Optimistic Crash Consistency

<u>Vijay Chidambaram</u>
Thanumalayan Sankaranarayana Pillai
Andrea Arpaci-Dusseau
Remzi Arpaci-Dusseau

UNIVERSITY OF WISCONSIN–MADISON

WISDOM

Crash Consistency Problem

Single file-system operation updates multiple on-disk data structures

System may crash in middle of updates

File-system is partially (incorrectly) updated

Performance OR Consistency

Crash-consistency solutions degrade performance

Users forced to choose between high performance and strong consistency

- Performance differs by 10x for some workloads

Many users choose performance

- ext3 default configuration did not guarantee crash consistency for many years
- Mac OSX `fsync()` does not ensure data is safe

"*The Fast drives out the Slow even if the Fast is wrong*"

- Kahan

Ordering and Durability

Crash consistency is built upon ordered writes

File systems conflate ordering and durability

- Ideal: {A, B} -> {C} (made durable later)
- Current scenario
  - {A, B} durable
  - {C} durable

Inefficient when only ordering is required

Can a file system provide
both
high performance
and strong consistency?

Is there a middle ground between:
high performance but no consistency
strong consistency but low performance?

Our solution
Optimistic File System (OptFS)

Journaling file system that provides performance and consistency by decoupling ordering and durability

Such decoupling allows OptFS to trade freshness for performance while maintaining crash consistency

Results

Techniques: checksums, delayed writes, etc.

OptFS provides strong consistency

- Equivalent to ext4 data journaling

OptFS improves performance significantly

- 10x better than ext4 on some workloads

New primitive osync() provides ordering among writes at high performance

Outline

Introduction

Ordering and Durability in Journaling

Optimistic File System

Results

Conclusion

Outline

Introduction

Ordering and Durability in Journaling

- Journaling Overview
- Realizing Ordering on Disks
- Journaling without Ordering

Optimistic File System

Results

Conclusion

Journaling Overview

Before updating file system, write note describing update

Make sure note is safely on disk

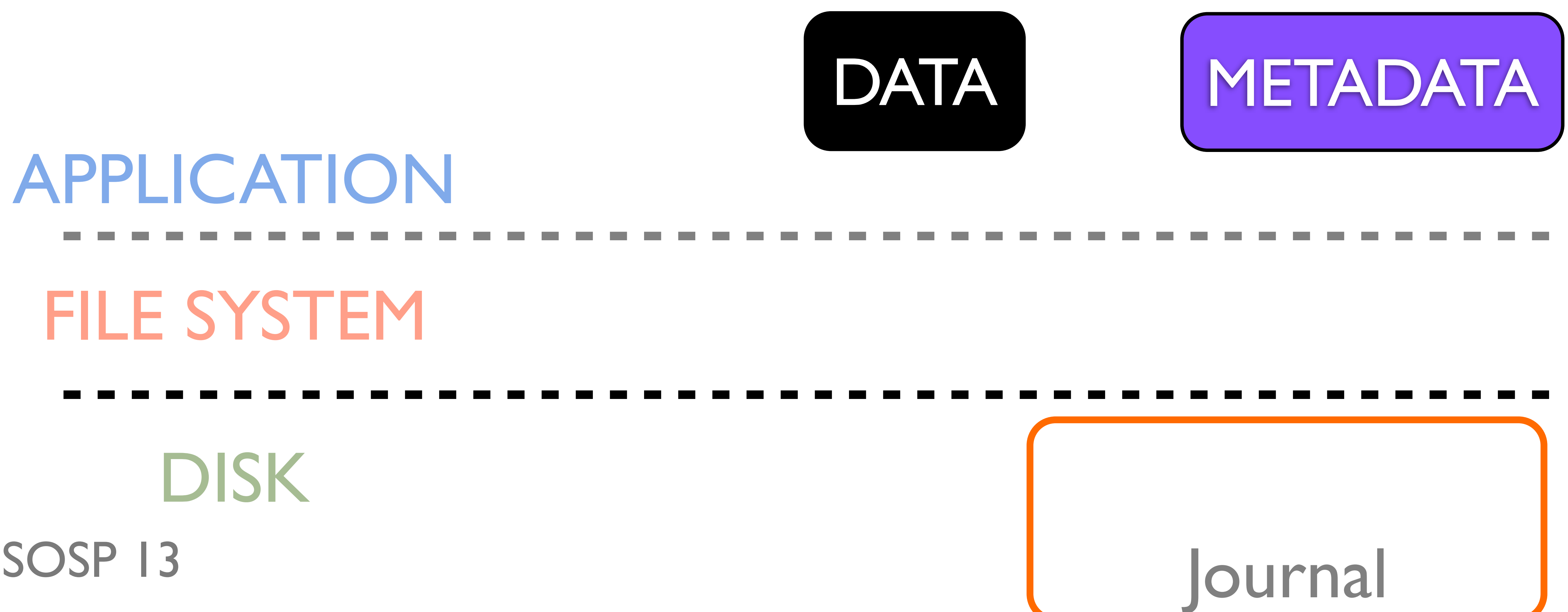
Once note is safe, update file system

- If interrupted, read note and redo updates

Journaling Overview

Workload: Creating and writing to a file

Journaling protocol (ordered journaling)

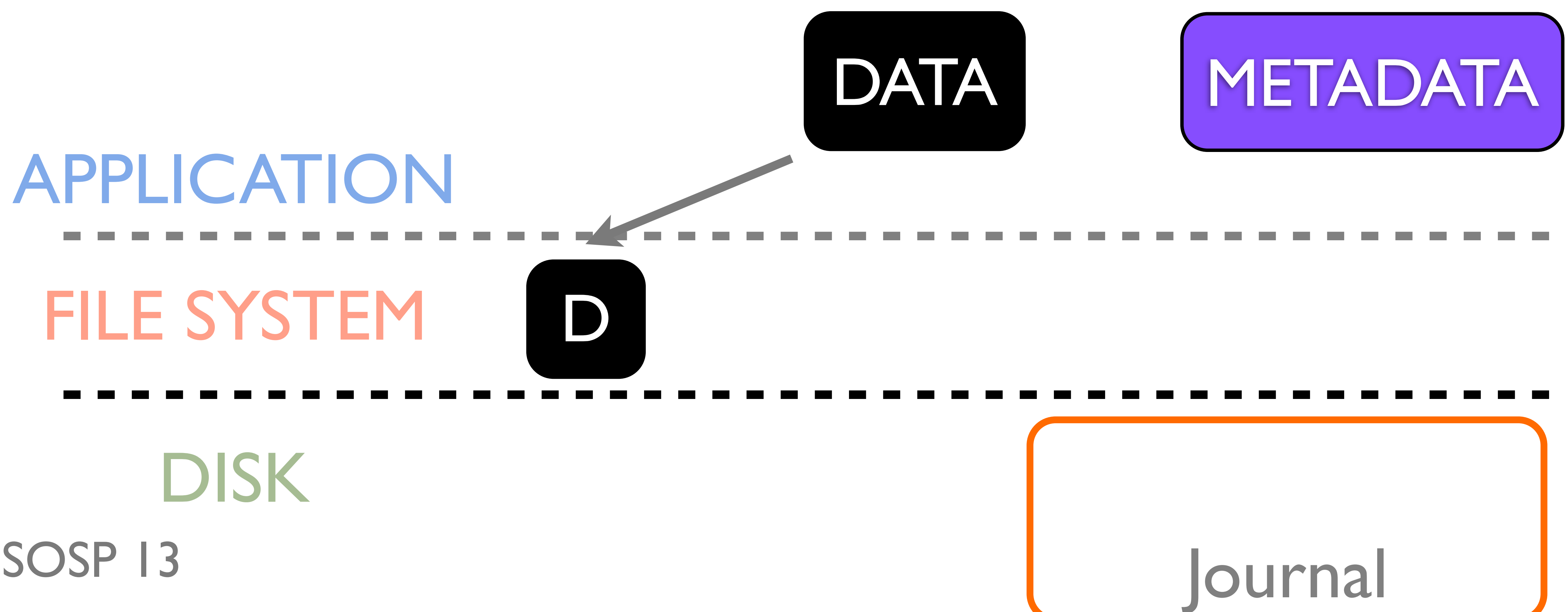


Journaling Overview

Workload: Creating and writing to a file

Journaling protocol (ordered journaling)

- Data write (D)

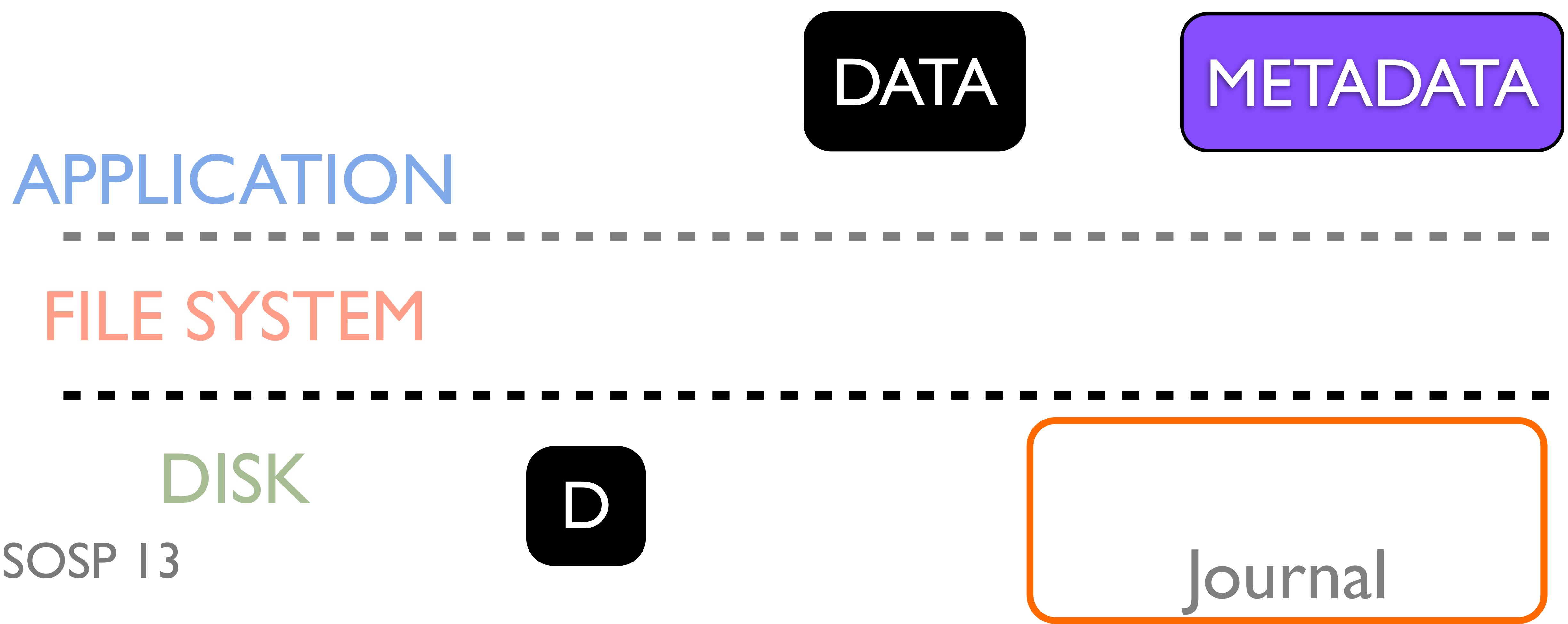


Journaling Overview

Workload: Creating and writing to a file

Journaling protocol (ordered journaling)

- Data write (D)

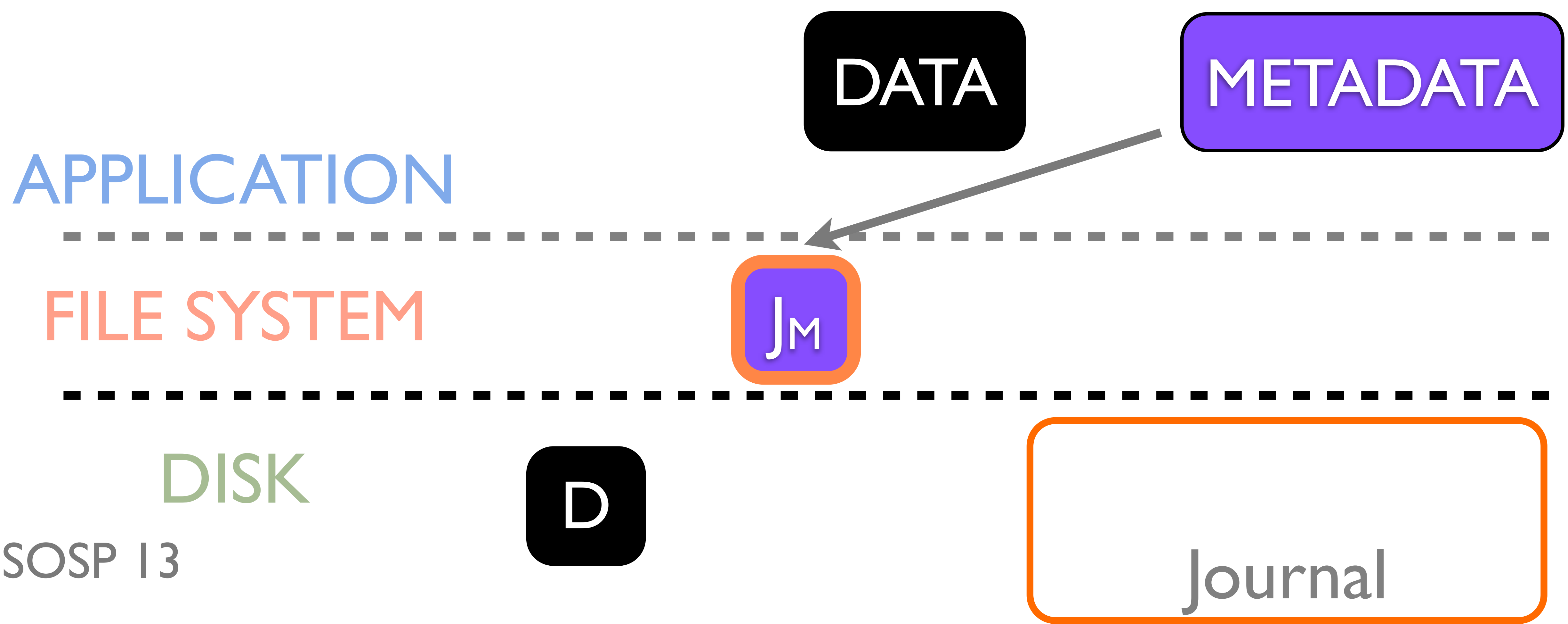


Journaling Overview

Workload: Creating and writing to a file

Journaling protocol (ordered journaling)

- Data write (D)
- Logging Metadata (J_M)

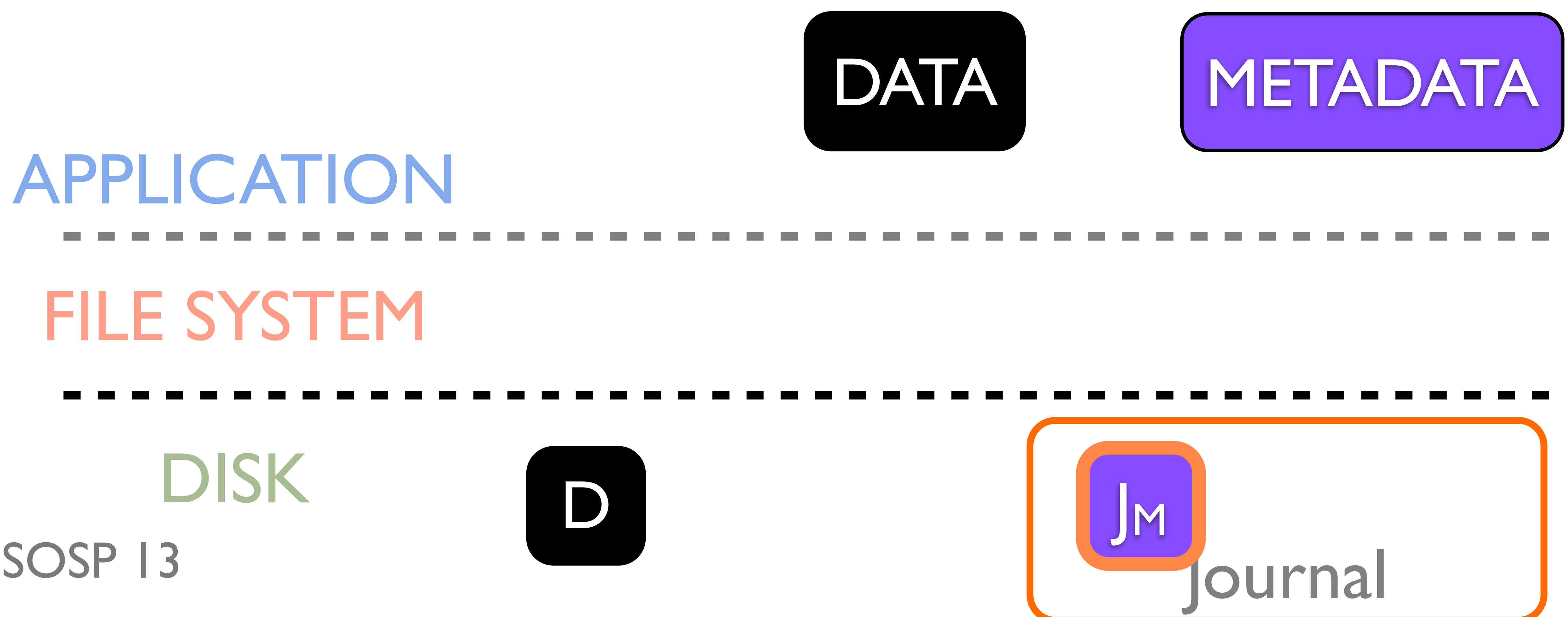


Journaling Overview

Workload: Creating and writing to a file

Journaling protocol (ordered journaling)

- Data write (D)
- Logging Metadata (J_M)

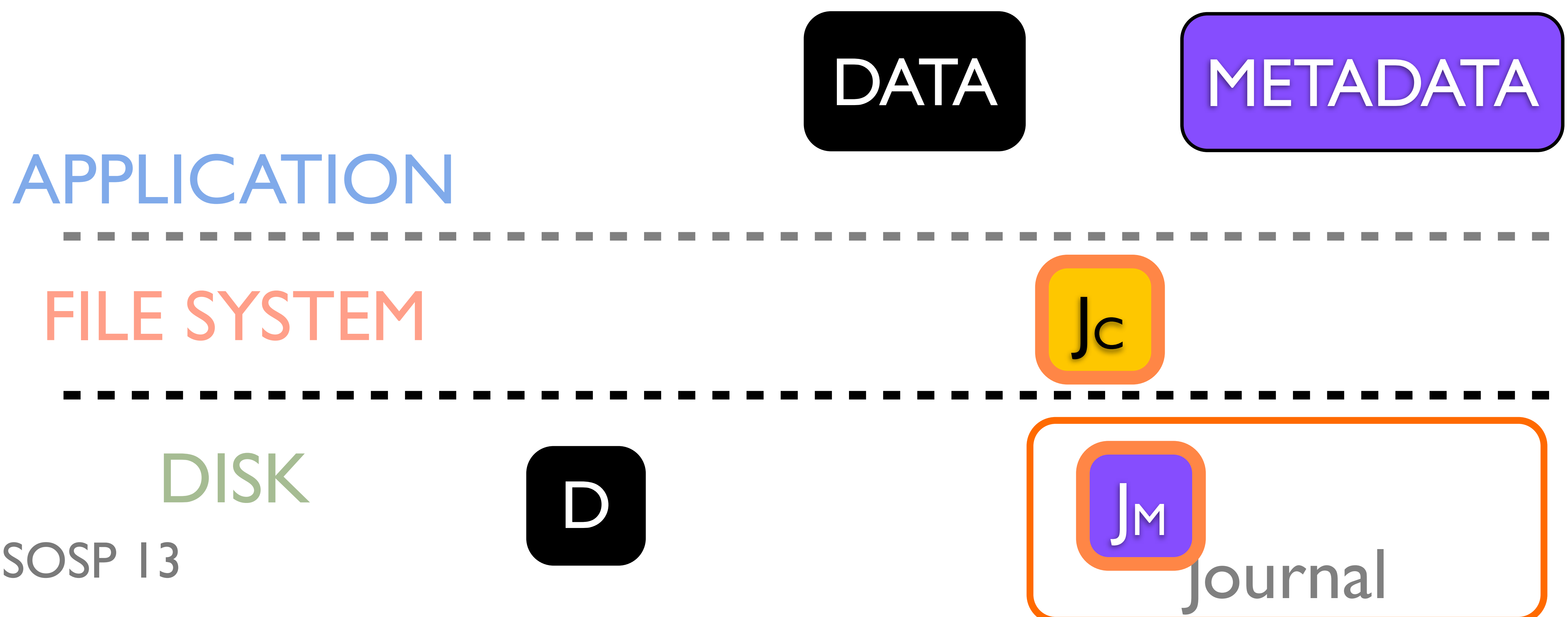


Journaling Overview

Workload: Creating and writing to a file

Journaling protocol (ordered journaling)

