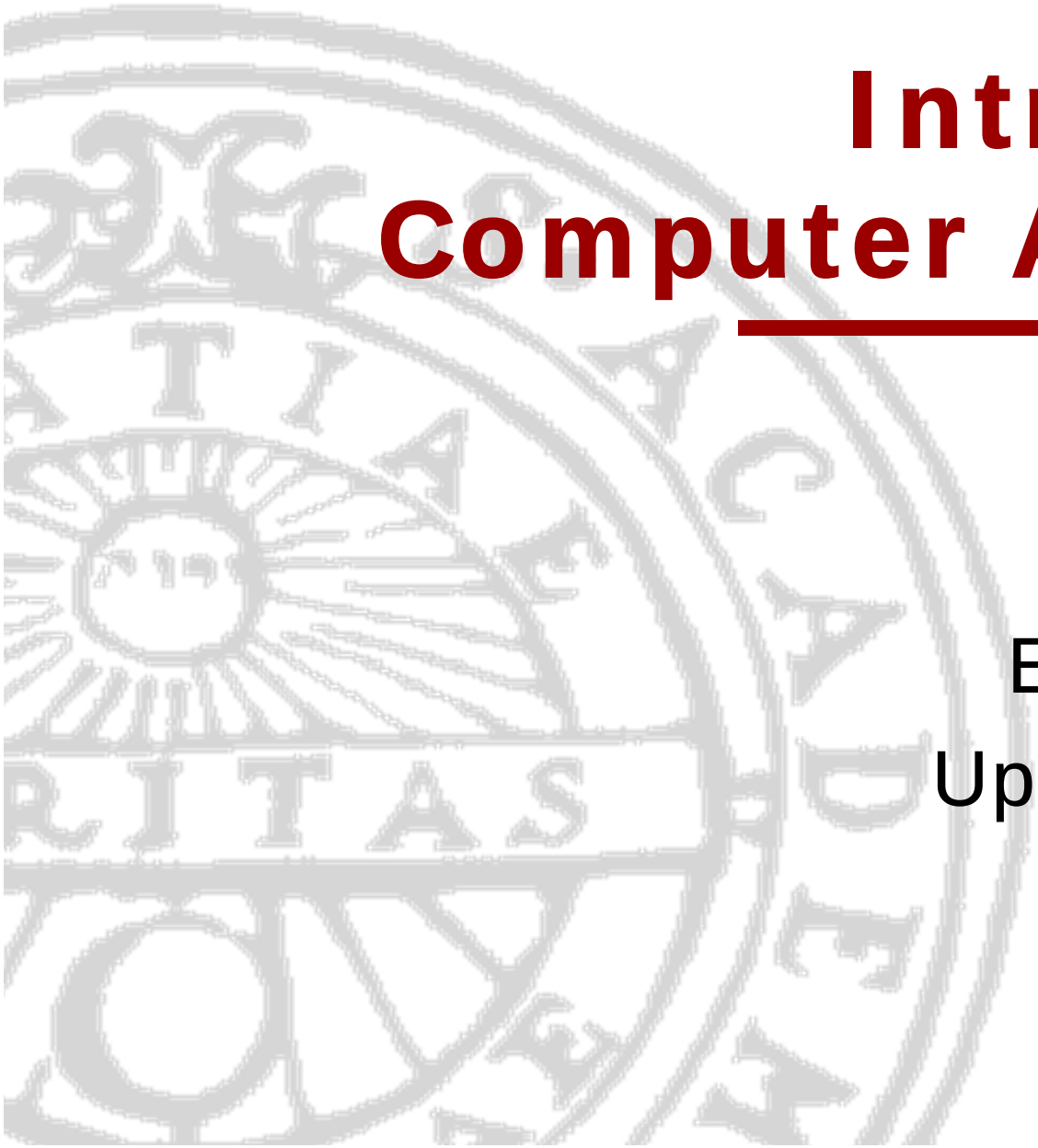


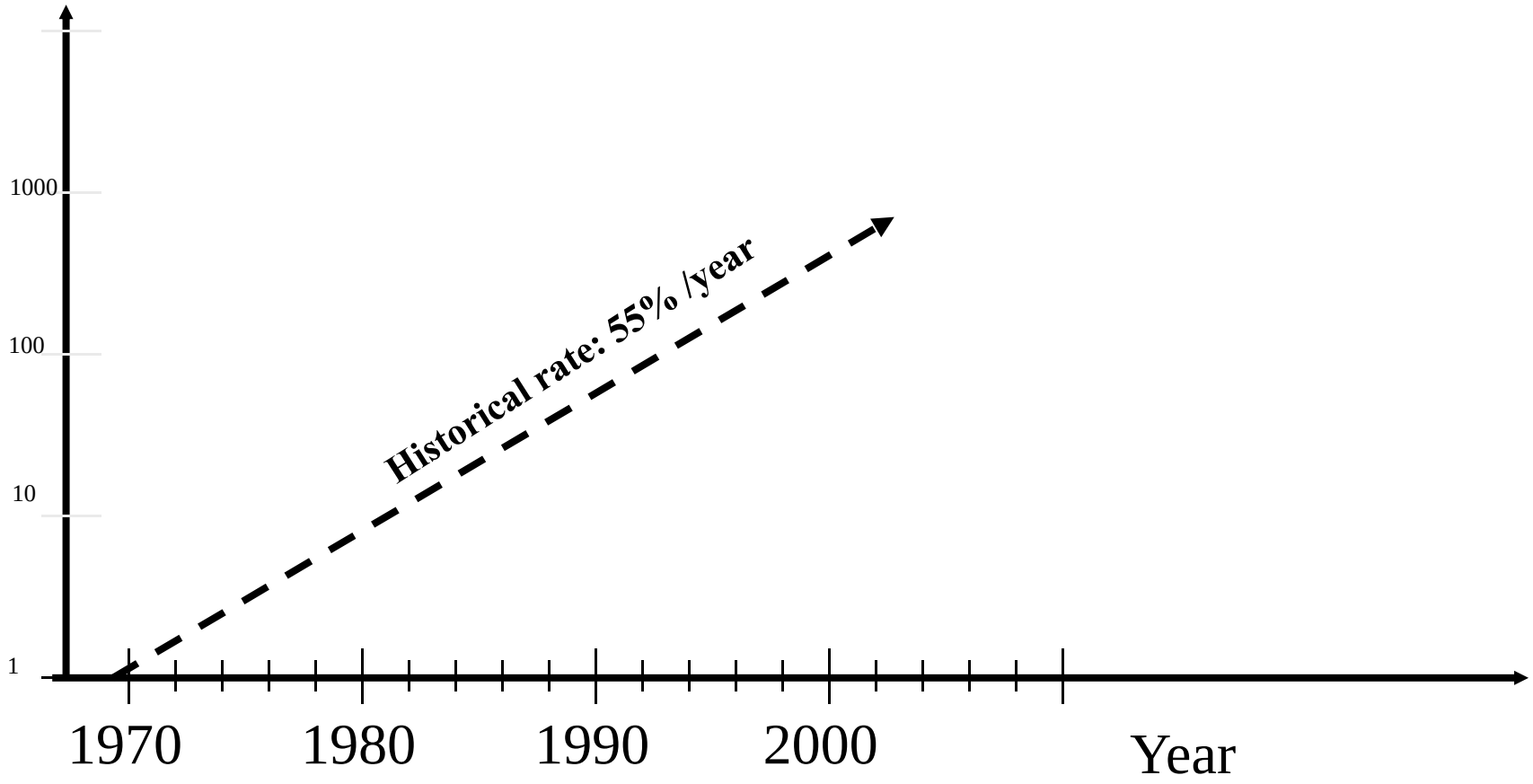
Introduction to Computer Architecture

Erik Hagersten
Uppsala University

UPPSALA UNIVERSITET

CPU Improvements

Relative Performance
[log scale]



Parallel Comp 2012

UPPSALA UNIVERSITET

How to get efficient architectures…

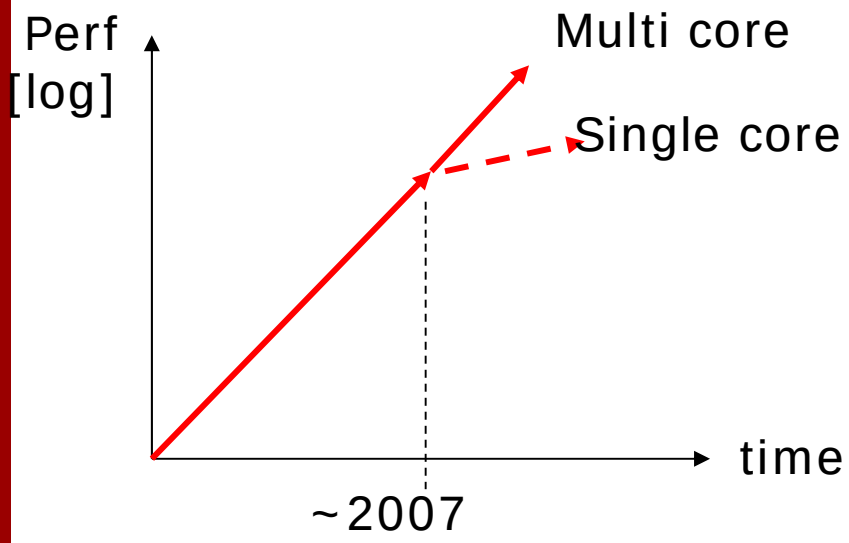
Uncool today

- ~~Increase clock rate~~
- Create and explore locality:
  a) Spatial locality
  b) Temporal locality
  c) Geographical locality
- Create and explore parallelism
  a) Instruction level parallelism (ILP)
  b) Thread level parallelism (TLP)
  c) Memory level parallelism (MLP)

© Erik Hagersten| user.it.uu.se/~eh

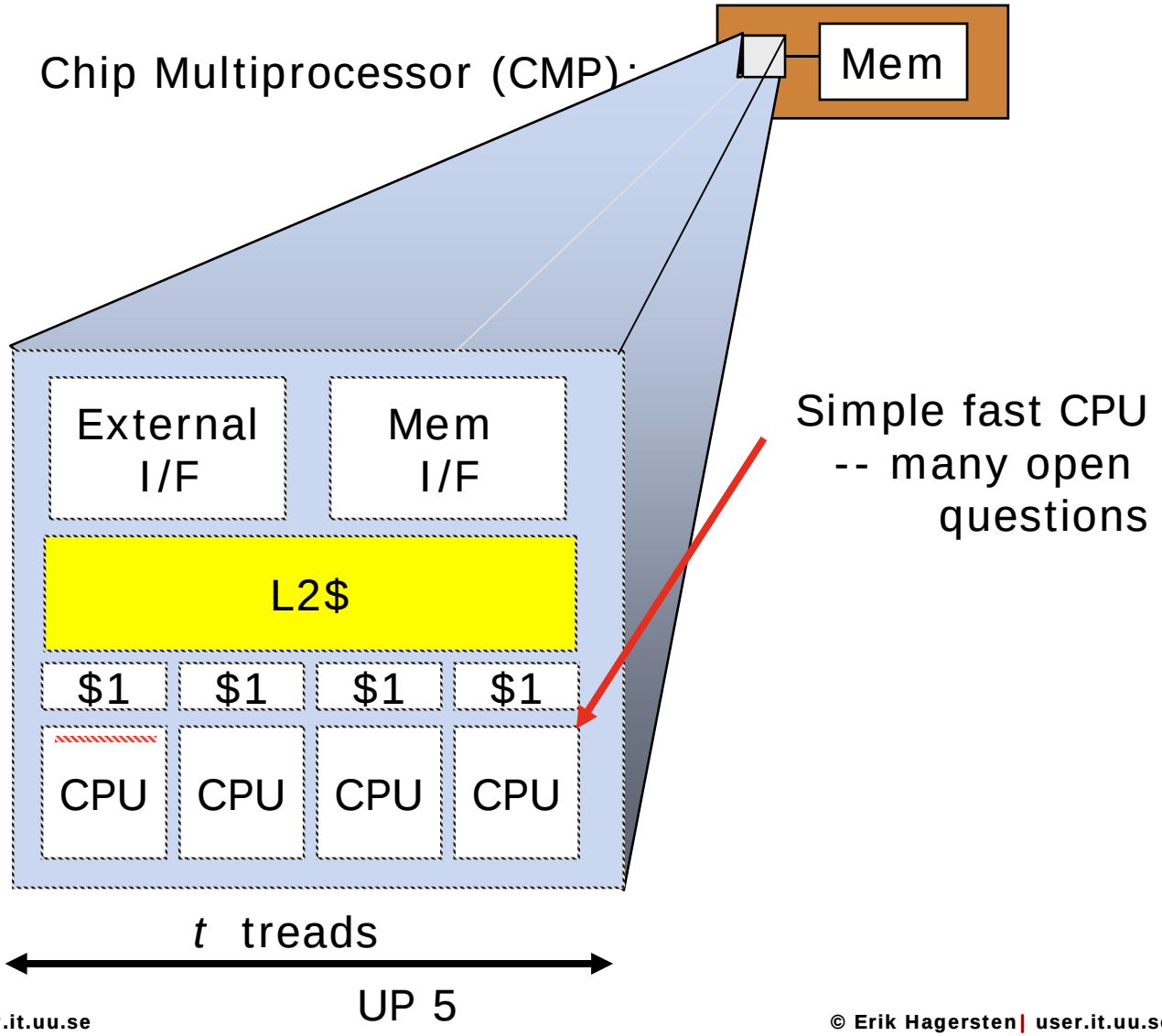
UPPSALA UNIVERSITET

Why Multicores?

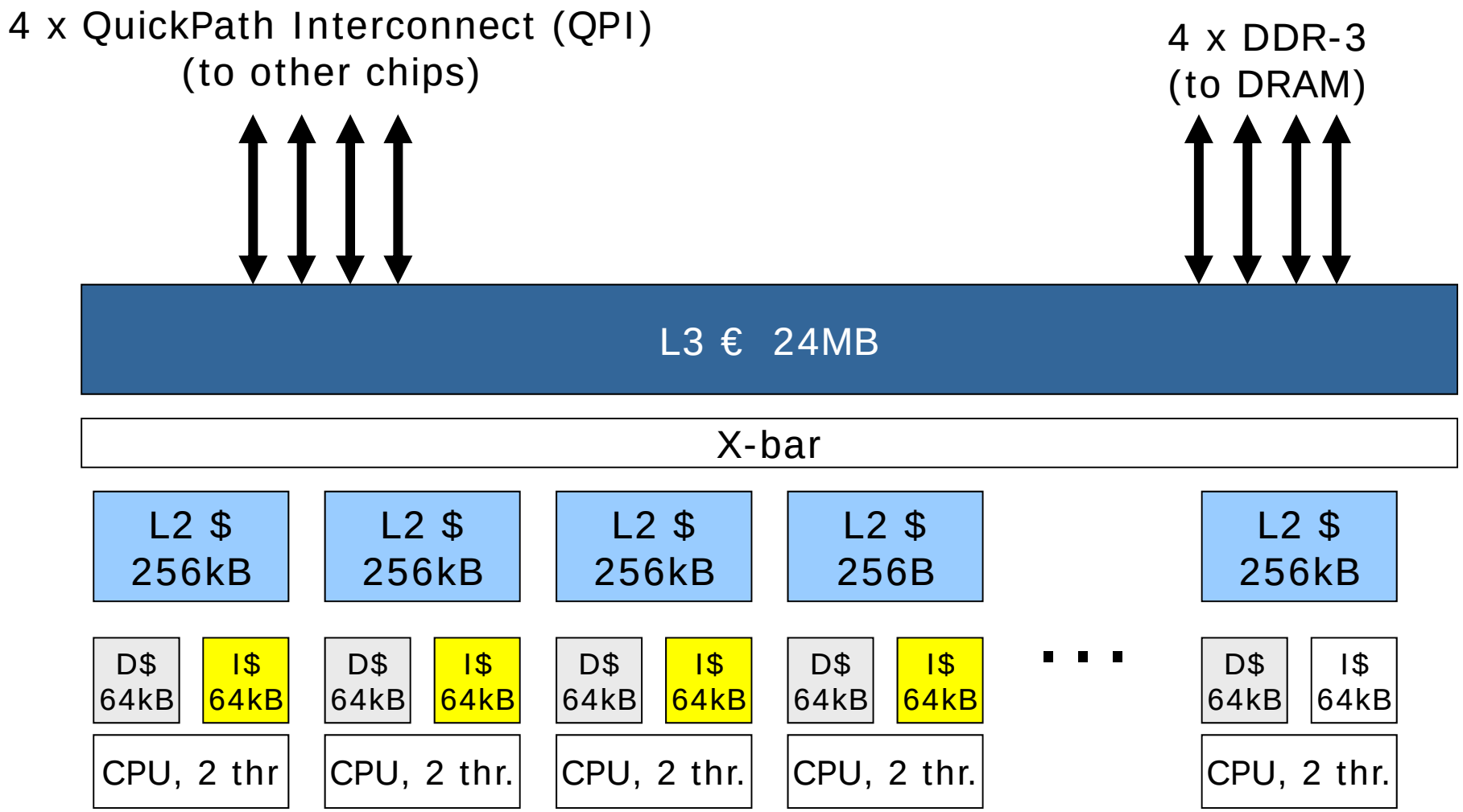


1. Not enough ILP & MLP in a single thread
2. Signal propagation delay » transistor delay
3. Power consumption

Chip Multiprocessor / Multicores



Intel: ”Nehalem-Ex” (i7)

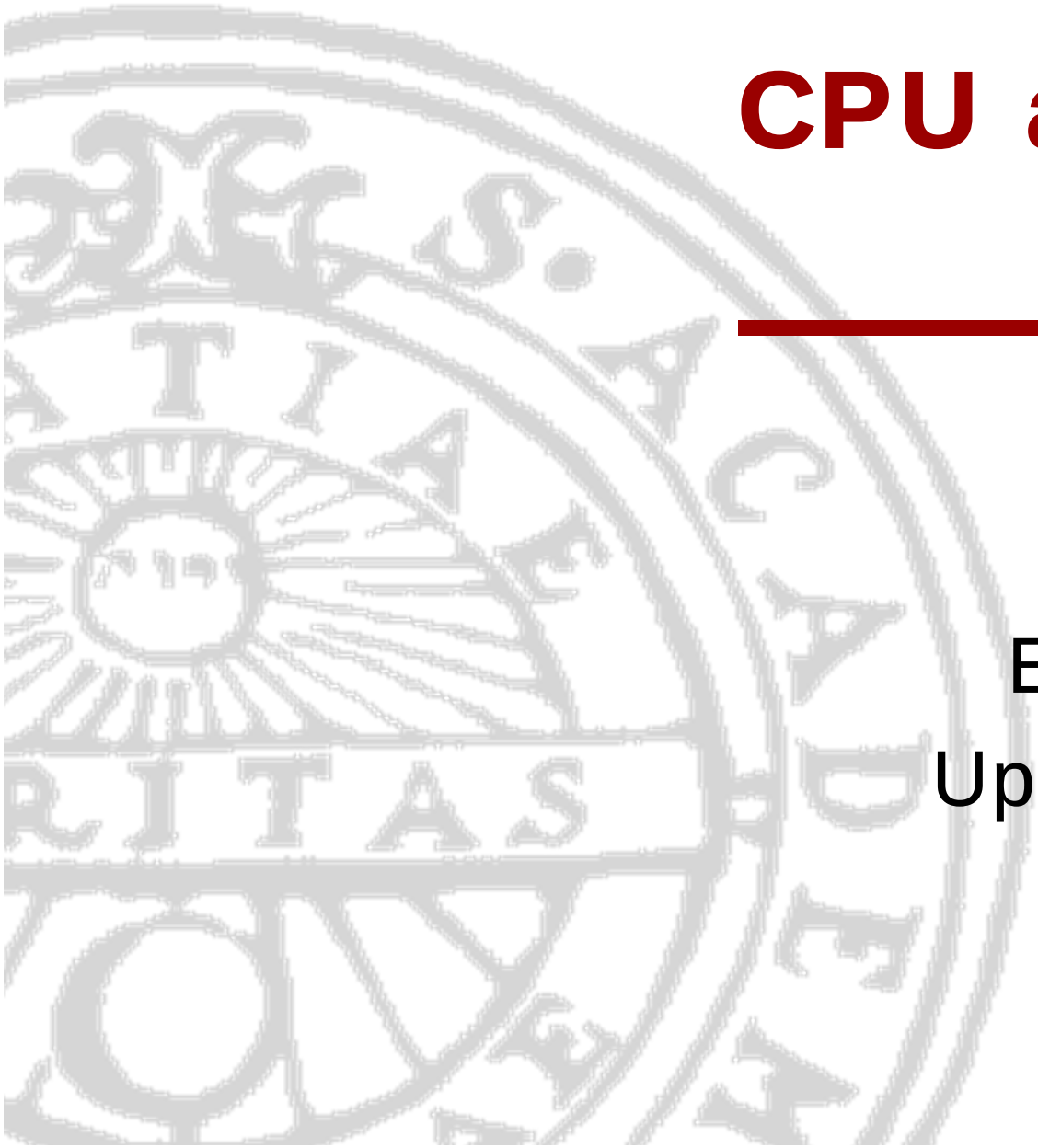


8 cores x 2 threads (SMT)

UPPSALA UNIVERSITET

Outline of these lectures

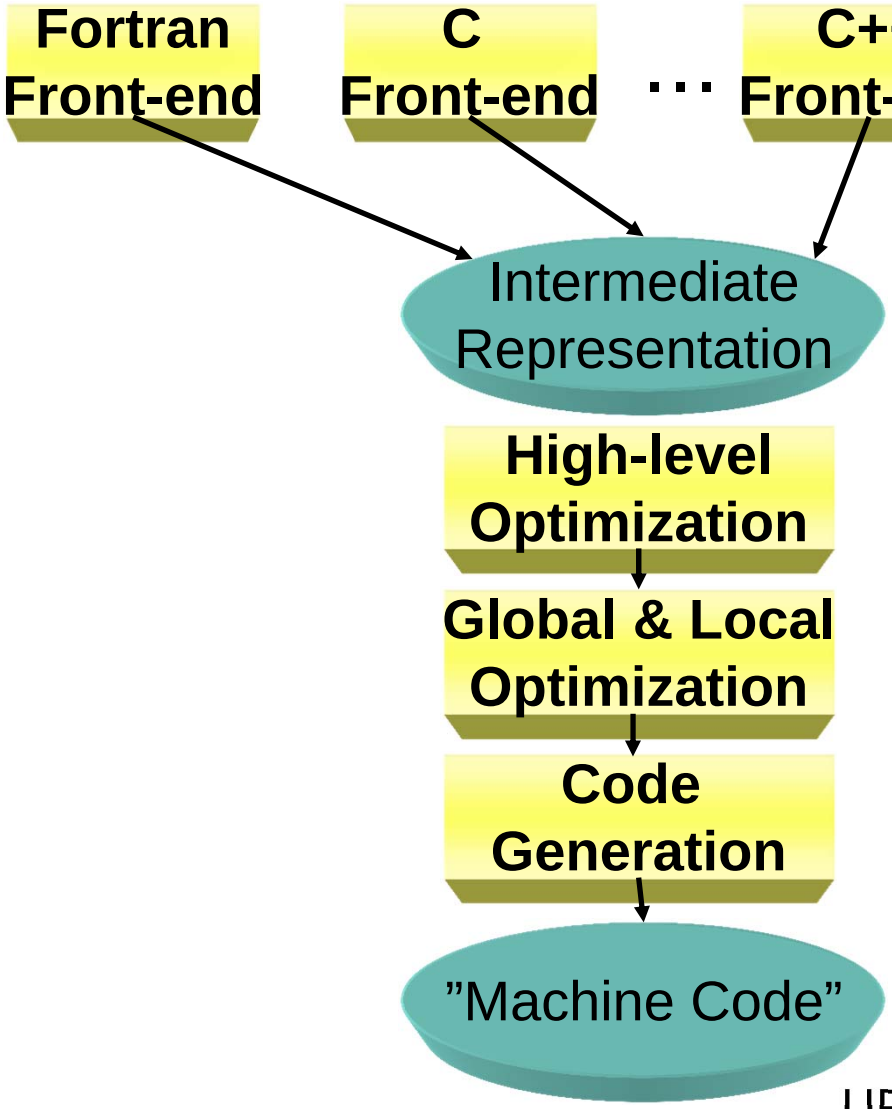
1. Uniprocessors
   - CPUs & pipelines: 3 problems
   - Memory, caches, VM, disks
   - Out-of-order execution
   - Branch prediction
2. Multiprocessors
3. Multicores & Manycores
4. Optimizing for speed



CPU architecture overview

Erik Hagersten
Uppsala University

Compiler Organization



Machine-independent Translation

Procedure in-lining
Loop transformation

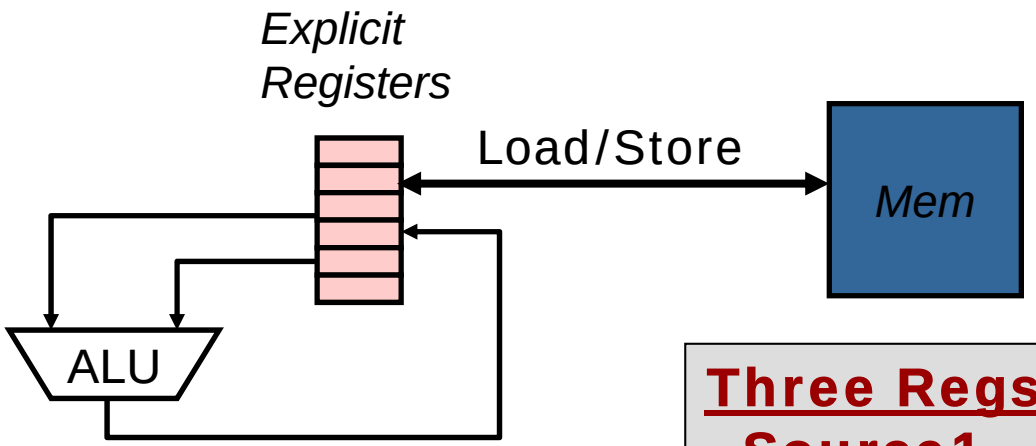
Register Allocation
Common sub-expressions

Instruction selection
constant folding

Load/Store architecture (e.g., "RISC")

