Aerie: Flexible File-System Interfaces to Storage-Class Memory

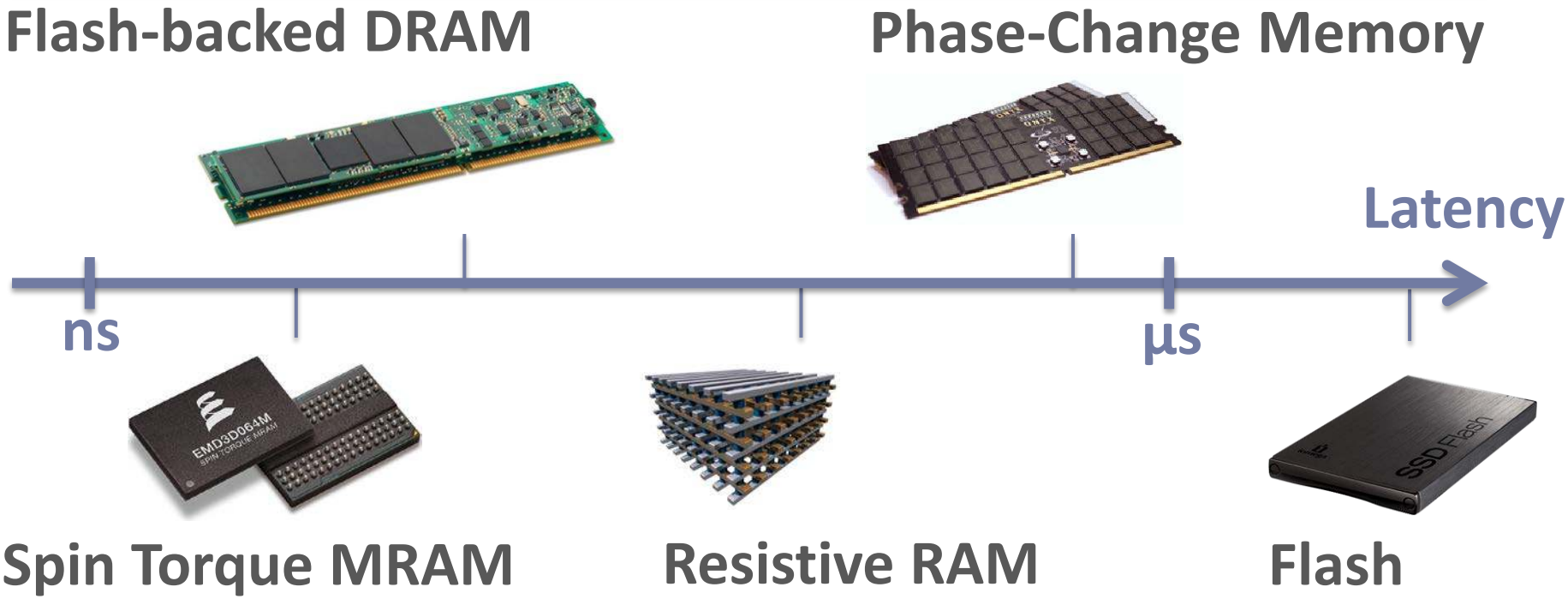
Haris Volos[†]

Sanketh Nalli, Sankaralingam Panneerselvam,
Venkatanathan Varadarajan, Prashant Saxena,
Michael M. Swift

[†] HP Labs

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Storage-Class Memory (SCM)



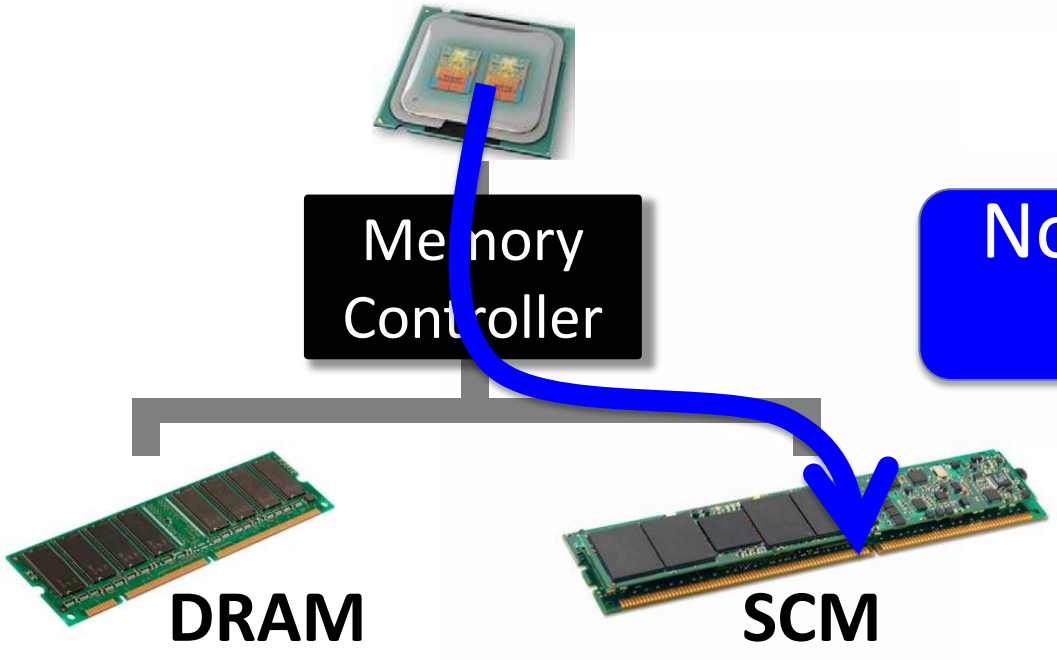
- Persistent
- Short access time

Software overhead matters

Storage-Class Memory (SCM)

- Persistent
- Short access time
- Byte addressable

Accessible via loads/stores



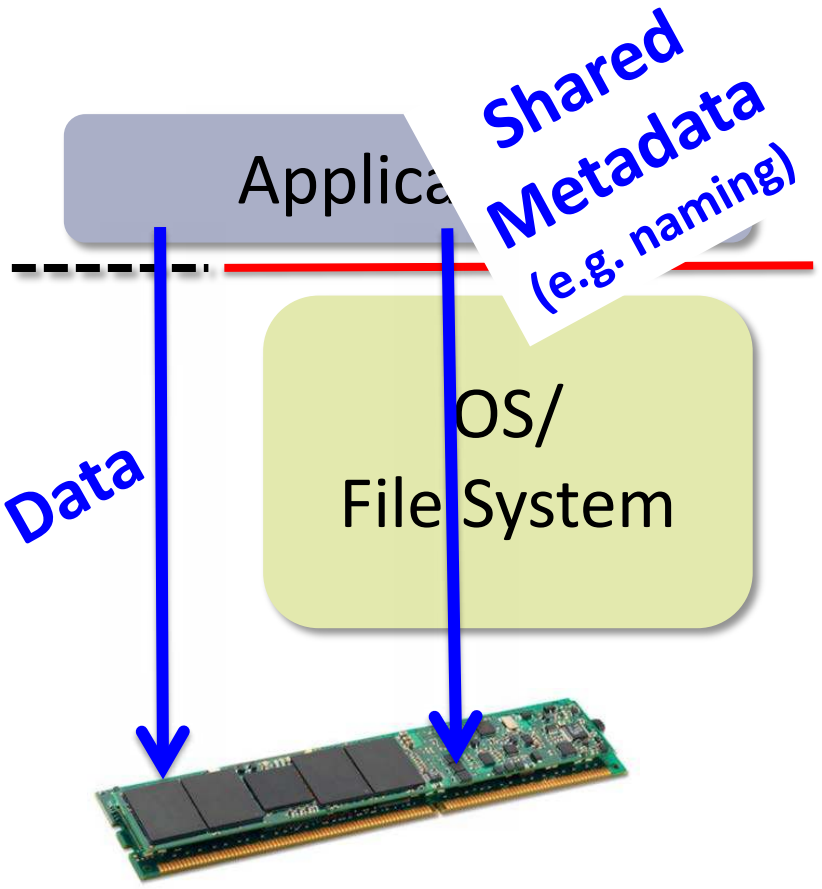
No software overhead

Accessing SCM today

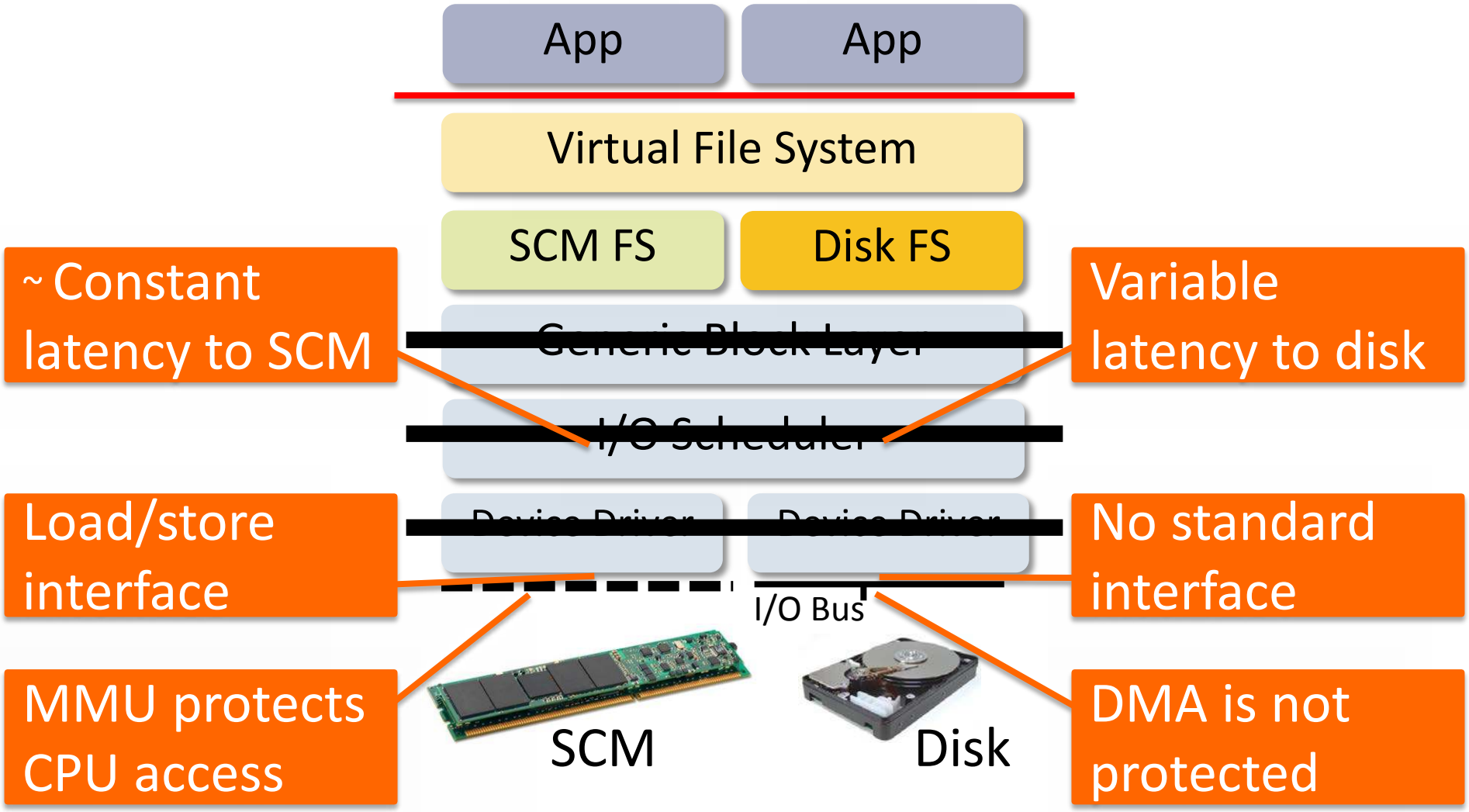
- Direct user-mode access for fast access to data
  - Moneta-D, PMFS, Quill, NV-Heaps, Mnemosyne

+

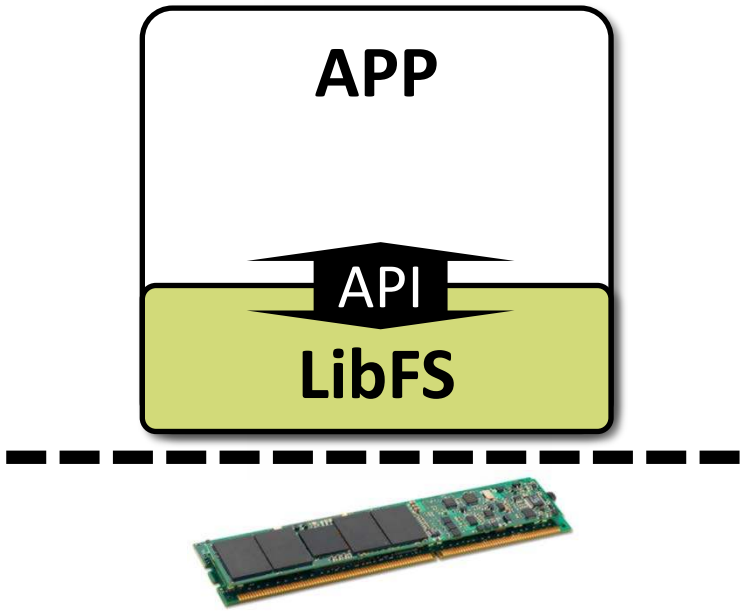
- File system for sharing
  - Shared namespace
  - Protection
  - Integrity



Does SCM need a kernel FS?



