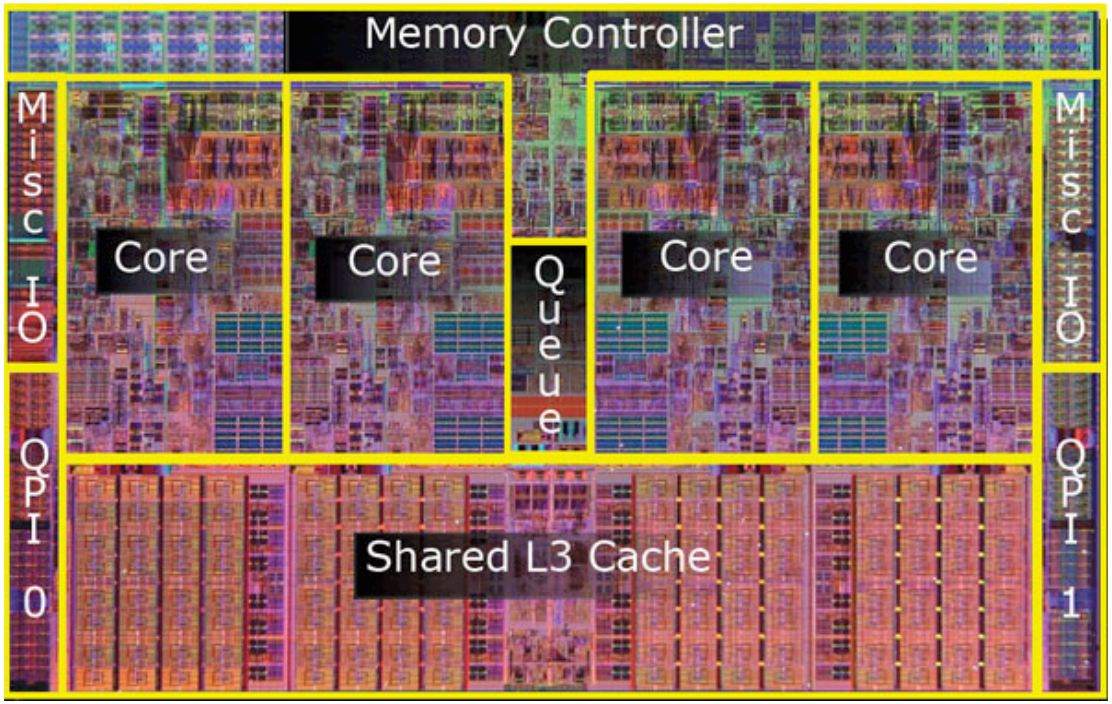
Threads and Cache Coherence in Hardware

- Previously, we introduced multi-cores.
  - Today we'll look at issues related to multi-core memory systems:
    1. Threads
    2. Cache coherence

