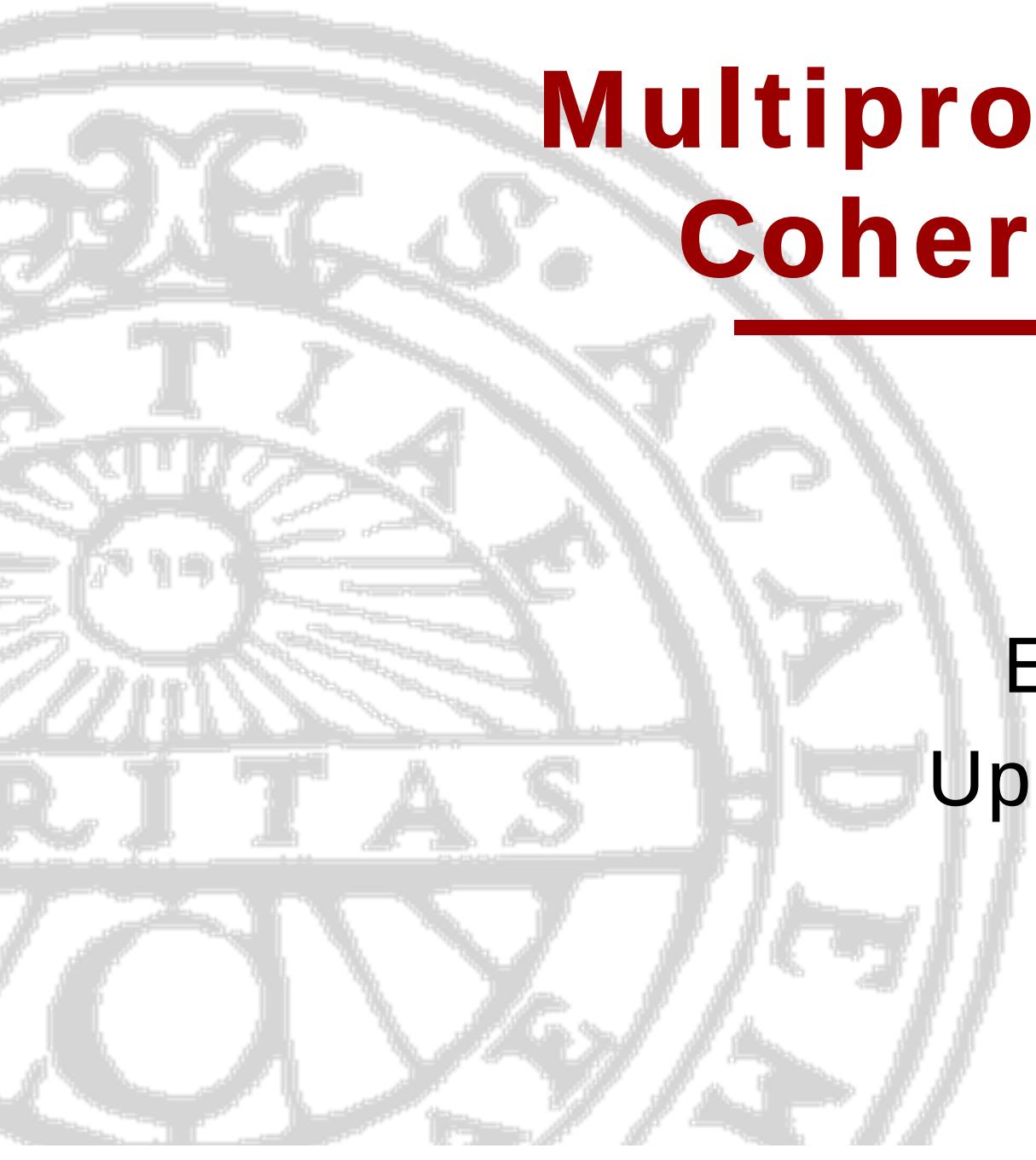


# Multiprocessors and Coherent Memory

Erik Hagersten
Uppsala University

UPPSALA UNIVERSITET

# Summing up Coherence

Good intuition, but too strong definition!

*There can be many copies of a datum, but only one value*

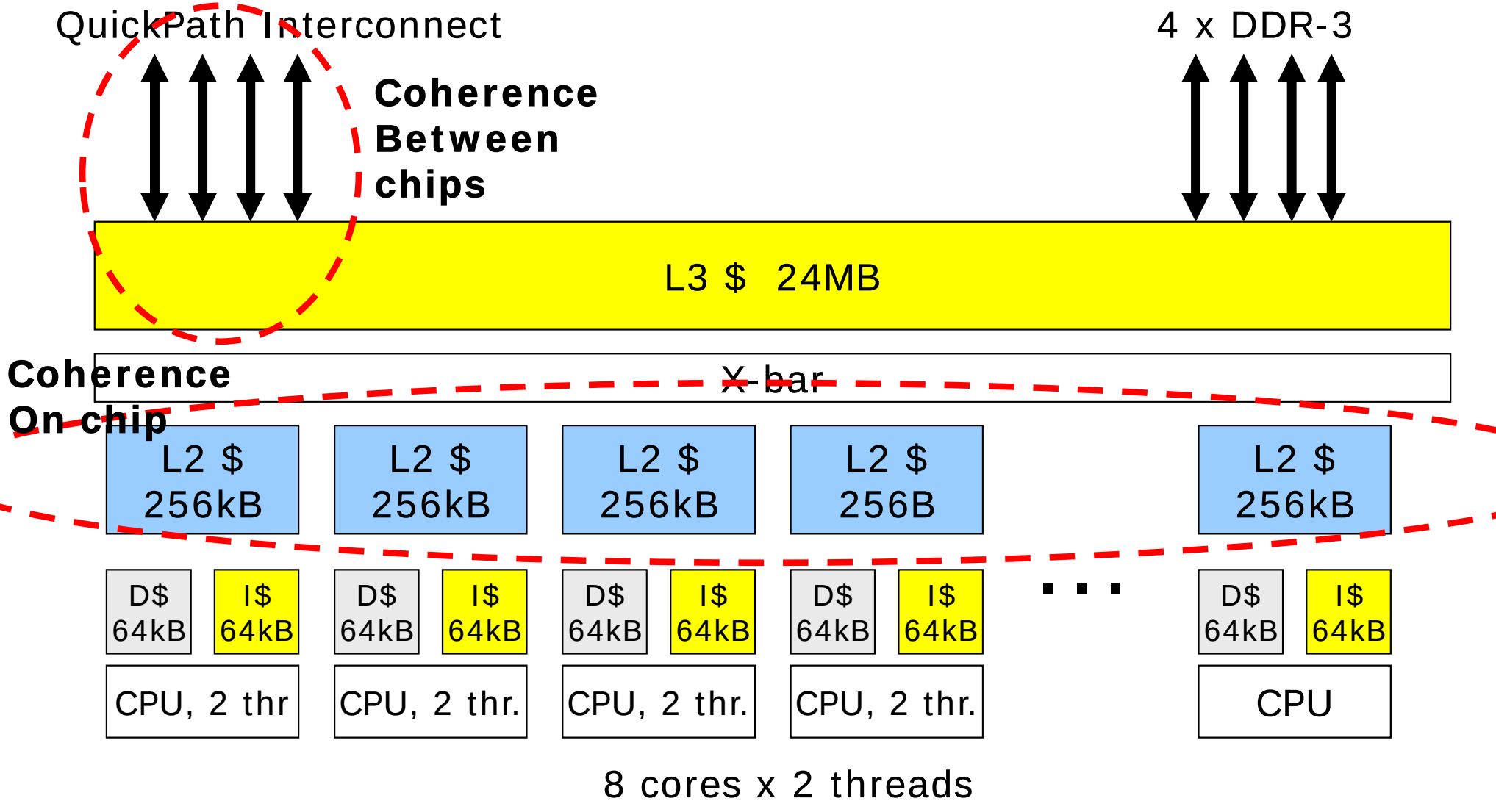
*There is a single global order of value changes to each datum*

*After the computer stops, all copies should have the same value*

Thread1={1,2,3,4,5,6,7…} Thread2={1,4,7…} ~~Thread3={1,8,7…}~~

AVDARK 2013

# Where does coherence matter?

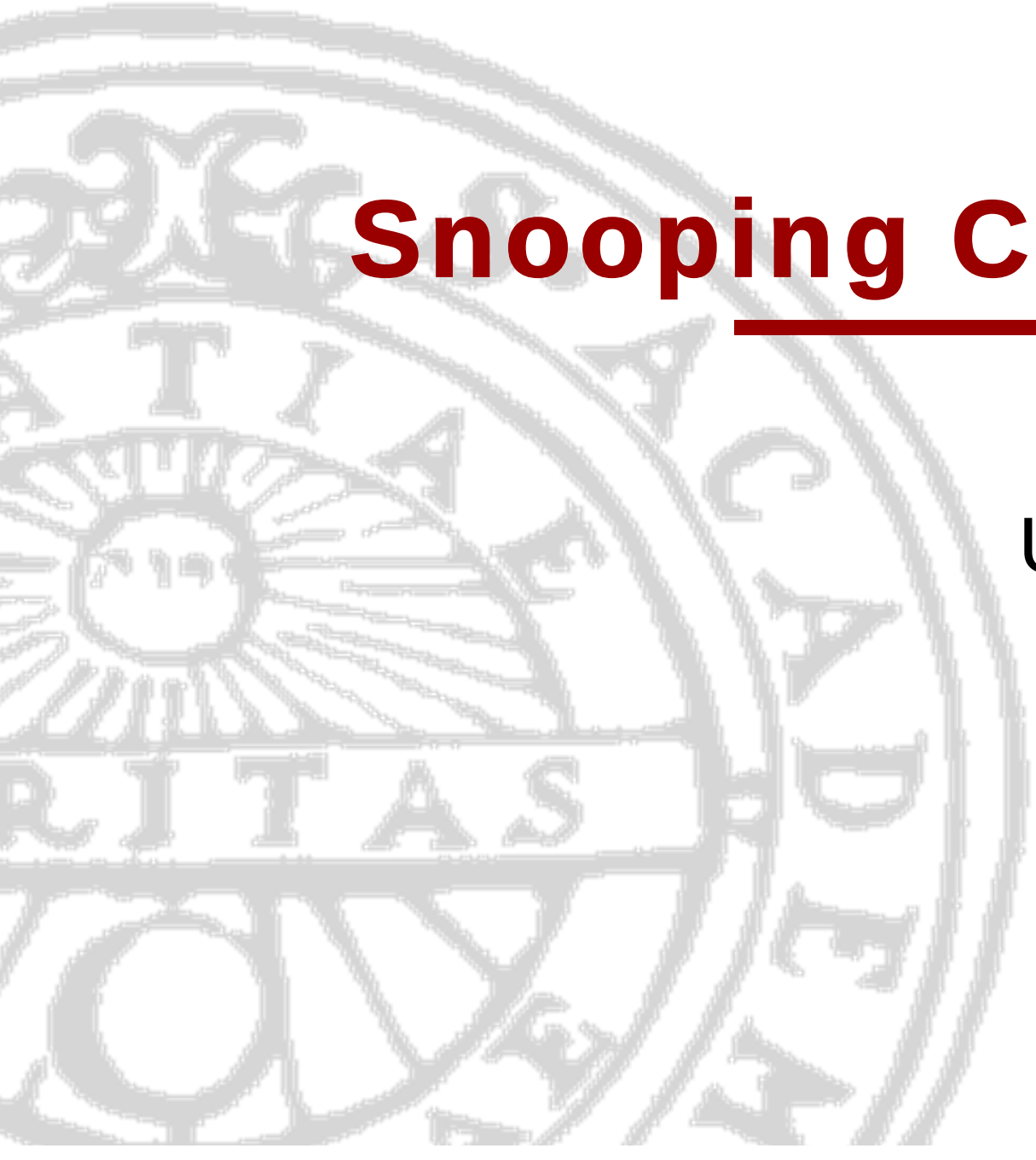


UPPSALA UNIVERSITET

# Summary Coherence

- Coherent shared-memory programming model requires coherence
- (There is also non-coherent shared memory, e.g. single-sided MPI, PGAS)
- All threads can read and write shared data.
- Coherent view of the value of a datum
- Often: Coherence is kept per cache line.

