Yao Yue

Coping with
failures
in datacenters

# Lies,
# Damned lies,
# and timeouts

**Ellen DeGeneres**
@TheEllenShow

Follow

If only Bradley's arm was longer. Best photo ever. #oscars

7:06 PM - 2 Mar 2014

**3,419,496** Retweets **2,403,744** Likes

223K 3.4M 2.4M

# Who am I?

Cache @ Twitter

Now working on performance in general

@thinkingfish

## Why do I want to give this talk?

# I've been telling my coworkers the same things for years...

# What do I know?

- My views are heavily influenced by two things:
  - In-memory, datacenter-scale caching
  - Twitter's environment

# Cache in datacenters

- Distributed in-memory KV store
- Serving many, many requests
- with very tight latency expectation

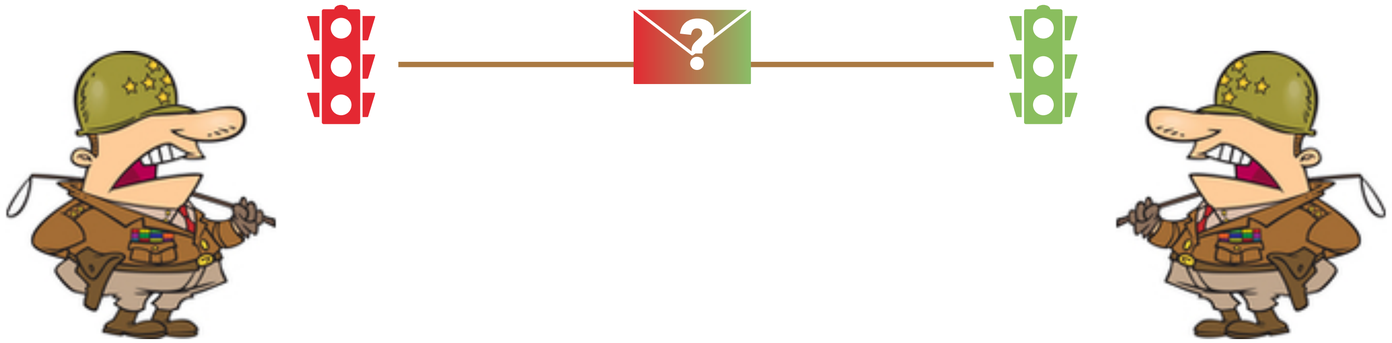
# Twitter's runtime

- Architecture: monolithic -> SOA/Microservices

- Owns datacenters
- Services mostly on JVMs
- Jobs deployed with containers
- Scale: up to many thousands of nodes per service

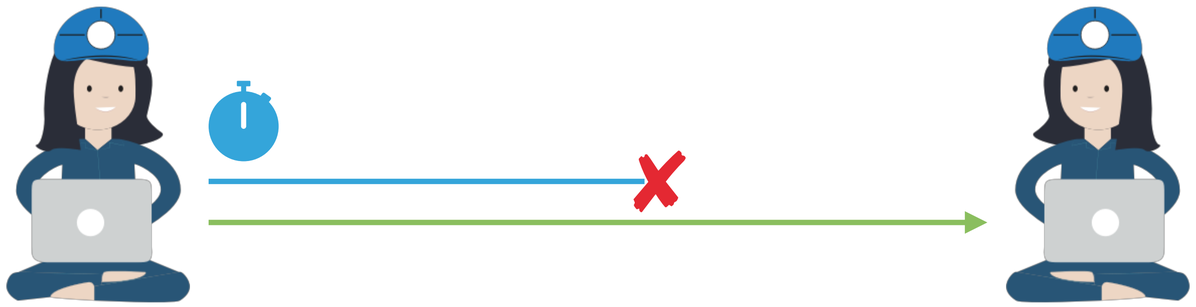
# Living in an imperfect world

# Two unhappy generals

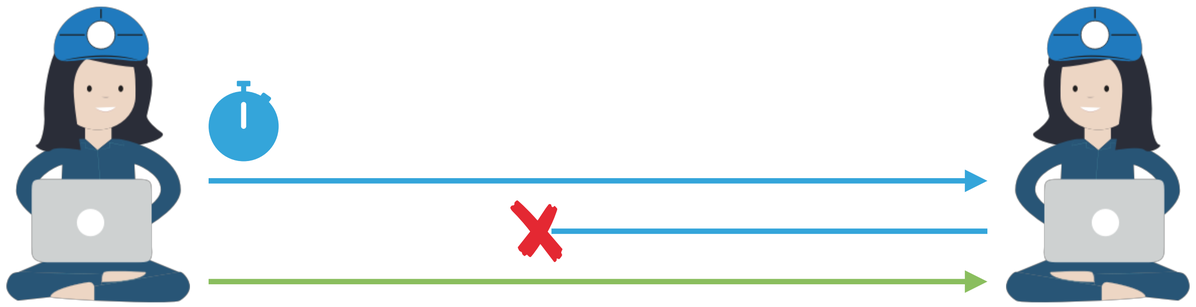



Two General Problem / Paradox

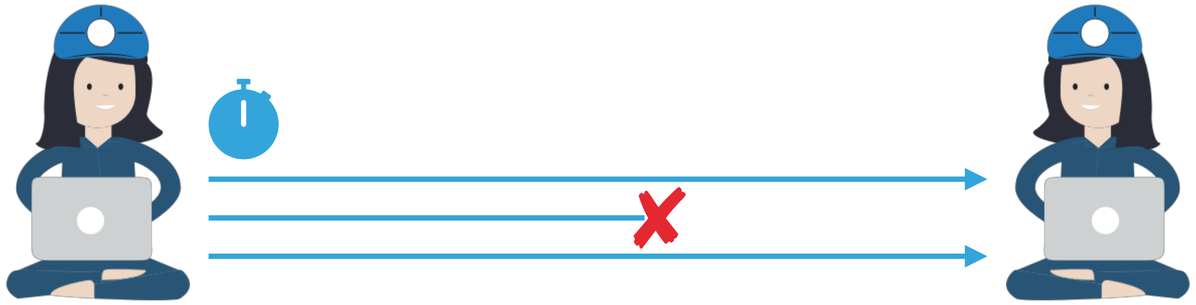
# The engineering approach to cope with failures



