VM and I/O

IO-Lite: A Unified I/O Buffering and Caching System

Vivek S. Pai, Peter Druschel, Willy Zwaenepoel

Software Prefetching and Caching for TLBs

Kavita Bala, M. Frans Kaashoek, William E. Weihl

General themes

- CPU, network bandwidth increasing rapidly
- Main memory, IPC unable to keep up
  - trend towards microkernels increase number of IPC transactions

General themes

- CPU, network bandwidth increasing rapidly
- Main memory, IPC unable to keep up
  - trend towards microkernels increase number of IPC transactions

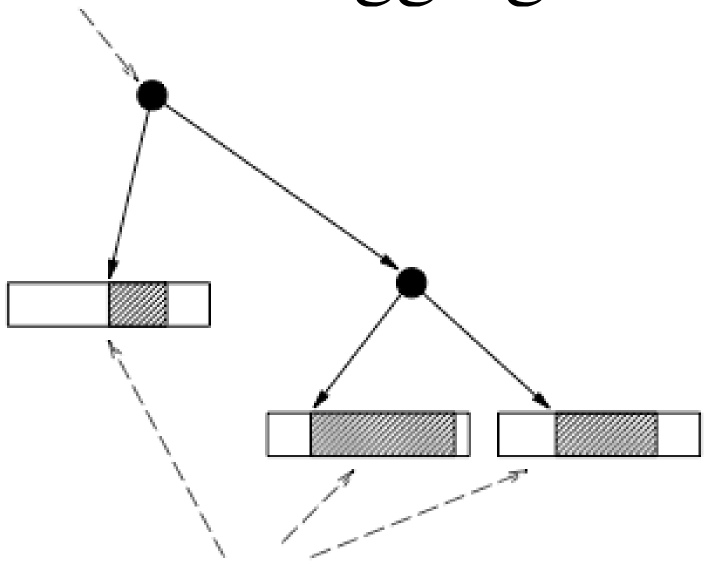
One remedy is to increase speed/bandwidth of IPC data (data moving between processes)

fbufs

- Attempts to increase bandwidth within network subsystem
- In a nutshell: provides immutable buffers shared among processes of subsystem
- Implemented using shared memory and page remapping in a specialized OS: the *x*-kernel

fbuf, details

- Incoming “packet data units” passed to higher protocols in fbufs
- PDUs are assembled into “application data units” by use of an aggregation ADT



fbufs, details

- fbuf interface does not support writes after producer fills buffer (PDU)
  - fbufs can be reused after consumer is finished; leads to *sequential* use of fbufs
  - applications shouldn’t have to modify data anyway

fbufs, details

- fbuf interface does not support writes after producer fills buffer (PDU)
  - fbufs can be reused after consumer is finished; leads to *sequential* use of fbufs
  - applications shouldn't have to modify data anyway
  - LIMITATION, especially in a more general system

Enter IO-Lite

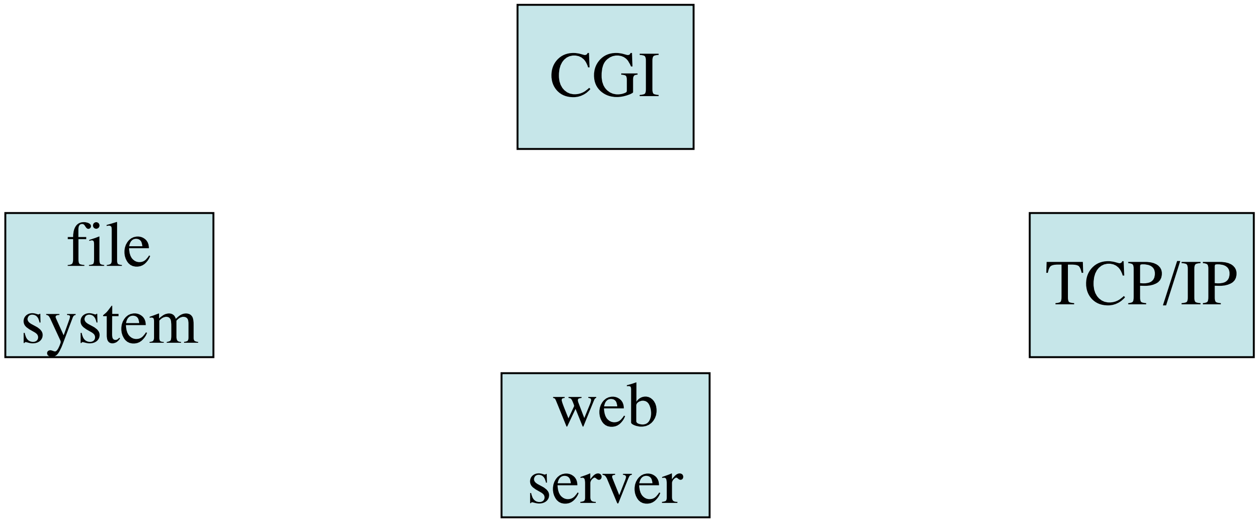
- Take fbufs, but make them
  - more general, accessible to the filesystem in addition to the network subsystem
  - more versatile, usable on standard OSes (not just *x*-kernel)
- Solves a more general problem: rapidly increasing CPUs (not just network bandwidth)

Before comparing them to fbufs...

- Problems in the “old way” of doing things
  - redundant data copying
  - redundant copies of data lying around
  - no special optimizations between subsystems

IO-Lite at a high level

- IO-Lite must provide system-wide buffers to prevent multiple copies
  - UNIX allocates filesystem buffer cache from different pool of kernel memory than, say, network buffers and application-level buffers

