Performance and Protection in the ZoFS User-space NVM File System

Mingkai Dong, Heng Bu, Jifei Yi, Benchao Dong, Haibo Chen

Institute of Parallel and Distributed Systems (IPADS),
Shanghai Jiao Tong University

Oct. 2019 @SOSP ’19

Non-volatile memory (NVM) is coming with attractive features

- Fast
  - Near-DRAM performance
- Persistent
  - Durable data storage
- Byte-addressable
  - CPU load/store access

File systems are designed for NVM

- NVM File systems in kernel
  - BPFS [SOSP ’09]
  - PMFS [EuroSys ’14]
  - NOVA [FAST ’16, SOSP ’17]
  - SoupFS [USENIX ATC ’17]
- User-space NVM file systems[1]
  - Aerie [EuroSys ’14]
  - Strata [SOSP ’17]

[1] These file systems also require kernel part supports.

User-space NVM file systems have benefits

- User-space NVM file systems[1]
  - Aerie [EuroSys ’14]
  - Strata [SOSP ’17]

✓Easier to develop, port, and maintain[2]

✓Flexible[3]

✓High-performance due to kernel bypass[3,4]

[1] These file systems also require kernel part supports.
[2] To FUSE or Not to FUSE: Performance of User-Space File Systems, FAST ’17
[3] Aerie: Flexible File-System Interfaces to Storage-Class Memory, EuroSys ’14
[4] Strata: A Cross Media File System, SOSP ’17

Metadata is indirectly updated in user space

- Updates to metadata are performed by trusted components
  - Trusted FS Service in Aerie
  - Kernel FS in Strata

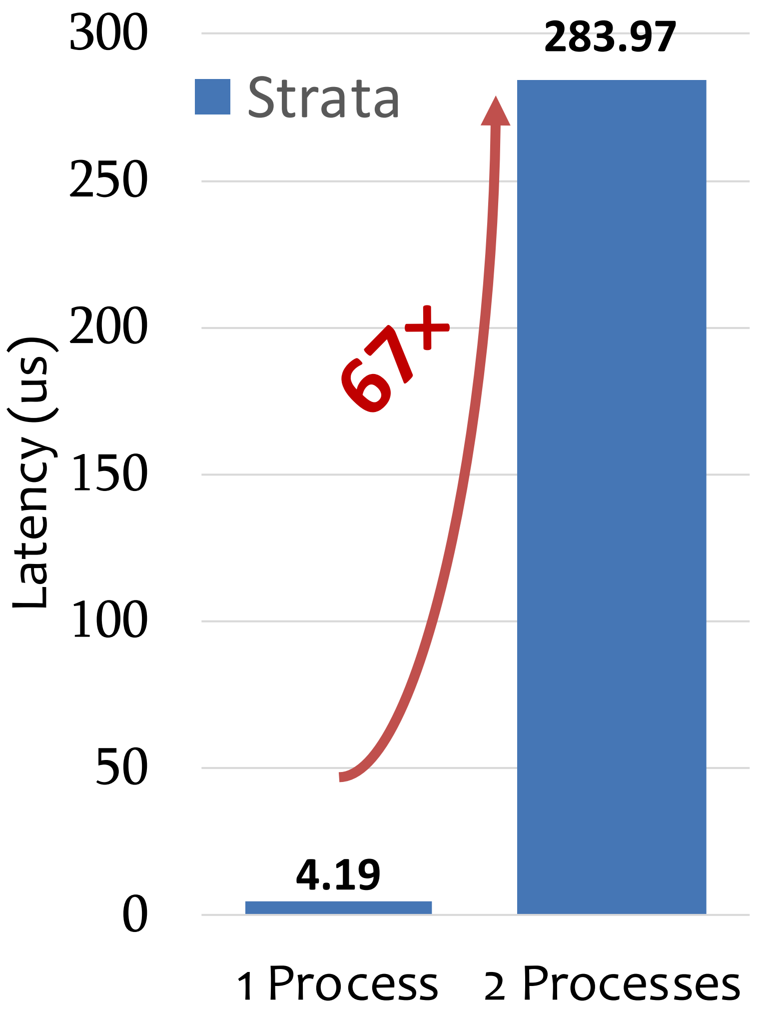
	Update data	Update metadata
Aerie	direct write	via IPCs
Strata	append a log in user space, digest in kernel	

Indirect updates!

Indirect updates are important but limit performance

- Indirect updates protect metadata
  - File system integrity
  - Access control
- Indirect updates limit performance!

Create empty files in a shared directory



Goal: fully exploit NVM performance

- **Problem:** Indirect updates protect metadata but limit performance
- **Our approach: Directly manage both data and metadata** in user-space libraries while **retaining protection**
  - Coffer: separate NVM protection from management
  - The kernel part protects coffers via paging
  - User-space libraries manage in-coffer structures (file data and metadata)
- **Results: Outperform** existing systems by **up to 82%** and **exploit full NVM bandwidth** in some scenarios

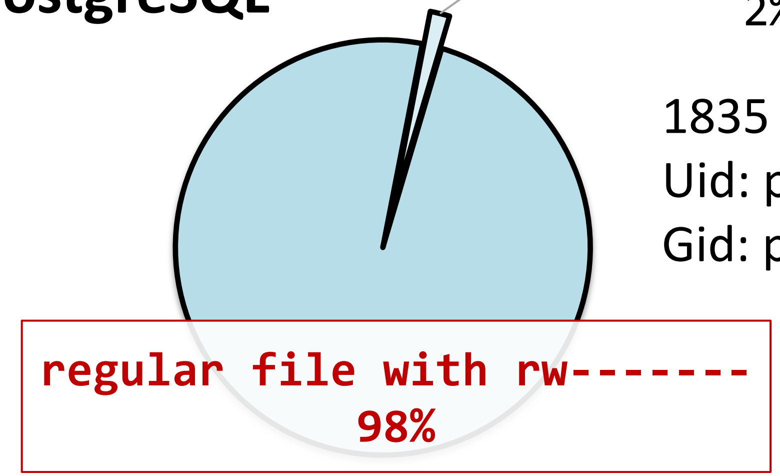
Outline

- Coffer
- Protection and isolation
- Evaluation

Files are stored with the same permissions that rarely change

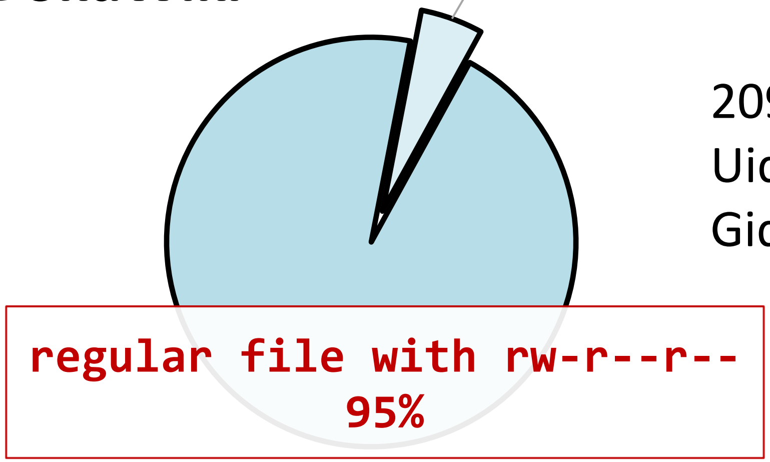
- Survey on database and webserver data files

PostgreSQL



1835 files
Uid: postgres
Gid: postgres

DokuWiki



20976 files
Uid: http
Gid: http

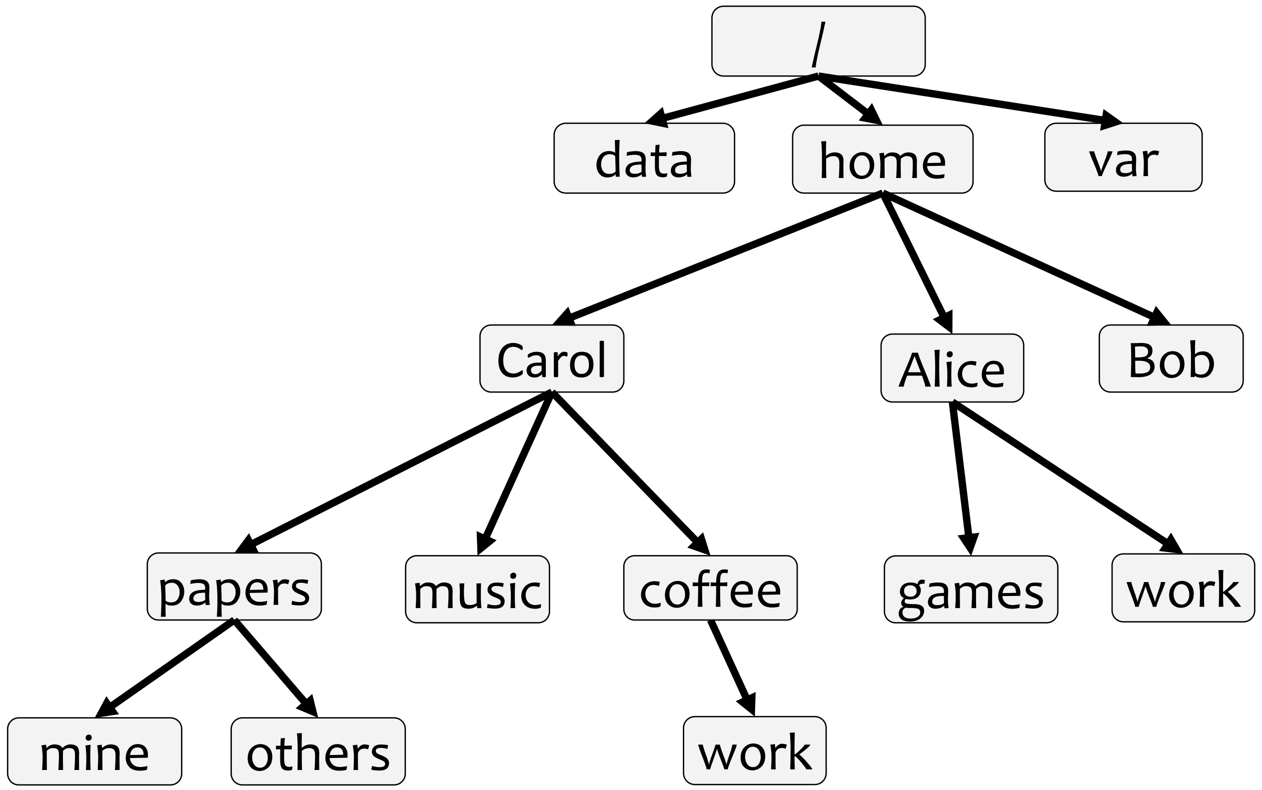
  – Most files have the same ownership & permission

Files are stored with the same permissions that rarely change

- Survey on database and webserver data files
  - Most files have the same ownership & permission
  - Ownership & permissions are seldom changed

