Mitigating Sync Amplification for Copy-on-Write Virtual Disk

QINGSHU CHEN, LIANG LIANG, YUBIN XIA,
HAIBO CHEN, HYUNSOO KIM

Institute of Parallel and Distributed Systems
Shanghai Jiao Tong University, China
Samsung Electronics Co., Ltd

FAST'16 The 14th USENIX Conference on File and Storage Technologies

Wide Use of CoW Virtual Disks

Popular storage backend for virtual machines

.vmdk
VMWare

CoW virtual disks provide many useful features

- E.g., dynamic growing of image size, snapshot
- Ease tasks such as VM deployment, backup

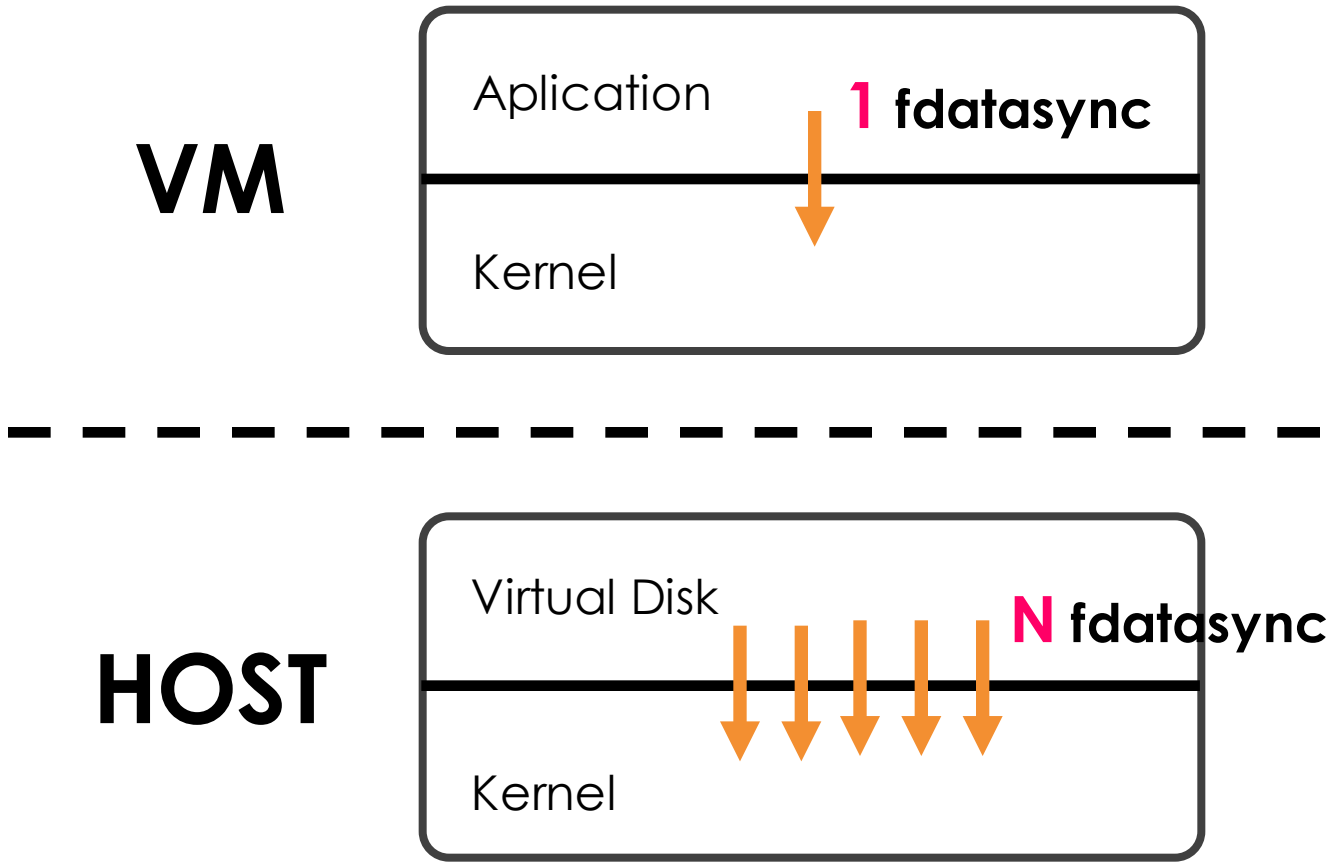
.qcow2
QEMU/
KVM

Virtual disks have been widely used in major cloud infrastructures

