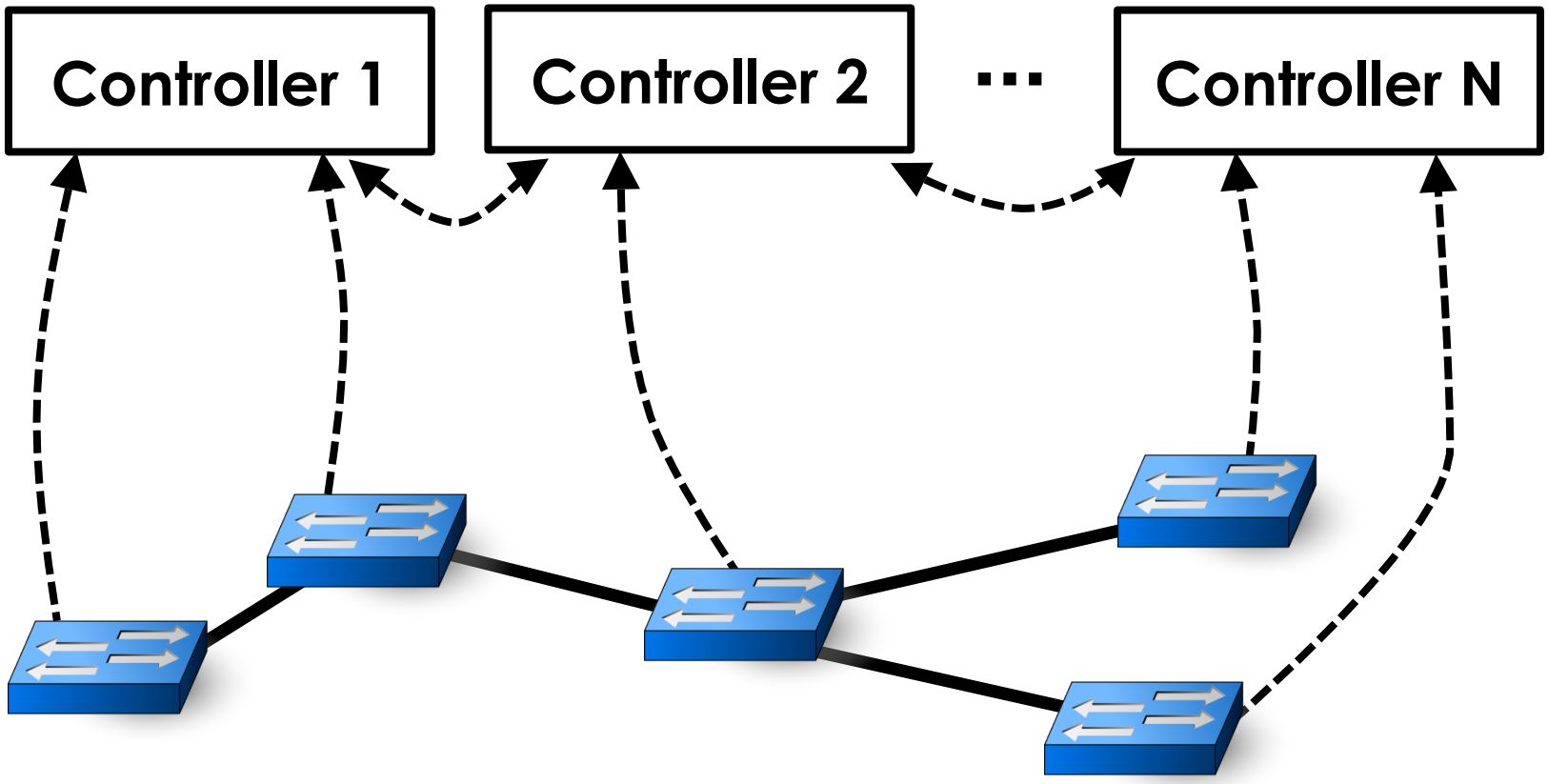
Troubleshooting SDN Control Software with Minimal Causal Sequences

Colin Scott, Andreas Wundsam, Barath Raghavan,
Aurojit Panda, Andrew Or, Jefferson Lai,
Eugene Huang, Zhi Liu, Ahmed El-Hassany,
Sam Whitlock, Hrishikesh B. Acharya, Kyriakos Zarifis,
Arvind Krishnamurthy, Scott Shenker

Berkeley
UNIVERSITY OF CALIFORNIA

SDN is a Distributed System



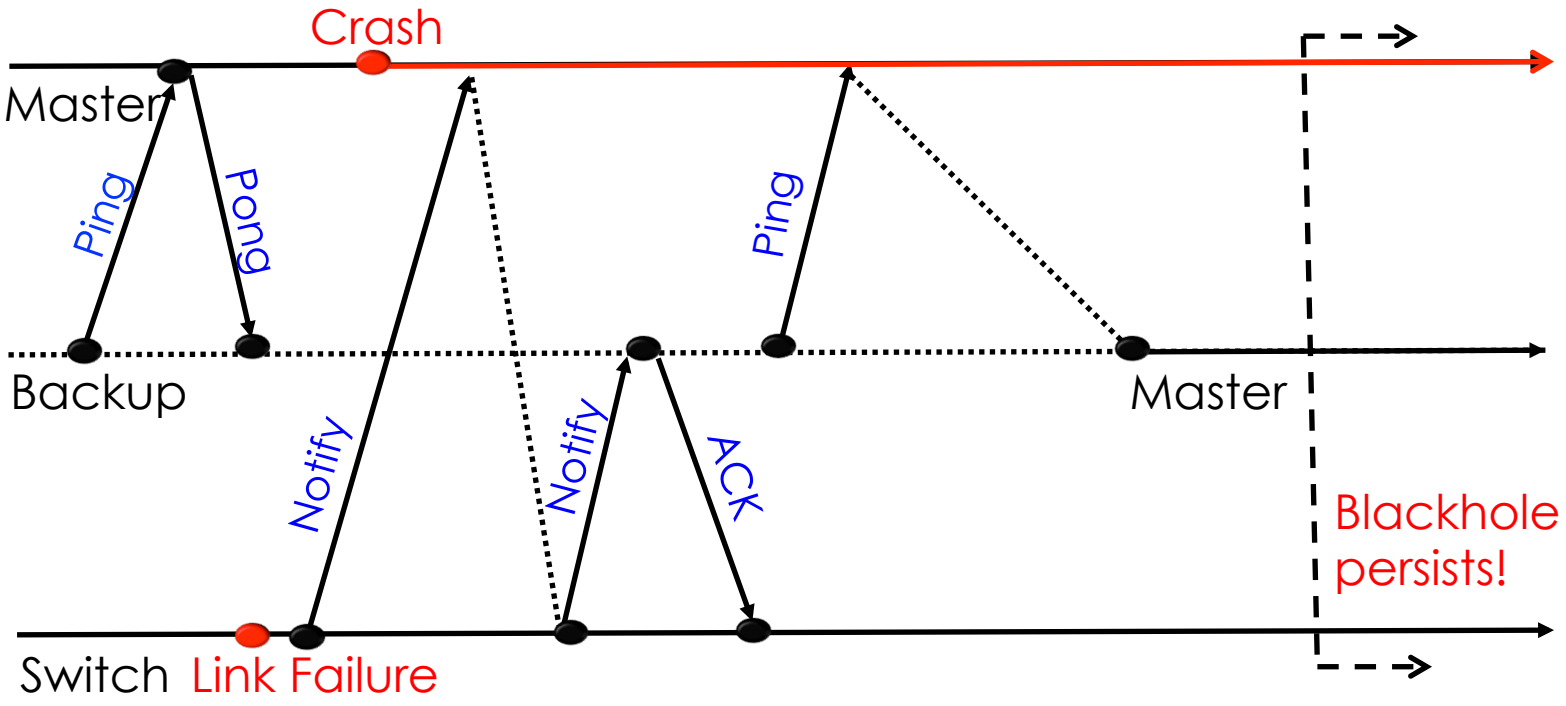
Distributed Systems are Bug-Prone

Distributed correctness faults:

- Race conditions
- Atomicity violations
- Deadlock
- Livelock
- …

+ Normal software bugs

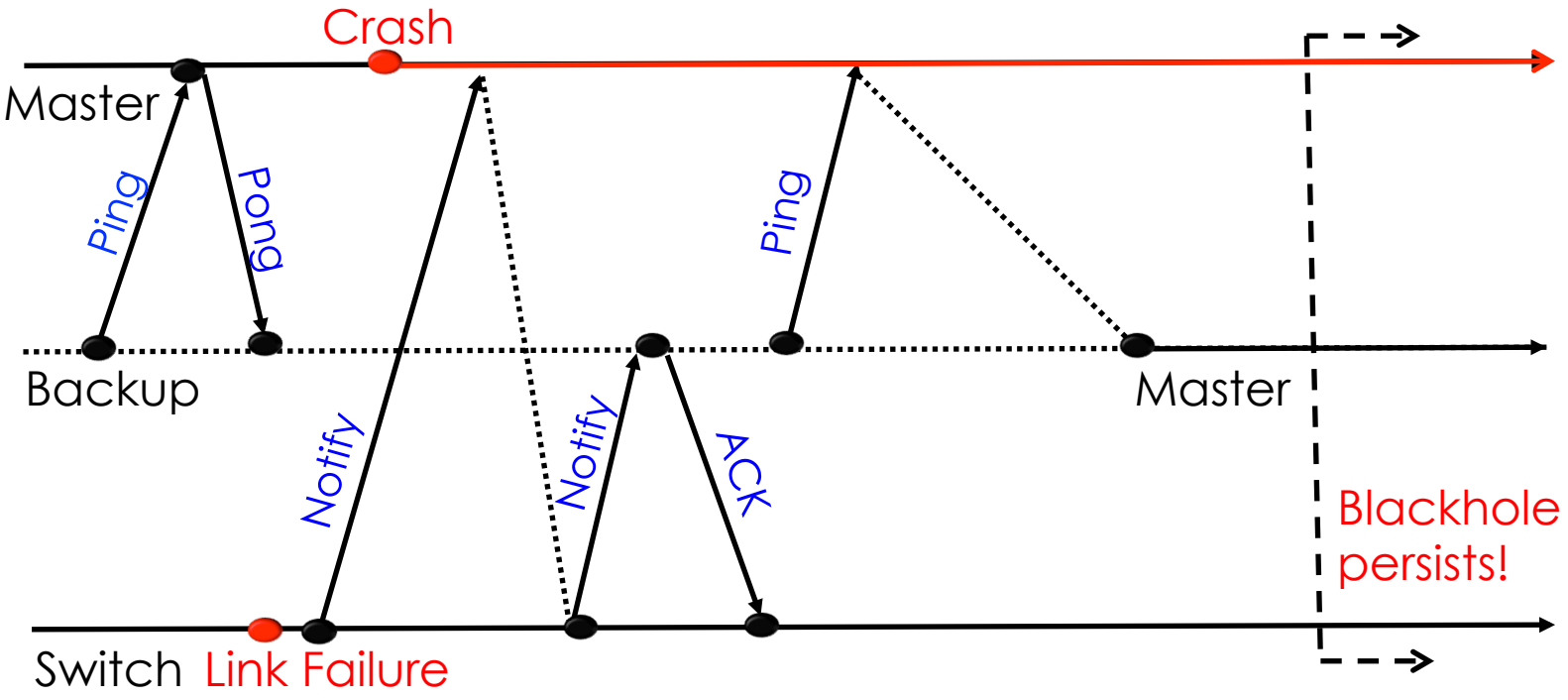
Example Bug (Floodlight, 2012)



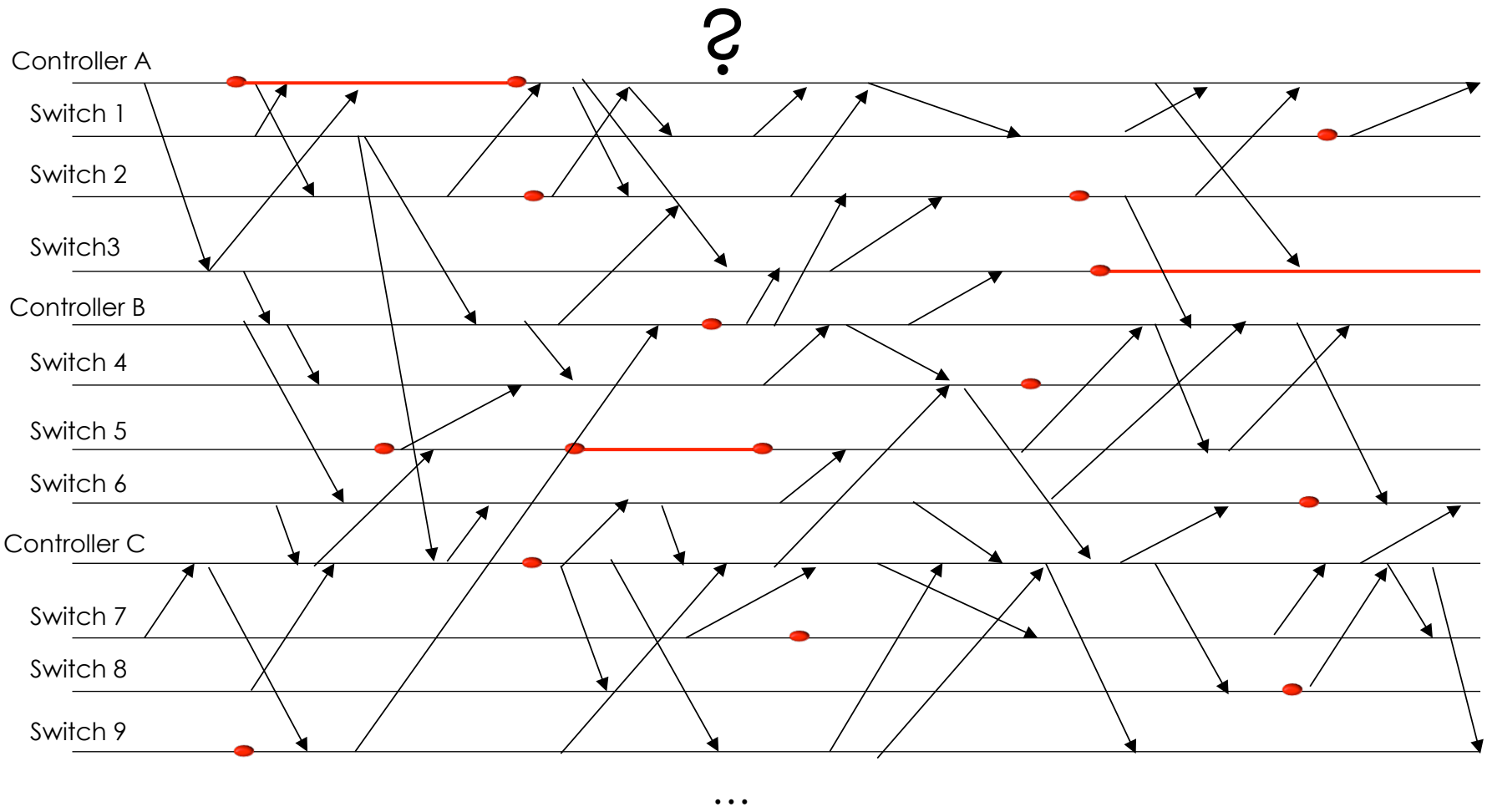
Best Practice: Logs

Human analysis of log files

Best Practice: Logs



Best Practice: Logs



Our Goal

Allow developers to focus on fixing the underlying bug

Problem Statement

Identify a **minimal** sequence of inputs that triggers the bug in a blackbox fashion

Why minimization?

