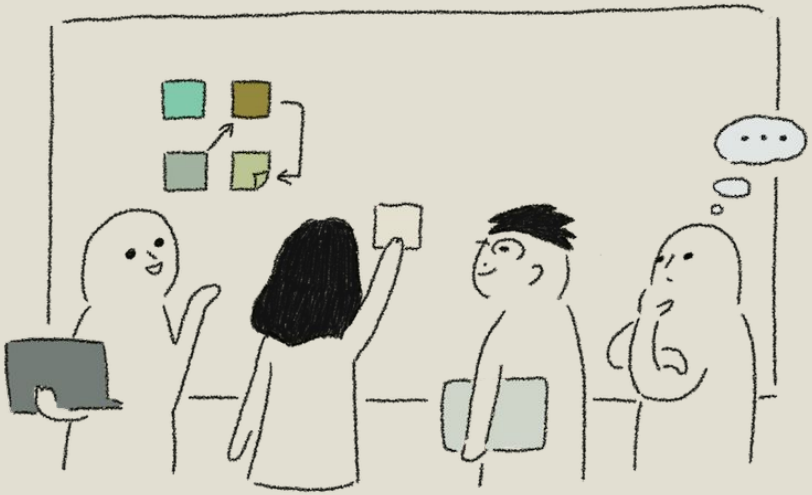
DATE 20250324

Session 4



backend

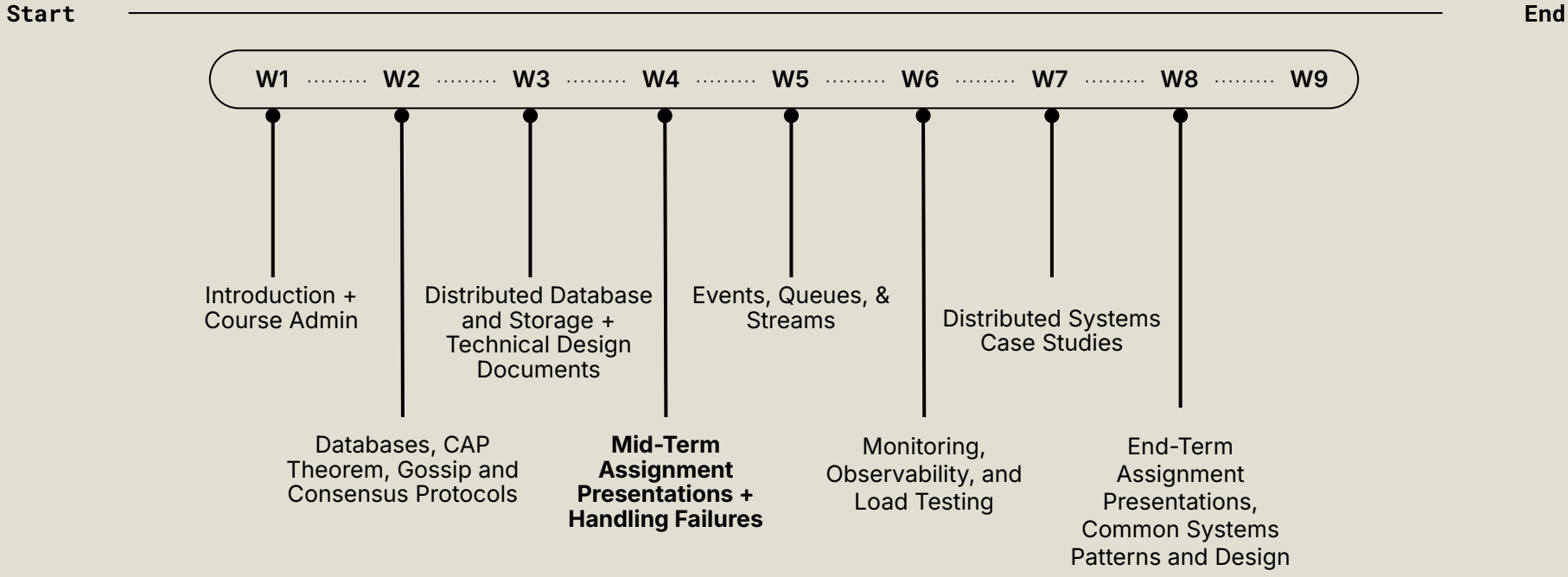
cohort #0 by open camp

#4's agenda

1 ..... admin matters (if any)
2 ..... sharing
3 ..... handling failures
4 ..... w4 assignment

admin matters

Curriculum: https://opencamp-cc.github.io/backend-curriculum/



4.1 sharing

Assignment #3

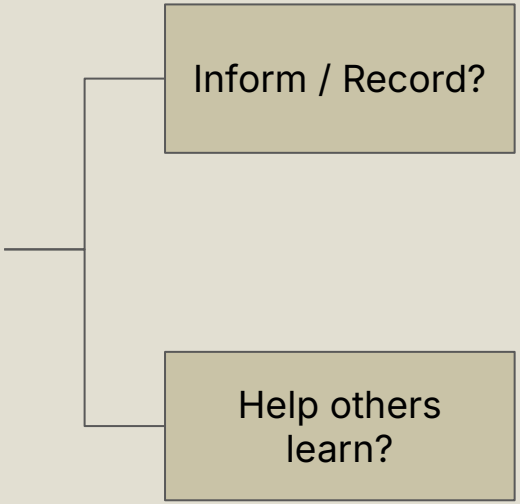
Sharing in W4:

- You sharing should be around 6-10 minutes with simple slides
- Share about your choice of protocol (why it was chosen), and your learnings when building a PoC (eg. surprises, challenges)
- What’s the tech stack you are planning to use to build the system?
- Do you need help with anything? Feel free to call out.

Is my audience...

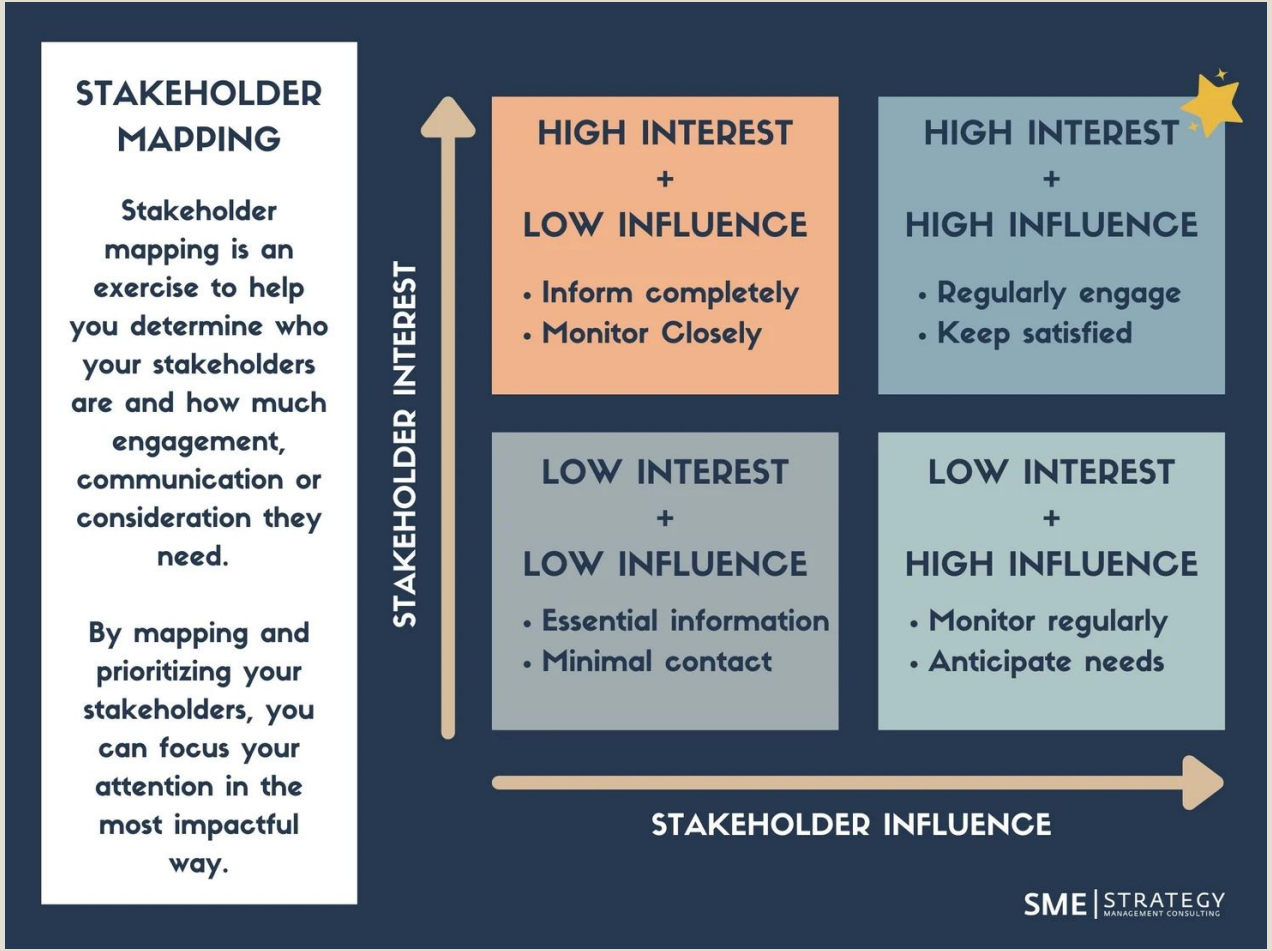
- Part of my organization?
  - Be direct, and include as much details as what your audience might want to know internally
- Public / Users/ Customers?
  - Include details that your users or customers may need to instill confidence without overly sharing too much internal details

Is my goal to...



Be direct, and include as much details as possible.

Include details and any insights from the journey / process, such as ‘In hindsight...”, and things like how to spot or identify similar challenges in future.



What is Stakeholder Engagement, and Why is it Important for Strategic Planning?

https://www.smestrategy.net/blog/stakeholder-engagement-management-for-strategic-planning

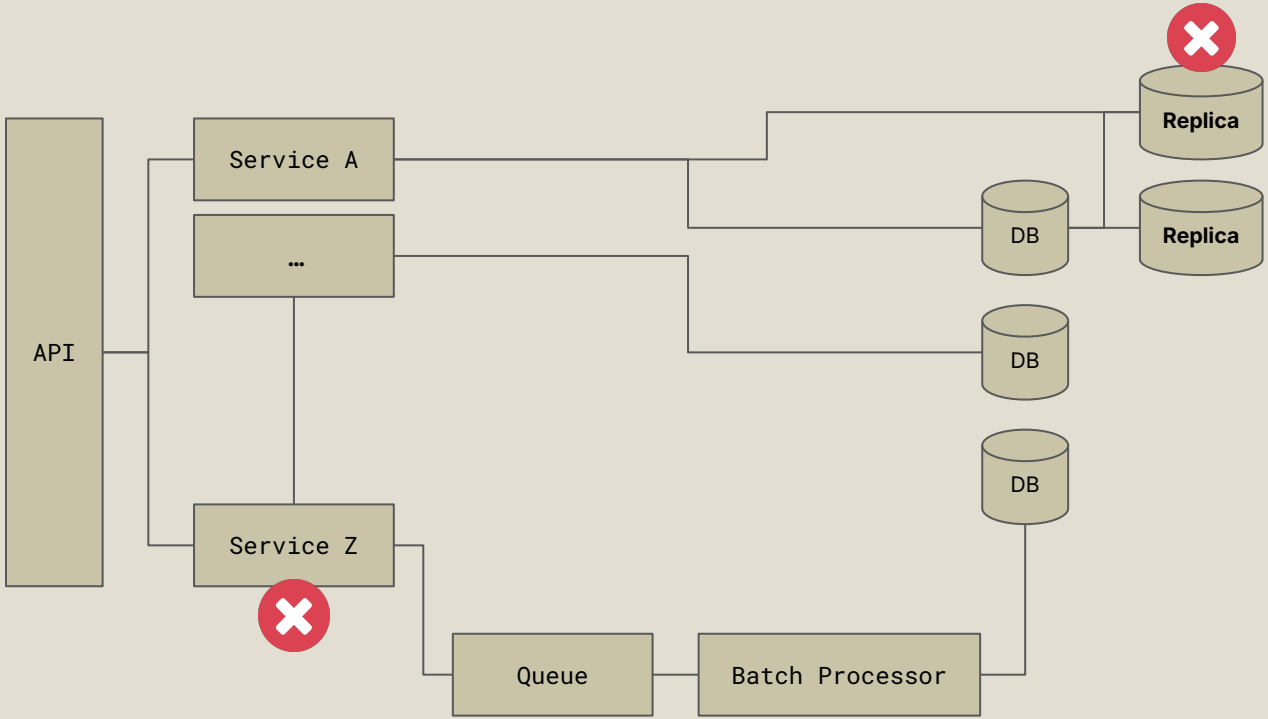
Commitments to long projects are not one-offs. They need to be renewed every once in a while.

Engage your stakeholders, your users, your team periodically and share progress and updates.

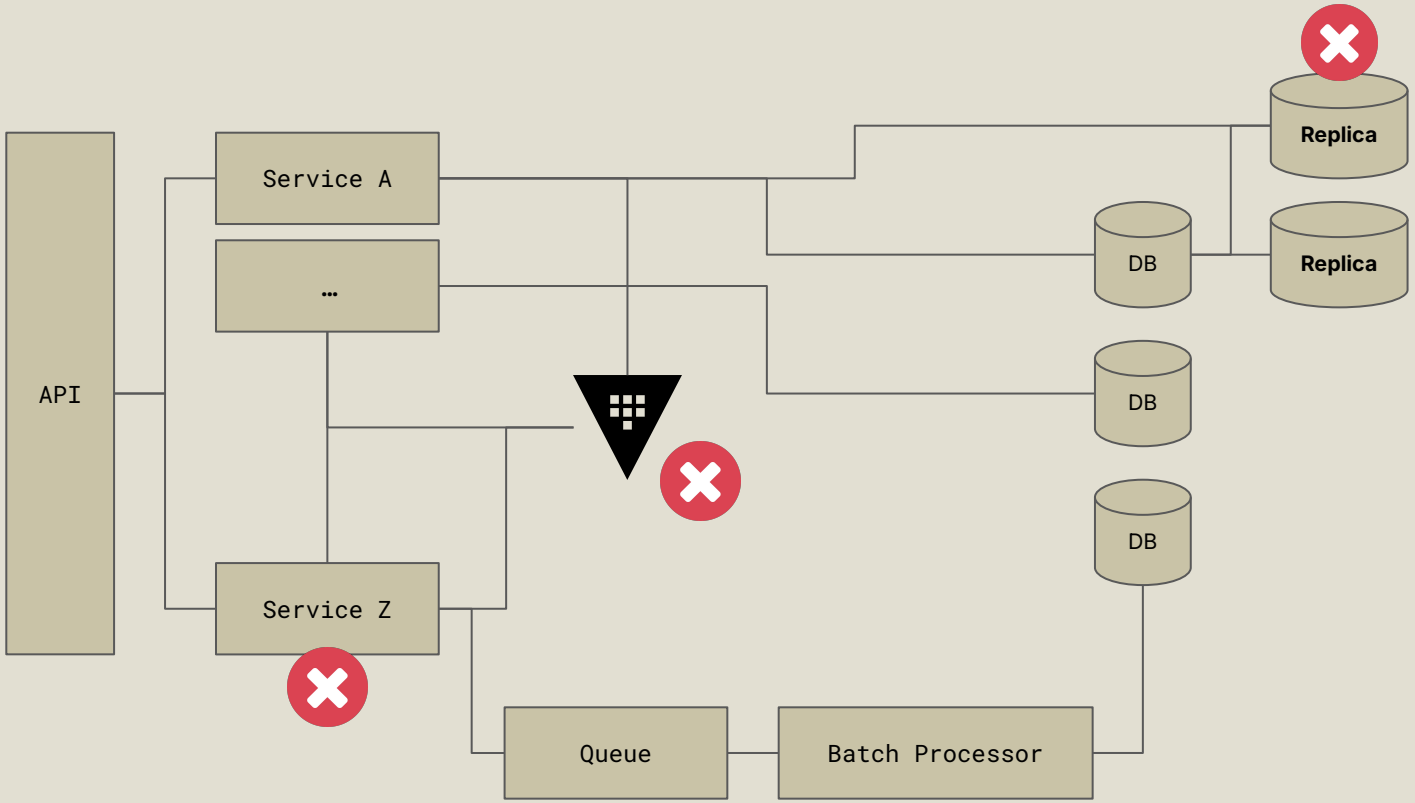
Otherwise, a project might be paused, stopped, or scraped when it comes to a new round of business planning season.

