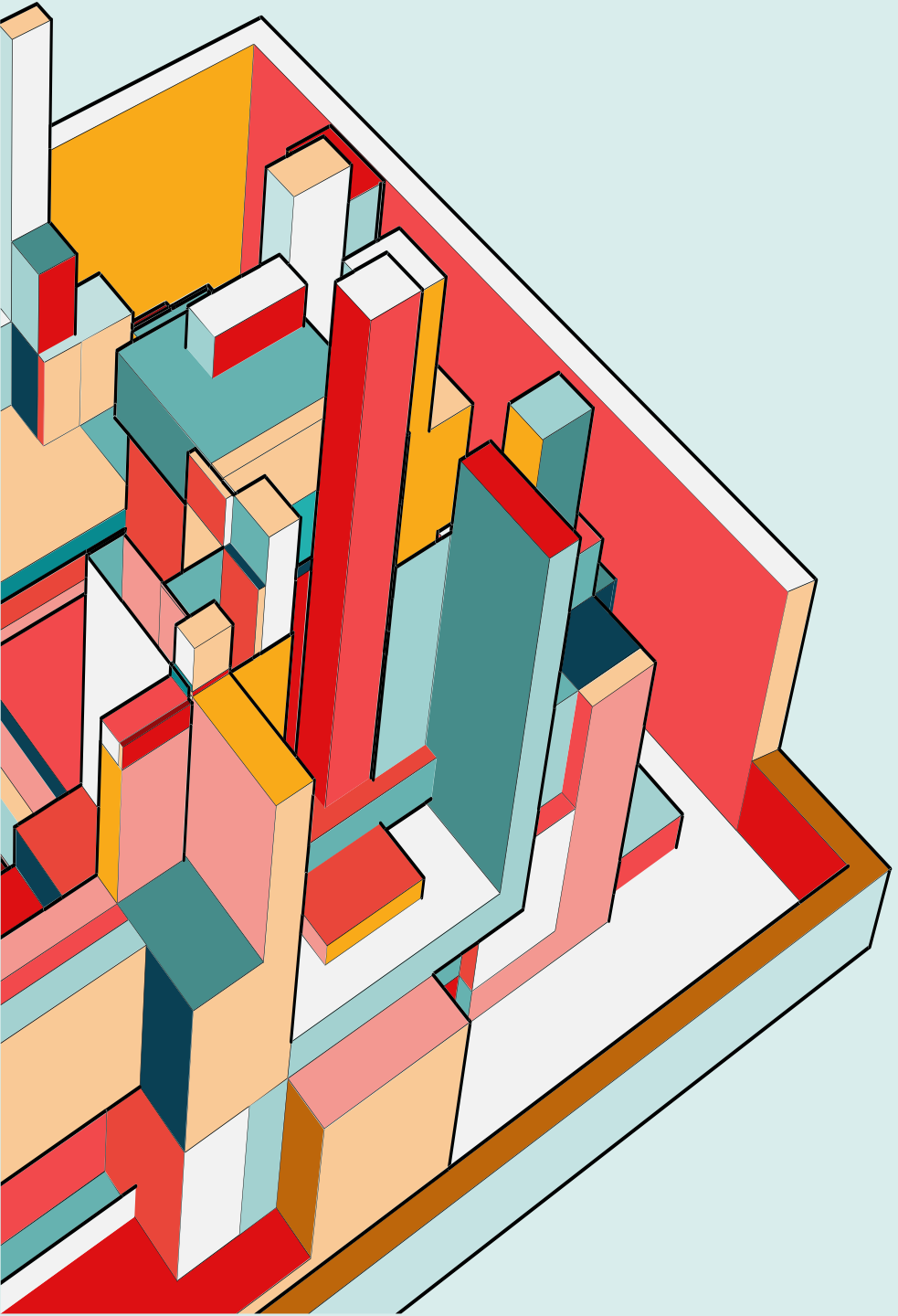
UNBREAKABLE BY DESIGN: THE SECRET TECH OF ANTIFRAGILE SOFTWARE

IRYNA DOHNDORF

15.10.2026

karakun



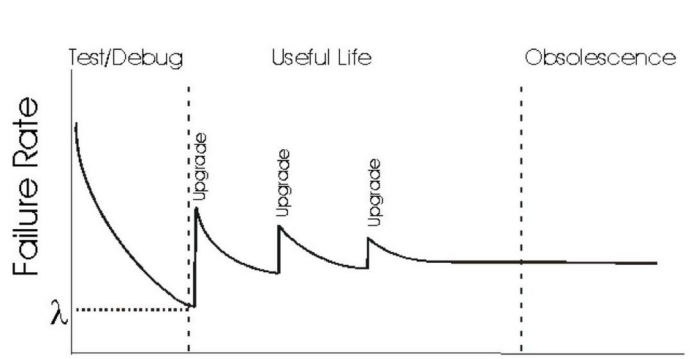
ALL THE THINGS WE'VE LEARNT ABOUT DEPENDABLE COMPUTER SYSTEMS...

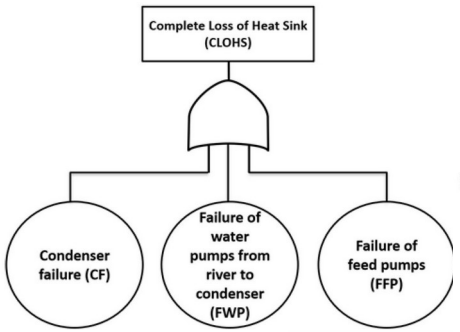
How We Benefit from Failures ?

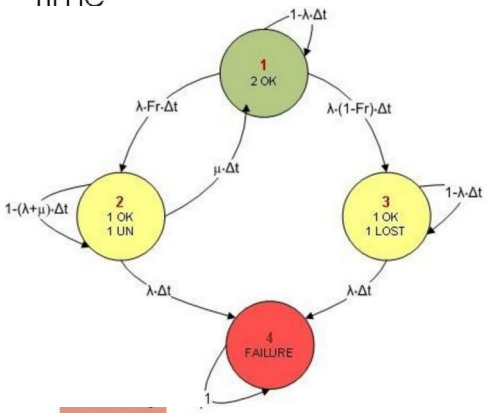
State of the Art

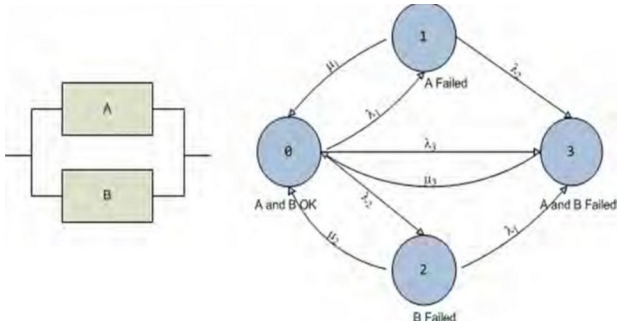
SYSTEM DEPENDABILITY MODELS

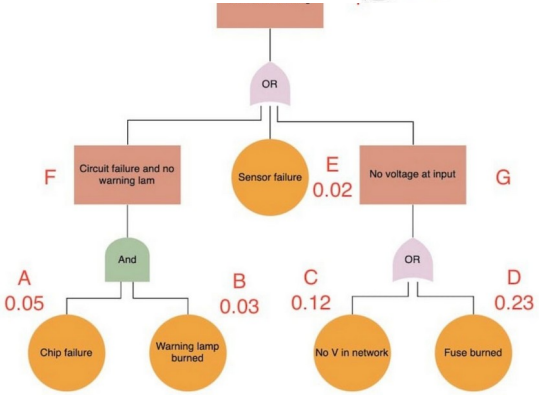
- Faults and failures
- Reliability, Safety and Securit
- System models, fault models
- Fault tree analysis
- Fault tolerance methods

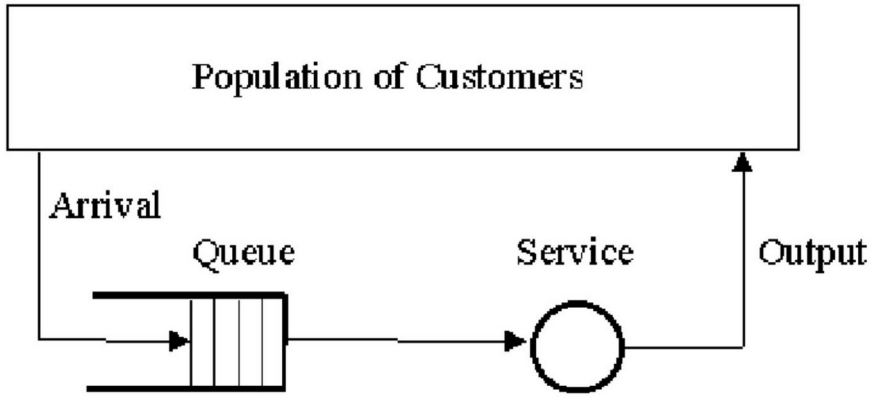




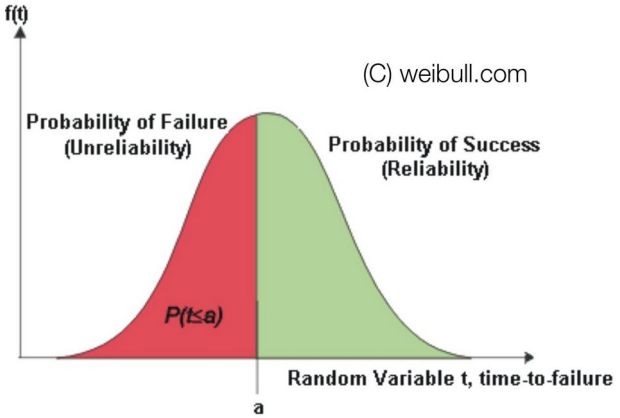








Single Server Queuing System



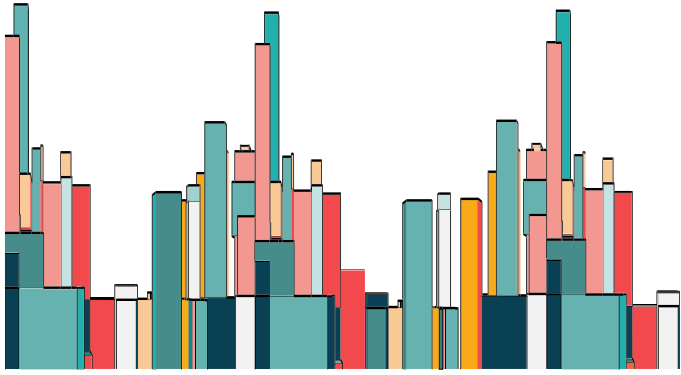
Unbreakable by design: the secret tech of antifragile software
Iryna Dohndorf - Karakun

SOFWARE DEVELOPMENT:
HOW DOES IT LOOK LIKE IN EVERYDAY PRACTICE ?

- Challenges to test as, e.g., platforms with many national service
- Unacceptable testing, even in all phases of the development process

- Inadequate testing / analysis tools
- Budgetary constraints
- Time pressure
- Knowledge gaps
- Inefficient communication

- Many bugs were not detected before the code went to production
- Even worse: not tested bug fixes go to production

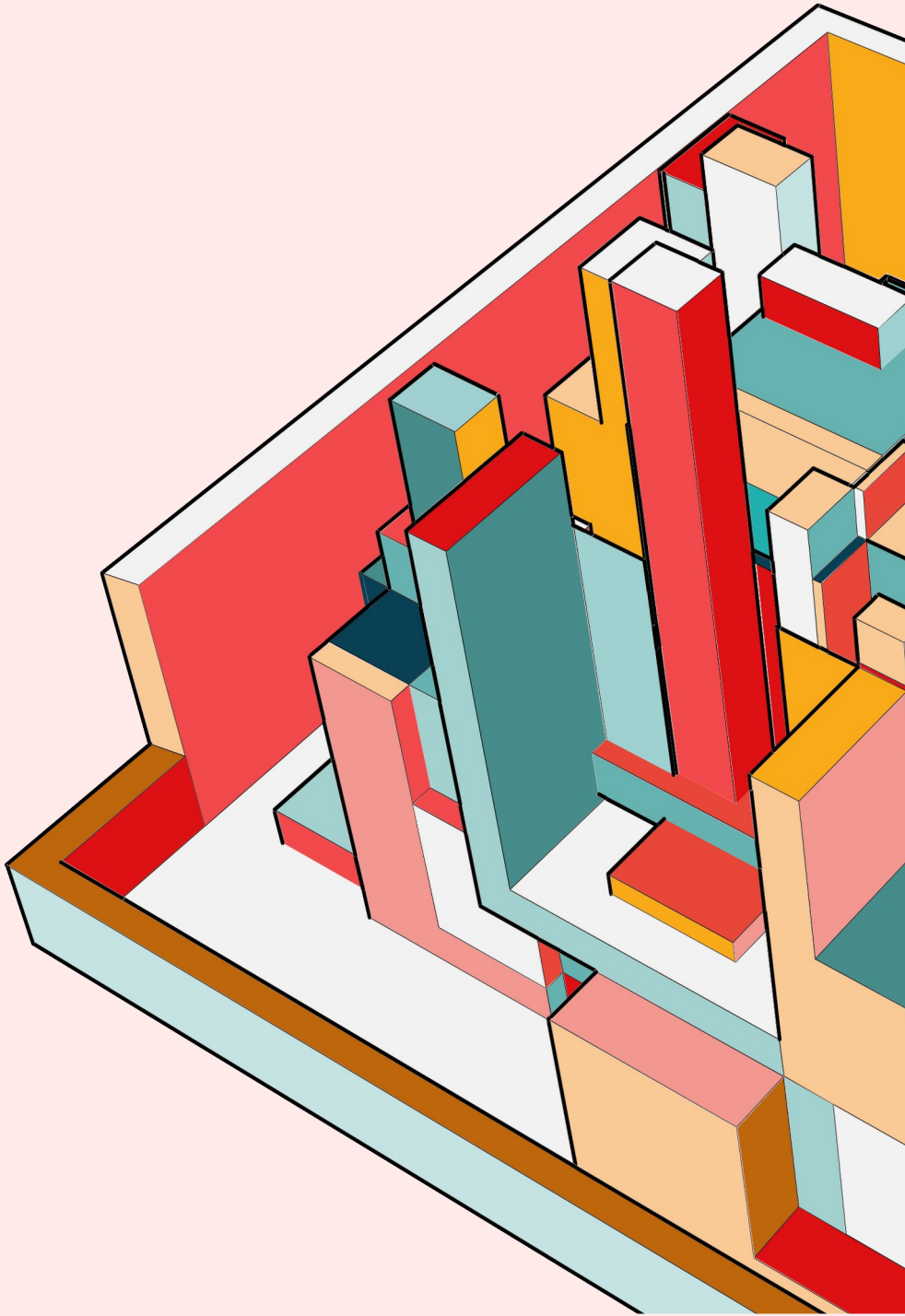


Basel One

Unbreakable by design: the secret tech of antifragile software
Iryna Dohndorf - Karakun

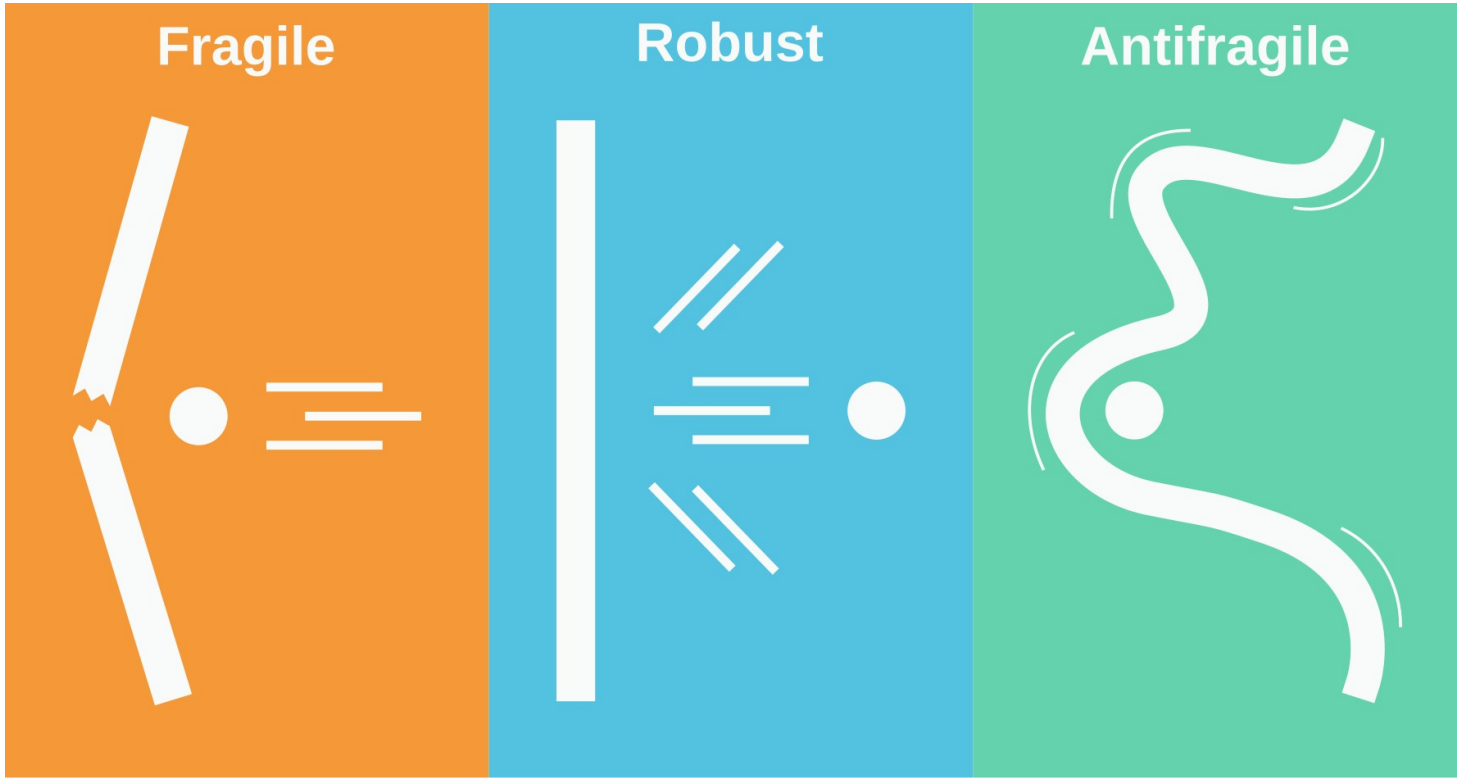
WHAT ARE ANTIFRAGILE SYSTEMS ?

