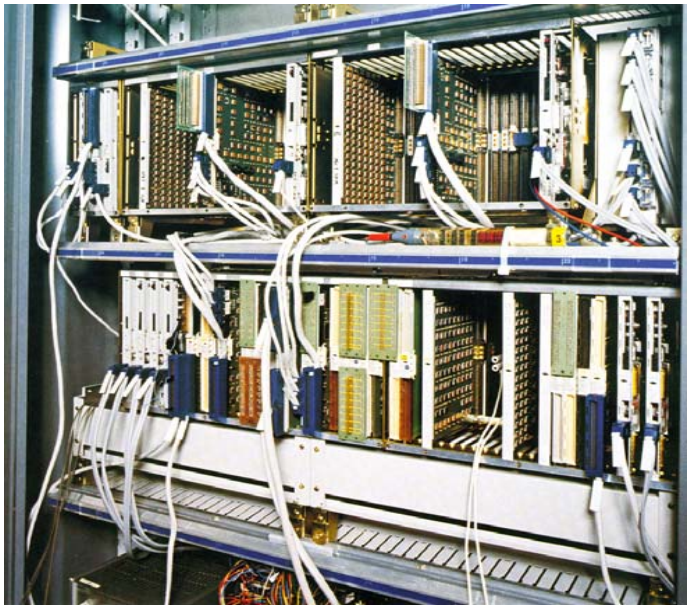
Introduction to Computer Architecture

Erik Hagersten
Uppsala University

25 years ago: APZ 212 @ 5MHz

”the AXE supercomputer”



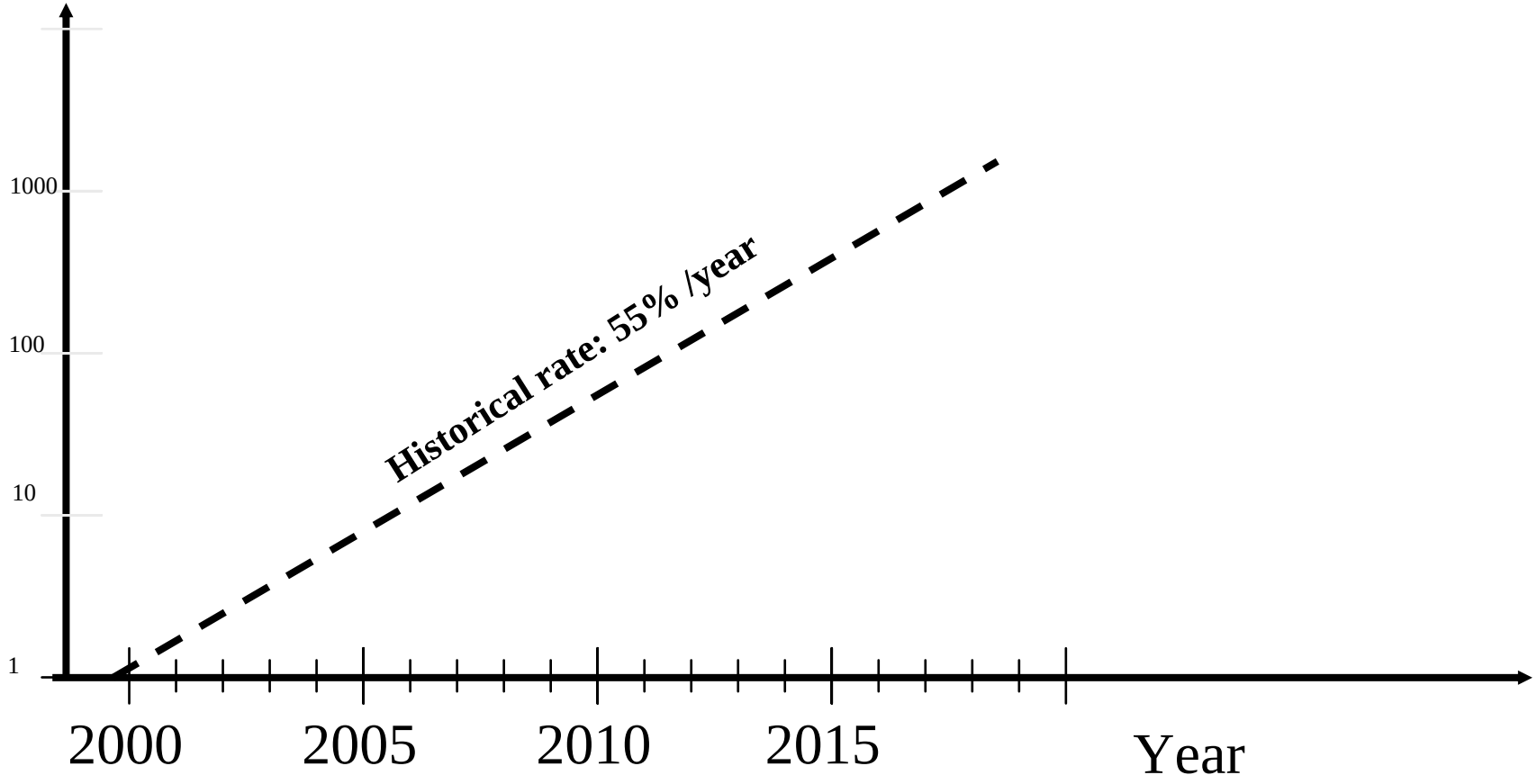
UPPSALA UNIVERSITET

APZ 212 marketing brochure quotes:

- ”Very compact”
  - 6 times the performance
  - 1/6:th the size
  - 1/5 the power consumption
- ”A breakthrough in computer science”
- ”Why more CPU power?”
- ”All the power needed for future development”
- ”...800,000 BHCA, should that ever be needed”
- ”SPC computer science at its most elegance”
- ”Using 64 kbit memory chips”
- ”1500W power consumption

CPU Improvements

Relative Performance
[log scale]



UPPSALA UNIVERSITET

Historically: ~55% improvement per year

- CPU architectural improvements
  - Pipelines,superscalar, O-O-O, smart caches, prefetching

  --> ~15% improvement per year
- Faster clock
  - shrinkage of the transistors, deeper pipelines...

  --> ~30% improvement per year
- More on-chip state
  - larger caches, reordering buffers, registers...

  --> ~10% improvement per year

UPPSALA UNIVERSITET

How to get efficient architectures…

- Creating and exploring:
  1) Locality
  2) Parallelism

UPPSALA UNIVERSITET

How to get efficient architectures…

- Creating and exploring:
  1) Locality
     a) Spatial locality
     b) Temporal locality
     c) Geographical locality
  2) Parallelism
     a) Instruction level
     b) Loop level
     c) Thread level

UPPSALA UNIVERSITET

Architecture complexity

- Normally hidden from a user
- Not important to most programmers
- Must be understood for
  - Getting high performance
  - Compilers writers
  - Operating system writers
  - Understanding some interesting bugs

UPPSALA UNIVERSITET

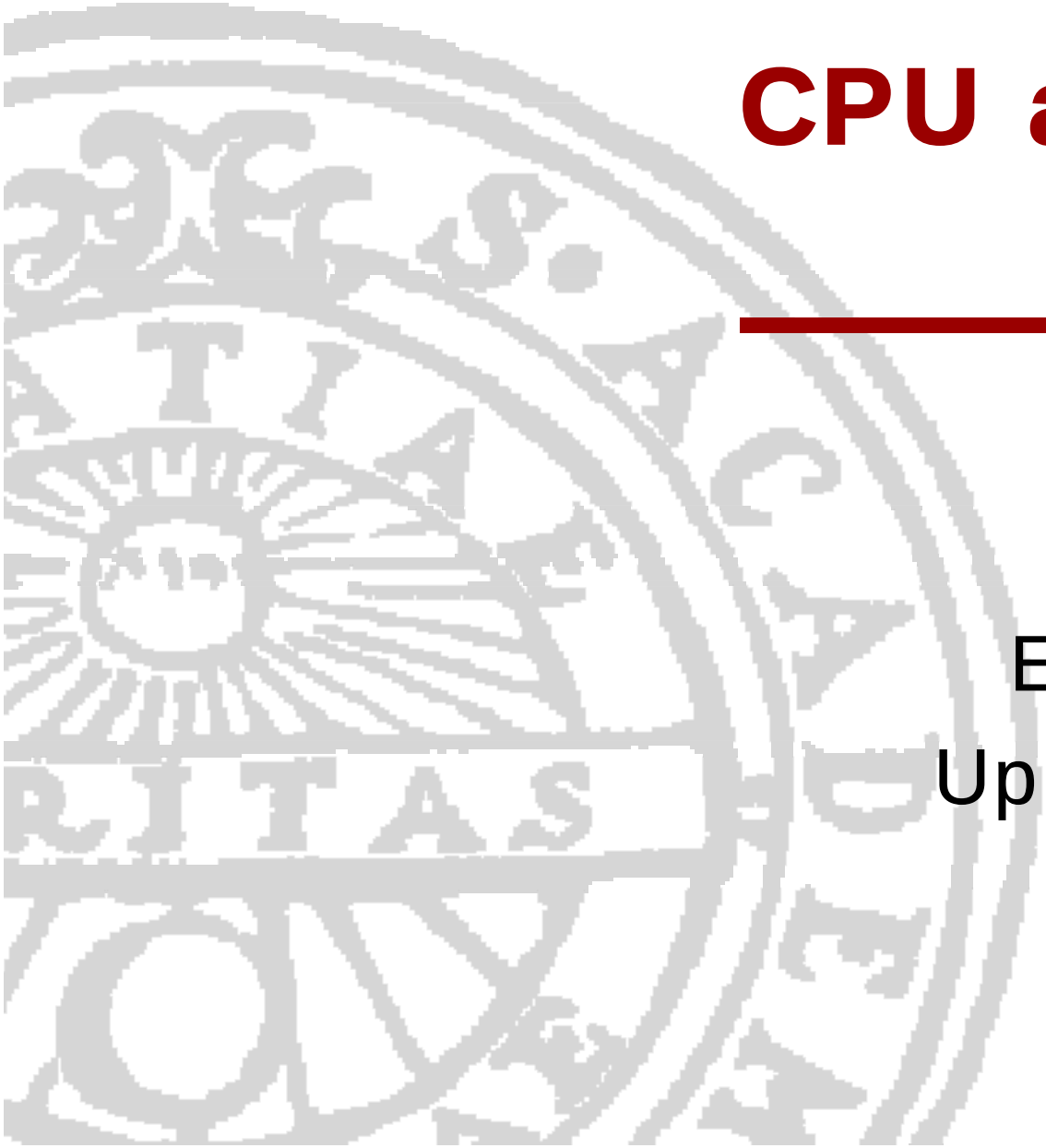
Outline of these lectures

1. Uniprocessors
2. Multiprocessors
3. Multicores
4. Optimizing for Speed

UPPSALA UNIVERSITET

Outline of these lectures

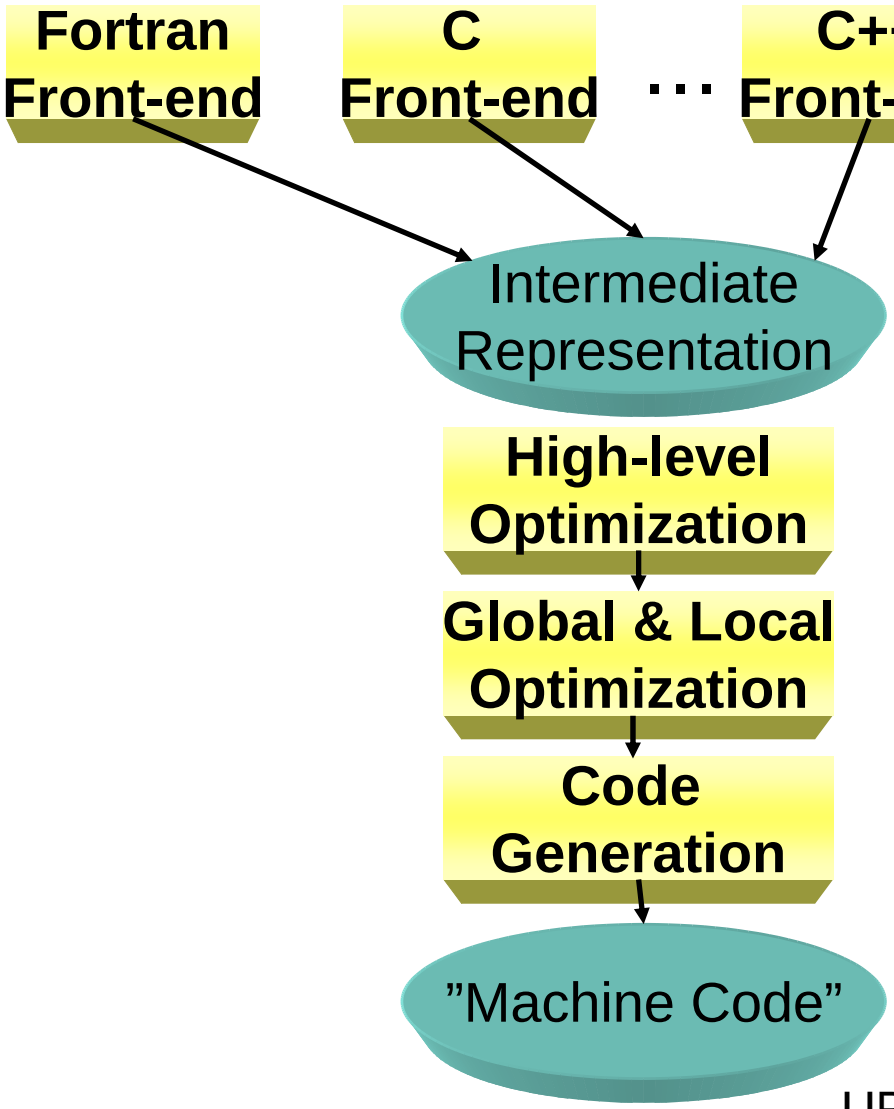
1. Uniprocessors
   - Pipelining, superscalars, …
   - Memory, caches, …
2. Multiprocessors
3. Multicores
4. Optimizing for speed



CPU architecture overview

Erik Hagersten
Uppsala University

Compiler Organization



Machine-independent Translation

Procedure in-lining
Loop transformation

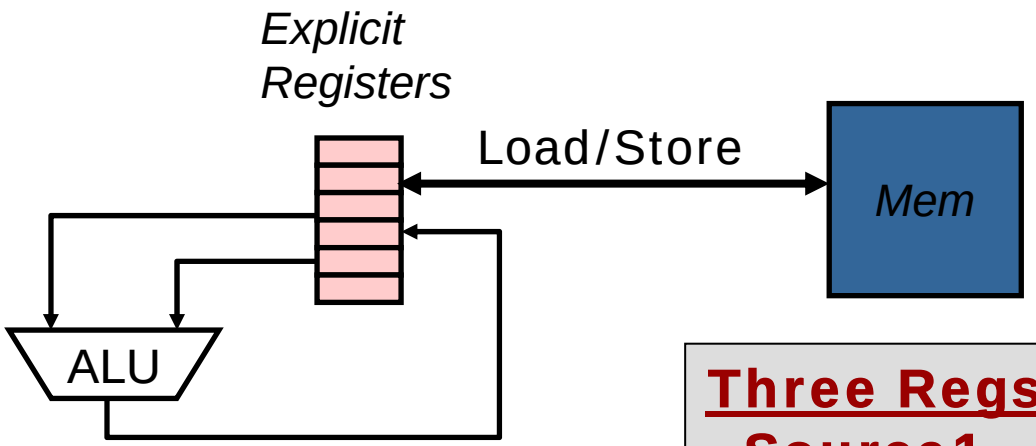
Register Allocation
Common sub-expressions

Instruction selection
constant folding

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

ALU ops: Reg -->Reg

Mem ops: Reg <--> Mem



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

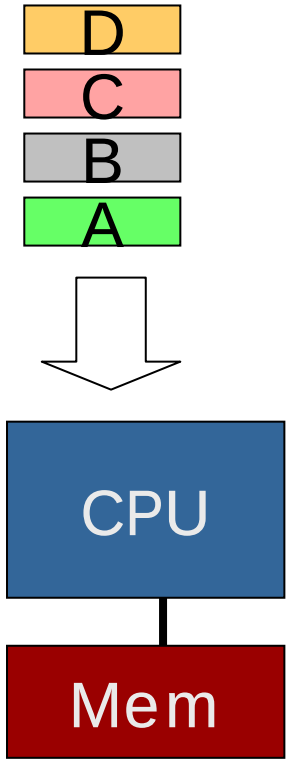
Example: C = A + B

```
Load R1, [A]
Load R3, [B]
Add R2, R1, R3
Store R2, [C]
```

Memory Accesses

Lifting the CPU hood (simplified…)

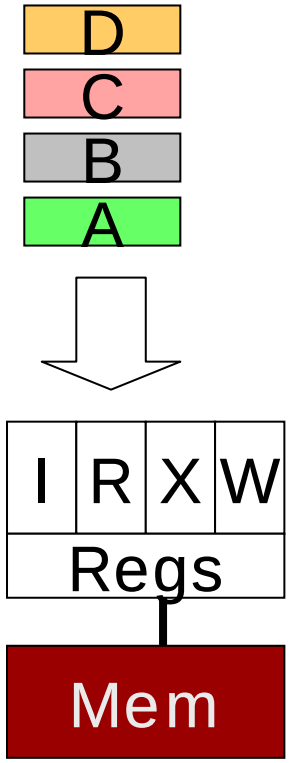
Instructions:



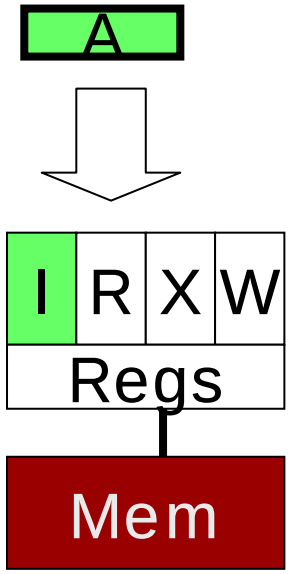
UPPSALA UNIVERSITET

Pipeline

Instructions:

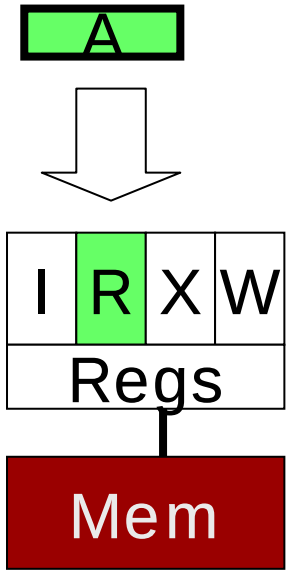


Pipeline

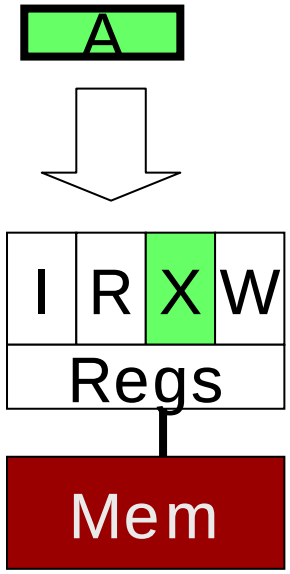


UPPSALA UNIVERSITET

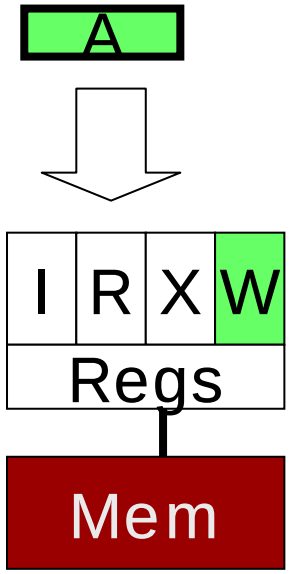
Pipeline



Pipeline



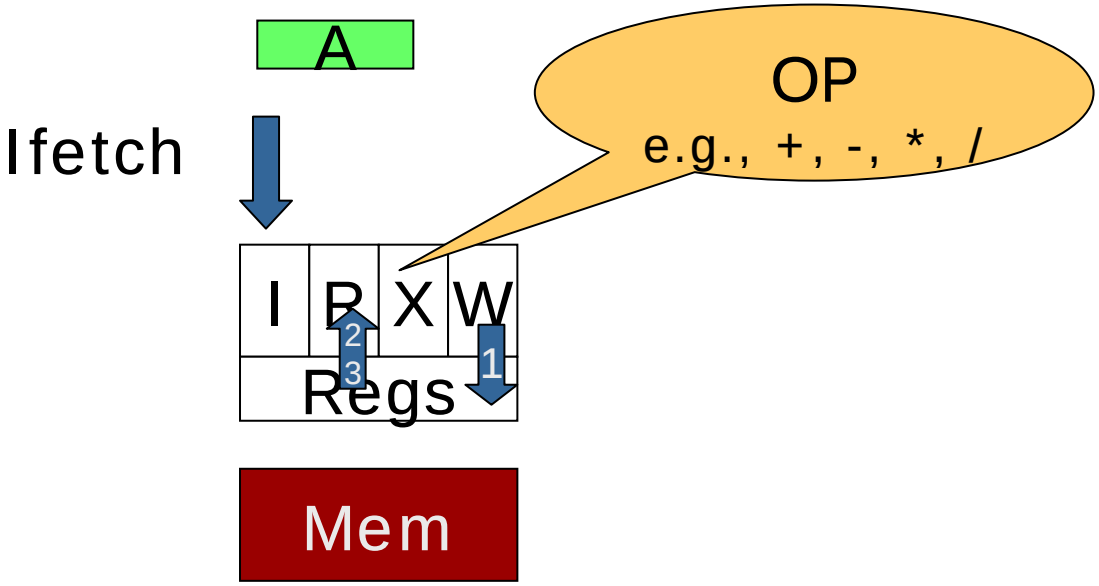
Pipeline:



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

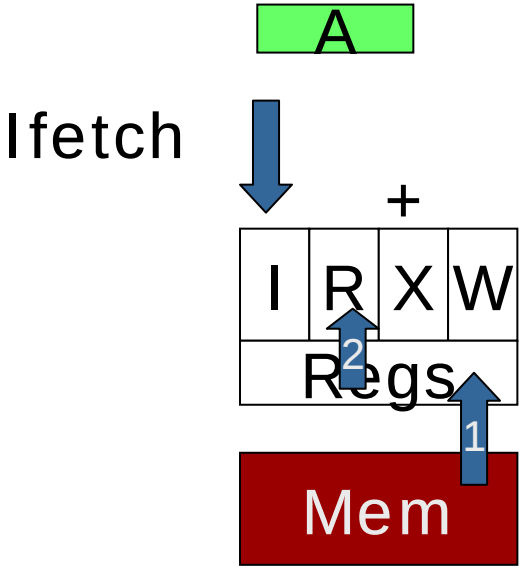
Register Operations:

R1 := R2 op R3



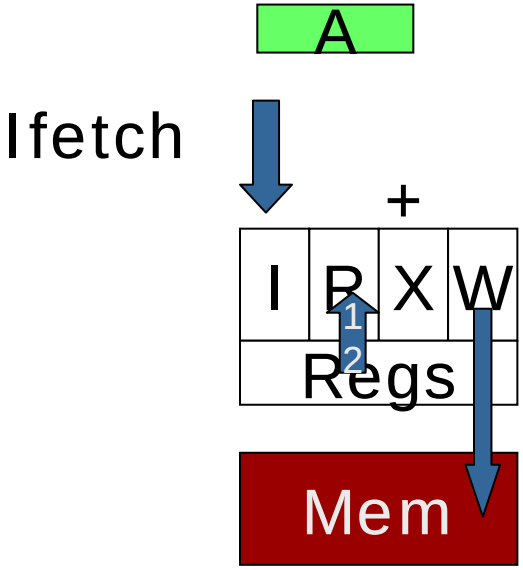
Load Operation:

LD R1, mem[cnst+R2]



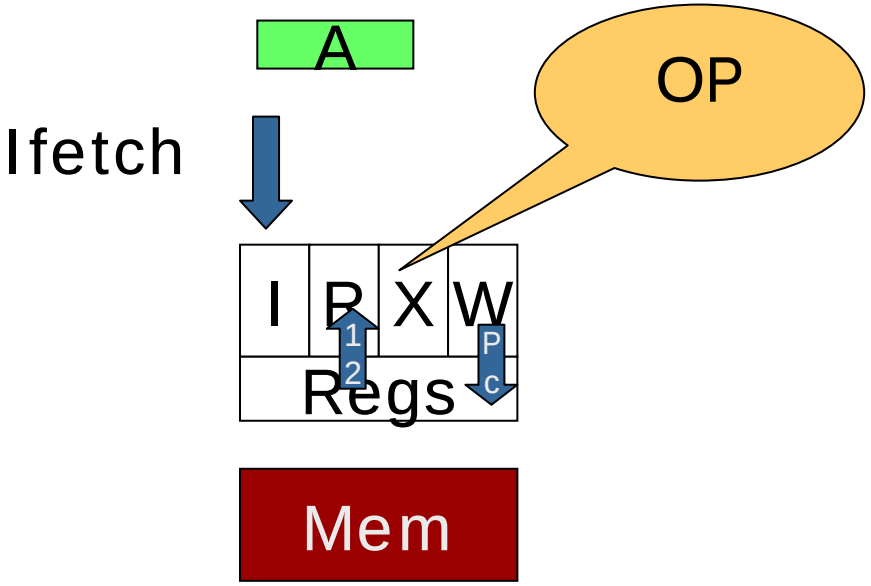
Store Operation:

ST mem[cnst+R1], R2



Branch Operations:

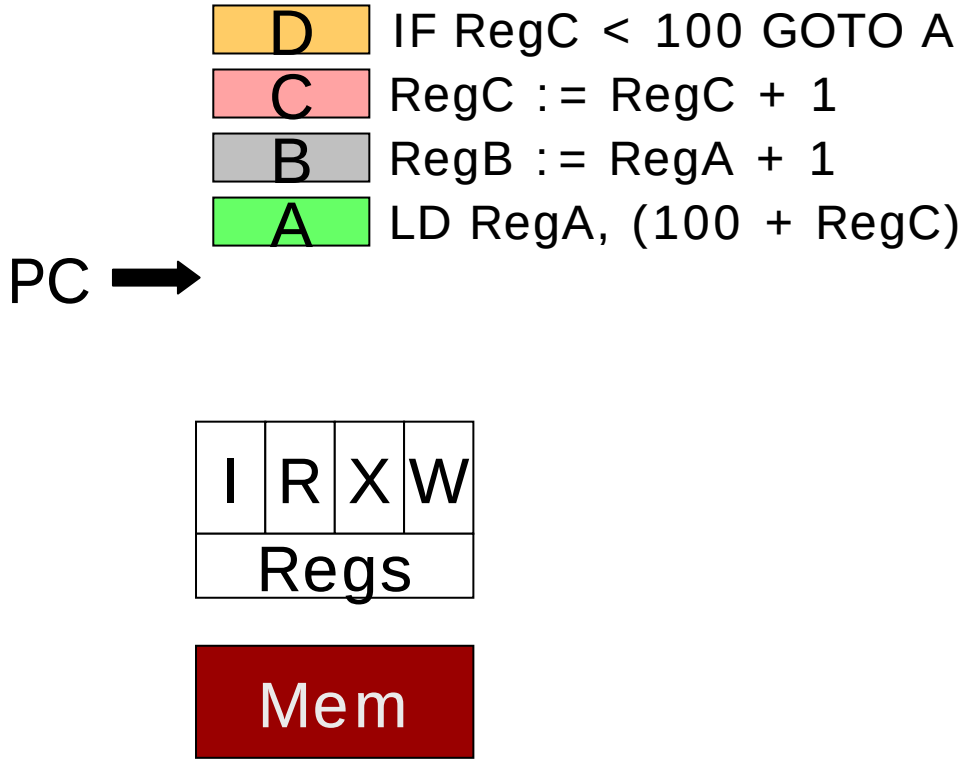
if (R1 Op Const) GOTO mem[R2]



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

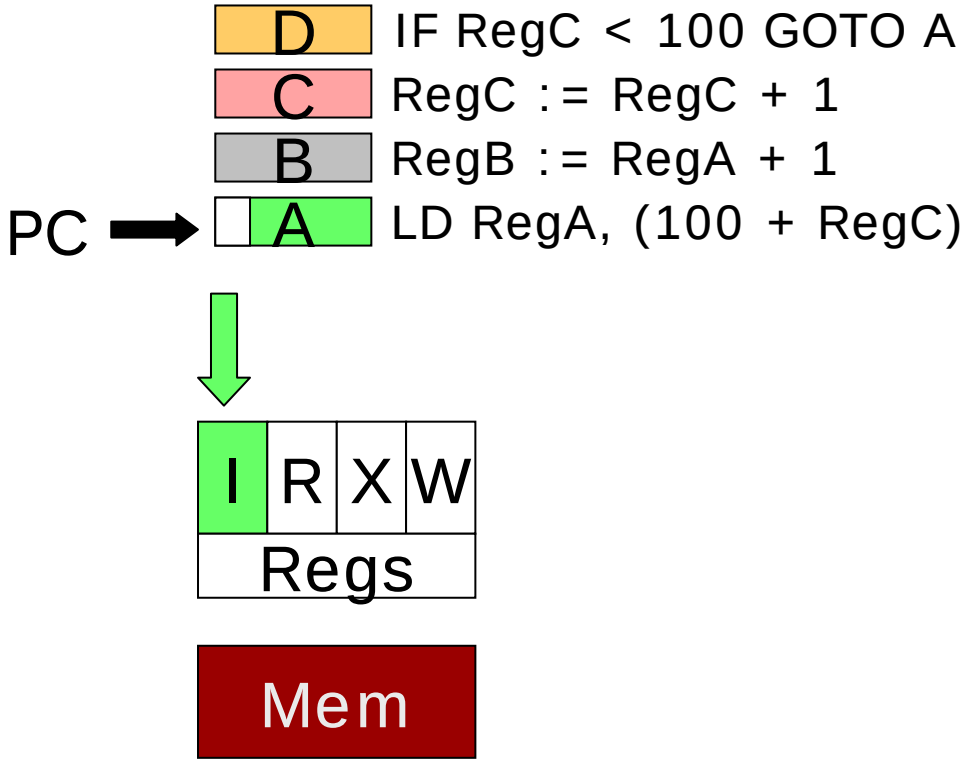

UPPSALA
UNIVERSITET

Initially


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

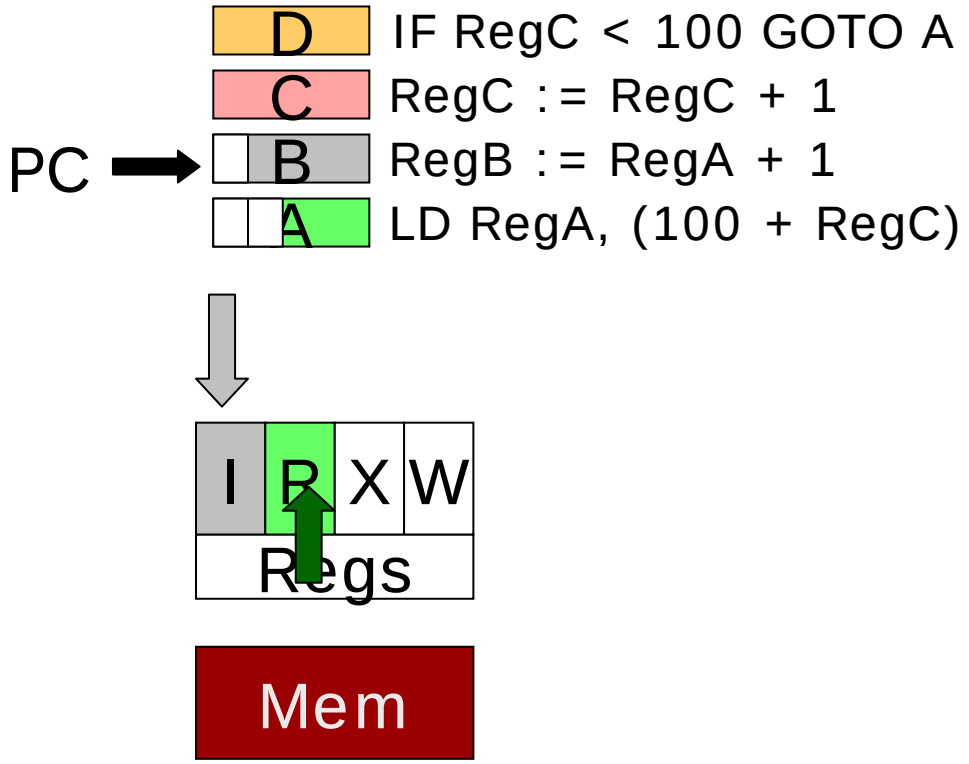

UPPSALA
UNIVERSITET

Cycle 1

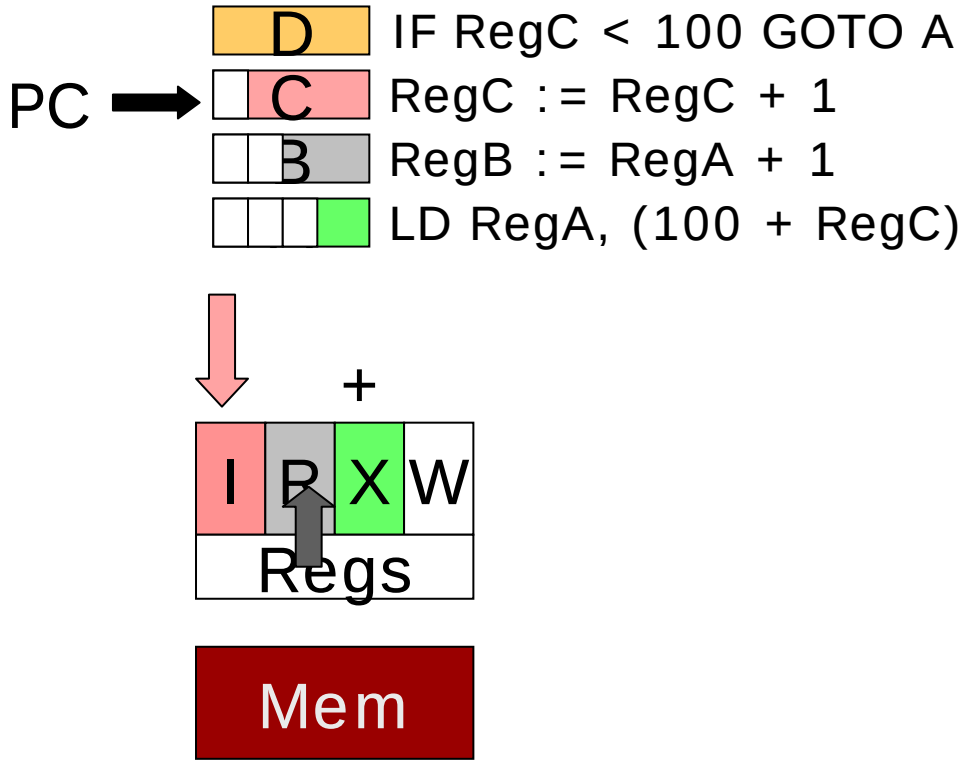

D
IF RegC < 100 GOTO A
C
RegC := RegC + 1
B
RegB := RegA + 1
PC
A
LD RegA, (100 + RegC)
I
R
X
W
Regs
Mem


UPPSALA
UNIVERSITET

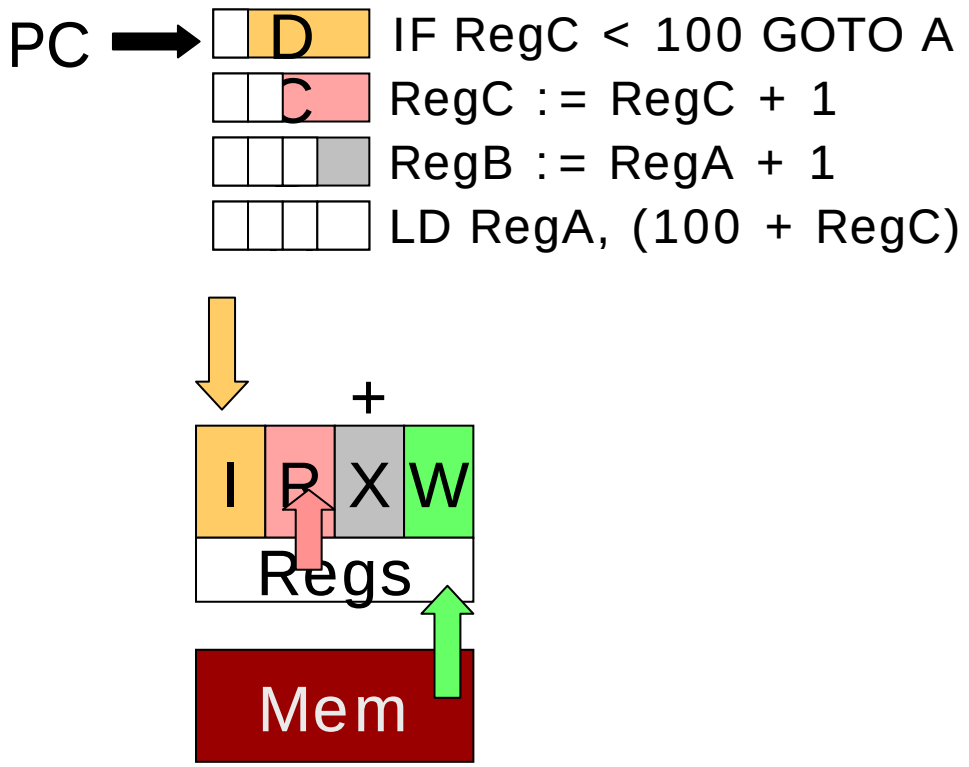
Cycle 2


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

Cycle 3

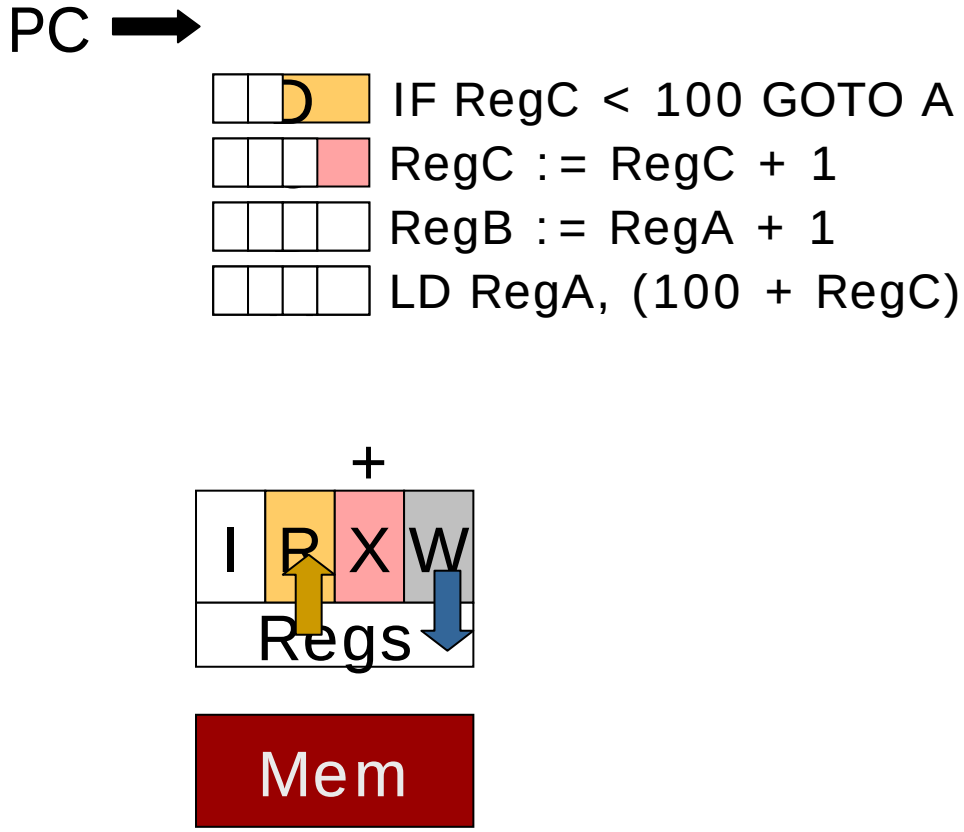

D
C
B
PC
IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)
+
I
B
X
W
Regs
Mem

