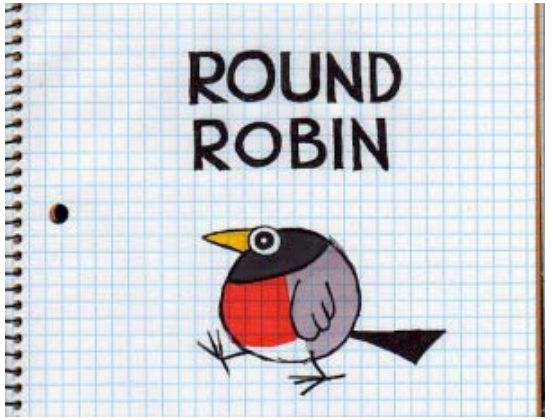
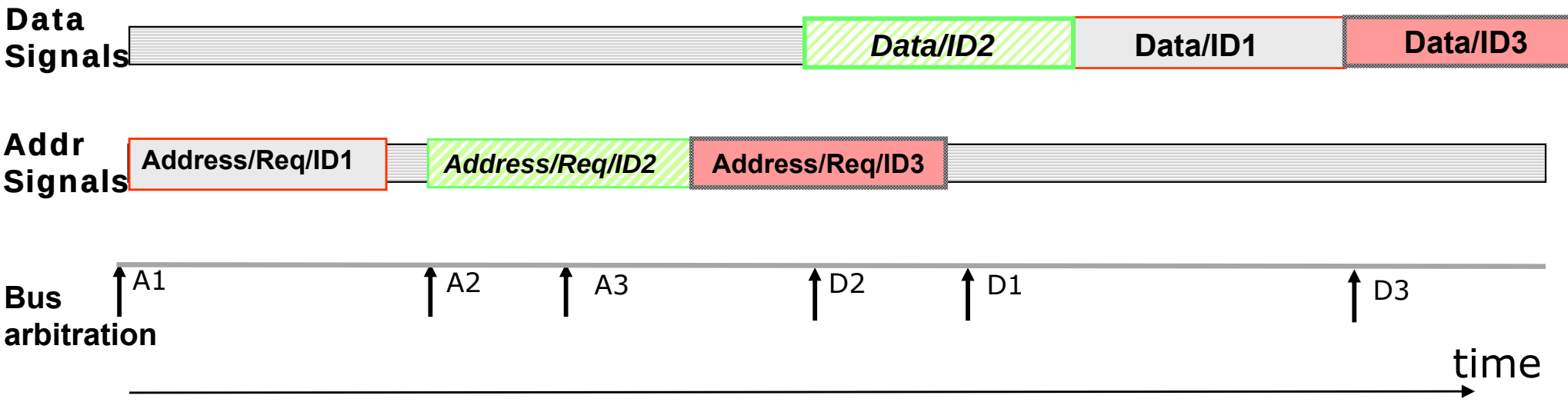
IRL @ Lab Lecture 2

