Weak Durability

Remzi H. Arpaci-Dusseau
University of Wisconsin-Madison

How Does a File System Write To Disk?

Journaling File System

Journaling File System

What is goal of journaling?

- Crash consistency

Journaling File System

What is goal of journaling?

- Crash consistency

How achieved?

- Use **write-ahead log** to record info about pending update
- If crash occurs during update, just replay what is in log to repair

Journaling File System

What is goal of journaling?

- Crash consistency

How achieved?

- Use **write-ahead log** to record info about pending update
- If crash occurs during update, just replay what is in log to repair

Turns multiple writes into single **atomic action**

Example: File Append

Example: File Append

What does file append do?

- Allocates new data block
- Fills data block with user data from write()
- Adds pointer to data block in metadata structure of file system (called an inode)

Example: File Append

What does file append do?

- Allocates new data block
- Fills data block with user data from write()
- Adds pointer to data block in metadata structure of file system (called an inode)

What on-disk structures are modified?

- **Bitmap** (to allocate block)
- **Inode** (to point to new block)
- **Data block** (to hold user data)

Example: File Append

What does file append do?

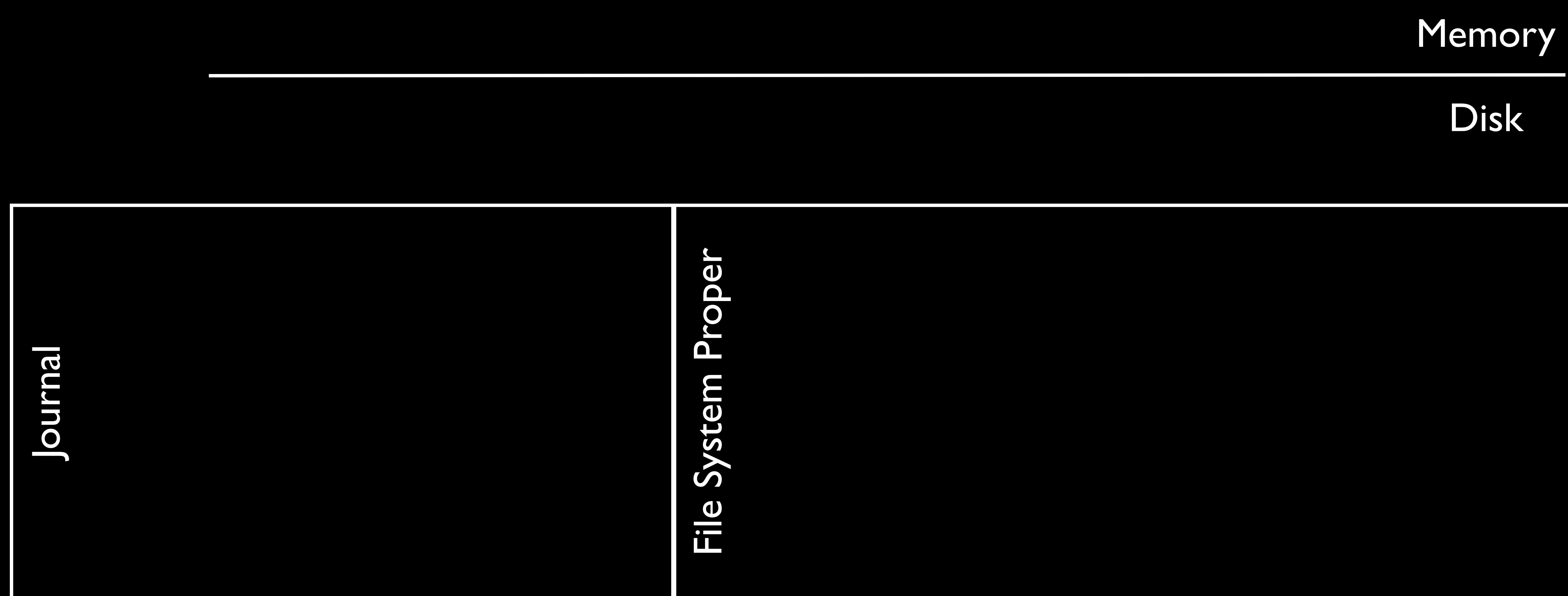
- Allocates new data block
- Fills data block with user data from write()
- Adds pointer to data block in metadata structure of file system (called an inode)

What on-disk structures are modified?

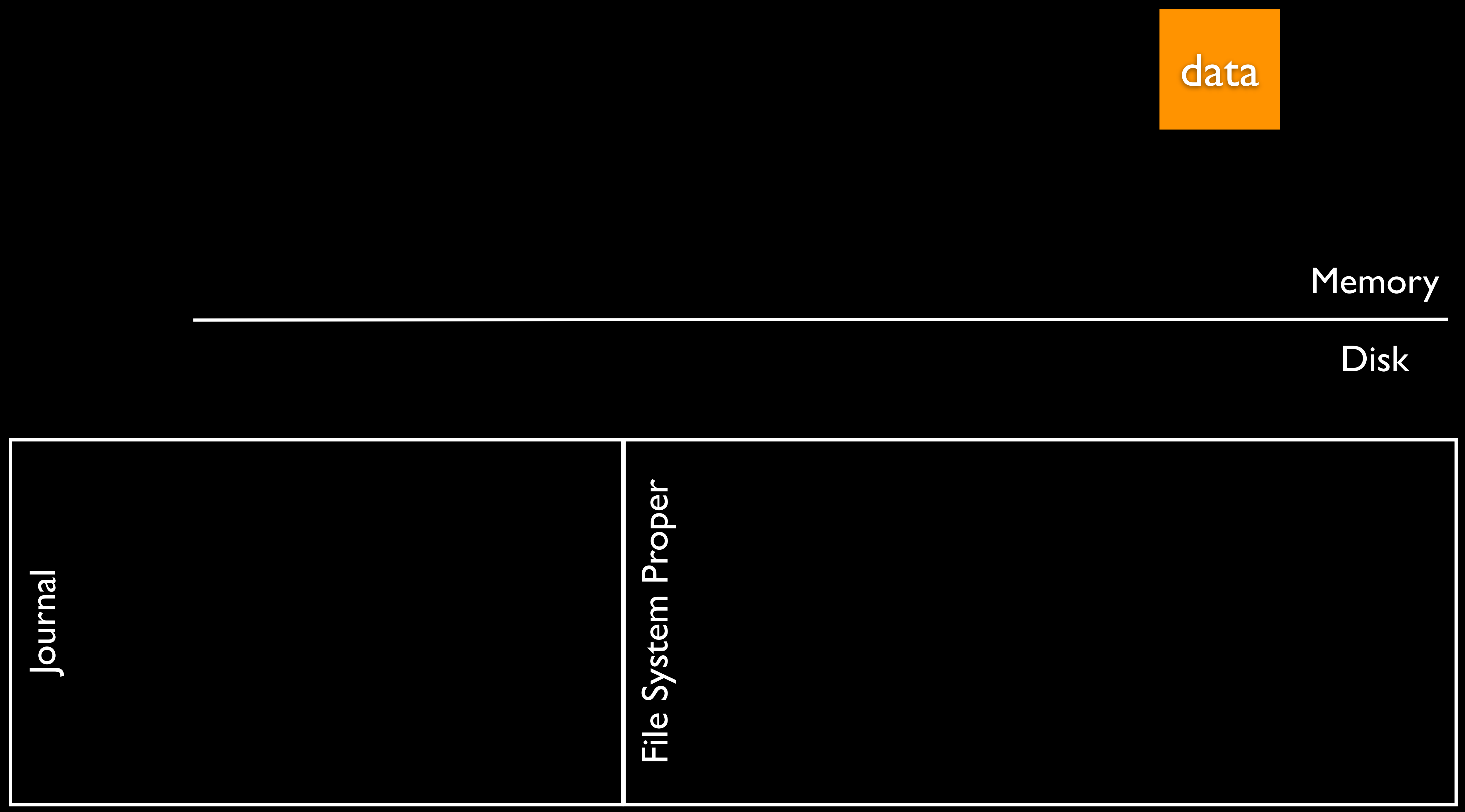
- **Bitmap** (to allocate block)
- **Inode** (to point to new block)
- **Data block** (to hold user data)

