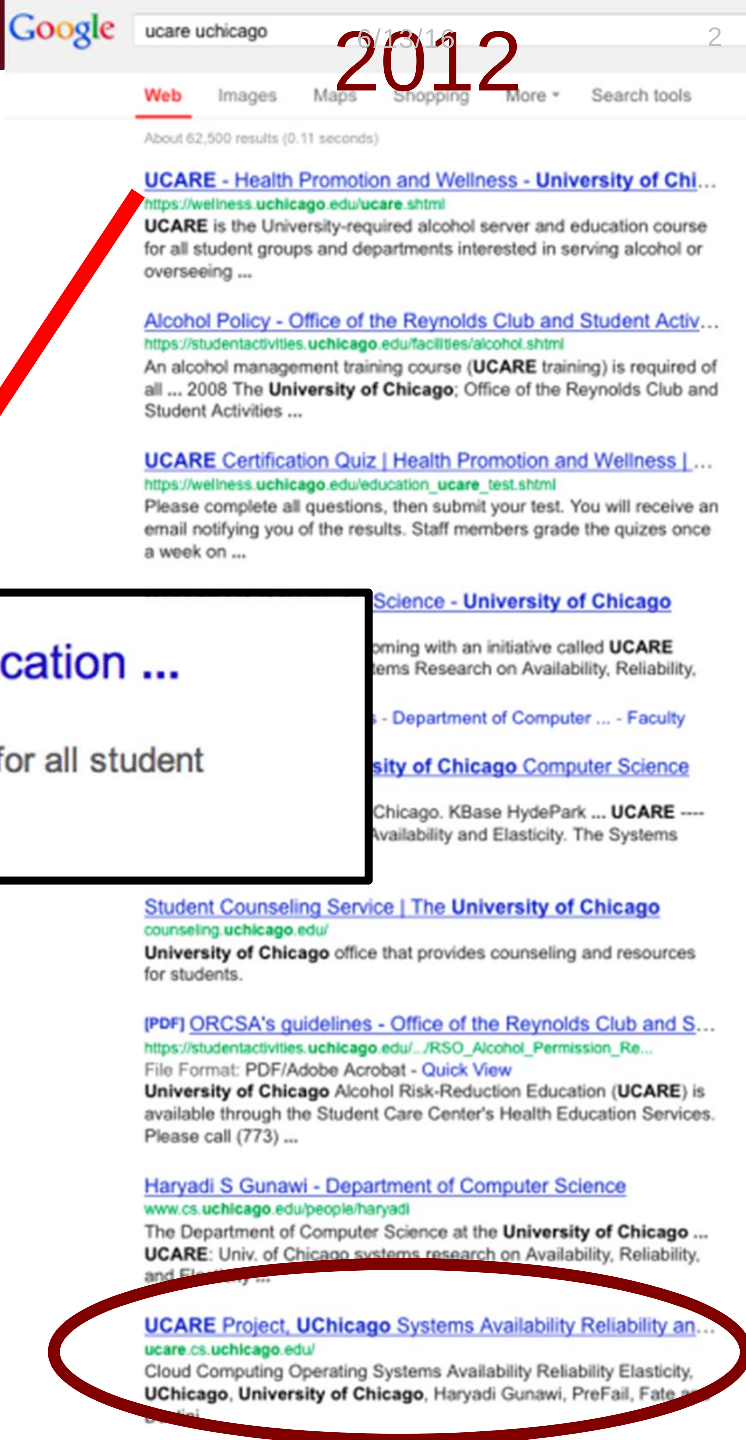
What New Bugs Live in the Cloud? (and how to exterminate them)

Haryadi Gunawi

THE UNIVERSITY OF CHICAGO

UCARE Lab

U of **C** systems research on **A**vailability, **R**eliability & **E**fficiency



2012

ucare uchicago

About 62,500 results (0.11 seconds)

UCARE - Health Promotion and Wellness - **University of Chi**...
https://wellness.**uchicago**.edu/**ucare**.shtml
UCARE is the University-required alcohol server and education course for all student groups and departments interested in serving alcohol or overseeing ...

Alcohol Policy - Office of the Reynolds Club and Student Activ...
https://studentactivities.**uchicago**.edu/facilities/alcohol.shtml
An alcohol management training course (**UCARE** training) is required of all ... 2008 The **University of Chicago**; Office of the Reynolds Club and Student Activities ...

UCARE Certification Quiz | Health Promotion and Wellness | ...
https://wellness.**uchicago**.edu/education_**ucare**_test.shtml
Please complete all questions, then submit your test. You will receive an email notifying you of the results. Staff members grade the quizes once a week on ...

Science - **University of Chicago**
oming with an initiative called **UCARE**
tems Research on Availability, Reliability,

s - Department of Computer ... - Faculty

sity of Chicago Computer Science
Chicago. KBase HydePark ... **UCARE** ----
Availability and Elasticity. The Systems

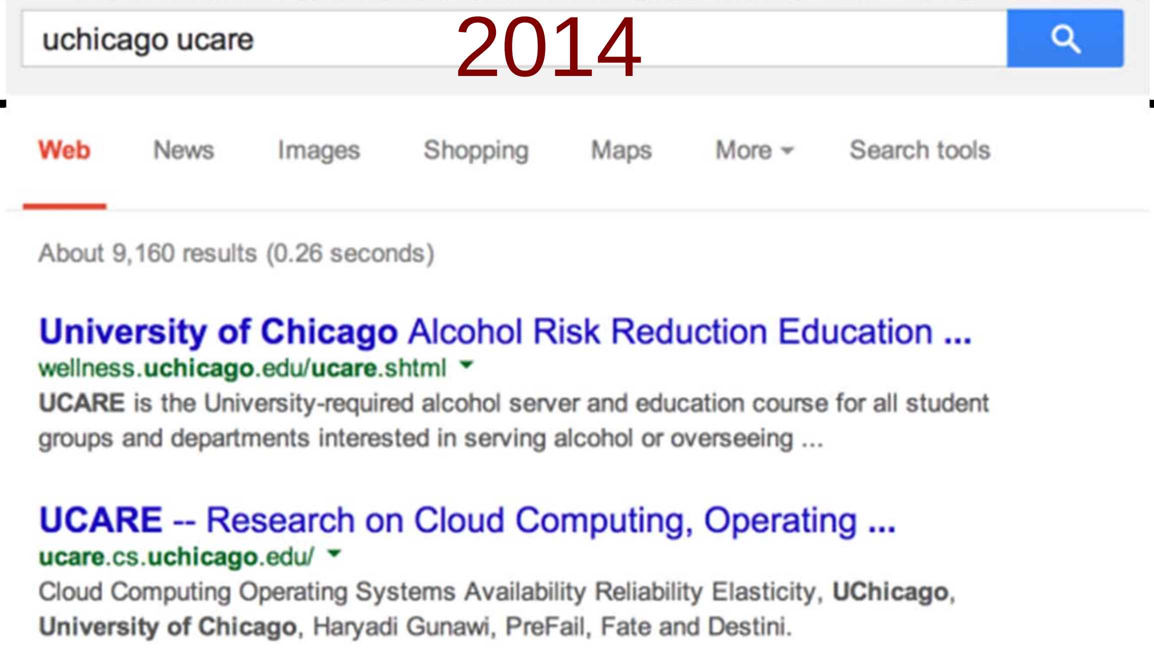
Student Counseling Service | The **University of Chicago**
counseling.**uchicago**.edu/
University of Chicago office that provides counseling and resources for students.

[PDF] ORCSA's guidelines - Office of the Reynolds Club and S...
https://studentactivities.**uchicago**.edu/.../RSO_Alcohol_Permission_Re...
File Format: PDF/Adobe Acrobat - Quick View
University of Chicago Alcohol Risk-Reduction Education (**UCARE**) is available through the Student Care Center's Health Education Services. Please call (773) ...

Haryadi S Gunawi - Department of Computer Science
www.cs.**uchicago**.edu/people/haryadi
The Department of Computer Science at the **University of Chicago** ... **UCARE**: Univ. of Chicago systems research on Availability, Reliability, and Elasticity ...

UCARE Project, **UChicago** Systems Availability Reliability an...
ucare.cs.**uchicago**.edu/
Cloud Computing Operating Systems Availability Reliability Elasticity, **UChicago**, **University of Chicago**, Haryadi Gunawi, PreFail, Fate and Destini.

University of Chicago Alcohol Risk Reduction Education **...**
wellness.**uchicago**.edu/**ucare**.shtml ▾
UCARE is the University-required alcohol server and education course for all student



2014

uchicago ucare

About 9,160 results (0.26 seconds)

University of Chicago Alcohol Risk Reduction Education **...**
wellness.**uchicago**.edu/**ucare**.shtml ▾
UCARE is the University-required alcohol server and education course for all student groups and departments interested in serving alcohol or overseeing ...

UCARE -- Research on Cloud Computing, Operating **...**
ucare.cs.**uchicago**.edu/ ▾
Cloud Computing Operating Systems Availability Reliability Elasticity, **UChicago**, **University of Chicago**, Haryadi Gunawi, PreFail, Fate and Destini.

What new bugs live in the cloud?

Datacenter distributed systems

# of Bug Reports	Jan 2014	Jan 2016
Hadoop+MR+Yarn	17454	23811
HDFS	5710	9605
HBase	10263	15062
Cassandra	6535	10960
ZooKeeper	1854	2350

We studied 3000+ issues

"New" classes of bugs

- Distributed concurrency bugs
  - + Timings of multiple failures

TaxDC [ASPLOS '16]
SAMC [OSDI '14]
FATE & DESTINI [NSDI '11]

- Non-deterministic performance bugs

The Tail at Store [FAST '16]
SPV [HotCloud '15]
Limpware [SoCC '13]
Tiny Tail [In Subm.]
Path-Based Spec. Exec. [In Subm.]

- Scalability bugs

SCk [In Subm.]

- Other outage-causing bugs:
  - SPOF/cascading bugs
  - Cross-layer upgrade bugs

Cloud Bug Study [SoCC '14]
Cloud Outage Study [In Subm.]

“New” classes of bugs

- ❑ Distributed concurrency (DC) bugs

TaxDC [ASPLOS ’16]
SAMC [OSDI ’14]

- ❑ Non-deterministic performance bugs
- ❑ Scalability bugs

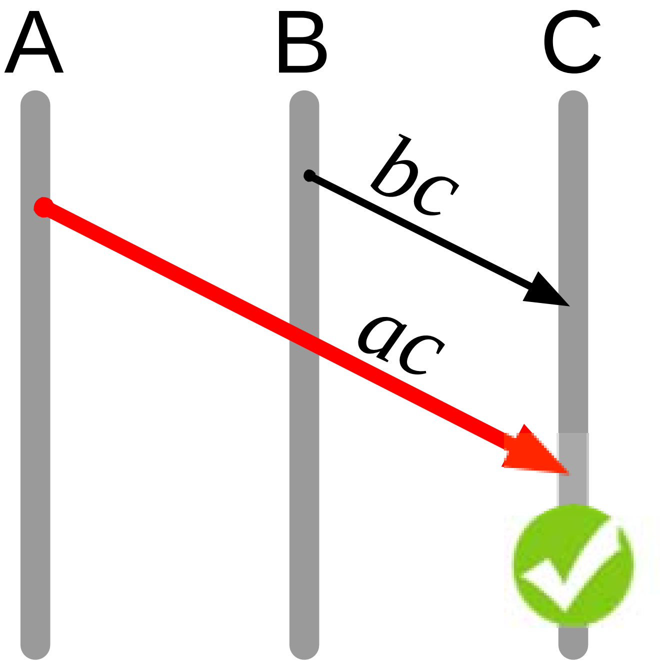
Distributed concurrency (DC) bug

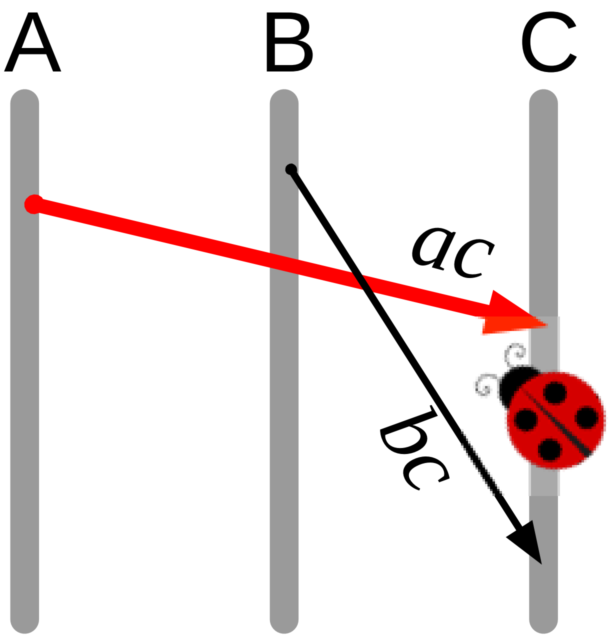
- Caused by non-deterministic timing of concurrent events involving more than one node
- Events: Messages, crashes, reboots, timeouts, local computations

6% of the bugs
in our study

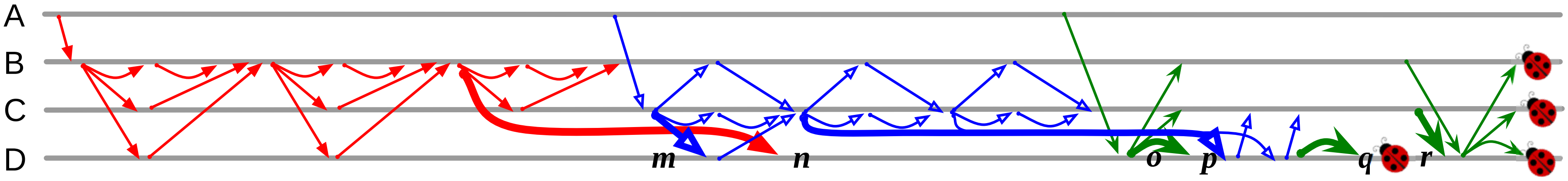
Data loss, downtimes,
inconsistent replicas,
hanging jobs, etc.

DC bug: a simple view





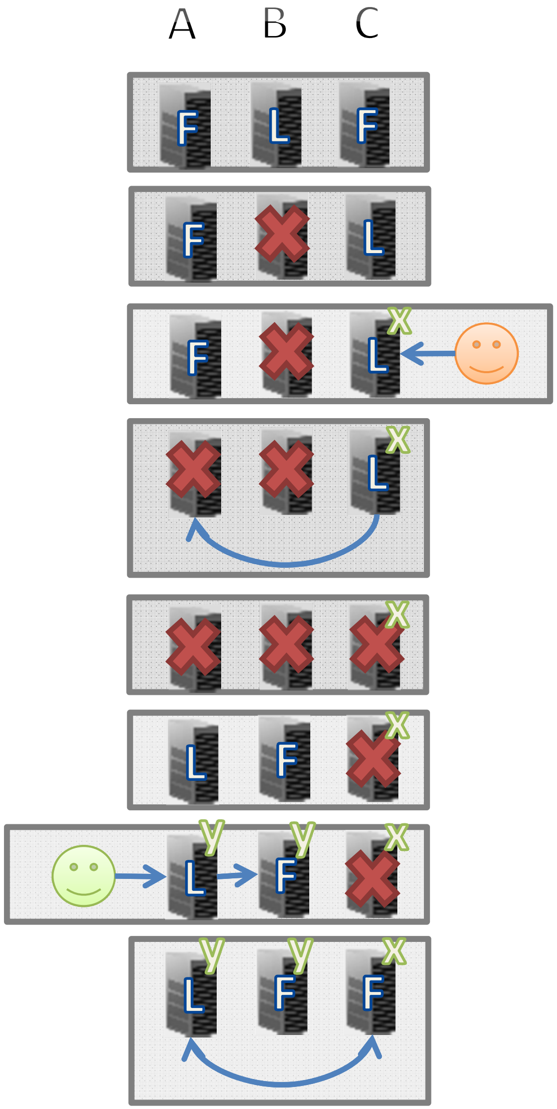
DC bug: a real view



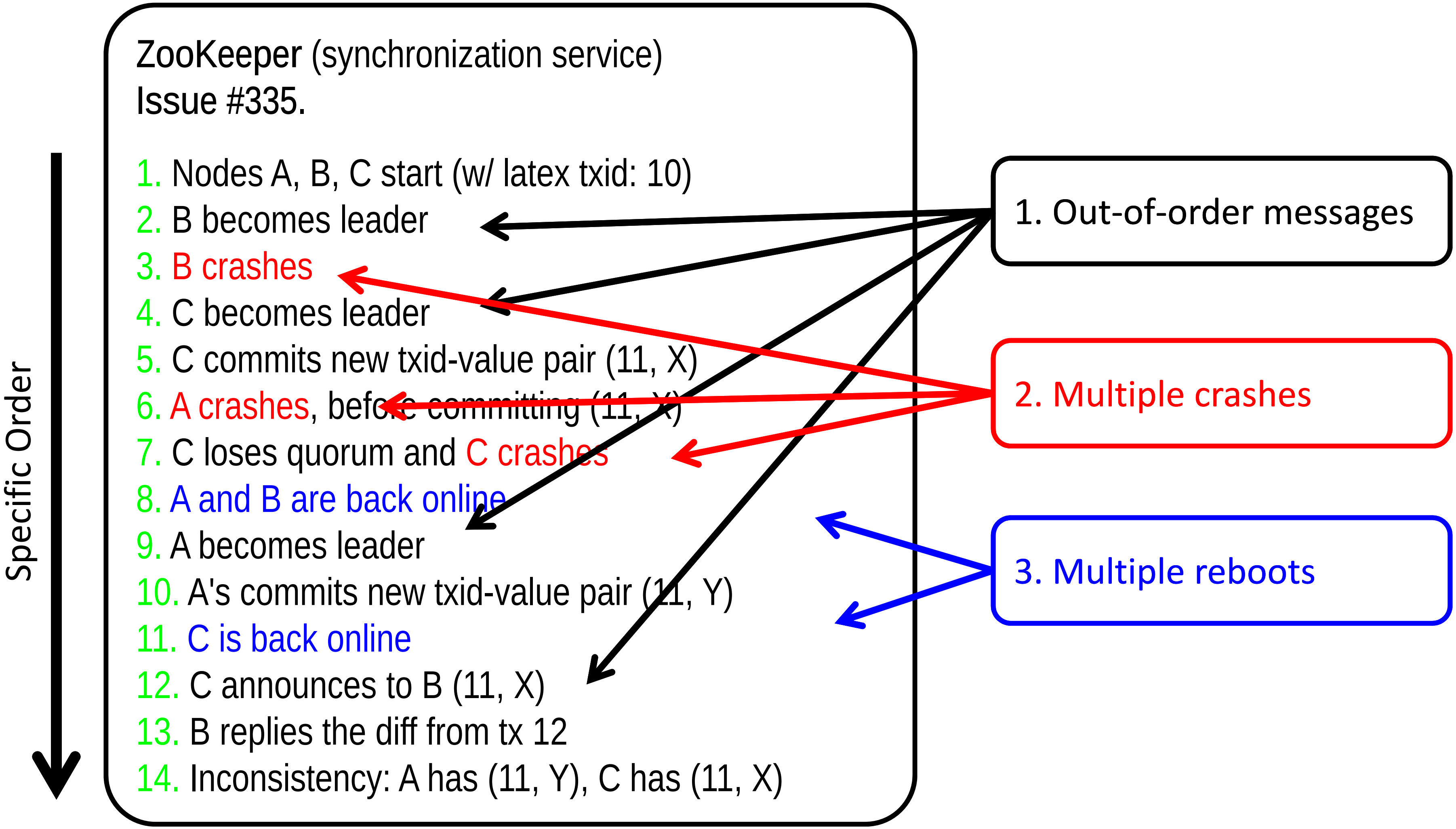
- Cassandra **Paxos** Bug (# 6023)
- **3 concurrent updates**
  - Red, blue, green
- **3 msg-msg races** must happen
  - m = prepare message for ballot 2, **BEFORE**
  - n = commit message for ballot 1
  - o = prepare message for ballot 3, **BEFORE**
  - p = propose message for ballot 2
  - q = promise message for ballot 3, **BEFORE**
  - r = promise message for ballot 3
  - (24 hours to understand)

ZooKeeper (synchronization service)
Issue #335.

1. Nodes A, B, C start (w/ latex txid: 10)
2. B becomes leader
3. B crashes
4. C becomes leader
5. C commits new txid-value pair (11, X)
6. A crashes, before committing (11, X)
7. C loses quorum and C crashes
8. A and B are back online
9. A becomes leader
10. A's commits new txid-value pair (11, Y)
11. C is back online
12. C announces to B (11, X)
13. B replies the diff from tx 12
14. Inconsistency: A has (11, Y), C has (11, X)



PERMANENTLY INCONSISTENT REPLICA



ZooKeeper (synchronization service)
Issue #335.

1. Nodes A, B, C start (w/ latex txid: 10)
2. B becomes leader
3. B crashes
4. C becomes leader
5. C commits new txid-value pair (11, X)
6. A crashes, before committing (11, X)
7. C loses quorum and C crashes
8. A and B are back online
9. A becomes leader
10. A's commits new txid-value pair (11, Y)
11. C is back online
12. C announces to B (11, X)
13. B replies the diff from tx 12
14. Inconsistency: A has (11, Y), C has (11, X)

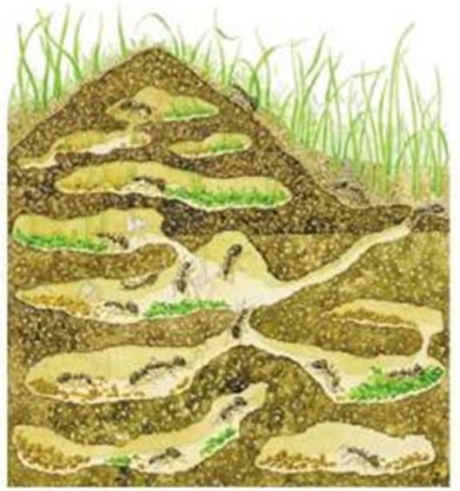
1. Out-of-order messages

2. Multiple crashes

3. Multiple reboots

① ② ③ **HAPPEN IN ANY ORDER**

How can we catch **deep** concurrency bugs in distributed systems?



© 2006 Encyclopædia Britannica, Inc.

