

The Multikernel: A new OS architecture for scalable multicore systems

Toward scalable operating
system for many core platform

Hao Luo
11/29/2011

Scalability 101

- In ideal world

more cores = higher performance

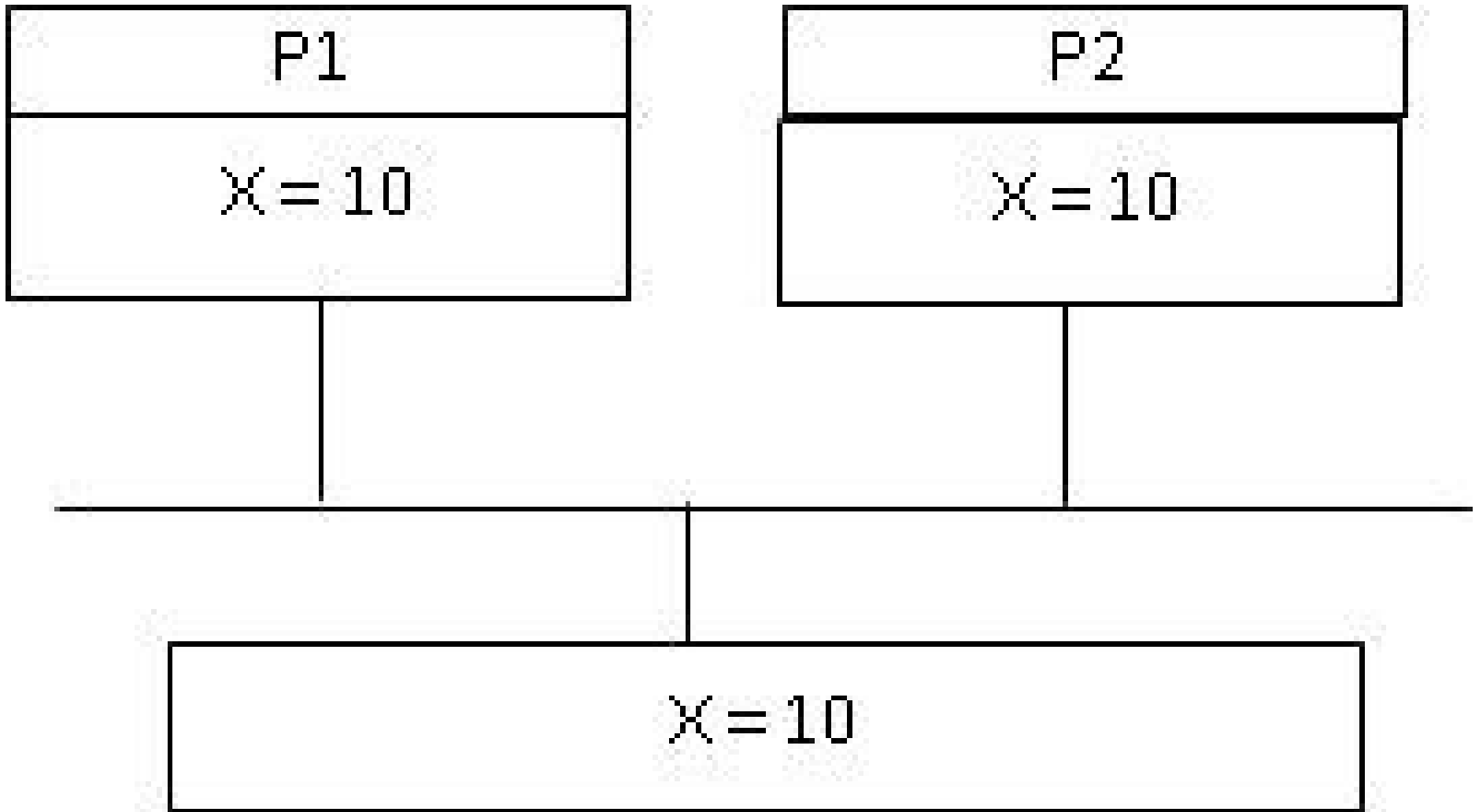
- In real life

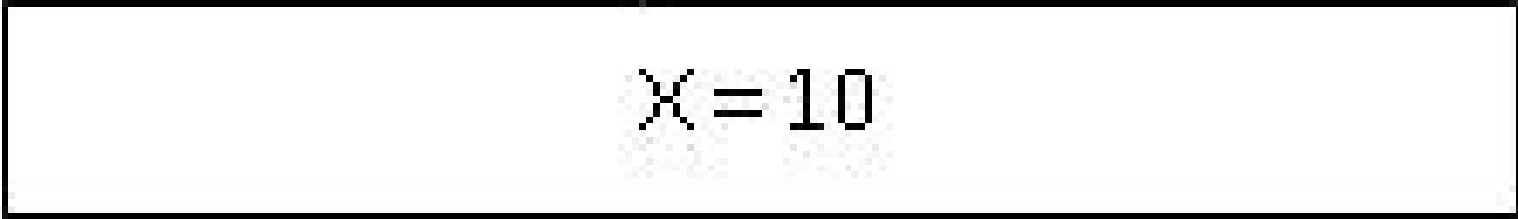
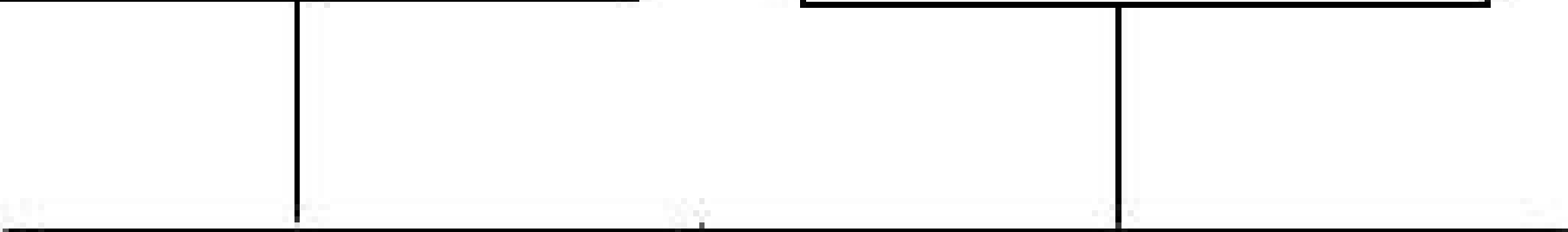
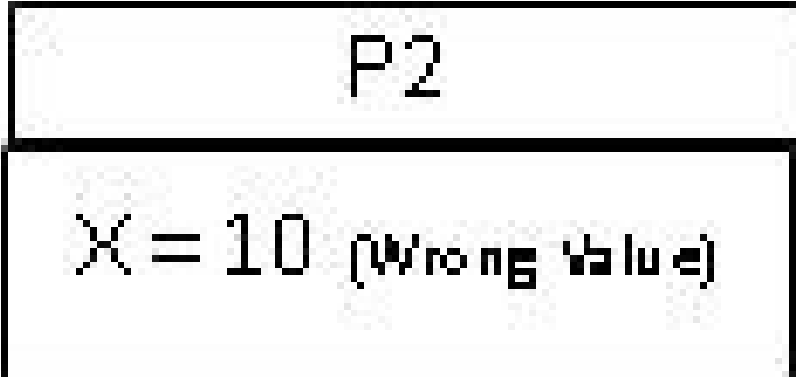
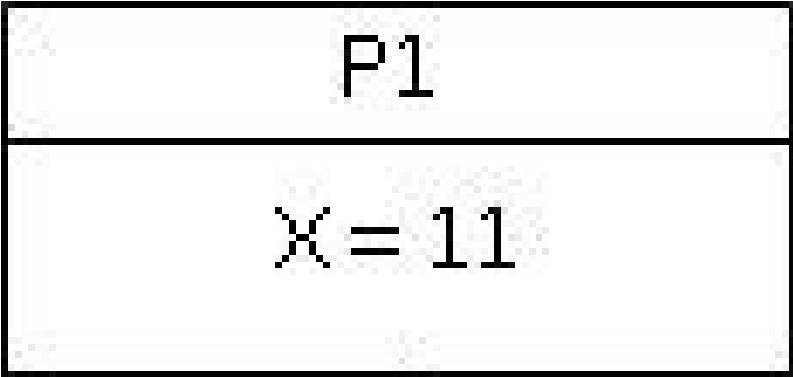
more cores ~~≠~~ higher performance