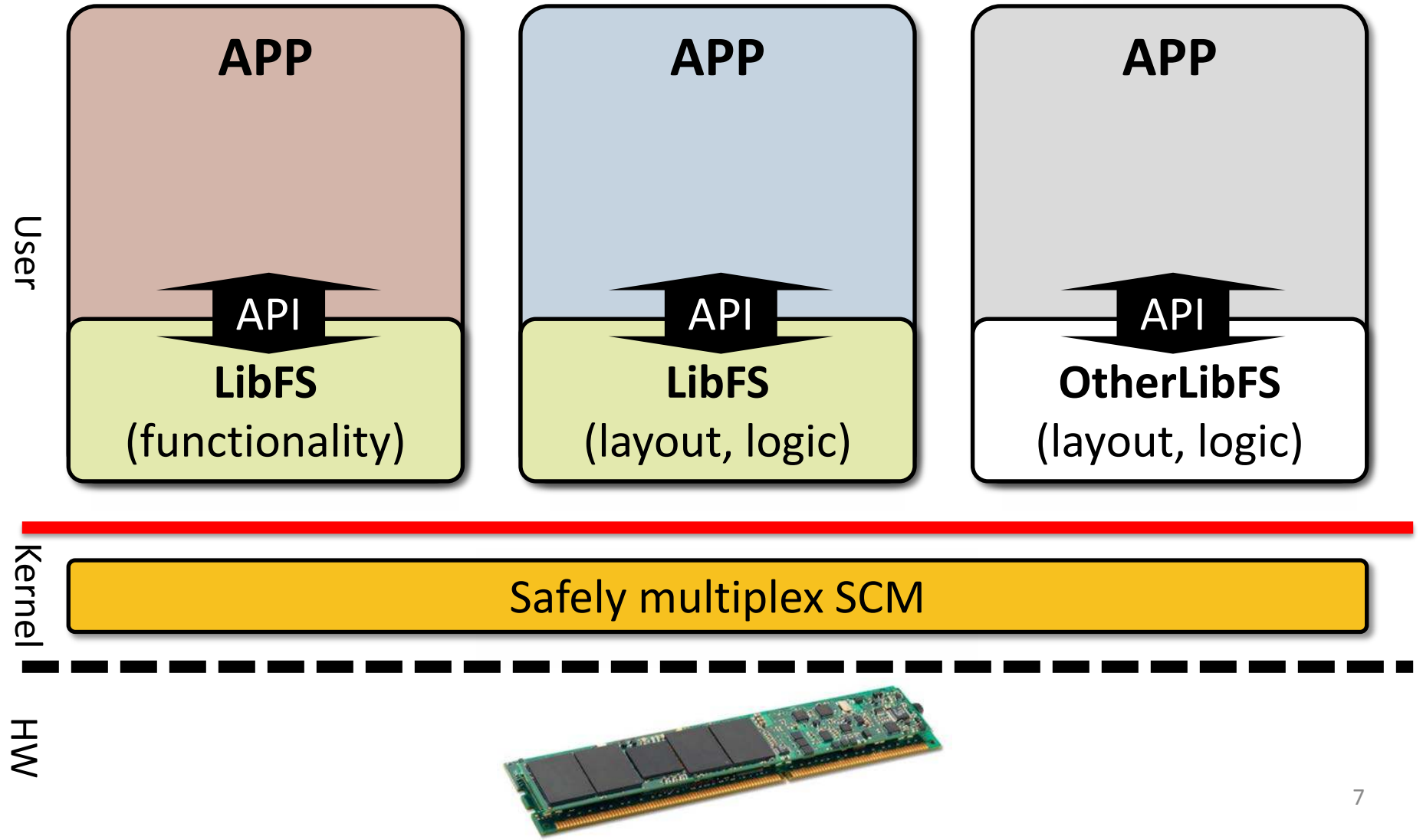
Library file systems (libFS)



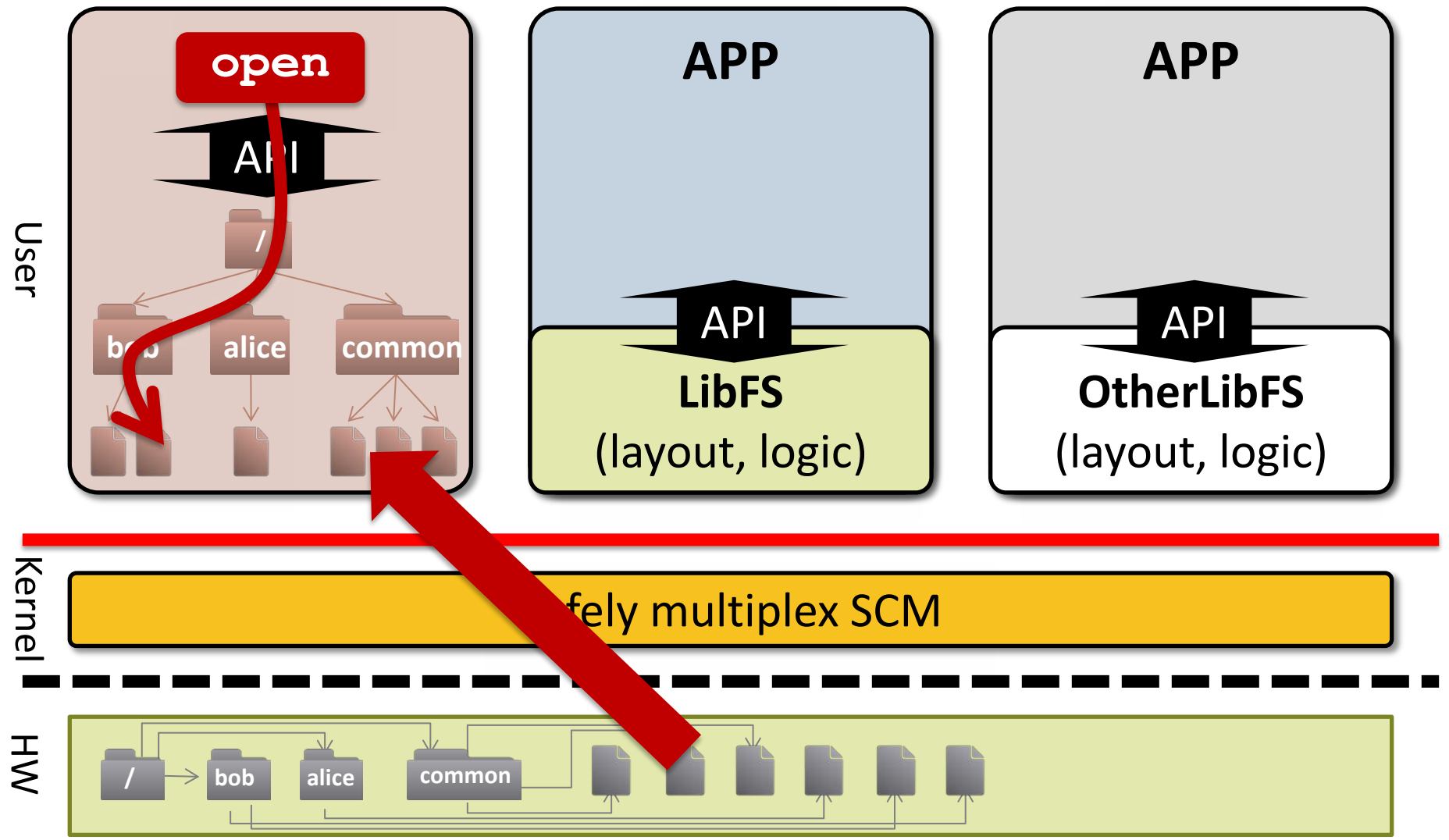
[Exokernel (MIT),
Nemesis (Cambridge)]

- Enable implementation flexibility
  - Optimize file-system interface semantics
  - Optimize operations regarding metadata

Aerie libFS in a nutshell



Aerie libFS in a nutshell

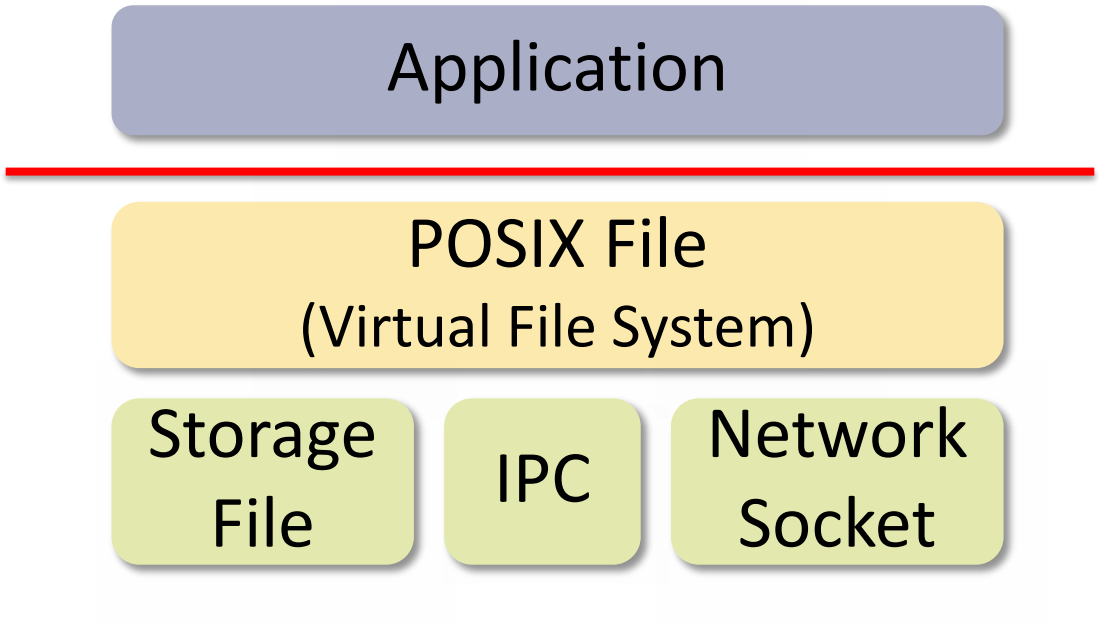


Outline

- Overview
- **Motivation: Interface flexibility**
- Aerie: In-memory library file systems
- Evaluation
- Conclusion