Smaller event traces are easier to understand

G. A. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. Psychological Review '56.

Minimal Causal Sequence

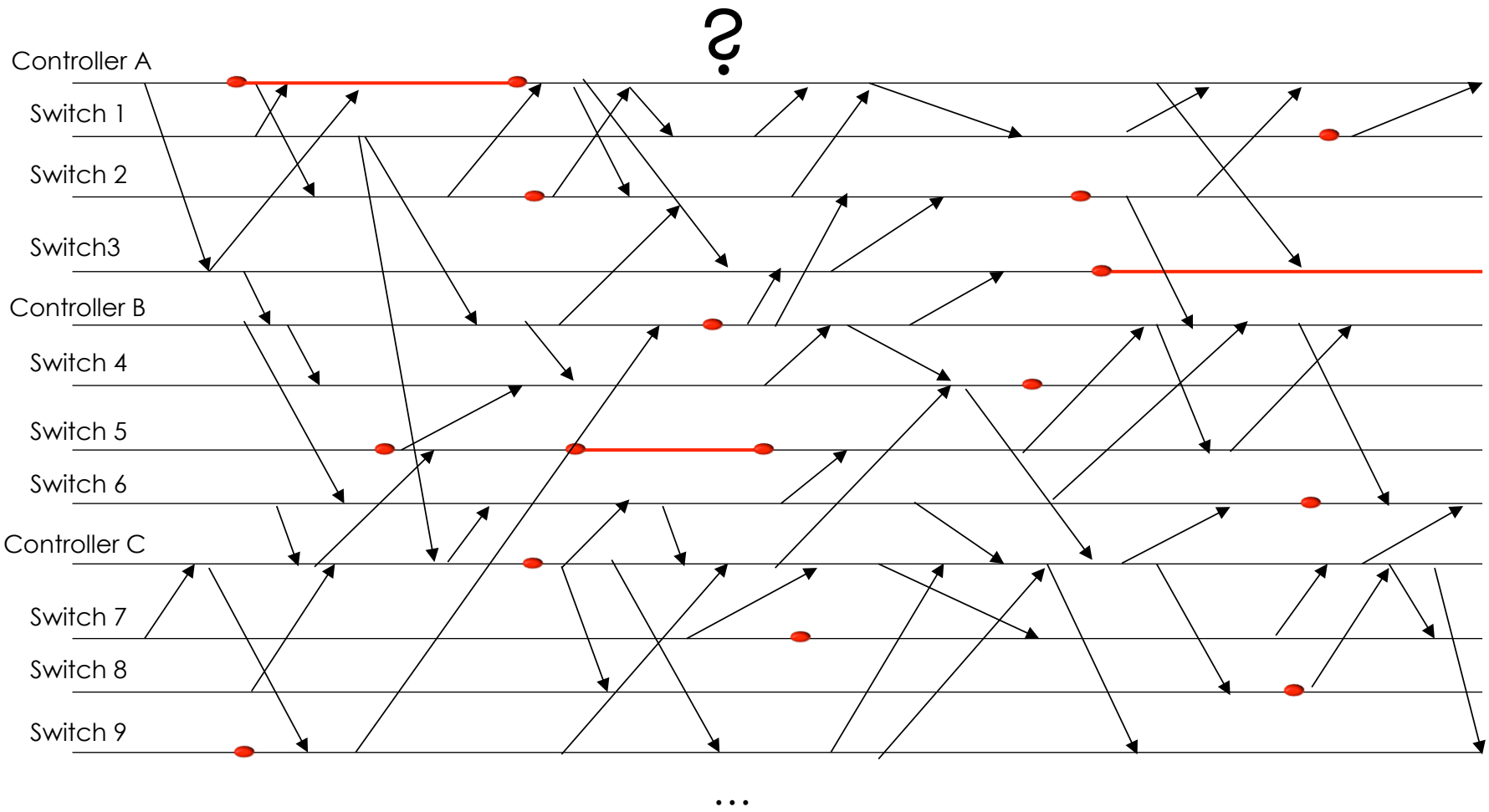
Output:

$$MCS \subset Trace \; s.t.$$

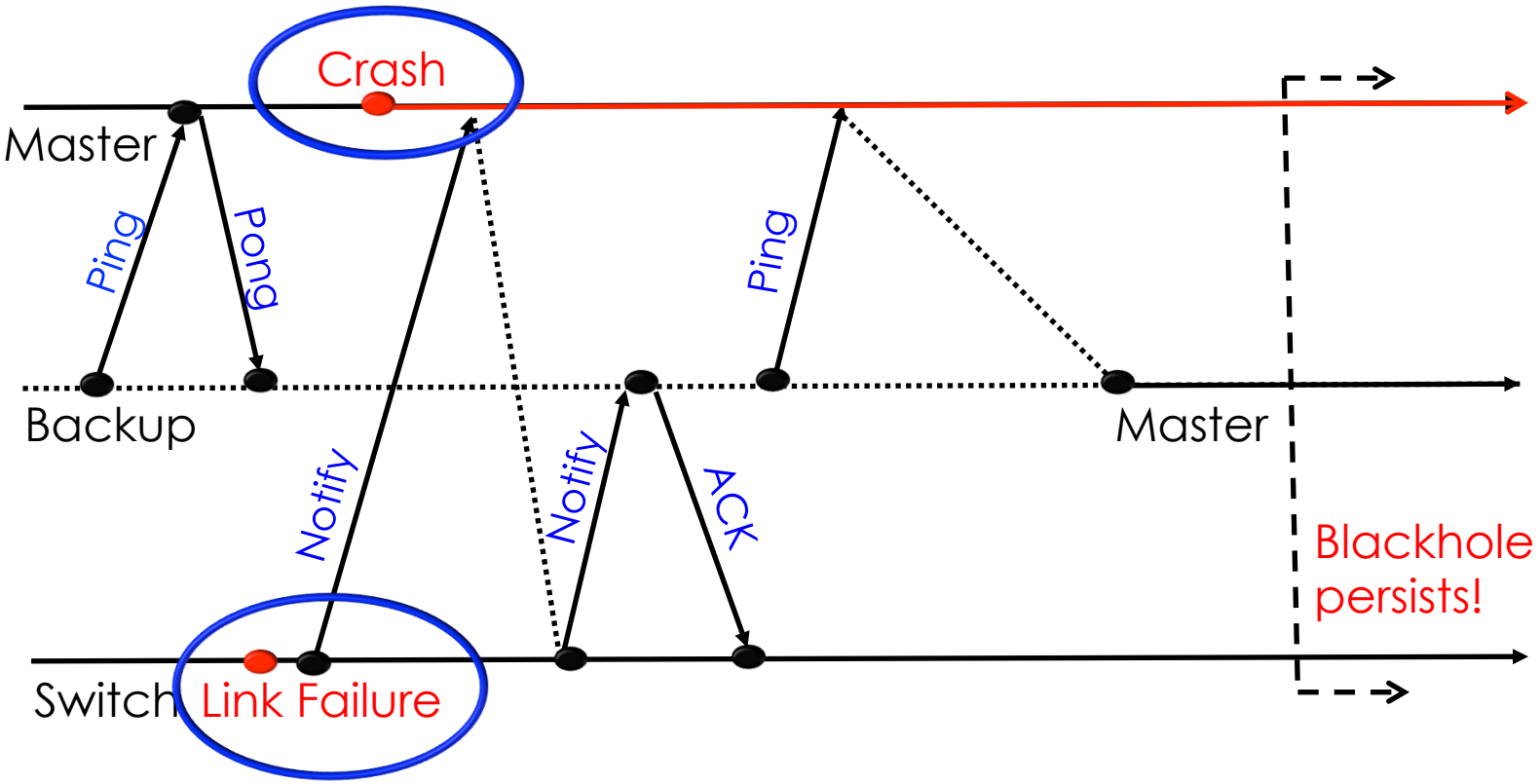
$$i.\; replay(MCS) \ni V$$ (i.e. violation occurs)

$$ii.\; \forall_{e \in MCS} replay(MCS - \{e\}) \not\ni V$$

Minimal Causal Sequence



Minimal Causal Sequence



Outline

- What are we trying to do?
- How do we do it?
- Does it work?

Where Bugs are Found

- Symptoms found:
  - On developer's local machine (unit and integration tests)

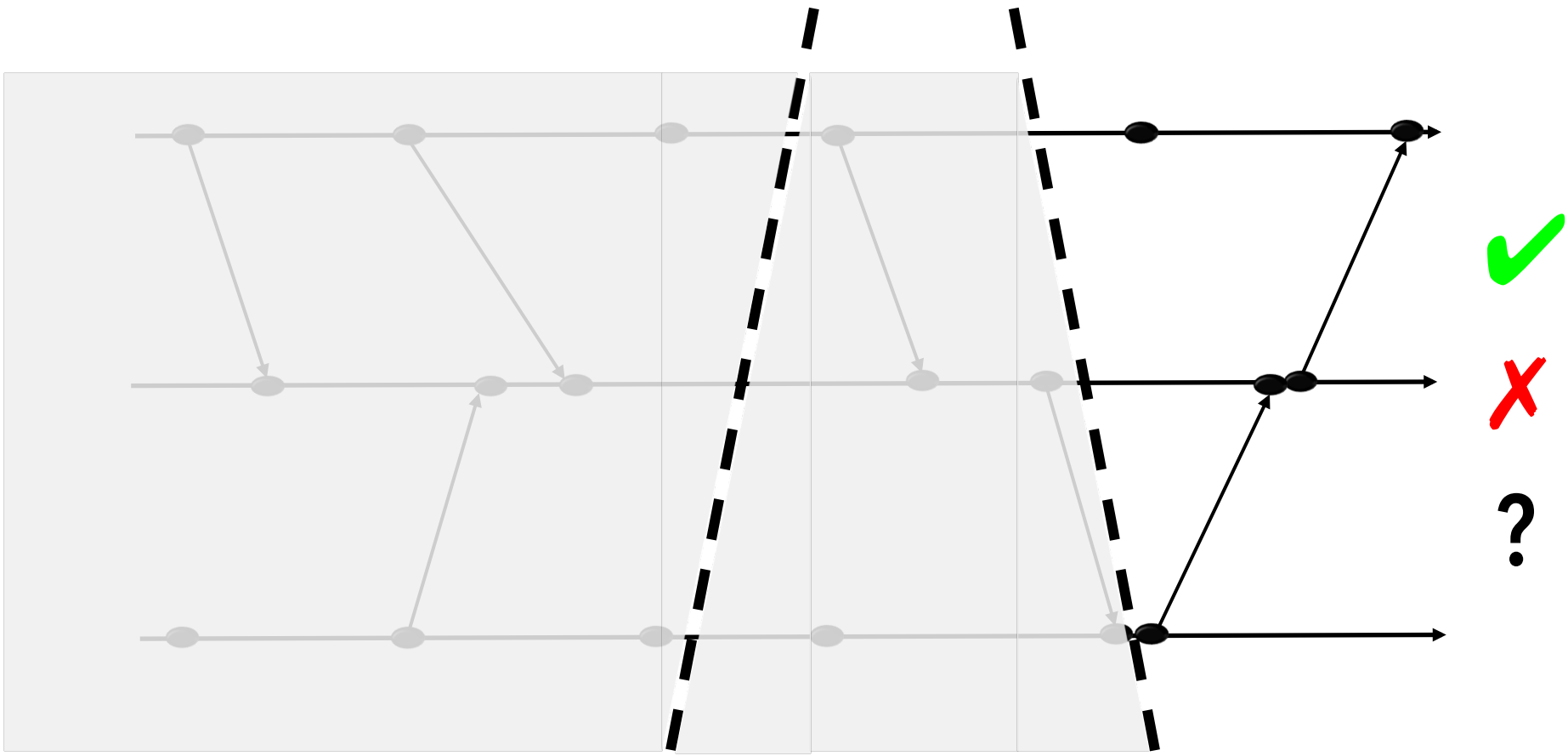
Where Bugs are Found

- Symptoms found:
  - On developer's local machine (unit and integration tests)
  - In production environment