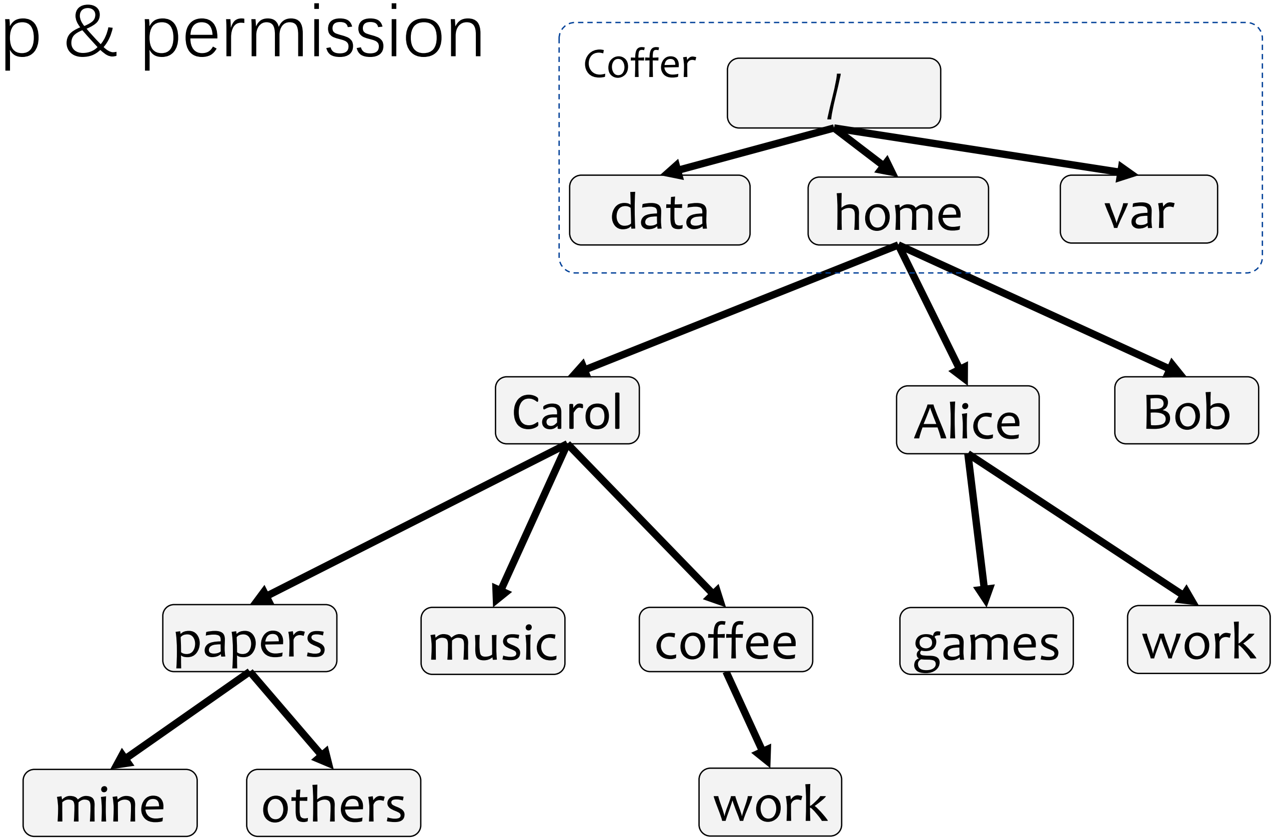
1. Group files with the same ownership & permission
2. Map their data and metadata to user space
3. Let user-space libraries manage these data and metadata directly