Definitions



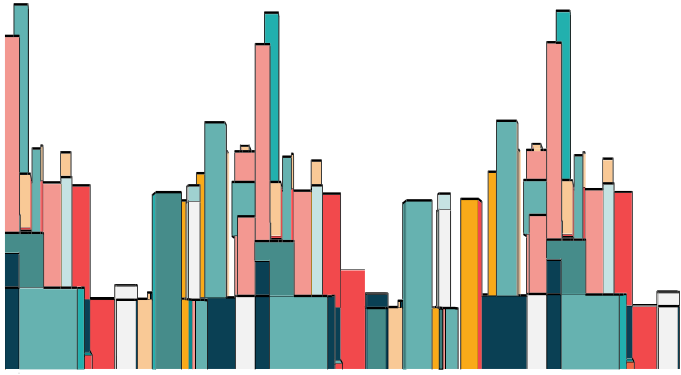


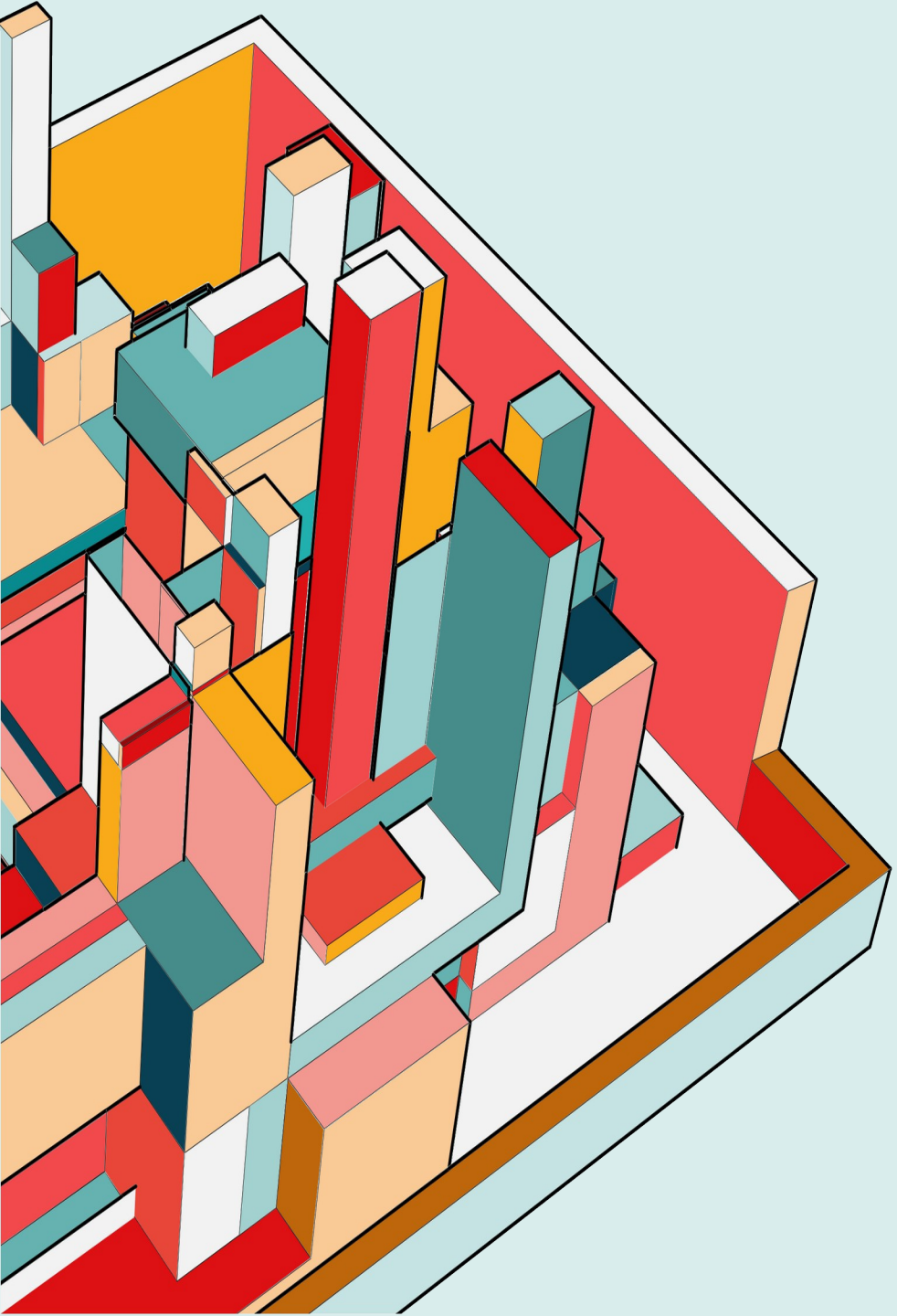
ANTIFRAGILE SOFTWARE SYSTEMS



Source: Philip Vervloesem: Thriving in uncertainty:
the promise of an antifragile supply chain

Unbreakable by design: the secret tech of antifragile software
Iryna Dohndorf - Karakun





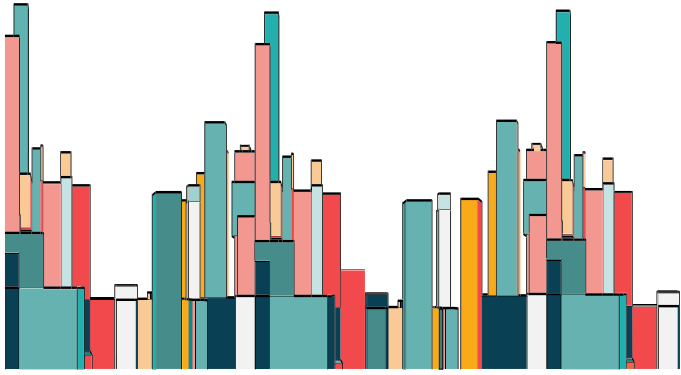
EXAMPLE OF ANTIFRAGILE TECHNIQUES

LLM Prompt Injection Tool

EXAMPLE: AUGUSTUS – LLM PROMPT INJECTION / PEN TESTING TOOL

- Open Source Vulnerability Scanner by Praetorian. Single Go binary, 210+ probes, 28 LLM providers
  - https://www.praetorian.com/blog/introducing-augustus-open-source-llm-prompt-injection/ and https://github.com/praetorian-inc/augustus
  - https://www.praetorian.com/blog/introducing-julius-open-source-llm-service-fingerprinting/ and https://github.com/praetorian-inc/julius





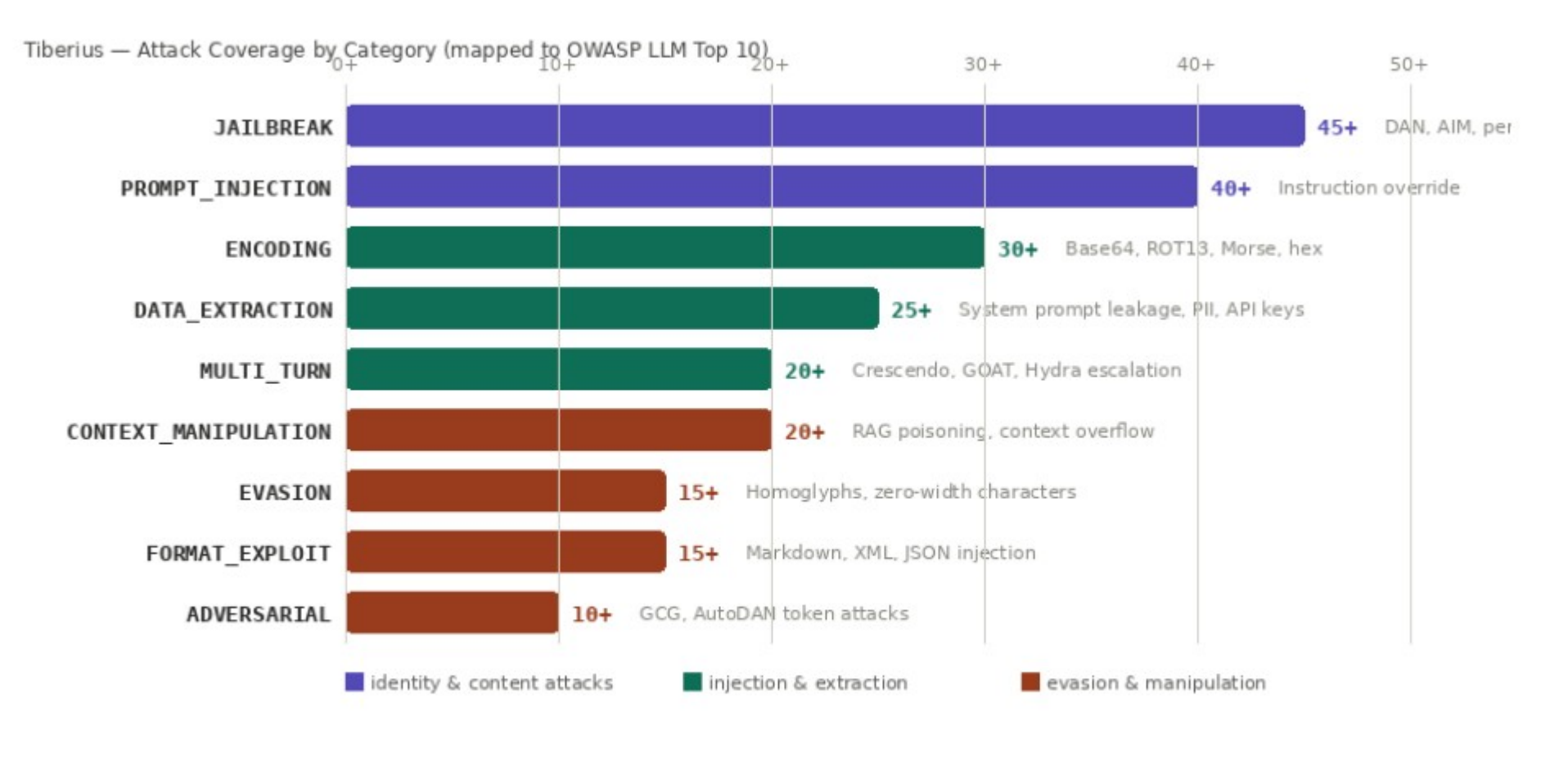


TIBERIUS

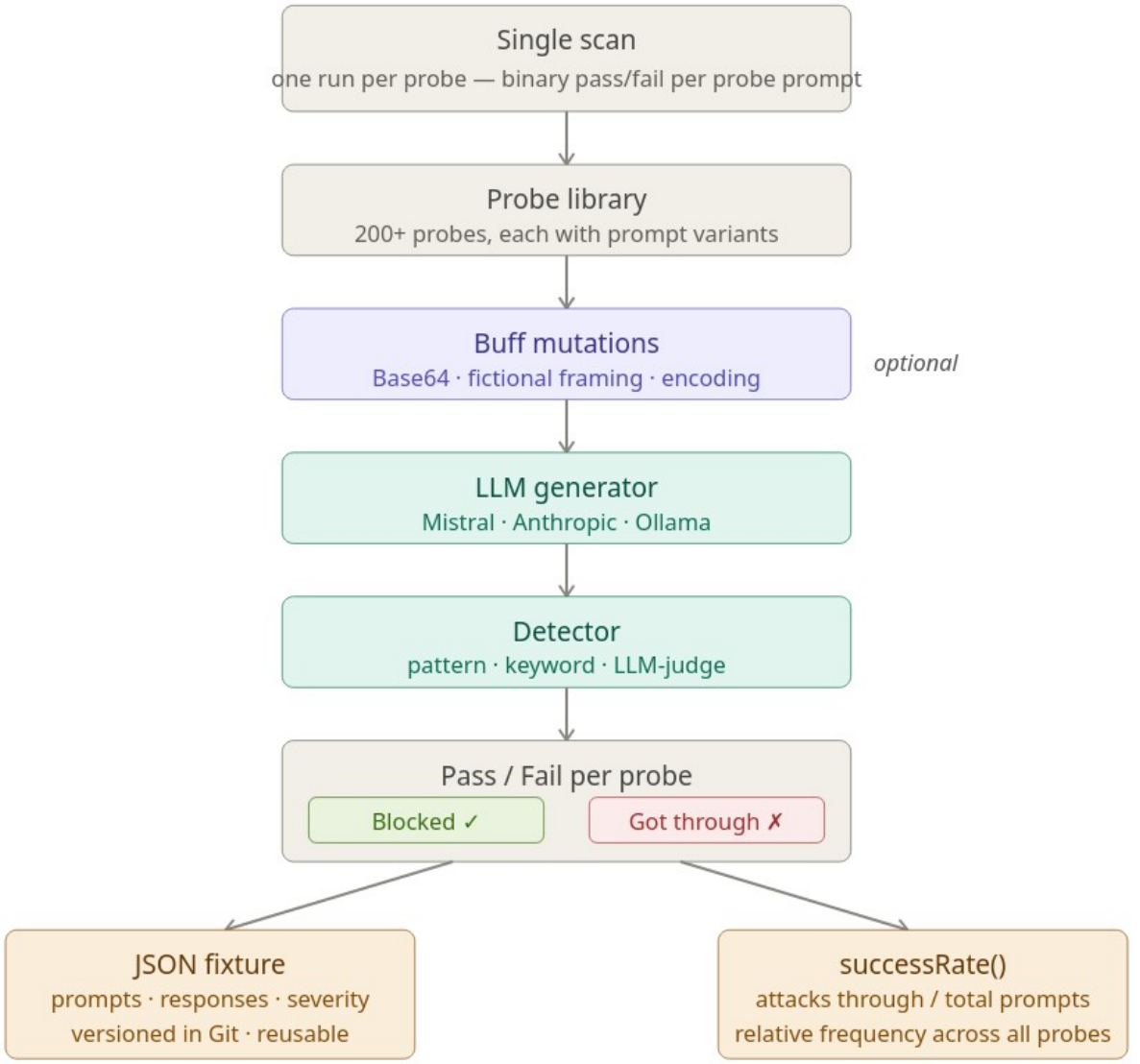
security & safety testing for LLM apps — in Java & JUnit 5

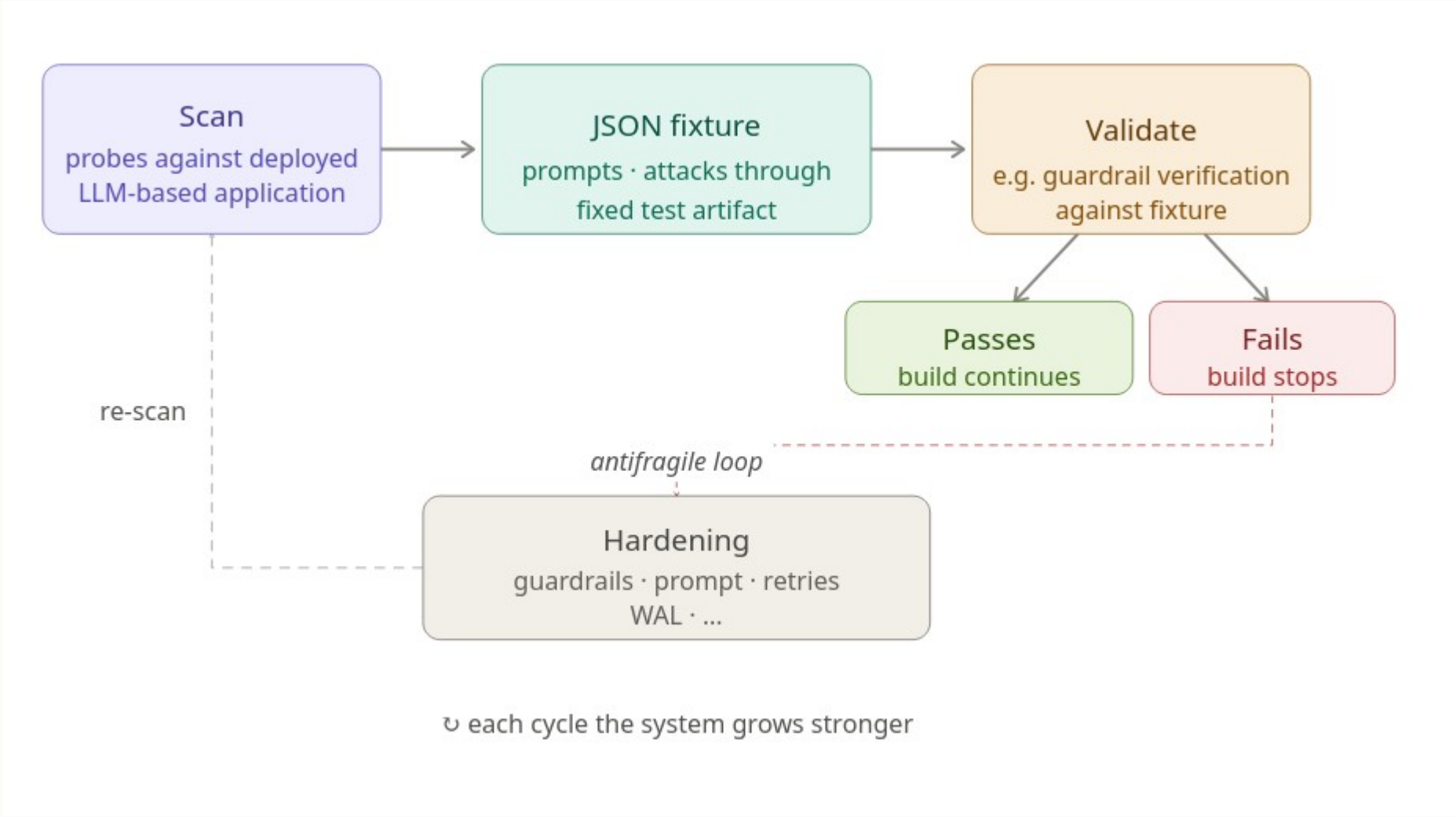
"Augustus had his guard. So does your LLM."