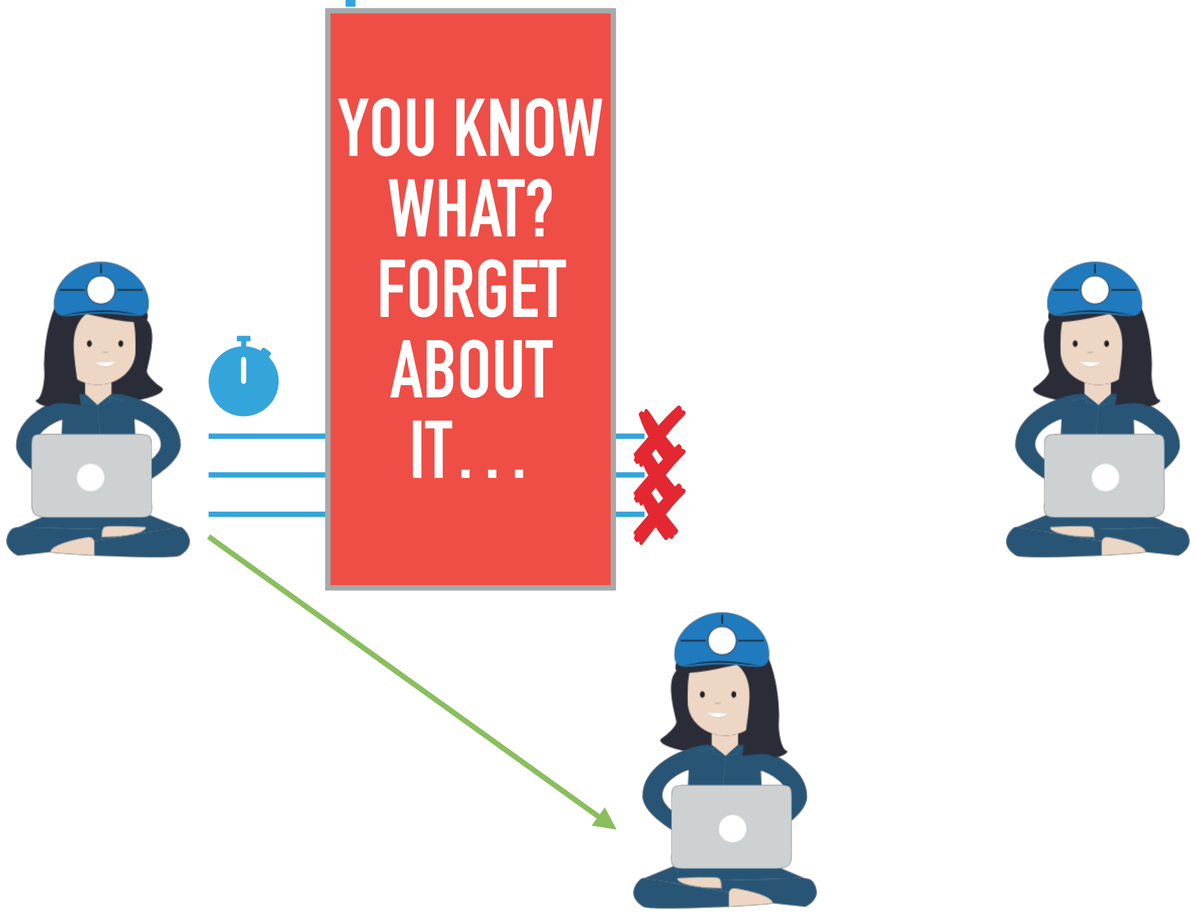
# The engineering approach to cope with failures



# The engineering approach to cope with failures



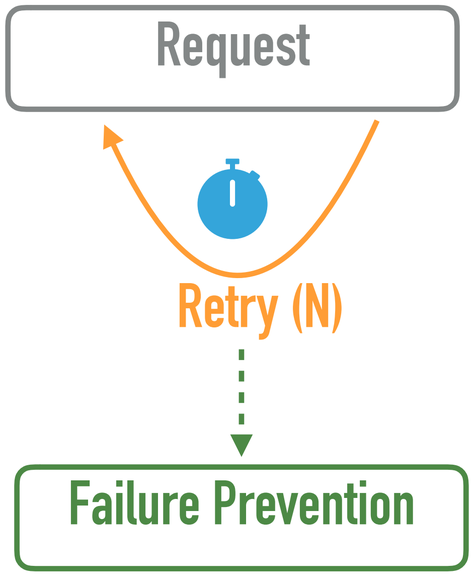
# The engineering approach to cope with failures



Timeouts, Retries, and preventions

# Coping with failure

# Coping with failure



# Timeout and retry

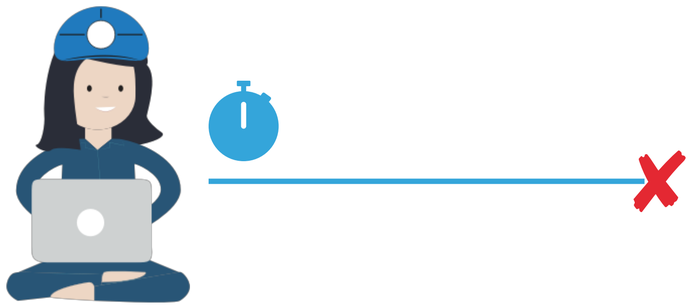
- Timeout: an *approximation* of failure
  - False positive is possible
- Retry: an *effort* to reduce failure
  - Has a cost
  - Has an effect on system

Timeout and retry get close to the heart of the difficulty of distributed systems, and yet we treat them casually because they're often presented as configuration.