ALU ops: Reg -->Reg

Mem ops: Reg <--> Mem



Three Regs:
- **Source1**
- **Source2**
- **Destination**

Example: C = A + B → **Compiler**

```
LD R1, [A]
LD R3, [B]
ADD R2, R1, R3
ST R2, [C]
```

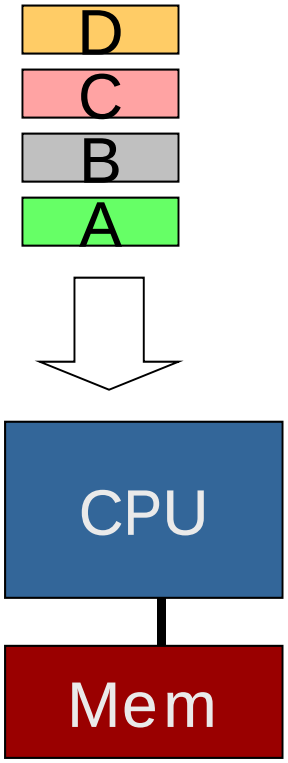
Memory Accesses

© Erik Hagersten| user.it.uu.se/~eh

UPPSALA UNIVERSITET

Lifting the CPU hood (simplified…)

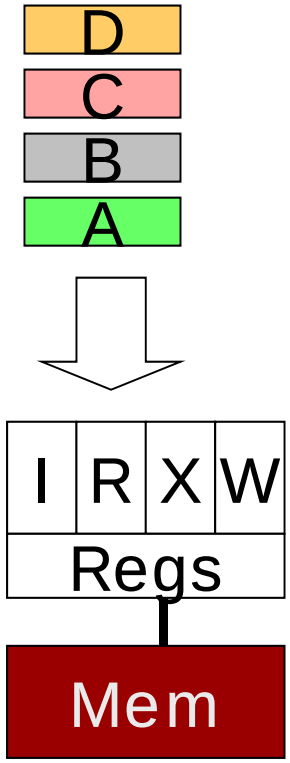
Instructions:



UPPSALA UNIVERSITET

Pipeline

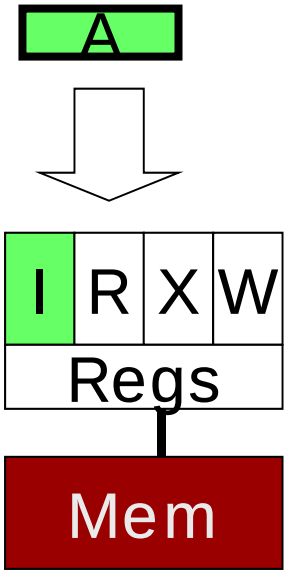
Instructions:



© Erik Hagersten| user.it.uu.se/~eh

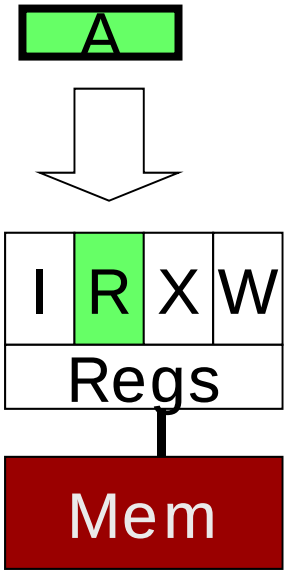
UPPSALA UNIVERSITET

Pipeline

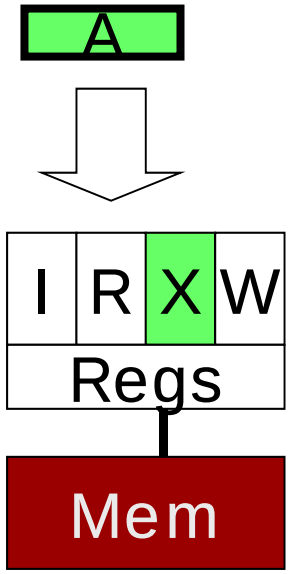


UPPSALA UNIVERSITET

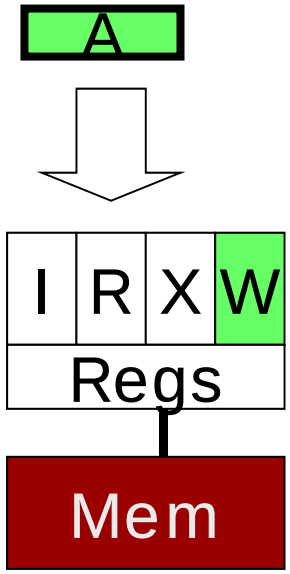
Pipeline



Pipeline



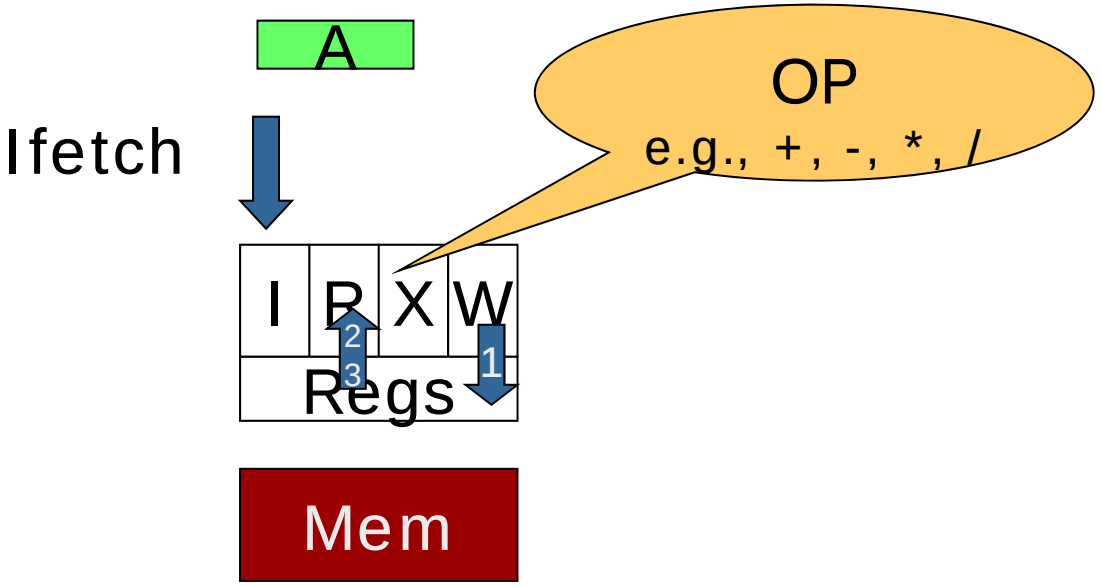
Pipeline:



I = Instruction fetch
R = Read register
X = Execute
W= Write register/mem

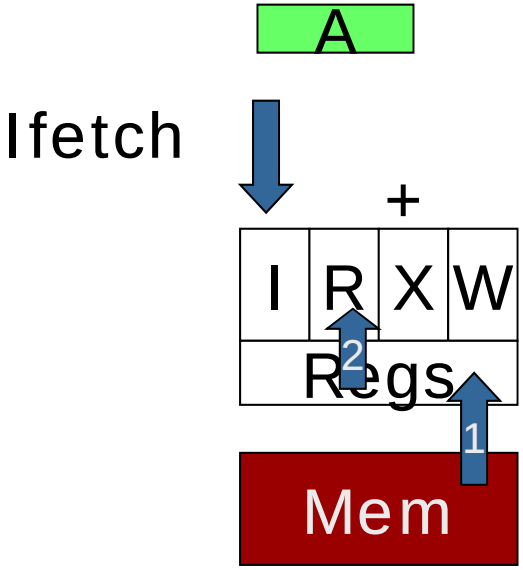
Register Operations [aka ALU operation]

ADD R1, R2, R3
a.k.a. R1 := R2 op R3



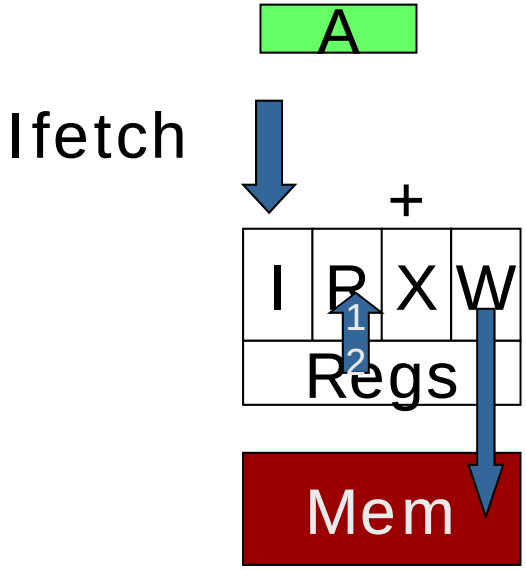
Load Operation:

LD R1, mem[cnst+R2]



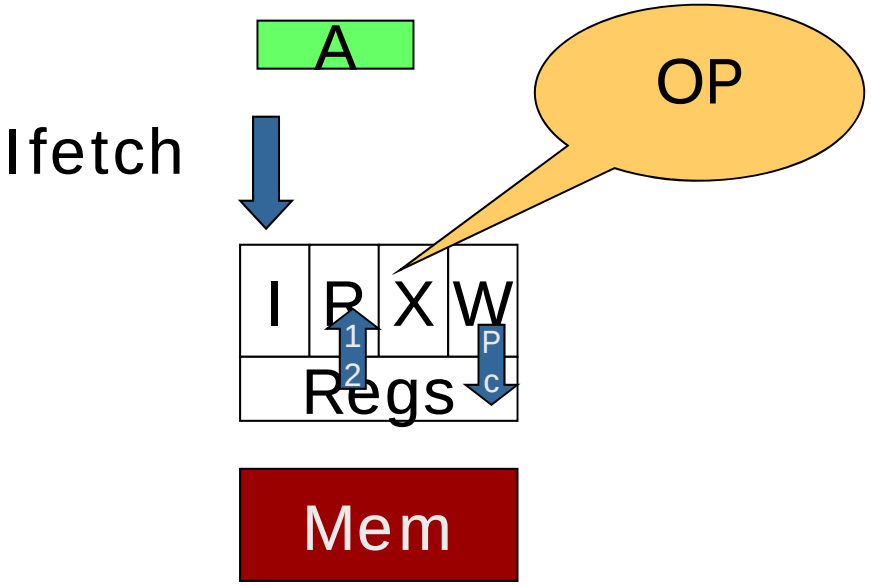
Store Operation:

ST R2, mem[cnst+R1]



Branch Operations:

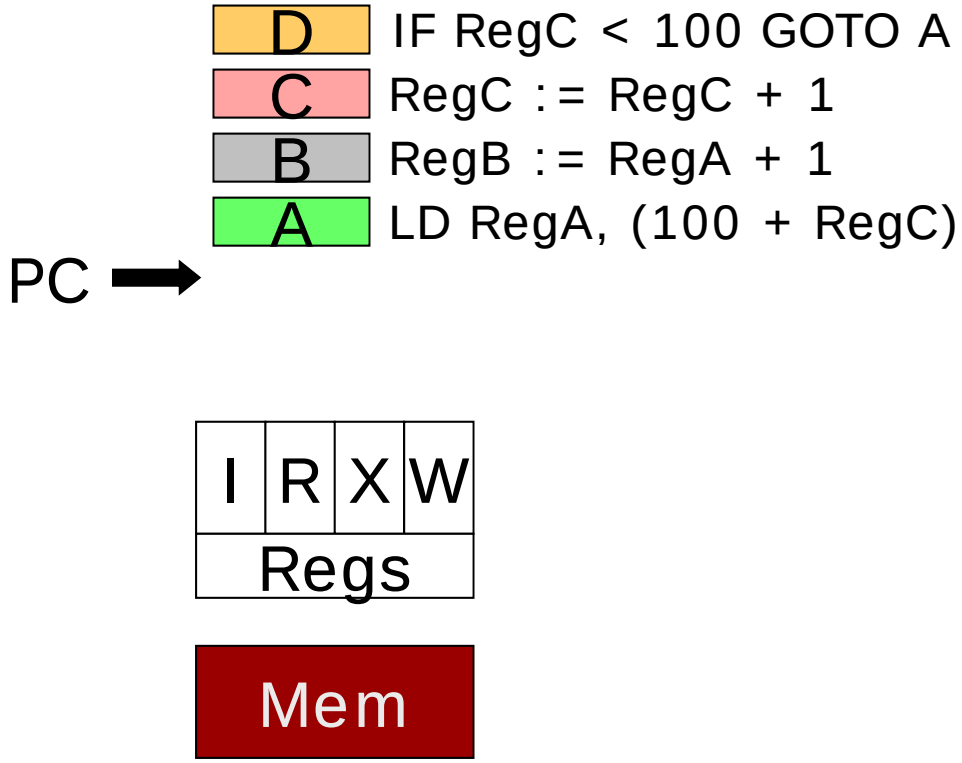
if (R1 Op Const) GOTO mem[R2]



PC = Program Counter.
A special register pointing to the next instruction to execute

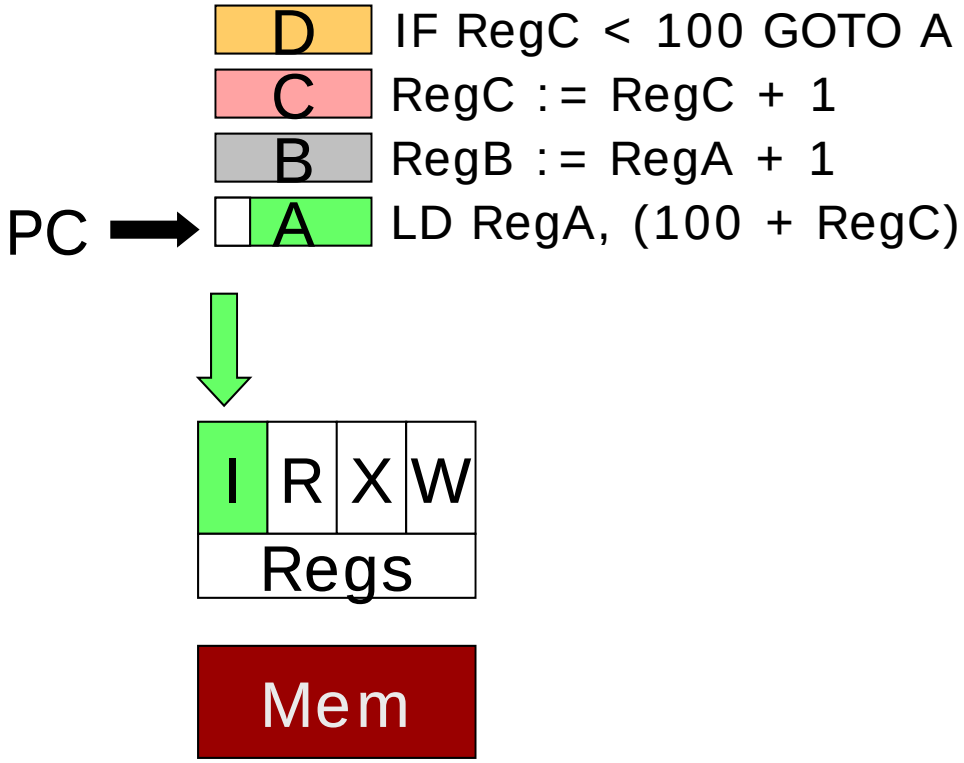
UPPSALA UNIVERSITET

Initially

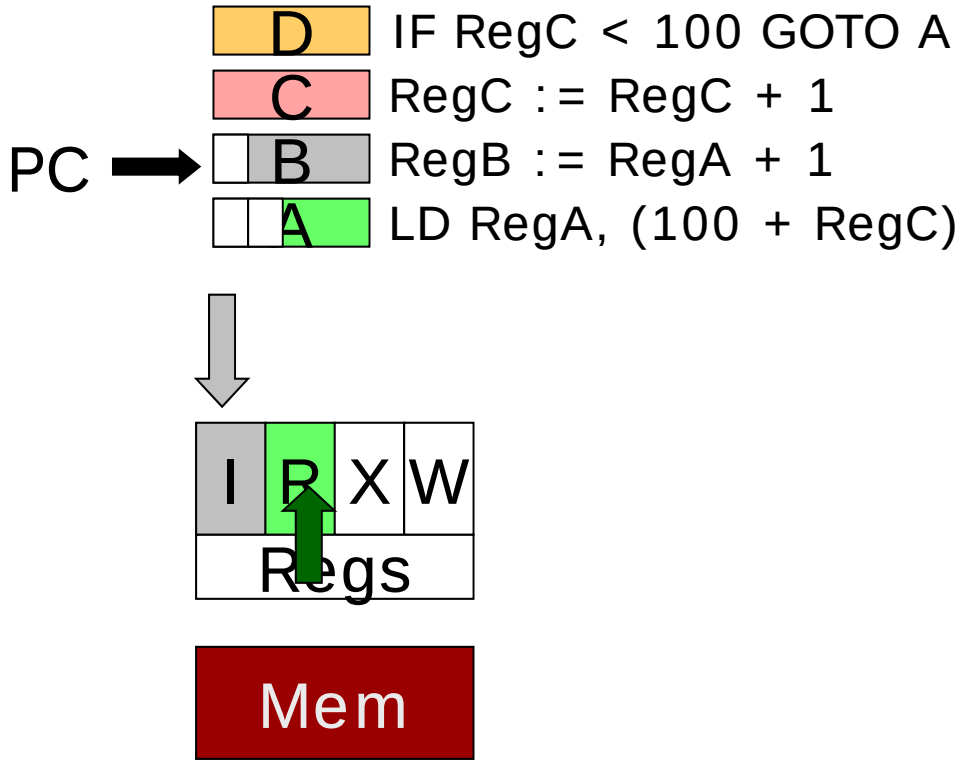



UPPSALA
UNIVERSITET

Cycle 1

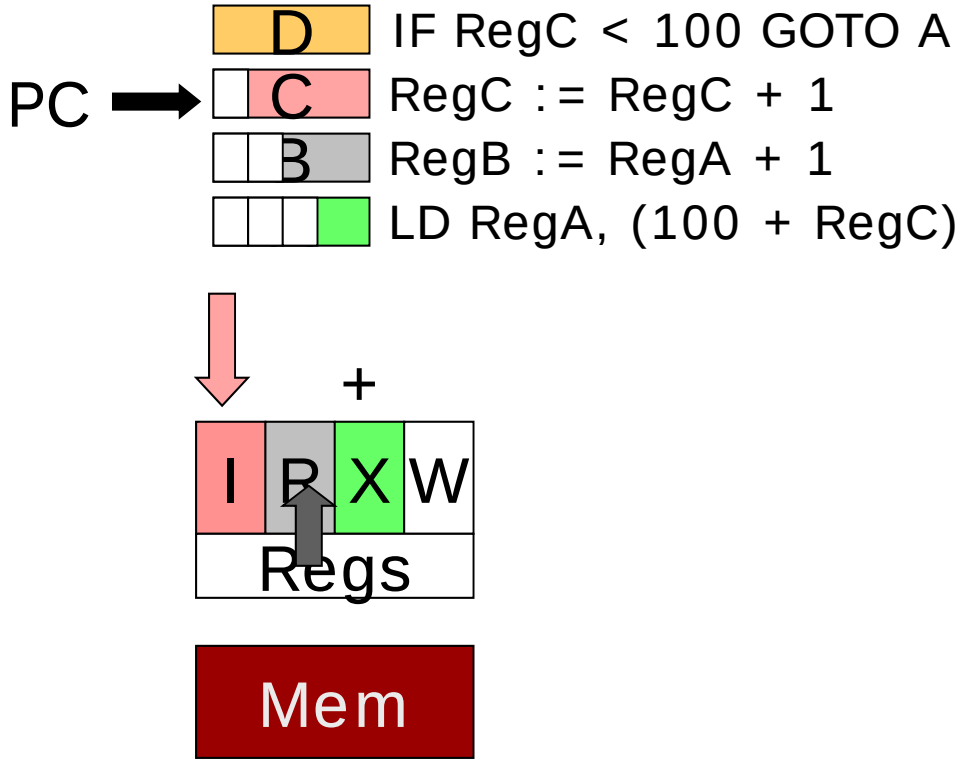

D
IF RegC < 100 GOTO A
C
RegC := RegC + 1
B
RegB := RegA + 1
PC
A
LD RegA, (100 + RegC)
I
R
X
W
Regs
Mem

Cycle 2


D
IF RegC < 100 GOTO A
C
RegC := RegC + 1
PC
B
RegB := RegA + 1
A
LD RegA, (100 + RegC)
I
R
X
W
Regs
Mem

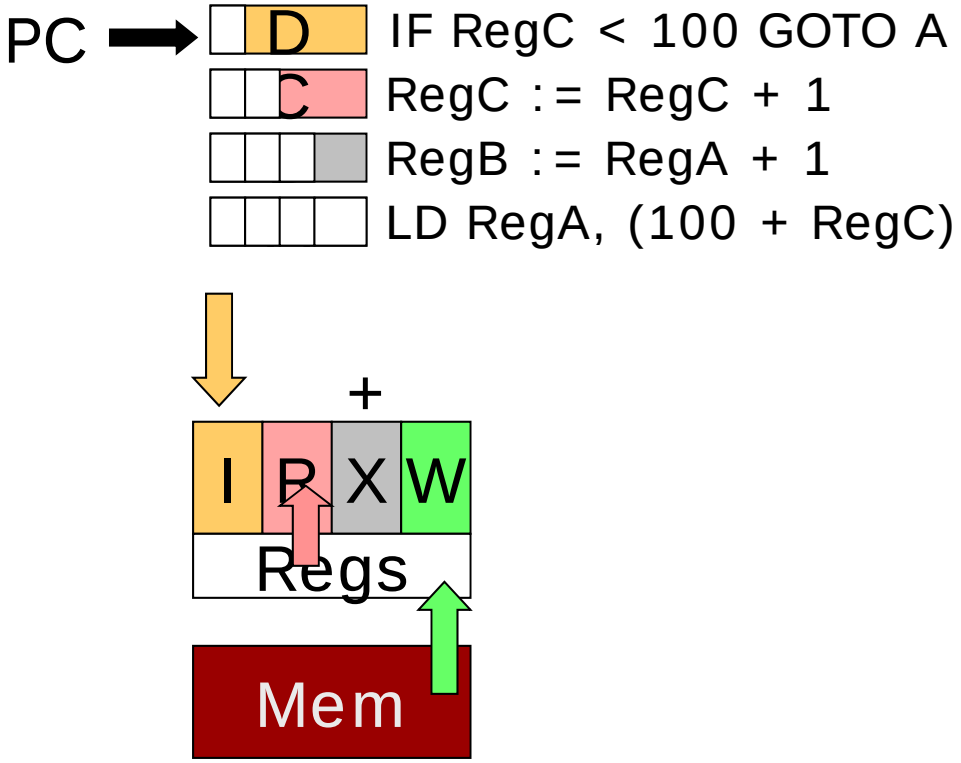

UPPSALA
UNIVERSITET

Cycle 3


D
IF RegC < 100 GOTO A
PC
C
RegC := RegC + 1
B
RegB := RegA + 1
LD RegA, (100 + RegC)
+
I
B
X
W
Regs
Mem

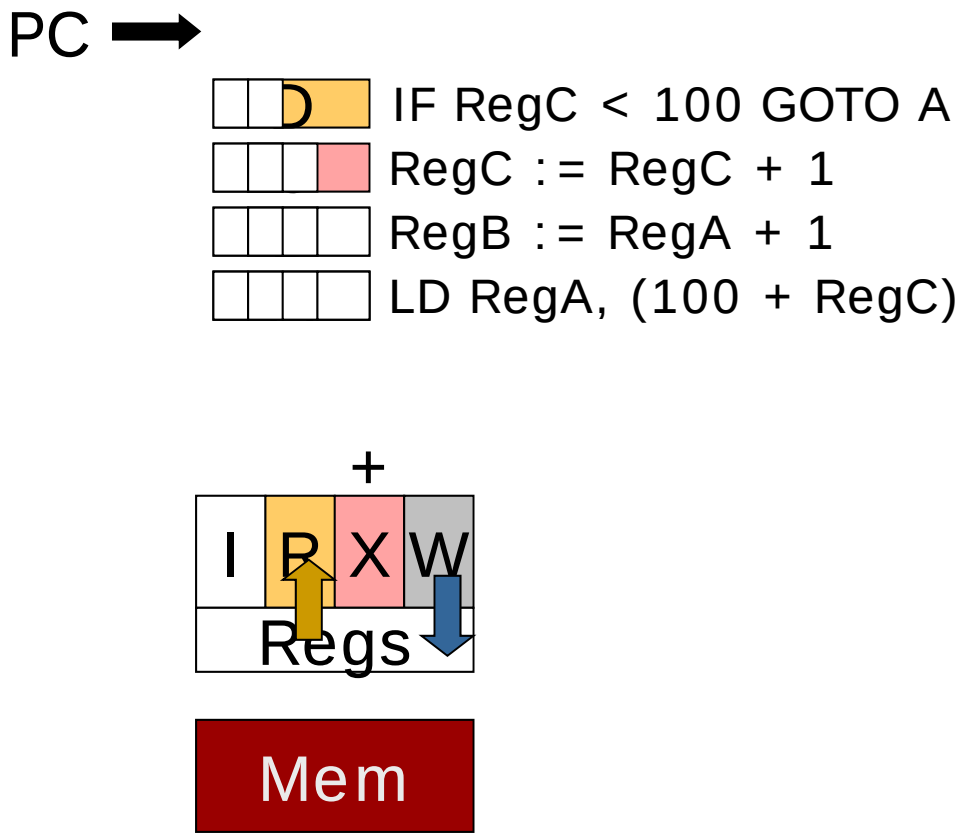
UPPSALA UNIVERSITET

Cycle 4

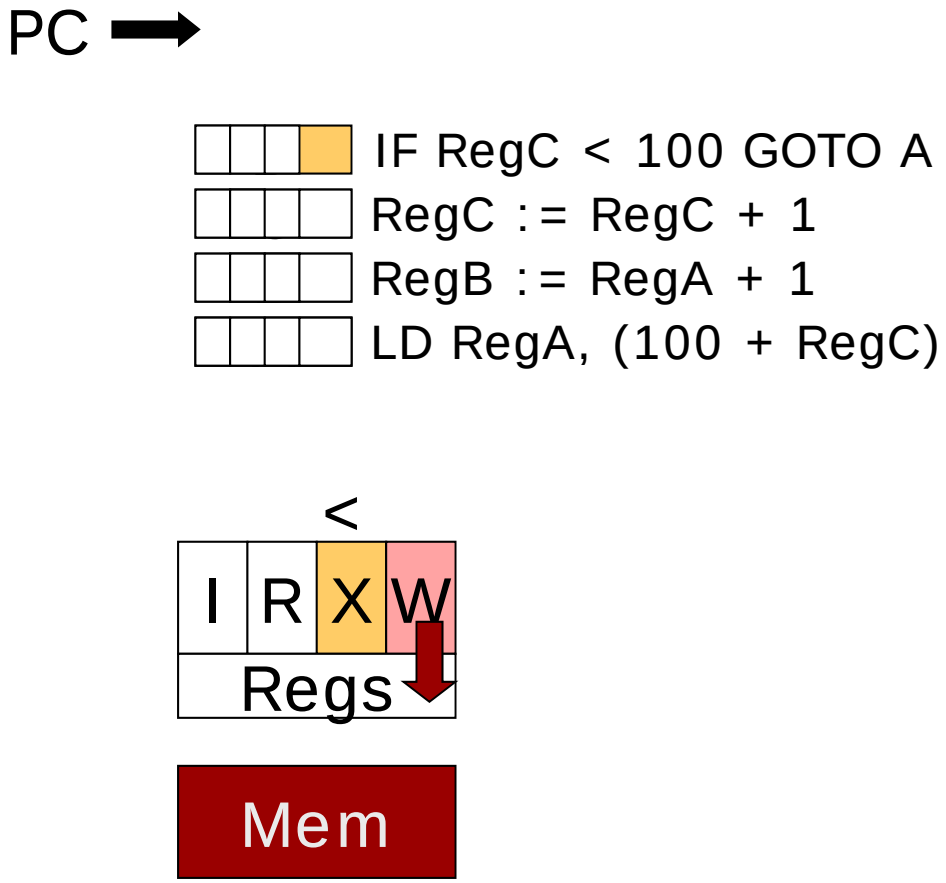



UPPSALA
UNIVERSITET

Cycle 5


PC
IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)
+
I
R
X
W
Regs
Mem

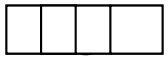
Cycle 6


PC
IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)
<
I
R
X
W
Regs
Mem

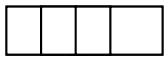
Cycle 7

PC ➡

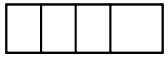
IF RegC < 100 GOTO A



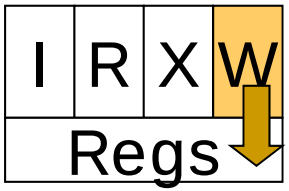
RegC := RegC + 1



RegB := RegA + 1

A: 

