

Multiprocessors

Erik Hagersten
Uppsala University



Outline of these lectures

1. Processor implementations
2. Caches and memory system
- 3. Multiprocessors**
4. HW optimizations
5. Multicore processors
6. SW optimizations



The era of the “supercomputer” multiprocessors in the 1980-90s

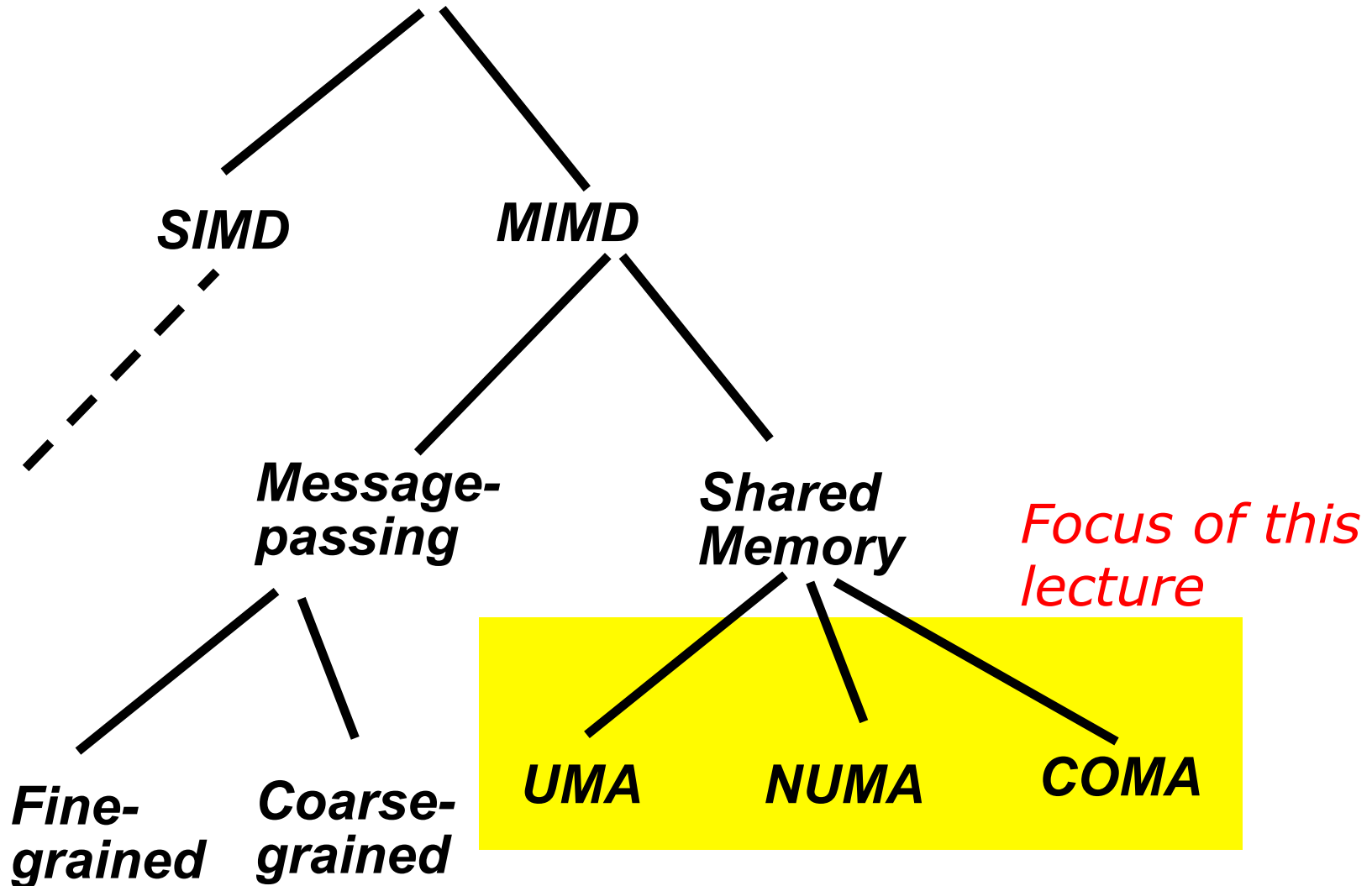
- The one with the most blinking lights wins
- The one with the strangest languages wins
- The niftier the better!



Multiprocessors 3



Taxomy for Architectures [Flynn]

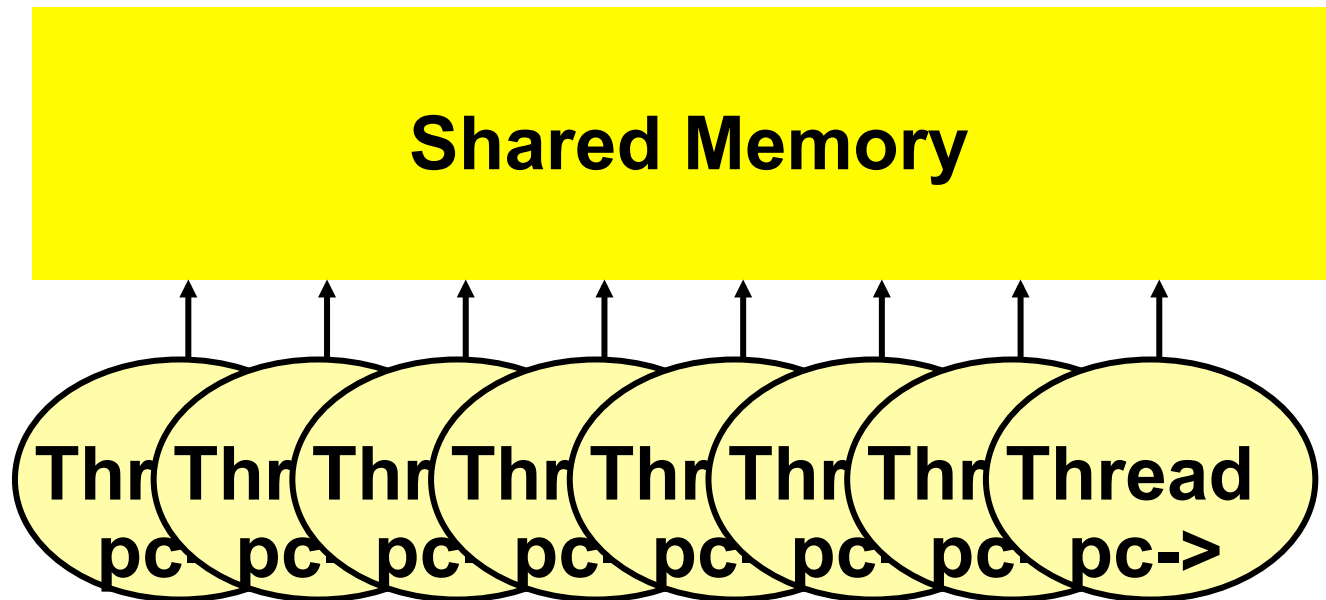




Coherent Shared Memory

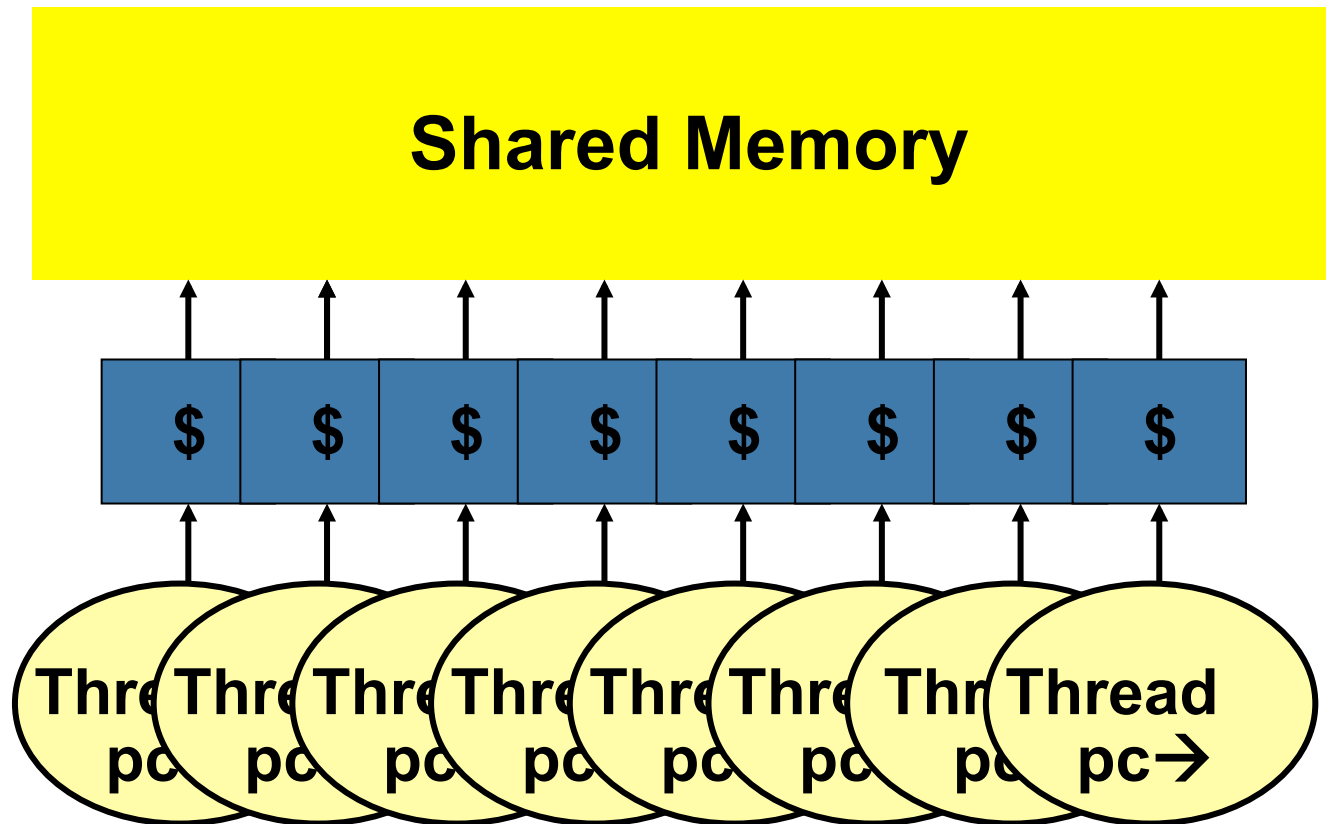
Erik Hagersten
Uppsala University

Programming Model: Shared Memory



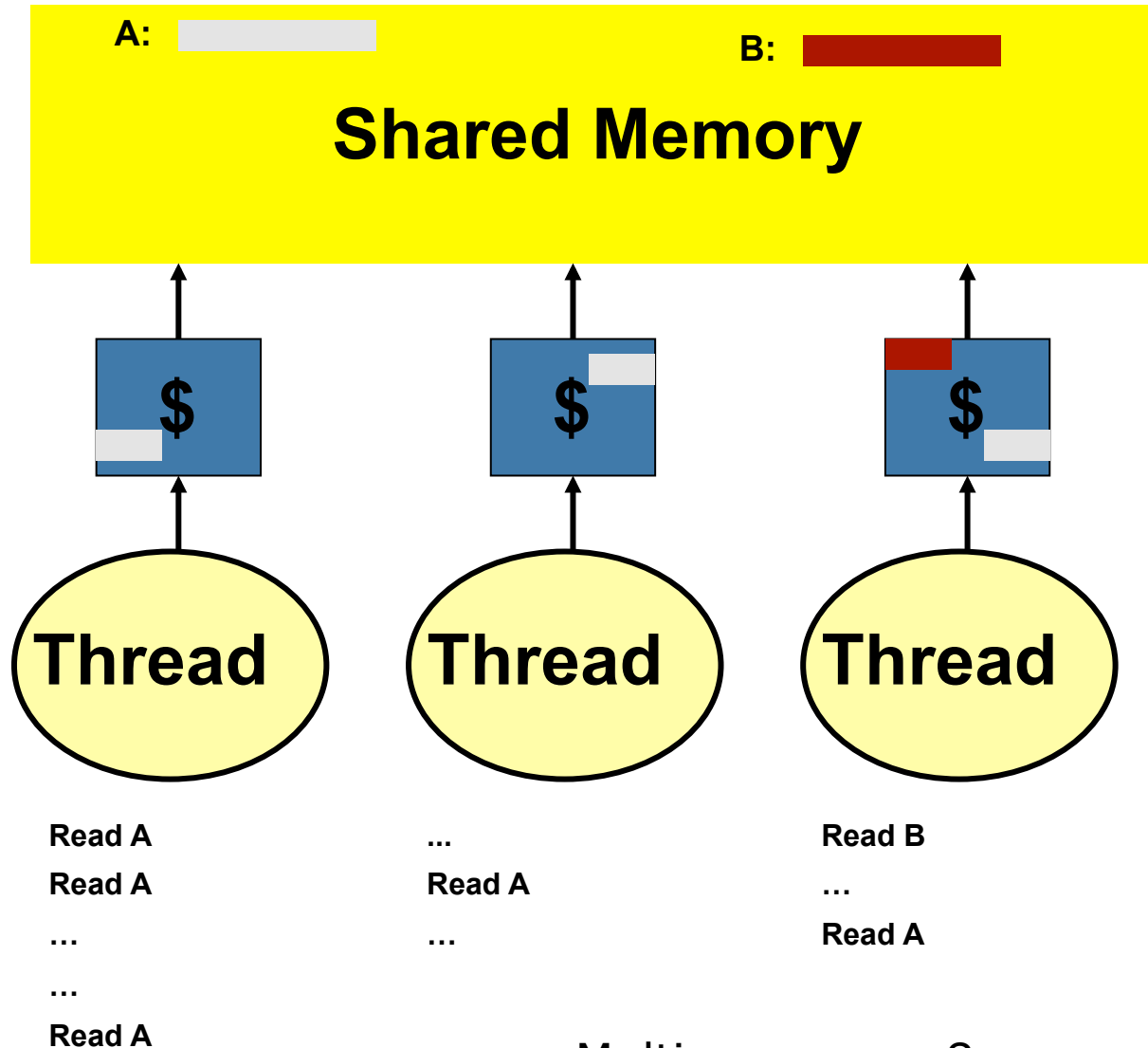
Thread-Level Parallelism (TLP)
Multiple Instructions, Multiple Data (MIMD)

