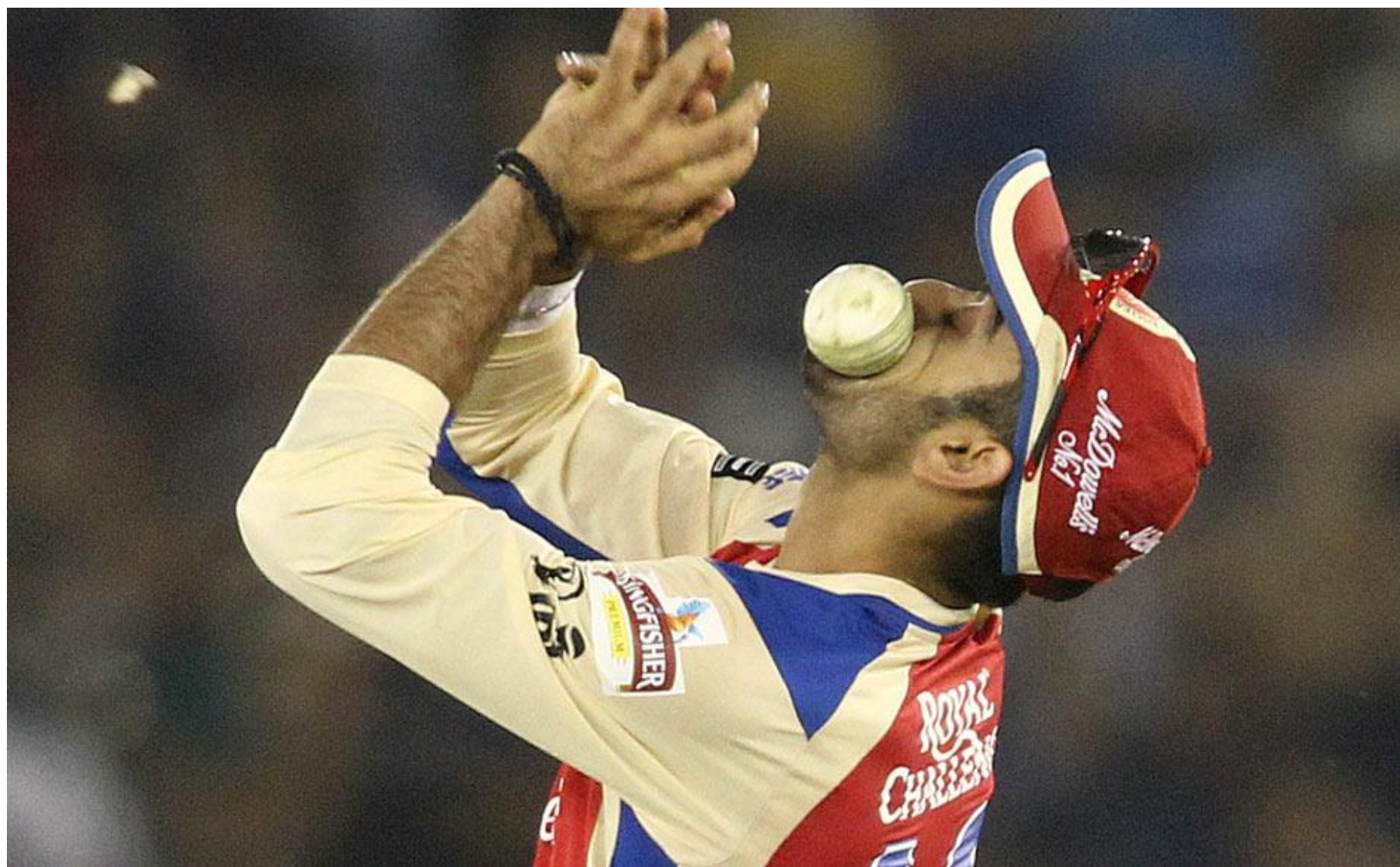




Responding in a timely manner

Martin Thompson - @mjpt777



Hard Real-time



Soft Real-time



Squidgy Real-time



The Unaware



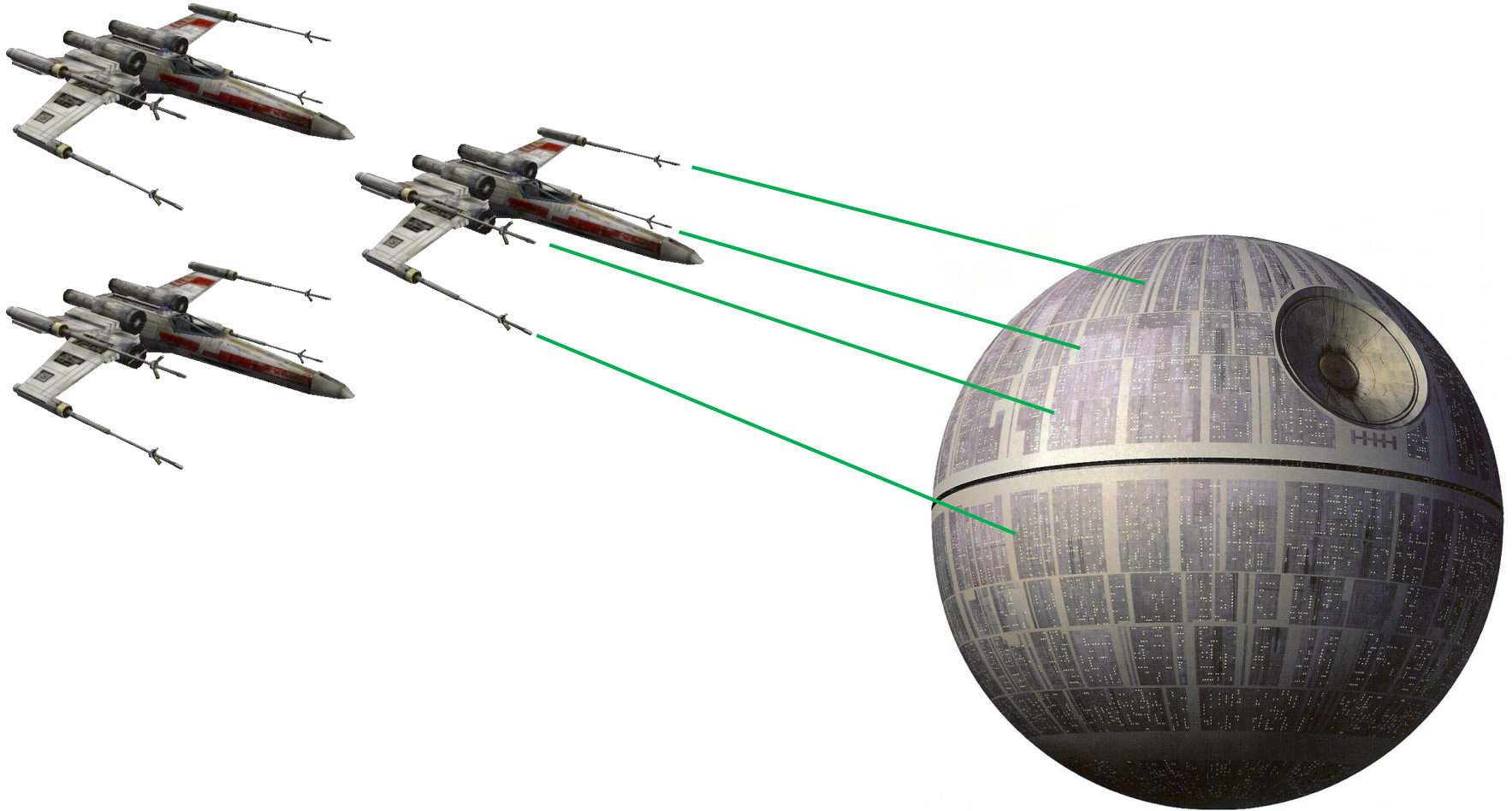
1. How to **Test** and **Measure**
2. A little bit of **Theory**
3. A little bit of **Practice**
4. Common **Pitfalls**
5. Useful **Algorithms** and **Techniques**

