Correlated Crash Vulnerabilities

Ramnatthan Alagappan, Aishwarya Ganesan, Yuvraj Patel, Thanumalayan Sankaranarayana Pillai, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Distributed Storage Systems

Central to building modern services

Distributed Storage Systems

Central to building modern services

Distributed Storage Systems

Central to building modern services

Reliability by Replication

Reliability of user data is important

Reliability by Replication

Reliability of user data is important

Core mechanism: Replication

Reliability by Replication

Reliability of user data is important

Core mechanism: Replication

Reliability by Replication

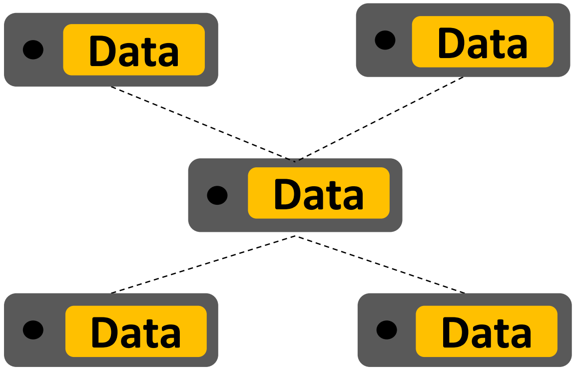
Reliability of user data is important

Core mechanism: Replication

Reliability by Replication

Reliability of user data is important

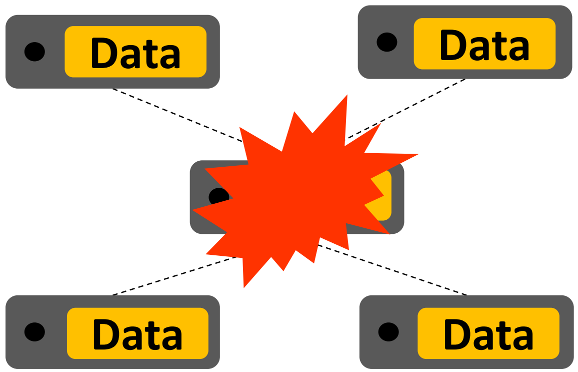
Core mechanism: Replication



Reliability by Replication

Reliability of user data is important

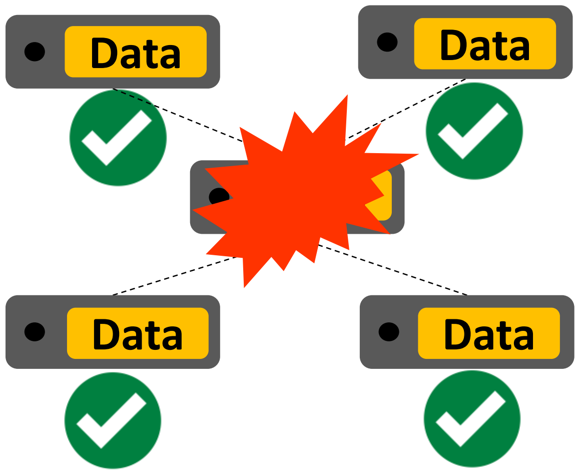
Core mechanism: Replication



Reliability by Replication

Reliability of user data is important

Core mechanism: Replication



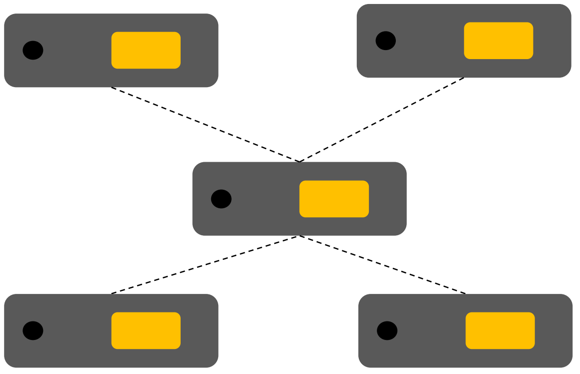
Distributed storage systems can endure single machine crashes

Our Focus: Correlated Crashes

All data replicas crash and recover together

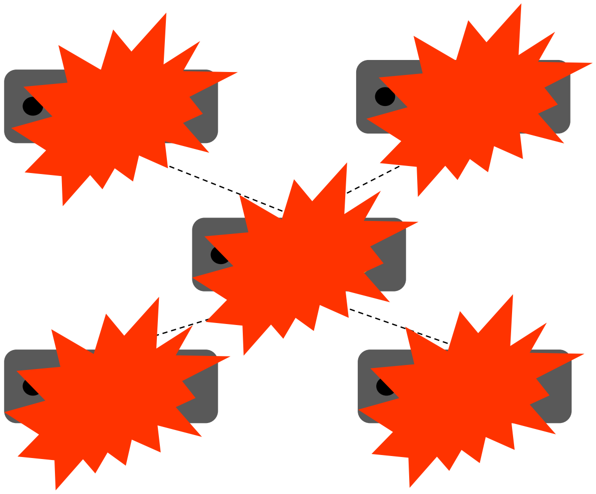
Our Focus: Correlated Crashes

All data replicas crash and recover together



Our Focus: Correlated Crashes

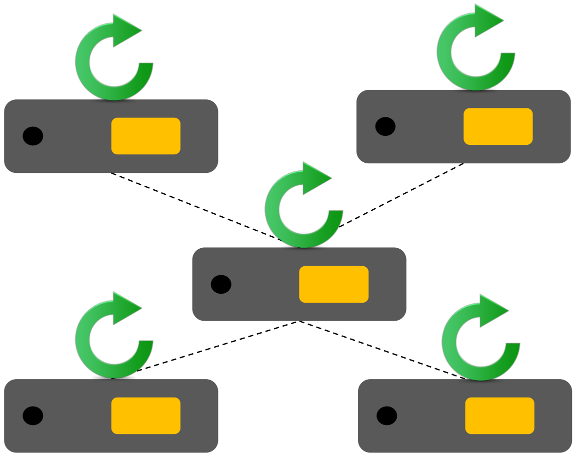
All data replicas crash and recover together



Obviously, unavailable during failure

Our Focus: Correlated Crashes

All data replicas crash and recover together



Obviously, unavailable during failure

However, correct data should be available after recovery

Our Focus: Correlated Crashes

How often do they happen?

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes utility grid affecting Google data centers

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes utility grid affecting Google data centers

Fat-fingered admin downs entire Joyent data center

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes utility grid affecting Google data centers

Fat-fingered admin downs entire Joyent data center

Truck ploughs into power cable at Rackspace

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes utility grid affecting Google data centers

Fat-fingered admin downs entire Joyent data center

Truck ploughs into power cable at Rackspace

Correlated reboots (kernel bugs, power outage, upgrades) - Ford et al., OSDI’ 10

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes utility grid affecting Google data centers

Fat-fingered admin downs entire Joyent data center

Truck ploughs into power cable at Rackspace

Correlated reboots (kernel bugs, power outage, upgrades) - Ford et al., OSDI' 10

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes affect Microsoft and Amazon data centers

Lightning strikes utility grid affecting Google data centers

Fat-fingered admin downs entire Joyent data center

Truck ploughs into power cable at Rackspace

Correlated reboots (kernel bugs, power outage, upgrades) - Ford et al., OSDI' 10

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes affect Microsoft and Amazon data centers

Catastrophic Storage Failure Slows Oregon Jobless Checks

Fat-fingered admin downs entire Joyent data center

Truck ploughs into power cable at Rackspace

Correlated reboots (kernel bugs, power outage, upgrades) - Ford et al., OSDI' 10

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes affect Microsoft and Amazon data centers

Catastrophic Storage Failure Slows Oregon Jobless Checks

Singapore Financial Exchange goes down …

Correlated reboots (kernel bugs, power outage, upgrades) - Ford et al., OSDI' 10

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes affect Microsoft and Amazon data centers

Catastrophic Storage Failure Slows Oregon Jobless Checks

Singapore Financial Exchange goes down …

Bank of Scotland and Halifax customers not able to access money

Our Focus: Correlated Crashes

How often do they happen?

More often than we think!

Lightning strikes affect Microsoft and Amazon data centers

Catastrophic Storage Failure Slows Oregon Jobless Checks

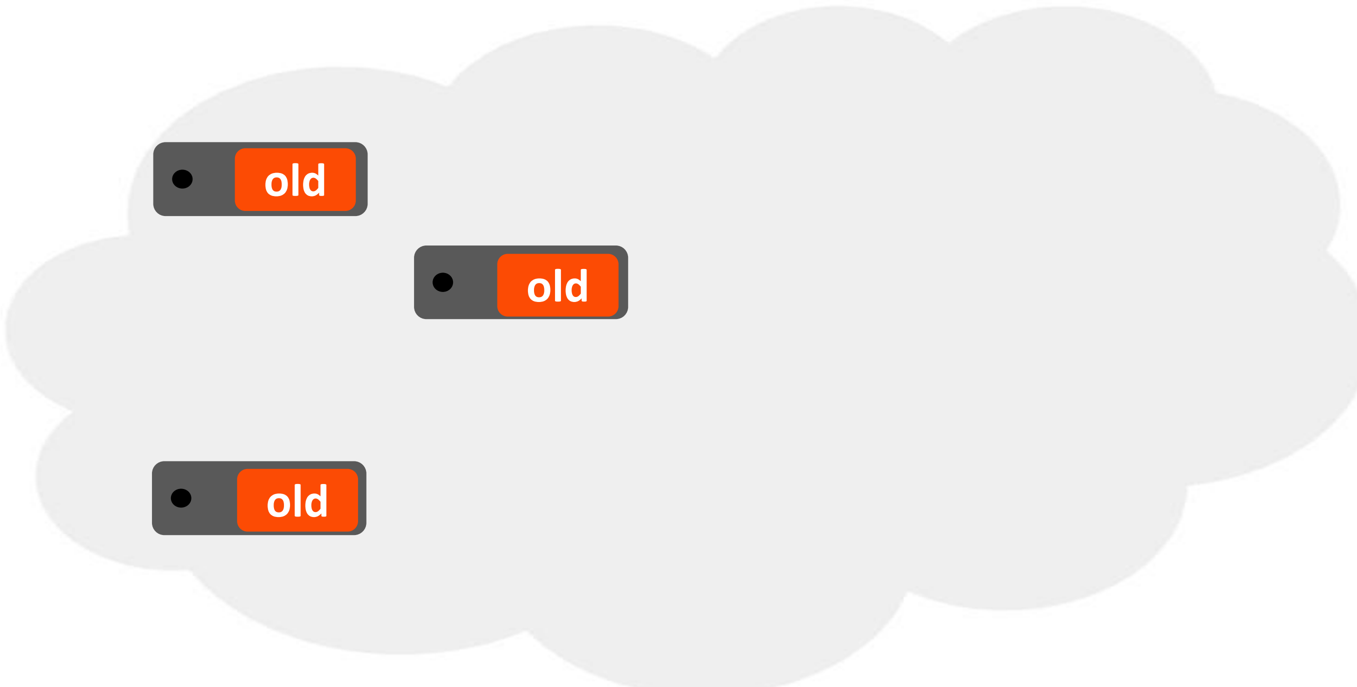
Singapore Financial Exchange goes down …

Bank of Scotland and Halifax customers not able to access money

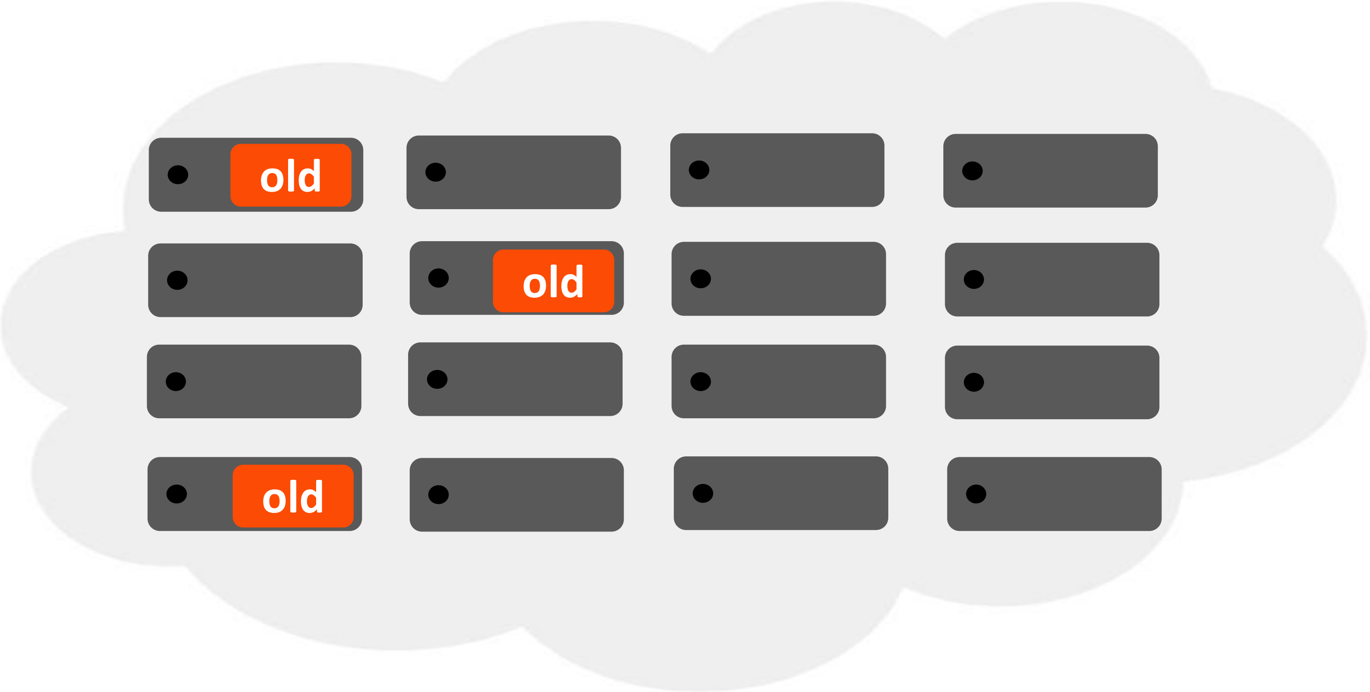
The Huffington Post, Gawker, Gizmodo, and Buzzfeed go down as data center flood …

