Computer Science 425
Distributed Systems

CS 425 / ECE 428

Fall 2013

Indranil Gupta (Indy)

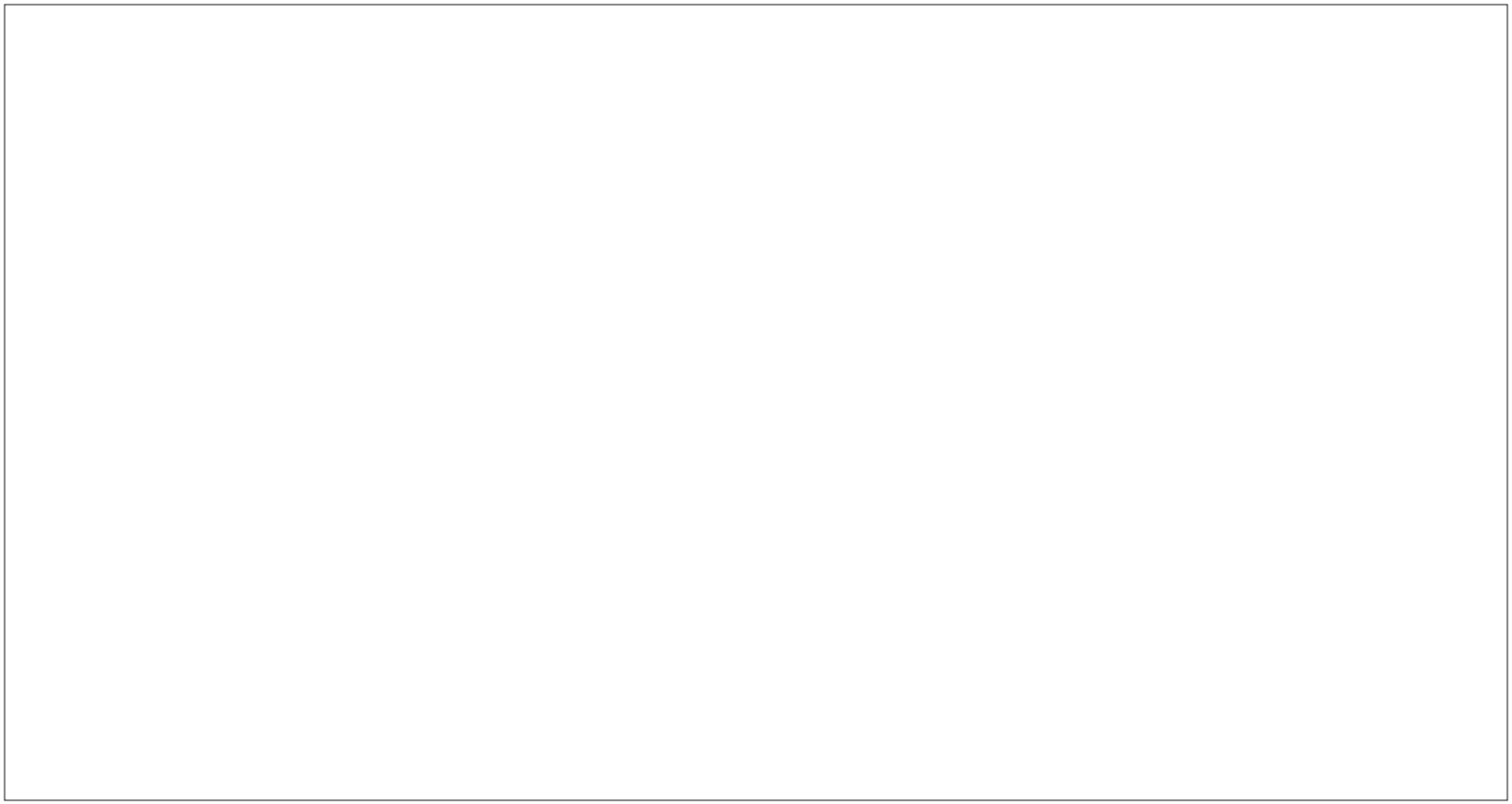
December 5, 2013

Lecture 28

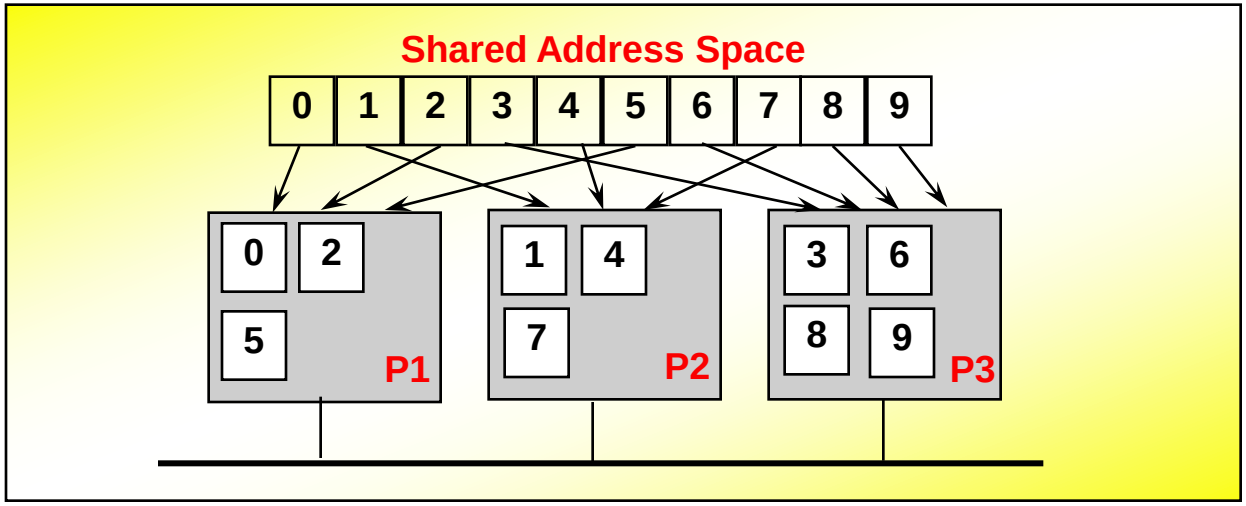
Distributed Shared Memory

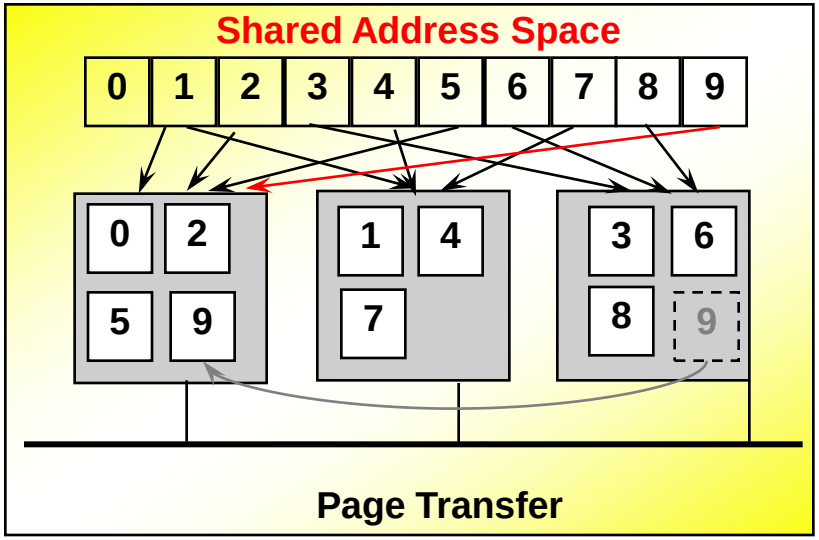
Section 6.5, Chapter 6 from Tanenbaum textbook

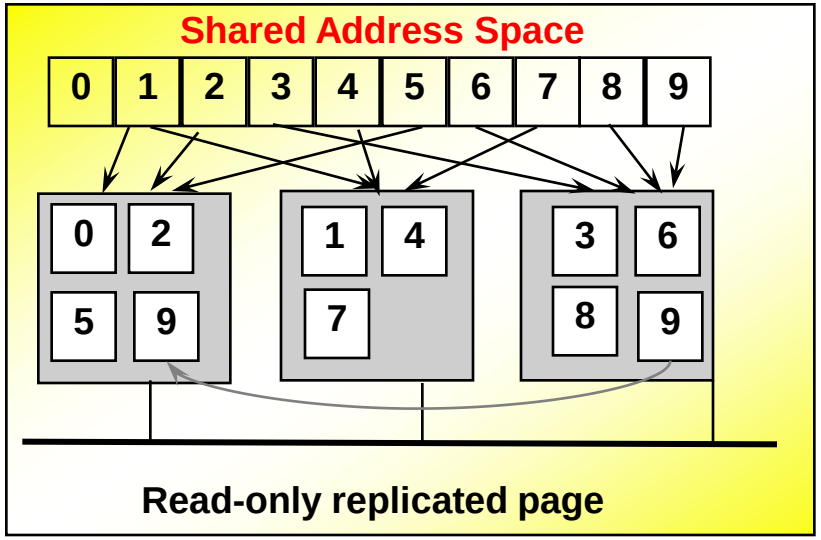
HW4 Due Now



Distributed Shared Memory

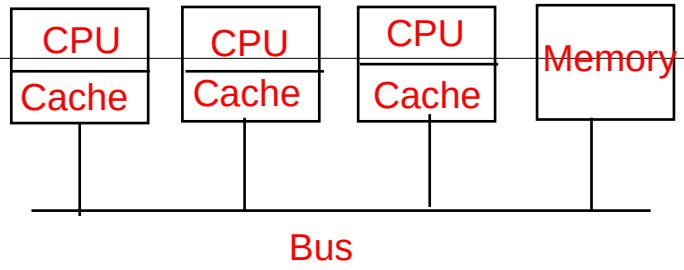




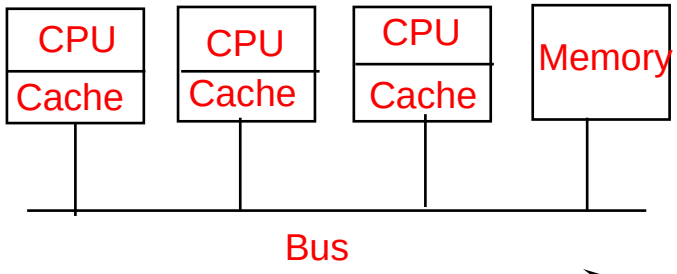


Shared Memory vs. Message Passing

- In a *multiprocessor*, two or more processors share a common main memory. Any process on a processor can read/write any word in the shared memory. All communication is through a bus.
  - E.g., Cray supercomputer
  - Called Shared Memory
- In a *multicomputer*, each processor has its own private memory. All communication using a network.
  - E.g., (Our favorite) EWS cluster. Or any datacenter really.