Test & Measure



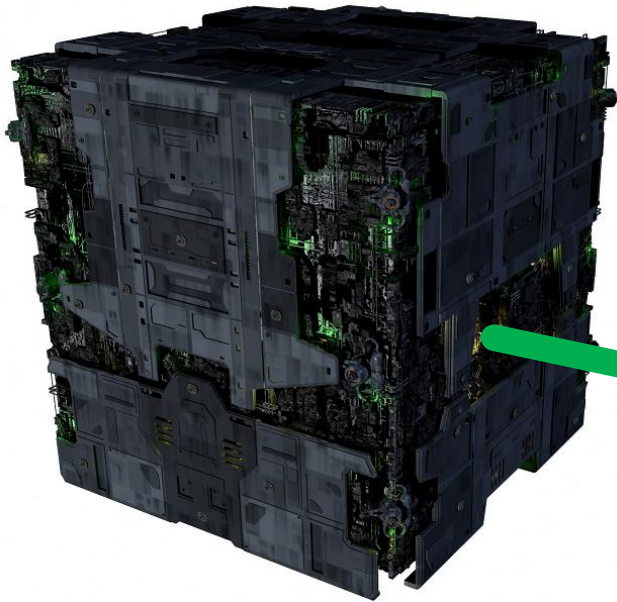
System Under Test

Distributed Load Generation Agents



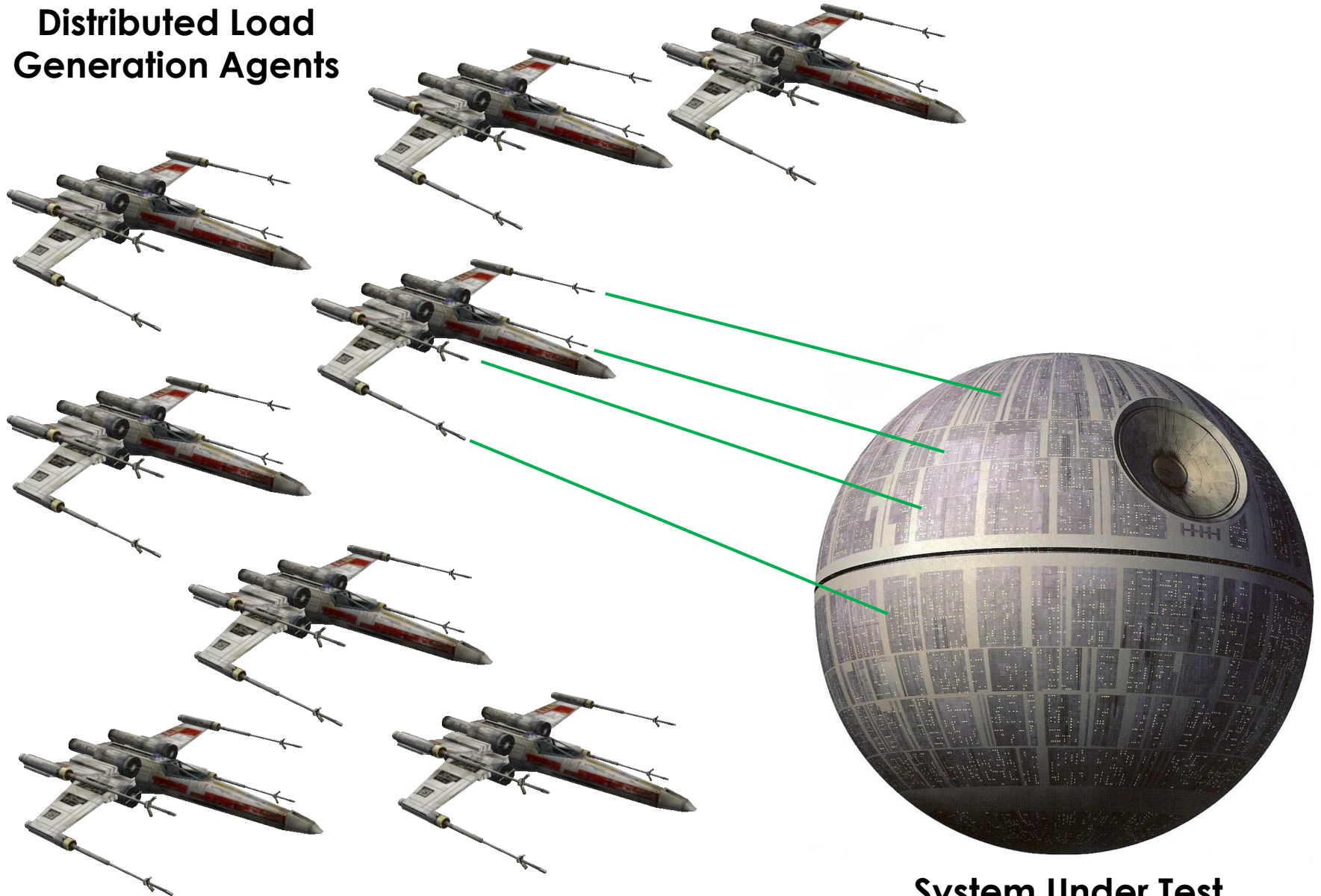
System Under Test

Distributed Load Generation Agents



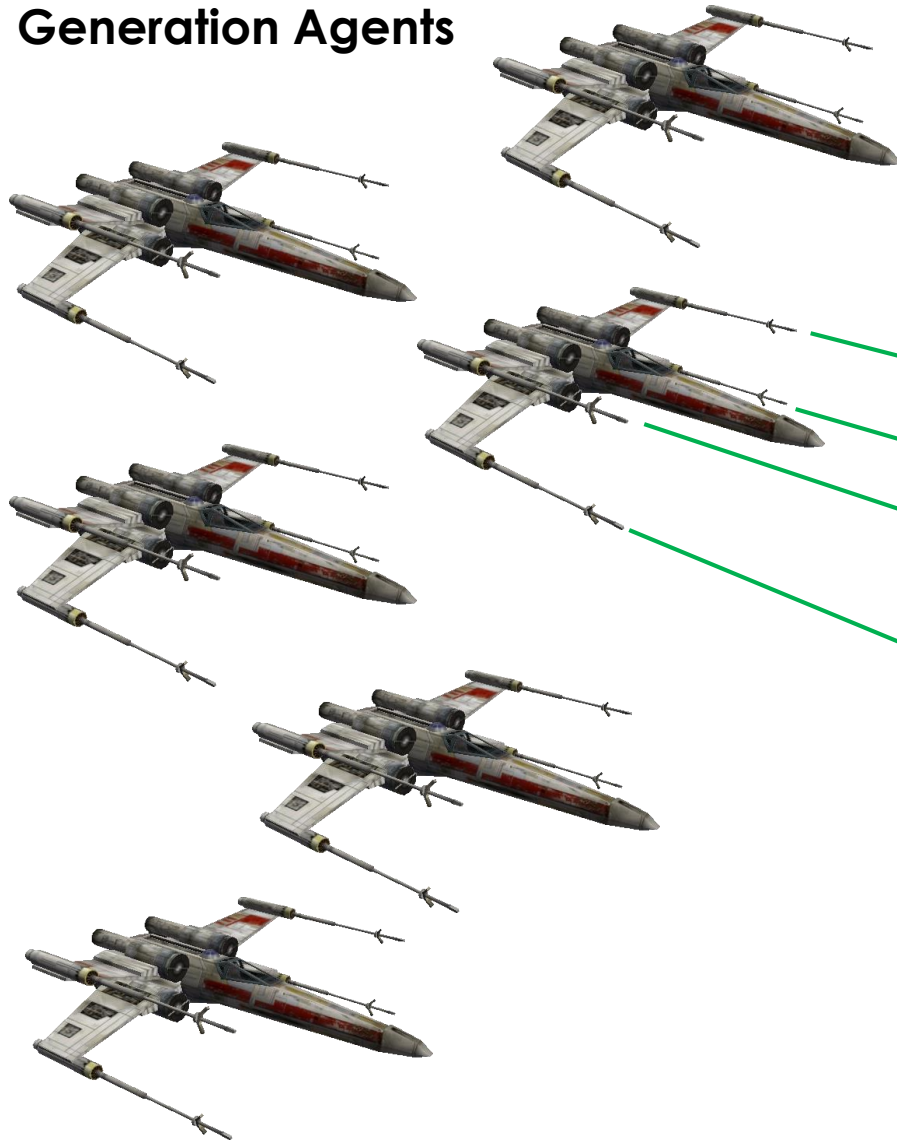
System Under Test

Distributed Load Generation Agents

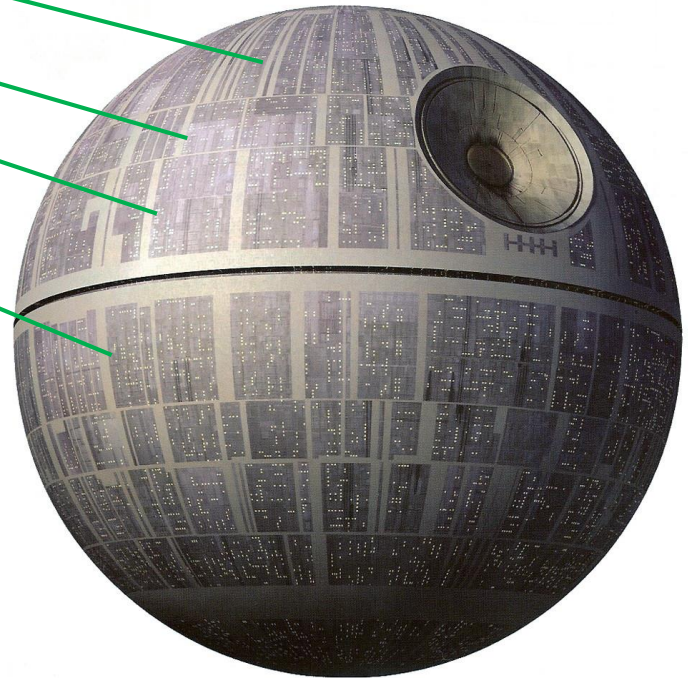
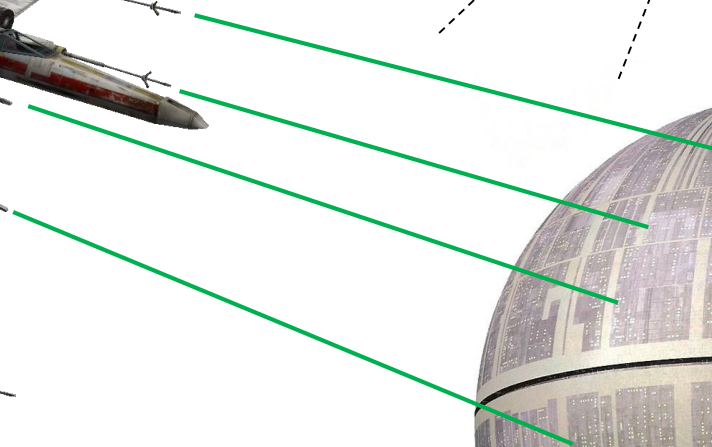
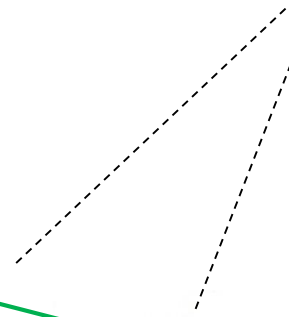
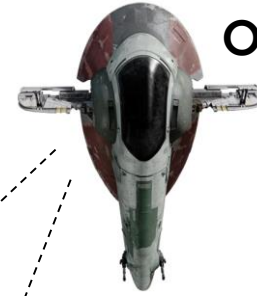


System Under Test

**Distributed Load
Generation Agents**



Observer

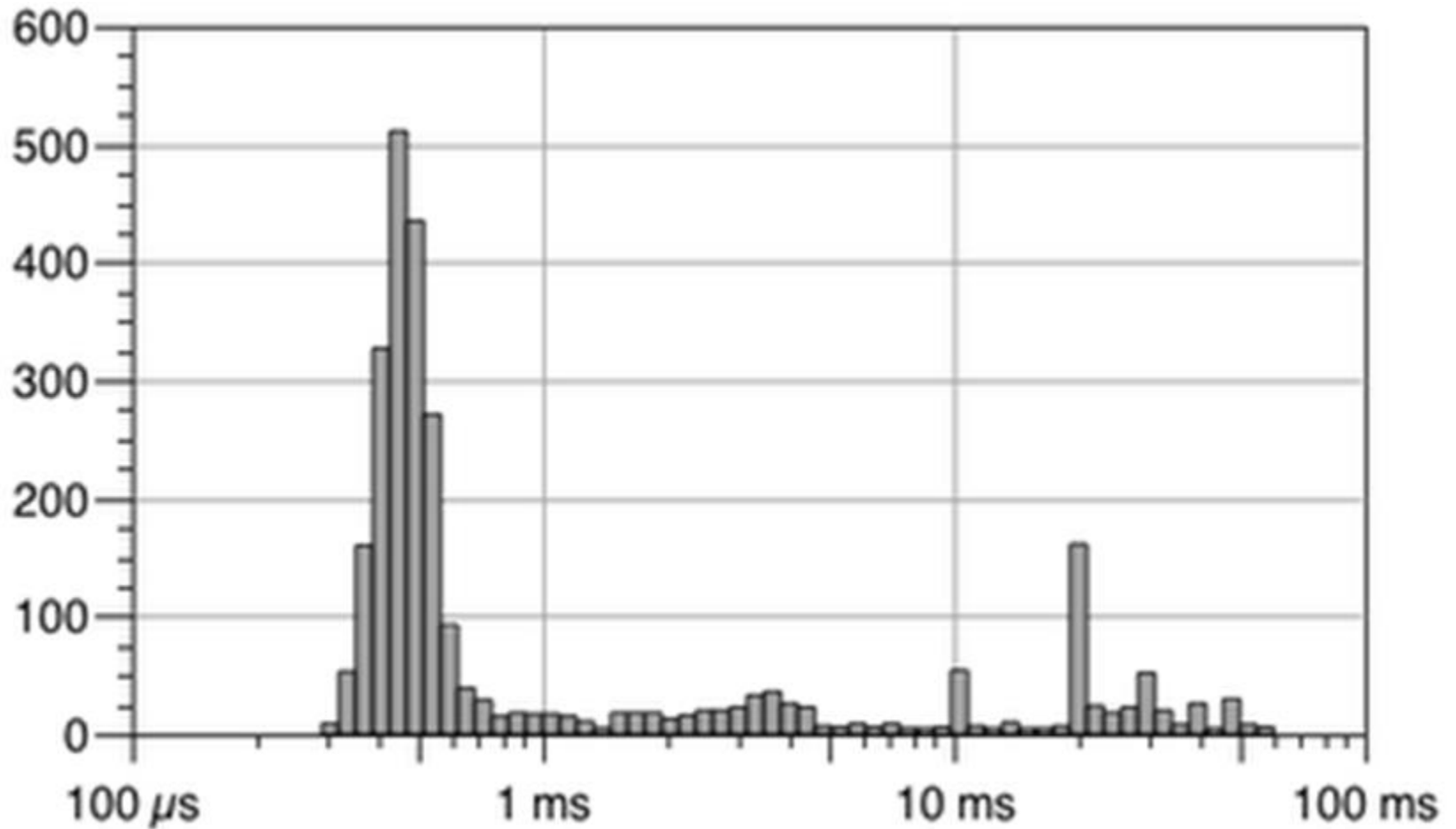


System Under Test

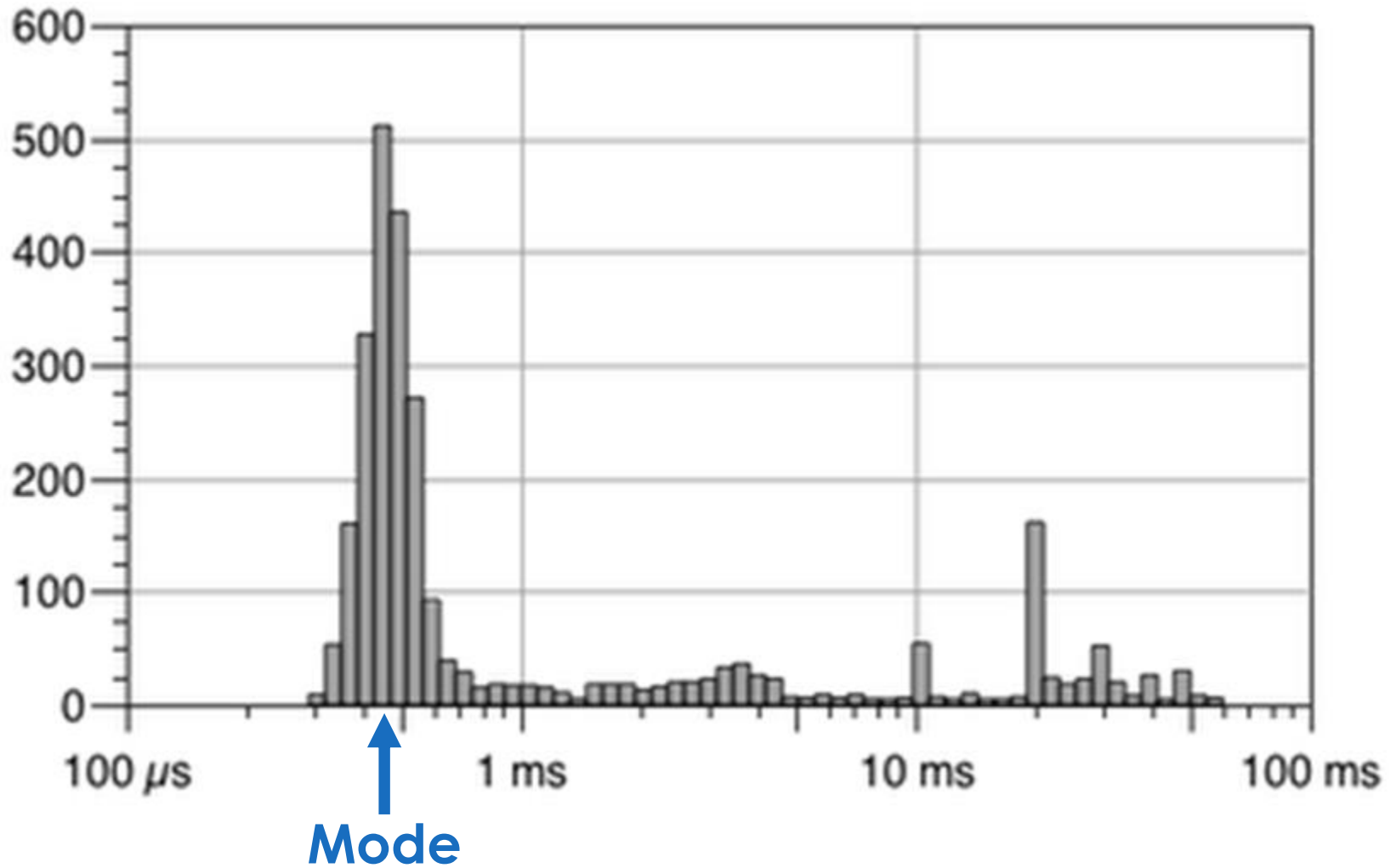
Pro Tip: Setup a continuous performance testing environment

