What’s the cost of a millisecond?

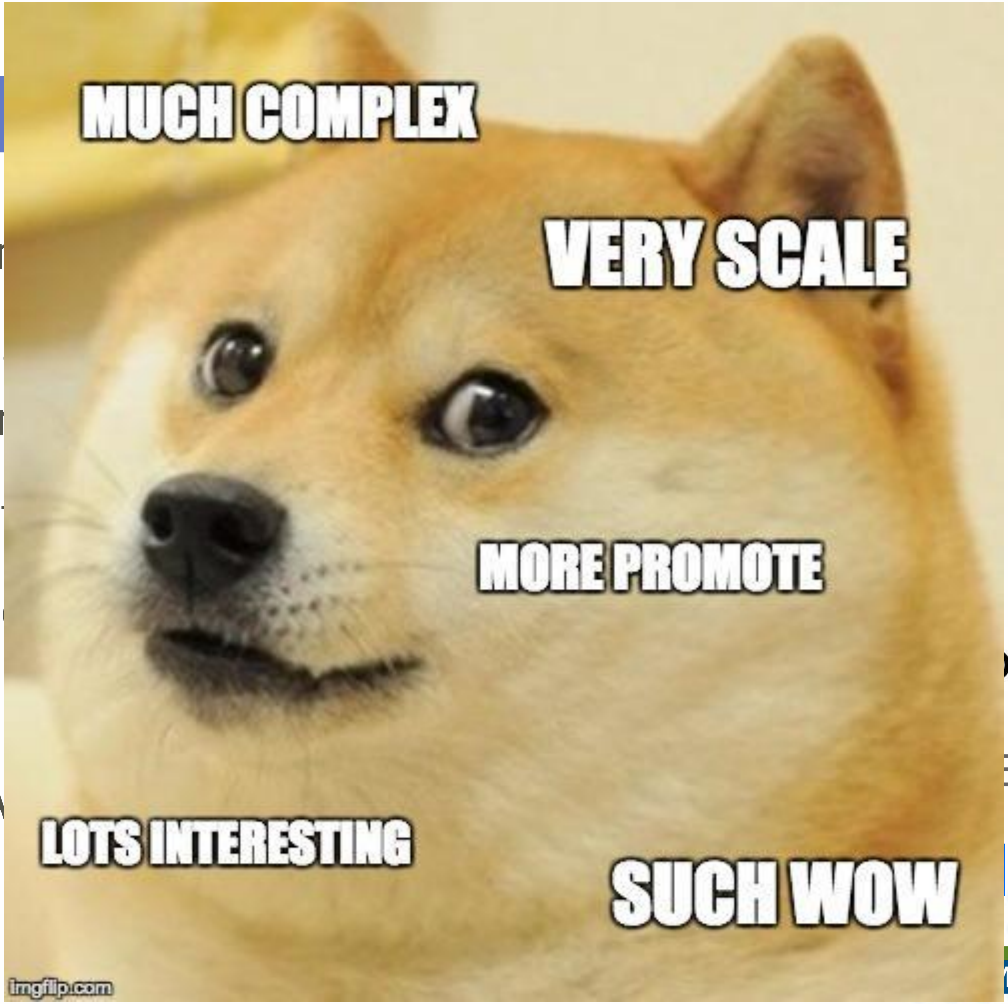
Avishai Ish-Shalom (@nukemberg)

ScyllaDB what?

+ The Real-Time Big Data Database
+ Drop-in replacement for Apache Cassandra and Amazon DynamoDB
+ 10X the performance & low tail latency
+ Open Source, Enterprise and Cloud options
+ Founded by the creators of KVM hypervisor
+ HQs: Palo Alto, CA, USA; Herzelia, Israel; Warsaw, Poland