Github: https://github.com/tiberius-security/tiberius

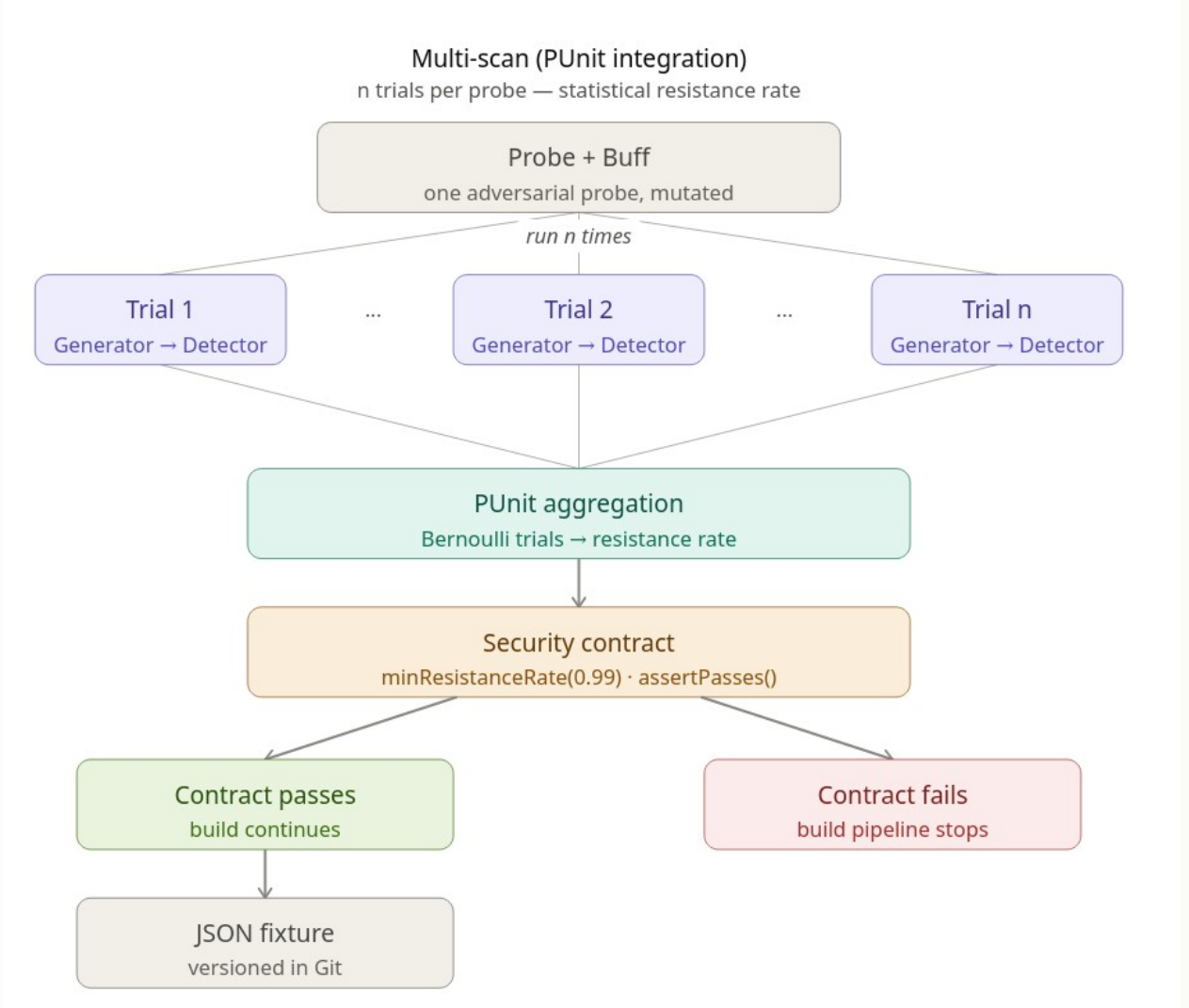


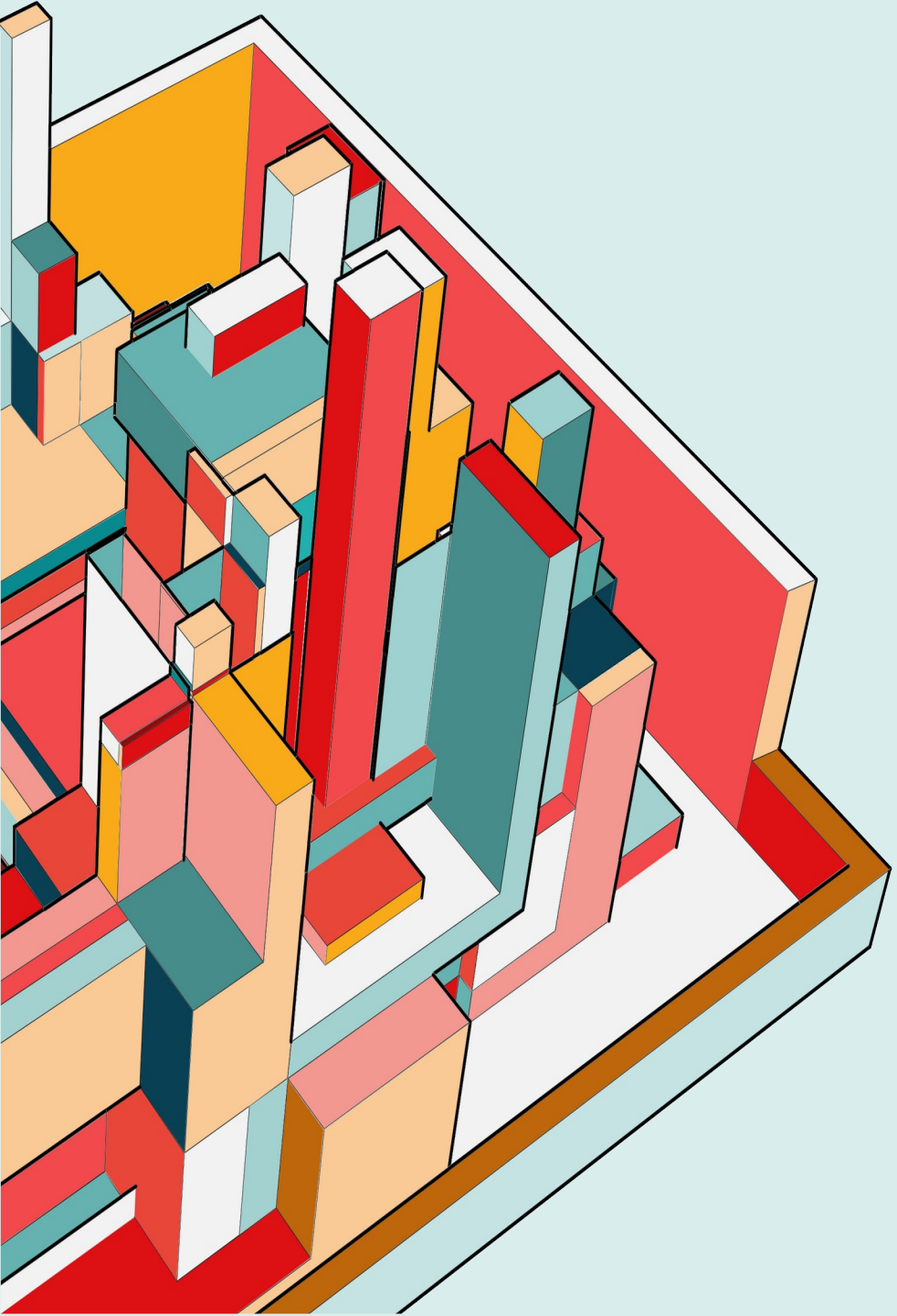
ATTACK COVERAGE — TIBERIUS


Single scan
one run per probe — binary pass/fail per probe prompt
Probe library
200+ probes, each with prompt variants
Buff mutations
Base64 · fictional framing · encoding
optional
LLM generator
Mistral · Anthropic · Ollama
Detector
pattern · keyword · LLM-judge
Pass / Fail per probe
Blocked ✓
Got through ✗
JSON fixture
prompts · responses · severity
versioned in Git · reusable
successRate()
attacks through / total prompts
relative frequency across all probes



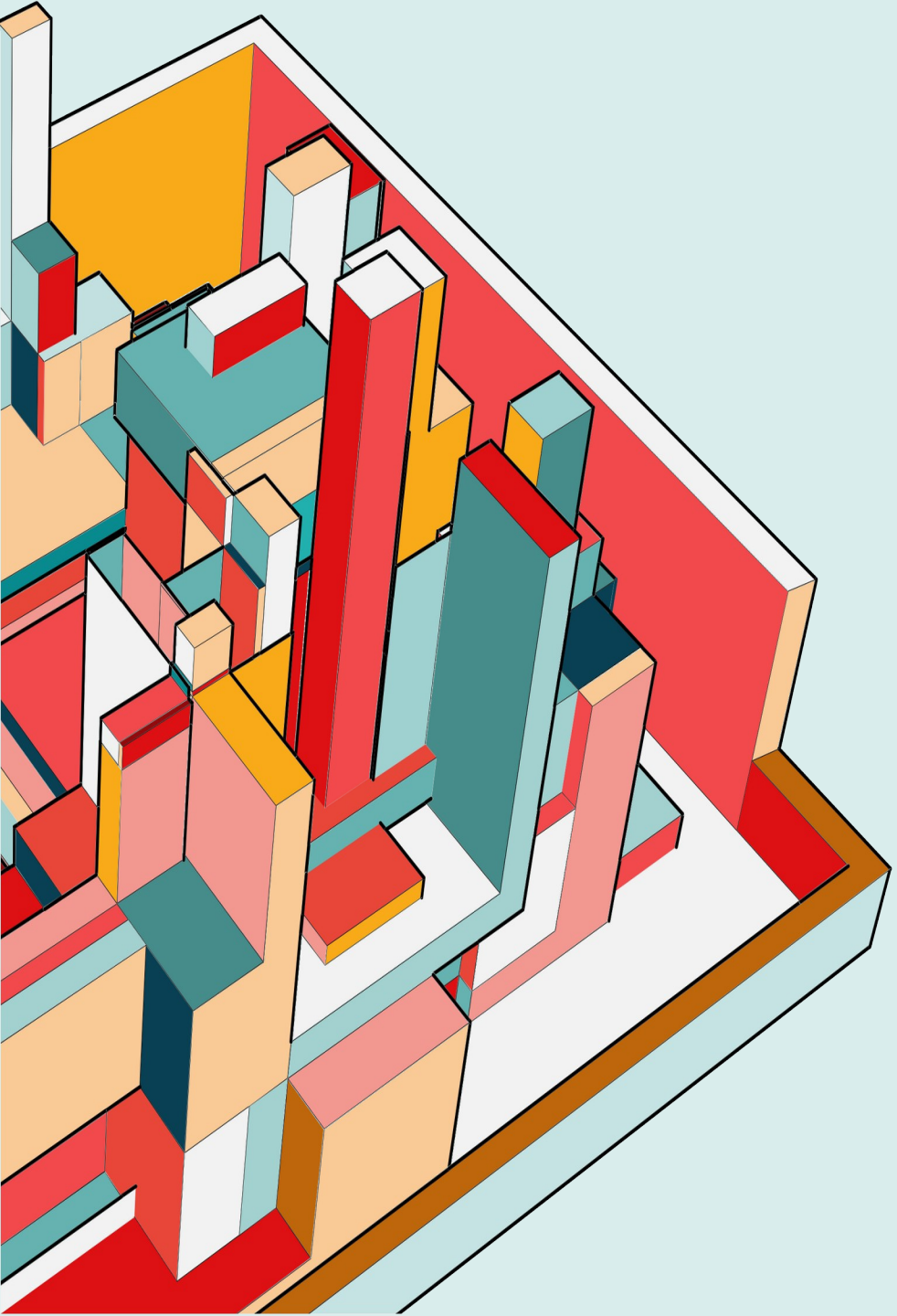
SCAN – FIXTURE - VALIDATE


Multi-scan (PUnit integration)
n trials per probe — statistical resistance rate
Probe + Buff
one adversarial probe, mutated
run n times
Trial 1
Generator → Detector
...
Trial 2
Generator → Detector
...
Trial n
Generator → Detector
PUnit aggregation
Bernoulli trials → resistance rate
Security contract
minResistanceRate(0.99) · assertPasses()
Contract passes
build continues
Contract fails
build pipeline stops
JSON fixture
versioned in Git



DEMO

LLM Prompt Injection Tool

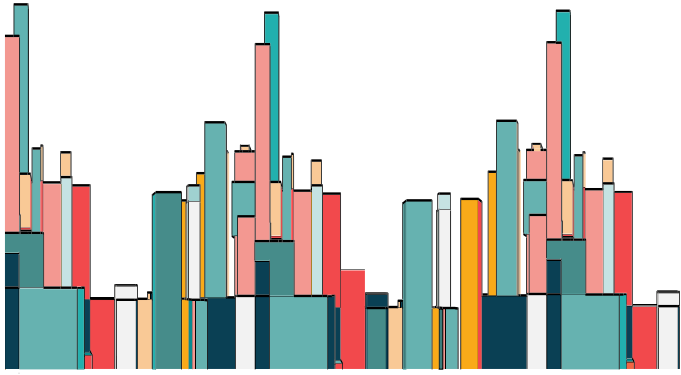


EXAMPLE OF ANTIFRAGILE TECHNIQUES

Resilient Data Platform with Write-Ahead Log at Netflix

NETFLIX RESILIENT DATA PLATFORM: CHALLENGES

- Accidential data loss and data corruption in DBs
- Network errors, downstream service outages
- System entropy across different datastores (e.g., writing to DB and Elasticsearch)
- The multiple partition update problem (e.g., building secondary indices on top of a NoSQL database)
- Data replication (in-region and across regions)
- Reliable retry mechanisms for real time data pipeline (at scale)



NETFLIX WRITE-AHEAD LOG SYSTEM ARCHITECTURE

- Multi-layer defense: WAL + Cache + Database + Replication

- Antifragile aspects
  - 1. No single point of failure: Every layer can fail independently
  - 2. Automatic learning: Kafka + Data Analysis. Each failure potentially improves the system
  - 3. Chaos Engineering: Intentional failures make the system stronger

```
User Request
    ↓
[EVCache Layer - Redis/Memcached] ← Fast reads (microseconds)
    ↓ (cache miss)
[Write-Ahead Log (WAL)] ← All mutations logged
    ↓
[Cassandra Database] ← Primary data store
    ↓
[Multi-Region Replication] ← Disaster recovery
```

How They Work Together - Simple Example:

Let's say you're watching Netflix and update your profile name from "John

```
Step 1: Request hits Layer 1 (Cache)
   → Cache updates immediately
   → You see "Johnny" right away! ⚡

Step 2: Request goes to Layer 2 (WAL)
   → Logs: "Change John → Johnny at 10:30:15 AM"
   → Stored safely ✅

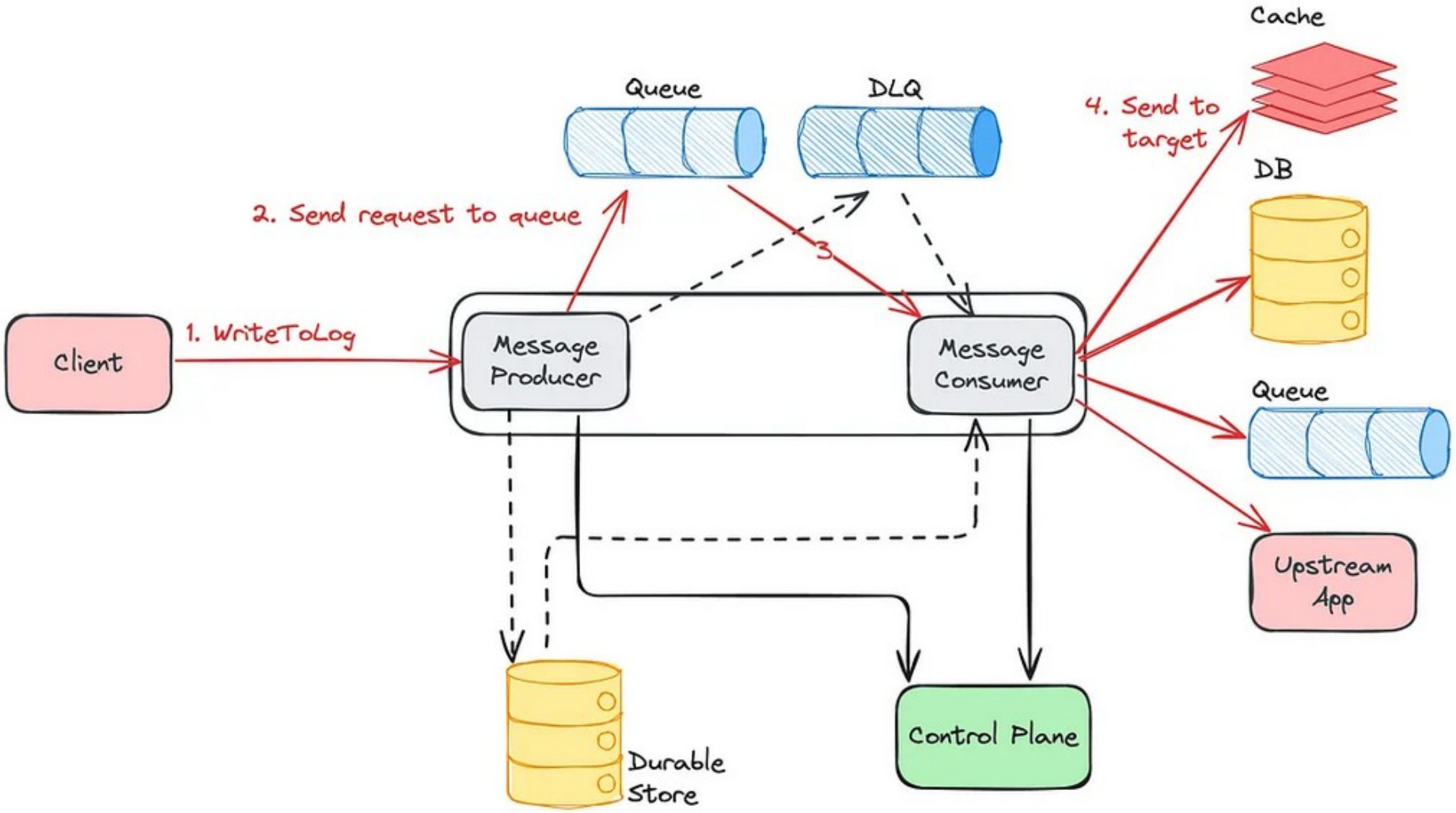
Step 3: Request goes to Layer 3 (Cassandra)
   → Writes to 3 servers in same region
   → Confirmed ✅

Step 4: Background process (Layer 4)
   → Copies to US-West region
   → Copies to EU region
   → Now data is globally safe ✅
```

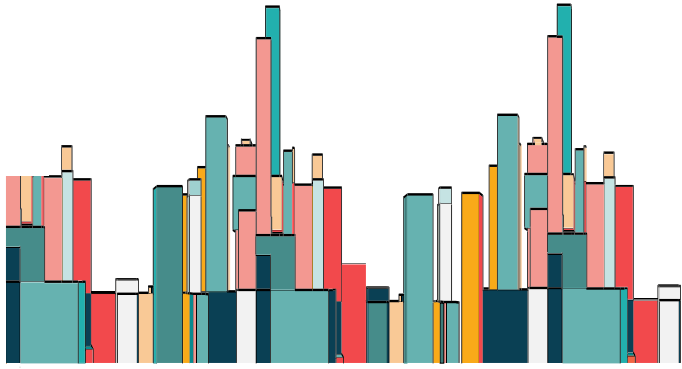
If database crashes at Step 3:

- ✅ Cache (Layer 1) still shows "Johnny"
- ✅ WAL (Layer 2) has the change recorded
- ✅ When DB recovers, WAL replays the change
- ✅ You never noticed anything was wrong!