Erik Hagersten
Uppsala University

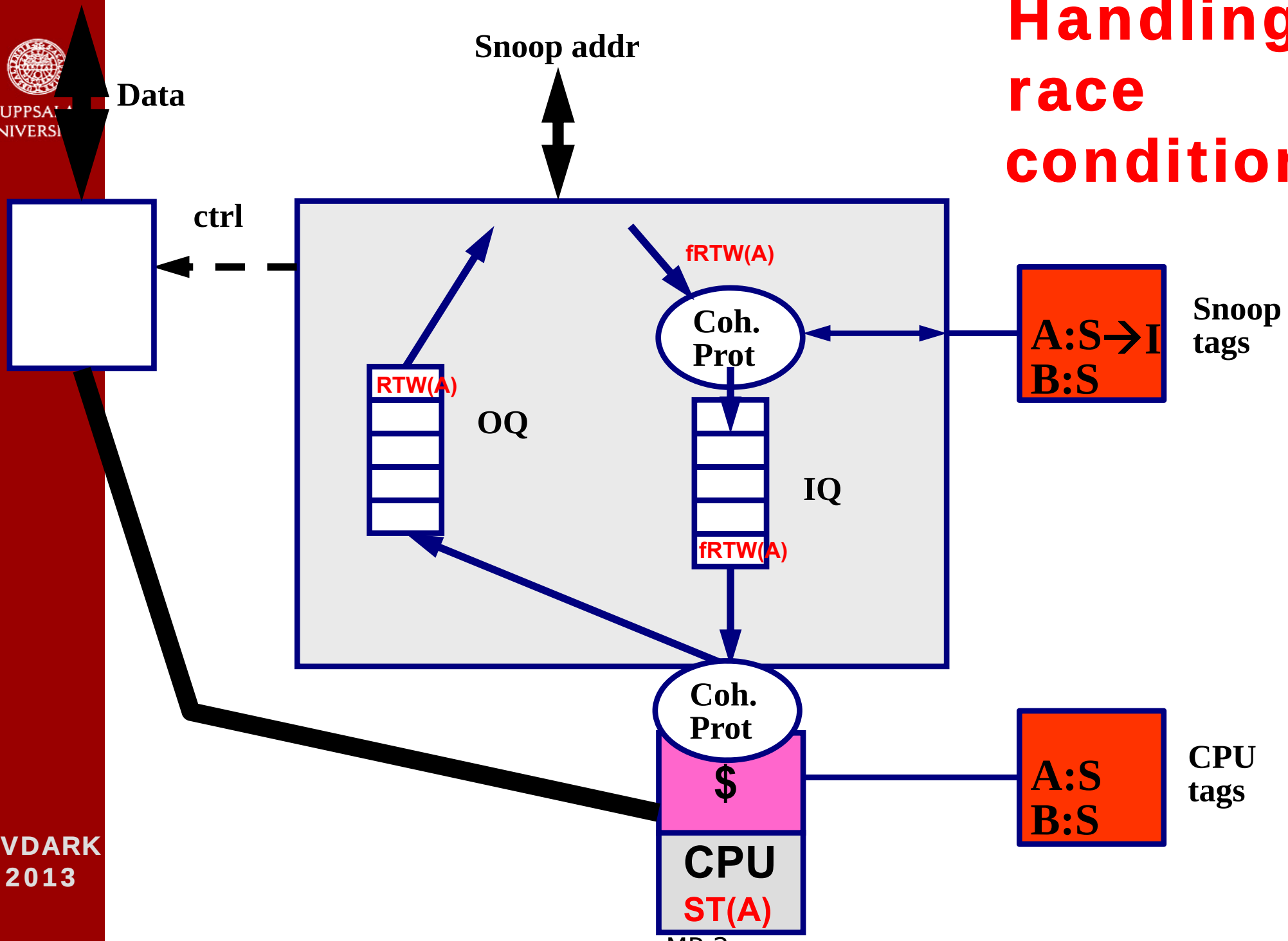
Split-Transaction Bus



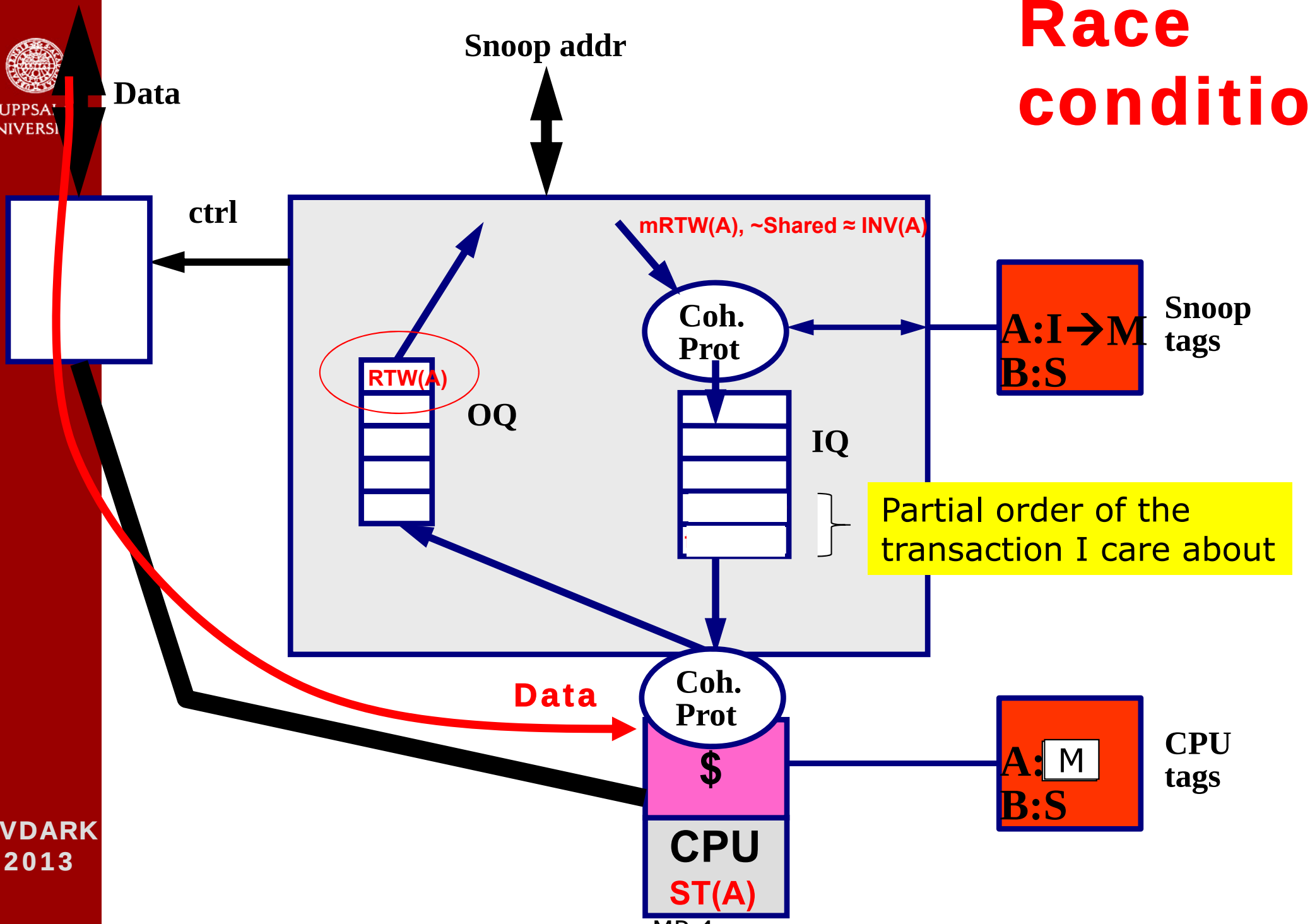
- Split bus transaction
  - Request transaction / Data transaction
  - Separate arbitration for addr/data transaction (**round robin**)
- Out-of-order response
  - Response is matched to request using unique identifiers (ID)



