Chaos Engineering: A Five-Year Retrospective

SREcon21
October 12-14, 2021

Mikolaj Pawlikowski
Software Engineering Lead
@mikopawlikowski

Sachin Kamboj
Software Engineer
@backslash42

Bloomberg Engineering

TechAtBloomberg.com

Table of Contents

TechAtBloomberg.com

Bloomberg

Engineering

Chaos Engineering

*"Chaos Engineering is the discipline of **experimenting** on a system in order to **build confidence** in the system's capability to **withstand turbulent conditions** in production."*

-- Principles of Chaos Engineering
https://principlesofchaos.org

TechAtBloomberg.com

Bloomberg

Engineering

Chaos Engineering

“ ***experimenting***
to ***build confidence***
to ***withstand turbulent conditions*** ”

-- Principles of Chaos Engineering
https://principlesofchaos.org

TechAtBloomberg.com

Bloomberg
Engineering

Chaos Engineering

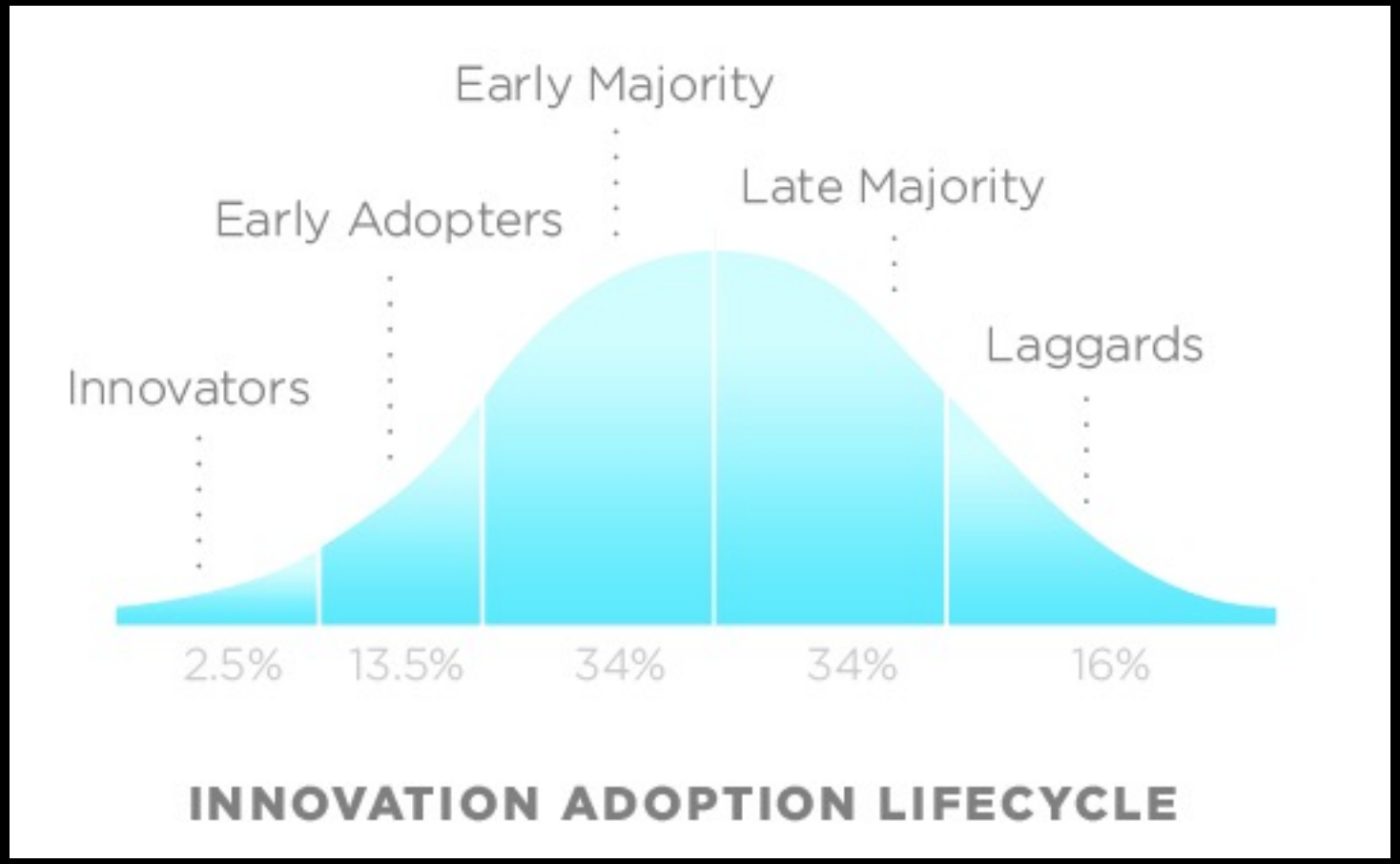
- ~~It's just Chaos Monkey~~
- ~~Testing in prod~~
- ~~Randomly breaking things in production~~

TechAtBloomberg.com

Bloomberg

Engineering

Evolution



TechAtBloomberg.com

Bloomberg

Engineering

https://en.wikipedia.org/wiki/Technology_adoption_life_cycle

Evolution

TechAtBloomberg.com

Bloomberg

Engineering

Evolution

- Motorola DynaTAC 8000X
- 1983-1994
- $3,995 in 1984 ($9,952 equivalent in 2020)
- 30 minutes talk time
- Store 30 phone numbers