  - Easier to build: One can take a large number of single-board computers, each containing a processor, memory, and a network interface, and connect them together. (called COTS="Components off the shelf")
  - Uses Message passing
- Message passing can be implemented over shared memory.
- Shared memory can be implemented over message passing.
- *Let's look at shared memory by itself.*



Cache Consistency – Write Through

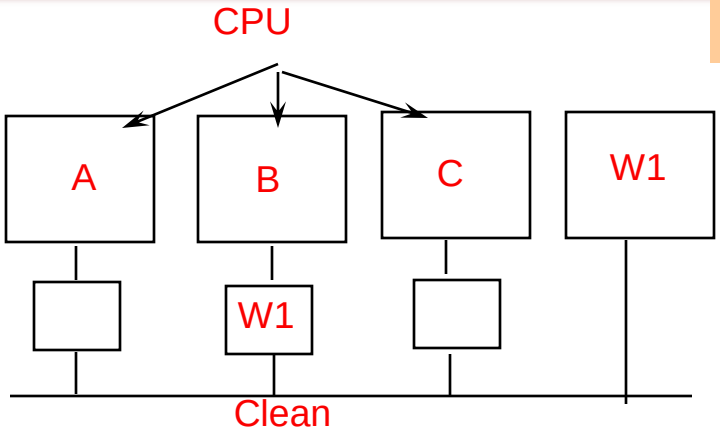


Event	Action taken by a cache in response to its own operation	Action taken by other caches in response (to a remote operation)
Read hit	Fetch data from local cache	(no action)
Read miss	Fetch data from memory and store in cache	(no action)
Write miss	Update data in memory and store in cache	Invalidate cache entry
Write hit	Update memory and cache	Invalidate cache entry

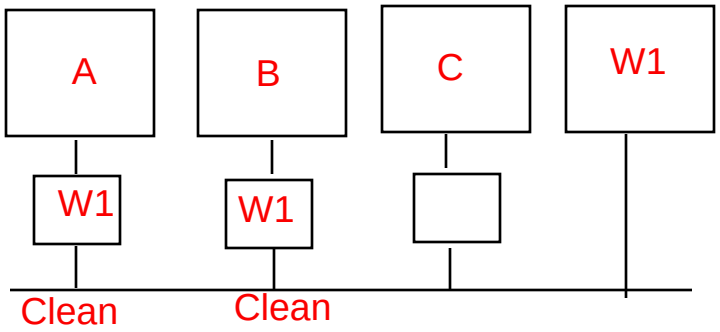
All the other caches see the write (because they are *snooping* on the bus) and check to see if they are also holding the word being modified. If so, they invalidate the cache entries.

