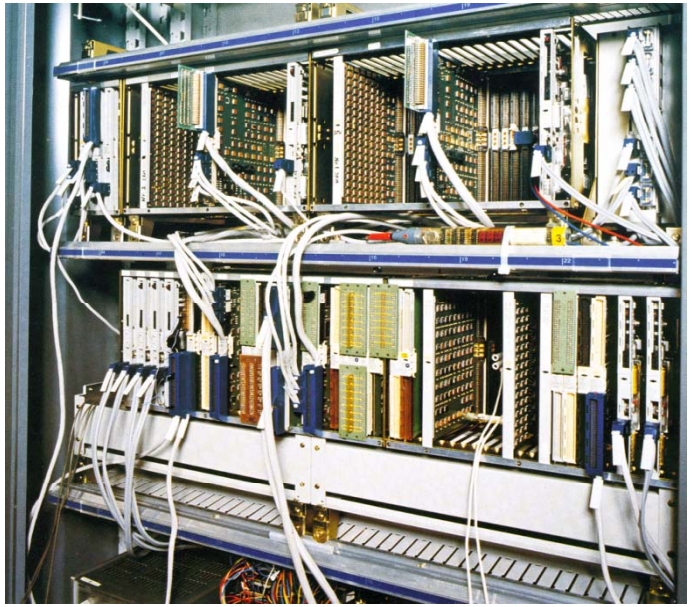
Introduction to Computer Architecture

Erik Hagersten
Uppsala University

UPPSALA UNIVERSITET

≈30 years ago: APZ 212 @ 5MHz

”the AXE supercomputer”



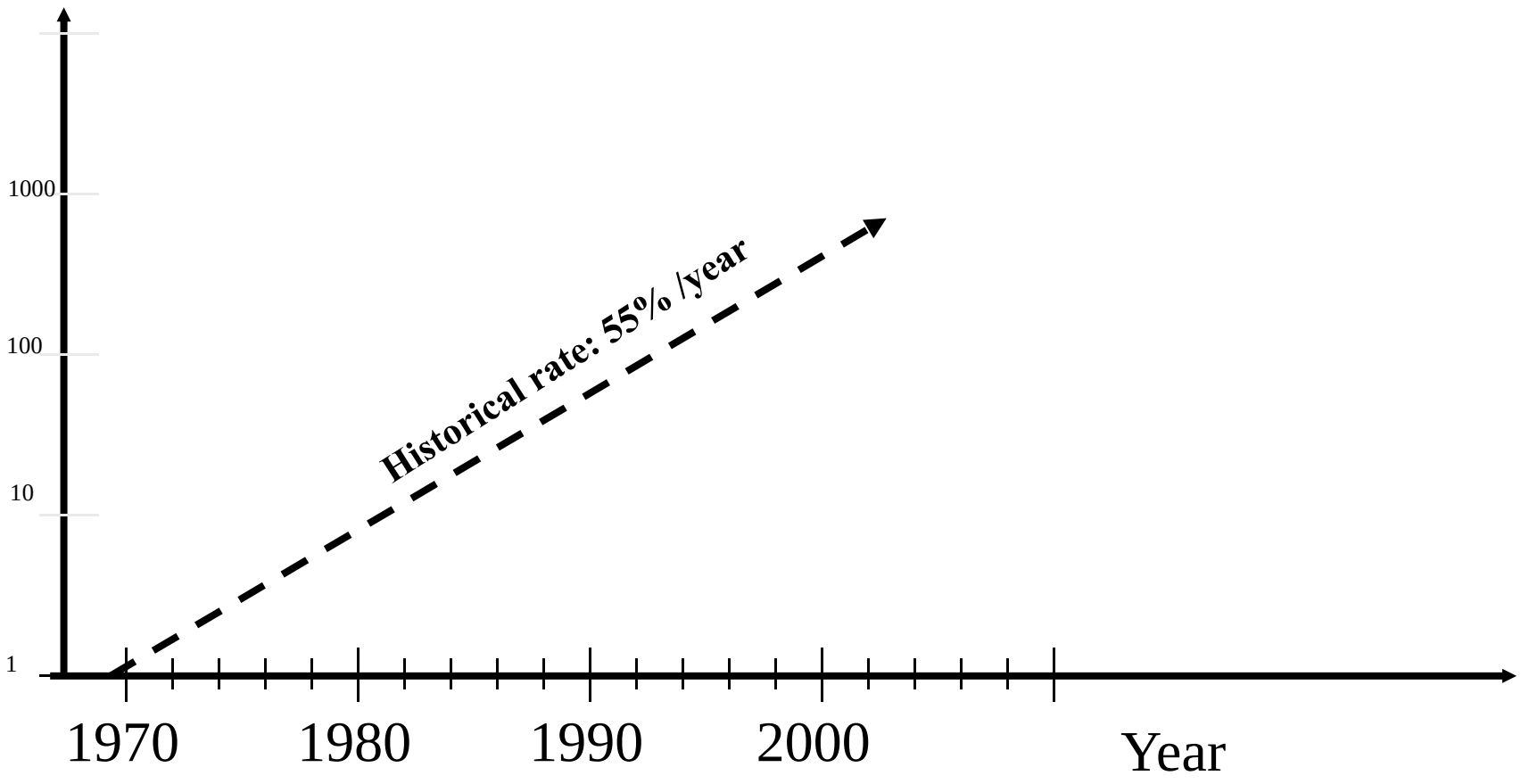
PDC Summer School 2011

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]

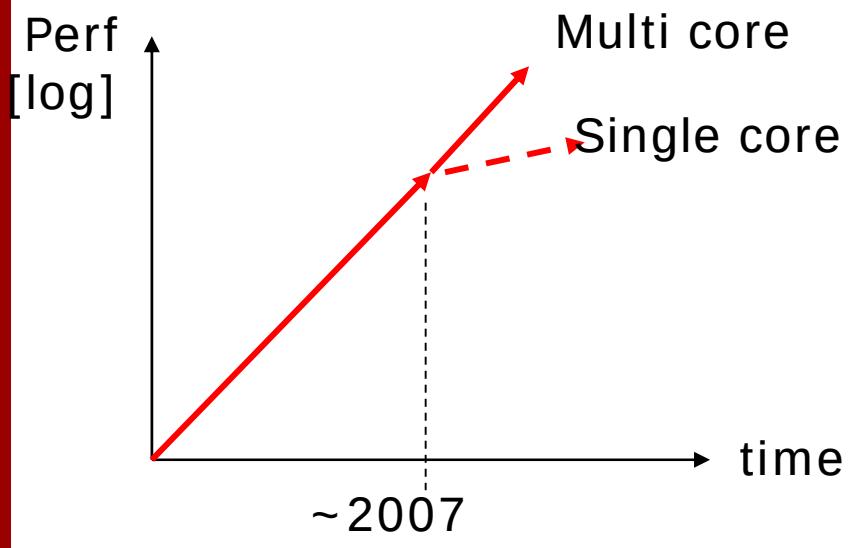


How to get efficient architectures…

Uncool today

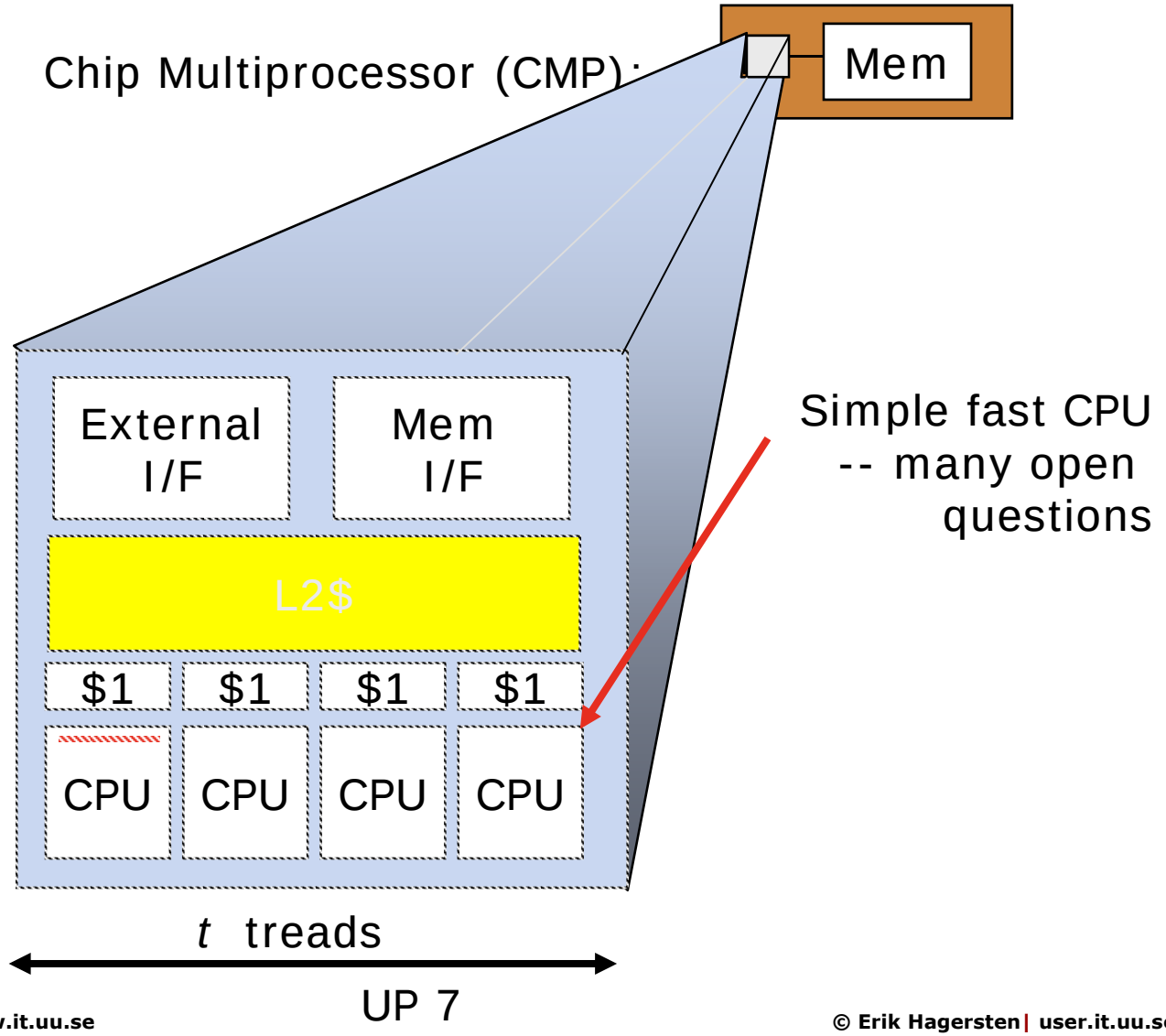
- ~~Increase clock rate~~
- Create and explore locality:
  a) Spatial locality
  b) Temporal locality
  c) Geographical locality
- Create and explore parallelism
  a) Instruction level parallelism (ILP)
  b) Thread level parallelism (TLP)
  c) Memory level parallelism (MLP)

Why Multicores?

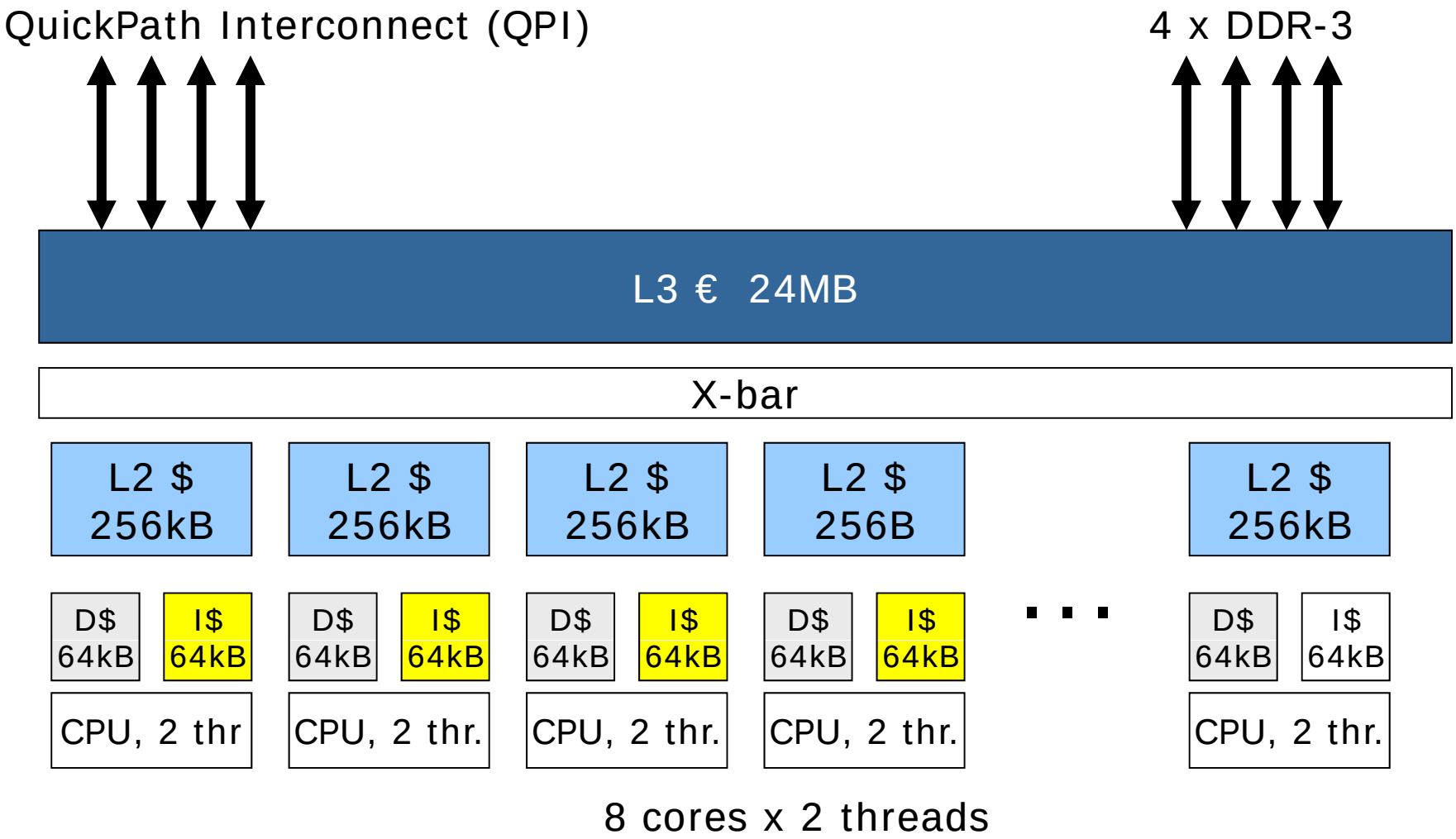


1. Not enough ILP & MLP in a single thread
2. Signal propagation delay » transistor delay
3. Power consumption

Chip Multiprocessor/ Multicores



Intel: ”Nehalem-Ex” (i7)



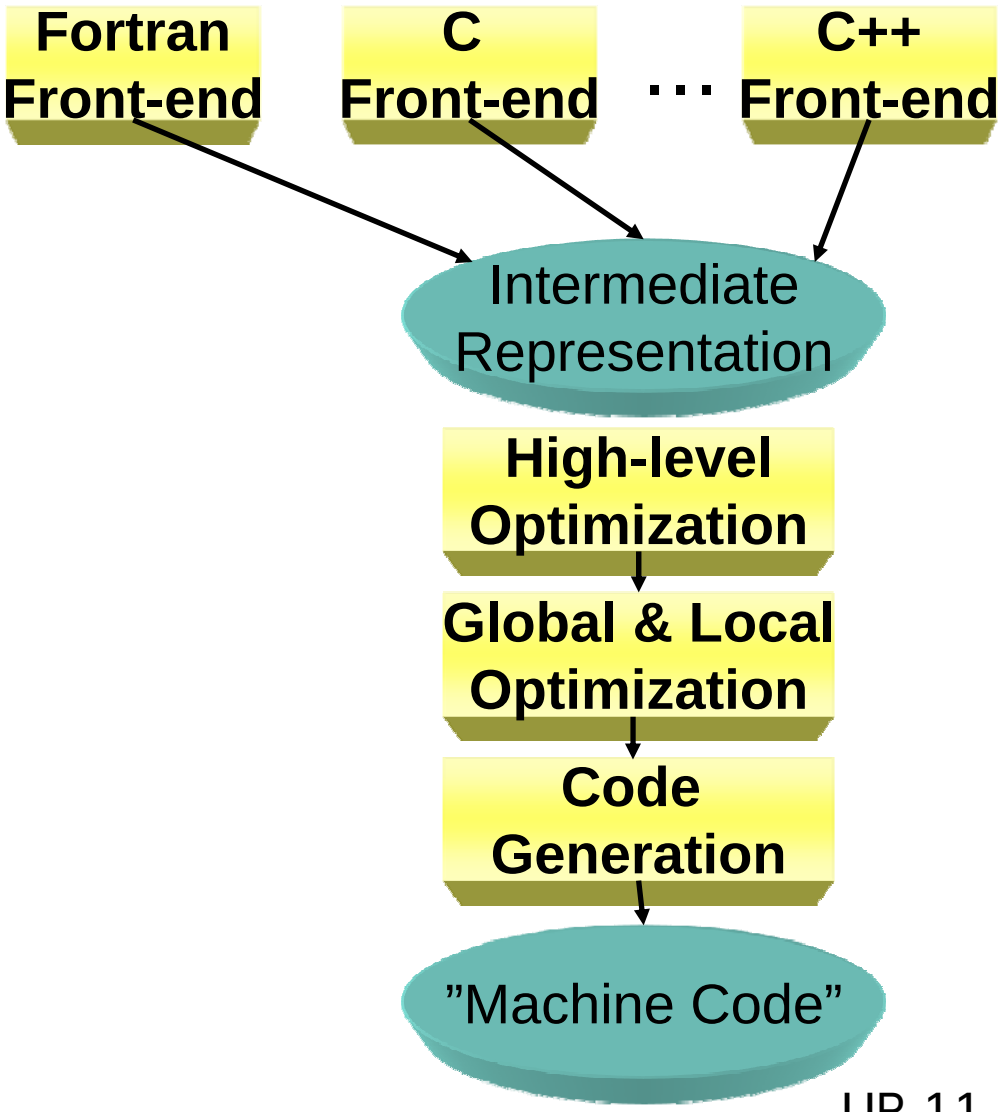
Outline of these lectures

1. Uniprocessors
   - CPUs & pipelines
   - Memory systems & caches
   - Disks
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

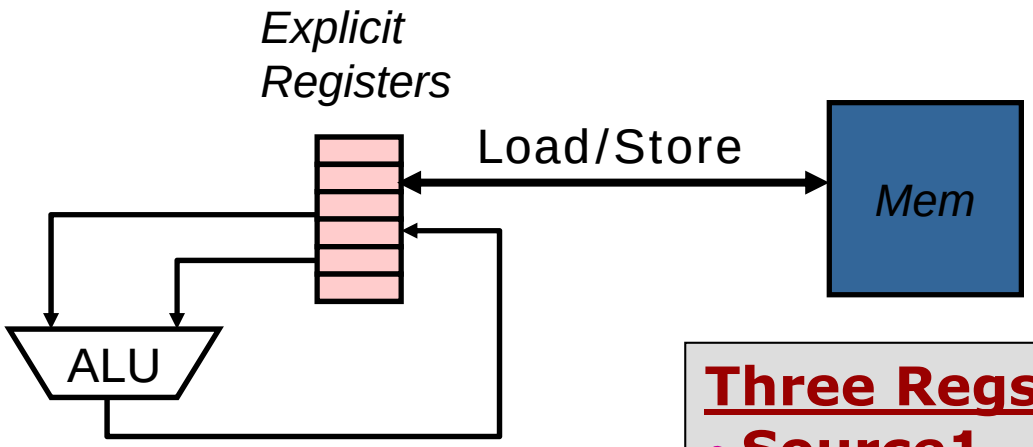
Register Allocation
Common sub-expressions