TechAtBloomberg.com

Bloomberg

Engineering

https://en.wikipedia.org/wiki/Motorola_DynaTAC

Evolution

TechAtBloomberg.com

Bloomberg

Engineering

Evolution

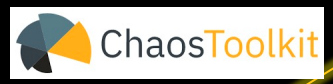
Innovators

TechAtBloomberg.com

Bloomberg

Engineering

Evolution



Early adopters

Innovators

TechAtBloomberg.com

Bloomberg

Engineering

Evolution

Early adopters

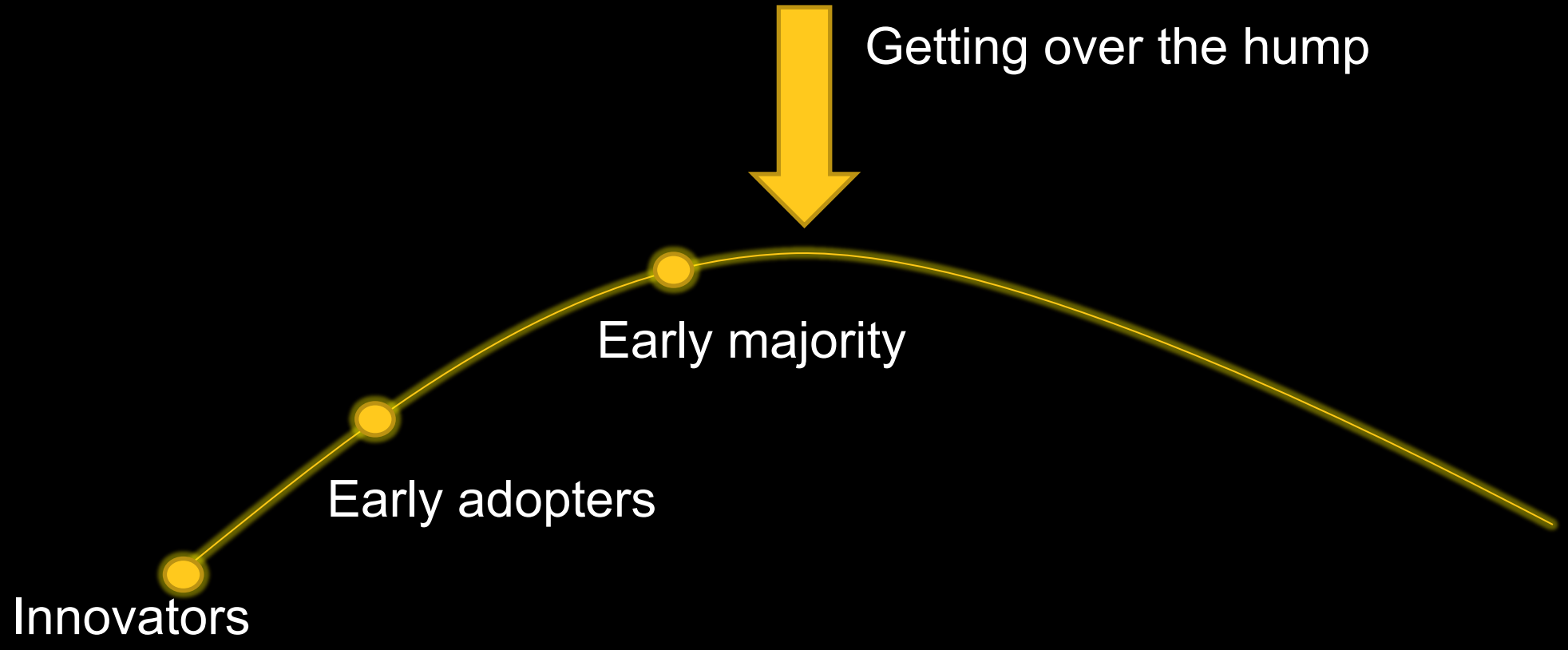
Innovators

TechAtBloomberg.com

Bloomberg

Engineering

Evolution



TechAtBloomberg.com

Bloomberg

Engineering

Evolution

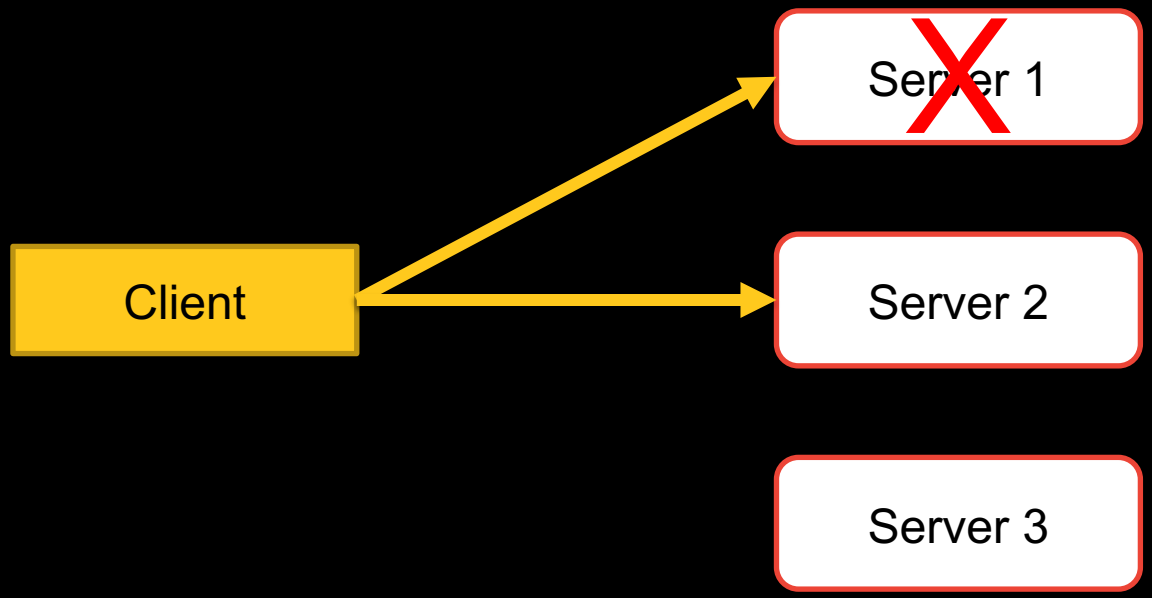
TechAtBloomberg.com

Bloomberg

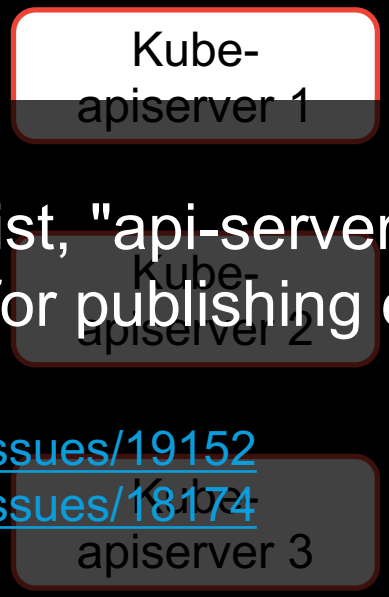
Engineering

Examples from the trenches

Example 1 – Are you really highly available?