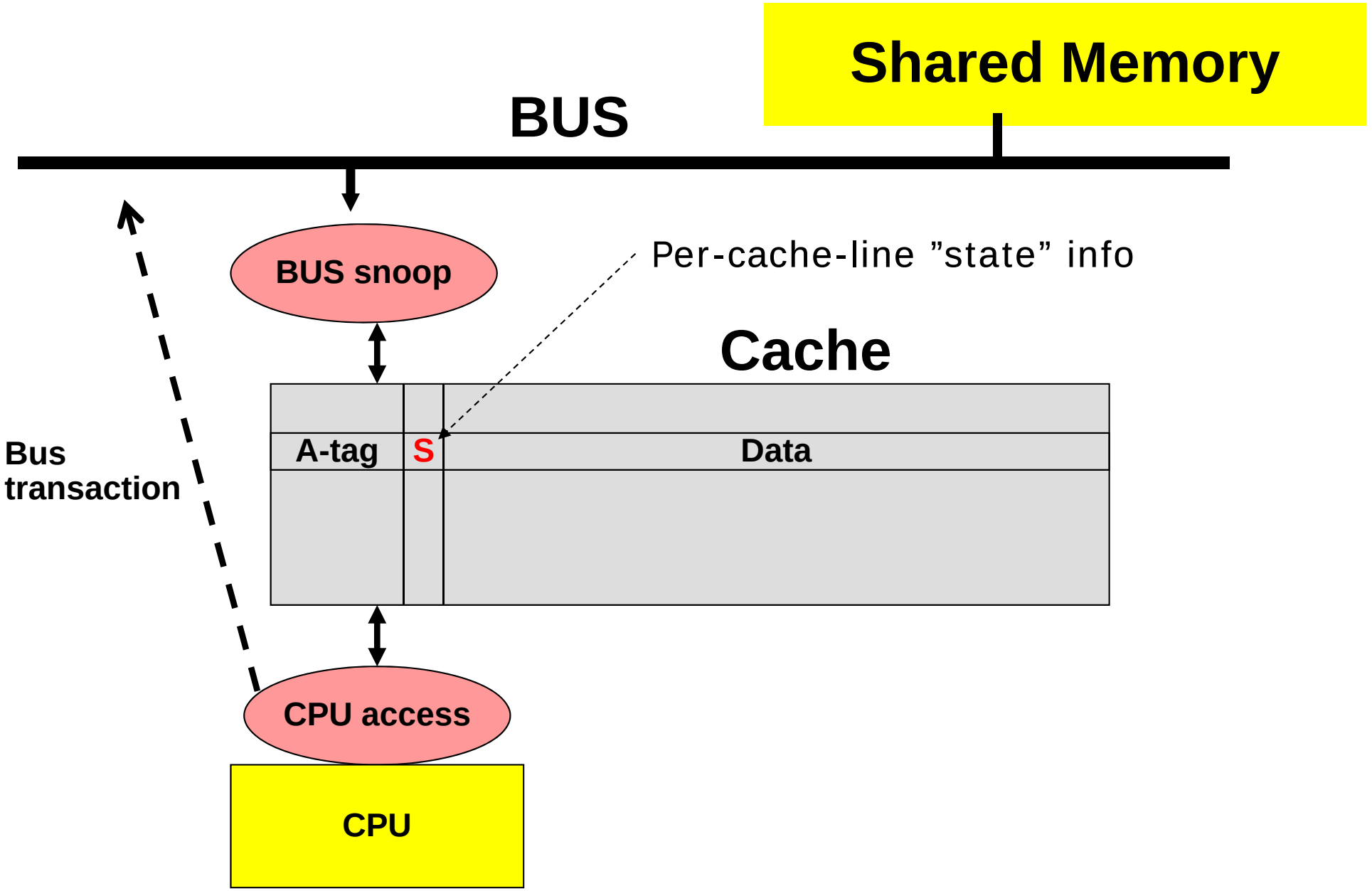
AVDARK 2013



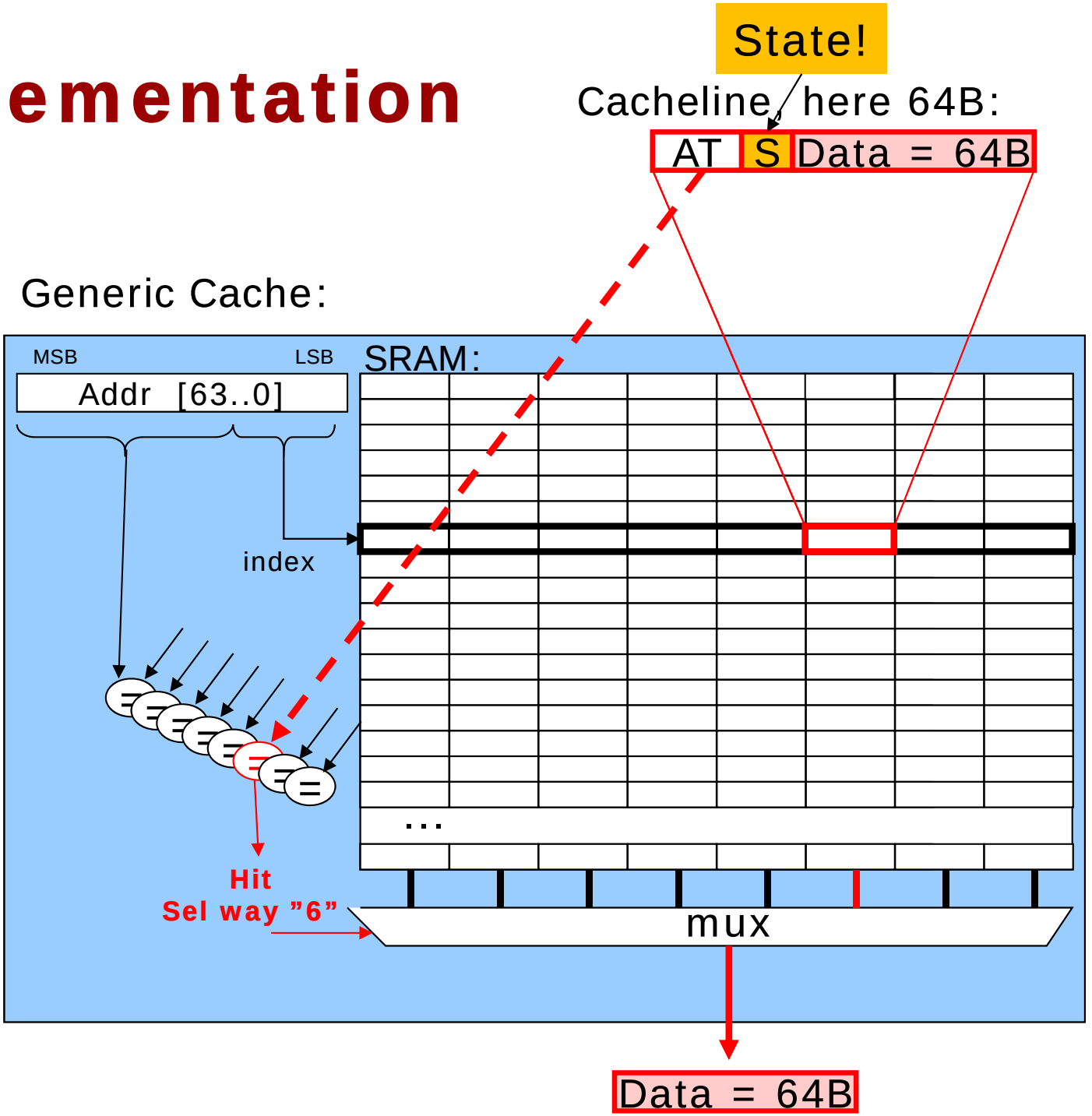
# Snooping Coherence

Erik Hagersten
Uppsala University

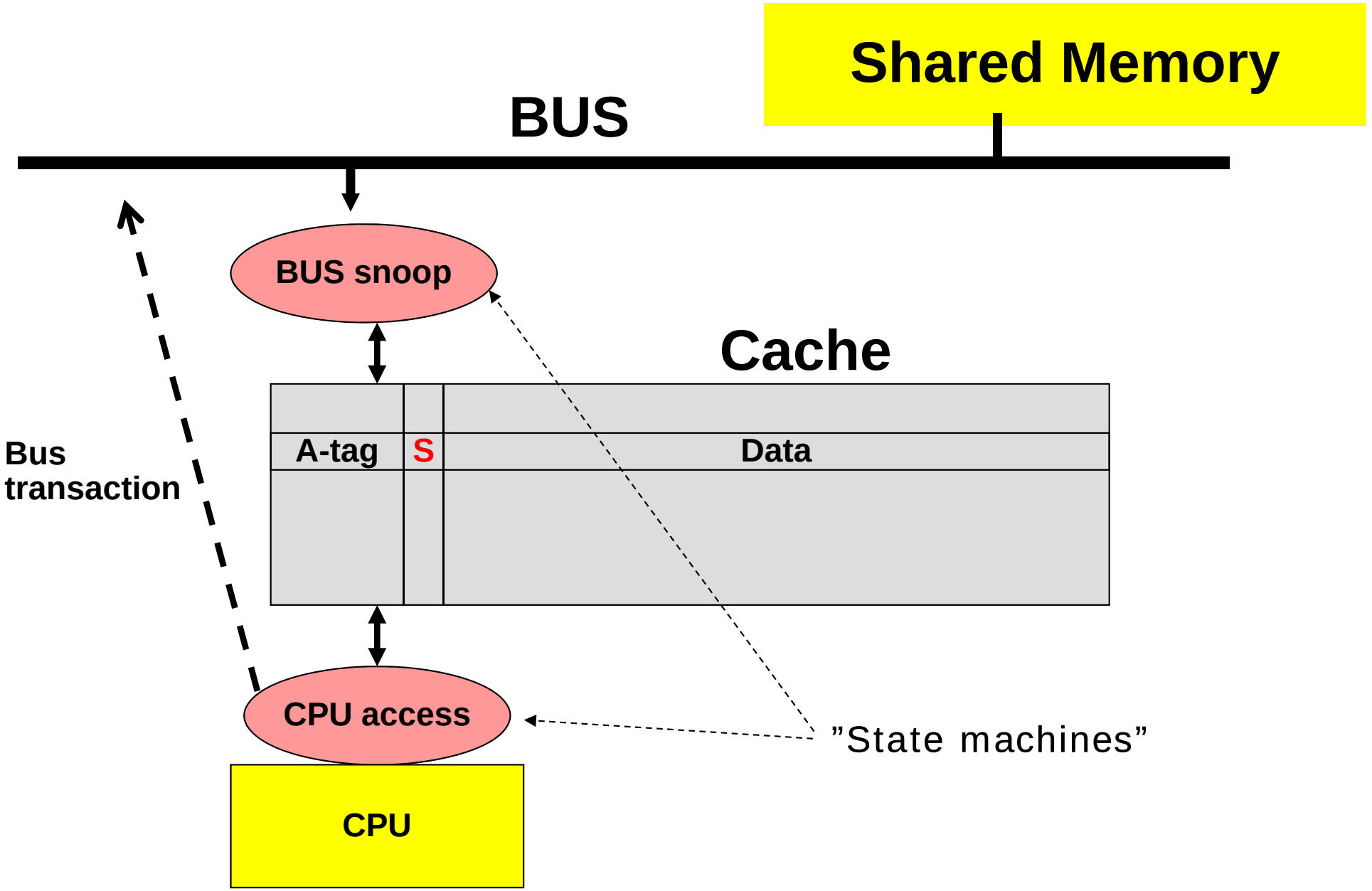
# Snoop-based Protocol Implementation



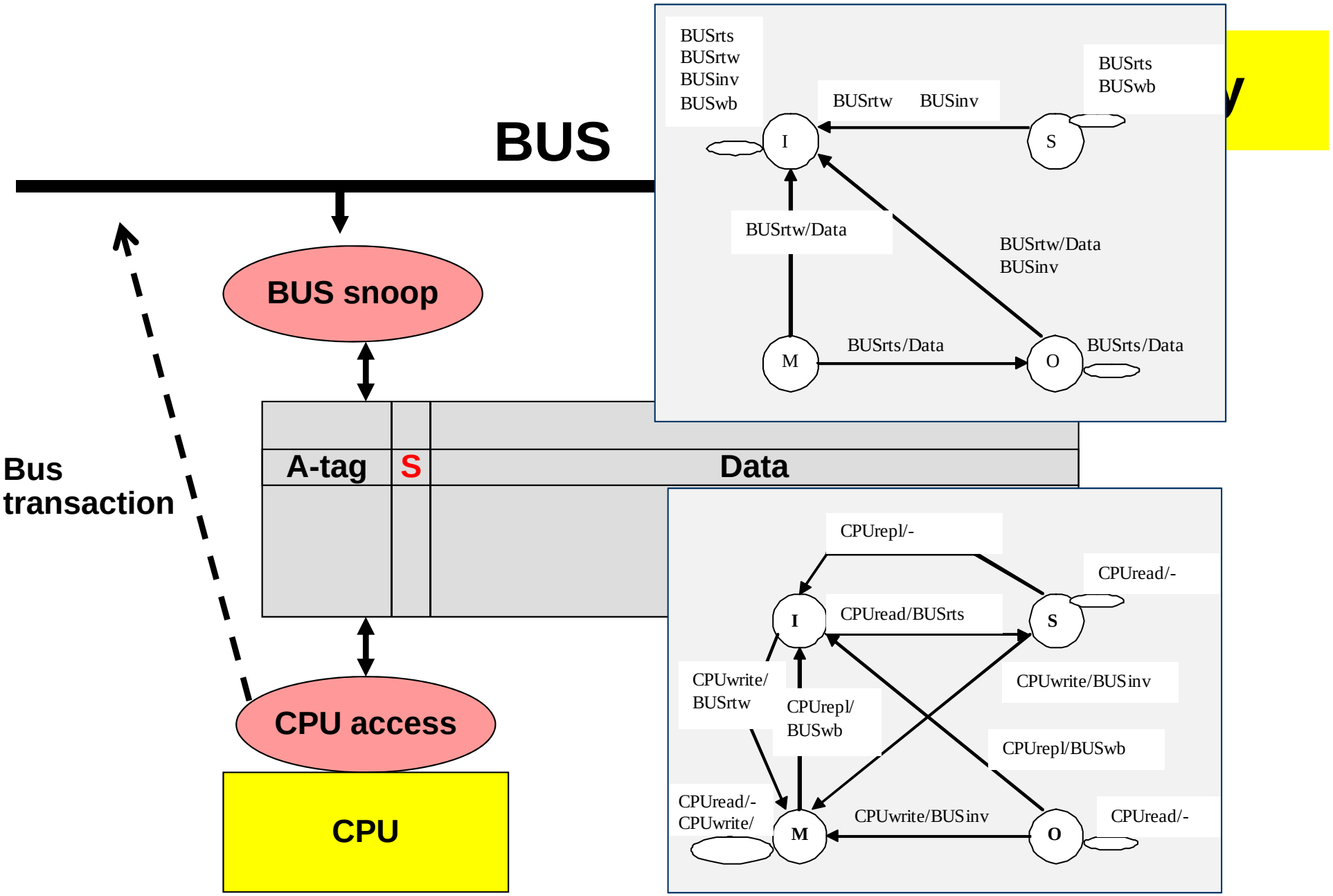
# Cache implementation



# Snoop-based Protocol Implementation



# Snoop-based Protocol Implementation



UPPSALA UNIVERSITET

# Example: MOSI Bus Snoop

**STATES:**

**M – Modified**:

**S – Shared:**

**O – Owner:**

**I – Invalid:**

**BUS TRANSACTIONS FROM OTHERS:**

**BUSrts**

**BUSrtw:**

**BUSinv**:

**BUSwb**

*Dirty: my value differs from the old value in mem. It is my job to update mem.

