
# Towards Reliable Cloud Microservices with Intelligent Operations

Ph.D. Oral Defense of Tianyi Yang

Supervised by Prof. Michael Lyu

Wednesday 17 August 2022

ARISE
Automated Reliable Intelligent Software Engineering

香港中文大學
The Chinese University of Hong Kong

# Online Cloud Services Are Everywhere

To-Consumer services

To-Business services

Cloud services

# Online Cloud Services' Reliability Is Crucial

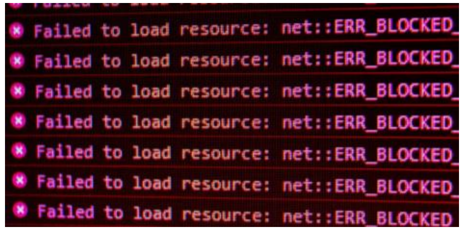
-- to both service providers and end users!

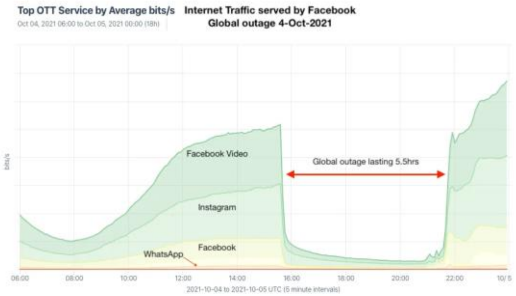
A Tiny Error

System Outage

User Dissatisfaction

Revenue Loss




**User satisfaction study shows Facebook vulnerable to Google+**

Facebook, which ranked last in a customer satisfaction study, has benefited from 'a monopoly of sorts' in the social networking market

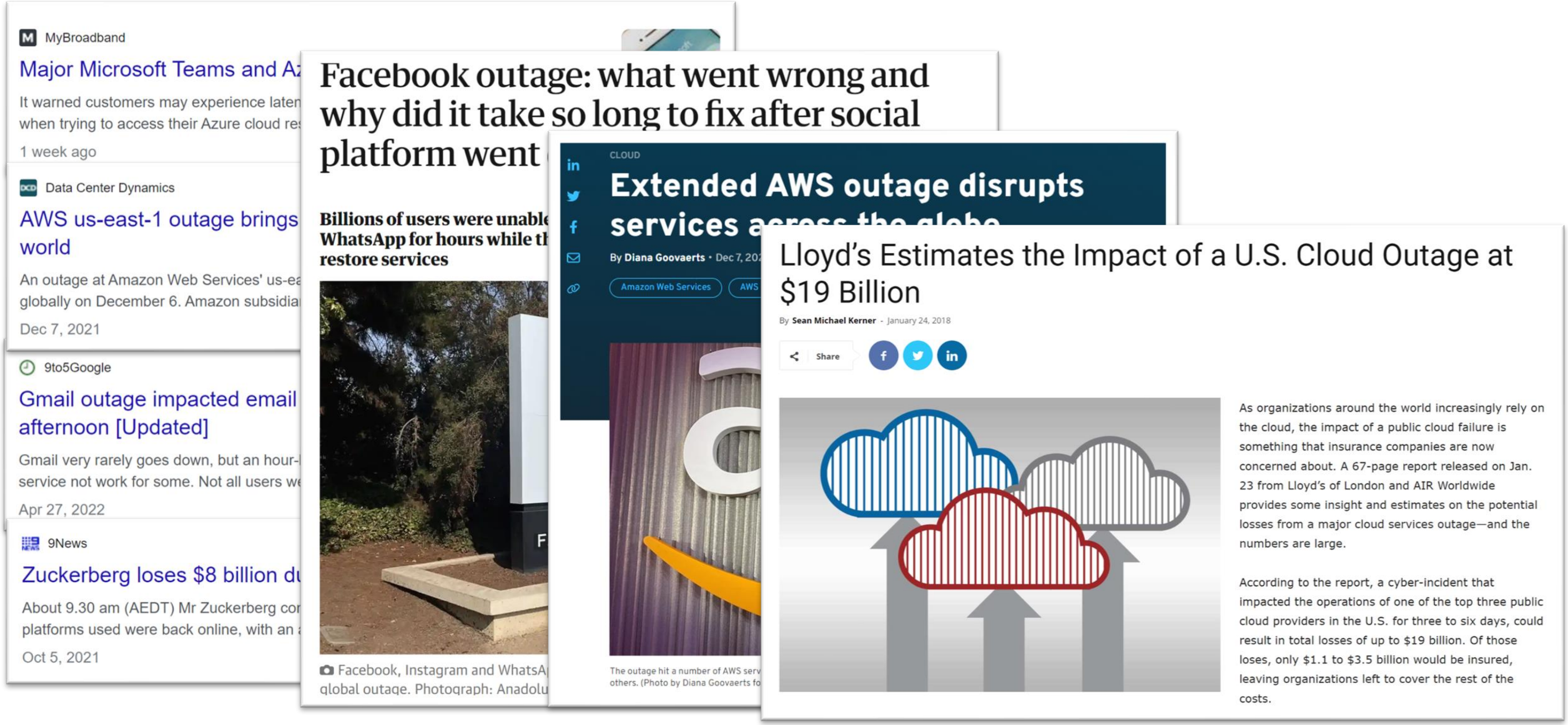

Facebook Parent Loses More Than $250 Billion in Market Value, Biggest U.S. Stock Market Drop in History

Meta Platforms shares drops after company cites headwinds from Apple iOS privacy changes, TikTok competition

By Todd Spangler

Update about the 4 October outage | Meta for Business (facebook.com)

# Real-world Examples




[Facebook outage: what went wrong and why did it take so long to fix after social platform went down? | Facebook | The Guardian]
[Extended AWS outage disrupts services across the globe | Fierce Telecom]

# Service Outage!

Reliability management of online services

is important,

but **challenging**,

due to the increasing complexity

and **distributed nature** of online

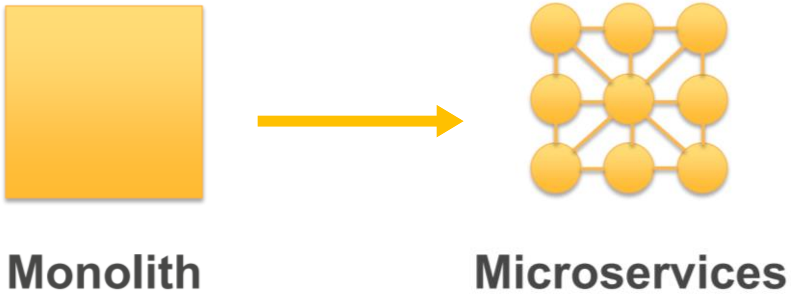
services.

# CONTENTS

1

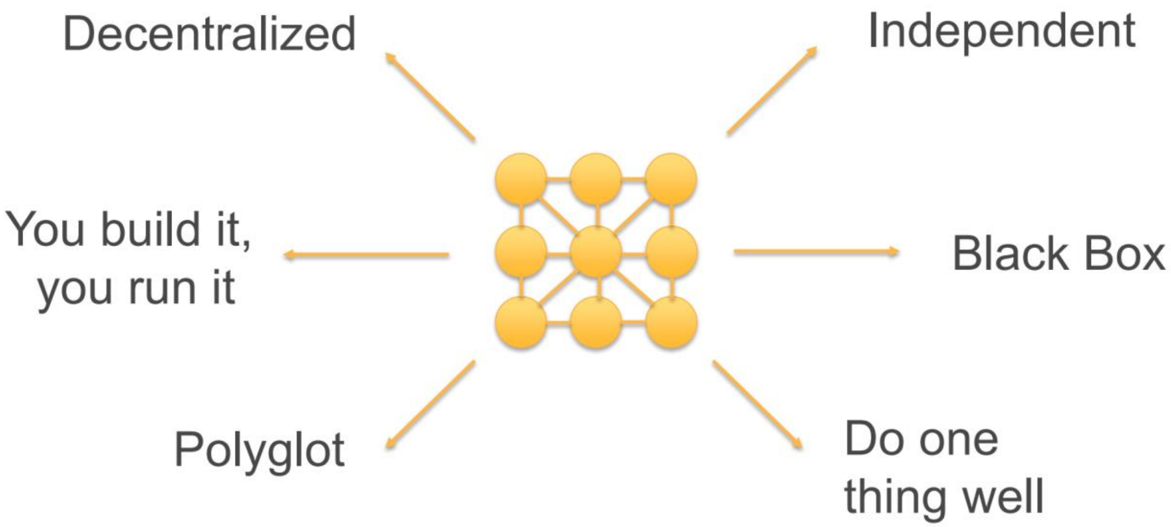
# Background and Contributions

# The Microservice Architecture




docker

kubernetes

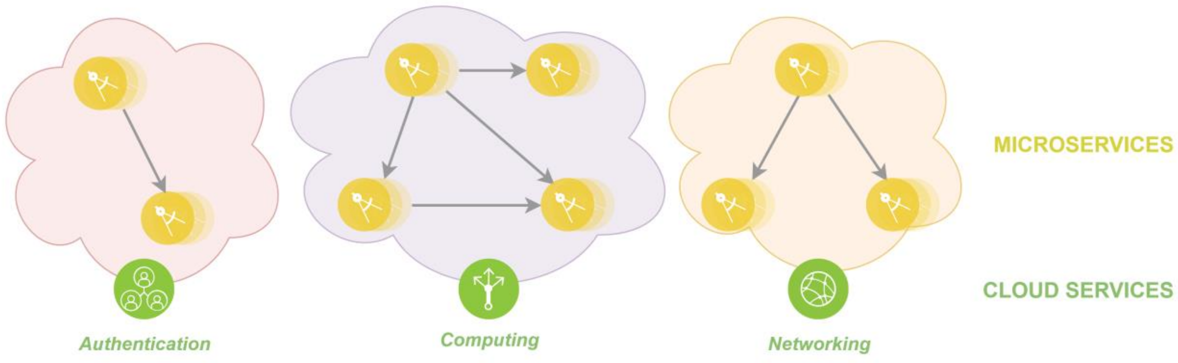



Microservices architecture is an approach in which a single application is composed of many **loosely coupled** and **independently deployable** small programs.

Microservices on AWS, AWS Summit Berlin 2016, Apr 12, 2016
[What are Microservices? | IBM]

# Online Service Systems Shift to Microservices

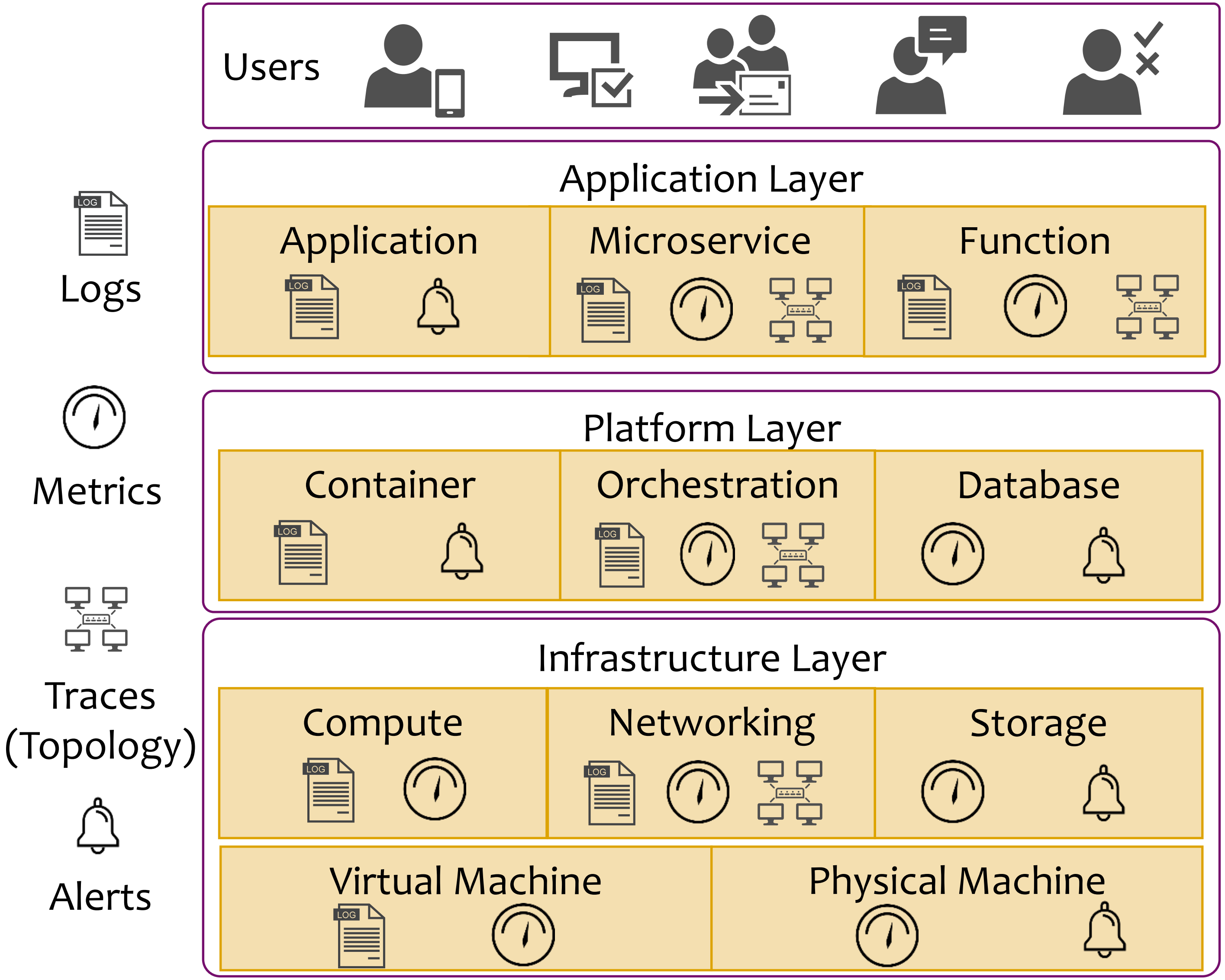
- Microservices collectively comprise multiple cloud services.
  - *Online services*: provide high-level APIs.
  - *Microservices*: collectively handle the external request via multiple chained invocations.

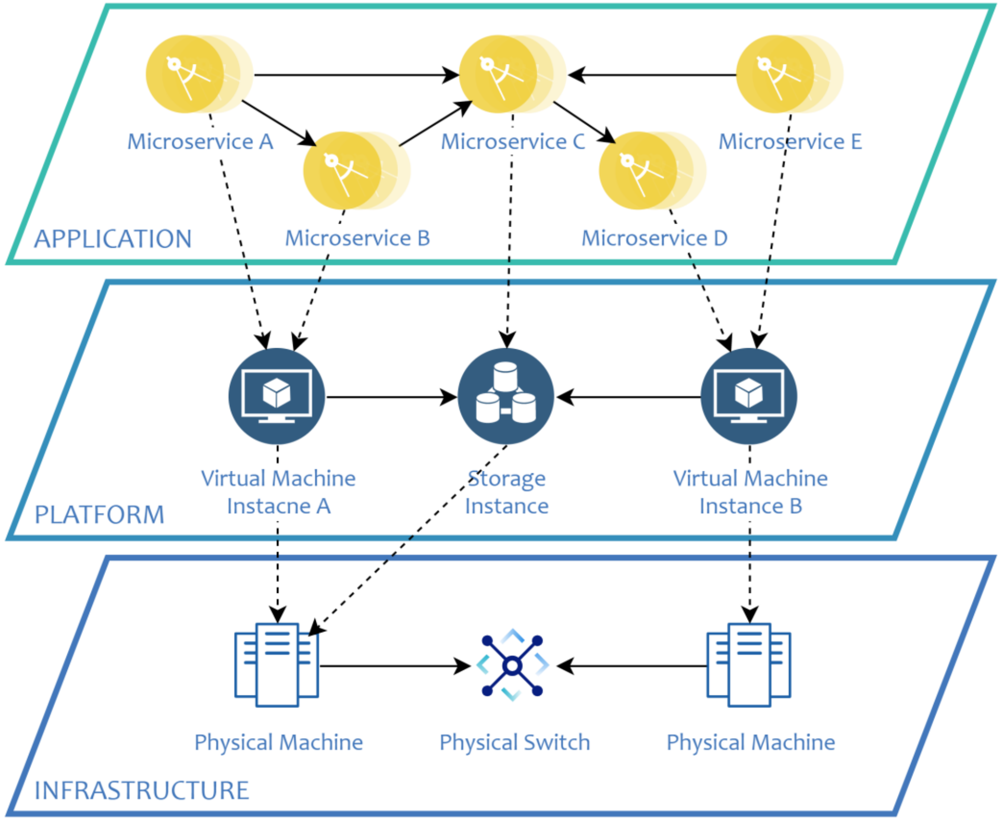



- Minor anomalies may magnify impact and escalate into system outages!

