Fast, Transparent Filesystem Microkernel Recovery with Ananke

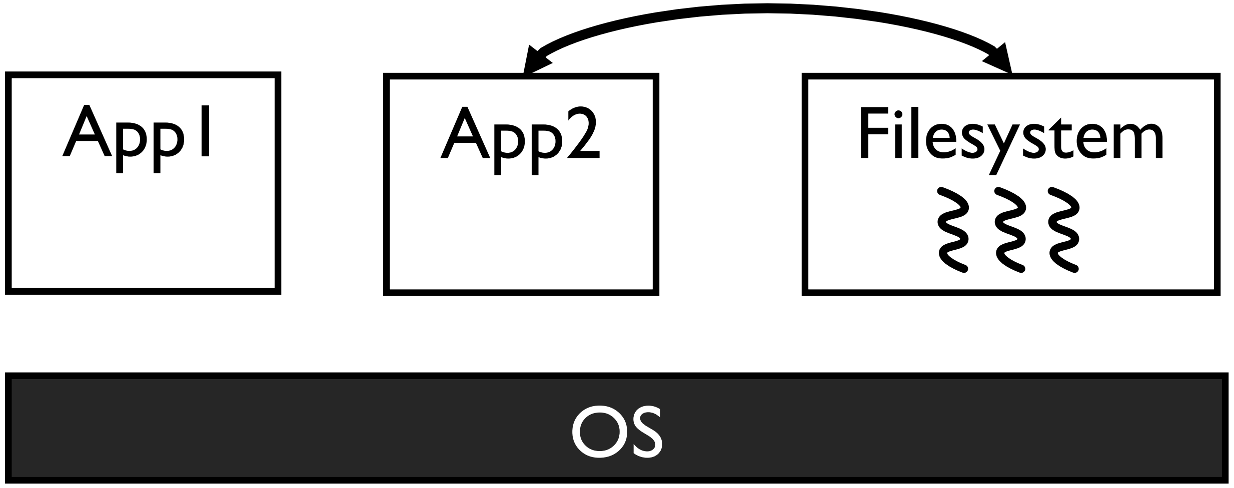
Jing Liu, Yifan Dai,

Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau

Microsoft

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

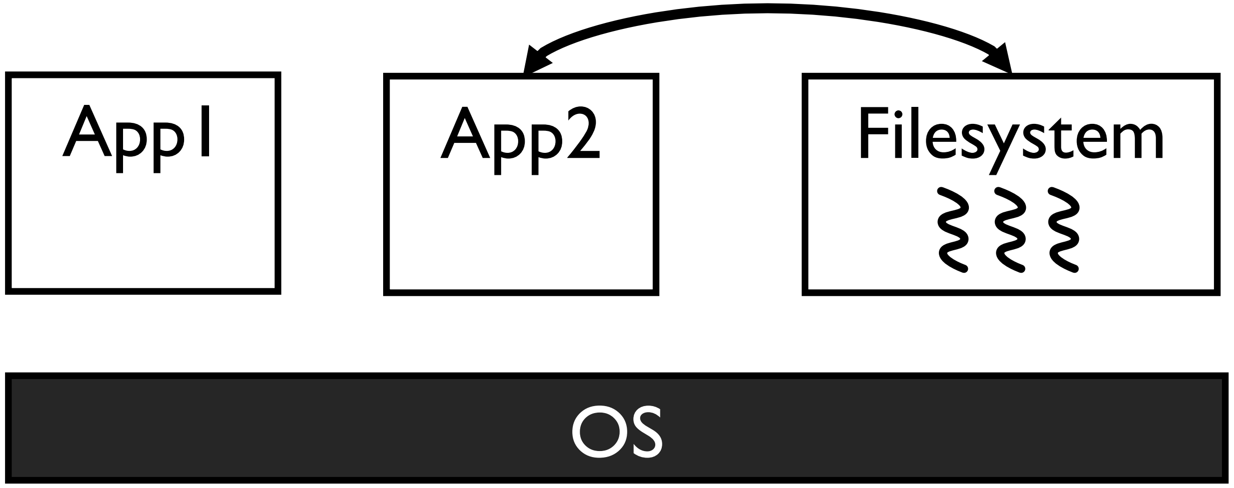
Filesystem Microkernels



Filesystems service as a user-space process

- No OS involvement

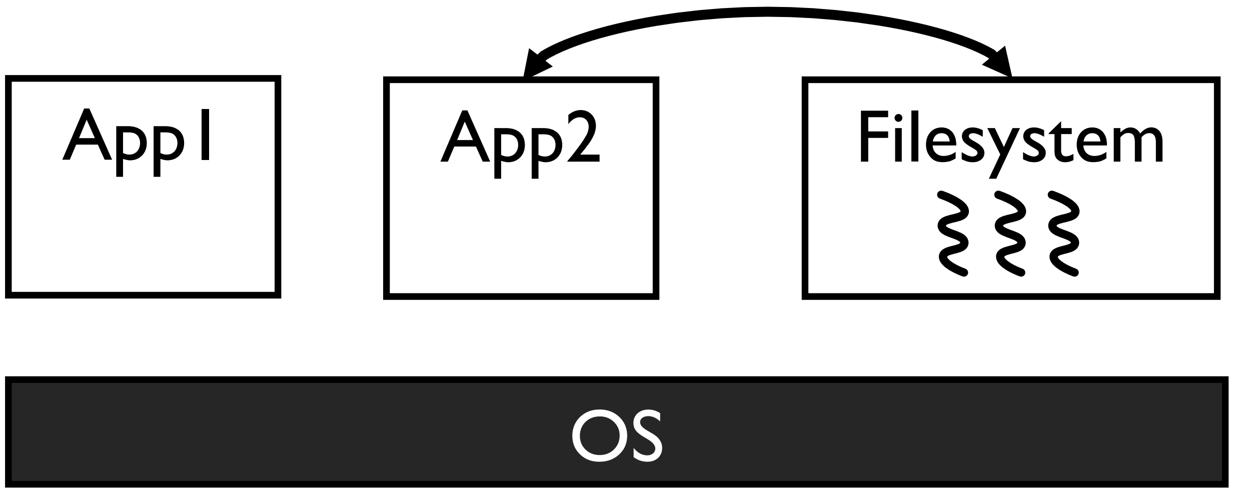
Benefits of Filesystem Microkernels



Filesystems service as a user-space process

- Better performance for modern IO devices and CPUs
- Easy to develop and upgrade
- Better fault isolation

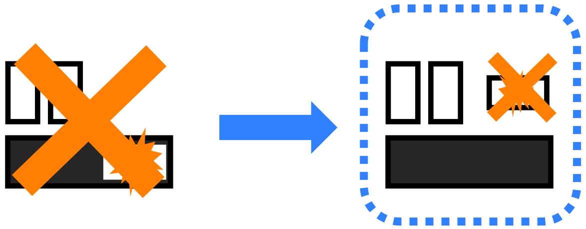
Benefits of Filesystem Microkernels



Filesystems service as a user-space process

- Better performance for modern IO devices and CPUs
- Easy to develop and upgrade
- Better fault isolation

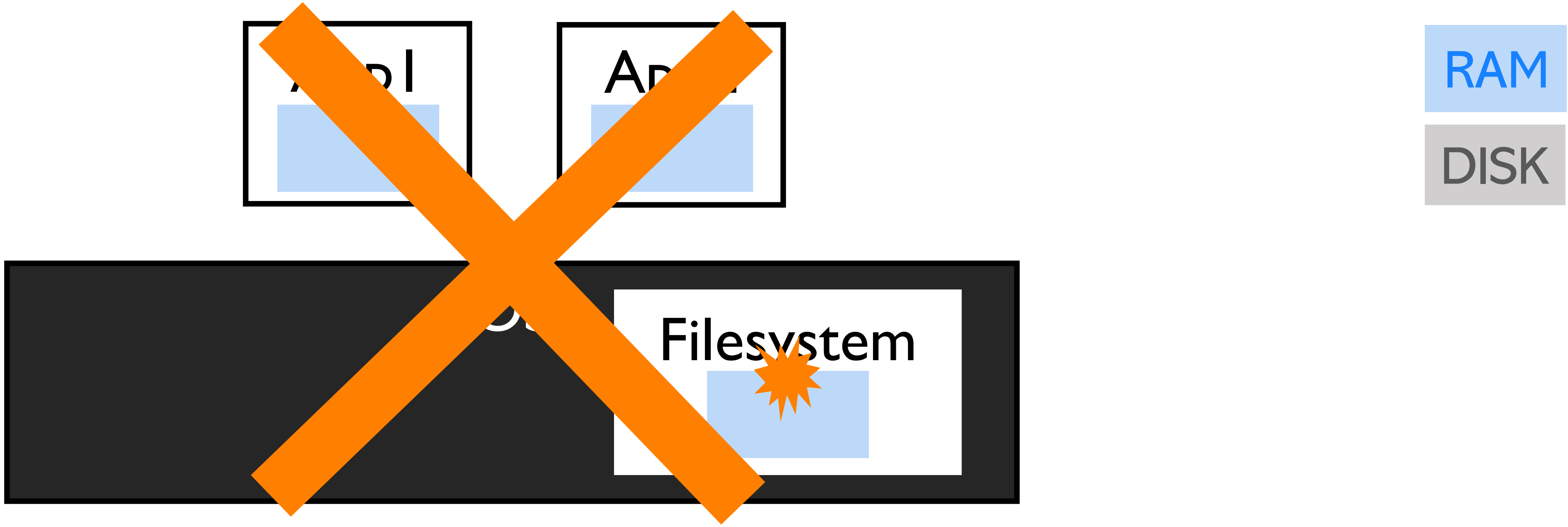
Systems: *uFS* (SOSP '21), *Hongmeng* (OSDI '23)



A new paradigm of crash recovery

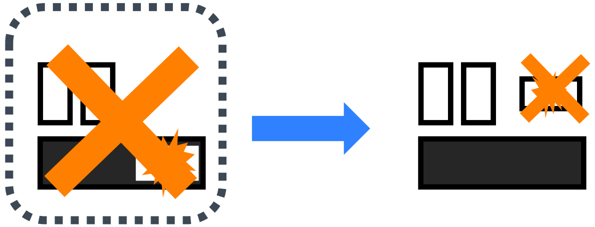
process crash recovery

Kernel Filesystem Failure ➡ Full-system Crash

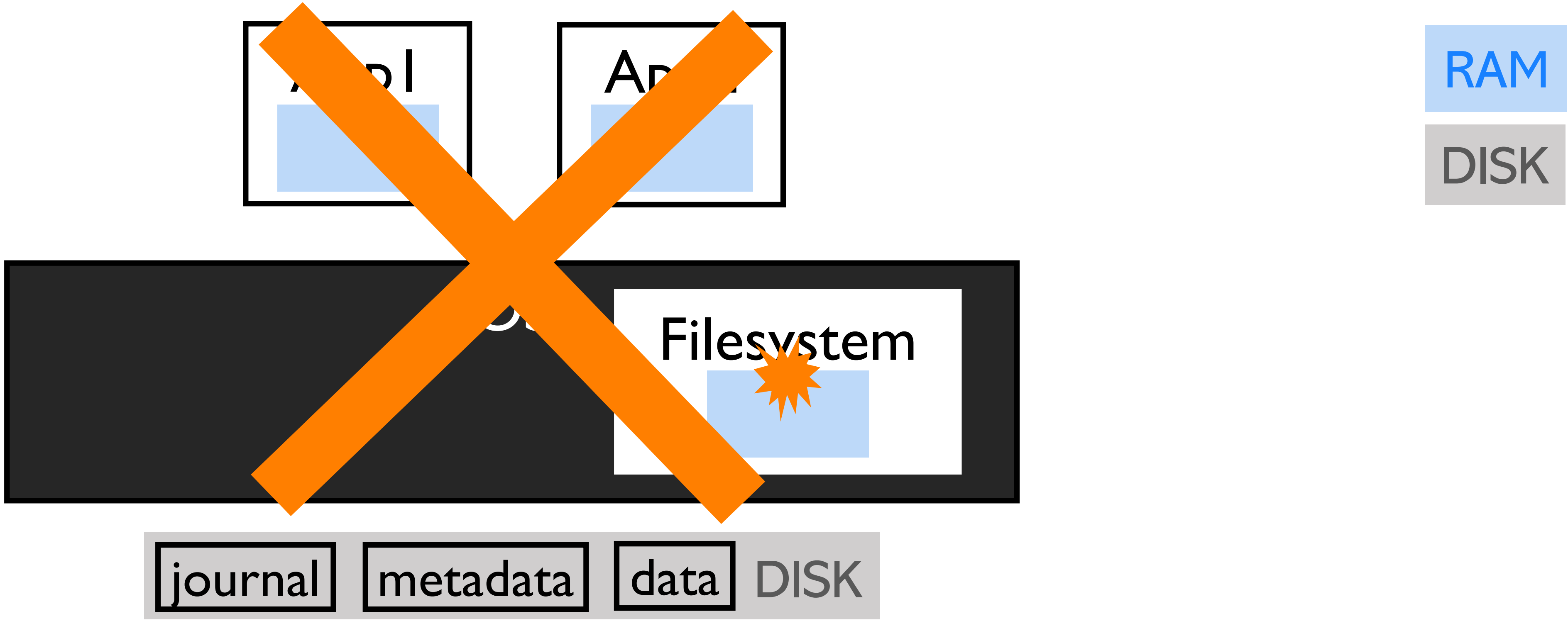


OS and all applications crash (i.e., full-system crash)