Instruction selection
constant folding

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

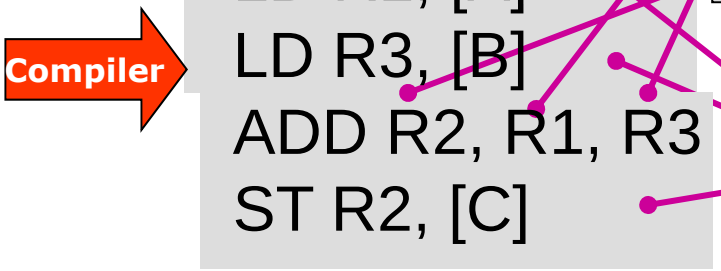
ALU ops: Reg -->Reg

Mem ops: Reg <--> Mem



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

Example: C = A + B

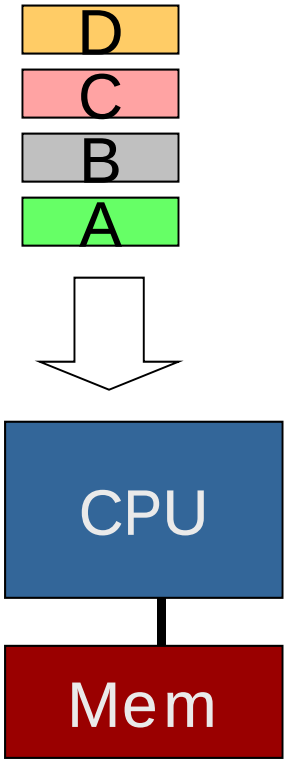



```
LD R1, [A]
LD R3, [B]
ADD R2, R1, R3
ST R2, [C]
```

Memory Accesses

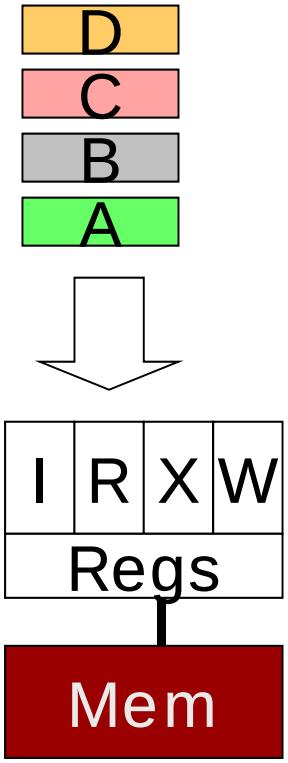
Lifting the CPU hood (simplified…)

Instructions:

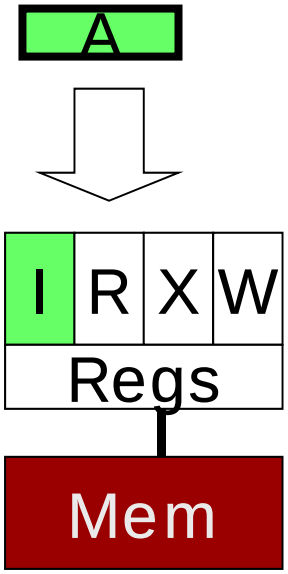


Pipeline

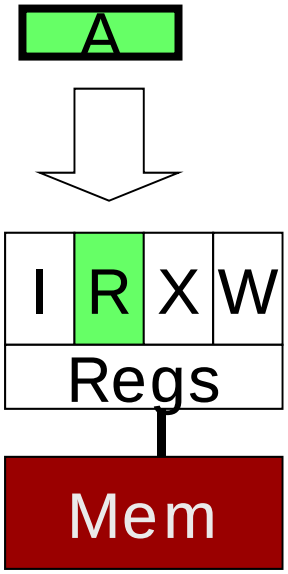
Instructions:



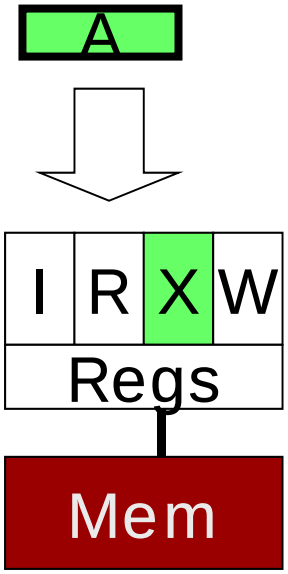
Pipeline



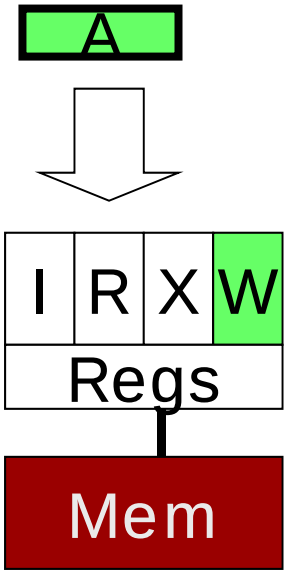
Pipeline



Pipeline



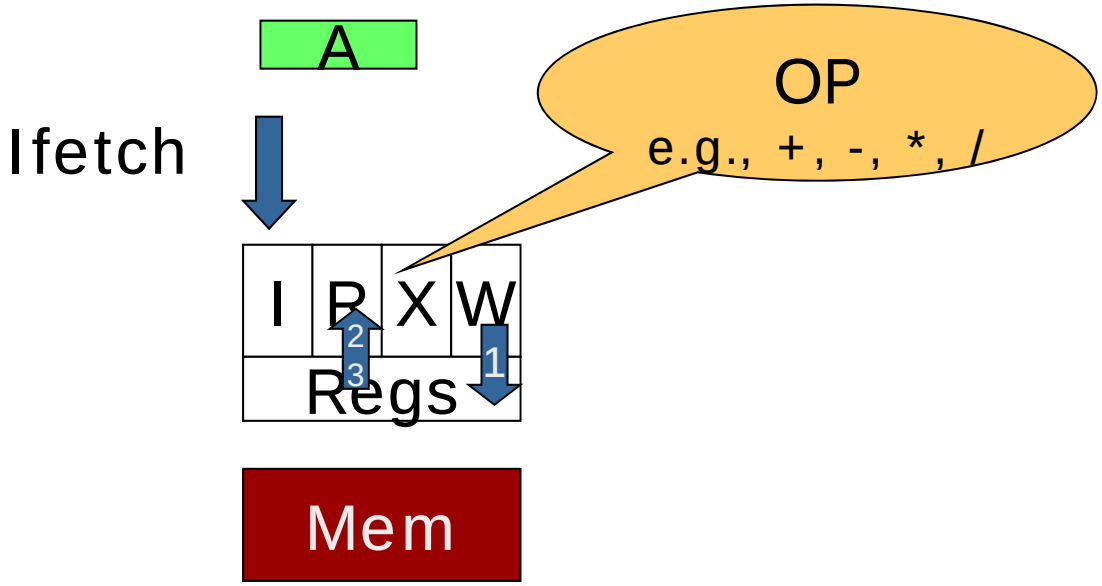
Pipeline:



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

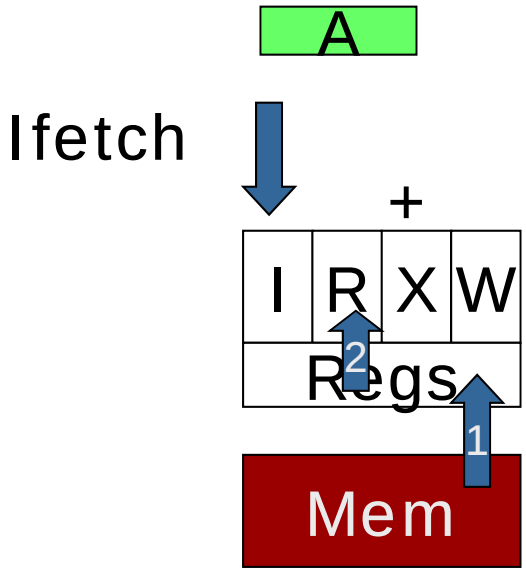
Register Operations [aka ALU operation]

R1 := R2 op R3



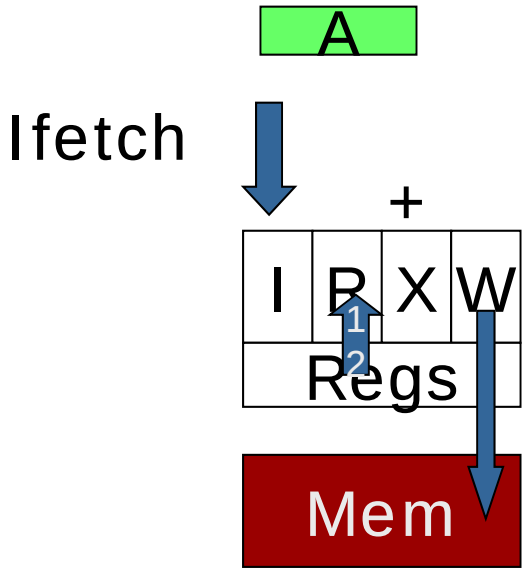
Load Operation:

LD R1, mem[cnst+R2]



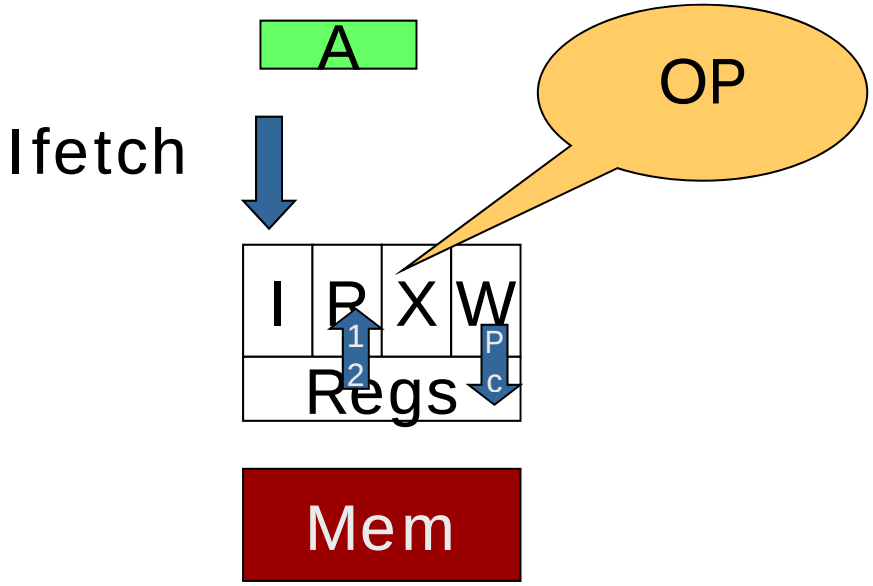
Store Operation:

ST R2, mem[cnst+R1]



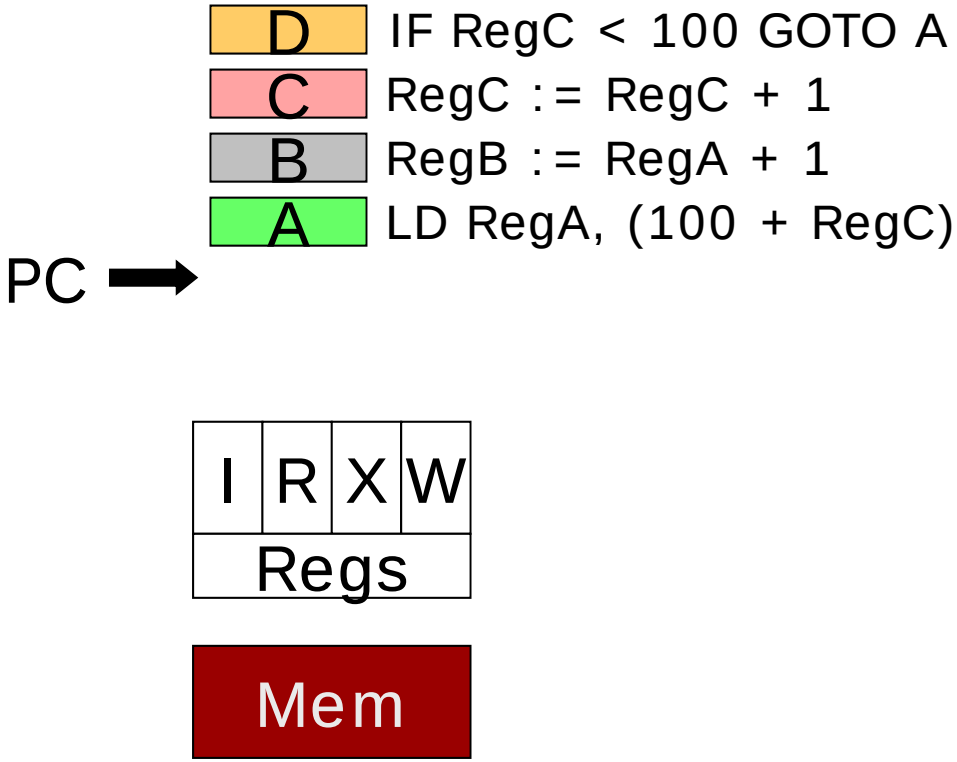
Branch Operations:

if (R1 Op Const) GOTO mem[R2]

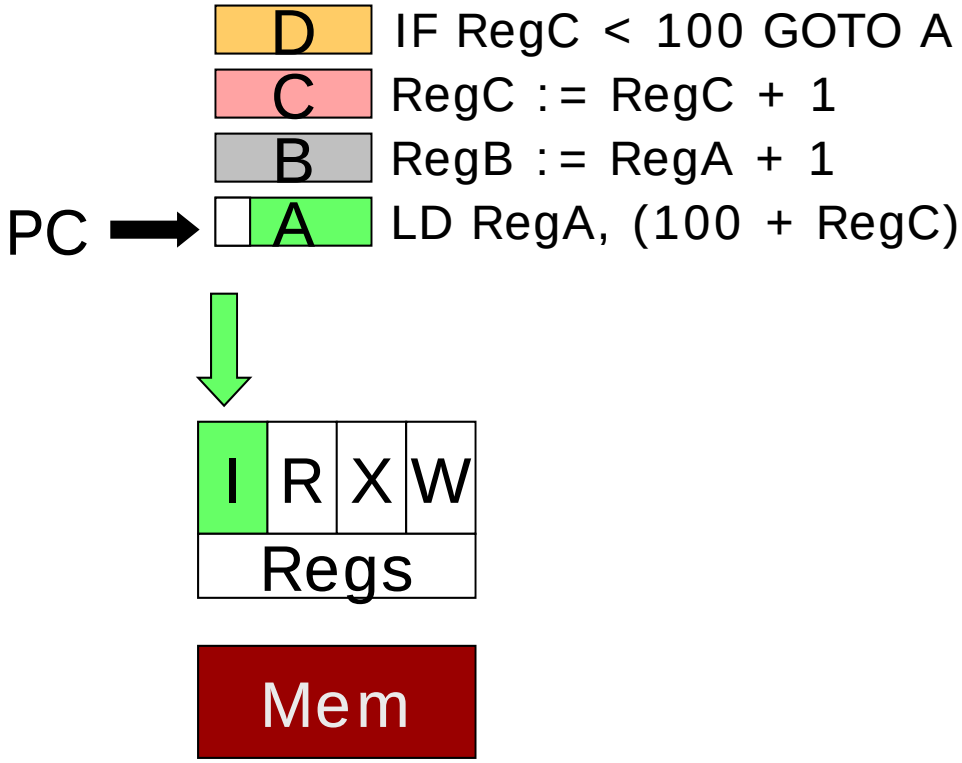


PC = Program Counter.
A special register pointing to the next instruction to execute