Scylla

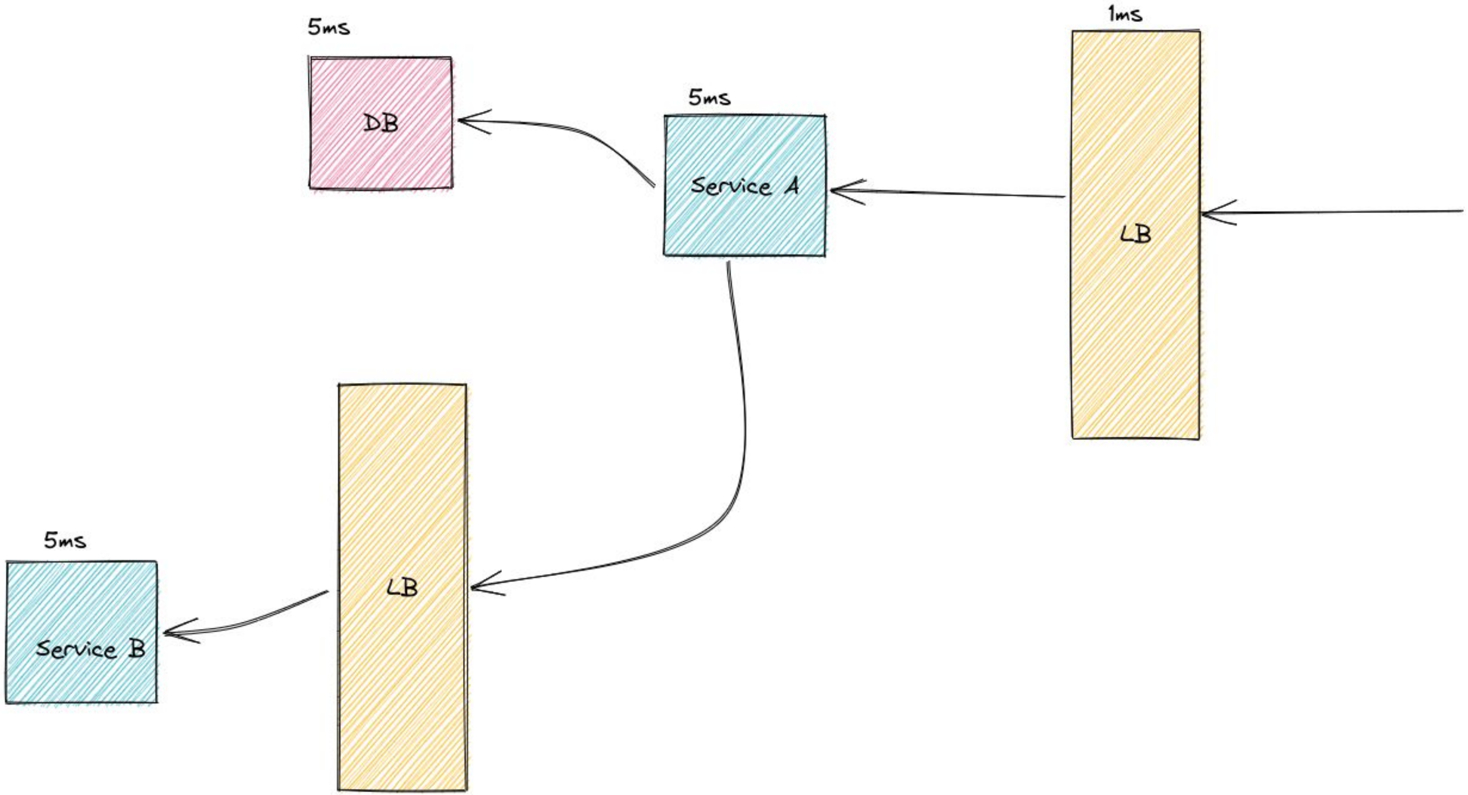
+ The Real-Ti
+ Drop-in repl
  and Amazo
+ 10X the per
+ Open Sourc
+ Founded by
+ HQs: Palo A
  Warsaw, Po



Santander
Grab
Fanatics
STARBUCKS COFFEE
SAMSUNG
SAMSUNG SDS
adgear
AppNexus
Zenly

