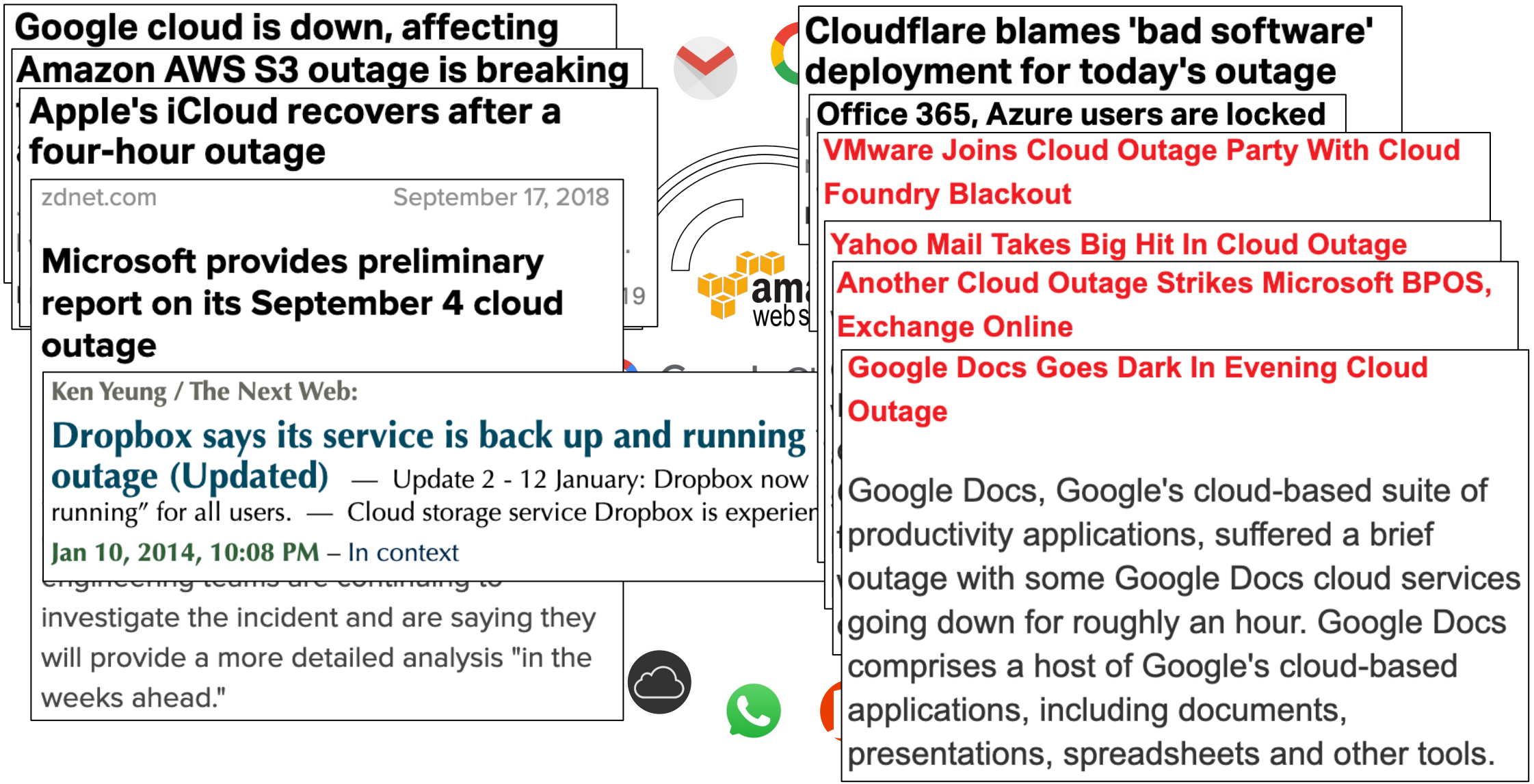
Demystifying the Fight Against Complexity: A Comprehensive Study of ***Live Debugging Activities*** in Production Cloud Systems