Pro Tip: Record Everything

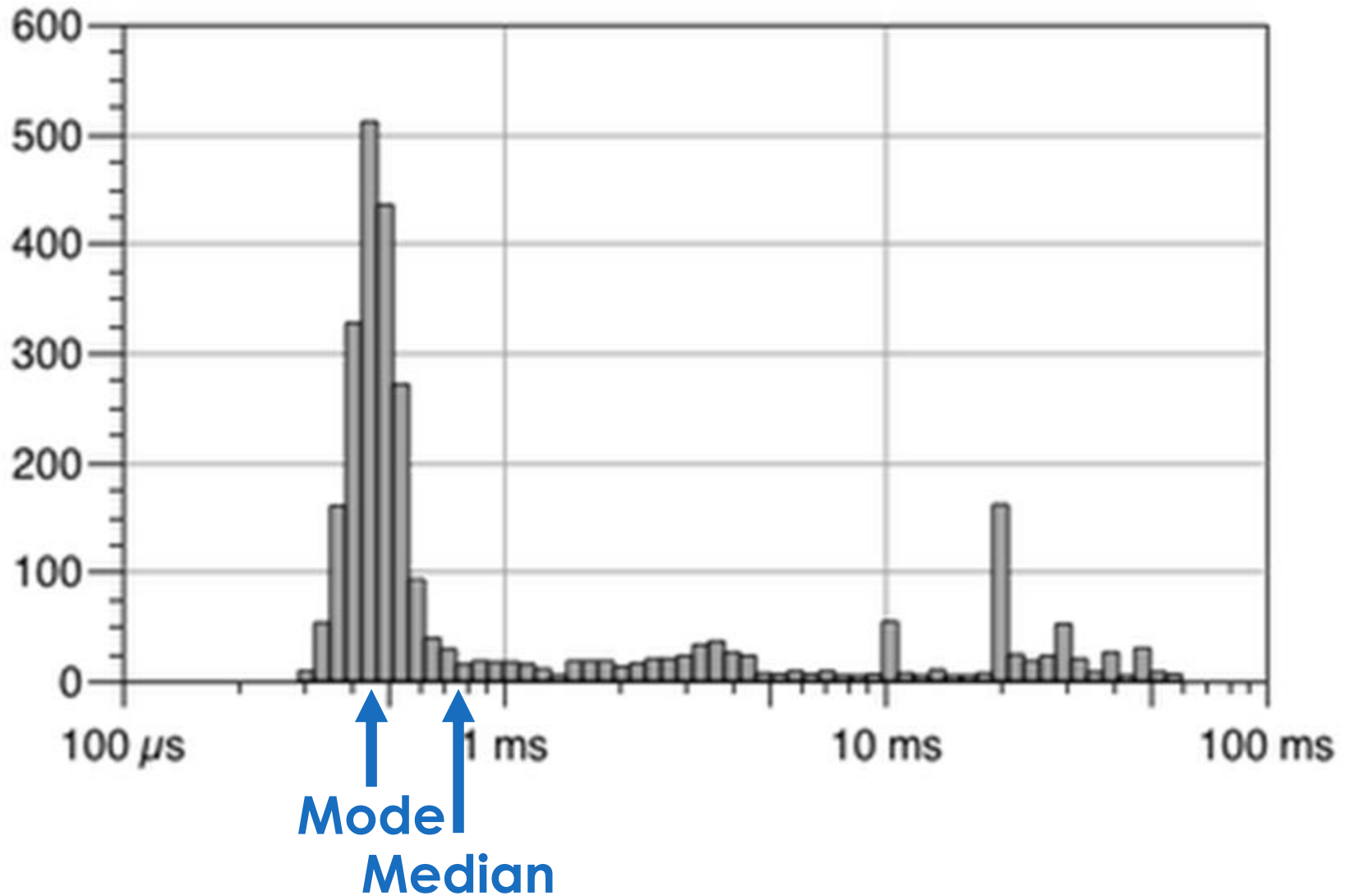
Latency Histograms



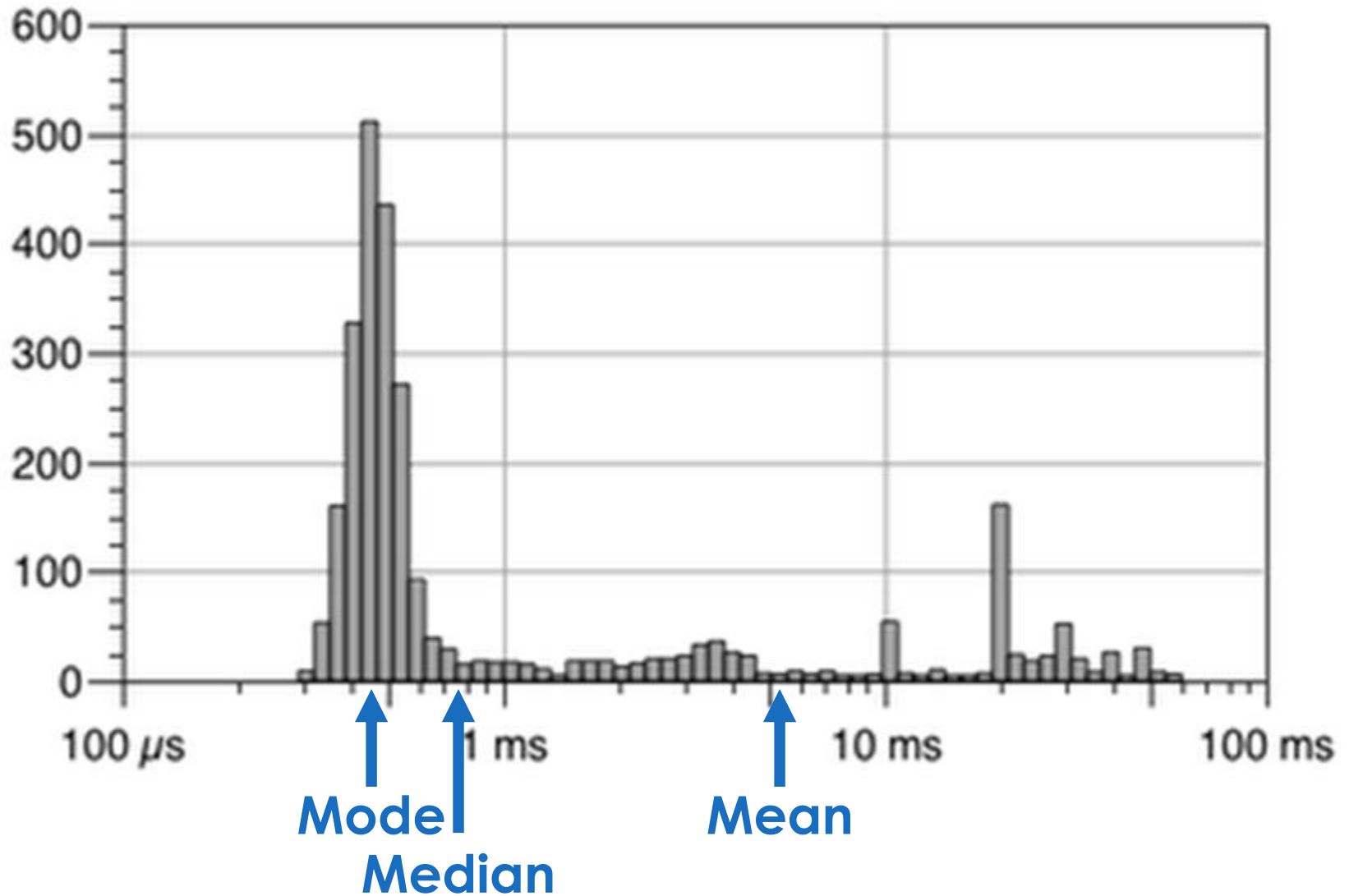
Latency Histograms



Latency Histograms



Latency Histograms



System: 1000 TPS, mean RT 50 μ s

System: 1000 TPS, mean RT 50 μ s

**What is the mean if you add in a
25ms GC pause per second?**

System: 1000 TPS, mean RT 50 μ s

**What is the mean if you add in a
25ms GC pause per second?**

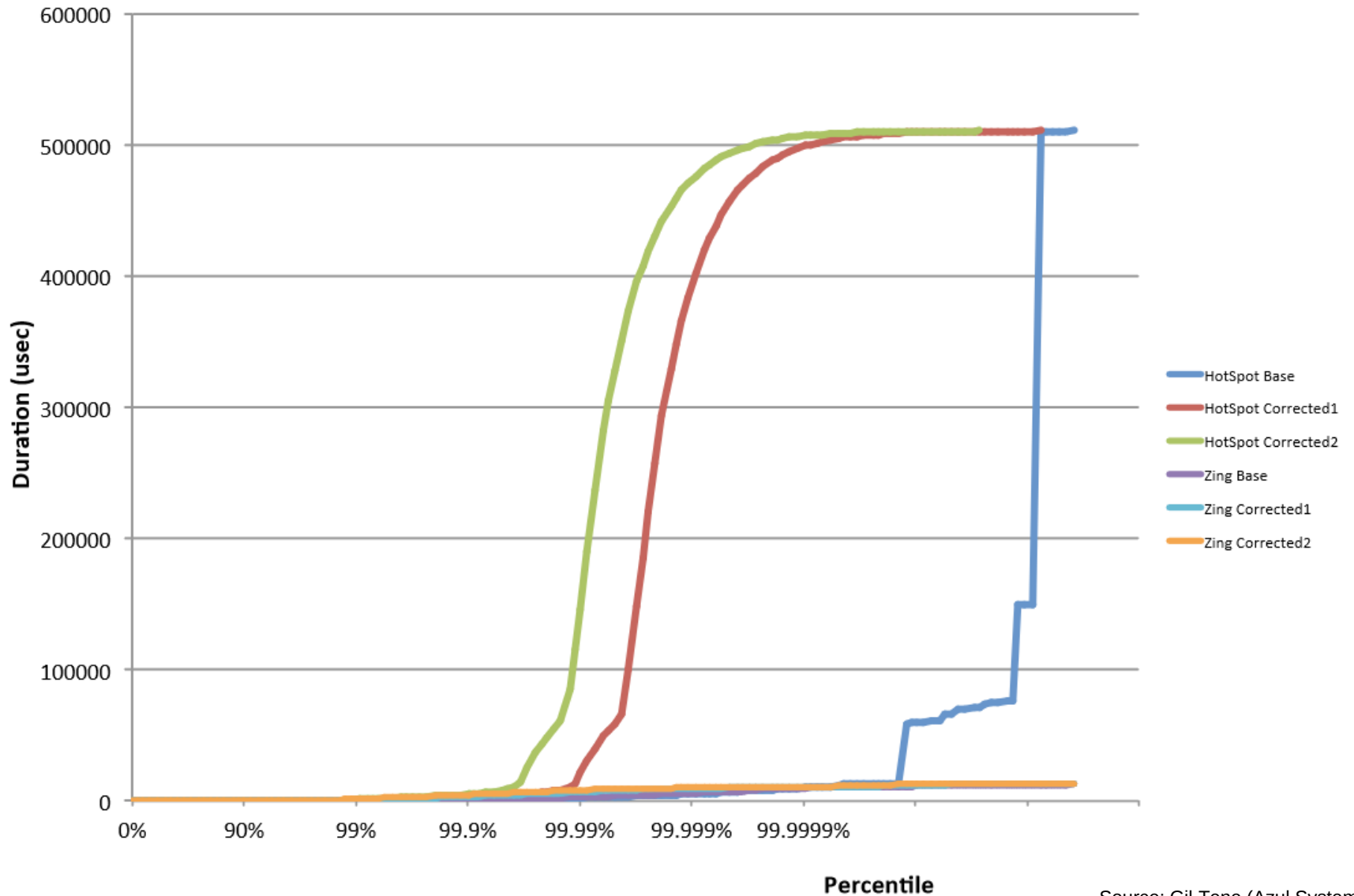
~300 μ s



***Forget averages,
it's all about percentiles***

Coordinated Omission

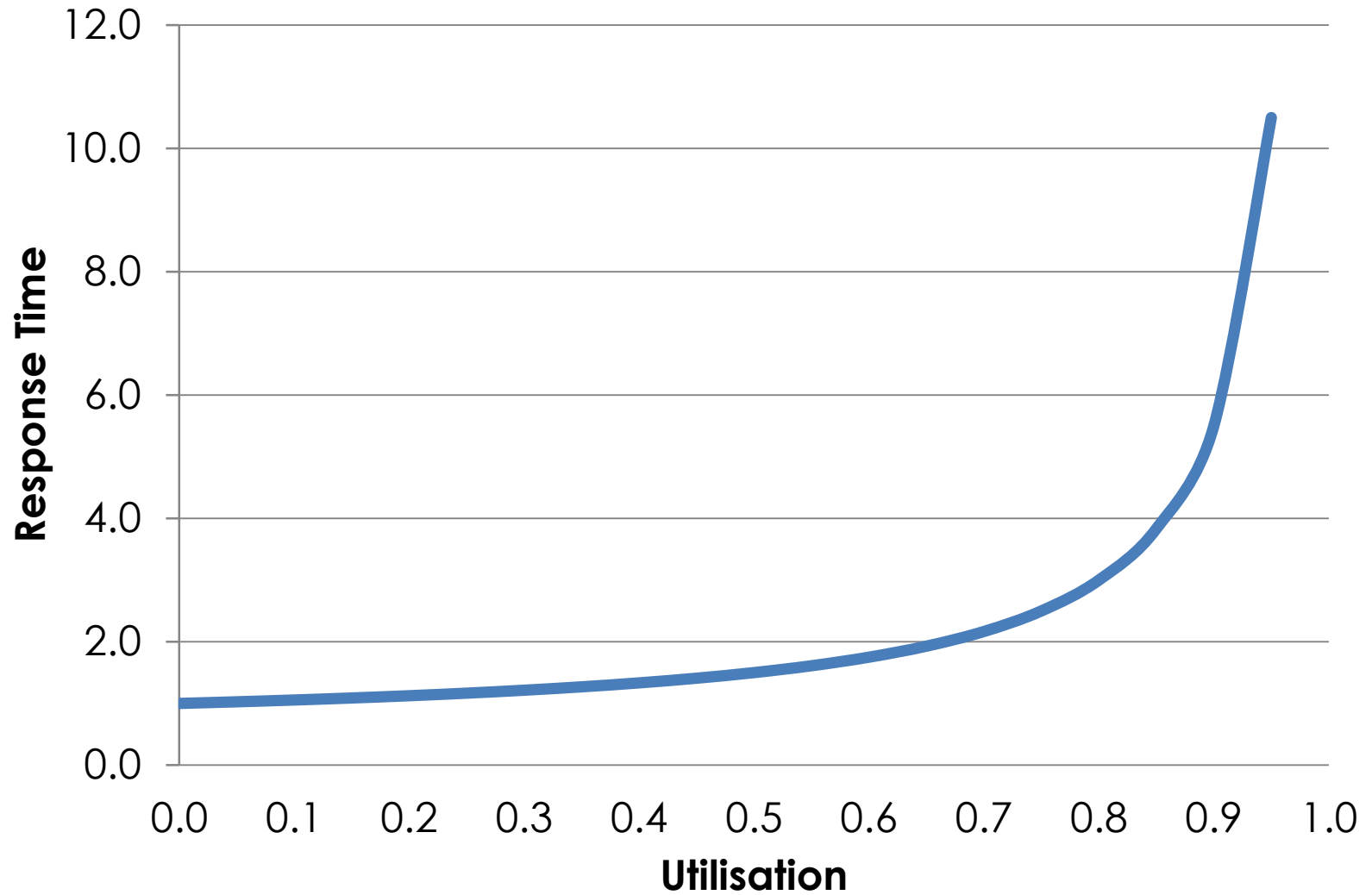
Duration by Percentile Distribution



Pro Tip: Don't deceive yourself

Theory