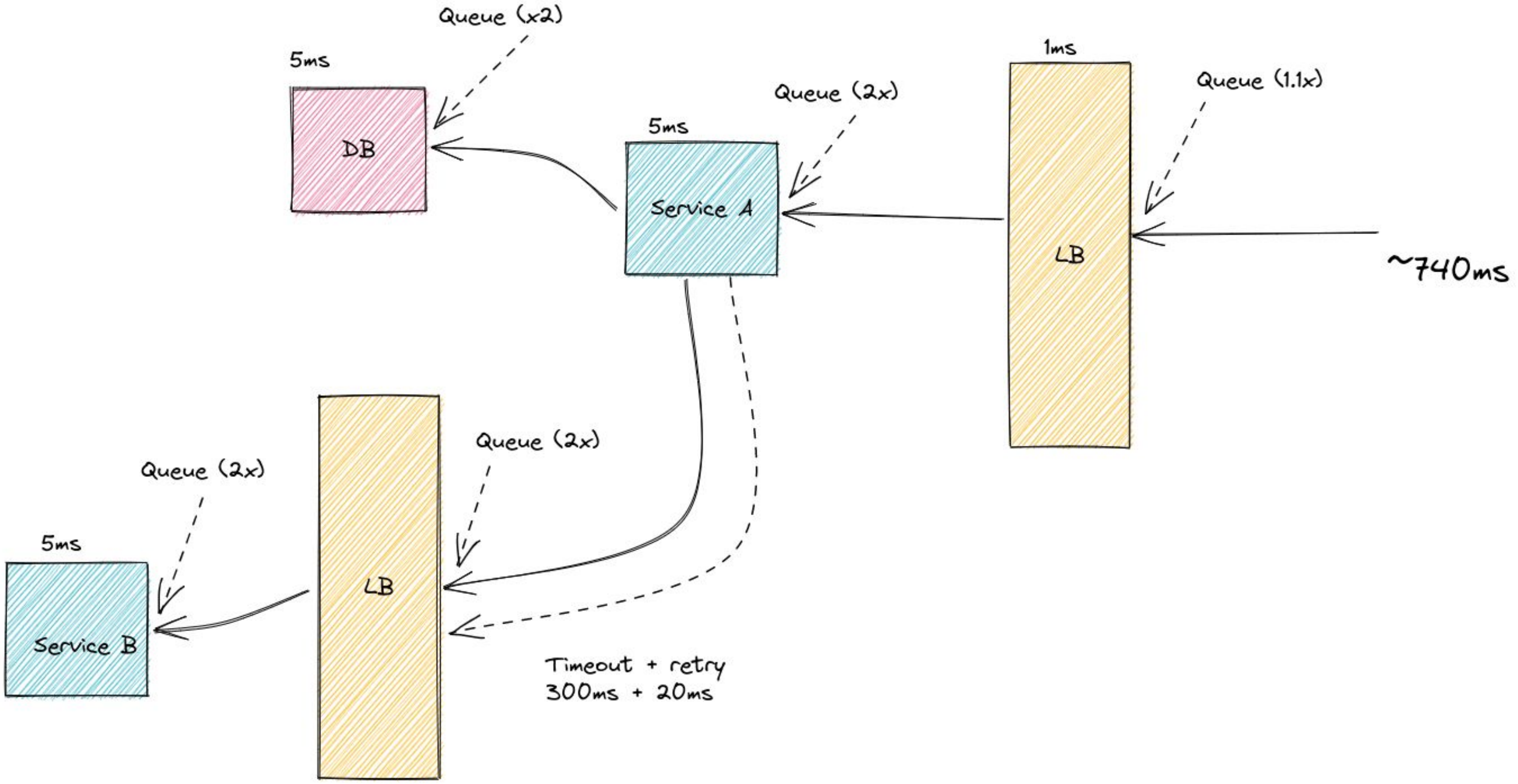
The curse of latency amplification

5ms + 5ms + 5ms + 1ms = 16ms, right?



SCYLLA.

