PARALLEL MEMORY ARCHITECTURE

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

- Announcement
  - Homework 7 is due tomorrow midnight
    - The last one! ☺
- This lecture
  - Communication in multiprocessors
  - Parallel memory architecture
  - Cache coherence protocol

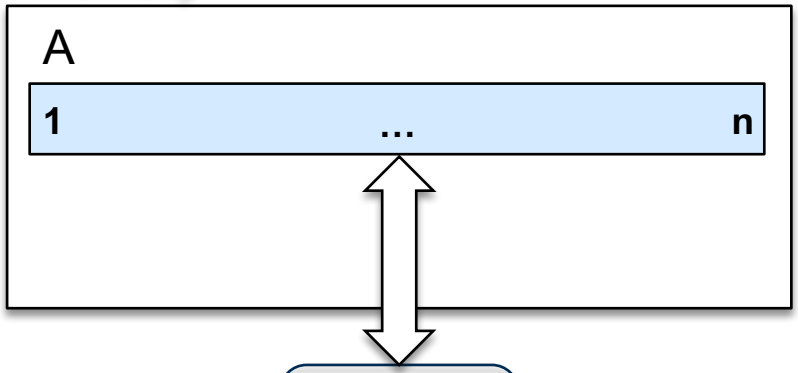
Example Code I

- A sequential application runs as a single thread

Kernel Function:

```
void kern (int start, int end) {
   int i;
   for(i=start; i<=end; ++i) {
      A[i] = A[i] * A[i] + 5;
   }
}
```

Memory



Processor

Single Thread

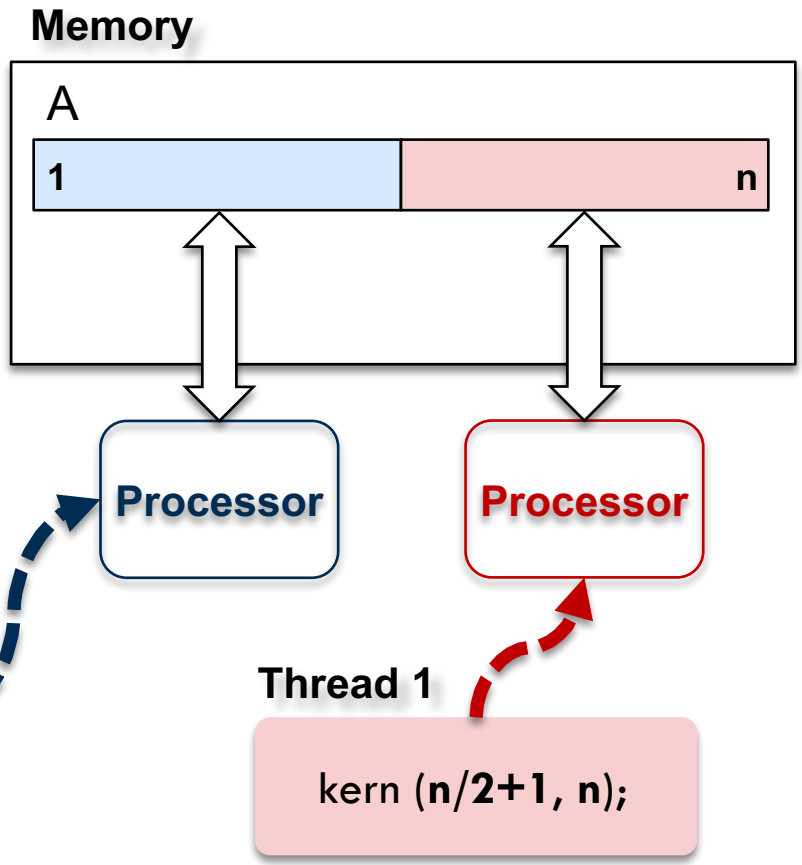
```
main() {
   …
   kern (1, n);
   …
}
```

Example Code I

- Two threads operating on separate partitions

Kernel Function:

```
void kern (int start, int end) {
   int i;
   for(i=start; i<=end; ++i) {
      A[i] = A[i] * A[i] + 5;
   }
}
```