Our Focus: Correlated Crashes

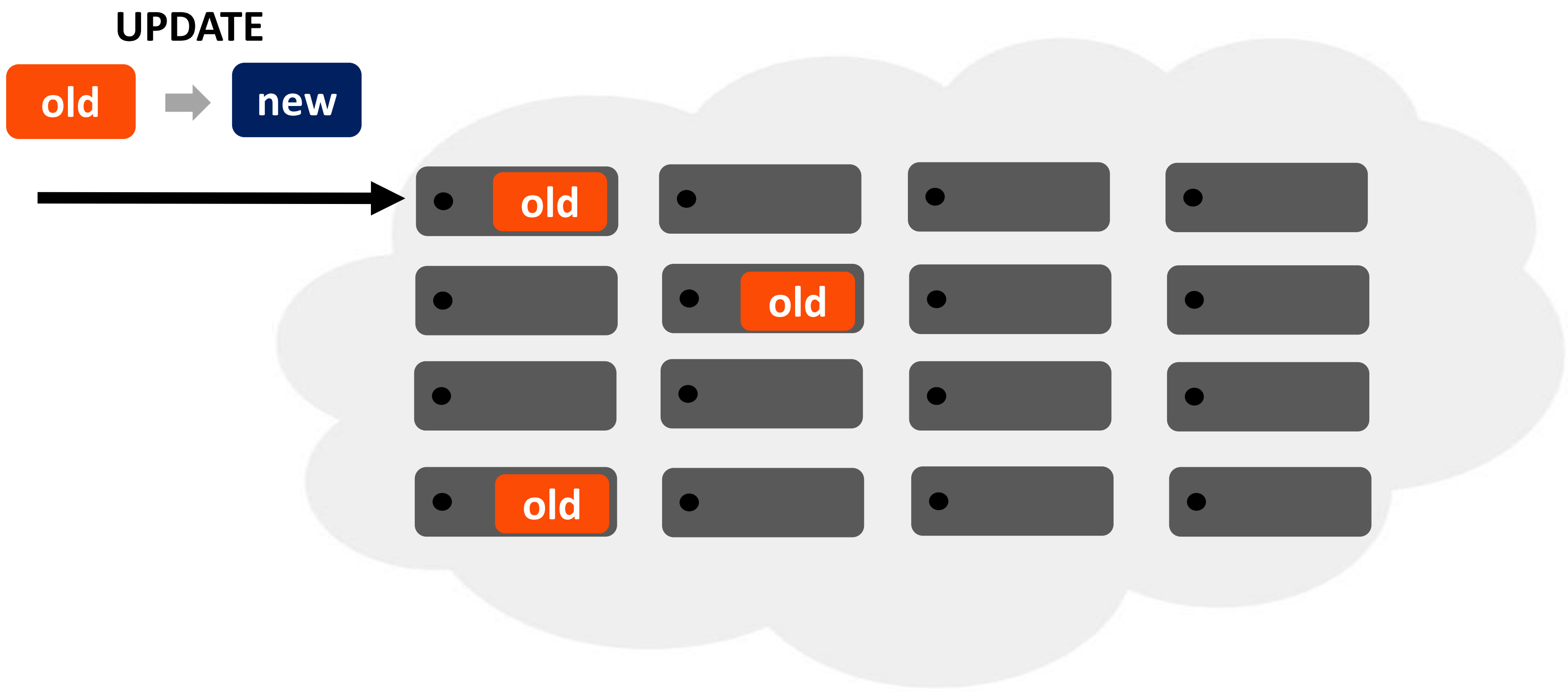
Our Focus: Correlated Crashes



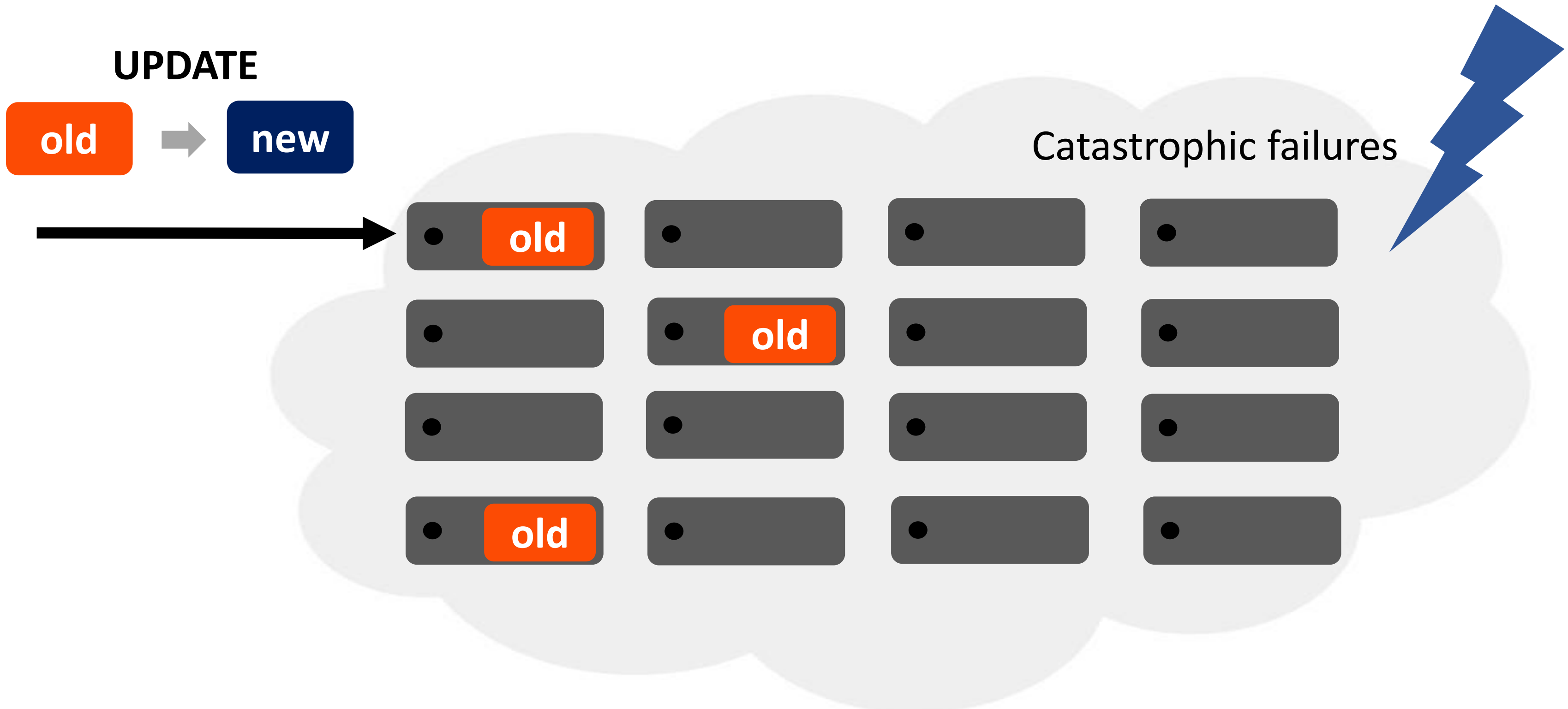
Our Focus: Correlated Crashes



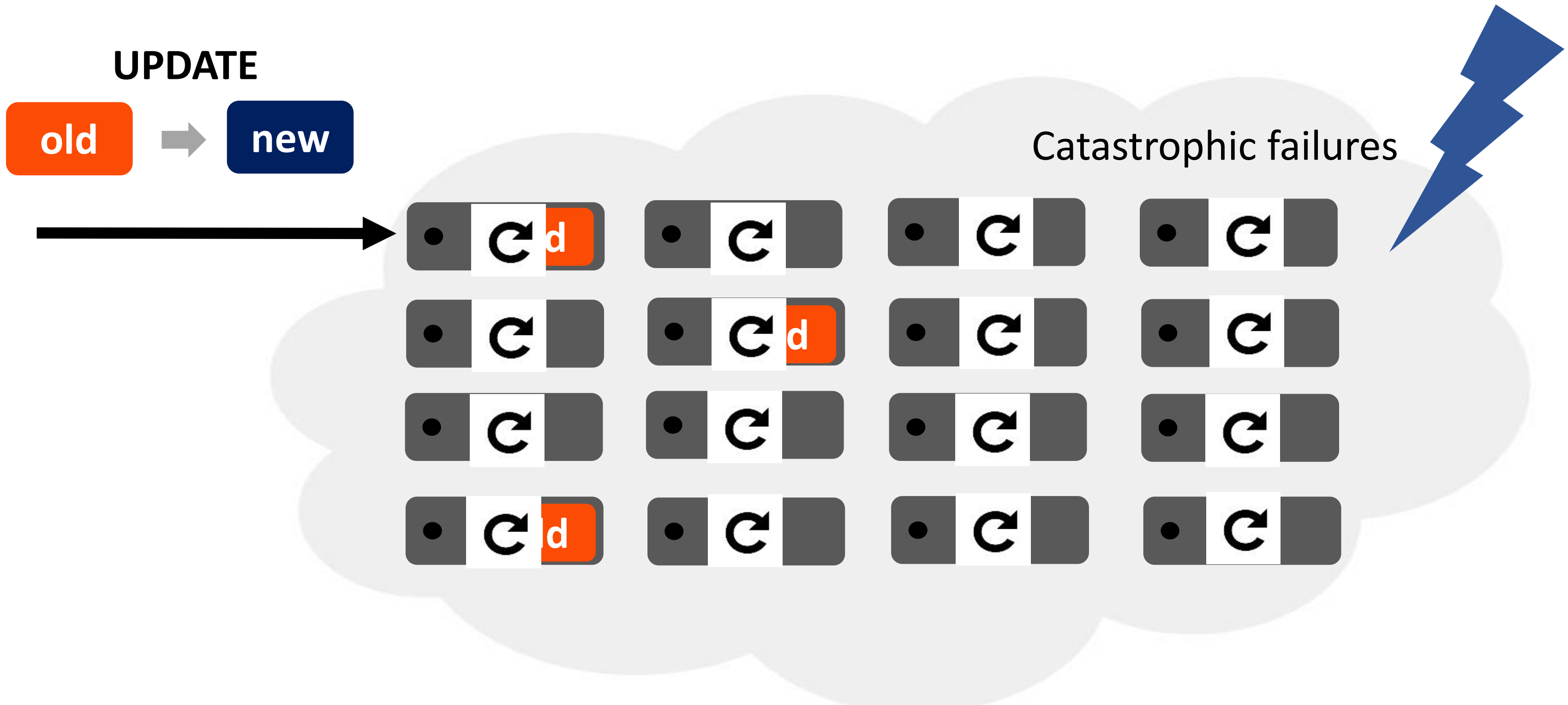
Our Focus: Correlated Crashes



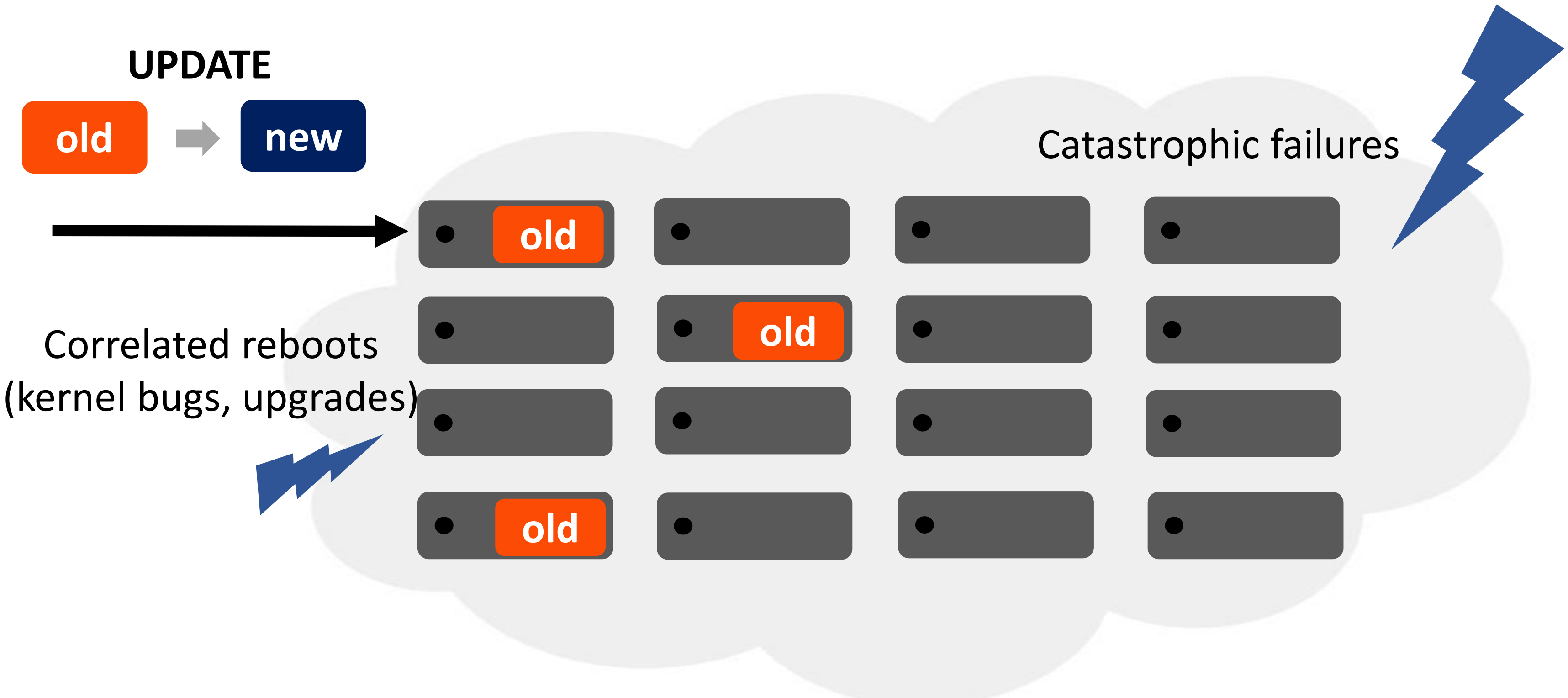
Our Focus: Correlated Crashes



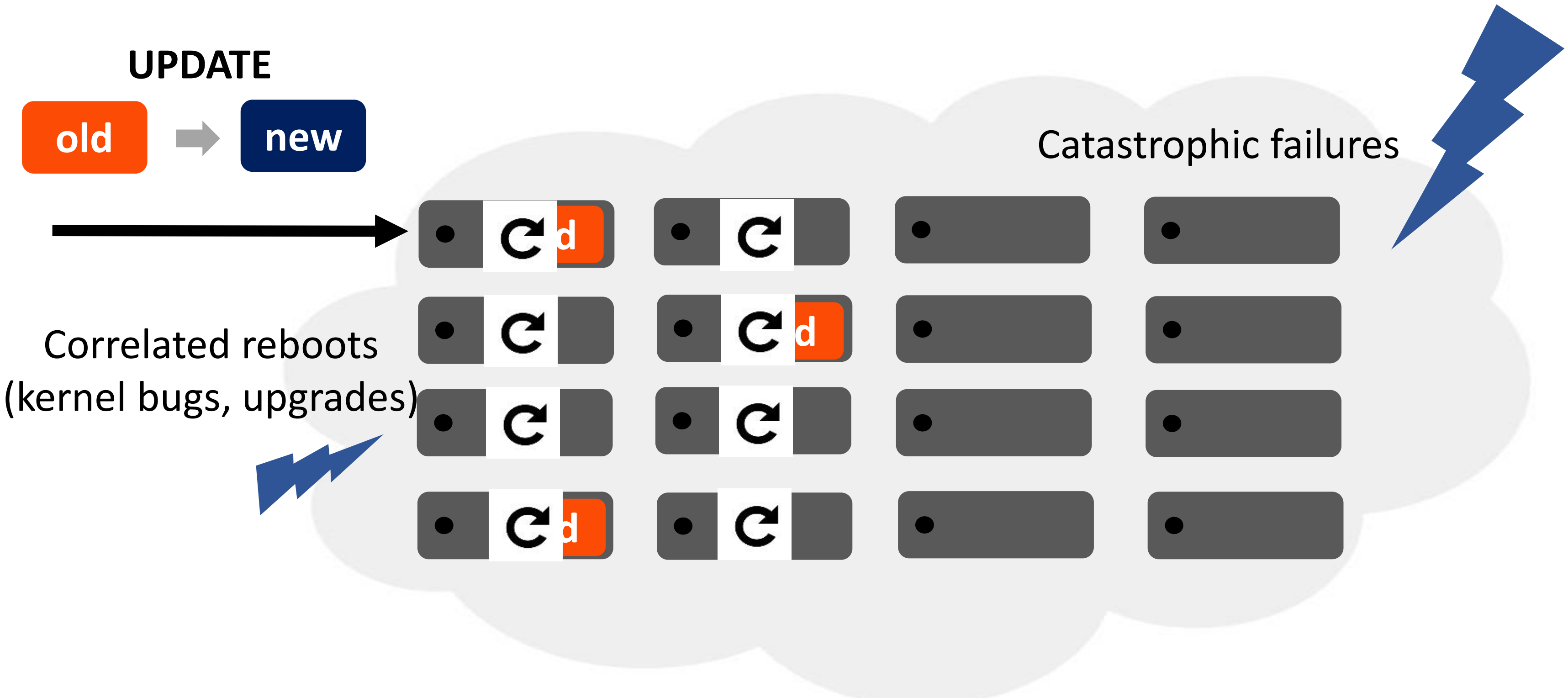
Our Focus: Correlated Crashes



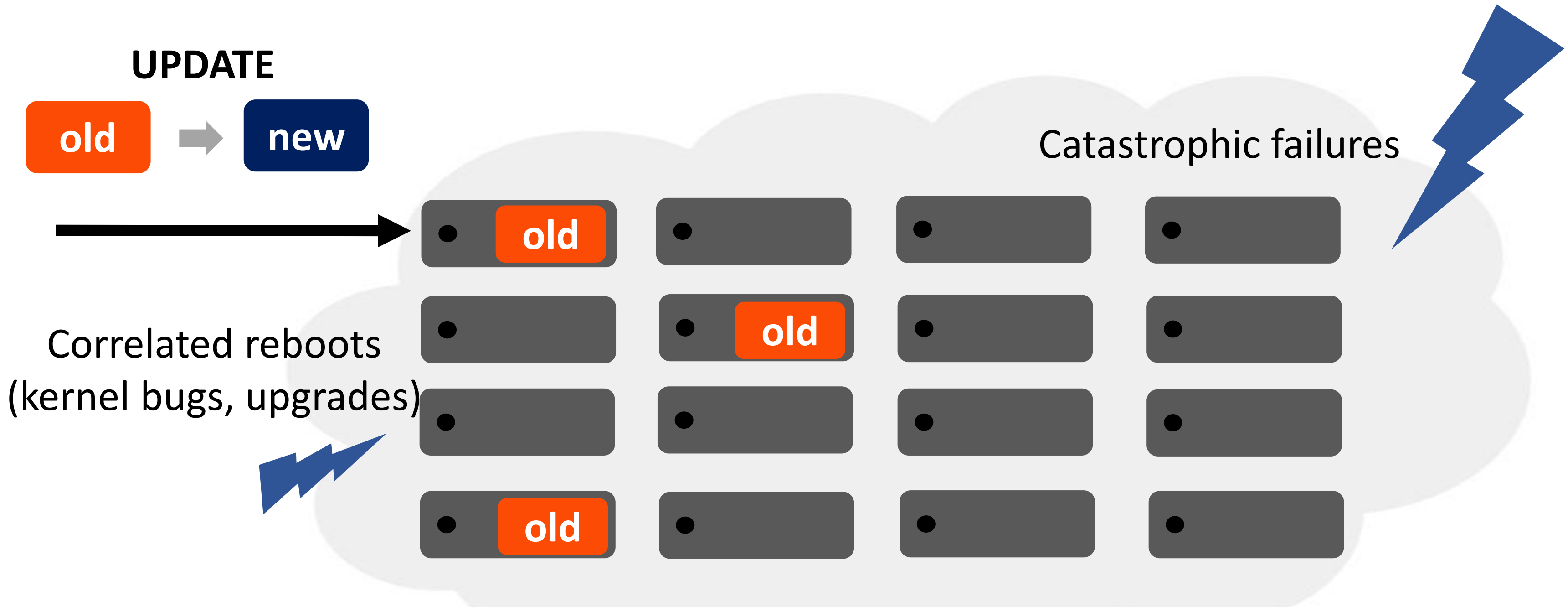
Our Focus: Correlated Crashes



Our Focus: Correlated Crashes

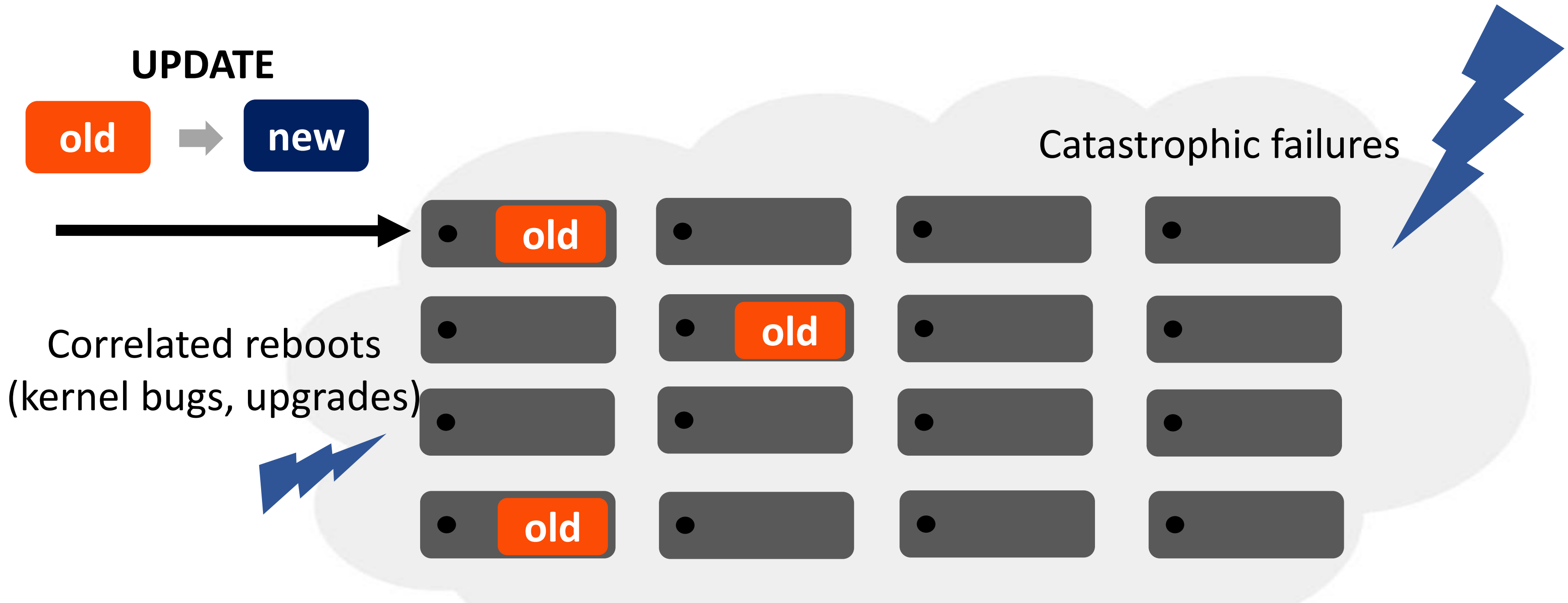


Our Focus: Correlated Crashes



In such crash scenarios, nodes left only with persistent state

Our Focus: Correlated Crashes



In such crash scenarios, nodes left only with persistent state

After recovery, users expect:

Either new (if acknowledged) or old

No corrupted data

Available after recovery

Our Focus: Correlated Crashes

UPDATE

old

res

Correlat
(kernel bu

In such c ate

After rec

Either new (if acknowledged) or old

No corrupted data

Available after recovery

File Systems Complicate Recovery

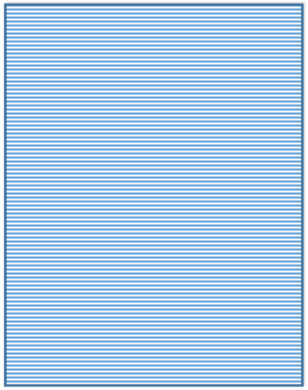
File Systems Complicate Recovery

Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI’14, Bornholt et al., ASPLOS’16]

File Systems Complicate Recovery

Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI'14, Bornholt et al., ASPLOS'16]

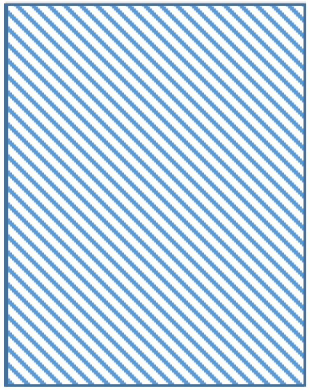
/foo



File Systems Complicate Recovery

Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI'14, Bornholt et al., ASPLOS'16]

/foo

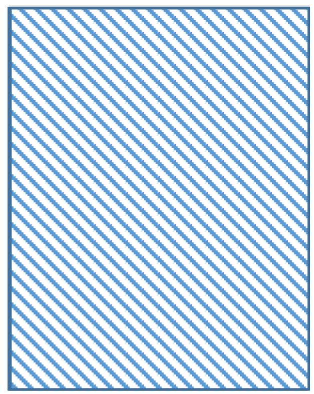


File Systems Complicate Recovery

Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI’14, Bornholt et al., ASPLOS’16]

/foo

pwrite(“/foo”, buf, 4K, 0)



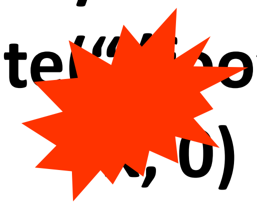
File Systems Complicate Recovery

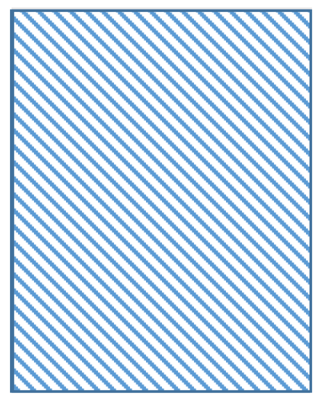
Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI’14, Bornholt et al., ASPLOS’16]

/foo

pwrite(“/ oo”, buf,

K, 0)



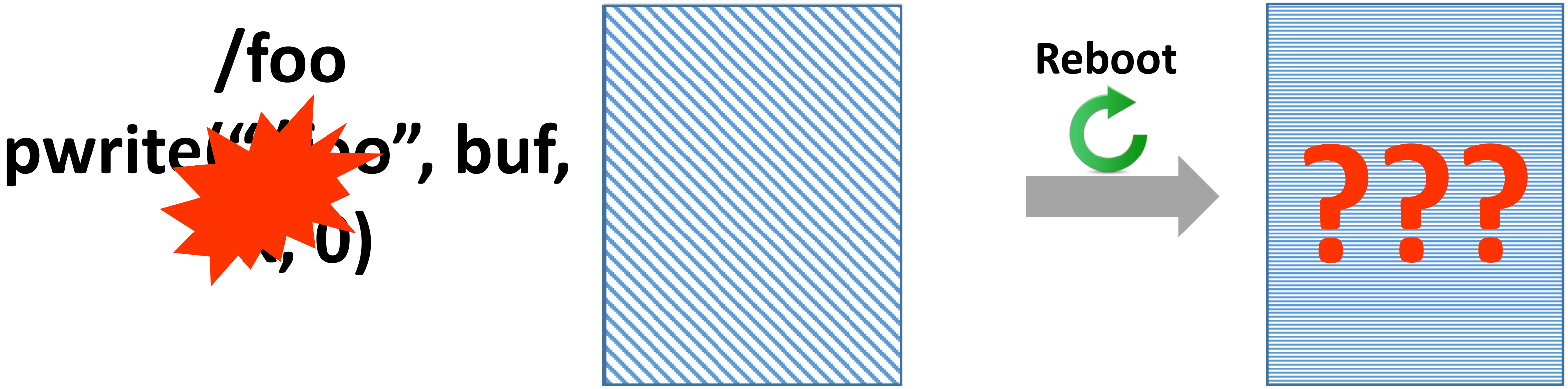


File Systems Complicate Recovery

Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI’14, Bornholt et al., ASPLOS’16]

