RELIABILITY IN MODERN CLOUD SYSTEMS

Summer 2025

LOGISTICS

ASSIGNMENT 1

- Assignment 1 grades posted
- assn1_grades branch in the private forks

ASSIGNMENT 2

- Assignment 2 has now been released
- Due Date: May 28th, 5:00 pm CEST
- Description
  - Writing and executing a workload
  - Using distributed tracing to monitor and analyze the system
  - Calculating latency percentiles

OBSERVABILITY DISCUSSION

PAPER SUMMARY

PAPER SUMMARY

- Dapper is (maybe was) the distributed tracing system deployed at Google in production
- First known deployment of distributed tracing in industry
- Tracing Overhead is around ~200ns for span creation
- Annotations provide more context to traces
- Sampling is important
  - First known use of sampling ☺
  - Adaptive sampling to capture a fixed rate of traces per unit time

DISCUSSION THEMES

- If you had a finite operational budget for observability, how would you distribute it among metrics, logs, or traces?
- What is the correct sampling rate for an application? Should this rate be fixed?
- What is the biggest problem with using raw logs?
- Where should you add tracing/instrumentation points in your system?

DISCUSSION THEMES

- What is the biggest problem with using raw logs?

DISCUSSION THEMES

- What is the biggest problem with using raw logs?

Raw logs are unstructured and full of detail. Extracting and processing logs to get actionable data is difficult and annoying to automate.

DISCUSSION THEMES

- If you had a finite operational budget for observability, how would you distribute it among metrics, logs, or traces?

DISCUSSION THEMES

- If you had a finite operational budget for observability, how would you distribute it among metrics, logs, or traces?

You need all 3! Metrics are good for monitoring, logs are good for human-annotated details, traces are good for connecting execution.

DISCUSSION THEMES

- What is the correct sampling rate for an application? Should this rate be fixed?

DISCUSSION THEMES

- What is the correct sampling rate for an application? Should this rate be fixed?

No fixed rate for an application. Each service should decide its own sampling rate so that each service can decide the perf-tracing tradeoff.

DISCUSSION THEMES

- Where should you add tracing/instrumentation points in your system?