Once more, with amplification



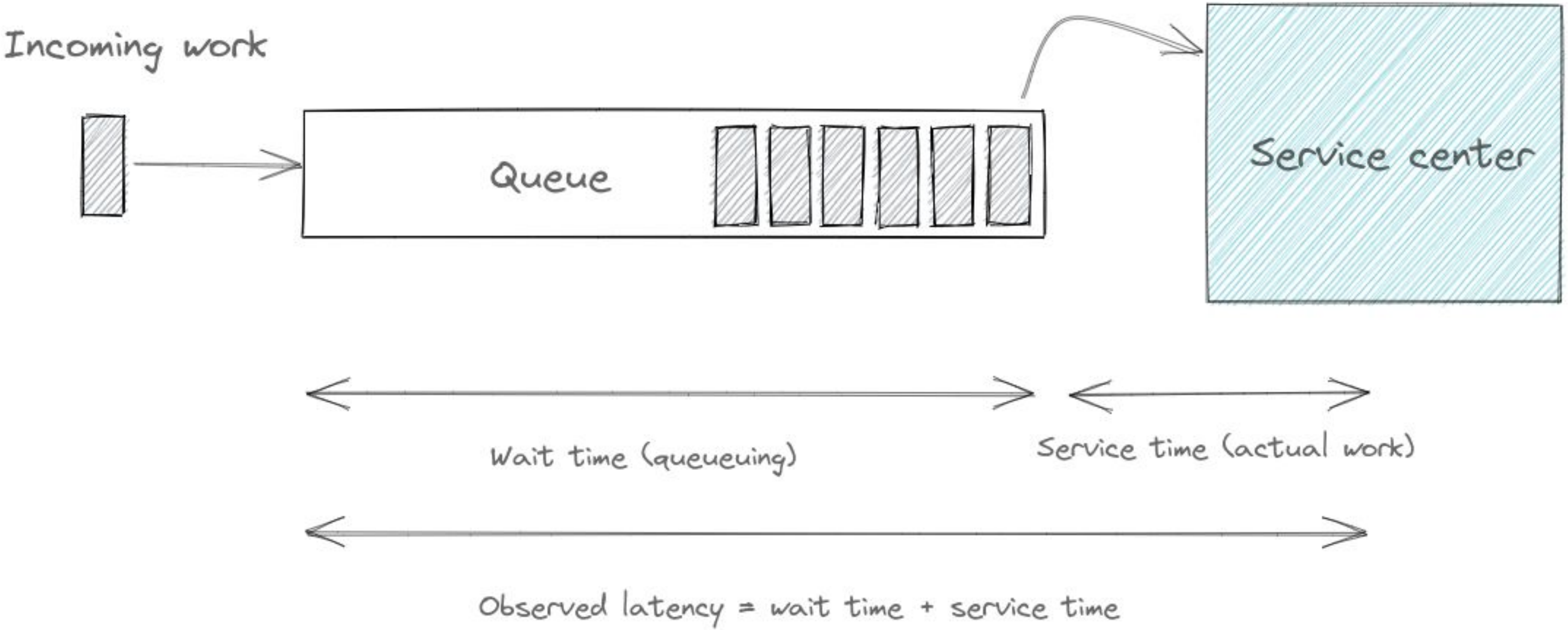
Oh sh!t

+ Every queue amplifies latency
+ A timeout has a large penalty
+ Retry has a penalty

And it all compounds. And transactions impact each other

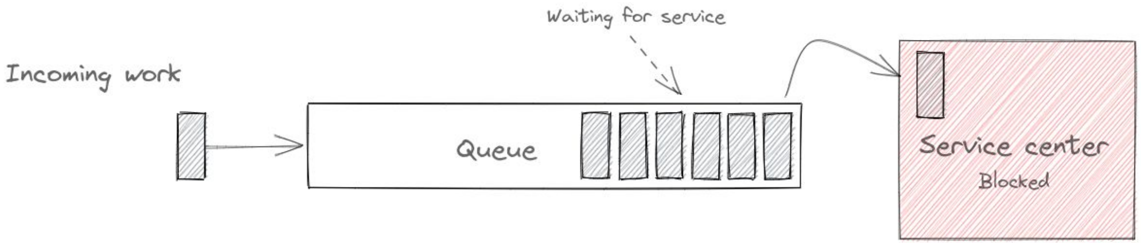
Queueing theory crash course

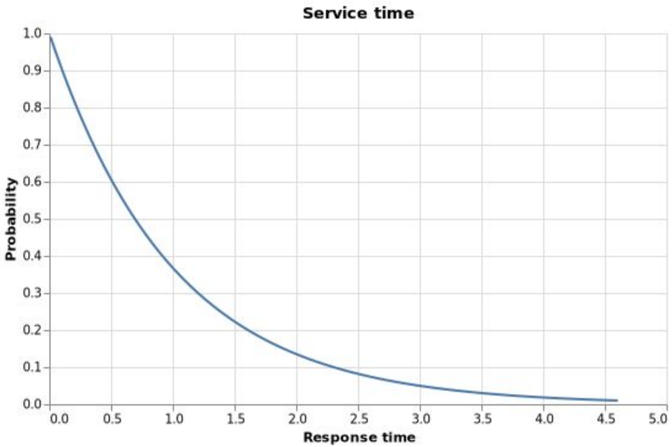
SCYLLA.