Applications also lose their progress

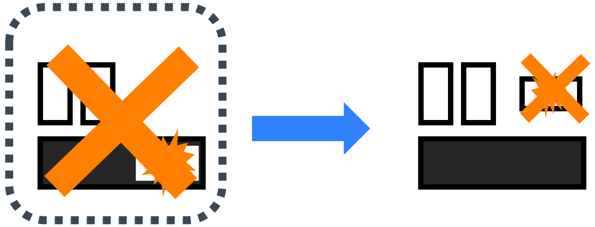


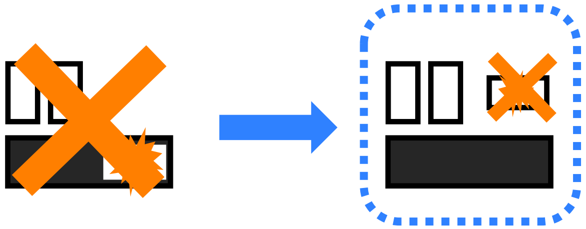
Kernel Filesystem Failure ➡ Full-system Crash



Filesystem crash is treated the same as power failure

Full-system crash recovery only utilizes on-disk states

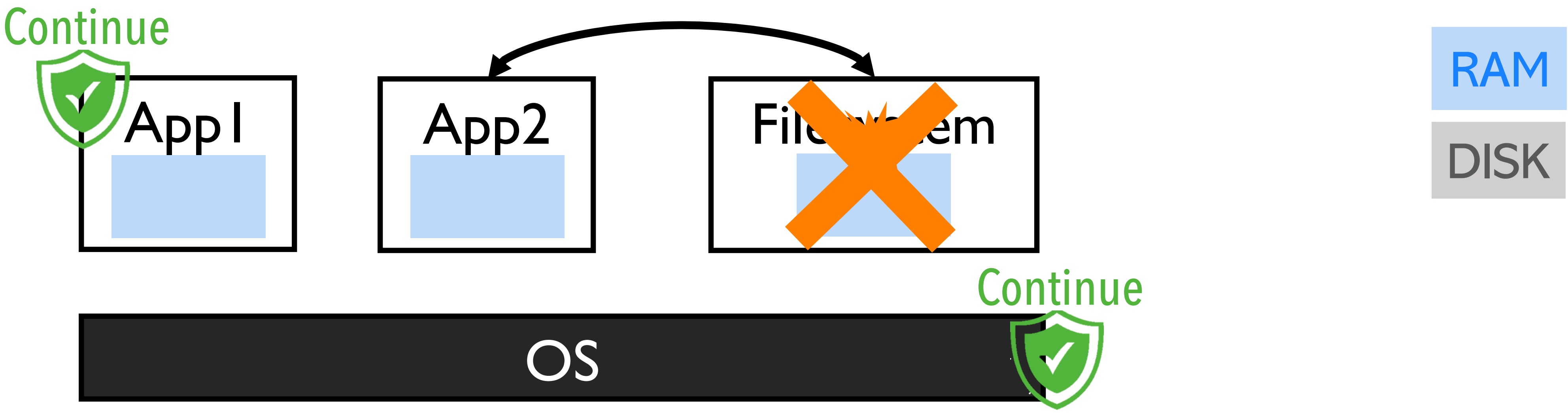




A new paradigm of crash recovery

process crash recovery

Microkernel Filesystem Failure ➡ Process Crash

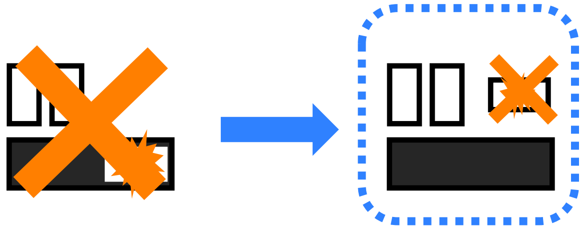


A new crash model

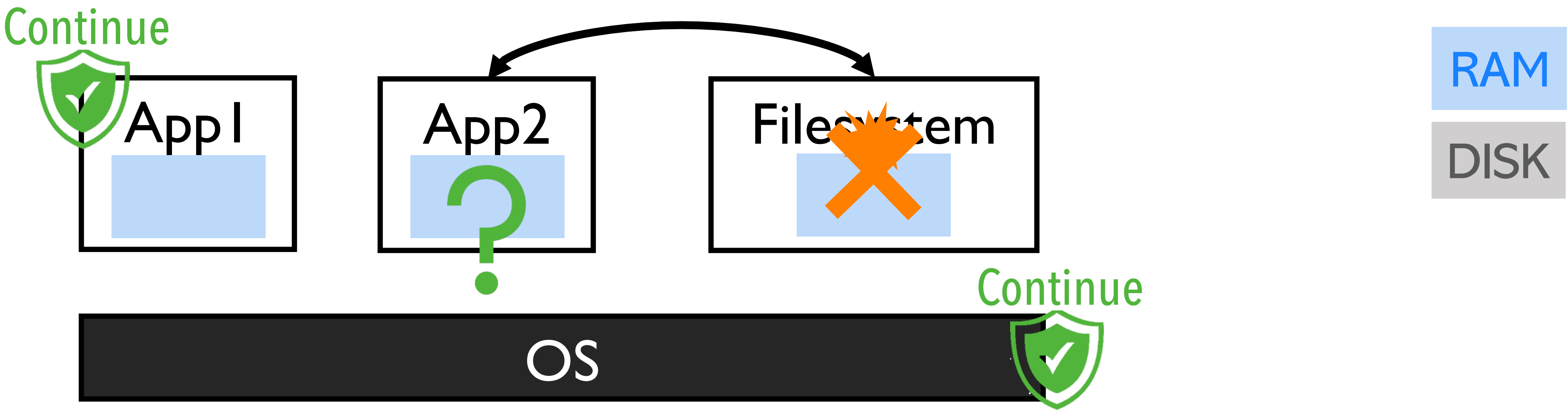
- Process crash, not full-system crash

Opportunities to continue

- Monolithic OS and some apps naturally continue



Microkernel Filesystem Failure ➡ Process Crash



A new crash model

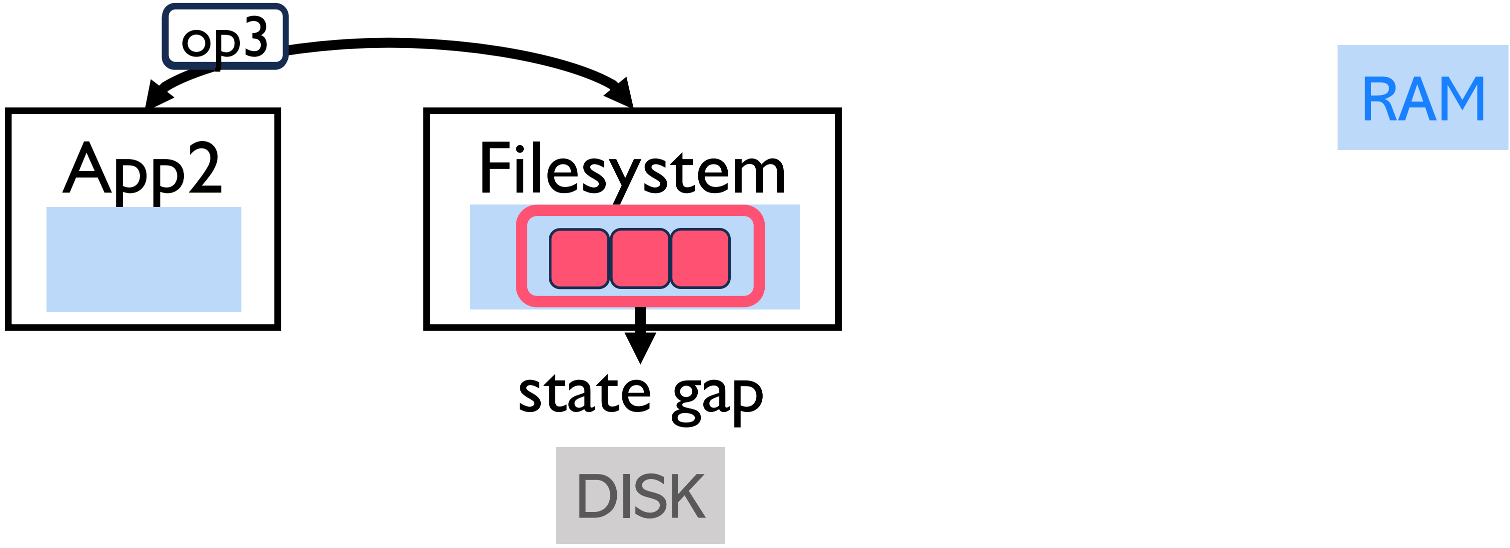
- Process crash, not full-system crash

Opportunities to continue

- Monolithic OS and some apps naturally continue

Can filesystem applications also continue?