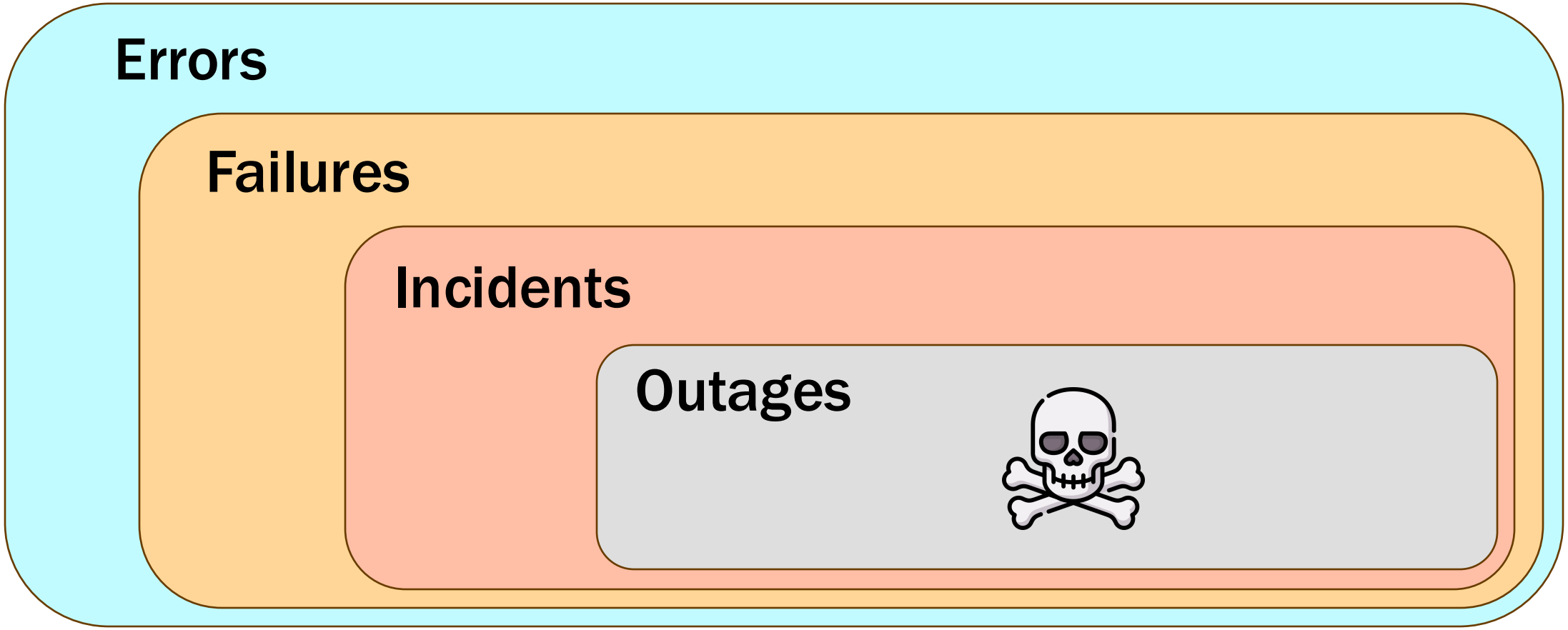
DISCUSSION THEMES

- Where should you add tracing/instrumentation points in your system?

Instrumentation Points should be added in all places where the request goes. Incomplete tracing only leaves gaps that might be important.

ERRORS, FAILURES, INCIDENTS, OUTAGES

THE ERROR FAMILY HIERARCHY



ERRORS

An error is simply a deviation from successful behavior

TYPES OF ERRORS (EXPECTATION)

Unexpected Errors

- Could be due to bugs in the system
- Eg: Divide by a zero error or null pointer exception
- Could be due to environmental conditions
- Eg: Timeout, connectivity

Expected Errors

- Handle incorrect arguments
- Eg: User with a specific username doesn’t exist
- Handle situations where there is no data
- Eg: Generating an error when there are no reviews for a specific item

TYPES OF ERRORS (PROPAGATION)

Non-Fatal Errors

- Errors that are internal and do not propagate back to the end-user
- These errors do not cause the request to fail

Fatal Errors

- Errors that propagate back to the end-user
- These errors cause the request to fail

NON-FATAL ERRORS

```
1 func GetUserInfo(user User) (Info, error) {
2     info, err := CheckInAmericasDB(user)
3     if  err == nil {
4         return info, nil
5     }
6     info, err = CheckInEuropeDB(user)
7     if err == nil {
8         return info, nil
9     }
10    info, err = CheckInAsiaPacDB(user)
11    if err == nil {
12        return info, nil
13    }
14    return nil, NOT_FOUND
15 }
```

CheckInAmericasDB and CheckInEuropeDB could both return an error

Error is only returned if all previous calls return an error

Fig.2 Listing (1), The Tale of Errors in Microservices, Dec. 2024 by Lee et al

FAILURES

- Failure is an observable problem in the system
  - Manifestation of an error that causes systems to behave in an unintended fashion
  - Failures could be caused by a single error or a combination of multiple errors
  - Not necessary that failure has to be caused by a fatal error!

TYPES OF FAILURES

Transient Failures

- Temporary
- Short-lived
- Do not require any fixes
- Not caused due to a design flaw in the system

Systemic Failures

- Persistent
- Remain until fixed
- Requires patching or bug fixes
- Caused due to logic bugs or design flaws

Definitions from “Try again: The tools and techniques behind resilient systems” by Marc Brooker