Distributed system model checker (dmck)

- Re-ordering all non-deterministic events
  - **Paths:** abcd, abdc, acbd, acdb, …
  - Find buggy paths/interleavings

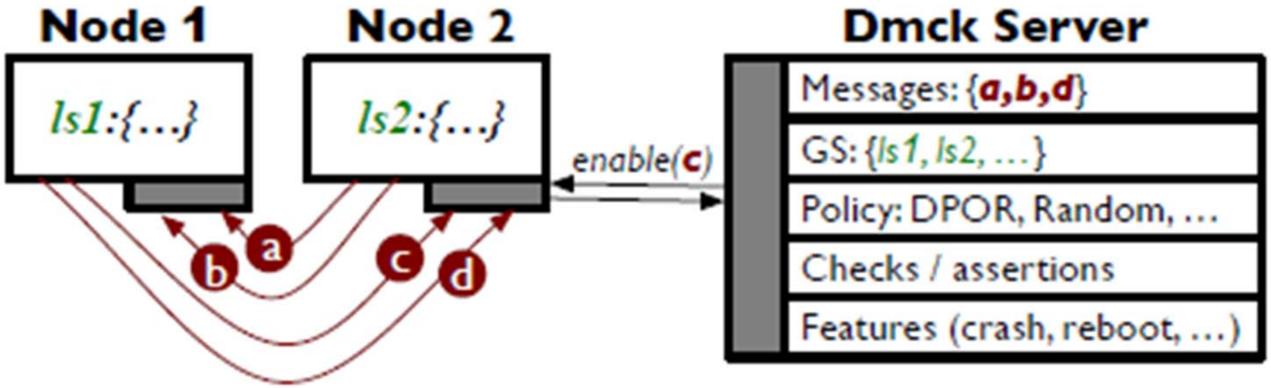
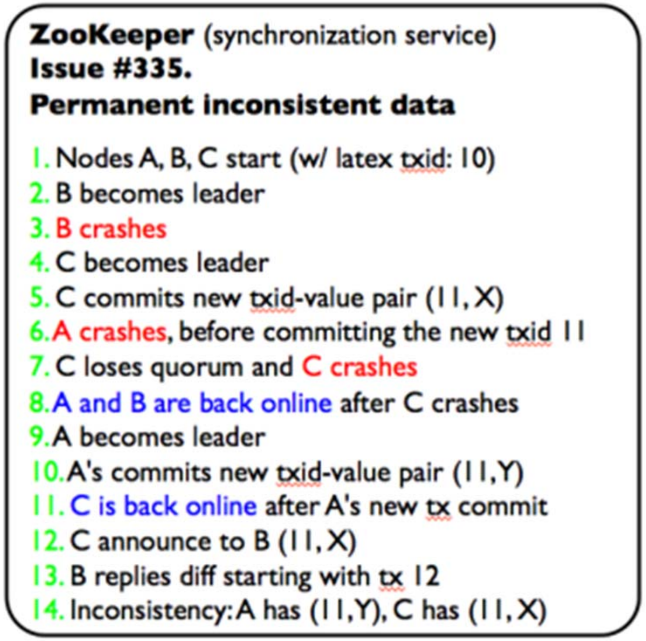


Figure 1: **DMCK.** *The figure illustrates a typical framework of a distributed system model checker (dmck).*

Event re-orderings by dmck



ZooKeeper (synchronization service)
Issue #335.
Permanent inconsistent data

1. Nodes A, B, C start (w/ latex txid: 10)
2. B becomes leader
3. B crashes
4. C becomes leader
5. C commits new txid-value pair (11, X)
6. A crashes, before committing the new txid 11
7. C loses quorum and C crashes
8. A and B are back online after C crashes
9. A becomes leader
10. A's commits new txid-value pair (11, Y)
11. C is back online after A's new tx commit
12. C announce to B (11, X)
13. B replies diff starting with tx 12
14. Inconsistency: A has (11, Y), C has (11, X)

3	2	6	1	2
4	7	9	2	1
1	1	3	3	3
5	4	4	4	4
2	5	5	5	5
6	6	1	6	7
7	3	7	7	6
8	8	8	8	8
9	11	2	9	9
12	10	10	10	10
11	9	11	11	11
10	12	13	12	14
14	14	12	13	12
13	13	14	14	13
✓	✓	✓	🐞	✓

• • •

SAMC:
Semantic-Aware Model Checking
for Fast Discovery of Deep DC Bugs

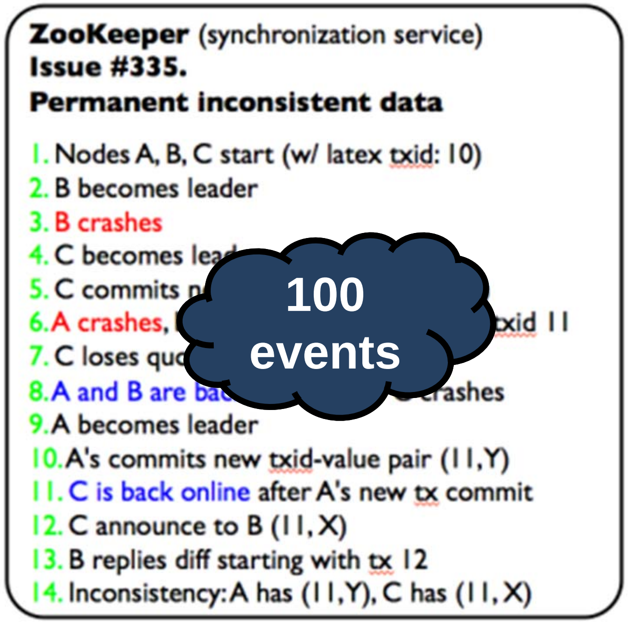
with Tanakorn Leesatapornwongsa,

Mingzhe Hao, Pallavi Joshi, and Jeffrey F. Lukman

[OSDI ’14]

What’s Wrong with Existing Model Checkers?

- Last 7 years
  - MaceMC [NSDI ’07], Modist [NSDI ’09], dBug [SSV ’10], Demeter [SOSP ’13], etc.
- **BUT**
  - – Too many events to permute
  - – Must add crashes and reboots
    - State-space explosion!
    - (skipped in existing checkers)
      - – **Cannot find deep bugs!**



ZooKeeper (synchronization service)
Issue #335.
Permanent inconsistent data

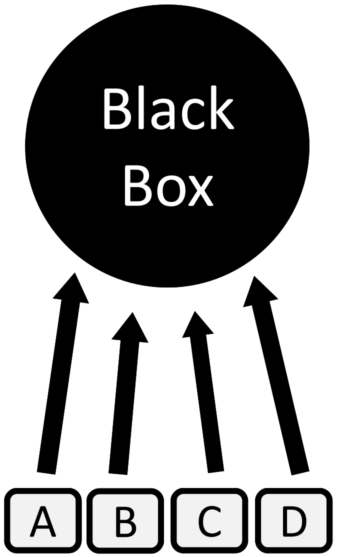
1. Nodes A, B, C start (w/ latex txid: 10)
2. B becomes leader
3. B crashes
4. C becomes lea
5. C commits n
6. A crashes, txid 11
7. C loses quo
8. A and B are bac crashes
9. A becomes leader
10. A's commits new txid-value pair (11,Y)
11. C is back online after A's new tx commit
12. C announce to B (11,X)
13. B replies diff starting with tx 12
14. Inconsistency: A has (11,Y), C has (11,X)

100 events

How can we catch deep bugs **REALLY FAST**?

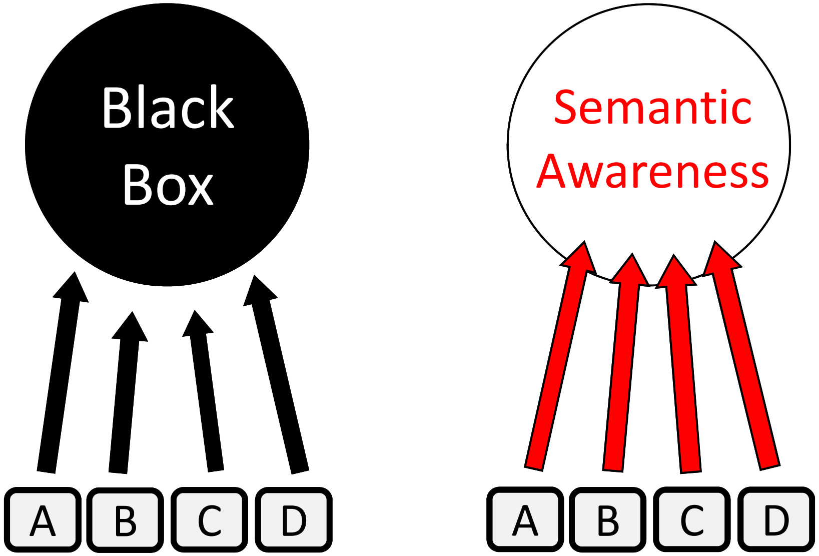
- Why are existing checkers slow?

- They treat target system as a black box
  - Must re-order everything

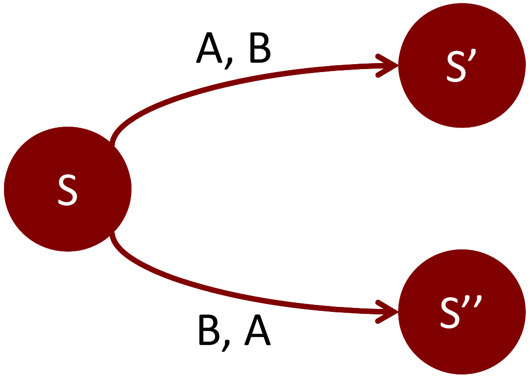


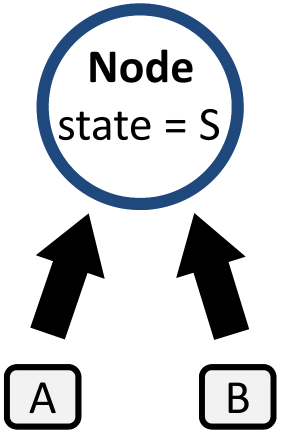
Black Box Model Checker
ABCD
ABDC
ACBD
ACDB
ADBC
. . .
(24 total)

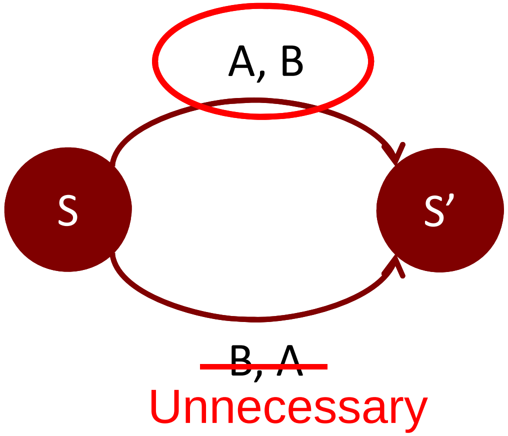
- How can we make model checkers fast?
  - – Exploit **semantic knowledge**
    - *E.g. knowledge of how messages are processed*
  - – Reduce unnecessary re-orderings



Dependency vs. Independency







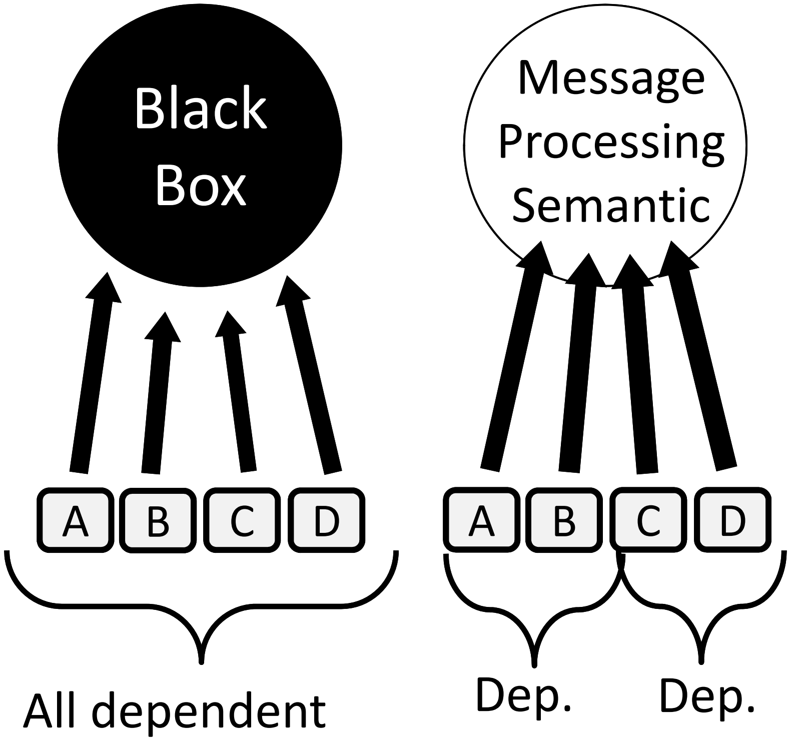
A, B = Dependent

A, B = Independent

Independent = No need to reorder

Black Box vs. SAMC

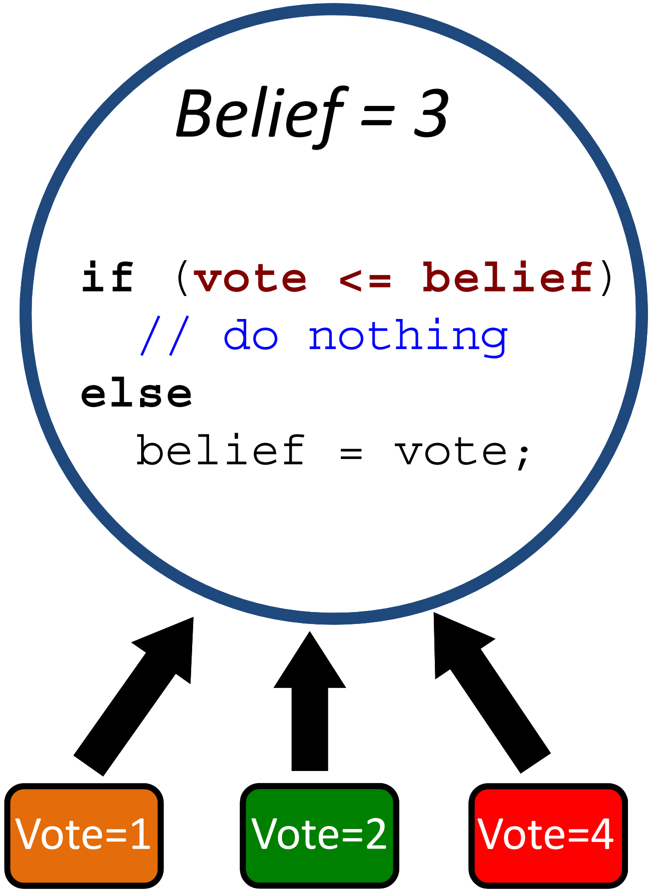
Black Box Model Checker
ABCD
ABDC
ACBD
ACDB
ADBC
ADCB
BACD
BADC
BCAD
BCDA
BDAC
. . .



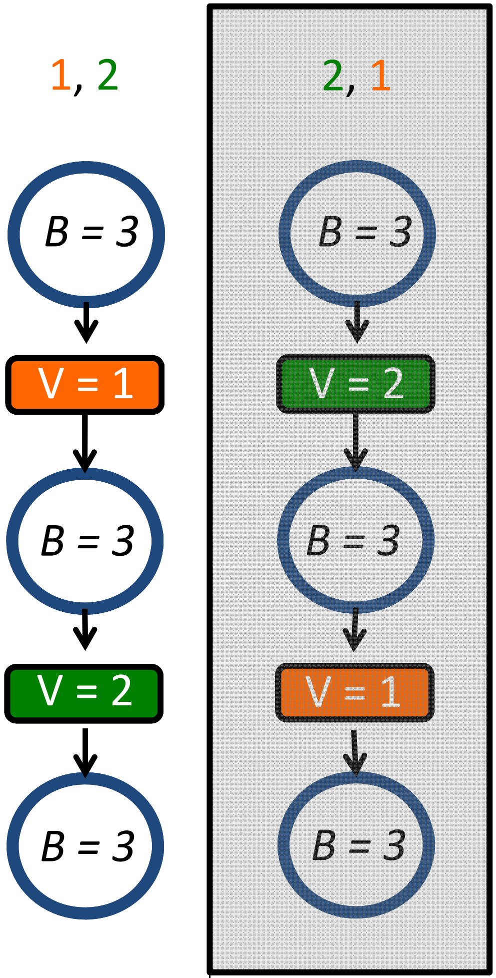
Unnecessary Re-orderings (lead to the same state)

SAMC with message processing semantic
ABCD
ABDC
~~ACBD~~
~~ACDB~~
~~ADBC~~
~~ADCB~~
BACD
BADC
~~BCAD~~
~~BCDA~~
. . .

Message Processing Semantic in a Leader Election



```
if (vote <= belief)
  // do nothing
else
  belief = vote;
```



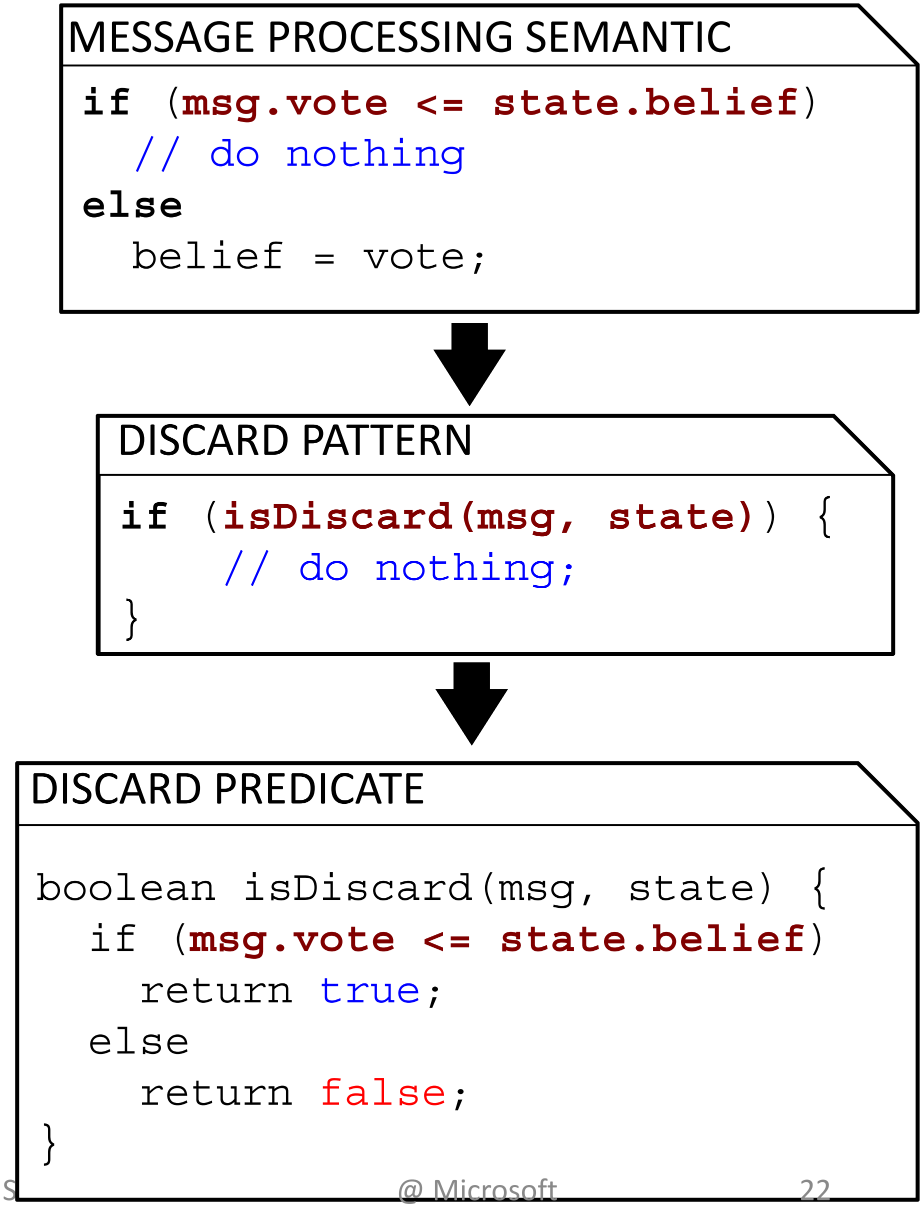
1 2 4
2 1 4

4 1 2
4 2 1

1 4 2
2 4 1

Unnecessary

- **Discard pattern**



MESSAGE PROCESSING SEMANTIC

```
if (msg.vote <= state.belief)
 // do nothing
else
 belief = vote;
```

DISCARD PATTERN

```
if (isDiscard(msg, state)) {
  // do nothing;
}
```

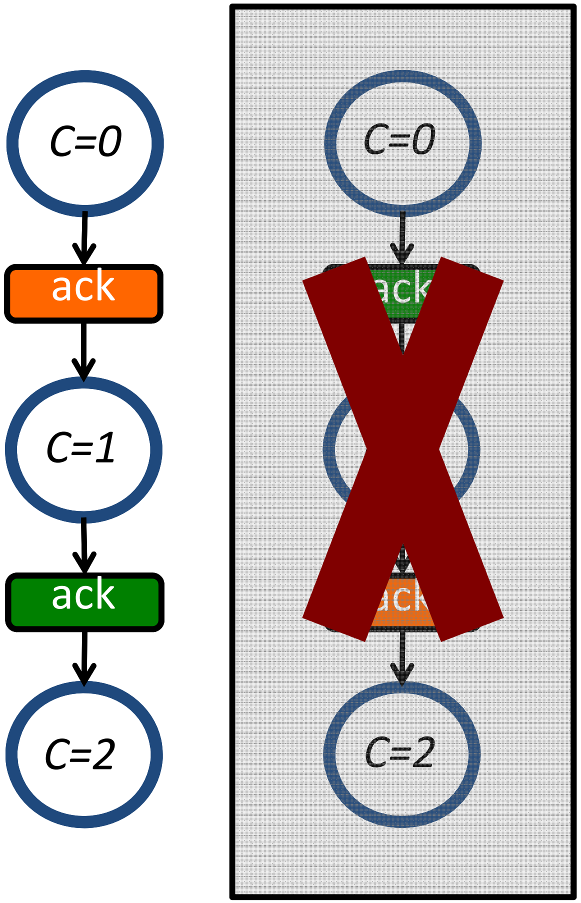
DISCARD PREDICATE

```
boolean isDiscard(msg, state) {
 if (msg.vote <= state.belief)
  return true;
 else
  return false;
}
```

- Discard pattern
- **Increment pattern**

```
if (msg.type == ack) {
  node.ackCount++;
}
```

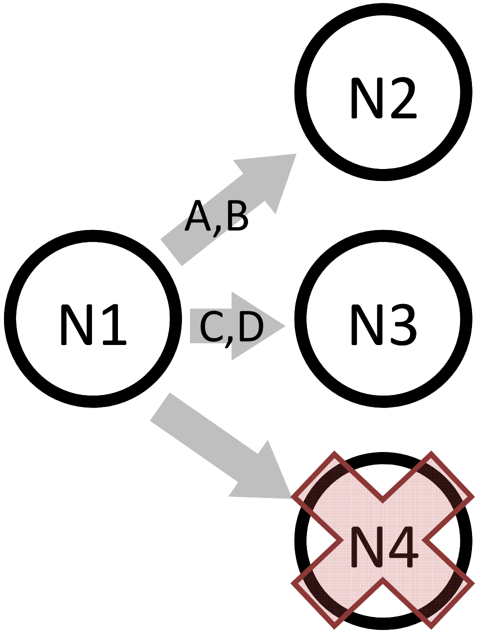
```
boolean isIncrement(msg, ls) {
  if (msg.type == ack)
    return true;
  else
    return false;
}
```



- **Constant pattern**

Local-Message Independence (LMI)

SAMC with Crashes

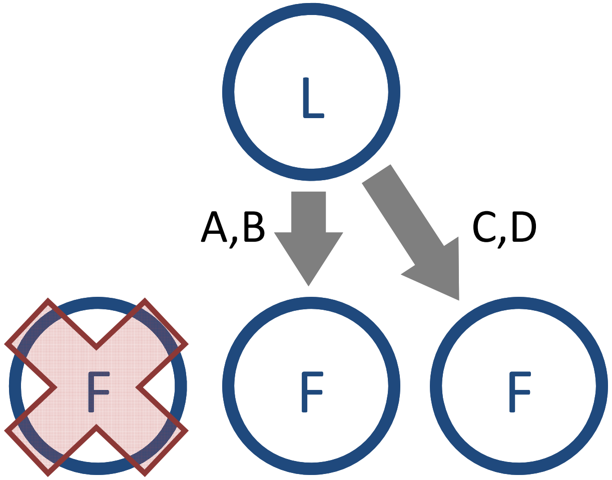


Black Box Model checker	SAMC with crash recovery semantic
ABCDX	ABCDX
ABCXD	~~ABCXD~~
ABXCD	~~ABXCD~~
AXBCD	~~AXBCD~~
XABCD	~~XABCD~~
ABDCX	~~ABDCX~~
ABDXC	~~ABDXC~~
. . .	. . .