Main Challenge: Recover the State Gap

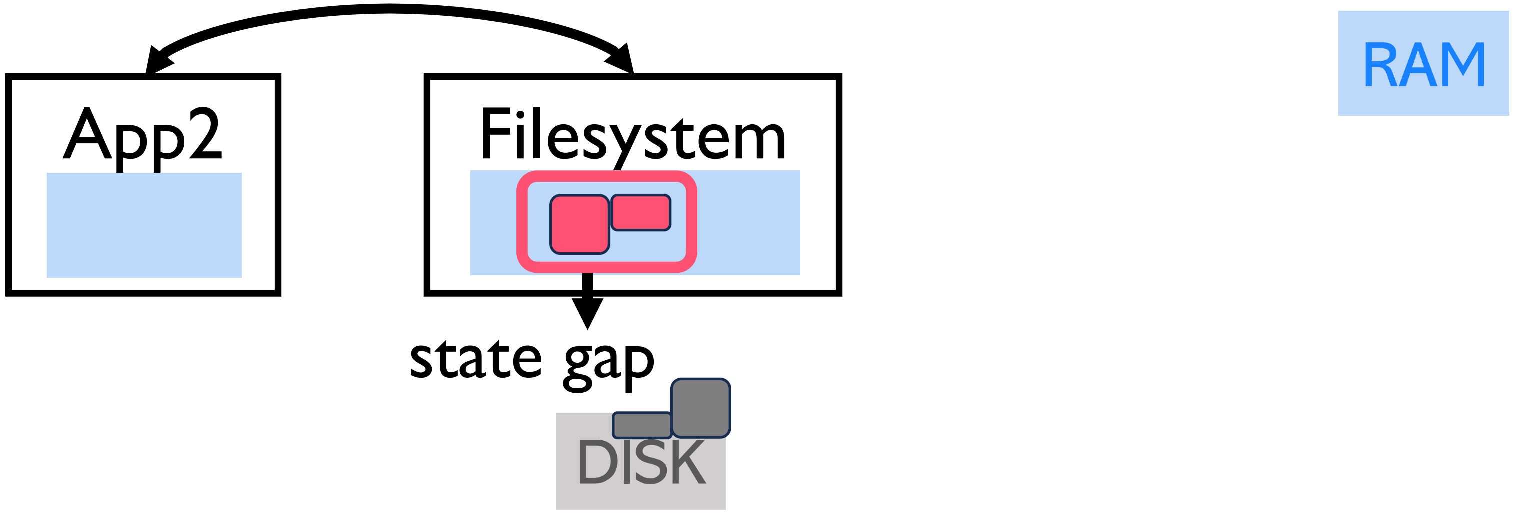


Filesystems buffer update in memory

State gap

- Difference between <u>on-disk states</u> and <u>application view</u>

Main Challenge: Recover the State Gap

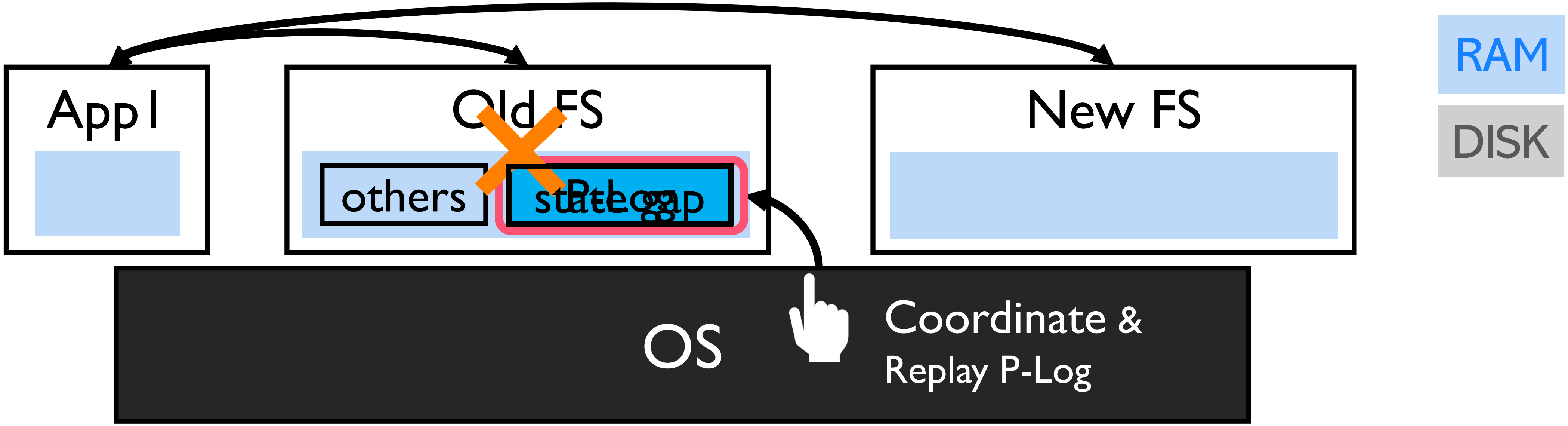


Filesystems buffer update in memory

State gap

- Difference between on-disk states and application view
- **Changes erratically**

Ananke: A Filesystem Microkernel that Supports Process Crash Recovery



P-Log: In-memory Data Structure

- Log the operations and other information

Ananke: A Filesystem Microkernel that Supports Process Crash Recovery

Novel mechanisms

- P-Log and AIM algorithm
- Kernel-coordinated speculative restart
- Lightweight detection of corruption

Ananke: A Filesystem Microkernel that Supports Process Crash Recovery

Novel mechanisms

- P-Log and AIM algorithm
- Kernel-coordinated speculative restart
- Lightweight detection of corruption

Implemented in uFS, a state-of-the-art filesystem microkernel

- Add ~4K LoC

Ananke: A Filesystem Microkernel that Supports Process Crash Recovery

Achieved fast and transparent recovery

- Failure transparency
  - Over 30,000 fault injection experiments
- Low common-path overhead
  - <2% in most cases
- Fast recovery
  - <400ms even for challenging workloads

Outline

- Introduction
- Challenges
- P-Log and AIM
- Evaluation
- Conclusion

Ananke: A Filesystem Microkernel that Supports Process Crash Recovery

Principled process crash recovery

- Challenges
  - Recover State Gap
  - Low Overhead
  - Robustness of Recovery

Ananke: A Filesystem Microkernel that Supports Process Crash Recovery

Principled process crash recovery

- Challenges

Novel mechanisms

- P-Log and AIM algorithm
- Kernel-coordinated speculative restart
- Lightweight detection of corruption

Recover State Gap

Low Overhead

Robustness of Recovery

Ananke: A Filesystem Microkernel that Supports Process Crash Recovery

Principled process crash recovery

- Challenges

Novel mechanisms

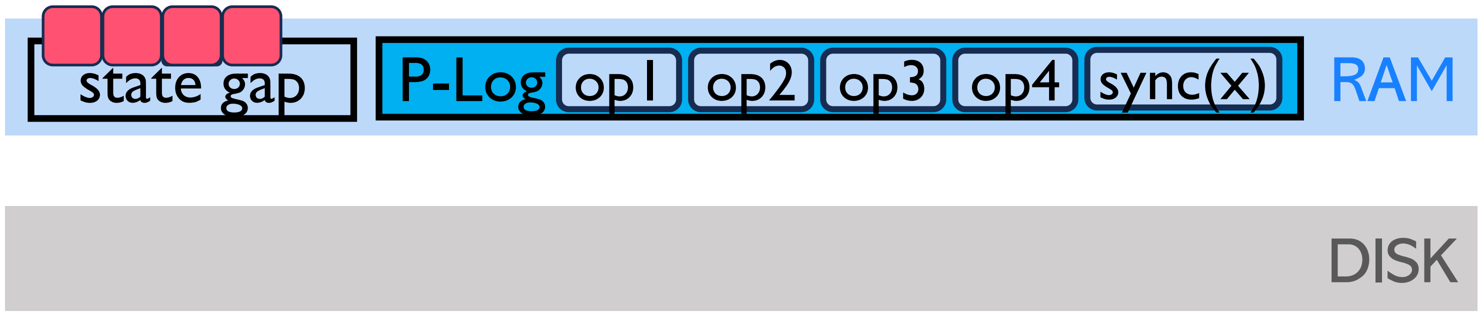
- **P-Log and AIM algorithm**
- Kernel-coordinated speculative restart
- Lightweight detection of corruption

Recover State Gap

Low Overhead

Robustness of Recovery

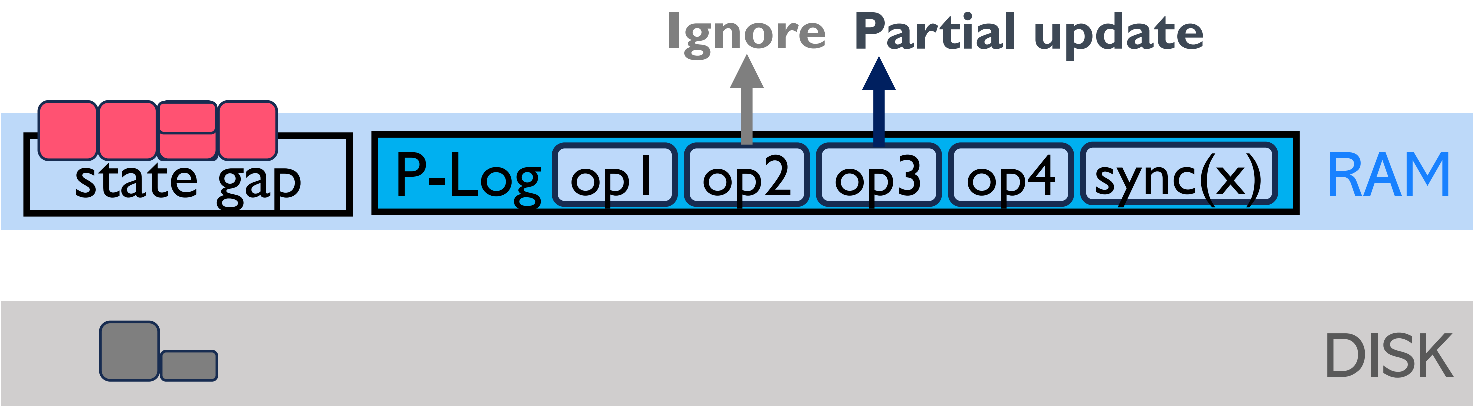
P-Log: In-memory Log for Process Crash Recovery



P-Log

- Log the operations and other information

P-Log: Challenge to Recover the State Gap



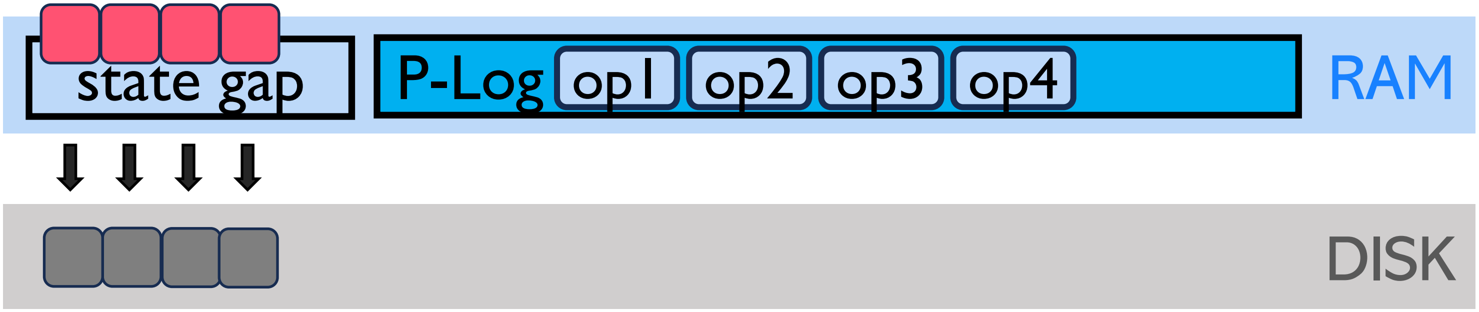
P-Log

- Log the operations and other information
- **Naively replaying is not sufficient**
  - An operation's update may have been durable
  - Part of an operation's update needs to be recovered

Trade-off: Fast, Transparent Recovery vs. Low Common-path Overhead

Incorrect due to the loss of state gap
Manual efforts and long recovery time

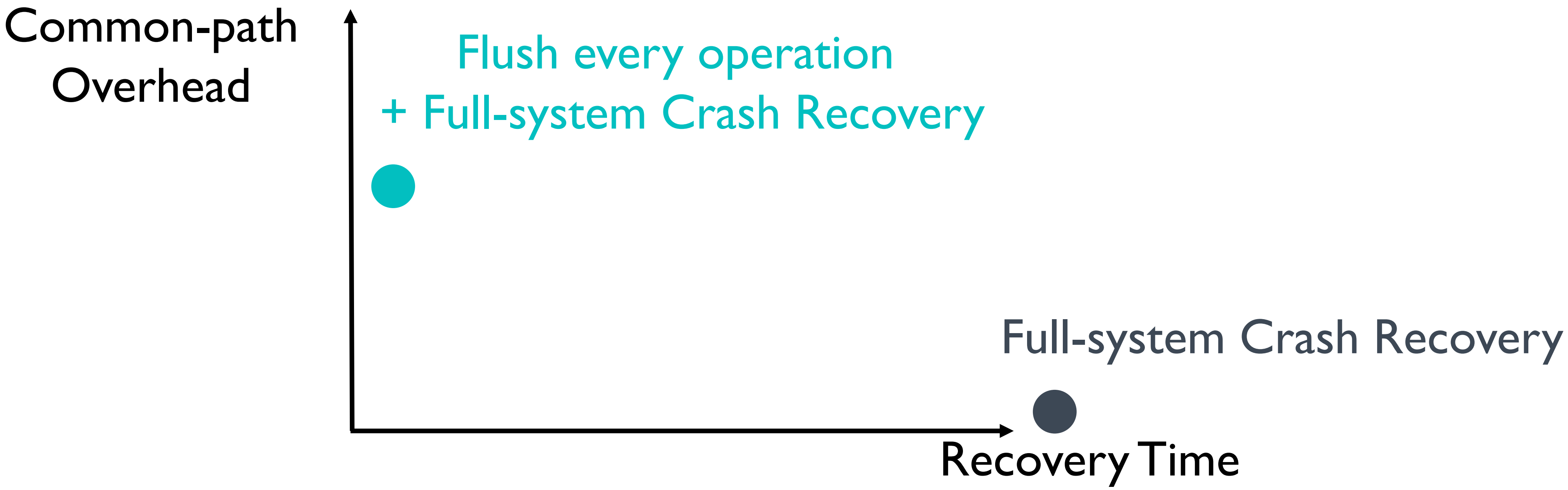
P-Log: Challenge to Recover the State Gap