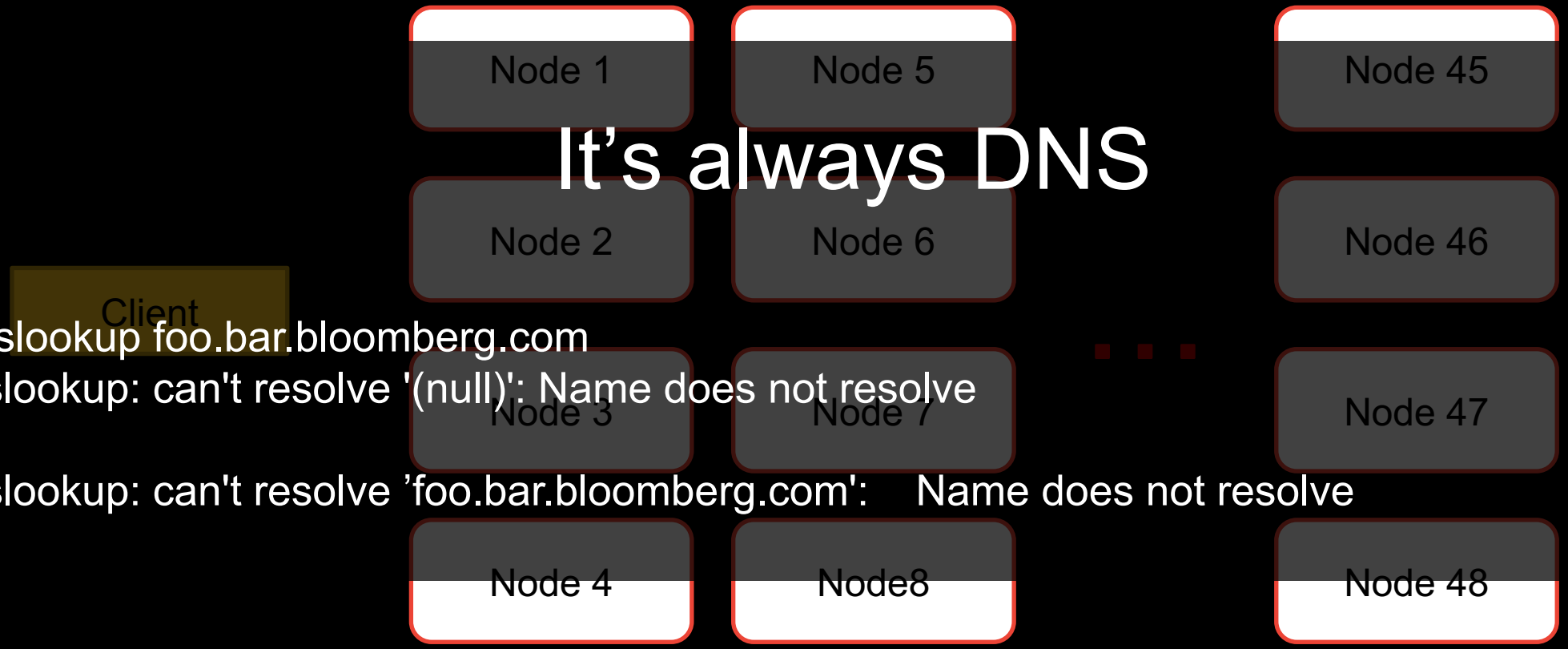


The story of the Kubernetes components



Flag for kubelet:
fs.StringSliceVar(&s.APIServerList, "api-servers", []string{},
"List of Kubernetes API servers for publishing events, and reading pods and services.
(ip:port), comma separated.")
https://github.com/kubernetes/kubernetes/issues/19152