Handling race conditions



Race condition



UPPSALA UNIVERSITET

A cascade of five stores to address A

Initially, A resides in CPU7's cache in state M

On the bus:

- **CPU1: RTW, ID=a**
- **CPU2: RTW, ID=b**
- **…**
- **CPU5: RTW, ID=f**

After the five bus transactions have been sent:

CPU tags		Snoop tags
I	**IQ1 = <mRTW$_{IDa}$, fRTW$_{IDb}$>**	*I*
I	**IQ2 = <mRTW$_{IDb}$, fRTW$_{IDc}$>**	*I*
	…	
I	**IQ5 = <mRTW$_{IDf}$>**	*M*
	…	
M	**IQ7= < fRTW$_{IDa}$>**	*I*

This cache initially had the data

AVDARK 2013

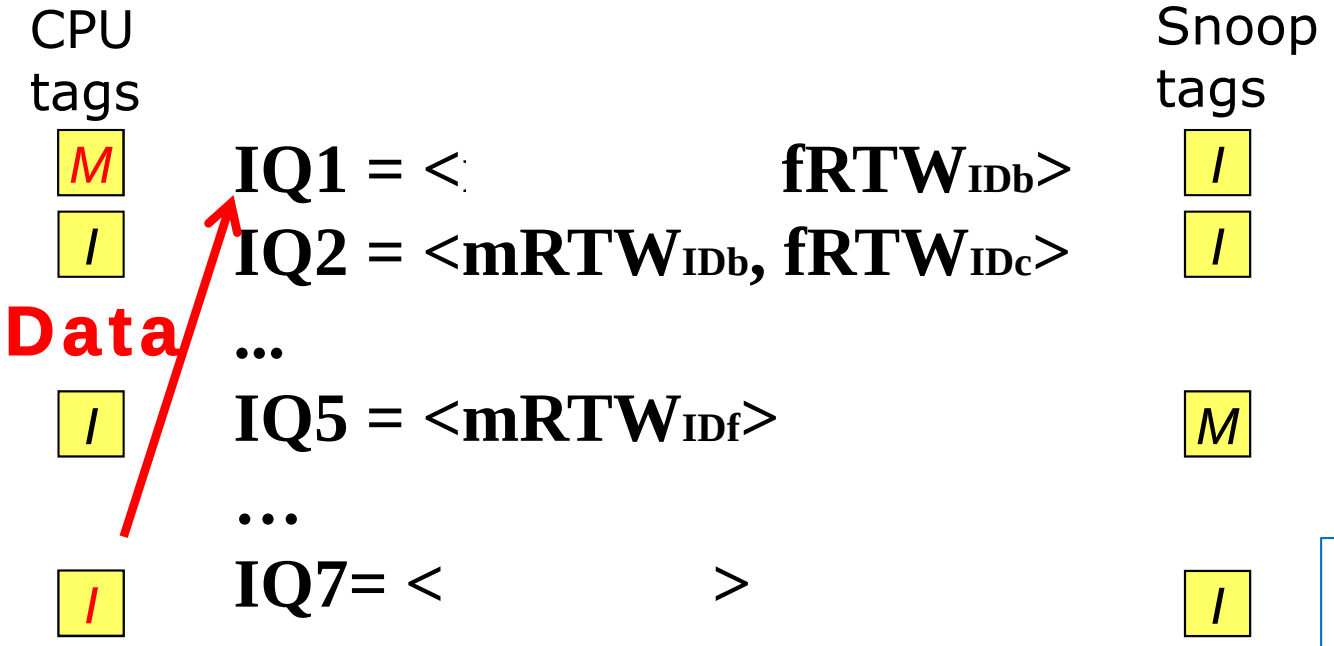
A cascade of five stores to address A

A initially resides in CPU7's cache in state M

On the bus:

- **CPU1: RTW, ID=a**
- **CPU2: RTW, ID=b**
- **…**
- **CPU5: RTW, ID=f**

Servicing the transaction in IQ7



A cascade of five stores to address A

A initially resides in CPU7's cache in state M

On the bus:

- **CPU1: RTW, ID=a**
- **CPU2: RTW, ID=b**
- **…**
- **CPU5: RTW, ID=f**

Servicing fRTW in IQ1

CPU tags	Data		Snoop tags
I	IQ1 = <	>	*I*
M	IQ2 = <	$fRTW_{IDc}$>	*I*
	…		
I	IQ5 = <$mRTW_{IDf}$>		*M*
	…		
I	IQ7= <	>	*I*

This cache initially had the data

What memory model is implemented assuming that everything else is ideal?