Loosely-coupled nature makes failure diagnosis difficult.

# Microservices Generates a Variety of Data



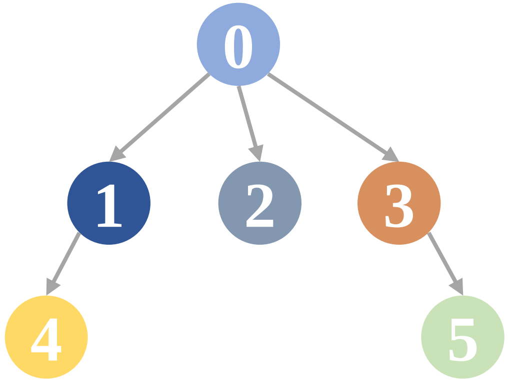



# Traces

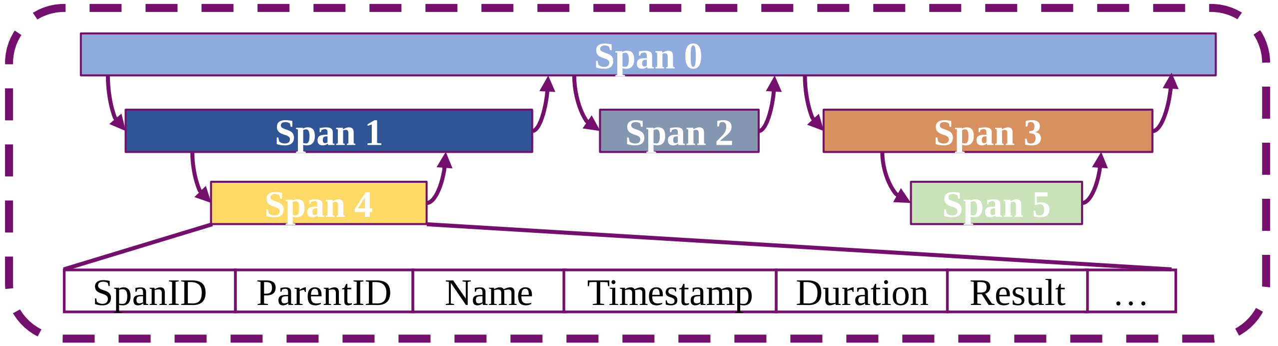
- Tracks the processing of each request.
- Terminologies
  - *Span log (abbr. span)*: a log recording the contextual information of each service invocation.
  - *Trace log (abbr. trace)*: all the spans that serve for the same request.

| | |
|---|---|
| *Span ID* | e22f30bdbfd09134 |
| *Parent Span ID* | b42a04bf18997d5d |
| *Name* | ts-preserve-service |
| *Timestamp* ($\mu$s) | 1618589098705000 |
| *Duration* ($\mu$s) | 1126 |
| *Result* | SUCCESS |
| *Trace ID* | c0d17d481f47bdd9 |
| *Additional Logs* | … |

A span generated by the train-ticket benchmark.




Service invocations for a request.




A trace with 6 spans.

# Monitoring Metrics

- Monitoring Metrics
  - Observes real-time statuses of microservice systems.
  - Timestamped data with fixed intervals.

- Terminologies
  - System performance metrics.
    - E.g., CPU usage, memory usage, NIC send/receive rate.
  - Business metrics.
    - E.g., Request latency, request error rate, and throughput.

# Logs & Alerts

- Logs
  - Semi-structured text printed by logging statements (e.g., `printf()`, `logger.info()`).

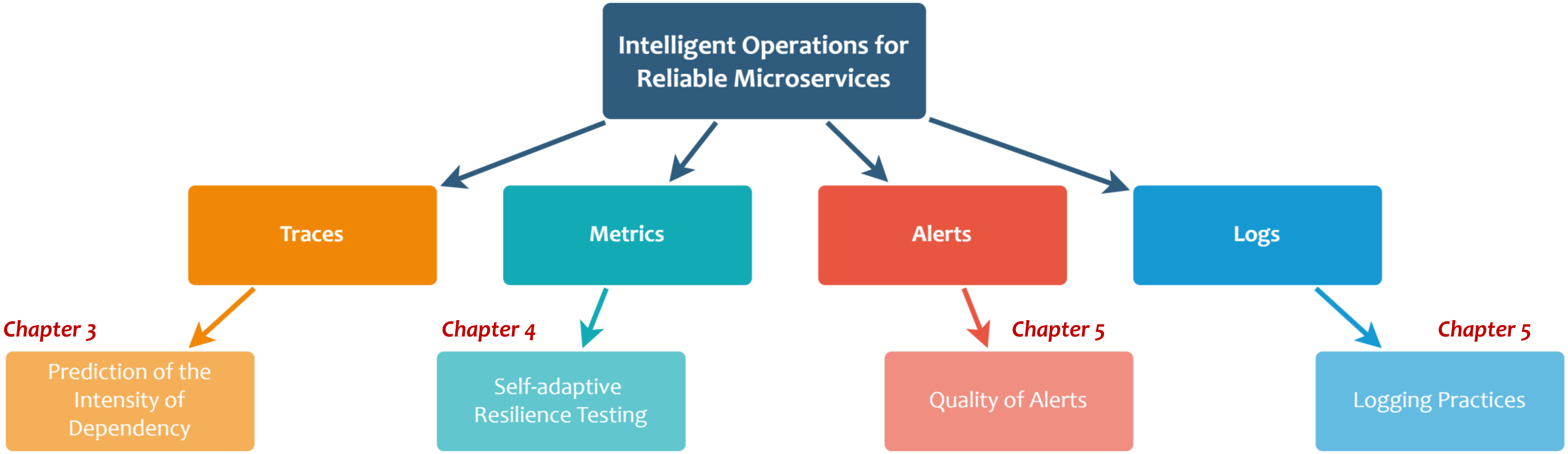
```
1 | 2008-11-09 20:55:54 PacketResponder 0 for block blk_321 terminating
2 | 2008-11-09 20:55:54 Received block blk_321 of size 67108864 from /10.251.195.70
3 | 2008-11-09 20:55:54 PacketResponder 2 for block blk_321 terminating
4 | 2008-11-09 20:55:54 Received block blk_321 of size 67108864 from /10.251.126.5
5 | 2008-11-09 21:56:50 10.251.126.5:50010:Got exception while serving  blk_321 to /10.251.127.243
6 | 2008-11-10 03:58:04 Verification succeeded for blk_321
7 | 2008-11-10 10:36:37 Deleting block blk_321 file /mnt/ hadoop/dfs/data/current/subdir1/blk_321
8 | 2008-11-10 10:36:50 Deleting block blk_321 file /mnt/ hadoop/dfs/data/current/subdir51/blk_321
```

- Alerts
  - Structured text notifications to call for immediate human intervention upon system anomalies.

| No. | Severity | Time | Service | Alert Title | Duration | Location |
|---|---|---|---|---|---|---|
| 1 | Major | 2021/05/18 06:36 | Block Storage | Failed to allocate new blocks, disk full | 10 min | Region=X;DC=1;... |
| 2 | Critical | 2021/05/18 06:38 | Database | Failed to commit changes ... | 2 min | Region=X;DC=1;... |
| 3 | Critical | 2021/05/18 06:39 | Database | Failed to commit changes ... | 5 min | Region=X;DC=1;... |

Alerts need to be promptly dealt with, but logs do not.

# Thesis Contribution




- The first empirical study on the intensity of dependency.
- The first method to quantify the intensity of microservice dependencies.
- Release an industrial dataset for reuse.

*[ASE'21]*

- The first empirical study on the failures of resilient and unresilient microservices.
- The first self-adaptive resilience testing framework.

*[ICSE'23]**

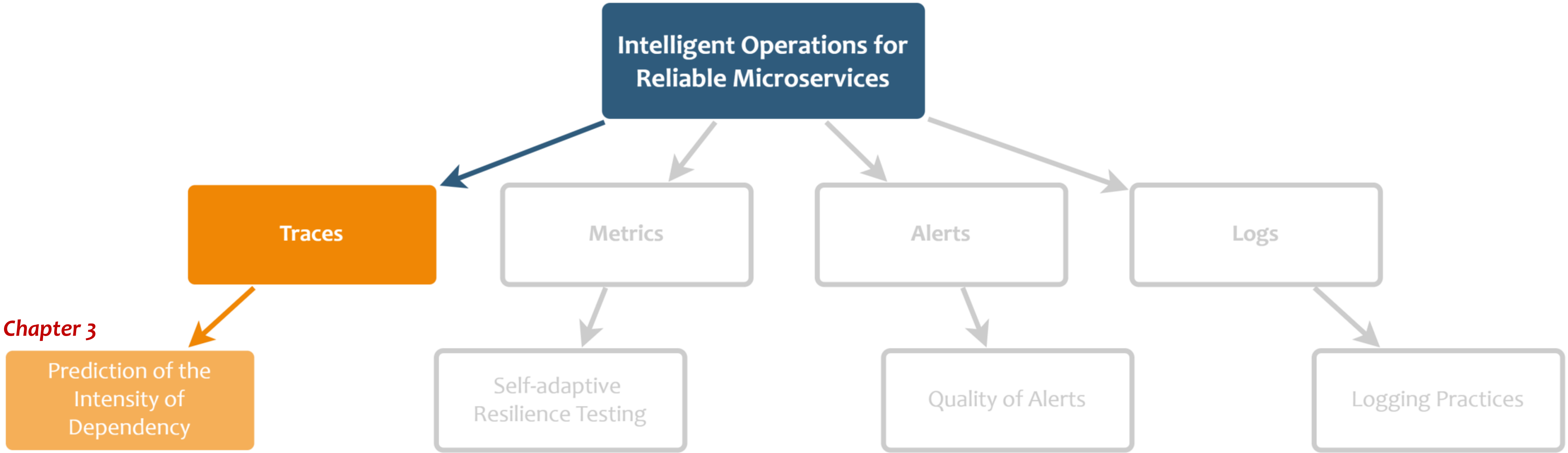
- Identify six antipatterns of alerts in a production cloud.
- Identify four postmortem reactions to antipatterns.
- Survey the current practice of logging for reliability.
- Propose directions on improving the quality of alerts and logs.

*[DSN'22, WWW'21]*

*[CSUR'21]*

** Under review by ICSE'23*

# Thesis Contribution



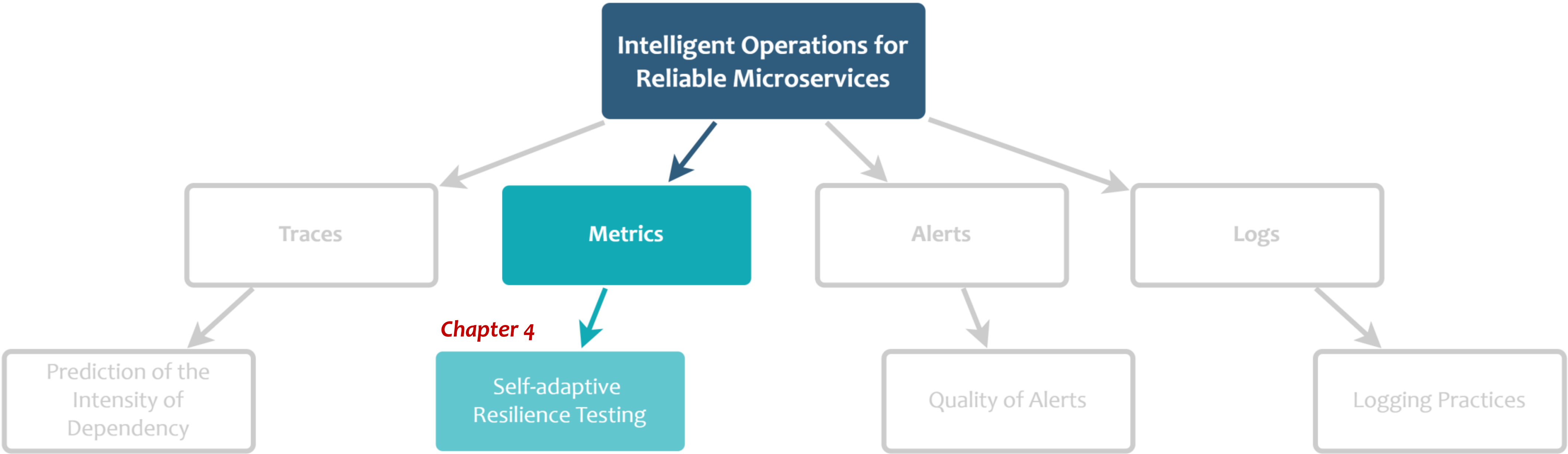

- The first empirical study on the intensity of dependency.
- The first method to quantify the intensity of microservice dependencies.
- Release an industrial dataset for reuse.

*[ASE'21]*

- The first empirical study on the failures of resilient and unresilient microservices.
- The first self-adaptive resilience testing framework.

- Identify six antipatterns of alerts in a production cloud.
- Identify four postmortem reactions to antipatterns.
- Survey the current practice of logging for reliability.
- Propose directions on improving the quality of alerts and logs.

# Thesis Contribution




- The first empirical study on the intensity of dependency.
- The first method to quantify the intensity of microservice dependencies.
- Release an industrial dataset for reuse.

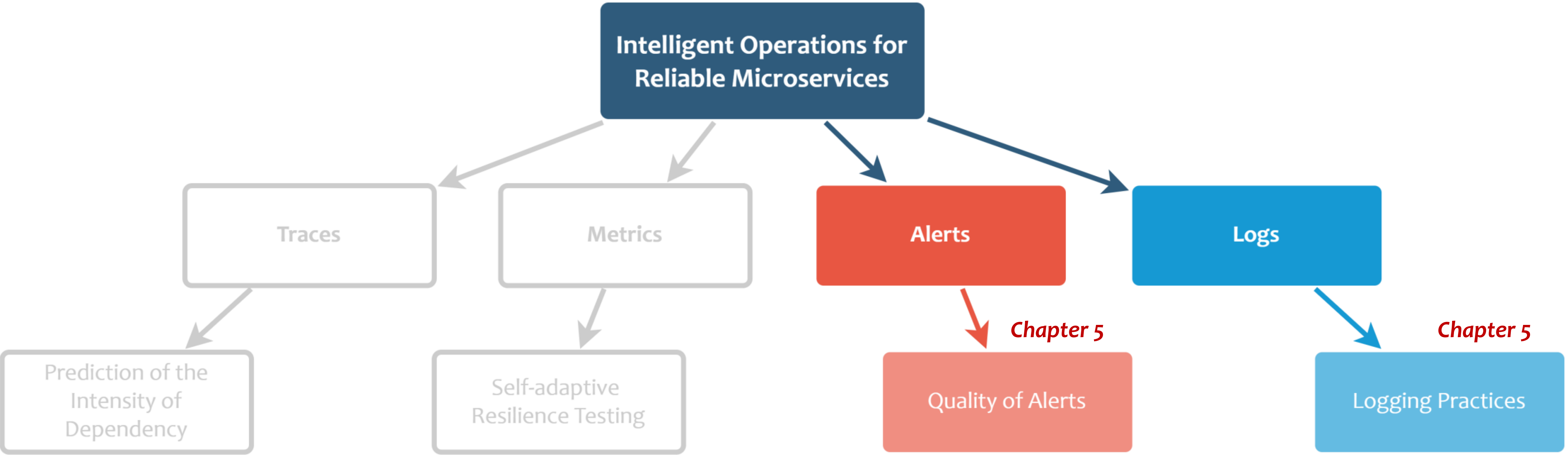
- The first empirical study on the failures of resilient and unresilient microservices.
- The first self-adaptive resilience testing framework.

[ICSE'23]*

- Identify six antipatterns of alerts in a production cloud.
- Identify four postmortem reactions to antipatterns.
- Survey the current practice of logging for reliability.
- Propose directions on improving the quality of alerts and logs.

** Under review by ICSE'23*

# Thesis Contribution




- The first empirical study on the intensity of dependency.
- The first method to quantify the intensity of microservice dependencies.
- Release an industrial dataset for reuse.

- The first empirical study on the failures of resilient and unresilient microservices.
- The first self-adaptive resilience testing framework.

- Identify six antipatterns of alerts in a production cloud.
- Identify four postmortem reactions to antipatterns.
- Survey the current practice of logging for reliability.
- Propose directions on automatic alert governance and improving the quality of logs.