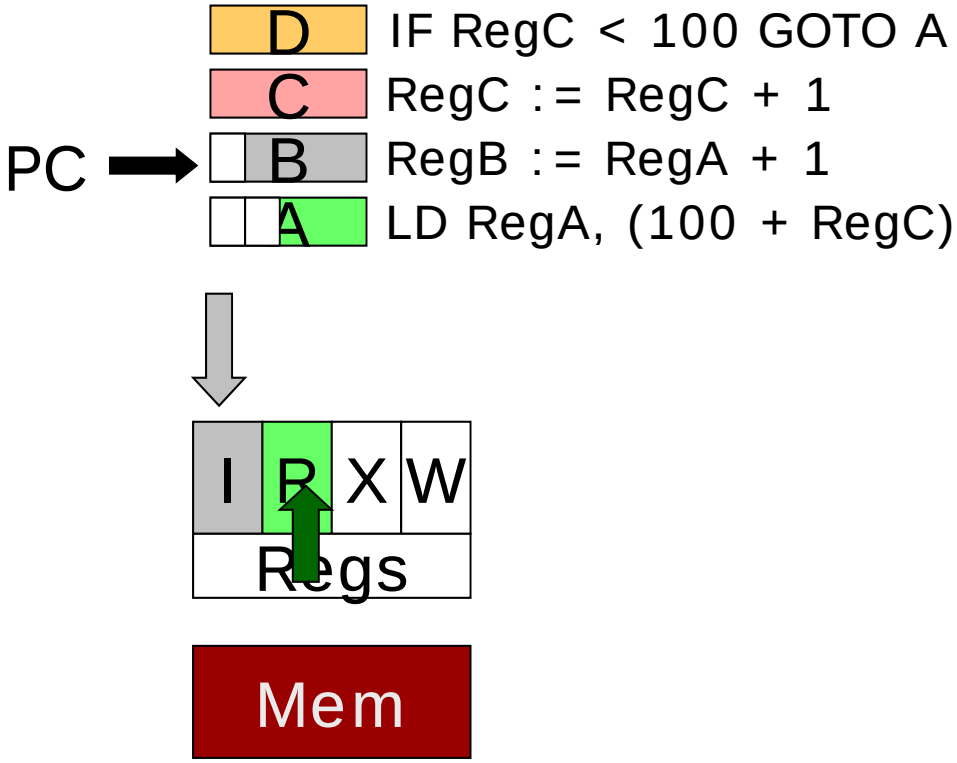
Initially


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

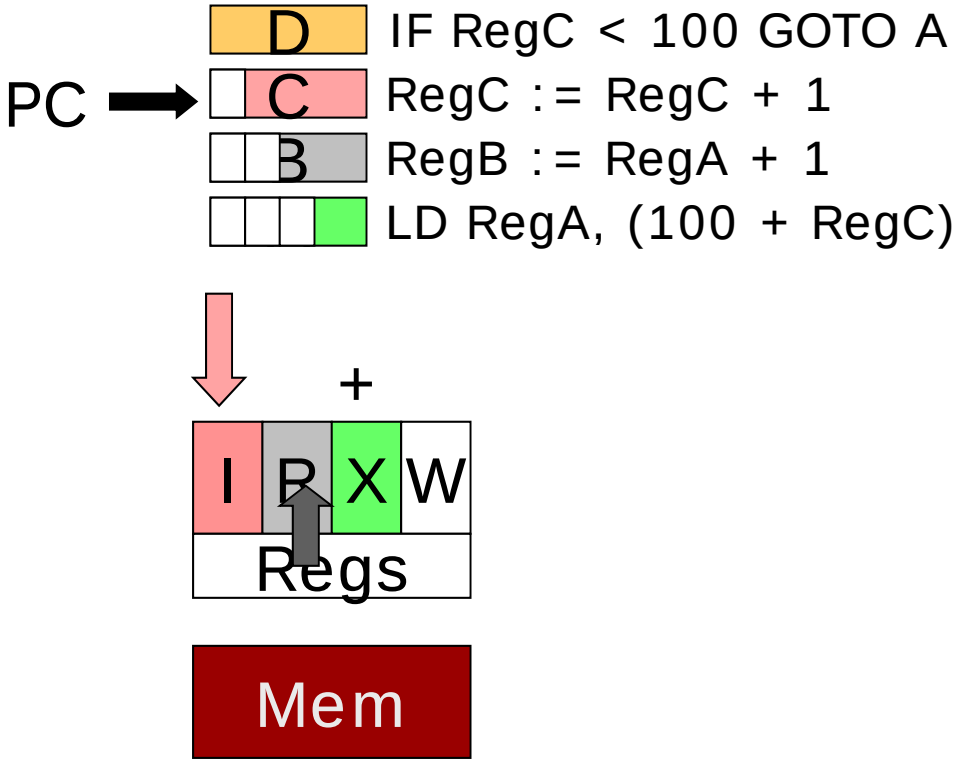
Cycle 1


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

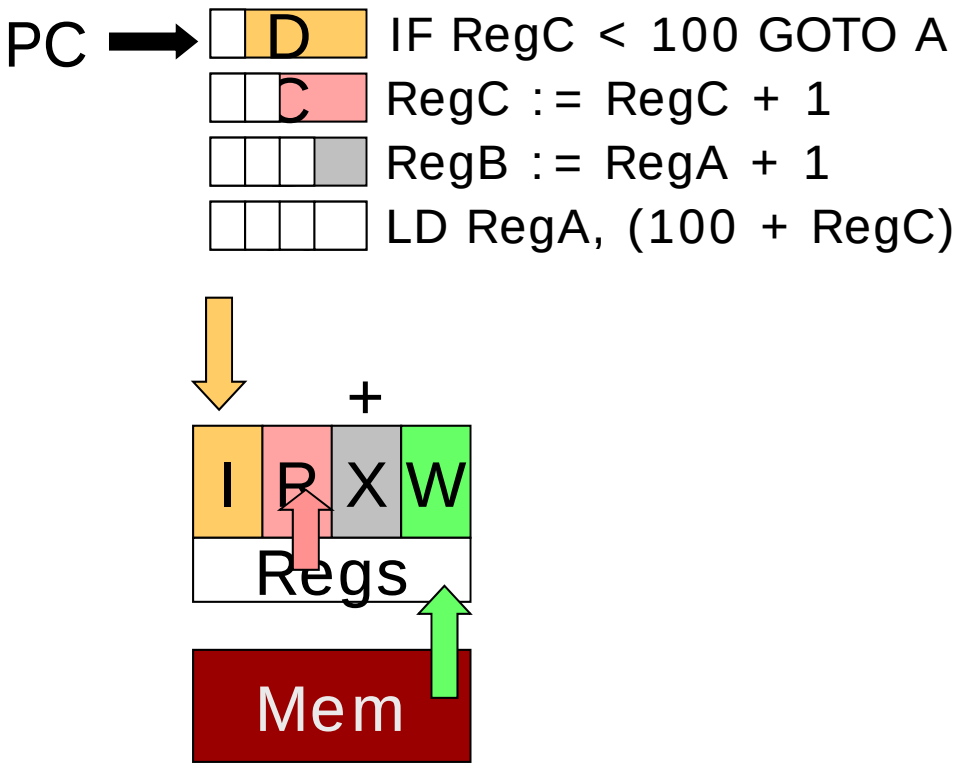
Cycle 2


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

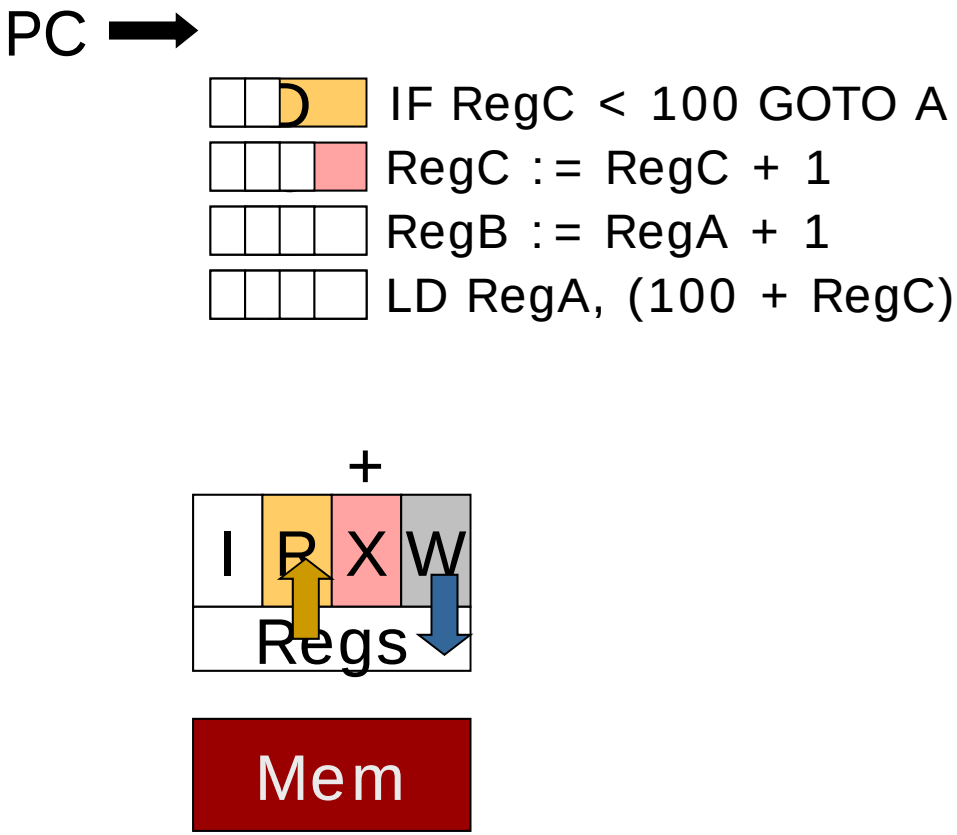
Cycle 3


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

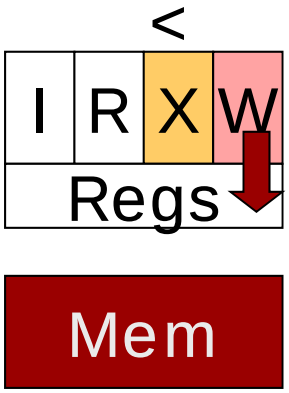
Cycle 5


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

Cycle 6

PC →

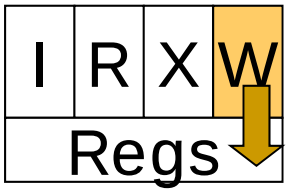
IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)



Cycle 7

PC ➡

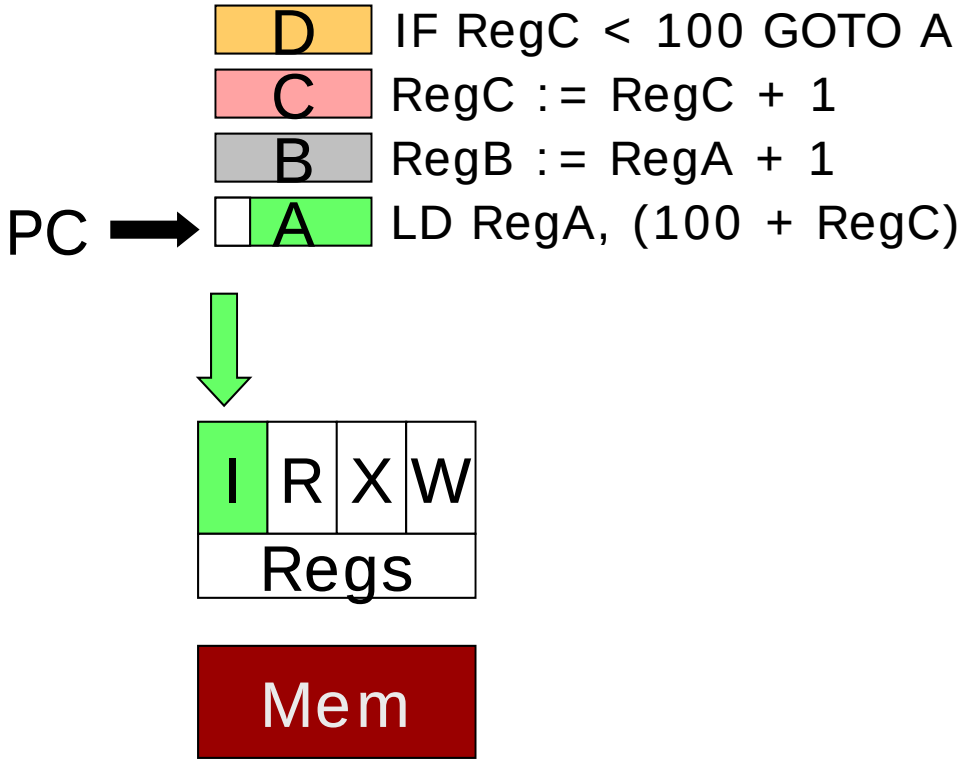
IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
A: LD RegA, (100 + RegC)



Branch: Addr(A) ➔ PC

Mem

Cycle 8

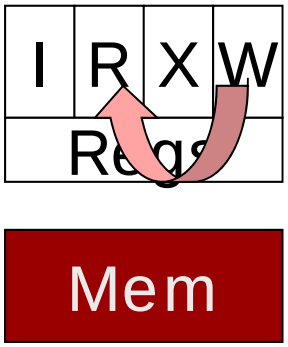

D
IF RegC < 100 GOTO A
C
RegC := RegC + 1
B
RegB := RegA + 1
PC
A
LD RegA, (100 + RegC)
I
R
X
W
Regs
Mem

Pipeline Challenges

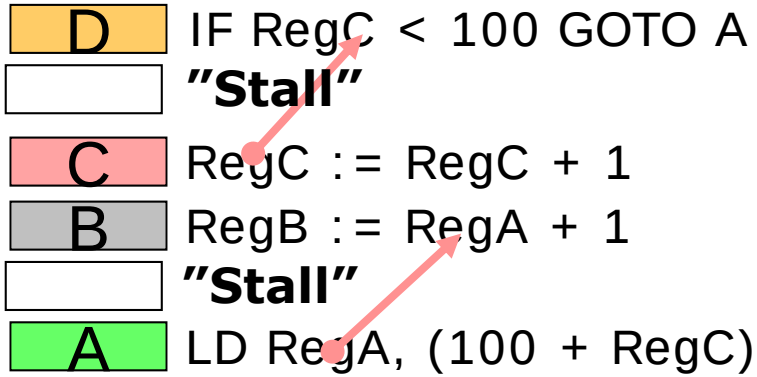
- Balance the pipeline stages
- Setup and hold time overhead
- Minimize pipeline stalls
- Predict and perform speculative work
- Undo speculative work

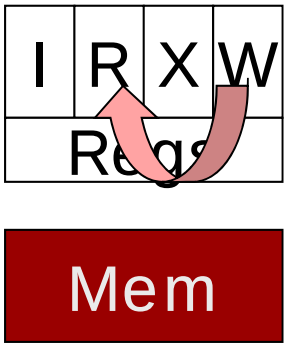
Data dependency ☹
Previous execution example wrong!

D	IF RegC < 100 GOTO A
C	RegC := RegC + 1
B	RegB := RegA + 1
A	LD RegA, (100 + RegC)

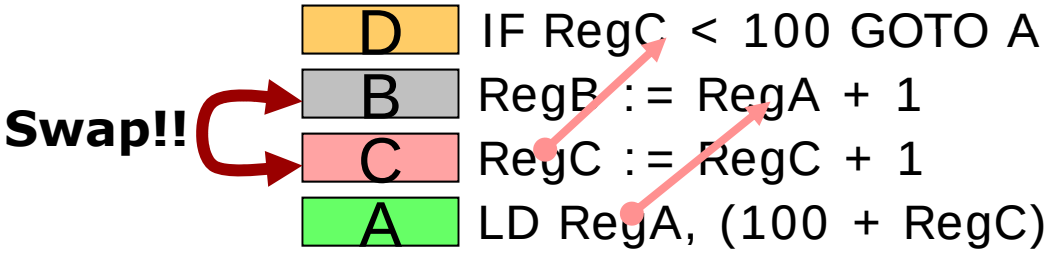


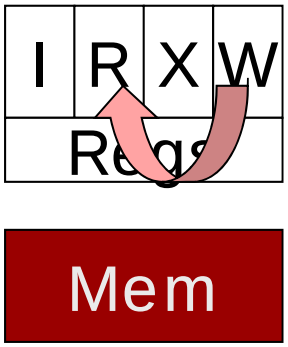
Data dependency fix 1: pipeline delays



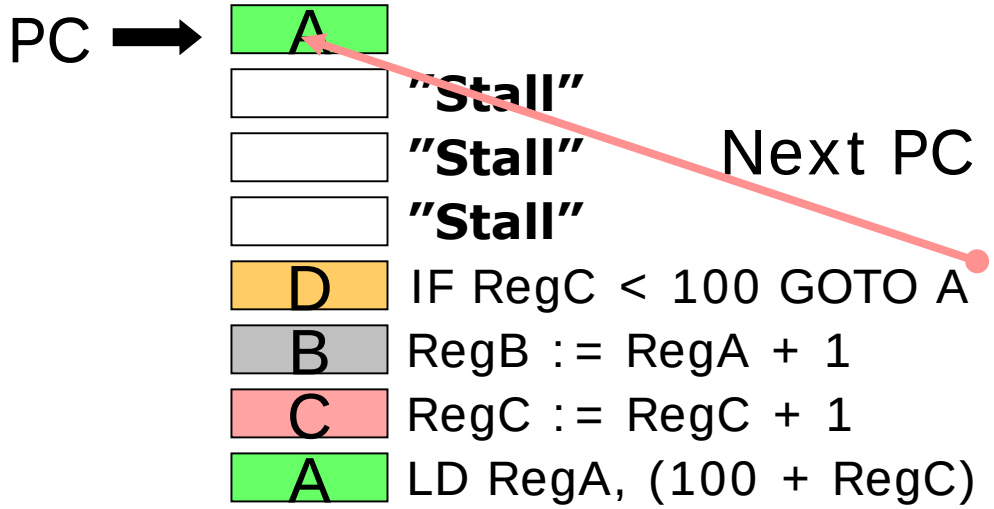


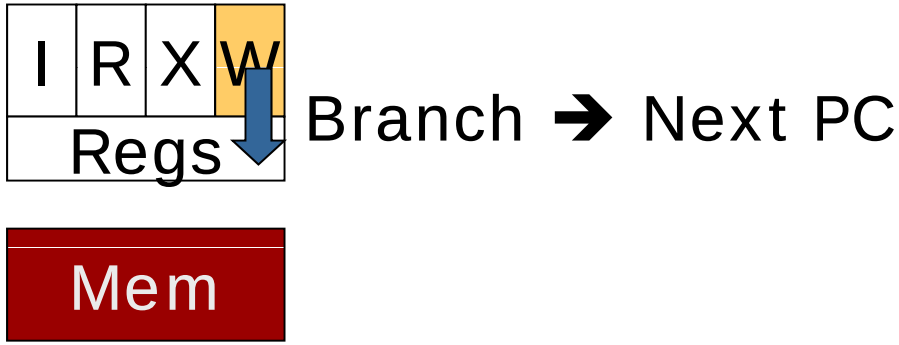
Data dependency fix 2: Compiler optimizations





Branch delays ☹



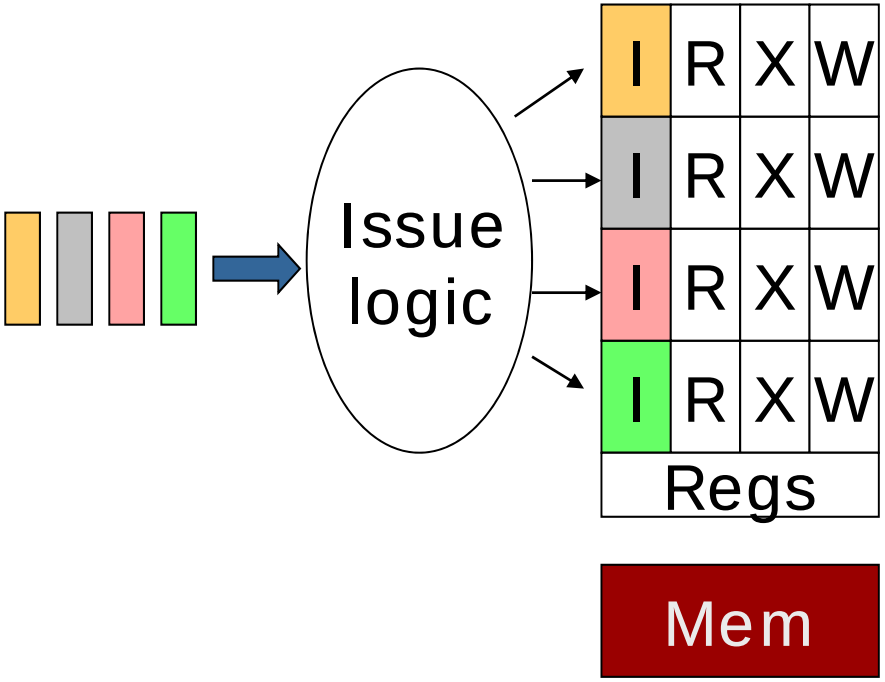


7 cycles per iteration of 4 instructions ☹

Need longer basic blocks with independent instr.

It is actually a lot worse!

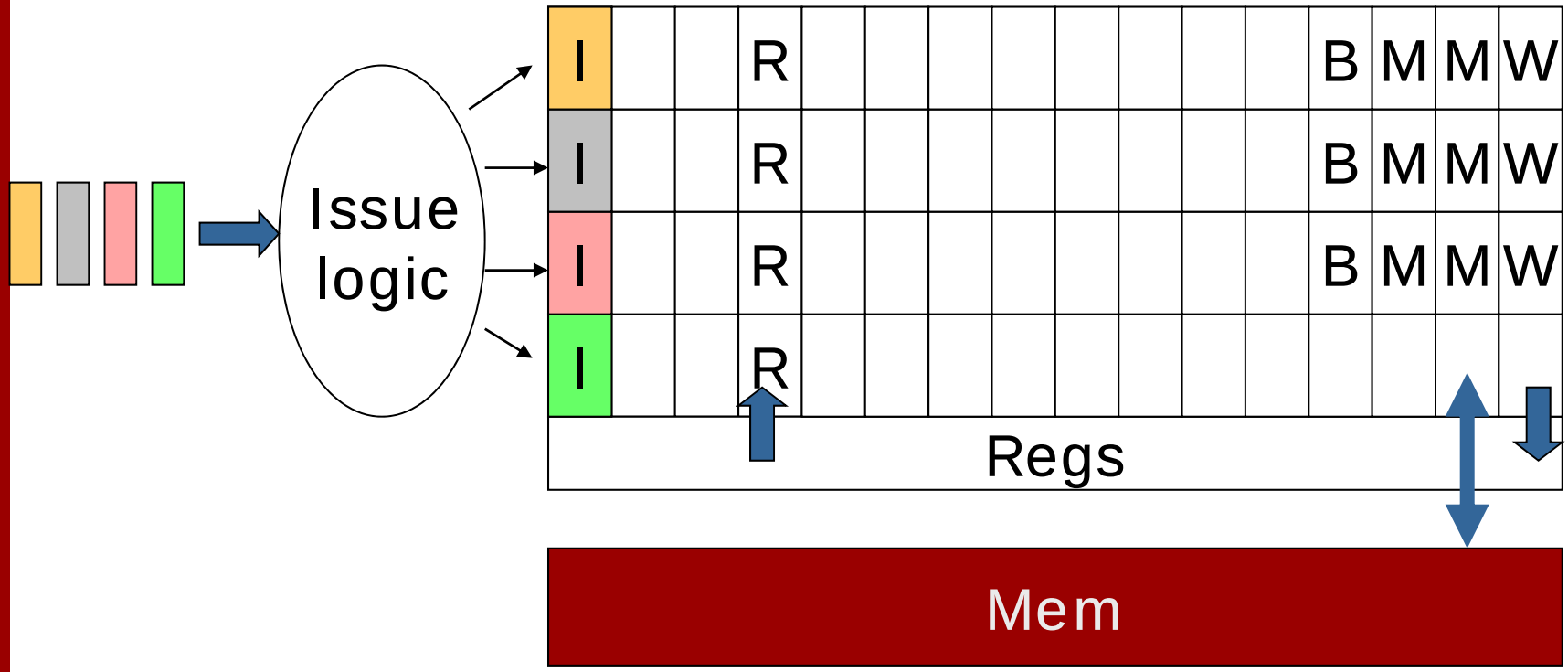
Modern CPUs:"superscalars" with ~4 parallel pipelines



+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)

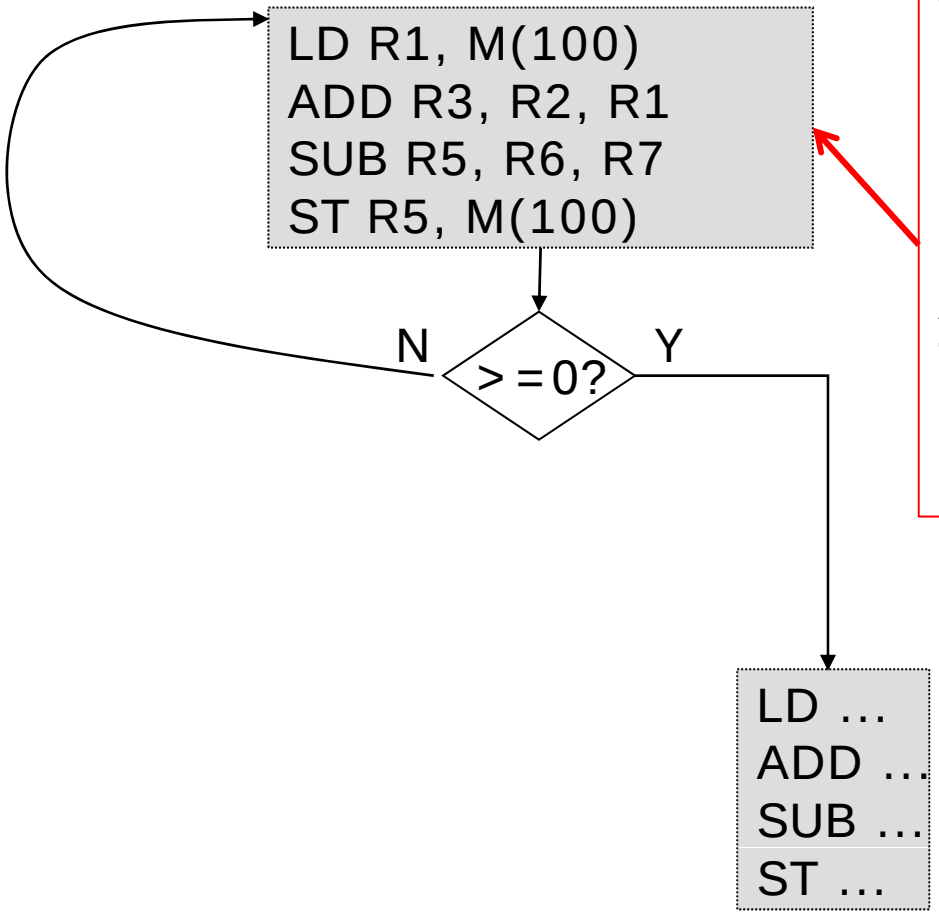
It is actually a lot worse!