break time

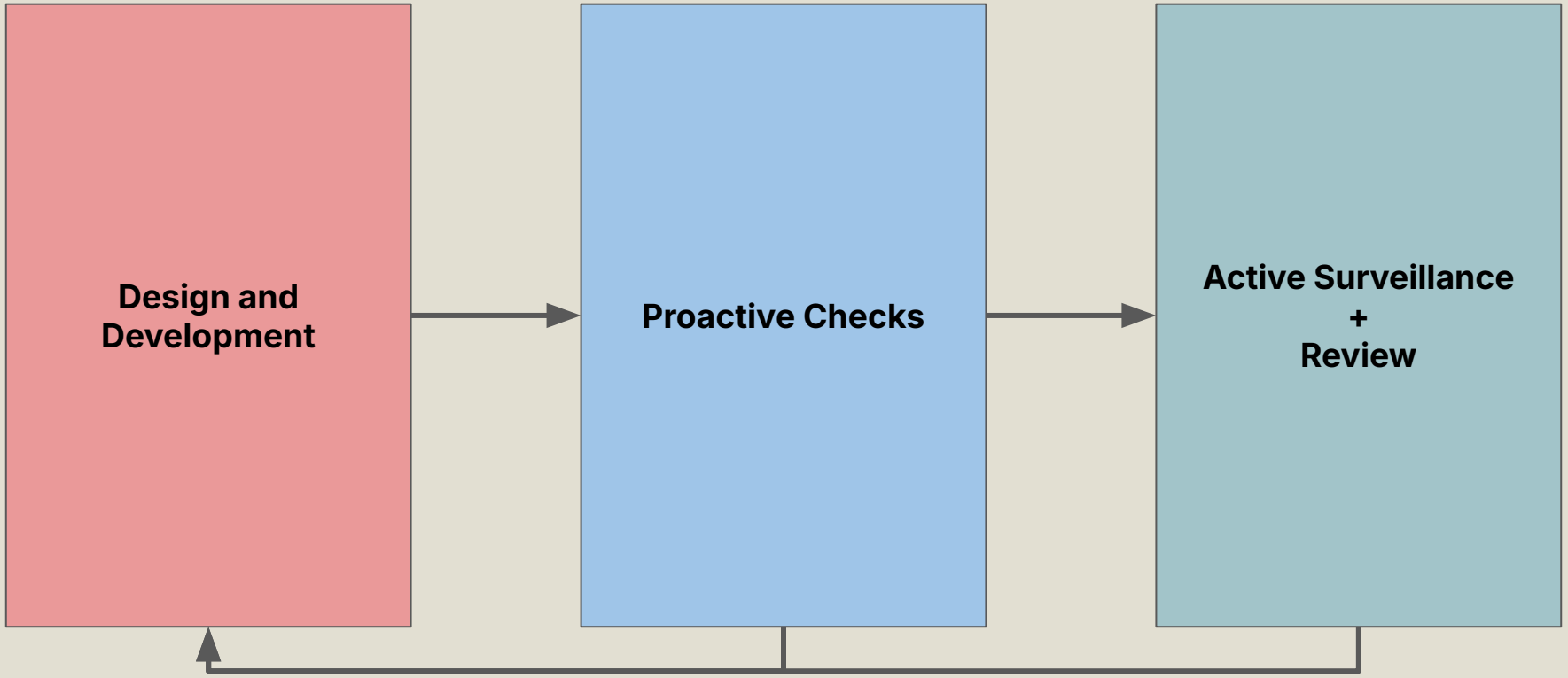
4.2 handling failures



Designing for Failures

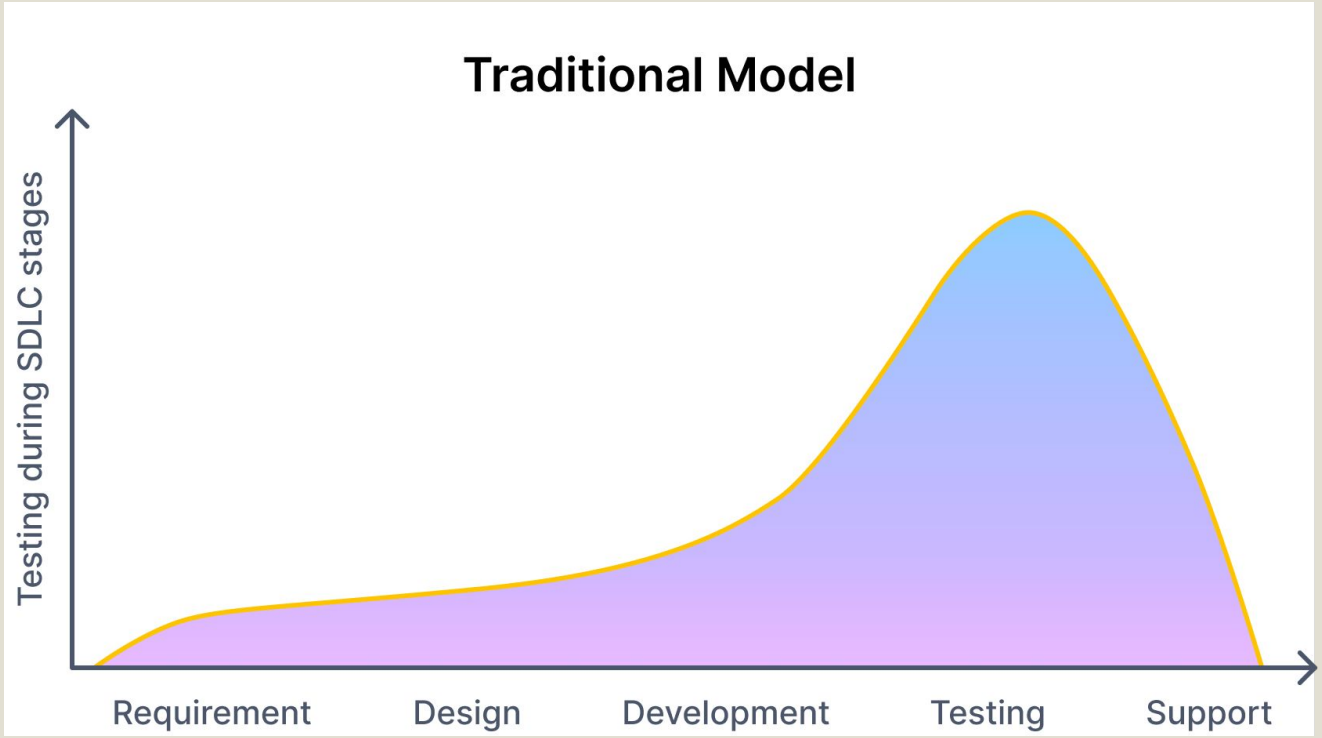


New Tools = Higher Overhead, Increased Risks



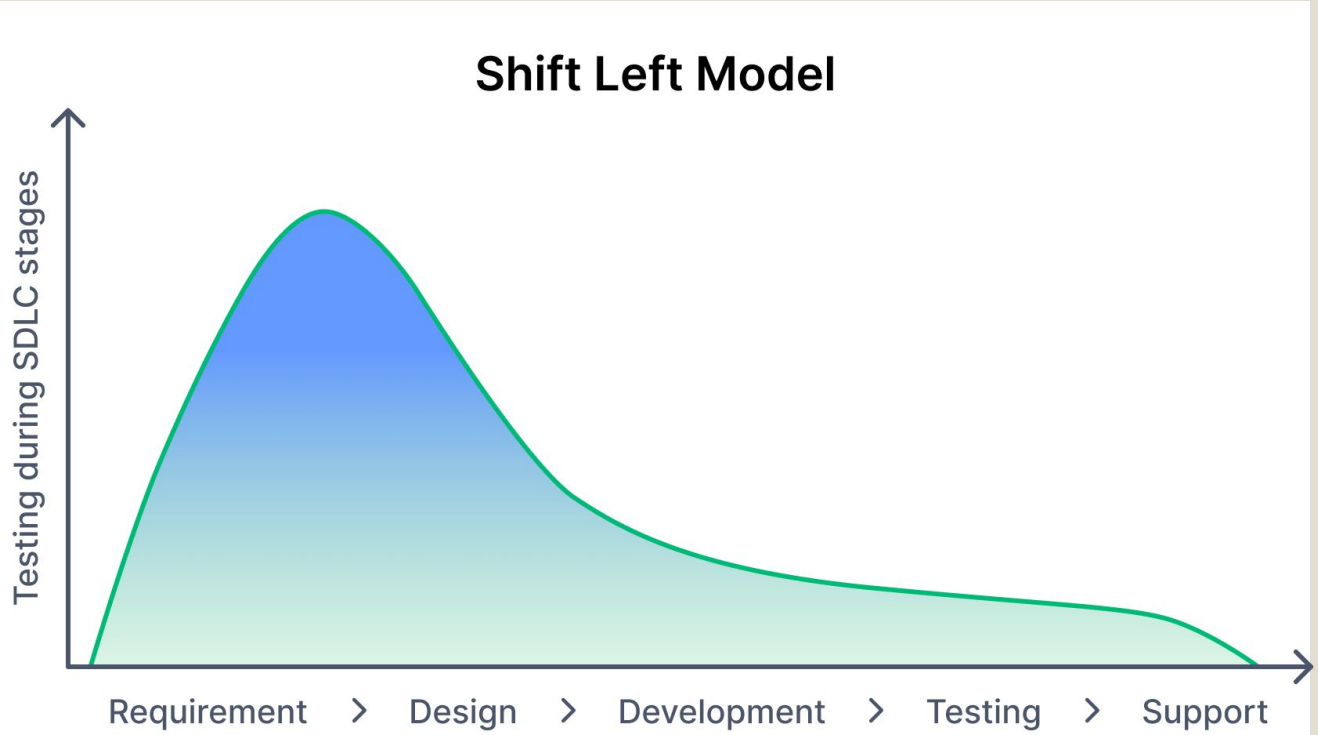
Failure Handling Methods, Categorized

4.2.1 design and development



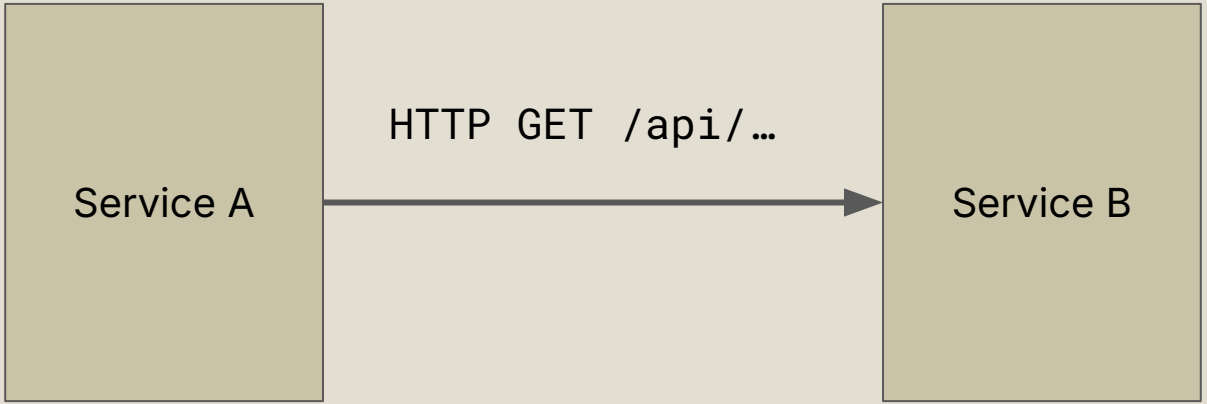
Traditional Model of Development

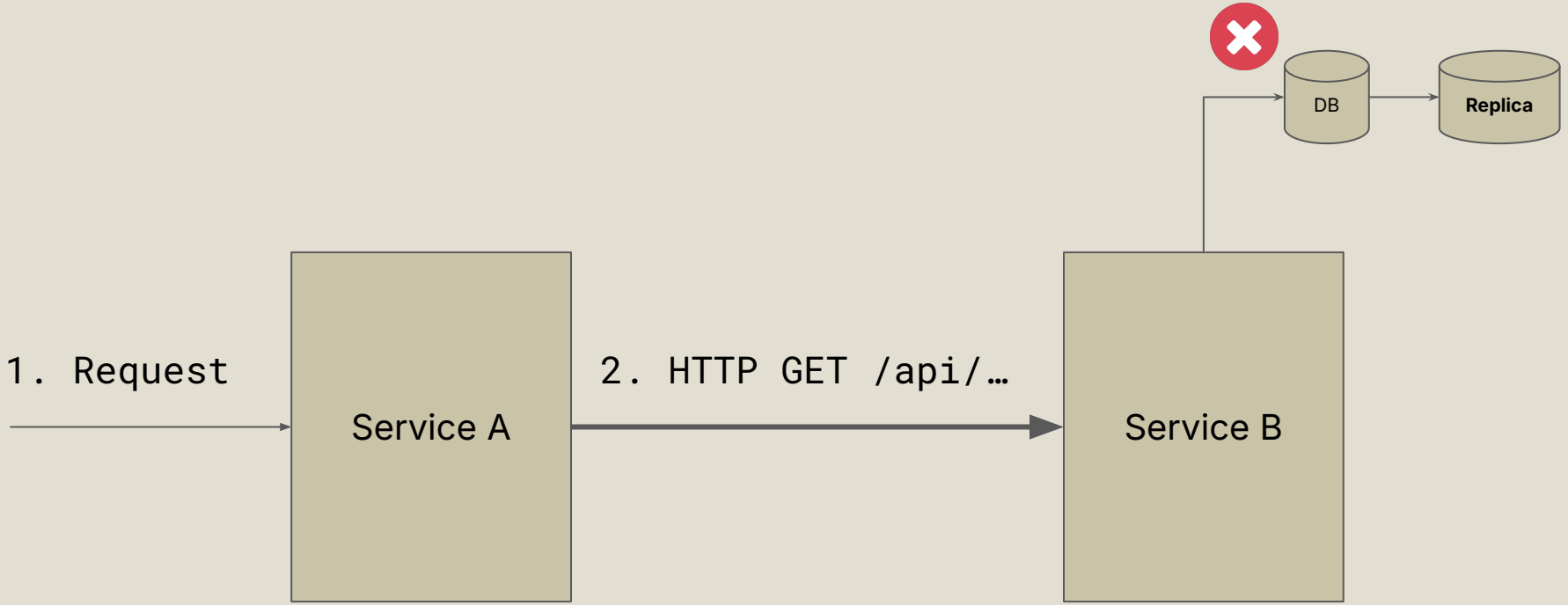
https://birdeatsbug.com/blog/shift-left-testing



Shift Left Model: Move Testing Earlier

https://birdeatsbug.com/blog/shift-left-testing


Service A
HTTP GET /api/…
Service B


DB
Replica
1. Request
Service A
2. HTTP GET /api/…
Service B

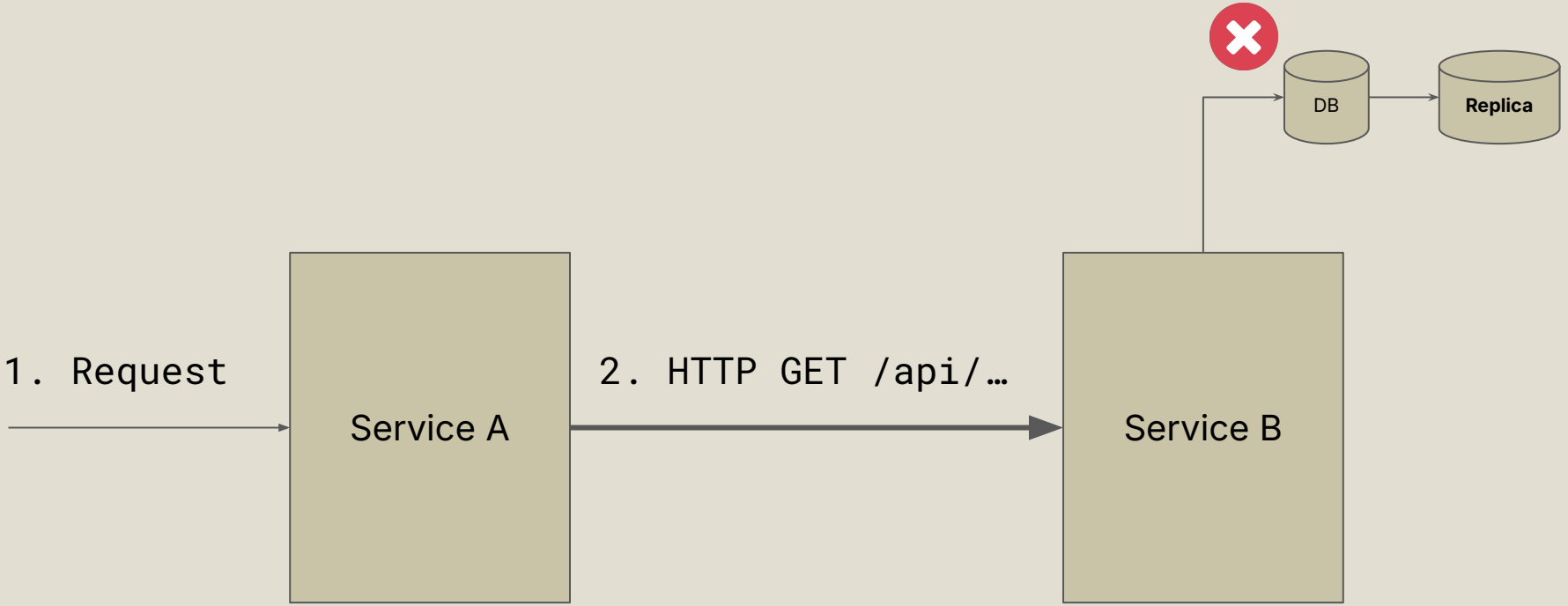
```
response, _ := http.Get("https://….something.com/api/…")
```

How long before timeout?