Unnecessary Re-orderings

Crash-Msg Independence

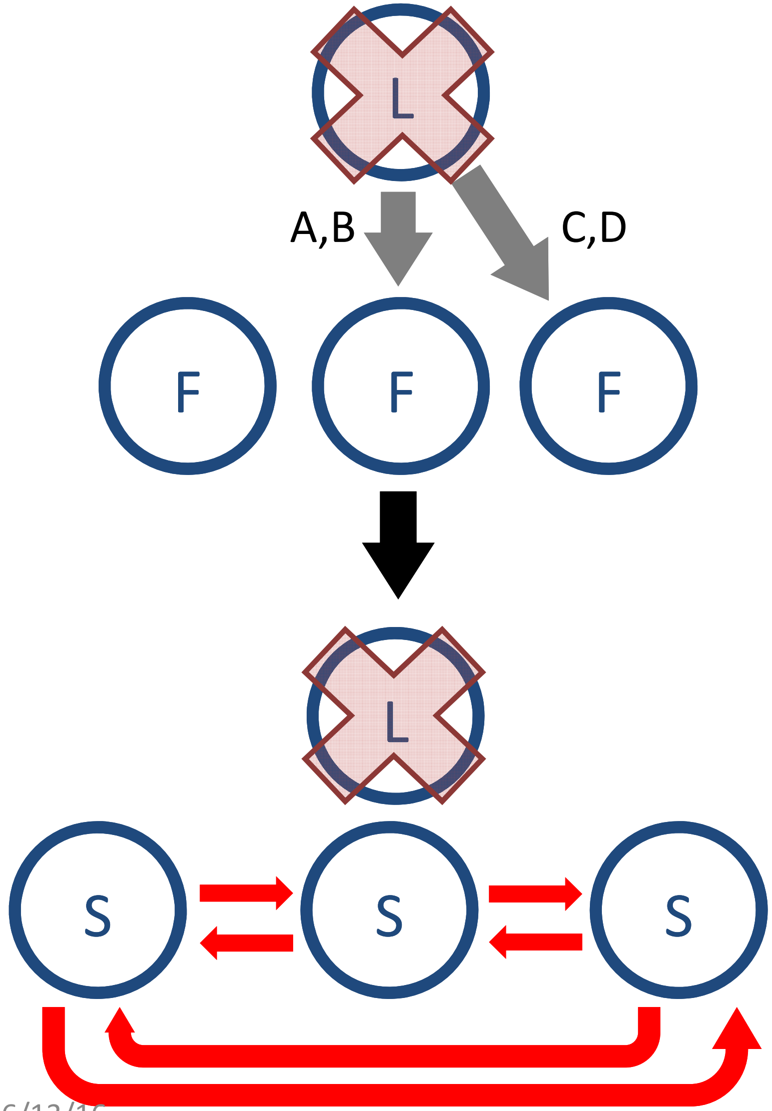
Black Box
ABCDX
~~ABCXD~~
~~ABXCD~~
~~AXBCD~~
~~XABCD~~
~~ABDCX~~
…



```
void handleCrash() {
  if (X == follower &&
isQuorum())
    followerCount--;
    // No new messages!!
}
```

Crash a **follower**
→
Local Impact
(no new messages &
only state changes in leader L)

Crash-Msg Independence

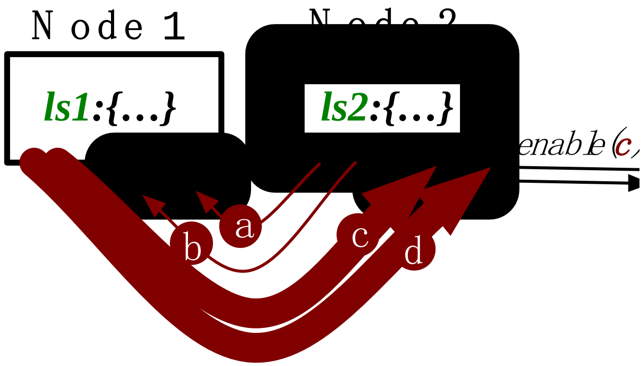


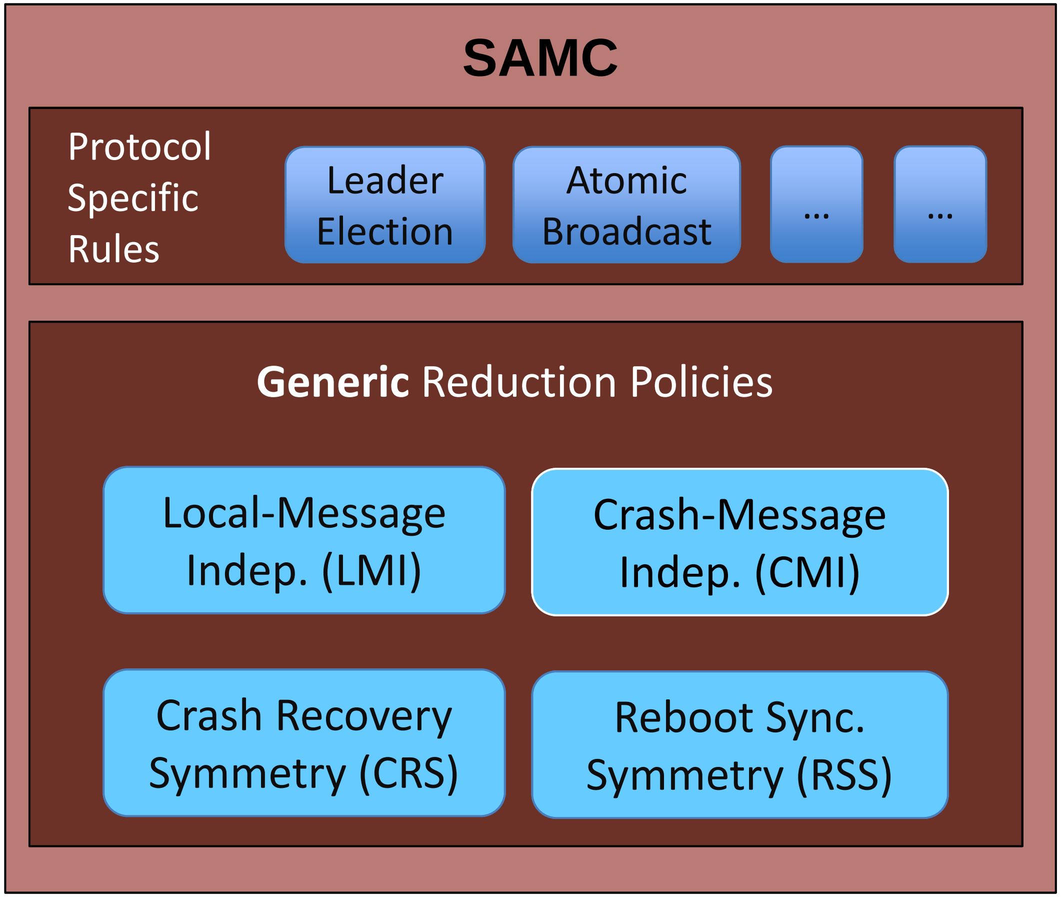
```
void handleCrash() {
  if (X == leader || !isQuorum())
    electLeader()
    // New messages created
}
```

Crash the **leader**
→ **Global Impact**
(cannot prune
re-orderings)

Black Box
ABCDX
ABCXD
ABXCD
AXBCD
XABCD
ABDCX
…

SAMC Architecture





SAMC

Protocol Specific Rules

Leader Election

Atomic Broadcast

…

…

Generic Reduction Policies

Local-Message Indep. (LMI)

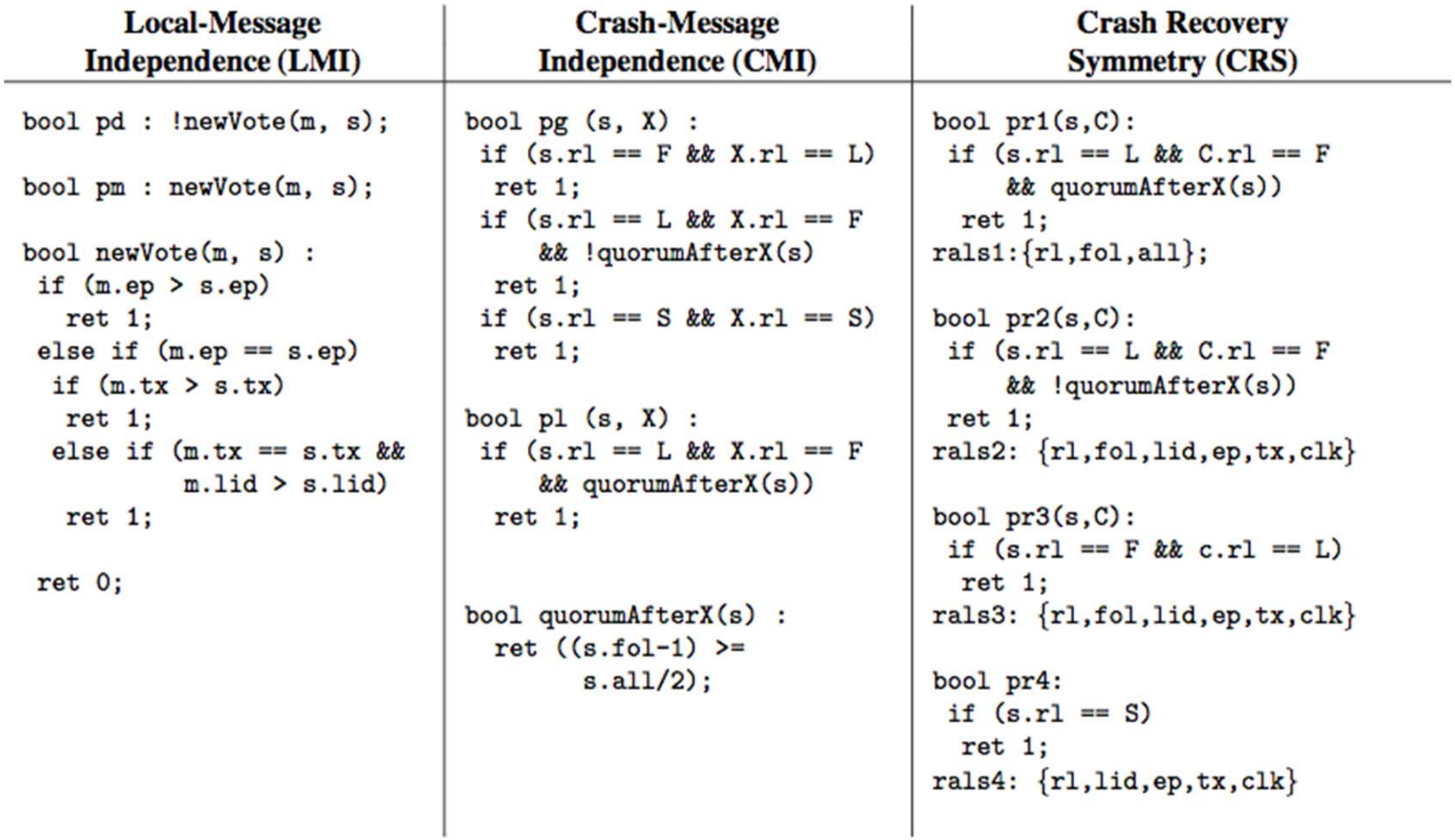
Crash-Message Indep. (CMI)

Crash Recovery Symmetry (CRS)

Reboot Sync. Symmetry (RSS)

Protocol-specific predicates (extra)

(e.g. ZooKeeper Leader Election)



Local-Message Independence (LMI)

```
bool pd : !newVote(m, s);

bool pm : newVote(m, s);

bool newVote(m, s) :
 if (m.ep > s.ep)
   ret 1;
 else if (m.ep == s.ep)
  if (m.tx > s.tx)
   ret 1;
  else if (m.tx == s.tx &&
           m.lid > s.lid)
   ret 1;

 ret 0;
```

Crash-Message Independence (CMI)

```
bool pg (s, X) :
 if (s.rl == F && X.rl == L)
  ret 1;
 if (s.rl == L && X.rl == F
     && !quorumAfterX(s)
  ret 1;
 if (s.rl == S && X.rl == S)
  ret 1;

bool pl (s, X) :
 if (s.rl == L && X.rl == F
     && quorumAfterX(s))
  ret 1;


bool quorumAfterX(s) :
  ret ((s.fol-1) >=
        s.all/2);
```

Crash Recovery Symmetry (CRS)

```
bool pr1(s,C):
 if (s.rl == L && C.rl == F
     && quorumAfterX(s))
  ret 1;
rals1:{rl,fol,all};

bool pr2(s,C):
 if (s.rl == L && C.rl == F
     && !quorumAfterX(s))
  ret 1;
rals2: {rl,fol,lid,ep,tx,clk}

bool pr3(s,C):
 if (s.rl == F && c.rl == L)
  ret 1;
rals3: {rl,fol,lid,ep,tx,clk}

bool pr4:
 if (s.rl == S)
  ret 1;
rals4: {rl,lid,ep,tx,clk}
```

- 35 LOC on average per protocol

Speed in Reaching Old Bugs

#executions/paths to reach the bugs (e.g., 2 paths = abcd, abdc)

Bug#	SAMC	Black-Box DPOR	Random	Random DPOR
ZooKeeper-335				
ZooKeeper-790				
ZooKeeper-975				
ZooKeeper-1075				
ZooKeeper-1419				
ZooKeeper-1492				
ZooKeeper-1653				
MapReduce-4748				
MapReduce-5489				
MapReduce-5505				
Cassandra-3395				
Cassandra-3626				
		5000+		

Speed in Reaching Old Bugs

#executions/paths to reach the bugs (e.g., 2 paths = abcd, abdc)

Bug#	SAMC	Black-Box DPOR		Random		Random DPOR	
	#exe	#exe	speedup	#exe	speedup	#exe	speedup
ZooKeeper-335	117	**5000+**	43+	1057	9	**5000+**	43+
ZooKeeper-790	7	14	2	225	32	82	12
ZooKeeper-975	53	967	18	71	1	163	3
ZooKeeper-1075	16	1081	68	86	5	250	16
ZooKeeper-1419	100	924	9	2514	25	987	10
ZooKeeper-1492	576	**5000+**	9+	**5000+**	9+	**5000+**	9+
ZooKeeper-1653	11	945	86	3756	341	3462	315
MapReduce-4748	4	22	6	6	2	6	2
MapReduce-5489	53	**5000+**	94+	**5000+**	94+	**5000+**	94+
MapReduce-5505	40	1212	30	**5000+**	125+	1210	30
Cassandra-3395	104	2552	25	191	2	550	5
Cassandra-3626	96	**5000+**	52+	**5000+**	52+	**5000+**	52

Summary

- Distributed concurrency bugs → hard to catch
- **Semantic-awareness** for model checking is powerful
  - Find bugs **2 - 340x** faster, **49x** on average

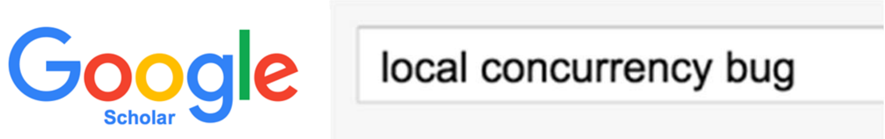
TaxDC:

Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems

with Tanakorn Leesatapornwongsa,

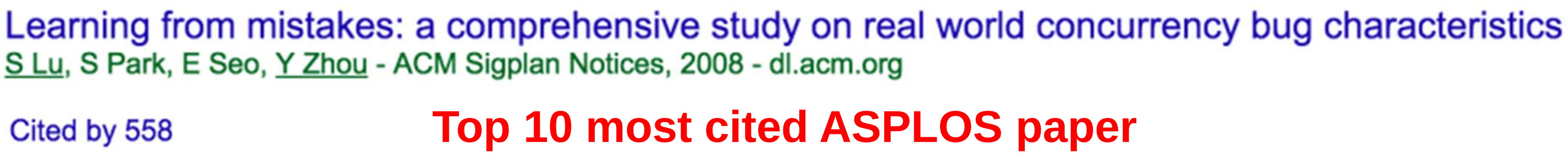
Jeffrey F. Lukman and Shan Lu

[ASPLOS '16]



local concurrency bug

(**LC** bug: multi-threaded single machine software)

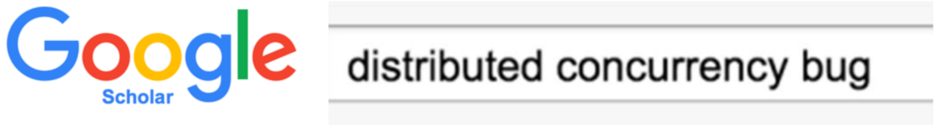


Learning from mistakes: a comprehensive study on real world concurrency bug characteristics
S Lu, S Park, E Seo, Y Zhou - ACM Sigplan Notices, 2008 - dl.acm.org

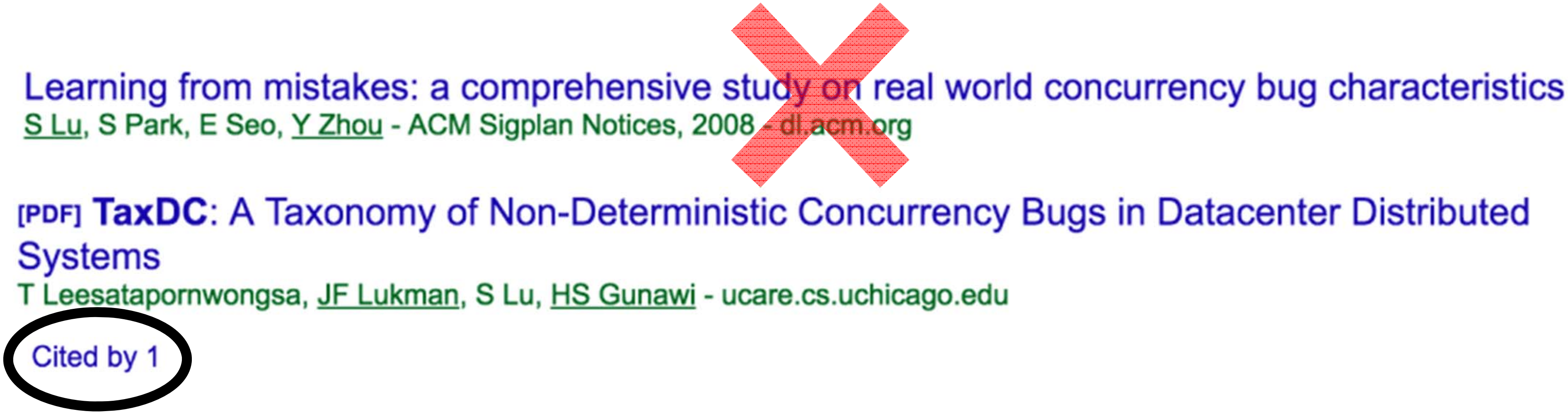
Cited by 558

Top 10 most cited ASPLOS paper

≠



distributed concurrency bug



Learning from mistakes: a comprehensive study on real world concurrency bug characteristics
S Lu, S Park, E Seo, Y Zhou - ACM Sigplan Notices, 2008 - dl.acm.org

[PDF] **TaxDC**: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems
T Leesatapornwongsa, JF Lukman, S Lu, HS Gunawi - ucare.cs.uchicago.edu

Cited by 1

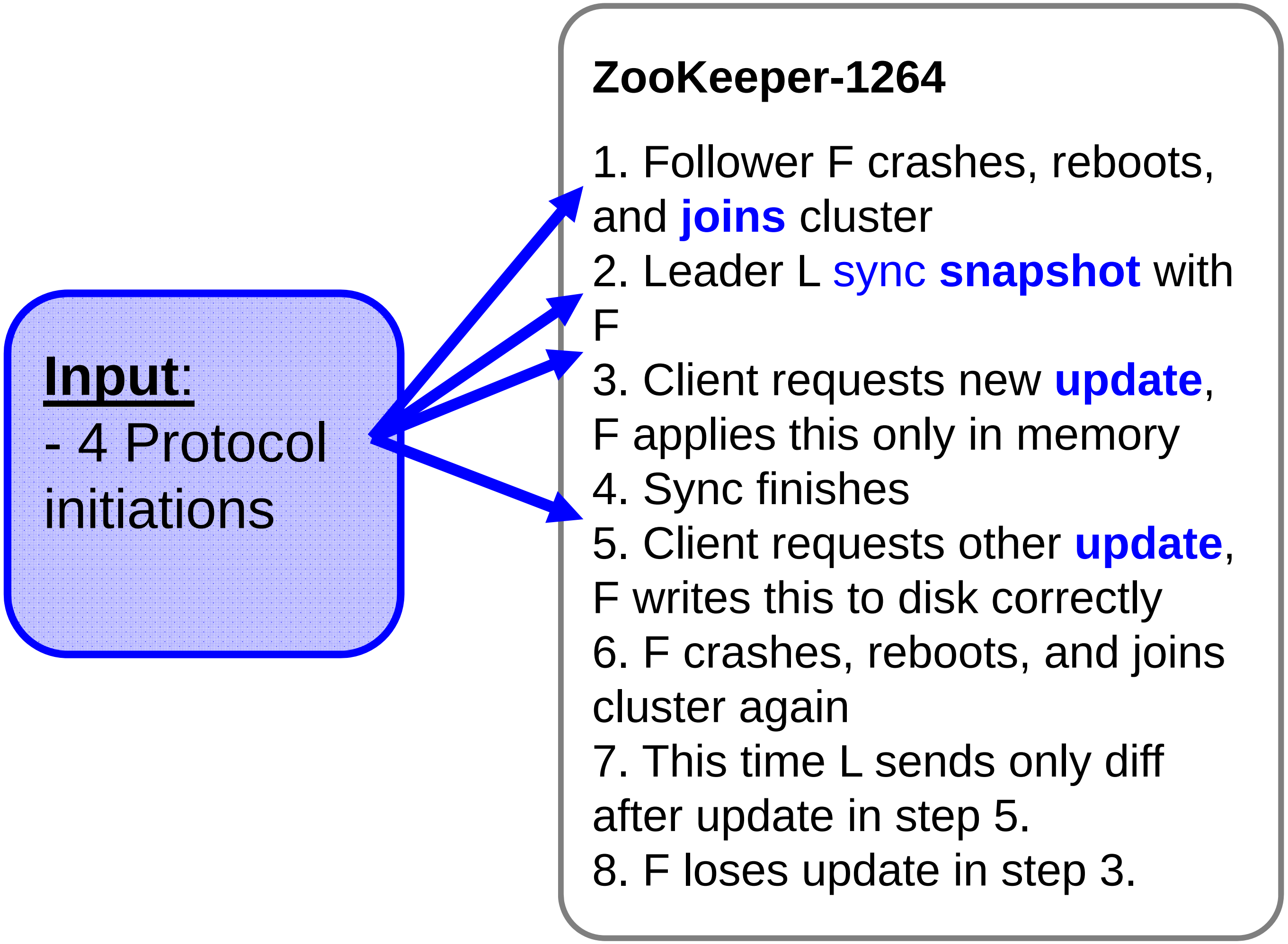
TaxDC

- □ Taxonomy of distributed concurrency bugs
- □ 104 bugs
- □ 4 varied distributed systems

cassandra hadoop APACHE HBASE Apache Zookeeper

- □ Bugs in 2011-2014
- □ Study description, source code, patches

Detailed Characteristics

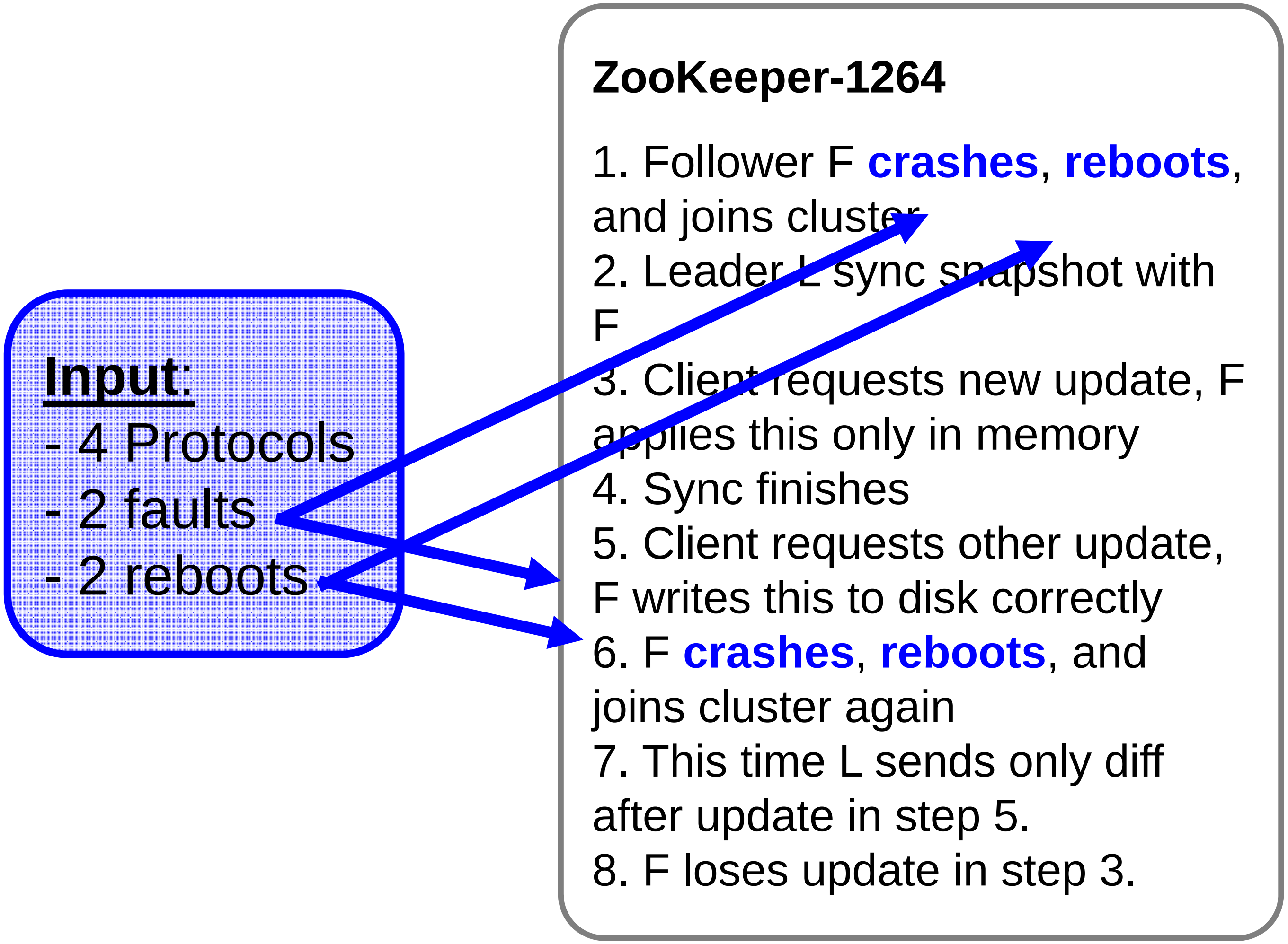


Input:
- 4 Protocol initiations

ZooKeeper-1264

1. Follower F crashes, reboots, and **joins** cluster
2. Leader L sync **snapshot** with F
3. Client requests new **update**, F applies this only in memory
4. Sync finishes
5. Client requests other **update**, F writes this to disk correctly
6. F crashes, reboots, and joins cluster again
7. This time L sends only diff after update in step 5.
8. F loses update in step 3.