- Data write (D)
- Logging Metadata (J_M)
- Logging Commit (J_C)

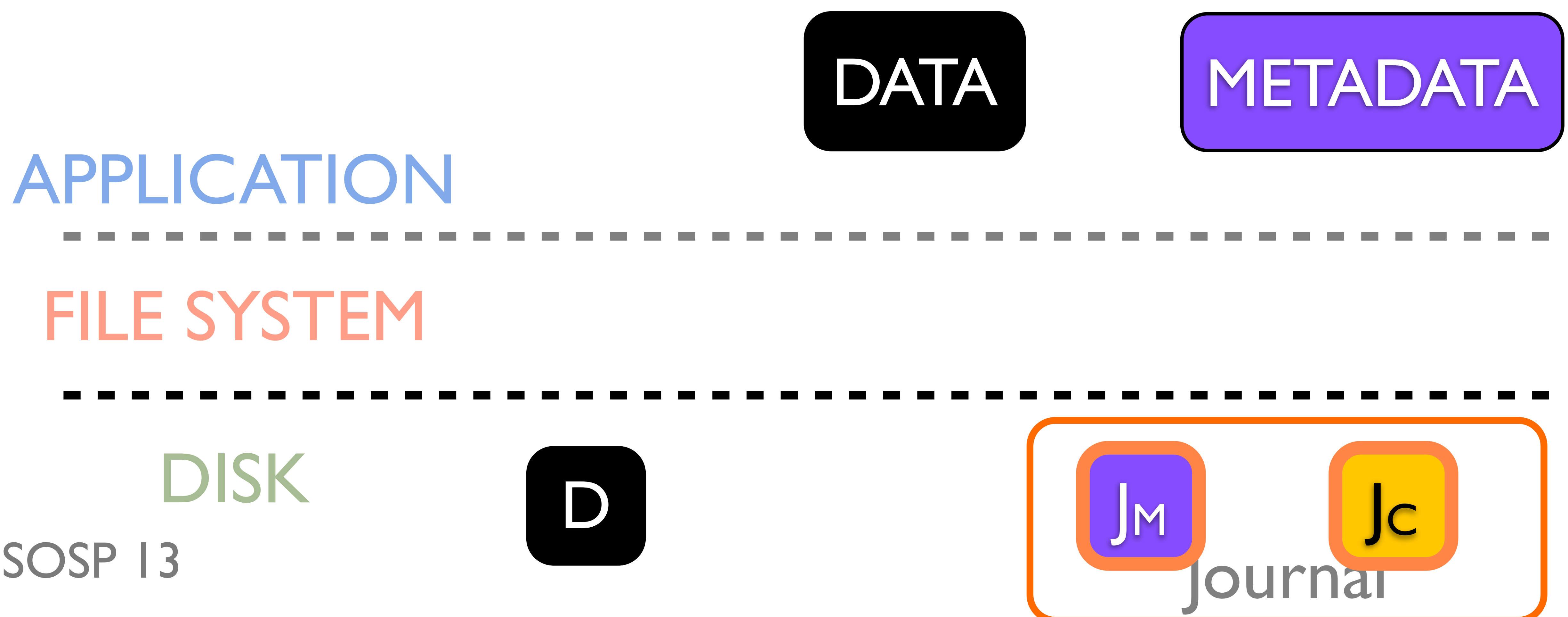


Journaling Overview

Workload: Creating and writing to a file

Journaling protocol (ordered journaling)

- Data write (D)
- Logging Metadata (J_M)
- Logging Commit (J_C)

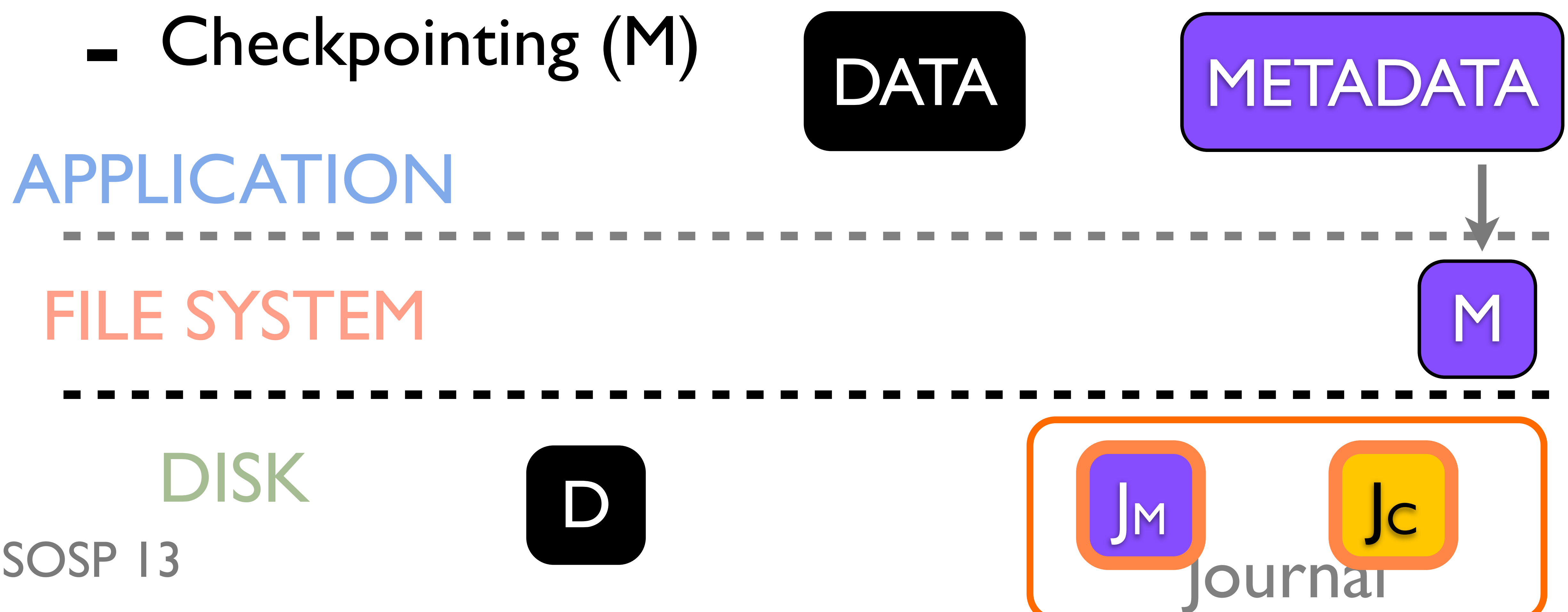


Journaling Overview

Workload: Creating and writing to a file

Journaling protocol (ordered journaling)

- Data write (D)
- Logging Metadata (J_M)
- Logging Commit (J_C)
- Checkpointing (M)

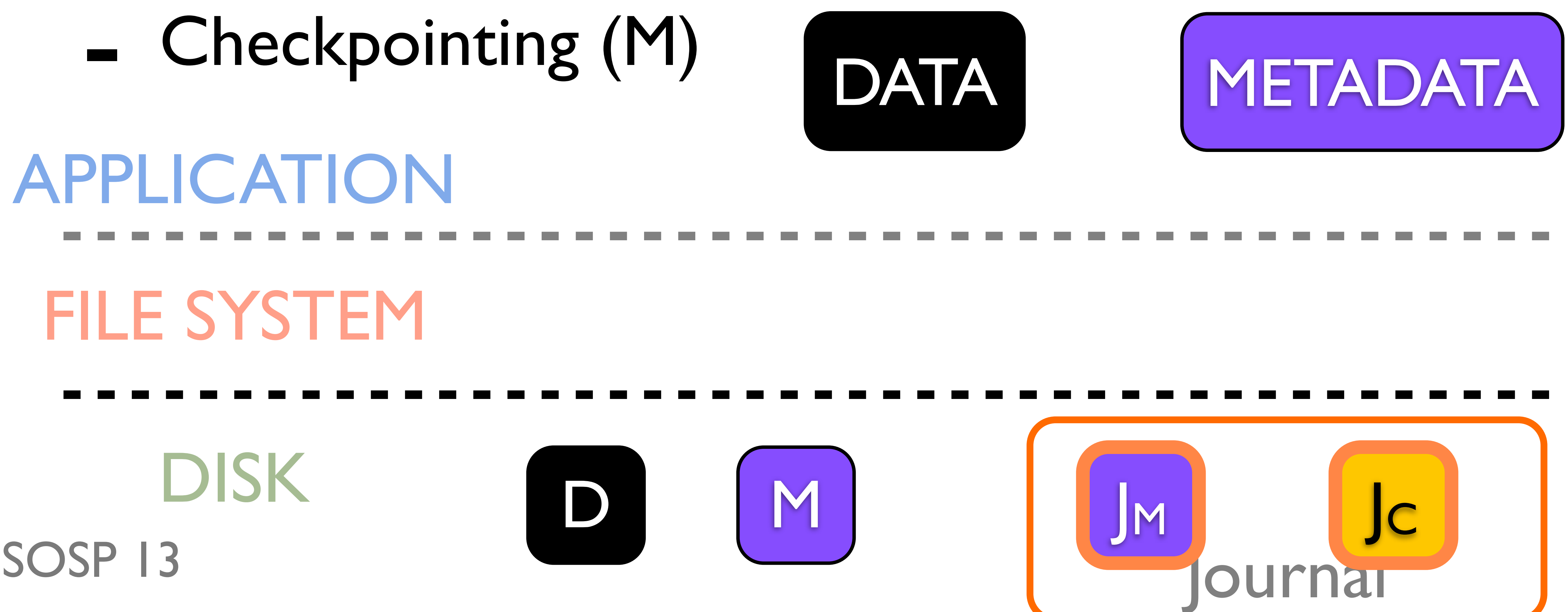


Journaling Overview

Workload: Creating and writing to a file

Journaling protocol (ordered journaling)

- Data write (D)
- Logging Metadata (J_M)
- Logging Commit (J_C)
- Checkpointing (M)



Outline

Introduction

Ordering and Durability in Journaling

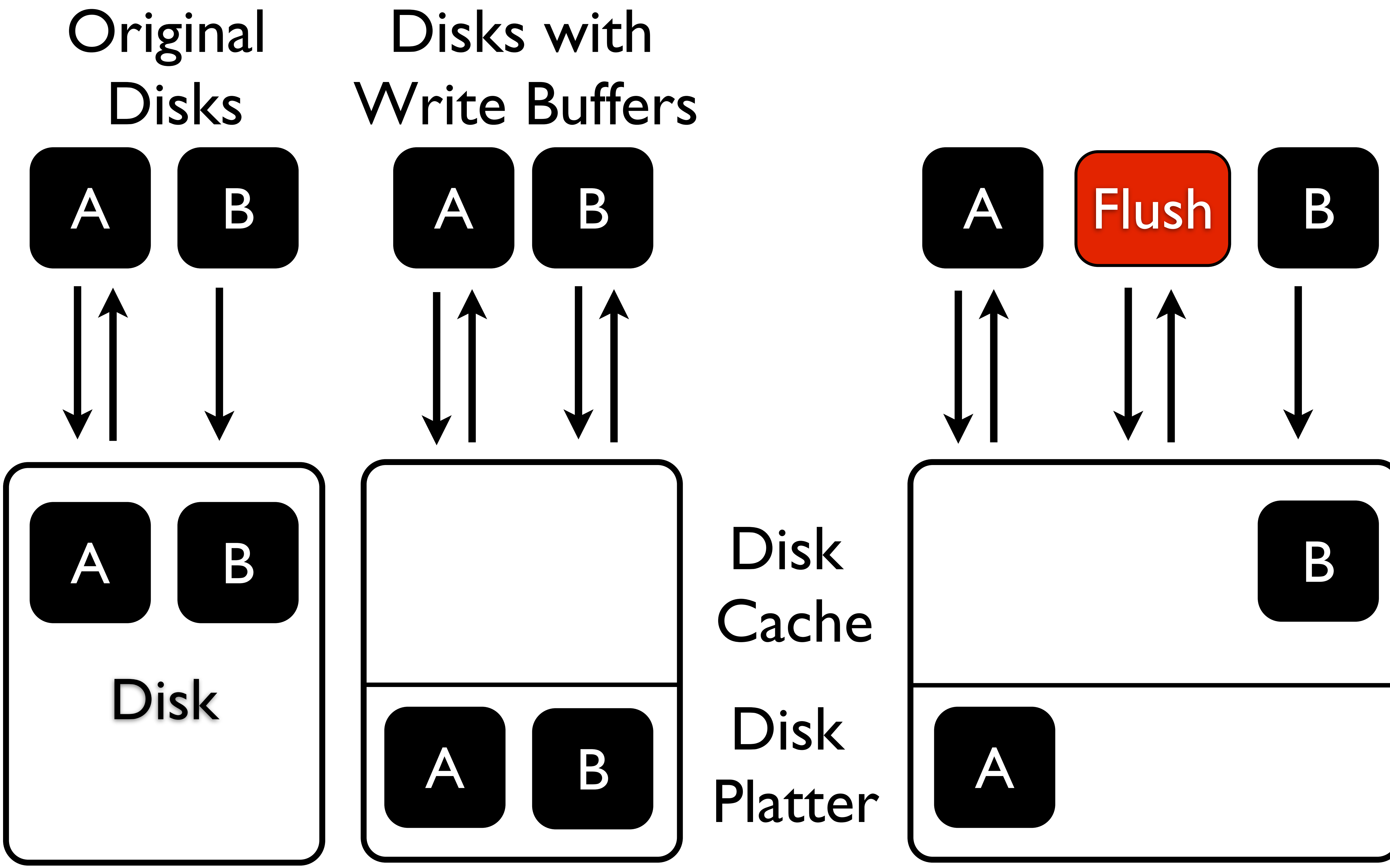
- Journaling Overview
- Realizing Ordering on Disks
- Journaling without Ordering

Optimistic File System

Results

Conclusion

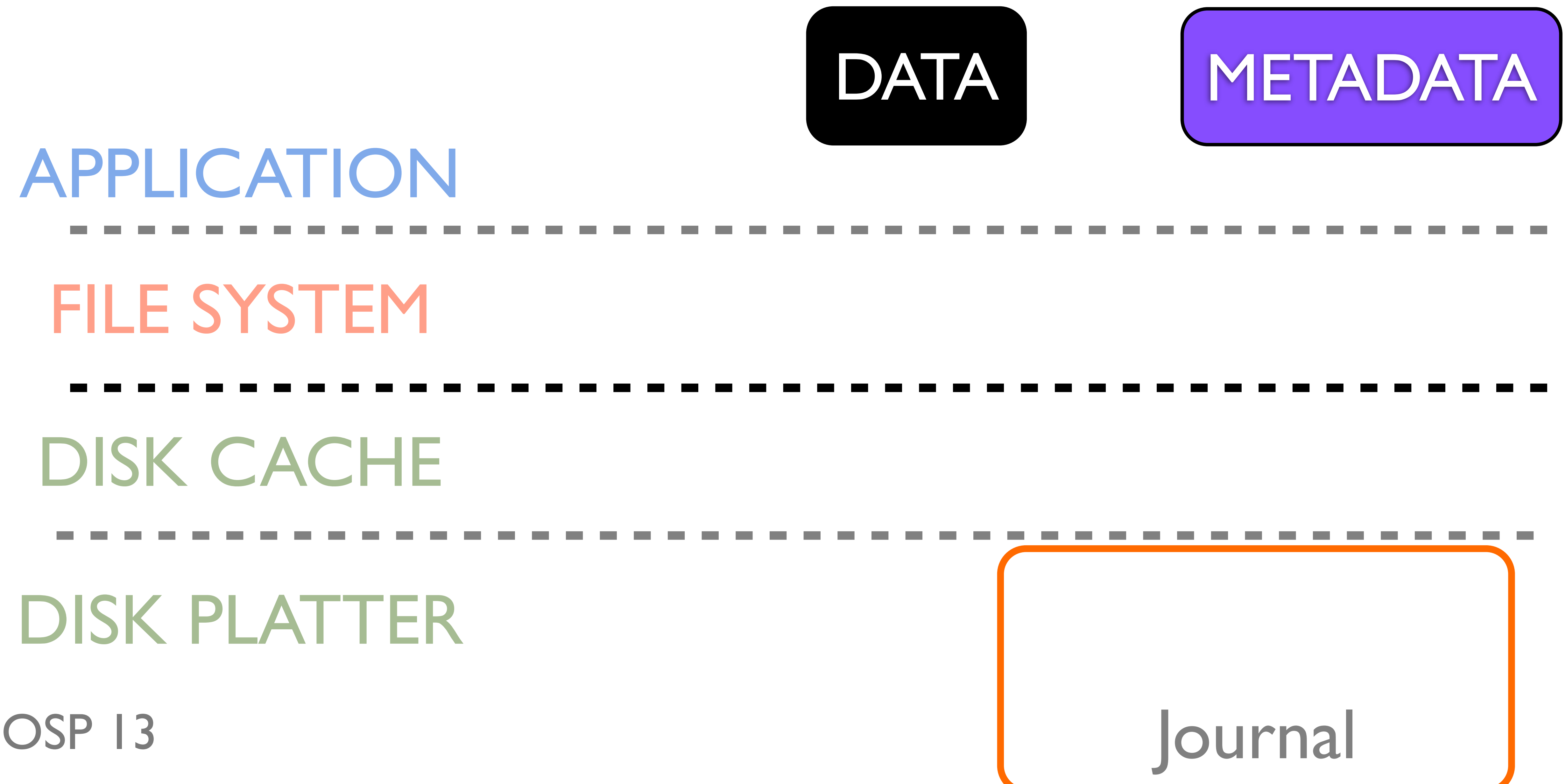
How Writes are Ordered



Journaling with Flushes

Journaling protocol

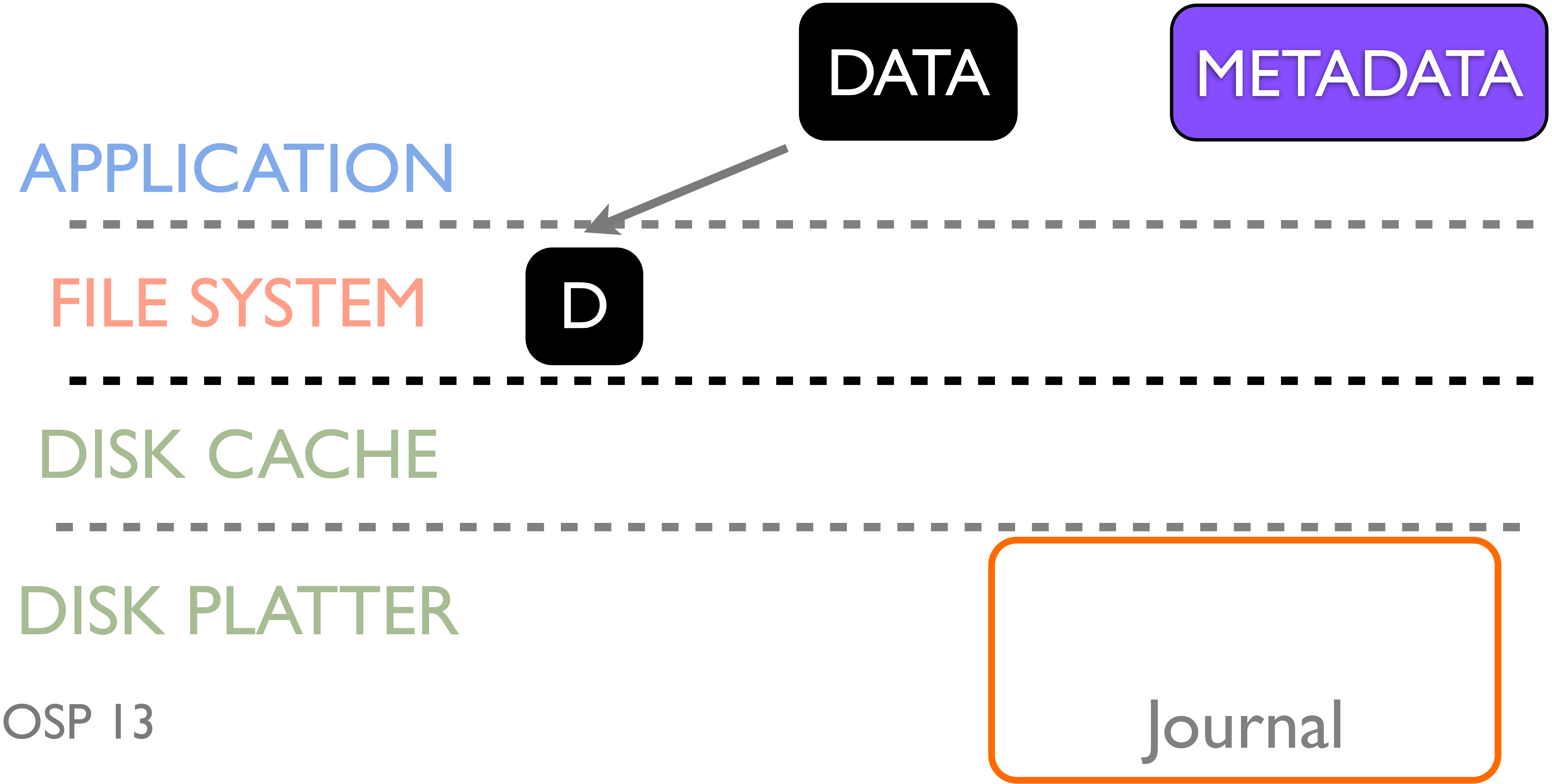
- Data write (D)