Adding Caches: Cuts latency and memory bandwidth

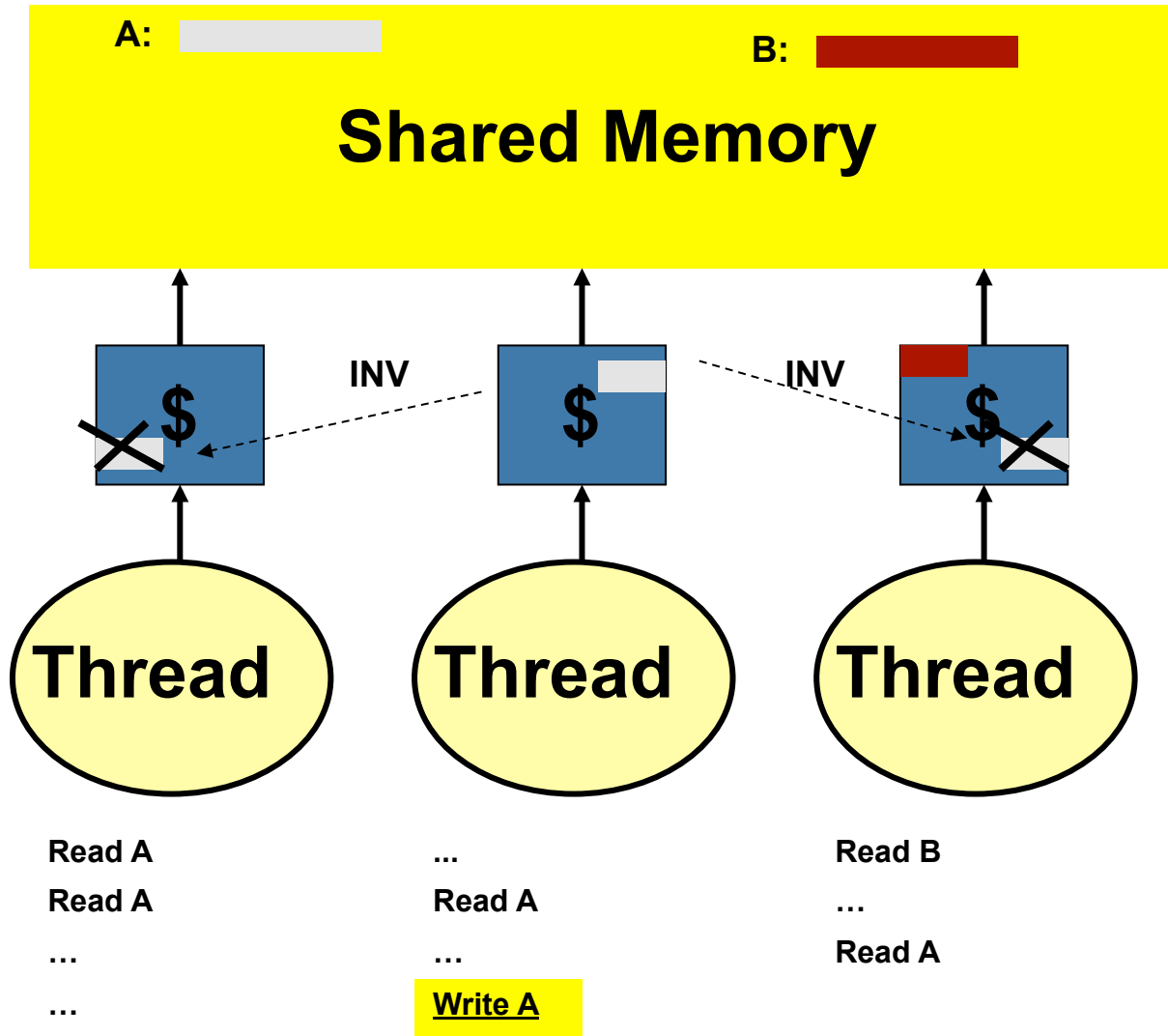


Caches:

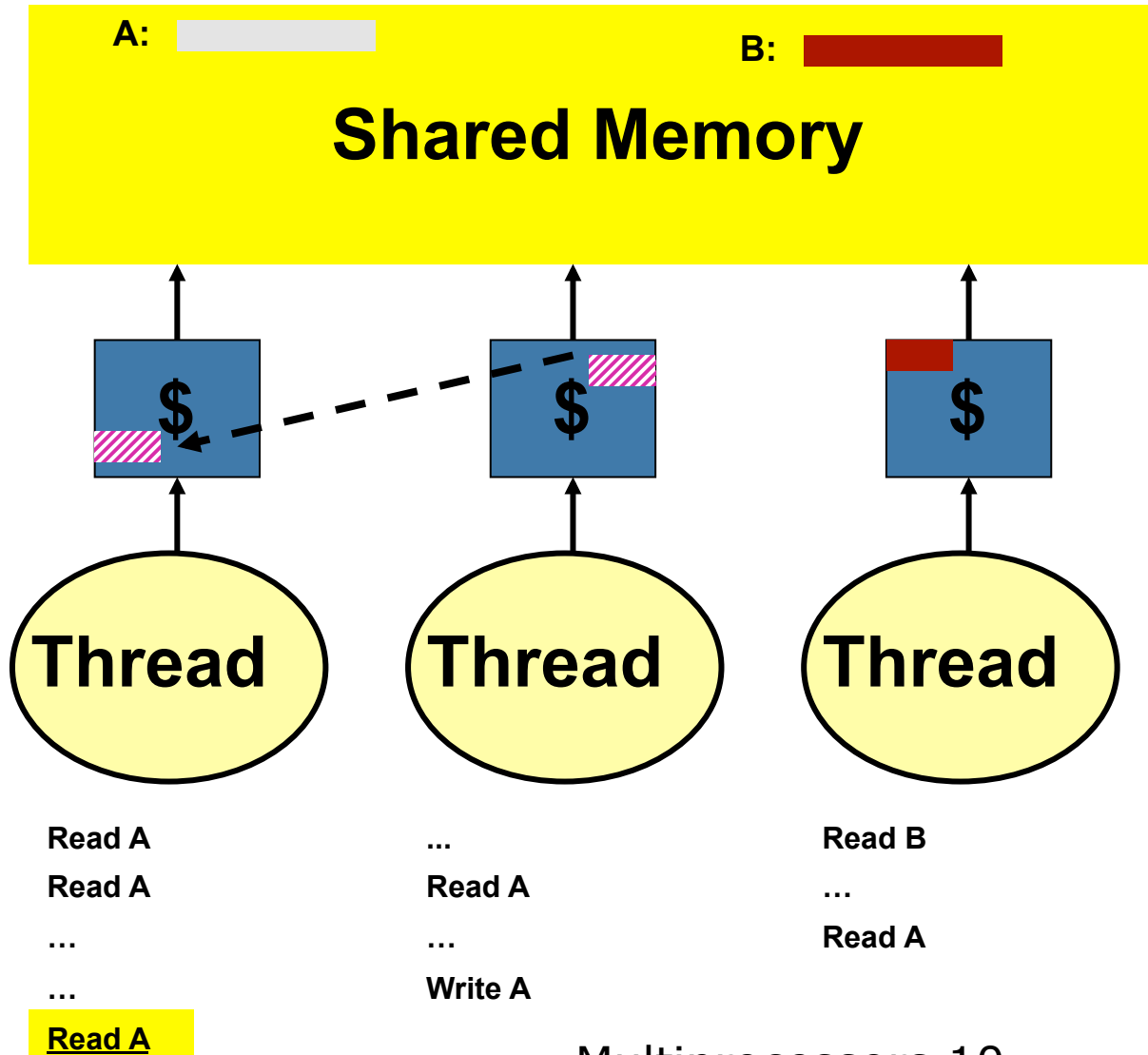
Automatic Replication of Data



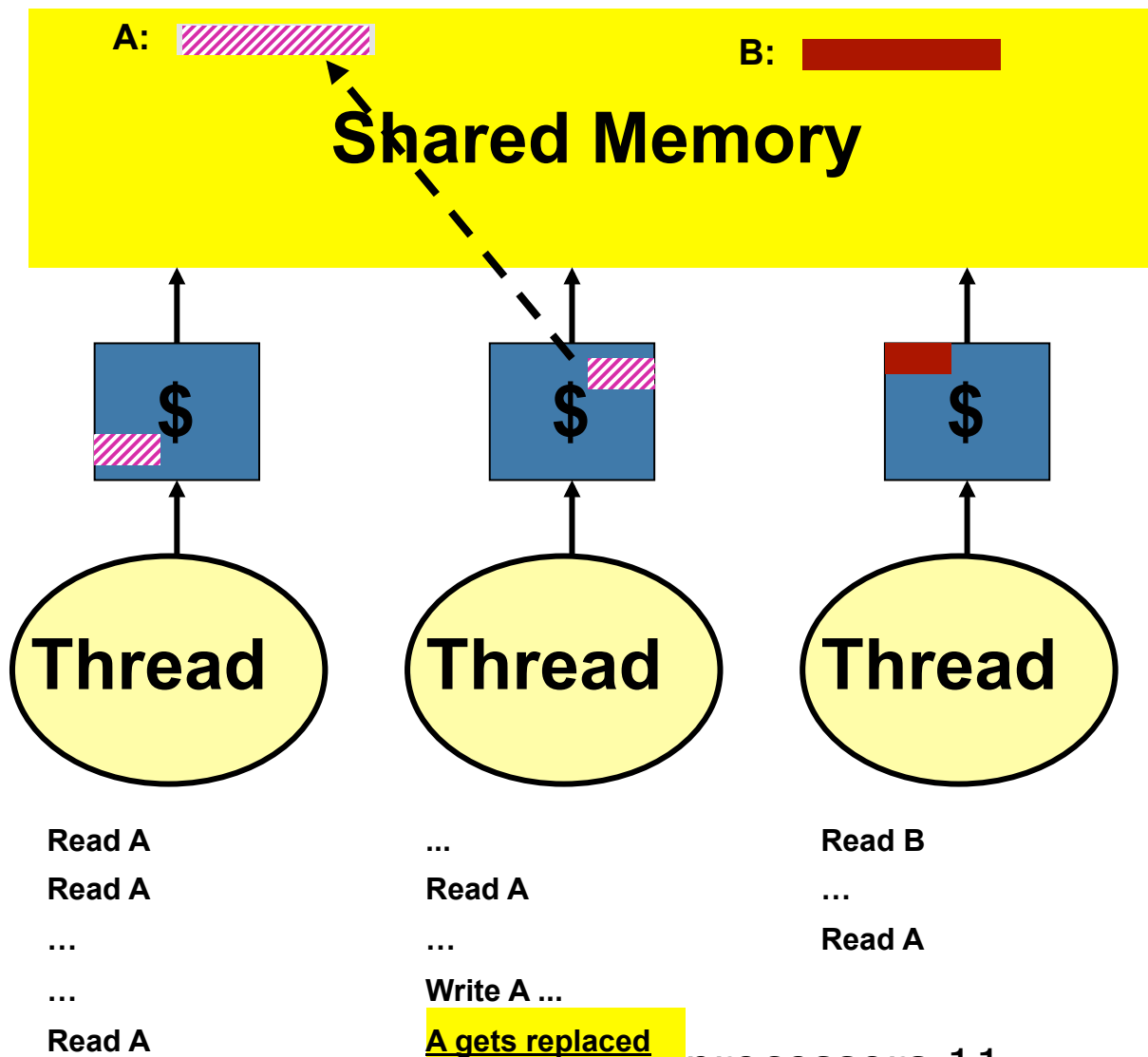
The Cache Coherent Memory System



The Cache Coherent \$2\$



Writeback





Summing up Coherence

Sloppy: there can be many copies of a datum, but only one value

Too strong definition!

