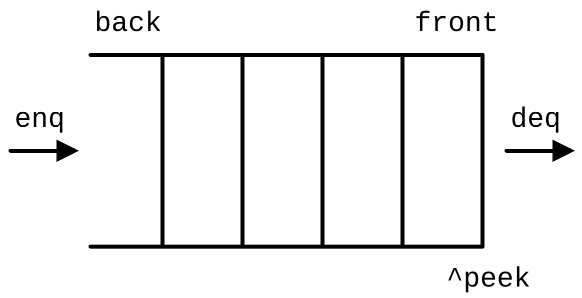
Queueing

A brief survey on the theory of waiting in line with computers, software, and systems

Meta

- About ~25 minutes
- Quick introduction into queueing theory and fundamentals
- Breadth-first survey of (hopefully) interesting topics
- Case studies and examples of queueing systems
- We'll skip most of the math, probability, derivations, proofs, etc.
- ...but provide references to resources where you can learn more!

What's a queue?



- *aka* a line! e.g. grocery store, theme parks, etc.
- Collection of entities: messages, people, requests
- First in, first out (FIFO) or first come, first serve (FCFS)
- Basic operations: enqueue, dequeue, peek

Where are queues?

Everywhere! e.g.

- Computers: CPU scheduling
- Disks and databases: write-ahead logs (WAL)
- Networking: packet queues
- Web servers: connection queues
- Customer support: ticketing systems
- Manufacturing: assembly lines
- Project management: work queues
- Utilities: water/plumbing
- Day-to-day: grocery lines, banks, traffic, DMV...

A day at the bank

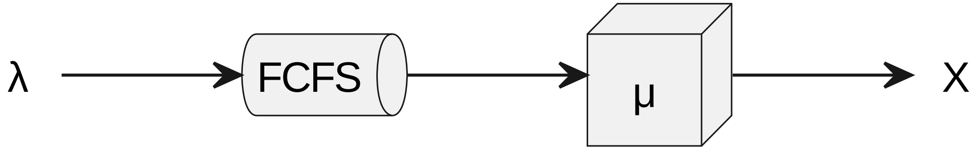
Given: a bank with **one** teller, customers taking **10 minutes** on average to serve and arriving at a rate of **5.8/hour**...

What is the expected wait time? How about with **two** tellers?

One teller	Two tellers
5 minutes	3 minutes
50 minutes	30 minutes
5 hours	3 hours

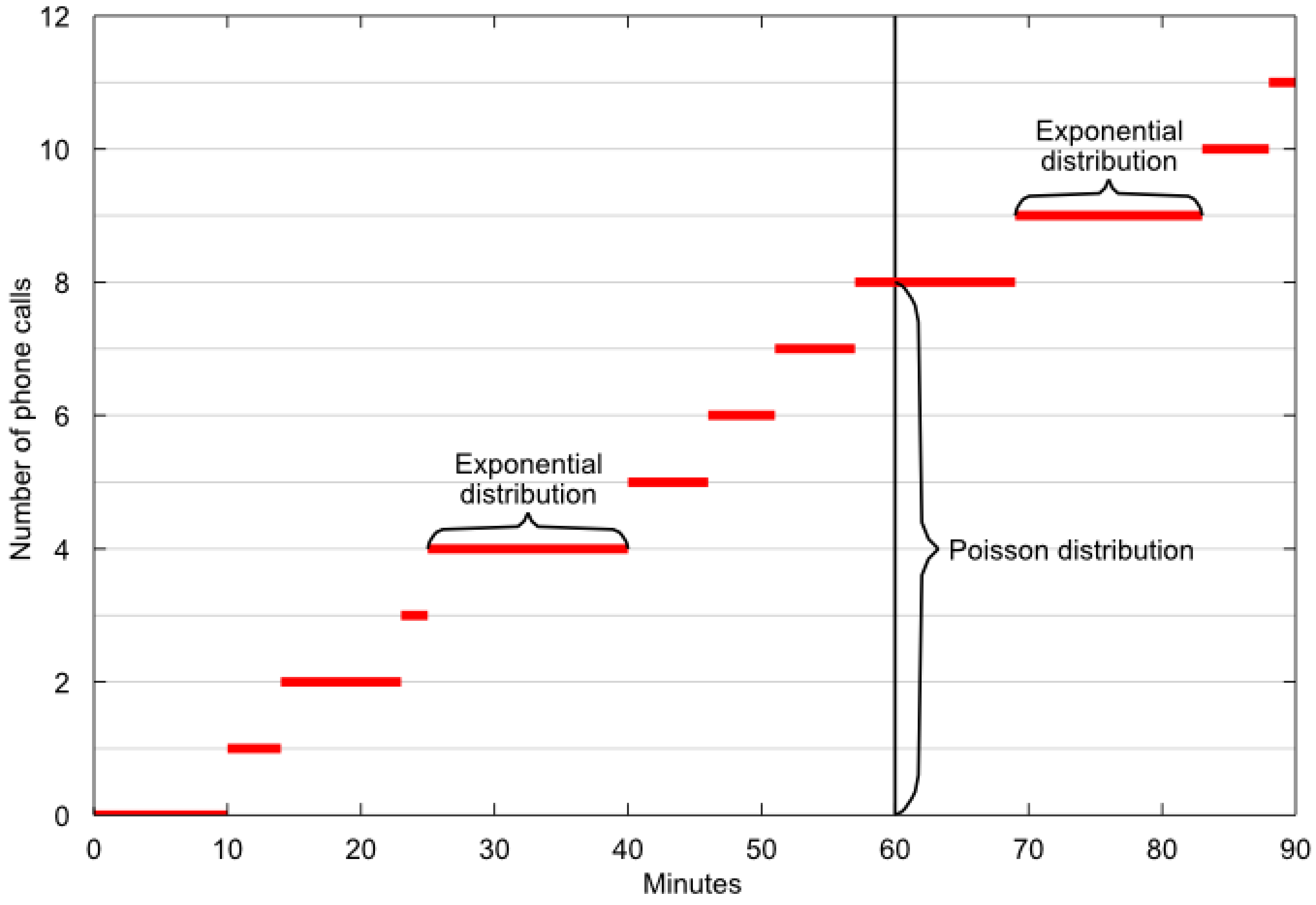
Ref: What happens when you add a new teller? (Cook 2008)

Terminology



- "M/M/1" or single-server queue
- service order, e.g. FCFS (default)
- λ: arrival rate
- μ: processing rate
- N: number of jobs in the queue
- X: throughput
- T: response time (time between arrival and departure, including waiting)
- 💡 An empty queue has the fastest response time

Distributions



💡 Arrivals (λ) and service time (μ) are distributions!