P-Log

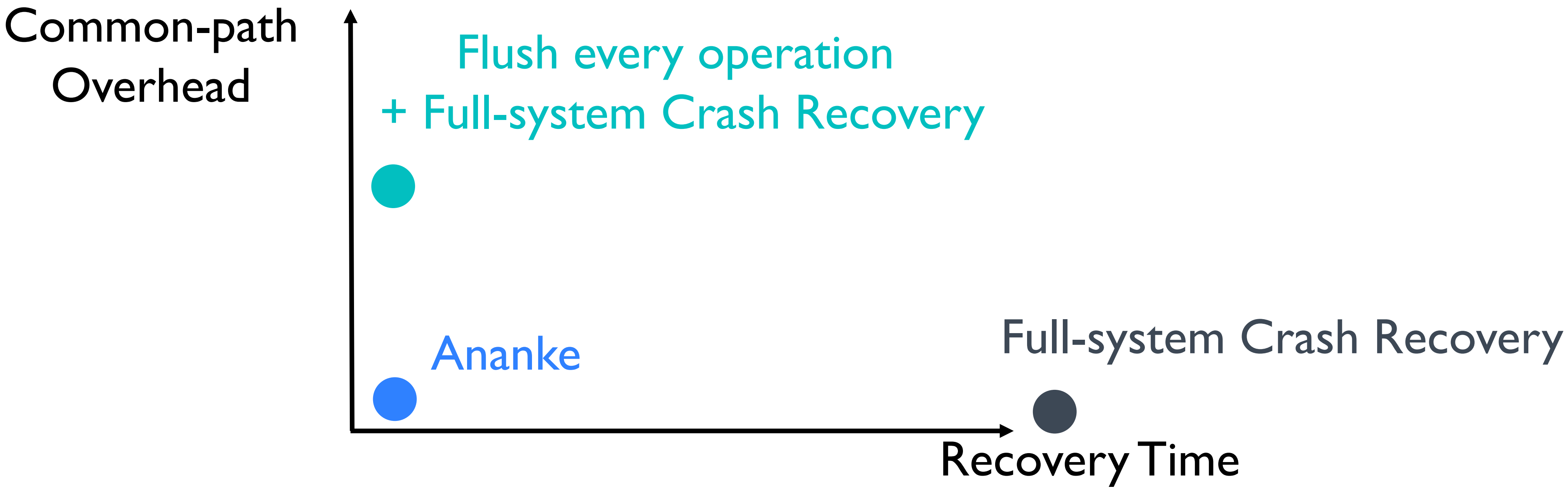
- Log the operations and other information
- **Control the common-path overhead**
  - Extra flushes can simplify the state gap, but incurs large overhead

Fast, Transparent Recovery AND Low Common-path Overhead



Large common-path overhead

Fast, Transparent Recovery AND Low Common-path Overhead

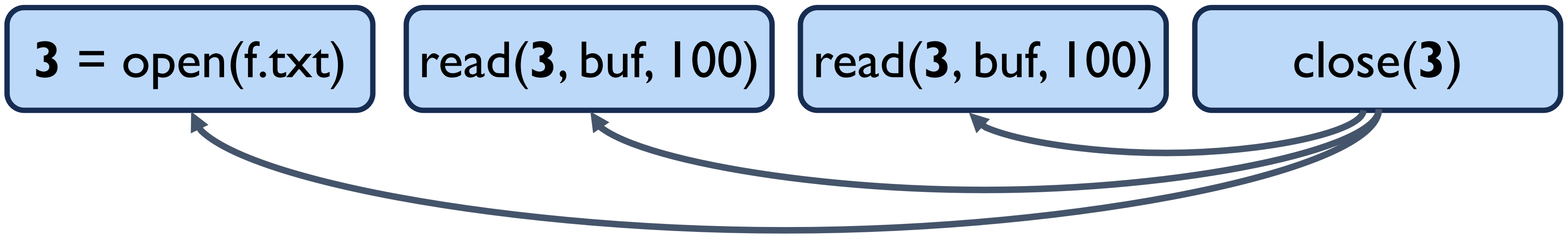


P-Log and AIM: recover the state gap without incurring extra flushes

Recover the Exact State Gap

❶ Subsequent operations may remove some (or all) of the changes from the state gap

- close() removes fd



remove the **fd**, and thus updates of several previous operations

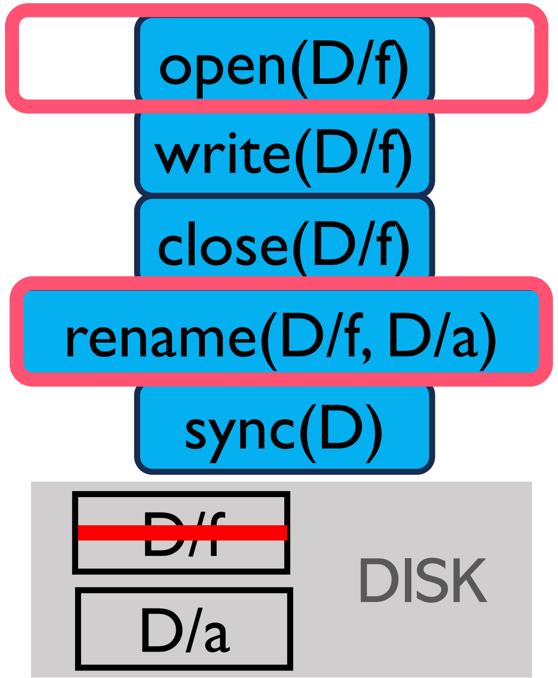
Recover the Exact State Gap

❶ Subsequent operations may remove some (or all) of the changes from the state gap

- close() removes an fd
- fsync()/sync()/background sync

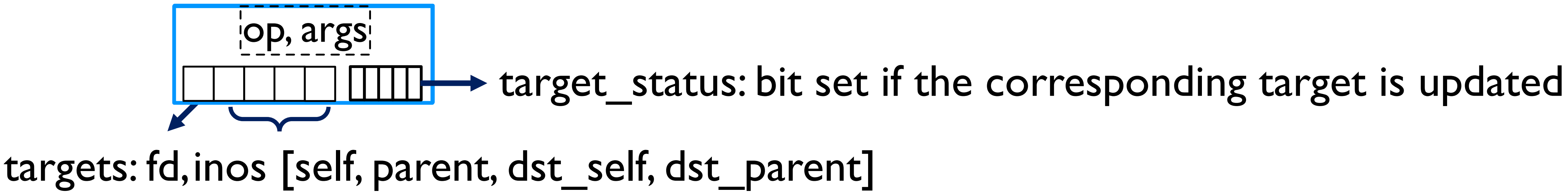
Recover the Exact State Gap

1. Subsequent operations may remove some (or all) of the changes from the state gap
2. Subsequent operations may alter the preconditions



Re-execute: open(D/f) cannot find it

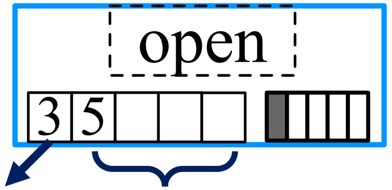
P-Log: In-memory Per-Core Log



A P-Log entry contains an array to record possible targets

- file descriptor and involved inodes (inos)

P-Log: In-memory Per-Core Log

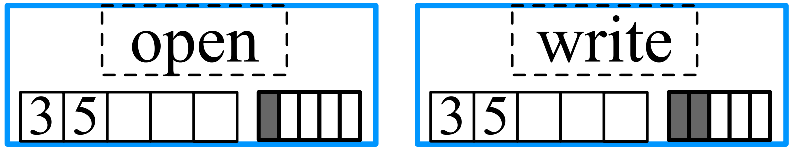


targets: fd, inos [self, parent, dst_self, dst_parent]

Workload

- 3 = open(f), write(3, “xx"), fsync(3), close(3)
- (file inode number = 5)

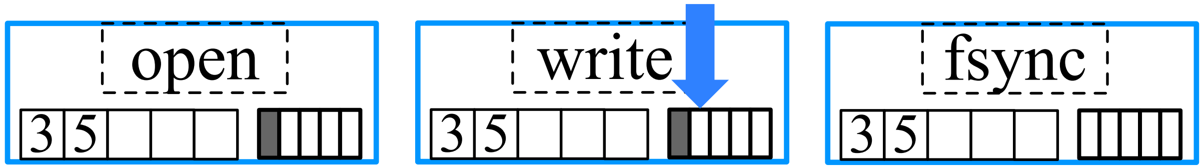
P-Log: In-memory Per-Core Log



Workload

- 3 = open(f), write(3, xxx), fsync(3), close(3)
- (file inode number = 5)

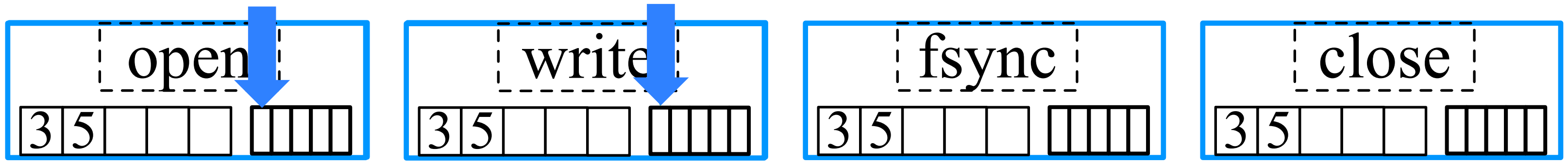
P-Log: In-memory Per-Core Log



Workload

- 3 = open(f), write(3, xxx), fsync(3), close(3)
- (file inode number = 5)

P-Log: In-memory Per-Core Log



Workload

- 3 = open(f), write(3, xxx), fsync(3), close(3)
- (file inode number = 5)

AIM (Act, Ignore, Modify) A I M

Transform P-Log entries into actions that can be executed by the original filesystem implementation

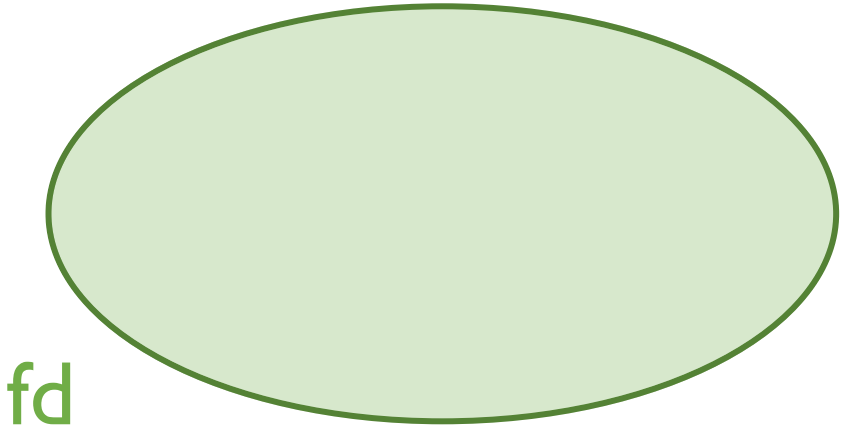
Ignore or not

- Ignore: a logged operation needs to be ignored

Act or Modify

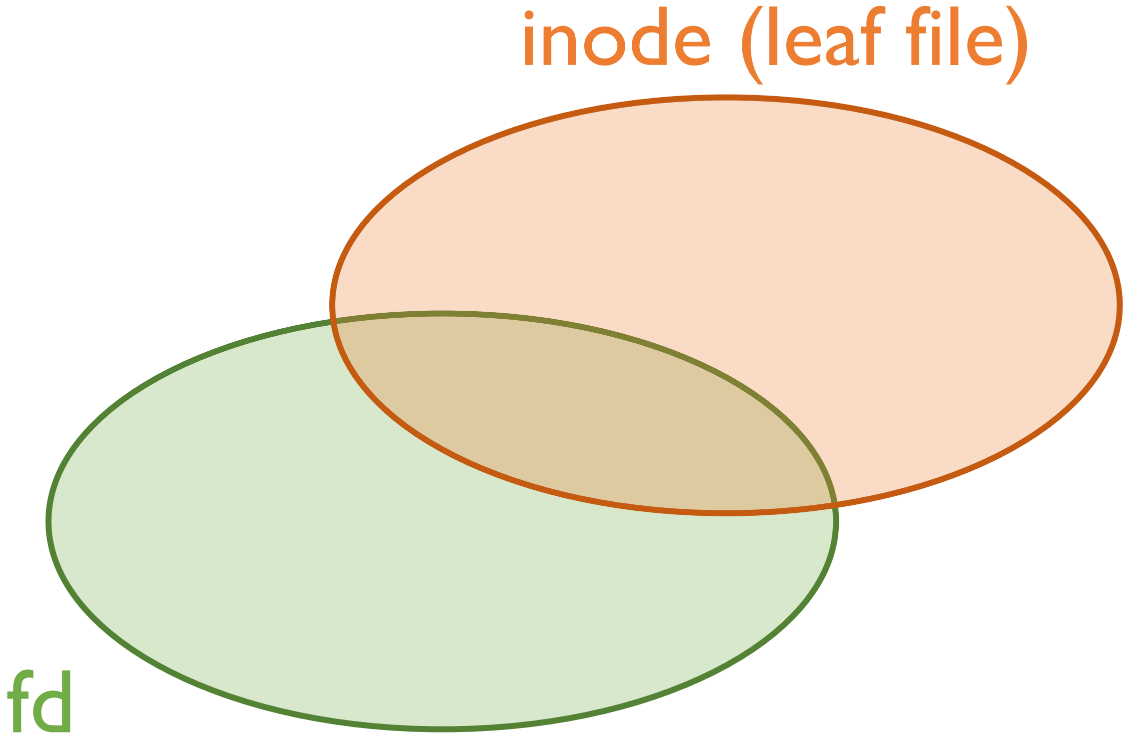
- Act: an operation will be directly replayed
- Modify: needs to take actions for an operation, but in a modified form
  - E.g., another operation type, different parameters