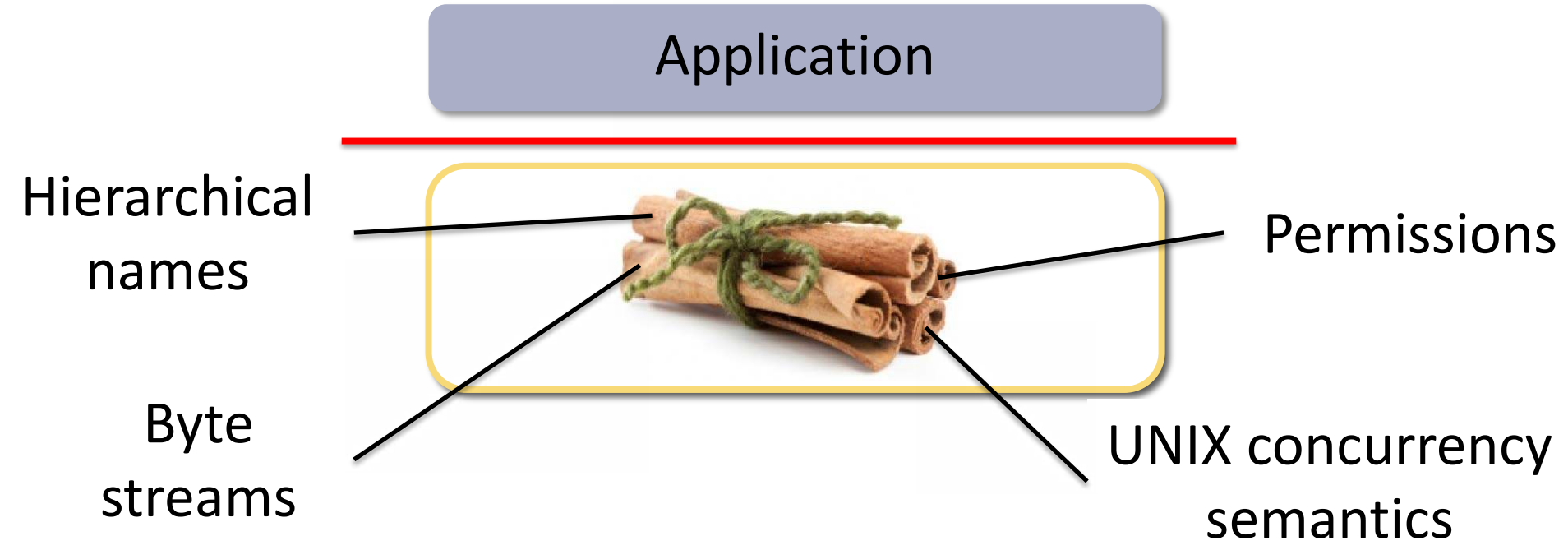
POSIX File: Expensive abstraction

- Universal abstraction: *Everything is a file*
  - Has generic-overhead cost



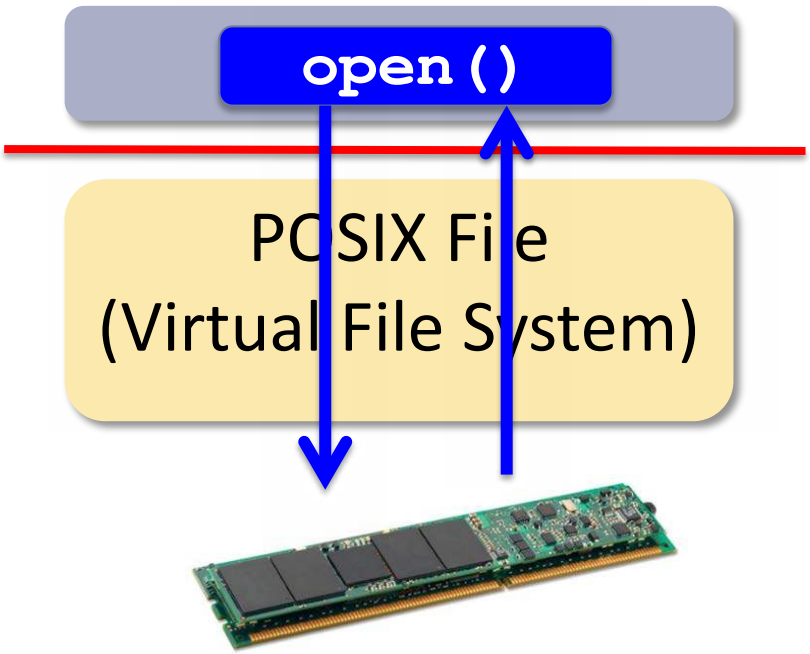
POSIX File: Expensive abstraction

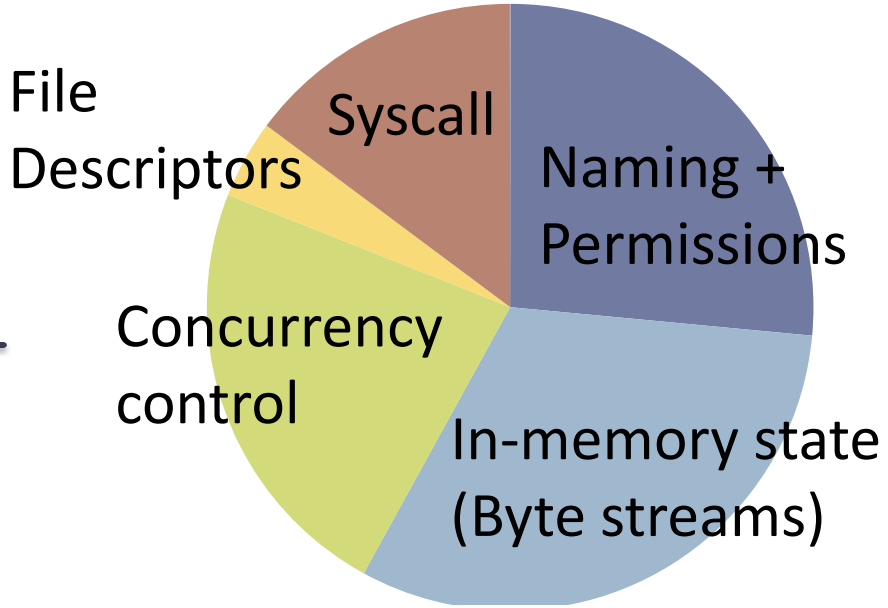
- Rigid interface and policies
  - Has fixed components and costs
  - Hinders application-specific customization



POSIX File: Expensive abstraction

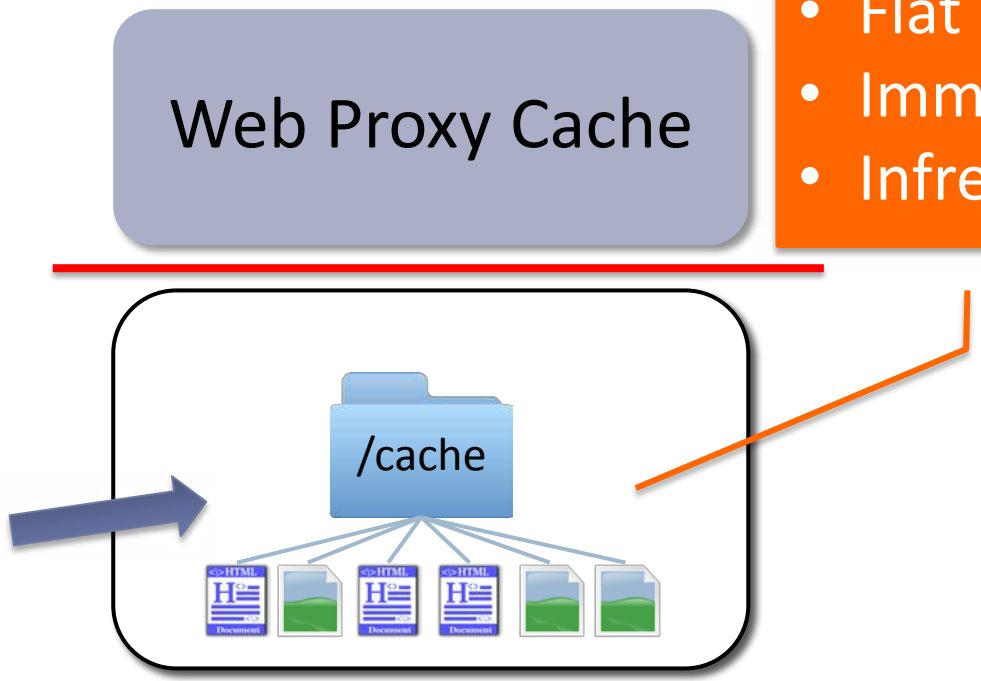
- Rigid interface and policies
  - Has fixed components and costs
  - Hinders application-specific customization





~2.5 µs ≈ 25x SCM latency

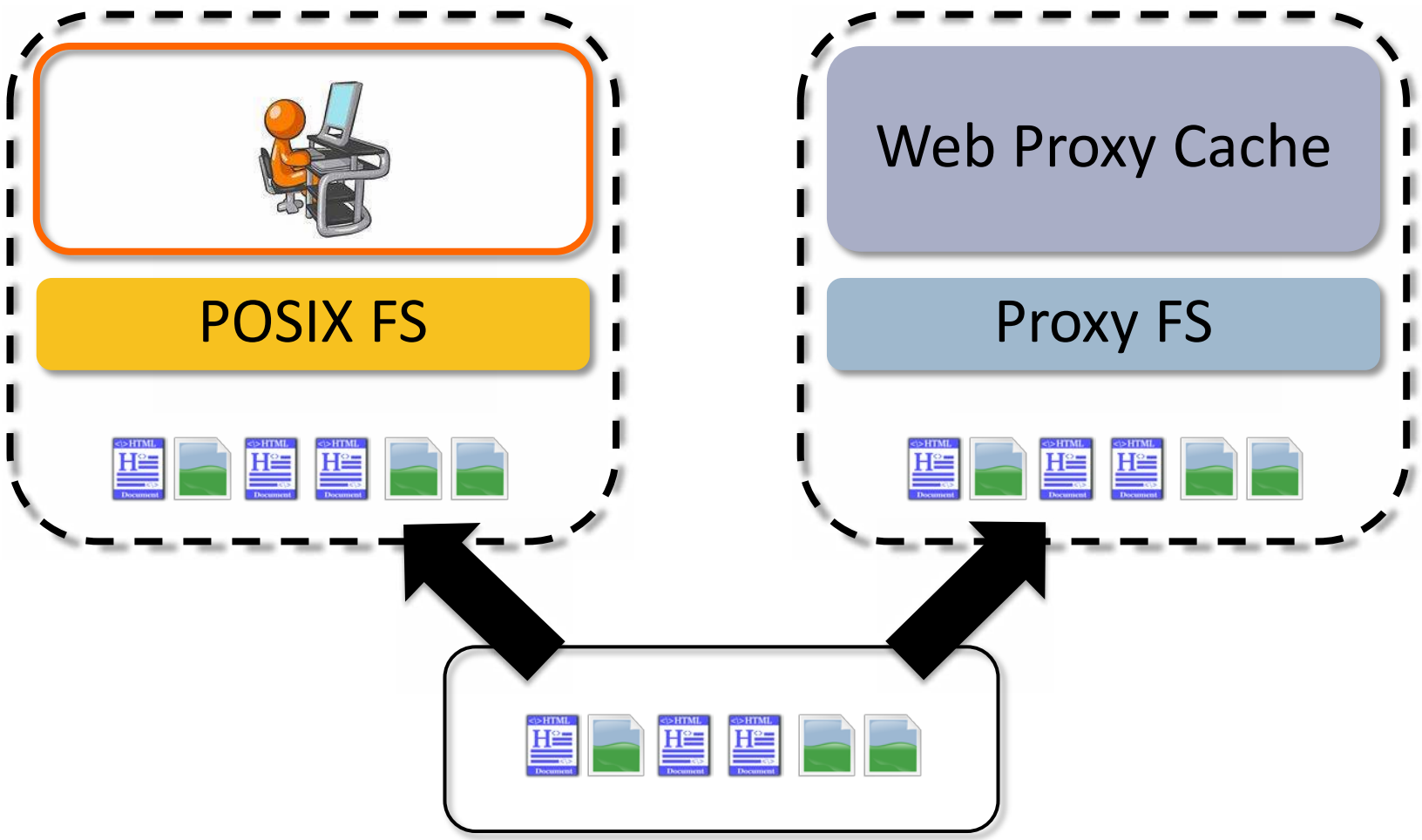
Motivating Example: Web Proxy



Characteristics

- Flat namespace
- Immutable files
- Infrequent sharing

Motivating Example: Web Proxy



Customizing the file system today

- Modify the kernel
- Add a layer over existing kernel file system
- Use a user-mode framework such as FUSE

Cumbersome options

Flexible interfaces more important than ever

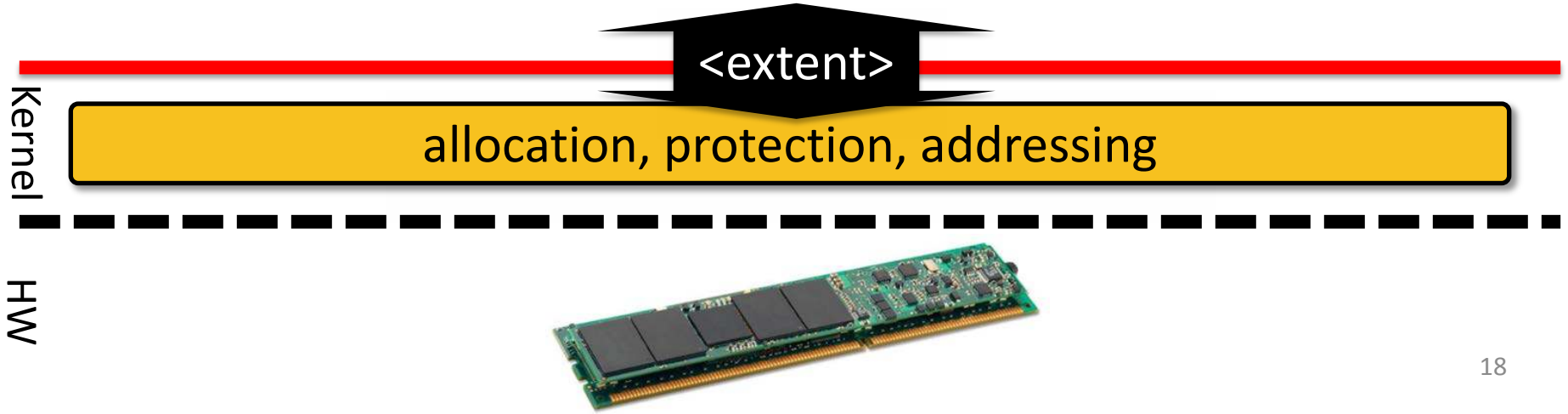
- Software interface overheads handicap fast SCM
- Flexible interface is a must for fast SCM
- Library file systems can help remove generic software overheads