"M" in M/M/1 is for Markovian (Poisson and Exponential)

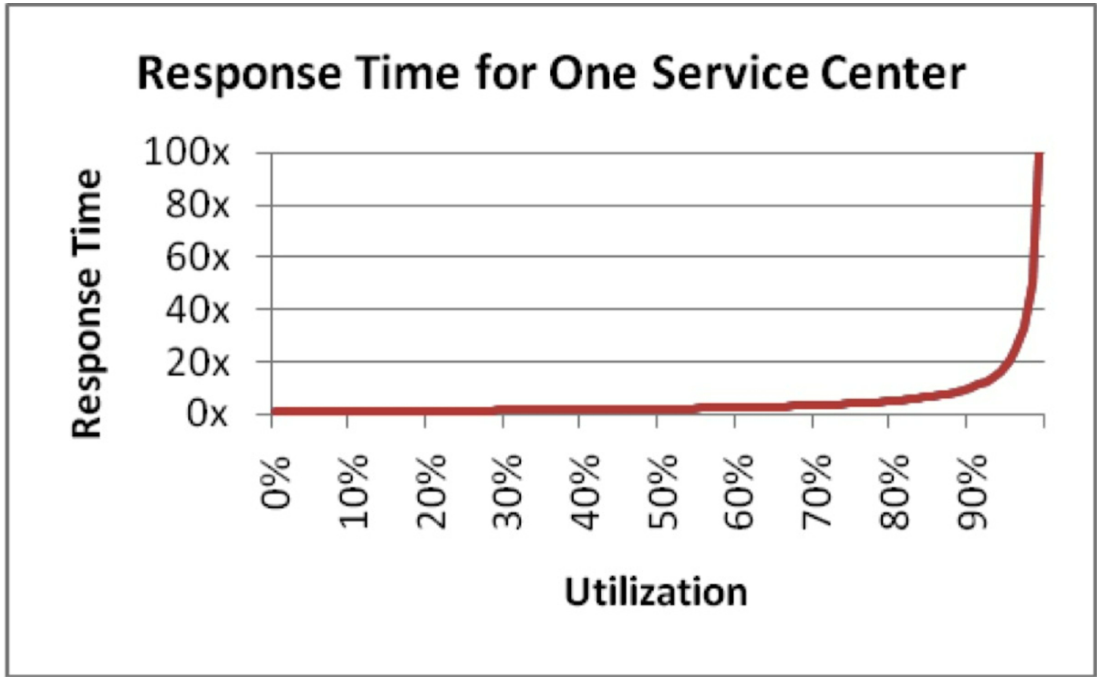
Bank tellers revisited

Given: customers taking **10 minutes** on average to serve and arriving at a rate of **5.8/hour**...

Skipping a whole bunch of math aka "rest of the owl"...

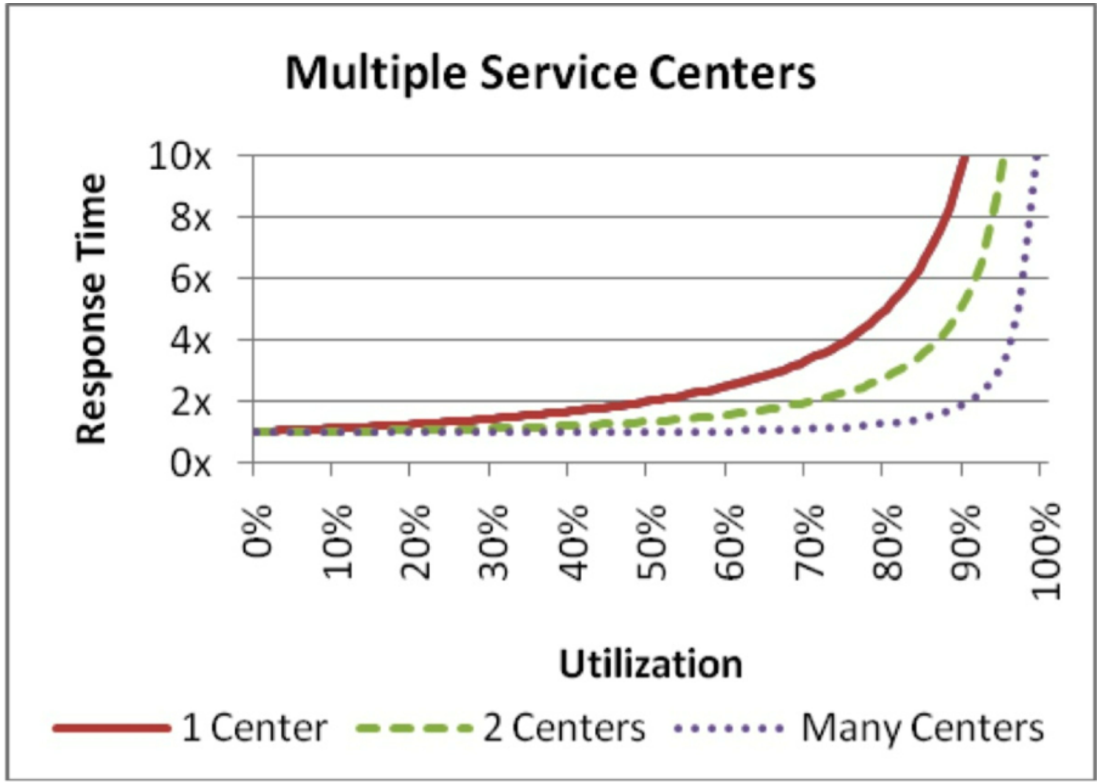
- $\lambda = 6.0, \mu = 5.8$
- 1 teller: $T_{Q,M/M/1} = \frac{\lambda}{\mu(\mu-\lambda)} = 4.83hrs$
- 2 tellers: $T_{Q,M/M/2} = \frac{\lambda^3}{\mu(4\mu^2-\lambda^2)} = 0.05hrs = 3.05min$