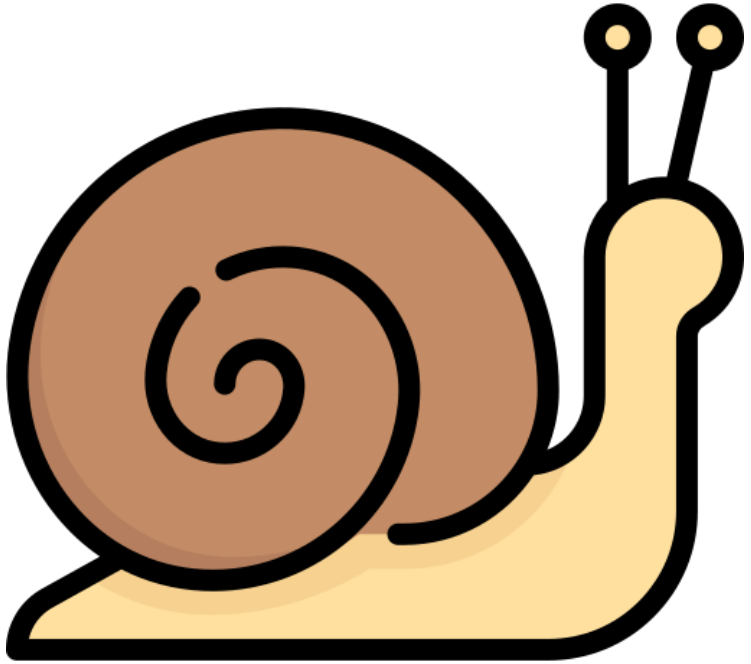
FAIL-STOP FAILURES

- Full crash of a component
- Potentially impacts the availability of the system
- Worst case scenario for a component as this requires full recovery

FAIL-SLOW FAILURES

- System is still functioning but with lower than expected system performance

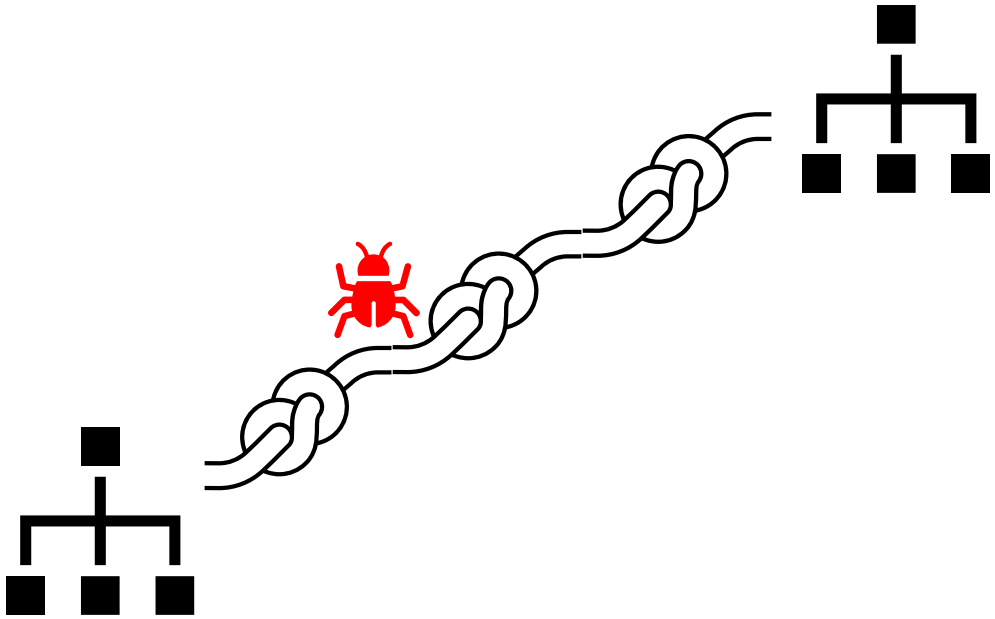
- Leads to “limplock” situations where the rate of progress in a system is very slow



One-Size-Fits-None: Understanding and Enhancing Slow-Fault Tolerance in Modern Distributed Systems, Apr’ 25, Liu et al