Cycle 4

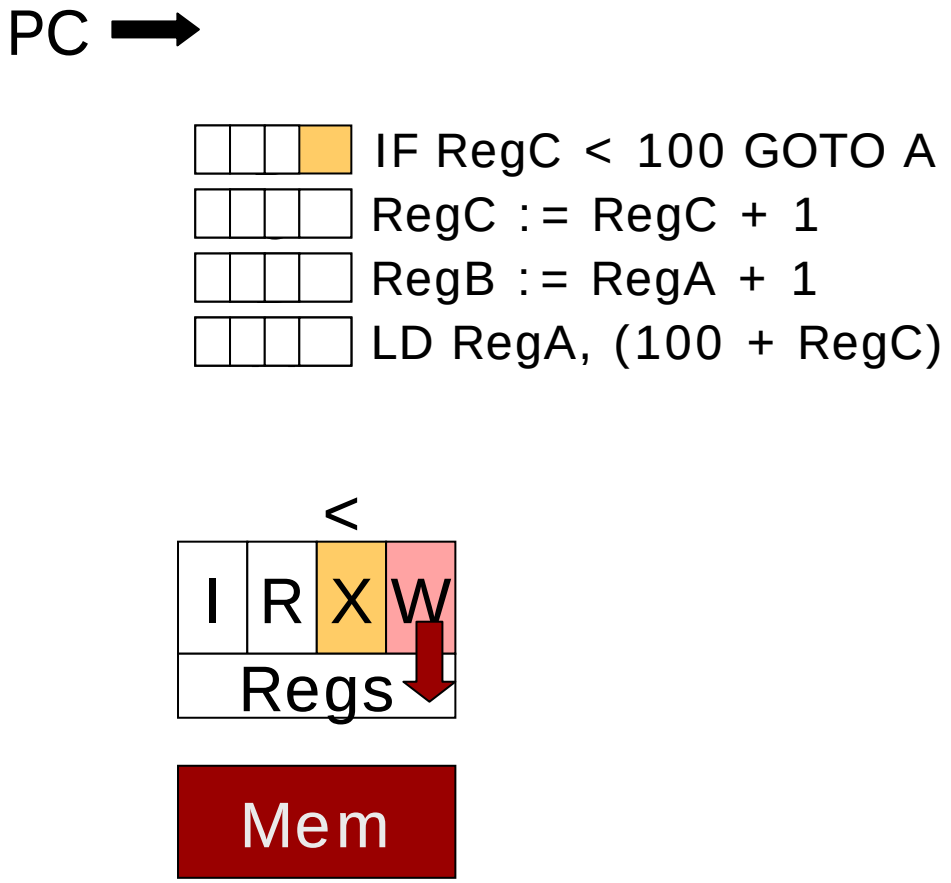

PC
D
C
IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)
+
I
P
X
W
Regs
Mem


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

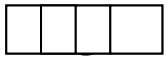
Cycle 6


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

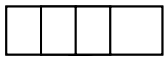
Cycle 7

PC ➡

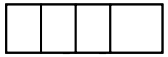
IF RegC < 100 GOTO A



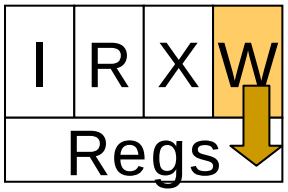
RegC := RegC + 1



RegB := RegA + 1

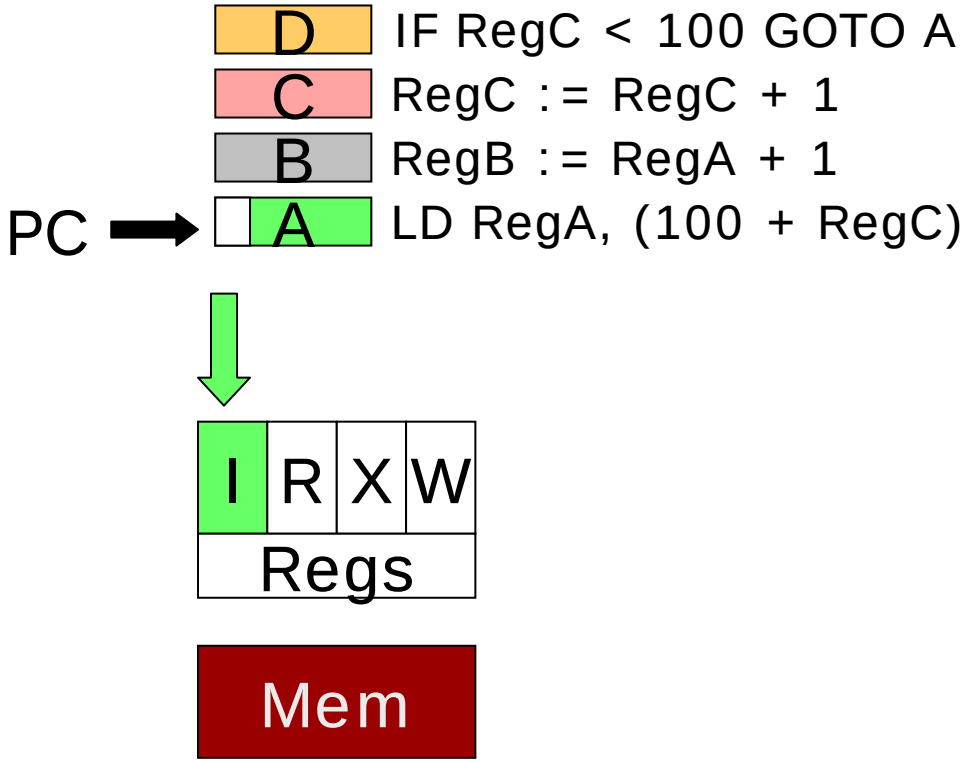
A: 

LD RegA, (100 + RegC)



Branch: Addr(A) ➔ PC

Cycle 8

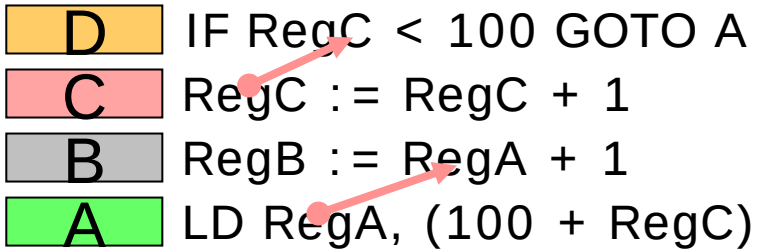

D
IF RegC < 100 GOTO A
C
RegC := RegC + 1
B
RegB := RegA + 1
PC
A
LD RegA, (100 + RegC)
I
R
X
W
Regs
Mem

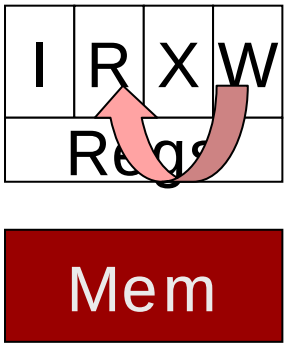
UPPSALA UNIVERSITET

Pipeline Challenges

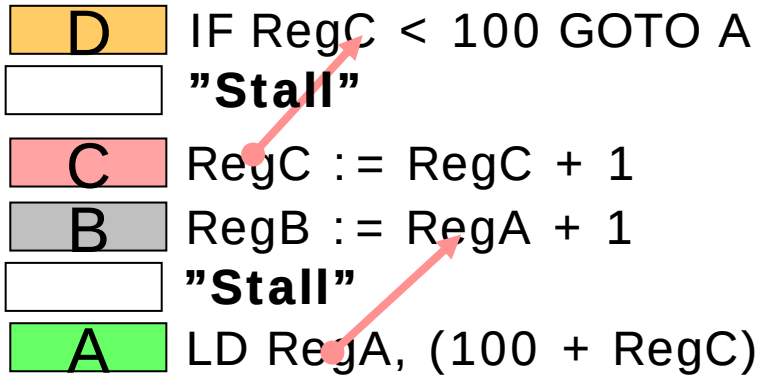
- Balance the pipeline stages
- Setup and hold time overhead
- Handle the feed-back cases (Ford did not have any….)
- Minimize pipeline stalls
- Predict and perform speculative work
- Undo speculative work

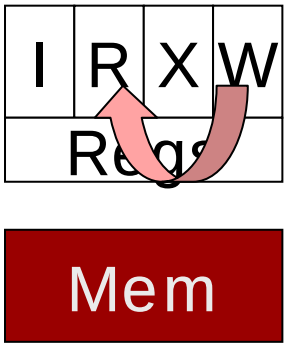
Data dependency ☹



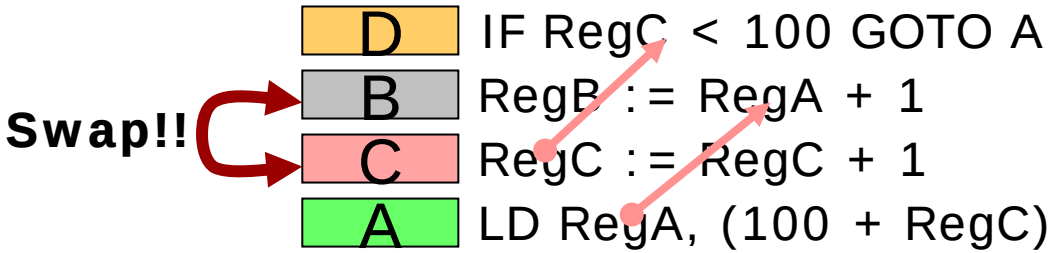


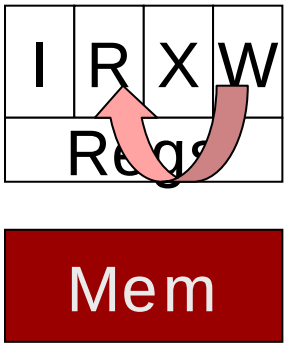
Data dependency ☹



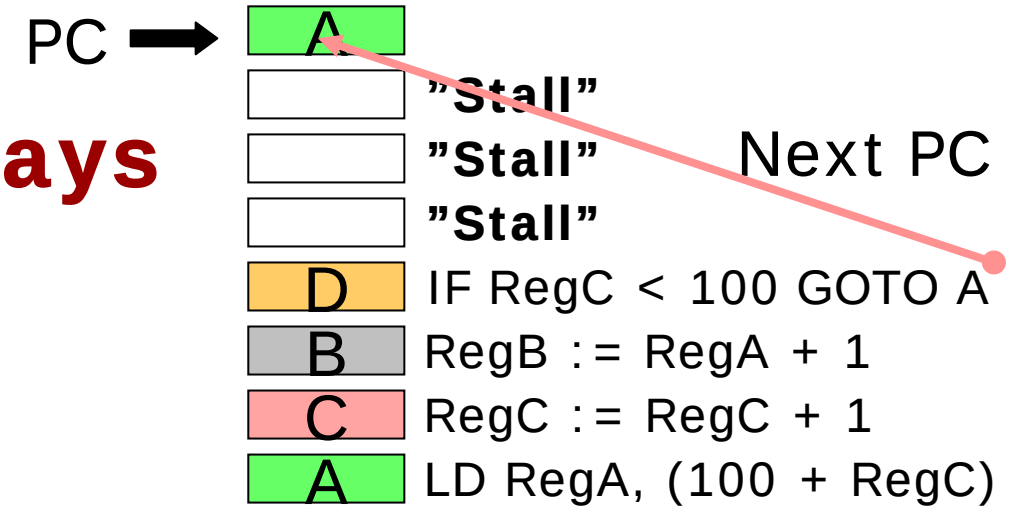


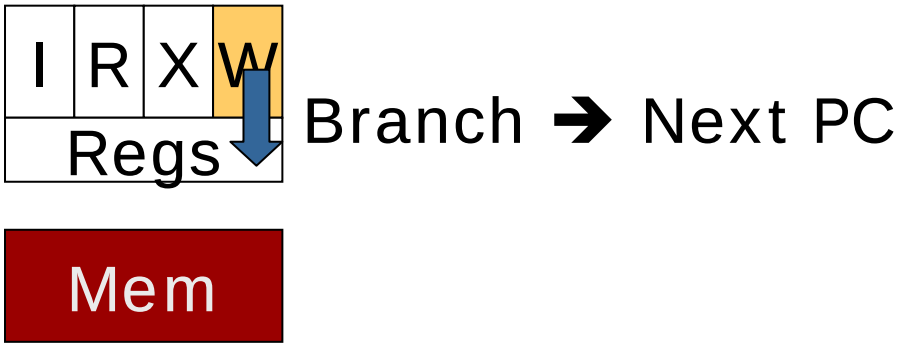
Compiler optimizations





Branch delays

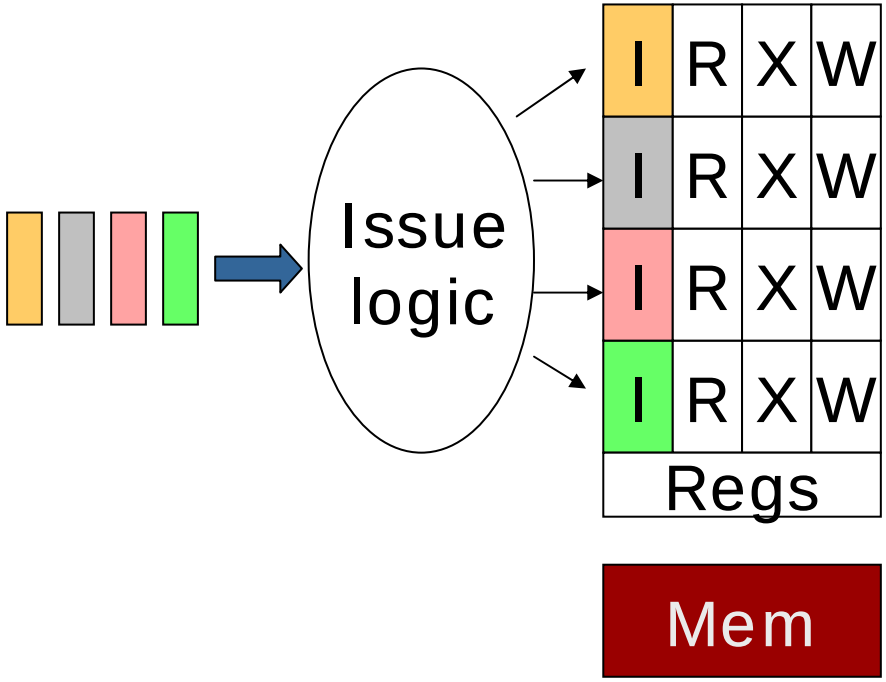




7 cycles per iteration of 4 instructions ☹
Need longer basic blocks with independent instr.

It is actually a lot worse!