Response time vs. utilization



- $T = \frac{\mu}{1-\rho}$, ρ: utilization (0.0-1.0)
- 💡 "It's very hard to use the last 15% of anything."
- 💡 "The slower the service center, the lower the maximum utilization you should plan for at peak load."
- Ref: The Every Computer Performance Book (Wescott 2013)

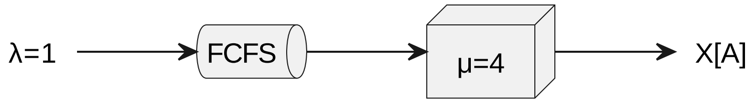
Response time with multiple service centers

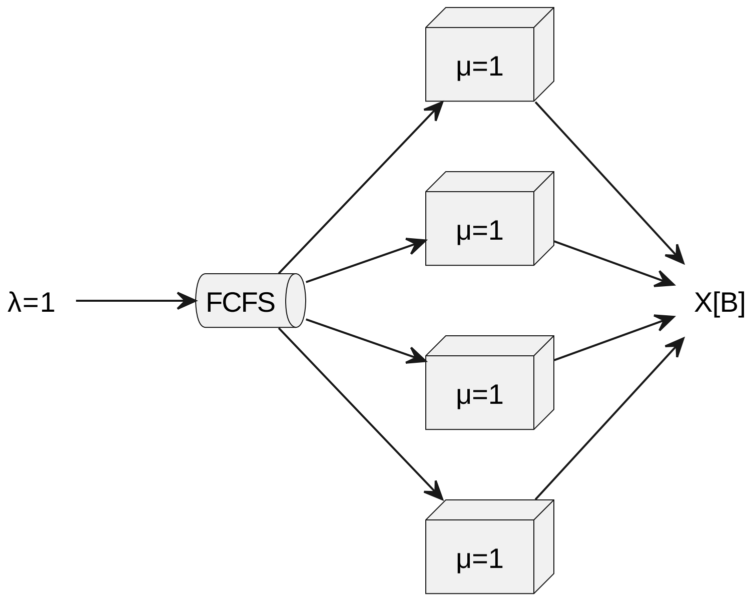


- 1 to 2 tellers: switch from red to dashed green **and** move left
- Adding parallelism increases throughput, but results in a steeper cliff
- 💡 "The closer you are to the edge, the higher the price for being wrong."
- Ref: The Every Computer Performance Book (Wescott 2013)

Two queueing systems compared

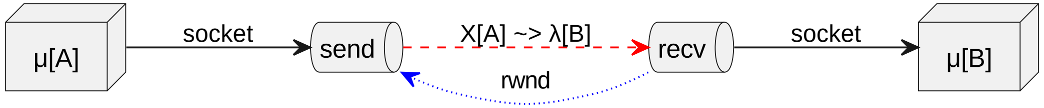
Which is better for response time? A single fast server or slower, parallel servers?

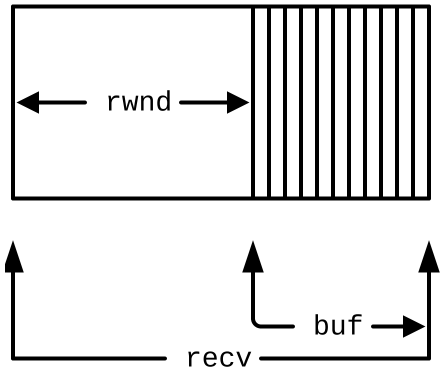




- Under low load?
- Under high job variability?
- 💡 Queuing results from *variability* in either service time or interarrival time!

