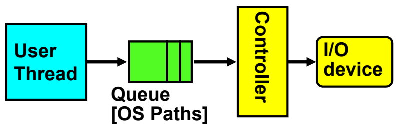
- Example Usage Statistics:
  - User requests 10 x 8KB disk I/Os per second
  - Requests & service exponentially distributed (C=1.0)
  - Avg. service = 20 ms (From controller+seek+rot+trans)
- Questions:
  - How utilized is the disk?
    - » Ans: server utilization, $\rho = \lambda T_{ser}$
  - What 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:

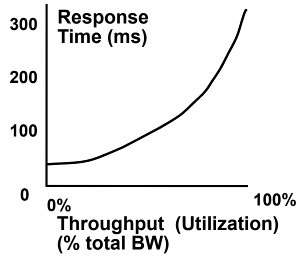
λ (avg # arriving customers/s) = 10/s
T_{ser} (avg time to service customer) = 20 ms (0.02s)
ρ (server utilization) = $\lambda \times T_{ser}$= 10/s x .02s = 0.2
T_q (avg time/customer in queue) = $T_{ser} \times u/(1 - u)$
= 20 x 0.2/(1-0.2) = 20 x 0.25 = 5 ms (0 .005s)
L_q (avg length of queue) = $\lambda \times T_q$=10/s x .005s = 0.05
T_{sys} (avg time/customer in system) =$T_q + T_{ser}$= 25 ms

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!

Optimize I/O Performance





Response Time = Queue + I/O device service time

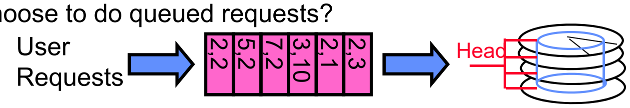
- How to improve performance?
  - Make everything faster ☺
  - More Decoupled (Parallelism) systems
    - » multiple independent buses or controllers
  - Optimize the bottleneck to increase service rate
    - » Use the queue to optimize the service
  - Do other useful work while waiting
- Queues absorb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

When is Disk Performance Highest?

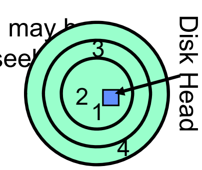
- When there are big sequential reads, or
- When there is so much work to do that they can be piggy backed (reordering queues—one moment)
- OK to be inefficient when things are mostly idle
- Bursts are both a threat and an opportunity
- <your idea for optimization goes here>
  - Waste space for speed?
- Other techniques:
  - Reduce overhead through user level drivers
  - Reduce the impact of I/O delays by doing other useful work in the meantime

Disk Scheduling (1/3)

- Disk can do only one request at a time; What order do you choose to do queued requests?

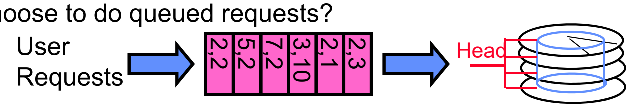


- FIFO Order
  - Fair among requesters, but order of arrival may be to random spots on the disk ⇒ Very long seeks
- SSTF: Shortest seek time first
  - Pick the request that's closest on the disk
  - Although called SSTF, today must include rotational delay in calculation, since rotation can be as long as seek
  - Con: SSTF good at reducing seeks, but may lead to starvation

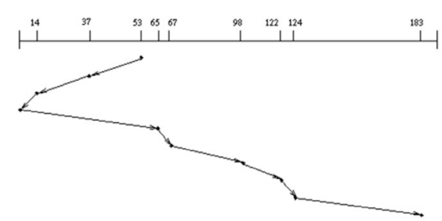


Disk Scheduling (2/3)

- Disk can do only one request at a time; What order do you choose to do queued requests?