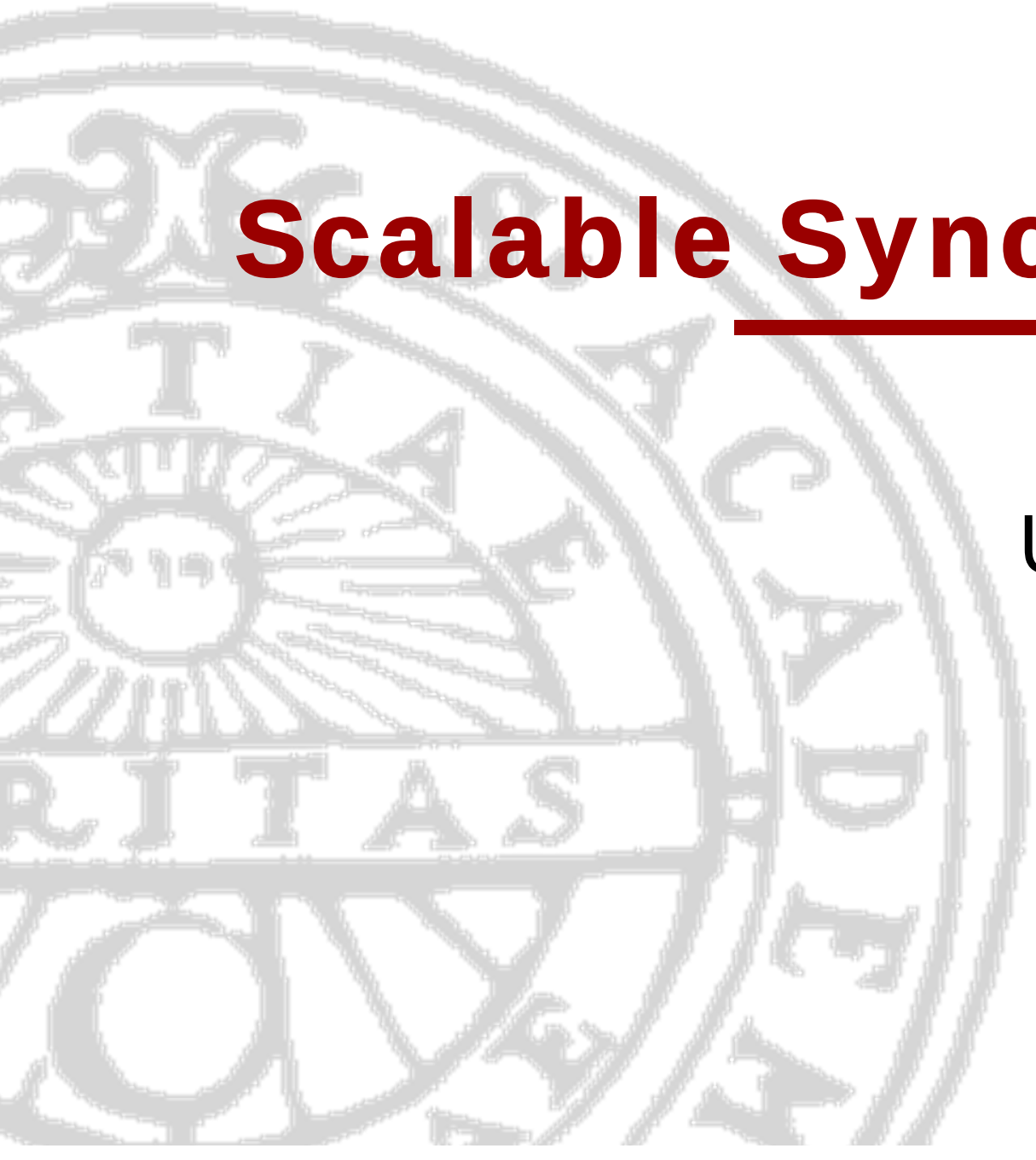
- ❑ Sequential Consistency
- ❑ Total Store Order
- ❑ Weak Consistency