A new abstraction: Coffer



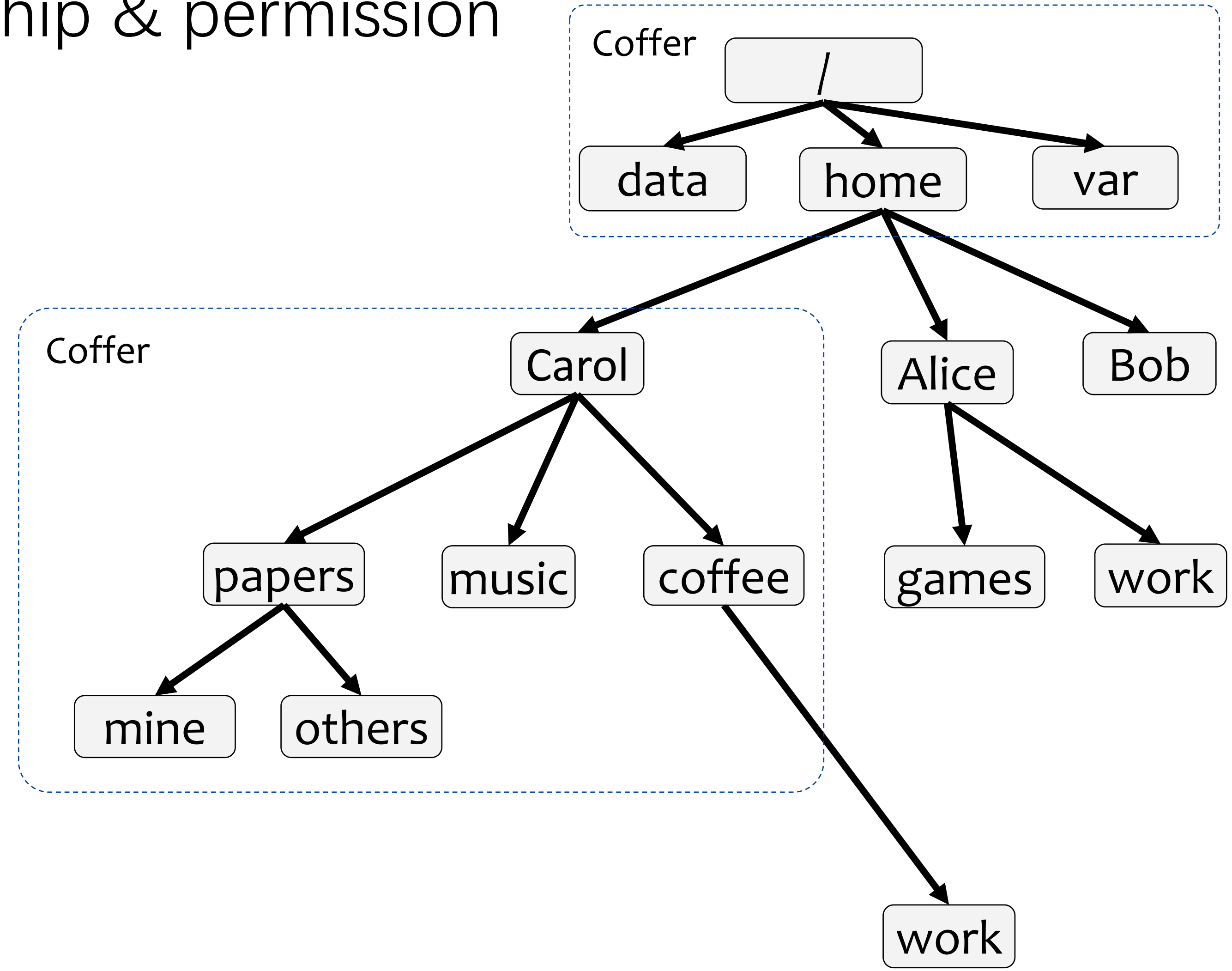
A new abstraction: Coffer

Group files with the same ownership & permission



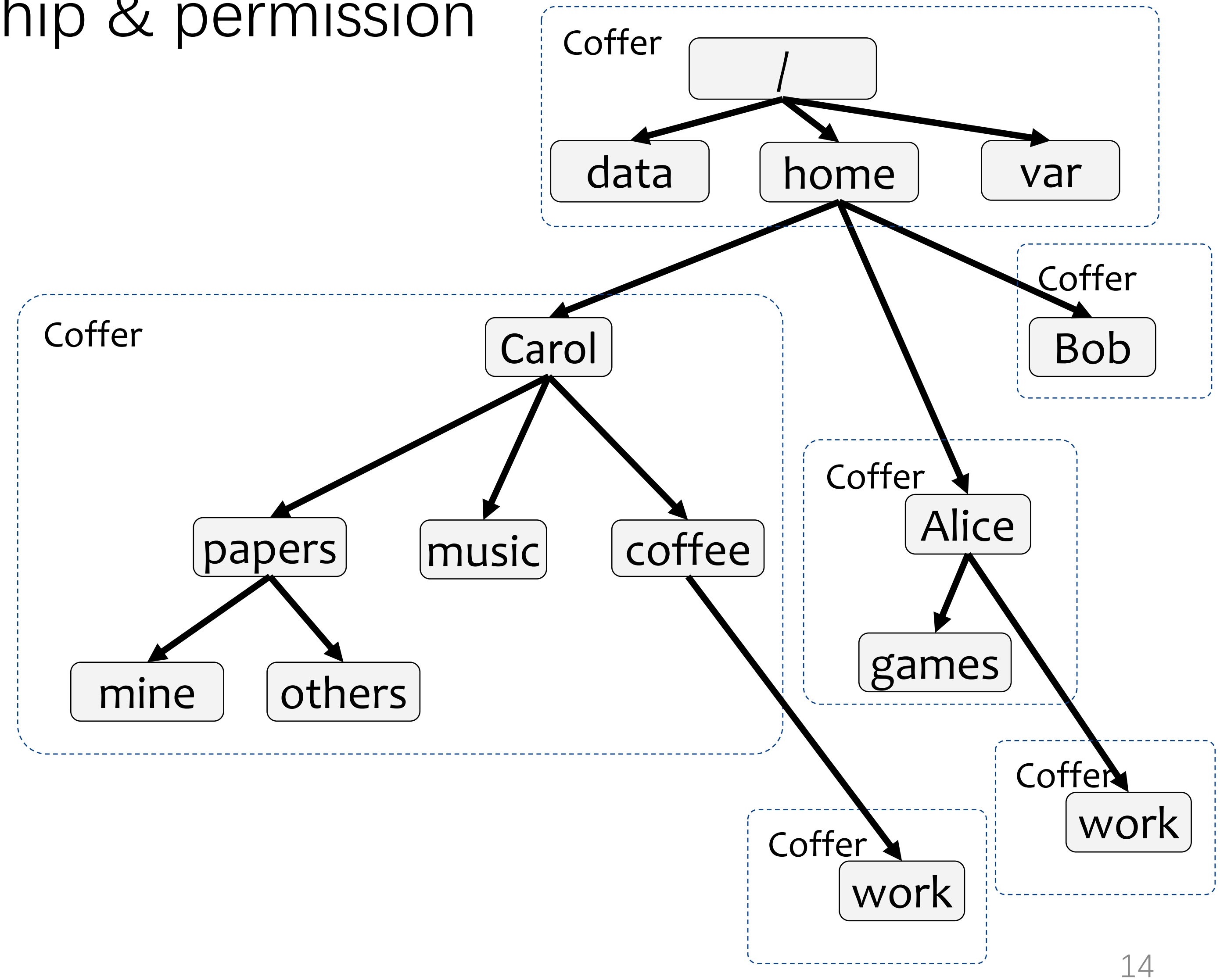
A new abstraction: Coffer

Group files with the same ownership & permission



A new abstraction: Coffer

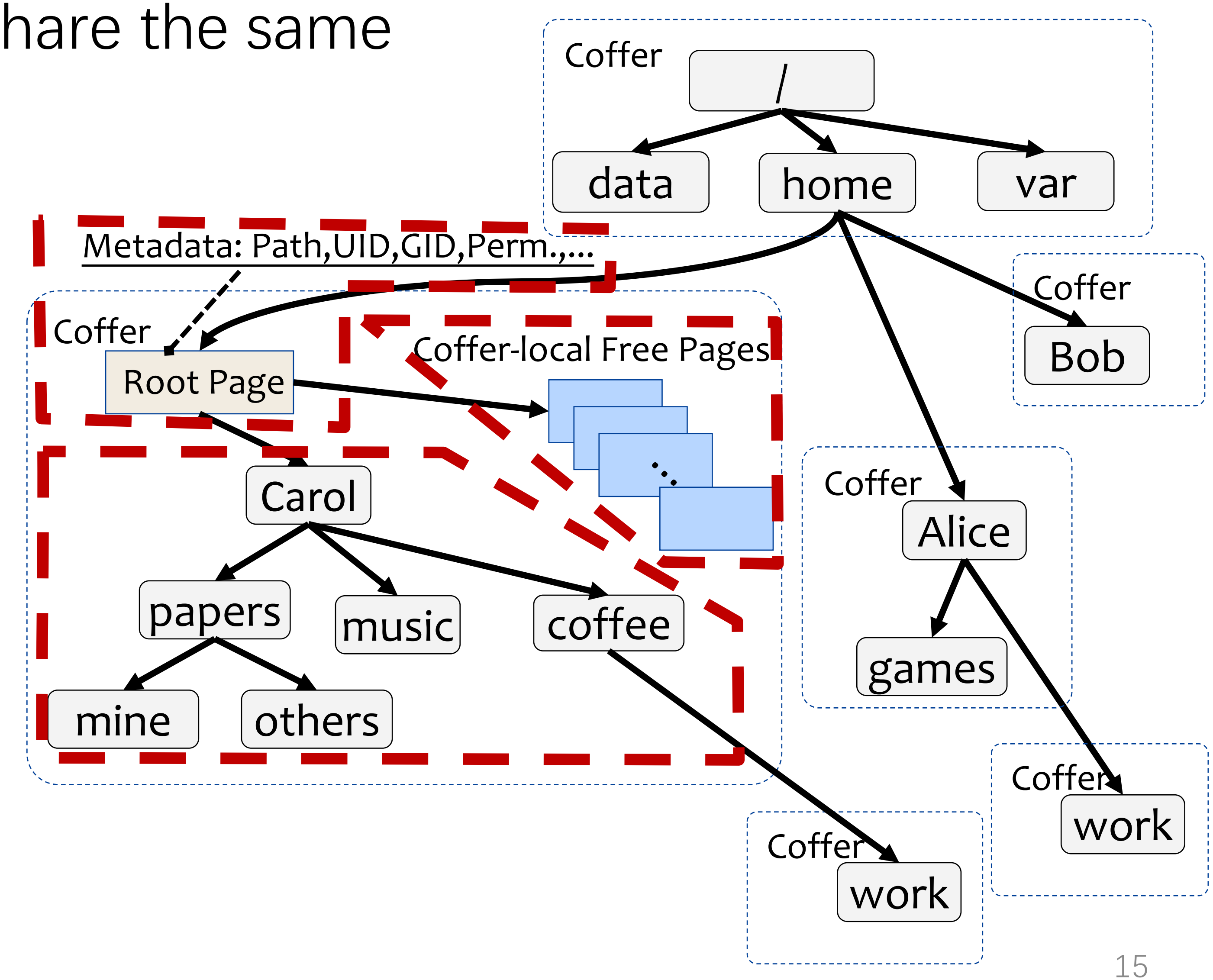
Group files with the same ownership & permission



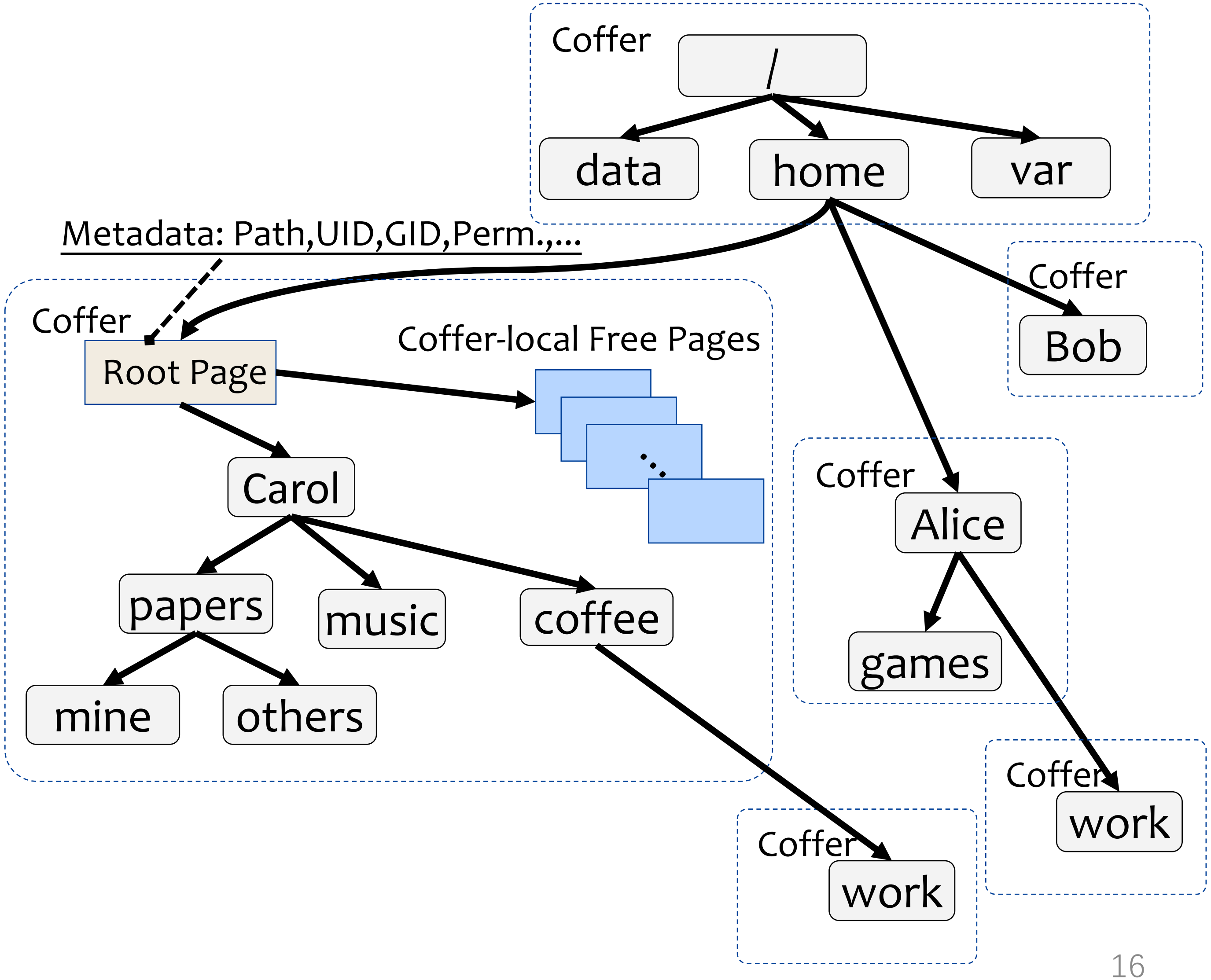
Coffer internals

A collection of NVM pages that share the same ownership & permission

- Files are organized by user-space FS libraries
- Local space management
- A root page with metadata
  - Path
  - Owner and permission



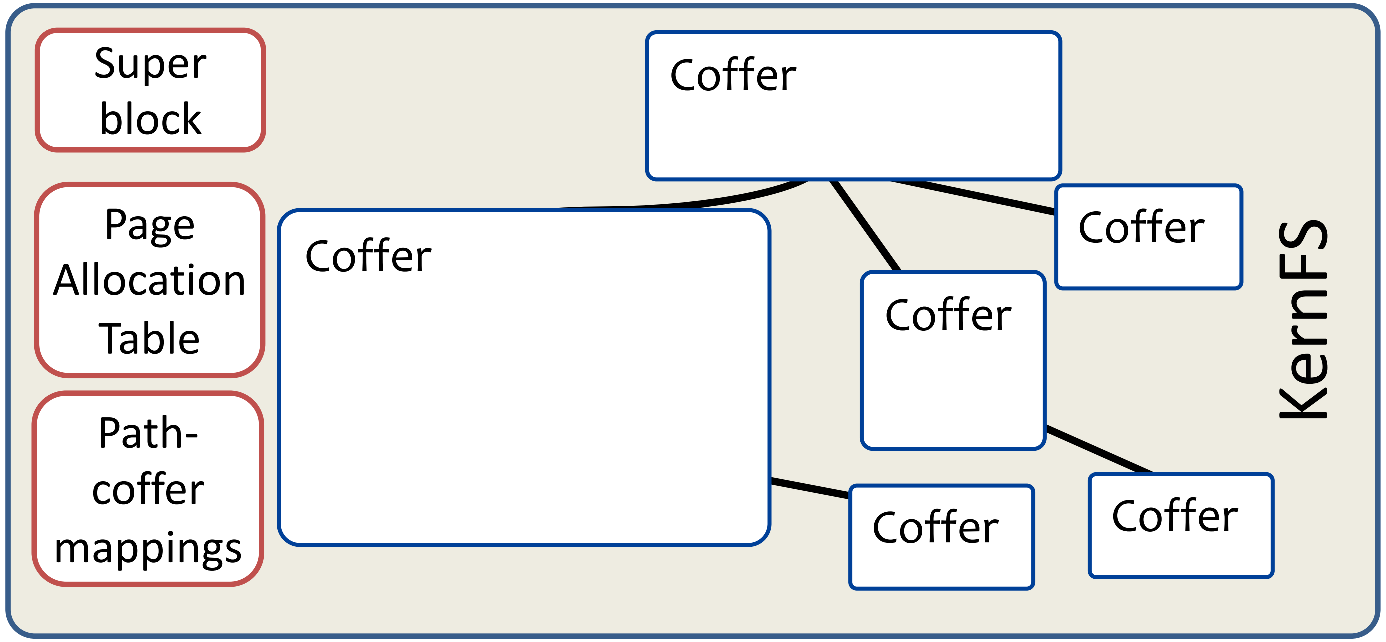
Coffer separates NVM protection from management



Coffer separates NVM protection from management

KernFS

- Protect coffer metadata
- Mange global free space



Coffer separates NVM protection from management

KernFS

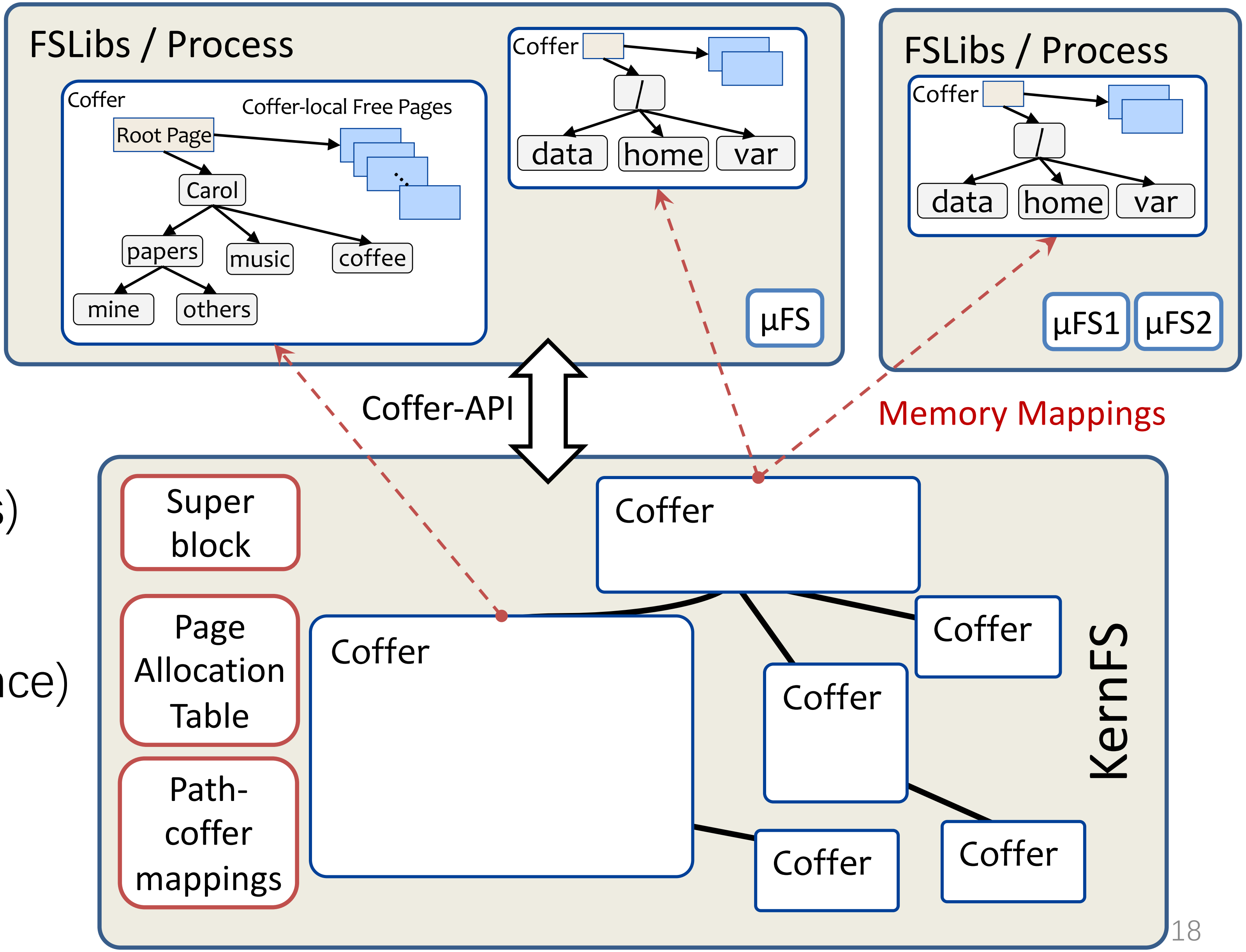
- Protect coffer metadata
- Mange global free space