Detailed Characteristics



Input:
- 4 Protocols
- 2 faults
- 2 reboots

ZooKeeper-1264

1. Follower F **crashes**, **reboots**, and joins cluster
2. Leader L sync snapshot with F
3. Client requests new update, F applies this only in memory
4. Sync finishes
5. Client requests other update, F writes this to disk correctly
6. F **crashes**, **reboots**, and joins cluster again
7. This time L sends only diff after update in step 5.
8. F loses update in step 3.

Detailed Characteristics

Input:
- 4 Protocols
- 2 faults
- 2 reboots

ZooKeeper-1264

1. Follower F crashes, reboots, and joins cluster
2. Leader L **sync** snapshot with F
3. Client requests new **update**, F applies this only in memory
4. **Sync** finishes
5. Client requests other update, F writes this to disk correctly
6. F **crashes**, reboots, and joins cluster again
7. This time L sends only diff after update in step 5.
8. F loses update in step 3.

Timing:
- Atomicity violation
- Fault Timing

Detailed Characteristics

Input:
- 4 Protocols
- 2 faults
- 2 reboots

ZooKeeper-1264

1. Follower F crashes, reboots, and joins cluster
2. Leader L sync snapshot with F
3. Client requests new update, F applies this only in memory
4. Sync finishes
5. Client requests other update, F writes this to disk correctly
6. F crashes, reboots, and joins cluster again
7. This time L sends only diff after update in step 5.
8. F loses update in step 3.

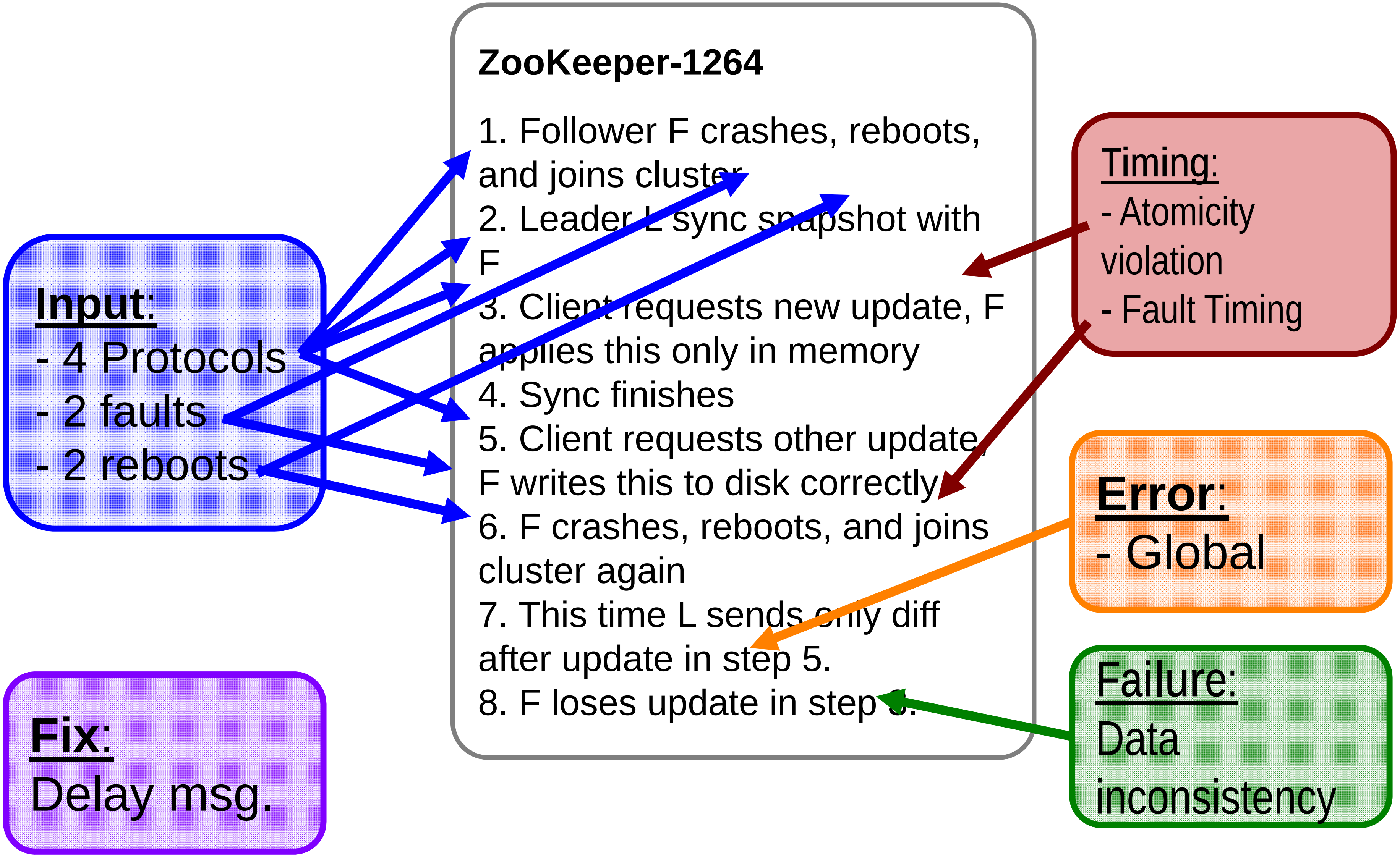
Timing:
- Atomicity violation
- Fault Timing

Error:
- Global

Failure:
Data inconsistency

Fix:
Delay msg.

Detailed Characteristics



ZooKeeper-1264

1. Follower F crashes, reboots, and joins cluster
2. Leader L sync snapshot with F
3. Client requests new update, F applies this only in memory
4. Sync finishes
5. Client requests other update, F writes this to disk correctly
6. F crashes, reboots, and joins cluster again
7. This time L sends only diff after update in step 5.
8. F loses update in step 5.

Input:
- 4 Protocols
- 2 faults
- 2 reboots

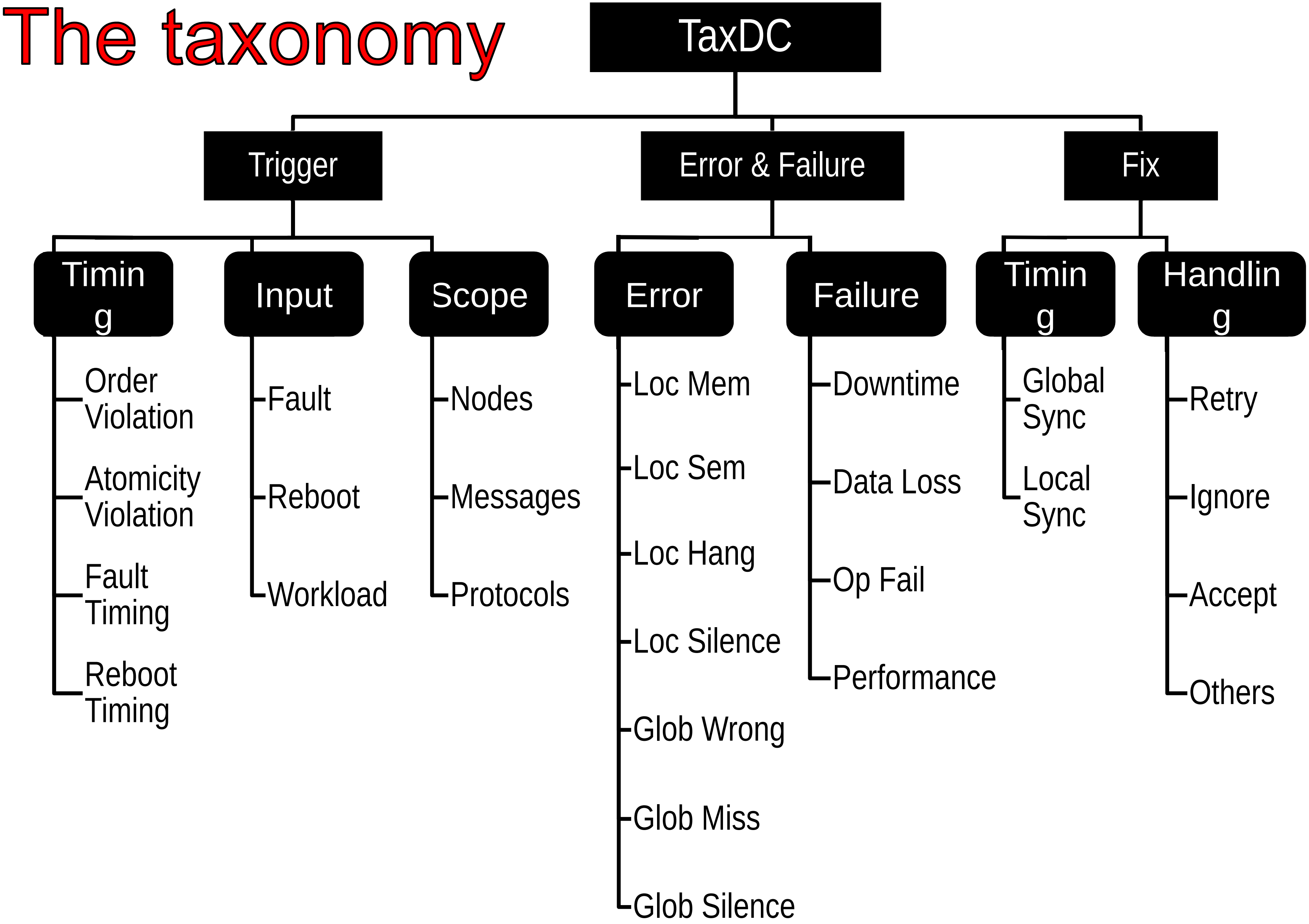
Fix:
Delay msg.

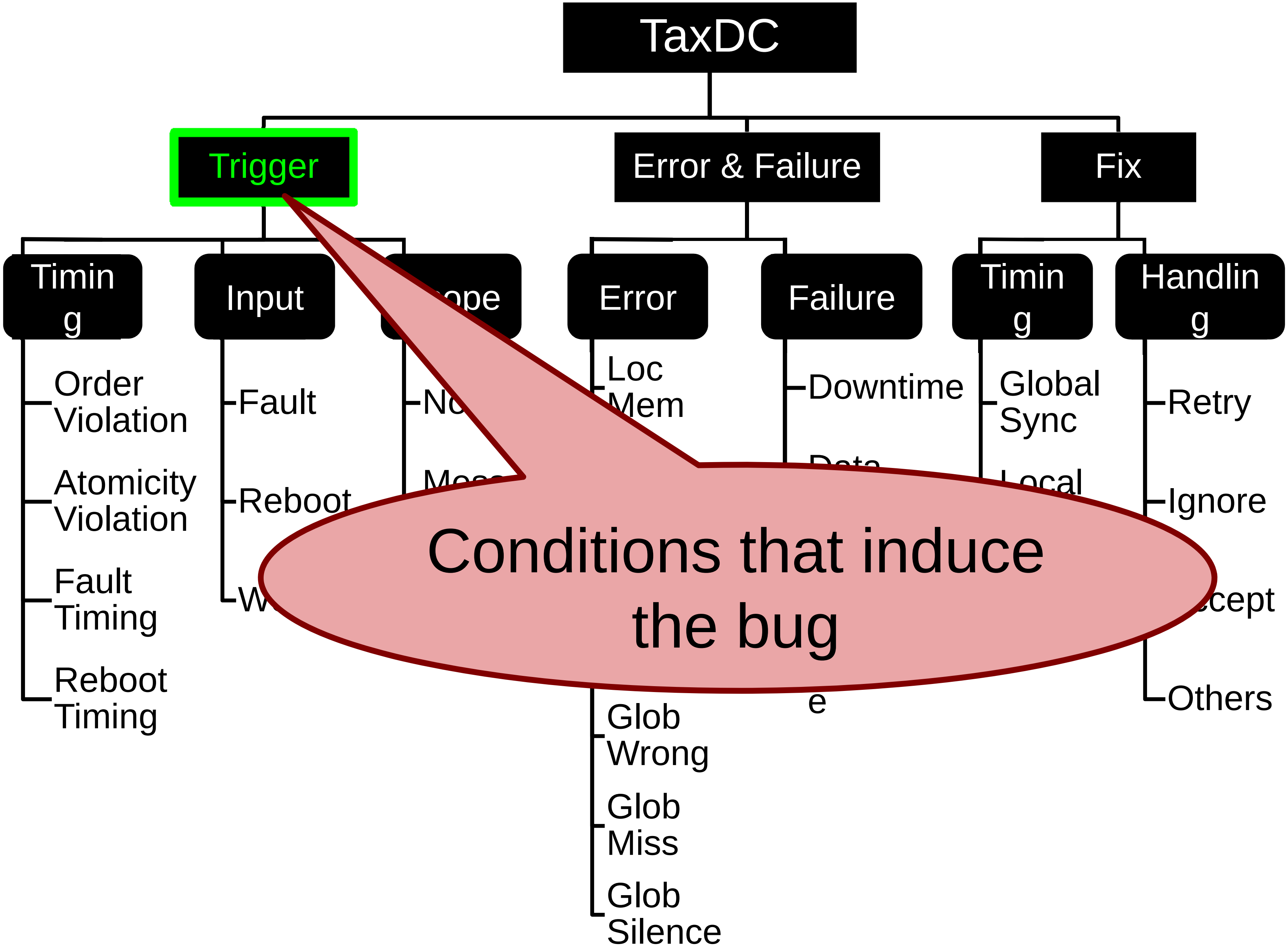
Timing:
- Atomicity violation
- Fault Timing

Error:
- Global

Failure:
Data inconsistency

The taxonomy





TaxDC

Trigger

Error & Failure

Fix

Timin g

Input

ope

Error

Failure

Timin g

Handlin g

Order Violation

Atomicity Violation

Fault Timing

Reboot Timing

Fault

Reboot

W

No

Mess

Loc Mem

Glob Wrong

Glob Miss

Glob Silence

Downtime

Data

e

Global Sync

Local

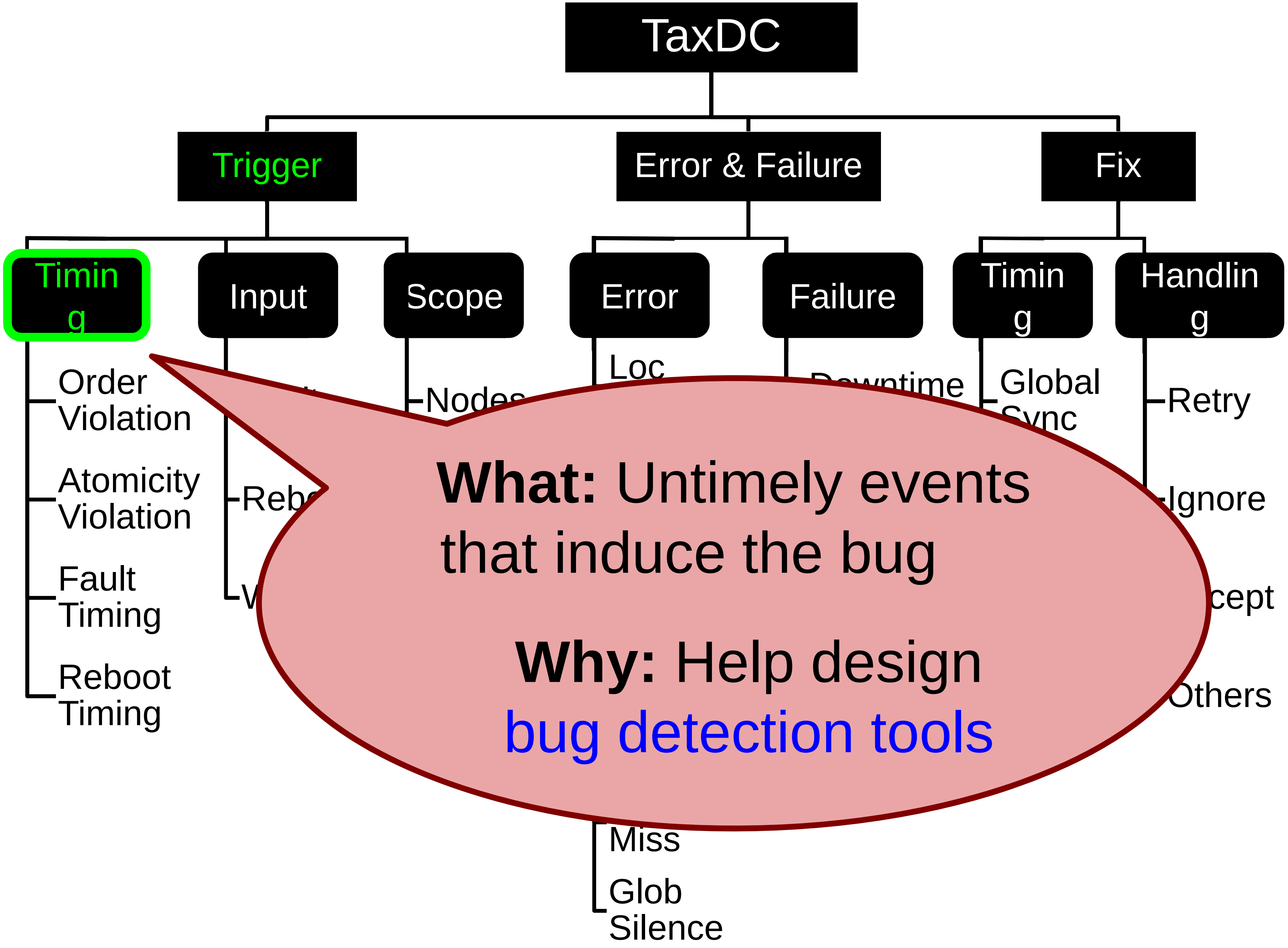
Retry

Ignore

ccept

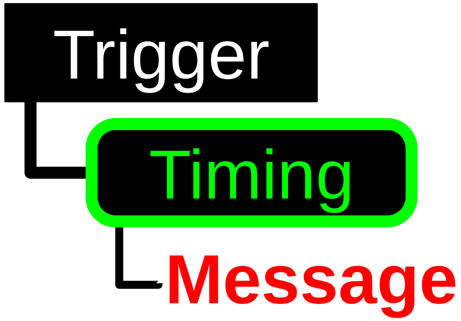
Others

Conditions that induce the bug



What: Untimely events that induce the bug

Why: Help design bug detection tools

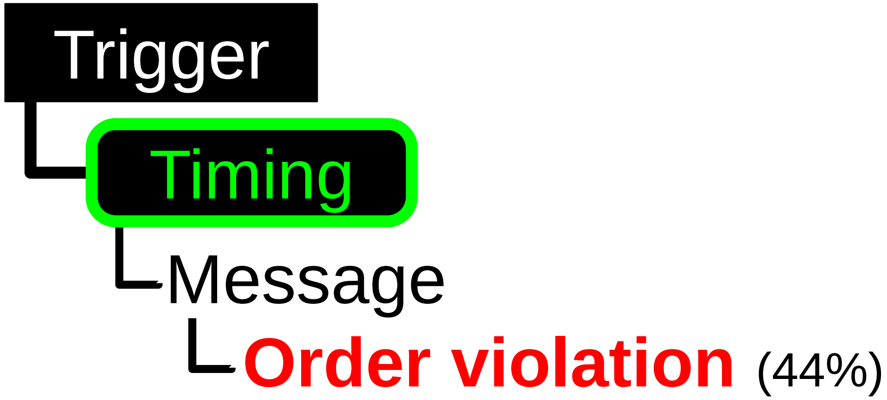


Trigger

Timing

Message

Messages arrive in untimely order



Trigger

Timing

Message

Order violation (44%)

*Y must happen **after** X*
*But Y happens **before** X*

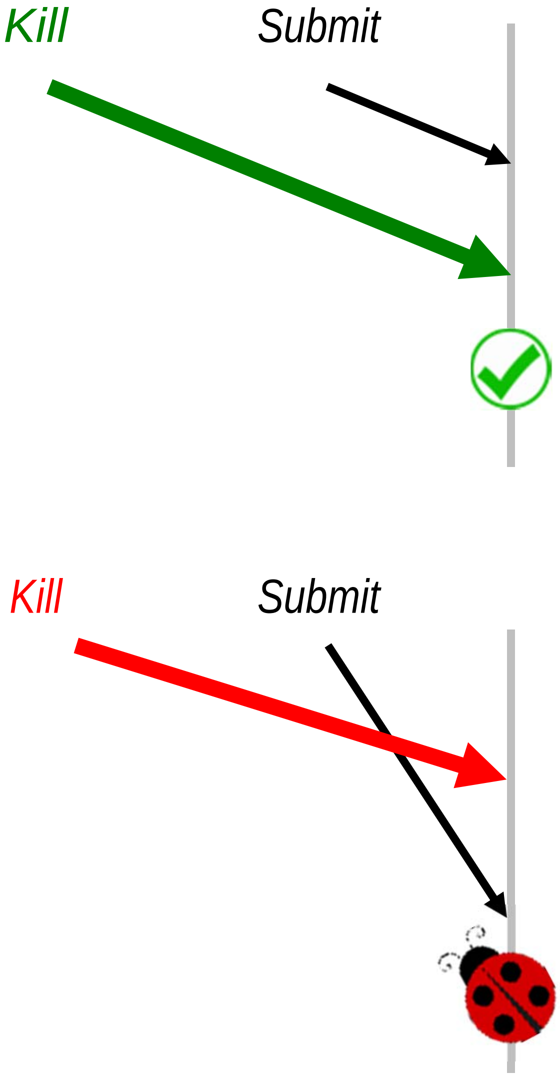
Trigger

└ Timing

└ Message

└ **Order violation** (44%)

└ Msg-msg race



*Y must happen **after** X*
*But Y happens **before** X*

Ex: MapReduce-3274

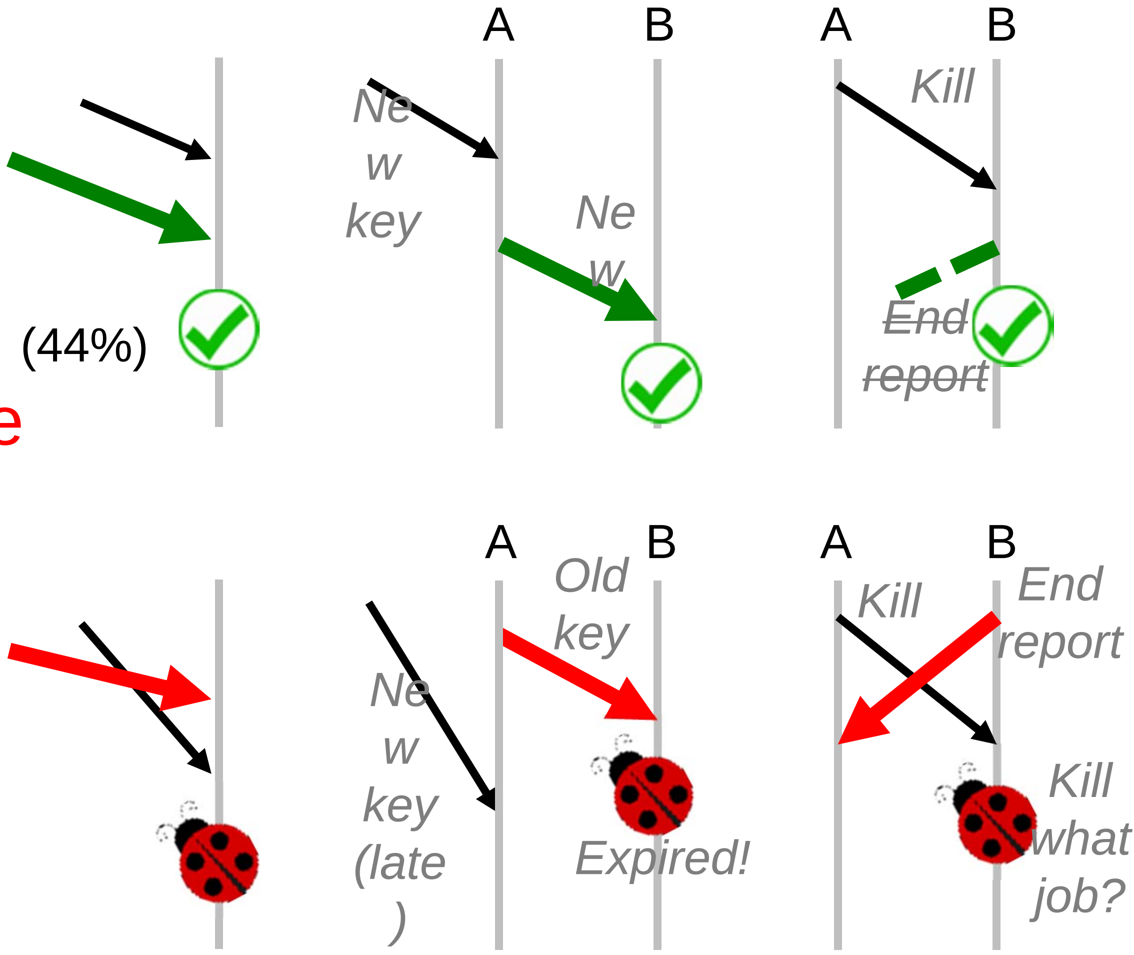
Trigger

└ Timing

└ Message

└ **Order violation** (44%)