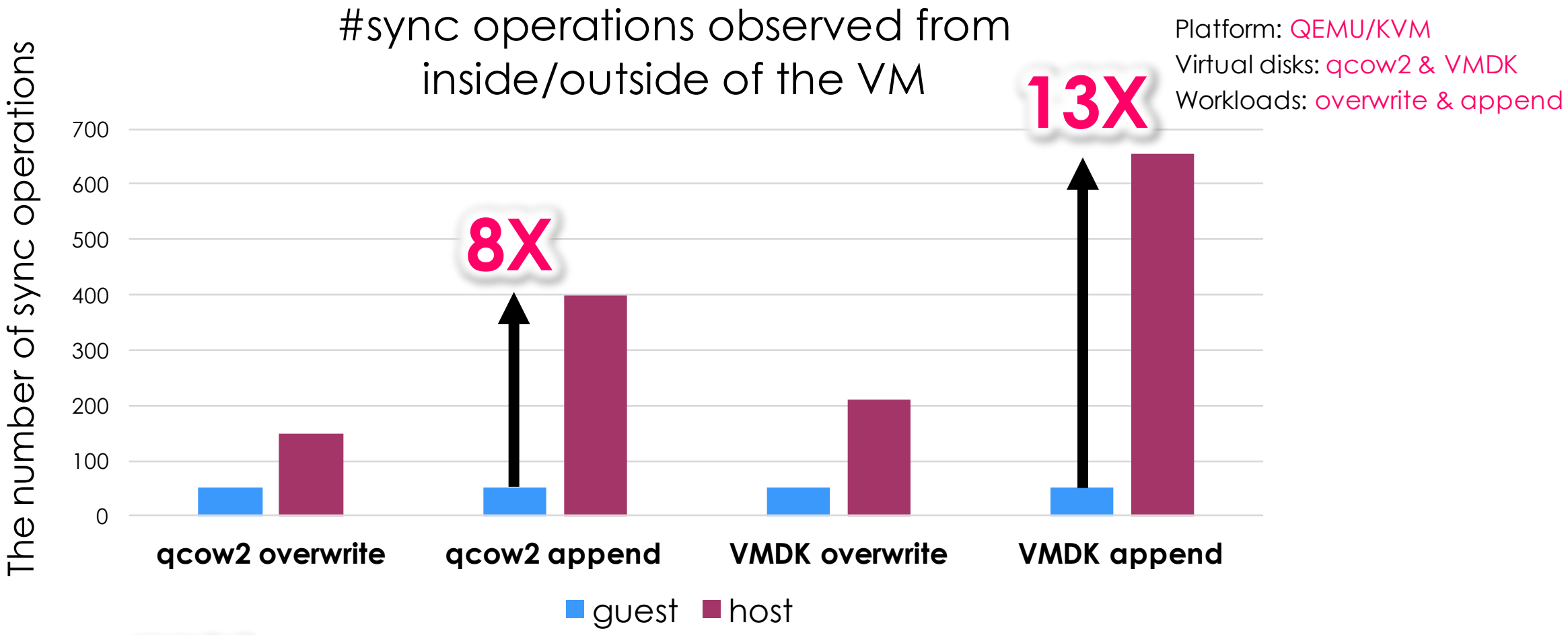
- E.g., OpenStack

Sync Amplification on CoW Virtual Disks

CoW virtual disks introduce sync amplification problem



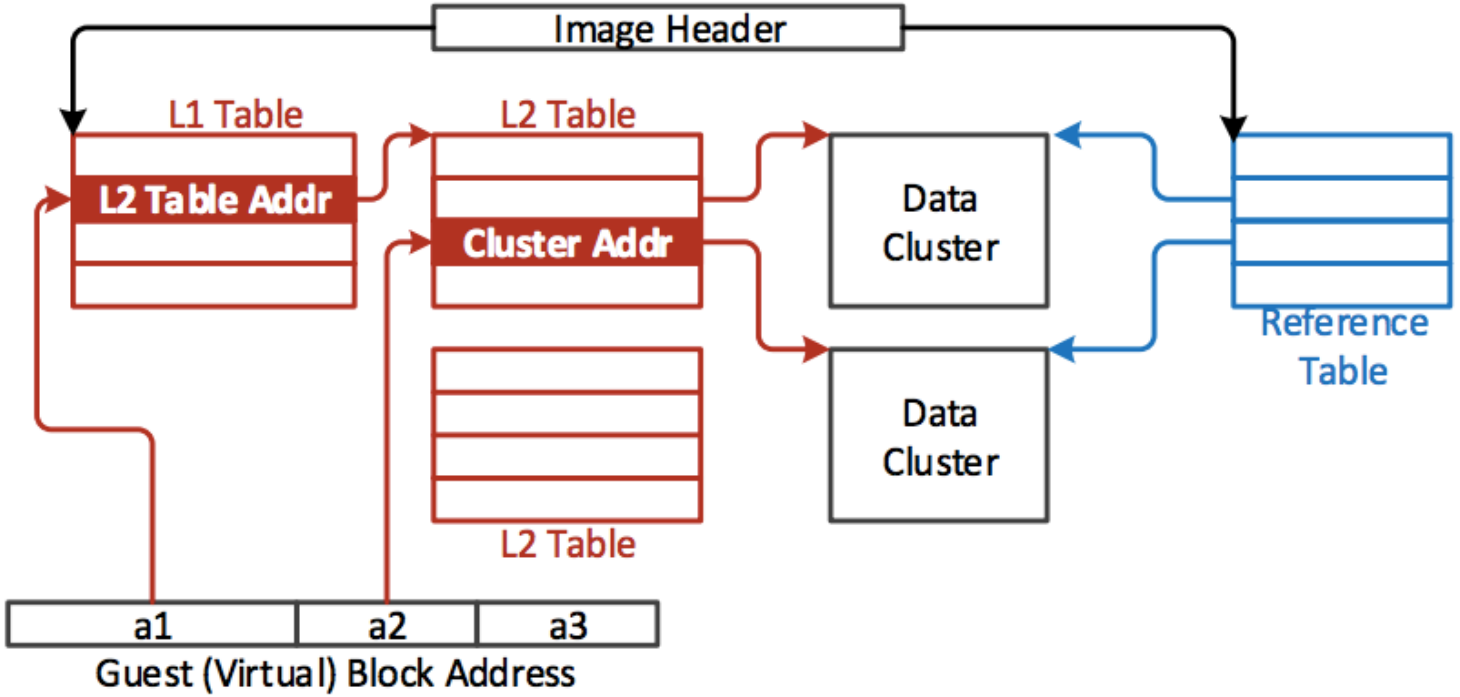
Sync Amplification on CoW Virtual Disks



50% Performance Slowdown for Varmail

How Sync Amplification Happens?

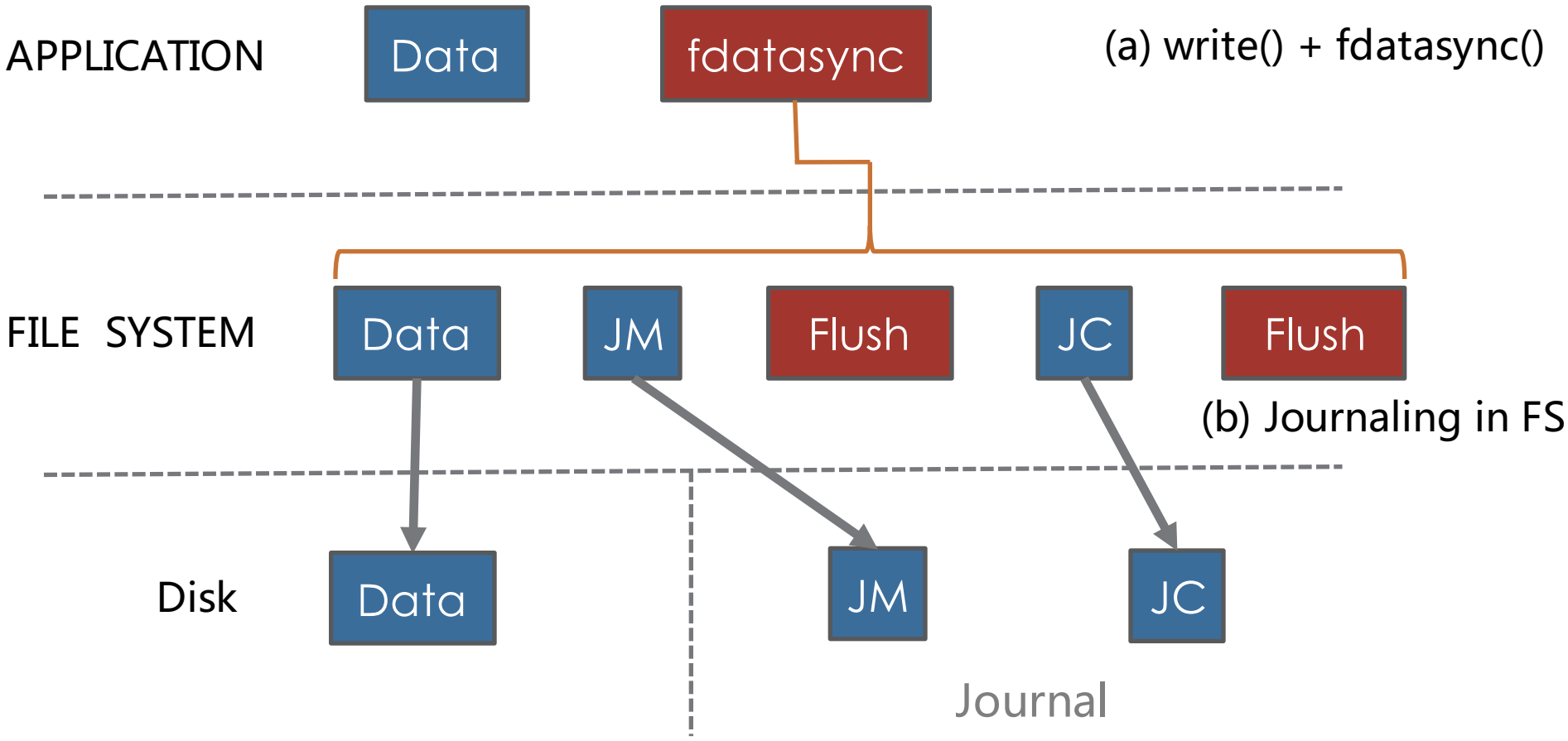
CoW virtual disks introduce additional metadata, which needs extra syncs to retain crash consistency



Metadata for qcow2:

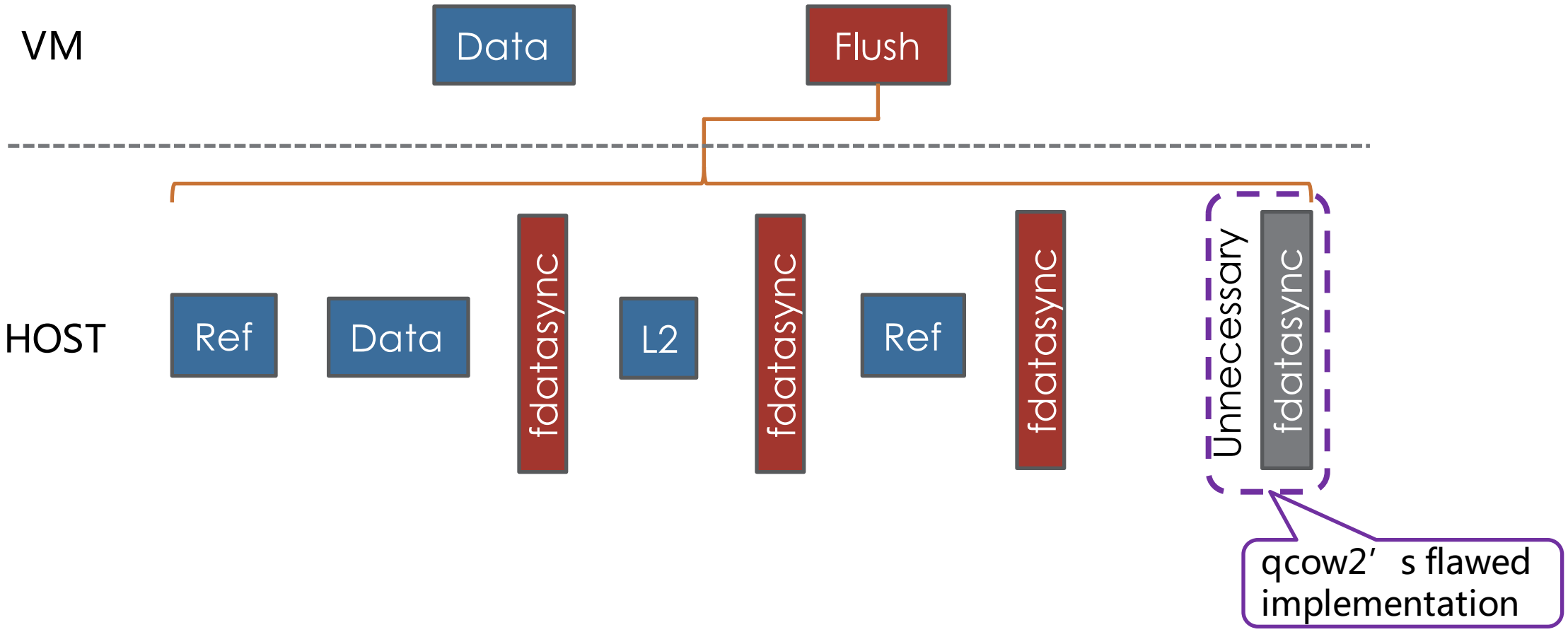
- A two level lookup table(L1 table, L2 table), a reference table

File System Journaling (2X)