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

PURDUE
UNIVERSITY

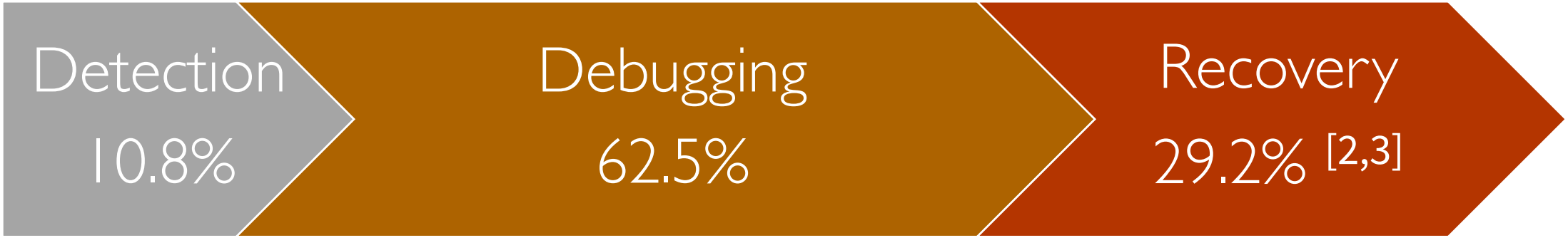
Most Internet services live on the cloud

Failures in cloud systems are catastrophic



Debugging in the cloud is costly

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



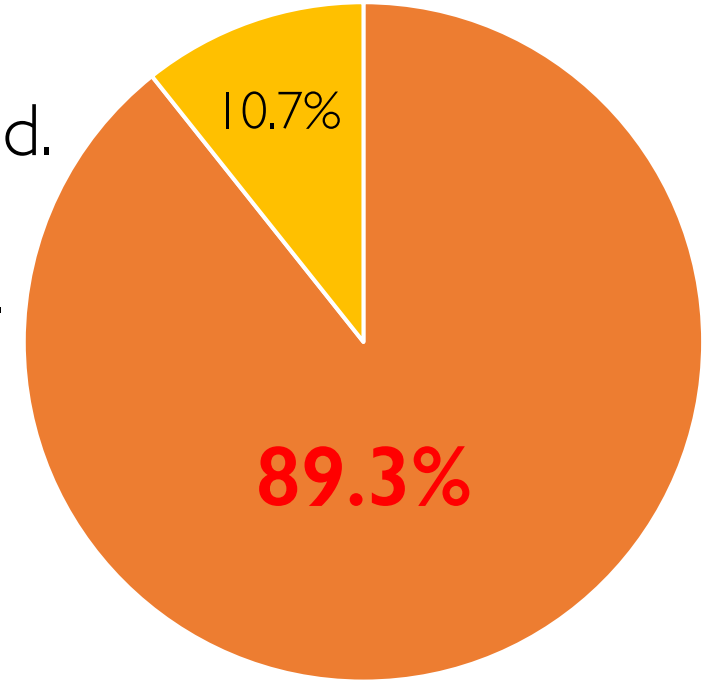
Cloud Failure Resolution Duration

[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 in production.