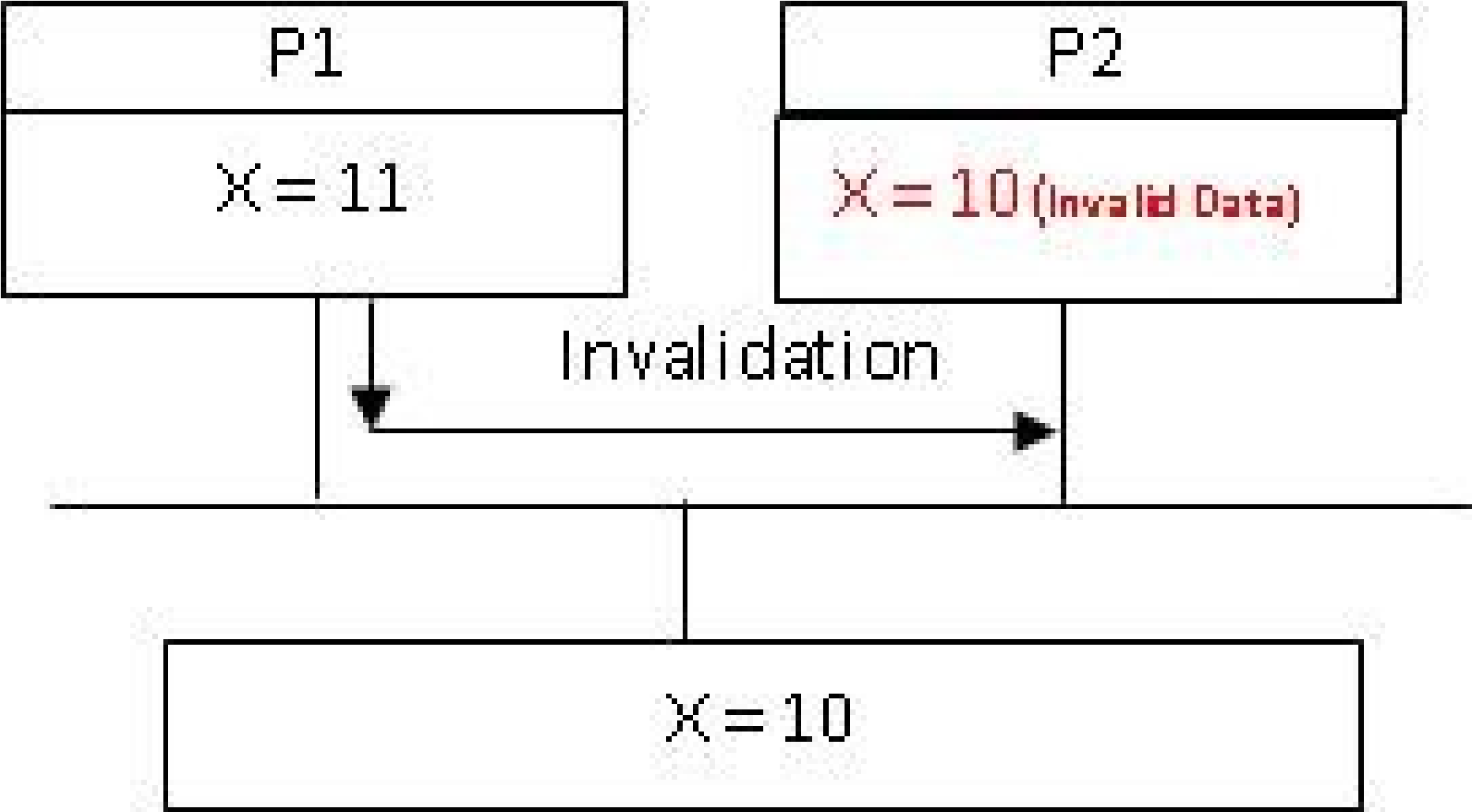
- Lock wait time when tasks need exclusive access to shared data
- The time cost for cache coherence protocol to fetch cache line into exclusive mode
- Tasks compete for limit sized hardware cache space
- Tasks compete for shared resources, e.g. interconnect, DRAM interface
- Too few tasks to keep cores busy