Coherence: *There is a single global order of value changes to each datum*

Memory order/model: *Defines the order between accesses to many data*



UPPSALA
UNIVERSITET

Implementing Coherence

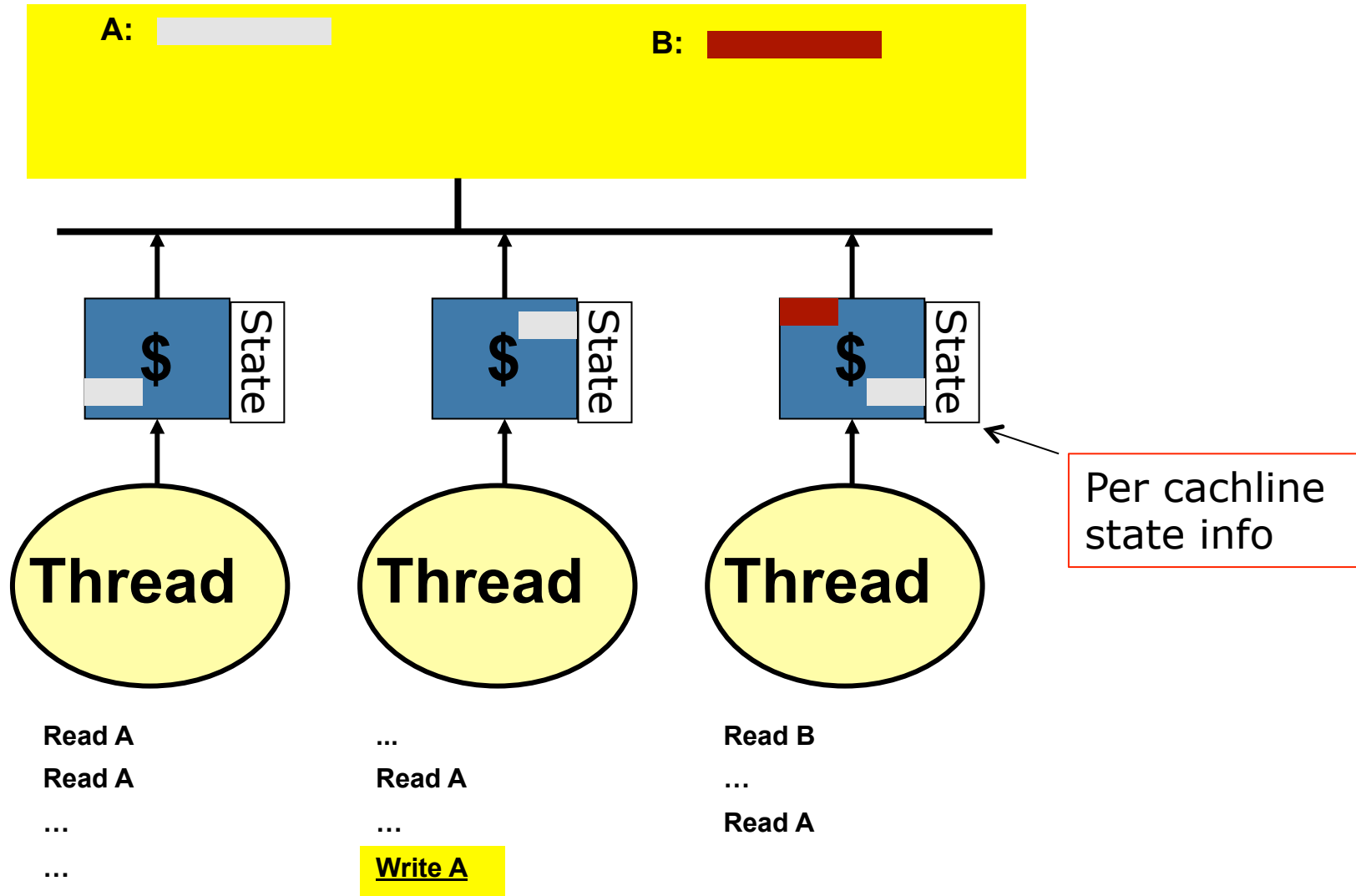
PDC
Summer
School
2017

Dept of Information Technology | www.it.uu.se

Multiprocessors 13

© Erik Hagersten | user.it.uu.se/~eh

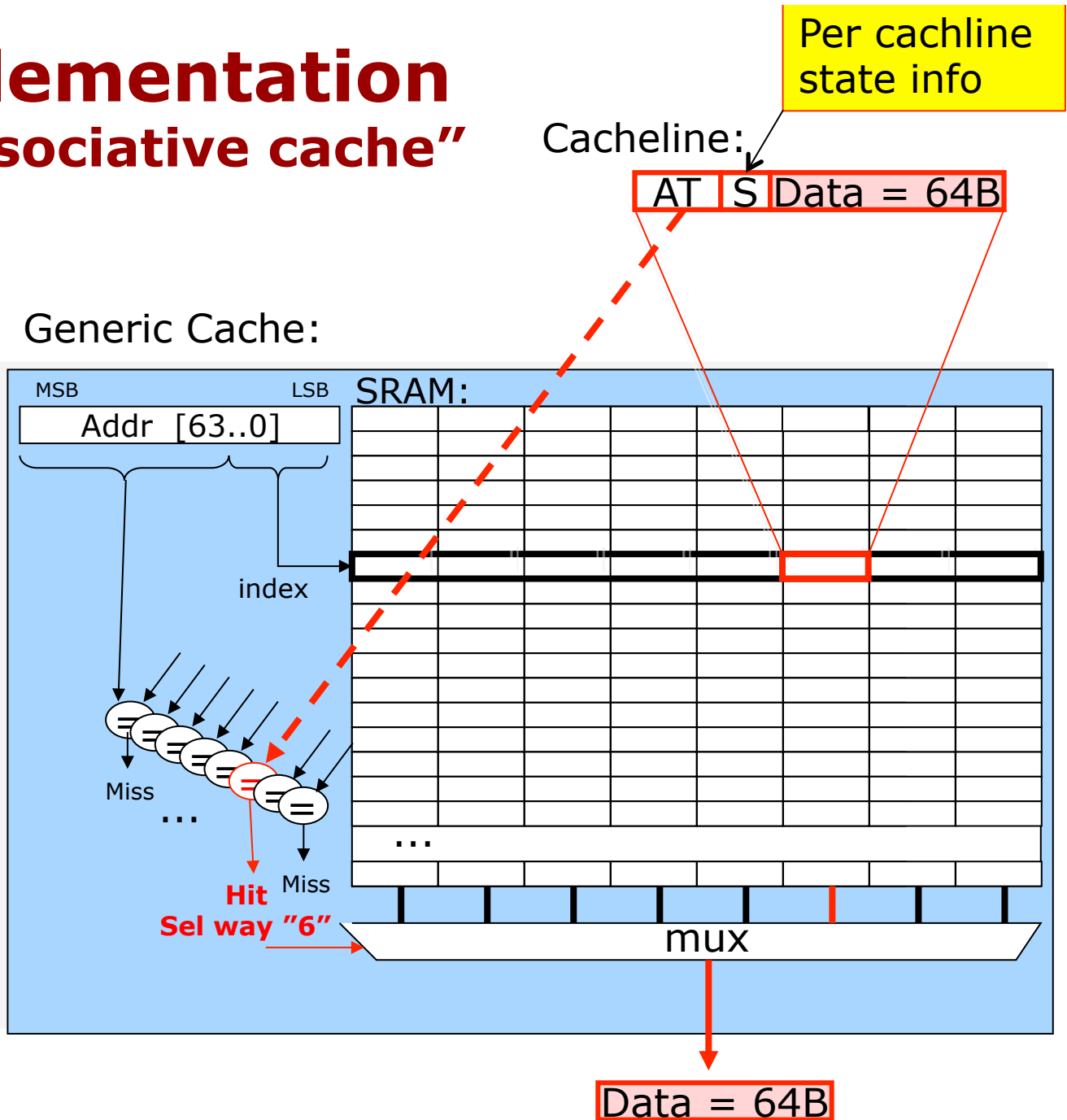
"Upgrade" in snoop-based





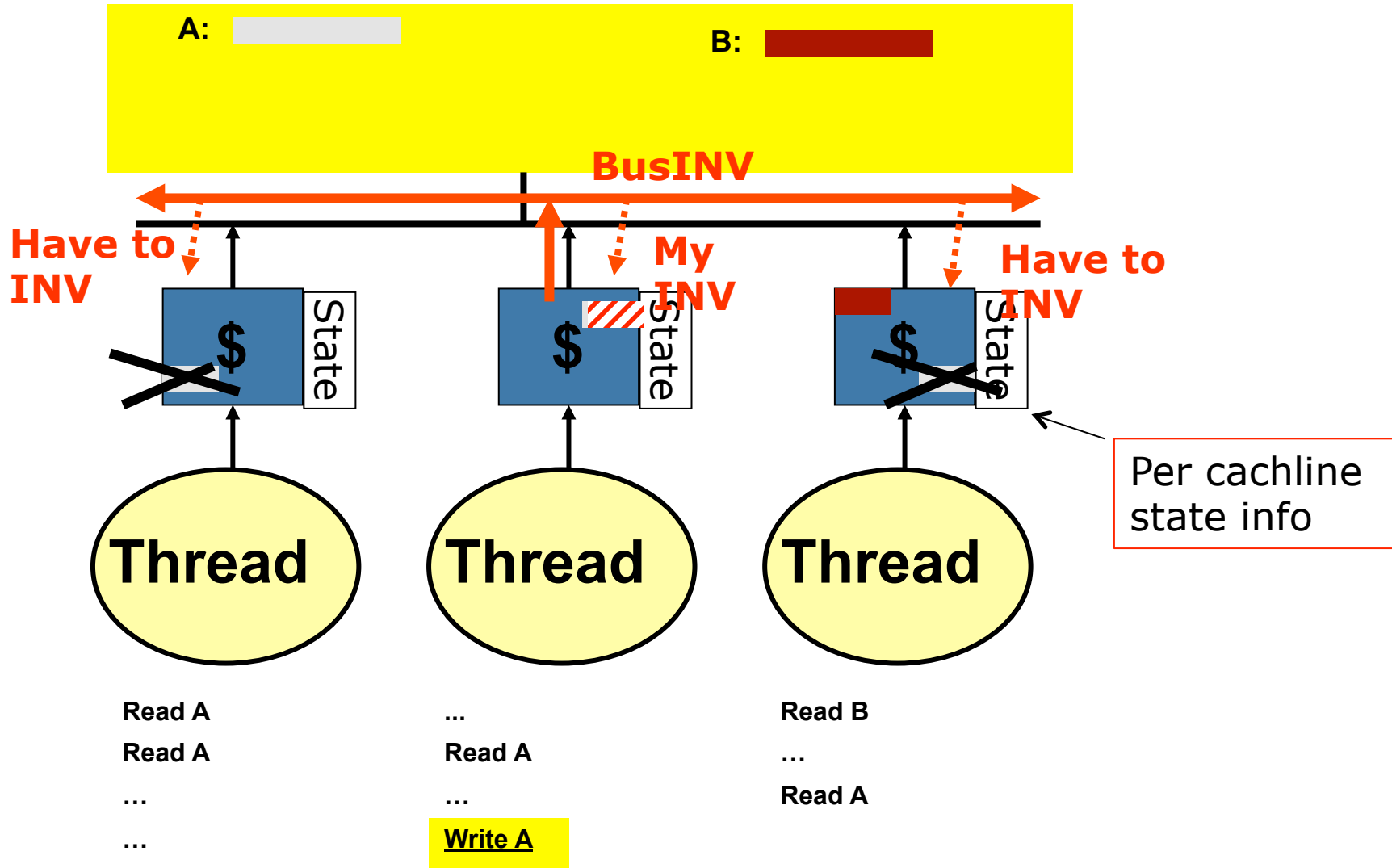
Cache implementation

"8-way set-associative cache"



