Torturing Databases for Fun and Profit

Mai Zheng[†], Joseph Tucek[‡], Dachuan Huang[†], Feng Qin[†], Mark Lillibridge[‡]
Elizabeth S Yang[‡], Bill W Zhao[‡], Shashank Singh[†]

[†] *The Ohio State University*

[‡] *HP Labs*

database








- ACID: atomicity, consistency, isolation, and durability - even under failures

List of databases survived

Everything is broken under simulated power faults

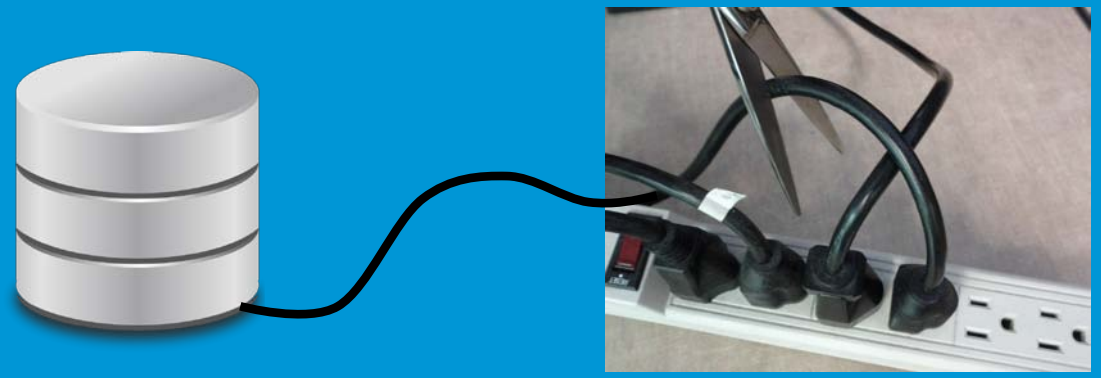
Database	File System	Workload					
		W-1	W-2	W-3	W-4.1	W-4.2	W-4.3
TokyoCabinet	ext3	D	D	D	ACD	ACD	ACD
	XFS	--	D	D	ACD	D	ACD
MariaDB	ext3	D	D	D	D	D	D
	XFS	D	D	D	D	D	D
LightningDB	ext3	--	--	--	--	--	D
	XFS	--	--	--	--	--	--
SQLite	ext3	D	D	--	D	D	D
	XFS	--	--	D	D	D	D
KVS-A	ext3	--	--	Hang	--	--	--
	XFS	--	--	--	--	--	--
SQL-A	ext3	D	D	D	D	D	D
	XFS	D	D	D	D	D	D
SQL-B	ext3	D	D	CD	CD	CD	CD
	XFS	CD	D	CD	CD	CD	CD
SQL-C	NTFS	D	D	D	D	D	D

Power faults cannot happen nowadays, right?

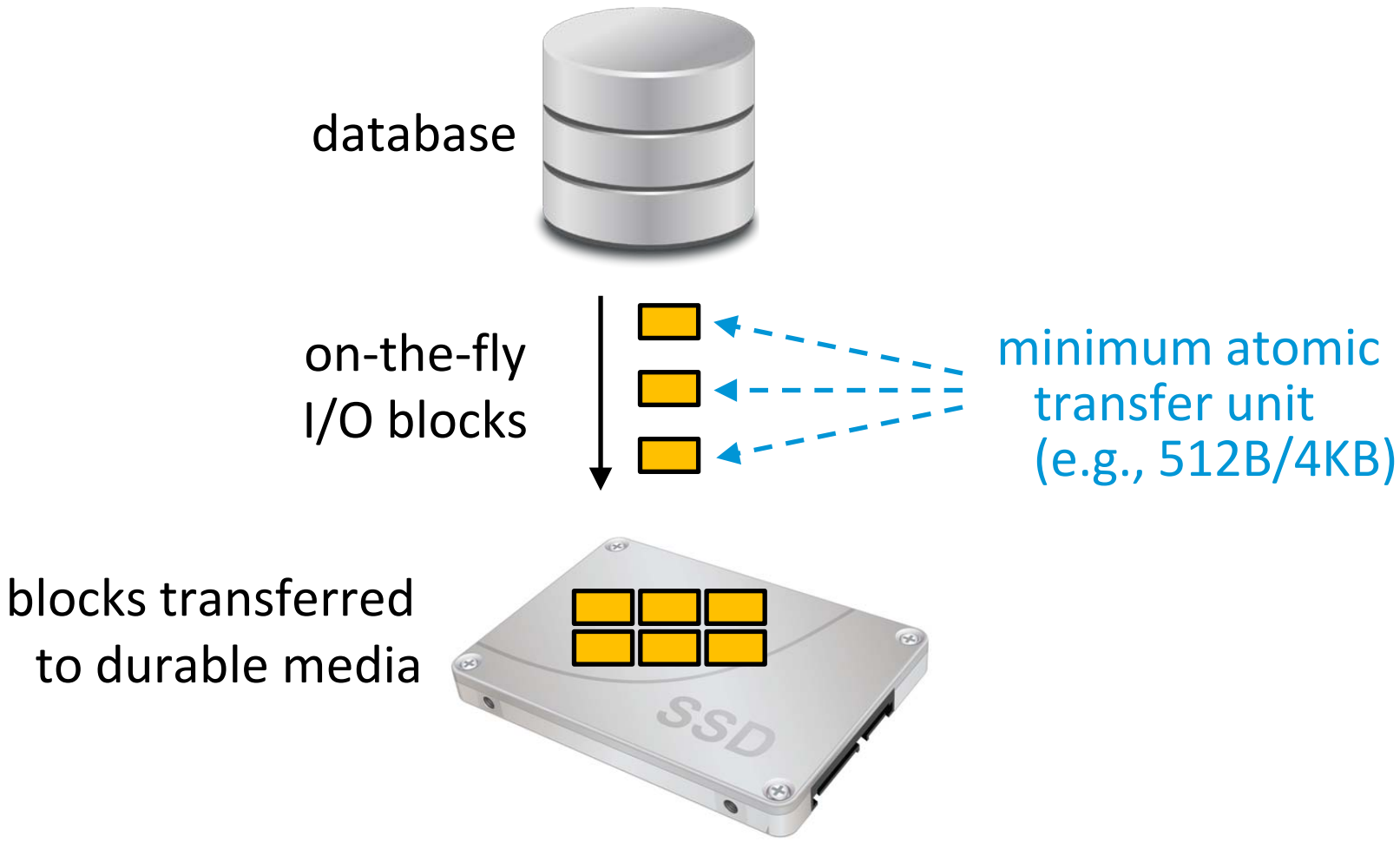

2013:"... POWER OUTAGE during Super Bowl ... because a RELAY DEVICE MALFUNCTIONED."


2014:“Internap Data Center OUTAGE Takes Down Livestream, StackExchange”
2014:“Data Center FIRE Leads to OUTAGE ...”
2014:“... ELECTRICAL FIRE took down ... primary data center... ALL POWER was OFF ...”
2013:“POWER OUTAGE knocks DreamHost customers offline ...”
2013:“ A data center POWER OUTAGE is being blamed for ...Visa downtime ...”
2013:“ A POWER OUTAGE at a key New Jersey data center ...”
2013:“... POWER OUTAGE during Super Bowl ... because a RELAY DEVICE MALFUNCTIONED.”
2012:“... HUNAM ERROR was responsible for a data center POWER OUTAGE ...”
2012:“POWER OUTAGE Hits London Data Center ...”
2012:“Amazon Data Center LOSES POWER During STORM ...”
2011:“Colocation provider Colo4 experienced a POWER OUTAGE ...”
2010:“About 3,000 servers at Montreal web host iWeb experienced an OUTAGE ...”
2010:“CAR CRASH Triggers Amazon POWER OUTAGE ...”

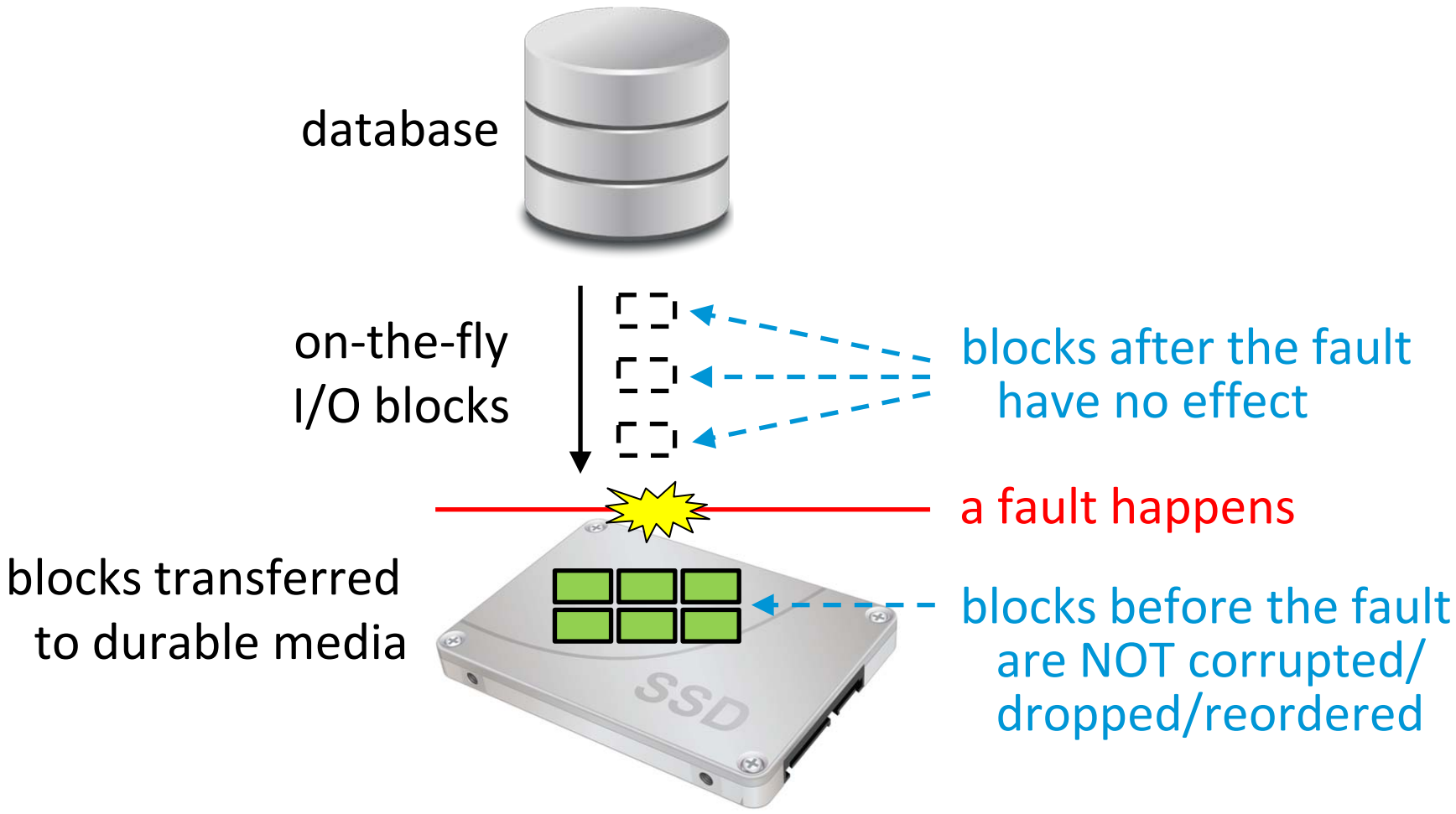
Database Torture 101



Fault Model: Clean termination of I/O stream



Fault Model: Clean termination of I/O stream

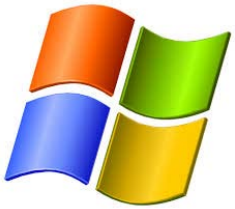


Why not introduce corruption/dropping/reordering?

- Unreasonable to require databases to handle arbitrary bad behavior introduced in the lower layers
- Simulated bad behavior w/o verification by real failures may be unrealistic
  - I/O path in kernel & device puts constraints on failure states

How do we test DBs on multiple OSes w/ high fidelity?

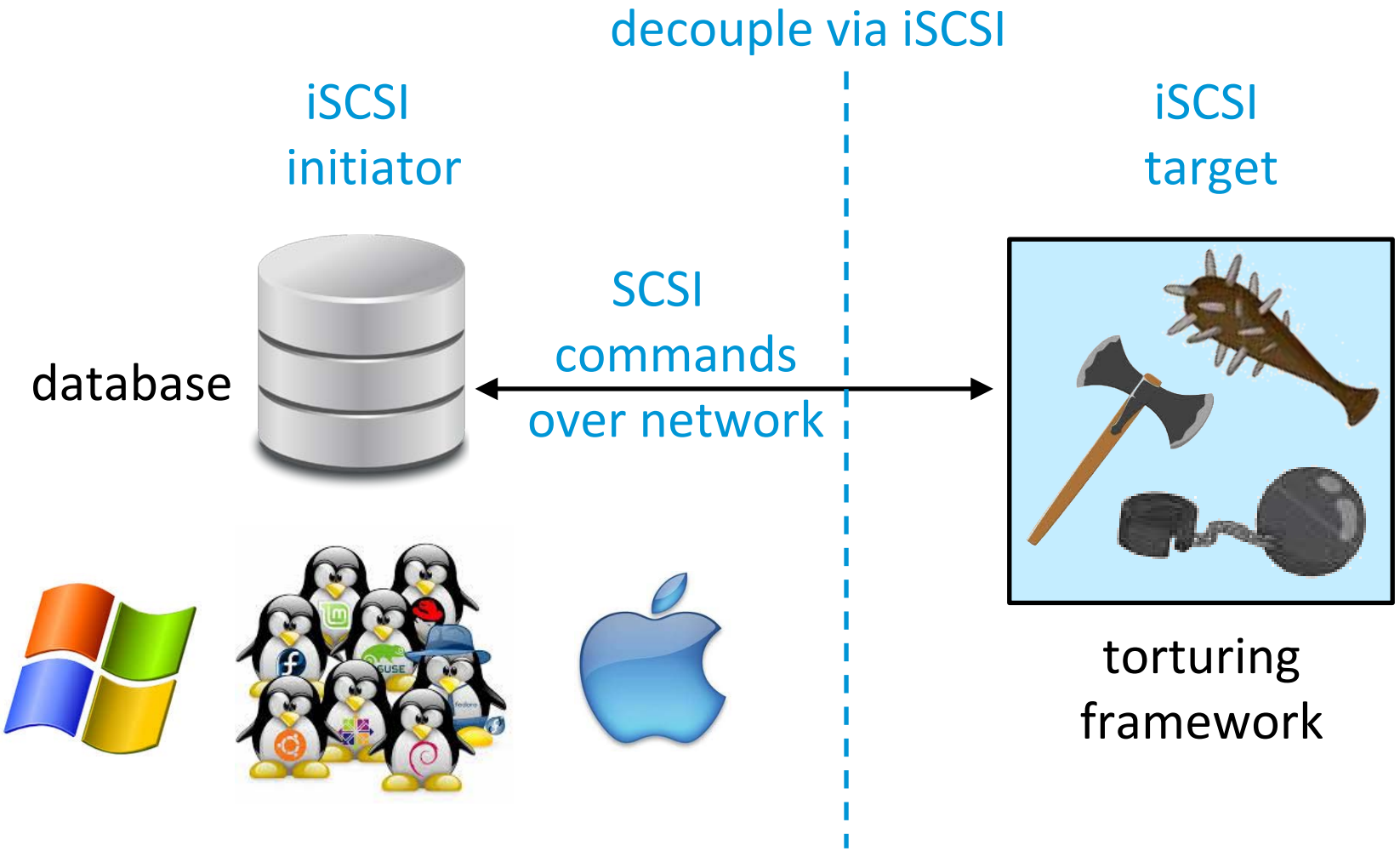
database



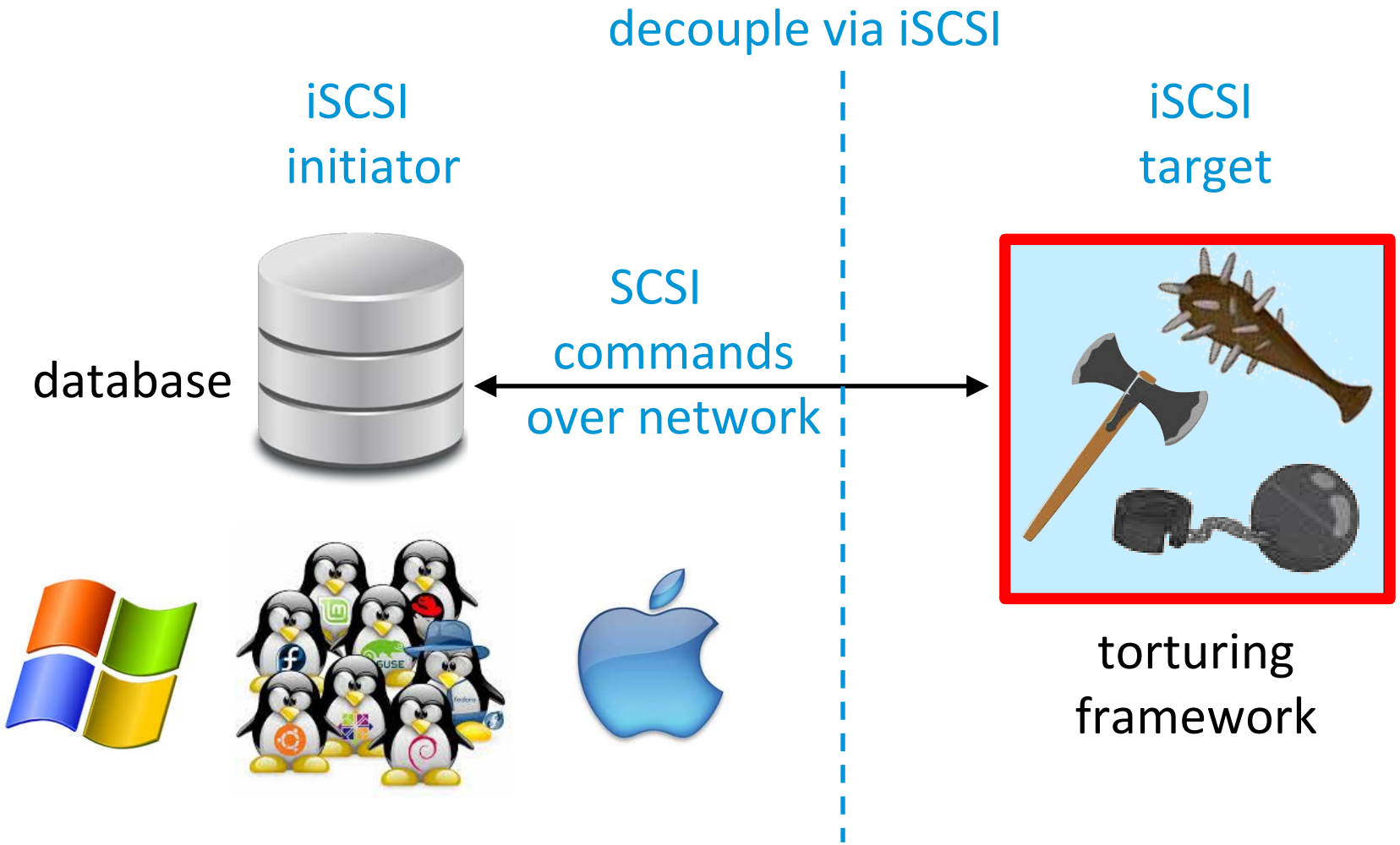
No disturbance on thread scheduling

No disturbance on interactions among DB, memory manager, FS, volume manager, I/O scheduler, …

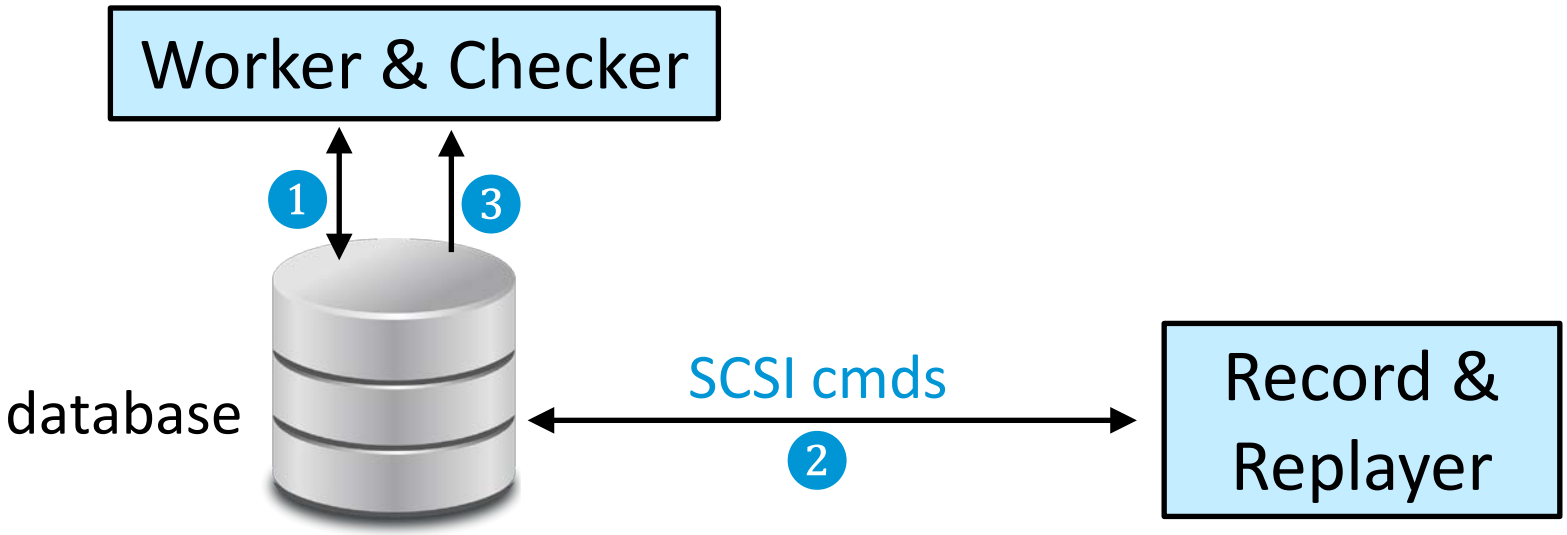
How do we test DBs on multiple OSes w/ high fidelity?



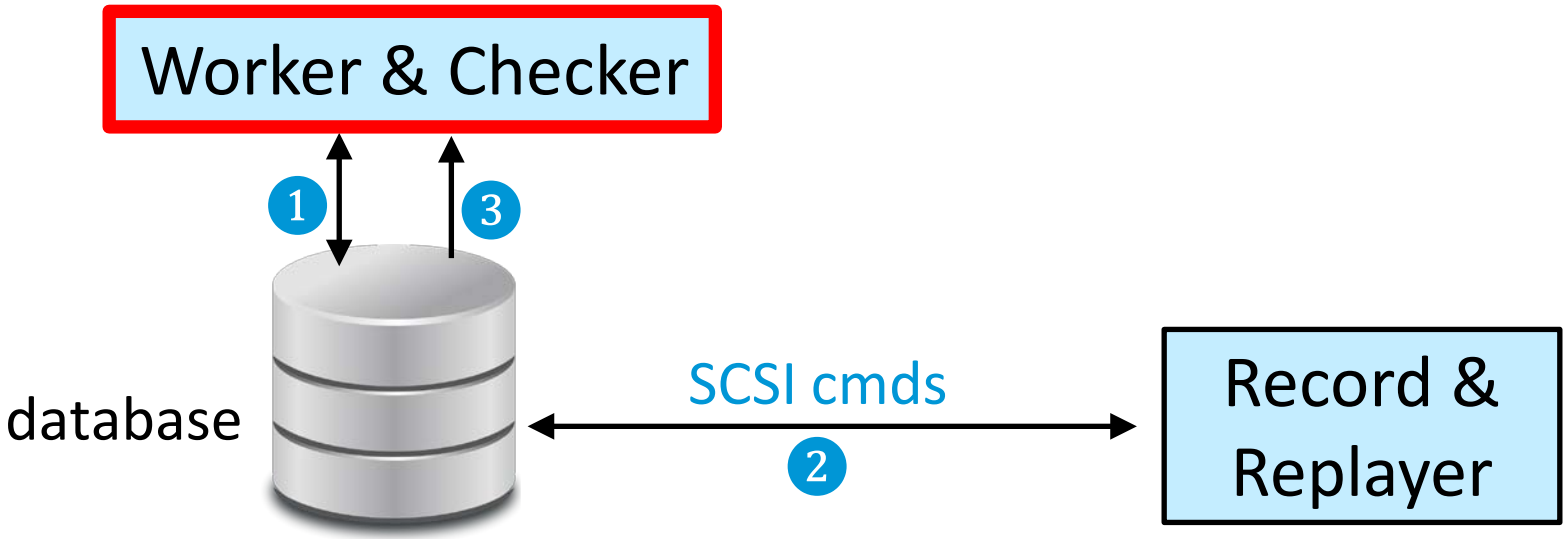
How do we test DBs on multiple OSes w/ high fidelity?



