Exporting Kernel Page Caching

for Efficient User-Level I/O

R.P. Spillane, S. Dixit. S. Archak, S. Bhanage, and E. Zadok

Stony Brook University

http://www.fsl.cs.sunysb.edu/

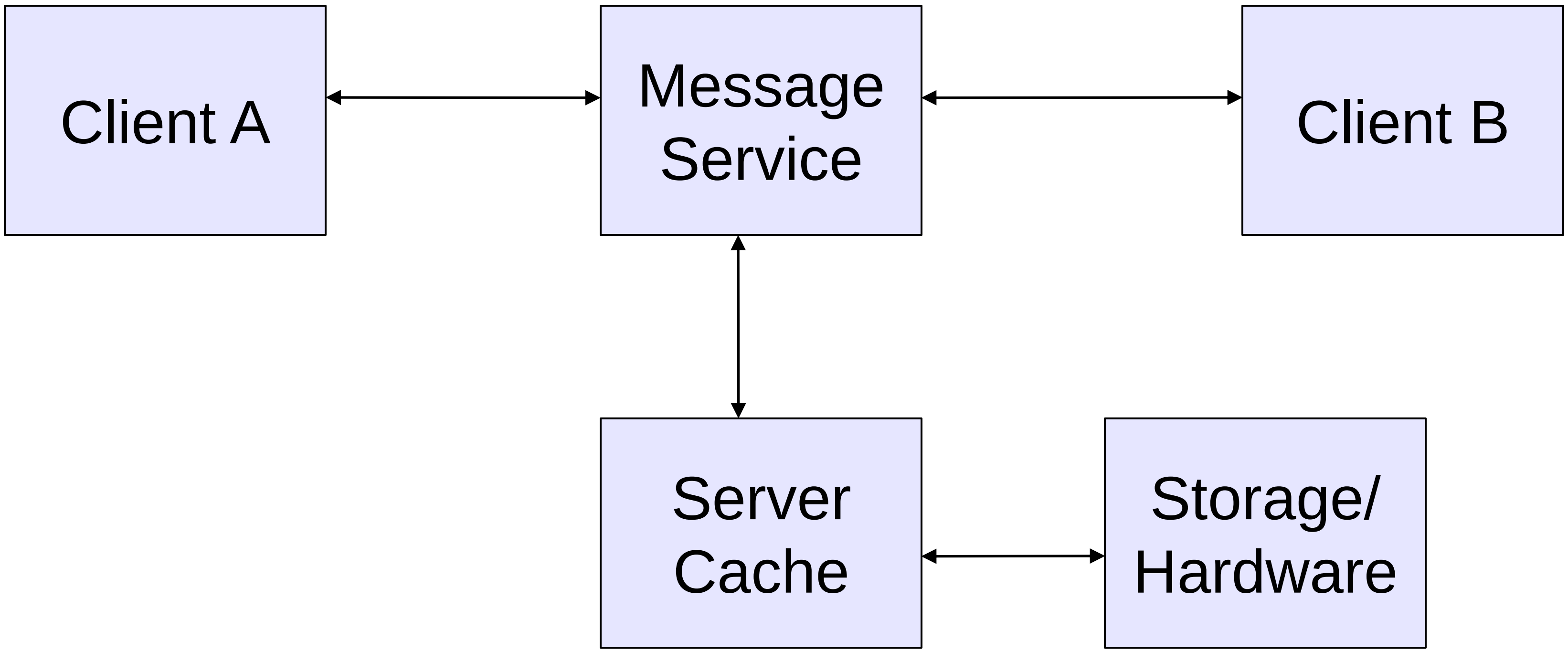
STONY
BROOK
STATE UNIVERSITY OF NEW YORK

The Problem

- Kernel obstructs mature user-level storage stacks
  - Write-ordering and fsync is still a mystery
  - Crude sharing of the page cache
  - Hard to be a system service provider
- So, I've got an OS I'd like to sell you…
  - New (mico-)kernels not easily adopted
  - 3.3 million lines of driver code in Linux 2.6

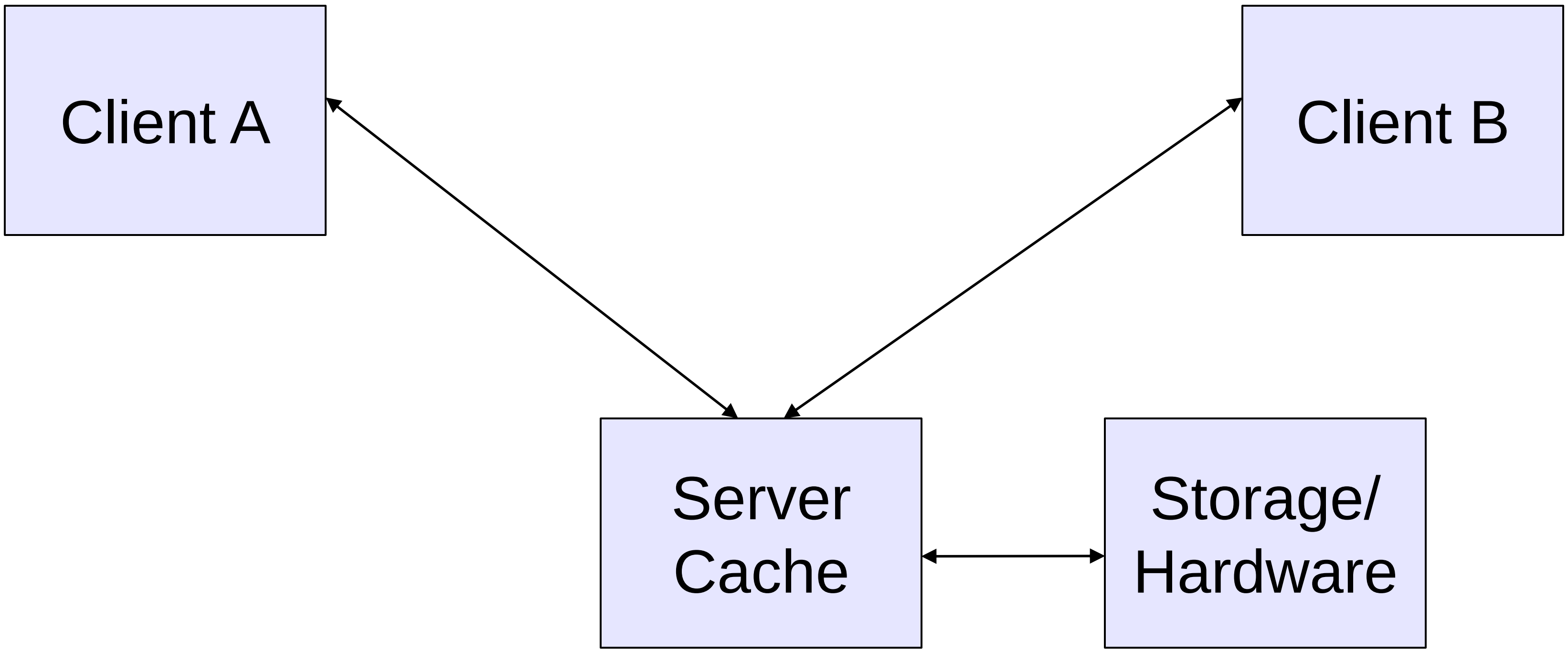
The case for cache lock-in

- Standard practice:



The case for cache lock-in

- Standard practice (faster, single-node):
- *Unsafe*



Client A

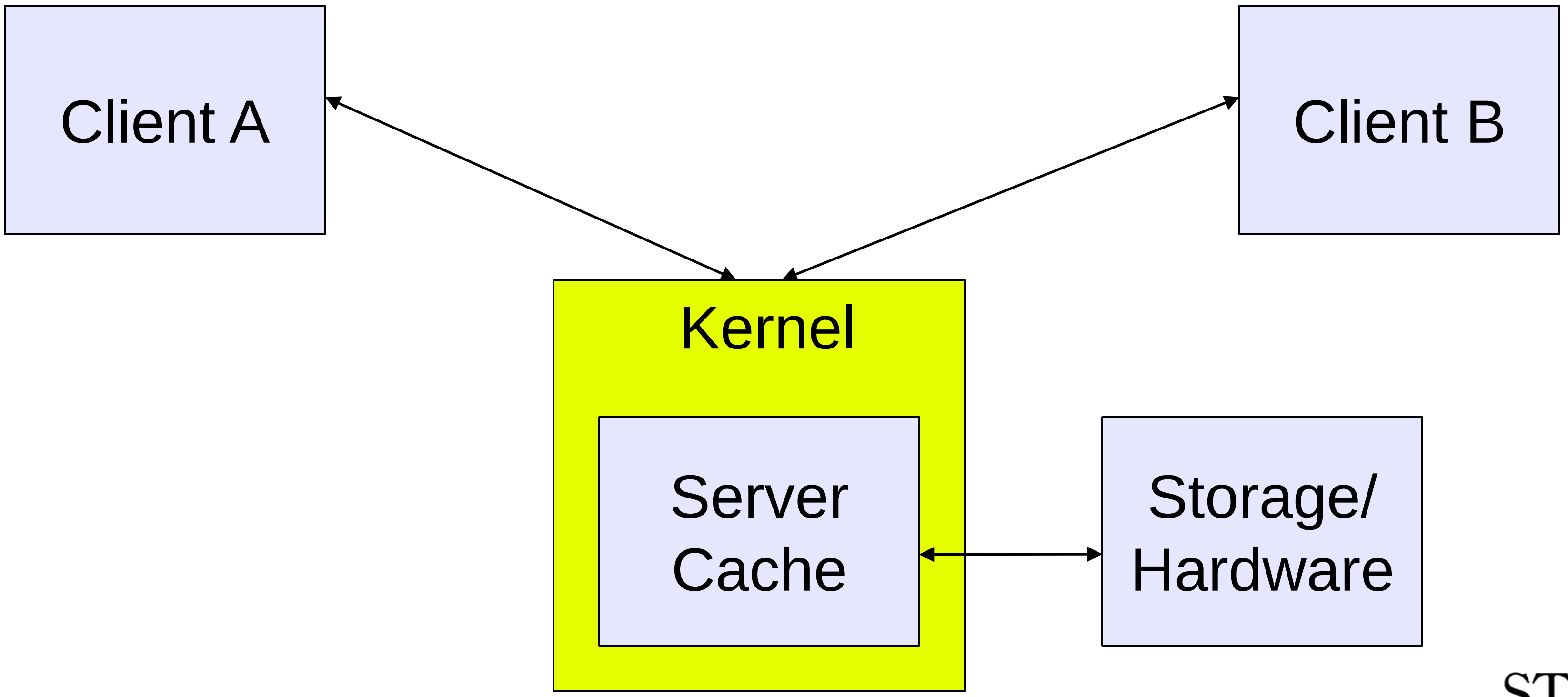
Client B

Server Cache

Storage/ Hardware

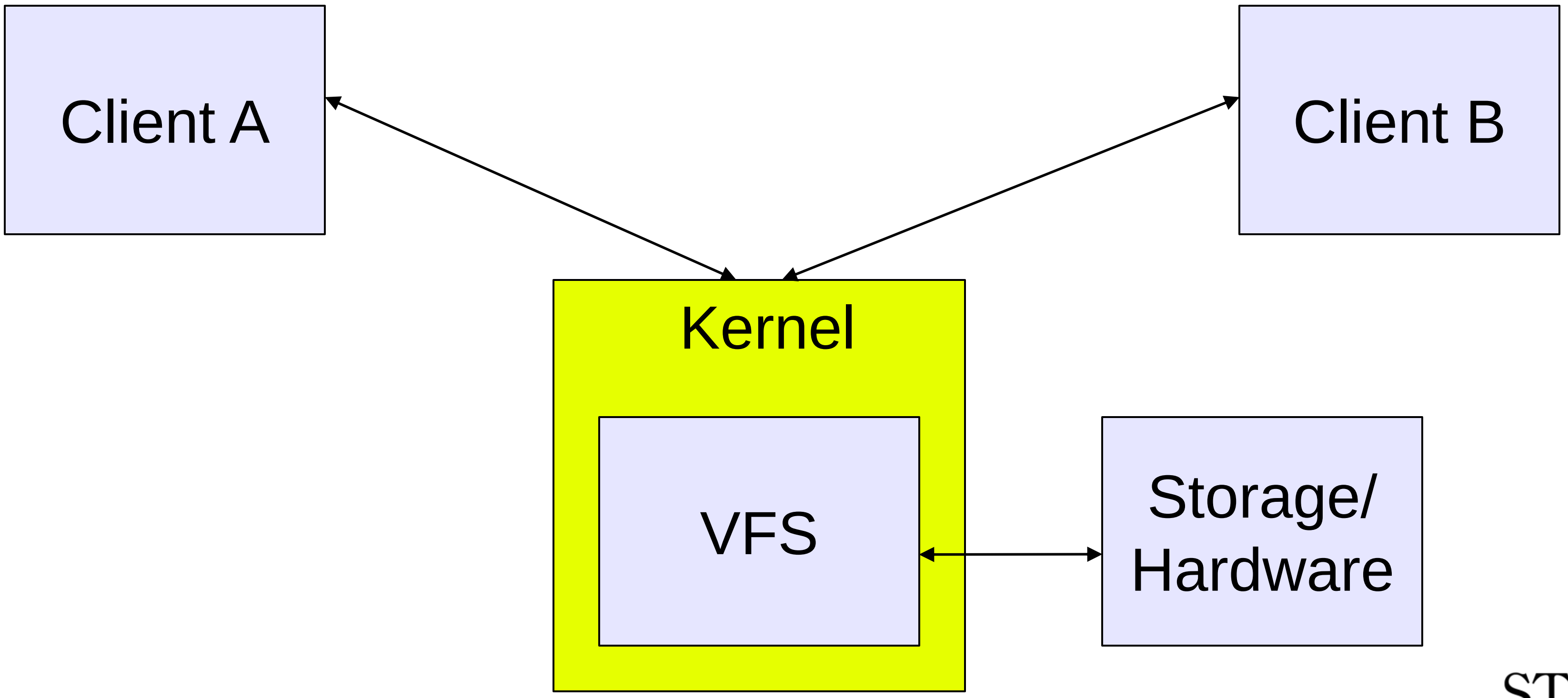
The case for cache lock-in

- Making shared caching ***safe***



The case for cache lock-in

- Do this for FS stuff and its the VFS



But what about *other* stuff??

Everything isn’t an “F”S

- Transactional APIs
  - Berkeley DB
  - Stasis
- Object Stores
  - BeFS
  - HFaD

Not all FSes are VFSes

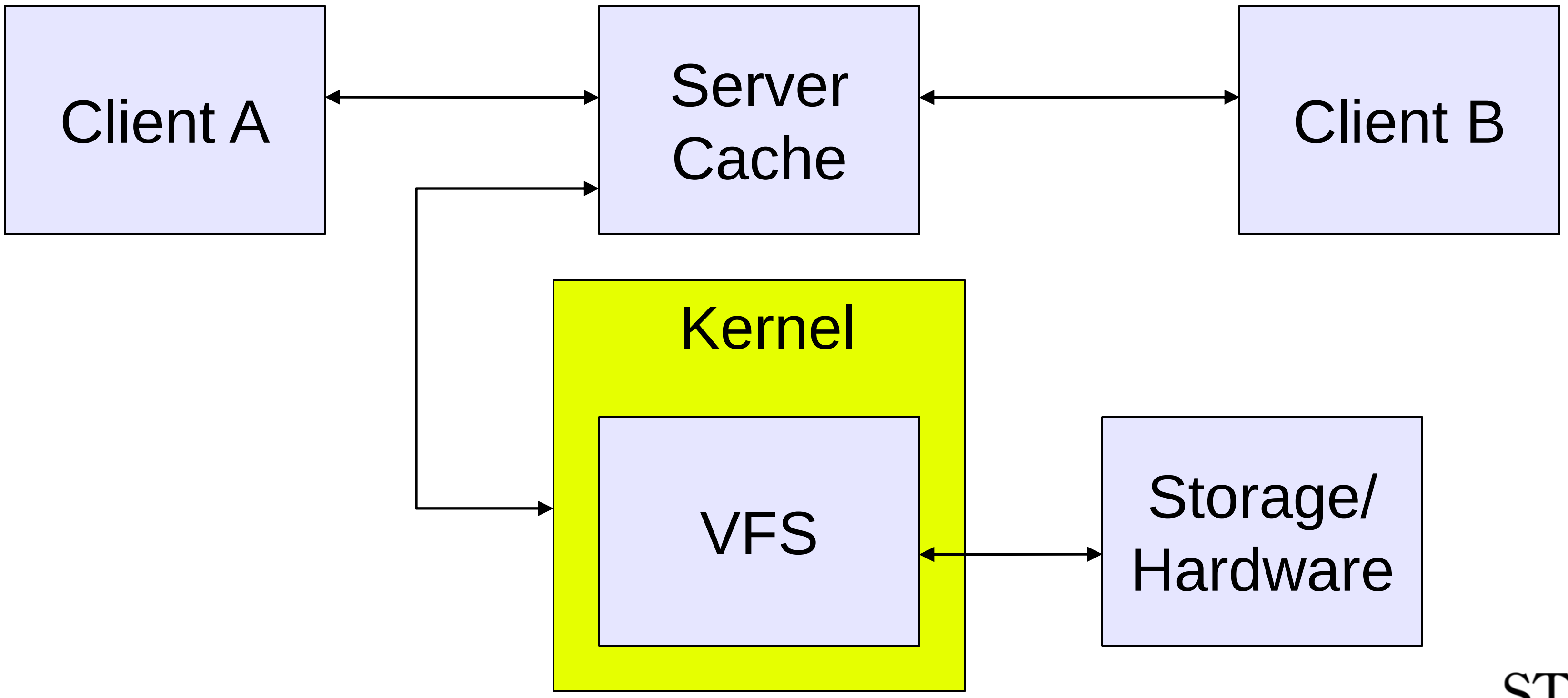
- Use VFS, use VFS caches
    - Provenance Tracking
    - Restartable File Systems
    - Transactional File Systems
    - Distributed Systems
- Use FUSE, use VFS caches
    - Either FUSE is slow or…
    - You use caching: bad (e.g., no provenance)

What we want

- Put cache in shared memory
- Protect it with required context switch
- Have some way of interacting with block devices
- Otherwise though, be a process
    - Ease of development
    - Controlled crashing

Doing it at user-level

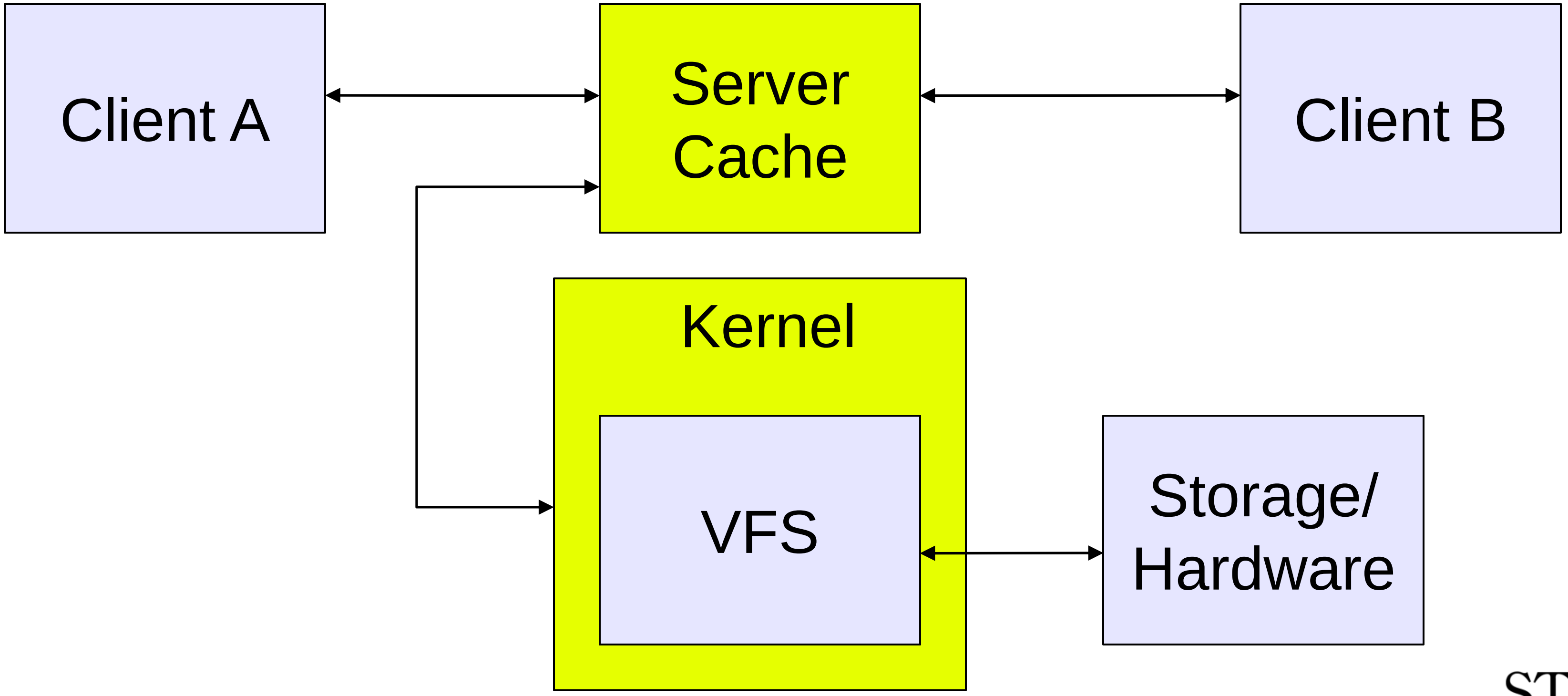
- Redundant service implementations...
- Naively insecure...