What memory model is implemented assuming the CPUs have store buffers

- ❑ Sequential Consistency
- ❑ Total Store Order
- ❑ Weak Consistency

Summary

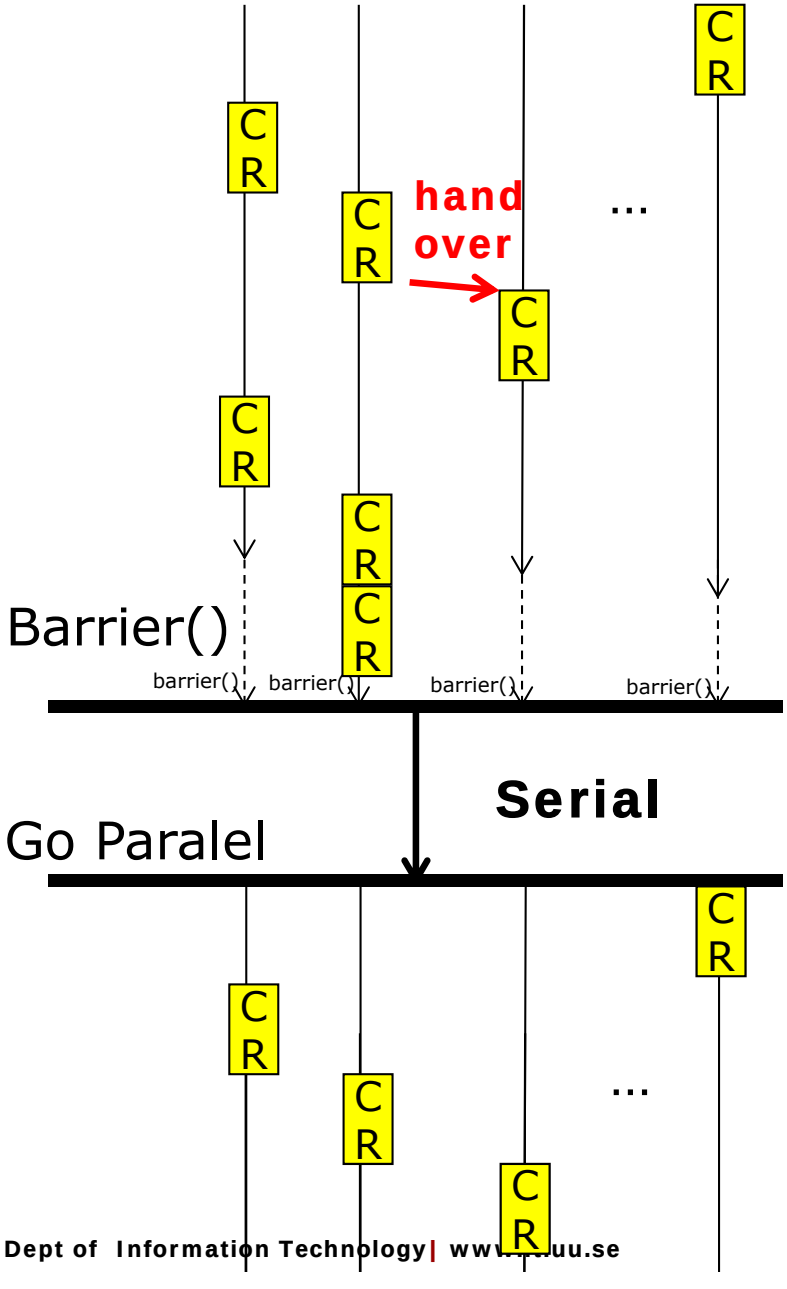
- Transactions are serviced by each cache in the order they occur on the bus (if at all)
- Each IQ contains that partial bus order
- A cache maintains its access rights to a CL until it services a related request from its IQ
- Reading mRTW-Shared (≈mINV) from IQ gives write permission right away
- mRTW and mRTS that requires data can only be serviced once data has arrived



Scalable Synchronization

Erik Hagersten
Uppsala University

Why do we care? Amdahl's Law Example



The upper-bound (UB) for optimizations:

If you optimize for a common case (e.g., 99%) the unoptimized part will limit the UB
→ The upper bound of the optimization (100x)