Source: https://netflixtechblog.com/building-a-resilient-data-platform-with-write-ahead-log-at-netflix-127b6712359a

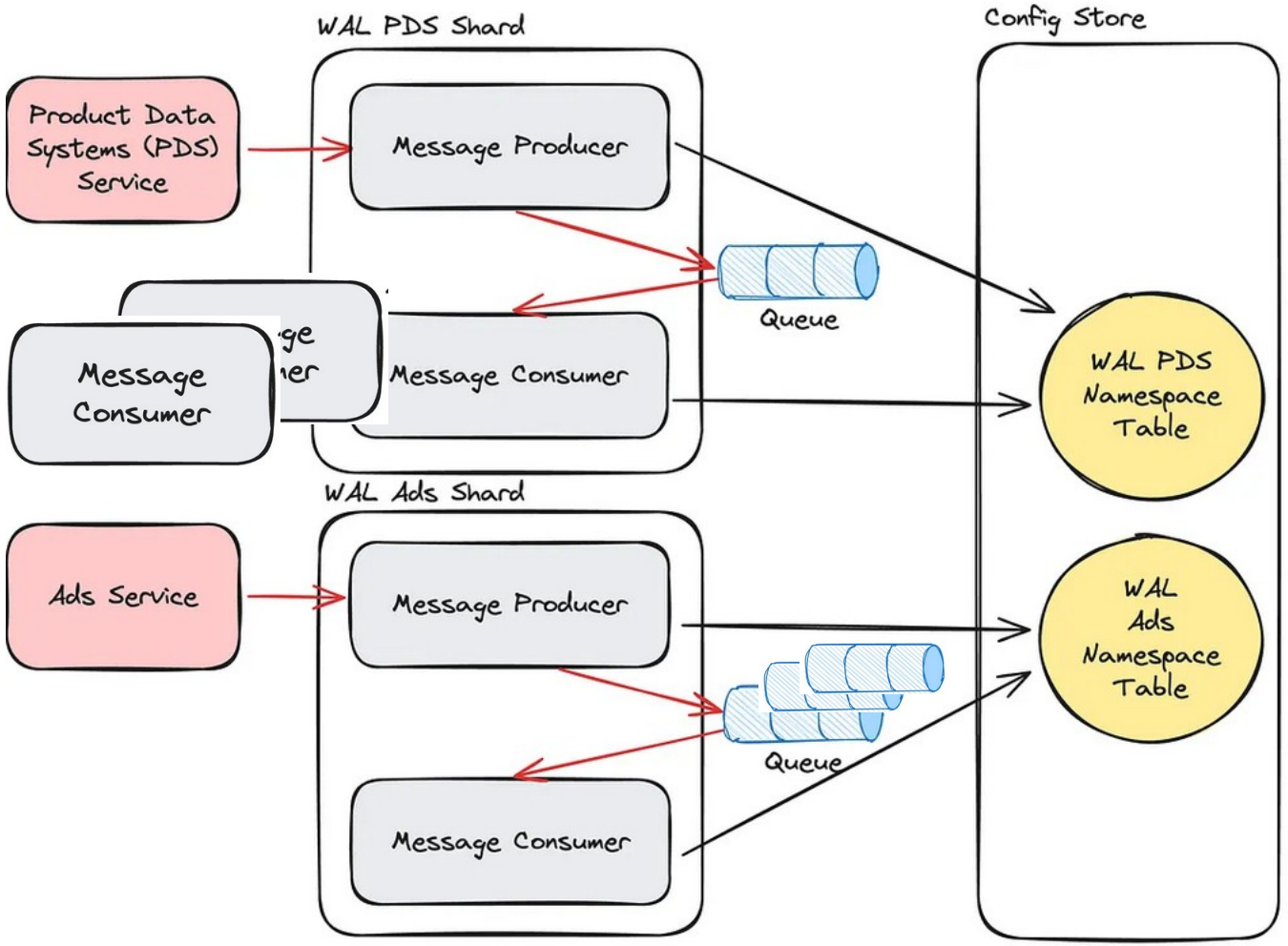


Architecture of WAL

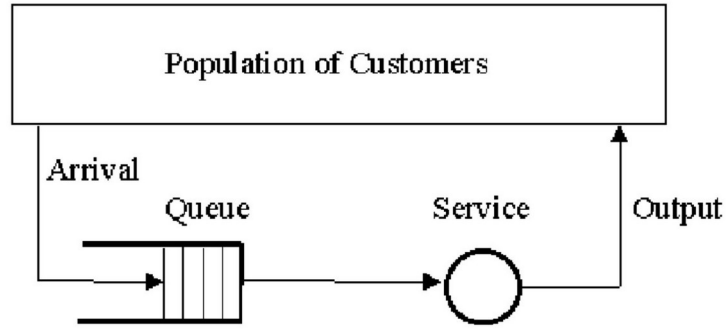


Source: https://netflixtechblog.com/building-a-resilient-data-platform-with-write-ahead-log-at-netflix-127b6712359a

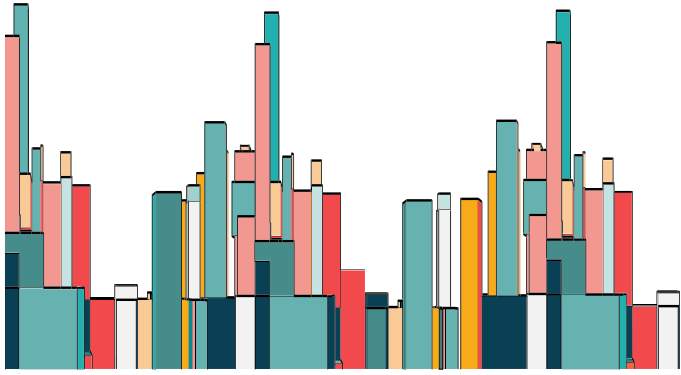
ANTIFRAGILE TECHNOLOGIES



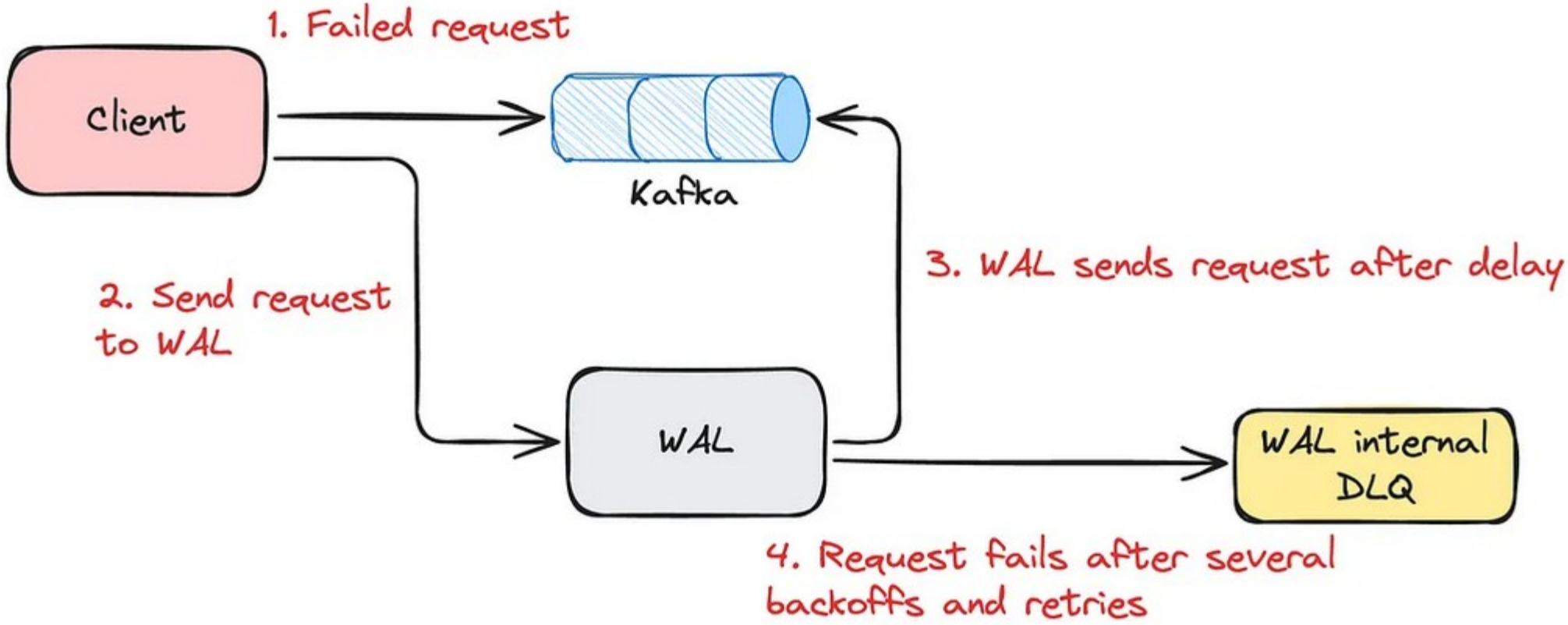
Deployment model of WAL



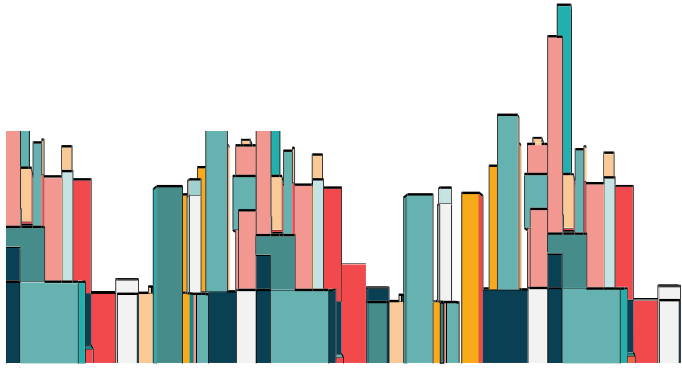
Single Server Queuing System



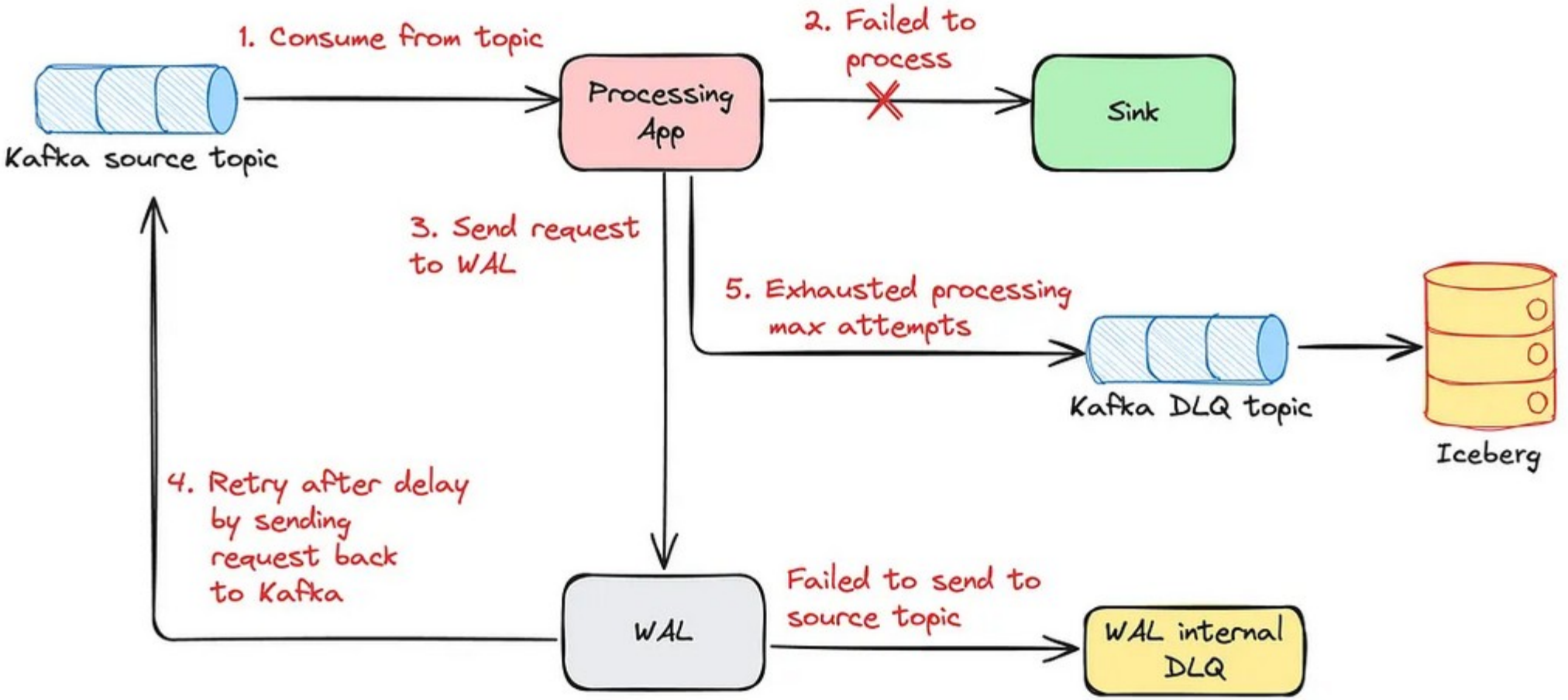
Source: https://netflixtechblog.com/building-a-resilient-data-platform-with-write-ahead-log-at-netflix-127b6712359a



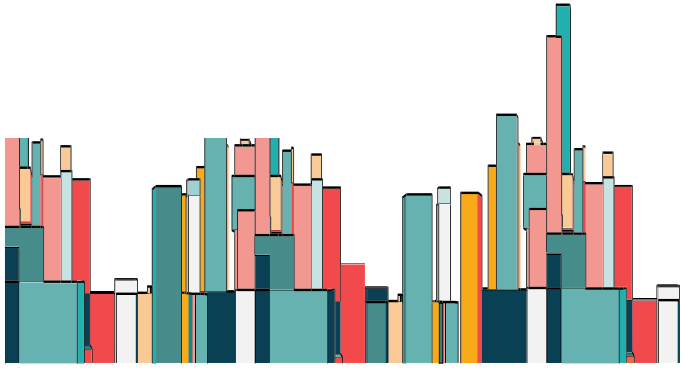
Backoff and delayed retries for clients producing to Kafka



Source: https://netflixtechblog.com/building-a-resilient-data-platform-with-write-ahead-log-at-netflix-127b6712359a

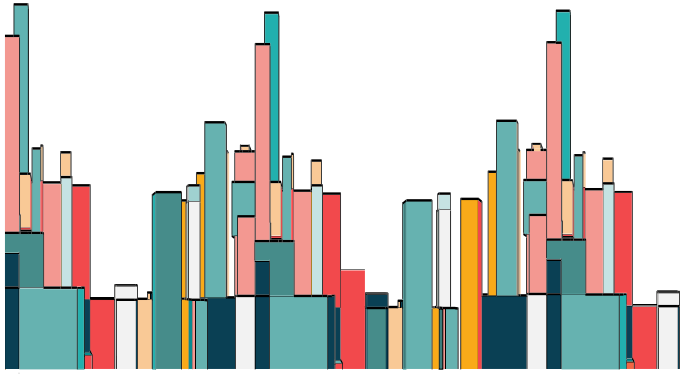


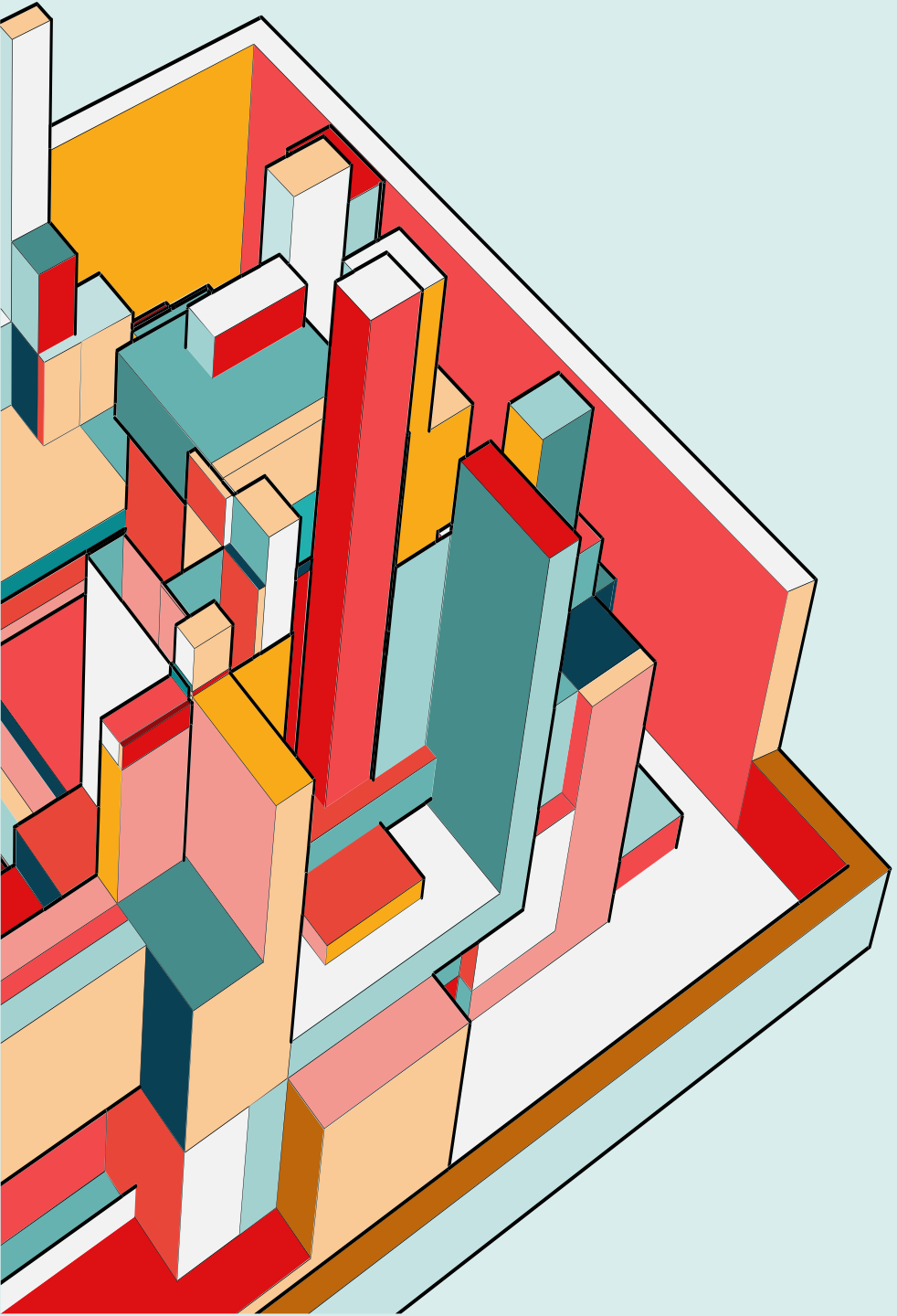
Backoff and delayed retries for clients consuming from Kafka



RESILIENT VS ANTIFRAGILE

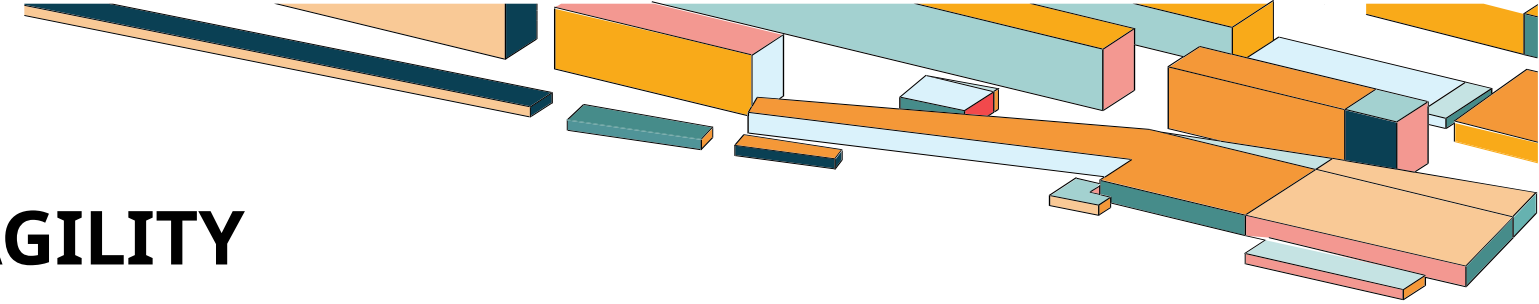
Aspect	Resilient System	Antifragile System (Netflix)
Failure Response	Recover to previous state	Improve beyond previous state
Cache Usage	Fixed TTL	Dynamic TTL based on failures
Replication	Static RF=3	Auto-adjusts based on failures
Monitoring	Alert on failures	Learn patterns, auto-adapt
Testing	Occasional tests	Continuous chaos engineering





TECHNICAL ASPECTS

Design principles
Anti-principles
Examples



THE PATH TO ANTIFRAGILITY

Design Principles/ Anti-Principles

- Modularity, Separate Processes / Monolith
- Weak Links - Isolatable / Inseparatable
- Redundancy / Uniqueness
- Diversity / Uniformity

Operational Principles / Anti-Principles

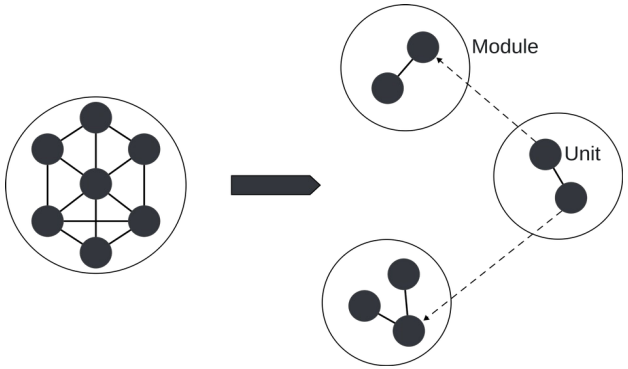
- Fail fast / Fail slow
- Feedback-Loops / No Feedback-Loops

MODULARITY

Goal: Errors shouldn't cause the whole system to crash

- Design of components that can operate independently of each other in order to minimize the impact of faults on other components.
- Modules have well-defined interfaces to ensure loose coupling and enable changes to one component without affecting others.