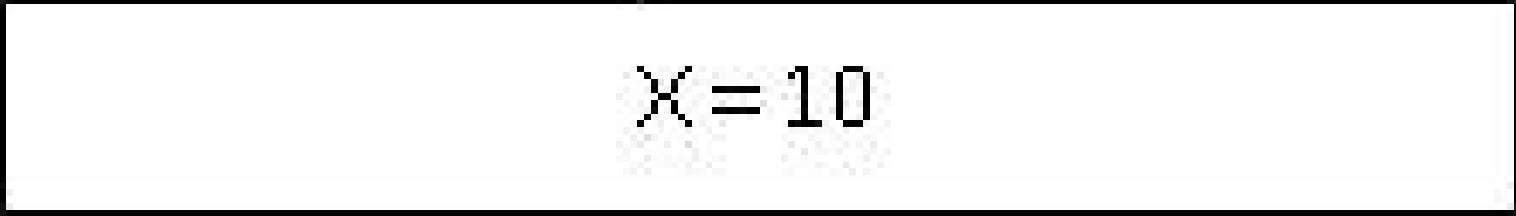
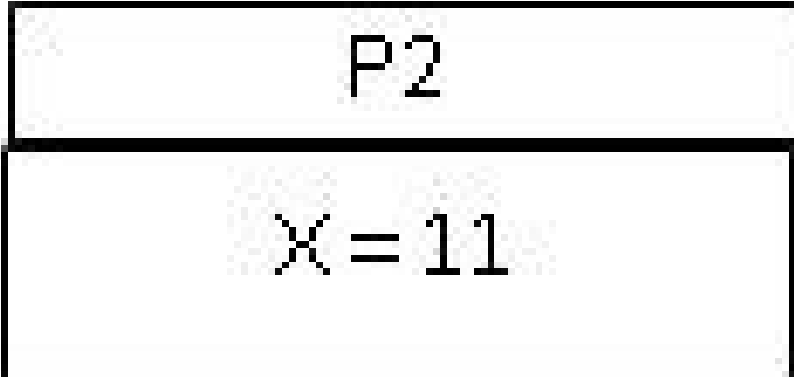
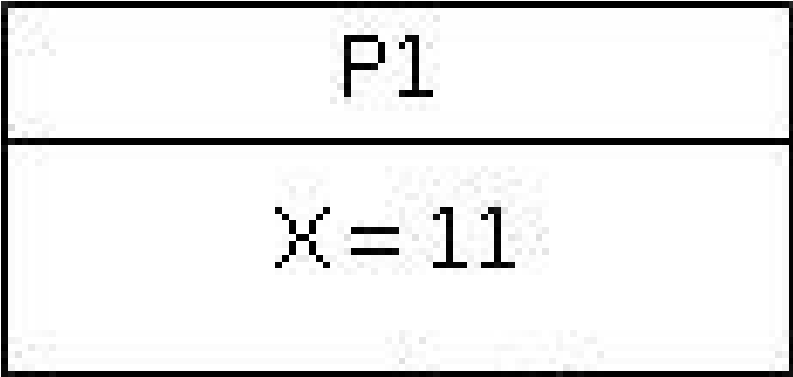
Cache misses