JM: metadata logged in the journal JC: journal commit block

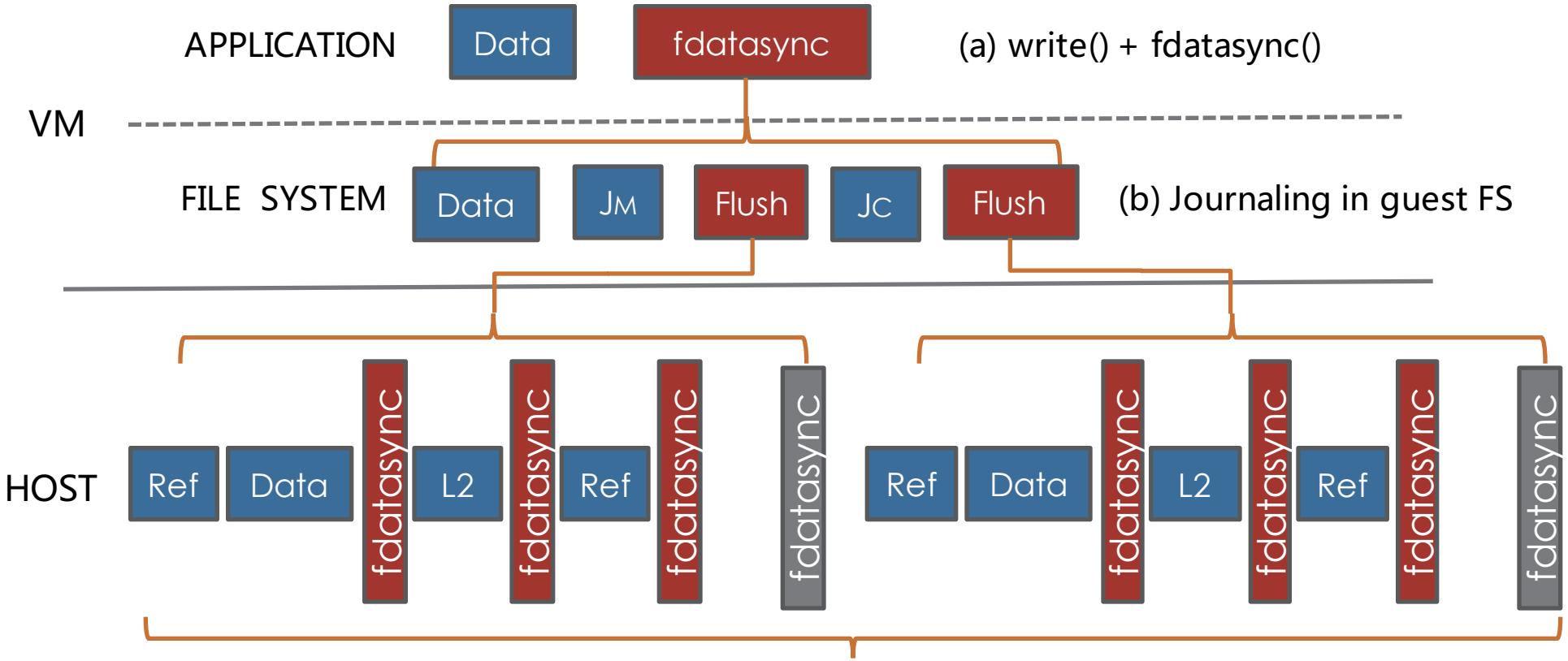
Qcow2 Updating Process (4X)



Ref: reference table L2: level-2 lookup table

Why 8X ?

The process of appending a block of data to an empty file in the guest VM:



8 sync operations

Virtual Disks Have Serious Sync Amplification Problem

Our Optimizations

Reduce the number of sync operations

- Per virtual disk journaling
- Dual-mode journaling

Reduce the overhead for each sync operation

- Adaptive preallocation

Our Optimizations

Reduce the number of sync operations

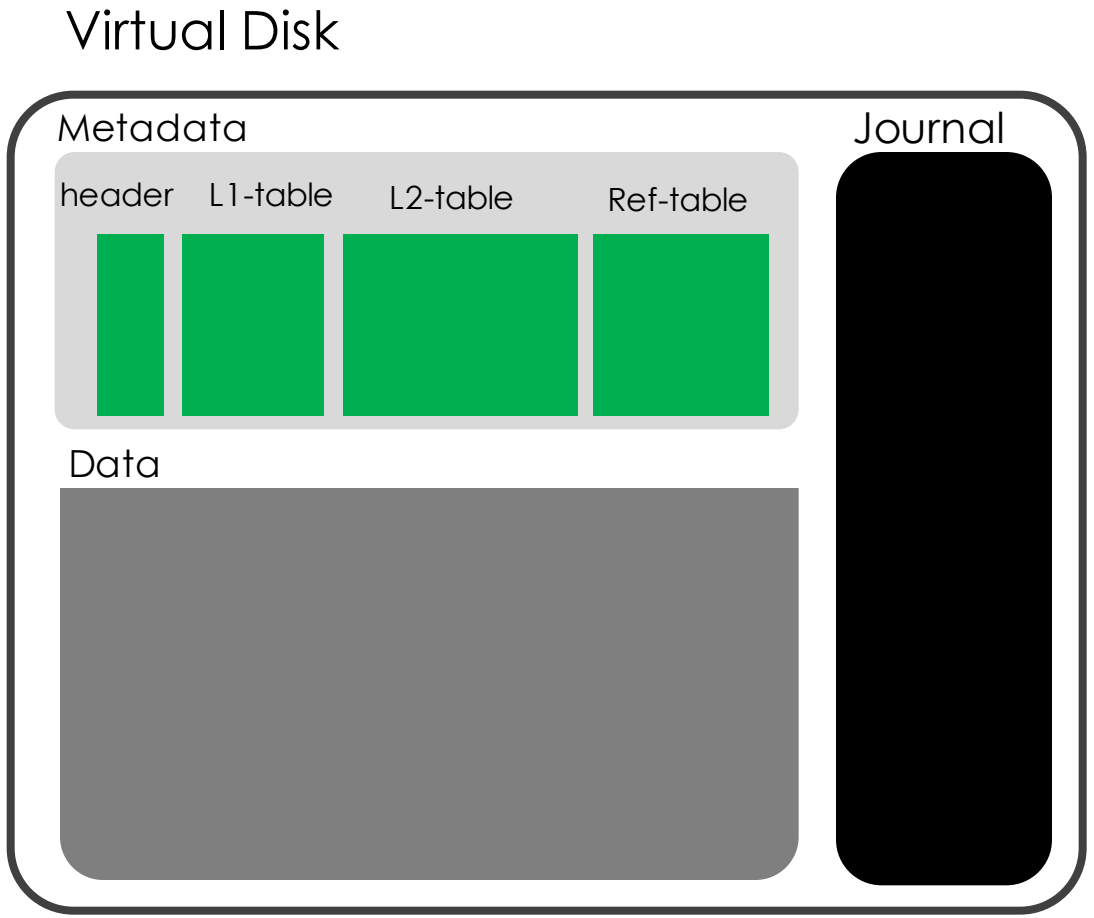
- Per virtual disk journaling
- Dual mode journaling

Reduce the overhead for each sync operation

- Adaptive preallocation

Per Virtual Disk Journaling

Maintains an independent and internal journal for each virtual disk

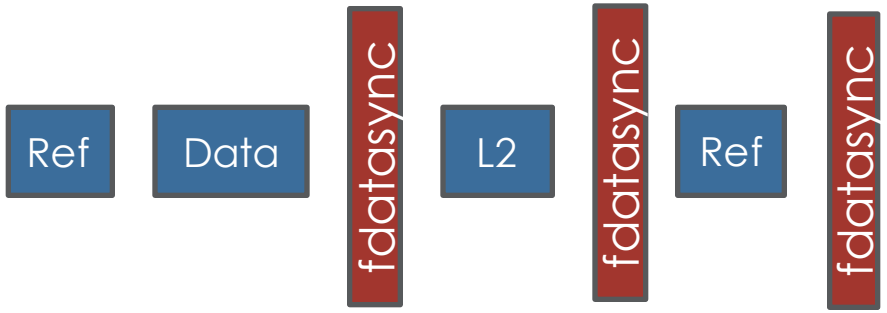


No-data Mode Journaling

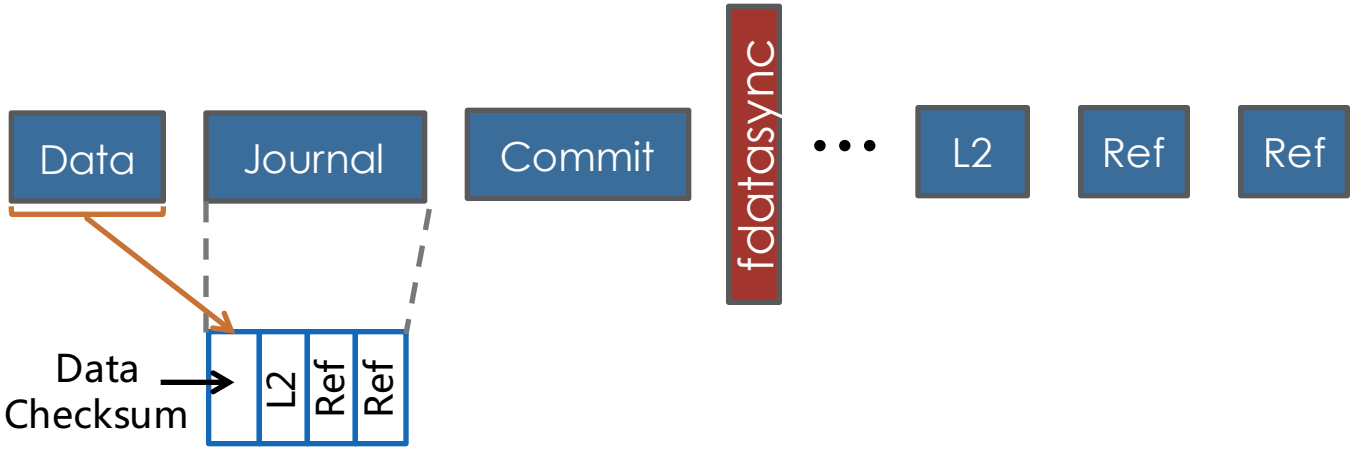
Only log data checksum and metadata in journal