Outline

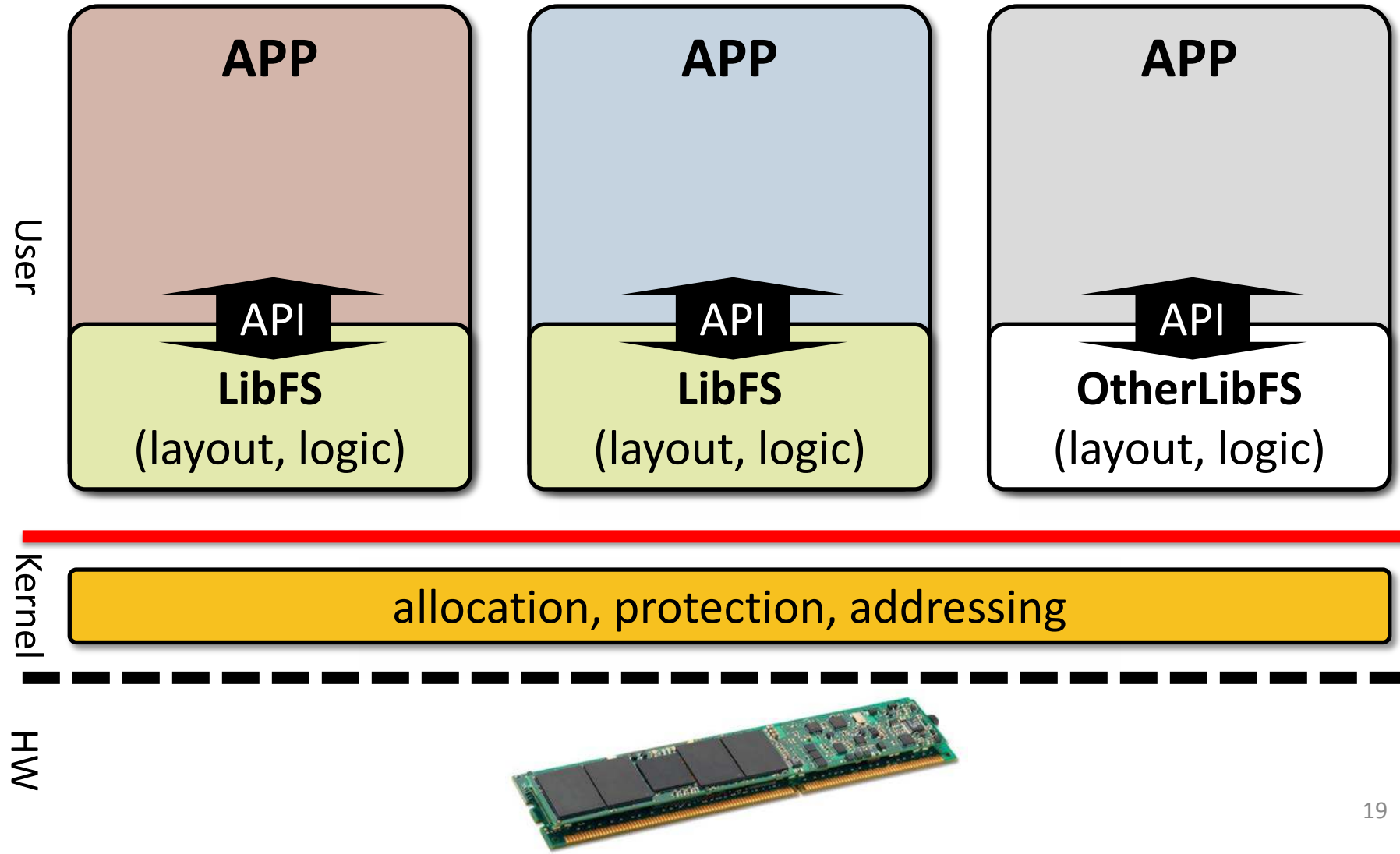
- Overview
- Motivation: Interface flexibility
- **Aerie: In-memory library file systems (libFS)**
- Evaluation
- Conclusion

Kernel safely multiplexes SCM

- Allocation: Allocates SCM regions (i.e. extents)
- Protection: Keeps track of region access rights
- Addressing: Memory-maps SCM regions



Library implements functionality

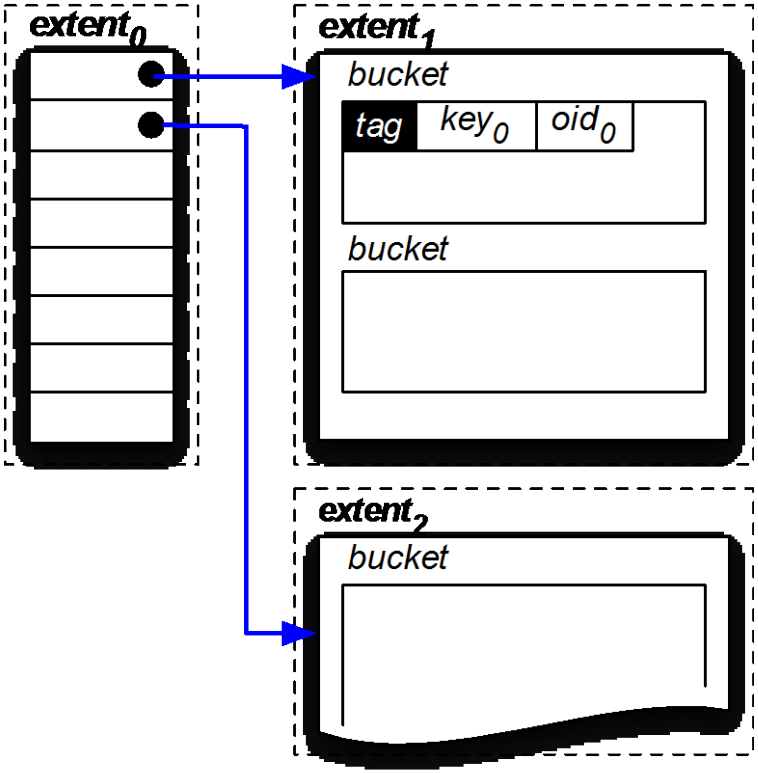


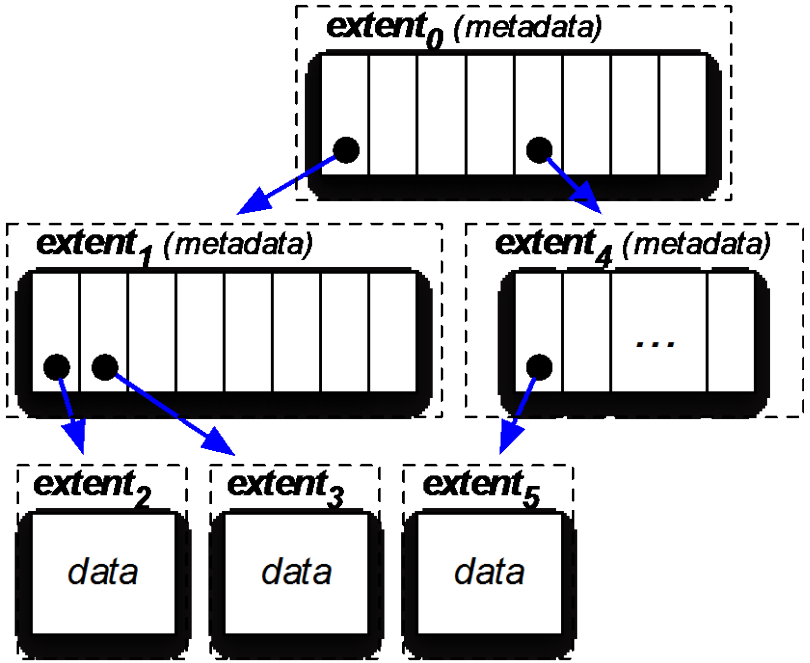
Implementing file-system features

- File-system objects
- Shared namespace
- Protection (access control)
- Integrity

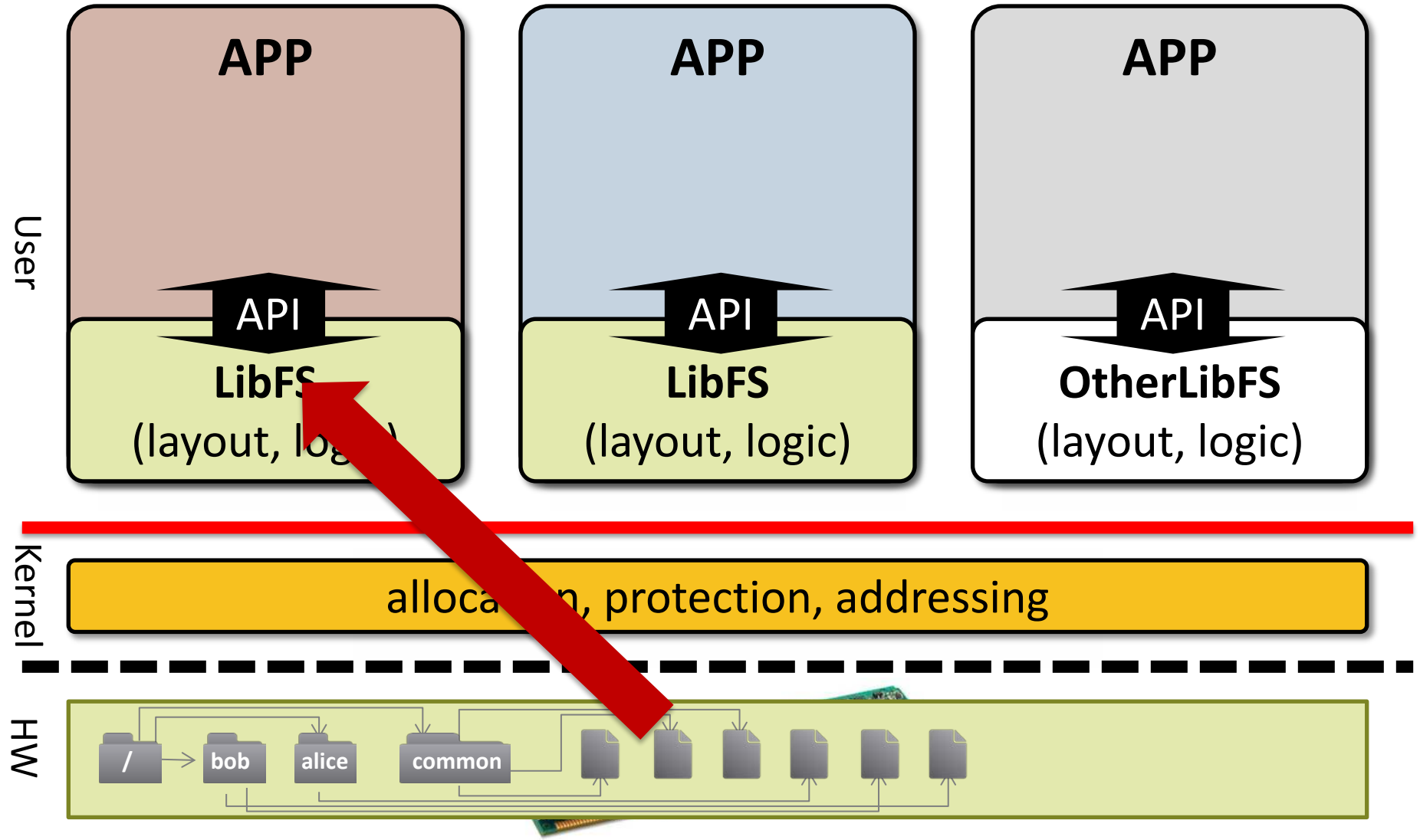
File-system objects build on SCM extents

- Collection (or directory)
  - key → object ID (oid)
- mFile (or memory file)
  - Offset → data extent ID