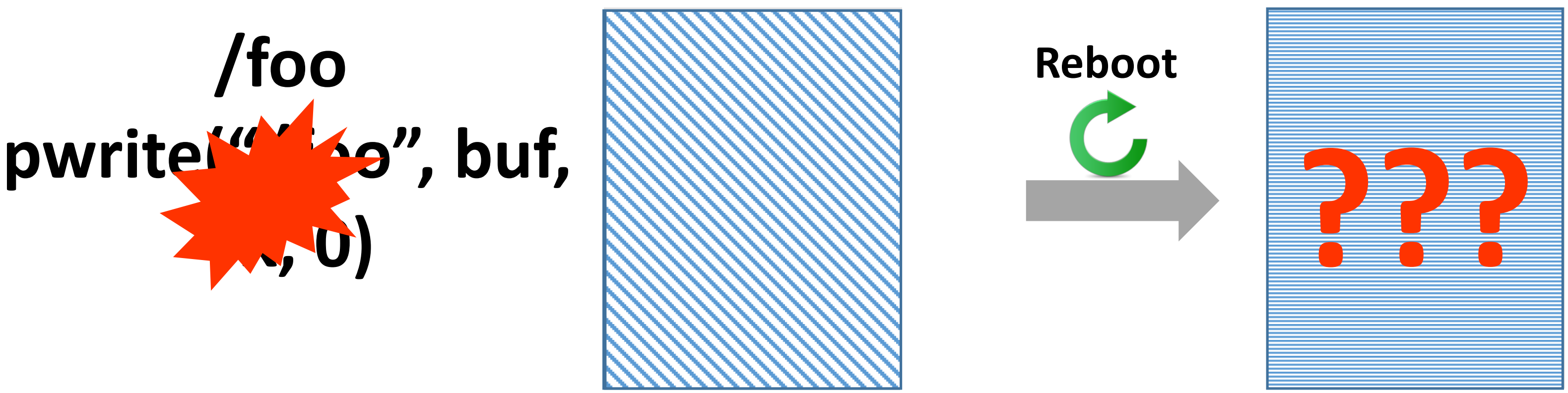
File Systems Complicate Recovery

Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI’14, Bornholt et al., ASPLOS’16]



File Systems Complicate Recovery

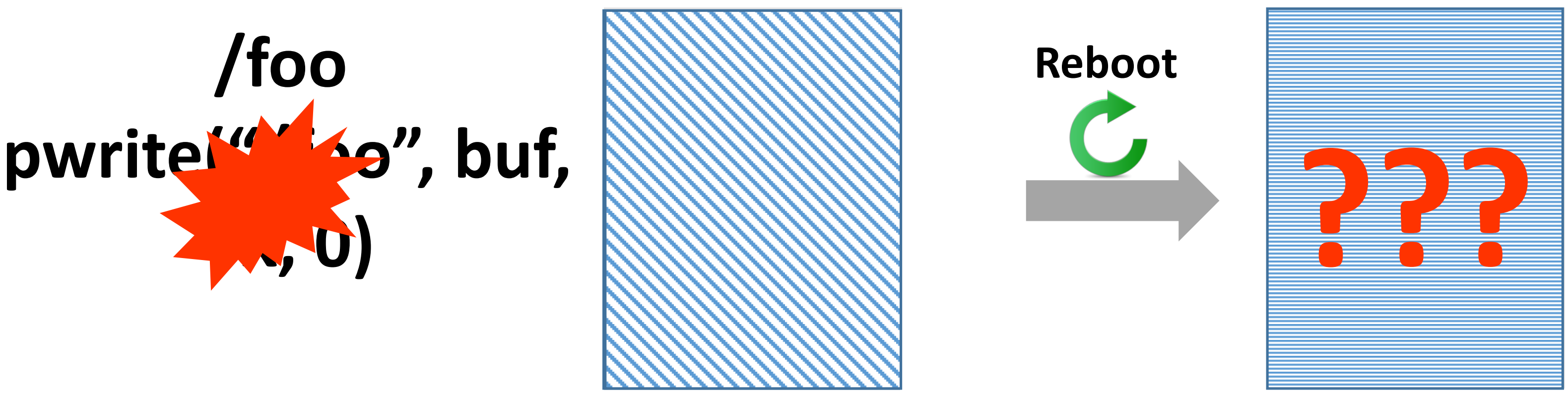
Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI’14, Bornholt et al., ASPLOS’16]



Can we expect the block to be atomically updated?

File Systems Complicate Recovery

Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI’14, Bornholt et al., ASPLOS’16]

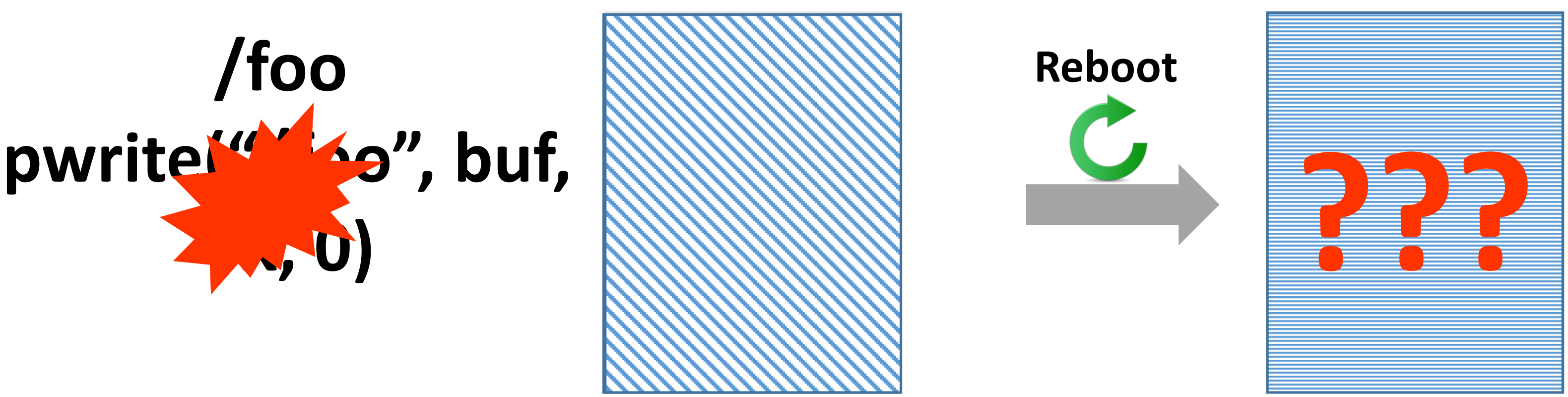


Can we expect the block to be atomically updated?

- On ext3 and ext4 (data journaling), btrfs

File Systems Complicate Recovery

Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI’14, Bornholt et al., ASPLOS’16]

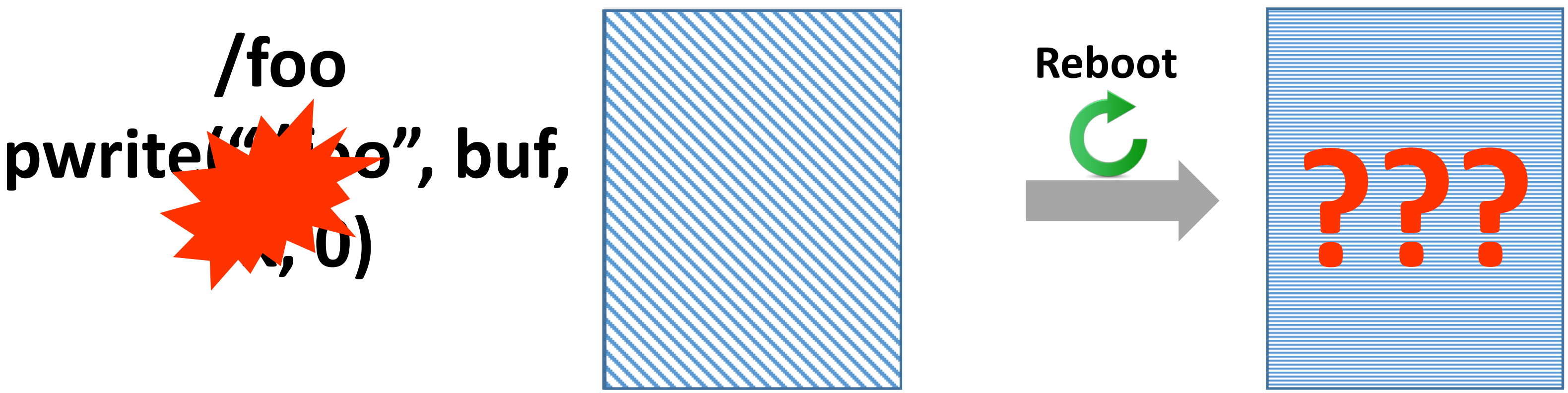


Can we expect the block to be atomically updated?

- On ext3 and ext4 (data journaling), btrfs
- On ext3 and ext4 (default), XFS, ext2 ***NO!***

File Systems Complicate Recovery

Local file systems influence persistent states that can occur after a crash [Pillai et al., OSDI’14, Bornholt et al., ASPLOS’16]



Can we expect the block to be atomically updated?

- On ext3 and ext4 (data journaling), btrfs
- On ext3 and ext4 (default), XFS, ext2 ***NO!***

Affects distributed storage systems

Focus of this study …

Focus of this study …

Do distributed storage systems violate user-level expectations in the presence of correlated crashes?

- How do distributed crash recovery protocols interact with file-system crash behaviors?

Focus of this study …

Do distributed storage systems violate user-level expectations in the presence of correlated crashes?

- How do distributed crash recovery protocols interact with file-system crash behaviors?

How to check for correlated crash vulnerabilities?

- PACE (Protocol-Aware Crash Explorer)
- Prunes state space using protocol knowledge

Results Summary

Results Summary

Applied PACE to eight systems: Redis, MongoDB, Kafka, ZooKeeper, RethinkDB, LogCabin, etcd, and iNexus

Results Summary

Applied PACE to eight systems: Redis, MongoDB, Kafka, ZooKeeper, RethinkDB, LogCabin, etcd, and iNexus

Found **26** unique vulnerabilities with severe consequences - data loss, corruption, unavailable clusters etc.,

- Many on commonly used file systems

Results Summary

Applied PACE to eight systems: Redis, MongoDB, Kafka, ZooKeeper, RethinkDB, LogCabin, etcd, and iNexus

Found **26** unique vulnerabilities with severe consequences - data loss, corruption, unavailable clusters etc.,

- Many on commonly used file systems

Many confirmed by developers

- Many fixed
- Some fundamentally hard

Overarching lessons

1. File-system crash behaviors impact distributed storage systems

2. Problems in local storage protocols are not fixed by distributed recovery protocols (in many cases)
   - False sense of reliability

Outline

Introduction and Motivation

Correlated Crash States

- **General Approach: Reachable States**
- **States due to File System Behaviors**

Protocol-Aware Crash Explorer

Vulnerability Study

Conclusion

How to capture persistent states that can occur during a correlated crash?

Approach: Finding Reachable States

Approach: Finding Reachable States

Node P ***Node Q***

Approach: Finding Reachable States

Node P ***Node Q***

 # State $P_{\emptyset}$ *# State $Q_{\emptyset}$*

Approach: Finding Reachable States

Node P

 # State $P_{\emptyset}$

send(Q, M1)

Node Q

 # State $Q_{\emptyset}$

recv(M1)

Approach: Finding Reachable States

Node P	***Node Q***
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
send(Q, M1)	
	recv(M1)
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*

Approach: Finding Reachable States

Node P	***Node Q***
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
send(Q, M1)	
	recv(M1)
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
write(fd, “foo”,3)	

Approach: Finding Reachable States

Node P

 # State $P_{\emptyset}$

send(Q, M1)

 # State $P_{\emptyset}$

write(fd, “foo”,3)

 # State P_1

Node Q

 # State $Q_{\emptyset}$

recv(M1)

 # State $Q_{\emptyset}$

Approach: Finding Reachable States

Node P	***Node Q***
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
send(Q, M1)	
	recv(M1)
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
write(fd, “foo”,3)	
# State P_1	
write(fd, “baz”,3)	
# State P_2	

Approach: Finding Reachable States

Node P	***Node Q***
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
send(Q, M1)	recv(M1)
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
write(fd, “foo”,3)	
# State P_1	
write(fd, “baz”,3)	
# State P_2	
send(Q, M2)	recv(M2)
# State P_2	*# State $Q_{\emptyset}$*

Approach: Finding Reachable States

Node P	***Node Q***
# State $P_ø$	*# State $Q_ø$*
send(Q, M1)	recv(M1)
# State $P_ø$	*# State $Q_ø$*
write(fd, “foo”,3)	
# State P_1	
write(fd, “baz”,3)	
# State P_2	
send(Q, M2)	recv(M2)
# State P_2	*# State $Q_ø$*
	write(fd, “bar”, 3)

Approach: Finding Reachable States

Node P	***Node Q***
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*
send(Q, M1)	recv(M1)
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*
write(fd, “foo”,3)	
# State P_1	
write(fd, “baz”,3)	
# State P_2	
send(Q, M2)	recv(M2)
# State P_2	*# State $Q_{\varnothing}$*
	write(fd, “bar”, 3)
	# State Q_1

Approach: Finding Reachable States

Node P	***Node Q***
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
send(Q, M1)	recv(M1)
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
write(fd, “foo”,3)	
# State P_1	
write(fd, “baz”,3)	
# State P_2	
send(Q, M2)	recv(M2)
# State P_2	*# State $Q_{\emptyset}$*
	write(fd, “bar”, 3)
	# State Q_1

Globally **reachable** persistent states

Approach: Finding Reachable States

Node P	***Node Q***		
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*	$<\mathbf{P_{\emptyset}}, \mathbf{Q_{\emptyset}}>$	Globally **reachable** persistent states
send(Q, M1)			
	recv(M1)		
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*		
write(fd, “foo”,3)			
# State P_1			
write(fd, “baz”,3)			
# State P_2			
send(Q, M2)			
	recv(M2)		
# State P_2	*# State $Q_{\emptyset}$*		
	write(fd, “bar”, 3)		
	# State Q_1		

Approach: Finding Reachable States

Node P	***Node Q***		
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*	$<\mathbf{P_{\varnothing}, Q_{\varnothing}}>$	Globally **reachable** persistent states
send(Q, M1)	recv(M1)		
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*		
write(fd, “foo”,3)			
# State P_1			
write(fd, “baz”,3)			
# State P_2			
send(Q, M2)	recv(M2)		
# State P_2	*# State $Q_{\varnothing}$*		
	write(fd, “bar”, 3)		
	# State Q_1	$<\mathbf{P_2, Q_1}>$	

Approach: Finding Reachable States

Node P ***Node Q***

 # State $P_{\emptyset}$ *# State $Q_{\emptyset}$* <$P_{\emptyset}$, $Q_{\emptyset}$>

Globally **reachable** persistent states

send(Q, M1) → recv(M1)

 # State $P_{\emptyset}$ *# State $Q_{\emptyset}$*

write(fd, “foo”,3)

 # State P_1 <P_1, $Q_{\emptyset}$>

write(fd, “baz”,3)

 # State P_2

send(Q, M2) → recv(M2)

 # State P_2

 # State $Q_{\emptyset}$

write(fd, “bar”, 3)

 # State Q_1

<P_2, Q_1>

Approach: Finding Reachable States

Node P ***Node Q***

 # State $P_\emptyset$ *# State $Q_\emptyset$* $\langle \boldsymbol{P_\emptyset}, \boldsymbol{Q_\emptyset} \rangle$

Globally **reachable** persistent states

send(Q, M1) → recv(M1)

 # State $P_\emptyset$ *# State $Q_\emptyset$*

write(fd, “foo”,3)

 # State P_1 $\langle \boldsymbol{P_1}, \boldsymbol{Q_\emptyset} \rangle$

write(fd, “baz”,3)

 # State P_2 $\langle \boldsymbol{P_2}, \boldsymbol{Q_\emptyset} \rangle$

send(Q, M2) → recv(M2)

 # State P_2

 # State $Q_\emptyset$

write(fd, “bar”, 3)

 # State Q_1

$\langle \boldsymbol{P_2}, \boldsymbol{Q_1} \rangle$

Approach: Finding Reachable States

Node P	***Node Q***	
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*	$\langle \mathbf{P_{\varnothing}, Q_{\varnothing}} \rangle$
send(Q, M1)	recv(M1)	
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*	
write(fd, "foo",3)		
# State P_1		$\langle \mathbf{P_1, Q_{\varnothing}} \rangle$
write(fd, "baz",3)		
# State P_2		$\langle \mathbf{P_2, Q_{\varnothing}} \rangle$
send(Q, M2)	recv(M2)	
# State P_2	*# State $Q_{\varnothing}$*	
	write(fd, "bar", 3)	
	# State Q_1	
		$\langle \mathbf{P_2, Q_1} \rangle$

Globally **reachable** persistent states

Consistent Cuts

Reachable Persistent Prefixes

Approach: Finding Reachable States

Node P ***Node Q***

\# State $P_{\emptyset}$ \# State $Q_{\emptyset}$ $<P_{\emptyset}, Q_{\emptyset}>$

send(Q, M1)

recv(M1)

\# State $P_{\emptyset}$ \# State $Q_{\emptyset}$

write(fd, “foo”,3)

\# State P_1 $<P_1, Q_{\emptyset}>$

write(fd, “baz”,3)

\# State P_2 $<P_2, Q_{\emptyset}>$

send(Q, M2)

recv(M2)

\# State P_2

\# State $Q_{\emptyset}$

write(fd, “bar”, 3)

\# State Q_1

$<P_2, Q_1>$

Globally **reachable** persistent states

Consistent Cuts

Reachable Persistent Prefixes

Not Reachable

$<P_{\emptyset}, Q_1>$ $<P_1, Q_1>$

File-System Behaviors: Reordering

Node P	***Node Q***
# State $P_\emptyset$	*# State $Q_\emptyset$*
send(Q, M1)	
	recv(M1)
# State $P_\emptyset$	*# State $Q_\emptyset$*
1 write(fd, “foo”,3)	
# State P_1	
2 write(fd, “baz”,3)	
# State P_2	<P_2, $Q_\emptyset$>
send(Q, M2)	
	recv(M2)
# State P_2	
	# State $Q_\emptyset$
	write(fd, “bar”, 3)
	# State Q_1

File-System Behaviors: Reordering

Node P	***Node Q***
# State $P_\varnothing$	*# State $Q_\varnothing$*
send(Q, M1)	
	recv(M1)
# State $P_\varnothing$	*# State $Q_\varnothing$*
1 write(fd, “foo”,3)	
# State P_1	
2 write(fd, “baz”,3)	
# State P_2	**<P_2, $Q_\varnothing$>**
send(Q, M2)	
	recv(M2)
# State P_2	*# State $Q_\varnothing$*
	write(fd, “bar”, 3)
	# State Q_1

File systems may **reorder** updates to disk

1 write(fd, “foo”,3)

2 write(fd, “baz”,3)

File-System Behaviors: Reordering

Node P	***Node Q***
# State $P_ø$	*# State $Q_ø$*
send(Q, M1)	recv(M1)
# State $P_ø$	*# State $Q_ø$*
1 write(fd, “foo”,3)	
# State P_1	
2 write(fd, “baz”,3)	
# State P_2	<P_2, $Q_ø$>
send(Q, M2)	recv(M2)
# State P_2	*# State $Q_ø$*
	write(fd, “bar”, 3)
	# State Q_1

File systems may **reorder** updates to disk

1 write(fd, “foo”,3)

2 write(fd, “baz”,3)

File-System Behaviors: Reordering

Node P	***Node Q***
# State $P_\emptyset$	*# State $Q_\emptyset$*
send(Q, M1)	recv(M1)
# State $P_\emptyset$	*# State $Q_\emptyset$*
1 write(fd, "foo",3)	
# State P_1	
2 write(fd, "baz",3)	
# State P_2	$\langle \boldsymbol{P_2}, \boldsymbol{Q_\emptyset} \rangle$
send(Q, M2)	recv(M2)
# State P_2	*# State $Q_\emptyset$*
	write(fd, "bar", 3)
	# State Q_1