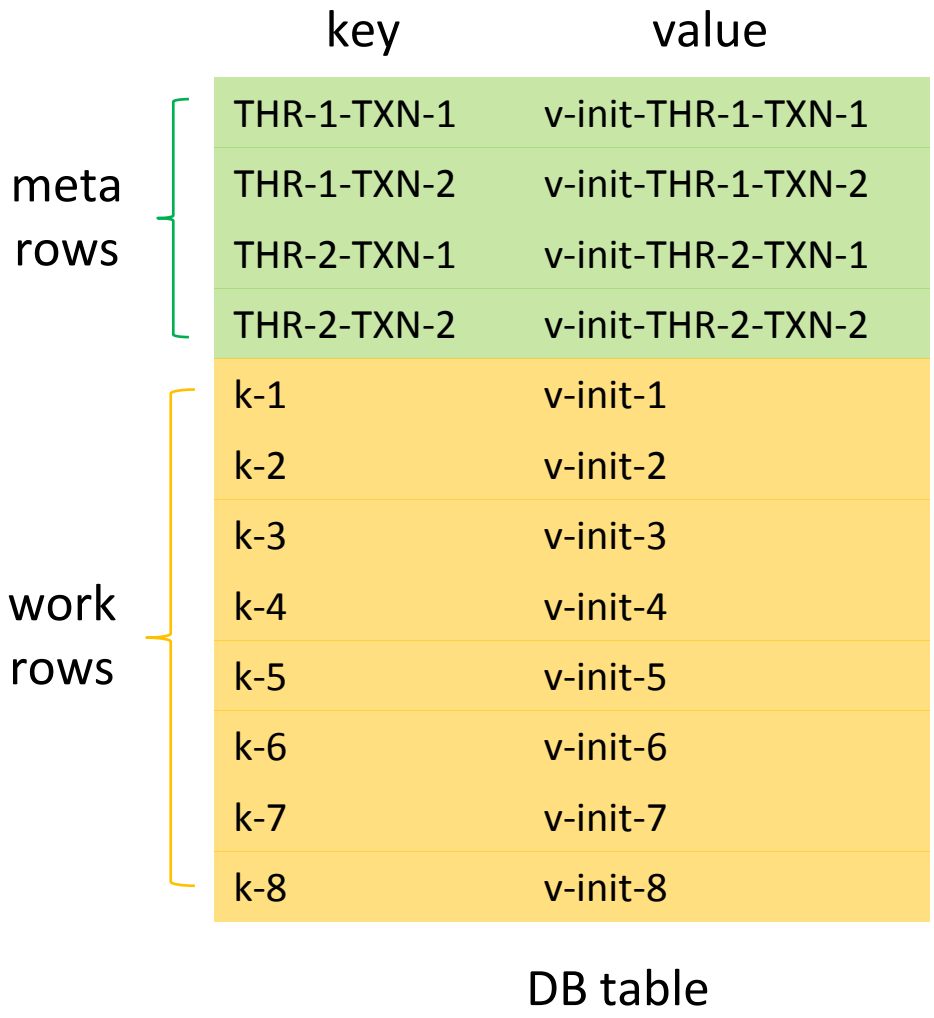
Framework Overview



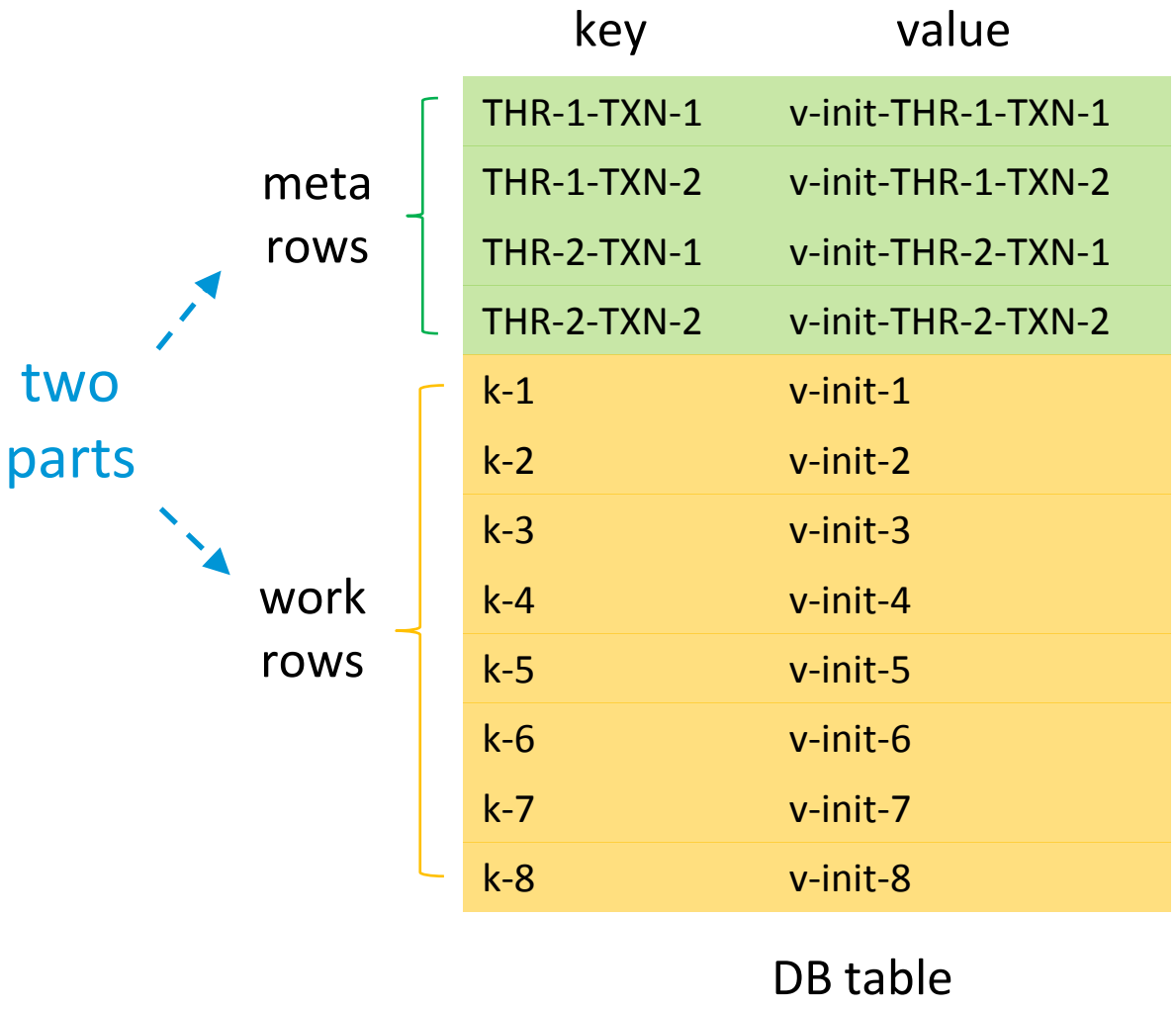
Framework Overview



Workload Example

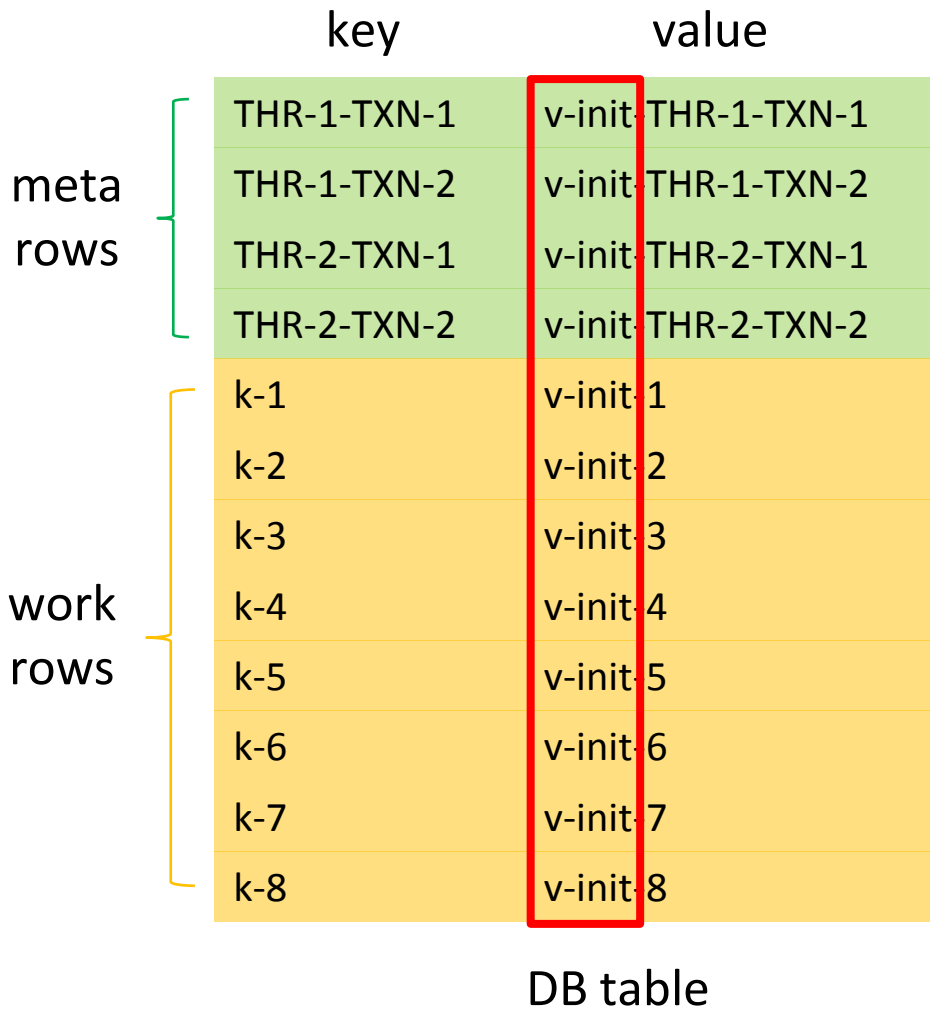


Workload Example

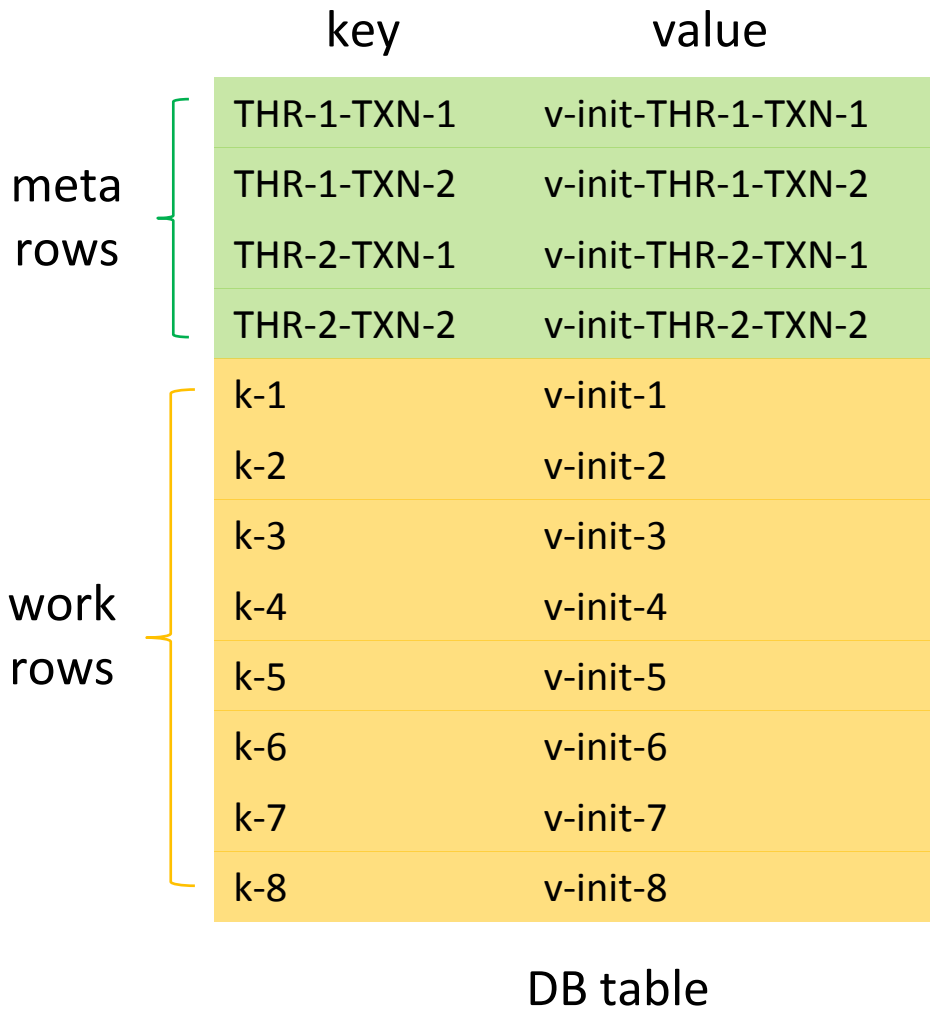


2 threads,
2 transactions
per thread

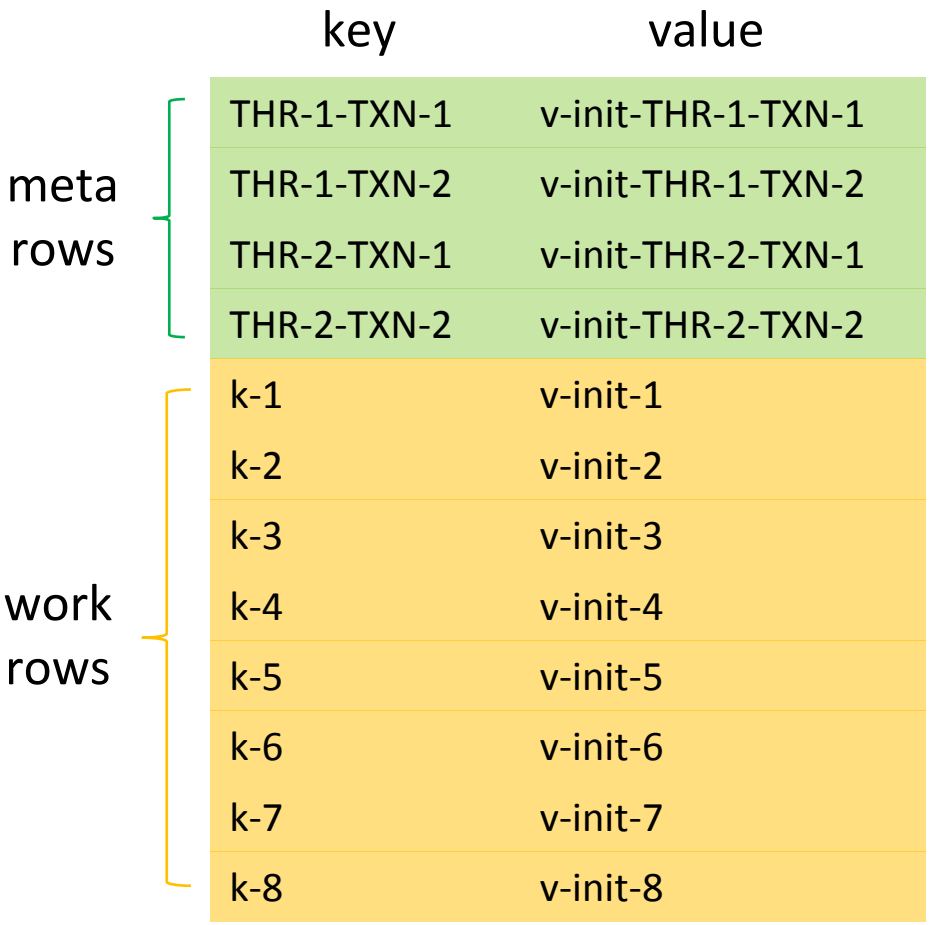
Has known initial state



Each transaction updates N random work rows + 1 meta row



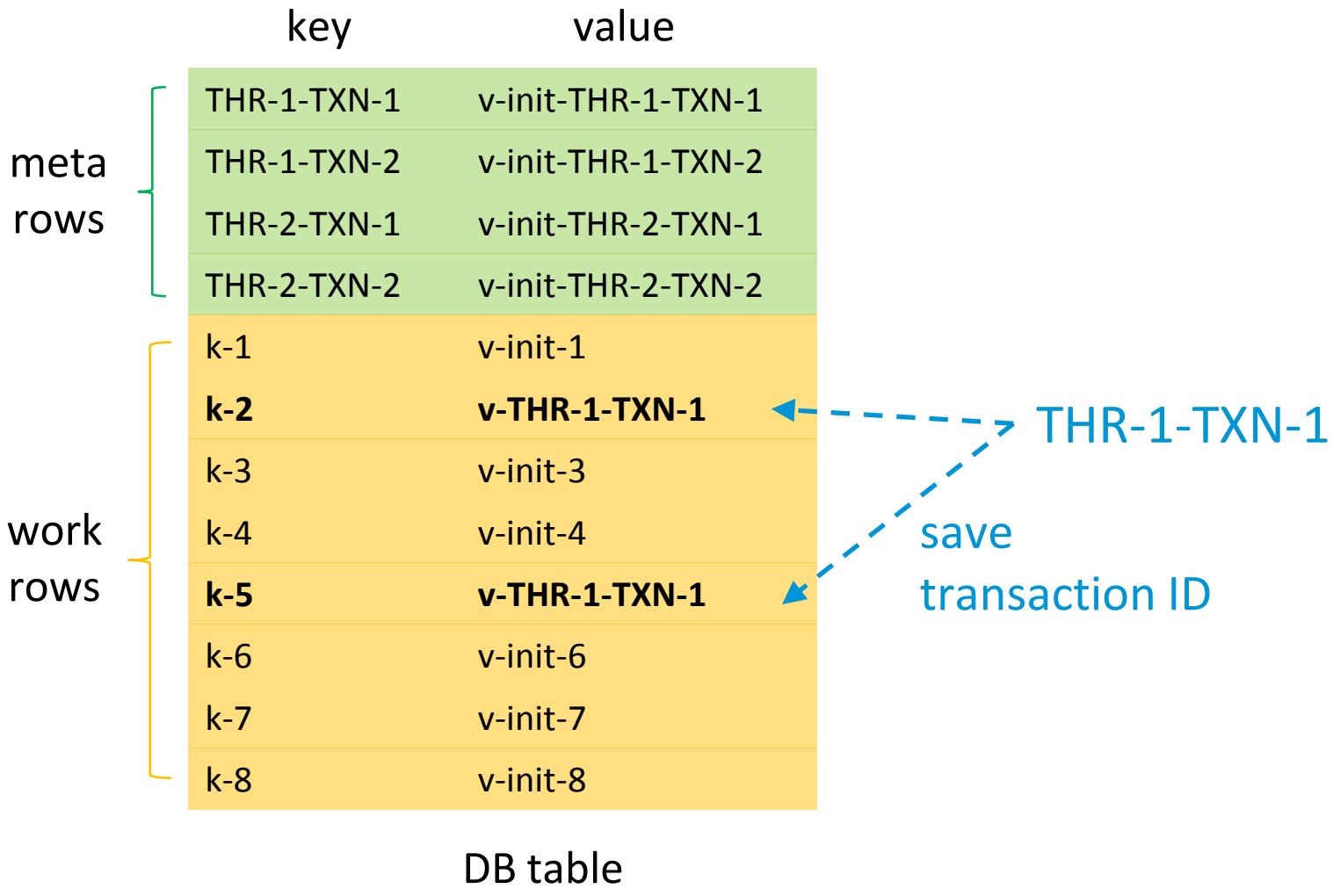
Each transaction updates N random work rows + 1 meta row



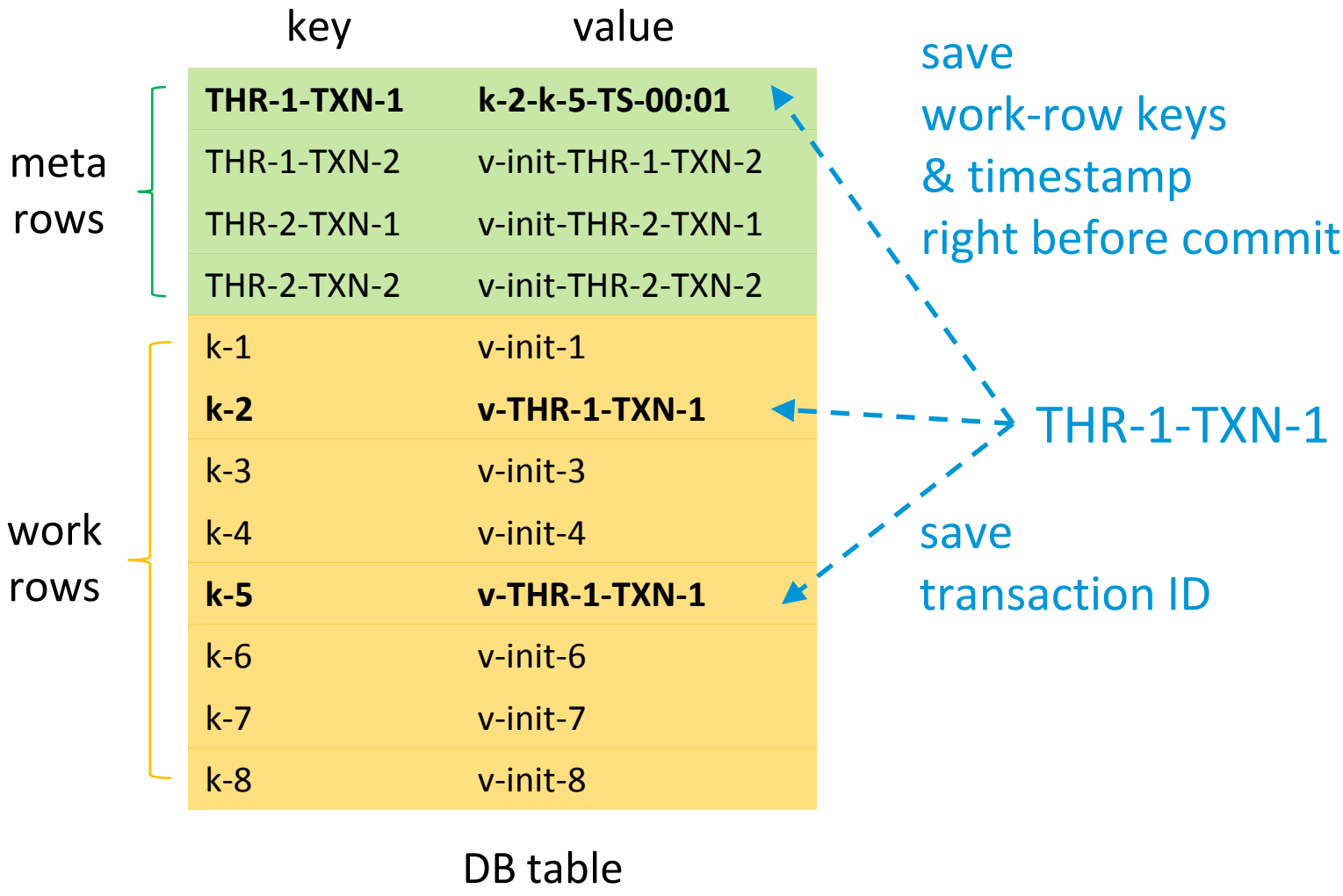
DB table

THR-1-TXN-1

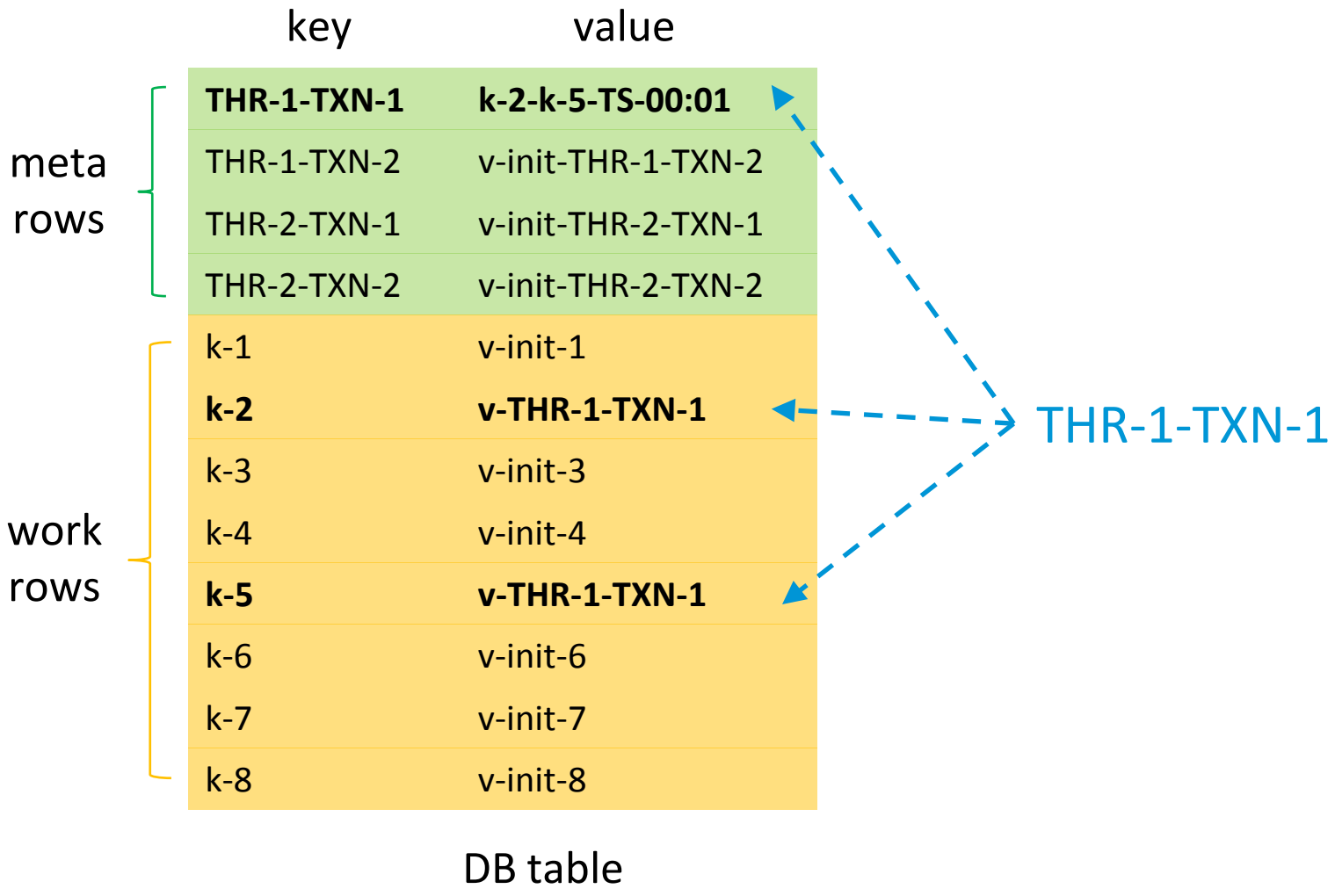
Each transaction updates N random work rows + 1 meta row