Must all be done **atomically**

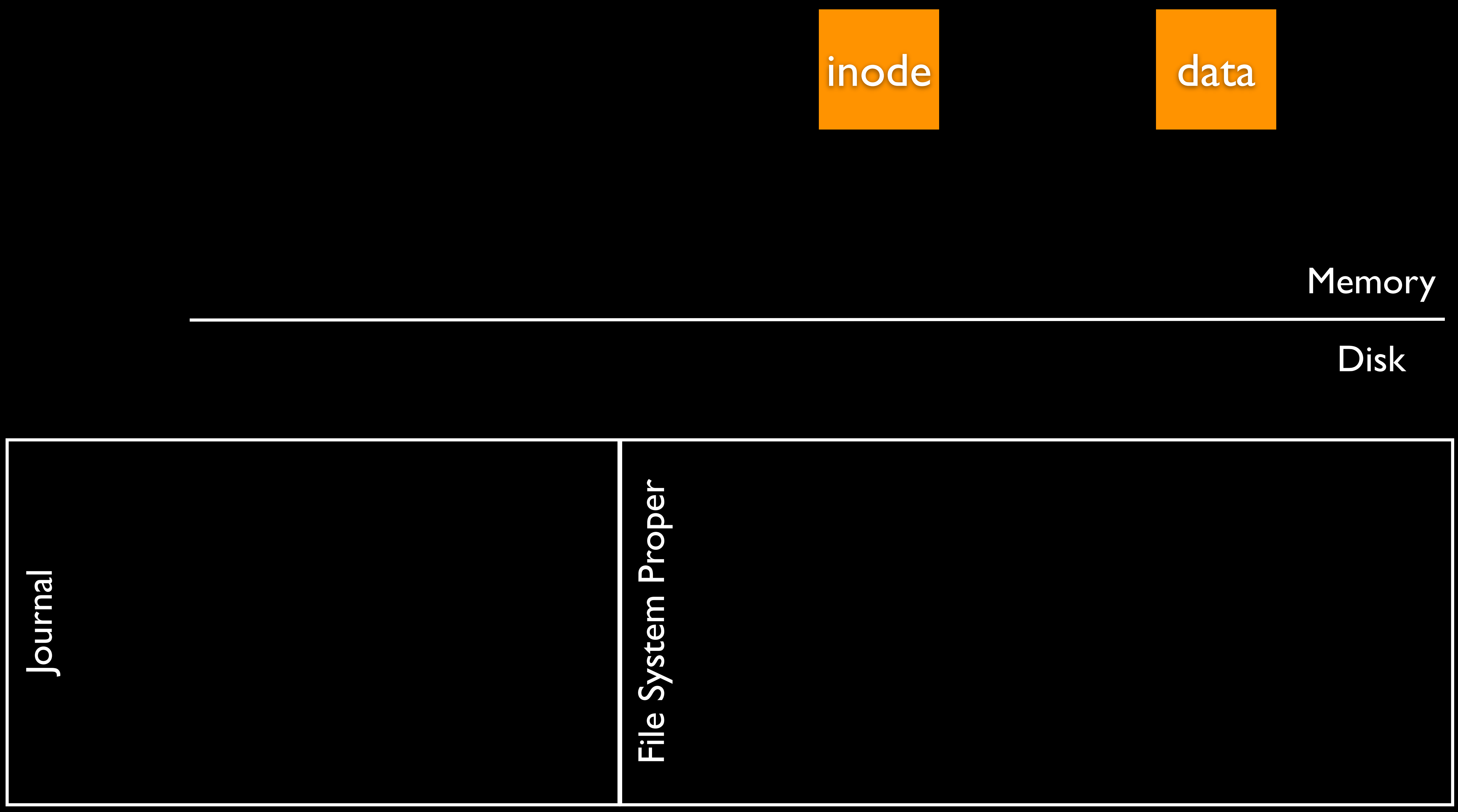
ext3 Ordered Journaling



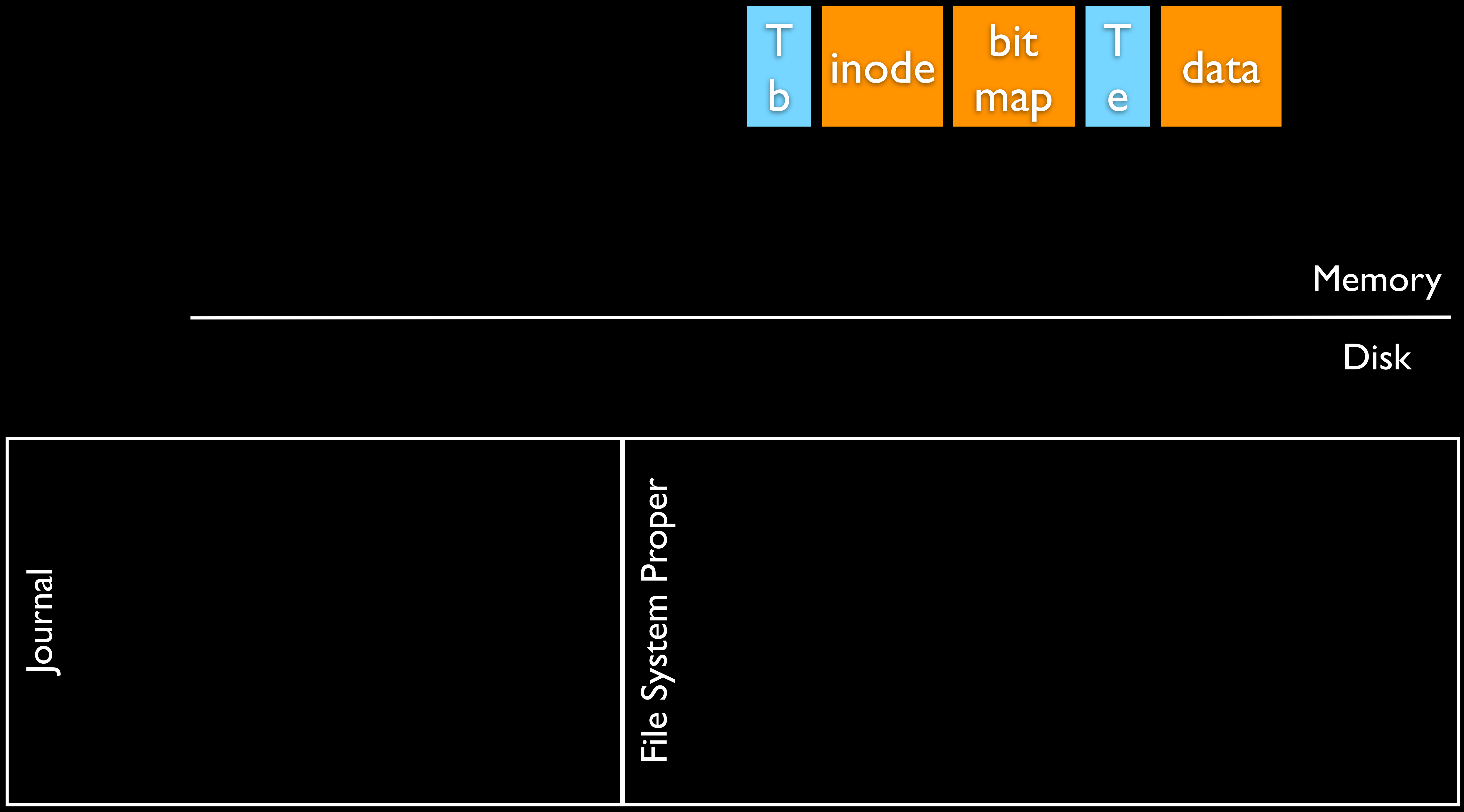
ext3 Ordered Journaling



ext3 Ordered Journaling



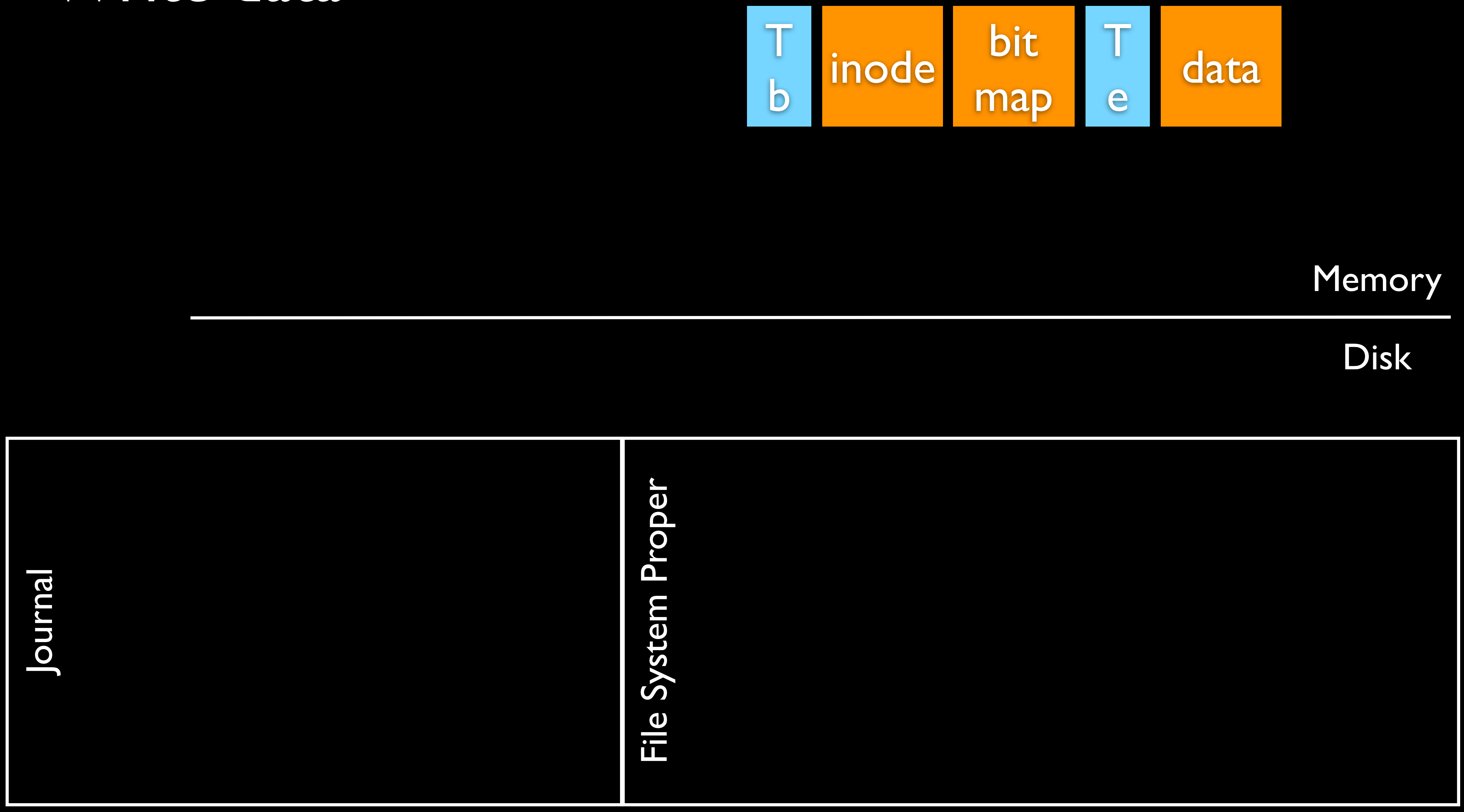
ext3 Ordered Journaling



ext3 Ordered Journaling

Protocol

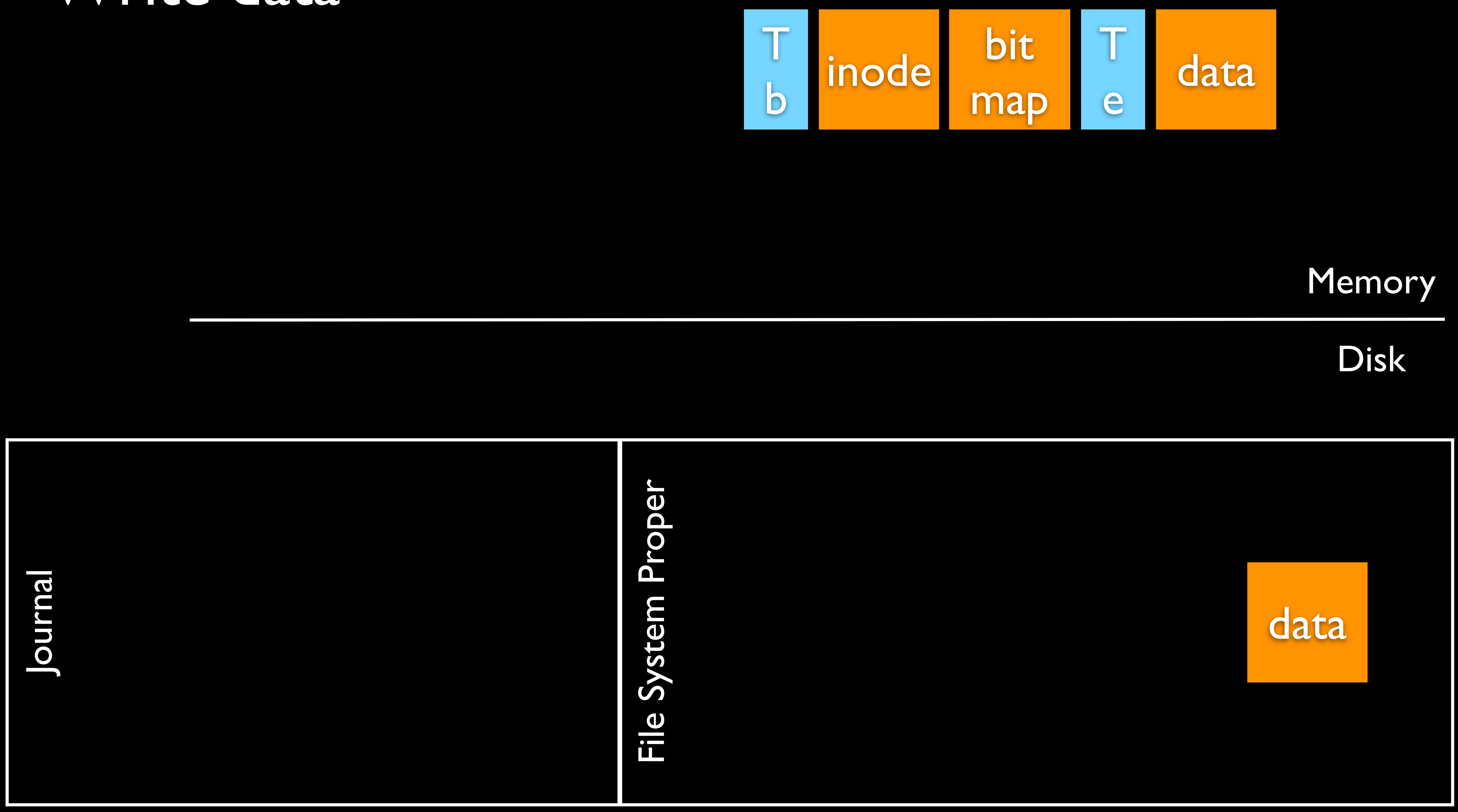
- Write data



ext3 Ordered Journaling

Protocol

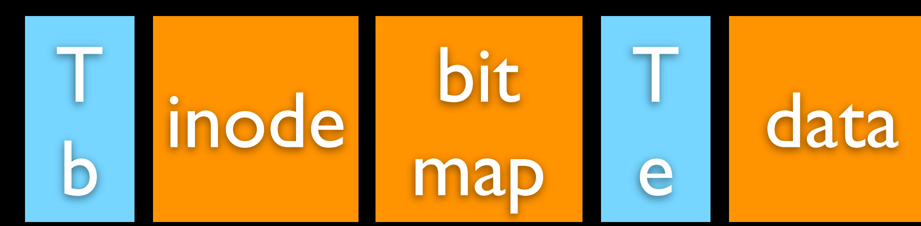
- Write data

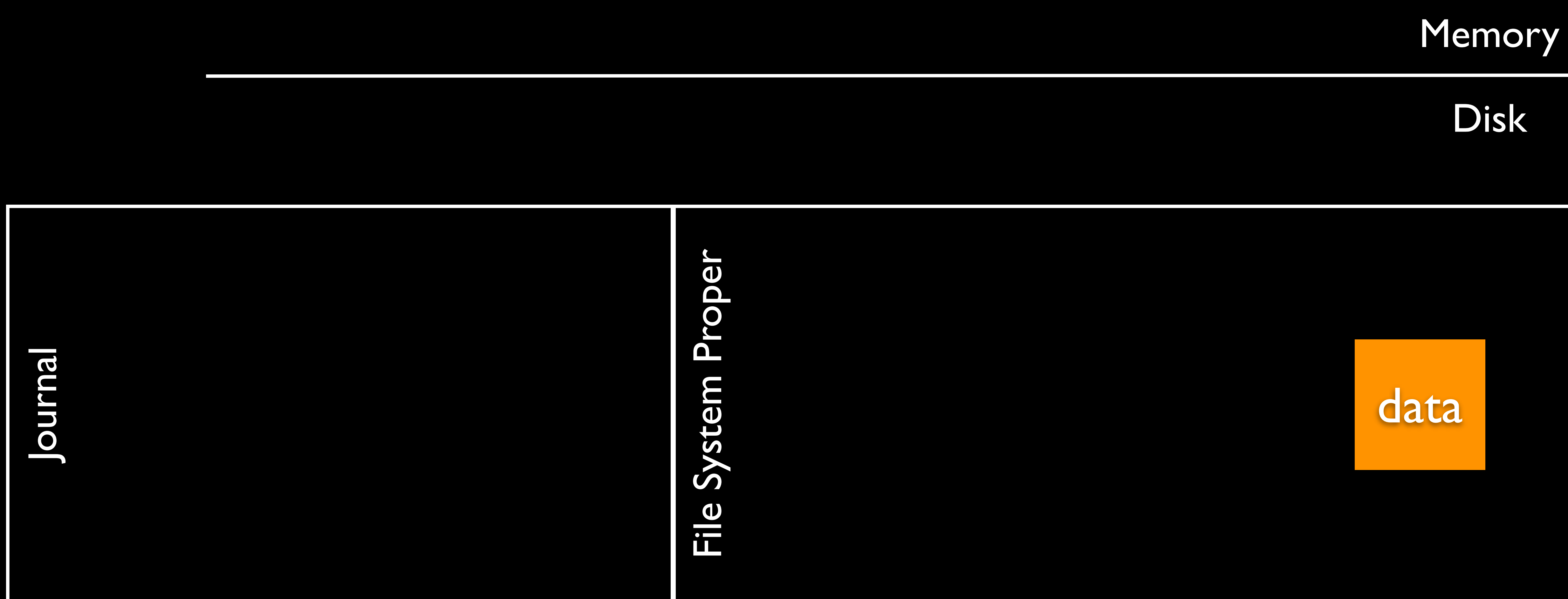


ext3 Ordered Journaling

Protocol

- Write data
- Write TxBegin+contents

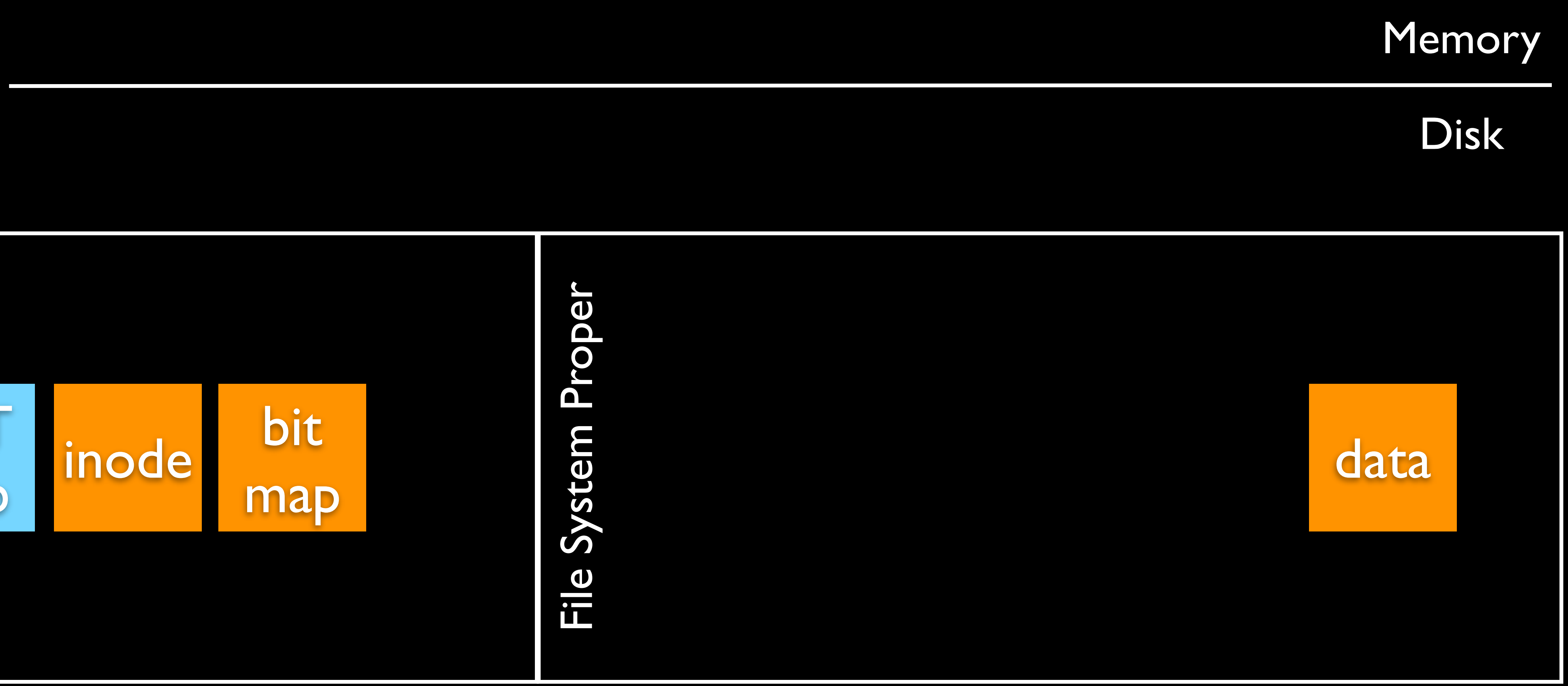




ext3 Ordered Journaling

Protocol

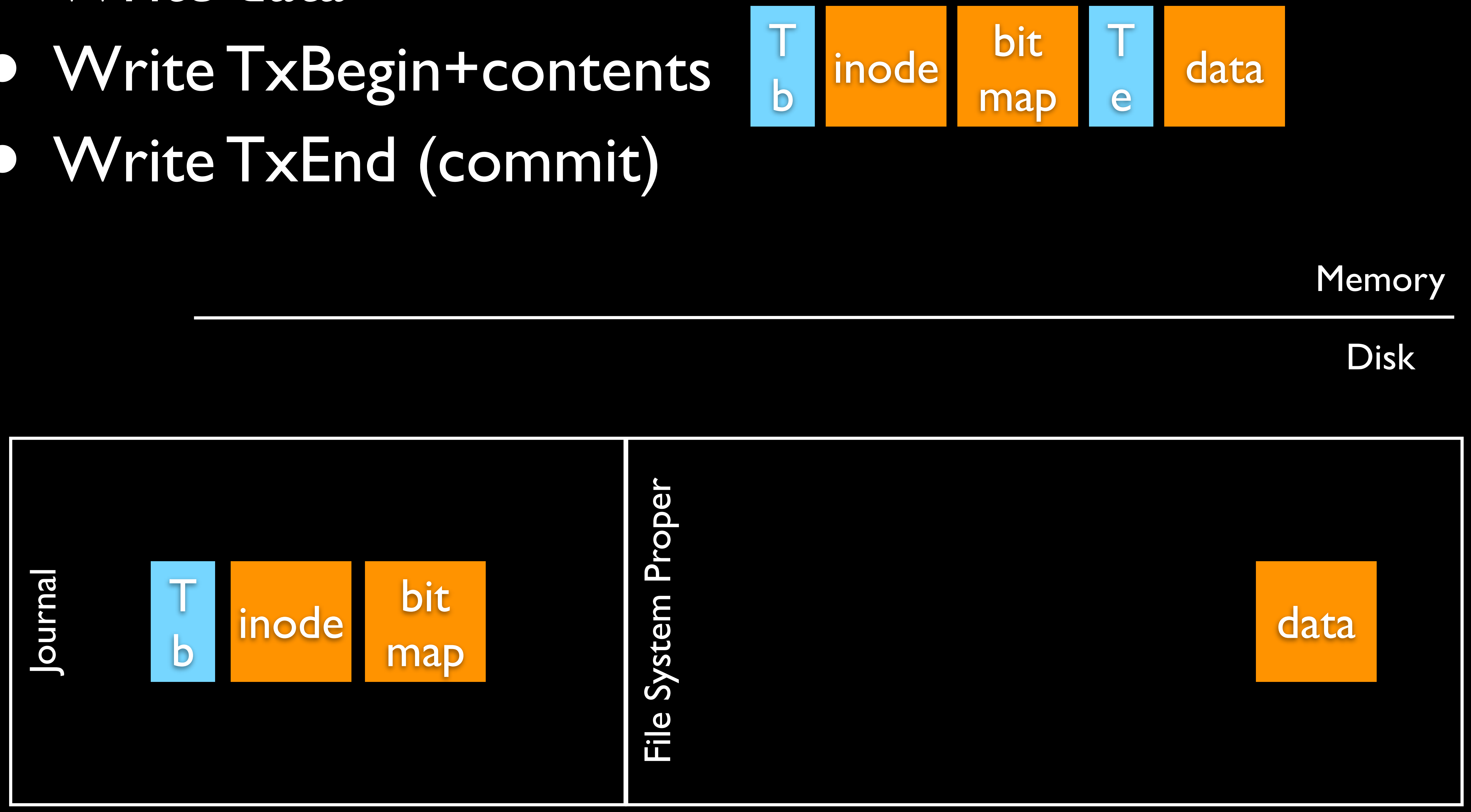
- Write data
- Write TxBegin+contents



ext3 Ordered Journaling

Protocol

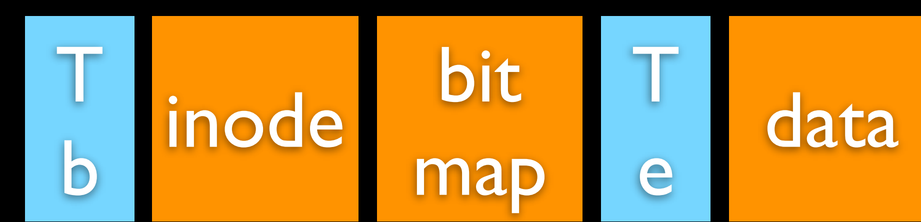
- Write data
- Write TxBegin+contents
- Write TxEnd (commit)



ext3 Ordered Journaling

Protocol

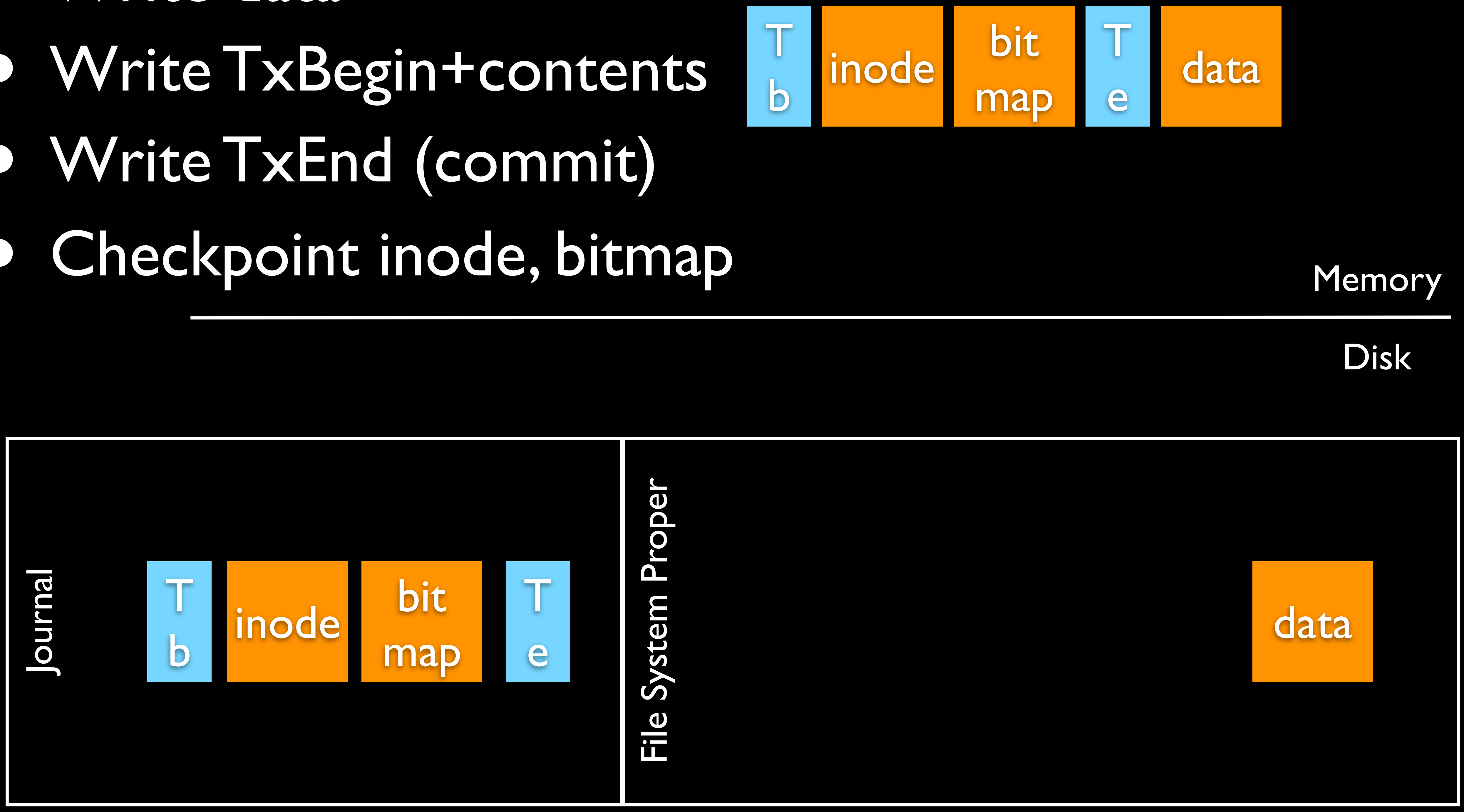
- Write data
- Write TxBegin+contents
- Write TxEnd (commit)



ext3 Ordered Journaling

Protocol

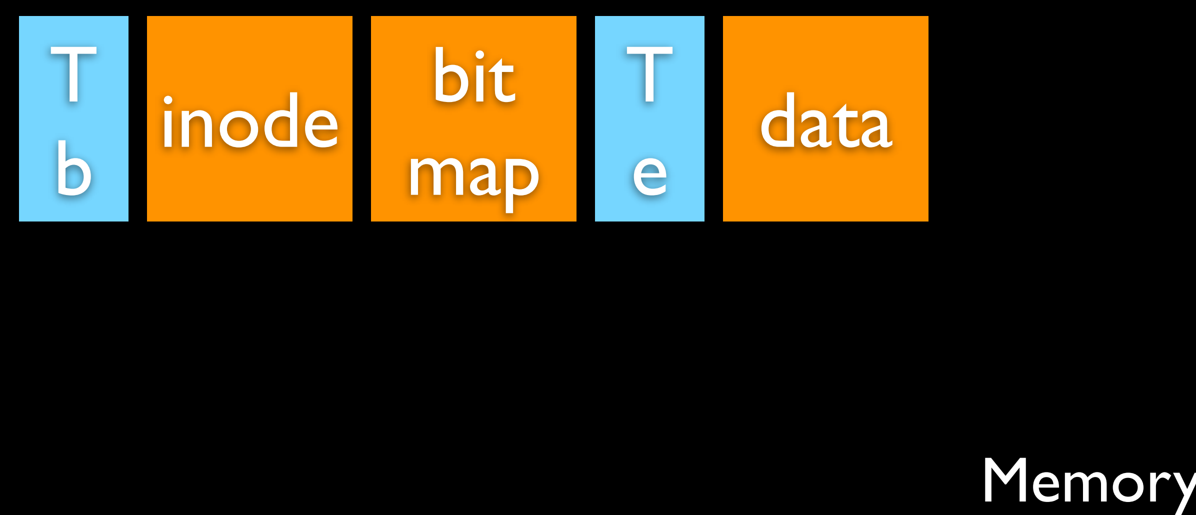
- Write data
- Write TxBegin+contents
- Write TxEnd (commit)
- Checkpoint inode, bitmap

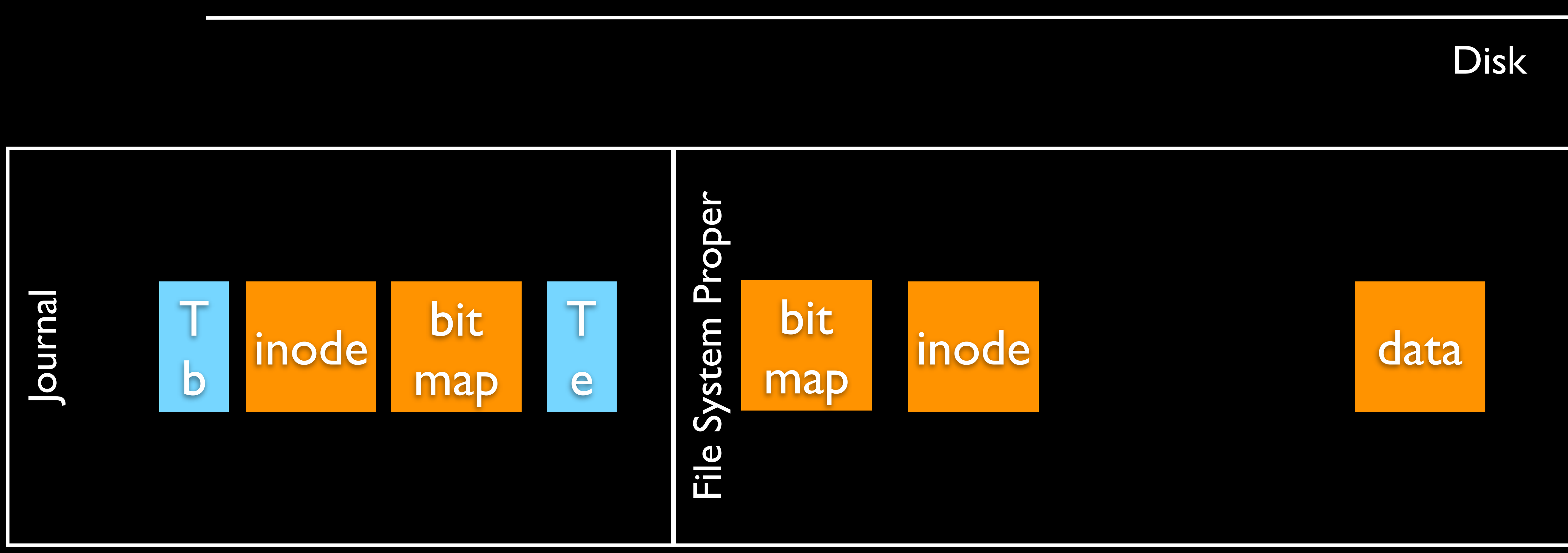


ext3 Ordered Journaling

Protocol

- Write data
- Write TxBegin+contents
- Write TxEnd (commit)
- Checkpoint inode, bitmap





Does This Work?
Is It Correct?

Assumptions About The “Perfect” Disk

Strong Durability

Strong Durability

On-disk (init)

- @ address=A, data=d1

Strong Durability

On-disk (init)

- @ address=A, data=d1

Action

- write(address=A, data=d1')

Strong Durability

On-disk (init)

- @ address=A, data=d1

Action

- write(address=A, data=d1')

On-disk (fini)

- @ address=A, data=d1'

Strong Durability

On-disk (init)

- @ address=A, data=d1

Action

- write(address=A, data=d1')

On-disk (fini)

- @ address=A, data=d1'

The **read after write** property

Strong Durability

On-disk (init)

- @A=d1

Action

- write(@A, d1')

On-disk (fini)

- @A=d1'

The **read after write** property

Ordering

Ordering

On-disk (init)

- @A=d1 and @B=d2

Ordering

On-disk (init)

- @A=d1 and @B=d2

Action

- write(@A, d1’)
- write(@B, d2’)

Ordering

On-disk (init)

- @A=d1 and @B=d2

Action

- write(@A, d1')
- write(@B, d2')

On-disk (after first write)

- @A=d1' and @B=d2

Ordering

On-disk (init)

- @A=d1 and @B=d2

Action

- write(@A, d1’)
- write(@B, d2’)

On-disk (after first write)

- @A=d1’ and @B=d2

On-disk (after second)

- @A=d1’ and @B=d2’

Single-Sector Atomicity

Single-Sector Atomicity

On-disk (init)

- @A=d1

Single-Sector Atomicity

On-disk (init)

- @A=d1

Single-Sector Atomicity

On-disk (init)

- @A=d1

Action

- write(@A, d1')

Single-Sector Atomicity

On-disk (init)

- @A=d1

Action

- write(@A, d1')

On-disk (fini)

- If write size == single sector (512 bytes):
  @A=d1'
- If write size >= single sector:
  @A=(mix of d1 and d1')
  (usual case without power loss: @A=d1')

But Are Disks Perfect?

Unfortunately, no!

Some older problems

- **Latent sector errors (LSEs)**
  - Can't read a certain block
- **Block corruption**
  - Can read a block but get wrong data
- See [Bairavasundaram '07, '08] for details on frequency and other fun facts

And a newer problem...

Modern Disk Caches

Modern Disk Caches

Caches: Critical for performance

- Cache tracks on reads
- Buffer writes before committing to surface

Modern Disk Caches

Caches: Critical for performance

- Cache tracks on reads
- Buffer writes before committing to surface

Example: Why buffering matters

- Write to random blocks
- Vary size of write requests
- Measure average write time

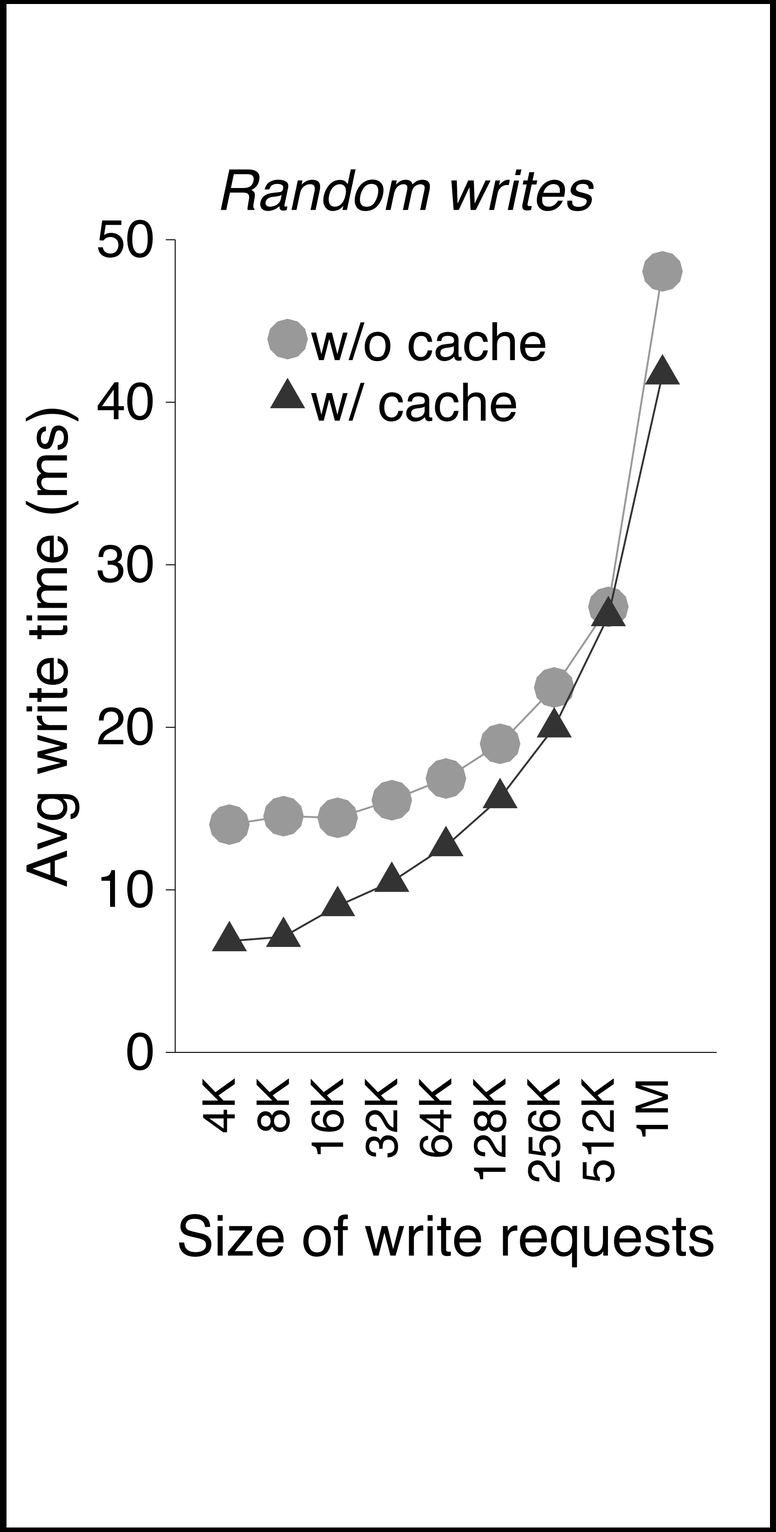
Modern Disk Caches

Caches: Critical for performance

- Cache tracks on reads
- Buffer writes before committing to surface

Example: Why buffering matters

- Write to random blocks
- Vary size of write requests
- Measure average write time



Caching + Writes

Caching + Writes

Caching: Handle with care during writes

- Why? Need careful ordering to implement modern update protocols
- Goal: Crash consistency

Caching + Writes

Caching: Handle with care during writes

- Why? Need careful ordering to implement modern update protocols
- Goal: Crash consistency

Modern approaches require ordering

- Journaling file systems (e.g., ext3)
- Copy-on-write file systems (e.g., ZFS)

Caching + Writes

Caching: Handle with care during writes

- Why? Need careful ordering to implement modern update protocols
- Goal: Crash consistency

Modern approaches require ordering

- Journaling file systems (e.g., ext3)
- Copy-on-write file systems (e.g., ZFS)

Trust cache flush to enforce ordering

- Write(A), Flush, Write(B); "guarantees" A reaches disk before B

Should We Trust Flush?

Should We Trust Flush?

Experts say “it depends”:

Should We Trust Flush?

Experts say “it depends”:

- Some drives ignore barriers/flush

Should We Trust Flush?

Experts say “it depends”:

- Some drives ignore barriers/flush

Documentation hints at problem:

Should We Trust Flush?

Experts say "it depends":

- Some drives ignore barriers/flush

Documentation hints at problem:

- From the fcntl man page on Mac OS X:

Should We Trust Flush?

Experts say "it depends":

- Some drives ignore barriers/flush

Documentation hints at problem:

- From the fcntl man page on Mac OS X:
  - F_FULLSYNC: Does the same thing as fsync(2) then asks the drive to flush all buffered data to device. Certain FireWire drives have also been known to ignore the request to flush their buffered data.

Should We Trust Flush?

Experts say “it depends”:

- Some drives ignore barriers/flush

Documentation hints at problem:

- From the fcntl man page on Mac OS X:
  - F_FULLSYNC: Does the same thing as fsync(2) then asks the drive to flush all buffered data to device. Certain FireWire drives have also been known to ignore the request to flush their buffered data.
- From VirtualBox documentation:

Should We Trust Flush?

Experts say "it depends":

- Some drives ignore barriers/flush

Documentation hints at problem:

- From the fcntl man page on Mac OS X:
  - F_FULLSYNC: Does the same thing as fsync(2) then asks the drive to flush all buffered data to device. Certain FireWire drives have also been known to ignore the request to flush their buffered data.
- From VirtualBox documentation:
  - If desired, the virtual disk images can be flushed when the guest issues the IDE FLUSH CACHE command. Normally these requests are ignored for improved performance.

Weak Durability

Weak Durability

On-disk (init)

Weak Durability

On-disk (init)

- @A=d1

Weak Durability

On-disk (init)

- @A=d1

Action

Weak Durability

On-disk (init)

- @A=d1

Action

- write(@A, d1’) at time=T

Weak Durability

On-disk (init)

- @A=d1

Action

- write(@A, d1’) at time=T

In-cache (fini)

Weak Durability

On-disk (init)

- @A=d1

Action

- write(@A, d1') at time=T

In-cache (fini)

- @A=d1' (delay write by **delta** time units)

Weak Durability

On-disk (init)

- @A=d1

Action

- write(@A, d1') at time=T

In-cache (fini)

- @A=d1' (delay write by **delta** time units)

On-disk (fini)

Weak Durability

On-disk (init)

- @A=d1

Action

- write(@A, d1’) at time=T

In-cache (fini)

- @A=d1’ (delay write by **delta** time units)

On-disk (fini)

- @A=d1 (time < T+delta)

Weak Durability

On-disk (init)

- @A=d1

Action

- write(@A, d1’) at time=T

In-cache (fini)

- @A=d1’ (delay write by **delta** time units)

On-disk (fini)

- @A=d1 (time < T+delta)
- @A=d1’ (time > T+delta && no power loss) or

Weak Durability

On-disk (init)

- @A=d1

Action

- write(@A, d1') at time=T

In-cache (fini)

- @A=d1' (delay write by **delta** time units)

On-disk (fini)

- @A=d1 (time < T+delta)
- @A=d1' (time > T+delta && no power loss) or
  @A=d1 (power loss at time < T+delta)

Dealing with Weak Durability

Outline

Method #1: Coerced Cache Eviction

Method #2: No-order File System

Final Thoughts

Coerced Cache Eviction

How to Reduce Trust on Disk Ordering?

Idea: Coercion

Idea: Coercion

Desire: Enforce ordering of Write(A), Write(B)

Idea: Coercion

Desire: Enforce ordering of Write(A), Write(B)

Method:

- Write (A)
- Write a bunch of other stuff (evicting A from cache in process)
- Write (B)

Idea: Coercion

Desire: Enforce ordering of Write(A), Write(B)

Method:

- Write (A)
- Write a bunch of other stuff (evicting A from cache in process)
- Write (B)

Called Coerced Cache Eviction (CCE)

- Use CCE to build FS that works despite faulty disk behavior

CCE: Outline

Disk Caching: A Study

Coerced Cache Eviction

Discreet-mode Journaling: Using CCE

Results

Goal: Learn Policy

Goal: Learn Policy

To build CCE, must know eviction policy

Goal: Learn Policy

To build CCE, must know eviction policy

- But, not published or well-known

Goal: Learn Policy

To build CCE, must know eviction policy

- But, not published or well-known

Idea: Use **microbenchmark** to discover

Goal: Learn Policy

To build CCE, must know eviction policy

- But, not published or well-known

Idea: Use **microbenchmark** to discover

- Write target eviction block T to disk

Goal: Learn Policy

To build CCE, must know eviction policy

- But, not published or well-known

Idea: Use **microbenchmark** to discover

- Write target eviction block T to disk
- Perform series of writes, varying number, data amount, sequential/random

Goal: Learn Policy

To build CCE, must know eviction policy

- But, not published or well-known

Idea: Use **microbenchmark** to discover

- Write target eviction block T to disk
- Perform series of writes, varying number, data amount, sequential/random
- Read back T and measure latency of read

Goal: Learn Policy

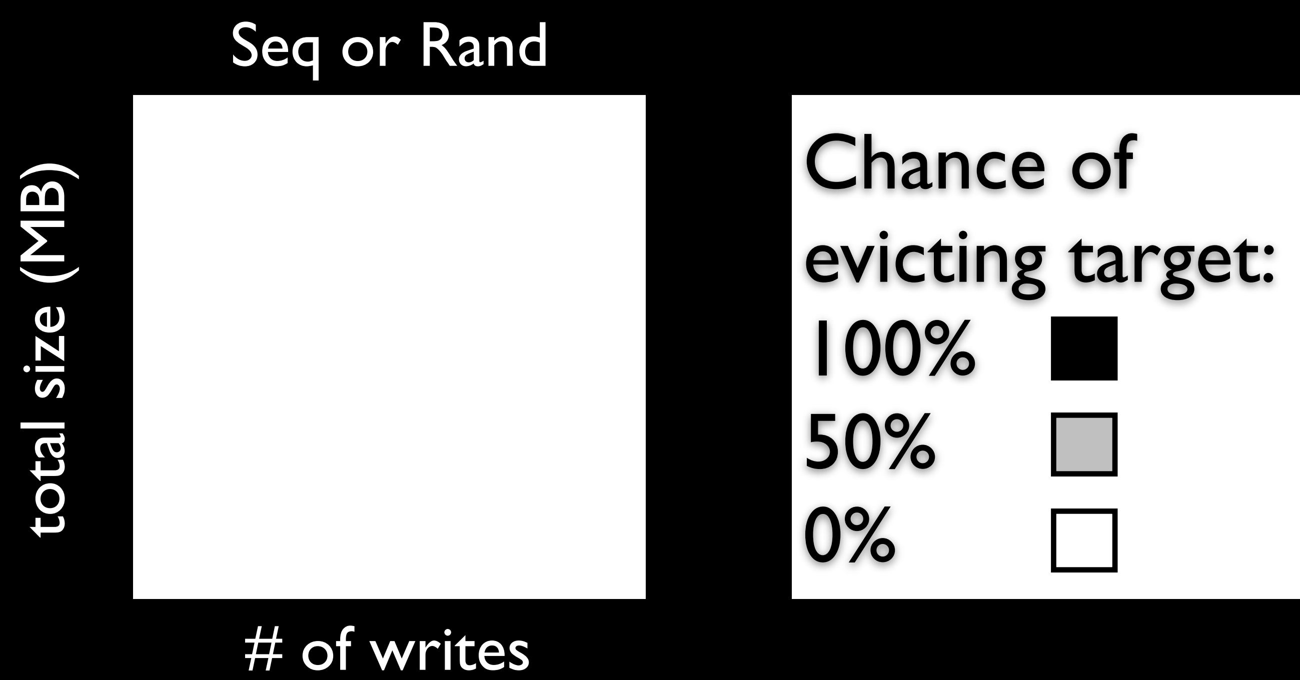
To build CCE, must know eviction policy

- But, not published or well-known

Idea: Use **microbenchmark** to discover

- Write target eviction block T to disk
- Perform series of writes, varying number, data amount, sequential/random
- Read back T and measure latency of read
  - If read is "slow", T was on disk;
    if read is "fast", T was still in memory