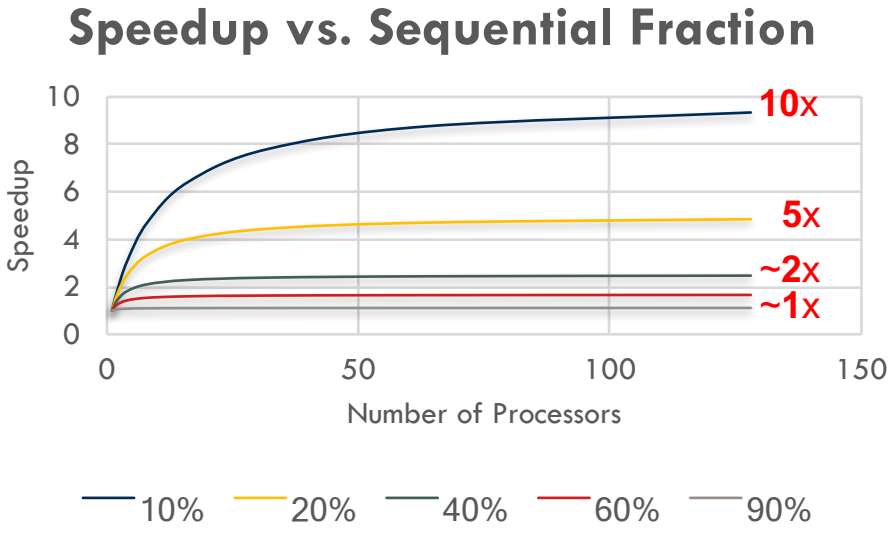


Performance of Parallel Processing

- Recall: Amdahl's law for theoretical speedup
  - Overall speedup is limited to the fraction of the program that can be executed in parallel

$$speedup = \frac{1}{f + \frac{1-f}{n}}$$

f: sequential fraction



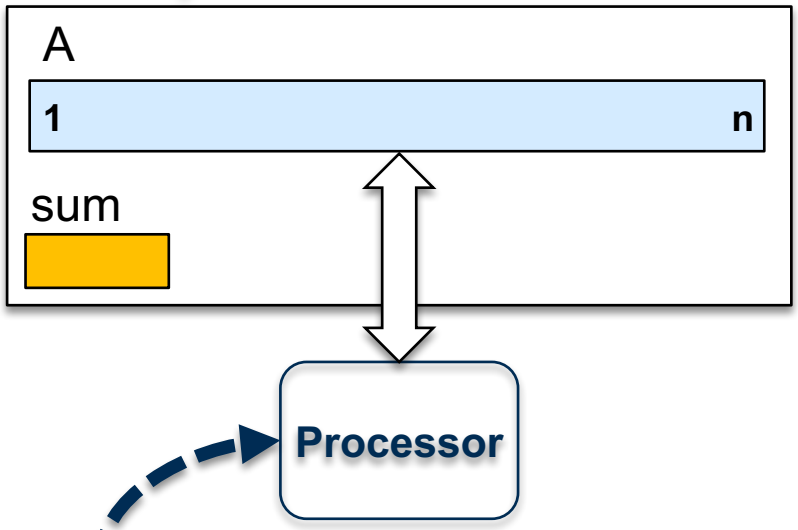
Example Code II

- A single location is updated every time

Kernel Function:

```
void kern (int start, int end) {
   int i;
   for(i=start; i<=end; ++i) {
      sum = sum * A[i];
   }
}
```