FSLibs

- User-space FS libraries (µFSs)
- Manage in-coffer structures (data, metadata and free space)

Coffer-APIs

- create, map, split, …



Outline

- Coffer
- Protection and isolation
- Evaluation

Protection and isolation

Hardware paging

- For each coffer map request
  - KernFS checks the ownership and permission of the coffer
  - Map the coffer to the process page table read-only/read-write accordingly
- Applications can access a coffer only if they have the permission

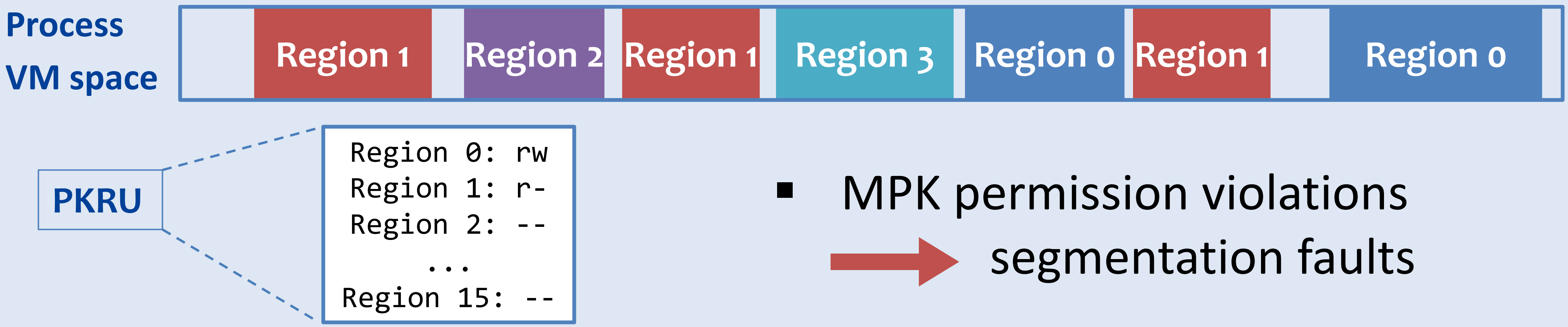
Protection and isolation

Hardware paging

- Applications can access a coffer only if they have the permission

Memory protection keys

A hardware feature that supplements paging



- MPK permission violations → segmentation faults

Protection and isolation

Hardware paging

- Applications can access a coffer only if they have the permission

Memory protection keys

- KernFS separates **different coffers** to **different memory protection regions** for each process
- Application threads can control its access to each coffer efficiently

Protection and isolation

Hardware paging

- Applications can access a coffer only if they have the permission

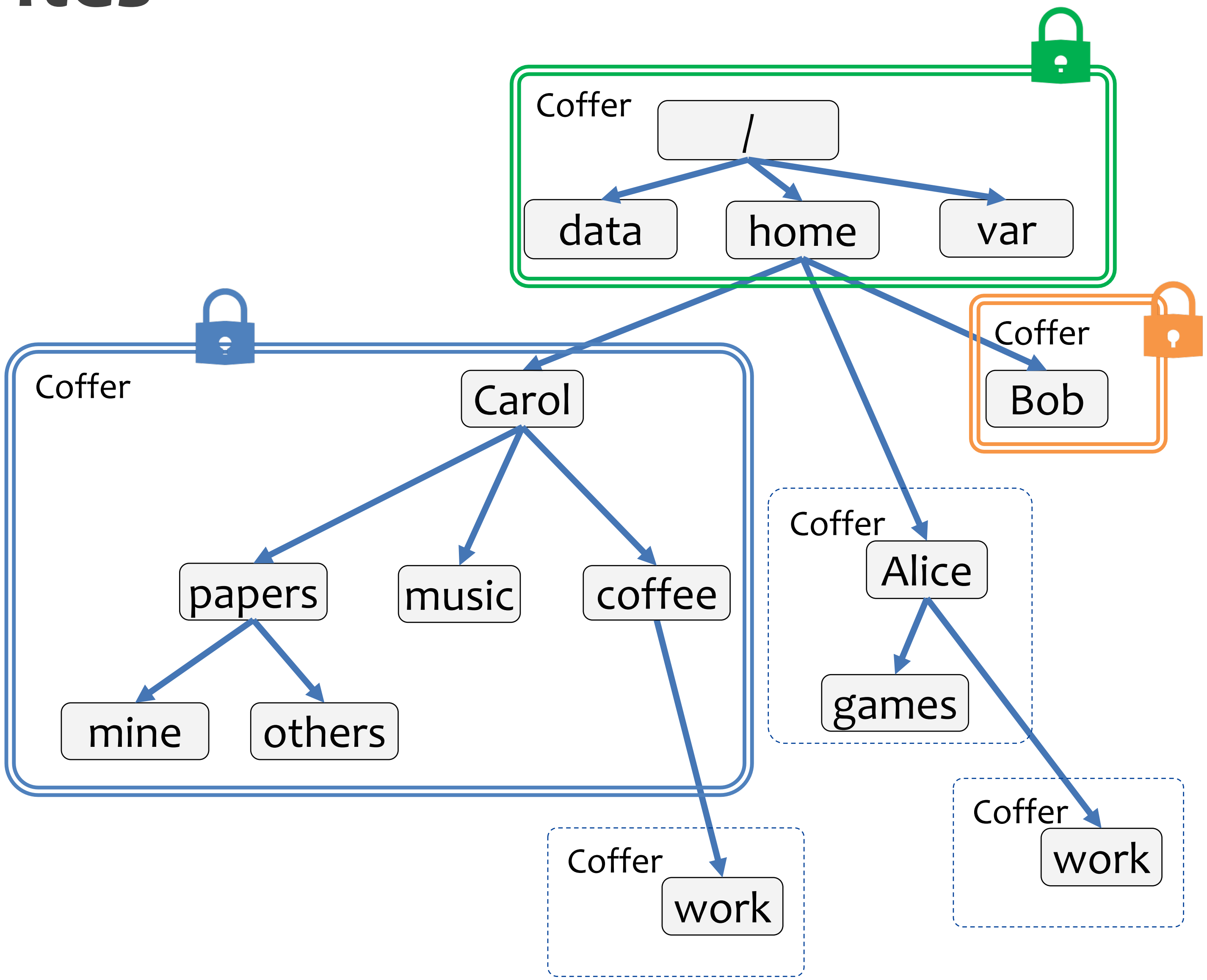
Memory protection keys

- Application threads can control its access to each coffer efficiently

Challenges

1. Stray writes 2. Malicious manipulations 3. Fate sharing

Challenge 1: stray writes



Challenge 1: stray writes

Problem: Stray writes corrupt metadata in mapped coffers

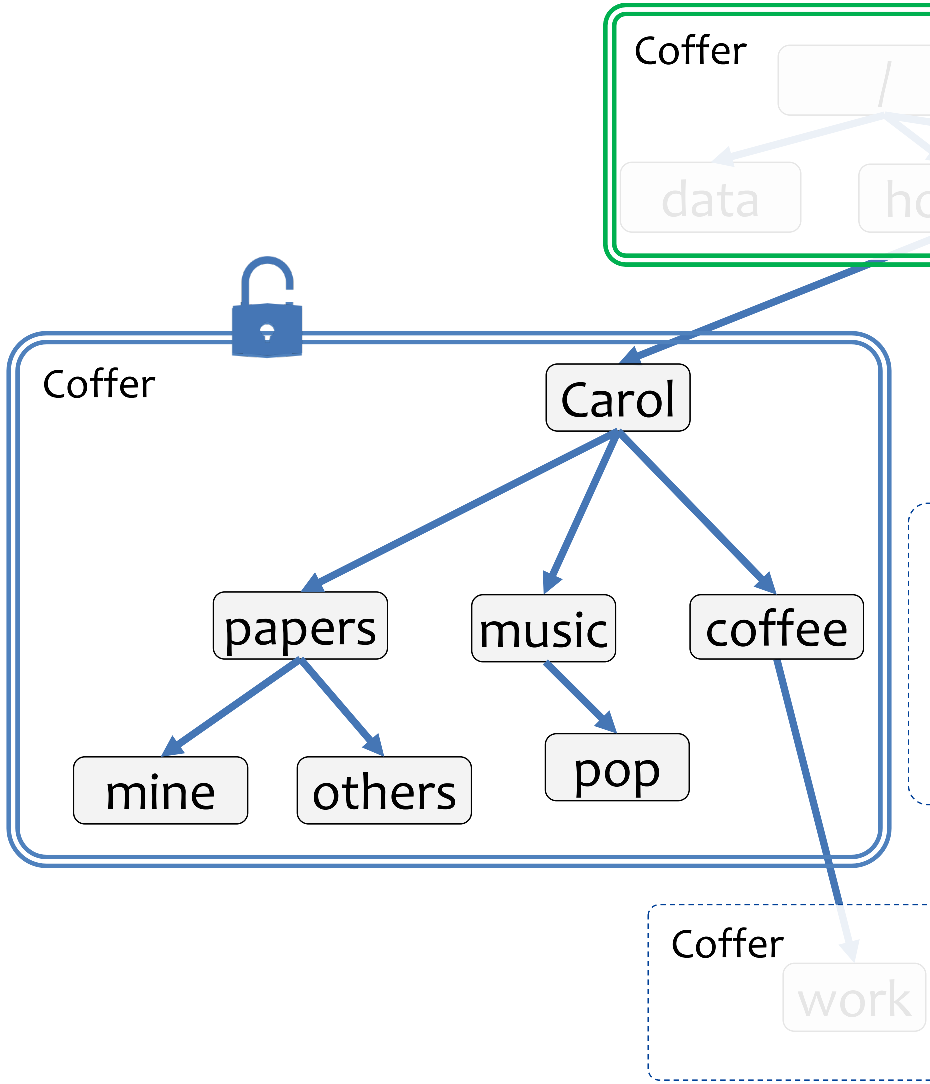
Approach: write windows[1]

- MPK regions are initialized as **non-accessible**
- When a µFS modifies a coffer