CROSS-SYSTEM INTERACTION FAILURES

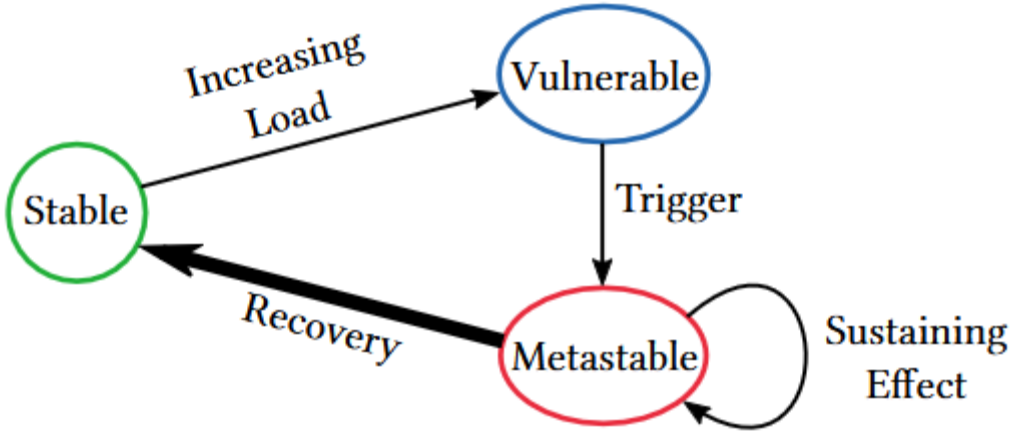
- Failures caused due to interaction between multiple systems
- Typical cause is due to upstream and downstream services interpreting things differently
- Root cause is not limited to any one system



METASTABILITY FAILURES

- System gets in a persistent "metastable" state where there is no more progress

- Caused due to a trigger that either artificially increases the load or reduces the processing capacity



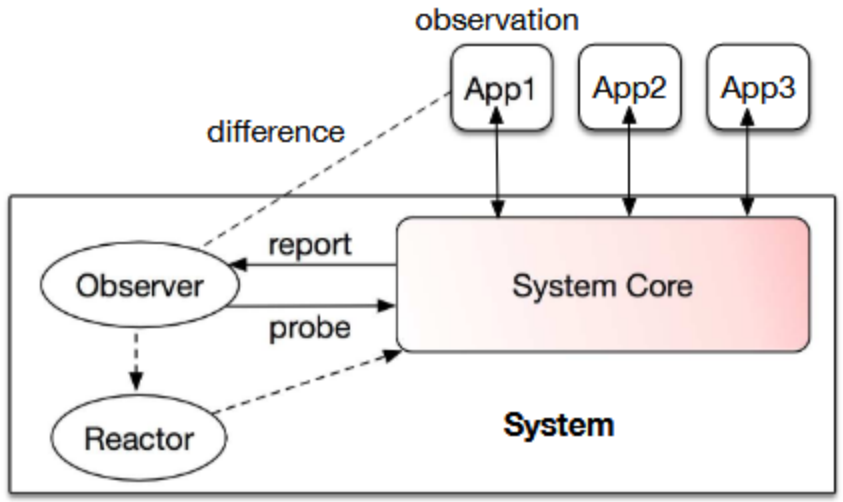
GRAY FAILURES

- Partial failure that does not cause a full outage
  - "A component appears to be working but is broken"

GRAY FAILURES

- Partial failure that does not cause a full outage
  - "A component appears to be working but is broken"

- **Differential Observability**: Not detected by all components causing different components to behave differently



INCIDENTS

Incidents are unplanned interruptions in a service or a perceived reduction in the quality of the service

INCIDENTS

Incidents are unplanned interruptions in a service or a perceived reduction in the quality of the service

Troubleshooting stages:

- Detection: How was an incident detected? If not, then why was not an incident detected?
- Root Cause Analysis: What was the root cause of the incident?
- Mitigation: How to resolve this incident? Can this be automated?

INCIDENTS – KEY NUMBERS

TTD: Time to Detect: Total time taken to detect an incident from when the incident happened

TTM: Time to Mitigate: Total time taken to mitigate the incident from the time it was detected

Common target is to resolve incidents within a specific time window

OUTAGES

- The most severe incidents can lead to an outage
- Outage is a complete loss in availability of the system
- This is the worst-case scenario for an application
  - Outages impact customers/users directly
  - Cause significant monetary loss

DISCUSSION THEMES

- What type of errors cause outages?
- What types of incidents do not lead to outages?
- Can you detect failures with 100% accuracy?