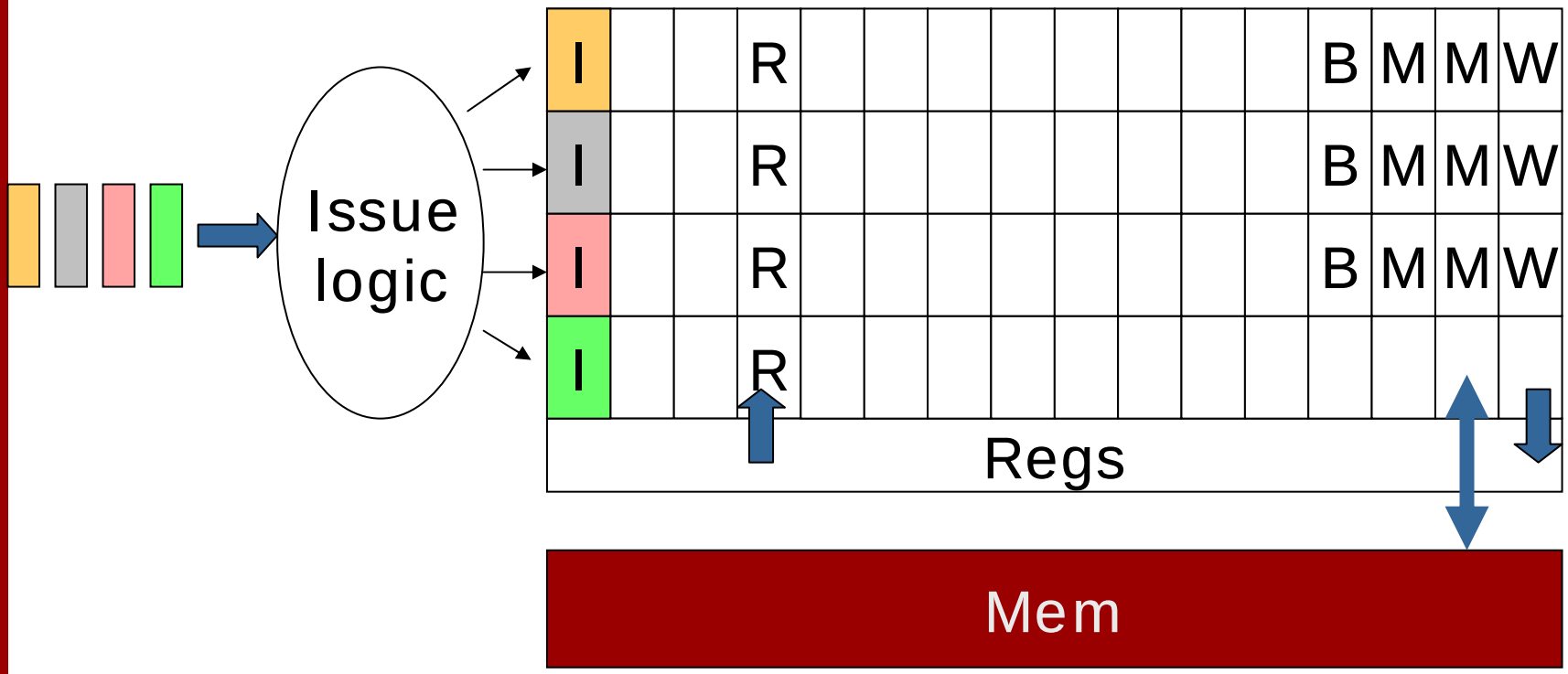
Modern CPUs:"superscalars" with ~4 parallel pipelines



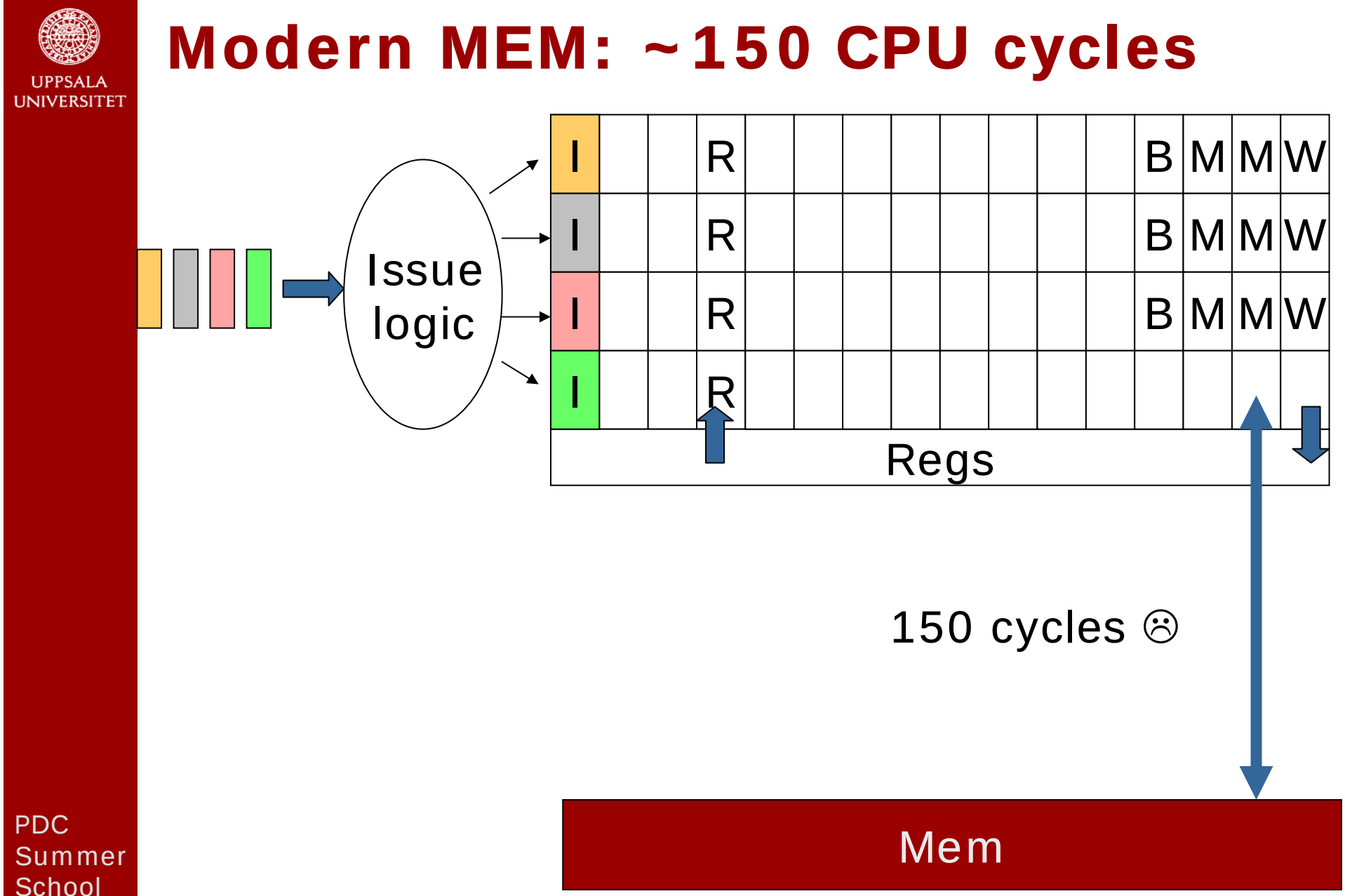
+Higher throughput

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

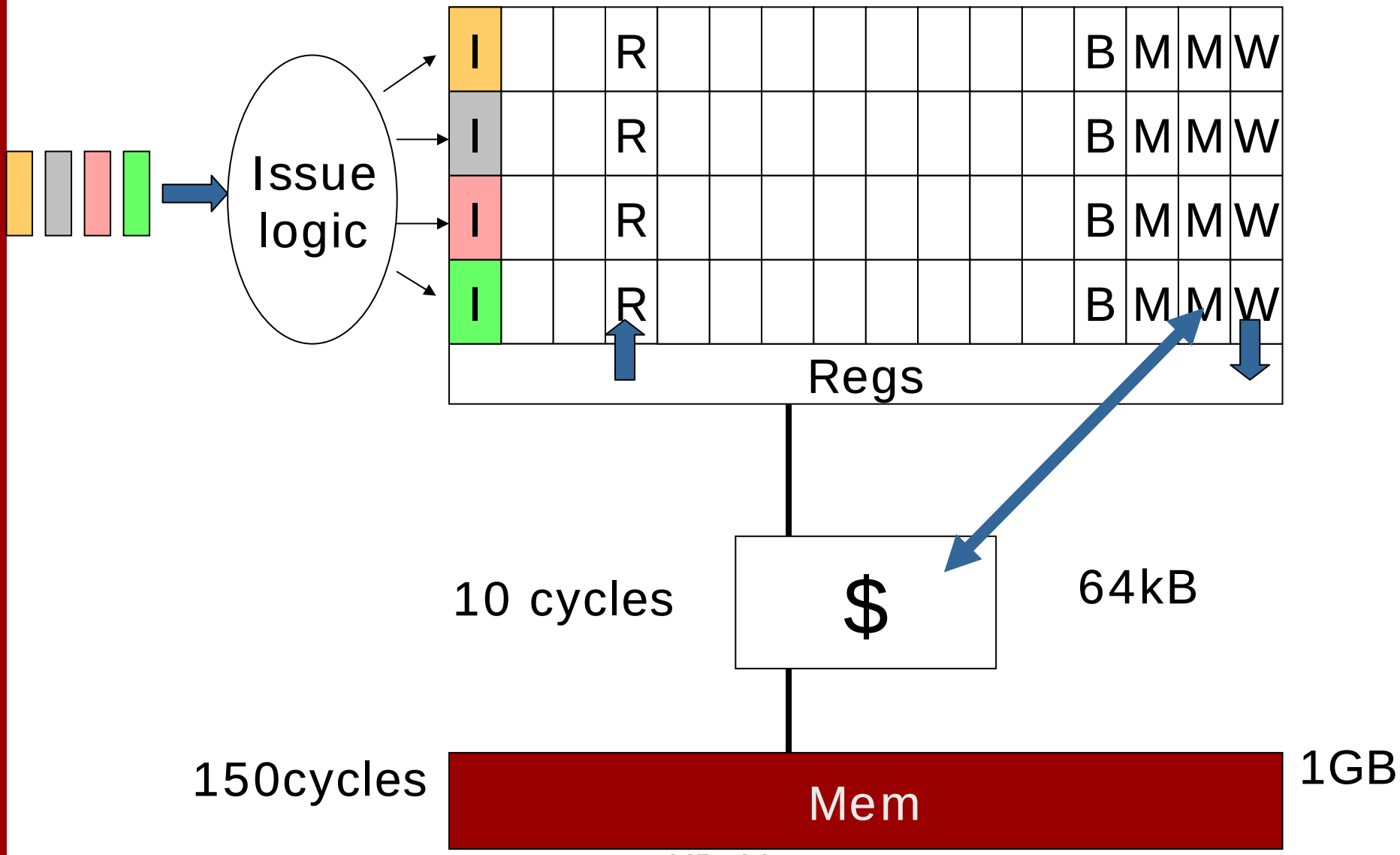
Modern CPUs: ~10-20 stages/pipe



+Shorter cycletime (higher GHz)
- Branch delay even more expensive
- Even harder to find ”enough” independent instr.


Modern MEM: ~150 CPU cycles
Issue logic
I
R
B
M
M
W
Regs
150 cycles ☹
Mem

Fix: Use a cache



UPPSALA UNIVERSITET

Common speculations in CPUs

- Caches [later]
- Address translation caches (TLBs) [later]
- Prefetching (SW&HW, Data & Instr.) [later]
- Branch prediction [later]
- Execute ahead [not covered in this course]

More complications

- Execute instructions out-of-order, but still make it look like an in-order execution [not covered]
- Multithreading (later)
- Multicore
- ...

Caches and more caches
or
spam, spam and spam

Erik Hagersten

Uppsala University, Sweden

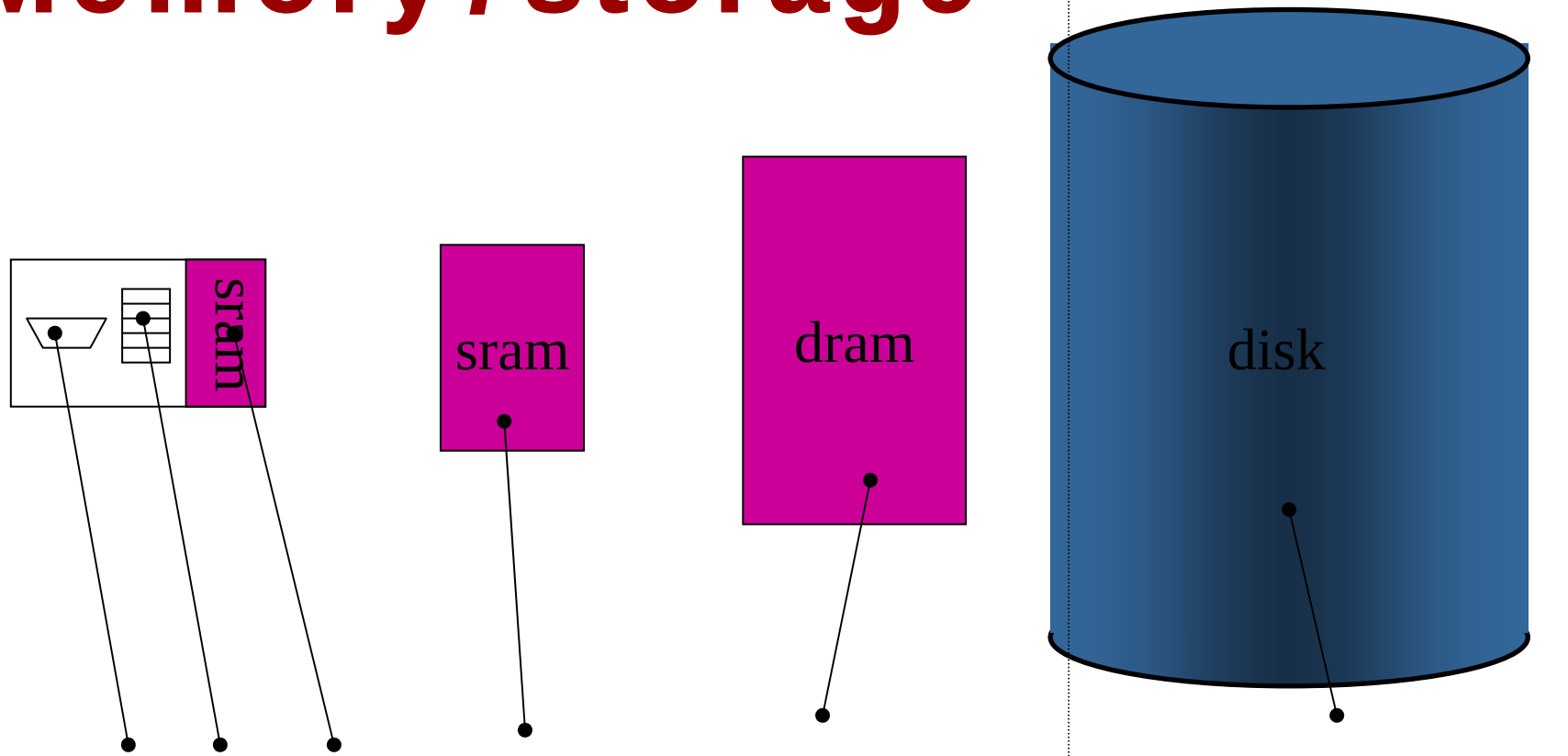
eh@it.uu.se

UPPSALA UNIVERSITET

Cache knowledge useful when...

- Designing a new computer
- Writing an optimized program
  - or compiler
  - or operating system …
- Implementing software caching
  - Web caches
  - Proxies
  - File systems

Memory/storage



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

1kB 64k 4MB 1GB 1 TB

UPPSALA UNIVERSITET

Speed/Size/Cost tradeoff

	Access time	#Entries
Sweden	300s	5 000 000
Uppsala	60s	100 000
Family	10s	200
Sweetheart(s)	1s	10

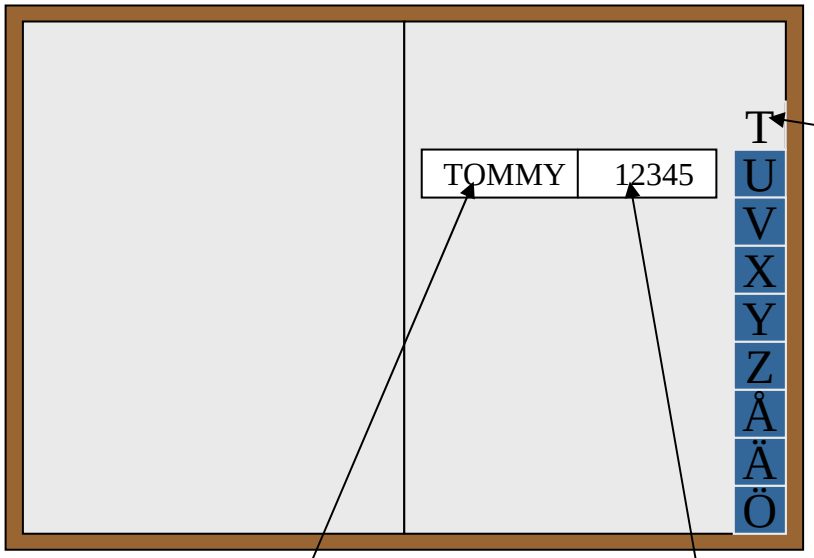
UPPSALA UNIVERSITET

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

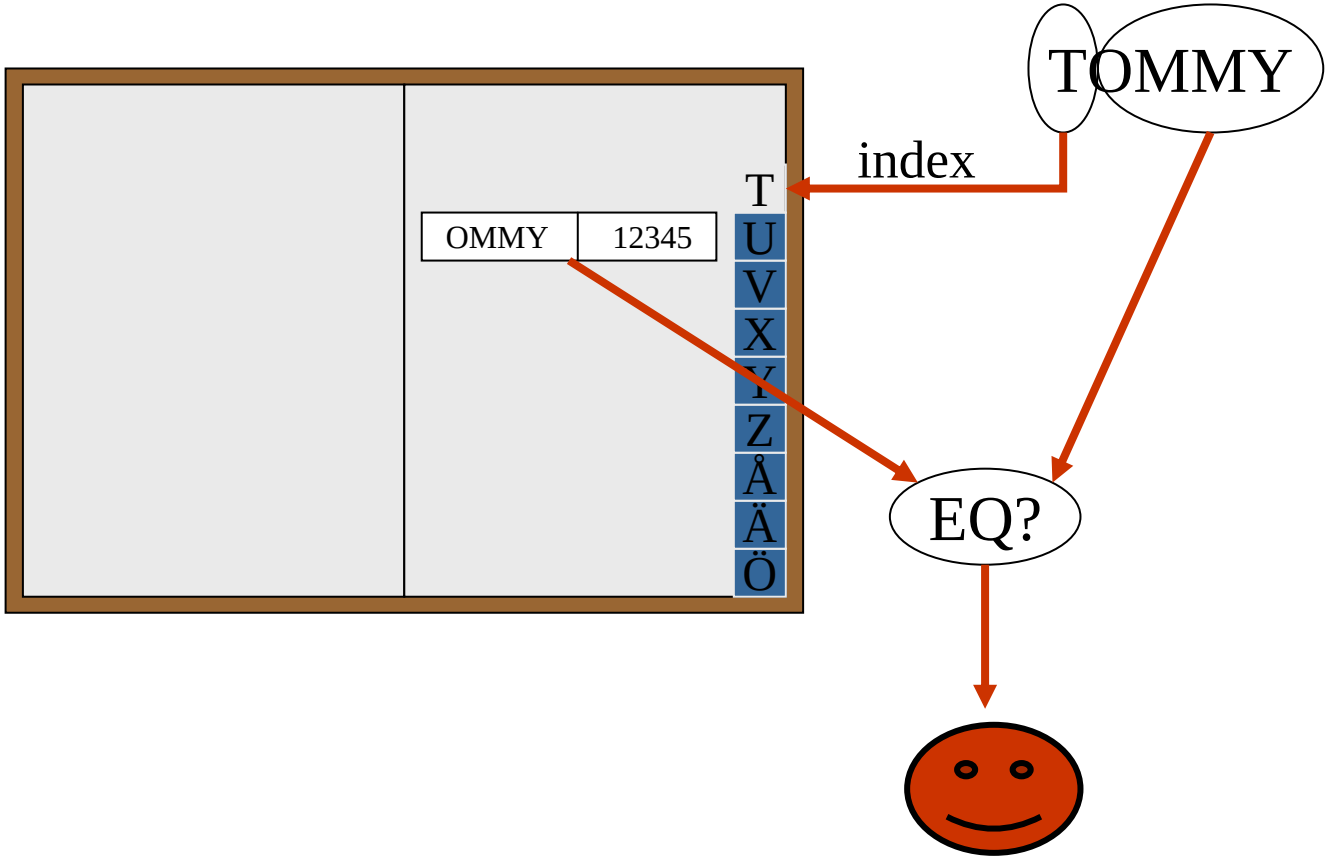
"Address Tag"