# What works intuitively may lead to catastrophes.

# Timeout

# Timeout could be misleading

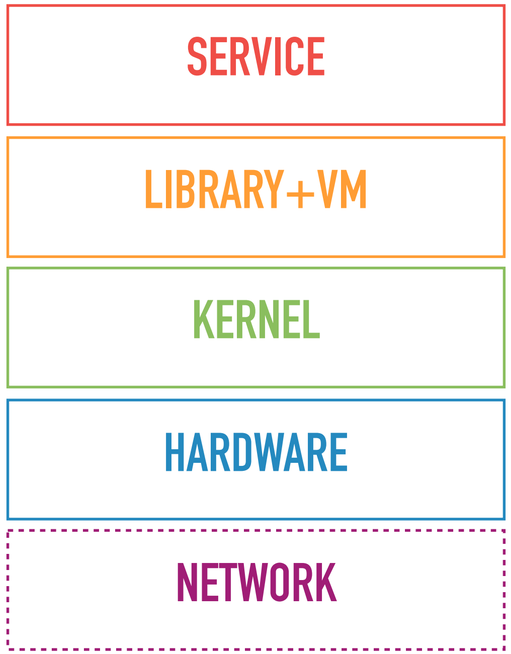


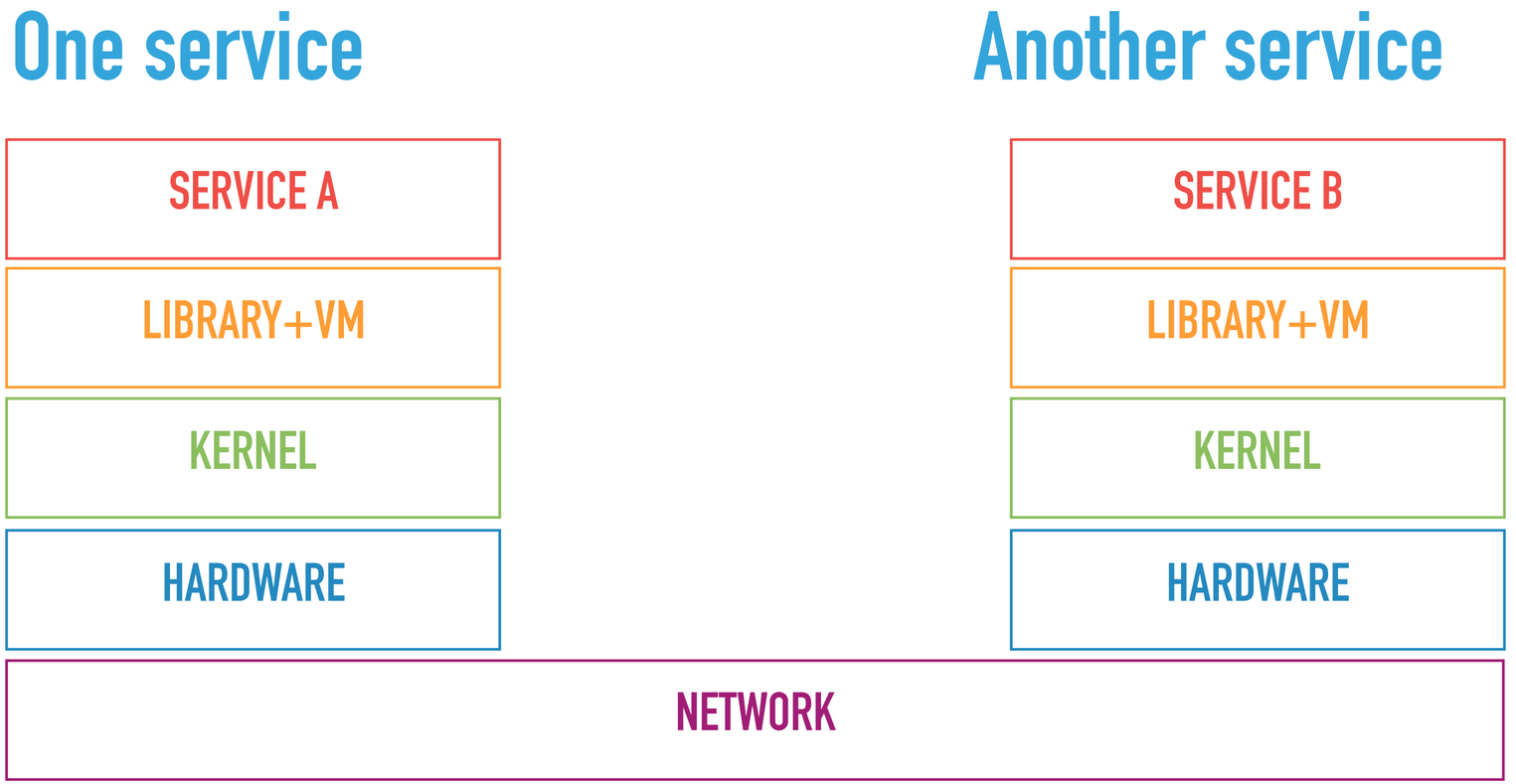
SERVICE A

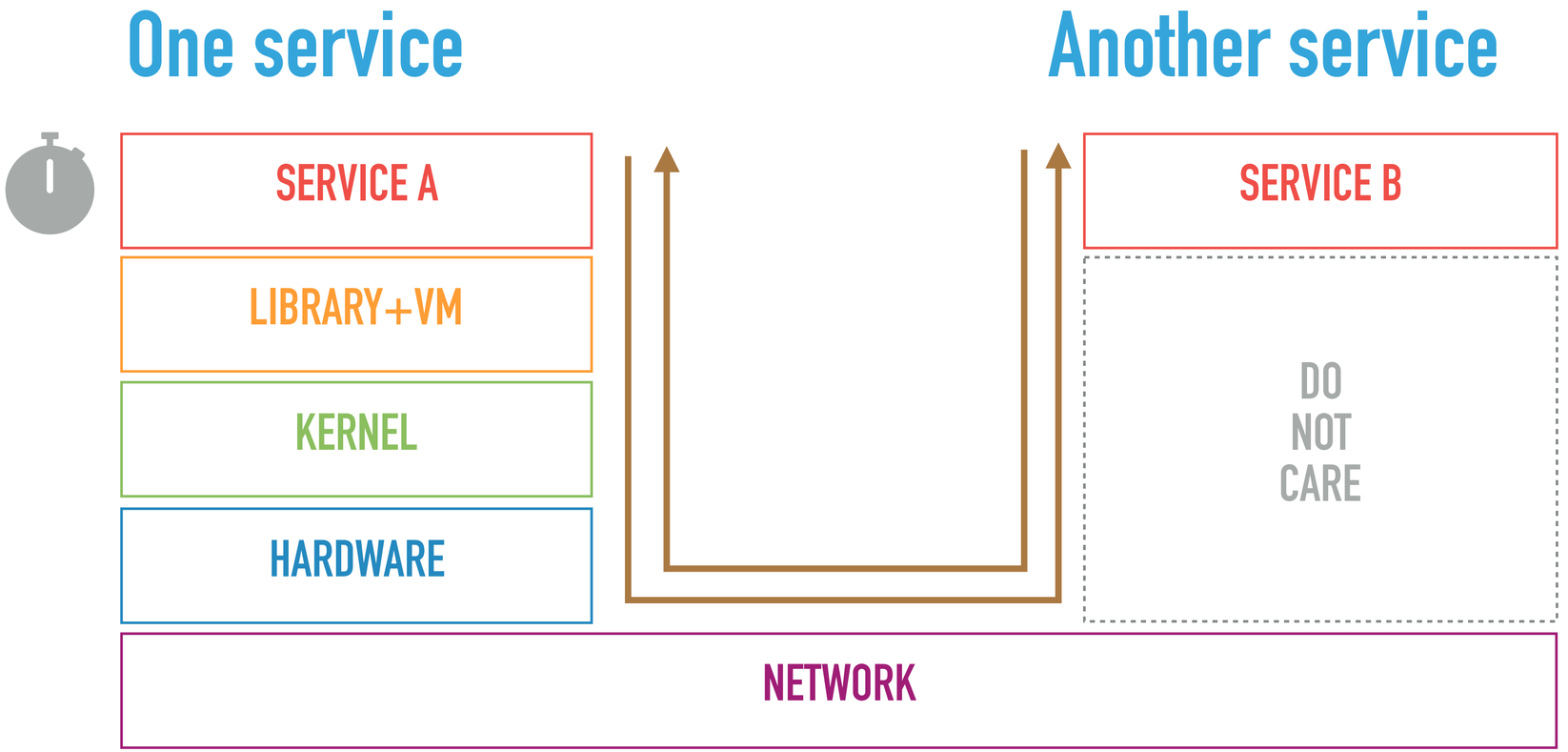
SERVICE B


TIMEOUT INFORMS ME ABOUT COMMUNICATION TO REMOTE SERVICE