File systems may **reorder** updates to disk

1. write(fd, "foo",3)
2. write(fd, "baz",3)

File-System Behaviors: Reordering

Node P	***Node Q***
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
send(Q, M1)	recv(M1)
# State $P_{\emptyset}$	*# State $Q_{\emptyset}$*
1 write(fd, "foo",3)	
# State P_1	
2 write(fd, "baz",3)	
# State P_2	<P_2, $Q_{\emptyset}$>
send(Q, M2)	recv(M2)
# State P_2	*# State $Q_{\emptyset}$*
	write(fd, "bar", 3)
	# State Q_1

File systems may **reorder** updates to disk

1 write(fd, "foo",3)

2 write(fd, "baz",3)

Reboot

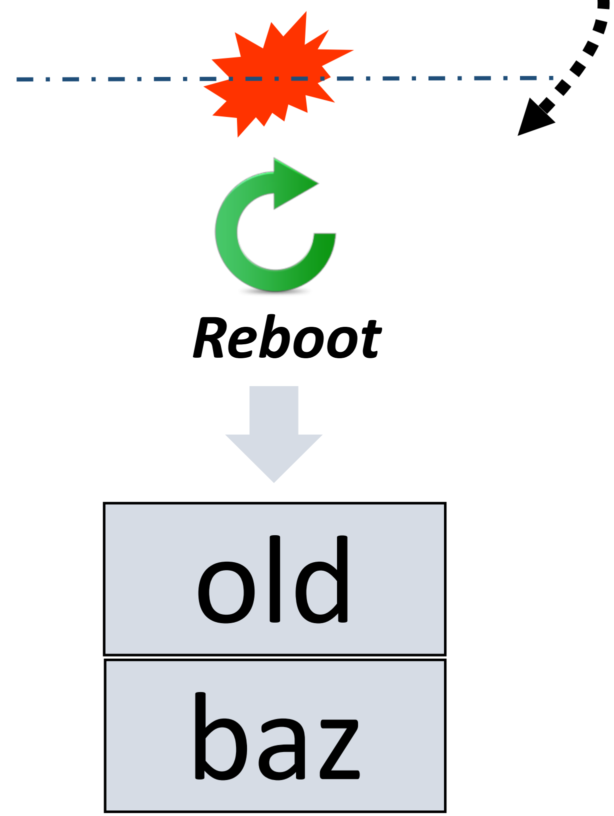
File-System Behaviors: Reordering

Node P	***Node Q***
# State $P_\varnothing$	*# State $Q_\varnothing$*
send(Q, M1)	recv(M1)
# State $P_\varnothing$	*# State $Q_\varnothing$*
1 write(fd, “foo”,3)	
# State P_1	
2 write(fd, “baz”,3)	
# State P_2	<**P_2, $Q_\varnothing$**>
send(Q, M2)	recv(M2)
# State P_2	*# State $Q_\varnothing$*
	write(fd, “bar”, 3)
	# State Q_1

File systems may **reorder** updates to disk

1 write(fd, “foo”,3)

2 write(fd, “baz”,3)



File-System Behaviors: Atomicity

Node P	***Node Q***
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*
send(Q, M1)	
	recv(M1)
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*
write(fd, “foo”,3)	
# State P_1	
write(fd, “baz”,3)	
# State P_2	
send(Q, M2)	
	recv(M2)
# State P_2	*# State $Q_{\varnothing}$*
	write(fd, “bar”, 3)
	# State Q_1

File-System Behaviors: Atomicity

Node P

 # State $P_{\emptyset}$

send(Q, M1)

 # State $P_{\emptyset}$

write(fd, “foo”,3)

 # State P_1

2 write(fd, “baz”,3)

 # State P_2

send(Q, M2)

 # State P_2

Node Q

 # State $Q_{\emptyset}$

recv(M1)

 # State $Q_{\emptyset}$

recv(M2)

 # State $Q_{\emptyset}$

write(fd, “bar”, 3)

 # State Q_1

File systems may **partially persist operations**

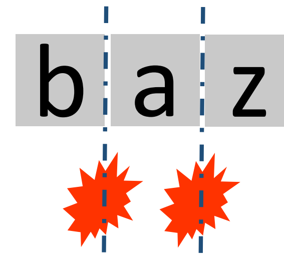
2 write(fd, “baz”,3)

File-System Behaviors: Atomicity

Node P	***Node Q***
# State $P_{\varnothing}$	# State $Q_{\varnothing}$
send(Q, M1)	recv(M1)
# State $P_{\varnothing}$	# State $Q_{\varnothing}$
write(fd, "foo",3)	
# State P_1	
2 write(fd, "baz",3)	
# State P_2	
send(Q, M2)	recv(M2)
# State P_2	# State $Q_{\varnothing}$
	write(fd, "bar", 3)
	# State Q_1

File systems may **partially persist operations**

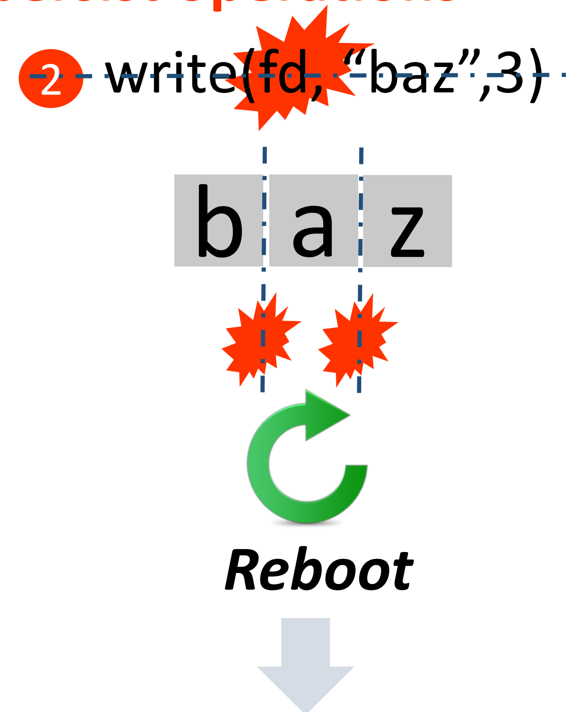
2 write(fd, "baz",3)



File-System Behaviors: Atomicity

Node P	***Node Q***
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*
send(Q, M1)	
	recv(M1)
# State $P_{\varnothing}$	*# State $Q_{\varnothing}$*
write(fd, "foo",3)	
# State P_1	
2 write(fd, "baz",3)	
# State P_2	
send(Q, M2)	
	recv(M2)
# State P_2	*# State $Q_{\varnothing}$*
	write(fd, "bar", 3)
	# State Q_1

File systems may **partially persist operations**



File-System Behaviors: Atomicity

Node P

 # State $P_\varnothing$

send(Q, M1)

 # State $P_\varnothing$

write(fd, "foo",3)

 # State P_1

2 write(fd, "baz",3)

 # State P_2

send(Q, M2)

 # State P_2

Node Q

 # State $Q_\varnothing$

recv(M1)

 # State $Q_\varnothing$

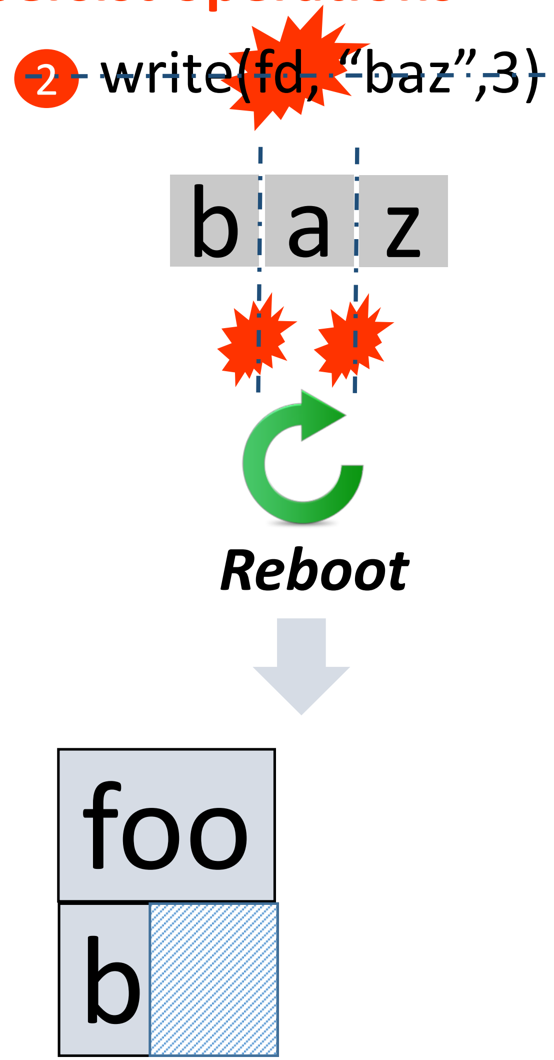
recv(M2)

 # State $Q_\varnothing$

write(fd, "bar", 3)

 # State Q_1

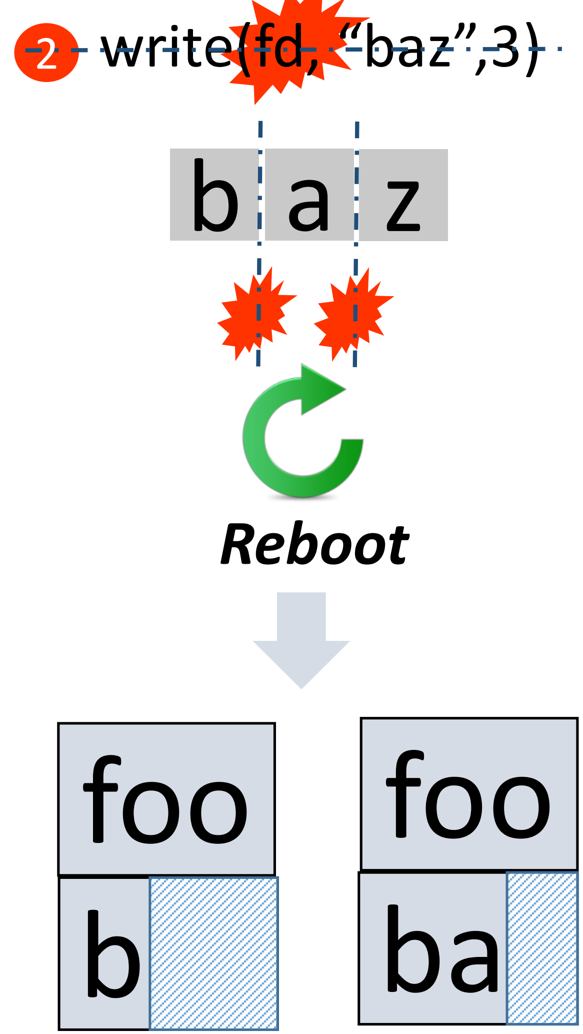
File systems may **partially persist operations**



File-System Behaviors: Atomicity

Node P	***Node Q***
# State $P_\emptyset$	*# State $Q_\emptyset$*
send(Q, M1)	recv(M1)
# State $P_\emptyset$	*# State $Q_\emptyset$*
write(fd, “foo”,3)	
# State P_1	
2 write(fd, “baz”,3)	
# State P_2	
send(Q, M2)	recv(M2)
# State P_2	*# State $Q_\emptyset$*
	write(fd, “bar”, 3)
	# State Q_1

File systems may **partially persist operations**



Modeling File-System Behaviors

Modeling File-System Behaviors

Reordering and partially persisting on crashes

– Relaxations

Modeling File-System Behaviors

Reordering and partially persisting on crashes

– Relaxations

Relaxations vary across file systems

Modeling File-System Behaviors

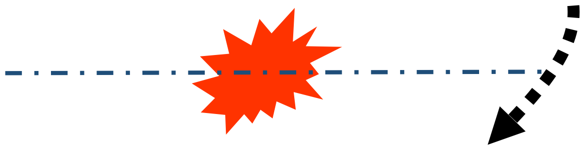
Reordering and partially persisting on crashes

– Relaxations

Relaxations vary across file systems

1 write(fd, “foo”,3)

2 write(fd, “baz”,3)



Modeling File-System Behaviors

Reordering and partially persisting on crashes

– Relaxations

Relaxations vary across file systems

1 write(fd, “foo”,3)

2 write(fd, “baz”,3)

Can happen on: ext3 and ext4 (writeback, ordered), ext2, XFS

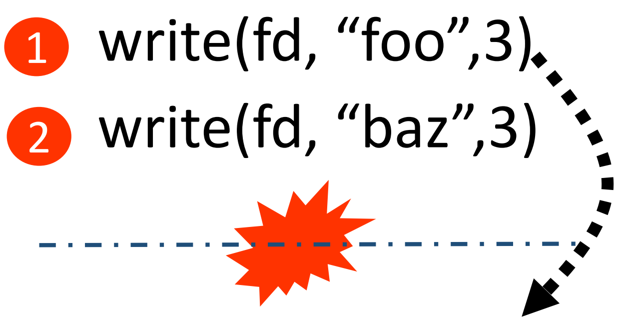
Not on: ext3 and ext4(data), btrfs

Modeling File-System Behaviors

Reordering and partially persisting on crashes

– Relaxations

Relaxations vary across file systems



Can happen on: ext3 and ext4 (writeback, ordered), ext2, XFS

Not on: ext3 and ext4(data), btrfs

Abstract Persistence Model (APM) [Pillai et., OSDI’14] defines relaxations allowed on a particular file system

Outline

Introduction

Correlated Crashes

PACE (Protocol-Aware Crash Explorer)

- **State Space and PACE Rules**
- **Methodology**

Vulnerability Study

Conclusion

Protocol-Aware Crash Explorer

Protocol-Aware Crash Explorer

Relaxations on one node results in many states for that node

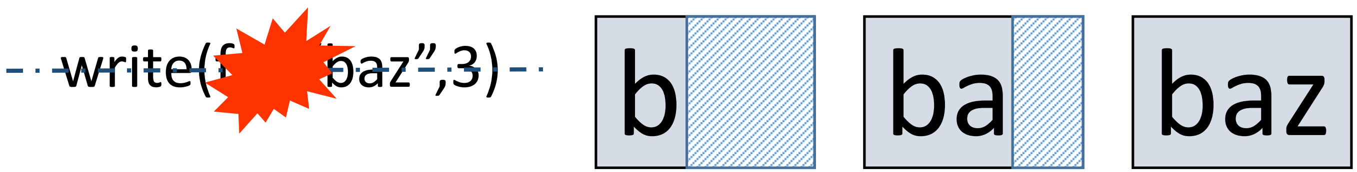
Protocol-Aware Crash Explorer

Relaxations on one node results in many states for that node

write(fd, “baz”,3)

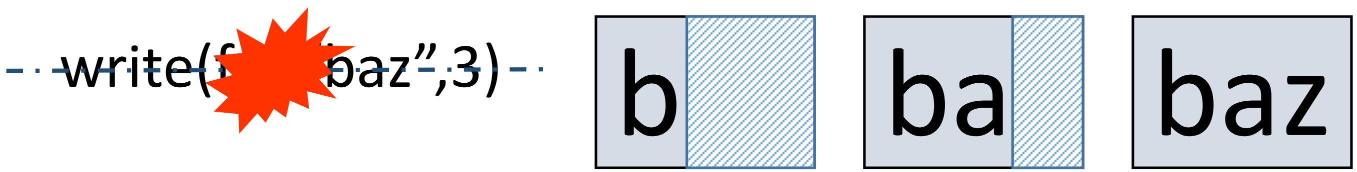
Protocol-Aware Crash Explorer

Relaxations on one node results in many states for that node



Protocol-Aware Crash Explorer

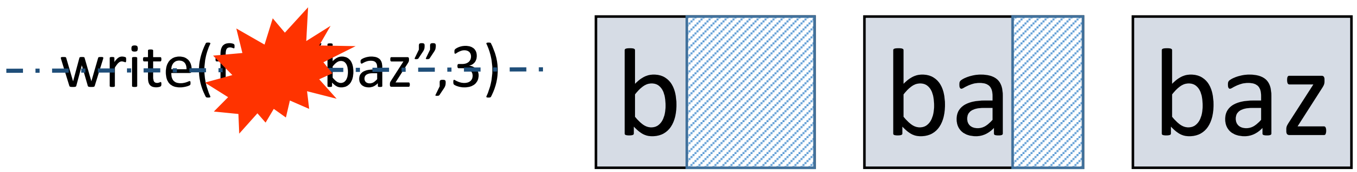
Relaxations on one node results in many states for that node



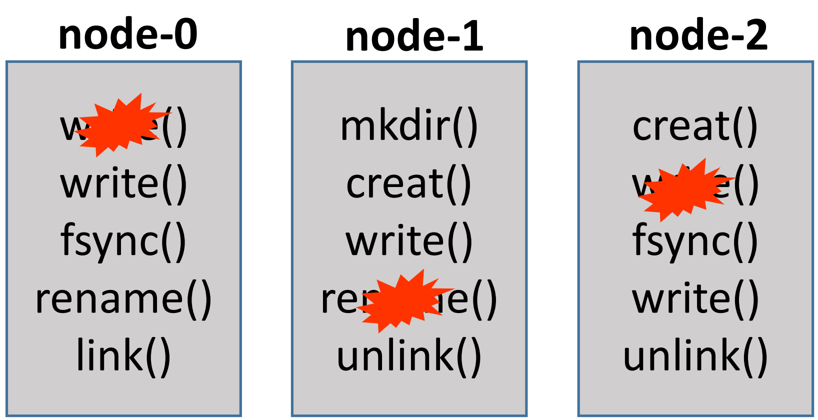
PACE needs to consider relaxations in different combinations of nodes on all reachable prefixes

Protocol-Aware Crash Explorer

Relaxations on one node results in many states for that node

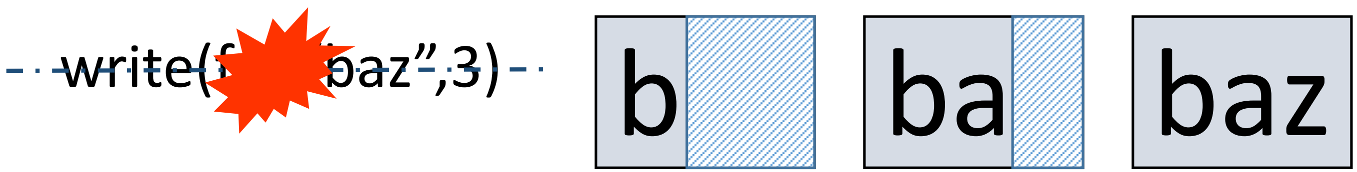


PACE needs to consider relaxations in different combinations of nodes on all reachable prefixes

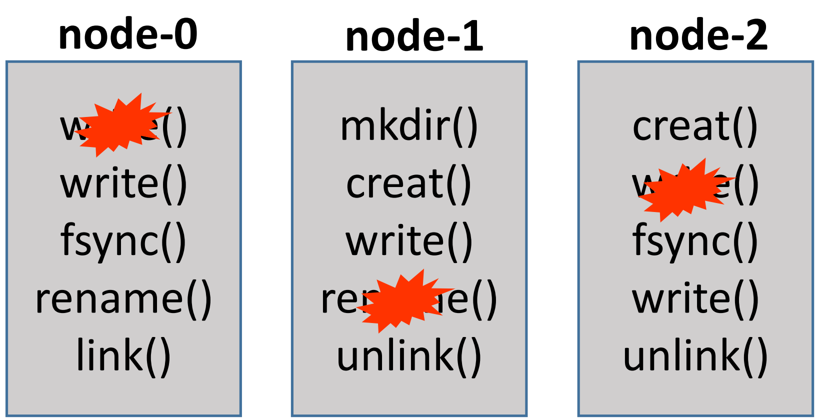


Protocol-Aware Crash Explorer

Relaxations on one node results in many states for that node



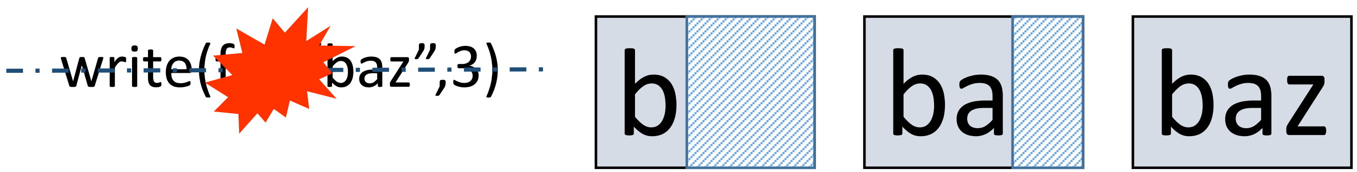
PACE needs to consider relaxations in different combinations of nodes on all reachable prefixes