Online debugging rate in cloud debugging

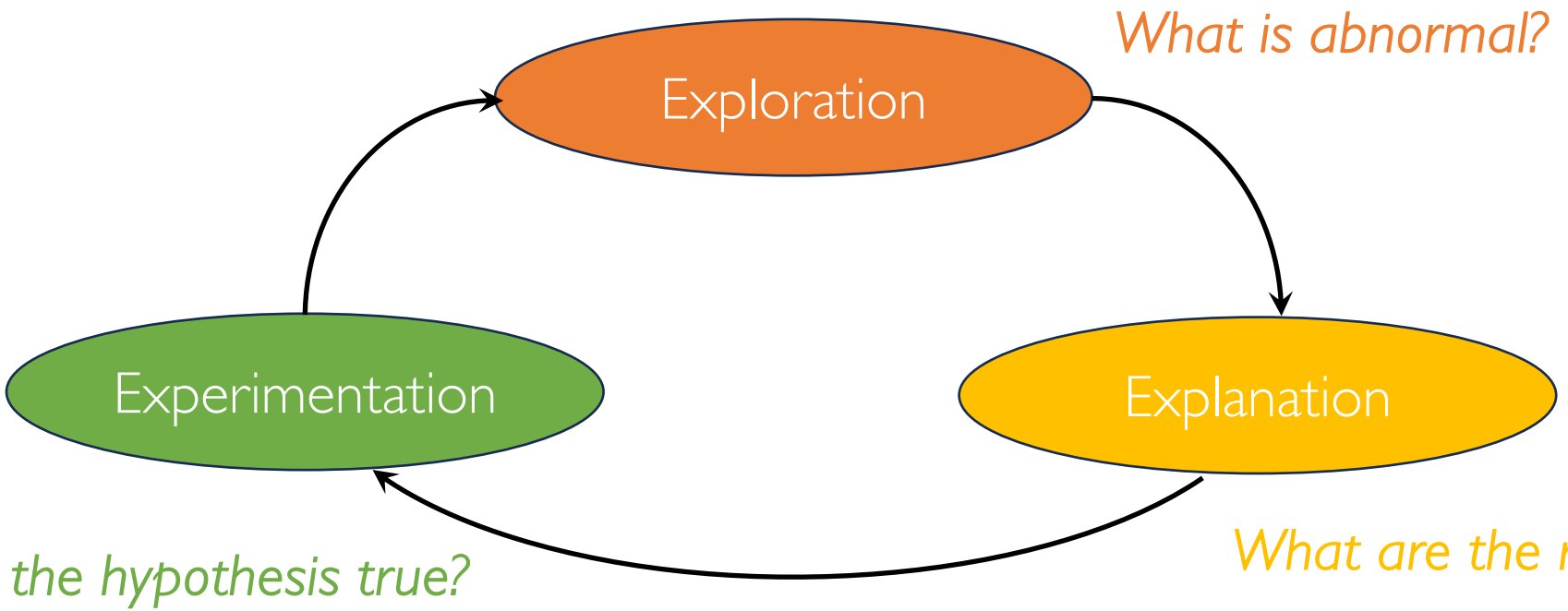


- Recent studies on cloud debugging are coarse-grained.
  - Ghosh et al. [SoCC'22]:
    - Categories of root causes ←→ mitigation & detection strategies.
  - Dogga et al. [ATC'23]:
    - Root cause labeling ←→ bug report.

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

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

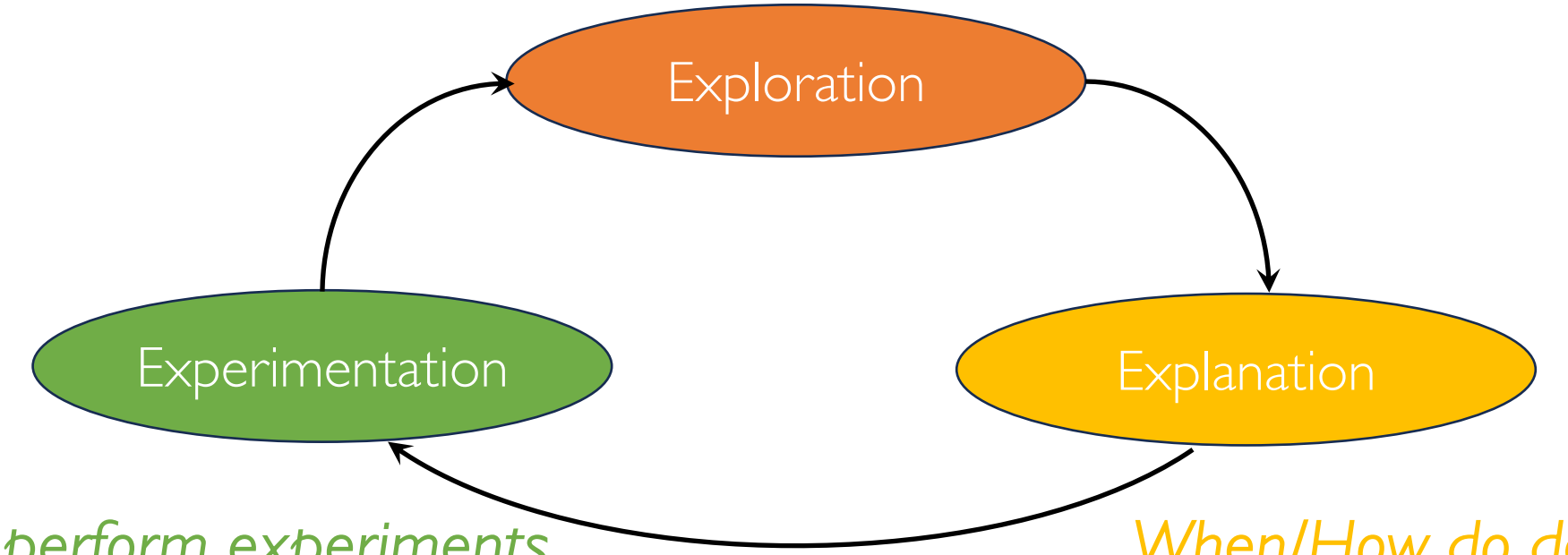
- A taxonomy of live debugging activities
  - Debugging strategies adopted
  - Challenges faced in each activity