Data sequence:

Original system:



No-data mode:

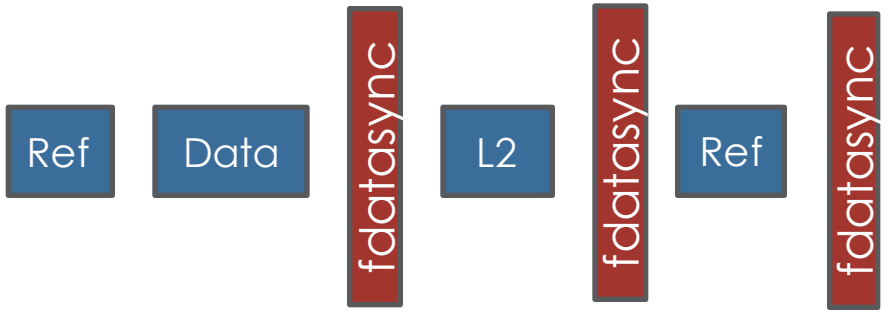


No-data Mode Journaling

Only log data checksum and metadata in journal

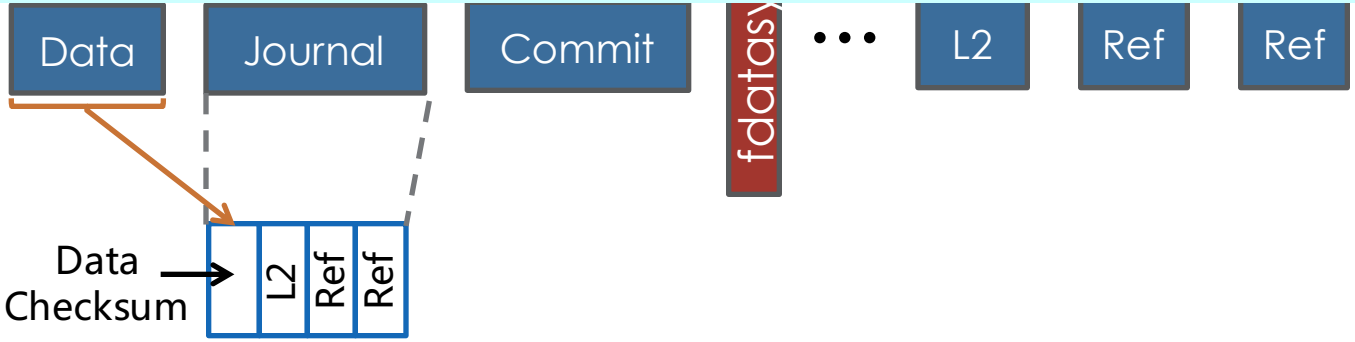
Data sequence:

Original system:



The number of fdatasync decreases from 3 to 1

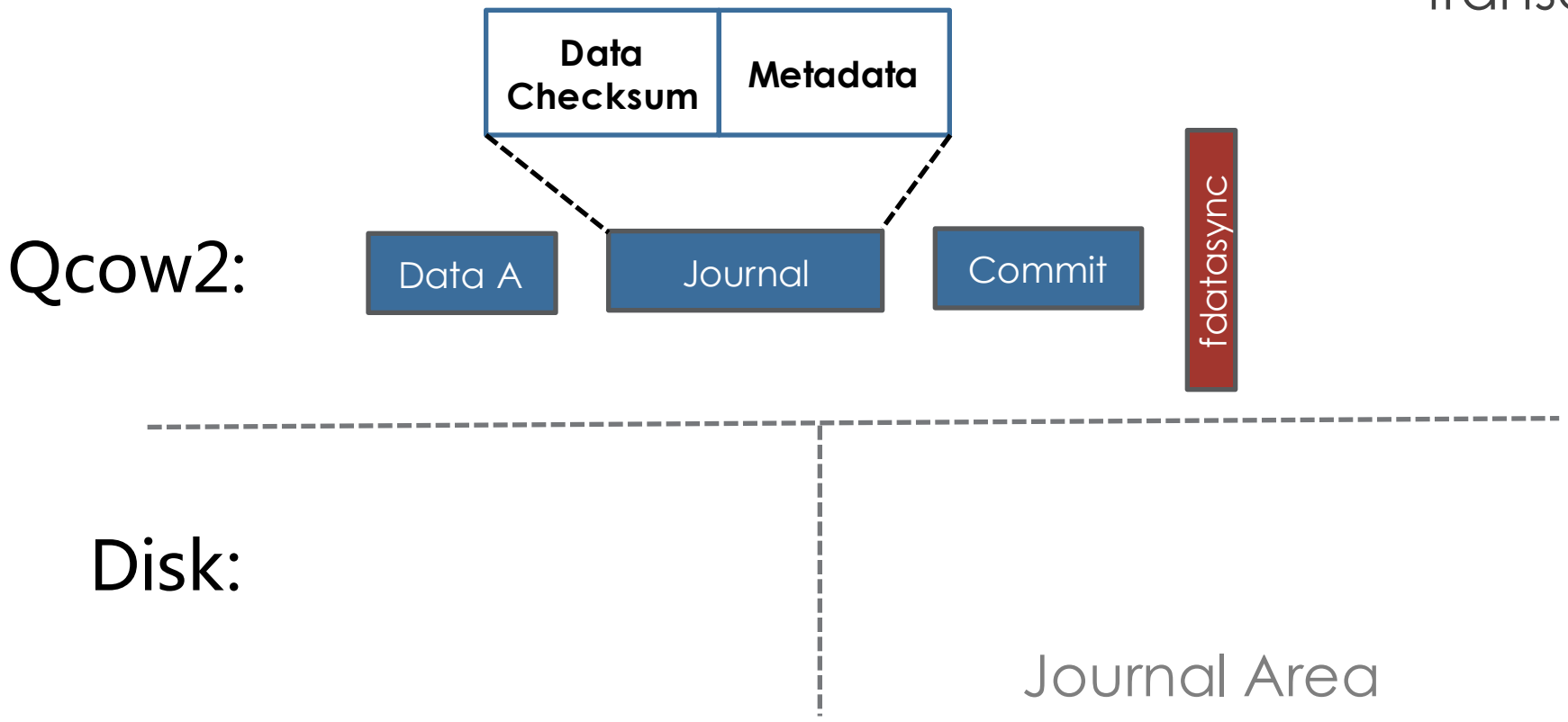
No-data mode:



The Overwriting Problem

Data overwriting may render inconsistent recovery for no-data mode journaling

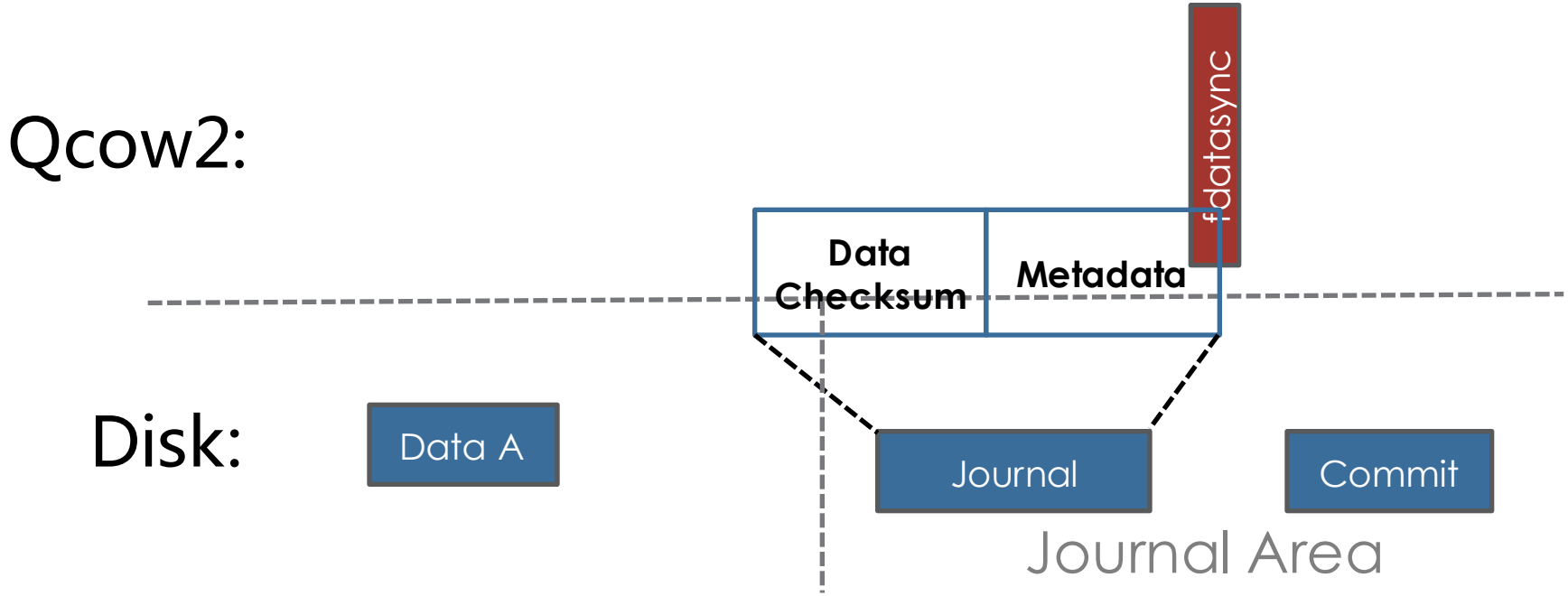
1) Commit the current transaction



The Overwriting Problem

Data overwriting may render inconsistent recovery for no-data mode journaling

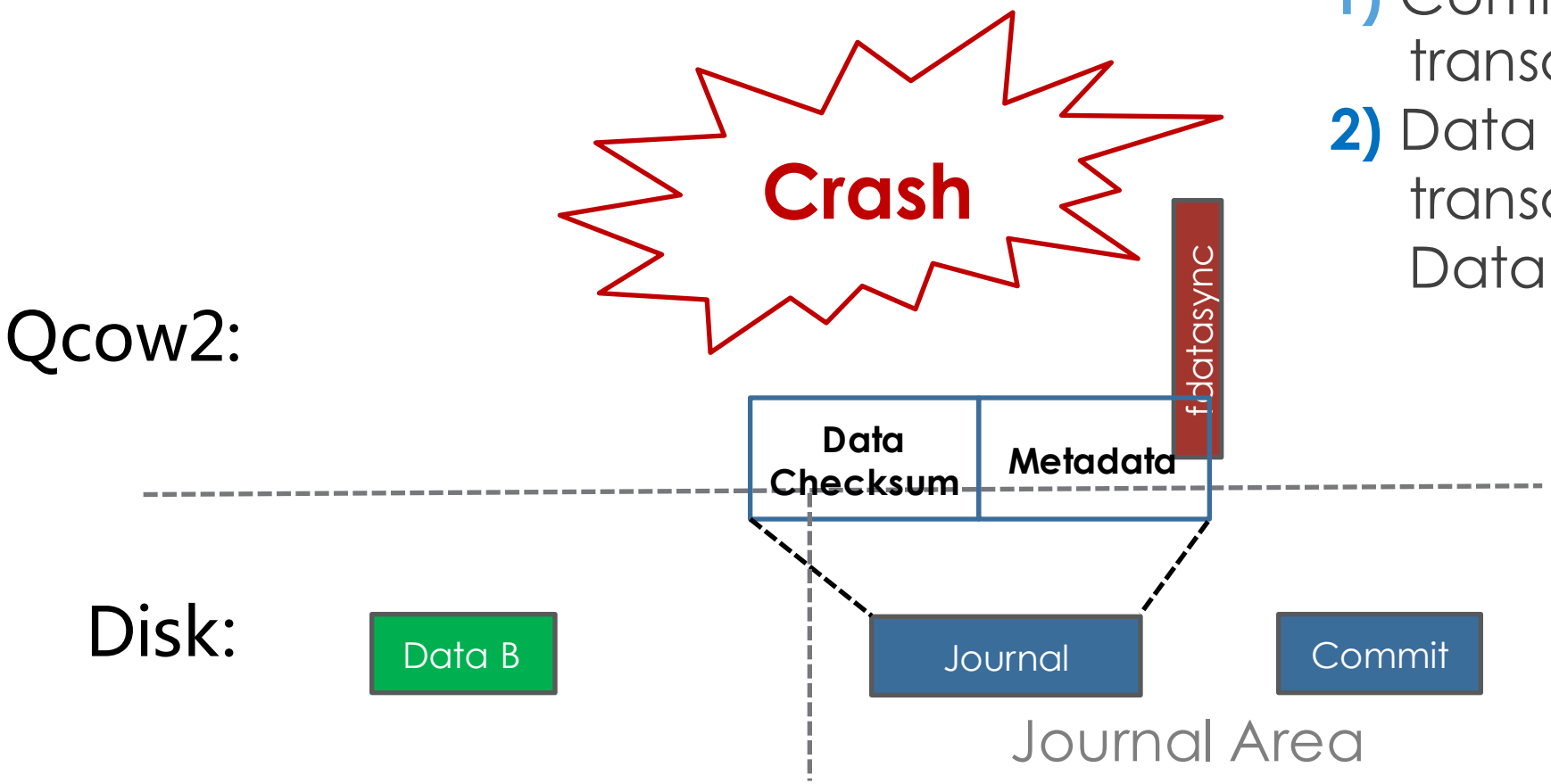
1) Commit the current transaction



The Overwriting Problem

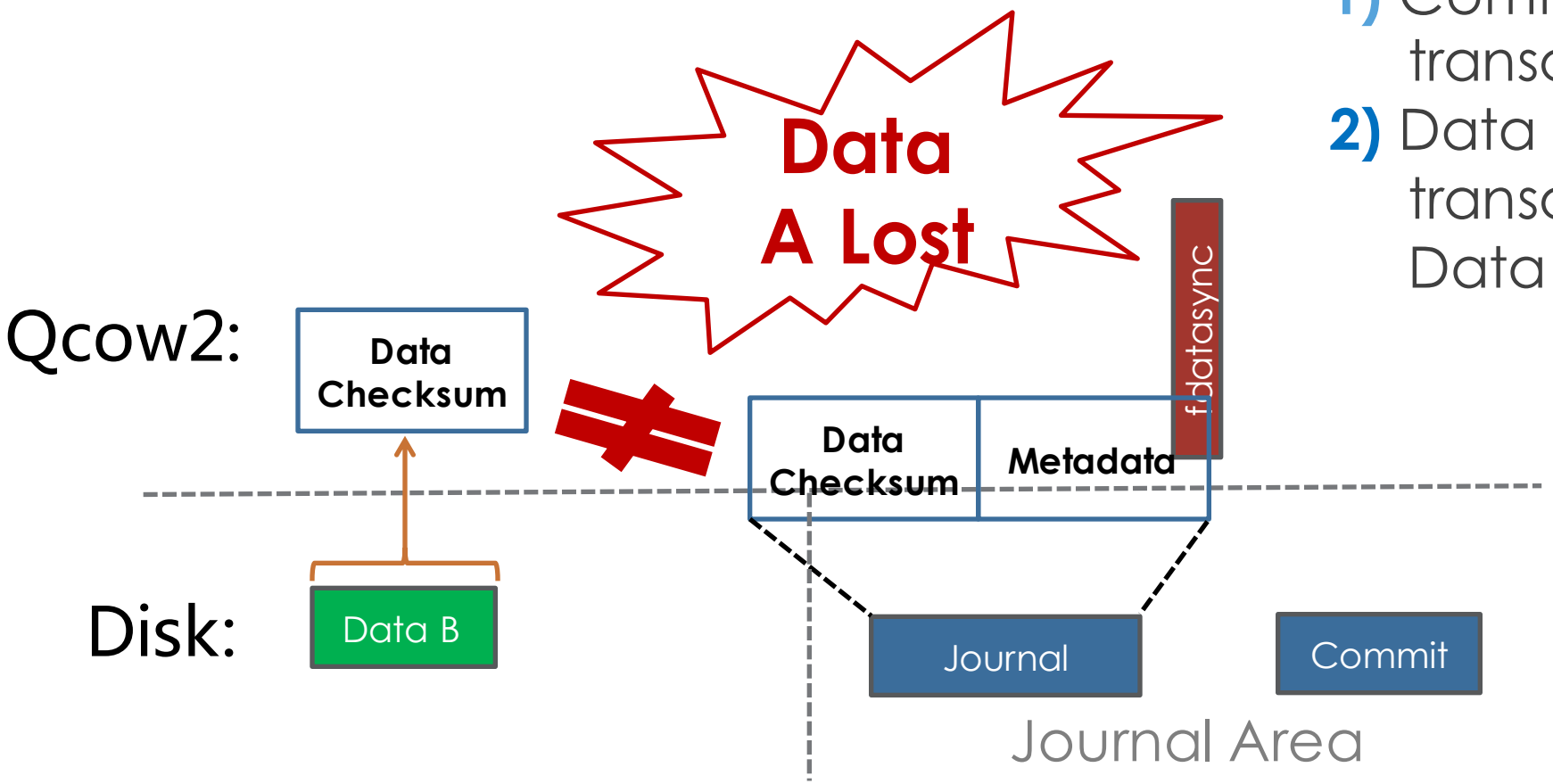
Data overwriting may render inconsistent recovery for no-data mode journaling

1) Commit the current transaction
2) Data B in the next transaction overwrites Data A