05/06/2010

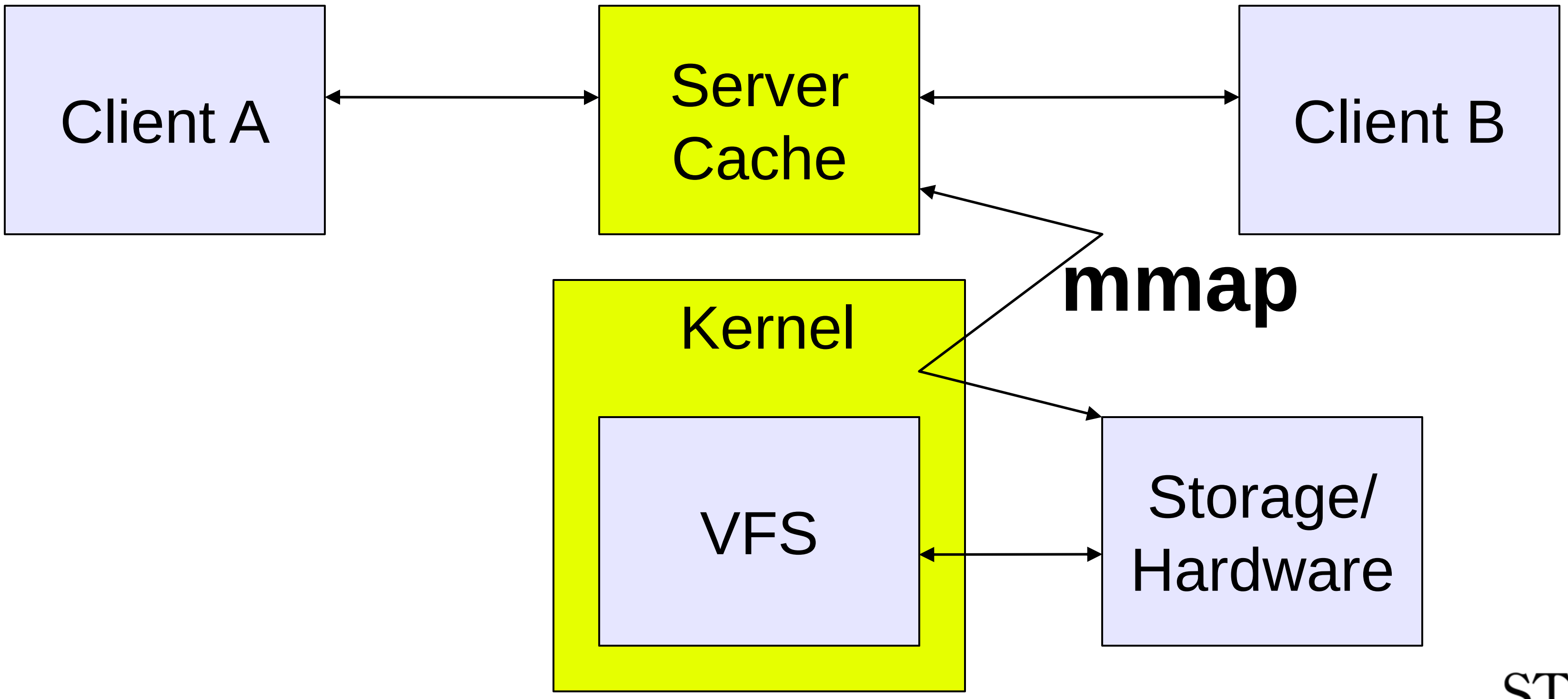
Fixing Problems (1)

- Use the same security model as kernel



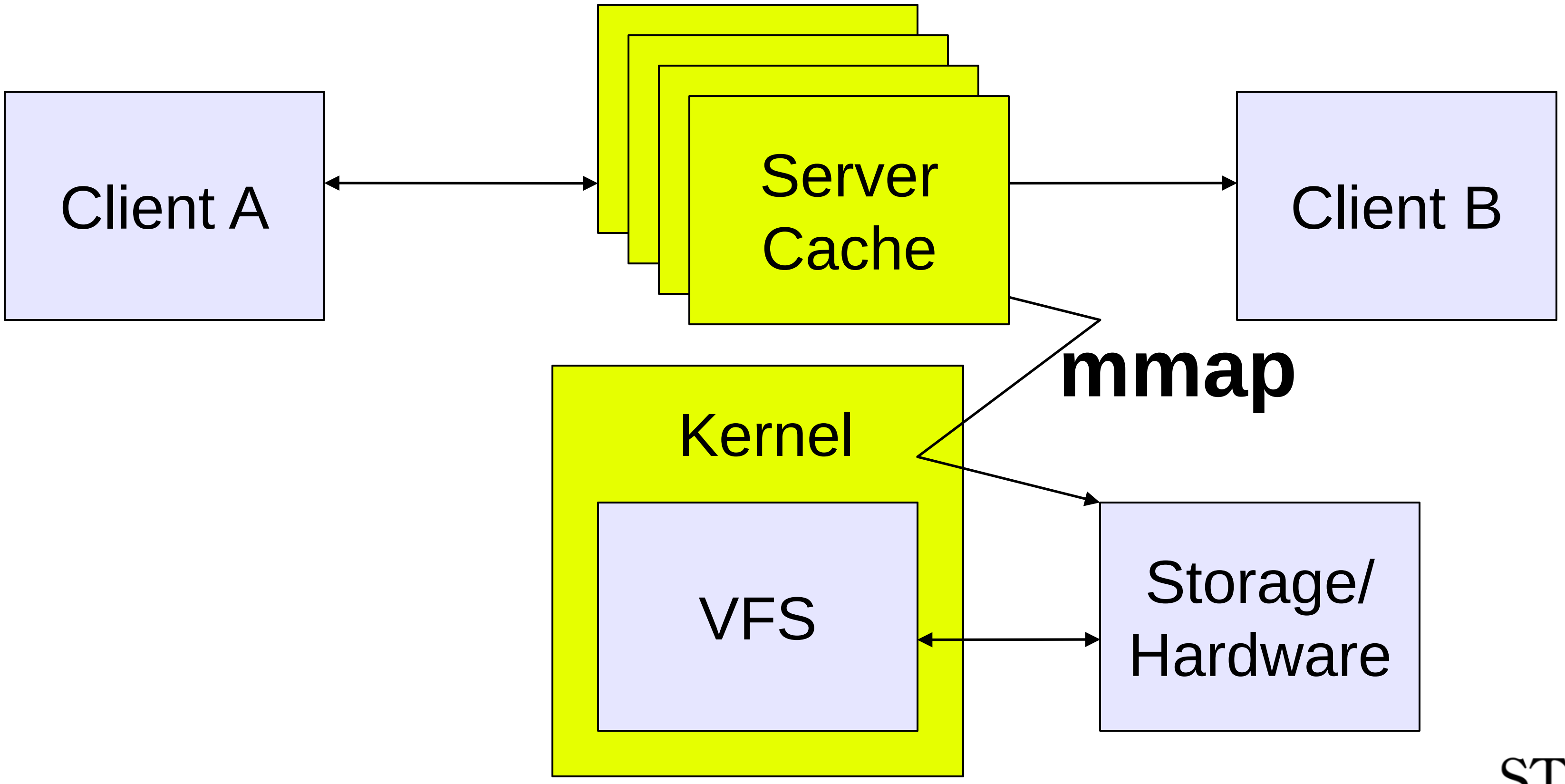
Fixing Problems (2)

- Minimize message overhead
- Extended mmap, re-implemented VFS



Getting what we wanted

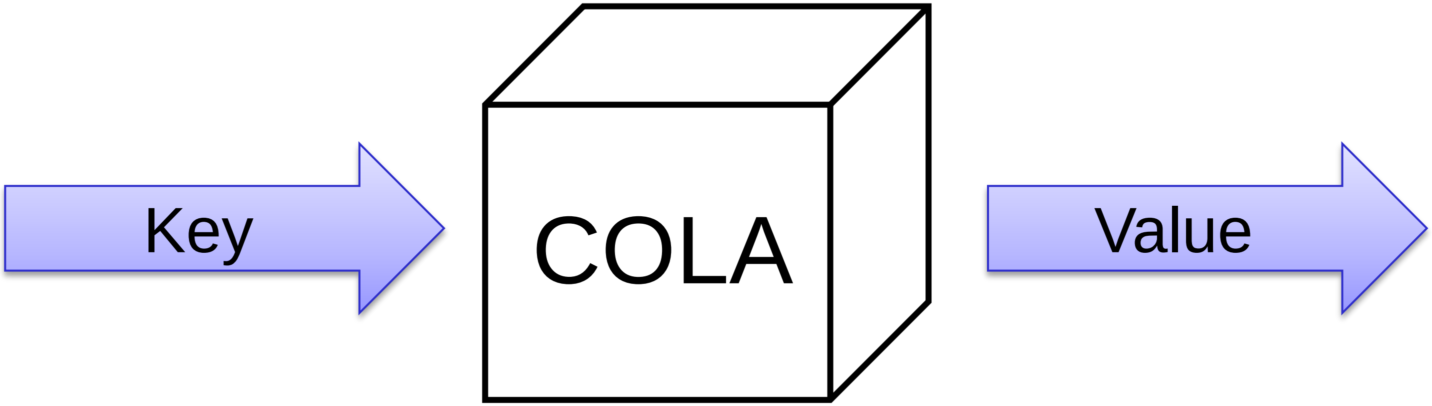
- Now you can provide system services
- Use the page cache and control I/O