1. Enable coffer access
2. Modify coffer
3. Disable coffer access

PKRU

Region	R	W
Blue	R	W
...	-	-



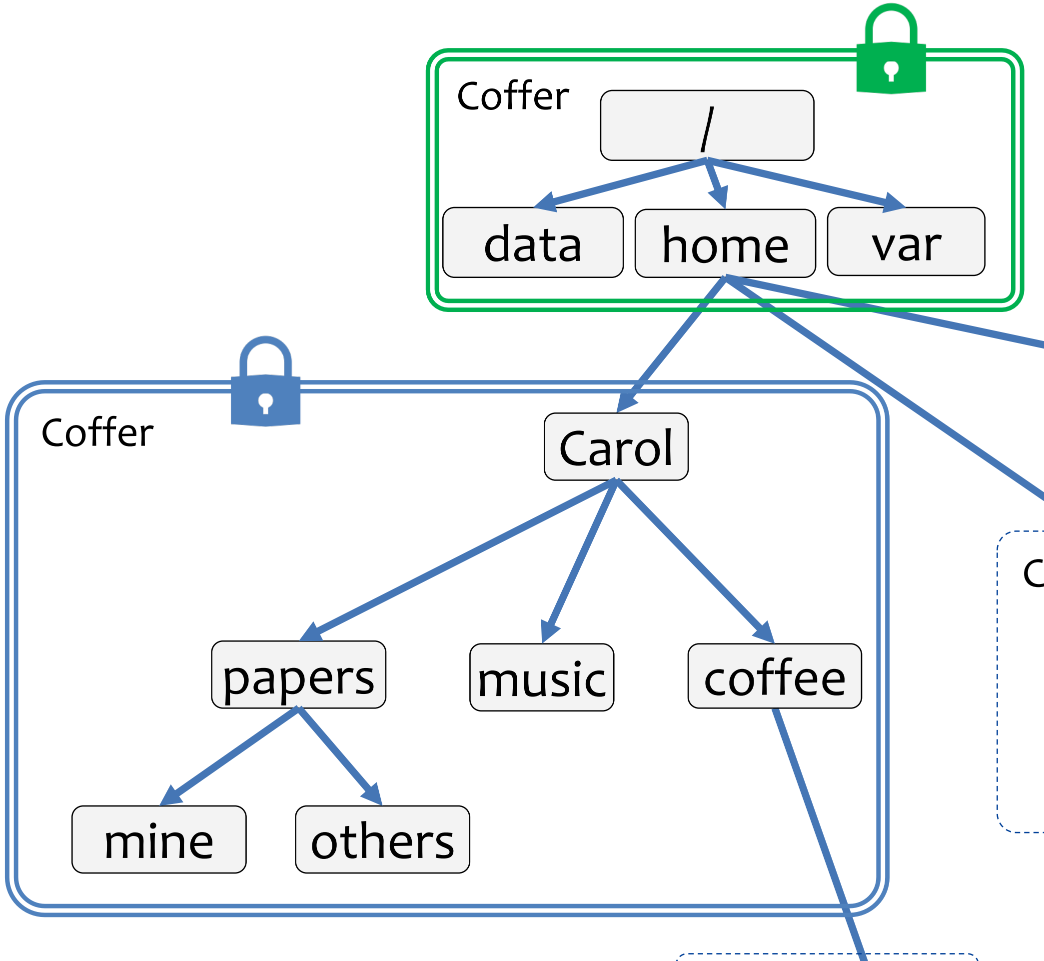
Result: Stray writes in application code cause segmentation faults due to MPK

[1] System Software for Persistent Memory, EuroSys '14

Challenge 2: malicious manipulations

Problem: An attacker manipulates a shared coffer to attack others!

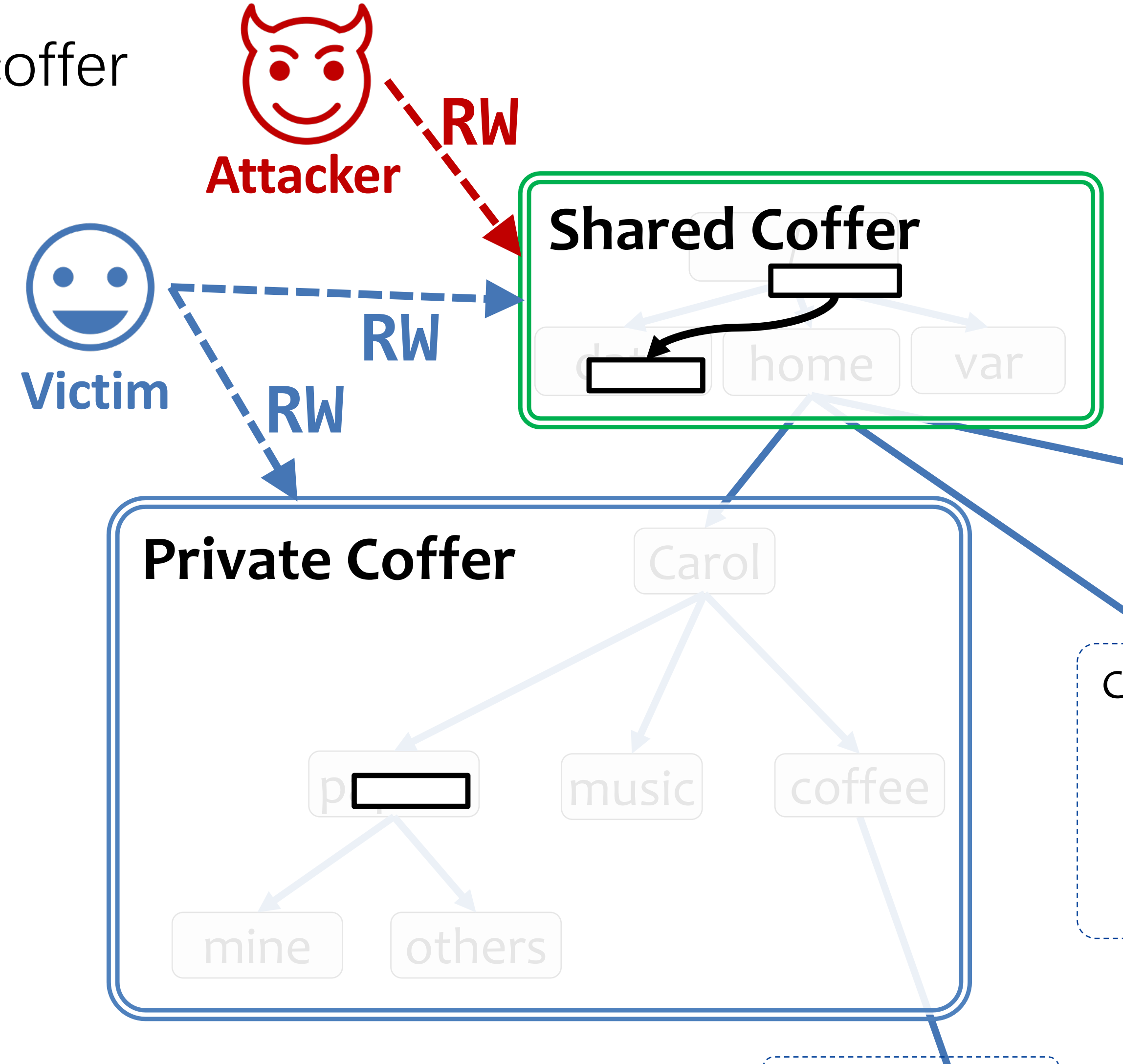
Case: manipulate a pointer to point to another coffer



Challenge 2: malicious manipulations

Problem: An attacker manipulates a shared coffer to attack others!

Case: manipulate a pointer to point to another coffer

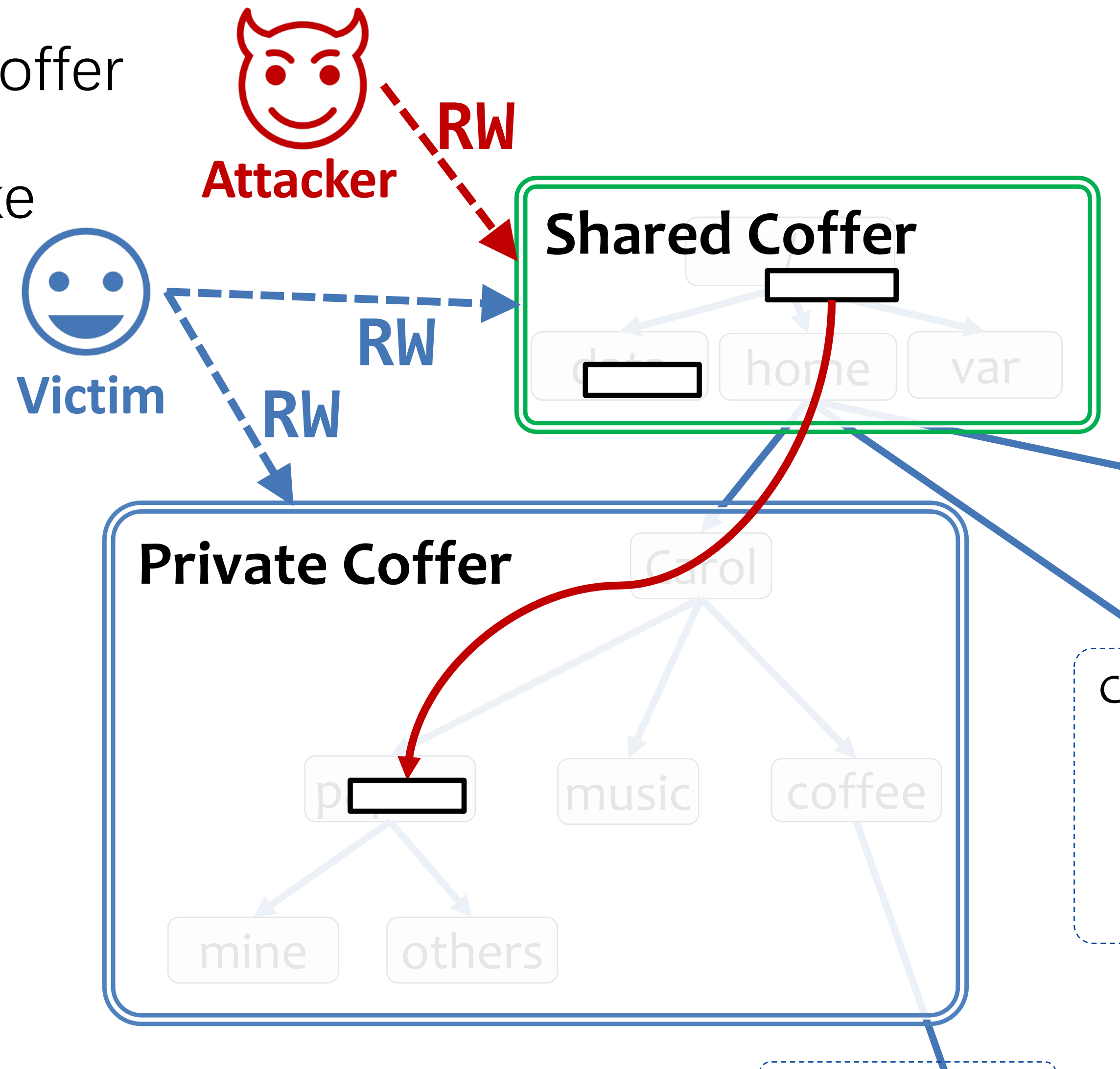


Challenge 2: malicious manipulations

Problem: An attacker manipulates a shared coffer to attack others!