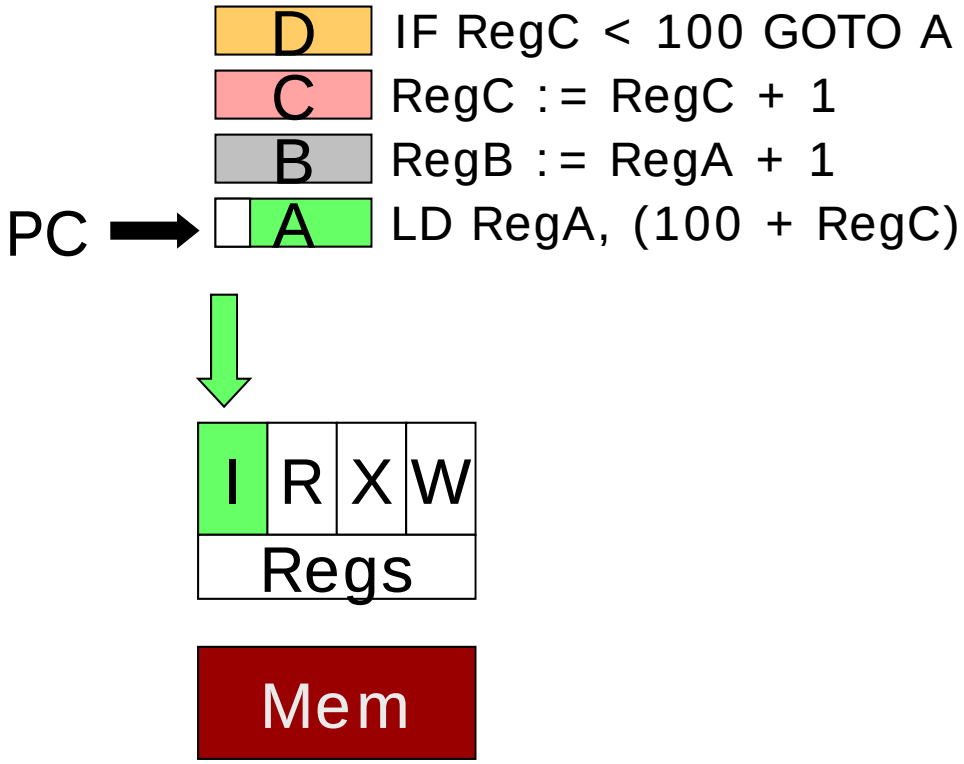
LD RegA, (100 + RegC)



Branch: Addr(A) ➔ PC


UPPSALA
UNIVERSITET

Cycle 8


D
IF RegC < 100 GOTO A
C
RegC := RegC + 1
B
RegB := RegA + 1
PC
A
LD RegA, (100 + RegC)
I
R
X
W
Regs
Mem

UPPSALA UNIVERSITET

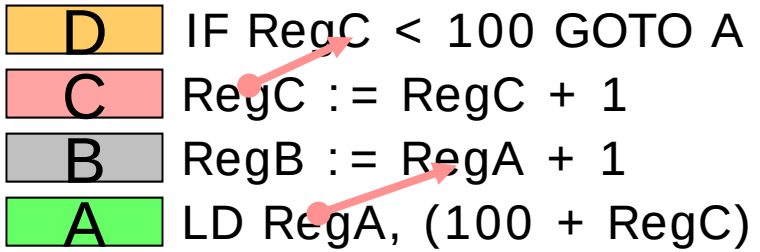
Pipeline Challenges

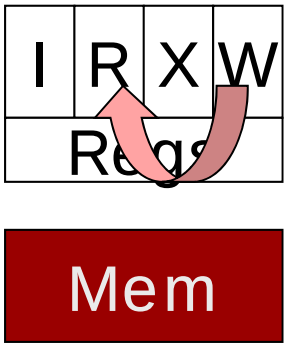
- Balance the pipeline stages
- Setup and hold time overhead
- Minimize pipeline stalls
- Predict and perform speculative work
- Undo speculative work

UPPSALA UNIVERSITET

Pipeline problem1: Data dependency

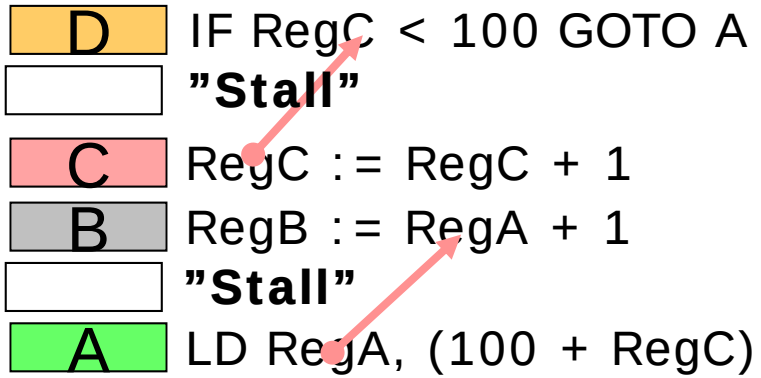
Previous execution example produce wrong results

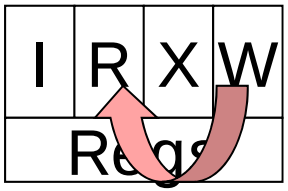




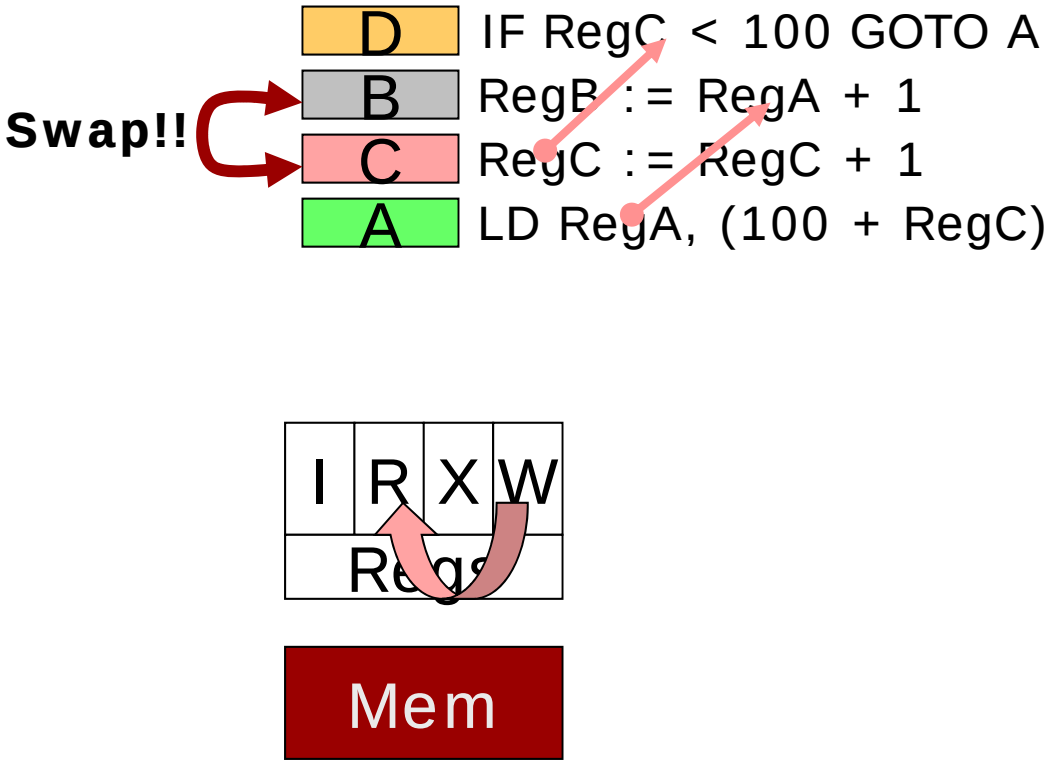
Parallel Comp 2012

Data dependency fix 1: pipeline delays (aka bubbles)

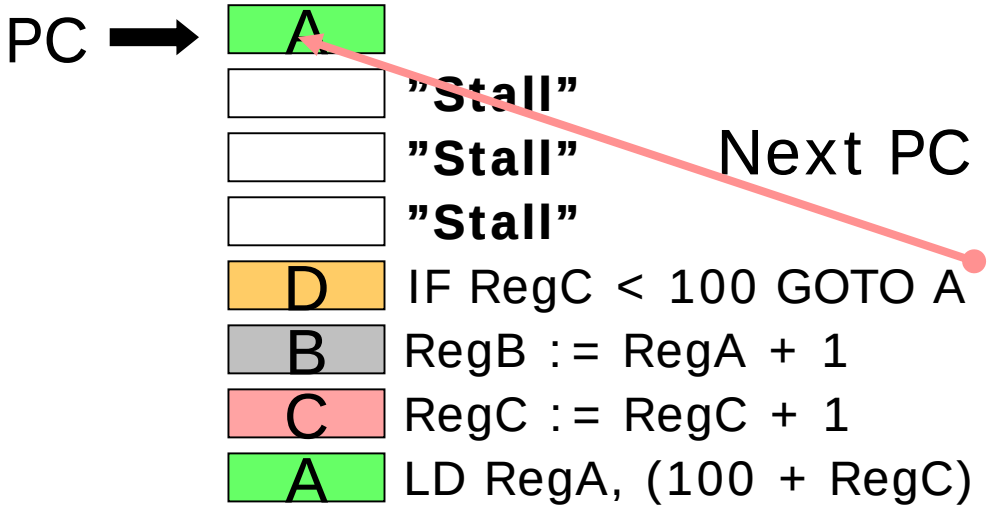


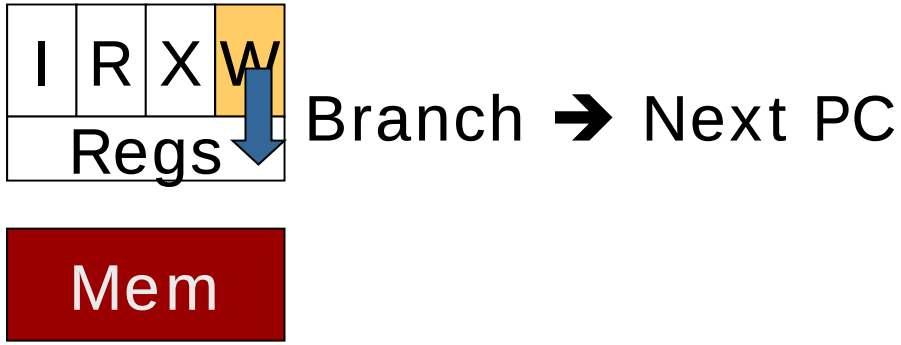


Data dependency fix 2: Compiler optimizations (sometimes)



Pipeline problem2: Branch delays ☹



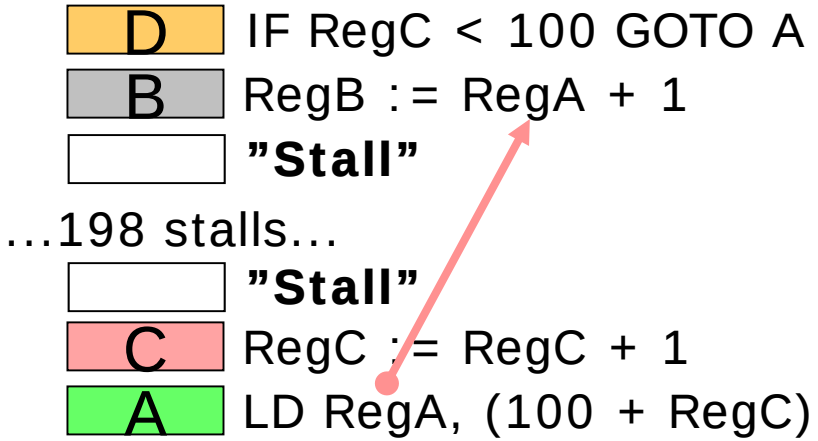


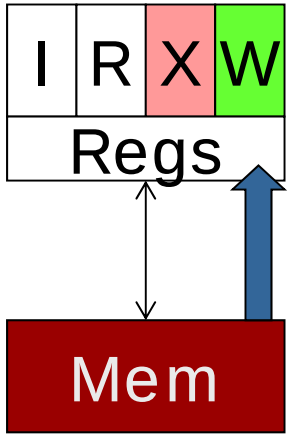
7 cycles per iteration of 4 instructions ☹

➔ Need longer basic blocks with many independent instr.

© Erik Hagersten| user.it.uu.se/~eh

Pipeline problem3: Slow Memory



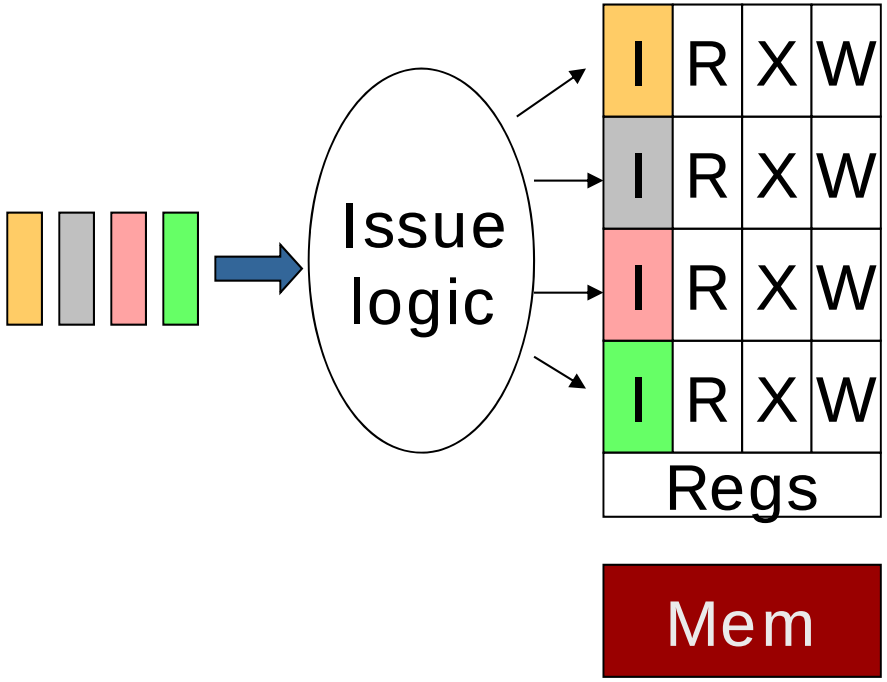


Load data from memory

~200 cycle access time

It is actually a lot worse!

Modern CPUs:"superscalars" with ~4 parallel pipelines

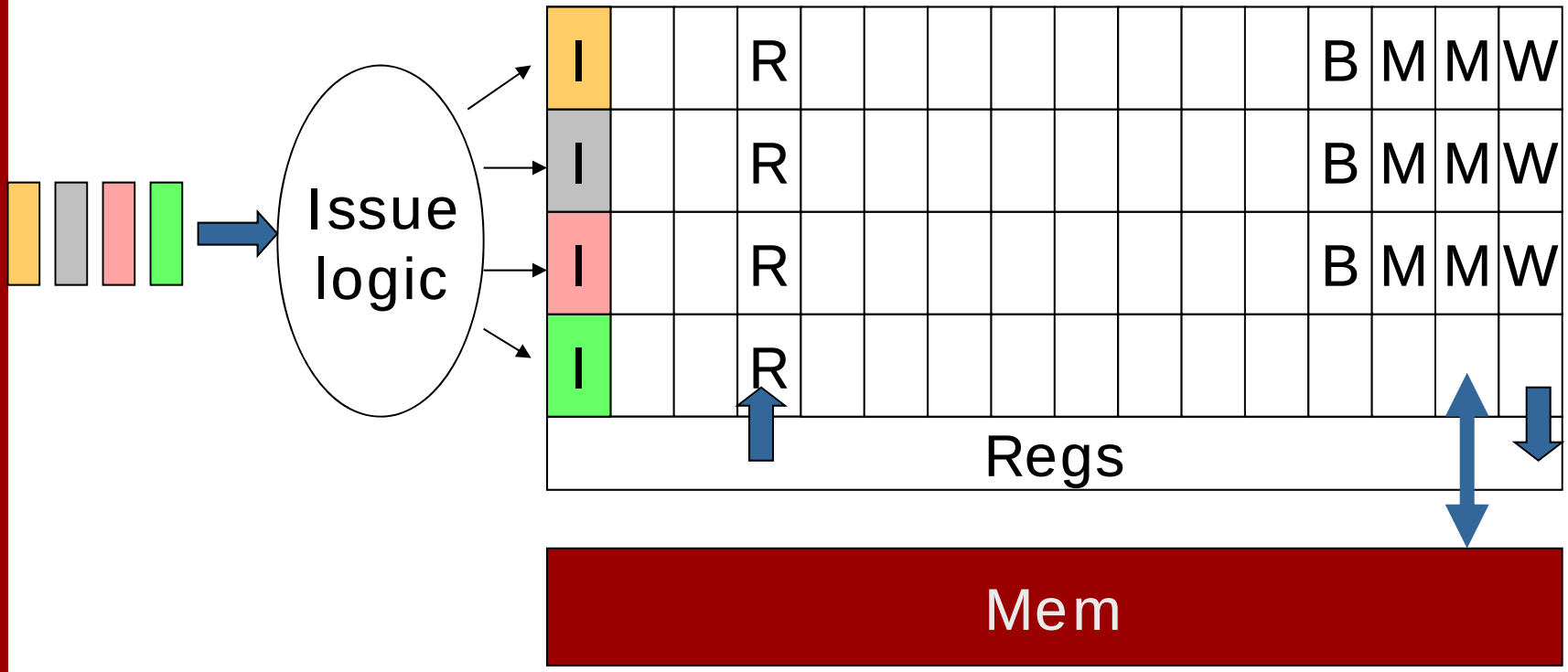


+Higher throughput

- More complicated architecture
- Branch delay more expensive (more instr. missed)
- Harder to find "enough" independent instr. (need 8 instr. between write and use)

It is actually a lot worse!

Modern CPUs: ~10-20 stages/pipe



+Shorter cycletime (higher GHz) ➔ deeper pipelines

- Branch delay even more expensive

- Even harder to find ”enough” independent instr.

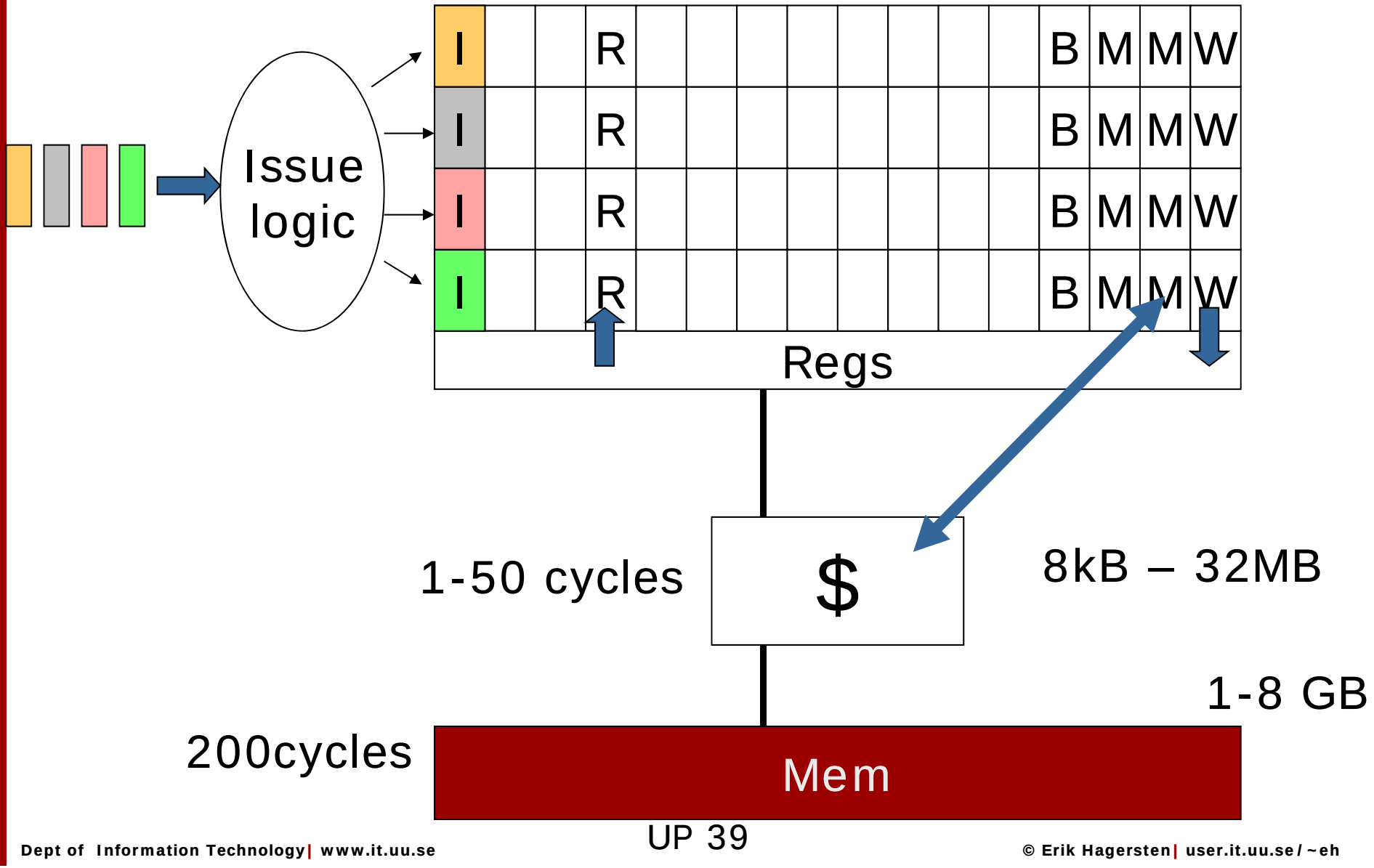
Caches and more caches
or
spam, spam and spam

Erik Hagersten

Uppsala University, Sweden

eh@it.uu.se

Pipeline problem3
Fix: Use a cache



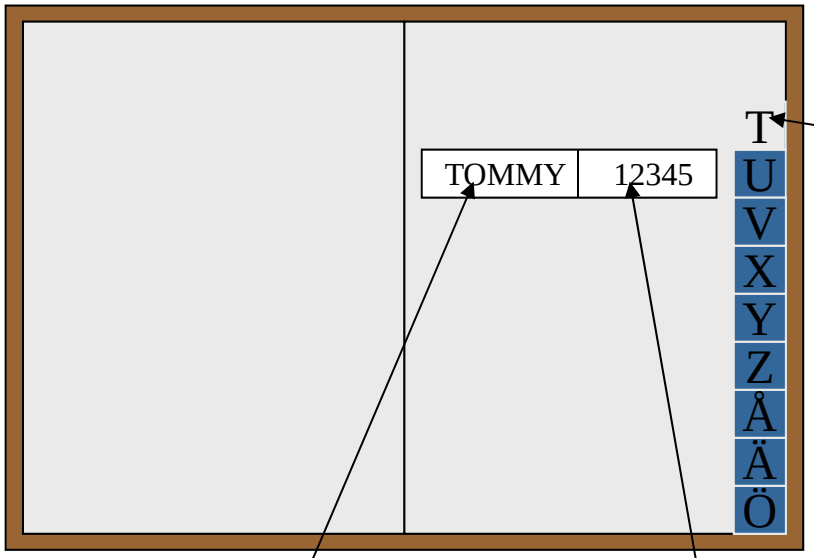
UPPSALA UNIVERSITET

Webster about “cache”