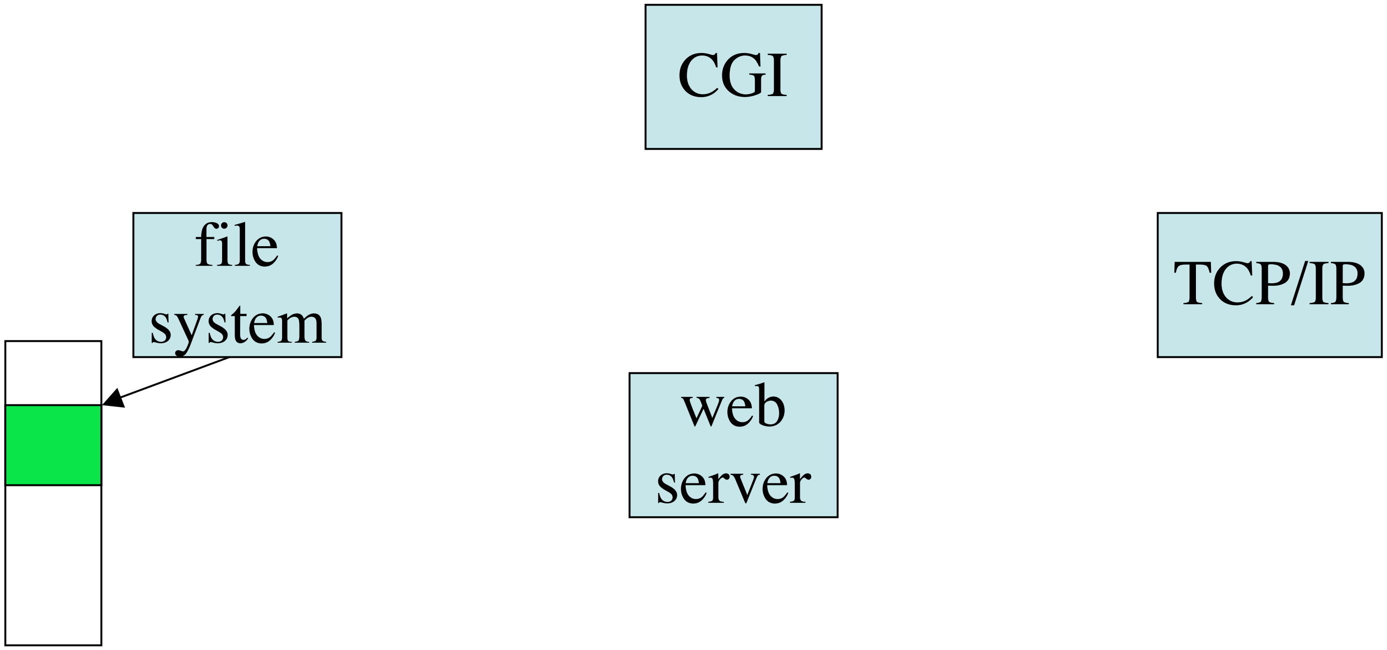


CGI

file system

TCP/IP

web server

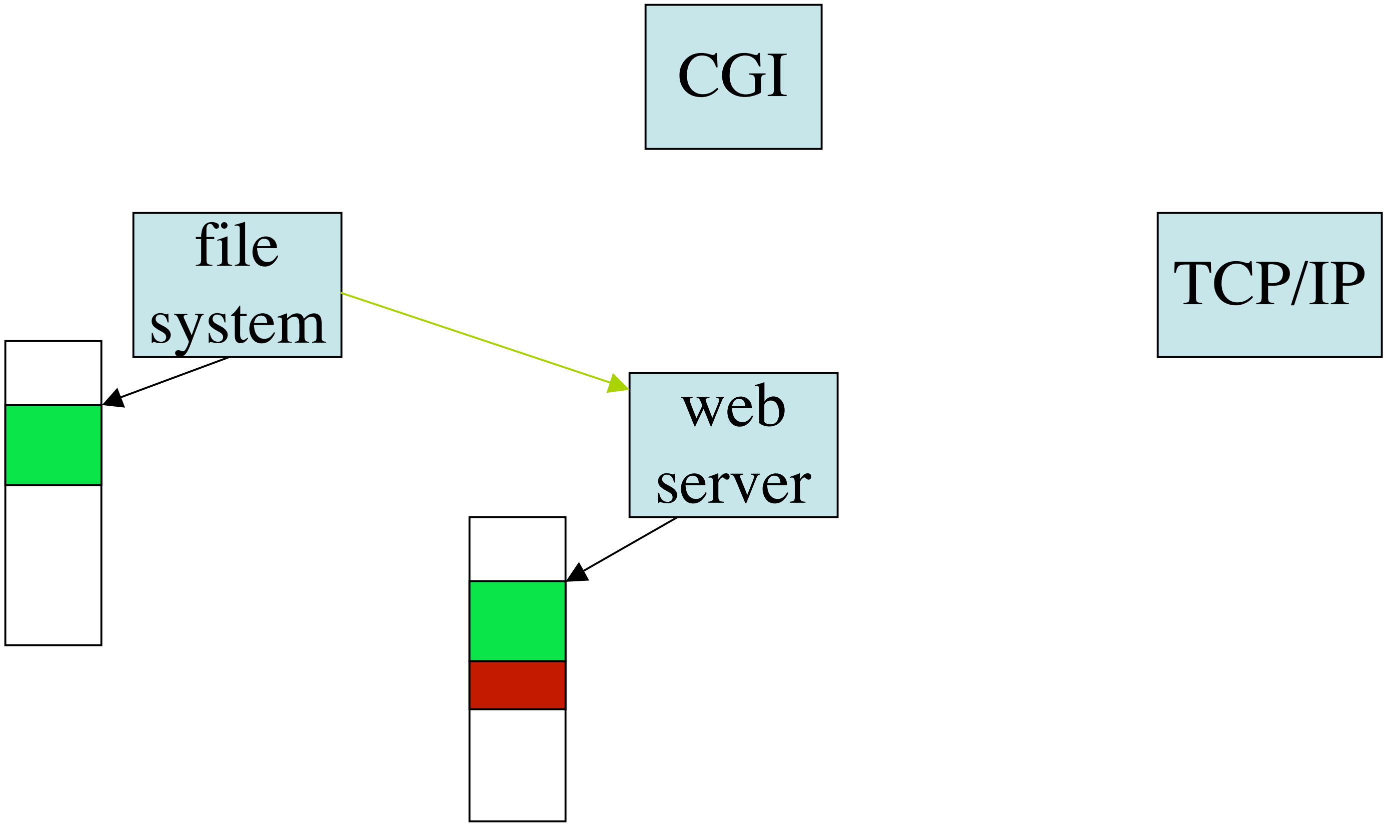


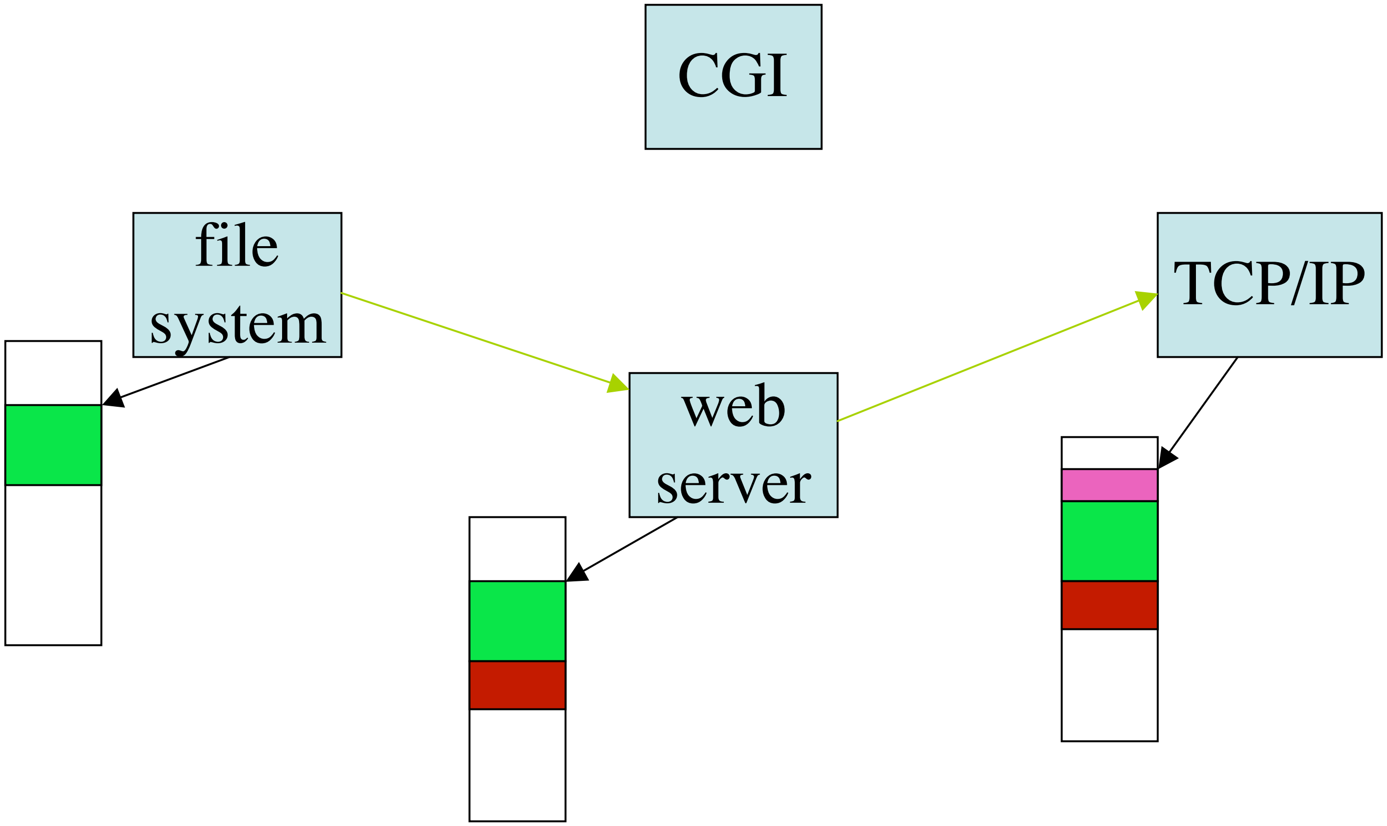
CGI

file system

TCP/IP

web server


CGI
file system
TCP/IP
web server

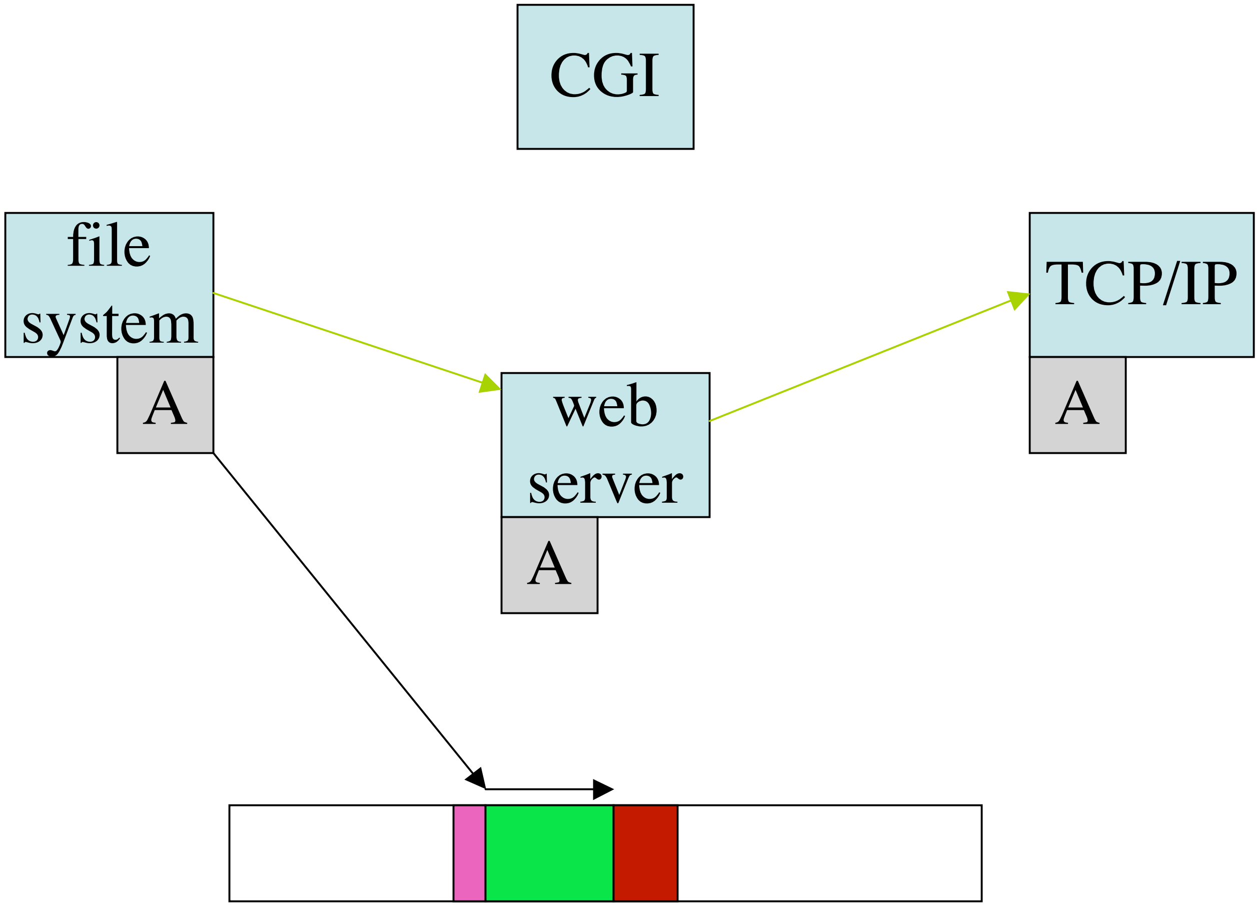


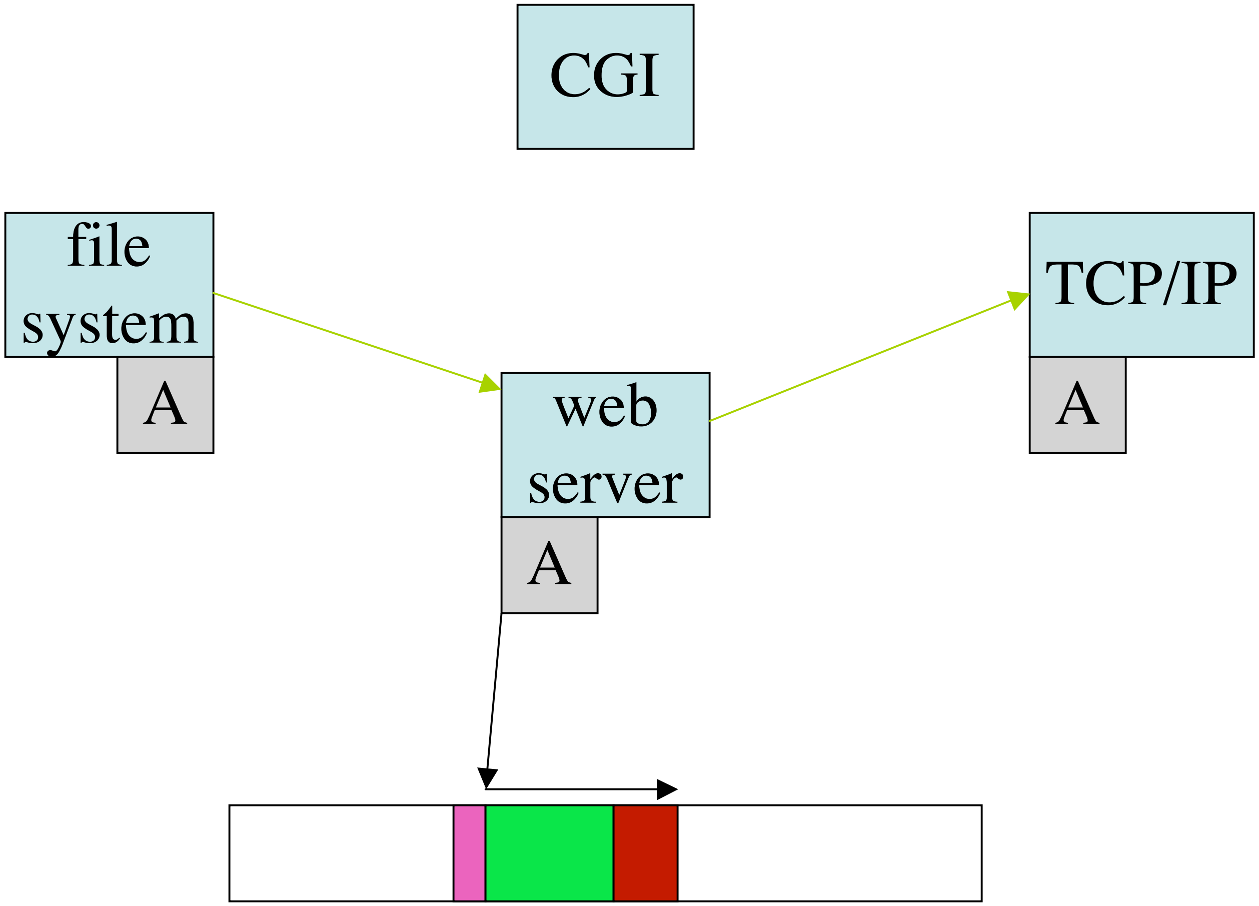
CGI

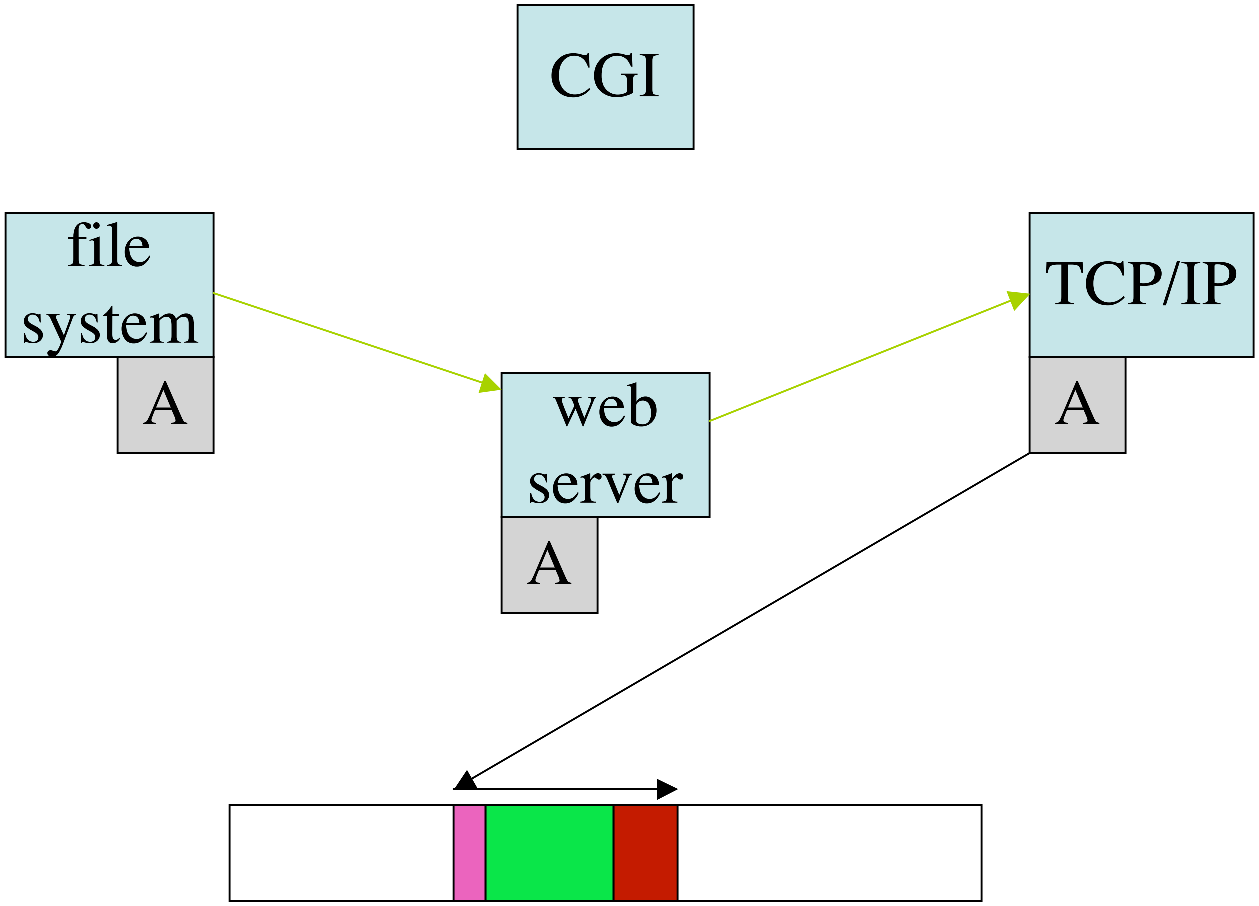
file system

web server

TCP/IP


CGI
file system
A
web server
A
TCP/IP
A


CGI
file system
A
web server
A
TCP/IP
A

