CompSci131

Parallel and Distributed Systems

Prof. A. Veidenbaum

Today's topics

- More coherence
- Reading assignment:
  - Today: lecture notes
  - Next time: the rest of 7.2, 7.3

Last Lecture Covered

- **Cache coherence, MSI protocol**

Last time: a 3-state protocol

- States:
  - Invalid
  - Shared (clean, possibly other copies)
  - Modified (modified, only copy)
  - In the MOESI notation, a MSI protocol

- An invalidation protocol

- Let us look at the state changes in the example
  - R_{p1}, R_{p2}, W_{p4}
  - P1 P2 P4

Another Write-invalidate Protocol: the Illinois Protocol

- States:
  - Invalid
  - (Valid) Exclusive (clean, only copy)
  - Shared (clean, possibly other copies)
  - Modified (modified, only copy)
  - In the MOESI notation, a MESI protocol

- One more state to improve performance

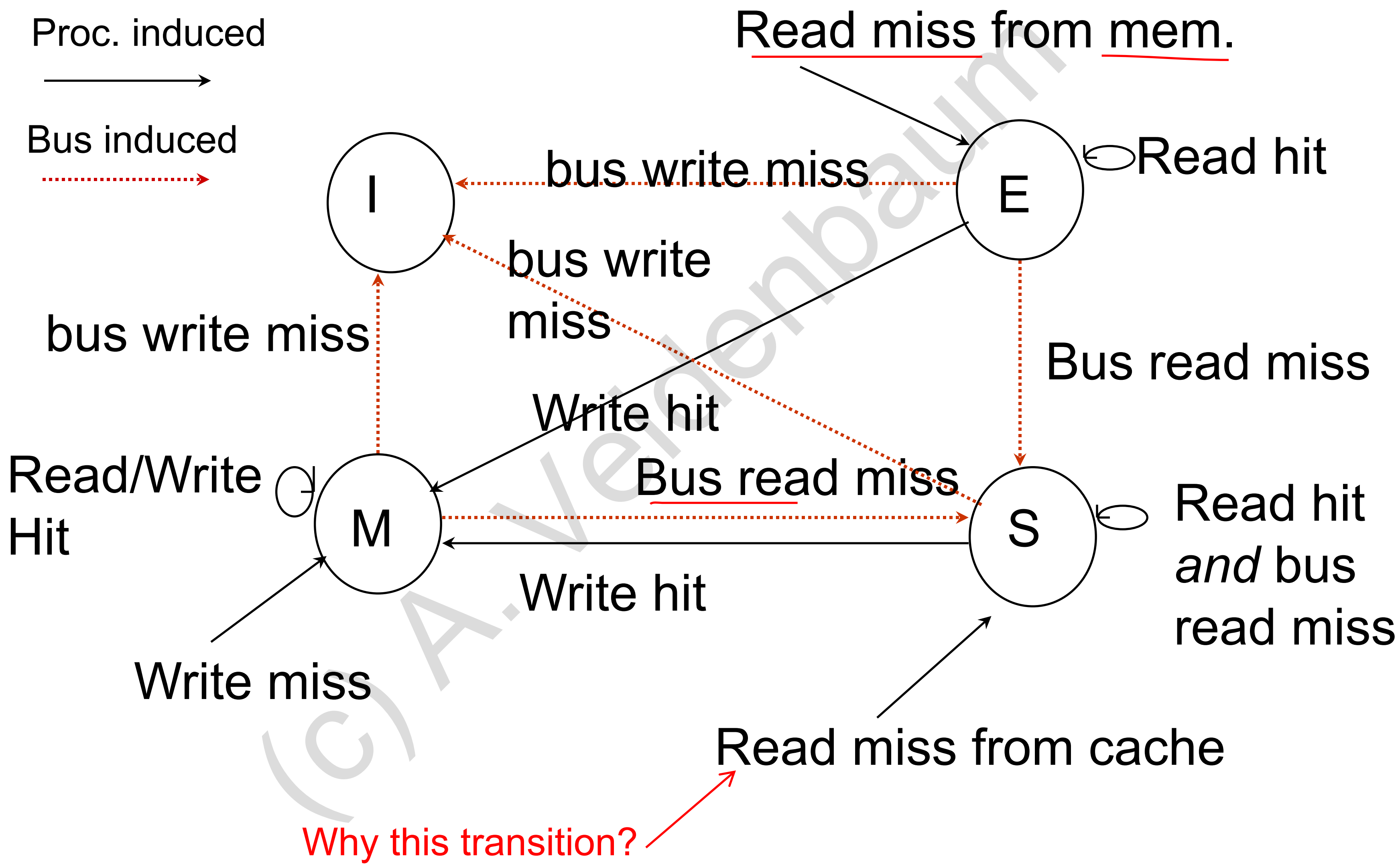
- Caches may supply data on bus misses

Illinois Protocol: Design Decisions

- The Exclusive state to enhance performance
  - On a write to a line in E state, no need to send an invalidation message
    - » occurs often for ***private*** variables

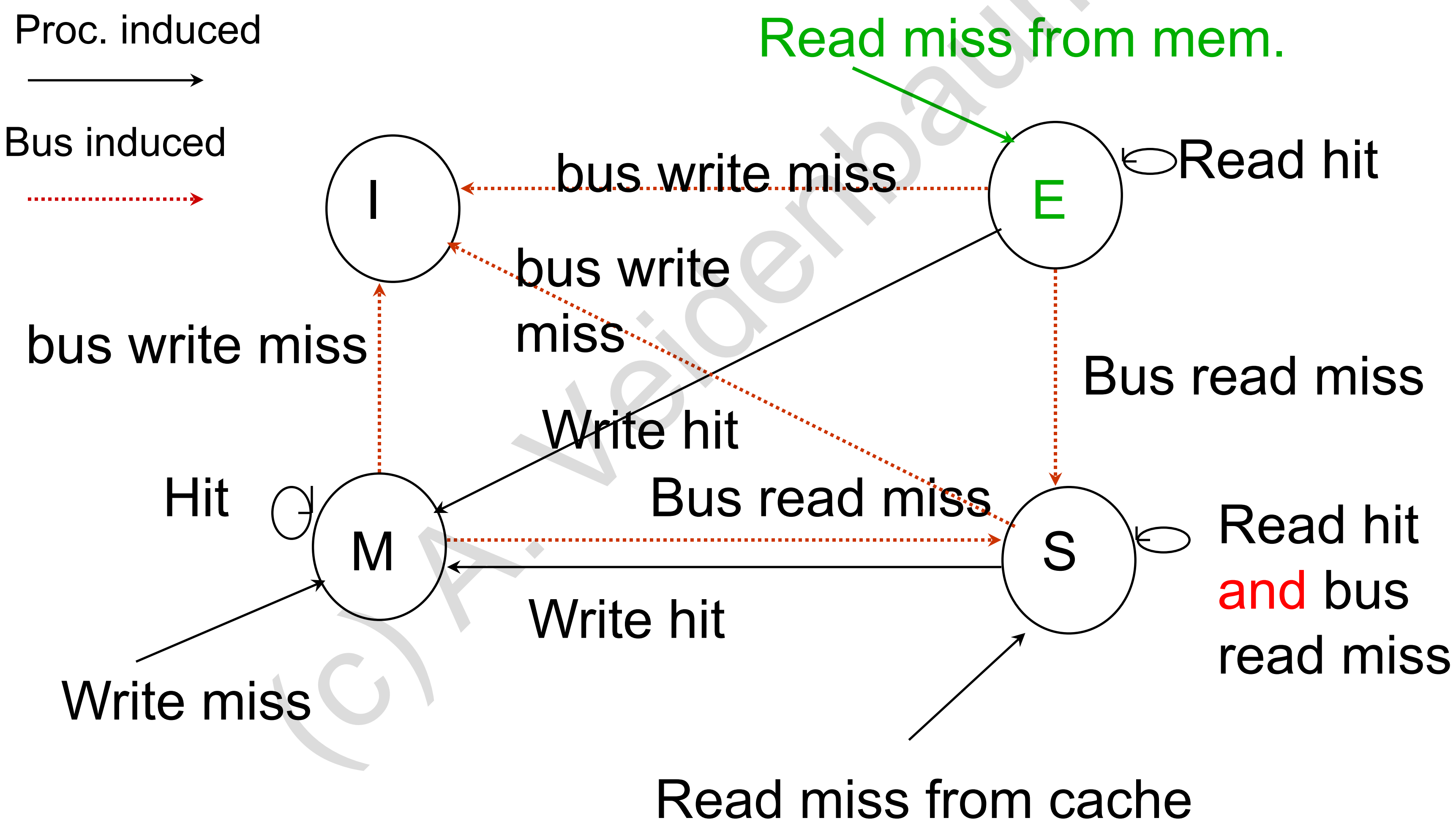
- A read miss
  - The line is not in any cache – get from memory
  - The line in E or S states in other caches – get from cache
    - » Memory access aborted
    - » If in more than one cache, which one?
      - From the first cache that grabs the bus
        - There is a bus arbiter already there
  - The line in M state – get from that cache
    - » AND initiate a memory update
- A write miss reads first (possibly w/ “Intent to Modify”)

