

Demystifying the Fight Against Complexity:  
A Comprehensive Study of ***Live Debugging  
Activities*** in Production Cloud Systems

**P. C. Sruthi**, Zinan Guo, Deming Chu, Zhengyan Chen, Yongle Zhang



# Most Internet services live on the cloud



# Failures in cloud systems are catastrophic

**Google cloud is down, affecting  
Amazon AWS S3 outage is breaking**

**Apple's iCloud recovers after a  
four-hour outage**

zdnet.com

September 17, 2018

**Microsoft provides preliminary  
report on its September 4 cloud  
outage**

Ken Yeung / The Next Web:

**Dropbox says its service is back up and running  
outage (Updated)** — Update 2 - 12 January: Dropbox now

running" for all users. — Cloud storage service Dropbox is experier

**Jan 10, 2014, 10:08 PM** — In context

Engineering teams are continuing to  
investigate the incident and are saying they  
will provide a more detailed analysis "in the  
weeks ahead."

**Cloudflare blames 'bad software'  
deployment for today's outage**

**Office 365, Azure users are locked**

**VMware Joins Cloud Outage Party With Cloud  
Foundry Blackout**

**Yahoo Mail Takes Big Hit In Cloud Outage**

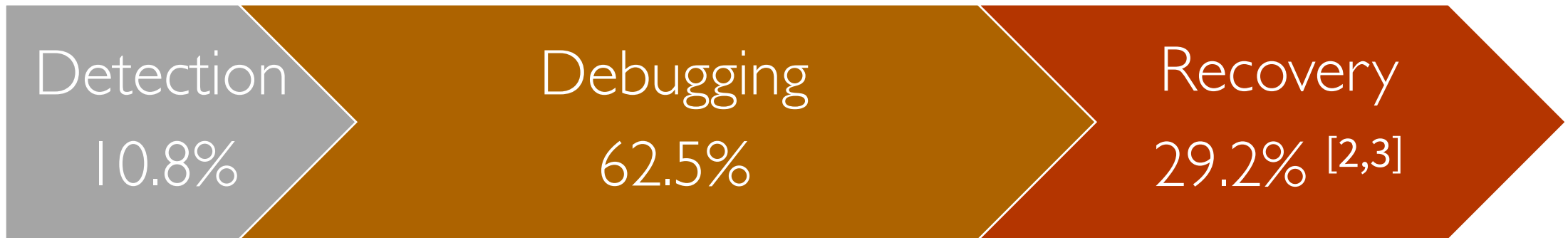
**Another Cloud Outage Strikes Microsoft BPOS,  
Exchange Online**

**Google Docs Goes Dark In Evening Cloud  
Outage**

Google Docs, Google's cloud-based suite of  
productivity applications, suffered a brief  
outage with some Google Docs cloud services  
going down for roughly an hour. Google Docs  
comprises a host of Google's cloud-based  
applications, including documents,  
presentations, spreadsheets and other tools.

# Debugging in the cloud is costly

- Randomly sampled 20 cloud failures from Google Cloud Incidents [1]
  - Average failure resolution duration: 3.88 hours



*Cloud Failure Resolution Duration*

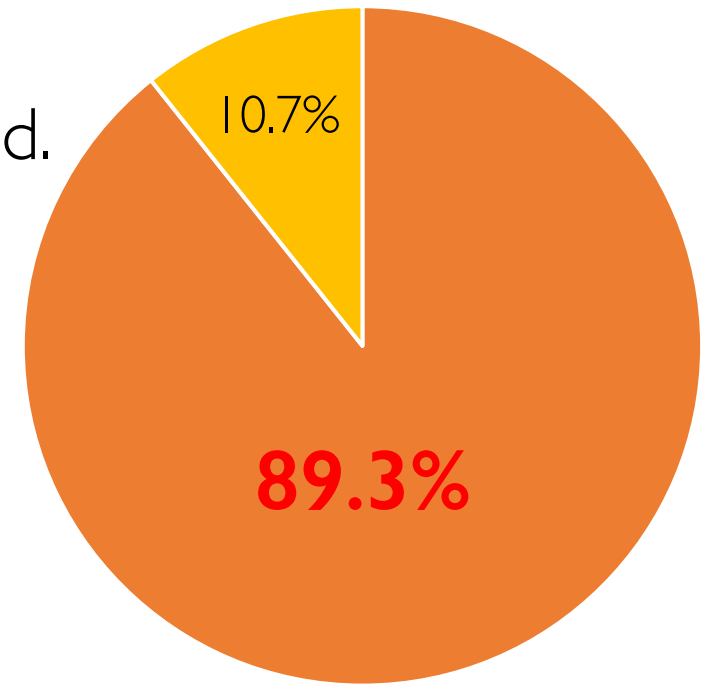
[1]: <https://status.cloud.google.com/summary>