Memory



Thread 0

```
main() {
   …
   kern (1, n);
   …
}
```

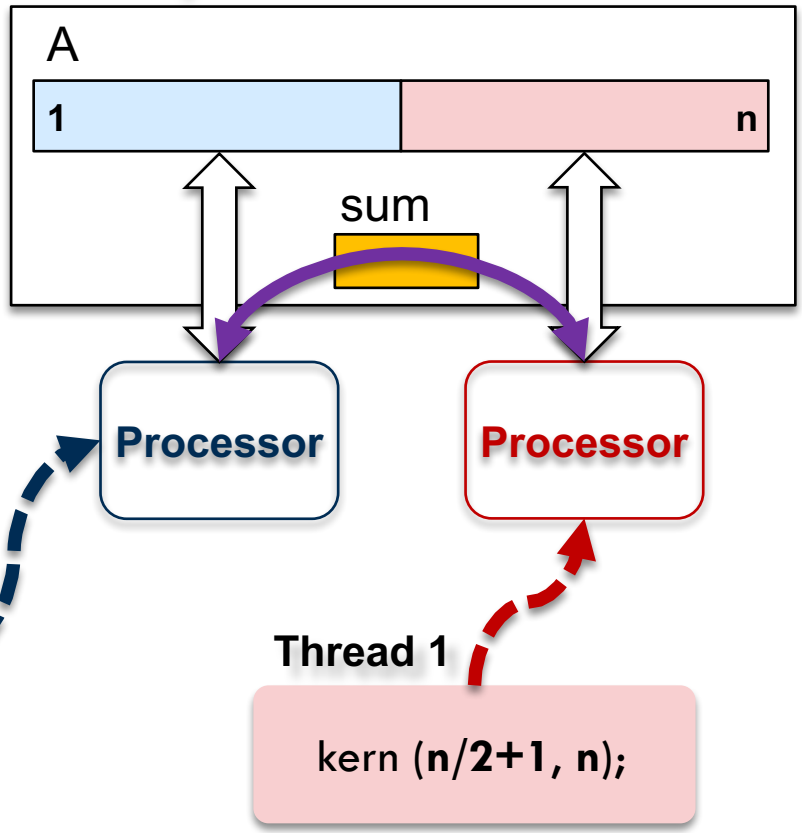
Example Code II

- Two threads operating on separate partitions

Kernel Function:

```
void kern (int start, int end) {
   int i;
   for(i=start; i<=end; ++i) {
      sum = sum * A[i];
   }
}
```

Memory

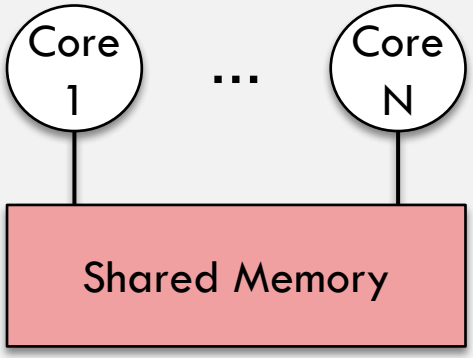


Communication in Multiprocessors

- How multiple processor cores communicate?

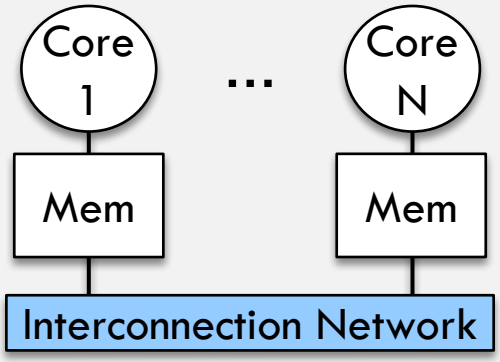
Shared Memory

- Multiple threads employ shared memory
- Easy for programmers (loads and stores)