Where Bugs are Found

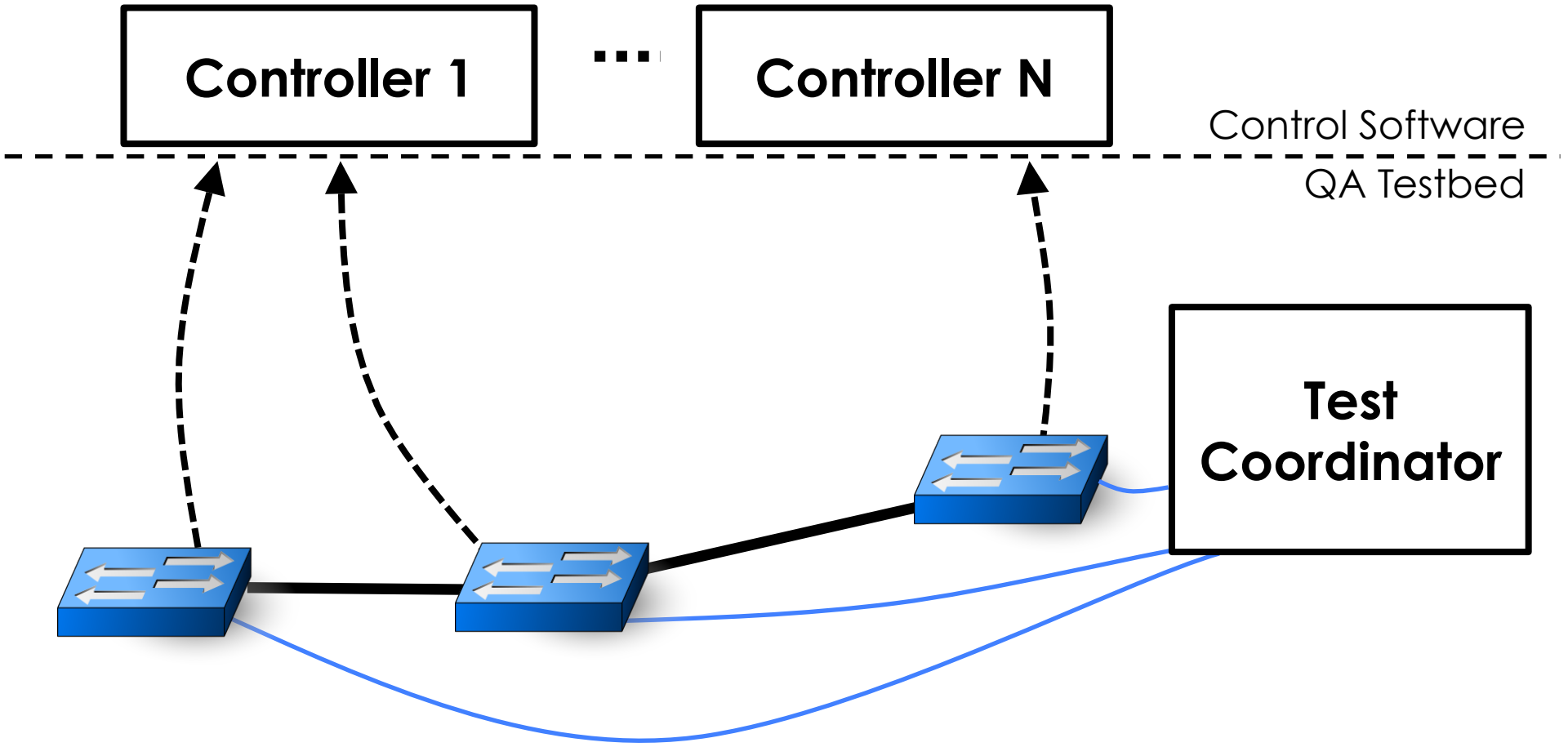
- Symptoms found:
  - On developer's local machine (unit and integration tests)
  - In production environment
  - On quality assurance testbed

Approach: Delta Debugging[1] Replay



1. A. Zeller et al. Simplifying and Isolating Failure-Inducing Input. IEEE TSE '02

Approach: Modify Testbed



Testbed Observables

- Invariant violation detected by testbed
- Event Sequence:

$$\tau_L = e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow \cdots e_m \rightarrow \cdots i_p$$

  - External events (link failures, host migrations,..) injected by testbed

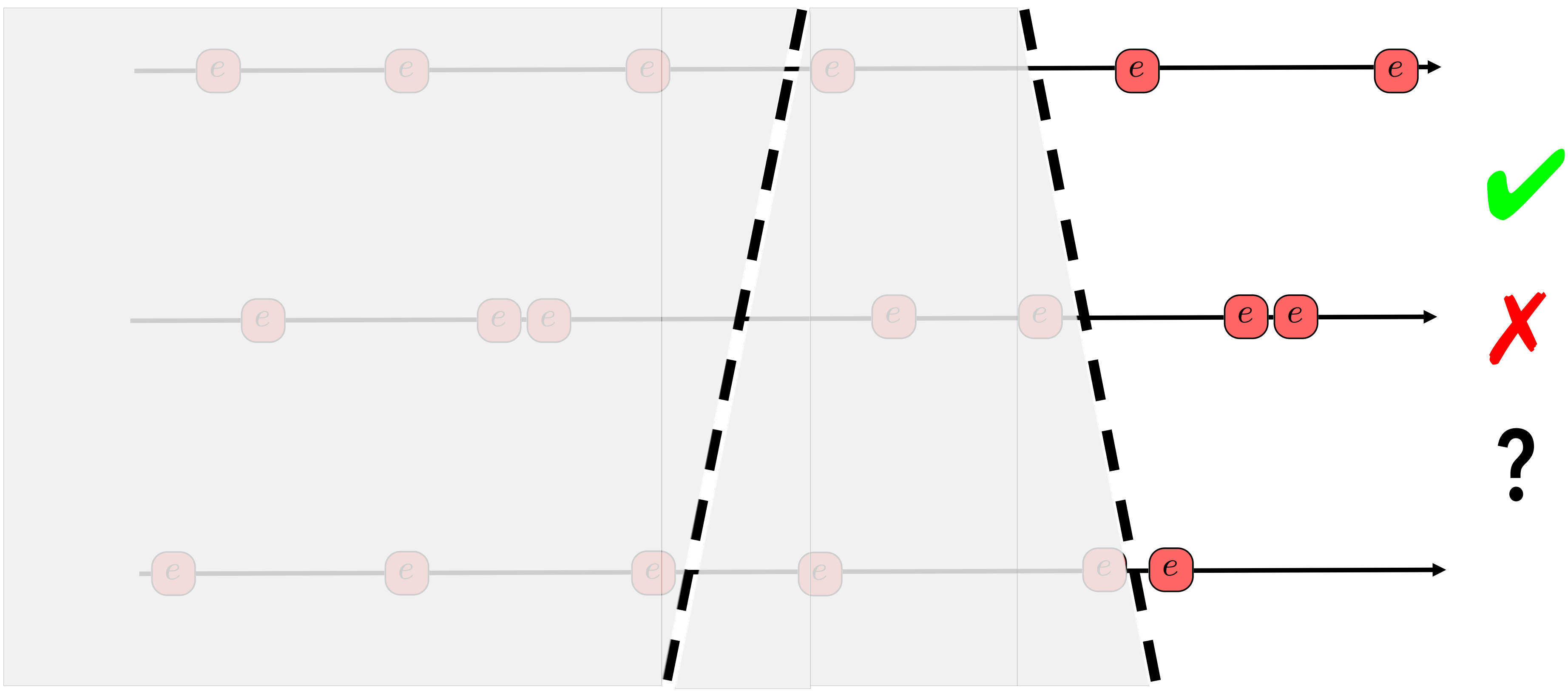
$$E_L = e_1, e_2 \cdots e_m$$

  - Internal events (message deliveries) observed by testbed (incomplete)

$$I_L = i_1, i_2 \cdots i_p$$

Approach: Delta Debugging[1] Replay

Events (link failures, crashes, host migrations) injected by test orchestrator



1. A. Zeller et al. Simplifying and Isolating Failure-Inducing Input. IEEE TSE '02

Key Point

Must Carefully Schedule Replay Events To Achieve Minimization!

Challenges

- Asynchrony
- Divergent execution
- Non-determinism

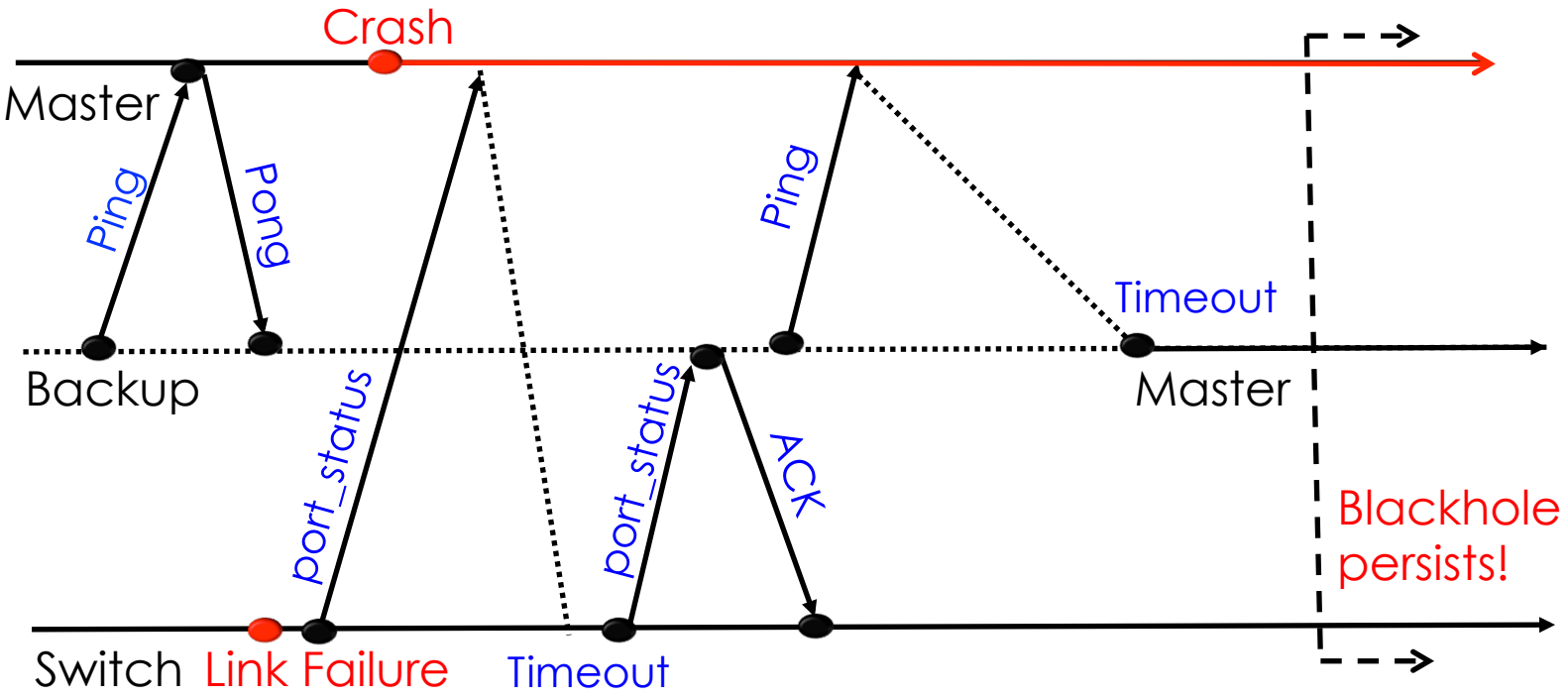
Challenge: Asynchrony

- Asynchrony definition:
  - No fixed upper bound on relative speed of processors
  - No fixed upper bound on time for messages to be delivered