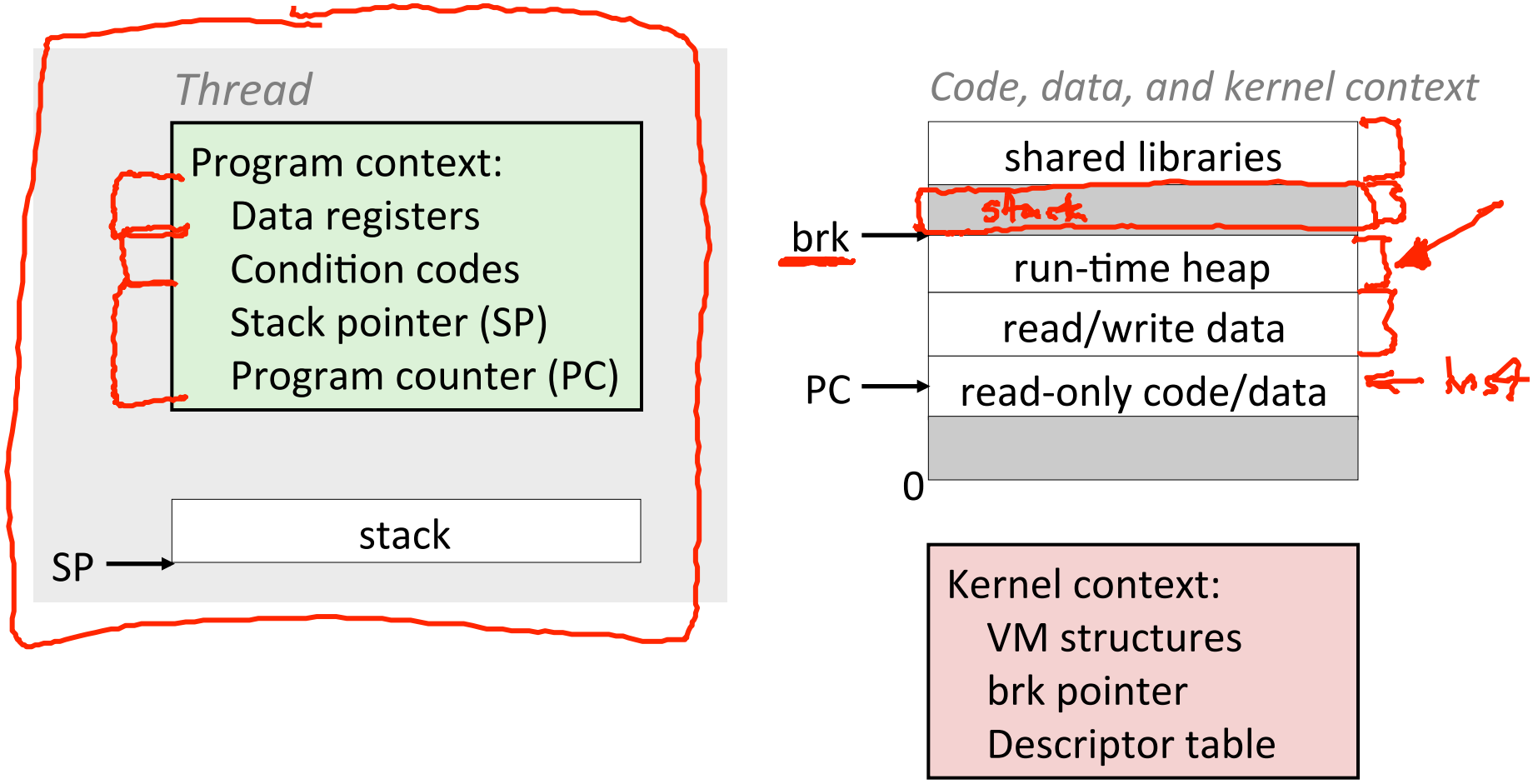


Intel Core i7

©2012 Craig Zilles

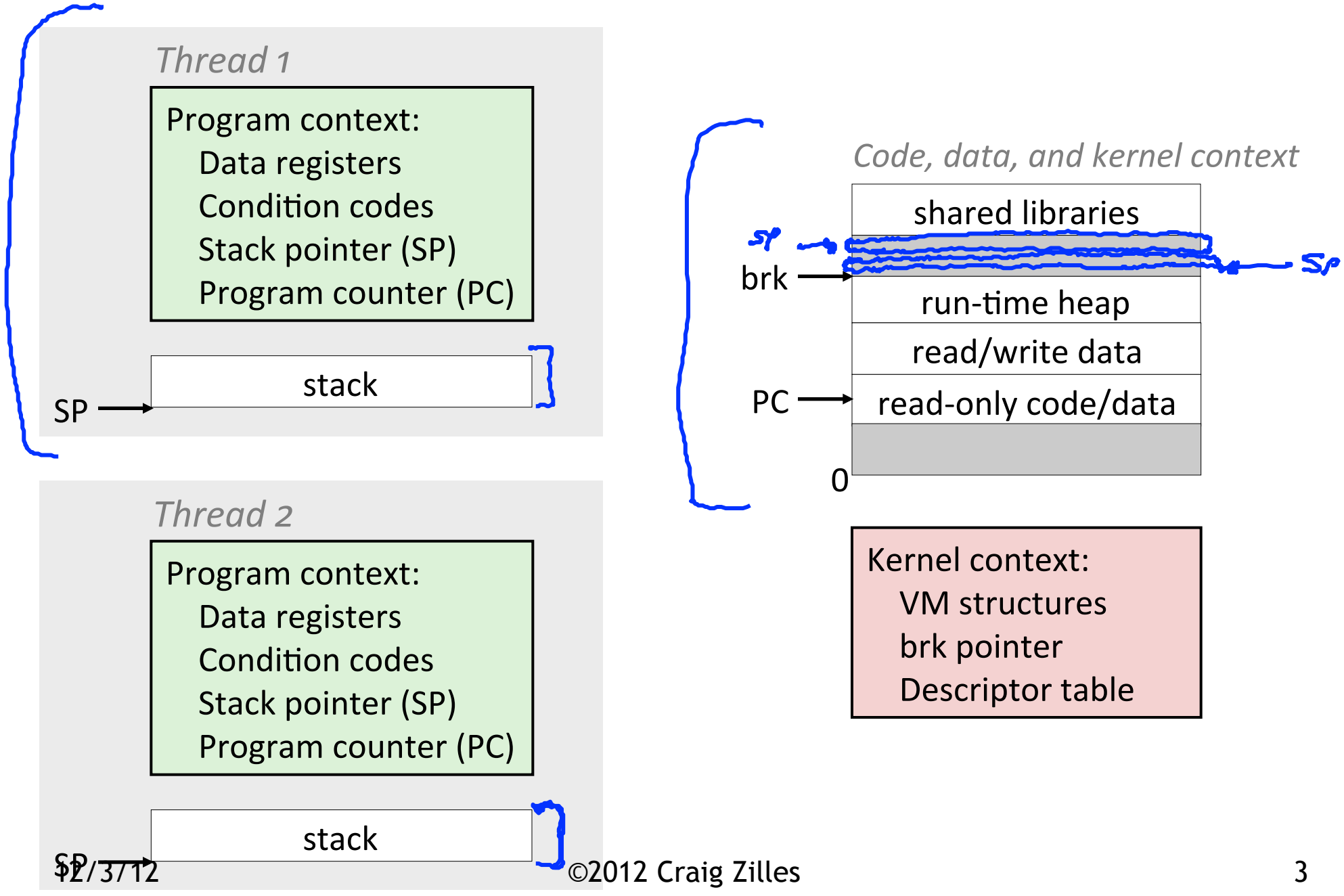
Process View

- Process = thread + code, data, and kernel context



©2012 Craig Zilles

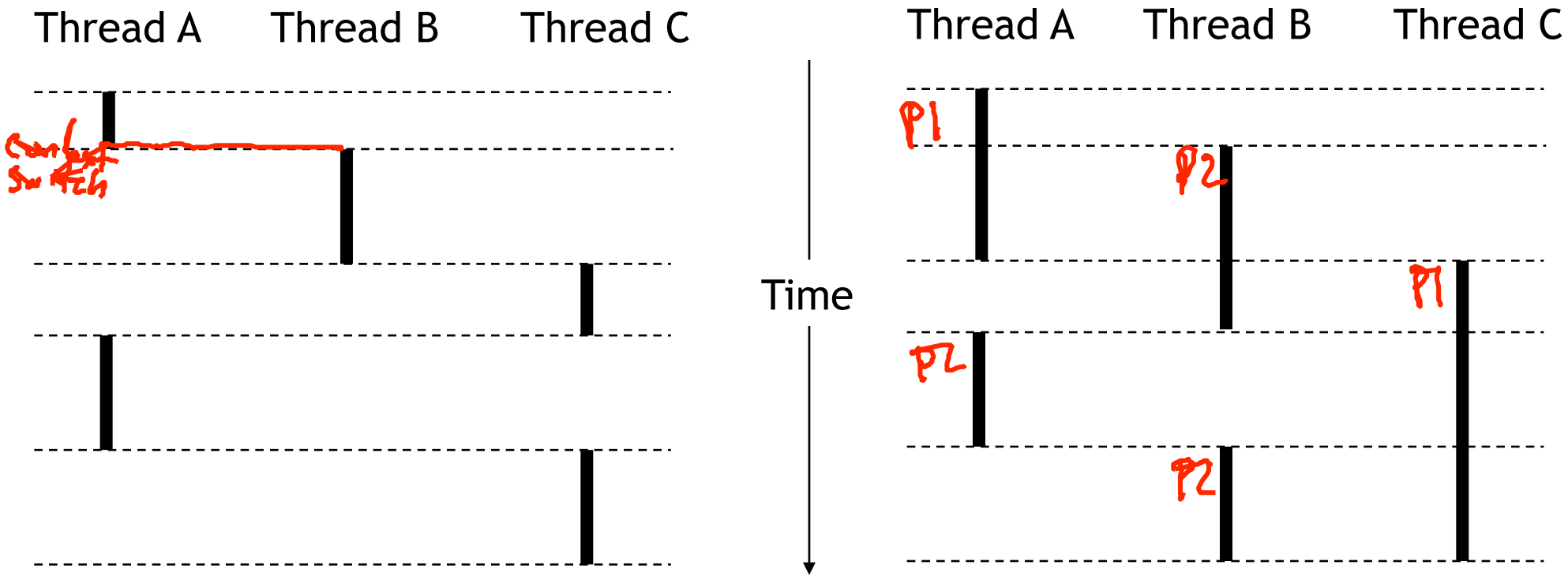
Process with Two Threads



©2012 Craig Zilles

Thread Execution

- Single Core Processor
  - Simulate concurrency by time slicing
- Multi-Core Processor
  - Can have true concurrency



Run 3 threads on 2 cores

©2012 Craig Zilles

How do threads from the same process communicate?