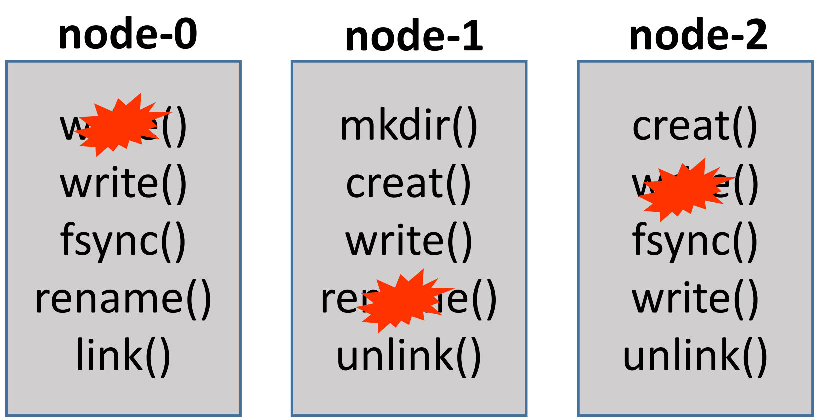
Relax on {0} {1} {2} {0,1} {0,2} {1,2} {0,1,2} ⇒ huge state space

Protocol-Aware Crash Explorer

Relaxations on one node results in many states for that node



PACE needs to consider relaxations in different combinations of nodes on all reachable prefixes



Relax on {0} {1} {2} {0,1} {0,2} {1,2} {0,1,2} ⇒ huge state space

PACE prunes this space using generic rules

PACE Pruning Rules

Replicated State Machine (RSM) approaches

Other (non-RSM) approaches

Details in the paper …

PACE: Effectiveness of Pruning

PACE: Effectiveness of Pruning

LogCabin (RSM)

- brute-force explored ~1M states (over a week) to find 2 vulnerabilities
- PACE explored only ~28K states (in under 8 hours) and found the same vulnerabilities

PACE: Effectiveness of Pruning

LogCabin (RSM)

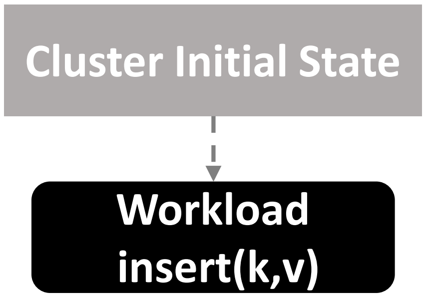
- brute-force explored ~1M states (over a week) to find 2 vulnerabilities
- PACE explored only ~28K states (in under 8 hours) and found the same vulnerabilities

Redis (non-RSM)

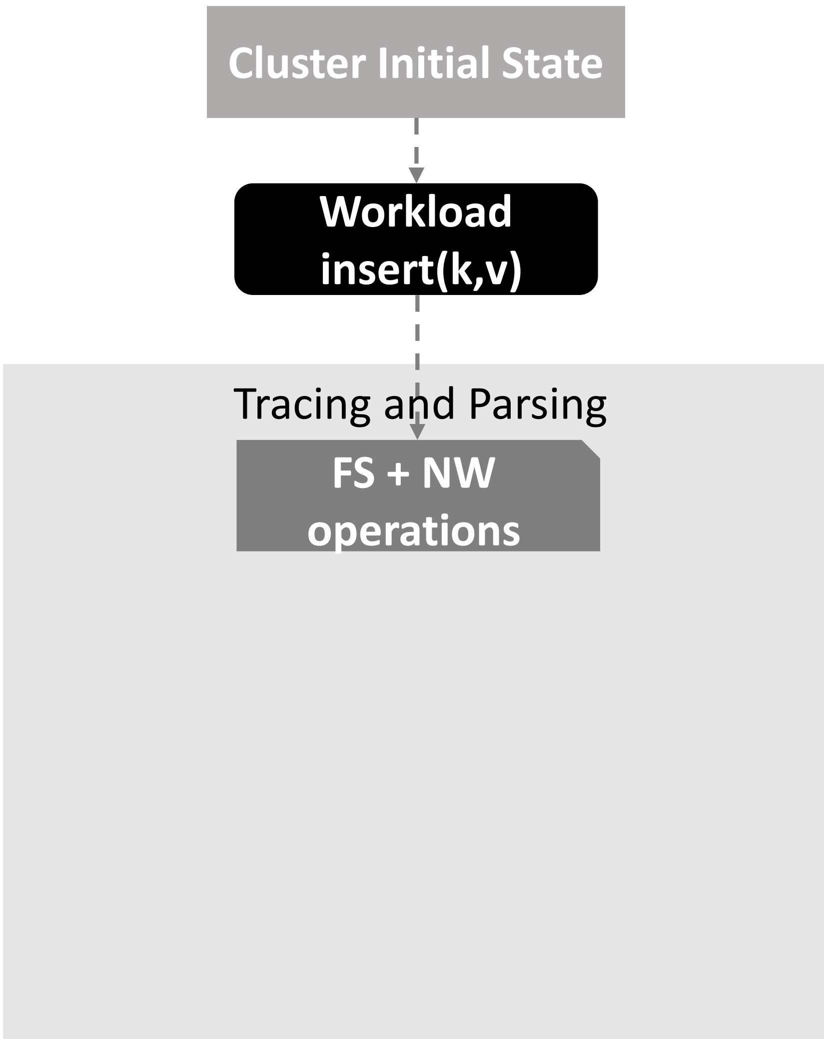
- PACE explored 11x fewer states than the brute – force search finding the same 3 vulnerabilities

PACE: Methodology

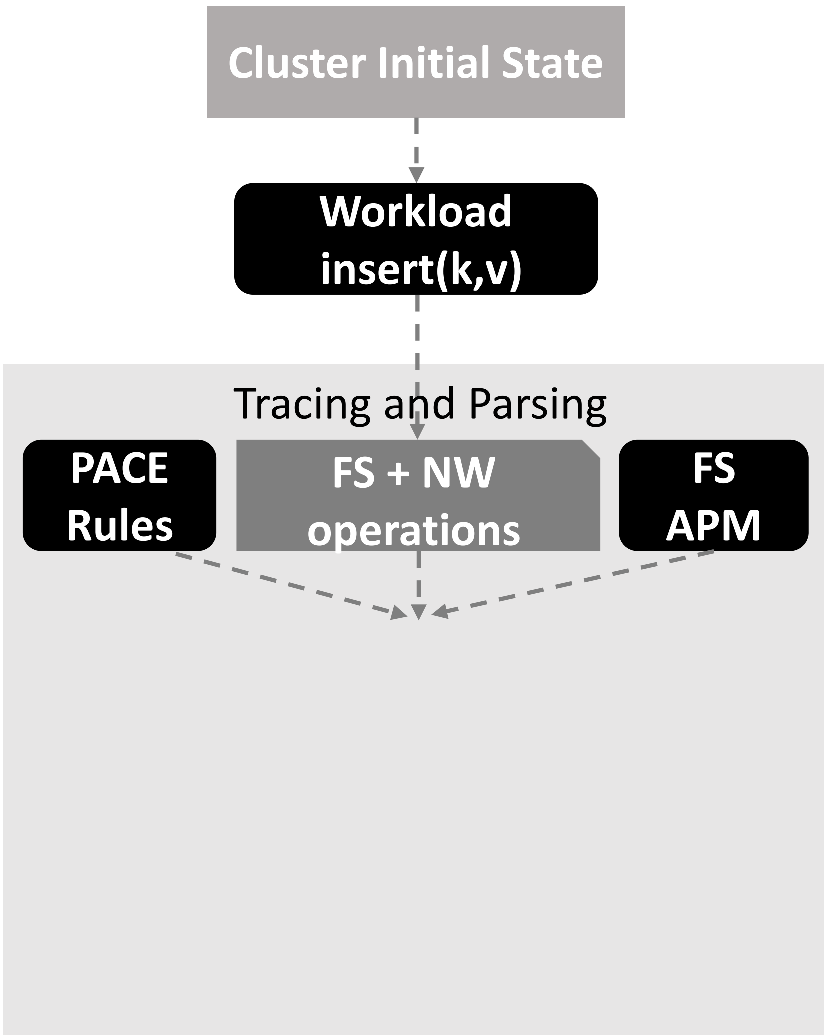
PACE: Methodology



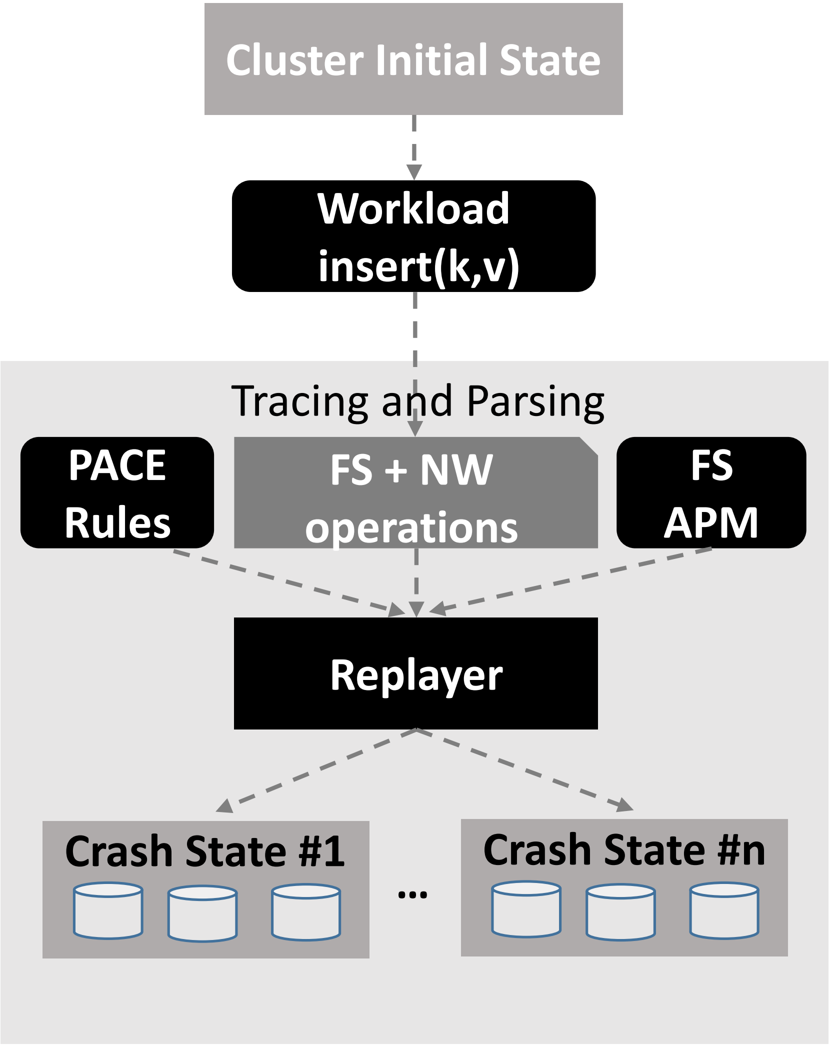
PACE: Methodology



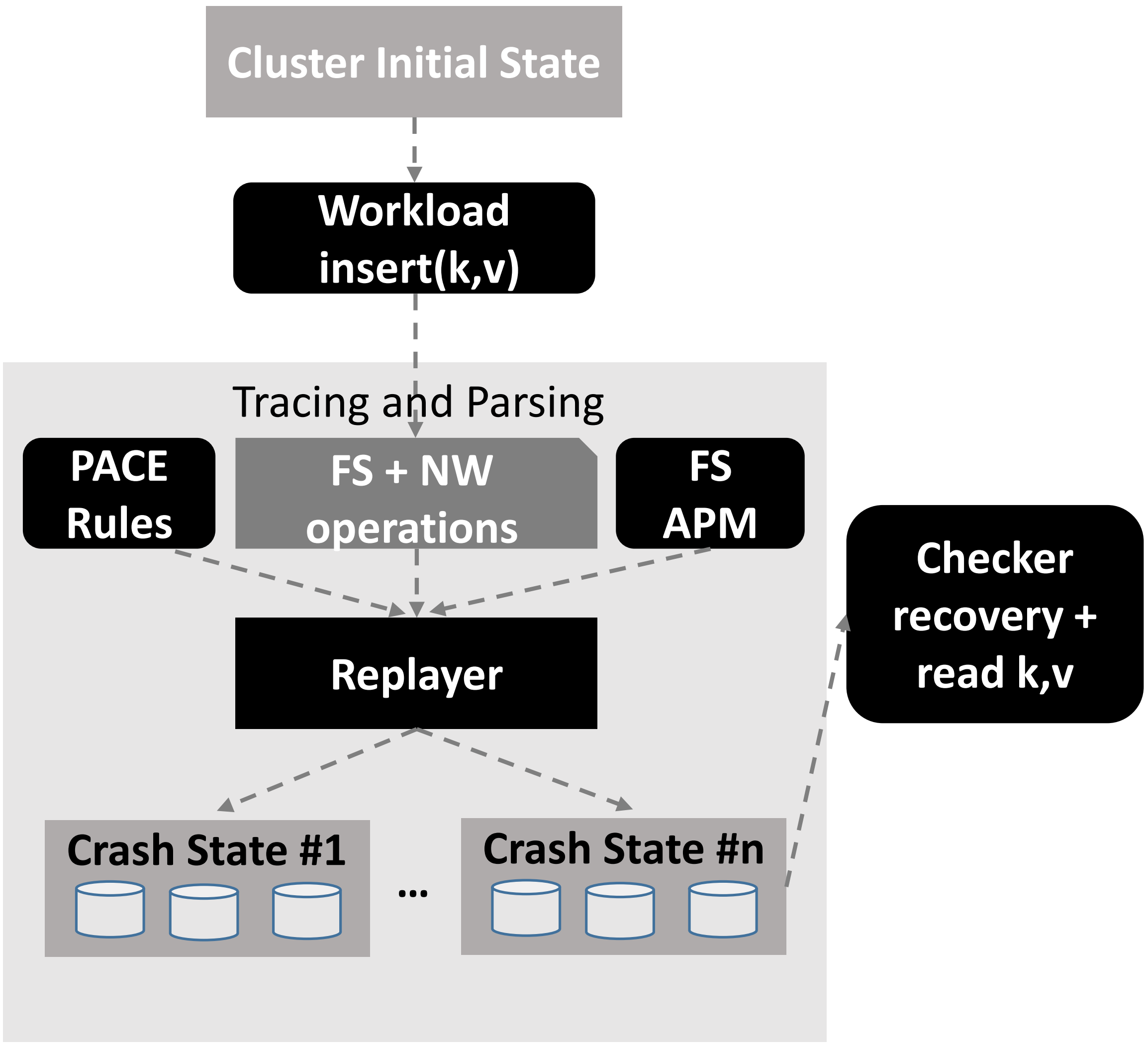
PACE: Methodology



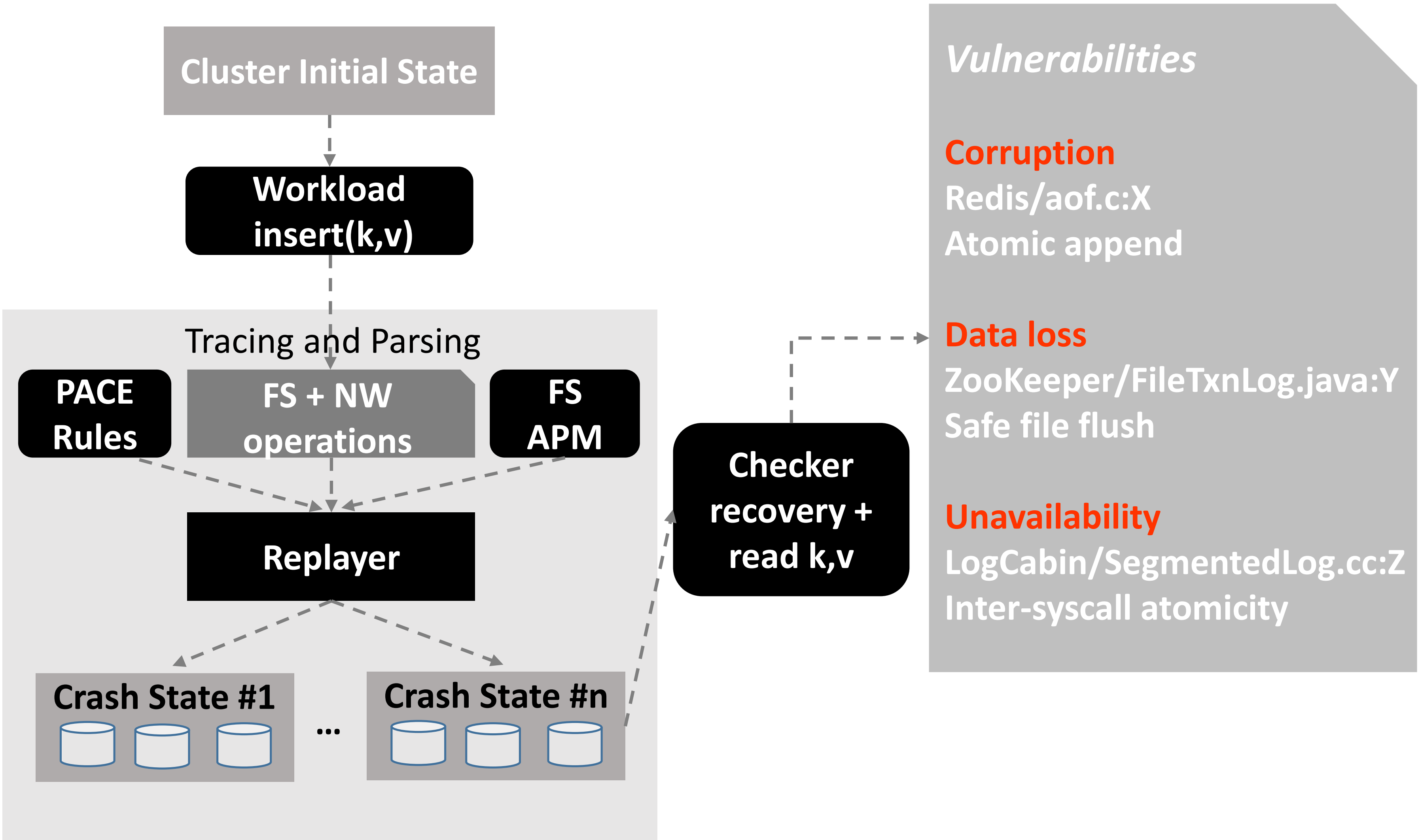
PACE: Methodology



PACE: Methodology



PACE: Methodology



Outline

Introduction

Correlated Crashes

Protocol-Aware Crash Explorer

Vulnerability Study

Concluding Remarks

Vulnerability Study: Systems

Database caches: Redis

Metadata stores: ZooKeeper, LogCabin, etcd

Real-time DB: RethinkDB

Document stores: MongoDB

Message Queues: Kafka

Key-value stores: iNexus

Safest configurations: synchronous replication, synchronous disk writes, checksums etc.,

Example: Redis

Example: Redis

Follower1 **Master** **Follower2**

Example: Redis

Follower1	Master	Follower2
creat(tmp) append(tmp) rename(tmp, tmp-bg) rename(tmp-bg, aof) fdatasync(aof)		creat(tmp) append(tmp) rename(tmp, tmp-bg) rename(tmp-bg, aof) fdatasync(aof)

Example: Redis

Update request

Follower1

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

Master

Follower2

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

Example: Redis

Update request

Follower1

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

Master

append(aof)

Follower2

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

Example: Redis

Update request

Follower1	**Master**	**Follower2**
creat(tmp) append(tmp) rename(tmp, tmp-bg) rename(tmp-bg, aof) fdatasync(aof)	append(aof) fdatasync(aof)	creat(tmp) append(tmp) rename(tmp, tmp-bg) rename(tmp-bg, aof) fdatasync(aof)

Example: Redis

Update request

Follower1

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

Master

append(aof)
fdatasync(aof)
send(data)
send(data)

Follower2

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

Example: Redis

Update request

Follower1

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

recv(data)
append(aof)
fdatasync(aof)
send(ack)

Master

append(aof)
fdatasync(aof)
send(data)
send(data)

Follower2

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

recv(data)
append(aof)
fdatasync(aof)
send(ack)

Example: Redis

Update request

Follower1

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

recv(data)
append(aof)
fdatasync(aof)
send(ack)

Master

append(aof)
fdatasync(aof)
send(data)
send(data)

recv(ack)
recv(ack)
send(client, ack)

Follower2

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

recv(data)
append(aof)
fdatasync(aof)
send(ack)

Acknowledgment

Example: Redis

Update request

Follower1

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

recv(data)
append(aof)
fdatasync(aof)
send(ack)

Master

append(aof)
fdatasync(aof)
send(data)
send(data)

recv(ack)
recv(ack)
send(client, ack)