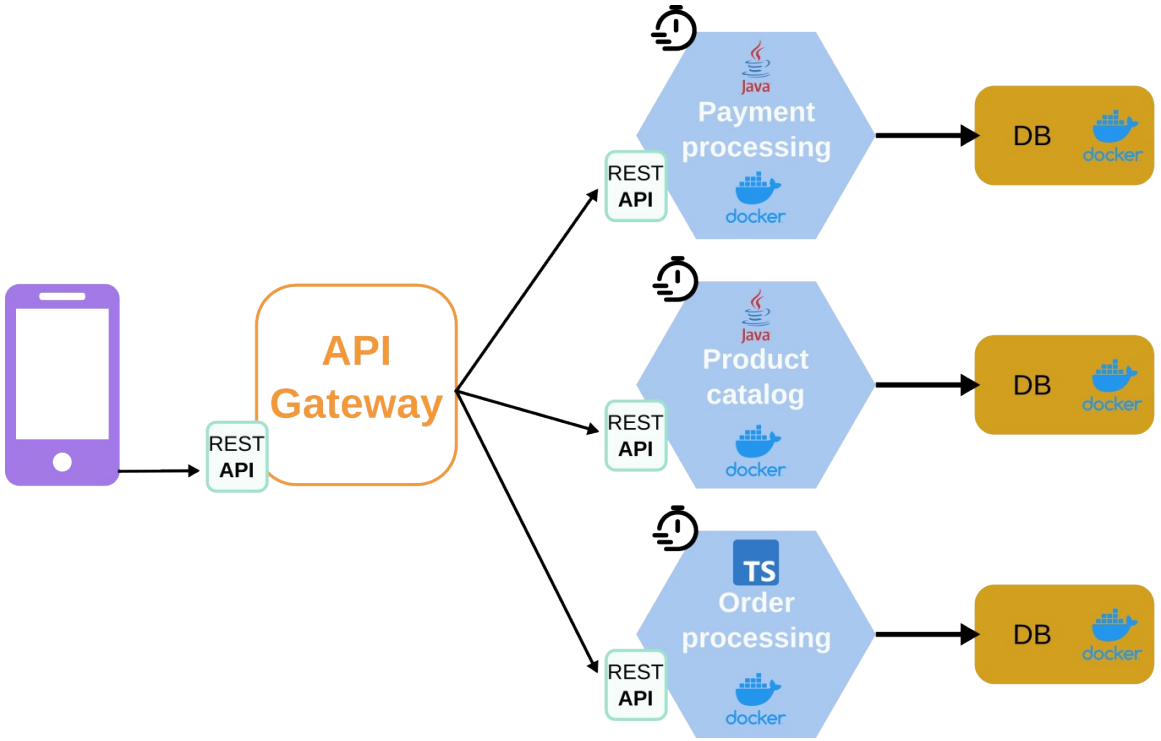
- Communication between modules must only take place via these interfaces; the internal state of a module must not be directly accessible.



Quelle: Kjell Jørgen Hole: Anti-fragile ICT Systems

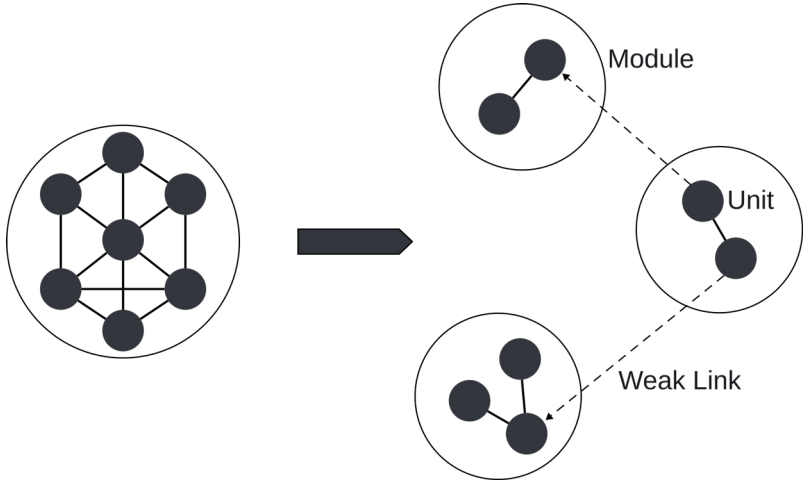
PRACTICE: MICROSERVICES

- Architecture in which an application consists of a collection of separate, independent services, each covering a specific business area.
- Microservices can be deployed independently, are scalable and can be written in different programming languages.
- Fast development cycles and fault tolerance
- Complexity in management, increased demands on network communication and the need for robust monitoring



WEAK LINKS

In a modular system, there are connections that are more fragile to disruptions or failures than others.

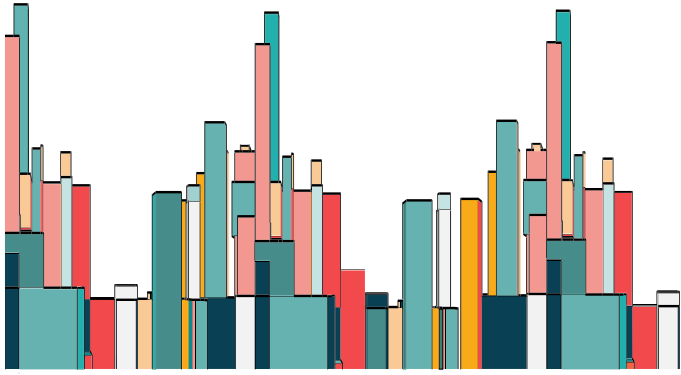


Goal: Minimize the spread of errors

Actions:

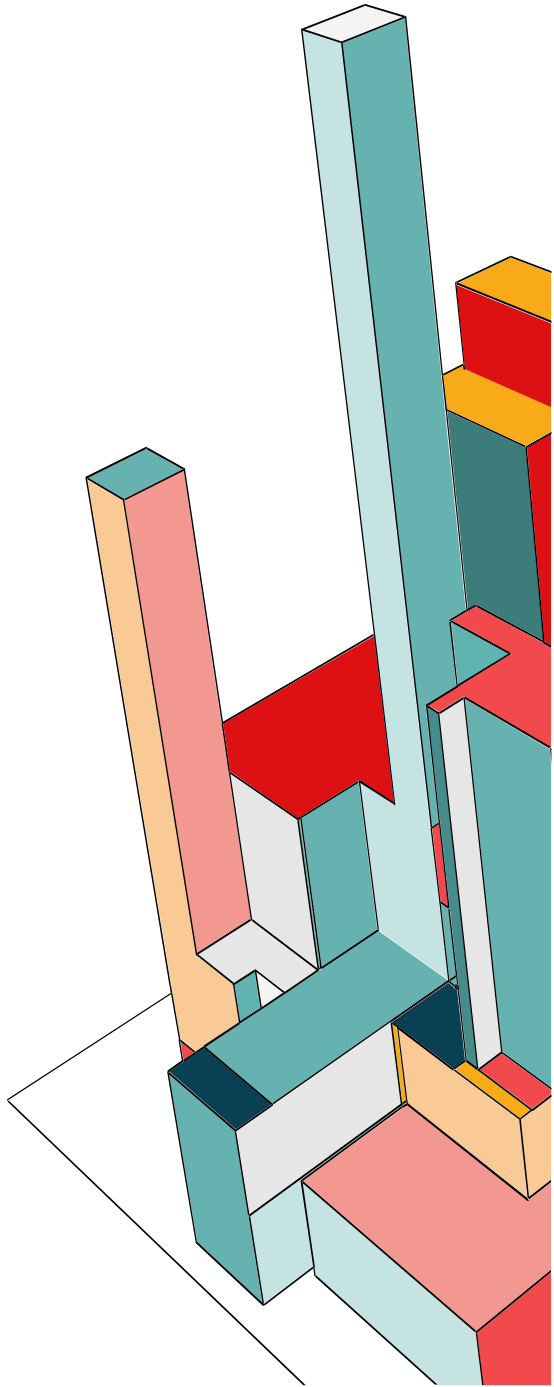
1. Identification of problematic connections
2. Reduction of negative effects (conversion to Weak Links)

Quelle: Kjell Jørgen Hole: Anti-fragile ICT Systems

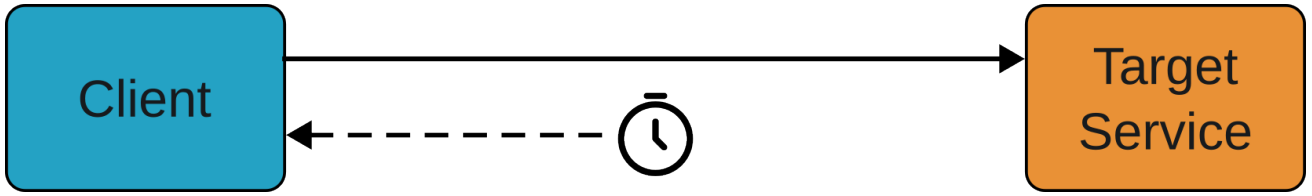


WEAK LINKS: PATTERNS

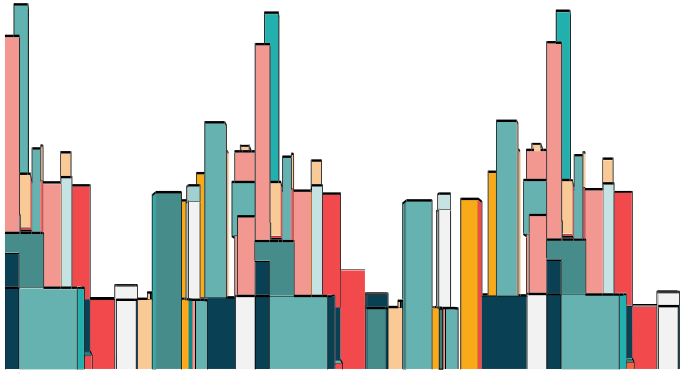
- Retry Pattern
- Circuit Breaker Pattern
- Bulkhead Pattern
- Timeout Pattern
- Health Check Pattern
- Fallback Pattern
- ...



TIMEOUT PATTERN

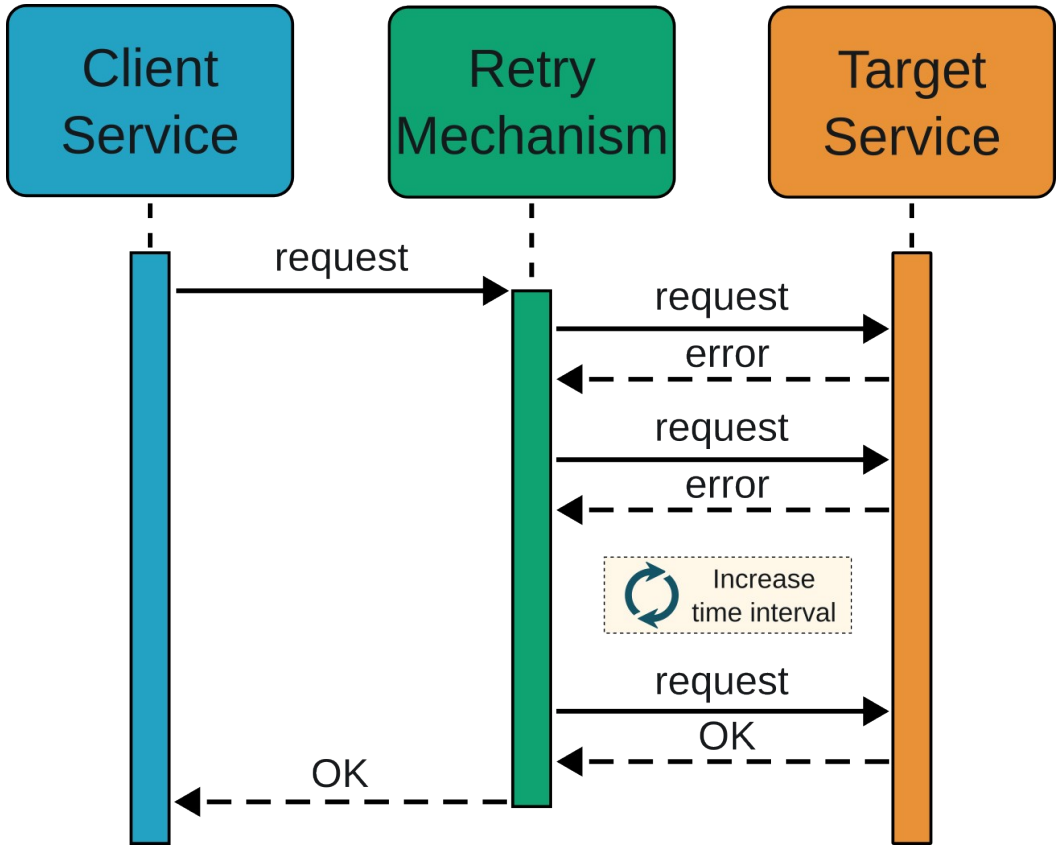


Goal: Preventing a call from being blocked



RETRY PATTERN

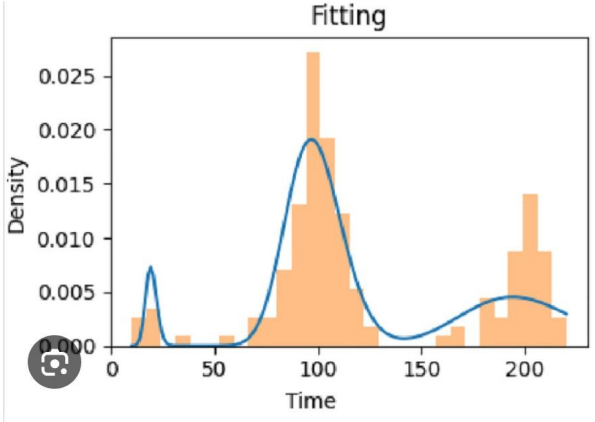
Goal: Absorb short-term service outages



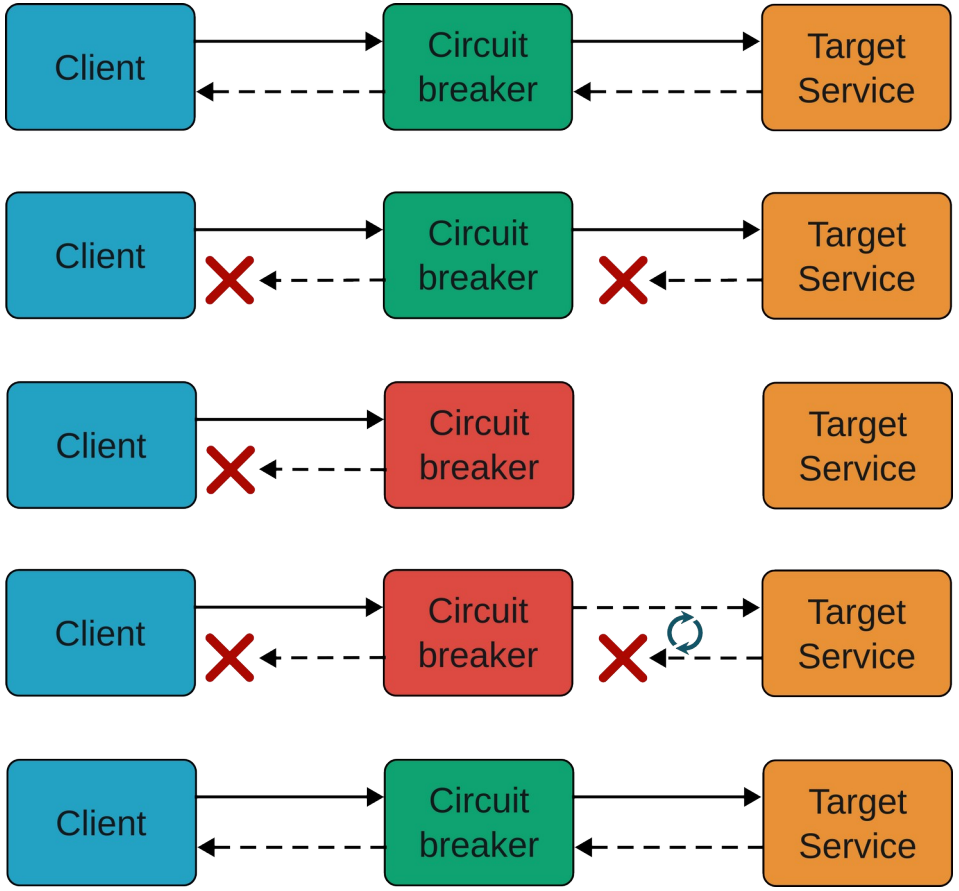
Time (sec)
1.240
1.608
4.206
9.140
11.036
15.075
17.912
20.604
22.032
26.300