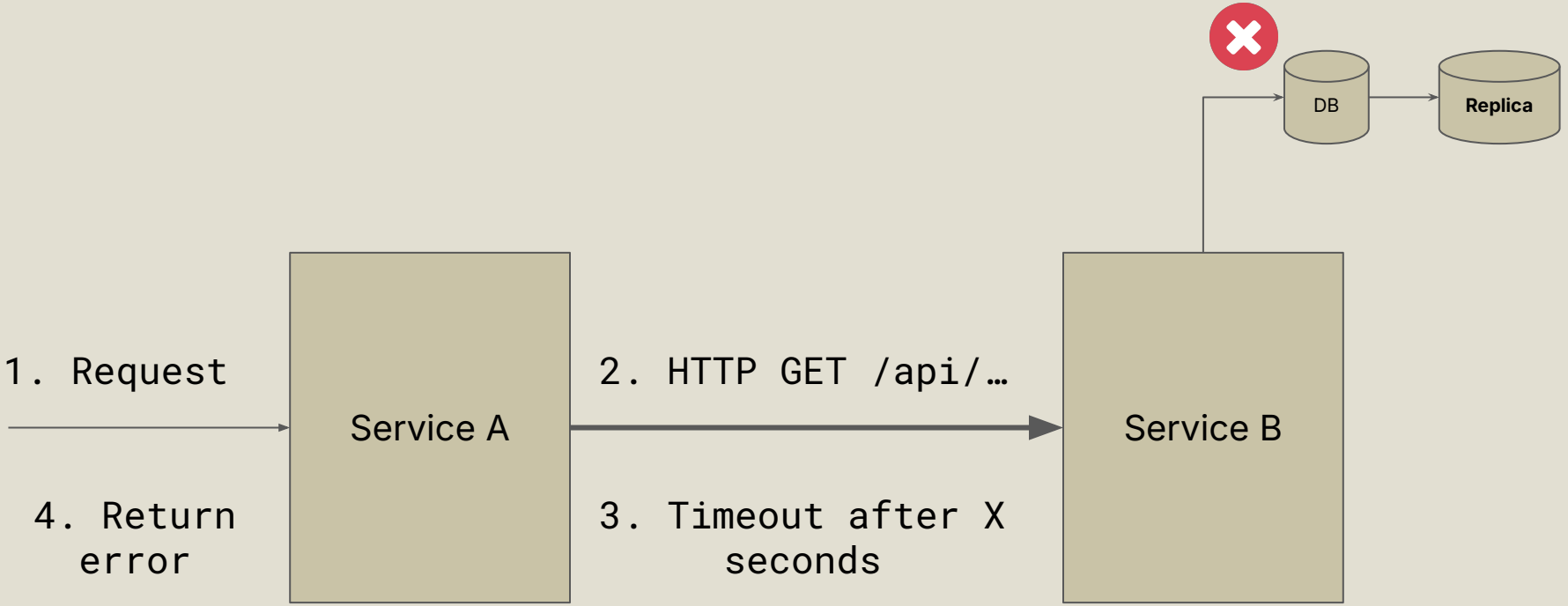
“Among other things, http.Client configures a timeout that short-circuits long-running connections.

The default for this value is 0, which is interpreted as “no timeout”.”

Don’t use Go’s default HTTP client (in production)
https://medium.com/@nate510/don-t-use-go-s-default-http-client-4804cb19f779

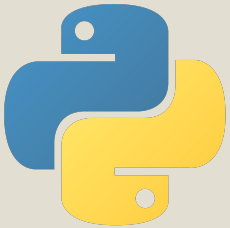


Request gets blocked, no processing continues



Request gets unblocked after X seconds, processing continues (prioritize Availability)

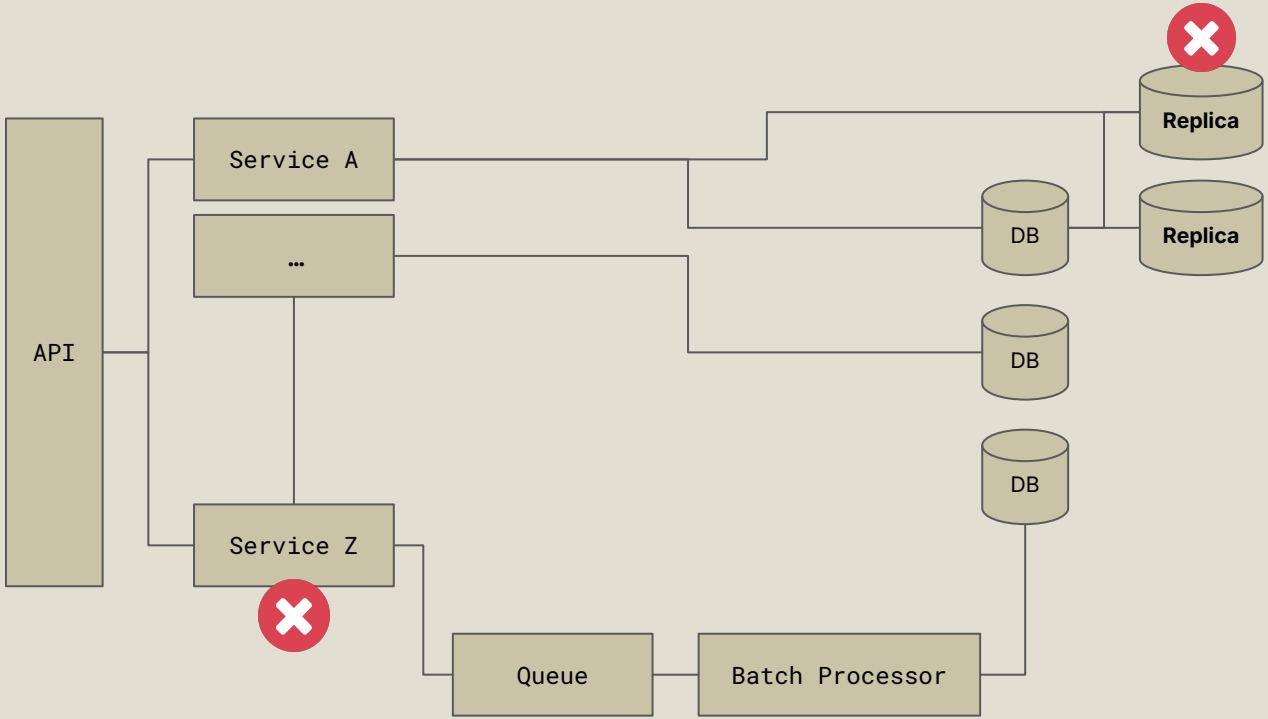
```go
var netClient = &http.Client{
    Timeout: time.Second * 5,      // 5 seconds
}response, _ := netClient.Get(url)
```



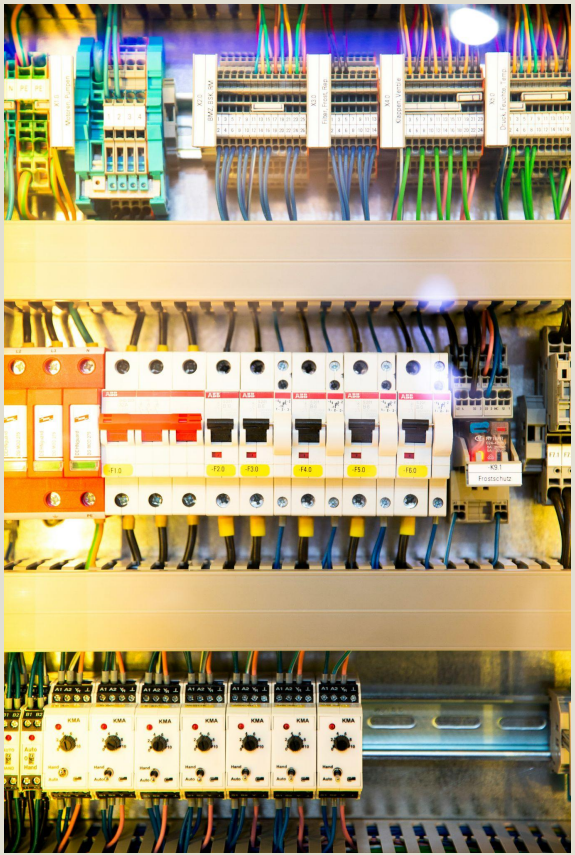
```python
import requests

r = requests.get(url, timeout=5)
```

Don’t use Go’s default HTTP client (in production)

https://medium.com/@nate510/don-t-use-go-s-default-http-client-4804cb19f779



Complex Systems: Dealing with Failures?



Circuit Breakers

“The basic idea behind the circuit breaker is very simple. You wrap a protected function call in a circuit breaker object, which monitors for failures.

Once the failures reach a certain threshold, the circuit breaker trips, and all further calls to the circuit breaker return with an error, without the protected call being made at all.”

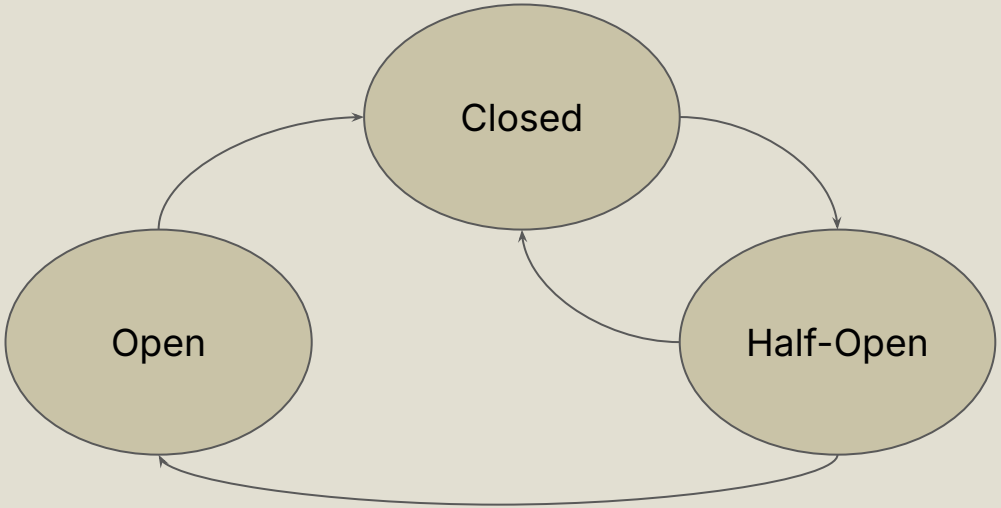
Circuit Breaker

https://martinfowler.com/bliki/CircuitBreaker.html

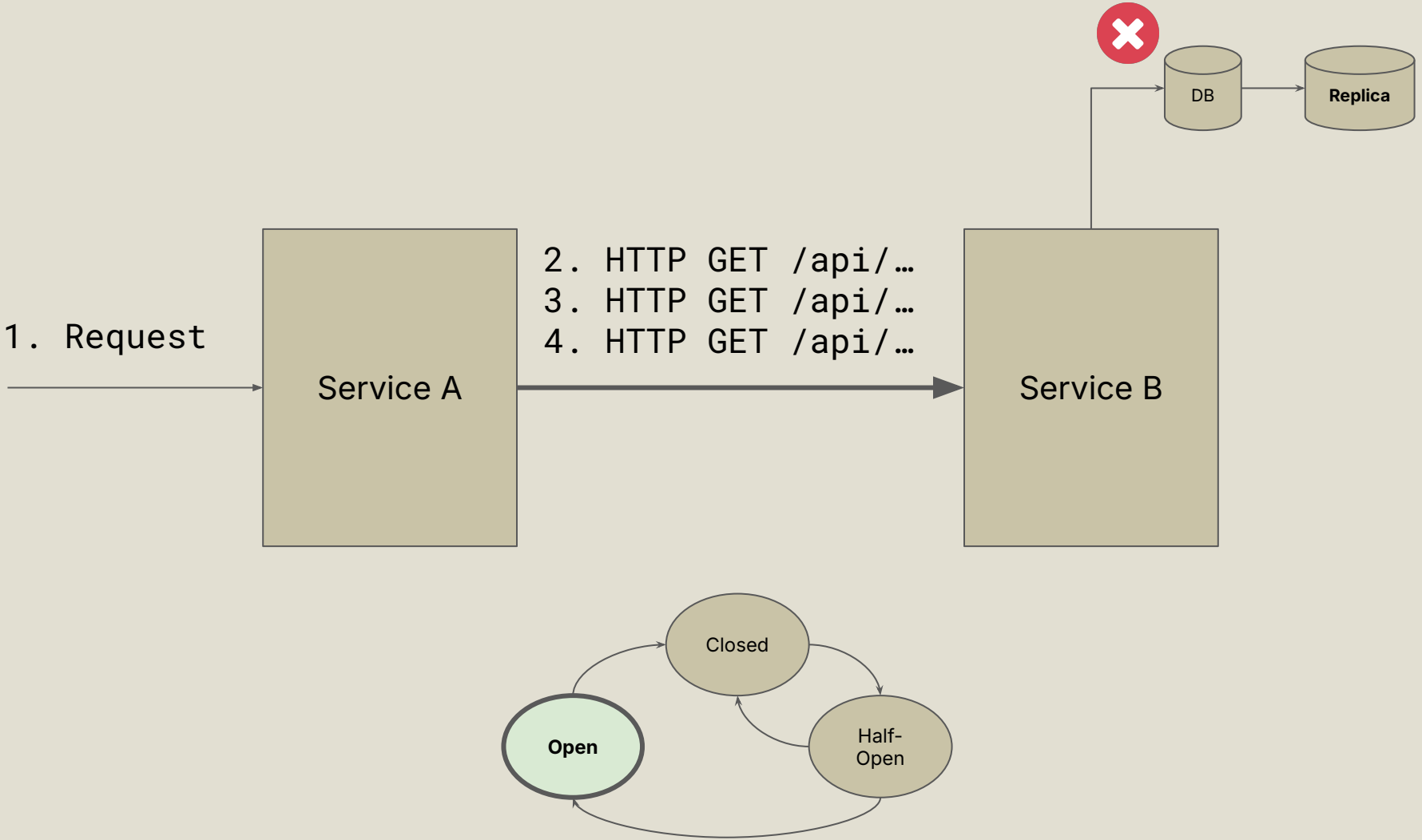
Closed: Normal operation, requests to the service pass through.

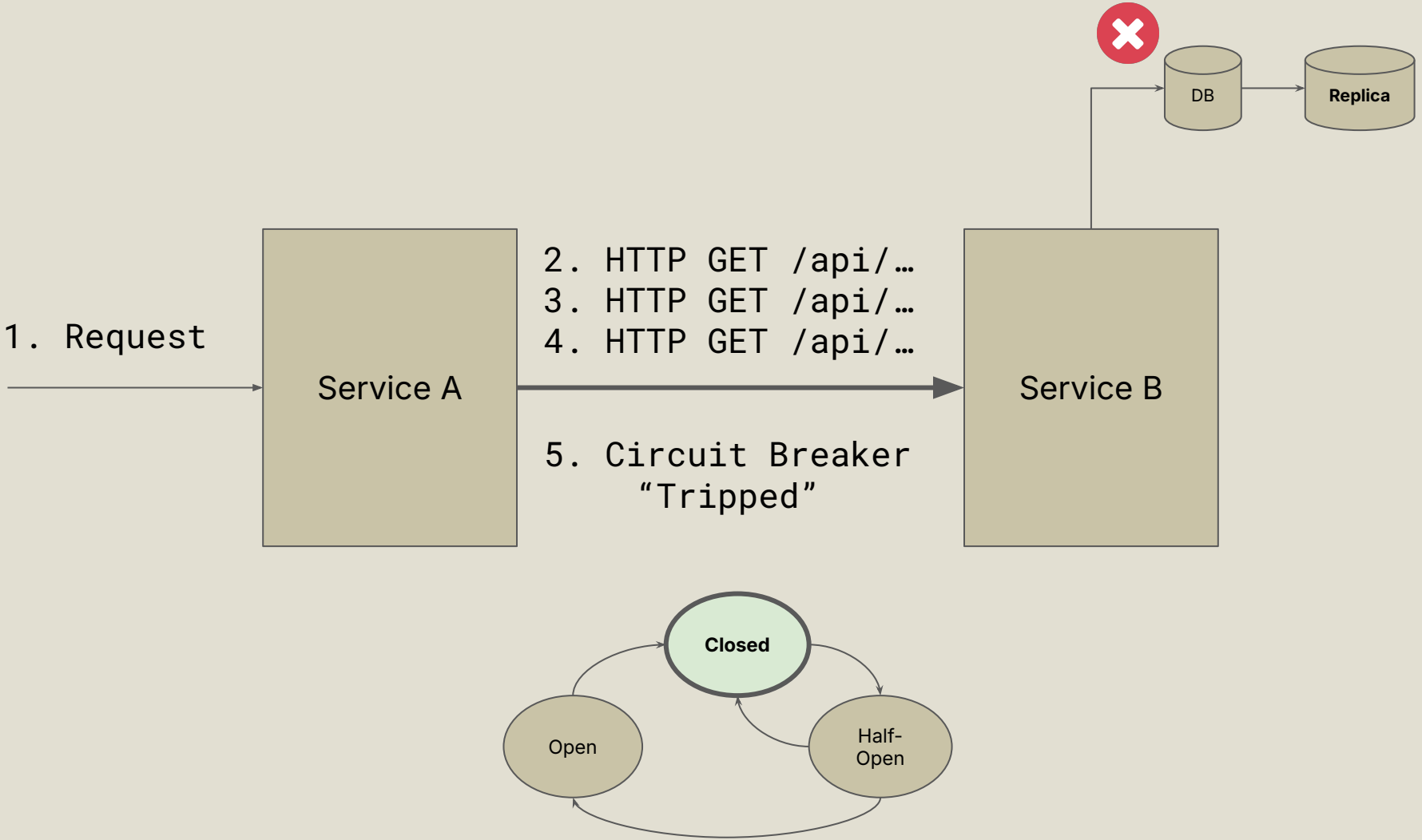
Open: Circuit is open; requests are blocked to allow the target service to recover.