"Data"

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

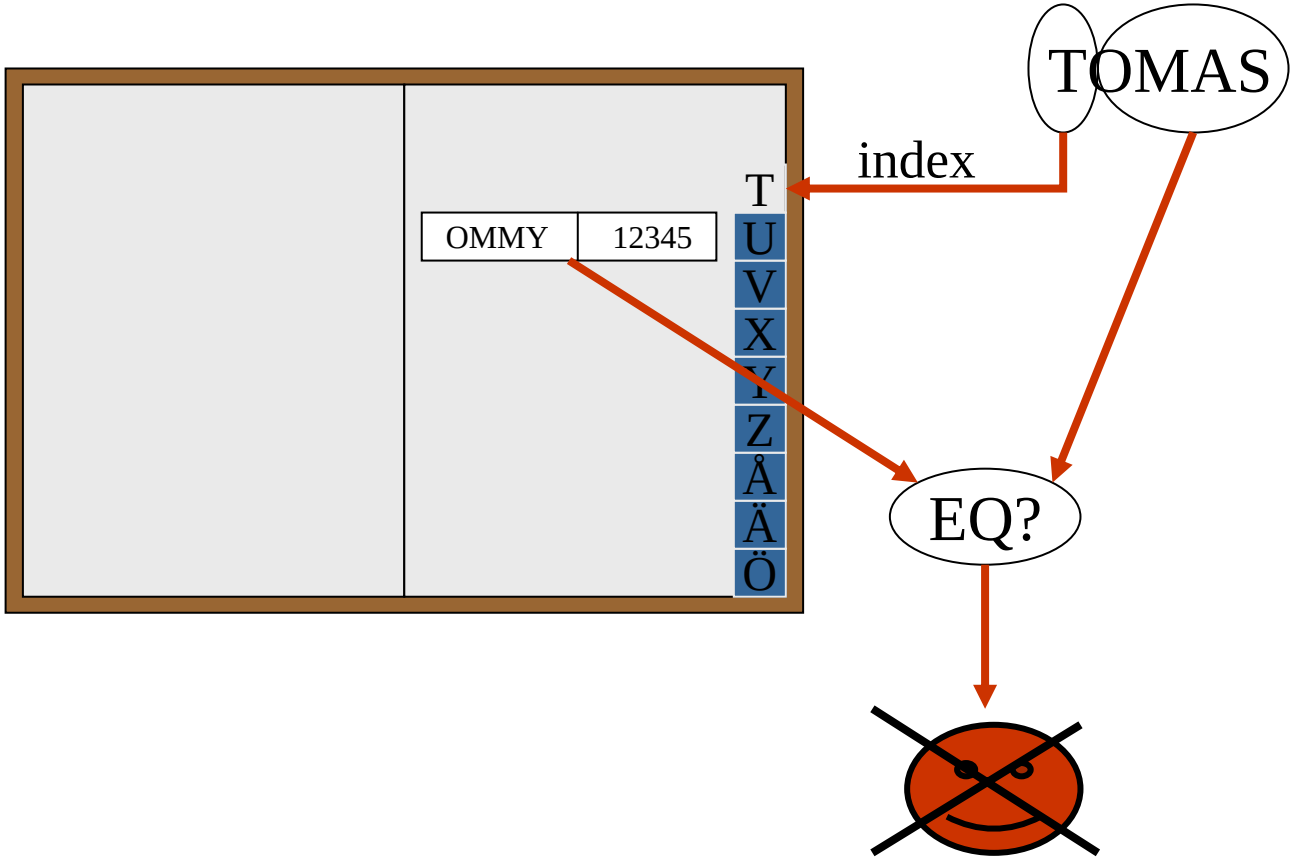
Address Book Cache

Looking for Tommy's Number



Address Book Cache

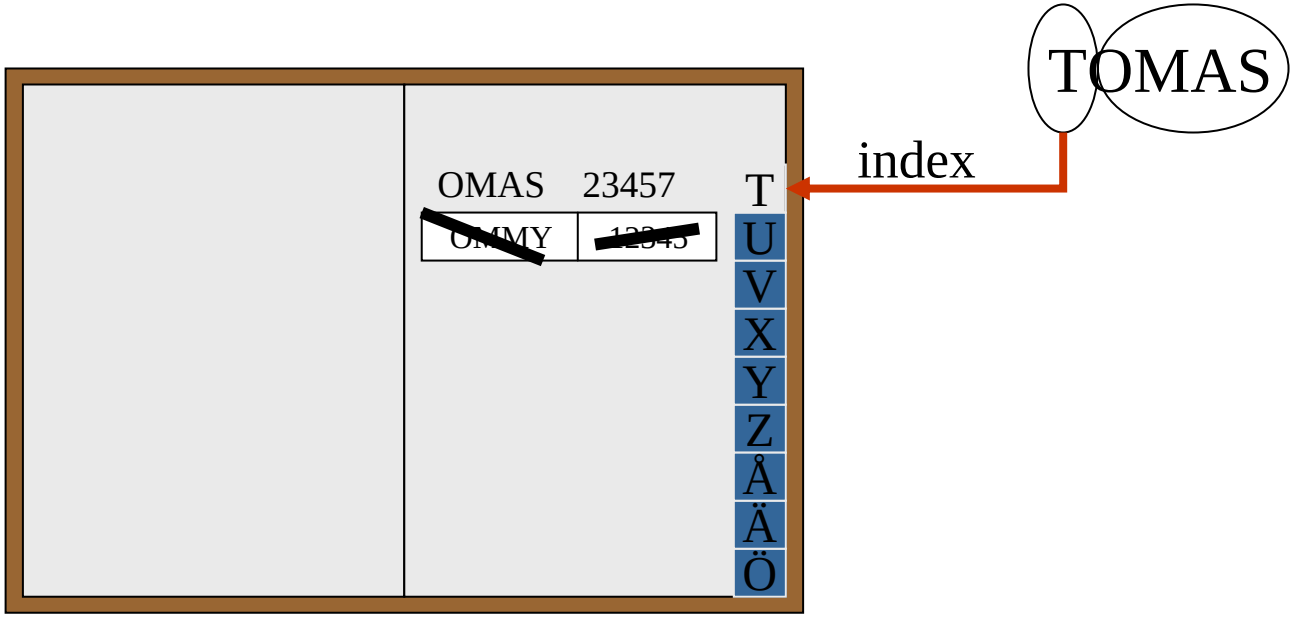
Looking for Tomas' Number



Miss!
Lookup Tomas' number in the telephone directory

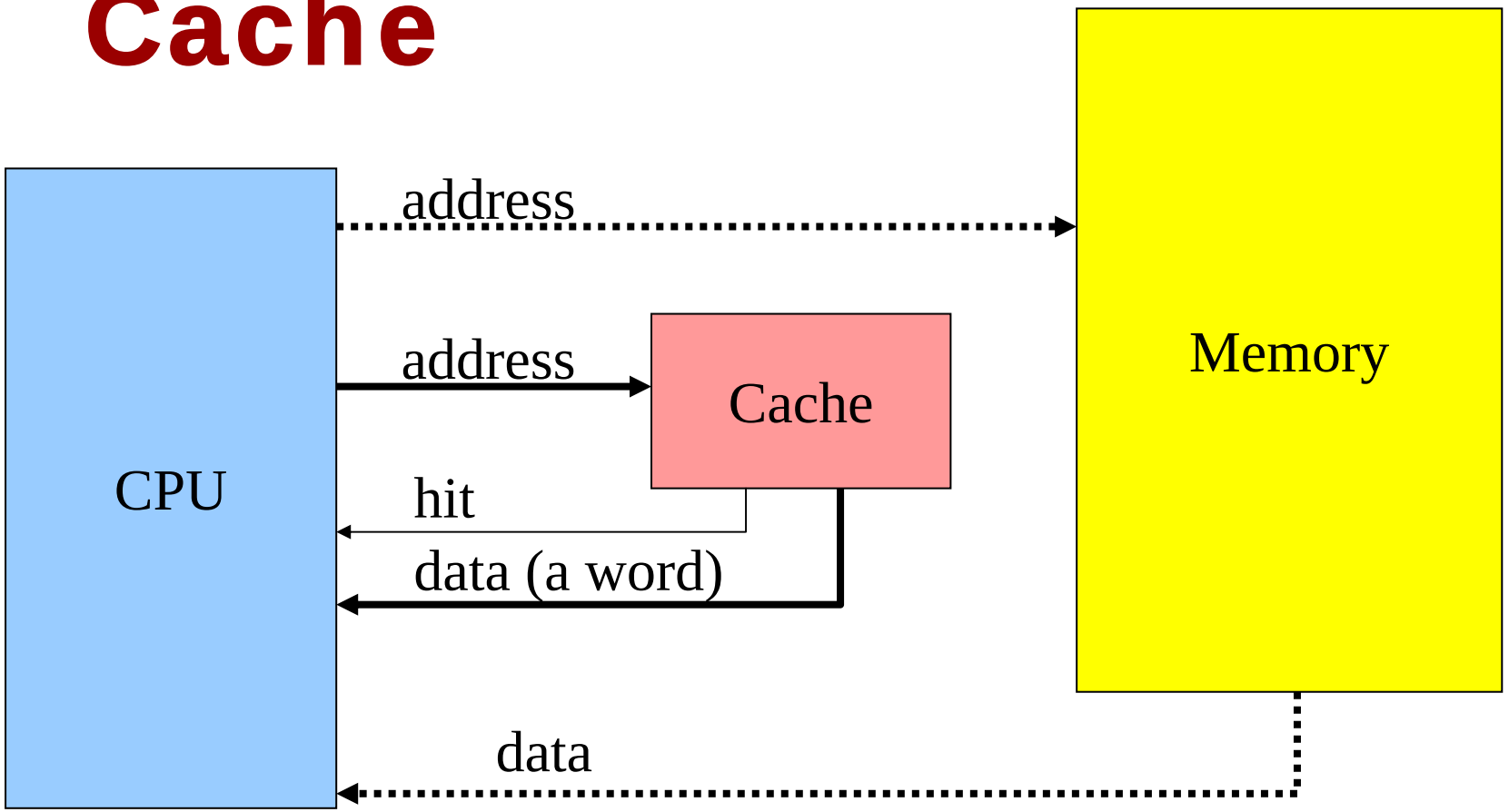
Address Book Cache

Looking for Tomas' Number

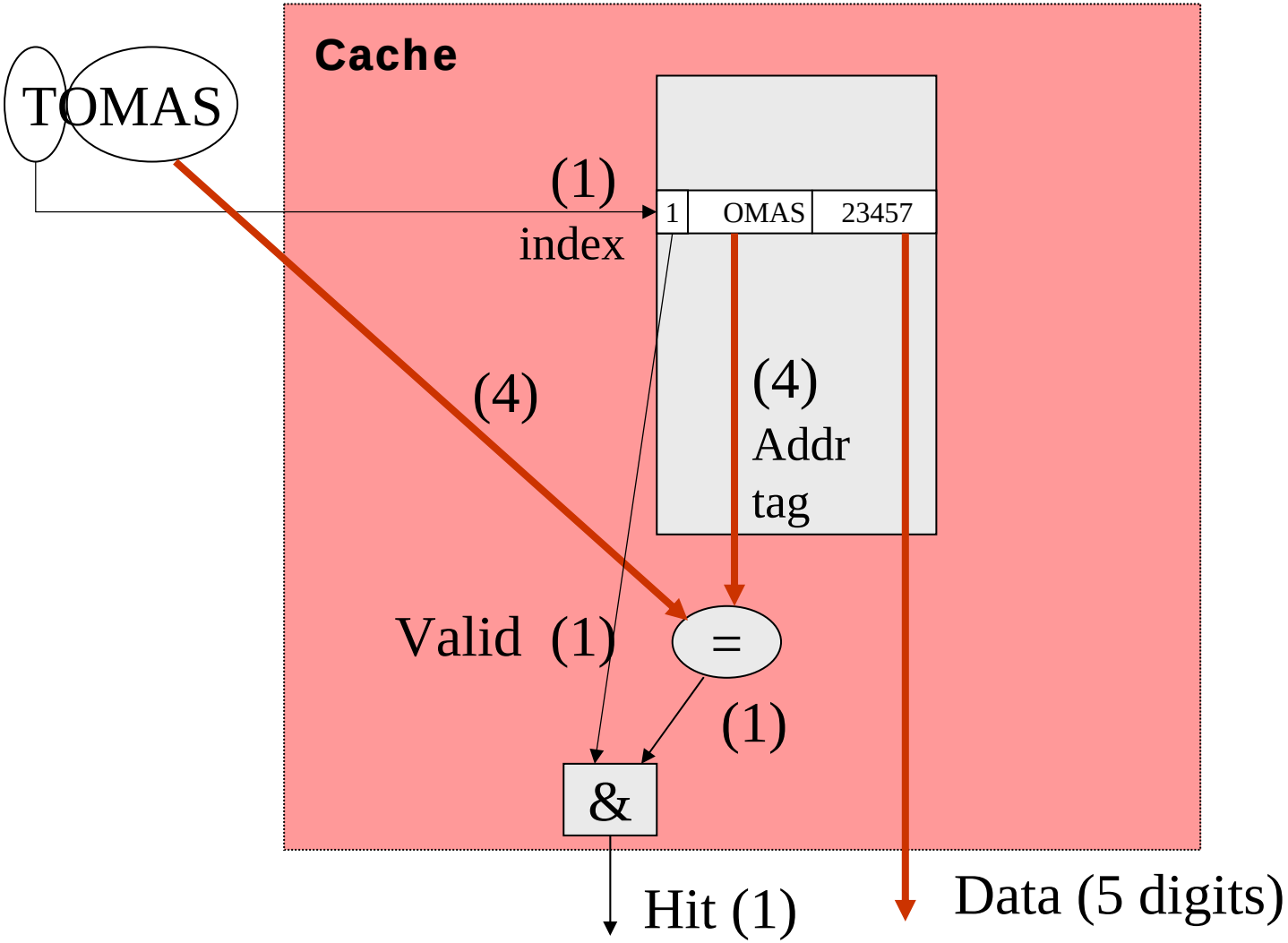


Replace TOMMY's data with TOMAS' data.
(Only one person per page = direct mapped cache)

Cache

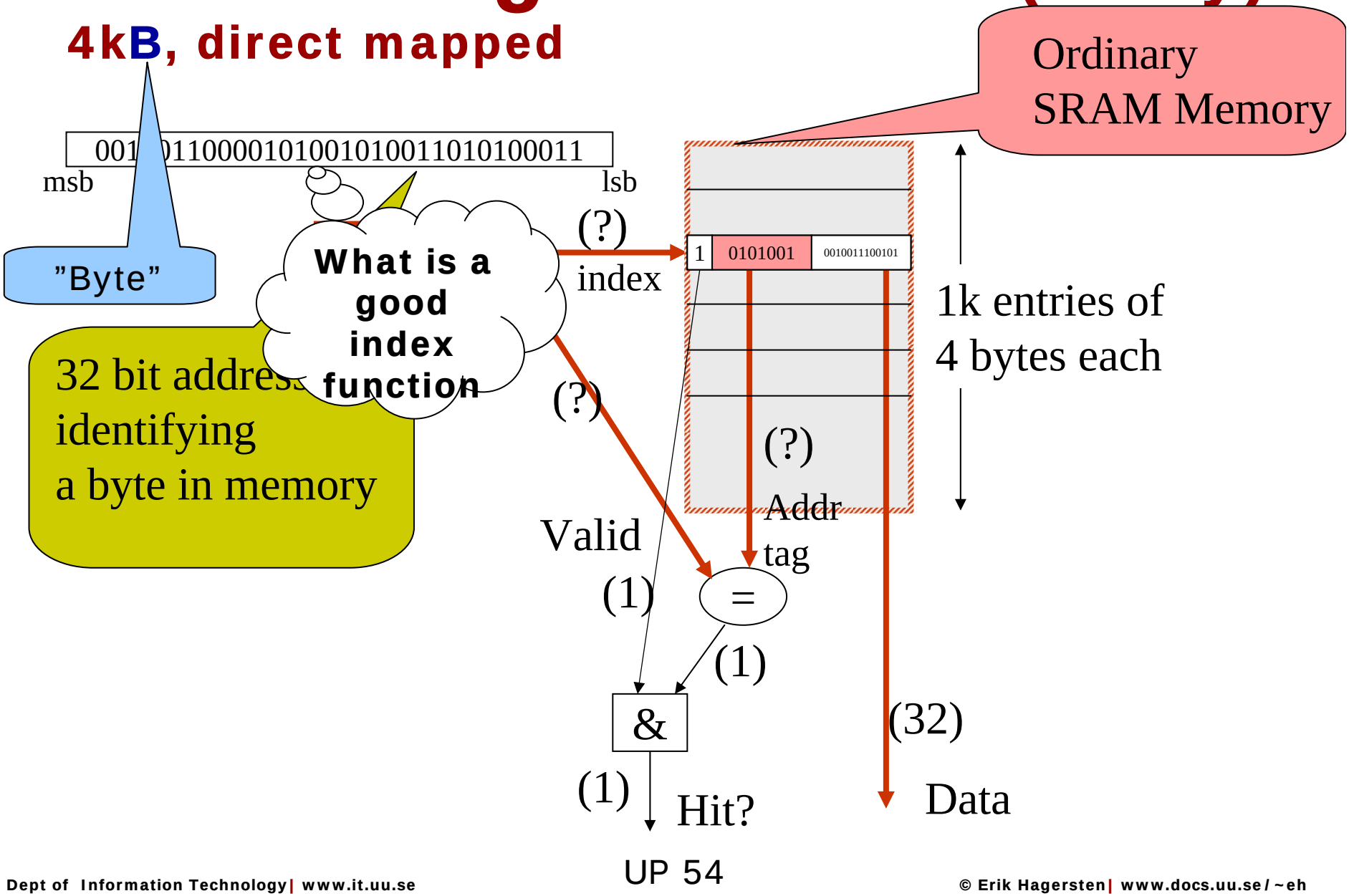


Cache Organization

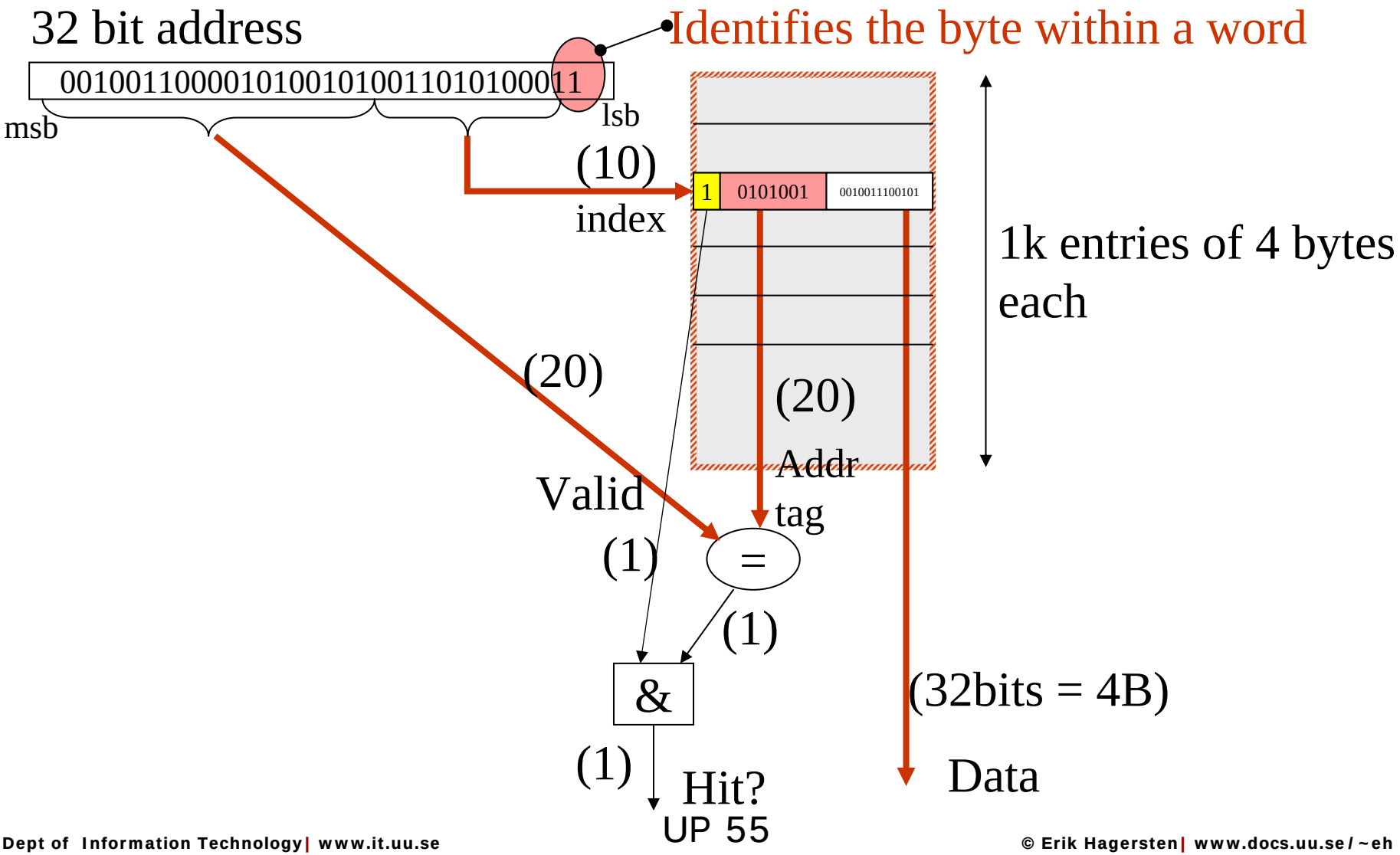


Cache Organization (really)