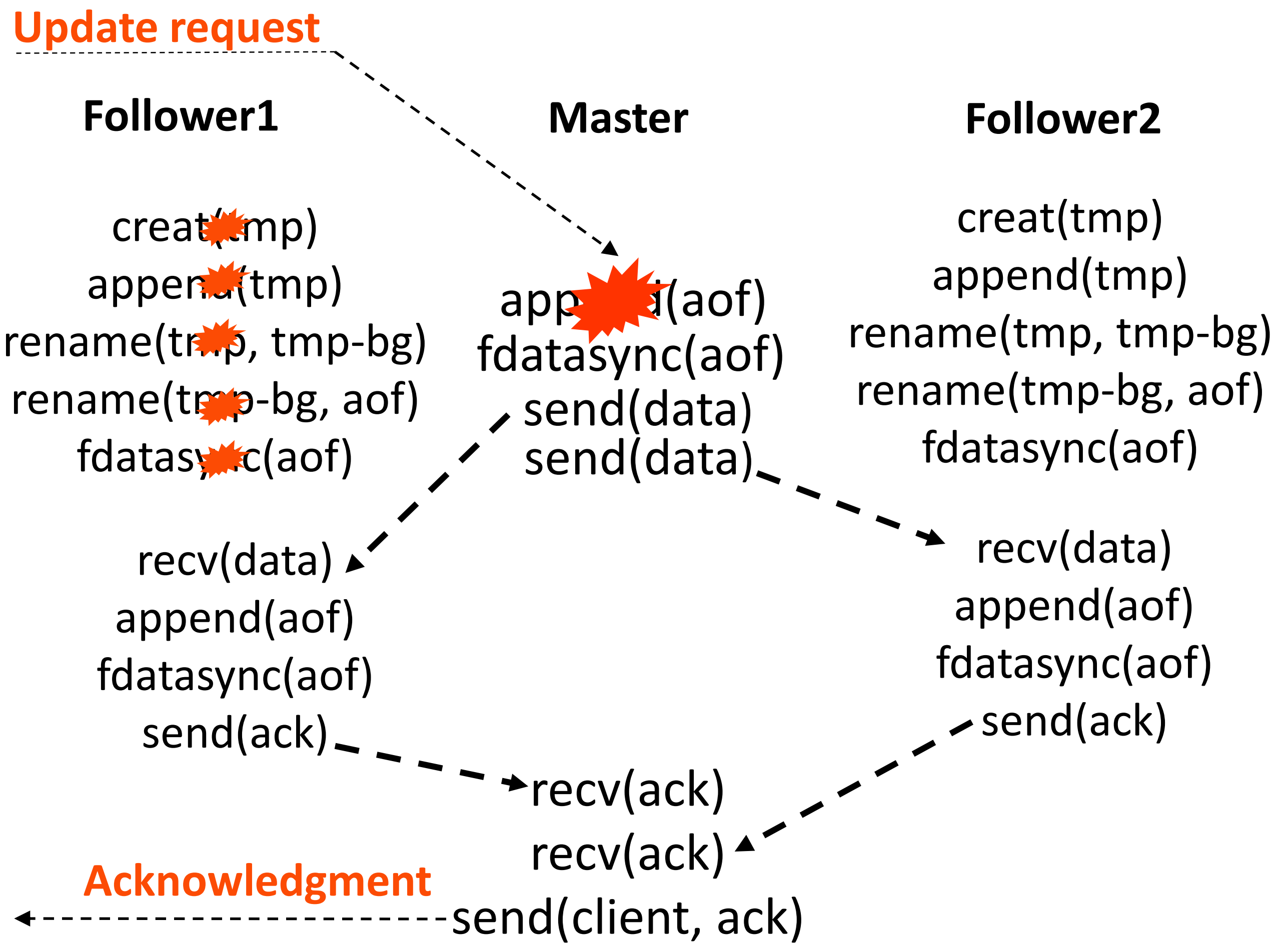
Follower2

creat(tmp)
append(tmp)
rename(tmp, tmp-bg)
rename(tmp-bg, aof)
fdatasync(aof)

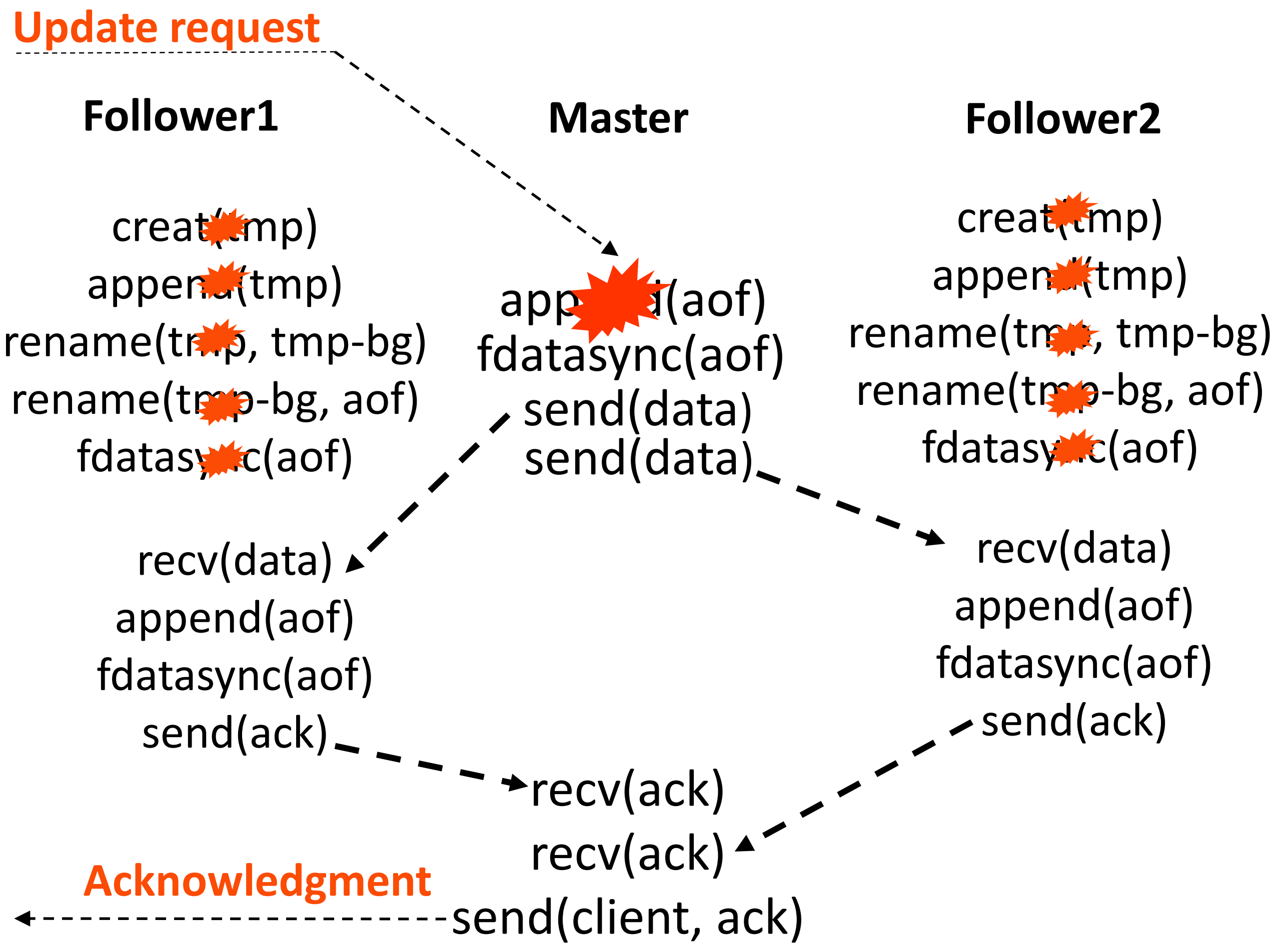
recv(data)
append(aof)
fdatasync(aof)
send(ack)