Dwork & Lynch. Consensus in the Presence of Partial Synchrony. JACM '88

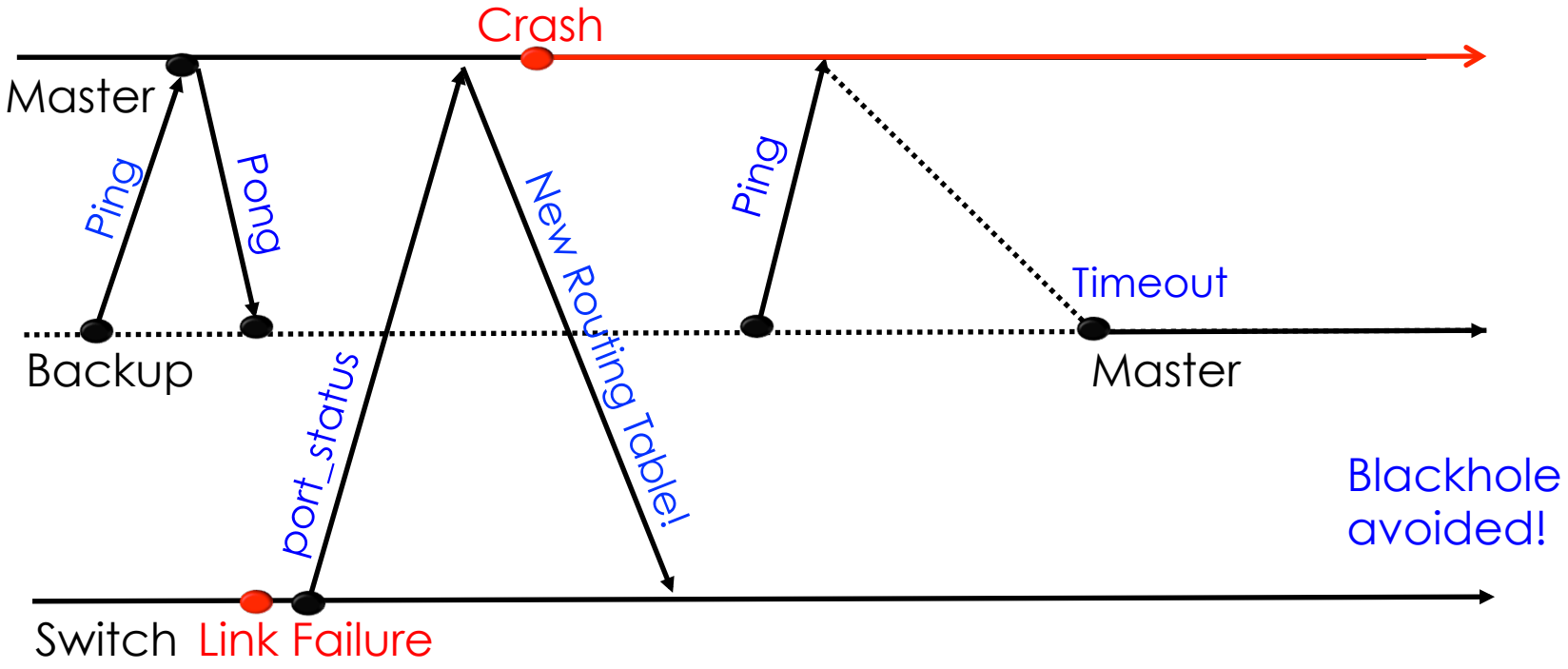
Challenge: Asynchrony

Need to maintain original event order

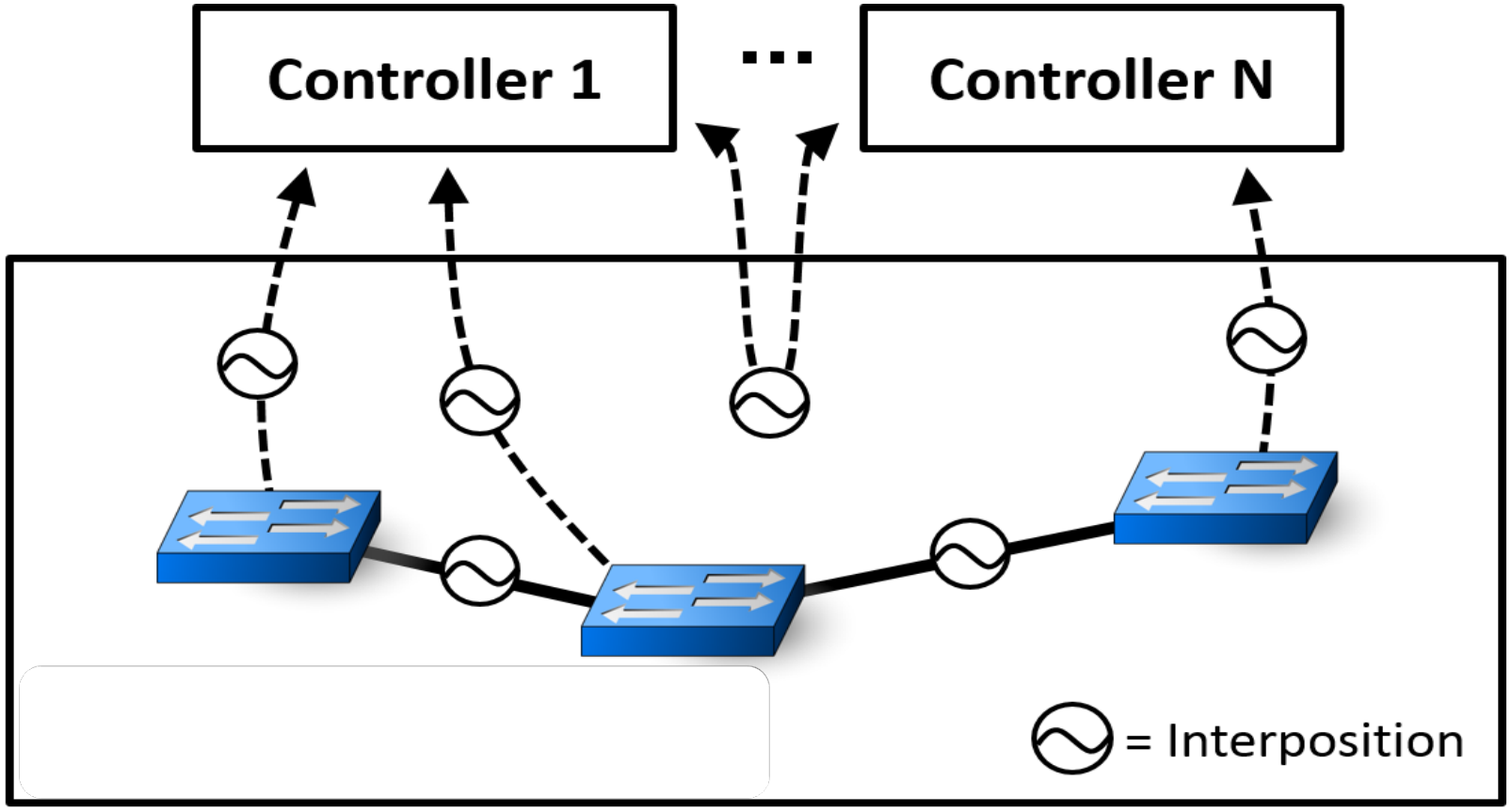


Challenge: Asynchrony

Need to maintain original event order



Coping with Asynchrony



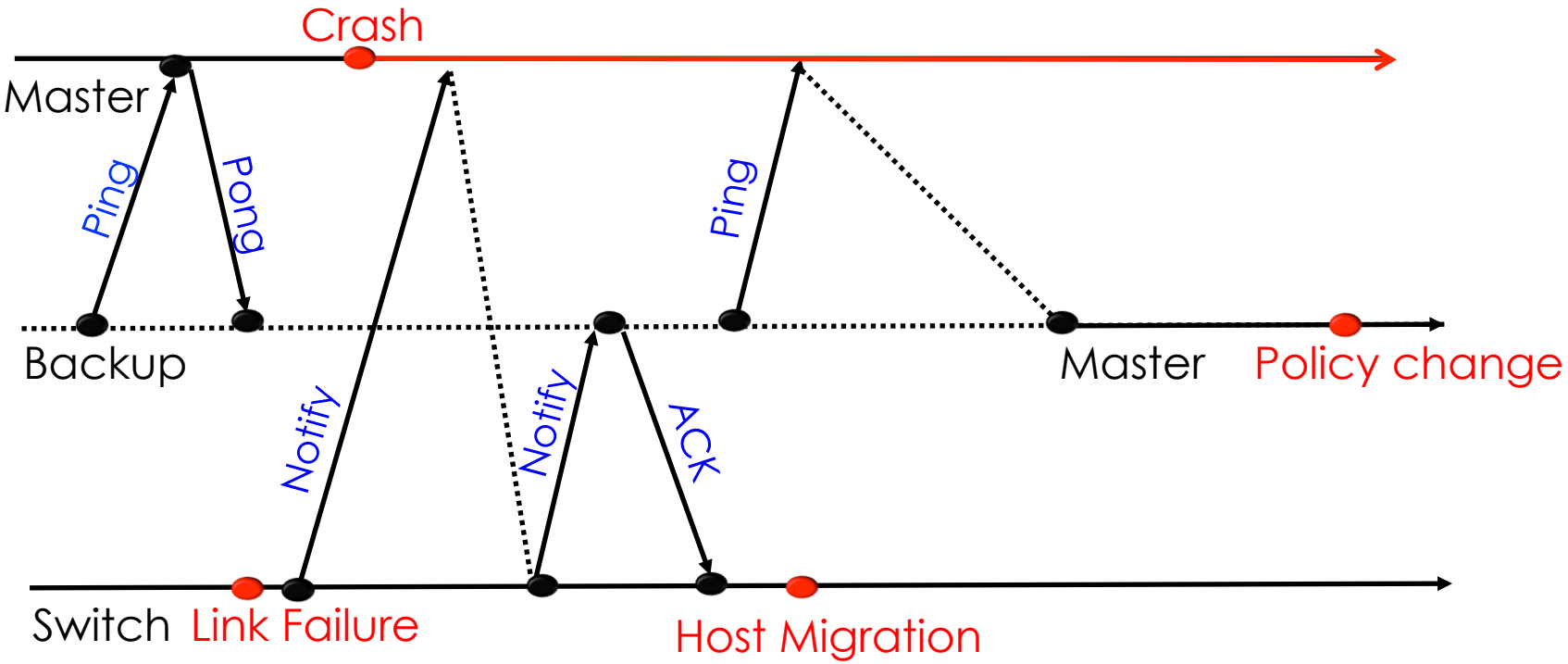
Use interposition to maintain causal dependencies

Challenge: Divergence

- Asynchrony
- Divergent execution
  - Syntactic Changes
  - Absent Events
  - Unexpected Events
- Non-determinism

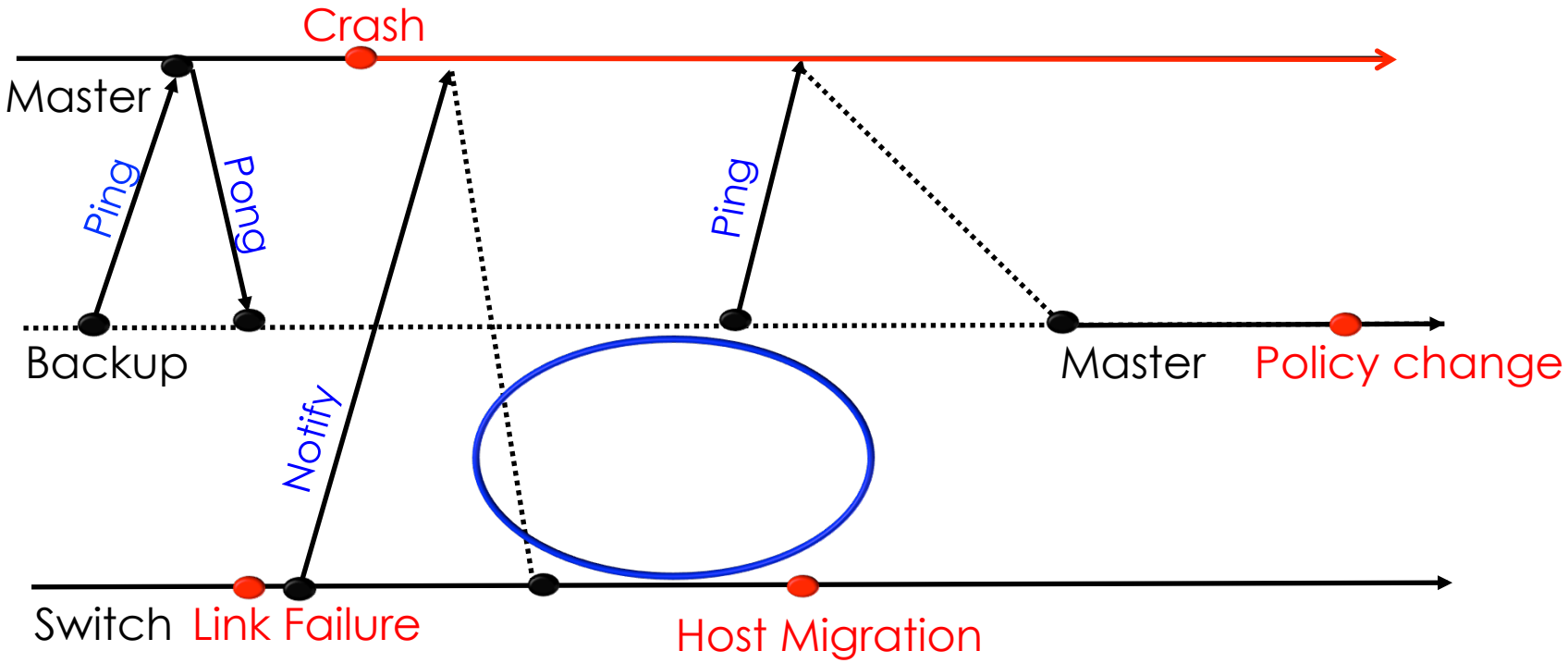
Divergence: Absent Internal Events

Prune Earlier Input..