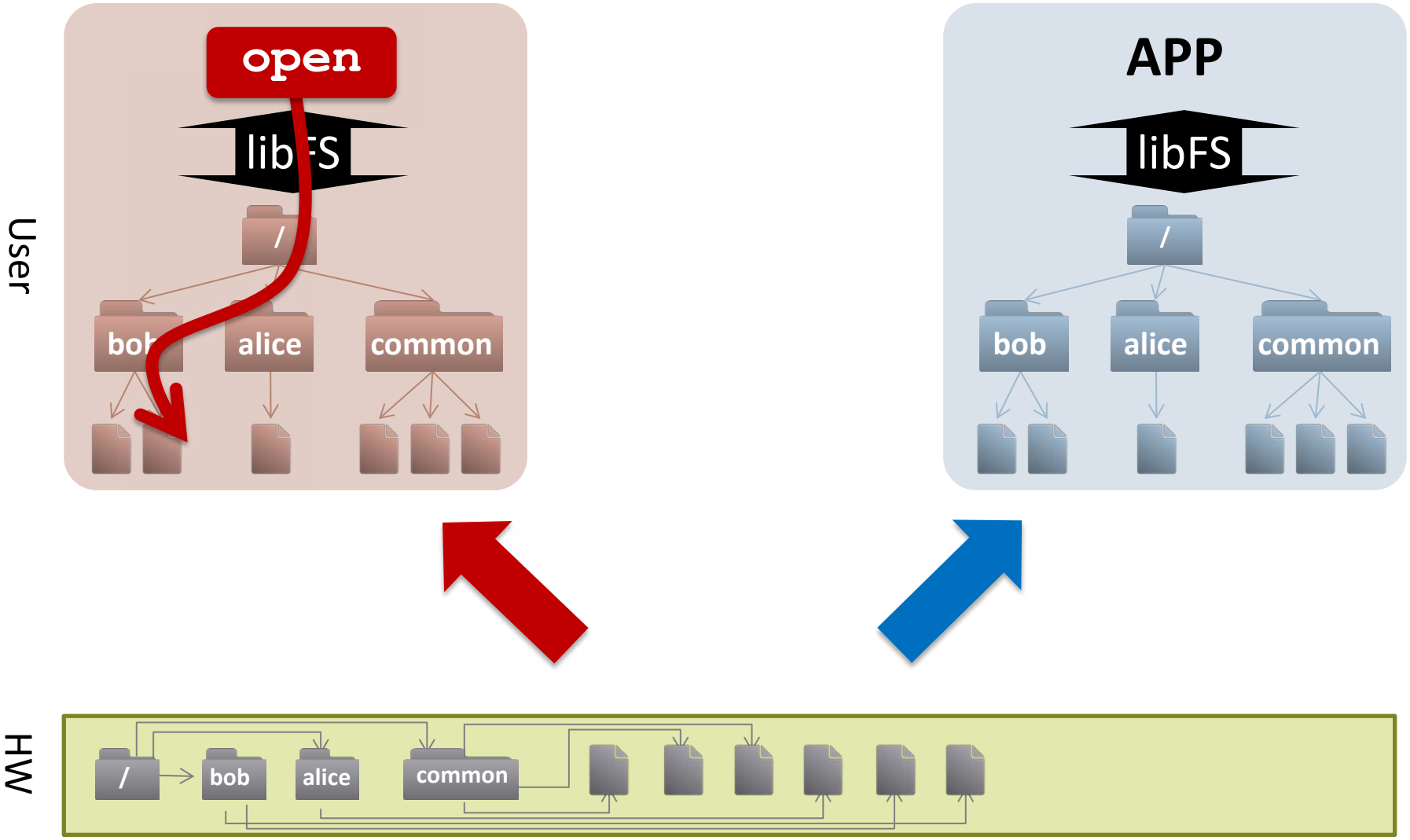




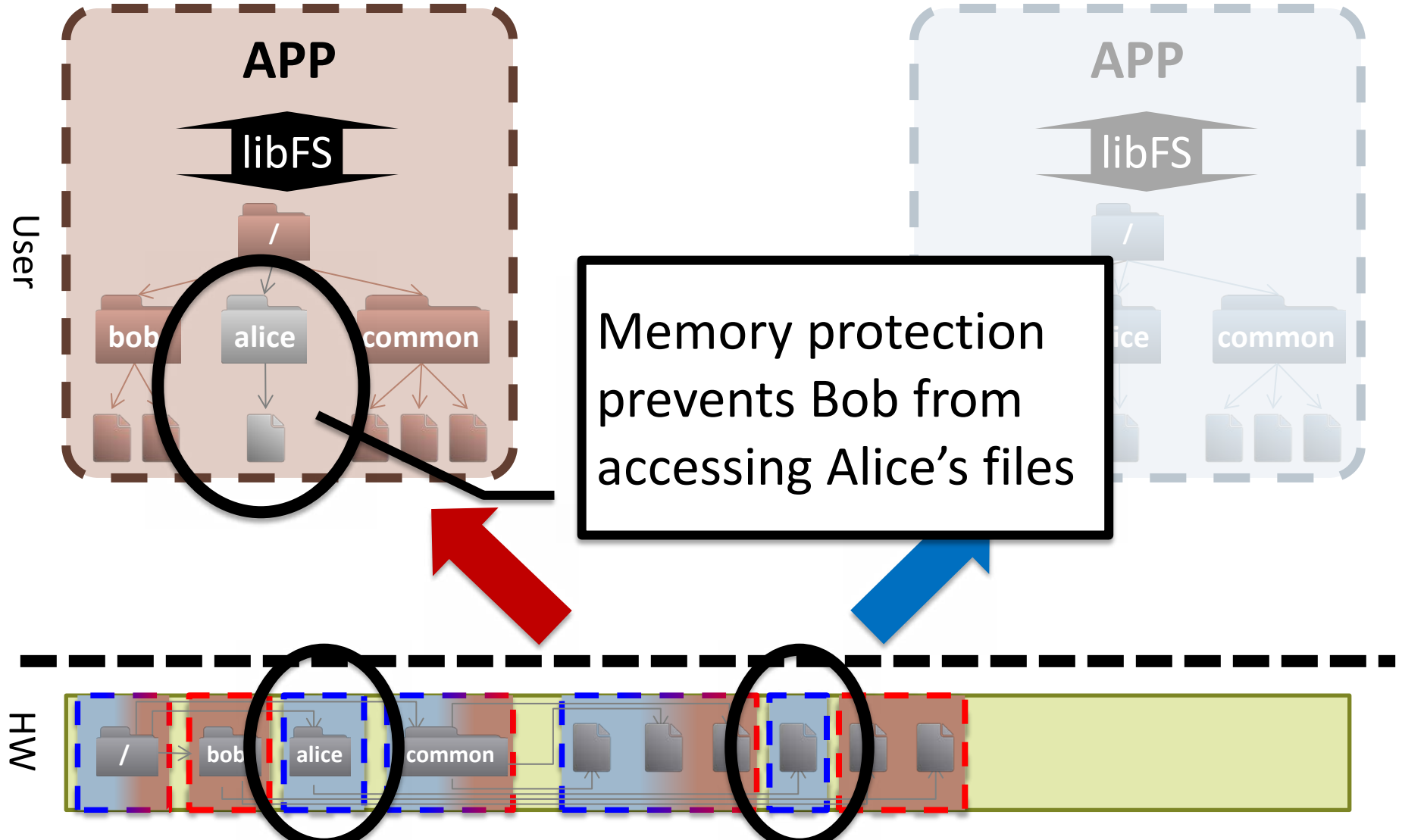
Shared namespace



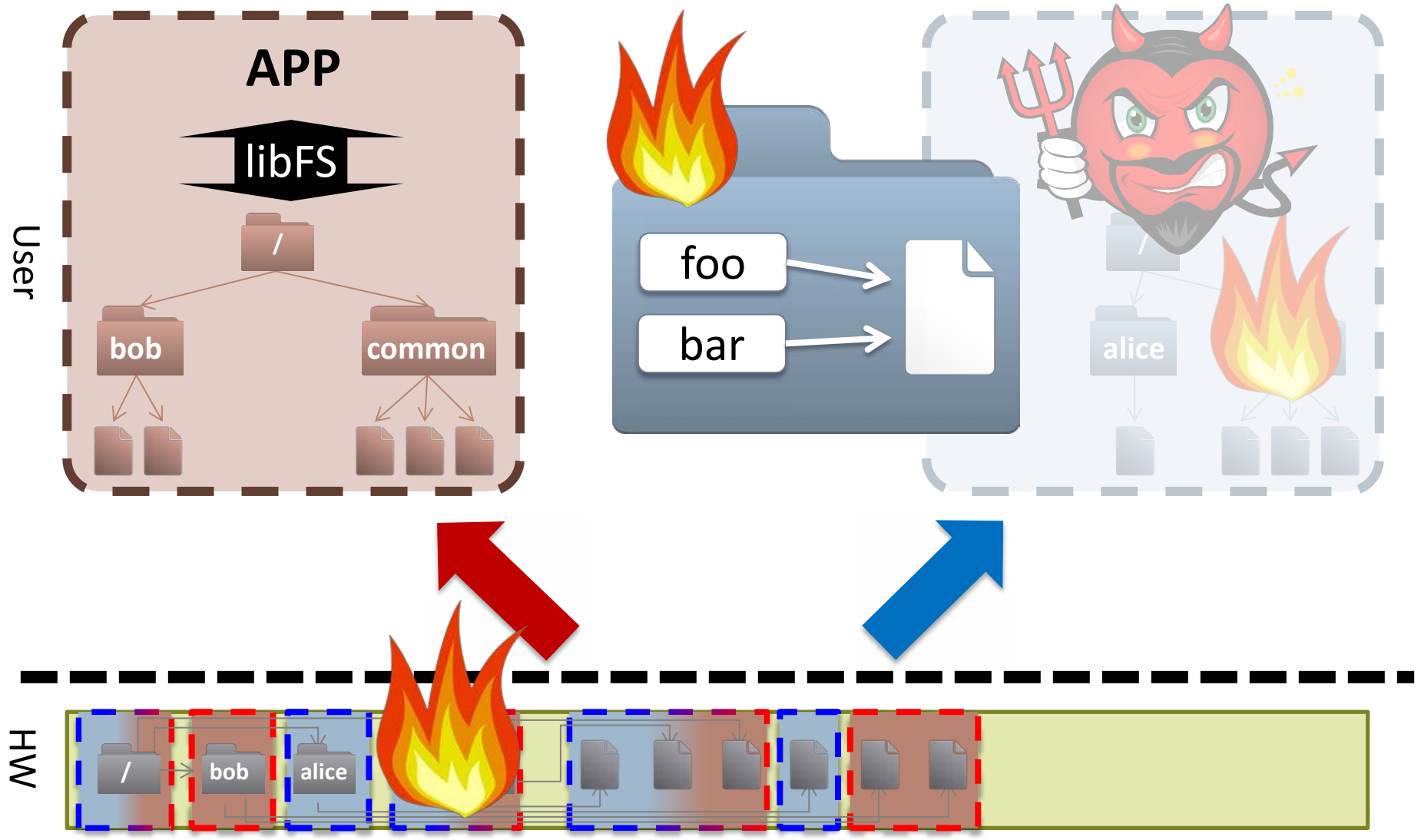
Shared namespace



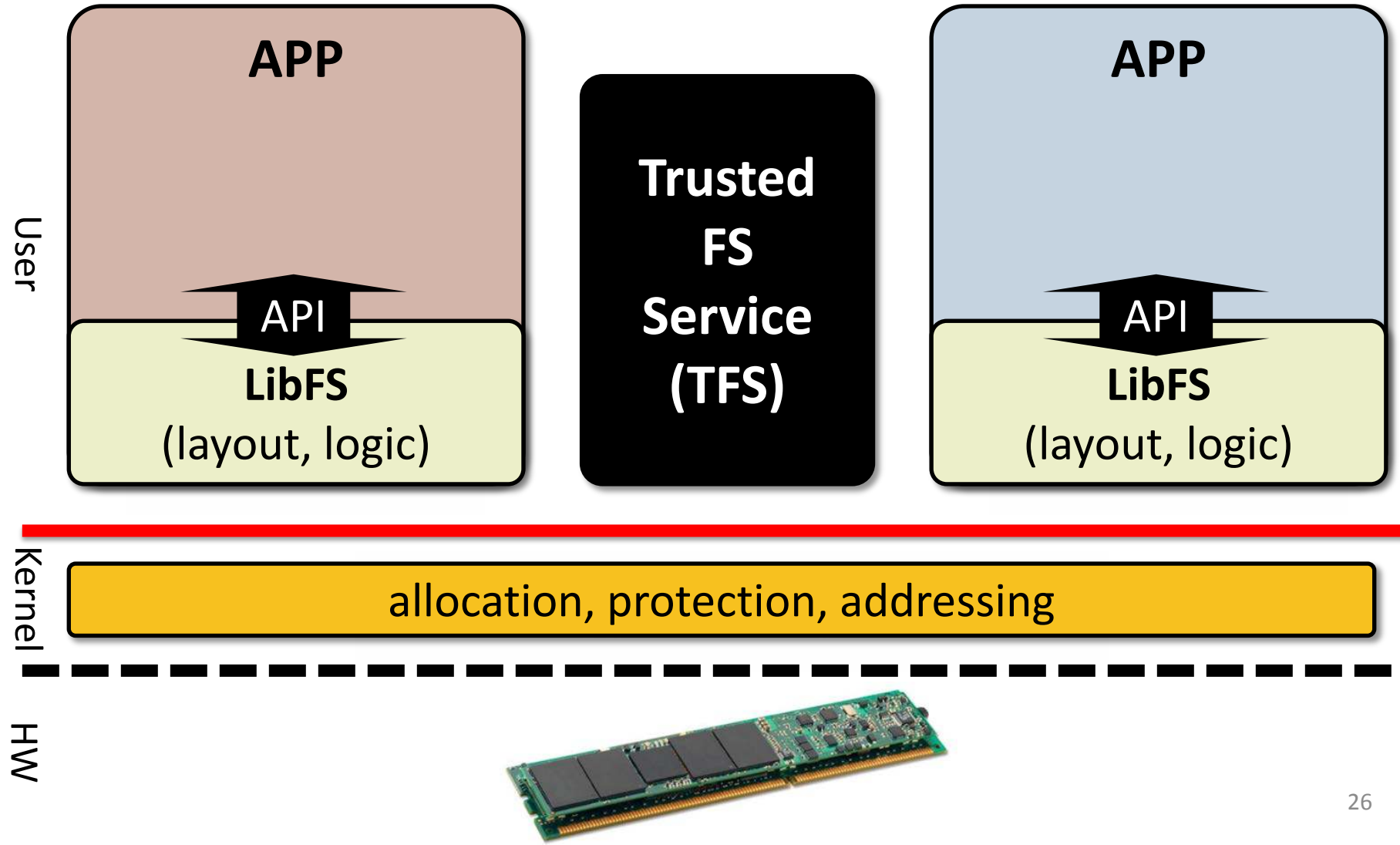
Decentralize access control via hardware-enforced permissions



