Caches and more caches
or
spam, spam, spam and spam

Erik Hagersten

Uppsala University, Sweden

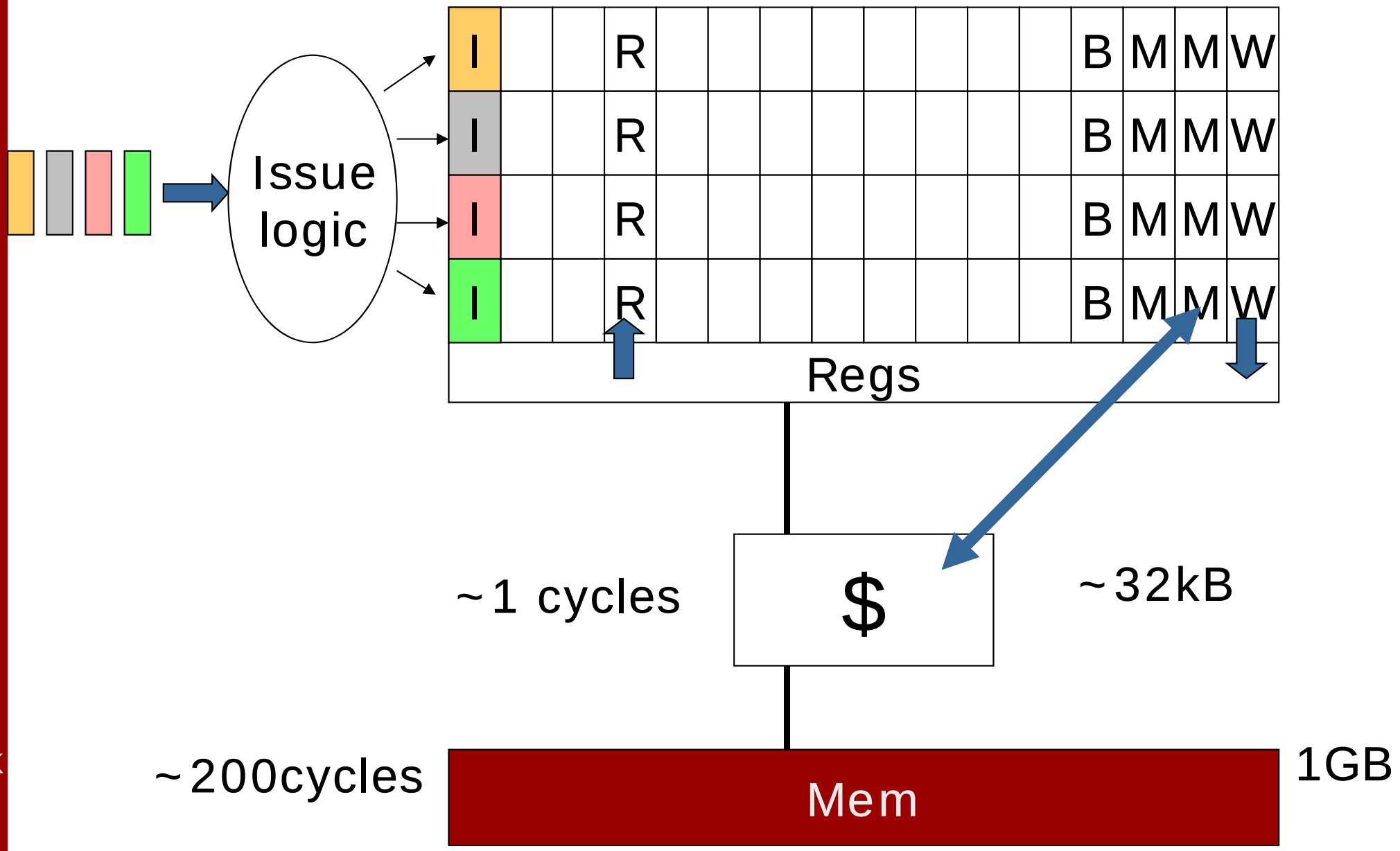
eh@it.uu.se

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

Fix 5: Use a cache



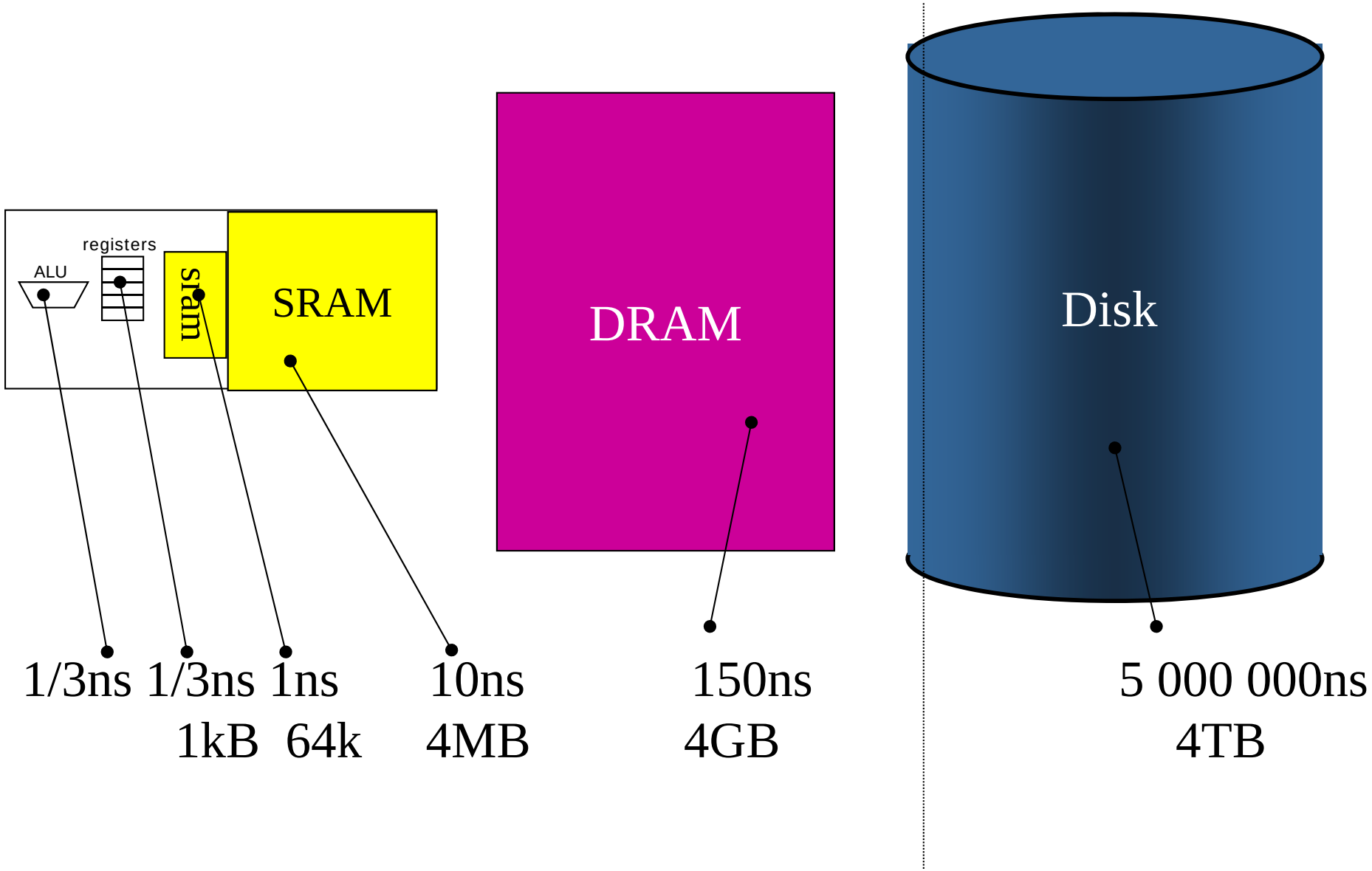
UPPSALA UNIVERSITET

Cache knowledge useful when...

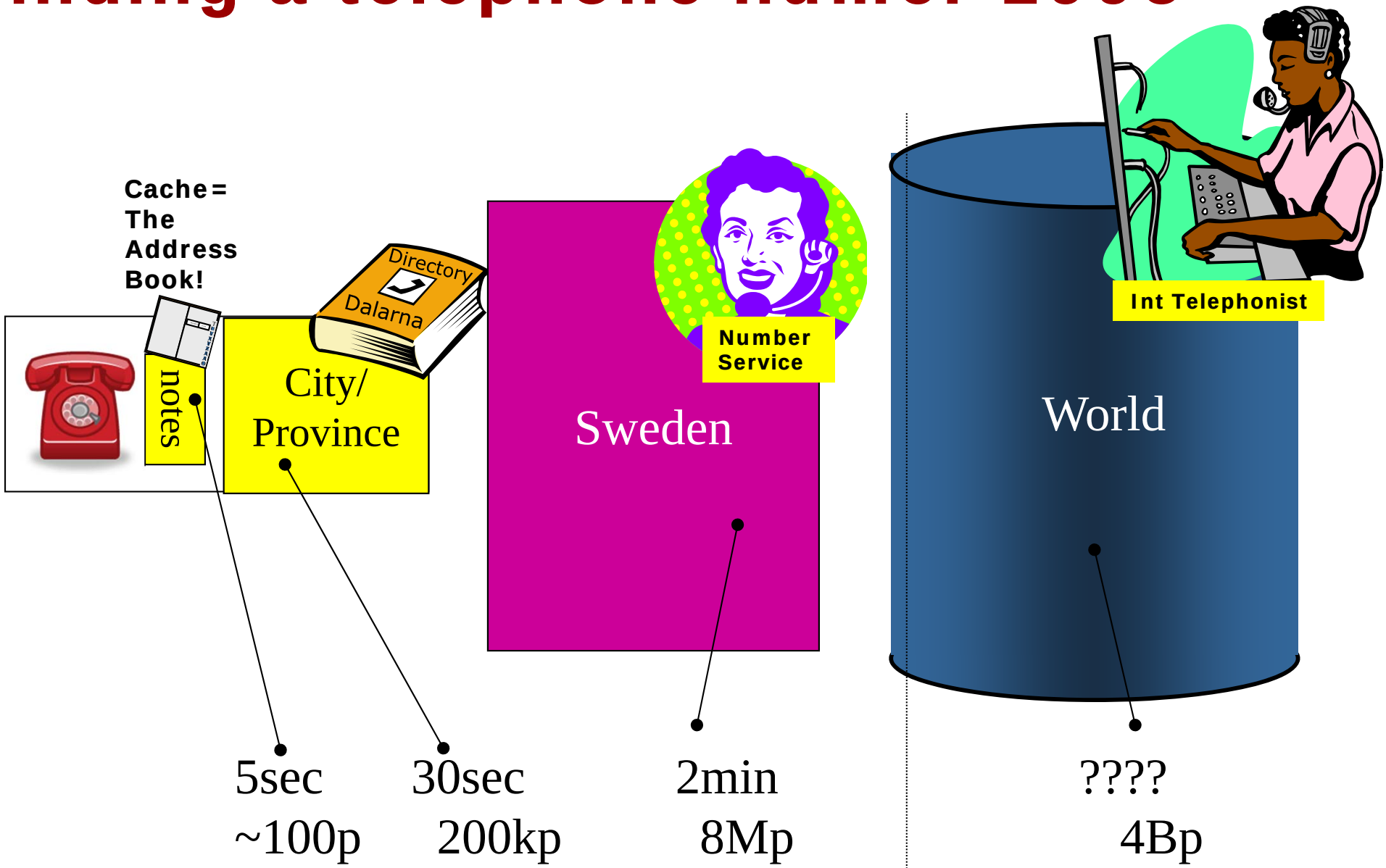
- Designing a new computer
- Writing an optimized program
  - or compiler
  - or operating system …
- Implementing software caching
  - Web caches
  - Proxies
  - File systems