A) Their registers are kept in sync; when one thread writes a register, the written value is copied to the corresponding register in all cores that are running threads from the same process.

B) Threads have distinct register files, but can read and write each other register files directly.

C) Threads have distinct register files and must communicate through memory by one thread writing a memory location and later another thread reading that same memory location.

D) Threads have distinct registers and memory and must communicate through network protocols like TCP/IP.

E) telepathy

©2012 Craig Zilles

(Hardware) Shared Memory

- The programming model for multi-core CPUs

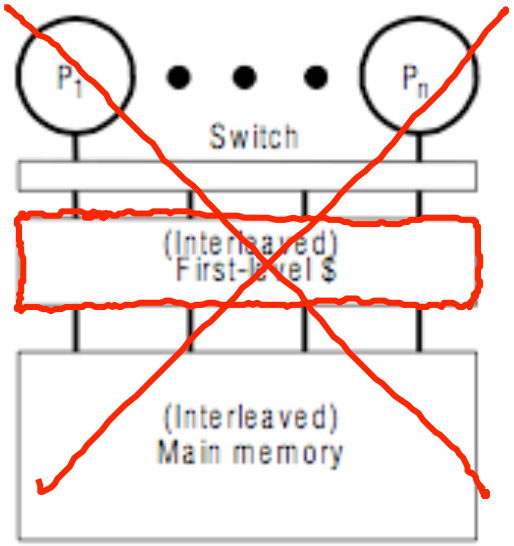
VA → PA

What is it?

- Memory system with a single global physical address space
  - So processors can communicate by reading and writing same memory
- Design Goal 1: Minimize memory latency
  - Use co-location & caches
- Design Goal 2: Maximize memory bandwidth
  - Use parallelism & caches