1. cache \'kash\ n [F, fr. cacher to press, hide, fr. (assumed) VL coacticare to press] together, fr. L coactare to compel, fr. coactus, pp. of cogere to compel - more at COGENT 1a: a hiding place esp. for concealing and preserving provisions or implements 1b: a secure place of storage 2: something hidden or stored in a cache

Address Book Cache

Looking for Tommy's Telephone Number



Indexing function

"Address Tag"

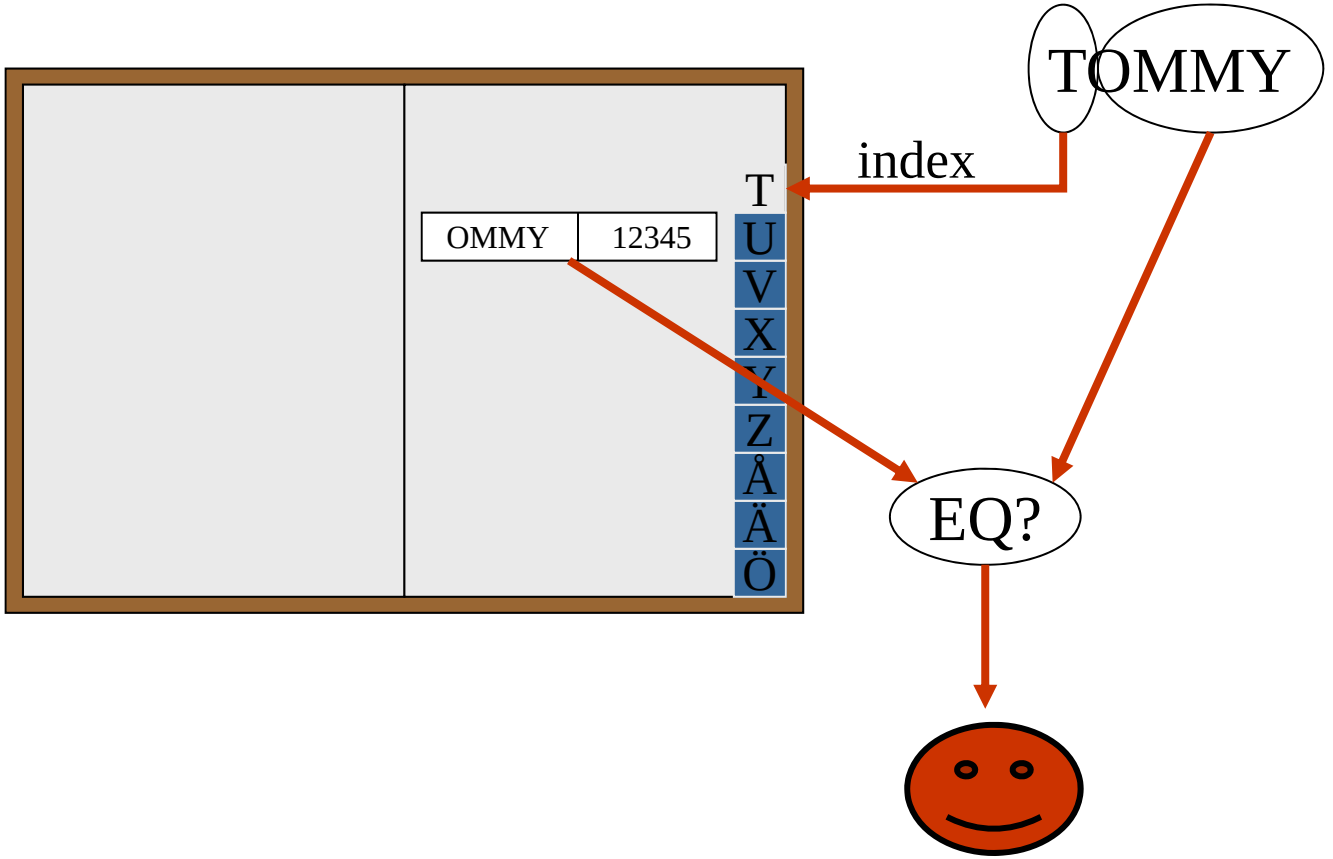
"Data"

One entry per page =>
Direct-mapped caches with 28 entries

© Erik Hagersten| user.it.uu.se/~eh

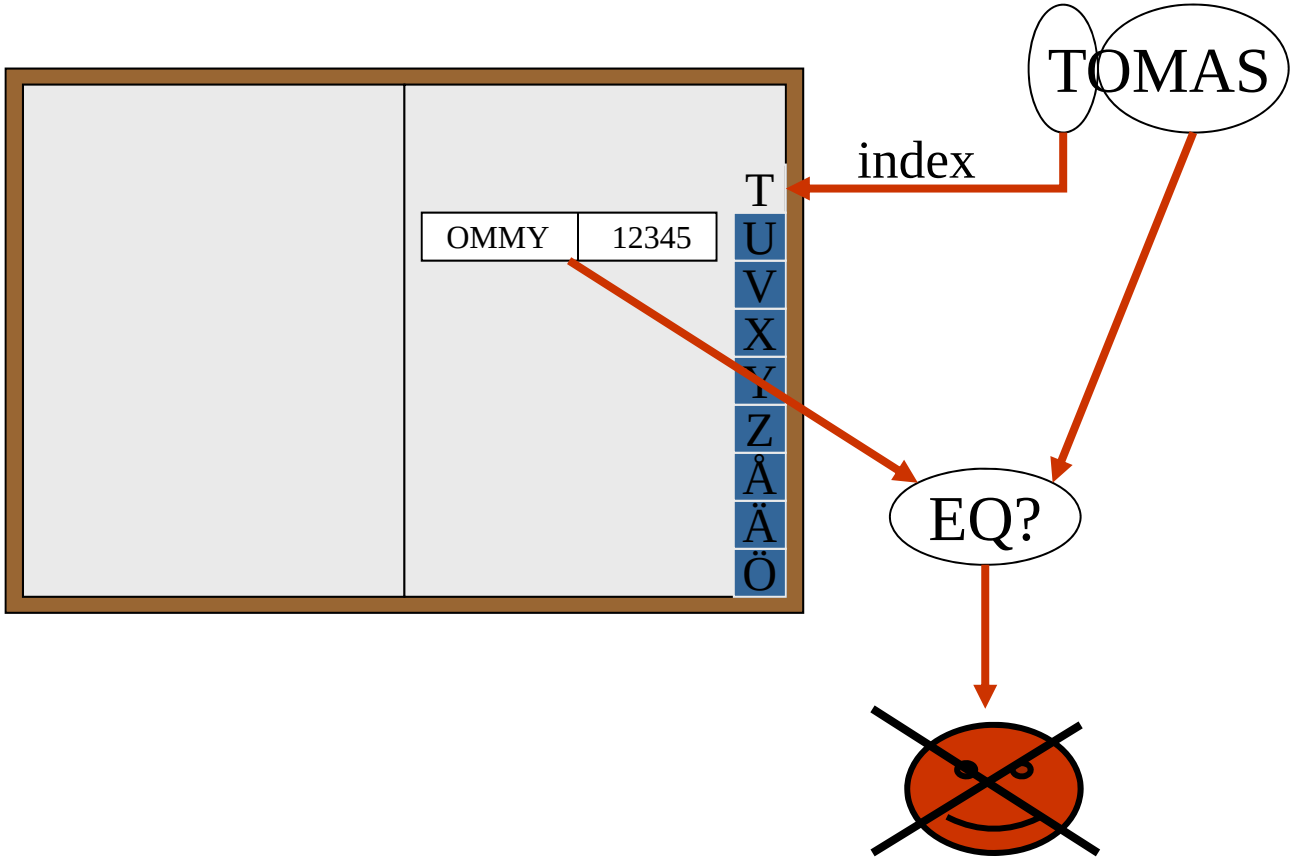
Address Book Cache

Looking for Tommy's Number



Address Book Cache

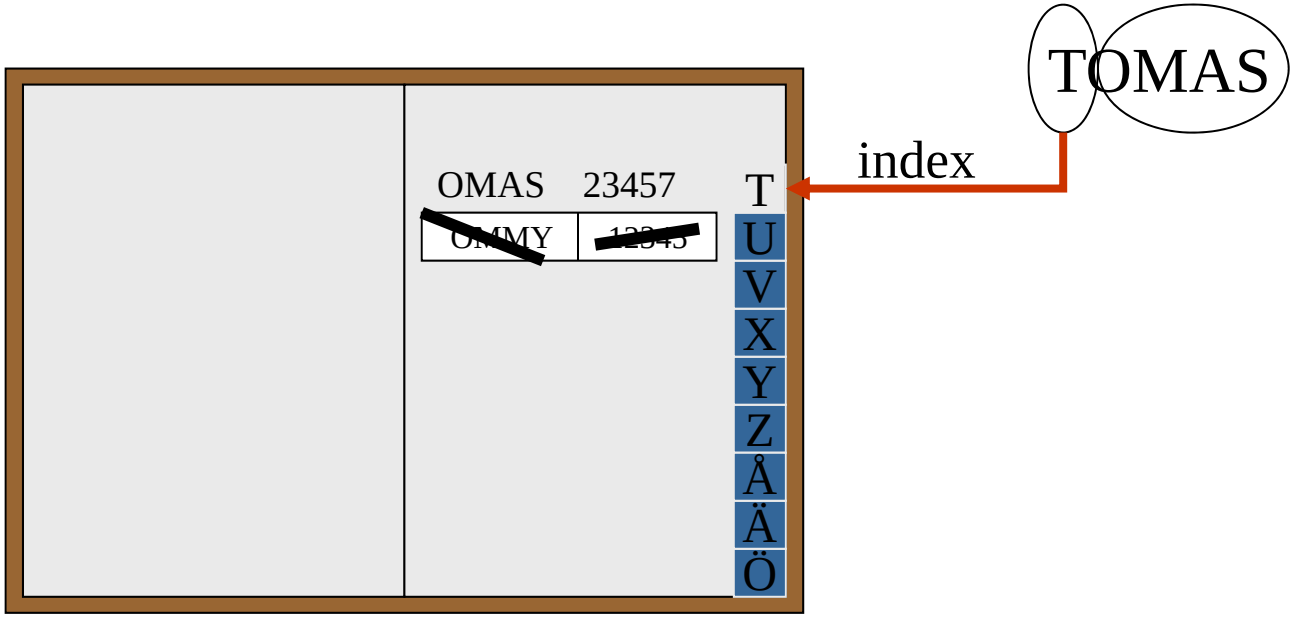
Looking for Tomas' Number



Miss!
Lookup Tomas' number in the telephone directory

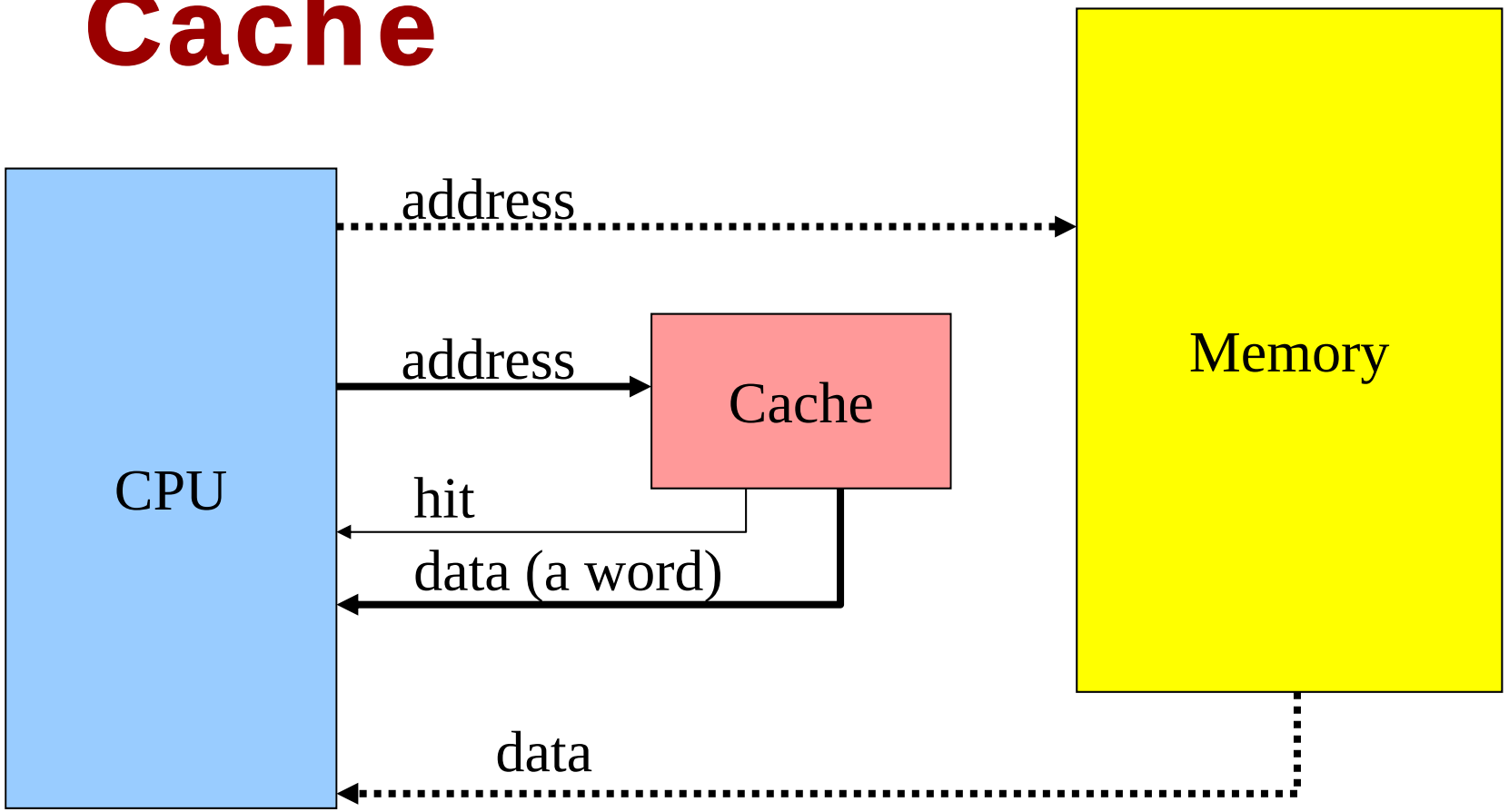
Address Book Cache

Looking for Tomas' Number

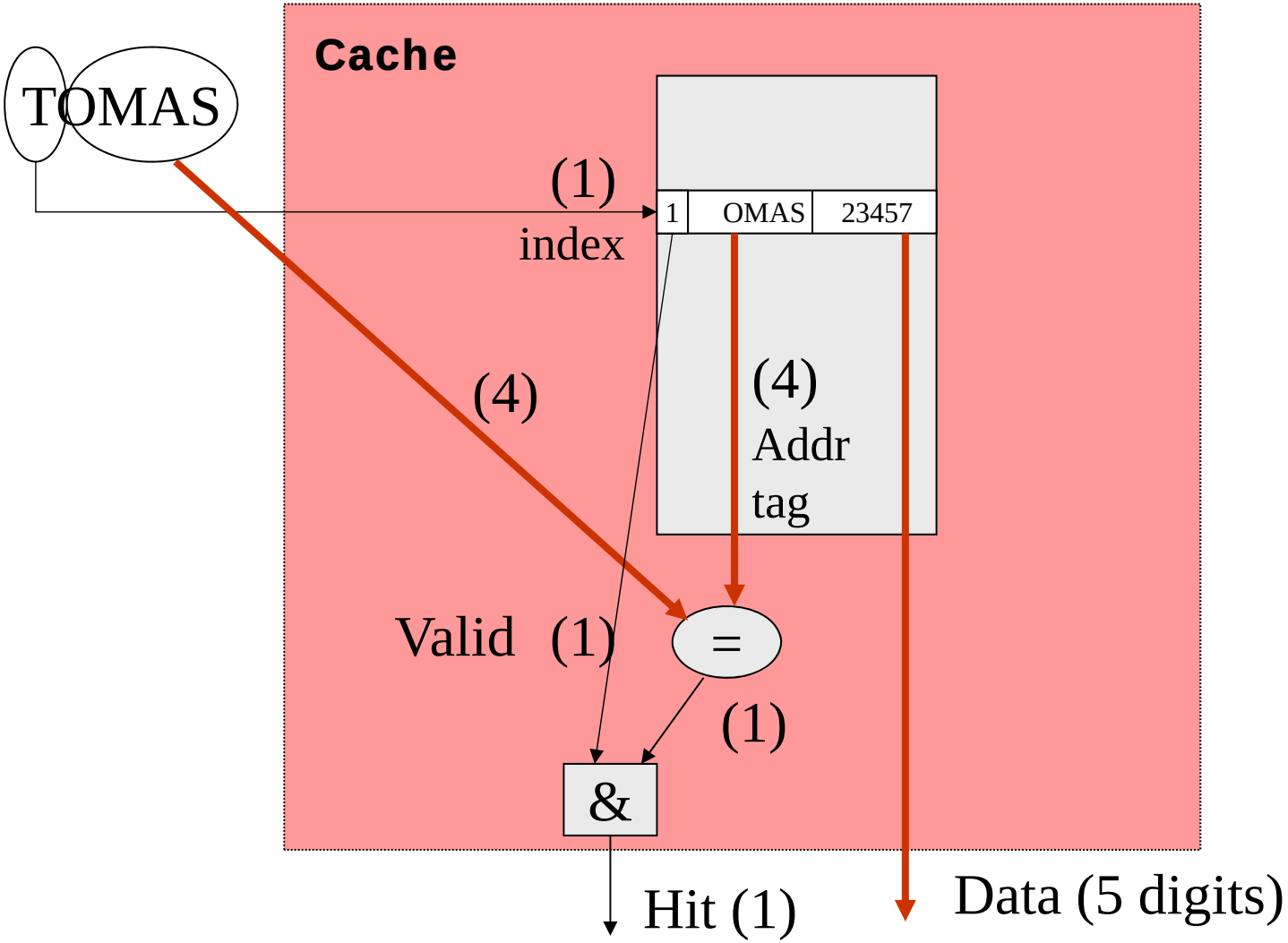


Replace TOMMY's data with TOMAS' data.
(Only one person per page = direct mapped cache)

Cache

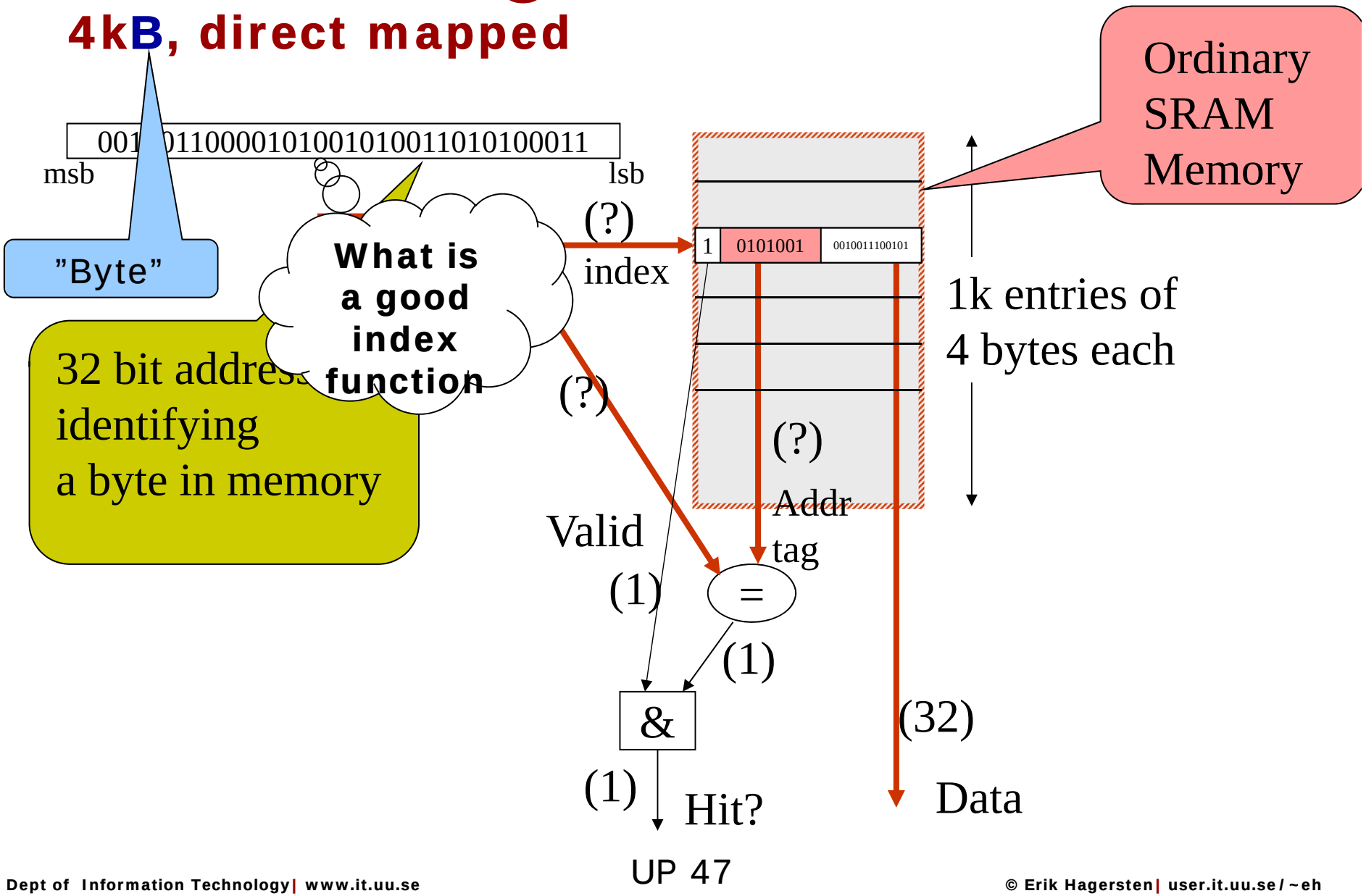


Cache Organization



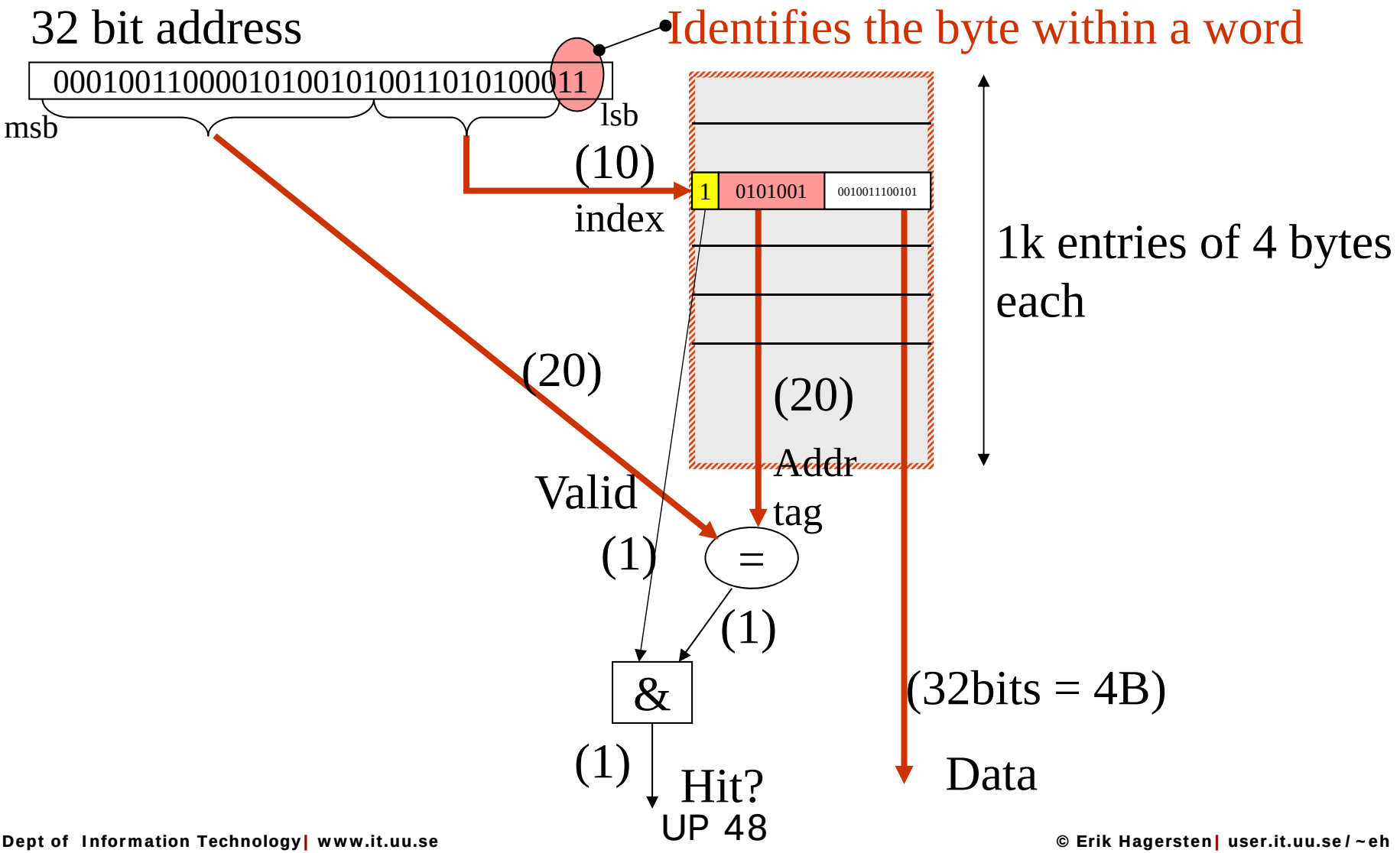
Cache Organization (really)