Intuition behind P-Log and AIM



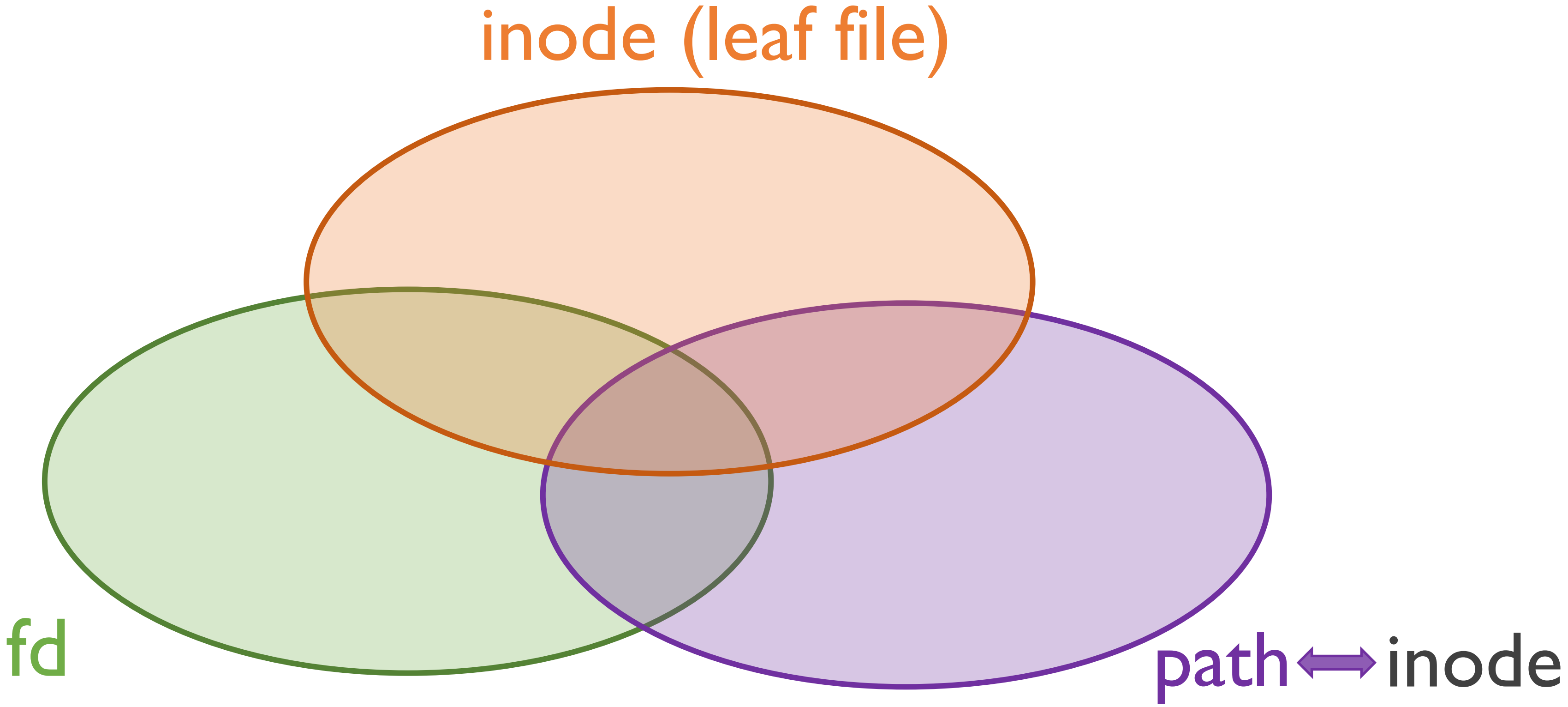
Applications change filesystem states upon three abstractions

Intuition behind P-Log and AIM



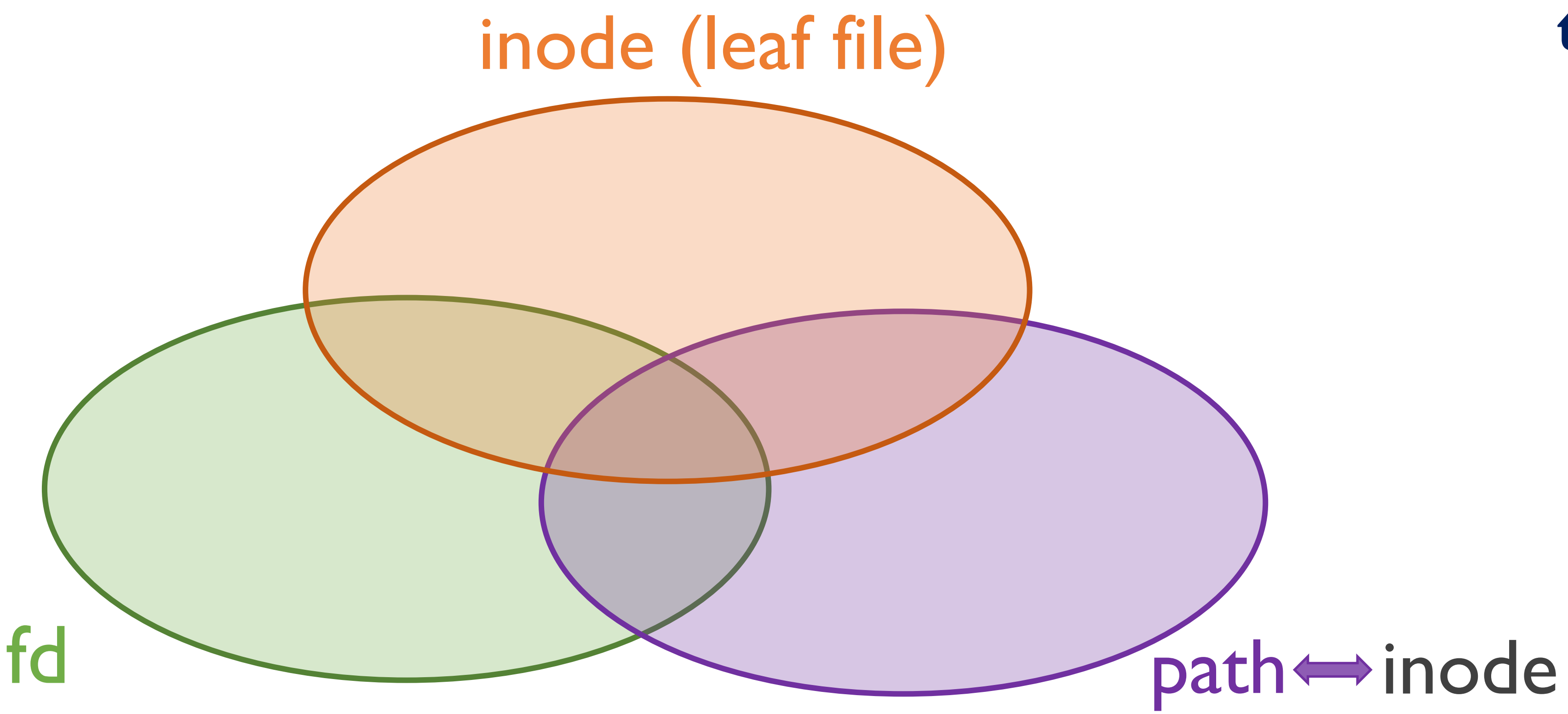
Applications change filesystem states upon three abstractions