Data Reads and Writes










Bottleneck

- Invalidate all cores
- Fetch most fresh data

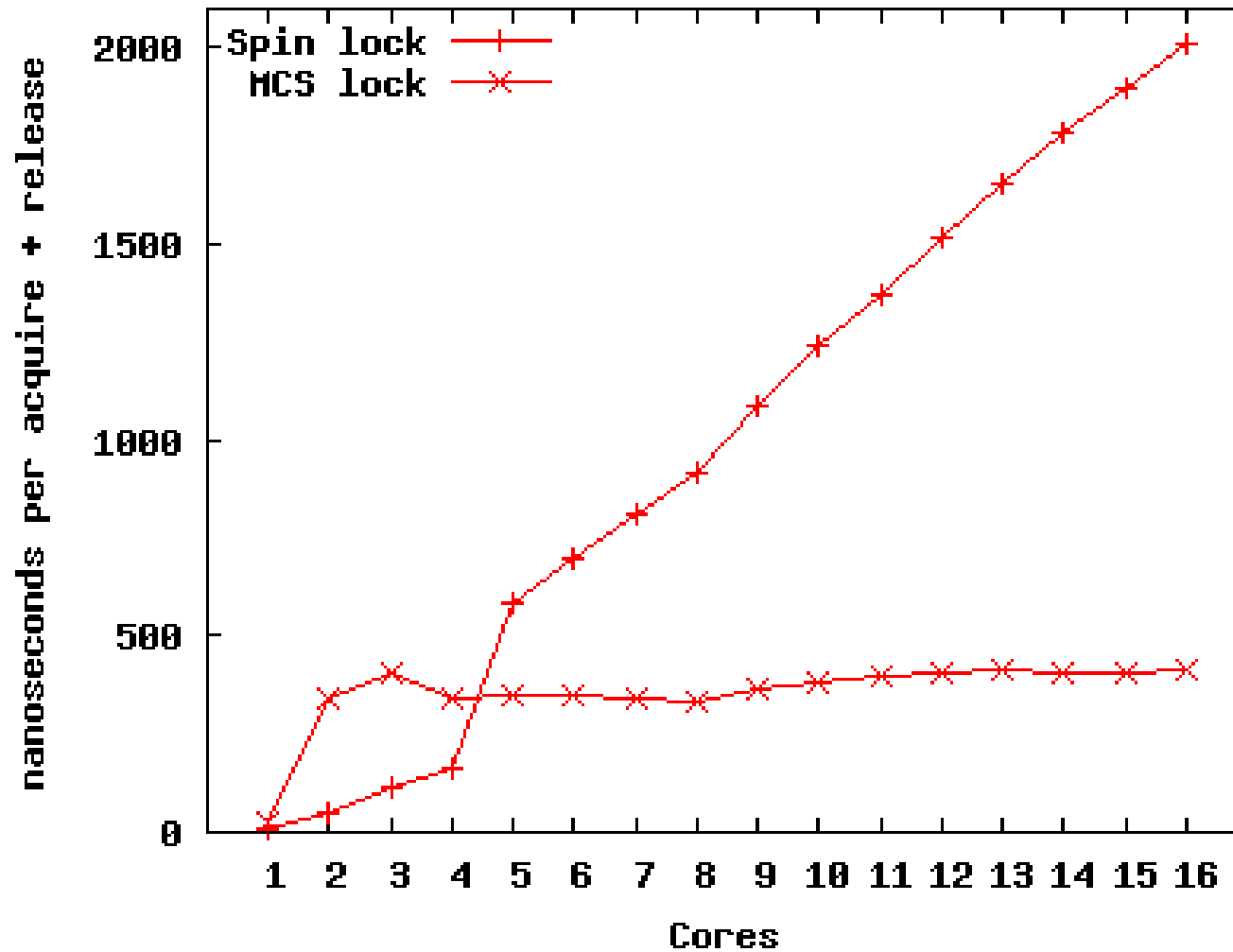
To Change or Not

- A. Exploit techniques already in use in larger multiprocessor systems [OSDI '10]
-  B. New OS architecture for future multicore platform [SOSP '09]

Challenges

1. Diverse system implementation

Spinlock v.s. MCS lock [OSDI '08]



2. Diverse Cores

- Different ISA
- Different performance characteristics
(e.g. GPU, x86, x64, ARM etc.)

3. Interconnect matters

Hardware resembles messaging network



High performance requires
adaptation to intercore-topology

4. Cache coherence is no panacea

On even 80 core platform, overhead is a concern

[SC '08]

5. Messages v.s. shared memory

Effective memory sharing is difficult

(Remember various kernel locking mechanism?)

Shared memory has worse cache efficiency

(Recall those cache updates!)

Shared memory has worse
cache efficiency

(Recall those cache updates!)

Local cache usage

(because message can
be located in cache)

Three Principles

- All inter-core communication explicit
- OS structure should be hardware-neutral
- View states as replicated instead of sharing

Explicit Inter-core Communication

Reasoning about interconnect usage

Exploit network optimization technique

Isolation and better resource manage

Three Principles

- All inter-core communication explicit
- OS structure should be hardware-neutral
- View states as replicated instead of sharing

Hardware Neutral Structure

Code reuse for diverse hardware

Late binding of both the protocol
implementation and message transport

Three Principles

- All inter-core communication explicit
- OS structure should be hardware-neutral
- View states as replicated instead of sharing