Case Study: CoblFS

- CoblFS is:
  - An interesting FS that doesn’t want to use the VFS caches
  - A storage stack that will control write ordering
  - Implemented completely from the ground up to provide a fair comparison of programming techniques

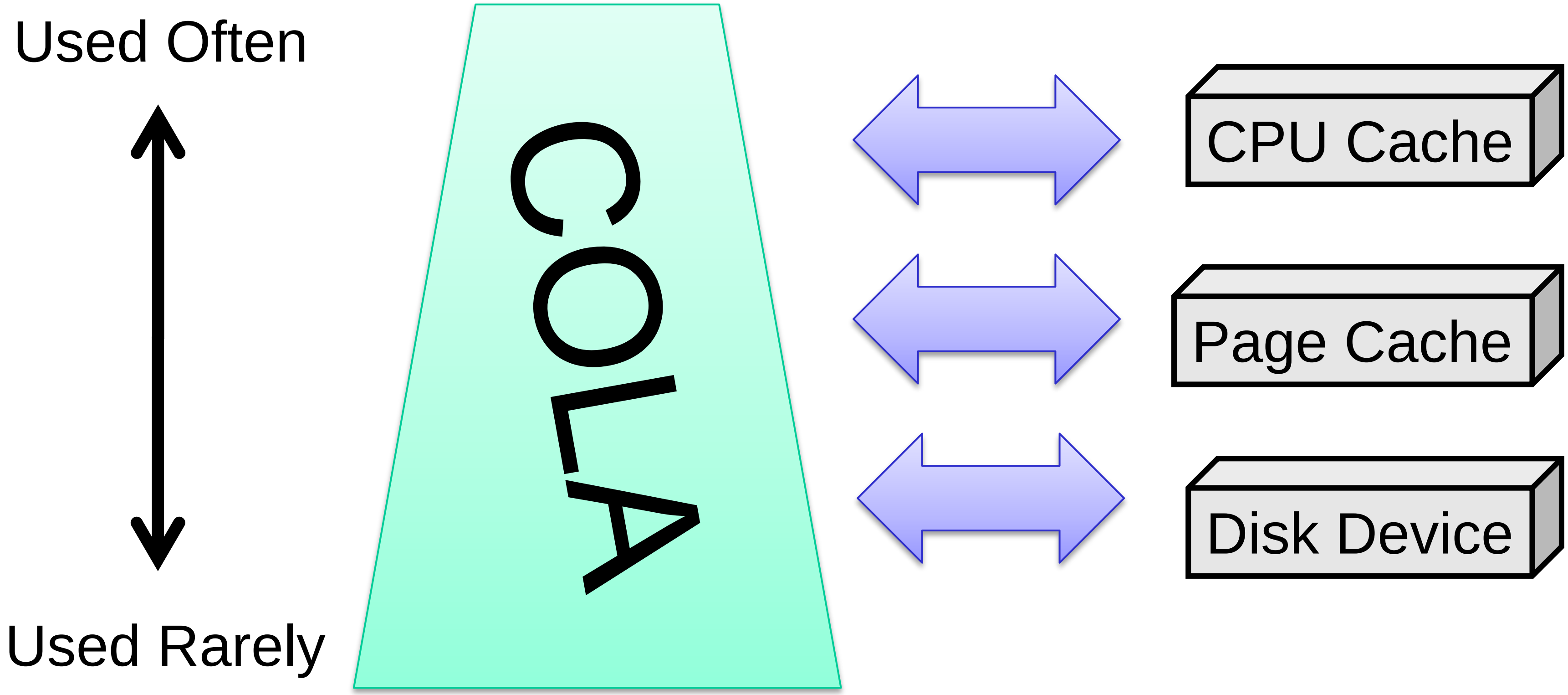
CobIFS uses the COLA



- Interesting properties
  - Very very very fast insertions/updates/dels
  - Somewhat slower lookups
  - Very simple
  - Cache oblivious…