- 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
		Source Code Anomaly
Experimentation (438)	Information Collection (209)	Instrumentation
		Probing
	Online Intervention (196)	
	Offline Reproduction (33)	

Outline: selected research questions

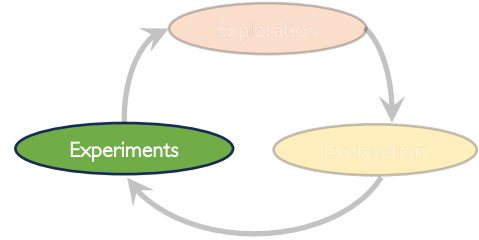
What are the challenges for observability?



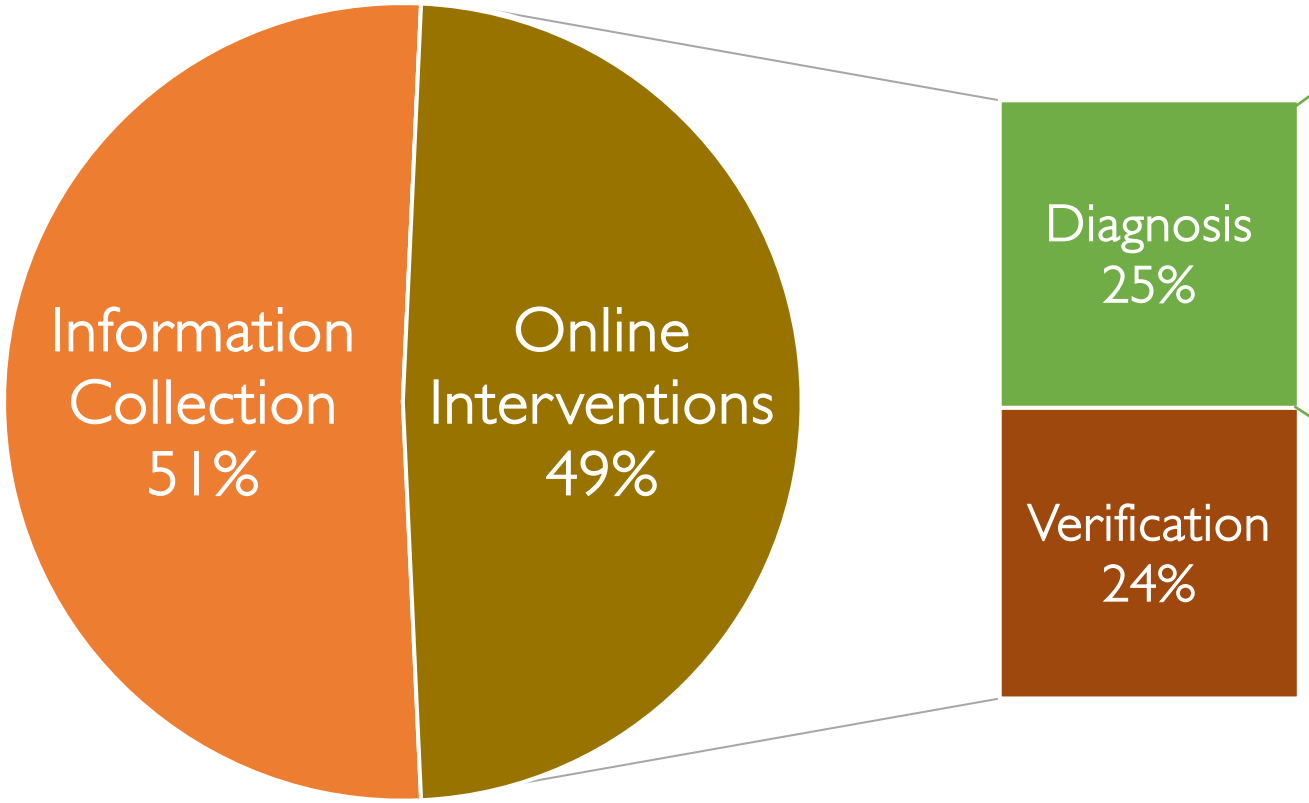
How to perform experiments and verify hypotheses?