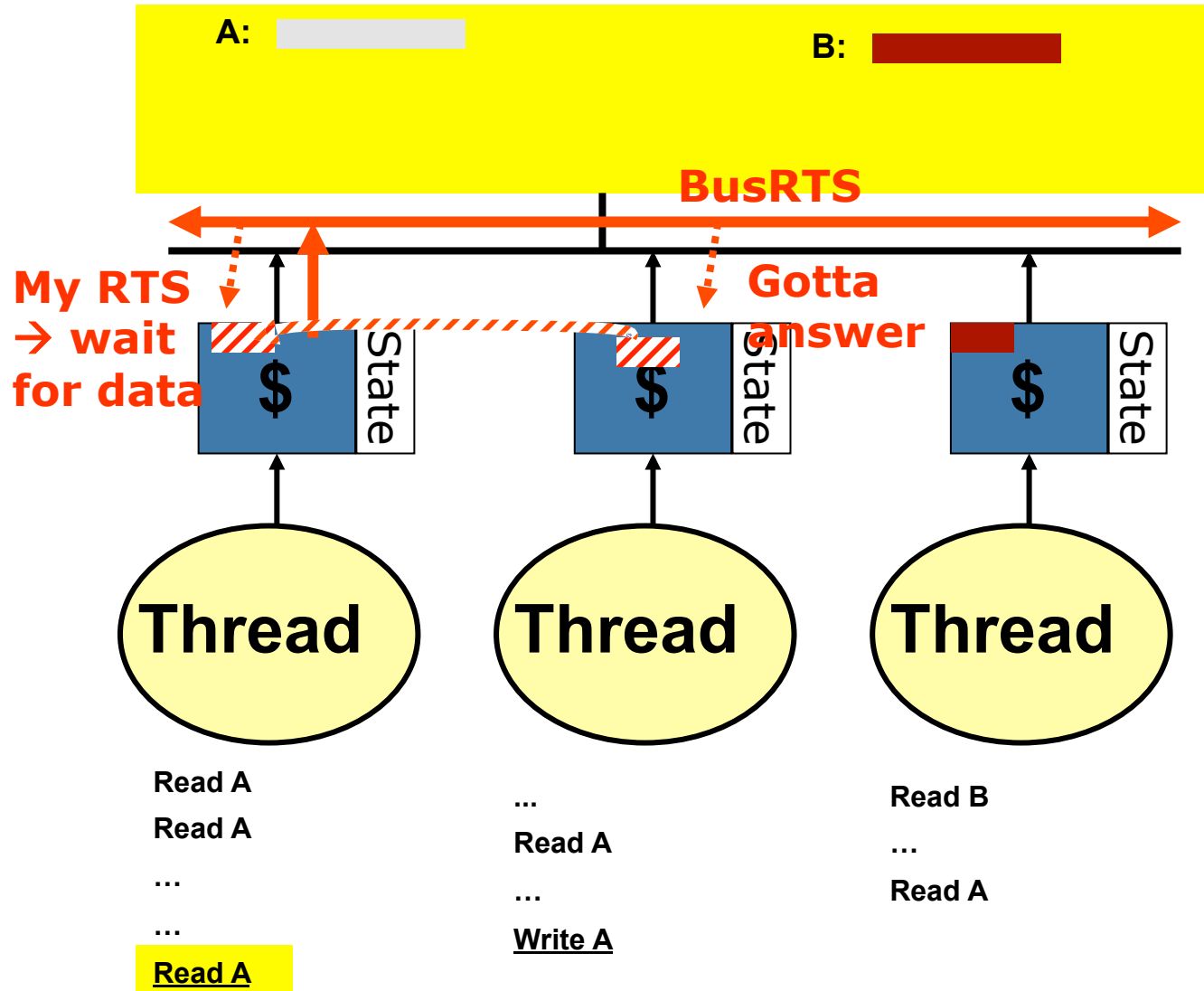


"Upgrade" in snoop-based



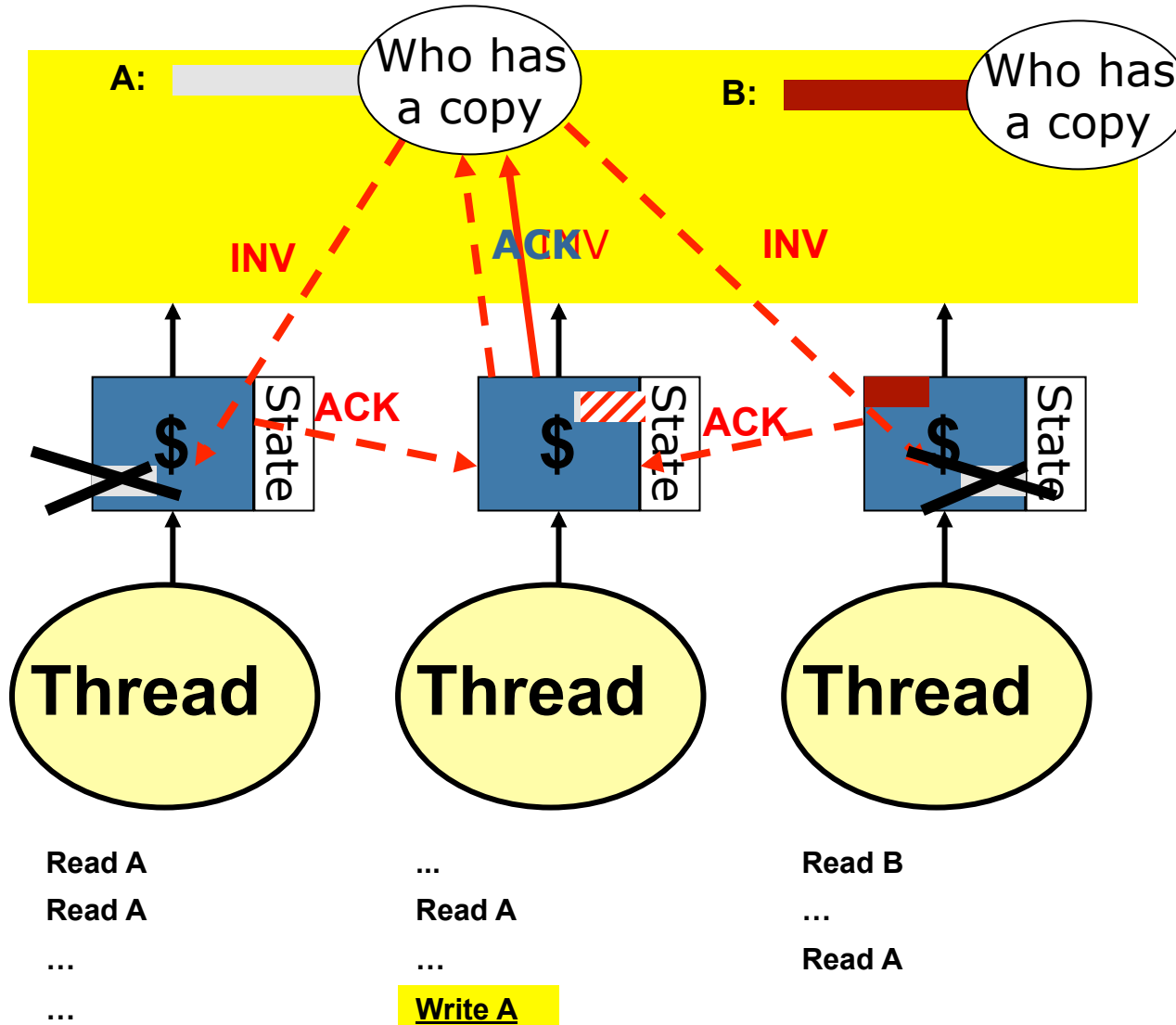


Cache-to-cache in snoop-based



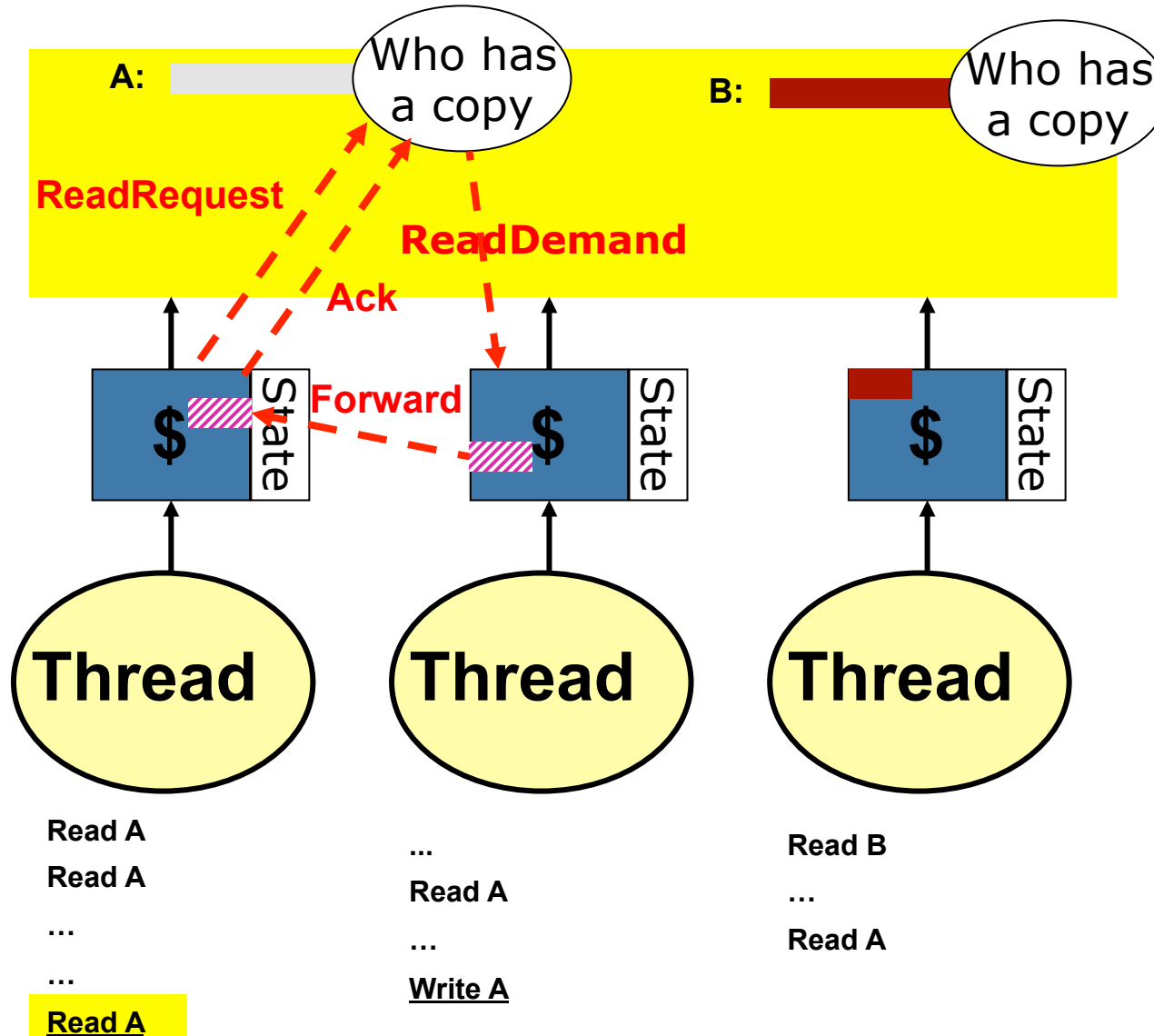


"Upgrade" in dir-based

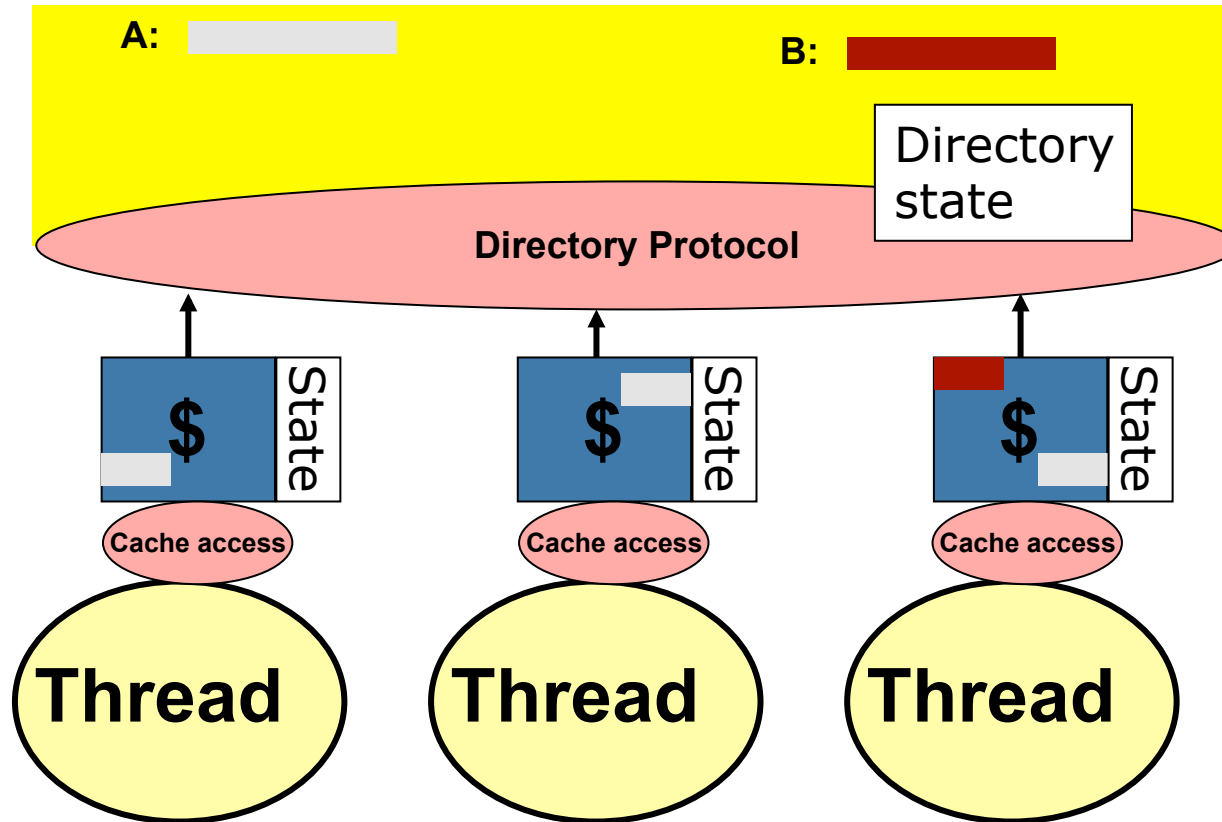




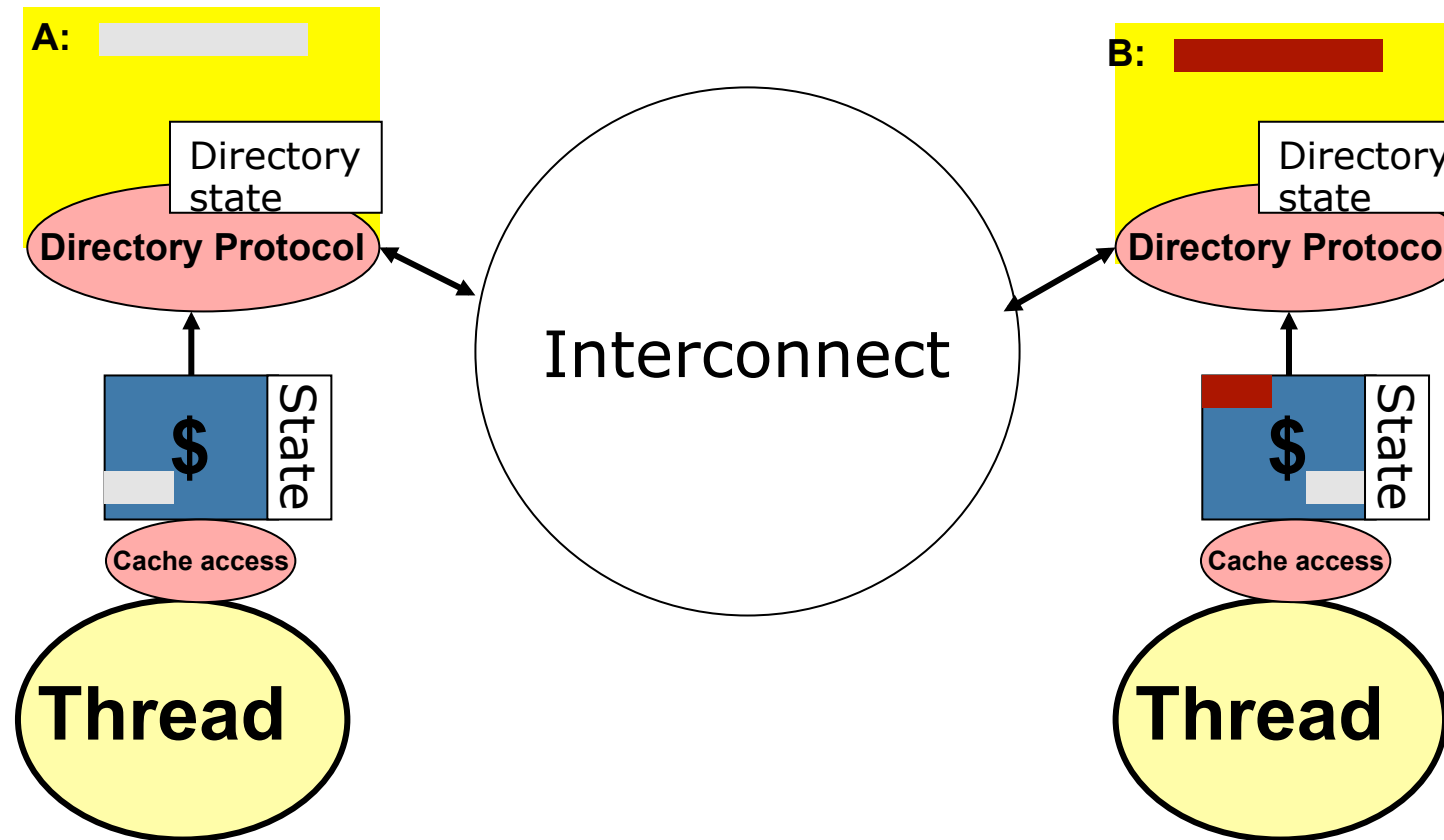
Cache-to-cache in dir-based



Directory-based coherence: Per-cachline info in the memory



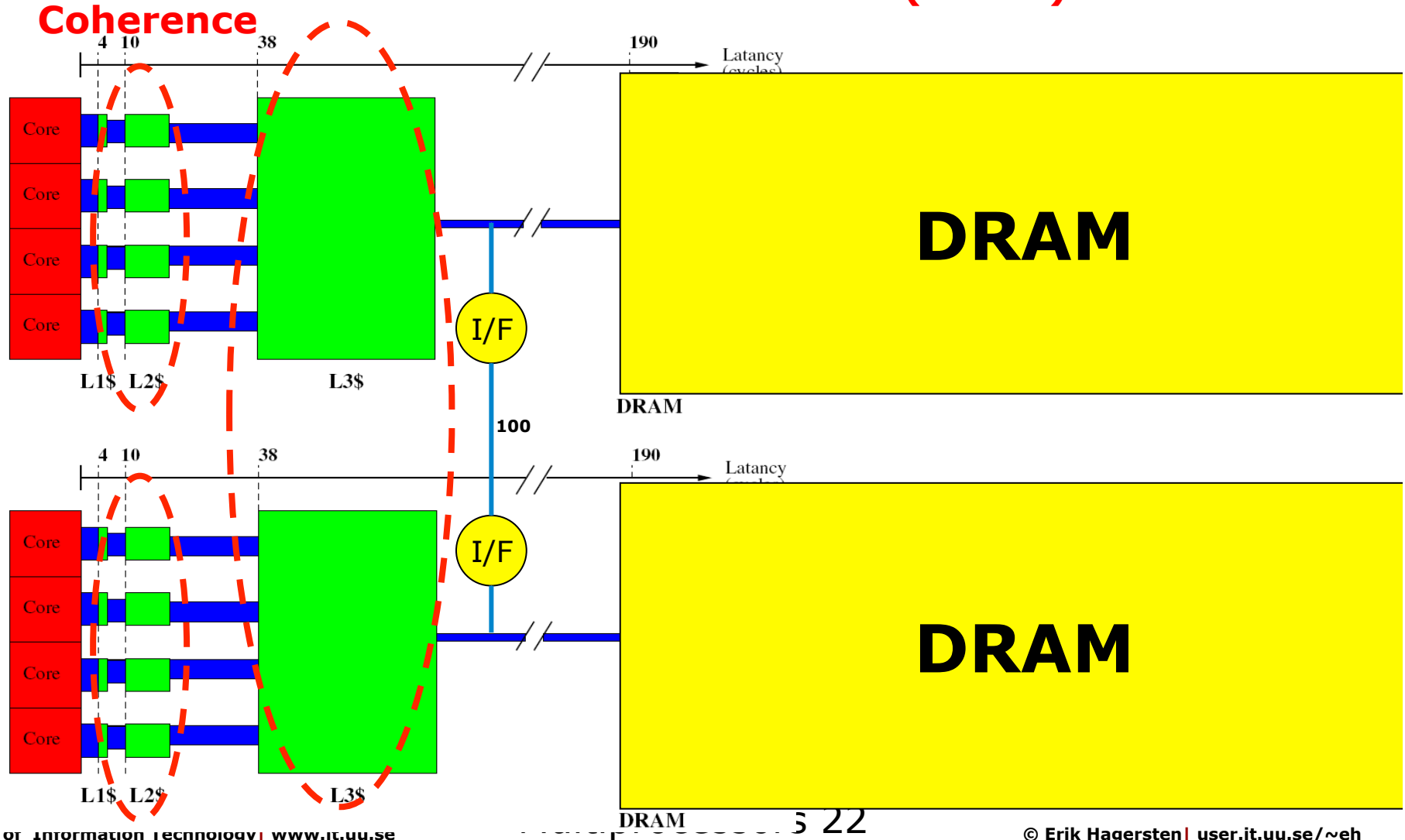
Directory-based snooping: NUMA. Per-cacheline info in the home node