AVDARK 2013

# Example: MOSI Bus Snoop

**STATES:**

**M – Modified**: My dirty* copy is the only cached copy

**S – Shared:** I have a clean copy, others may also have a copy

**O – Owner:** I have a dirty copy, others may also have a copy

**I – Invalid:** I have no valid copy in my cache (including cache miss)


**BUS TRANSACTIONS FROM OTHERS:**

**BUSrts** ReadtoShare. Reading the data

**BUSrtw:** ReadToWrite. Reading the data with the intention to modify it right away

**BUSinv**: Invalidating other caches copies

**BUSwb**: Writing data back dirty data to memory

**Dirty: my value differs from the old value in mem. It is my job to update mem*


**Input-signal/Reply-signal**

Meaning: If you are in state M and see BUSrts, goto state O and reply with Data

UPPSALA UNIVERSITET

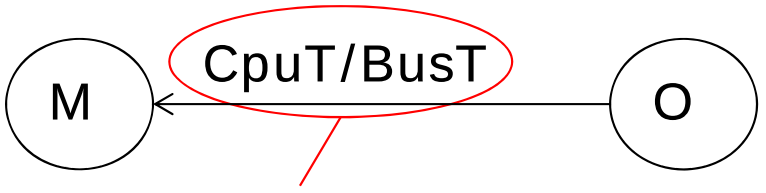
# Example: CPU access MOSI

**FROM MY CPU:**

**CPUread** Caused by a Load instruction

**CPUwrite**: Caused by a Store or Atomic instruction

**CPUrepl**: Caused by a replacement of this cachline (caused by murphy ☺)


**Input-signal/Reply-signal**

Meaning: If you are in state M and see BUSrts, goto state O and reply with Data

# The MOSI Bus Snoop

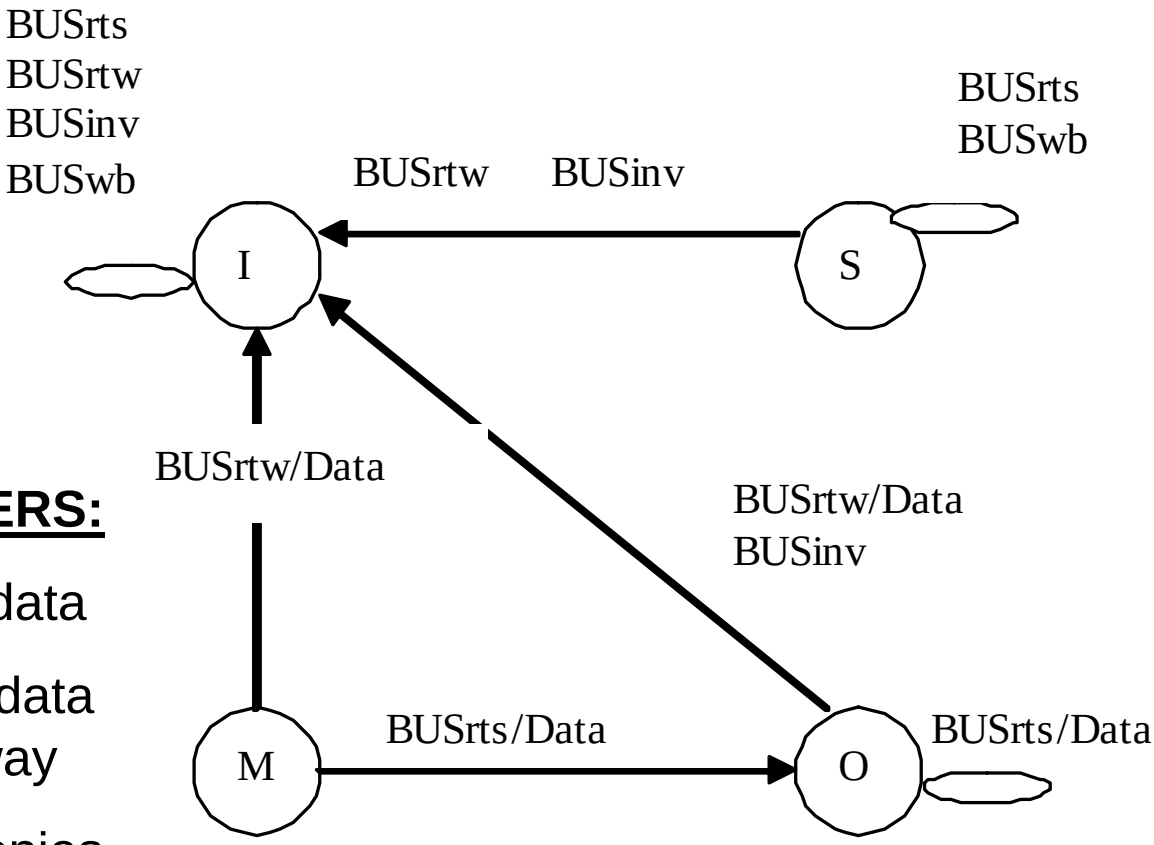

**STATES:**

**M – Modified**: My dirty* copy is the only cached copy

**S – Shared:** I have a clean copy, others may also have a copy

**O – Owner:** I have a dirty copy, others may also have a copy

**I – Invalid:** I have no valid copy in my cache (including cache miss)




**BUS TRANSACTIONS FROM OTHERS:**

**BUSrts** ReadtoShare. Reading the data

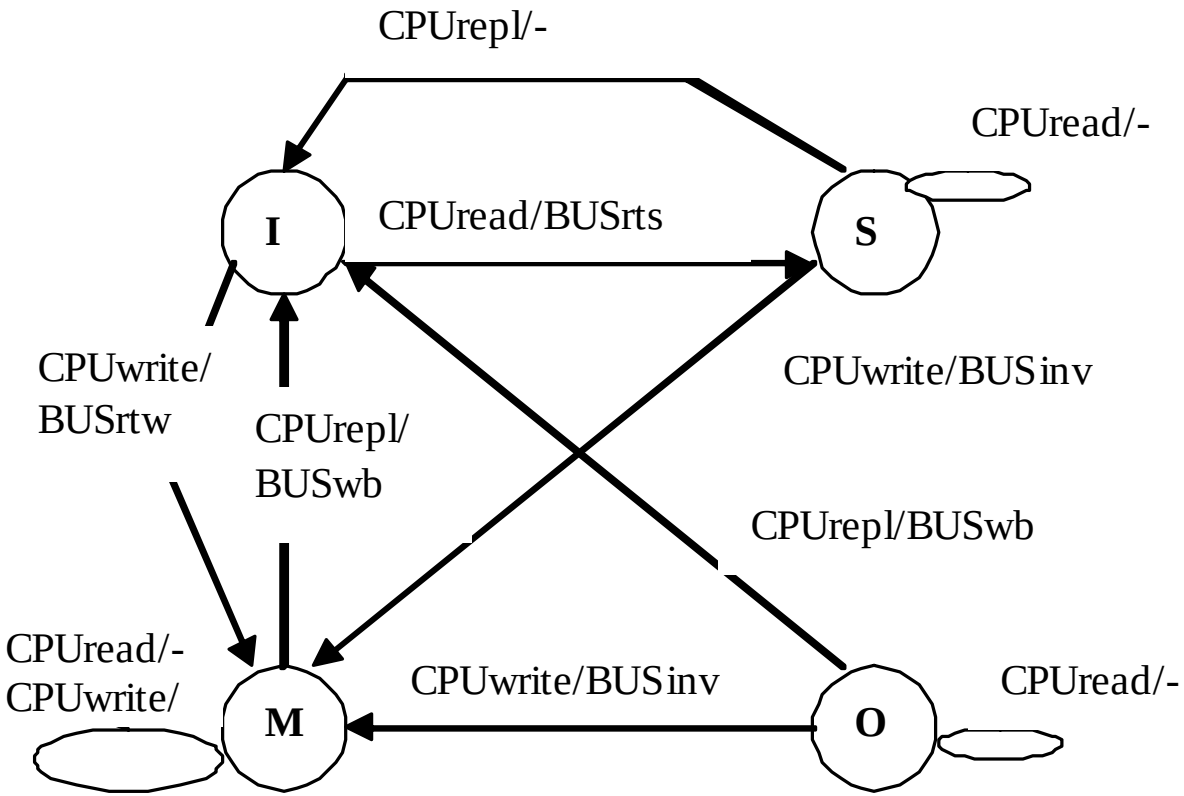
**BUSrtw:** ReadToWrite. Reading the data with the intention to modify it right away

**BUSinv**: Invalidating other caches copies

**BUSwb**: Writing data back to memory

**Dirty: my value differs from the old value in mem*

# The MOSI CPU access

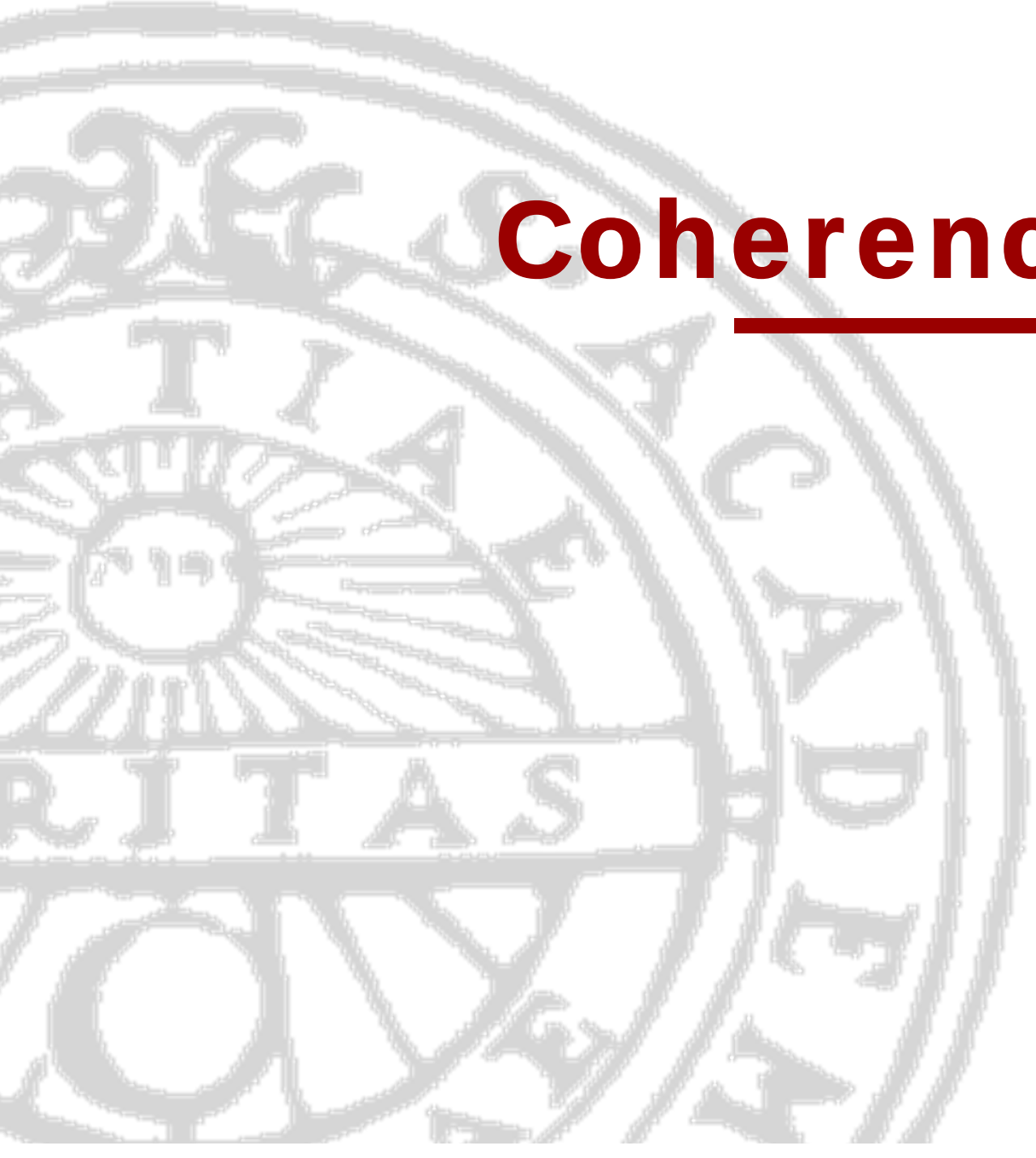



**FROM MY CPU:**

**CPUread** Caused by a Load instruction

**CPUwrite**: Caused by a Store or Atomic instruction

**CPUrepl**: Caused by a replacement of this cachline (caused by murphy ☺)



# Coherence exercises

# Example of a state transition sheet:

| CPU action | Bus Transaction (if any) | State/value after the CPU action | | | | | | Data is provided by [Cache 1, 2, 3 or Mem] (if any) |
|---|---|---|---|---|---|---|---|---|
| | | CPU1 | | CPU2 | | CPU3 | | |
| | | A | B | A | B | A | B | |
| Initially | | I | I | I | I | I | I | |
| CPU3: LD A | RTS(A) | | | | | S/1 | | Mem |
| CPU2: LD A | RTS(A) | | | S/1 | | | | Mem |
| CPU1: LD A | RTS(A) | S/1 | | | | | | Mem |
| CPU3: LDA | — | | | | | | | — |
| | | | | | | | | |

UPPSALA UNIVERSITET

# Example during IRL Class:

All the three RISC CPUs (e.g. cores in a multicore) in a MOSI shared-memory sequentially consistent multiprocessor execute the following code almost at the same time:

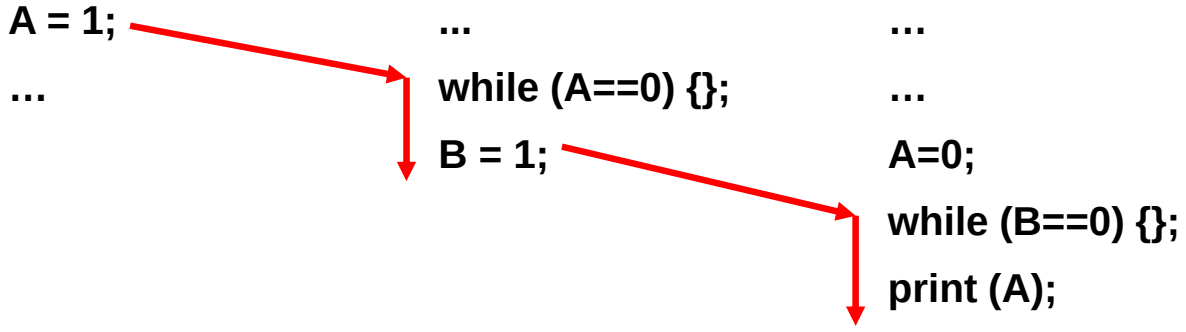
```
CPU1:
     A = 1;

CPU2:

while (A == 0) {};
    B = 1;

CPU3
   A = 0;
   while (B == 0){};
   printf ("%d", A);
```

CPU3 will start slightly ahead of the others and will execute its first memory instruction (LD/ST) before CPU2 starts its first memory instruction, while CPU1 will start last. **After this, the threads will take turns performing one memory instruction each in that order.**
Initially A and B reside in memory only and both have the value 0.
After a long time, show the effects of replacement.




AVDARK 2013

UPPSALA UNIVERSITET

| Mem Instr | Bus | 1:A | 1:B | 2:A | 2:B | 3:A | 3:B | Data |
|---|---|---|---|---|---|---|---|---|
| 3:ST A | RTW(A) | | | | | M/0 | | Mem |
| 2:LD A | RTS(A) | | | S/0 | | O/0 | | $3 |
| 1:ST A | RTW(A) | M/1 | | I | | I | | $3 |
| 3:LD B | RTS(B) | | | | | | S/0 | Mem |
| 2:LD A | RTS(A) | O/1 | | S/1 | | | | $1 |
| 3:LD B | -- | | | | | | | -- |
| 2:ST B | RTW(B) | | | | M/1 | | I | -- |
| 3:LD B | RTS(B) | | | | O/1 | | S/1 | $2 |
| 3:LD A | RTS(A) | | | | | S/1 | | $1 |
| LONG TIME ELAPSE | | | | | | | | |
| 1:REPL A | WB(A) | I | | | | | | $1 |
| 2:REPL A | -- | | | | | | | -- |
| 2:REPL B | WB(B) | | | | I | | | $2 |
| 3:REPL A | -- | | | | | | | -- |
| 3:REPL B | -- | | | | | | | -- |
| | | | | | | | | |
| | | | | | | | | |

AVDARK 2013

# IRL Memory Models

UPPSALA UNIVERSITET

# Dekker's Algorithm (mutual exclusion)

Initially A = B = 0

"fork"

A := 1

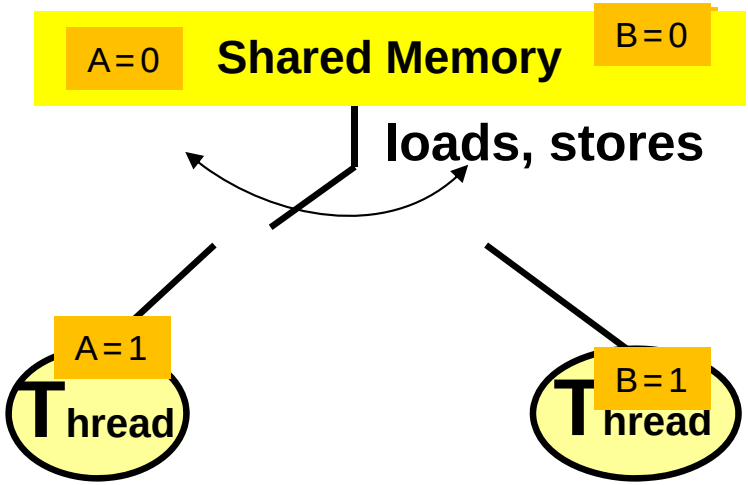
if (B== 0) print("A won")

B := 1

if (A == 0)  print("B won")

Does the write become globally visible before the read is performed?