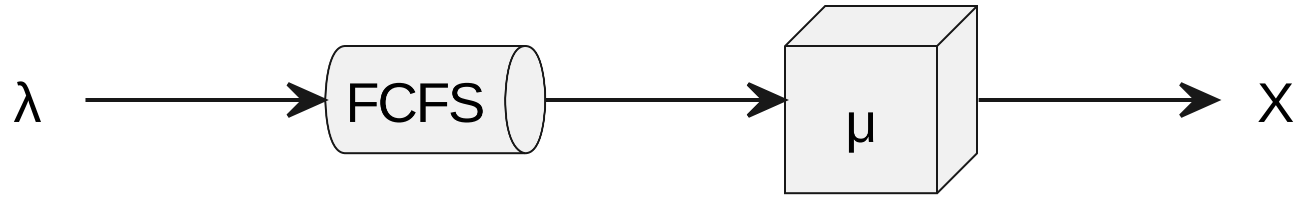
TCP: Transmission control protocol





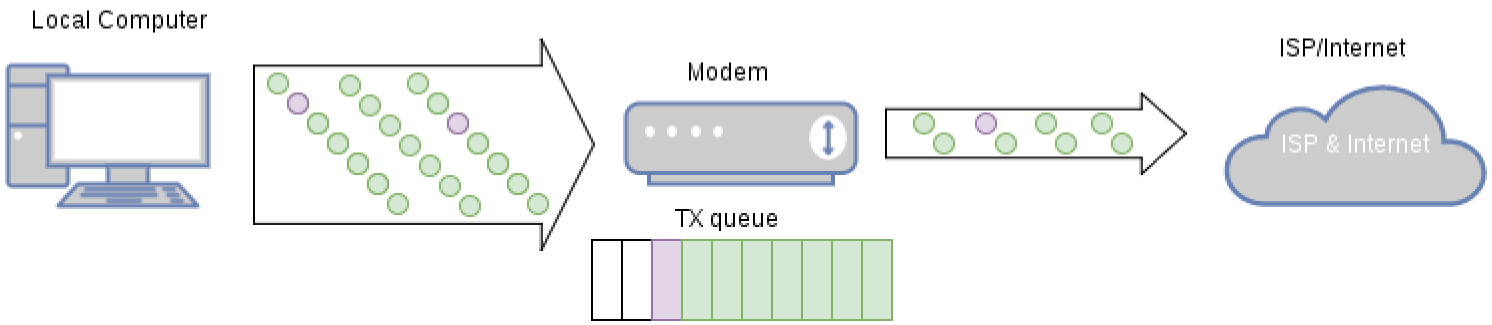
- *Caveat: gross simplification!*
- Two queues: send and receive buffers
- Packet loss: send > recv/arrivals
- 💡 **Backpressure**:
  - **B** advertises the receive window (rwnd)
  - **A** adjusts sending to not overwhelm **B**
- Backpressure is sometimes good and sometimes bad (more on this later!)
- Here it prevents **overload** (next!)
- Ref: TCP Flow Control (Wikipedia)

Overload



- Queues in practice have a size limit; going over it is "overload"
- When your system overloads, you have to:
  - Reject incoming work, e.g. load-shedding, drop packets
  - Interrupt ongoing work, e.g. task preemption, suspension
  - Change *something*
- FCFS: time in queue is pretty bad, for *everyone*
- 💡 Arrival rate (λ) < service rate (μ): *necessary* for stability, but not *sufficient*!

Bufferbloat



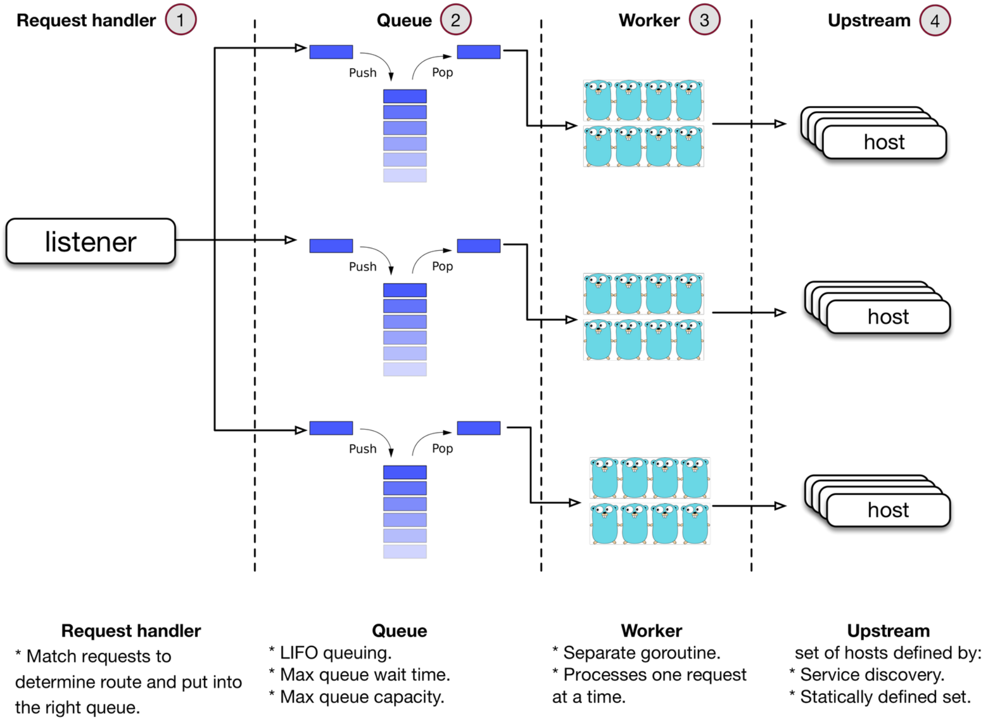
- Undesirable side effect of TCP congestion control
- "Bloated" buffers at a bunch of places in your system leads to lots of waiting, which result in latency and **lag**!
- Lots of super interesting algorithms on how to handle this, e.g. CoDel
- 💡 Queue size matters: size according to how much you (or your customers) are willing to wait

Ref: Controlling Queue Delay (Nichols 2012)

Ref: bufferbloat.net

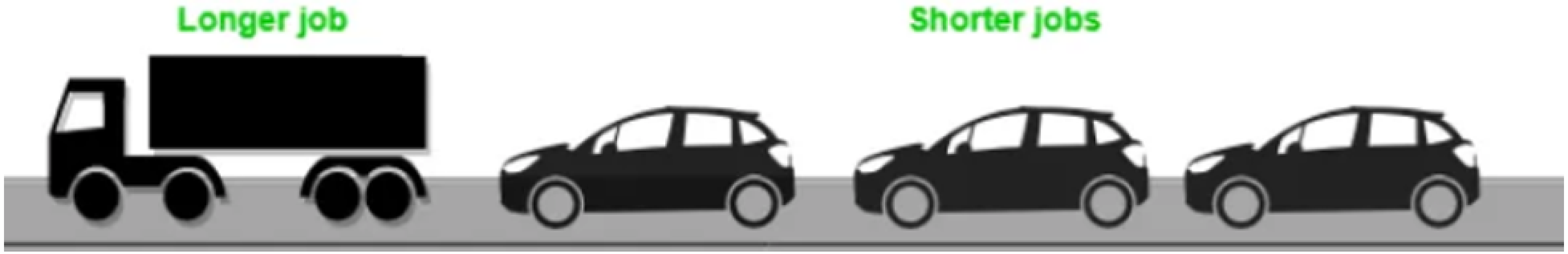
FCFS to LCFS

- What if we changed the default queue behavior?
- Trade-off *fairness* for *response time* (latency)
- Instead of all customers having a bad time, only some do (and they probably already retried or gave up)