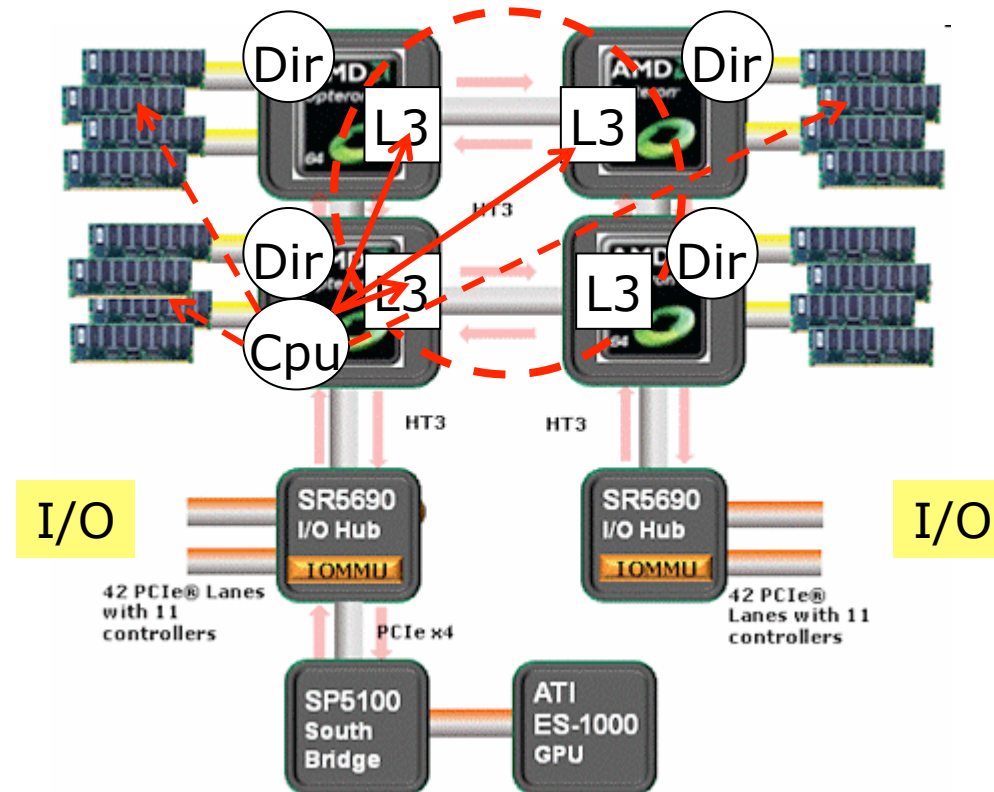
Multisocket

Coherence = Non-Uniform (NUMA)



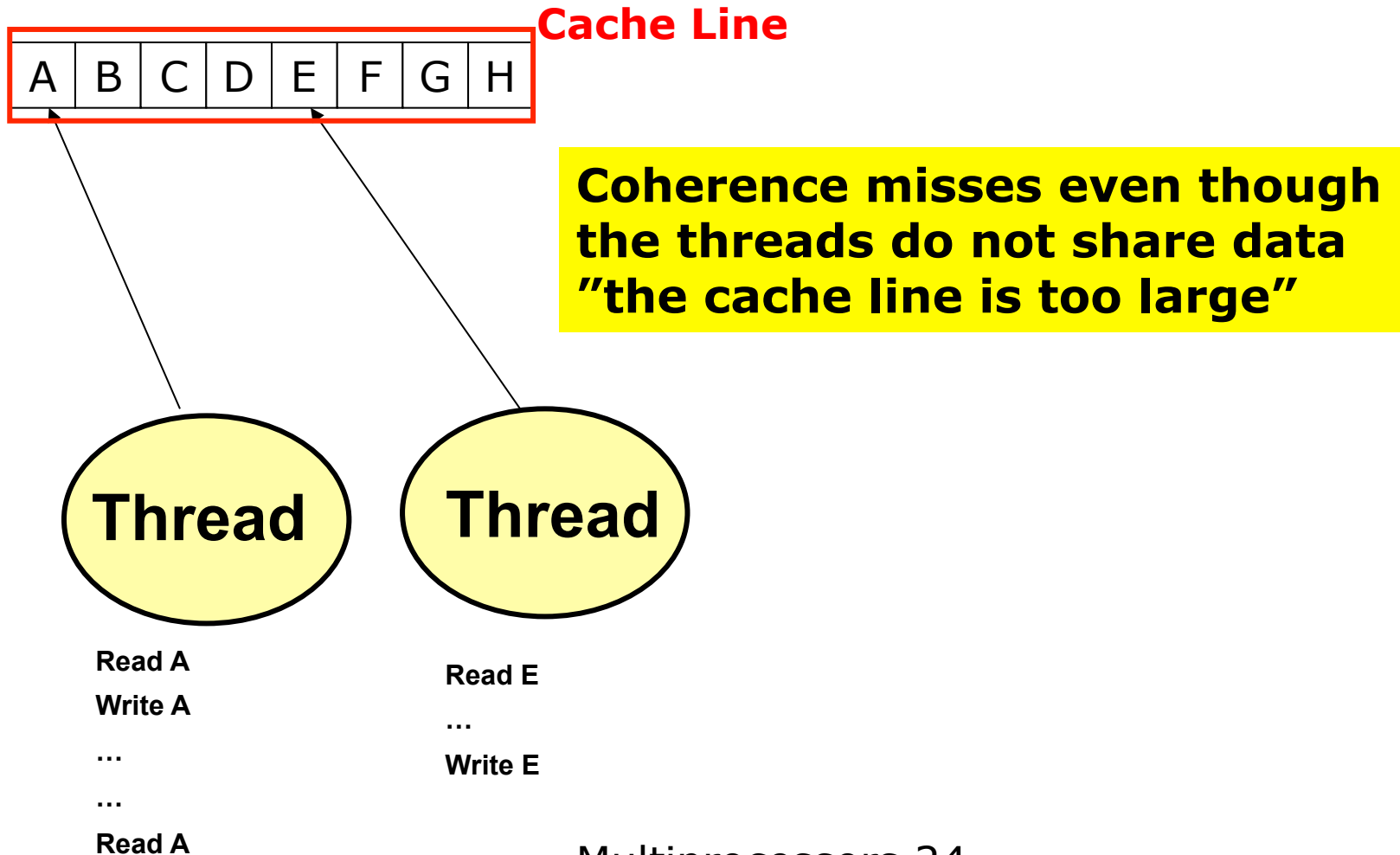
AMD Multi-socket Architecture (same applies to Intel multi-sockets)

Coherence = Non-Uniform





False sharing: Coherence is maintained with a cache-line granularity





More Cache Lingo

- **Capacity miss** – too small cache
- **Conflict miss** – limited associativity
- **Compulsory miss** – accessing data the first time
- **Coherence miss** – I would have had the data unless it had been invalidated by someone else
- **Upgrade miss** (only for writes) – I would have had a writable copy, but gave away readable data and downgraded myself to read-only
- **False sharing:** Coherence/downgrade is caused by a shared cacheline, to by shared data:

False sharing example:

Read A ...
 ... Read D
 Write A ...
 ... Write D
 Read A

cacheline:

A, B, C, D

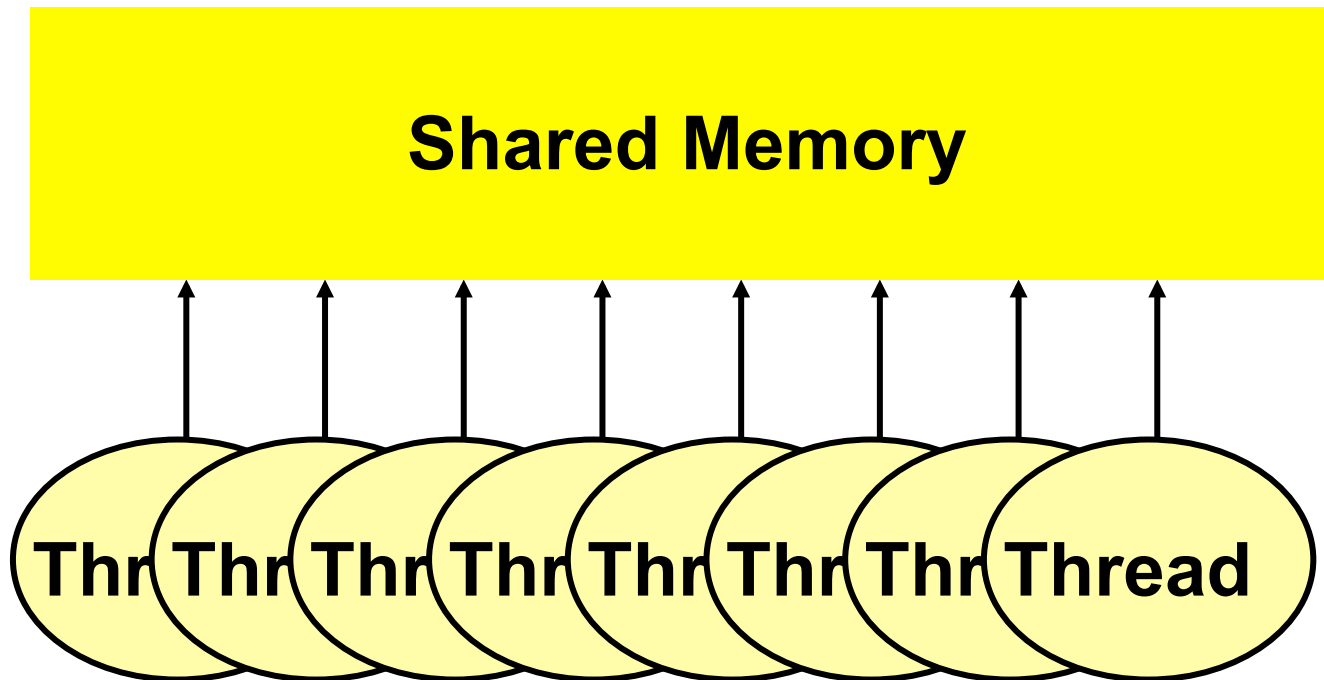


Memory Ordering (aka Memory Consistency) -- tricky but important stuff

Erik Hagersten
Uppsala University
Sweden



The Shared Memory Programming Model (Pthreads/OpenMP, ...)





Memory Ordering

- Coherence defines a per-datum valuechange order
- Memory model defines the valuechange order for all the data.

Where Memory Models Matter

■ Flag synchronization

(initially flag = 0 and A = 0)

```

...
A = 1;
flag = 1;
...
while (flag != 1) {};
X = A;
print(X);

```



Trick question

What value will be printed?

- 0
- 1
- Undefined (either 0 or 1)

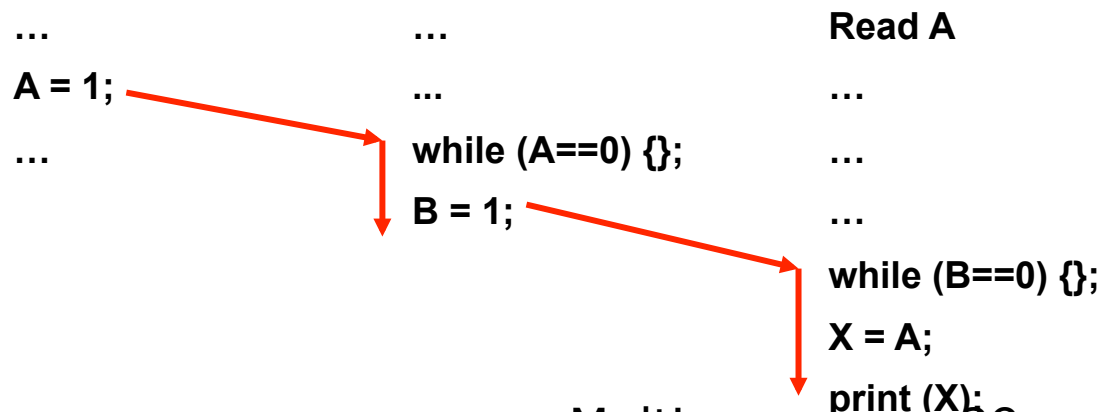
■ Causality (Causal correctness)

(Initially A = 0 and B = 0)

```

...
A = 1;
...
while (A==0) {};
B = 1;
...
Read A
...
while (B==0) {};
X = A;
print (X);

```

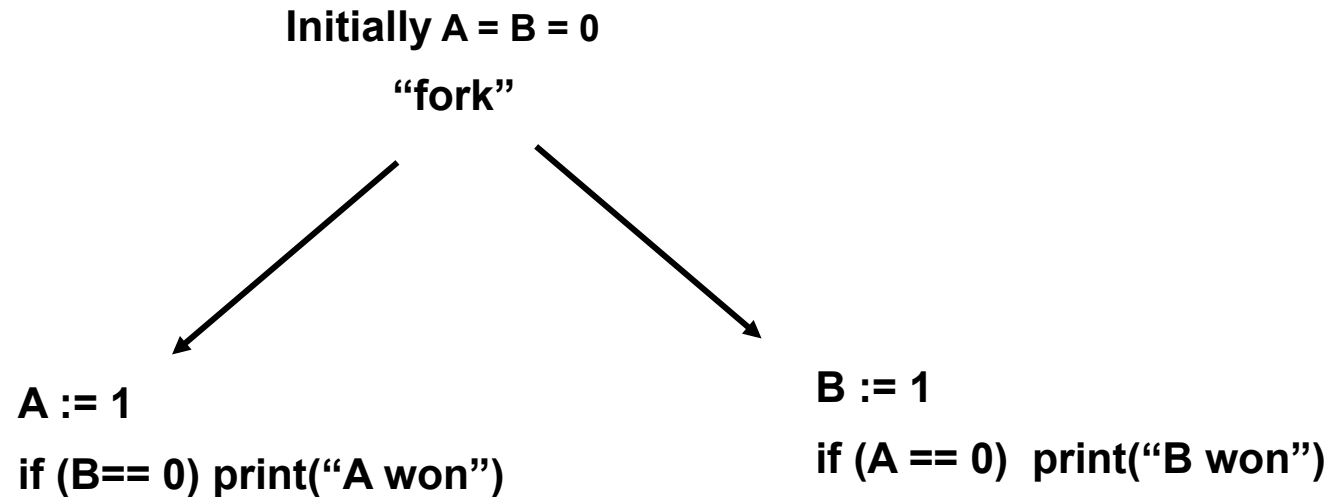


Trick question

What value will be printed?

- 0
- 1
- Undefined (either 0 or 1)

Dekker's Algorithm



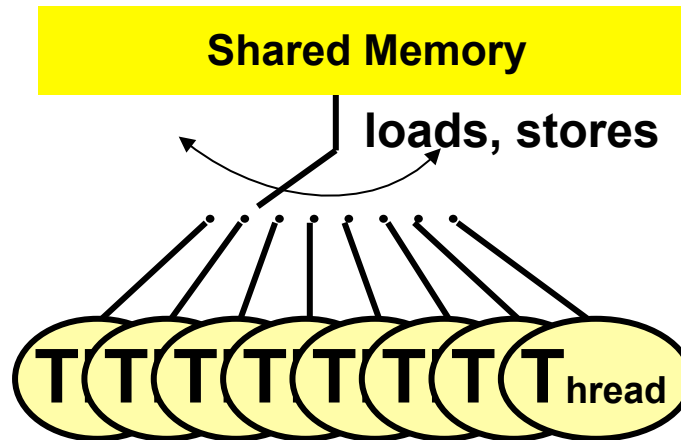
Q: Is it possible that both A and B win?