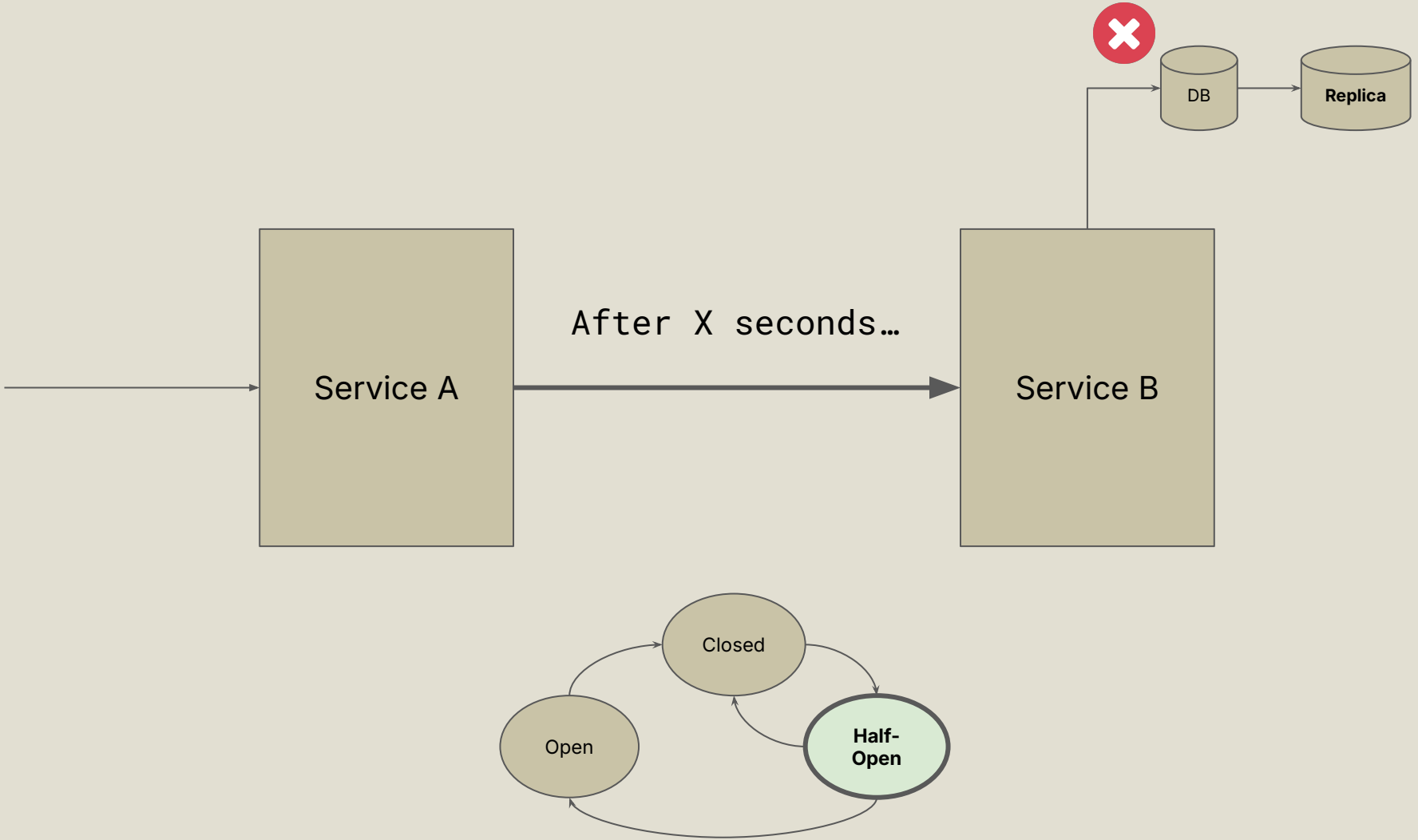
Half-Open: Partially re-enabled to test if the underlying problem is resolved.

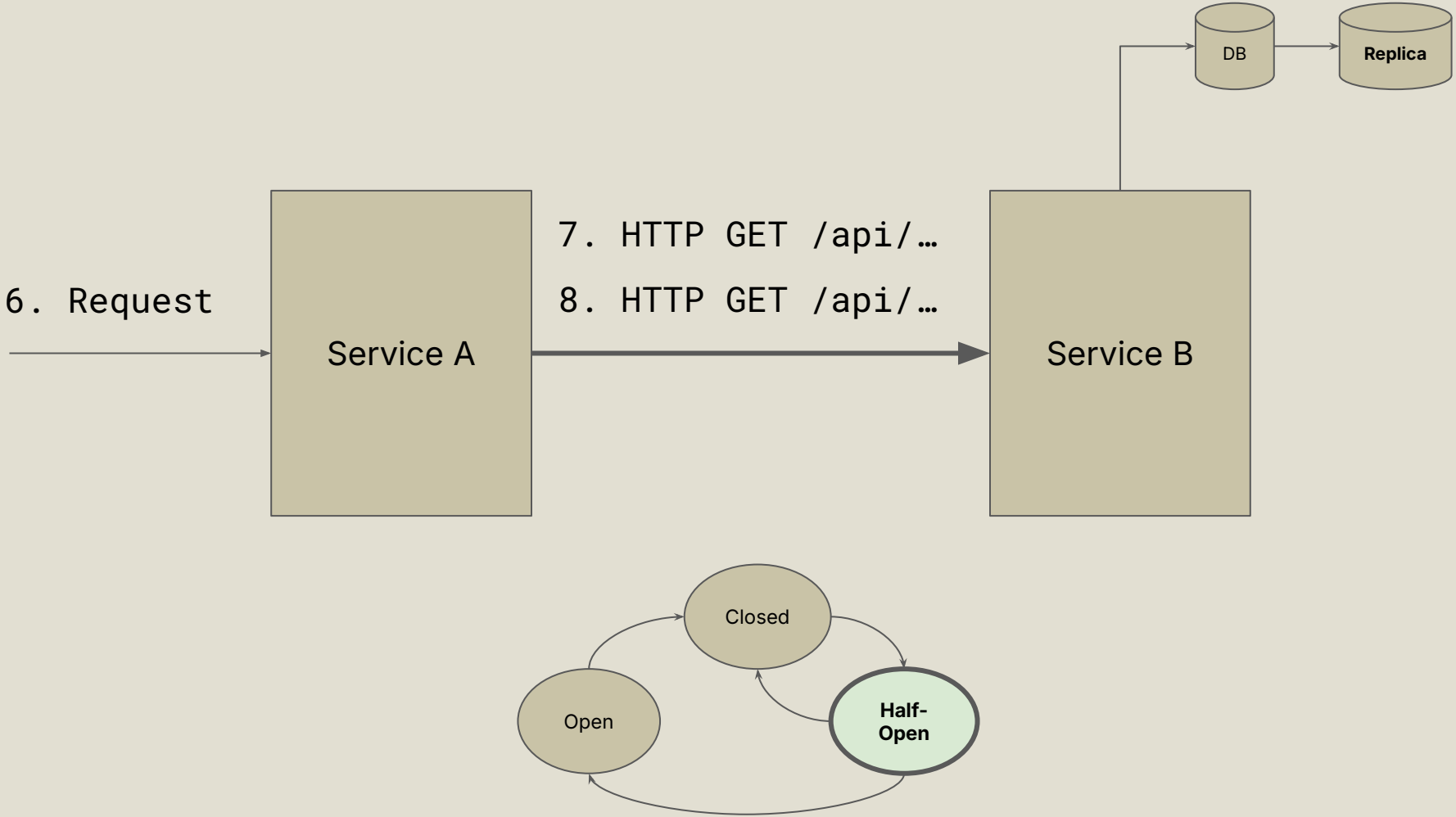


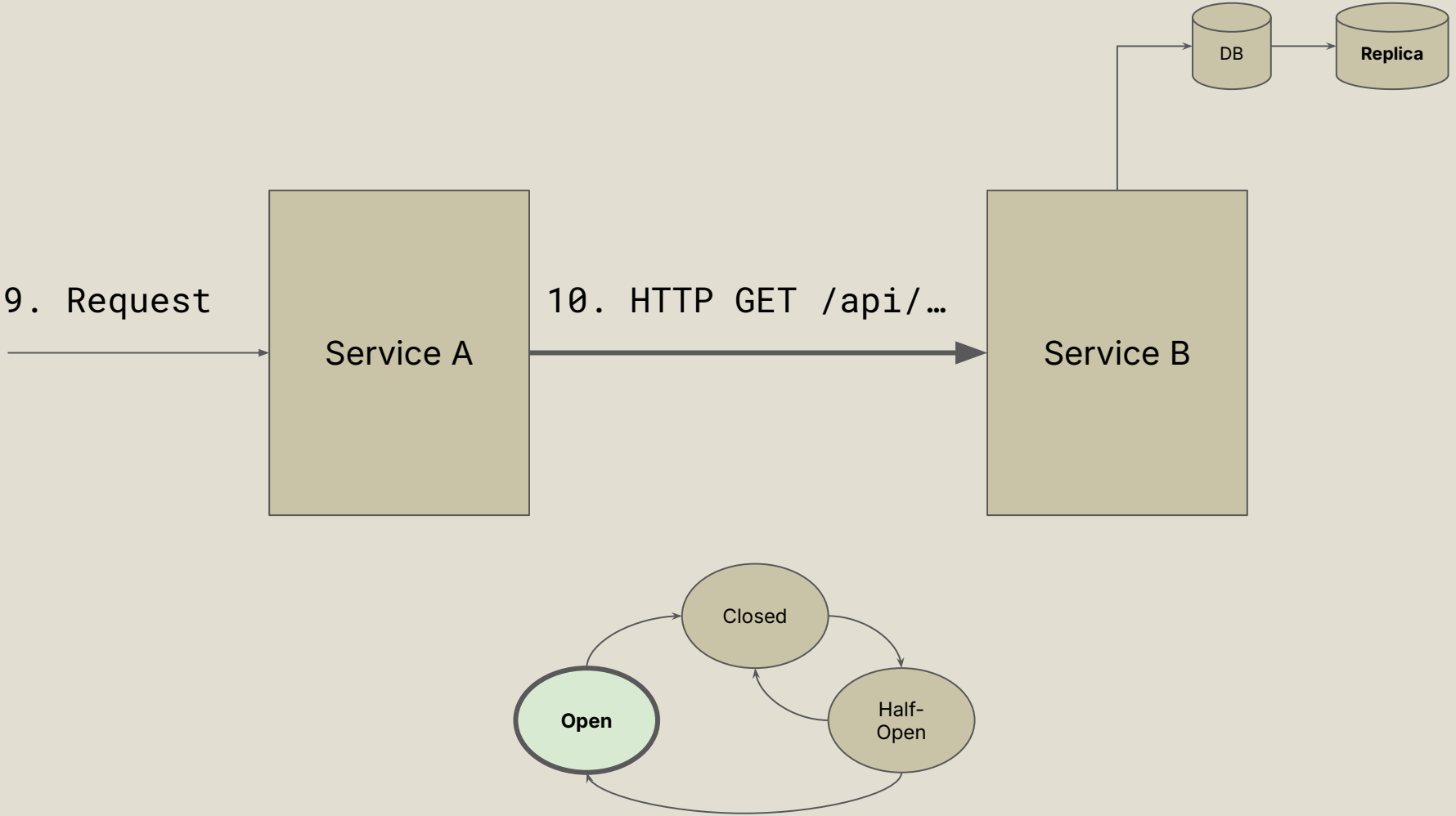
Circuit Breaker: States

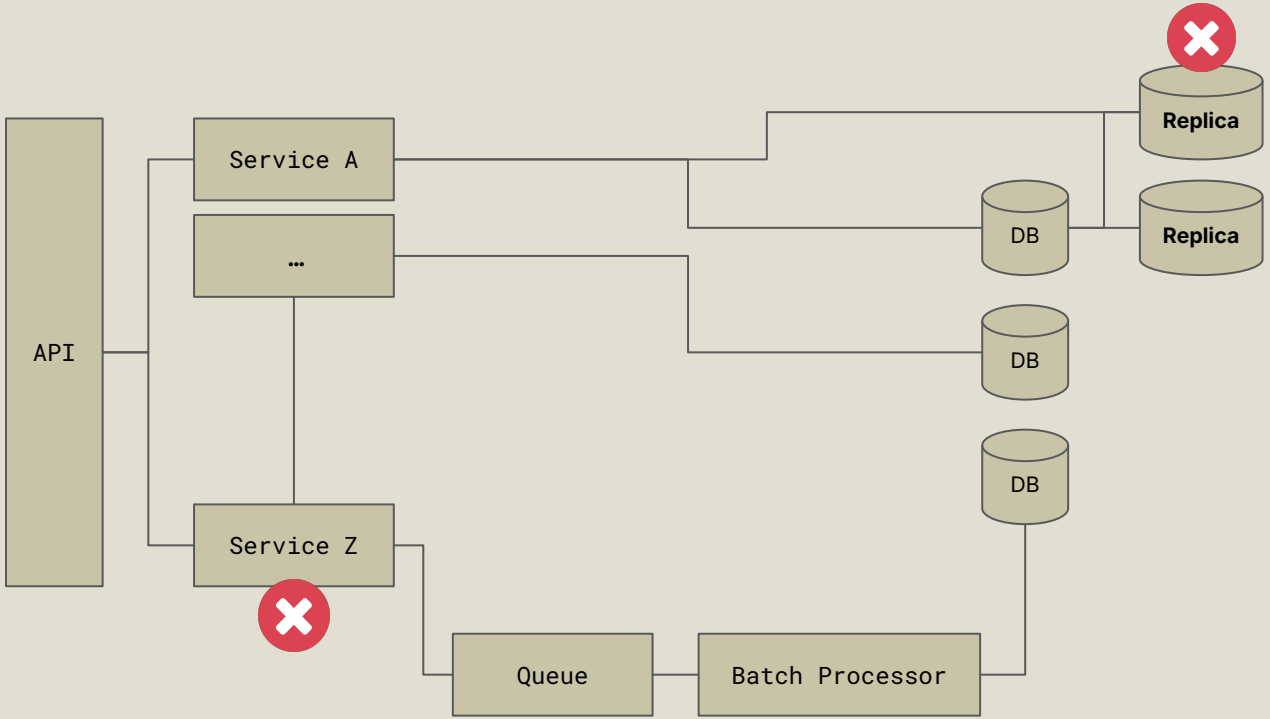

DB
Replica
1. Request
Service A
2. HTTP GET /api/…
3. HTTP GET /api/…
4. HTTP GET /api/…
Service B
Closed
Open
Half-
Open


DB
Replica
1. Request
Service A
2. HTTP GET /api/…
3. HTTP GET /api/…
4. HTTP GET /api/…
5. Circuit Breaker
"Tripped"
Service B
Closed
Open
Half-
Open


DB
Replica
Service A
After X seconds…
Service B
Closed
Open
Half-
Open


DB
Replica
6. Request
Service A
7. HTTP GET /api/…
8. HTTP GET /api/…
Service B
Closed
Open
Half-
Open


DB
Replica
9. Request
Service A
10. HTTP GET /api/…
Service B
Closed
Open
Half-
Open



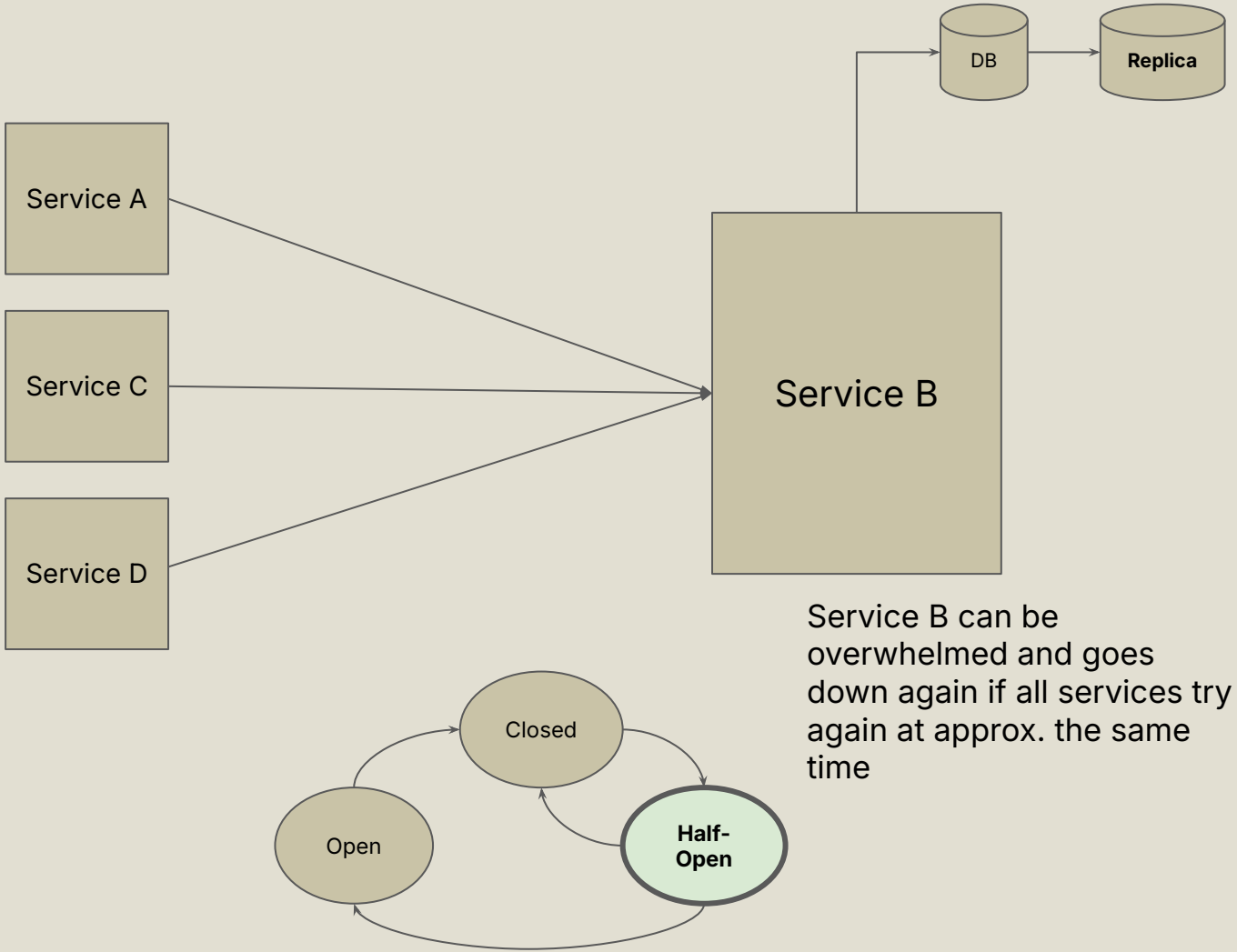
Will timeouts + Circuit Breaker be sufficient?