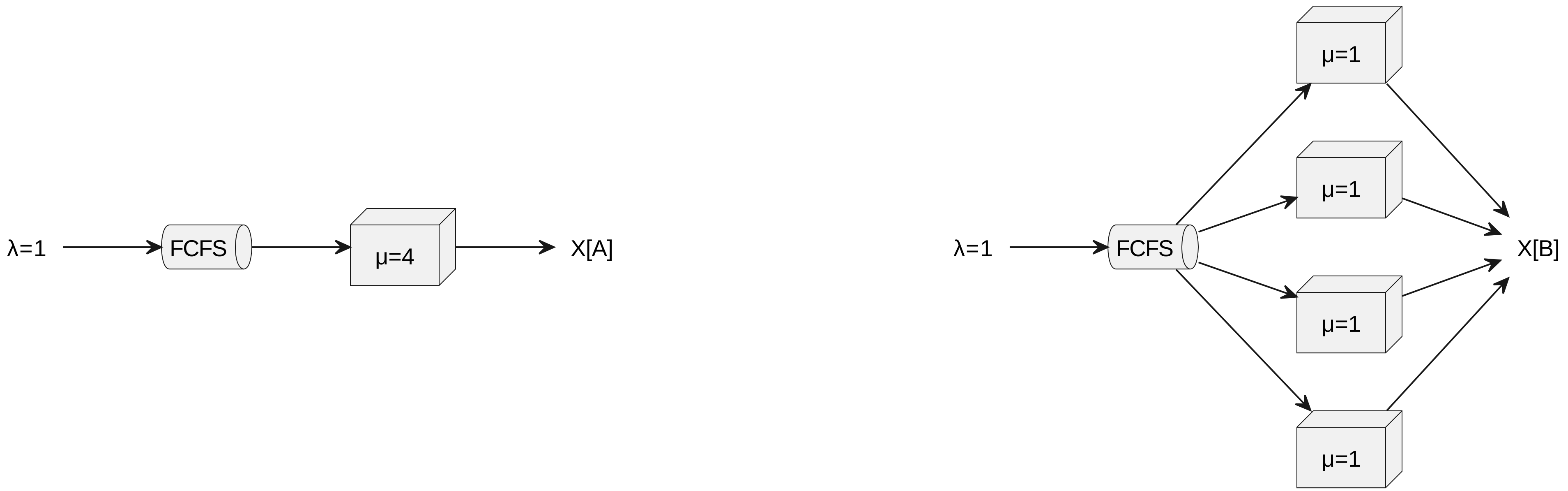
Ref: [Meet Bandaid, the Dropbox service proxy] (Dropbox 2018)

Convoy effect



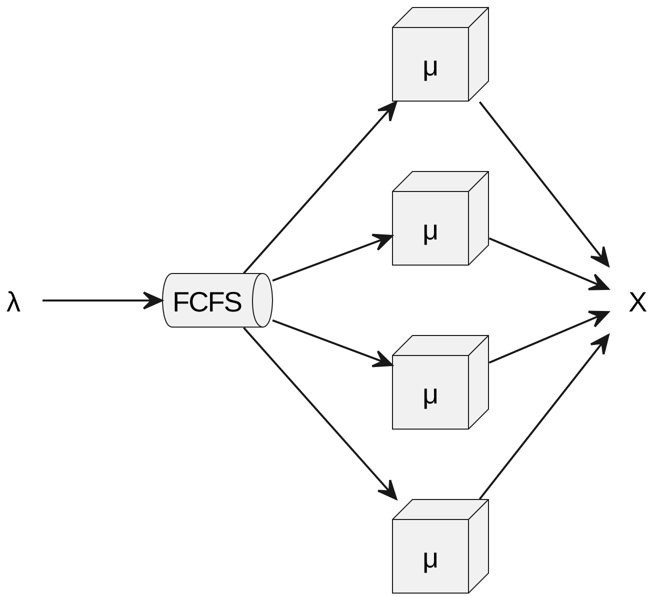
- Not to be confused with buffer bloat, but very similar
- Long job causes a backlog of much shorter jobs
- Convoy: things that can go faster being stuck behind things that are slower
- Very real problem in operating system architecture, e.g. older, cooperative scheduling systems would totally hang and crash if only one program did! (Pre-Windows 95)
- Generally fixed with multi-level (priority) queues and **task preemption**

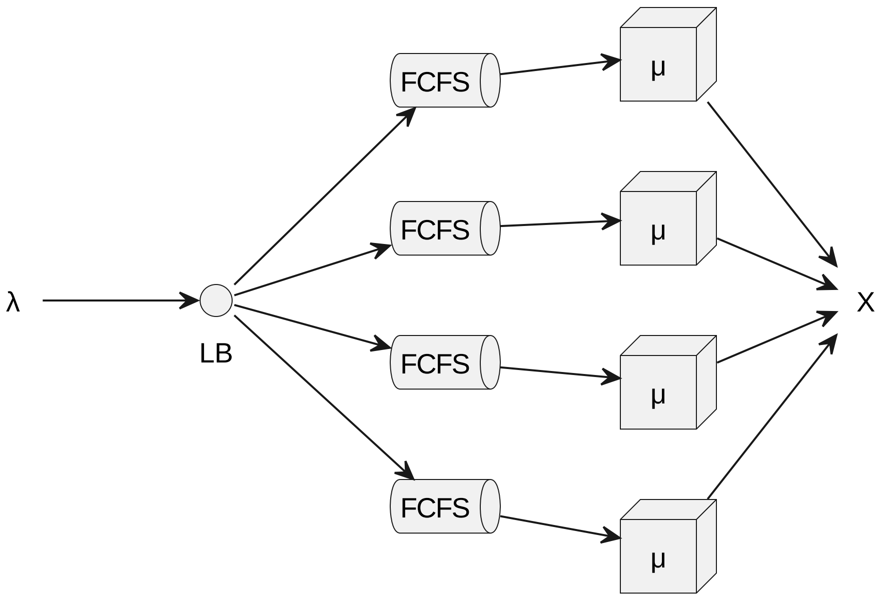
Revisiting the two systems with preemption



- With preemption, the M/M/1 can simulate and (mostly) match the M/M/4
- Suspending jobs, context-switching: how one CPU core can do many, *many* things
- Common to use separate priority queues, e.g. multilevel feedback queue (MLFQ)