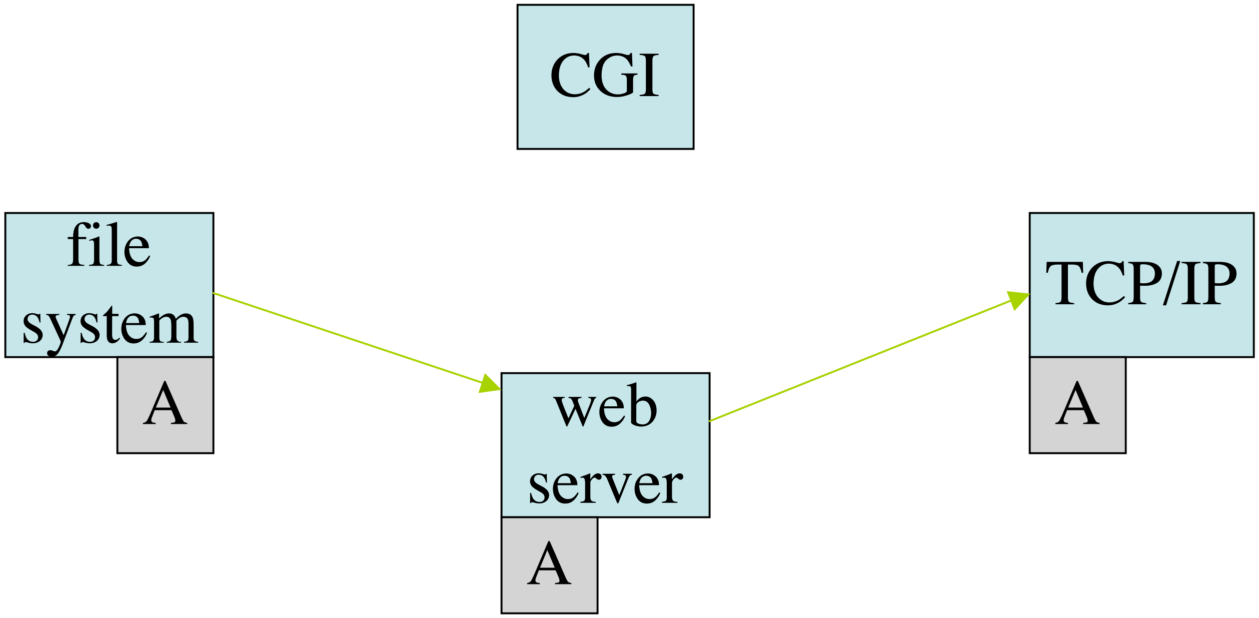

CGI
file system
A
web server
A
TCP/IP
A

Access Control Lists

- Processes must be granted permission to view buffers
  - each buffer pool has an ACL for this purpose
  - for each buffer space, list of processes granted permission to access it

Consequence of ACLs

- Producer must know data path to consumer
  - gets slightly tricky with incoming network packets
  - must use *early demultiplexing* (mentioned as a common enough technique)