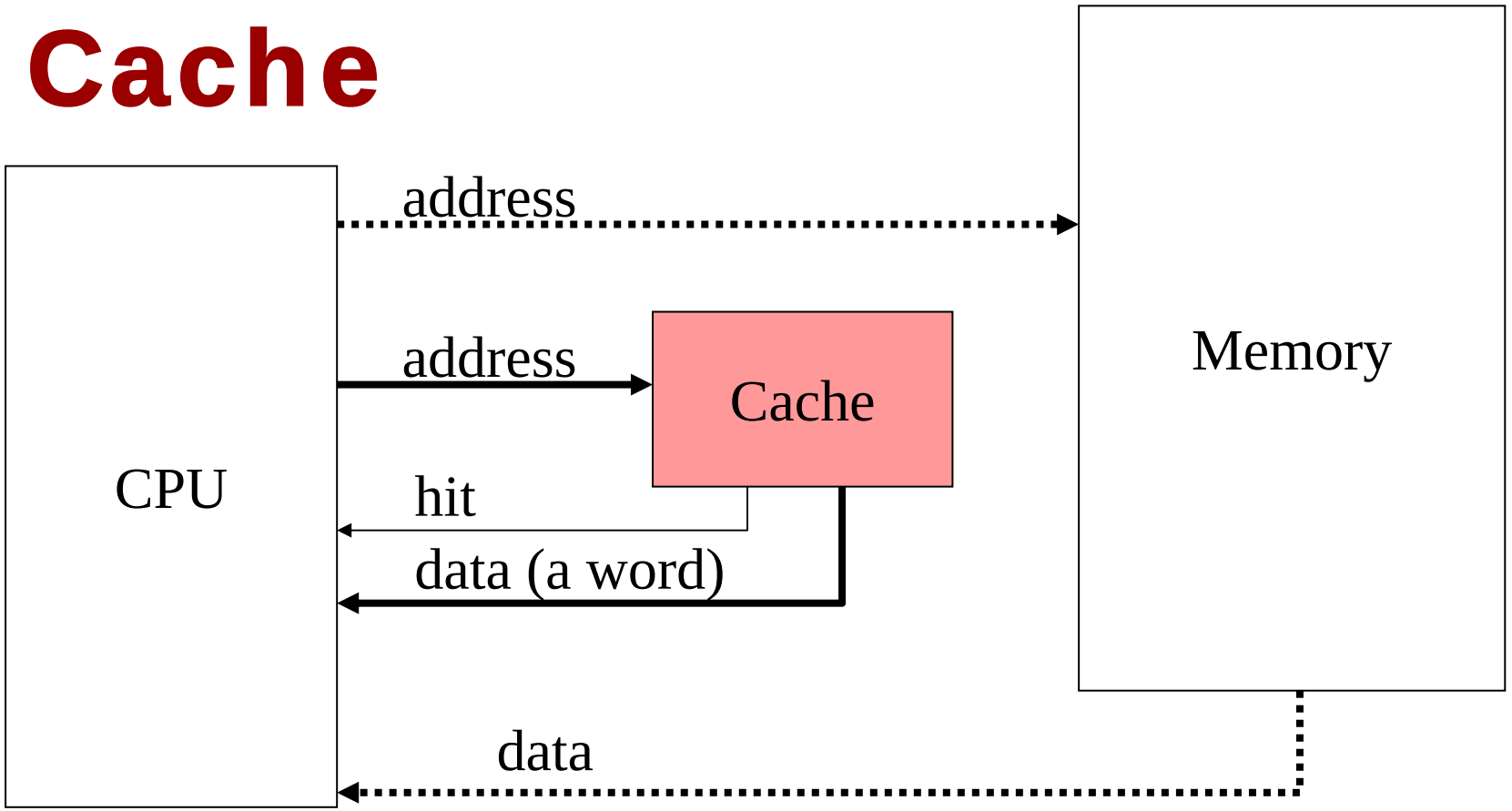
4kB, direct mapped



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

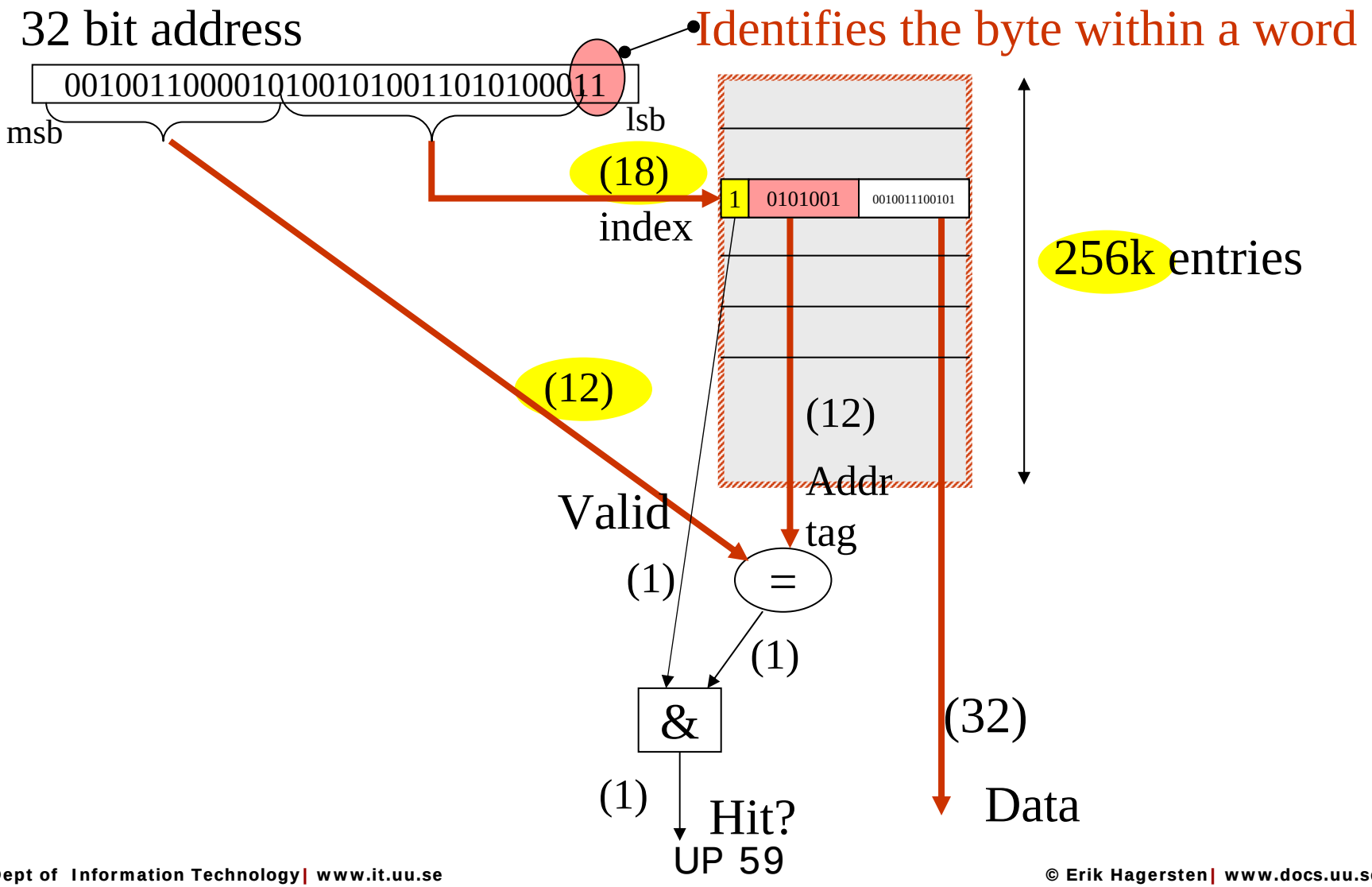
- Mark Hill's three "Cs"
  - Compulsory miss (touching data for the first time)
  - Capacity miss (the cache is too small)
  - Conflict misses (non-optimal cache implementation)
- (Multiprocessors)
  - Communication (imposed by coherence)
  - False sharing (side-effect from large cache blocks)

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)

Cache Organization
1MB, direct mapped



UPPSALA UNIVERSITET

Pros/Cons Large Caches

++ The safest way to get improved hit rate

-- SRAMs are very expensive!!

-- Larger size ==> slower speed

more load on "signals"

longer distances

-- (power consumption)

-- (reliability)

UPPSALA UNIVERSITET

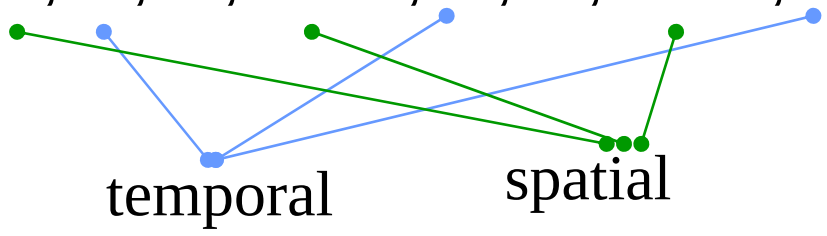
Why do you hit in a cache?

- Temporal locality
  - Likely to access the same data again soon
- Spatial locality
  - Likely to access nearby data again soon

Typical access pattern:

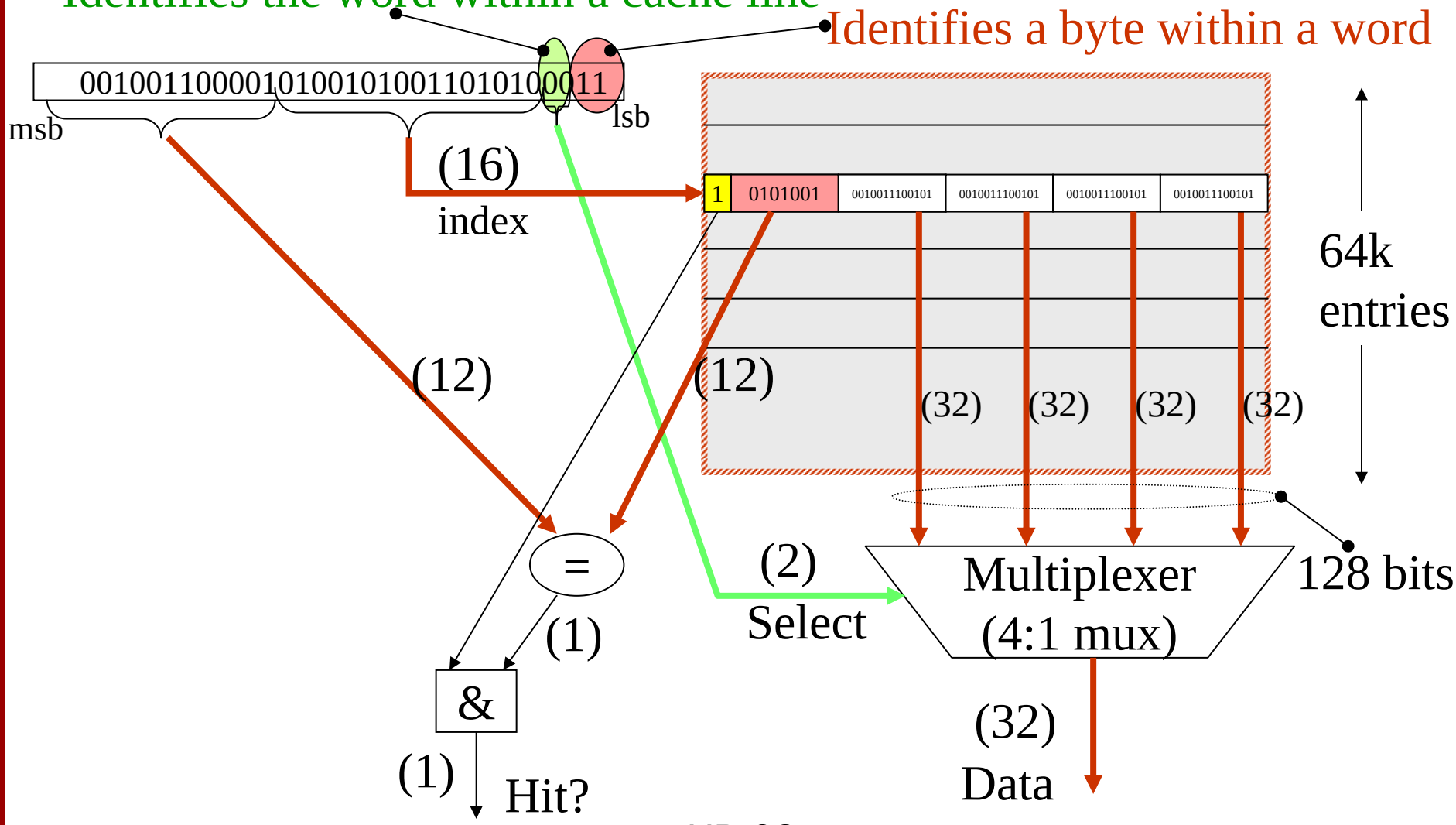
(inner loop stepping through an array)

A, B, C, A+1, B, C, A+2, B, C, ...



Fetch more than a word: cache blocks

1MB, direct mapped, Cache Line (aka Cache Block) = 16B



UPPSALA UNIVERSITET

Pros/Cons Large Cache Lines

++ Explores spatial locality

++ Fits well with modern DRAMs

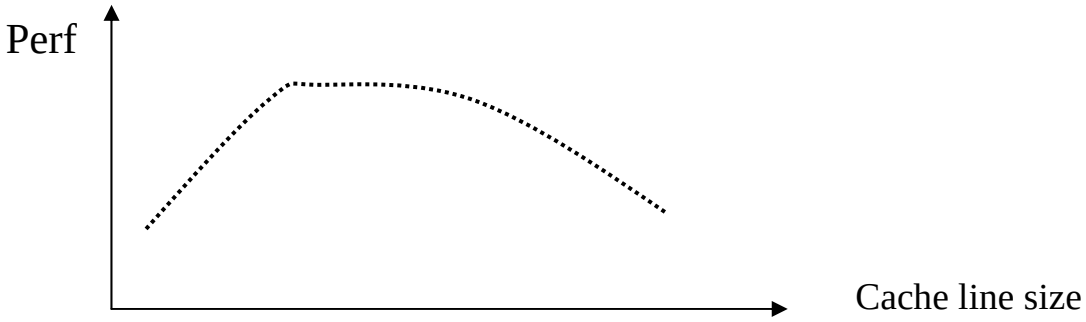
* first DRAM access slow

* subsequent accesses fast (“page mode”)

-- Poor usage of SRAM & BW for some patterns

-- Higher miss penalty (fix: critical word first)

-- (False sharing in multiprocessors)



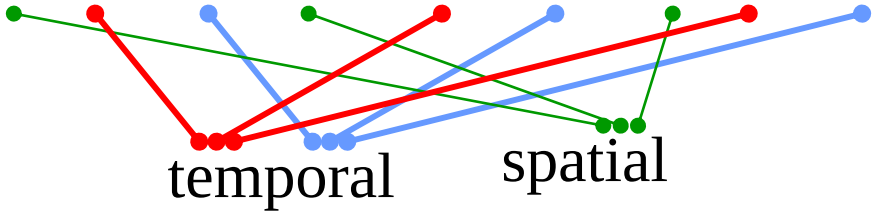
UPPSALA UNIVERSITET

Cache Conflicts

Typical access pattern:

(inner loop stepping through an array)

A, B, C, A+1, B, C, A+2, B, C, ...