# One service



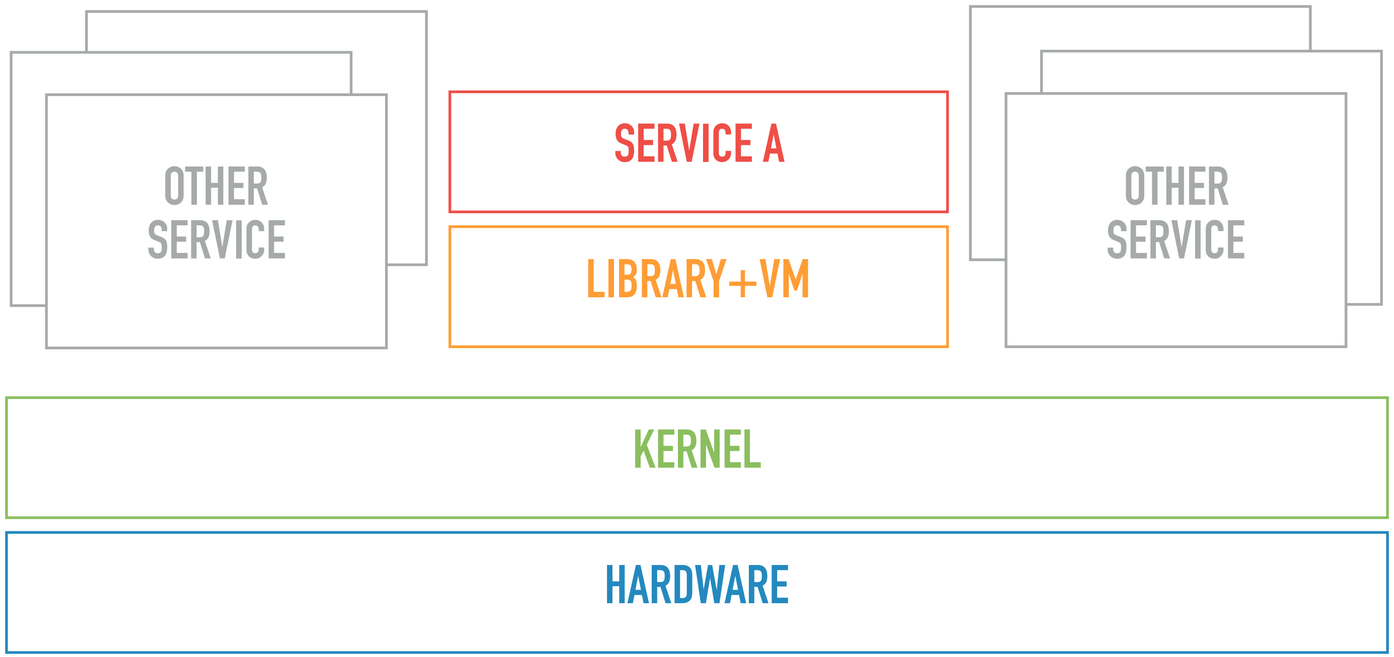

One service
SERVICE A
LIBRARY+VM
KERNEL
HARDWARE
Another service
SERVICE B
LIBRARY+VM
KERNEL
HARDWARE
NETWORK


One service
Another service
SERVICE A
LIBRARY+VM
KERNEL
HARDWARE
SERVICE B
DO
NOT
CARE
NETWORK