CGI

file system

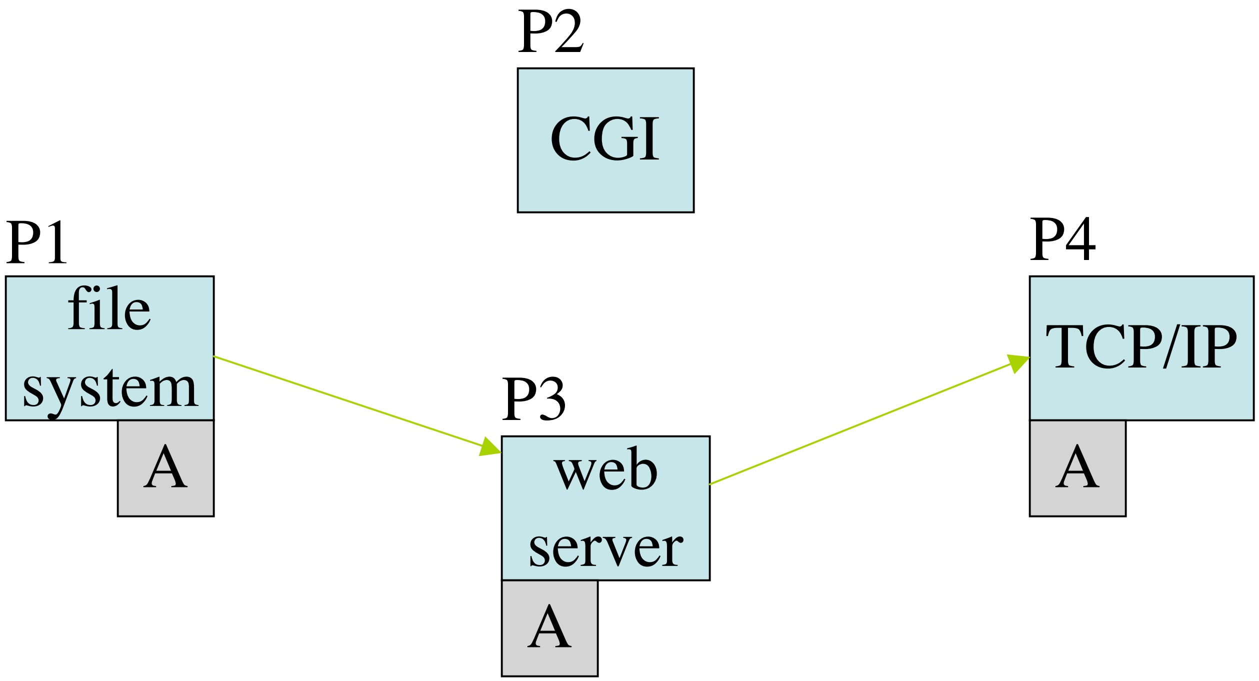
A

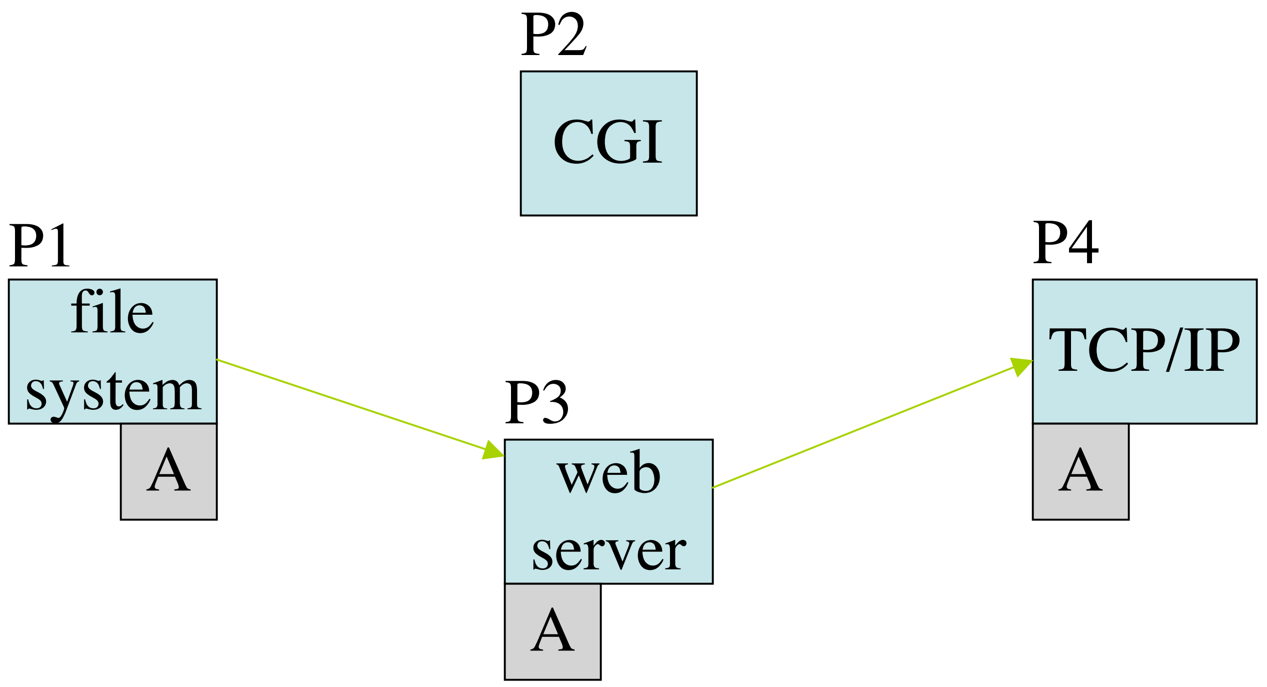
web server

A

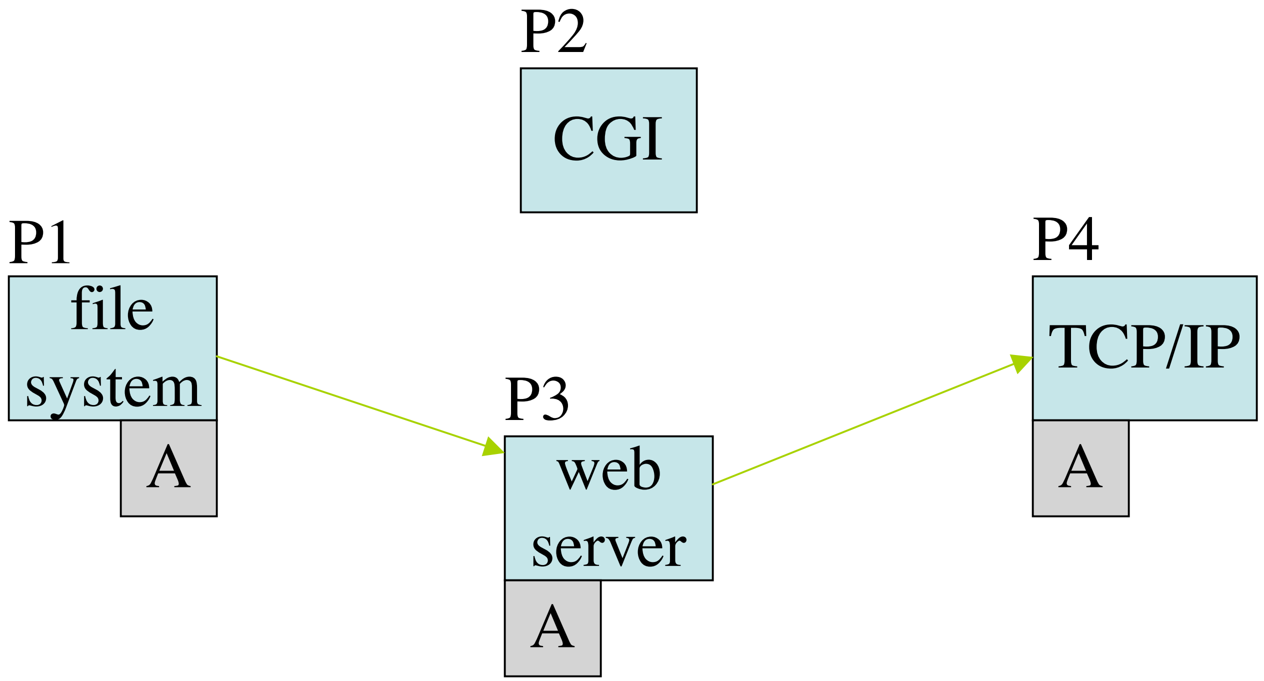
TCP/IP

A


P2
CGI
P1
file system
A
P3
web server
A
P4
TCP/IP
A



Buffers:	1	2	3
ACLs:	P1, P2	P1, P3, P4	P4



Buffers:	1	2	3
ACLs:	P1, P2	P1, P3, P4	P4

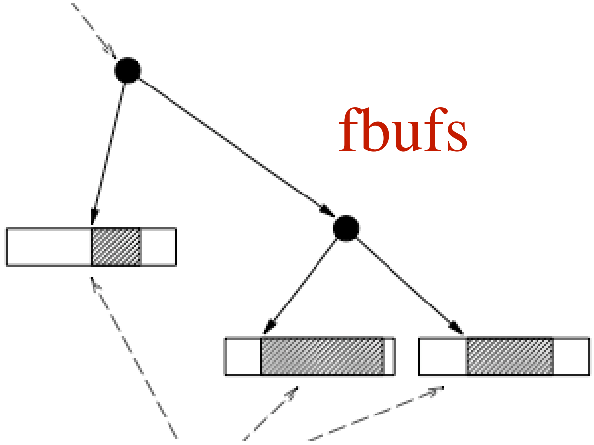
Pipelining

- Abstractly represents good modularity
- Conceptually data *moves* through pipeline from producer to consumer
- IO-Lite comes close to implementing this in practice
  - when the path is known ahead of time, context switches are the biggest overheads in pipeline

immutable --> mutable

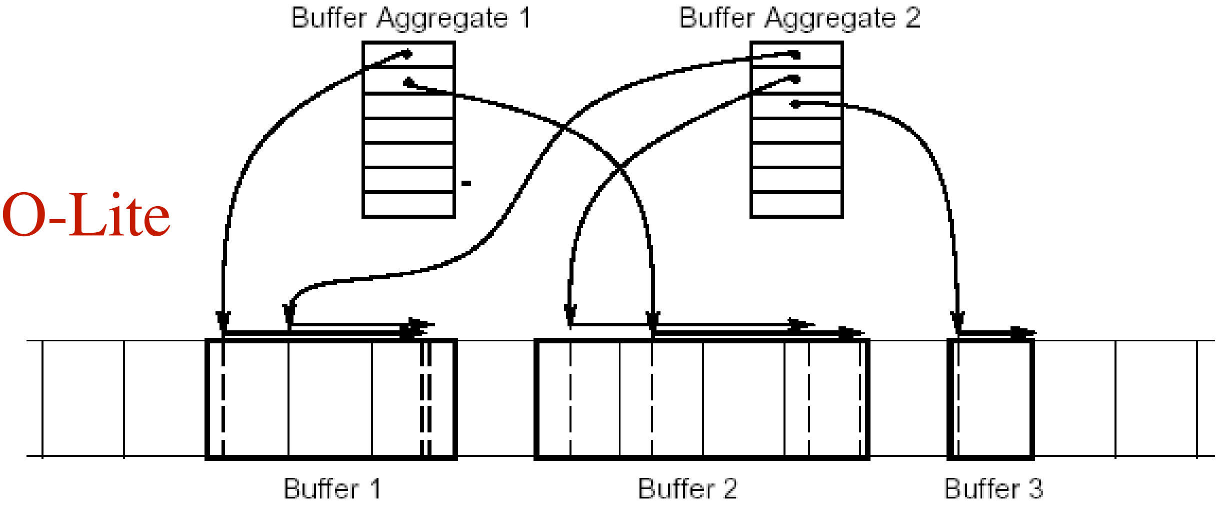
- Data in an OS must be manipulated in various ways
  - network protocols (same as fbufs)
  - modifying cached files (i. e., to send to various clients via a network/writing checksums)
- IO-Lite must support *concurrent* buffer use among sharing processes