Cache Oblivious in Practice

Large, Pinned `mmap`



Used Often

Used Rarely

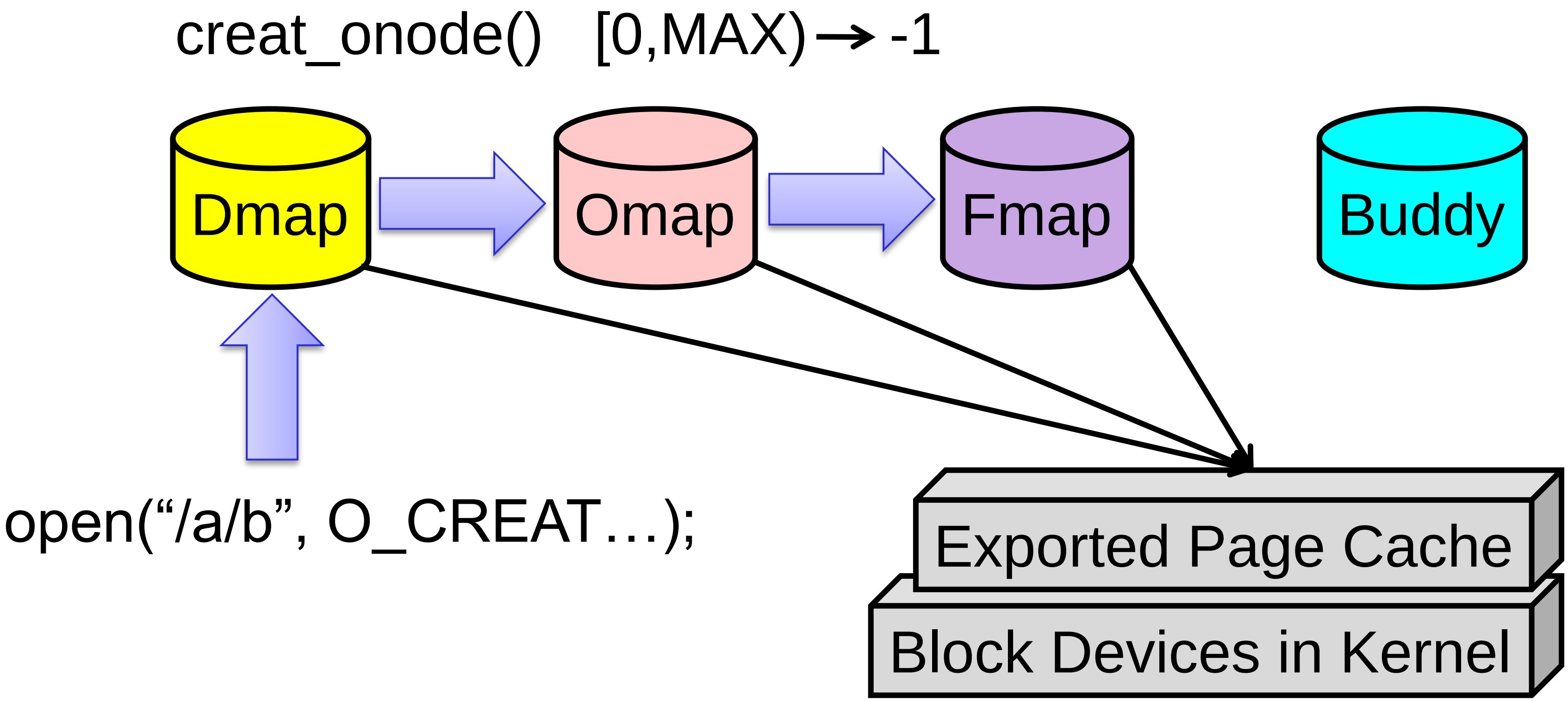
COLA

CPU Cache

Page Cache

Disk Device

CobIFS Schema: Create



creat_onode() [0,MAX) → -1

Dmap

Omap

Fmap

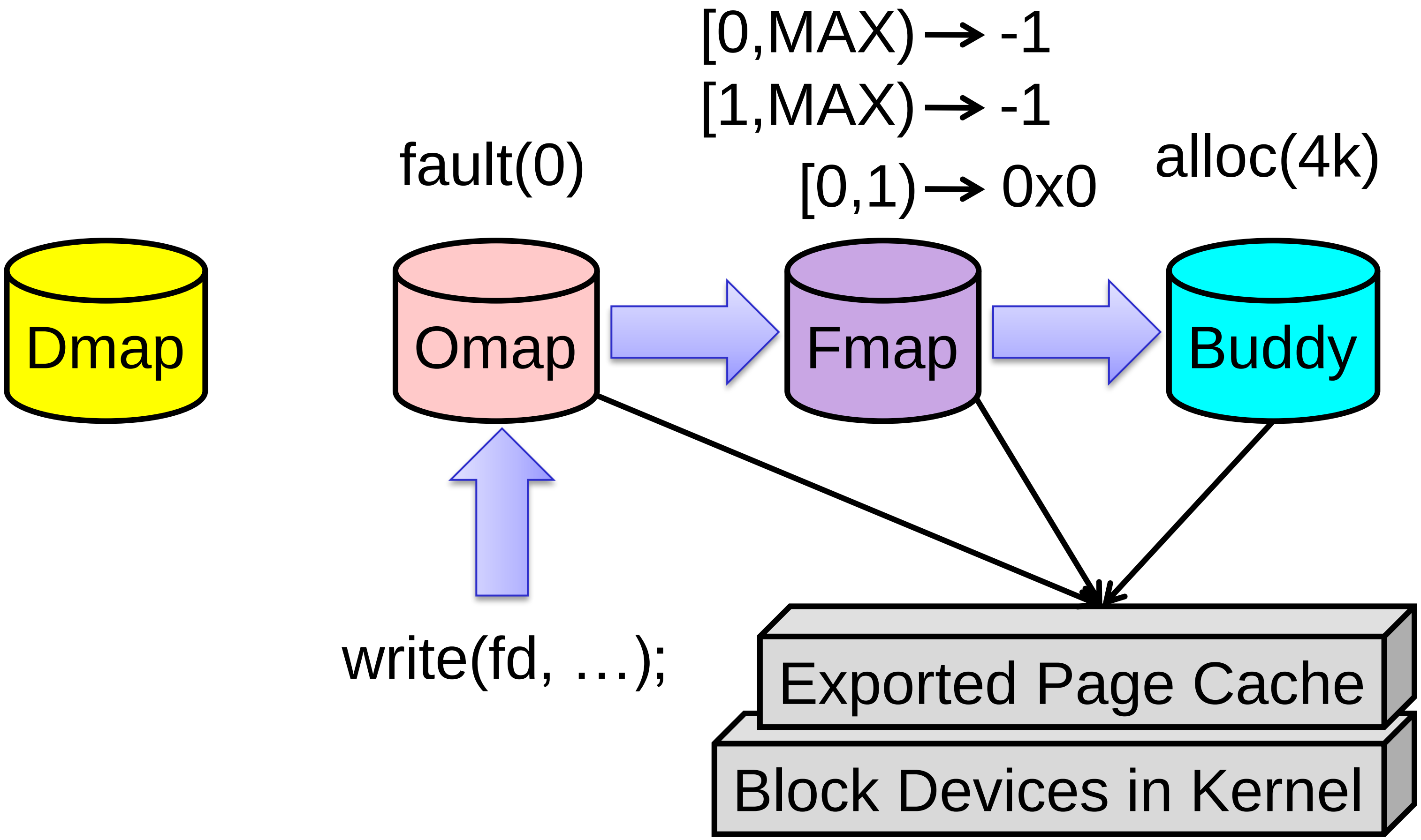
Buddy

open(“/a/b”, O_CREAT…);

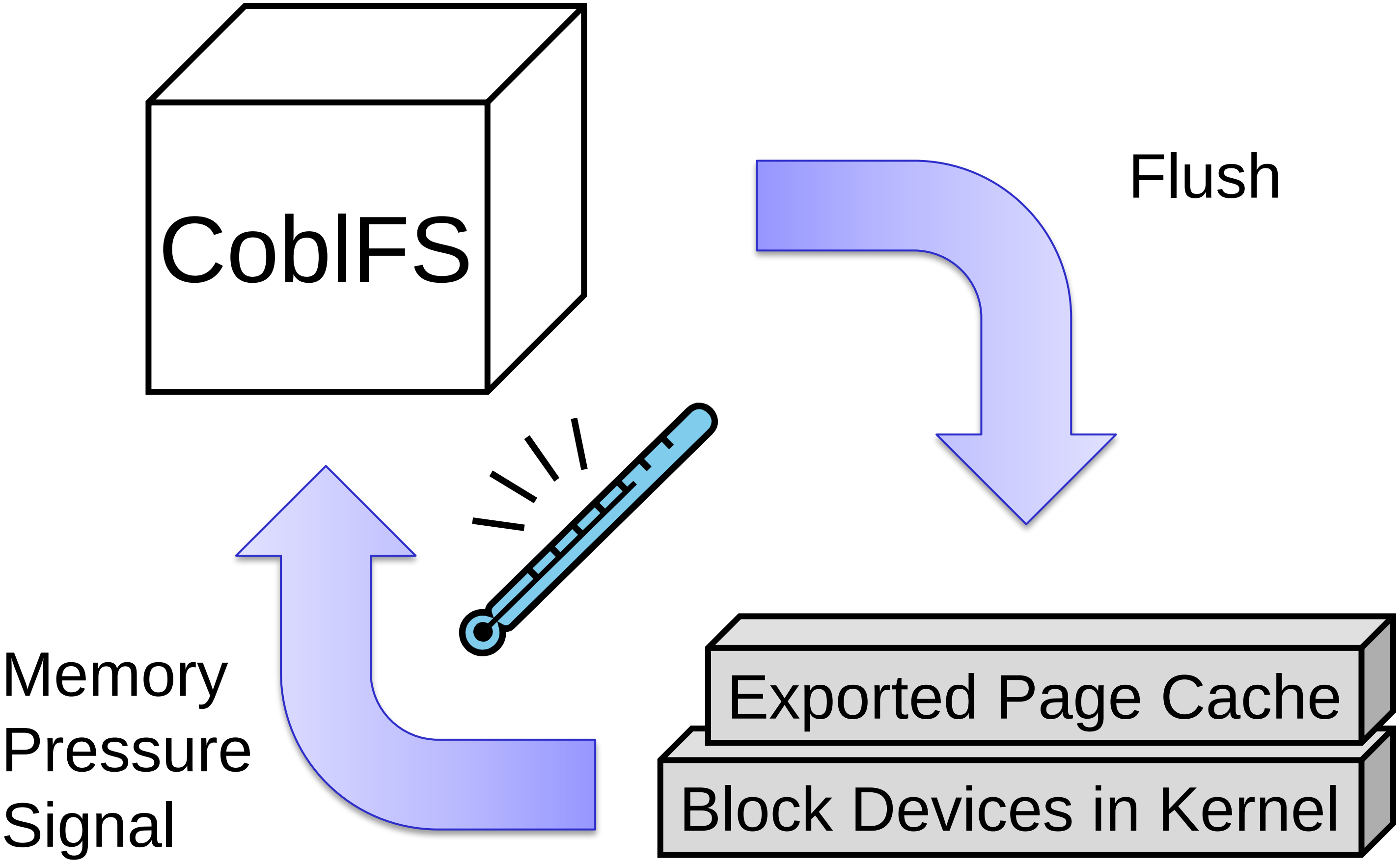
Exported Page Cache

Block Devices in Kernel

CobIFS Schema: Write (fault)



CoblFS Schema: Flush



CoblFS Schema: Flush

- 1) Take locks on all indexes
- 2) Flush all COLAs to new region
- 3) Write new COLA state to journal
- 4) Write checksum, flush journal

Evaluation

- We used FileBench
  - Had to use our lab’s distribution (Linux)
  - We used the default workloads
  - FileBench couldn’t do 100 million files
- Micro benchmark: Hotset
- System benchmarks
  - Webserver
  - Fileserver
  - Videoserver (read paper)

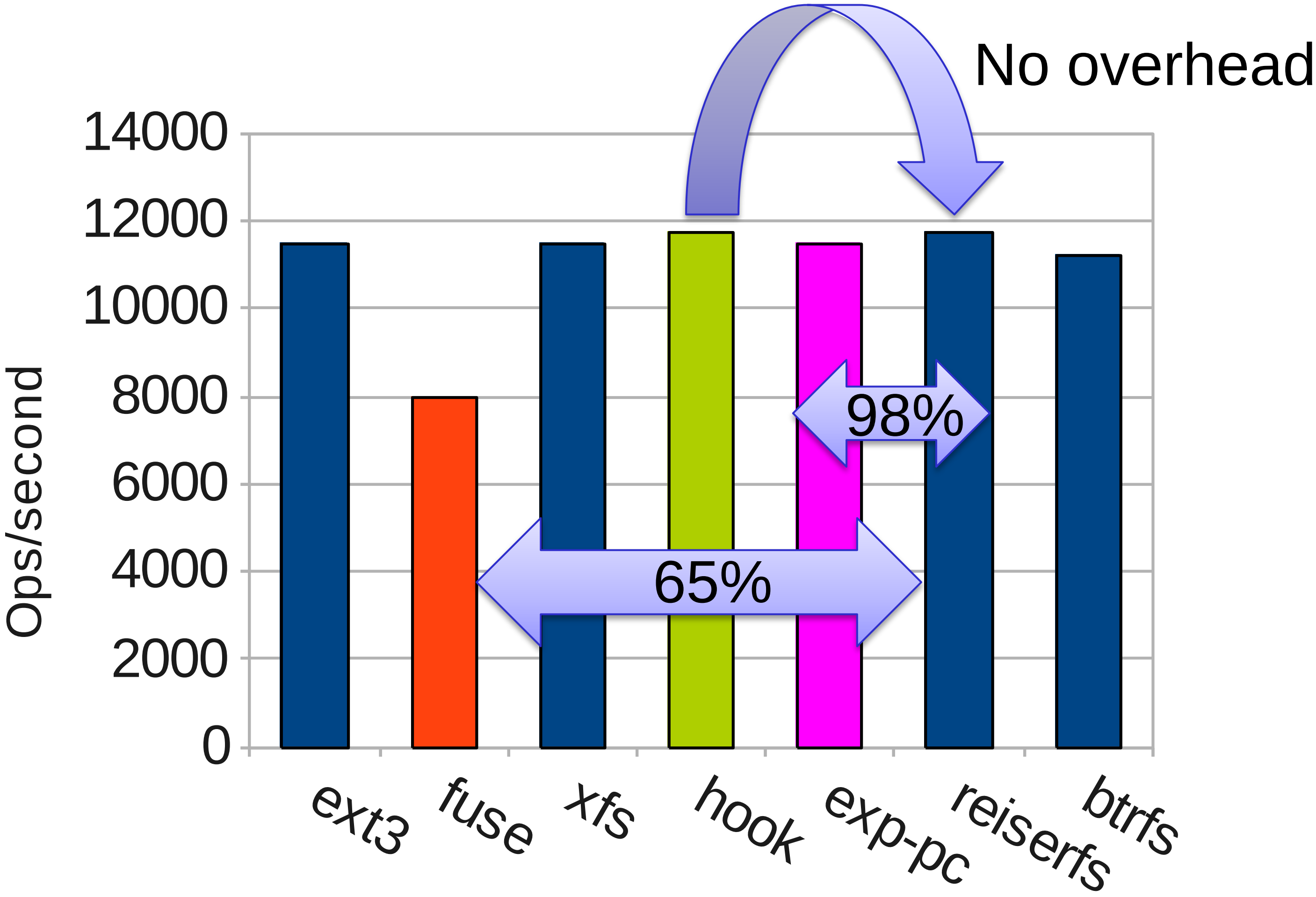
In/Out-of-Cache

- We found FS performance same when:
  - You make sure everything is in RAM
  - Or, make sure everything is out of RAM
- Hotset by definition is always in-cache
- We discuss other system benchmarks out-of-cache