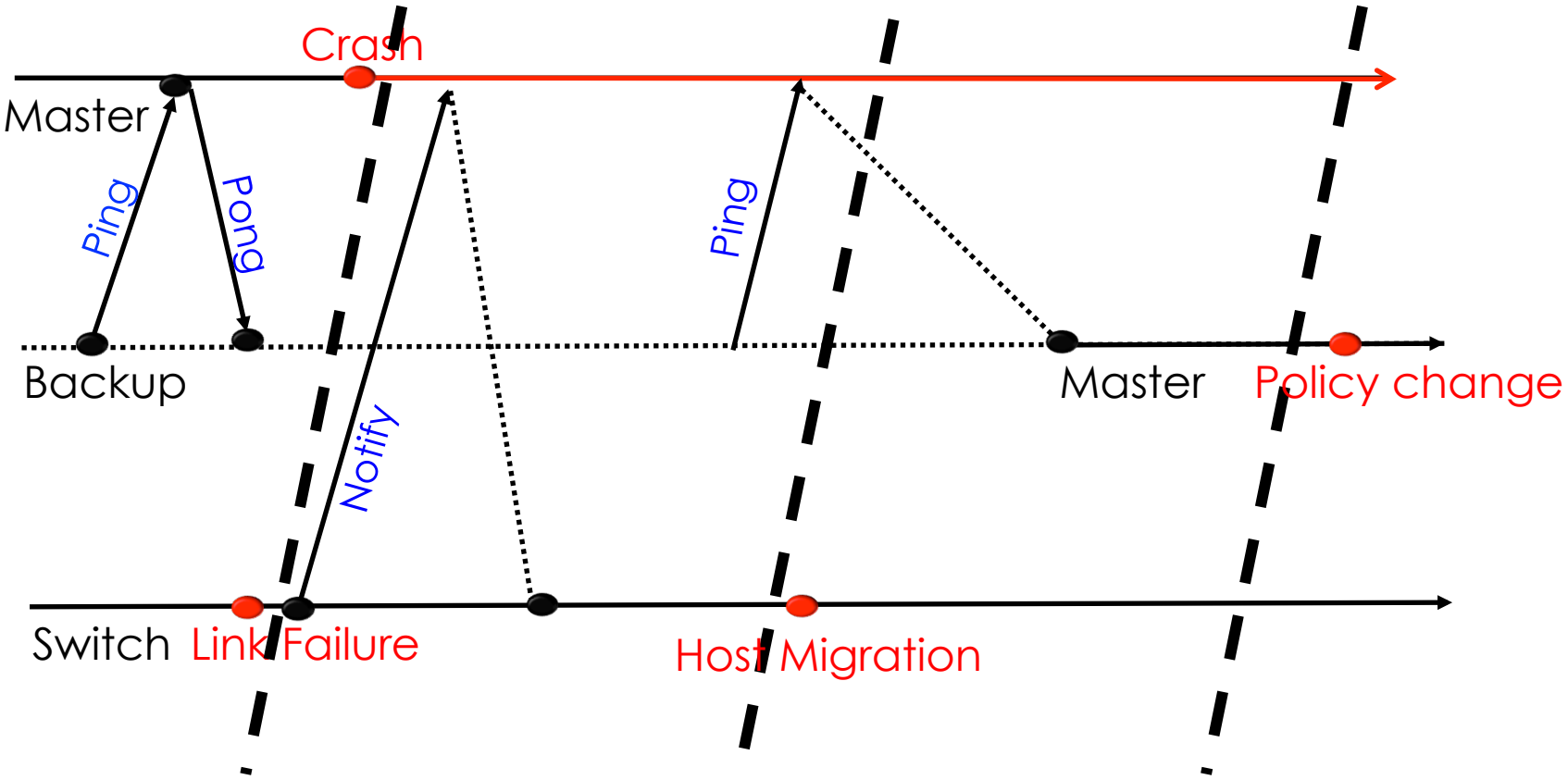
Divergence: Absent Internal Events

Some Events No Longer Appear



Solution: Peek Ahead

Infer which internal events will occur



Challenge: Non-determinism

- Asynchrony
- Divergent execution
- Non-determinism

Coping With Non-Determinism

- Replay multiple times per subsequence
- Assuming i.i.d., probability of not finding bug modeled by:

$$f(p,n) = (1-p)^n$$

- If not i.i.d., override gettimeofday(), multiplex sockets, interpose on logging statements

Approach Recap

- Replay events in QA testbed
- Apply delta debugging to inputs
- Asynchrony: interpose on messages
- Divergence: infer absent events
- Non-determinism: replay multiple times

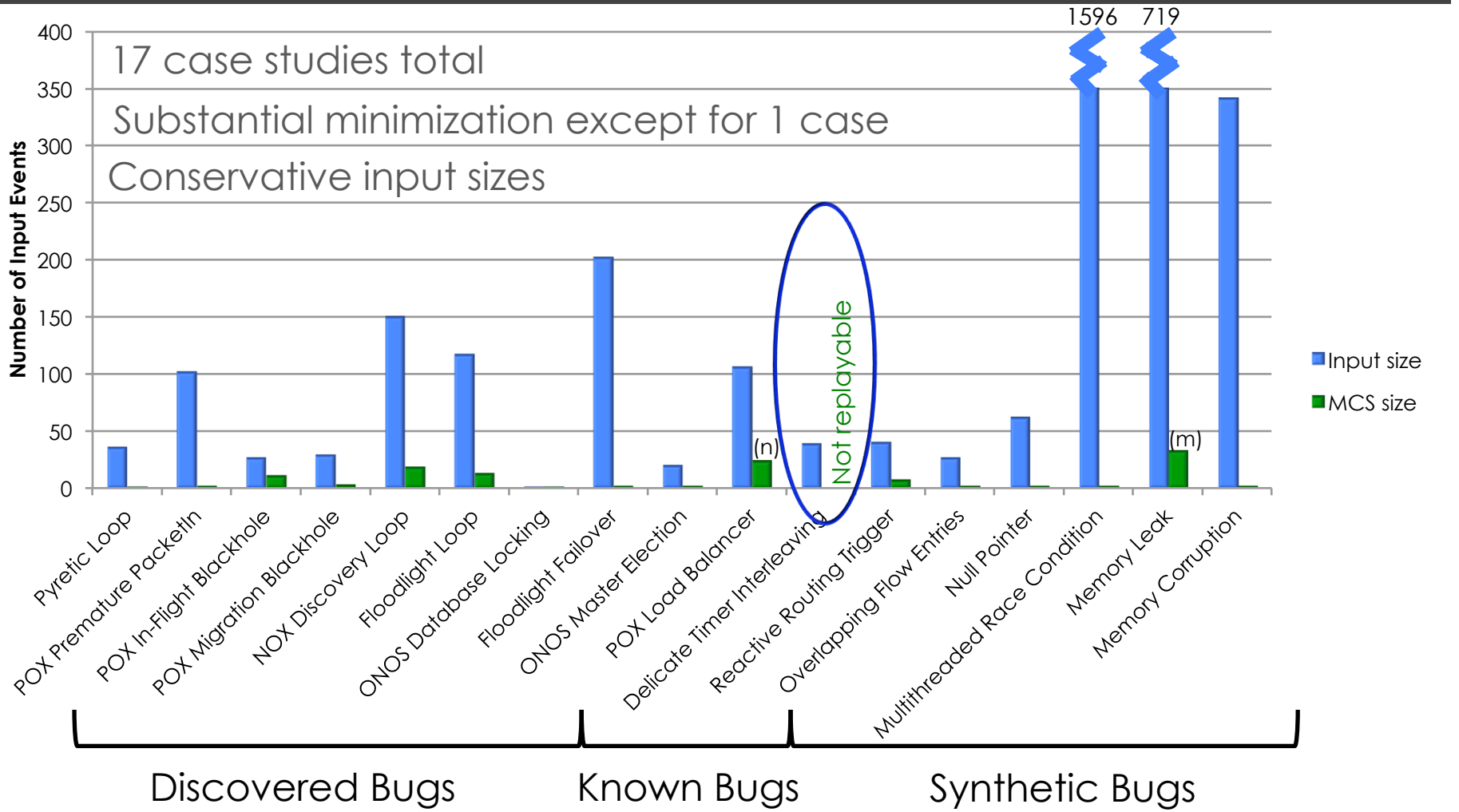
Outline

- What are we trying to do?
- How do we do it?
- Does it work?

Evaluation Methodology

- Evaluate on 5 open source SDN controllers (Floodlight, NOX, POX, Frenetic, ONOS)
- Quantify minimization for:
  - Synthetic bugs
  - Bugs found in the wild
- Qualitatively relay experience troubleshooting with MCSes

Case Studies



Comparison to Naïve Replay

- Naïve replay: ignore internal events
- Naïve replay often not able to replay at all
  - 5 / 7 discovered bugs not replayable
  - 1 / 7 synthetic bugs not replayable
- Naïve replay did better in one case
  - 2 event MCS vs. 7 event MCS with our techniques

Qualitative Results

- 15 / 17 MCSes useful for debugging
  - 1 non-replayable case (not surprising)
  - 1 misleading MCS (expected)

Related Work

[8] T. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. JACM '96.
[9] K. M. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. ACM TOCS '85.
[10] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, O. Fox, and E. Brewer. Pinpoint: Problem Determination in Large, Dynamic Internet Services. DSN '02.
[11] J. Choi and A. Zeller. Isolating Failure-Inducing Thread Schedules. SIGSOFT '02.
[12] K. Claessen and J. Hughes. QuickCheck: a Lightweight Tool for Random Testing of Haskell Programs. ICFP '00.
[13] K. Claessen, M. Palka, N. Smallbone, J. Hughes, H. Svensson, T. Arts, and U. Wiger. Finding Race Conditions in Erlang with QuickCheck and PULSE. ICFP '09.
[14] J. Clause and A. Orso. A Technique for Enabling and Supporting Debugging of Field Failures. ICSE '07.
[15] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay. OSDI '02.
[16] Floodlight Controller. http://tinyurl.com/ntjxa6l.
[17] Floodlight FIXME comment. Controller.java, line 605. http://tinyurl.com/af6nhjj.
[18] R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica. X-Trace: A Pervasive Network Tracing Framework. NSDI '07.
[19] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A Network Programming Language. ICFP '11.
[20] D. Geels, G. Altekar, S. Shenker, and I. Stoica. Replay Debugging For Distributed Applications. ATC '06.
[21] P. Godefroid and N. Nagappan. Concurrency at Microsoft - An Exploratory Survey. CAV '08.
[22] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network, Sec. 3.4. SIGCOMM '09.
[23] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System For Networks. CCR '08.
[24] D. Gupta, K. Yocum, M. Mcnett, A. C. Snoeren, A. Vahdat, and G. M. Voelker. To Infinity and Beyond: TimeWarped Network Emulation. NSDI '06.
[25] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. NSDI '14.
[26] J. Huang and C. Zhang. An Efficient Static Trace Simplification Technique for Debugging Concurrent Programs. SAS '11.
[27] J. Huang and C. Zhang. LEAN: Simplifying Concurrency Bug Reproduction via Replay-Supported Execution Reduction. OOPSLA '12.
[28] N. Jalbert and K. Sen. A Trace Simplification Technique for Effective Debugging of Concurrent Programs. FSE '10.
[29] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed Diagnosis in Enterprise Networks. SIGCOMM '09.
[30] P. Kazemian, M. Change, H. Zheng, G. Varghese, N. McKeown, and S. Whyte. Real Time Network Policy Checking Using Header Space Analysis. NSDI '13.
[31] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking For Networks. NSDI '12.
[32] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time. NSDI '13.
[33] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. CACM '78.
[34] K. H. Lee, Y. Zheng, N. Sumner, and X. Zhang. Toward Generating Reducible Replay Logs. PLDI '11.
[35] C.-C. Lin, V. Jalaparti, M. Caesar, and J. Van der Merwe. DEFINED: Deterministic Execution for Interactive Control-Plane Debugging. ATC '13.
[36] X. Liu. WiDs Checker: Combating Bugs in Distributed Systems. NSDI '07.
[37] X. Liu, Z. Guo, X. Wang, F. Chen, X. Lian, J. Tang, M. Wu, M. F. Kaashoek, and Z. Zhang. D^{3}S: Debugging Deployed Distributed Systems. NSDI '08.
[38] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King. Debugging the Data Plane with Anteater. SIGCOMM '11.
[39] J. Mccauley. POX: A Python-based OpenFlow Controller. http://www.noxrepo.org/pox/about-pox/.
[40] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. SIGCOMM CCR '08.
[41] G. A. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. Psychological Review '56.
[42] M. Musuvathi, S. Qadeer, T. Ball, G. Basler, P. A. Nainar, and I. Neamtiu. Finding and Reproducing Heisenbugs in Concurrent Programs. SOSP '08.
[43] ON.Lab. Open Networking Operating System. http://onlab.us/tools.html.
[44] S. Park, S. Lu, and Y. Zhou. CTrigger: Exposing Atomicity Violation Bugs from their Hiding Places. ASPLOS '09.
[45] S. Park, Y. Zhou, W. Xiong, Z. Yin, R. Kaushik, K. H. Lee, and S. Lu. PRES: Probabilistic Replay with Execution Sketching on Multiprocessors. SOSP '09.
[46] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending Networking into the Virtualization Layer. HotNets '09.
[47] J. Regehr, Y. Chen, P. Cuoq, E. Eide, C. Ellison, and X. Yang. Test-case Reduction for C Compiler Bugs. PLDI '12.
[48] P. Reynolds, C. Killian, J. L. Winer, J. C. Mogul, M. A. Shah, and A. Vadhat. Pip: Detecting the Unexpected in Distributed Systems. NSDI '06.
[49] V. Soundararajan and K. Govil. Challenges in Building Scalable Virtualized Datacenter Management. OSR '10.
[50] S. Tallam, C. Tian, R. Gupta, and X. Zhang. Enabling Tracing of Long-Running Multithreaded Programs via Dynamic Execution Reduction. ISSTA '07.
[51] G. Tel. *Introduction to Distributed Algorithms*. Thm. 2.21. Cambridge University Press, 2000.
[52] A. Thompson. http://tinyurl.com/qgc387k.
[53] J. Tucek, S. Lu, C. Huang, S. Xanthos, and Y. Zhou. Triage: Diagnosing Production Run Failures at the User's Site. SOSP '07.
[54] M. Weiser. Program Slicing. ICSE '81.
[55] A. Whitaker, R. Cox, and S. Gribble. Configuration Debugging as Search: Finding the Needle in the Haystack. SOSP '04.
[56] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann. OFRewind: Enabling Record and Replay Troubleshooting for Networks. ATC '11.
[57] S. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. A Survey of Fault Localization Techniques in Computer Networks. Science of Computer Programming '04.
[58] A. Zeller. Yesterday, my program worked. Today, it does not. Why? ESEC/FSE '99.
[59] A. Zeller and R. Hildebrandt. Simplifying and Isolating Failure-Inducing Input. IEEE TSE '02.
[60] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown. Automatic Test Packet Generation. CoNEXT '12.
[61] H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, N. McKeown, and A. Vahdat. Libra: Divide and Conquer to Verify Forwarding Tables in Huge Networks. NSDI '14.