Journaling with Flushes

Journaling protocol

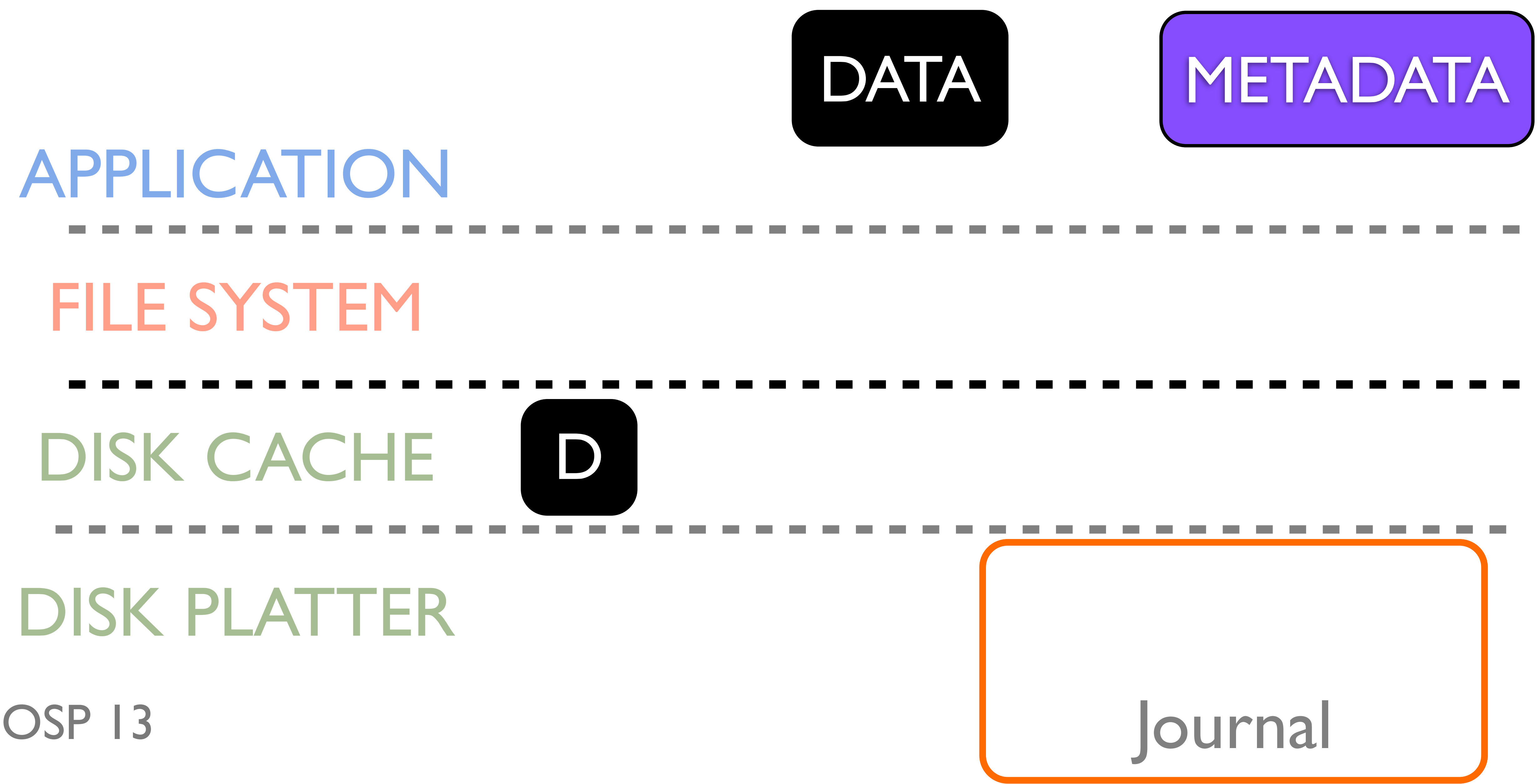
- Data write (D)



Journaling with Flushes

Journaling protocol

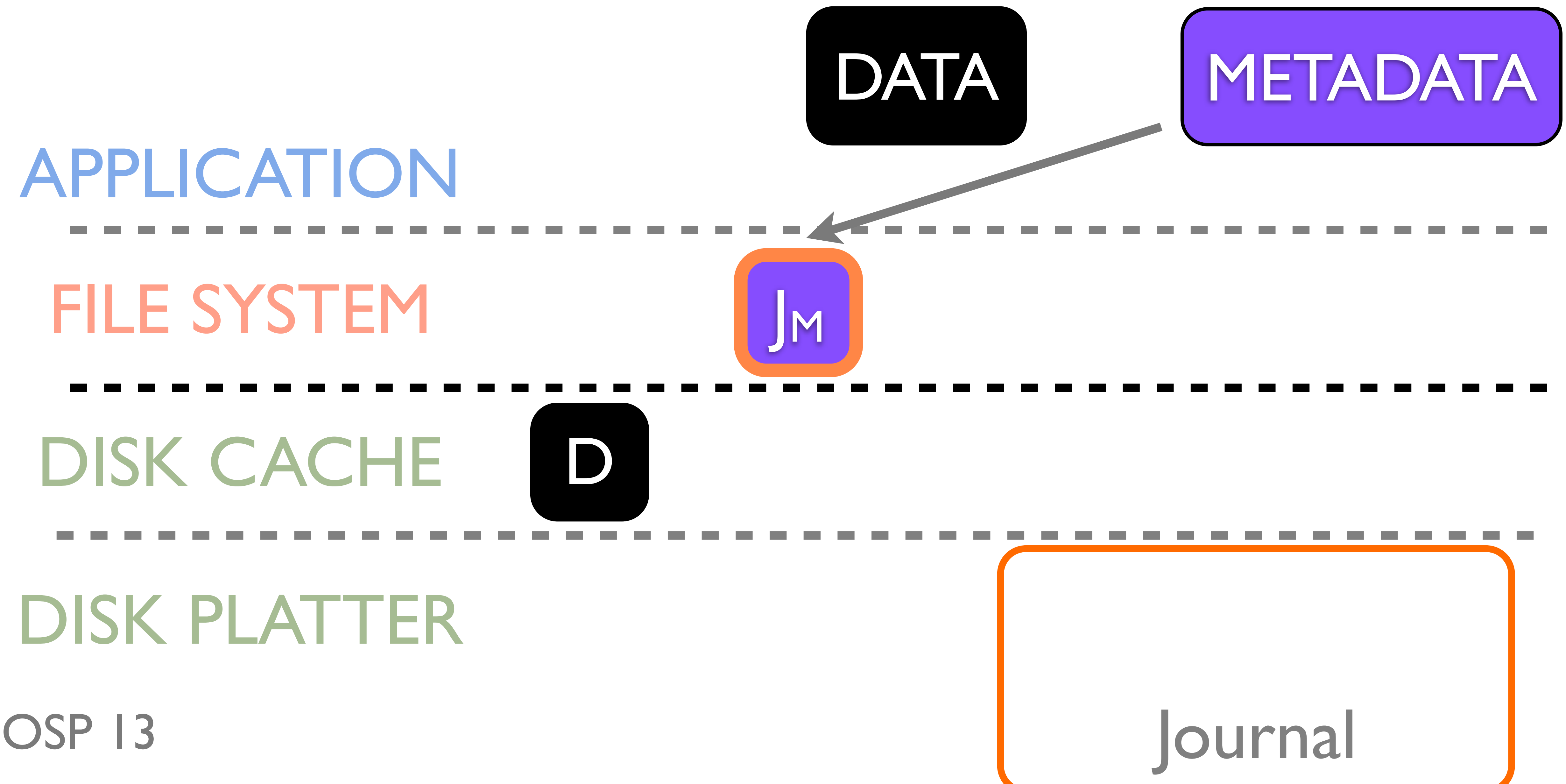
- Data write (D)



Journaling with Flushes

Journaling protocol

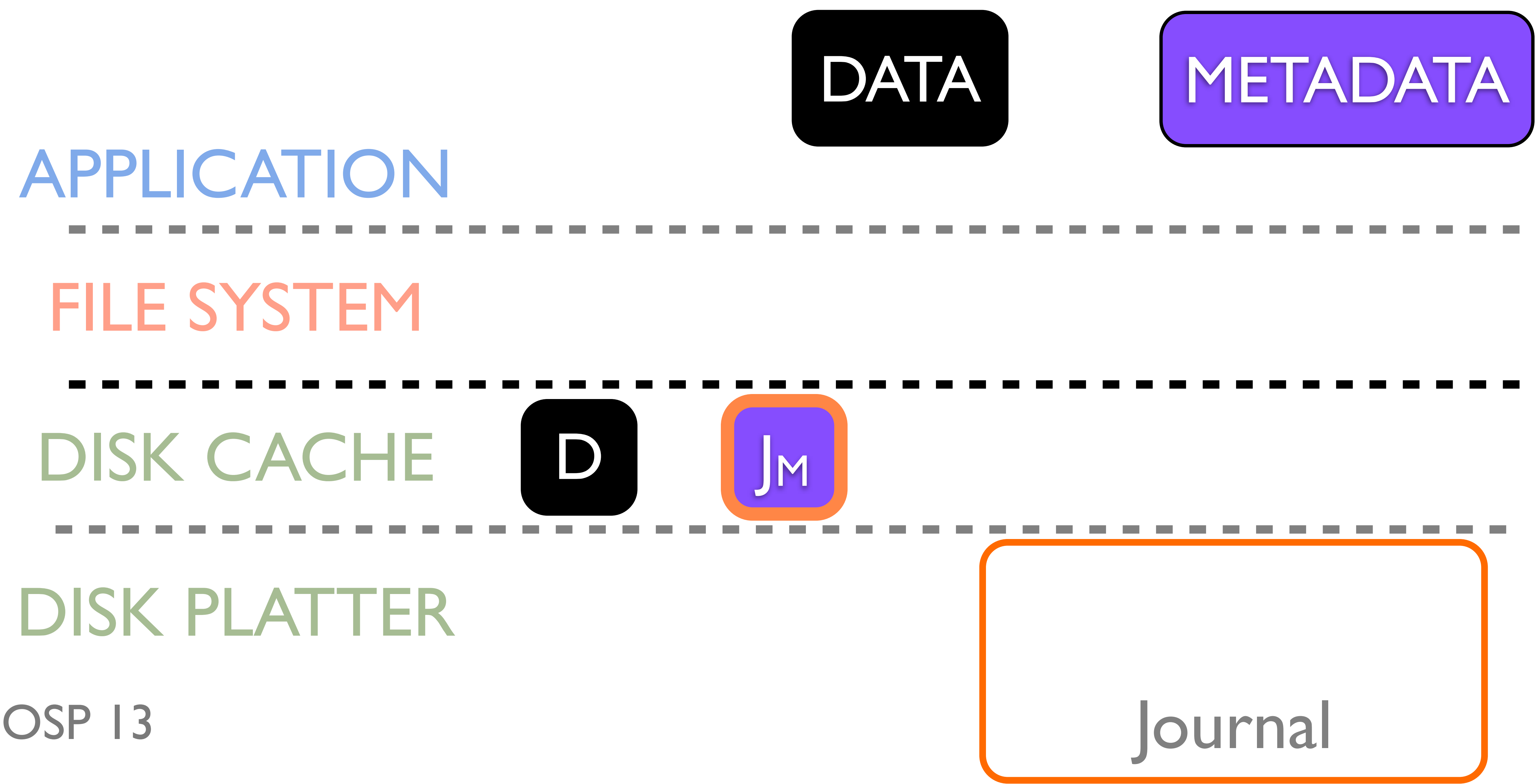
- Data write (D)
- Logging Metadata (J_M)



Journaling with Flushes

Journaling protocol

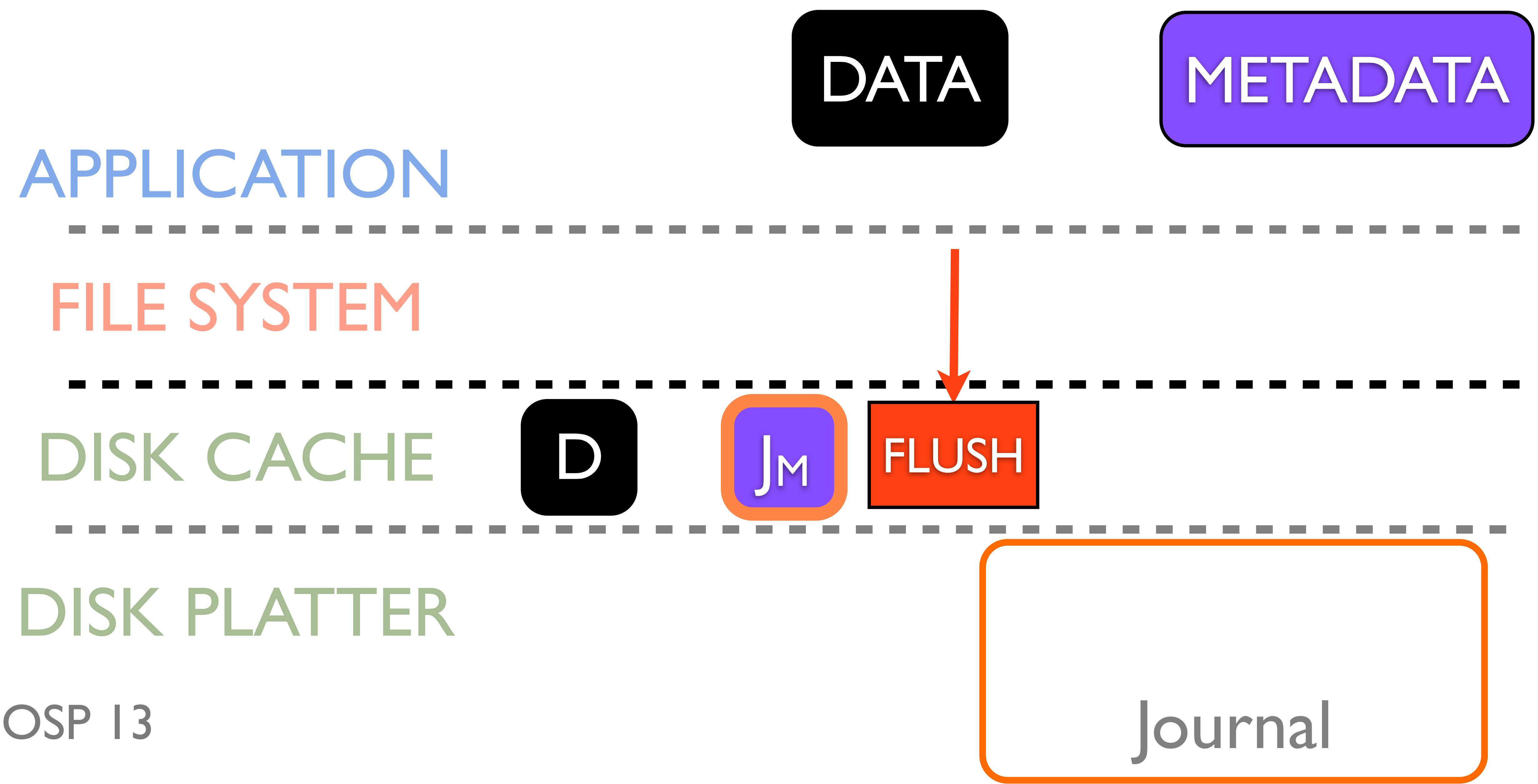
- Data write (D)
- Logging Metadata (J_M)



Journaling with Flushes

Journaling protocol

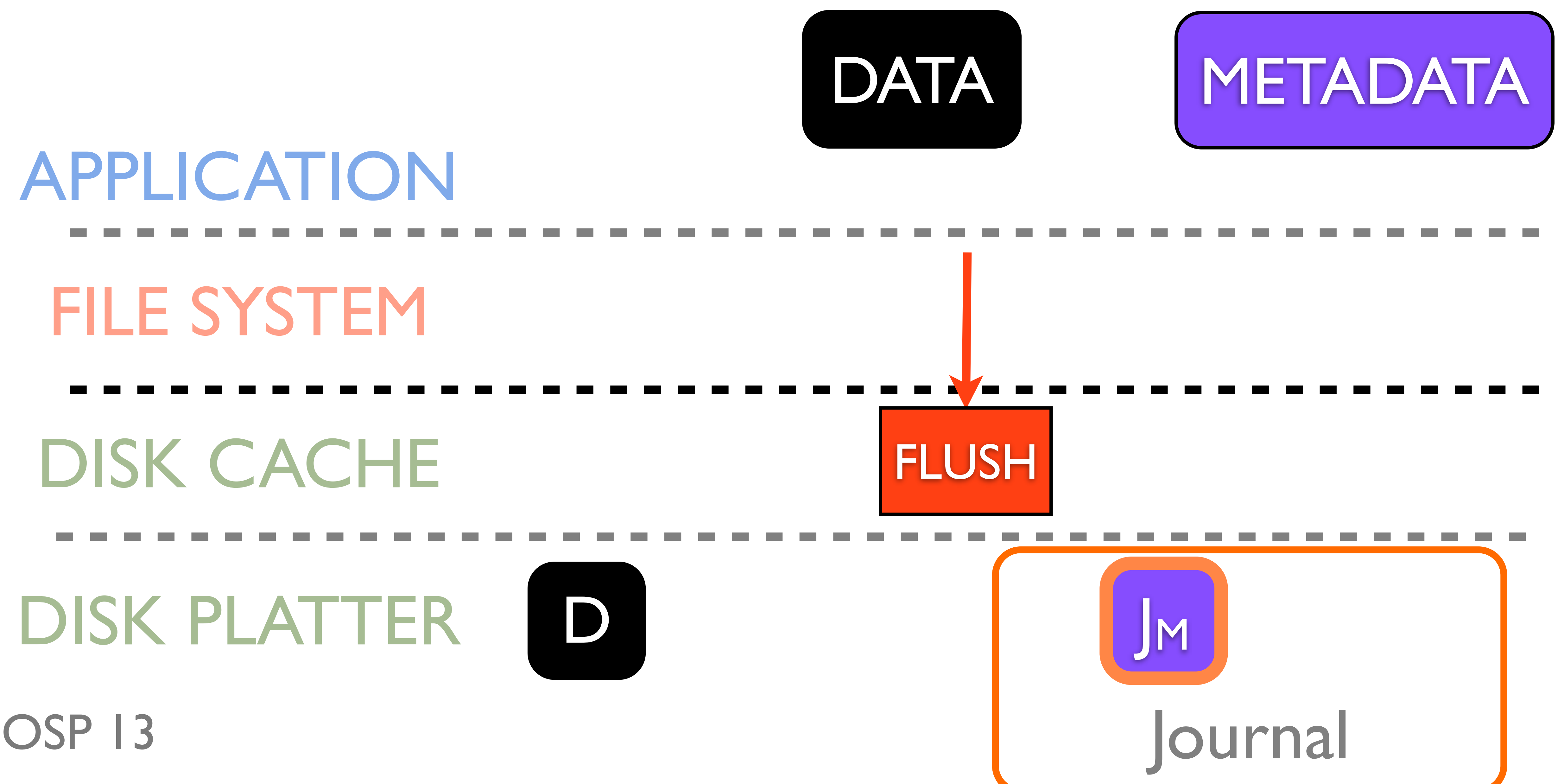
- Data write (D)
- Logging Metadata (J_M)



