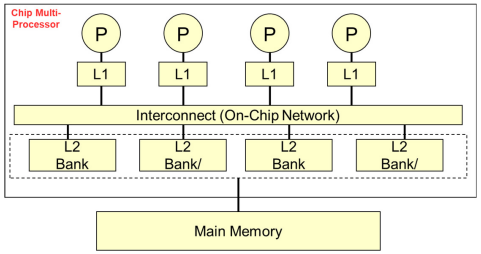
CS356 Unit 14

Coherency

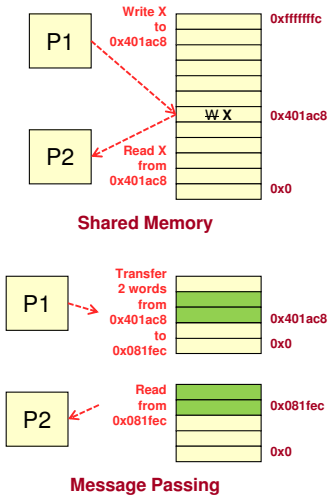
Multicores

- Rather than overlapping and parallelizing instructions from one thread on one core, another approach is to just have multiple processor cores
  - Each core executes separate threads
  - Threads may be from _____ and working on _____ data
  - When sharing data, caching issues arise!
  - Let's explore these issues



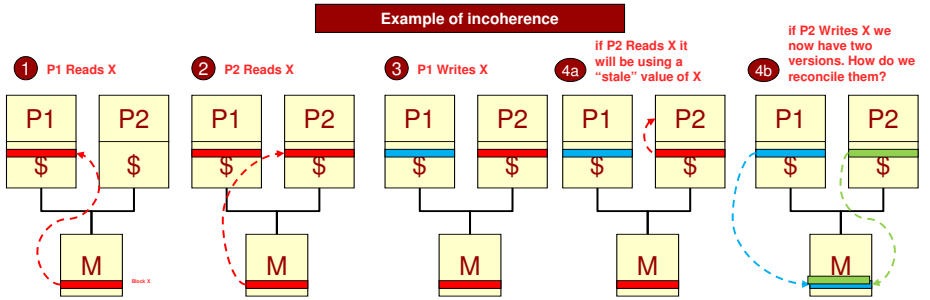
Communication Paradigms

- Shared Memory (common for multicores)
  - Each processor shares the same memory via a shared _____
  - Communication is _____ (implicitly) handled by the HW.
    SW program just has to perform normal loads and stores to shared memory locations (i.e. _____ **variables, or pointers to the same variable**)
  - Coherence of data becomes an issue
- Message Passing
  - Each processor has its own memory/address space
  - Communication must be explicitly defined by the SW program/processor (either by DMA or some other mechanism)



Cache Coherency

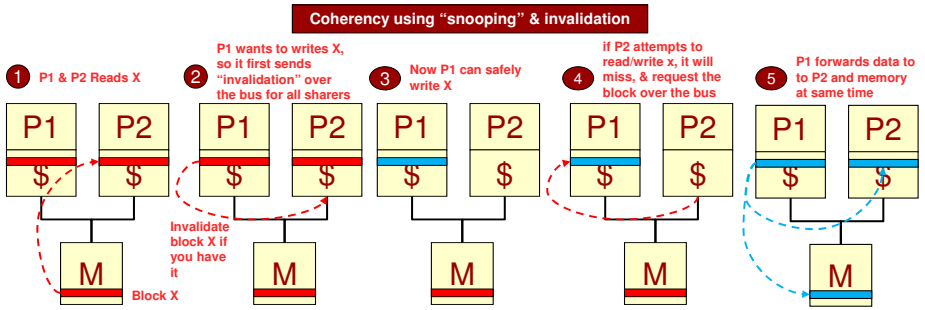
- Problem: Multiple cached copies of same memory block may lead to _____ problems (think 'git' merge conflicts)
  - Each processor can get their own copy, _____ it, and perform calculations on their own _____ values…INCOHERENT!
- Solution: Snoopy/snooping caches…