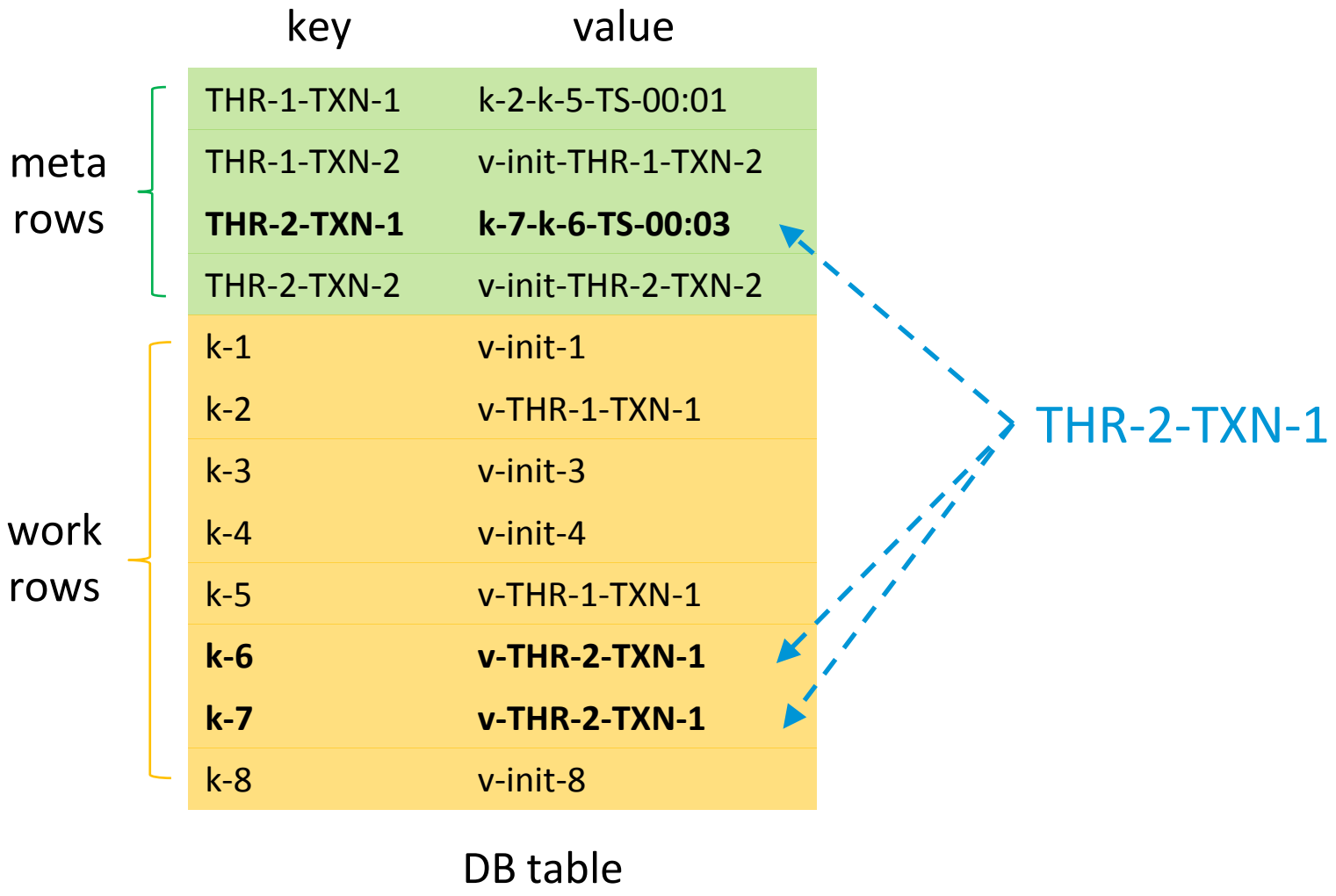
Each transaction updates N random work rows + 1 meta row



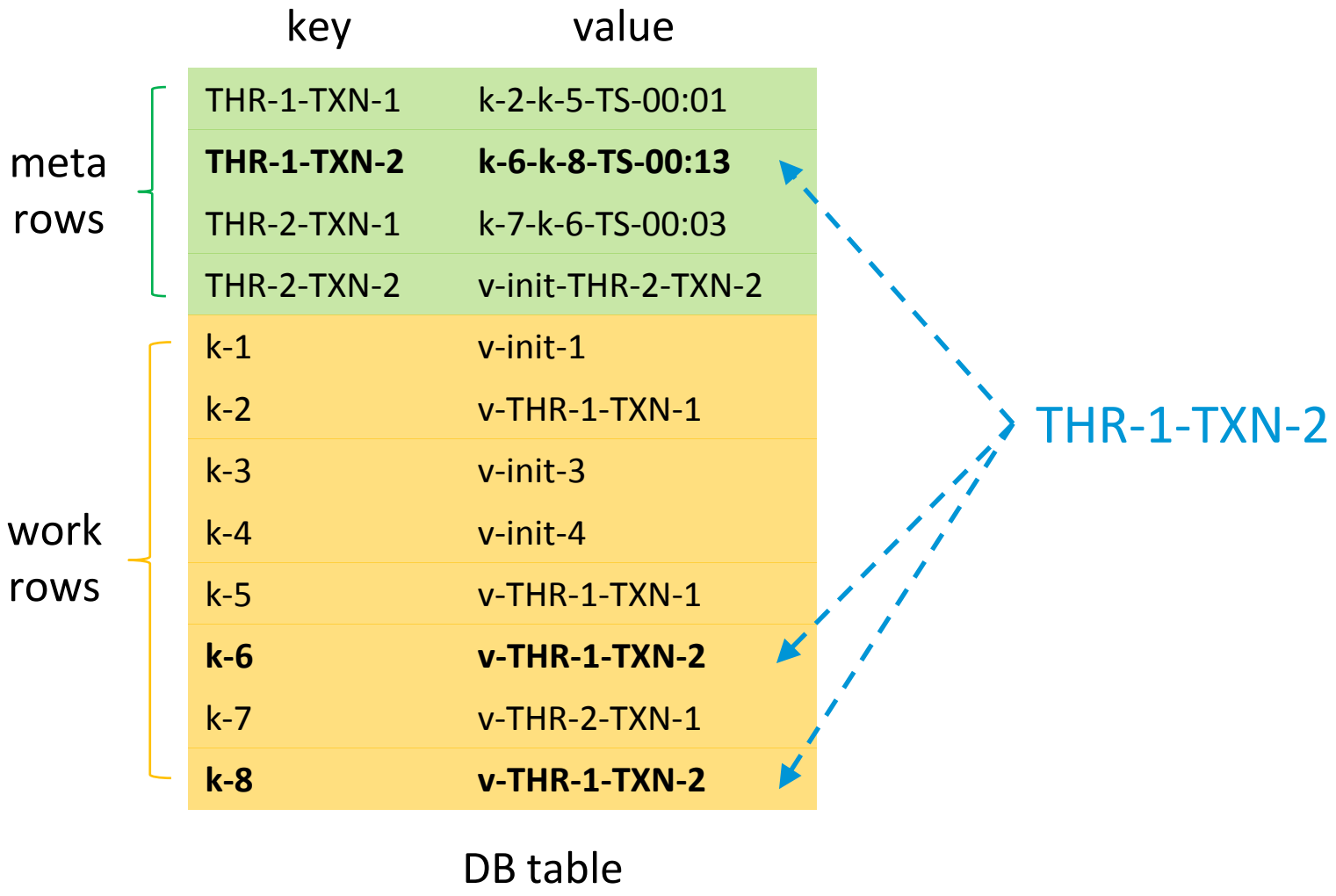
Fully exercise concurrency control



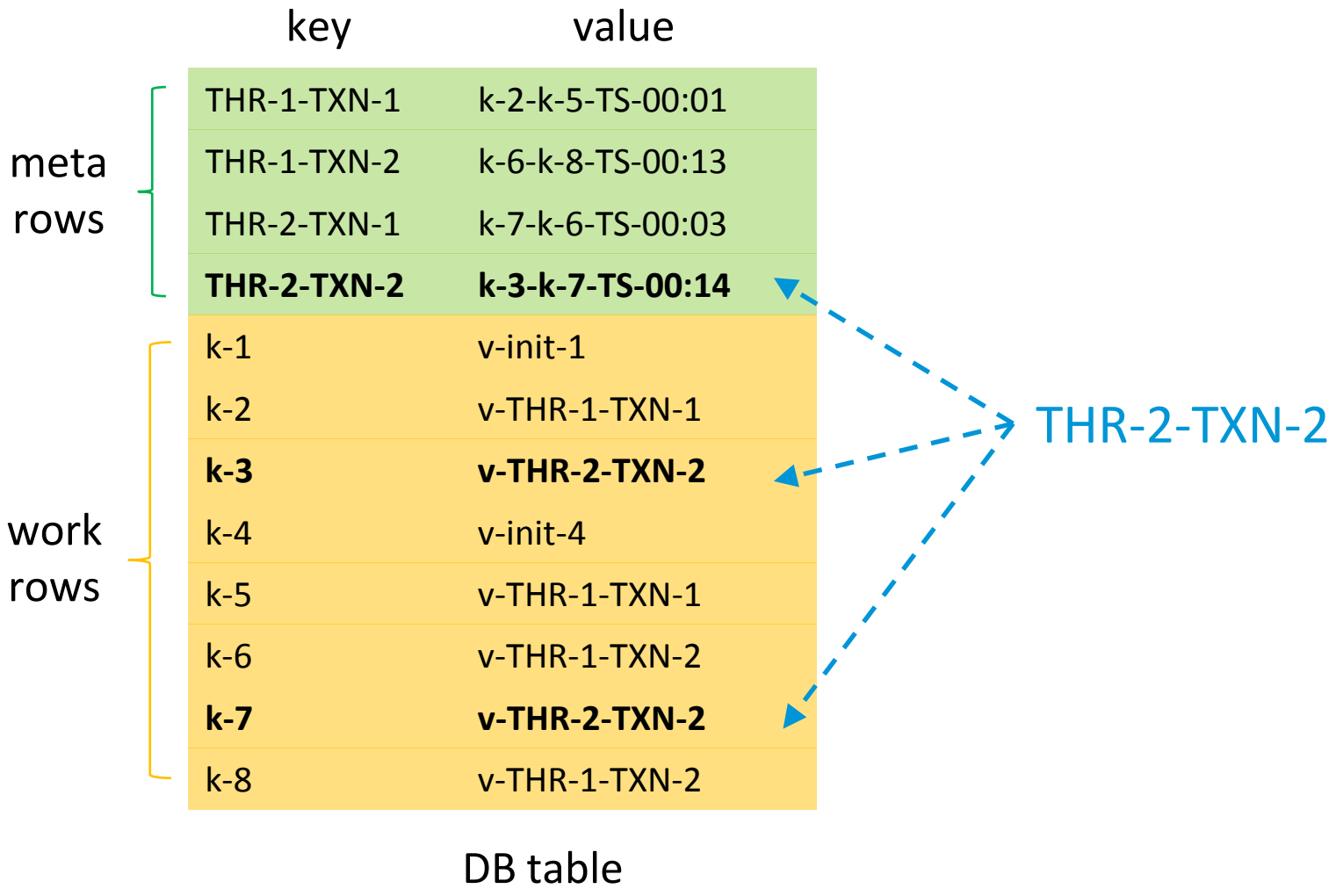
Fully exercise concurrency control



Fully exercise concurrency control

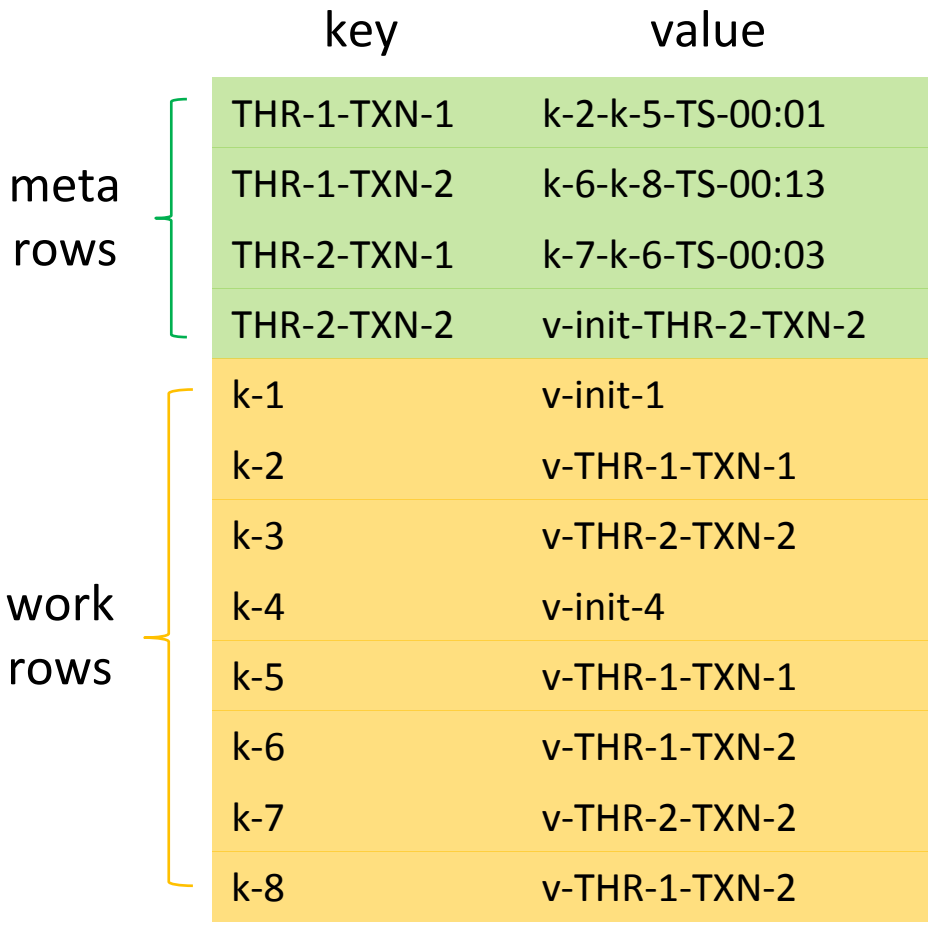


Fully exercise concurrency control



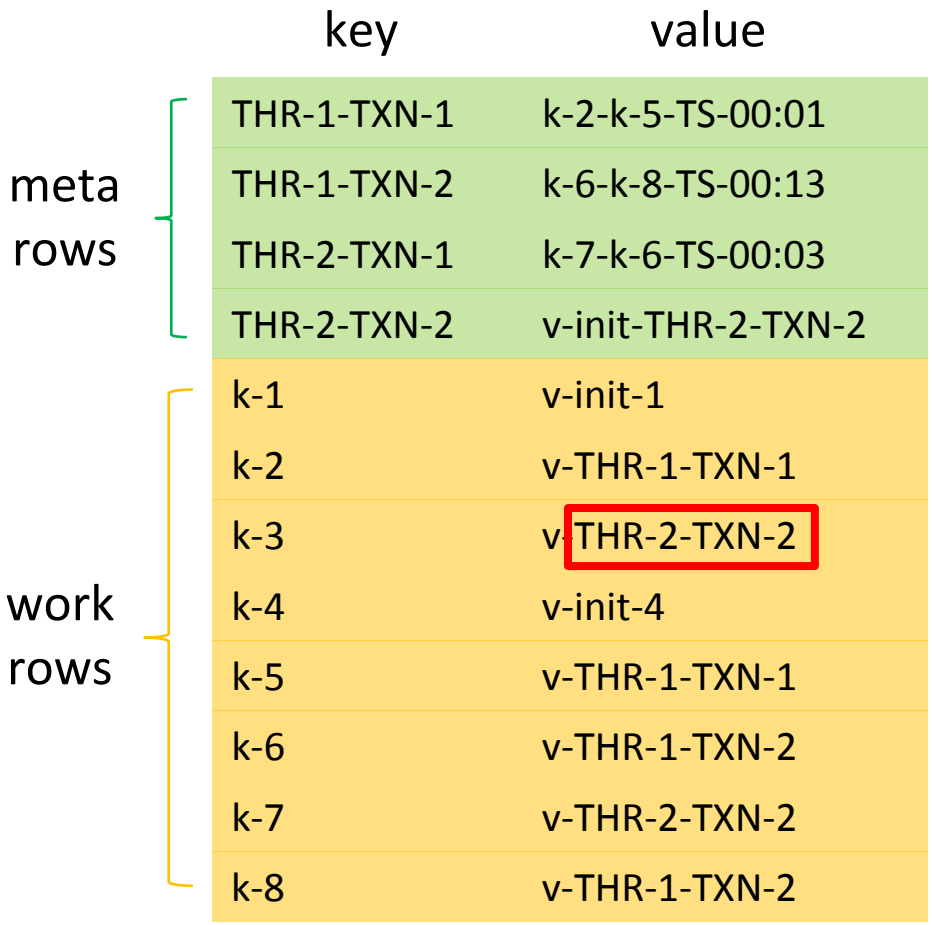
A power fault just happened during our workload ...

Is there any ACID violation after recovery?



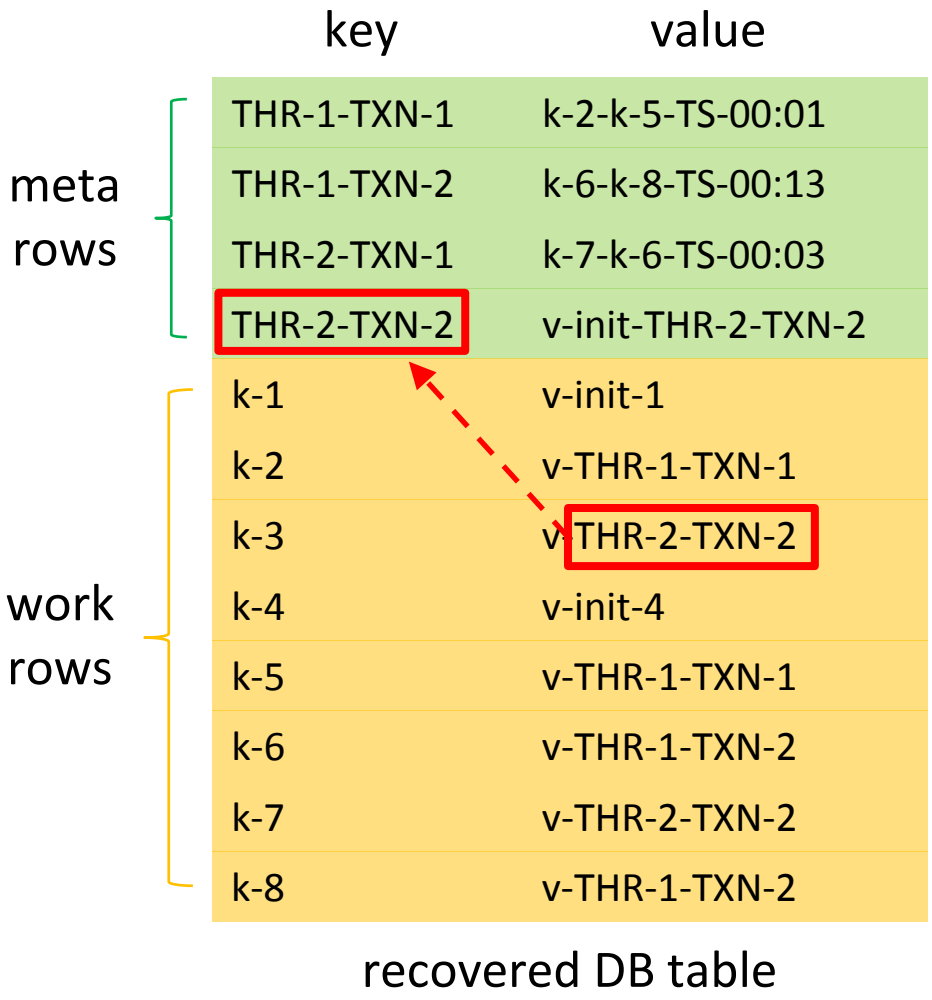
recovered DB table

Is there any ACID violation after recovery?



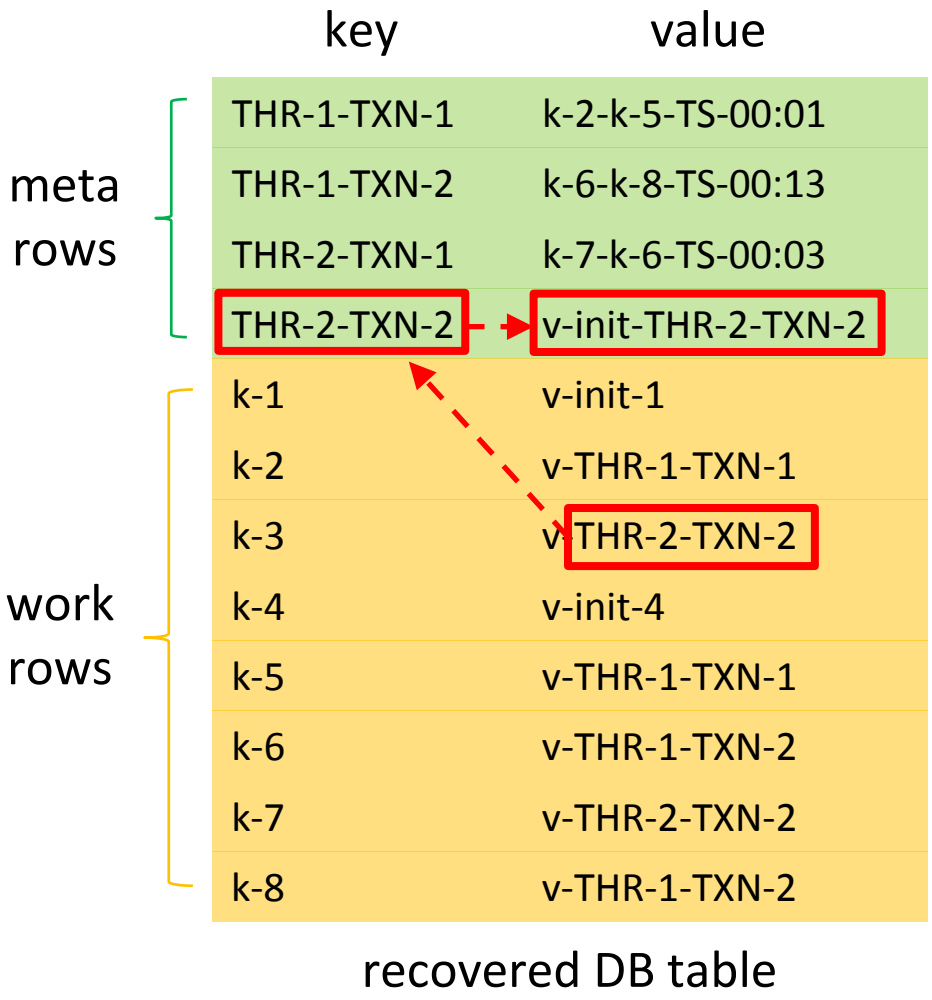
recovered DB table

Is there any ACID violation after recovery?