Journaling with Flushes

Journaling protocol

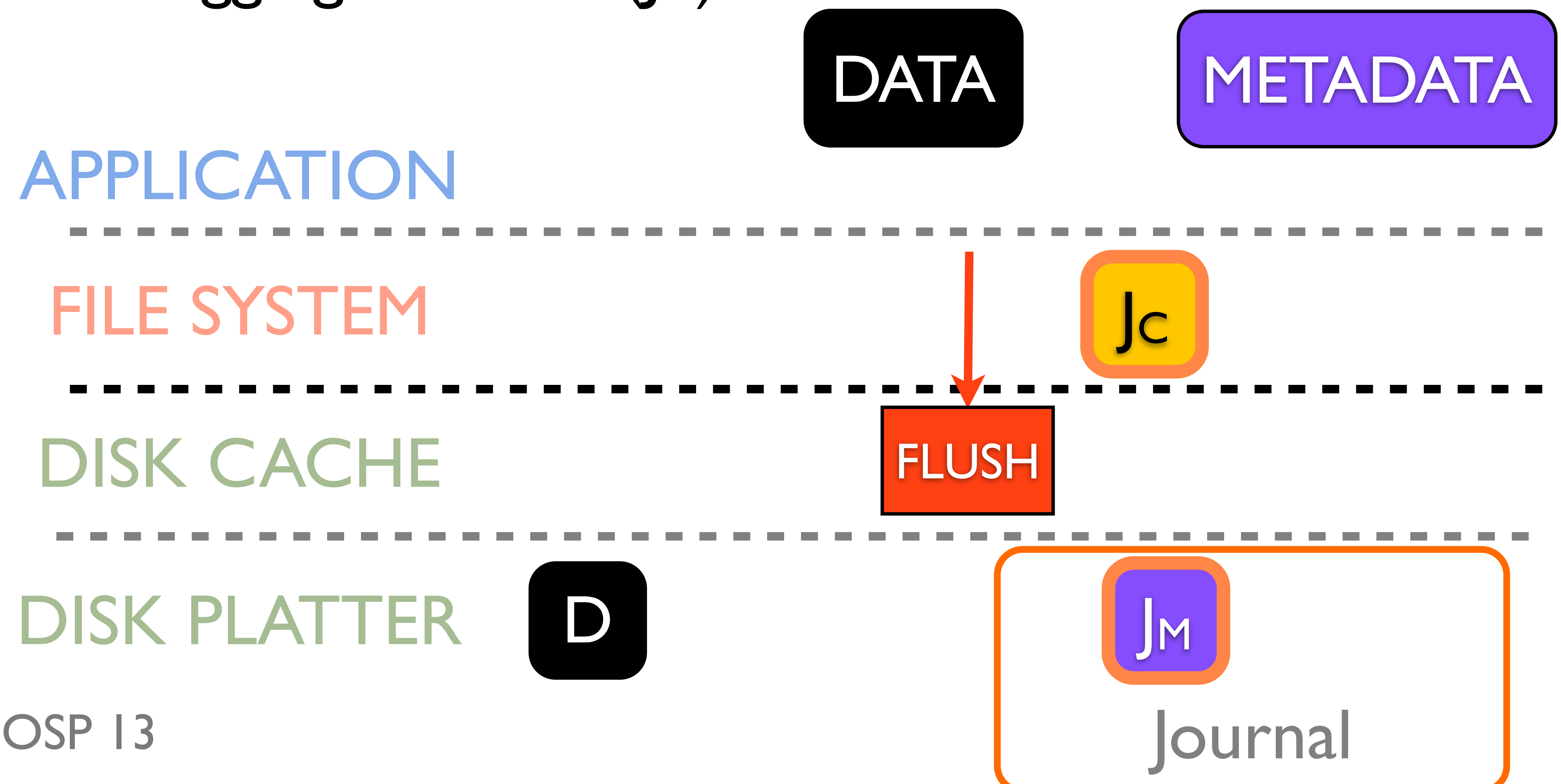
- Data write (D)
- Logging Metadata (J_M)



Journaling with Flushes

Journaling protocol

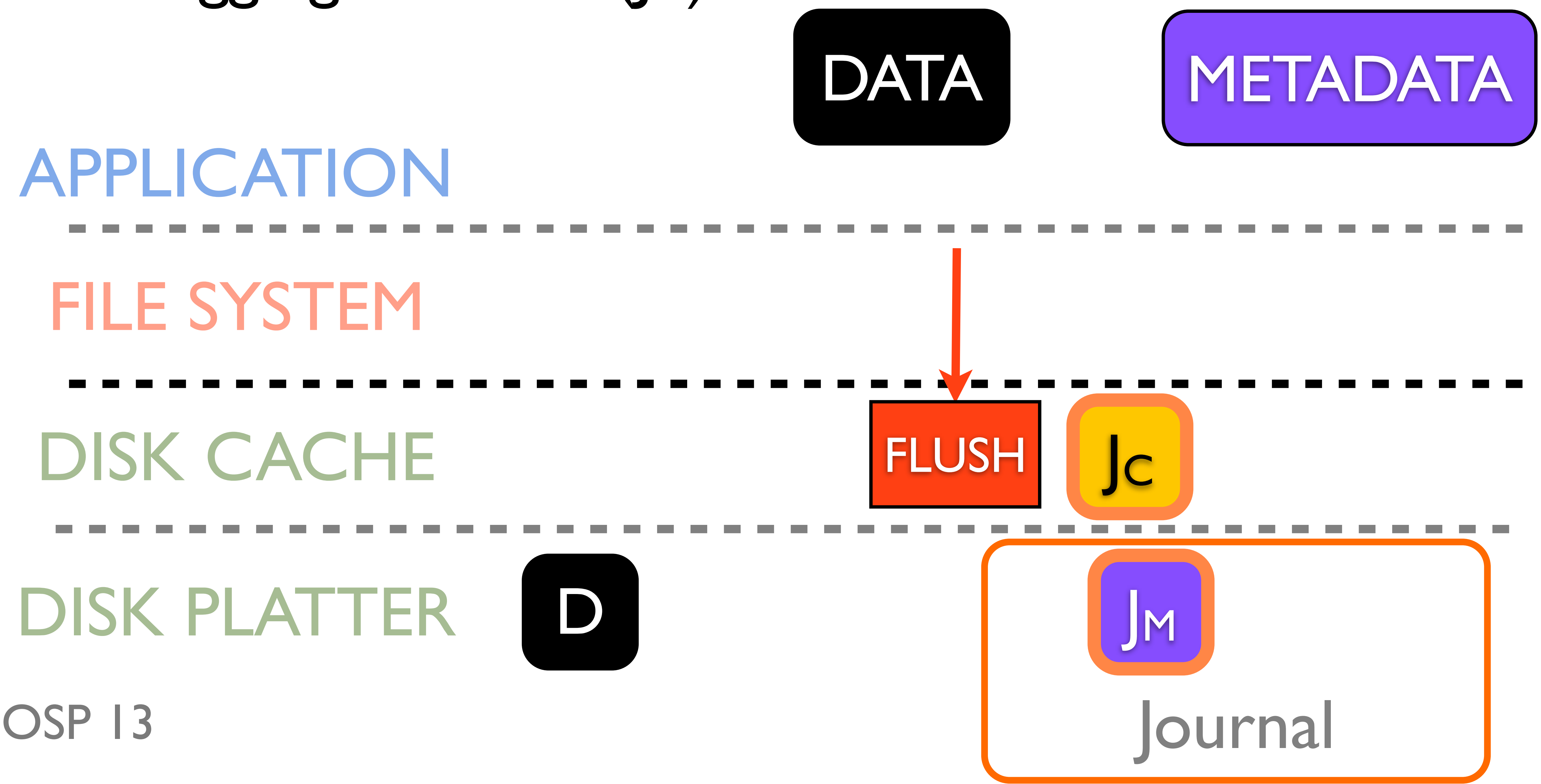
- Data write (D)
- Logging Metadata (J_M)
- Logging Commit (J_C)



Journaling with Flushes

Journaling protocol

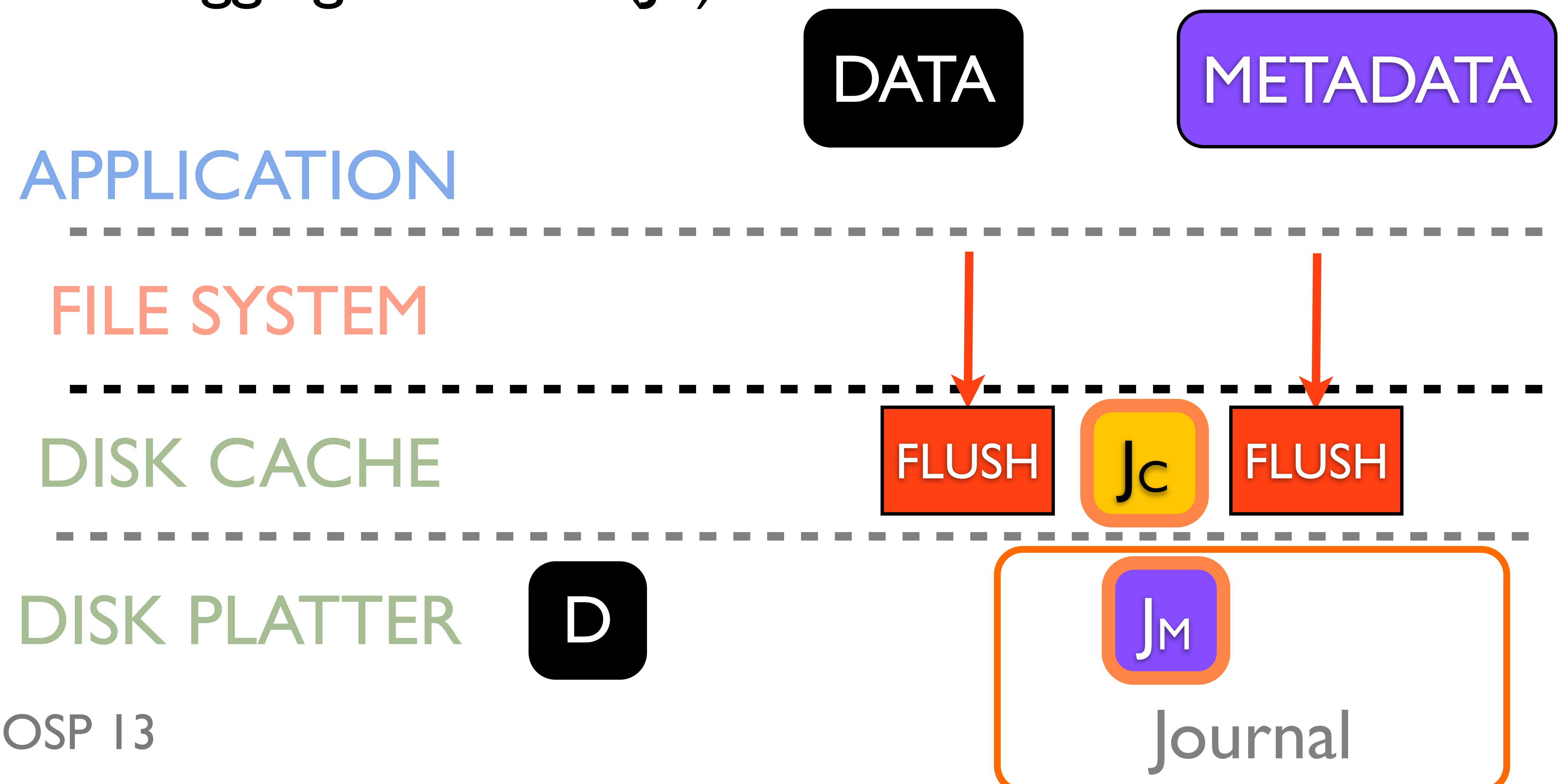
- Data write (D)
- Logging Metadata (J_M)
- Logging Commit (J_C)



Journaling with Flushes

Journaling protocol

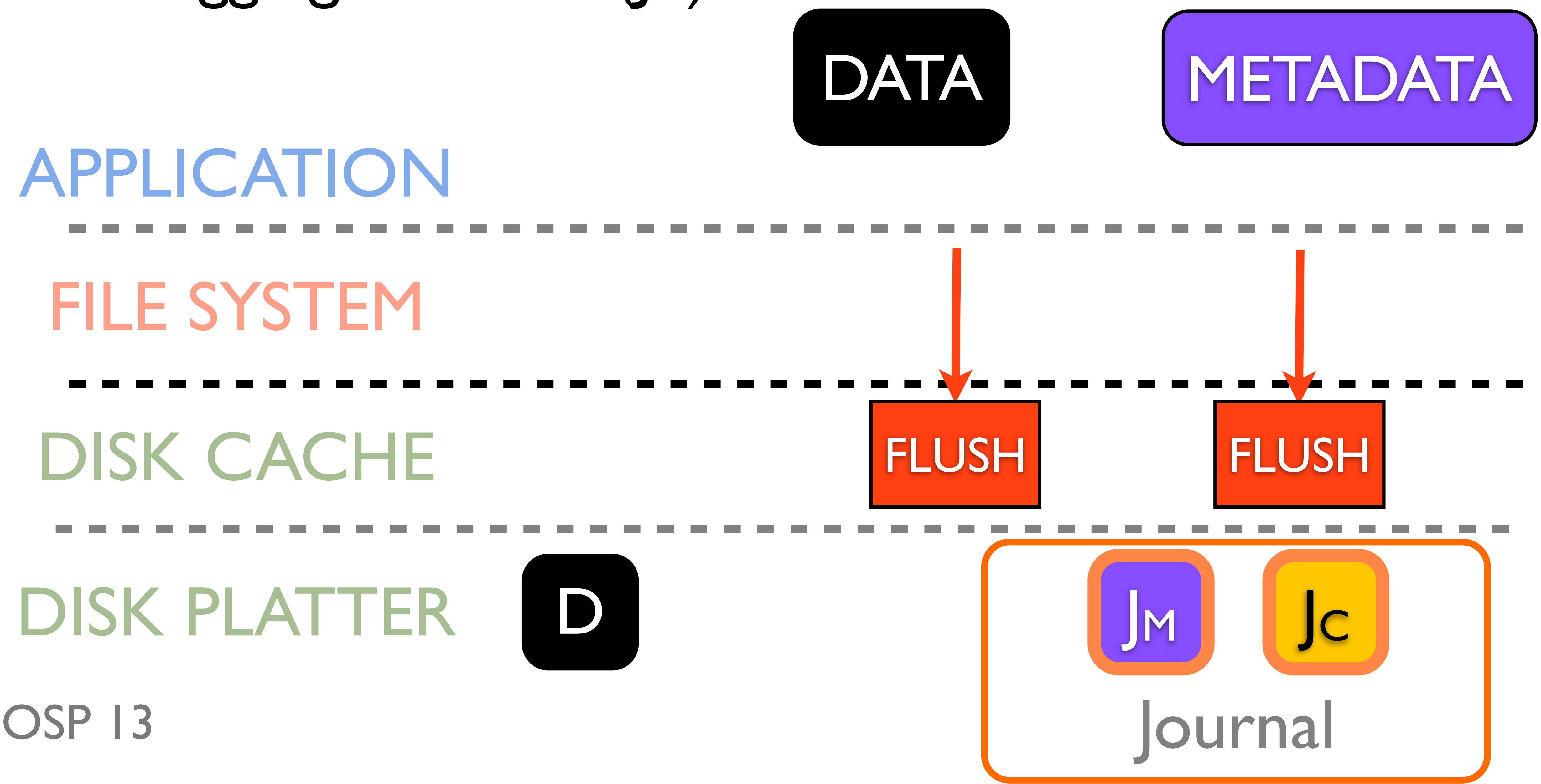
- Data write (D)
- Logging Metadata (J_M)
- Logging Commit (J_C)



Journaling with Flushes

Journaling protocol

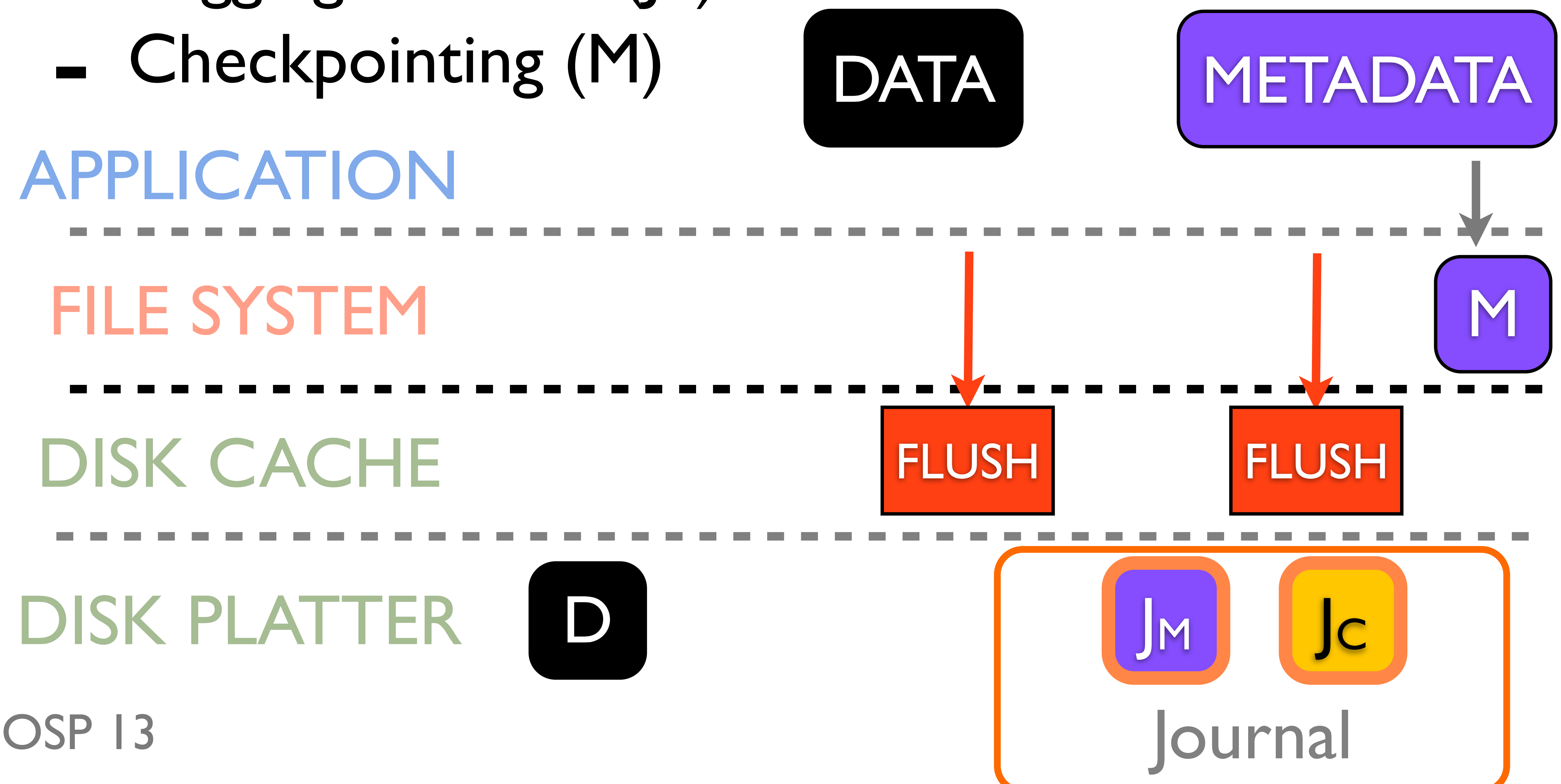
- Data write (D)
- Logging Metadata (J_M)
- Logging Commit (J_C)