When/How do developers formulate multiple hypotheses?

How to perform experiments & verify hypotheses?



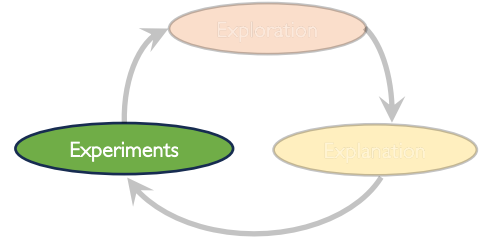
The Tie btw. Passive & Proactive Experiments



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**!

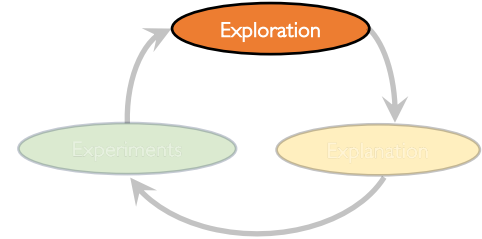
Developers want more intervention mechanisms



- 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

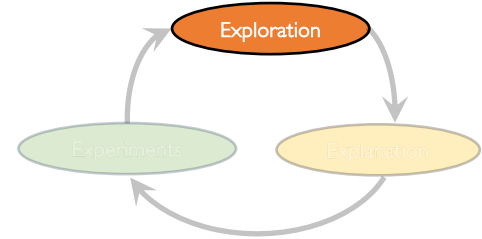
What are the challenges for observability?



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

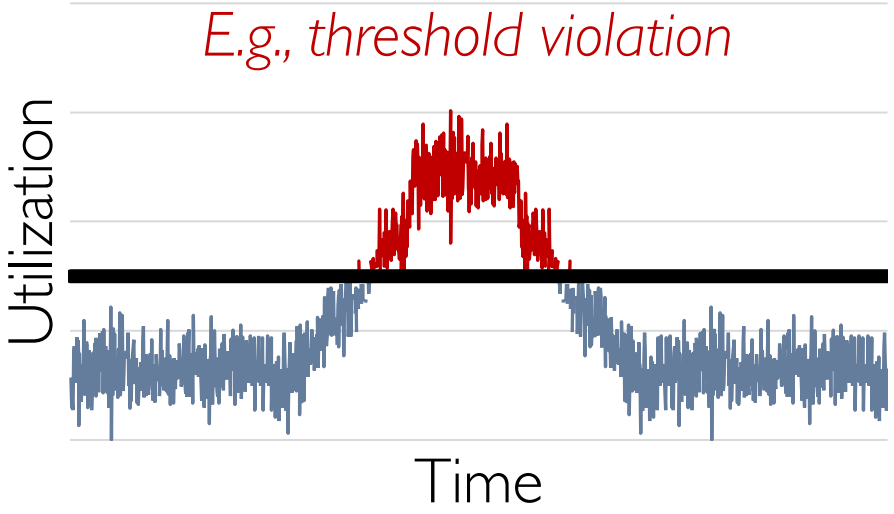
System-agnostic data	System-specific data
CPU Utilization	Zookeeper ZNodes
Traffic Volume	Spark Executors
Memory	HDFS Read Offsets
Latency	systemd cookie values
…	…

What are the challenges for observability?