Message Passing

- Explicit communication through interconnection network
- Simple hardware

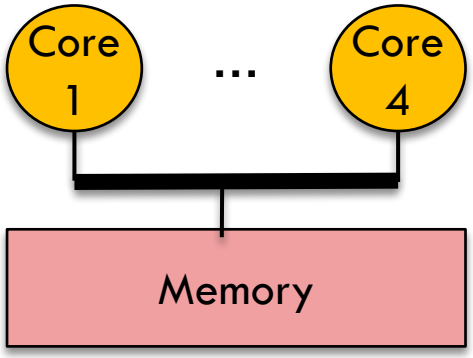


Shared Memory Architectures

Uniform Memory Access

- Equal latency for all processors
- Simple software control

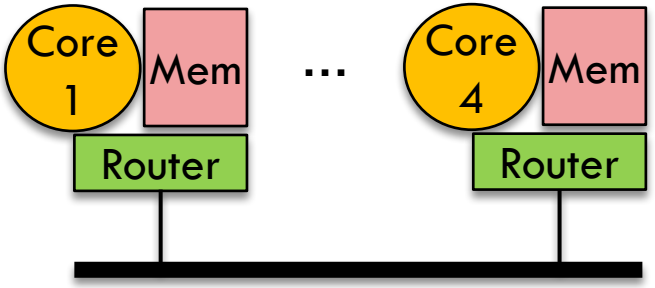
Example UMA



Non-Uniform Memory Access

- Access latency is proportional to proximity
  - Fast local accesses

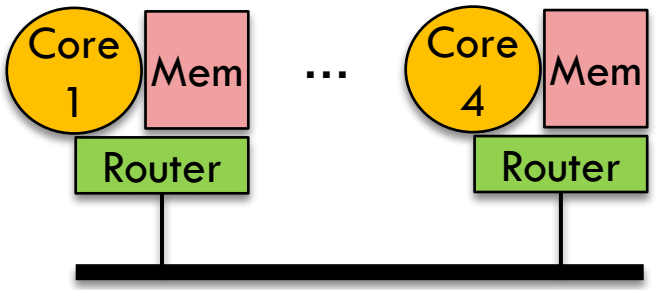
Example NUMA



Network Topologies

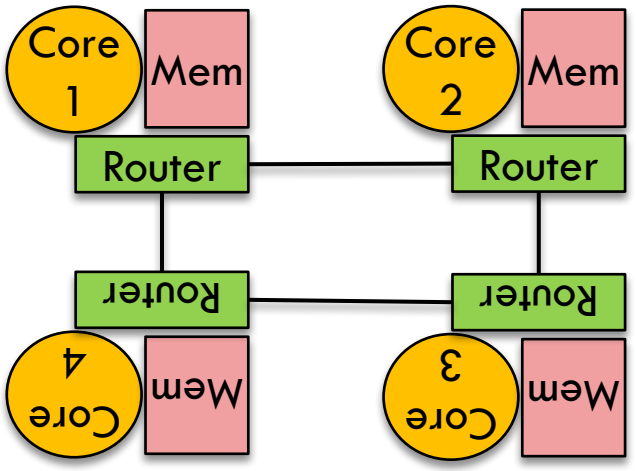
Shared Network

- Low latency
- Low bandwidth
- Simple control
  - e.g., bus



Point to Point Network

- High latency
- High bandwidth
- Complex control
  - e.g., mesh, ring

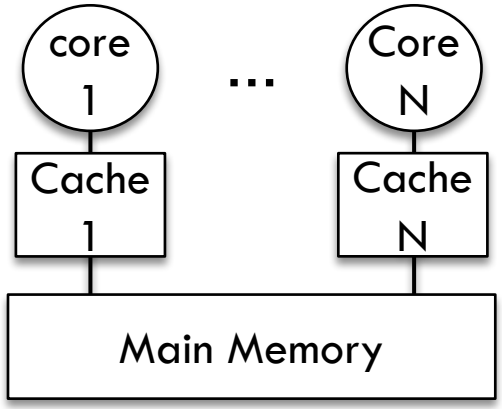


Challenges in Shared Memories

- Correctness of an application is influenced by
  - Memory consistency
    - All memory instructions appear to execute in the program order
    - Known to the programmer
  - Cache coherence
    - All the processors see the same data for a particular memory address as they should have if there were no caches in the system
    - Invisible to the programmer

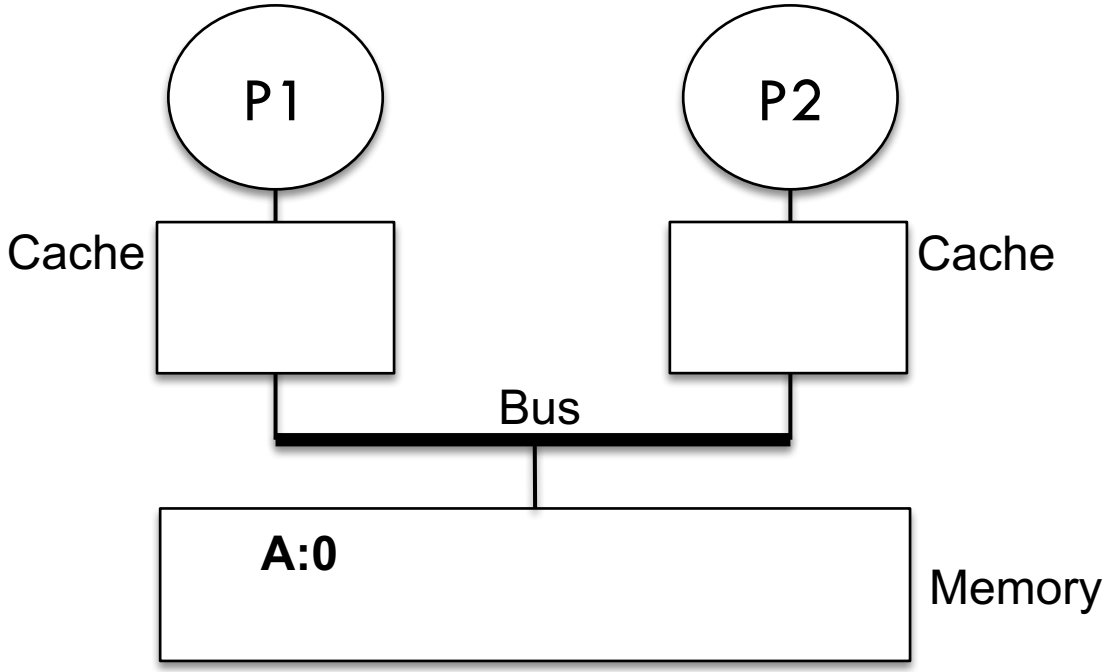
Cache Coherence Problem

- Multiple copies of each cache block
  - In main memory and caches
- Multiple copies can get inconsistent when writes happen
  - Solution: propagate writes from one core to others