AVDARK 2013

Memory hierarchy

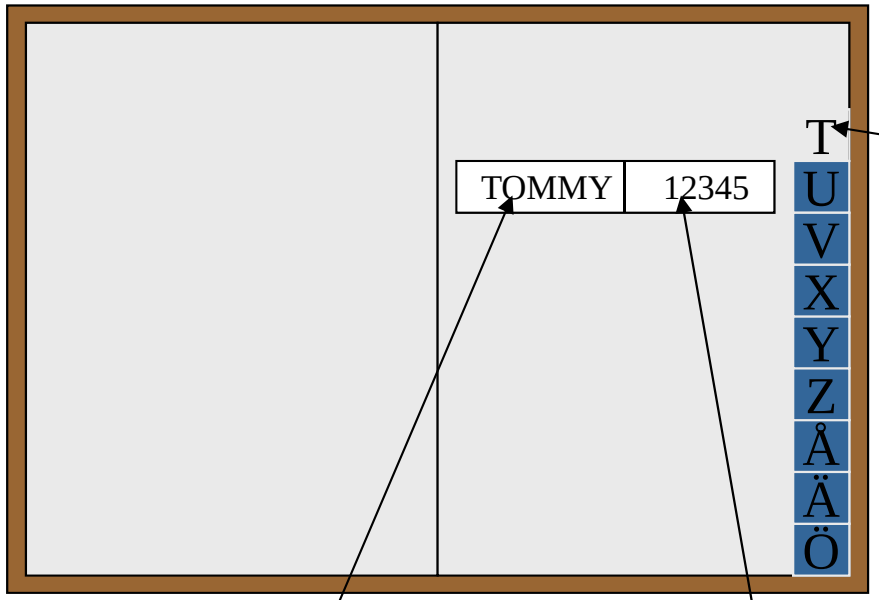


Finding a telephone numer 1995



Address Book Cache

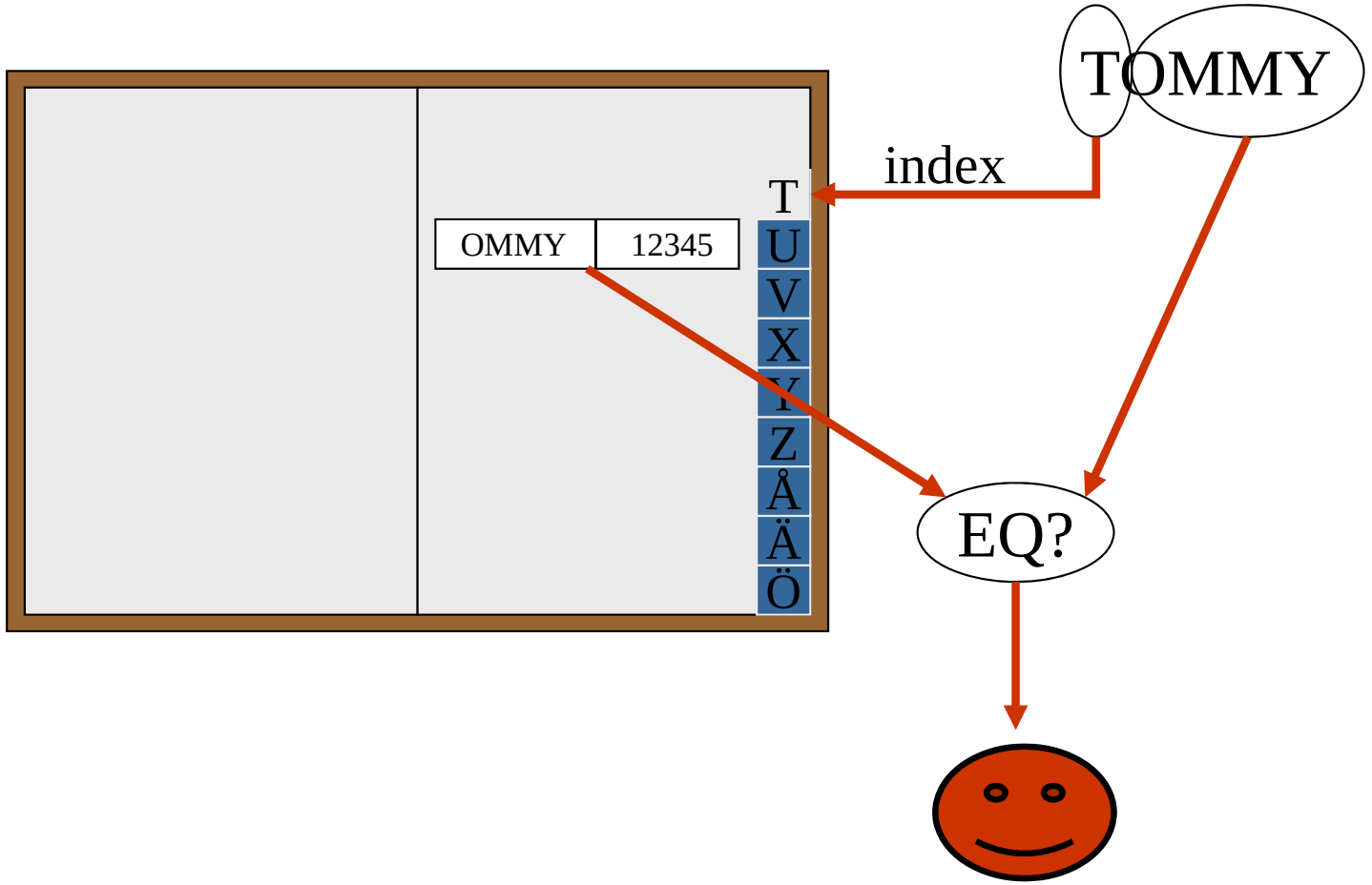
Looking for Tommy's Telephone Number



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

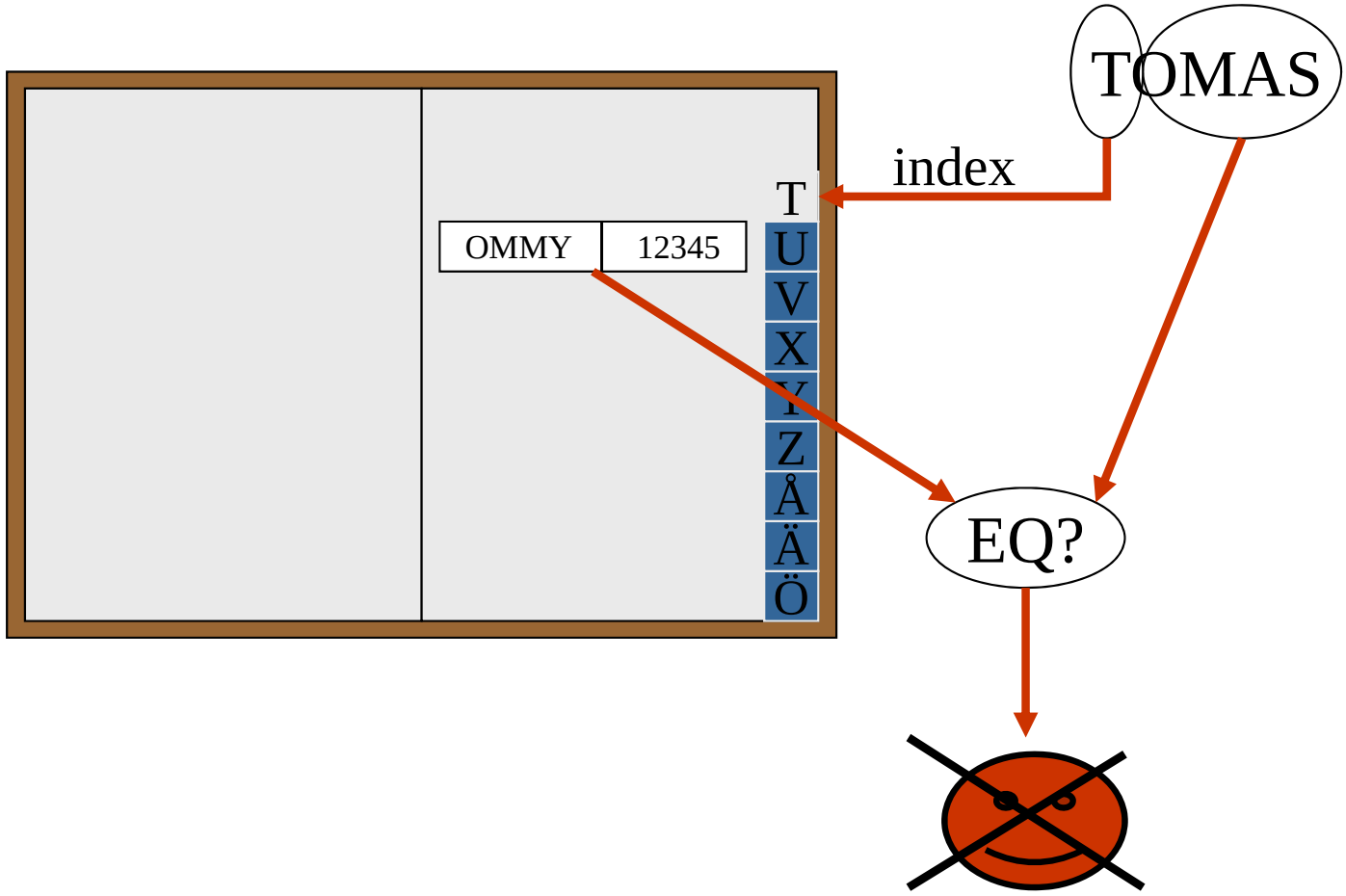
Address Book Cache

Looking for Tommy's Number



Address Book Cache

Looking for Tomas' Number

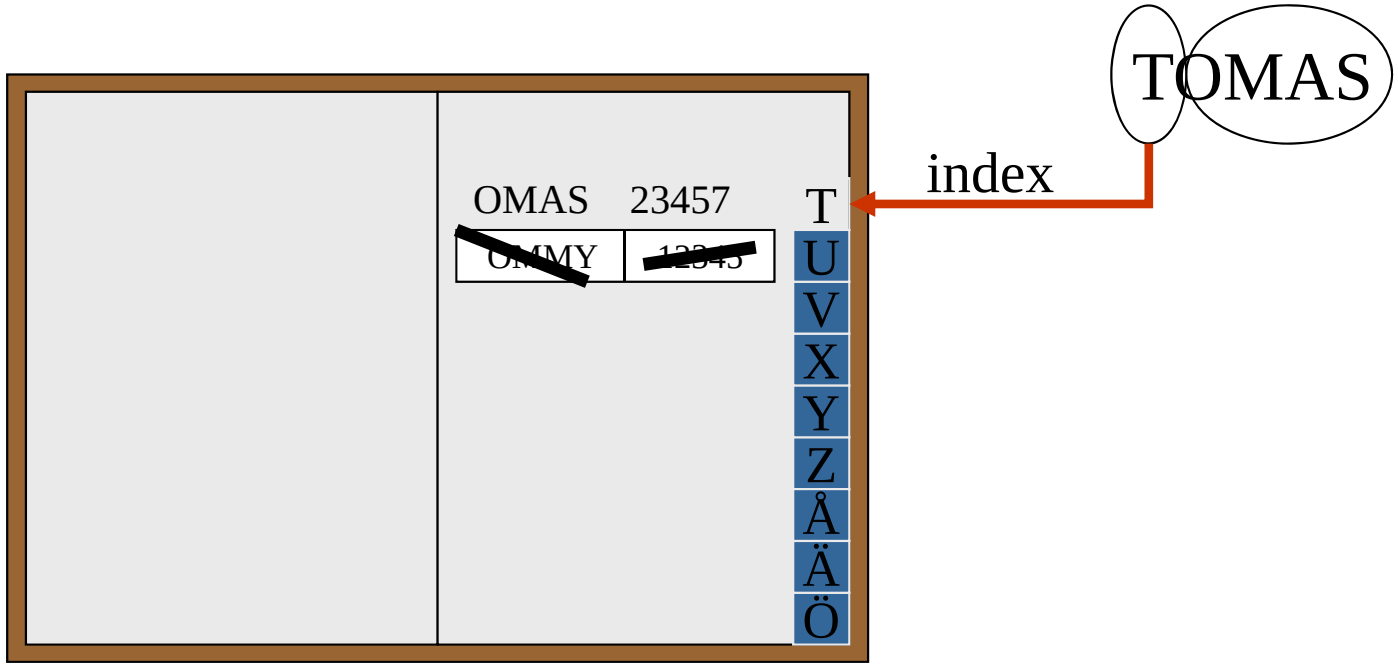


Miss!

Lookup Tomas' number in the telephone directory

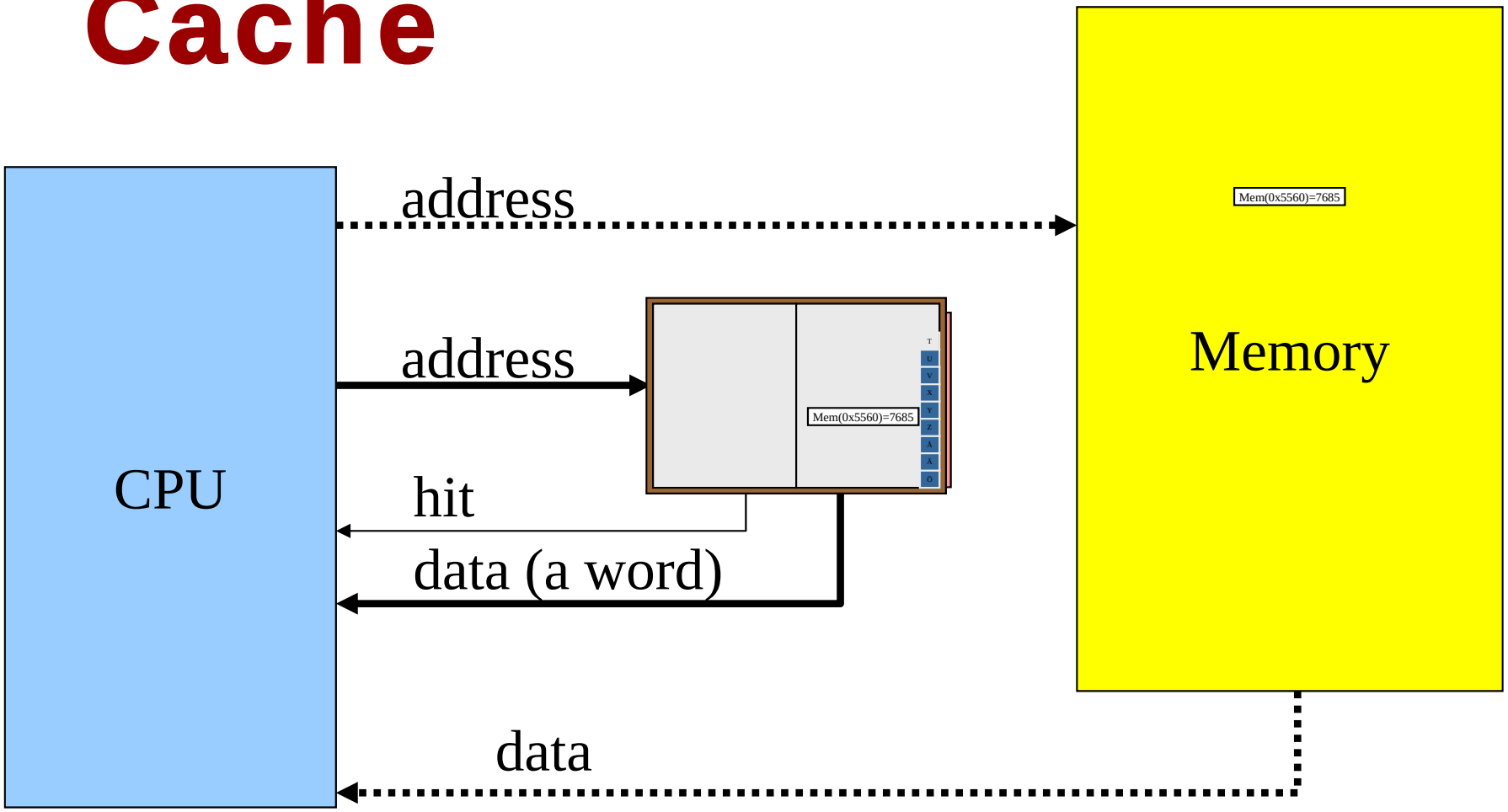
Address Book Cache

Looking for Tomas' Number



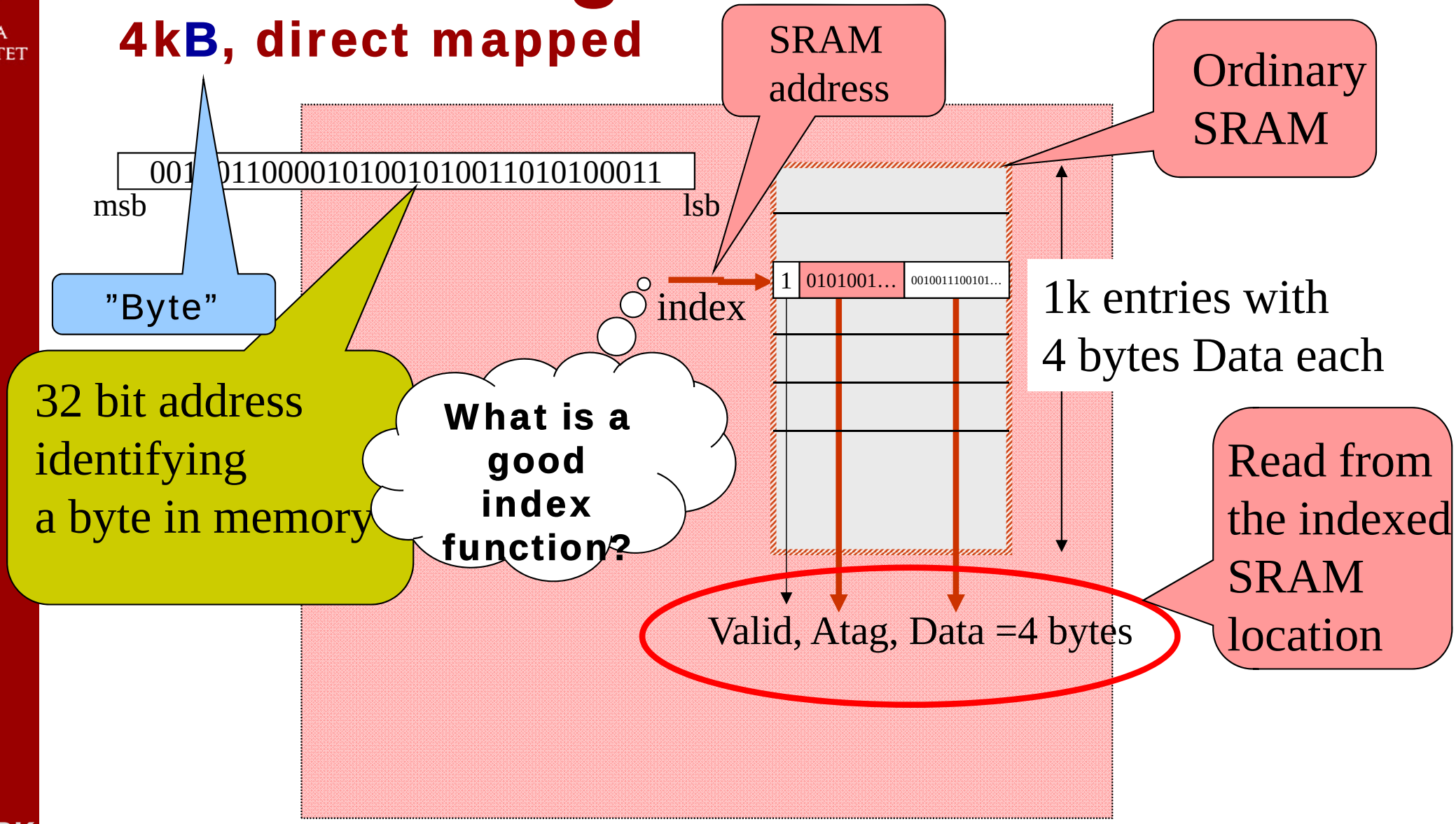
Replace TOMMY's data with TOMAS' data.
There is no other choice (direct mapped)

Cache

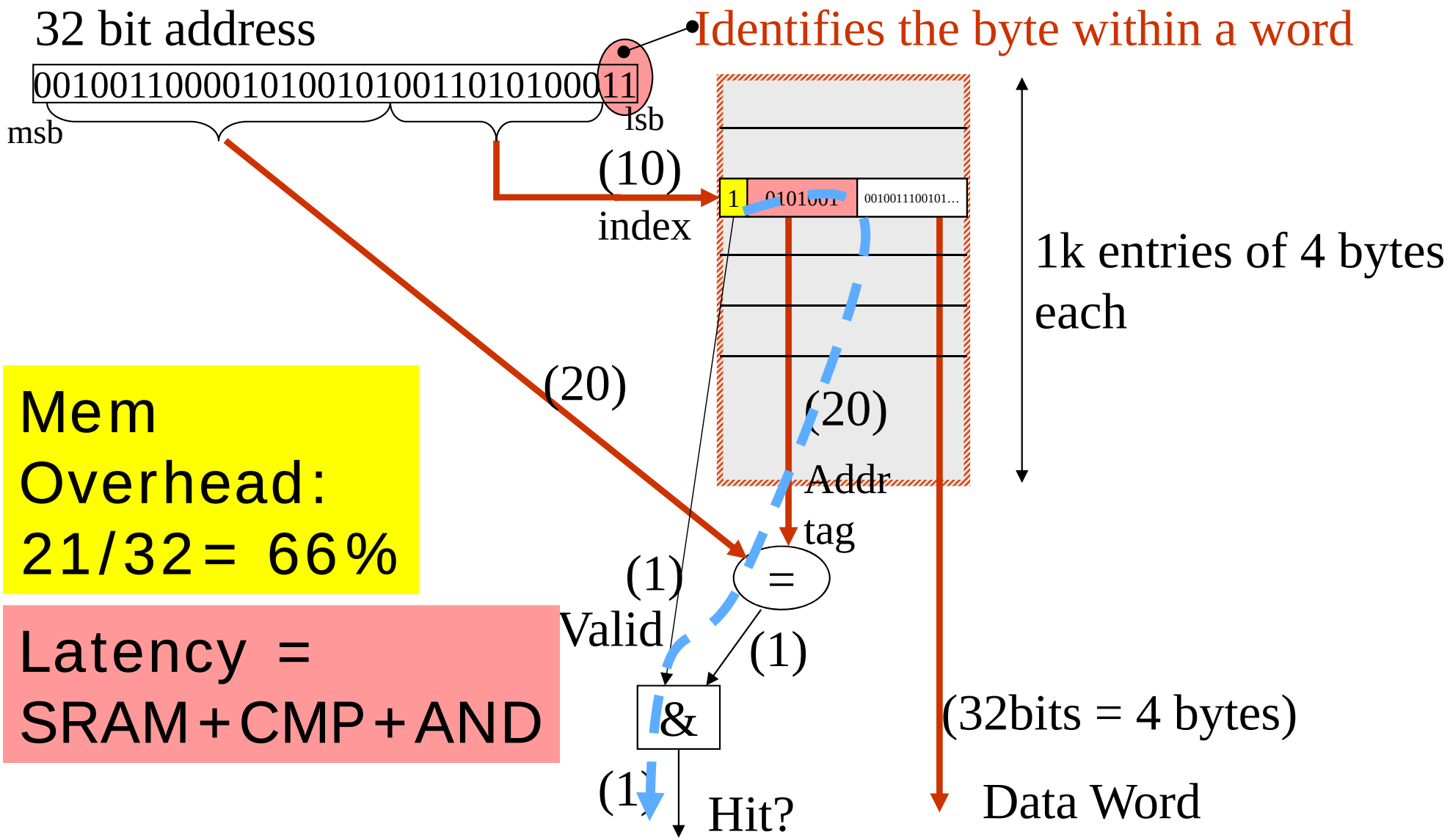

address
Mem(0x5560)=7685
Memory
address
CPU
Mem(0x5560)=7685
hit
data (a word)
data