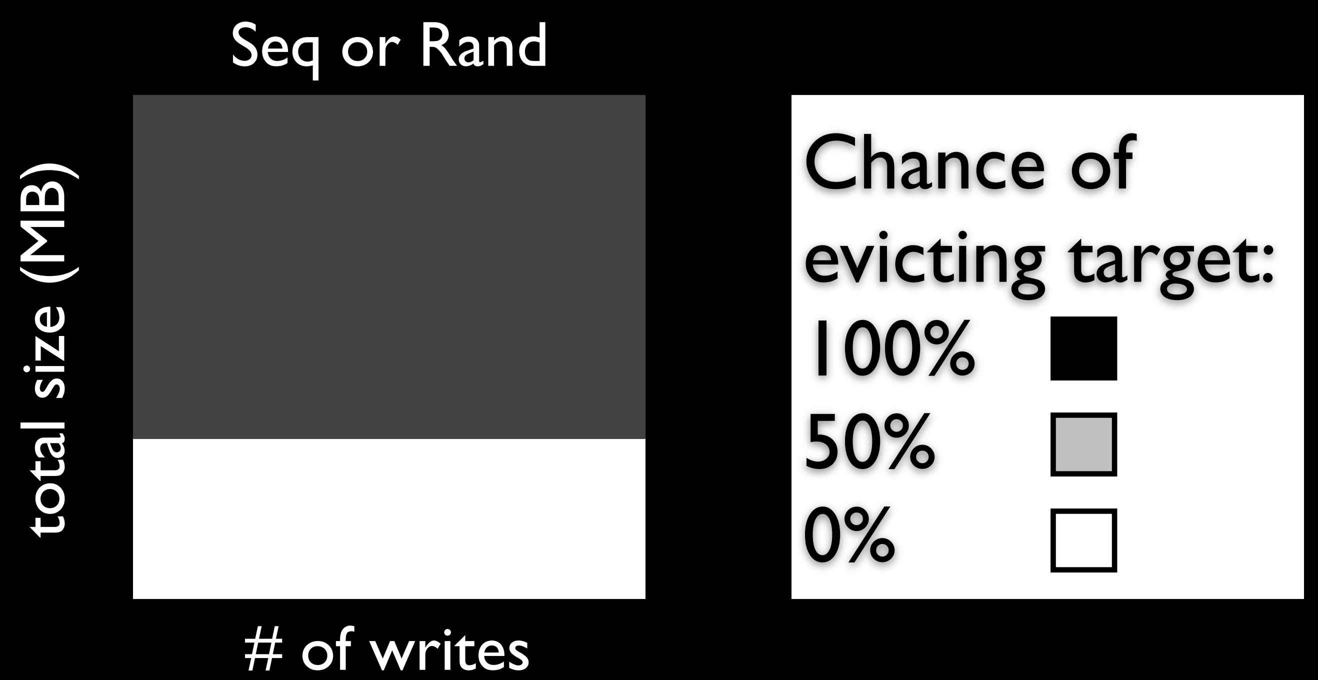
Eviction Graph



Graphs: Sequential and Random patterns

- Vary # of writes (x-axis) + amount (y-axis)
- Observe results to determine effective flush

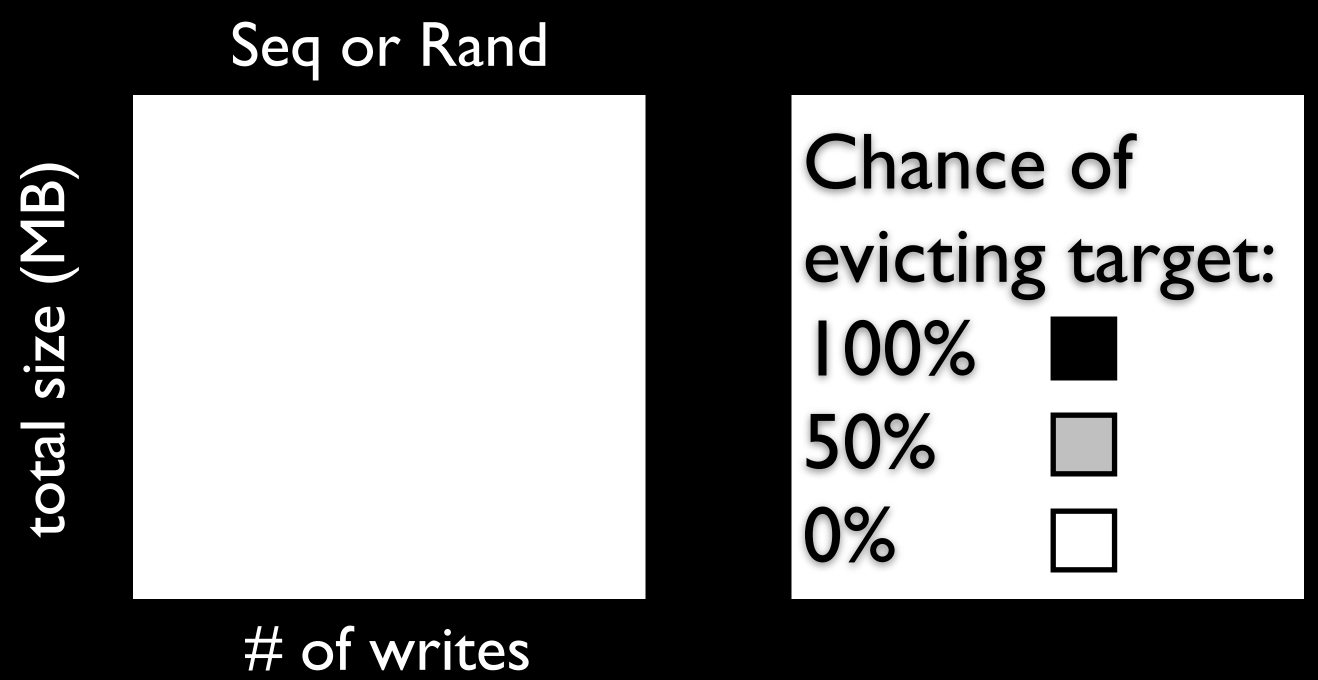
Eviction Graph



Graphs: Sequential and Random patterns

- Vary # of writes (x-axis) + amount (y-axis)
- Observe results to determine effective flush

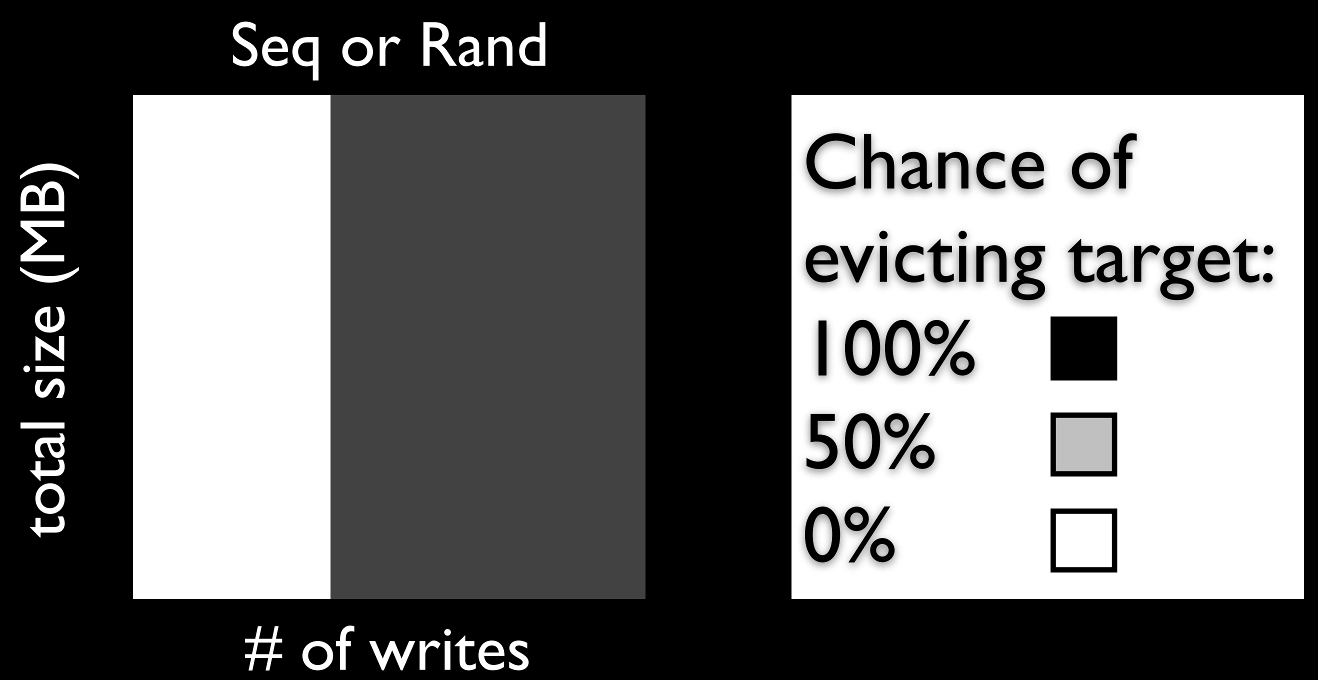
Eviction Graph



Graphs: Sequential and Random patterns

- Vary # of writes (x-axis) + amount (y-axis)
- Observe results to determine effective flush

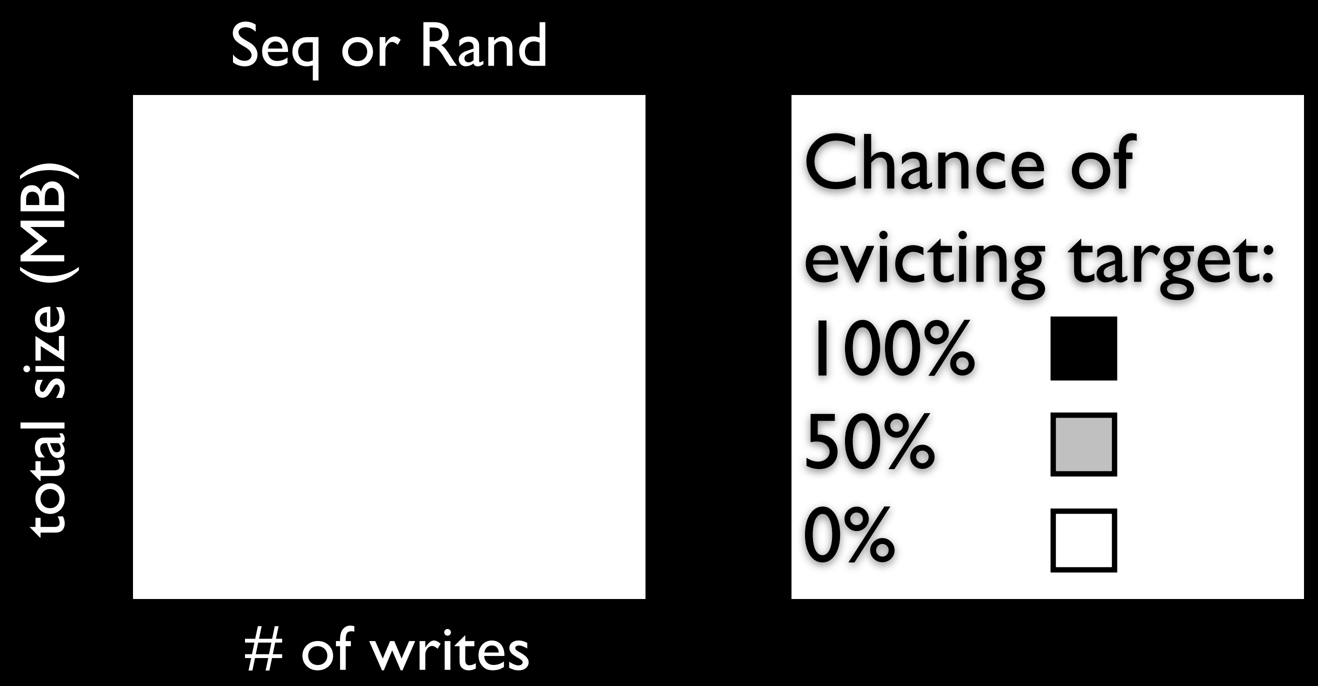
Eviction Graph



Graphs: Sequential and Random patterns

- Vary # of writes (x-axis) + amount (y-axis)
- Observe results to determine effective flush

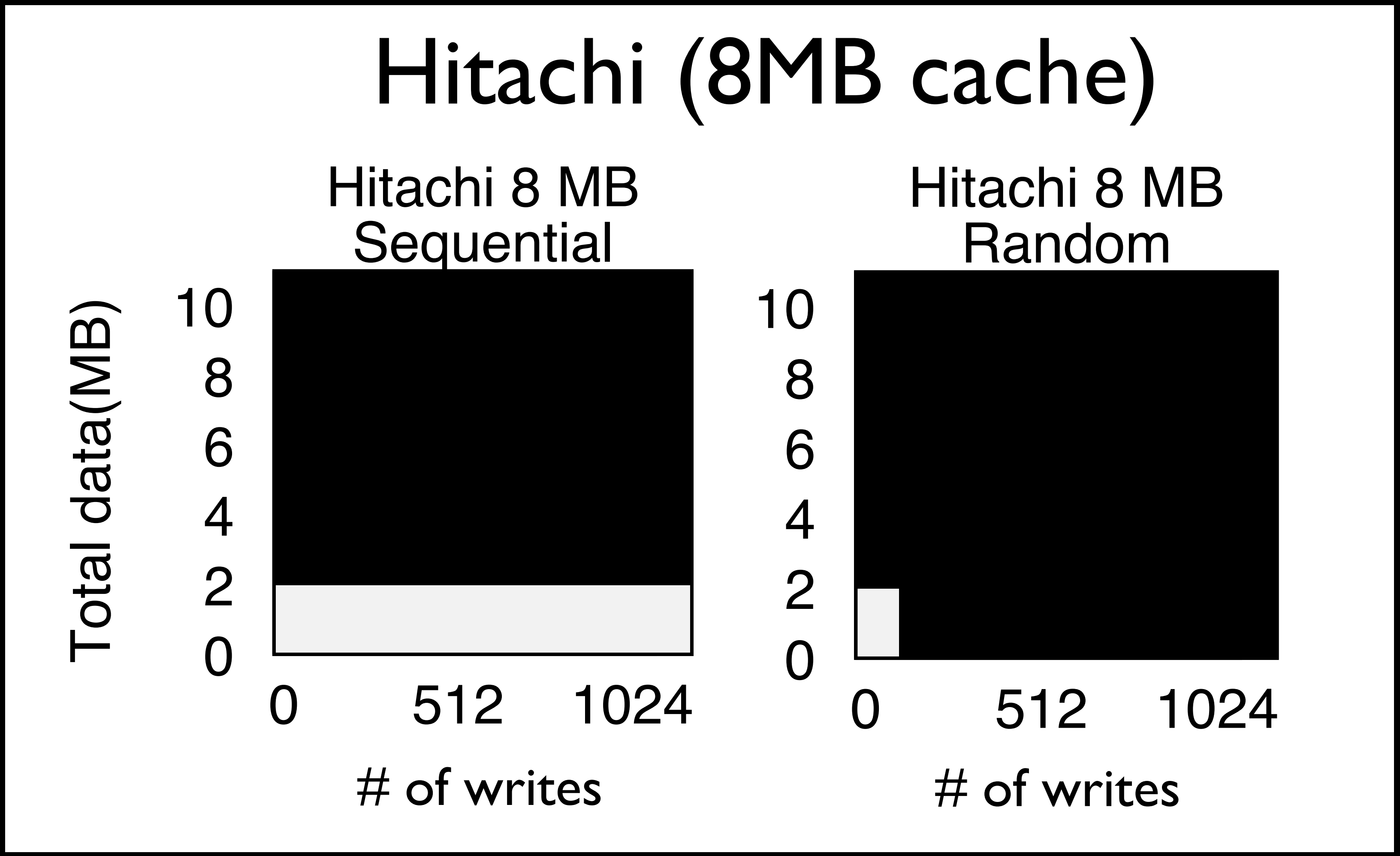
Eviction Graph



Graphs: Sequential and Random patterns

- Vary # of writes (x-axis) + amount (y-axis)
- Observe results to determine effective flush

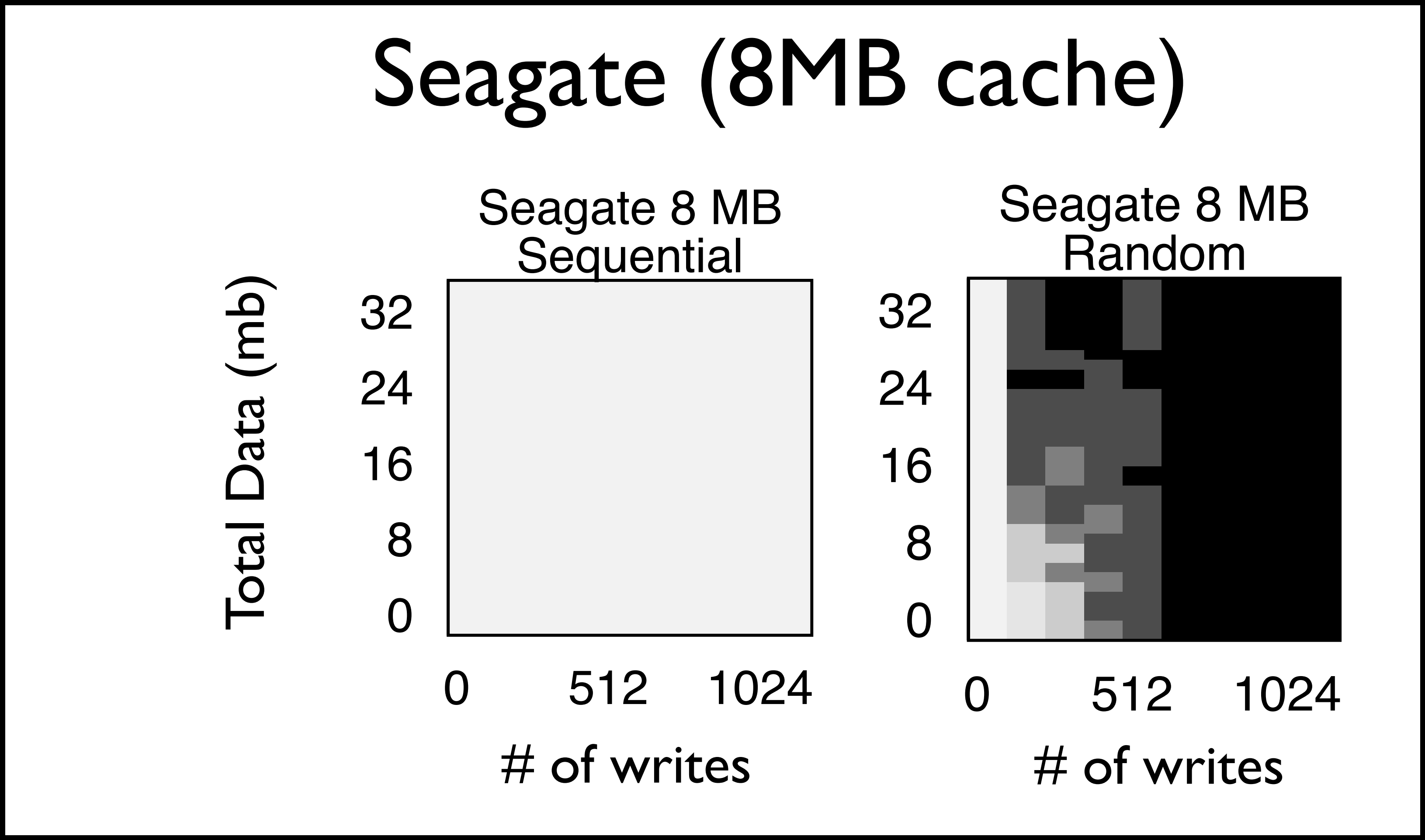
Results



Hitachi flush strategy:

- Over 2MB always flushes cache
- Pick **sequential write of > 2MB** (most efficient choice)

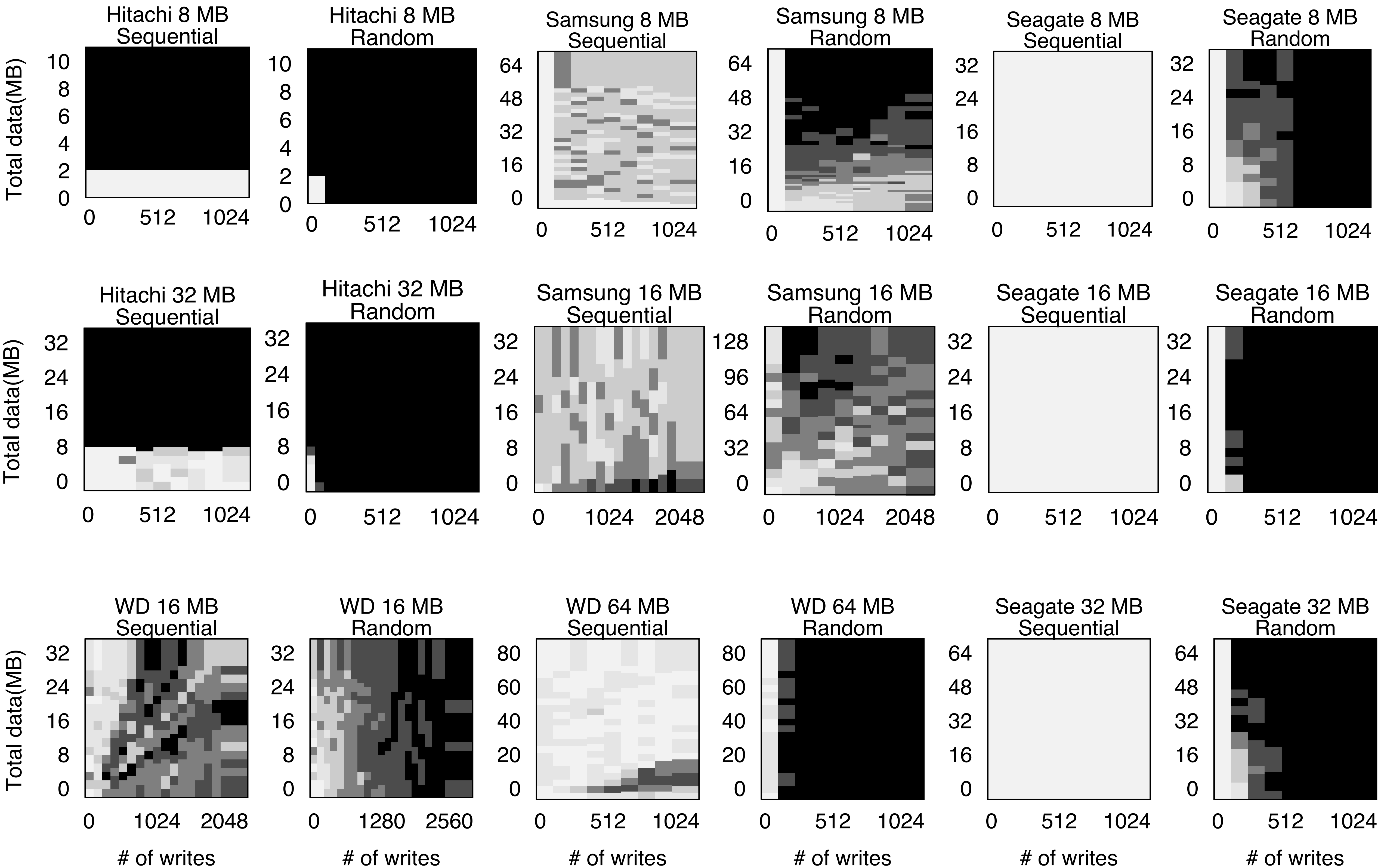
Seagate Disk



Seagate flush strategy:

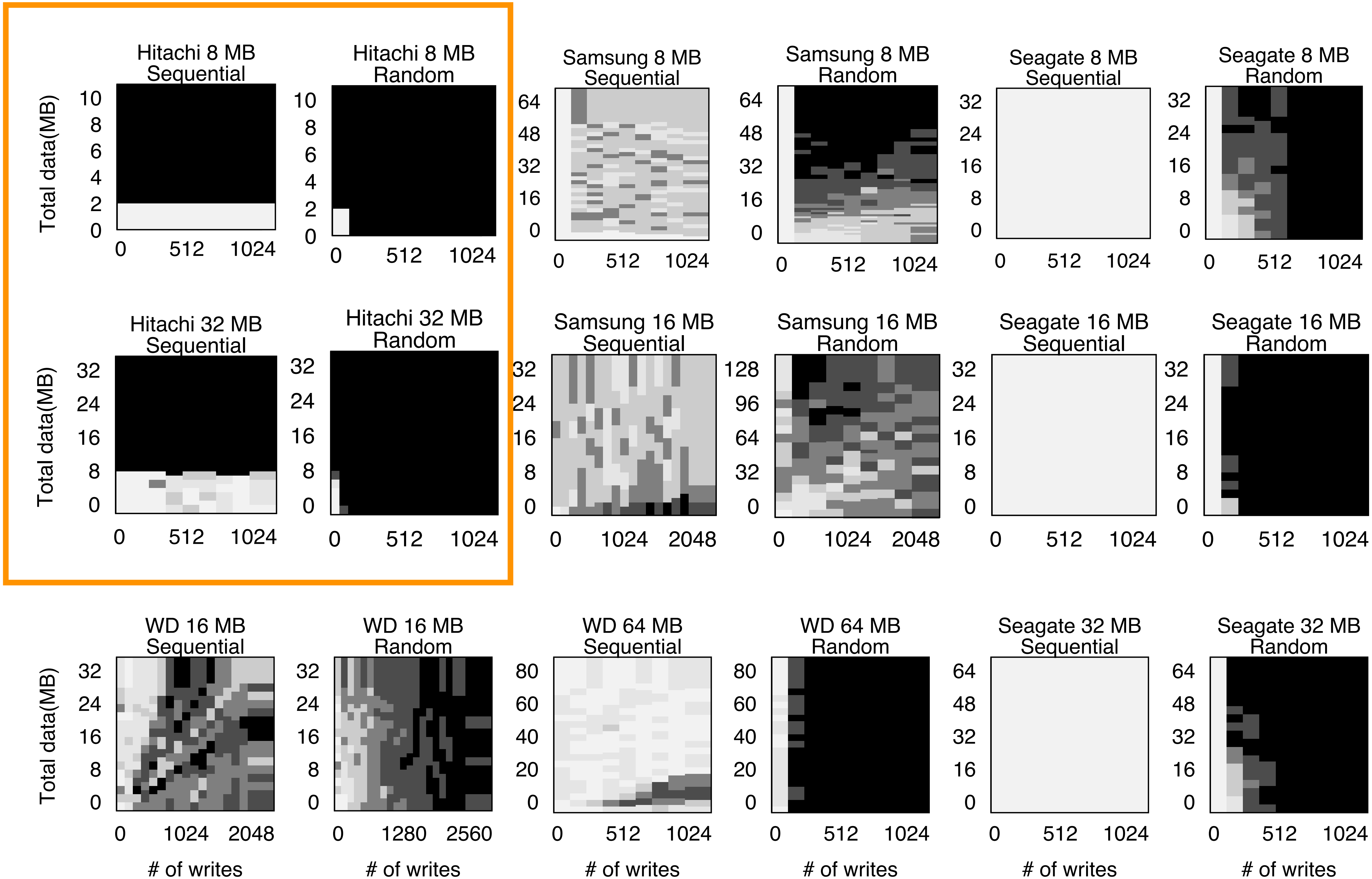
- No amount of sequential writes flush cache
- Random writes do better (but not LRU)

All Results



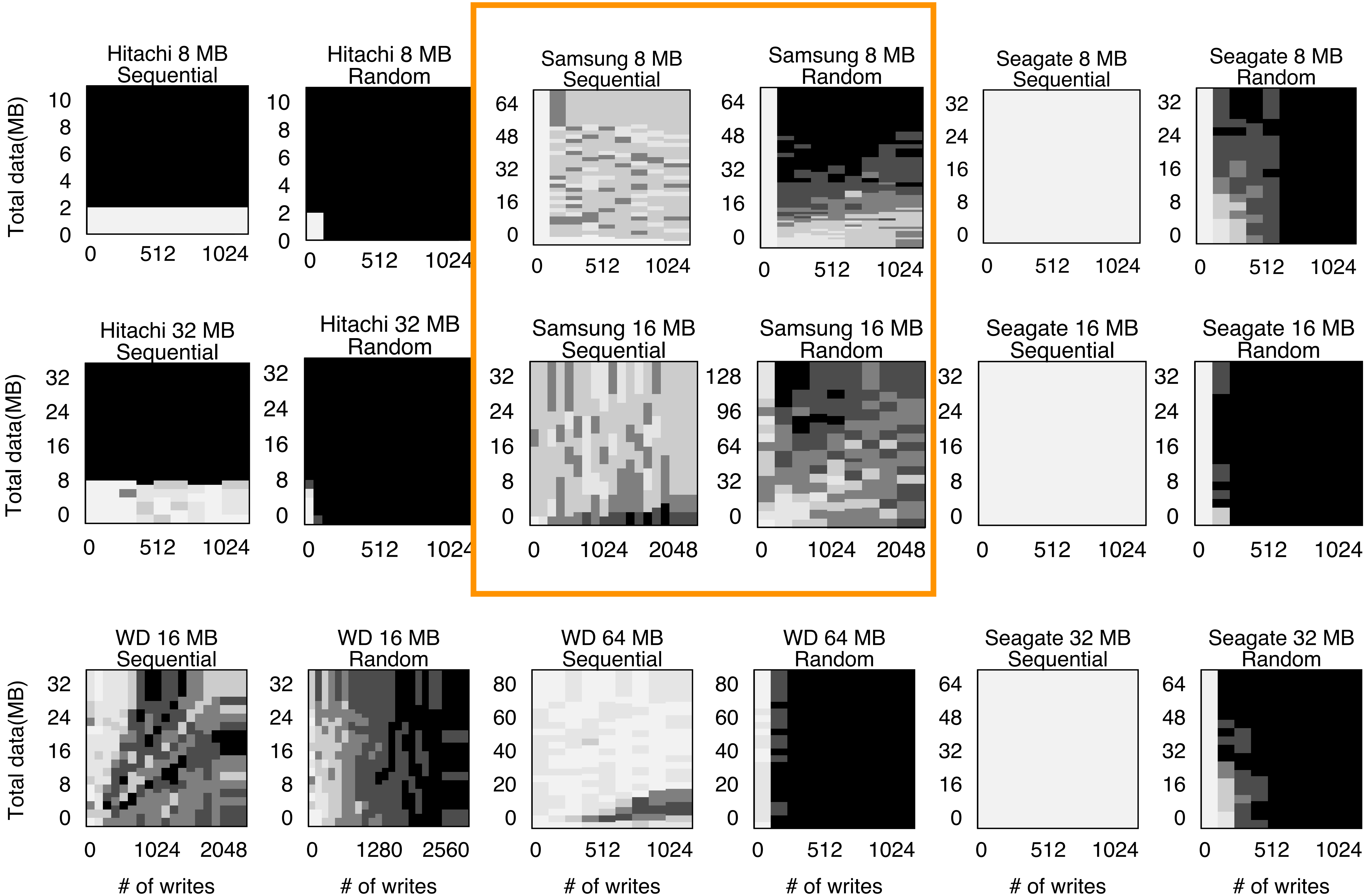
Hitachi 8 MB Sequential
Hitachi 8 MB Random
Samsung 8 MB Sequential
Samsung 8 MB Random
Seagate 8 MB Sequential
Seagate 8 MB Random
Hitachi 32 MB Sequential
Hitachi 32 MB Random
Samsung 16 MB Sequential
Samsung 16 MB Random
Seagate 16 MB Sequential
Seagate 16 MB Random
WD 16 MB Sequential
WD 16 MB Random
WD 64 MB Sequential
WD 64 MB Random
Seagate 32 MB Sequential
Seagate 32 MB Random
Total data(MB)
# of writes
0
512
1024
1280
2048
2560