“A thundering herd incident for an API typically occurs when a large number of clients or services simultaneously send requests to an API after a period of unavailability or delay.

This could be caused by either services that you own or third party services retrying requests after a period of downtime or instability.”

Thundering Herd Problem

https://encore.dev/blog/thundering-herd-problem

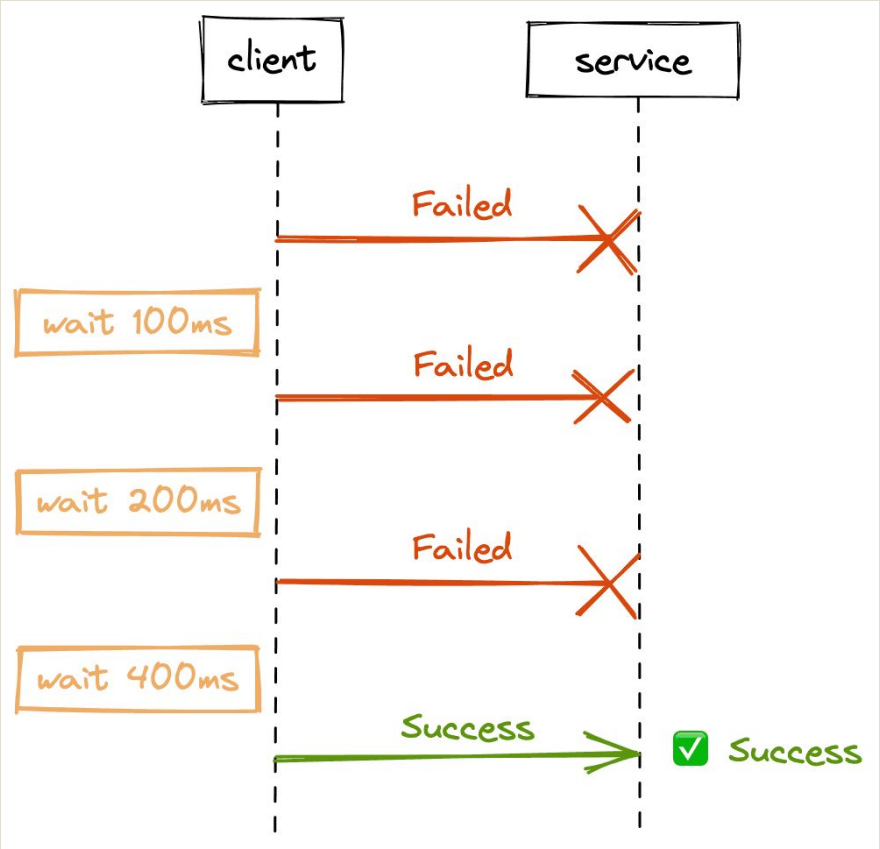


Service B can be overwhelmed and goes down again if all services try again at approx. the same time

“The gist is that if you have 100 people run to a doorway, the doorway might come crashing down.

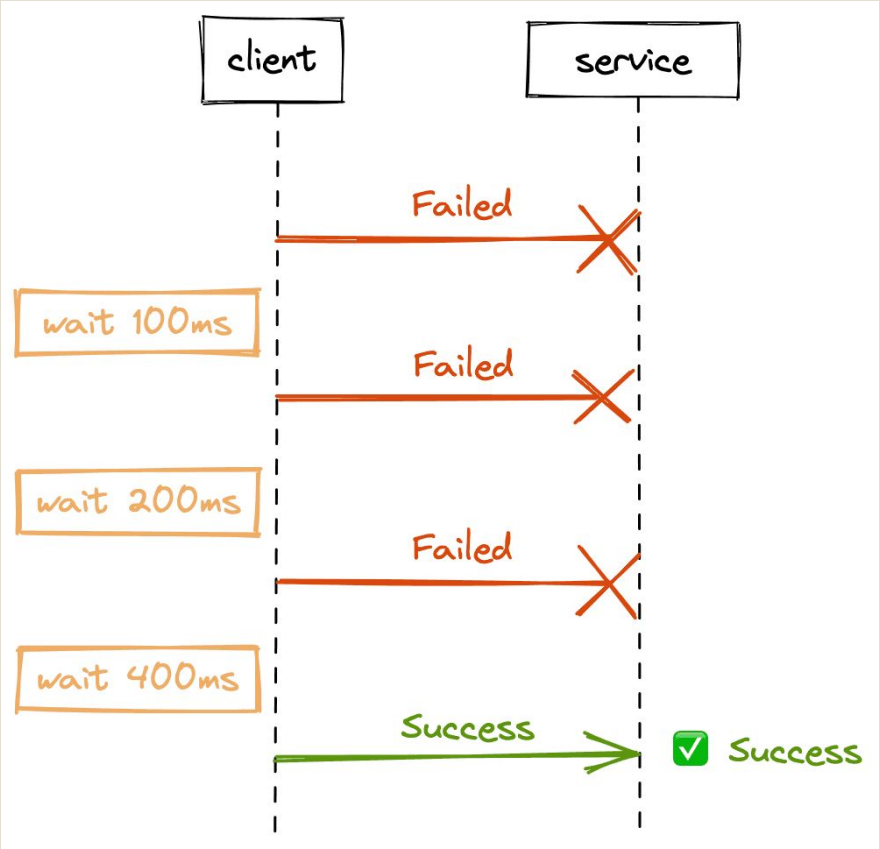
If instead everyone ran at different speeds and arrived at random intervals, the doorway is still usable and the queue pressure is significantly lessened.”

https://medium.com/paypal-tech/thundering-herd-jitter-63a57b38919d



Exponential Backoff

https://www.tylercrosse.com/ideas/exponential-backoff



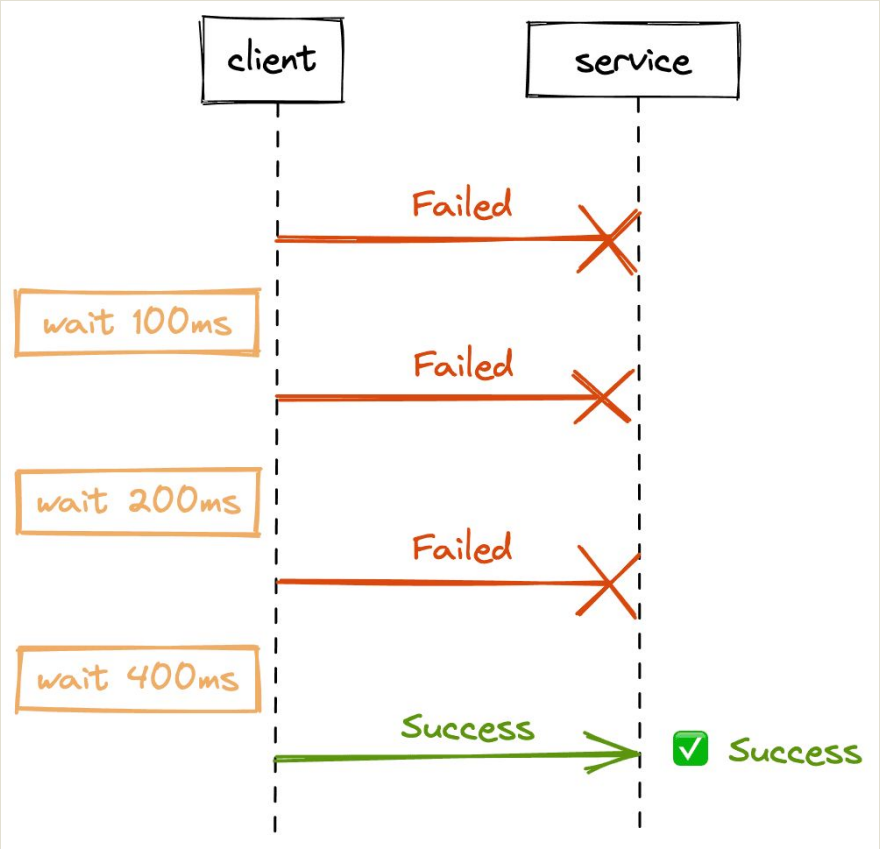
Problem: Still may cause thundering herd!

100ms, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, 51200ms…

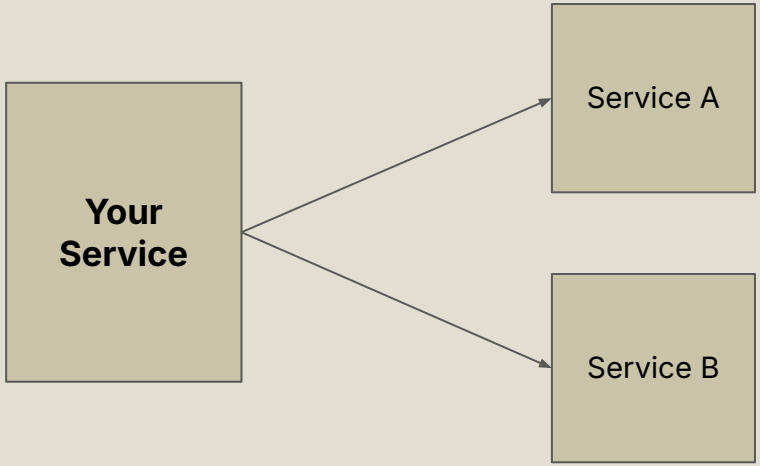
random Xms +

random Xms +

random Xms +

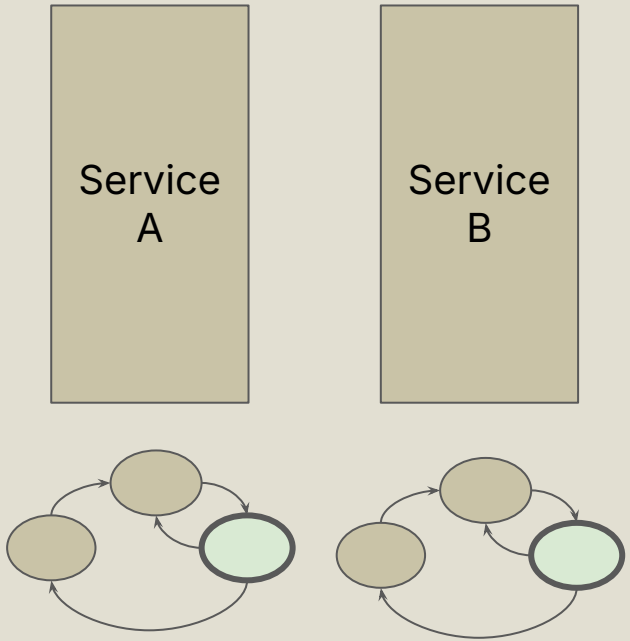


Solution: Add a random jitter

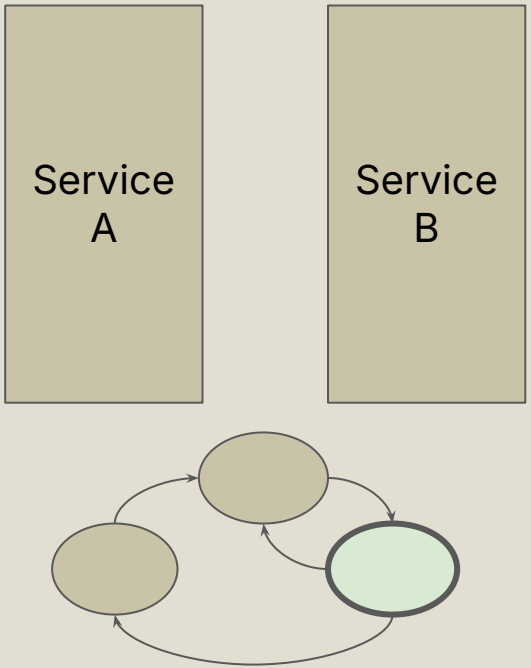


Implementing Circuit Breakers

1. Per Service

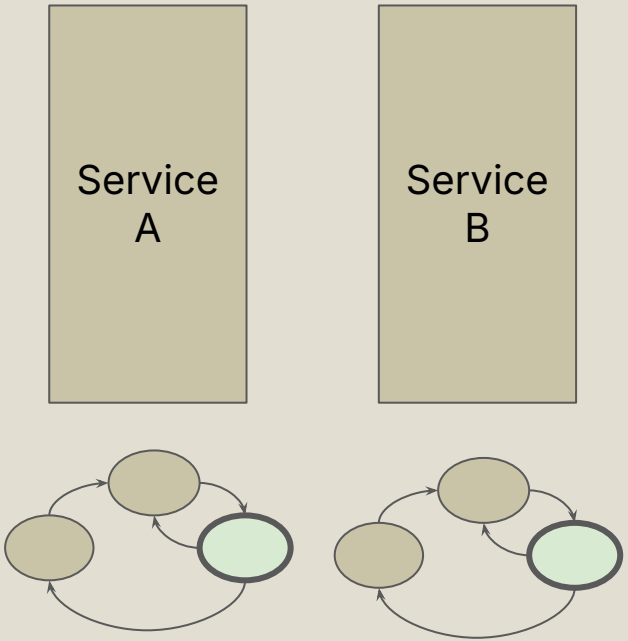


2. Global



Two Options for Circuit Breakers

1. Per Service



- Ideal if services don’t have much interdependencies

- Faster recovery time when a service goes down and returns

- More confusing to know which state each service is in when there is a large scale failure

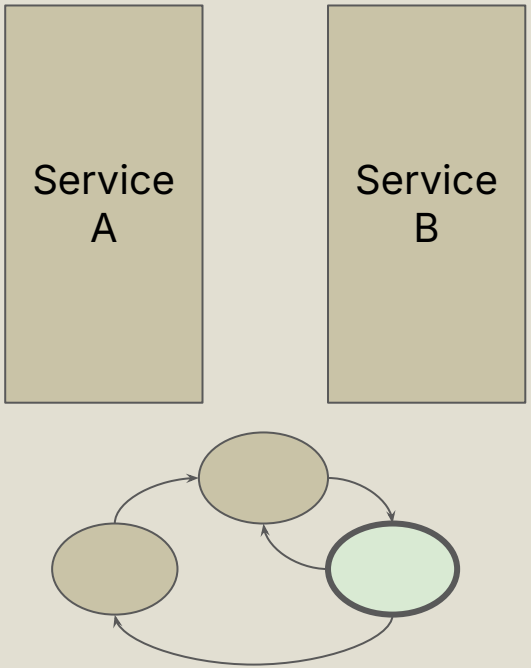
Per Service Circuit Breaker

2. Global

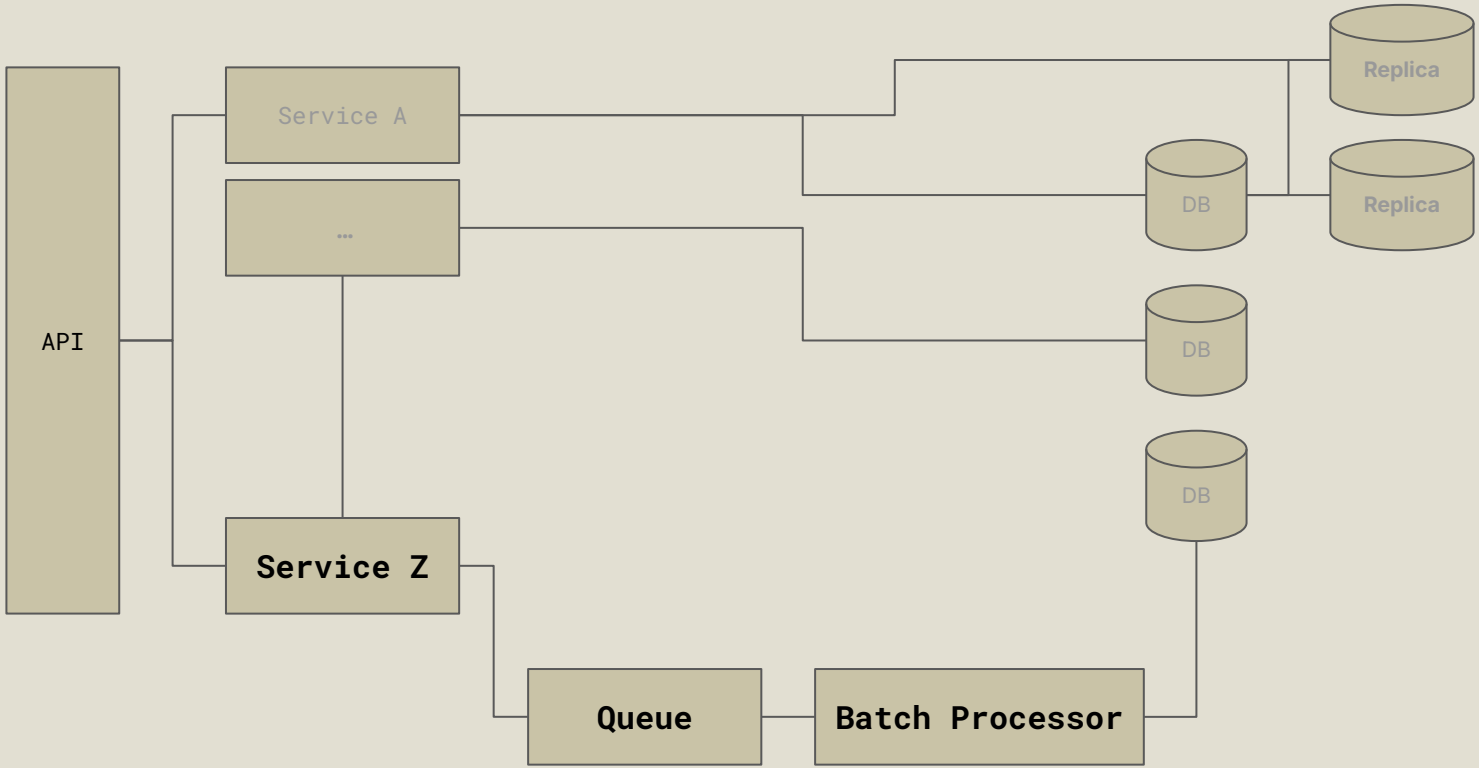
- Ideal if services have lots of interdependencies