# Sequential Consistency (Lamport)




**Initially A = B = 0**

**"fork"**

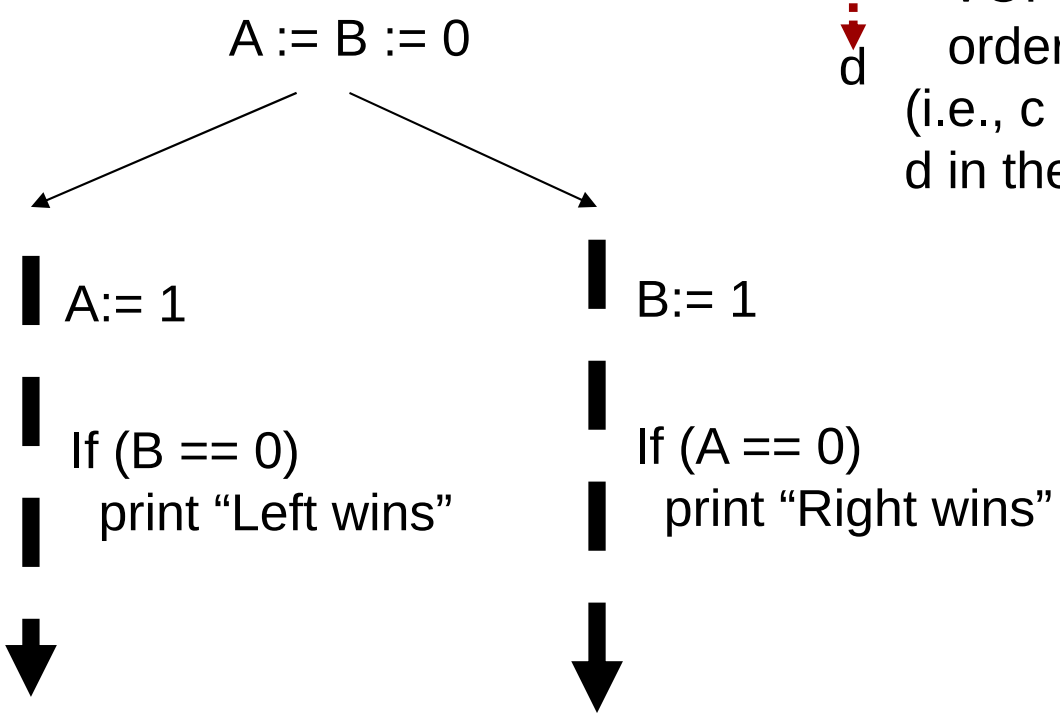
**A := 1**

**if (B== 0) print("A won")**

**B := 1**

**if (A == 0) print("B won")**

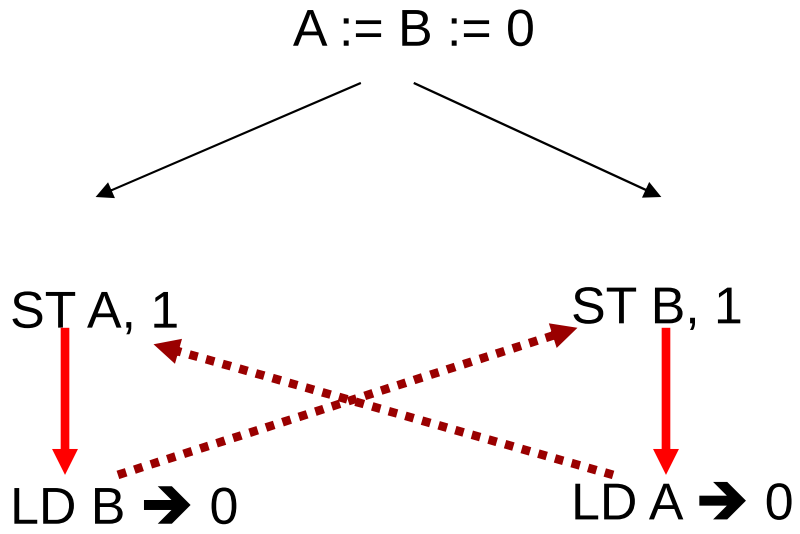
# Can the case “both win” happen under SC?




## Acess graph

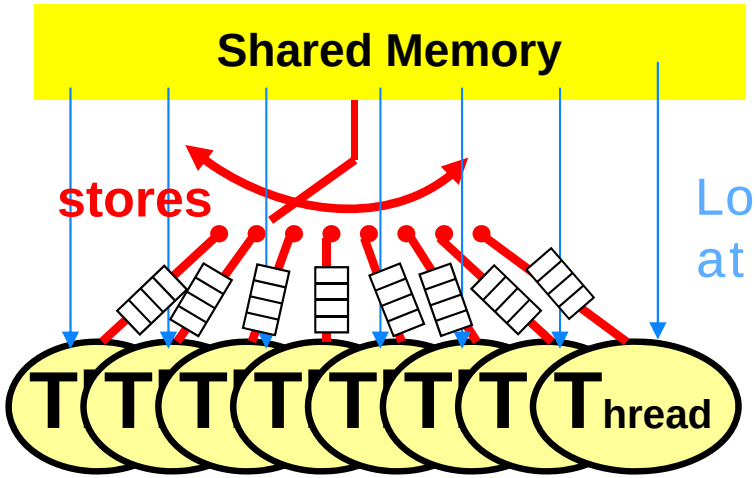
c
d
= VO: Value order: c < d
(i.e., c happened before d in the global order)

a
b
= PO: Program order: a < b
(the order specified by the program)




**Cyclic access graph → Not SC (there is no global order)**


# "Almost intuitive memory model" Total Store Ordering [TSO] (P. Sindhu)

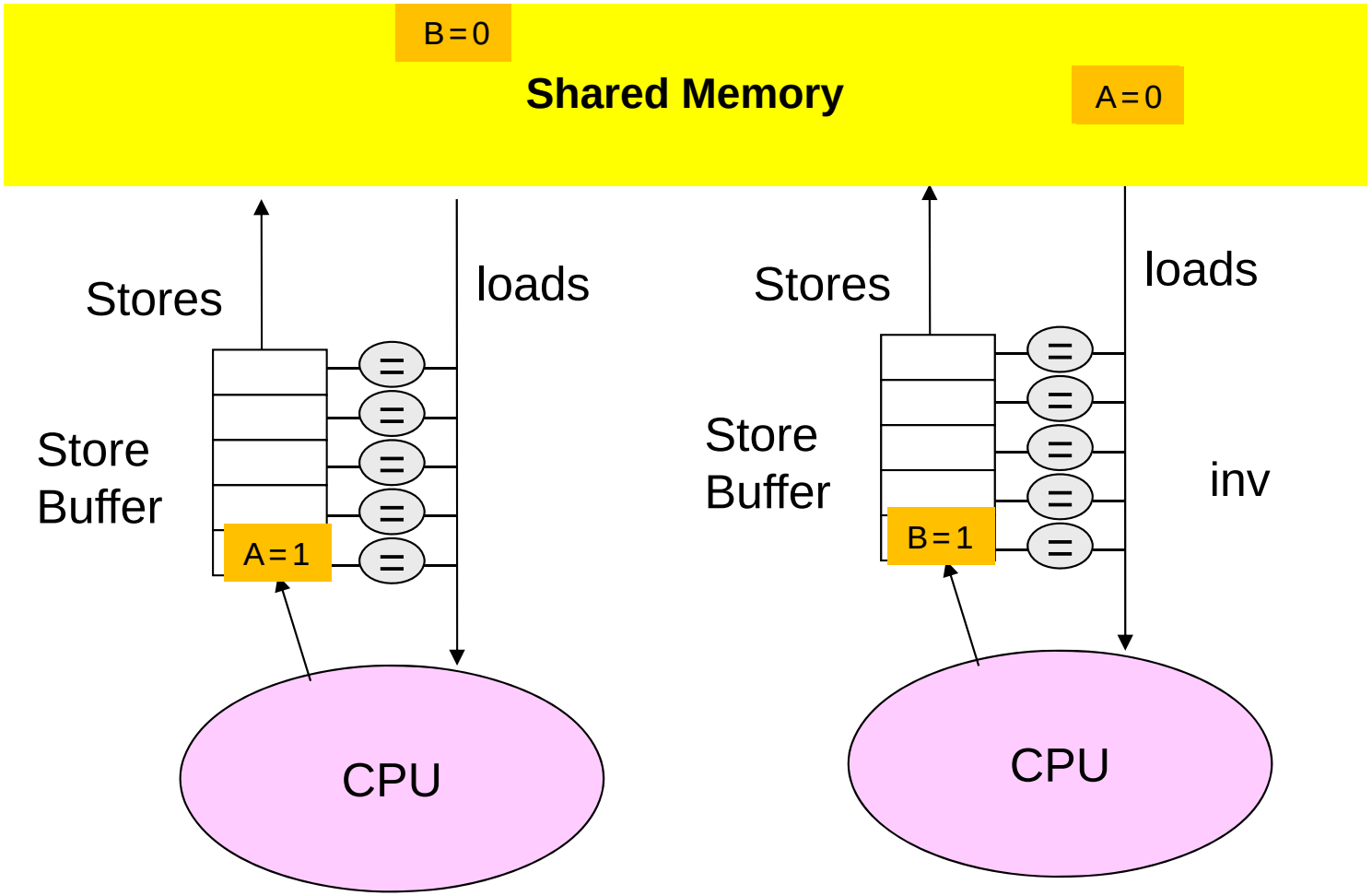



Loads(each thread can have at most one outstanding load)

- Global *interleaving* [order] for <u>all</u> stores from different threads (own stores excepted)

**Q: What happens if the SB gets full?**

# TSO HW Model

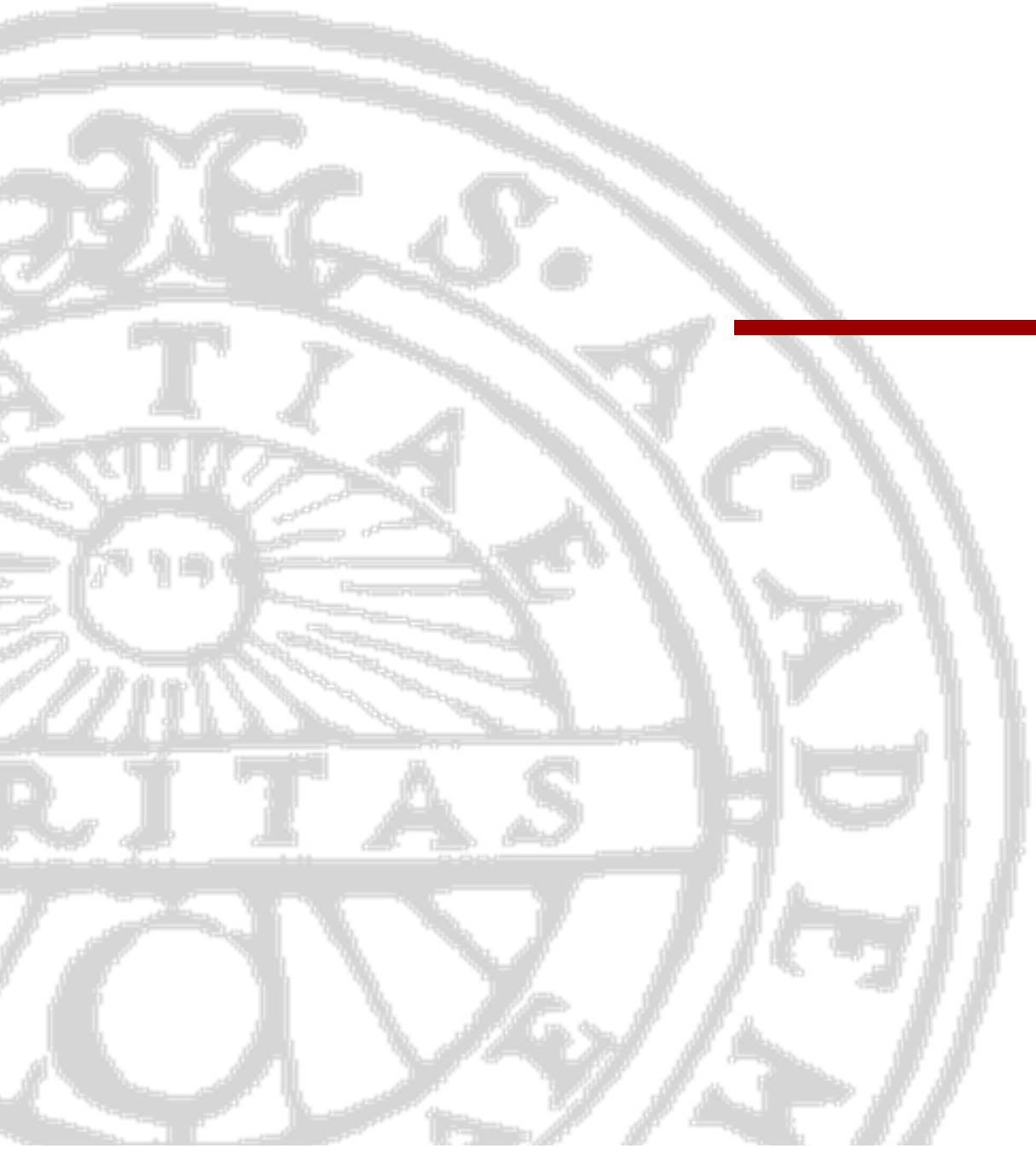



**A = 1**

**if (B== 0) print("A won")**

**B = 1**

**if (A == 0) print("B won")**



# IRL Sync

# A Bad Example: ”POUNDING”

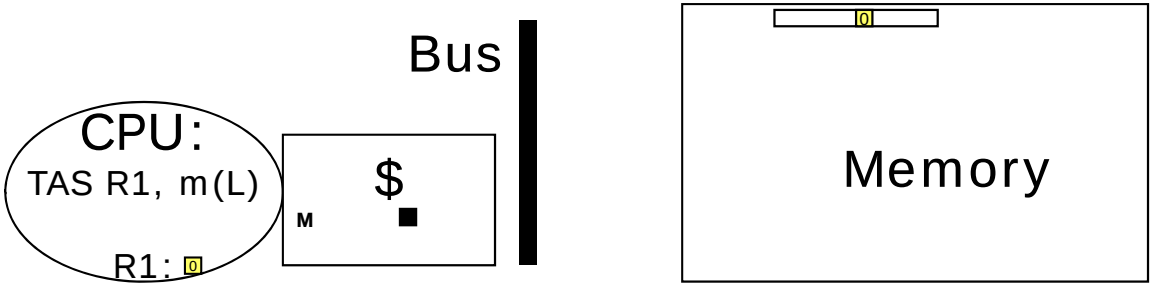
**How is TAS treated by the coherence protocol?**
- ❑ Like a CPU read operation
- ❑ Like a CPU write opreration
- ❑ By performing the ”SWAP” atomically in DRAM

```
proc lock(lock_variable) {
    while (TAS[lock_variable]==1)  {}      /* pound on the lock until free */
}

proc unlock(lock_variable) {
    lock_variable := 0
}
```