Intuition behind P-Log and AIM



Applications change filesystem states upon three abstractions

Intuition behind P-Log and AIM

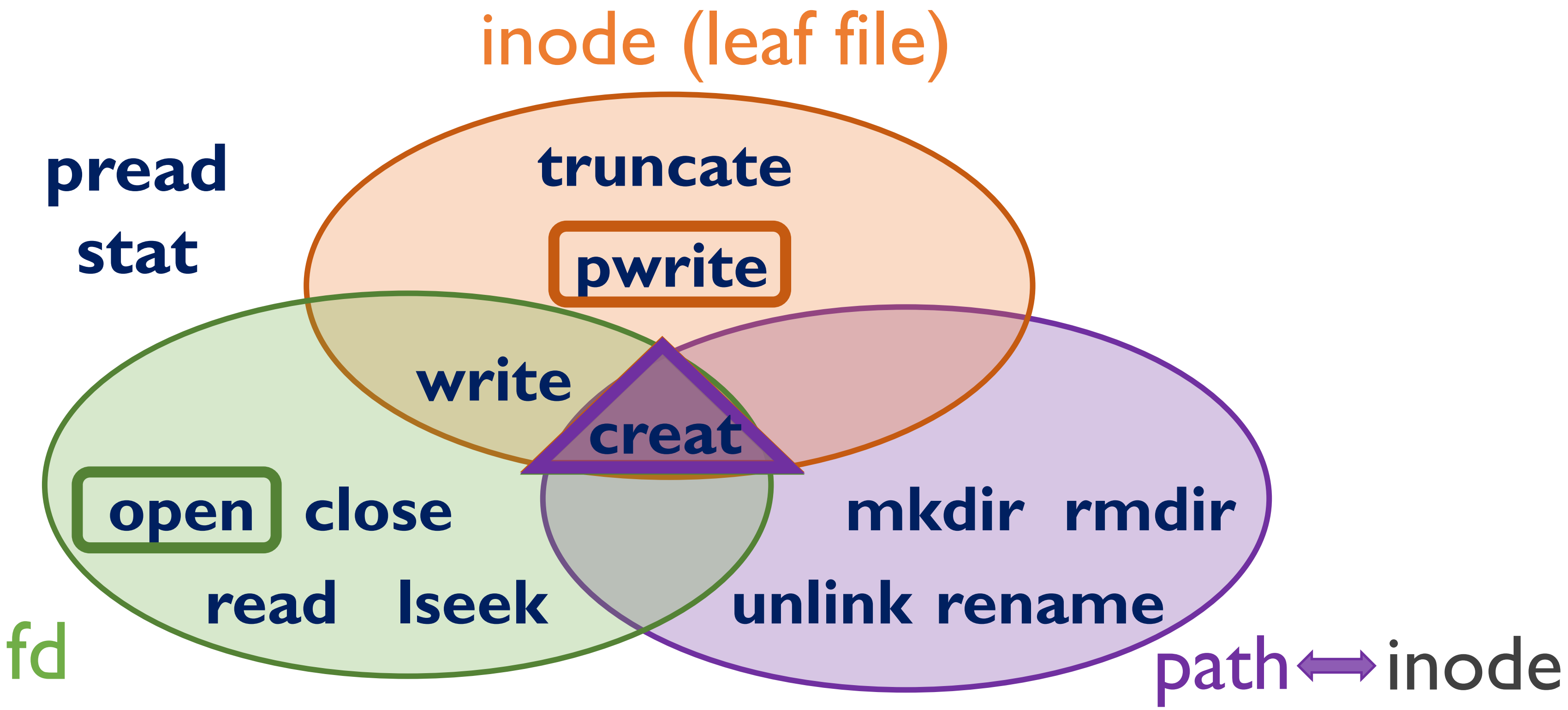


stat
pread
truncate
write
creat
mkdir
rmdir
unlink
rename
open
close
read
lseek

Applications change filesystem states upon three abstractions

- Use Filesystem APIs

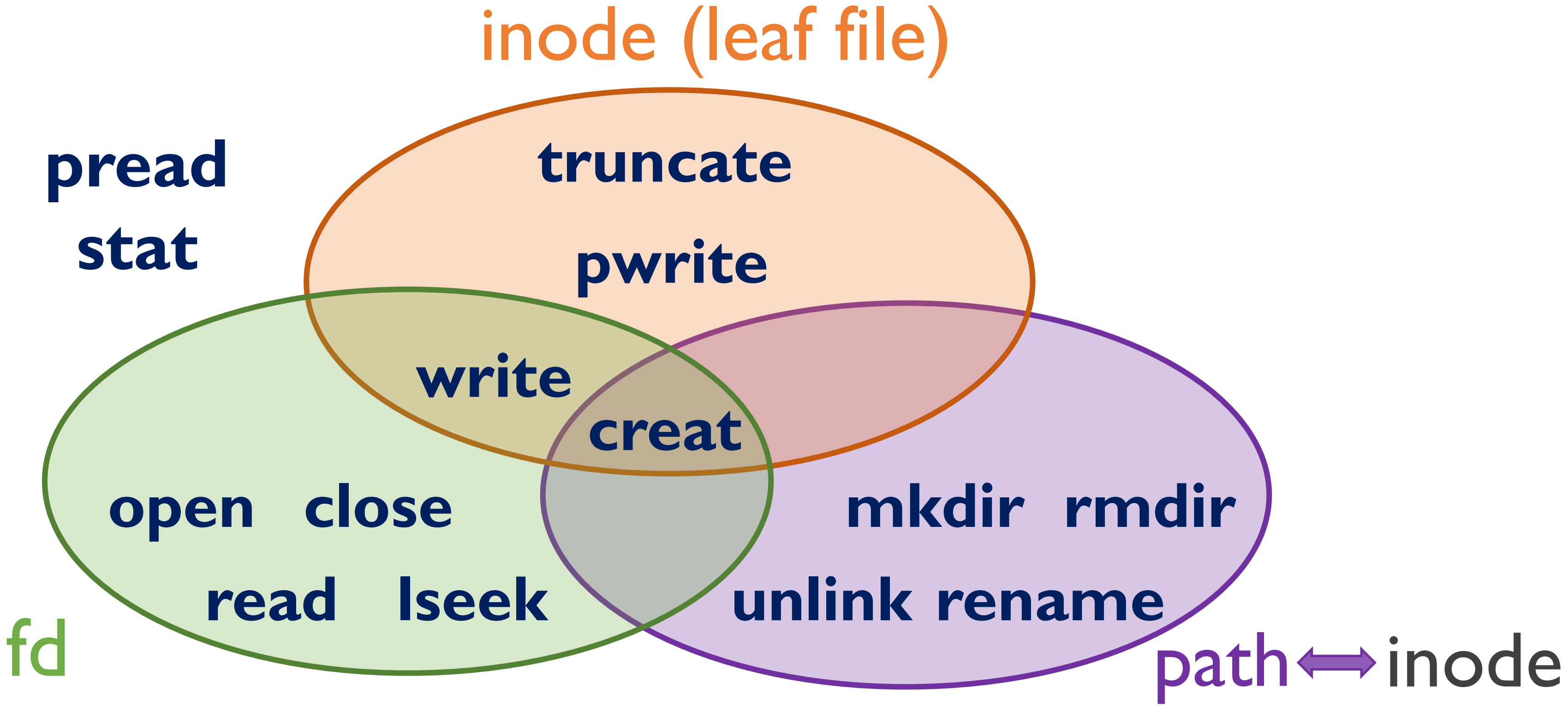
Intuition behind P-Log and AIM

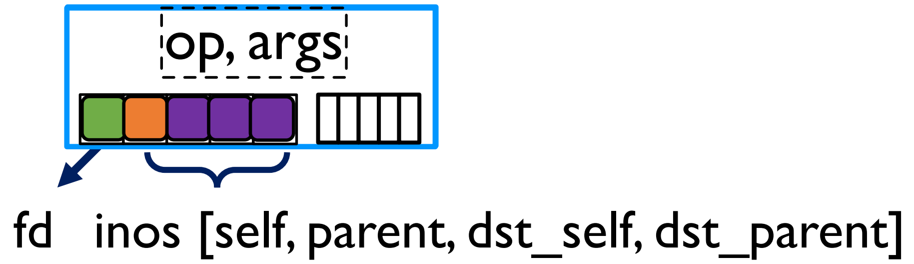


Applications change filesystem states upon three abstractions

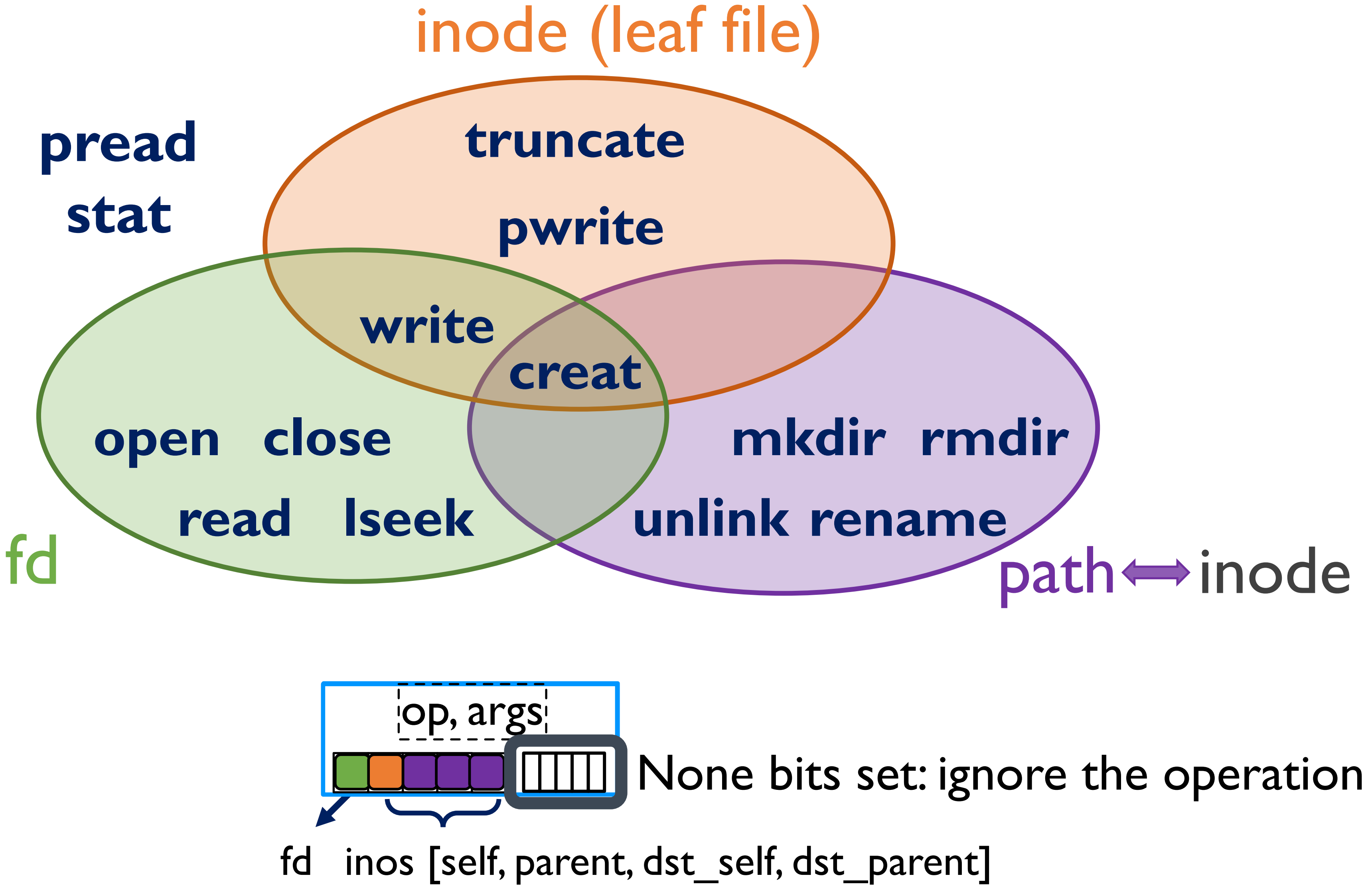
- Use Filesystem APIs

Checking the Bit Can Decide: How Much of an Operation's Update are in the State Gap





Checking the Bit Can Decide: How Much of an Operation's Update are in the State Gap



AIM (Act, Ignore, Modify) A I M

Transform p-log into actions that can be executed by the original filesystem implementation

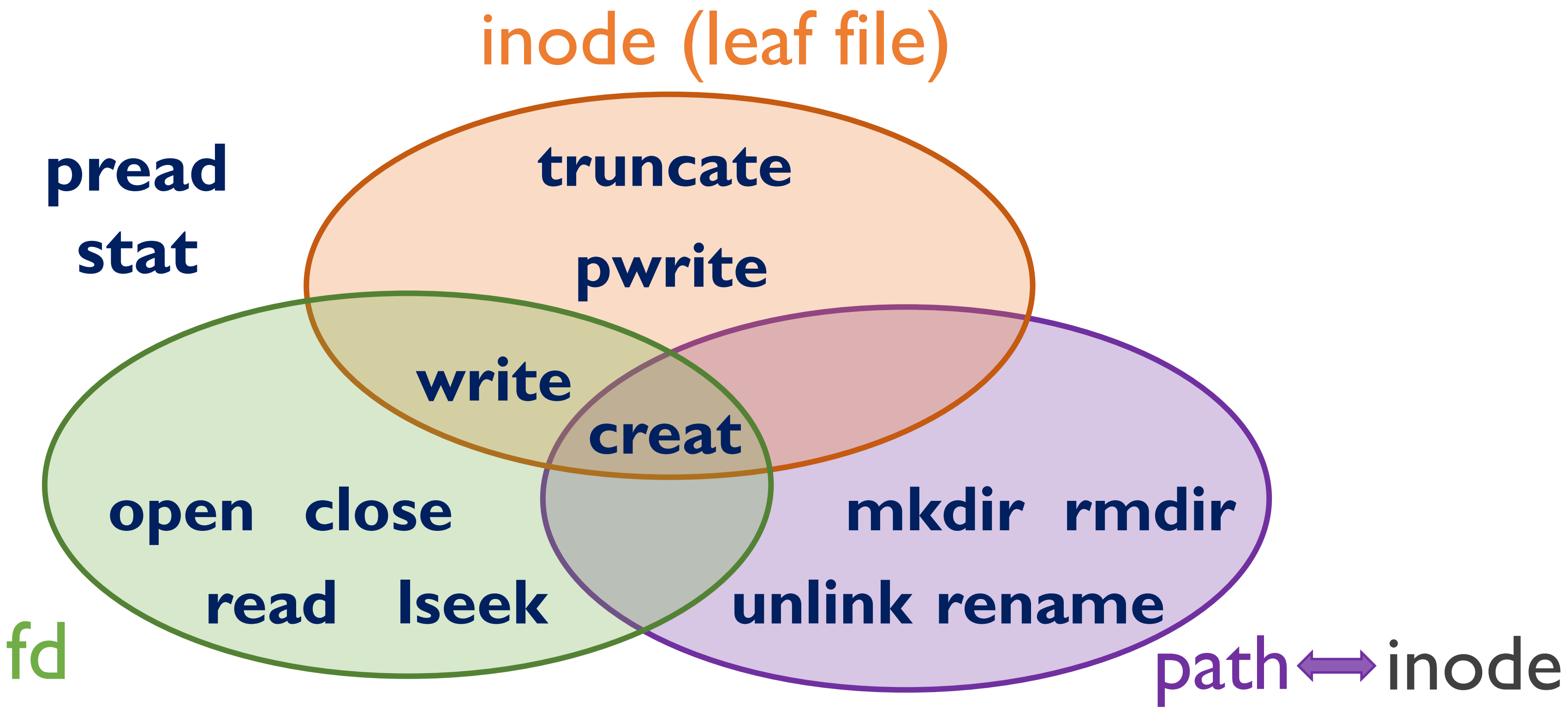
Ignore or not (Checking the bits & Fast in the common path)