immutable --> mutable

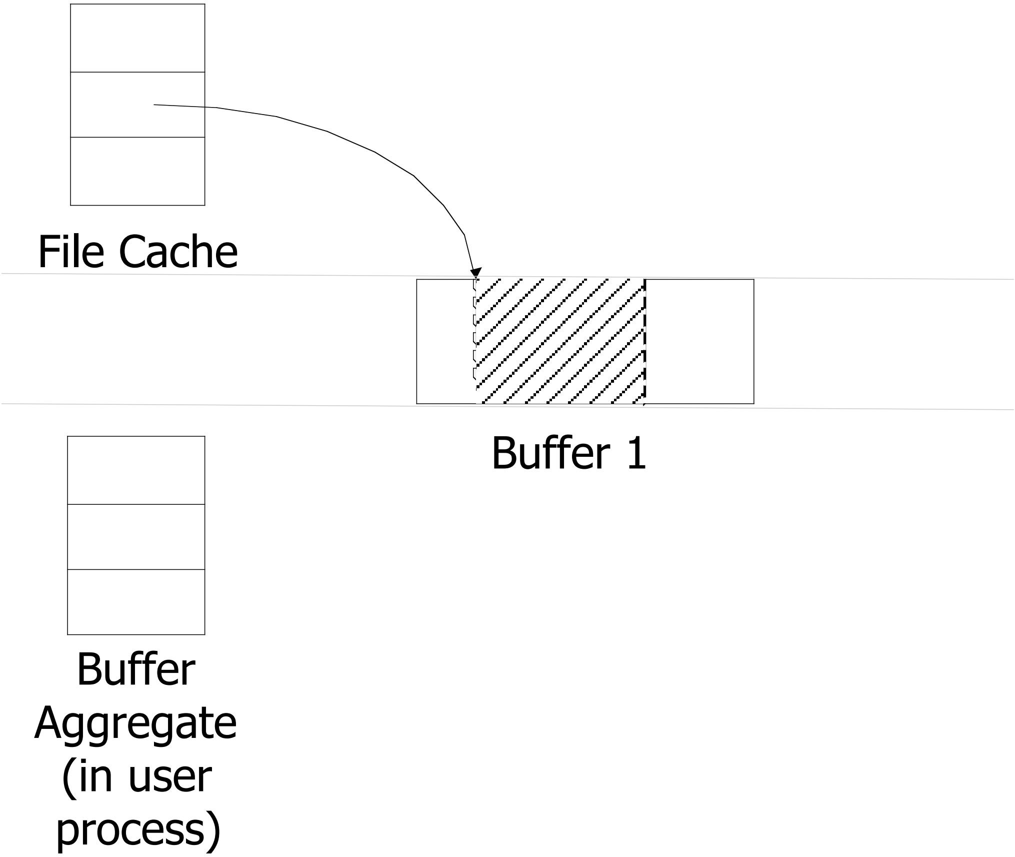


fbufs

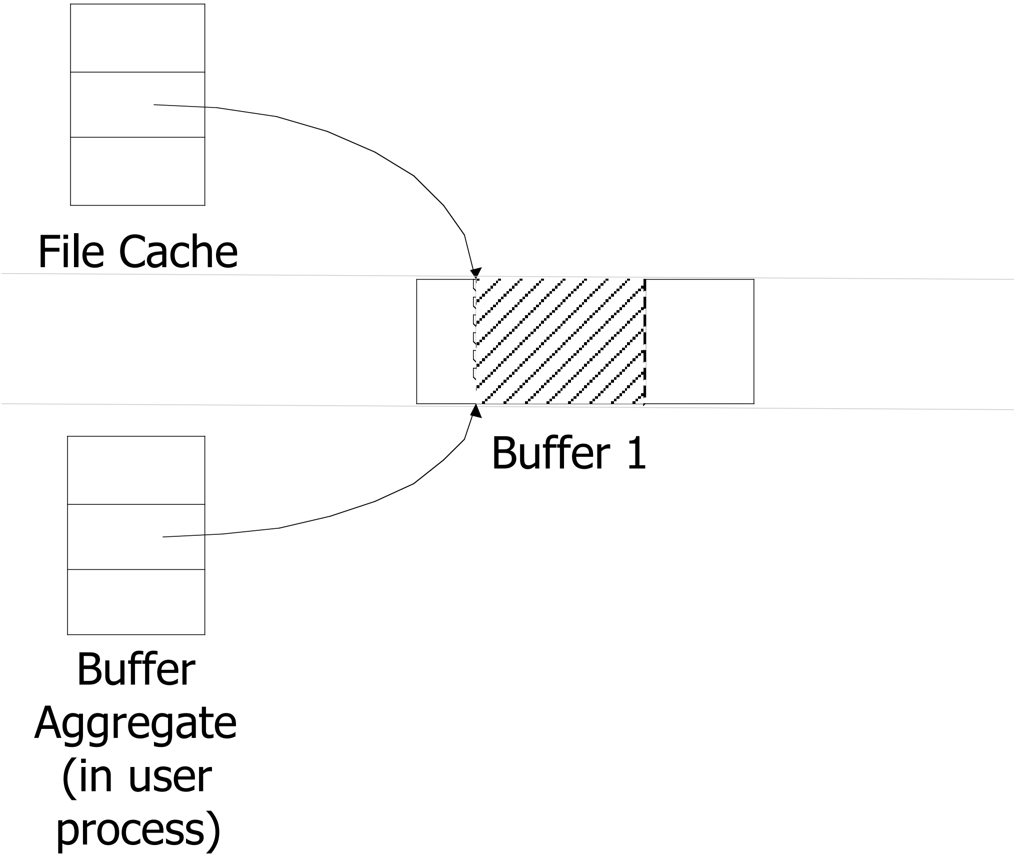
IO-Lite



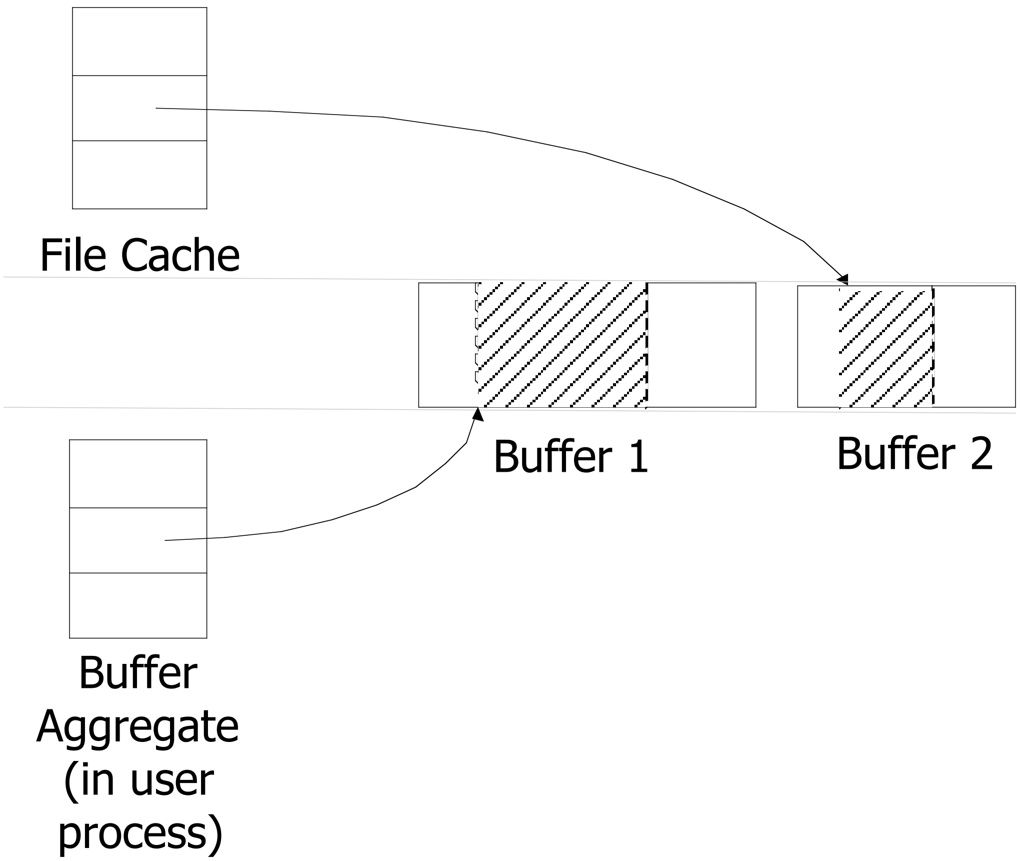
immutable --> mutable



immutable --> mutable



immutable --> mutable



Consequences of mutable bufs

- Whole buffers are rewritten
  - same as if there was no IO-Lite -- same penalty as a data copy
- Bits and pieces of files are rewritten
  - what this system was designed for -- ADT handles modified sections nicely
- Too many bits and pieces are rewritten
  - IO-Lite uses mmap to make it contiguous -- usually results in a kernel memory copy

Evicting I/O pages

- LRU policy on unreferenced bufs (if one exists)
- Otherwise, LRU on referenced bufs
  - since bufs can have multiple references, might require multiple write-backs to disk
- Tradeoff between size of I/O cache and size of VM pages
  - greater than 50% replaced pages are IO-Lite, evict one to reduce the number

The bad news

- Applications must be modified to use special IO-Lite read/write calls
- Both applications at either end of a UNIX pipe must use library to gain benefits of IO-Lite's IPC

The good news

- Many applications can take further advantage of IPC
  - computing packet checksums only once

The good news

- Many applications can take further advantage of IPC
  - computing packet checksums only once

<generation #, addr> --> I/O buf data

Flash-Lite

- Flash web server modified to use IO-Lite
- HTTP
  - up to 43% faster than Flash
  - up to 137% faster than Apache
- Persistent HTTP (less TCP overhead)
  - up to 90% network saturation
- Dynamic pages have advantage because of IPC between server and CGI program

HTTP/PHTTP

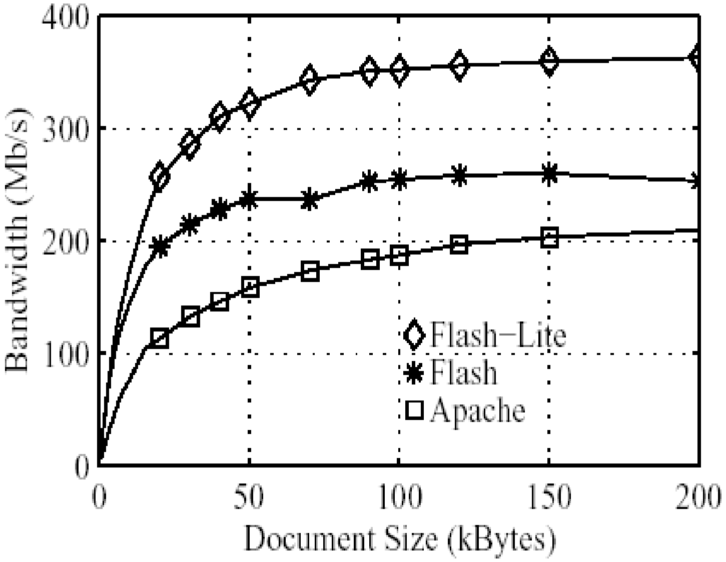


Figure 3: HTTP

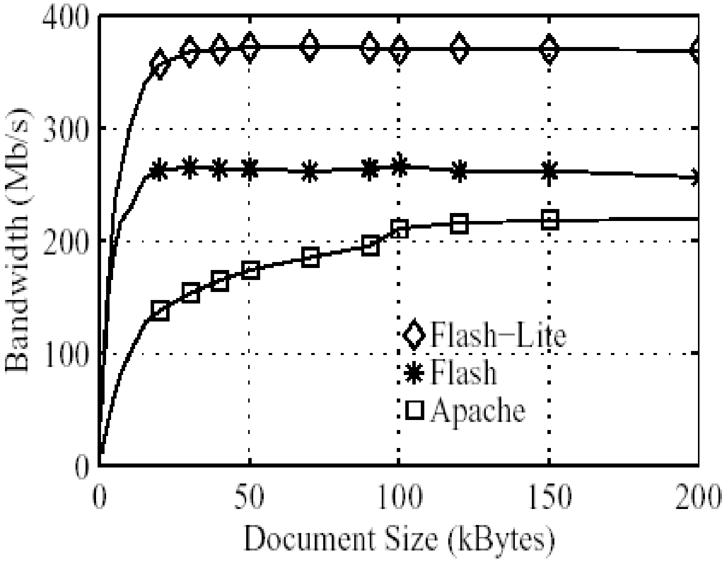


Figure 4: Persistent HTTP

PHTTP with CGI

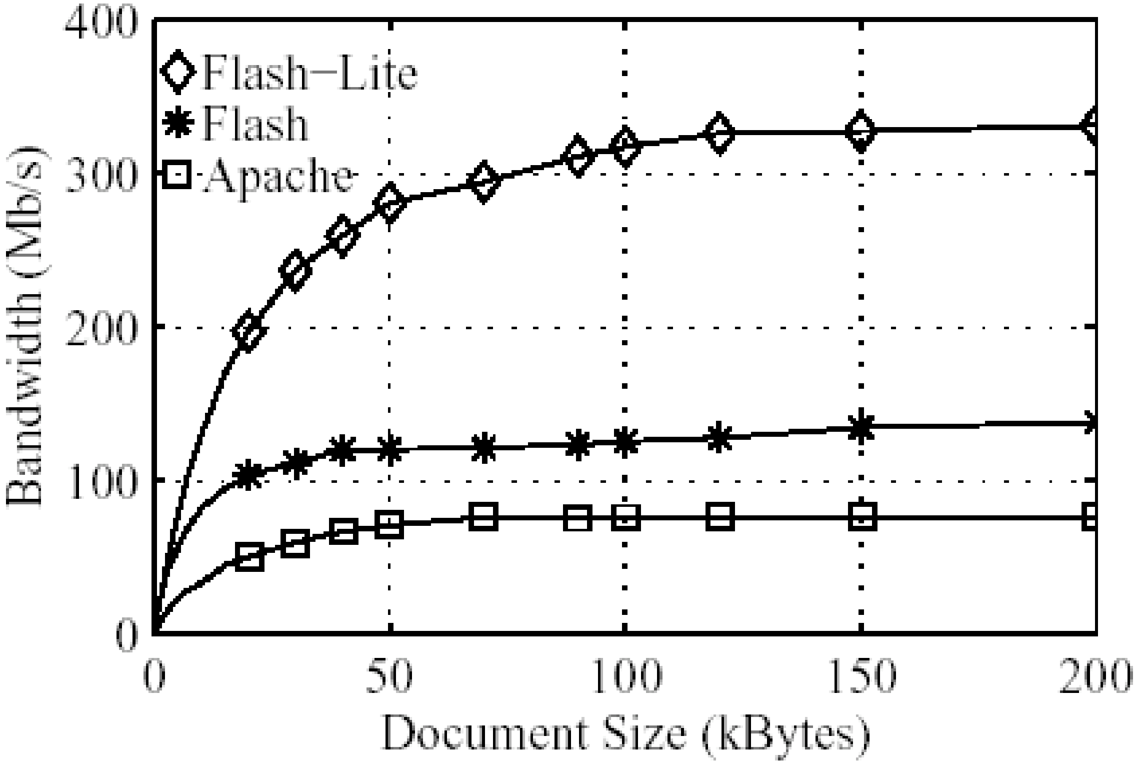
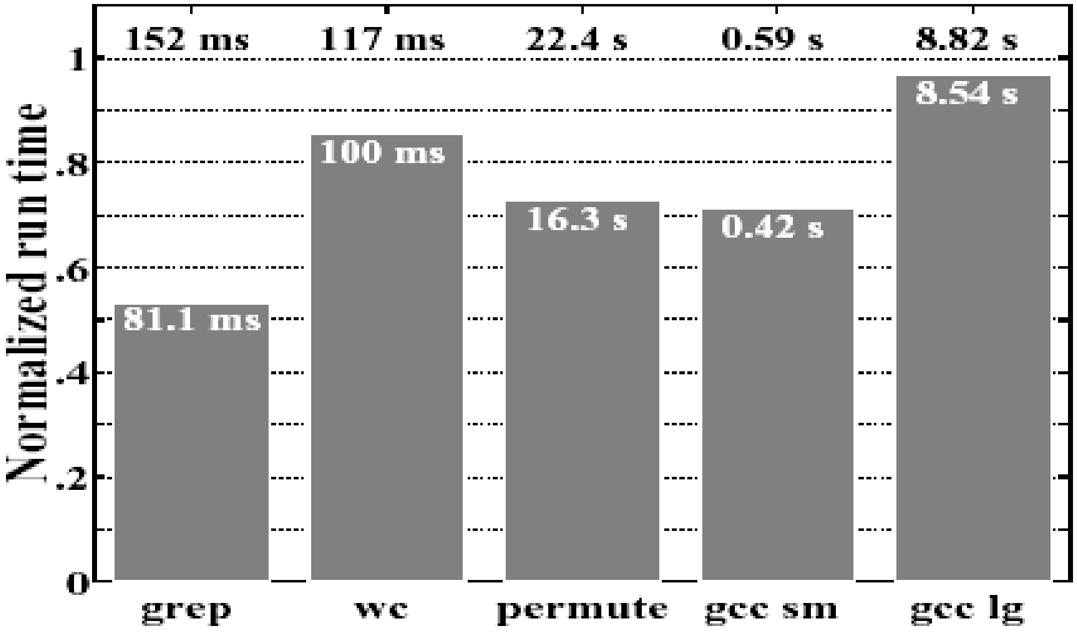