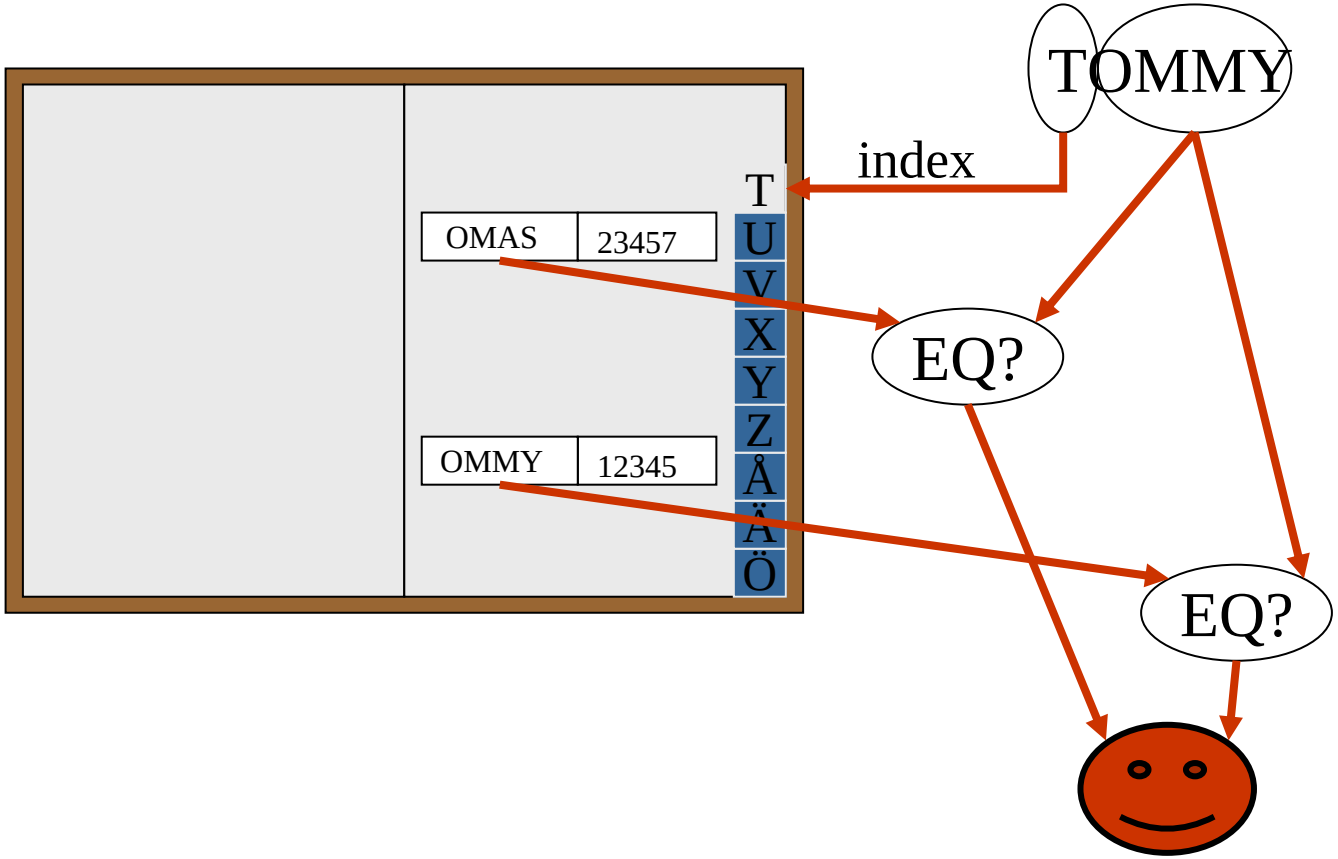


What if B and C index to the same cache location

- Conflict misses -- big time!
- Potential performance loss 10-100x

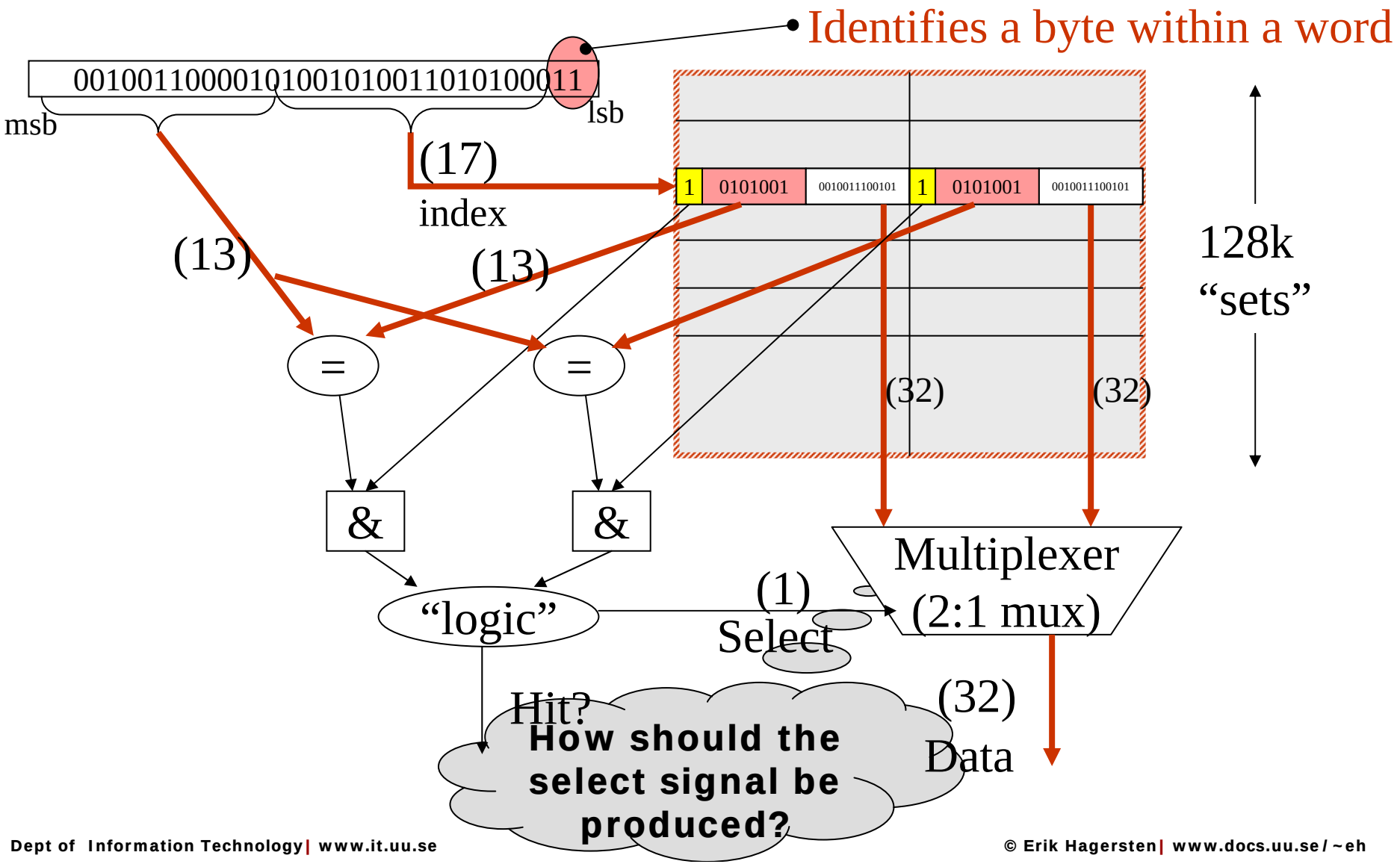
Address Book Cache

Two names per page: First index -- then search.



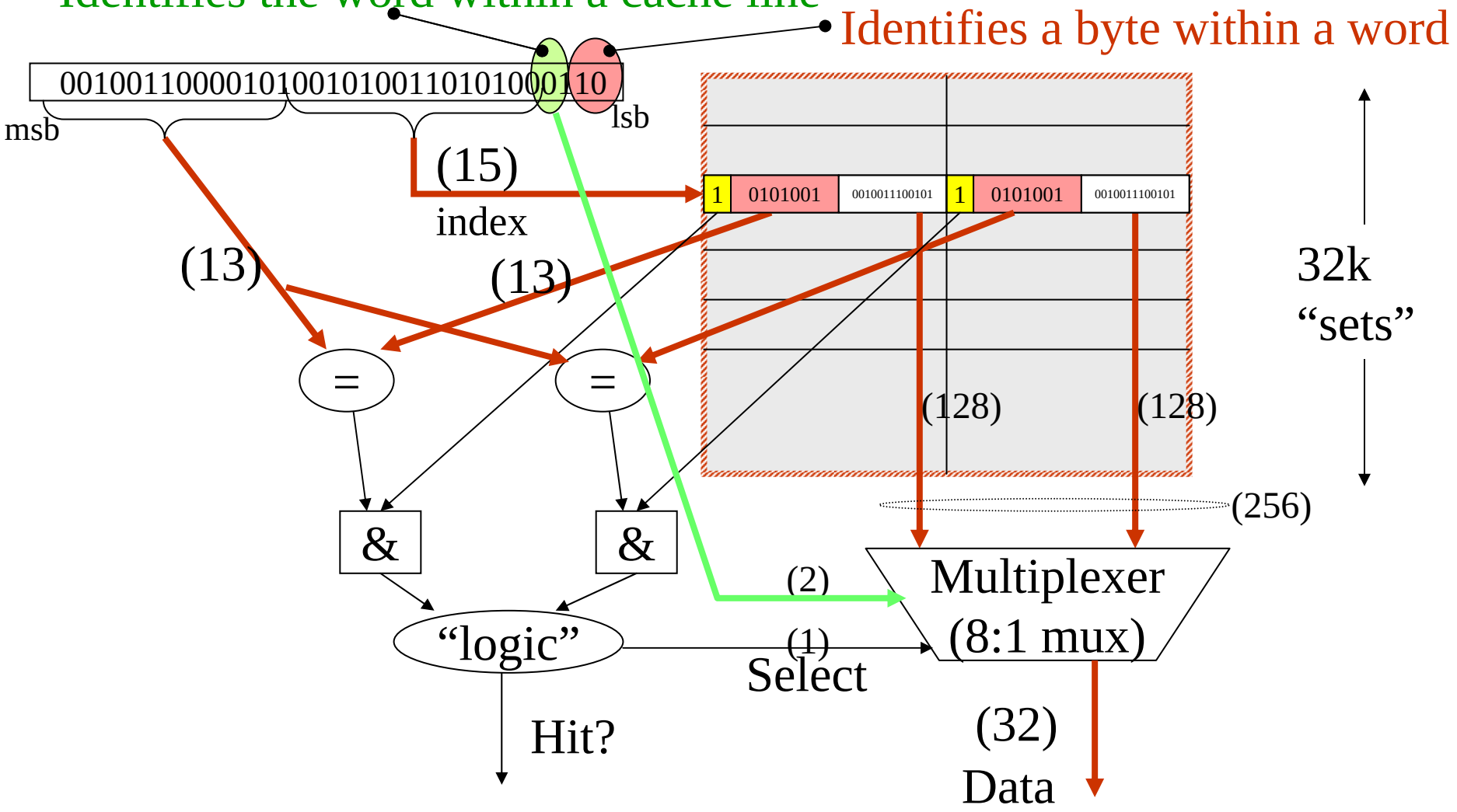
Avoiding conflict: More associativity

1MB, 2-way, CL=4B



A combination thereof

1MB, 2-way, CL=16B



UPPSALA UNIVERSITET

Pros/Cons Associativity

++ Avoids conflict misses

-- Slower access time

-- More complex implementation
comparators, muxes, ...

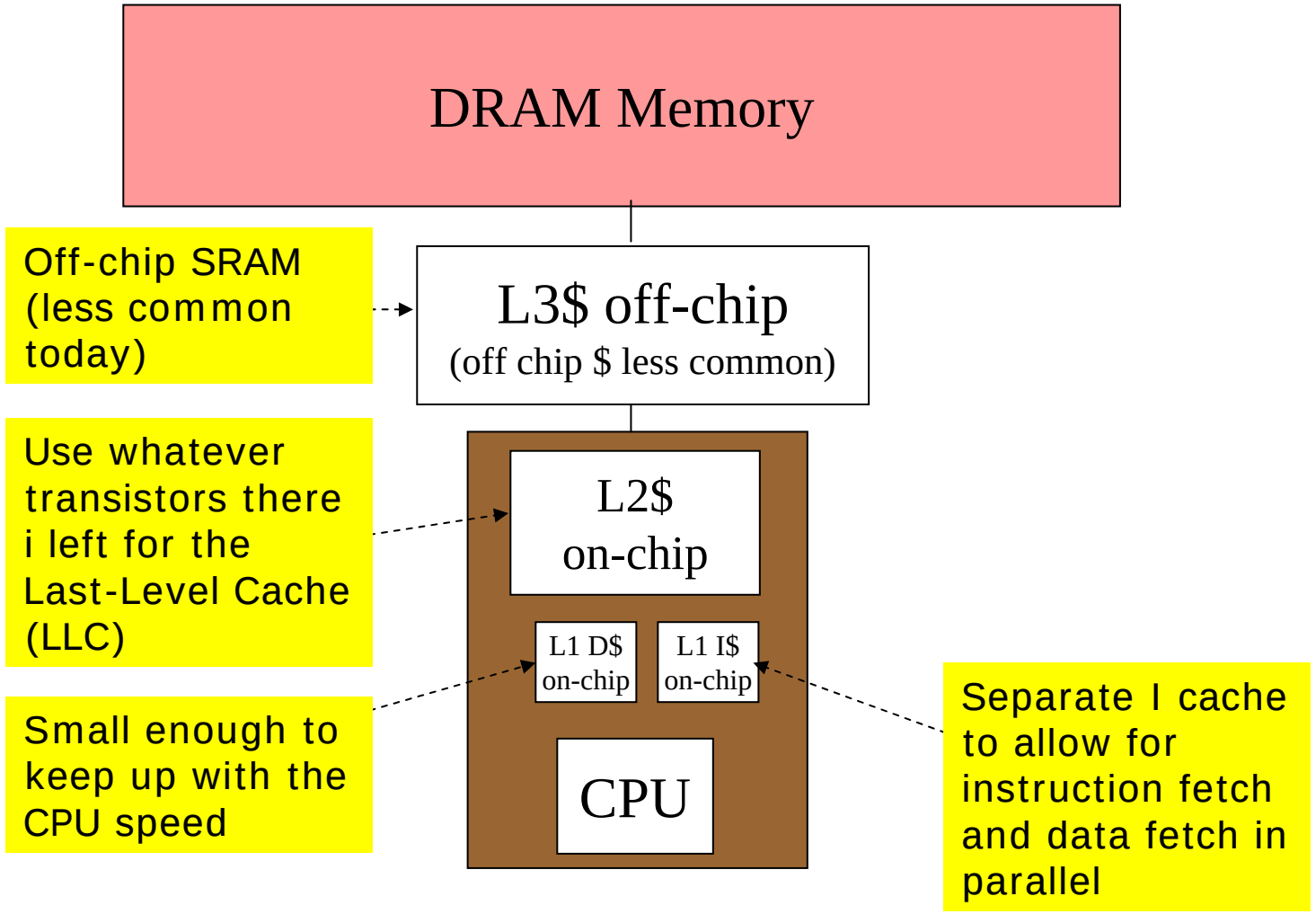
-- Requires more pins (for external SRAM...)

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...

Cache Hierarchy of Today



UPPSALA UNIVERSITET

Take-away message: Caches

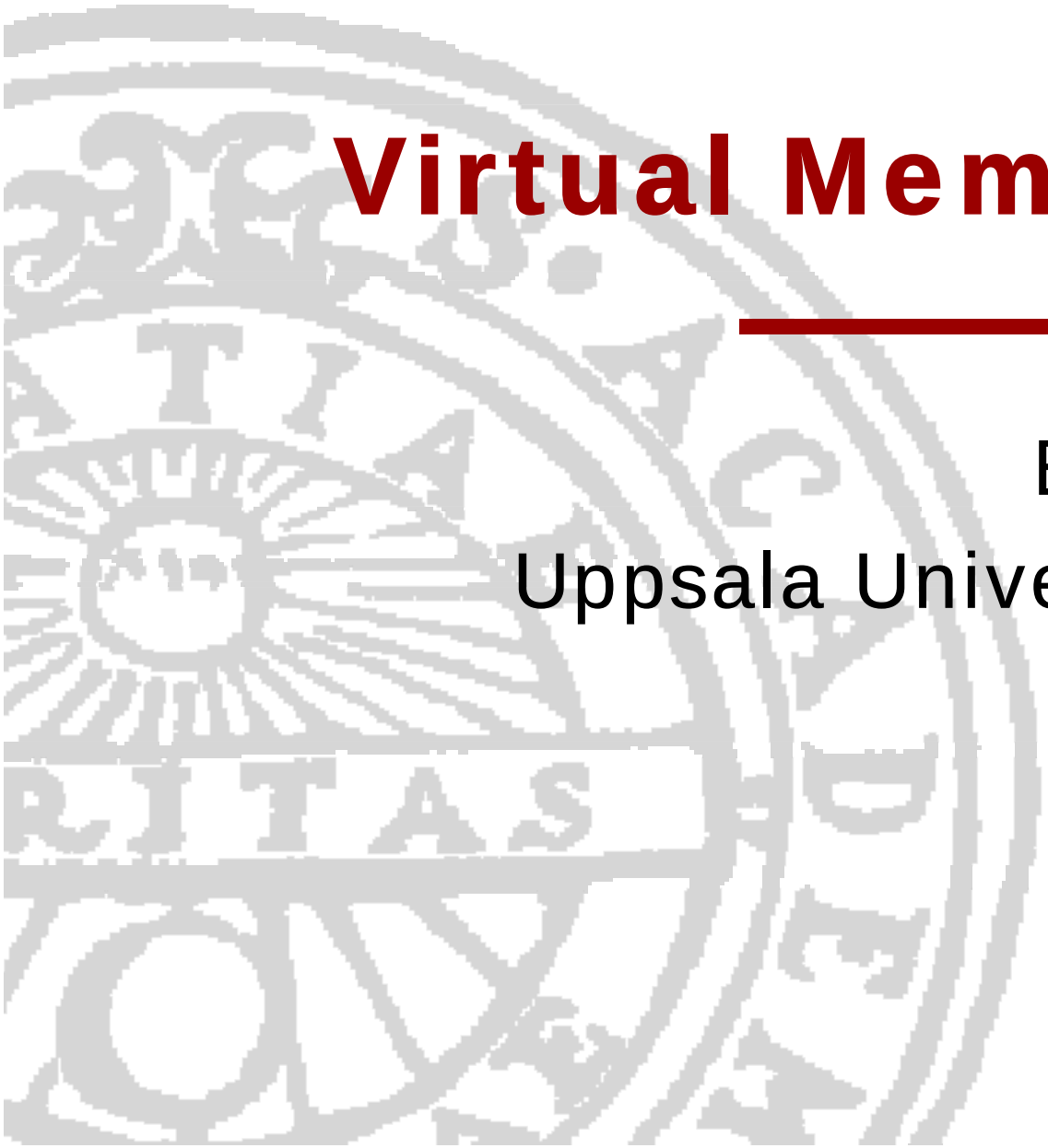
- Cache are fast but small
- Cache space is cache-line chunks (~64bytes)
- LSB part of the address is used to find the ”set” (aka, indexing)
- There is a limited number of cache lines per set (associativity)
- Typically, several levels of caches

- The most important target for optimizations

UPPSALA UNIVERSITET

How are we doing?