Case: manipulate a pointer to point to another coffer

The victim accesses **the private coffer** by mistake



Challenge 2: malicious manipulations

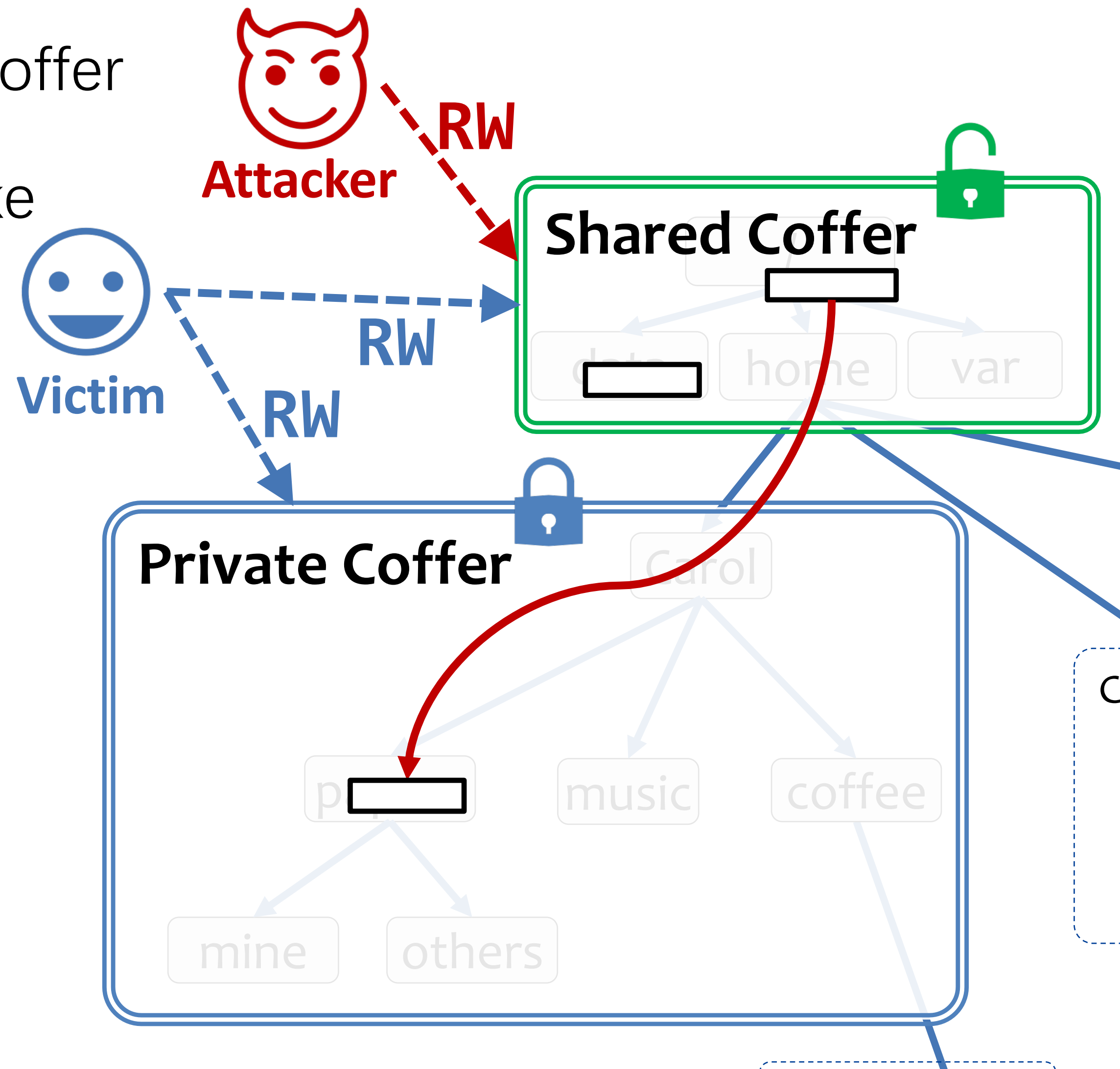
Problem: An attacker manipulates a shared coffer to attack others!

Case: manipulate a pointer to point to another coffer

The victim accesses **the private coffer** by mistake

Approach: **At most one** coffer is accessible
at any time for each thread

Result: Following manipulated pointers
triggers **segmentation faults!**

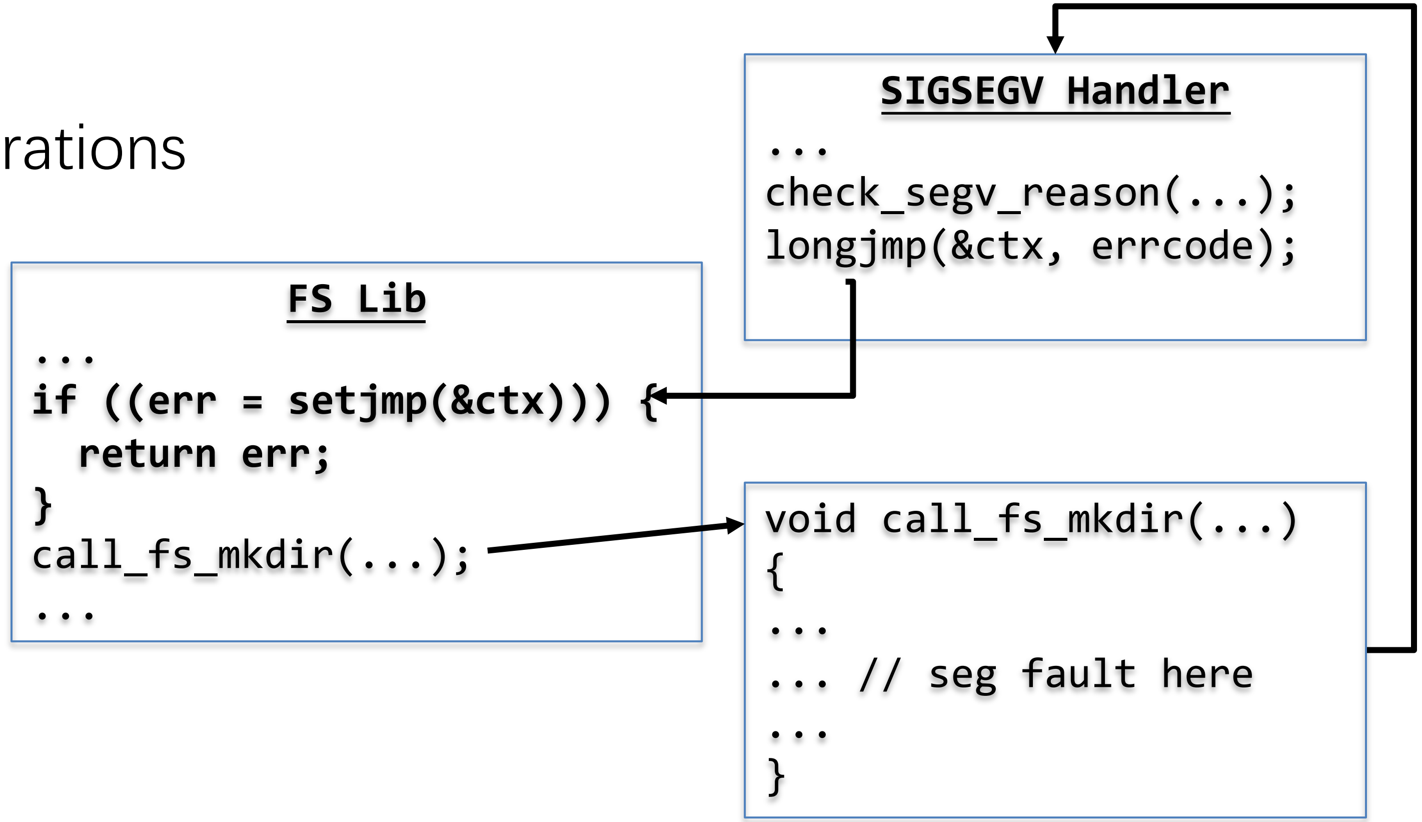


Challenge 3: fate sharing

Problem: An error in FS libraries can terminate the whole process!

Approach:

1. Setjmp before user-space FS operations
2. Hook the SIGSEGV handler
3. Jump back and return error code



Result: Segmentation faults are reported to the application as an FS error code!

ZoFS: An example user-space NVM FS library (μFS)

Directory

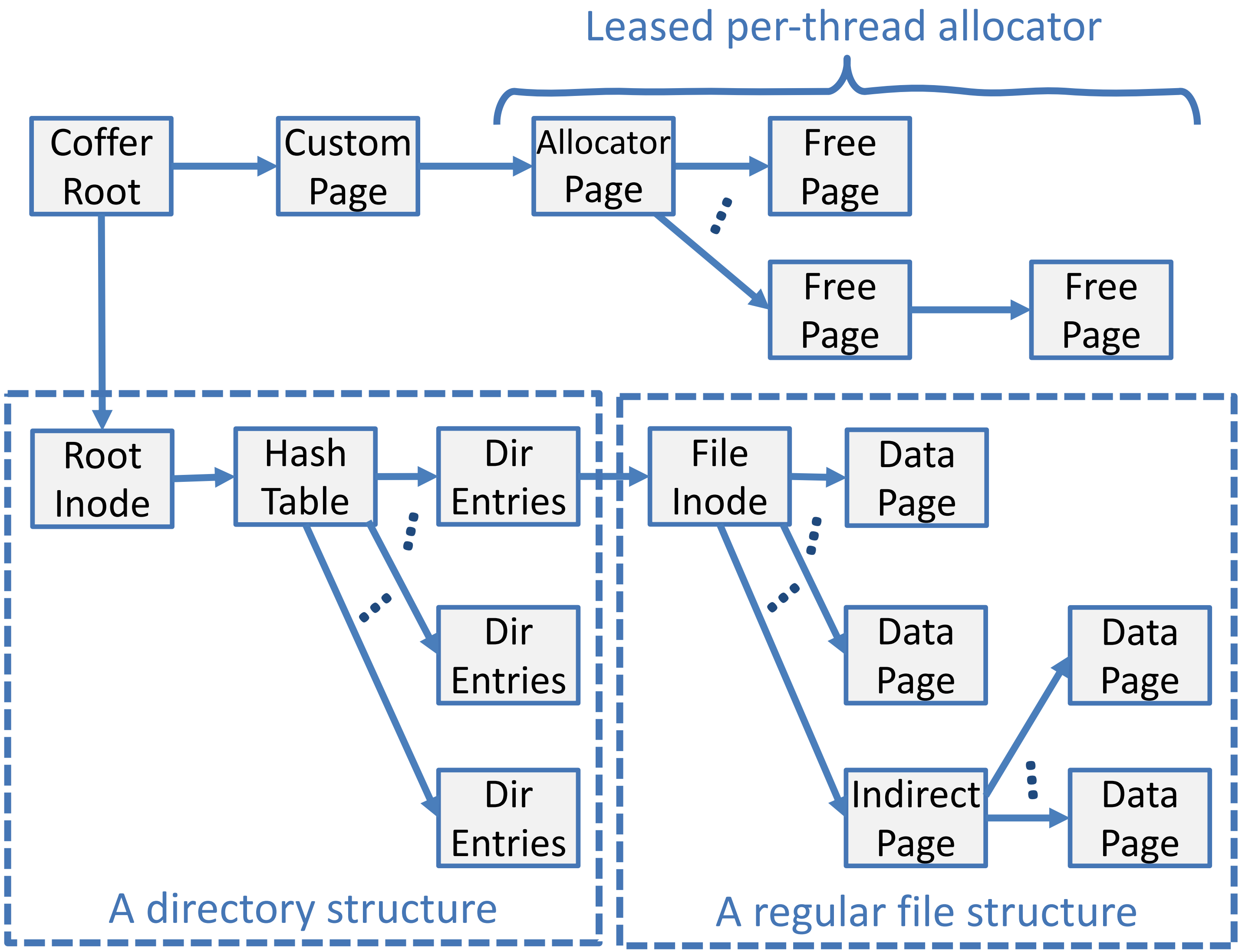
- Adaptive two-level hash tables

File structures

- Simple direct/indirect data blocks

Local space management

- Leased per-thread allocators



Outline

- Coffer
- Protection and isolation
- Evaluation

Evaluation Questions

- Can ZoFS **scale** and **fully exploit NVM performance**?
- How much performance benefit comes from the **direct updates** in user space?
- How does ZoFS perform in synthetic workloads and real applications?