The Overwriting Problem

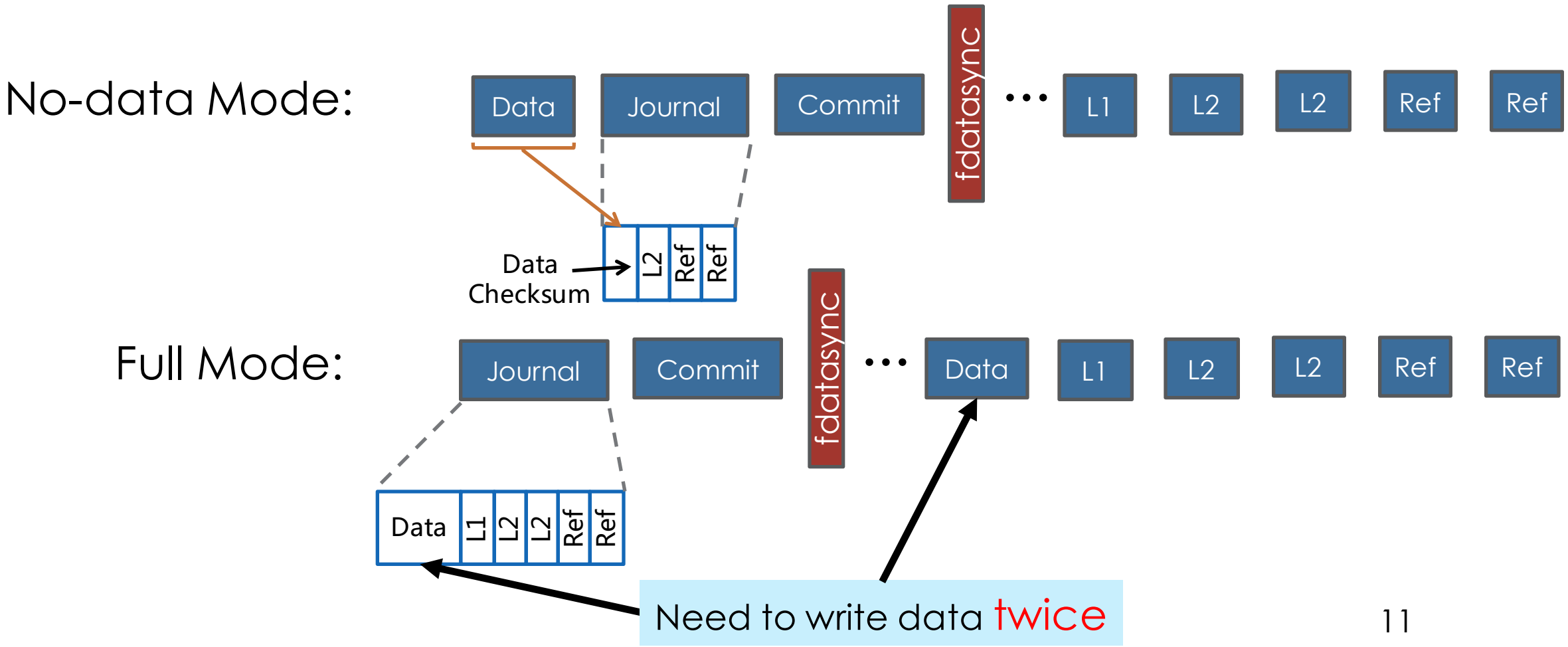
Data overwriting may render inconsistent recovery for no-data mode journaling



1) Commit the current transaction
2) Data B in the next transaction overwrites Data A

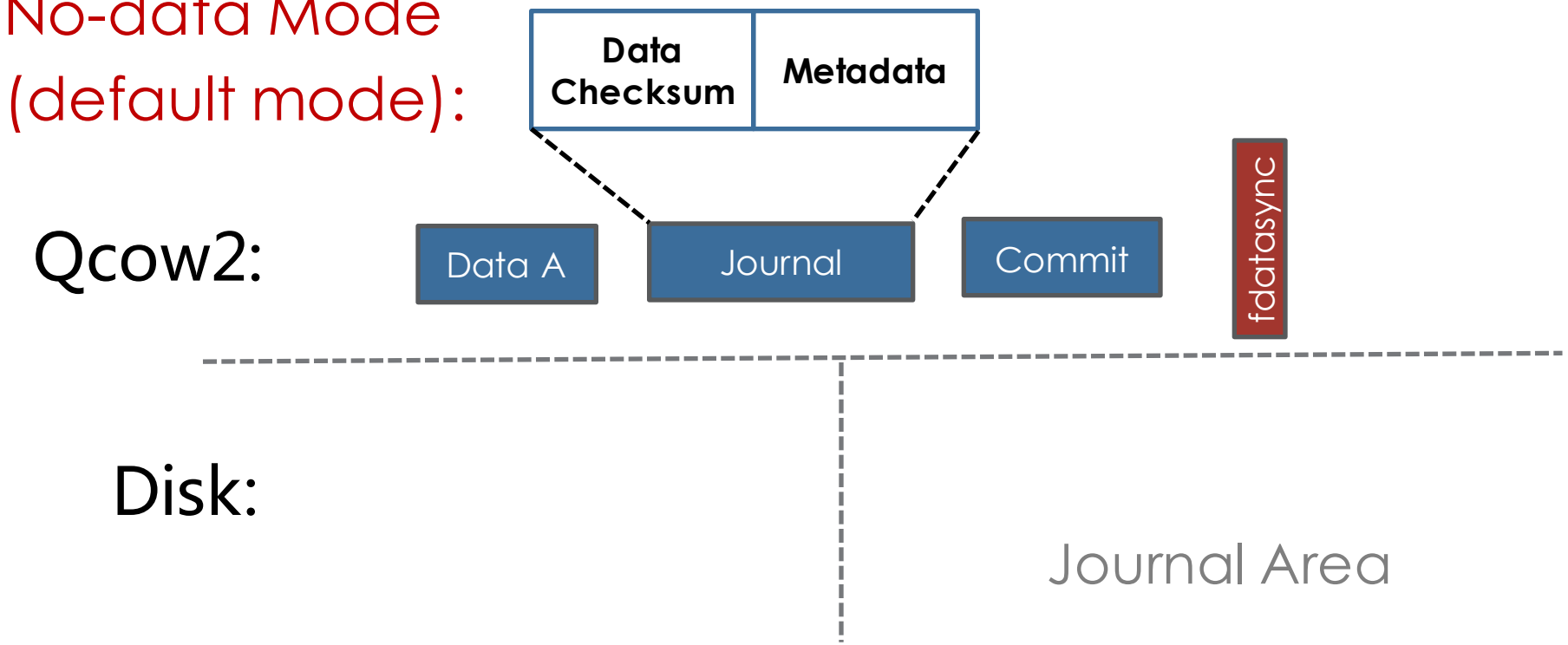
Full Mode Journaling

The journal contains both the metadata and the data



Dynamic Switch b/w Two Journaling Modes

No-data Mode (default mode):



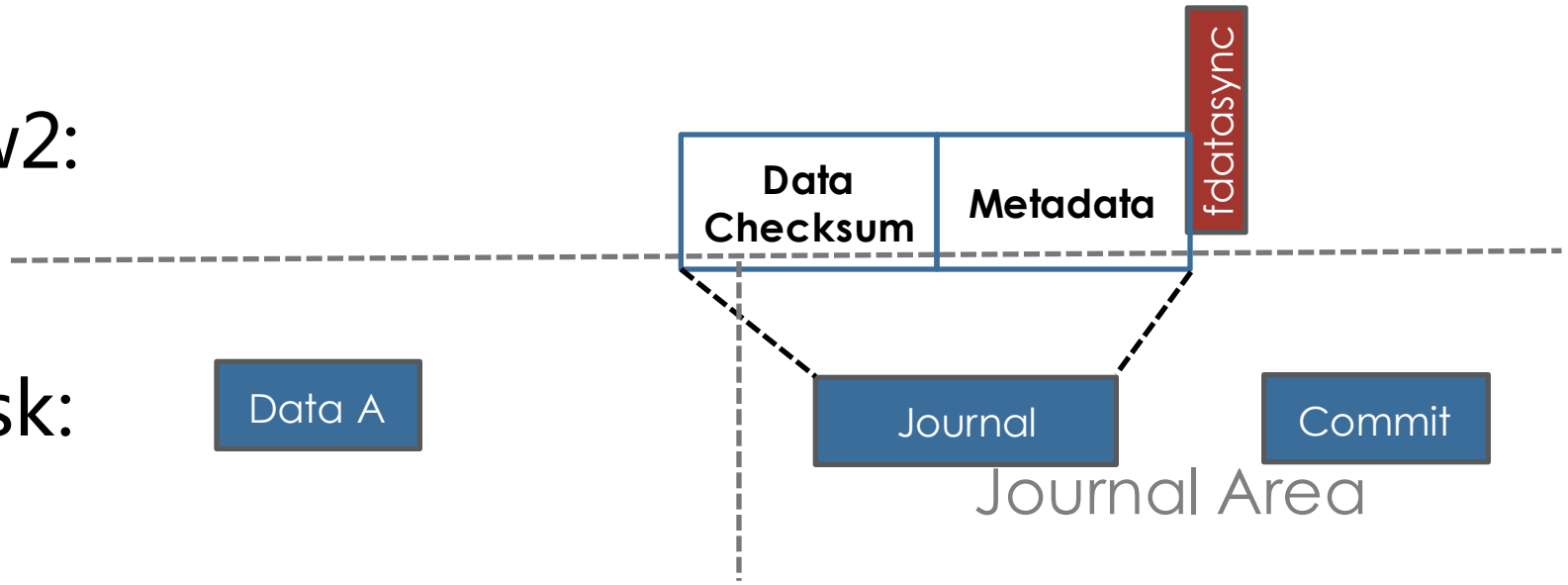
Dynamic Switch b/w Two Journaling Modes

1) Commit the transaction

No-data Mode
(default mode):