All Results

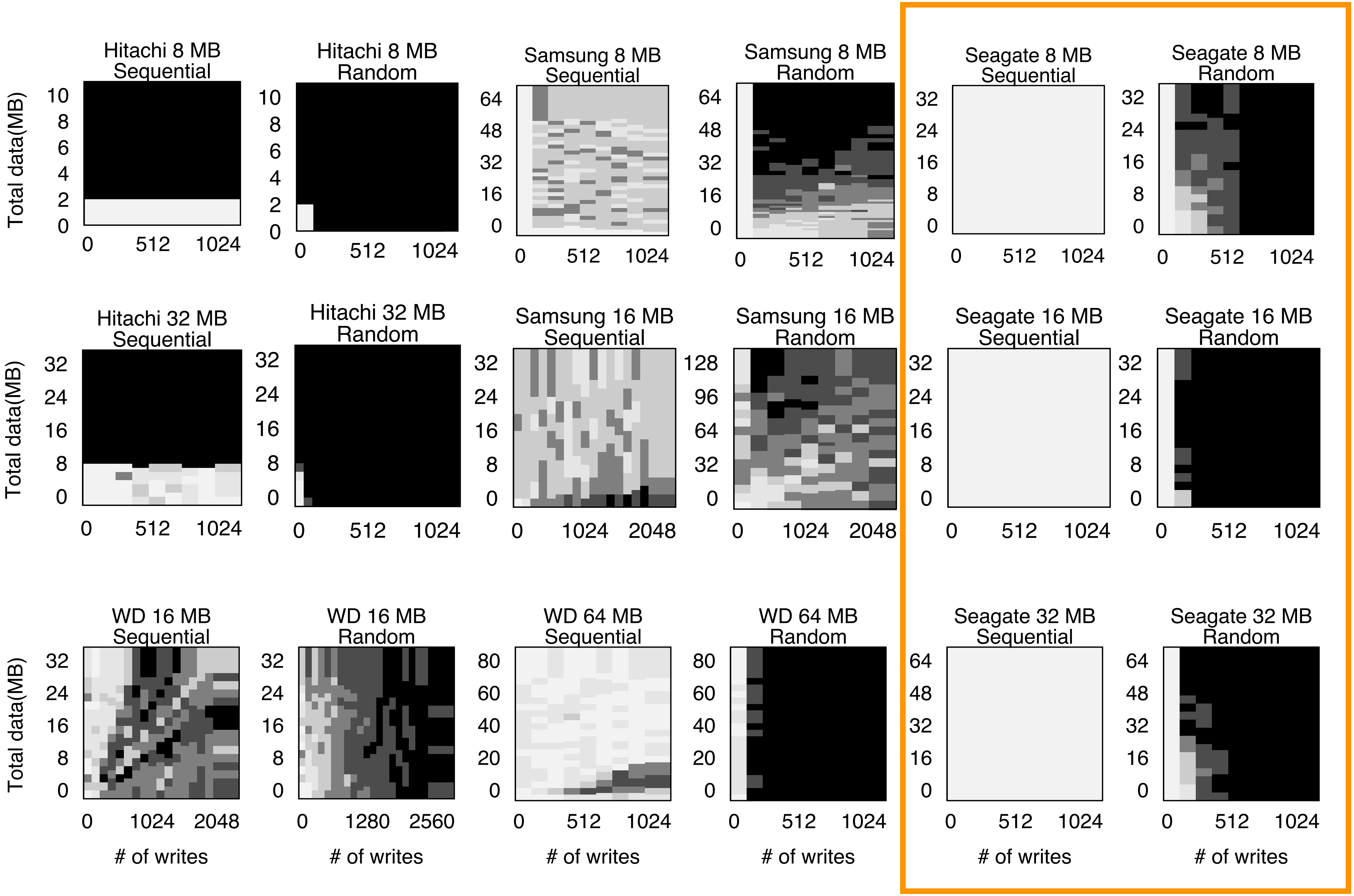


Hitachi 8 MB Sequential
Hitachi 8 MB Random
Samsung 8 MB Sequential
Samsung 8 MB Random
Seagate 8 MB Sequential
Seagate 8 MB Random
Hitachi 32 MB Sequential
Hitachi 32 MB Random
Samsung 16 MB Sequential
Samsung 16 MB Random
Seagate 16 MB Sequential
Seagate 16 MB Random
WD 16 MB Sequential
WD 16 MB Random
WD 64 MB Sequential
WD 64 MB Random
Seagate 32 MB Sequential
Seagate 32 MB Random
Total data(MB)
# of writes

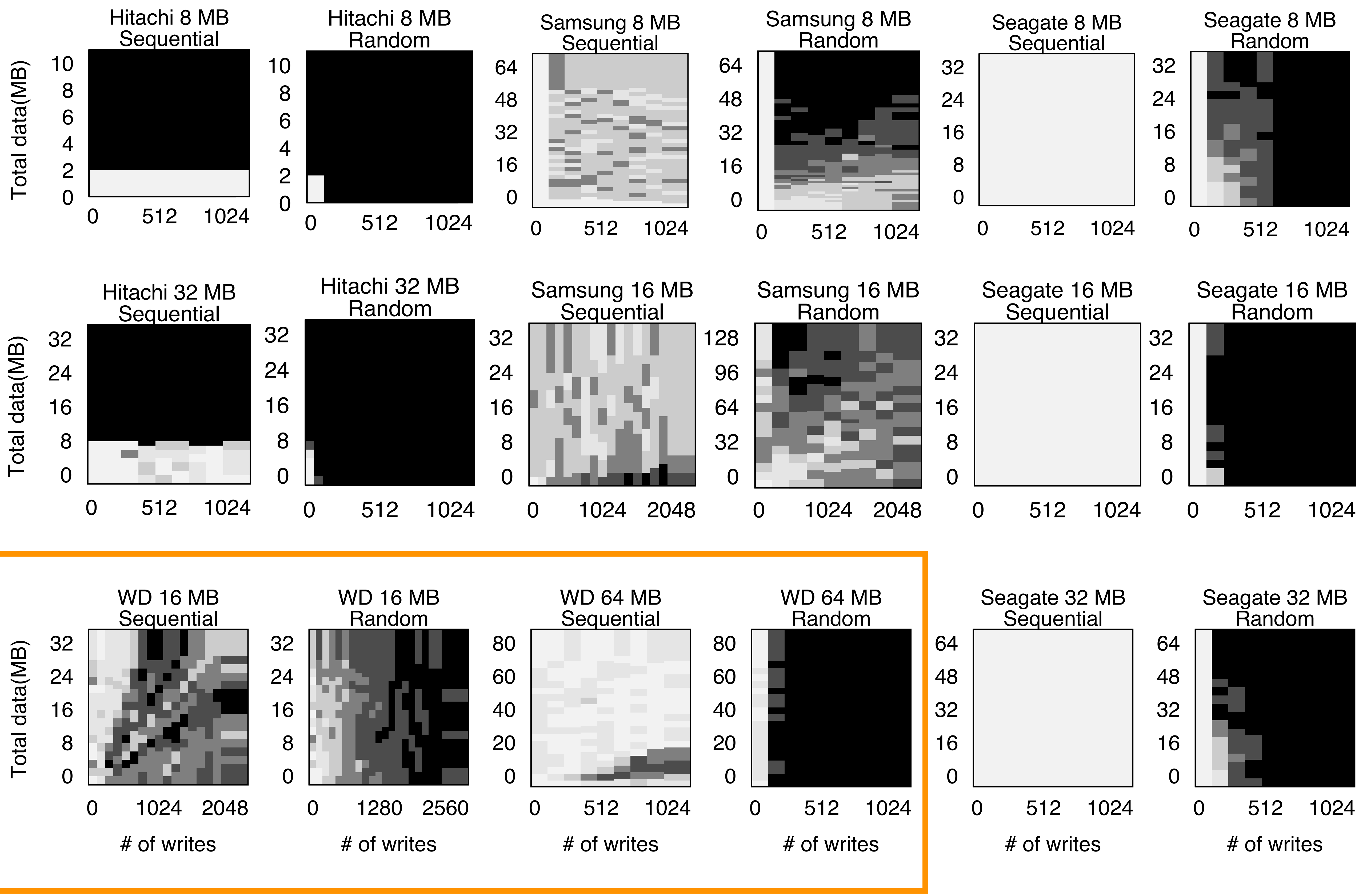
All Results


Hitachi 8 MB Sequential
Hitachi 8 MB Random
Samsung 8 MB Sequential
Samsung 8 MB Random
Seagate 8 MB Sequential
Seagate 8 MB Random
Hitachi 32 MB Sequential
Hitachi 32 MB Random
Samsung 16 MB Sequential
Samsung 16 MB Random
Seagate 16 MB Sequential
Seagate 16 MB Random
WD 16 MB Sequential
WD 16 MB Random
WD 64 MB Sequential
WD 64 MB Random
Seagate 32 MB Sequential
Seagate 32 MB Random
Total data(MB)
# of writes

All Results


Hitachi 8 MB Sequential
Hitachi 8 MB Random
Samsung 8 MB Sequential
Samsung 8 MB Random
Seagate 8 MB Sequential
Seagate 8 MB Random
Hitachi 32 MB Sequential
Hitachi 32 MB Random
Samsung 16 MB Sequential
Samsung 16 MB Random
Seagate 16 MB Sequential
Seagate 16 MB Random
WD 16 MB Sequential
WD 16 MB Random
WD 64 MB Sequential
WD 64 MB Random
Seagate 32 MB Sequential
Seagate 32 MB Random
Total data(MB)
# of writes

All Results


Hitachi 8 MB Sequential
Hitachi 8 MB Random
Samsung 8 MB Sequential
Samsung 8 MB Random
Seagate 8 MB Sequential
Seagate 8 MB Random
Hitachi 32 MB Sequential
Hitachi 32 MB Random
Samsung 16 MB Sequential
Samsung 16 MB Random
Seagate 16 MB Sequential
Seagate 16 MB Random
WD 16 MB Sequential
WD 16 MB Random
WD 64 MB Sequential
WD 64 MB Random
Seagate 32 MB Sequential
Seagate 32 MB Random
Total data(MB)
# of writes
0
512
1024
1280
2048
2560

Result Summary

Some drives are easy to coerce

- Hitachi

Some drives are harder

- Western Digital

Families of drives seem to be similar

Challenges

- Random policies
- Increasing cache sizes

CCE: Outline

~~Disk Caching: A Study~~

~~Coerced Cache Eviction~~

Discreet-mode Journaling: Using CCE

Results

Discreet Journaling

discreet |disˈkrēt|

adjective (**discreeter**, **discreetest**)

careful and circumspect in one's speech or actions, especially to avoid causing offense:

we made some discreet inquiries.

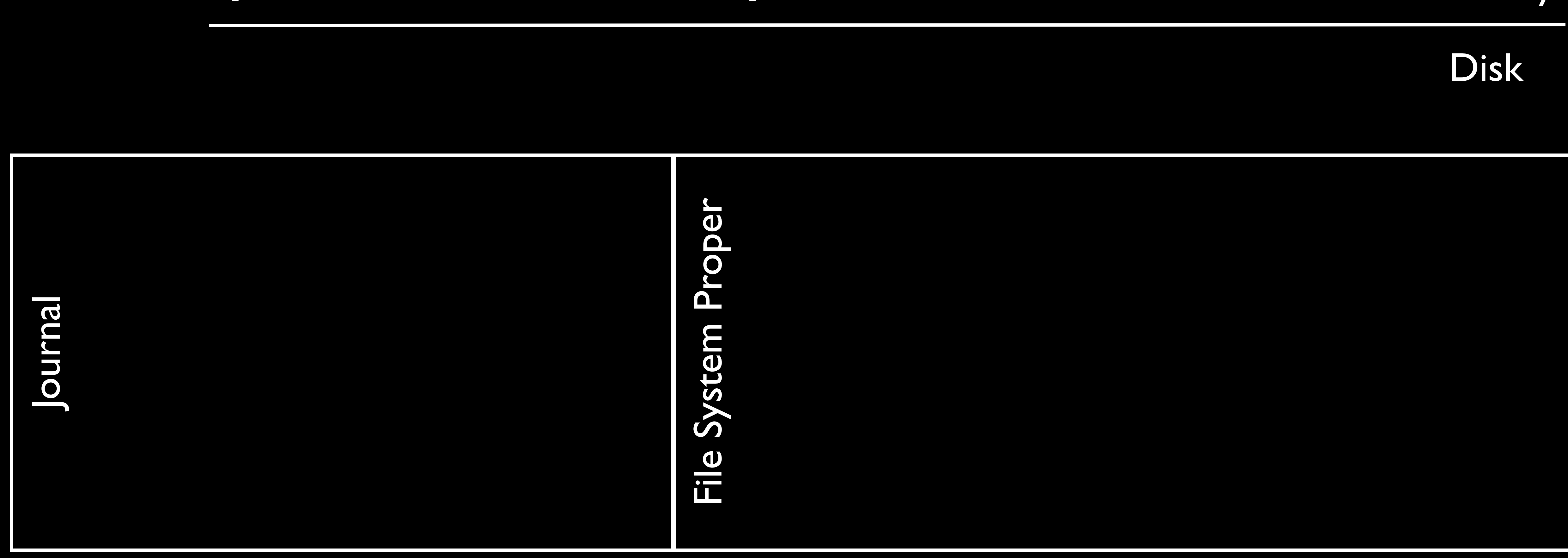
Discreet Journaling

- Use CCE to discreetly enforce write ordering

Typical Journaling

Example: File Append

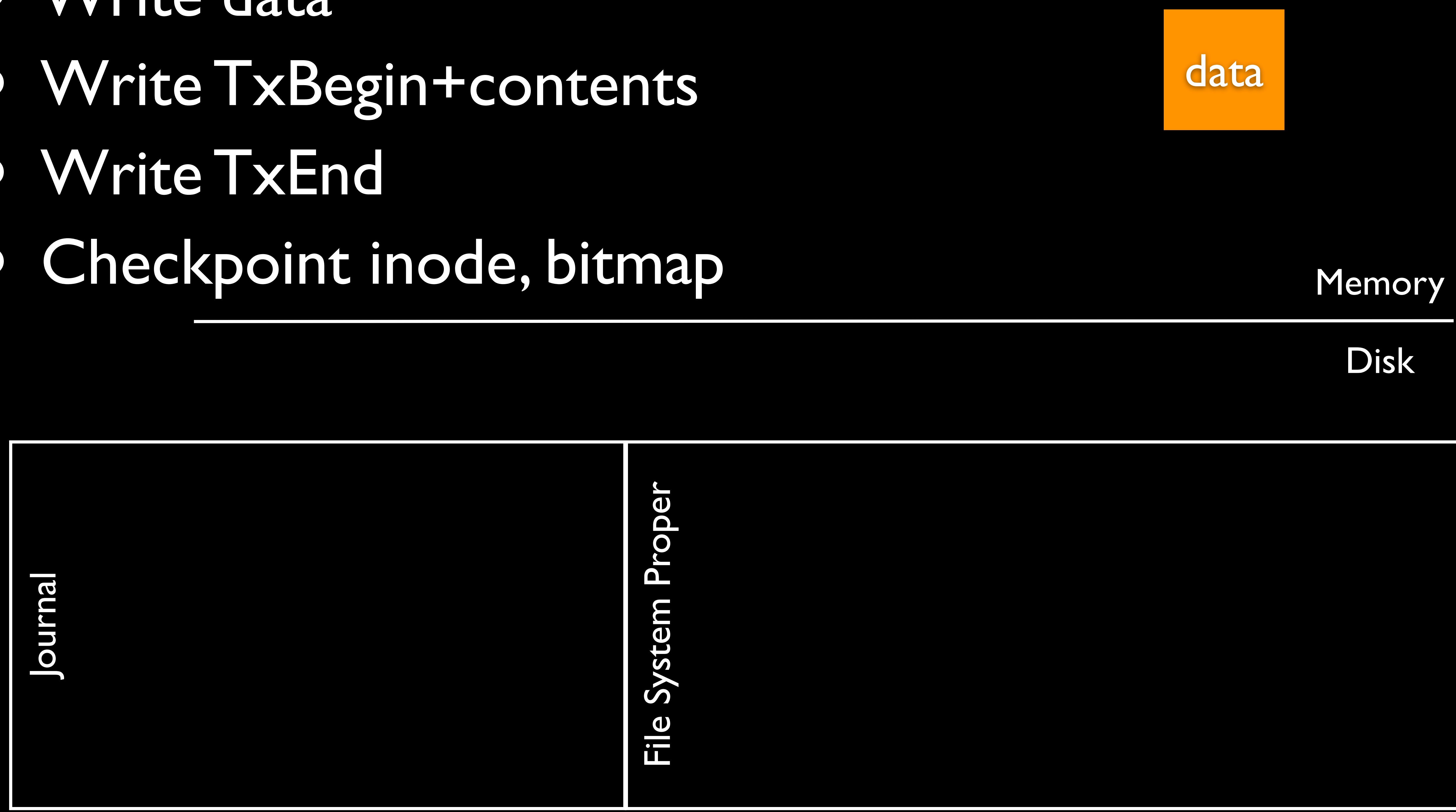
- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap



Typical Journaling

Example: File Append

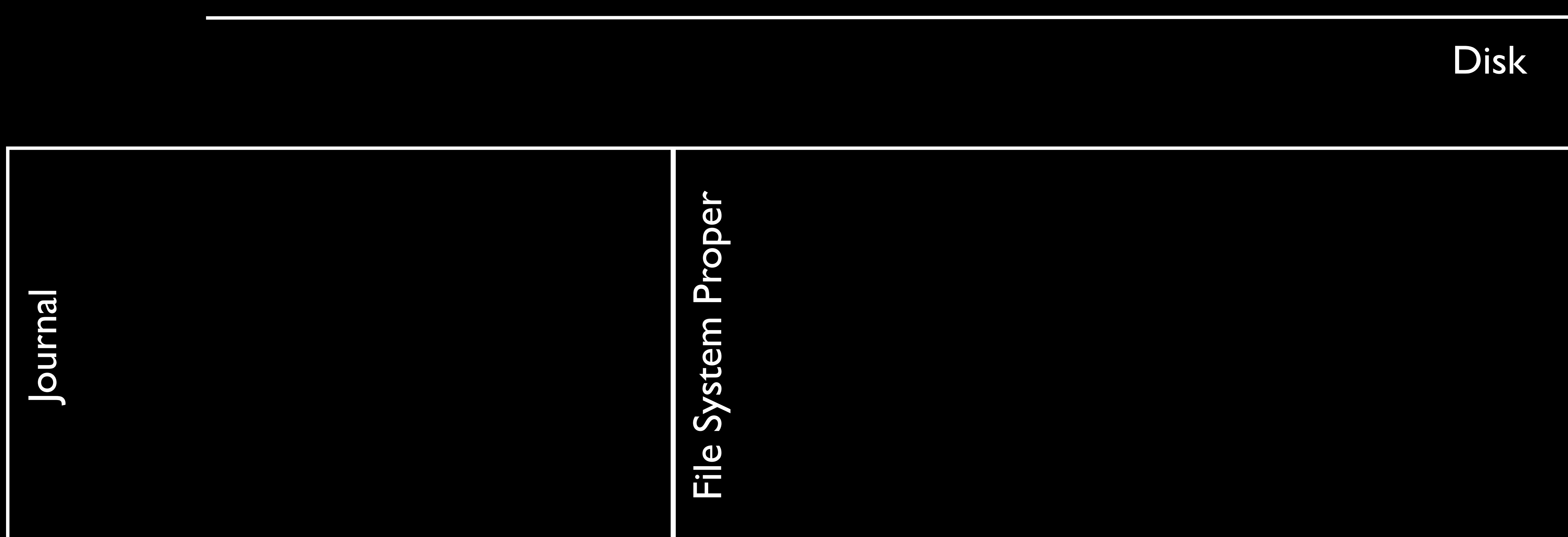
- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap



Typical Journaling

Example: File Append

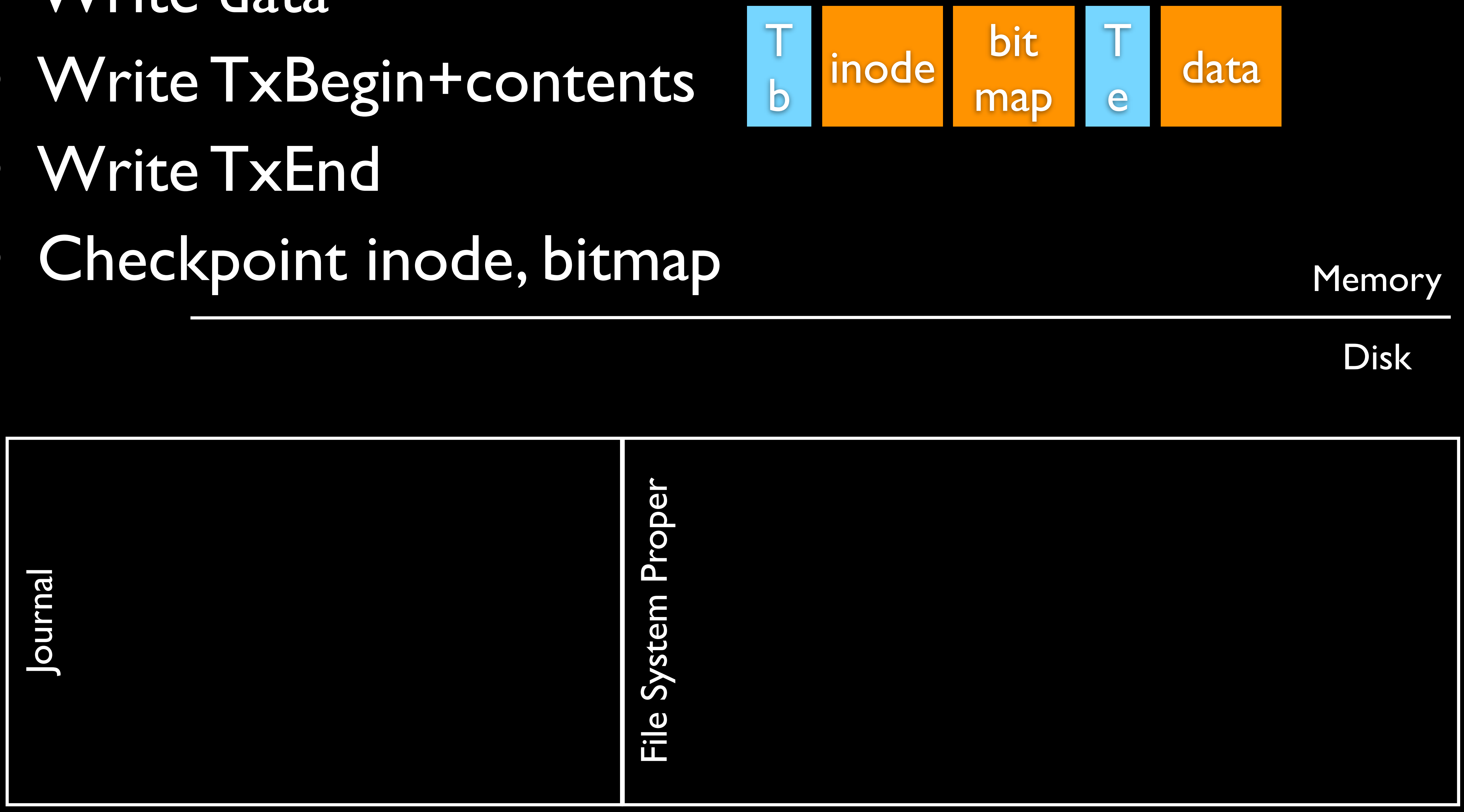
- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap



Typical Journaling

Example: File Append

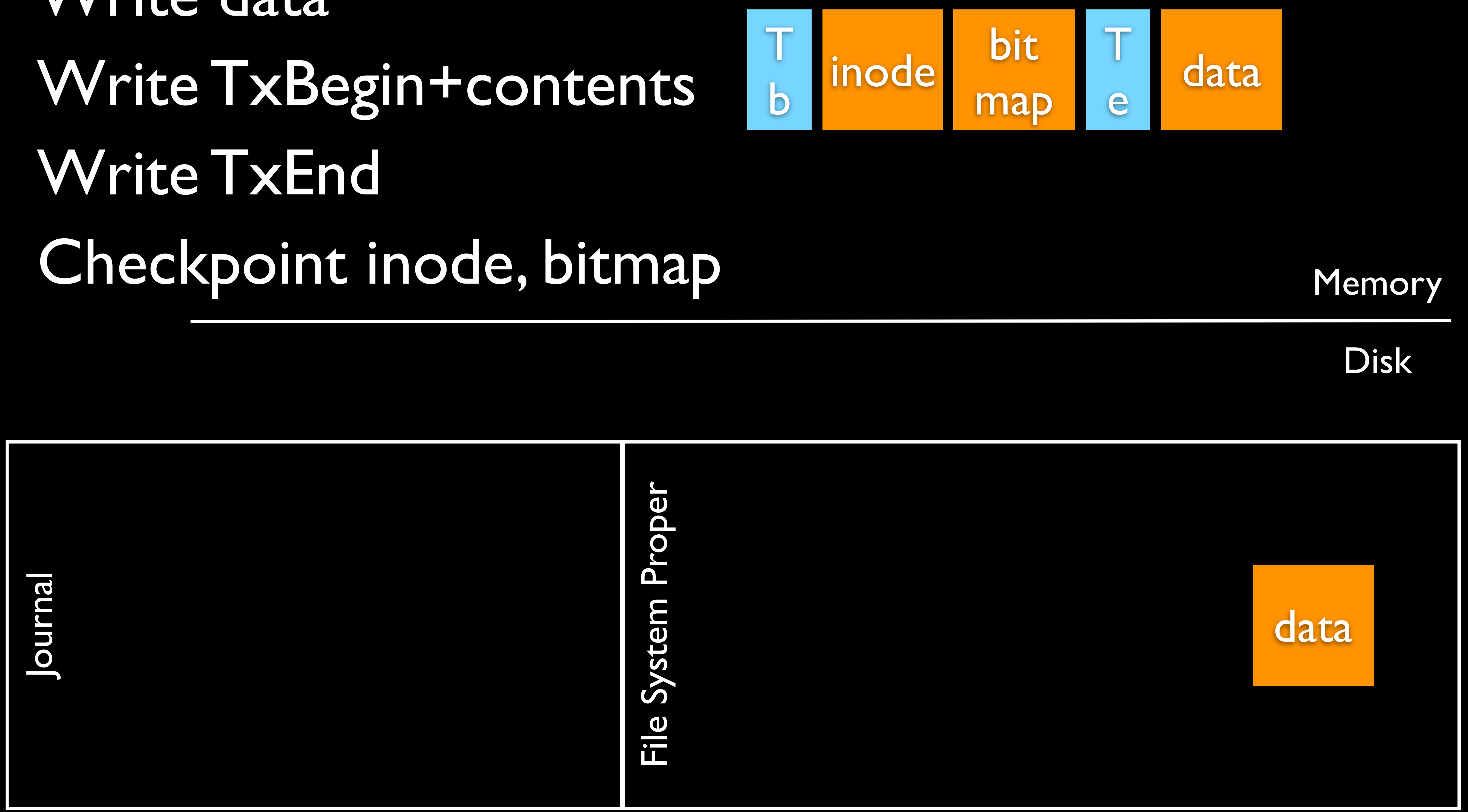
- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap



Typical Journaling

Example: File Append

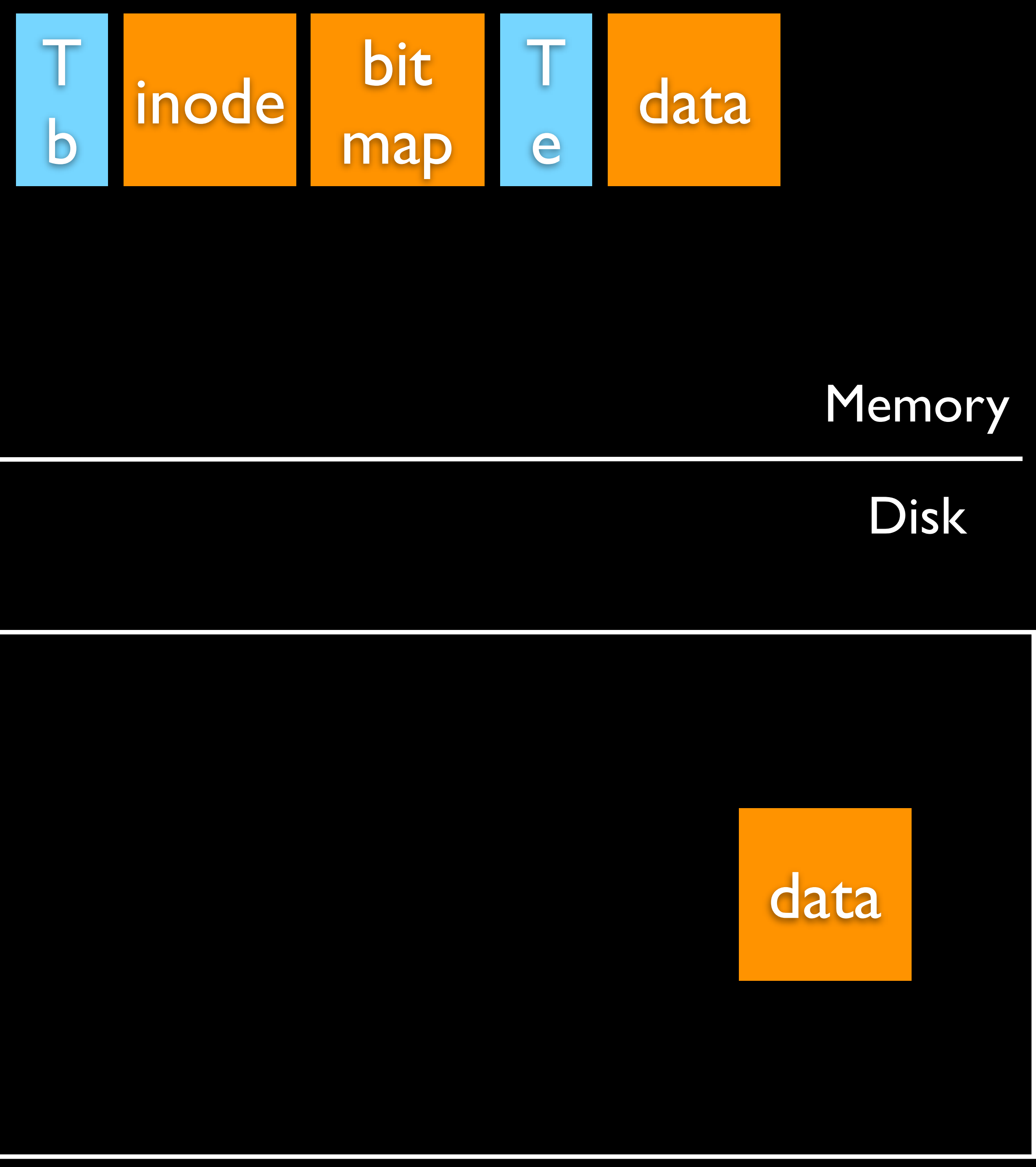
- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap



Typical Journaling

Example: File Append

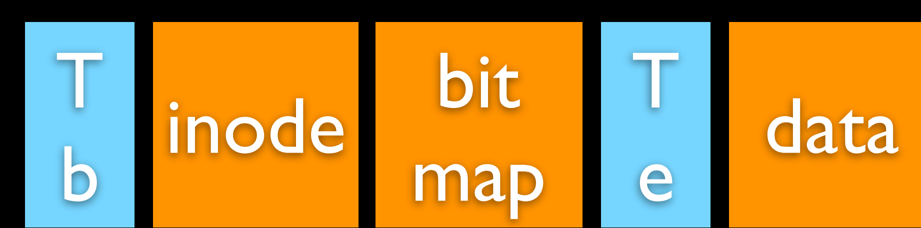
- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap

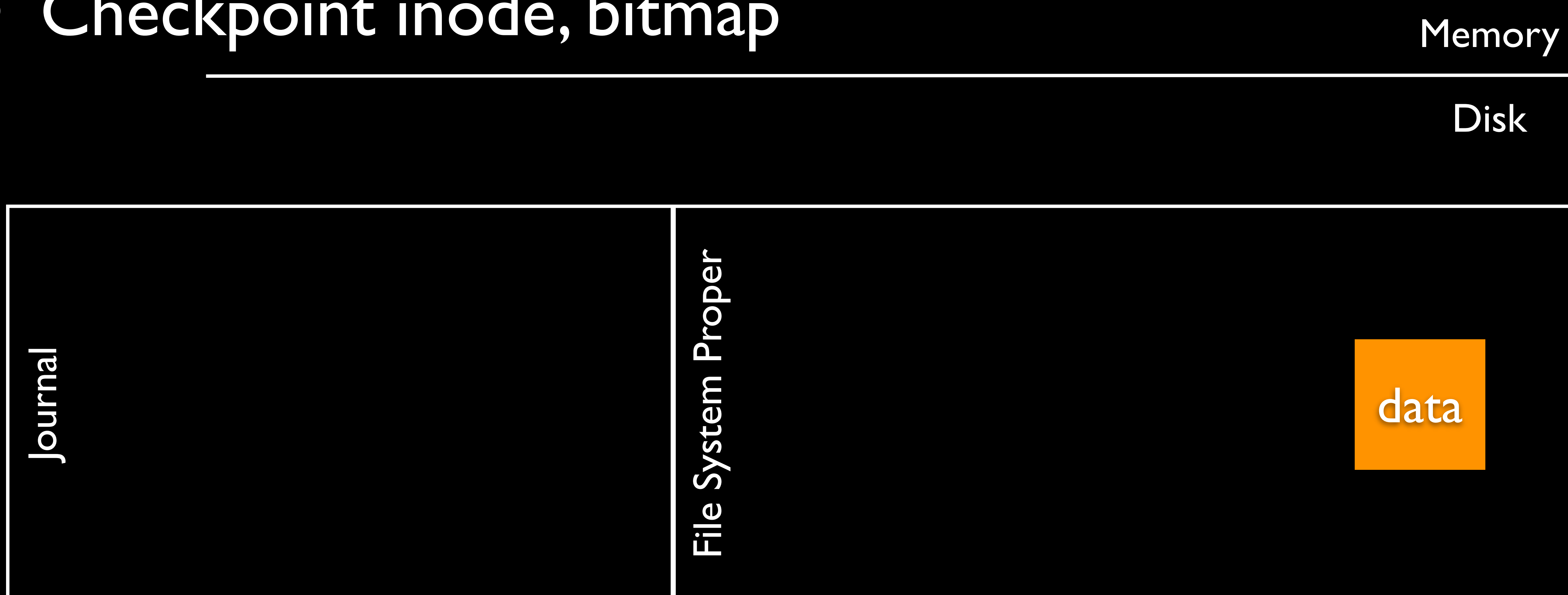


Typical Journaling

Example: File Append

- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap

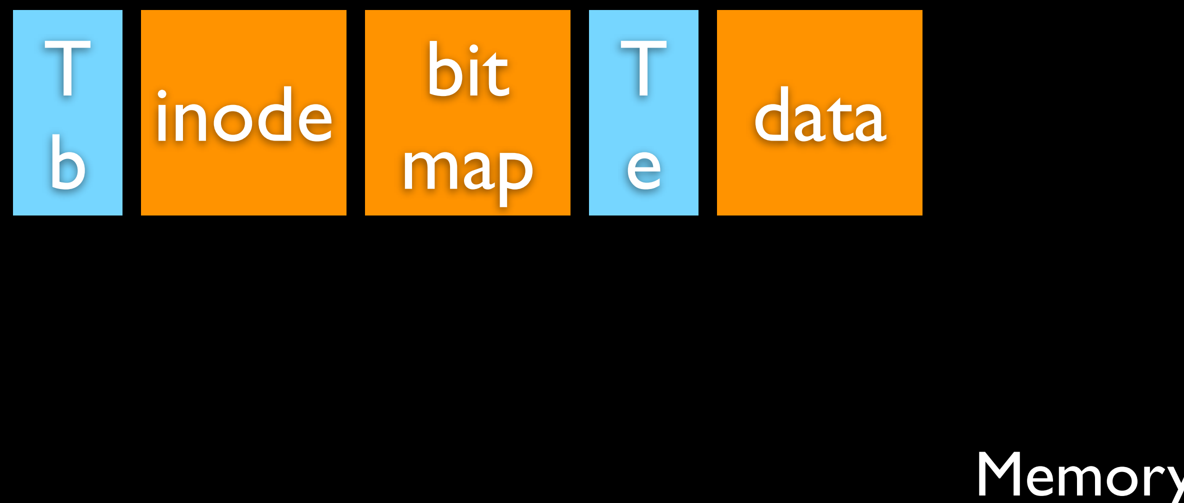


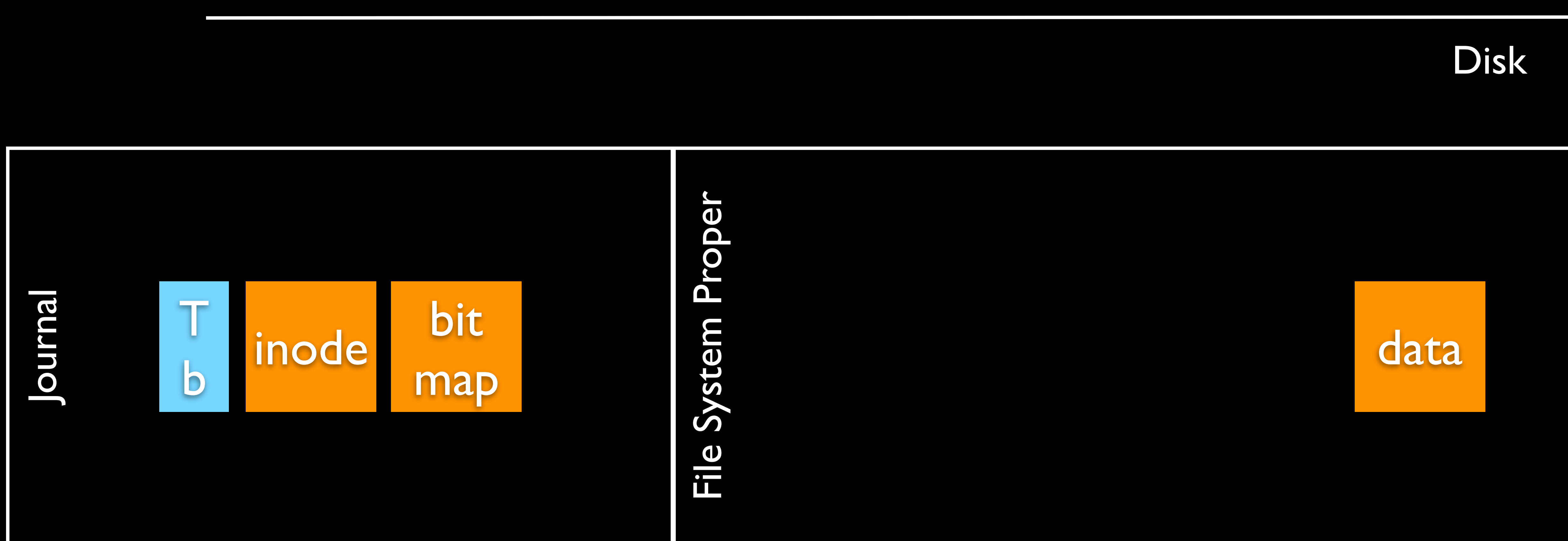


Typical Journaling

Example: File Append

- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap

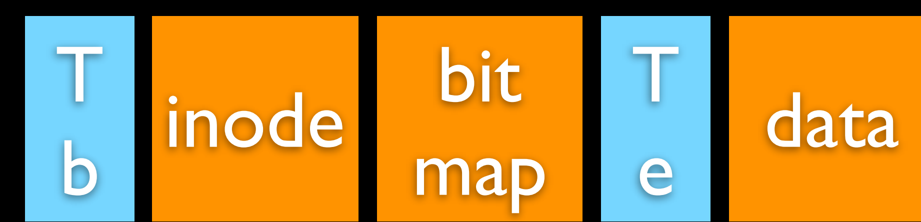


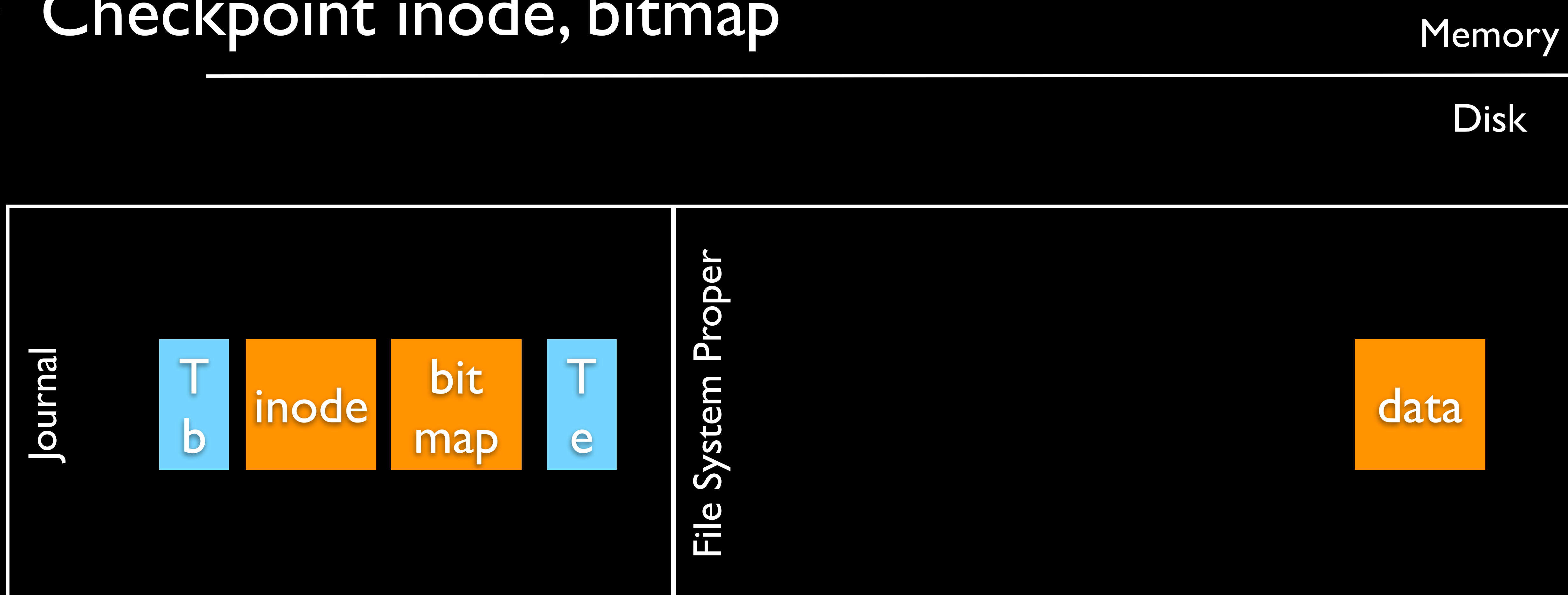


Typical Journaling

Example: File Append

- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap

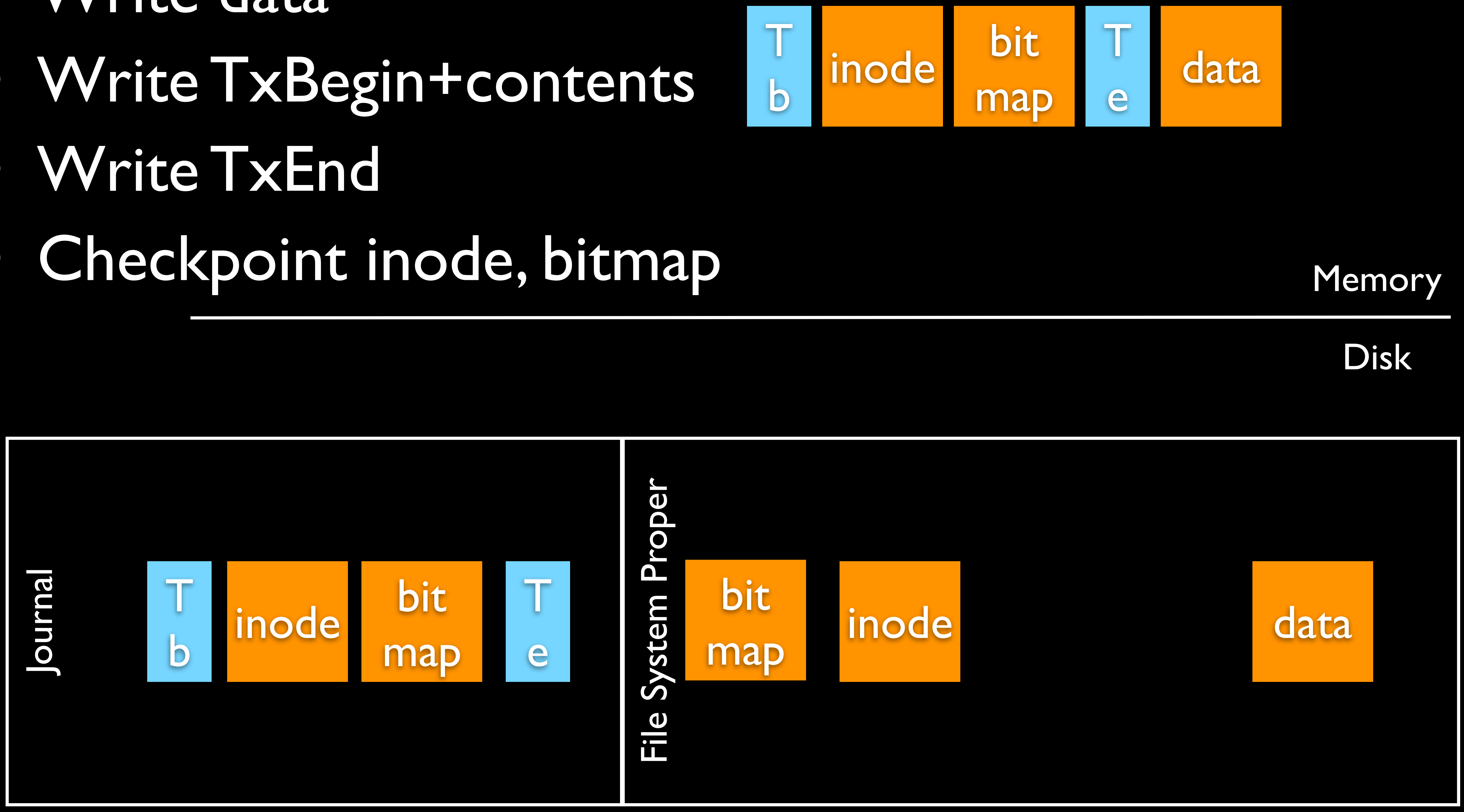




Typical Journaling

Example: File Append

- Write data
- Write TxBegin+contents
- Write TxEnd
- Checkpoint inode, bitmap



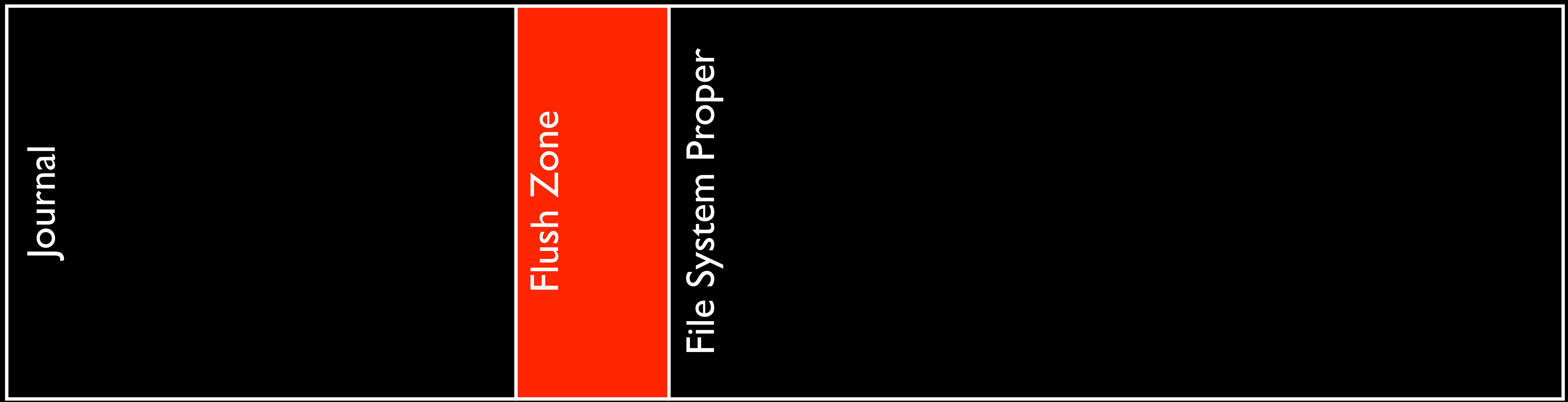
Discreet Journaling

Same basic protocol

- e.g., data in place, metadata to journal, etc.

Additions

- On-disk flush zone
- CCE at all ordering points;
  writes issued to flush zone to flush cache



CCE: Outline

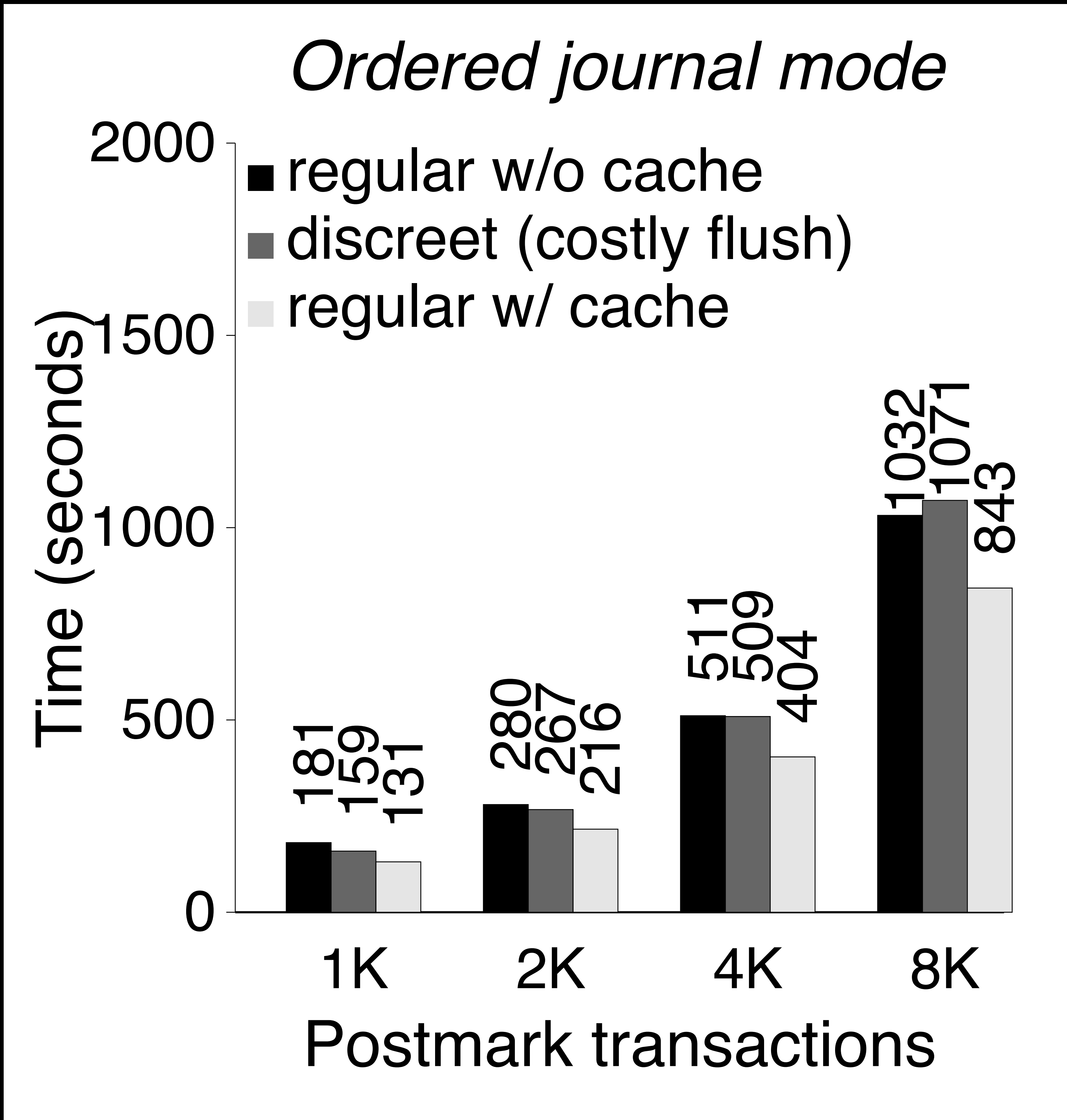
~~Disk Caching: A Study~~

~~Coerced Cache Eviction~~

~~Discreet-mode Journaling: Using CCE~~

Results

Performance



Benchmark

- Postmark

Vary

- Tx size

Plot

- Total time (s)

Results

- Performance without trust

Summary

Disk caches

- What if we don't trust them to flush?

Coerced Cache Eviction

- Method to enforce ordering without trust

Discreet ext3

- Uses CCE to build crash-consistent write protocol without explicit disk support
- Performance is good enough (usually)
- Depends strongly on exact replacement algorithm

Orderless File Systems

Classic Approach: ext2

One classic approach: ext2-style consistency

- Write blocks to disk in any order
- Upon crash, run fsck to fix before mount

Problems

- Slow: Check time is prohibitive
  (and have to fully check before mount)
- Weak: Doesn't provide many guarantees

Can we do better?

NoFS

NoFS: No-order File System

- Writes blocks to disk in any order
- Provides reasonable consistency guarantees

Backpointer-Based Consistency (BBC)