Figure 6: P-HTTP/FastCGI

Something else fbufs can’t do

- Non-network applications
- Fewer memory copies across IPC

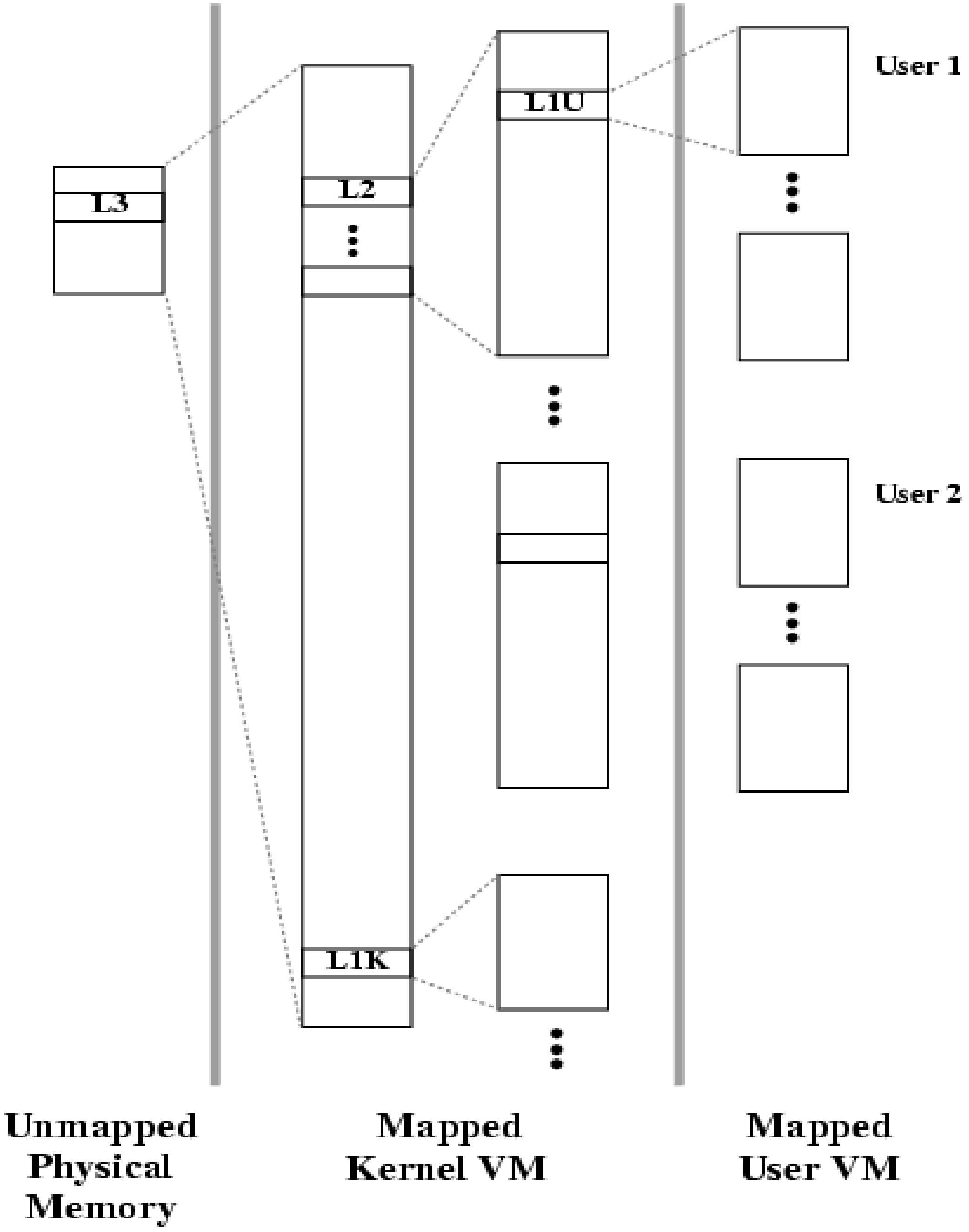


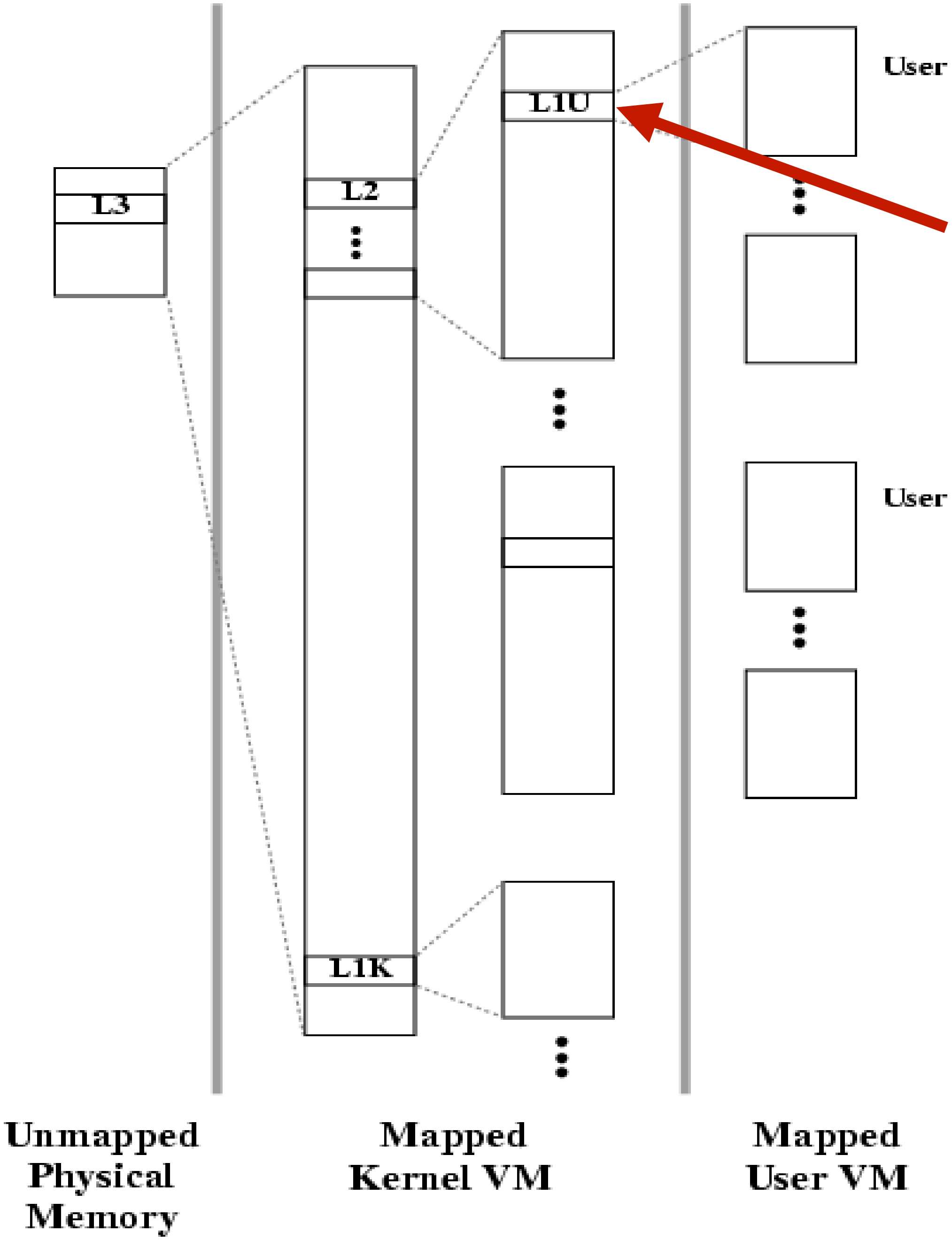
On to prefetching/caching…

- Once again, CPU speeds far exceed main memory speeds
- Tradeoff
  - prefetch too early --> less cache space
  - cache too long --> less room for prefetching
- Try to strike a balance

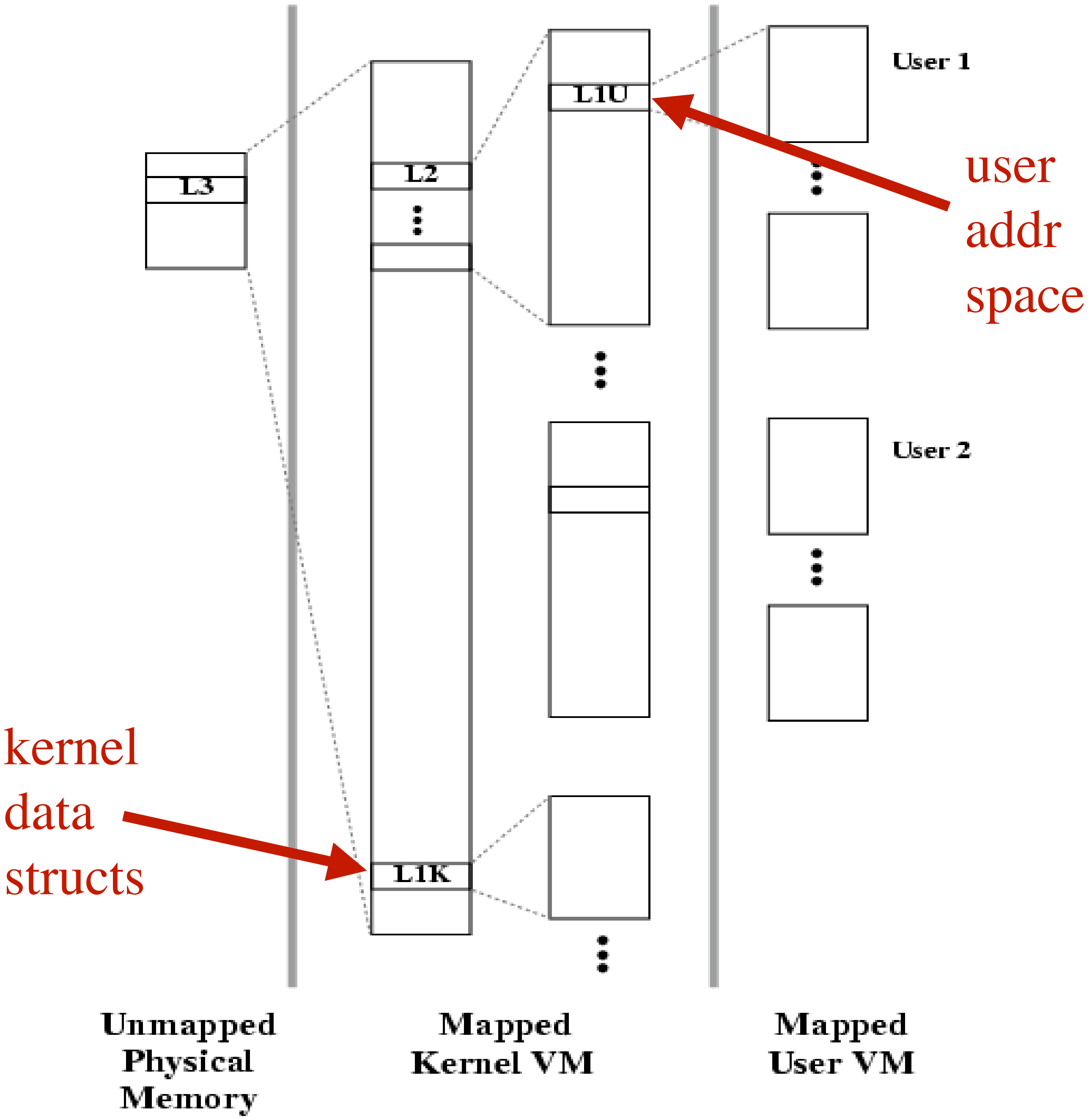
Let’s focus on the TLB

- Microkernel modularity pays a price: more TLB misses
- Solution in software -- no hardware mods
- Handles only kernel misses -- 50% of total


L3
L2
L1U
User 1
User 2
L1K
Unmapped Physical Memory
Mapped Kernel VM
Mapped User VM