Memory Ordering

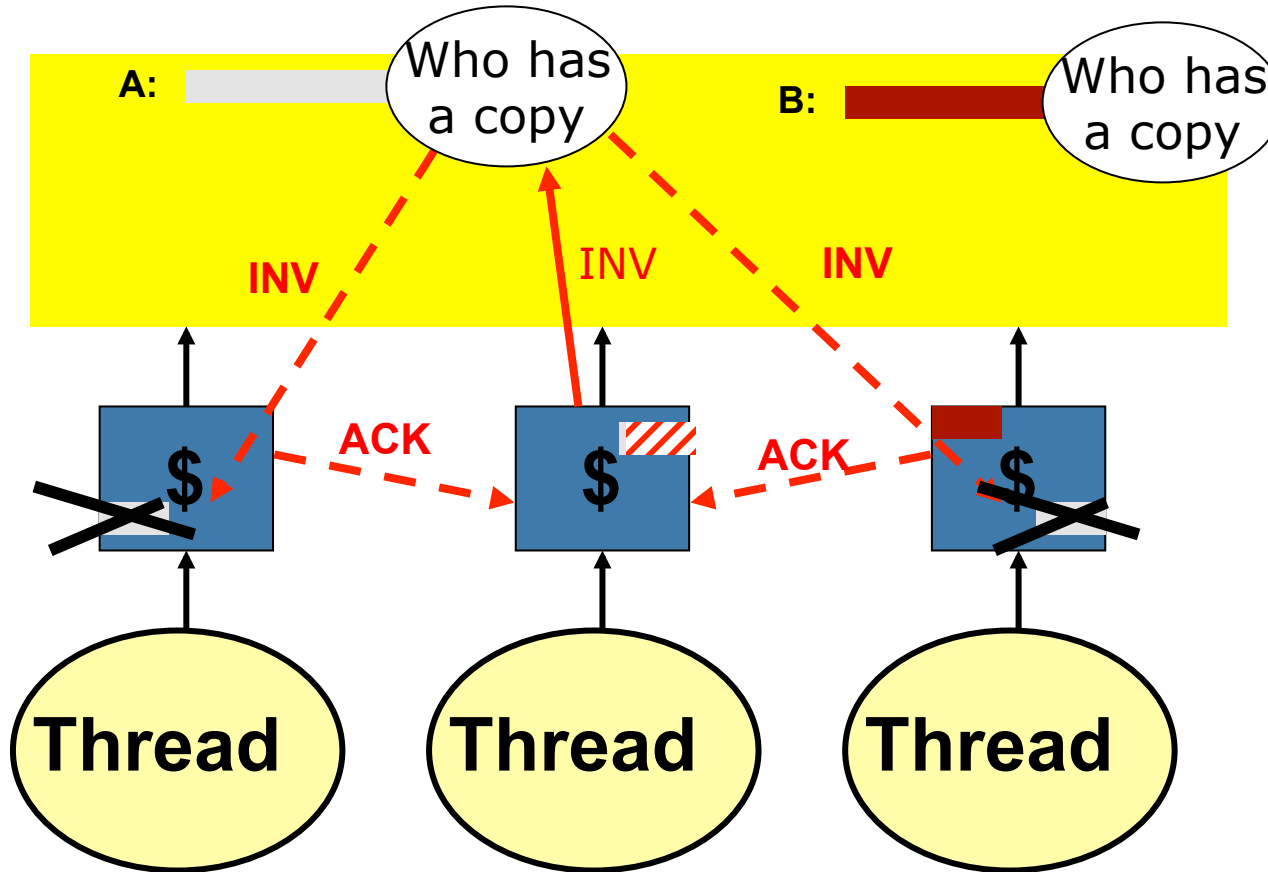
- Defines the [observable] memory order: *If a thread has seen that A happened before B, what order may other threads observe?*
- Is a "contract" between the HW and SW guys
- Without it, you can not say much about the result of a parallel execution

“The intuitive memory order” Sequential Consistency (Lamport)



- ✱ Global order achieved by *interleaving* all memory accesses from different threads
- ✱ “Programmer’s intuition is maintained”
 - Flag synchronization? Yes
 - Store causality? Yes
 - Does Dekker work? Yes
- ✱ Unnecessarily restrictive ==> performance penalty

One implementation of SC in dir-based coherence



Read A
Read A

...

...

Read X
Read A

...

Write A
Read C

Read B

Read X must complete before starting Read A

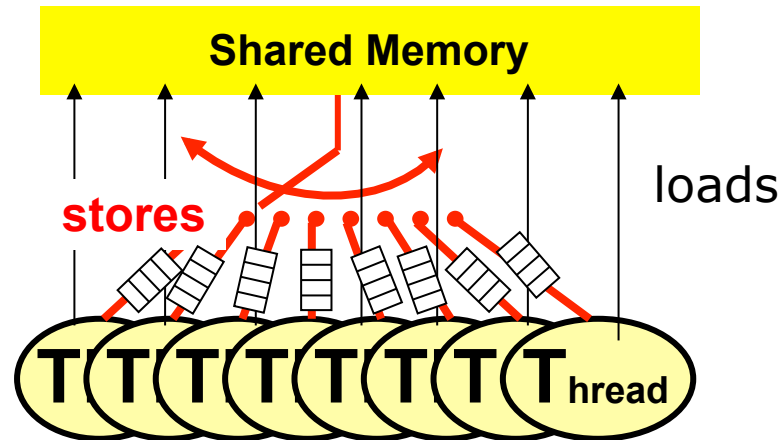
Read A

Must receive all ACKs before continuing

Multiprocessors 33

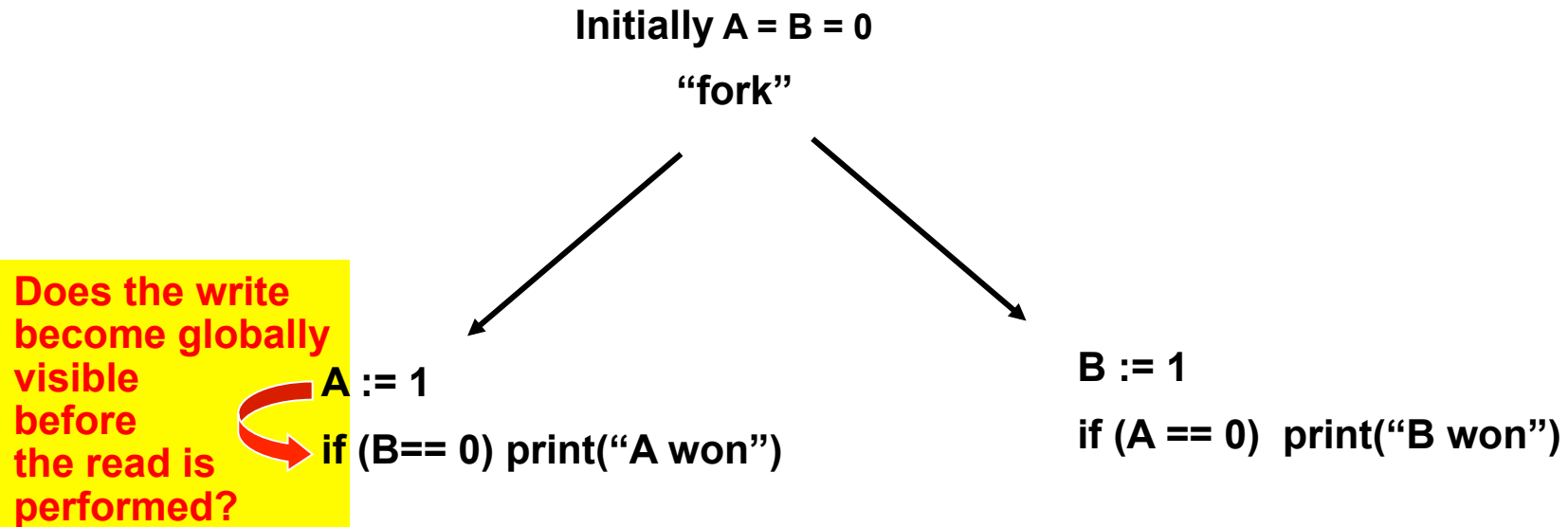
“Almost intuitive memory model”

Total Store Ordering [TSO] (P. Sindhu)



- ✱ Global *interleaving* [order] for all stores from different threads (own stores excepted)
- ✱ “Programmer’s intuition is maintained”
 - Flag synchronization? Yes
 - Store causality? Yes
 - Does Dekker work? No
- ✱ Unnecessarily restrictive ==> performance penalty

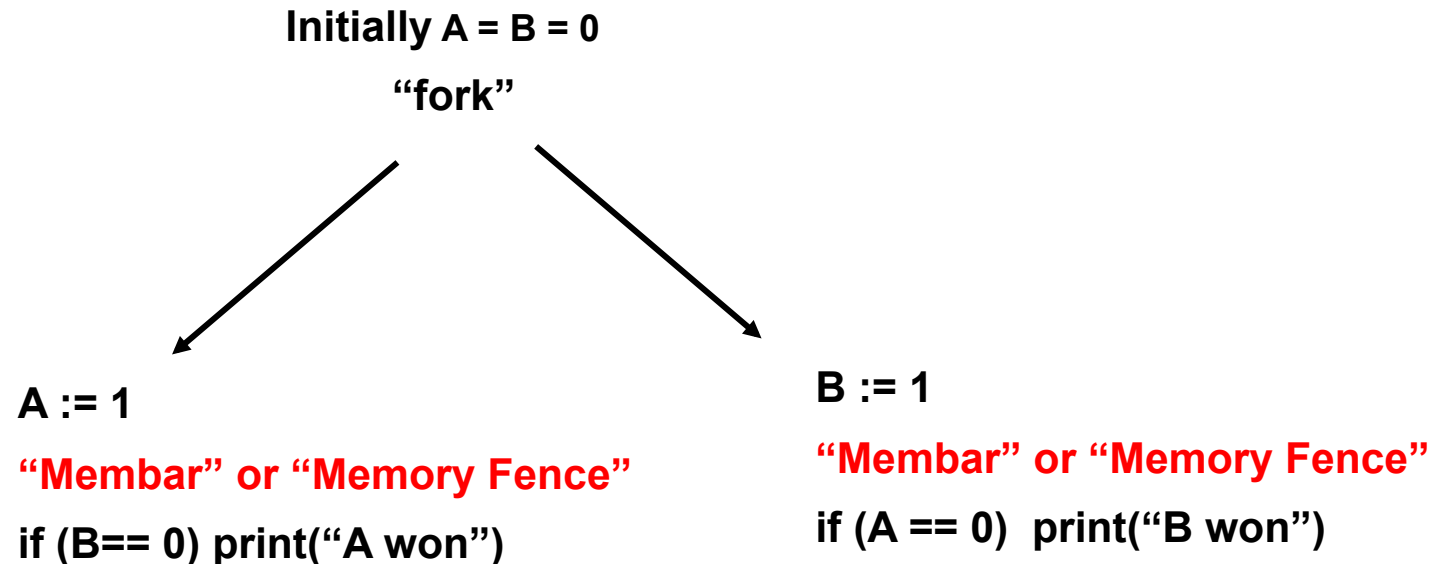
Dekker's Algorithm, TSO



Q: Is it possible that both A and B wins?

- Left: The read (i.e., test if $B == 0$) can bypass the store ($A := 1$)
- Right: The read (i.e., test if $A == 0$) can bypass the store ($B := 1$)
- both loads can be performed before any of the stores
- yes, it is possible that both wins
- → Dekker's algorithm breaks

Dekker's Algorithm for TSO



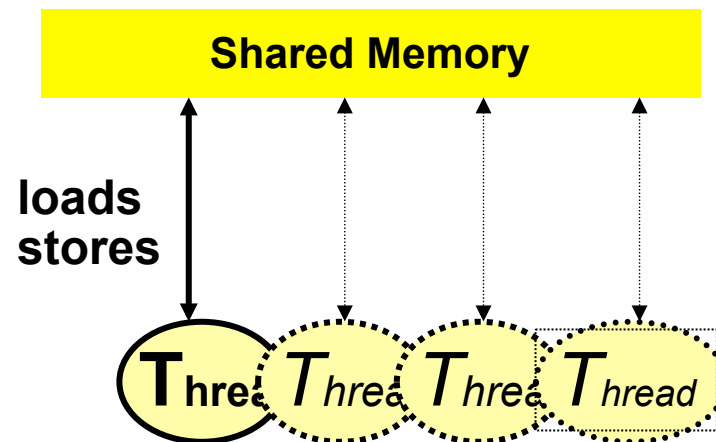
Q: Is it possible that both A and B win?

Membar: The read is started after all previous stores have been "globaly ordered"

→ behaves like SC

→ Dekker's algorithm works!

Weak/release Consistency (M. Dubois, K. Gharachorloo)



- **Most accesses are unordered**
- **“Programmer’s intuition is not maintained”**
 - Flag synchronization? No
 - Store causality? No
 - Does Dekker work? No
- **Global order only established when the programmer explicitly inserts memory barrier instructions**

++ Better performance!!

--- Interesting bugs!!

Multiprocessors 37

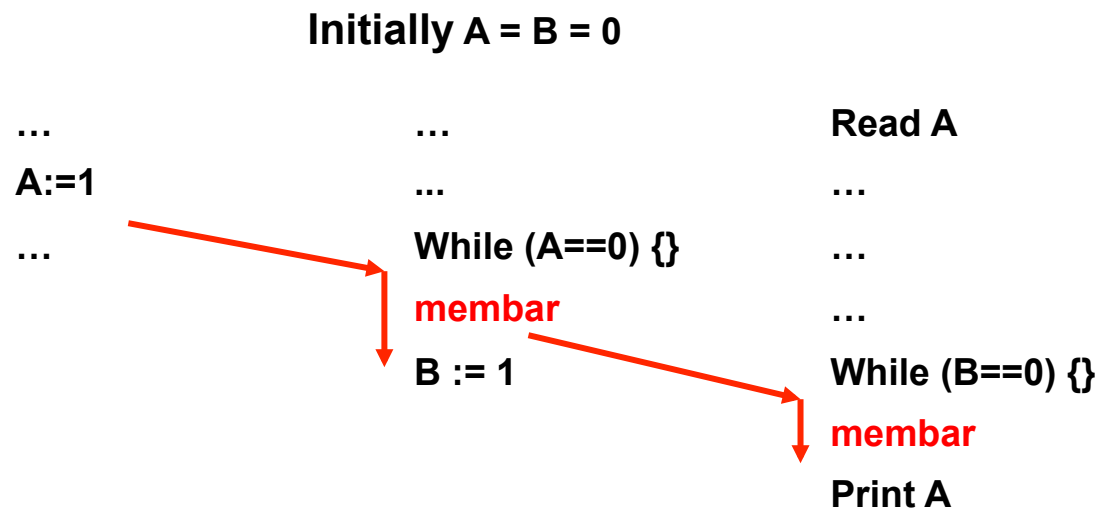
Weak/Release consistency

- New flag synchronization needed

```

A := data;           while (flag != 1) {};
membar;             membar;
flag := 1;          X := A;
  
```

- Dekker's: same as TSO
- Causal correctness provided for this code



Q: What value will get printed?
Answer: 1



Learning more about memory models

Shared Memory Consistency Models: A Tutorial
by Sarita Adve, Kouroush Gharachorloo
in IEEE Computer 1996

RTFM: Read the manual of the system you are working on!
(Different microprocessors and systems supports different memory models.)

Issue to think about:

What code reordering may compilers really do?
Sometimes have to use "volatile" declarations in C!