Cache Organization

4kB, direct mapped



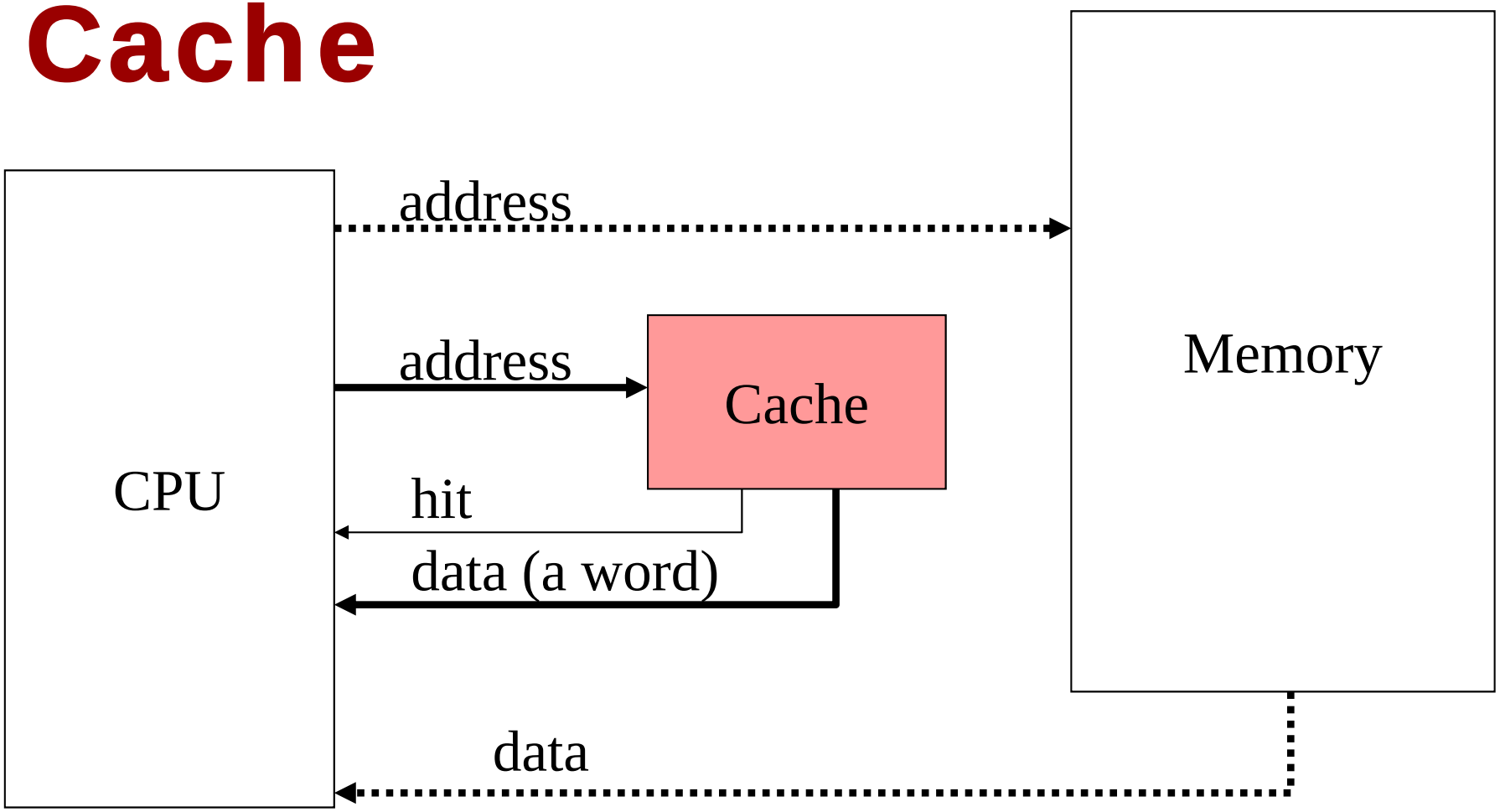
Cache Organization (really) 4kB, direct mapped



Mem Overhead: 21/32 = 66%

Latency = SRAM + CMP + AND

Cache



Hit: Use the data provided from the cache

Not-Hit: Use data from memory and also store it in the cache

Improving Caches

Erik Hagersten
Uppsala University

UPPSALA UNIVERSITET

Cache performance parameters

- Cache “hit rate” [%]
- Cache “miss rate” [%] (= 1 - hit_rate)
- Hit time [CPU cycles]
- Miss time [CPU cycles]
- Hit bandwidth
- Miss bandwidth
- Write strategy
- ….

UPPSALA UNIVERSITET

How to rate architecture performance?

Marketing:

- Frequency, Number of “cores”, …

Architecture “goodness”:

- CPI = Cycles Per Instruction
- IPC = Instructions Per Cycle

Benchmarking:

- SPEC-fp, SPEC-int, …
- TPC-C, TPC-D, …
- Varning: Using a unrepresentative benchmarks can be missleading

UPPSALA UNIVERSITET

Cache performance example

Assumption:

Infinite bandwidth

A perfect 1.0 Cycles-Per-Instruction [CPI] CPU

100% instruction cache hit rate

Total number of cycles =

#Instr. * ((1 - mem_ratio) * 1 +
mem_ratio * avg_mem_accesstime) =

= #Instr * ((1- mem_ratio) +
mem_ratio * (hit_rate * hit_time +
(1 - hit_rate) * miss_time)

CPI = 1 -mem_ratio +
mem_ratio * (hit_rate * hit_time +
(1 - hit_rate) * miss_time)

UPPSALA UNIVERSITET

Example Numbers

CPI = 1 - mem_ratio +
mem_ratio * (hit_rate * hit_time) +
mem_ratio * (1 - hit_rate) * miss_time)

mem_ratio = 0.25
hit_rate = 0.85
hit_time = 3
miss_time = 100

CPI = 0.75 + 0.25 * 0.85 * 3 + 0.25 * 0.15 * 100 =

0.75 + 0.64 + 3.75 = 5.14
CPU HIT MISS

AVDARK 2013

UPPSALA UNIVERSITET

What if ...

CPI = 1 -mem_ratio +
mem_ratio * (hit_rate * hit_time) +
mem_ratio * (1 - hit_rate) * miss_time)

mem_ratio = 0.25
hit_rate = 0.85
hit_time = 3
miss_time = 100

	CPU	HIT	MISS
== >	0.75	+ 0.64	+ 3.75 = 5.14

- Twice as fast CPU ==> 0.37 + 0.64 + 3.75 = 4.77
- Faster memory (70c) ==> 0.75 + 0.64 + 2.62 = 4.01
- Improve hit_rate (0.95) => 0.75 + 0.71 + 1.25 = 2.71

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-ideal cache implementation) (too many names starting with "H")
- (Multiprocessors)
  - Communication (imposed by communication)
  - False sharing (side-effect from large cache blocks)

UPPSALA UNIVERSITET

How to get more effective caches:

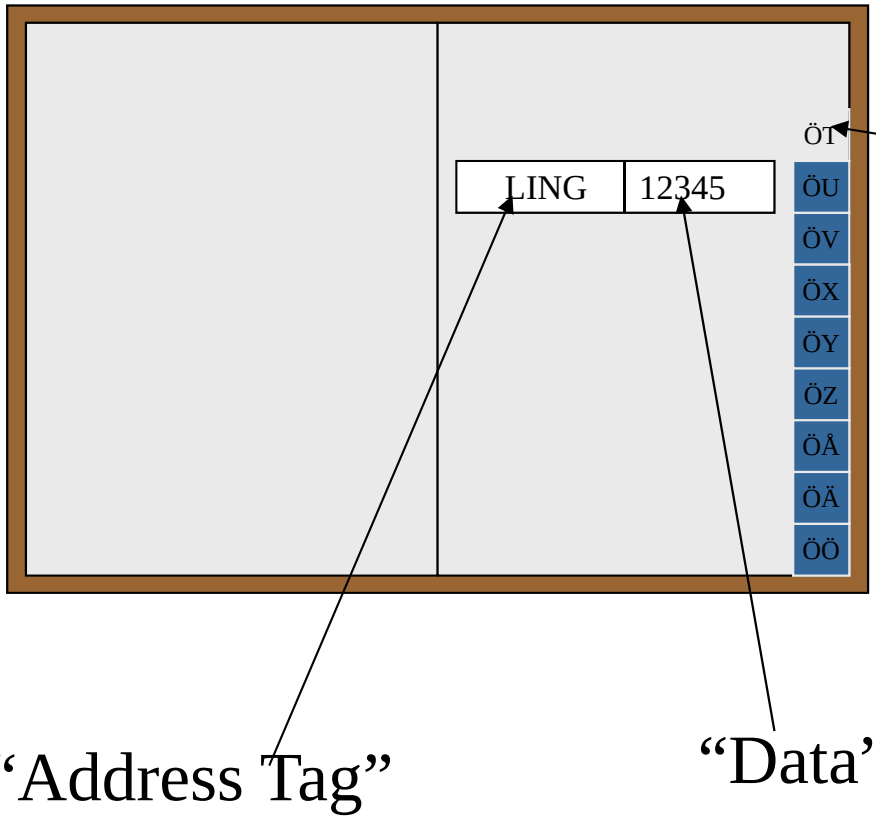
- Larger cache (more capacity)
- Cache block size (larger cache lines)
- More placement choice (more associativity)
- Innovative caches (victim, skewed, ...)
- Cache hierarchies (L1, L2, L3, CMR)
- Latency-hiding (weaker memory models)
- Latency-avoiding (prefetching)
- Cache avoiding (cache bypass)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

Avoiding Capacity Misses –

a huge address book

784 pages

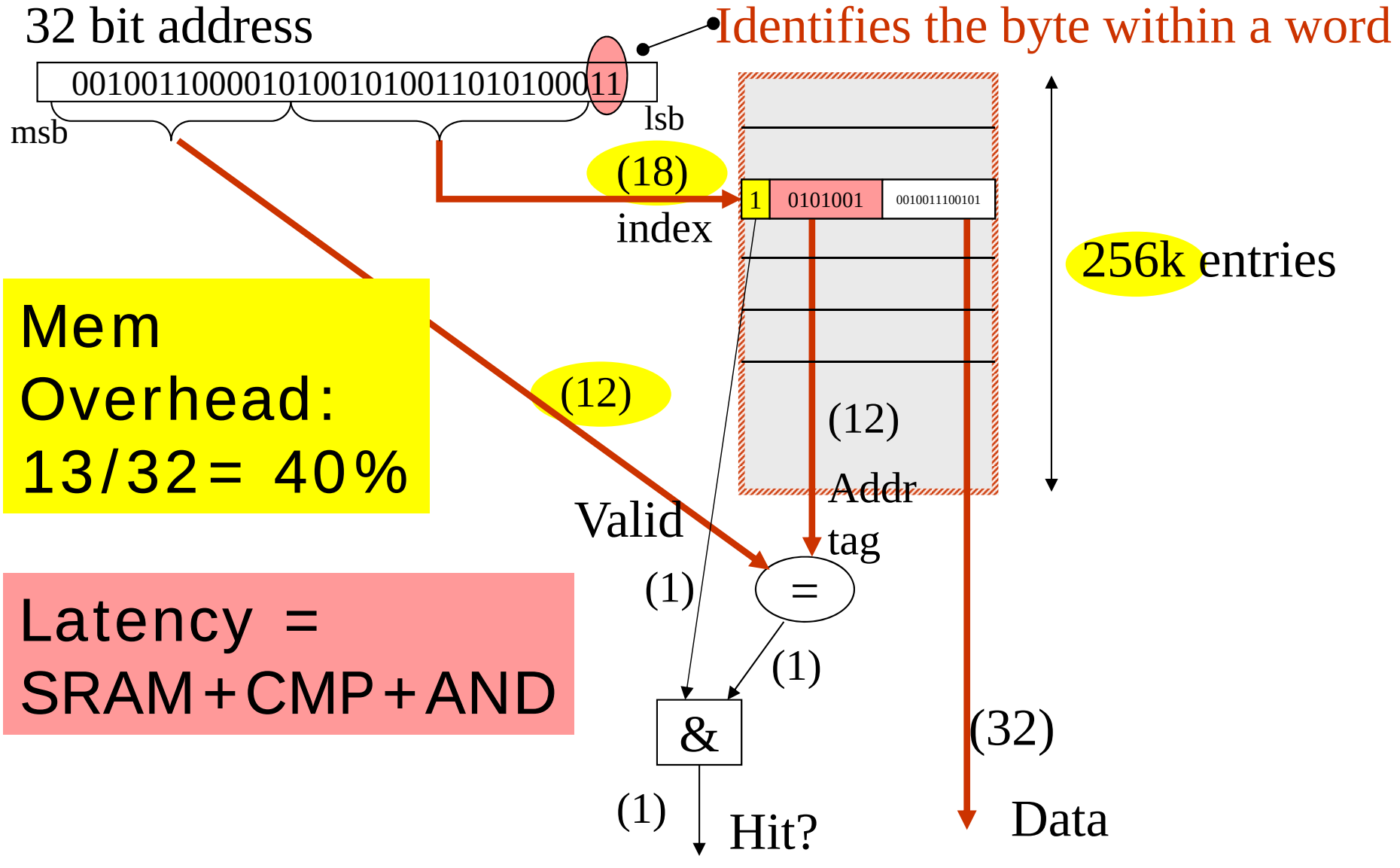
Here: still one entry per page , i.e., directed mapped cache



One entry per page =>

Direct-mapped caches with 28^2 entries ➔ 784 entries

Cache Organization
1MB, direct mapped



Mem Overhead: 13/32 = 40%

Latency = SRAM + CMP + AND

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 a “signals”

longer distances

-- Power consumption (static and dynamic)

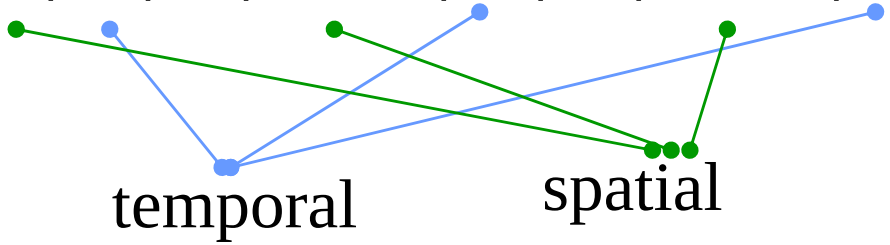
Why do you hit in a cache?