- Every pointed-to object has backpointer to object that points to it

Results

- Simple, lightweight, performant FS
- No need for ordering or pre-mount fsck

NoFS: Outline

BBC: Basic idea

Implementing NOFS

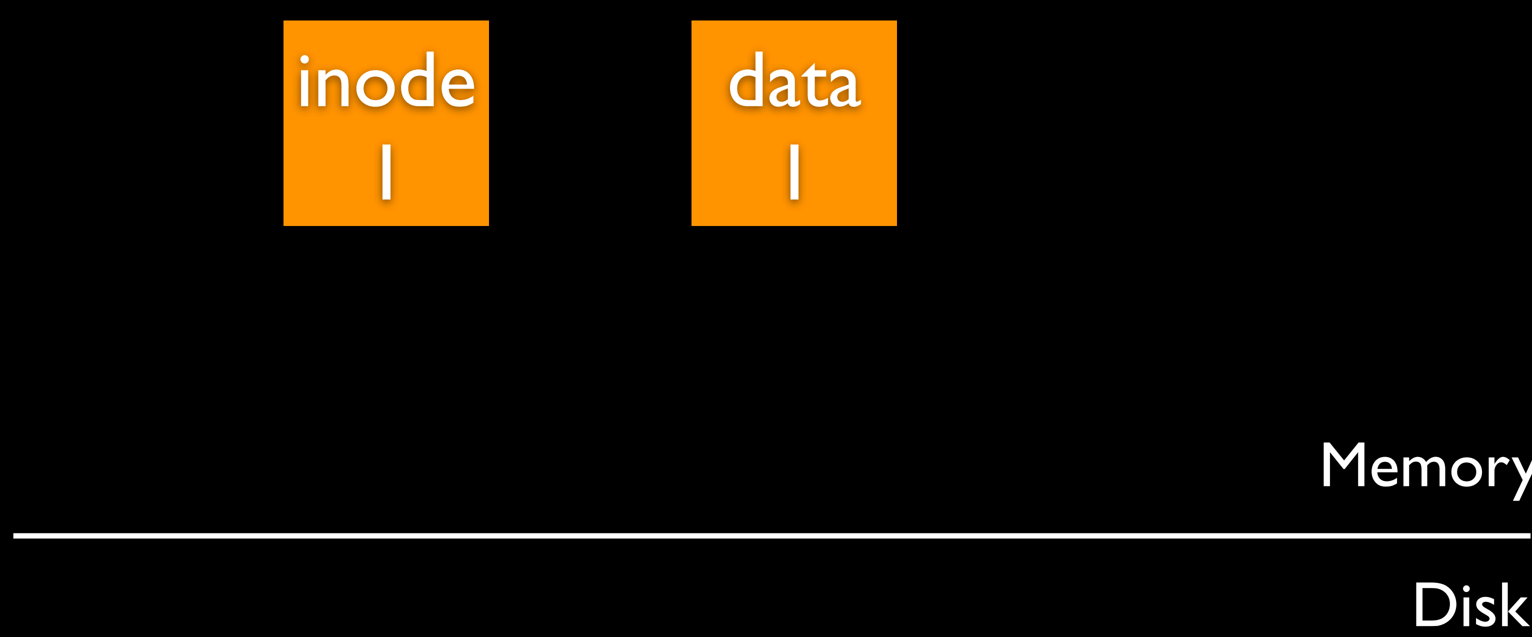
Results

Why Inconsistency Arises Without Order

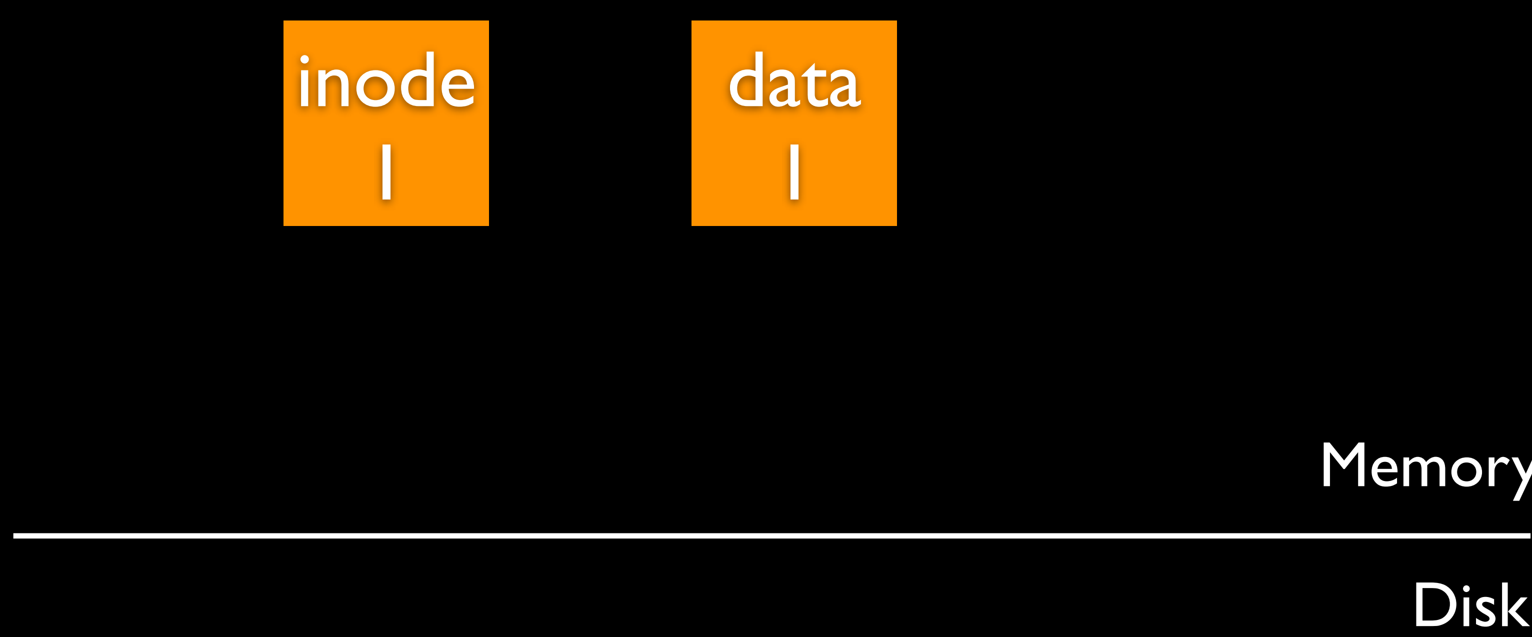
Memory

Disk

Why Inconsistency Arises Without Order

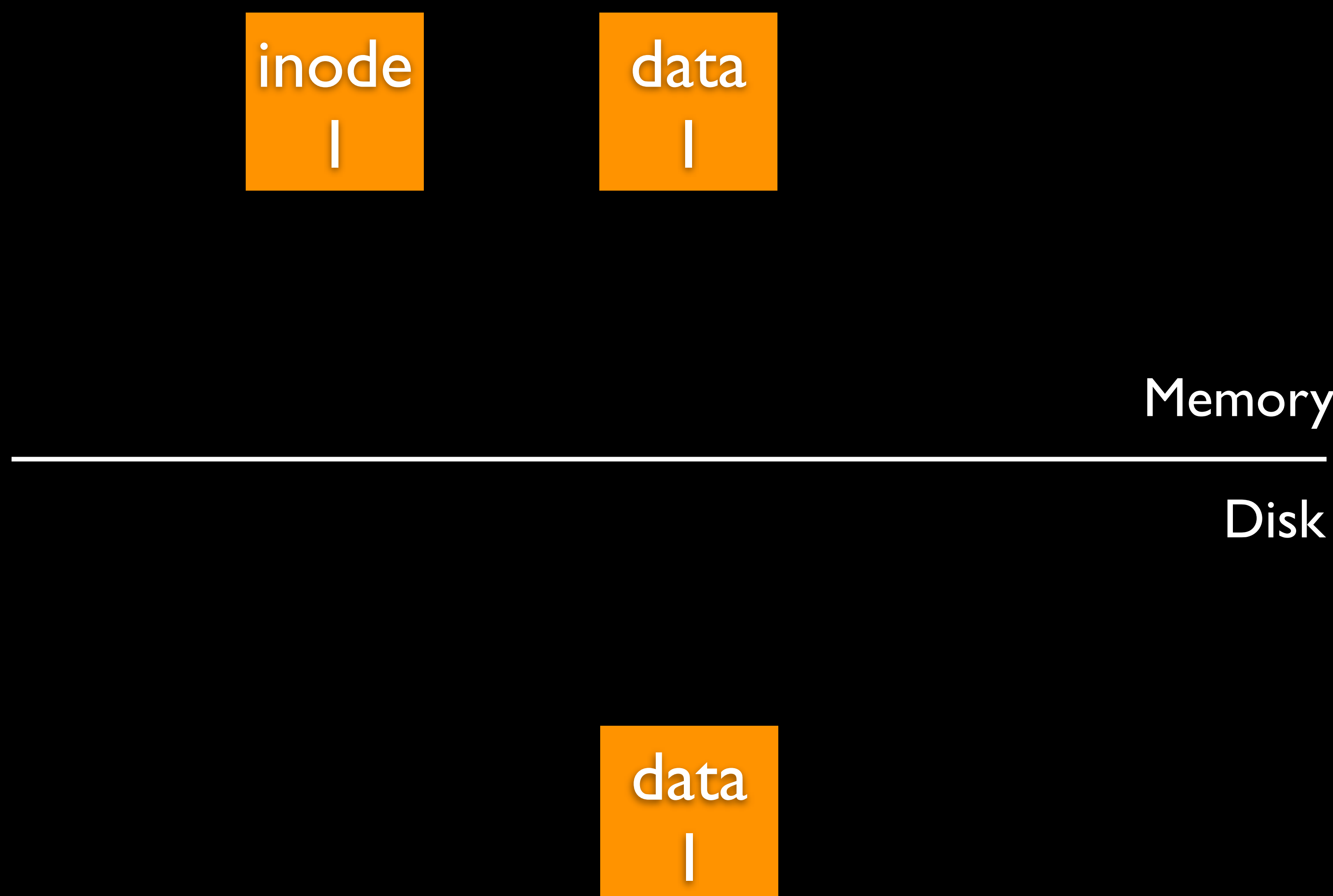


Why Inconsistency Arises Without Order

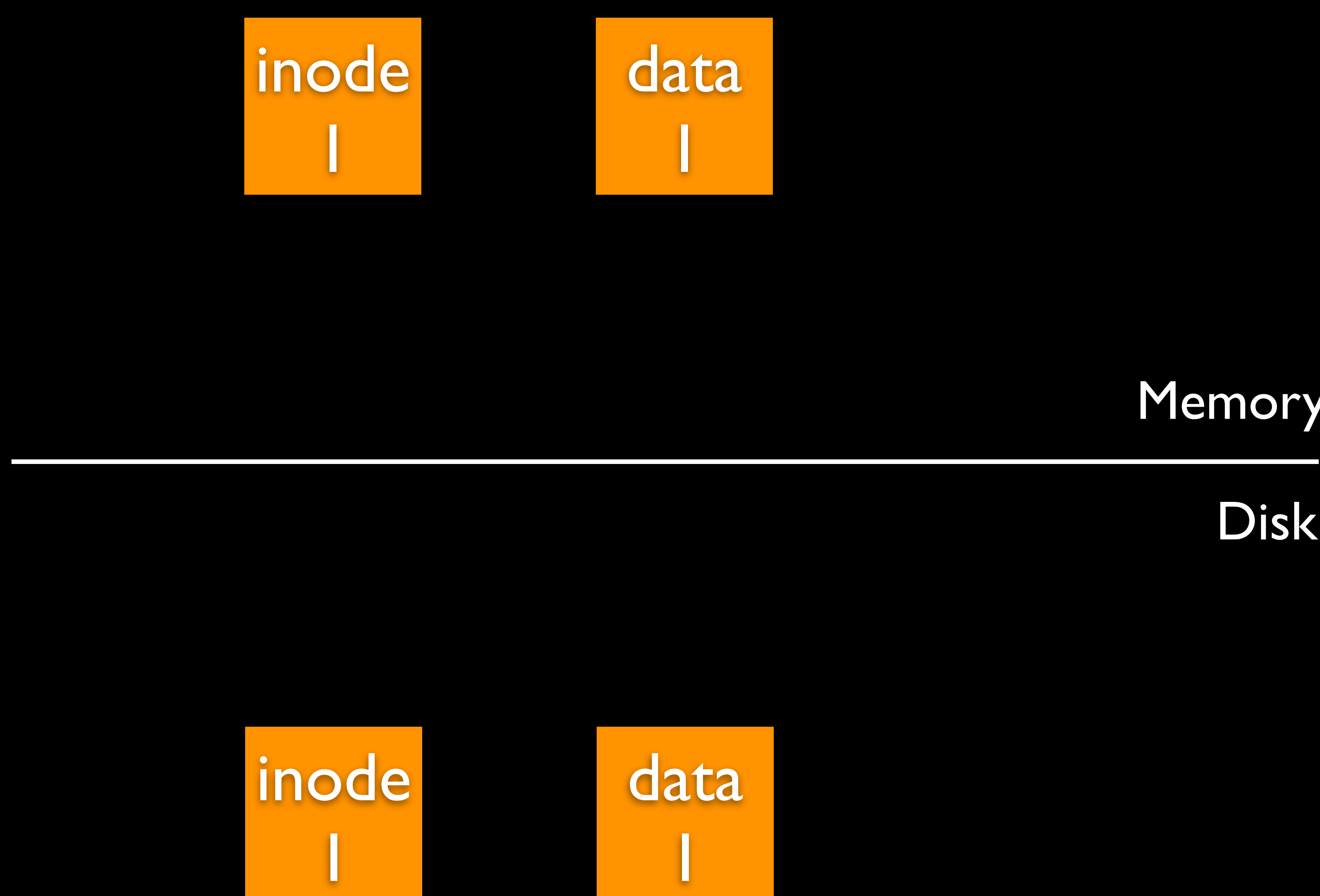


Why Inconsistency Arises Without Order

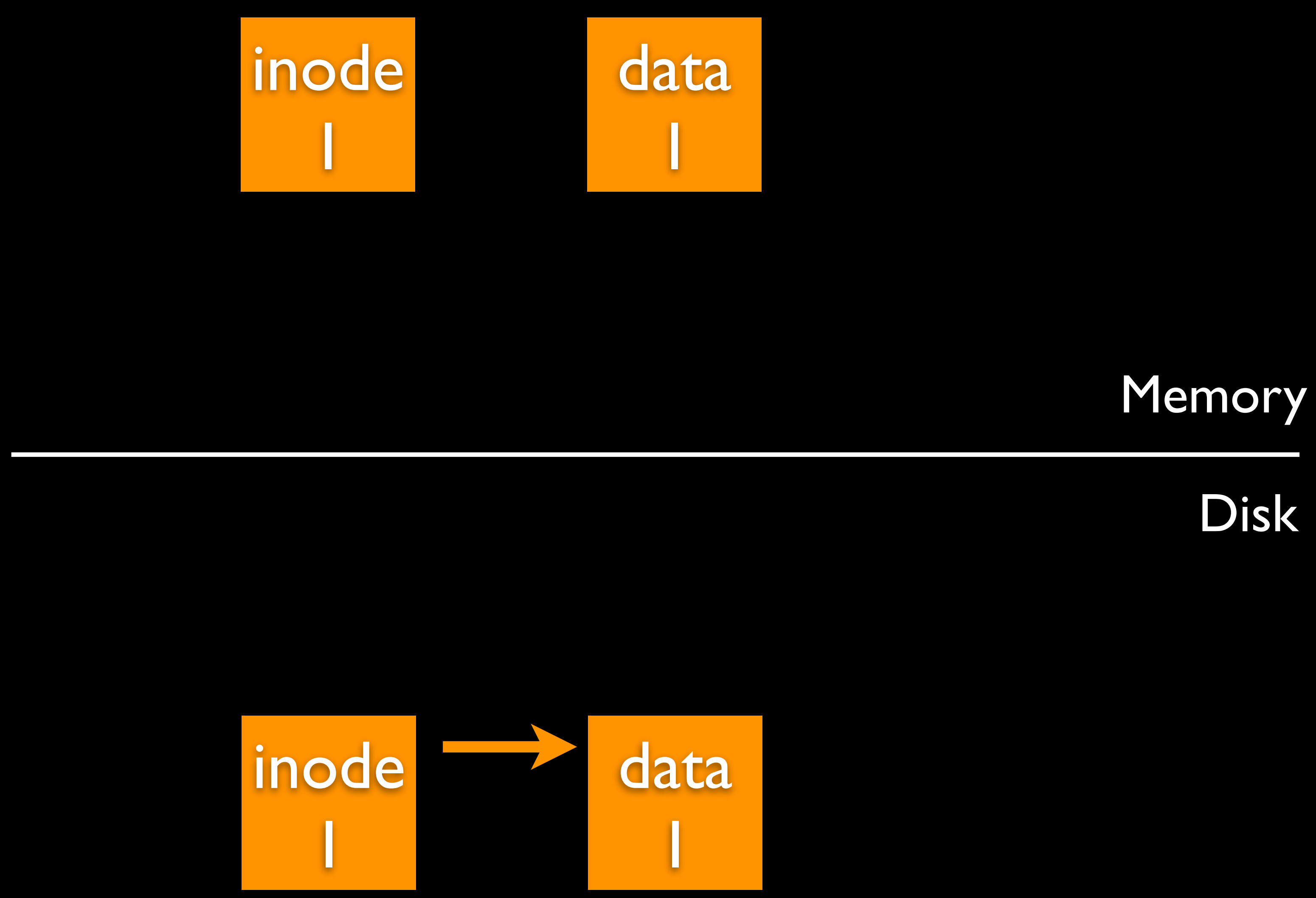
Why Inconsistency Arises Without Order



Why Inconsistency Arises Without Order

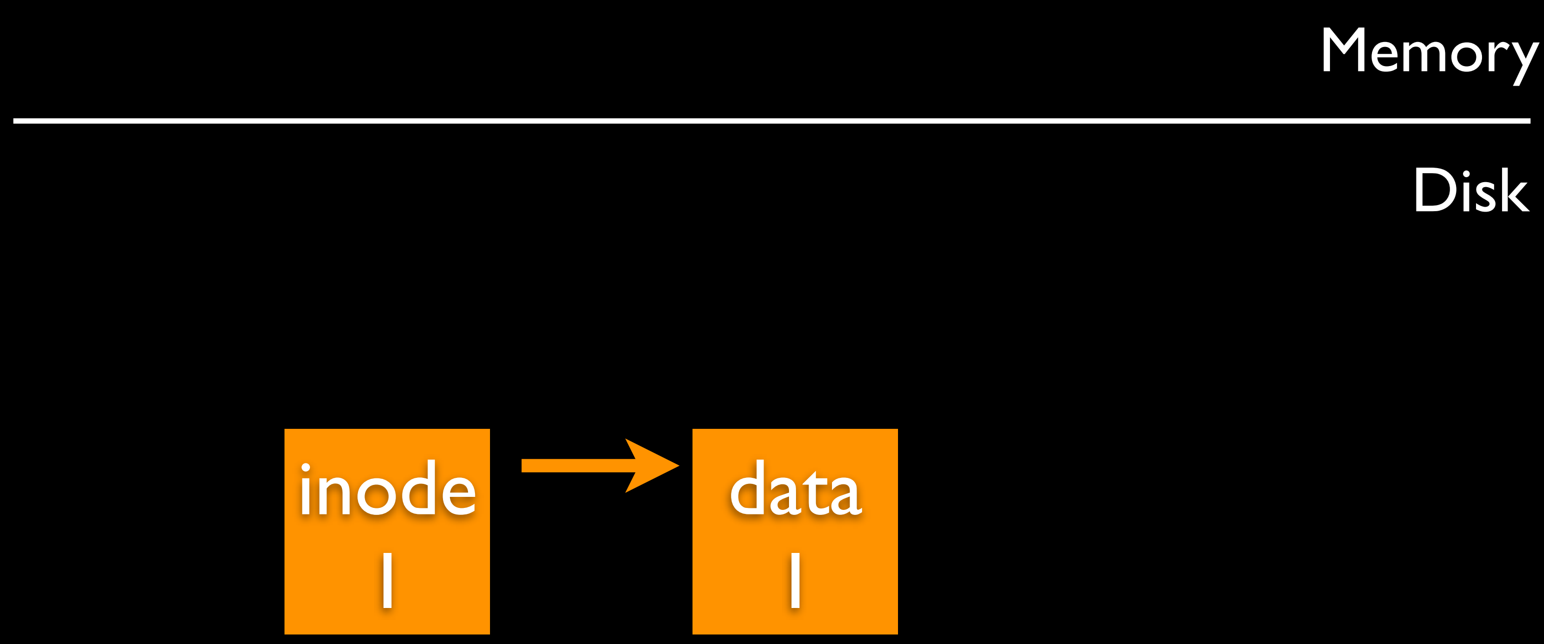


Why Inconsistency Arises Without Order

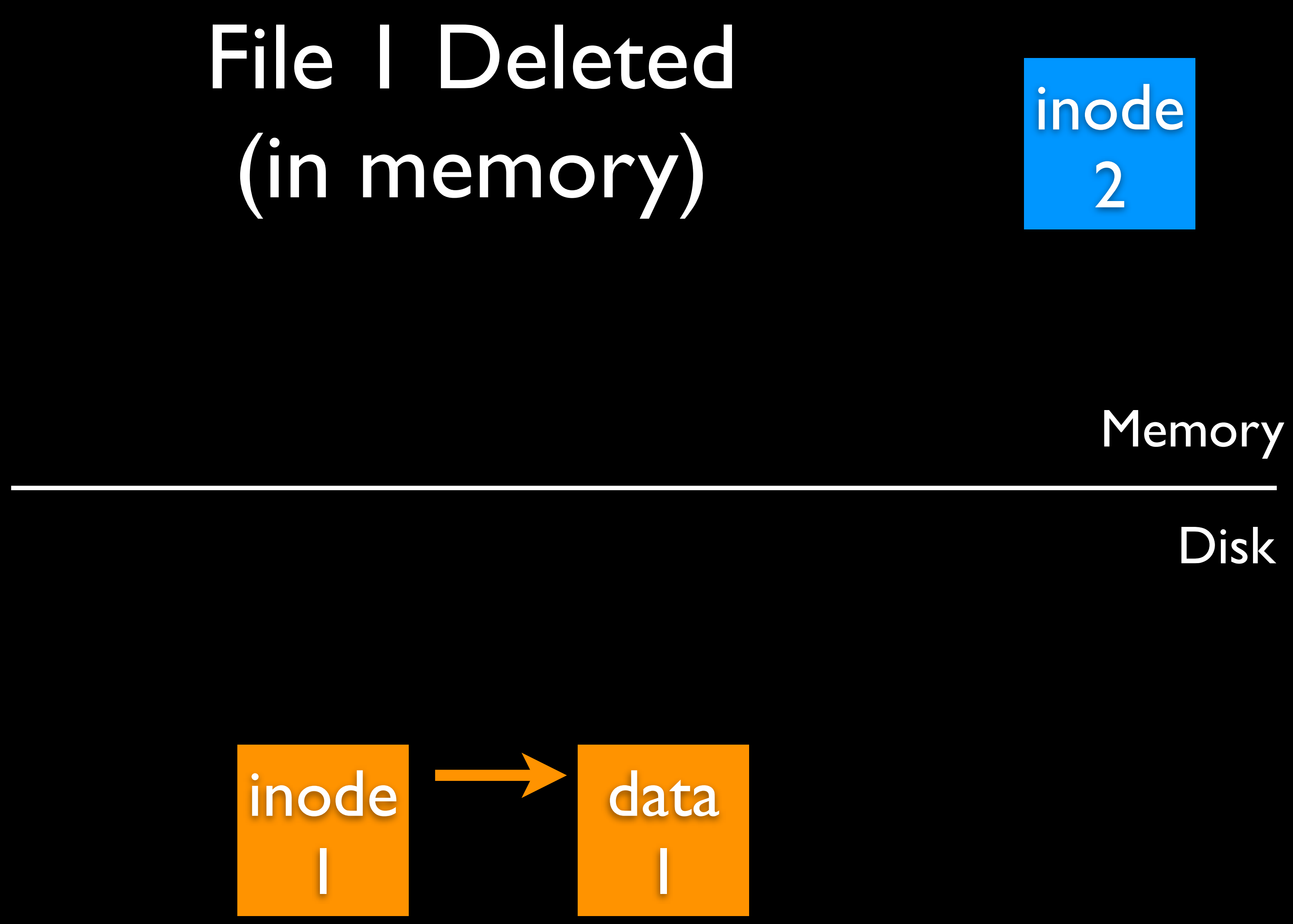


Why Inconsistency Arises Without Order

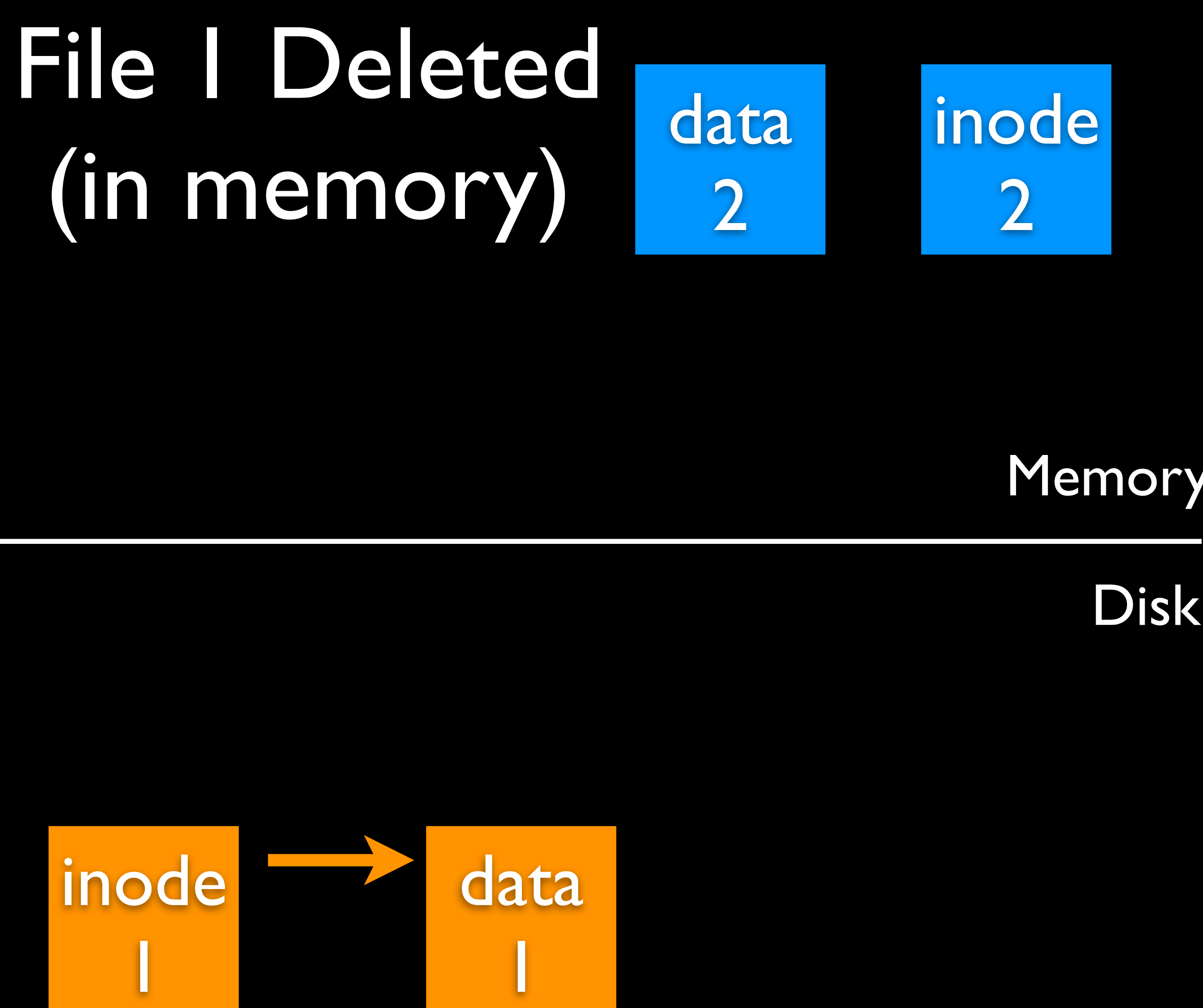
File 1 Deleted
(in memory)