Central vs. multiple queues





Offline, batch systems

Larger queues

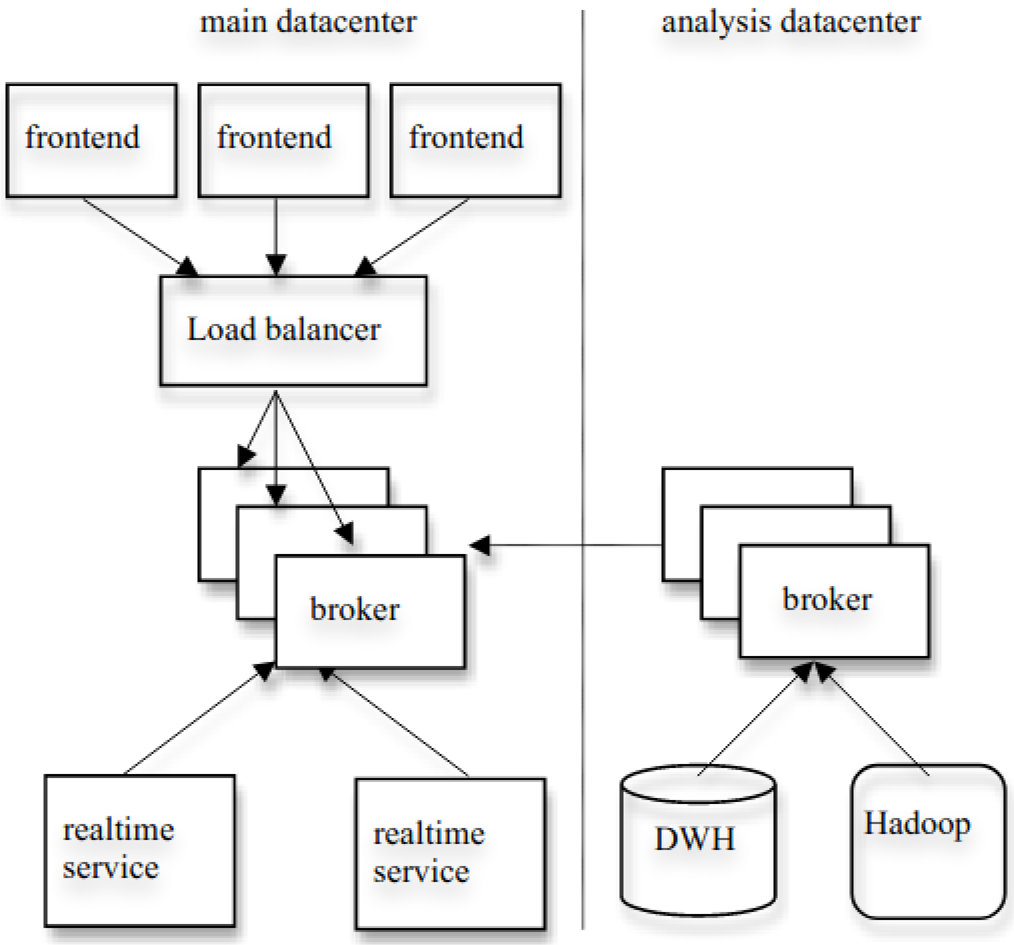
Optimized for throughput

Online, live systems

Critical to load-balancing

Optimized for latency

Kafka



- Messaging queues used to decouple producers from consumers
- Queues managed by replicated brokers
- Distributed system with "levers" and tradeoffs:
    - Throughput
    - Latency
    - Durability
    - Availability

Ref: Kafka: a Distributed Messaging System for Log Processing (Kreps 2011)

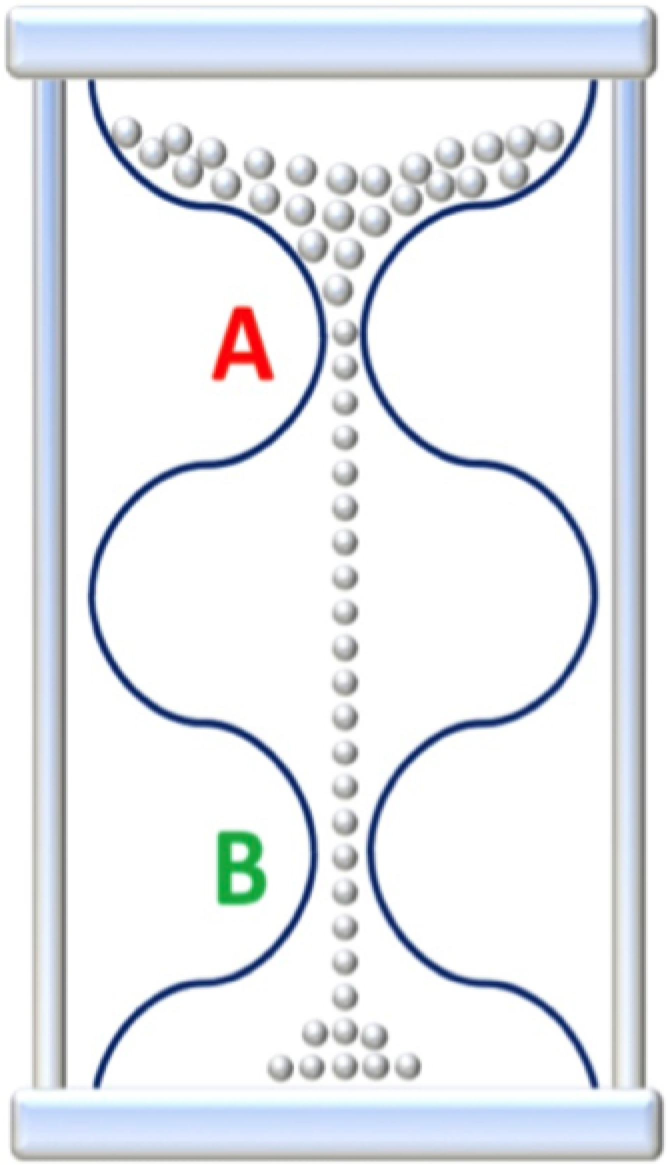
Performance

Queueing is central to systems performance:

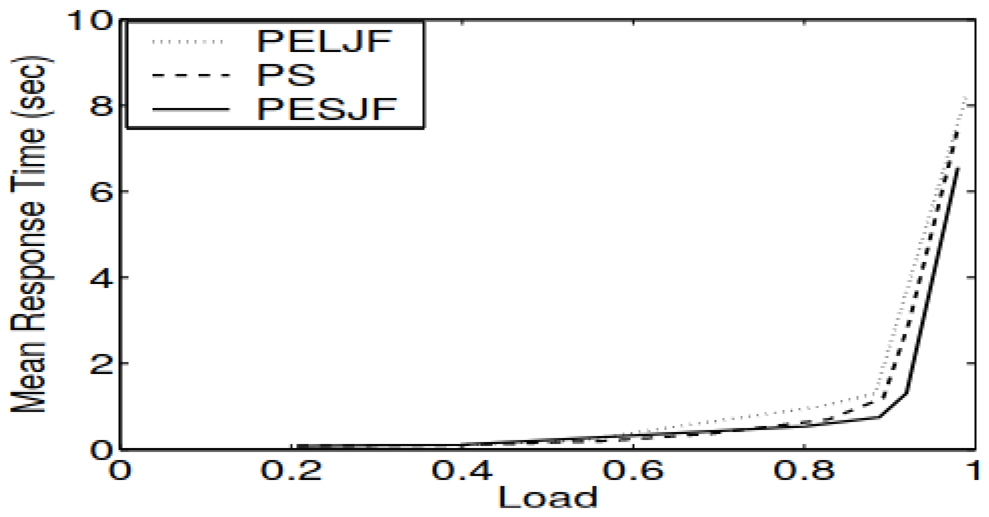
- Latency (response time)
- Throughput

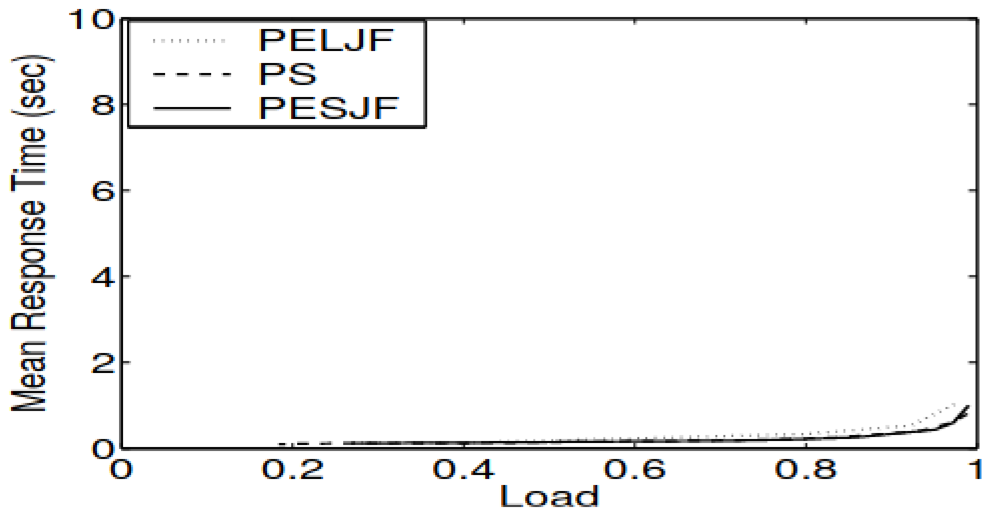
Hidden bottlenecks

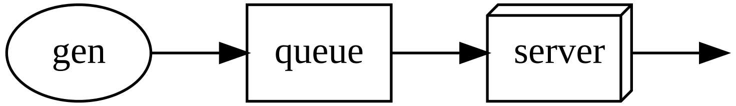
- Once you fix a bottleneck (**A**), a new, *hidden* one (**B**) might appear!
- Testing component **B** in isolation may have surfaced this earlier
- Typically more common in *batch* systems where you're concerned with throughput
- aka *whack-a-mole*

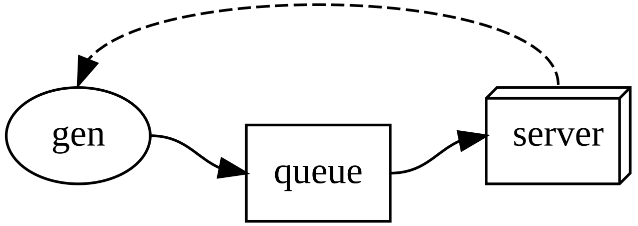


Open versus closed









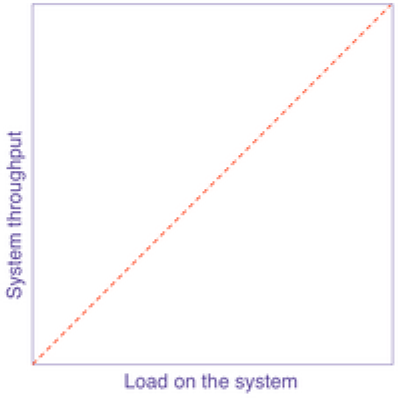
💡 If your load test is a closed-loop load generator, it might be lying to you!