[2]: **Normalized Average Percentage:** Calculated by normalizing the time spent on each activity by the total time for each case before averaging..

[3]: Total percentages (10.8% + 62.3% + 29.2%) > 100% due to (1) rounding up the normalized percentage in each case and (2) time attributed to multiple activities because of vague description in reports.

# Understanding of cloud debugging is limited

- Most existing debugging studies focus on offline debugging of failed tests.
  - 89.3% of cloud failures we studied are debugged completely *Online debugging rate in cloud debugging*
- Recent studies on cloud debugging are coarse-grained.
  - Ghosh et al. [SoCC'22]:
    - Categories of root causes  $\leftrightarrow$  mitigation & detection strategies.
  - Dogga et al. [ATC'23]:
    - Root cause labeling  $\leftrightarrow$  bug report.

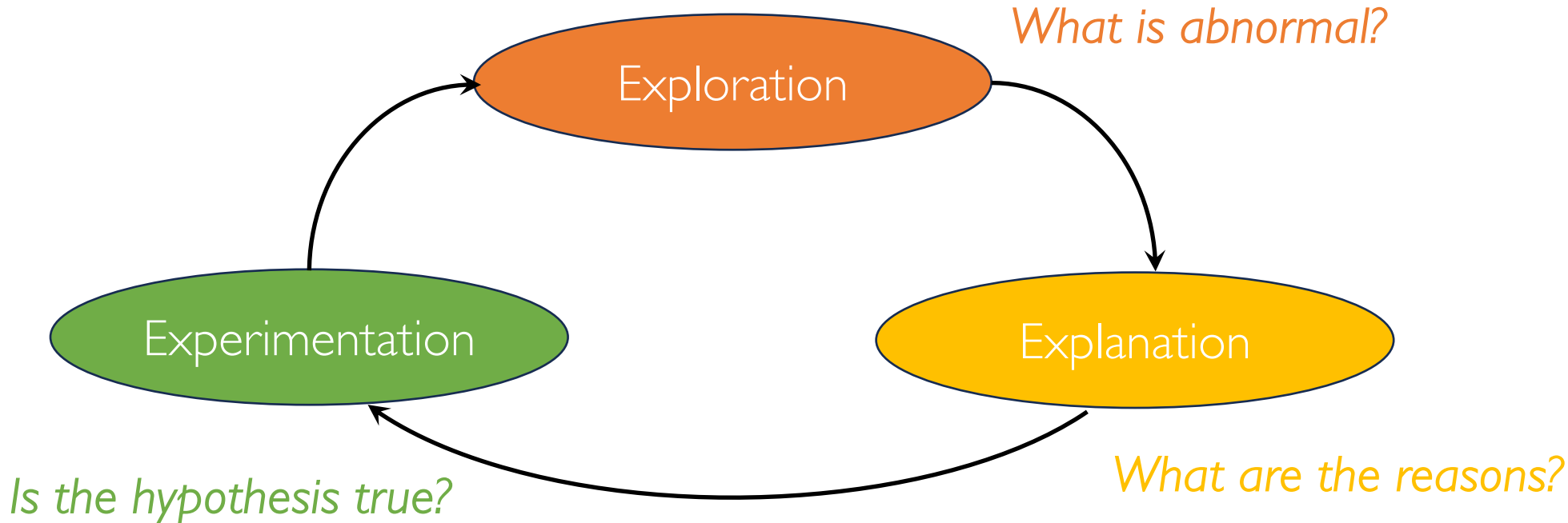
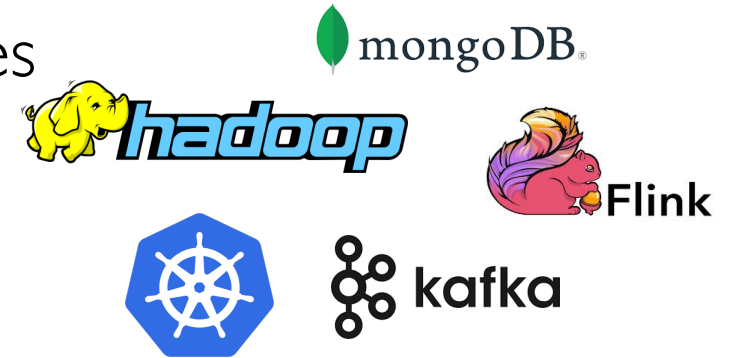