View states as replicated

Required for no memory sharing

Support for changeable hardware setting

Barrelfish



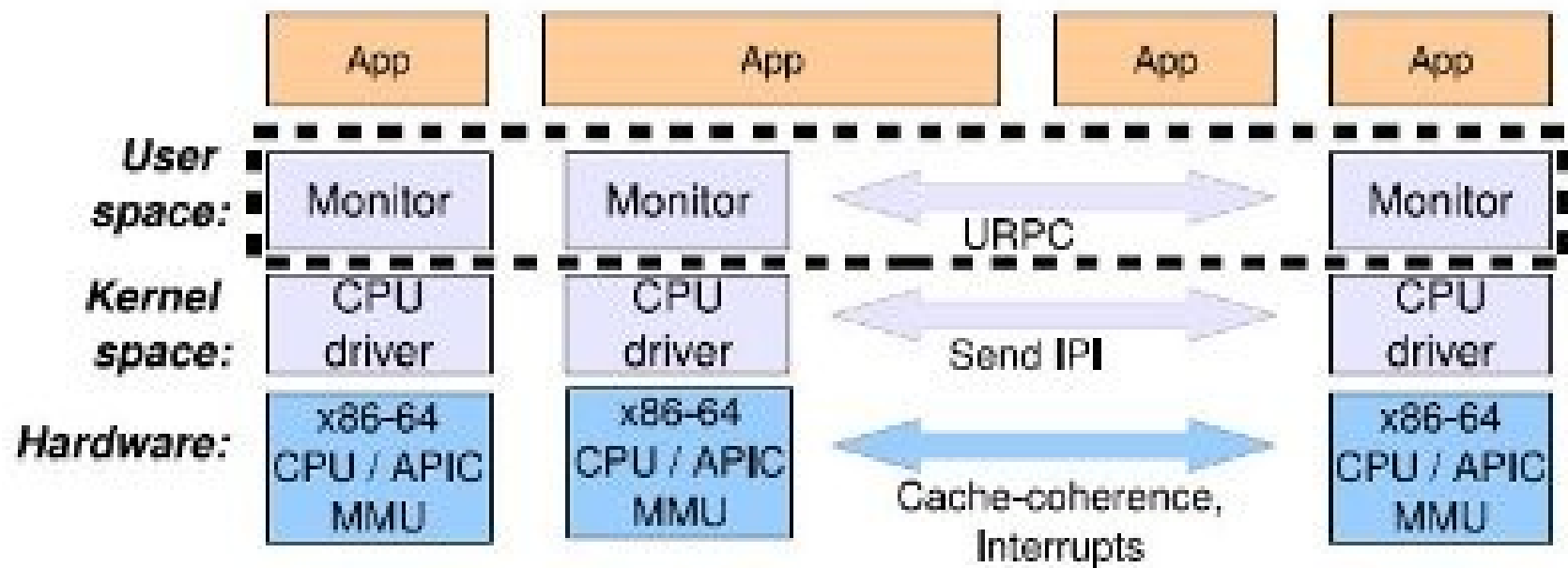


Figure 5: Barrelfish structure

CPU driver

Similar to exokernel

No mm responsibility, very little work, abstract very little for performance

Share no state with other cores

Event-driven Single-threaded

Nonpreemptable

Serially processes events

- Trap from user space

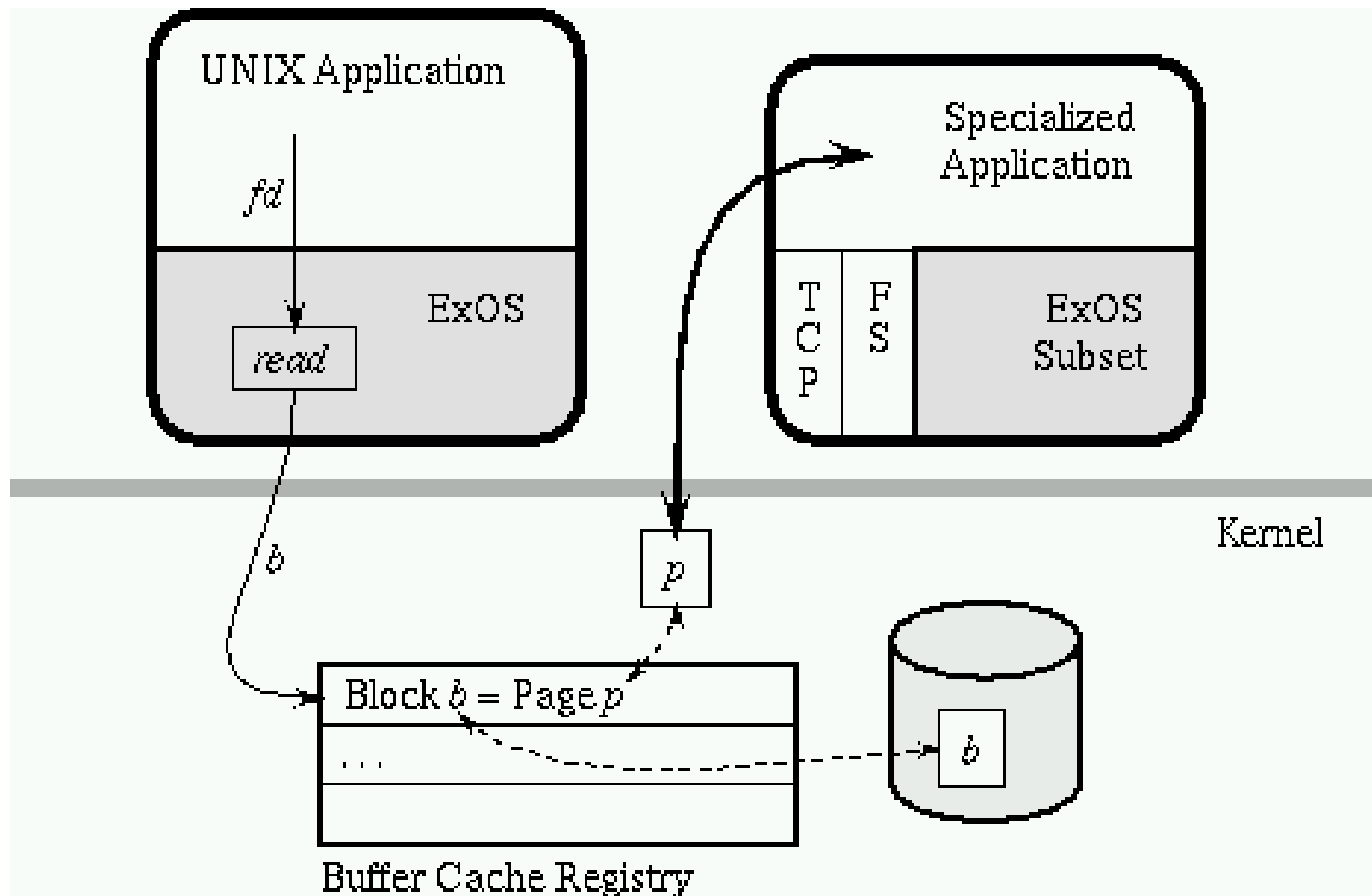
- Device interrupts

- Interrupts from other cores

Implements lightweight, asynchronous communication

Checking correctness of memory usage

Exokernel [SOSP '97]