Time interval	Counts
[0, 5]	3
[5, 10]	1
[10, 15]	1
[15, 20]	2
[20, 25]	2
⋮	⋮

Real Time Data -> Retry Time Interval



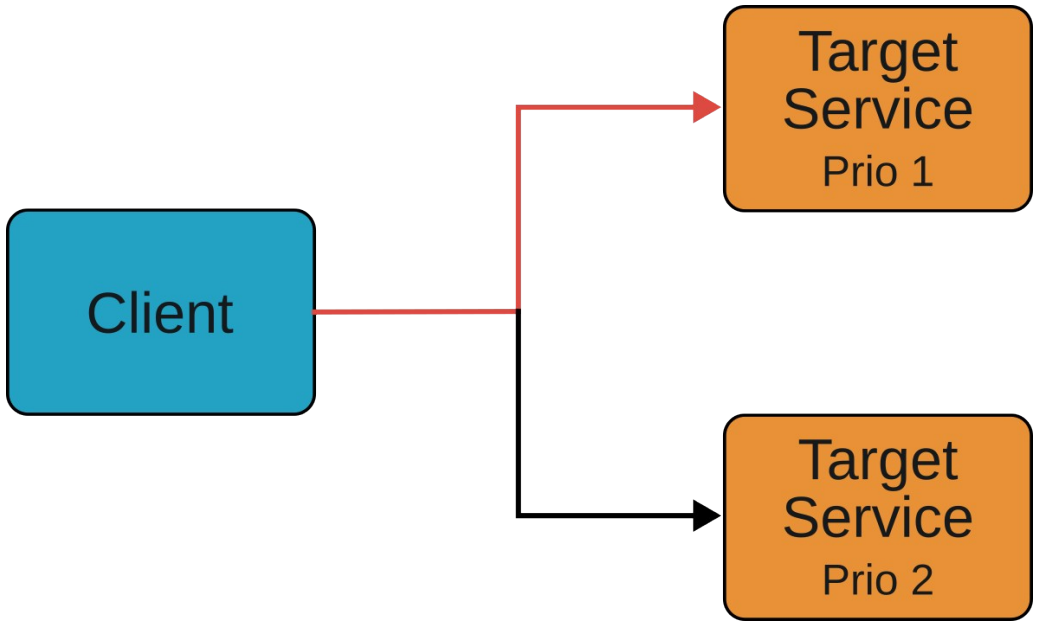
CIRCUIT BREAKER PATTERN



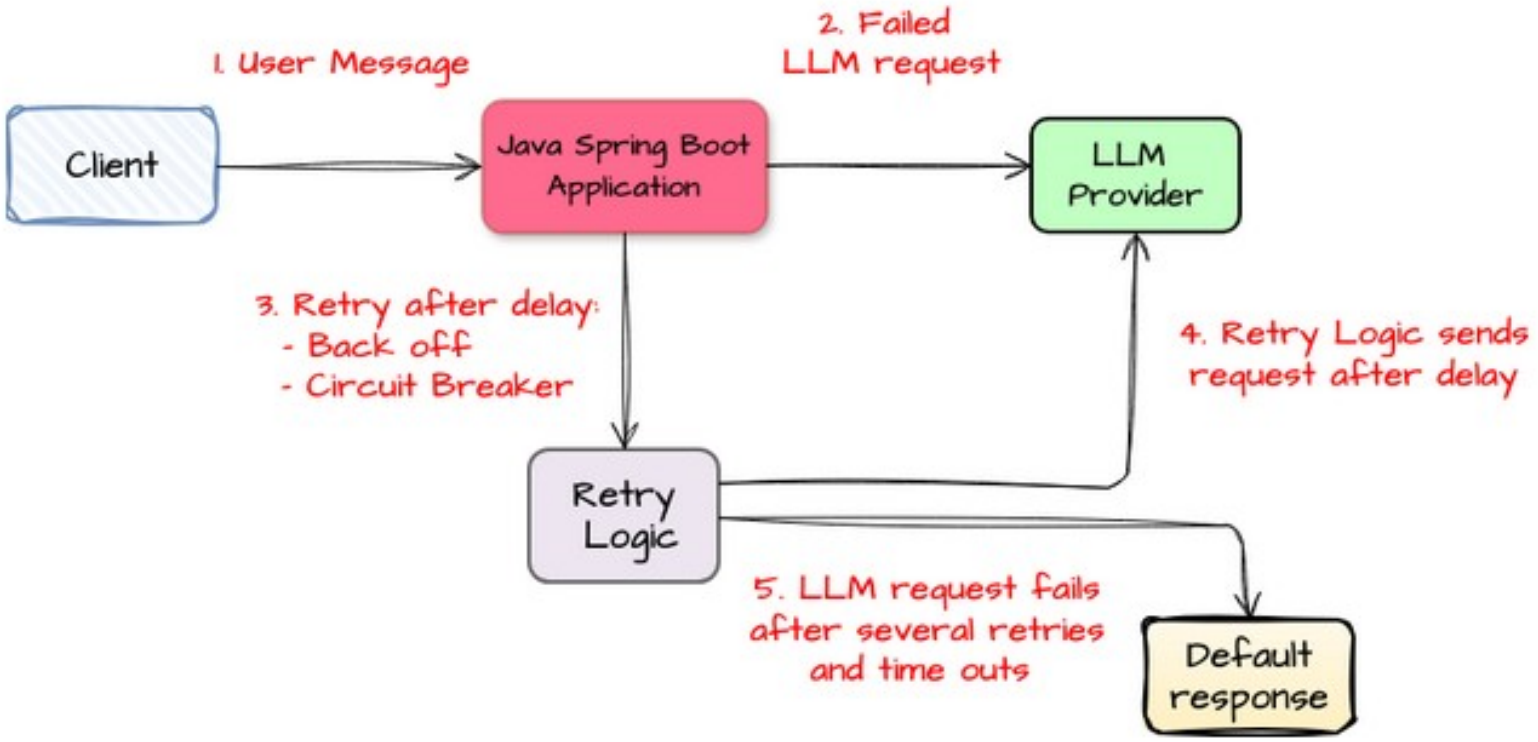
Goal: Fast detection of errors

FALLBACK PATTERN

Goal: Work around the error



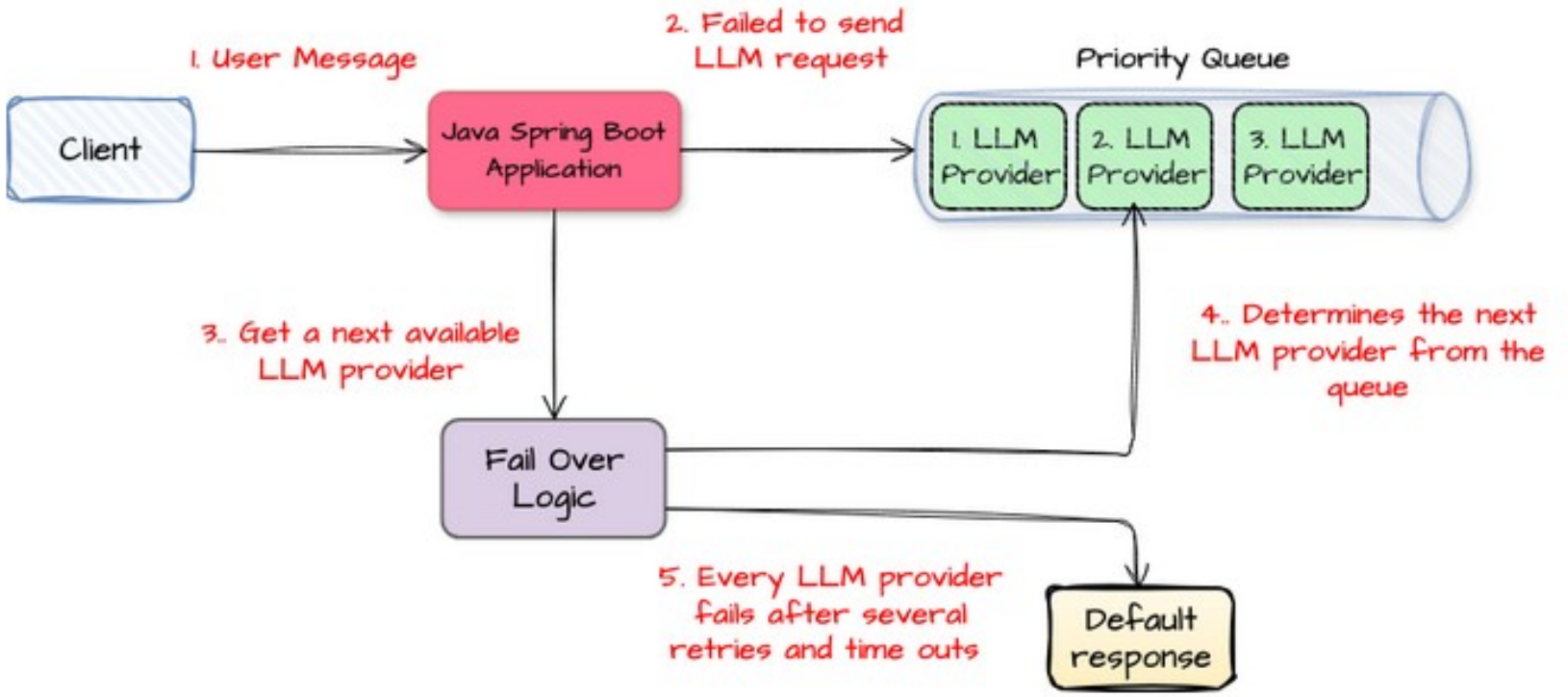
EXAMPLE FALLBACK PATTERN: LLM PROVIDER



Retries, back offs, and time outs for applications sending LLM requests.

Basel One

EXAMPLE FALLBACK PATTERN: LLM PROVIDER



The LLM provider fail over mechanism using the priority queue.

Basel One

EXAMPLE FALLBACK PATTERN: LLM PROVIDER

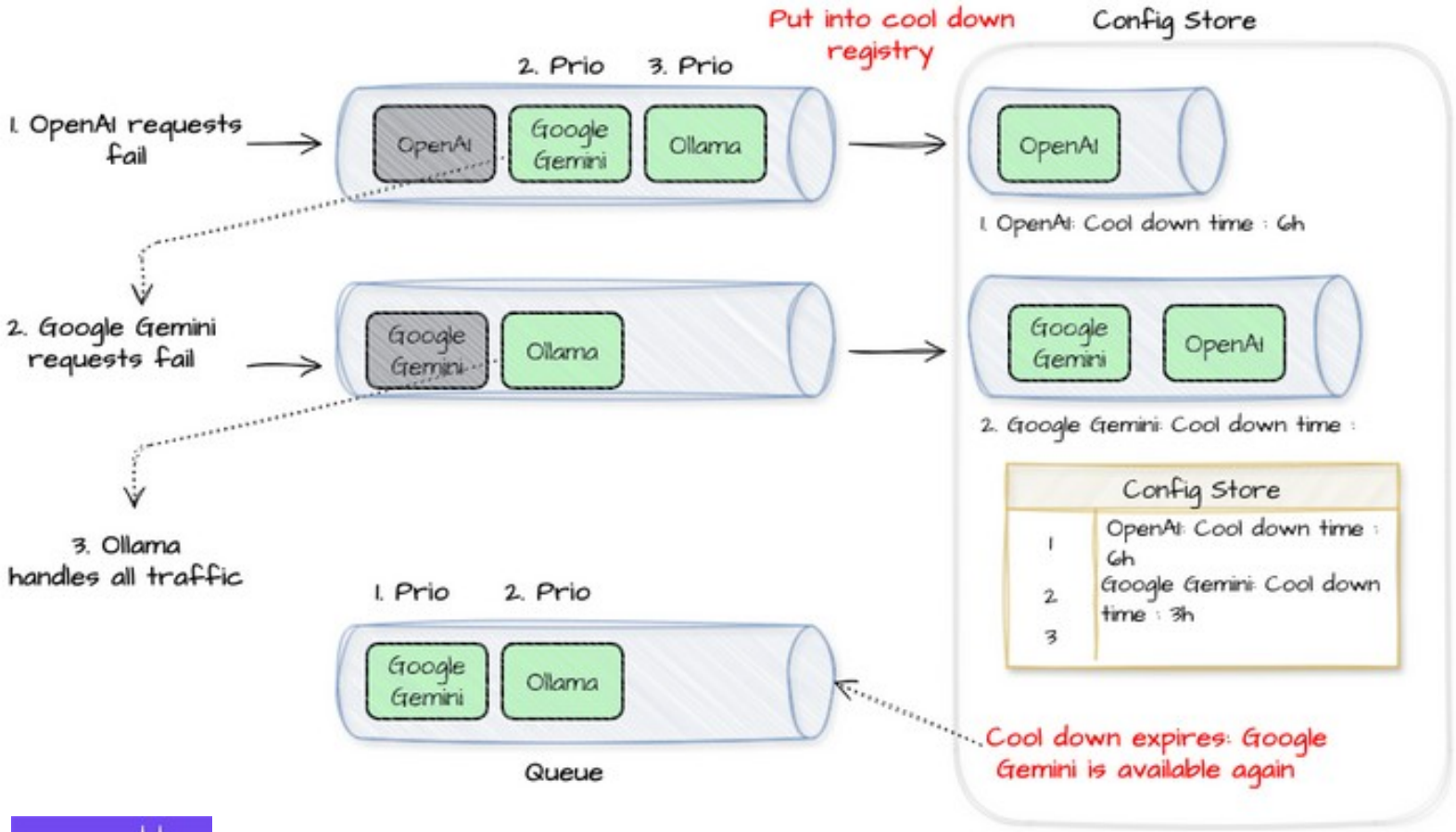
```
# Provider: Standard OpenAI API
llm.providers.openai.type=OPENAI
llm.providers.openai.api-key=${OPENAI_API_KEY:}
llm.providers.openai.model-name=gpt-4o-mini
llm.providers.openai.priority=1
llm.providers.openai.timeout=60
# Cooldown temperature: 0=frozen, 0.5=slow, 1.0=normal, 2.0=fast recovery
llm.providers.openai.cooldown-temperature=1.0

# Provider: Local Ollama
llm.providers.ollama-local.type=OLLAMA
llm.providers.ollama-local.base-url=http://localhost:11434
llm.providers.ollama-local.model-name=llama3.2:1b
llm.providers.ollama-local.priority=2
llm.providers.ollama-local.timeout=30
llm.providers.ollama-local.cooldown-temperature=1.0
```

Excerpt from the application.properties file. Provider priorities llm.providers.openai.priority = 1, and llm.providers.ollama-local.priority = 2 are hard coded.

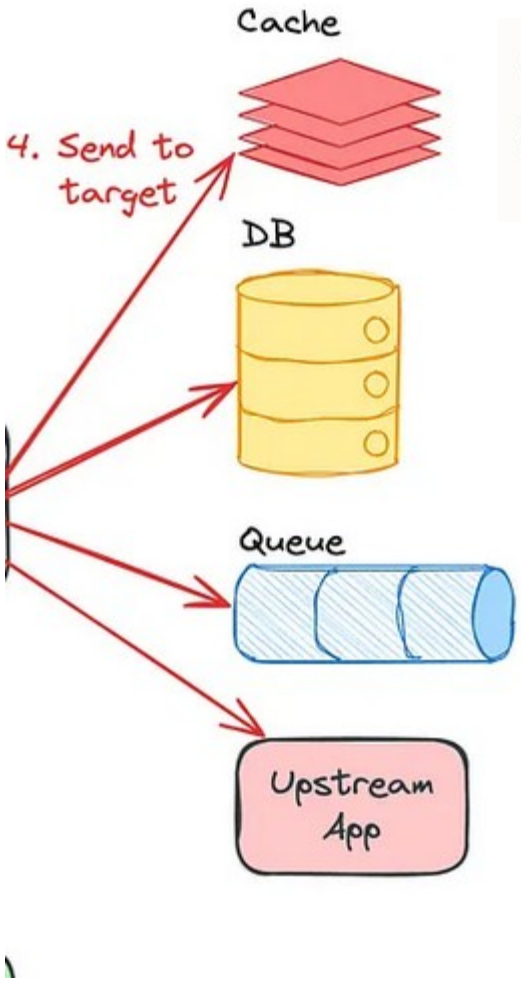
Basel One

EXAMPLE SELF-HEALING: LLM PROVIDER COOL DOWN



Basel One

Source: https://netflixtechblog.com/building-a-resilient-data-platform-with-write-ahead-log-at-netflix-127b6712359a

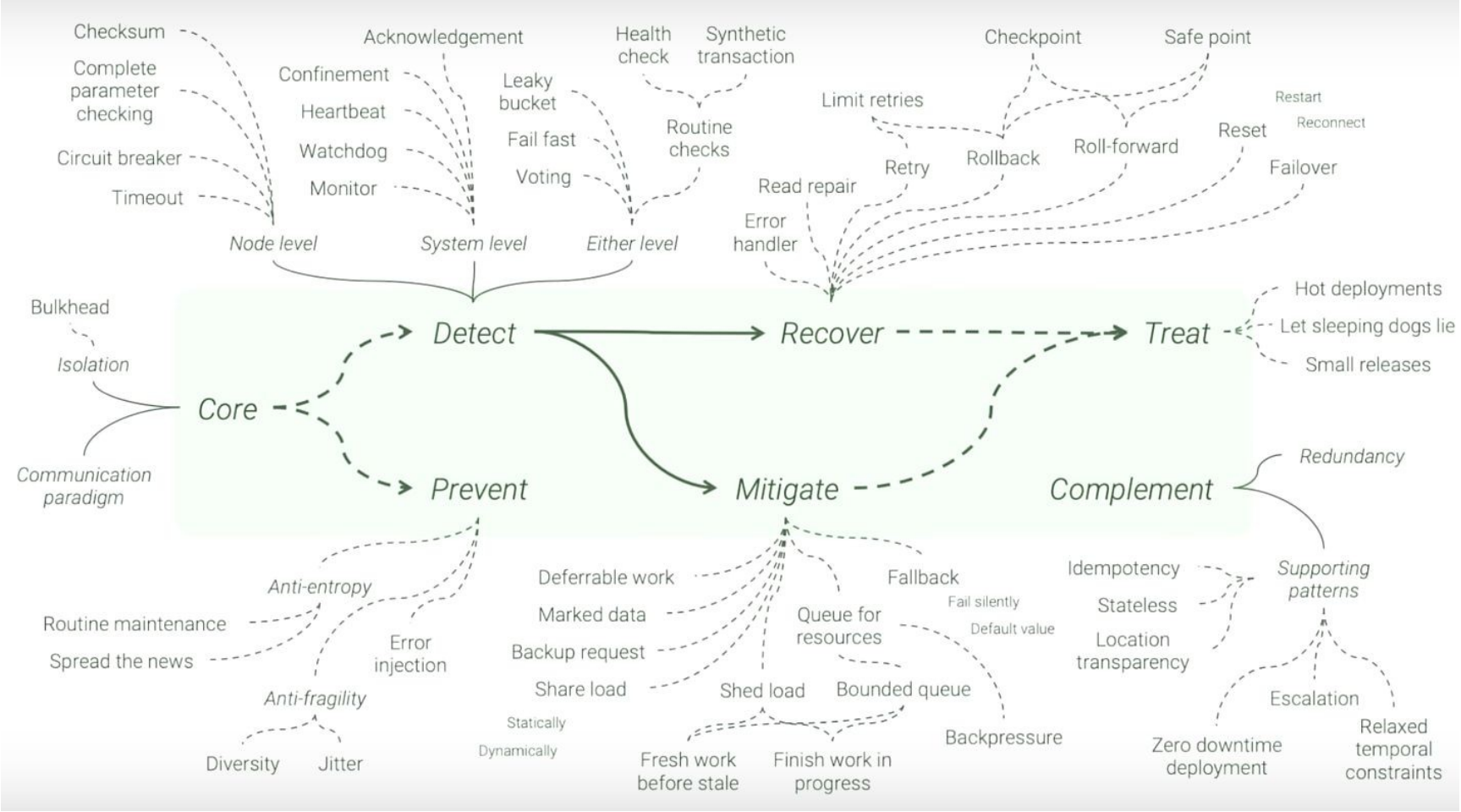



```
# When corruption detected, immediately extend cache TTL
cache.set_ttl("user:123", extended_ttl=86400)  # 24 hours instead of 1 hour
```

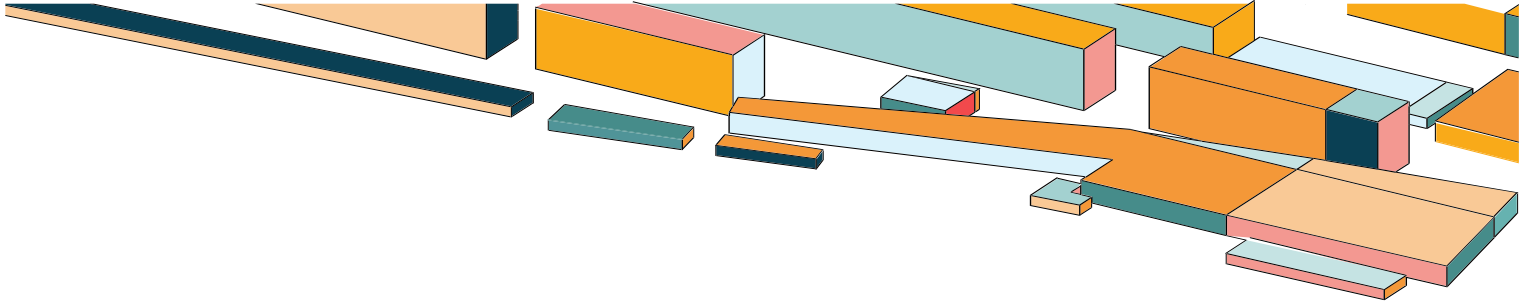
Example Failure: ABORT fault at /api/inventory/check caused OrderService to crash

```
@CircuitBreaker(name = "inventory", fallbackMethod = "fallback")
@Retry(name = "inventory", fallbackMethod = "fallback")
private InventoryStatus checkInventory(final String productId) {
    return restTemplate.getForObject(
  inventoryServiceUrl + "/check/" + productId, InventoryStatus.class);
}
private InventoryStatus fallback(final String productId, final Exception e) {
  log.warn("Inventory check failed, using cached value", e);
  return inventoryCache.getOrDefault(productId, InventoryStatus.optimisticDefault());
}
```

THAT WAS JUST A SMALL SELECTION....