https://github.com/kubernetes/kubernetes/issues/18174

Example 2 – What happens when you scale?



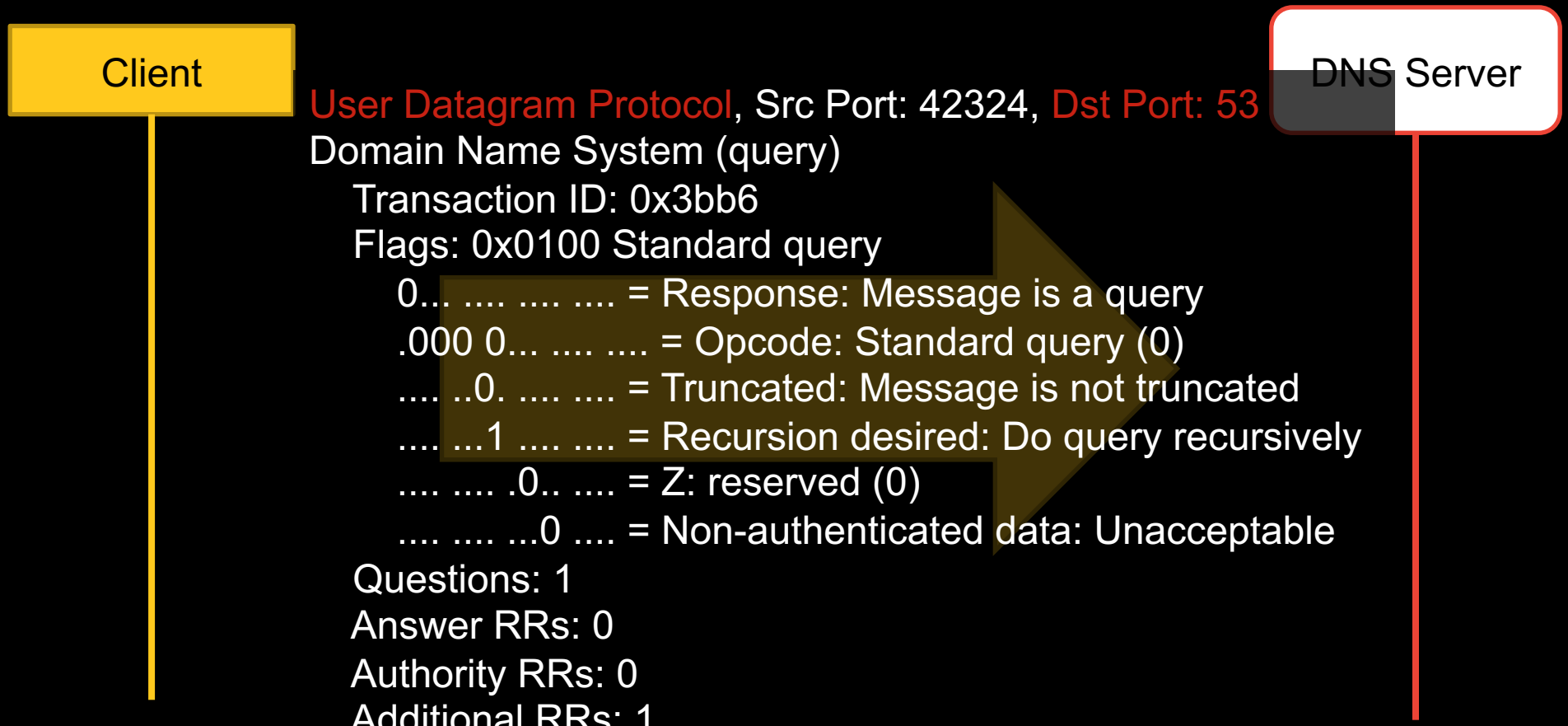

```
$ nslookup foo.bar.bloomberg.com
  nslookup: can't resolve '(null)': Name does not resolve

  nslookup: can't resolve ’foo.bar.bloomberg.com':    Name does not resolve
```


Bloomberg

Engineering

Example 2 – What happens when you scale?