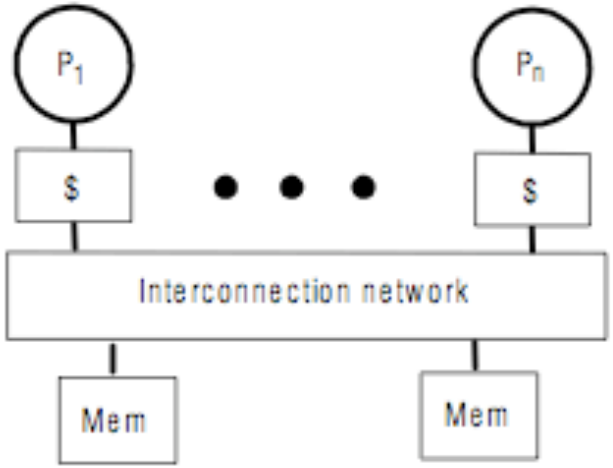
©2012 Craig Zilles

Some example memory-system designs

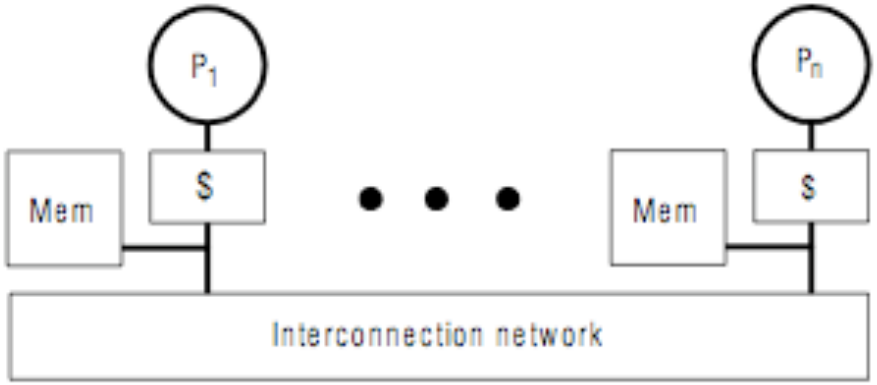


(a) Shared cache

(b) Bus-based shared memory



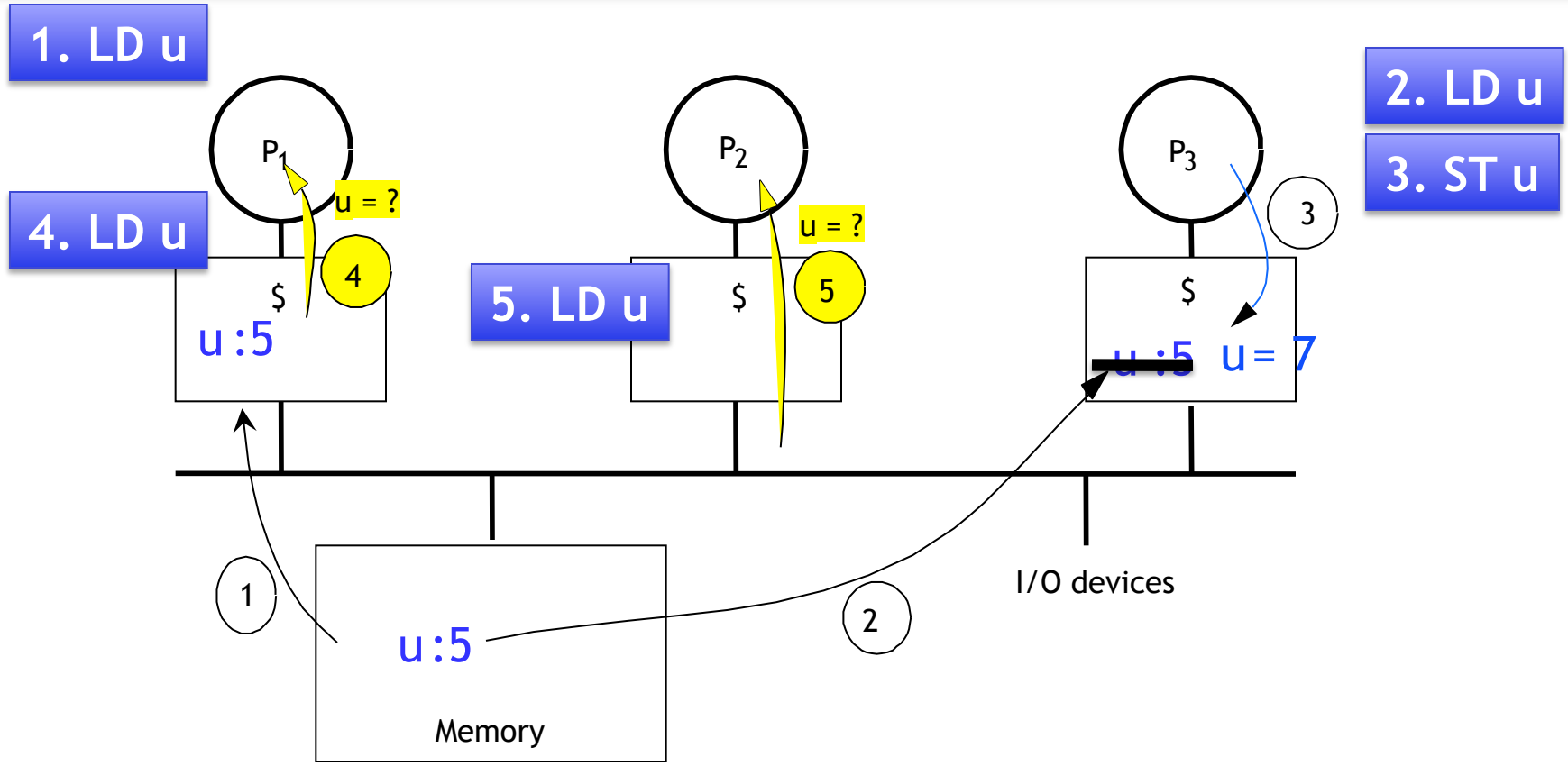
(c) Dancehall



(d) Distributed-memory

©2012 Craig Zilles

The Cache Coherence Problem



- Multiple copies of a block can easily get inconsistent
  - Processor writes; I/O writes
- Processors could see different values for u after event 3

©2012 Craig Zilles

Hw Cache Coherence

- According to Webster's dictionary …
  - Cache: a secure place of storage
  - Coherent: logically consistent