Queuing Theory



Queuing Theory

Kendall Notation

M/D/1

Queuing Theory

$$r = s (2 - \rho) / 2 (1 - \rho)$$

r = mean response time

s = service time

ρ = utilisation

Queuing Theory

$$r = s(2 - \rho) / 2(1 - \rho)$$

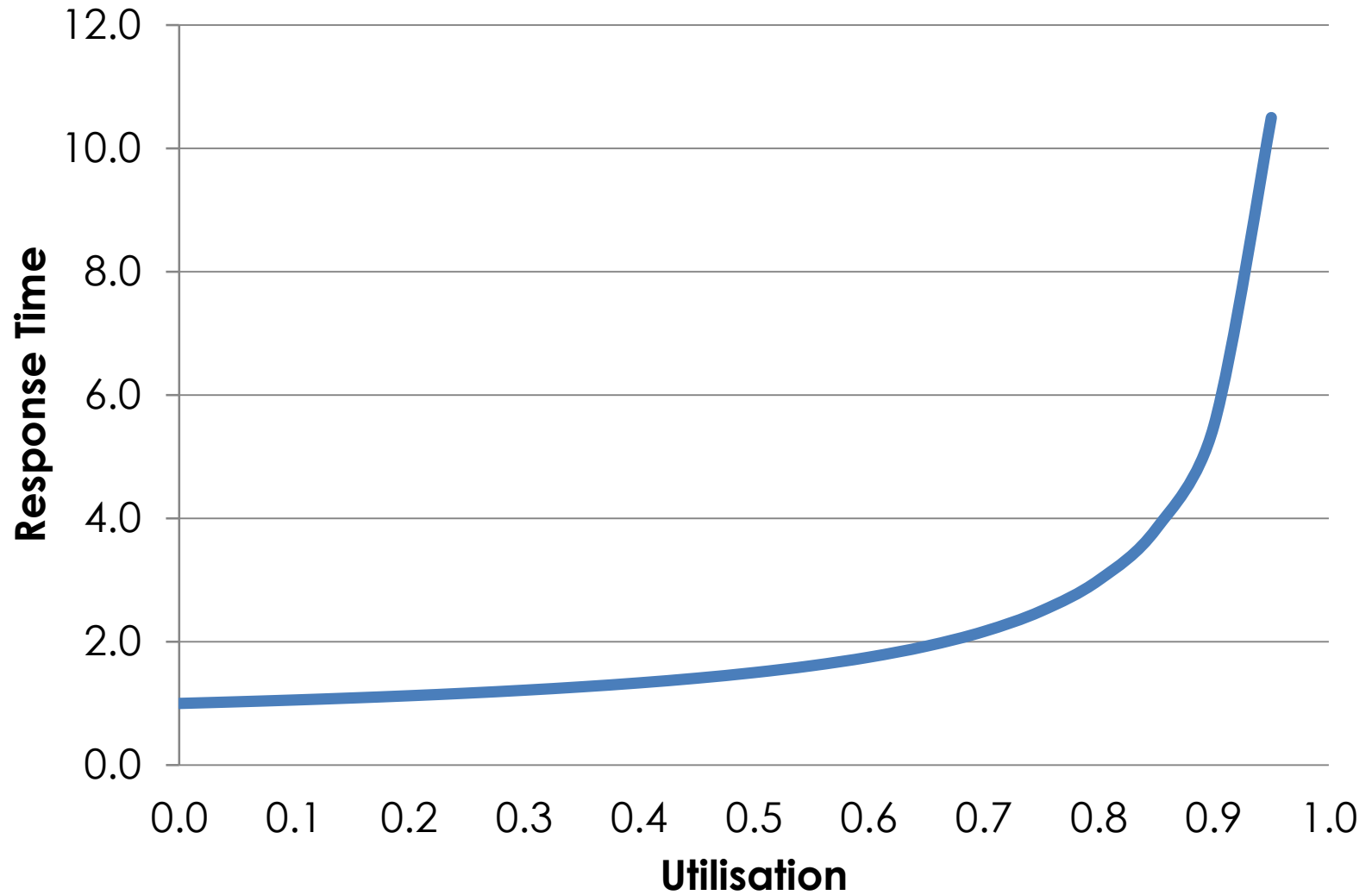
r = mean response time

s = service time

ρ = utilisation

Note: $\rho = \lambda * (1 / s)$

Queuing Theory



Pro Tip: Ensure that you have sufficient capacity

Queuing Theory

Little's Law: $L = \lambda * W$

L = mean queue length

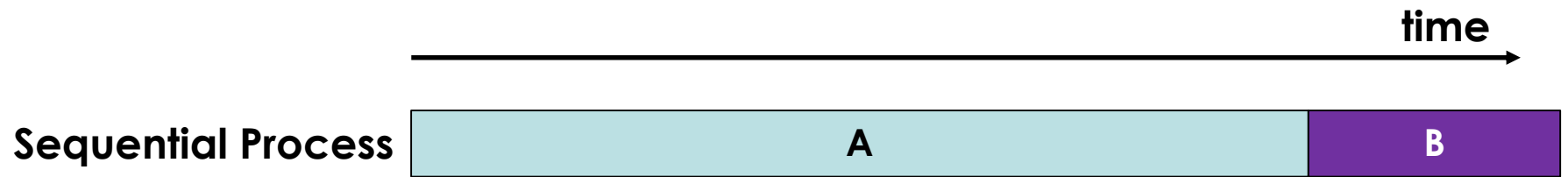
λ = mean arrival rate

W = mean time in system

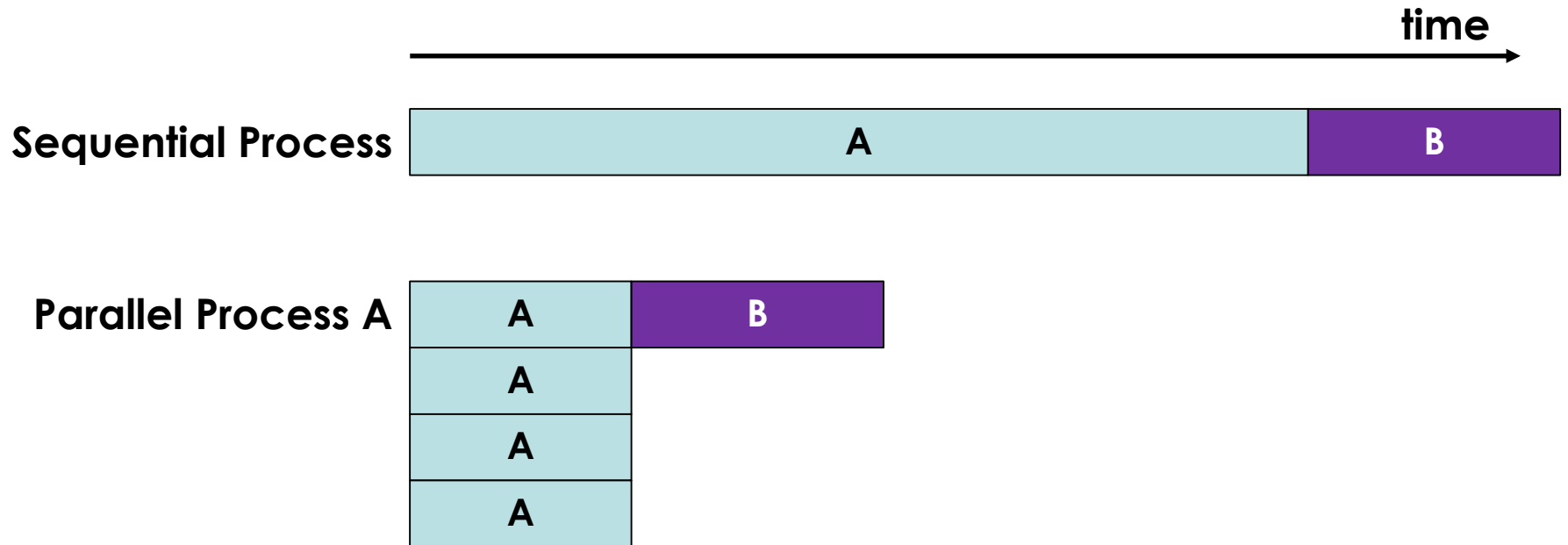
Pro Tip: Bound queues to meet response time SLAs