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

Typical access pattern:

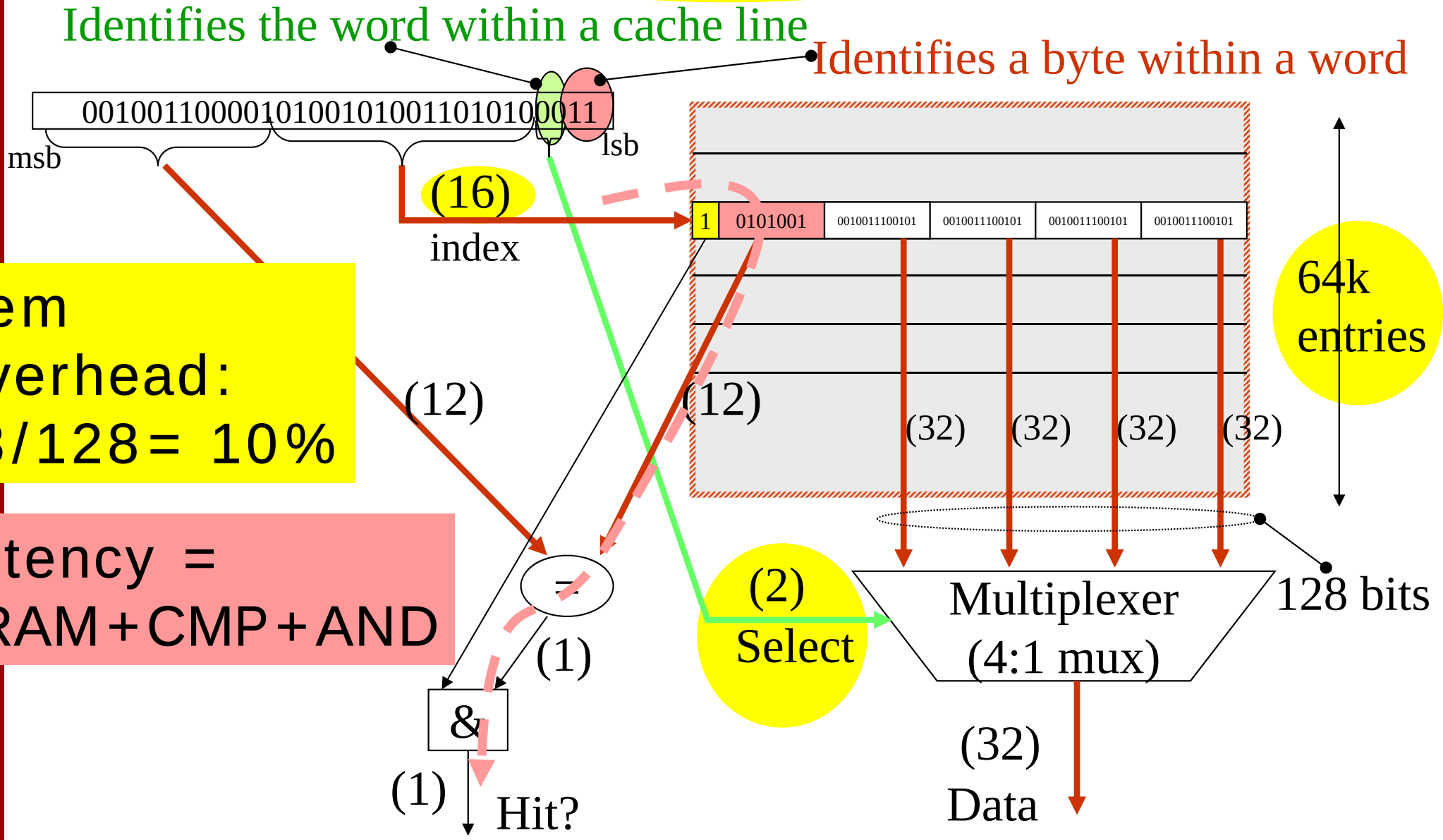
(inner loop stepping through an array)

A, B, C, A+1, B, C, A+2, B, C, …



Fetch more than a word to the cache: cache blocks(a.k.a cache lines)

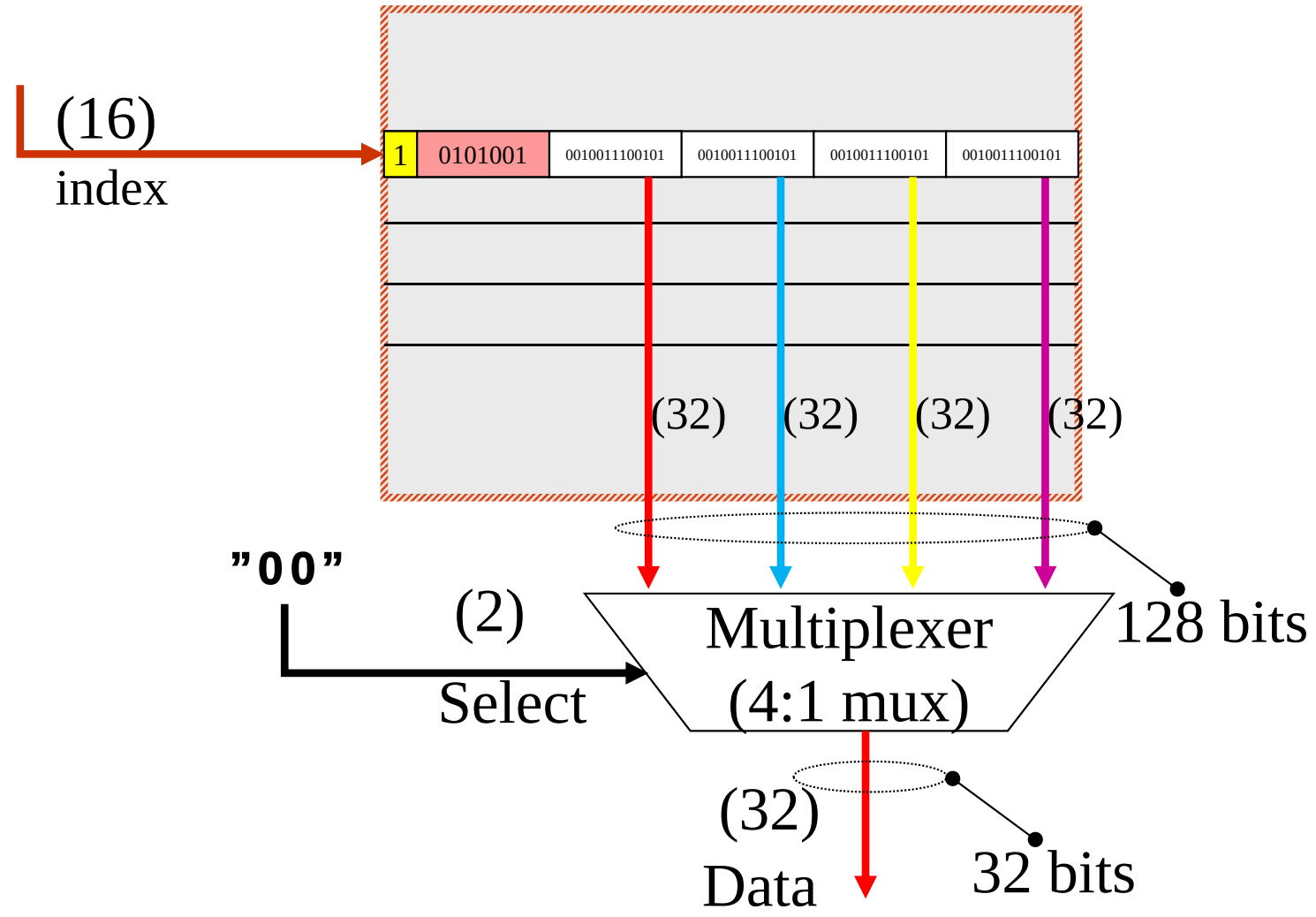
1MB, direct mapped, CacheLine=16B



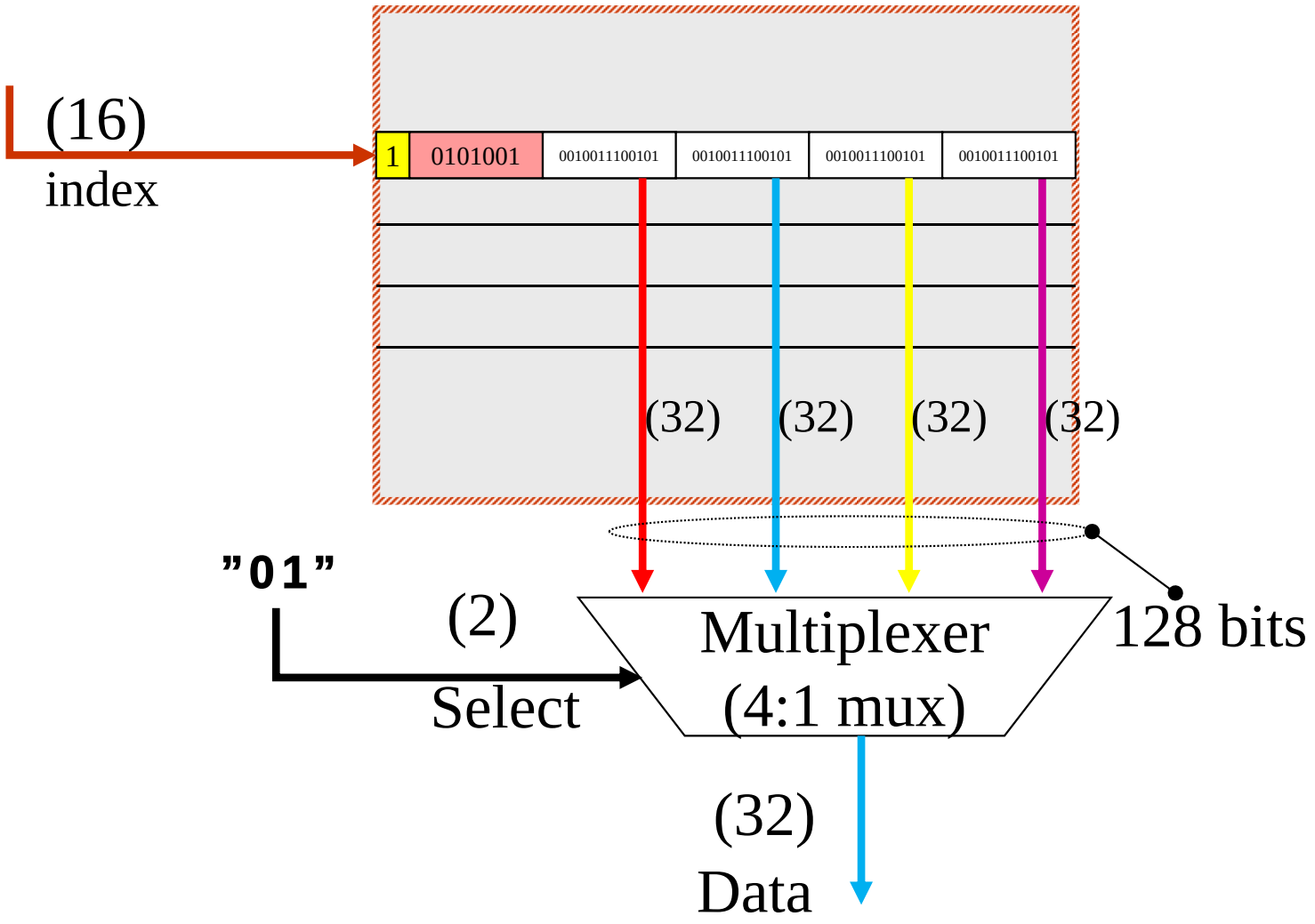
Mem Overhead: 13/128 = 10%

Latency = SRAM + CMP + AND

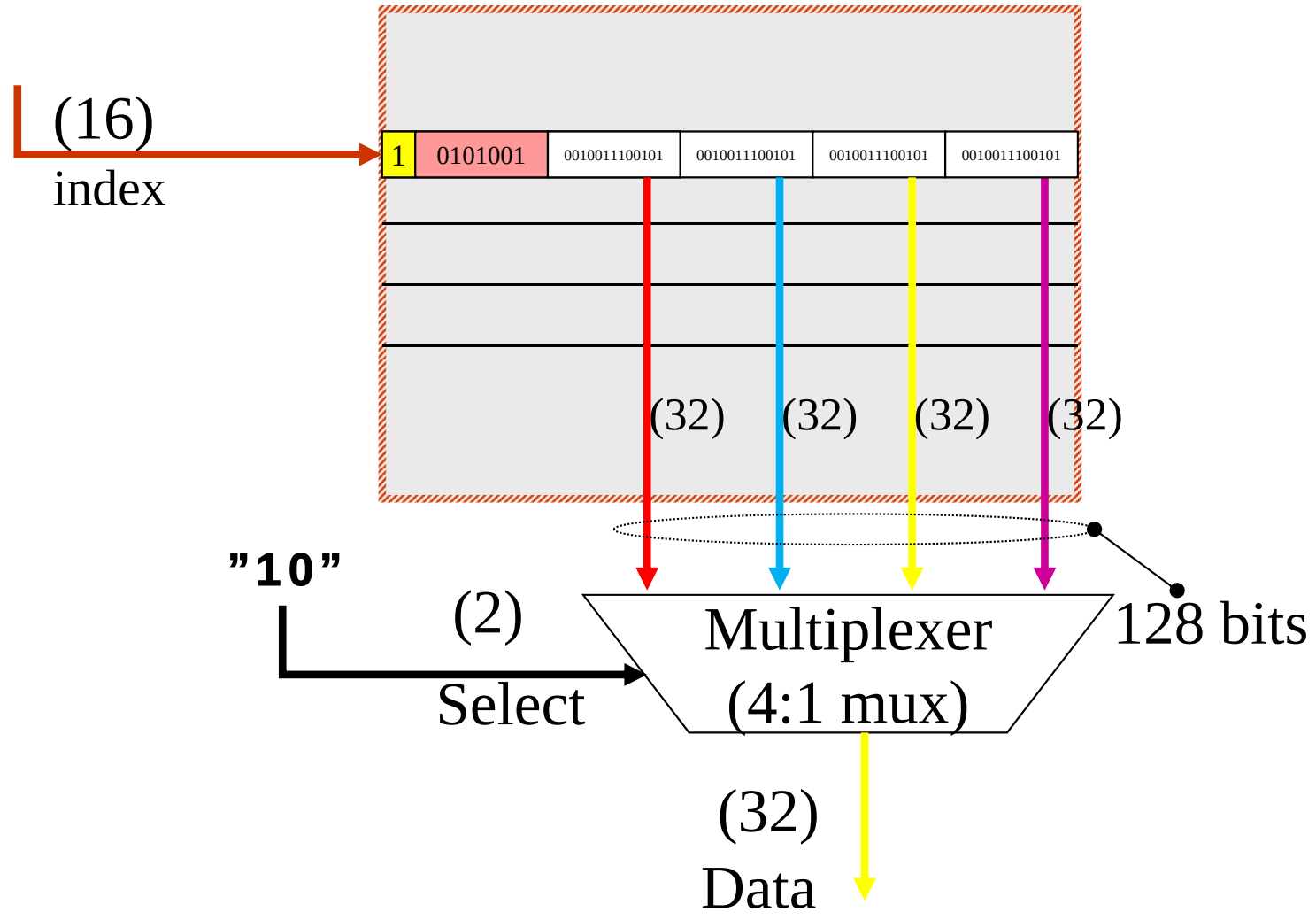
Multiplexer, or Mux for short



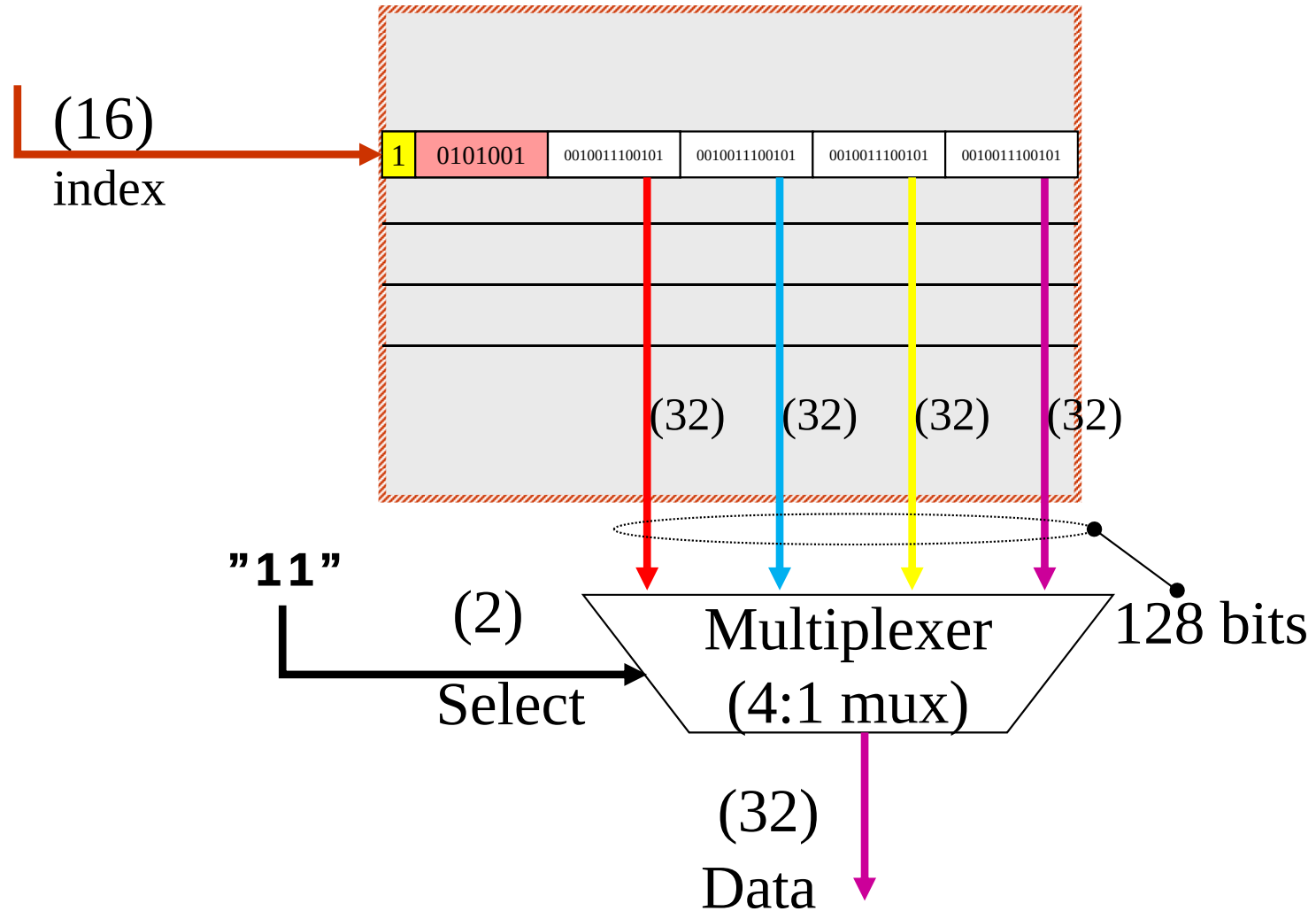
Multiplexer, or Mux for short



Multiplexer, or Mux for short



Multiplexer, or Mux for short



Pros/Cons Large Cache Lines

++ Explores spatial locality

++ Fits well with modern DRAMs

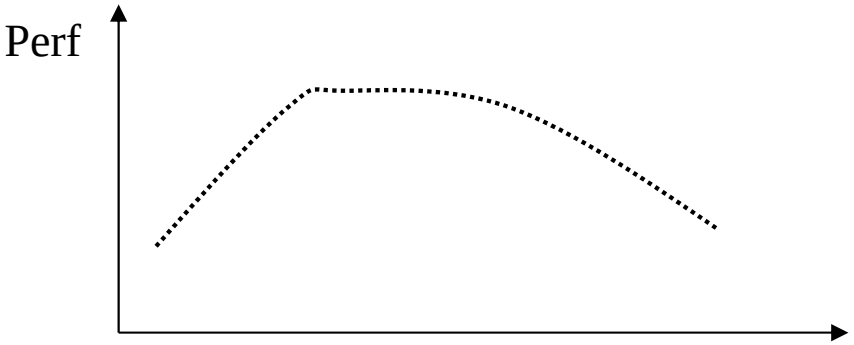
  * first DRAM access slow

  * subsequent accesses fast ("page mode")