*[DSN'22, WWW'21]*

*[CSUR'21]*

# CONTENTS

2

# Predicting the Intensity of Dependency

# A Survey of the Outages in AWS

# AWS Post-Event Summaries

### AWS Post-Event Summaries

The following is a list of post-event summaries from major service events that impacted AWS service availability:

- Summary of the Amazon Kinesis Event in the Northern Virginia (US-EAST-1) Region, November, 25th 2020
- Summary of the Amazon EC2 and Amazon EBS Service Event in the Tokyo (AP-NORTHEAST-1) Region, August 23, 2019
- Summary of the Amazon EC2 DNS Resolution Issues in the Asia Pacific (Seoul) Region (AP-NORTHEAST-2), November 24, 2018.
- Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region, February 28, 2017.
- Summary of the AWS Service Event in the Sydney Region, June 8, 2016.
- Summary of the Amazon DynamoDB Service Disruption and Related Impacts in the US-East Region, September 20, 2015.
- Summary of the Amazon EC2, Amazon EBS, and Amazon RDS Service Event in the EU West Region, August 7, 2014.
- Summary of the Amazon SimpleDB Service Disruption, June 13, 2014.
- Summary of the December 17th event in the South America Region (SA-EAST-1), December 20, 2013.
- Summary of the December 24, 2012 Amazon ELB Service Event in the US-East Region, December 24, 2012.
- Summary of the October 22, 2012 AWS Service Event in the US-East Region, October 22, 2012.
- Summary of the AWS Service Event in the US East Region, July 2, 2012.
- Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region, April 29, 2011.

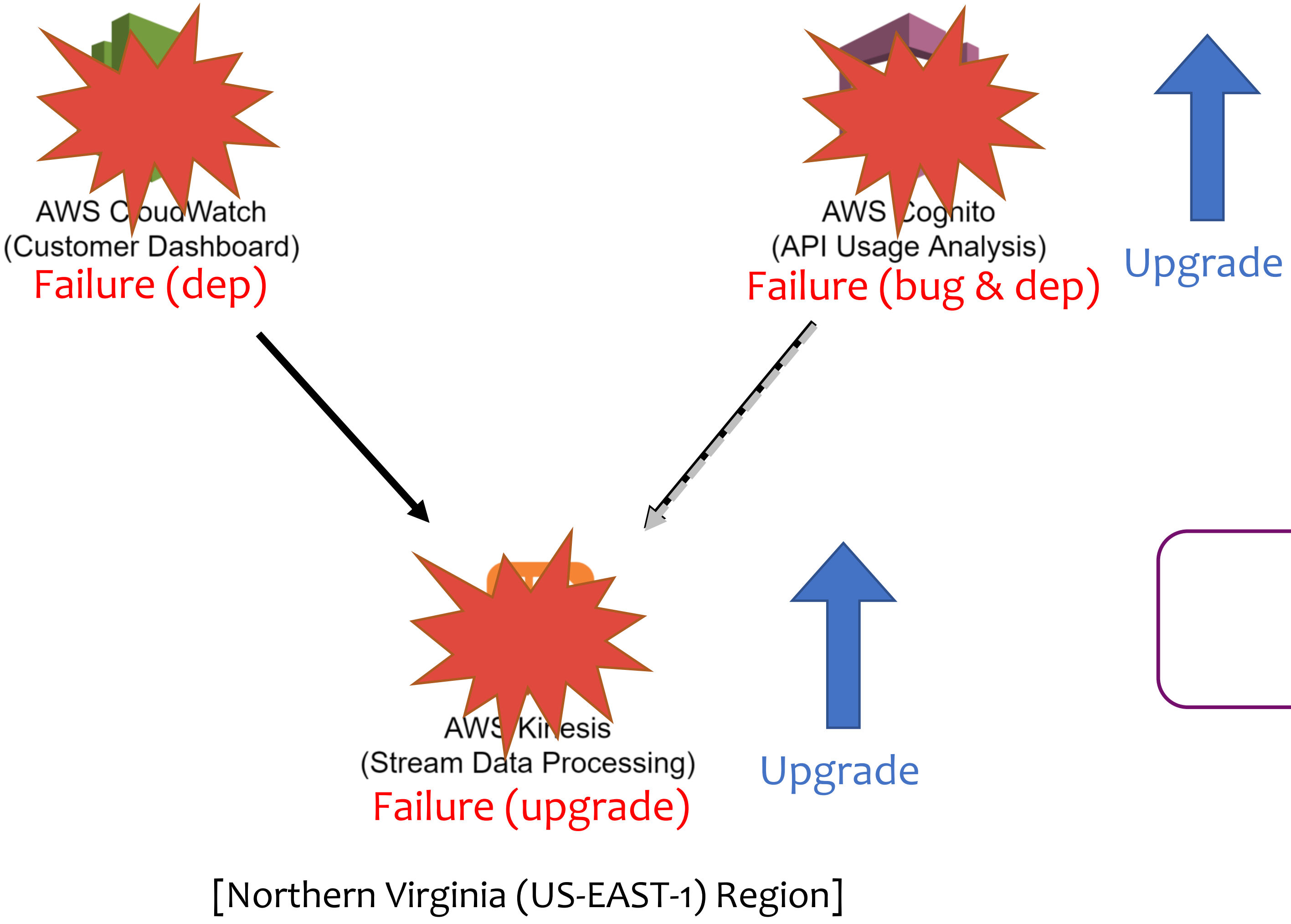
Slow Recovery

Cascading failure

**5 out of 13** Amazon Web Service (AWS) outages are related to service dependency!

AWS Post-Event Summaries

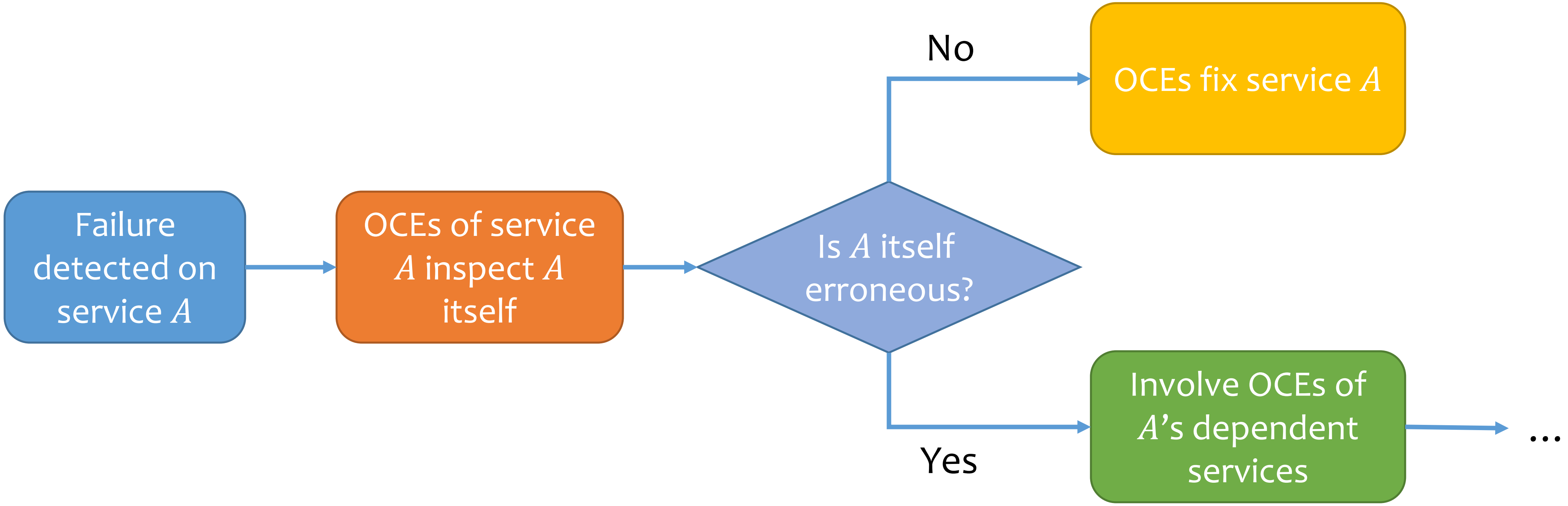
# AWS Kinesis Event on Nov 25th, 2020




Reduced dependency can accelerate failure recovery.

[Summary of the Amazon Kinesis Event in the Northern Virginia (US-EAST-1) Region]

# Drawbacks of Current Failure Diagnosis Methods



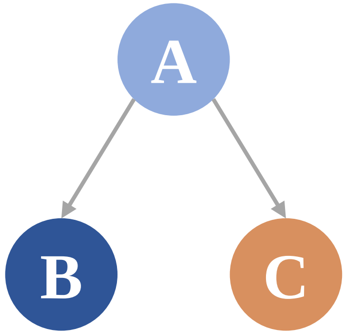

*Because each team only have a local view of the whole system.*

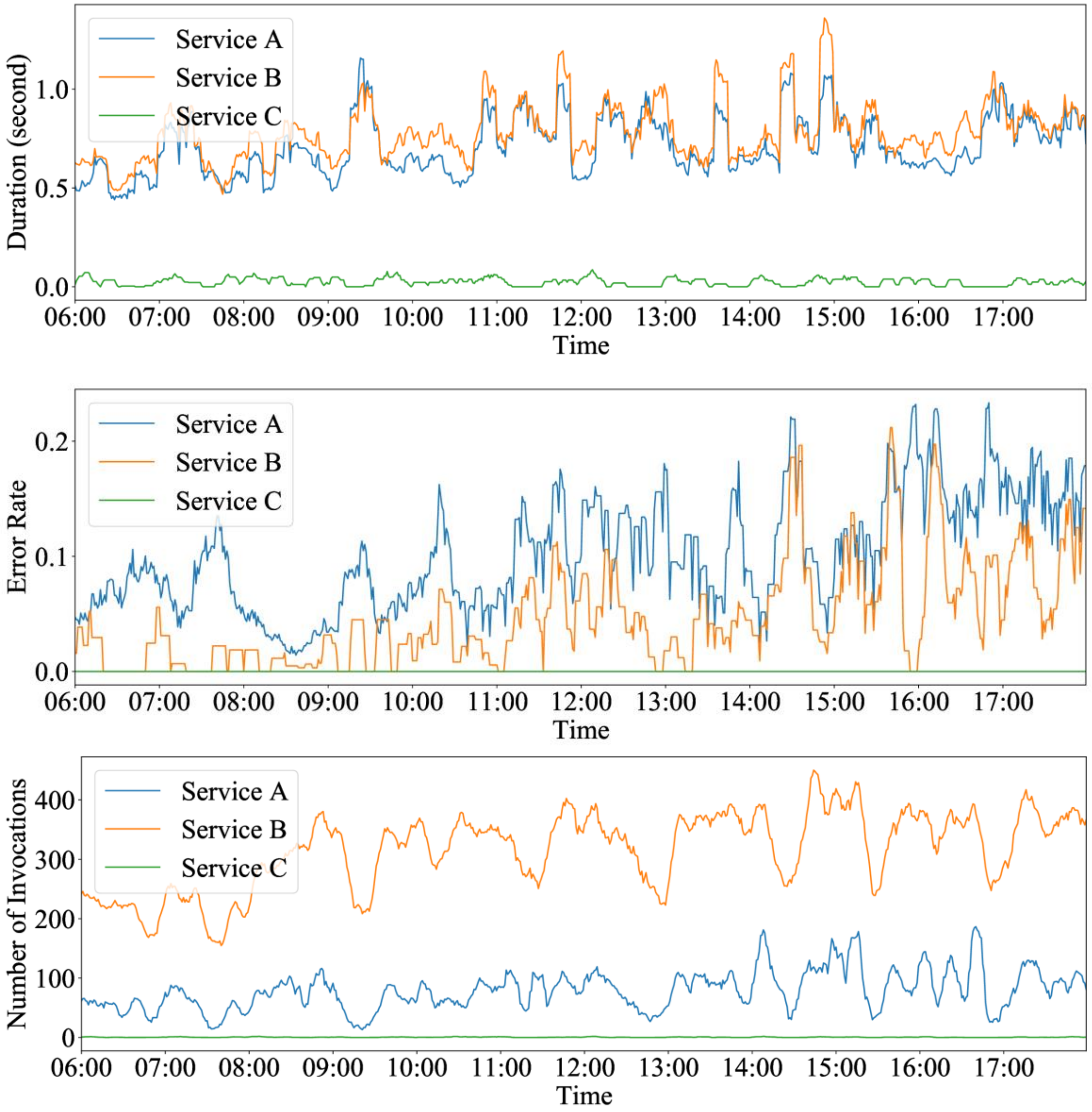
Current practice is inefficient and dependent on the human experience.

* OCE: On-Call Engineer

# Intensity of Service Dependency

- The *intensity of dependency* between $A \rightarrow B$ is higher than the intensity of dependency between $A \rightarrow C$, due to
  - Functionality
  - Fault tolerance



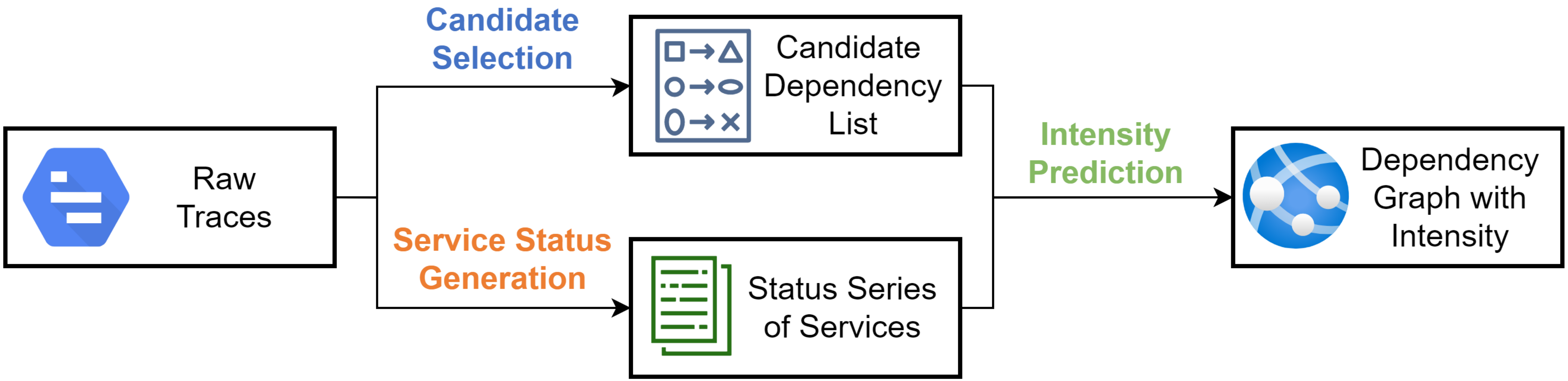



# Intensity of Service Dependency

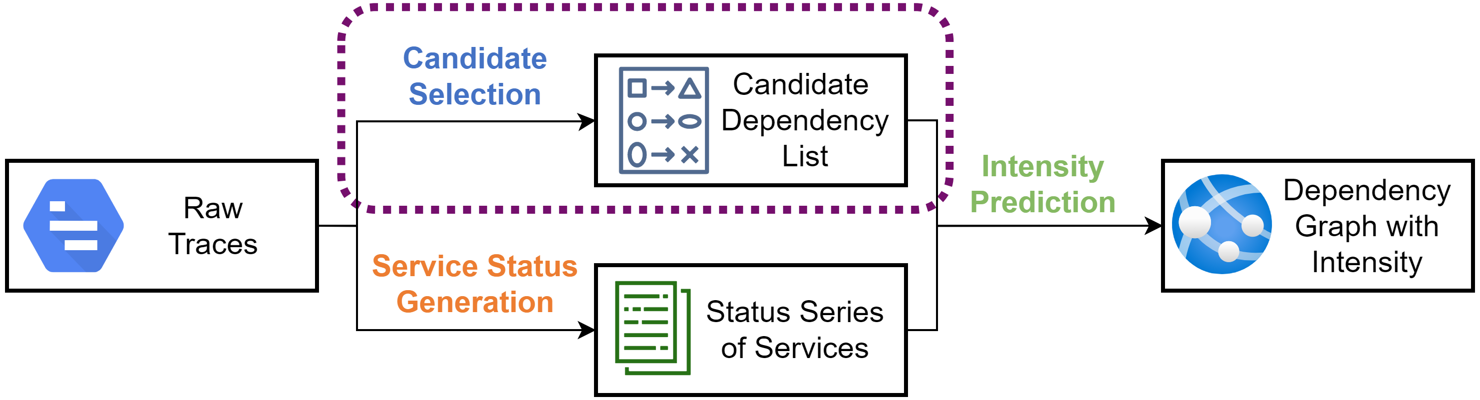
*We define the intensity of dependency between two services as how much the status of the callee service influences the status of the caller service.*

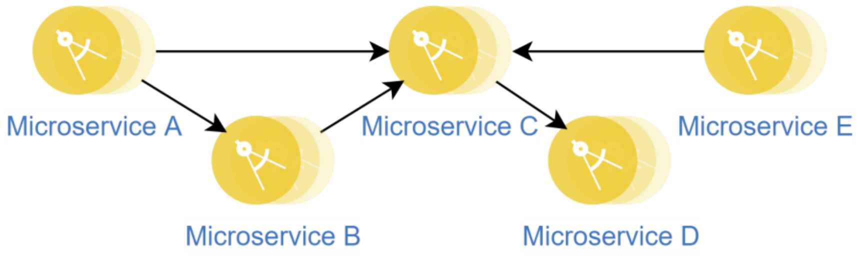
- Intensity is inherently determined by the program logic of services.
- Manual maintenance of intensity is hard due to the fast-evolving nature.
- But we could **predict** the intensity of dependency from traces.