Features

- Great modularity
- Modules use message passing for communication
- Kernel provides very little resource abstraction

Monitor

- Coordinate system states
- Single-core user-level application supports for OS scheduling
- Handles user-level application requests for state information
 - Memory allocation table
 - Address space mapping
 - Virtual memory management
- Inter-core communication thru' URPC
 - Shared memory used as message channel
 - Sender writes message to cache line
 - Receiver polls cache line to read message

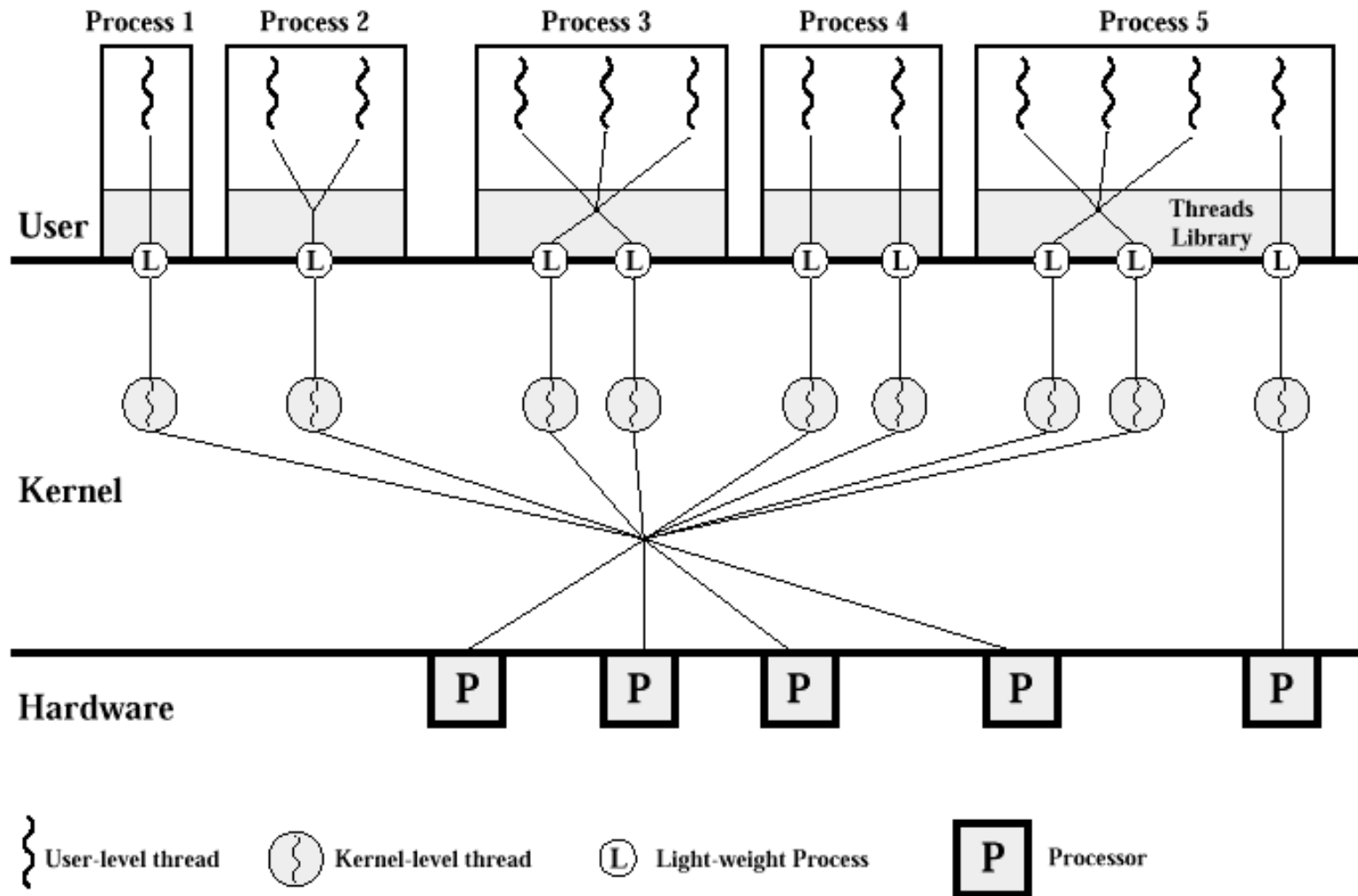
Process

- similar mechanism to Scheduler Activation
- process represented as dispatchers
- Dispatcher is a kernel object, works similar to **LWP**.
- Each dispatcher(**LWP**) is bound to one core(**kernel thread**).
- Dispatchers provide upcall interfaces for CPU driver to schedule.
- Each dispatcher runs a local user-level thread scheduler.