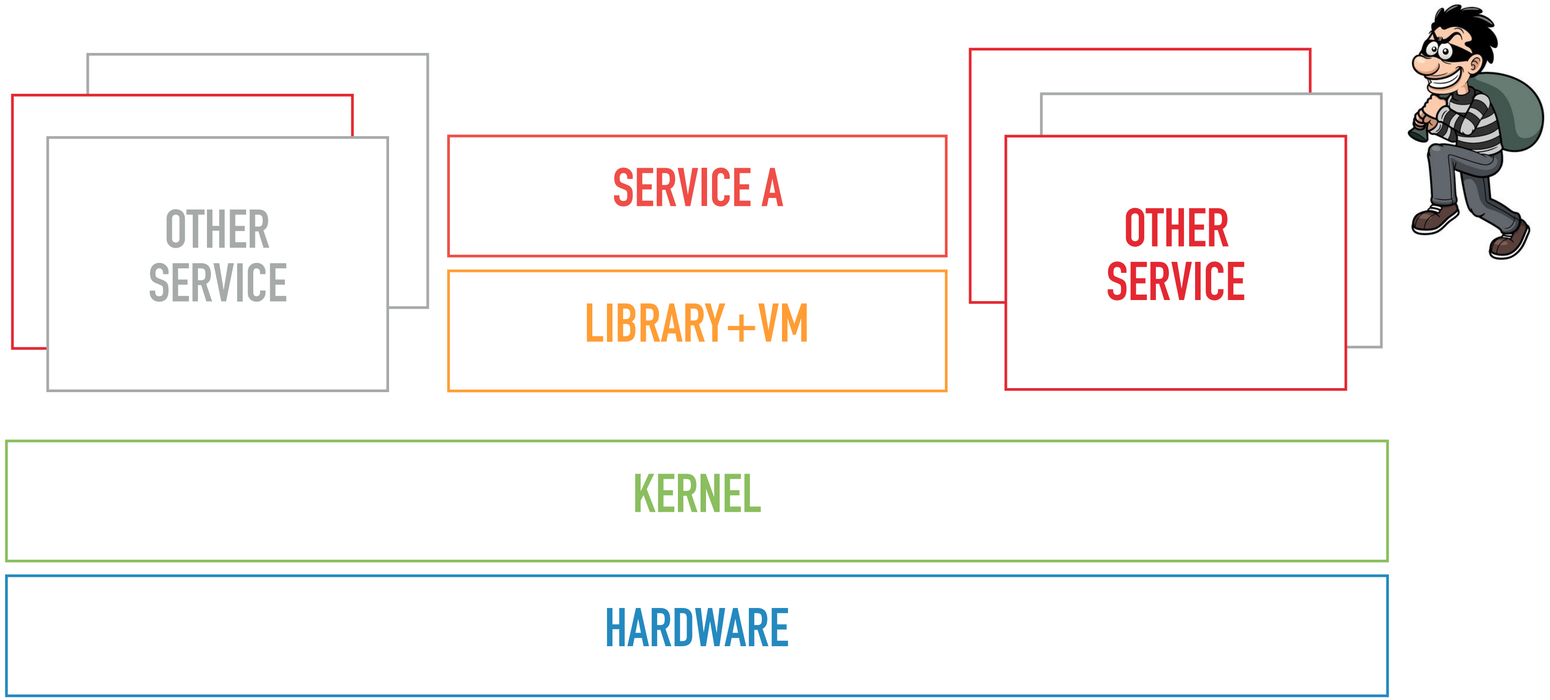
# One service

| | |
|---|---|
| SERVICE | resource contention, head-of-line blocking, locking… |
| LIBRARY+VM | garbage collection, function calls with indeterministic timing… |
| KERNEL | system calls with indeterministic timing, background tasks, unfair scheduling… |
| HARDWARE | blocking IO, congestion, cycle stealing… |
| NETWORK | packet drop, queue backup… |

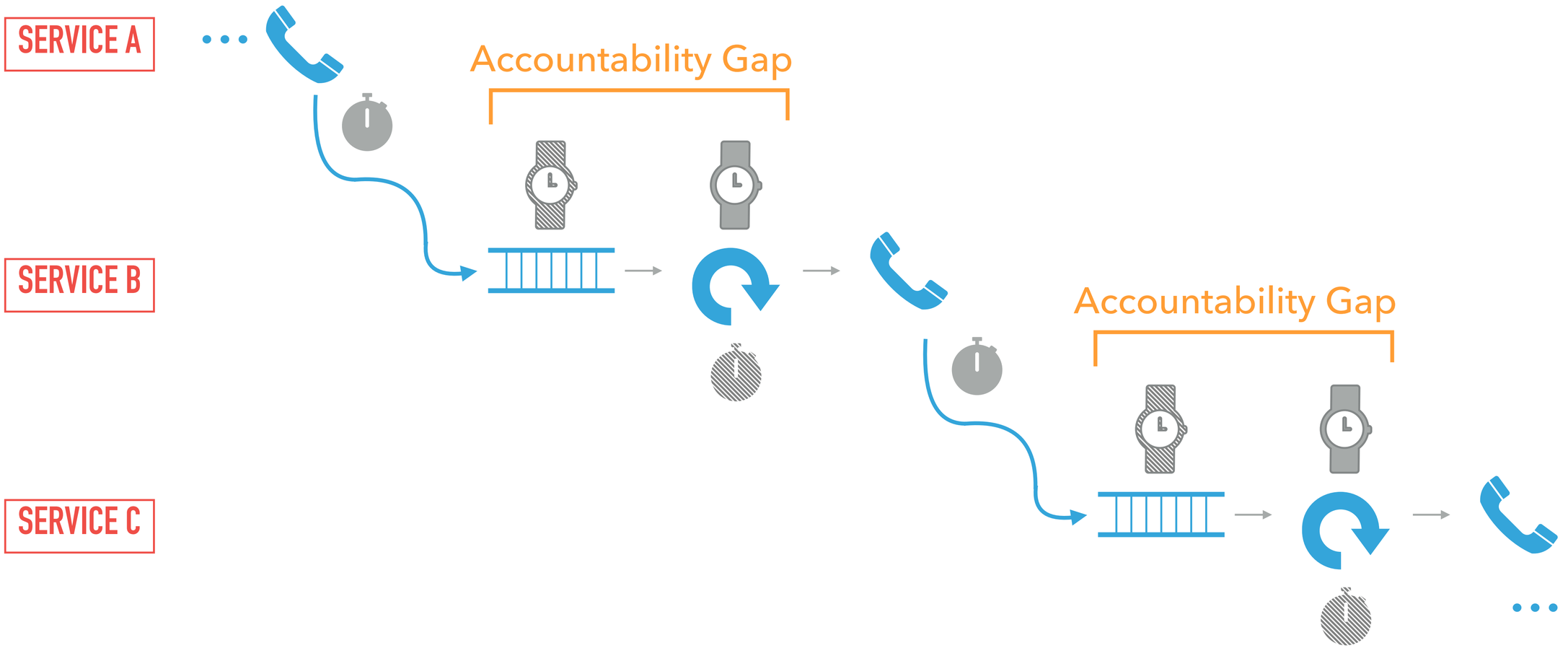
# Multitenant runtime



# Multi-tenancy with Noisy Neighbors



# Timeout cascade



# Inconvenient truths about timeout

- Timeouts often do not indicate remote service health
- The optimal timeout is a moving target
- Often less predictable in shared environment
- Have gaps in overall timeline