Cache Consistency – Write Once

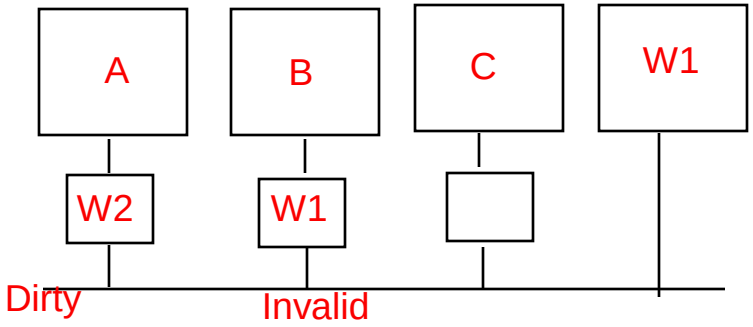
•For write, at most one CPU has valid access



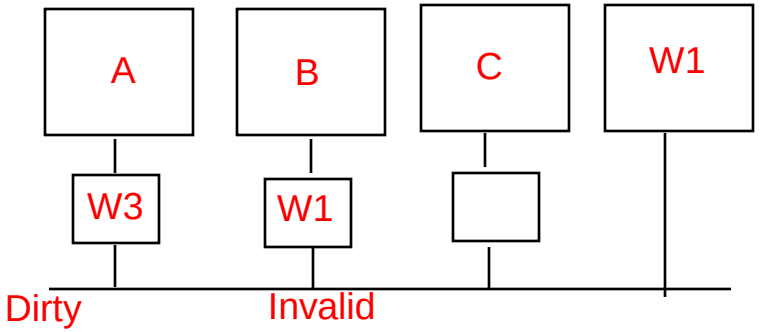
Initially both the memory and B have an updated entry of word W.



A reads word W and gets W1. B does not respond but the memory does

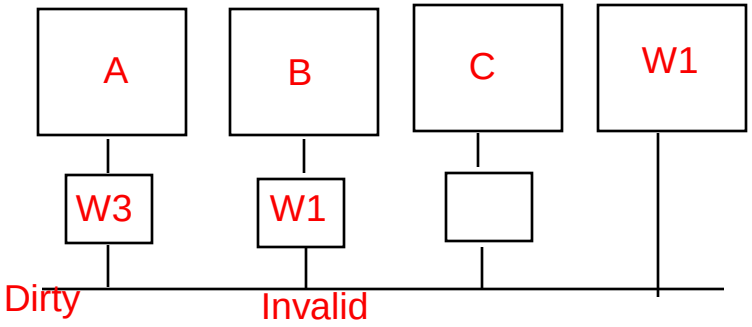


A writes a value W2. B snoops on the bus, and invalidates its entry. A′s copy is marked as dirty.

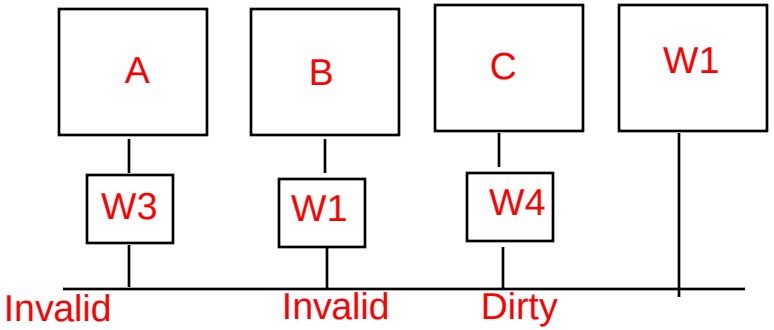


A writes W again. This and subsequent writes by A are done locally, without any bus traffic.

Cache Consistency – Write Once



A writes a value W3. No bus traffic is incurred



C writes W. A sees the request by snooping on the bus, asserts a signal that inhibits memory from responding, provides the values. A invalidates it own entry.
C now has the only valid copy.

The cache consistency protocol is built upon the notion of snooping and built into the memory management unit (MMU).
All above mechanisms are implemented in hardware for efficiency.

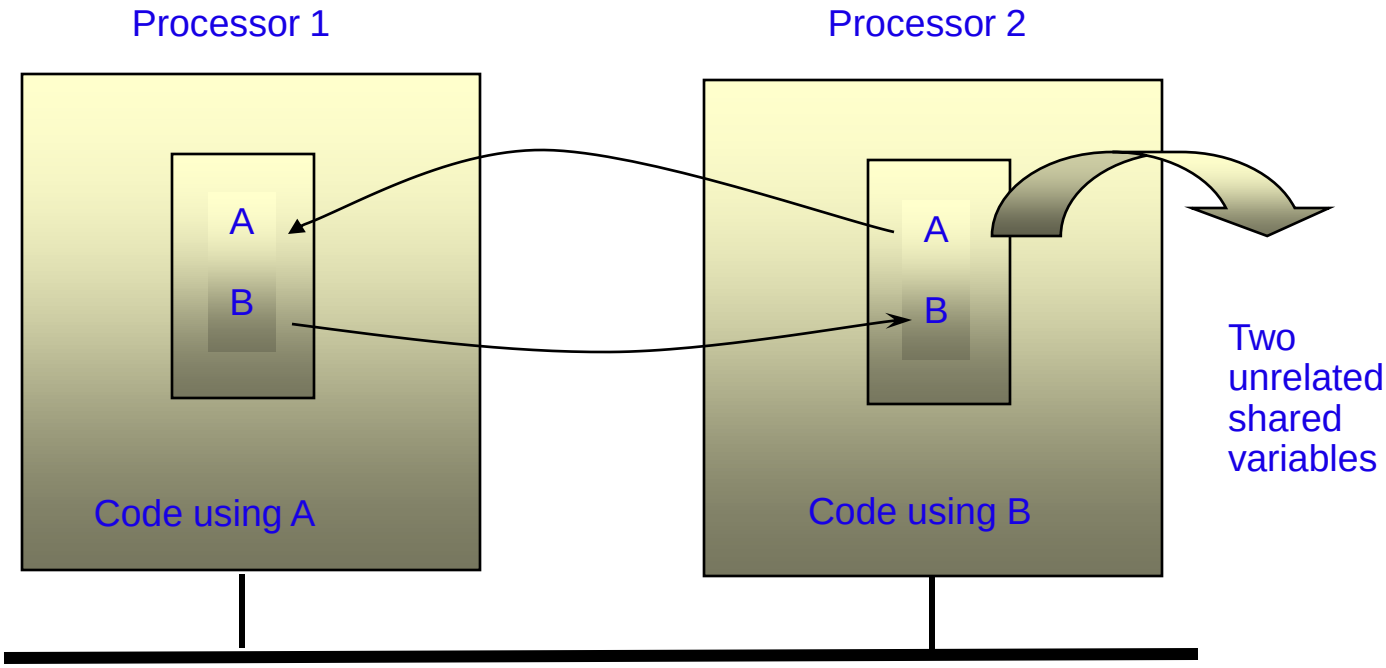
The above shared memory can be implemented using message passing instead of the bus.