└ Msg-msg race



Receive-receive

MapReduce-3274

Receive-send

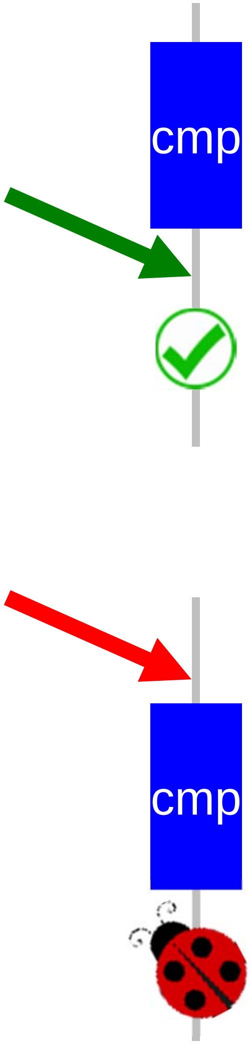
HBase-5780

Send-send race

MapReduce-5358

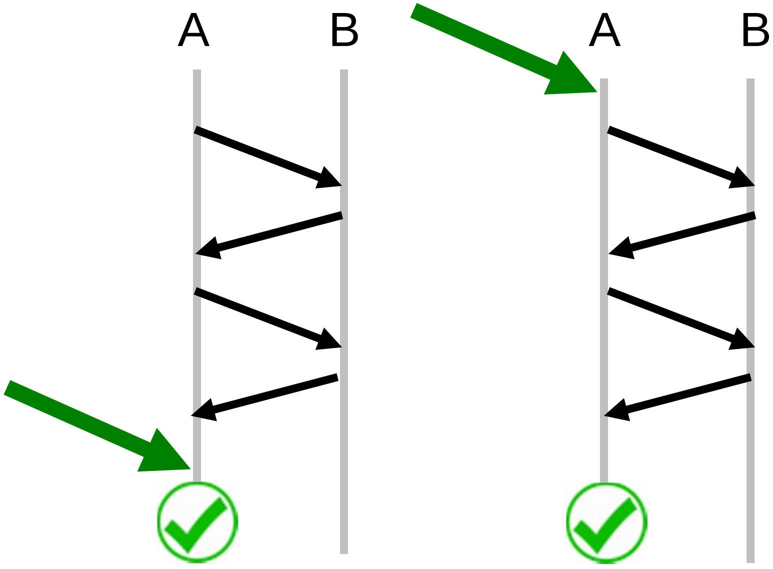
Trigger

- Timing
  - Message
    - **Order violation** (44%)
      - Msg-msg race
      - Msg-compute race

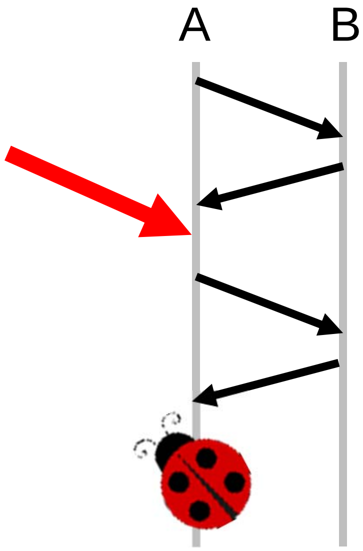


Ex: MapReduce-4157

Trigger
- Timing
  - Message
    - Order violation (44%)
    - **Atomicity violation** (20%)

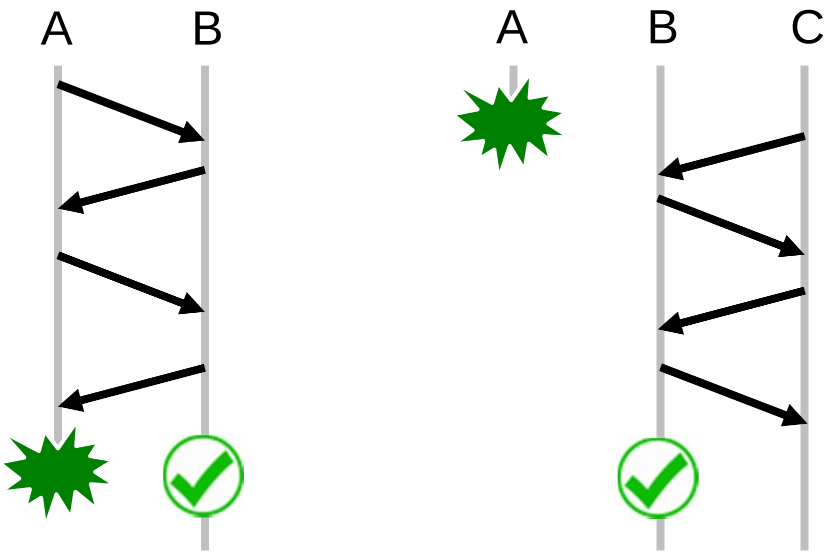


*A message comes in the **middle** of "atomic" operation*

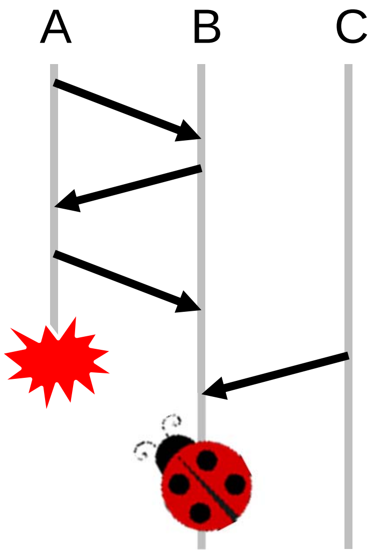


Ex: Cassandra-1011, Hbase-4729, MapReduce-5009, Zookeeper-1496

Trigger
- Timing
  - Message
  - **Fault** (21%)



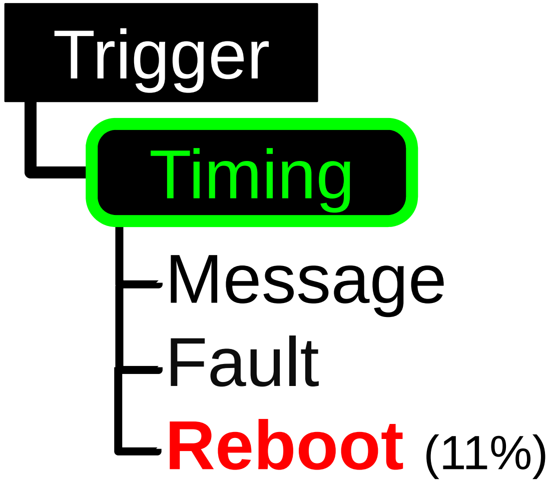
Fault at specific timing



***No** fault timing in **LC** bugs*

***Only** in **DC** bugs*

Ex: Cassandra-6415, Hbase-5806, MapReduce-3858, Zookeeper-1653



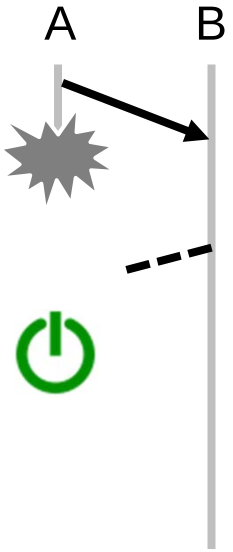
Trigger

Timing

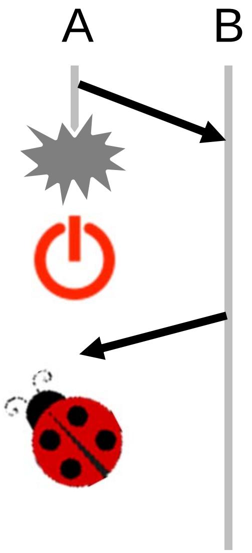
Message

Fault

Reboot (11%)



Reboot at specific timing

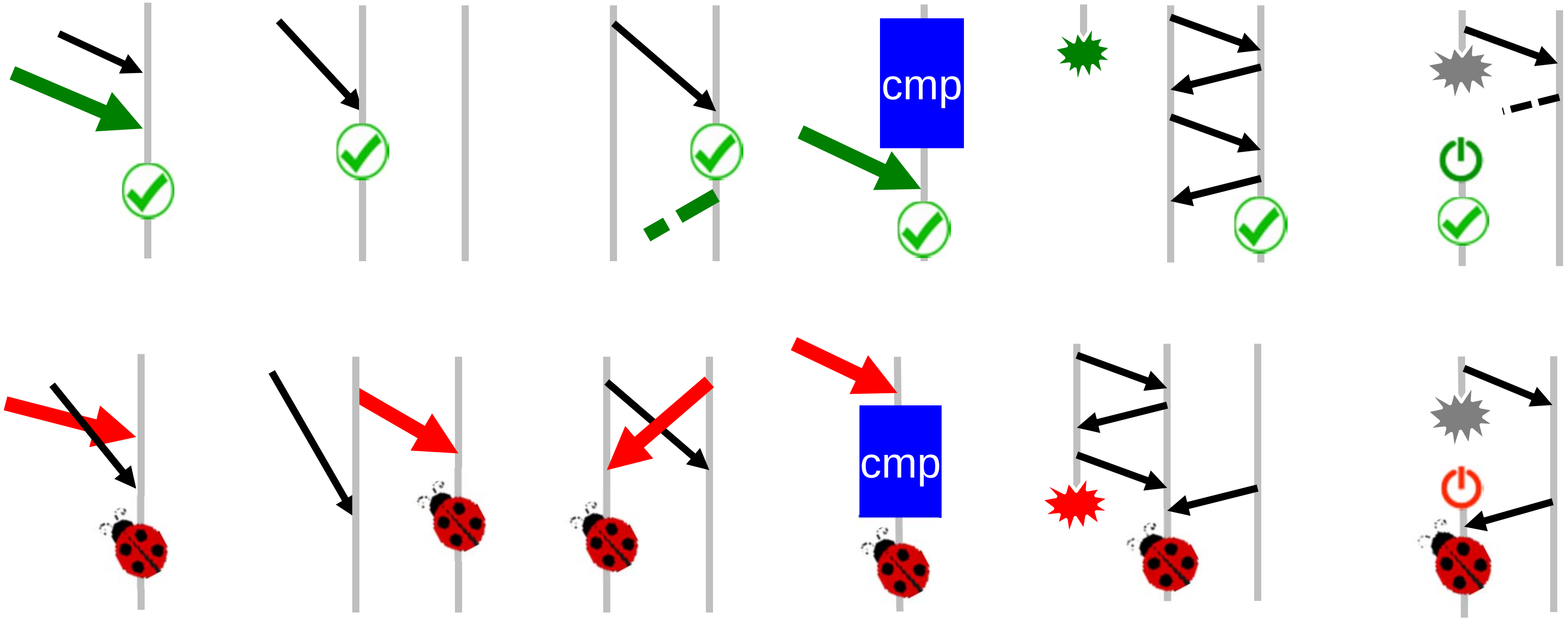


Ex: Cassandra-2083, Hadoop-3186, MapReduce-5489, Zookeeper-975

Trigger

Timing

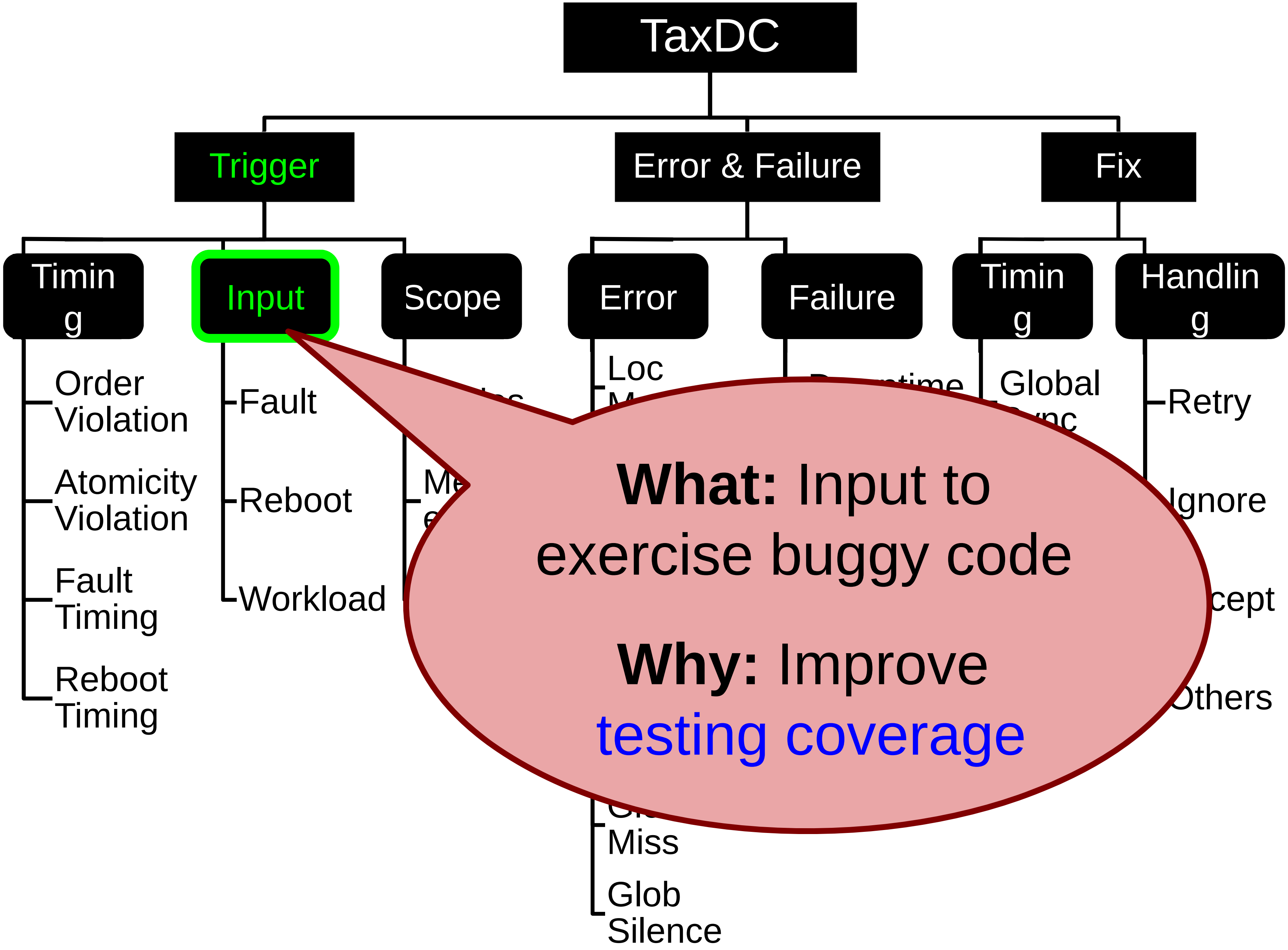
Implication: **simple patterns** can inform **pattern-based bug detection** tools, etc.



Message timing

Fault timing

Reboot timing



TaxDC

Trigger

Error & Failure

Fix

Timin
g

Input

Scope

Error

Failure

Timin
g

Handlin
g

Order Violation

Atomicity Violation

Fault Timing

Reboot Timing

Fault

Reboot

Workload

Me
e

Loc

Miss

Glob Silence

Global

Retry

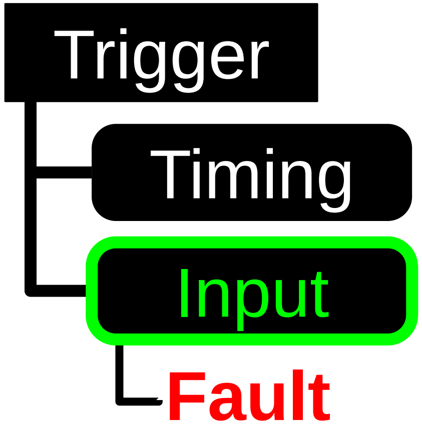
Ignore

cept

Others

What: Input to exercise buggy code

Why: Improve testing coverage



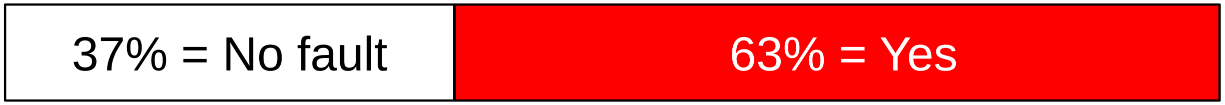
Trigger

Timing

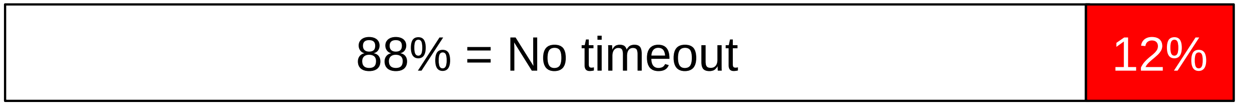
Input

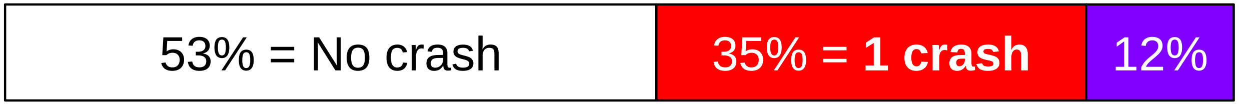
Fault

“How many bugs require fault injection?”

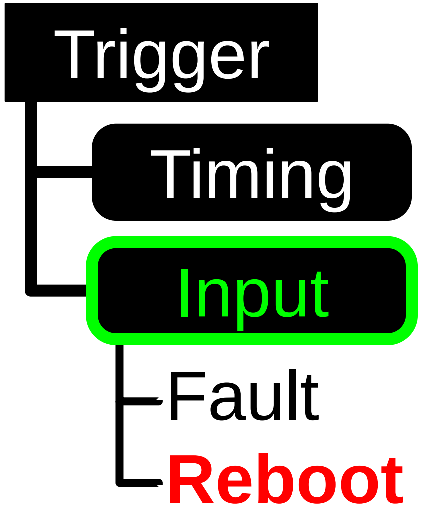


“What kinds of fault? & How many times?”

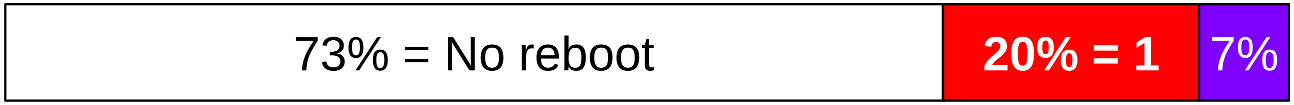


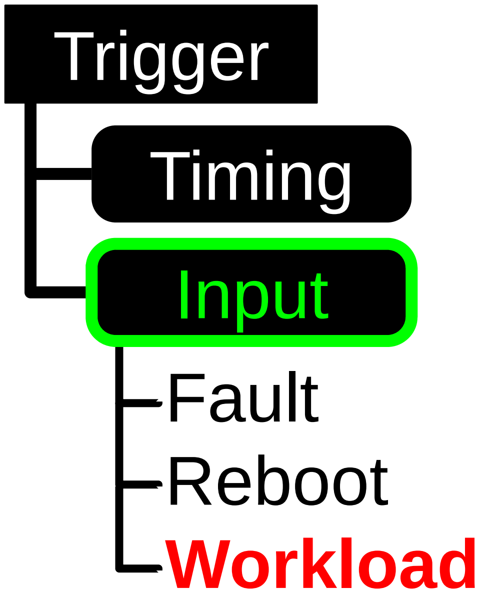


Real-world DC bugs are NOT just about message re-ordering, but faults as well



"How many reboots?"





Trigger

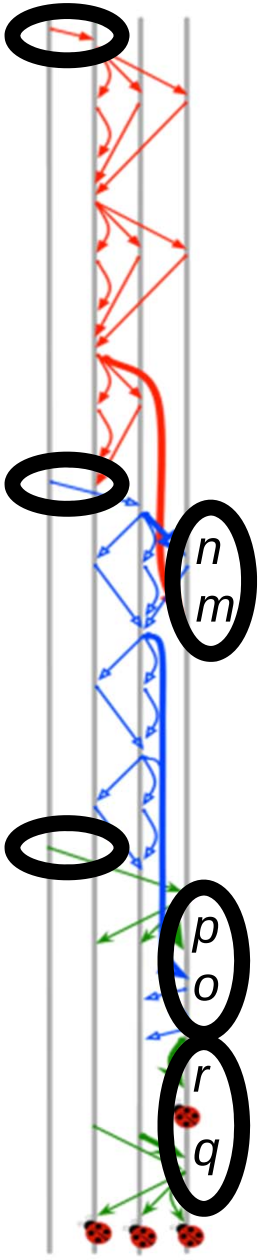
Timing

Input

Fault

Reboot

Workload



Cassandra Paxos bug

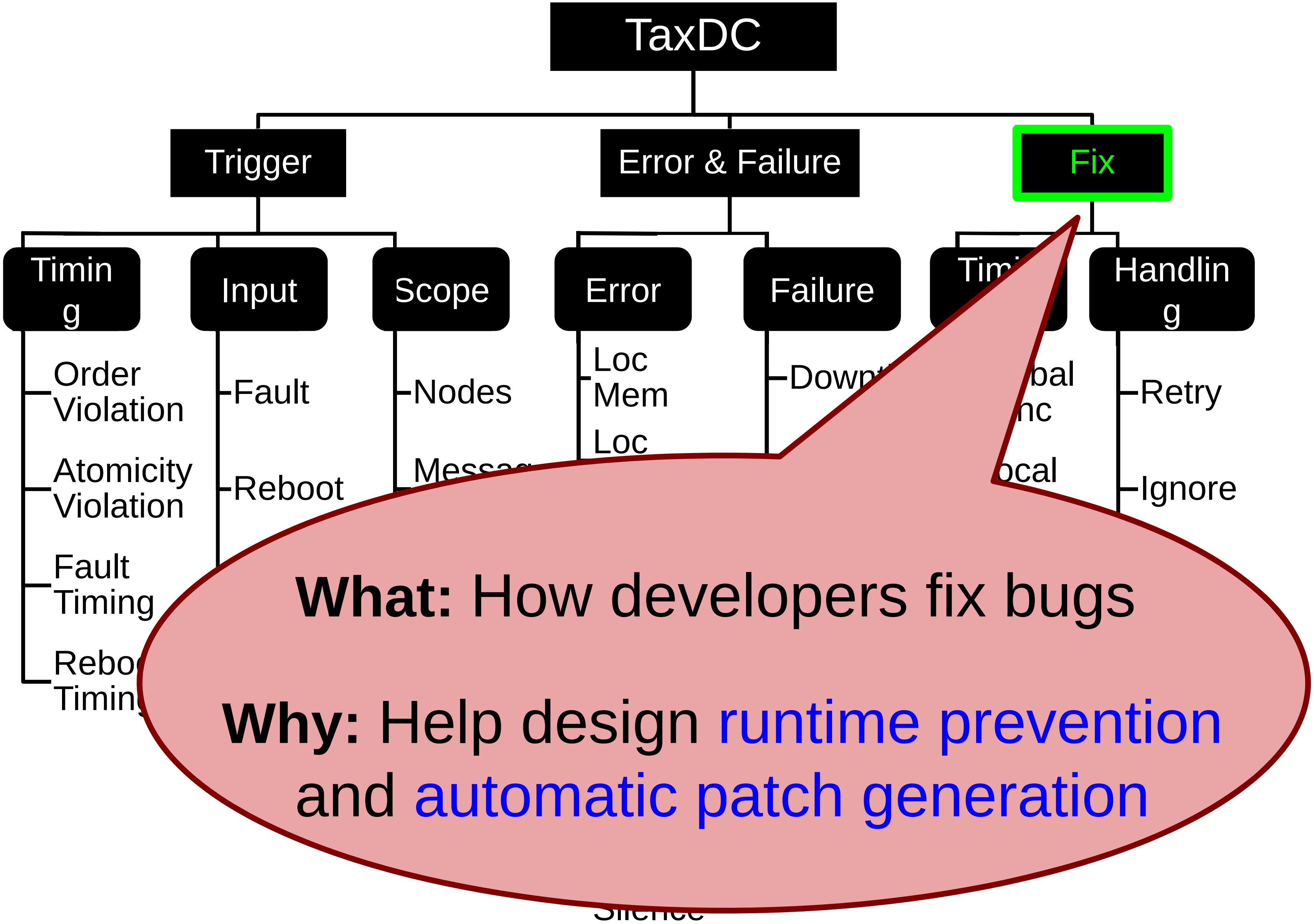
(Cassandra-6023)

3 concurrent
user requests!