4kB, direct mapped

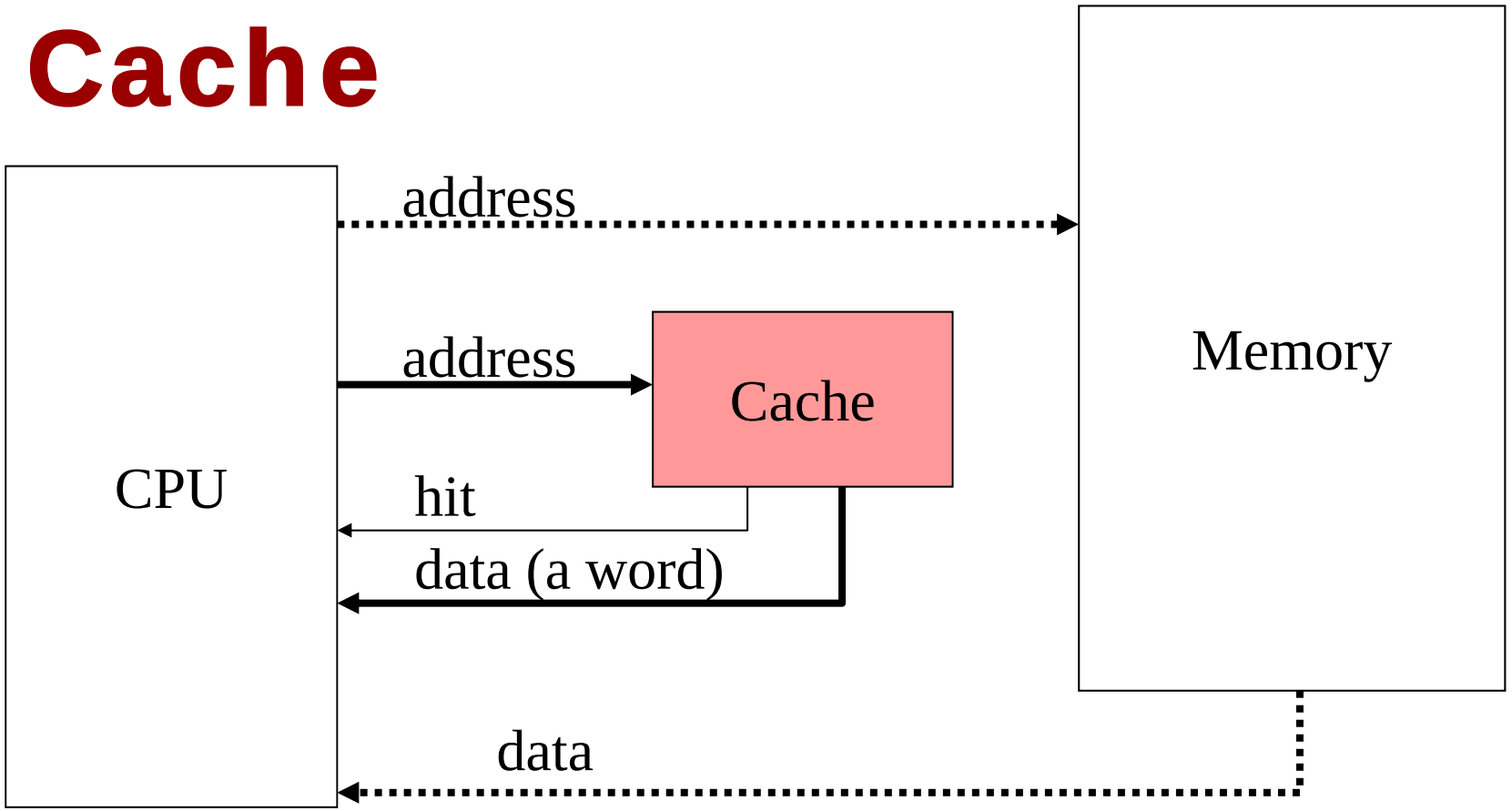


© Erik Hagersten| user.it.uu.se/~eh

Cache Organization 4kB, direct mapped



Cache



Hit: Use the data provided by the cache
~Hit: Use data from memory and also store it in the cache

UPPSALA UNIVERSITET

Why do you miss in a cache

- Mark Hill's three "Cs"
  - Compulsory miss (touching data for the first time)
  - Capacity miss (the cache is too small)
  - Conflict misses (non-optimal cache implementation)
- (Multiprocessors)
  - Communication (imposed by coherence)
  - False sharing (side-effect from large cache blocks)

© Erik Hagersten| user.it.uu.se/~eh

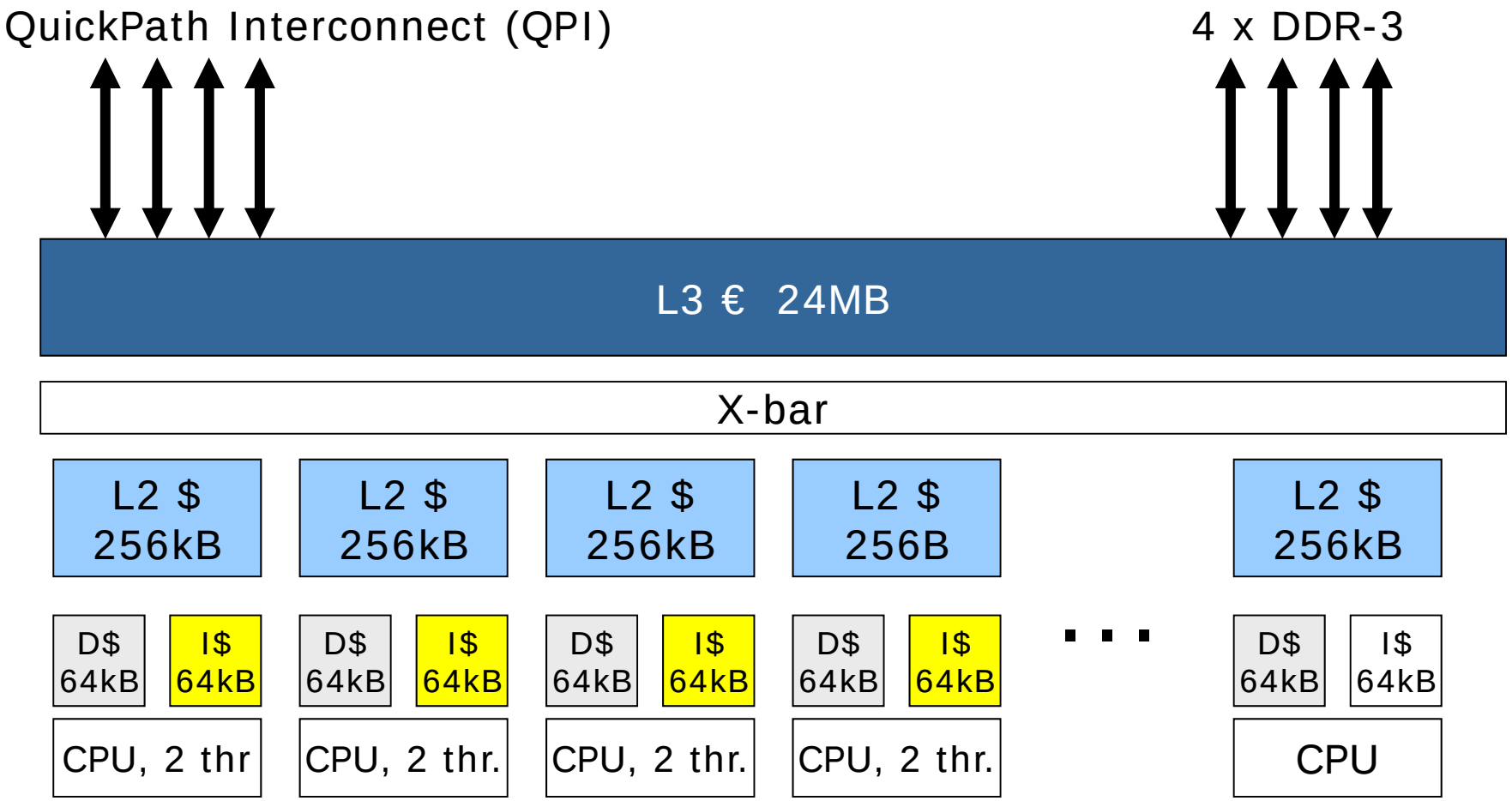
UPPSALA UNIVERSITET

How to get more effective caches?

- Larger cache (more capacity)
- Cache block size (larger cache lines)
- More placement choice (more associativity)
- Innovative caches (victim, skewed, …)
- Cache hierarchies (L1, L2, L3, CMR)
- Latency-hiding (weaker memory models)
- Latency-avoiding (prefetching)
- Cache avoiding (cache bypass)

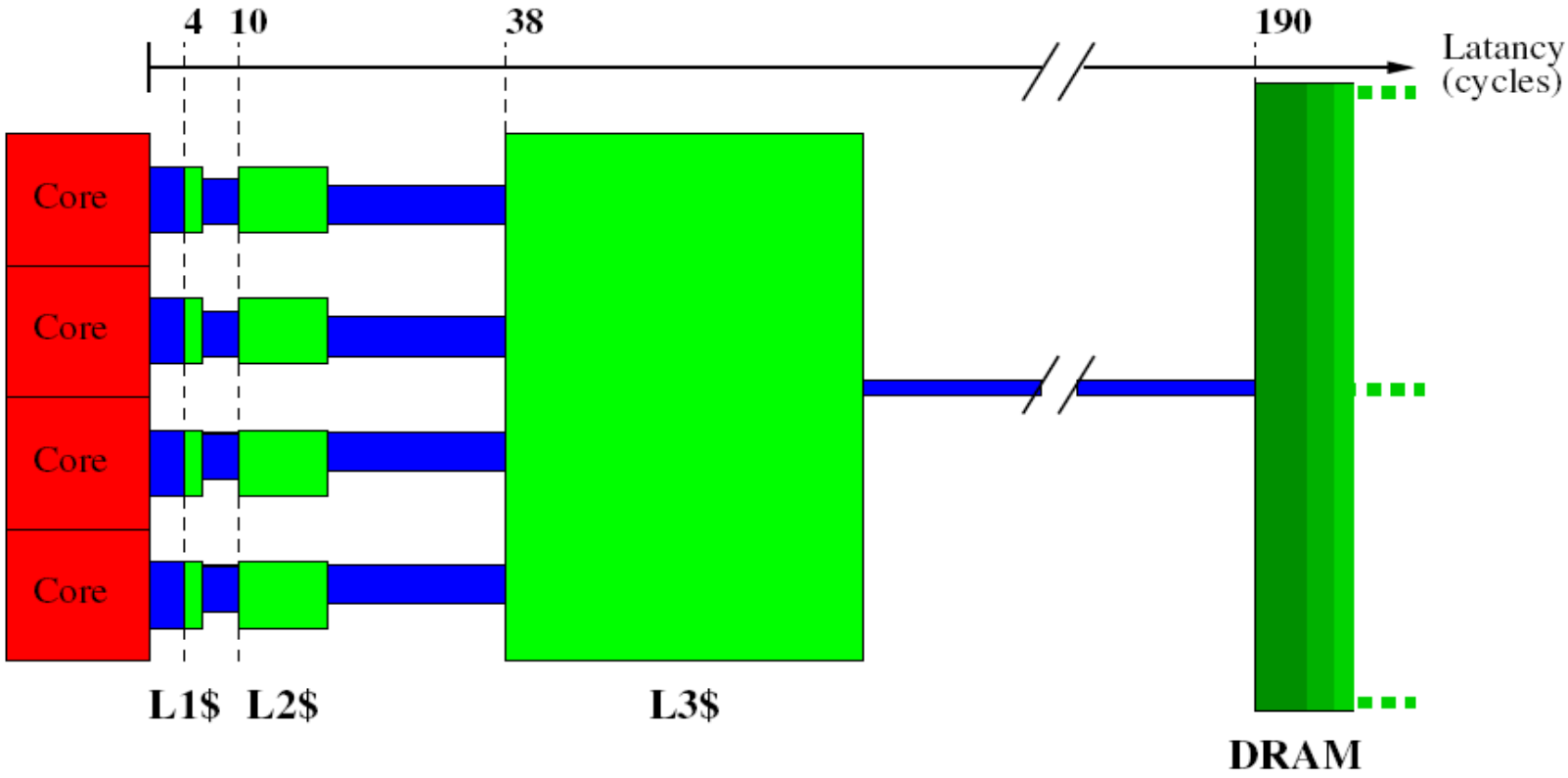
© Erik Hagersten| user.it.uu.se/~eh

Intel: ”Nehalem-Ex” (i7)



8 cores x 2 threads

Cache Capacity / Latency / BW



Cache implementation

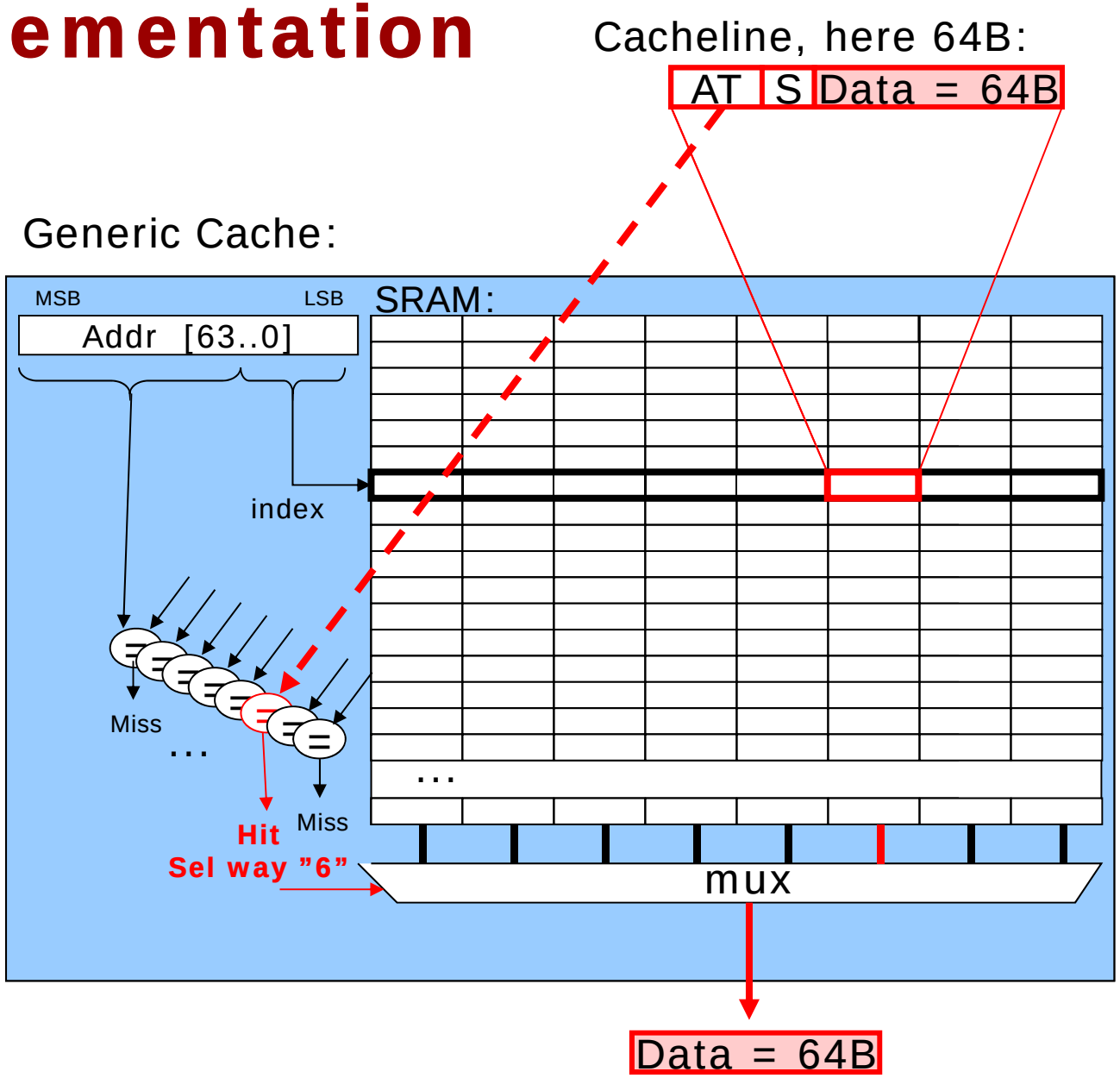
Caches at all level roughly work like this:

L3 € 24MB

L2 $ 256kB

D1 ¢ 64kB

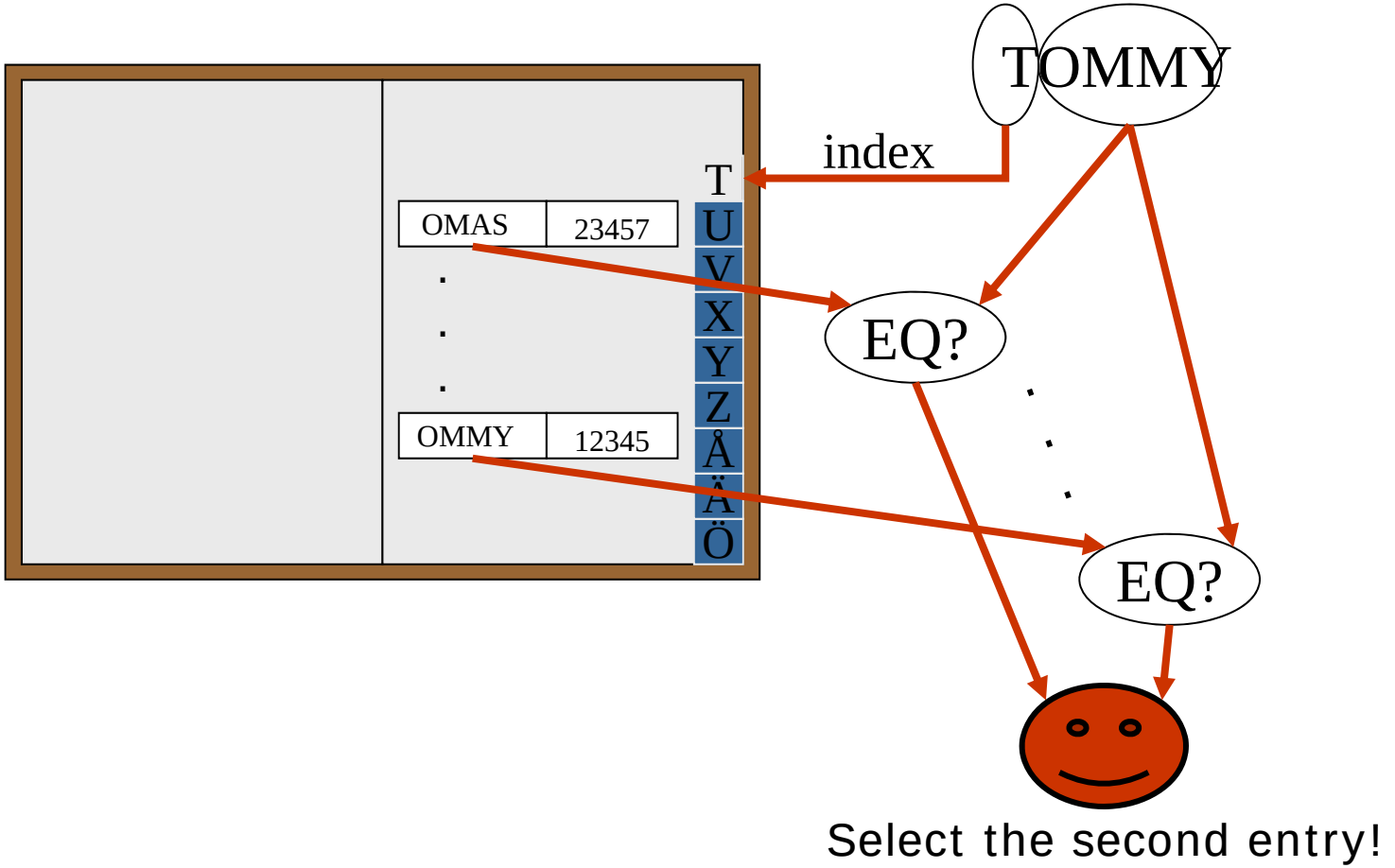
I1 ¢ 64kB



© Erik Hagersten| user.it.uu.se/~eh

Address Book Analogy

Eight names per page: index first, then search.

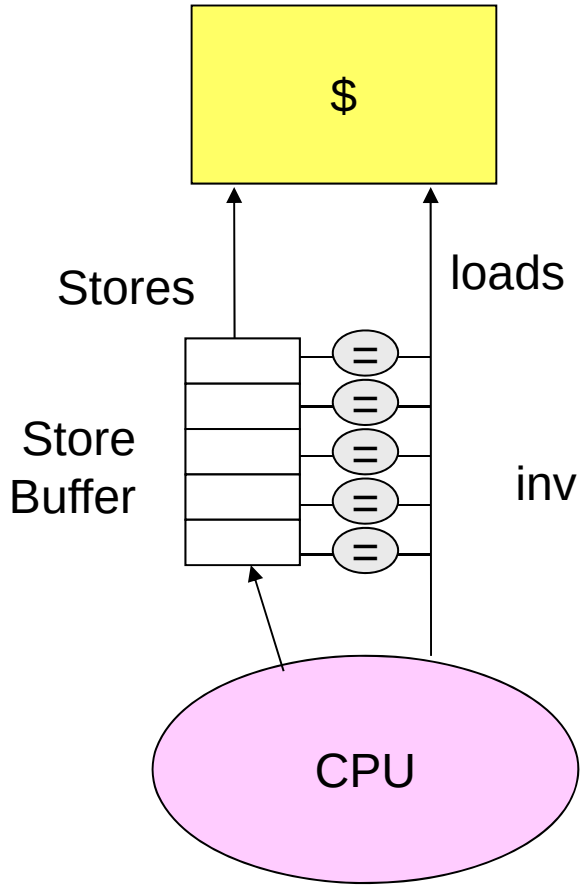


UPPSALA UNIVERSITET

Who to replace?
Picking a "victim"

- Least-recently used (LRU)
  - Considered the best algorithm
  - Only practical up to 4-way (16 bits/CL)
- Not most recently used
  - Remember who used it last: 8-way -> 3 bits/CL
- Pseudo-LRU
  - Course Time stamps, used in the VM system
- Random replacement
  - Can't continuously to have "bad luck...

Store buffer: LD bypass ST



➔Stores are moved off the critical path

UPPSALA UNIVERSITET

Cache lingo

Cacheline: Data chunk move to/from a cache

Cache set: Fraction of the cache identified by the index

Associativity: Number of alternative storage places for a cacheline

Replacement policy: picking the victim to throw out from a set (LRU/Random/Nehalem)

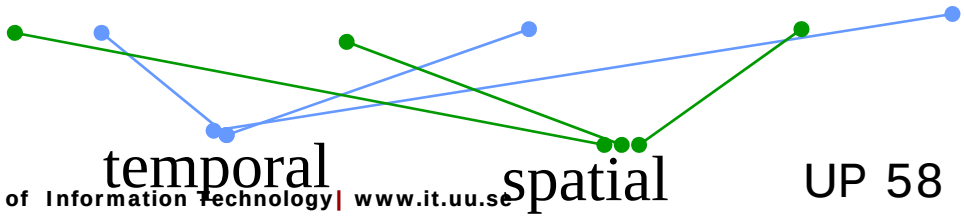
Temporal locality: Likelihood to access the same data again soon

Spatial locality: Likelihood to access nearby data again soon

Typical access pattern:

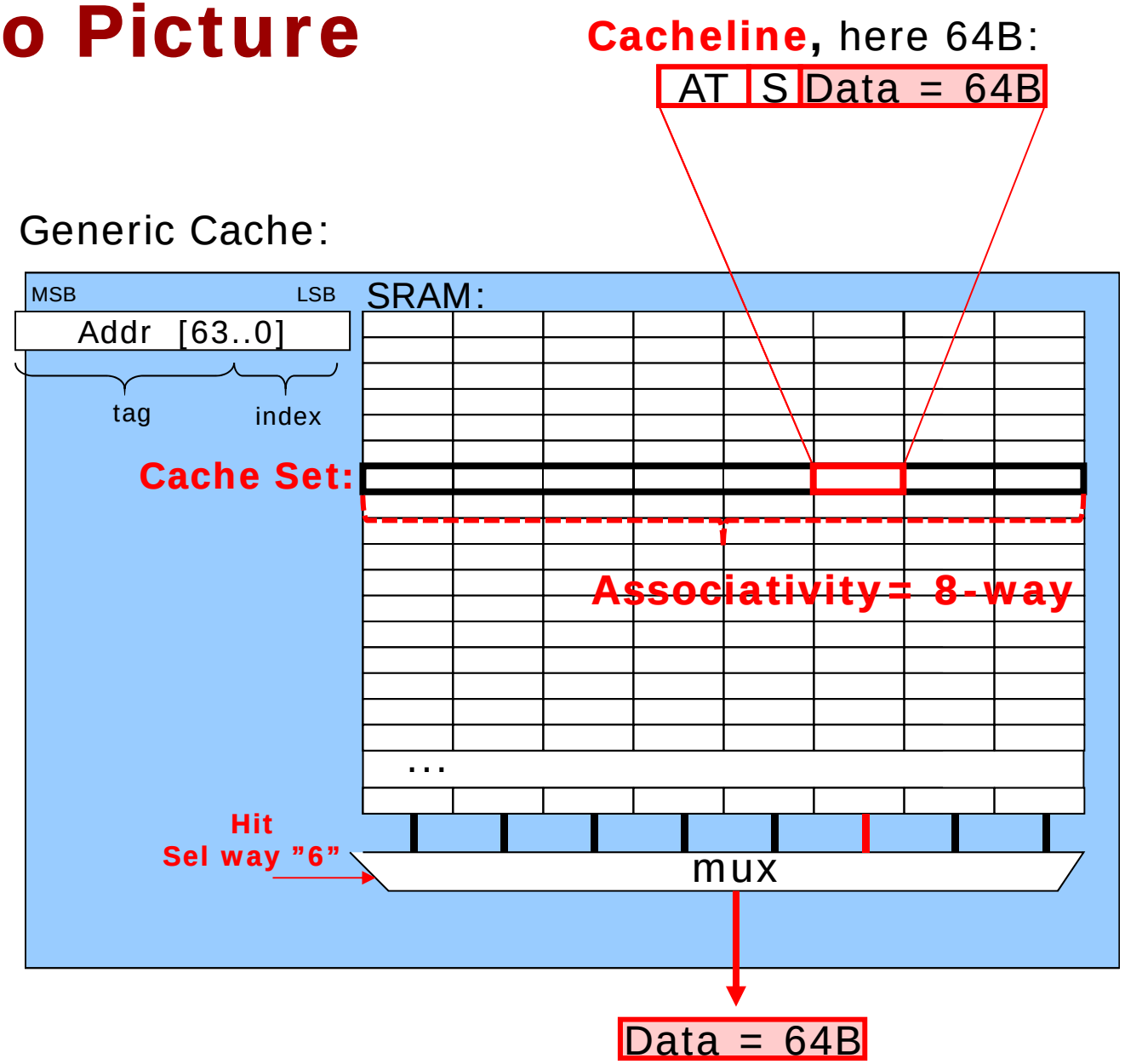
(inner loop stepping through an array)

A, B, C, A+4, B, C, A+8, B, C, ...