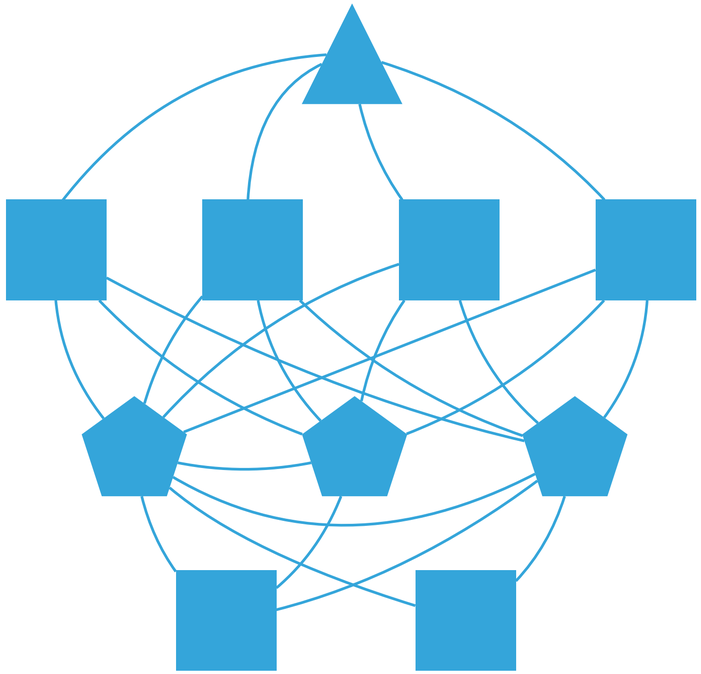
# Chained dependencies

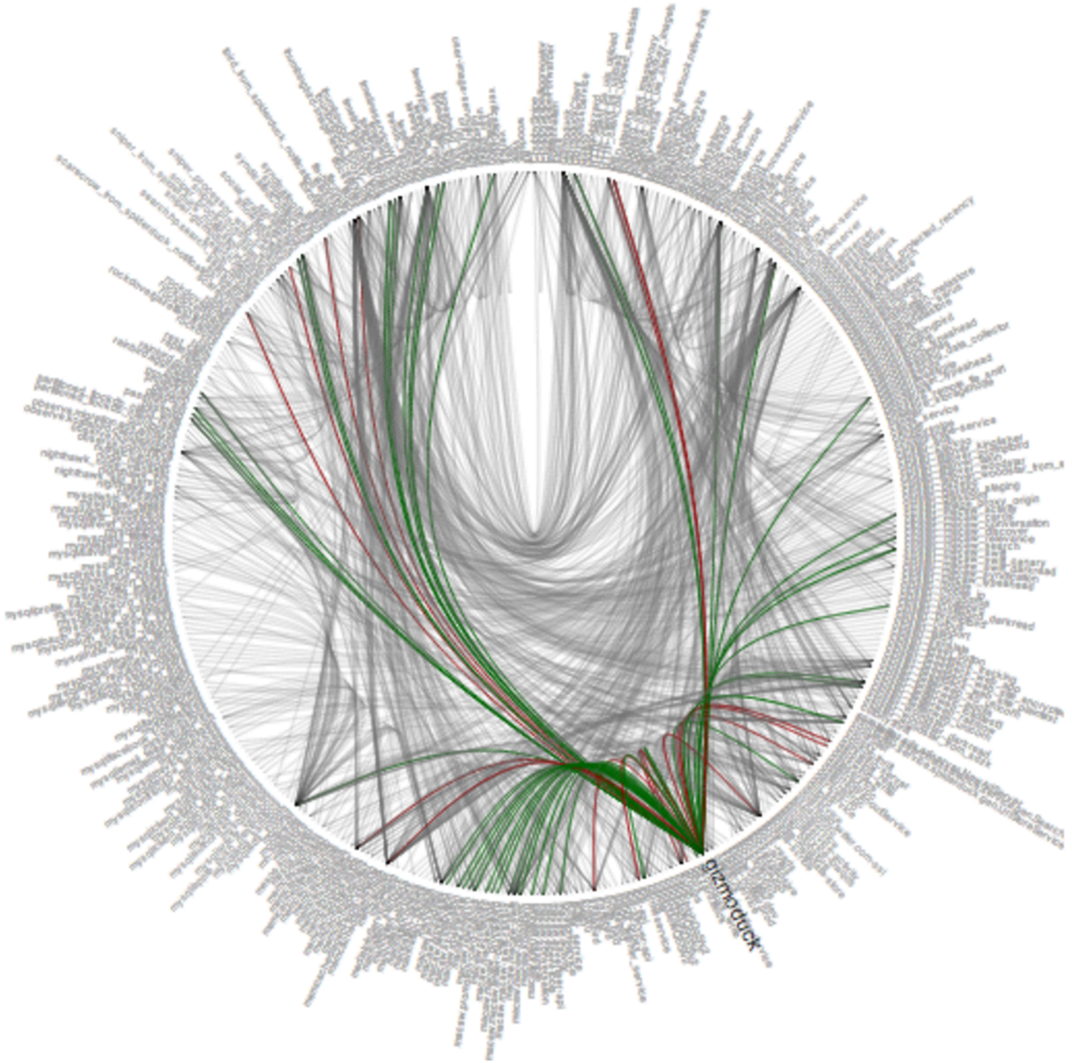

Q: A's requests time out, but not B's, which on-call engineer(s) should be paged?

# Chained dependencies


Q: A's requests time out, *so do B's*, which on-call engineer(s) should be paged?