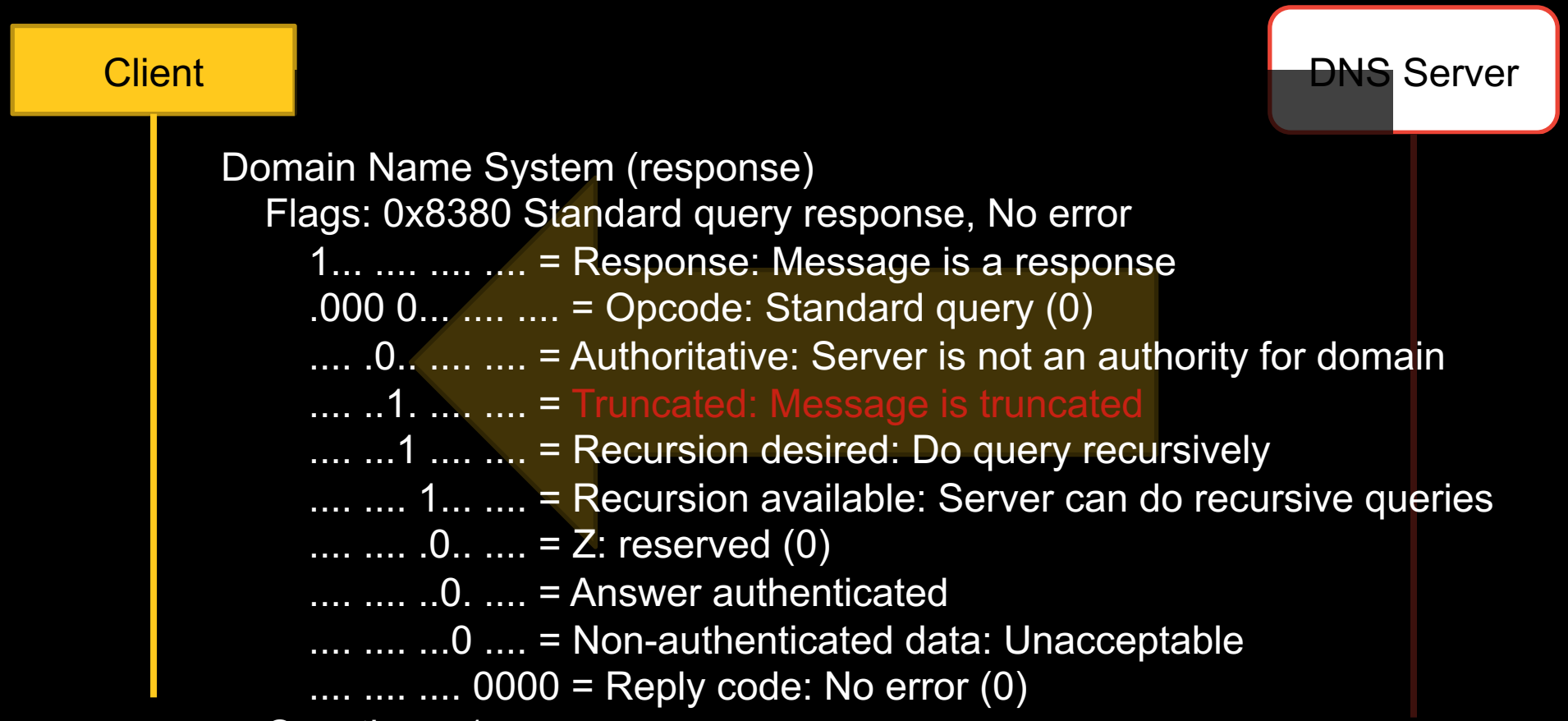


TechAtBloomberg.com

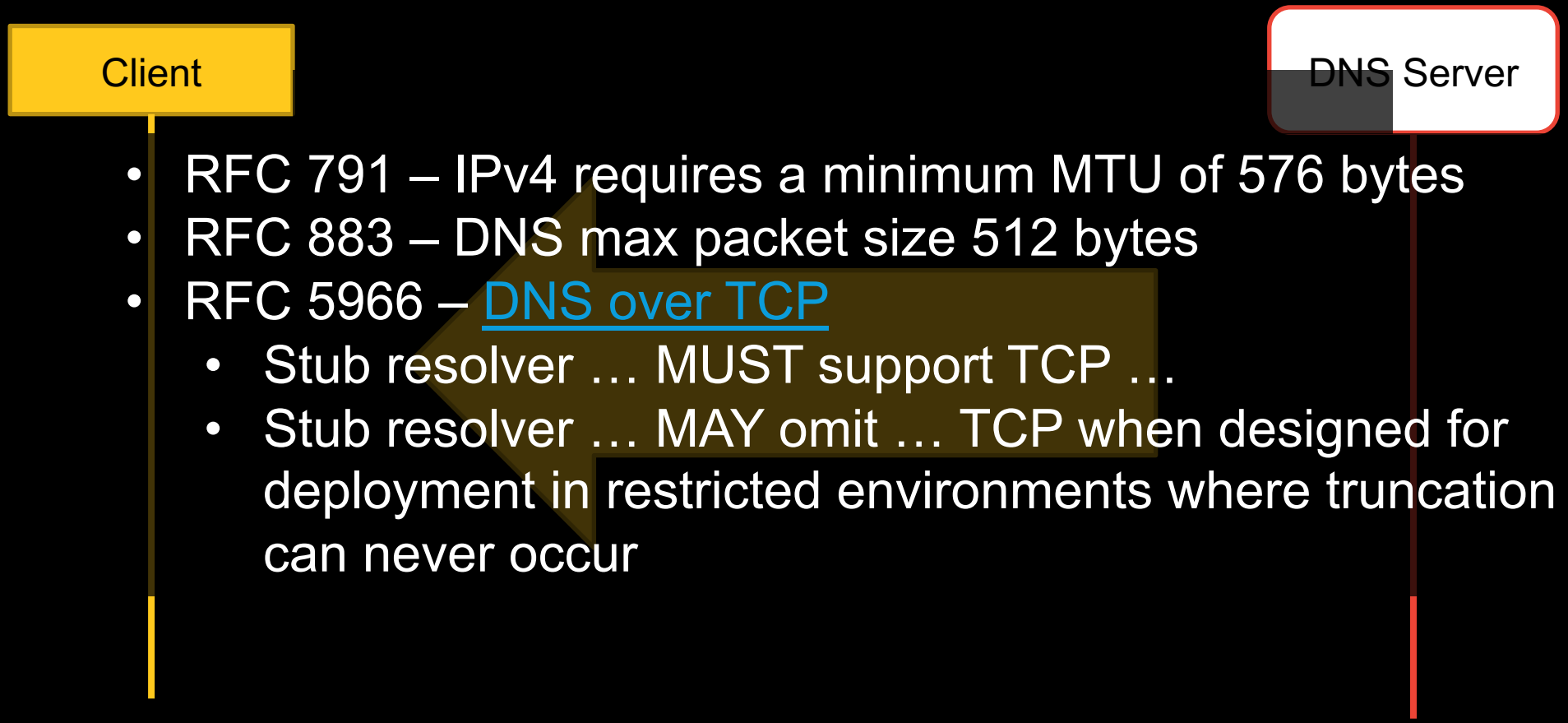
Bloomberg

Engineering

Example 2 – What happens when you scale



Example 2 – What happens when you scale



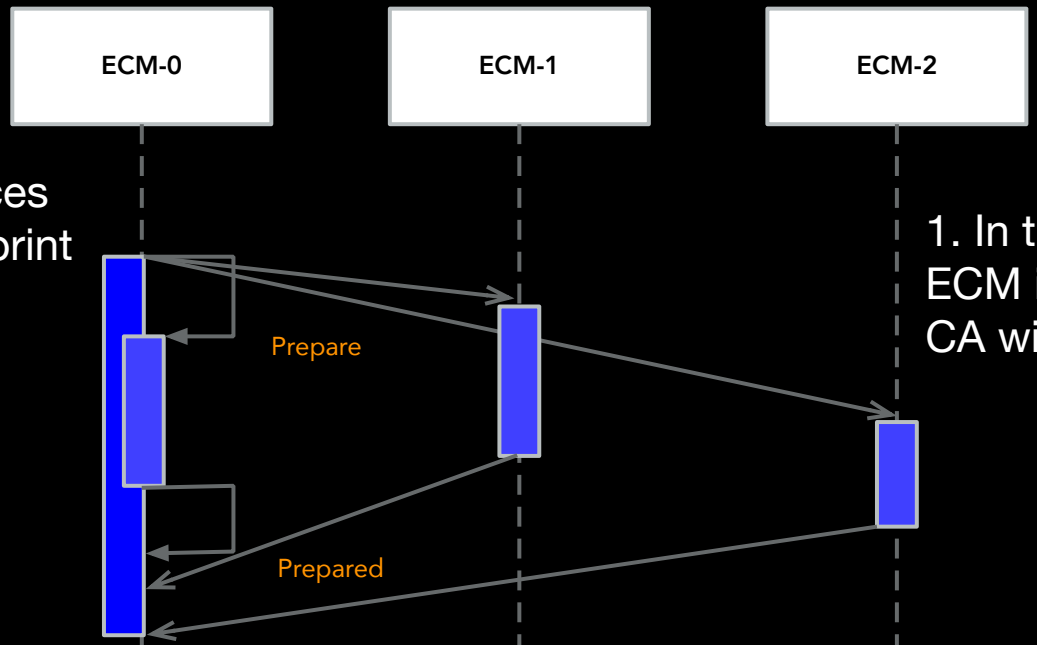
- RFC 791 – IPv4 requires a minimum MTU of 576 bytes
- RFC 883 – DNS max packet size 512 bytes
- RFC 5966 – DNS over TCP
  - Stub resolver … MUST support TCP …
  - Stub resolver … MAY omit … TCP when designed for deployment in restricted environments where truncation can never occur

Example 3 – Can you reproduce bugs?