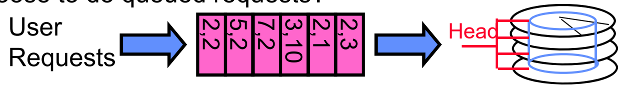


- SCAN: Implements an Elevator Algorithm: take the closest request in the direction of travel
  - No starvation, but retains flavor of SSTF

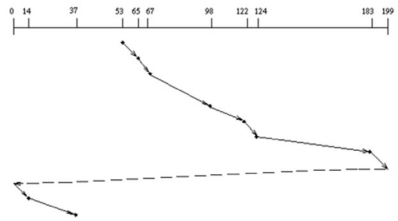


Disk Scheduling (3/3)

- Disk can do only one request at a time; What order do you choose to do queued requests?



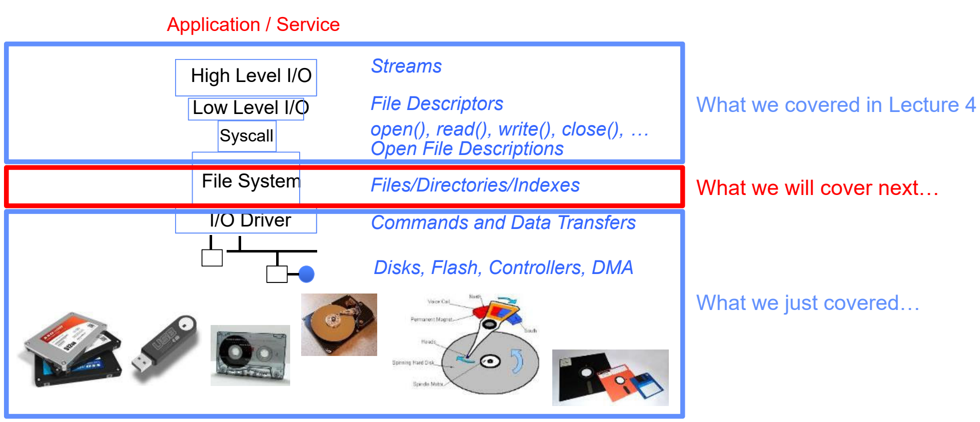
- C-SCAN: Circular-Scan: only goes in one direction
  - Skips any requests on the way back
  - Fairer than SCAN, not biased towards pages in middle



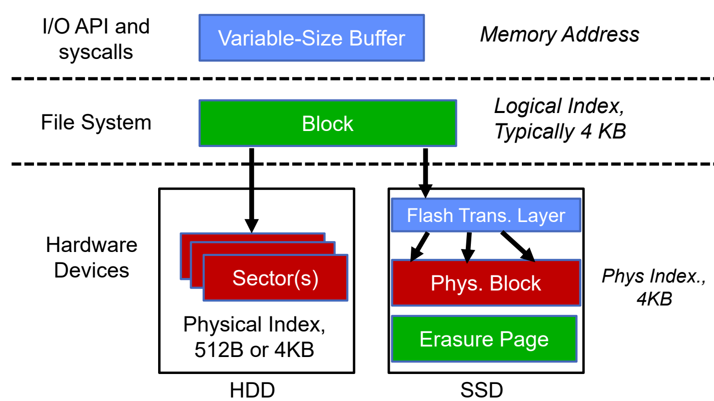
Recall: How do we Hide I/O Latency?

- Blocking Interface: "Wait"
  - When request data (*e.g.,* read() system call), put process to sleep until data is ready
  - When write data (*e.g.,* write() system call), put process to sleep until device is ready for data
- Non-blocking Interface: "Don't Wait"
  - Returns quickly from read or write request with count of bytes successfully transferred to kernel
  - Read may return nothing, write may write nothing
- Asynchronous Interface: "Tell Me Later"
  - When requesting data, take pointer to user's buffer, return immediately; later kernel fills buffer and notifies user
  - When sending data, take pointer to user's buffer, return immediately; later kernel takes data and notifies user

Recall: I/O and Storage Layers



From Storage to File Systems



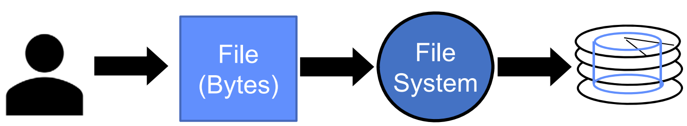
Building a File System

- 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 ⇒ 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**

Conclusion

- 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 $\rightarrow \infty$

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