# AID: Predicting the Aggregated Intensity of Dependency



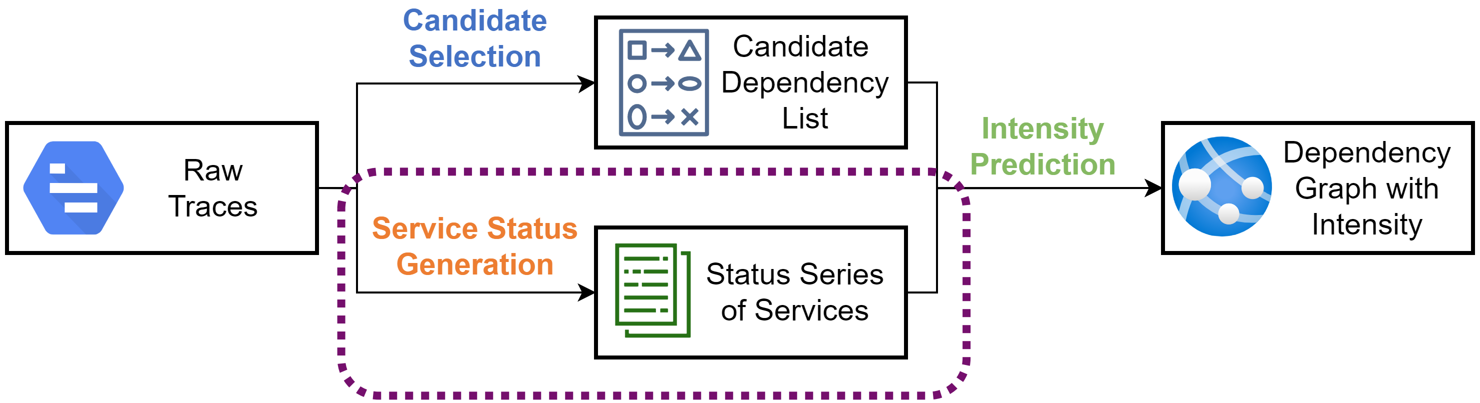
# AID: Candidate Selection

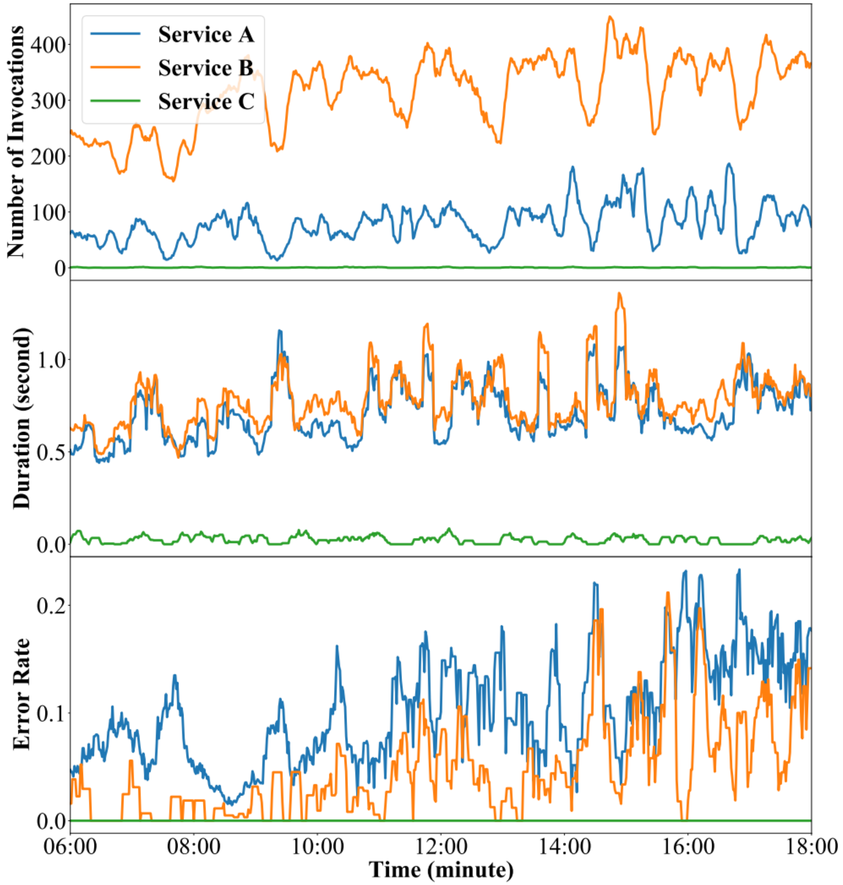



- Objective
  - Select the candidate invocation pairs $(caller, callee)$ from raw traces where $caller$ directly invokes $callee$.
- Method
  - Iterate over all spans to get the invocation pairs.
  - Get the invocation pairs if the cloud system have a centralized database of invocation.

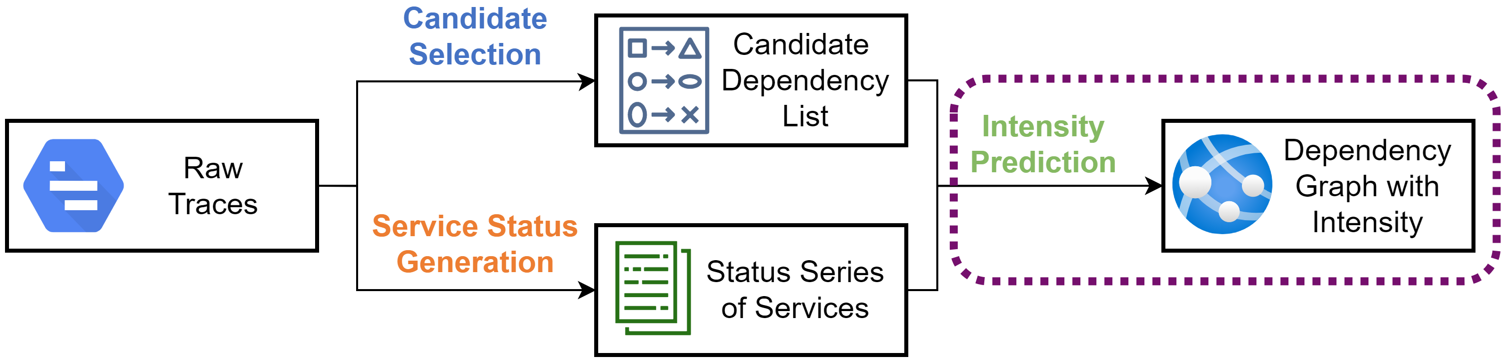
# AID: Service Status Series Generation

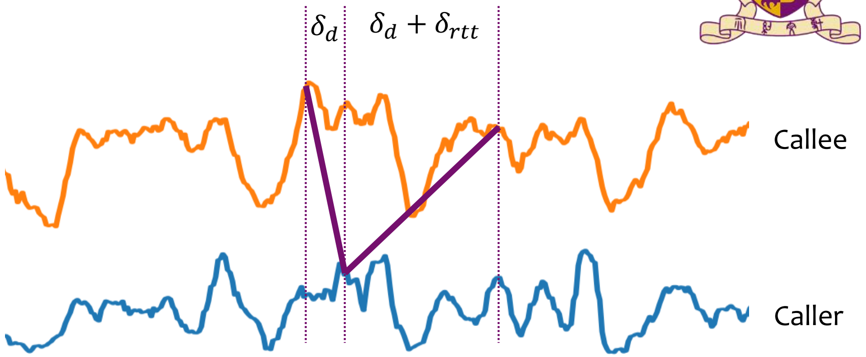



- Three aspects of indicators of service status
  - *Number of Invocations*
  - *Durations of Invocations*
  - *Error of Invocations*
- Method: calculate the number of invocations, average duration, and error rate of all spans of a service in a fixed time interval (e.g., 1 minute).

# AID: Intensity Prediction




- Idea: the more similar two services' status series are, the higher the intensity is.
- Method
  - Dynamic Status Warping.
  - Similarity Normalization & Aggregation.

$$d_{status}^{(P_i,C_i)} = \frac{d_{status}^{(P_i,C_i)} - \min(d_{status}^{(P,C)})}{\max(d_{status}^{(P,C)}) - \min(d_{status}^{(P,C)})} \qquad I^{(P_i,C_i)} = \frac{1}{3}\sum_{status \in S} d_{status}^{(P_i,C_i)},\ S = \{invo, err, dur\}$$

**Algorithm 1:** Dynamic Status Warping

**Input:** The status series of caller service and callee service $status^P$, $status^C$; duration series of callee $dur^C$, estimated round trip time $\delta_{rtt}$, max time drift $\delta_d$

**Output:** The similarity between two status series

Set the warping window $w = \max(dur^C) + \delta_{rtt}$
$K = length(status^C)$
$N = length(status^P)$
Initialize the cost matrix $\mathbf{C} \in \mathbb{R}^{K \times N}$, set the initial values as $+\infty$
$\mathbf{C}_{1,1} = (status_1^P - status_1^C)^2$
**for** $i = 2 \ldots \min(\delta_d, K)$ **do** // Initialize the first column
  $\mathbf{C}_{i,1} = \mathbf{C}_{i-1,1} + (status_1^P - status_i^C)^2$
**end**
**for** $j = 2 \ldots \min(w + \delta_d, N)$ **do** // Initialize the first row
  $\mathbf{C}_{1,j} = \mathbf{C}_{1,j-1} + (status_j^P - status_1^C)^2$
**end**
**for** $i = 2 \ldots K$ **do**
  **for** $j = \max(2, i - \delta_d) \ldots \min(N, i + w + \delta_d)$ **do**
    $\mathbf{C}_{i,j} = \min(\mathbf{C}_{i-1,j-1}, \mathbf{C}_{i-1,j}, \mathbf{C}_{i,j-1}) + (status_j^P - status_i^C)^2$
  **end**
**end**
**return** $\mathbf{C}_{K,N}$

# Experiment Settings

- Dataset
  - *Industry*[1] : Production Huawei Cloud traces.
  - *TT*[2] : Simulated traces by the Train-Ticket benchmark.
- Manual labeling
  - *Industry* : By engineers of Huawei Cloud.
  - *TT* : By two PhD students familiar with the benchmark.

DATASET STATISTICS.

| Dataset | TT | Industry |
|---|---|---|
| # Microservices | 25 | 192 |
| # Spans | 17,471,024 | About 1.0e10 |
| # Strong | 18 | 67 |
| # Weak | 1 | 8 |

[1] We only labeled 75 dependencies that the engineers are familiar with.
[2] FudanSELab/train-ticket: Train Ticket - A Benchmark Microservice System (github.com)

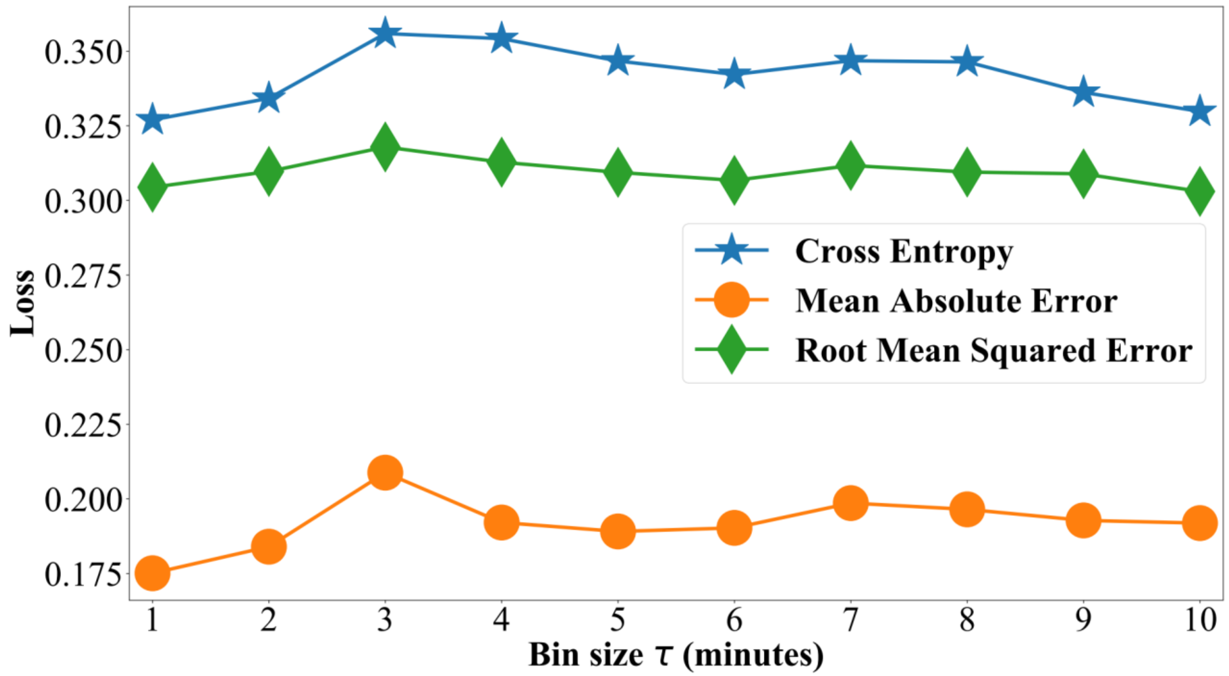
# Effectiveness of Intensity Prediction

PERFORMANCE COMPARISON OF DIFFERENT METHODS ON TWO DATASETS

| Dataset | Method | Metric | | |
|---|---|---|---|---|
| | | CE | MAE | RMSE |
| TT | Pearson | 0.6872 | **0.3305** | 0.4388 |
| | Spearman | 0.7512 | 0.3735 | 0.4697 |
| | Kendall | 0.6464 | 0.3749 | 0.4577 |
| | AID | **0.4562** | 0.3435 | **0.3859** |
| Industry | Pearson | 0.6076 | 0.4524 | 0.4563 |
| | Spearman | 0.6030 | 0.4501 | 0.4537 |
| | Kendall | 0.6258 | 0.4636 | 0.4656 |
| | AID | **0.3270** | **0.1751** | **0.3044** |

- Parameter Settings
  - Bin size $\tau = 1\ min$
  - Estimated round trip time $\delta_{rtt} = 0$
  - Max time drift
    - $\delta_d = 1\ min$ (for Industry dataset)
    - $\delta_d = 0\ min$ (for TT dataset)

# Ablation Study

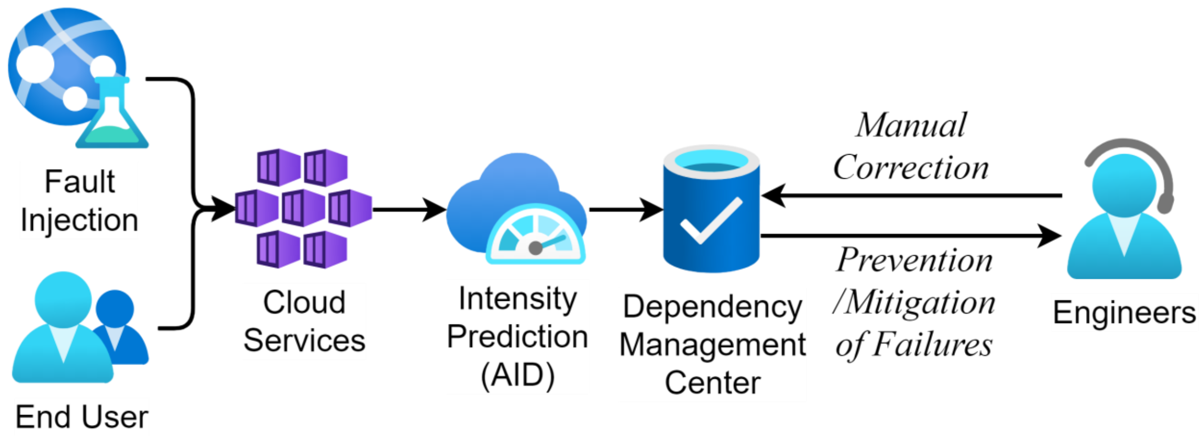



THE IMPACT OF DIFFERENT SIMILARITY MEASURES

| Dataset /Bin size | Method | Metric | | |
|---|---|---|---|---|
| | | CE | MAE | RMSE |
| TT /1min | $AID_{DSW}$ | 0.4562 | 0.3435 | 0.3859 |
| | $AID_{DTW}$ | 0.4494 | 0.3467 | 0.3832 |
| Industry /1min | $AID_{DSW}$ | 0.3270 | 0.1751 | 0.3044 |
| | $AID_{DTW}$ | 0.3584 | 0.1996 | 0.3169 |

# Use Cases of AID

- Mitigation of Cascading Failures
  - Limit the traffic to critical cloud services.
  - Recover the dependencies marked as "strong" first.
- Optimization of Dependencies
  - Dependency management system detects strong dependencies and reminds engineers.
  - Discovered more than ten unnecessary dependencies within four months.