“How many protocol initiations to run as input?”

20% = 1	29% = 2	24% = 3	27% = 4+

Implication: **multiple protocols**
for **DC testing**



TaxDC

Trigger

Error & Failure

Fix

Timin
g

Order Violation

Atomicity Violation

Fault Timing

Rebo
Timin

Input

Fault

Reboot

Scope

Nodes

Messag

Error

Loc
Mem

Loc

Silence

Failure

Down

Timi

bal
nc

ocal

Handlin
g

Retry

Ignore

What: How developers fix bugs

Why: Help design runtime prevention and automatic patch generation

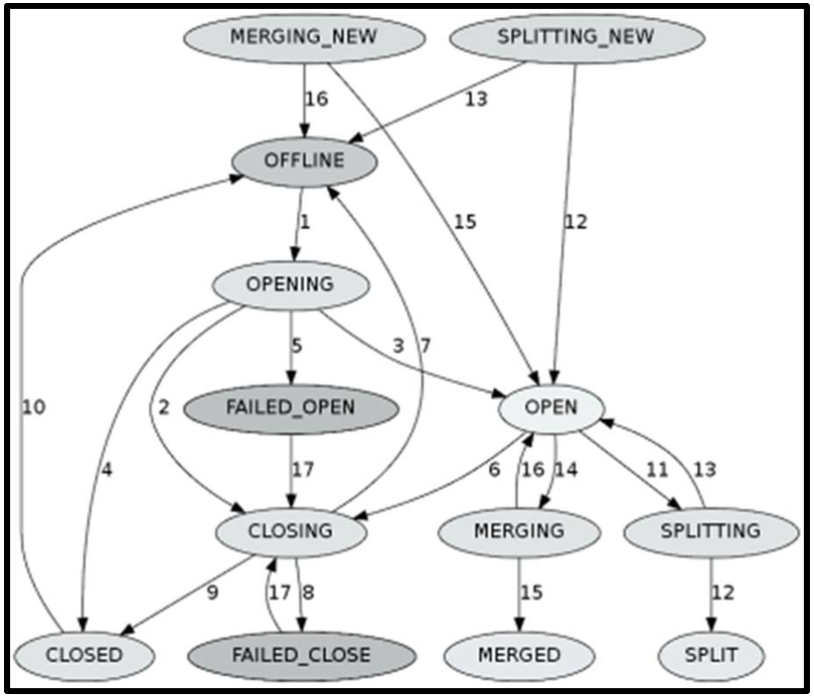
Trigger

Error

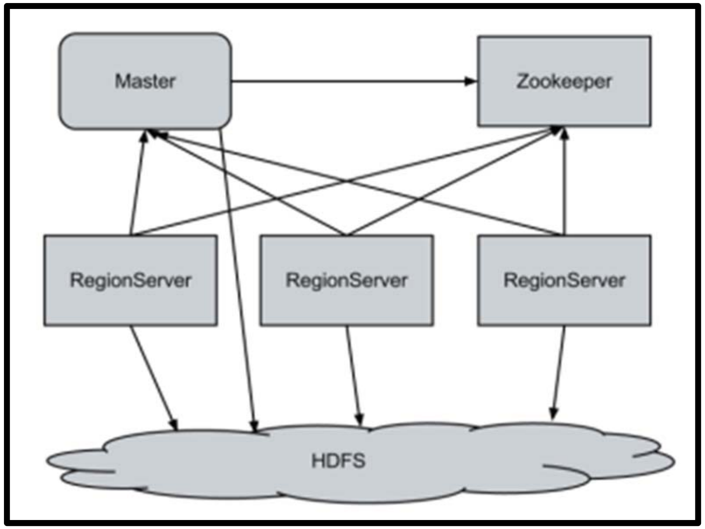
Fix

Complex

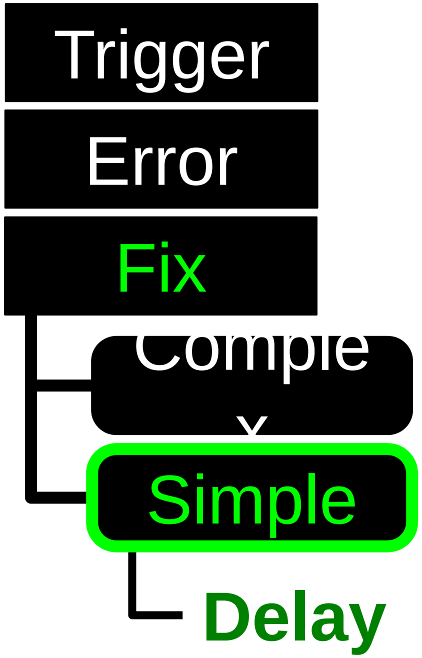
Add new states & transitions



Similar to fixing LC bugs:
add synchronization
e.g. lock()



Add Global Synchro-nization



Trigger

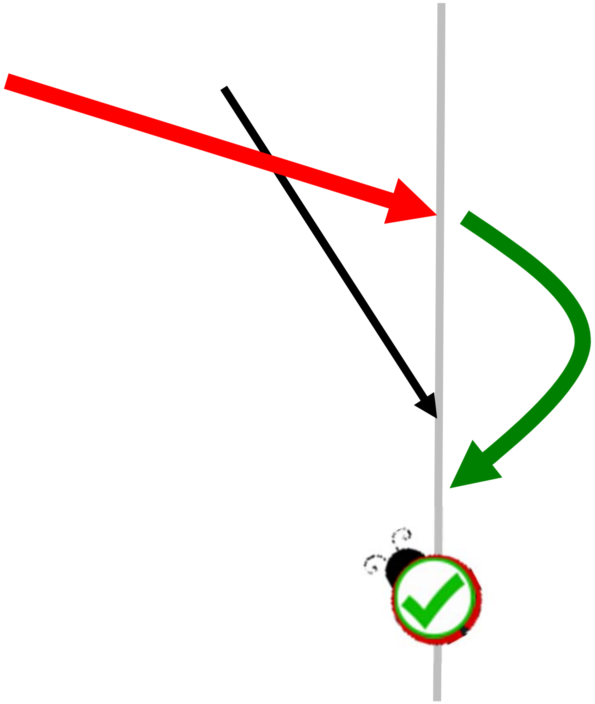
Error

Fix

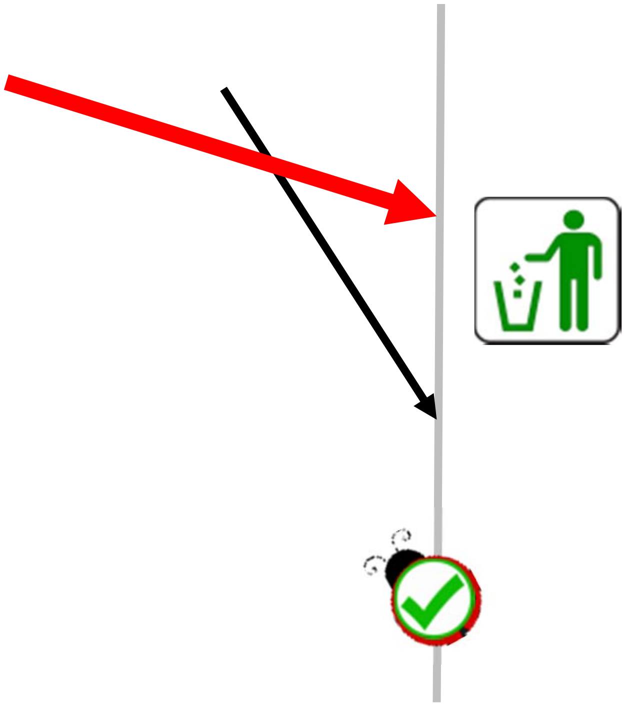
Complex

Simple

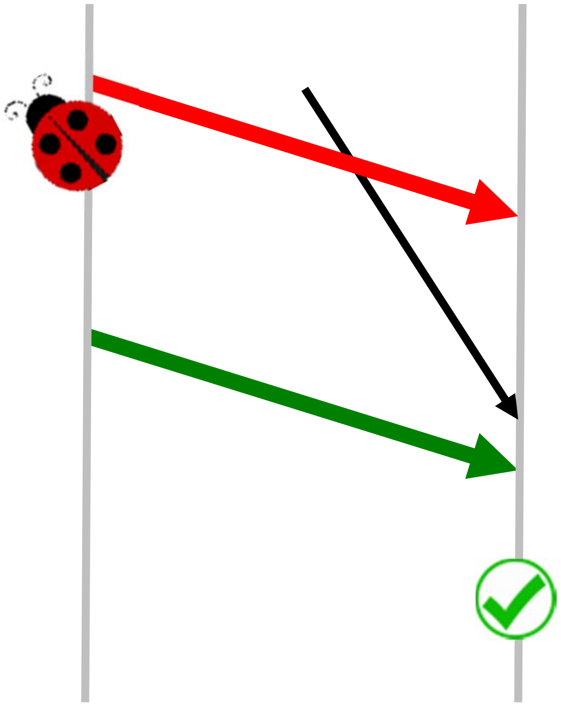
Delay

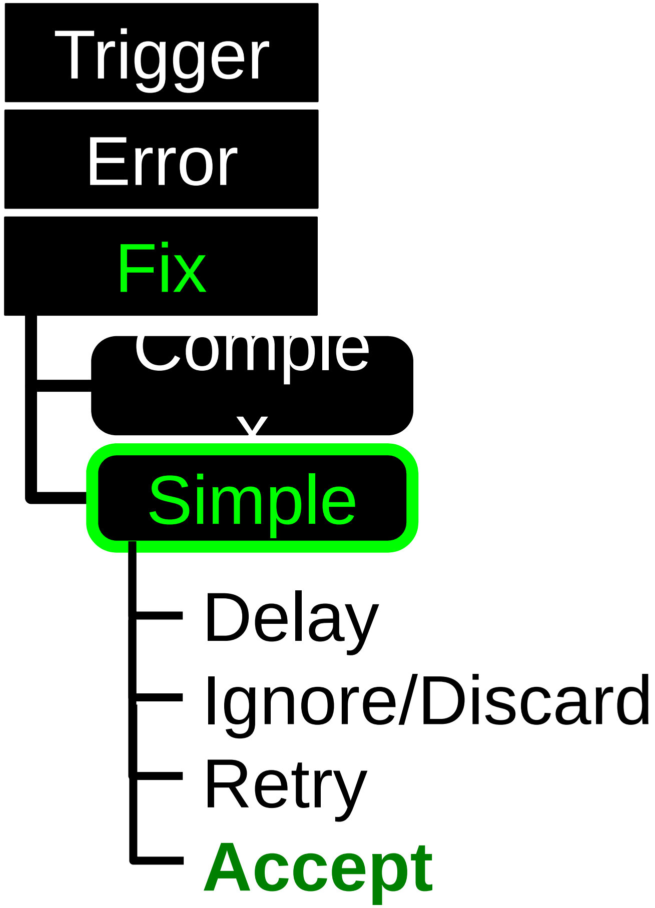


- Trigger
- Error
- Fix
  - Complex
  - Simple
    - Delay
    - **Ignore/discard**



- Trigger
- Error
- Fix
  - Complex
  - Simple
    - Delay
    - Ignore/Discard
    - **Retry**





Trigger

Error

Fix

Compile

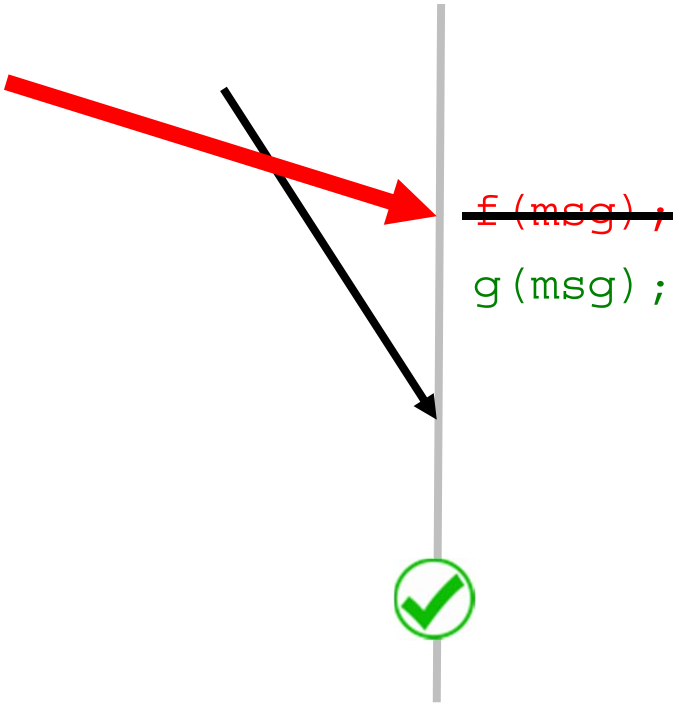
Simple

Delay

Ignore/Discard

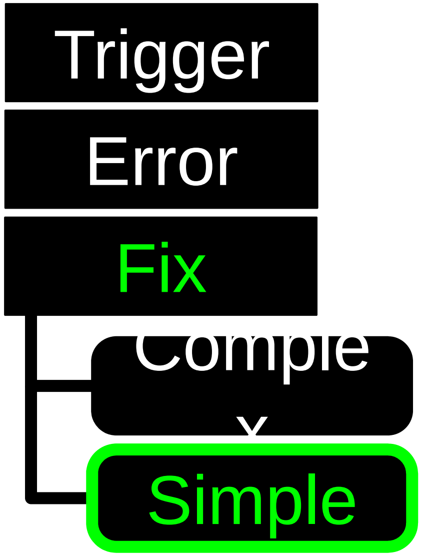
Retry

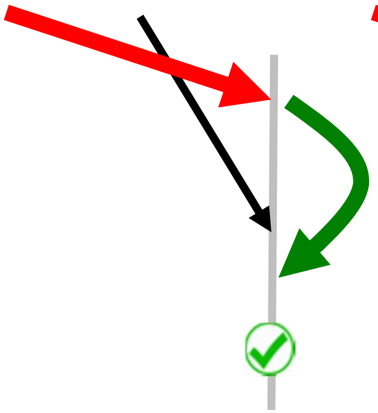
Accept

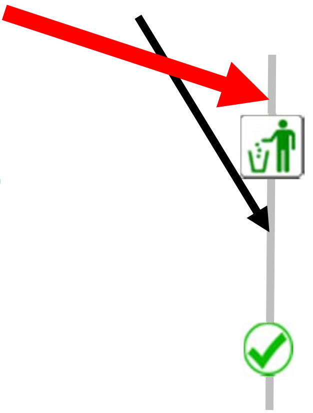


~~f(msg);~~

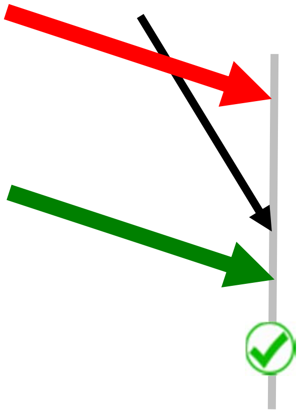
g(msg);

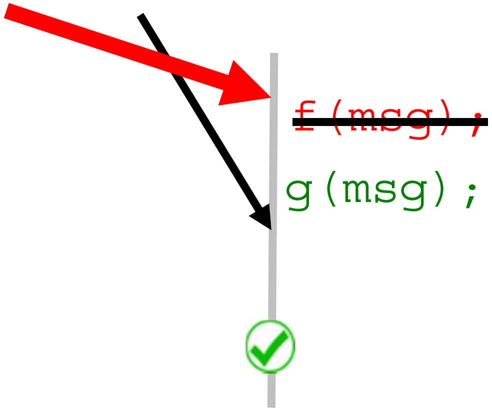



Delay

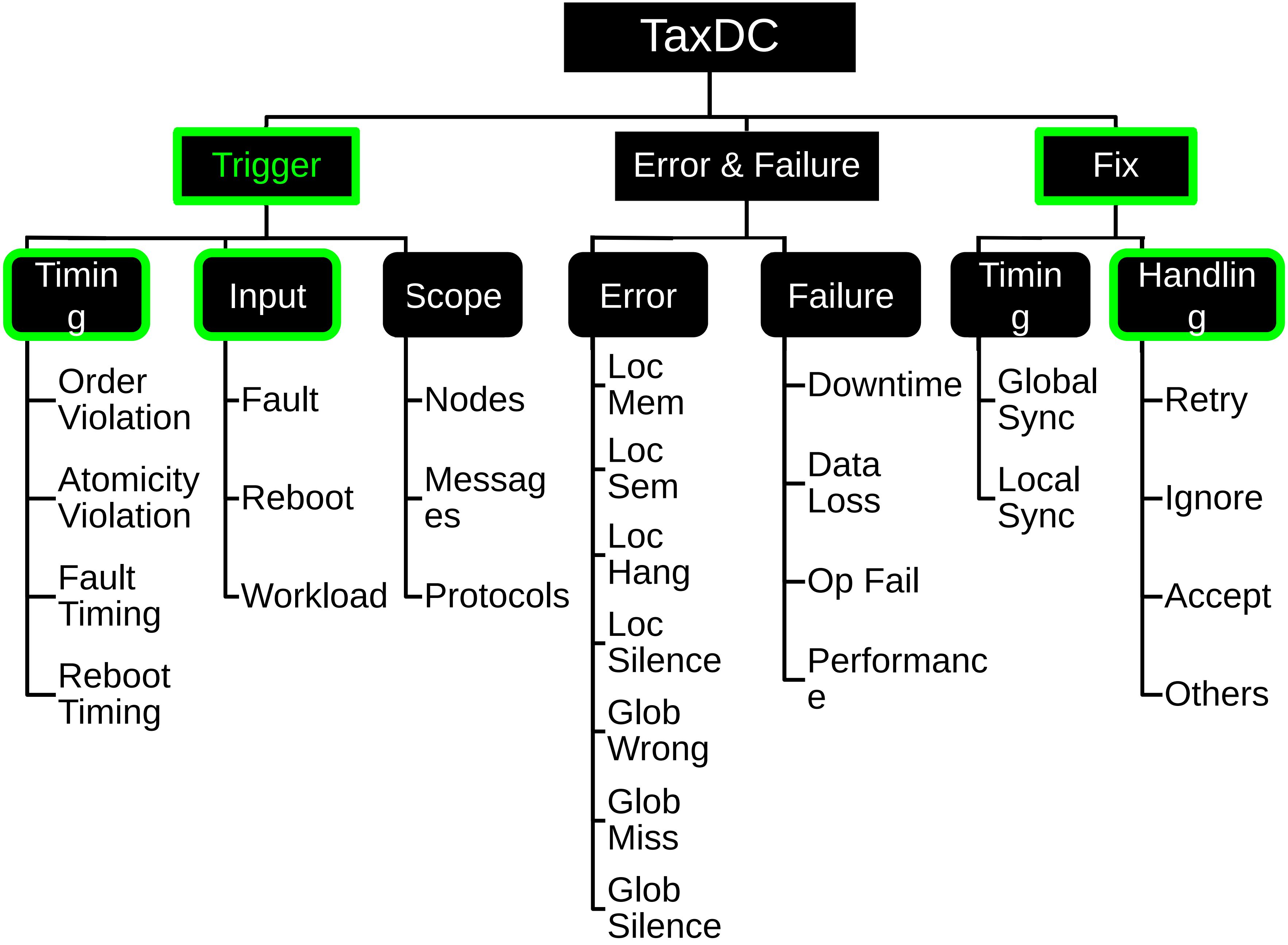

Ignore

40% are easy to fix
(***no*** *new computation logic*)


Retry



Accept



- TaxDC
  - Trigger
    - Timing
      - Order Violation
      - Atomicity Violation
      - Fault Timing
      - Reboot Timing
    - Input
      - Fault
      - Reboot
      - Workload
    - Scope
      - Nodes
      - Messages
      - Protocols
  - Error & Failure
    - Error
      - Loc Mem
      - Loc Sem
      - Loc Hang
      - Loc Silence
      - Glob Wrong
      - Glob Miss
      - Glob Silence
    - Failure
      - Downtime
      - Data Loss
      - Op Fail
      - Performance
  - Fix
    - Timing
      - Global Sync
      - Local Sync
    - Handling
      - Retry
      - Ignore
      - Accept
      - Others

Challenges & Opportunities in …

- ❑ Distributed system model checker
- ❑ Formal verification
- ❑ DC bug detection
- ❑ Runtime prevention

Distributed System Model Checkers

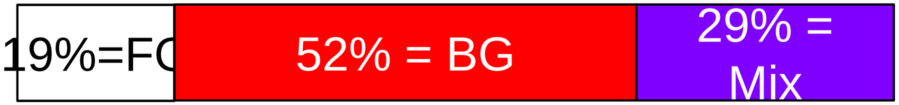
Event	Modist *NSDI'11*	Demeter *SOSP'11*	MaceMC *NSDI'07*	SAMC *OSDI'14*	Reality
Message	✓	✓	✓	✓	✓
Crash	✓	✓	✓	✓	✓
Multiple crashes	✗	✗	✗	✓	✓
Reboot	✗	✗	✓	✓	✓
Multiple reboots	✗	✗	✗	✓	✓
Timeout	✓	✓	✓	✗	✓
Computation	✓	✓	✗	✗	✓
Disk fault	✗	✗	✗	✗	✓

Formal Verification

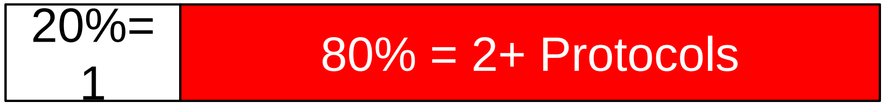
- State-of-the-art
  - Verdi *[PLDI '15]*
    - Raft update protocol
  - IronFleet *[SOSP '15]*
    - Paxos update protocol
    - Lease-based read/write

- Challenges

Foreground & Background



#Protocol interactions

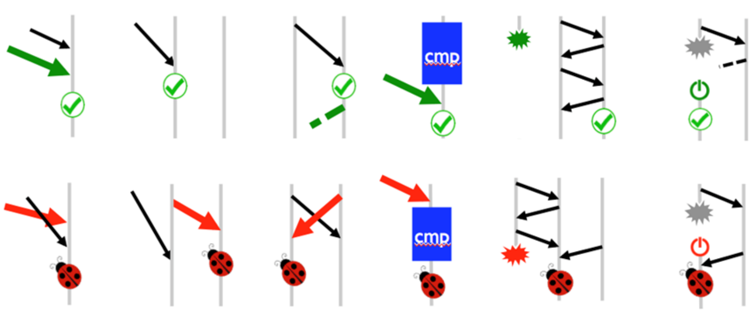


Only verify foreground protocols

Foreground & background

DC Bug Detection

- State-of-the-art: **LC** bug **detection**
  - Pattern-based detection
  - Error-based detection
  - Statistical bug detection
- Opportunities: **DC** bug **detection**?
  - Pattern-based detection



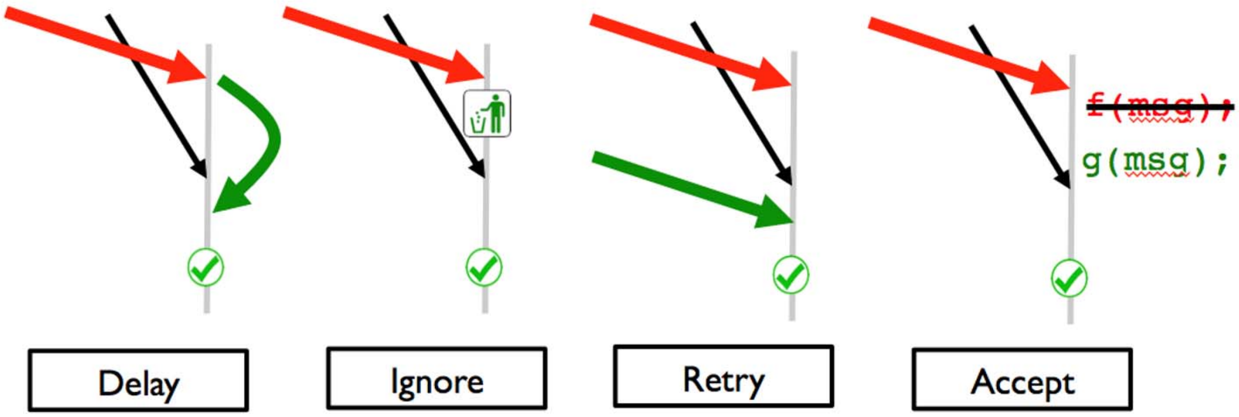
Runtime Failure Prevention

- State-of-the-art:
  LC bug prevention
  - Deadlock Immunity [*OSDI ‘08*]
  - Aviso [*ASPLOS ‘13*]
  - ConAir [*ASPLOS ‘13*]
  - *(many more)*

- Opportunities:
  DC bug **prevention**

Fixes

60% = Complex	40% = Simple



Dev’s comments on DC bugs

- *“Do we have to rethink this entire [HBase] root and meta ’huh hah’? There isn’t a week going by without some new bugs about races between splitting and assignment [distributed protocols].” — hbase4397*
- *“That is one monster of a race!” — mr3274*
- *“This has become quite messy, we didn’t foresee some of this during design, sigh.” — mr4819*
- *“Great catch, Sid! Apologies for missing the race condition” — mr4099*
- *“We have already found and fix many cases … however it seems exist many other cases.” — hb6147*

"New" classes of bugs …

- ❑ Distributed concurrency bugs
- ❑ Non-deterministic performance bugs

Limpware [SoCC '13]
Detect performance bugs [HotCloud '15]
Path-Based Spec. Exec. [In Subm.]