Qcow2:

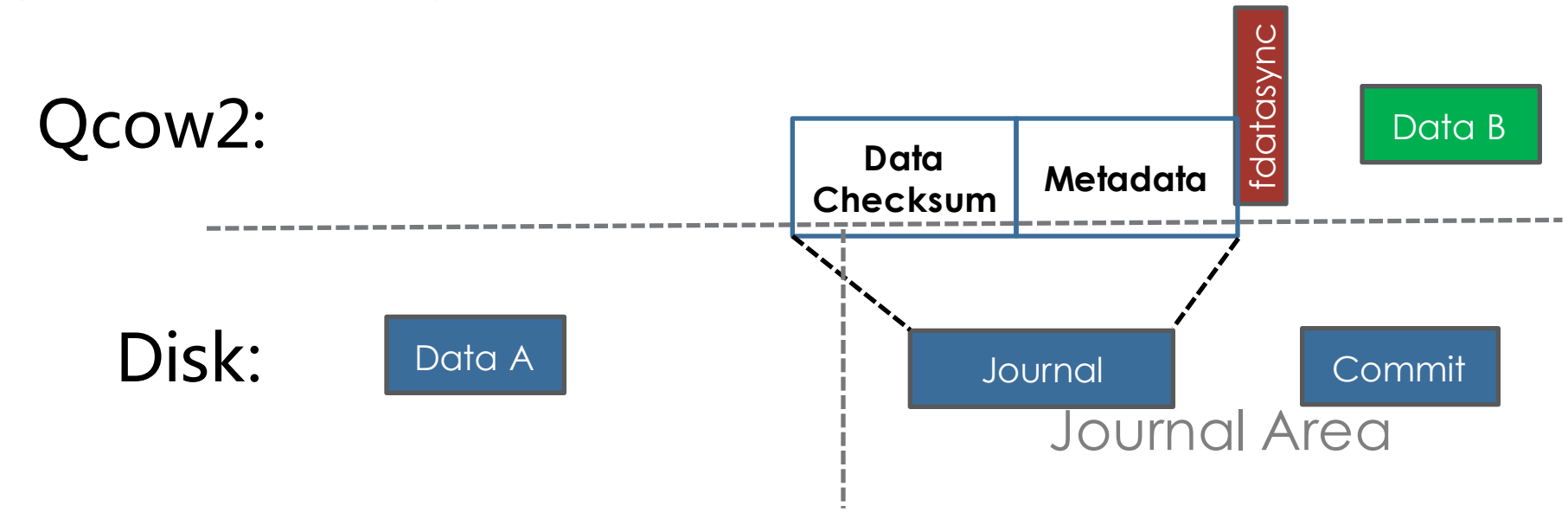
Disk:



Dynamic Switch b/w Two Journaling Modes

1) Commit the transaction
2) Data B in the next transaction wants to overwrite Data A

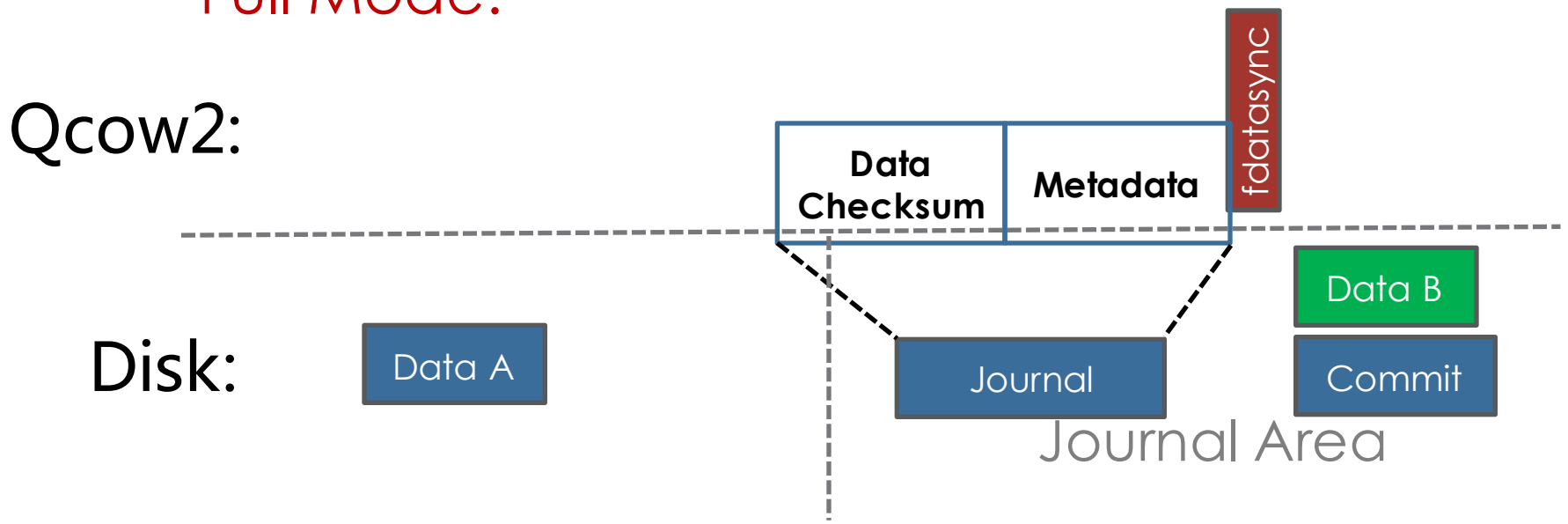
No-data Mode (default mode):



Dynamic Switch b/w Two Journaling Modes

1) Commit the transaction
2) Data B in the next transaction wants to overwrite Data A

Full Mode:



Our Optimizations

Reduce the number of sync operations

- Per virtual disk journaling
- Dual mode journaling

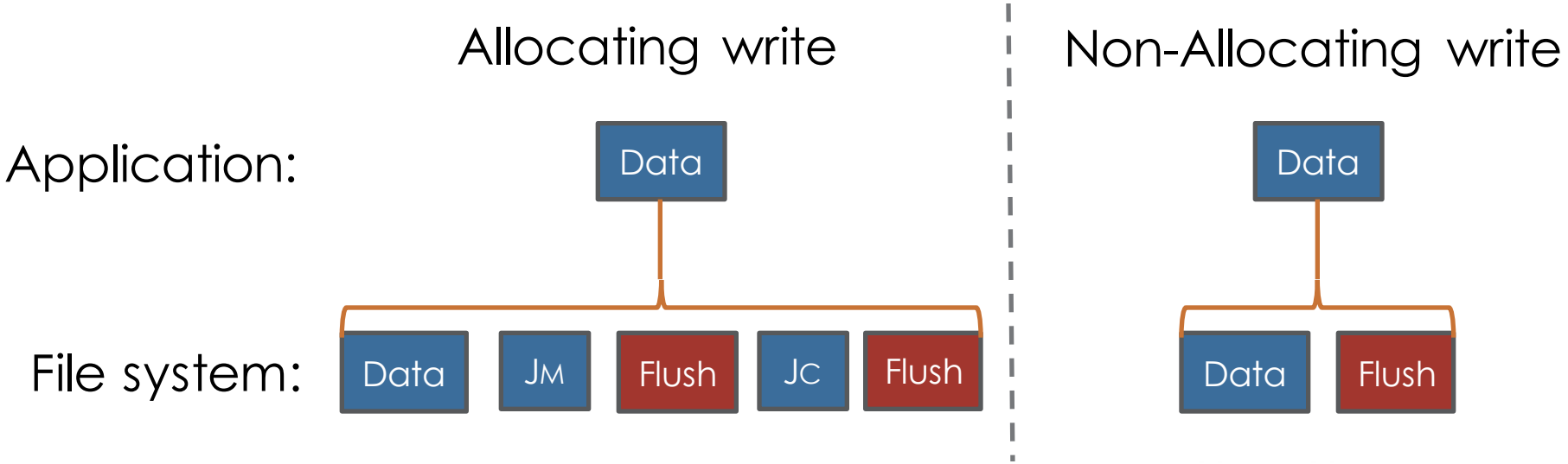
Reduce the overhead for each sync operation

- Adaptive preallocation

Allocating Write *VS* Non-Allocating Write

Whether the FS needs to allocate new block for a write?

Allocating write & Non-Allocating write



Non-allocating write is much faster than allocating write

Adaptive Preallocation

When the image file grows, append more blocks than those actually required

An allocating write in the future can be transformed into a non-allocating write

Implementation

Implemented the optimizations on QEMU-2.1.2 (1300 LoC)

Remove unnecessary sync operation (Caused by qcow2's flawed implementation of QEMU)

Evaluation Questions

Can our optimized system retain crash consistency?

How does our optimized system perform?