Conclusion

- Possible to automatically minimize execution traces for SDN control software
- System (23K+ lines of Python) evaluated on 5 open source SDN controllers (Floodlight, NOX, POX, Frenetic, ONOS) and one proprietary controller

ucb-sts.github.com/sts/

- Currently generalizing, formalizing approach

Backup

Related work

- Thread Schedule Minimization
  - Isolating Failure-Inducing Thread Schedules. SIGSOFT '02.
  - A Trace Simplification Technique for Effective Debugging of Concurrent Programs. FSE '10.
- Program Flow Analysis
  - Enabling Tracing of Long-Running Multithreaded Programs via Dynamic Execution Reduction. ISSTA '07.
  - Toward Generating Reducible Replay Logs. PLDI '11.
- Best-Effort Replay of Field Failures
  - A Technique for Enabling and Supporting Debugging of Field Failures. ICSE '07.
  - Triage: Diagnosing Production Run Failures at the User's Site. SOSP '07.

Bugs are costly and time consuming

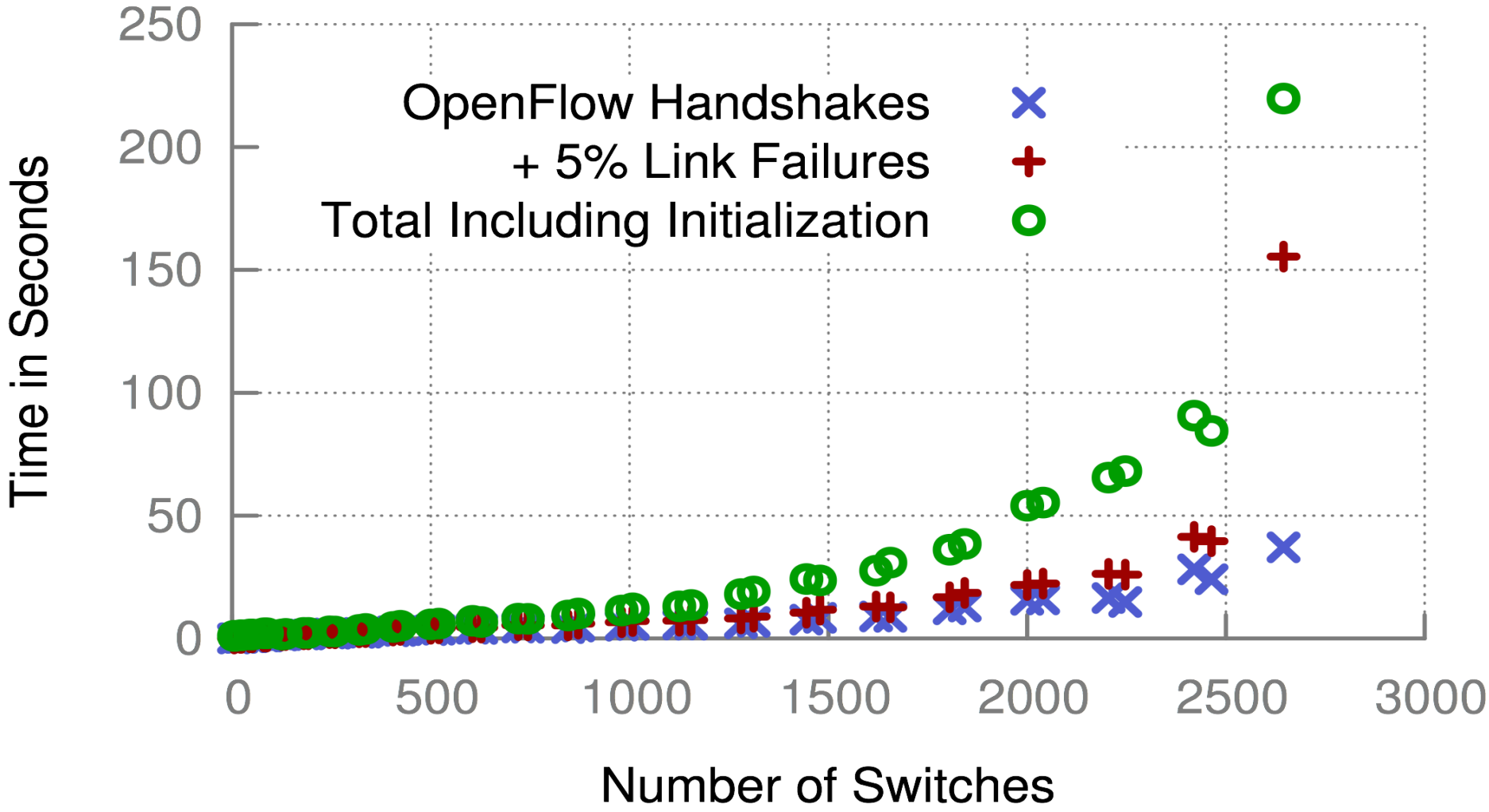
- Software bugs cost US economy $59.5 Billion in 2002 [1]
- Developers spend ~50% of their time debugging [2]
- Best developers devoted to debugging

1. National Institute of Standards and Technology 2002 Annual Report
2. P. Godefroid et al., Concurrency at Microsoft- An Exploratory Study. CAV '08

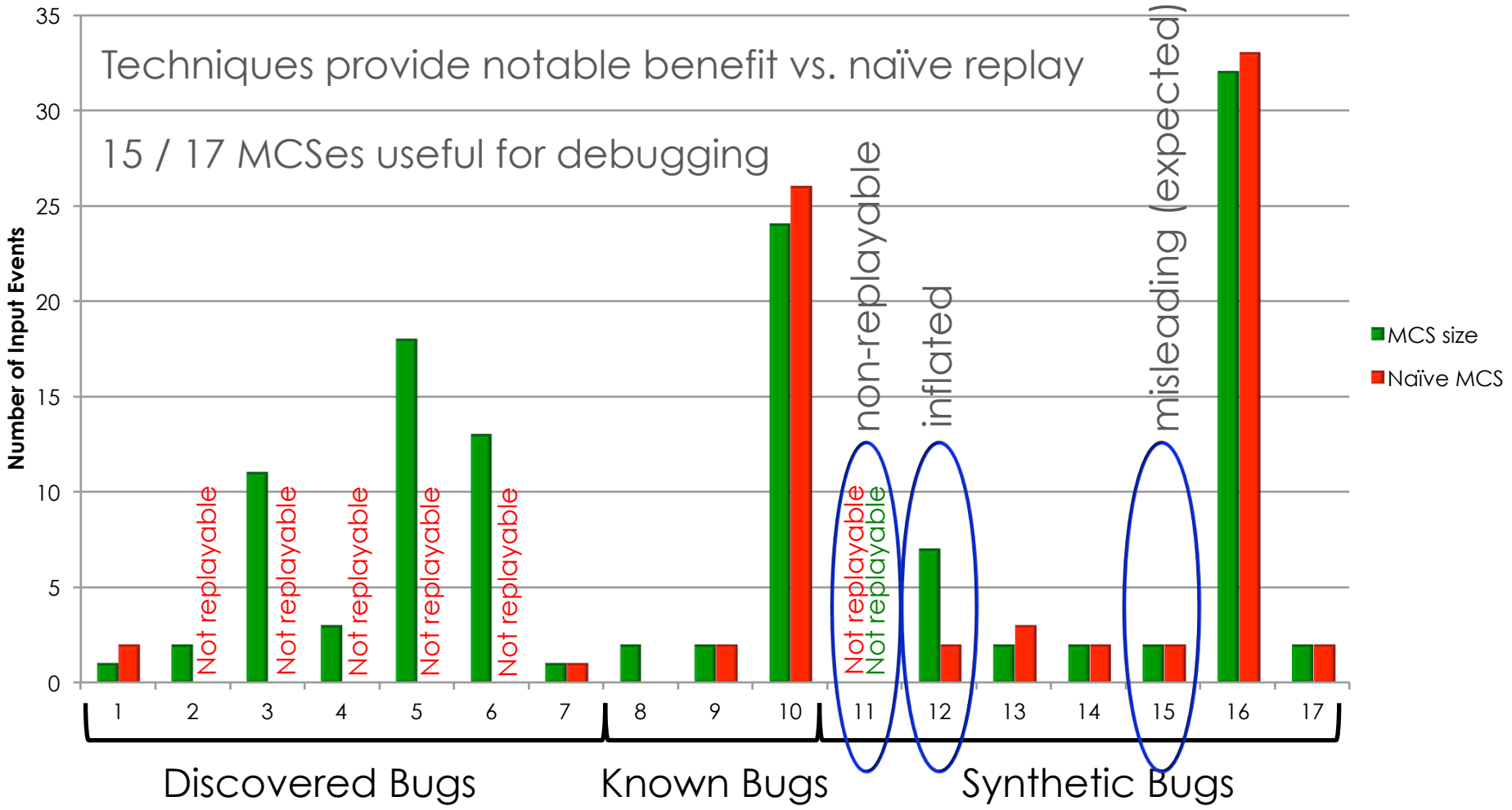
Ongoing work

- Formal analysis of approach
- Apply to other distributed systems (databases, consensus protocols)
- Investigate effectiveness of various interposition points
- Integrate STS into ONOS (ON.Lab) development workflow

Scalability



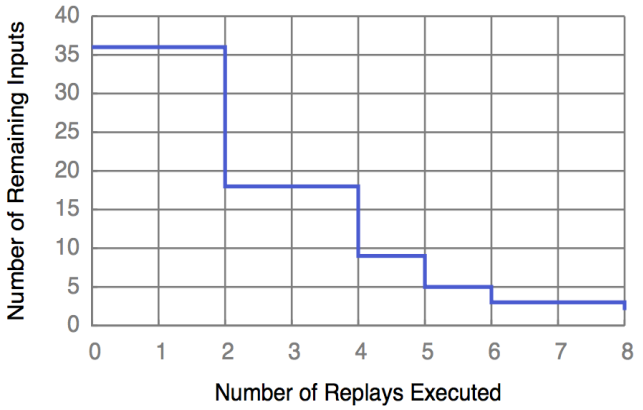
Case Studies



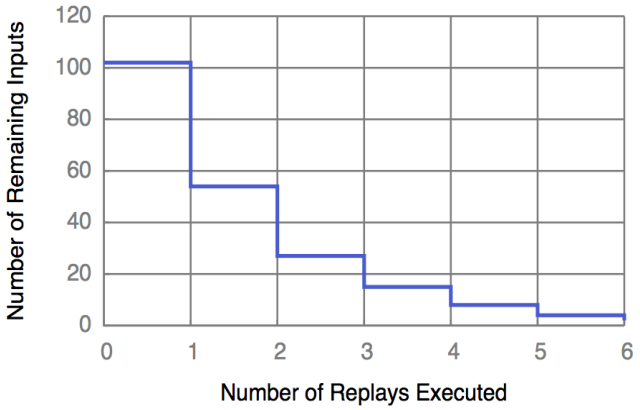
Case Studies

	Bug Name	Topology	Runtime (s)	Input Size	MCS Size	MCS WI	MCS Helpful?
Newly Found	Pyretic Loop	3 switch mesh	266.2	36	1	2	Yes
	POX Premature PacketIn	4 switch mesh	249.1	102	2	NR	Yes
	POX In-Flight Blackhole	2 switch mesh	1478.9	27	11	NR	Yes
	POX Migration Blackhole	4 switch mesh	1796.0	29	3	NR	Yes
	NOX Discovery Loop	4 switch mesh	4990.9	150	18	NR	Indirectly
	Floodlight Loop	3 switch mesh	27930.6	117	13	NR	Yes
	ONOS Database Locking	2 switch mesh	N/A	1	1	1	N/A
Known	Floodlight Failover	2 switch mesh	-	202	2	-	Yes
	ONOS Master Election	2 switch mesh	2746.0	20	2	2	Yes
	POX Load Balancer	3 switch mesh	2396.7	106	24 (N+1)	26	Yes
Synthetic	Delicate Timer Interleaving	3 switch mesh	N/A	39	NR	NR	No
	Reactive Routing Trigger	3 switch mesh	525.2	40	7	2	Indirectly
	Overlapping Flow Entries	2 switch mesh	115.4	27	2	3	Yes
	Null Pointer	20 switch FatTree	157.4	62	2	2	Yes
	Multithreaded Race Condition	10 switch mesh	36967.5	1596	2	2	Indirectly
	Memory Leak	2 switch mesh	15022.6	719	32 (M+2)	33	Indirectly
	Memory Corruption	4 switch mesh	145.7	341	2	2	Yes

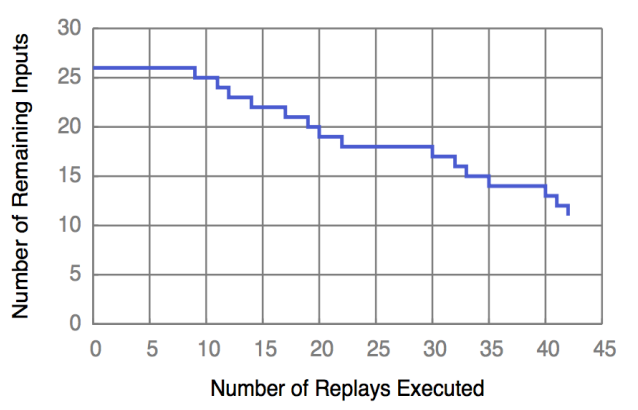
Runtime



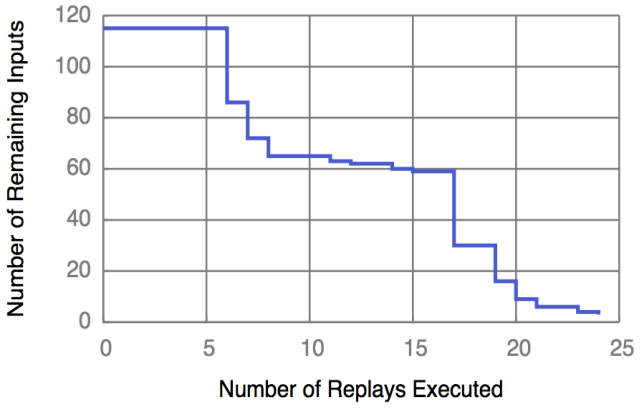
(a) Pyretic Loop.