**Can we go parallel to
speedup?**

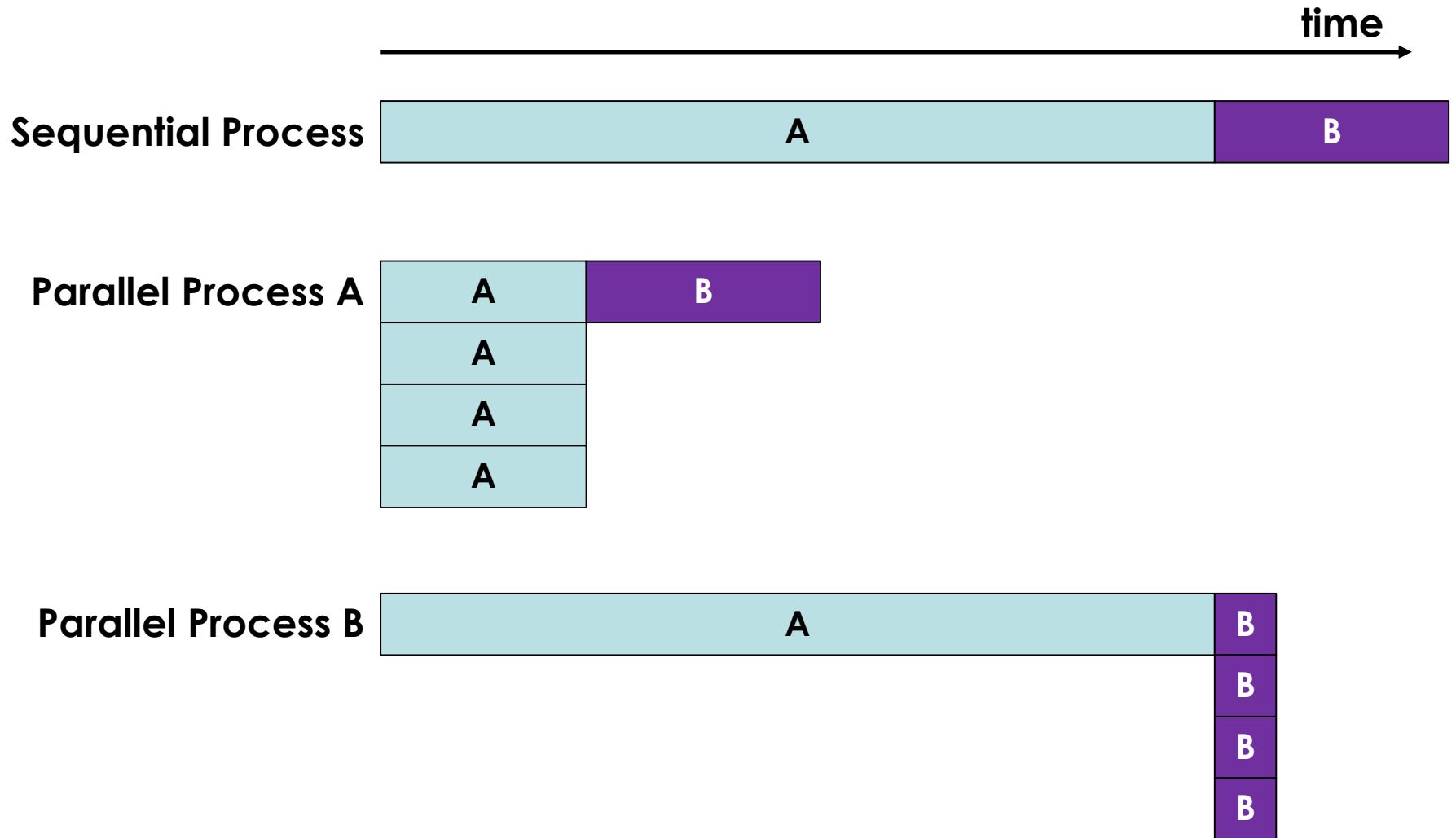
Amdahl's Law



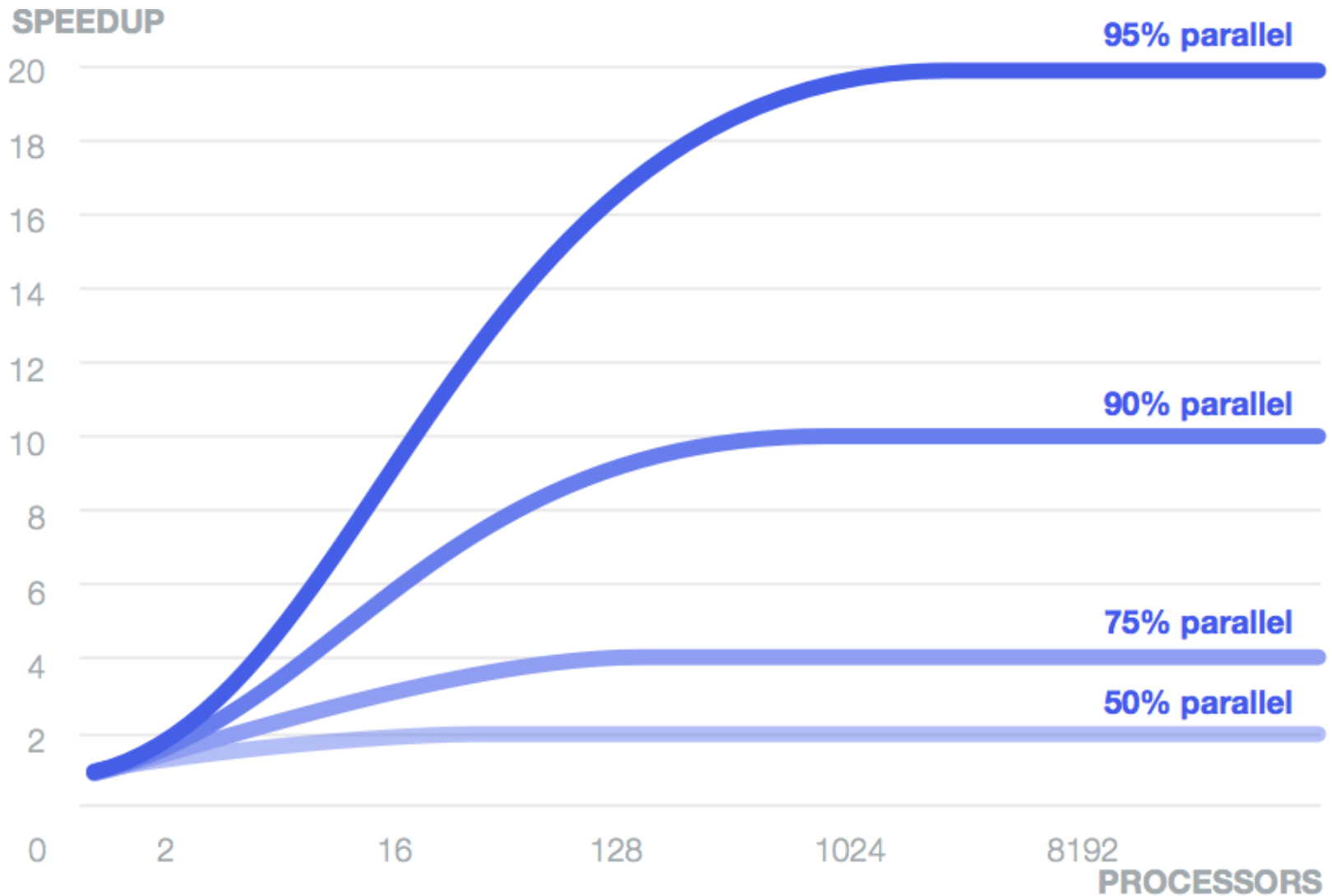
Amdahl's Law



Amdahl's Law



Amdahl's Law



Universal Scalability Law

$$C(N) = N / (1 + \alpha(N - 1) + ((\beta * N) * (N - 1)))$$

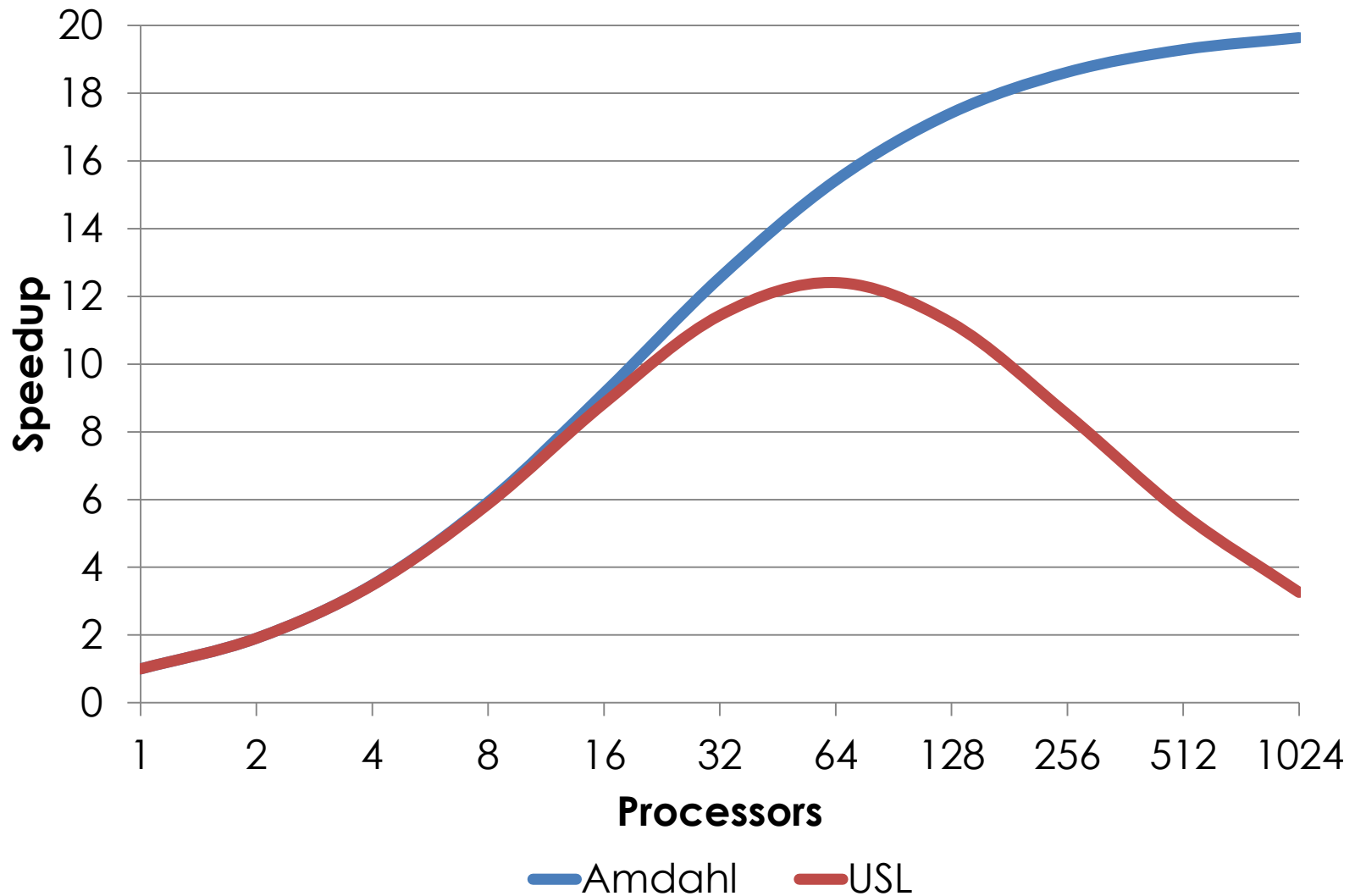
C = capacity or throughput

N = number of processors

α = **contention** penalty

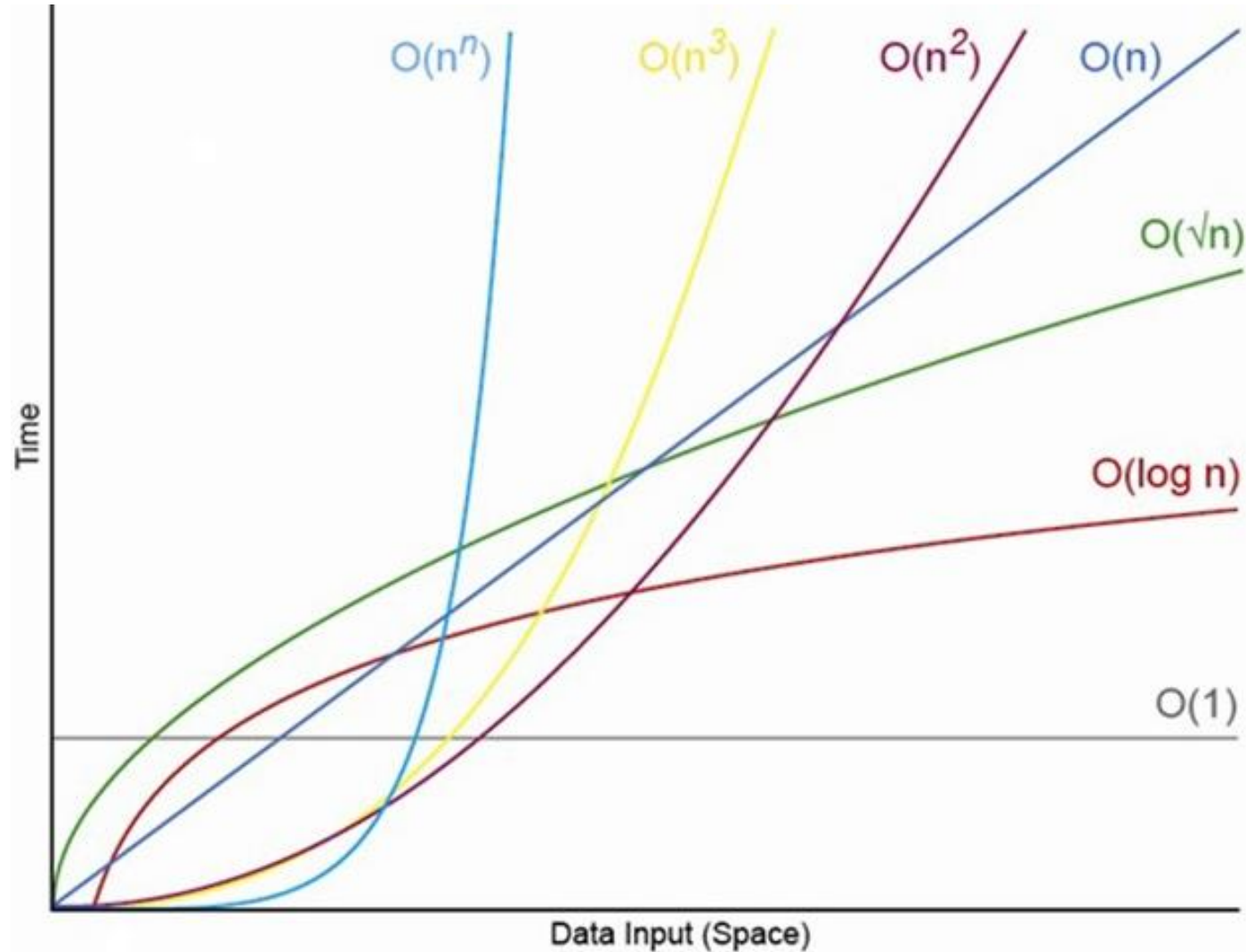
β = **coherence** penalty

Universal Scalability Law

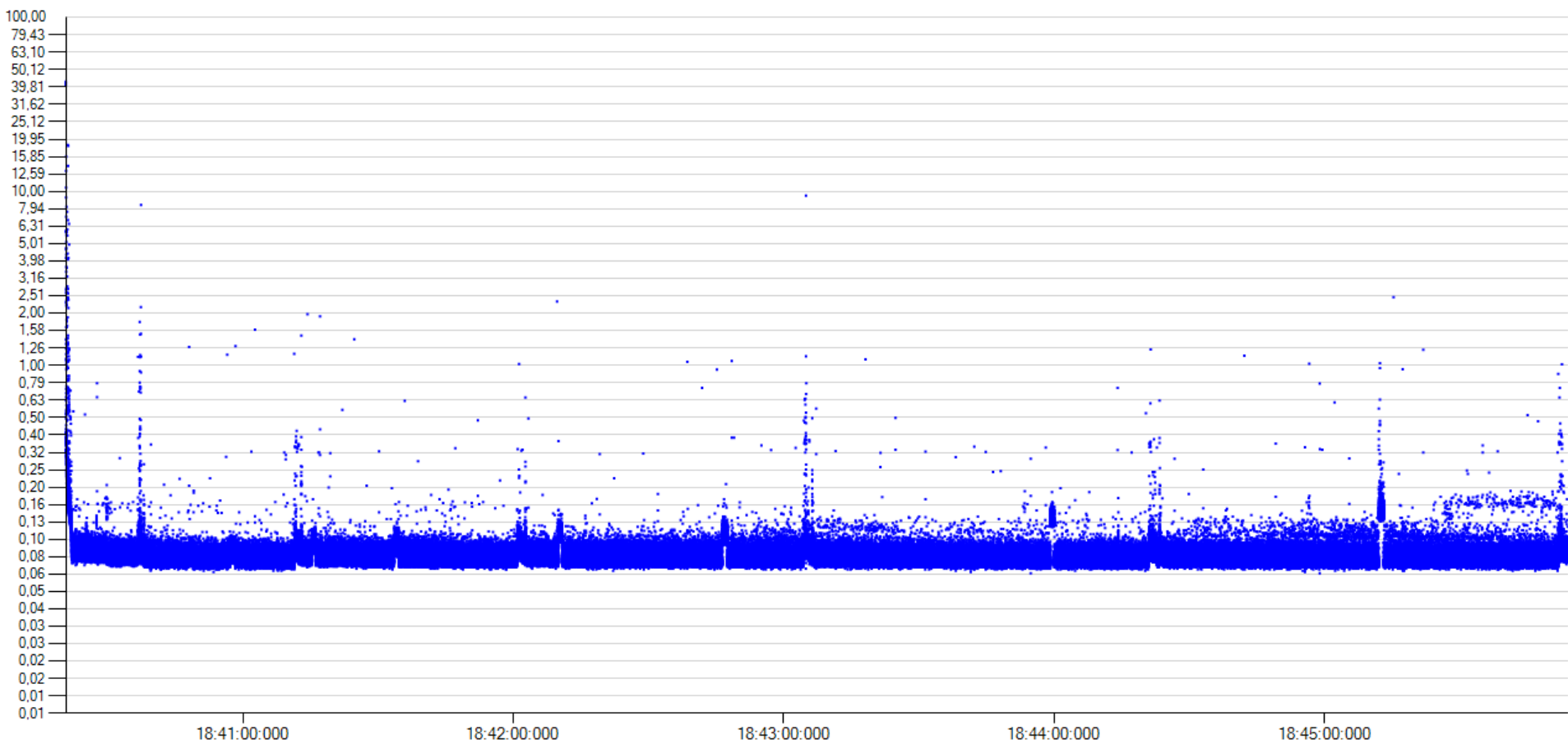


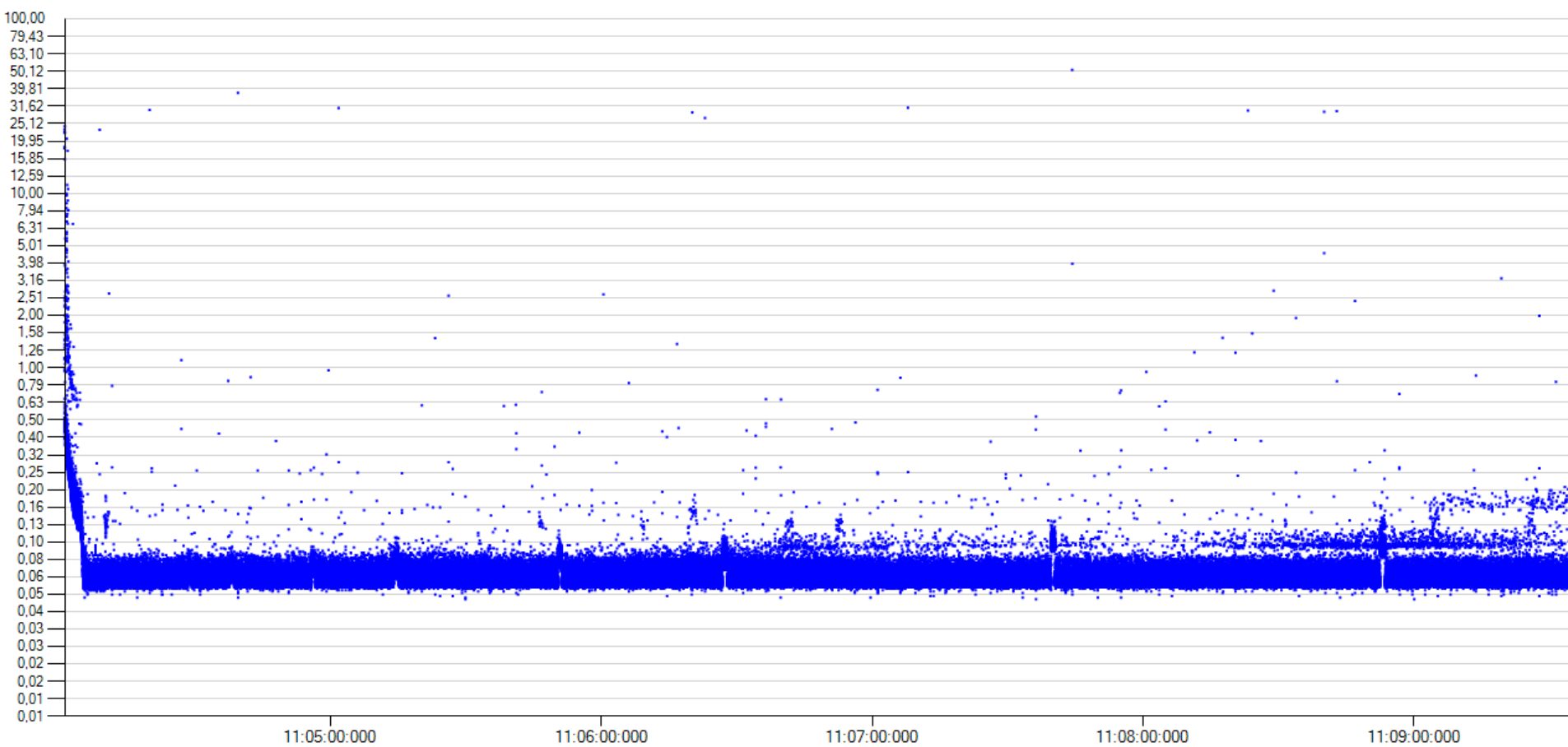
What about the service time?