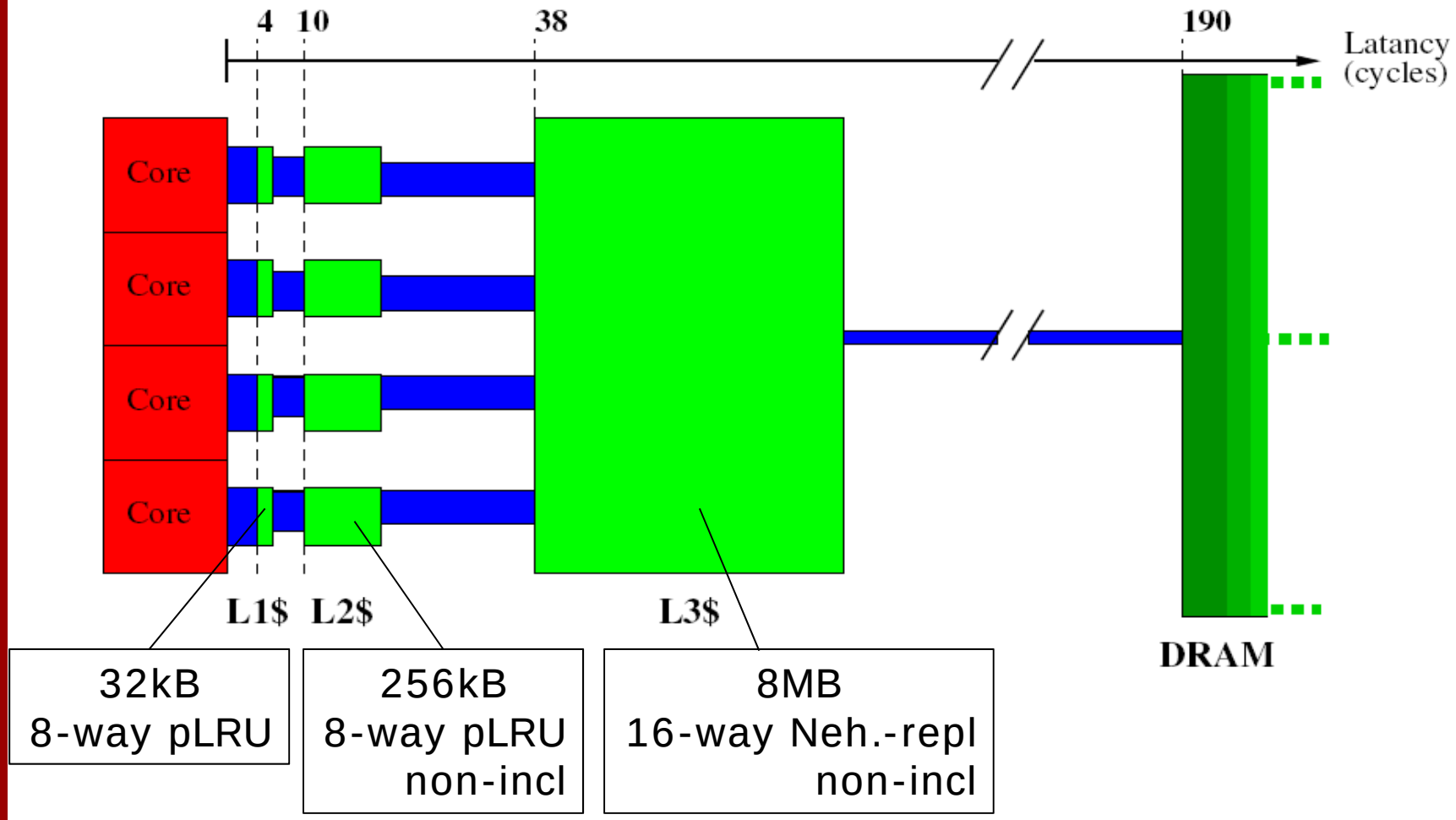


© Erik Hagersten| user.it.uu.se/~eh

Cache Lingo Picture

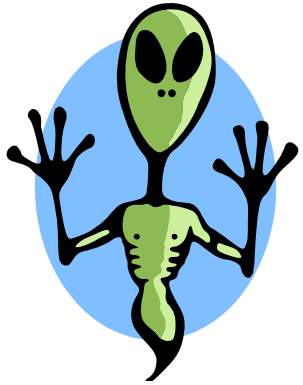


Exempel Nehalem i7 (one example)



© Erik Hagersten| user.it.uu.se/~eh

HW prefetching



...a little green man that anticipates your next memory access and prefetches the data to the cache.

Improves MLP!

- **Sequential prefetching:** Sequential streams [to a page]. Some number of prefetch streams supported. Often only for L2 and L3.
- **PC-based prefetching:** Detects strides from the same PC. Most often for L1.
- **Adjacent prefetching:** On a miss, also bring in the "neighboring" cache line. Often only for L2 and L3.

UPPSALA UNIVERSITET

Take-away message: Caches

- Cache are fast but small
- Cache data travels in cache-line chunks (~64bytes)
- LSB part of the address is used to find the ”set” (aka, indexing)
- There is a limited number of cache lines per set = ”associativity” (#names on addr book page)
- Typically, several levels of caches

- Caches are most important target for optimizations

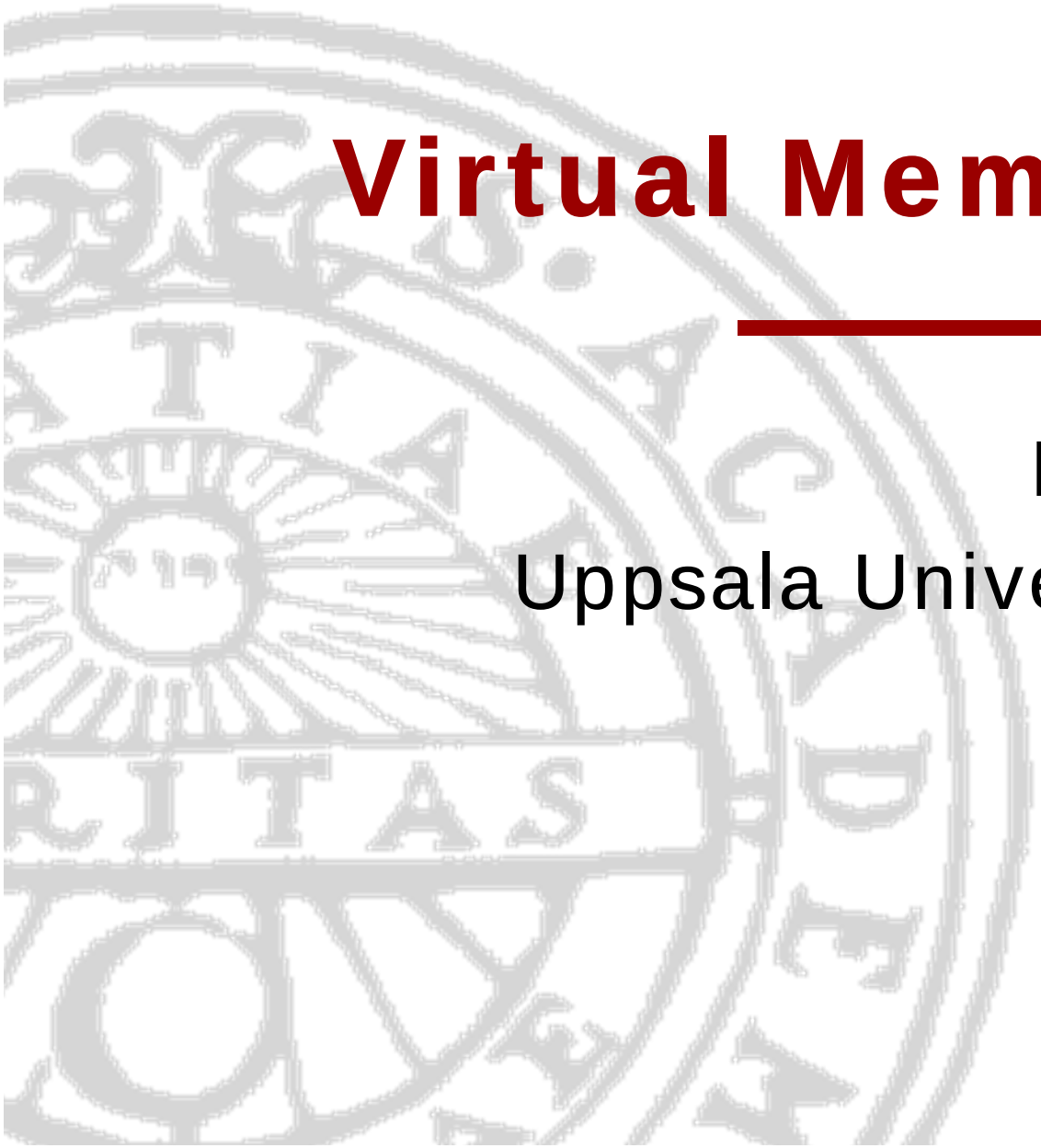
© Erik Hagersten| user.it.uu.se/~eh

UPPSALA UNIVERSITET

How are we doing?

- Create and explore locality:
  a) Spatial locality
  b) Temporal locality
  c) Geographical locality
- Create and explore parallelism
  a) Instruction level parallelism (ILP)
  b) Thread level parallelism (TLP)
  c) Memory level parallelism (MLP)

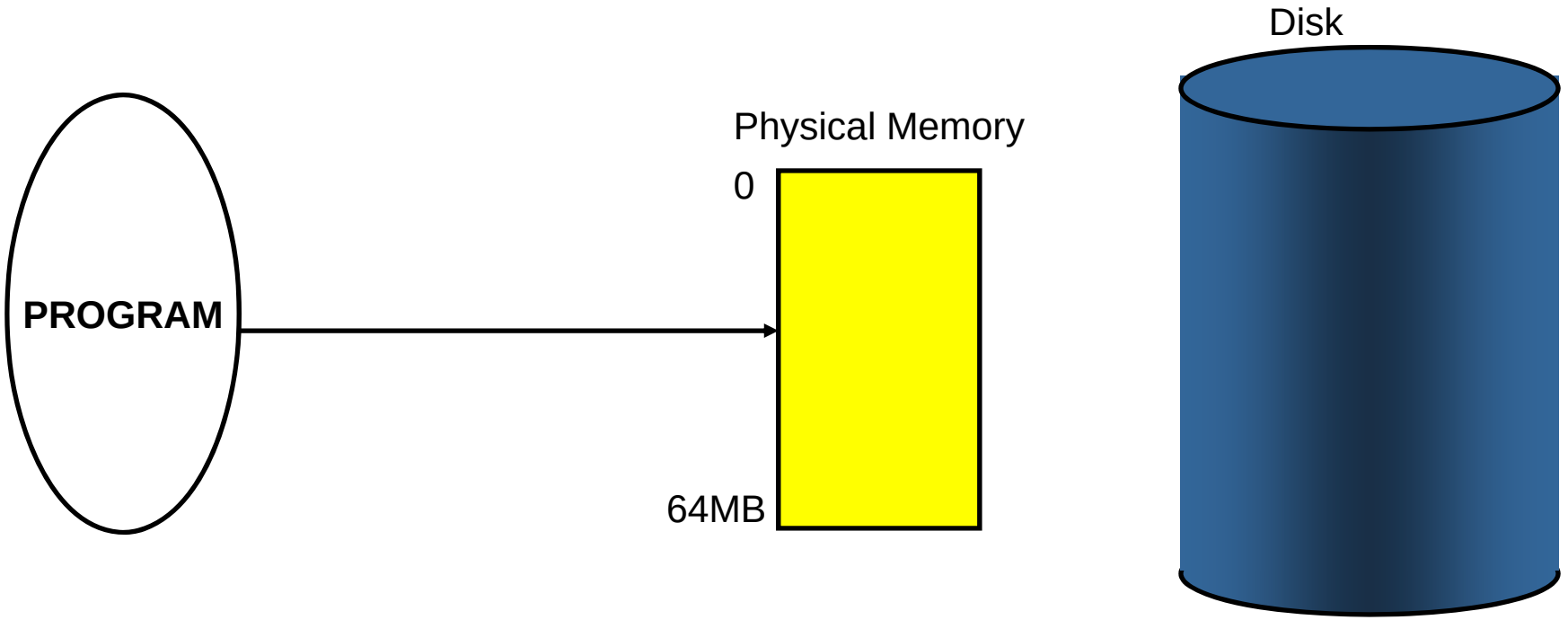
© Erik Hagersten| user.it.uu.se/~eh



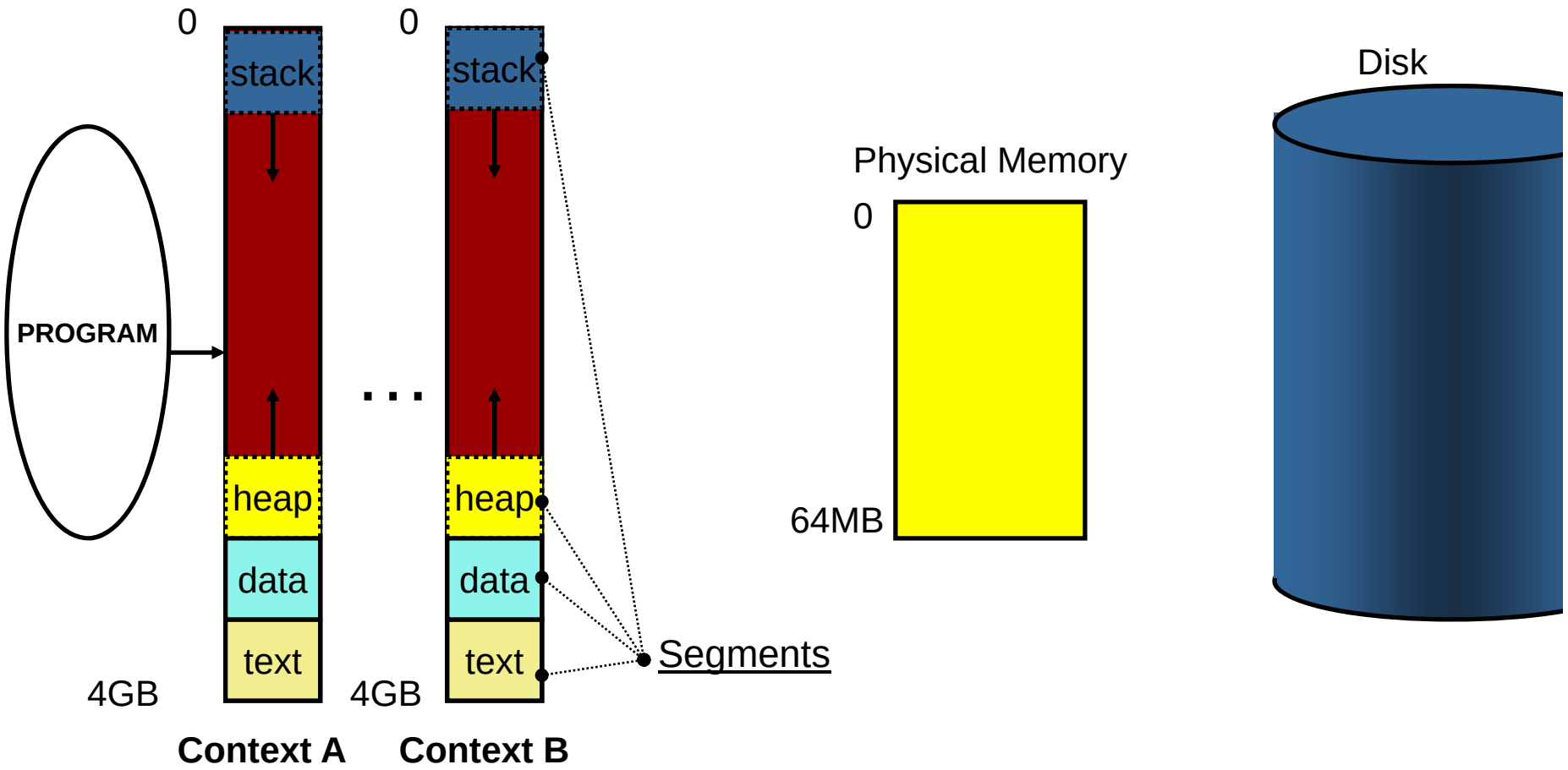
Virtual Memory System

Erik Hagersten
Uppsala University, Sweden
eh@it.uu.se

Physical Memory

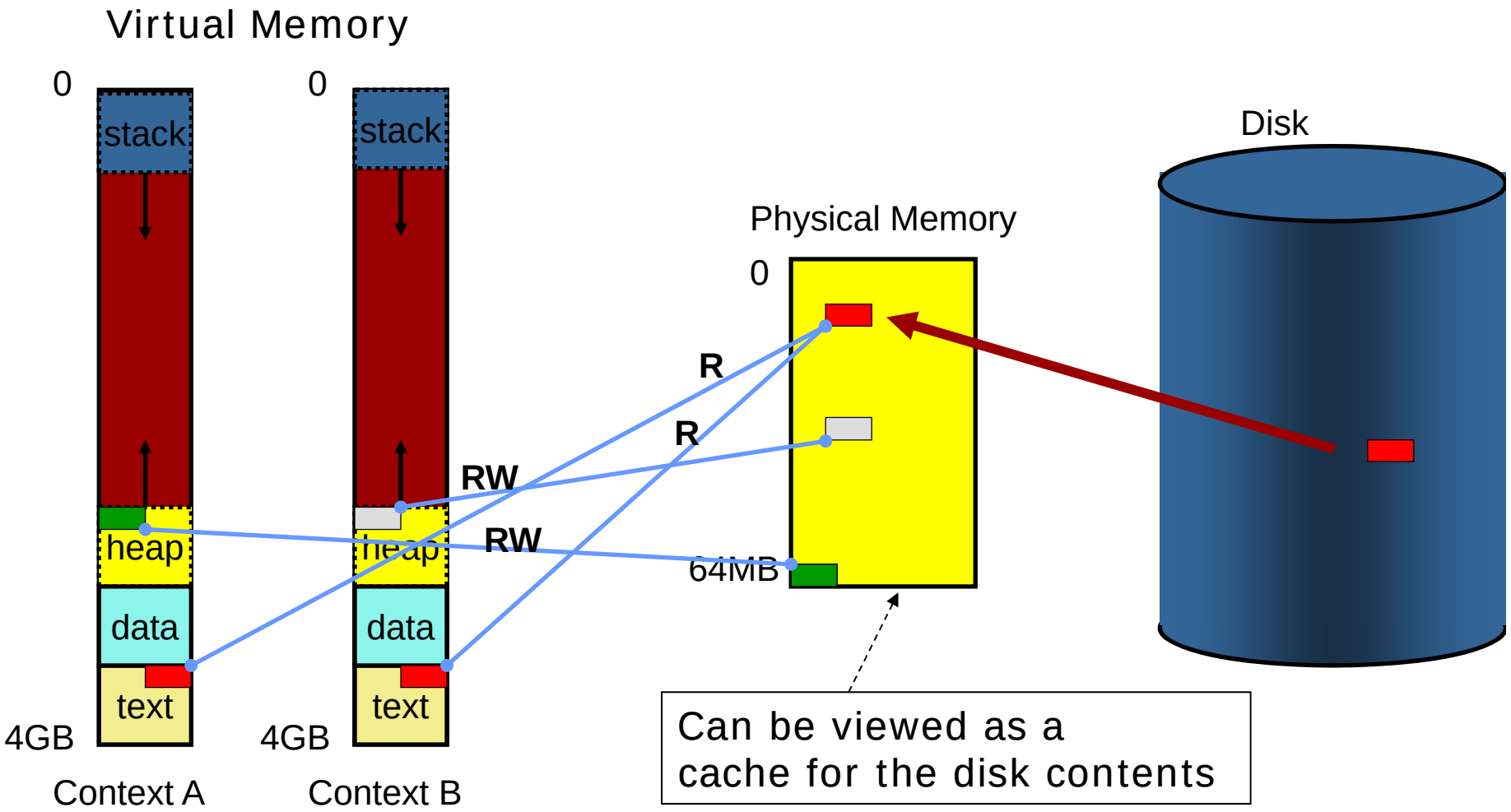


Virtual and Physical Memory



© Erik Hagersten| user.it.uu.se/~eh

Translation & Protection

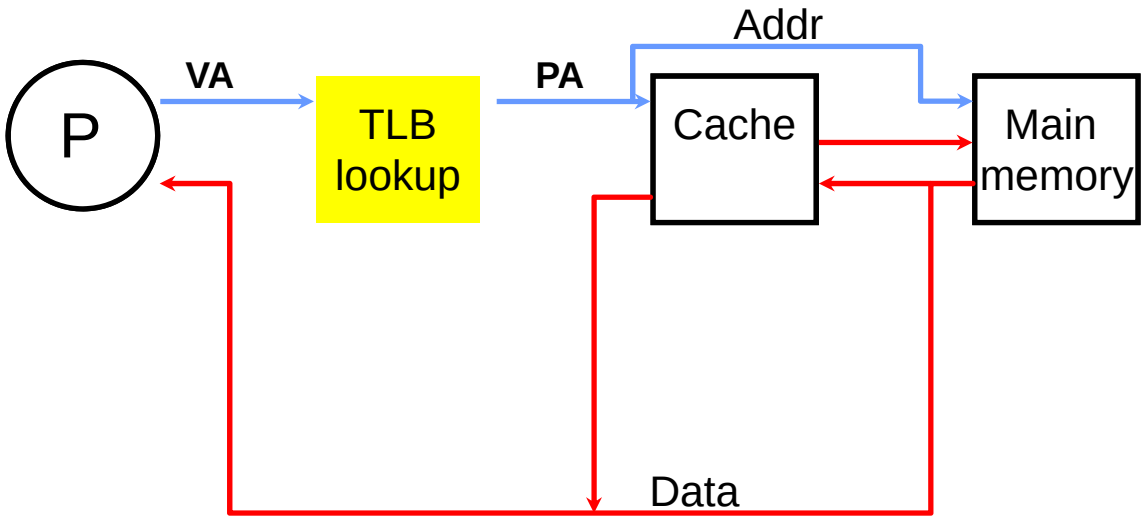


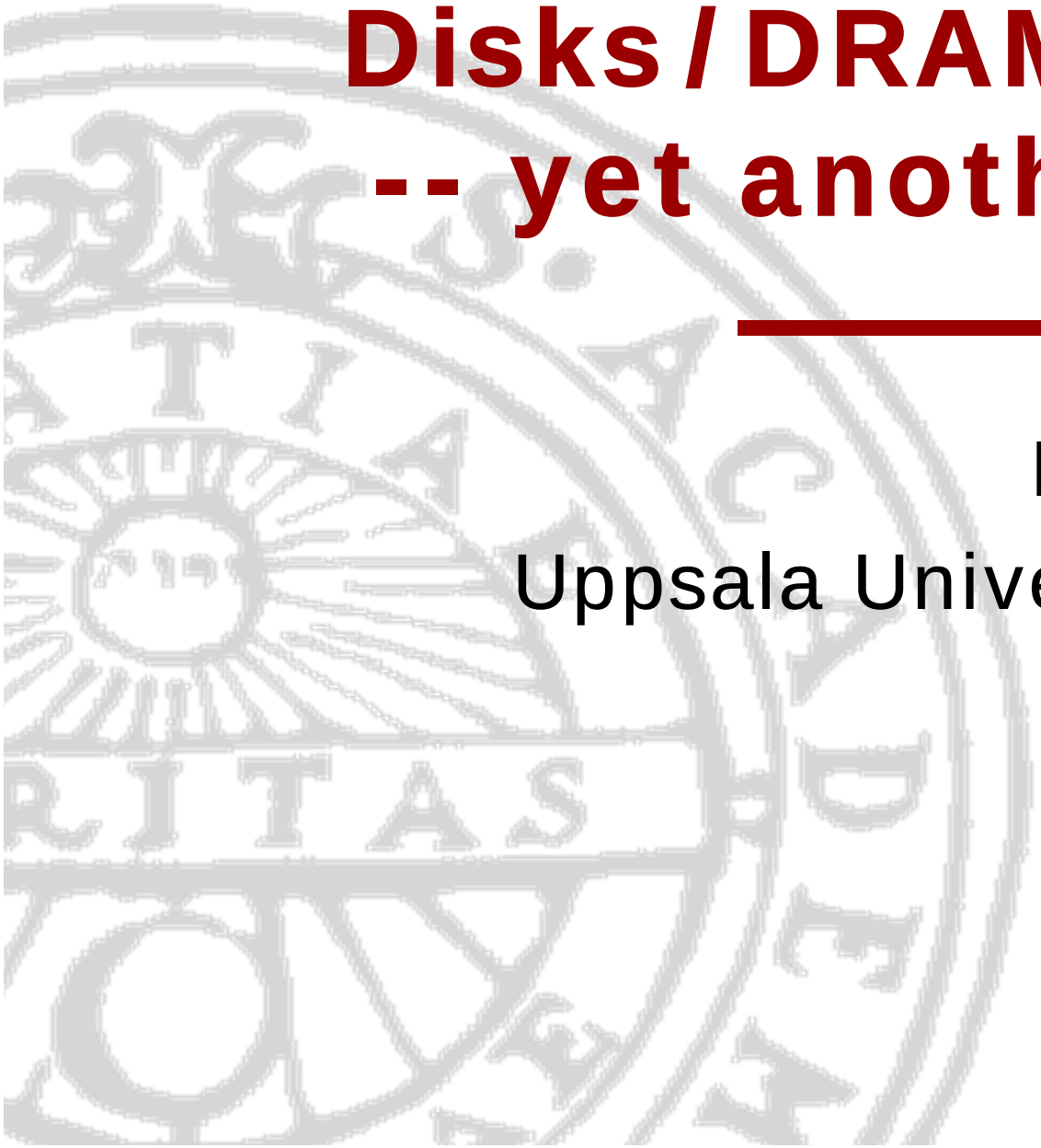
Fast address translation

How to quickly and cheaply find the right physical page in the [fully associativity cache called] physical memory?