Modern CPUs: ~10-20 stages/pipe



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

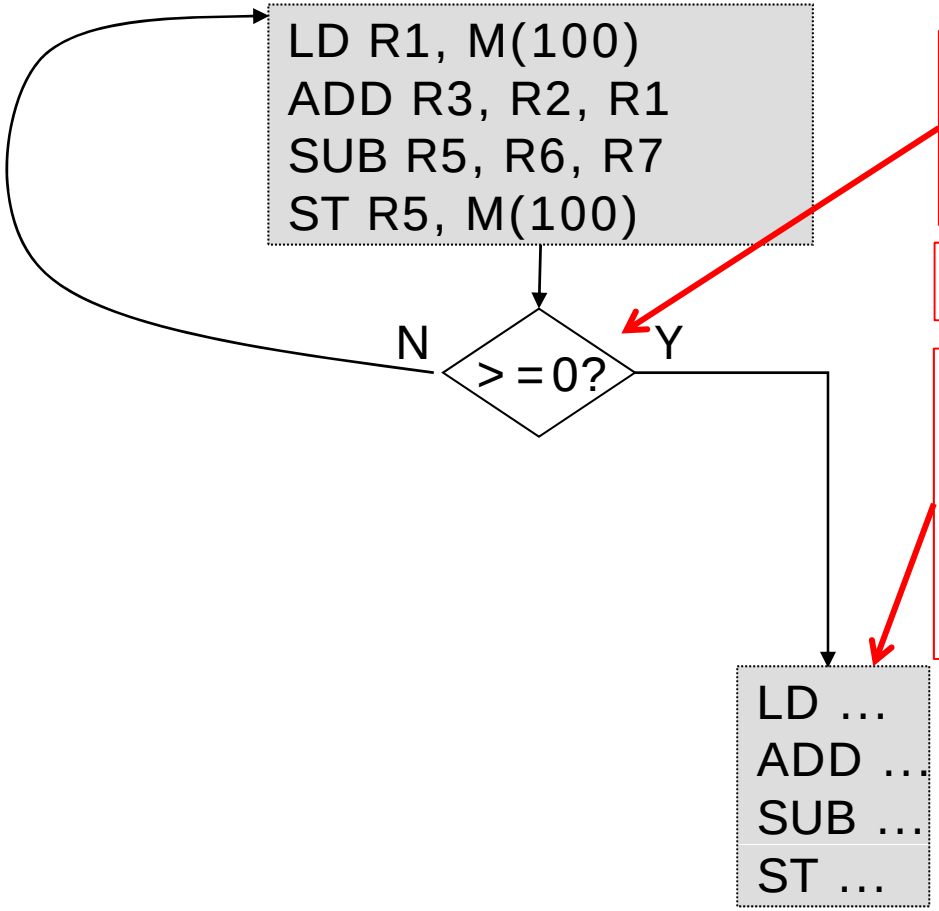
Fix: Out-of-order execution Improving ILP



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) **<u>after</u>** R3

Fix: Branch prediction

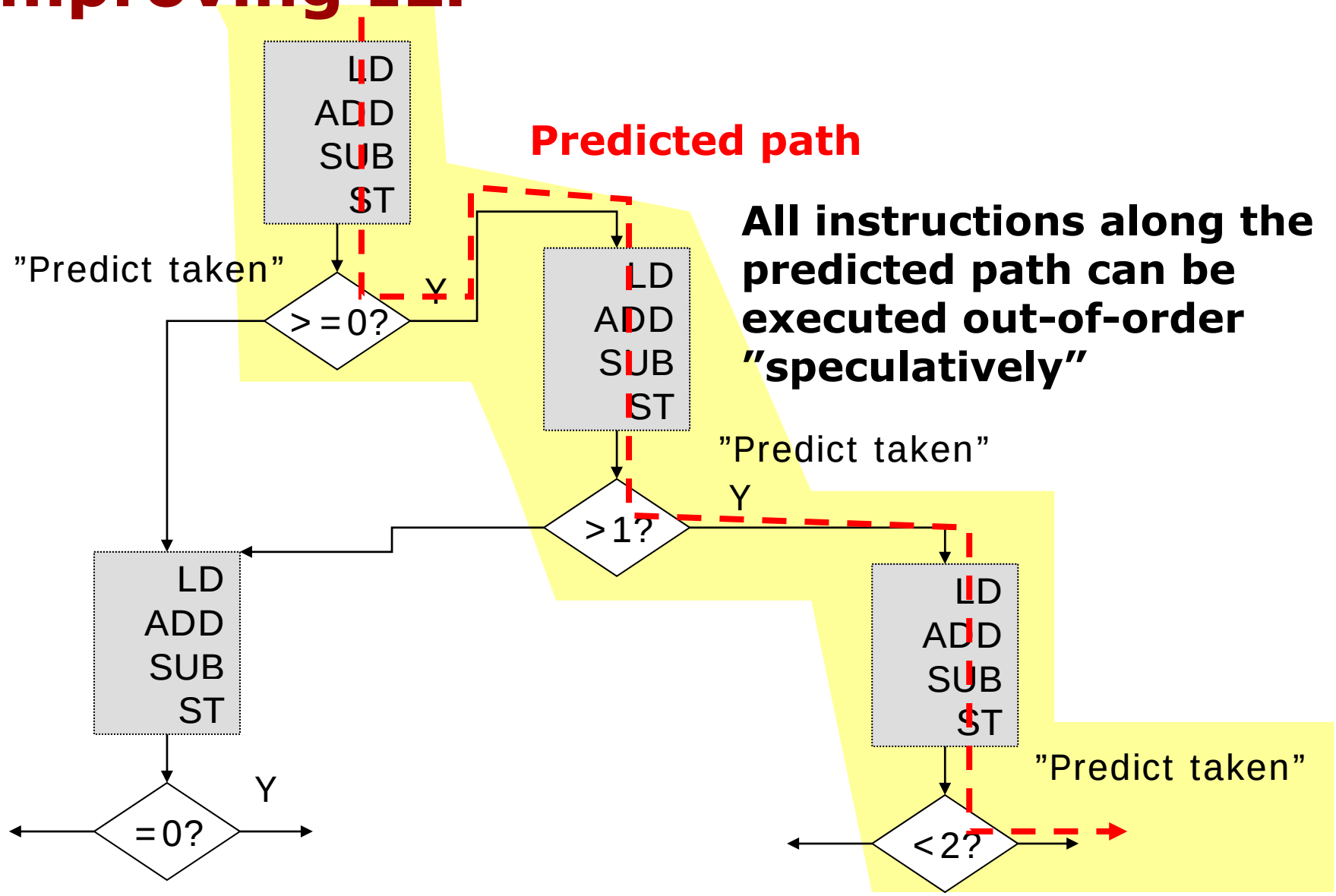


The HW can guess if the branch is taken or not and avoid branch stalls if the guess is correct,

Assume the guess is ”Y”.

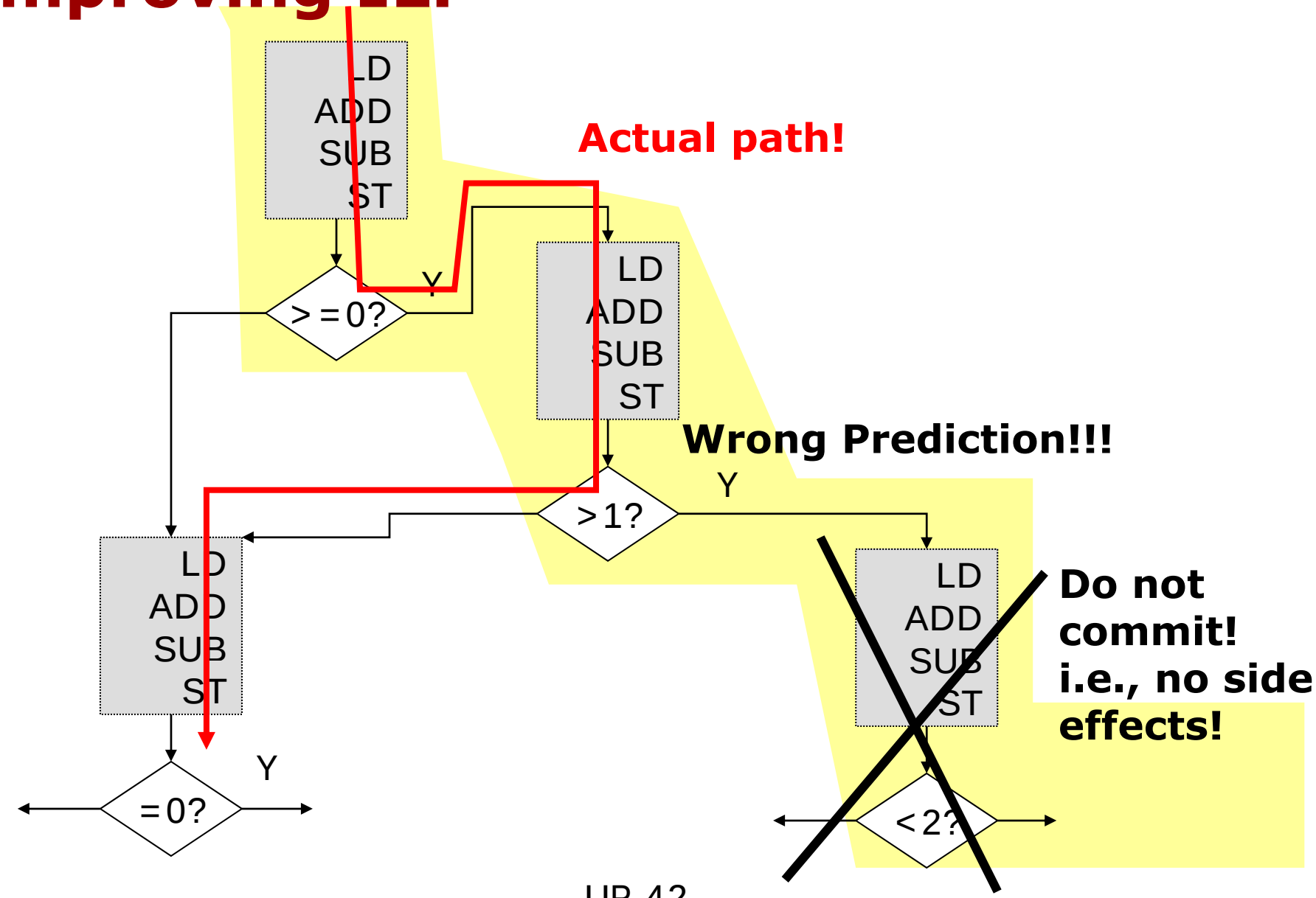
The HW can start executing these instruction **<u>before</u>** the outcome the the branch is known, but cannot allow any ”side-effect” to take place until the outcome is known.

Fix: Scheduling Past Branches Improving ILP



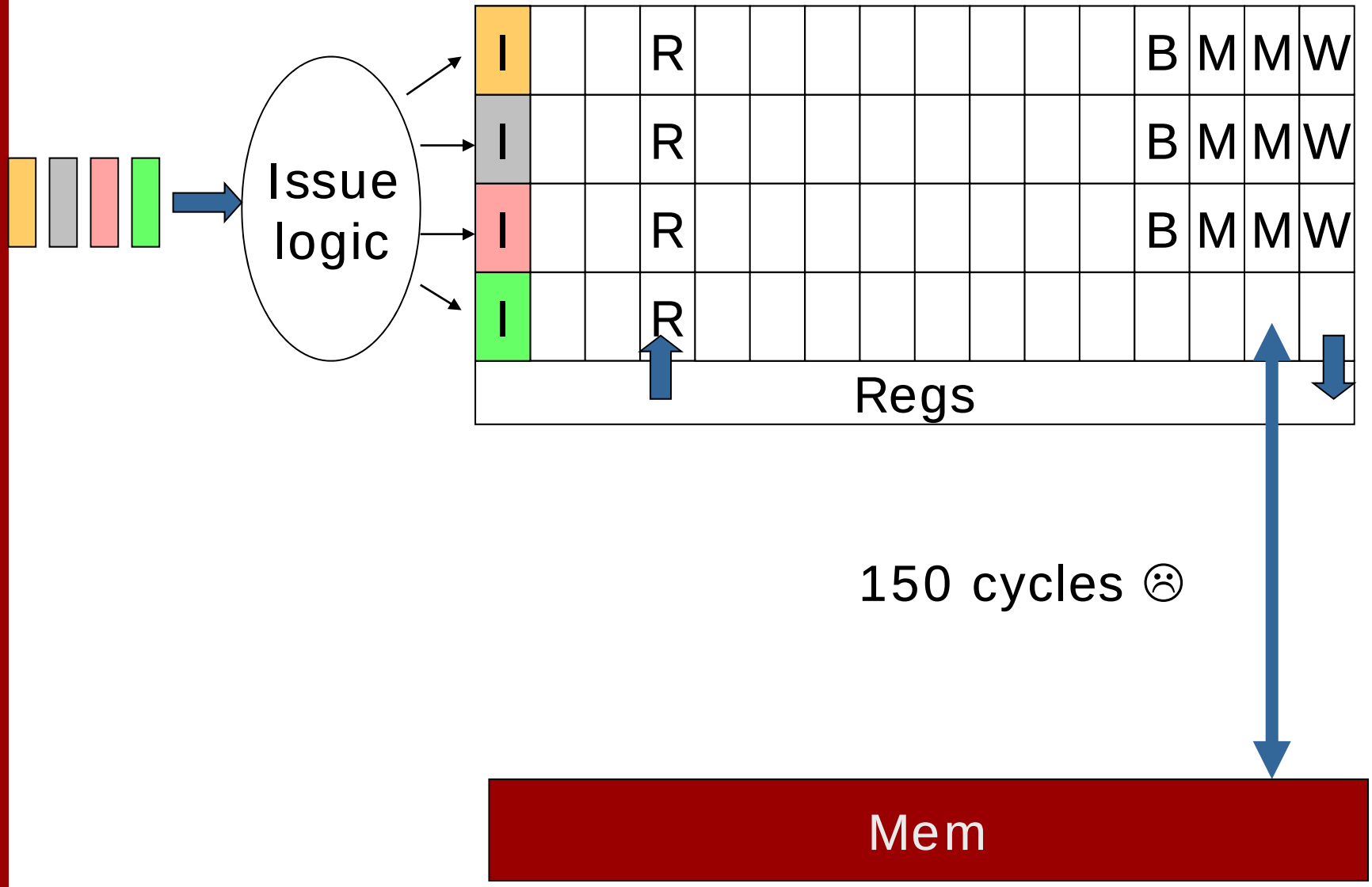
All instructions along the predicted path can be executed out-of-order "speculatively"

Fix: Scheduling Past Branches Improving ILP

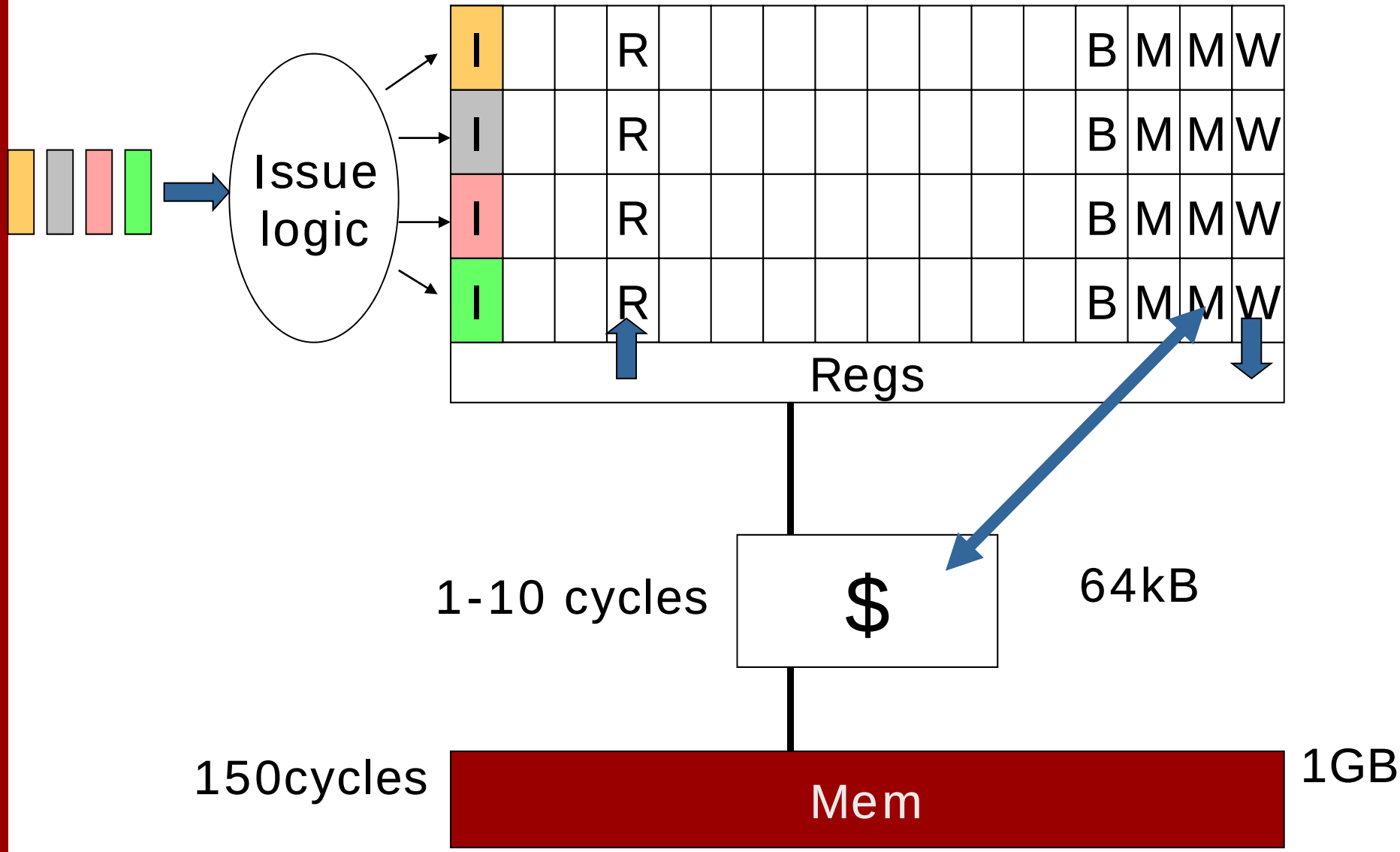


It is actually a lot worse!