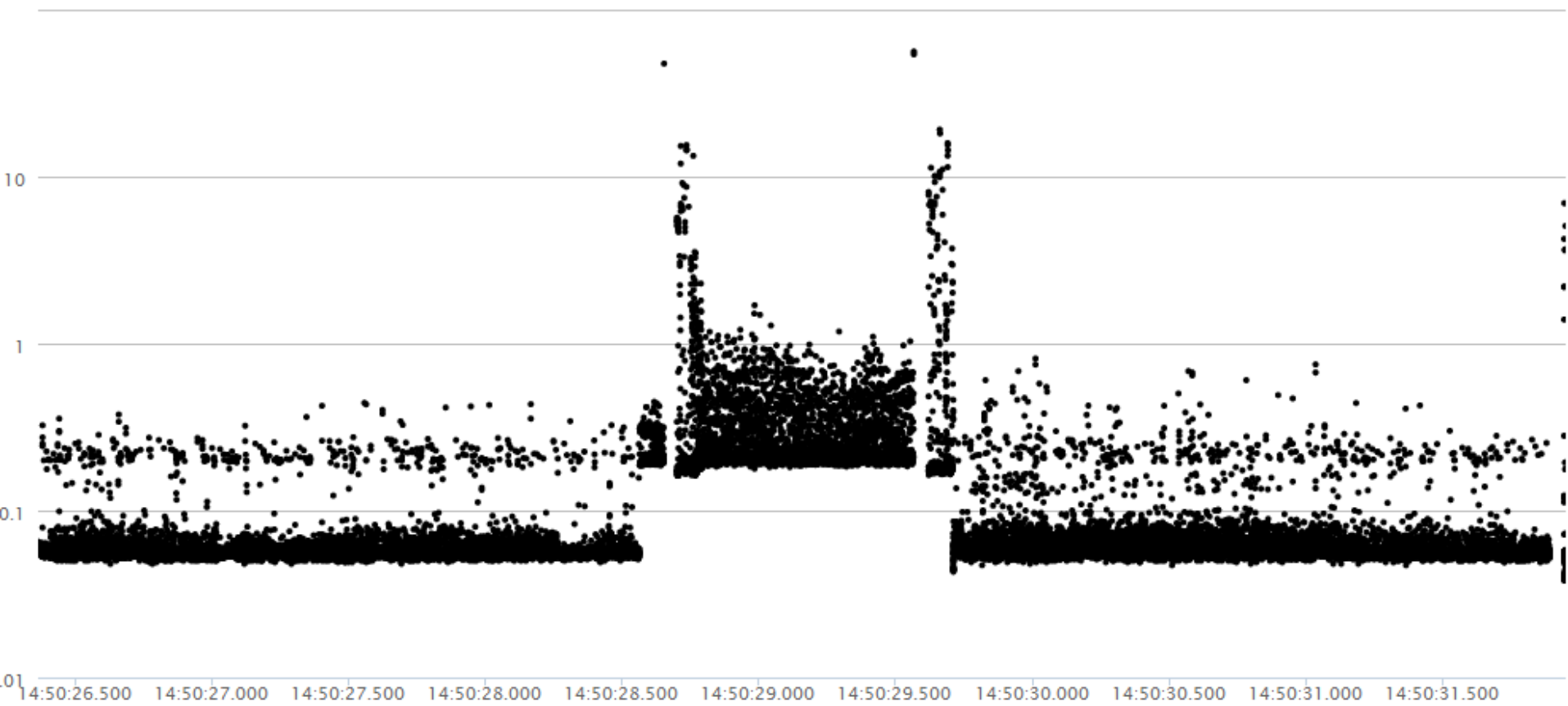
Order of Algorithms

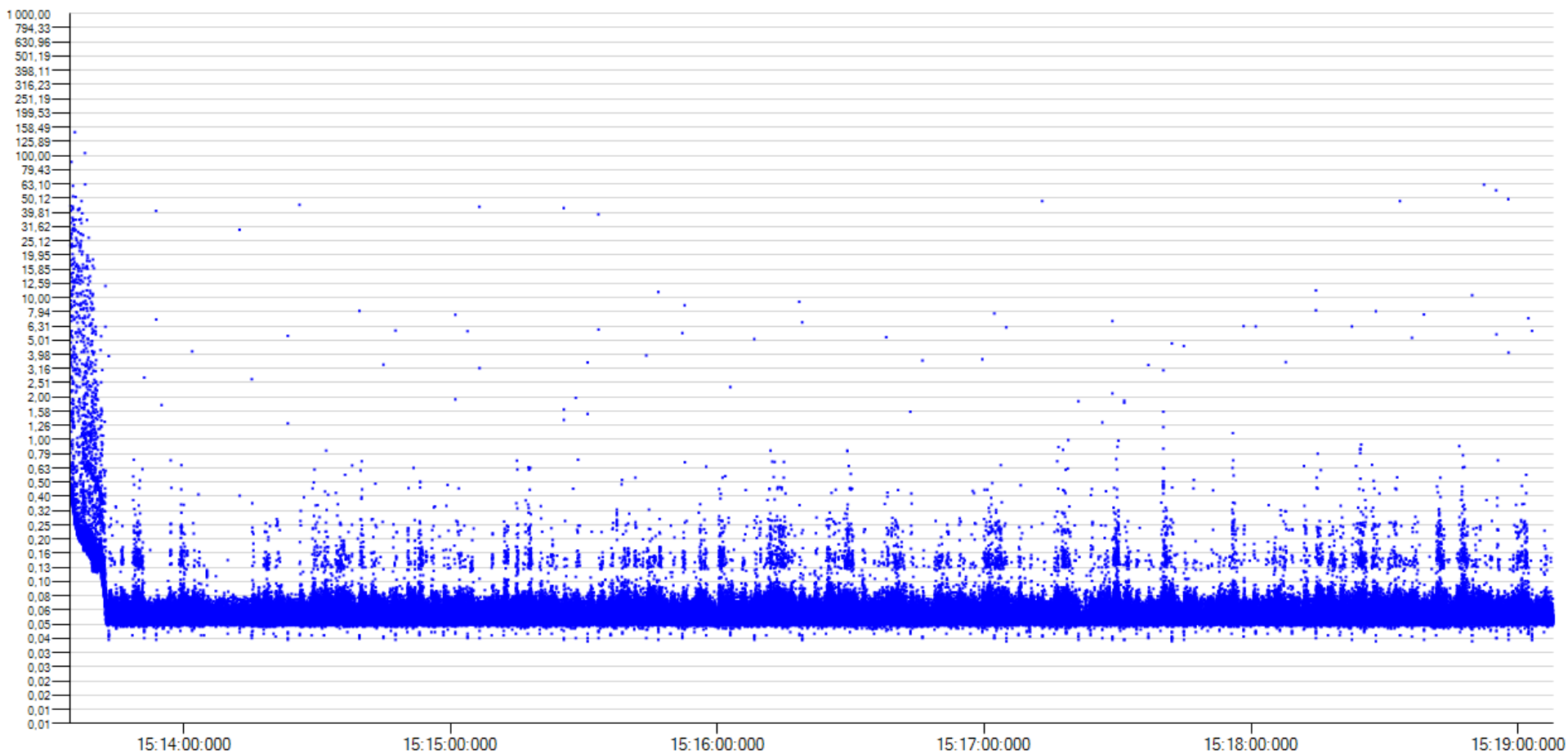


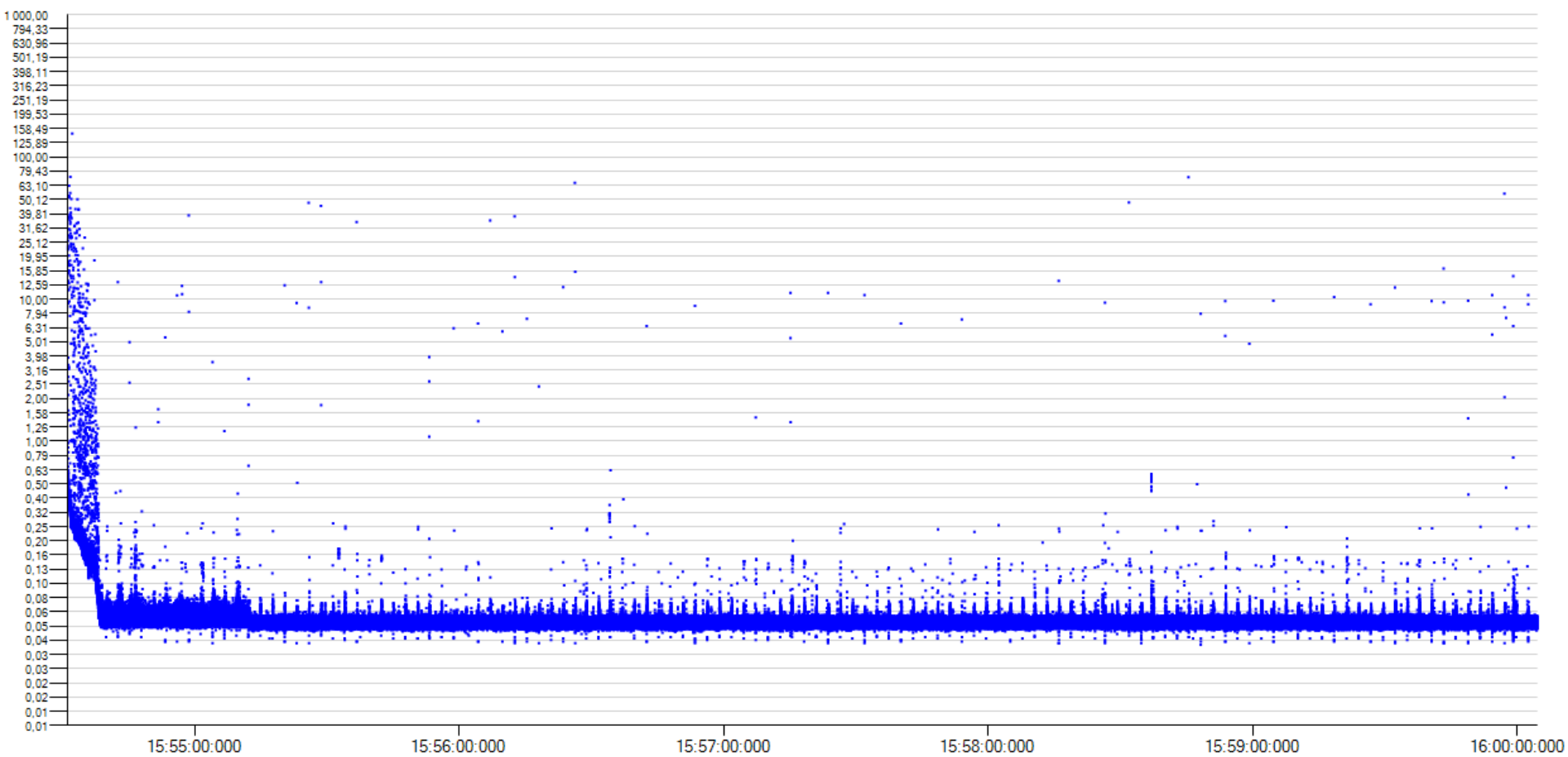
Practice







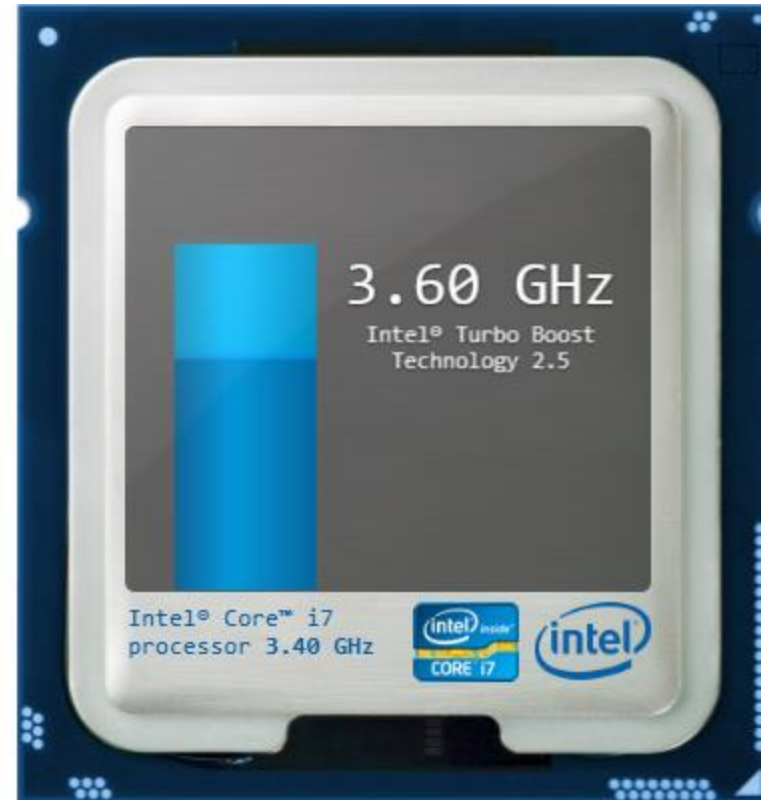




Pitfalls

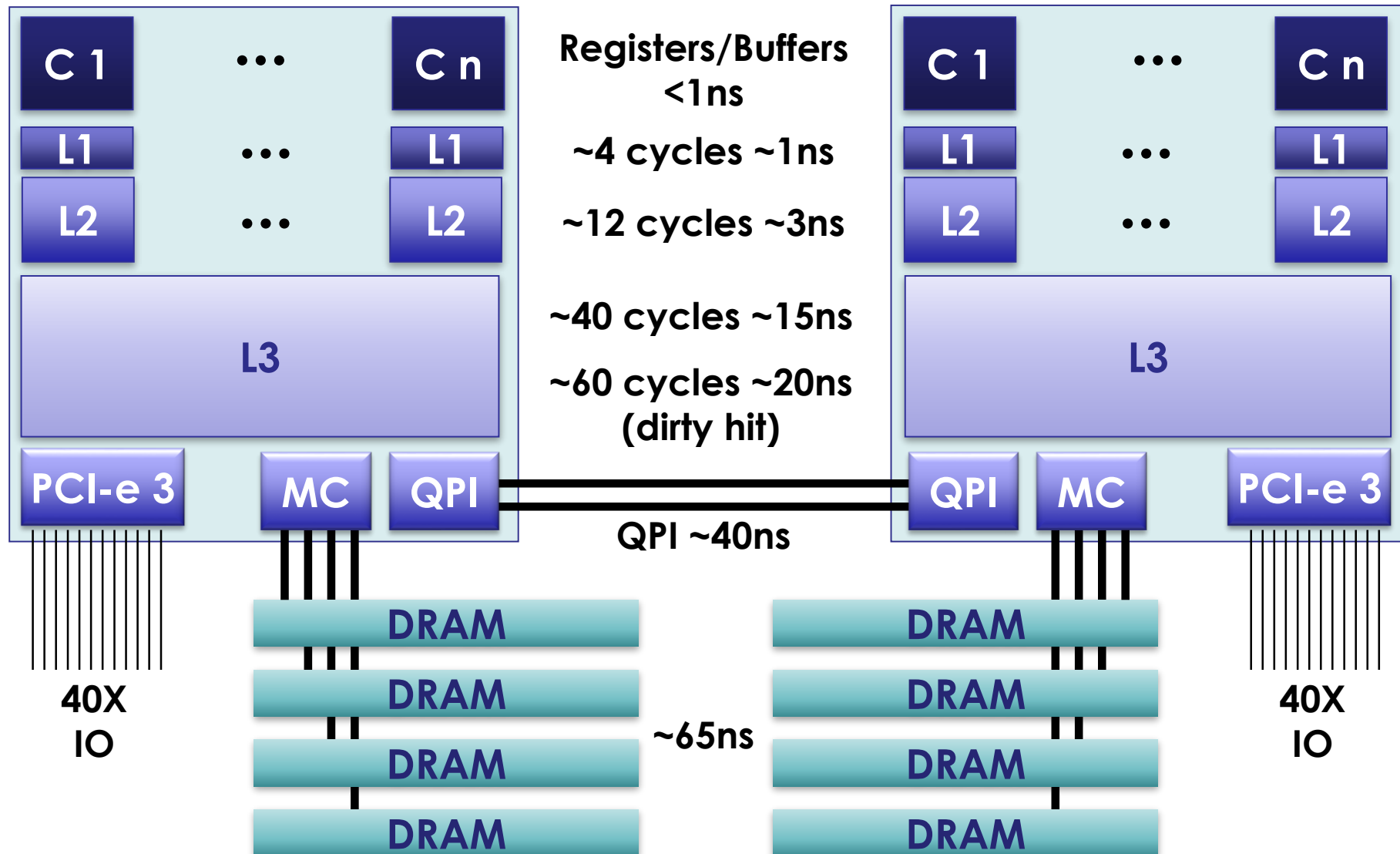
Modern Processors

SMTs?



Hyperthreading?

Non-Uniform Memory Architecture (NUMA)



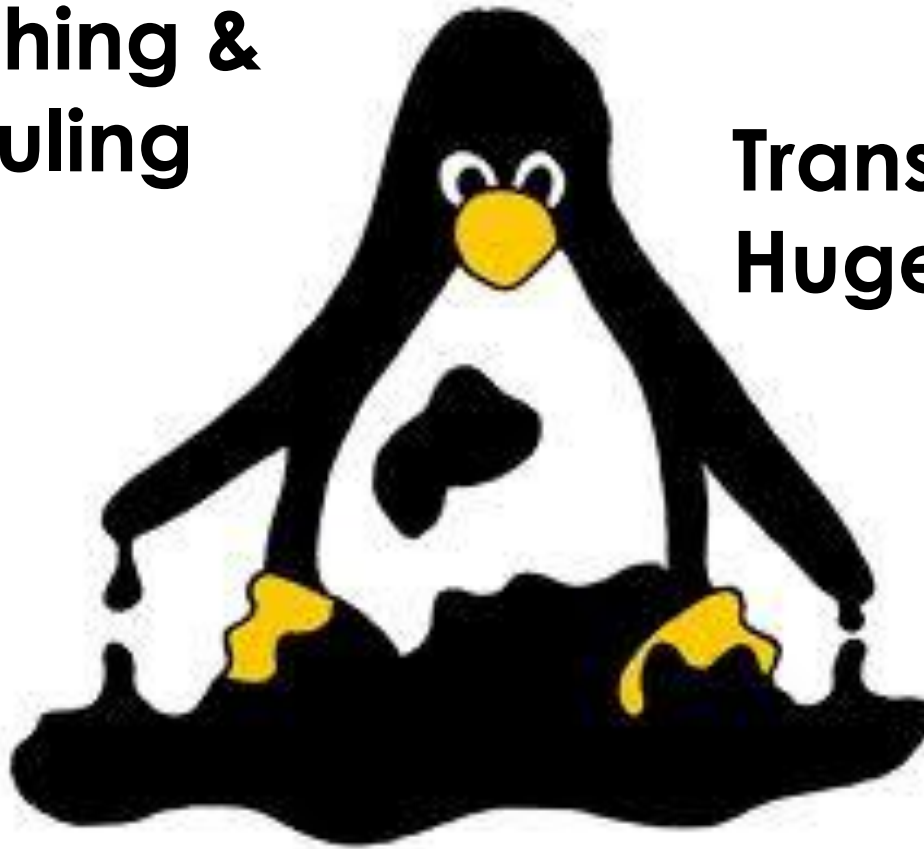
* Assumption: 3GHz Processor

Virtual Memory Management

**Page Flushing &
IO Scheduling**

**Transparent
Huge Pages**

Swap???



vm.min_free_kbytes

Safepoints in the JVM



**Garbage Collection, De-optimisation,
Biased Locking, Stack traces, etc.**

Virtualization



System Calls

Notification

```
public class SomethingUseful
{
    // Lots of useful stuff

    public void handOffSomeWork()
    {
        // prepare for handoff

        synchronized (this)
        {
            someObject.notify();
        }
    }
}
```

Notification

```
public class SomethingUseful
{
    // Lots of useful stuff

    public void handOffSomeWork()
    {
        // prepare for handoff

        synchronized (this)
        {
            someObject.
        }
    }
}
```



Law of Leaky Abstractions

***“All non-trivial abstractions,
to some extent, are leaky.”***

- Joel Spolsky

Law of Leaky Abstractions

“The detail of underlying complexity cannot be ignored.”