- Ignore: a logged operation needs to be ignored

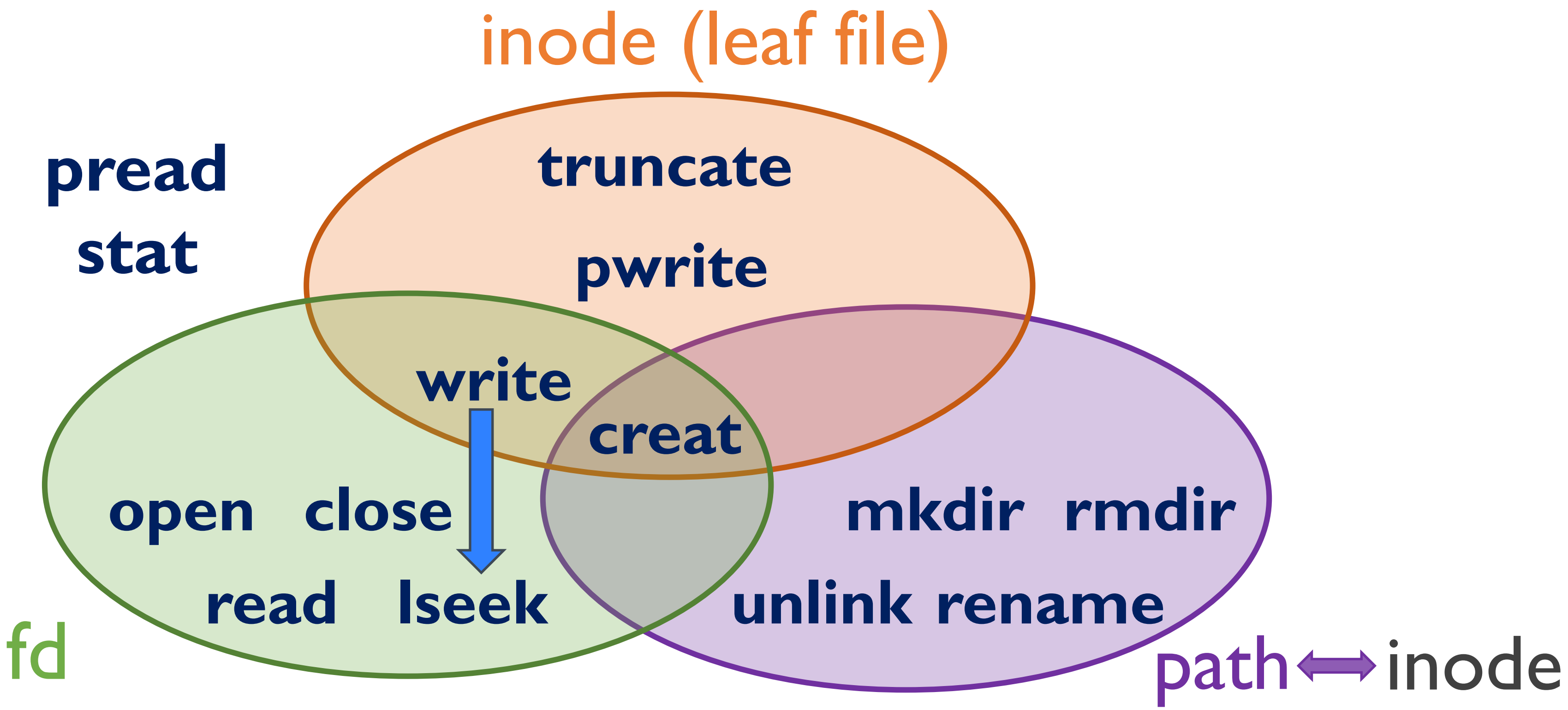
Act or Modify

- Act: an operation will be directly replayed
- Modify: needs to take actions for an operation, but in a modified form

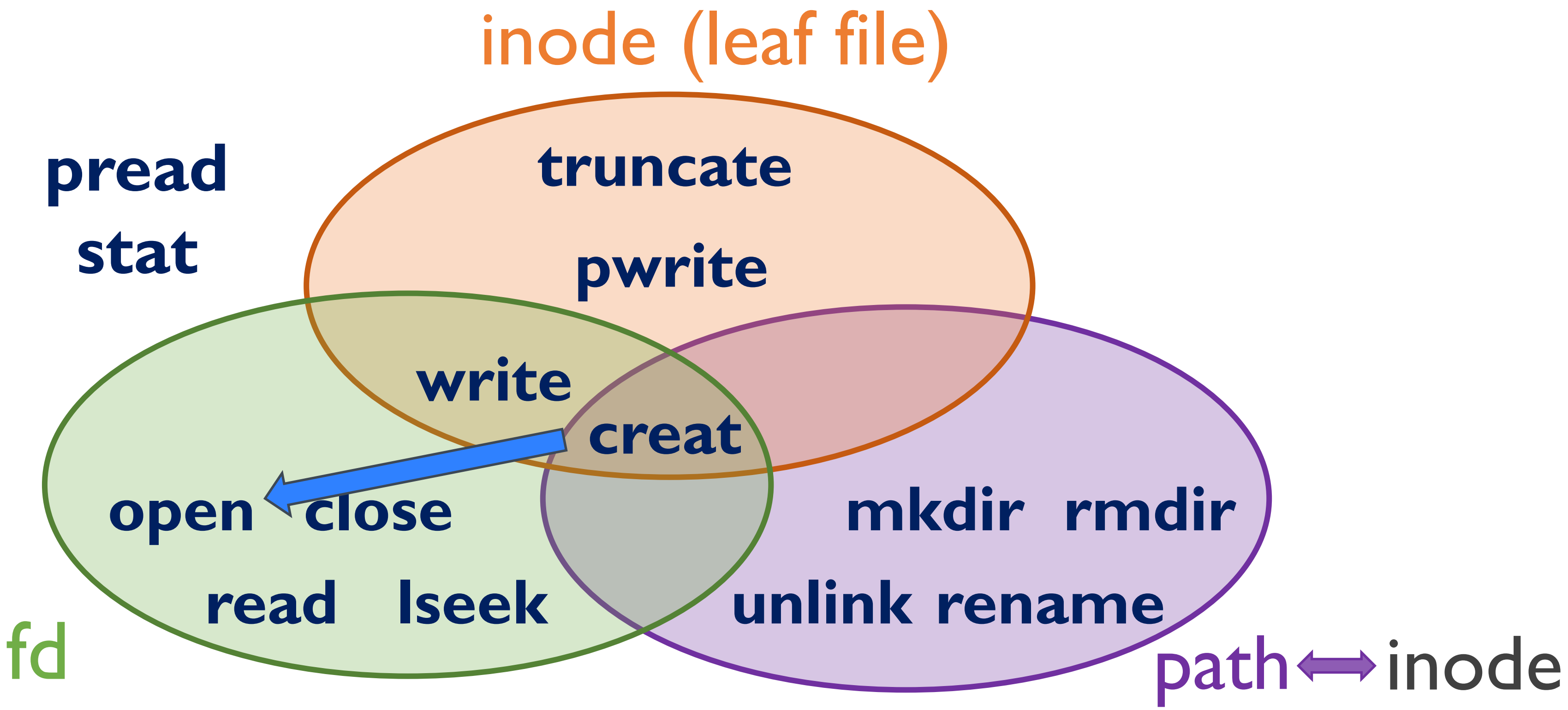
Modify: Decide the Resulting Form during Recovery



Modify: Decide the Resulting Form during Recovery



Modify: Decide the Resulting Form during Recovery



AIM (Act, Ignore, Modify) A I M

Transform P-Log entries into actions that can be executed by the original filesystem implementation

Ignore or not (Checking the bits & Fast in the common path)

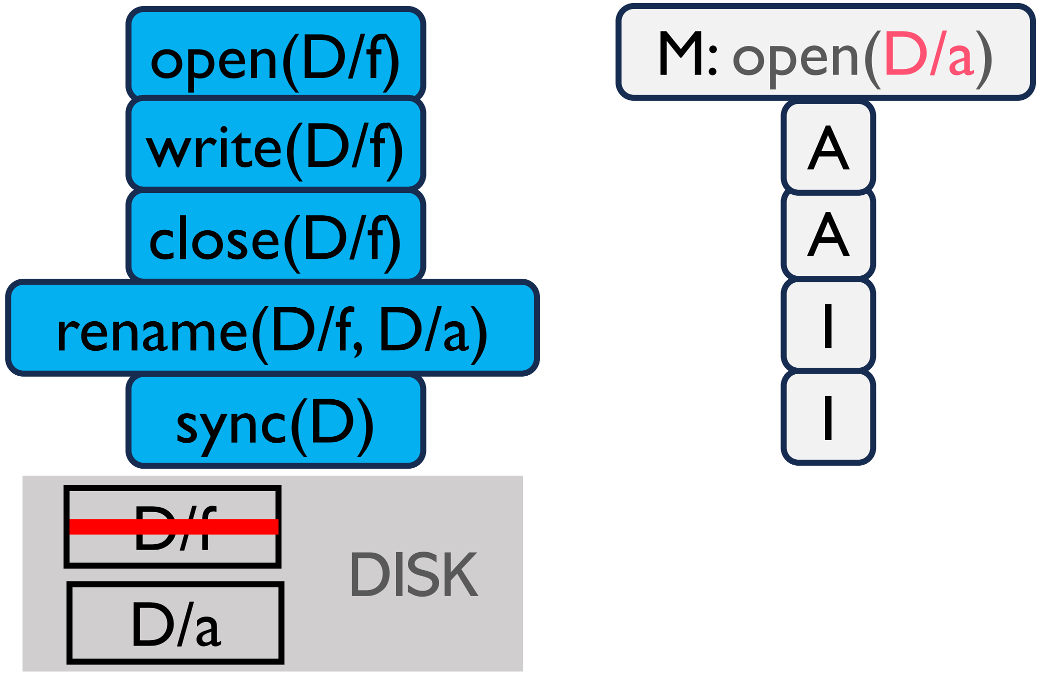
- Ignore: a logged operation needs to be ignored

Act or Modify (During recovery)

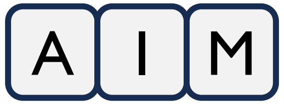
- Act: an operation will be directly replayed
- Modify: needs to take actions for an operation, but in a modified form

AIM (Act, Ignore, Modify) A I M

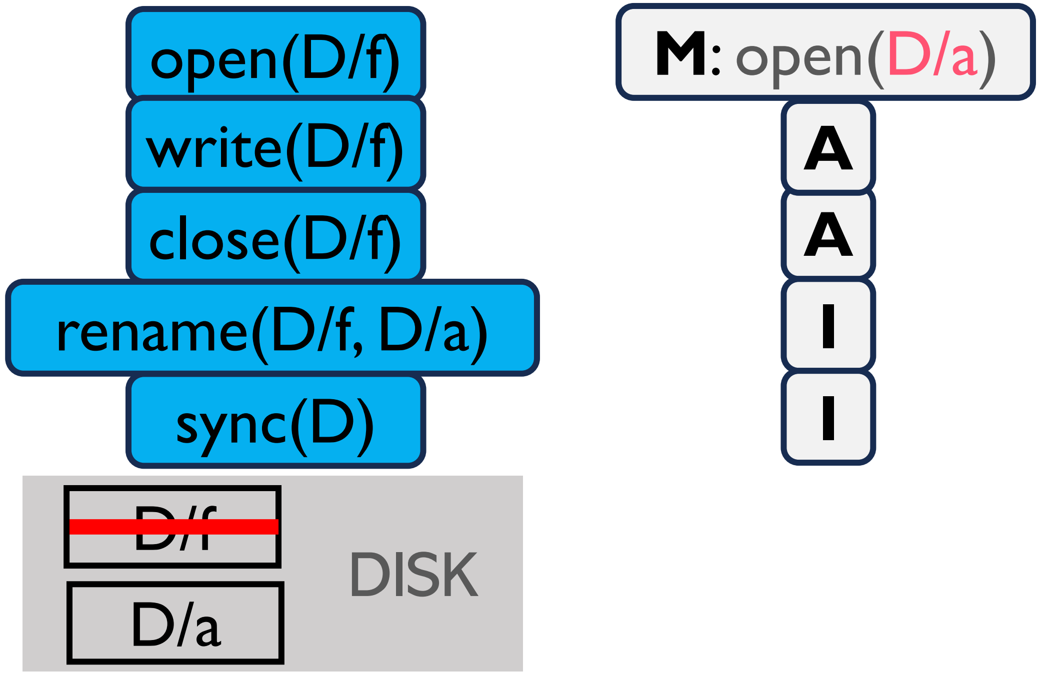
1. Subsequent operations may remove some (or all) of the changes from the state gap
2. Subsequent operations may alter the preconditions



AIM (Act, Ignore, Modify)



1. Subsequent operations may remove some (or all) of the changes from the state gap
2. Subsequent operations may alter the preconditions