Illinois Protocol: State Diagram



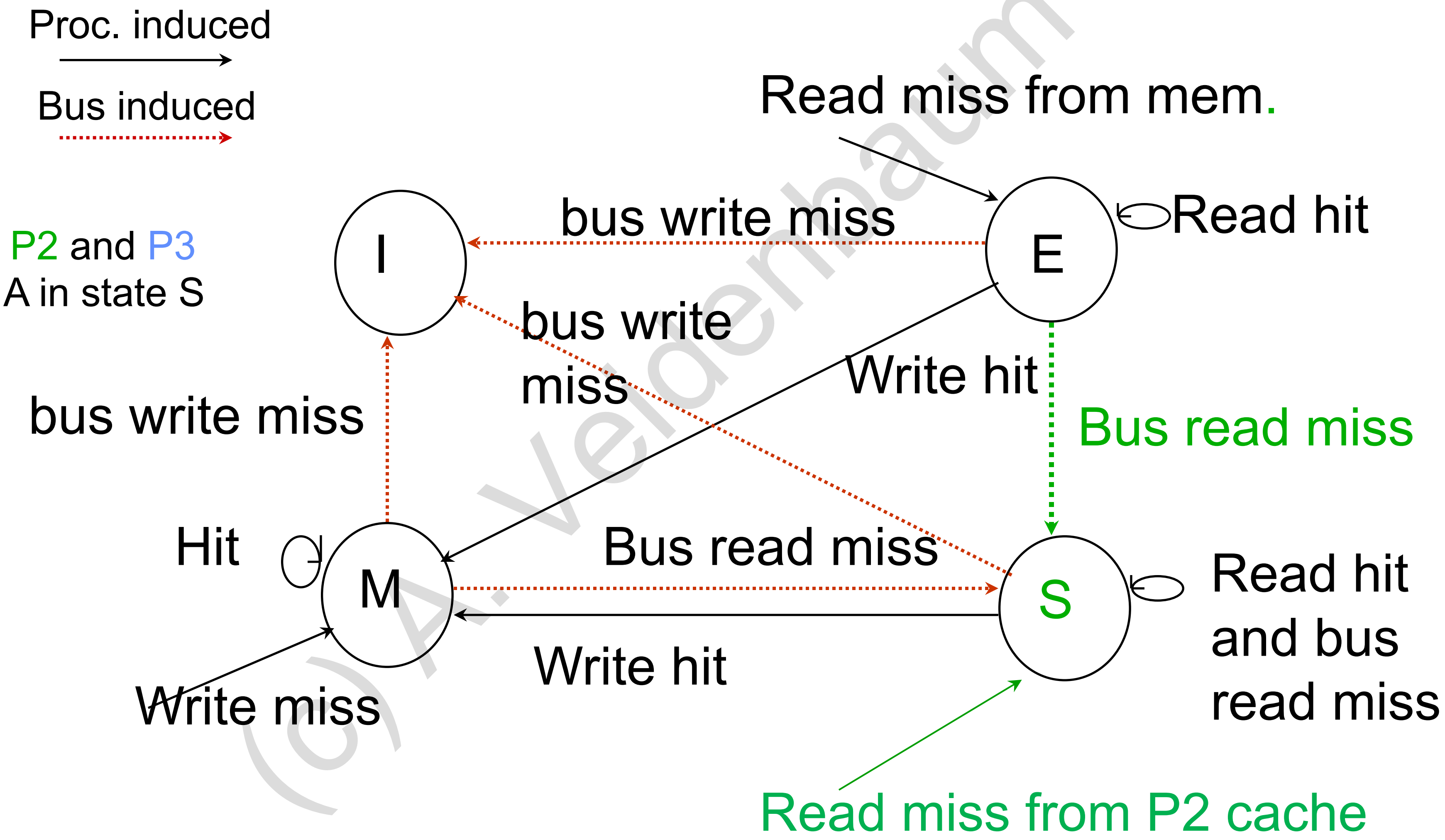
P2 reads A

A only present in memory



Example: P3 reads A

Both P2 and P3 have A in state S



Proc. induced

Bus induced

I

E

M

S

Read miss from mem.