Evaluation Setup

Two 10-core Intel® Xeon® Gold 5215M CPUs

384 GB DDR4 DRAM

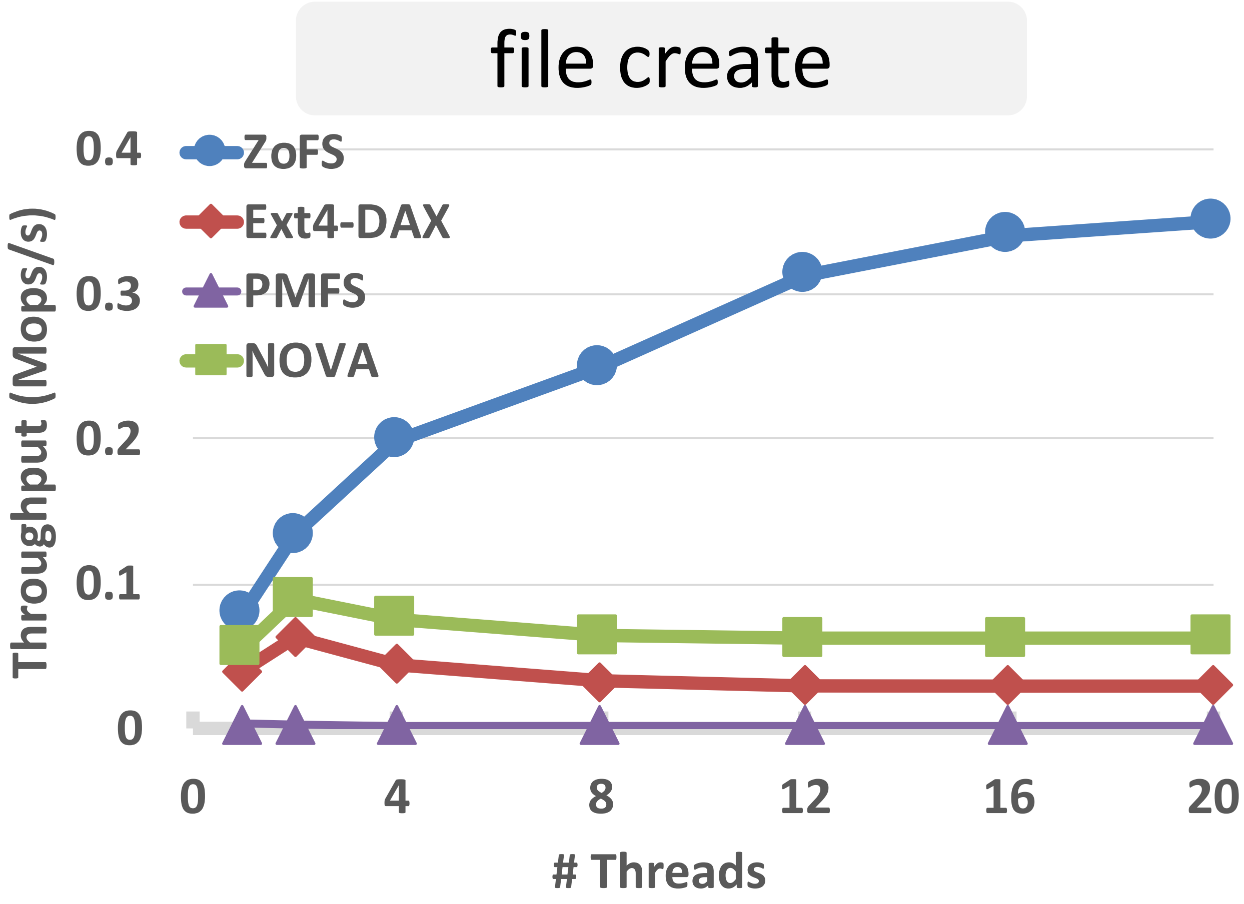
1.5 TB Intel® Optane™ DC Persistent Memory

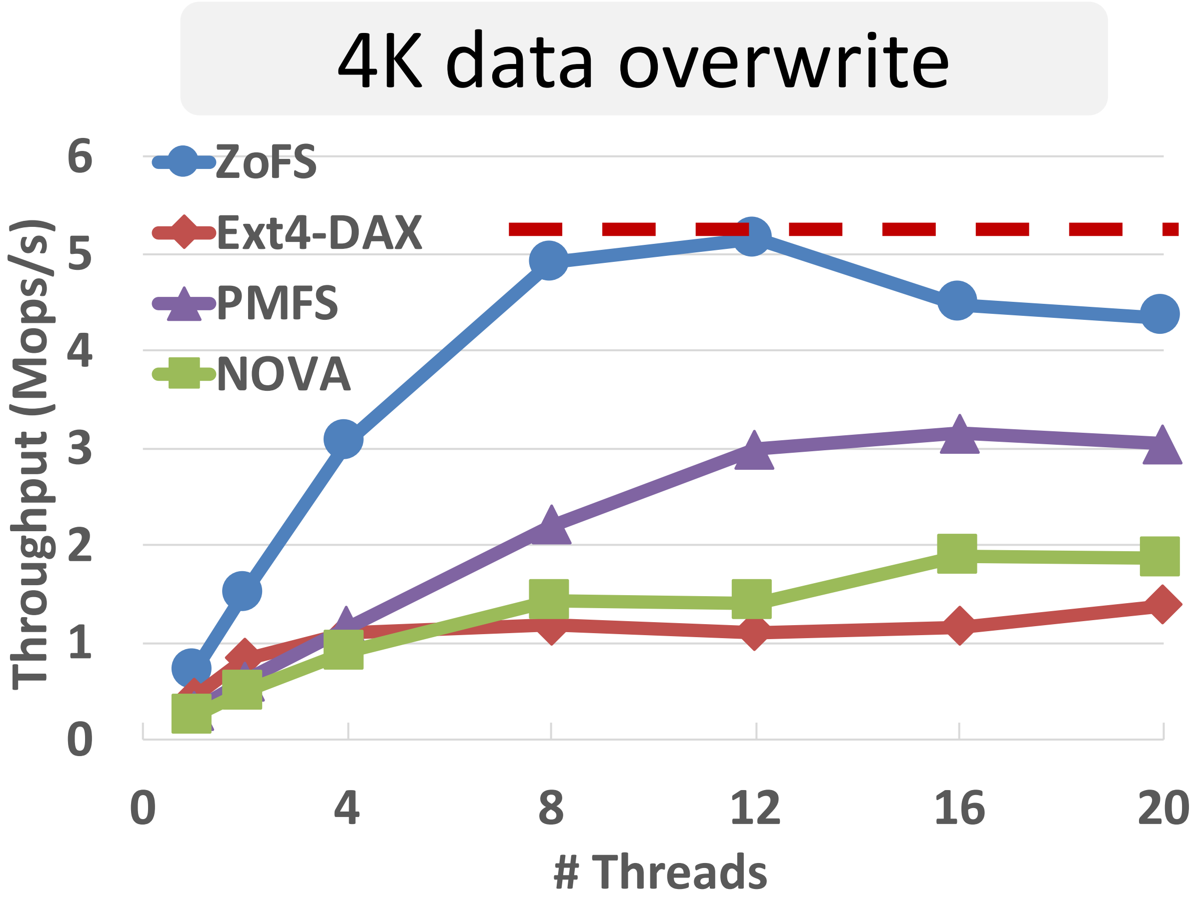
All experiments on NUMA 0 with hyper-threading disabled

File Systems: Ext4-DAX, PMFS, NOVA, Strata

Benchmarks: FxMark, Filebench, LevelDB and TPC-C on SQLite

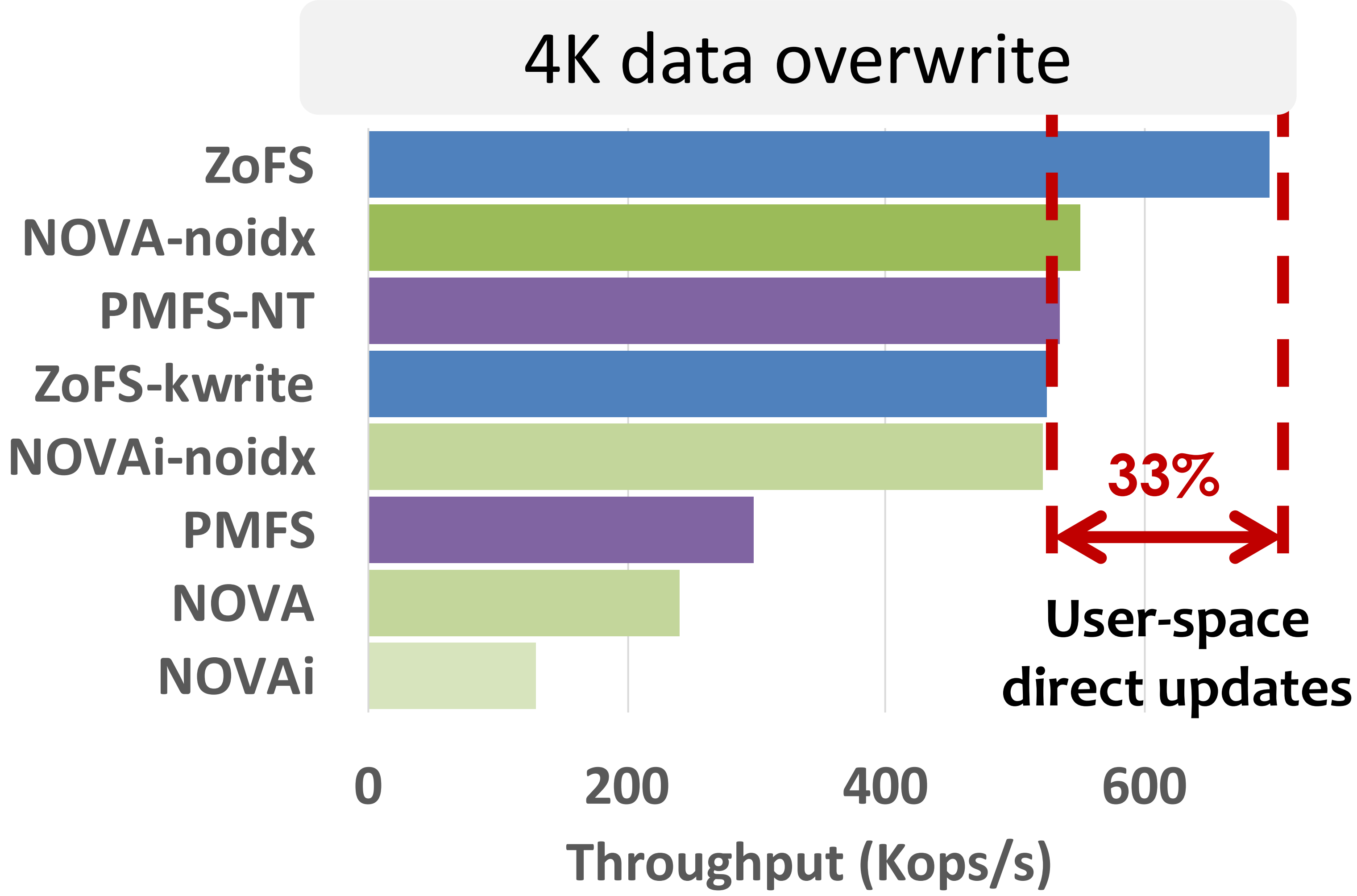
FxMark





ZoFS **scales well** and reaches the **maximal NVM bandwidth** of our platform!