X86's current memory model

Common view in academia: TSO

If you ask Intel:

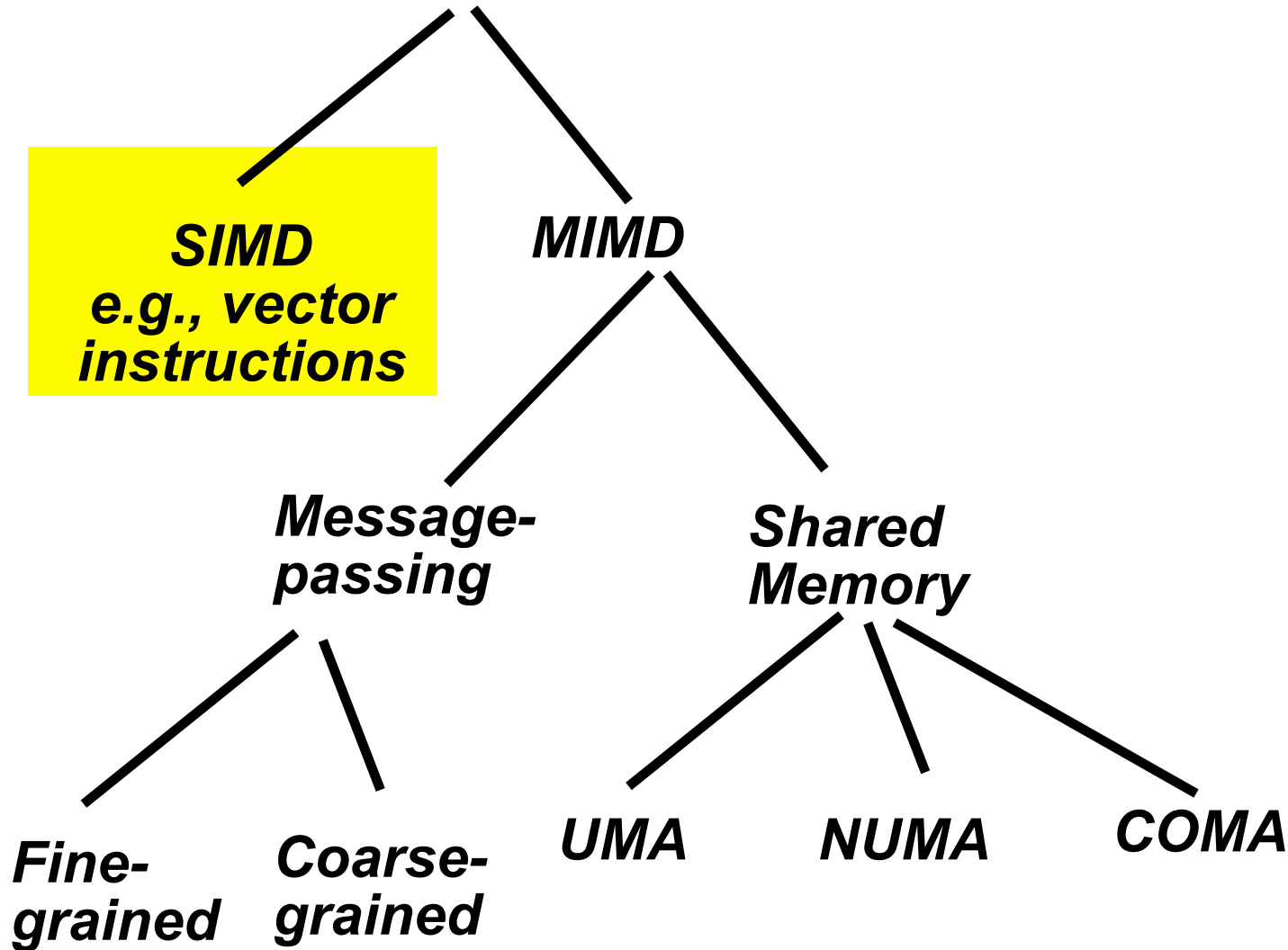
- Processor consistency with causal correctness for non-atomic memory ops
- TSO for atomic memory ops

- Video presentation:

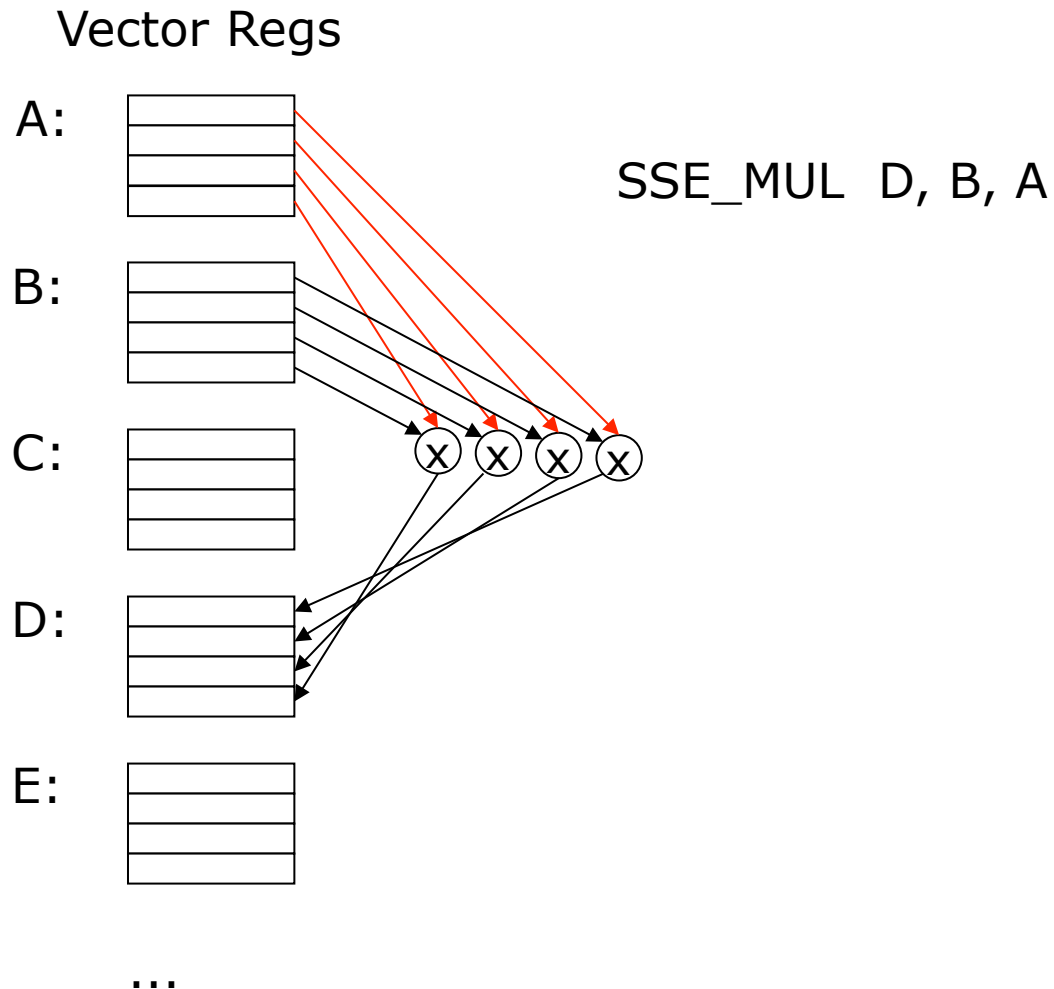
<http://www.youtube.com/watch?v=WUfvvFD5tAA&hl=sv>



A few words about SIMD



Examples of vector instructions





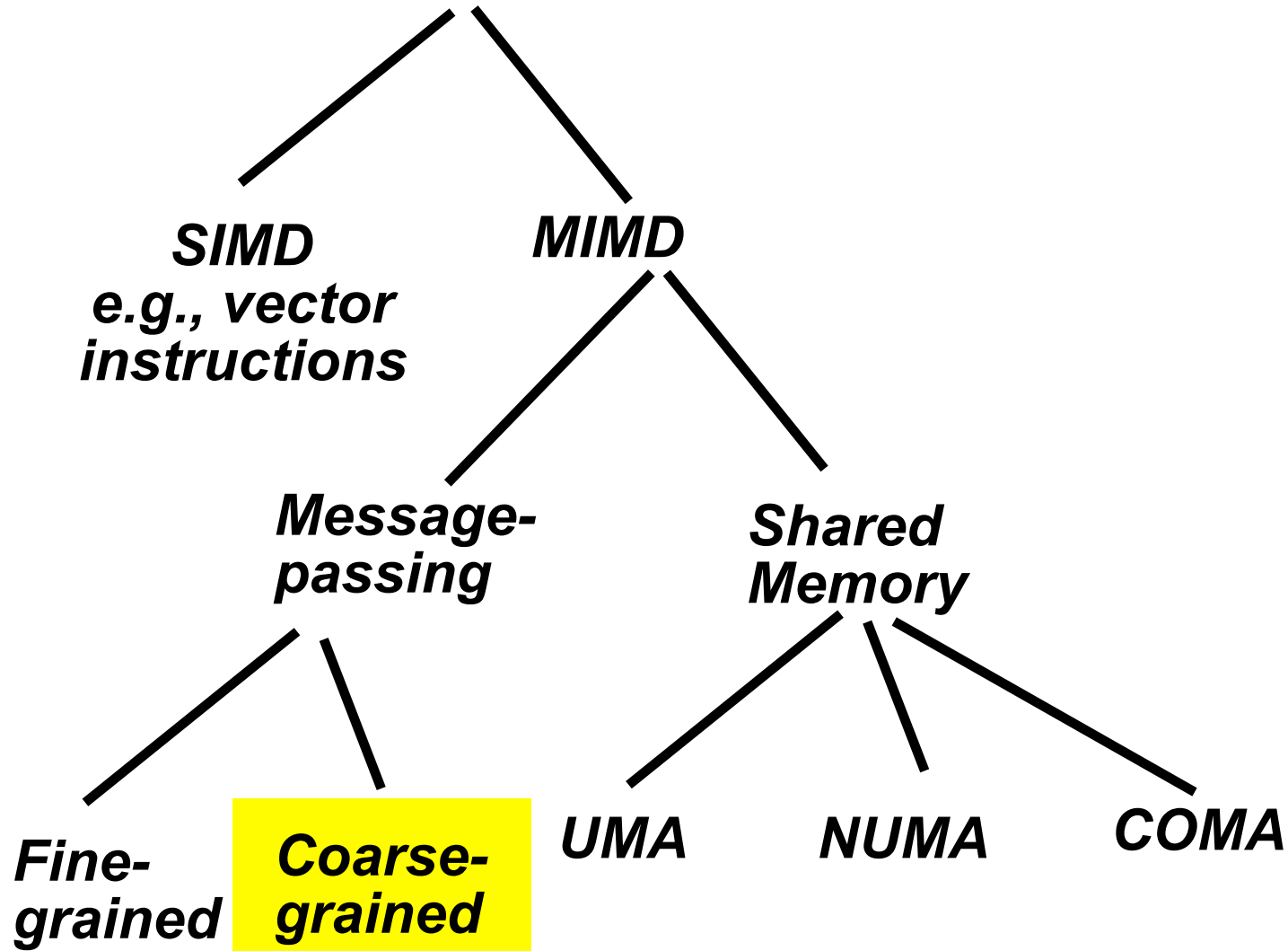
x86 Vector instructions

- MMX: 64 bit vectors (e.g., two 32bit ops)
- SSE: 128 bit vectors(e.g., four 32 bit ops)
- AVX: 256 bit vectors(e.g., eight 32 bit ops)
(in Sandy Bridge, ~y2011)
- Xeon Phi: 512 bit vectors

- GPUs: Vector-ish instructions
A bit more general for "diverge code"

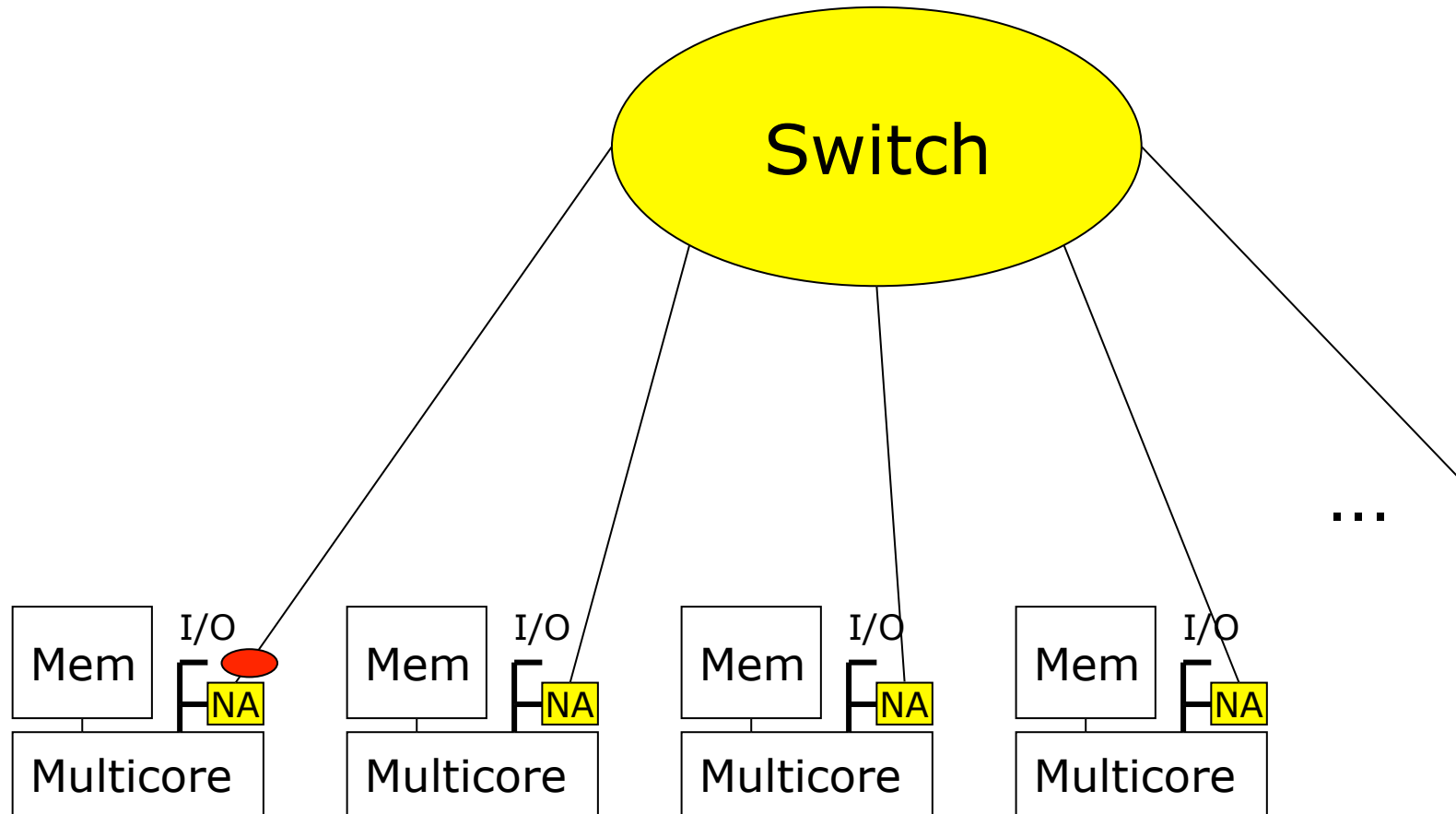


A few words about Message-passing





A scalable "supercomputer"



```
X = vec[i];  
MPI_send(X, to_dest);  
...
```

```
...  
MPI_receive(Y, from_source);  
print (Y);
```