- Bigger impact if only one service goes down and returns

- Easier to know which state the circuit breaker is in when there is a large scale failure



Global Circuit Breaker for All Services

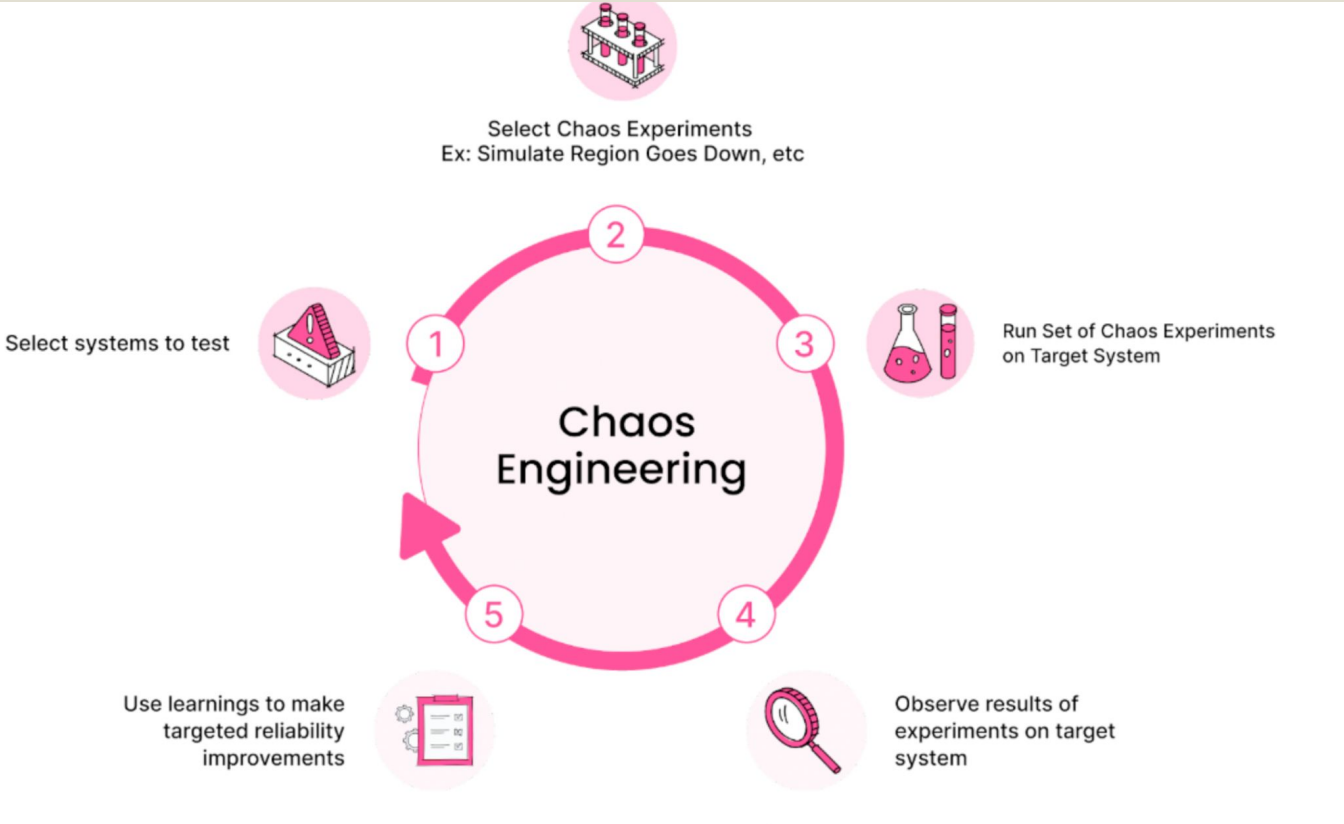


Queues: Async Processing and Retrieval
Will cover more in W5

	Essential?	Risks
Timeouts	Yes	Infinite timeouts
Circuit Breaker	Depends	Heavily loaded services might not be able to recover if they are swamped
Jitter	Depends	Can be implemented even for timeouts, add some randomness per API call. Reduces impact of thundering herd

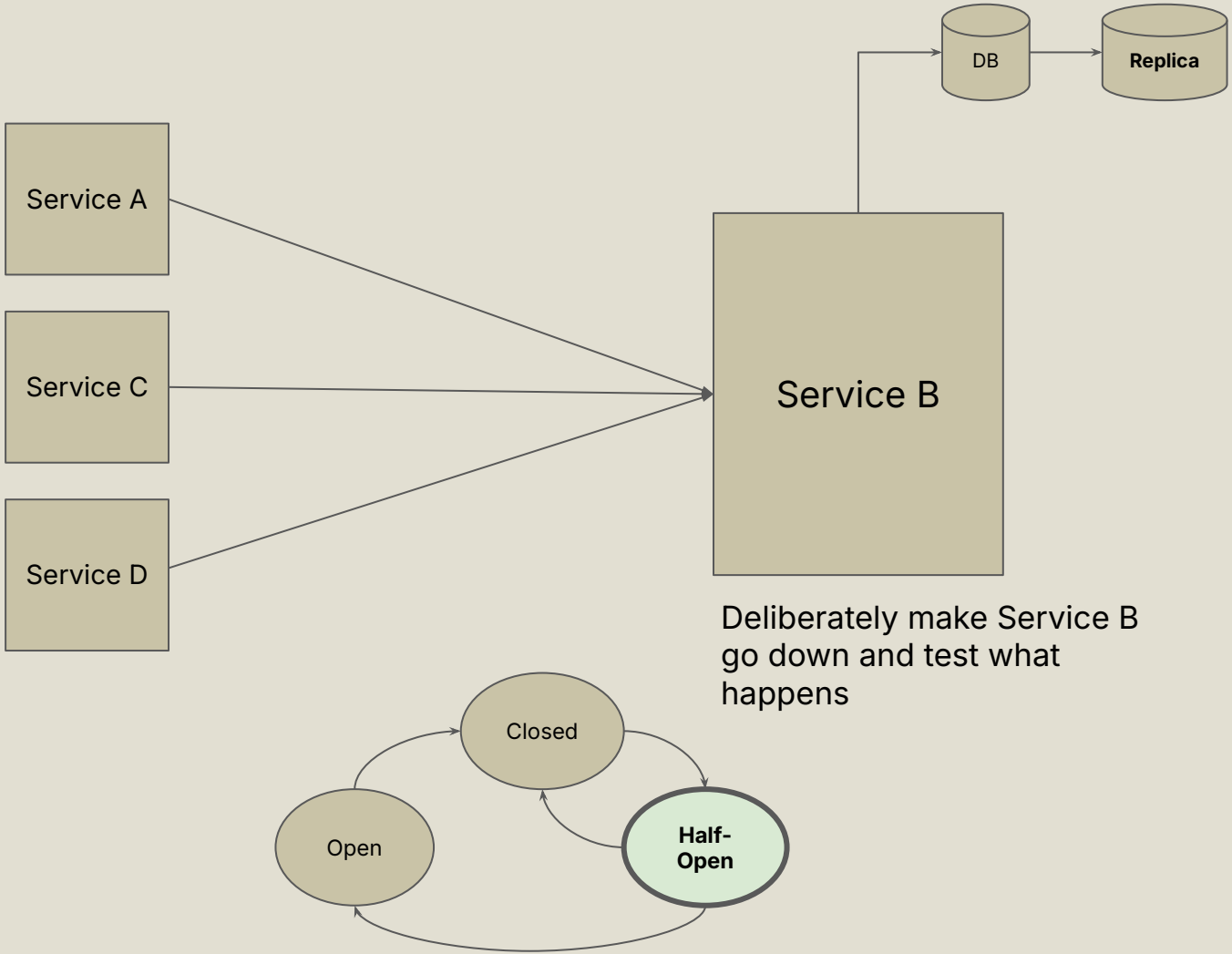
Summary: Design and Development

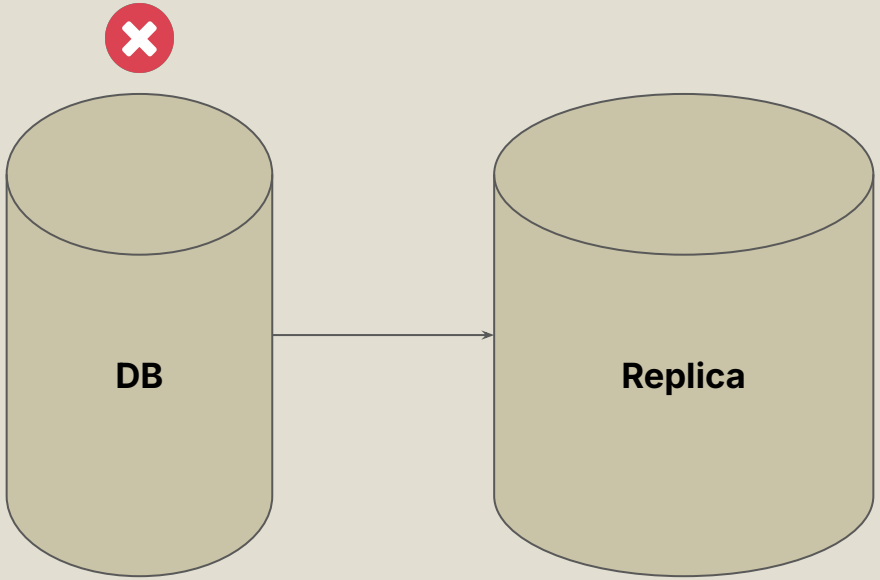
4.2.2 proactive



Chaos Engineering

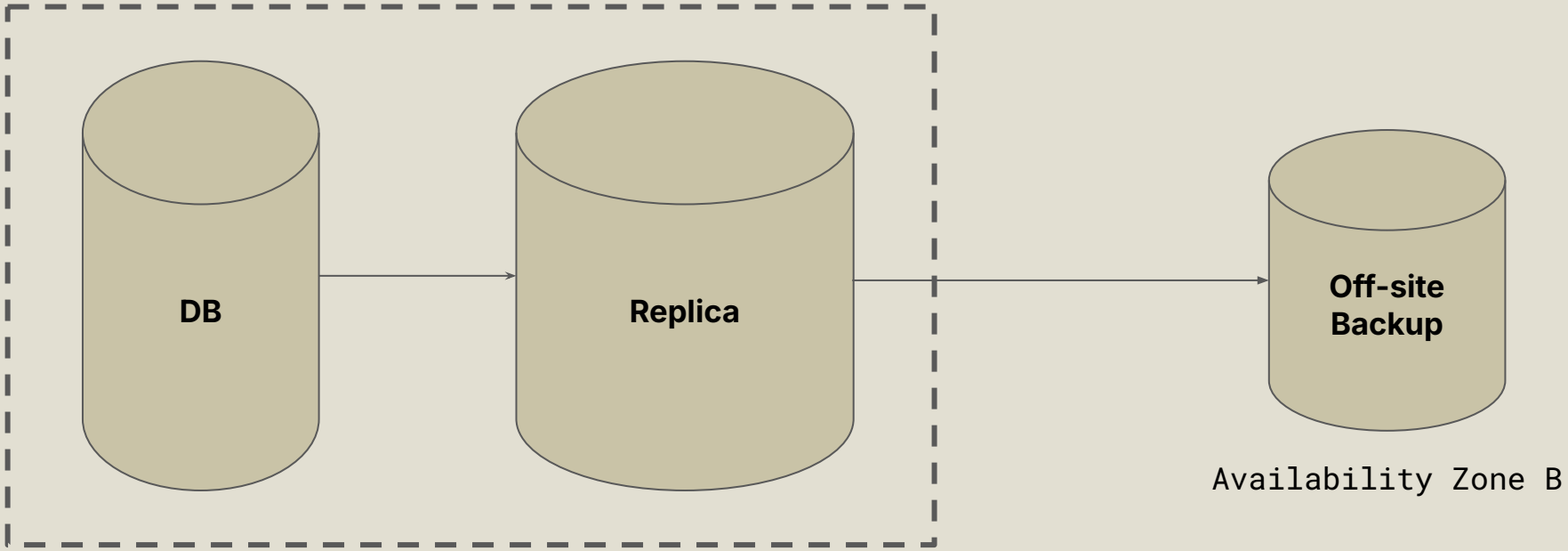
https://developer.harness.io/docs/chaos-engineering/concepts/chaos101/


DB
Replica
Service A
Service C
Service D
Service B
Deliberately make Service B go down and test what happens
Closed
Open
Half-Open



Shut down main DB - will be replica be able to take over as the main DB?

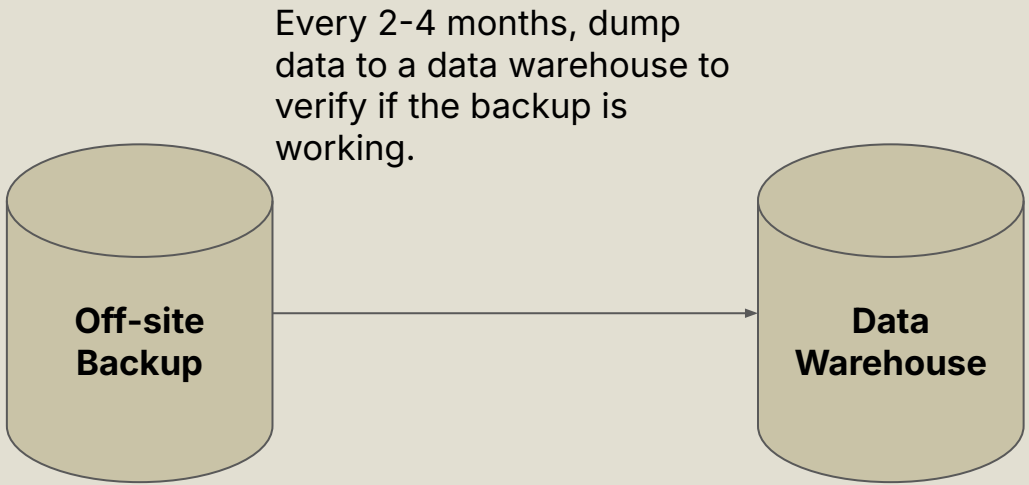
Failover Recovery



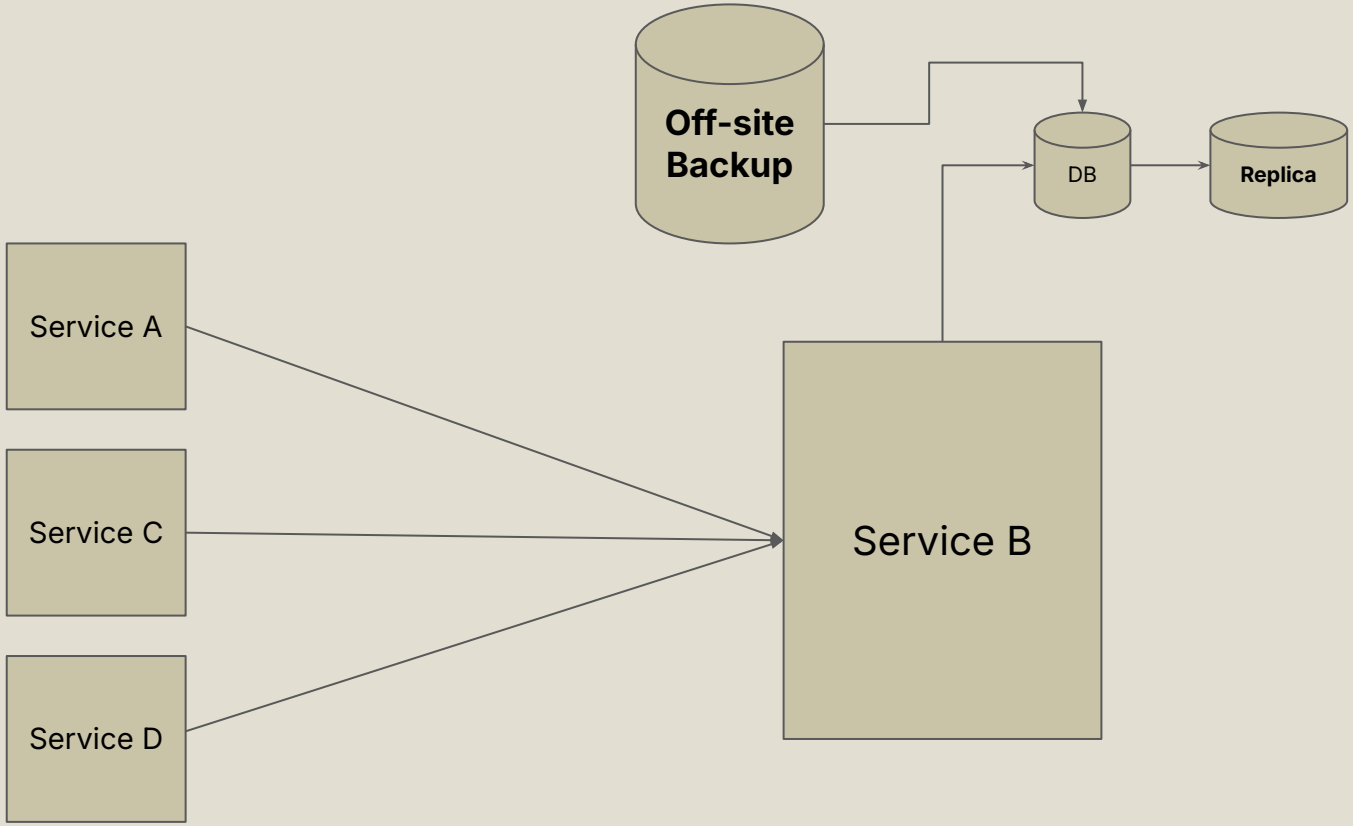
Can we launch a new DB with the off-site backup?