Journaling with Flushes

Journaling protocol

- Data write (D)
- Logging Metadata (J_M)
- Logging Commit (J_C)
- Checkpointing (M)



Outline

Introduction

Ordering and Durability in Journaling

- Journaling Overview
- Realizing Ordering on Disks
- Journaling without Ordering

Optimistic File System

Results

Conclusion

Journaling without Ordering

Practitioners turn off flushes due to performance degradation

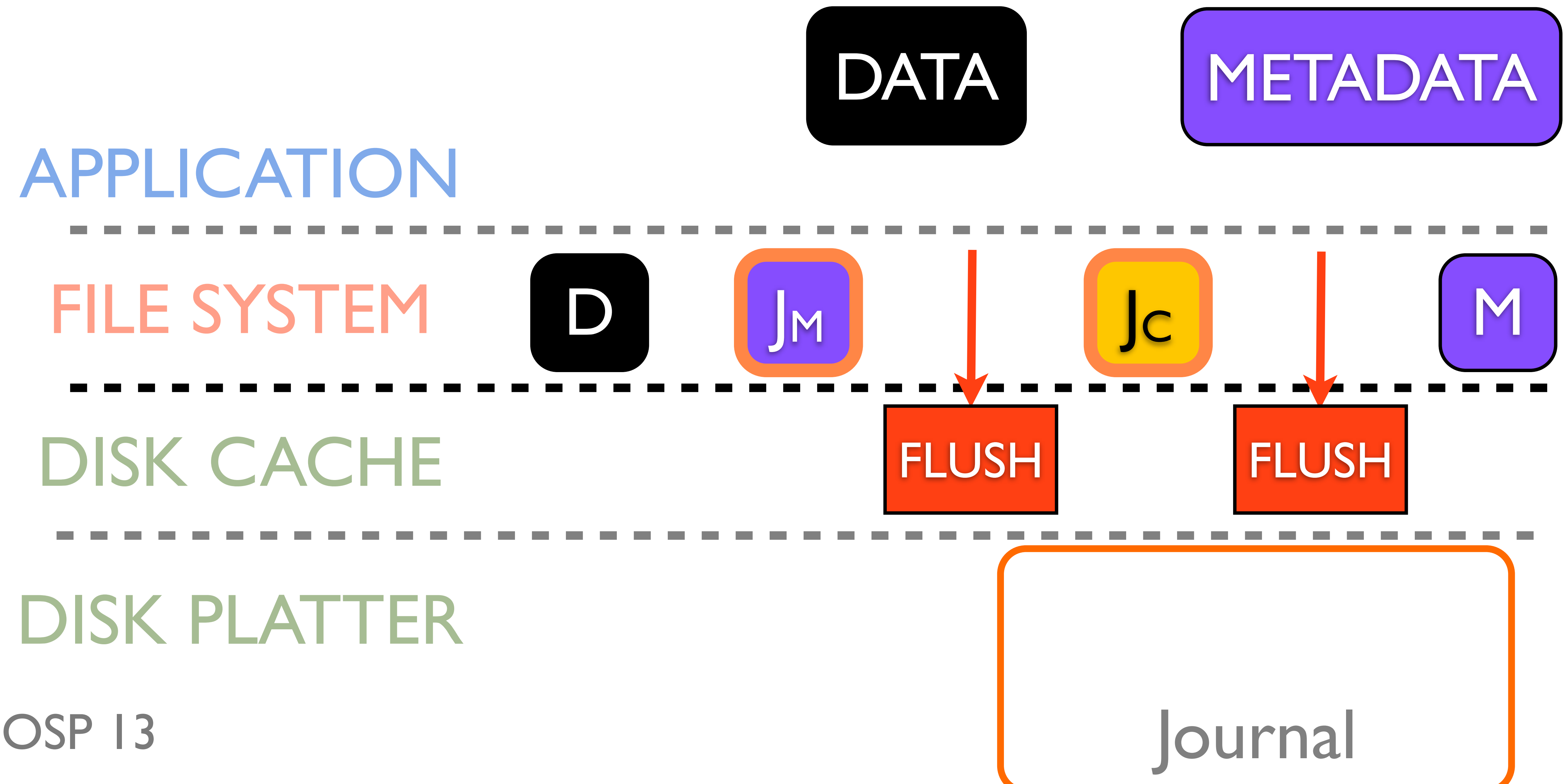
- Ex: ext3 by default did not enable flushes for many years

Observe crashes do not cause inconsistency for some workloads

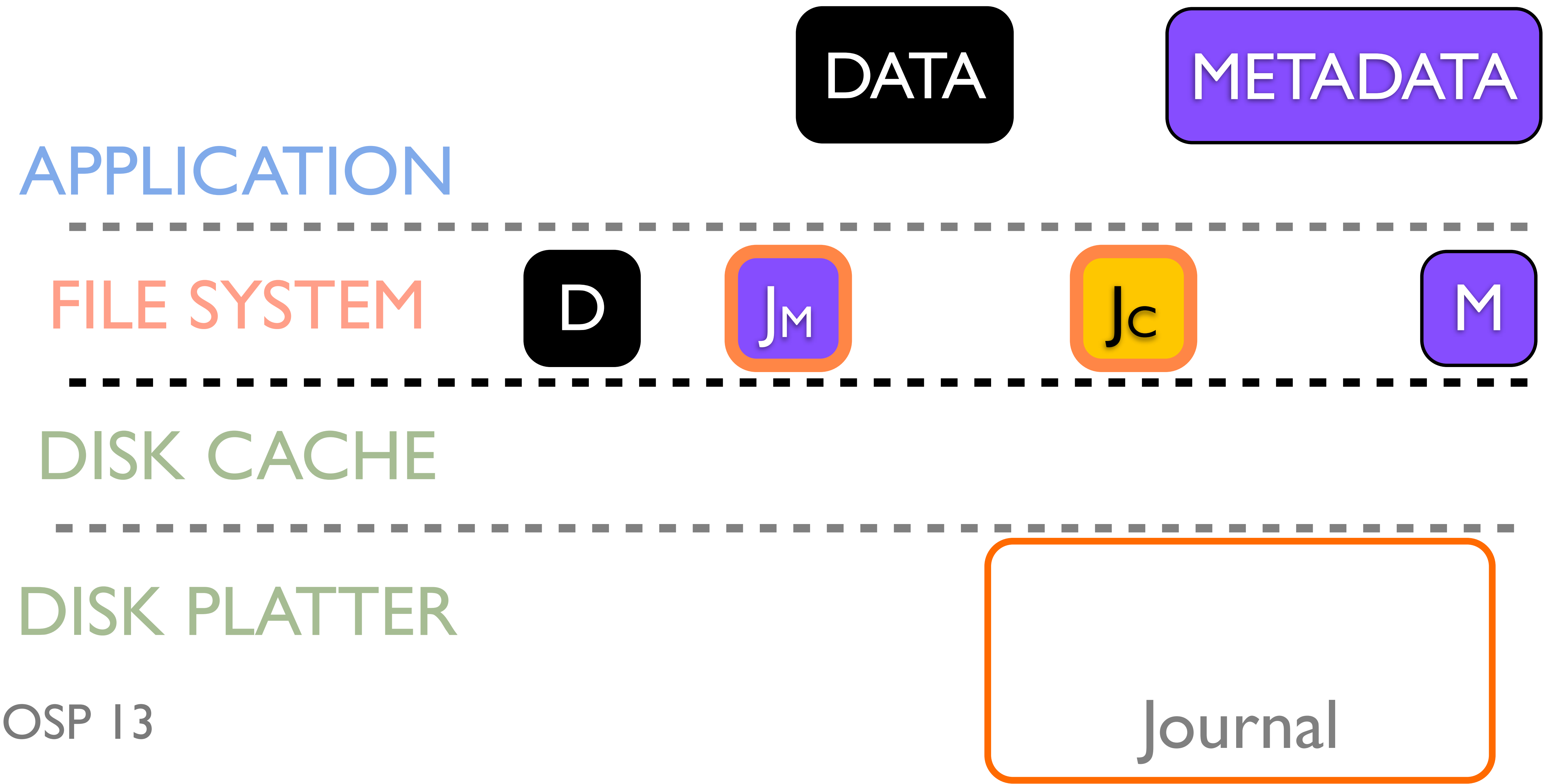
We term this probabilistic crash consistency

- Studied in detail

Journaling without Ordering

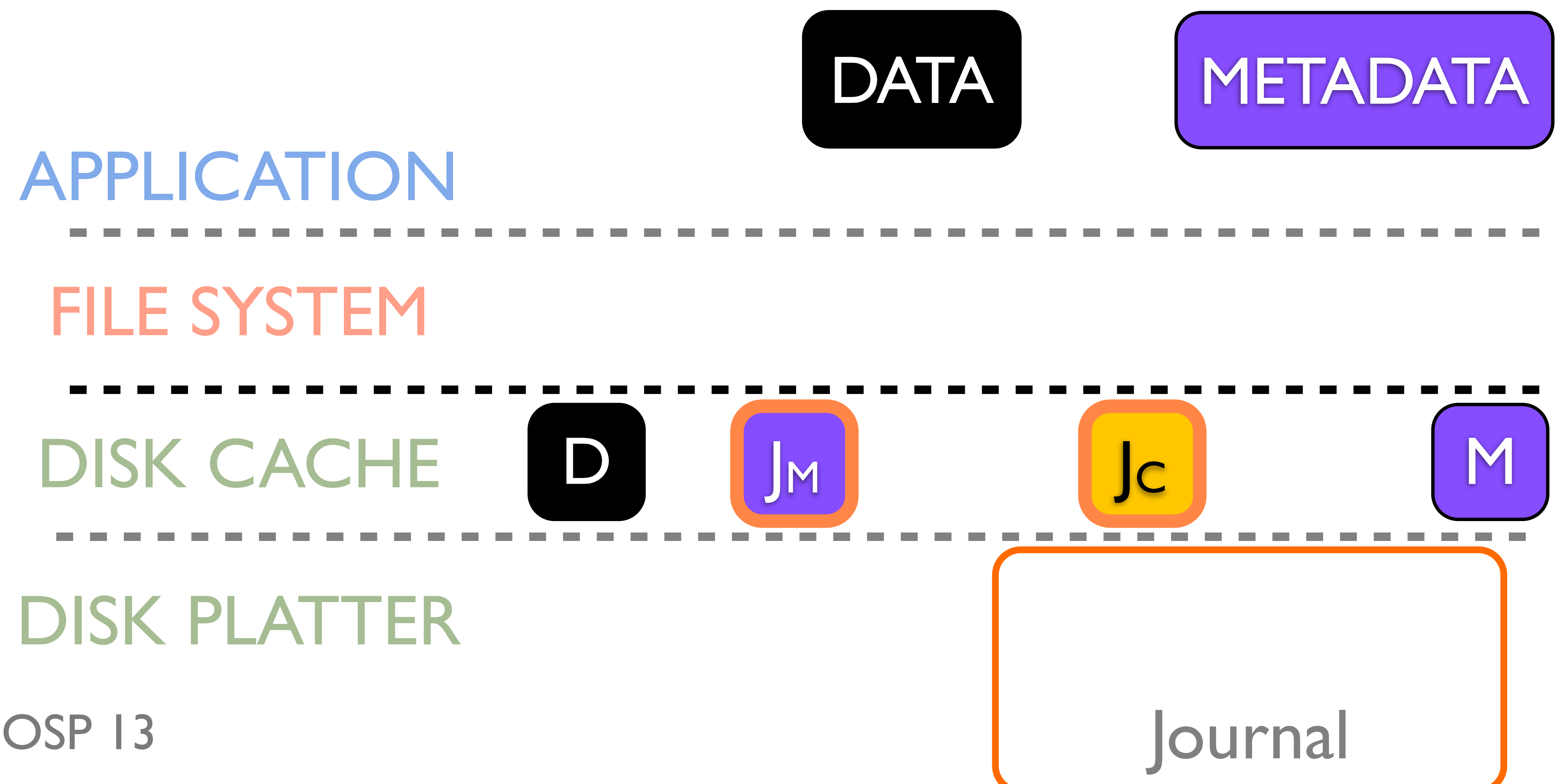


Journaling without Ordering



Journaling without Ordering

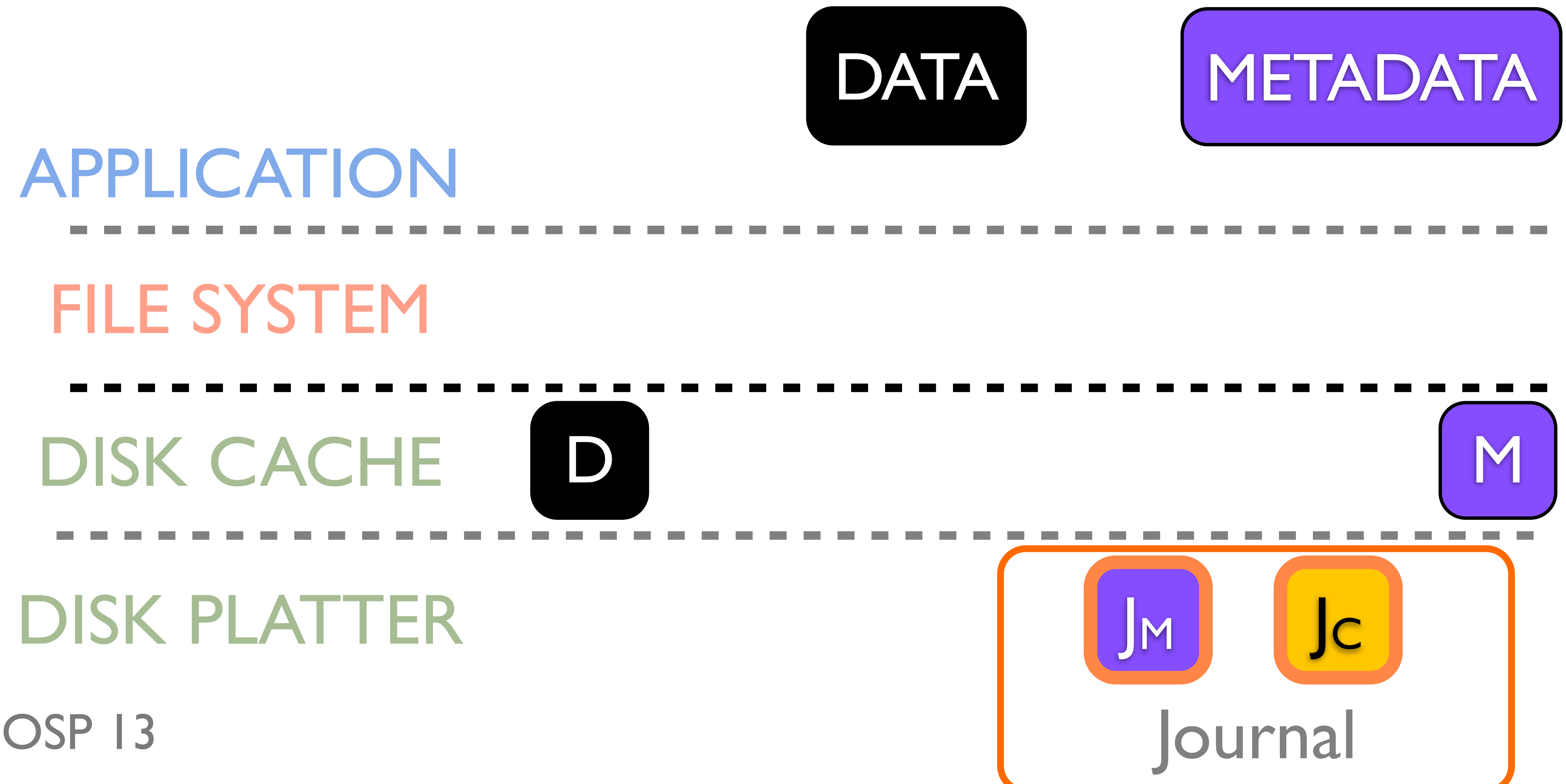
Without flushes, blocks may be reordered



Journaling without Ordering

Without flushes, blocks may be reordered

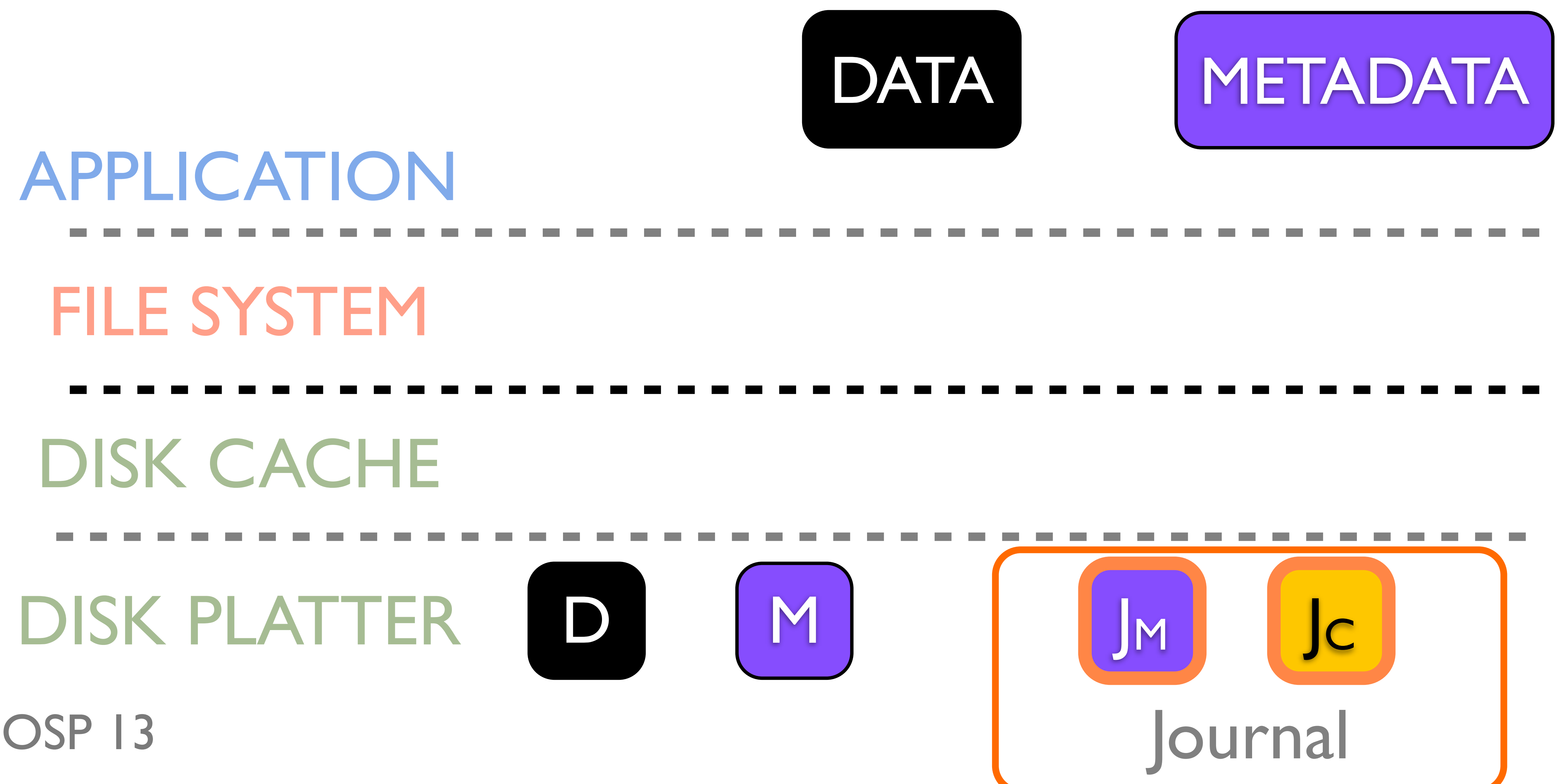
- Ex: J_C and J_M written first as disk head near journal