***Already deployed in*** HUAWEI

# Summary of Chapter 3

First to identify the concept of aggregated intensity of dependency for failure diagnosis and failure recovery.

First method to quantify the intensity of dependencies between different services.

Experiments on simulated & industrial datasets show its effectiveness and efficiency.

Successfully deployed in Huawei Cloud.

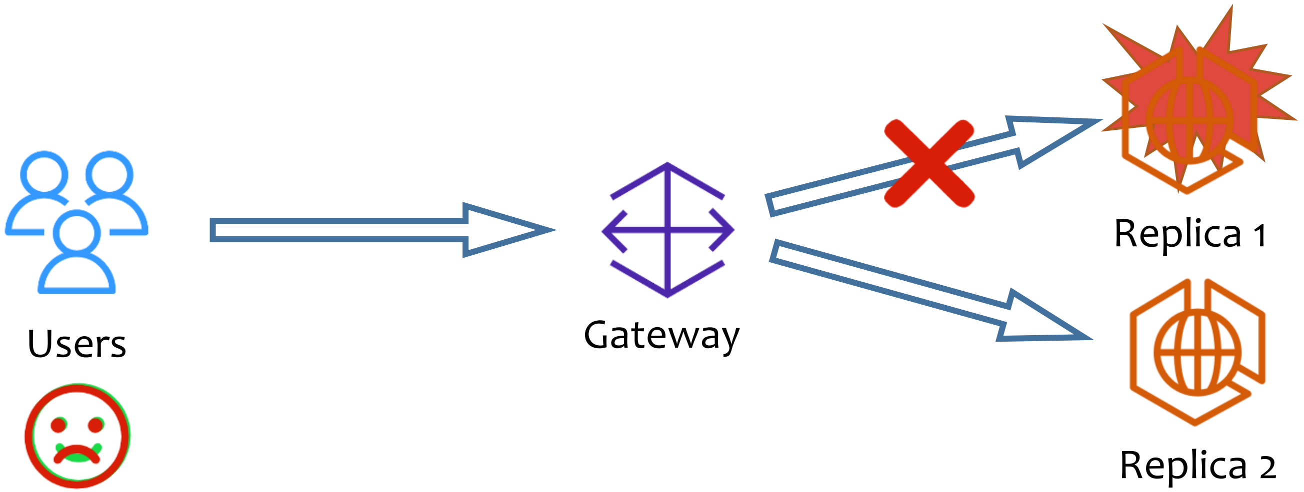
# CONTENTS

3

# Self-adaptive Microservice Resilience Testing

# Resilience of Online Services

*Resilience: the ability to maintain performance at an acceptable level and recover the service back to normal under service failures.*



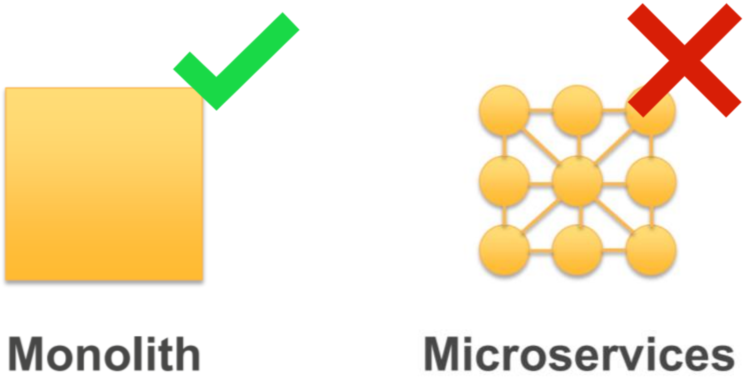

[Test software resiliency - IBM Garage Practices]

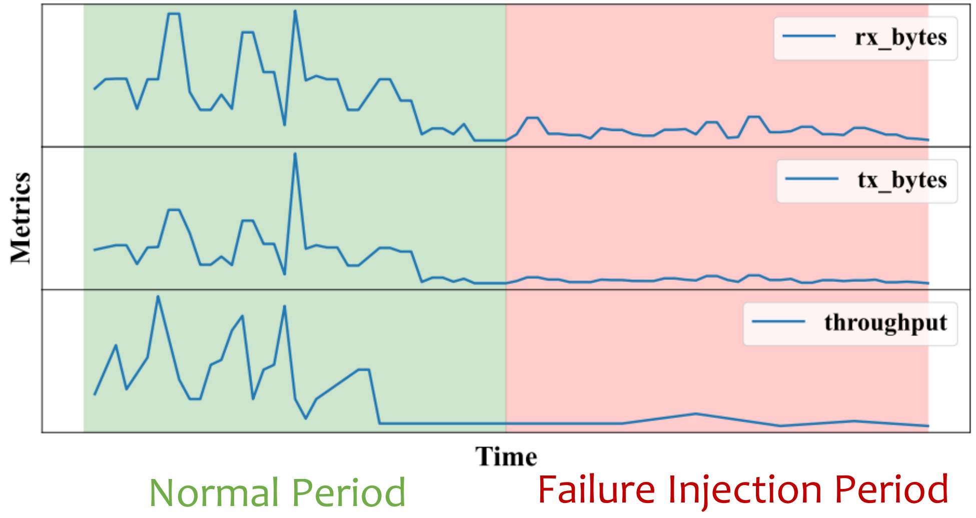
# Current Practice for Resilience Testing

Set rules → Inject failures → Evaluate results

| Failure type | Network jam |
|---|---|
| Metrics to monitor | Rx_bytes, tx_bytes, throughput |
| Passing criteria | Request throughput recover within 5 minutes |

An example rule set



# Issues of Current Practice: Scalability

- Scalability Issue
  - Manual identification of the failure rule sets relies heavily on domain expertise.
  - Fast-evolving nature of microservices requires frequent updates of failure rule sets.


Manual identification of failure rule sets does not scale.

# Issues of Current Practice: Adaptivity

- Adaptivity Issue
  - PASS/FAIL cannot depict the subtle difference in an online service's resilience.
  - Reasons
    - The impact of a failure is diversiform in a microservice system.
    - Online services can be in a gray-failure status instead of fail as a whole.

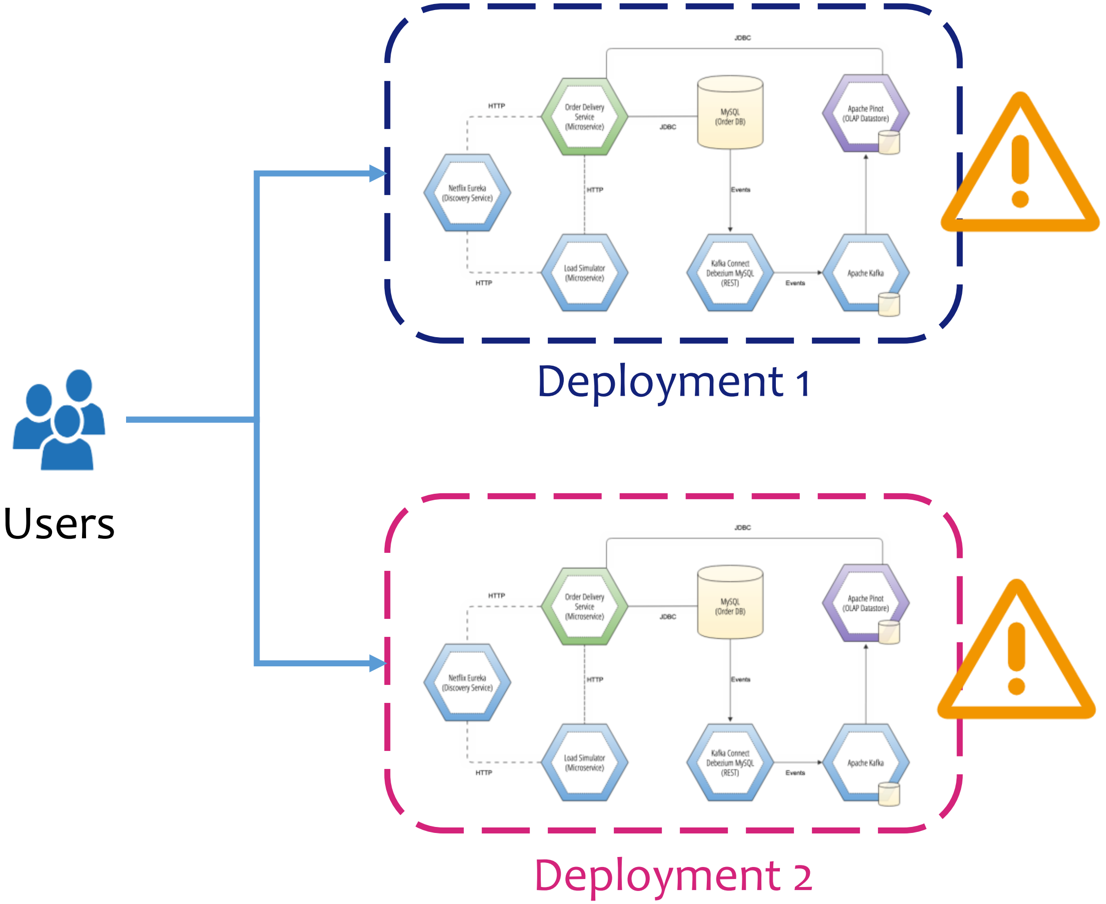


Defining fixed failure rule sets for evaluating resilience is inadaptive.

# Characteristics of Resilient Microservices

- Inject failures into two deployments of the same microservice benchmark system.
  - One **with common resilience measures**
  - One **without common resilience measures**

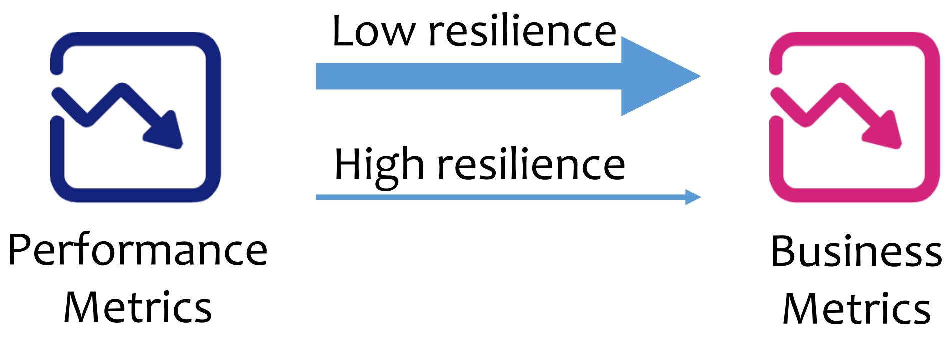
- Compare the manifestation of failures on the two deployments.



# Characteristics of Resilient Microservices

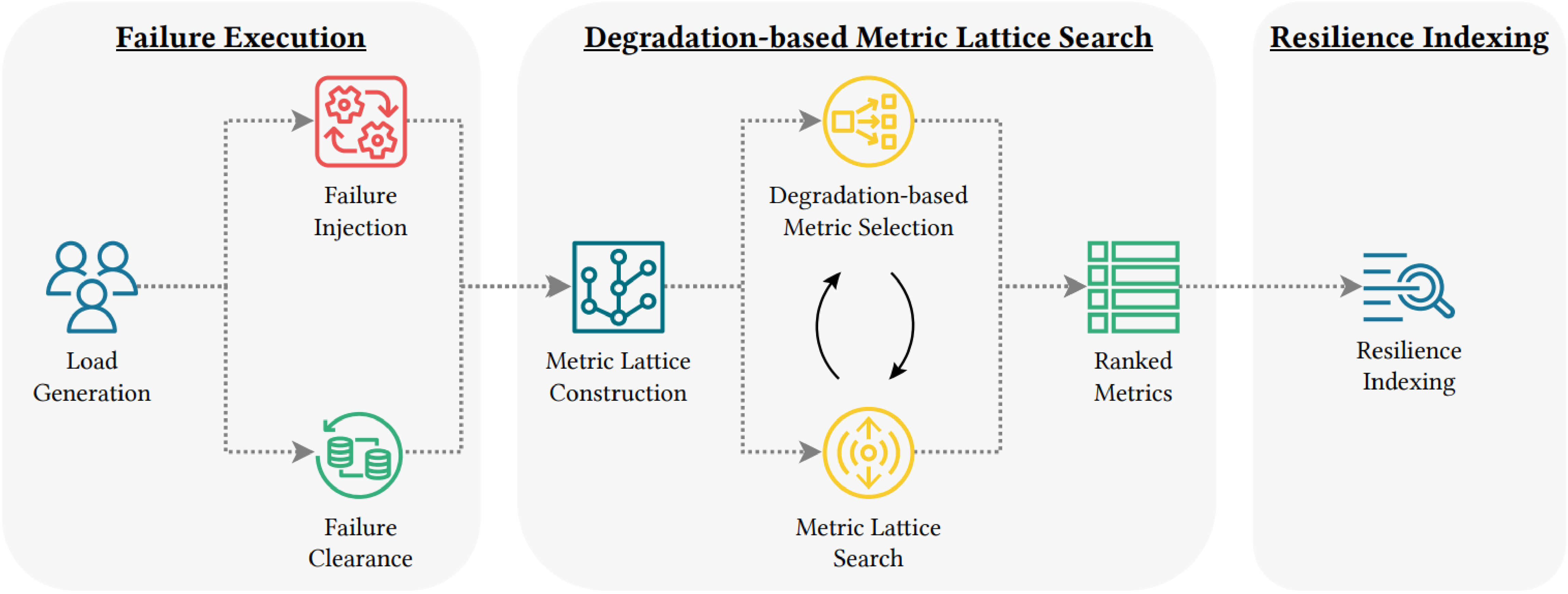
- Service degradation manifests the impact of the injected failures.
  - Measured by the performance difference between the normal period and the fault-injection period.

- Insight
  - The less degradation propagation from **system performance metrics** to **business metrics**,
  - The higher the resilience.




| Failure | Degradation w/o resilience mechanisms | Degradation w/ resilience mechanisms |
|---|---|---|
| Container CPU overload | High container CPU usage, slow response speed | Decreased but acceptable response speed |
| Container TCP disconnection | Connection error within container | Return to normal response speed shortly |
| Container instance killed | Instance offline, unresponsive microservice endpoint | Response normally after some time |
| (More in the thesis) …… | | |

# AVERT: A Self-adaptive Resilience Testing Framework

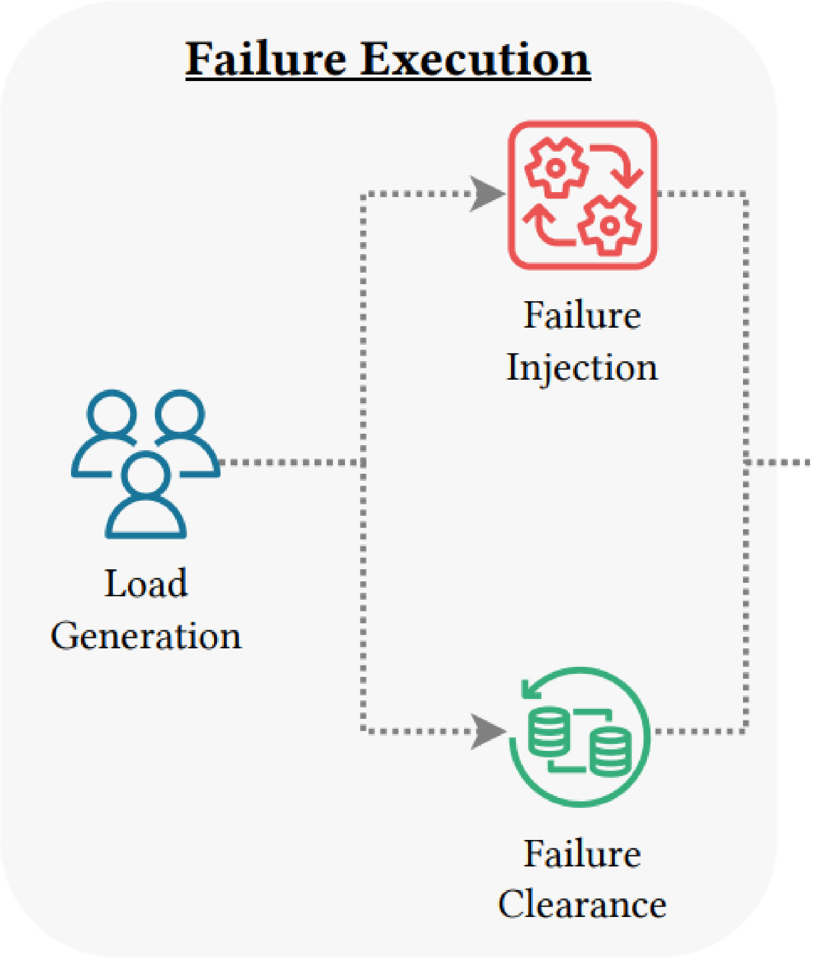



Collect monitoring metrics.