Hardware

- We had 6 identical machines
  - 2.8GHz Xeon CPU
  - 1GiB of RAM
  - Maxtor DiamondMax 10 7,200 RPM SATA
  - Centos 5.3x86-64
  - Identical newly formatted 30GiB partitions

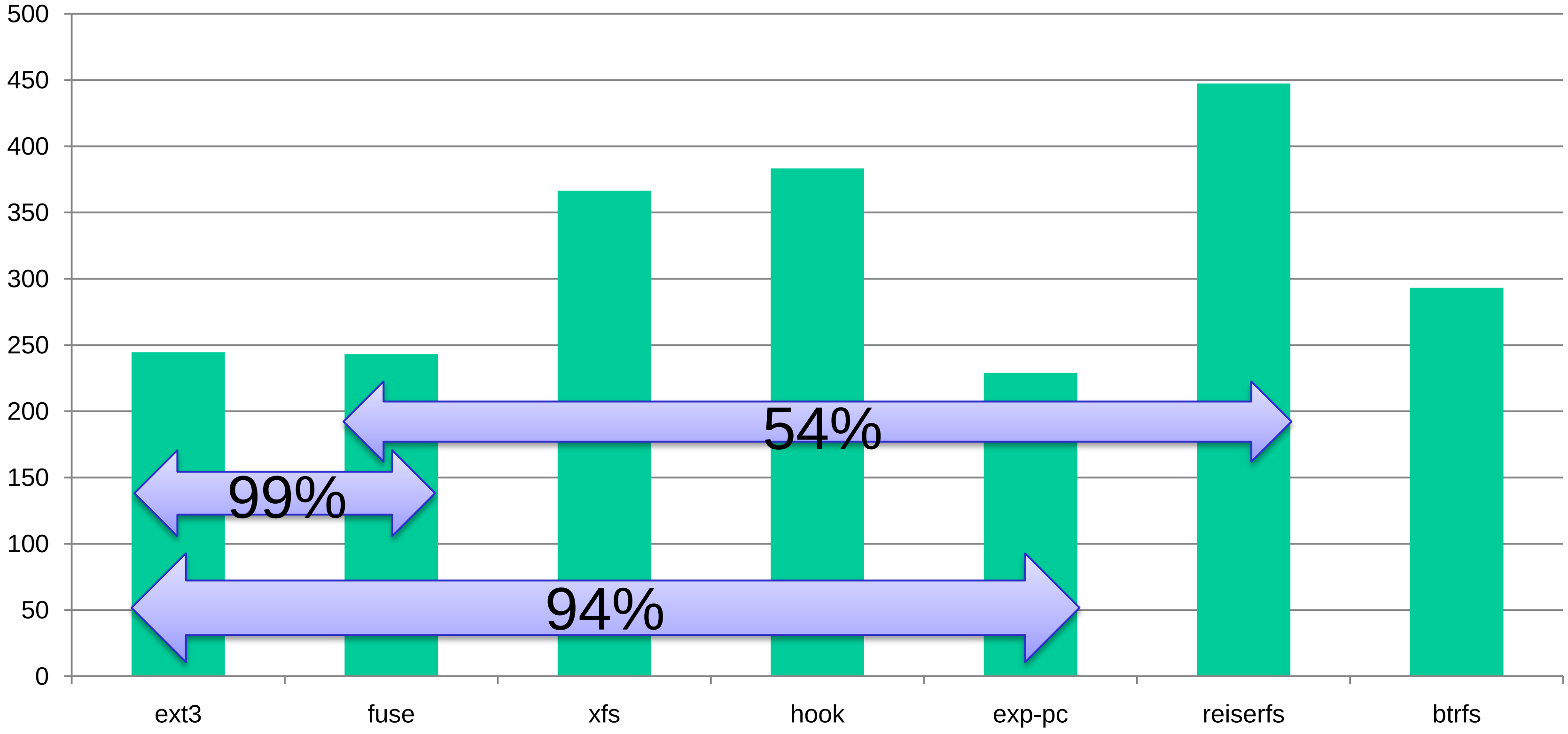
Hotset (IC)


No overhead

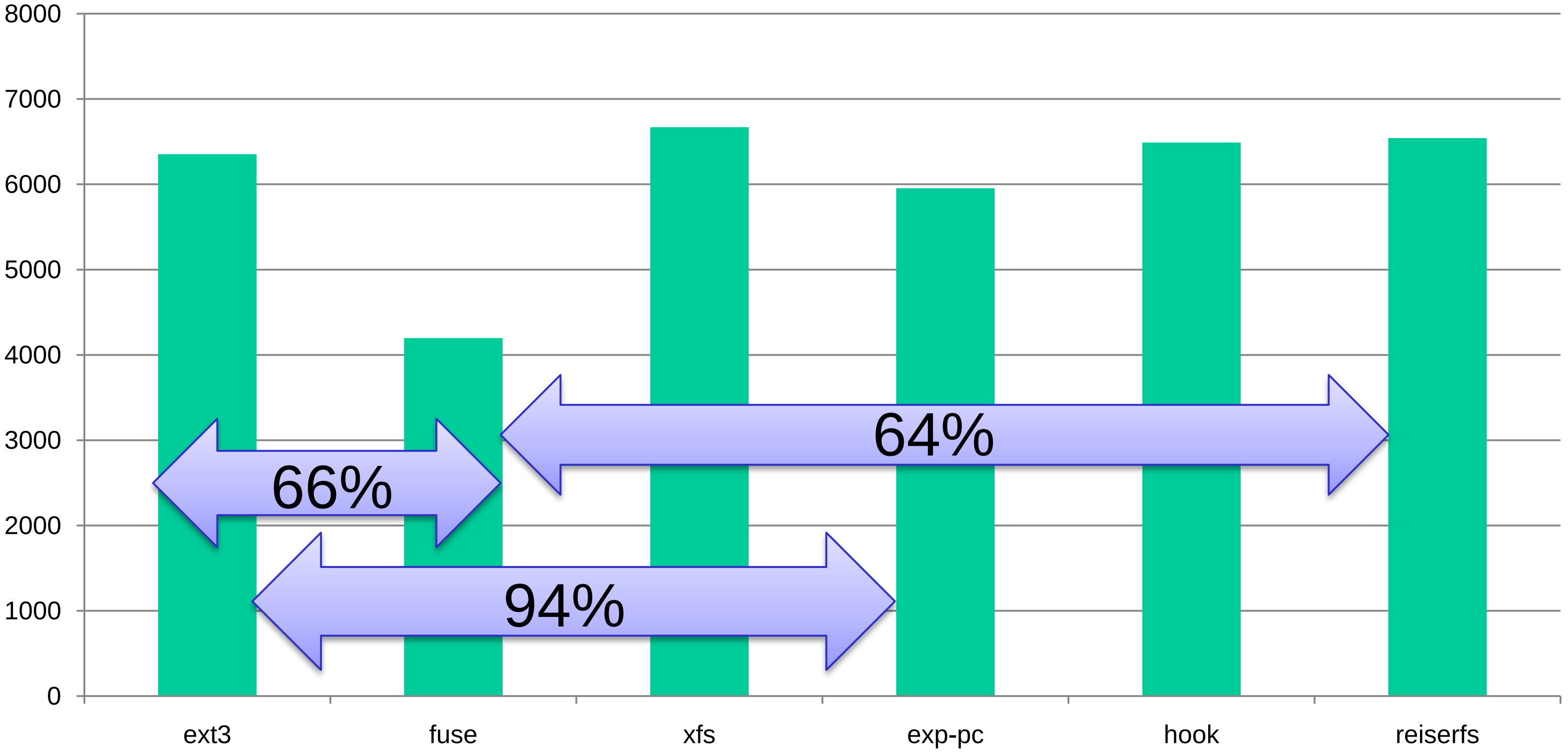
98%

65%

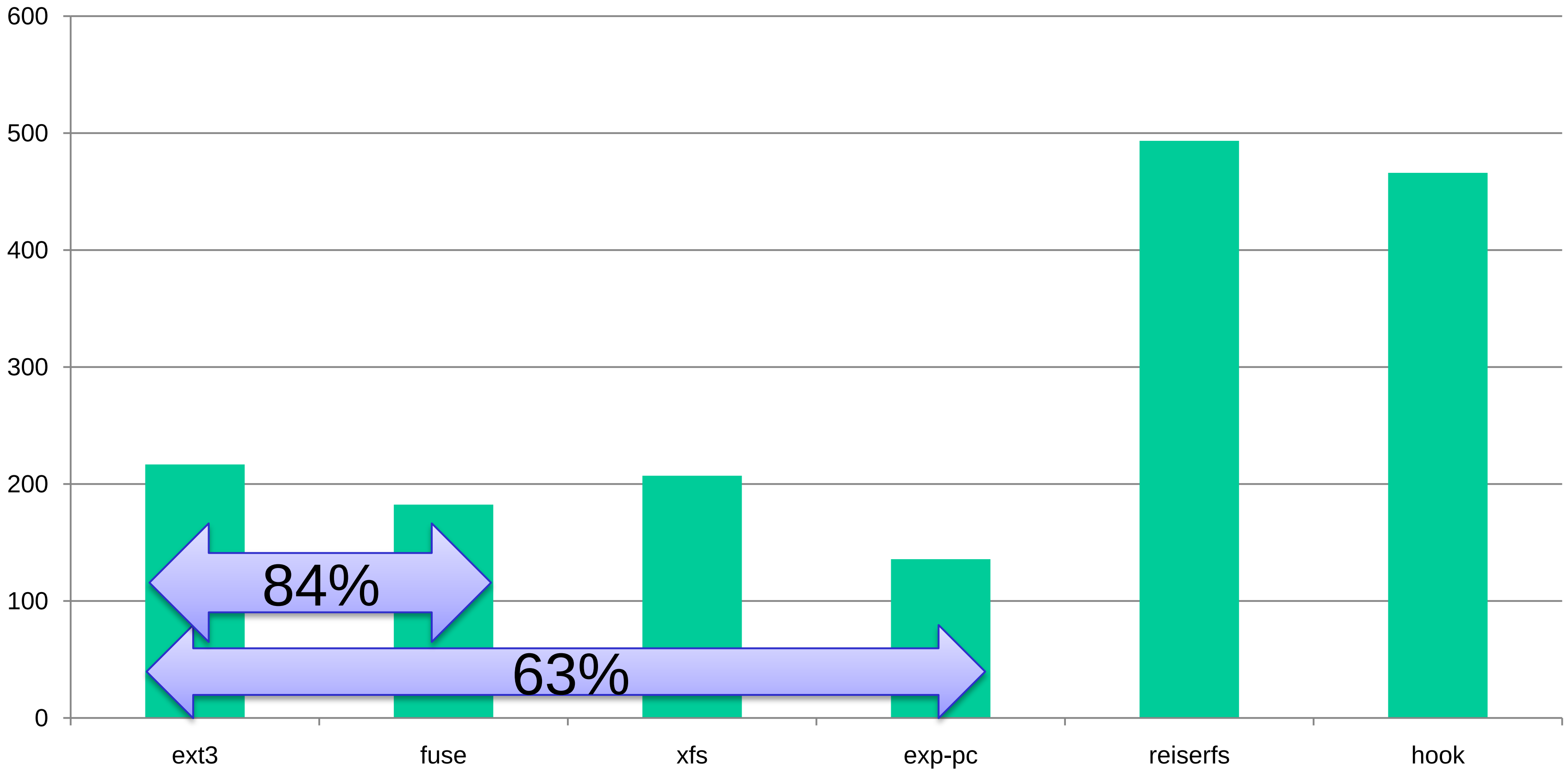
Webserver (OOC)