Quelle: http://alex-ii.github.io/notes/2019/04/22/7_quests_resilient_design.html

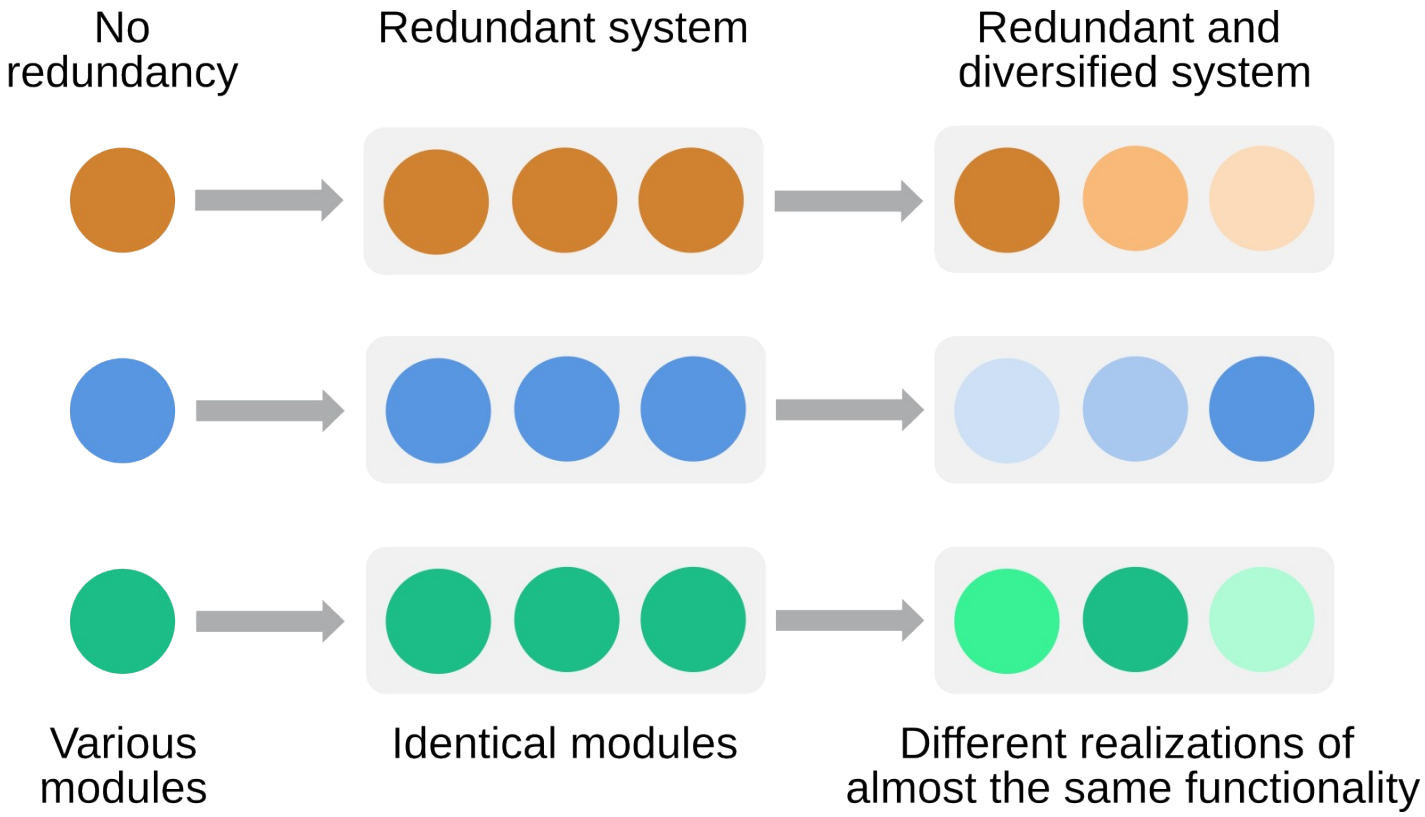


REDUNDANCY

Why is redundancy a necessary feature of antifragile systems?

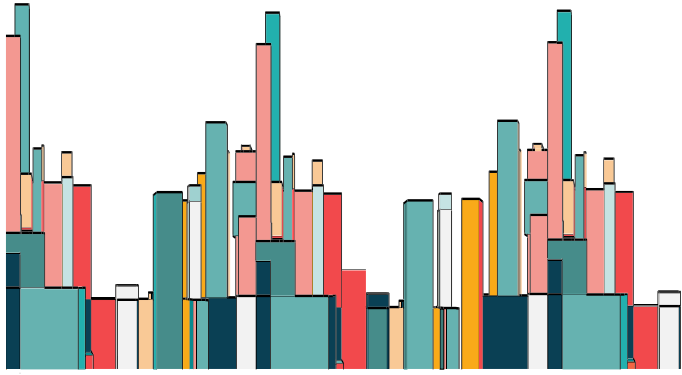
- Increased Reliability
  - If one subsystem fails, another subsystem takes over its function
- Increased Dependability and Availability
  - Redundant systems can detect and correct errors (fault tolerance)
  - Systems can learn from errors and adapt (self-healing)
- Reduction of overloads

DIVERSITY



Quelle: Kjell Jørgen Hole: Anti-fragile ICT Systems

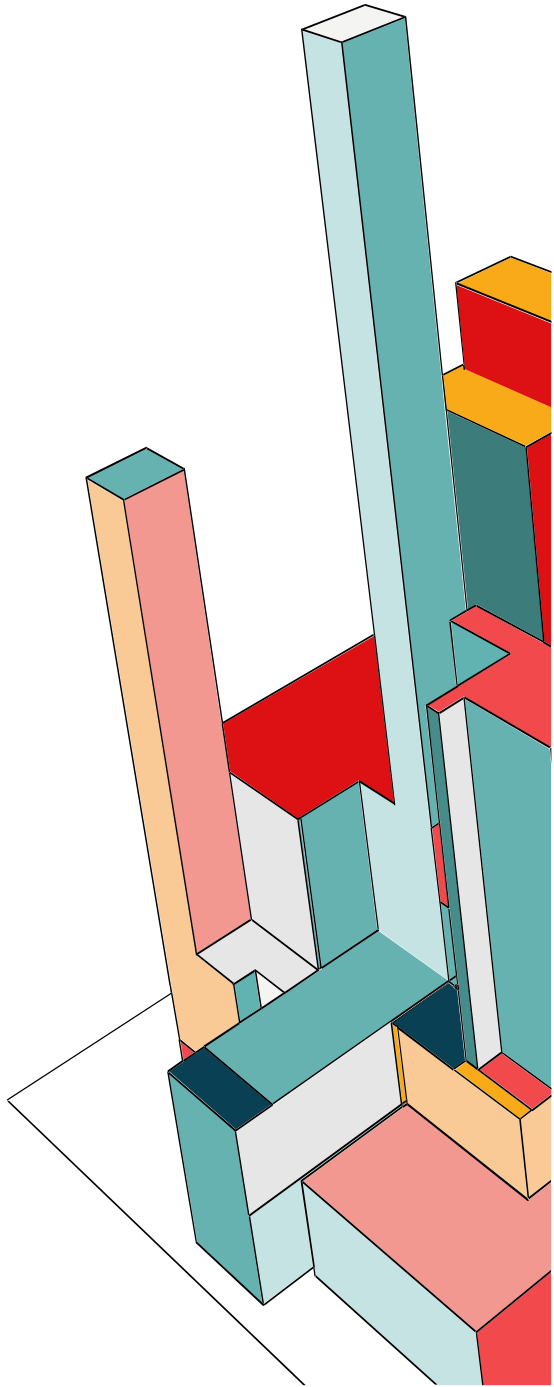
- Diversity reduces the probability of simultaneous failure of many modules
- Diversity prevents errors from spreading easily



Unbreakable by design: the secret tech of antifragile software
Iryna Dohndorf - Karakun

OPERATING PRINCIPLES

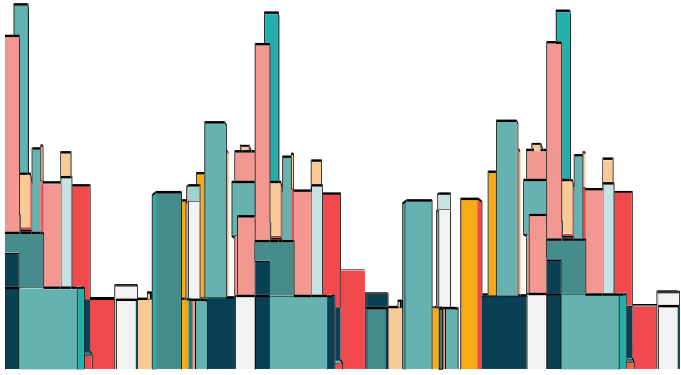
- Antifragility is not achieved solely by a system on its own; various aspects of operation and development processes are equally relevant.
- It is impossible to predict all negative effects on a system during development.
- Applying the principles of modularity, weak links, redundancy and diversity makes it more difficult for errors to spread and allows local vulnerabilities to be identified at an early stage.
- Identifying these errors/vulnerabilities and improving the system accordingly ultimately leads to an antifragile system.

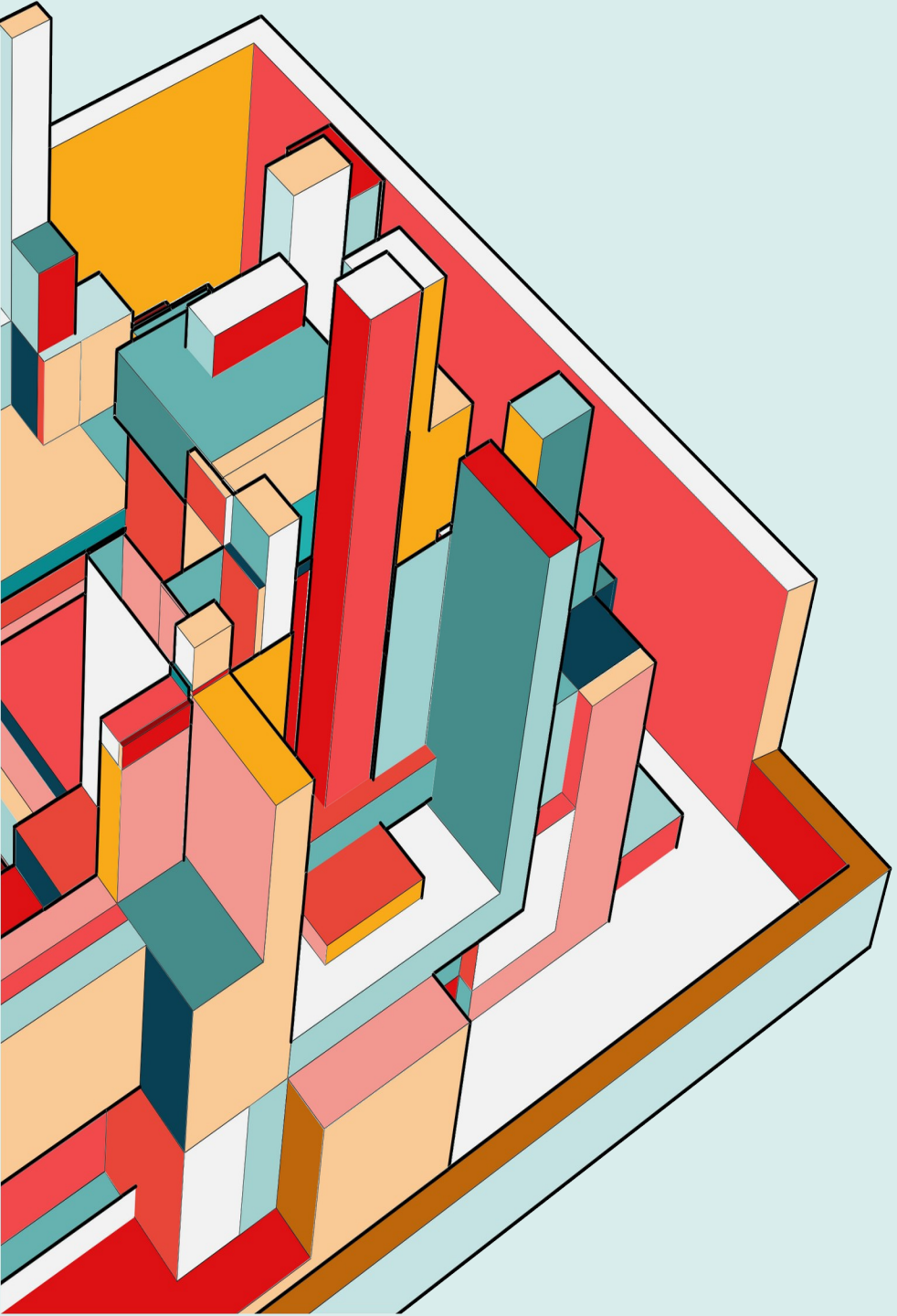


OPERATING PRINCIPLES

- Continuous monitoring of the system and analysis of data to identify potential for improvement
- Collecting and analyzing log files to identify errors and trends
- The architecture of the system should be adaptable and able to adjust to new requirements and findings.
- Conducting penetration tests

- Continuous improvement of the system based on the acquired information
- Automatic delivery of code changes to quickly fix errors





FAULT INJECTIONS

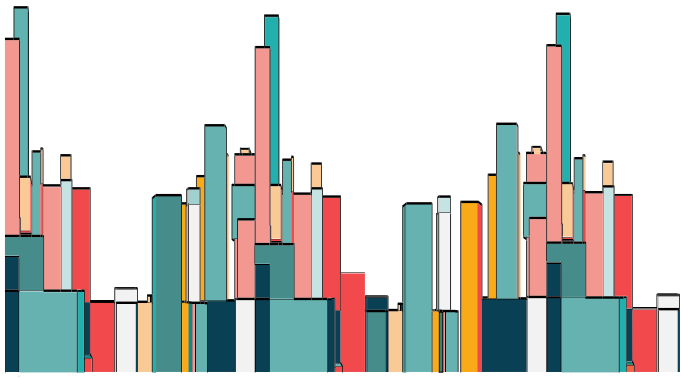
RESILIENCE TESTING VIA FAULT INJECTIONS

- Controlled failure simulation
- Runtime triggers
- Stress-testing systems by introducing faults

    1. Compile-time injection: Altering logic to simulate
    2. buffer overflows or invalid data inputs
    3. Runtime injection: Network latency,
    4. database disconnections, or memory exhaustions

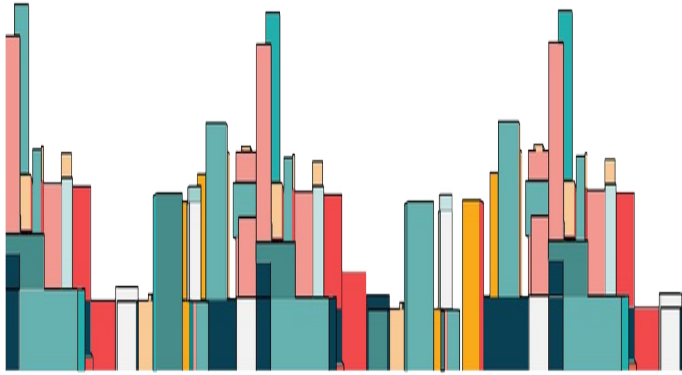
Validates:

- failure modes, error-handling, redundancy, fail over mechanisms



FAULT INJECTION & CHAOS ENGINEERING

- Chaos Engineering: more exploratory and systematic, experiments on entire system
- Tools:
  - Netflix’s Chaos Monkey
  - AWS Fault Injection Simulator
  - Latency Monkey
  - Netflix’s Failure Injection Testing (FIT)
  - AWS Game Days
  - Toxiproxy



FAULT INJECTION WORKFLOW

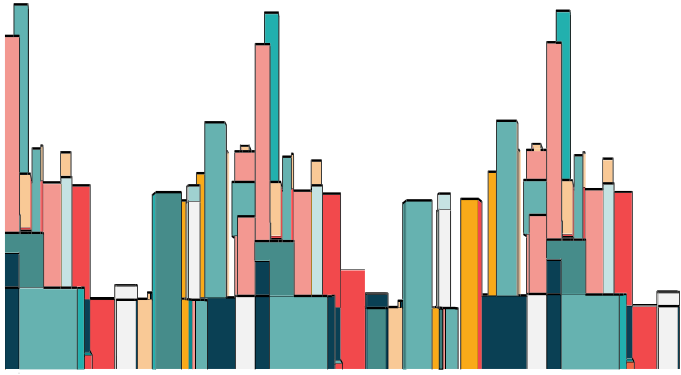
1. Target identification: Select critical components (e.g., APIs, databases, microservices)