Ref: Open Versus Closed: A Cautionary Tale (Schroeder 2006)

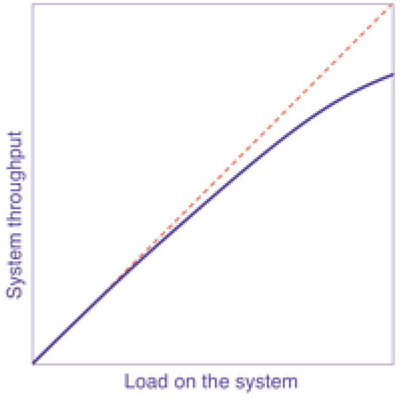
Universal scaling law (USL)

$$X(N) = \frac{\mu N}{1-\alpha(N-1)+\beta N(N-1)}$$

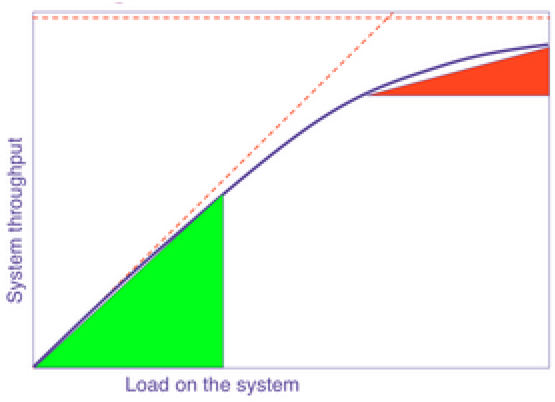
- Concurrency: μ is the slope in ideal parallelism
- Contention: α comes from queueing on shared resources
- Coherency: β comes from cross-talk and consensus
- 🤔 Maybe this also applies to humans and organizations?



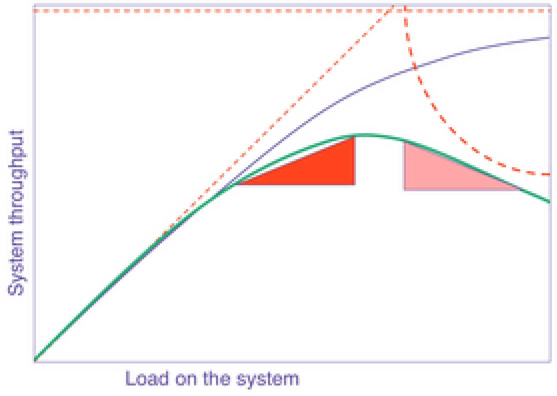
$\alpha = 0, \beta = 0$



$\alpha > 0, \beta = 0$



$\alpha >> 0, \beta = 0$

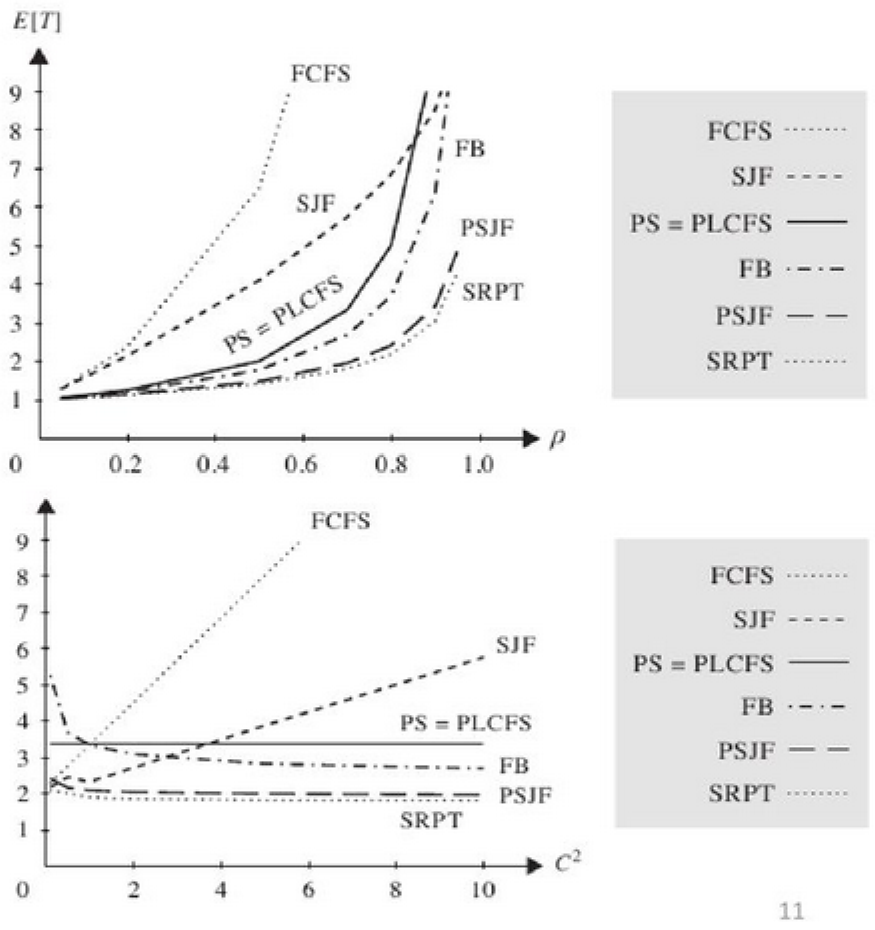


$\alpha >> 0, \beta > 0$

Ref: Guerilla Capacity Planning (Gunther 2007)

Scheduling

Algorithm	Description
(RAND)	Randomly choose
(RR)	Round robin
FCFS	First come, first serve
SJF	Shortest job first
PLCFS	Preemptive LCFS
FB	Foreground-background
PSJF	Preemptive SJF
SRPT	Shortest remaining processing time



Ref: Task Assignment with Unknown Duration (Harchol-Balter 2002)

Key takeaways

- Queueing is critical to performance, both response time and throughput
- Full (and overflowing) queues are bad for response time
- Lots of unintuitive trade-offs in architecture and algorithm choice

There's *so* much more. Where to learn more about queueing?


FACTORIO

Reading

- Controlling Queue Delay (Nichols 2012)
- The Every Computer Performance Book (Wescott 2013)
- Guerilla Capacity Planning (Gunther 2007)
- Kafka: a Distributed Messaging System for Log Processing (Kreps 2011)
- Meet Bandaid, the Dropbox service proxy (Dropbox 2018)
- Open Versus Closed: A Cautionary Tale (Schroeder 2006)
- Performance Modeling and Design of Computer Systems (Harchol-Balter 2013)
- Task Assignment with Unknown Duration (Harchol-Balter 2002)

Thanks! 👋