Is the backup usable?

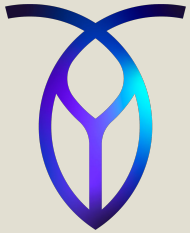
Backup Testing

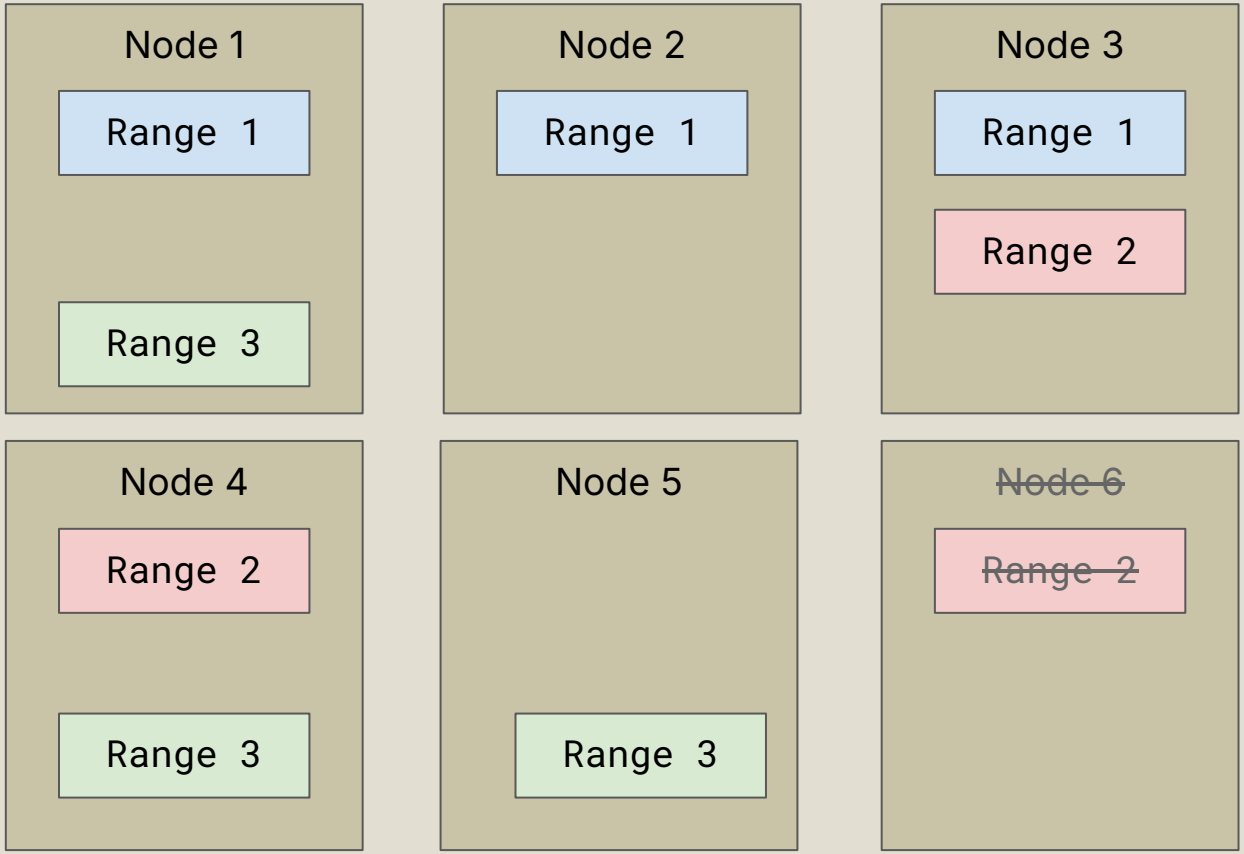


Tip 1: Test backups by dumping data to a Warehouse



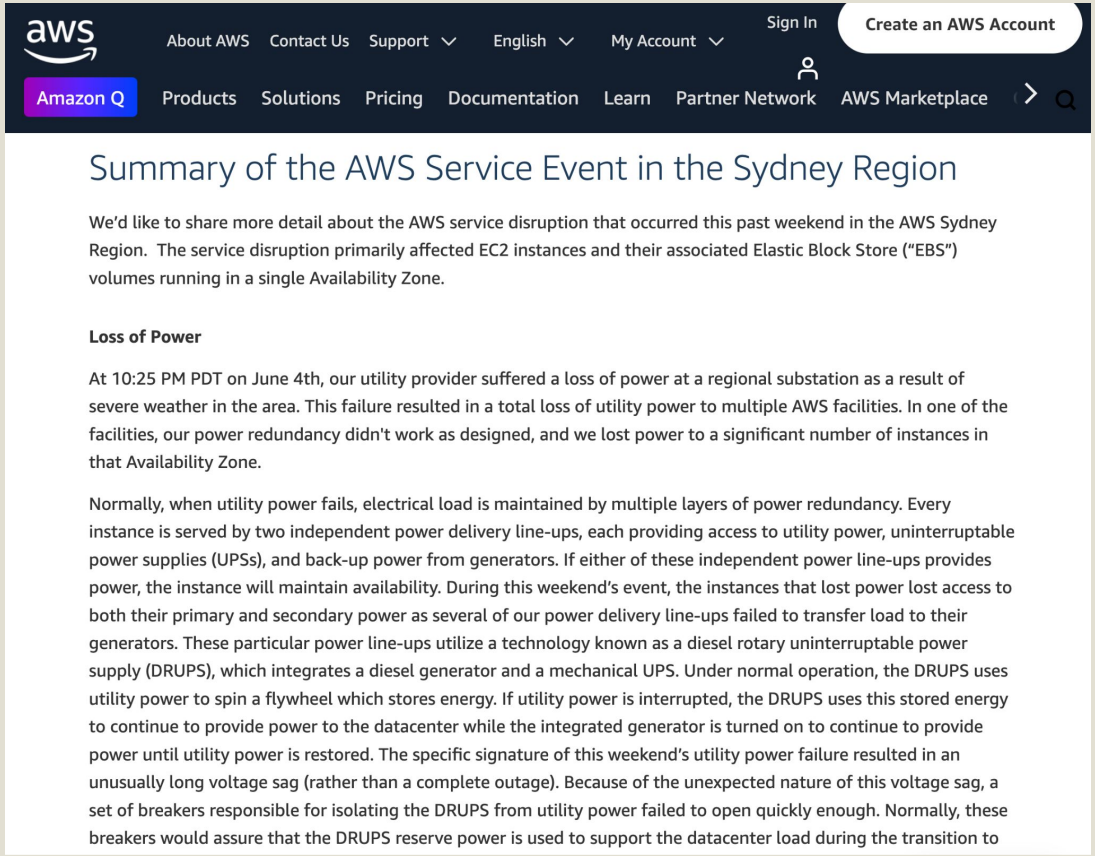
Tip 2: Test backups by restoring data on Staging





Node down? Spin up a new one!

CockroachDB will automatically rebalance the ranges



AWS Service Event in Sydney

https://aws.amazon.com/message/4372T8/

Don’t be like this during a disaster

Runbook: Database Failover

Prerequisites:

- Have access to the Database cluster
- ...

Steps:

- Check replica status by running ...
- `Run ./pg_ctl promote -D /…/data`
- `Check …`

Monitor:

- Monitor the cluster for at least 15 mins...

Note: Be sure to test the runbook periodically!

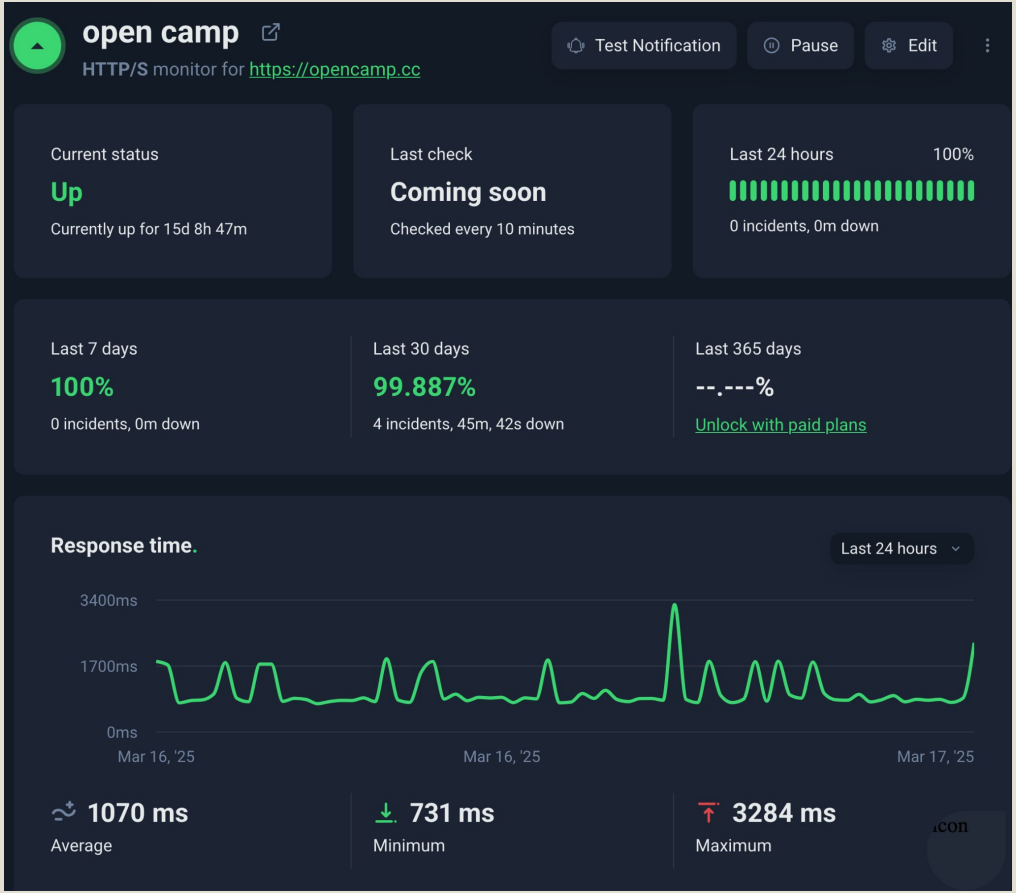
The worst thing is to have unreliable and untrusted runbooks when they are needed.

Disaster Recovery Runbook

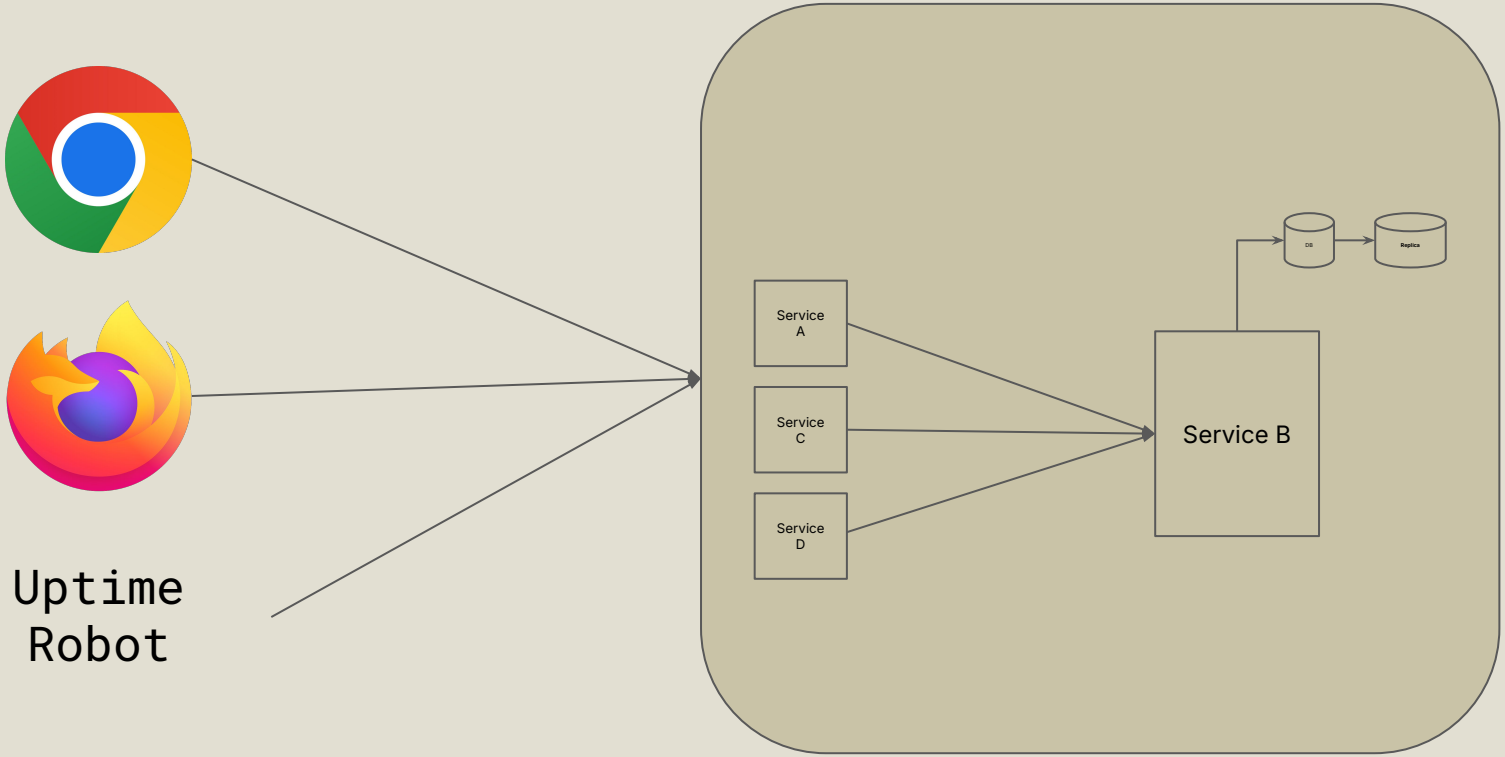
	Essential?	Risks
DB Failover	Yes	Not able to recovery quickly, affecting operational load
DB Backup Test	Yes	Backups might not be usable when they are really needed
Runbook	Depends	Messing up or not knowing what to do during an incident. May cause another incident if not careful