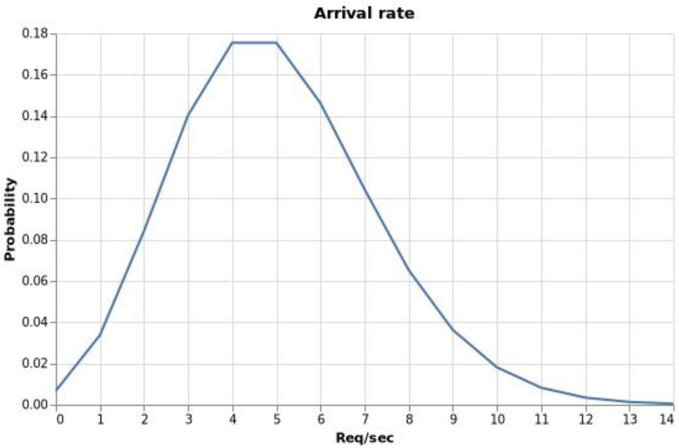

Incoming work
Queue
Service center
Wait time (queueing)
Service time (actual work)
Observed latency = wait time + service time

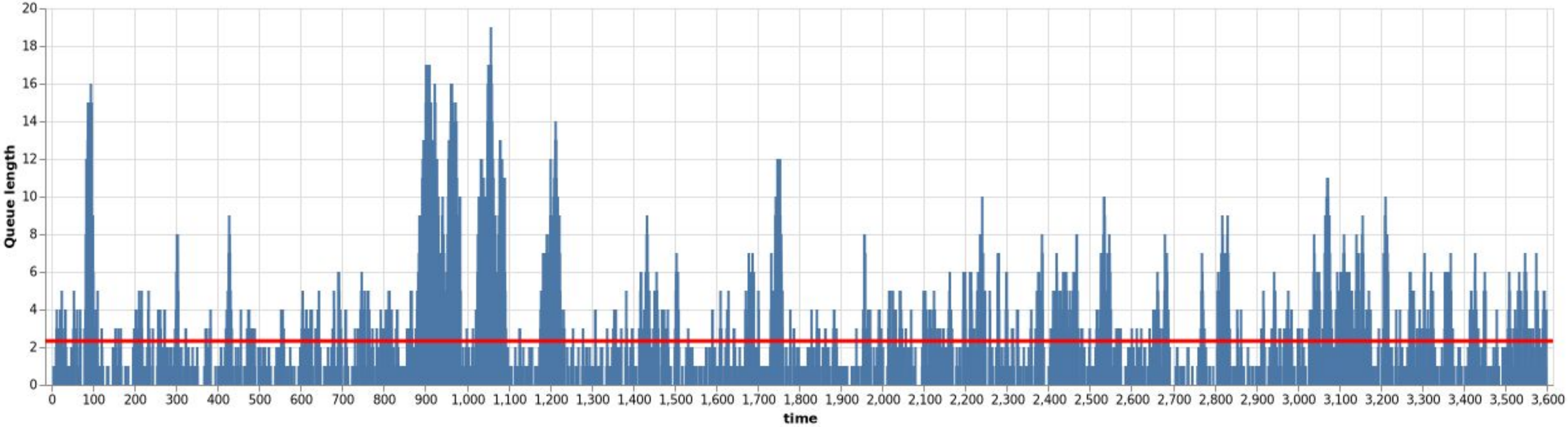
Head of line blocking

- When some task takes longer, service center is “blocked”
- Other tasks in the queue are blocked by the “head of line”
- A single slow task will cause a *bunch of other tasks* to wait
  - Bad news for latency high percentiles




Service time
Probability
Response time
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Arrival rate
Probability
Req/sec
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