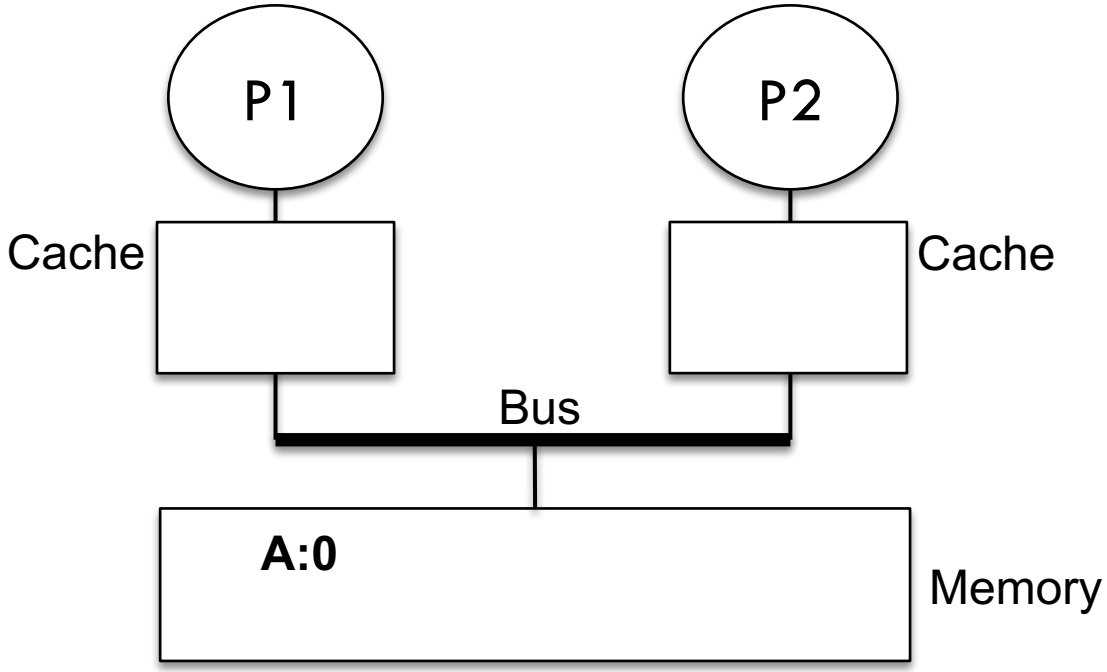
Scenario 1: Loading From Memory

- Variable A initially has value 0
- P1 stores value 1 into A
- P2 loads A from memory and sees old value 0



Scenario 2: Loading From Cache

- P1 and P2 both have variable A (value 0) in their caches
- P1 stores value 1 into A
- P2 loads A from its cache and sees old value