# Oftentimes modern application architecture is a ~~mesh~~ Mess



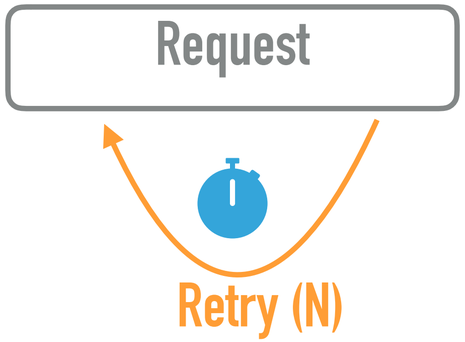


# correlation != causality

# timeouts != causality

# Retry

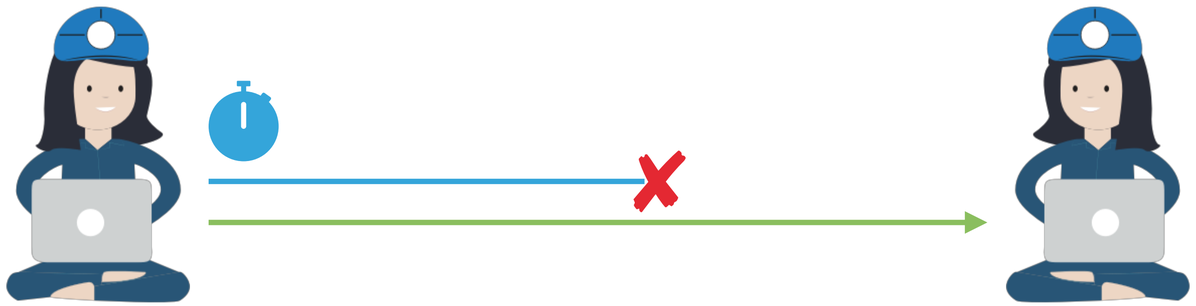
# Retries carry risks





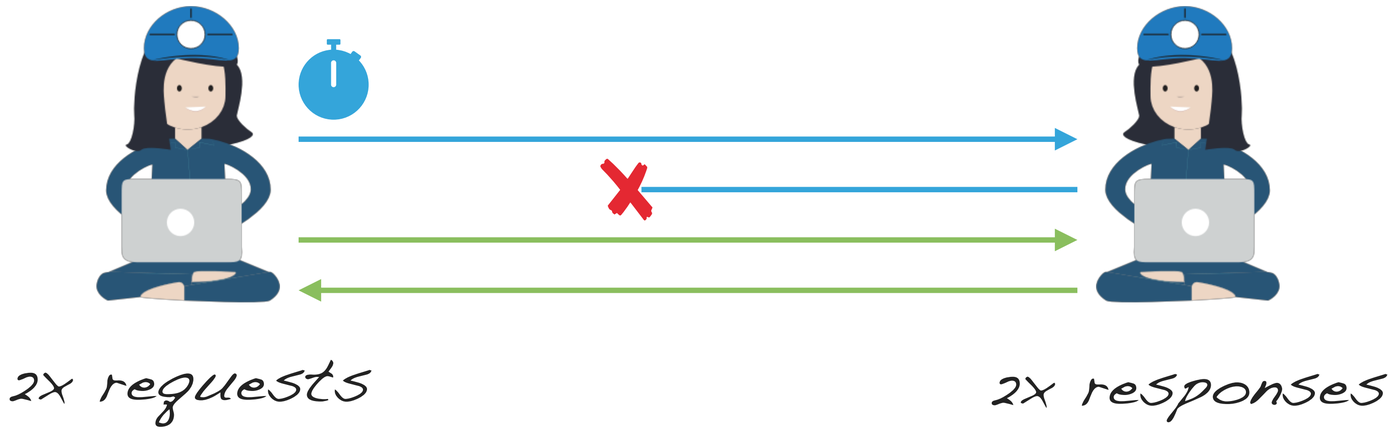
RETRIES IMPROVE THE SUCCESS RATE OF SERVICE

# Retry is extra work



2x requests

# Retry is extra work



# Retry != Replay

- Potential behavioral change
- State change
- Hidden retries in lower stack
  - e.g. TCP

# Other retry key decisions

- How many times?
- How to set timeout for each retry?
- How much delay between retries?
- Where to send it?

# Inconvenient truths about retry

- Retries can create positive feedback loop
- Retries can change system state and behavior
- Sophisticated retry configuration has many knobs

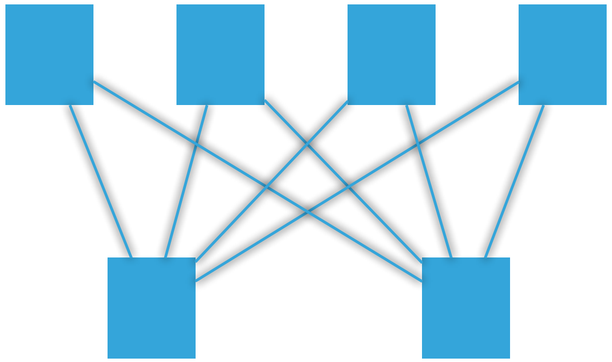
# Chained dependencies


If C is slow…

# Chained dependencies

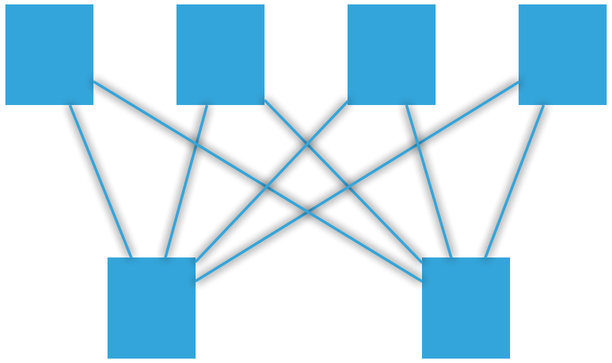

If B is slow…

# Retry at scale– “How to DDoS A cache?”




- Timeout ⇒ connection teardown
- Large number of clients
- Stateful backend, fixed route
- Full connectivity mesh
- Variable connections per route
- Container-based deploy

# Retry at scale– “How to DDoS A cache?”




Transient hotspot

⇒ initial timeouts (closing connections)

⇒ connection storm (fixed route)

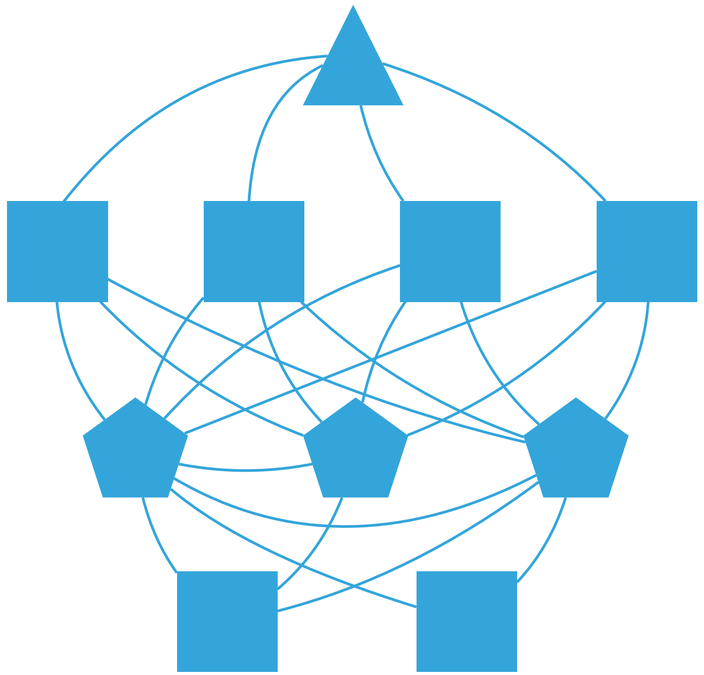
⇒ massive timeouts (full mesh)

⇒ bigger storm (multiple, variable connections)

⇒ NIC saturation / backend offline

⇒ frontend failing, site down

# Service mesh can make retries significantly worse



- Top-level retries affect whole system
- Bigger multipliers toward the bottom
- Hard to predict bottleneck