**There is no study on how debugging in production cloud (live debugging) is performed step by step!**

■ Debugged in production ■ Debugged offline

# A fine-grained study of cloud live debugging

- 93 documented production cloud debugging experiences
  - from 14 widely-deployed open-source distributed systems
  - with **step-by-step** hypothesis formulation & verification
  - 6 ~ 48 steps (avg. 19) / case



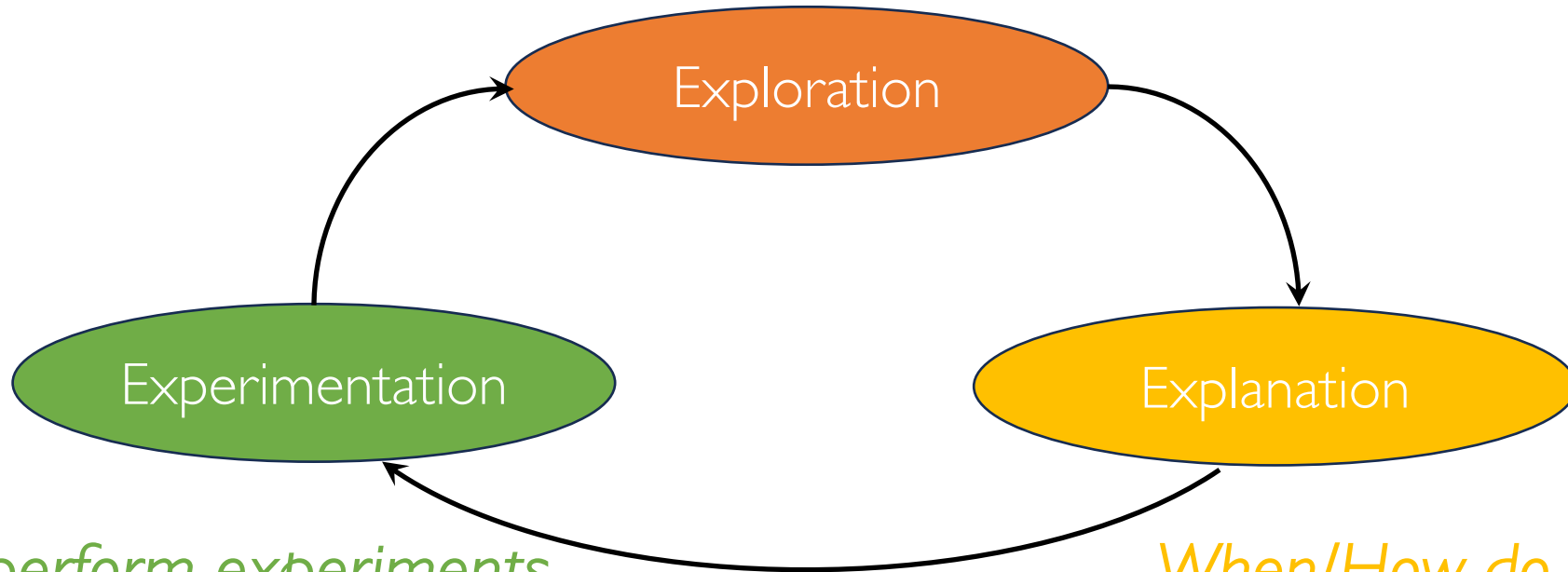
# Contributions

- A taxonomy of live debugging activities
  - Debugging strategies adopted
  - Challenges faced in each activity
- Novel debugging techniques
  - for cloud-specific/amplified challenges