-- Poor usage of SRAM & BW for some patterns

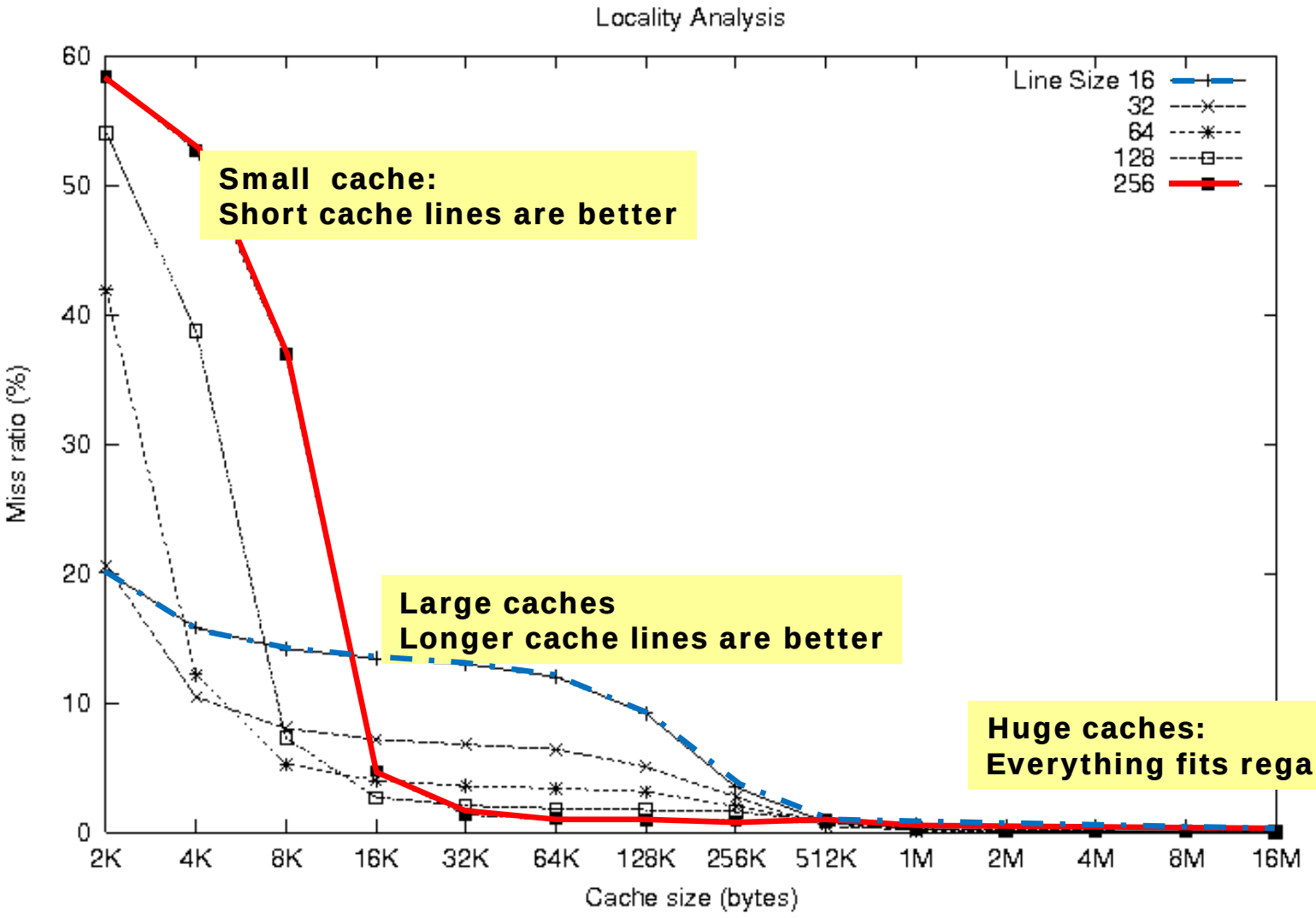
-- Higher miss penalty (fix: critical word first)

-- (False sharing in multiprocessors)



UART: StatCache Graph

matrix multiply



Small cache:
Short cache lines are better

Large caches
Longer cache lines are better

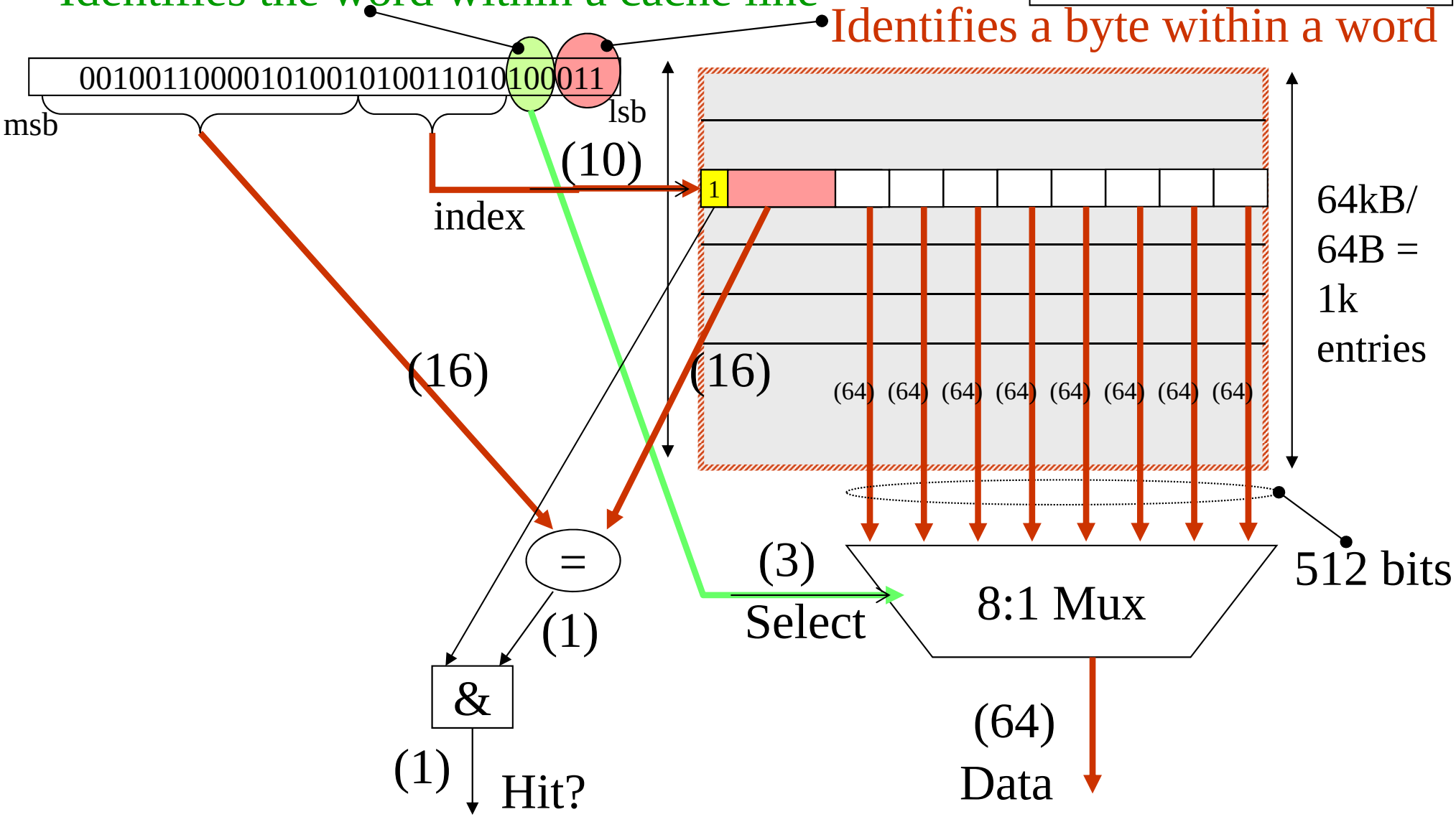
Huge caches:
Everything fits regardless of CL size

Note: this is just a single example, but the conclusion typically holds for most applications.

Thanks: Dr. Erik Berg

Let's build one together

Cache size = 64kB
Cache line = 64B
Word size = 8B
32 bit address



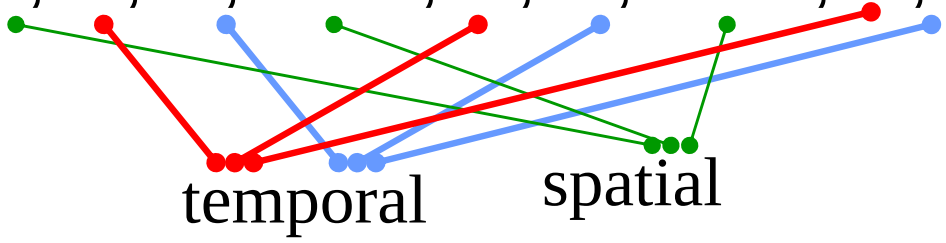
UPPSALA UNIVERSITET

Cache Conflicts

Typical access pattern:

(inner loop stepping through an array)

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

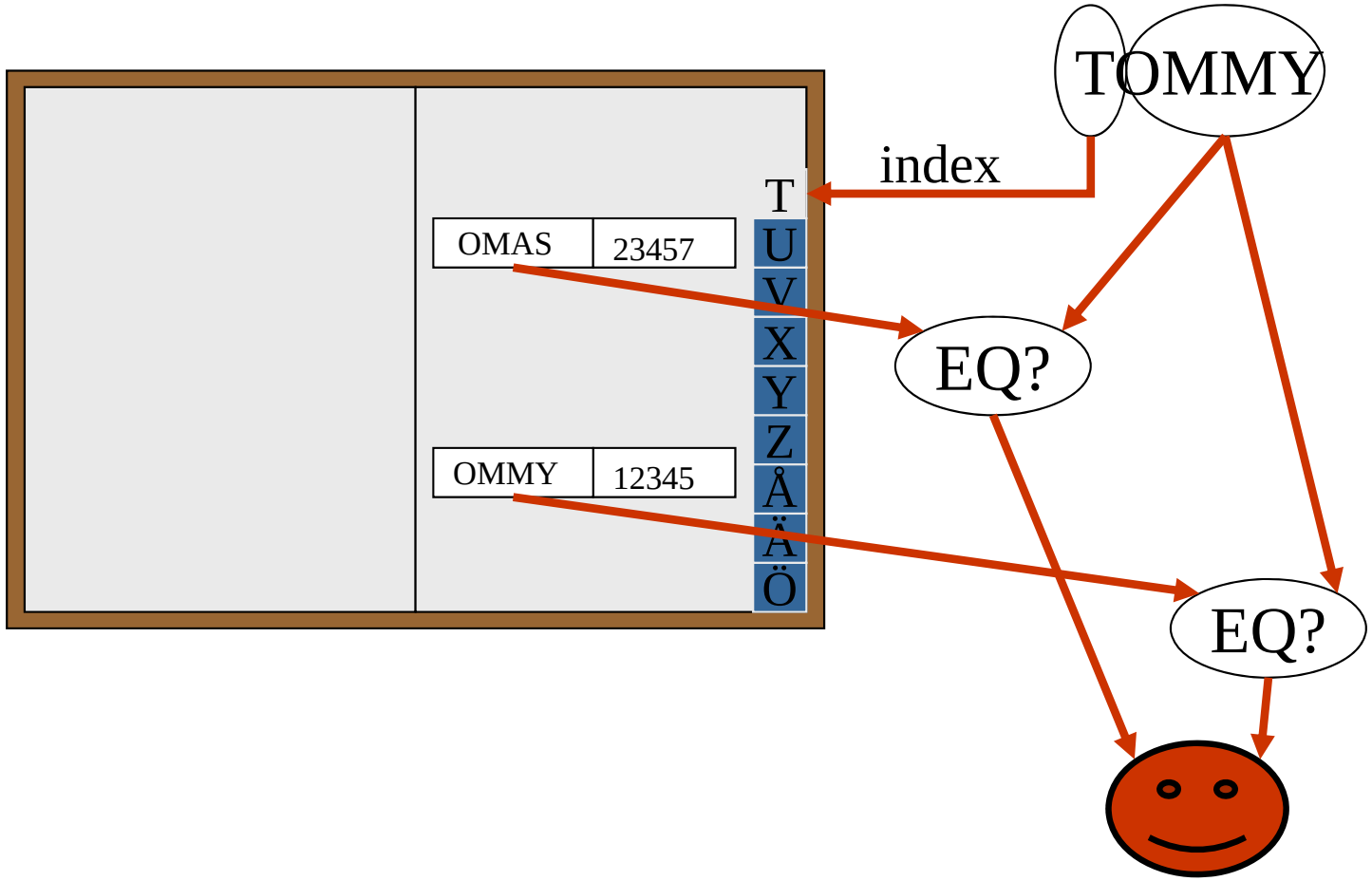


What if B and C index to the same cache location

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

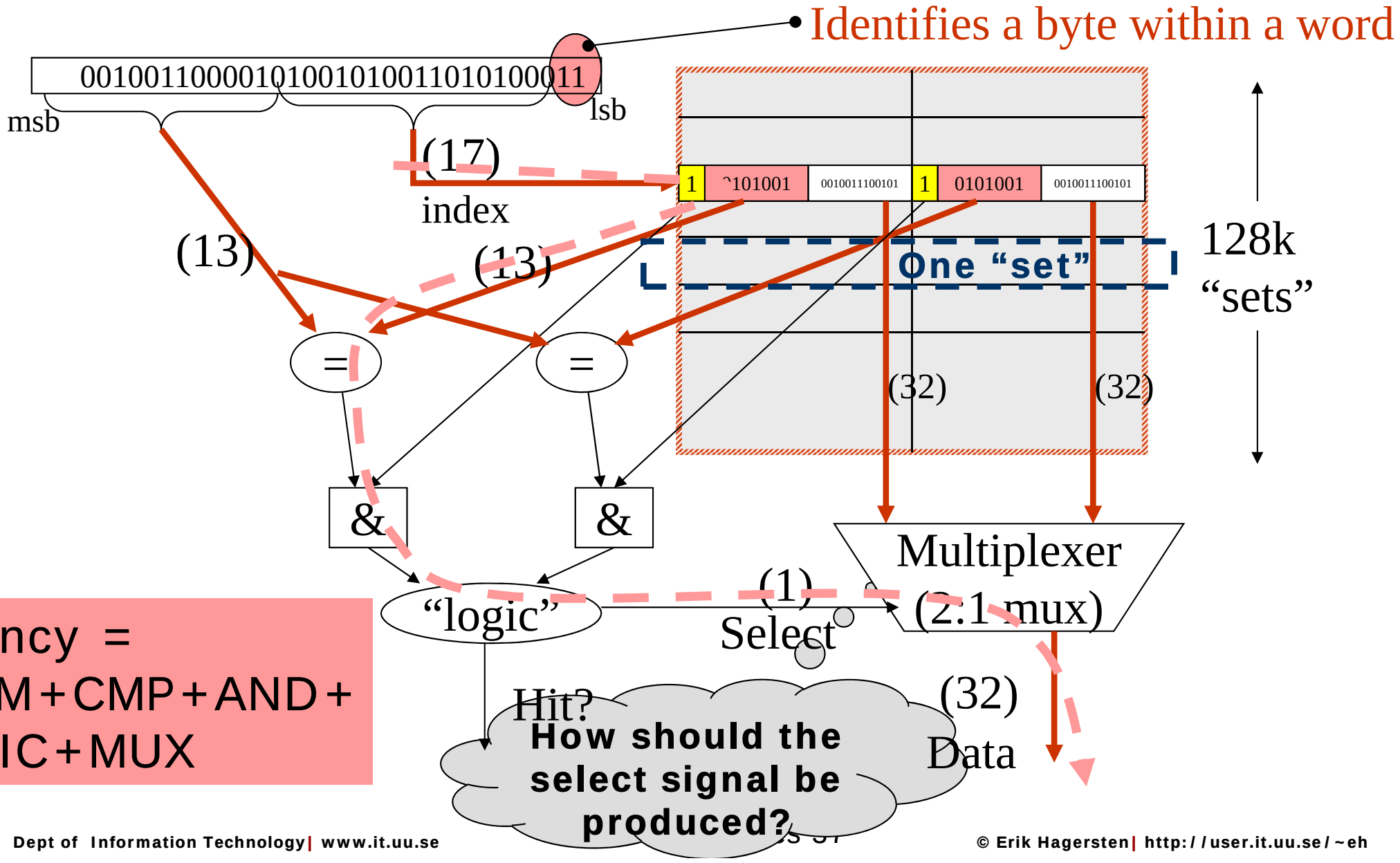
Address Book Cache

Two names per page: index first, then search.