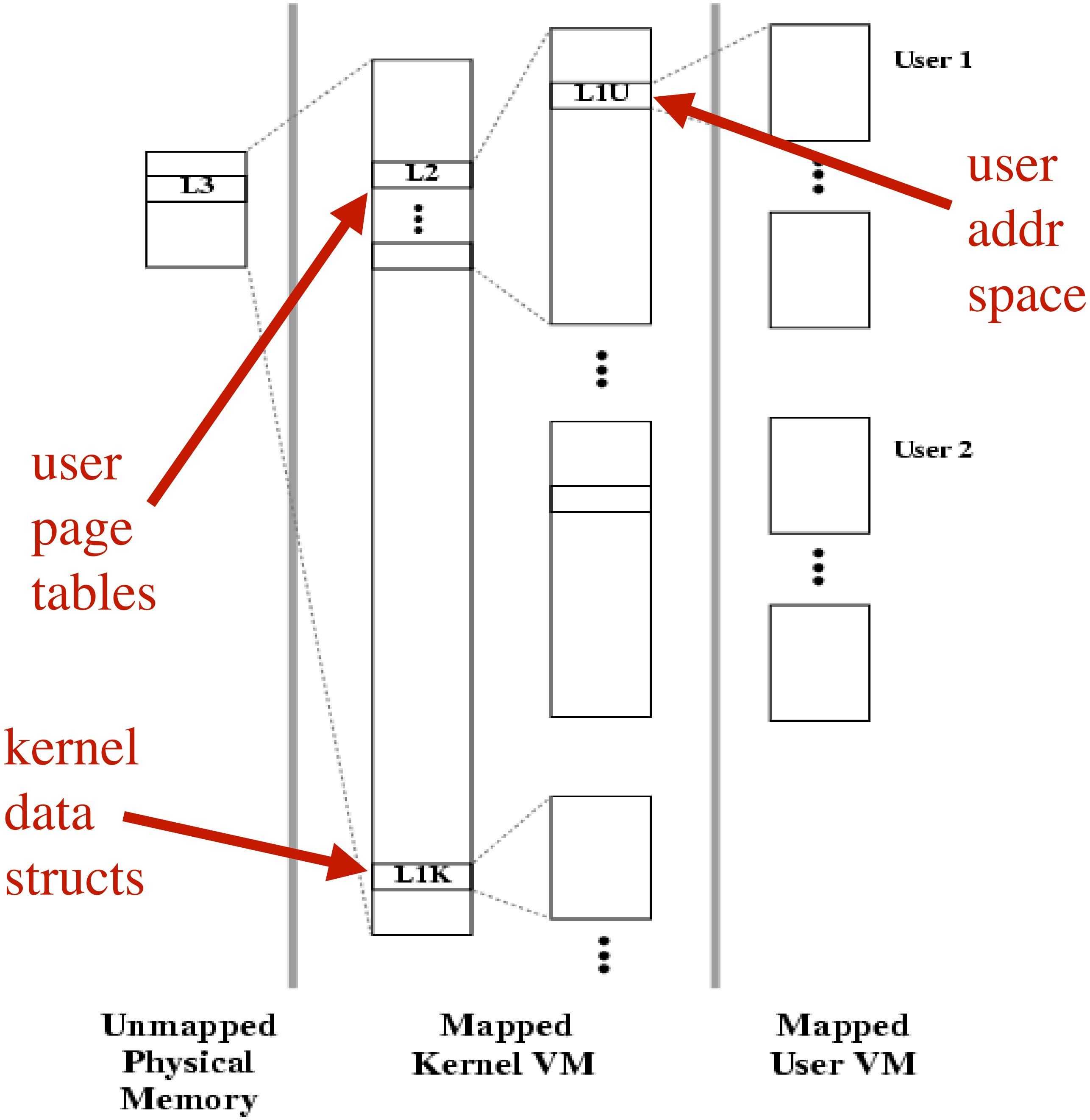


user
addr
space



user addr space

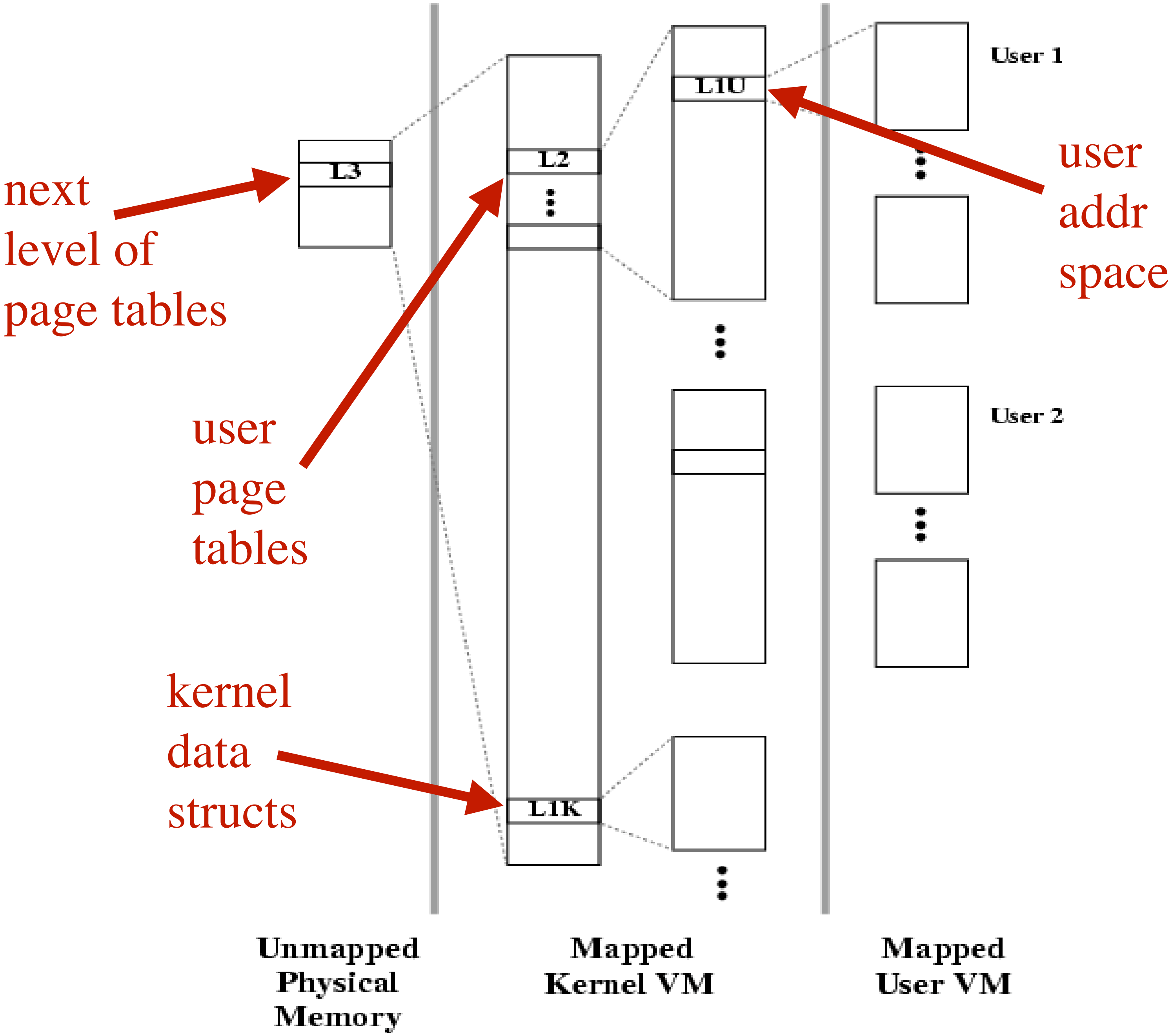
kernel data structs



user addr space

user page tables

kernel data structs

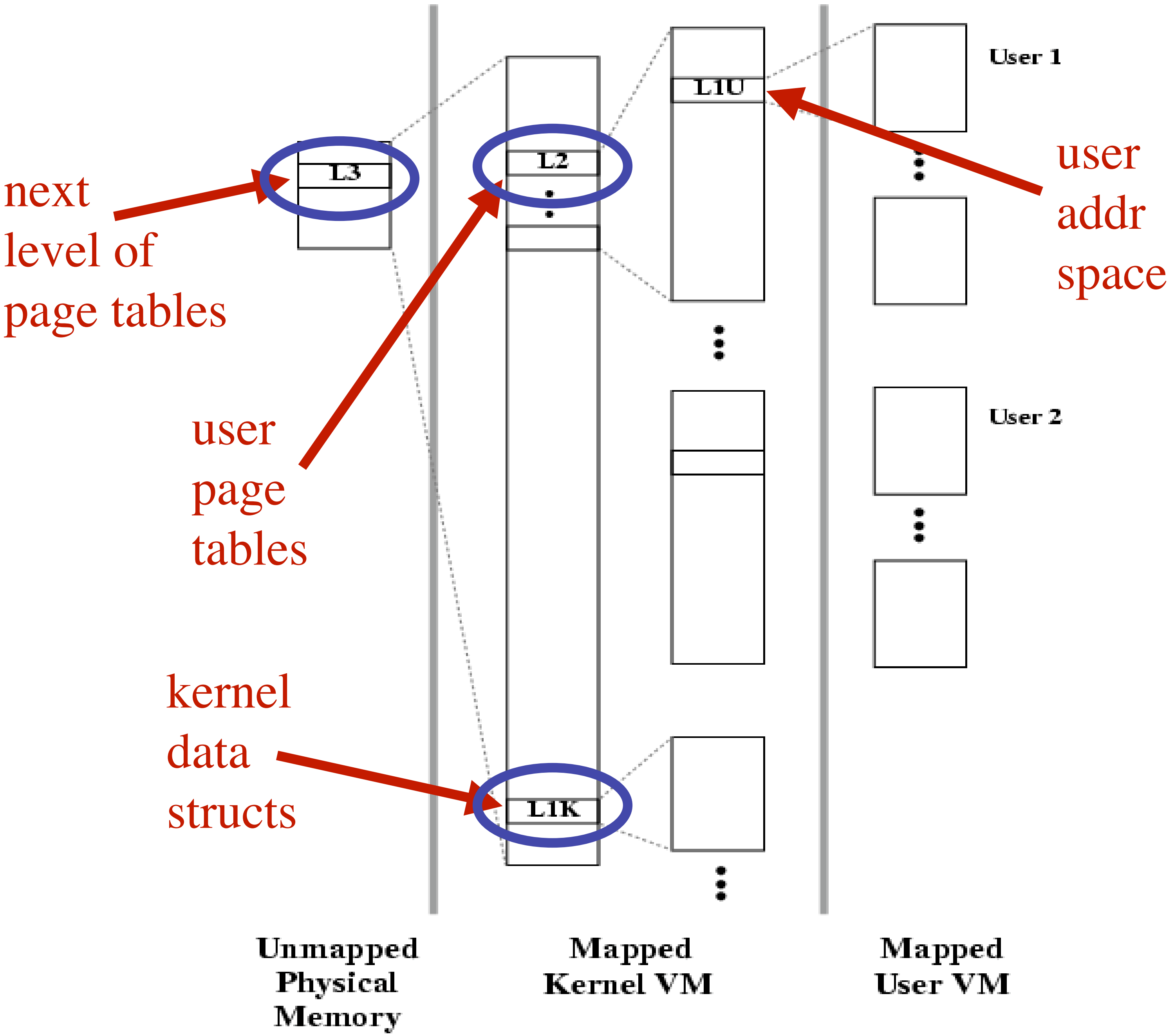


next level of page tables

user page tables

kernel data structs

user addr space



next level of page tables

user page tables

kernel data structs

user addr space

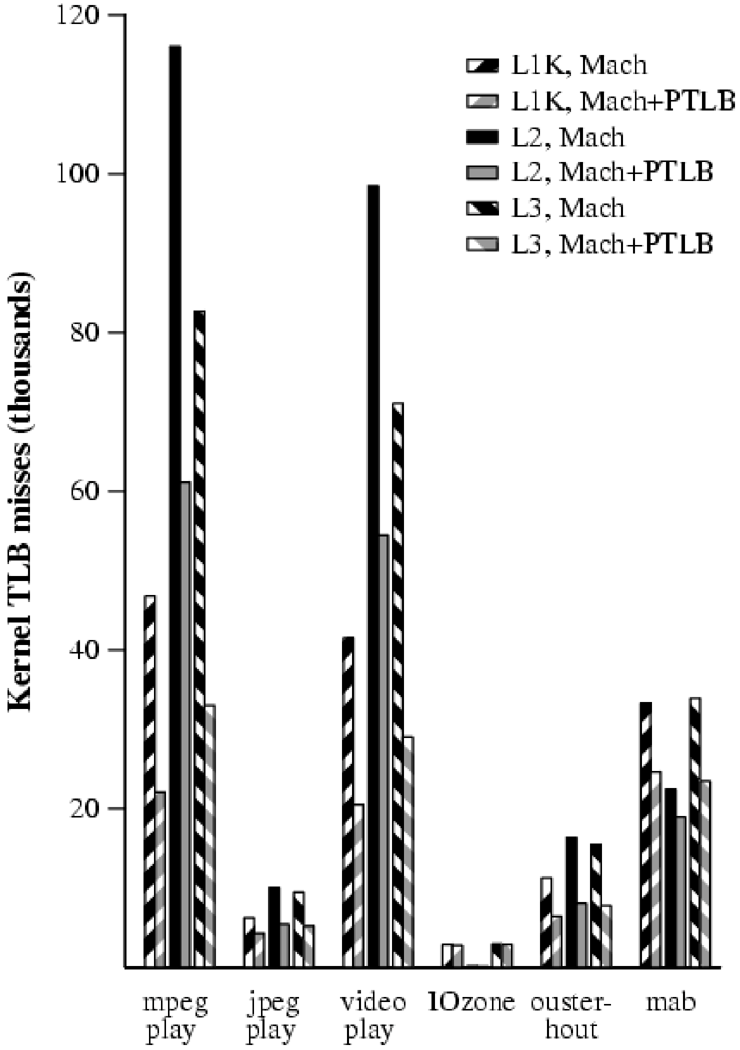
Prefetching

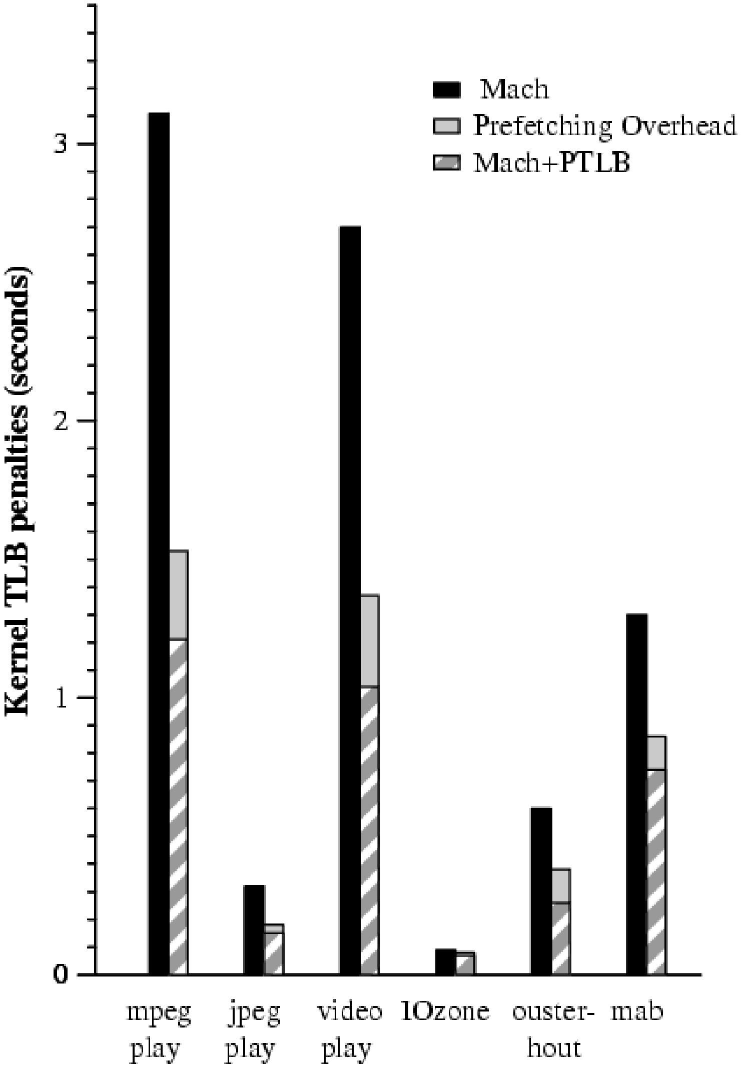
- Prefetch on IPC path
  - concurrency in separate domains increases misses
  - fetch L2 mappings to process stack, code, and data segments
- Generic trap handles misses first time, caches them in flat PTLB for future hash lookups

Caching

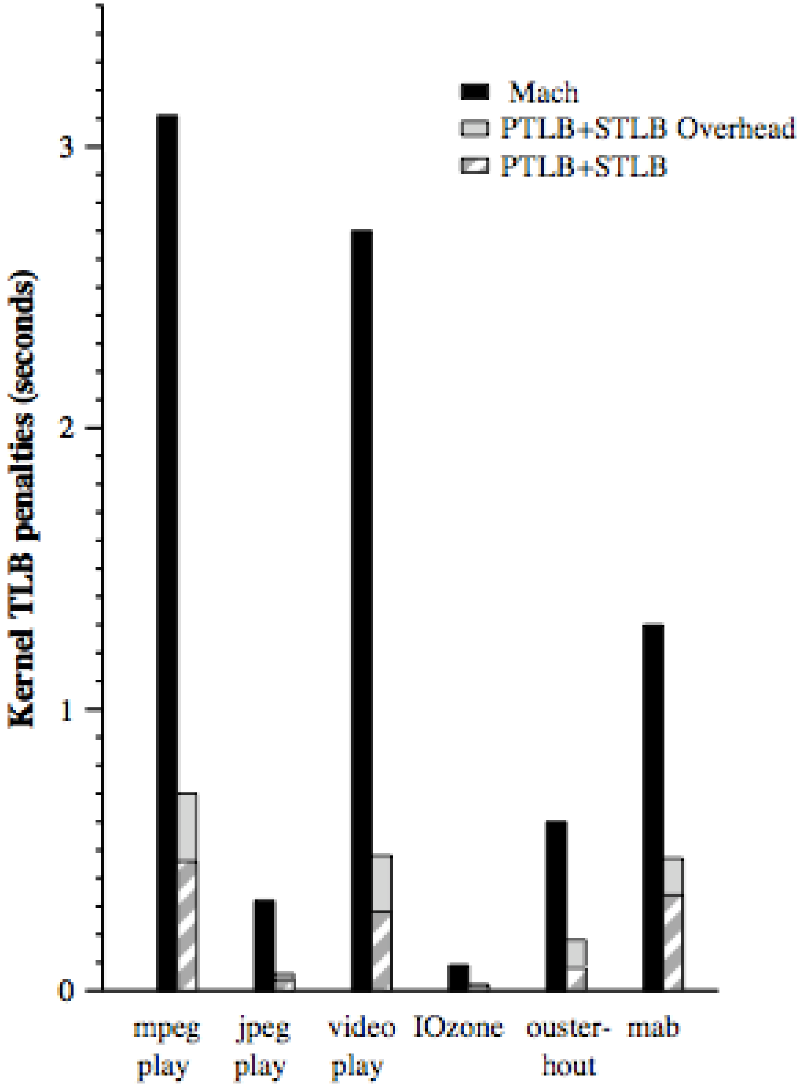
- Goal: avoid cascaded misses in page table
  - entries evicted from TLB are cached in STLB
  - adds 4-cycle overhead to most misses in general trap handler
- When using STLB, don't prefetch L3