Rank the metrics by degradation.

Index the resilience.

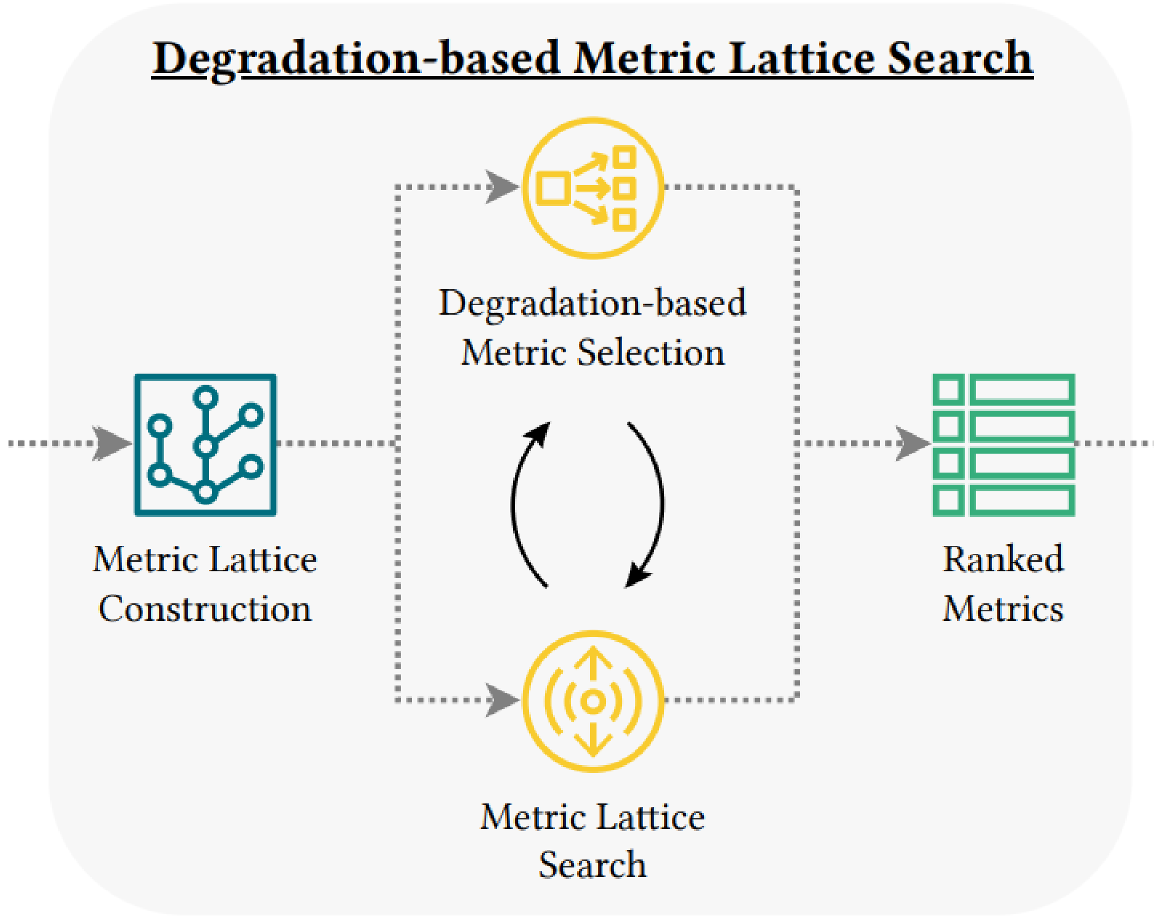
# AVERT: Failure Execution



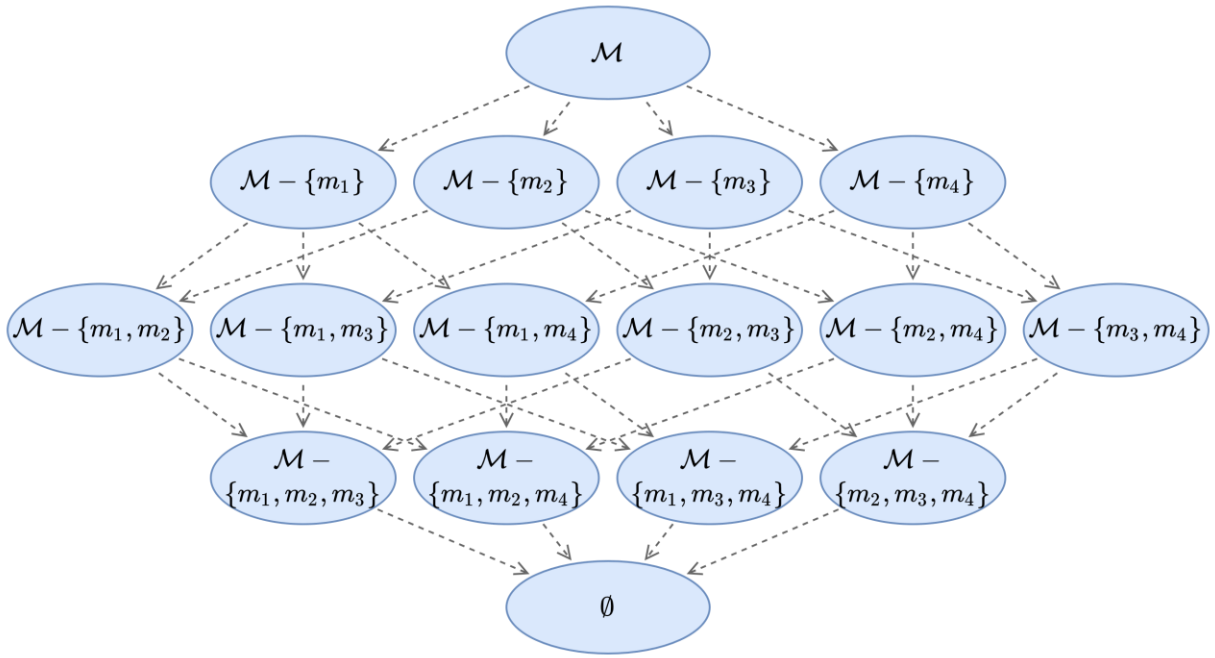

- Two phases for each type of failure.
  - Failure injection & Failure clearance.
- Data collected
  - Two types of metrics
    - Business metrics $\boldsymbol{B}$
    - System performance metrics $\boldsymbol{P}$
- Denote all metrics as $\boldsymbol{M}$

$$\boldsymbol{M} = \boldsymbol{B} \cup \boldsymbol{P} = \{m_1, m_2, \ldots, m_M\}$$
$$\exists i, m_i \in \boldsymbol{B} \vee m_i \in \boldsymbol{P}$$

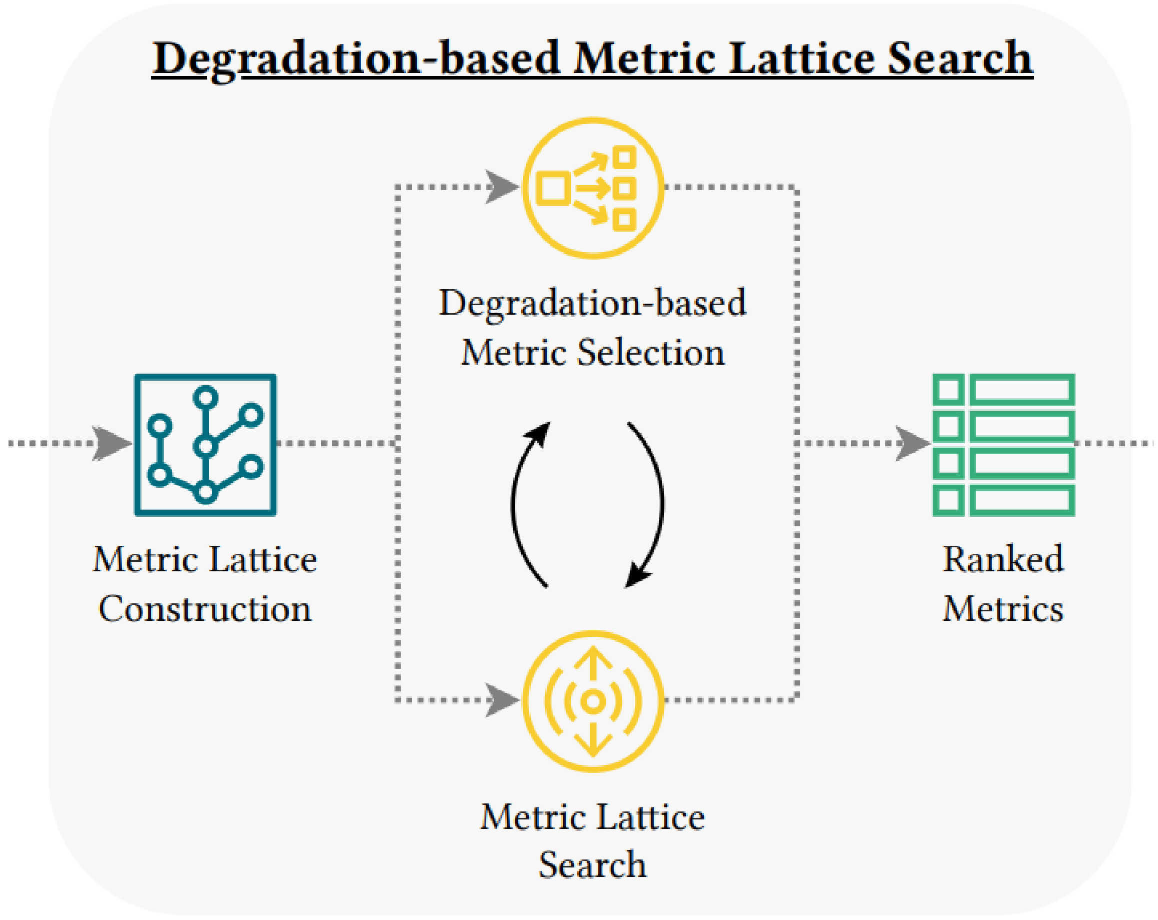
# AVERT: Degradation-based Metric Lattice Search



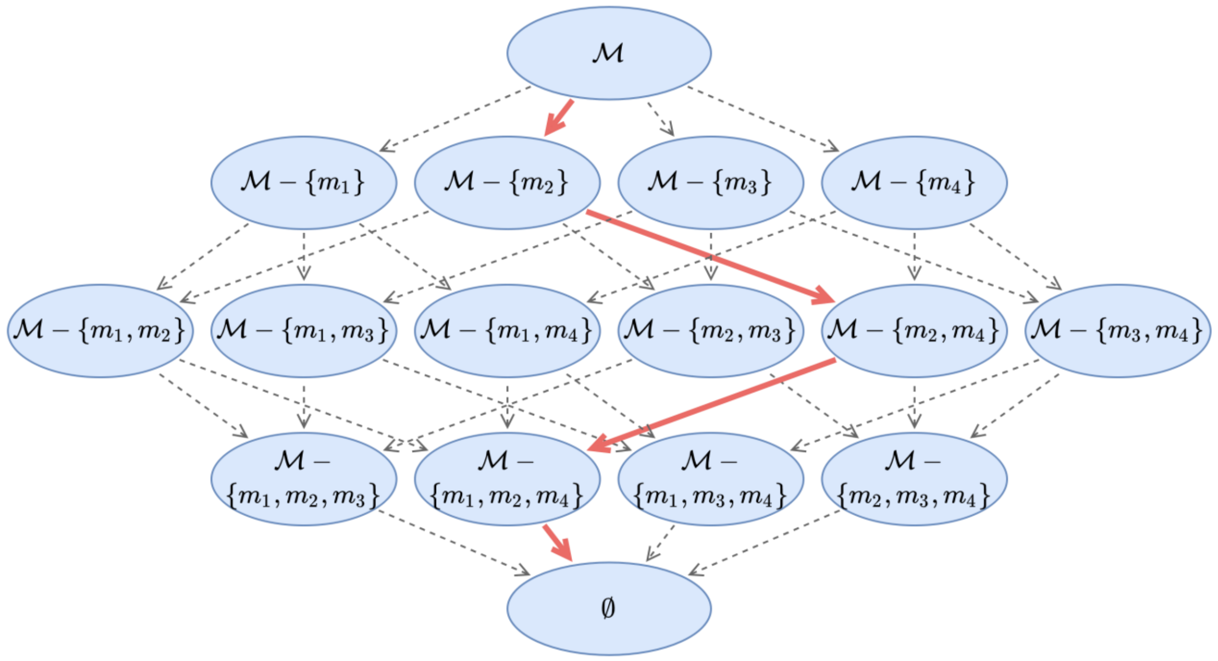

- Construct the metric lattice from the power set of $\boldsymbol{M}$.
  - Each node is a subset of $\boldsymbol{M}$.
  - Ordered by the subset-superset relation.



# AVERT: Degradation-based Metric Lattice Search

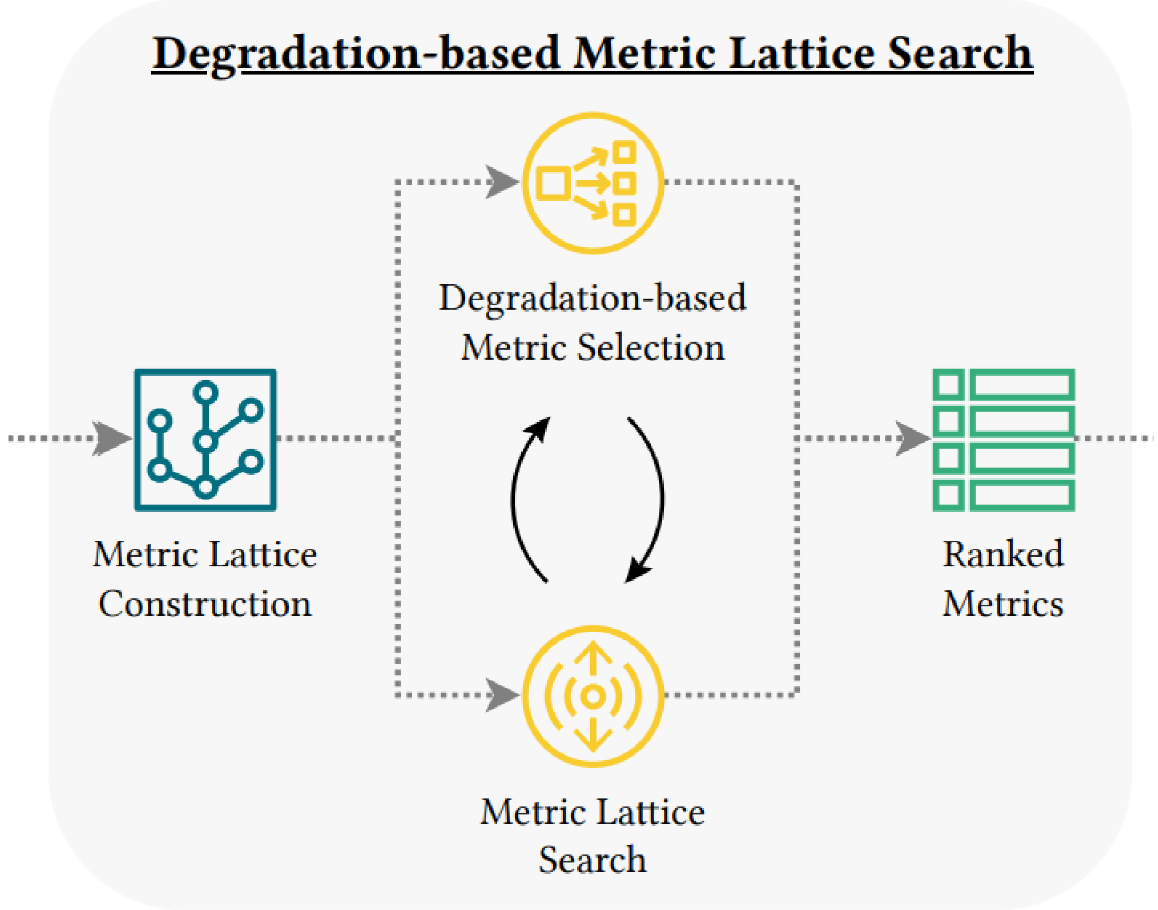



- Idea
  - Depth-first search from the upmost node to the bottommost node.
  - Select the metric that contributes most to the overall service degradation.




Please check the detailed algorithm in the thesis.