**If two threads are waiting for the lock**
- ❑ They will both spin locally in their cache
- ❑ They will create coherence traffic by invalidating each other
- ❑ They will both block and need to be woken up by the OS

*Assume: The function TAS[addr] returns the current memory value at addr and **atomically** writes the busy pattern “1” to the memory*



UPPSALA UNIVERSITET

# Optimistic Test&Set Lock ”spinlock”

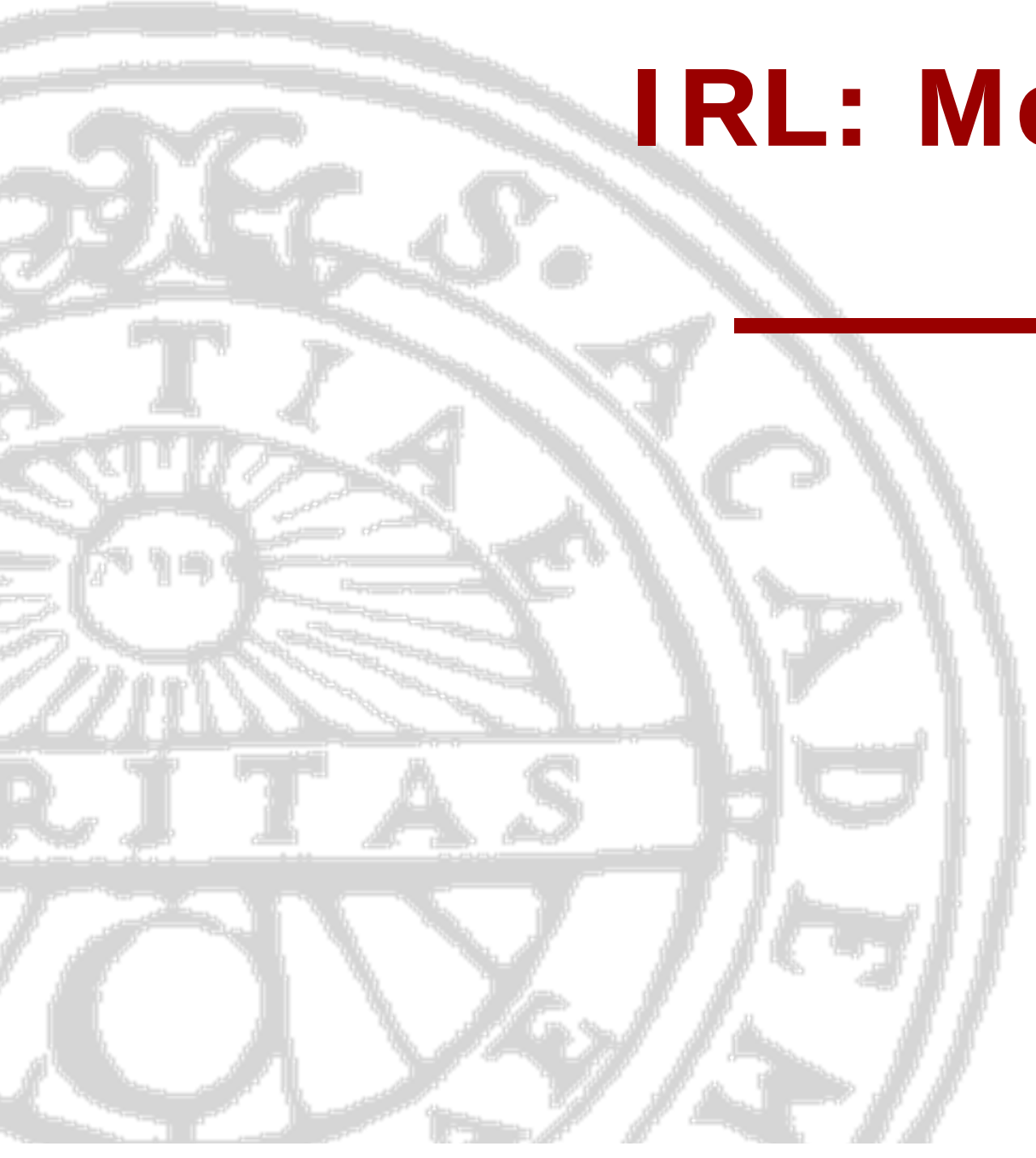
```
proc lock(lock_variable) {
    while true {
      if (TAS[lock_variable] == 0)  break;     /* pound on the lock once, done if TAS==0 *
      while(lock_variable != 0) {}       /* spin locally in your cache until ”0” observed*/
  }
}

proc unlock(lock_variable) {
    lock_variable := 0
}
```

**If two threads are waiting for the lock**

- ❑ They will mostly spin locally in their cache
- ❑ They will create coherence traffic all the time by invalidating each other
- ❑ They will both block and need to be woken up by the OS

**Much less coherence traffic!!**
**-- still lots of traffic at lock handover!**
**More on this during Scalable Synchronization**



# IRL: More Problem solving

UPPSALA UNIVERSITET

# Extra Example not used in IRL Class:

All the three RISC CPUs in a **MOSI** shared-memory (sequentially consistent) multiprocessor executes the following code almost at the same time:

```
while(A != my_id){};    /* this is a primitive kind of lock */
B = B + A;
A = A + 1;              /* this is a primitive kind of unlock */
while (A != 4) {};      /* this is a primitive kind of barrier sync */
<after a long time>
<some other execution replaces A and B from the caches, if still
present>
```

**Initially, CPU1 has its local variable my_id=1, CPU has my_id=2 and CPU3 has my_id=3 and the globally shared variables A is equal to 1 and B is equal to 0**.
Assume that CPU3, 2 and then 1 first make one memory reference (i.e, a load or a store) each , after which they repeats that memory access interleaving.

The following four bus transaction types can be seen on the snooping bus connecting the CPUs:

- **RTS:** ReadtoShare (reading the data with the intention to read it)
- **RTW,** ReadToWrite (reading the data with the intention to modify it)
- **WB:** Writing data back to memory
- **INV:** Invalidating other caches copies

Show every state change and/or value change of A and B in each CPU's cache according to one possible interleaving of the memory accesses. After the parallel execution is done for all of the CPUs, the cache lines still in the caches will be replaced. These actions should also be shown. For each line, also state what bus transaction occurs on the bus (if any) as well as which device is providing the corresponding data (if any).

UPPSALA UNIVERSITET

| Mem Instr | Bus | 1:A | 1:B | 2:A | 2:B | 3:A | 3:B | Data |
|---|---|---|---|---|---|---|---|---|
| 3:LD A | RTS(A) | | | | | S/1 | | MEM |
| 2:LD A | RTS(A) | | | S/1 | | | | MEM |
| 1:LD A | RTS(A) | S/1 | | | | | | MEM |
| 3,2:LD A | -- | | | | | | | -- |
| 1:LD B | RTS(B) | | S/0 | | | | | MEM |
| 3,2:LD A | -- | | | | | | | -- |
| 1:ST B | INV(B) | | M/1 | | | | | -- |
| 3,2:LD A | -- | | | | | | | -- |
| 1:ST A | INV(A) | M/2 | | I | | I | | -- |
| 3:LD A | RTS(A) | O/2 | | | | S/2 | | $1 |
| 2:LD A | RTS(A) | | | S/2 | | | | $1 |
| 1,3:LD A | -- | | | | | | | -- |
| 2:LD B | RTS(B) | | O/1 | | S/1 | | | $1 |
| 1,3:LD A | -- | | | | | | | -- |
| 2:ST B | INV(B) | | I | | M/2 | | | -- |
| 1,3:LD A | -- | | | | | | | -- |
| 2:ST A | INV(A) | I | | M/3 | | I | | -- |

AVDARK 2013

UPPSALA UNIVERSITET

| Mem Instr | Bus | 1:A | 1:B | 2:A | 2:B | 3:A | 3:B | Data |
|---|---|---|---|---|---|---|---|---|
| TRANSPORT | | | | M/3 | M/2 | | | |
| 1:LD A | RTS(A) | S/3 | | O/3 | | | | $2 |
| 3:LD A | RTS(A) | | | | | S/3 | | $2 |
| 2,1:LD A | -- | | | | | | | -- |
| 3:LD B | RTS(B) | | | | O/2 | | S/2 | $2 |
| 2,1:LD A | -- | | | | | | | -- |
| 3:ST B | INV(B) | | | | I | | M/5 | -- |
| 2,1:LD A | -- | | | | | | | -- |
| 3:ST A | INV(A) | I | | I | | M/4 | | -- |
| 2:LD A | RTS(A) | | | S/4 | | O/4 | | $3 |
| 1:LD A | RTS(A) | S/4 | | | | | | $3 |
| 3:LD A | -- | | | | | | | -- |
| LONG TIME | | | | | | | | |
| 1,2:REPL A | -- | I | | I | | | | -- |
| 3:REPL A | WB(A) | | | | | I | | $3 |
| 3:REPL B | WB(B) | | | | | | I | $3 |
| | | | | | | | | |

AVDARK 2013

UPPSALA UNIVERSITET

# Using MSI

All the three RISC CPUs in a **MSI** shared-memory (sequentially consistent) multiprocessor executes the following code almost at the same time:

```
while(A != my_id){};    /* this is a primitive kind of lock */
B = B + A;
A = A + 1;              /* this is a primitive kind of unlock */
while (A != 4) {};      /* this is a primitive kind of barrier sync */
<after a long time>
<some other execution replaces A and B from the caches, if still
present>
```

**Initially, CPU1 has its local variable my_id=1, CPU has my_id=2 and CPU3 has my_id=3 and the globally shared variables A is equal to 1 and B is equal to 0**.
Assume that CPU3, 2 and then 1 first make one memory reference (i.e, a load or a store) each , after which they repeats that memory access interleaving.

The following four bus transaction types can be seen on the snooping bus connecting the CPUs:

- **RTS:** ReadtoShare (reading the data with the intention to read it)
- **RTW,** ReadToWrite (reading the data with the intention to modify it)
- **WB:** Writing data back to memory
- **INV:** Invalidating other caches copies

Show every state change and/or value change of A and B in each CPU's cache according to one possible interleaving of the memory accesses. After the parallel execution is done for all of the CPUs, the cache lines still in the caches will be replaced. These actions should also be shown. For each line, also state what bus transaction occurs on the bus (if any) as well as which device is providing the corresponding data (if any).

UPPSALA UNIVERSITET

| Mem Instr | Bus | 1:A | 1:B | 2:A | 2:B | 3:A | 3:B | Data |
|---|---|---|---|---|---|---|---|---|
| 3:LD A | RTS(A) | | | | | S/1 | | MEM |
| 2:LD A | RTS(A) | | | S/1 | | | | MEM |
| 1:LD A | RTS(A) | S/1 | | | | | | MEM |
| 3,2:LD A | -- | | | | | | | -- |
| 1:LD B | RTS(B) | | S/0 | | | | | MEM |
| 3,2:LD A | -- | | | | | | | -- |
| 1:ST B | INV(B) | | M/1 | | | | | -- |
| 3,2:LD A | -- | | | | | | | -- |
| 1:ST A | INV(A) | M/2 | | I | | I | | -- |
| 3:LD A | RTS(A) | S/2 | | | | S/2 | | $1 |
| 2:LD A | RTS(A) | | | S/2 | | | | MEM |
| 1,3:LD A | -- | | | | | | | -- |
| 2:LD B | RTS(B) | | S/1 | | S/1 | | | $1 |
| 1,3:LD A | -- | | | | | | | -- |
| 2:ST B | INV(B) | | I | | M/2 | | | -- |
| 1,3:LD A | -- | | | | | | | -- |
| 2:ST A | INV(A) | I | | M/3 | | I | | -- |

AVDARK 2013

UPPSALA UNIVERSITET

| Mem Instr | Bus | 1:A | 1:B | 2:A | 2:B | 3:A | 3:B | Data |
|---|---|---|---|---|---|---|---|---|
| TRANSPORT | | | | M/3 | M/2 | | | |
| 1:LD A | RTS(A) | S/3 | | S/3 | | | | $2 |
| 3:LD A | RTS(A) | | | | | S/3 | | MEM |
| 2,1:LD A | -- | | | | | | | -- |
| 3:LD B | RTS(B) | | | | S/2 | | S/2 | $2 |
| 2,1:LD A | -- | | | | | | | -- |
| 3:ST B | INV(B) | | | | I | | M/5 | -- |
| 2,1:LD A | -- | | | | | | | -- |
| 3:ST A | INV(A) | I | | I | | M/4 | | -- |
| 2:LD A | RTS(A) | | | S/4 | | S/4 | | $3 |
| 1:LD A | RTS(A) | S/4 | | | | | | MEM |
| 3:LD A | -- | | | | | | | -- |
| LONG TIME | | | | | | | | |
| 1,2:REPL A | -- | I | | I | | | | -- |
| 3:REPL A | -- | | | | | I | | -- |
| 3:REPL B | WB(B) | | | | | | I | $3 |
| | | | | | | | | |

AVDARK 2013

UPPSALA UNIVERSITET

# Using MOESI

All the three RISC CPUs in a **MOESI** shared-memory (sequentially consistent) multiprocessor executes the following code almost at the same time:

```
while(A != my_id){};    /* this is a primitive kind of lock */
B = B + A;
A = A + 1;              /* this is a primitive kind of unlock */
while (A != 4) {};      /* this is a primitive kind of barrier sync */
<after a long time>
<some other execution replaces A and B from the caches, if still
present>
```

**Initially, CPU1 has its local variable my_id=1, CPU has my_id=2 and CPU3 has my_id=3 and the globally shared variables A is equal to 1 and B is equal to 0**.
Assume that CPU3, 2 and then 1 first make one memory reference (i.e, a load or a store) each , after which they repeats that memory access interleaving.

The following four bus transaction types can be seen on the snooping bus connecting the CPUs:

- **RTS:** ReadtoShare (reading the data with the intention to read it)
- **RTW,** ReadToWrite (reading the data with the intention to modify it)
- **WB:** Writing data back to memory
- **INV:** Invalidating other caches copies

Show every state change and/or value change of A and B in each CPU's cache according to one possible interleaving of the memory accesses. After the parallel execution is done for all of the CPUs, the cache lines still in the caches will be replaced. These actions should also be shown. For each line, also state what bus transaction occurs on the bus (if any) as well as which device is providing the corresponding data (if any).