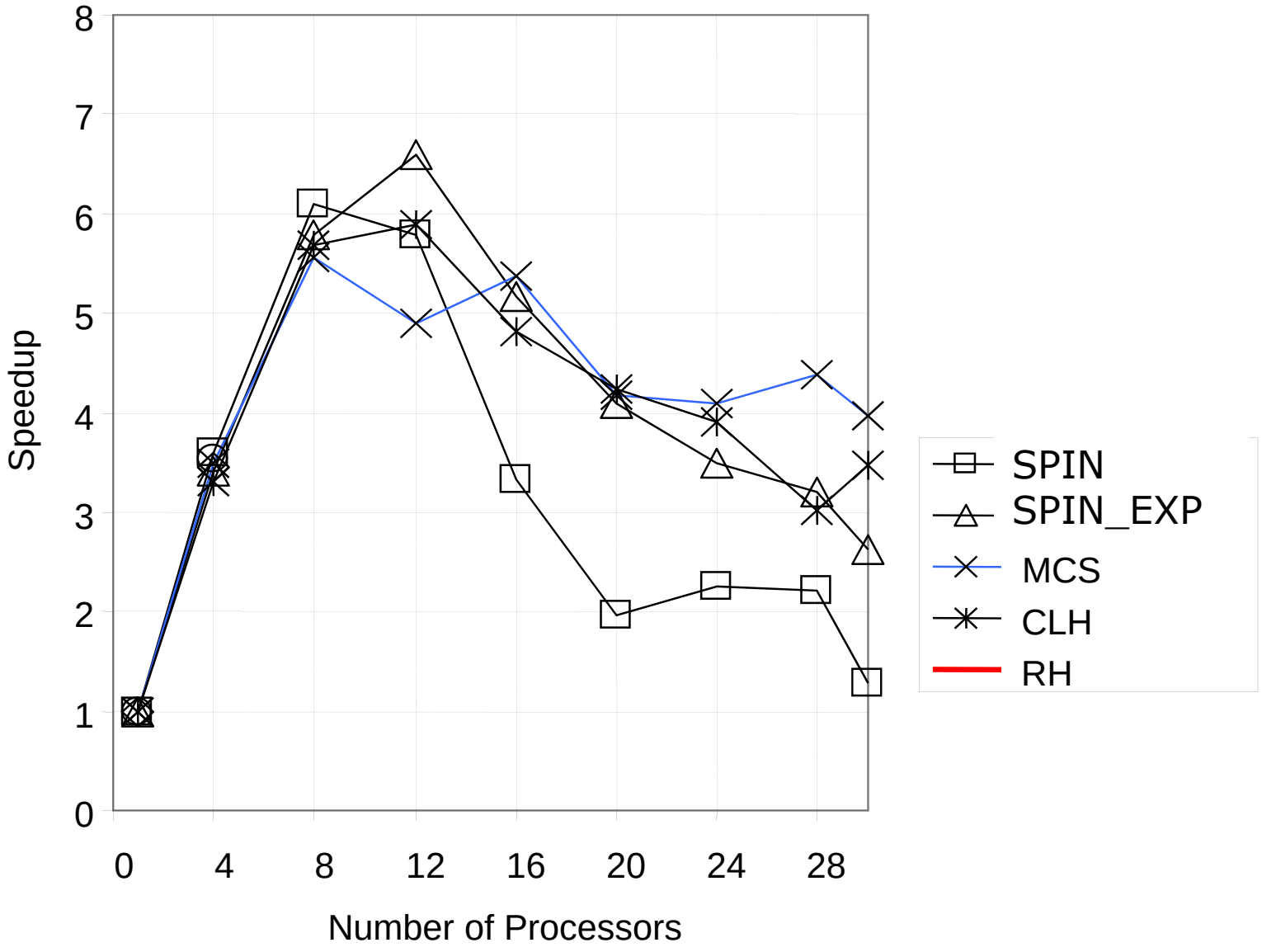
Example, parallelization:
If 99% of the code is parallel → UB: 100x