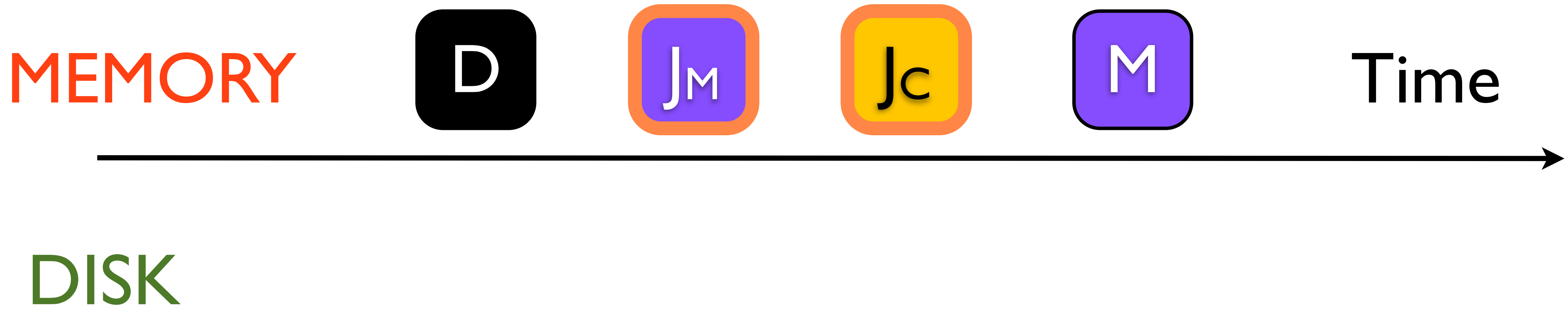
Journaling without Ordering

Without flushes, blocks may be reordered

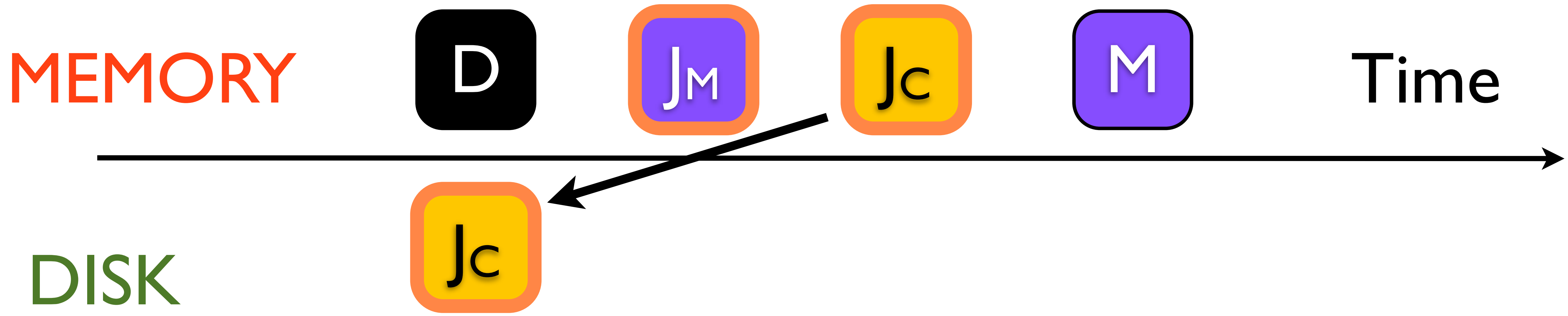
- Ex: J_C and J_M written first as disk head near journal



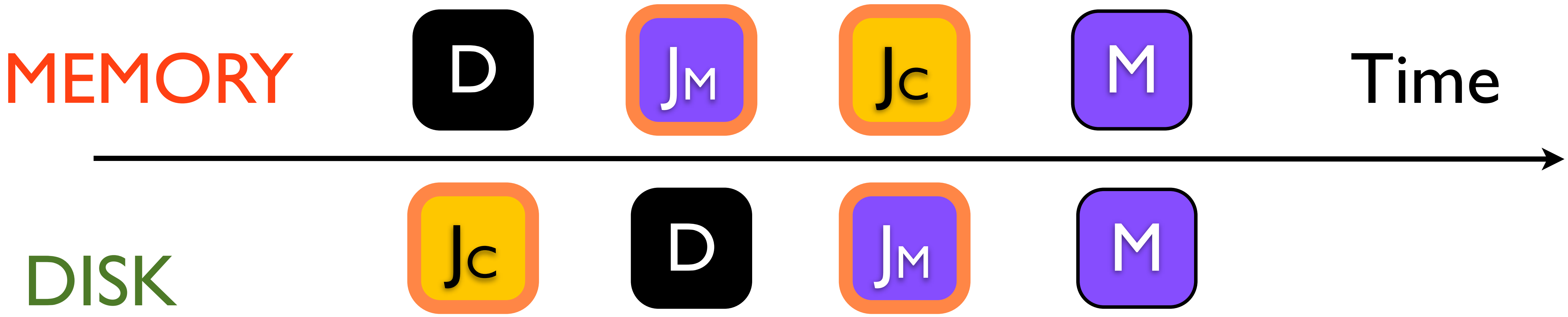
Probabilistic Crash Consistency



Probabilistic Crash Consistency

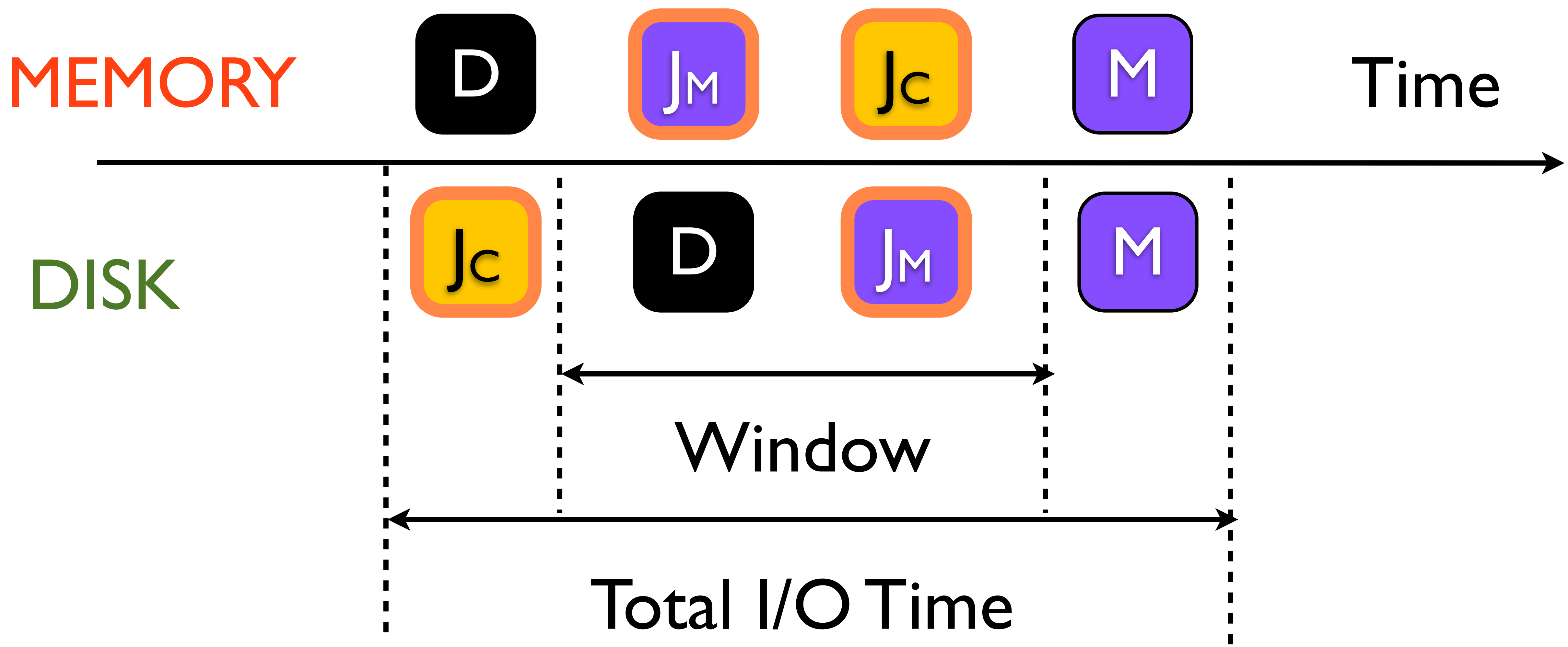


Probabilistic Crash Consistency



Probabilistic Crash Consistency

Re-ordering leads to windows of vulnerability



P-inconsistency = Time in window(s) / Total I/O Time

Probabilistic Crash Consistency

p-inconsistency for different workloads

- Read-heavy workloads have low p-inconsistency
- Database workloads have high p-inconsistency

See paper for detailed study

- Factors that affect p-inconsistency

Turning off flushing provides performance, but does not ensure consistency

Additional techniques required to obtain both performance and consistency

Outline

Introduction

Ordering and Durability in Journaling

Optimistic File System

- Overview
- Handling Re-Ordering
- New File-system Primitives

Results

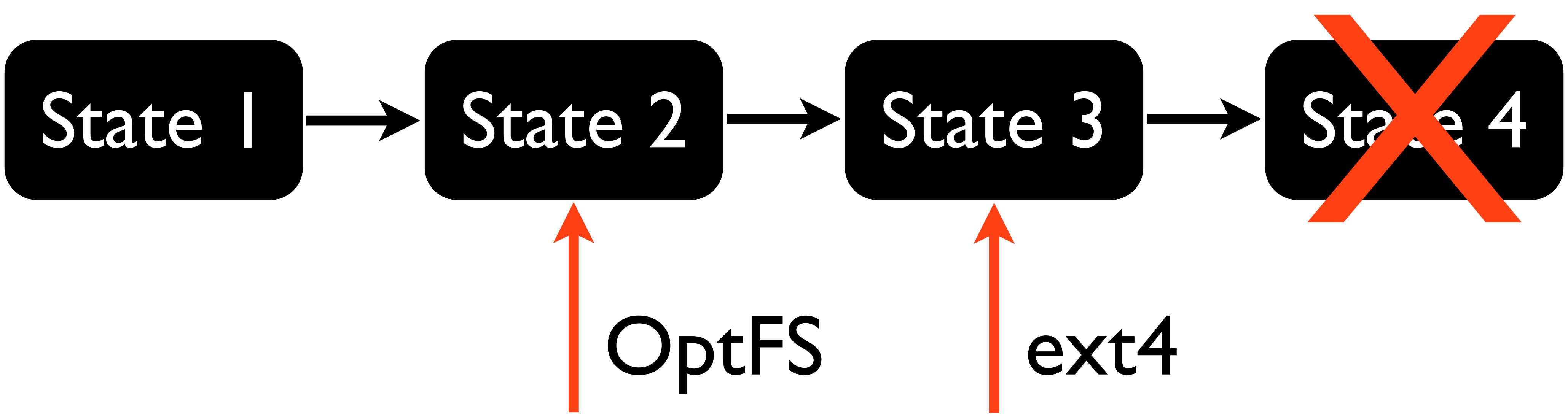
Conclusion

Optimistic File System

Achieves both performance and consistency by trading on new axis

Freshness indicates how up-to-date state is after a crash

OptFS provides strong consistency while trading freshness for increased performance



Optimistic File System

Eliminates flushes in the common case

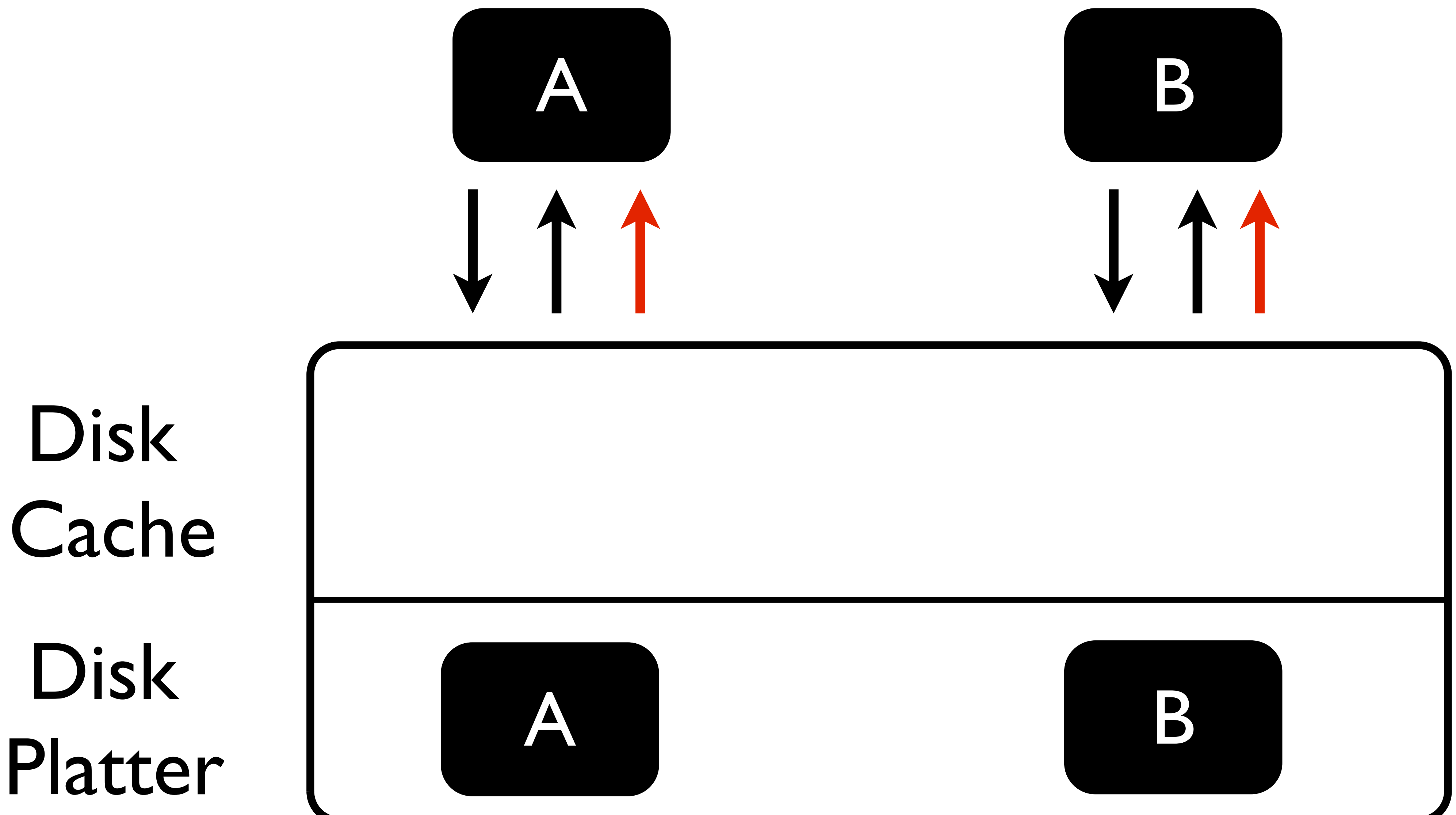
Blocks may be re-ordered without flushes

Optimistic Crash Consistency handles re-orderings with different techniques

- Some re-orderings are detected after crash
- Some re-orderings are prevented from occurring

Modified Disk Interface

Asynchronous Durability Notifications (ADN)
signal when block is made durable



Modified Disk Interface

ADNs increase disk freedom

- Blocks can be destaged in any order
- Blocks can be destaged at any time
- Only requirement is to inform upper layer

OptFS uses ADNs to control what blocks are dirty at the same time in disk cache

- Re-ordering can only happen among these blocks

Outline

Introduction

Ordering and Durability in Journaling

Optimistic File System

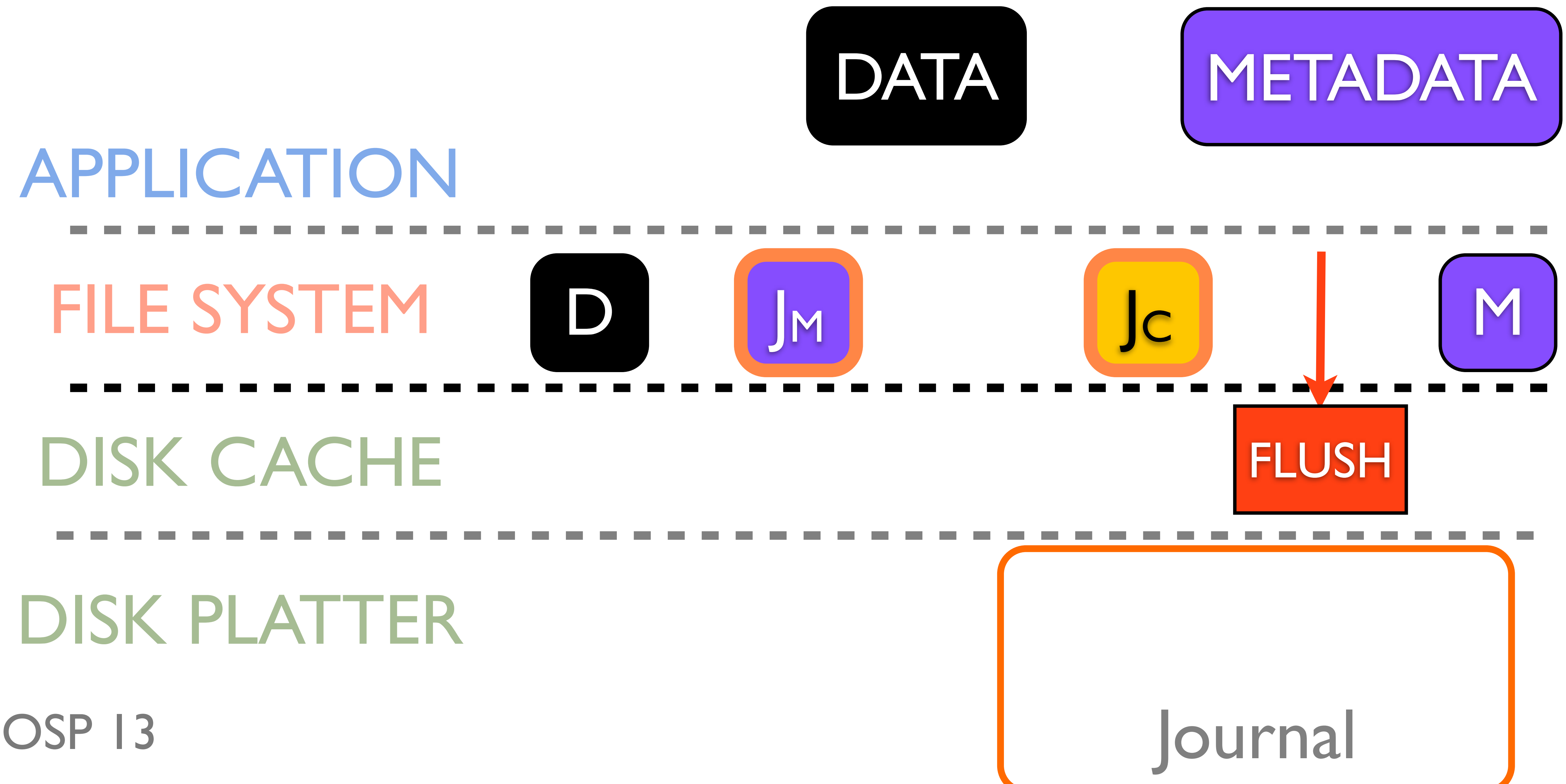
- Overview
- Handling Re-Ordering
- New File-system Primitives