- https://github.com/kubernetes/kubernetes/issues/87615

- Bug reported on January 28, 2020
- Lots of reports, but not reliably reproducible for a whole year
- Or 1 week of chaos engineering

Example 4 – Proactively find bugs



0. Validate that all instances have the same CA fingerprint and a recent heartbeat

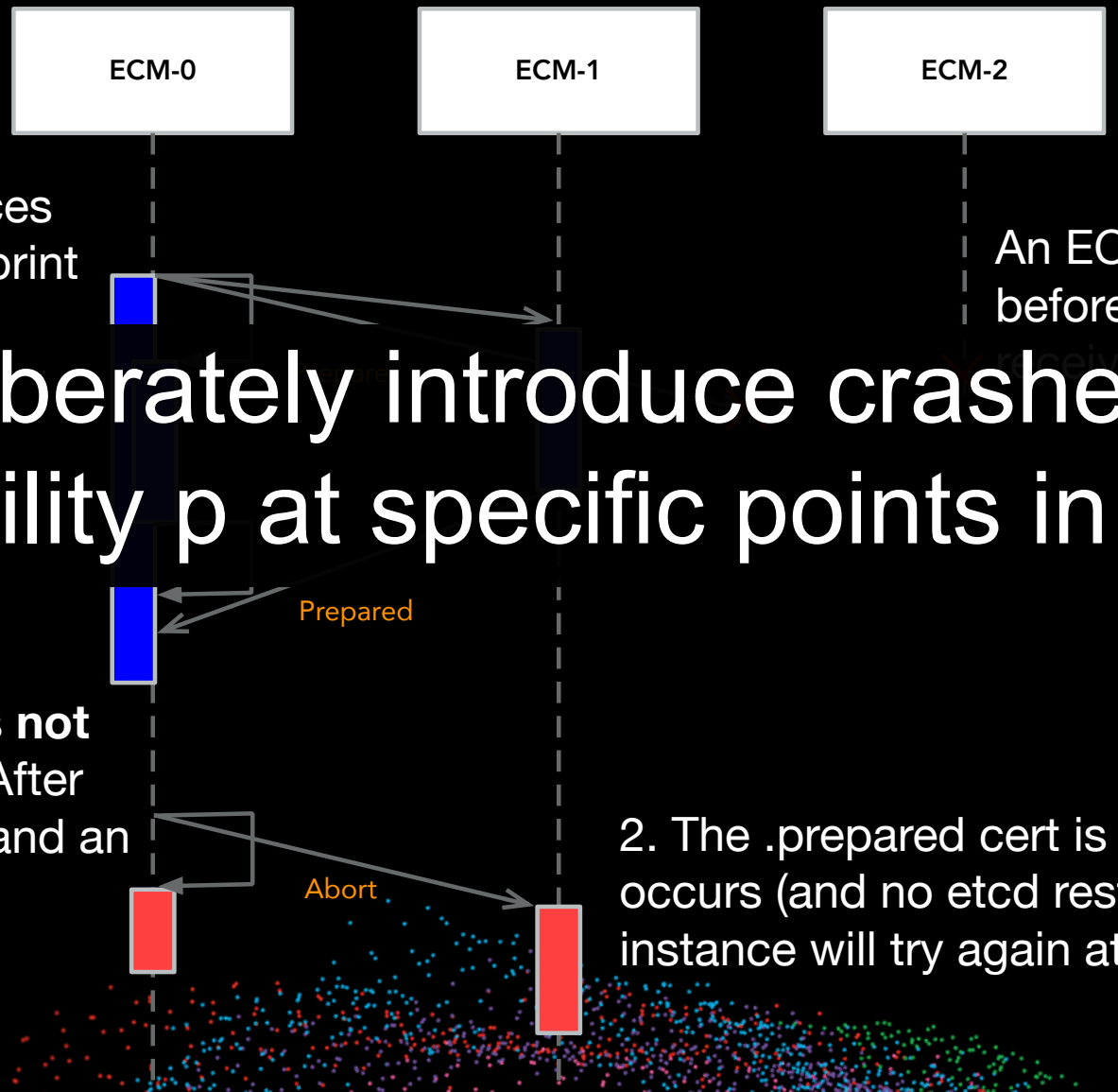
1. In the prepare step, each ECM instance writes the new CA with a .prepared prefix

2. On receiving a prepared message from all nodes, send a commit message

3. In the commit step, each ECM instance removes the .prepared suffix and restarts etcd

The committed message can be omitted, the heartbeat will take care of things

Example 4 – Proactively find bugs



0. Validate that all instances have the same CA fingerprint and a recent heartbeat

An ECM instance crashes before the prepare message is received or the message is lost

Deliberately introduce crashes with probability p at specific points in the code

1. A prepared message is **not** received from all nodes. After timeout the lock is freed and an abort message is sent

2. The .prepared cert is deleted. No rotation occurs (and no etcd restarts) and another instance will try again at the next opportunity.

Classes of failures

- Node failures
- Network Partitions
- Dropped packets
- Limits
- Blast Radius
- Increased Latency
- Noisy neighbors
- Thundering herds
- Bad configurations

Lessons learned

- Boring is good
- Experiments tend to be cheap
- It’s just code
- A mindset shift
- “Good” <= ROI <= “Awesome”
- Retrofitting is expensive

TechAtBloomberg.com

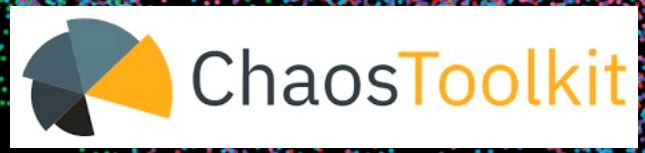
Bloomberg

Engineering

Getting started in 2021

- **Tools readily available**
  - Open Source & also commercial
- **Books and training material also available**
  - I should know, I wrote one of these ☺
- **Cost of entry is low**