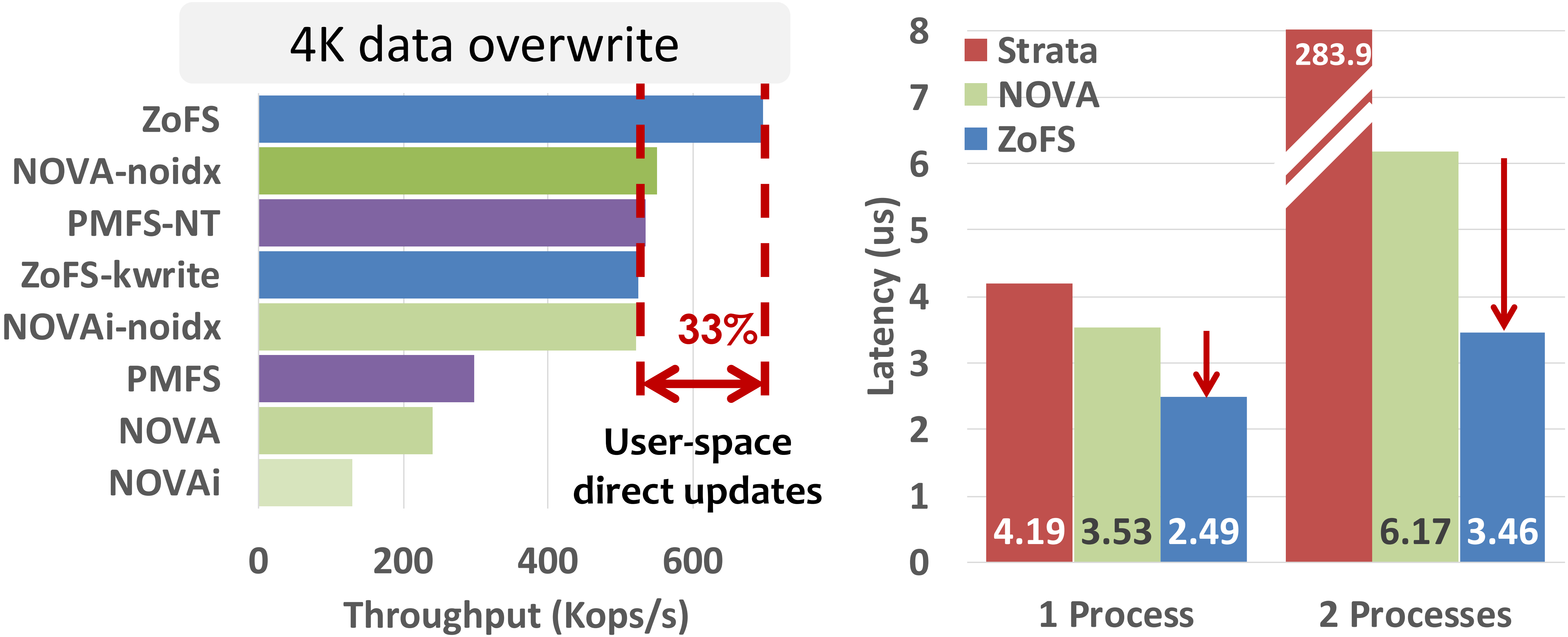
Breakdown: direct updates boost the performance



ZoFS-kwrite: implement write in kernel and call via system calls

Direct updates in user space improves the performance by 33%

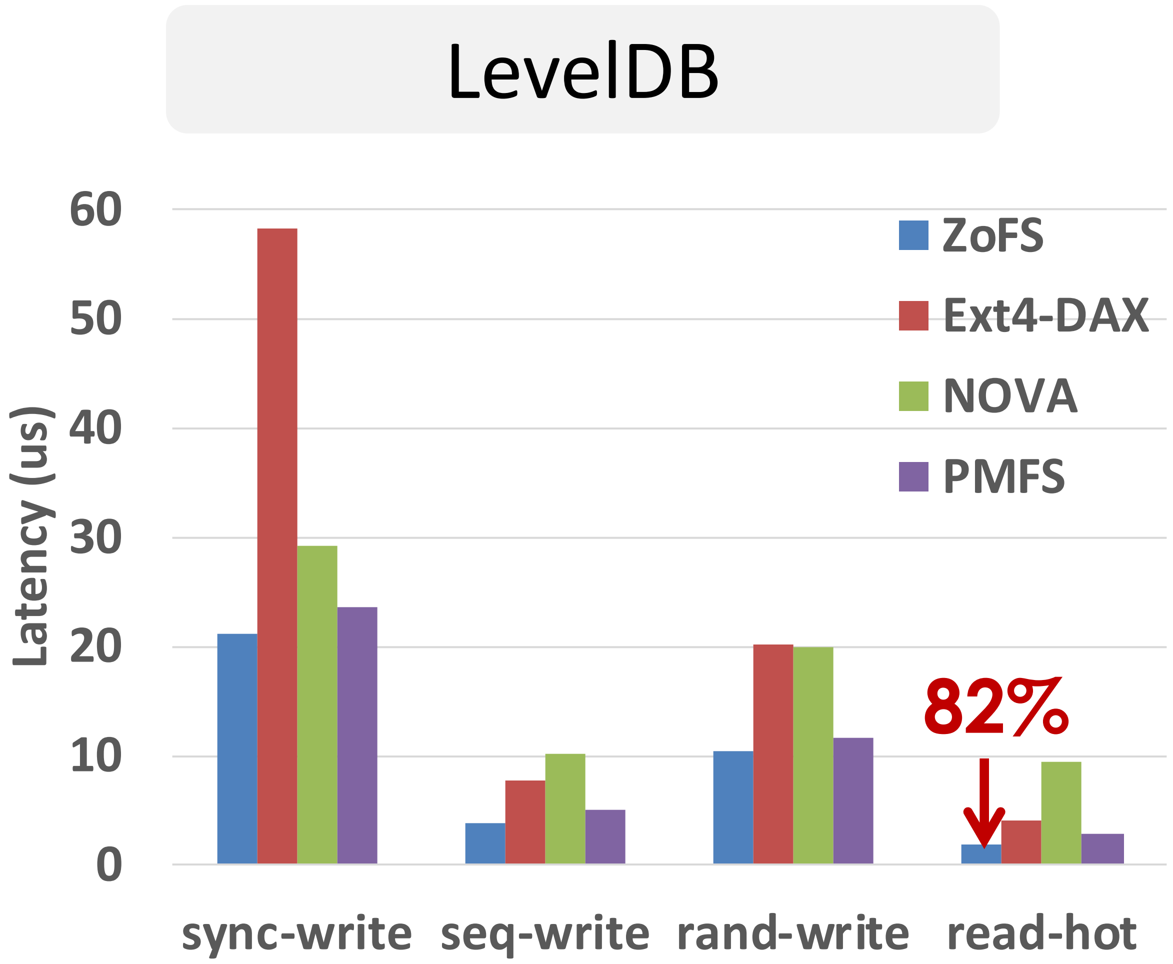
Breakdown: direct updates boost the performance

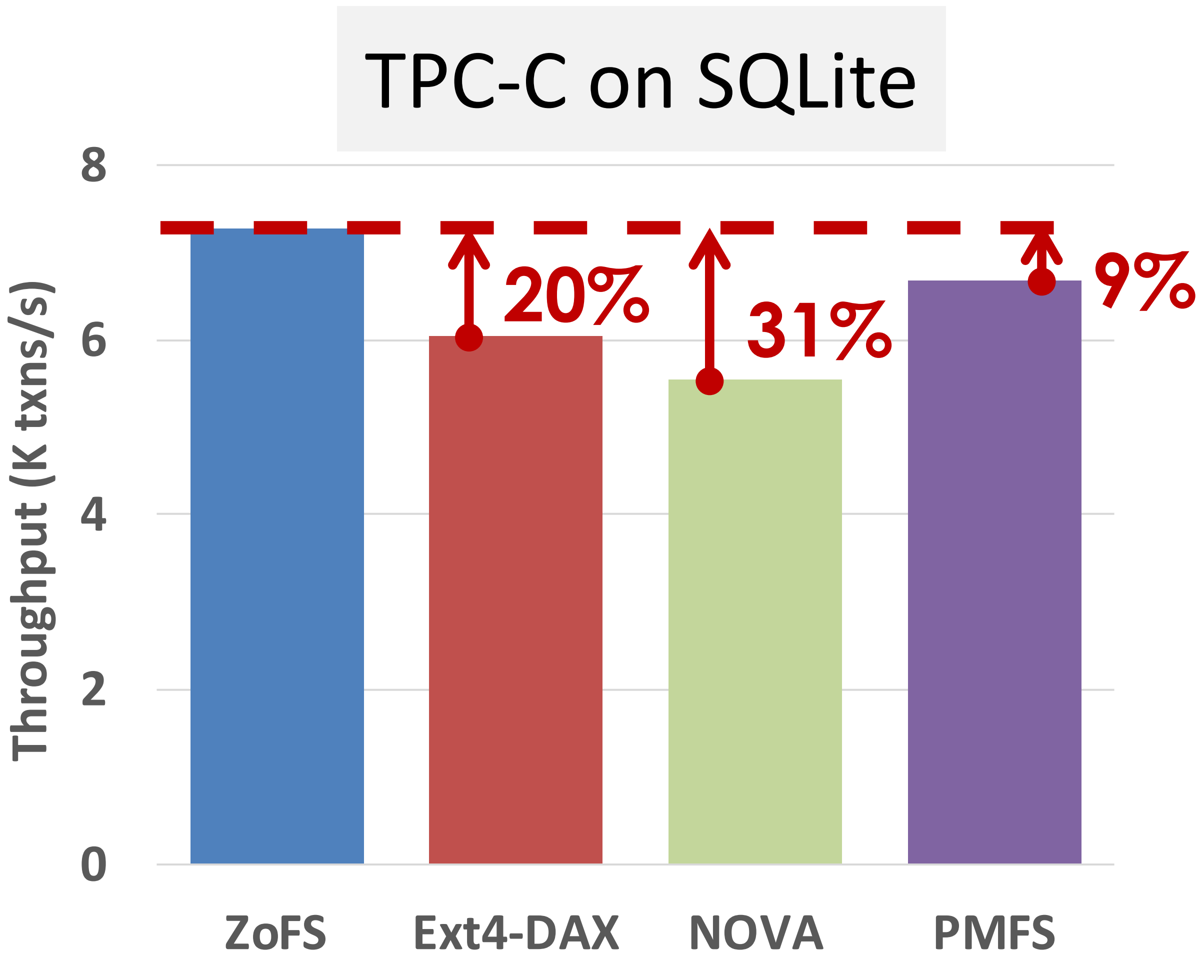


ZoFS-kwrite: implement write in kernel and call via system calls

Direct updates in user space improves the performance by 33%

LevelDB and SQLite





ZoFS reduces LevelDB latency by **up to** 82% and improves SQLite throughput by **up to** 31%

Conclusion

IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

- Non-volatile memory: fast, persistent, and byte-addressable
- Problem: no direct metadata updates in user space, underexploited NVM performance
- Coffers: separating NVM protection from management, directly managing data and metadata while embracing protection and isolation
- ZoFS built upon coffers show improved performance against existing NVM file systems

Thanks and Questions? :)