Results

Conclusion

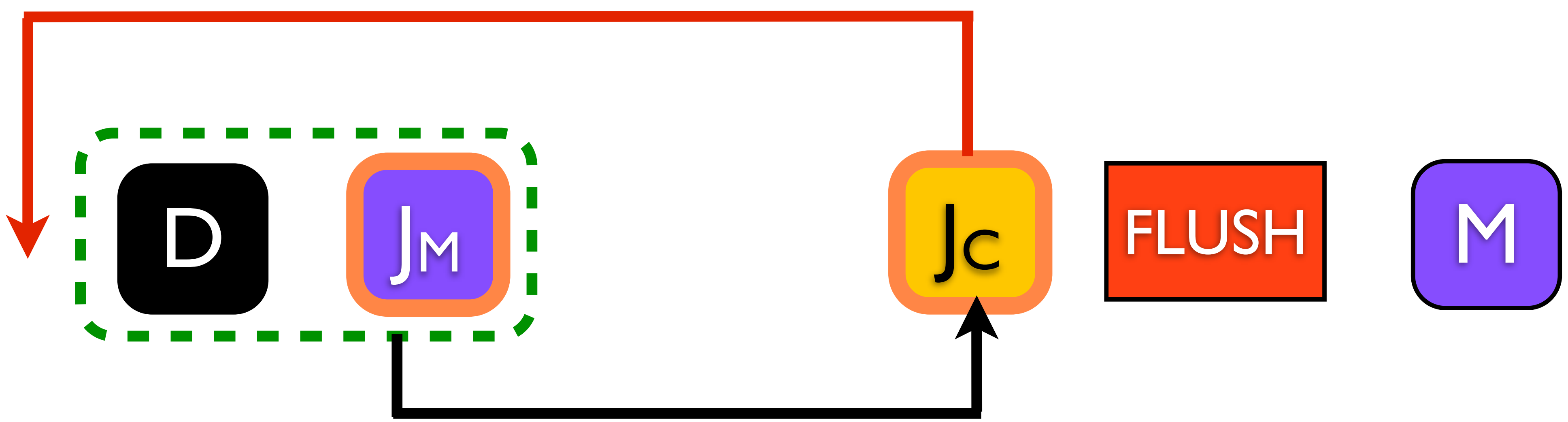
Handling Re-Ordering: Removing Flush #1

- Flush after J_M is removed
  - Checksums used to handle reordering



Technique #1: Checksums

J_C could be re-ordered before D or J_M



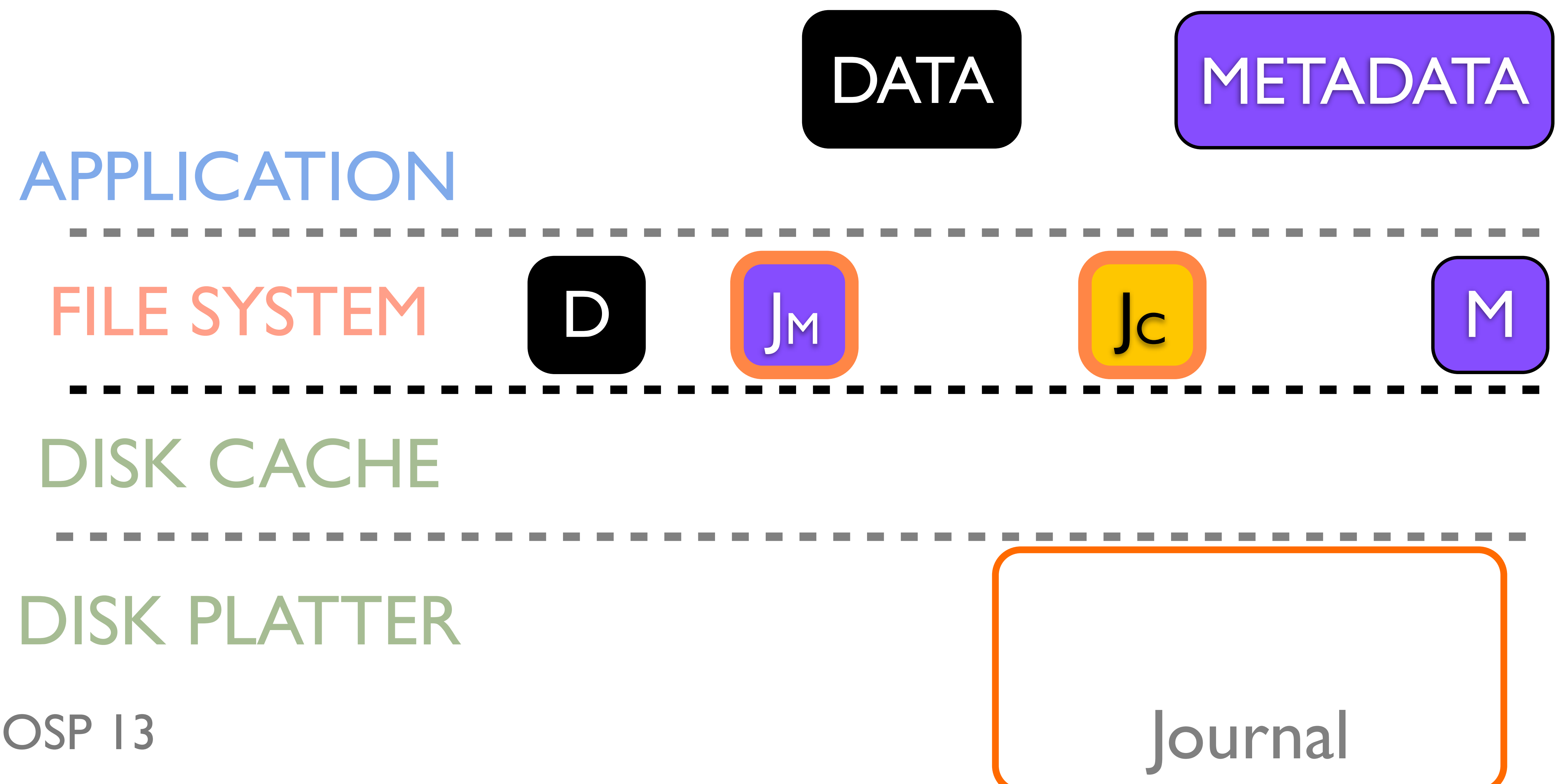
Re-ordering detected using checksums

- Computed over data and metadata
- Checked during recovery
- Mismatch indicates blocks were lost during crash

Handling Re-Ordering: Removing Flush #2

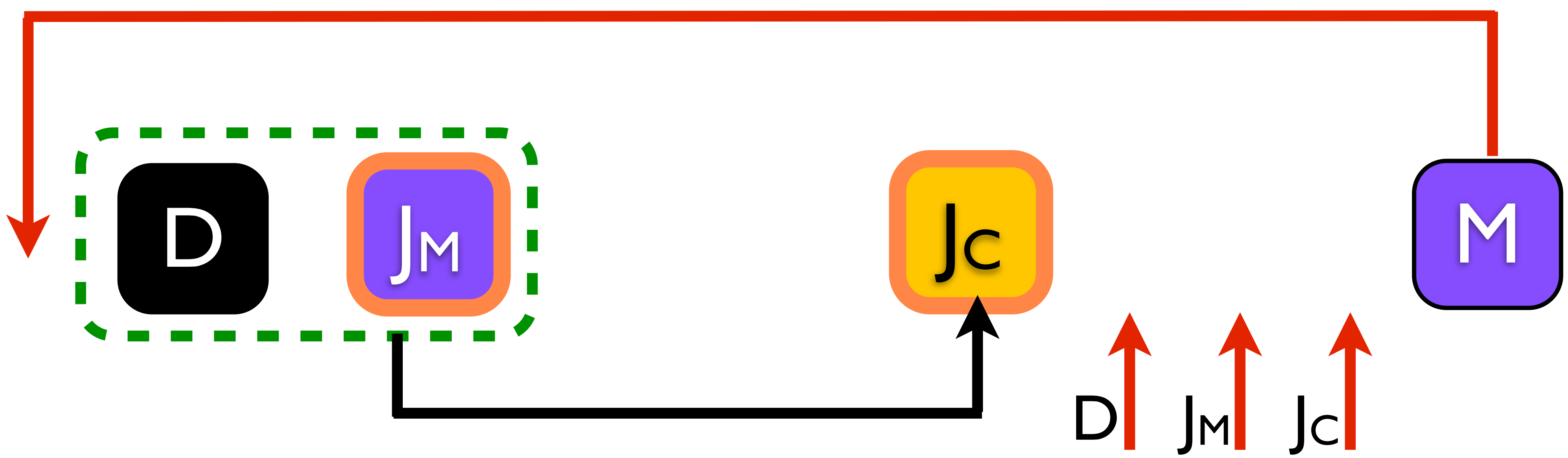
Flush after J_C is removed

- Delayed writes used to prevent reordering



Technique #2: Delayed Writes

M could be re-ordered before D or J_M or J_C

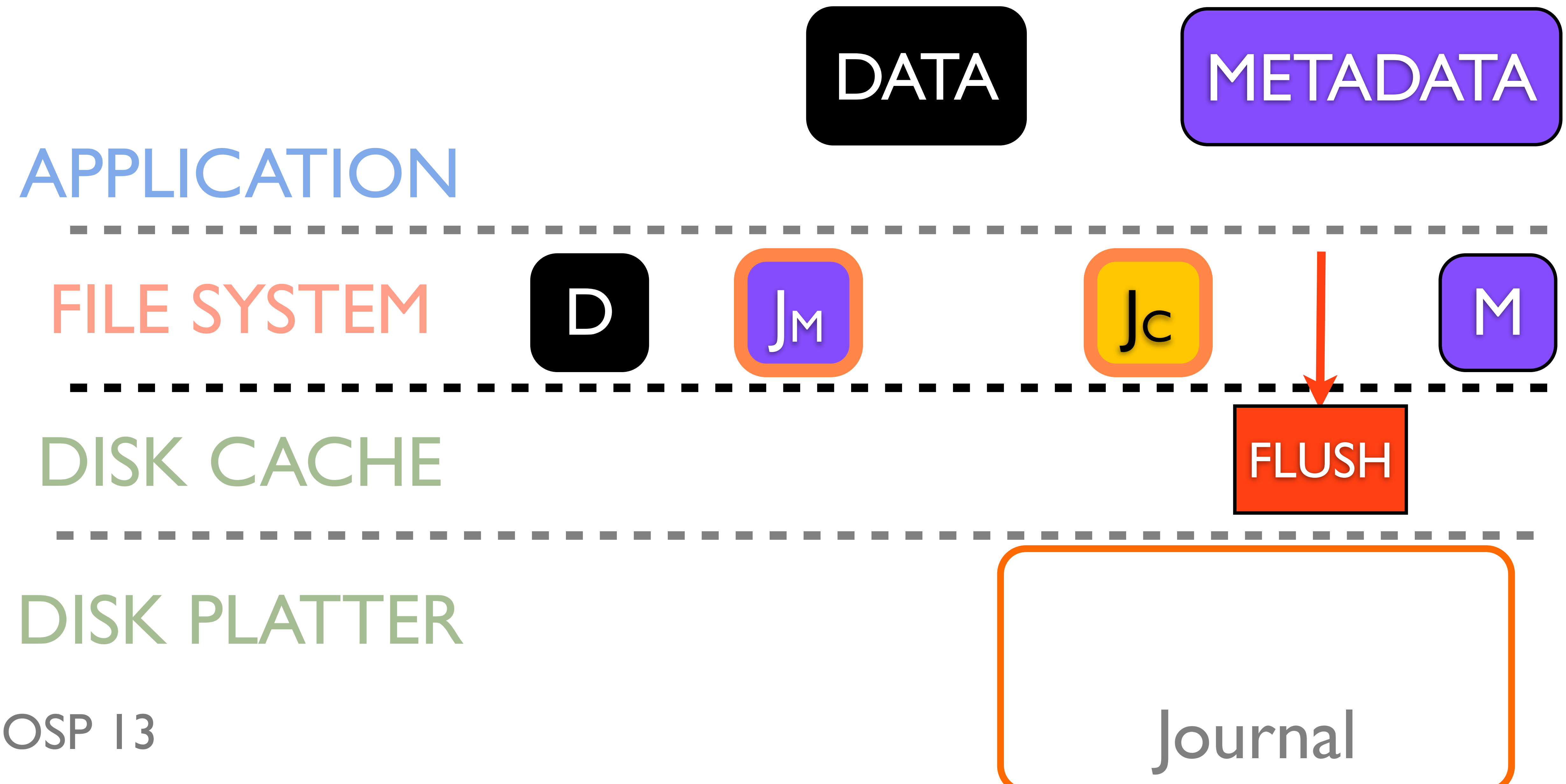


Re-ordering prevented using delayed writes

- Wait until ADN arrive for D, J_M, and J_C
- Then issue M to disk cache
- Invariant: D/J_M/J_C and M never dirty in cache together