Granularity of Chunks

- When a processor references a word that is absent, it causes a *page fault*.
- On a page fault,
  - the missing page is just brought in from a remote processor.
  - A *region* of 2, 4, or 8 pages including the missing page may also be brought in.
    - » Locality of reference: if a processor has referenced one word on a page, it is likely to reference other neighboring words in the near future.
- **Region size**
  - Small => too many page transfers
  - Large => *False sharing*
  - Above tradeoff also applies to page size

False Sharing

Page consists of two variables A and B



Occurs because: Page size > locality of reference
Unrelated variables in a region cause large number of pages transfers
Large page sizes => more pairs of unrelated variables

Achieving Sequential Consistency

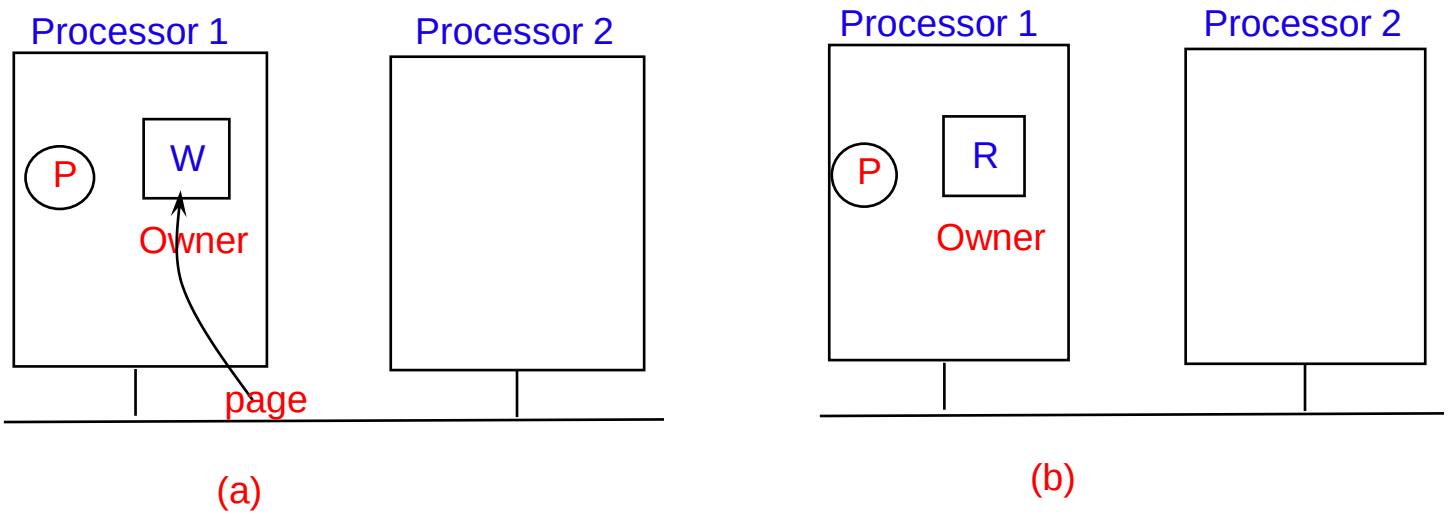
- **Achieving consistency is less challenging if:**
  - **Pages are not replicated, or…**
  - **Only read-only pages are replicated.**
- **But don't want to compromise performance.**
- **Two approaches are taken in Distributed Shared Memory (DSM)**
  - ***Invalidate*: The address of the modified word is broadcast, but the new value is not. Other processors invalidate their copies. (Similar to example in first few slides for multiprocessor)**
  - ***Update*: the write is allowed to take place locally, but the address of the modified word and its new value are broadcast to all the other processors. Each processor holding the word copies the new value, i.e., updates its local value.**

  Page-based DSM systems typically use an invalidate protocol instead of an update protocol. ? [Why?]

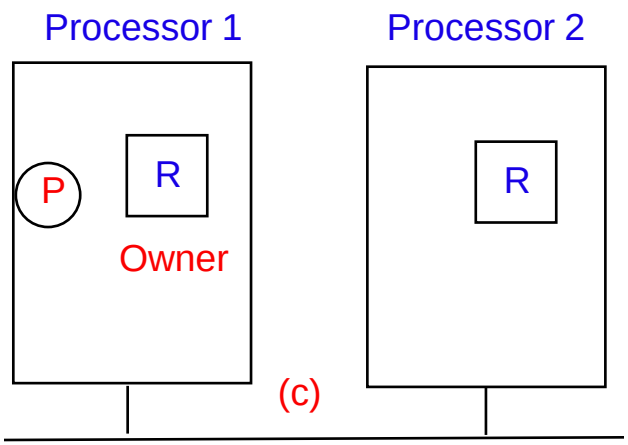
Invalidation Protocol to Achieve Consistency