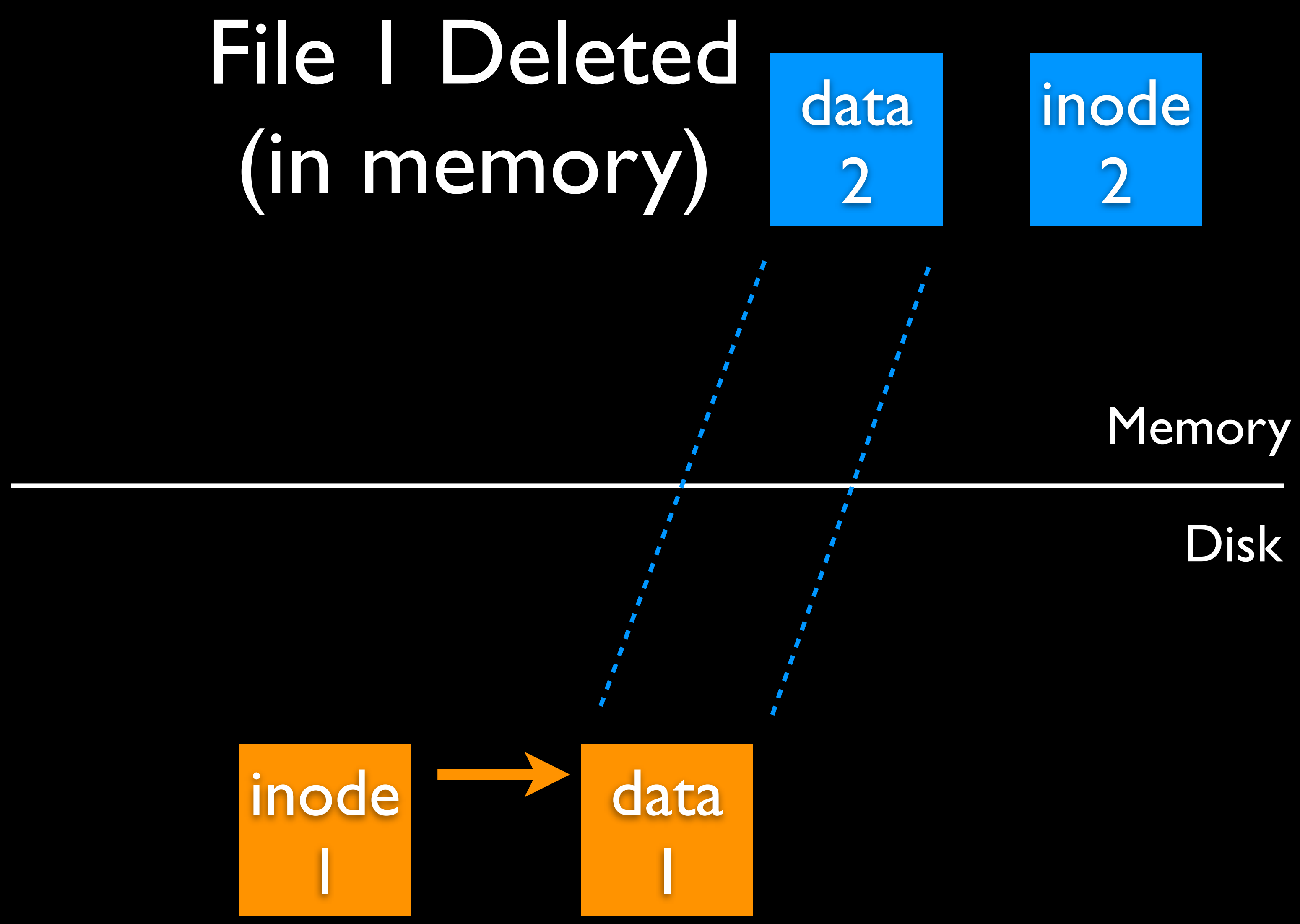
Why Inconsistency Arises Without Order



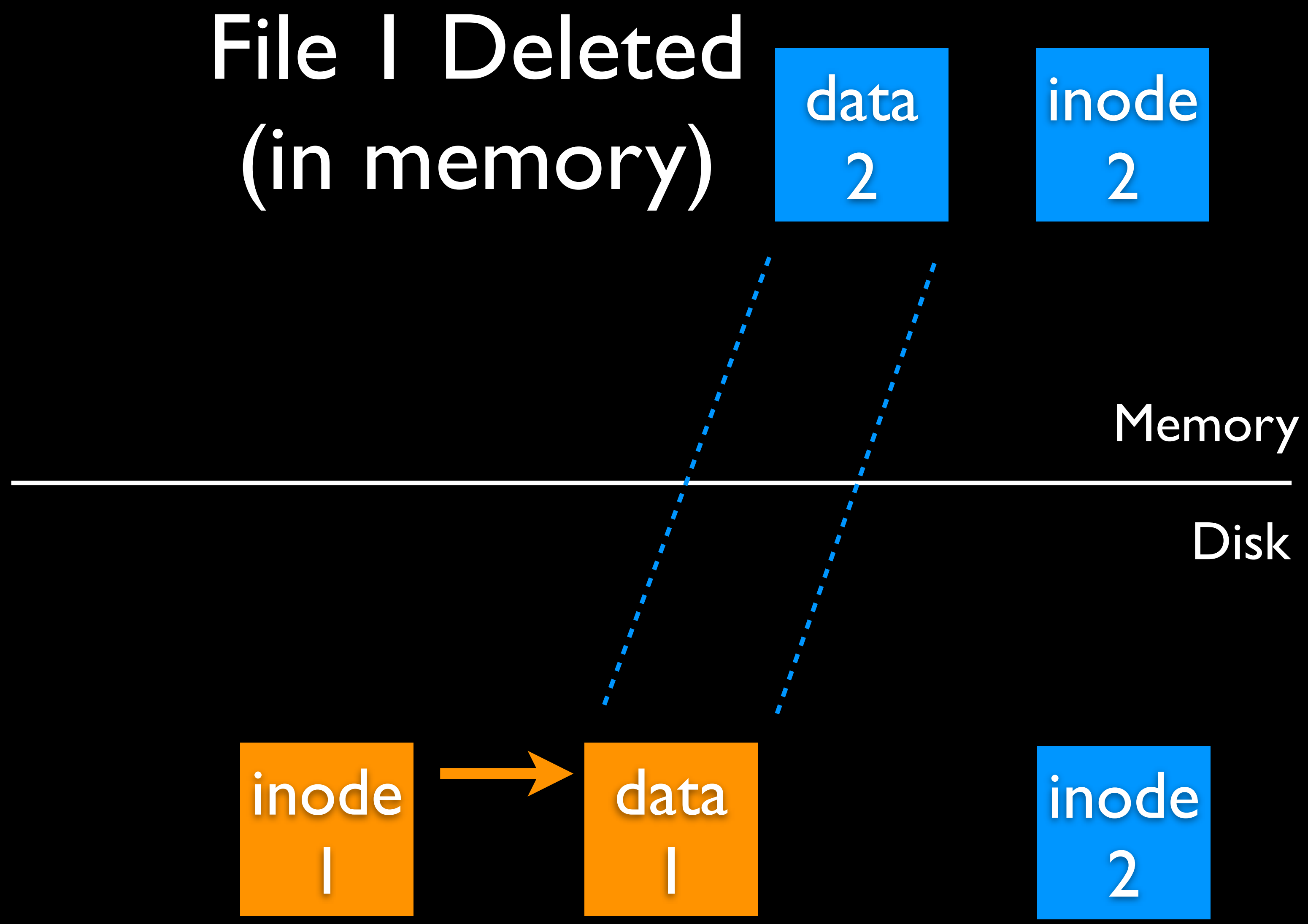
Why Inconsistency Arises Without Order



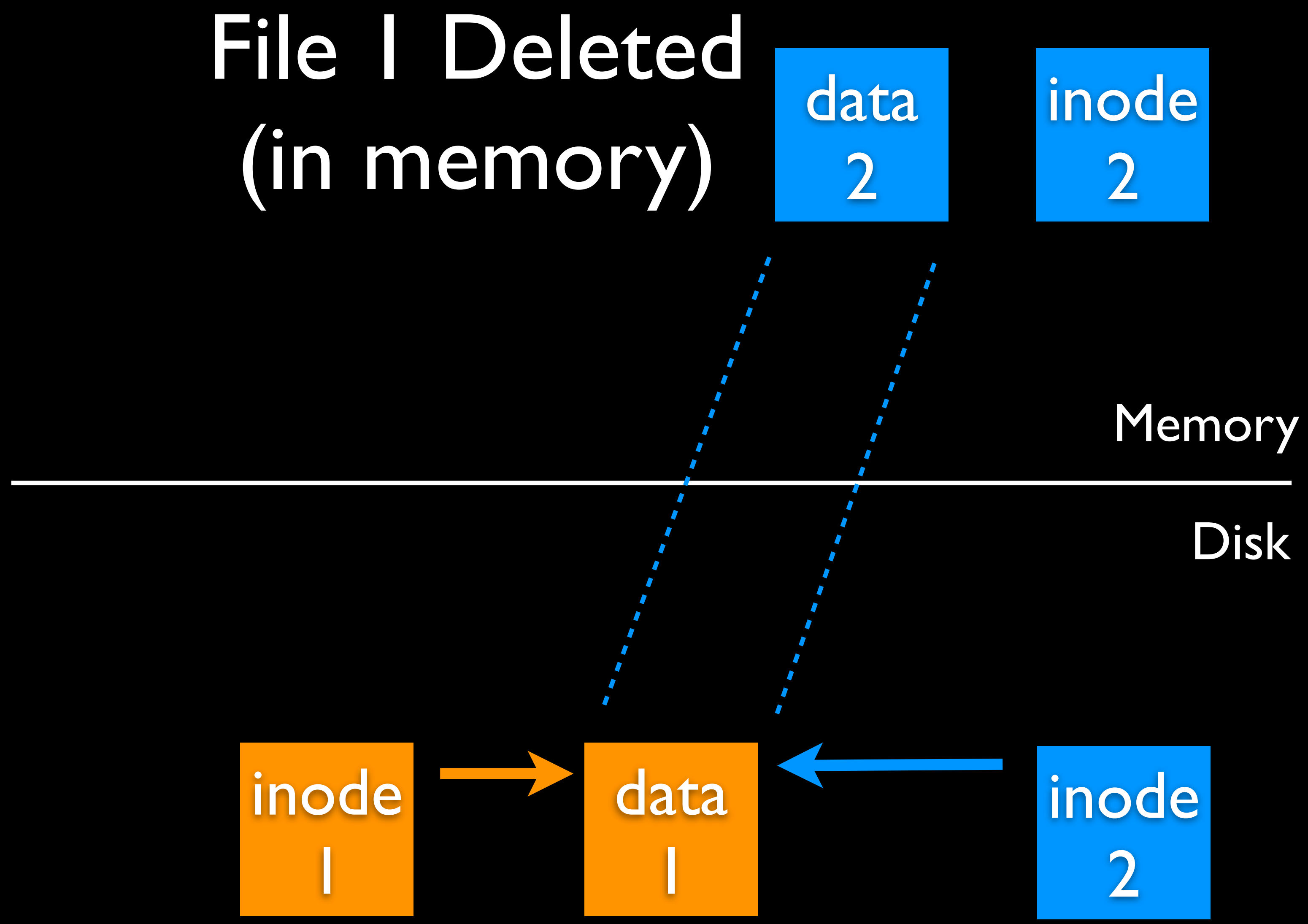
Why Inconsistency Arises Without Order



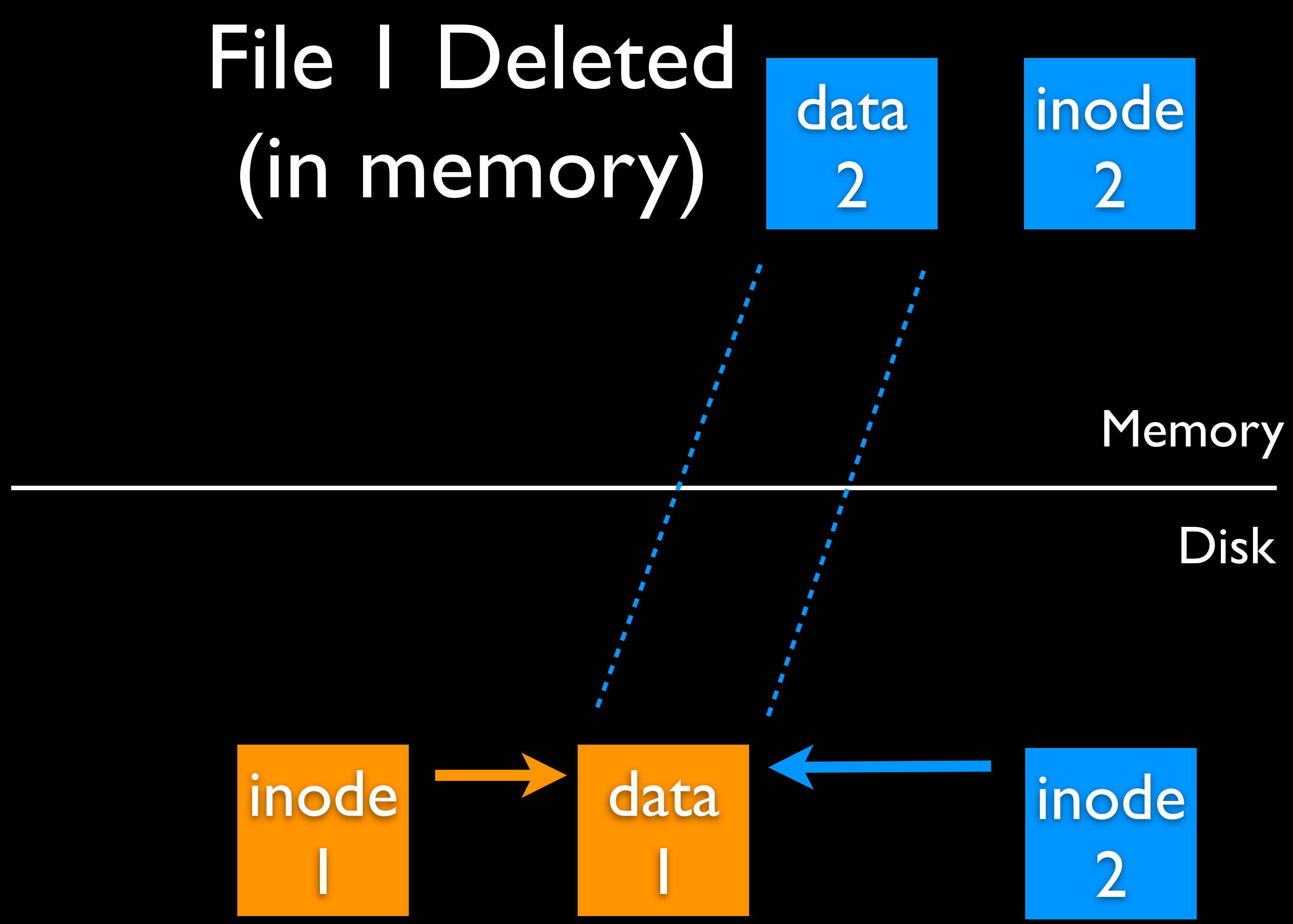
Why Inconsistency Arises Without Order



Why Inconsistency Arises Without Order



Why Inconsistency Arises Without Order



Disk image now inconsistent

Backpointer-Based Consistency

Backpointer-Based Consistency

Simple idea:

- Each pointed-to object points back at its parent
- Agreement implies consistency

Backpointer-Based Consistency

Simple idea:

- Each pointed-to object points back at its parent
- Agreement implies consistency

Examples:

- Data block: Add pointer to its inode
- Directory block: Use existing "." entry
- Inode: Add pointers to all directories it is in (requires multiple back pointers in inode)

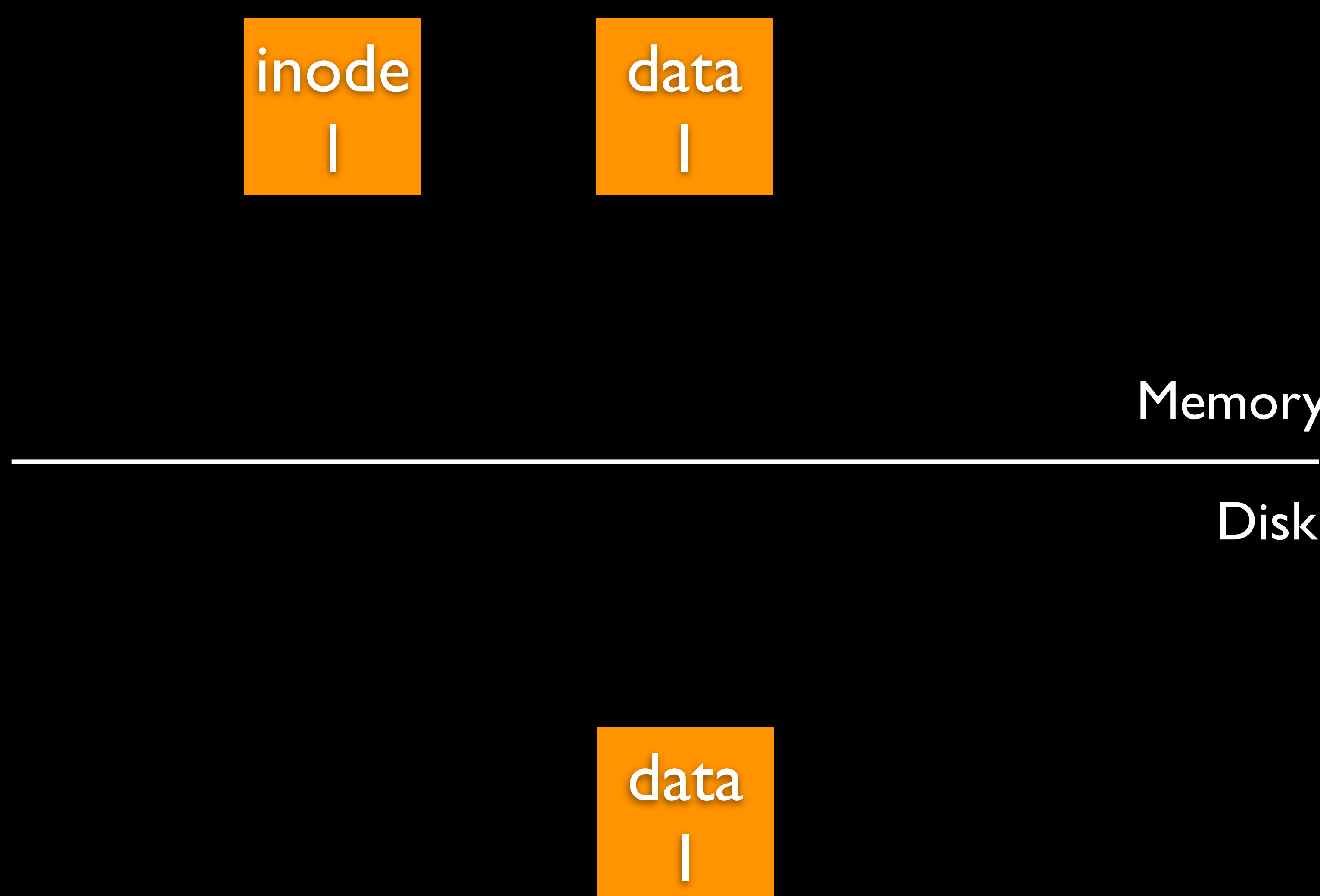
Why Consistency Arises With BBC

Memory

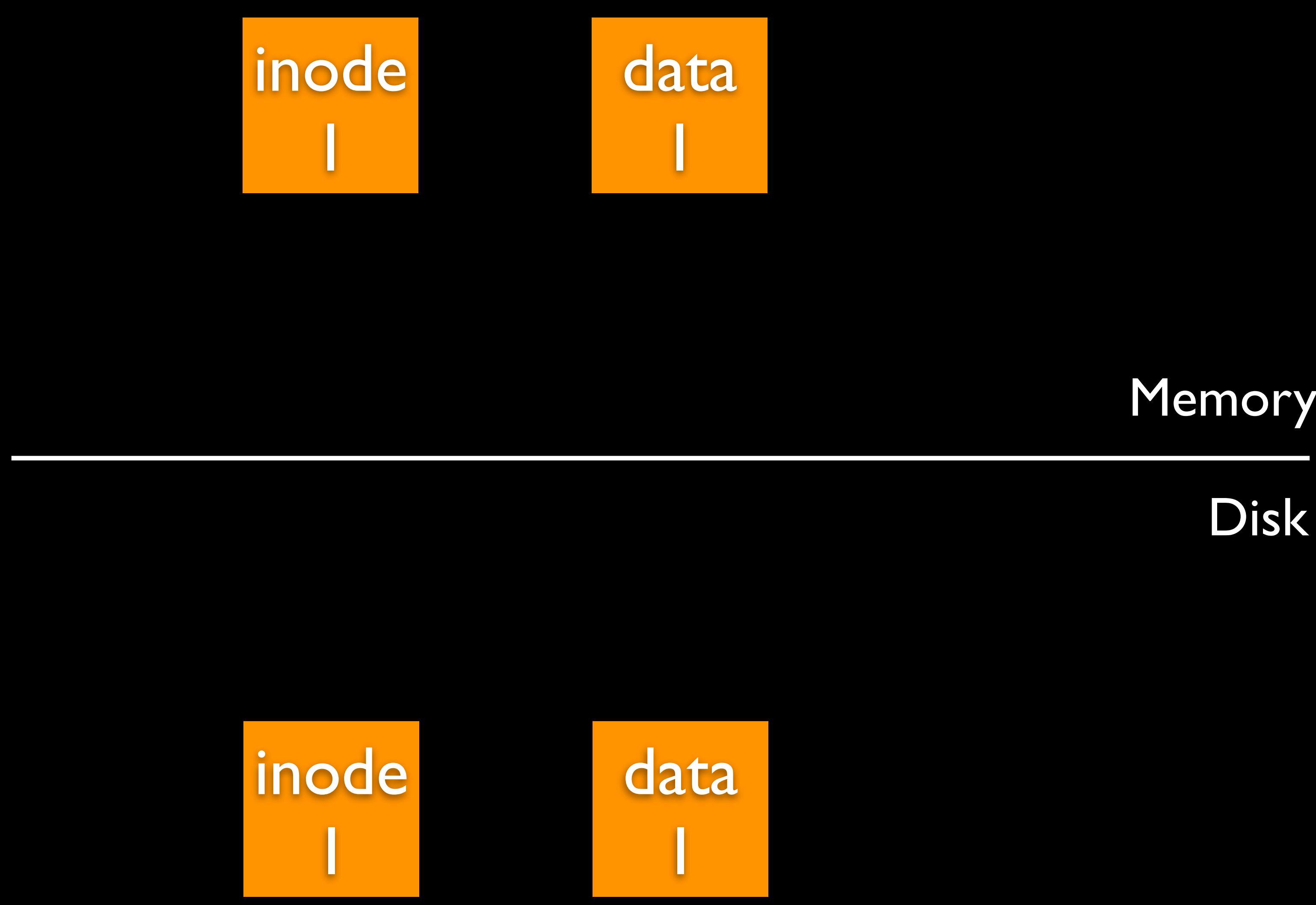
Disk

Why Consistency Arises With BBC

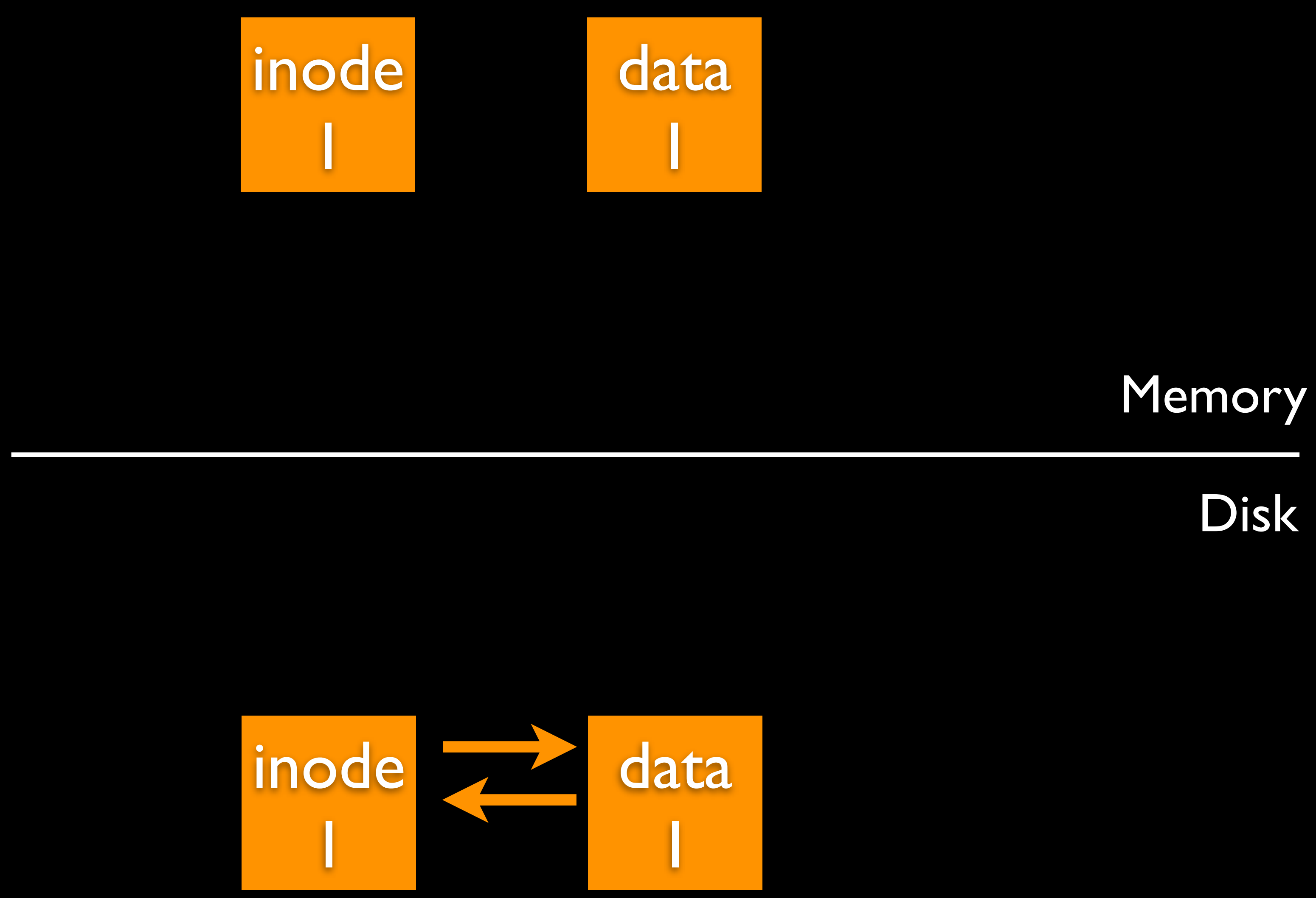
Why Consistency Arises With BBC



Why Consistency Arises With BBC

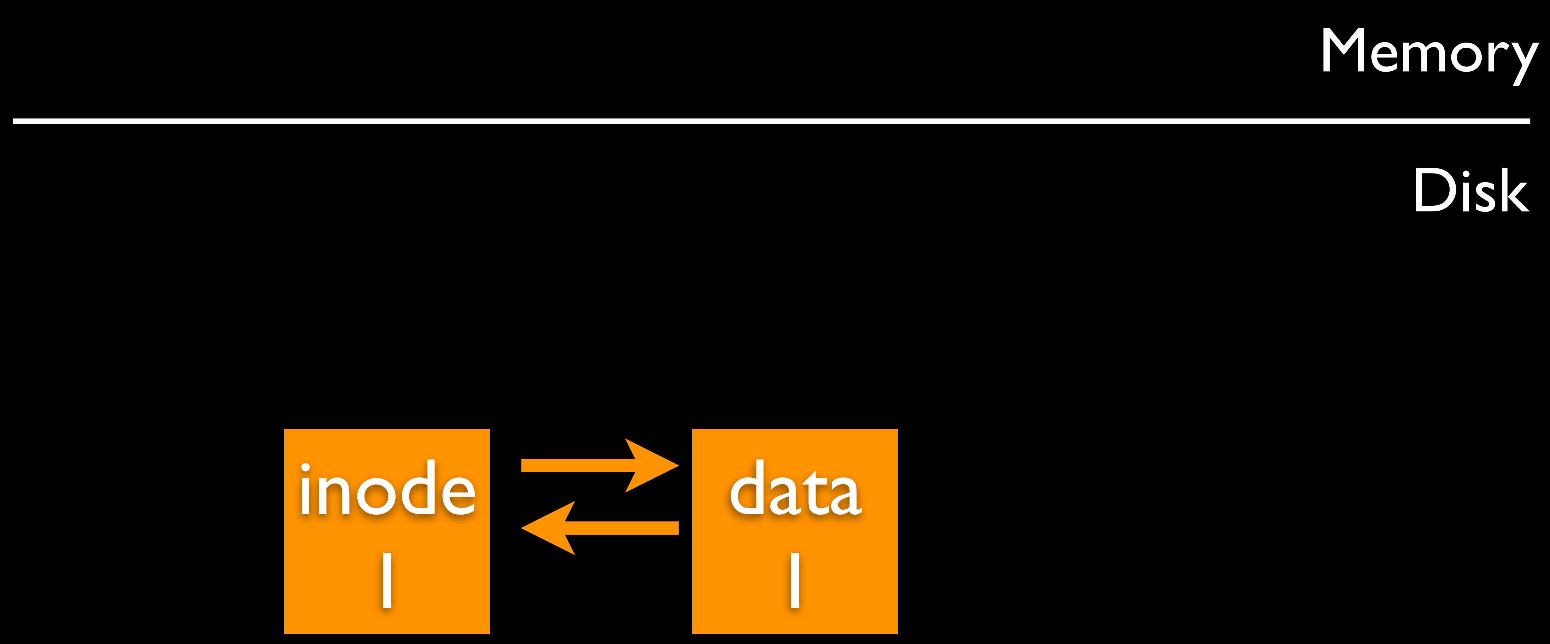


Why Consistency Arises With BBC

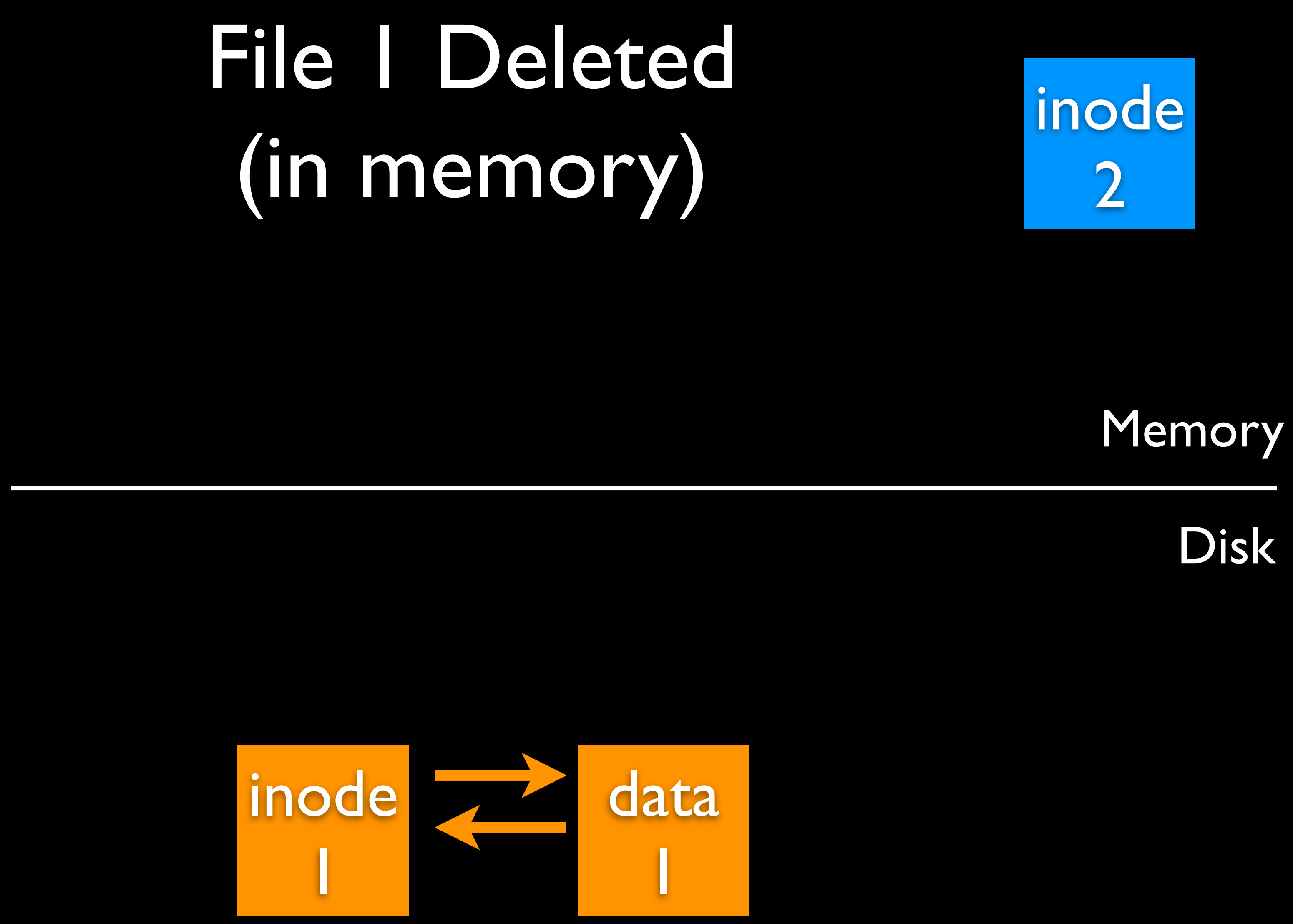


Why Consistency Arises With BBC

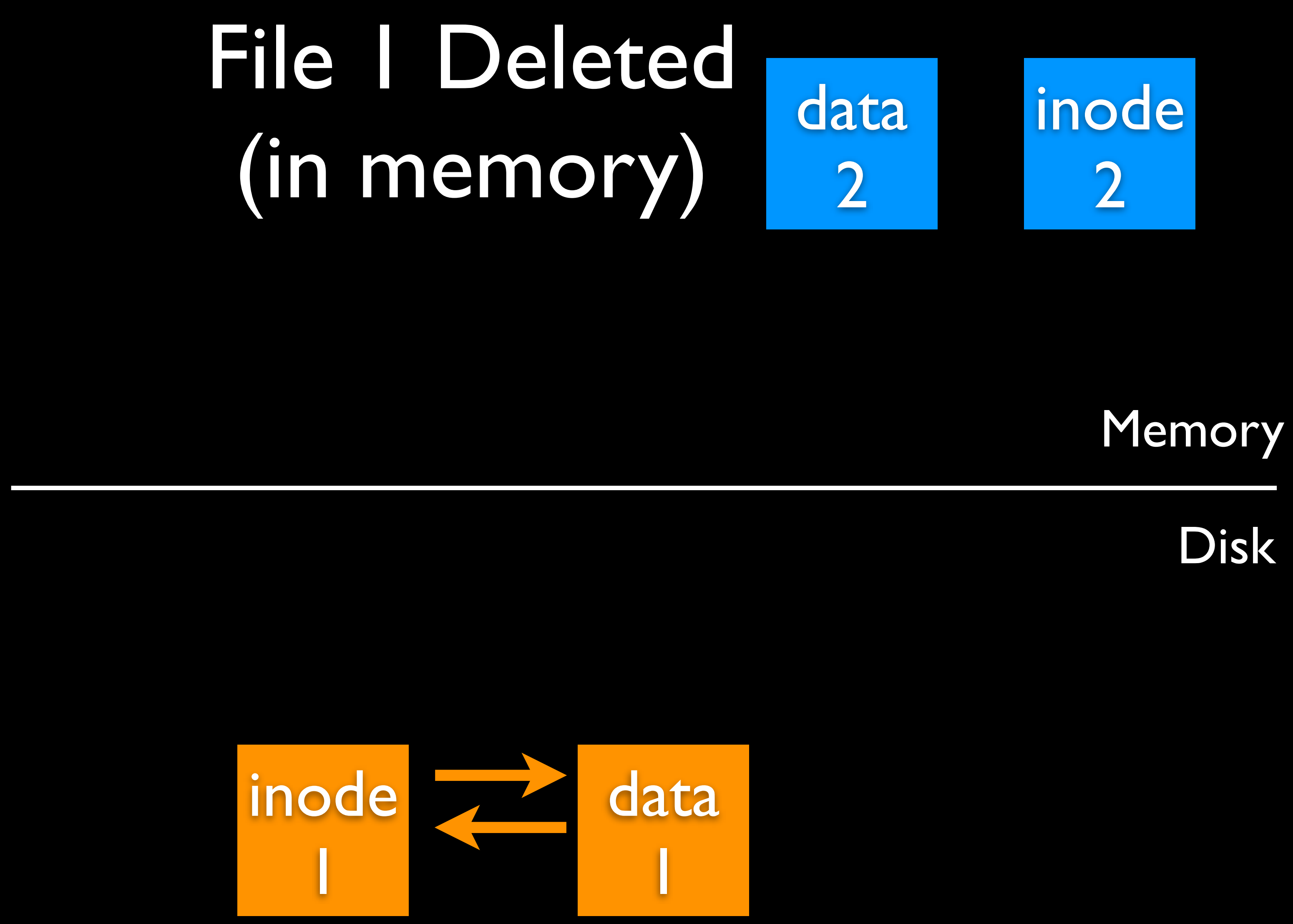
File 1 Deleted
(in memory)