Activity	Mechanism	
Explanation (640)	Model Analysis (640)	State Transition
		Quantity Contribution
Exploration (452)	Correlation (303)	Locality
		Execution Comparison
	Anomaly Detection (149)	Event Anomaly
		State Anomaly
Experimentation (438)	Information Collection (209)	Source Code Anomaly
		Instrumentation
		Probing
		Online Intervention (196)
		Offline Reproduction (33)

# Outline: selected research questions

*What are the challenges for observability?*

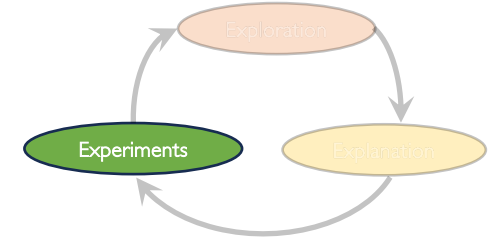


*How to perform experiments and verify hypotheses?*

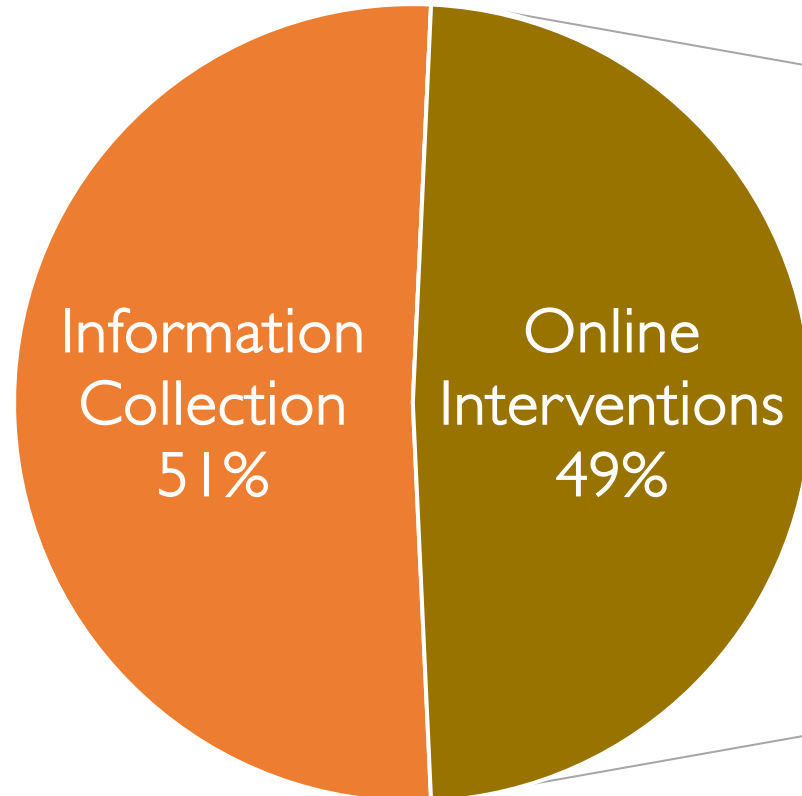
*When/How do developers formulate multiple hypotheses?*



# How to perform experiments & verify hypotheses?



The Tie btw. Passive & Proactive Experiments



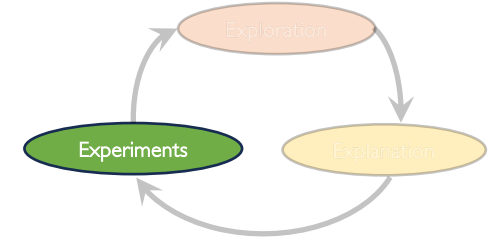
Diagnosis  
25%

Verification  
24%

Intervention	%
Config change	41%
Restart	30%
Command	14%
Upgrade	9%
Code change	6%

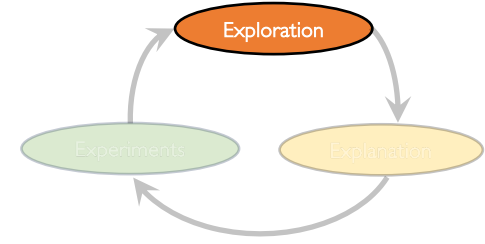
Proactive experiments which **alter execution in production** are performed as often as passive experiments which **collect information and make observations**!