- Store the most commonly used address translations in a **cache**—***Translation Look-aside Buffer*** (TLB)

  = = > *The caches rears their ugly faces again!*





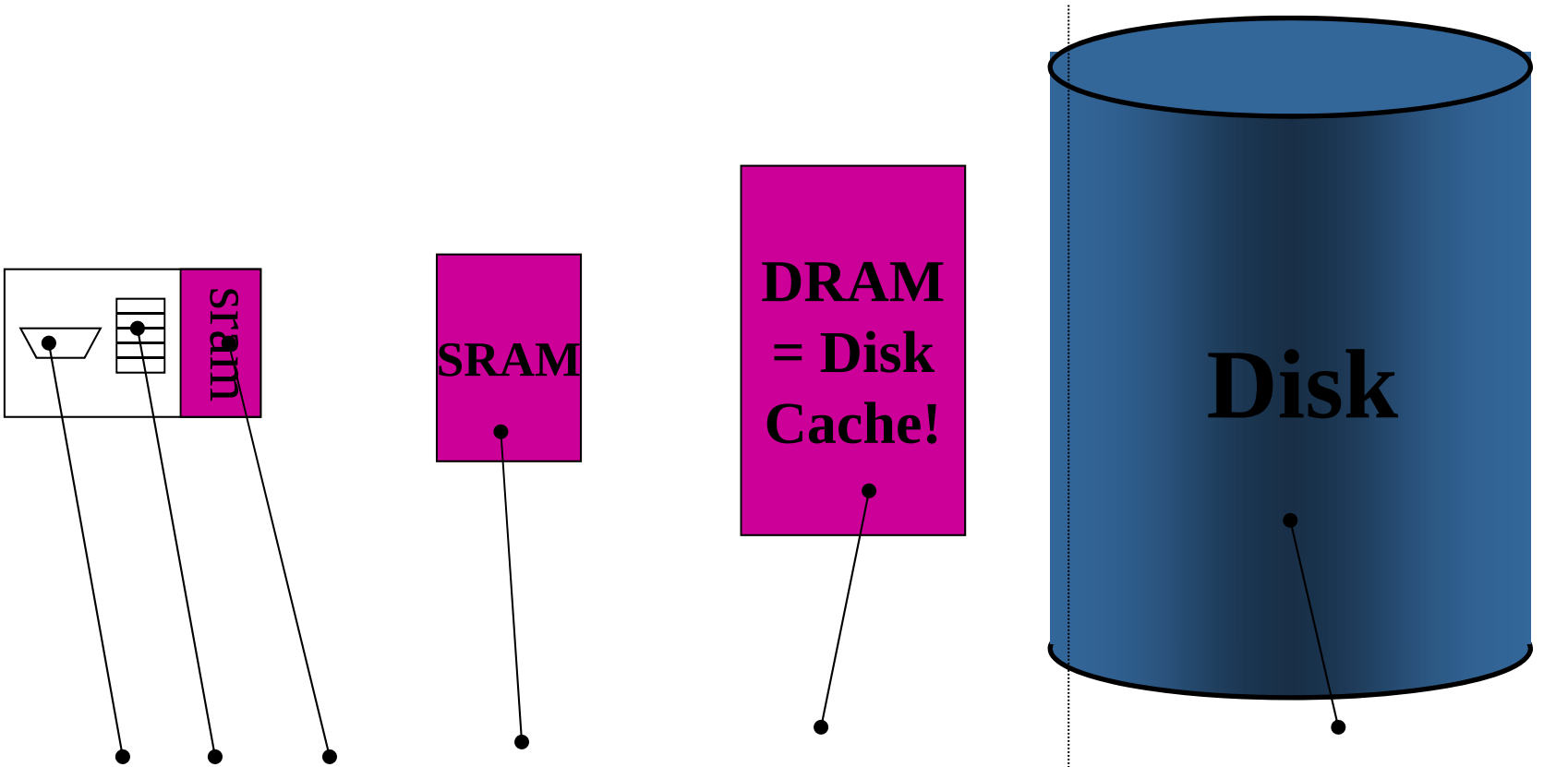
Disks/DRAM Memory -- yet another cache

Erik Hagersten

Uppsala University, Sweden

eh@it.uu.se

Memory/storage



2000: 1ns 1ns 3ns 10ns 150ns 5 000 000ns

1kB 64k 4MB 1GB 1 TB

UPPSALA UNIVERSITET

VM dictionary

Virtual Memory System	The “cache” languge
Virtual address	~Cache address
Physical address	~Cache location, “way info”
Page	~Huge cache block
Page fault	~Extremely painfull $miss
Page-fault handler	~The software filling the $
Page-out	Write-back if dirty
Any physical page frame can map any virtual page	Fully associative cache

UPPSALA UNIVERSITET

Caches Everywhere…

- L1 D cache
- L1 I cache
- L2 cache
- L3 cache
- ITLB
- DTLB
- Virtual memory system
- Disks
- Branch predictors

UPPSALA UNIVERSITET

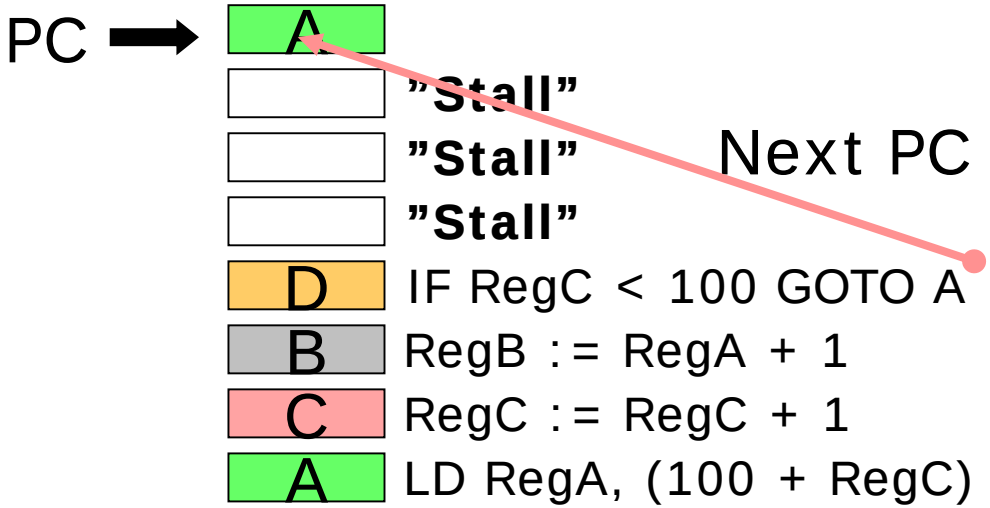
How are we doing?

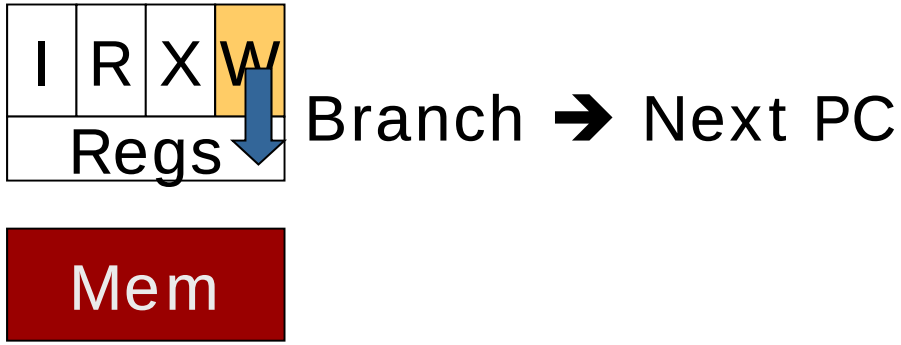
- Create and explore locality:
  a) Spatial locality
  b) Temporal locality
  c) Geographical locality
- Create and explore parallelism
  a) Instruction level parallelism (ILP)
  b) Thread level parallelism (TLP)
  c) Memory level parallelism (MLP)

Branch prediction

Erik Hagersten
Uppsala University, Sweden
eh@it.uu.se

Pipeline problem2: Branch delays ☹

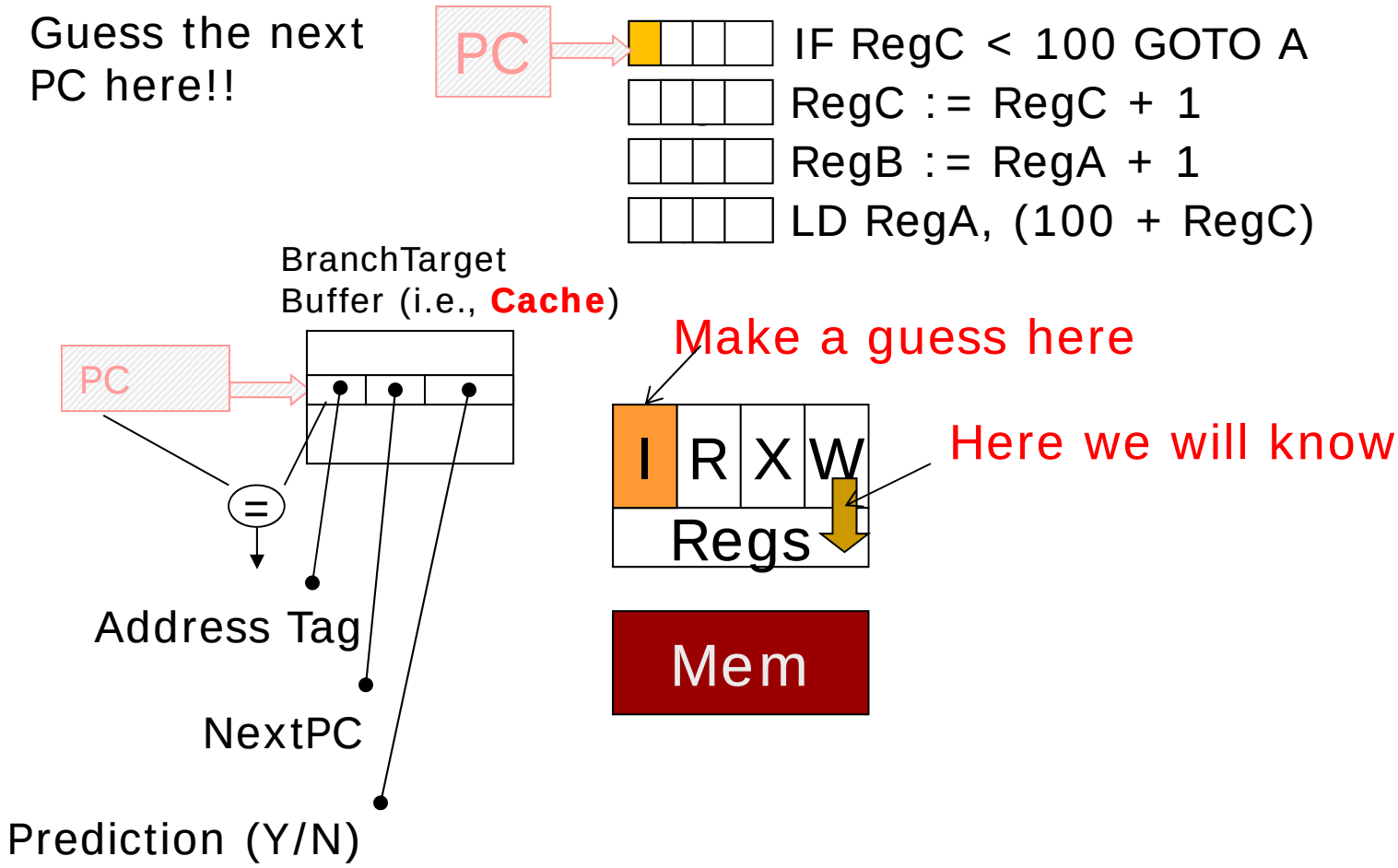




7 cycles per iteration of 4 instructions ☹

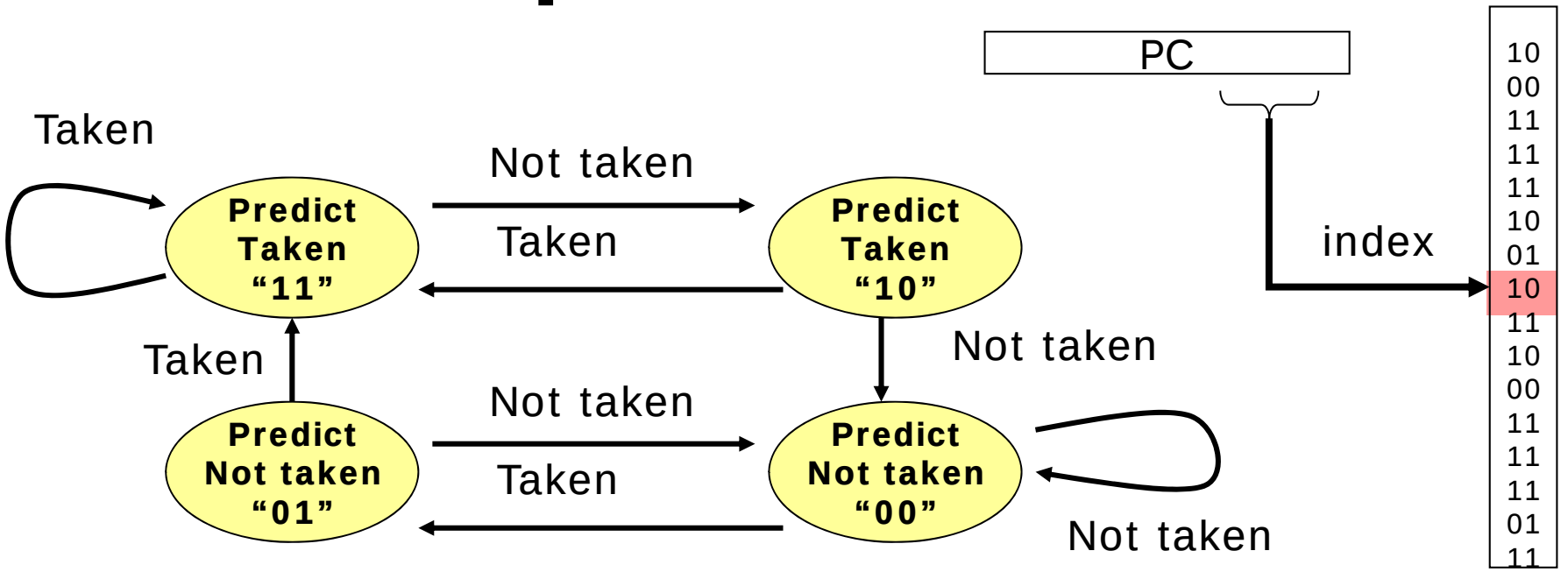
➔ Need longer basic blocks with many independent instr.

Branch Predictor Based on History



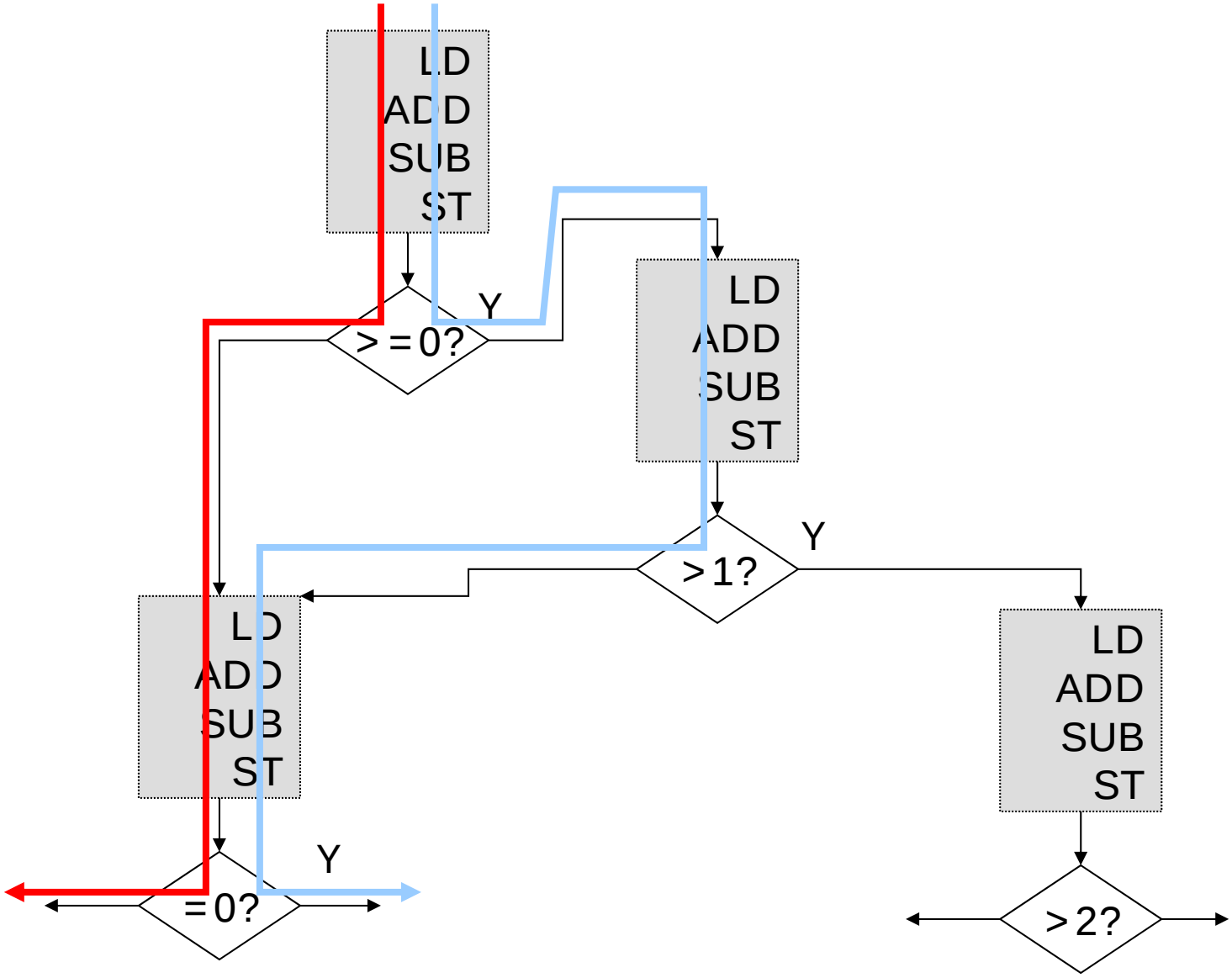
© Erik Hagersten| user.it.uu.se/~eh

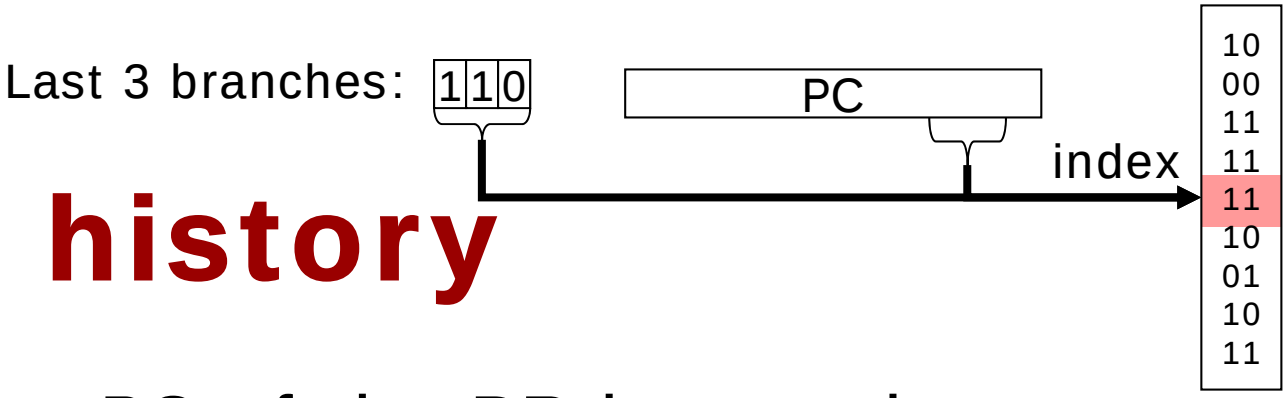
A two-bit prediction scheme



- Requires prediction to miss twice in order to change prediction => better performance

Dynamic Scheduling Of Branches



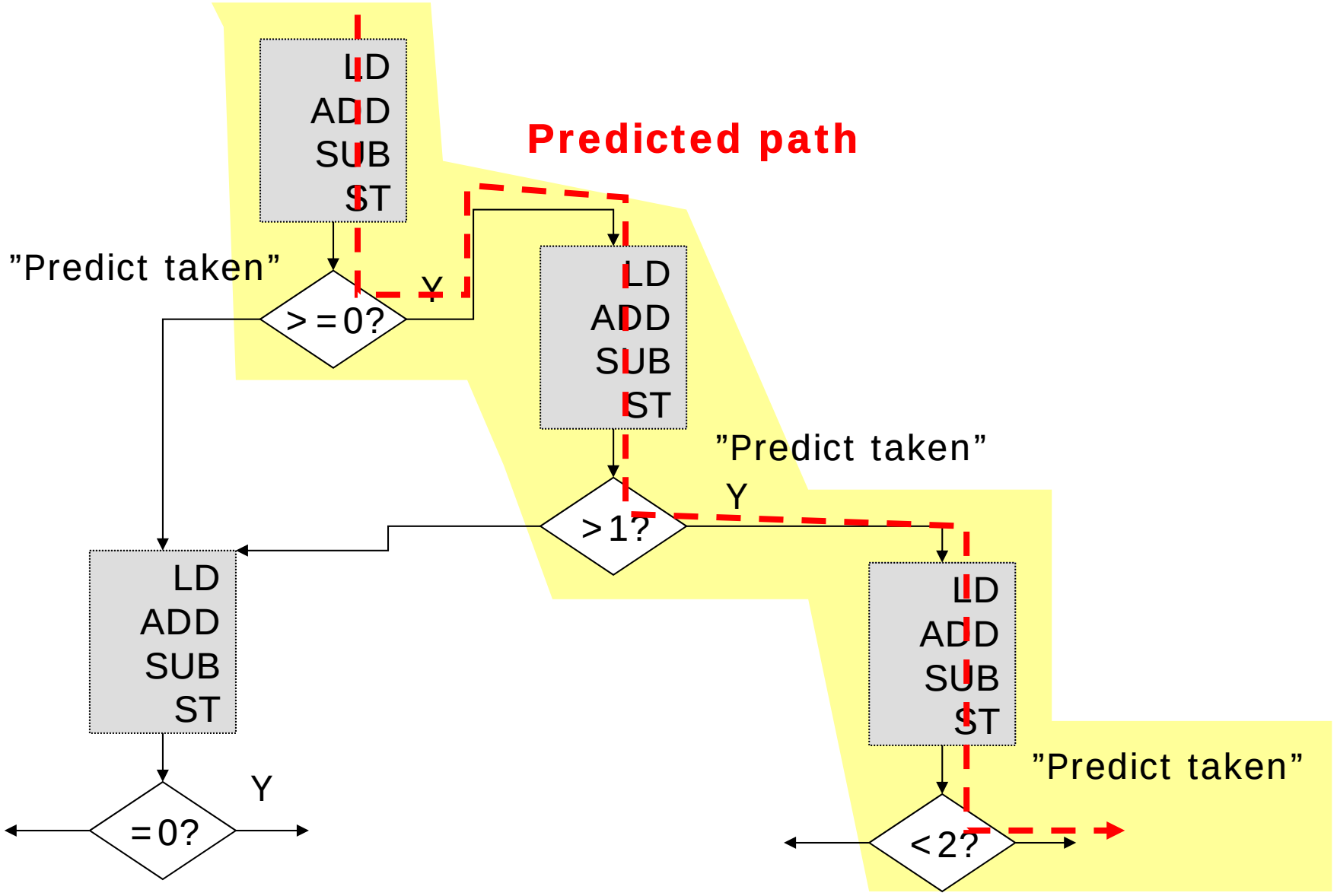


N-level history

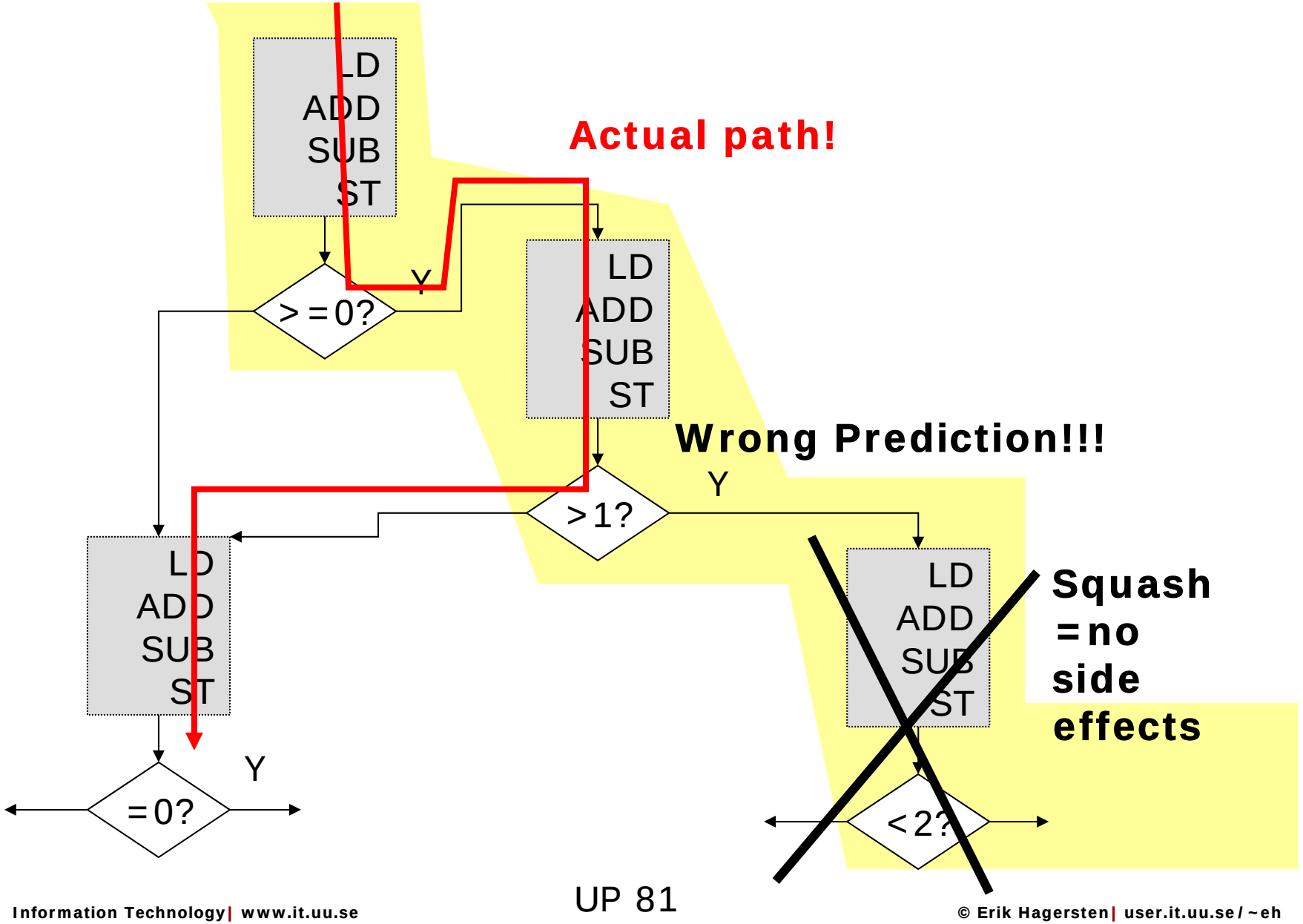
- Not only the PC of the BR instruction matters, also how you've got there is important
- Approach:
  - Record the outcome of the last N branches in a vector of N bits
  - Include the bits in the indexing of the branch table
- Pros/Cons: Same BR instruction may have multiple entries in the branch table