Snoopy or Snoopy

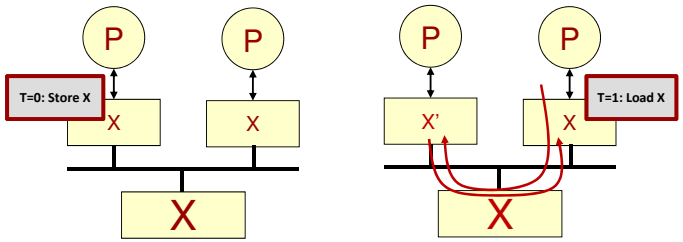
Solving Cache Coherency

- If no writes, multiple copies are fine
- Two options: When a block is modified
  - Go out and _____ else's copy
  - _____ all other sharers and make them come back to you to get a fresh copy
- _____ caches using invalidation policy is most common
  - Caches _____ on the bus looking for invalidation messages
  - If another cache needs a block you have the latest version of, forward it to mem & others



Coherence Definition

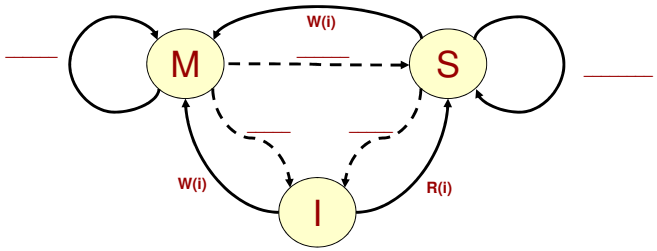
- A memory system is coherent if the **value returned on a Load instruction is always the value given by the _____ instruction by any processor to the same address**
- To implement this ability we need a protocol (set of rules) to track the "state" of each cache block



Write Back Cache Coherency Protocols

- Write invalidate protocols ("Ownership Protocols")
- Basic 3-state (MSI) Protocol
  - I = _____: Not in cache or invalidated earlier
  - RO (Read-Only) = _____: Processors has a valid block but so might other caches. Thus it is only safe to read their copy (but not write to it)
  - RW (Read-Write) = _____: Processors has modified (written) to the block and it is guaranteed to have the only copy (no other caches have a copy). Thus it is only safe to read or write.

Write Invalidate Snoopy Protocol

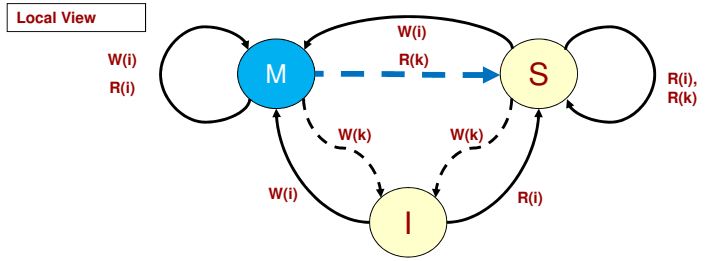


- R(i) = Read by the local processor
- W(i) = Write by the local processor
- R(k) = Read by some remote processor, k
- W(k) = Write by some remote processor, k

ISCA '90 Tutorial "Memory System Architectures for Tightly-coupled Multiprocessors", Michel Dubois and Faye A. Briggs © 1990.