# Developers want more intervention mechanisms



- Extra intervention mechanisms (knobs) developer requested:
  - Timeout thresholds
  - Data sizes
  - Queue sizes
- Extra intervention mechanisms (knobs) developer implemented:
  - Synchronization
    - Remove/add lock
    - Change lock type
  - Data flow
    - Skipping items in collections satisfying some constraint

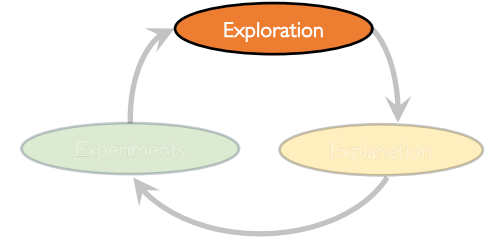
# What are the challenges for observability?



- **What** to observe – system-specific or system-agnostic data?

System-agnostic data	System-specific data
CPU Utilization	Zookeeper ZNodes
Traffic Volume	Spark Executors
Memory	HDFS Read Offsets
Latency	systemd cookie values
...	...

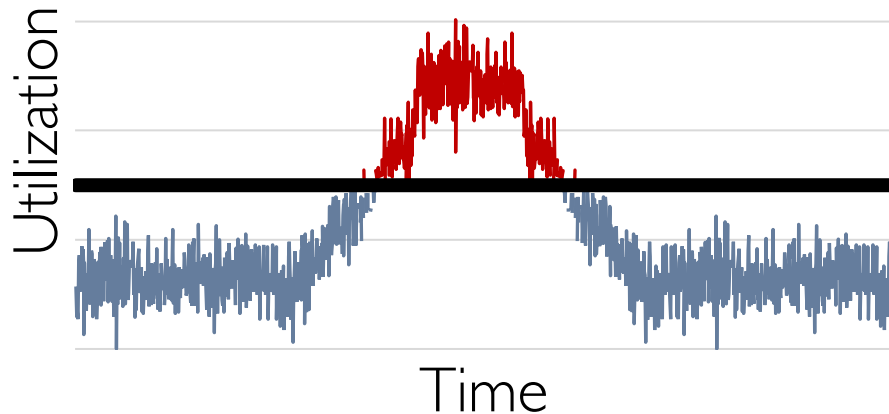
# What are the challenges for observability?



- **What** to observe – system-specific or system-agnostic data?
- **How** to alert – are default alerting rules (invariants) enough?

## Conventional Invariants

*E.g., threshold violation*



## Unconventional Invariants

Numerical Delta

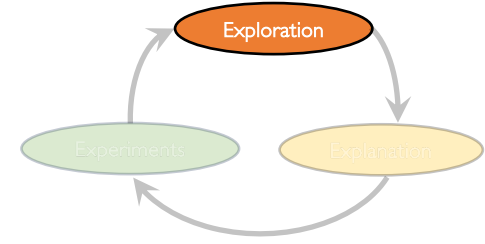
Data Consistency

Temporal Distance between events

Process State Relation

var_1	var_2	var_1 - var_2
57	10	47
63	16	47
9	-38	47

# What are the challenges for observability?

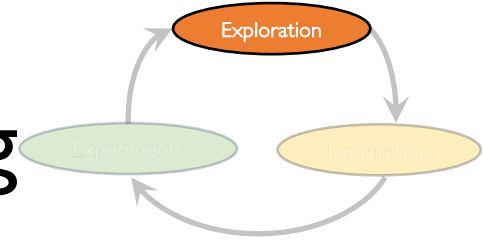


- **What** to observe – system-specific or system-agnostic data?
- **How** to alert – are default alerting rules (invariants) enough?