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

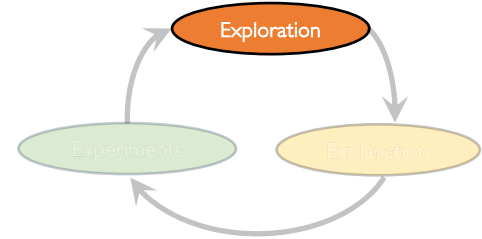
Conventional Invariants



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?

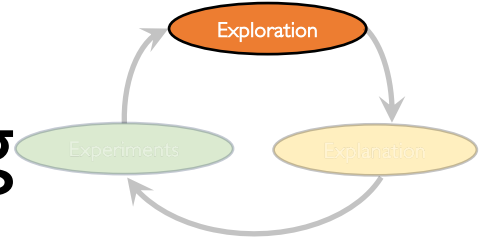


- **What** to observe – system-specific or system-agnostic data?
- **How** to alert – are default alerting rules (invariants) enough?

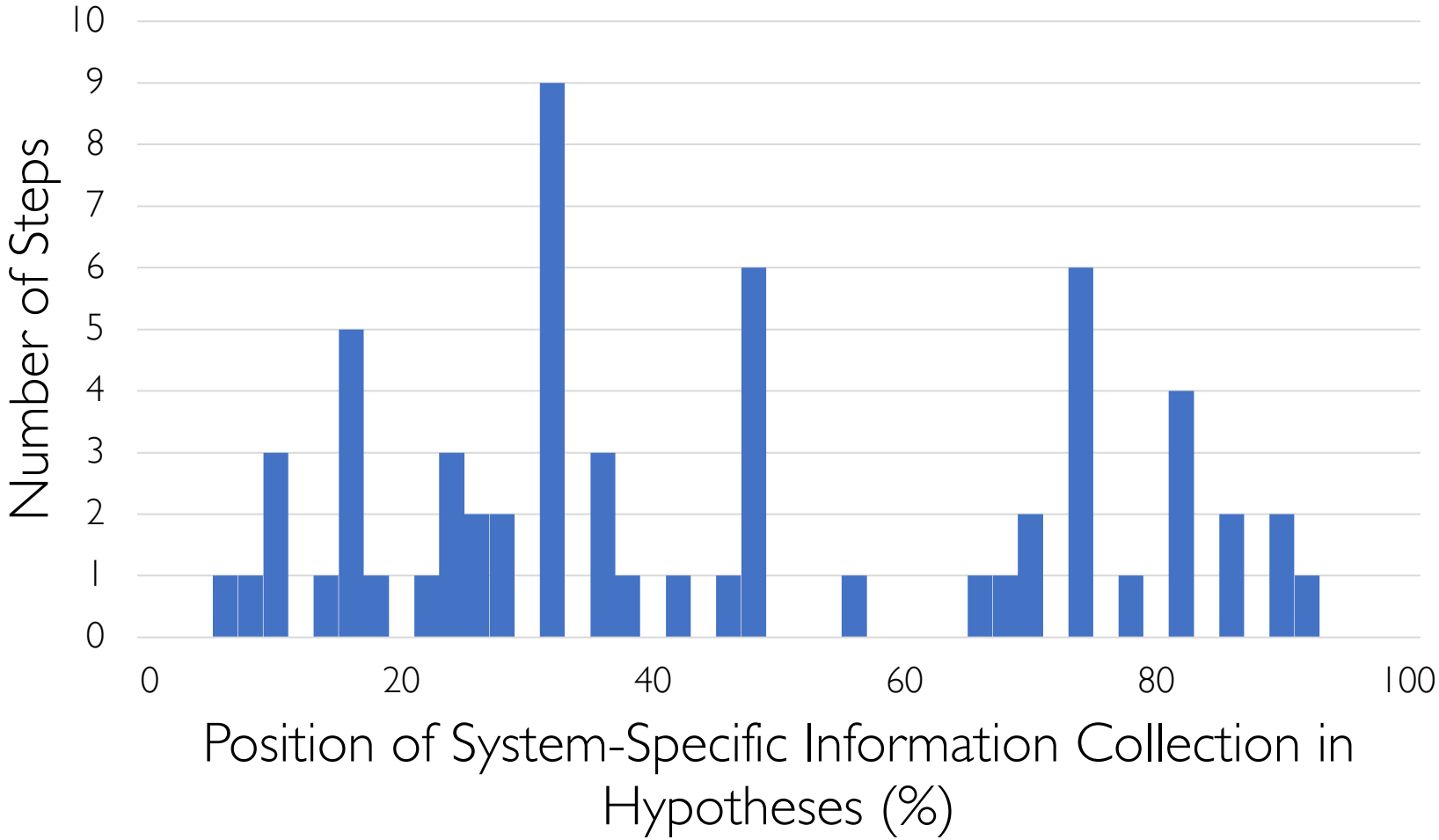
	System-agnostic data	System-specific data
Default invariants	**79.5%**	16.5%
Unconventional invariants	3.4%	0.6%