UPPSALA UNIVERSITET

| Mem Instr | Bus | 1:A | 1:B | 2:A | 2:B | 3:A | 3:B | Data |
|---|---|---|---|---|---|---|---|---|
| 3:LD A | RTS(A) | | | | | **E/1** | | MEM |
| 2:LD A | RTS(A) | | | S/1 | | **S/1** | | MEM |
| 1:LD A | RTS(A) | S/1 | | | | | | MEM |
| 3,2:LD A | -- | | | | | | | -- |
| 1:LD B | RTS(B) | | **E/0** | | | | | MEM |
| 3,2:LD A | -- | | | | | | | -- |
| 1:ST B | ~~INV(B)~~ -- | | M/1 | | | | | -- |
| 3,2:LD A | -- | | | | | | | -- |
| 1:ST A | INV(A) | M/2 | | I | | I | | -- |
| 3:LD A | RTS(A) | O/2 | | | | S/2 | | $1 |
| 2:LD A | RTS(A) | | | S/2 | | | | $1 |
| 1,3:LD A | -- | | | | | | | -- |
| 2:LD B | RTS(B) | | O/1 | | S/1 | | | $1 |
| 1,3:LD A | -- | | | | | | | -- |
| 2:ST B | INV(B) | | I | | M/2 | | | -- |
| 1,3:LD A | -- | | | | | | | -- |
| ... | | | | | | | | |

AVDARK 2013

UPPSALA UNIVERSITET

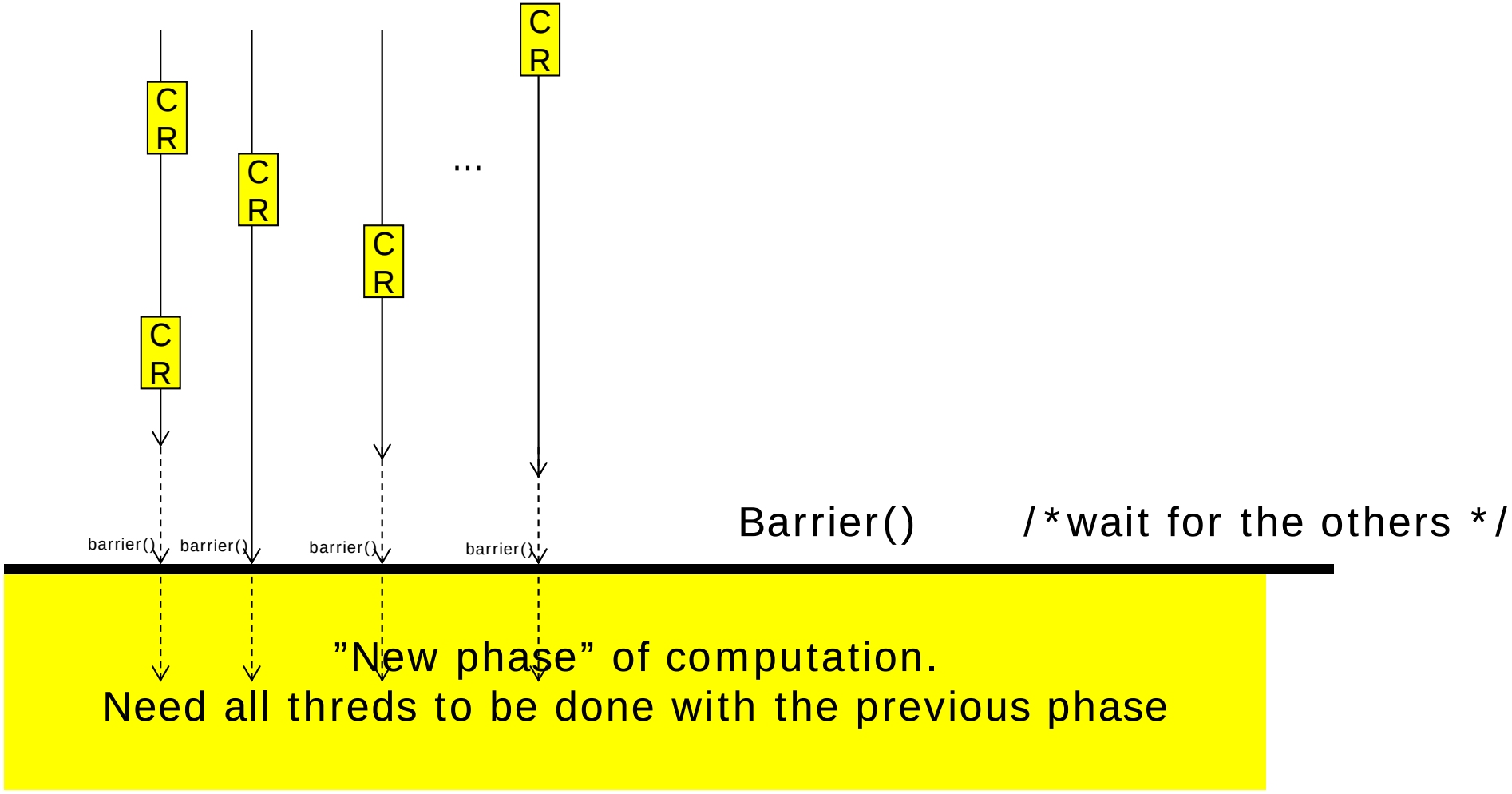
# Sync example

All the three RISC CPUs in a **MOSI** shared-memory (sequentially consistent) multiprocessor executes the following code almost at the same time:

```
barrier(B, 3);
```

Assume that CPU3, 2 and then 1 first make one memory reference (i.e, a load, store or atomic) each , after which they repeats that memory access interleaving.  Fill out the state transition sheet for the entire execution including cache evictions at the and.


# Barrier Synchronization



UPPSALA UNIVERSITET

# Sync example

All the three RISC CPUs in a **MOSI** shared-memory (sequentially consistent) multiprocessor executes the following code almost at the same time:

```
barrier(B, 3);
```

Assume that CPU3, 2 and then 1 first make one memory reference (i.e, a load, store or atomic) each , after which they repeats that memory access interleaving. Fill out the state transition sheet for the entire execution including cache evictions at the and. Initially B.ctr = 0 and B.L = 0. Both reside in memory only.

```
DEF barrier (bar_name, p) {    /* we know that this barrier is way too simple,
    right? */
  lock(bar_name.L);
  bar_name.ctr++ ;                    /* globally increment the barrier count
    */
  unlock(bar_name.L);
  while (bar_name.ctr < p) {};  /* wait for the last thread */
}

DEF lock(lock_variable) {                        /* This is an optimistic
    spinlock */
  while true {
       if (TAS[lock_variable] == 0)  break; /* Pound once. Done if TAS returns 0
    */
       while(lock_variable != 0) {}              /* Spin locally */
    }
}

DEF unlock(lock_variable) {
    lock_variable = 0;
```

UPPSALA UNIVERSITET

| Mem Instr | Bus | 1:ctr | 1:L | 2:ctr | 2:L | 3:ctr | 3:L | Data |
|---|---|---|---|---|---|---|---|---|
| 3:TAS L in cs | RTW(L) | | | | | | M/FF | Mem |
| 2:TAS L | RTW(L) | | | | M/FF | | I | 3 |
| 1:TAS L | RTW(L) | | M/FF | | I | | | 2 |
| 3:LD CTR | RTS(ctr) | | | | | S/0 | | Mem |
| 2:LD L spin | RTS(L) | | O/FF | | S/FF | | | 1 |
| 1:LD L spin | -- | | | | | | | -- |
| 3:ST CTR | INV(ctr) | | | | | M/1 | | -- |
| 2:LD L spin | -- | | | | | | | -- |
| 1:LD L spin | -- | | | | | | | -- |
| 3:ST L unlock | RTW(L) | | I | | I | | M/0 | 1 |
| 2:LD L spin | RTS(L) | | | | S/0 | | O/0 | 3 |
| 1:LD L spin | RTS(L) | | S/0 | | | | | 3 |
| 3:LD CTR | -- | | | | | | | -- |
| 2:TAS L in cs | INV(L) | | I | | M/FF | | I | -- |
| 1:TAS L | RTW(L) | | M/FF | | I | | | 2 |
| 3:LD CTR | -- | | | | | | | -- |
| 2: LD CTR | RTS(ctr) | | | S/1 | | O/1 | | 3 |

AVDARK 2013

UPPSALA UNIVERSITET

AVDARK 2013

| Mem Instr | Bus | 1:ctr | 1:L | 2:ctr | 2:L | 3:ctr | 3:L | Data |
|---|---|---|---|---|---|---|---|---|
| Transport | | | M/FF | S/1 | I | O/1 | I | |
| 1:LD L spin | -- | | | | | | | -- |
| 3:LD CTR | -- | | | | | | | -- |
| 2:ST CTR | INV(ctr) | | | M/2 | | I | | -- |
| 1:LD L spin | -- | | | | | | | -- |
| 3:LD CTR | RTS(ctr) | | | O/2 | | S/2 | | 2 |
| 2:ST L unlock | RTW(L) | | I | | M/0 | | | 1 |
| 1:LD L spin | RTS(L) | | S/0 | | O/0 | | | 2 |
| 3+2:LD CTR | -- | | | | | | | -- |
| 1:TAS L in cs | INV(L) | | M/FF | | I | | | -- |
| 3+2:LD CTR | -- | | | | | | | -- |
| 1:LD CTR | RTS(ctr) | S/2 | | | | | | 2 |
| 3+2:LD CTR | -- | | | | | | | -- |
| 1:ST CTR | INV(ctr) | M/3 | | I | | I | | -- |
| 3:LD CTR done | RTS(ctr) | O/3 | | | | S/3 | | 1 |
| 2:LD CTR done | RTS(ctr) | | | S/3 | | | | 1 |
| ... | ... | | | | | | | |

UPPSALA UNIVERSITET

# A Naiive Centralized Barrier

```
BARRIER (bar_name, p) {

 LOCK(bar_name.lock) {
    if (bar_name.ctr == p) bar_name.ctr = 0; /* init count*/
    bar_name.ctr++;                          /* globally increment the barrier count */
 }
 UNLOCK(bar_name.lock)

  while (bar_name.ctr < p) {};               /* wait for the last thread */

 }
```

UPPSALA UNIVERSITET

# A More Complicated Centralized Barrier

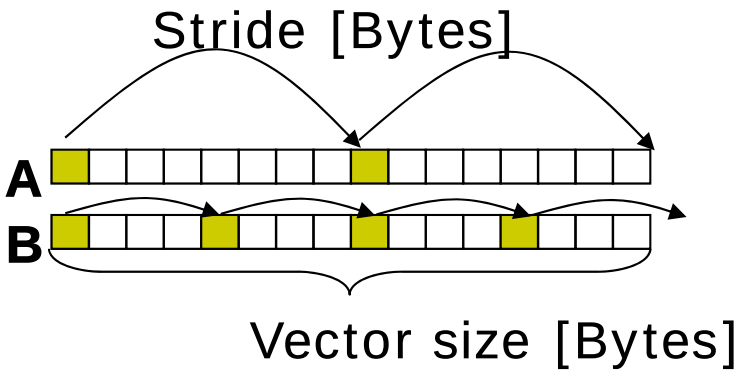
```
BARRIER (bar_name, p) {
int loops;
loops = 0;

local_sense = !(local_sense)  ;          /* toggle private sense variable
                                            each time the barrier is used */
 LOCK(bar_name.lock);
    bar_name.counter++;                  /* globally increment the barrier count */
    if (bar_name.counter == p)  {        /* everybody here yet ? */
        bar_name.flag = local_sense;     /* release waiters*/
        UNLOCK(bar_name.lock)
     }
    else
      {  UNLOCK(bar_name.lock);
        while (bar_name.flag != local_sense) {     /* wait for the last guy */
          if (loops++ > UNREASONABLE) report_warning(pid)}
     }
```

AVDARK 2013

# Microbenchmarks

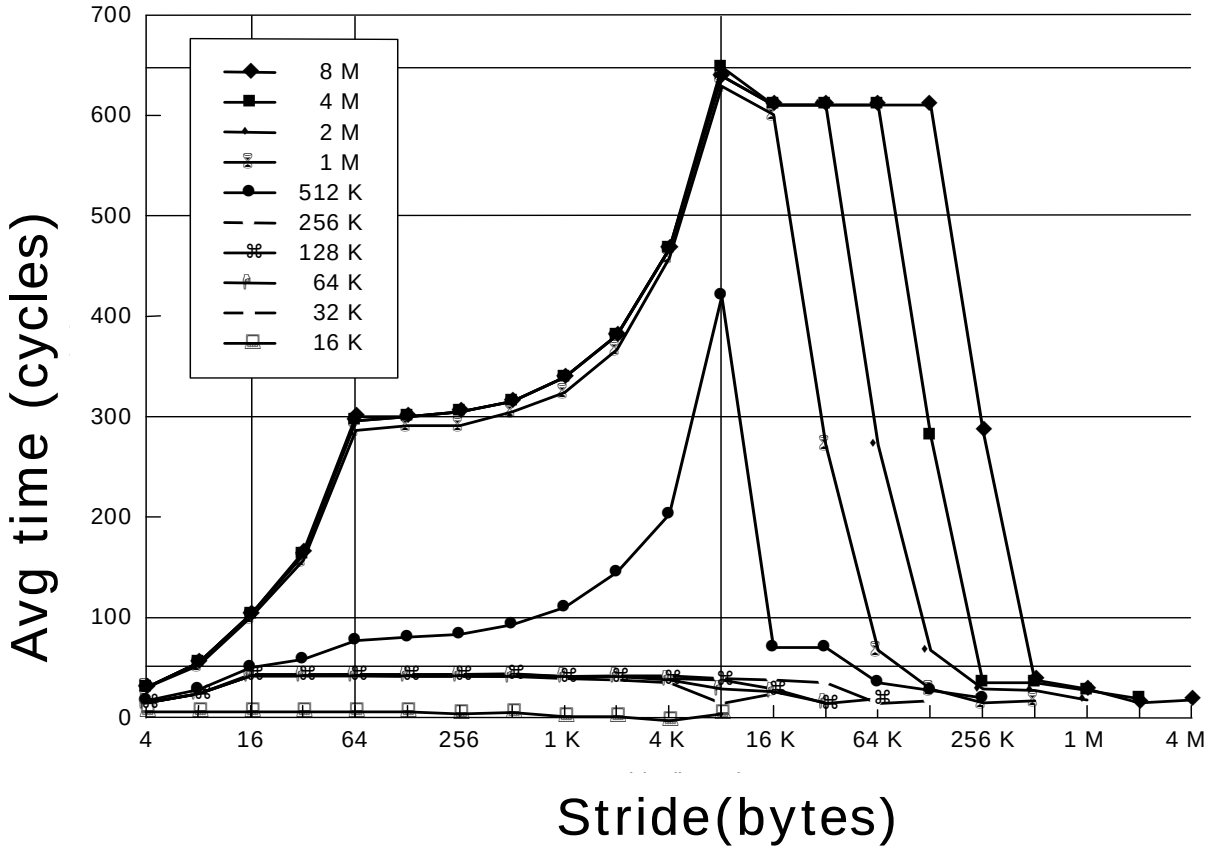
# Stepping through the array

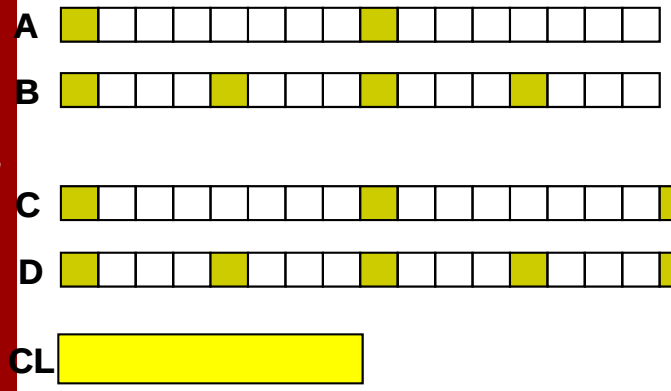



# Micro Benchmark Signature

```
for (times = 0; times < Max; times++) /* many times*/

  for (i=0; i < ArraySize; i = i + Stride)
    dummy = A[i]; /* touch an item in the array */
```