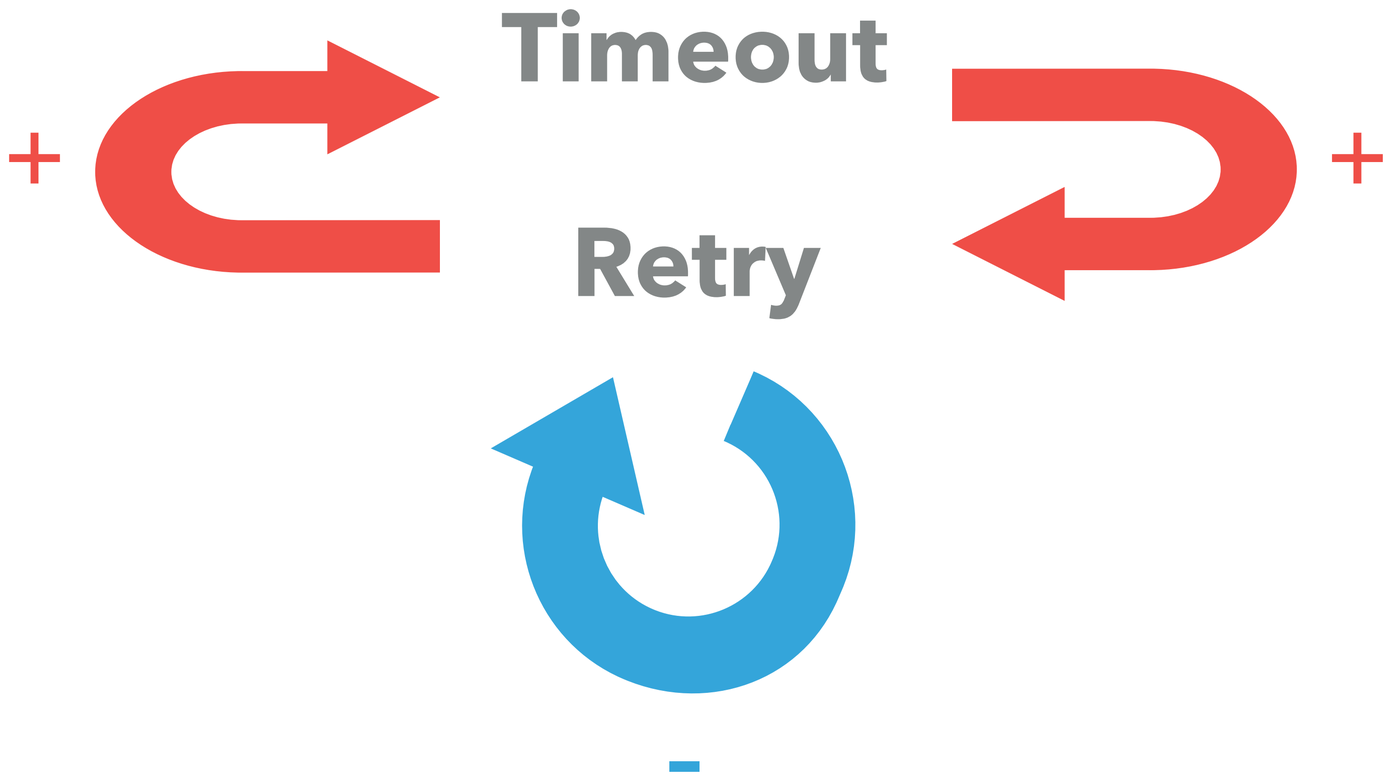
# Wrong implicit assumption, single input, single feedback loop

# How to improve?

# Address the positive feedback loop

# Slow down, retry less

# Introduce other feedback loop



# Good timeouts

- Minimize false positives caused by local stack
- Attempt to root cause with other signals

# Good retries

- Customize for purpose of request
- Be as conservative as possible

- Circuit breakers for timeout induced retries
  - State-based rules
  - Back-pressure

# Manage impact across services

- Enforce top-down budget
- Apply and act on back pressure

Using micro services? Your task just got much harder…

- Tracing helps

# drill, baby, drill

- Test common failure scenarios
- Test failures at different parts
- Test partial failures
- Test *relaxed* timeout/retry/escalation config

# Example: a typical cache config

*(Guide w/ examples, 2K+ words excl. code)*

- Overall timeout: 500ms*
- Request timeout: 150ms*
- Connect timeout: 200ms
- Pipelining: request timeout does not reset connection

CACHE SLO: P999 <5MS

- Read retry: 2 tries*, no write-back, no backoff
- Write retry: 3 tries*, random backoff (5-200ms)
- Overall retry budget: 20% of requests, minimum 10 retries per second (10-sec credit window)

- Blackhole: 5 consecutive failures*, revive after 30 seconds
- Centralized topology manager, changes dampened >1min

# Reduce variance

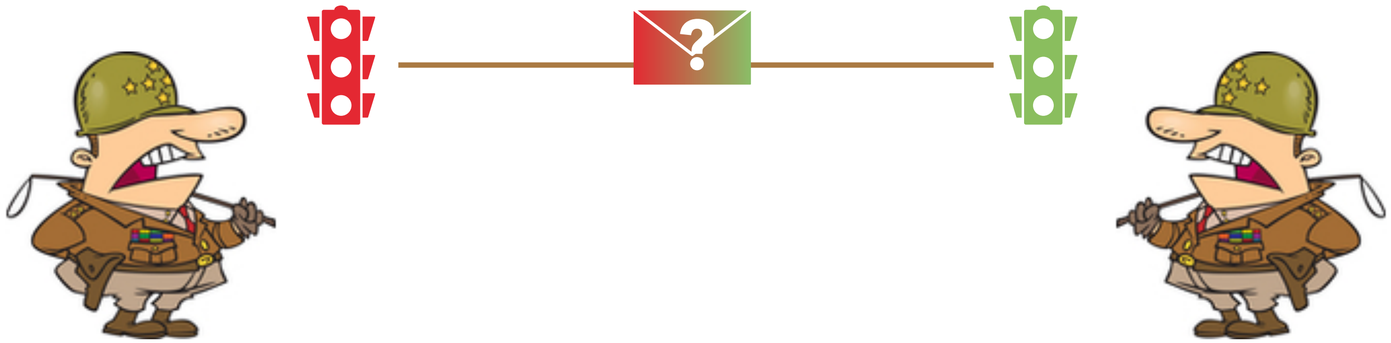
# Example: Making cache more predictable

- Staying off shared hosts
- Rate limiting new connections with iptables
- Setting CPU affinity for sirq, application

- Removing shared locks
- All blocking IO on background threads

# There's no magic–

# Important to know when to give up



Understand common anomalies,
Break the loop,
Form a bigger picture,

and you'll be happy most of the time.

# Questions?