Tools



https://github.com/powerfulseal/powerfulseal
https://chaostoolkit.org
https://github.com/alexei-led/pumba
https://github.com/Shopify/toxiproxy
https://github.com/Netflix/chaosmonkey
https://byteman.jboss.org/
https://github.com/storax/kubedoom
https://litmuschaos.io/

More: https://github.com/dastergon/awesome-chaos-engineering

TechAtBloomberg.com

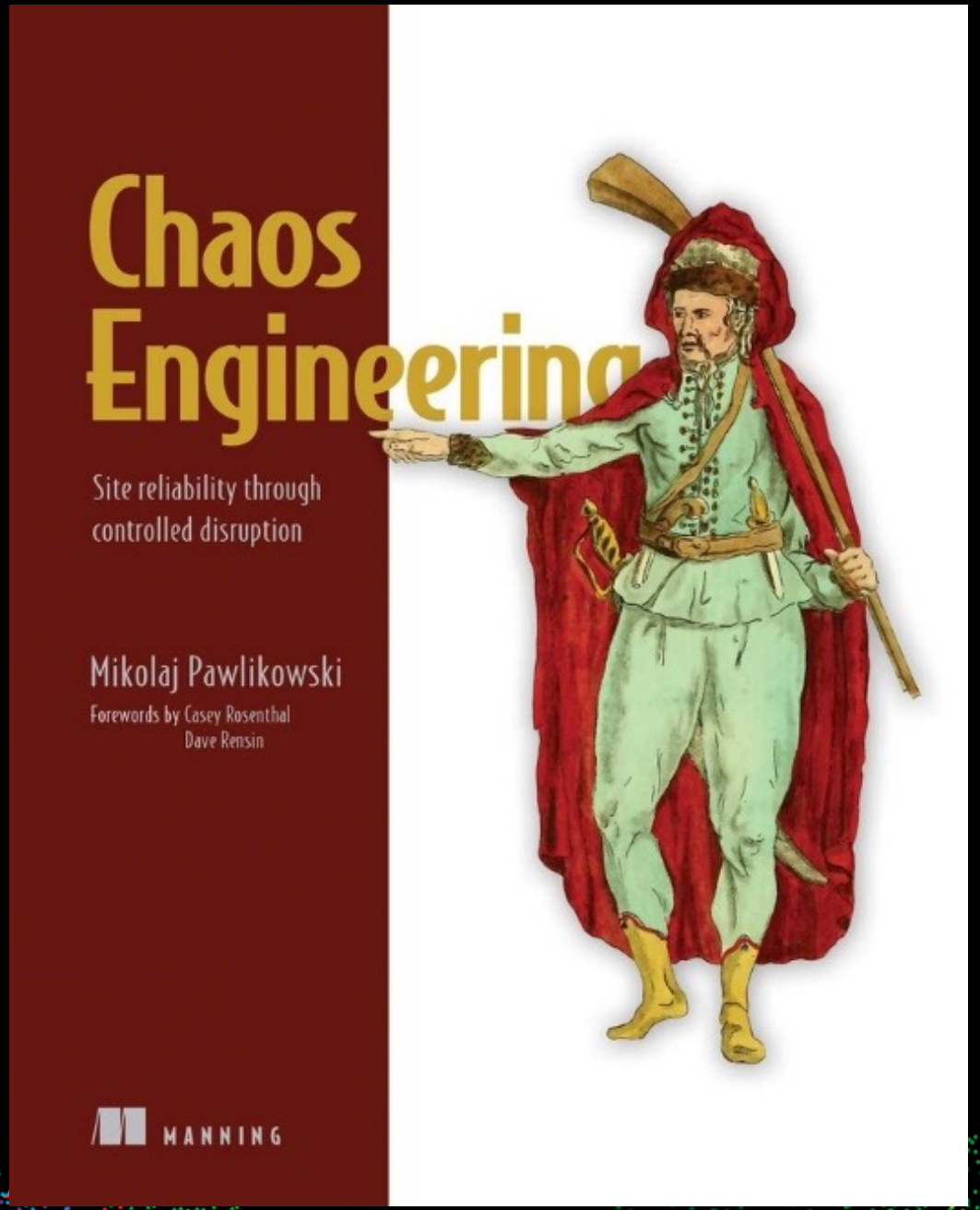
Bloomberg

Engineering

Chaos Engineering: Site Reliability Through Controlled Disruption

Manning

https://www.manning.com/books/chaos-engineering



TechAtBloomberg.com

Let’s connect!

Mikolaj Pawlikowski

Author "Chaos Engineering: Crash test your applications" | Engineering Lead at B...

Sachin Kamboj

Bloomberg Engineering - Software Infrastructure

TechAtBloomberg.com

Photo Credits

Photos from unsplash.com:

- Eirik Solheim - https://unsplash.com/@eirikso (old phones)
- Meritt Thomas - https://unsplash.com/@merittthomas (unicorn)

TechAtBloomberg.com

Bloomberg

Engineering

Thank you!

https://www.bloomberg.com/careers

Bloomberg

Engineering

TechAtBloomberg.com