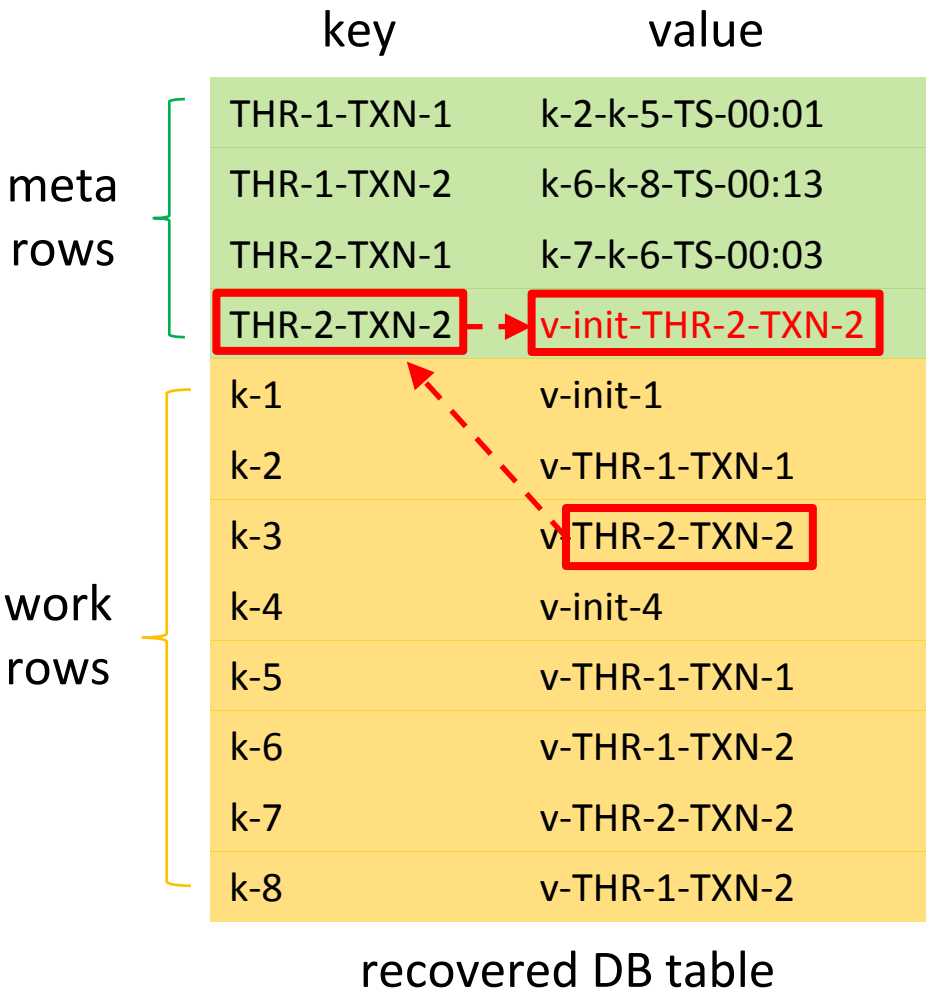
recovered DB table

Is there any ACID violation after recovery?



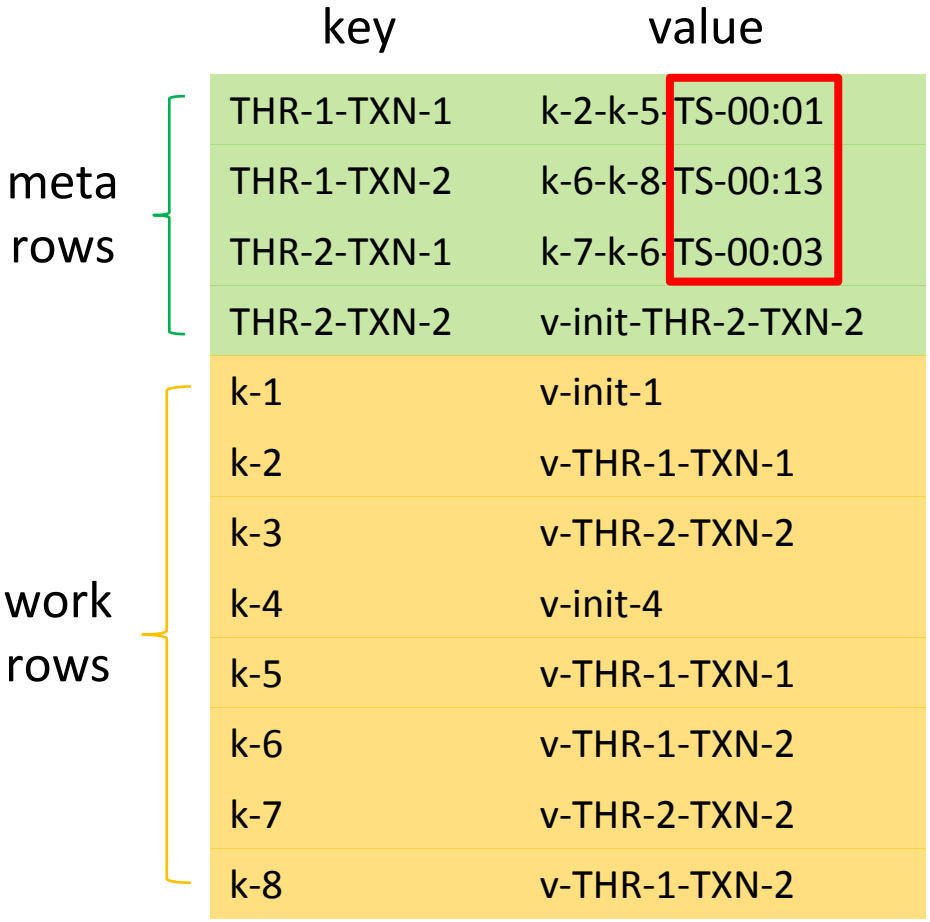
recovered DB table

Is there any ACID violation after recovery?



Atomicity violation!
should have been
updated w/ work rows

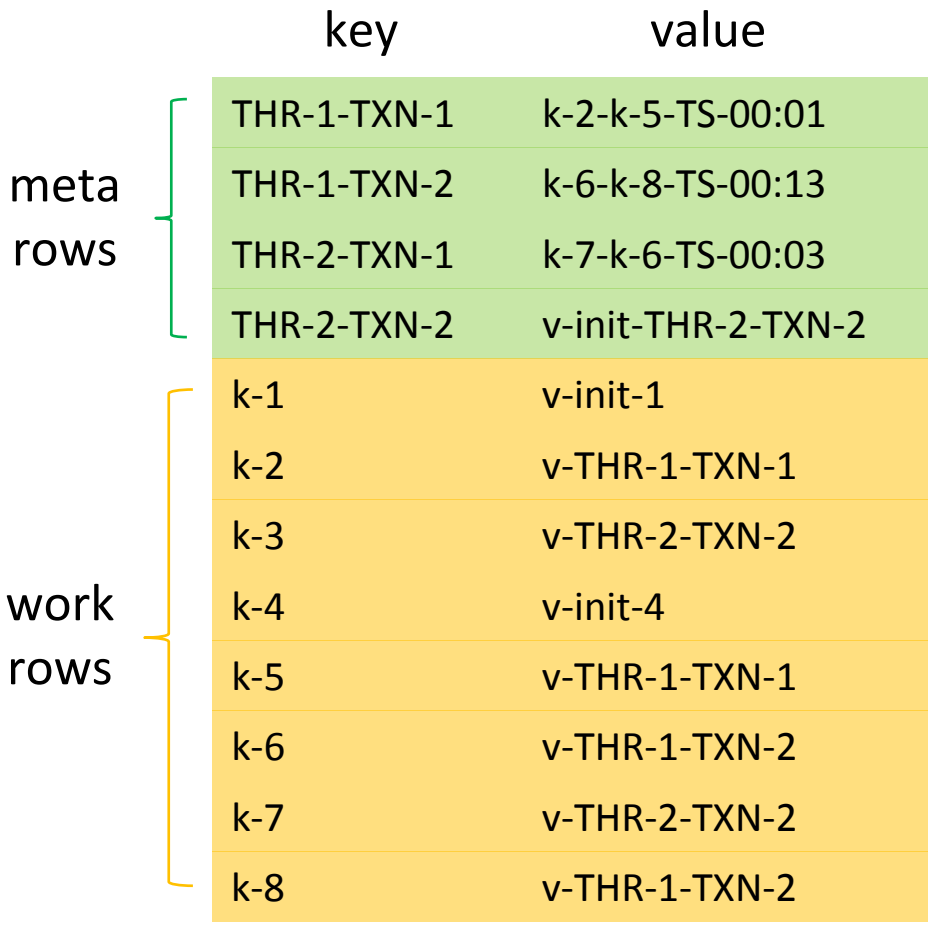
Is there any ACID violation after recovery?



recovered DB table

allow checking time & order related properties

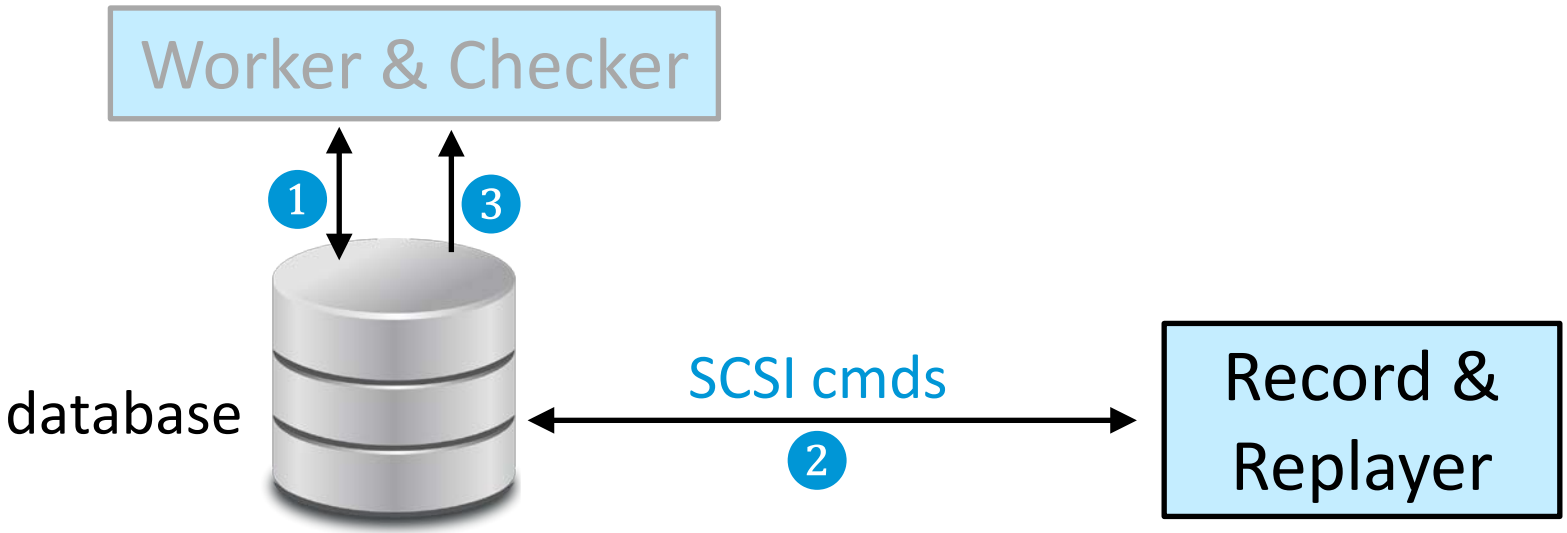
Is there any ACID violation after recovery?



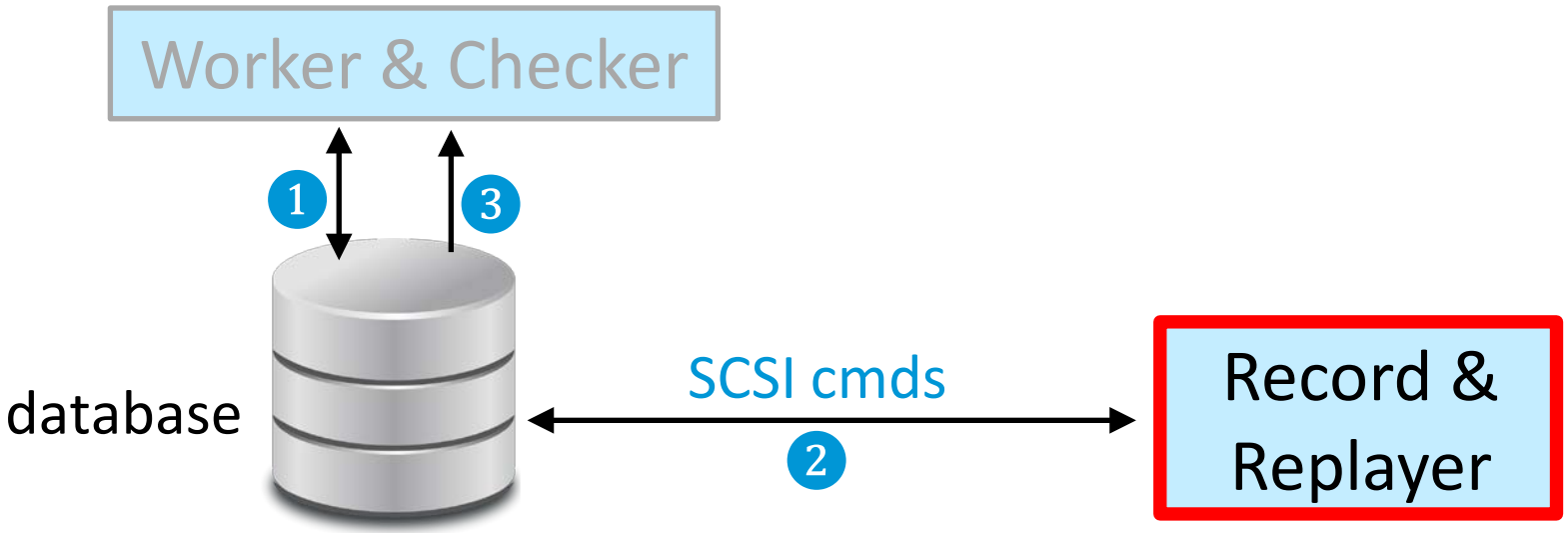
recovered DB table

More workloads & ACID checking in the paper

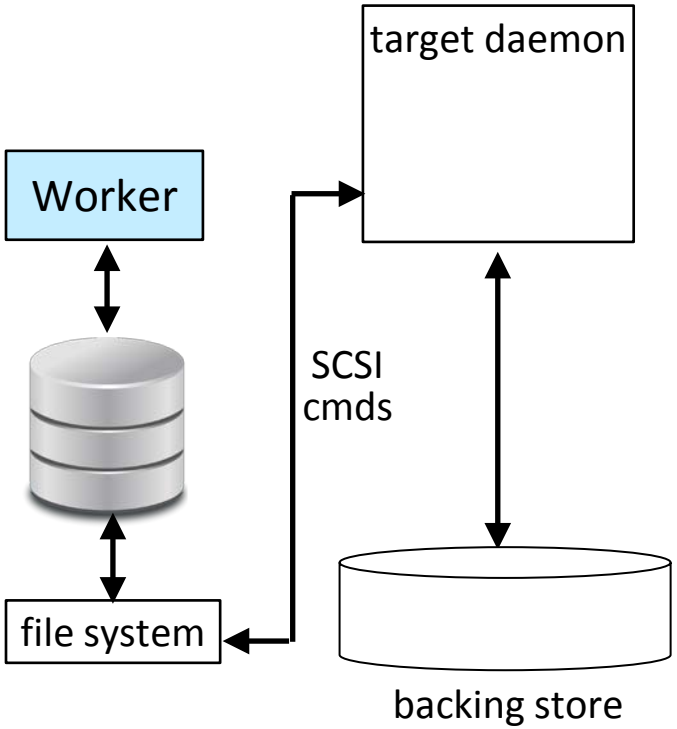
Framework Overview



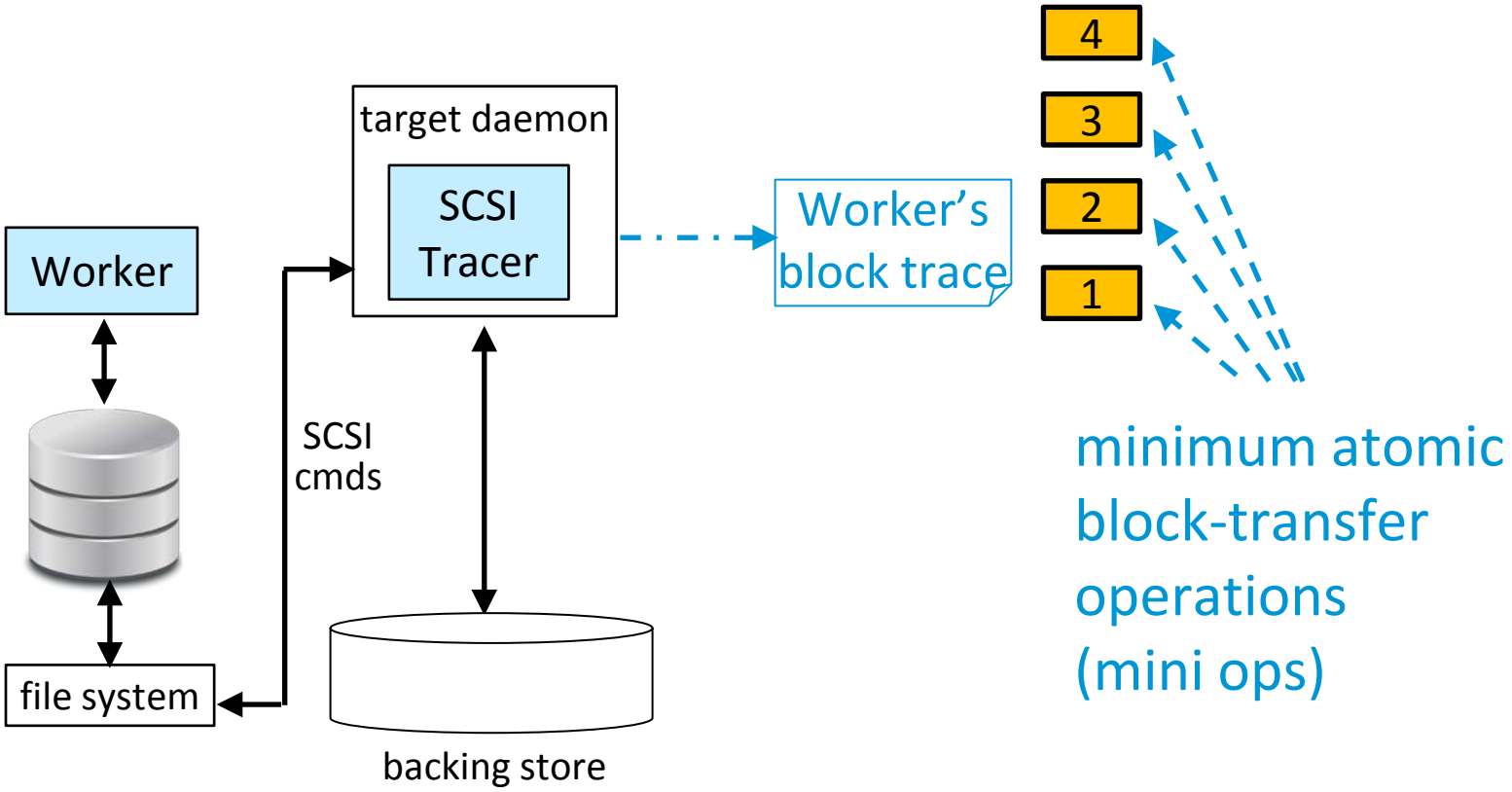
Framework Overview



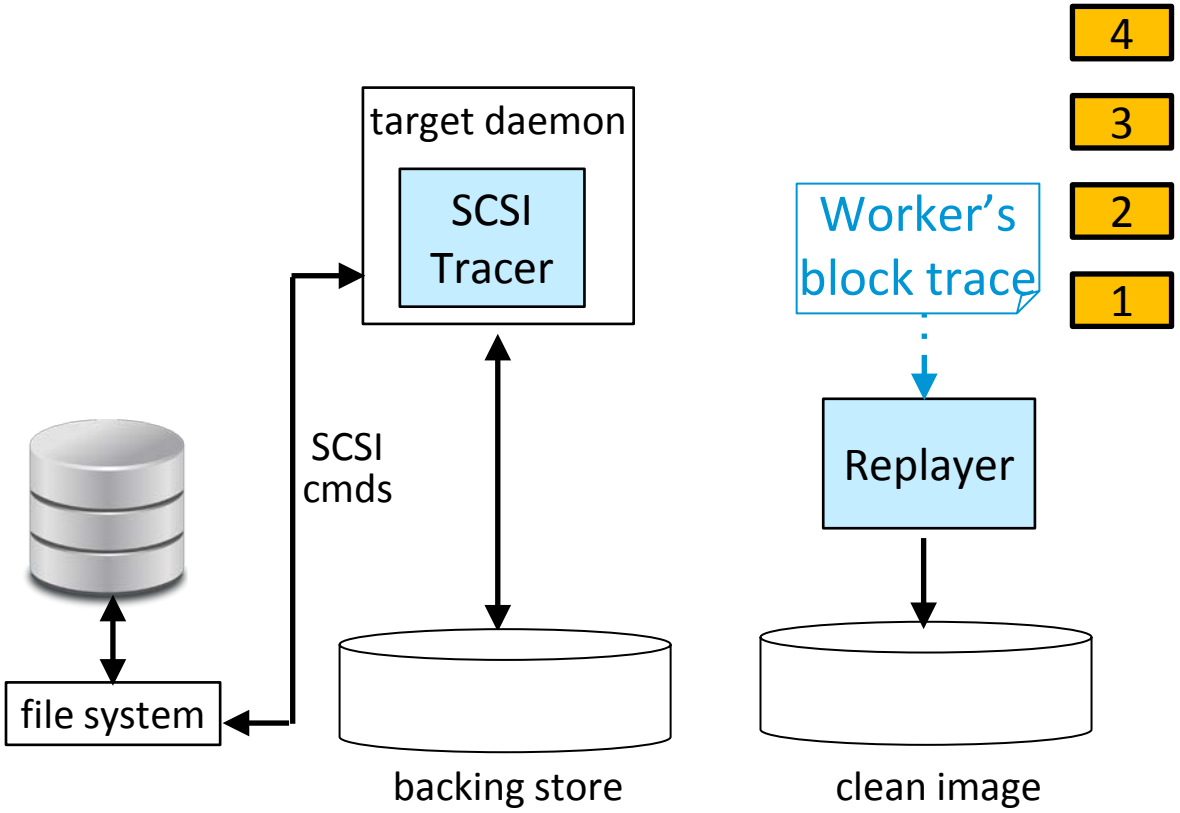
Capturing I/O trace without kernel modification



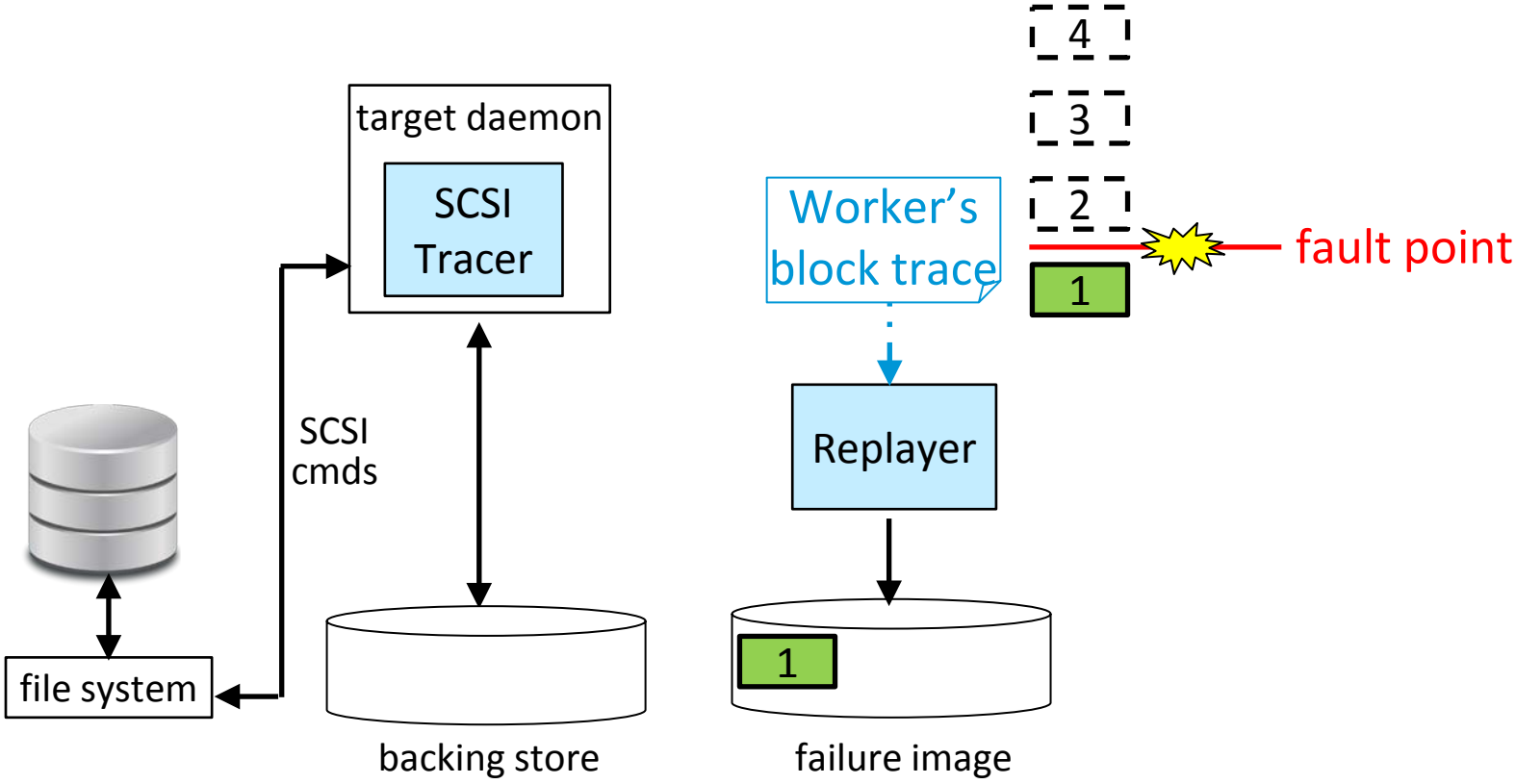
Capturing I/O trace without kernel modification