Mechanical Sympathy

Responding in the presence of failure



IT'S ONLY
A FLESH WOUND

Algorithms & Techniques

Clean Room Experiments



- sufficient CPUs
- `intel_idle.max_cstate=0`
- `cpufreq`
- `isocpus`
- `numctl`, `cgroups`, `affinity`
- “Washed” SSDs
- network buffer sizing
- `jHiccup`

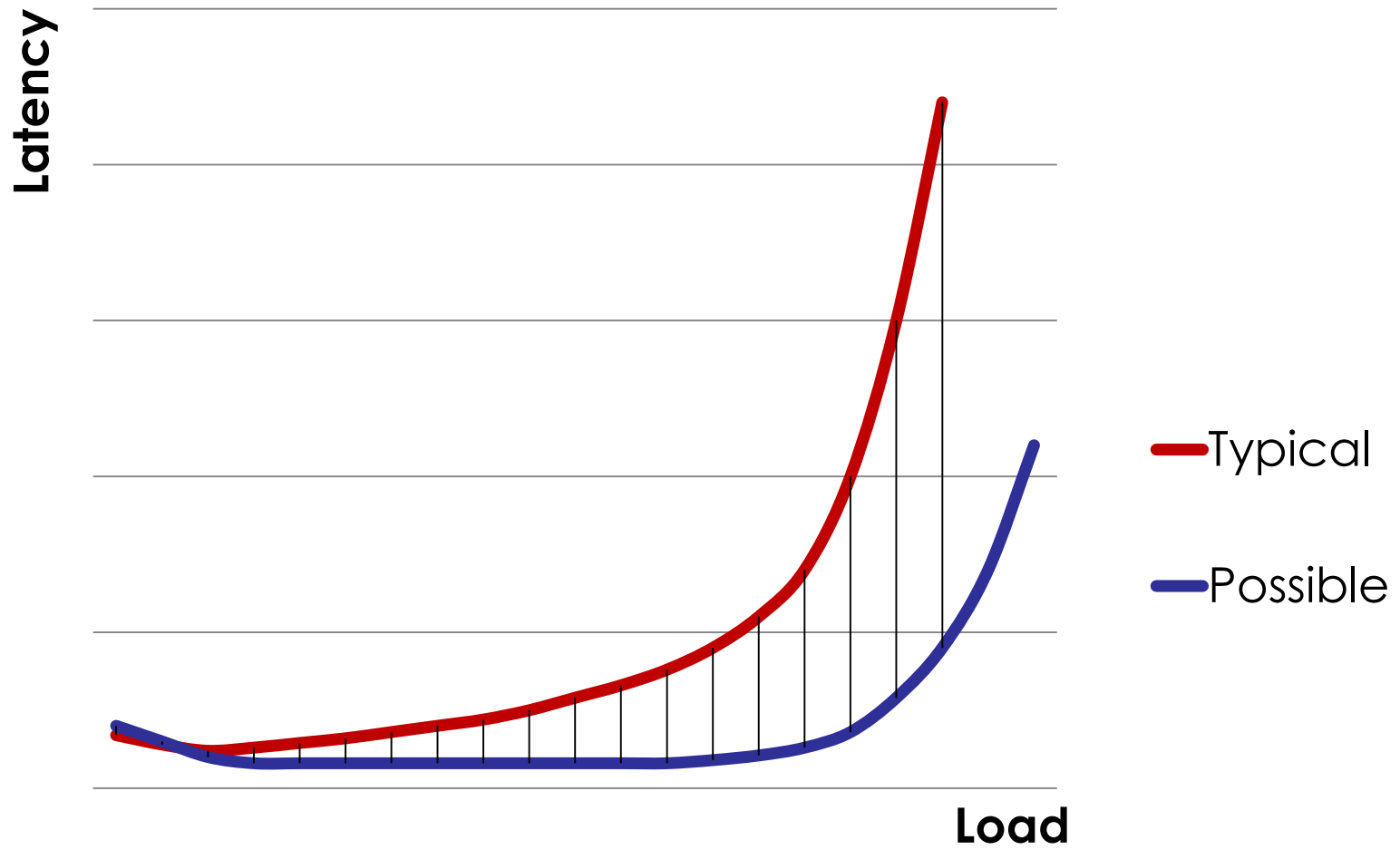
- tune your stack!
- Mechanical Sympathy

Profiling

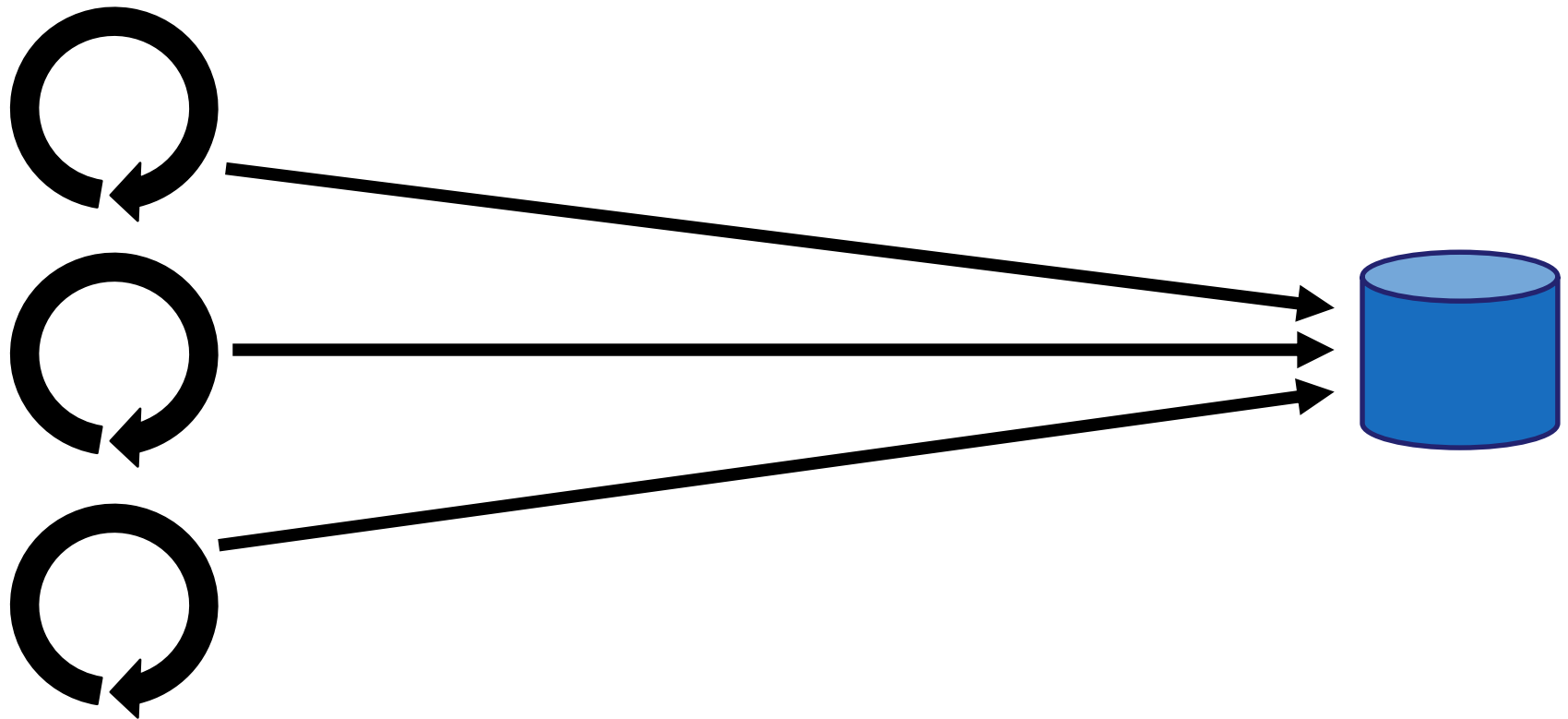


Pro Tip: Incorporate telemetry
and histograms

Smart Batching

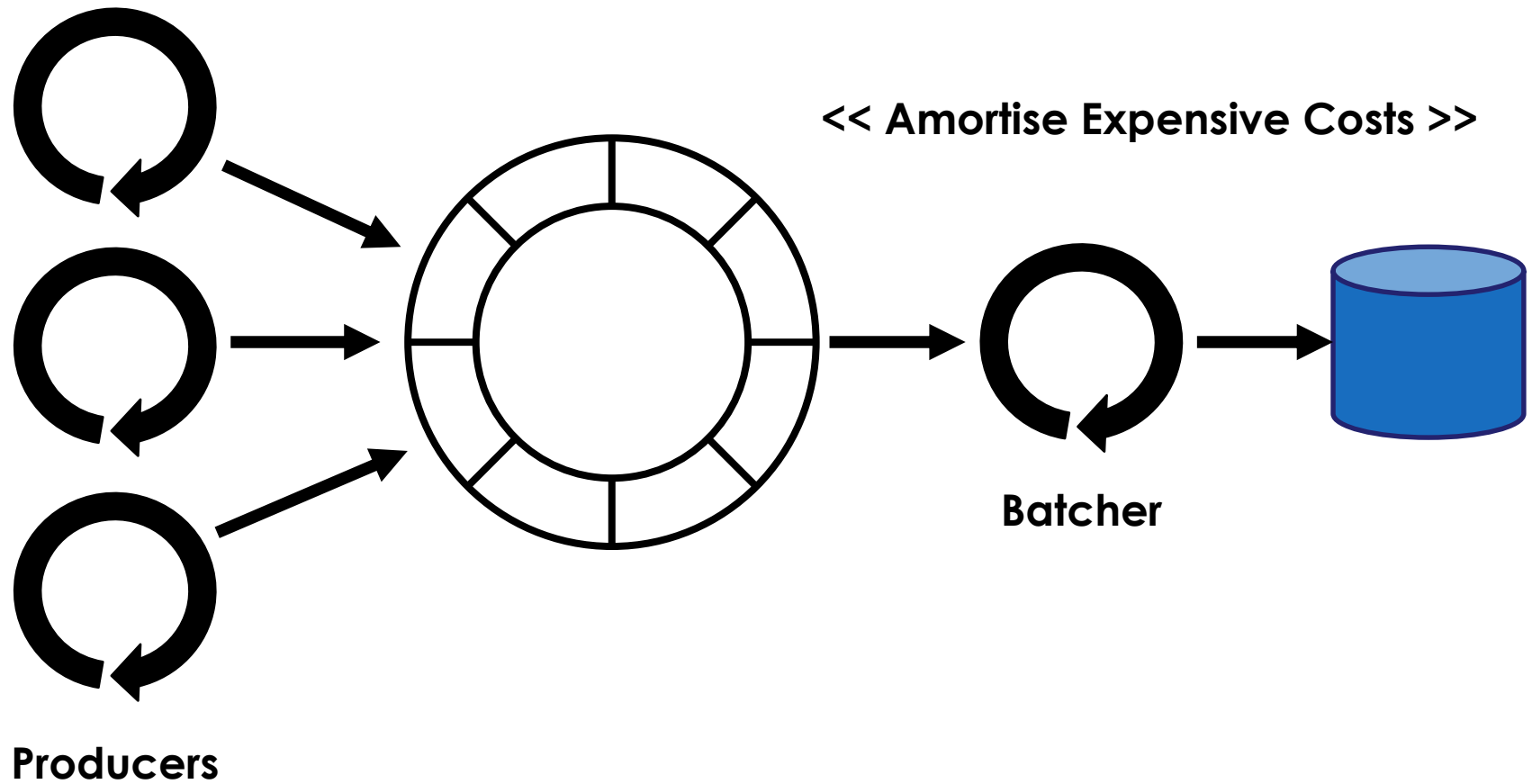


Smart Batching



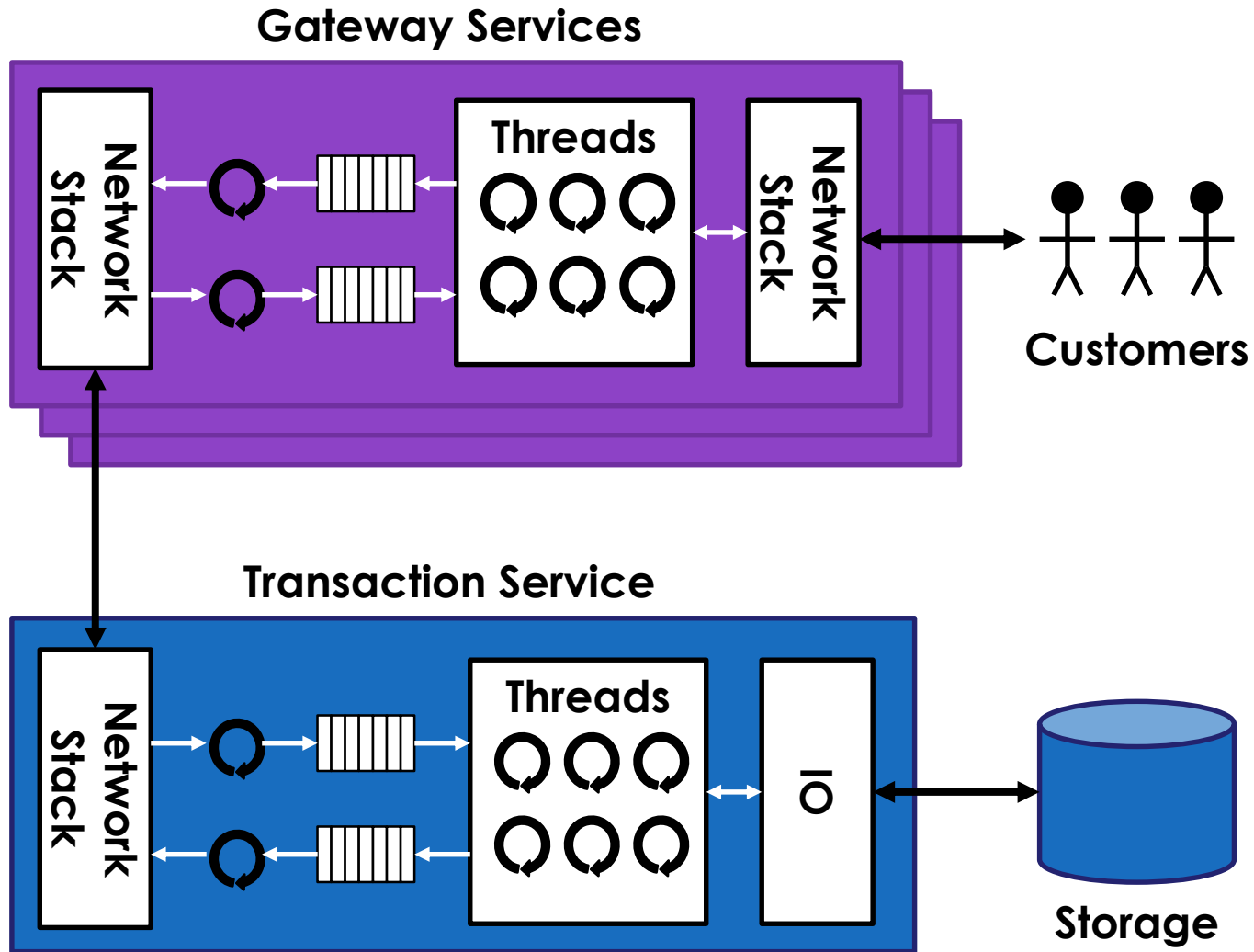
Producers

Smart Batching



Pro Tip: Amortise the
Expensive Costs

Applying Backpressure



Non-Blocking Design

“Get out of your own way!”