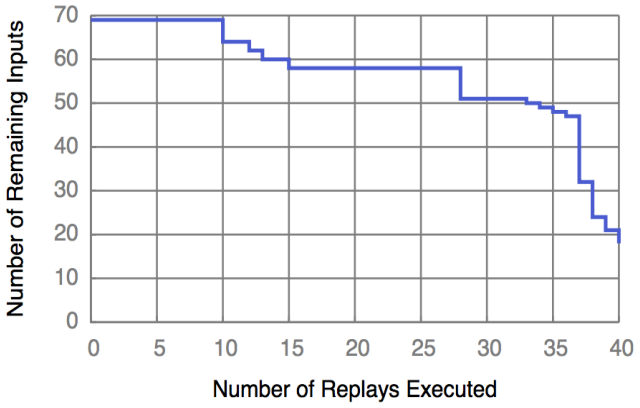
(b) POX Premature PacketIn.



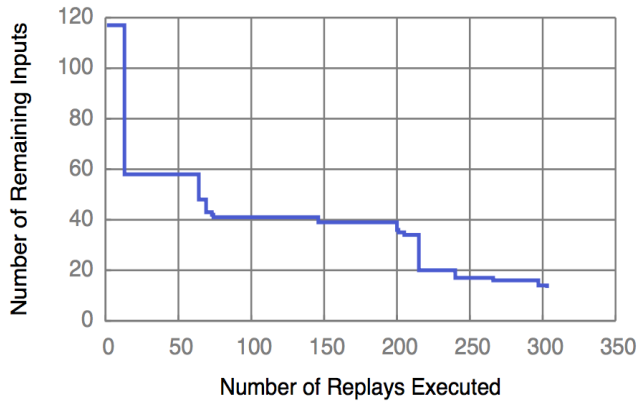
(c) POX In-Flight Blackhole.



(d) POX Migration Blackhole.

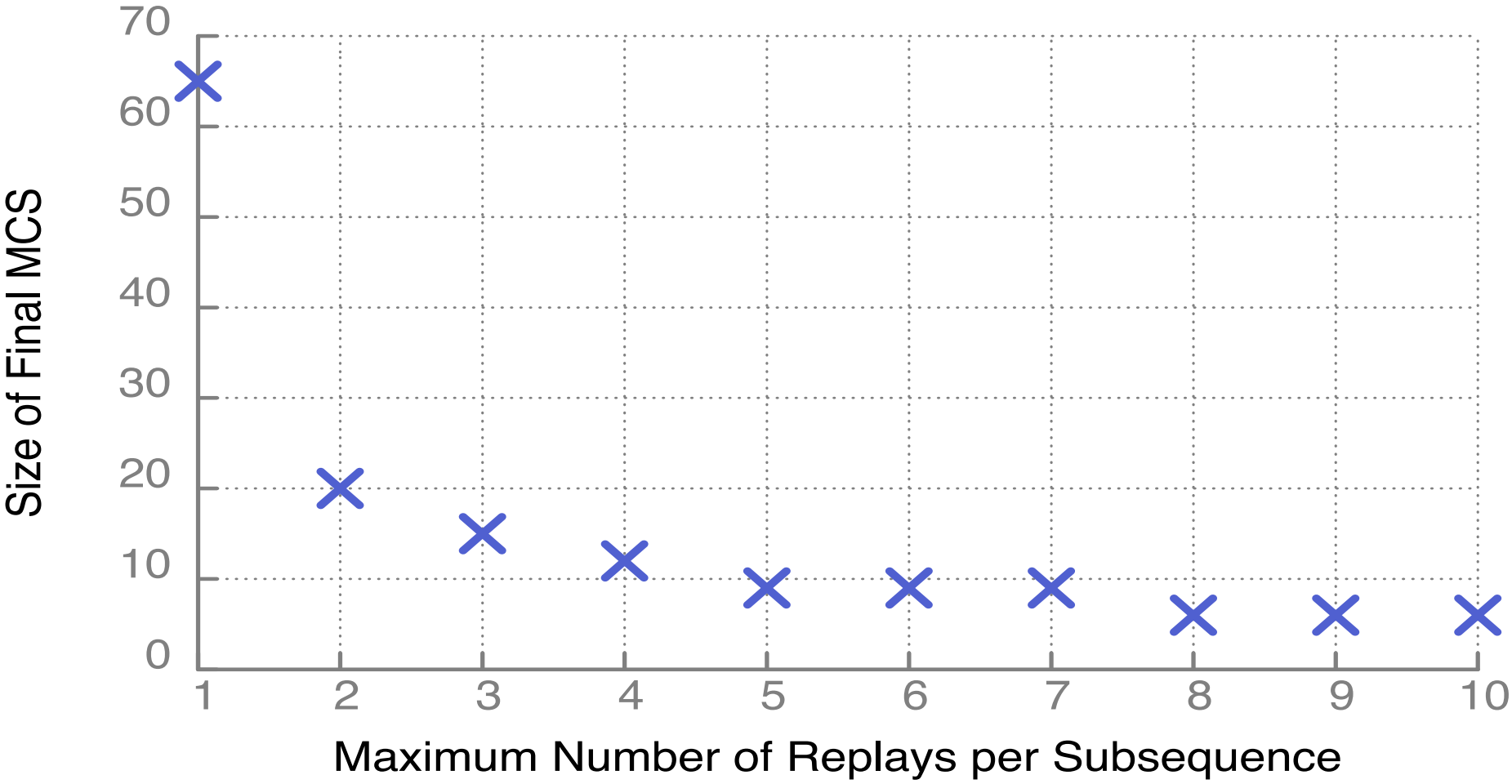


(e) NOX Discovery Loop.



(f) Floodlight Loop.

Coping with Non-Determinism

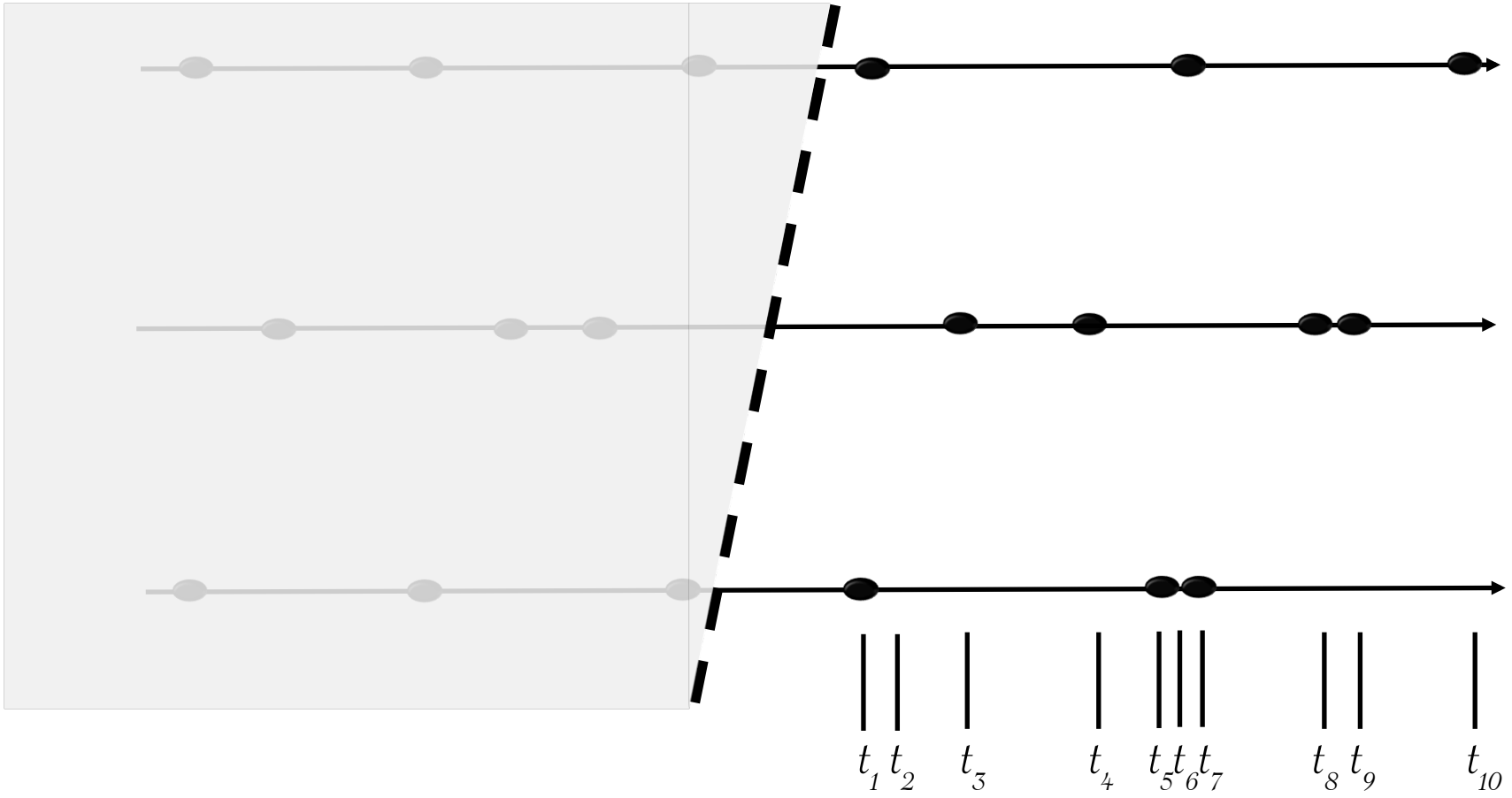


Replay Requirements

- Need to maintain original happens-before relation
- Includes internal events
  - Message Deliveries
  - State Transitions

Naïve Replay Approach

Schedule events according to wall-clock time



Complexity

Best Case	**Worst Case**
- Delta Debugging: Ω(log n) replays - Each replay: O(n) events - Total: Ω(nlog n)	- Delta Debugging: O(n) replays - Each replay: O(n) events - Total: $O(n^2)$

Assumptions of Delta Debugging

- *Monotonic:*

$$P \oplus C = \chi \Rightarrow P \oplus (C \cup C') \neq \checkmark$$

- *Unambiguous:*

$$P \oplus C = \chi \wedge P \oplus C' = \chi \Rightarrow P \oplus (C \cap C') \neq \checkmark$$

- *Consistent*

$$P \oplus C \neq ?$$

Local vs. Global Minimality

Definition 8 (Global minimum). *A set* $c \subseteq c_\chi$ *is called the* global minimum *of* c_χ *if:* $\forall c' \subseteq c_\chi \cdot \left(|c'| < |c| \Rightarrow test(c') \neq \chi\right)$ *holds.*

Definition 10 (n-minimal test case). *A test case* $c \subseteq c_\chi$ *is* n-minimal *if:* $\forall c' \subset c \cdot |c| - |c'| \leq n \Rightarrow \left(test(c') \neq \chi\right)$ *holds. Consequently,* c *is 1-minimal if* $\forall \delta_i \in c \cdot test\left(c - \{\delta_i\}\right) \neq \chi$ *holds.*

Forensic Analysis of Production Logs

- Logs need to capture causality: Lamport Clocks or accurate NTP
- Need clear mapping between input/internal events and simulated events
- Must remove redundantly logged events
- Might employ causally consistent snapshots to cope with length of logs