- ❑ Scalability bugs

A “limpware” anecdote

(limping hardware)

Limping NIC!

- ❑ *“… 1Gb NIC card on a machine that suddenly only transmits at 1 kbps,*
- ❑ *this slow machine caused a chain reaction upstream*
- ❑ *in such a way that the 100 node cluster began to crawl at a snail's pace.*
- ❑ *making the system non-available for all practical purposes.”* – Borthak

Cascading impact!

Limpware, really?

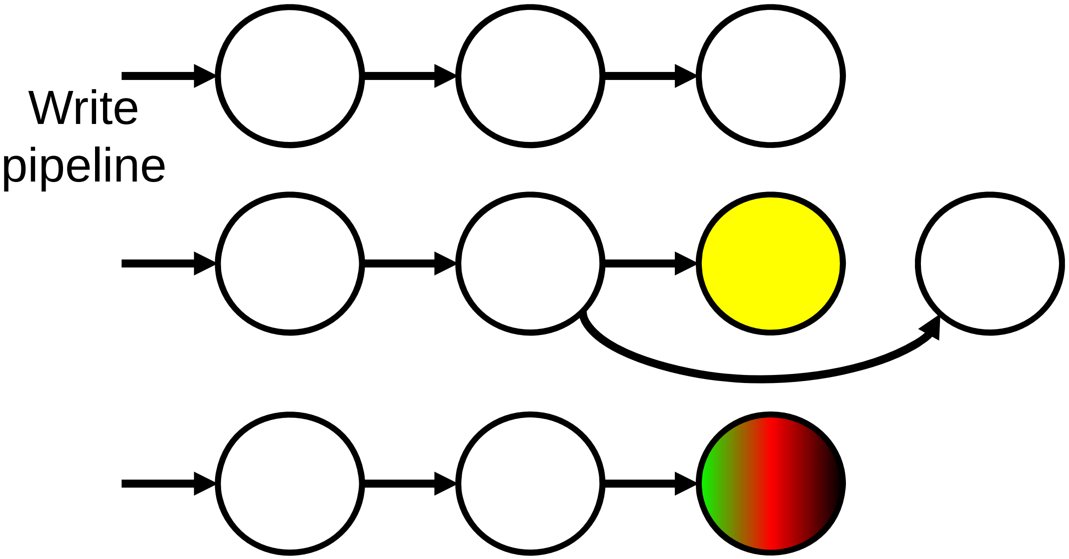
- *"In 2011, one of the DDN 9900 units had 4 servers having high wait times on I/O for a certain set of disk LUNs. The maximum wait time was 103 seconds. This was left uncorrected for 50 days."* – Kasick of CMU, Harms of Argonne
- *"The disk attempts to re-read each block multiple times before responding."* – Baptist of Cleversafe
- *"On Intrepid, we had a bad batch of optical transceivers with an extremely high error rate. That results in an effective throughput of 1-2 Kbps."* – Harms of Argonne
- Many others: ***"Yes, we've seen that in production"***

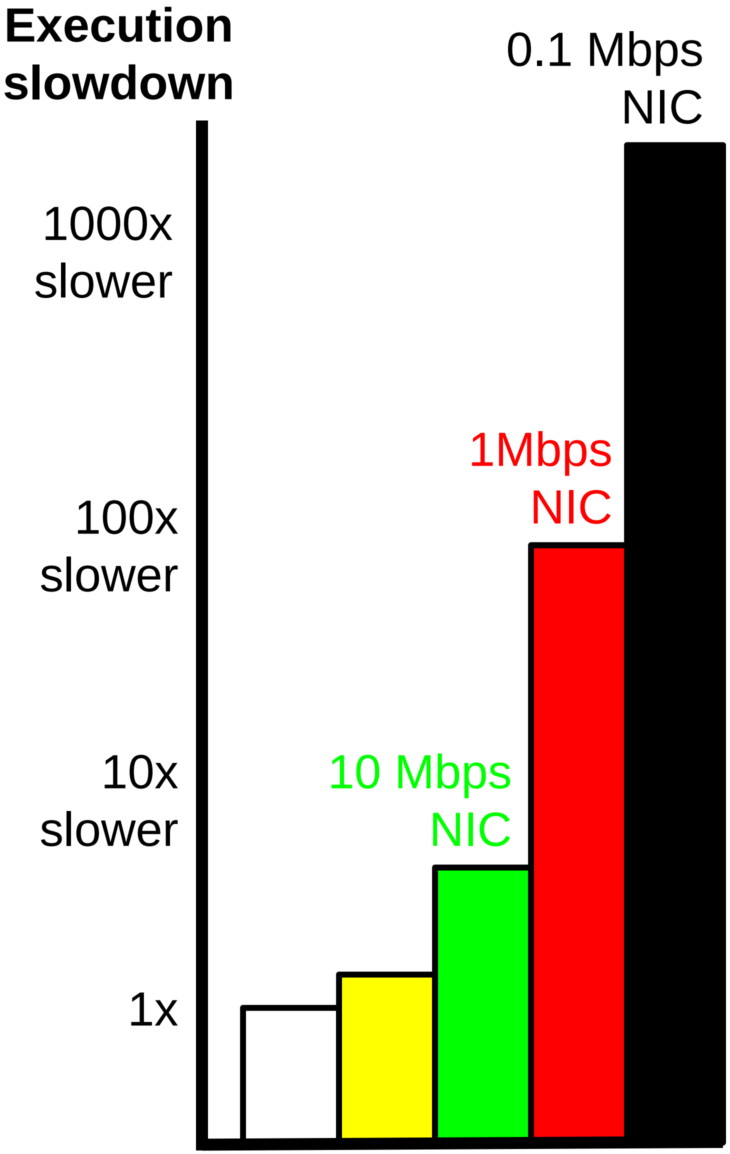
Limpware impacts?

- Modern distributed systems are …
  - … fault tolerant
  - … limpware tolerant?
- Limpware-injection experiments
  - Run HDFS, Hadoop, ZooKeeper, Cassandra, Hbase
  - Run load-intensive workload + inject limpware
    - E.g. slow a NIC to 1 Mbps, 0.1 Mbps, etc.

An example

- Run a distributed protocol
  - E.g., write pipeline in HDFS
- Measure slowdowns under:
  - No failure, crash, a limping NIC





Benchmarks

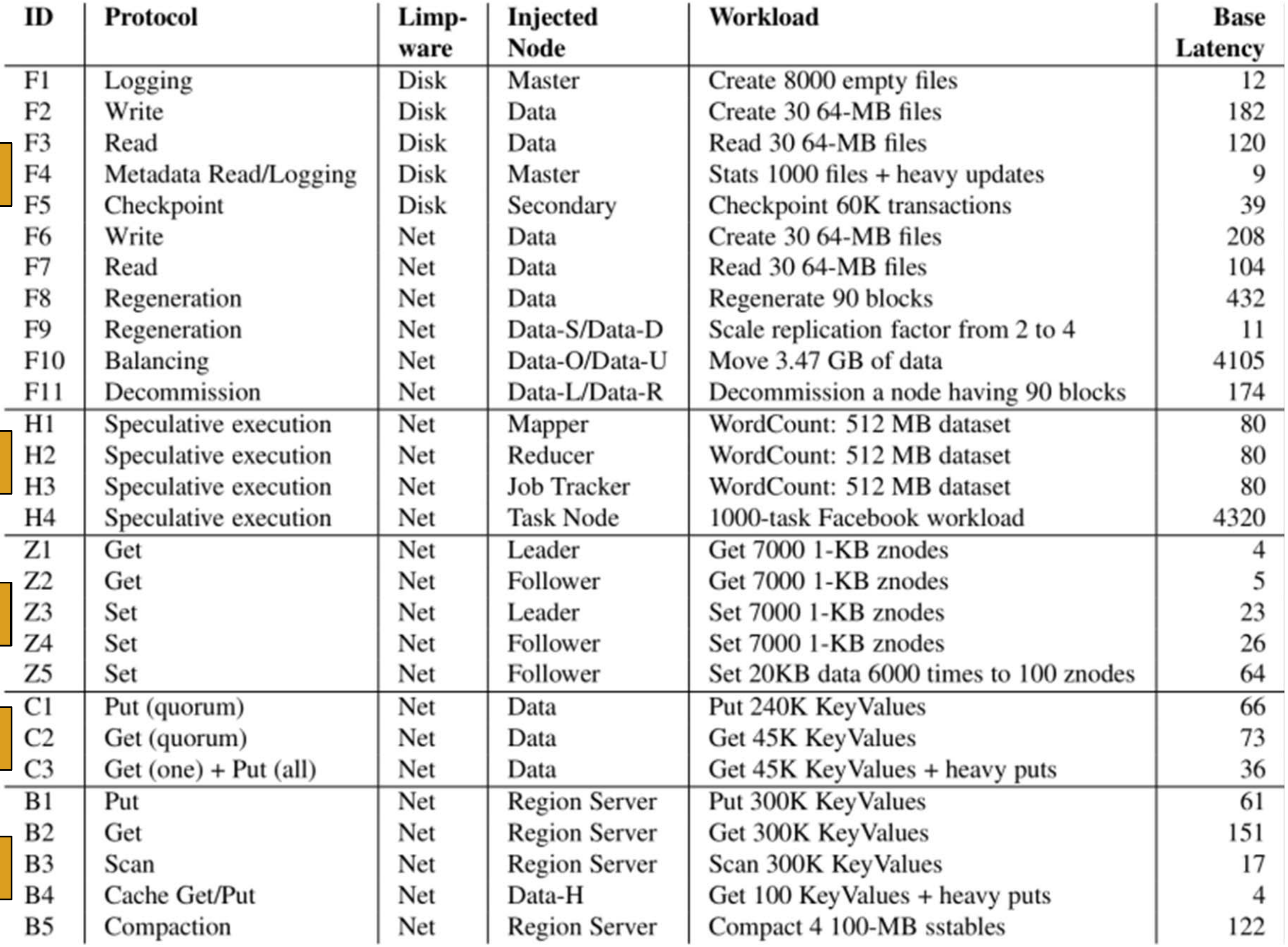
HDFS

Hadoop

ZooKeepe
r r

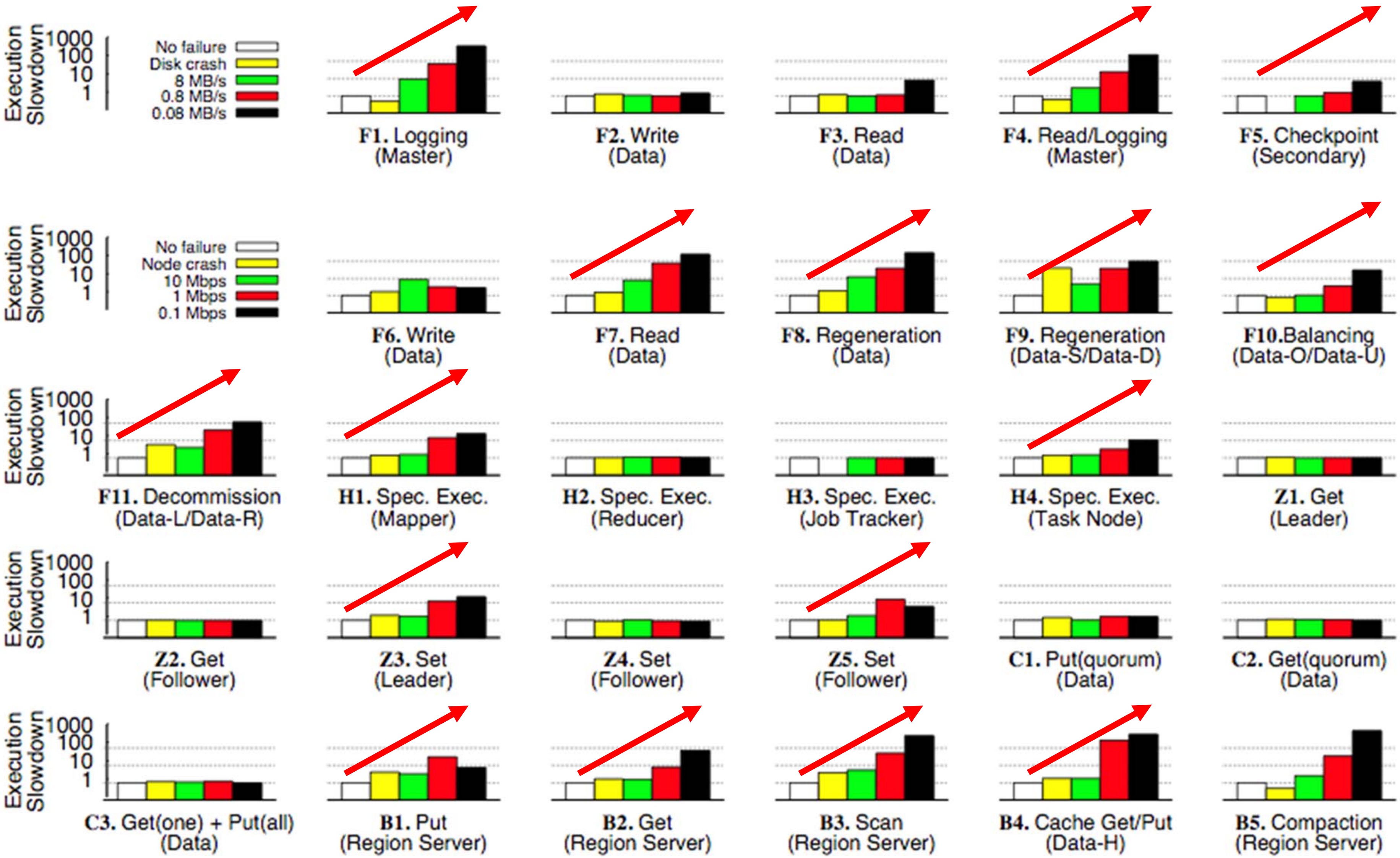
Cassandr
a

HBase



ID	Protocol	Limp-ware	Injected Node	Workload	Base Latency
F1	Logging	Disk	Master	Create 8000 empty files	12
F2	Write	Disk	Data	Create 30 64-MB files	182
F3	Read	Disk	Data	Read 30 64-MB files	120
F4	Metadata Read/Logging	Disk	Master	Stats 1000 files + heavy updates	9
F5	Checkpoint	Disk	Secondary	Checkpoint 60K transactions	39
F6	Write	Net	Data	Create 30 64-MB files	208
F7	Read	Net	Data	Read 30 64-MB files	104
F8	Regeneration	Net	Data	Regenerate 90 blocks	432
F9	Regeneration	Net	Data-S/Data-D	Scale replication factor from 2 to 4	11
F10	Balancing	Net	Data-O/Data-U	Move 3.47 GB of data	4105
F11	Decommission	Net	Data-L/Data-R	Decommission a node having 90 blocks	174
H1	Speculative execution	Net	Mapper	WordCount: 512 MB dataset	80
H2	Speculative execution	Net	Reducer	WordCount: 512 MB dataset	80
H3	Speculative execution	Net	Job Tracker	WordCount: 512 MB dataset	80
H4	Speculative execution	Net	Task Node	1000-task Facebook workload	4320
Z1	Get	Net	Leader	Get 7000 1-KB znodes	4
Z2	Get	Net	Follower	Get 7000 1-KB znodes	5
Z3	Set	Net	Leader	Set 7000 1-KB znodes	23
Z4	Set	Net	Follower	Set 7000 1-KB znodes	26
Z5	Set	Net	Follower	Set 20KB data 6000 times to 100 znodes	64
C1	Put (quorum)	Net	Data	Put 240K KeyValues	66
C2	Get (quorum)	Net	Data	Get 45K KeyValues	73
C3	Get (one) + Put (all)	Net	Data	Get 45K KeyValues + heavy puts	36
B1	Put	Net	Region Server	Put 300K KeyValues	61
B2	Get	Net	Region Server	Get 300K KeyValues	151
B3	Scan	Net	Region Server	Scan 300K KeyValues	17
B4	Cache Get/Put	Net	Data-H	Get 100 KeyValues + heavy puts	4
B5	Compaction	Net	Region Server	Compact 4 100-MB sstables	122

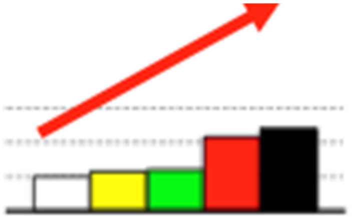
Fail-stop tolerant, but **not** limpware tolerant (no failover)



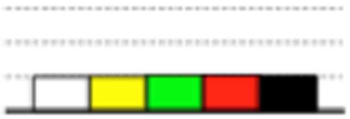
(The root causes are in Limpware paper [SOCC '13]; this talk focuses on Hadoop MapReduce)

Hadoop MapReduce

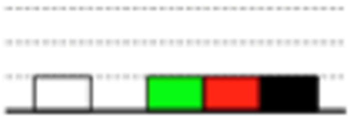
- ❑ Supposedly tail tolerant
- ❑ Why **not** limpware tolerant?
- ❑ Why Speculative Execution fails?



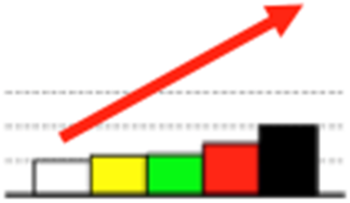
H1. Spec. Exec. (Mapper)



H2. Spec. Exec. (Reducer)



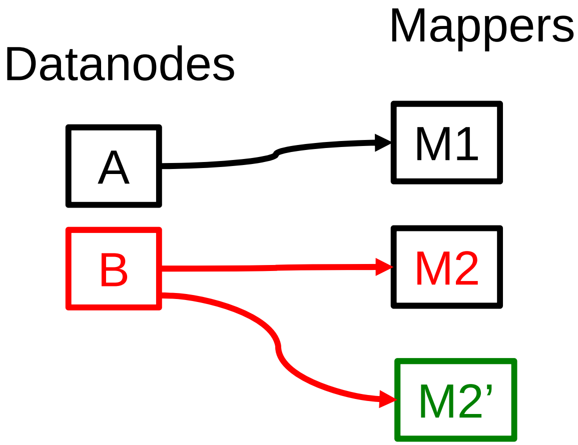
H3. Spec. Exec. (Job Tracker)



H4. Spec. Exec. (Task Node)

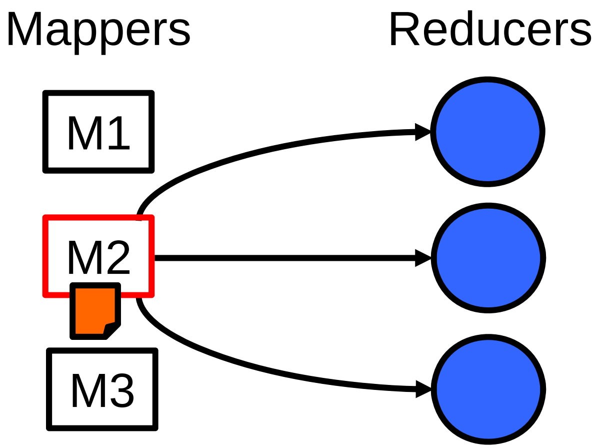
Loophole #1

- Backup task reads from the same slow datanode
  - Hadoop and HDFS don't cooperate
  - No history of bad "paths"



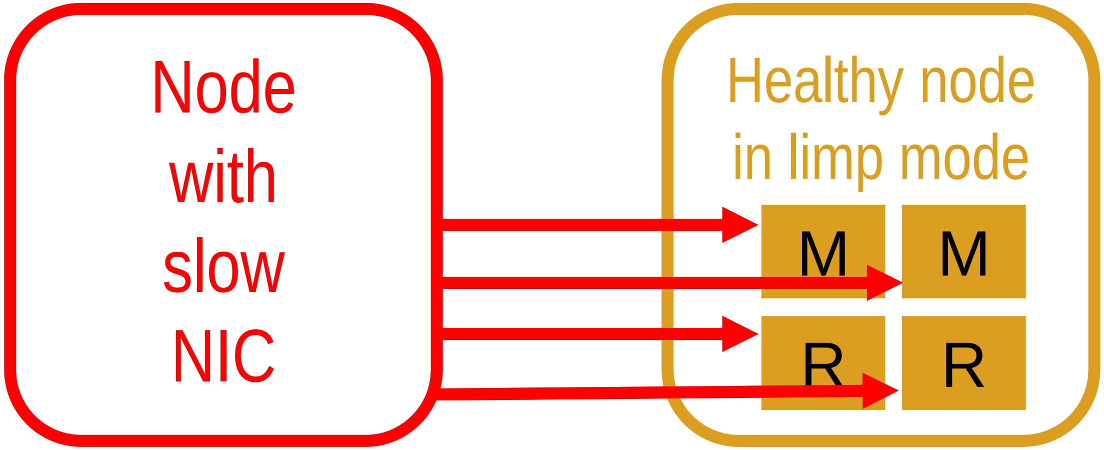
Loophole #2

- All reducers fetch from a mapper with a slow NIC
  - **All** reducers slow → **no** straggler
  - M2 reads data locally (**not** slow)
- *(many other loopholes in the paper)*



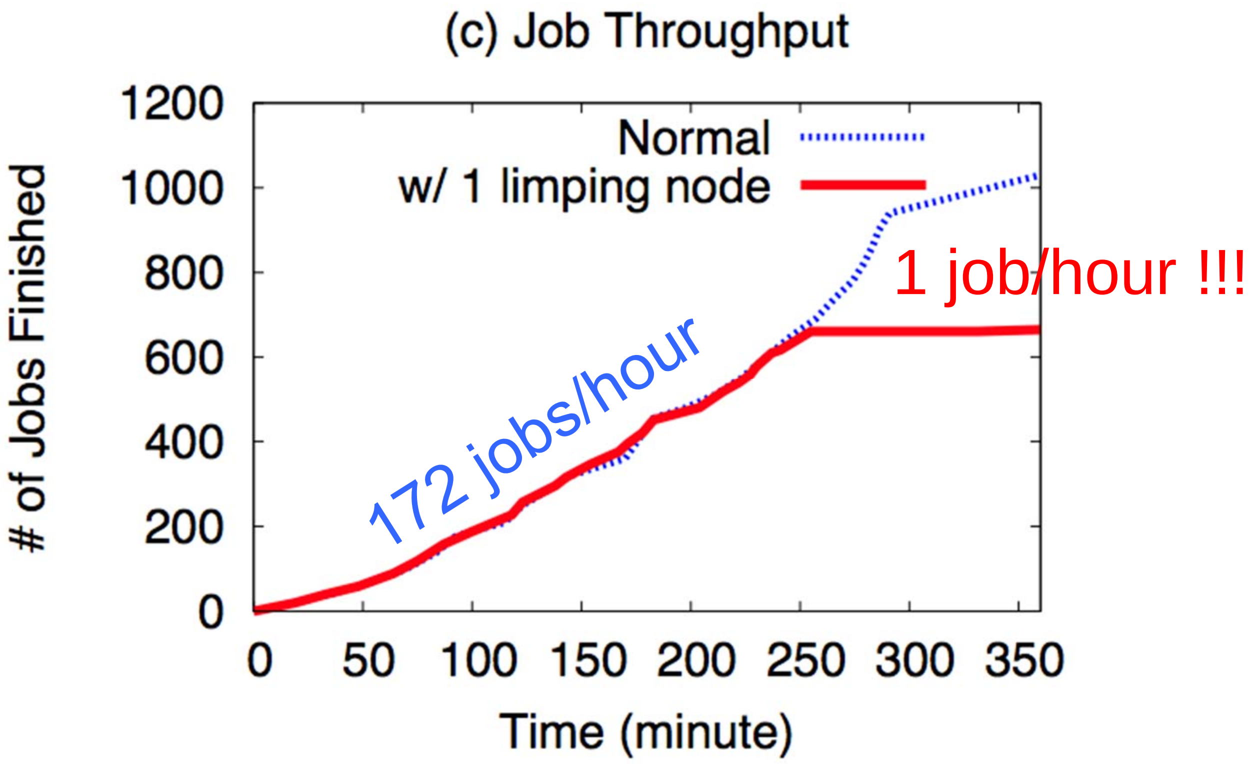
Cascading failures

- ❑ A limping NIC → limping tasks
  - (Limping tasks are slower by orders of magnitude)
- ❑ Limping tasks use up slots → limping node
  - If all slots are used → node is “unavailable”
- ❑ All nodes in limp mode → limping cluster



Cluster collapse

- Macrobenchmark: Facebook Hadoop workload
  - **30-node cluster**
  - One node w/ limping NIC (0.1 Mbps)



(c) Job Throughput

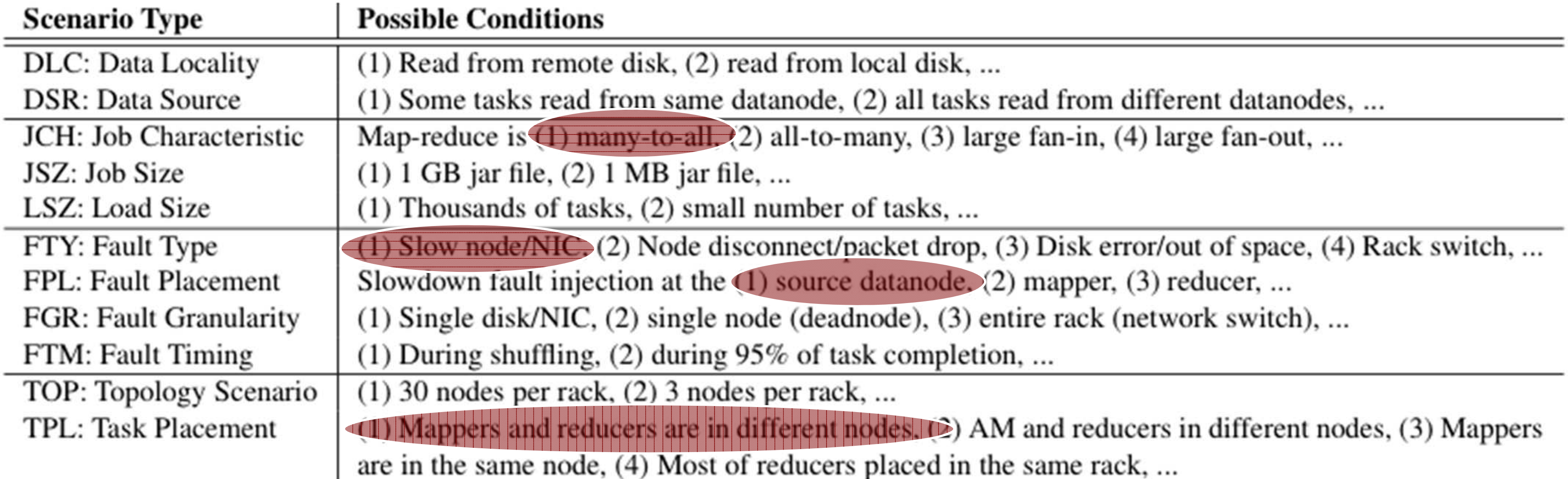
172 jobs/hour

1 job/hour !!!