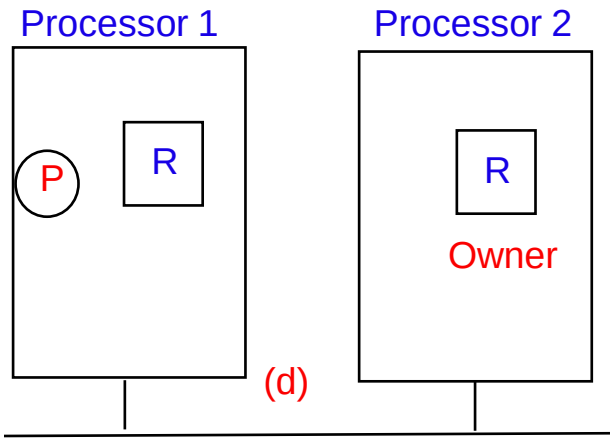
- **Owner = Processor with latest copy of page**
- **Each page is either in R or W state.**
  - **When a page is in W state, only one copy exists, located at one processor (called current "owner") in read-write mode.**
  - **When a page is in R state, the current/latest owner has a copy (mapped read-only), but other processors may have copies.**

Suppose Processor 1 is attempting a read: Different scenarios

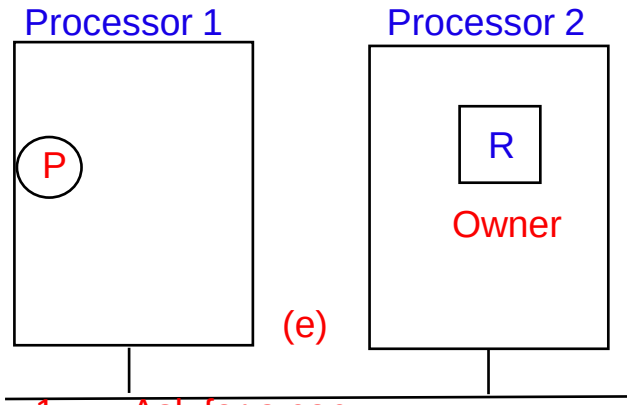


Invalidation Protocol (Read)

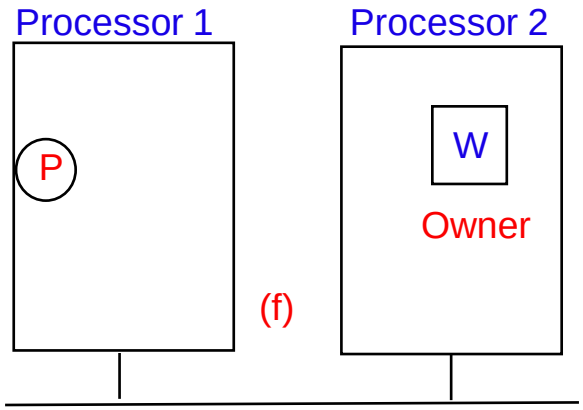




In the first 4 cases, the page is mapped into its address space, and no trap occurs.



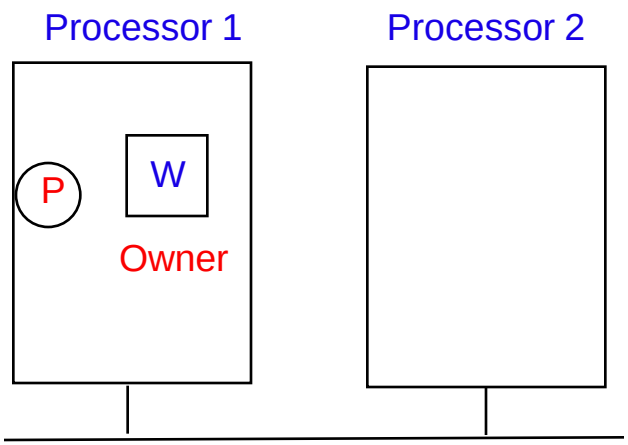
1. Ask for a copy
2. Mark page as R
3. Do read

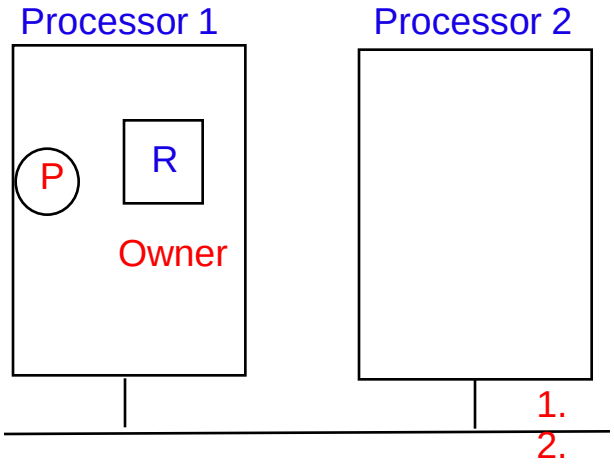


1. Ask P2 to degrade its copy to R
2. Ask for a copy
3. Mark page as R
4. Do read

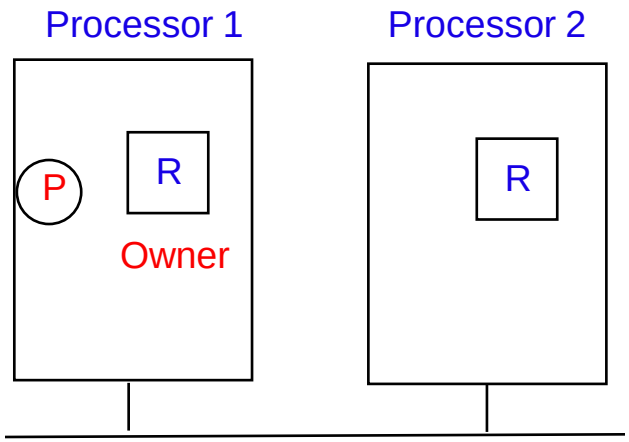
Invalidation Protocol (Write)

Suppose Processor 1 is attempting a write: Different scenarios

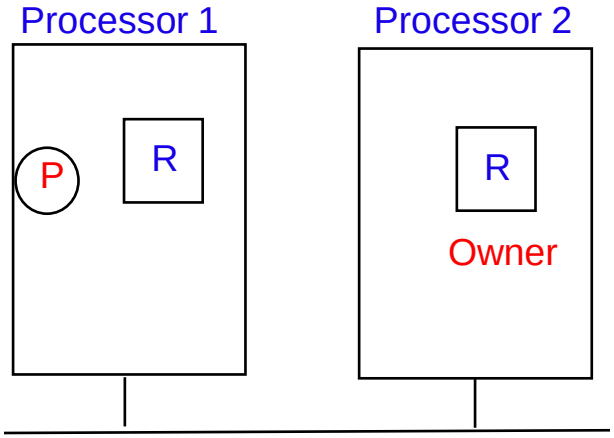




1. Mark page as W
2. Do write



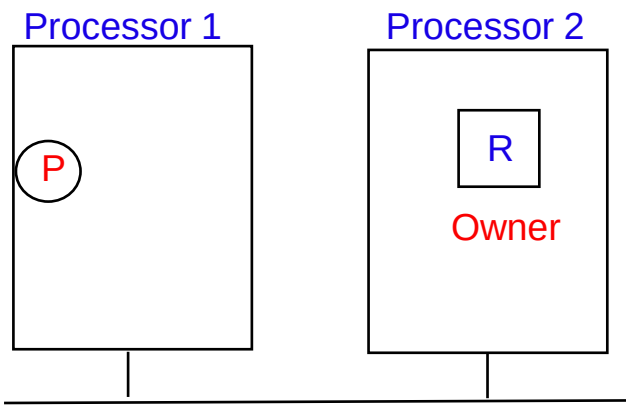
1. Invalidate other copies
2. Mark local page as W
3. Do write