L1: size=4kB CL= 32B

L2: size=256kB CL= 128B

Page: size=16kB Memory: Lat = 220

TLB: reach=1MB #Entries= 64

Microbenchmark

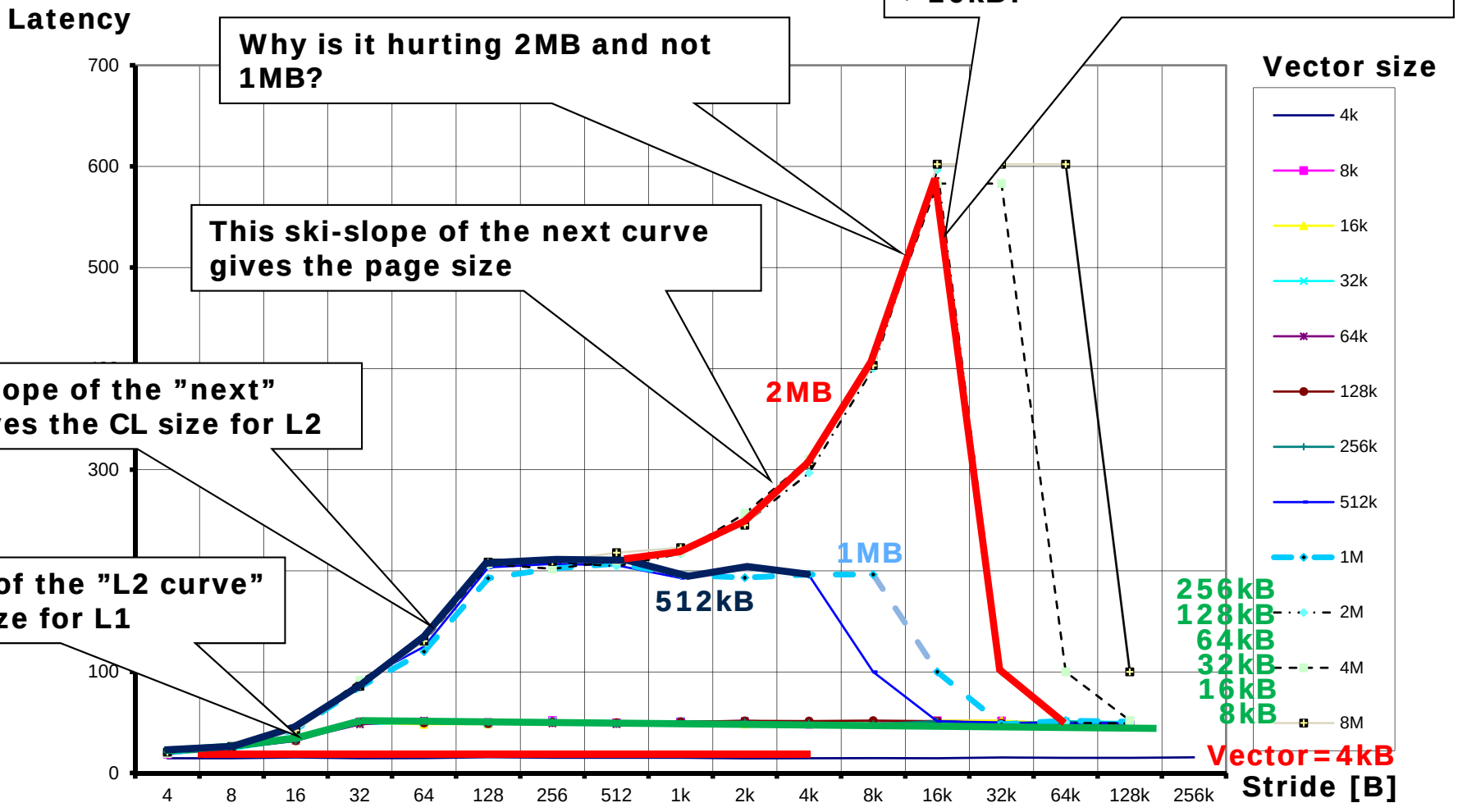
Why is it falling for strides >16kB?

Why is it hurting 2MB and not 1MB?