Formalizing the problem

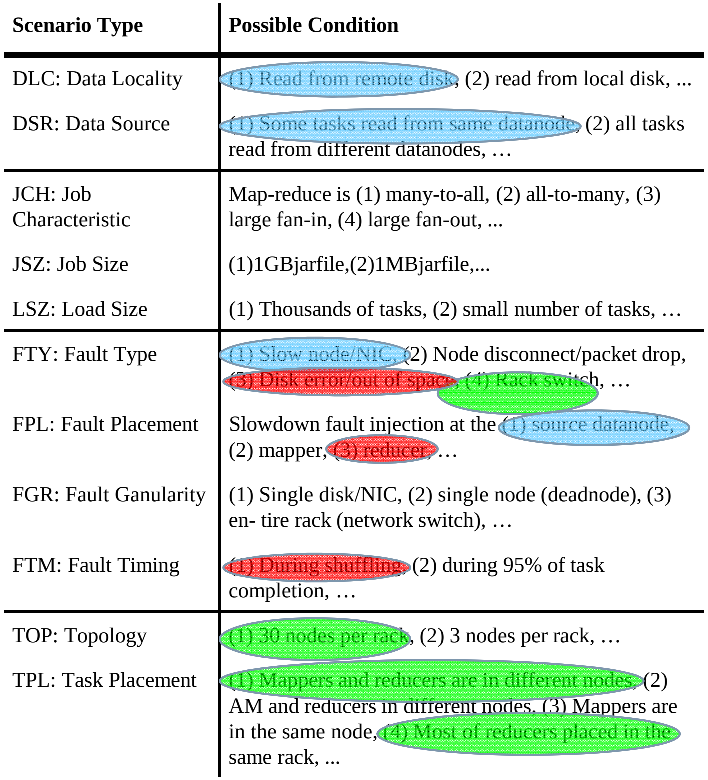
- A job = **various** deployment scenarios
- Untriggered speculative execution
  - (DSR_1 & FTY_1 & FPL_1 & DLC_1) or
  - (JCH_1 & TPL_1 & FTY_1 & FPL_2) or …

Unanticipated scenario



Scenario Type	Possible Conditions
DLC: Data Locality	(1) Read from remote disk, (2) read from local disk, ...
DSR: Data Source	(1) Some tasks read from same datanode, (2) all tasks read from different datanodes, ...
JCH: Job Characteristic	Map-reduce is (1) many-to-all, (2) all-to-many, (3) large fan-in, (4) large fan-out, ...
JSZ: Job Size	(1) 1 GB jar file, (2) 1 MB jar file, ...
LSZ: Load Size	(1) Thousands of tasks, (2) small number of tasks, ...
FTY: Fault Type	(1) Slow node/NIC, (2) Node disconnect/packet drop, (3) Disk error/out of space, (4) Rack switch, ...
FPL: Fault Placement	Slowdown fault injection at the (1) source datanode, (2) mapper, (3) reducer, ...
FGR: Fault Granularity	(1) Single disk/NIC, (2) single node (deadnode), (3) entire rack (network switch), ...
FTM: Fault Timing	(1) During shuffling, (2) during 95% of task completion, ...
TOP: Topology Scenario	(1) 30 nodes per rack, (2) 3 nodes per rack, ...
TPL: Task Placement	(1) Mappers and reducers are in different nodes, (2) AM and reducers in different nodes, (3) Mappers are in the same node, (4) Most of reducers placed in the same rack, ...

Non-deterministic performance bugs



Scenario Type	Possible Condition
DLC: Data Locality	(1) Read from remote disk, (2) read from local disk, ...
DSR: Data Source	(1) Some tasks read from same datanode, (2) all tasks read from different datanodes, …
JCH: Job Characteristic	Map-reduce is (1) many-to-all, (2) all-to-many, (3) large fan-in, (4) large fan-out, ...
JSZ: Job Size	(1)1GBjarfile,(2)1MBjarfile,...
LSZ: Load Size	(1) Thousands of tasks, (2) small number of tasks, …
FTY: Fault Type	(1) Slow node/NIC, (2) Node disconnect/packet drop, (3) Disk error/out of space, (4) Rack switch, …
FPL: Fault Placement	Slowdown fault injection at the (1) source datanode, (2) mapper, (3) reducer, …
FGR: Fault Ganularity	(1) Single disk/NIC, (2) single node (deadnode), (3) en- tire rack (network switch), …
FTM: Fault Timing	(1) During shuffling, (2) during 95% of task completion, …
TOP: Topology	(1) 30 nodes per rack, (2) 3 nodes per rack, …
TPL: Task Placement	(1) Mappers and reducers are in different nodes, (2) AM and reducers in different nodes, (3) Mappers are in the same node, (4) Most of reducers placed in the same rack, ...

- Untriggered Speculative Execution
  - MR-70001 = JCH$_1$ & TPL$_1$ & FPL$_2$ & FTY$_1$
  - MR-70002 = DSR$_1$ & DLC$_1$ & FPL$_1$ & FTY$_1$
  - MR-5533 = FTY$_2$ & FPL$_3$ & TPL$_3$
  - …
- O(n) Recovery
  - MR-5251 = FTY$_3$ & FPL$_3$ & FTM$_1$
  - MR-5060 = TPL$_1$ & TPL$_3$ & FTY$_1$ & FPL$_2$
  - MR-1800 = TPL$_1$ & TPL$_4$ & FTY$_4$ & TOP$_1$
  - …
- Long lock contention
  - MR-9191 = FTY$_3$ & FPL$_3$ & FTM$_1$
  - MR-9292 = TPL$_1$ & TPL$_3$ & FTY$_1$ & FPL$_2$
  - MR-9393 = TPL$_1$ & TPL$_4$ & FTY$_4$ & TOP$_1$
  - …

Perf. Model Checking [HotCloud '15]

- **Goal**: Permute many topological/failure/placement scenarios
- Real Java code → Colored Petri Nets (CPN) model
  - Automated conversion ("compiler")
  - Abstract system-level constructs
    - E.g., queues, tasks, resources, locks
- Permute the scenarios in CPN
- Abstract performance faults
  - Boolean result: limping or not
  - No need for precise latency/bandwidth predictions
- Test the buggy scenarios in real runs

Path Based Spec. Exec. [In Subm.]

- Hadoop SE:
  - **Straggler**: if task **T**'s progress is **slower** than the rest
  - Task T is just a progress score → fundamental flaw
- Our observation:
  - Task T is a **path**
  - **Map path**: source datanode → map node
  - **Shuffle path**: map node → reduce node
  - **Output path**: reduce node → pipeline of datanodes
- **PBSE:** Path-based speculative execution
  - It's about the progress of individual "paths"
  - SE algorithm is based on path progress
  - Diverse paths: no single point of path failure

Conclusion

- Distributed concurrency bugs
- Non-deterministic performance bugs
- Scalability bugs
- Other outage-causing bugs:
  - SPOF/cascading bugs
  - Cross-layer upgrade bugs

The complexity of cloud-scale hardware and software ecosystem has ***outpaced*** *existing testing, debugging, and verification tools.*

Many new classes of bugs to hunt!

Thank you!
Questions?

UCARE

CERES
Center for Unstoppable Computing

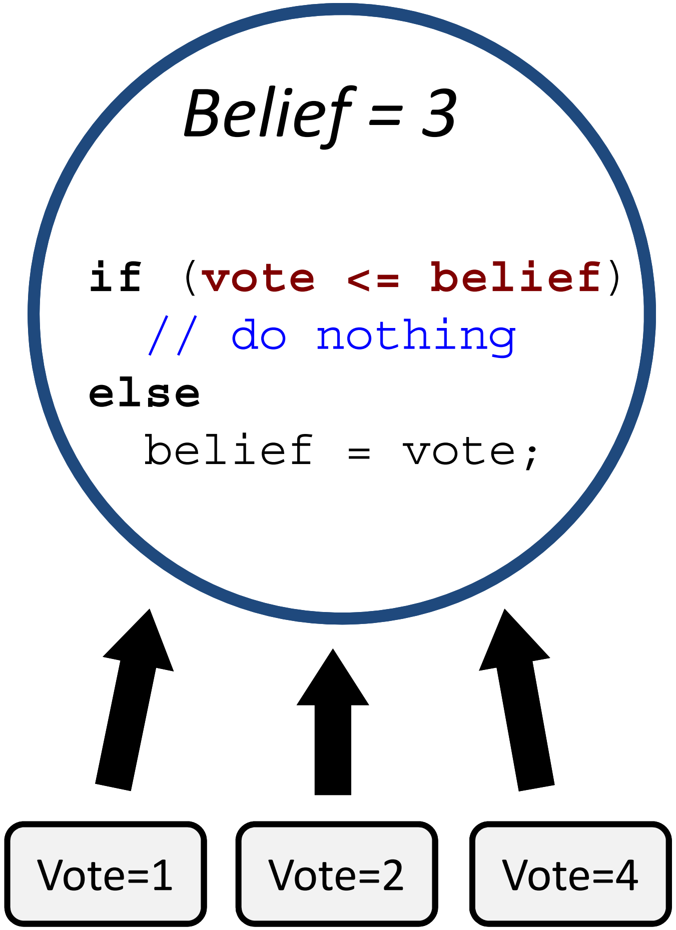
ucare.cs.uchicago.edu

ceres.cs.uchicago.edu

EXTRA

Extra -- SAMC

Message Processing Semantic in a Leader Election



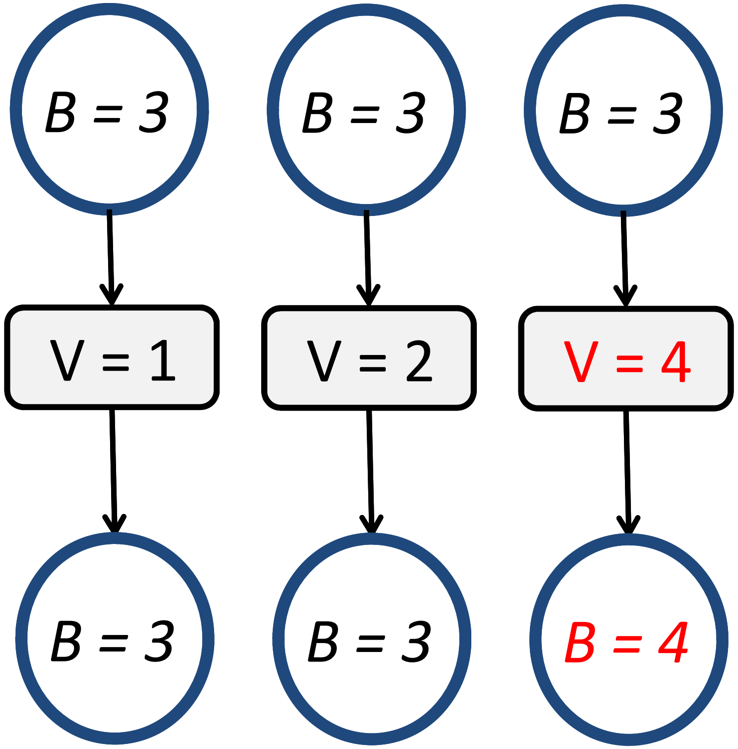
Belief = 3

```
if (vote <= belief)
   // do nothing
else
   belief = vote;
```

Vote=1

Vote=2

Vote=4



B = 3

B = 3

B = 3

V = 1

V = 2

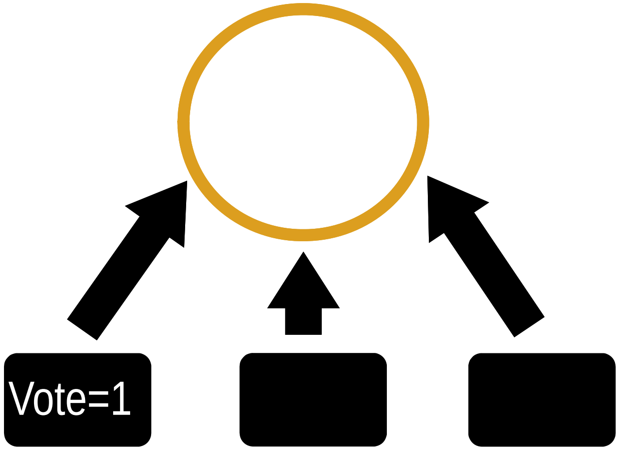
V = 4

B = 3

B = 3

B = 4

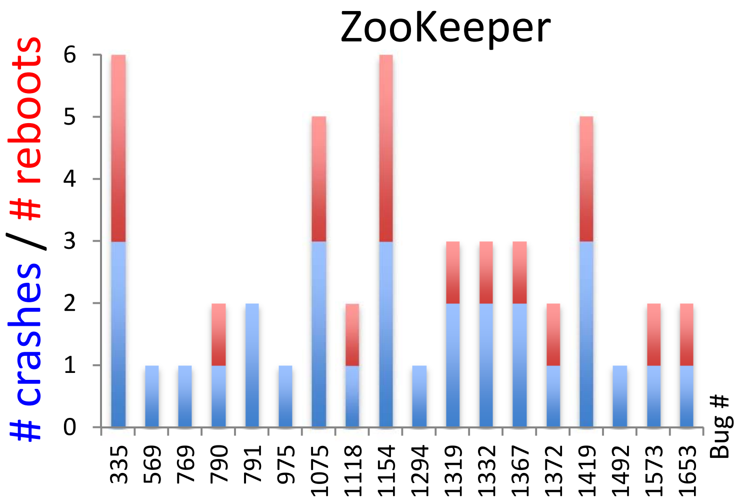
SAMC server logic (extra)

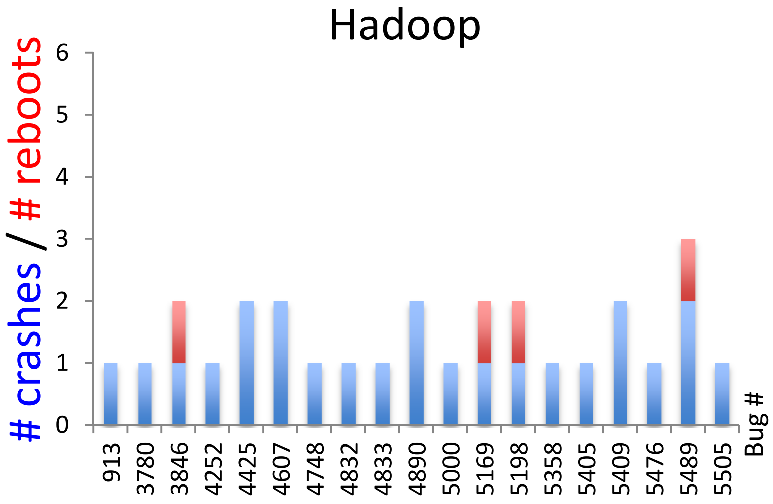


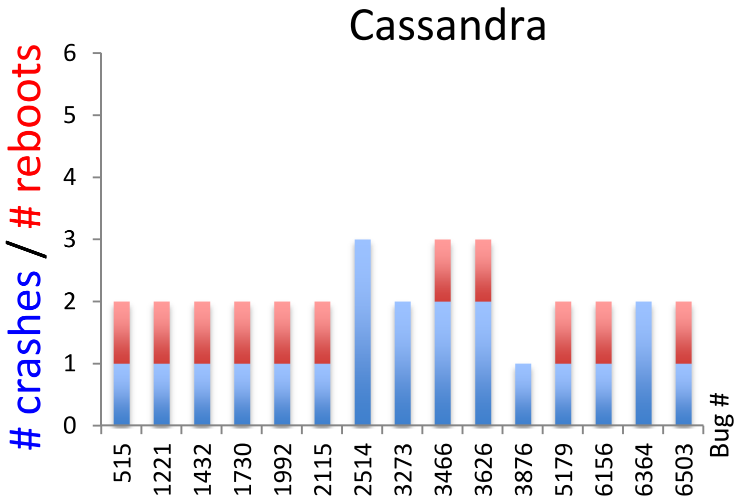
vote	belief	isDiscard
1	3	true
2	3	true
4	3	false

m_x	m_y	discard(m_x)	discard(m_y)	Independent
1	2	true	true	✔
1	4	true	false	x
2	4	true	false	x

+ Crashes and Reboots (sometimes multiple of them)

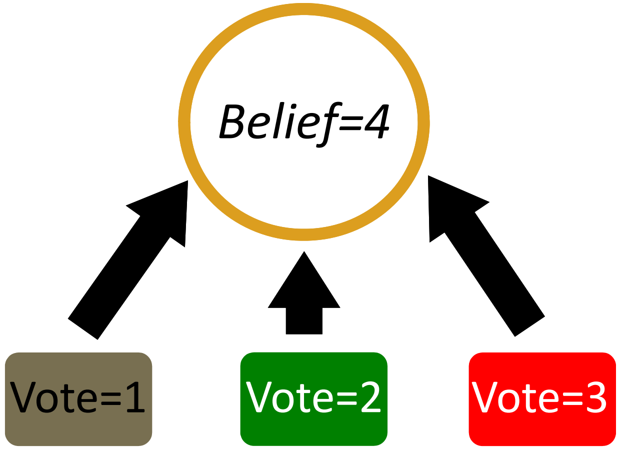




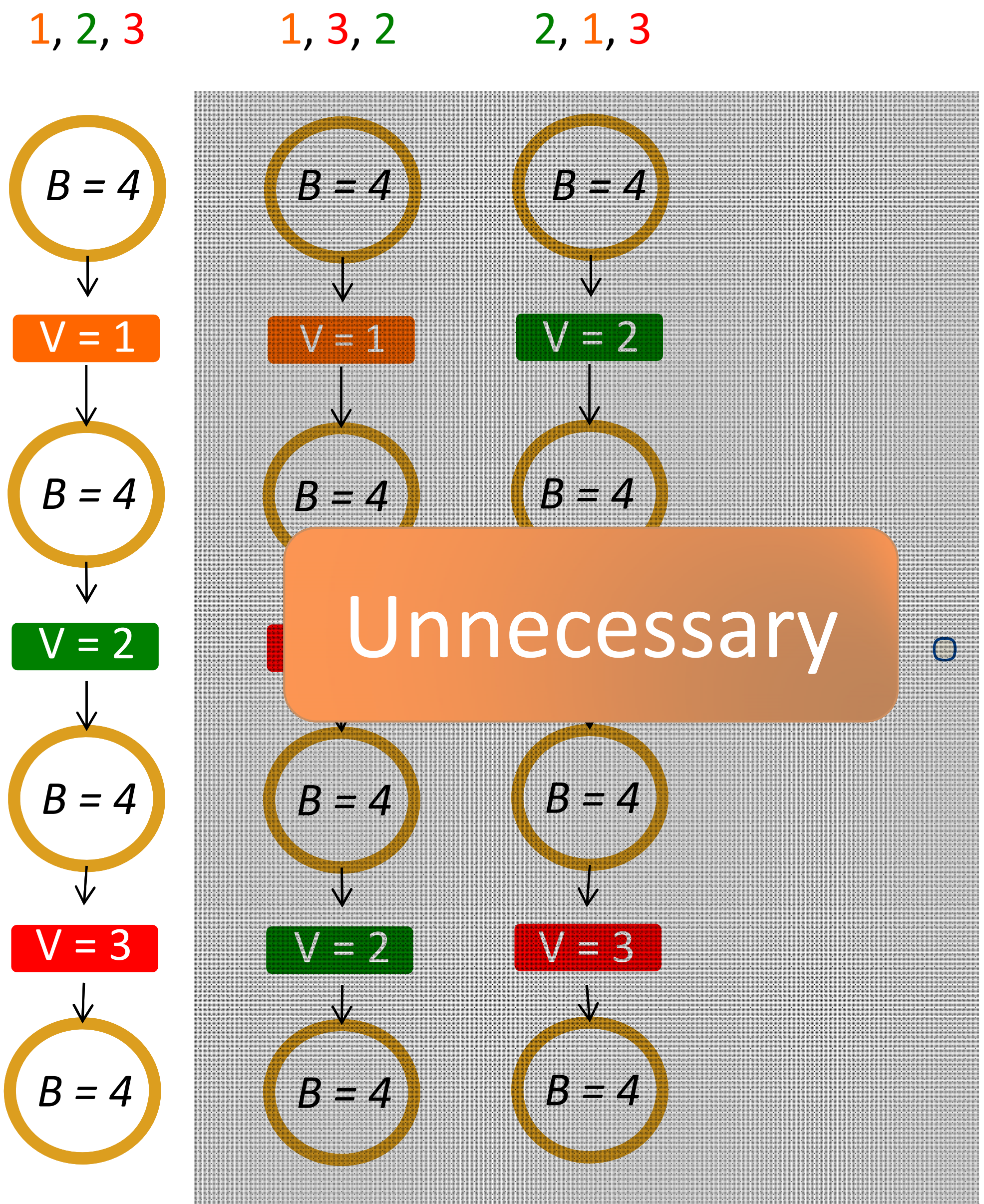


x-axis is bug number
y-axis is number of crashes
and reboots

Removing Re-orderings via Message Processing Semantic



```
if (vote <= belief)
    // do nothing
else
    belief = vote;
```



Unnecessary

Errors, Faults, Failure

- To quote the Software Engineering Body of Knowledge
- Different cultures and standards may use somewhat different meanings for these terms, which have led to attempts to define them.
- Partial definitions taken from standard (IEEE610.12-90) are:
- Error: “A difference…between a computed result and the correct result”
- Fault: “An incorrect step, process, or data definition in a computer program”
- Failure: “The [incorrect] result of a fault”
- Mistake: “A human action that produces an incorrect result”