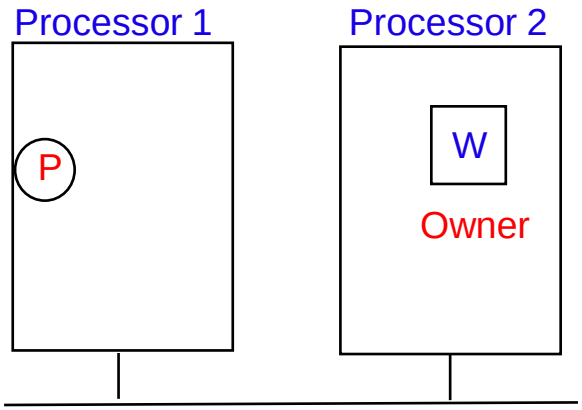
1. Invalidate other copies
2. Ask for ownership
3. Mark page as W
4. Do write

Invalidation Protocol (Write)

Suppose Processor 1 is attempting a write: Different scenarios



1. Invalidate other copies
2. Ask for ownership
3. Ask for a page
4. Mark page as W
5. Do write

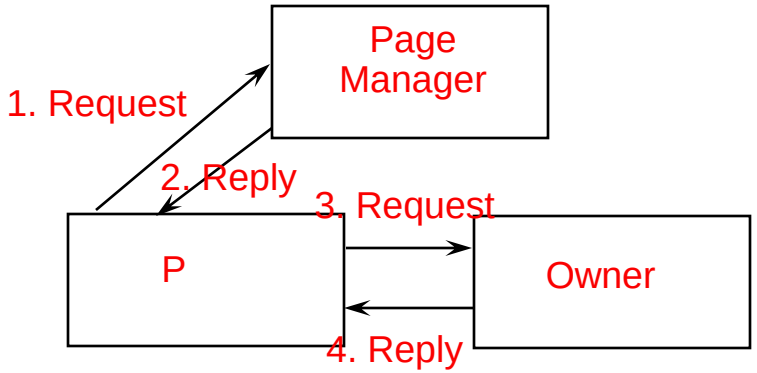


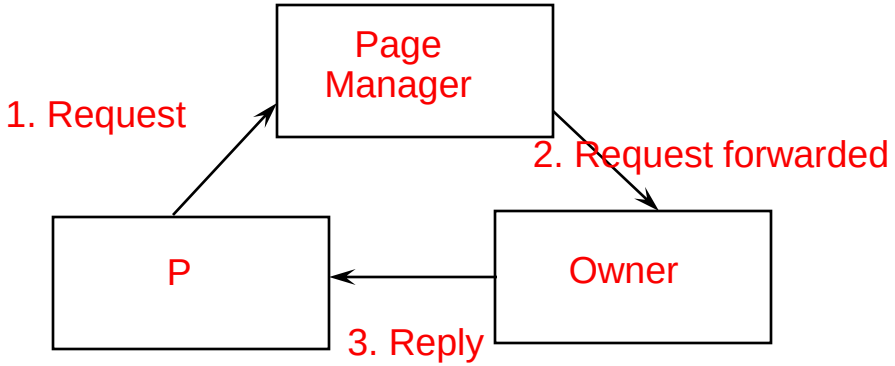
1. Invalidate other copies
2. Ask for ownership
3. Ask for a page
4. Mark page as W
5. Do write

Finding the Owner

- Owner is the processor with latest updated copy. How do you locate it?

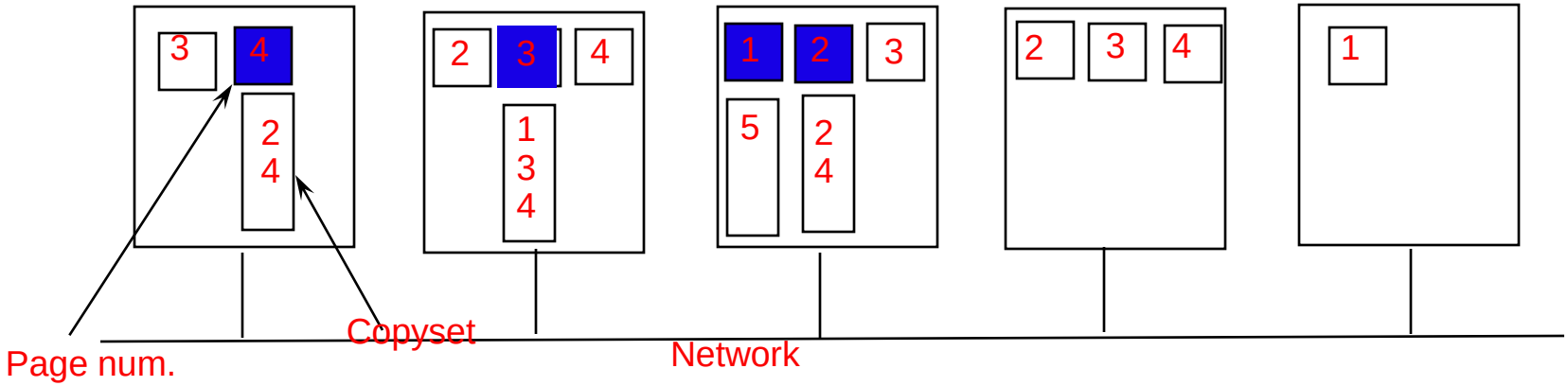
1. Do a broadcast, asking for the owner to respond.
   - Broadcast interrupts each processor, forcing it to inspect the request packet.
   - An optimization is to include in the message whether the sender wants to read/write and whether it needs a copy.
2. Designate a page manager to keep track of who owns which page.
   - A page manager uses incoming requests not only to provide replies but also to keep track of changes in ownership.
   - Potential performance bottleneck
   - Multiple page managers
     - » Map pages to page managers using the lower-order bits of page number.