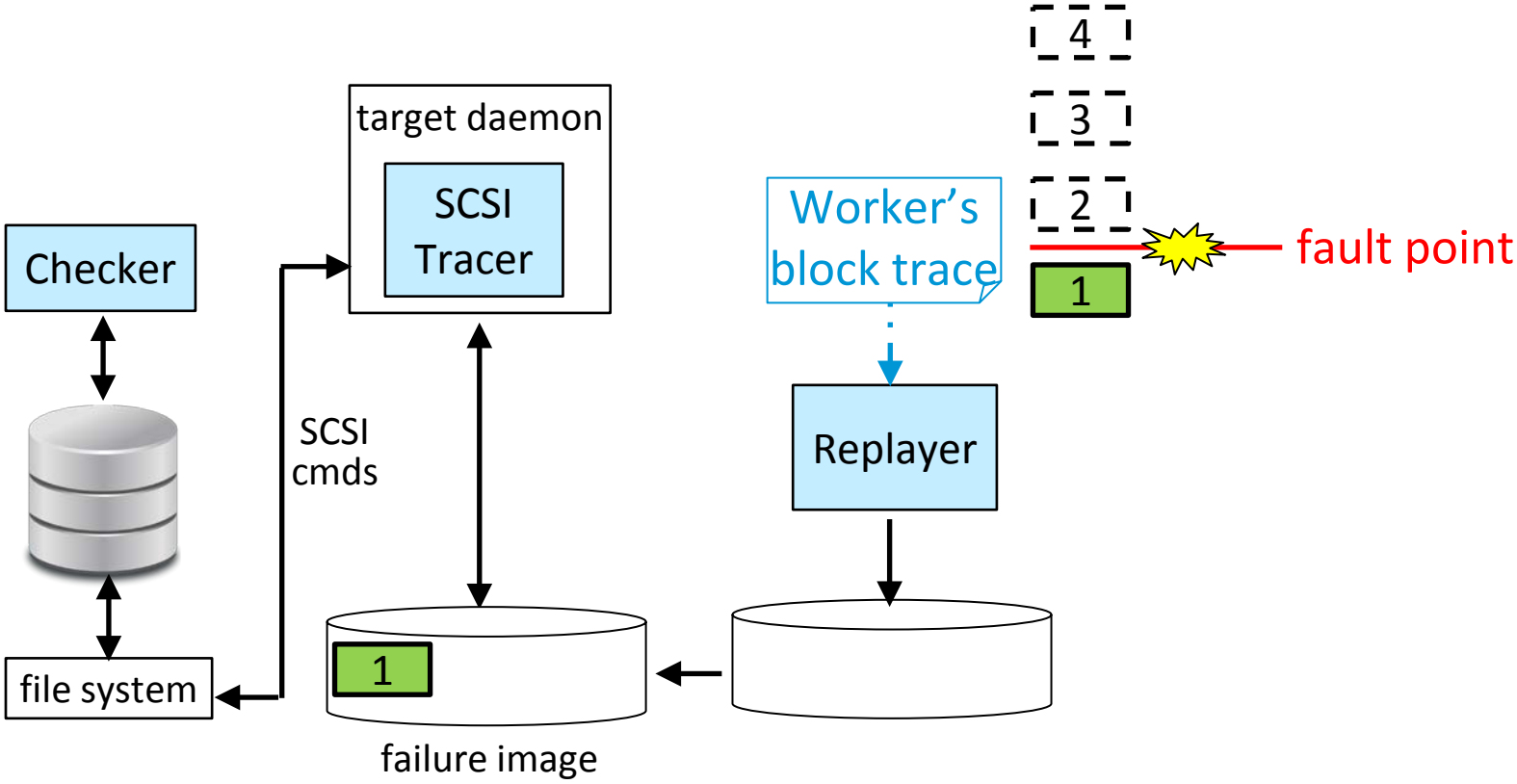
Constructing a post-fault disk image



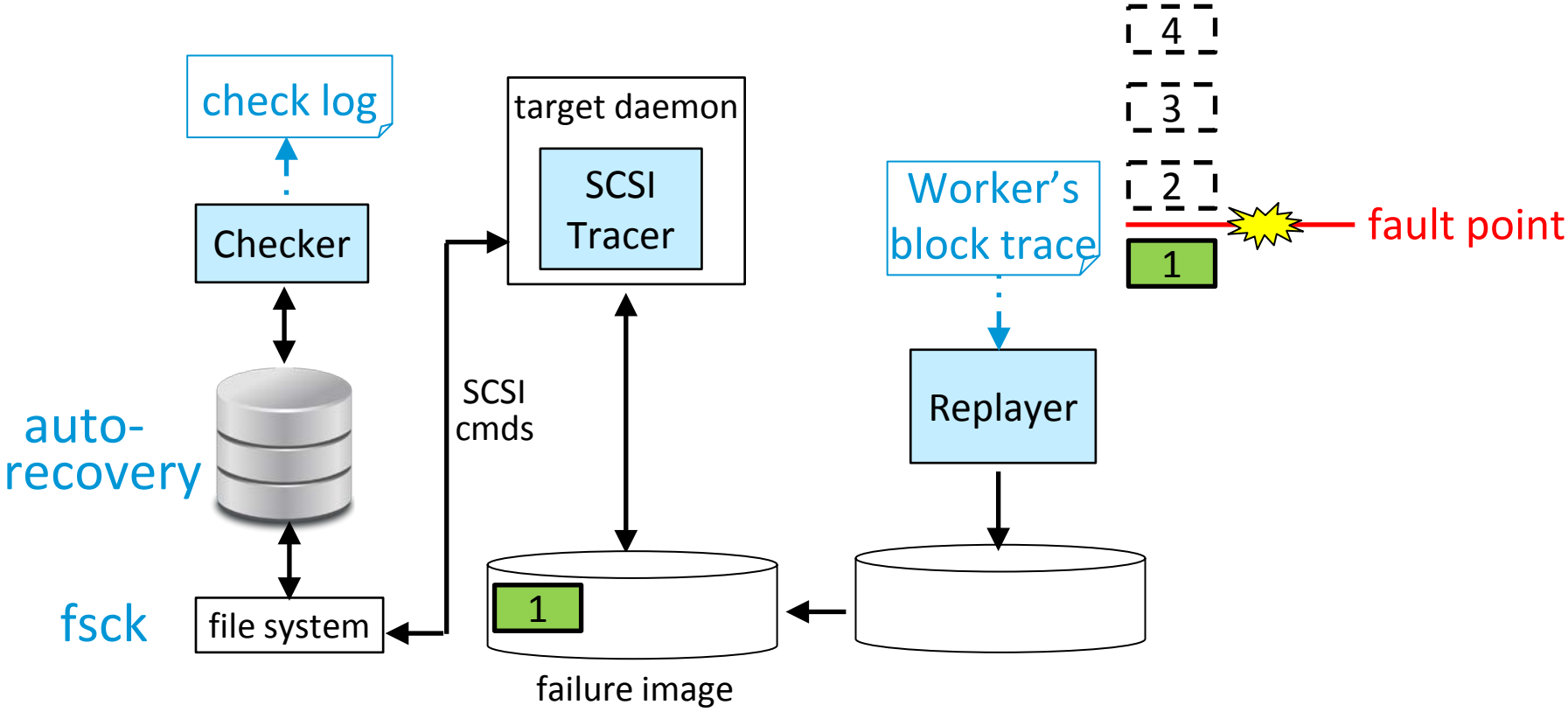
Constructing a post-fault disk image



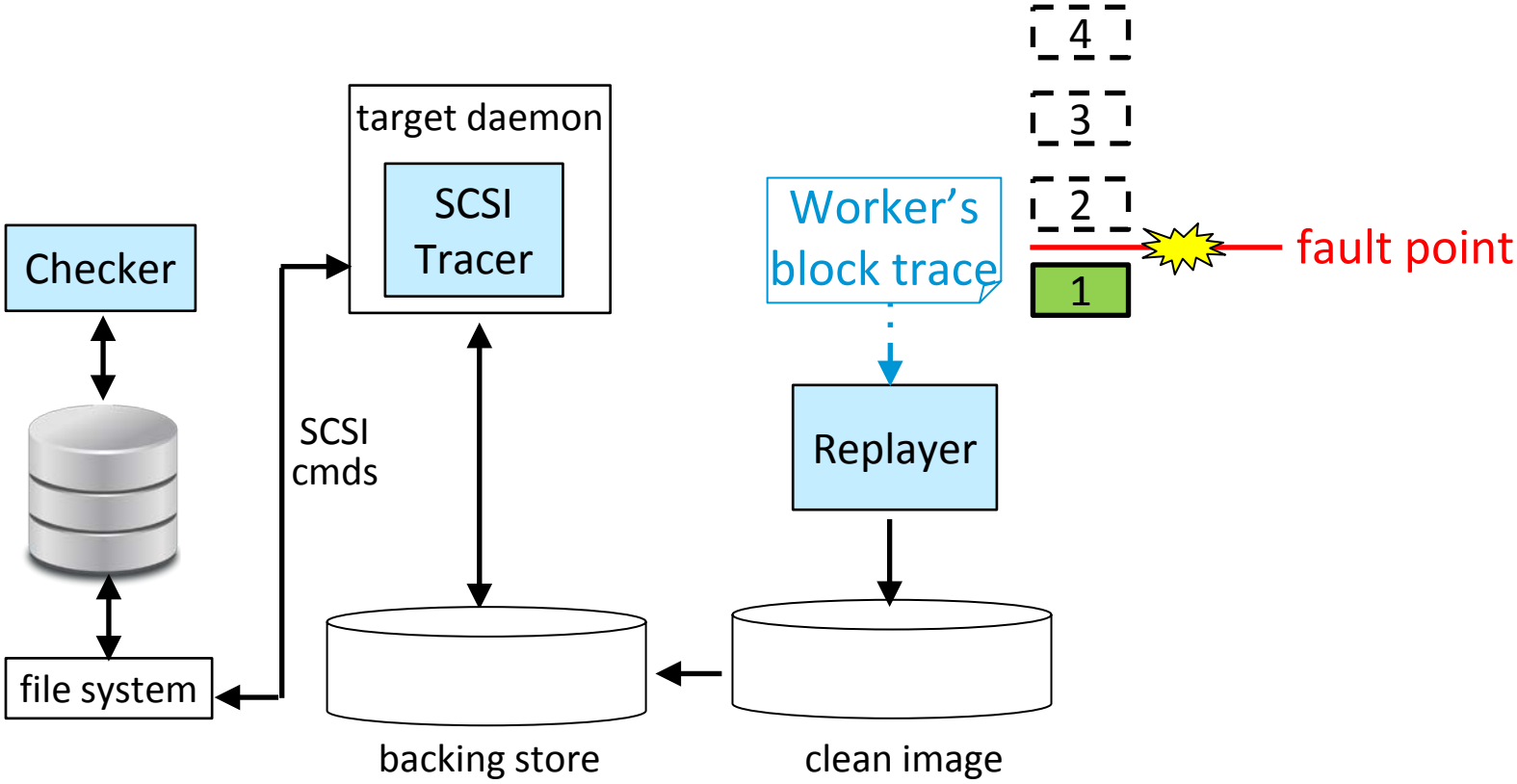
Checking the post-fault DB



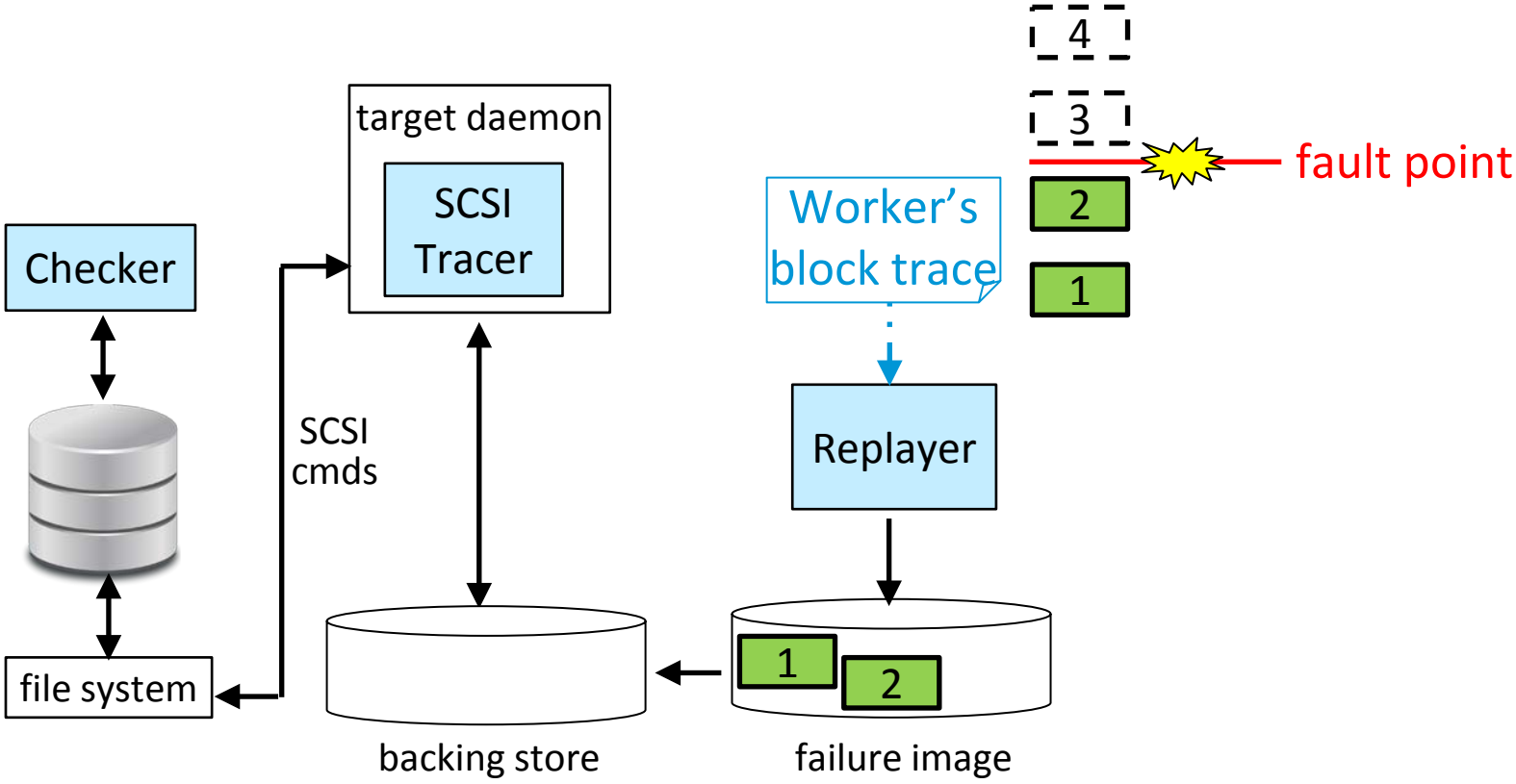
Checking the post-fault DB



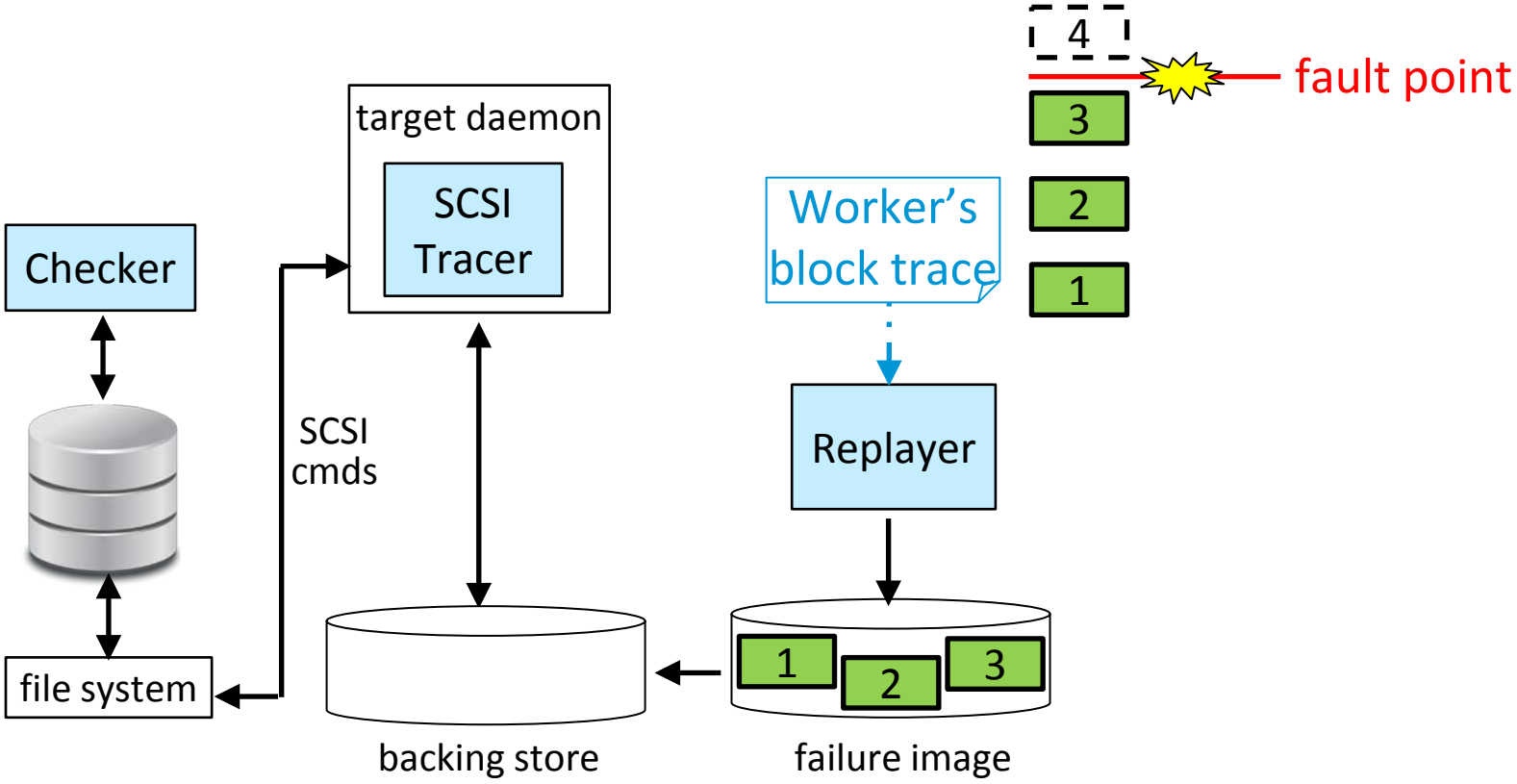
Testing different fault points easily



Testing different fault points easily

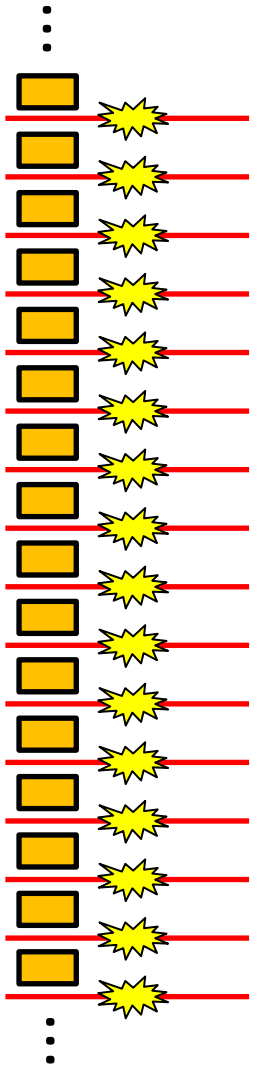


Testing different fault points easily



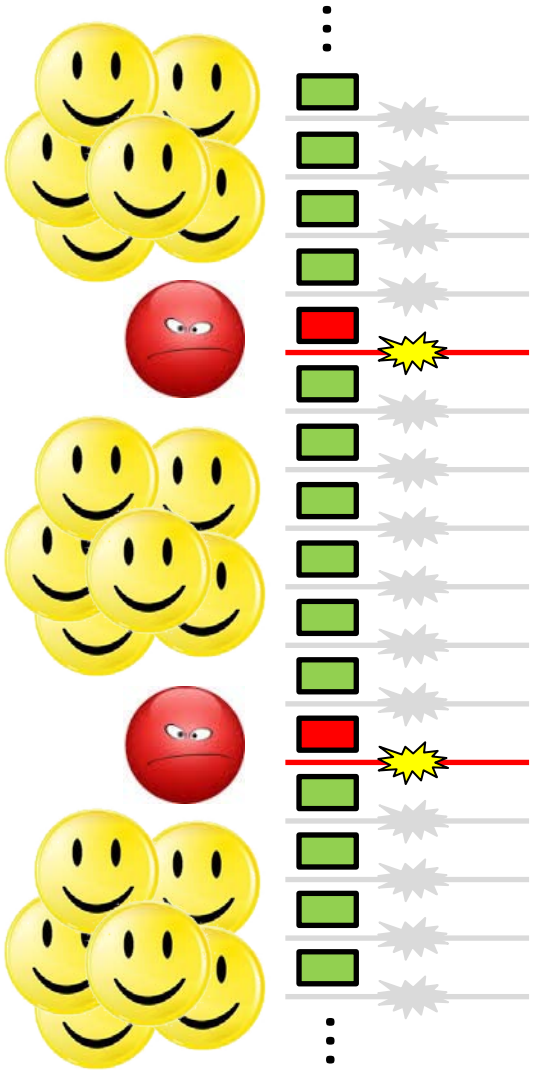
The framework is not good enough

- Sometimes need several days
  - too many mini operations, too many potential fault points



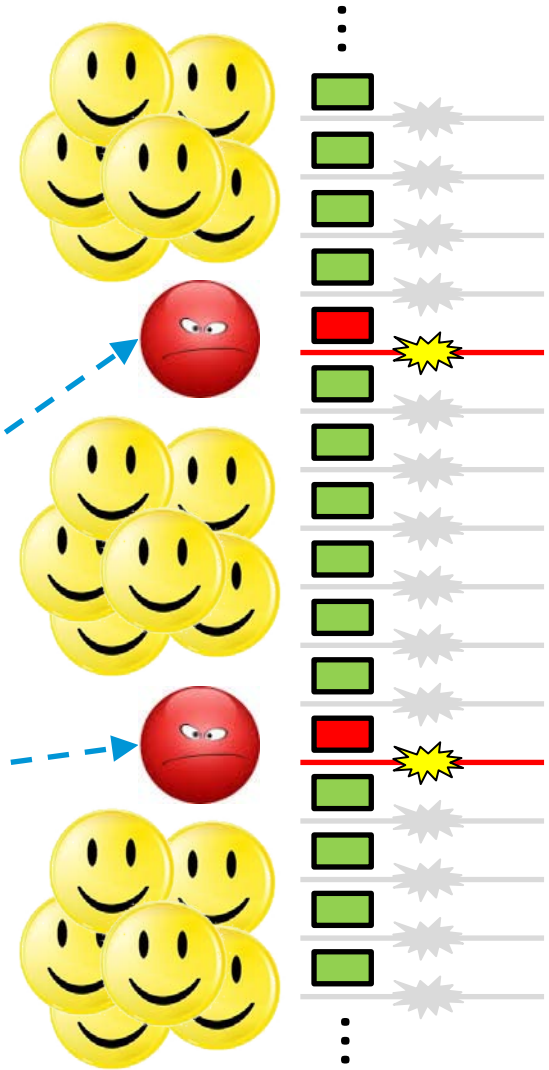
The framework is not good enough

- Sometimes need several days
  - too many mini operations, too many potential fault points
- We tried sampling
  - but only a few fault points trigger ACID violations