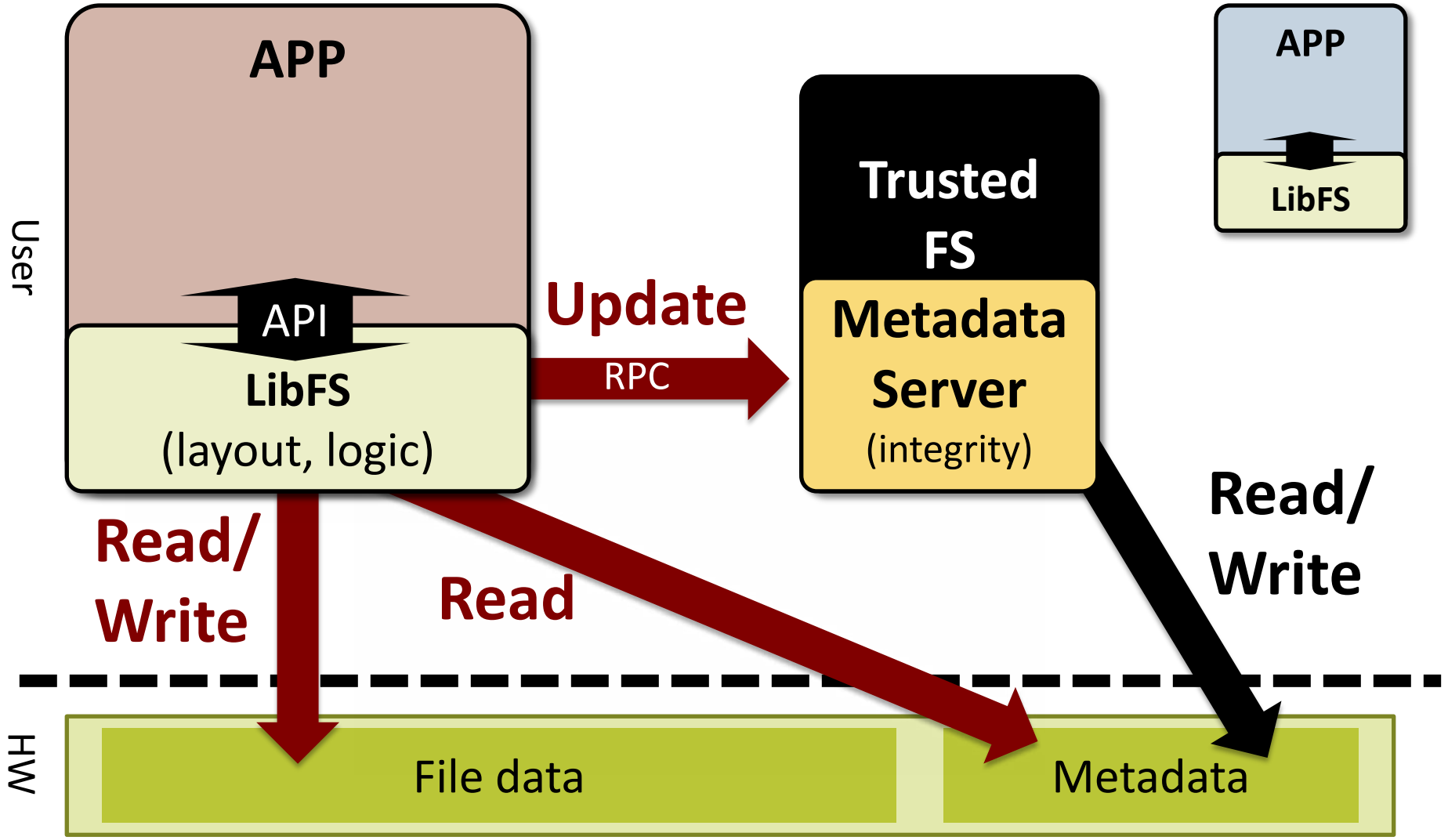
Hardware protection cannot guarantee integrity