Instrumentation Complexity

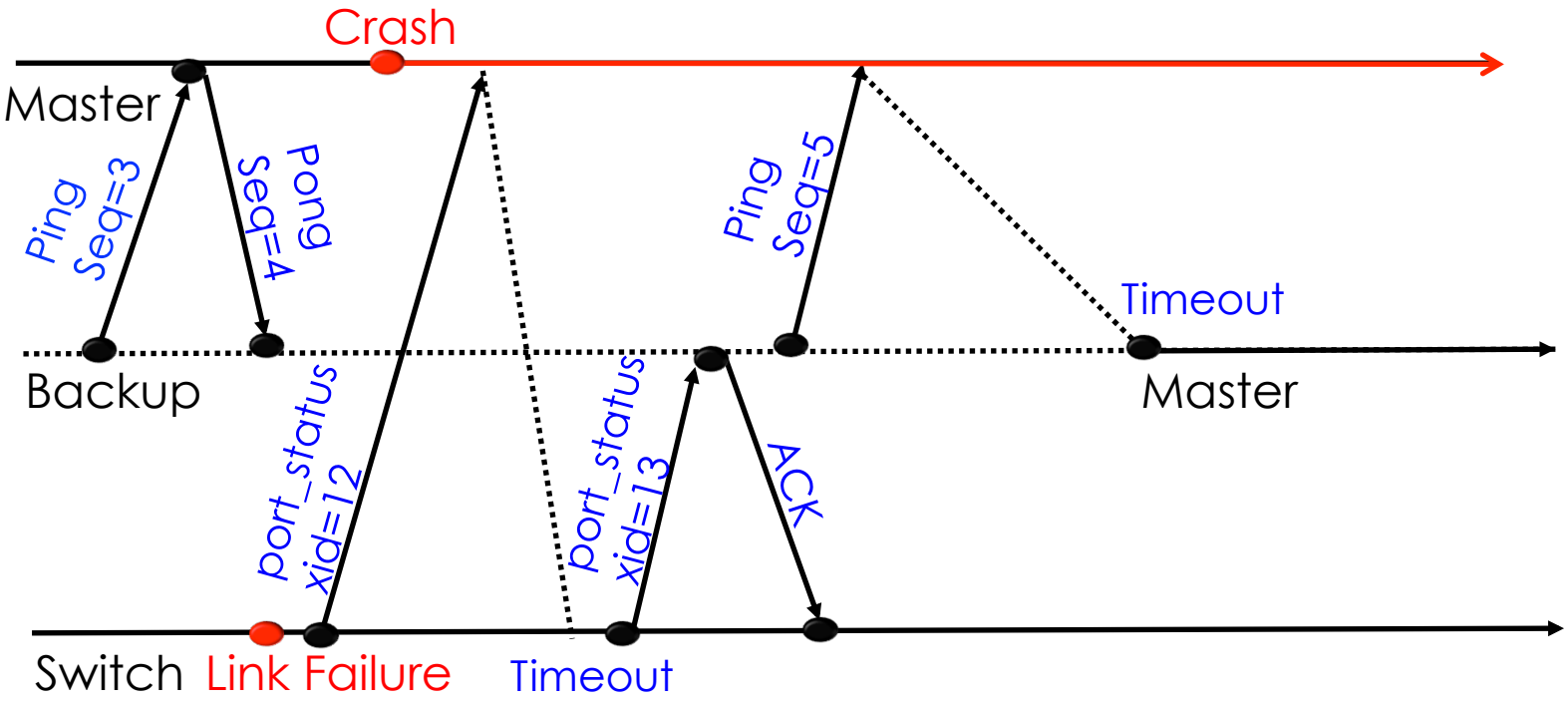
- Code to override gettimeofday(), interpose on logging statements, and multiplex sockets:
  - 415 LOC for POX (Python)
  - 722 LOC for Floodlight (Java)

Improvements

- Many improvements:
  - Parallelize delta debugging
  - Smarter delta debugging time splits
  - Apply program flow analysis to further prune
  - Compress time (override gettimeofday)

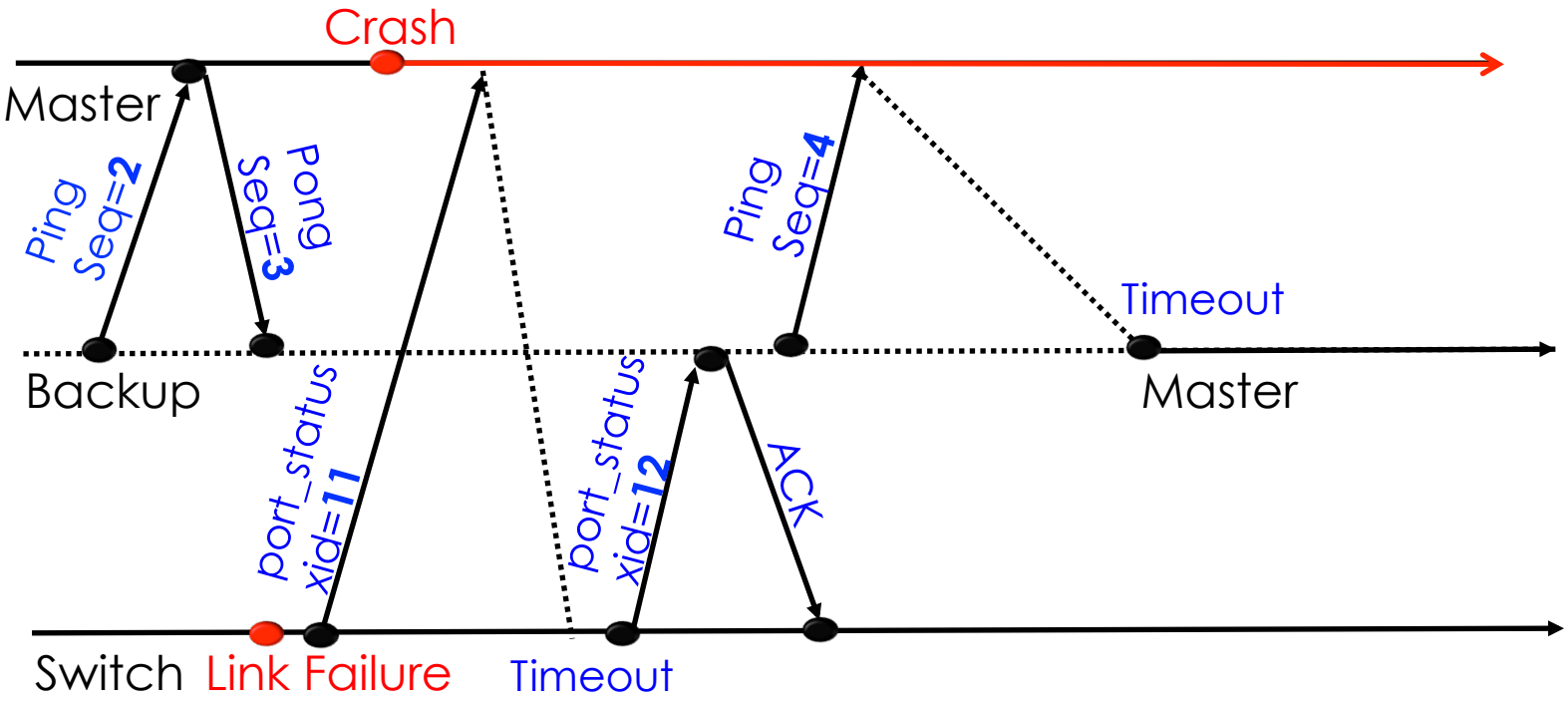
Divergence: Syntactic Changes

Prune Earlier Input..



Divergence: Syntactic Changes

Sequence Numbers Differ!



Solution: Equivalence Classes

Mask Over Extraneous Fields

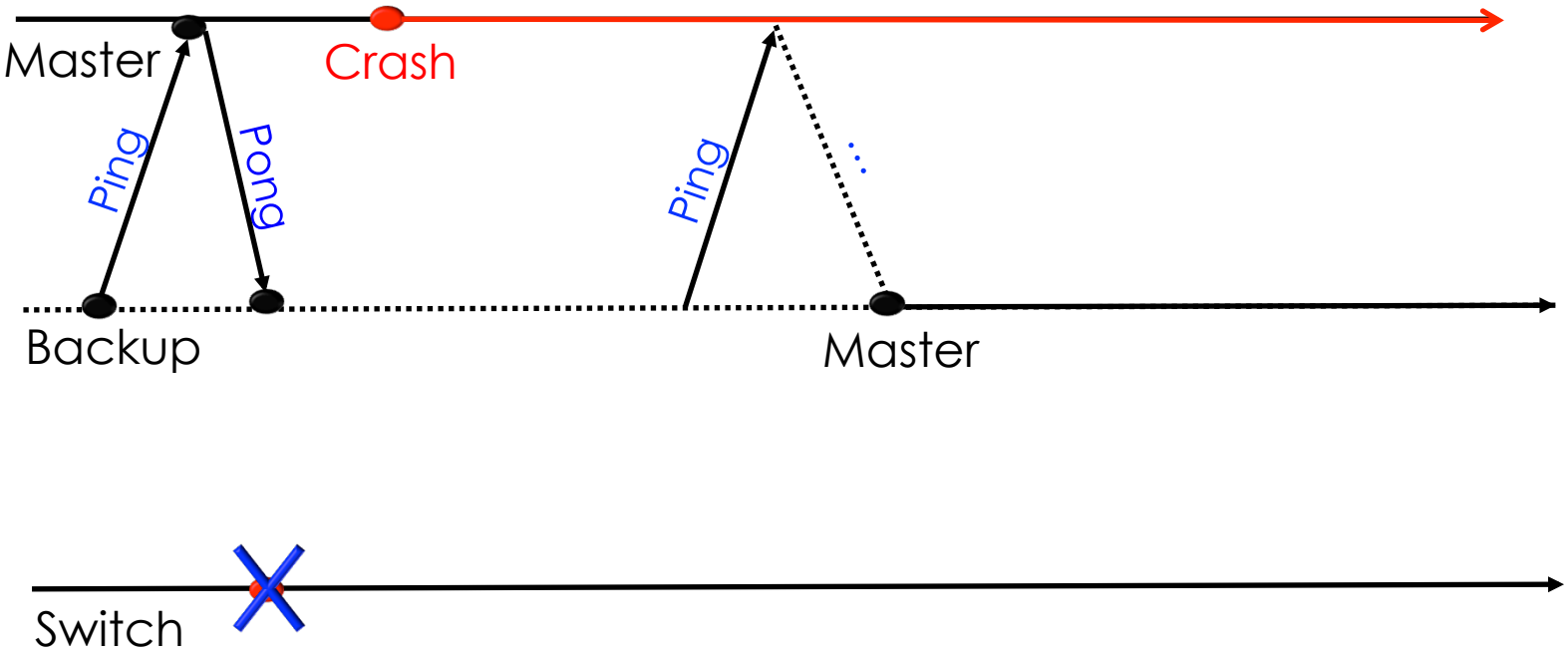
Internal message	**Masked values**
OpenFlow messages	xac id, cookie, buffer id, stats
packet_out/in payload	all values except src, dst, data
Log statements	varargs parameters to printf

Solution: Peek ahead

procedure PEEK(*input subsequence*)
 $inferred \leftarrow [\,]$
 for e_i *in subsequence*
  $checkpoint\ system$
  $inject\ e_i$
  $\Delta \leftarrow |e_{i+1}.time - e_i.time| + \epsilon$
  $record\ events\ for\ \Delta\ seconds$
  $matched \leftarrow original\ events\ \&\ recorded\ events$
  $inferred \leftarrow inferred + [e_i] + matched$
  $restore\ checkpoint$
 return $inferred$

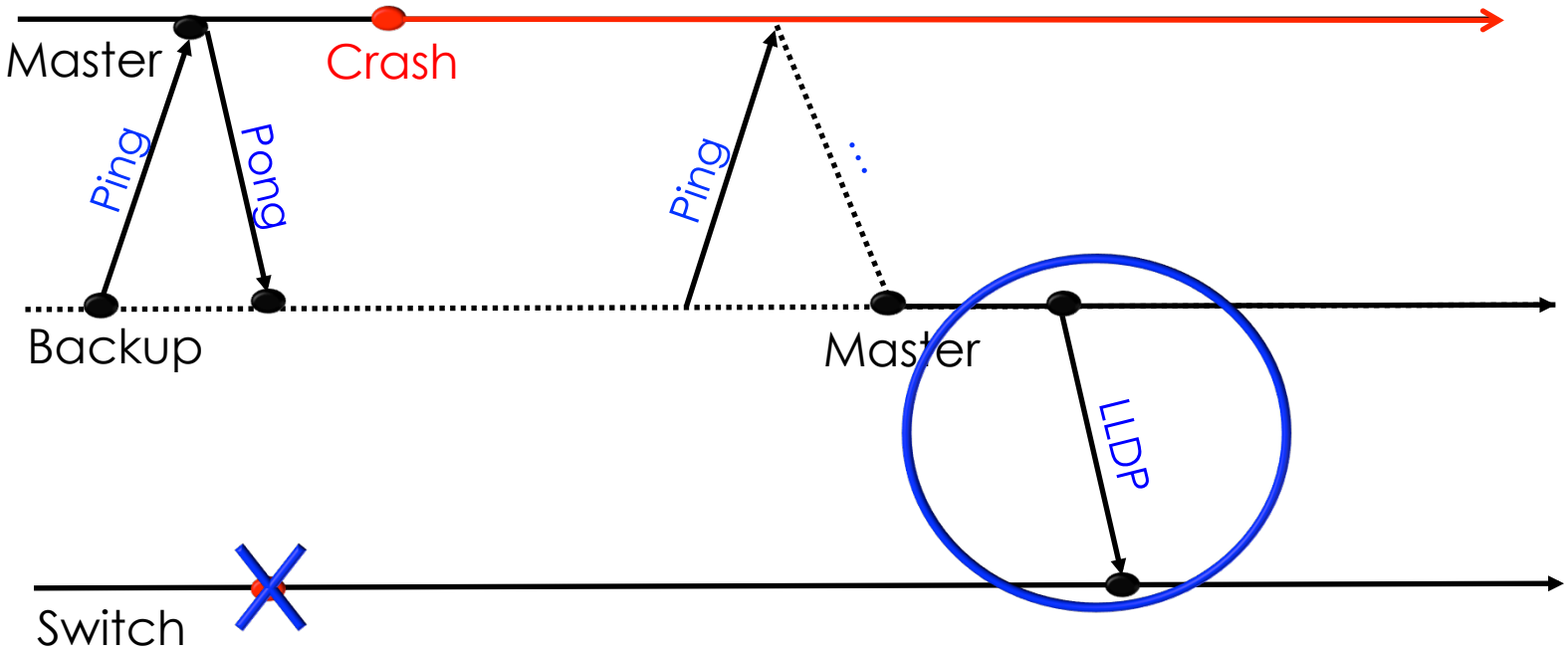
Divergence: Unexpected Events

Prune Input..



Divergence: Unexpected Events

Unexpected Events Appear



Solution: Emperical Heuristic

Theory:

- Divergent paths → Exponential possibilities

Practice:

- Allow unexpected events through