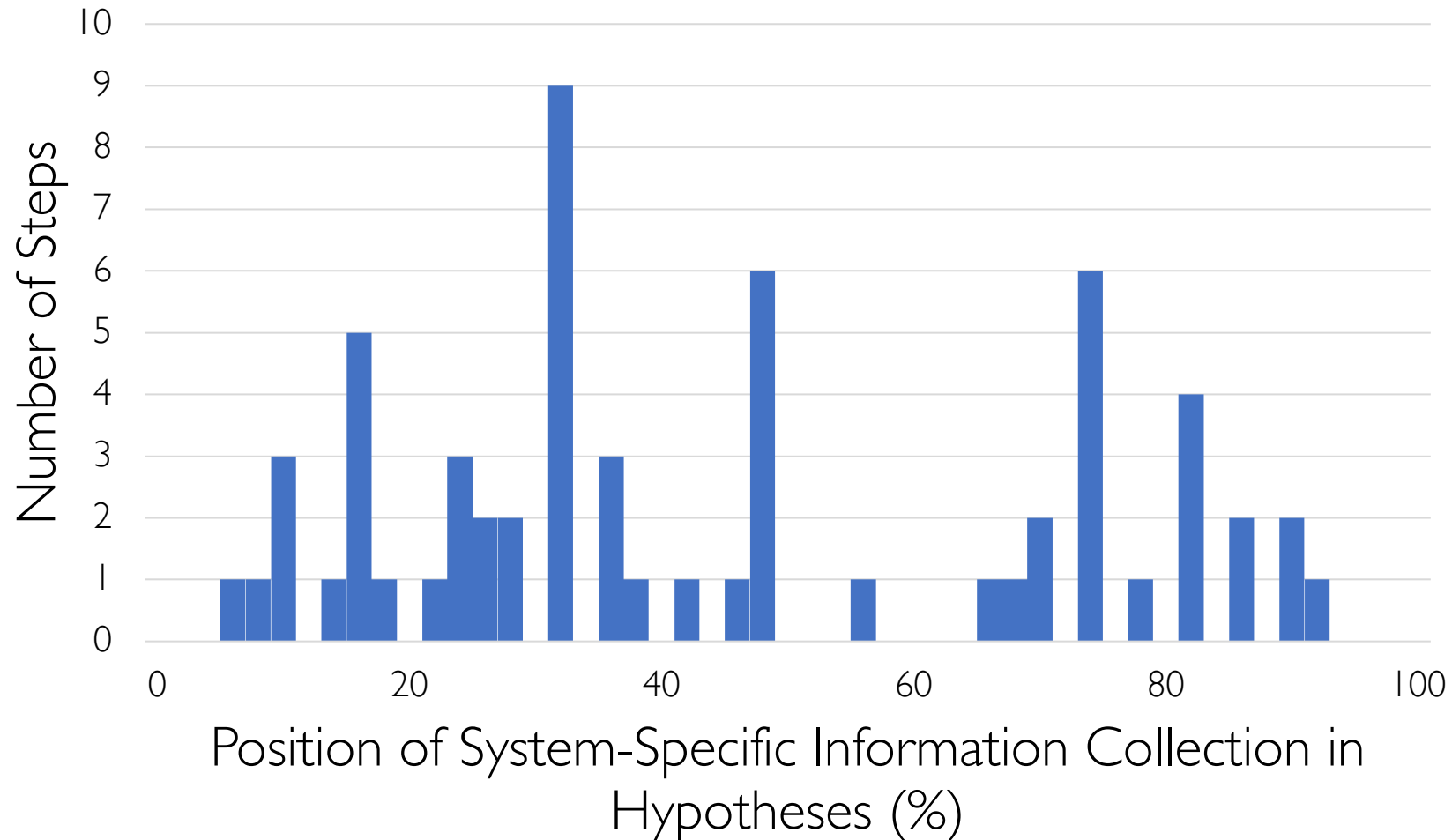
	System-agnostic data	System-specific data
Default invariants	<b>79.5%</b>	16.5%
Unconventional invariants	3.4%	0.6%

Most anomalies used in debugging can be captured by system-agnostic data and default alerting rules.