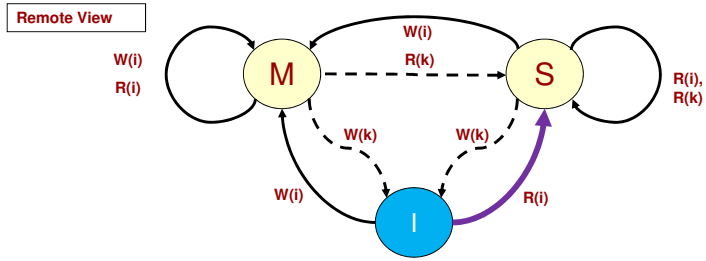
Remote Read

Local View



If you have the only copy and another processor wants to read the data

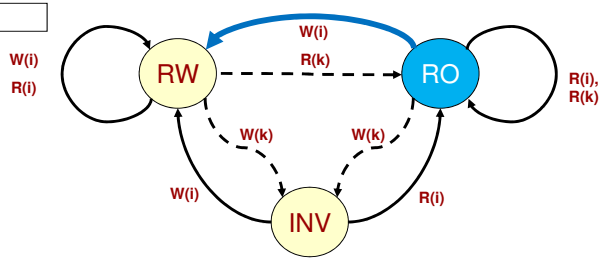
Remote View



The other processor goes from invalid to read-only

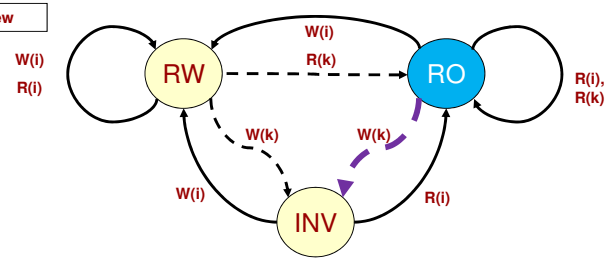
Local Write

Local View



Upgrade your access

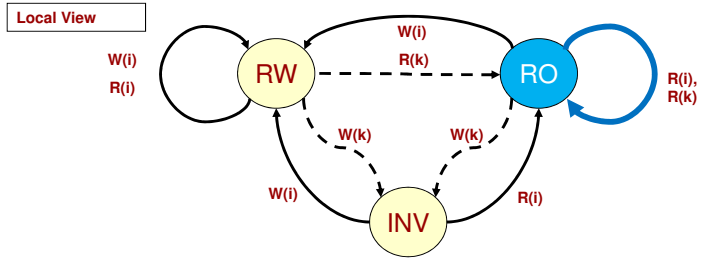
Remote View



Invalidate others' copy so no one else has the block

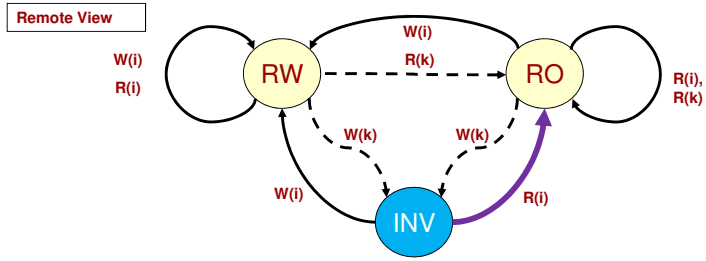
Remote Read

Local View



No change

Remote View



Remote processor gets a copy too

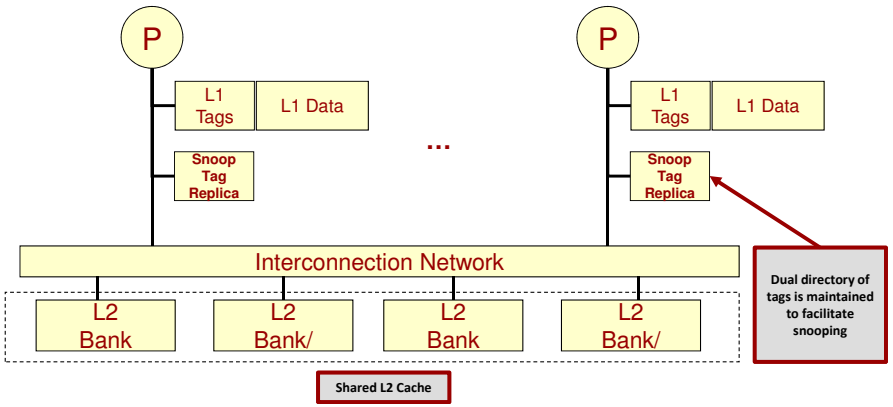
Coherency Example

Processor Activity	P1 $ Content	P1 Block State (M,S,I)	P2 $ Content	P2 Block State (M,S,I)	Memory Contents
	-	-	-	-	A
P1 reads block X					
P2 reads block X					
P1 writes block X=B					
P2 reads block X					

Another Coherency Example

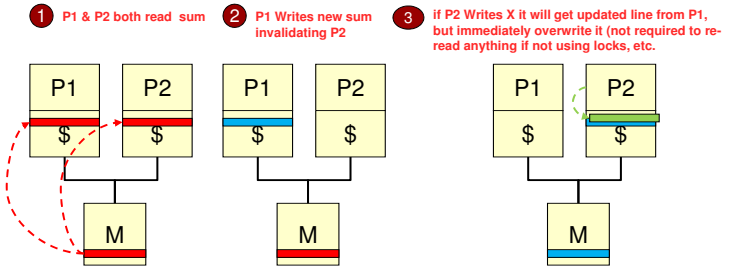
Processor Activity	P1 $ Content	P1 Block State (M,S,I)	P2 $ Content	P2 Block State (M,S,I)	Memory Contents
	-	-	-	-	A
P1 reads block X					
P1 writes X=B					
P2 writes X=C					
P1 reads block X					

Coherence Implementation



Is Cache Coherency = Atomicity?