Avoiding conflict: More associativity

1MB, 2-way set-associative, CL=4B



Latency = SRAM+CMP+AND+ LOGIC+MUX

UPPSALA UNIVERSITET

Pros/Cons Associativity

++ Avoids conflict misses

-- Slower access time

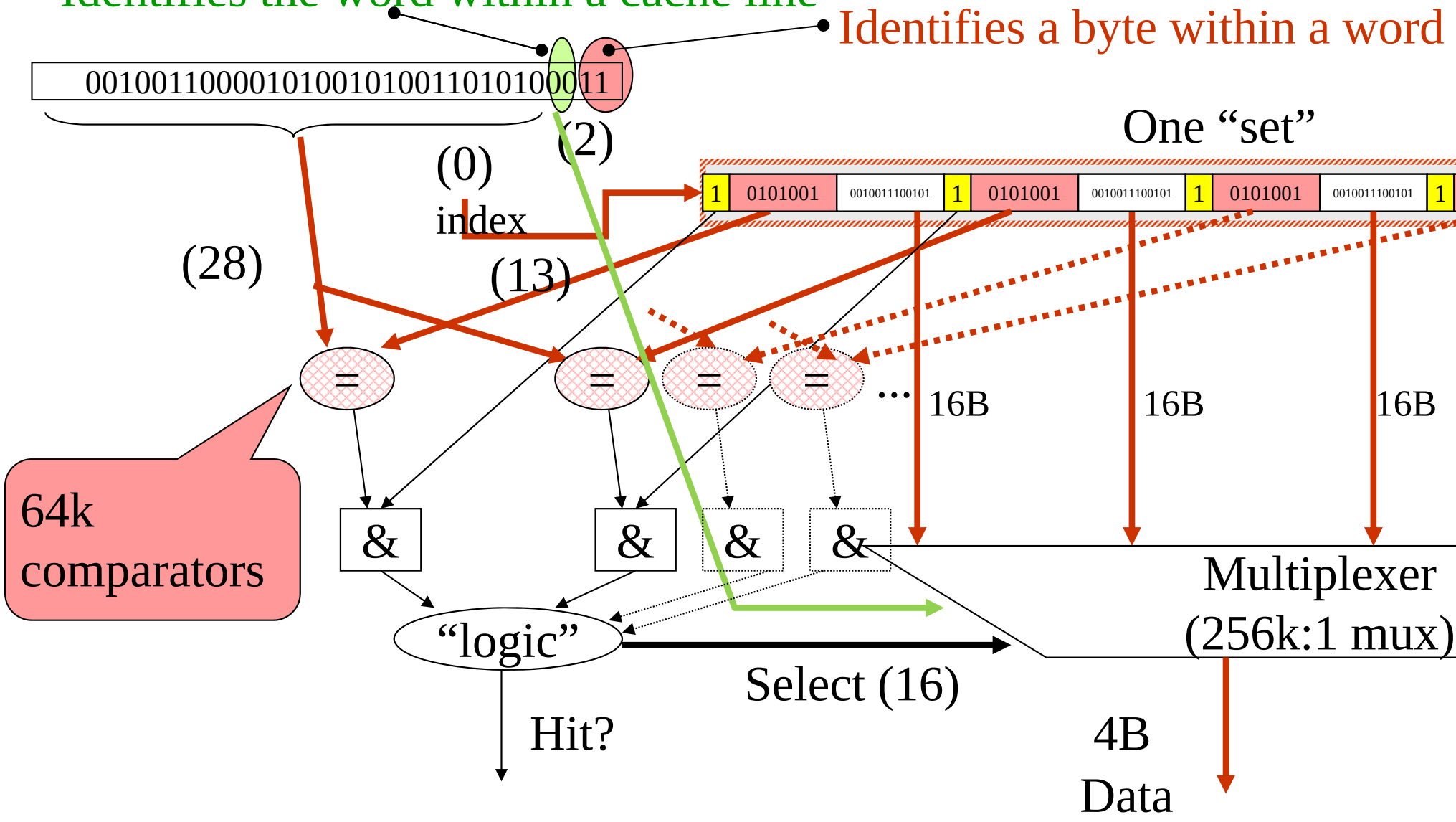
-- More complex implementation
comparators, muxes, ...

--- Higher dynamic power consumption

AVDARK 2013

Going all the way…!

1MB, fully associative, CL=16B



UPPSALA UNIVERSITET

Fully Associative

- Very expensive
- Only used for small caches (and sometimes TLBs)

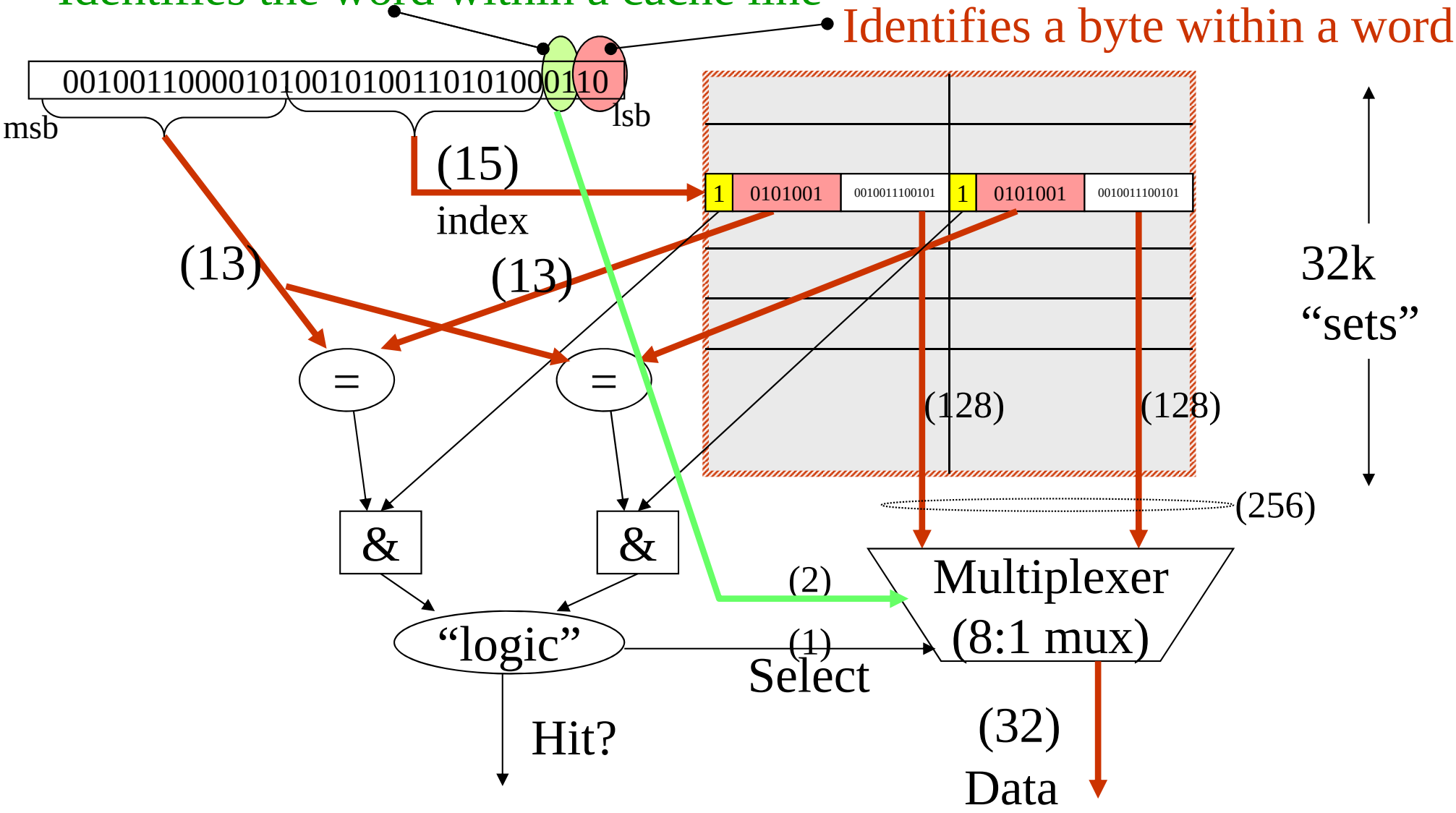
CAM = Contents-addressable memory

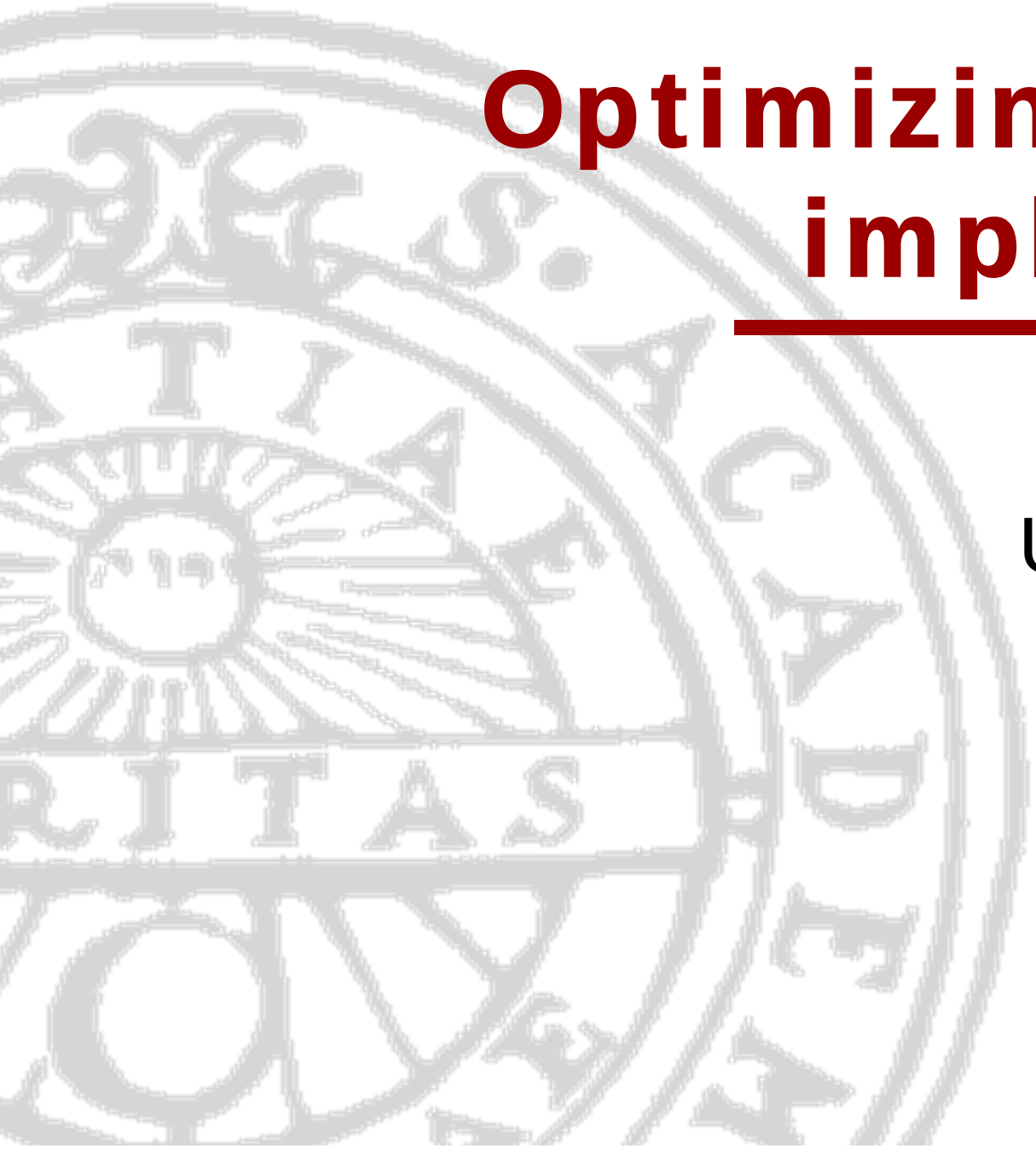
- ~Fully-associative cache storing key+data
- Provide key to CAM and get the associated data

AVDARK 2013

A combination thereof

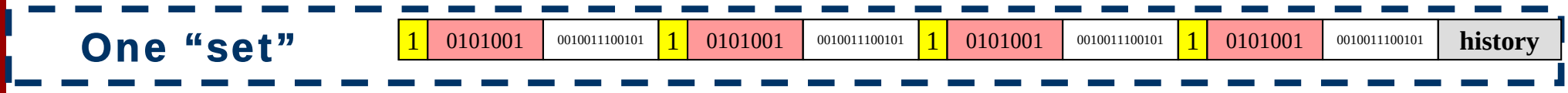
1MB, 2-way, CL=16B, word=4B





Optimizing the cache implementation

Erik Hagersten
Uppsala University



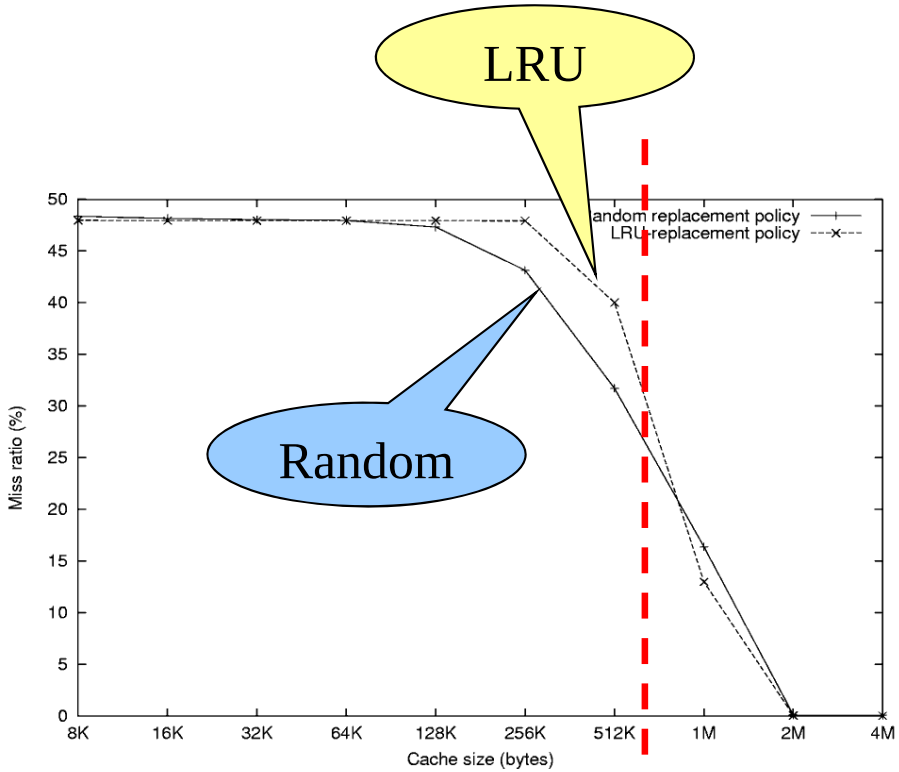
Who to replace?