Acknowledgment

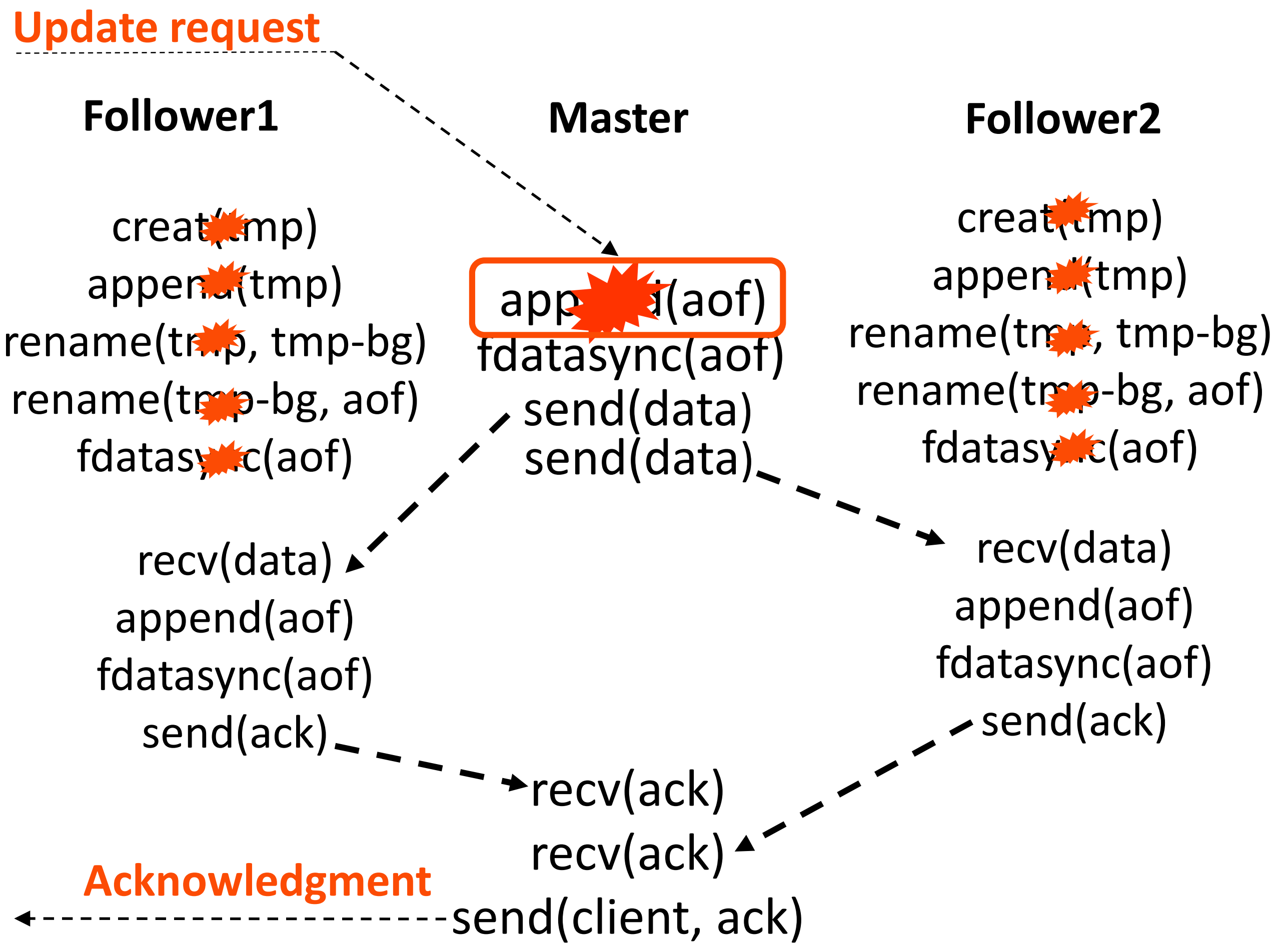
Example: Redis



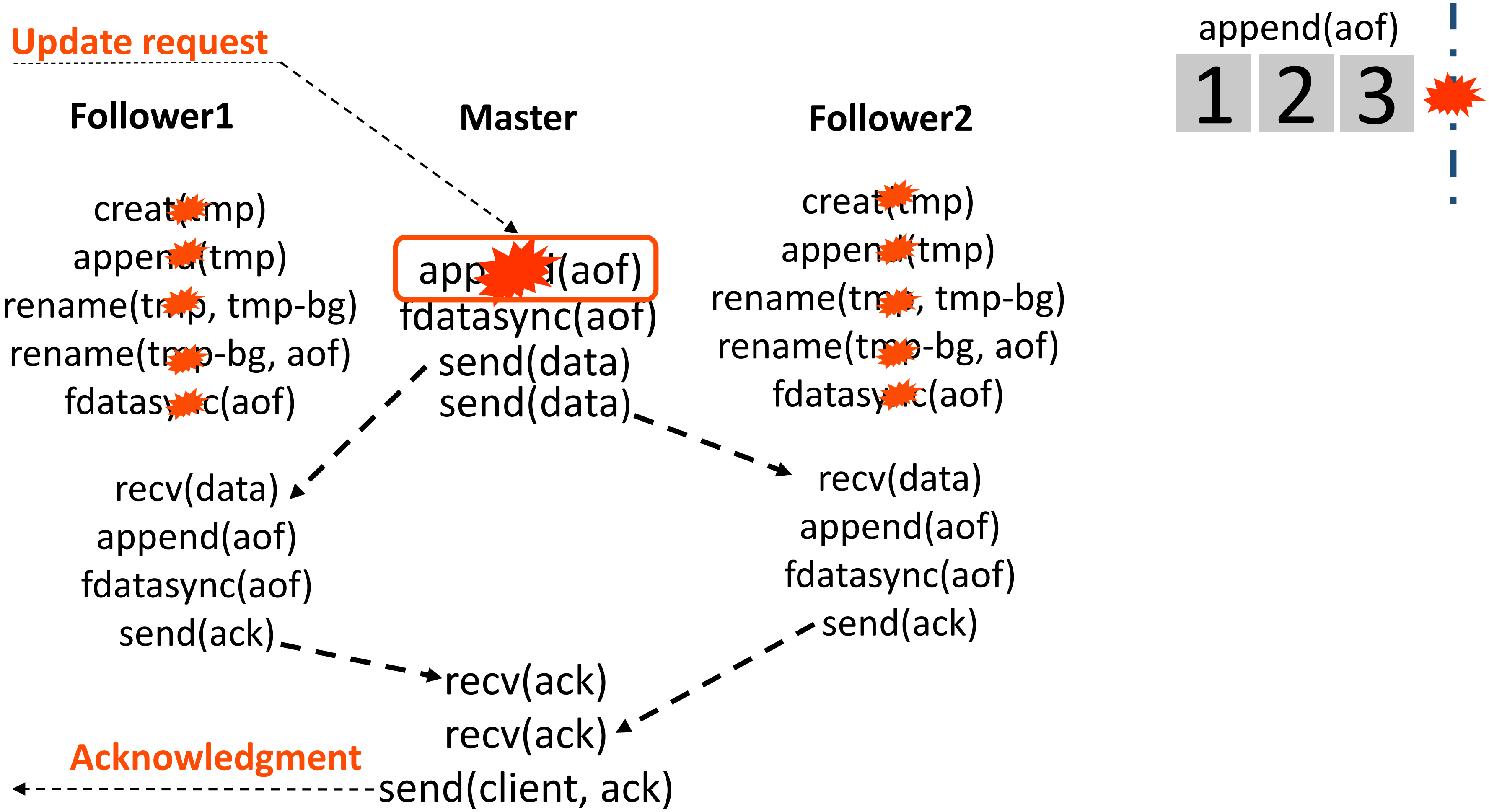
Example: Redis



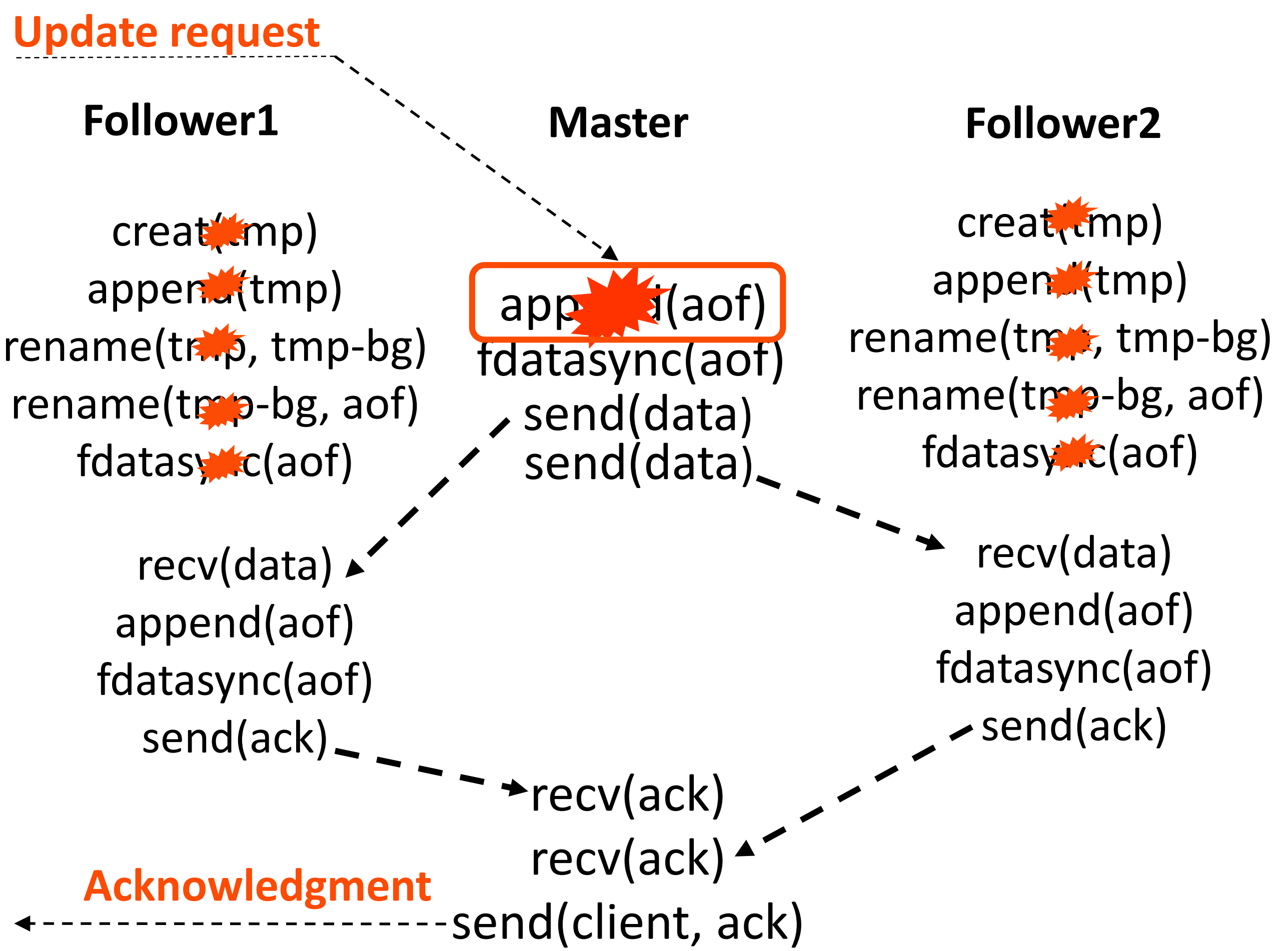
Example: Redis

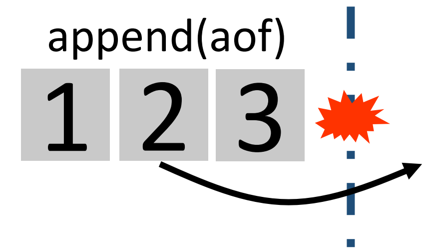


Example: Redis

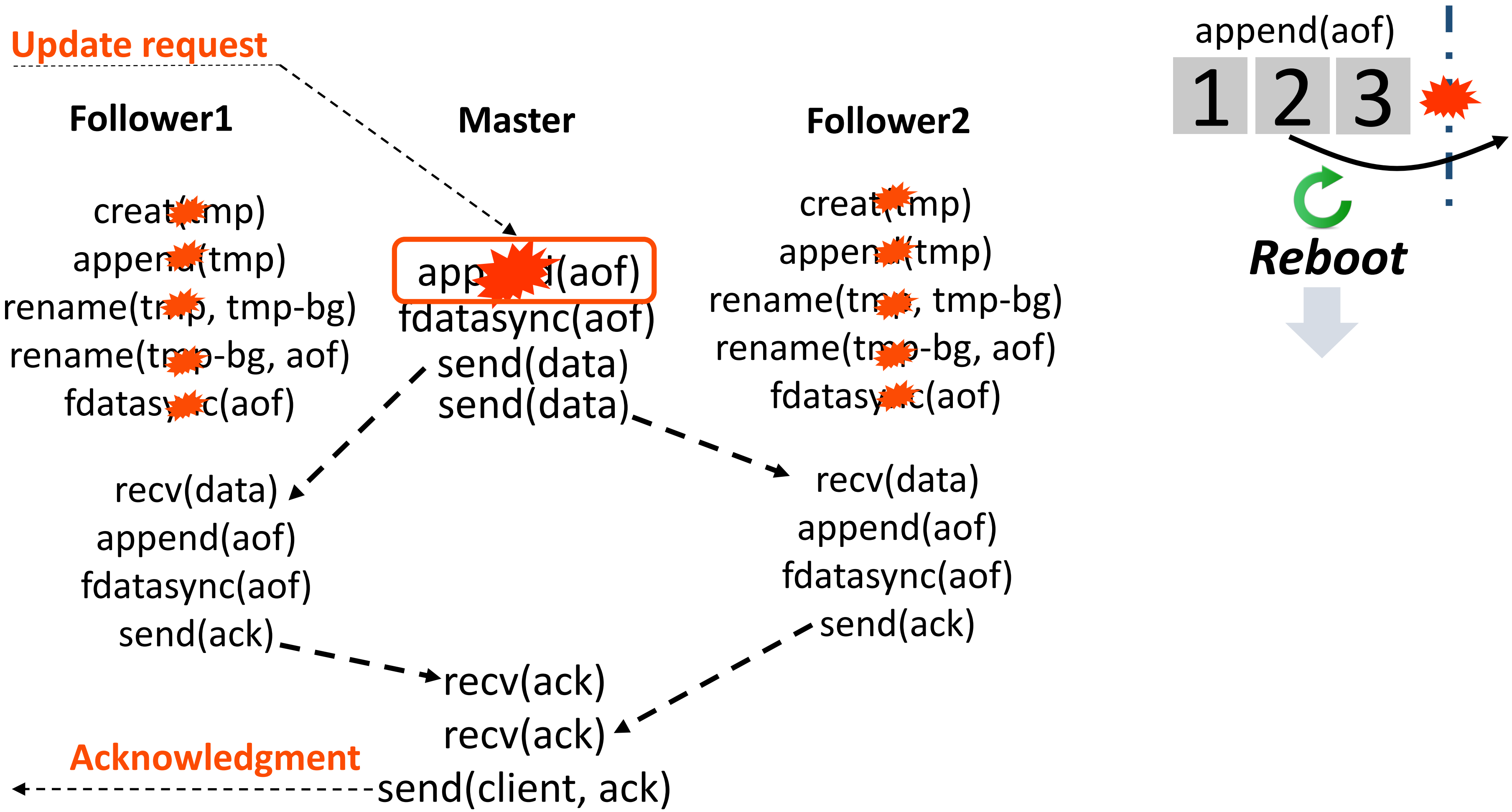


Example: Redis

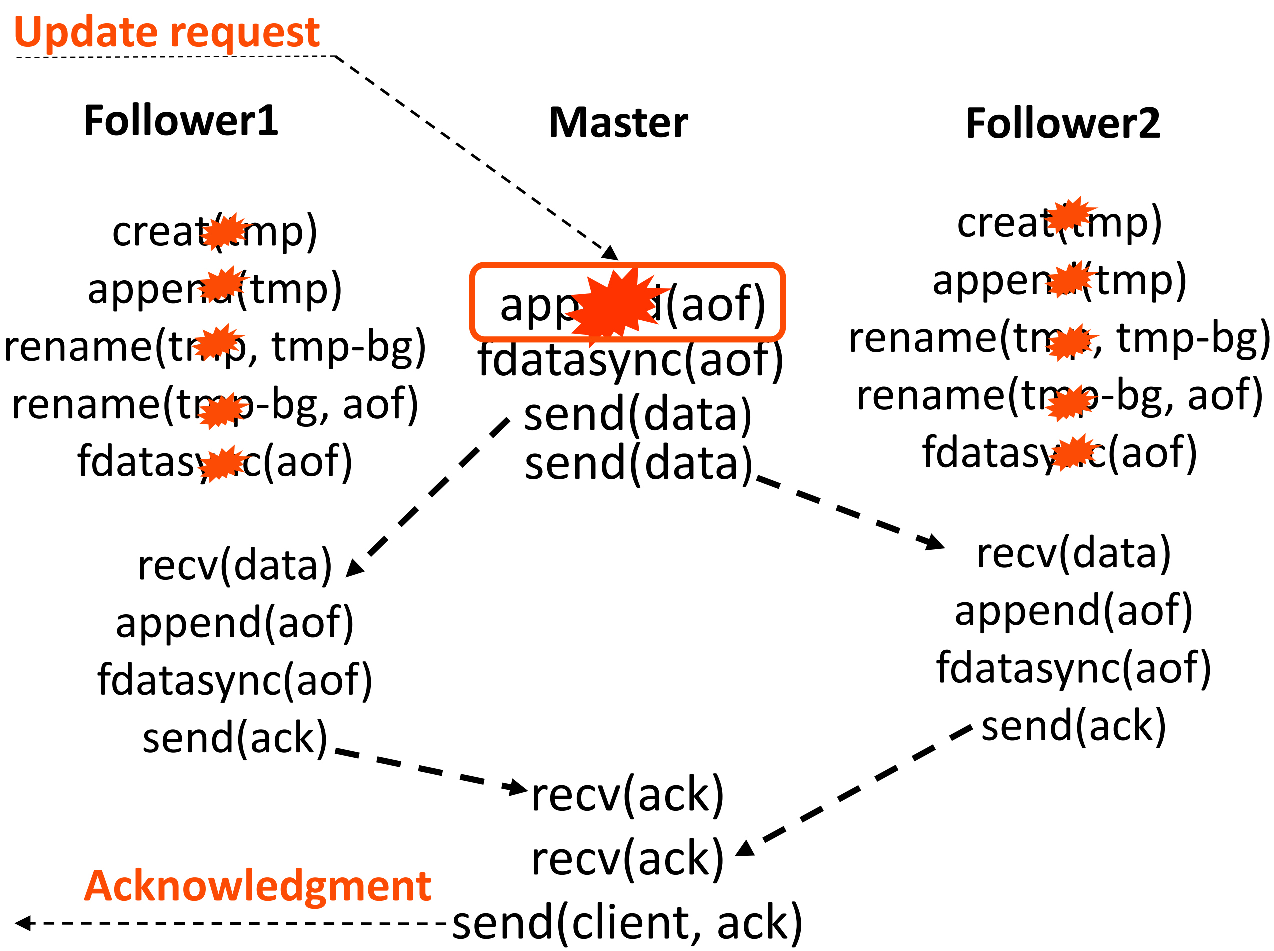


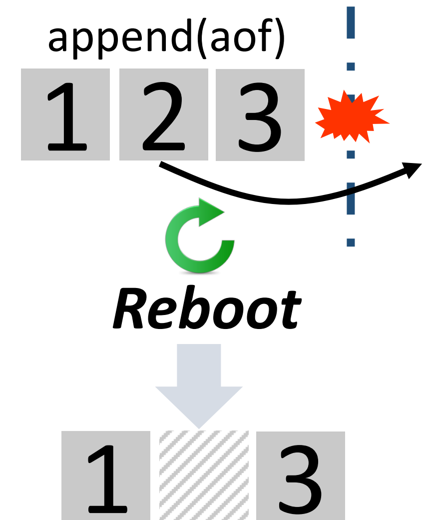


Example: Redis

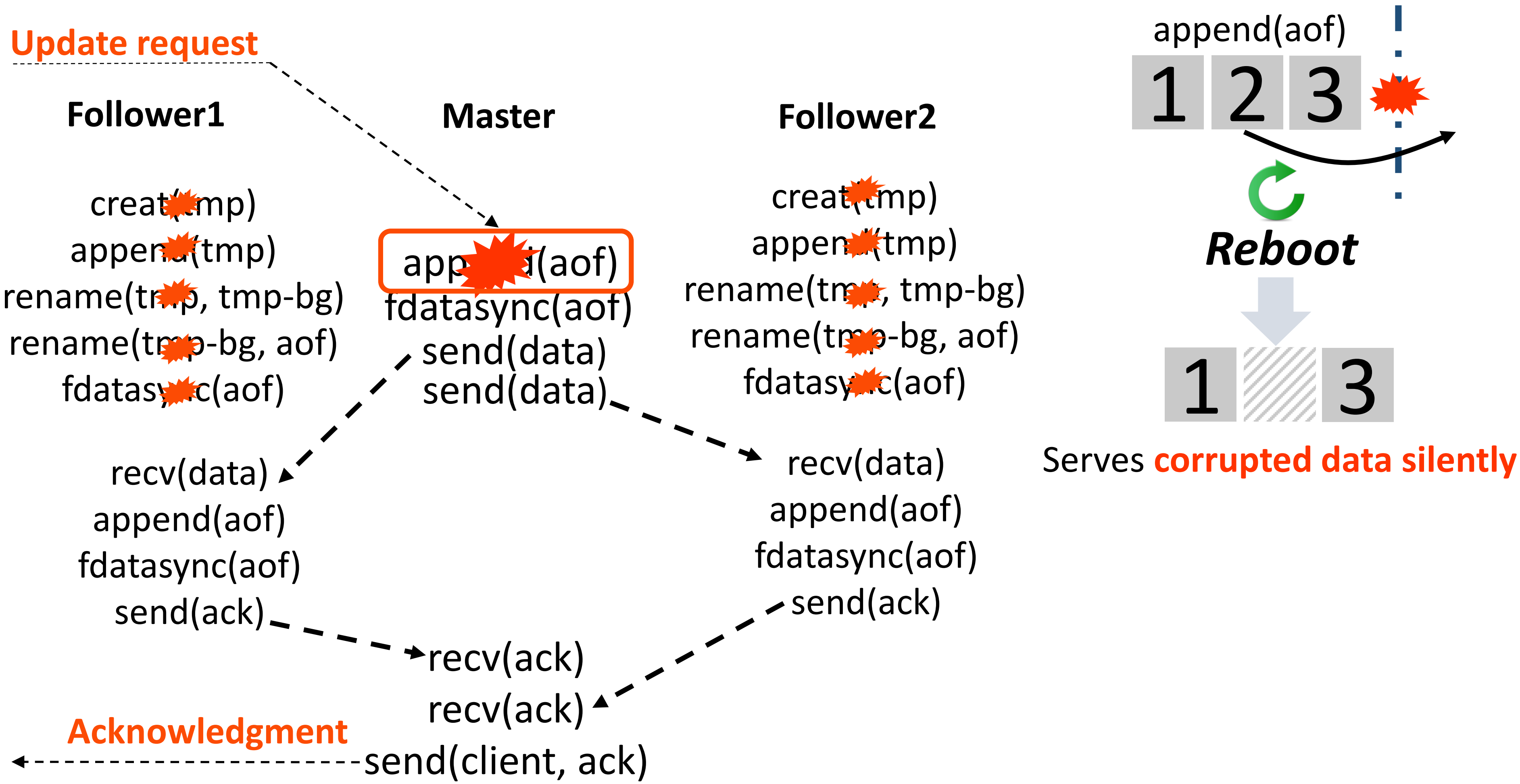


Example: Redis

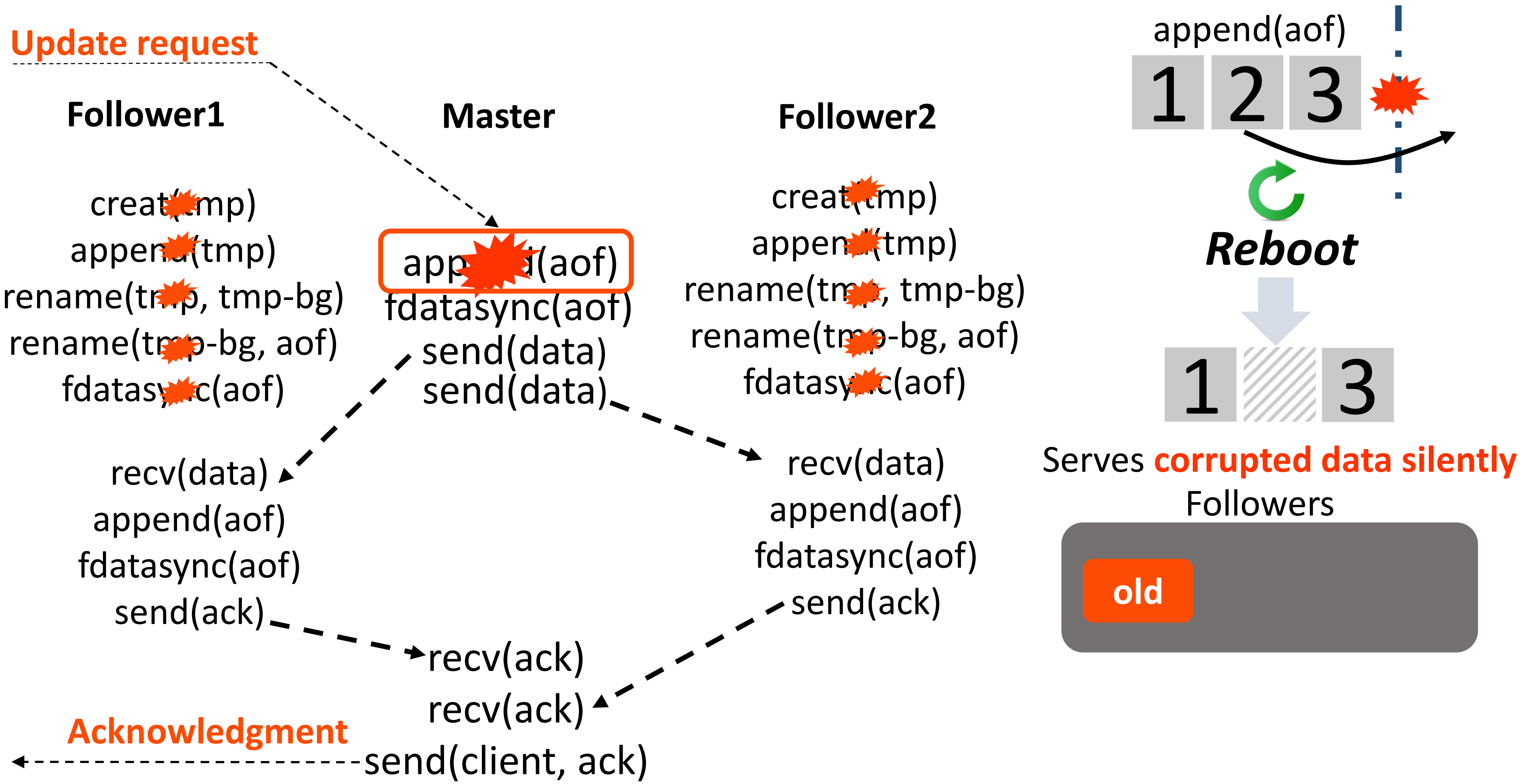




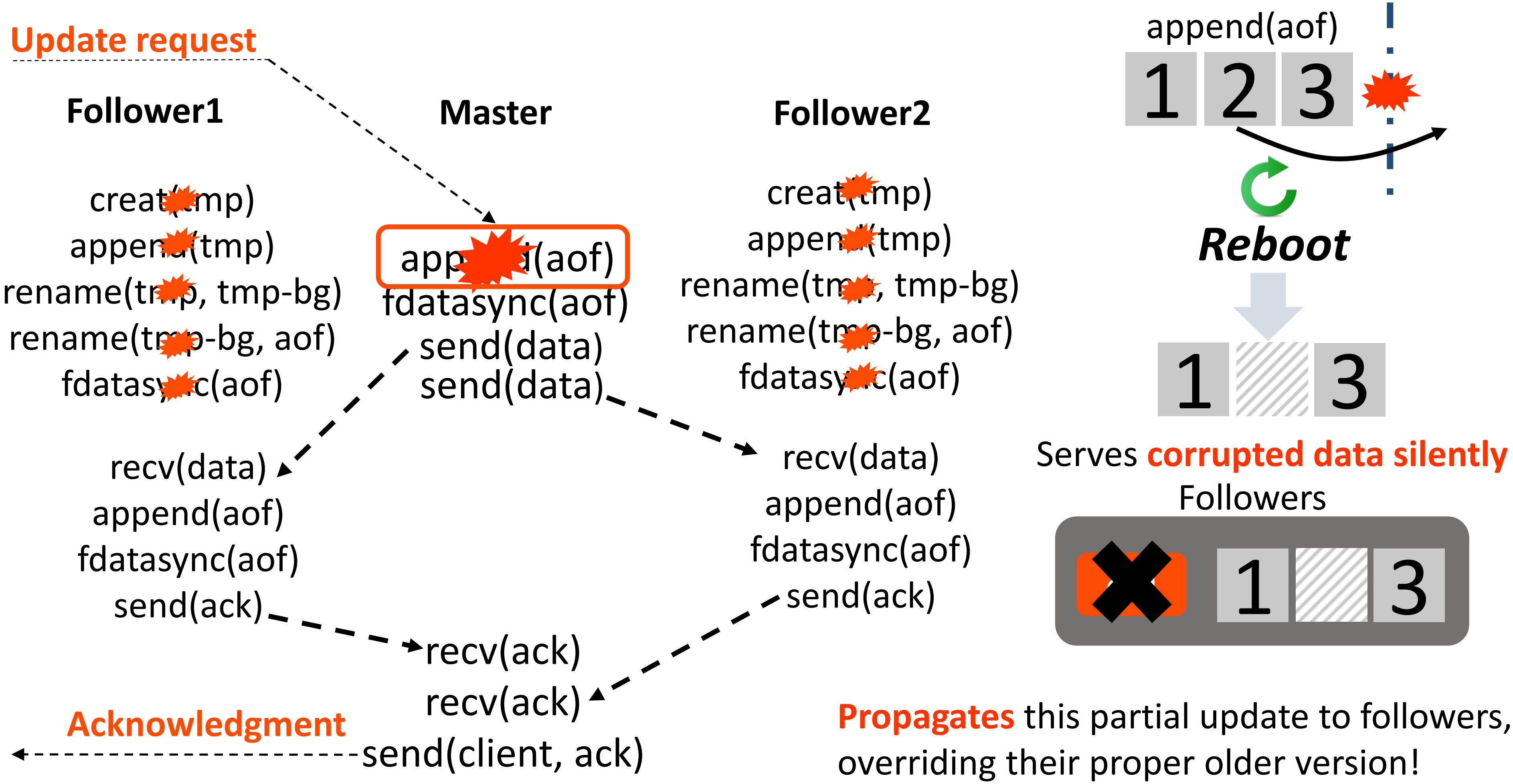
Example: Redis



Example: Redis

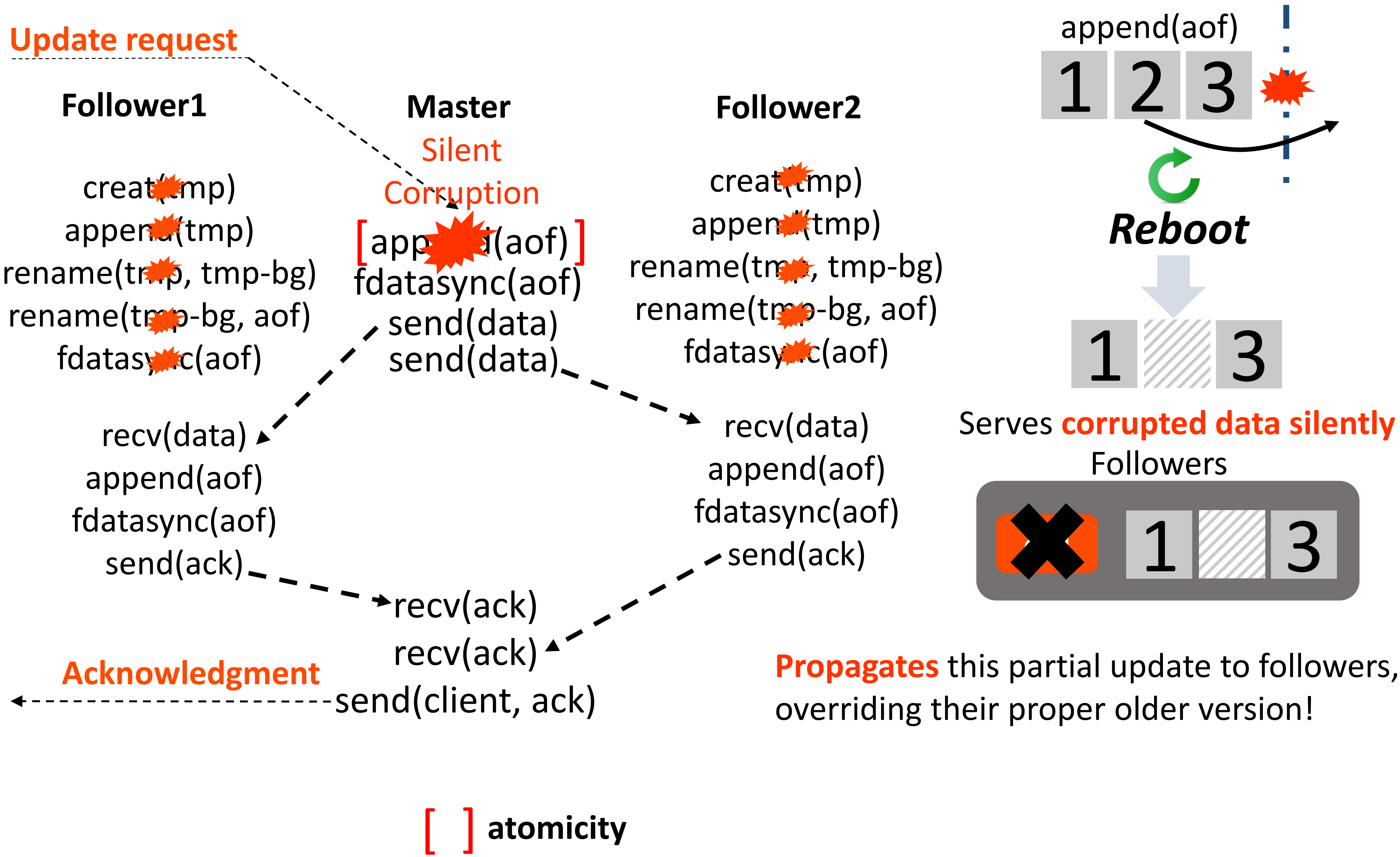


Example: Redis

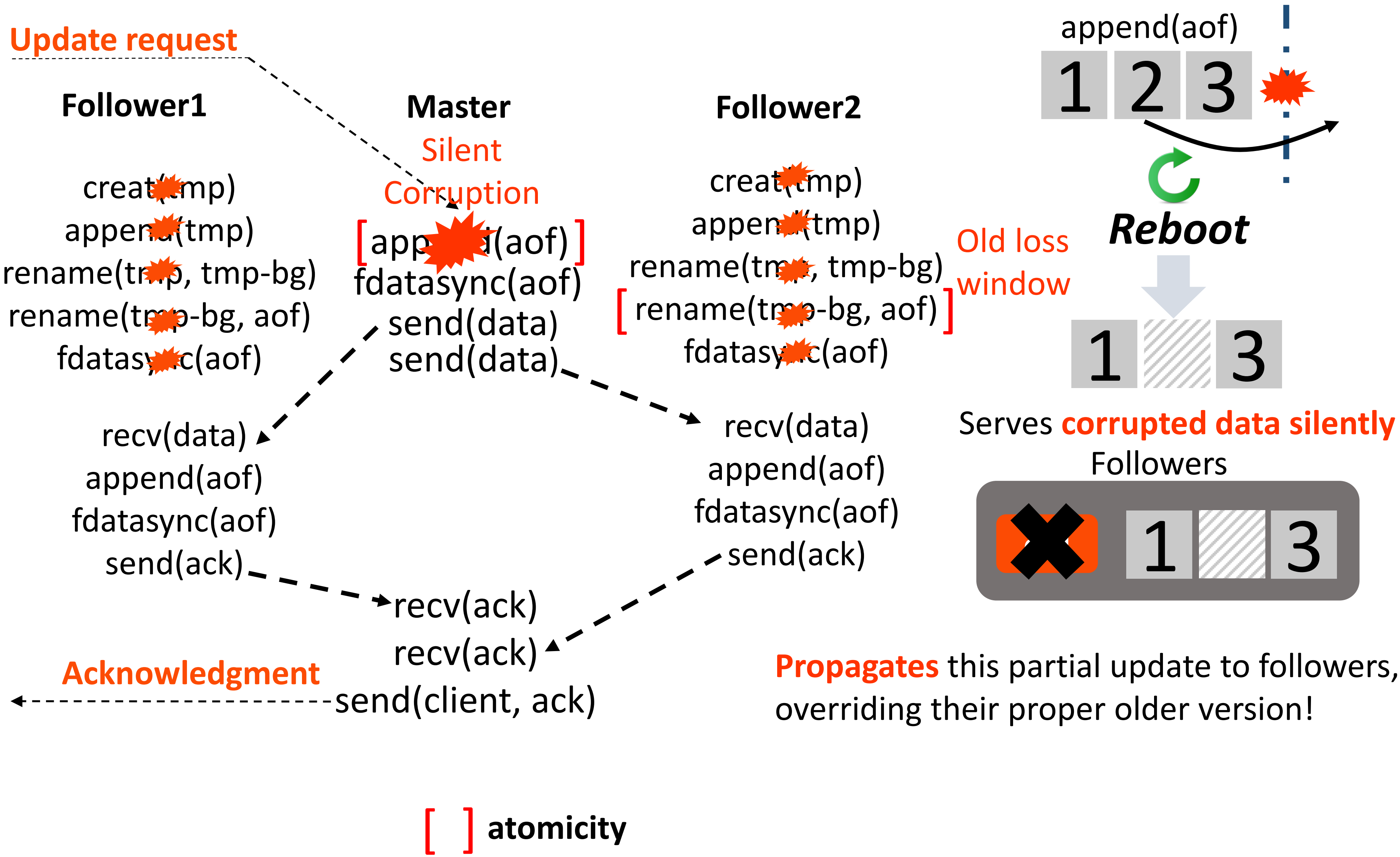


Propagates this partial update to followers, overriding their proper older version!

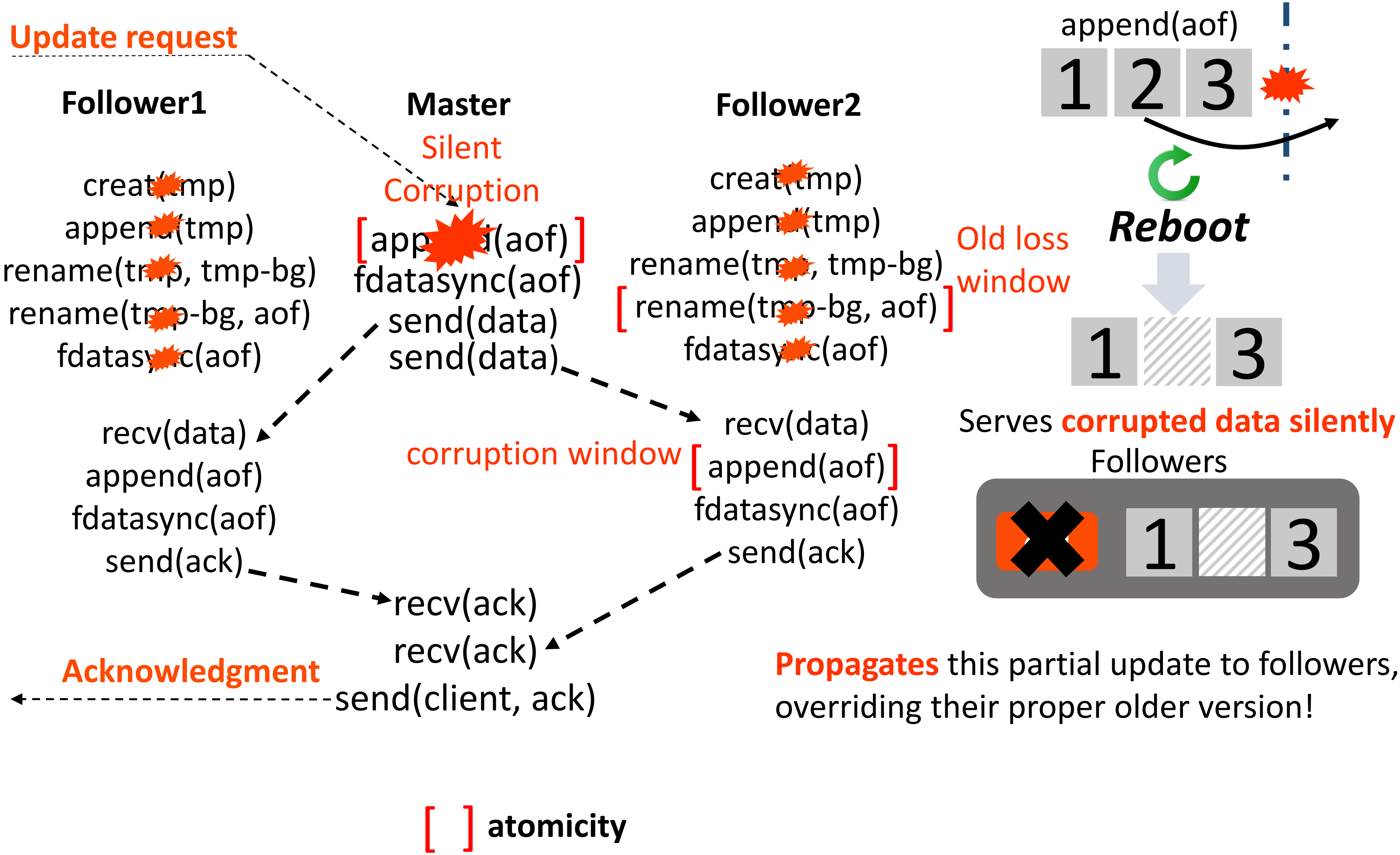
Example: Redis



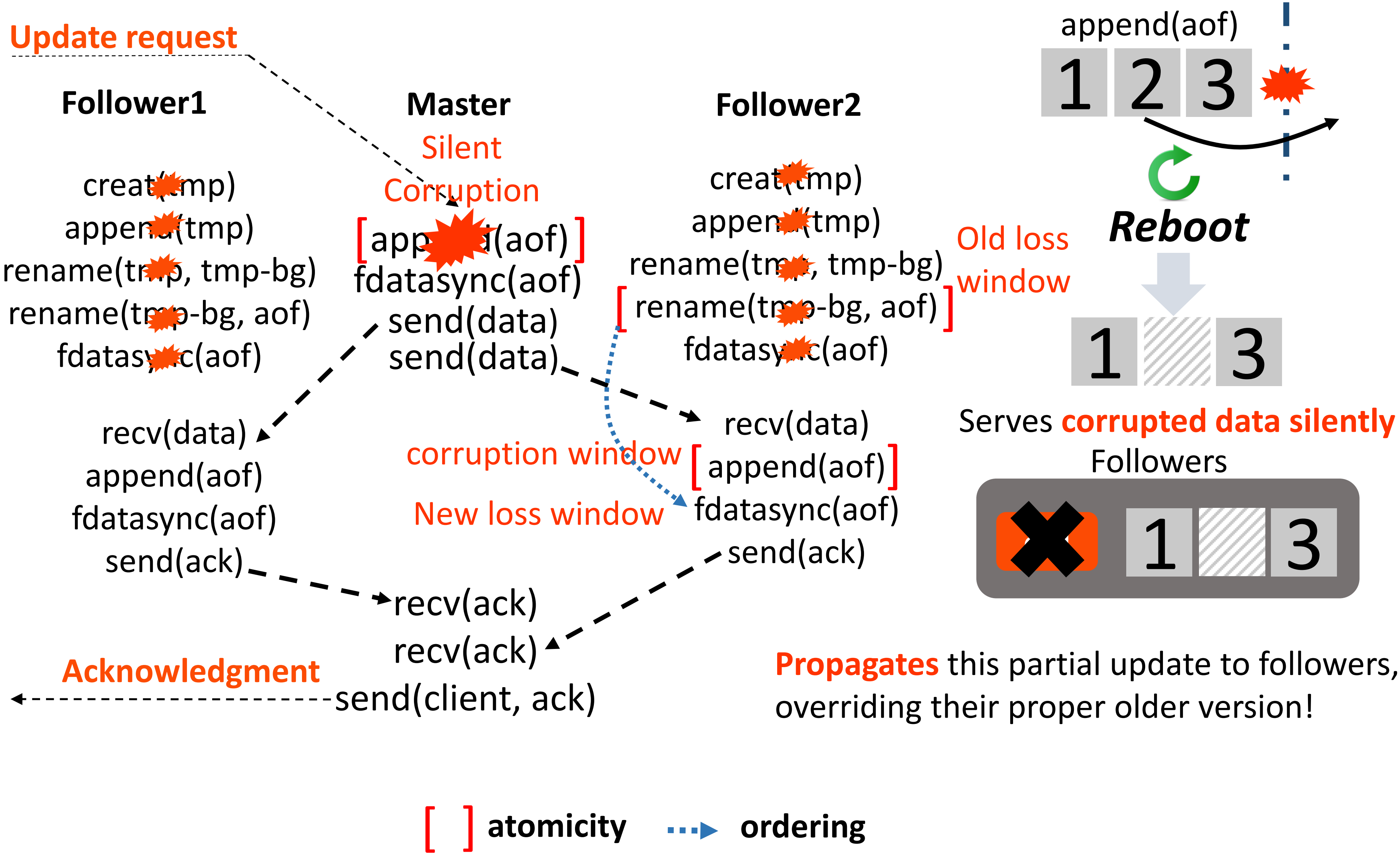
Example: Redis



Example: Redis



Example: Redis



Example: Redis

Update request

Follower1

Master

Follower2

append(aof)

1 2 3

1. File-system crash behaviors impact distributed storage systems

2. Problems in local storage protocols are not fixed by distributed recovery protocols

send(client, ack)

overriding their proper older version!

[ ] **atomicity** ordering

Vulnerability Consequences

Vulnerability Consequences

On ext2 – weak crash guarantees

Vulnerability Consequences

On ext2 – weak crash guarantees

Silent Corruption	Data Loss		Cluster Unavailability		Window		Total
	Old Commit	New Commit	Metadata Corruption	User Data Corruption	Silent Corruption	Data Loss	

Vulnerability Consequences

On ext2 – weak crash guarantees

Silent Corruption	Data Loss		Cluster Unavailability		Window		Total
	Old Commit	New Commit	Metadata Corruption	User Data Corruption	Silent Corruption	Data Loss	

Vulnerability Consequences

On ext2 – weak crash guarantees

Silent Corruption	Data Loss		Cluster Unavailability		Window		Total
	Old Commit	New Commit	Metadata Corruption	User Data Corruption	Silent Corruption	Data Loss	

Vulnerability Consequences

On ext2 – weak crash guarantees

Silent Corruption	Data Loss		Cluster Unavailability		Window		Total
	Old Commit	New Commit	Metadata Corruption	User Data Corruption	Silent Corruption	Data Loss	

Vulnerability Consequences

On ext2 – weak crash guarantees

Silent Corruption	Data Loss		Cluster Unavailability		Window		Total
	Old Commit	New Commit	Metadata Corruption	User Data Corruption	Silent Corruption	Data Loss	

Vulnerability Consequences

On ext2 – weak crash guarantees

	Silent Corruption	Data Loss		Cluster Unavailability		Window		Total
		Old Commit	New Commit	Metadata Corruption	User Data Corruption	Silent Corruption	Data Loss	
Redis	1					1	2	3
ZooKeeper			1	4	1			6
LogCabin		1		1			1	2
etcd			1	1	2			3
MongoDB-WT				1				1
MongoDB-Rocks			3	3				5
Kafka			3					3
iNexus		1	1	2				3

Vulnerability Consequences

On ext2 – weak crash guarantees

	Silent Corruption	Data Loss		Cluster Unavailability		Window		Total
		Old Commit	New Commit	Metadata Corruption	User Data Corruption	Silent Corruption	Data Loss	
Redis	1					1	2	3
ZooKeeper			1	4	1			6
LogCabin		1		1			1	2
etcd			1	1	2			3
MongoDB-WT				1				1
MongoDB-Rocks			3	3				5
Kafka			3					3
iNexus		1	1	2				3

Vulnerability Consequences

On ext2 – weak crash guarantees

	Silent Corruption	Data Loss		Cluster Unavailability		Window		Total
		Old Commit	New Commit	Metadata Corruption	User Data Corruption	Silent Corruption	Data Loss	
Redis	1					1	2	3
ZooKeeper			1	4	1			6
LogCabin		1		1			1	2
etcd			1	1	2			3
MongoDB-WT				1				1
MongoDB-Rocks			3	3				5
Kafka			3					3
iNexus		1	1	2				3

Vulnerability Consequences

On ext2 – weak crash guarantees

	Silent Corruption	Data Loss		Cluster Unavailability		Window		Total
		Old Commit	New Commit	Metadata Corruption	User Data Corruption	Silent Corruption	Data Loss	
Redis	1					1	2	3
ZooKeeper			1	4	1			6
LogCabin		1		1			1	2
etcd			1	1	2			3
MongoDB-WT				1				1
MongoDB-Rocks			3	3				5
Kafka			3					3
iNexus		1	1	2				3