- **Cache Coherence:** keep storage logically consistent
  - Coherence requires enforcement of 2 properties

1. Write propagation
   - All writes eventually become visible to other processors
2. Write serialization
   - All processors see writes to same block in same order

P1 P2 P3 P4

x=2 x=9 x=? x=?

©2012 Craig Zilles

Cache Coherence Invariant

- Each block of memory is in exactly one of these 3 states:

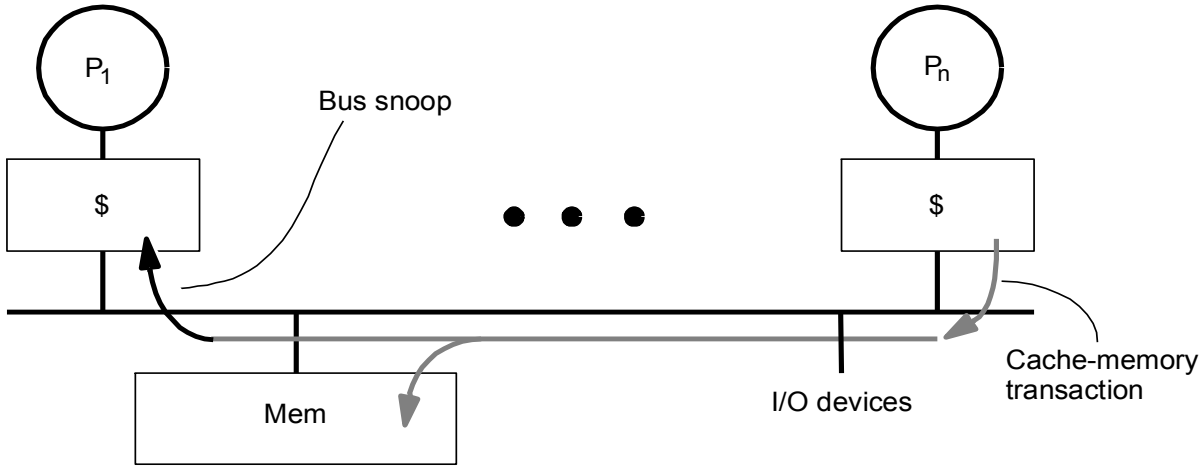
1. **Uncached**: Memory has the only copy
2. **Writable**: Exactly 1 cache has the block and only that processor can write to it.
3. **Read-only**: Any number of caches can hold the block, and their processors can read it.