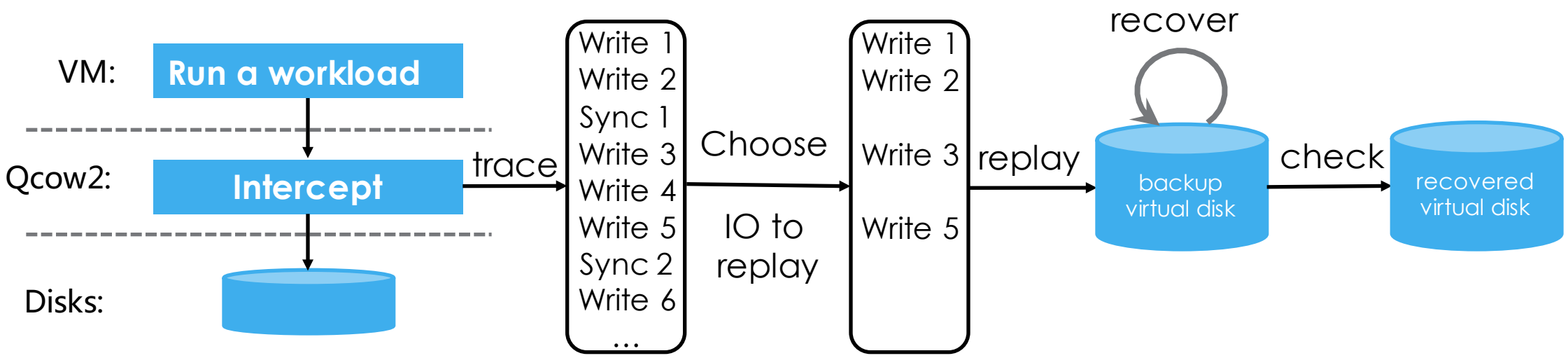
- Different workloads
- Different types of disk

Evaluation Platform

Hardware	Software
4-core Intel Xeon E3-1230 CPU 8G DDR3 Memory 1 TB WDC HDD 120G Samsung 850 EVO SSD	Hypervisor: KVM Host OS: Ubuntu 14.04 Guest OS: Ubuntu 12.04 FS: Both Ext4 in guest & host VM cache mode: write back

Can our optimized system retain crash consistency?

Implemented a trace and replay framework



We simulated 400 crash scenarios, our optimized system can recovery correctly for all scenarios

How does our optimized system perform?

Benchmarks:

- varmail & tpcc

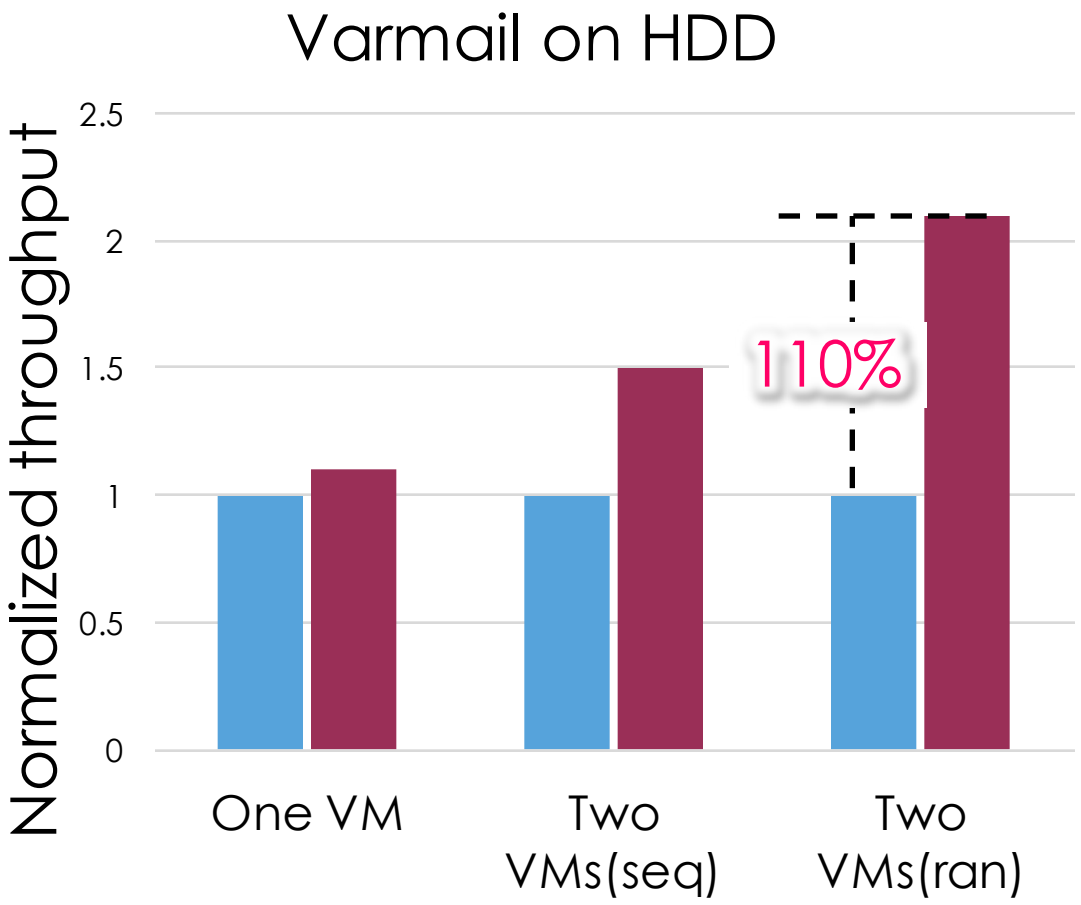
Disk type:

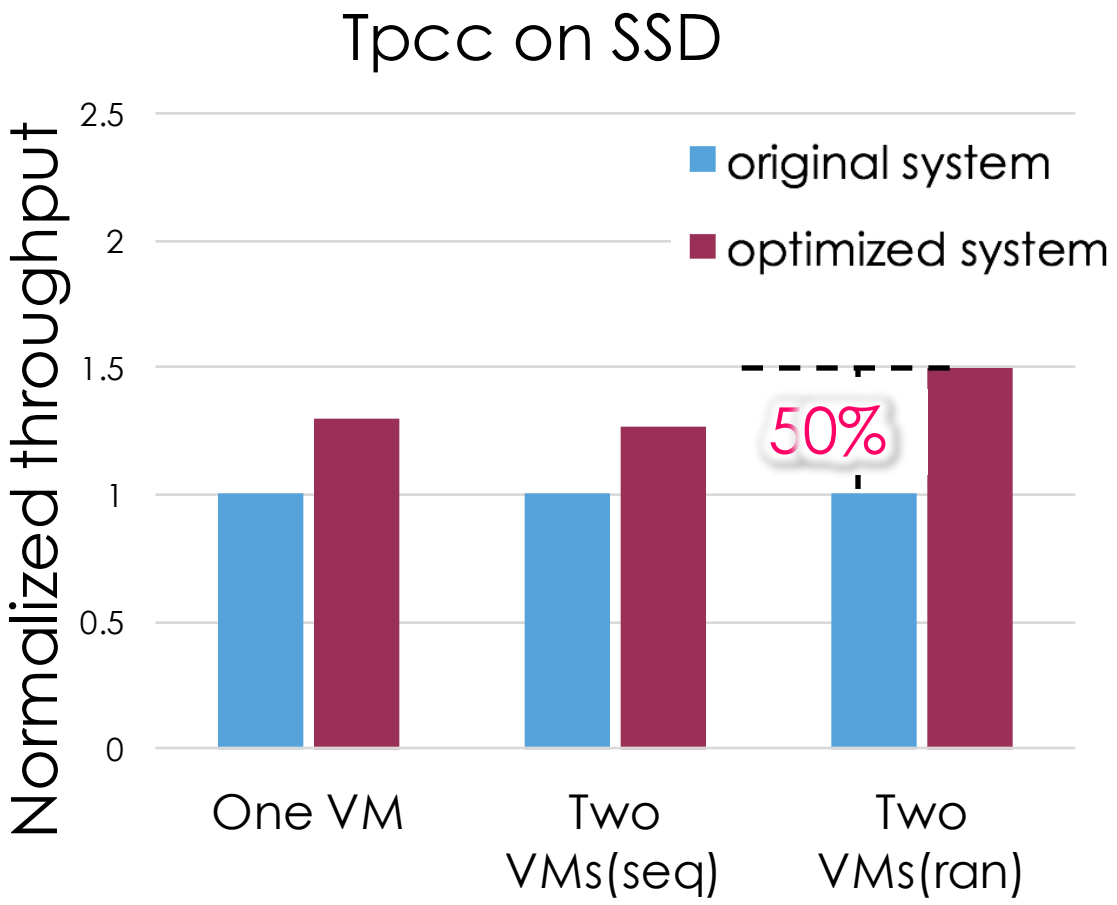
- HDD and SSD

Three settings:

- **One VM:** boot one VM and runs the tests
- **Two VMs (seq)**: boot two VMs, one runs the tests, one runs a sequential workload
- **Two VMs (ran)**: boot two VMs, one fruns the tests, one runs a random workload

Performance





Conclusion

Uncover the sync amplification problem on copy-on-write virtual disks

Three optimizations to minimize sync operations while preserving crash consistency

Notably improve some I/O intensive workloads

Conclusion

Uncover the sync amplification problem on copy-on-write virtual disks

Three optimizations to minimize sync operations while preserving crash consistency

Notably improve some I/O intensive workloads

Thank you

http://ipads.se.sjtu.edu.cn

Institute of Parallel and Distributed Systems