- Creating and exploring:
  1) Locality
     a) Spatial locality
     b) Temporal locality
     c) Geographical locality
  2) Parallelism
     a) Instruction level
     b) Loop level
     c) Thread level



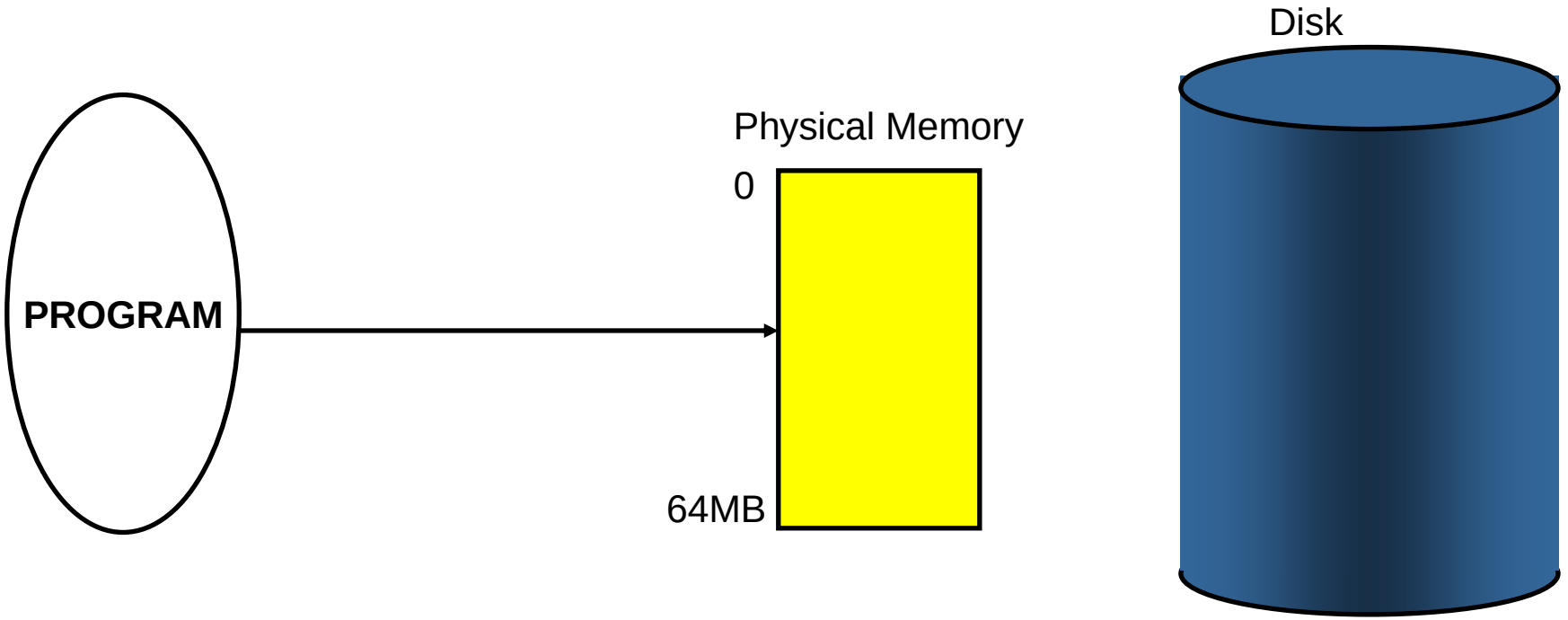
Virtual Memory System

Erik Hagersten

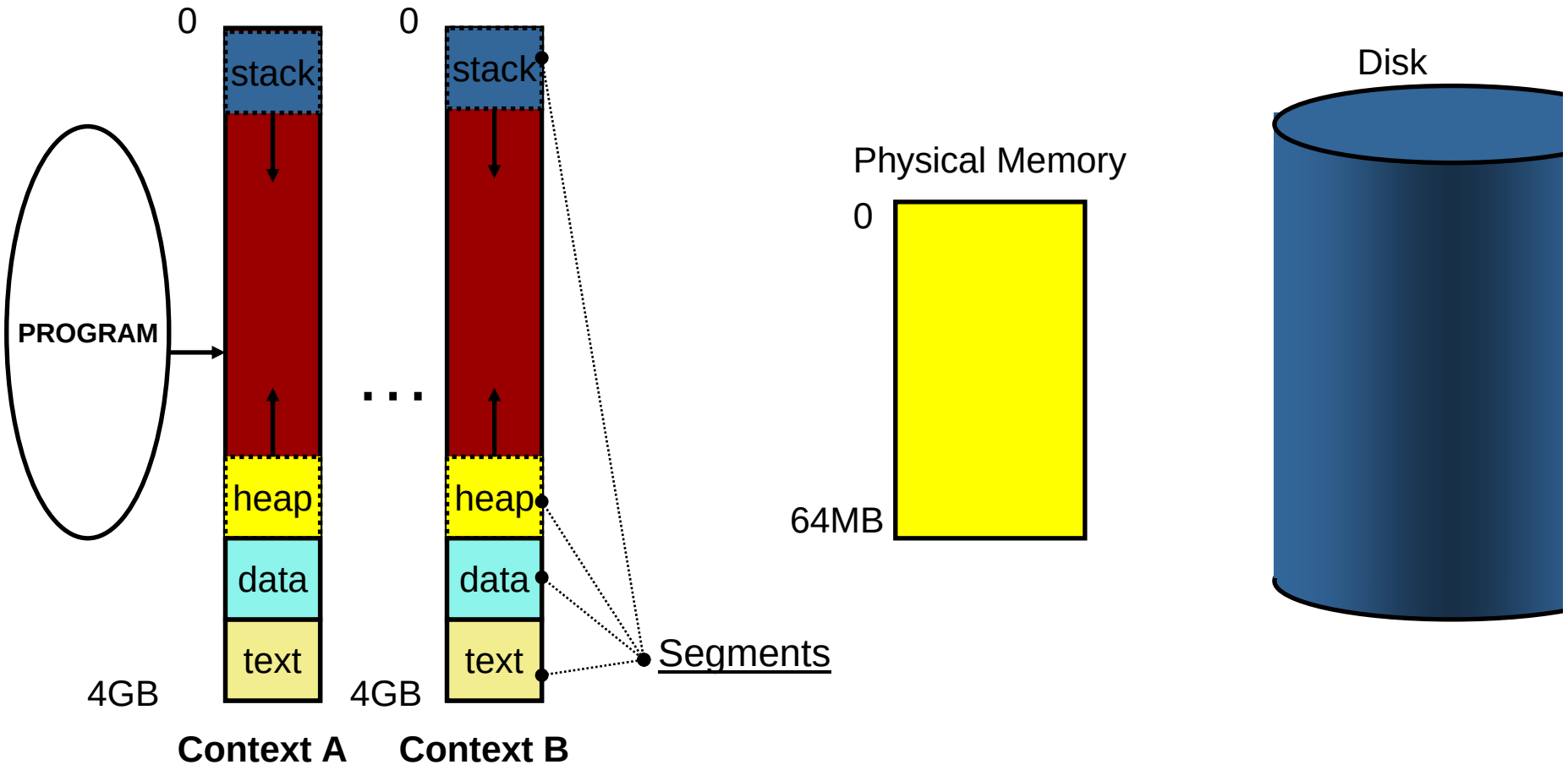
Uppsala University, Sweden

eh@it.uu.se

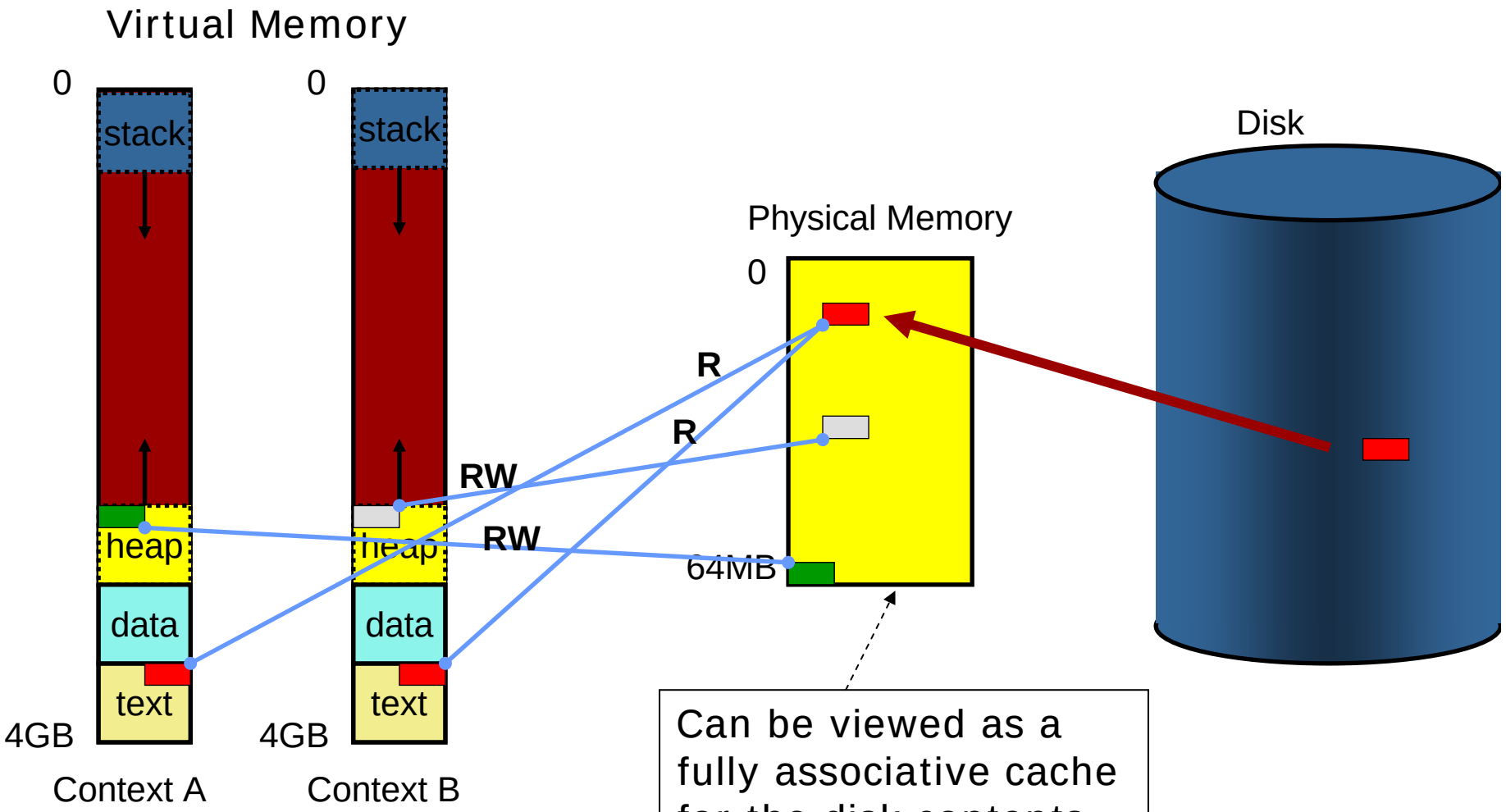
Physical Memory



Virtual and Physical Memory



Translation & Protection

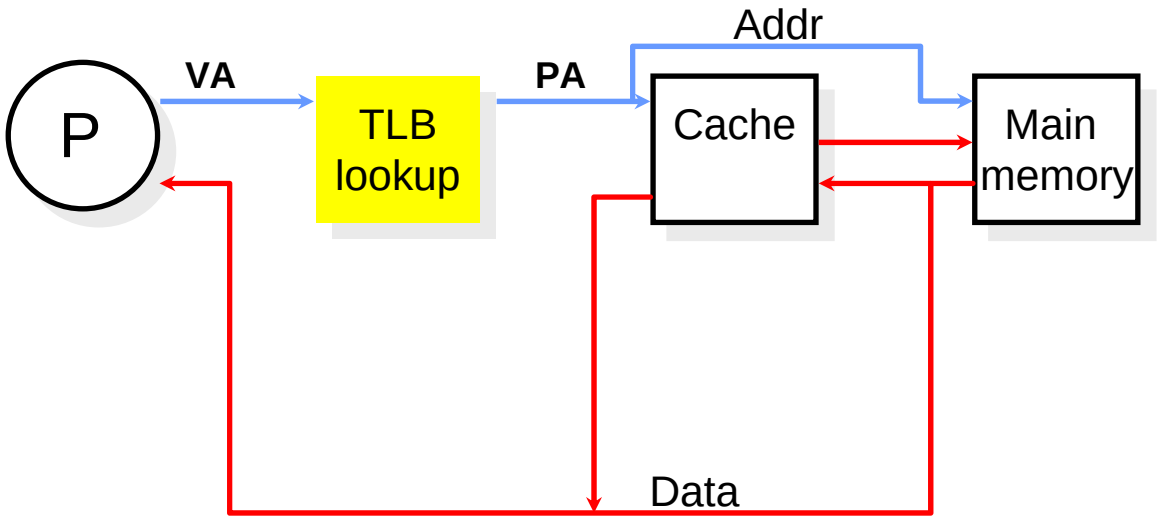


Fast address translation

How do we avoid three extra memory references for each original memory reference?

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

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



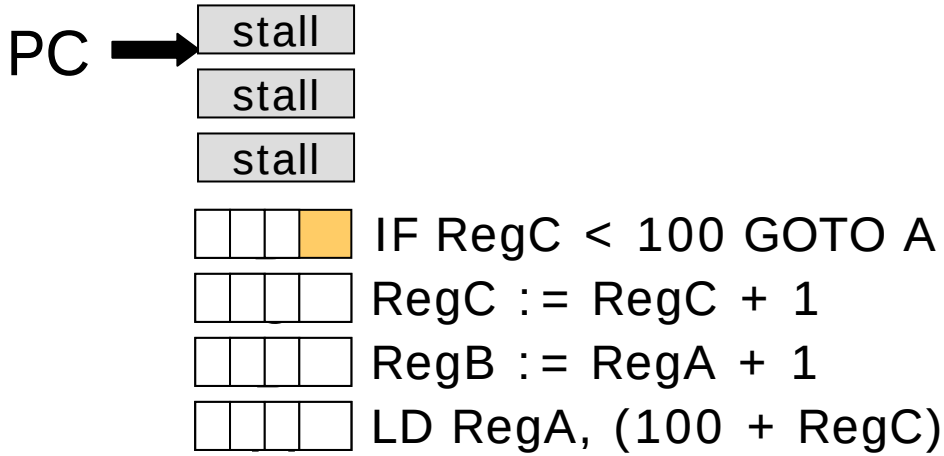
Branch prediction

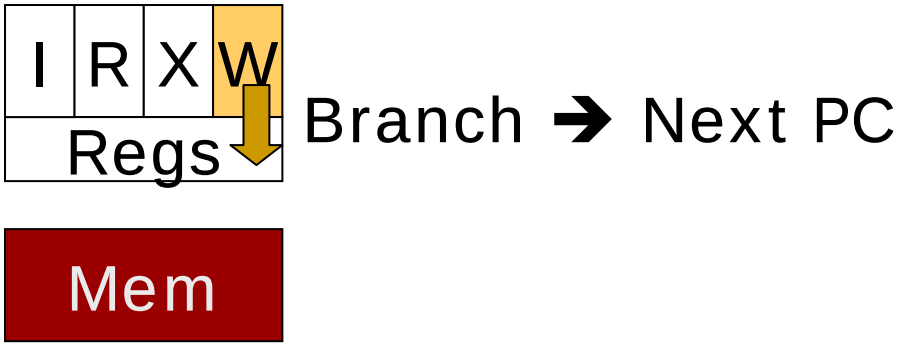
Erik Hagersten

Uppsala University, Sweden

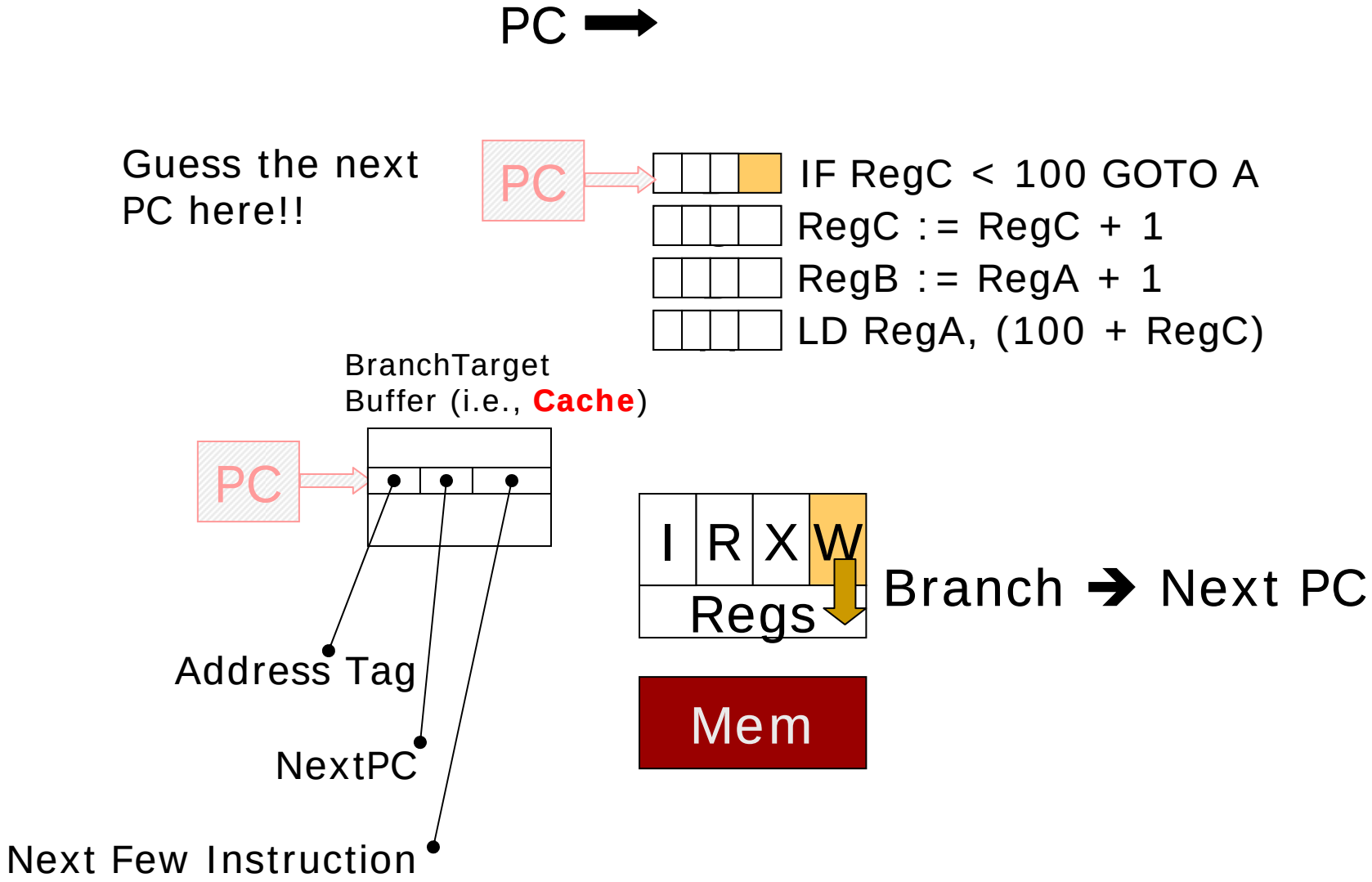
eh@it.uu.se

Speed up calculation of Next PC





Branch Predictor Based on History



UPPSALA UNIVERSITET

Caches Everywhere…

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