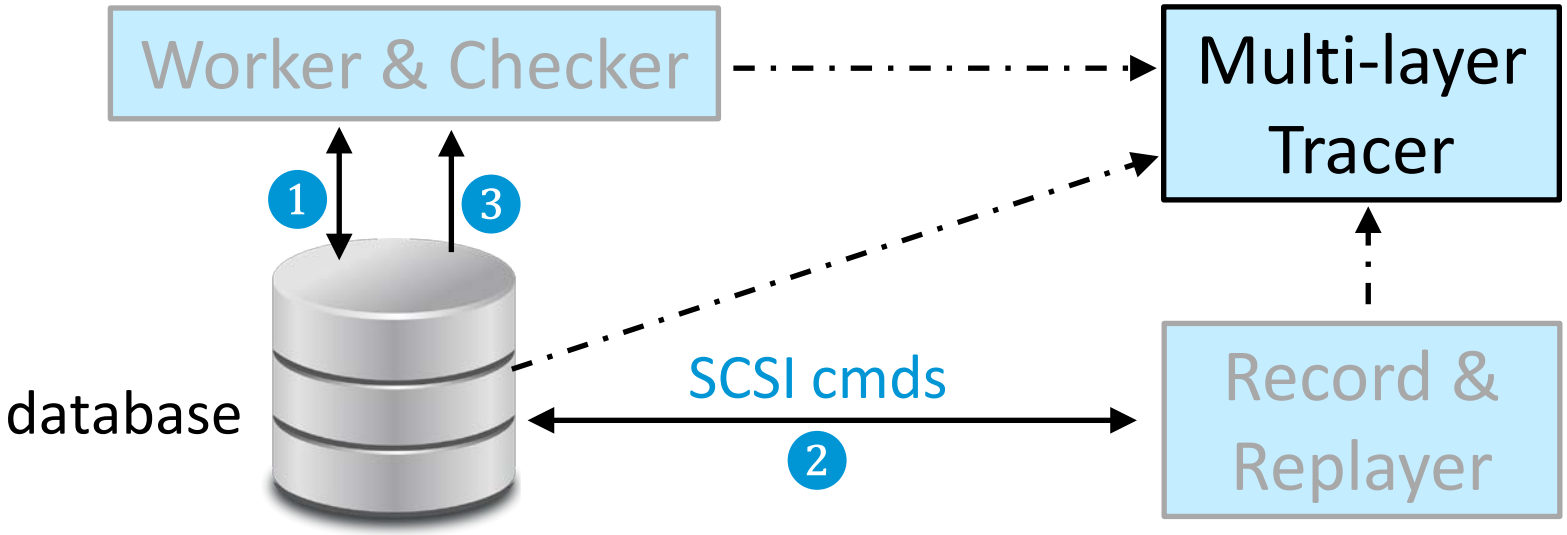
The framework is not good enough

- Sometimes need several days
  - too many mini operations, too many potential fault points
- We tried sampling
  - but only a few fault points trigger ACID violations
- Don’t know why

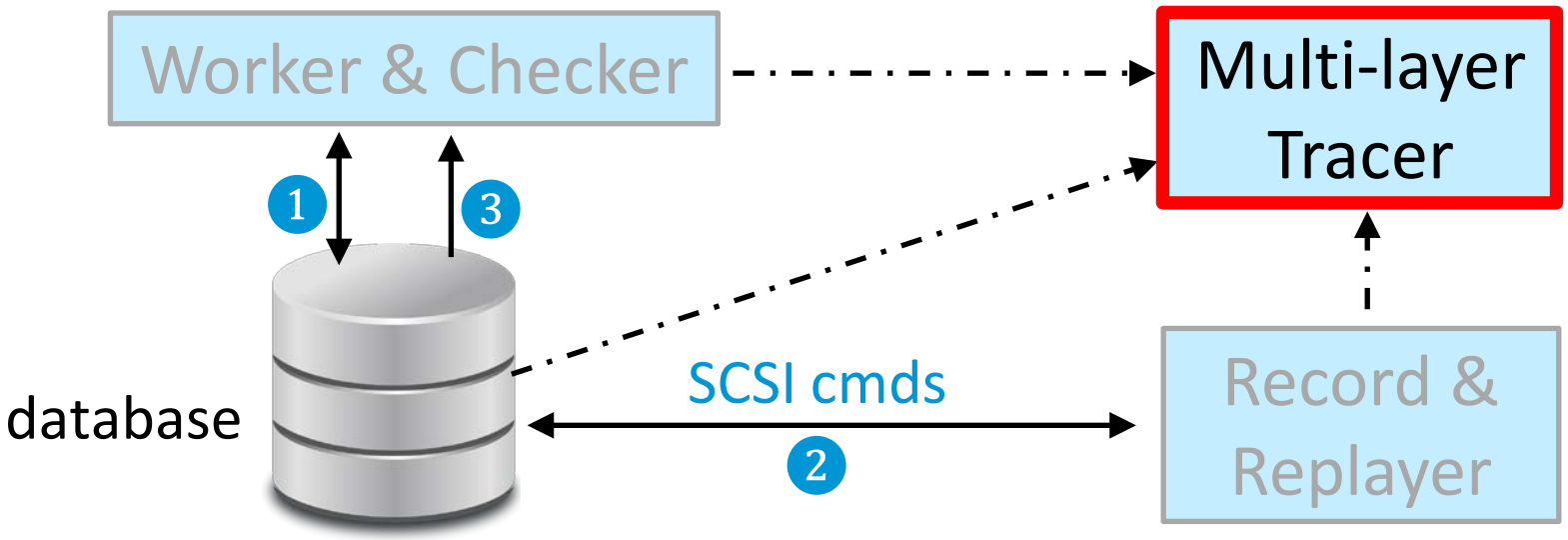


Enhanced Design

Framework Overview

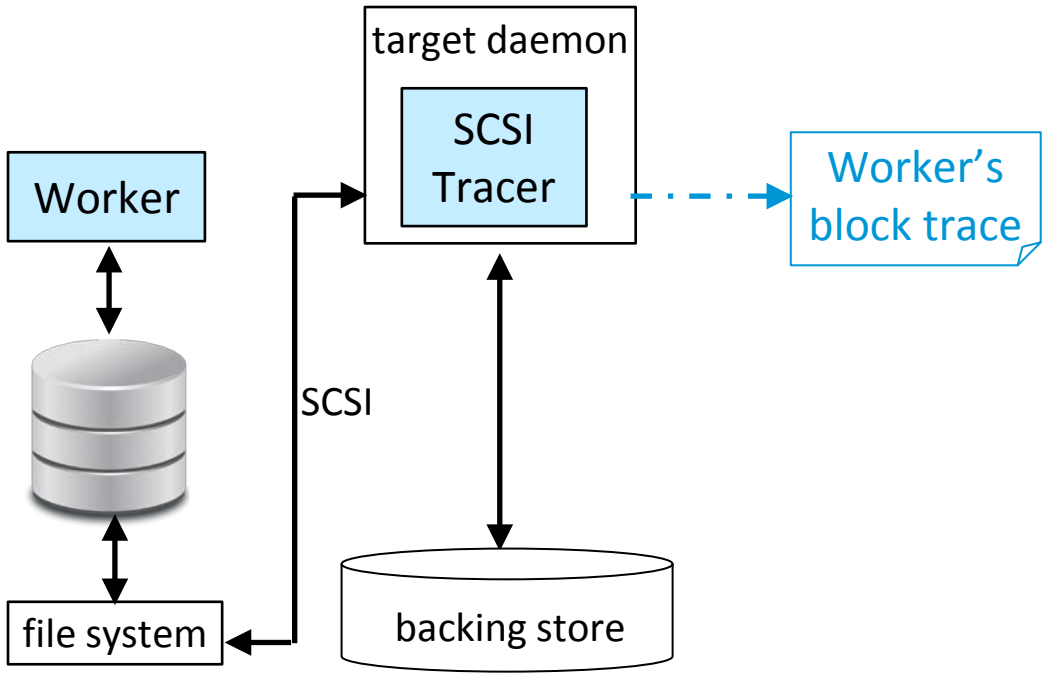


Framework Overview



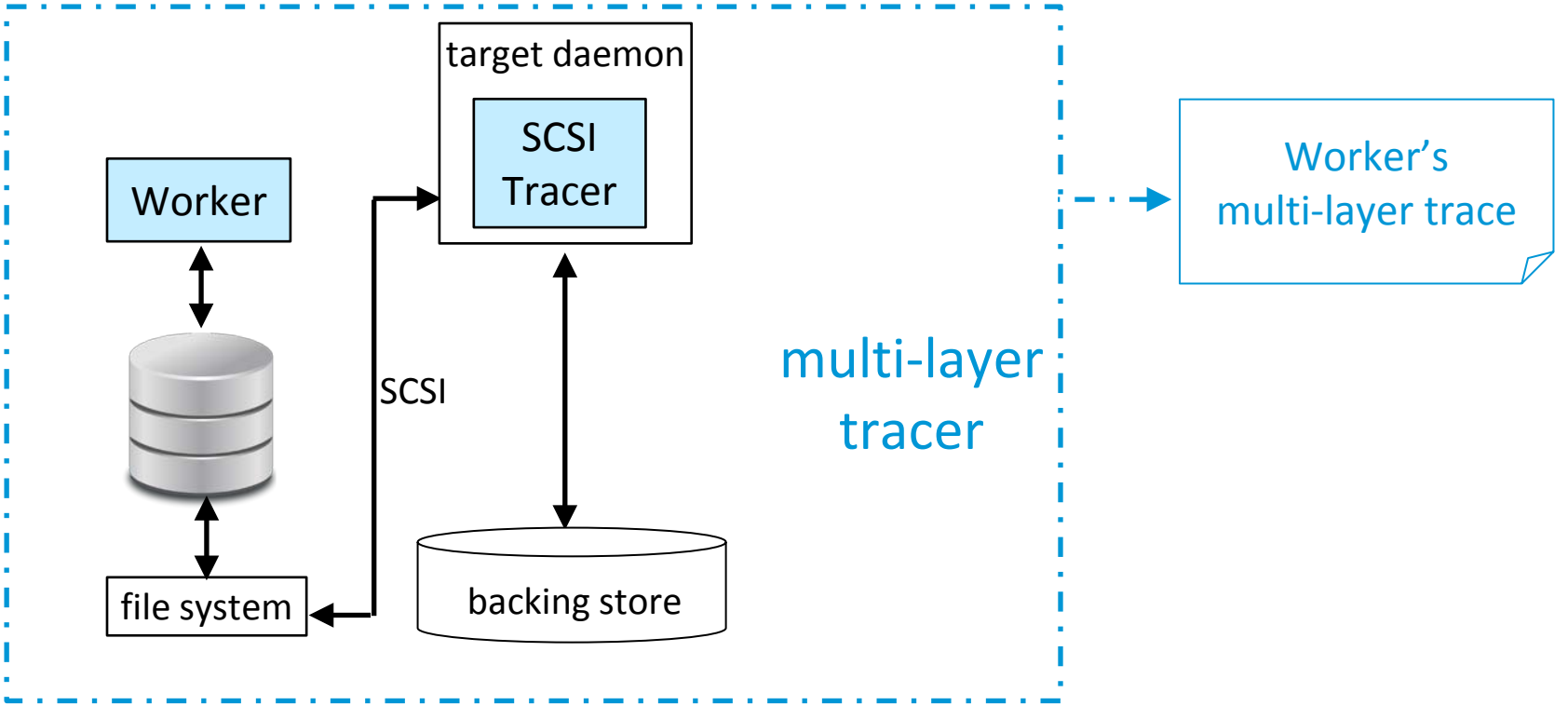
Original trace provides little semantics

op#	content	LBA
1	0a080101 ...	1012
2	0a080001 ...	6541
3	98393bc0 ...	9598
4	00000100 ...	9602



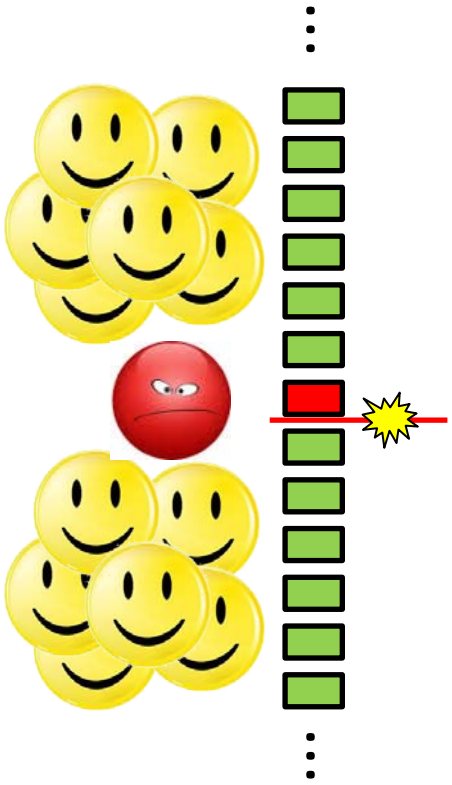
Enhancing w/ more context

op#	content	LBA	timestamp	SCSI cmd#	file	syscall
1	0a080101 ...	1012	139...013065	1	x.db	msync(x.db)
2	0a080001 ...	6541	139...210438	2	x.log	fsync(x.log)
3	98393bc0 ...	9598	139...355253	3	fs-j	fsync(x.log)
4	00000100 ...	9602	139...506097	3	fs-j	fsync(x.log)



What makes some fault points special?

Checking result



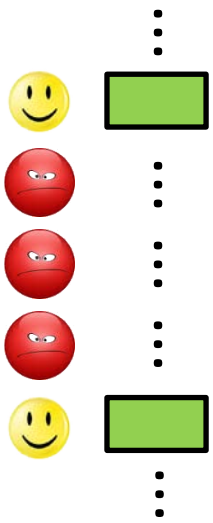
Worker's multi-layer trace

op#	content	LBA	ts	cmd#	file	syscall
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	**…**	**…**	**…**	**…**	**…**	**…**
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…

anything special?

5 patterns found from 2 databases

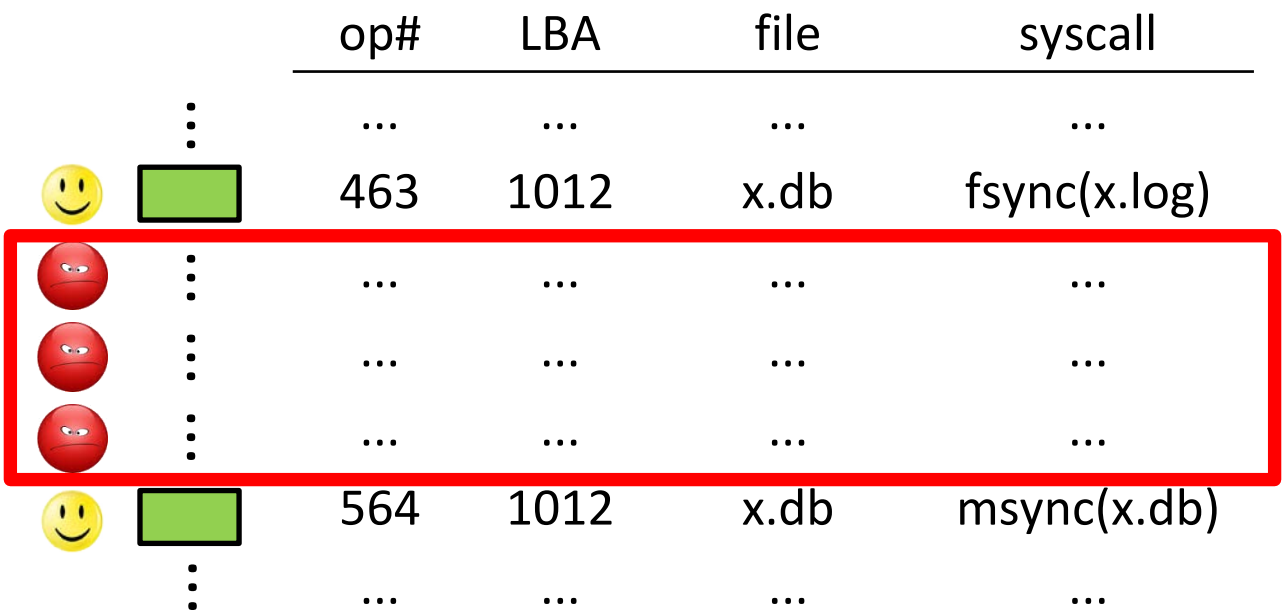
- MMAP_p : unintended update to mmap'ed blocks



op#	LBA	file	syscall
…	…	…	…
463	1012	x.db	fsync(x.log)
…	…	…	…
…	…	…	…
…	…	…	…
564	1012	x.db	msync(x.db)
…	…	…	…

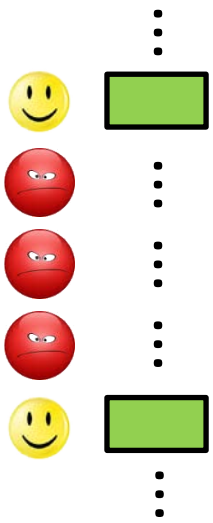
5 patterns found from 2 databases

- MMAP_p : unintended update to mmap'ed blocks



5 patterns found from 2 databases

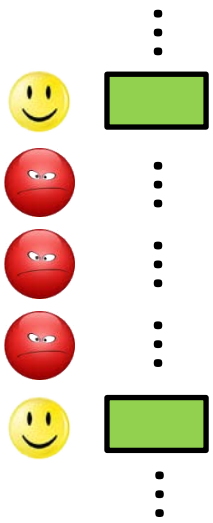
- $MMAP_p$: unintended update to mmap'ed blocks



op#	LBA	file	syscall	
…	…	…	…	
463	1012	x.db	fsync(x.log)	
…	…	…	…	
…	…	…	…	
…	…	…	…	
564	1012	x.db	msync(x.db)	intended
…	…	…	…	

5 patterns found from 2 databases

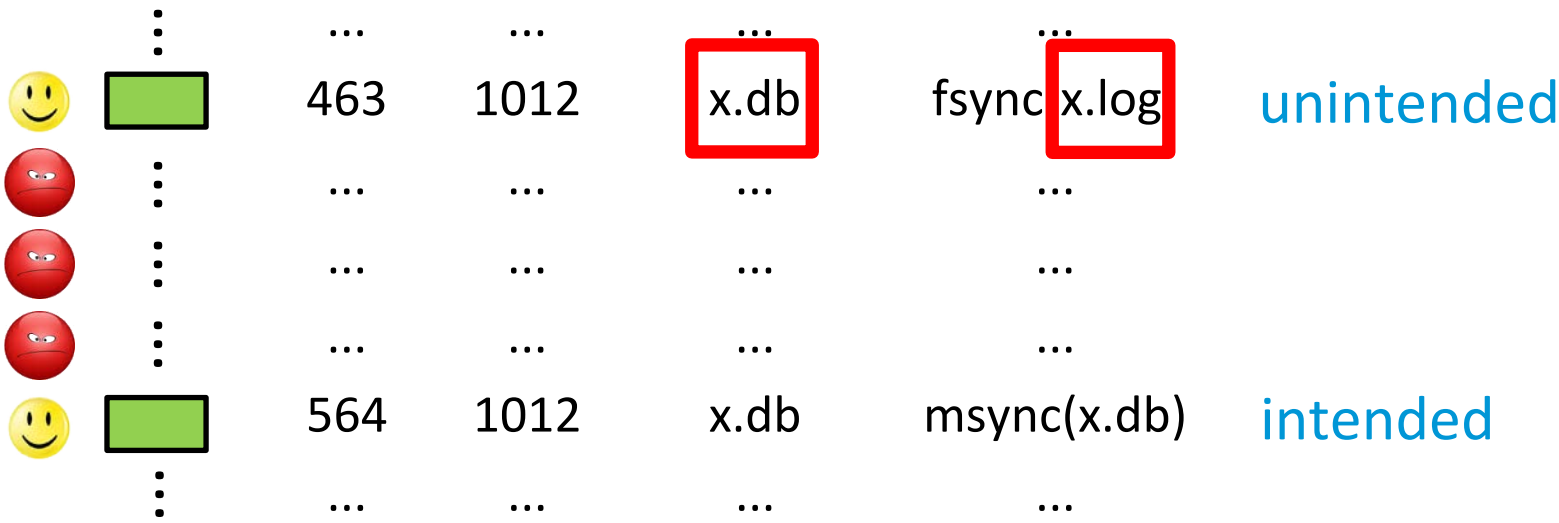
- $MMAP_p$: unintended update to mmap'ed blocks



op#	LBA	file	syscall	
…	…	…	…	
463	1012	x.db	fsync(x.log)	
…	…	…	…	
…	…	…	…	
…	…	…	…	
564	1012	x.db	msync(x.db)	intended
…	…	…	…	

5 patterns found from 2 databases

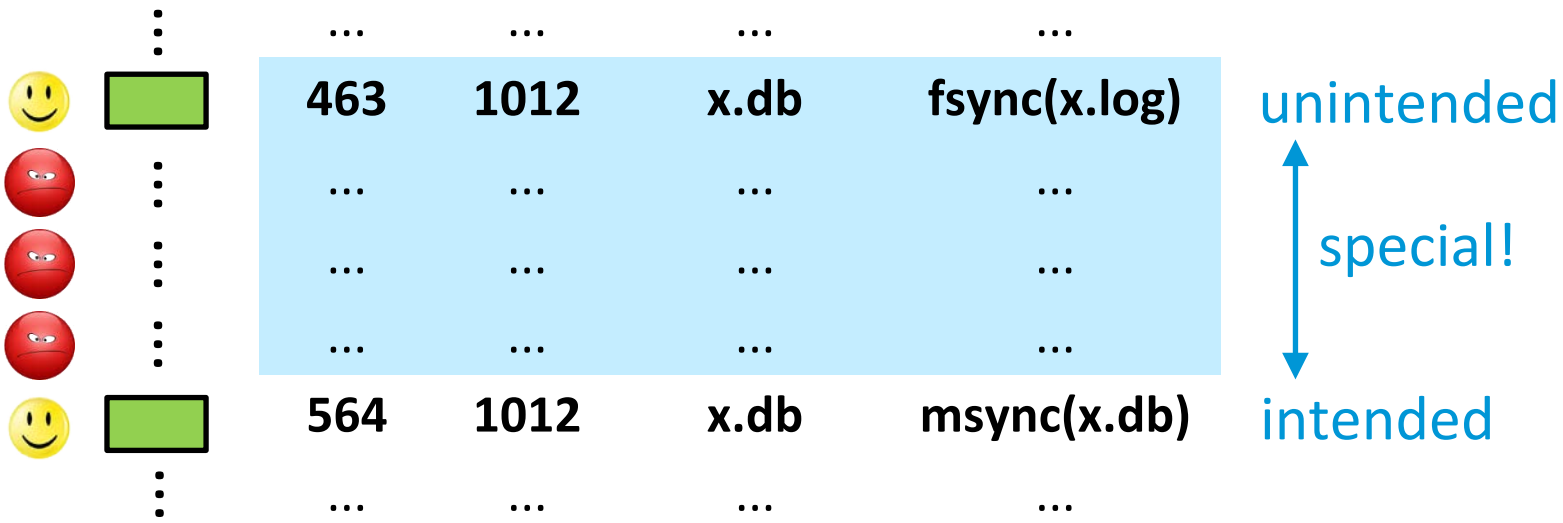
- $MMAP_p$: unintended update to mmap'ed blocks