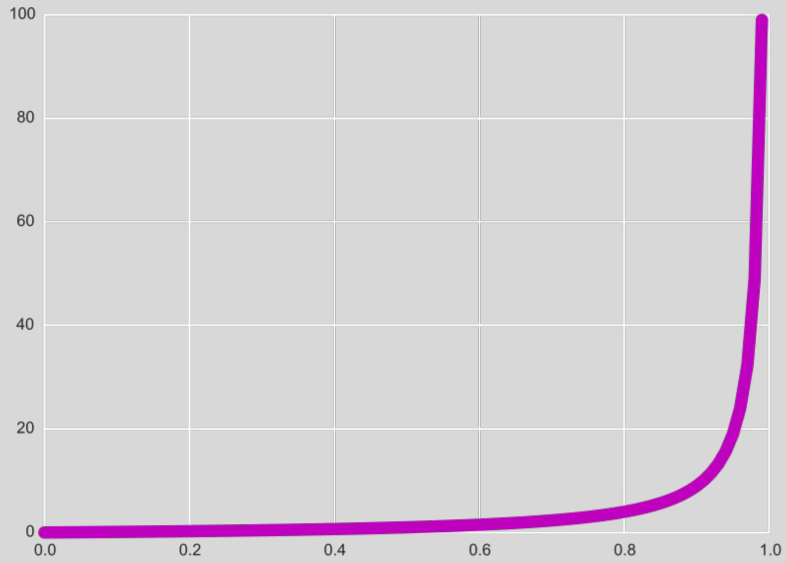

Queue length
time
20
18
16
14
12
10
8
6
4
2
0
100
200
300
400
500
600
700
800
900
1,000
1,100
1,200
1,300
1,400
1,500
1,600
1,700
1,800
1,900
2,000
2,100
2,200
2,300
2,400
2,500
2,600
2,700
2,800
2,900
3,000
3,100
3,200
3,300
3,400
3,500
3,600

Capacity & Latency

+ Latency (and queue size) rises to infinity as utilization approaches 1
+ Decent latency -> over capacity

http://queuemulator.gh.scylladb.com/

SCYLLA



$$Q \propto \frac{\rho}{1-\rho}$$

ρ = arrival rate / service rate = utilization

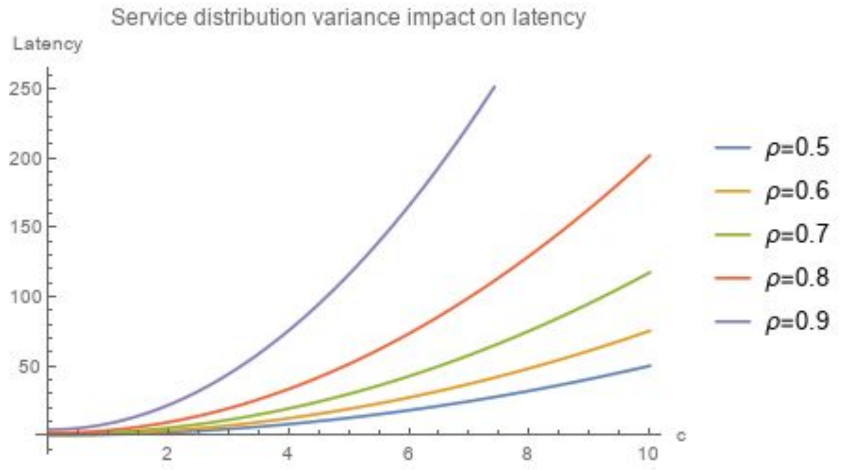
Q = Queue length

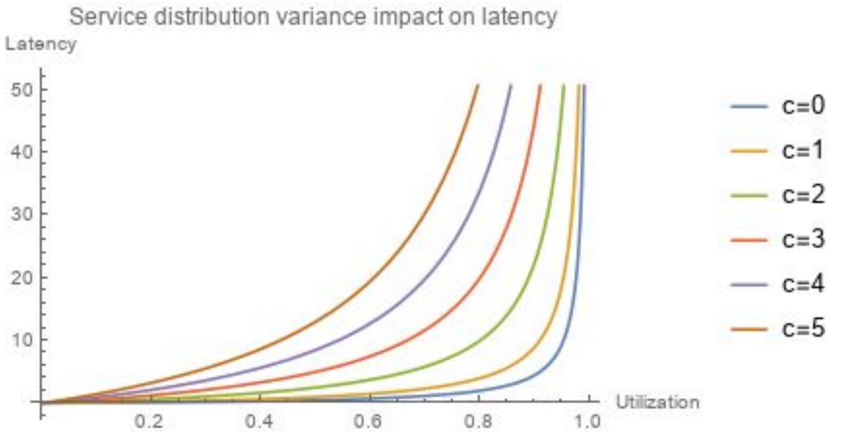
Kingman formula

$$\mathbb{E}[W_q] \approx \left(\frac{\rho}{1-\rho}\right)\left(\frac{c_a^2 + c_s^2}{2}\right)\tau$$