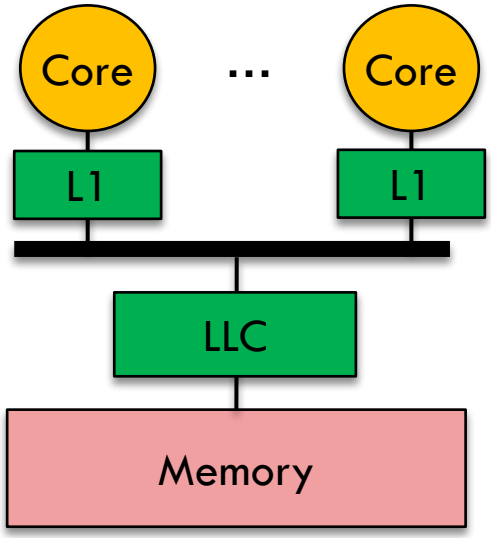
Cache Coherence

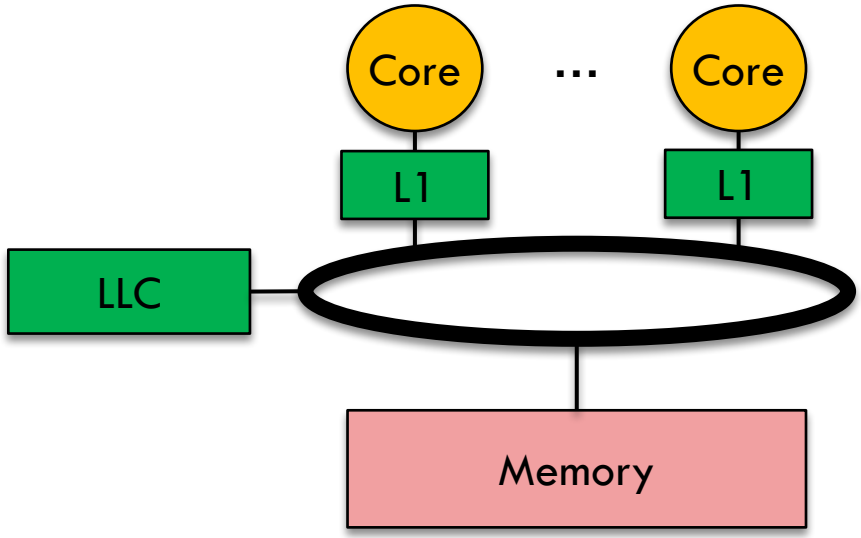
- The key operation is update/invalidate sent to all or a subset of the cores
  - Software based management
    - Flush: write all of the dirty blocks to memory
    - Invalidate: make all of the cache blocks invalid
  - Hardware based management
    - Update or invalidate other copies on every write
    - Send data to everyone, or only the ones who have a copy

- Invalidation based protocol is better. Why?

Snoopy Protocol

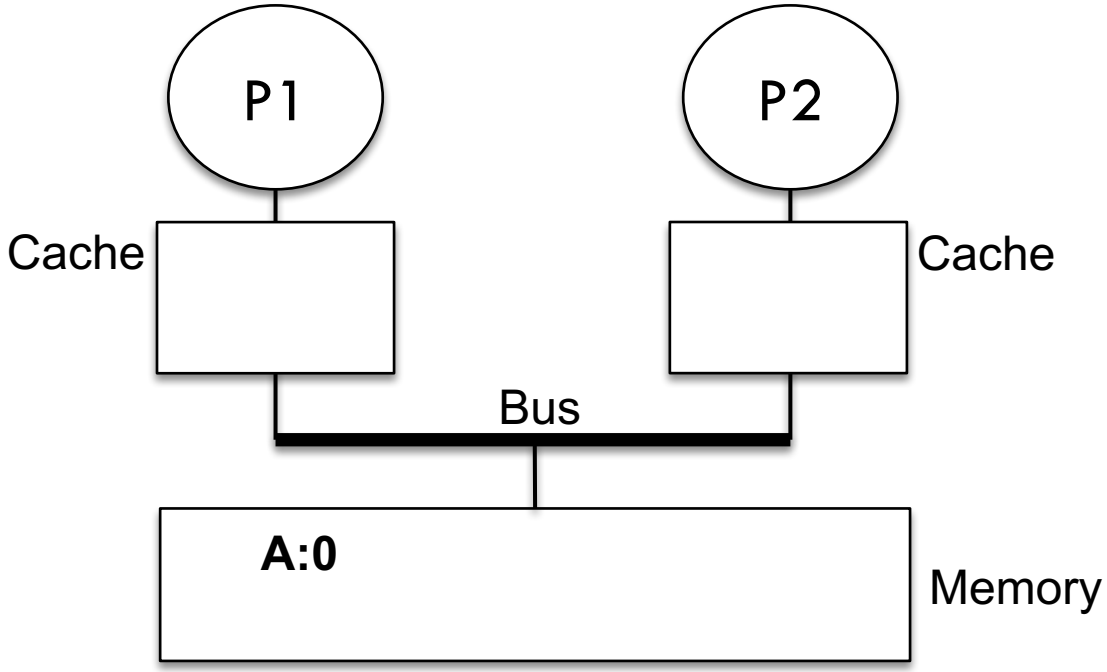
- Relying on a broadcast infrastructure among caches
  - For example shared bus
- Every cache monitors (snoop) the traffic on the shared media to keep the states of the cache block up to date





Simple Snooping Protocol

- Relies on write-through, write no-allocate cache
- Multiple readers are allowed
  - Writes invalidate replicas
- Employs a simple state machine for each cache unit



Simple Snooping State Machine

- Every node updates its one-bit valid flag using a simple finite state machine (FSM)
- Processor actions
  - Load, Store, Evict
- Bus traffic
  - BusRd, BusWr