implicit flush of dirty blocks by kernel or FS under heavy transactions

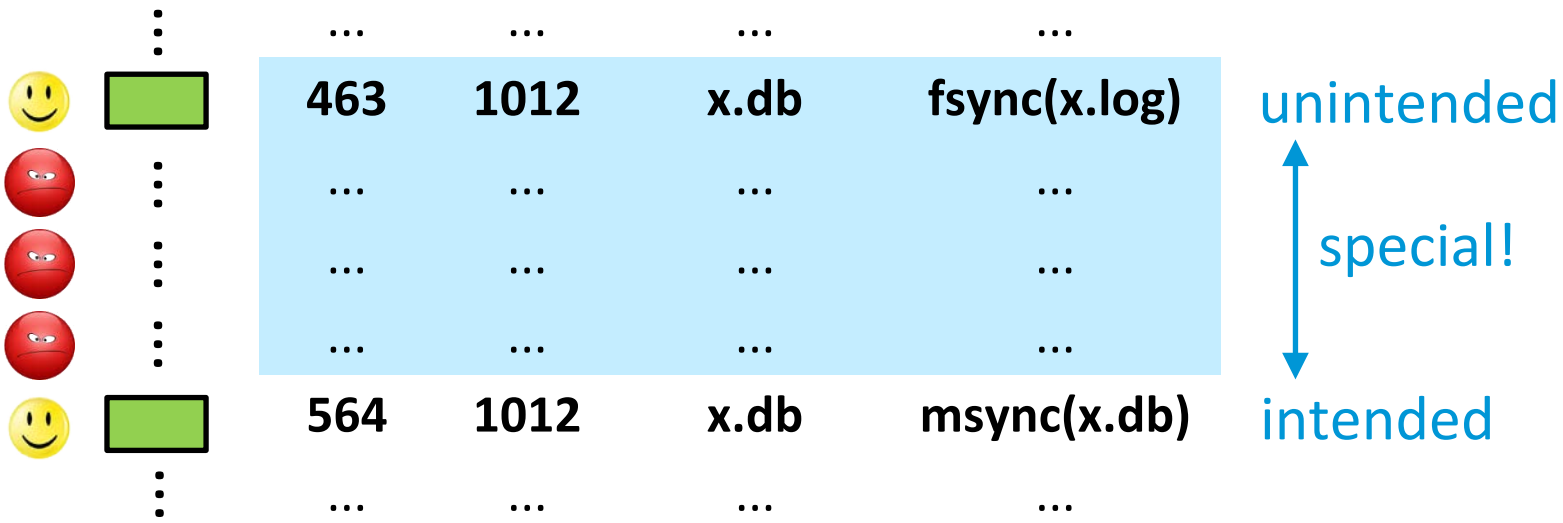
5 patterns found from 2 databases

- $MMAP_p$: unintended update to mmap'ed blocks



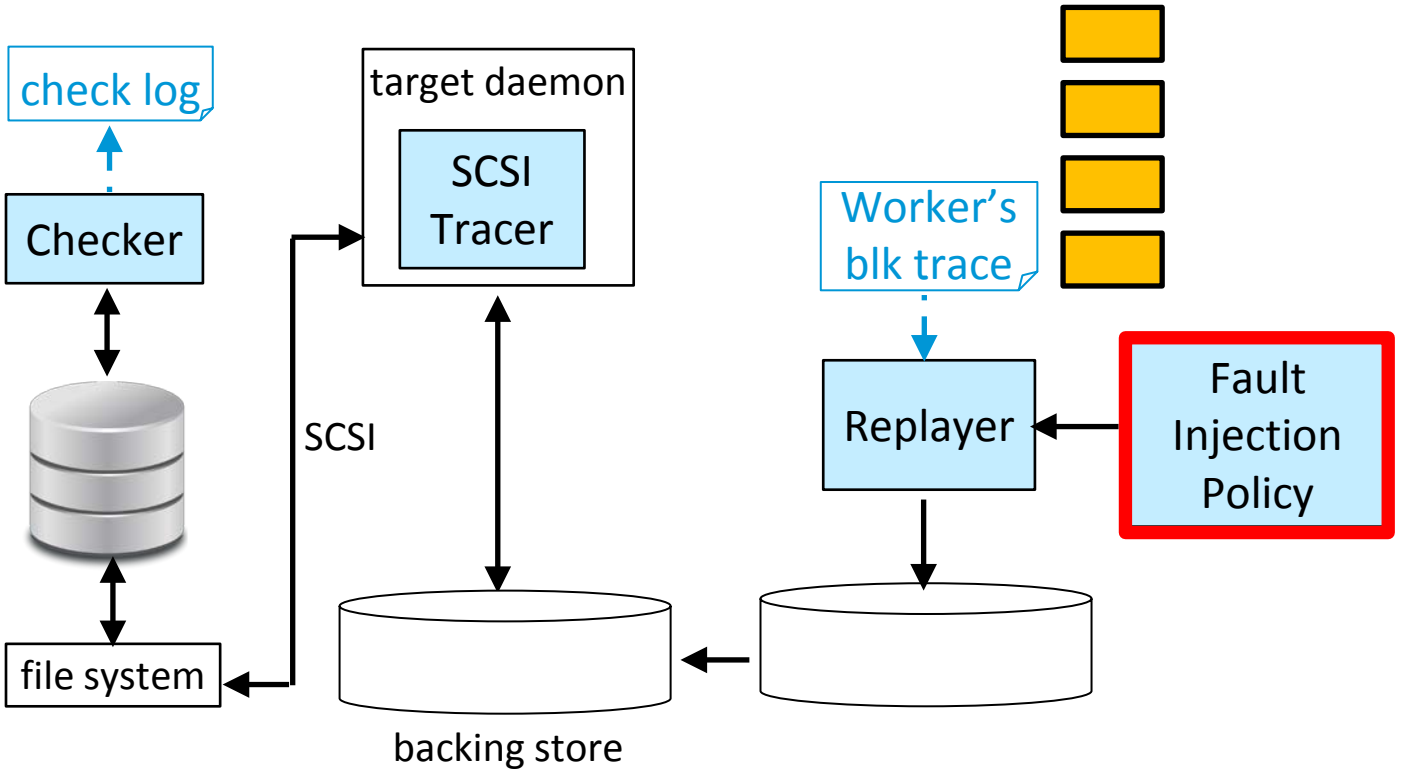
5 patterns found from 2 databases

- $MMAP_p$: unintended update to mmap'ed blocks



- Four more patterns: REP_p , $JUMP_p$, $HEAD_p$, $TRAN_p$

Add fault injection policy to determine where to inject faults



Alternative to Exhaustive: Pattern-based Ranking

Worker's multi-layer trace

op#	LBA	cmd#	file	syscall	
1	348	1	x.db	msyc(x.db)	1
2	352	2	x.log	fsync(x.log)	2
3	356	2	x.log	fsync(x.log)	3
4	360	2	x.log	fsync(x.log)	4
5	364	2	x.log	fsync(x.log)	5
6	370	3	x.log	fsync(x.log)	6
7	348	4	x.db	fsync(x.log)	7
8	906	5	fs-j	fsync(x.log)	8

Scoreboard

$MMAP_p$	REP_p	$JUMP_p$	$HEAD_p$	$TRAN_p$	total score

Alternative to Exhaustive: Pattern-based Ranking

Worker's multi-layer trace

op#	LBA	cmd#	file	syscall
1	348	1	x.db	msyc(x.db)
2	352	2	x.log	fsync(x.log)
3	356	2	x.log	fsync(x.log)
4	360	2	x.log	fsync(x.log)
5	364	2	x.log	fsync(x.log)
6	370	3	x.log	fsync(x.log)
7	348	4	x.db	fsync(x.log)
8	906	5	fs-j	fsync(x.log)

Scoreboard

	$MMAP_p$	REP_p	$JUMP_p$	$HEAD_p$	$TRAN_p$	total score
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	1					
8	1					

Alternative to Exhaustive: Pattern-based Ranking

Worker's multi-layer trace

op#	LBA	cmd#	file	syscall
1	348	1	x.db	msyc(x.db)
2	352	2	x.log	fsync(x.log)
3	356	2	x.log	fsync(x.log)
4	360	2	x.log	fsync(x.log)
5	364	2	x.log	fsync(x.log)
6	370	3	x.log	fsync(x.log)
7	348	4	x.db	fsync(x.log)
8	906	5	fs-j	fsync(x.log)

Scoreboard

	$MMAP_p$	REP_p	$JUMP_p$	$HEAD_p$	$TRAN_p$	total score
1	0	1				
2	0	0				
3	0	0				
4	0	0				
5	0	0				
6	0	0				
7	1	1				
8	1	0				

Alternative to Exhaustive: Pattern-based Ranking

Worker's multi-layer trace

op#	LBA	cmd#	file	syscall
1	348	1	x.db	msyc(x.db)
2	352	2	x.log	fsync(x.log)
3	356	2	x.log	fsync(x.log)
4	360	2	x.log	fsync(x.log)
5	364	2	x.log	fsync(x.log)
6	370	3	x.log	fsync(x.log)
7	348	4	x.db	fsync(x.log)
8	906	5	fs-j	fsync(x.log)

Scoreboard

	$MMAP_p$	REP_p	$JUMP_p$	$HEAD_p$	$TRAN_p$	total score
1	0	1	0			
2	0	0	0			
3	0	0	0			
4	0	0	0			
5	0	0	0			
6	0	0	1			
7	1	1	1			
8	1	0	1			

Alternative to Exhaustive: Pattern-based Ranking

Worker's multi-layer trace

op#	LBA	cmd#	file	syscall
1	348	1	x.db	msyc(x.db)
2	352	2	x.log	fsync(x.log)
3	356	2	x.log	fsync(x.log)
4	360	2	x.log	fsync(x.log)
5	364	2	x.log	fsync(x.log)
6	370	3	x.log	fsync(x.log)
7	348	4	x.db	fsync(x.log)
8	906	5	fs-j	fsync(x.log)

Scoreboard

	$MMAP_p$	REP_p	$JUMP_p$	$HEAD_p$	$TRAN_p$	total score
1	0	1	0	0		
2	0	0	0	1		
3	0	0	0	0		
4	0	0	0	0		
5	0	0	0	0		
6	0	0	1	0		
7	1	1	1	0		
8	1	0	1	0		

Alternative to Exhaustive: Pattern-based Ranking

Worker's multi-layer trace

op#	LBA	cmd#	file	syscall
1	348	1	x.db	msyc(x.db)
2	352	2	x.log	fsync(x.log)
3	356	2	x.log	fsync(x.log)
4	360	2	x.log	fsync(x.log)
5	364	2	x.log	fsync(x.log)
6	370	3	x.log	fsync(x.log)
7	348	4	x.db	fsync(x.log)
8	906	5	fs-j	fsync(x.log)

Scoreboard

	$MMAP_p$	REP_p	$JUMP_p$	$HEAD_p$	$TRAN_p$	total score
1	0	1	0	0	1	
2	0	0	0	1	1	
3	0	0	0	0	0	
4	0	0	0	0	0	
5	0	0	0	0	0	
6	0	0	1	0	1	
7	1	1	1	0	1	
8	1	0	1	0	1	

Alternative to Exhaustive: Pattern-based Ranking

Worker's multi-layer trace

op#	LBA	cmd#	file	syscall
1	348	1	x.db	msyc(x.db)
2	352	2	x.log	fsync(x.log)
3	356	2	x.log	fsync(x.log)
4	360	2	x.log	fsync(x.log)
5	364	2	x.log	fsync(x.log)
6	370	3	x.log	fsync(x.log)
7	348	4	x.db	fsync(x.log)
8	906	5	fs-j	fsync(x.log)

Scoreboard

	$MMAP_p$	REP_p	$JUMP_p$	$HEAD_p$	$TRAN_p$	total score
1	0	1	0	0	1	**2**
2	0	0	0	1	1	**2**
3	0	0	0	0	0	**0**
4	0	0	0	0	0	**0**
5	0	0	0	0	0	**0**
6	0	0	1	0	1	**2**
7	1	1	1	0	1	**4**
8	1	0	1	0	1	**3**

Alternative to Exhaustive: Pattern-based Ranking

Worker's multi-layer trace

op#	LBA	cmd#	file	syscall	
1	348	1	x.db	msyc(x.db)	1
2	352	2	x.log	fsync(x.log)	2
3	356	2	x.log	fsync(x.log)	3
4	360	2	x.log	fsync(x.log)	4
5	364	2	x.log	fsync(x.log)	5
6	370	3	x.log	fsync(x.log)	6
7	348	4	x.db	fsync(x.log)	7
8	906	5	fs-j	fsync(x.log)	8

Scoreboard

	$MMAP_p$	REP_p	$JUMP_p$	$HEAD_p$	$TRAN_p$	total score
1	0	1	0	0	1	**2**
2	0	0	0	1	1	**2**
3	0	0	0	0	0	**0**
4	0	0	0	0	0	**0**
5	0	0	0	0	0	**0**
6	0	0	1	0	1	**2**
7	1	1	1	0	1	**4**
8	1	0	1	0	1	**3**

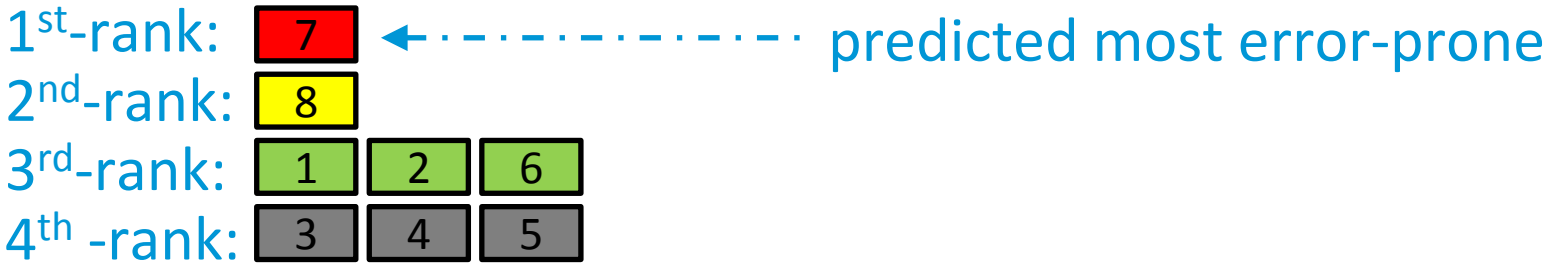
1st-rank: 7 ← predicted most error-prone

2nd-rank: 8

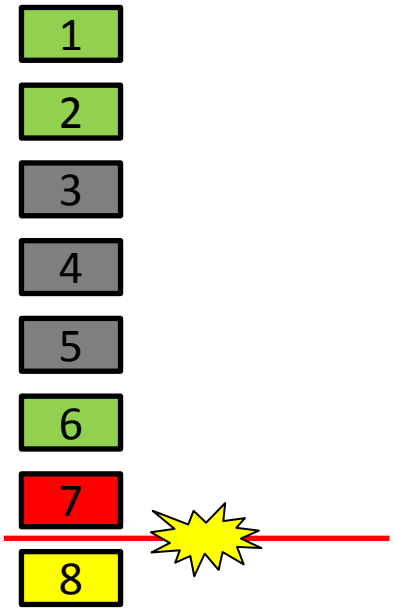
3rd-rank: 1 2 6

4th -rank: 3 4 5

Alternative to Exhaustive: Pattern-based Ranking



Alternative to Exhaustive: Pattern-based Ranking



1st-rank: 7 ← predicted most error-prone

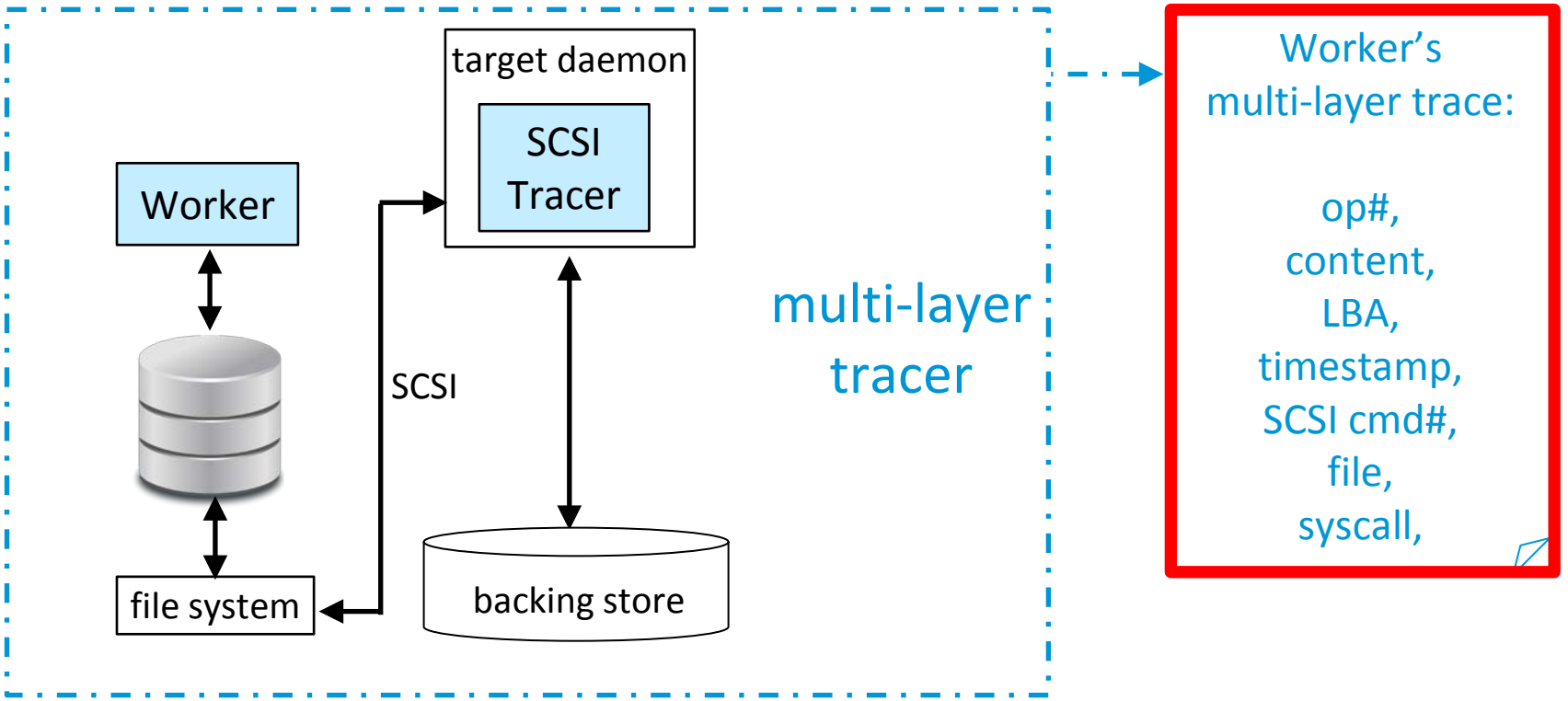
2nd-rank: 8

3rd-rank: 1 2 6

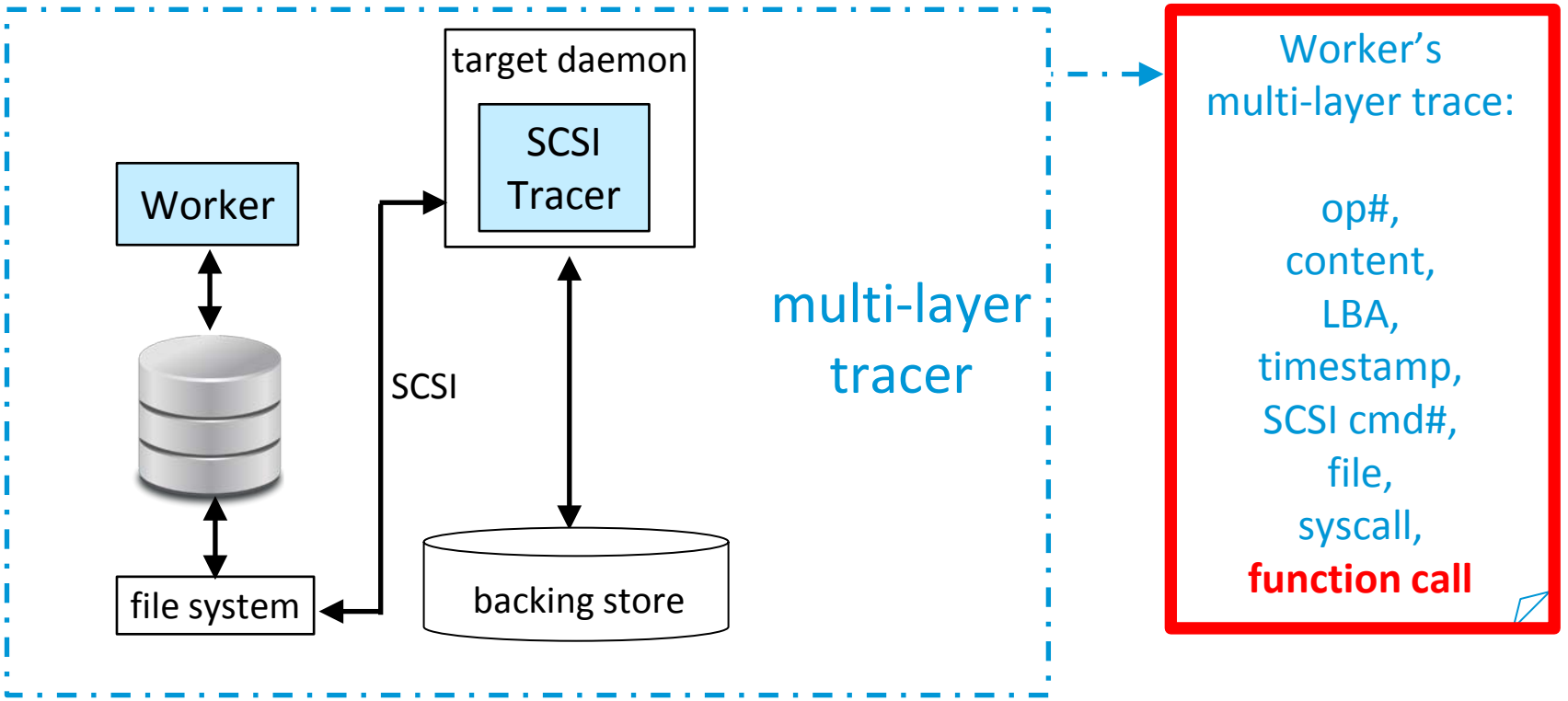
4th -rank: 3 4 5

Diagnosis Support

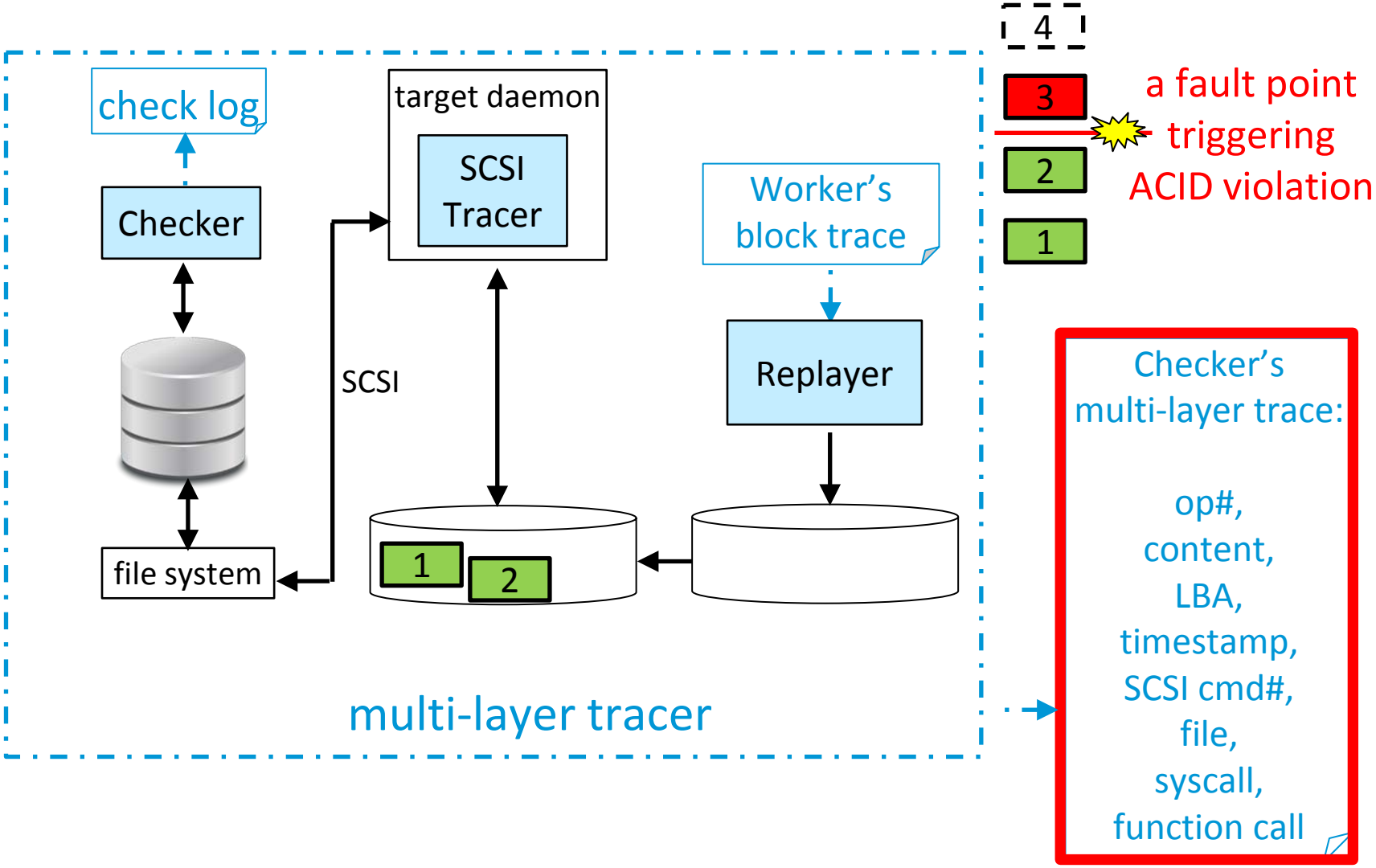
Worker's multi-layer trace helps understand what happened at fault time



Add function-call tracing to disclose more semantics for diagnosis



Enable same tracing during checking to see why recovery didn't work



Results

Experimental Environment

- 8 databases
  - Open-source: TokyoCabinet, MariaDB, LightningDB, SQLite
  - Commercial: KVS-A, SQL-A, SQL-B, SQL-C
- 4 workloads
- 3 file systems
  - ext3, XFS, NTFS
- Several operating systems
  - Linux: RHEL 6, Debian6, Ubuntu 12 LTS
  - Windows 7 Enterprise

Not a single DB can survive all tests

DB	FS	W-1	W-2	W-3	W-4.1	W-4.2	W-4.3	A	C	I	D
TokyoCabinet	ext3	D	D	D	ACD	ACD	ACD	0.15%	0.14%	0	16.05%
	XFS	--	D	D	ACD	D	ACD	<0.01%	0.01%	0	4.38%
MariaDB	ext3	D	D	D	D	D	D	0	0	0	1.36%
	XFS	D	D	D	D	D	D	0	0	0	0.49%
LightningDB	ext3	--	--	--	--	--	D	0	0	0	0.05%
	XFS	--	--	--	--	--	--	0	0	0	0
SQLite	ext3	D	D	--	D	D	D	0	0	0	19.15%
	XFS	--	--	D	D	D	D	0	0	0	10.60%
KVS-A	ext3	--	--	Hang	--	--	--	0	0	0	0
	XFS	--	--	--	--	--	--	0	0	0	0
SQL-A	ext3	D	D	D	D	D	D	0	0	0	3.31%
	XFS	D	D	D	D	D	D	0	0	0	0.92%
SQL-B	ext3	D	D	CD	CD	CD	CD	0	8.96%	0	3.24%
	XFS	CD	D	CD	CD	CD	CD	0	7.77%	0	3.90%
SQL-C	NTFS	D	D	D	D	D	D	0	0	0	8.08%