This ski-slope of the next curve gives the page size

This ski-slope of the ”next” curve” gives the CL size for L2

This ski-slope of the ”L2 curve” gives the CL size for L1




# In-class: Guess the Cache

L1: size= 32kB CL= 32B

L2: size= 1MB CL= 64B

Page: size=4kB

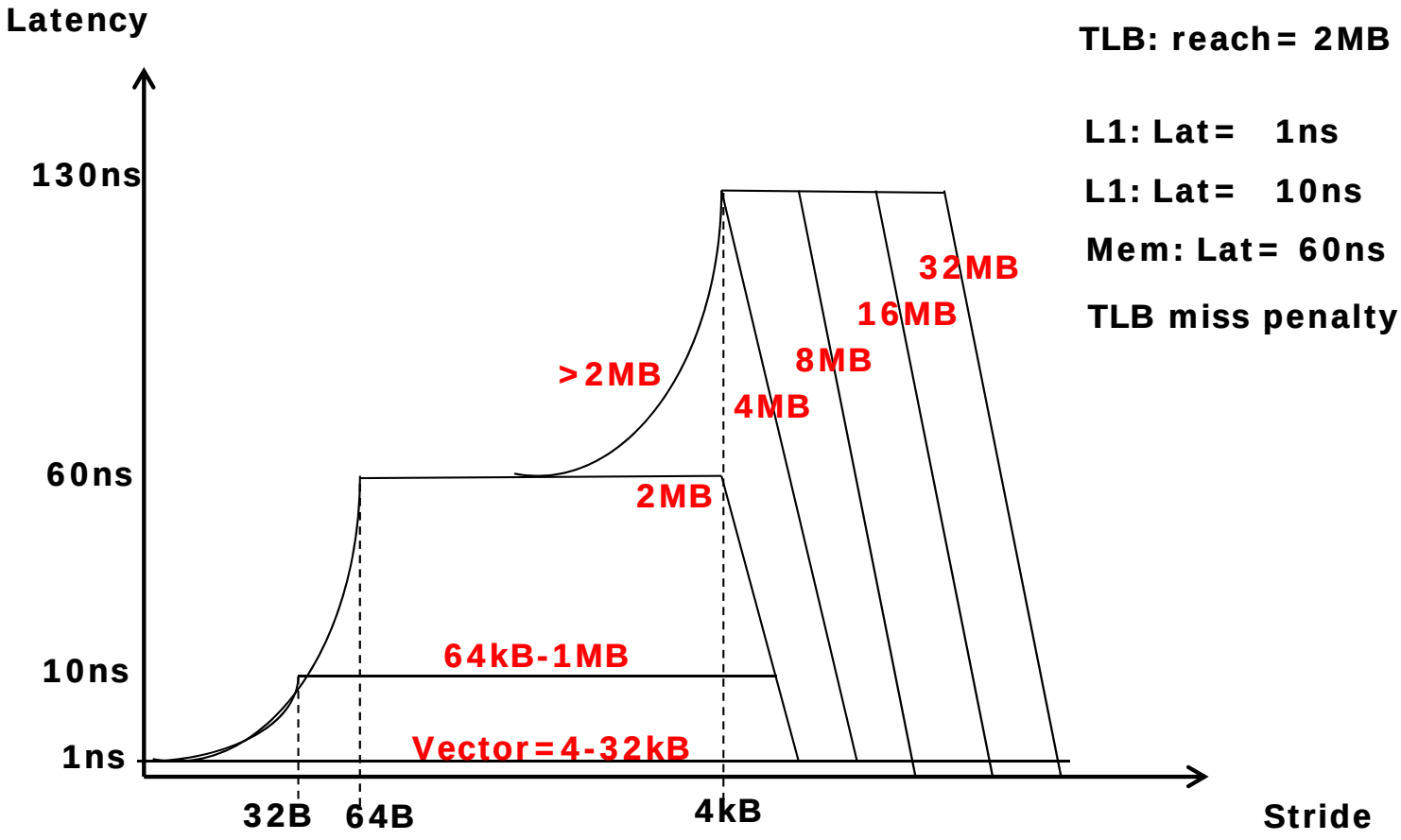
TLB: reach= 2MB #Entries= 512

L1: Lat= 1ns

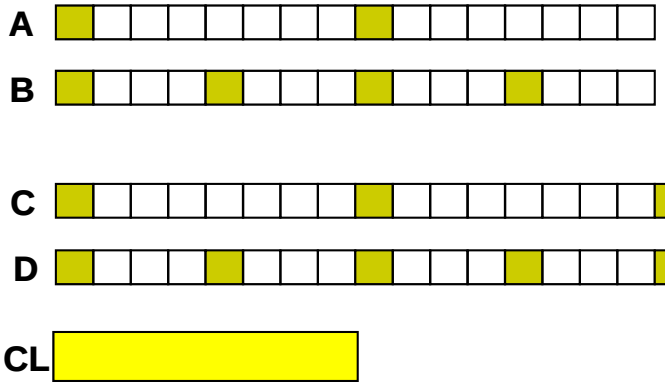
L1: Lat= 10ns

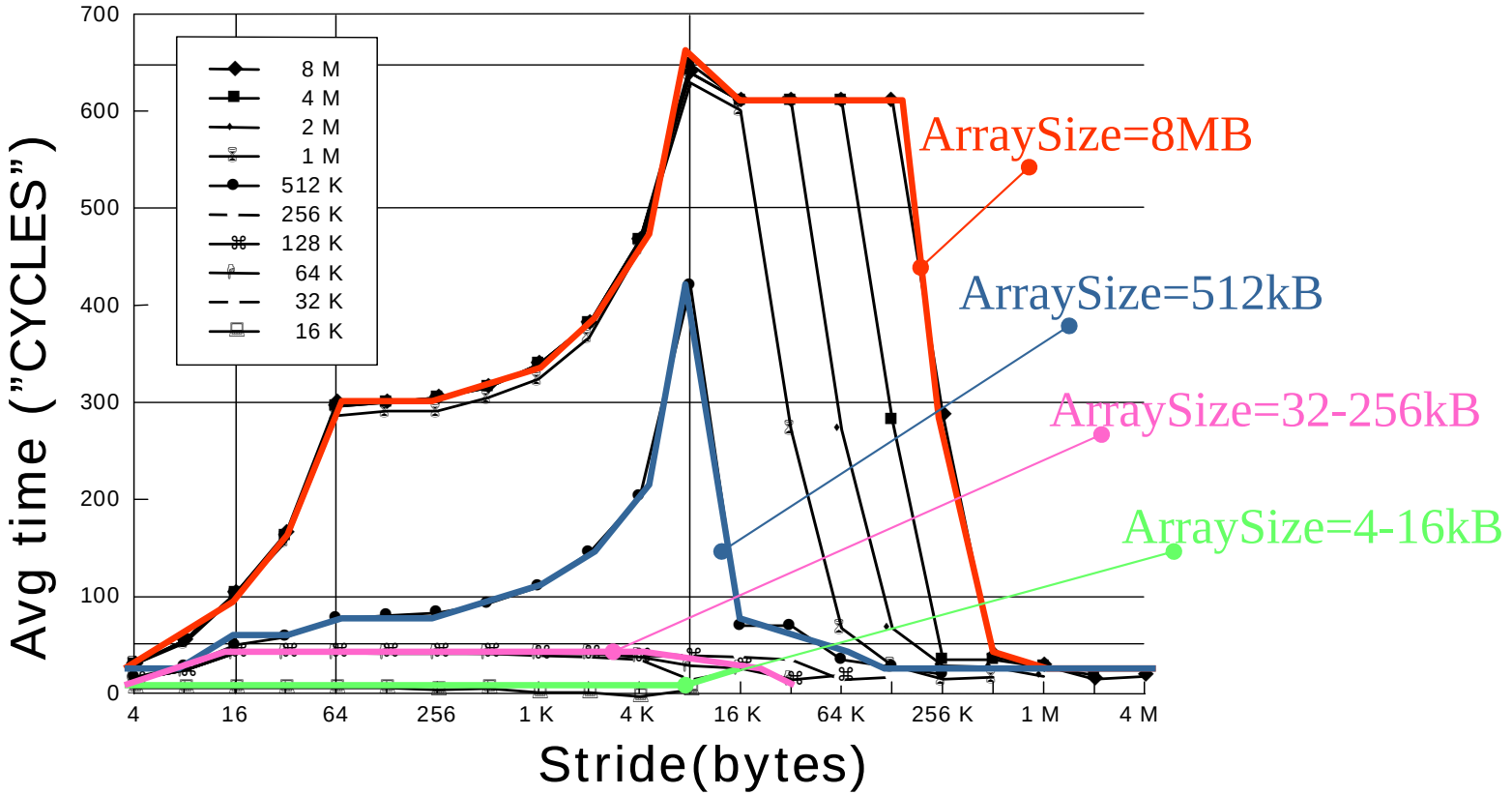
Mem: Lat= 60ns

TLB miss penalty: 70ns



# Micro Benchmark Signature

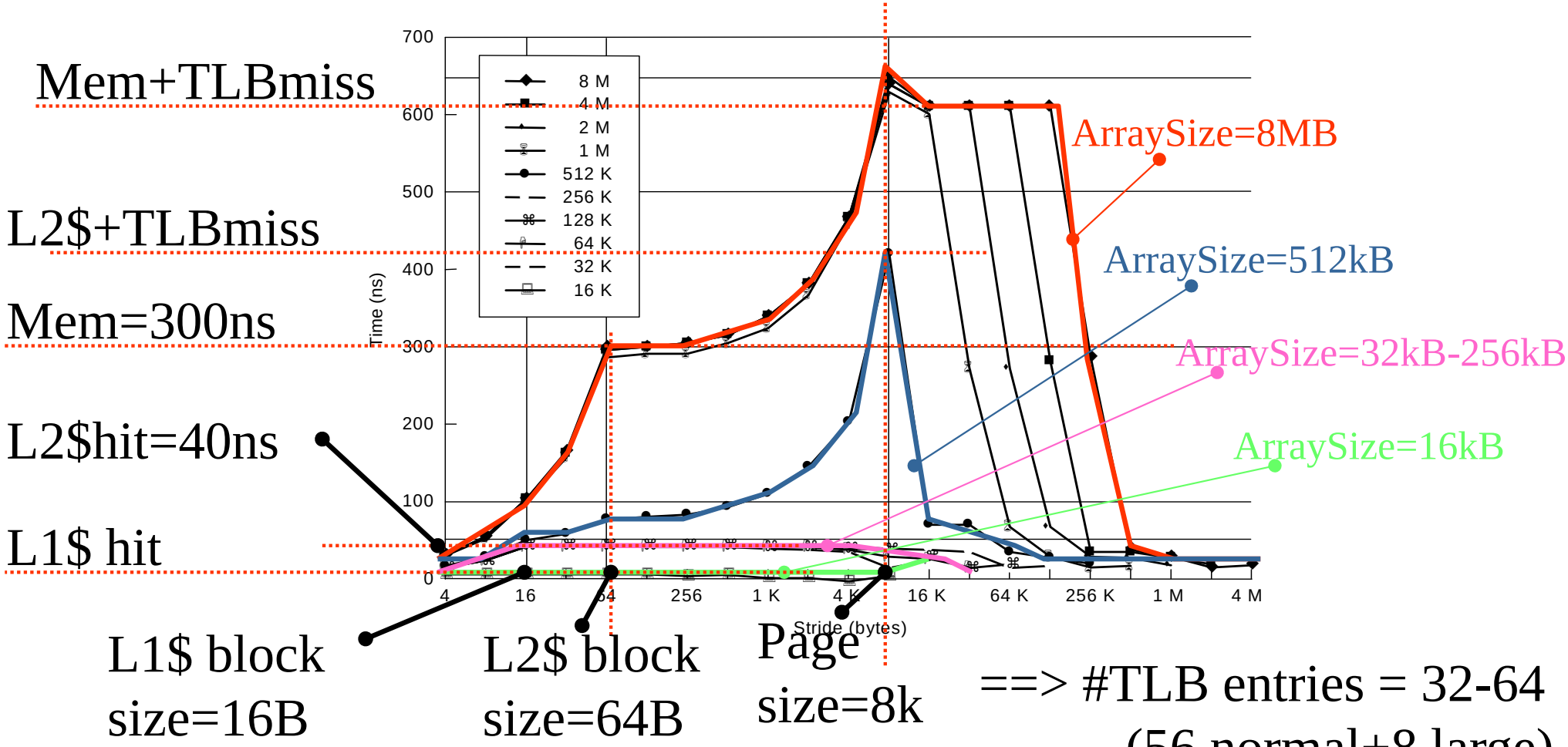


# Micro Benchmark Signature

```
for (times = 0; times < Max; time++) /* many times*/

  for (i=0; i < ArraySize; i = i + Stride)
    dummy = A[i]; /* touch an item in the array */
```

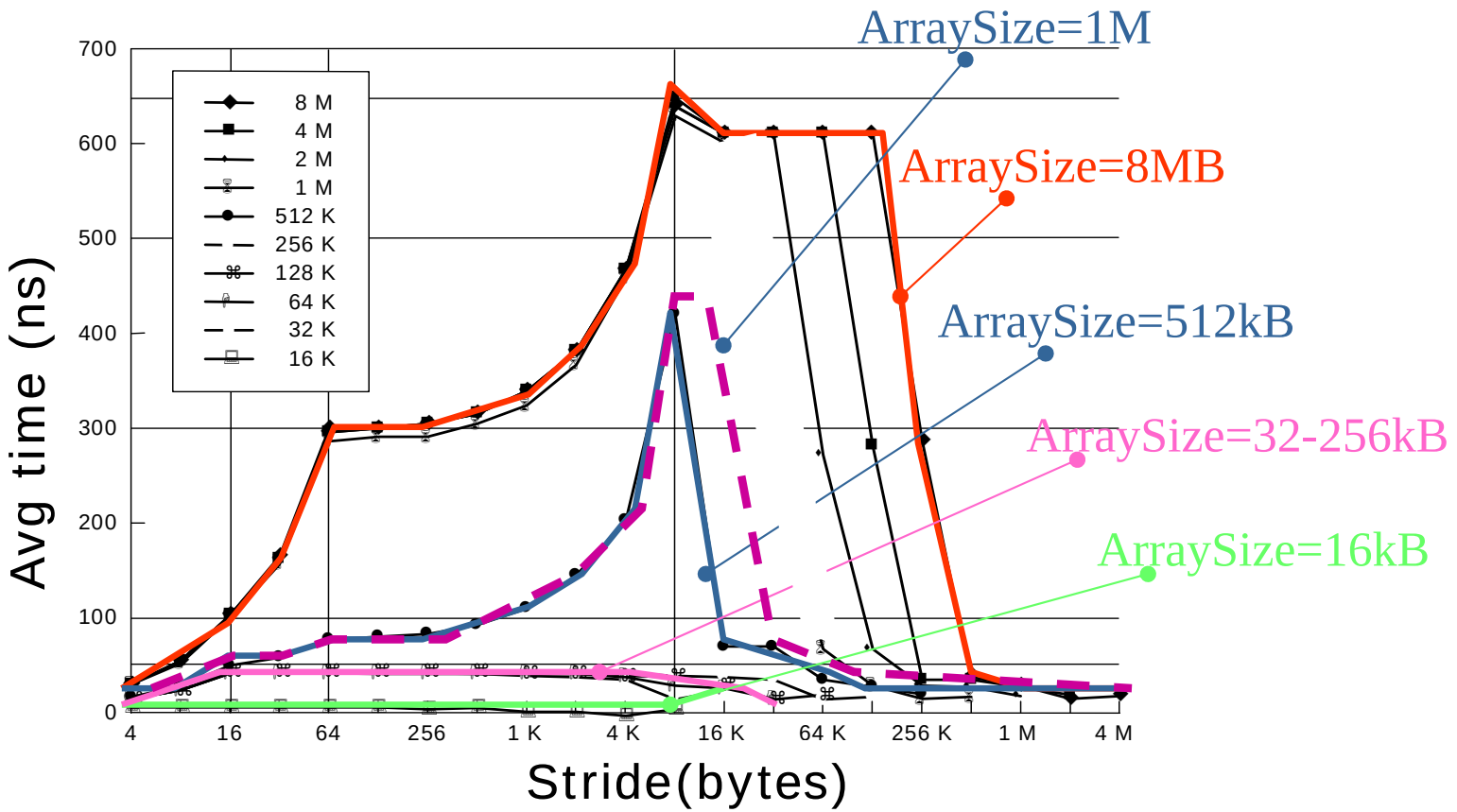


UPPSALA UNIVERSITET

# Twice as large L2 cache ???

```
for (times = 0; times < Max; time++) /* many times*/

  for (i=0; i < ArraySize; i = i + Stride)
    dummy = A[i]; /* touch an item in the array */
```

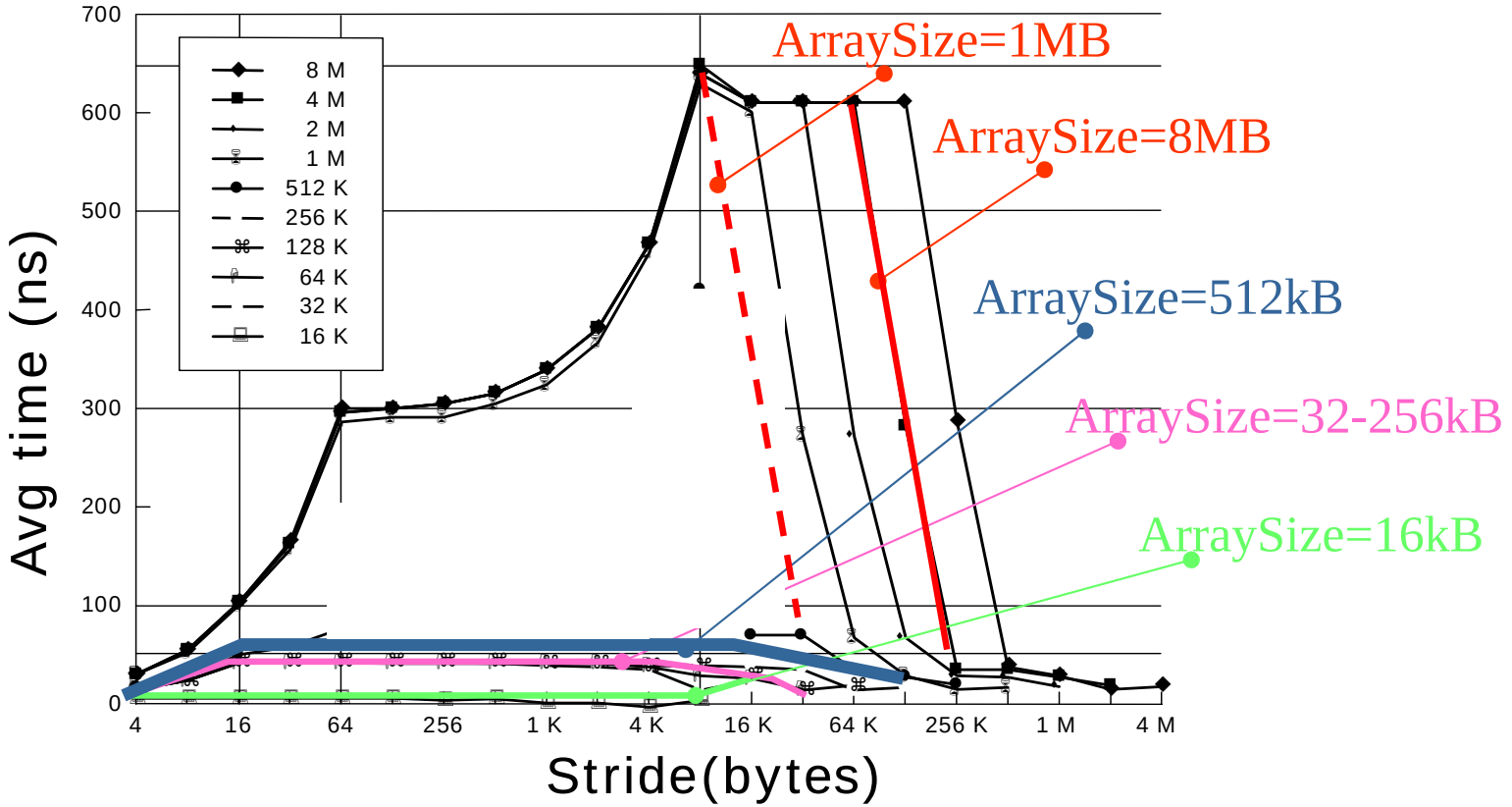



AVDARK 2013


# Twice as large TLB…

```
for (times = 0; times < Max; time++) /* many times*/

  for (i=0; i < ArraySize; i = i + Stride)
    dummy = A[i]; /* touch an item in the array */
```



# Survey Coh & Mem mods

UPPSALA UNIVERSITET

# My take-home message

- The correct answers were sometime wrong
- Sound quality
- Even shorter videos
- More questions during videos
- This part was harder to follow, use more examples
- More ”hover-over hints”
- Question where the screen goes blank is no hit
- Didn’t learn much from the first part of the lab
- Problem-solving in IRL is better the PPT
- More meat during lab lectures
- Please share some questions/stats with us

AVDARK 2013

UPPSALA UNIVERSITET

1 Try to make the **sound** as clear as possible. Questions are good, more questions. Maybe even shorter videos?

2 I think this course is perfect. It is nice to get the information from different channels like videos, labs and classes. The only thing that I can come up with is that it can be **more questions** in the videos. They help you stay focused through the whole video, and also it's **better when the video is around 20** minutes rather than 60. If it is 20 minutes you can **watch it while eating breakfast for example.**

3 The questions from the last batch had some problrms. In one question there was not a **correct answer** and on another one the correct one was not right. The last lecture from the previous batch had really low volume.

4 Frågorna där **skärmen blankades till vitt** så att all hela sammanhanget försvann var ingen hit.

5 **More hands on examples** we can all work through **in class** like today when the power went off. It's bringing in the active learning aspect as opposed to running through the examples on PowerPoint. I for one, get more out of it, I'm sure there are others that feel the same way.

6 I think that it was better in the first videos where it was given some explanation to the answers when **hovering the mouse**.

7 **More questions** in the videos

8 The IRL classes are very helpful when Erik holds them. It really sums up the important parts of the videos and gives a deeper understanding of the concepts in the course. The one **lecture we had with the lab assistants did not do much** for understanding the course or understanding the lab assignment. I would really appreciate a good lab preparation lecture. The labs would be so much more helpful for understanding the course when you **actually understand the assignment**.

9 Can't say that learned anything new during the first lab. It was more bothersome since the **terms during the lectures were different** from the terms used in the skeleton code. So it took awhile to understand what was what. However, the bonus hand-in was a great way to gain a better understanding of the course material.

10 I would like **more questions** during the web lectures. That way we need to use our heads more frequently.

11 The **first part of the lab was mostly** just understanding and **getting into the code** written in the file. I didn't feel like I gained any particular insights into caching or somesuch. Also the hints were not really hints when we mostly only had to change one function (access), they just confused us as to where the changes needed to be. The **video lectures can be a chore** to get through but they explain things well. I prefer regular lectures though.

12 Can't come up with anything for you to improve

13 I don't know. I think it is good!

14 This time it was **not clear.** :(

UPPSALA UNIVERSITET

15 Can't think of anything, except another **Lab soon would be nice**!
16 With the exception of some technical problems (i.e. **bad sound** in one of videos and the "order the consistency models" question didn't work correctly), I have no complaints.
17 Would be nice to have a **cheatsheet** or something for all the terms being thrown around so that if we what something means we can easily go back and check without going through a bunch of videos. All in all very good and understandable! The cache lab really made me get it!
18 Maybe I've misunderstood some recaps to the slides to be answered question, or people aren't asking anything during video lectures. If not any of these, I would like to hear what people are asking from the videos during the IRL classes. **A few words about answering statistics** could also be in place and spending a little more time where a lot of people got them wrong. Also: please hint in the videos that a quiz is coming. I've watched a few videos directly through Youtube and swapped to scalable-learning when a question comes (in one of those red boxes). However, some times a question is not hinted for - it just pops up. This may make me miss some quiz.
19 Compared to last time, the length of the video was perfect.
20 **More focused videos**
21 **This lecture was a bit confusing**, consider giving more examples to concepts rather than glossing over them OR explain that you are glossing over them before you begin to explain the concept itself. There were several times where I would rewind to understand something that I later found out was not as necessary.
22 **Shorter videos** were definitely better, it is however difficult to judge how much the labs help as we have only done one lab.
23 The **videos** are a good idea, but sometimes they are **too long**, I think that if we could read some of the things said in the video, some videos wouldn't be that necessary. I also think that we should try **more excercises in class** (or maybe more hours of class per week)


On a scale 1(low) to 5 (high), how helpful are the videos for understanding this course.
Question 1
Videos
Number of Students
40
20
0
0
0
9
19
24
1
2
3
4
5
Highcharts.com
On a scale 1(low) to 5 (high), how helpful are the lab(s) for understanding this course.
Question 2
Labs
Number of Students
40
20
0
0
2
11
18
21
1
2
3
4
5
Highcharts.com
On a scale 1(low) to 5 (high), how helpful are the IRL classes for understanding this course.
Question 3
IRL
Number of Students
40
20
0
0
1
9
17
25
1
2
3
4
5
Highcharts.com

Goto 19!