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

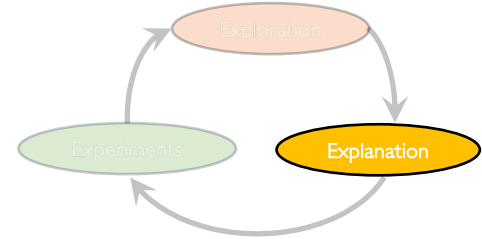
System-specific data usage spreads across debugging



- **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.

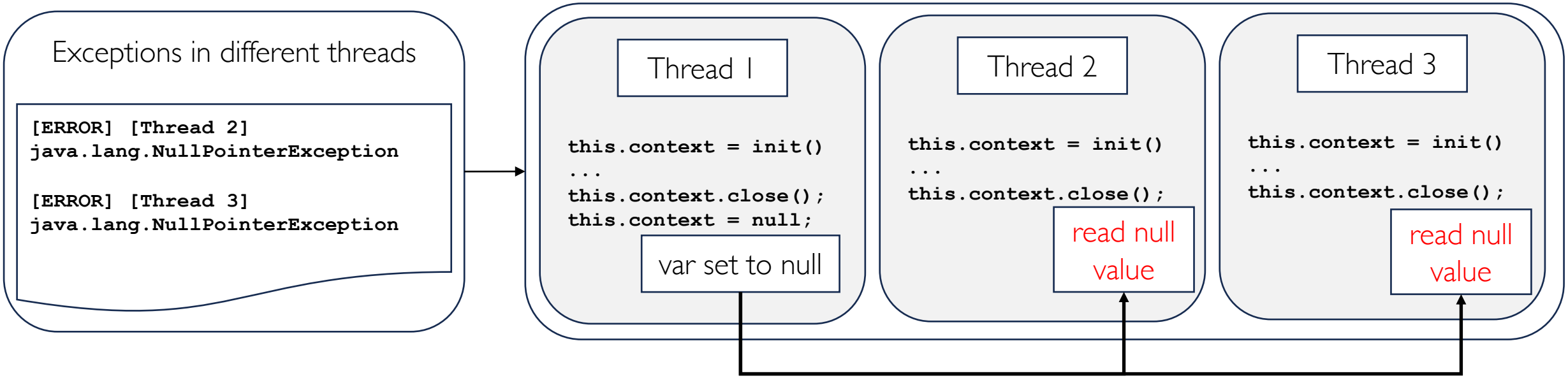
Model	Causal link
Component dependency model	Component dependency (static / dynamic)
Fault model	Control flow involving error (partial node failure, omission failure) or recovery
Data flow model	Data flow
Concurrency model	Interleaving (thread / network message)
Delay contribution model	Contributors,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