only

invariant | inˈve(ə)rēənt |
noun Mathematics
a function, quantity, or property that remains unchanged when a specified transformation is applied.

©2012 Craig Zilles

Snoopy Cache Coherence Schemes



- Bus is a broadcast medium & caches know what they have

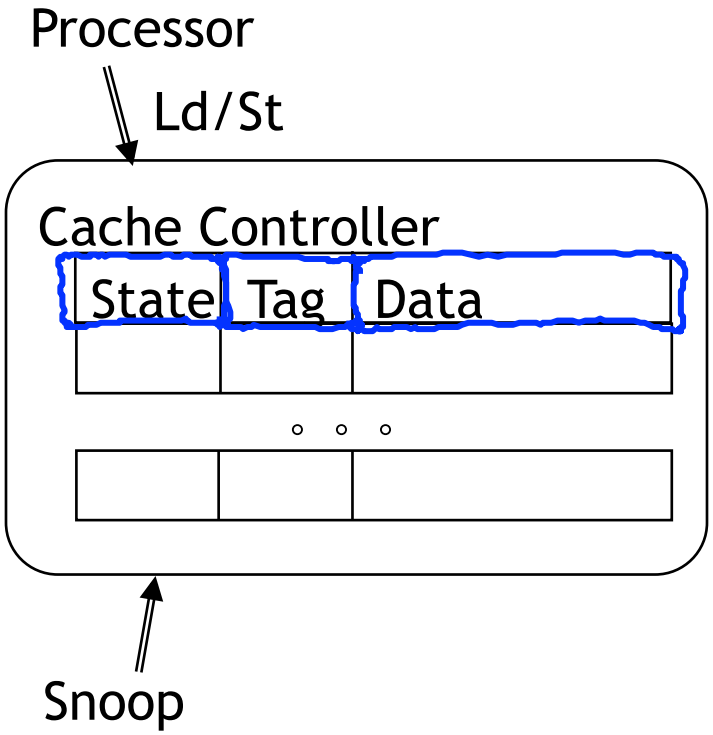
- Cache controller "snoops" all transactions on the shared bus
  - Relevant transaction if for a block it contains
  - Take action to ensure coherence
    - Invalidate or supply value

  Depends on state of the block and the protocol

©2012 Craig Zilles

Maintain the invariant by tracking "state"

- Every cache block has an associated state
  - This will supplant the valid and dirty bits
- A **cache controller** updates the state of blocks in response to processor and snoop events and generates bus transactions
- Snoopy protocol
  - set of states
  - state-transition diagram
  - actions



©2012 Craig Zilles

MSI protocol

This is the simplest possible protocol, corresponding directly to the 3 options in our invariant

- **Invalid State**: the data in the cache is not valid.
- **Shared State:** multiple caches potentially have copies of this data; they will all have it in **shared** state. Memory has a copy that is consistent with the cached copy.
- **Dirty or Modified:** only 1 cache has a copy. Memory has a copy that is inconsistent with the cached copy. Memory needs to be updated when the data is displaced from the cache or another processor wants to read the same data.