File Server (IC)



File Server (OOC)

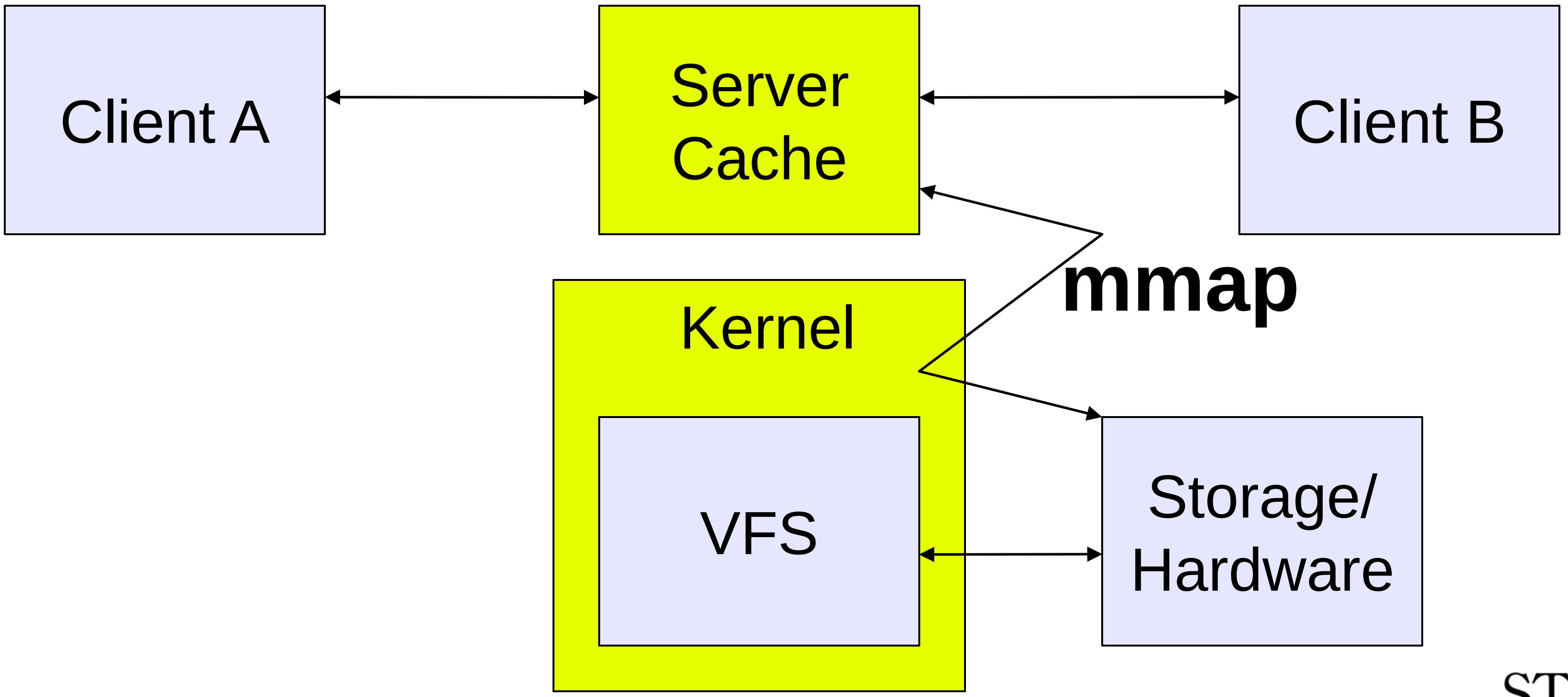


Conclusions

- One size does not fit all
- User-level can be kernel-fast
- Our approach is practical and scalable
- `mmap` needs some work

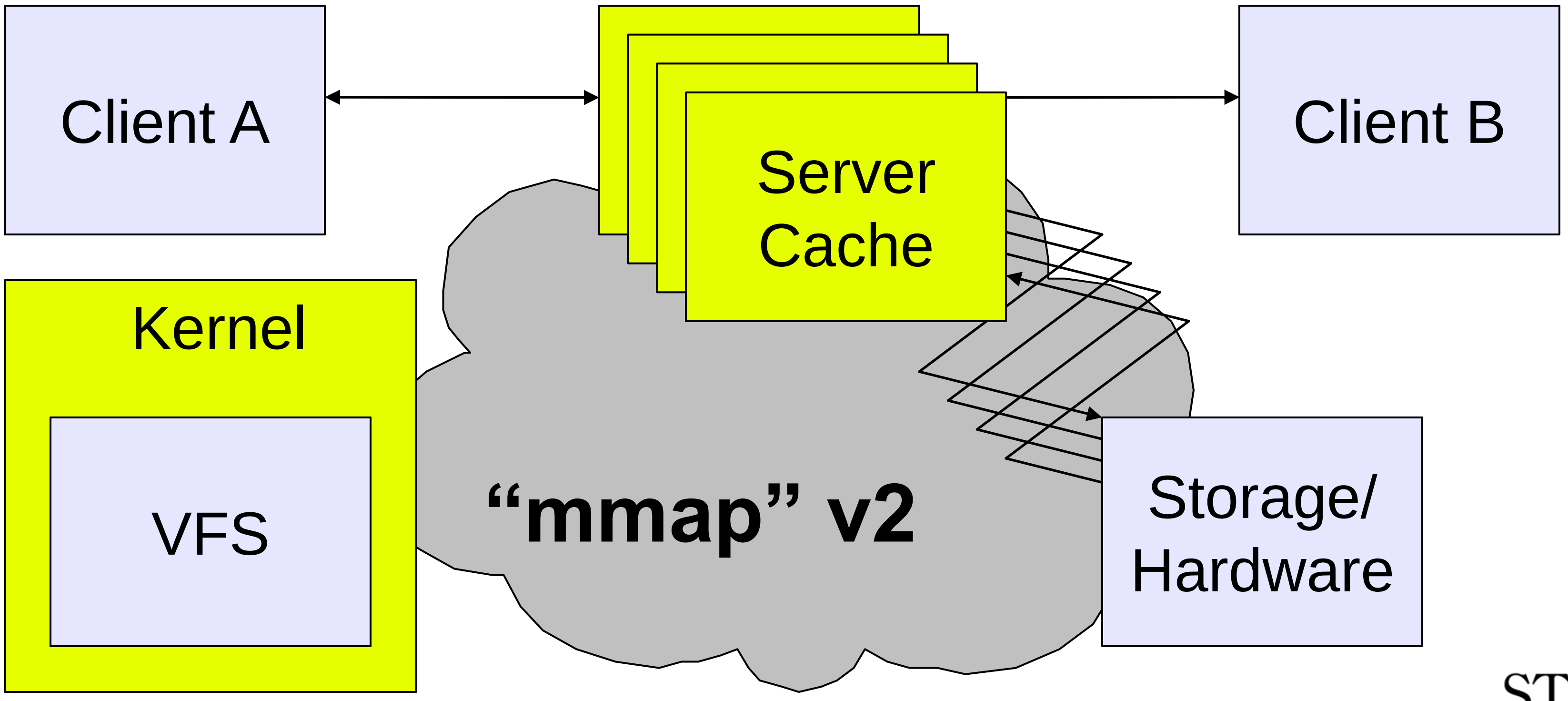
What next?