- Does coherency take the place of locking/synchronization?
- ___, cache coherence only serializes _____ and does not serialize entire _____ sequences
  - Coherence simply ensures two processors don't read two different values of the same memory location
- Consider two threads performing: `sum += thread_val`



False Sharing

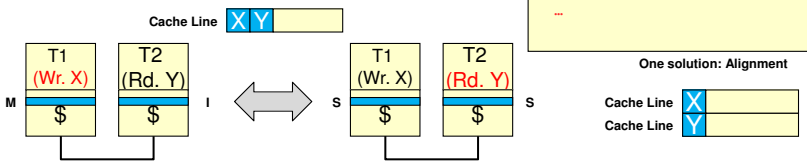
- Thread-independent (i.e. non-shared) variables allocated on the same _____
- Can cause a large performance _____ due to cache coherence (invalidates, etc.)

```
int x = 0;
int y = 0;
void t1() {
  for(int x=N; x > 0; x--)
  { }
  y = 1;
  ...
}
void t2() {
  while( y == 0);
  { }
  printf("Y was set to 1\n");
}
```

False Sharing Example

```
int x = 0;
int y __attribute__ ((aligned (64))) = 0;
  …
```

One solution: Alignment



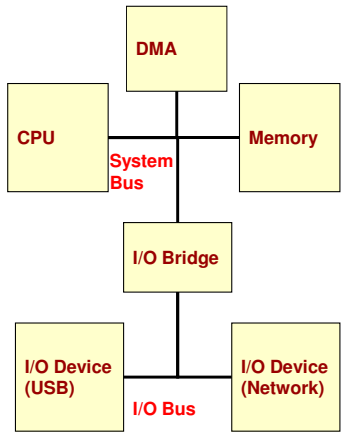
Locks and Contention

- The more threads compete for a lock, the slower performance will be
  - Continuous sequence of invalidate, get exclusive access to check lock, see it is already taken, repeat
- Options
  - Use special locks (e.g. queuing locks where a thread takes a number and waits until it is called rather than continuously checking if the lock is free)
  - Lock Granularity: Use separate locks for each element in a data structure rather than the one lock for the whole structure
  - Others that you will explore in CS350

DMA ENGINES

Direct Memory Access (DMA)

- Large buffers of data often need to be copied between:
  - Memory and I/O (video data, network traffic, etc.)
  - Memory and Memory (OS space to user app. space)
- DMA devices are small hardware devices that copy data from a source to destination freeing the processor to do "real" work

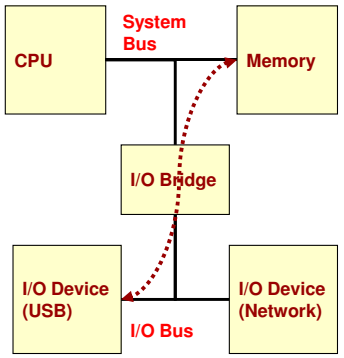


Data Transfer w/o DMA

- Without DMA, processor would have to move data using a loop
- Move 16Kwords pointed to by (%esi) to (%edi)

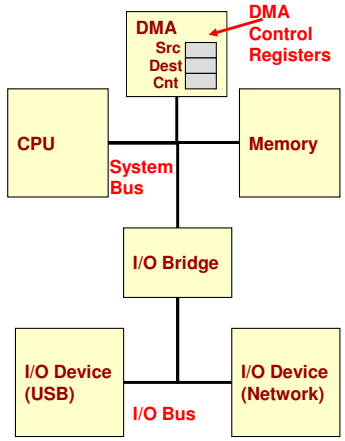
```
        movl   $16384,%ecx
AGAIN:  movl   (%esi),%eax
        movl   %eax,(%edi)
        add    $4,%esi
        add    $4,%edi
        sub    $1,%ecx
        jnz    AGAIN
```

- Processor wastes valuable execution time moving data



Data Transfer w/ DMA

- Processor sets values in DMA control registers
  - Source Start Address
  - Dest. Start Address
  - Byte Count
  - Control & Status (Start, Stop, Interrupt on Completion, etc.)
- DMA becomes "bus-master" (controls system bus to generate reads and writes) while processor is free to execute other code
  - Small problem: Bus will be busy
  - Hopefully, data & code needed by the CPU will reside in the processor's cache



DMA Engines

- Systems usually have multiple DMA engines/channels
- Each can be configured to be started/controlled by the processor or by certain I/O peripherals
  - Network or other peripherals can initiate DMA's on their behalf
- Bus arbiter assigns control of the bus
  - Usually winning requestor has control of the bus until it relinquishes it (turns off its request signal)