DRAM access: ~150 CPU cycles



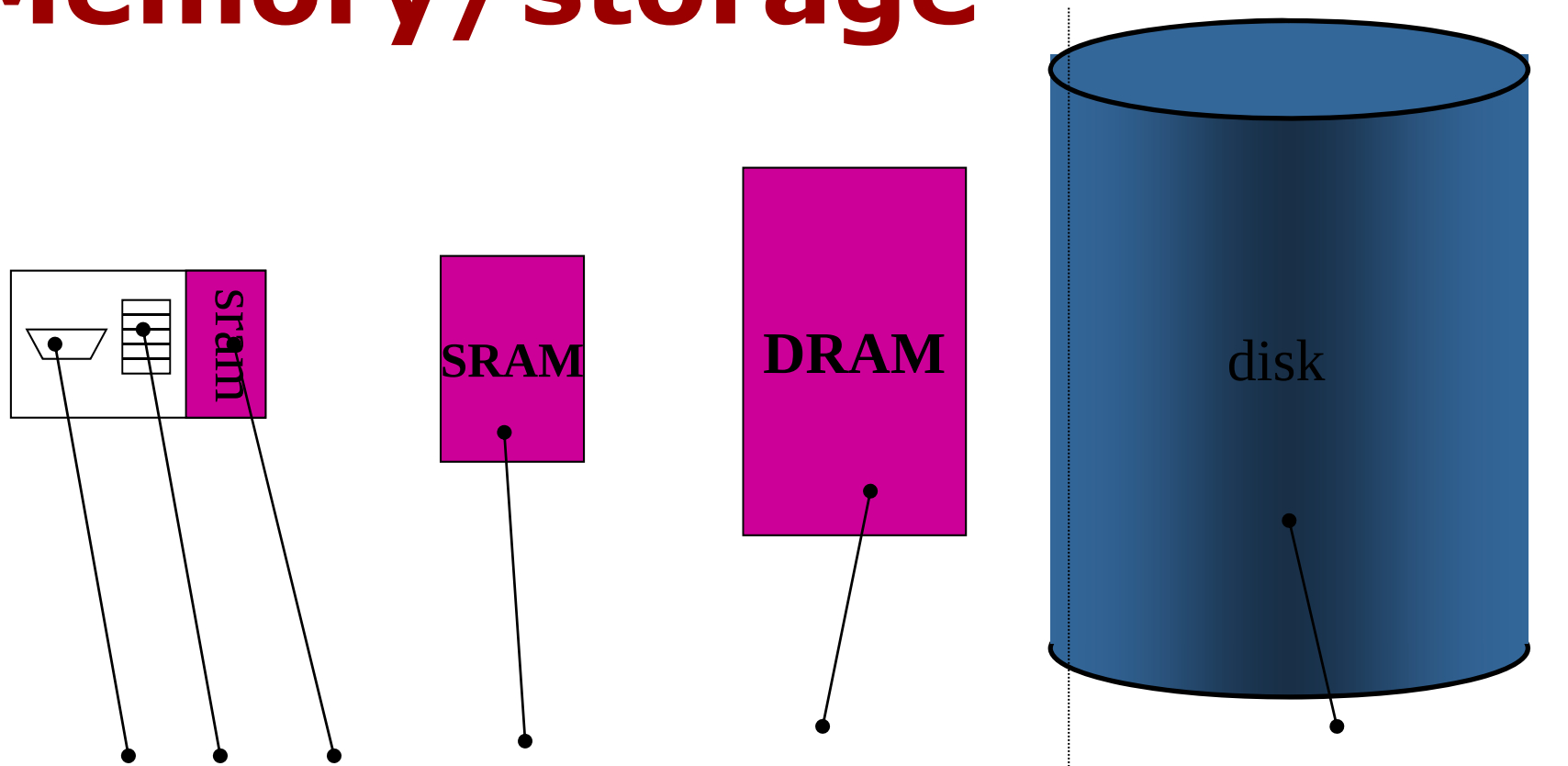
Fix: Use a cache



Caches and more caches
or
spam, spam and spam

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

Memory/storage



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

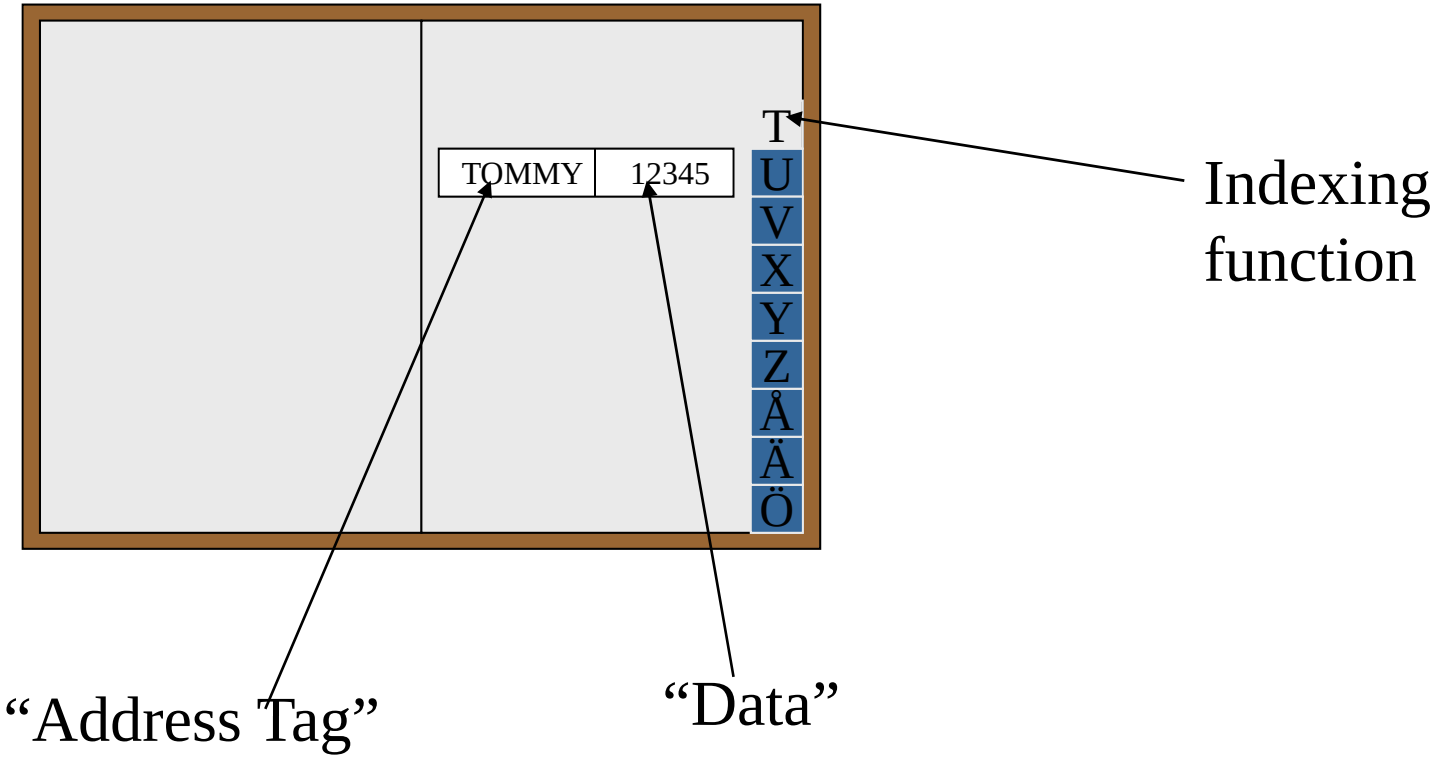
1kB 64k 4MB 1GB 1 TB

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

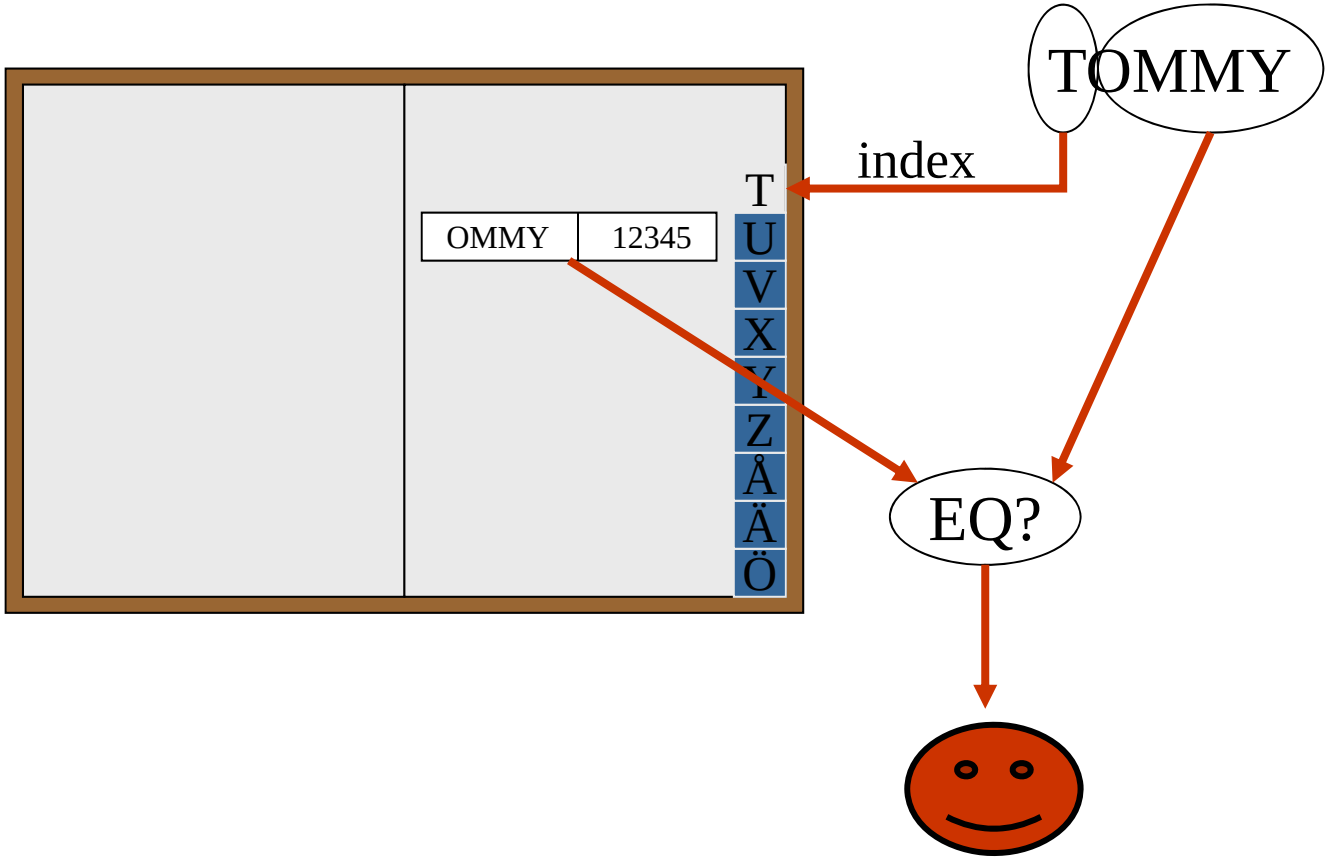
Looking for Tommy's Telephone Number



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

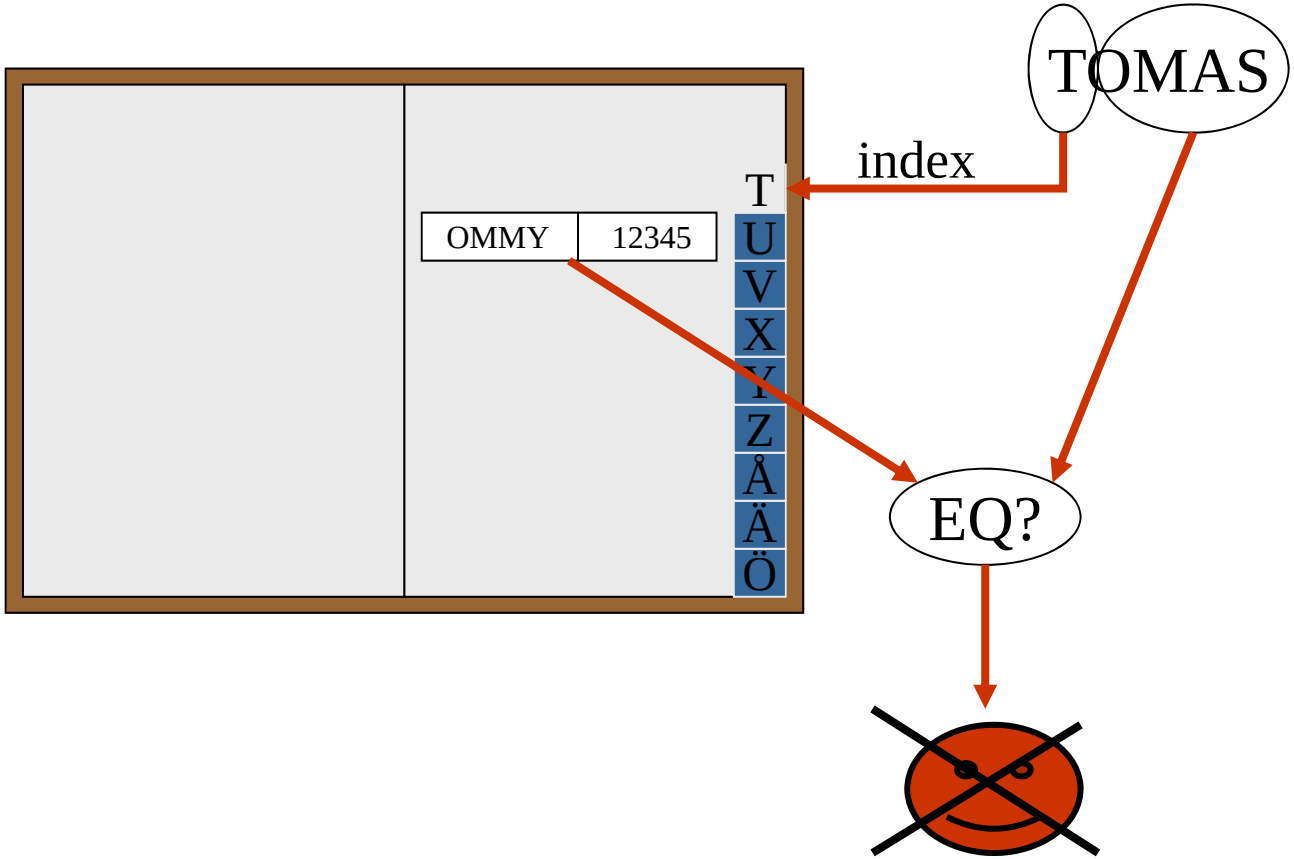
Address Book Cache

Looking for Tommy's Number



Address Book Cache

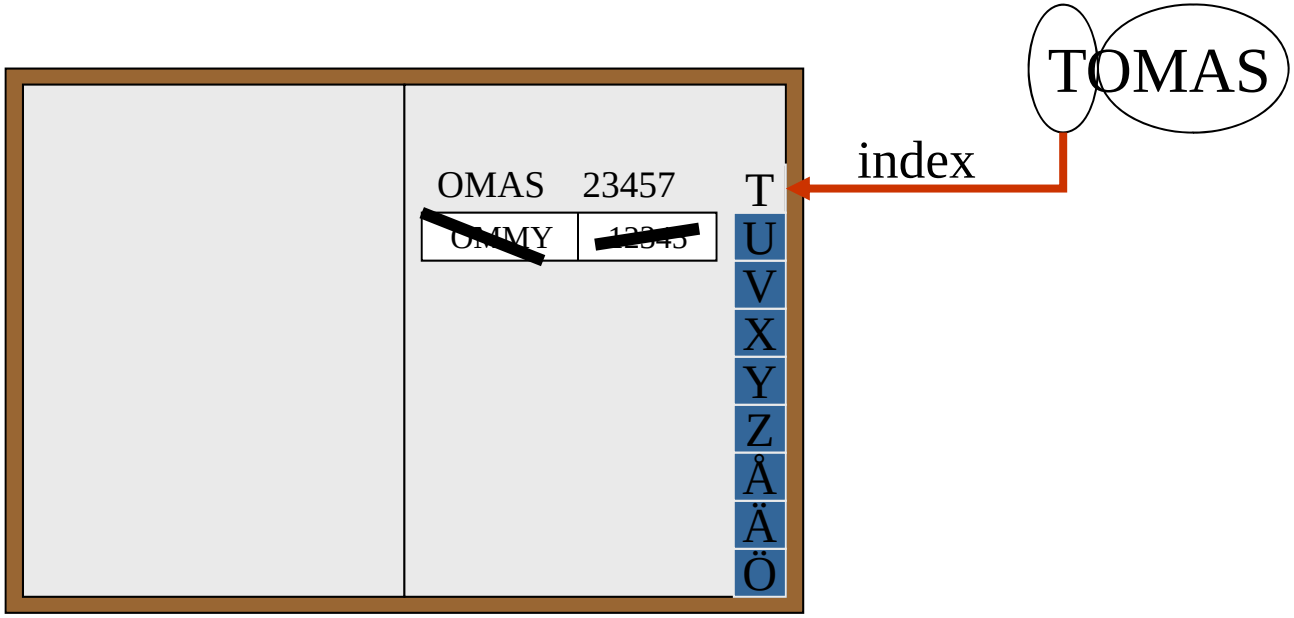
Looking for Tomas' Number



Miss!
Lookup Tomas' number in the telephone directory

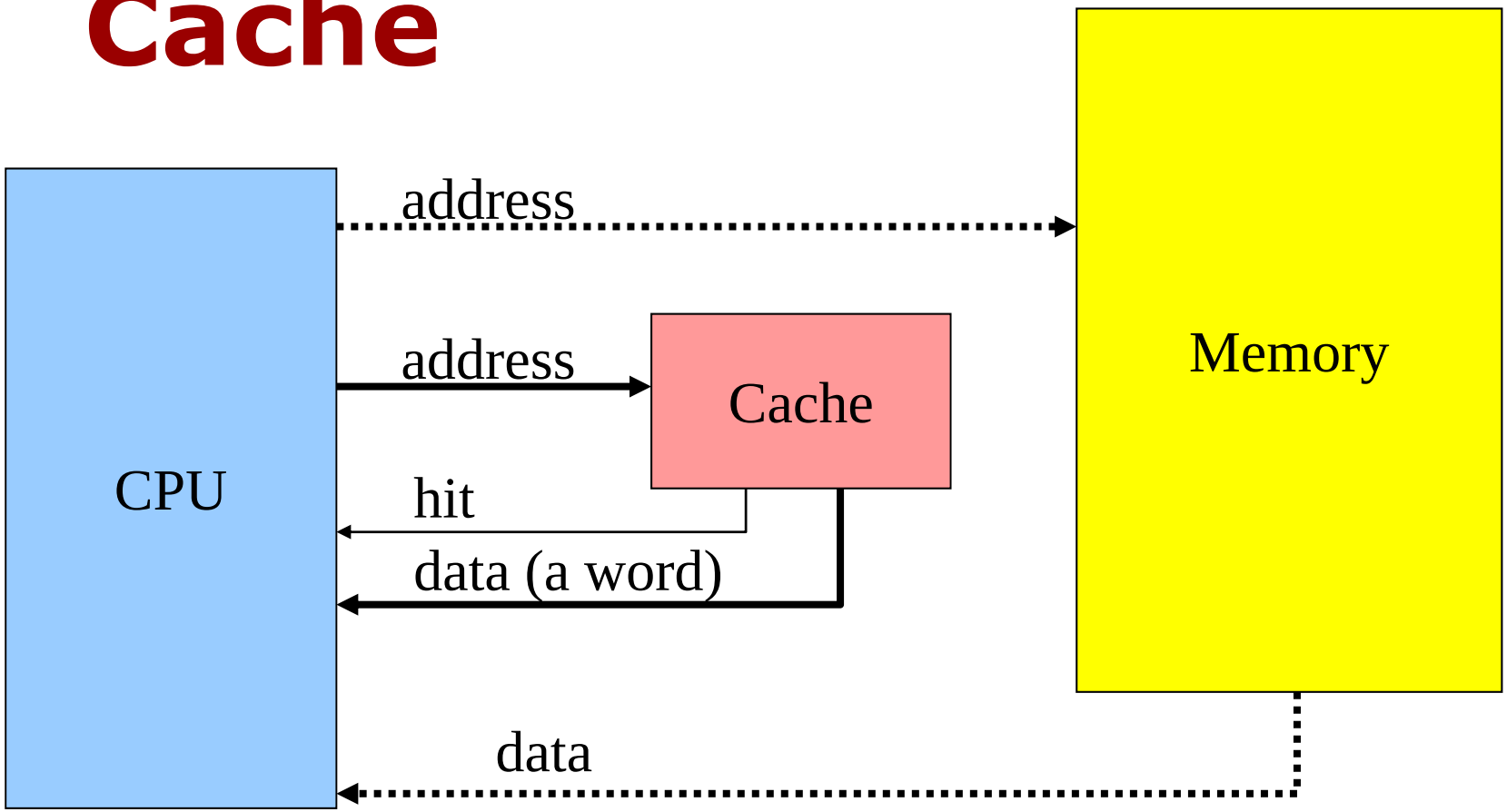