  - usually evicts useful cached entries
- In fact, using both caching + prefetching only improves performance if have a lot of IPCs, such as in servers

Performance -- PTLB

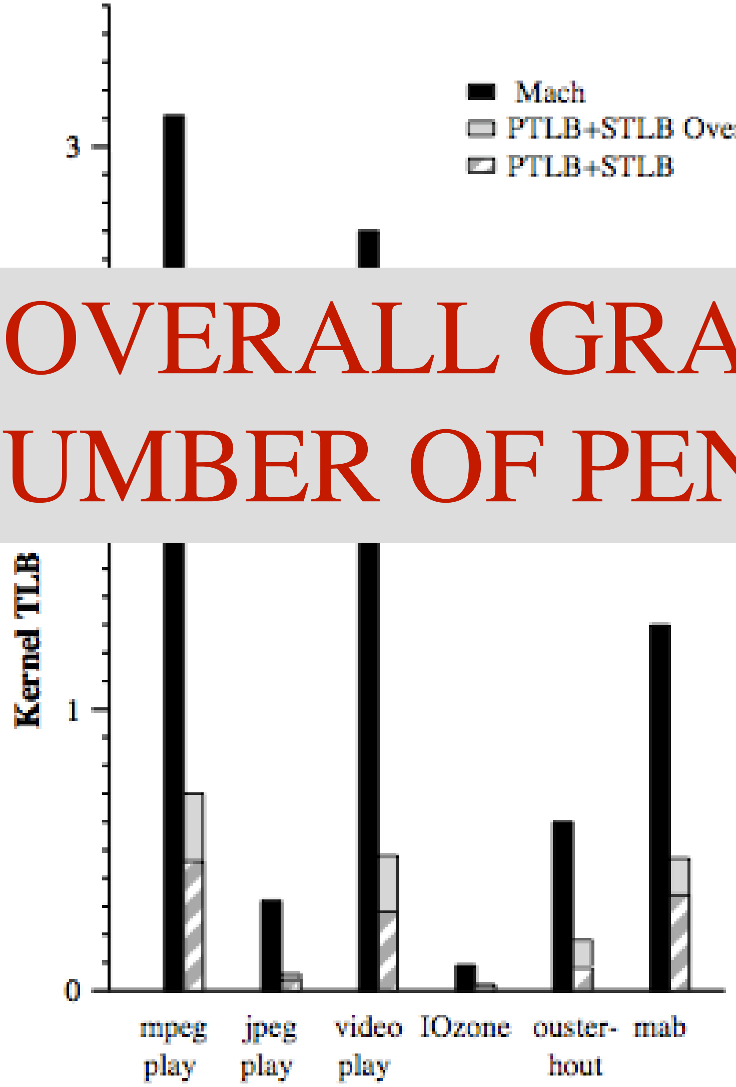




Performance -- overall



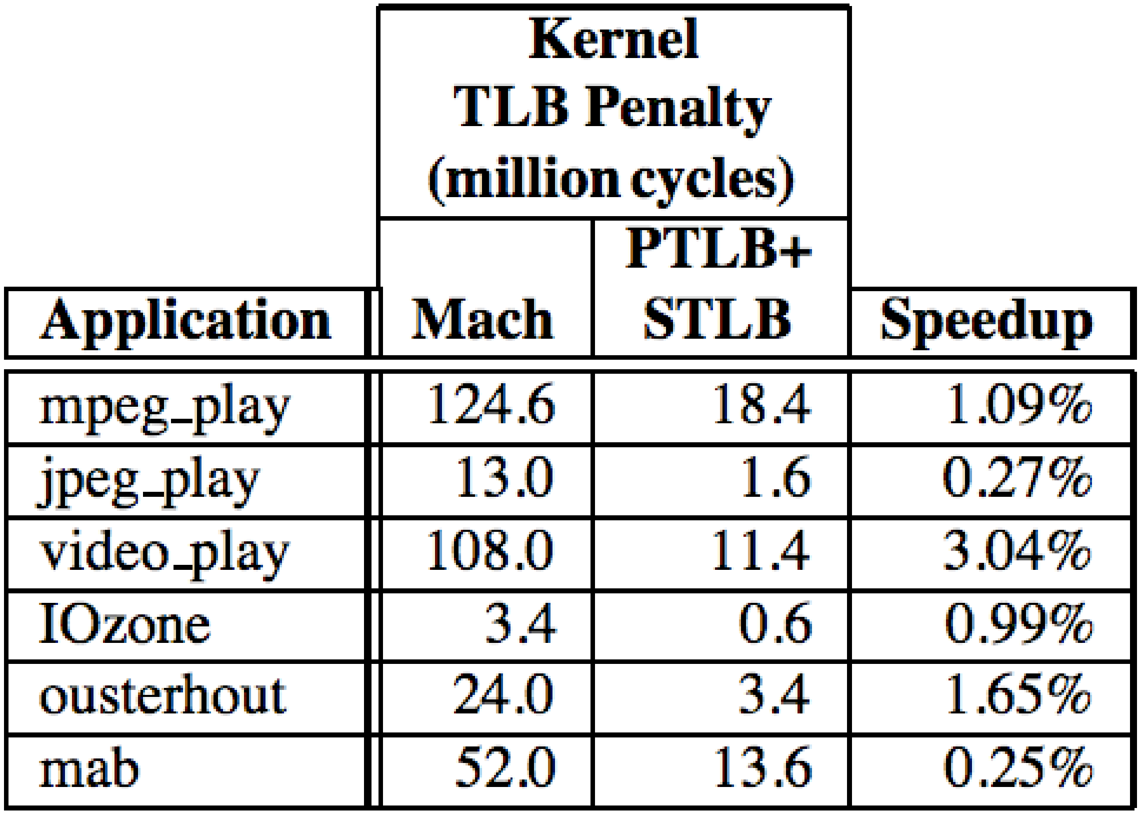
Performance -- overall



BUT NO OVERALL GRAPH GIVEN FOR NUMBER OF PENALTIES

Amdahl’s Law in action

- Overall performance only marginally better



	Kernel TLB Penalty (million cycles)		
Application	Mach	PTLB+ STLB	Speedup
mpeg_play	124.6	18.4	1.09%
jpeg_play	13.0	1.6	0.27%
video_play	108.0	11.4	3.04%
IOzone	3.4	0.6	0.99%
ousterhout	24.0	3.4	1.65%
mab	52.0	13.6	0.25%

Summary

- Bridging the gap between memory speeds and CPU is worthwhile
- Microkernels have fallen out of favor
  - but could come back
  - relatively slow memory is still a problem
- Sharing resources between processes without placing too many restrictions on the data is a good approach