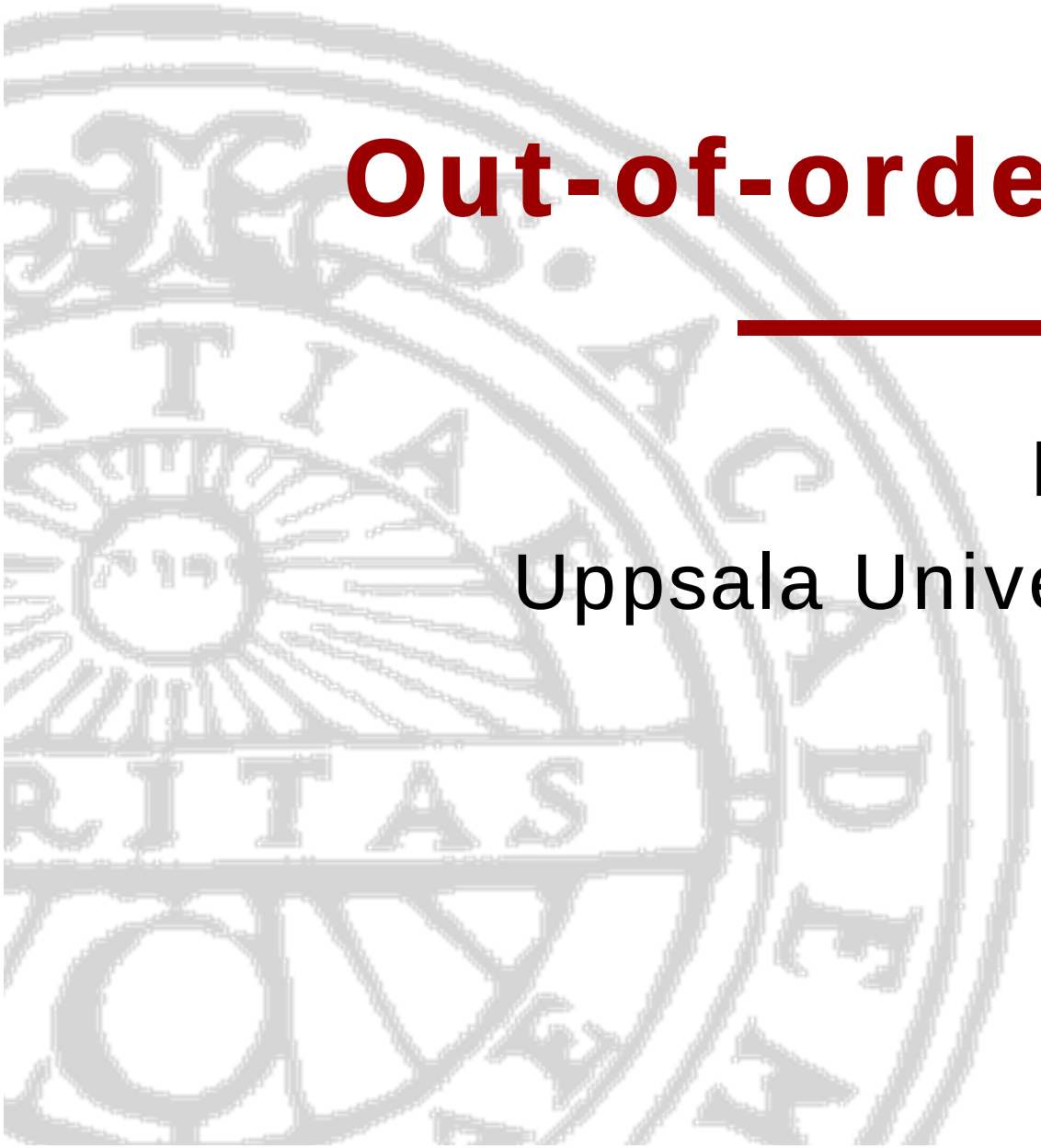
(N,M) prediction = N levels of M-bit prediction

Branch prediction



Misspredication

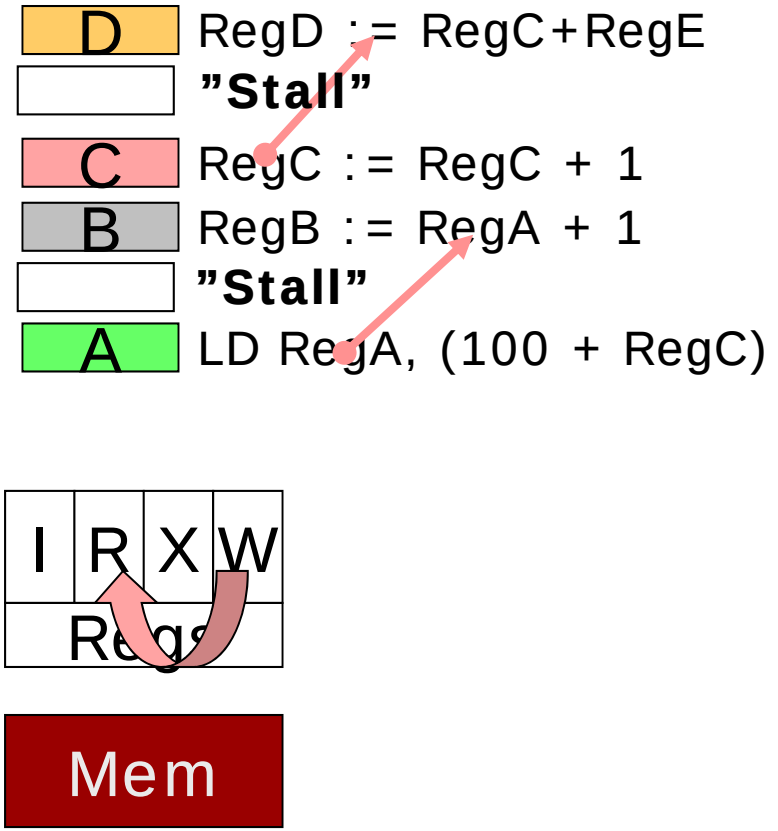




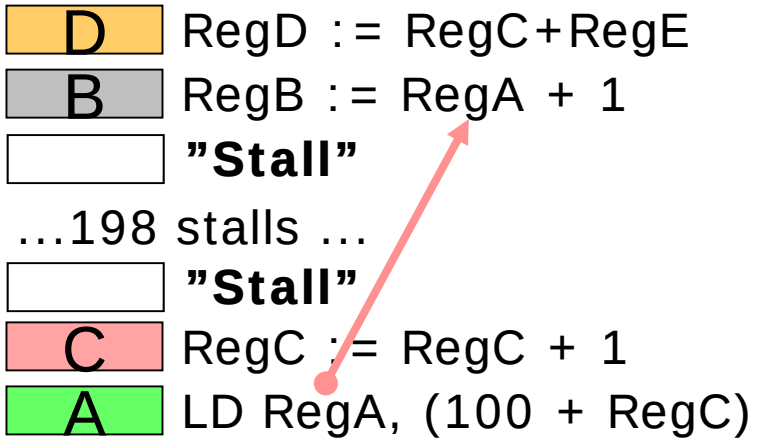
Out-of-order Execution

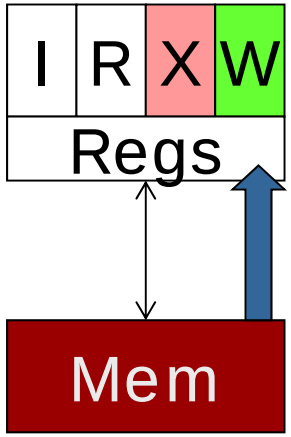
Erik Hagersten
Uppsala University, Sweden
eh@it.uu.se

Data dependency fix 1: pipeline delays (aka bubbles)



Slow Memory Makes it Worse

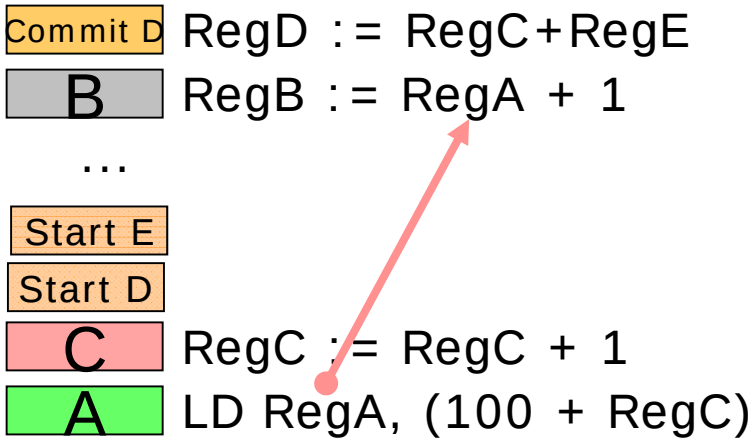




Load data from memory

~200 cycle access time

Fix: Out-of-order execution



Nifty Hardware:

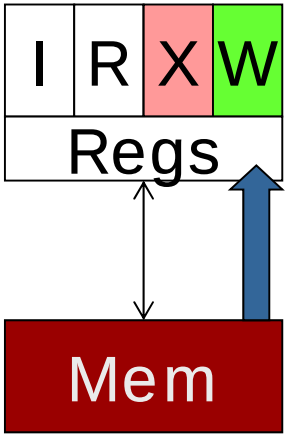
Start executing new instructions while an earlier instruction i stalled

Commit instructions in-order, i.e., apply instruction side-effects in program order

Solve data dependencies among in-flight instructions dynamically (e.g. C → D)

The reorder-buffer (ROB) stores instruction in flight and maintain their order

If an instruction gets an exception (e.g, the TLB complains) squash the following in-flight instructions and handle the exception



~200 cycle access time

ROB typically has space for ~100-200 instructions

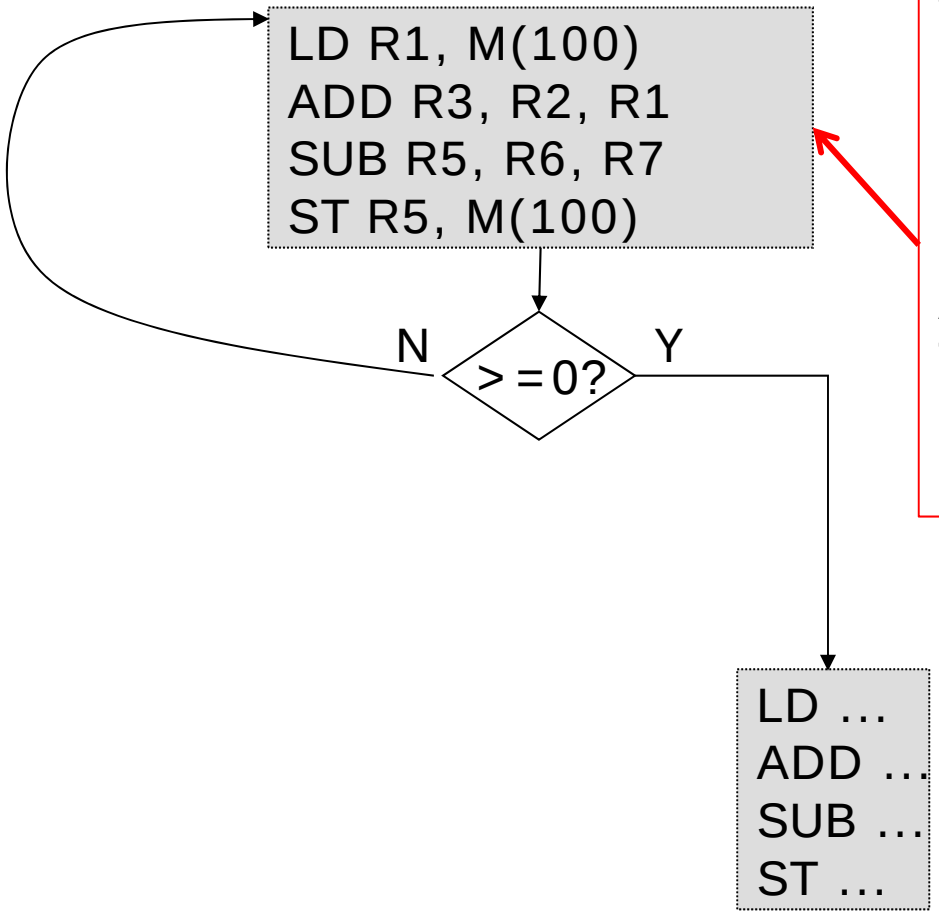
UPPSALA UNIVERSITET

(Tomasulo's Algorithm)

- IBM 360/91 mid 60's
- High performance without compiler support
- Extended for modern architectures
- Many implementations (PowerPC, Pentium…)

© Erik Hagersten| user.it.uu.se/~eh

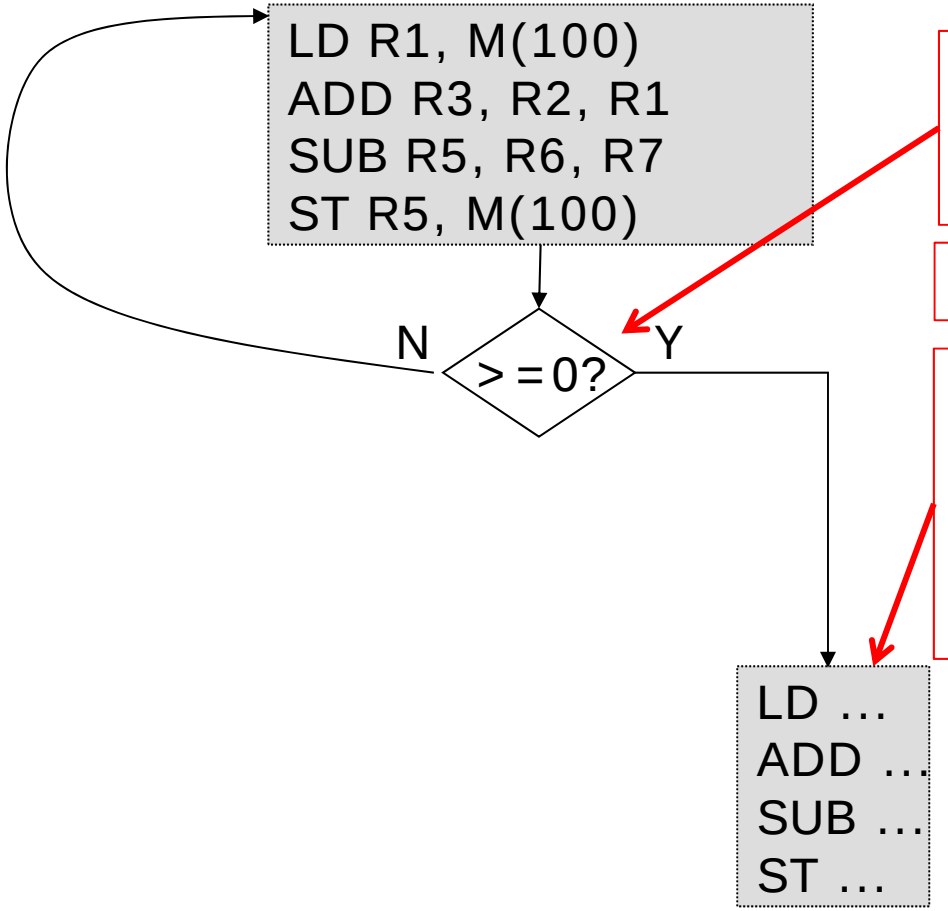
Out-of-order execution Improving ILP



The HW may execute instructions in a different order, but will make the ”side-effects” of the instructions appear in order.

Assume that LD takes a long time. The ADD is dependent on the LD ☹
Start the SUB and ST before the ADD
Update R5 and M(100) **after** R3

Branch prediction

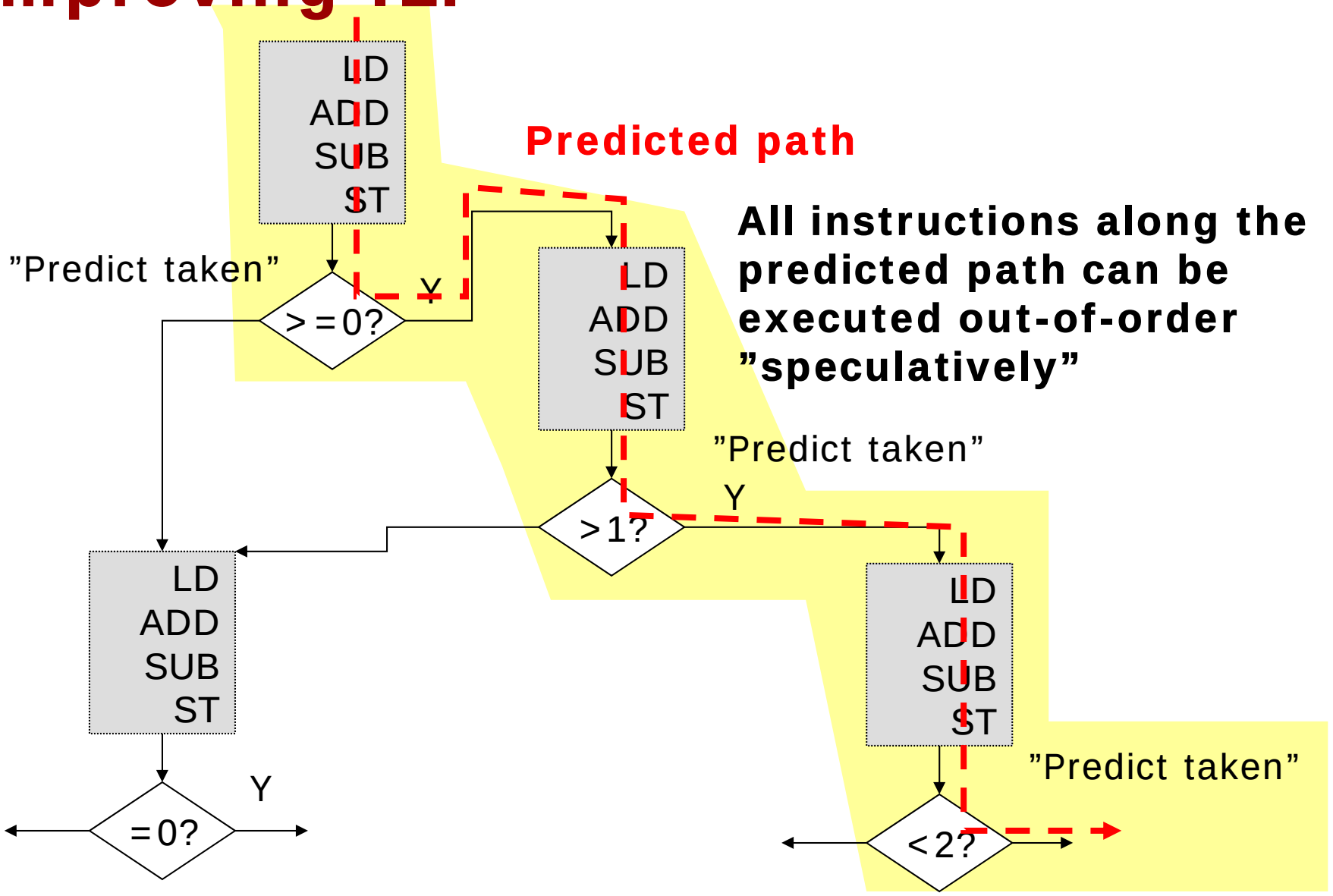


The HW can guess if the branch is taken or not and avoid branch stalls if the guess is correct,

Assume the guess is ”Y”.

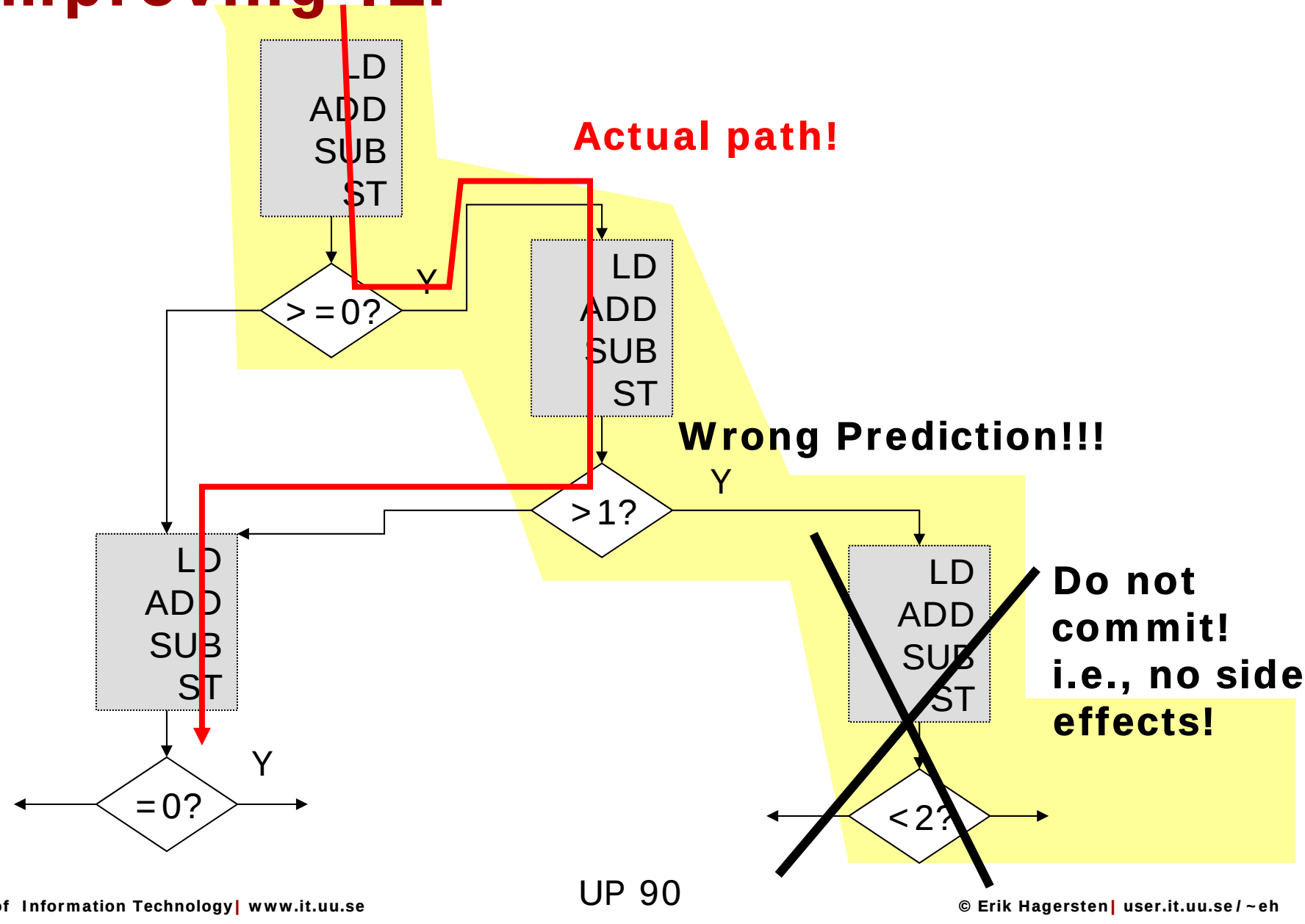
The HW can start executing these instruction **before** the outcome of the branch is known, but cannot allow any ”side-effect” to take place until the outcome is known.

Fix: Scheduling Past Branches Improving ILP



All instructions along the predicted path can be executed out-of-order "speculatively"

Fix: Scheduling Past Branches Improving ILP



UPPSALA UNIVERSITET

How are we doing?

- Create and explore locality:
  a) Spatial locality
  b) Temporal locality
  c) Geographical locality
- Create and explore parallelism
  a) Instruction level parallelism (ILP)
  b) Thread level parallelism (TLP)
  c) Memory level parallelism (MLP)

© Erik Hagersten| user.it.uu.se/~eh