- The higher the variance, the worse the latency/utilization curve gets
- On both service rate and arrival rate
- high variance ⇒ run at low utilization

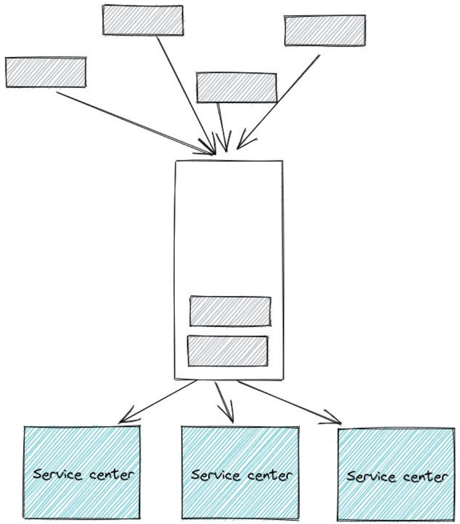
Oh and btw your percentile curve is worse too*





Tasks should be independent, but...

- Shared resources have queues
  - Disks, CPUs, Thread pools, Load balancers, connection pools, DB locks, sockets…
- Head-of-line blocking → cross task interaction
  - Slow tasks raise latency of unrelated tasks
  - Arrival spikes
- High variance service makes this worse
- Parallel queues are less susceptible, but are less efficient
  - Some queues will be starved → lower utilization, throughput



Executive summary

+ High utilization → high latency
  + Non-linear!
+ High variance → high latency
+ Shared queues*→ higher throughput, lower latency
+ **Never** use unlimited queues

* For identical service centers

Amplification sources

Queueing

- Queues are everywhere
  - LB, locks, resource pools, sockets, event loop… and ofc, queues
- Non linear rise in latency when load rises
  - Very problematic when running near capacity limits
- Often not monitored

Timeouts

Break when something takes too long (or won't complete)

+ Timeout values often arbitrary
+ Often wayyyy too long
  + Example: HikariCP `acquire()` min timeout = 250ms (!!!)
+ Often blocking service centers/other resources

Power of ten syndrome: 100 is a bogus number

Retry

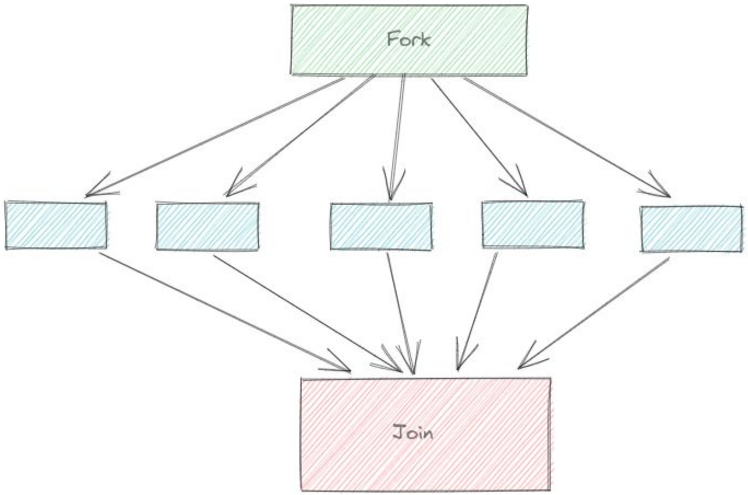
If at first you don't succeed, try again!