Picking a “victim”

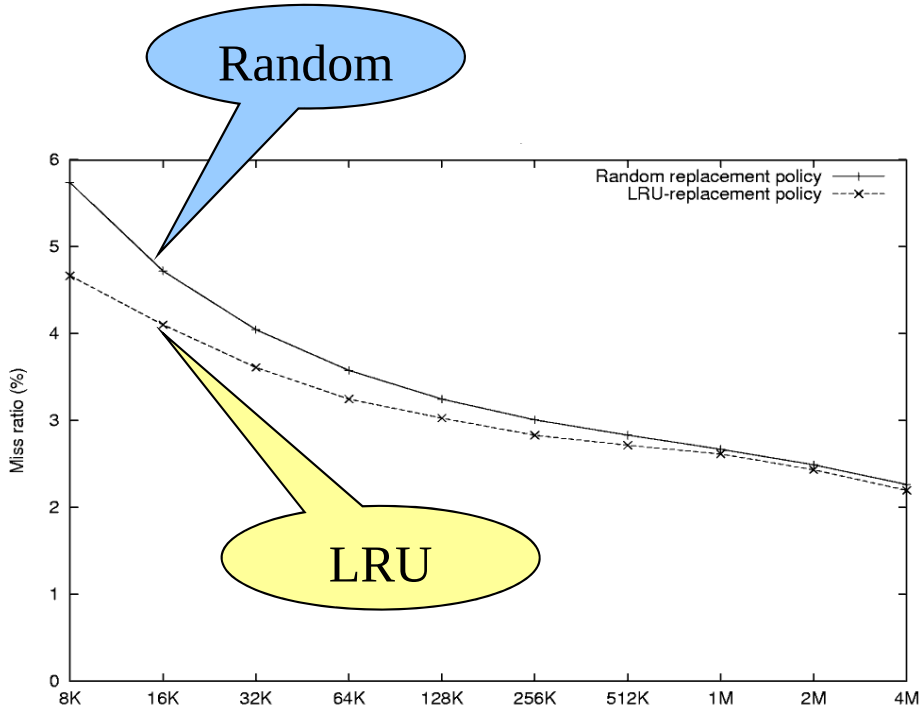
- Least-recently used (LRU) replacement
  - Throw out the longest unused cache line
  - Considered the “best” algorithm (which is not always true…)
  - Only practical up to limited number of ways
- Not most recently used
  - Remember who used it last: 8-way -> 3 bits/CL
- Pseudo-LRU
  - E.g., based on course time stamps.
  - Used in the VM system
- Random replacement
  - Can’t continuously to have “bad luck…

UPPSALA UNIVERSITET

Cache Model: Random vs. LRU



art (SPEC 2000)



equake (SPEC 2000)

Data set size

AVDARK 2013

UPPSALA UNIVERSITET

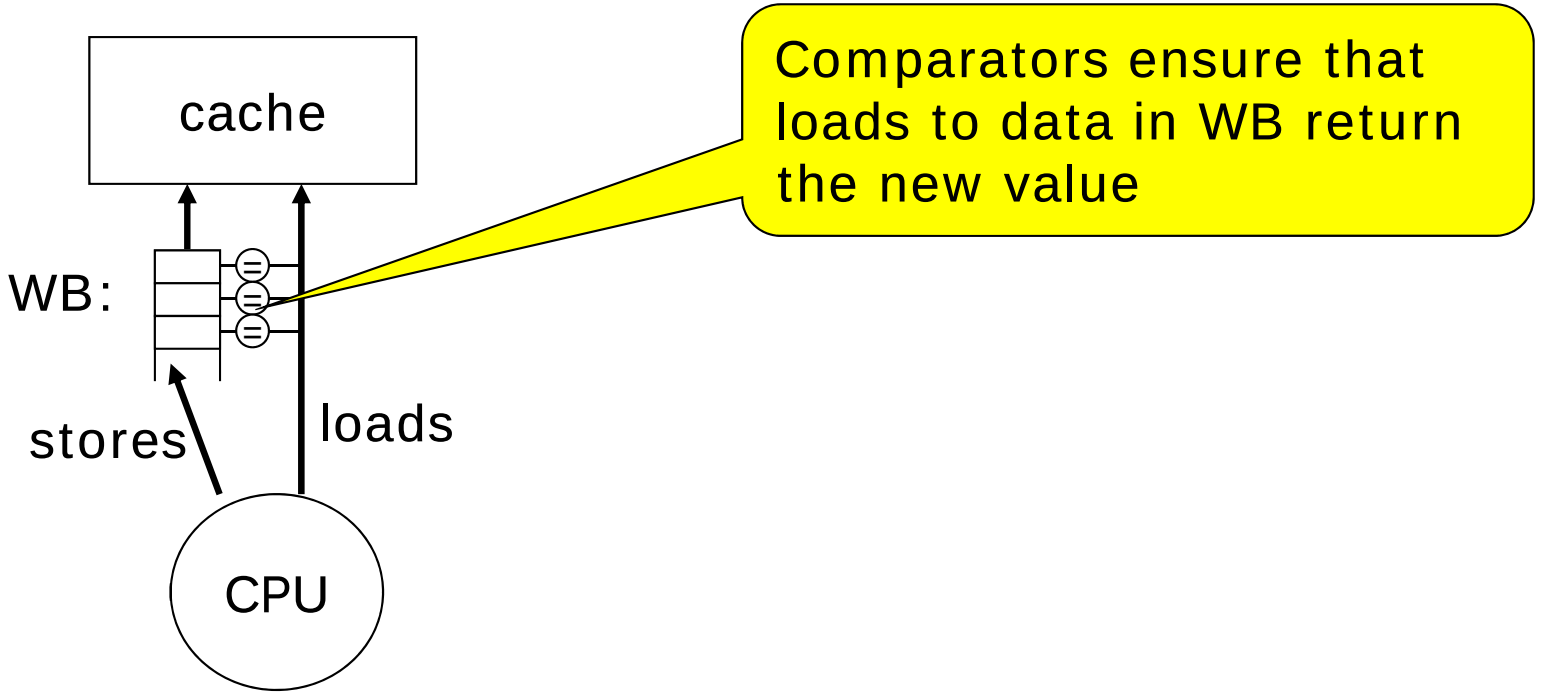
Handling dirty cache lines

- Write-back caches
  - Implementation: A “dirty bit” per cache line indicates an altered cache line
  - Write dirty data back to memory (next cache level) at replacement, called write-backs
- Write-through
  - Always write through to memory (the the next level) as well

  ➔ data will never be dirty ➔ no write-backs

Write Buffer / Store Buffer

- Reads are more important than writes
- Do not need the old value of the word to perform a store!



UPPSALA UNIVERSITET

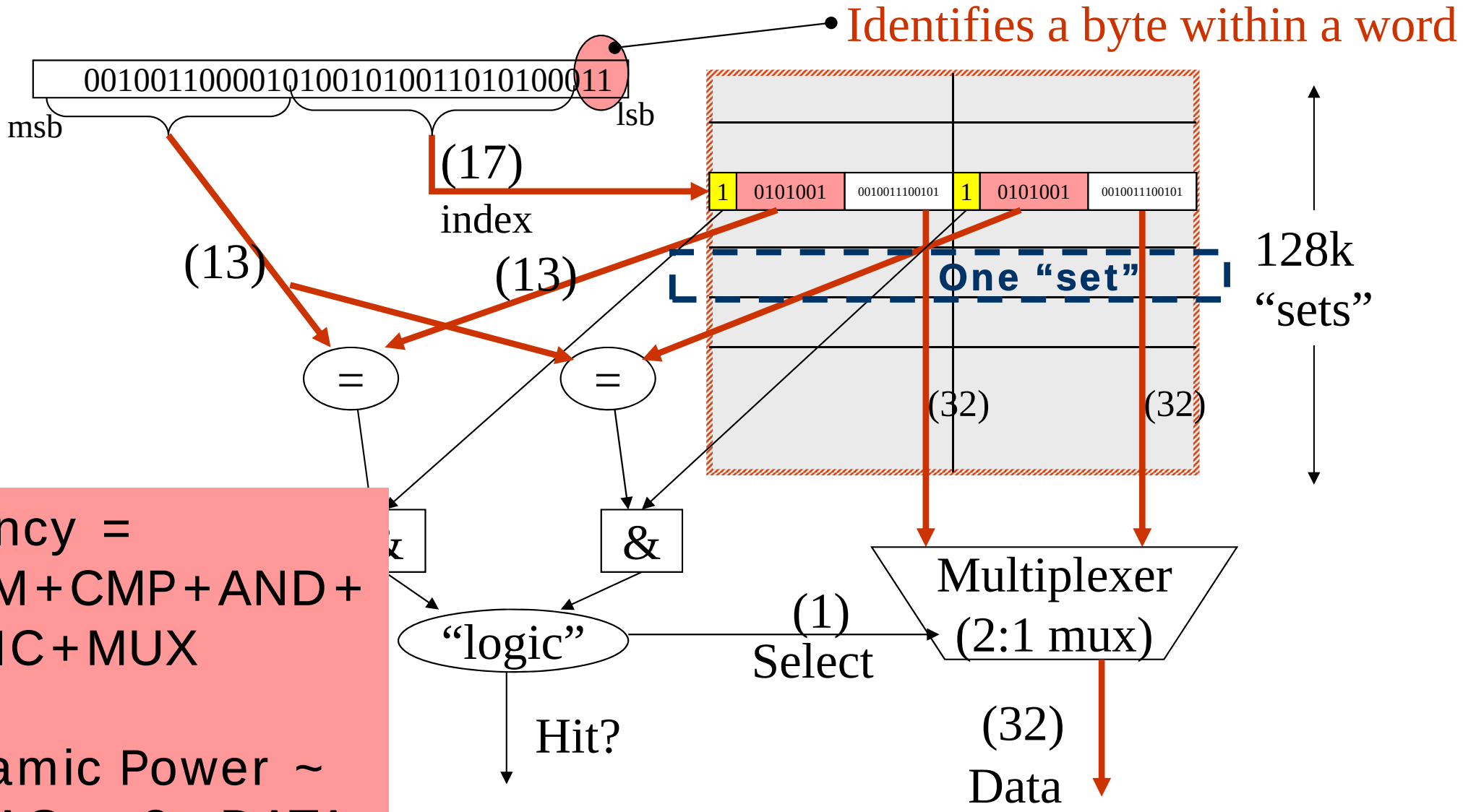
Power consumption in caches

- Static power or leakage power
  - Current leaks through transistors
  - Proportional to the number of transistors used
  - ➔ proportional to the cache capacity
- Dynamic power
  - Extra energy needed to read SRAMs bits
  - ➔ proportional to the number of SRAM bits read
- Which power dominates?
  - Associative and fast (L1) cache: Dynamic
  - Large slower cache (LLC): Static

AVDARK 2013

Fast cache access (L1), → parallel implementation

1MB, 2-way set-associative, CL=4B



Latency = SRAM+CMP+AND+ LOGIC+MUX

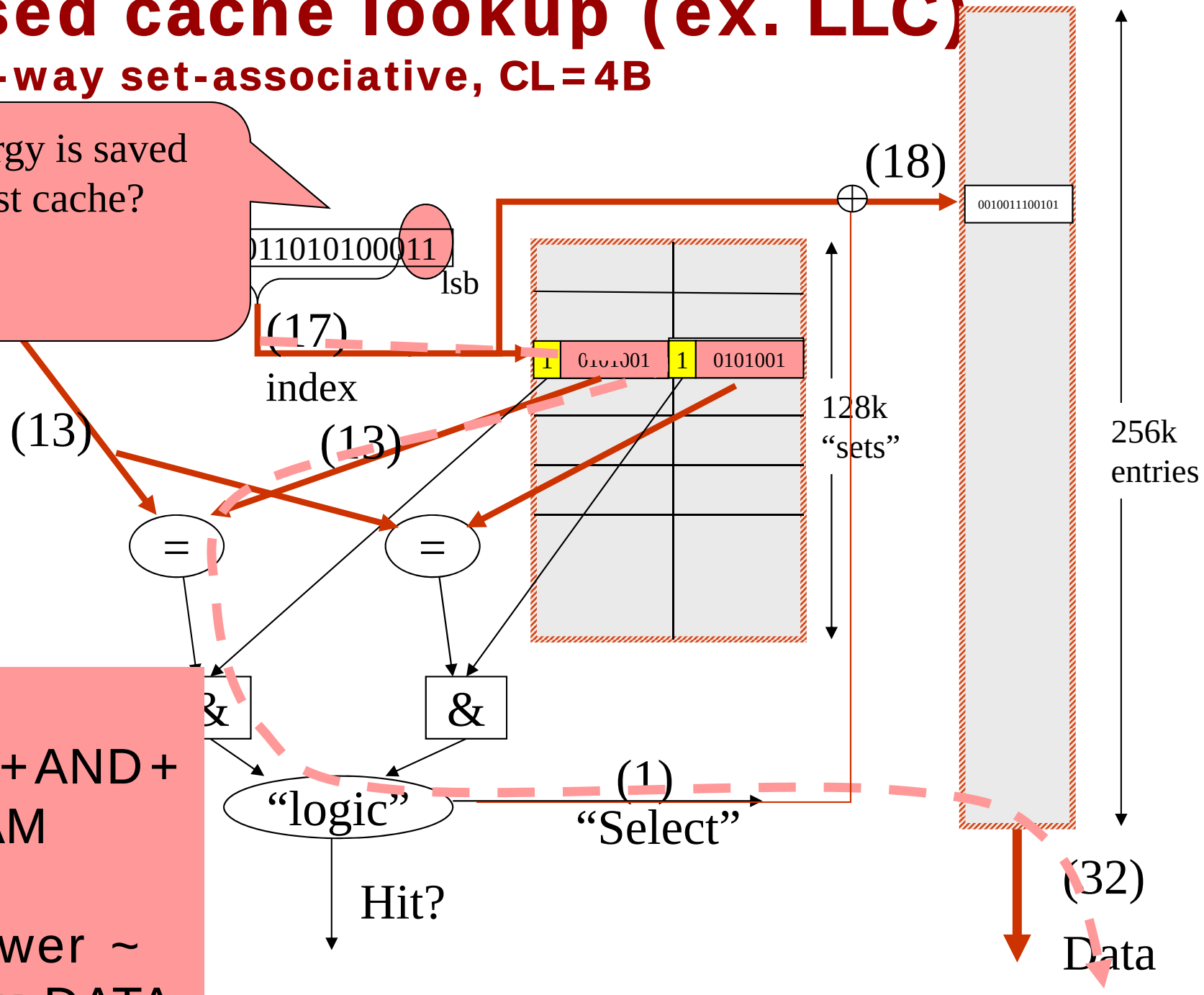
Dynamic Power ~ 2x TAG + 2x DATA

Phased cache lookup (ex. LLC)

1MB, 2-way set-associative, CL=4B

What kind of energy is saved compared with fast cache?

- ❑ Static
- ❑ Dynamic



Latency = SRAM+CMP+AND+ LOGIC+SRAM

Dynamic Power ~ 2x TAG + **1x** DATA

UPPSALA UNIVERSITET

Topology of caches: Harvard Arch

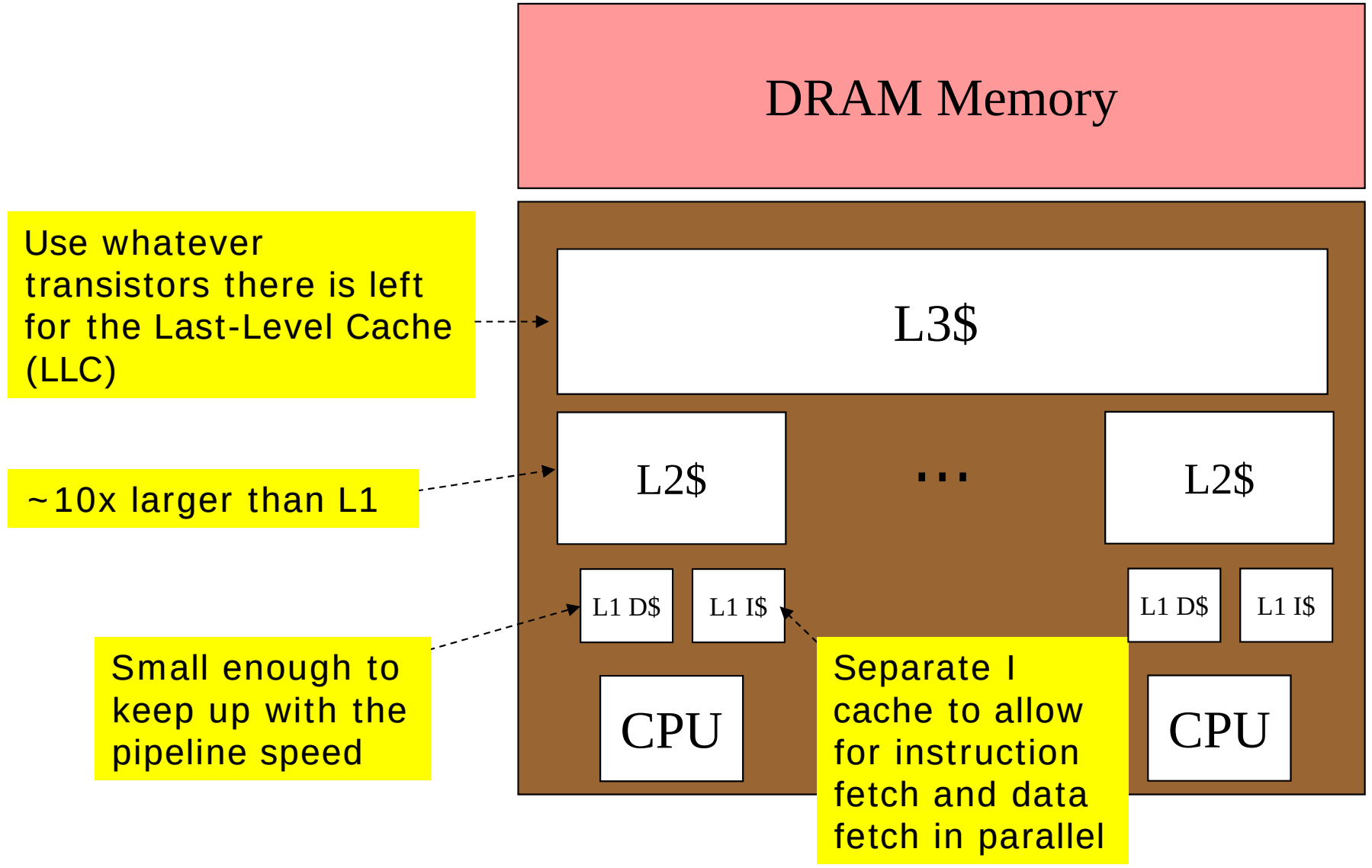
- CPU needs a new instruction each cycle
- 25% of instruction LD/ST
- Data and Instr. have different access patterns