Read hit

bus write miss

bus write miss

Write hit

Bus read miss

bus write miss

Hit

Bus read miss

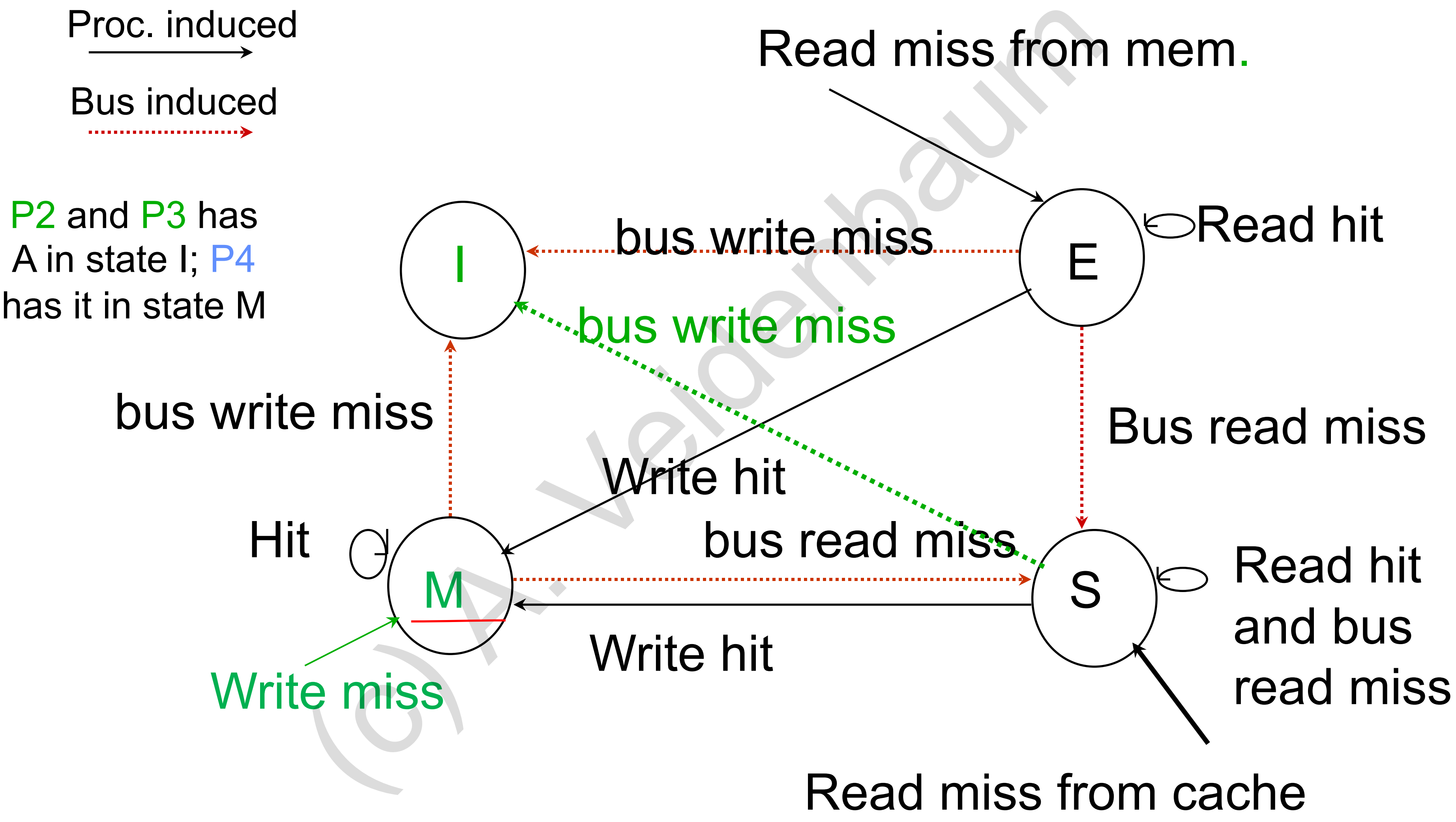
Write hit

Read hit and bus read miss

Write miss

Read miss from P2 cache

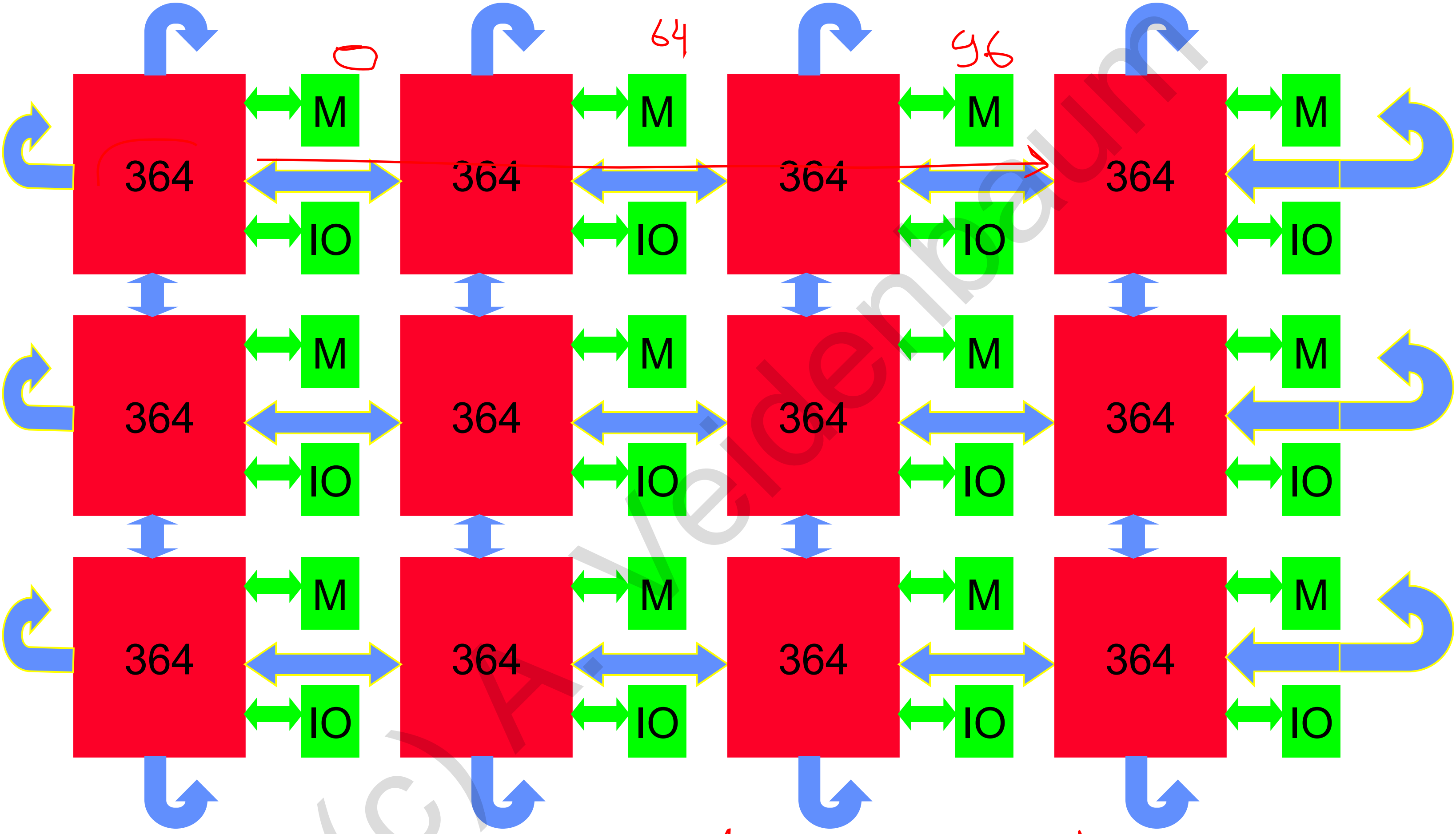
Example: P4 writes A



Cache Coherence without snooping

- Snooping is not possible except on bus/ring
  - These are not scalable networks
  - Have to use more complex networks
- Broadcast / multicast is slow, not that easy
  - In Multistage Interconnection Networks (MINs), potential for message blocking is very high
  - In mesh-based networks, broadcast to every node is very inefficient
- In both case network delays can be very high

A system using torus interconnect



M=Private+Shared

A $\bmod_m = 64$

- **What to do about cache coherence?**
  - **Have no caches (Cray MTA)**
  - **By hardware:**
    - **have a data structure to record the state of a line**