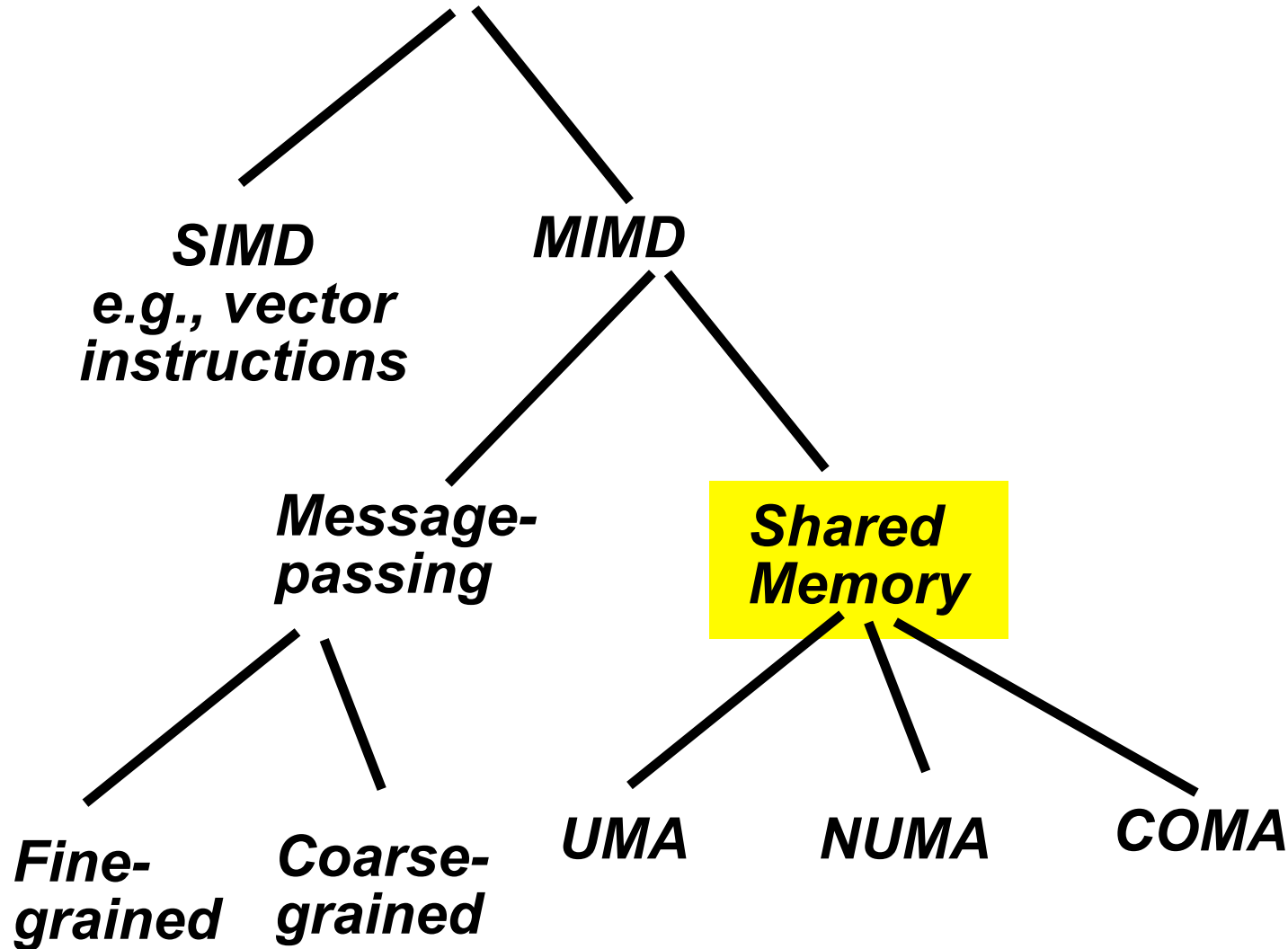


MPI inside a multicore?

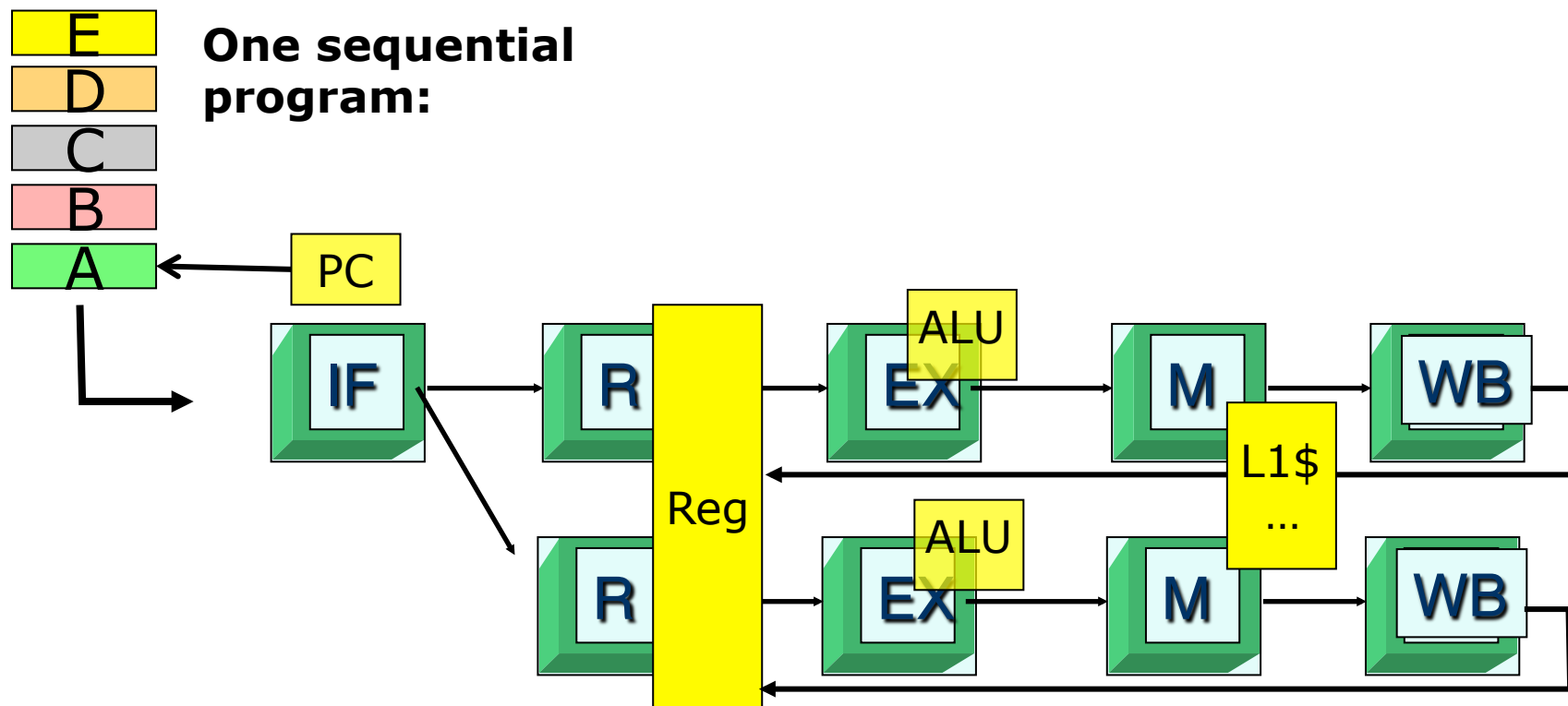
- MPI can be implemented on top of coherent shared memory
- Coherent shared memory can not [cheaply] be implemented on top of MPI
- Many options for parallelism within a "node":
 - ✱ OpenMP
 - ✱ MPI
 - ✱ Posix threads
 - ✱ ...



A few words about simultaneously multithreading (SMT) or “Hyper-threading”



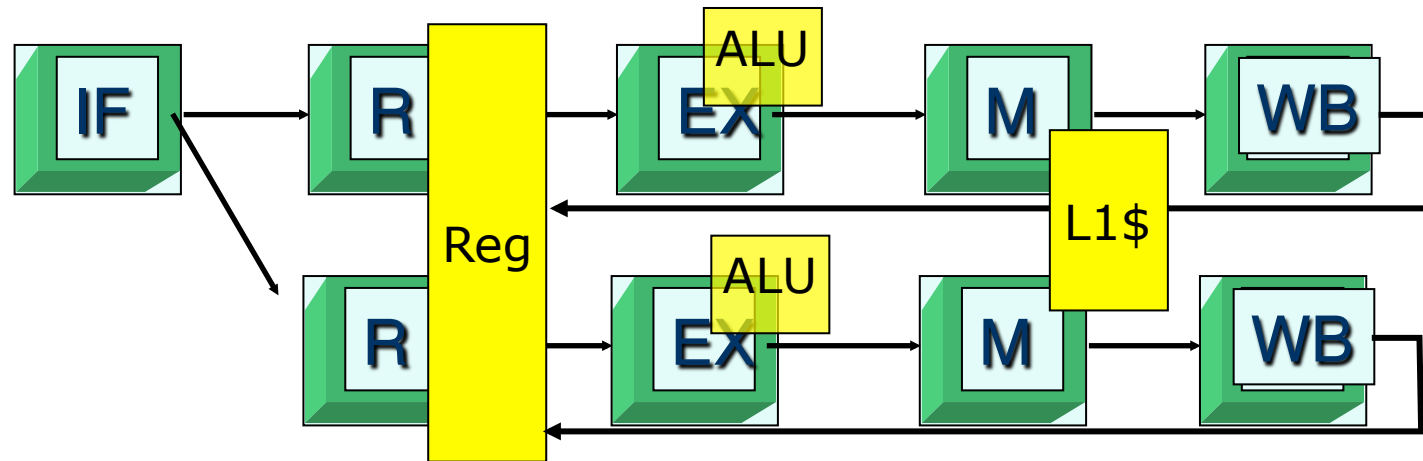
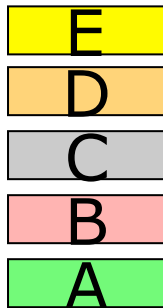
A 5-stage 2-way superscalar pipeline





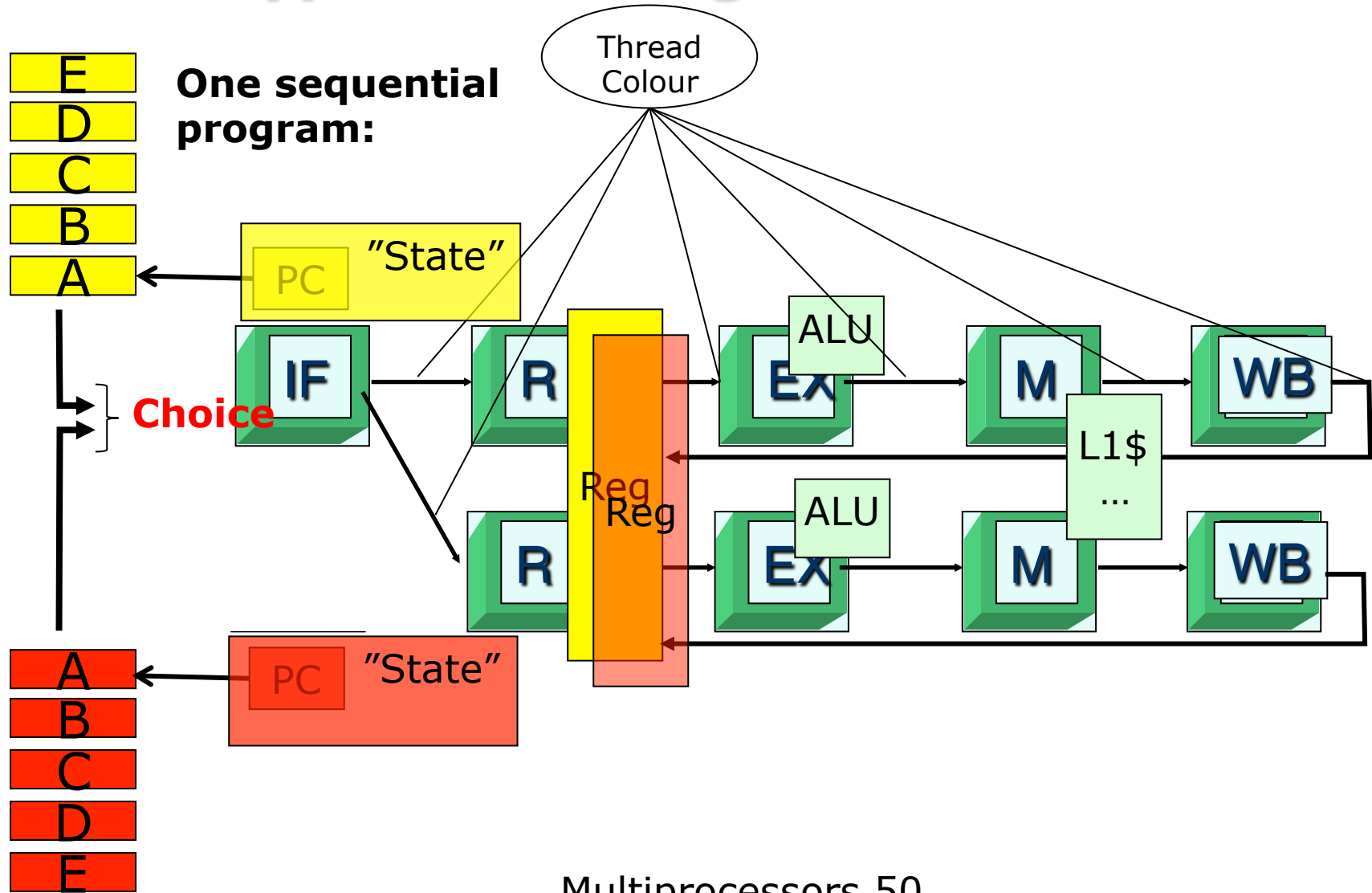
A 5-stage superscalar pipeline

One sequential
program:





A 5-stage 2-way superscalar pipeline, Simultaneously Multithreaded 2-ways (SMT) a.k.a. "HyperThreading"





Choosing between different threads

- Fixed interleaving (Xeon Phi, HEP 1982!!, ...)
 - ✱ Each of N threads executes one instruction every N :th cycles
 - ✱ If thread is not ready to go during its slot \rightarrow bubble
- Hardware-controlled thread scheduling
 - ✱ E.g., hardware keeps track of which threads are ready to go (Niagra-1)
 - ✱ E.g., picks next thread to execute based on hardware priority scheme (\sim Hypertreading)
 - ✱ I-count: Chose the thread with least Instr in-flight
 - ✱ Course-grained: Run one thread until it "blocks"



How are we doing?

- Create and explore locality:
 - ✓ a) Spatial locality
 - ✓ b) Temporal locality
- Create and explore parallelism
 - ✓ a) Instruction level parallelism (ILP)
 - ✓ b) Thread level parallelism (TLP)
 - c) Memory level parallelism (MLP)
- Speculative execution
 - a) Out-of-order execution
 - b) Branch prediction
 - c) Prefetching