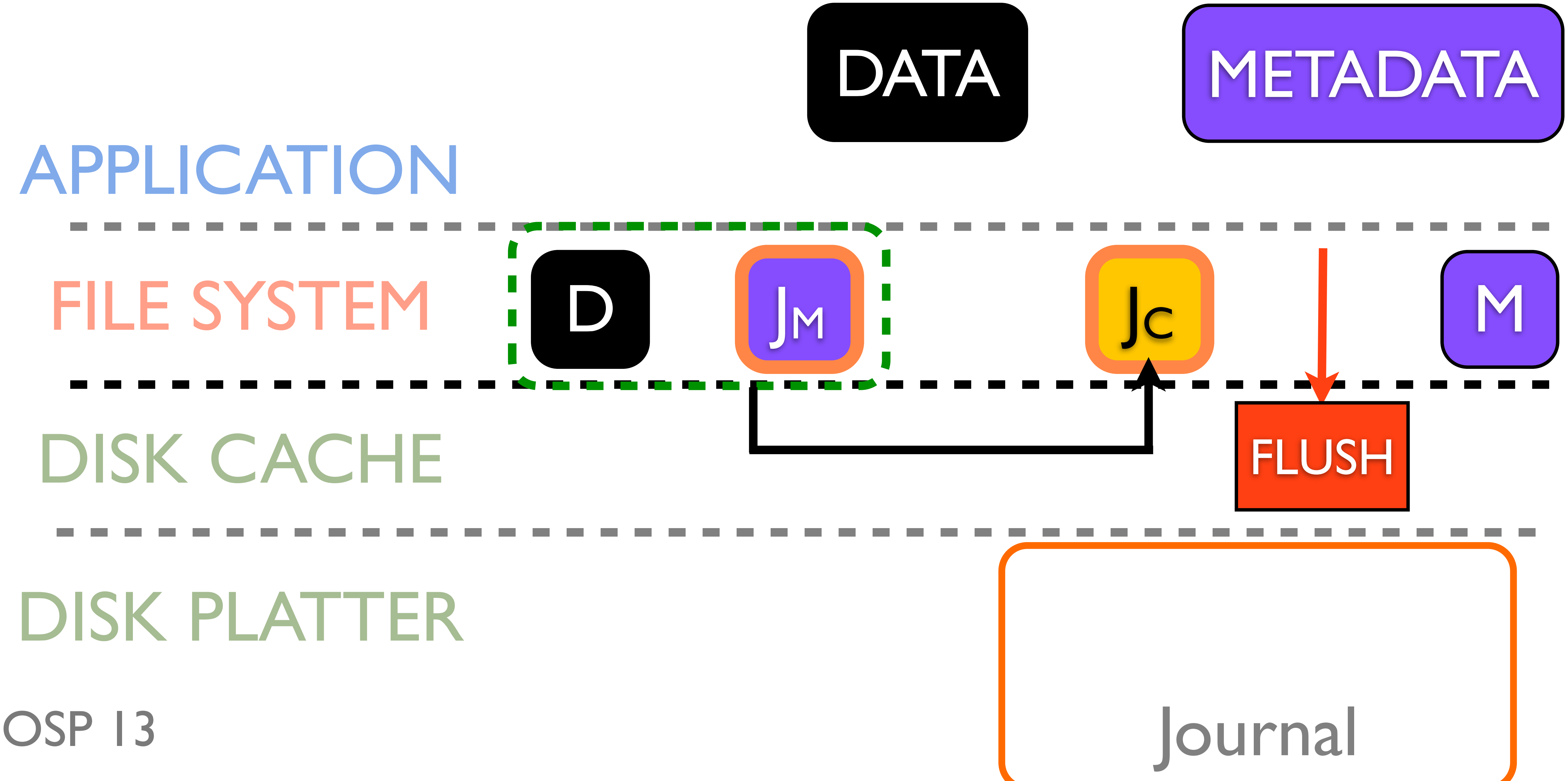
Optimistic Journaling

Checksums and Delayed Writes handle reordering from removing flushes



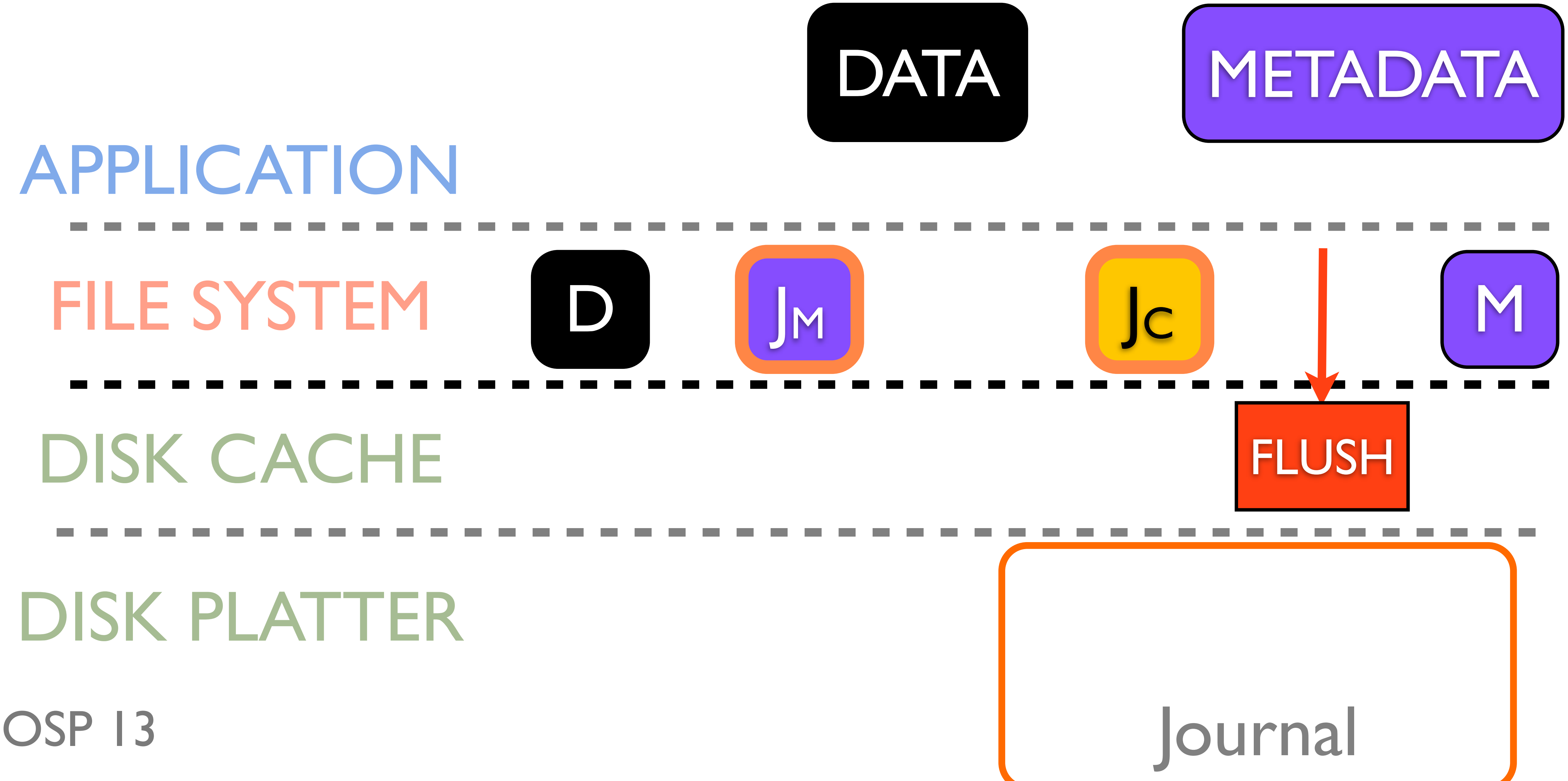
Optimistic Journaling

Checksums and Delayed Writes handle reordering from removing flushes



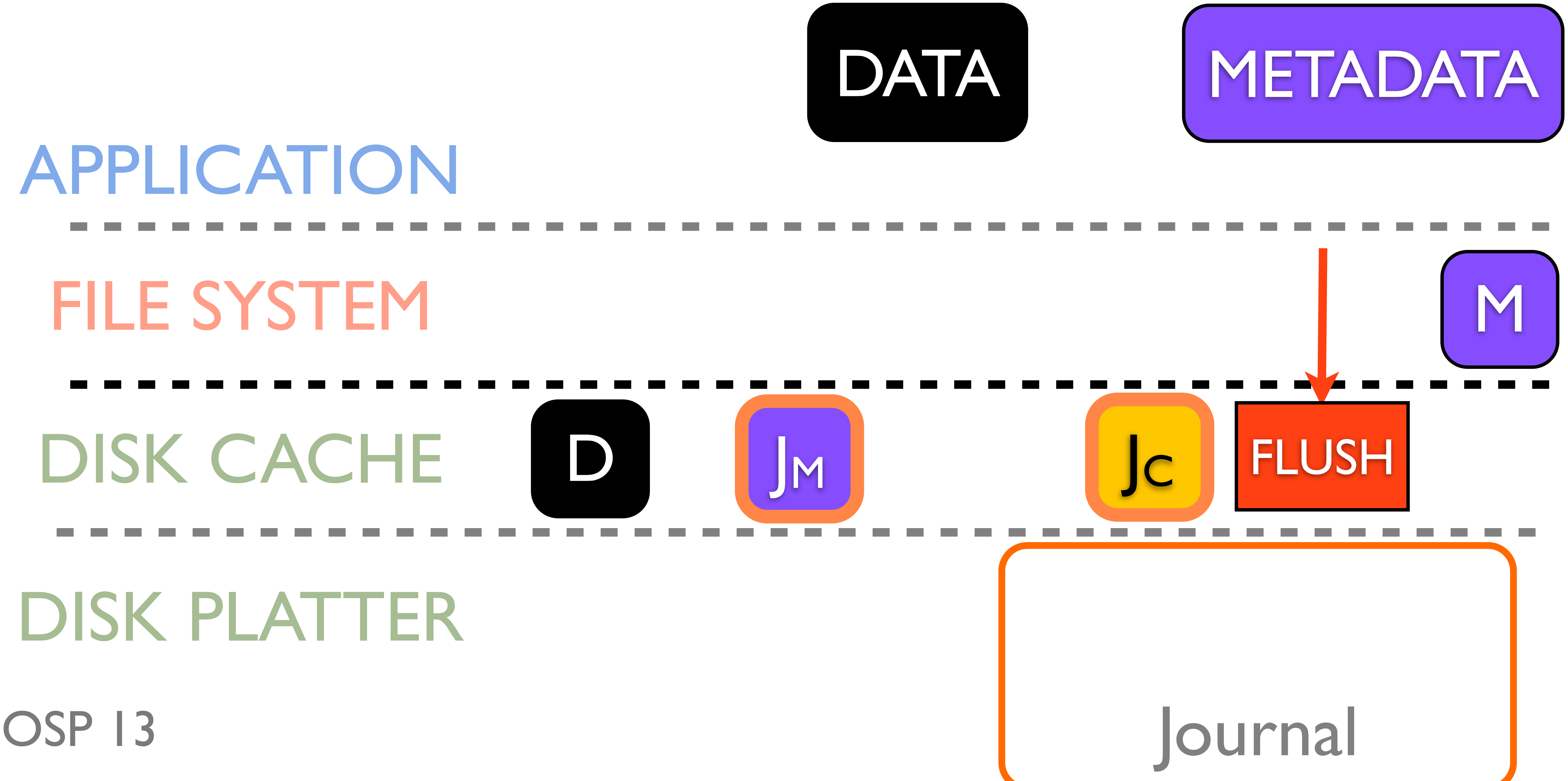
Optimistic Journaling

Checksums and Delayed Writes handle reordering from removing flushes



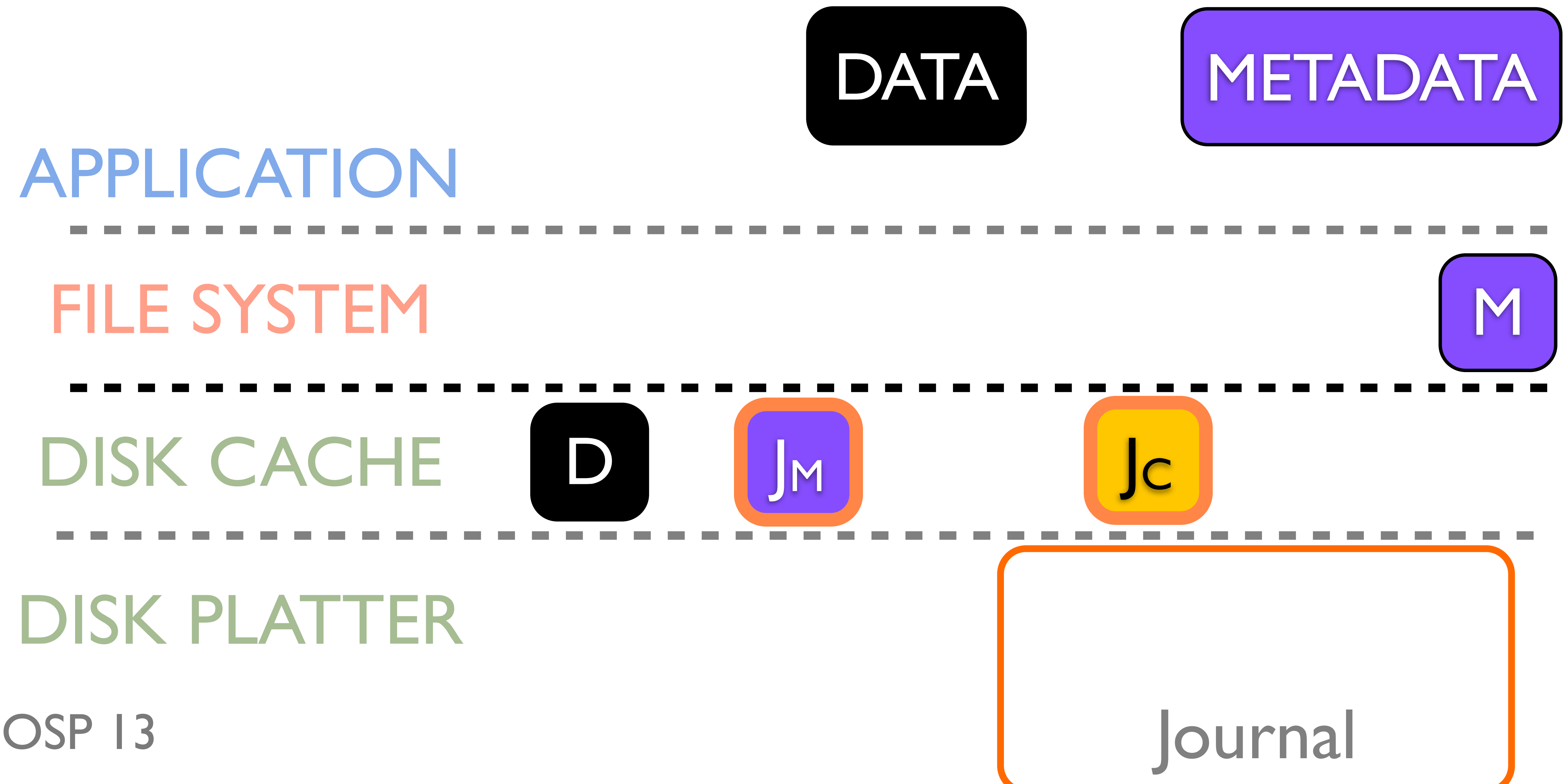
Optimistic Journaling

Checksums and Delayed Writes handle reordering from removing flushes



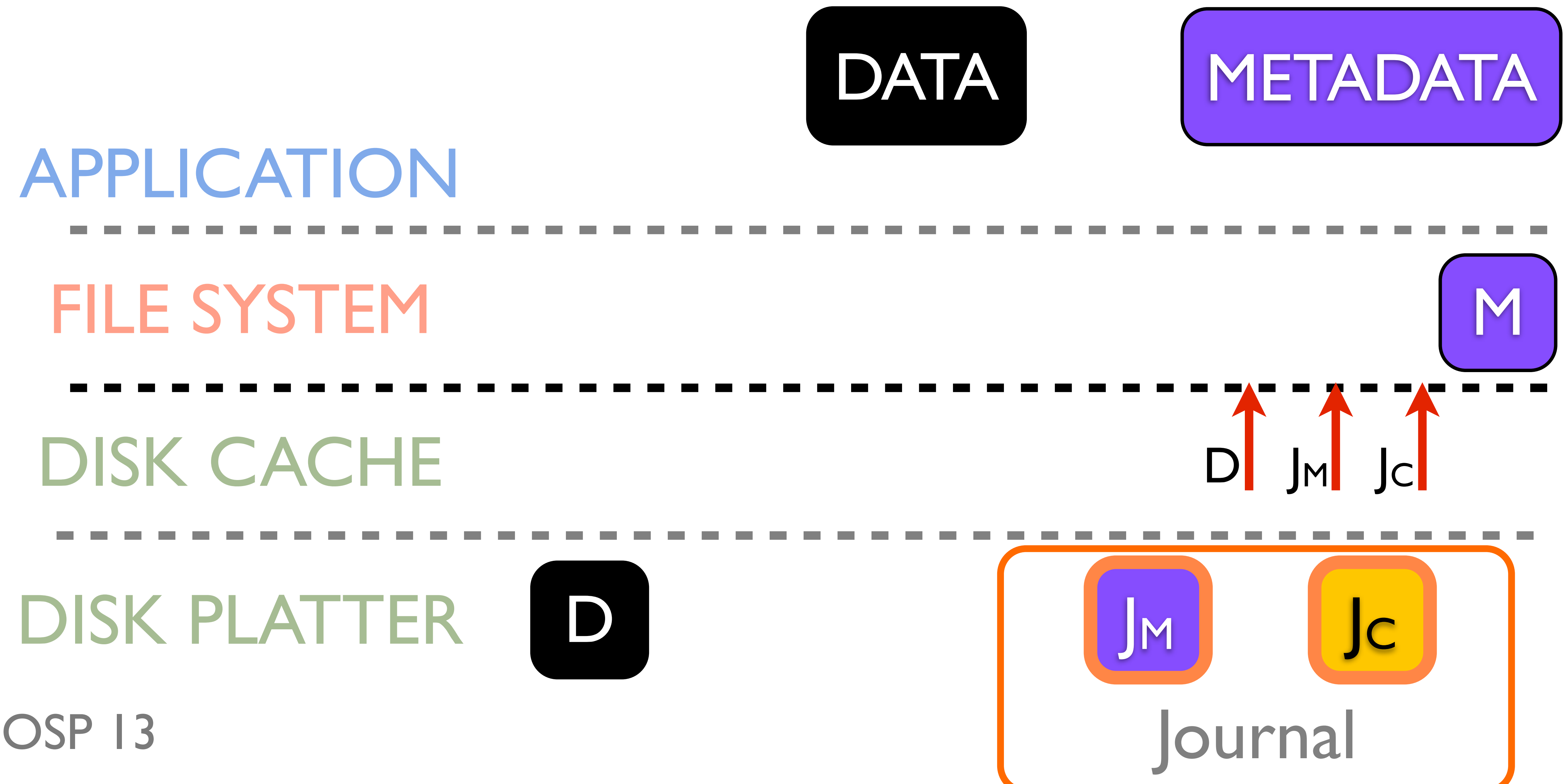
Optimistic Journaling

Checksums and Delayed Writes handle reordering from removing flushes



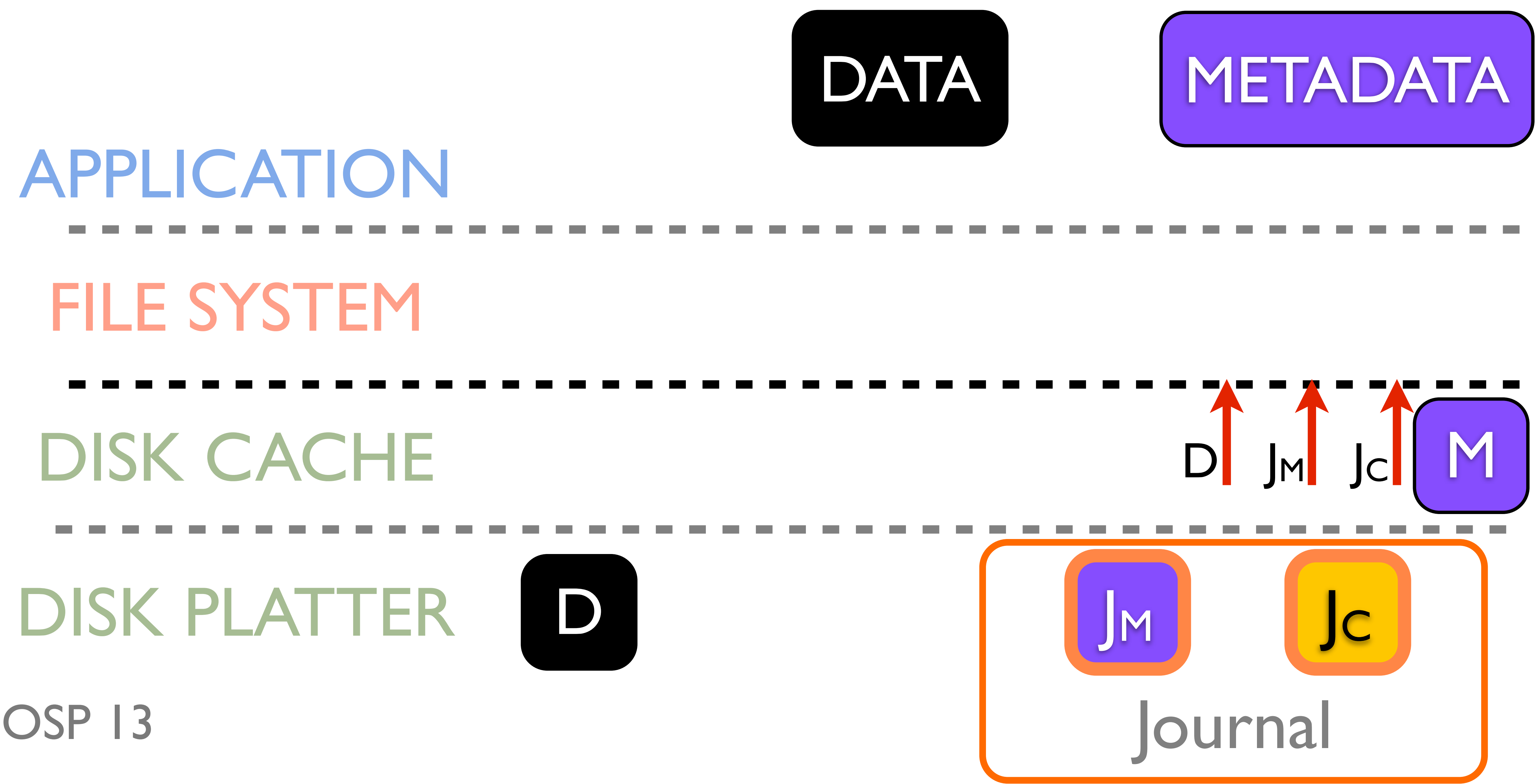
Optimistic Journaling

Checksums and Delayed Writes handle reordering from removing flushes



Optimistic Journaling

Checksums and Delayed Writes handle reordering from removing flushes



Optimistic Techniques

Other Techniques

- In-order journal recovery and release
- Reuse after notification
- Selective data journaling

See paper for more details

Outline

Introduction

Ordering and Durability in Journaling

Optimistic File System

- Overview
- Handling Re-Ordering
- New File-system Primitives

Results

Conclusion

File-system Primitives

`fsync()` provides ordering and durability

OptFS splits `fsync()`

- `osync()` for only ordering and high performance
- `dsync()` for durability

Primitives can increase performance

- Ex: SQLite

`write(log)`	`write(log)`
`fsync(log)`	`osync(log)`
`write(header)`	`write(header)`
`fsync(header)`	`dsync(header)`

Implementation

OptFS based on ext4 code

- Around 3000 lines of modified/added code

Required modifications to

- Journaling layer
- Virtual Memory subsystem

ADNs were emulated using timeouts

- Block received by disk at time T
- Block durable at time T+D
- D = 30 s in our implementation (conservative)

Outline

Introduction

Ordering and Durability in Journaling

Optimistic File System

Results

Conclusion

Evaluation

Does OptFS preserve file-system consistency after crashes?

- OptFS consistent after 400 random crashes

How does OptFS perform?

- OptFS 4-10x better than ext4 with flushes

Can meaningful application-level consistency be built on top of OptFS?

- Studied gedit and SQLite on OptFS

Testing Application-Level Consistency

Methodology

- Start from initial disk image
- Run application
  - Replace fsync() with osync()
  - Trace writes
- Re-order writes
- Drop writes after random point
- Replay writes on initial disk image
- Examine application state on new image

SQLite Consistency

SQLite Consistency

SQLite Consistency

SQLite Consistency

Initial Image

SQLite Consistency

ext4 without flushes 73%

SQLite Consistency

Zero inconsistencies with
OptFS
or
ext4 with flushes

SQLite Consistency

50% ext4 with flushes 50%

SQLite Consistency

50%	ext4 with flushes	50%
76%	OptFS	24%

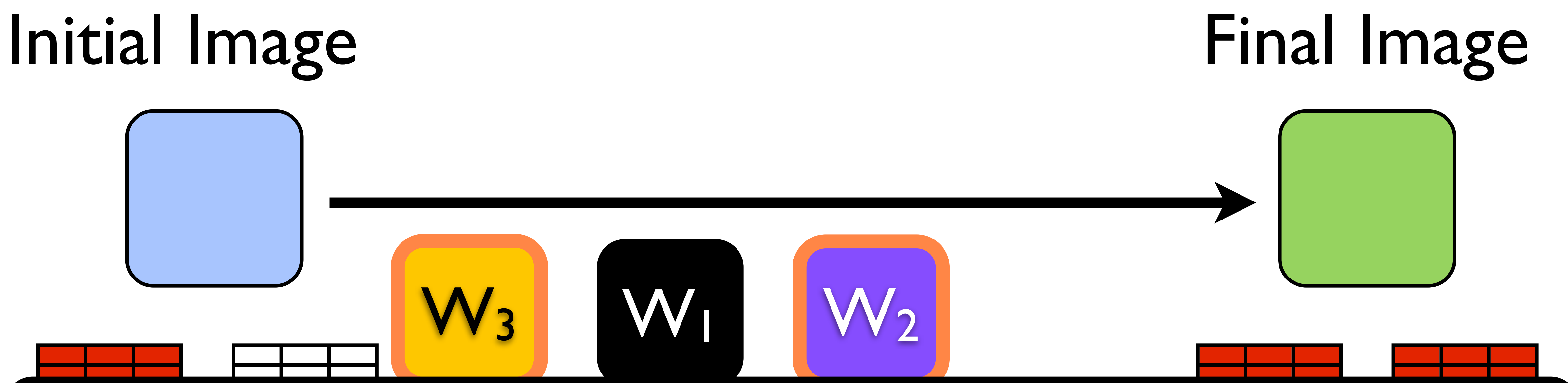
osync() changes semantics from ACID to ACI-(Eventual Durability)

SQLite Consistency

			Time
50%	ext4 with flushes	50%	150 ms
76%	OptFS	24%	15 ms

osync() changes semantics from ACID to
ACI-(Eventual Durability)

SQLite Consistency



SQLite is able to provide ACI semantics with osync(), at 10x performance

50%	ext4 with flushes	50%	150 ms
76%	OptFS	24%	15 ms

osync() changes semantics from ACID to ACI-(Eventual Durability)

Outline

Introduction

Ordering and Durability in Journaling

Optimistic File System

Results

Conclusion

Summary

Problem: providing both performance and consistency

Solution: decoupling ordering and durability in OptFS

Eventual Durability maintains consistency while trading freshness for increased performance

osync() provides a cheap primitive to order application writes

Conclusion

Storage-stack layers are increasing

- 18 layers between application and storage [Thereska13]
- Interfaces that provide freedom to each layer are the way forward

First impulse: trade consistency for performance

- Trade-off not required in distributed systems [Escriva12]
- By trading freshness, we can obtain both consistency and high performance

Thank You

Source code

http://research.cs.wisc.edu/adsl/Software/optfs/

http://github.com/vijay03/optfs

Questions?

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

WISDOM