©2012 Craig Zilles

Actions

Processor Actions:

- **Load**
- **Store**
- **Eviction**: processor wants to replace cache block

Bus Actions:

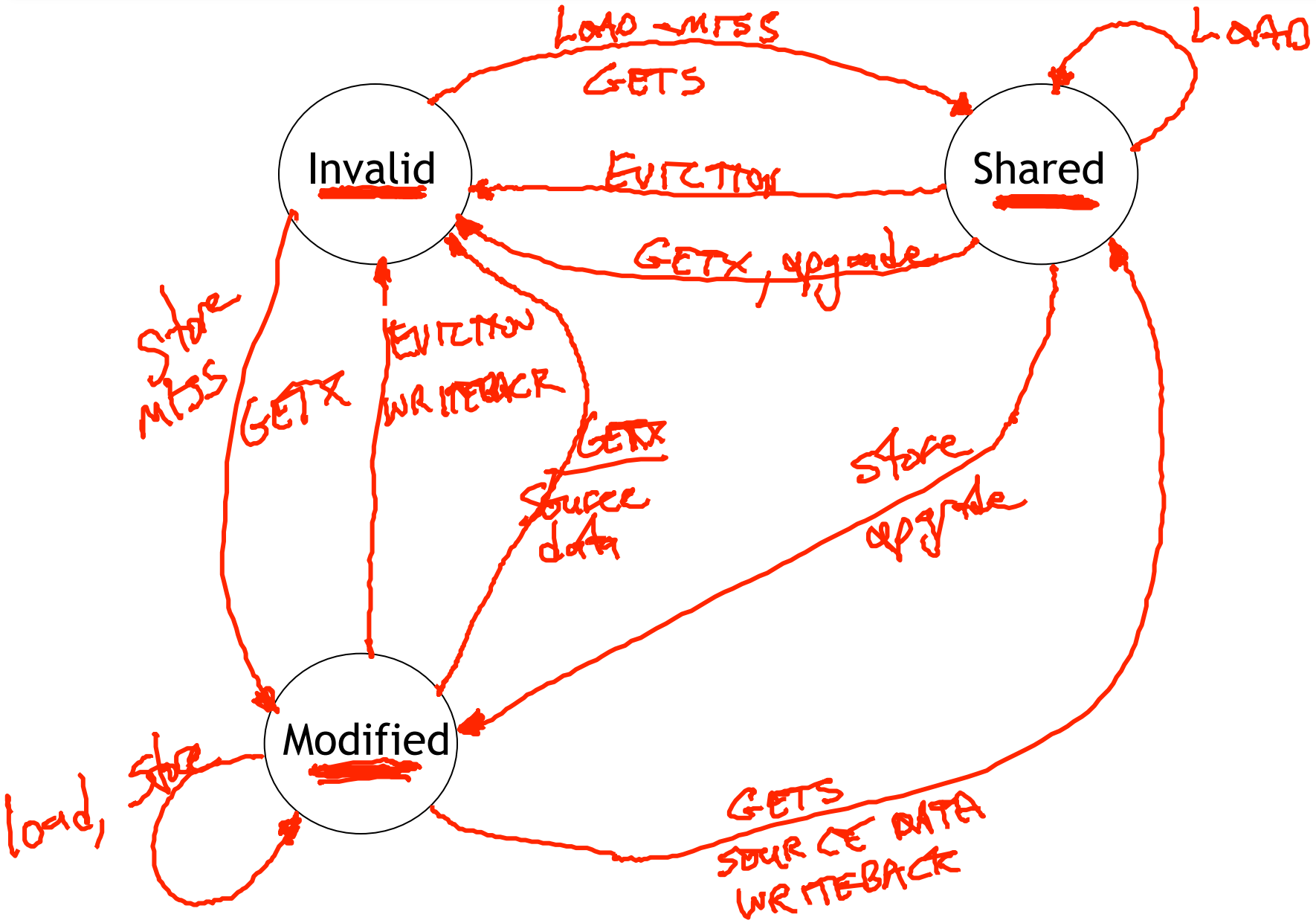
- **GETS**: request to get data in shared state
- **GETX**: request for data in modified state (*i.e.*, eXclusive access)
- **UPGRADE**: request for exclusive access to data owned in shared state

Cache Controller Actions:

- **Source Data**: this cache provides the data to the requesting cache
- **Writeback**: this cache updates the block in memory

©2012 Craig Zilles

MSI Protocol



©2012 Craig Zilles