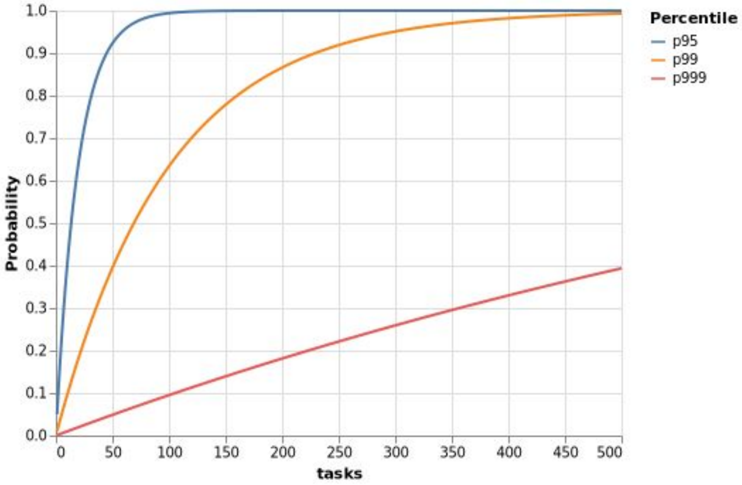
+ But this takes even more time
+ Especially if you have long timeouts
+ How many retries?
    + Do they all have the same timeout?

Fork/Join

Spawn multiple parallel tasks, wait for all

+ Blocked until last task is complete
+ High probability of hitting at least 1 high percentile





Deep stack (aka The curse of microservices)

- Every cross service call has amplification
- And they compound: $A_1*A_2*A_3$...
- Need to wait for all - in sequence; Like fork/join only worse
  - Growing probability of at least one p95/failure/timeout/overload...

Combating latency amplification

SCYLLA

Proper timeouts

For the love of god, measure!

+ Use latency percentiles/histogram to determine correct timeouts
  + failure rate ↔ max latency
+ Compare with failure cost (e.g. reconnect)
+ Timeouts don’t have to be static!
  + E.g. `timeout = Min(P999[last 5m], 300ms)`
  + Lower timeout on high load

Timeout budget

+ Global latency budget for request
  + Pass on request context
+ Decrement actual processing on every stage
+ Timeout = min(remaining budget, local timeout)
+ Preemptive abort: fail if not enough budget

	Budget	Work
Service 1	500ms	123ms
Service 2	377ms	72ms
Service 3	305ms	287ms
Service 4	18ms	reject

Very useful with microservices, but needs protocol support

Parallel dispatch

+ Double dispatch: ask twice, wait for first answer
  + But also costs twice
+ Speculative execution: get data before you need it
+ Branch prediction: get data you *might* need
+ Harvest/yield: ask multiple shards, replicas; proceed with the answers you got within the timeout

Smarter retries

- Speculative retries: retry even without failure, wait for first answer
  - Cheaper than double dispatch, very effective
  - Your API is idempotent, right?
- Second retry can have shorter timeout (use the budget, Luke!)
- Probabilistic retries: why retry if you can’t succeed

Capacity/latency management

Overloaded service centers will have higher latency amplification

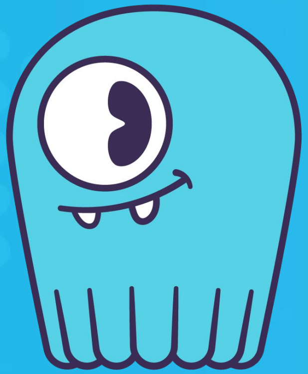
+ Limit concurrency according to Little's law
+ Cap queue lengths
+ Run slower service with lower utilization
+ Run high variance services with lower utilization
+ Backpressure, backpressure, backpressure
+ Implement load shedding
+ Circuit breakers

Reducing variance

- Separate services with different latency characteristics
- Complex semantics →high performance variance; Use caution
- You can’t do better than your backend. DB choices matter
- No preemption (Node/Golang), no QoS
  - Cooperative yield
  - Break into small tasks
- Be minded of GC, scheduled tasks, data structure shuffles, background tasks, etc.

Summary: what's the cost of a millisecond?

- Much higher than you think, especially at the bottom of the stack
- High percentiles have disproportionate impact
  - Forget about averages
- Latency amplification is the most common reason for low utilization
- Need to actively combat amplification



SCYLLA

Thank you

United States
1900 Embarcadero Road
Palo Alto, CA 94303

Israel
4 Maskit, building C
Herzelia, Israel

www.scylladb.com
@scylladb