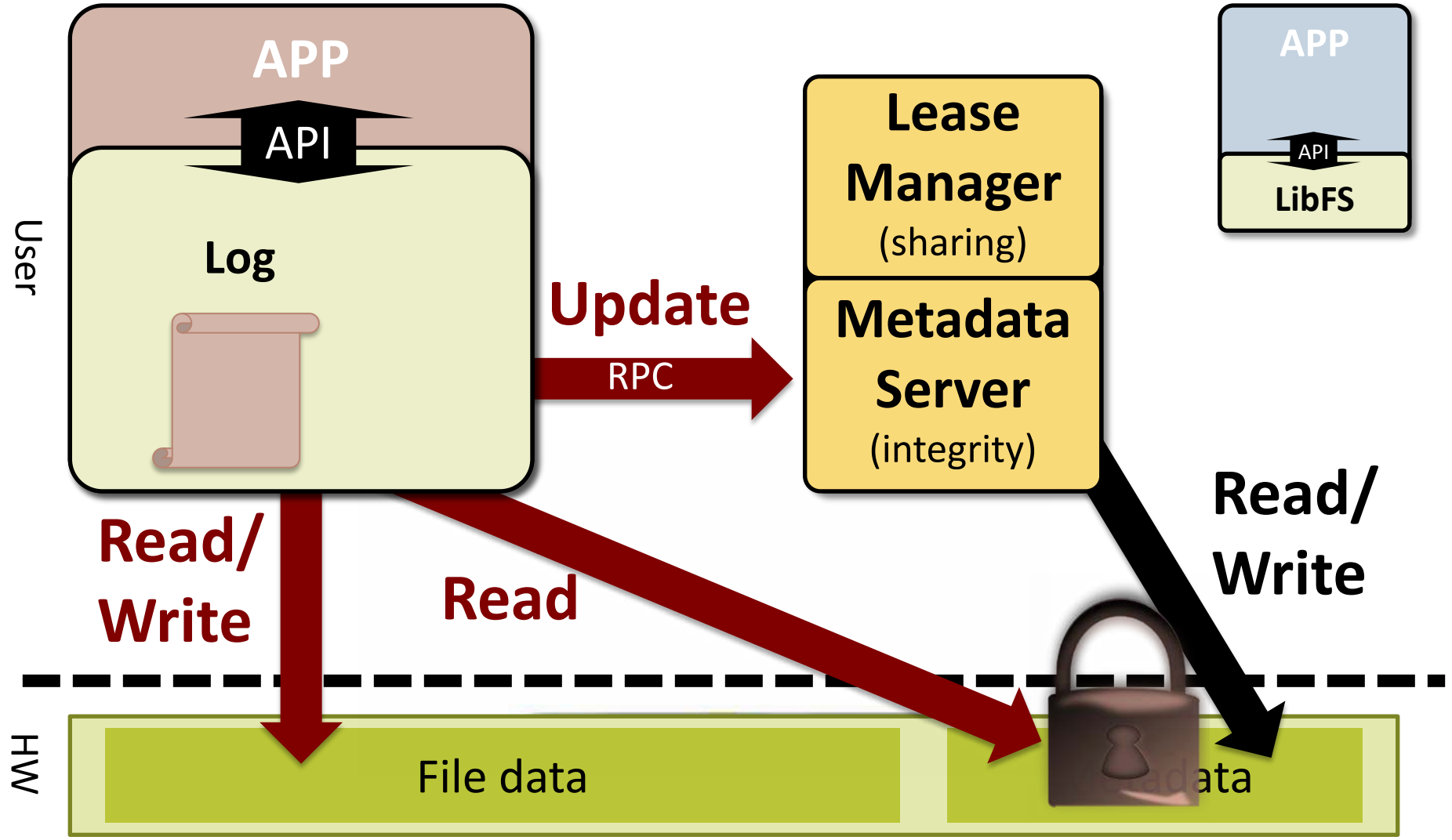
Integrity via Trusted File Service



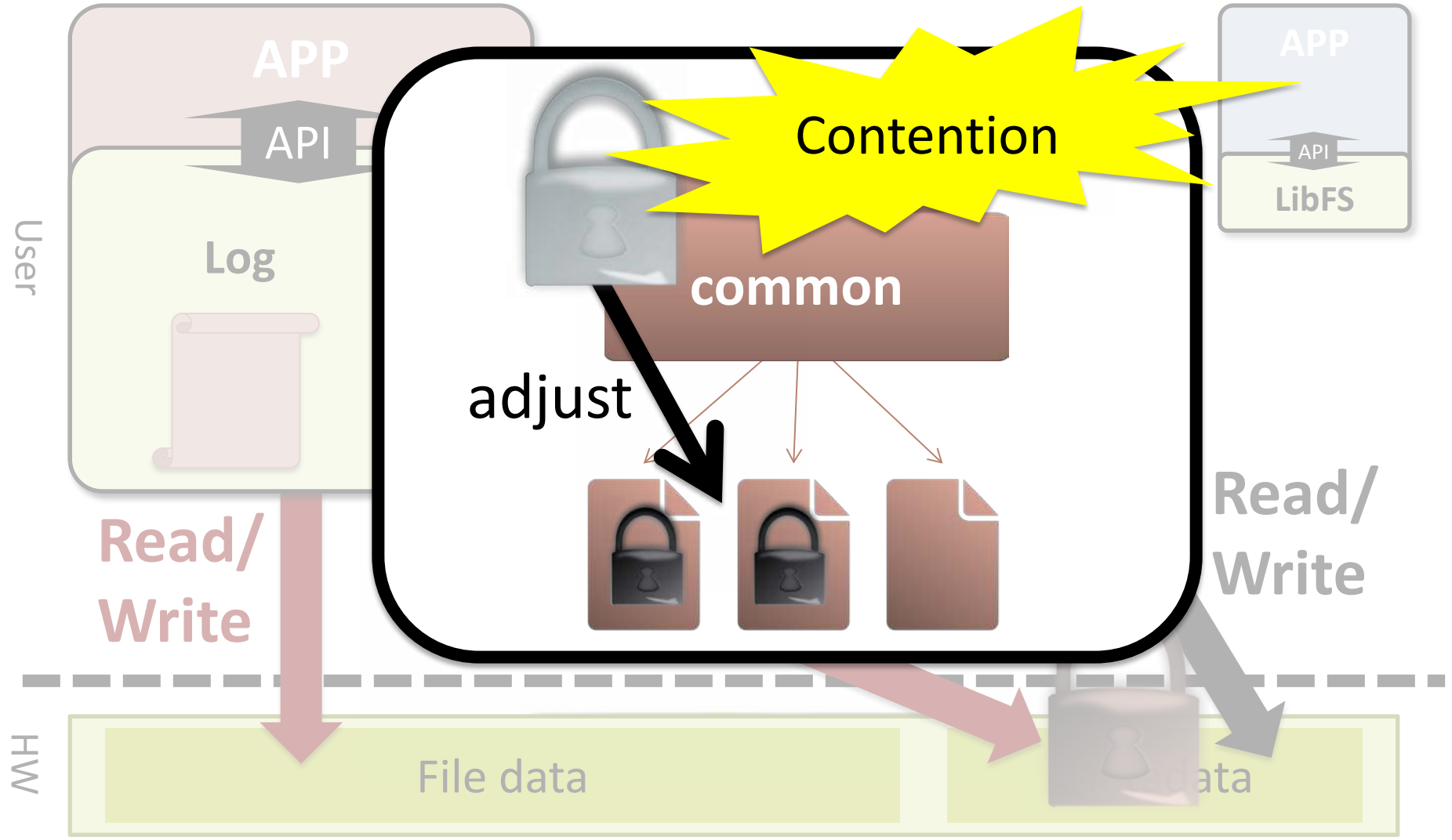
Decentralized architecture



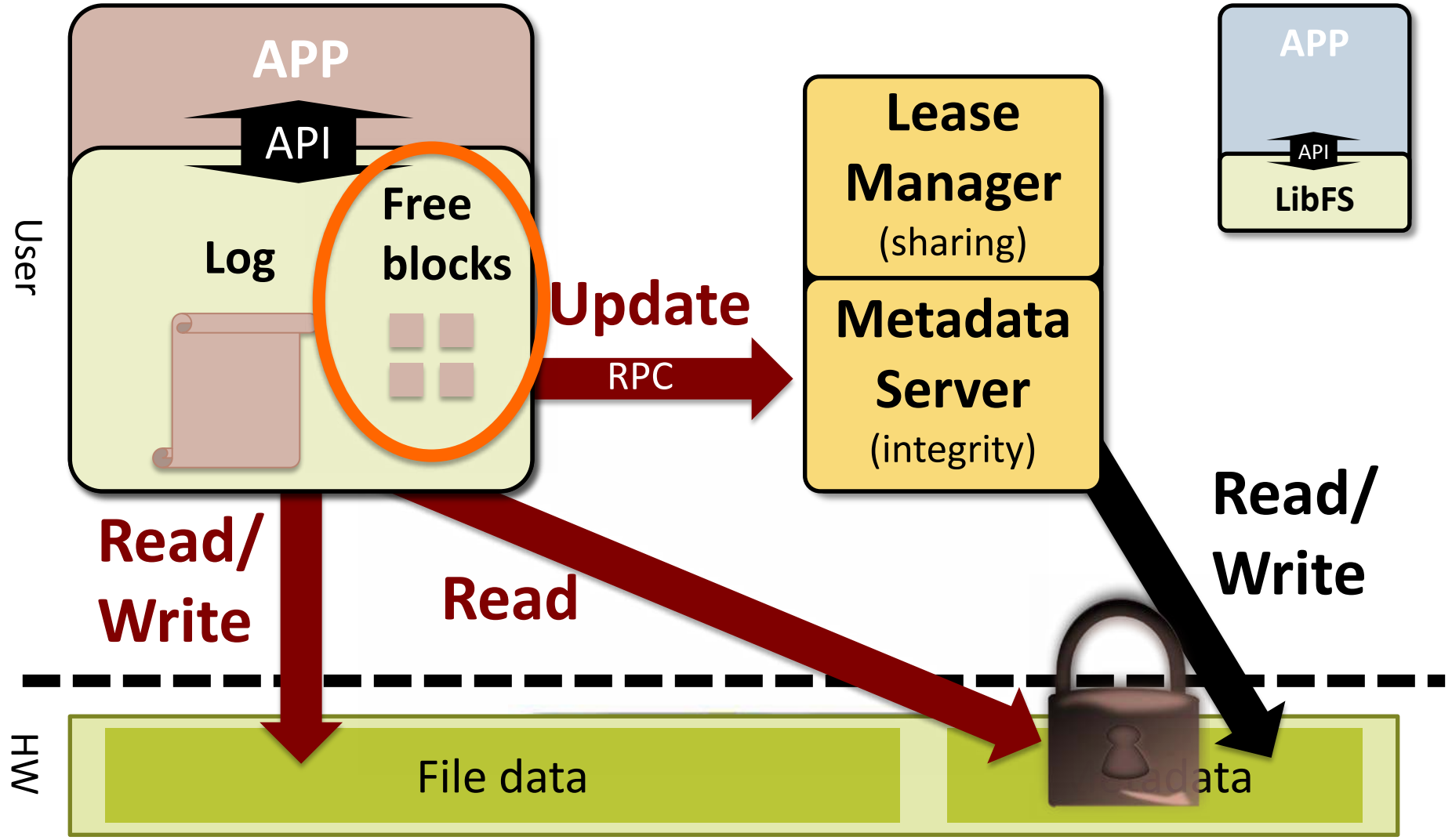
Reducing communication: Hierarchical leases + Batching



Reducing communication: Hierarchical leases + Batching



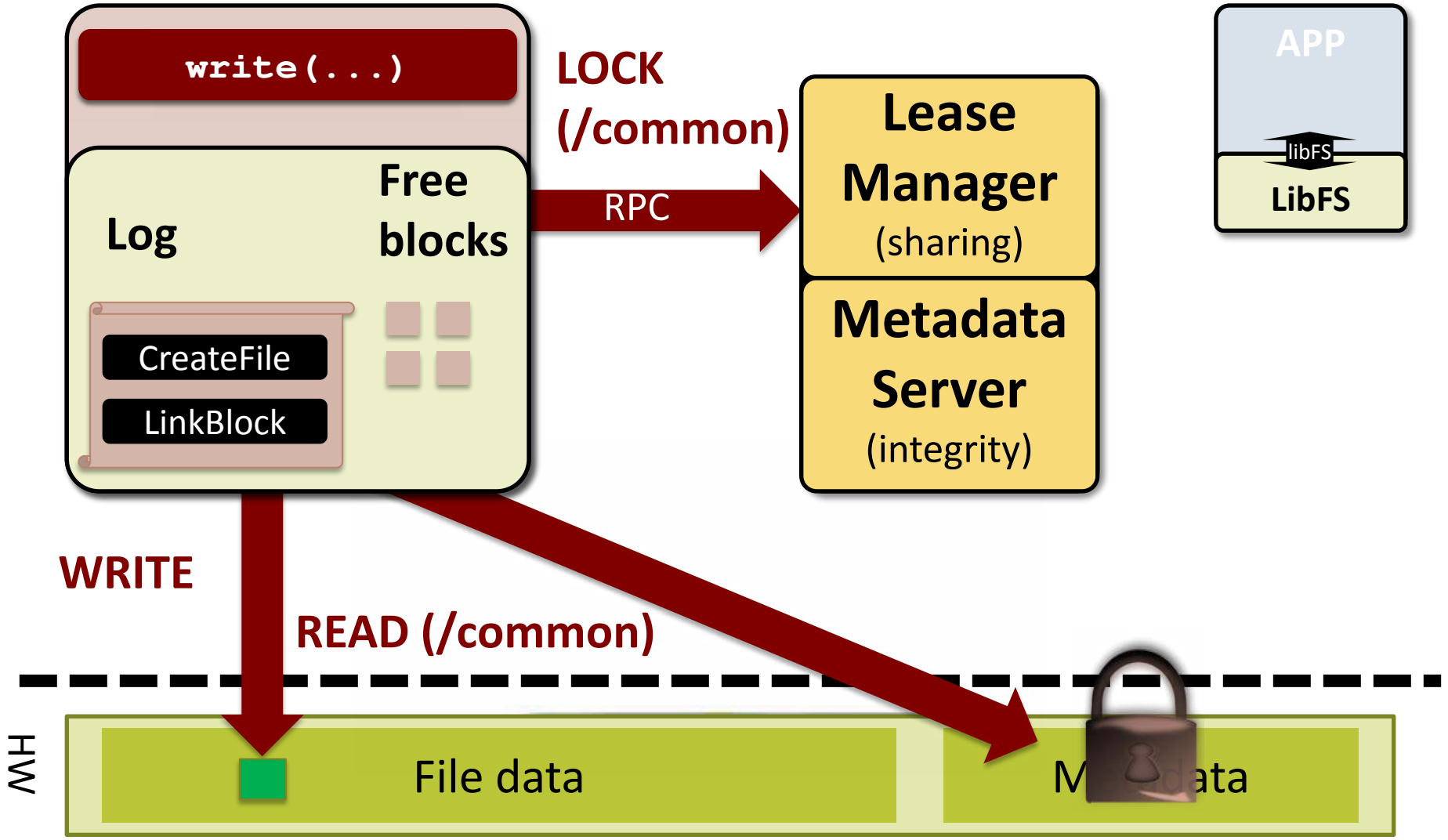
Reducing communication: Hierarchical leases + Batching



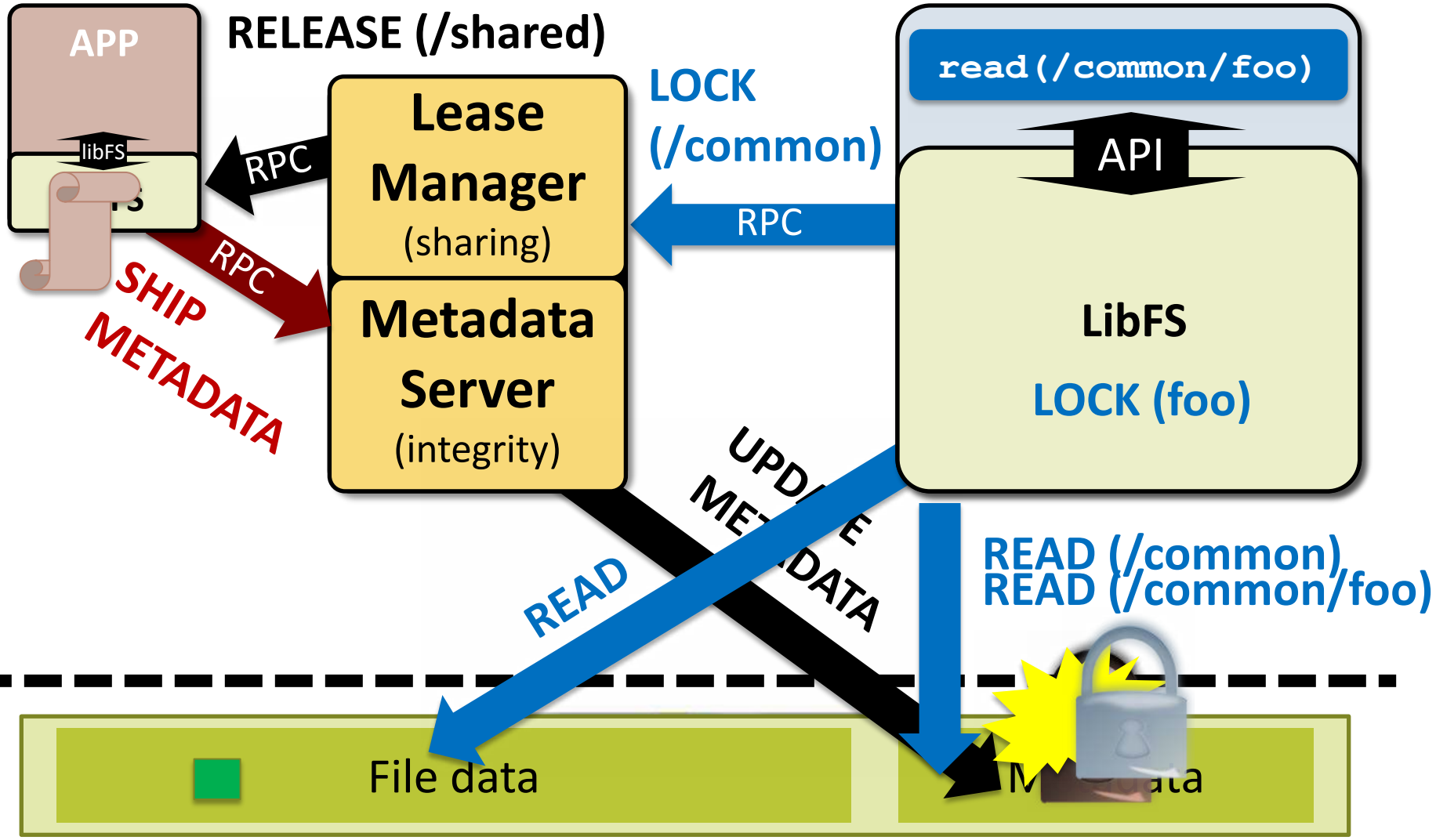
Prototype Implementation

- Extent API by Linux 3.2.2 x86-64 kernel modifications
- Communication via loopback RPC
- Crash consistency through
  - x86 CLFLUSH instruction (cache line flush)
  - Redo logging
- SCM emulation using DRAM

Example: A shared file



Example: A shared file



File Systems

Functionality: PXFS

- POSIX interface: open/read/write/unlink
- Hierarchical namespace
- POSIX concurrency semantics
- File byte streams