2. Fault specification: Define scenarios (network packet loss, API timeout, killing processes)

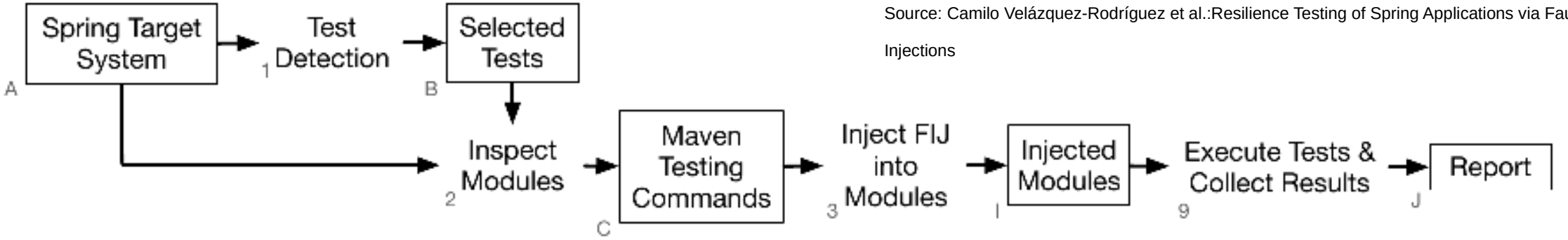
3. Fault application: Use tools to inject faults

4. Observation: Monitor system responses, recovery times, error rates

5. Improvement: Refine code, configs and architectures



INJECTING FAULTS TO JAVA SPRING MICROSERVICES



Source: Camilo Velázquez-Rodríguez et al.:Resilience Testing of Spring Applications via Fault Injections

Goal: Inject faults in the communication channels of Spring microservices → test resilience

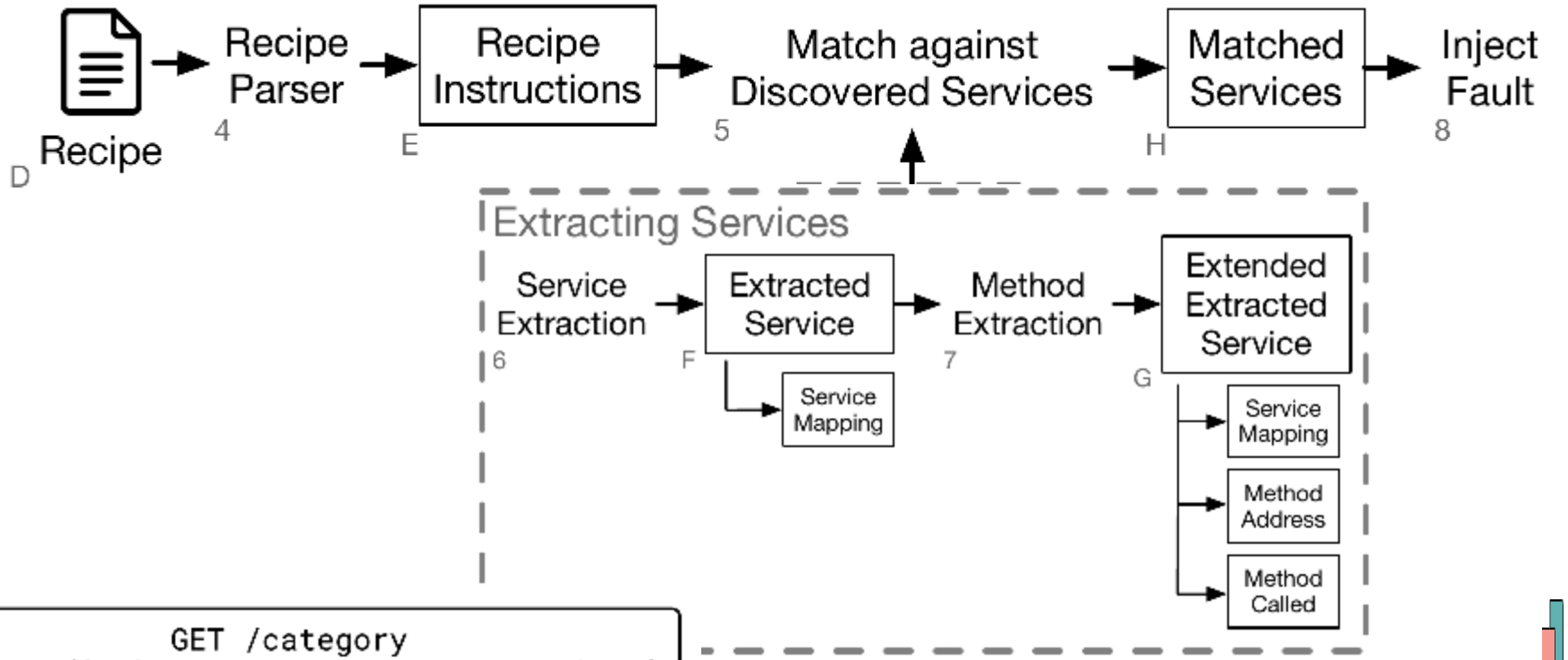
1. Test detection using @SpringBootTest, and classgraph
2. Manually written recipes
3. AspectJ: extracts the services' info at run time when test makes an API call
4. Simulates failure when matched services are called in tests

```
"recipes": {
    "Abort": {
        "Dst": "/ping",
        "Method": "POST"
    }
}
```

```
"recipes": {
    "Delay": {
        "Dst": "/pong",
        "Method": "GET"
        "Time": 5000
    }
}
```

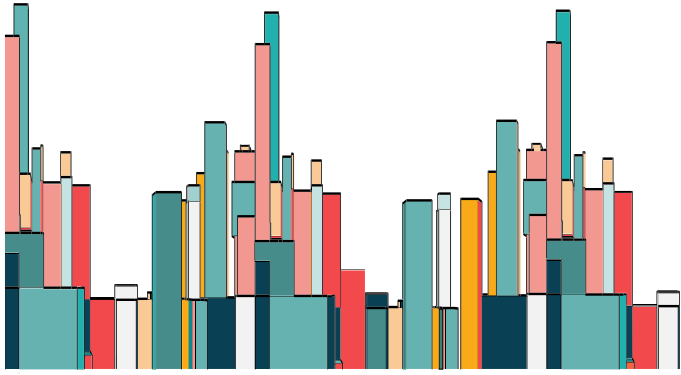
FAULT INJECTOR TOOL

Source: Camilo Velázquez-Rodríguez et al.:Resilience Testing of Spring Applications via Fault Injections




```
GET /category
faults: {'webui-CategoryRest.java:16': 3}
links: []
```

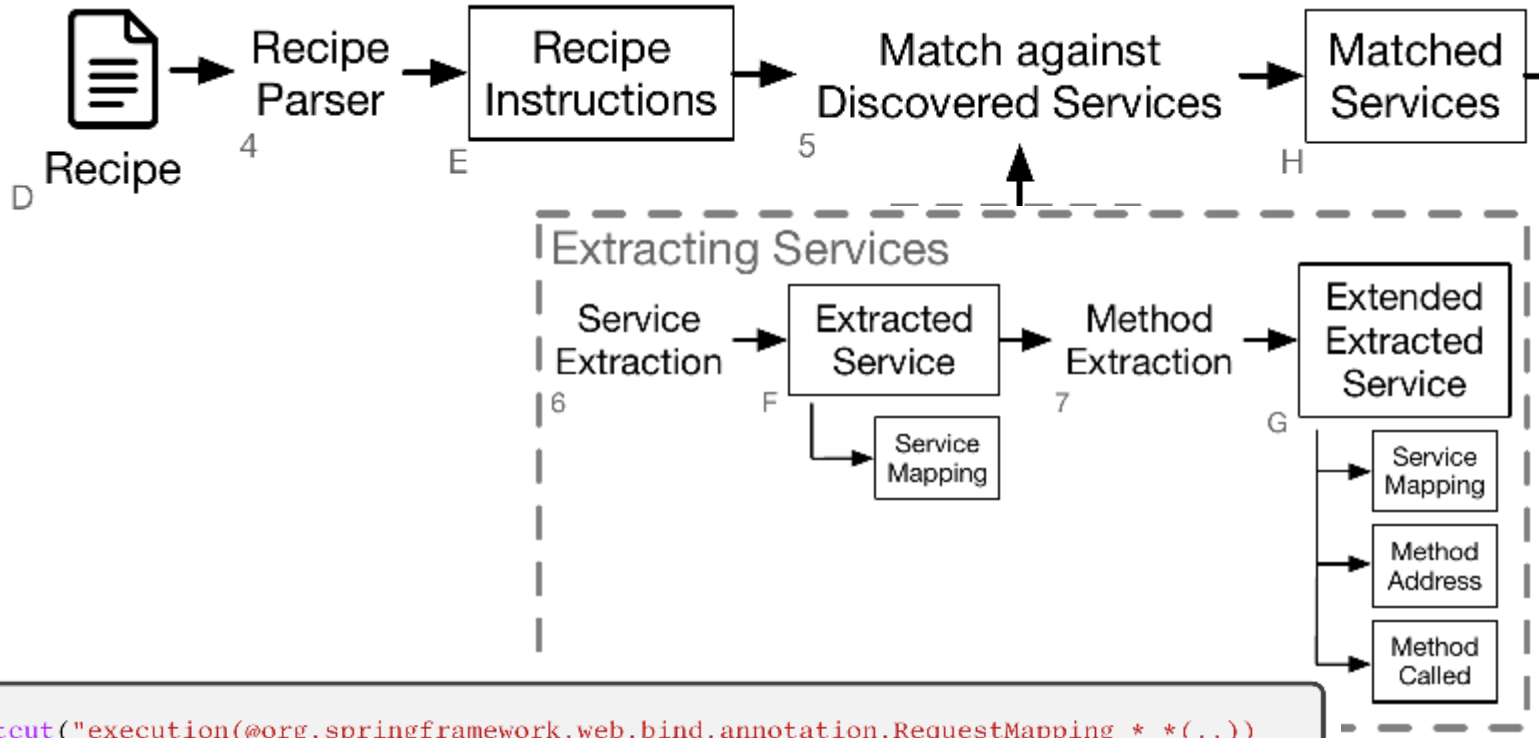
```
500 Internal Server Error
```



Unbreakable by design: the secret tech of antifragile software
Iryna Dohndorf - Karakun

FAULT INJECTOR TOOL

Source: Camilo Velázquez-Rodríguez et al.:Resilience Testing of Spring Applications via Fault Injections



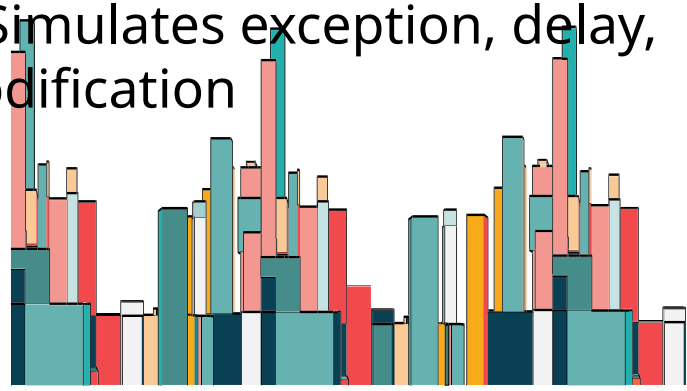

```
@Pointcut("execution(@org.springframework.web.bind.annotation.RequestMapping * *(..))
    && @annotation(mapping)")

@Pointcut("call(public * org.springframework.web.client.RestTemplate.getFor*(..)) " +
          "|| call(public * org.springframework.web.client.RestTemplate.postFor*(..)) " +
          "|| call(public * org.springframework.web.client.RestTemplate.put(..)) " +
          "&& within(@org.springframework.web.bind.annotation.RequestMapping *)")
```

1. Uses ApectJ pointcuts to extract service mappings

2. TracePoint model with classname, methodName, targetURL, httpMethod, serviceName

3. Simulates exception, delay, modification

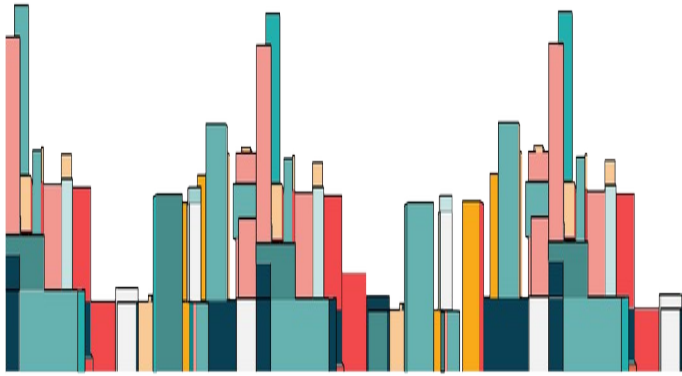


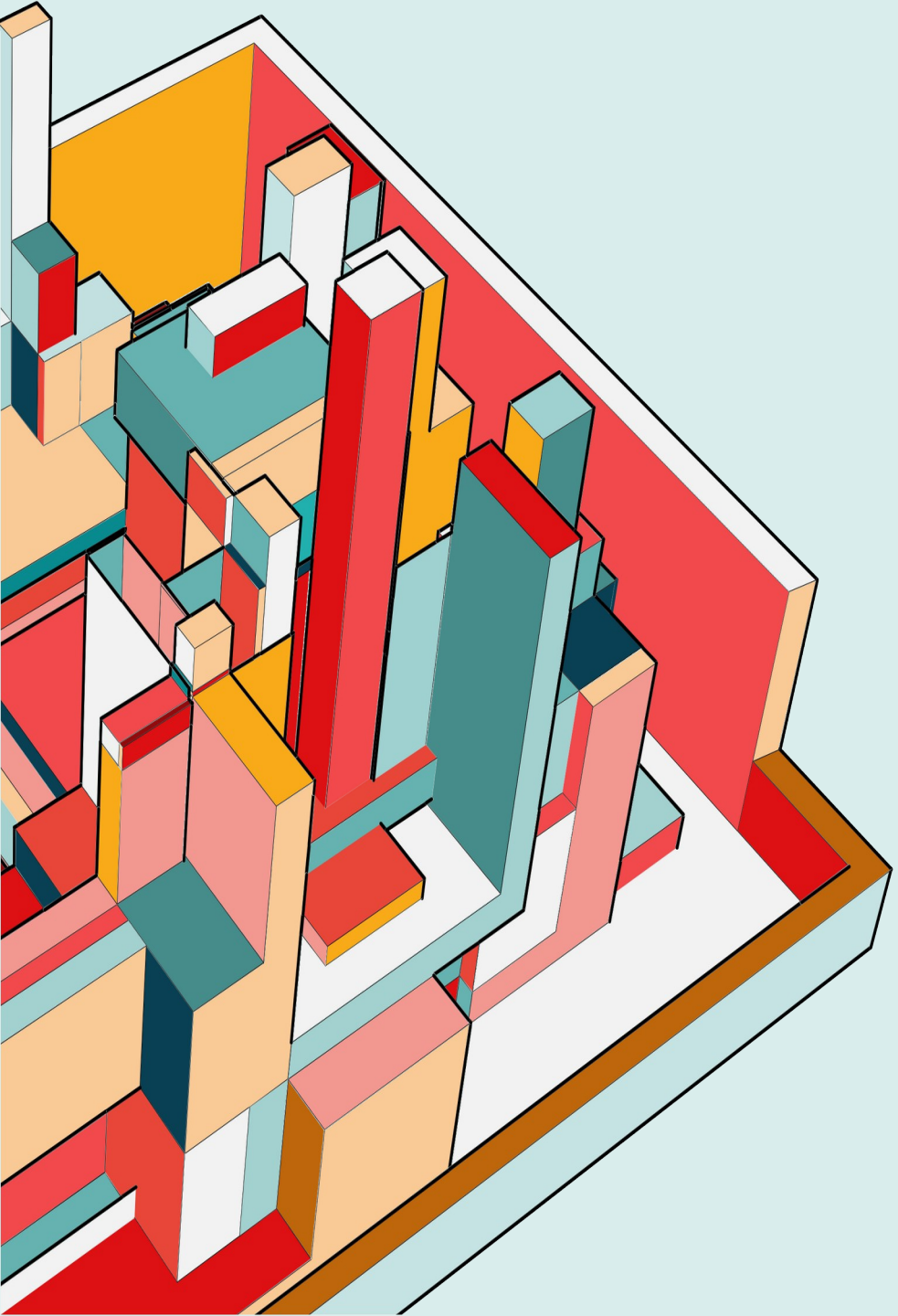
antifragile software

FAULT INJECTION FOR LLM-ENRICHED JAVA APPS

- LLM API faults:
  - Time outs, rate limits, token limits, malformed responses
- Embedding faults:
  - Dimension mitmatch, empty vectors, service unavailable
- Vector Store faults:
  - Connection failure, slow queries, wrong results
- Tool Execution faults:
  - Invalid outputs, and unhandled exceptions
- Memory Faults:
  - Context overflow, retrieval failure, corruption
- Content Faults:
  - Hallucination injection, safety triggers, format errors







CONCLUSION

WHAT WE COVERED

- Definition & Motivation
- Examples
- Design Principles & Anti-Principles
- Testing technique: Fault Injections

THANK YOU FOR YOUR ATTENTION !!



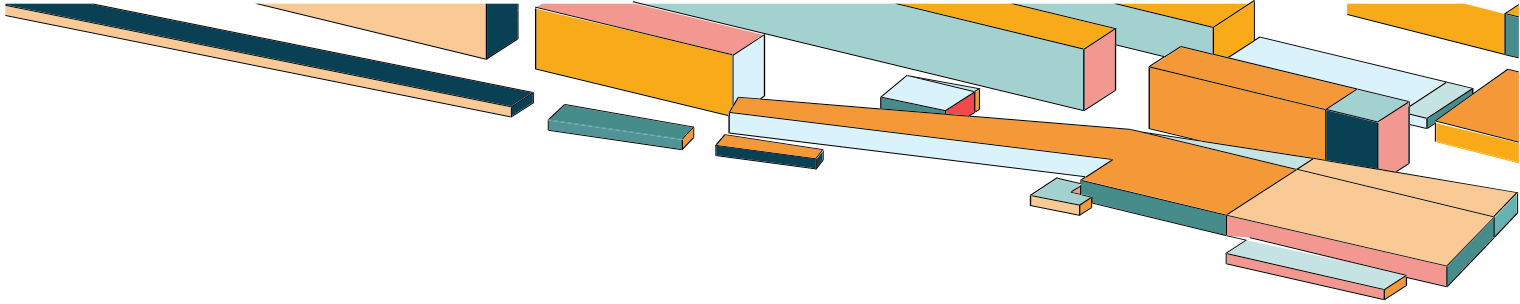
iryna.dohndorf@karakun.com

https://linkedin.com/in/iryna-dohndorf/

Karakun AG

Elisabethenanlage 25
4051 Basel
Schweiz

T. +41 61 551 36 00
E. info@karakun.com
W. www.karakun.com



REFERENCES

[1]: Nassim Nicholas Taleb: Antifragilität - Anleitung für eine Welt, die wir nicht verstehen, Albrecht-Knaus-Verlag, 2013

[2]: Kjell Jørgen Hole: Anti-fragile ICT Systems, Springer, 2016

[3]: Kjell Jørgen Hole: Tutorial on systems with antifragility to downtime, Springer, 2022

[4]: Martin Monperrus: Principles of Antifragile Software, Association for Computing Machinery, 2017

[5]: Daniel Russo, Paolo Ciancarin: Towards Antifragile Software Architectures, 2017

[6]: Cristian Axenie, Oliver López-Corona , Michail A. Makridis, Matteo Saveriano, Alexandru Stancu & Jeffrey West: Antifragility in complex dynamical systems, 2024

Basel One