Current Architecture



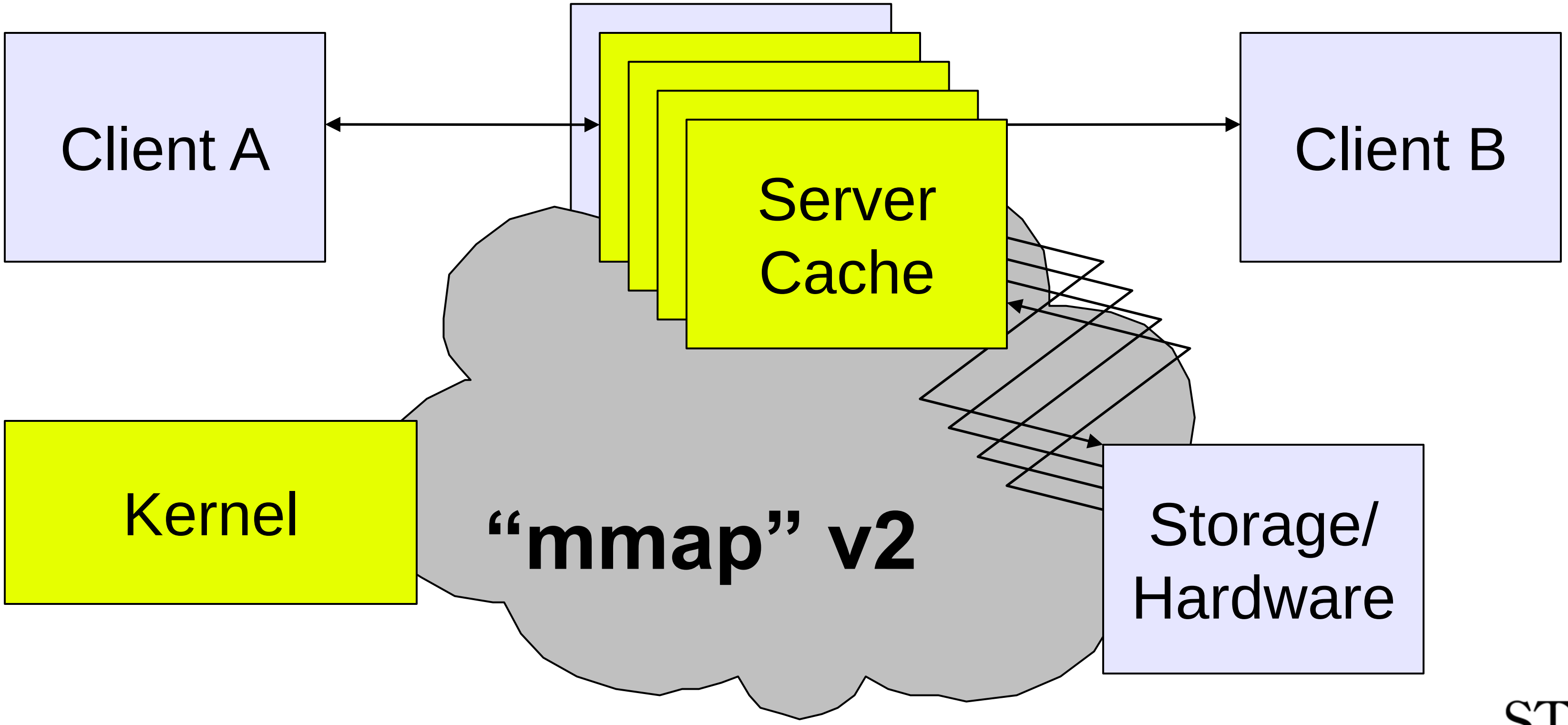
Where we want to go (1)

- Expose mmap “v2”



Where we want to go (2)

- Port and remove in-kernel VFS



05/06/2010

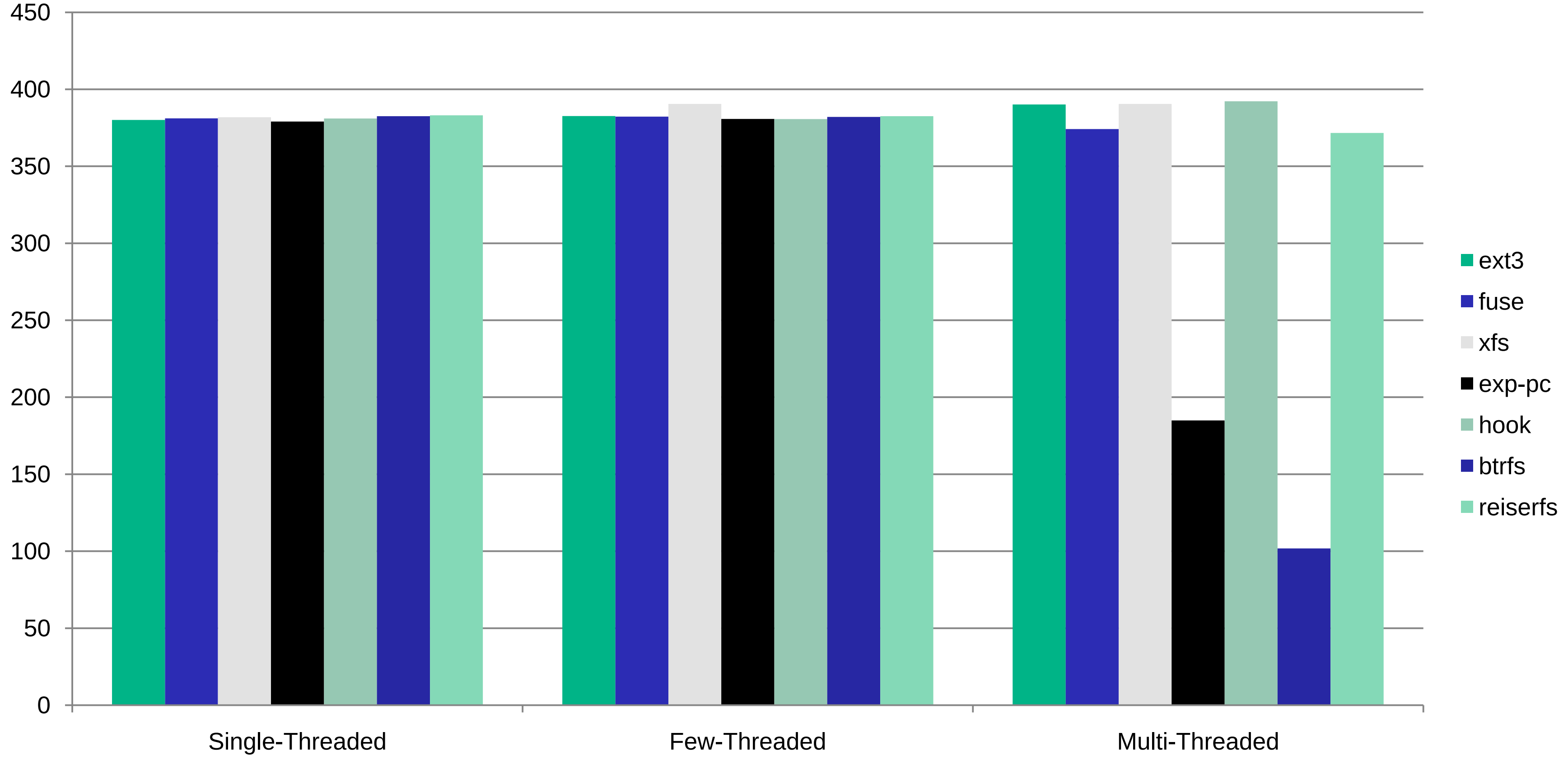
Reconfigurable VFS

- One size does not fit all
- File Systems won't go away
  - Refer to locally stored/controlled large-ish objects
  - Nothing else is sacred
- Examples
  - Naming
  - Distributing
  - Modularity

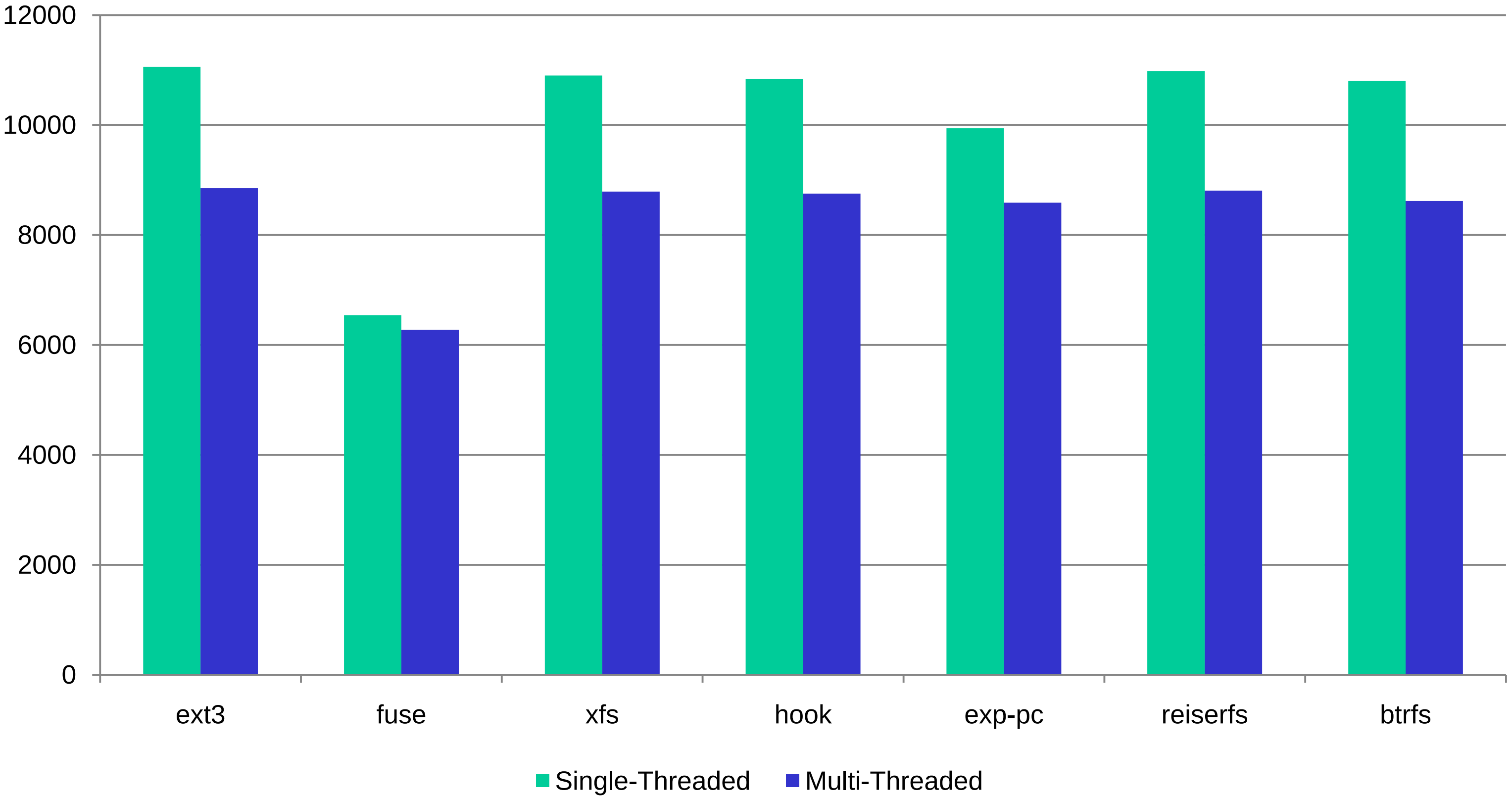
Q&A

- Richard P. Spillane
- necro351@gmail.com
- ezk@cs.sunysb.edu
- www.fsl.cs.sunysb.edu/~rick
- www.fsl.cs.sunysb.edu/
- We've got git/cvs/svn repos of our work, and want to find collaborators to share with

Video Server (OOC)



Webserver (IC)


ext3
fuse
xfs
hook
exp-pc
reiserfs
btrfs
Single-Threaded
Multi-Threaded