File Systems

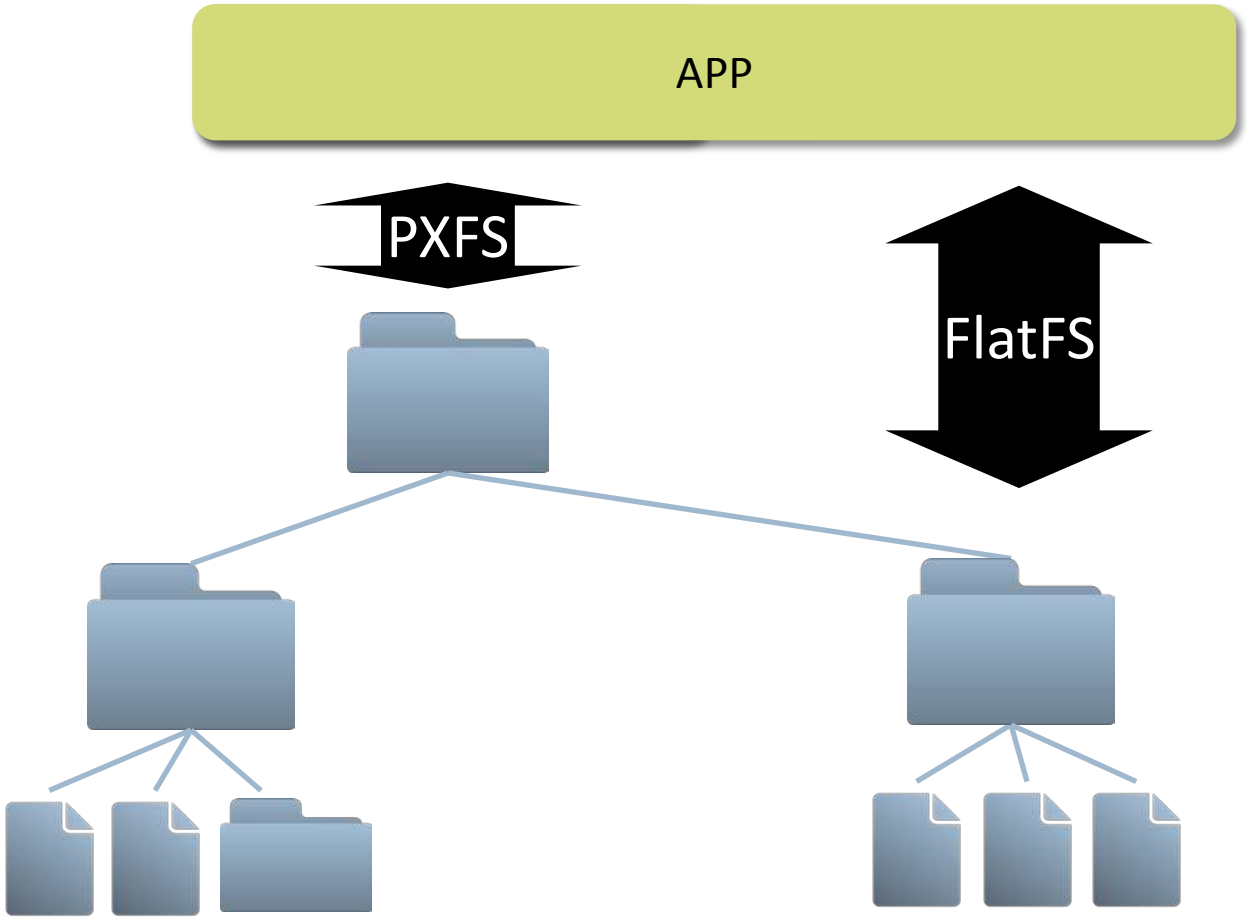
Functionality: PXFS

- POSIX interface: open/read/write/unlink
- Hierarchical namespace
- POSIX concurrency semantics
- File byte streams

Optimization: FlatFS

- Key-value interface: put/get/erase
- Flat namespace
  - Simplifies name resolution
- KV-store concurrency semantics
  - Reduce in-memory state
- Short, immutable files
  - Simplify storage allocation

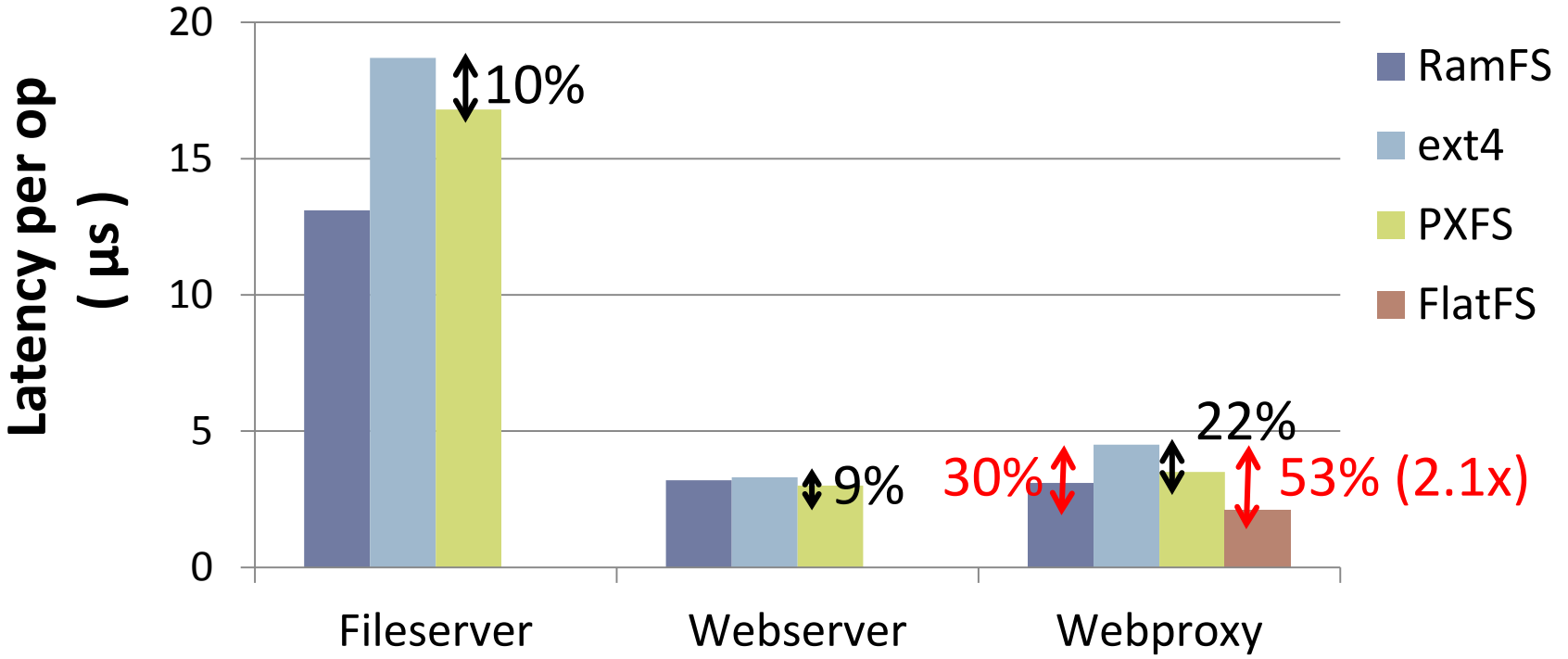
File Systems



Performance Evaluation

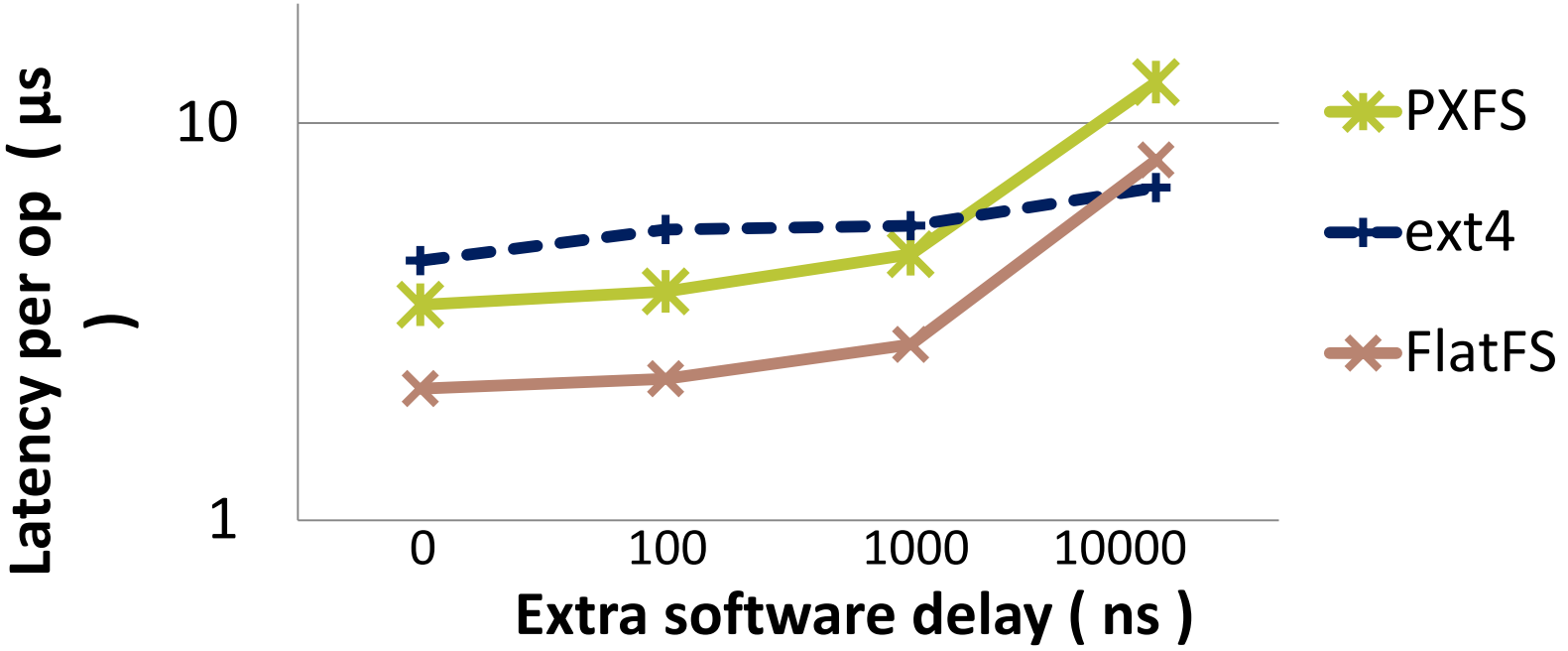
- Performance model
  - Writes to DRAM + software created delay
  - Reads to DRAM
- Configurations
  - RamFS: In-memory kernel FS
  - Ext4: ext4fs + RAM-disk
  - LibFS: PXFS and FlatFS
- Filebench workloads: Fileserver, Webserver, Webproxy

Application-workload performance



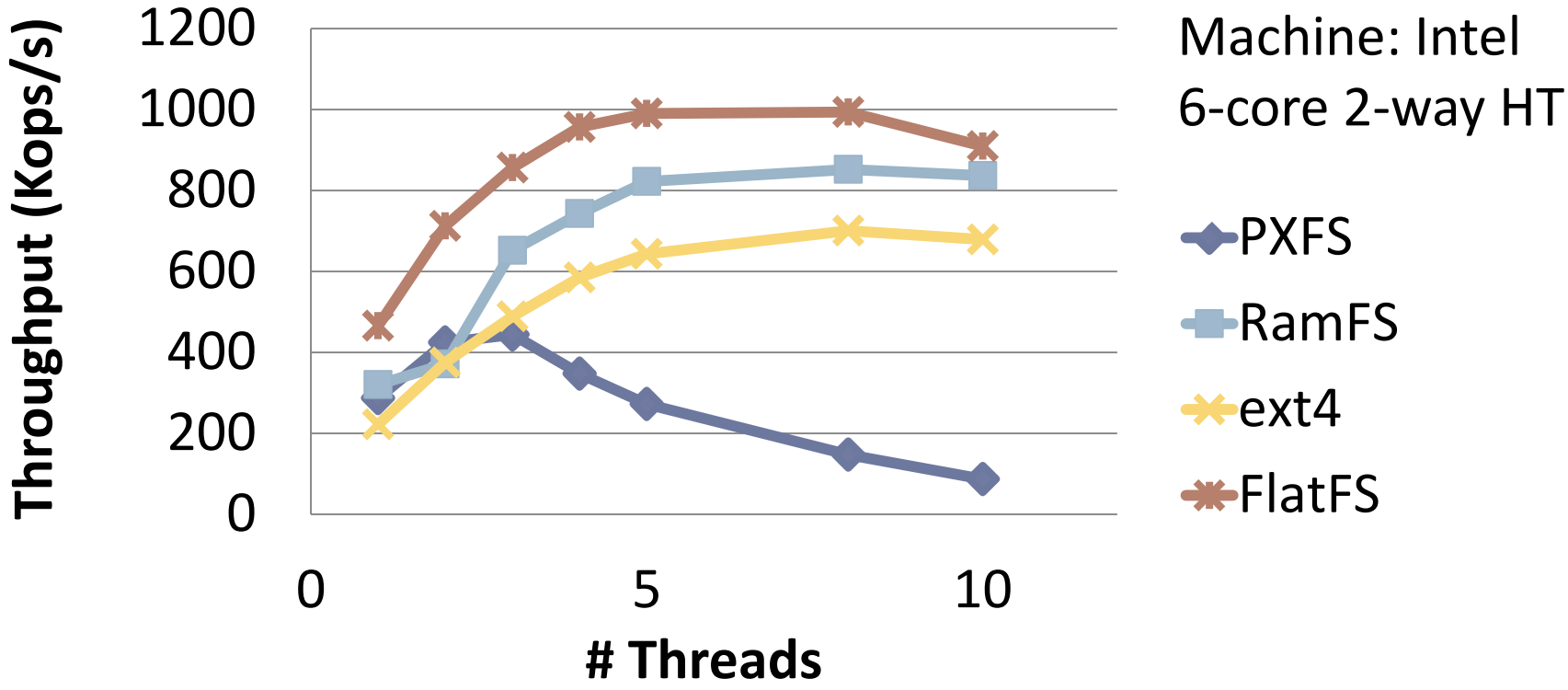
- PXFS performs better than kernel-mode FS
- FlatFS exploits app semantics to improve performance

Sensitivity to SCM performance: Webproxy



- Shorter SCM latencies favor
  - Direct access via load/store instructions
  - Interface specialization

Scalability: Webproxy



- FlatFS retains its benefits over kernel-mode file systems

Conclusion

- Software interface overheads handicap fast SCM
- Flexible interface is a must for fast SCM
- Aerie: Library file systems help remove generic overheads for higher performance
  - FlatFS improves performance by up to 110%

Thank you! Questions?