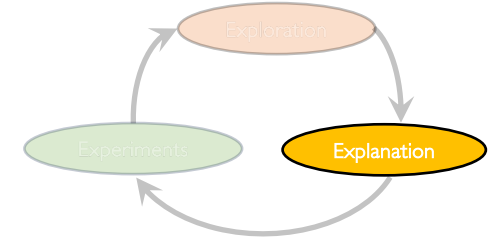
# System-specific data usage spreads across debugging



- **When** is system-specific data used?



# When & how to formulate multiple hypotheses?



- Avoid random guesses: experts debug by enumerating immediate causes w.r.t. a model & suspecting the model's correctness.
  - 80.4% of debugging steps formulating multiple hypotheses are performed this way.

Model	Causal link
<b>Component dependency model</b>	Component dependency (static / dynamic)
Fault model	Control flow involving error (partial node failure, omission failure) or recovery
<b>Data flow model</b>	Data flow
Concurrency model	Interleaving (thread / network message)
<b>Delay contribution model</b>	Contributors, threshold, channel endpoints
Quantity contribution model	Contributors, threshold

# Cloud-specific/amplified challenges & debugging tech.

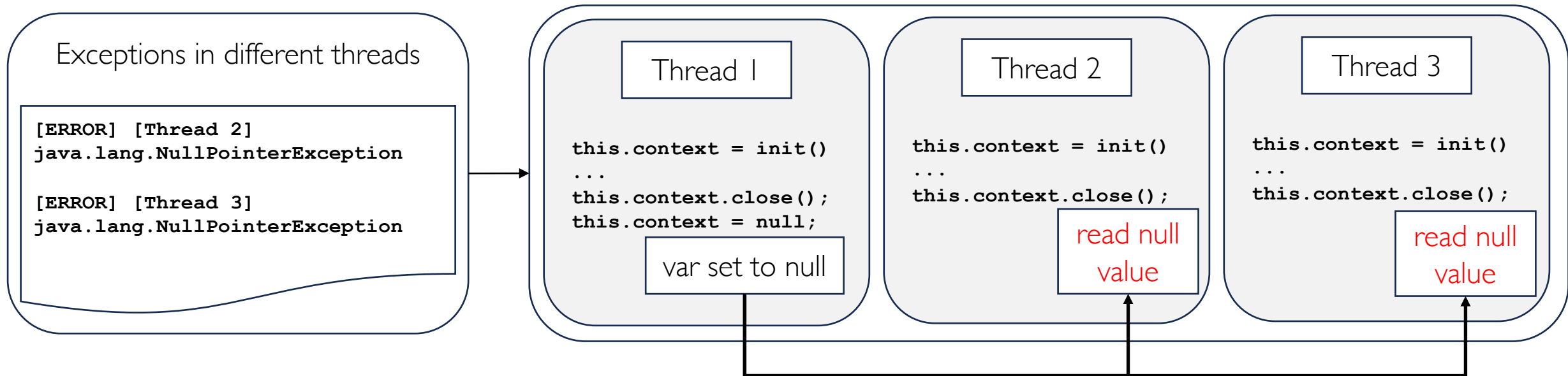
- **Concurrency bugs**
- Partial failures
- Cascading failures
- Slow failures
- Dynamism in microservices



# Challenging debugging tasks are done lazily & scoped

- **Debugging concurrency bugs:**

- Lazy: without concurrency-related anomaly, avoid reasoning about concurrency.
- Scoped: concurrency-related anomaly helps scope the interleaving to be considered.



# Conclusion



**Thank you!**

<https://github.com/zlab-purdue/socc-24-debugging-study>

- A fine-grained study of live debugging experiences in production cloud.
  - Taxonomies of debugging activities, techniques, and challenges.
- Highlighted findings
  - Experiments: Interventions are performed frequently for debugging purposes.
  - Observability: The usage of system-specific data spreads across debugging.
  - Explanation: Experts debug systematically by following immediate causes in models.

*More findings in the paper!*