      - **Called a (coherence) directory**
  - **By software:**
    - **Disallow or limit caching of shared variables (Cray 3TD)**
    - **Compiler assisted**

Directory Protocols

- Central directory has "state" for each line
  - Consult directory on every access
    - Not scalable
- Distributed directory
  - Distribute the directory using "home nodes"
    - Home the memory where a line resides
  - Complex protocols
- Information in the directory:
  - Whether a line is cached anywhere
  - Its state (say one of MSI)
  - Where are the copies of the line
    - List of processors with a copy

Protocol

- Every miss needs to go to this home node
  - Protocol is write-invalidate
  - Let us do the MSI protocol on this
    - P1(R), P2(R), P4(W)

P1 P2 P4 Memory

Directory Size

- A directory entry is a state vector associated with the line
  - For an *n* processor system, an (*n*+*1*) bit vector
  - Bit 0, clean/dirty
  - Bits 1-n: “location” vector
    - » Bit *i* is set if *i*th cache has a copy
- Memory overhead:
  - For a 256 processor system, 257 bits / block
    - » If a block is 64 bytes, overhead = 257 / (64 * 8)
      - over 25%
  - This data structure is not scalable
- On average, there are only 4 to 8 “sharers”

Software solutions for parallel programs

1. A programmer declares private/shared vars
   – No coherence action on private
   – Shared is not cached

2. Compiler managed
   – Main Assumption: caches can be temporarily incoherent as long as no incorrect data usage occurs.
     » Assumption 1: compiler or user declares parallelism
       • Say OpenMP loops
     » FACT: Data produced in a parallel loop cannot be used in the parallel loop
       • Otherwise the loop is not parallel
     » Assumption 3: compiler generates invalidations to enforce coherence before/after parallel loop(s)