How does the Owner Find the Copies to Invalidate

- **Broadcast a msg giving the page num. and asking processors holding the page to invalidate it.**
  - **Works only if broadcast messages are reliable and can never be lost. Also expensive.**
- **The owner (or page manager) for a page maintains a *copyset* list giving processors currently holding the page.**
  - **When a page must be invalidated, the owner (or page manager) sends a message to each processor holding the page (or a multicast) and waits for an acknowledgement.**



Strict and Sequential Consistency

- **Different types of consistency: a tradeoff between accuracy and performance.**
- **Strict Consistency (one-copy semantics)**
  - ***Any read to a memory location x returns the value stored by the most recent write operation to x.***
  - **When memory is strictly consistent, all writes are instantaneously visible to all processes and an *absolute global time order* is maintained.**
  - **Similar to "Linearizability"**
- **Sequential Consistency**

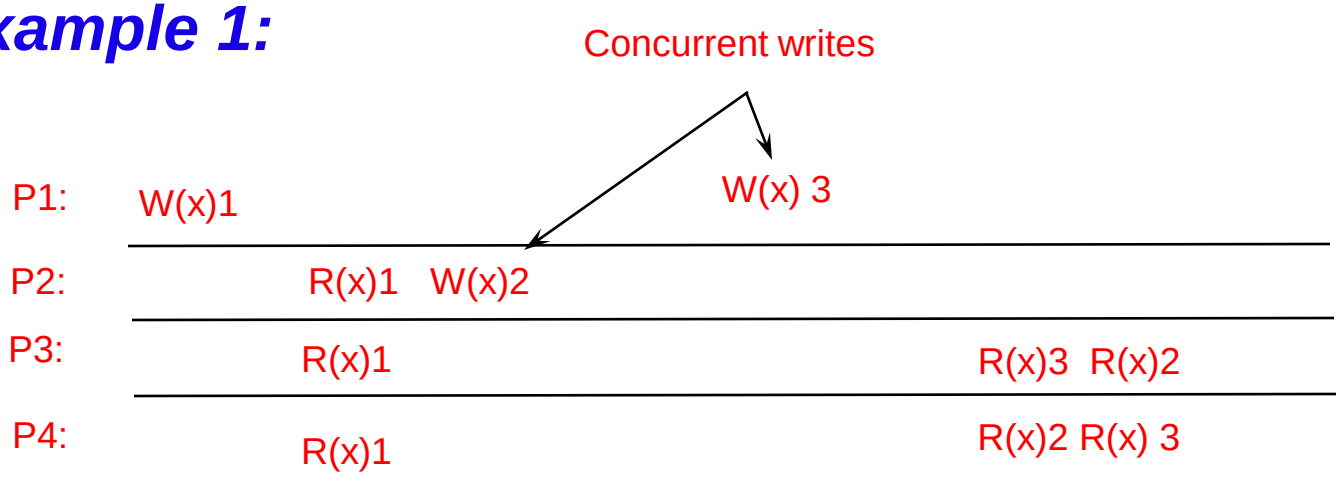
  For any execution, a sequential order can be found for all ops in the execution so that
  - ***The sequential order is consistent with individual program orders***
  - ***Any read to a memory location x should have returned (in the actual execution) the value stored by the most recent write operation to x in this sequential order.***

How to Determine the Sequential Order?

- *Example: Given H_1= W(x)1 and H_2= R(x)0 R(x)1, how do we come up with a sequential order (single string S of ops) that satisfies seq. cons.*
  - *Program order must be maintained*
  - *Memory coherence must be respected: a read to some location, x must always return the value most recently written to x.*
- *Answer: S= R(x)0 W(x)1 R(x)1*

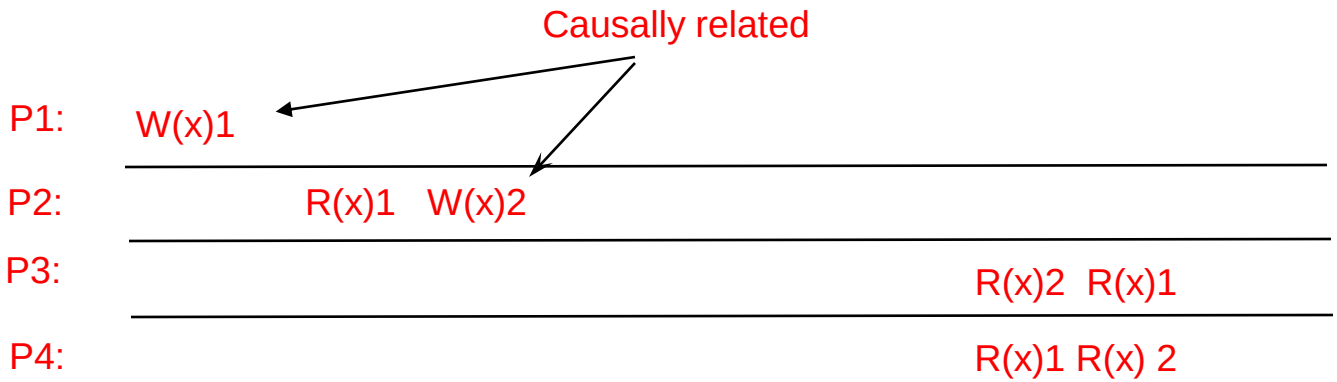
Causal Consistency

- ***Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.***
- ***Example 1:***