- **Don't hog any resource**
- **Always try to make progress**
- **Enables Smart Batching**

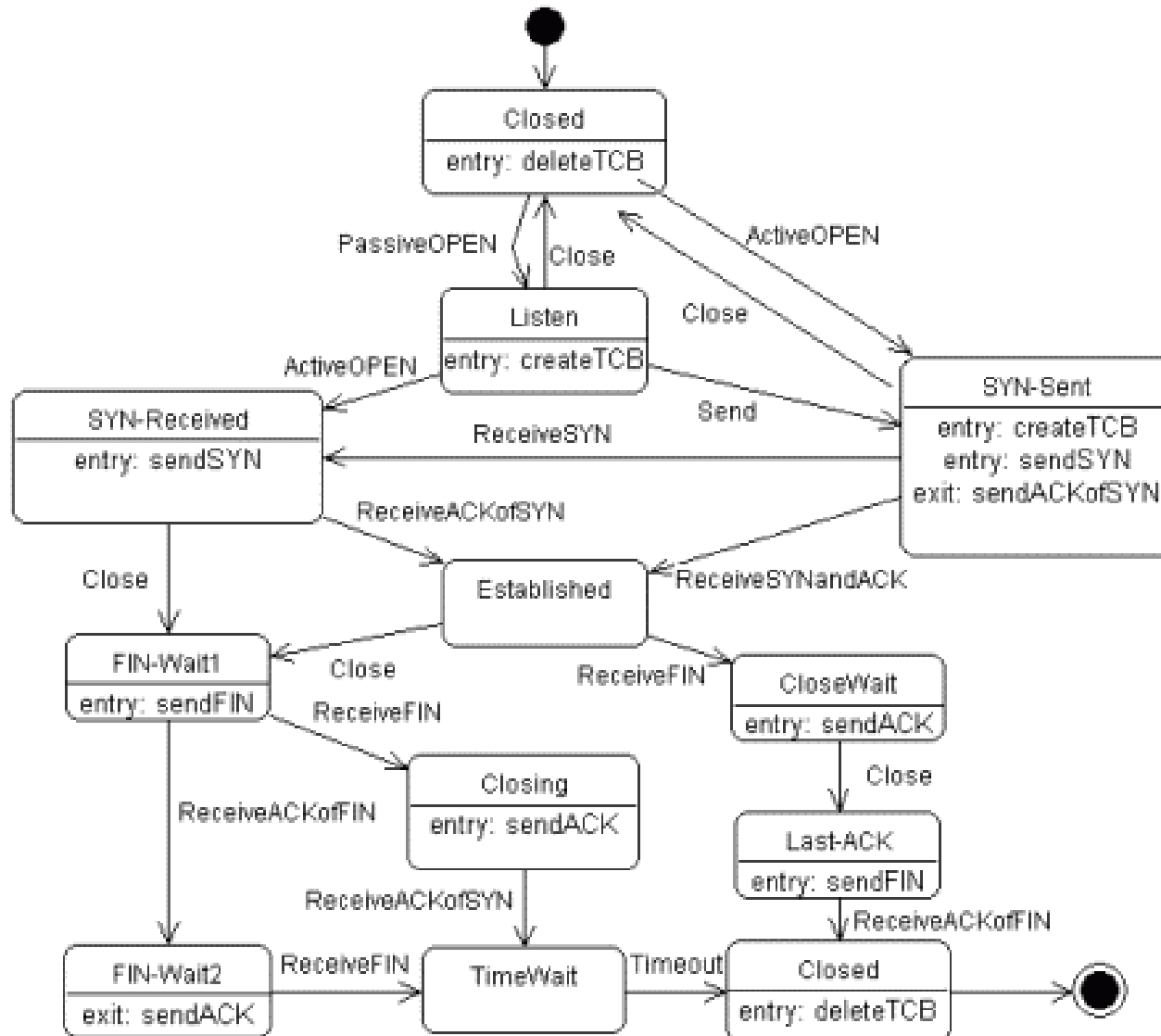
Pro Tip: Beware of
hogging resources in
synchronous designs

Lock-Free Concurrent Algorithms



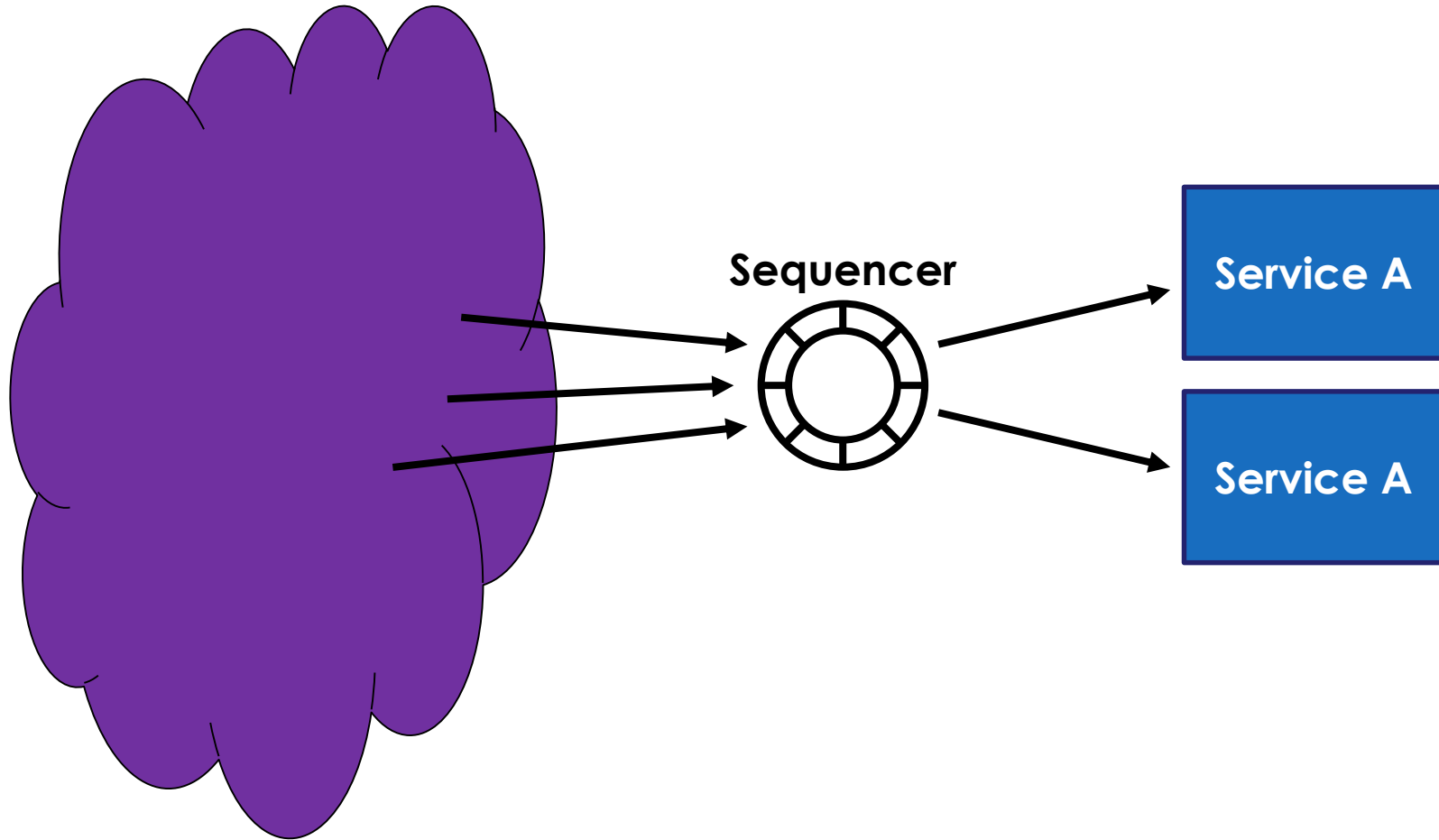
- Agree protocols of interaction
- Don't get a 3rd party involved, i.e. the OS
- Keep to user-space
- Beat the "notify()" problem

Observable State Machines

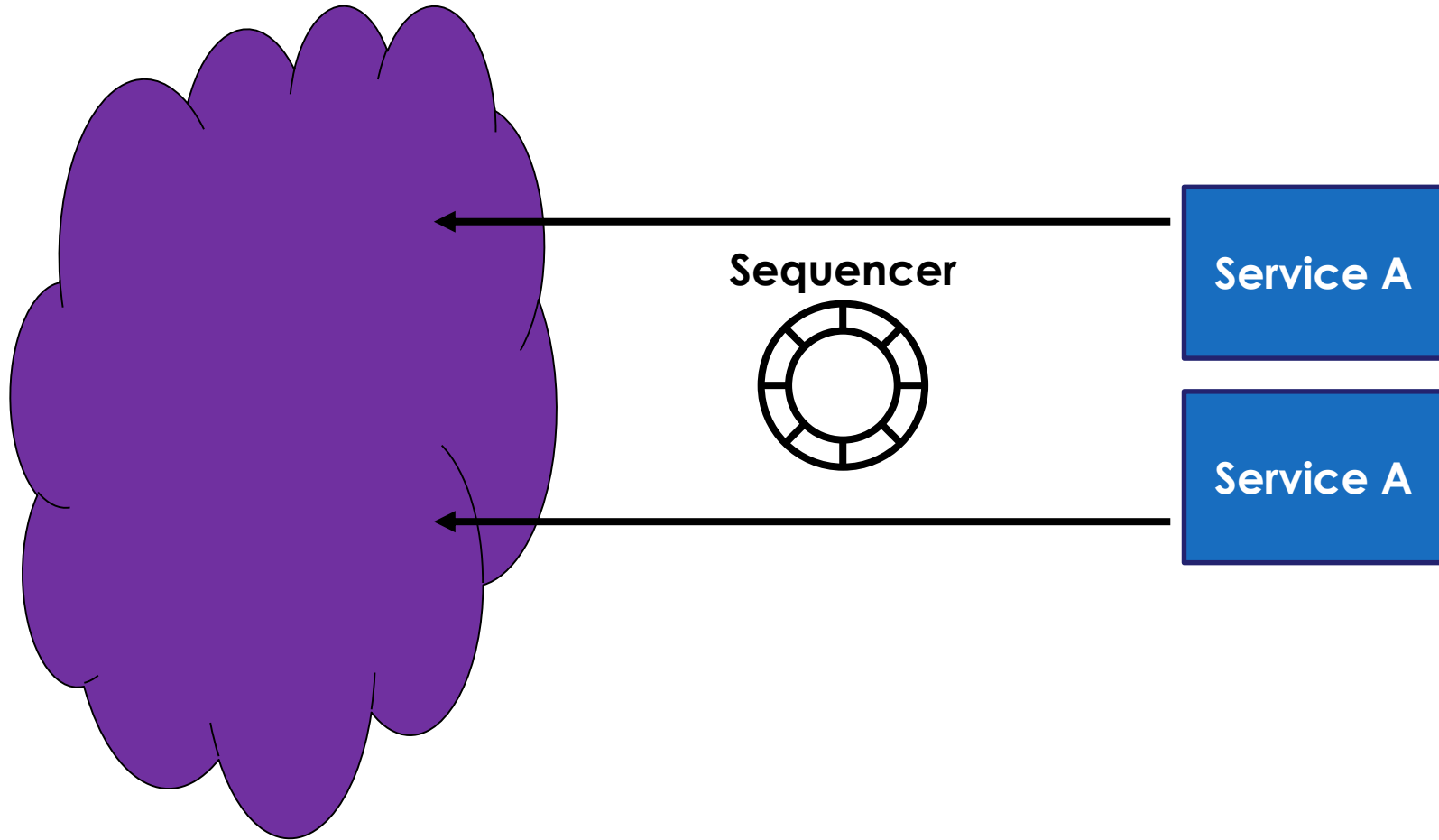


Pro Tip: Observable state machines make monitoring easy

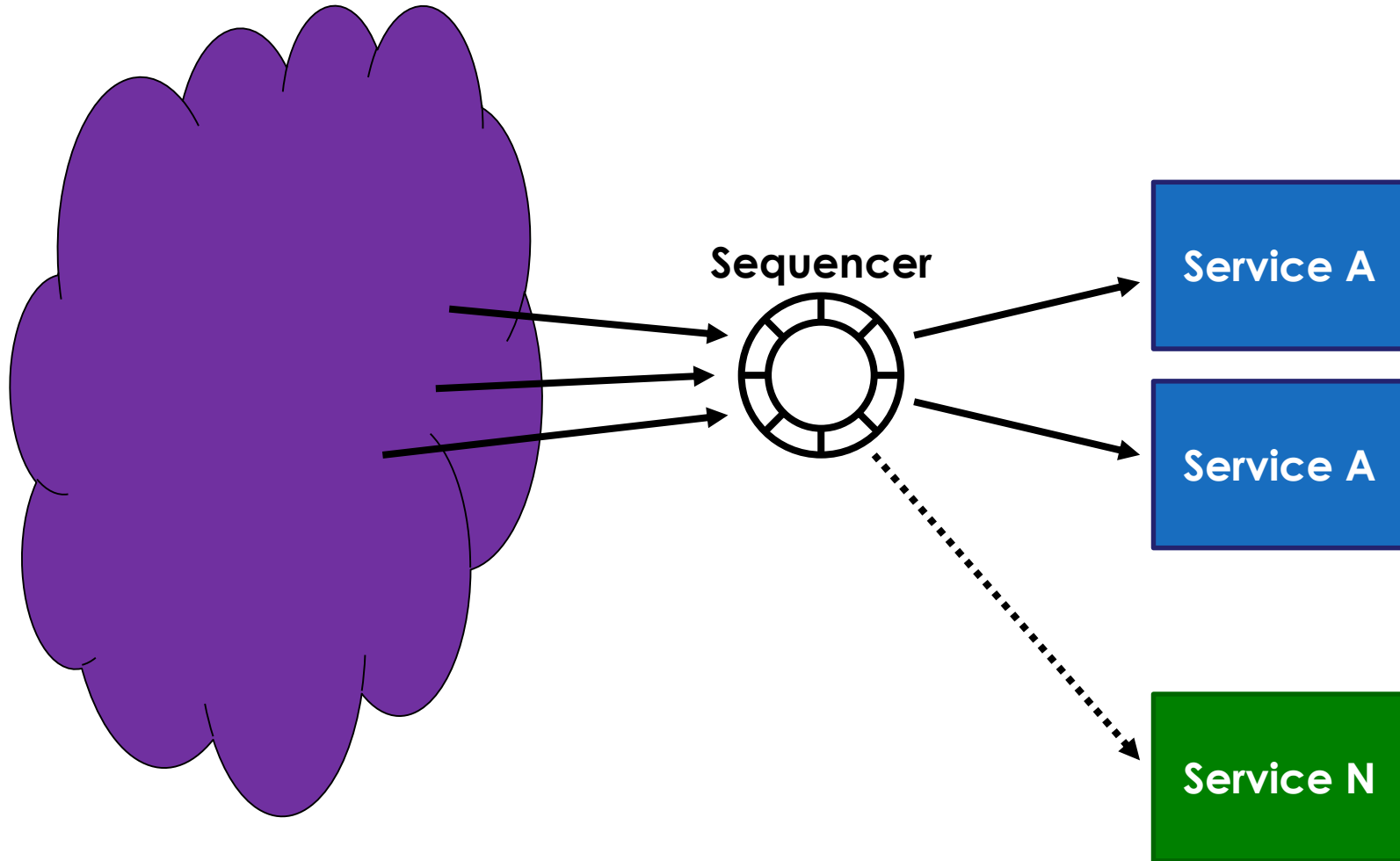
Cluster for Response and Resilience



Cluster for Response and Resilience



Cluster for Response and Resilience



Data Structures and $O(?)$ Models

**Is there a world beyond
maps and lists?**

In closing...





WINTER IS COMING

The Internet of Things (IoT)

“There will be **X** connected devices by 2020...”

Where **X is 20 to 75 Billion**

***If you cannot control
arrival rates...***

***...you have to think hard
about improving service times!***

***...and/or you have to think hard
about removing all contention!***

Questions?

Blog: <http://mechanical-sympathy.blogspot.com/>

Twitter: @mjpt777

“It does not matter how intelligent you are, if you guess and that guess cannot be backed up by experimental evidence – then it is still a guess.”

- Richard Feynman