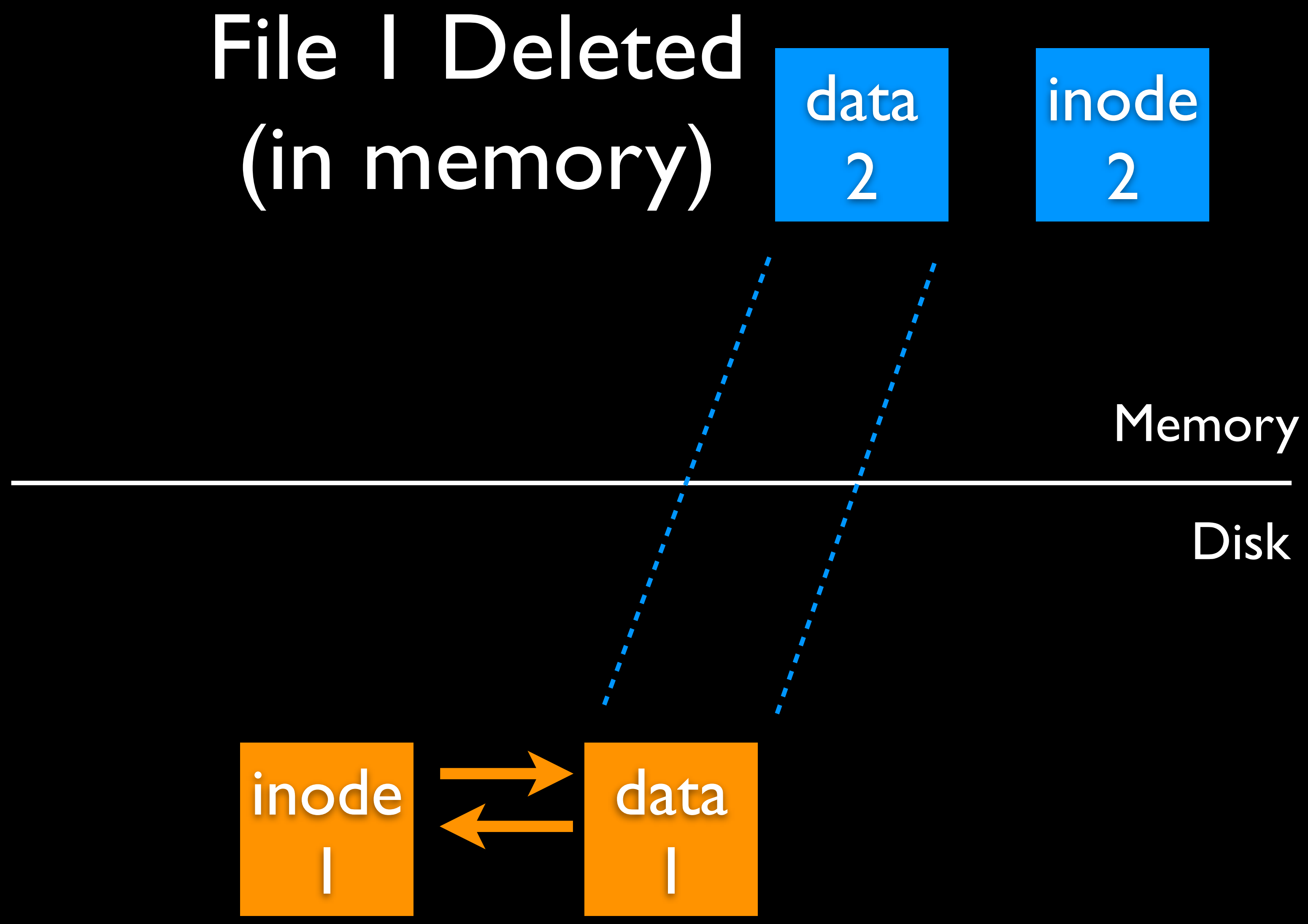
Why Consistency Arises With BBC



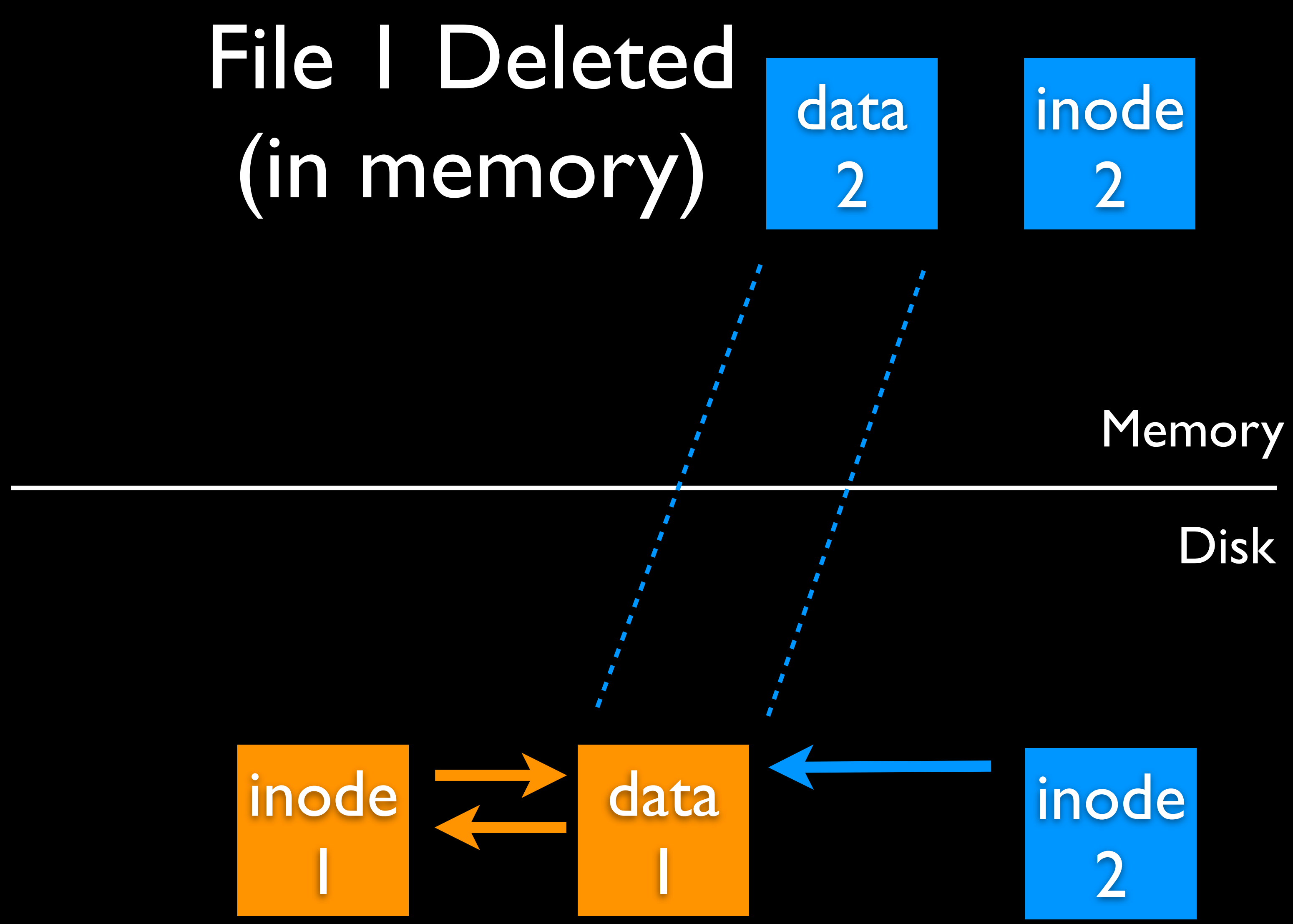
Why Consistency Arises With BBC



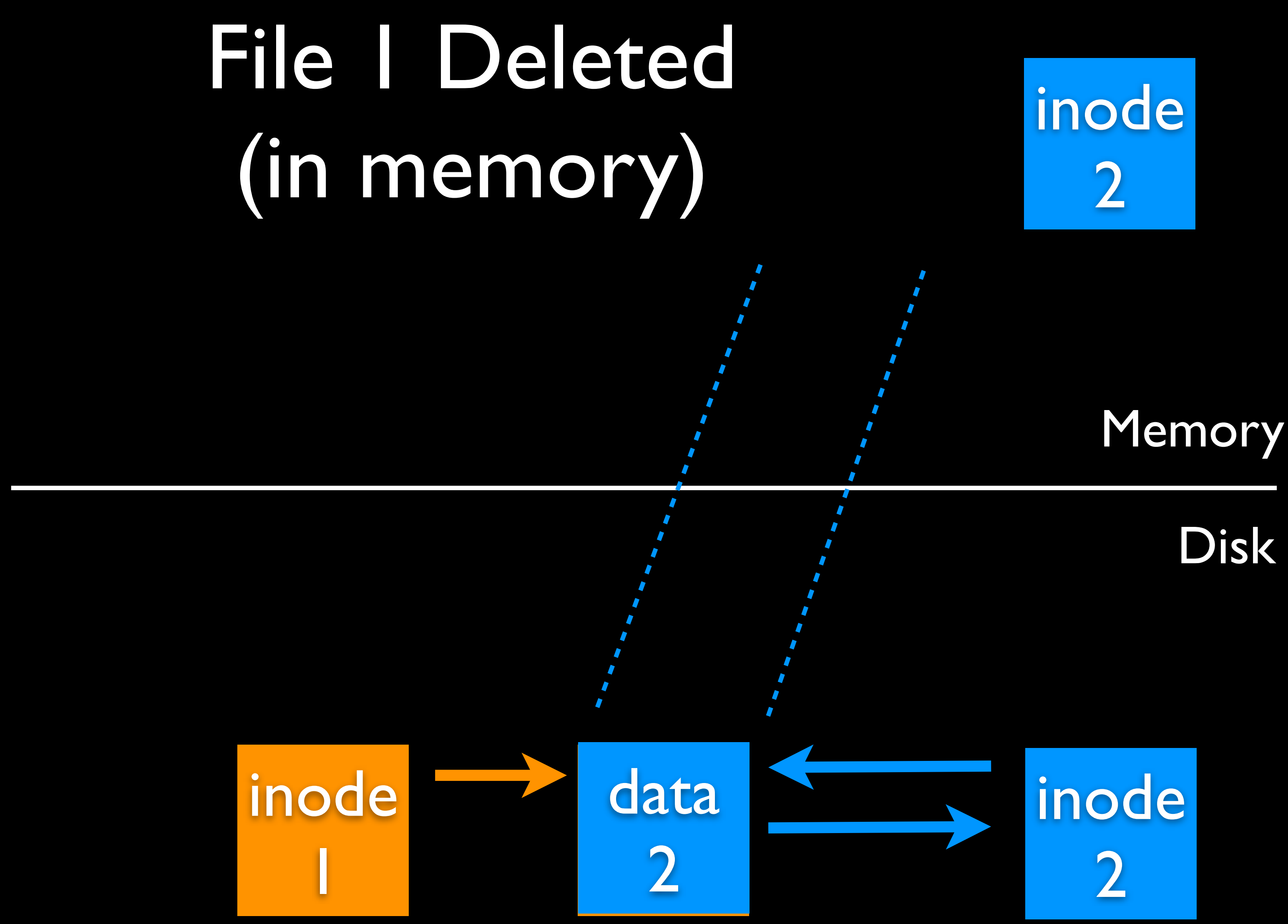
Why Consistency Arises With BBC



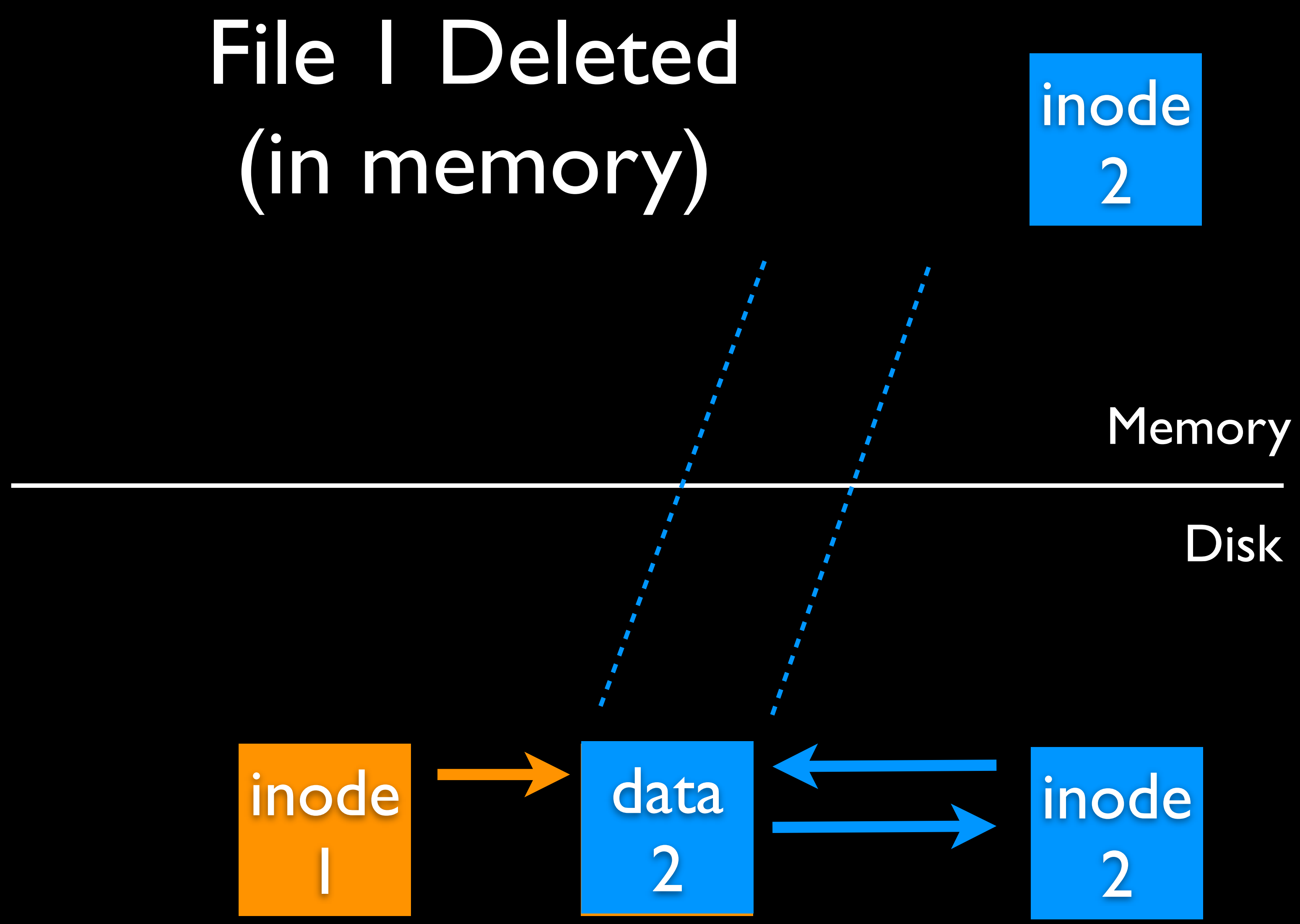
Why Consistency Arises With BBC



Why Consistency Arises With BBC



Why Consistency Arises With BBC



Disk image not consistent, but can detect and repair

When To Repair?

Inode scan (i-scan)

- At mount time, scan on-disk inodes to determine block ownership and build consistent image
- No bitmaps persisted, must assemble!
- Key feature: Done in background

Problem: Inode accessed before i-scan is done

- But all is well: Just check each data block on read() or write() path (slow but consistent)

Similar issues for data-block scan (d-scan) - skipped

NoFS: Outline

~~BBC: Basic idea~~

Implementing NoFS

Results

NoFS Implementation

Basic NoFS:

- Linux ext2 + backpointers
- Fat inodes to accommodate hard links

No pre-mount fsck: Mount immediately

- Just background i-scan and d-scan

Some limitations:

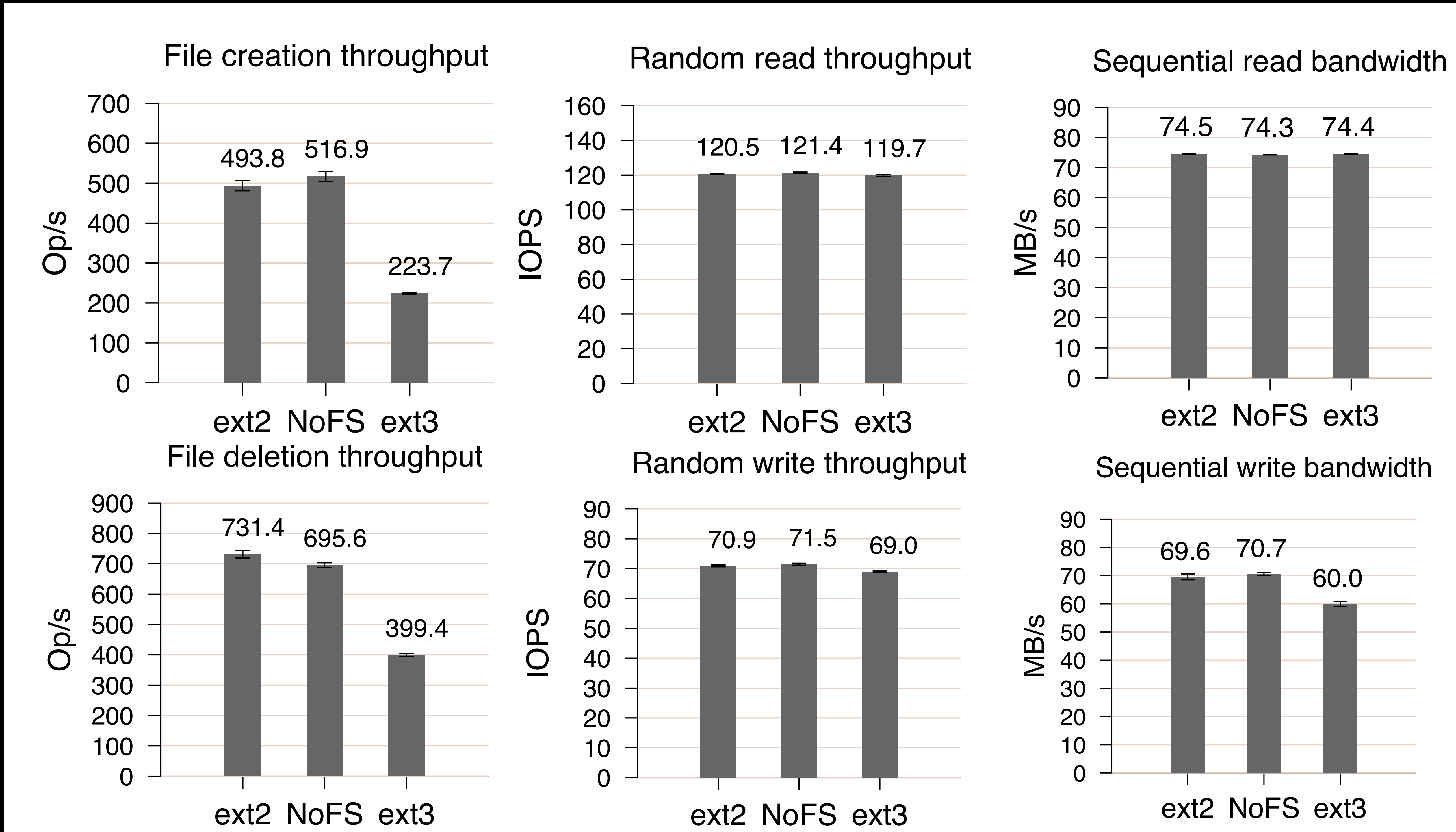
- No transactions (makes rename() weird)
- Lower performance before scans complete (e.g., stat() of unverified inode)
- Assumes 4KB+backpointer atomic write

NoFS: Outline

~~BBC: Basic idea~~

~~Implementing NOFS~~

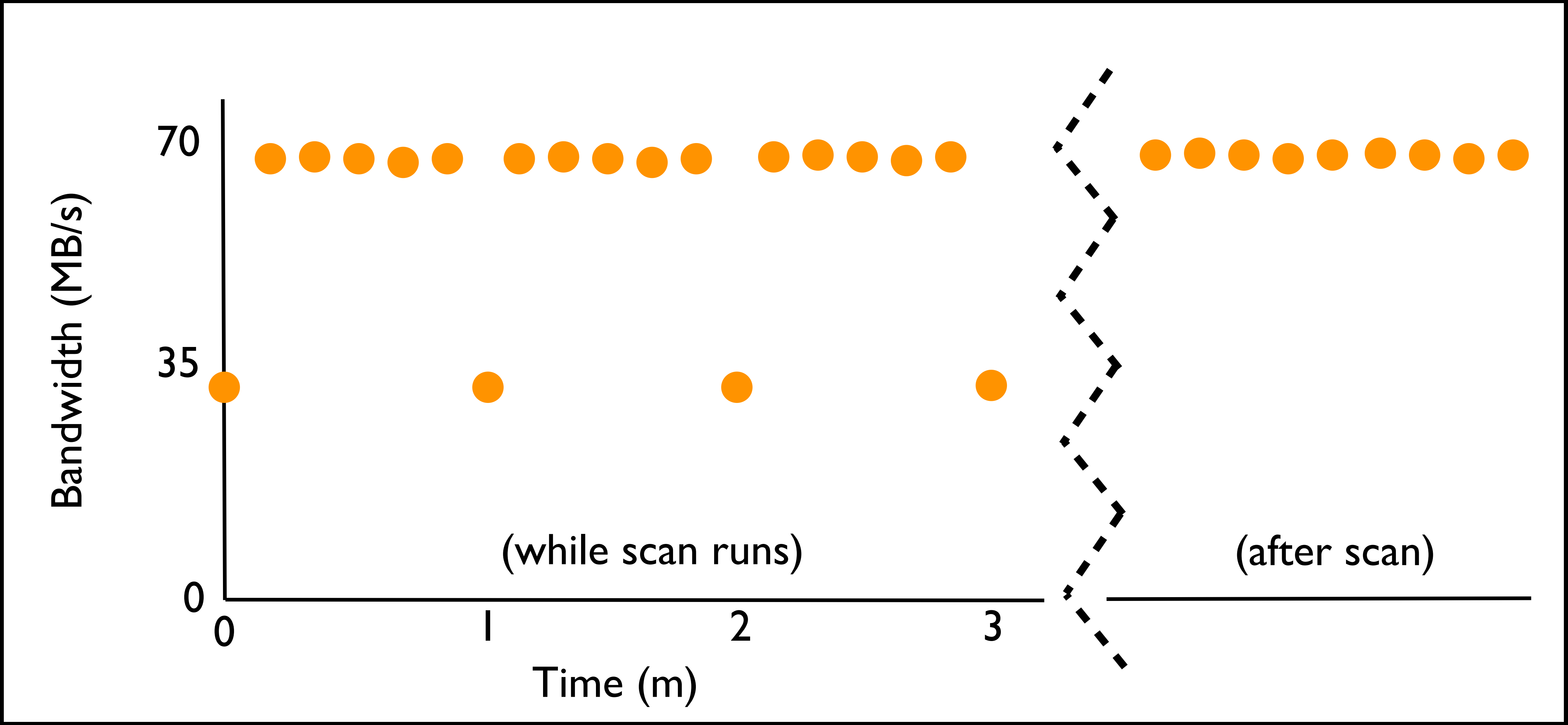
Results



Microbenchmark analysis:

- Performance similar to ext2

Performance



Performance (Periodic Sequential Write):

- Cost felt while periodic scans run
- Later: Scans complete & performance unaffected

Summary

Consistency without ordering

- NoFS: Uses backpointers to provide consistency without trusting disk ordering

Analysis

- Provable consistency guarantees
- Performance is usually good
- Limits: Lack of atomicity, performance during scans

Concluding Thoughts

"The fast drives out the slow, even if the fast is wrong"
W. Kahan

Summary

Modern disks

- The "fast" thing is to report success, even if write has not reached disk
- Formalized as **weak durability**

What we did

- Coerce the cache in a Discreet FS
- Avoid need for ordering with NoFS

Main goal: Build working file systems despite the presence of weak durability

Directions

Directions

Low-level interfaces

- e.g., tell me when, not force me now
- e.g., informed read
- Are other interfaces less amenable to cheating by device vendor?

Directions

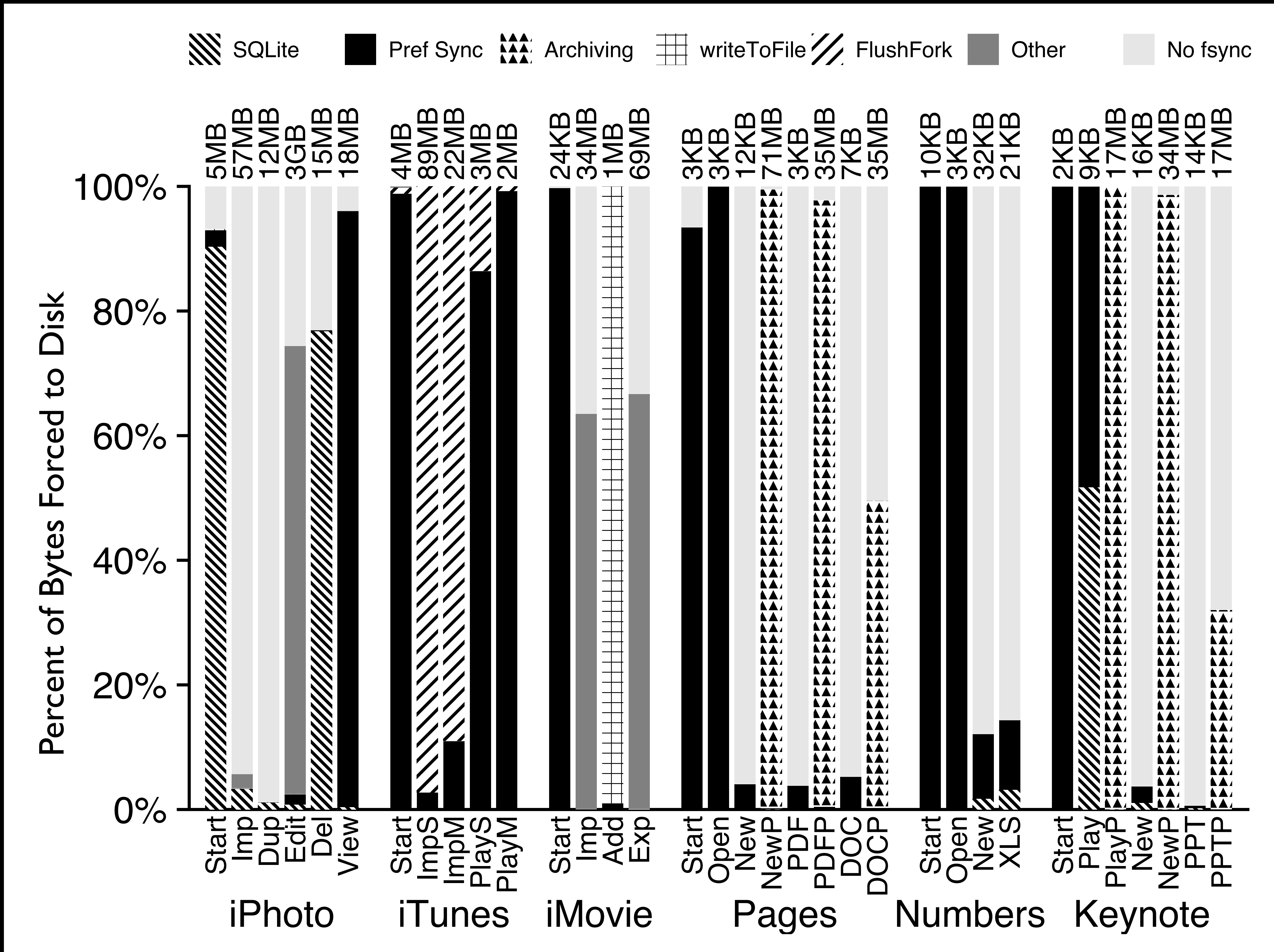
Low-level interfaces

- e.g., tell me when, not force me now
- e.g., informed read
- Are other interfaces less amenable to cheating by device vendor?

High-level interfaces

- The problem with fsync()

Fsync() is common!



Directions

Low-level interfaces

- e.g., tell me when, not force me now
- e.g., informed read
- Are other interfaces less amenable to cheating by device vendor?

High-level interfaces

- The problem with fsync()
- Real goal: Understand what applications actually need, instead of just build the same POSIX file system again

Thanks!

Thanks!

Led by Professors
Andrea Arpaci-Dusseau and
Remzi Arpaci-Dusseau

Thanks!

Led by Professors
Andrea Arpaci-Dusseau and
Remzi Arpaci-Dusseau

Real work done by:
Vijay Chidambaran, Deepak Ramamurthi, Yupu Zhang, Abhishek Rajimwale, Tyler Harter, Chris Dragga, Mike Vaughn, Lakshmi Bairavasundaram
[papers at DSN '11, FAST '12, Sigmetrics '07, FAST '08, and SOSP '11]

Thanks!

Led by Professors
Andrea Arpaci-Dusseau and
Remzi Arpaci-Dusseau

Real work done by:
Vijay Chidambaran, Deepak Ramamurthi, Yupu Zhang, Abhishek Rajimwale, Tyler Harter, Chris Dragga, Mike Vaughn, Lakshmi Bairavasundaram [papers at DSN '11, FAST '12, Sigmetrics '07, FAST '08, and SOSP '11]

More @ www.cs.wisc.edu/adsl