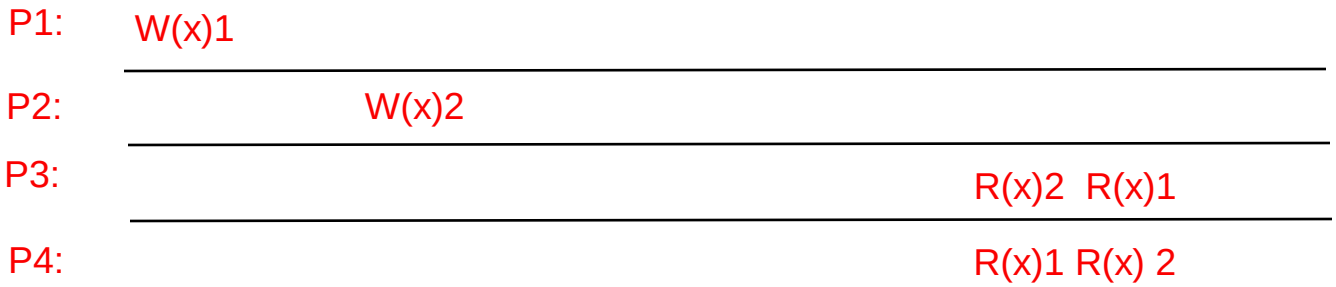


This sequence obeys causal consistency

Causal Consistency



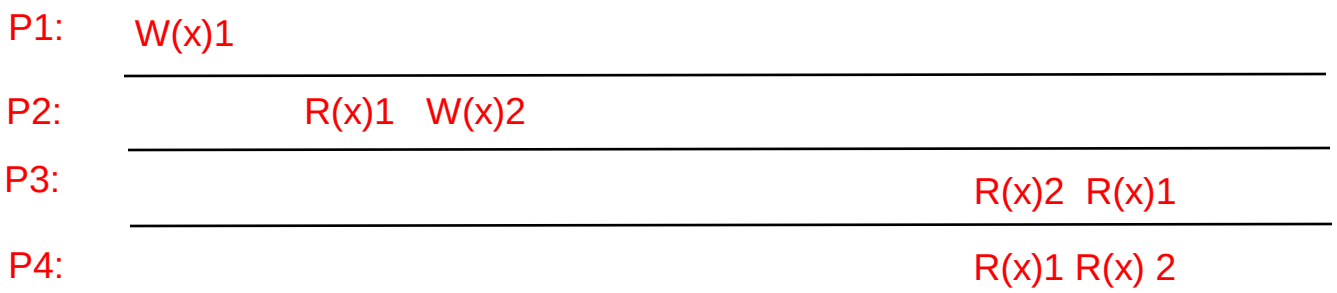
This sequence does not obey causal consistency



This sequence obeys causal consistency

Pipelined RAM Consistency

- ***Any pair of writes that are both done by a single processor are received by all other processors in the same order. A pair of writes from different processes may be seen in different orders at different processors.***



This sequence is allowed with PRAM consistent memory

Cache Coherence

- **Discussion so far focuses on single individual variable**
- **Sequential/causal/PRAM consistency needs to obey ordering across *all* variable accesses.**
- **But sometimes you only care for each variable independently**

P1: W(x)1 R(y)0

P2: W(y)1 R(x)0

- **Accesses to x and y can be linearized into R(x)0, W(x)1, and separately, R(y)0, W(y)1**
- **The history is coherent (per variable), but not sequentially consistent**

Weak Consistency

- **Not all the processors require seeing all the writes, let alone seeing them in order.**
  - **E.g, a process is inside a critical section reading/writing some variables in a tight loop. Other processes are not supposed to touch the variables until the first process has left the critical section.**
- **A special synchronization variable is introduced. When a synchronization completes, all writes done on that processor are propagated outward and all writes done on other processors are brought in.**
  - **Access to synchronization variables are sequentially consistent.**
  - **No access to a synchronization variable is allowed to be performed until all previous writes have completed everywhere else.**
  - **Accessing a synchronization variable "flushes the pipeline", both to and from this processor**
  - **No data access (read/write) is allowed until all previous accesses to synchronization variables have been performed.**

Weak Consistency

P1: W(x)1 W(y) 2 S

P2:

P3: R(y)2 R(x)0 S

P4: R(x)0 R(y) 2 S

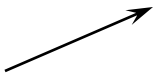
This sequence is allowed with weak consistent memory (but is not recommended)

P1: W(x)1 W(y) 2 S

P2:

P3: S R(y)2 R(x)1

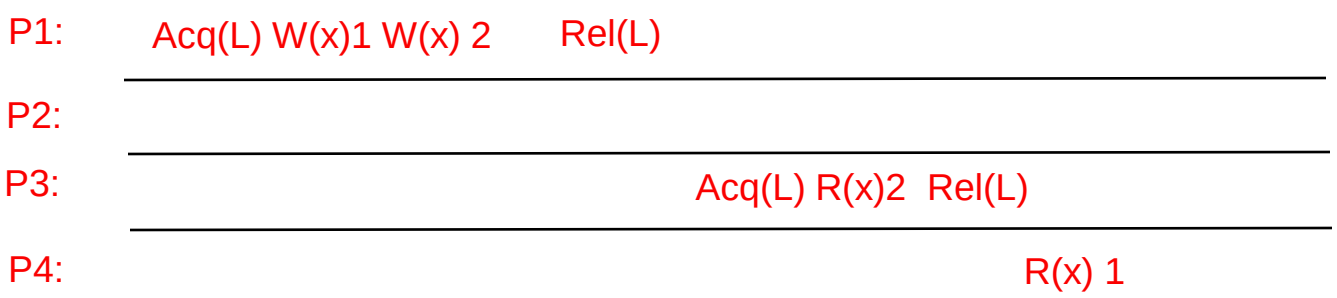
P4: S R(x)1 R(y) 2



The memory in P3 and P4 has been brought up to date

Release Consistency

- **Two special synchronization variables are defined: *Acquire* and *Release***
  - **Before any read/write op is allowed to complete at a processor, all previous acquire's at all processors must be completed**
  - **Before a release is allowed to complete at a processor, all previous read/write ops. must be propagated to all processors**
  - **Acquire and release ops are sequentially consistent w.r.t. one another**



This sequence is allowed with release consistent memory

Is that Bear dead?

- DSM is an old area, and is less active research-wise.
  - In other words, many say it is “dead”
- However, faster networks in datacenters
  - Infiniband and CLOS

    =>
  - RDMA: Remote Direct Memory Access
  - Servers can read each others’ memories
- Time to rejuvenate DSM?
- Also, many consistency model ideas in key-value/NoSQL storage systems originate from DSM
- Is that Bear really starting to wake up from hibernation?

Announcements

- **MP4 due this Sunday. Demos next Monday (watch Piazza/wiki for signup sheet)**
- Mandatory to attend next Tuesday's lecture (last lecture of course)

- **Final Exam, December 19 (Thursday), 7.00 PM - 10.00 PM**
  - **David Kinley Hall – 114 (1DKH-114)**
  - **1407 W. Gregory Drive, Urbana IL 61801**
  - **Do not come to our regular Siebel classroom!**
  - **Also on website schedule**
  - **Allowed to bring a cheat sheet: two sides only, at least 1 pt font**
- **Conflict exam**
  - Please email course staff email by this Thursday (Dec 5) if you need to take a conflict exam