Address Book Cache

Looking for Tomas' Number

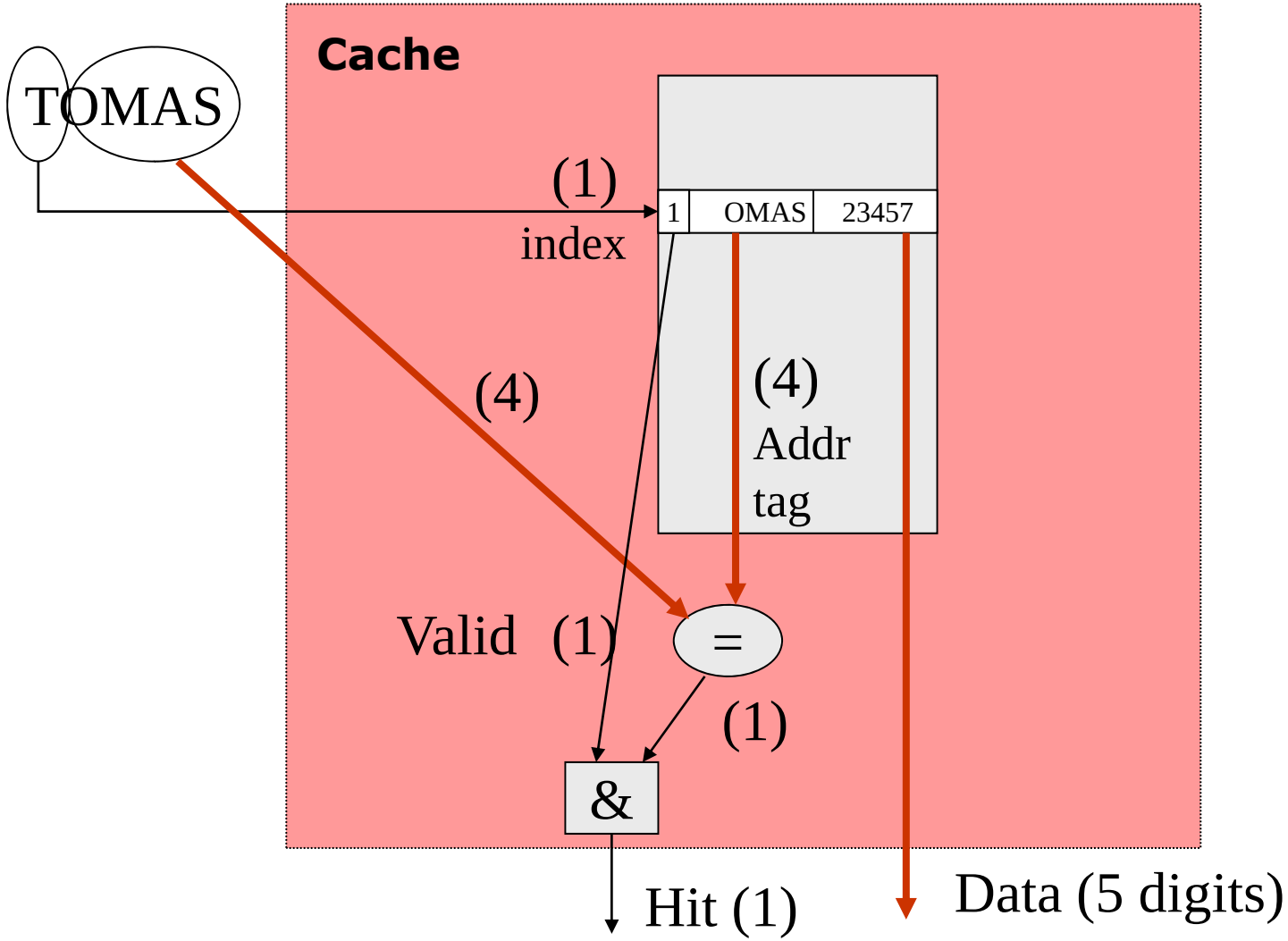


Replace TOMMY's data with TOMAS' data.
(Only one person per page = direct mapped cache)

Cache

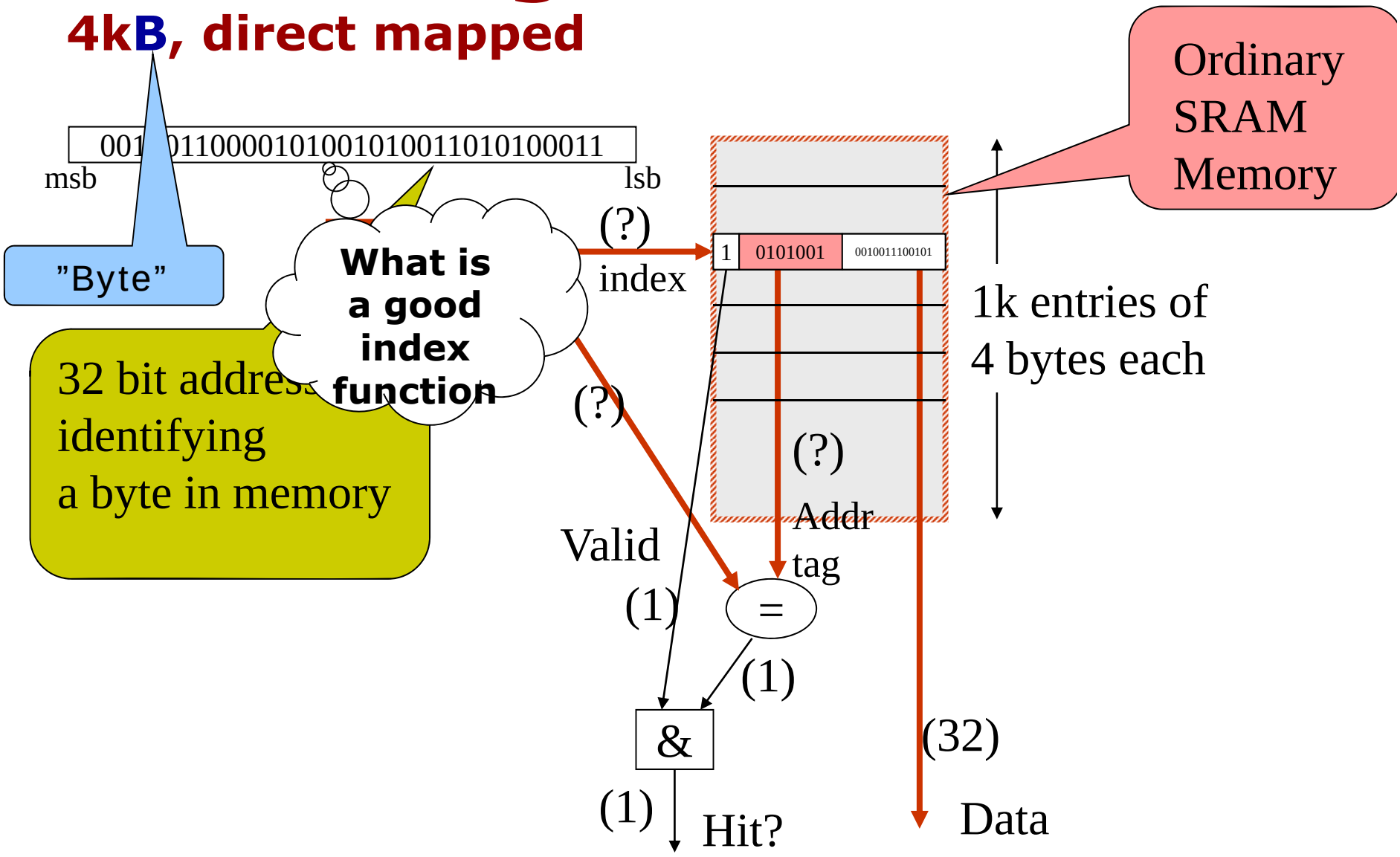


Cache Organization

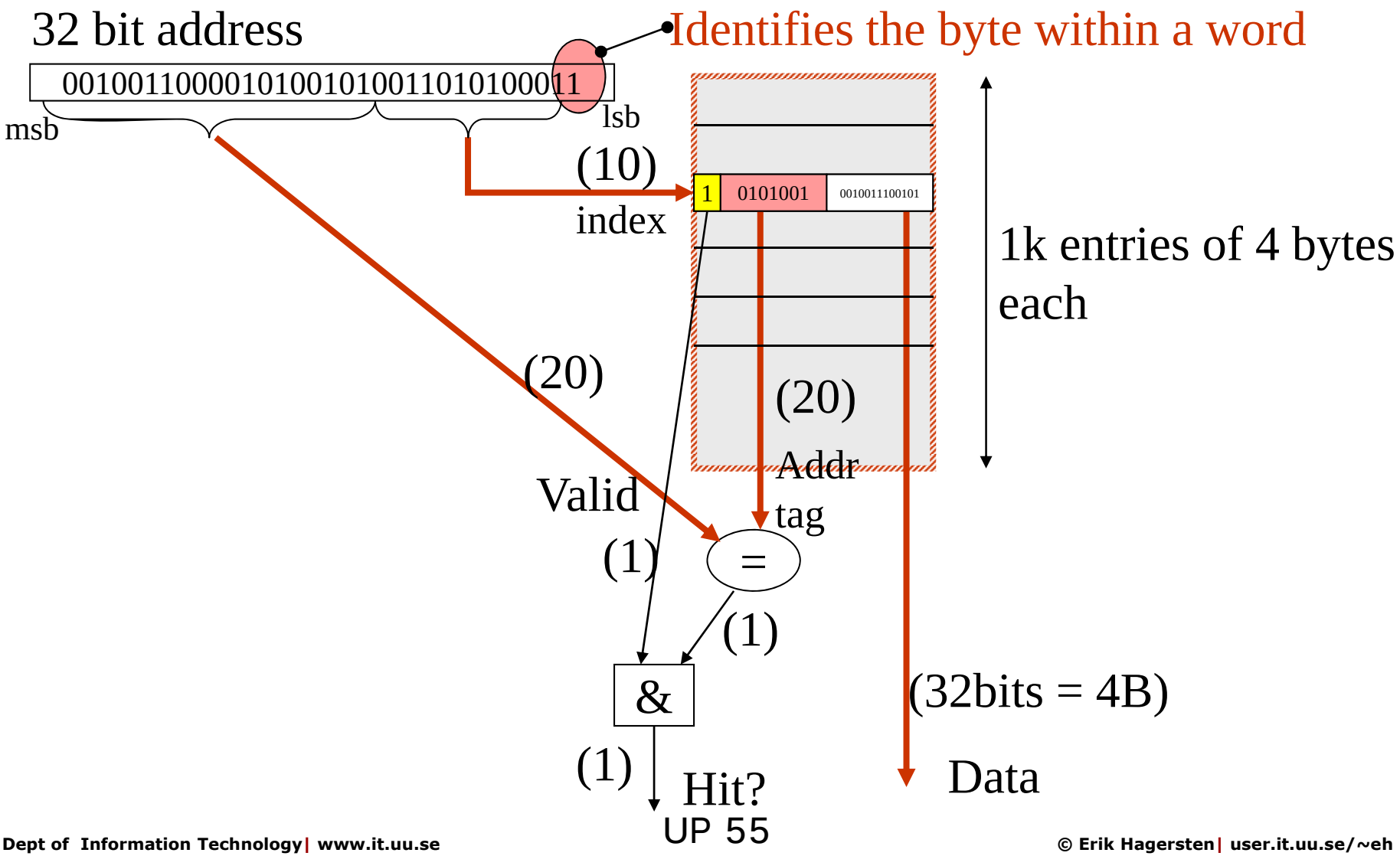


Cache Organization (really)

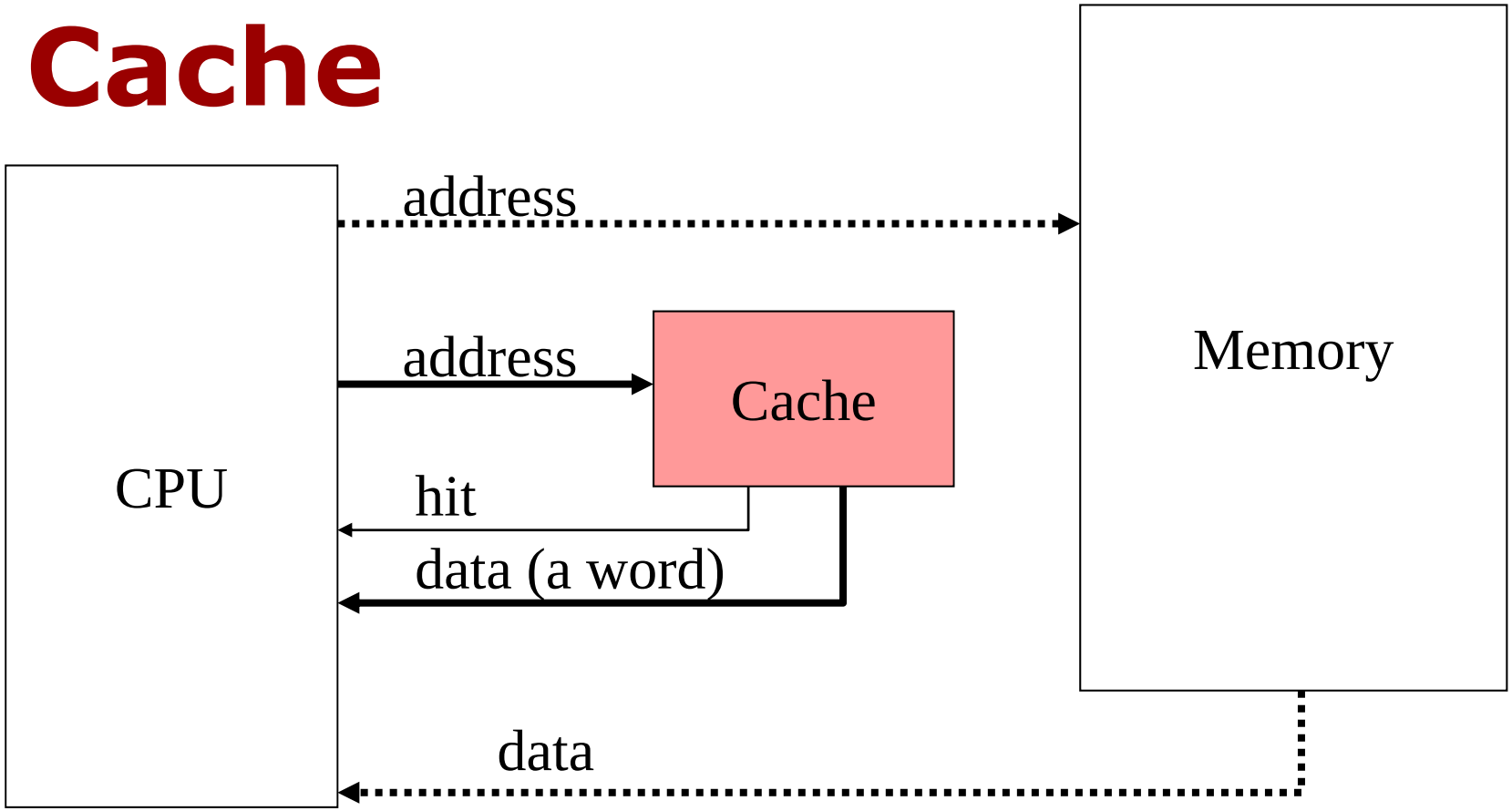
4kB, direct mapped



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

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)

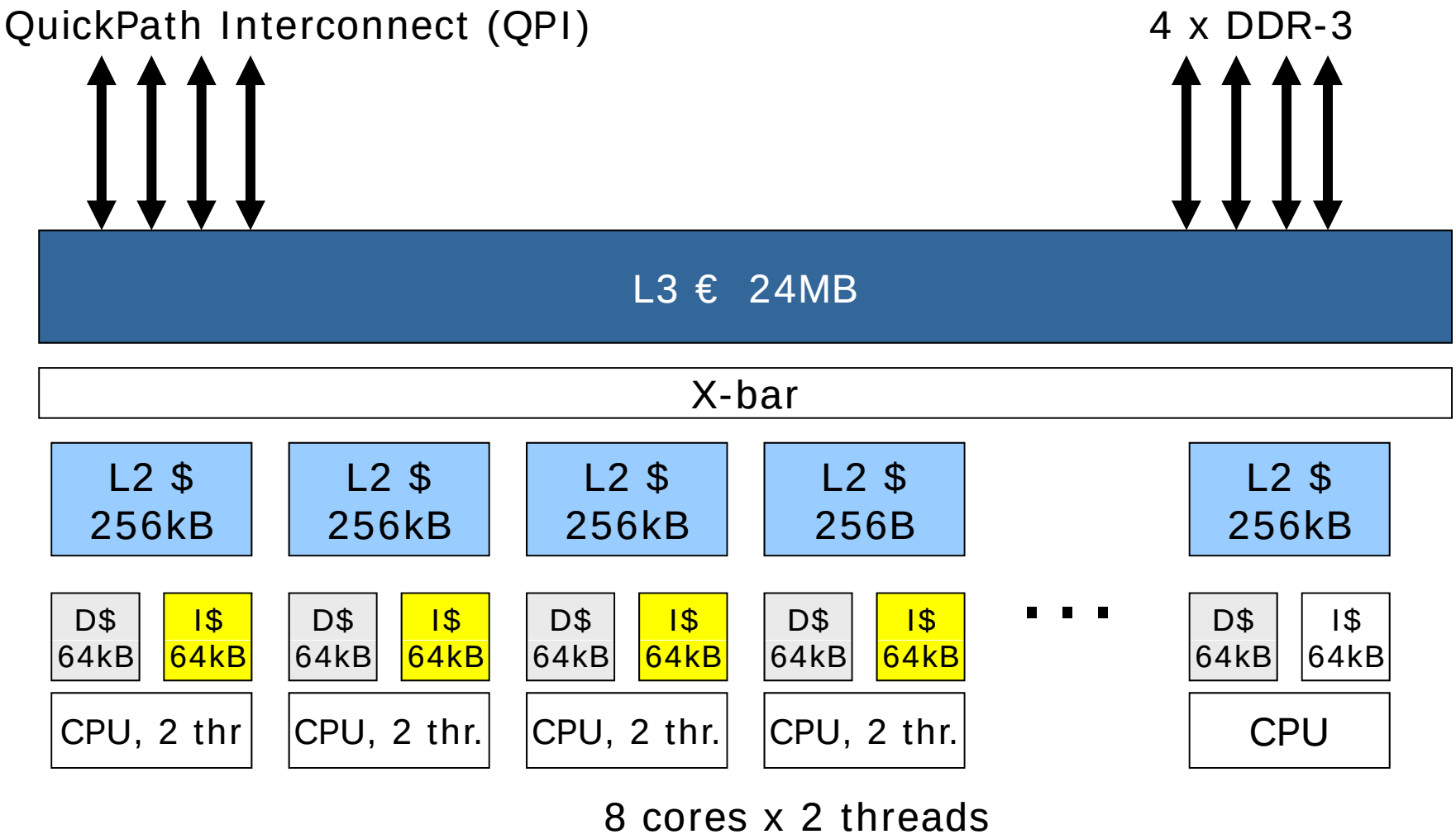
How to get more effective caches?

- 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)

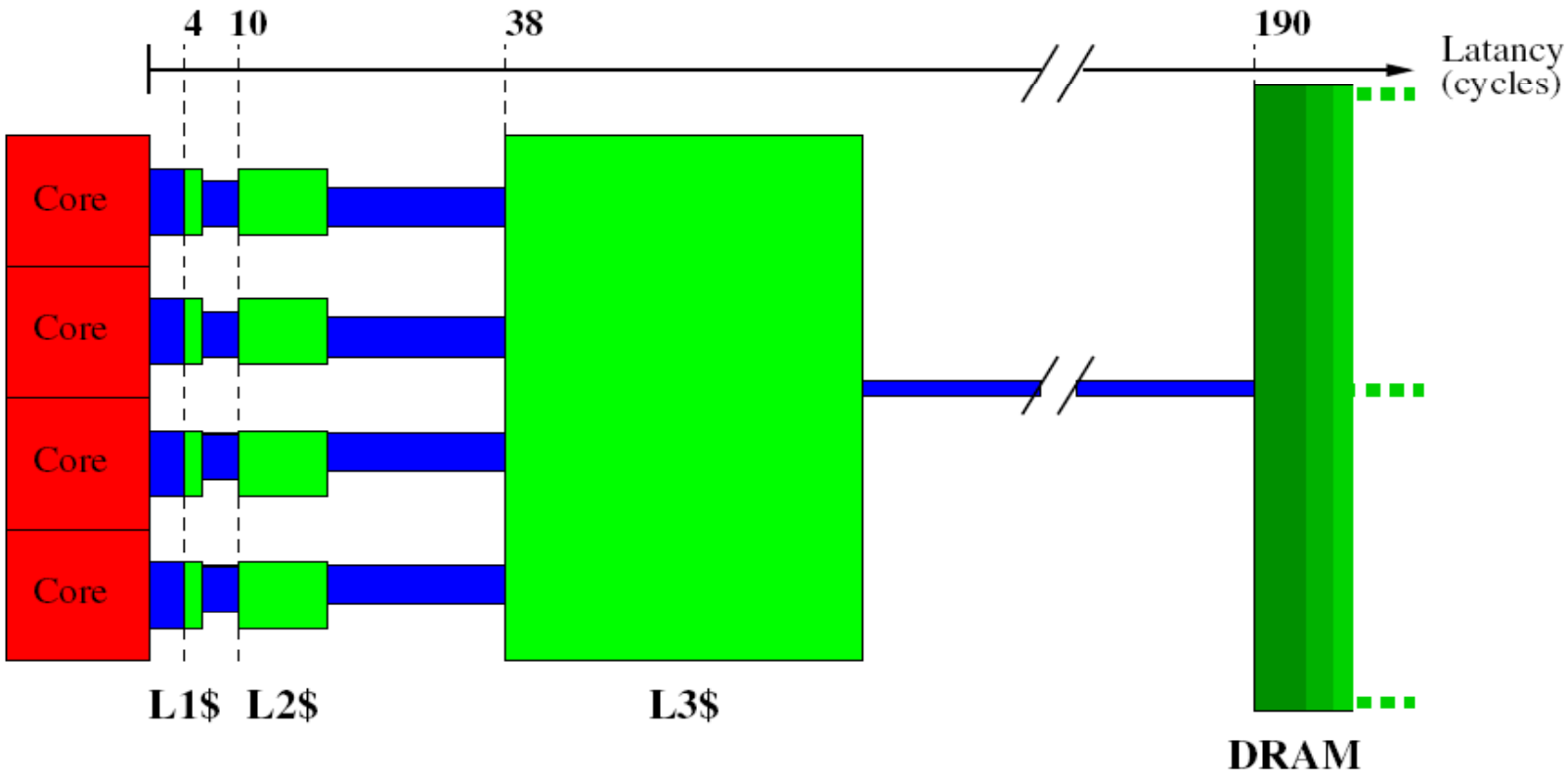
Who to replace?
Picking a "victim"

- 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...