Scheduler Activation

- Scheduler Activations is a **threading mechanism**, also "N:M" strategy, that is mapping N user threads onto M kernel threads
- (implemented in **Solaris**)
- User-level thread and kernel-level thread scheduling needs coordination. LWP provides "**upcall**" interface for kernel to notify user thread scheduler about its scheduling decision(blocking, termination, creation)

Solaris Threading Model [6]



Memory Management

- **Capability model** borrowed from **seL4**
- Export the management of kernel resources to user level and subjects them to the same **capability-based access control** as user resources
- Think of file descriptor in Linux
Reference to kernel object + Access rights
- User-level monitors dynamically allocate raw physical memory
- Kernel-level CPU drivers verify correct usage of memory

Capability Model

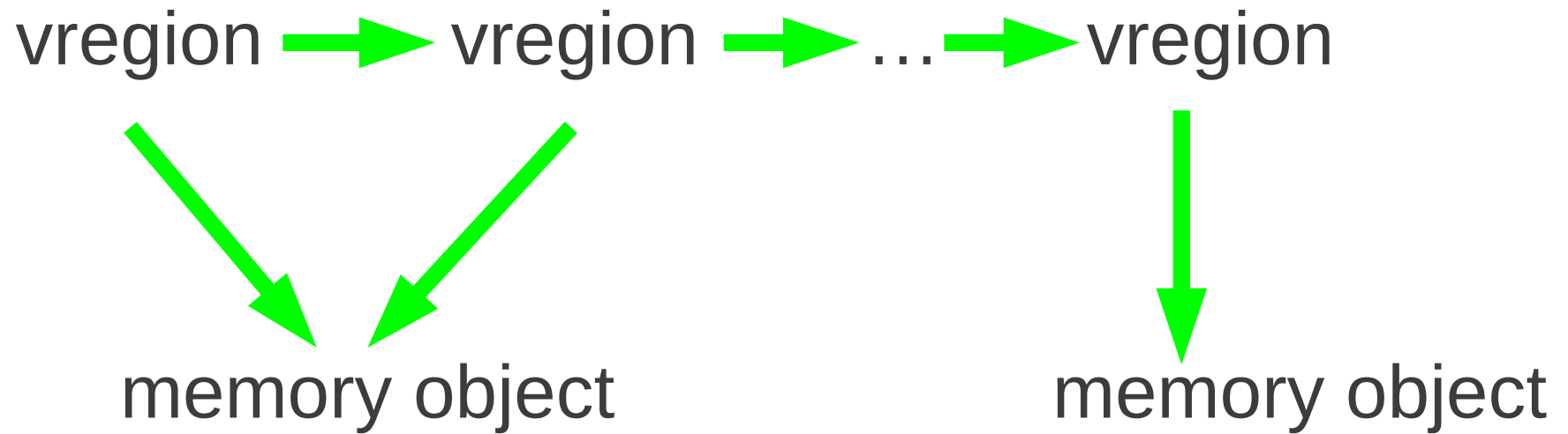
- Processes access memory thru' **capabilities**. Each capability has a **type**, e.g. "raw physical memory", "dispatcher". Capability type determines the access rights, type-specific operations(invocations) etc.
- Kernel maintain a tree of capabilities to track memory usage. The children are **retyped** from parent (as partitioning memory and refining access rights).
- Capability can also be **copied** to allow memory sharing.
- When the memory needs to be deallocated, its capability can be **deleted** or **revoked** (delete itself and all its descendents).

Virtual Memory

- Pagetable represented by **Vspace**
- **Pmap**: translate hardware specific page table mapping, deciding where a memory object is mapped to.
only hardware-dependent part
- **Vregion**: a contiguous block of virtual address space
- **memory object**: manages a block of physical memory of various types

Vspace structure

Pmap



Key to Scalability

Modularity

Locality

Reference

- [1] An Analysis of Linux Scalability to Many Cores, Silas Boyd-Wickizer, et al. OSDI '10
- [2] The Multikernel: A New OS Architecture for Scalable Multicore Systems, Andrew Baumann, et al. SOSPP '09
- [3] Corey: An Operating System for Many cores, Silas Boyd-Wickizer, et al. OSDI '08
- [4] Programming the Intel 80-core network-on-a-chip terascale processor, Timothy G. Mattson et al. SC '08
- [5] Application Performance and Flexibility on Exokernel Systems, M, Frans Kaasshoek, et al. SOSPP '97
- [6] Programming in C Unix System Calls and Subroutines using C, A.D. Marshall, 1994-2005

THANKS