# AVERT: Degradation-based Metric Lattice Search




**Algorithm 3:** Degradation-based Metric Selection

**Input:** The monitoring metric subset $\mathcal{M}'$; The monitoring metrics during the failure injection period $\mathcal{M}'^f$; The monitoring metrics during the failure clearance period $\mathcal{M}'^n$

**Output:** The metric $m_i \in \mathcal{M}'$ where $m_i$ contribute most to the overall service degradation

```
Function MetricSelection(M′, M′^f, M′^n):
    T = length of the monitoring metrics
    D = []
    for m_i ∈ M′ do
        // Compute the performance difference of each individual metric
        for t = 1...T do
            δ_i(t) = |m_i^f(t) − m_i^n(t)|
        end
        δ̂_i = δ_i − δ̄_i // Normalize δ_i
        D = [D; δ̂_i] // Concatenate the normalized performance difference
    end
    δ_PC1 = PCA(D, dim = 1) // Reduce to one dimension via Principal Component Analysis
    // Select the metric that contribute most to the performance difference
    for δ̂_i ∈ D do
        c_i = Contribution(δ_PC1, δ̂_i)
    end
    cmax = max(c_i)
    imax = arg max_i(c_i)
    return cmax, m_imax
End
```

Compute performance degradation

Select the metric with highest contribution

Please check the detailed algorithm in the thesis.

# AVERT: Resilience Indexing


- Idea
  - If the degradation of system performance metrics cannot **propagate** to the degradation of business metrics, the resilience is higher.
- Approach
  - Calculate the degradation contributed by $\boldsymbol{B}$ and $\boldsymbol{P}$.

$$D_{\mathcal{P}} = \sum_{m_i \in \mathcal{P}} \frac{c_i}{\log_2(rank(m_i; \mathcal{L}) + 1)}$$

$$D_{\mathcal{B}} = \sum_{m_i \in \mathcal{B}} \frac{c_i}{\log_2(rank(m_i; \mathcal{L}) + 1)}$$

  - Calculate the propagation.

$$r = \frac{1}{1 + e^{D_{\mathcal{B}} - D_{\mathcal{P}}}}$$

# Experiment Settings

- Dataset
  - *TT*[1]
    - The Train-Ticket benchmark
    - Env: Kubernetes
    - No. of failures: 24
  - *SN*[2]
    - The Social-Network benchmark
    - Env: docker compose
    - No. of failures 10

| **Dataset** | $\lvert\mathcal{B}\rvert$ | $\lvert\mathcal{P}\rvert$ | *#Microservices* | *#Failures* |
|---|---|---|---|---|
| *Train-Ticket* | 30 | 209 | 15 | 24 |
| *Social-Network* | 50 | 325 | 25 | 10 |

- Manual labeling of resilience
  - Done by two PhD students.
  - Verified by experienced engineers of Huawei.

[1] FudanSELab/train-ticket: Train Ticket - A Benchmark Microservice System (github.com)
[2] delimitrou/DeathStarBench: Open-source benchmark suite for cloud microservices (github.com)

# Effectiveness

Table 4.3: Performance Comparison of AVERT on Two Datasets

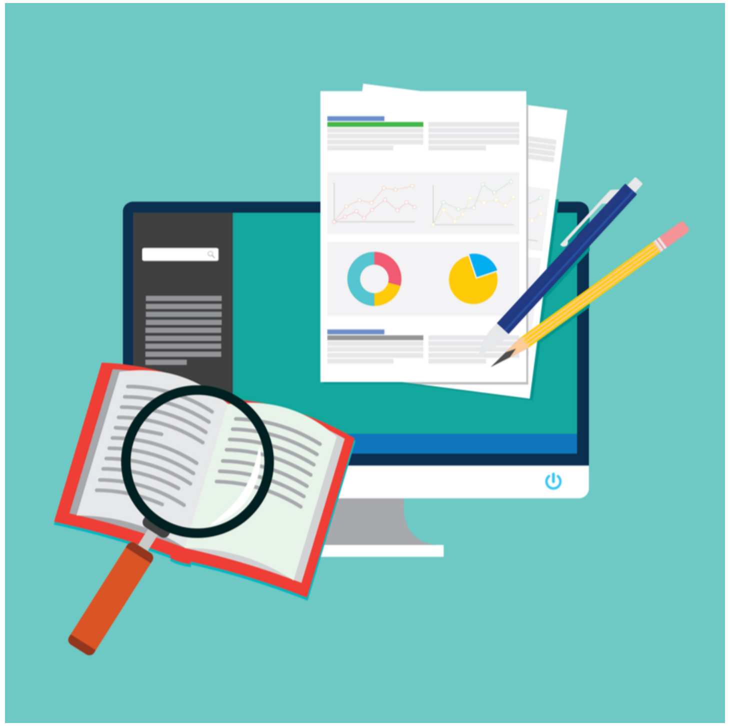
| Method | *Train-Ticket* | | | *Social-Network* | | |
|---|---|---|---|---|---|---|
| | *CE* | *MAE* | *RMSE* | *CE* | *MAE* | *RMSE* |
| SVC | 0.8864 | 0.4875 | 0.5594 | 0.7483 | 0.4426 | 0.5165 |
| RF | 0.6973 | 0.4259 | 0.5005 | 0.5646 | 0.3787 | 0.4416 |
| ET | 0.8766 | 0.4682 | 0.5470 | 0.6546 | 0.4199 | 0.4893 |
| **AVERT** | **0.1775** | **0.1572** | **0.1842** | **0.1159** | **0.1078** | **0.1203** |

Table 4.4: Ablation Study of AVERT on Two Datasets

| Method | *Train-Ticket* | | | *Social-Network* | | |
|---|---|---|---|---|---|---|
| | *CE* | *MAE* | *RMSE* | *CE* | *MAE* | *RMSE* |
| AVERT-euc | 0.3379 | 0.2735 | 0.3067 | 0.1874 | 0.1655 | 0.1905 |
| AVERT-corr | 0.2320 | 0.1985 | 0.2296 | 0.2532 | 0.2148 | 0.2449 |
| AVERT-cid | 0.1784 | 0.1589 | **0.1810** | 0.3131 | 0.2542 | 0.2933 |
| **AVERT-dtw** | **0.1775** | **0.1572** | 0.1842 | **0.1159** | **0.1078** | **0.1203** |

# Use Cases of AVERT

- Automatic Resilience Indexing
- Selection of the Vulnerable Metrics



# Summary of Chapter 4

An empirical study to demonstrate the feasibility of self-adaptive resilience testing for microservice systems.

First self-adaptive resilience testing framework, AVERT, that can automatically index the resilience of a microservice system to different failures.

AVERT measures the degradation propagation from system performance metrics to business metrics. The higher the propagation, the lower the resilience.

Evaluation on two open-source benchmark microservice systems indicates the effectiveness and efficiency.

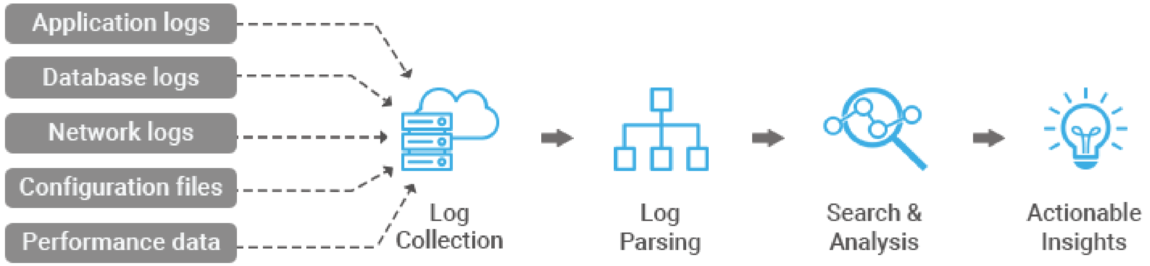
# CONTENTS

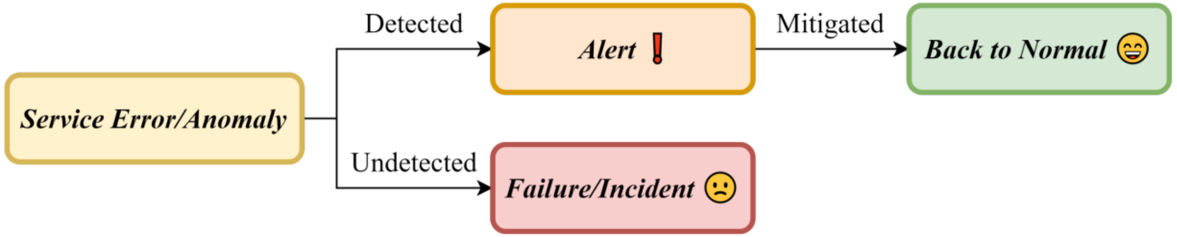
4

# Empirical Study on Alerting and Logging

# Why the Quality of Alerts and Logs Matters?

- Logs and alerts are important for reliability assurance.
- But the generation and processing of alerts are highly empirical.

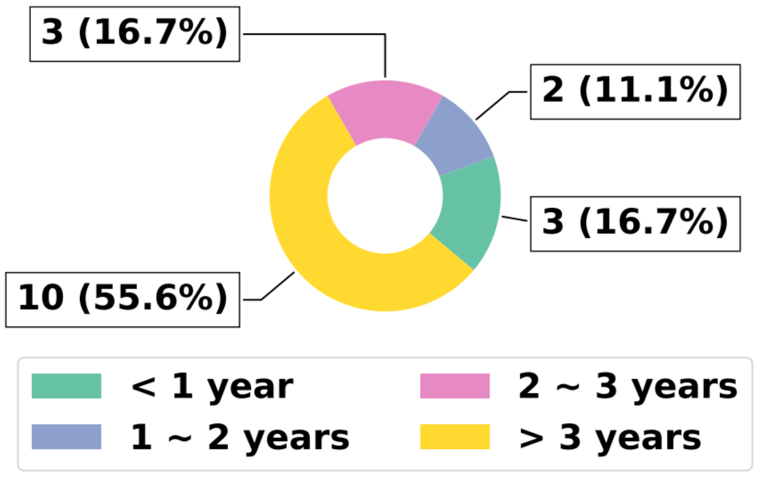


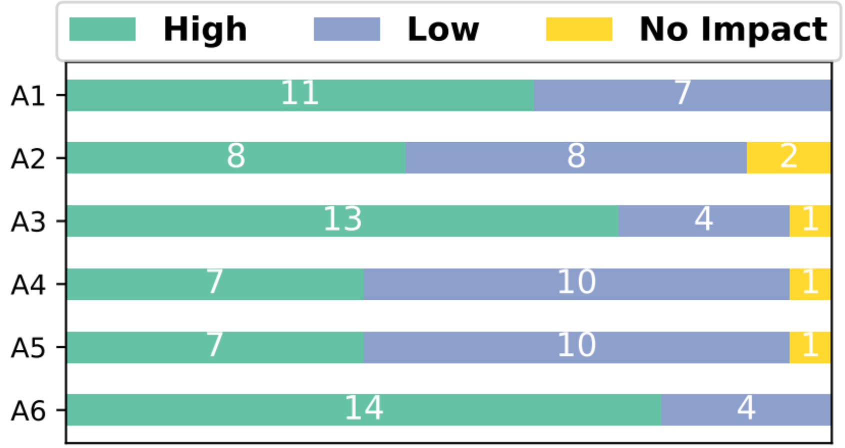

# Anti-patterns of Alerts in Cloud Systems

Quantitative inspection of 4 million alerts in 2 years + Interviews with 18 OCEs.

- Individual anti-patterns
  - [A1] Unclear Name or Description.
  - [A2] Misleading Severity.
  - [A3] Improper and Outdated Generation Rule.
  - [A4] Transient and Toggling Alerts.
- Collective anti-patterns
  - [A5] Repeating Alerts.
  - [A6] Cascading Alerts.




Working Experience as an OCE

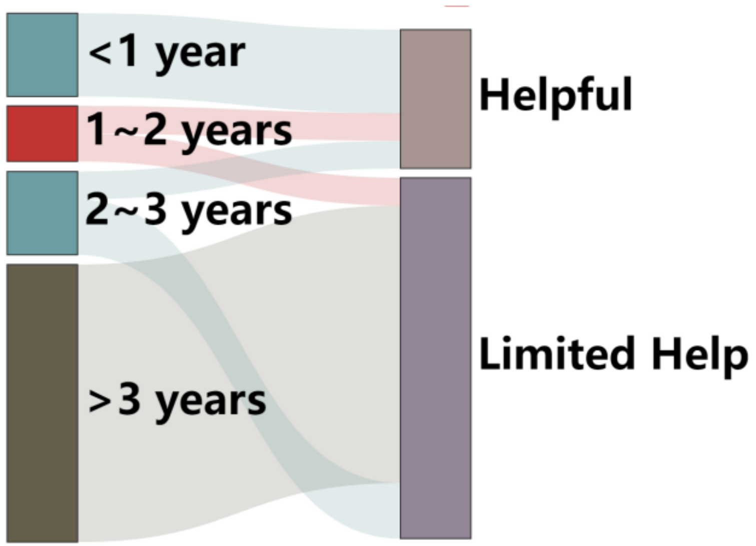



Impact of Anti-patterns

# Standard Alert Processing Procedure

| SOP for alert `nginx_cpu_usage_over_80` | |
|---|---|
| **Description** | CPU usage of nginx instance is higher than 80% |
| **Generation Rule** | Continuously check the CPU usage of nginx instance, generate the alert when usage is higher than 80%. |
| **Potential Impact** | Affects the forwarding of all requests. |
| **Possible Causes** | a) The workload is too high. b) ...... |
| **Steps to Diagnose** | Step 1: execute command `top -bn1` in the instance.<br>Step 2: ...... |

An example Standard Operation Procedure



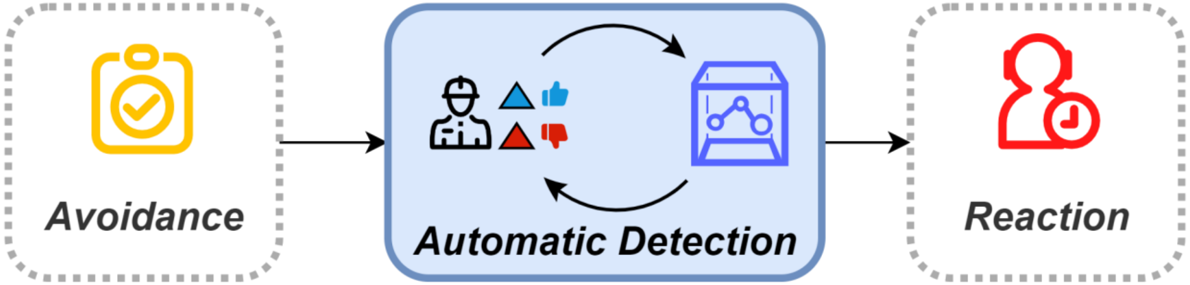

Answers to "Overall Helpfulness" regarding OCEs' working experience.

# Reactions to Anti-patterns

- Reactions
  - [R1] Alert Blocking.
  - [R2] Alert Aggregation.
  - [R3] Alert Correlation Analysis.
  - [R4] Emerging Alert Detection.