Intel: ”Nehalem-Ex” (i7)



Cache Capacity/Latency/BW



Cache implementation

Caches at all level roughly work like this:

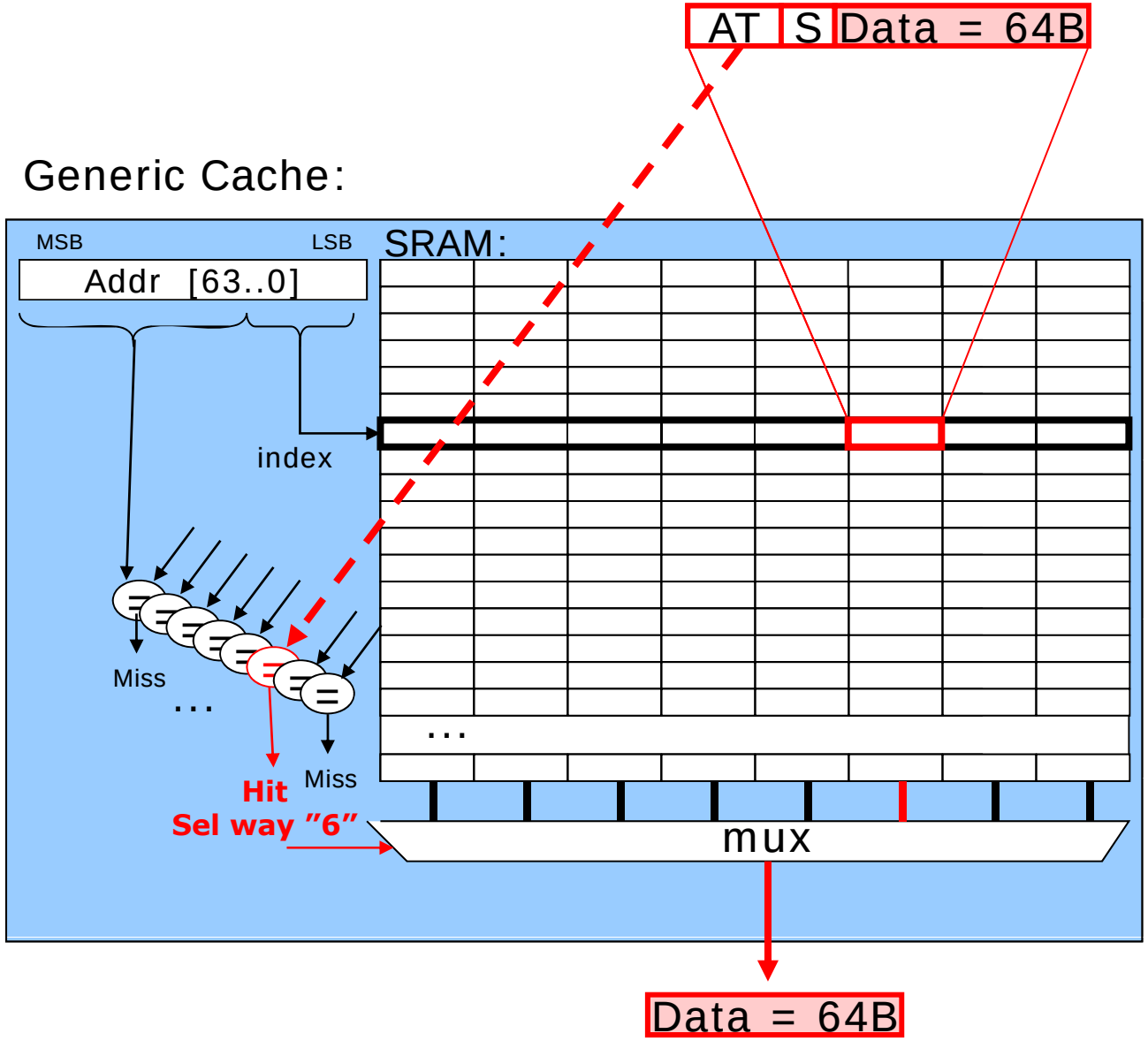
L3 € 24MB

L2 $ 256kB

D1 ¢ 64kB

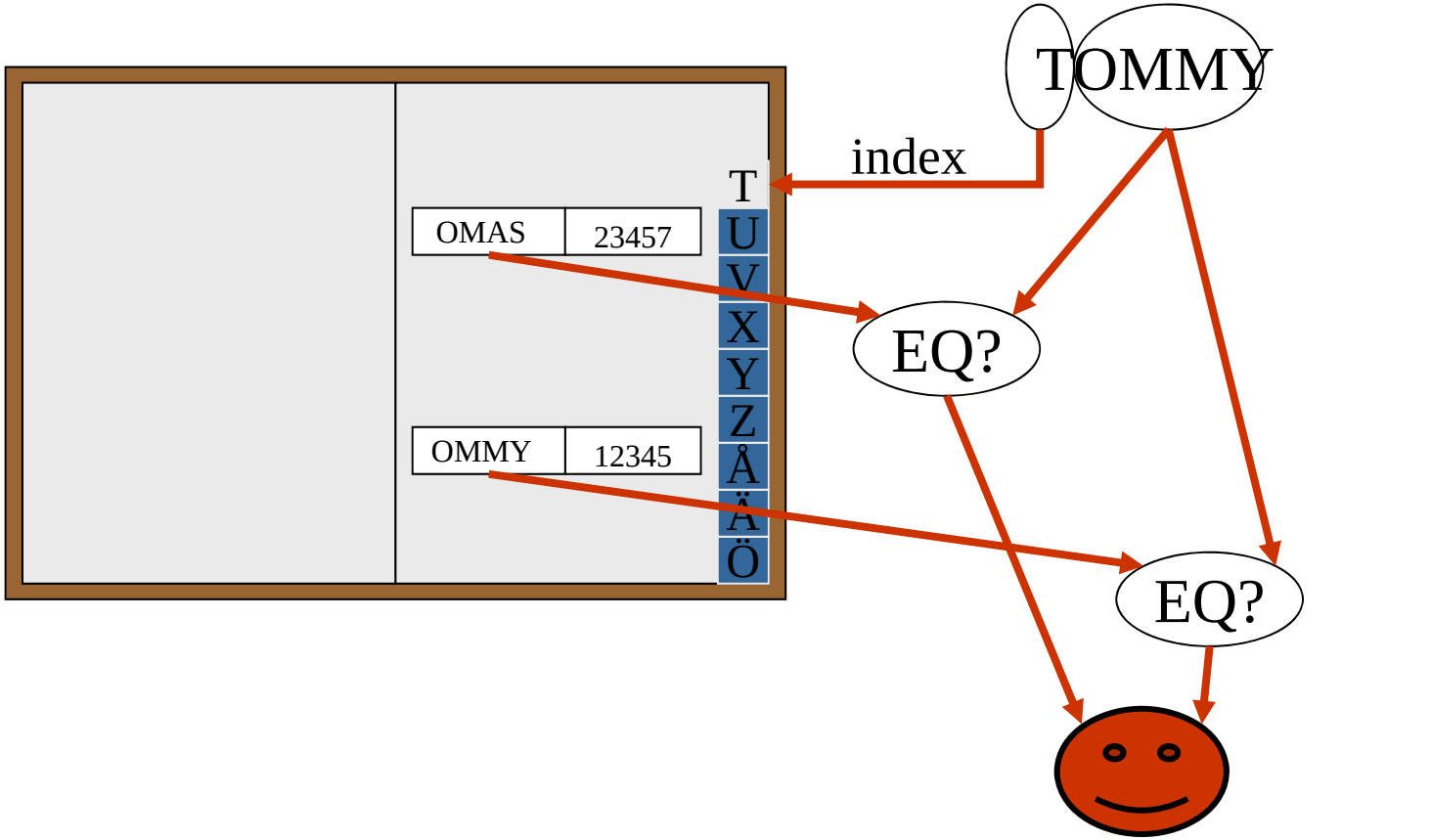
I1 ¢ 64kB

Cacheline, here 64B:



Address Book Analogy

Two names per page: index first, then search.



Select the second entry!

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)

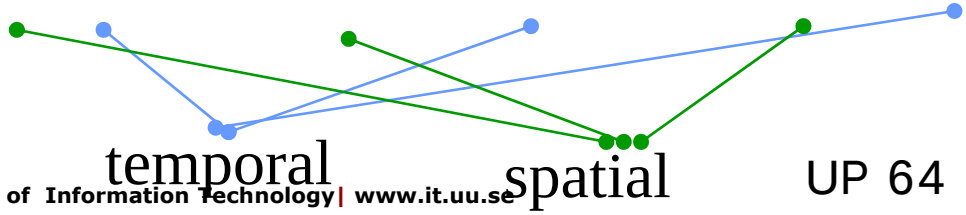
Temporal locality: Likelihood to access the same data again soon

Spatial locality: Likelihood to access nearby data again soon

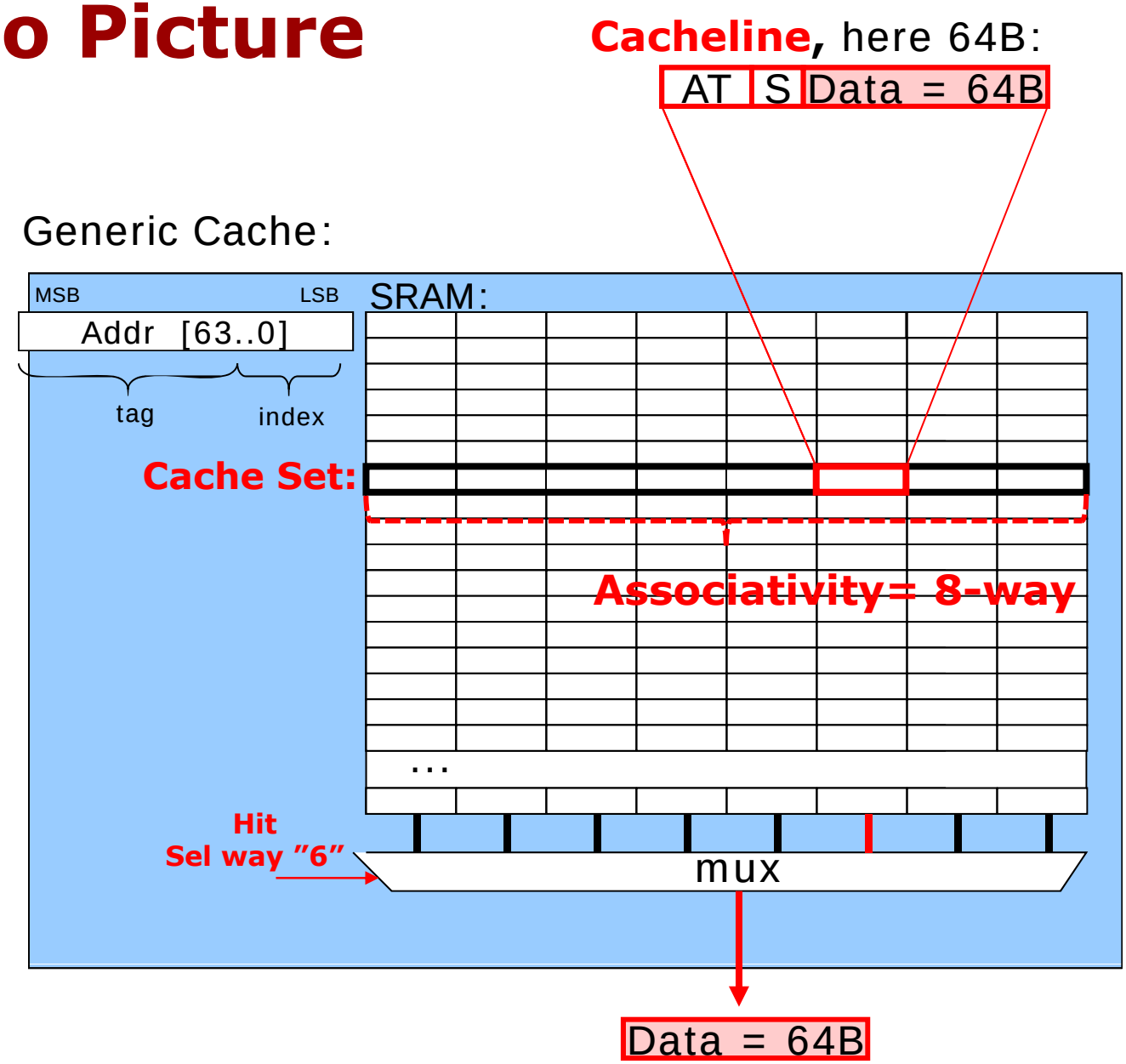
Typical access pattern:

(inner loop stepping through an array)

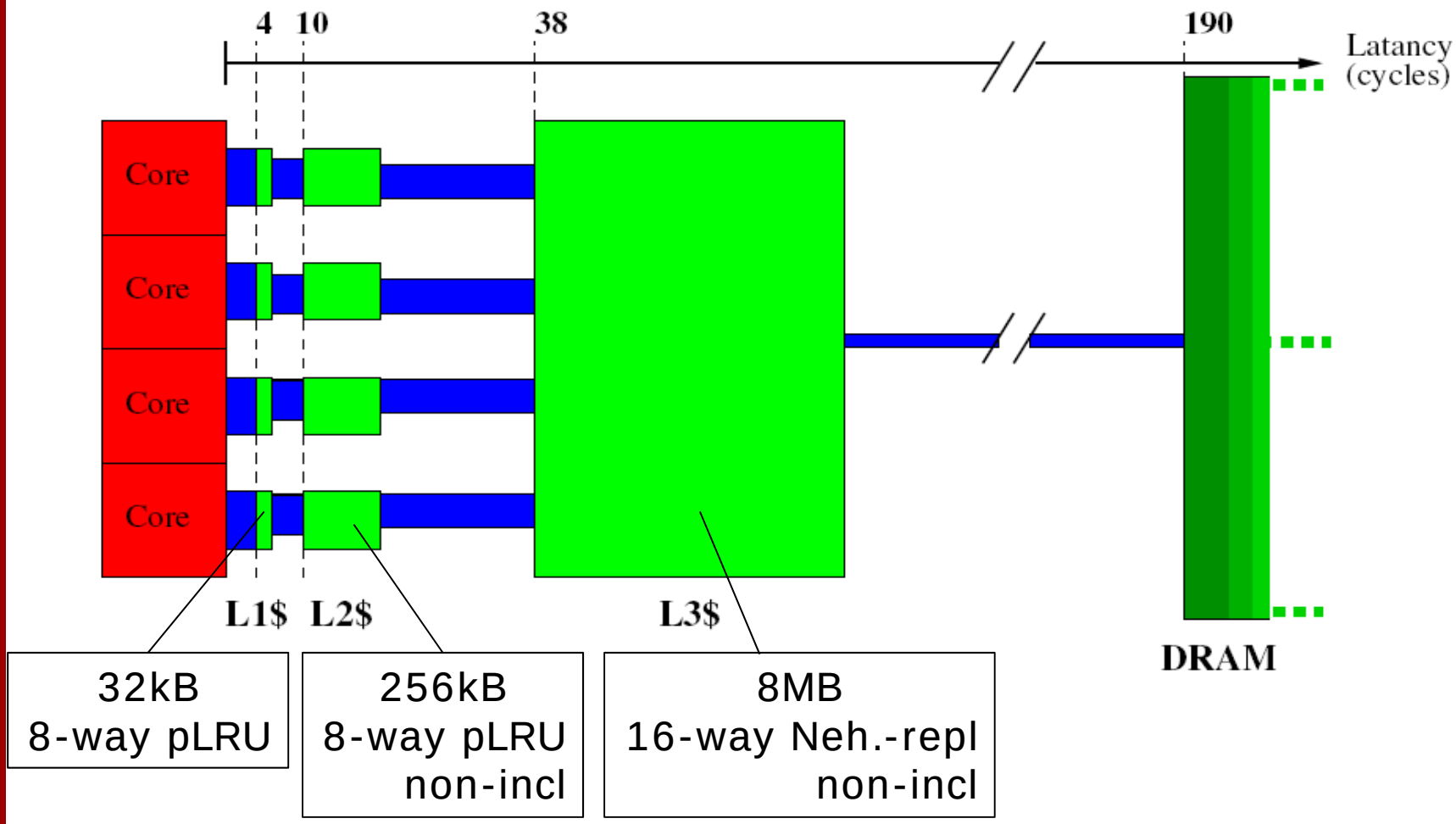
A, B, C, A+4, B, C, A+8, B, C, ...



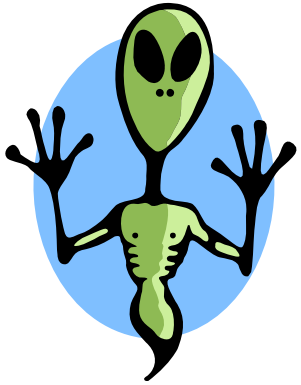
Cache Lingo Picture



Exempel Nehalem i7 (one example)



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.

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)
- Typically, several levels of caches

- Caches are most important target for optimizations

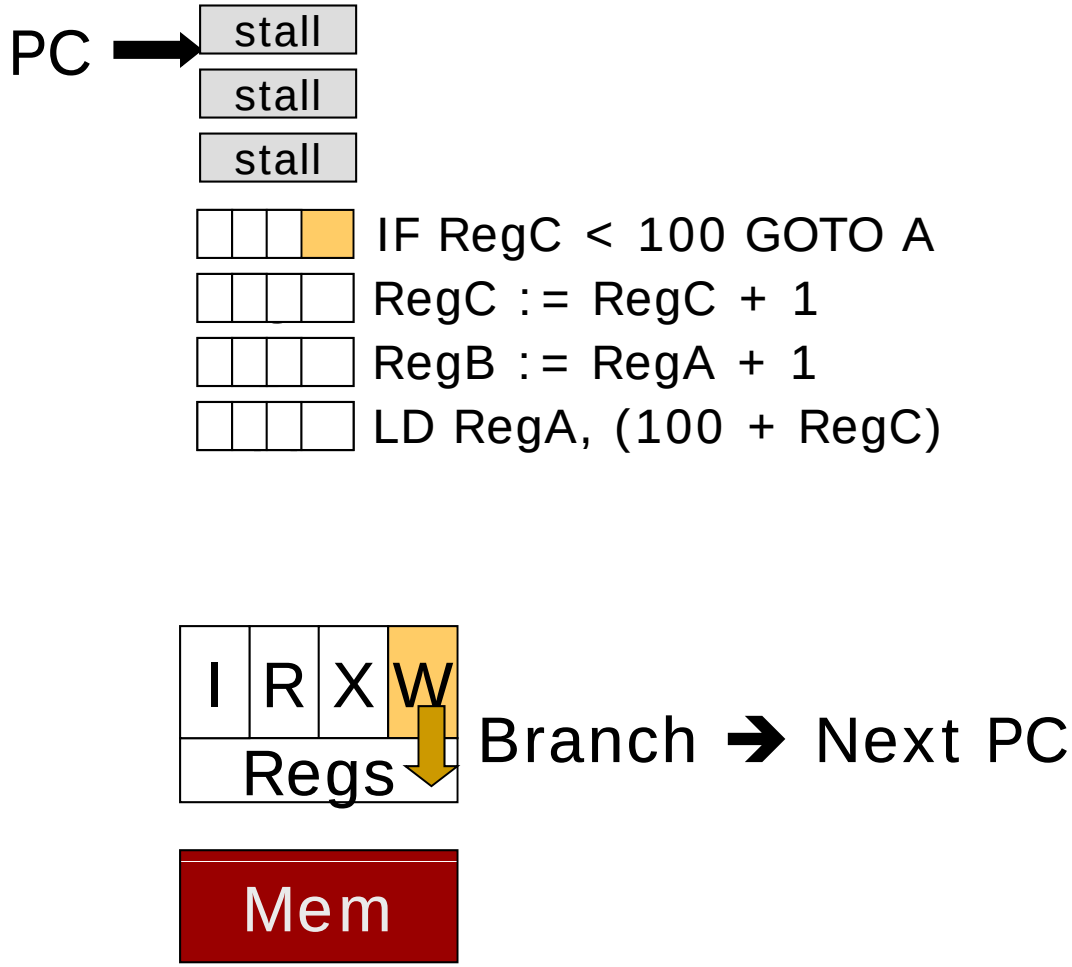
How are we doing?