If 5% of parallel execution is spent in a CS: → UB: 20x

If CS time + handover is 10% of execution
→ UB: 10x

If the traffic generated due to CS changes is 20% of total BW → UB: 5x

Ex: Splash Raytrace Application Speedup



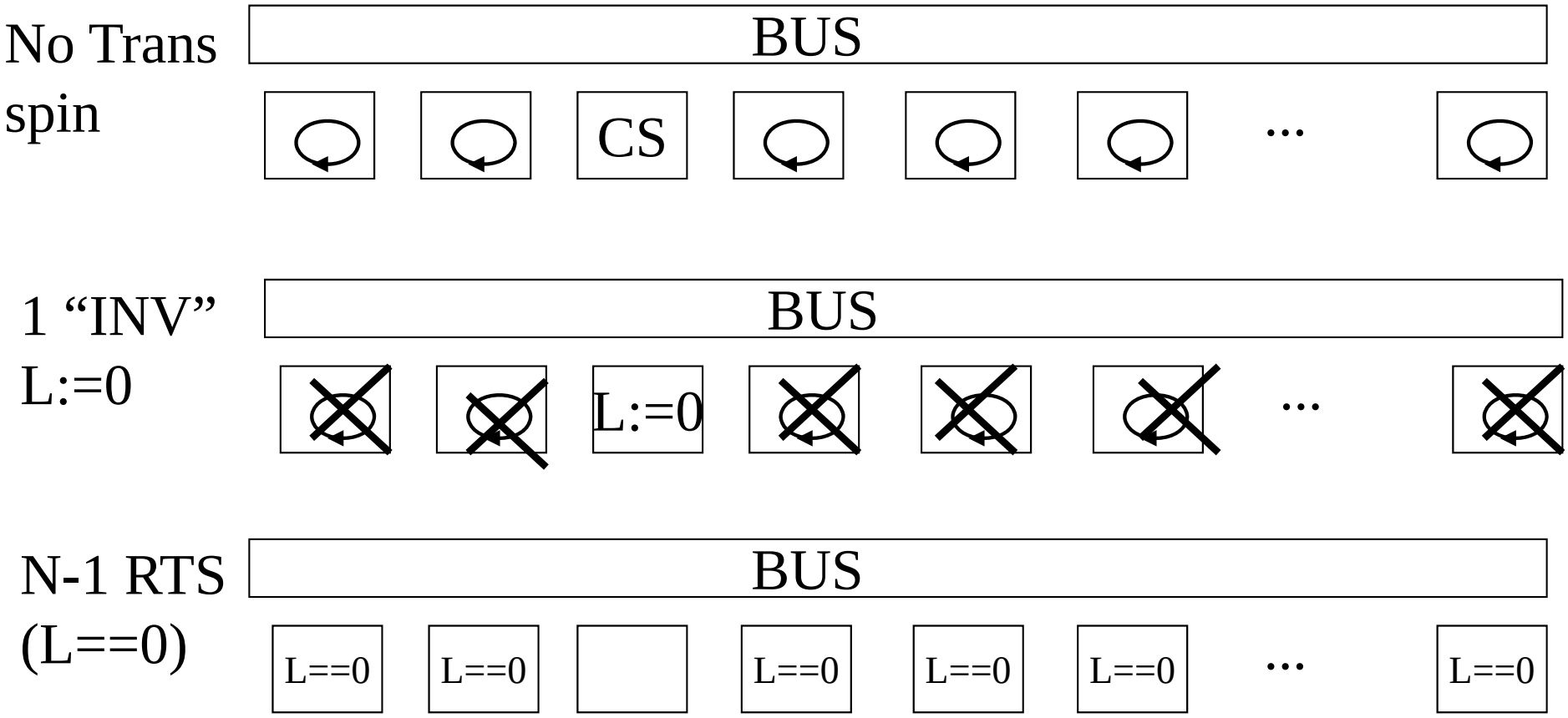
UPPSALA UNIVERSITET

Optimistic Test&Set Lock ”spinlock”