Effectiveness of Reactions

# Automatic Evaluation of the Quality of Alerts




- Criteria to measure the quality of alerts
    - Indicativeness
    - Precision
    - Handleability

- Incorporating human knowledge and machine learning to evaluate the three aspects of alerts

# Mechanism of Logging

```java
public void setTemperature(Integer temperature) {
  // ...
  logger.debug("Temperature set to {}. Old temperature was {}.", t, oldT);
  if (temperature.intValue() > 50) {
    logger.info("Temperature has risen above 50 degrees.");
  }
}
```

Verbosity Level

Static Text

Dynamic Content

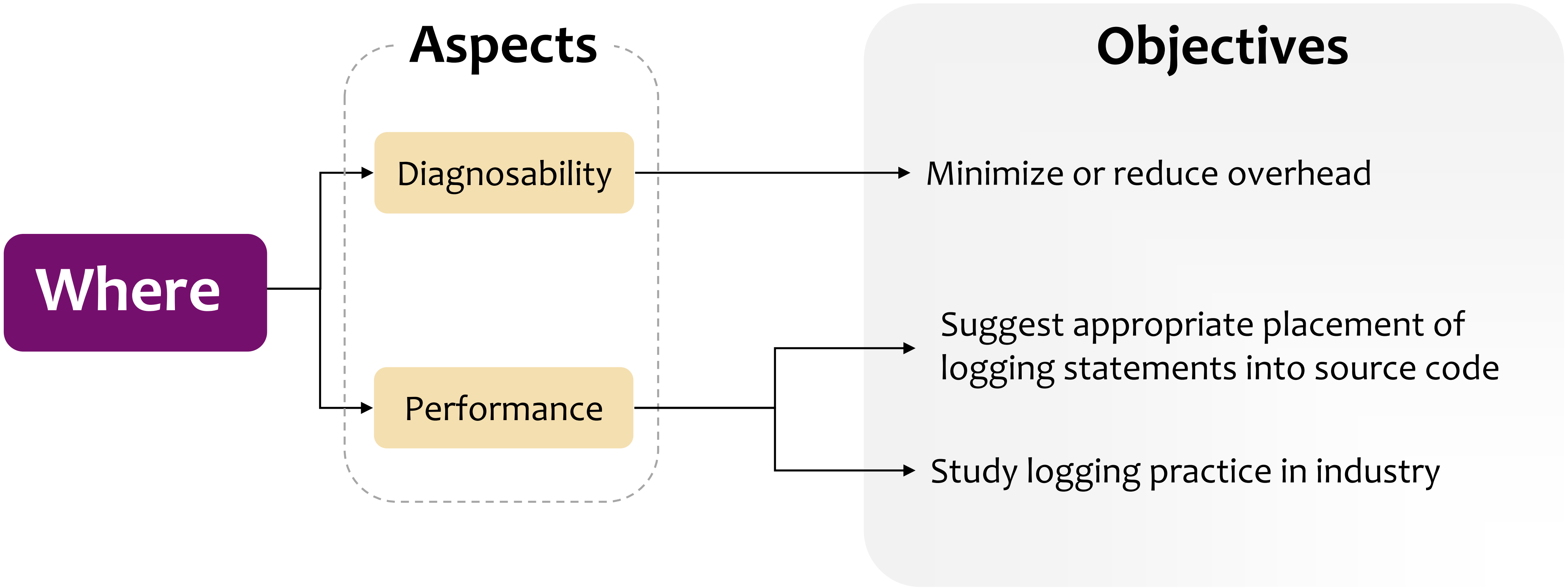
⇓

```
0 [setTemperature] DEBUG Wombat - Temperature set to 61. Old temperature was 42.
0 [setTemperature] INFO Wombat - Temperature has risen above 50 degrees.
```

# Challenges for Logging

| Challenges | Meaning | | Aspects |
|---|---|---|---|
| Where to log | Determining the appropriate location of logging statements. | Each challenge exhibits one or more aspects. → | Diagnosability |
| What to log | Providing sufficient and concise ***verbosity level***, ***static text***, and ***dynamic content***. | | Maintenance |
| How to log | The systematical design pattern and maintenance of logging statements. | | Performance |

# Where to log


Aspects
Objectives
Where
Diagnosability
Performance
Minimize or reduce overhead
Suggest appropriate placement of logging statements into source code
Study logging practice in industry

# What to log

**What**

## Aspects

## Objectives

- **Diagnosability**
  - Enhance existing logging code to aid debugging
  - Suggest proper variables and text description in log
- **Maintenance**
  - Determine whether a logging statement is likely to change in the future
  - Study logging practice in industry
- **Performance**
  - Study the performance overhead and energy impact of logging in mobile app
  - Automatically change the log level of a system in case of anomaly

# How to log

## Aspects

## Objectives

**How**

- Diagnosability
  - Characterize the anti-patterns in the logging code
  - Optimize the implementation of logging mechanism to facilitate failure diagnosis
- Maintenance
  - Determine whether a logging statement is likely to change in the future
  - Characterize and detect duplicate logging code
- Performance
  - Characterize and detect the anti-patterns in the logging code
  - Characterize and prioritize the maintenance of logging statements
  - Study the relationship between logging characteristics and the code quality
  - Propose new abstraction or programming paradigm of logging

# Improving the Quality of Logs

- Prospective Directions
  - Analysis-Oriented Logging
  - Automated Generation of Logging Statements
- Best Practices for Logging
  - Always follow the logging standards
  - Keep proper quantity of log messages

# Summary of Chapter 5

First empirical study on characterizing and mitigating anti-patterns of alerts in an industrial cloud system.

Identify four individual anti-patterns, two collective anti-patterns, and four postmortem reactions.

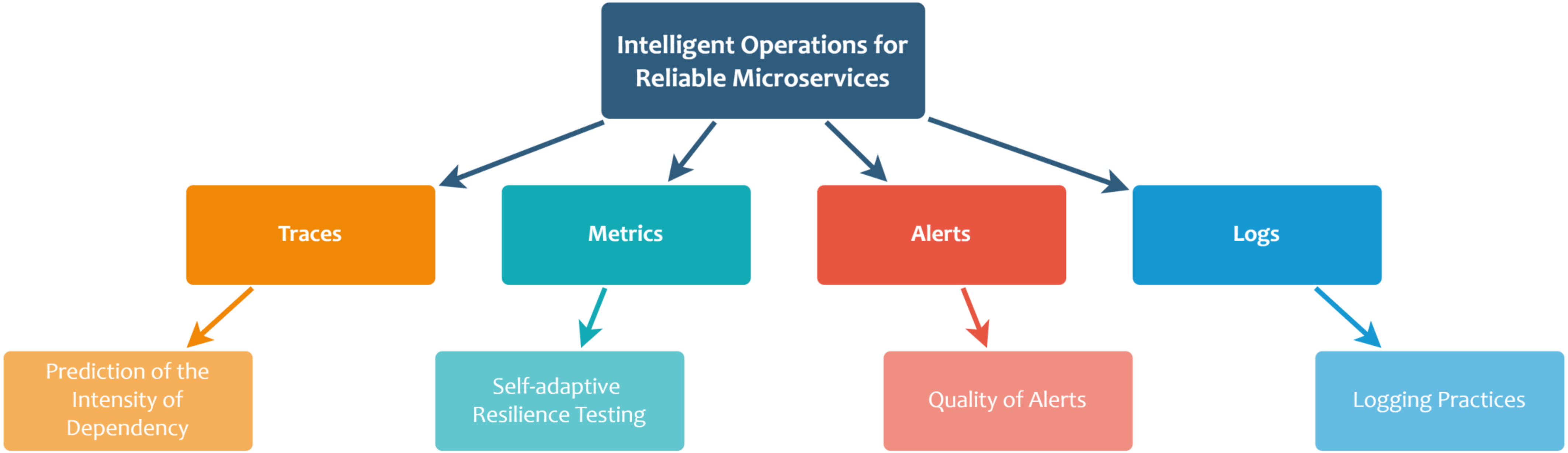
Propose directions on improving the quality of alerts and logs.

# CONTENTS

5

# Conclusion and Future Work

# Conclusion




- The first empirical study on the intensity of dependency.
- The first method to quantify the intensity of microservice dependencies.
- Release an industrial dataset for reuse.

*[ASE'21]*

- The first empirical study on the failures of resilient and unresilient microservices.
- The first self-adaptive resilience testing framework.

*[ICSE'23]**

- Identify six antipatterns of alerts in a production cloud.
- Identify four postmortem reactions to antipatterns.
- Survey the current practice of logging for reliability.
- Propose directions on improving the quality of alerts and logs.

*[DSN'22, WWW'21]*

*[CSUR'21]*

** Under review by ICSE'23*

# Future Work

**Multiple data type**

**Single data type**

Traces

Metrics

Alerts

Logs

Fusing Multiple Data Sources for Intelligent Operations

Trace Compression based on Service Topology

Human-in-the-loop Auto QoA Evaluation

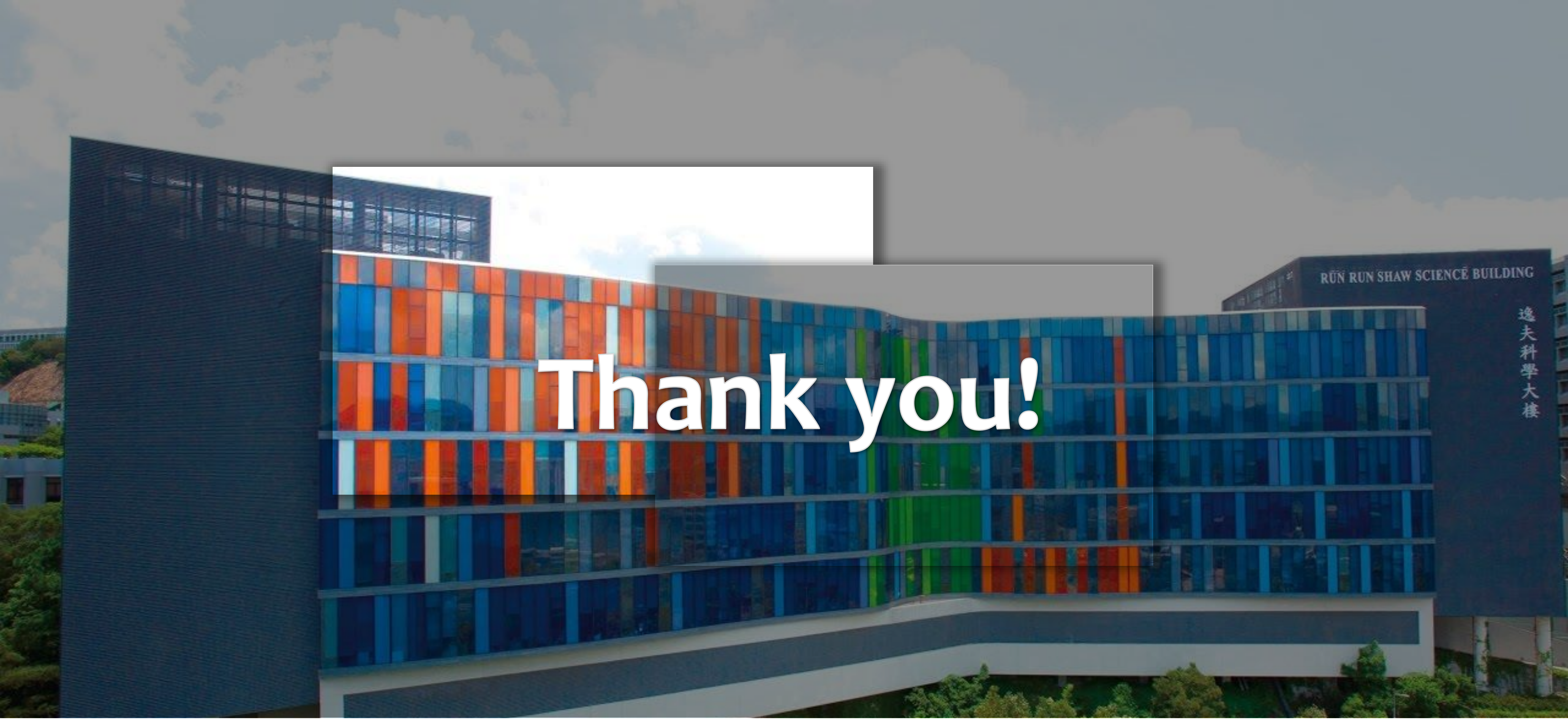
Automated Generation of Logging Statements

Analysis-Oriented Logging

# Publications

- **Tianyi Yang**, Jiacheng Shen, Yuxin Su, Xiaoxue Ren, Xiao Ling, Yongqiang Yang, and Michael R. Lyu. 2021. Characterizing and Mitigating Anti-patterns of Alerts in Industrial Cloud Systems. In Proceedings of the 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'22) June 27-30, 2022, Baltimore, Maryland, USA. IEEE, 2022, pp. 393-401.
- **Tianyi Yang**, Jiacheng Shen, Yuxin Su, Xiao Ling, Yongqiang Yang, and Michael R. Lyu. 2021. AID: Efficient Prediction of Aggregated Intensity of Dependency in Large-scale Cloud Systems. In Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE'21) November 15-19, 2021, Australia. IEEE/ACM, 2021, pp. 653-665.
- **Tianyi Yang**, Cuiyun Gao, Jingya Zang, David Lo, and Michael R. Lyu. 2021. TOUR: Dynamic Topic and Sentiment Analysis of User Reviews for Assisting App Release. In Companion Proceedings of the Web Conference 2021 (WWW'21), April 19–23, 2021, Ljubljana, Slovenia. ACM, 2021, pp. 708–712.
- Jiacheng Shen, **Tianyi Yang**, Yuxin Su, Yangfan Zhou, and Michael R. Lyu. 2021. Defuse: A Dependency-Guided Function Scheduler to Mitigate Cold Starts on FaaS Platforms. In Proceedings of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS'21) July 7-10, 2021, Washington DC, USA. IEEE, 2021, pp. 194-204.
- Shilin He, Pinjia He, Zhuangbin Chen, **Tianyi Yang**, Yuxin Su, and Michael R. Lyu. 2021. A Survey on Automated Log Analysis for Reliability Engineering. ACM Computing Survey (CSUR), April, 2021. ACM, New York, NY, USA. ACM, 2021, pp. 1-37.
- (Under review, 1st author) AVERT: A Self-adaptive Resilience Testing Framework for Microservice Systems
- (Under review, 1st author) Managing Service Dependency for Cloud Reliability: The Industrial Practice
- (Under review, 2nd author) Eadro: Integrating Anomaly Detection and Root Cause Localization on Multi-source Monitoring Data for Microservice
- (Under review, 2nd author) HADES: Heterogeneous Anomaly Detection for Software Systems via Attentive Multi-modal Learning
- (Under review, 4th author) ScaleStore: Scalable and Fault Tolerant Key-Value Store on Disaggregated Memor

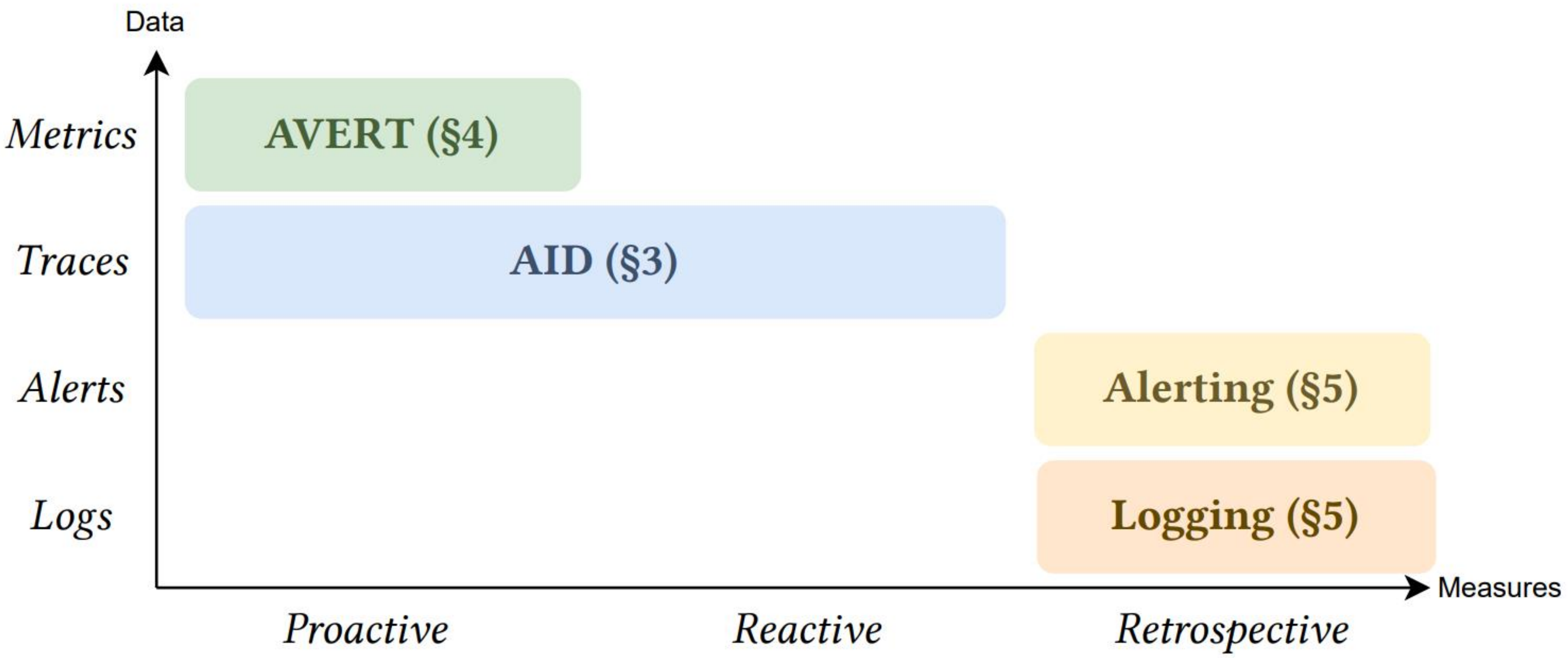
# Thank you!




ARISE
Automated Reliable Intelligent Software Engineering

香港中文大學
The Chinese University of Hong Kong

# Categorization of the Research Thesis



# Questions

- Why data-driven?
  - Adaptivity: Data-driven approach can adapt to various types of online services with different programming languages.
  - Practicality: Non-intrusive, like a plug-in module for online services.

- Why to use such types of monitoring data?
  - Such data types are universal in microservice architectures.

# Evaluation Metrics

$$CE = \frac{1}{N} \sum_{i=1}^{N} -[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

$$MAE = \frac{\sum_{i=1}^{N} |y_i - p_i|}{n}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (y_i - p_i)^2}{N}}$$

The smaller, the better.