Durability violation is most common

DB	FS	W-1	W-2	W-3	W-4.1	W-4.2	W-4.3	A	C	I	D
TokyoCabinet	ext3	D	D	D	ACD	ACD	ACD	0.15%	0.14%	0	16.05%
	XFS	--	D	D	ACD	D	ACD	<0.01%	0.01%	0	4.38%
MariaDB	ext3	D	D	D	D	D	D	0	0	0	1.36%
	XFS	D	D	D	D	D	D	0	0	0	0.49%
LightningDB	ext3	--	--	--	--	--	D	0	0	0	0.05%
	XFS	--	--	--	--	--	--	0	0	0	0
SQLite	ext3	D	D	--	D	D	D	0	0	0	19.15%
	XFS	--	--	D	D	D	D	0	0	0	10.60%
KVS-A	ext3	--	--	Hang	--	--	--	0	0	0	0
	XFS	--	--	--	--	--	--	0	0	0	0
SQL-A	ext3	D	D	D	D	D	D	0	0	0	3.31%
	XFS	D	D	D	D	D	D	0	0	0	0.92%
SQL-B	ext3	D	D	CD	CD	CD	CD	0	8.96%	0	3.24%
	XFS	CD	D	CD	CD	CD	CD	0	7.77%	0	3.90%
SQL-C	NTFS	D	D	D	D	D	D	0	0	0	8.08%

Some violations are difficult to trigger

DB	FS	W-1	W-2	W-3	W-4.1	W-4.2	W-4.3	A	C	I	D
TokyoCabinet	ext3	D	D	D	ACD	ACD	ACD	0.15%	0.14%	0	16.05%
	XFS	--	D	D	ACD	D	ACD	<0.01%	0.01%	0	4.38%
MariaDB	ext3	D	D	D	D	D	D	0	0	0	1.36%
	XFS	D	D	D	D	D	D	0	0	0	0.49%
LightningDB	ext3	--	--	--	--	--	D	0	0	0	0.05%
	XFS	--	--	--	--	--	--	0	0	0	0
SQLite	ext3	D	D	--	D	D	D	0	0	0	19.15%
	XFS	--	--	D	D	D	D	0	0	0	10.60%
KVS-A	ext3	--	--	Hang	--	--	--	0	0	0	0
	XFS	--	--	--	--	--	--	0	0	0	0
SQL-A	ext3	D	D	D	D	D	D	0	0	0	3.31%
	XFS	D	D	D	D	D	D	0	0	0	0.92%
SQL-B	ext3	D	D	CD	CD	CD	CD	0	8.96%	0	3.24%
	XFS	CD	D	CD	CD	CD	CD	0	7.77%	0	3.90%
SQL-C	NTFS	D	D	D	D	D	D	0	0	0	8.08%

Some violations are difficult to trigger

DB	FS	W-1	W-2	W-3	W-4.1	W-4.2	W-4.3	A	C	I	D
TokyoCabinet	ext3	D	D	D	ACD	ACD	ACD	0.15%	0.14%	0	16.05%
	XFS	--	D	D	ACD	D	ACD	<0.01%	0.01%	0	4.38%
MariaDB	ext3	D	D	D	D	D	D	0	0	0	1.36%
	XFS	D	D	D	D	D	D	0	0	0	0.49%
LightningDB	ext3	--	--	--	--	--	D	0	0	0	0.05%
	XFS	--	--	--	--	--	--	0	0	0	0
SQLite	ext3	D	D	--	D	D	D	0	0	0	19.15%
	XFS	--	--	D	D	D	D	0	0	0	10.60%
KVS-A	ext3	--	--	Hang	--	--	--	0	0	0	0
	XFS	--	--	--	--	--	--	0	0	0	0
SQL-A	ext3	D	D	D	D	D	D	0	0	0	3.31%
	XFS	D	D	D	D	D	D	0	0	0	0.92%
SQL-B	ext3	D	D	CD	CD	CD	CD	0	8.96%	0	3.24%
	XFS	CD	D	CD	CD	CD	CD	0	7.77%	0	3.90%
SQL-C	NTFS	D	D	D	D	D	D	0	0	0	8.08%

Case Study: A TokyoCabinet Bug

- Failure symptoms

  faults injected in a region of operations cause:

  - A violation: a transaction is partially committed

  - D violation: some rows are irretrievable

  - C violation: retrievable rows by range query and point queries are different

Case Study: A TokyoCabinet Bug

Why recovery didn't work?

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...100..., x.tcb

tcbdbget()
...
```

Checker's trace when ACID violations were found

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...101..., x.tcb
tchdbwalrestore()
tcbdbget()
...
```

Checker's trace when no violation was found

Delta Debugging
[Zeller, SIGSOFT'02/FSE-10]

Case Study: A TokyoCabinet Bug

Why recovery didn't work?

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...100..., x.tcb
//no tchdbwalrestore()
tcbdbget()
...
```

Checker's trace when ACID violations were found

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...101..., x.tcb
tchdbwalrestore()
tcbdbget()
...
```

Checker's trace when no violation was found

Delta Debugging
[Zeller, SIGSOFT'02/FSE-10]

Case Study: A TokyoCabinet Bug

Why recovery didn't work?

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...100..., x.tcb
//no tchdbwalrestore()
tcbdbget()
...
```

Checker's trace when ACID violations were found

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...101..., x.tcb
tchdbwalrestore()
tcbdbget()
...
```

Checker's trace when no violation was found

Delta Debugging
[Zeller, SIGSOFT'02/FSE-10]

Case Study: A TokyoCabinet Bug

What happened at fault time?

Worker's trace
around the bug-triggering
fault points op#30–#90

```
...
mmap(8192, x.tcb, 0)
...
fsync(x.tcb.wal)
//op#,  LBA,  content, file,          syscall
op#26, 630, ....., x.tcb.wal, fsync(x.tcb.wal)
op#27, 960, ....., fs-j ,          fsync(x.tcb.wal)
op#28, 964, ....., fs-j ,          fsync(x.tcb.wal)
op#29, 480, ...100..., x.tcb ,     fsync(x.tcb.wal)
...
msync(x.tcb)
 //op#, LBA,  content, file, syscall
op#91, 480, ..101..., x.tcb, msync(x.tcb)
...
```

Why recovery didn't work?

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...100..., x.tcb
//no tchdbwalrestore()
tcbdbget()
...
```

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...101..., x.tcb
tchdbwalrestore()
tcbdbget()
...
```

Case Study: A TokyoCabinet Bug

What happened at fault time?

Worker's trace
around the bug-triggering
fault points op#30–#90

```
...
mmap(8192, x.tcb, 0)
...
fsync(x.tcb.wal)
//op#,  LBA,  content, file,        syscall
op#26, 630, ....., x.tcb.wal, fsync(x.tcb.wal)
op#27, 960, ....., fs-j ,         fsync(x.tcb.wal)
op#28, 964, ....., fs-j ,         fsync(x.tcb.wal)
op#29, 480, ...100..., x.tcb ,    fsync(x.tcb.wal)
...
msync(x.tcb)
 //op#, LBA,  content, file, syscall
op#91, 480, ...101..., x.tcb, msync(x.tcb)
...
```

Why recovery didn't work?

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...100..., x.tcb
//no tchdbwalrestore()
tcbdbget()
...
```

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...101..., x.tcb
tchdbwalrestore()
tcbdbget()
...
```

Case Study: A TokyoCabinet Bug

What happened at fault time?

Worker's trace
around the bug-triggering
fault points op#30–#90

```
...
mmap(8192, x.tcb, 0)
...
fsync(x.tcb.wal)
//op#,  LBA,  content, file,        syscall
op#26, 630, ....., x.tcb.wal, fsync(x.tcb.wal)
op#27, 960, ....., fs-j ,         fsync(x.tcb.wal)
op#28, 964, ....., fs-j ,         fsync(x.tcb.wal)
op#29, 480, ...100..., x.tcb ,    fsync(x.tcb.wal)
...
msync(x.tcb)
 //op#, LBA,  content, file, syscall
op#91, 480, ...101..., x.tcb, msync(x.tcb)
...
```

← – – Intended update

Why recovery didn't work?

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...100..., x.tcb
//no tchdbwalrestore()
tcbdbget()
...
```

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...101..., x.tcb
tchdbwalrestore()
tcbdbget()
...
```

Case Study: A TokyoCabinet Bug

What happened at fault time?

Worker's trace
around the bug-triggering
fault points op#30–#90

```
...
mmap(8192, x.tcb, 0)
...
fsync(x.tcb.wal)
//op#,  LBA,  content, file,        syscall
op#26, 630, ....., x.tcb.wal, fsync(x.tcb.wal)
op#27, 960, ....., fs-j ,      fsync(x.tcb.wal)
op#28, 964, ....., fs-j ,      fsync(x.tcb.wal)
op#29, 480, ...100..., x.tcb ,  fsync(x.tcb.wal)
...
msync(x.tcb)
 //op#, LBA,  content, file, syscall
op#91, 480, ...101..., x.tcb, msync(x.tcb)
...
```

Why recovery didn't work?

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...100..., x.tcb
//no tchdbwalrestore()
tcbdbget()
...
```

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...101..., x.tcb
tchdbwalrestore()
tcbdbget()
...
```

← - - Intended update

Case Study: A TokyoCabinet Bug

What happened at fault time?

Worker's trace
around the bug-triggering
fault points op#30–#90

```
...
mmap(8192, x.tcb, 0)
...
fsync(x.tcb.wal)
//op#,  LBA,  content, file,        syscall
op#26, 630, ....., x.tcb.wal, fsync(x.tcb.wal)
op#27, 960, ....., fs-j ,        fsync(x.tcb.wal)
op#28, 964, ....., fs-j ,        fsync(x.tcb.wal)
op#29, 480, ...100..., x.tcb ,   fsync(x.tcb.wal)
...
msync(x.tcb)
 //op#, LBA,  content, file, syscall
op#91, 480, ...101..., x.tcb, msync(x.tcb)
...
```

← – – Unintended update!

← – – Intended update

Why recovery didn't work?

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...100..., x.tcb
//no tchdbwalrestore()
tcbdbget()
...
```

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...101..., x.tcb
tchdbwalrestore()
tcbdbget()
...
```

Case Study: A TokyoCabinet Bug

What happened at fault time?

Worker's trace
around the bug-triggering
fault points op#30–#90

```
...
mmap(8192, x.tcb, 0)
...
fsync(x.tcb.wal)
//op#,  LBA,  content, file,          syscall
op#26, 630, ....., x.tcb.wal, fsync(x.tcb.wal)
op#27, 960, ....., fs-j ,          fsync(x.tcb.wal)
op#28, 964, ....., fs-j ,          fsync(x.tcb.wal)
op#29, 480, ...100..., x.tcb ,     fsync(x.tcb.wal)
...
msync(x.tcb)
 //op#, LBA,  content, file, syscall
op#91, 480, ...101..., x.tcb, msync(x.tcb)
...
```

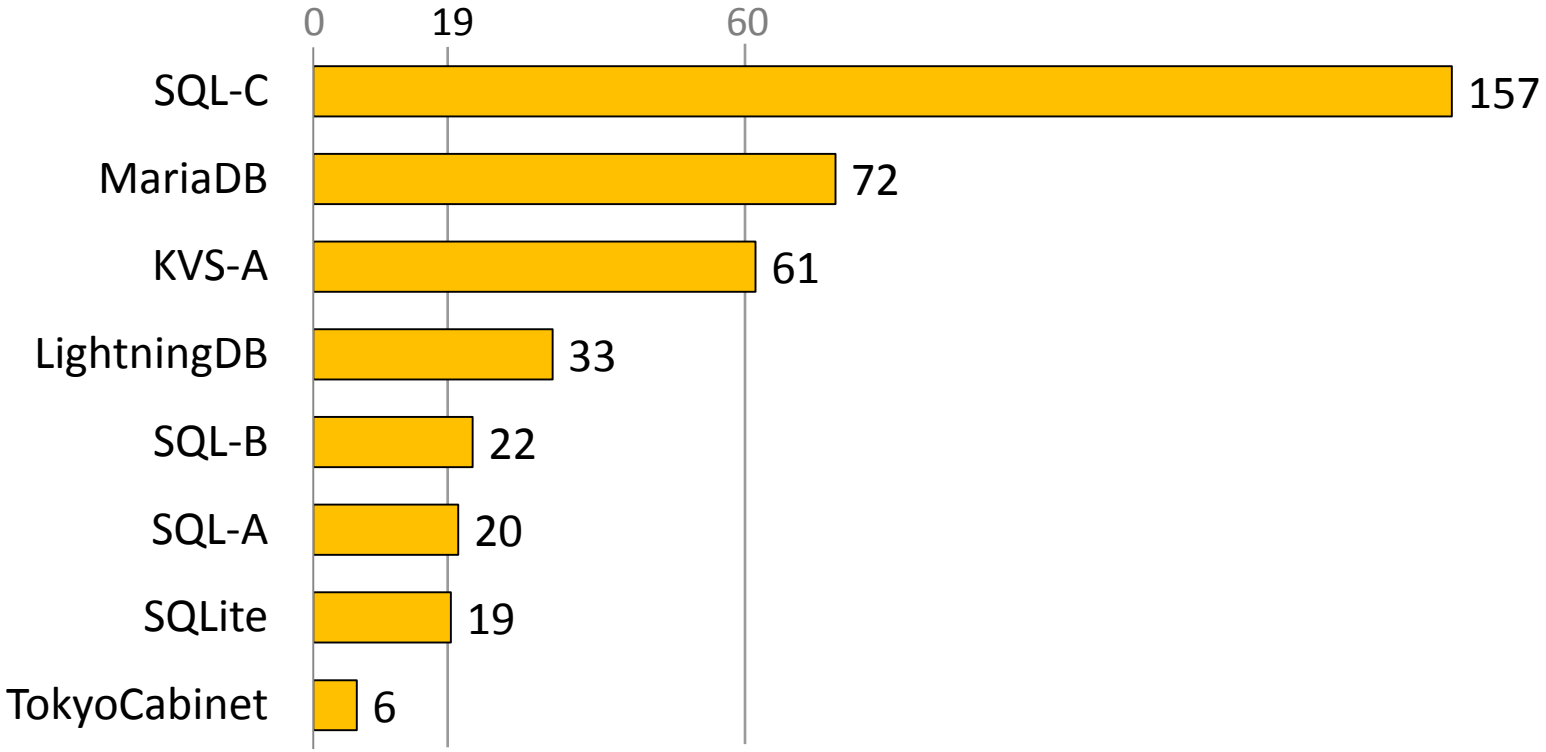
Why recovery didn't work?

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...100..., x.tcb
//no tchdbwalrestore()
tcbdbget()
...
```

```
...
tchdbopenimpl(x.tcb)
...
open(x.tcb) = 3
read(x.tcb) = 256
//op#, LBA,  content, file
op#1, 480, ...101..., x.tcb
tchdbwalrestore()
tcbdbget()
...
```

One solution:
Failure-atomic msync()
[Park *et.al.*, EuroSys'13]

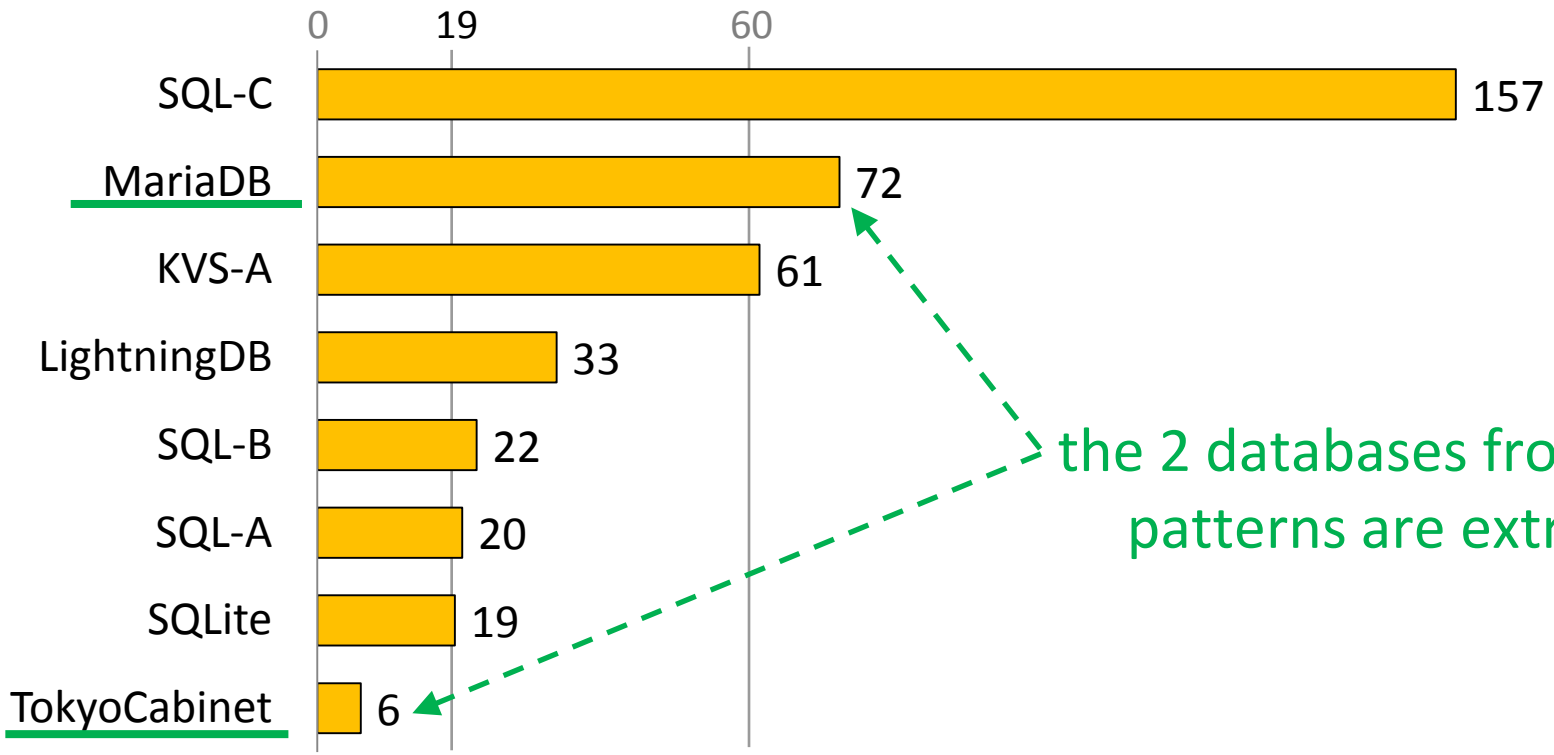
Patterns reduce required test points greatly while achieving similar coverage



reduction factor =

$$\frac{\#\ of\ test\ points\ under\ exhaustive\ policy}{\#\ of\ test\ points\ under\ pattern\ policy}$$

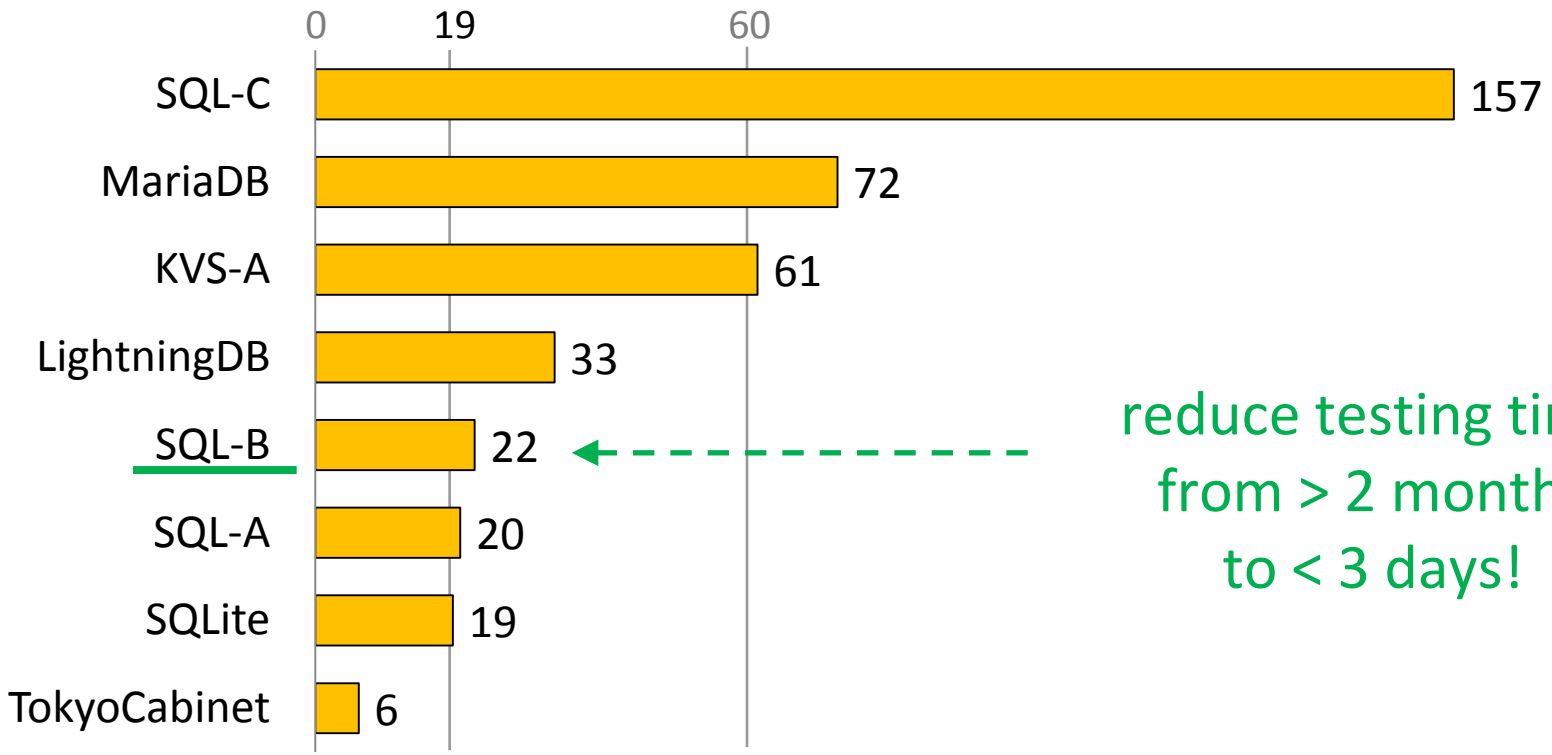
Patterns reduce required test points greatly while achieving similar coverage



reduction factor =

$$\frac{\#\ of\ test\ points\ under\ exhaustive\ policy}{\#\ of\ test\ points\ under\ pattern\ policy}$$

Patterns reduce required test points greatly while achieving similar coverage



reduce testing time from > 2 months to < 3 days!

reduction factor =

$$\frac{\#\ of\ test\ points\ under\ exhaustive\ policy}{\#\ of\ test\ points\ under\ pattern\ policy}$$

Conclusions & Future Work

- A wake-up call

  - traditional testing methodology may not be enough for today's complex storage systems

  - thorough testing requires purpose-built workloads and intelligent fault injection techniques

Conclusions & Future Work

- A wake-up call

  - traditional testing methodology may not be enough for today's complex storage systems

  - thorough testing requires purpose-built workloads and intelligent fault injection techniques

- Different layers in OS can help in different ways

  - iSCSI: fault injection w/ high portability & high fidelity

  - LBA & syscall: generic behavior patterns

  - combined multi-layer info: clear whole picture of complicated scenarios

Conclusions & Future Work

- We should bridge the gaps of understanding/assumptions!

between
DB & OS

between
User & DB

Conclusions & Future Work

- We should bridge the gaps of understanding/assumptions!

Pop Quiz: true or false?

between DB & OS	“mmap'ed files are not updated until msync()”	
	“file-length update are persistent after fdatasync()”	
between User & DB	“durability is provided by the default configuration”	
	“transactions are durable after COMMIT”	

Conclusions & Future Work

- We should bridge the gaps of understanding/assumptions!

Pop Quiz: true or false?

between DB & OS	“mmap'ed files are not updated until msync()”	false
	“file-length update are persistent after fdatasync()”	
between User & DB	“durability is provided by the default configuration”	
	“transactions are durable after COMMIT”	

Conclusions & Future Work

- We should bridge the gaps of understanding/assumptions!

Pop Quiz: true or false?

between DB & OS	“mmap'ed files are not updated until msync()”	false
	“file-length update are persistent after fdatasync()”	depends!
between User & DB	“durability is provided by the default configuration”	depends!
	“transactions are durable after COMMIT”	depends!

Conclusions & Future Work

- We should bridge the gaps of understanding/assumptions!

Pop Quiz: true or false?

between DB & OS	“mmap'ed files are not updated until msync()”	false
	“file-length update are persistent after fdatasync()”	depends!
between User & DB	“durability is provided by the default configuration”	depends!
	“transactions are durable after COMMIT”	depends!

Thank you!