Vulnerabilities on Real File Systems

System	ext2	ext3-w	ext3-o	ext4-o	ext3-j	btrfs
Redis	3	1				1
ZooKeeper	6	3	1	1	1	3
LogCabin	2	1	1	1	1	1
etcd	3	2				
MongoDB-WT	1					
MongoDB-R	5	2	2	2		3
Kafka	3					
iNexus	2		1	1		2
Total	**26**	**9**	**5**	**5**	**2**	**10**

Vulnerabilities on Real File Systems

System	ext2	ext3-w	ext3-o	ext4-o	ext3-j	btrfs
Redis	3	1				1
ZooKeeper	6	3	1	1	1	3
LogCabin	2	1	1	1	1	1
etcd	3	2				
MongoDB-WT	1					
MongoDB-R	5	2	2	2		3
Kafka	3					
iNexus	2		1	1		2
Total	**26**	**9**	**5**	**5**	**2**	**10**

Vulnerabilities on Real File Systems

System	ext2	ext3-w	ext3-o	ext4-o	ext3-j	btrfs
Redis	3	1				1
ZooKeeper	6	3	1	1	1	3
LogCabin	2	1	1	1	1	1
etcd	3	2				
MongoDB-WT	1					
MongoDB-R	5	2	2	2		3
Kafka	3					
iNexus	2		1	1		2
Total	**26**	**9**	**5**	**5**	**2**	**10**

Vulnerabilities on Real File Systems

System	ext2	ext3-w	ext3-o	ext4-o	ext3-j	btrfs
Redis	3	1				1
ZooKeeper	6	3	1	1	1	3
LogCabin	2	1	1	1	1	1
etcd	3	2				
MongoDB-WT	1					
MongoDB-R	5	2	2	2		3
Kafka	3					
iNexus	2		1	1		2
Total	**26**	**9**	**5**	**5**	**2**	**10**

Vulnerabilities on Real File Systems

System	ext2	ext3-w	ext3-o	ext4-o	ext3-j	btrfs
Redis	3	1				1
ZooKeeper	6	3	1	1	1	3
LogCabin	2	1	1	1	1	1
etcd	3	2				
MongoDB-WT	1					
MongoDB-R	5	2	2	2		3
Kafka	3					
iNexus	2		1	1		2
Total	**26**	**9**	**5**	**5**	**2**	**10**

Vulnerabilities on Real File Systems

System	ext2	ext3-w	ext3-o	ext4-o	ext3-j	btrfs
Redis	3	1				1
ZooKeeper	6	3	1	1	1	3
LogCabin	2	1	1	1	1	1
etcd	3	2				
MongoDB-WT	1					
MongoDB-R	5	2	2	2		3
Kafka	3					
iNexus	2		1	1		2
Total	**26**	**9**	**5**	**5**	**2**	**10**

Vulnerabilities on Real File Systems

System	ext2	ext3-w	ext3-o	ext4-o	ext3-j	btrfs
Redis	3	1				1
ZooKeeper	6	3	1	1	1	3
LogCabin	2	1	1	1	1	1
etcd	3	2				
MongoDB-WT	1					
MongoDB-R	5	2	2	2		3
Kafka	3					
iNexus	2		1	1		2
Total	**26**	**9**	**5**	**5**	**2**	**10**

Vulnerabilities on Real File Systems

System	ext2	ext3-w	ext3-o	ext4-o	ext3-j	btrfs
Redis	3	1				1
ZooKeeper	6	3	1	1	1	3
LogCabin	2	1	1	1	1	1
etcd	3	2				
MongoDB-WT	1					
MongoDB-R	5	2	2	2		3
Kafka	3					
iNexus	2		1	1		2
Total	**26**	**9**	**5**	**5**	**2**	**10**

Less vulnerabilities on ordered file systems

Summary

Summary

Distributed storage systems violate user-level expectations during correlated crashes

Summary

Distributed storage systems violate user-level expectations during correlated crashes

Popular, well-tested systems are vulnerable

Summary

Distributed storage systems violate user-level expectations during correlated crashes

Popular, well-tested systems are vulnerable

Local file system crash behaviors directly affect distributed storage systems

Summary

Distributed storage systems violate user-level expectations during correlated crashes

Popular, well-tested systems are vulnerable

Local file system crash behaviors directly affect distributed storage systems

In many cases, distributed recovery protocols do not fix problems in local storage protocols

Conclusion

Conclusion

Reliability is crucial in distributed storage systems – primary choice for storing large amounts of data

Conclusion

Reliability is crucial in distributed storage systems – primary choice for storing large amounts of data

Complex ways to fail and subtle interactions between components

Conclusion

Reliability is crucial in distributed storage systems – primary choice for storing large amounts of data

Complex ways to fail and subtle interactions between components

Conclusion

Reliability is crucial in distributed storage systems – primary choice for storing large amounts of data

Complex ways to fail and subtle interactions between components

Software and results soon @ http://research.cs.wisc.edu/adsl/Software/pace/

Conclusion

Reliability is crucial in distributed storage systems – primary choice for storing large amounts of data

Complex ways to fail and subtle interactions between components

Software and results soon @ http://research.cs.wisc.edu/adsl/Software/pace/

Thank you!