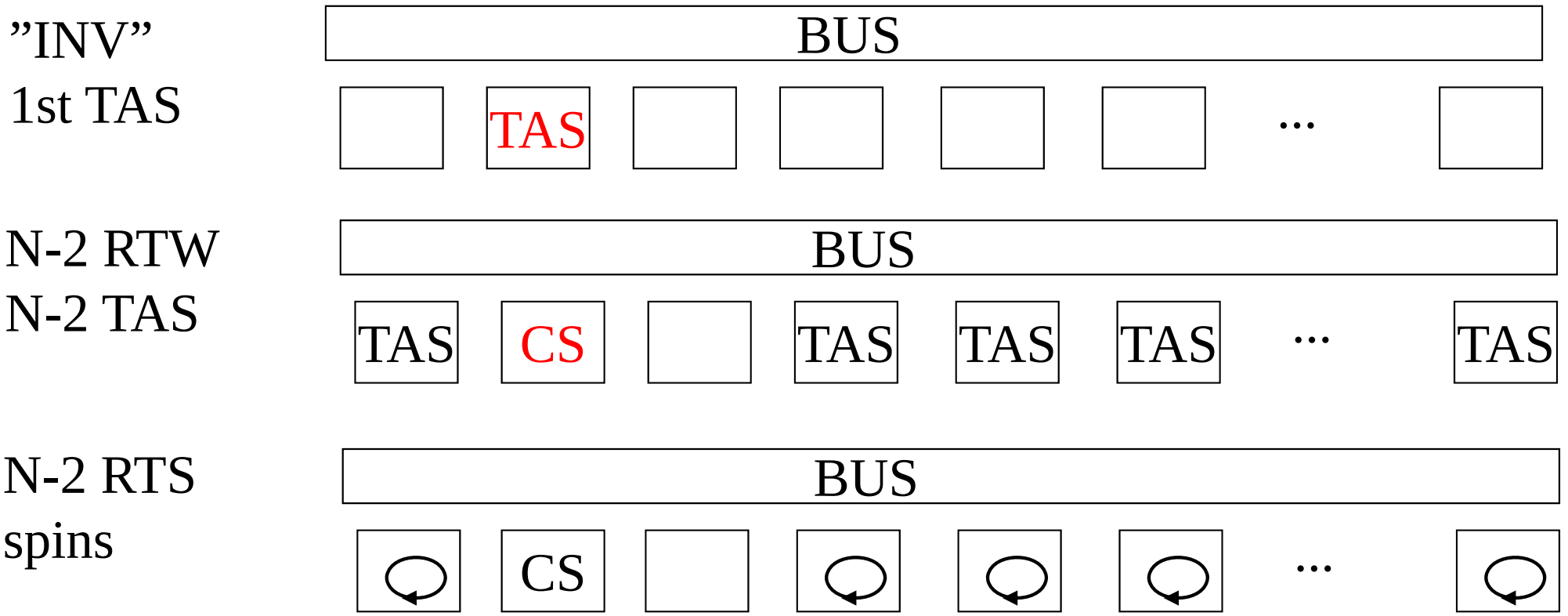
```
proc lock(lock_variable) {
    while true {
      if (TAS[lock_variable] ==0)  break;  /* Pound on the lock once, done if TAS==0 */
      while(lock_variable != 0) {}          /* spin locally in your cache until ”0” observed*
   }
}

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

AVDARK 2013

It could still get messy!
(Assuming round-robin snooping bus)



...messy (part 2)



Problem 1: Snoop activity slows down the hand-over

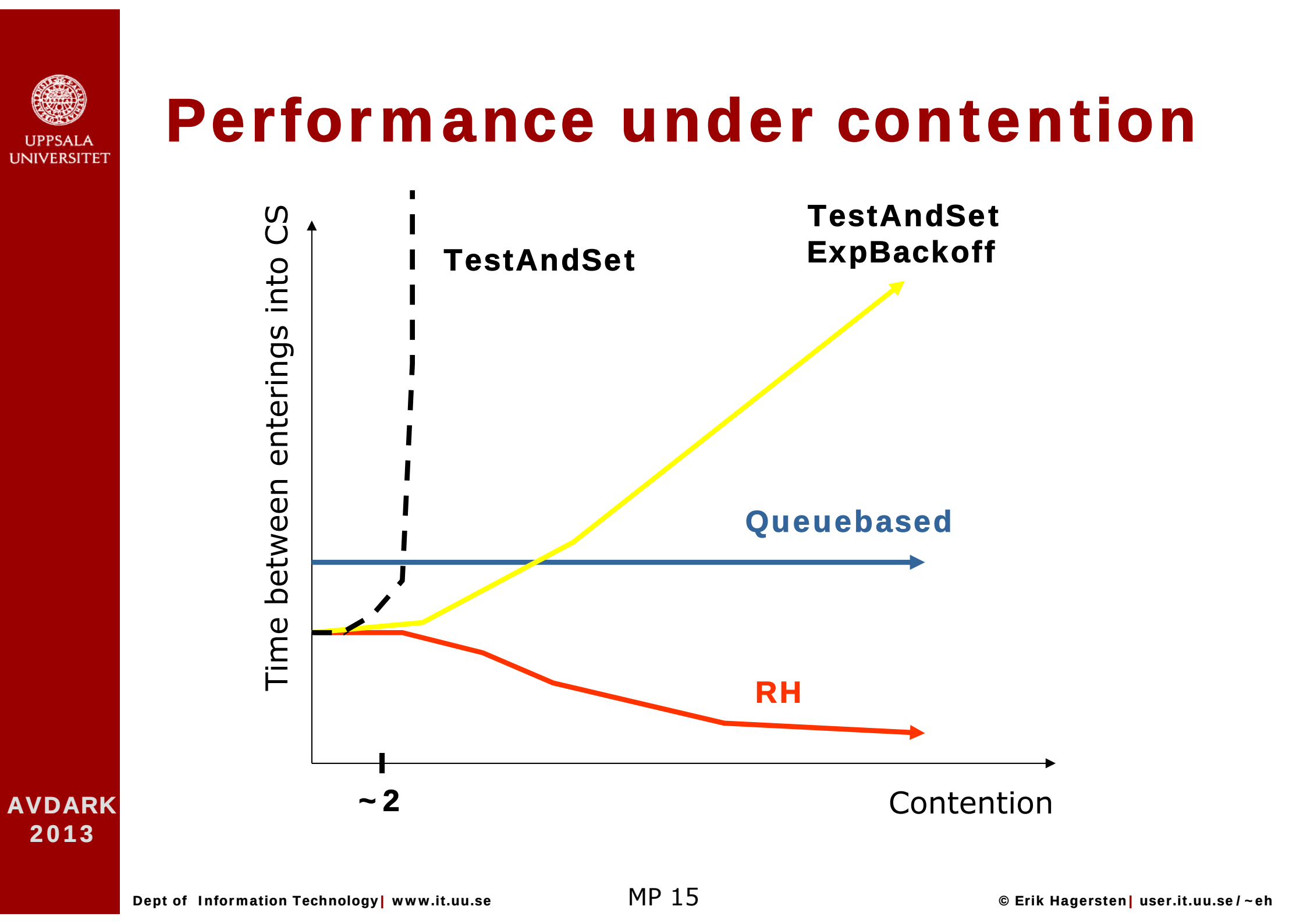
Problem 2: The new winner in CS is slowed down by more traffic

How many bus transaction between exit CS until a new CS entry (incl. lock/unlock)

- 1 "INV"
- (N-1) RTS + (N – 2) RTW
- "INV" + (N-1) RTS + "INV"

How many total bus transaction are queued up when CS changes "owner"

- 2 "INV" + (N-1) RTS + (N-2) RTW + (N-2) RTS
- (N-1) RTS + (N – 1) RTW
- "INV" + (N-1) RTS + "INV"


UPPSALA UNIVERSITET
Performance under contention
TestAndSet
TestAndSet ExpBackoff
Queuebased
RH
Time between enterings into CS
~2
Contention
AVDARK 2013
Dept of Information Technology| www.it.uu.se
MP 15
© Erik Hagersten| user.it.uu.se/~eh