Outline

- Introduction
- Challenges
- P-Log and AIM
- Evaluation
- Conclusion

Evaluation

Failure transparency

- Over 30,000 fault injection experiments under various applications, covering different state gap
- Over 3,000 Memory corruption experiments

Fast recovery

Low overhead in common-path

- Performance overhead & memory overhead

Empirical Evaluation of Recovering the State Gap

Five real-world applications

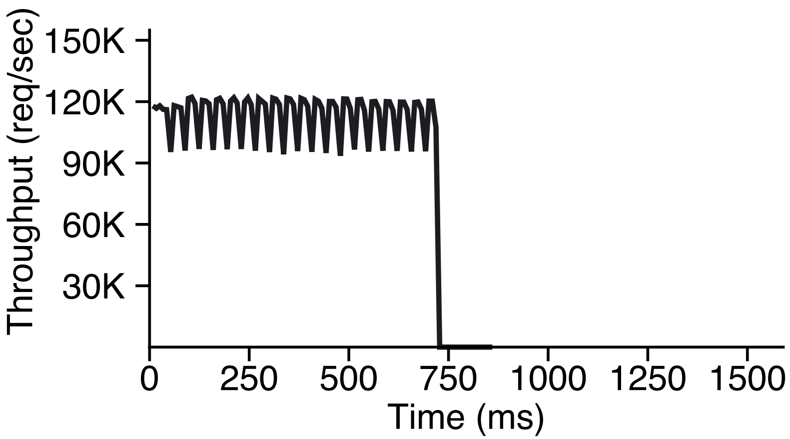
- Sort, copy, unzip, SQLite, LevelDB
- Inject a process crash after each operation in sequences

Over 30,000 cases

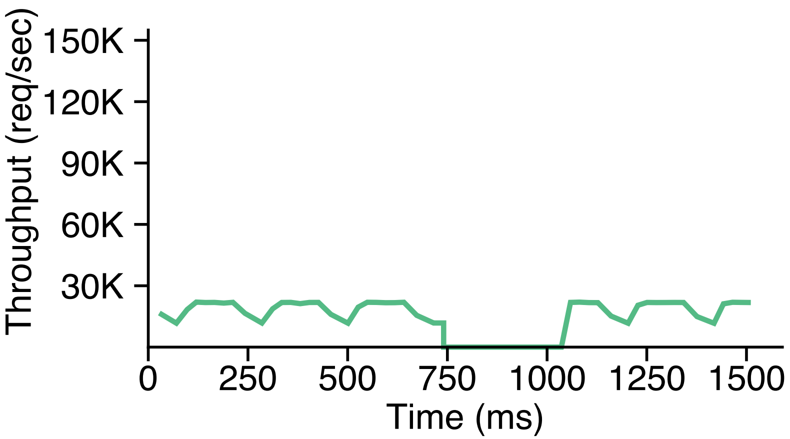
- Provide failure transparency

Fast, Transparent Recovery & Low Overhead During Normal Execution

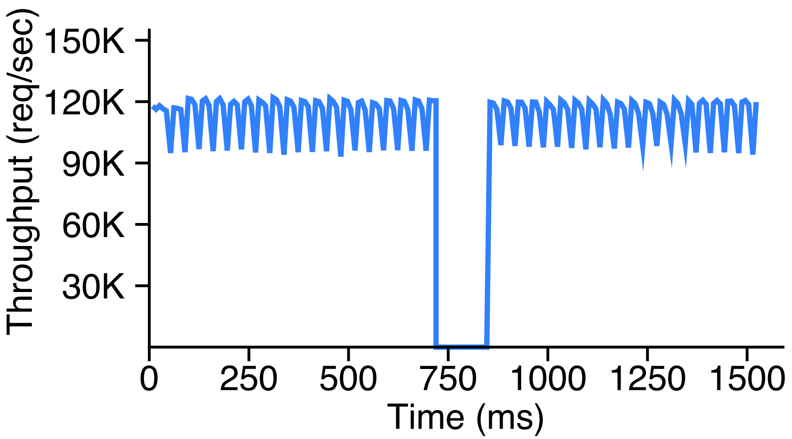
Workload: LevelDB (load)



full-system crash recovery



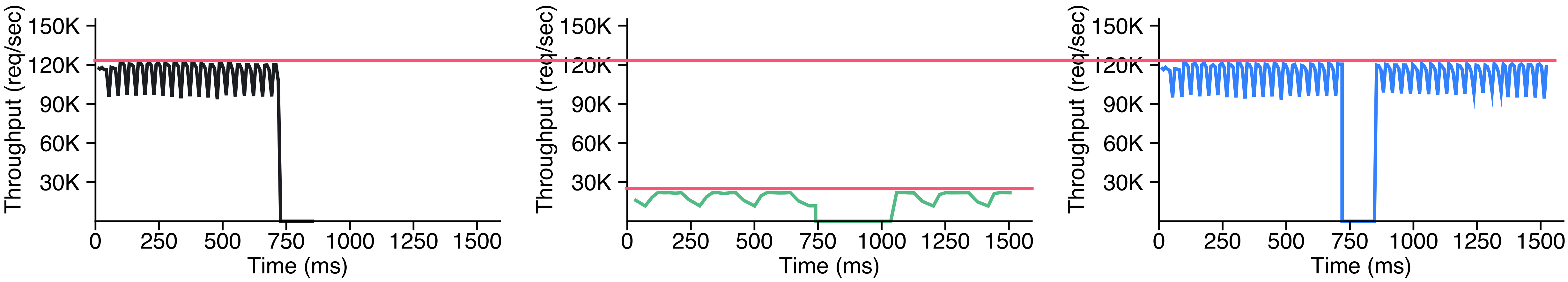
full-system crash recovery + flush every operation



full-system crash recovery + process crash recovery (Ananke)

Low Overhead in the Common Path

Workload: LevelDB (load)



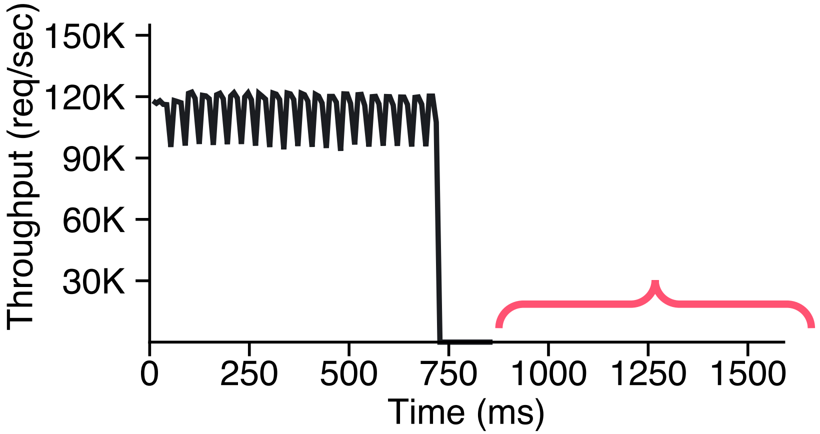
full-system crash recovery

full-system crash recovery + flush every operation

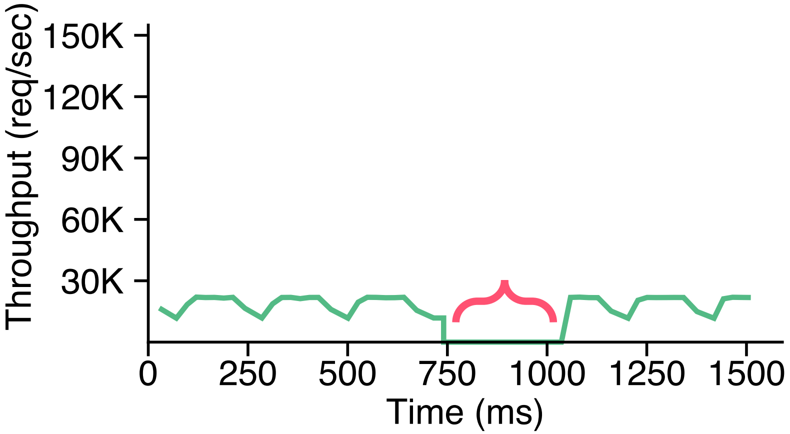
full-system crash recovery + process crash recovery (Ananke)

Fast, Transparent Recovery

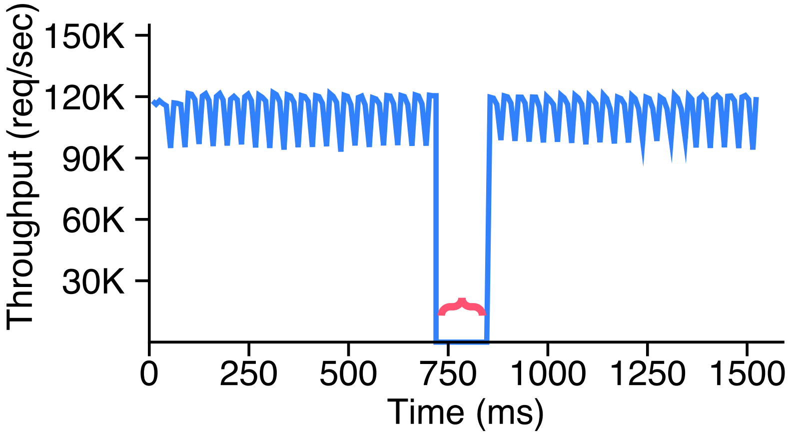
Workload: LevelDB (load)



full-system crash recovery



full-system crash recovery + flush every operation



full-system crash recovery + process crash recovery (Ananke)

Conclusion

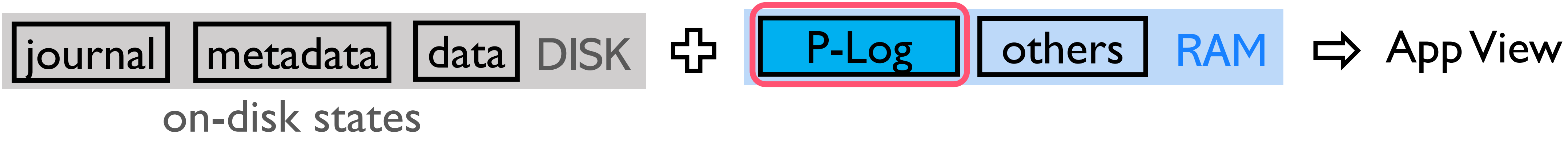
Ananke: fast, transparent filesystem process crash recovery

- Implemented in uFS—a state-of-the-art filesystem microkernel
- Novel mechanisms:
  - P-log and AIM to recover state gap
  - Others to improve recovery performance and robustness
- Thorough evaluation: fault injection and overhead analysis

Conclusion

Ananke: fast, transparent filesystem process crash recovery

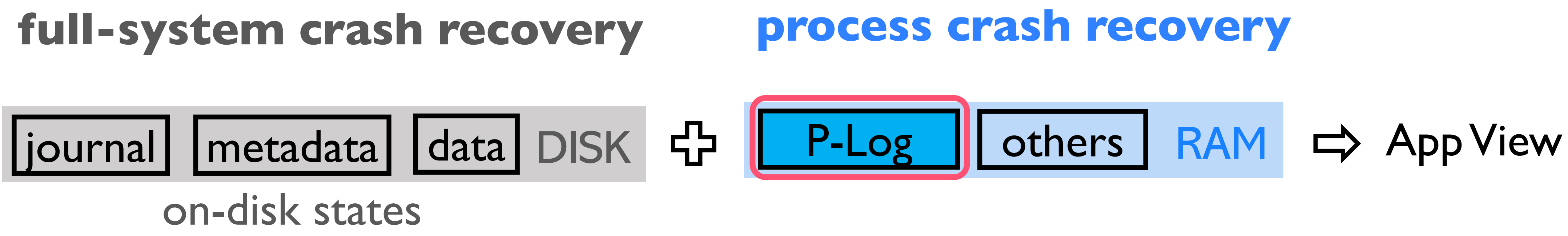
- Implemented in uFS—a state-of-the-art filesystem microkernel
- Novel mechanisms:
  - P-log and AIM to recover state gap
  - Others to improve recovery performance and robustness
- Thorough evaluation: fault injection and overhead analysis



Conclusion

Separate **process crash recovery** from **full-system crash recovery**

- Filesystem process crash is not the same as power-failure
  - Opportunity for transparent recovery
- Improve the guarantee of local filesystem services



Ananke and Process Crash Recovery

See the paper (or email me: jingliu3@microsoft.com) for:

- Principles and challenges for process crash recovery
- Detailed design of P-log and AIM
- Correctness and performance evaluation under various applications

Thank you for listening!