- 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

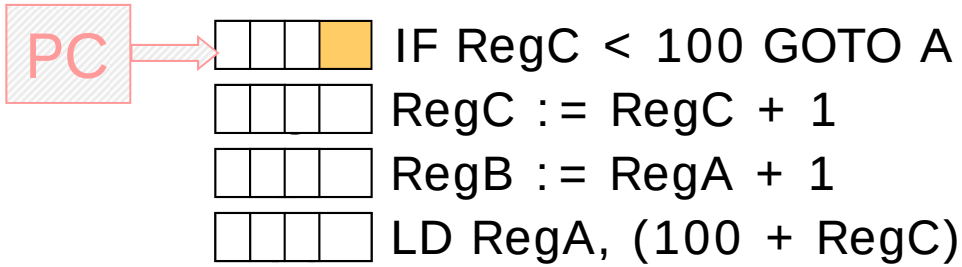
Speed up calculation of Next PC

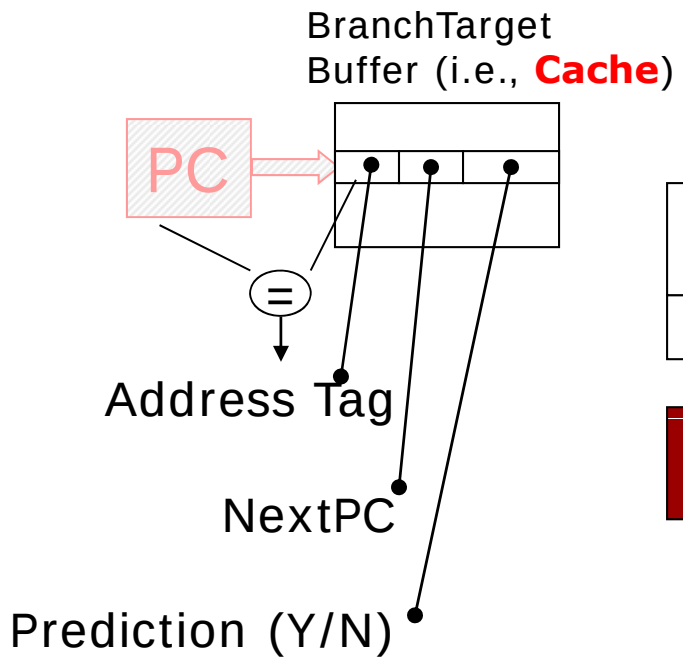


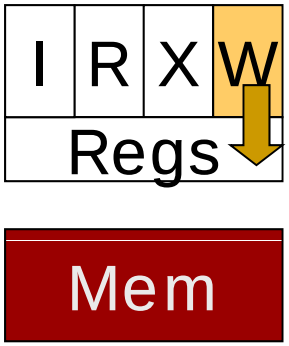
Branch Predictor Based on History

PC →

Guess the next PC here!!





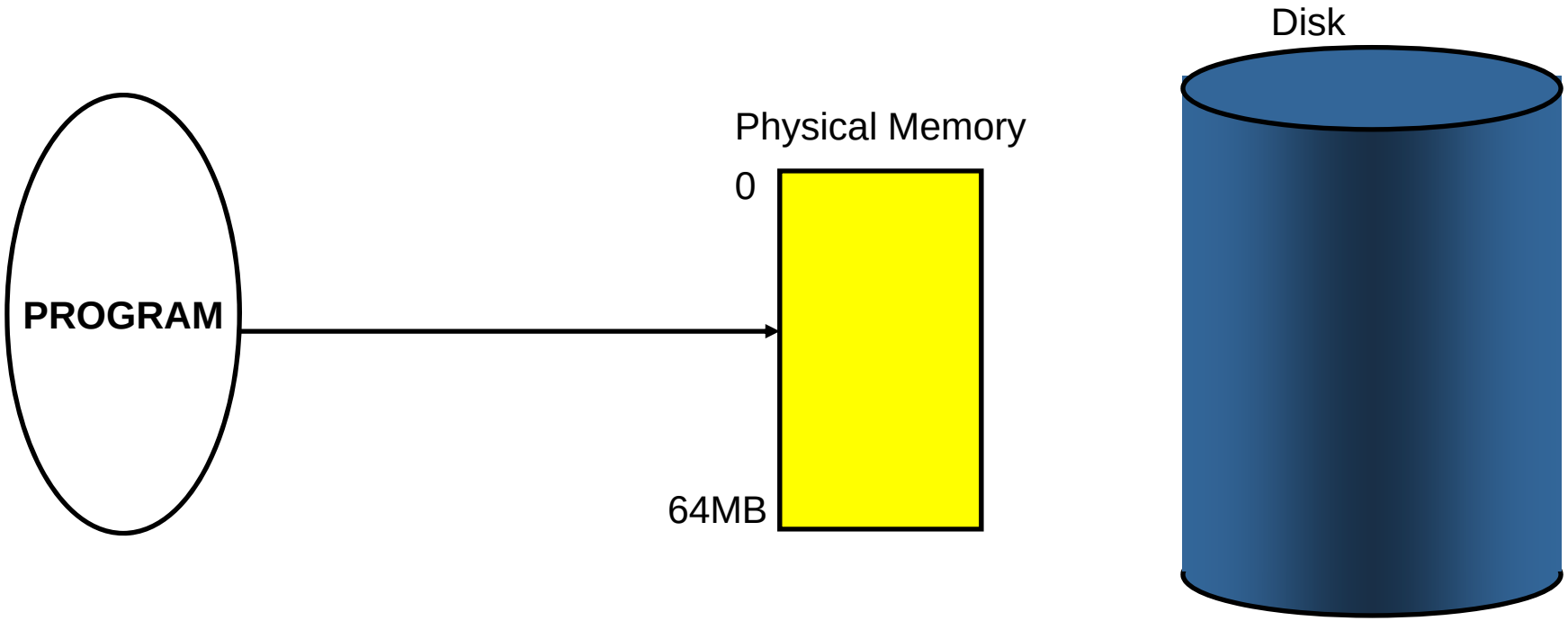


Branch ➔ Next PC

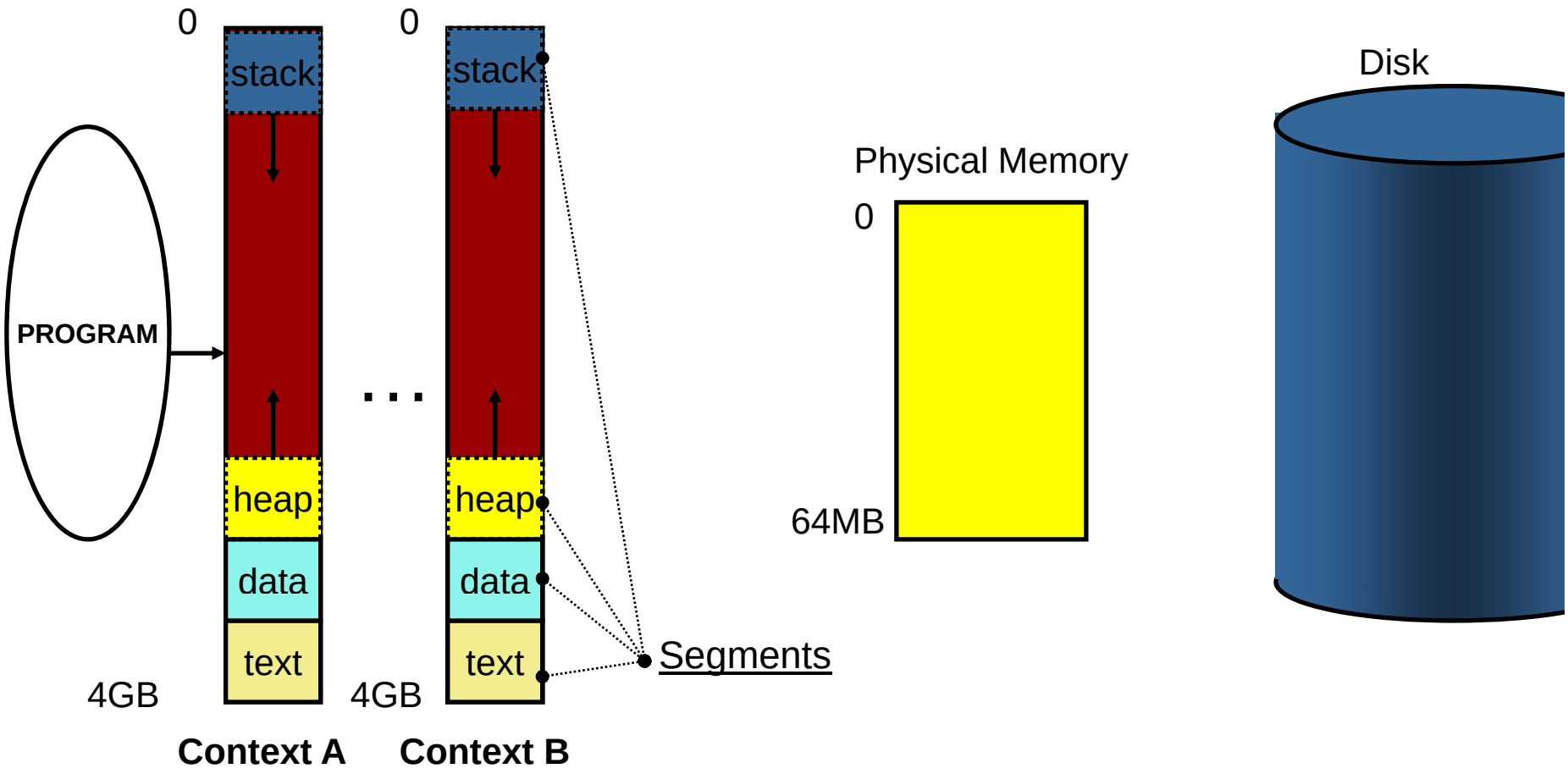
Virtual Memory System

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

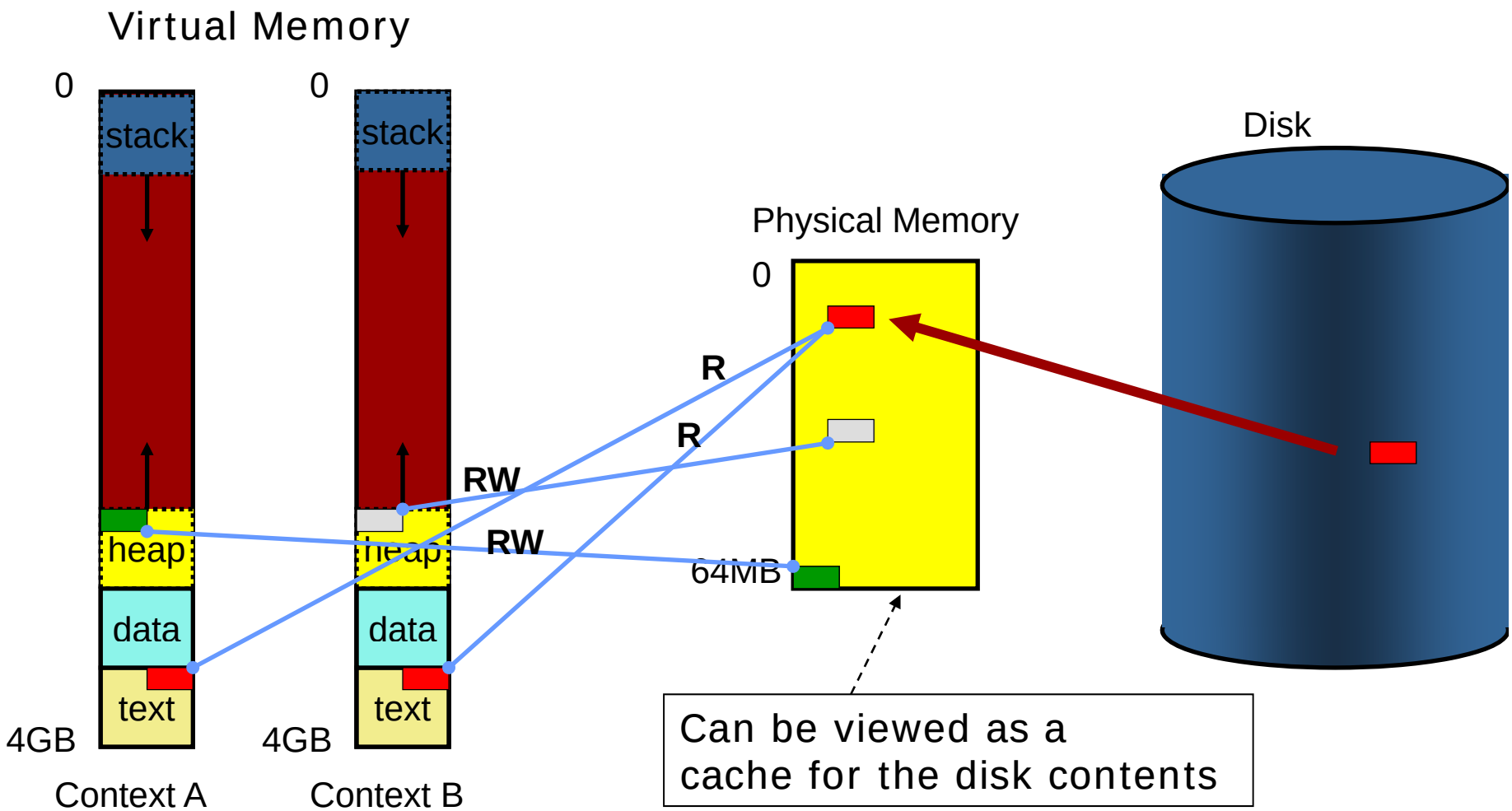
Physical Memory



Virtual and Physical Memory



Translation & Protection

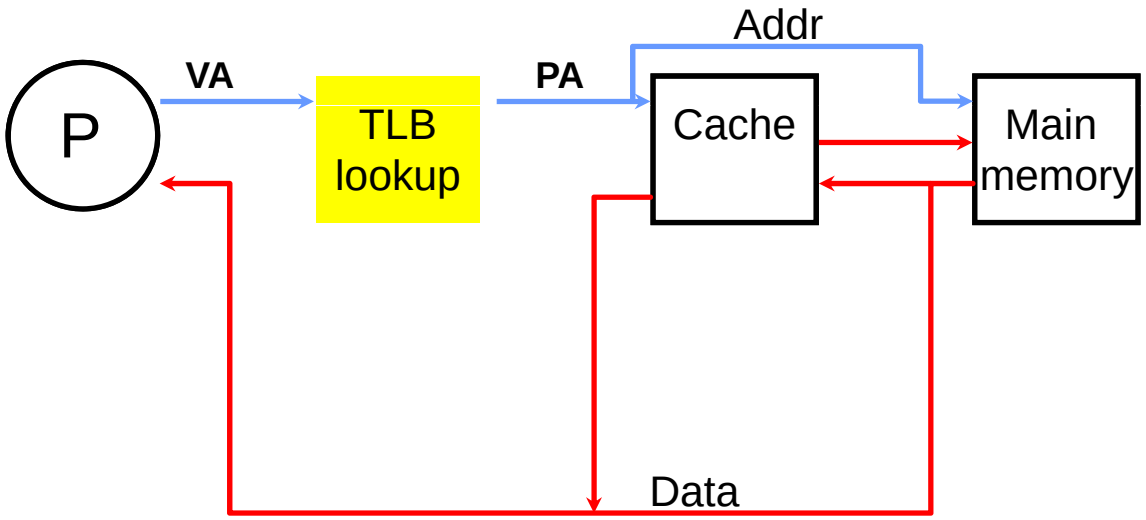


Fast address translation

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)

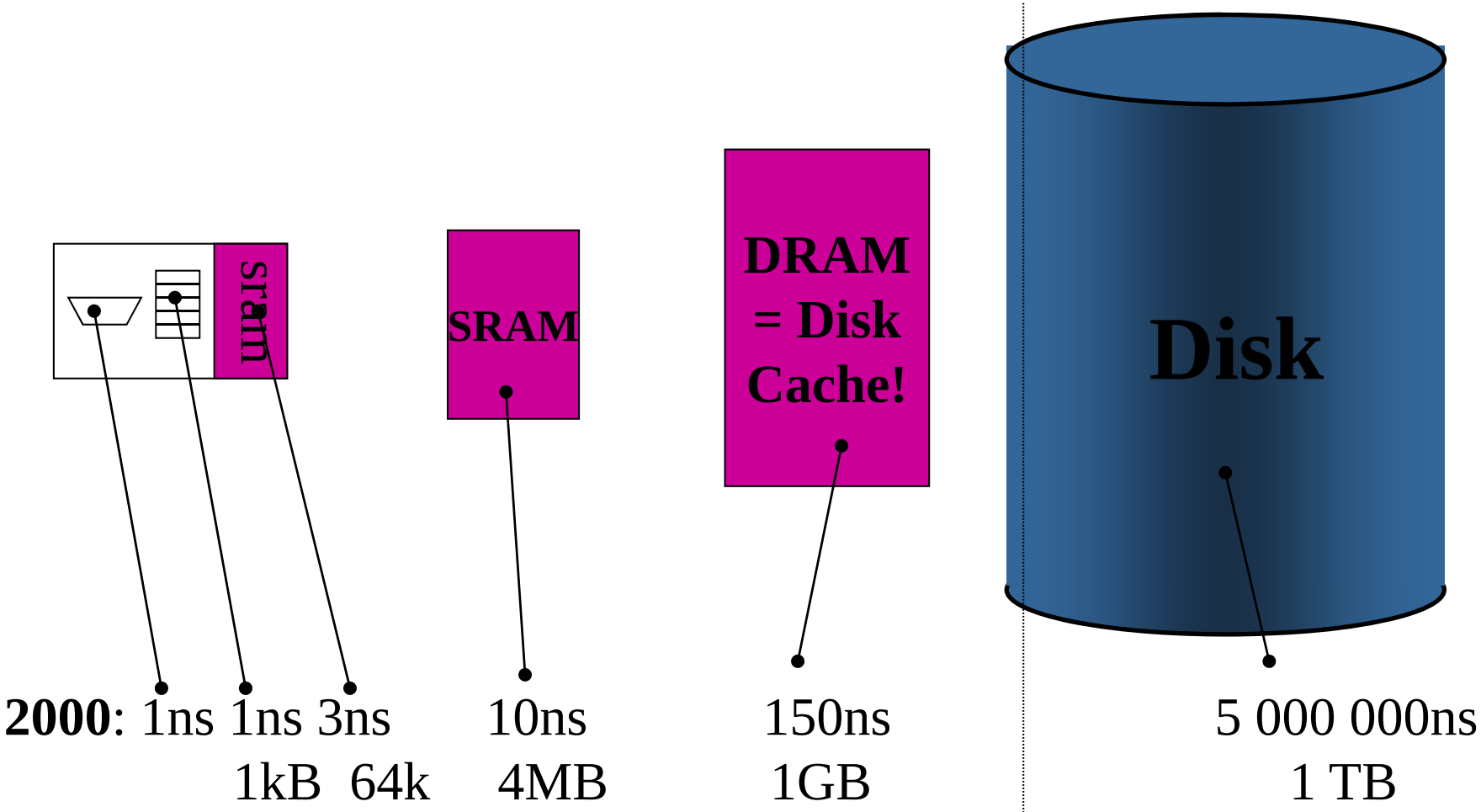
  ==> *The caches rears their ugly faces again!*



Disks/DRAM Memory -- yet another cache

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

Memory/storage



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

Caches Everywhere…

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