Summary: Proactive Checks

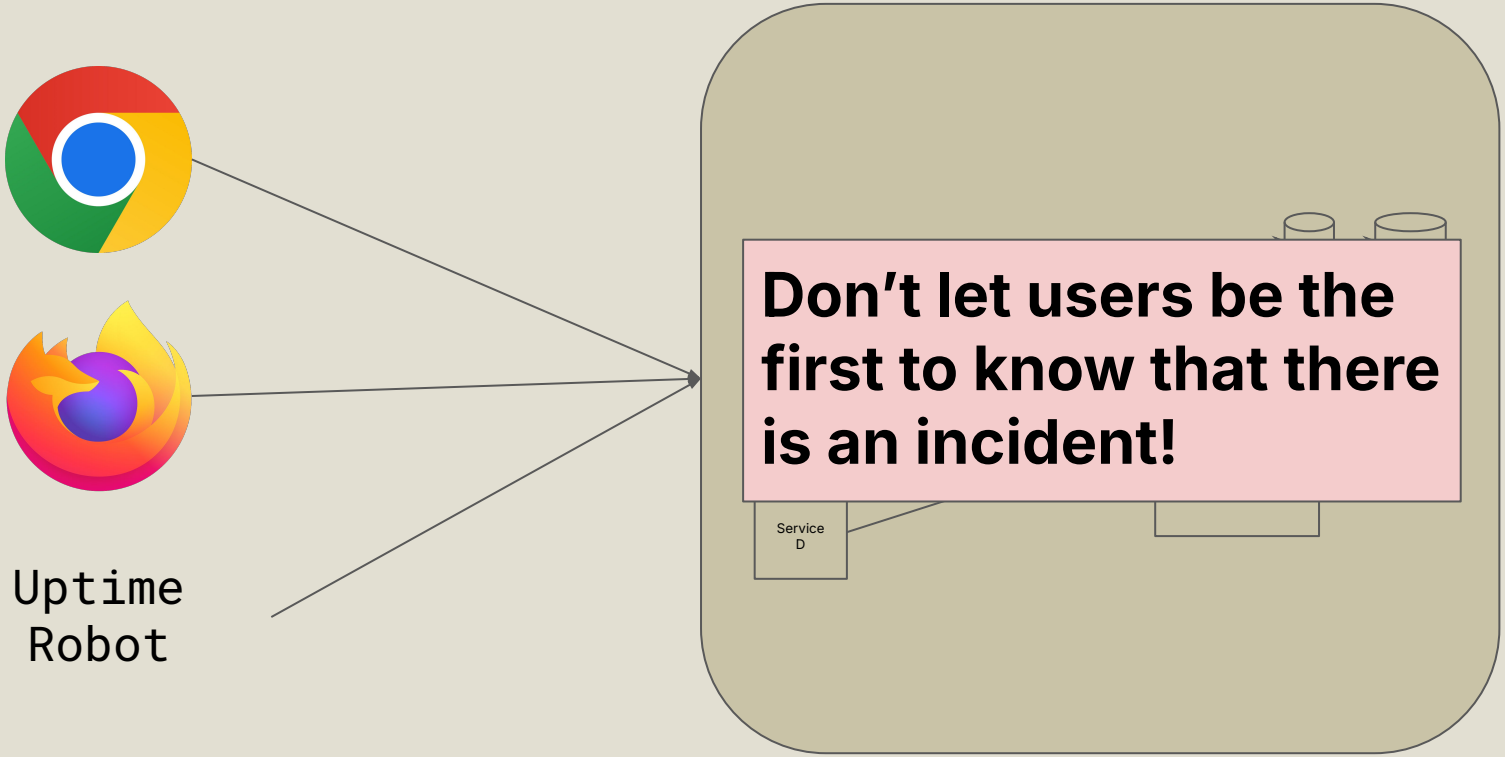
4.2.3 active surveillance



Uptime Monitoring



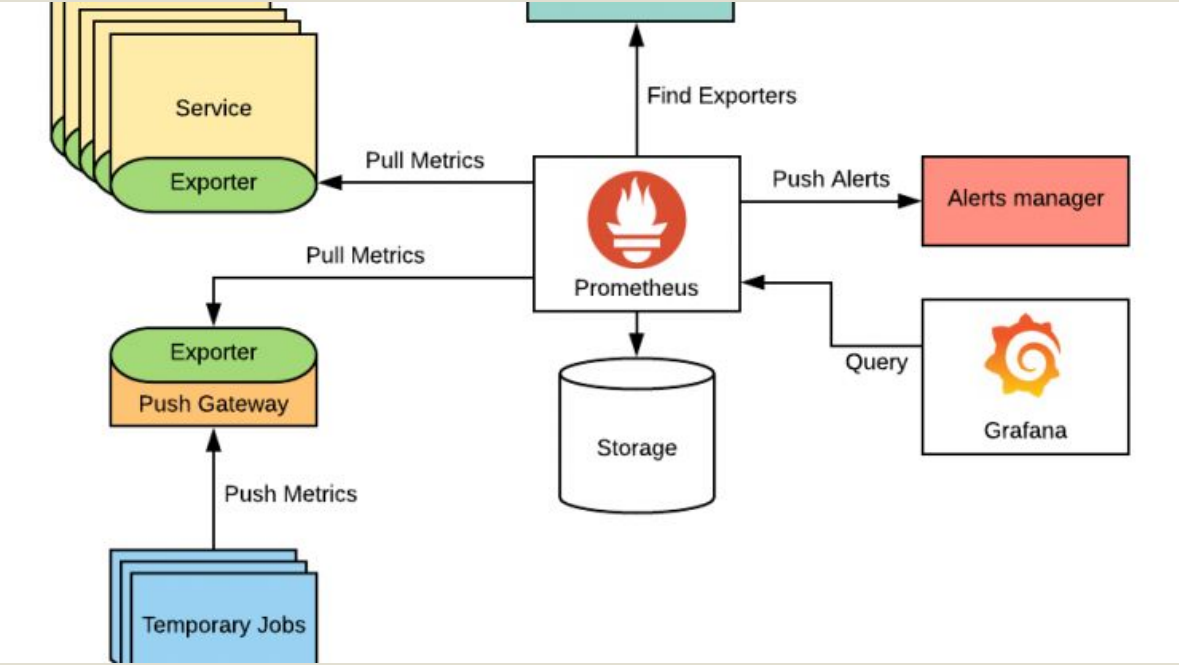
External Monitoring



External Monitoring

AWS Cloudwatch

https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/GettingStarted.html



Prometheus + Grafana Stack

https://www.devopsschool.com/blog/what-is-prometheus-and-how-it-works/

A time series database (TSDB) is a database optimized for time-stamped or time series data.

Time series data are simply measurements or events that are tracked, monitored, downsampled, and aggregated over time.

This could be server metrics, application performance monitoring, network data, sensor data, events, clicks, trades in a market, and many other types of analytics data.

Time Series DB

https://www.influxdata.com/time-series-database/

+

+

How much to invest?

Uptime Target: 99.99%, etc.

Recovery Point Objective (RPO): The maximum amount of data loss (measured by time) that an organization can tolerate.

Recovery Time Objective (RTO): The maximum length of time it should take to restore normal operations following an outage.

Three Key Metrics

Uptime Target	Downtime Allowed
99.5%	Daily: 7m 12s Weekly: 50m 24s **Monthly: 3h 37m 21s** Quarterly: 10h 52m 2.2s Yearly: 1d 19h 28m 8.8s
99.9%	Daily: 1m 26s Weekly: 10m 4.8s **Monthly: 43m 28s** Quarterly: 2h 10m 24s Yearly: 8h 41m 38s

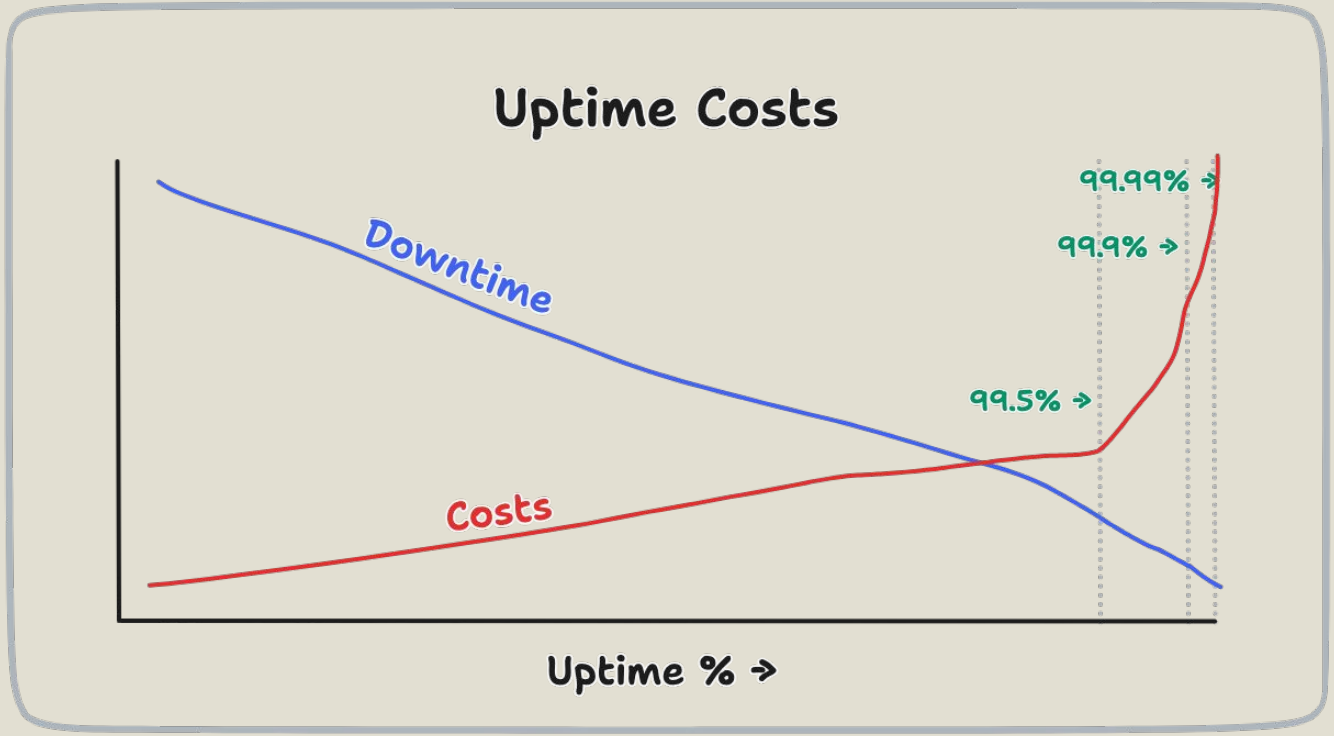
Uptime Calculator

https://uptime.is/

Uptime Target	Downtime Allowed
99.95%	Daily: 43s Weekly: 5m 2.4s **Monthly: 21m 44s** Quarterly: 1h 5m 12s Yearly: 4h 20m 49s
99.99%	Daily: 8.6s Weekly: 1m 0.48s **Monthly: 4m 21s** Quarterly: 13m 2.4s Yearly: 52m 9.8s

Uptime Calculator

https://uptime.is/



Uptime Guarantees – A Pragmatic Perspective

https://world.hey.com/itzy/uptime-guarantees-a-pragmatic-perspective-736d7ea4

“**Amazon's Service Level Agreement ("SLA") for EC2 specifies 99.5% uptime for a single machine instance**.”

“**Running two machines, one in each of two such independent availability zones, allows you to multiply the probabilities and get 99.75% uptime**. In reality, Amazon's SLA actually specifies 99.99% for this combination.”

Uptime Guarantees – A Pragmatic Perspective

https://world.hey.com/itzy/uptime-guarantees-a-pragmatic-perspective-736d7ea4

Recovery Point Objective (RPO): The maximum amount of data loss (measured by time) that an organization can tolerate.

Time to last backup: 1 hour? 1 day? 1 week?

Every organization has a different appetite for amount of data loss that can be accepted.

RPO Guidelines

Recovery Time Objective (RTO): The maximum length of time it should take to restore normal operations following an outage.

Usually related to uptime targets. In most cases, organizations want this to be as low as possible, eg. 30 mins to 1 hour.

RTO Guidelines

RFCs / Request for Comments

Used to propose, or suggest new changes that will require feedback from others / stakeholders.

Example: New Database selection

Technical Plan

Typically used to document how changes are being planned and will be executed.

Example: Database Migration process

Retrospective / Post-Mortems

When an event occurs, retrospectives and post-mortems are run to capture learnings.

Does not necessarily need to be a negative event.

Example: DB downtime retrospective

Most common types of technical docs

An incident postmortem brings teams together to take a deeper look at an incident and figure out what happened, why it happened, how the team responded, and what can be done to prevent repeat incidents and improve future responses.

Blameless postmortems do all this without any blame games.

In a blameless postmortem, it's assumed that every team and employee acted with the best intentions based on the information they had at the time. Instead of identifying—and punishing—whoever screwed up, **blameless postmortems focus on improving performance moving forward.**

How to run a blameless postmortem

https://www.atlassian.com/incident-management/postmortem/blameless

	Essential?	Risks
External Monitoring	Yes	Sometimes internal monitoring fails, so it's better to have a failsafe
Internal Monitoring	Yes	Essential to know where things are failing
Uptime + RPO + RTO objectives	Depends	Not all teams or organizations need this right away. But it's good to have some uptime objectives set up

Summary: Active Surveillance

1. **It’s the cheapest to design and account for failures during design phase.** After that, the cost increases dramatically as you might need to redesign, or deal with downtimes.

2. **Proactive methods such as Chaos Engineering and Disaster Recovery tests are best done periodically with runbooks**. Never wait until an actual disaster to test them.

3. Monitoring and Observability are crucial to meet RTO and RPO objectives. More about them in W6.

W4 Summary: Handling Failures

p.s.
Current Trends

What's the difference between observability and monitoring?

In DevOps, observability and monitoring are two distinct data-based processes. You use them to successfully maintain and manage the health and performance of distributed microservice architectures and their infrastructure. Distributed systems work by exchanging data between tens to hundreds or thousands of different components.

Monitoring is the process of collecting data and generating reports on different metrics that define system health. Observability is a more investigative approach. It looks closely at distributed system component interactions and data collected by monitoring to find the root cause of issues. It includes activities like trace path analysis, a process that follows the path of a request through the system to identify integration failures. Monitoring collects data on individual components, and observability looks at the distributed system as a whole.

Read about DevOps

Monitoring vs Observability

https://aws.amazon.com/compare/the-difference-between-monitoring-and-observability/

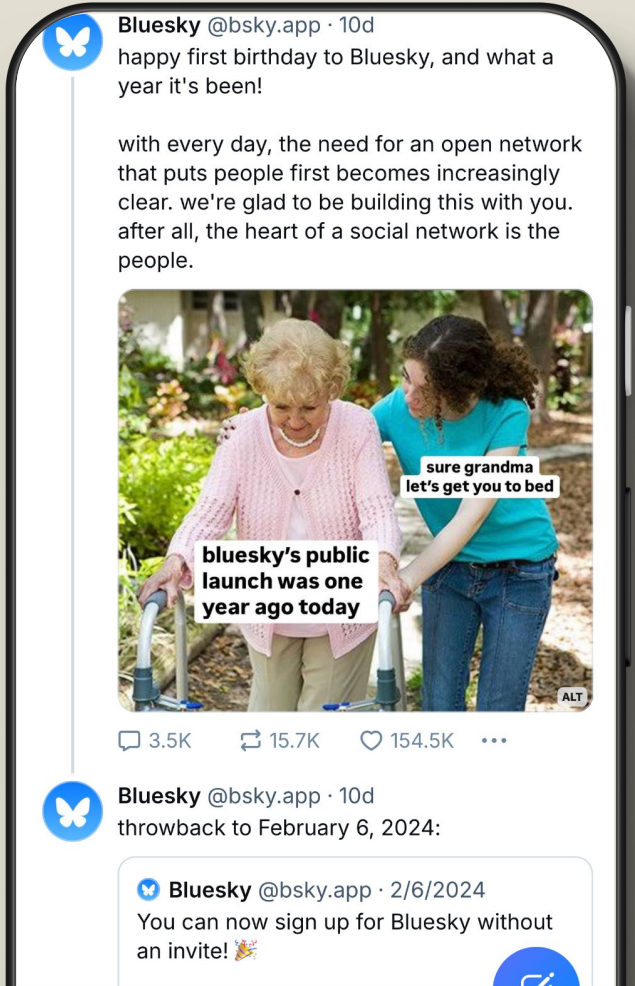
"It grounded planes, shut down banks, and took radio stations off the air. It will likely cost some of the world's biggest companies $5 billion or more in damages."

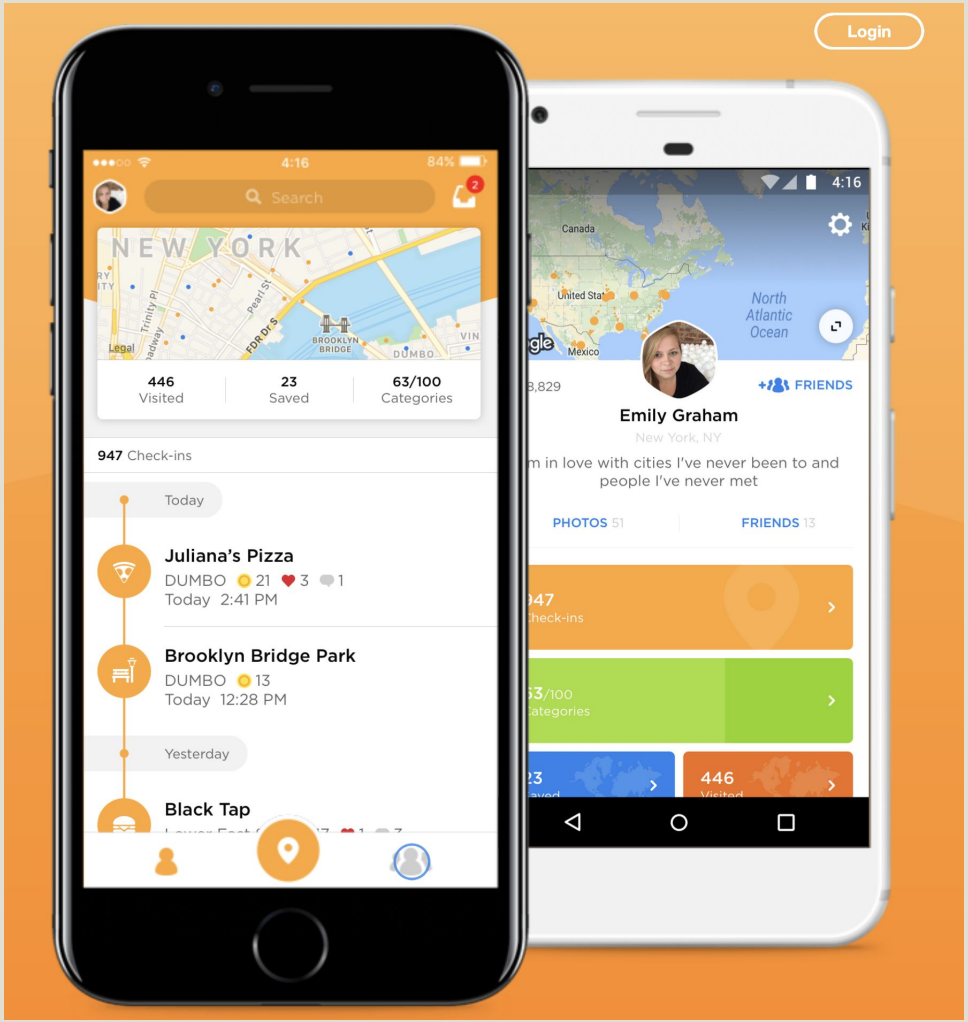
CrowdStrike Disaster

https://leaddev.com/software-quality/crowdstrike-disaster-lesson-about-testing

Assignment

Extend a Distributed Social Media Platform




Login
4:16
84%
Search
NEW YORK
446
Visited
23
Saved
63/100
Categories
947 Check-ins
Today
Juliana's Pizza
DUMBO 21 3 1
Today 2:41 PM
Brooklyn Bridge Park
DUMBO 13
Today 12:28 PM
Yesterday
Black Tap
4:16
Canada
North Atlantic Ocean
Mexico
FRIENDS
Emily Graham
New York, NY
PHOTOS 51
FRIENDS 13
446

Assignment

W1 → Distributed Social Media Research
W2 → Review W1 + Build PoC
W3 → Design 1st Draft of System + Sharing
W4 → Implementation
W5 → Implementation (Queues)
W6 → Implementation (Load Testing)
W7 → Technical Retrospective
W8 → Complete System Implementation