==> Separate D and I first level cache

==> Unified 2nd and 3rd level caches

AVDARK 2013

Cache Hierarchy of Today



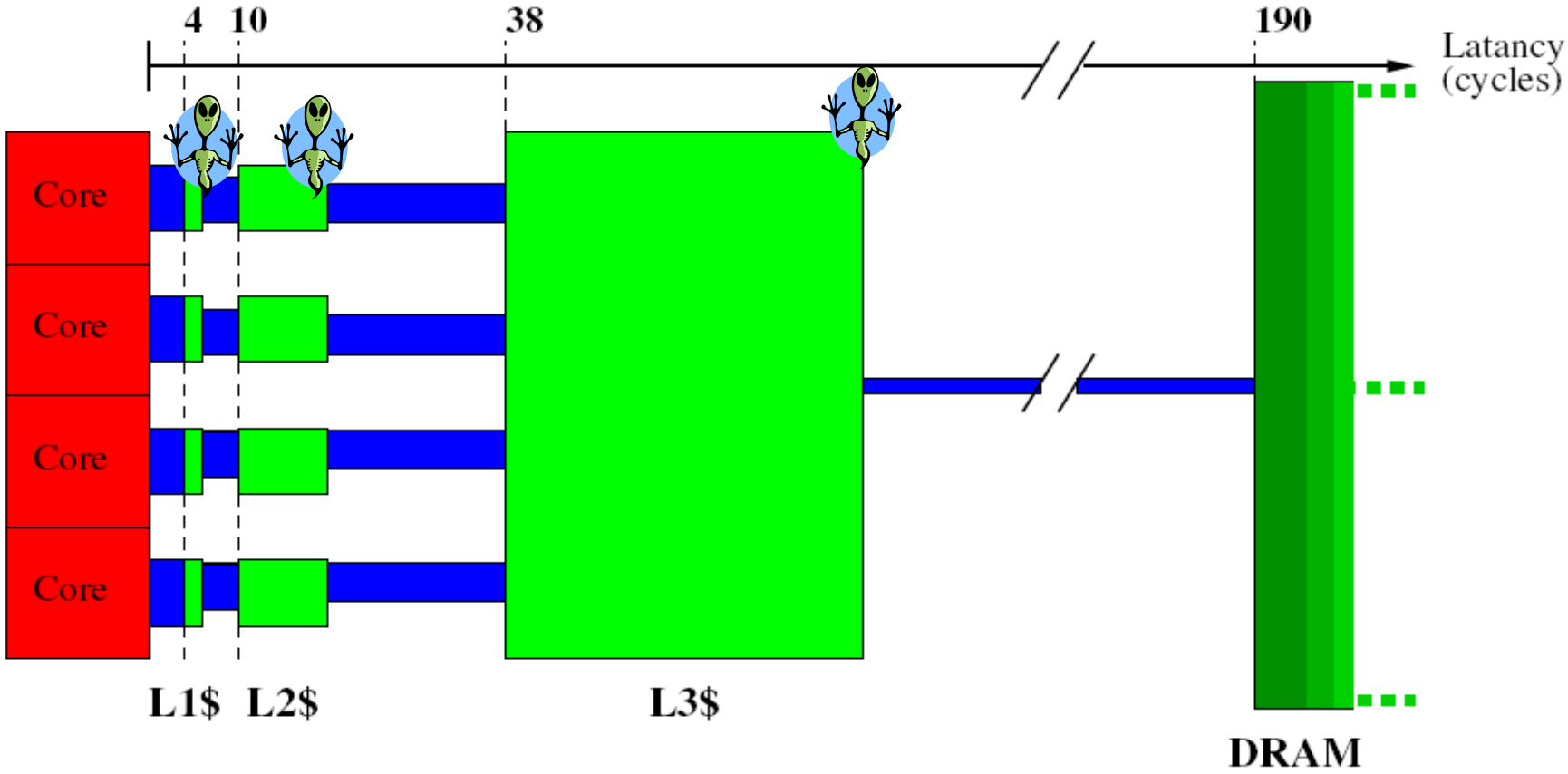
HW prefetching

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

Improves MLP (memory-level parallelism)

- 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. Often for L1 caches.
- Adjacent prefetching: On a miss, also bring in the "neighbouring" cache line. Often only for L2 and L3.

Data Cache Capacity / Latency / BW



Cache implementation

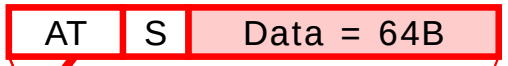
L3 € 8MB

L2 $ 256kB

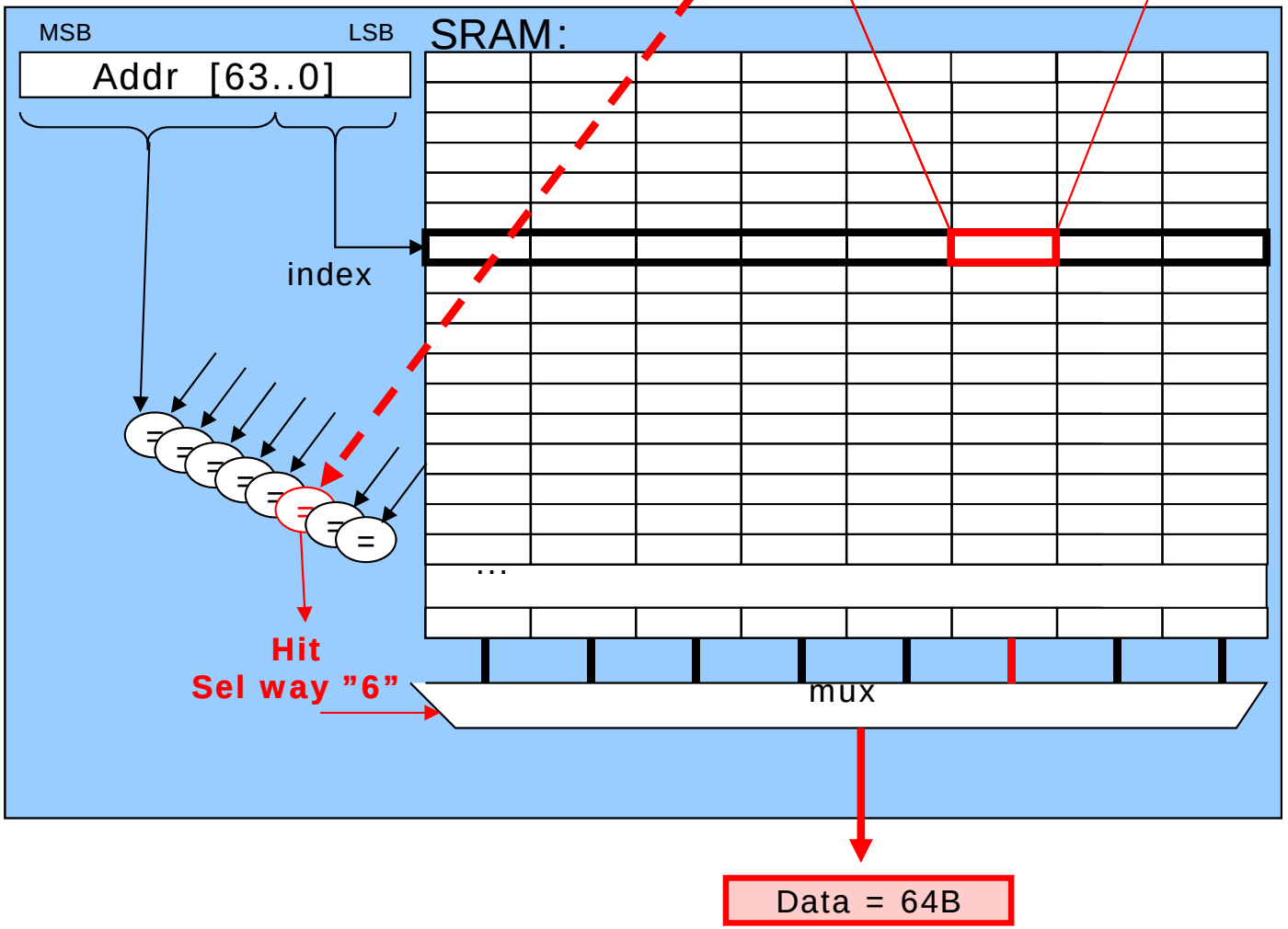
D1 ¢ 64kB

I1 ¢ 64kB

Cacheline, here 64B:



Generic Cache:



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)

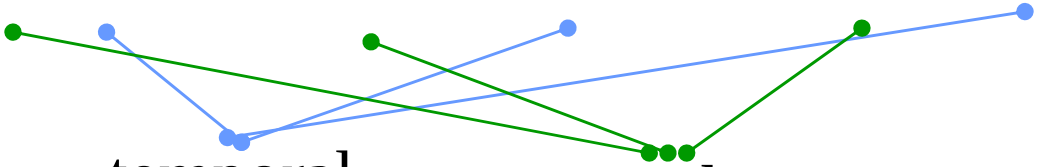
Temporal locality: Likelihood to access the same data again soon

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, ...



UPPSALA UNIVERSITET

Inclusive/Exclusive

- Cache hit in higher level caches. Three options:

Inclusive (Intel)

- Install a copy of the data in L1 and keep the data in L2
- When the L2 data is replaced, force eviction from L1
- If the data in not in L2, it is not in L1 either

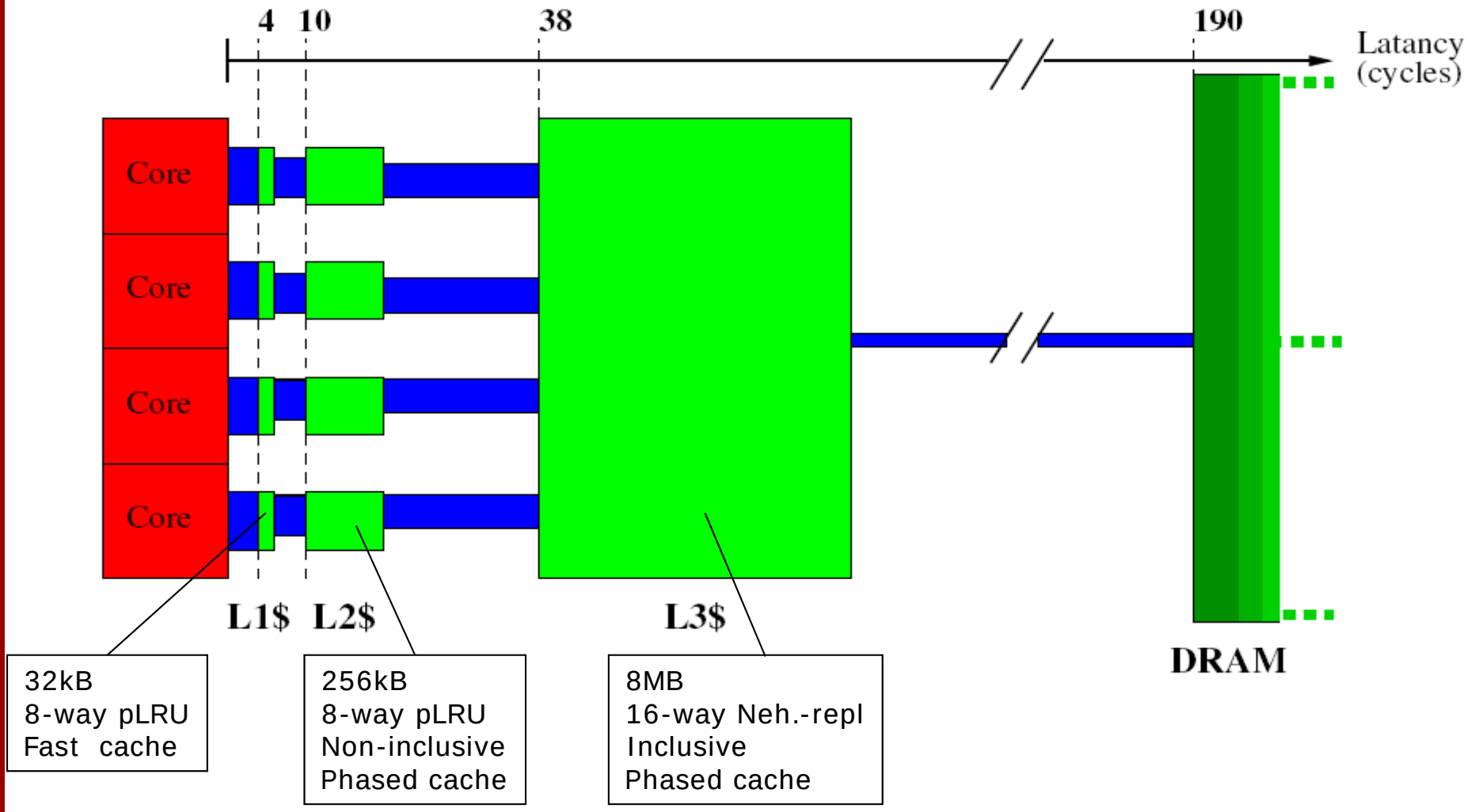
Non-inclusive (~AMD)

- Install a copy of the data in L1 and keep the data in L2
- When the L2 data is replaced, L1 data survives
- If the data in not in L2, it may be in L1

Exclusive

- Move the data from L2 to L1
- At L1 replacement, move the data back to L2
- Data is either in L1 or L2, but never in both

Nehalem i7 (one example)



UPPSALA UNIVERSITET

Take-away message: Caches

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

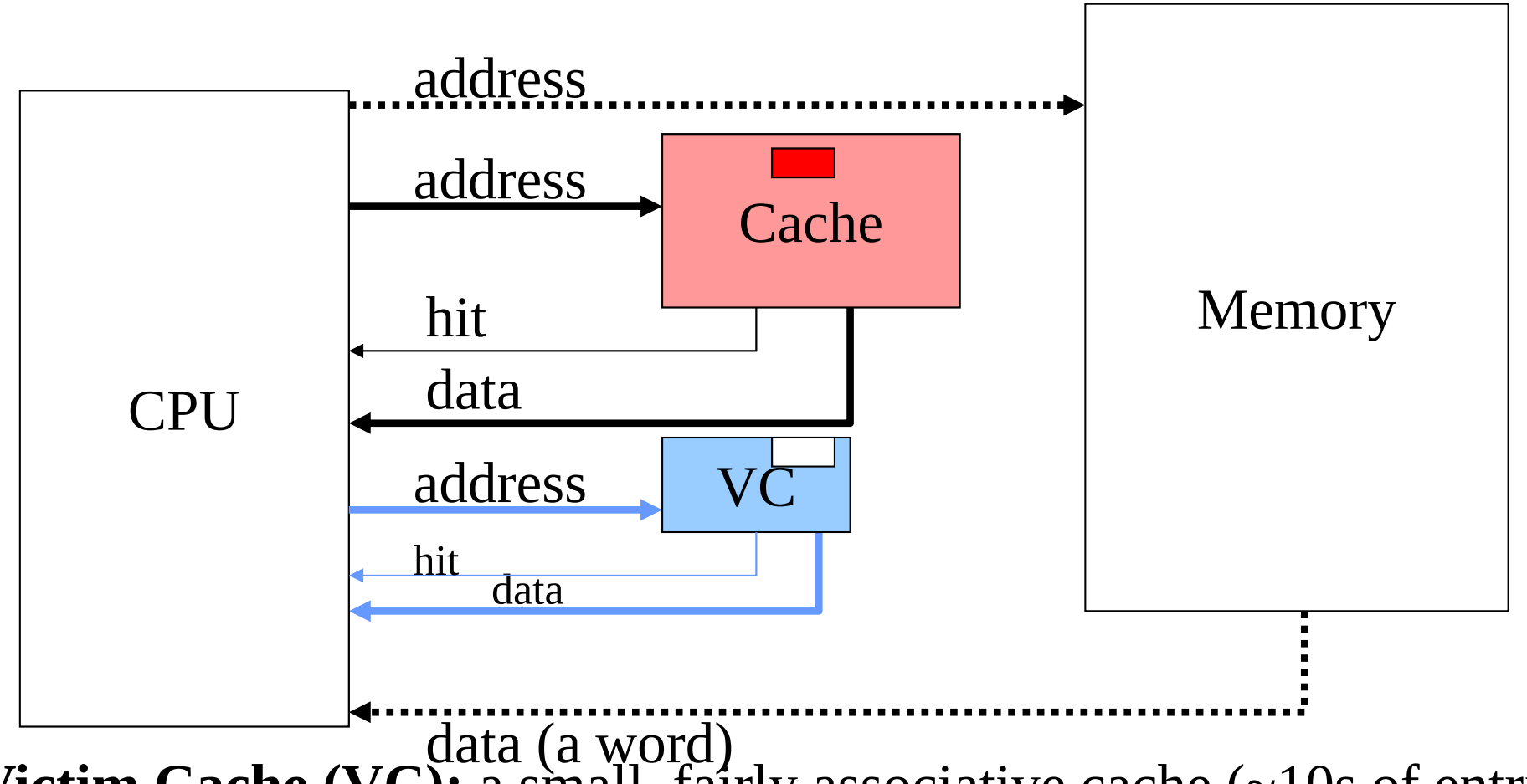
- The most important target for optimizations

AVDARK 2013

Innovative caches

Erik Hagersten
Uppsala University

Innovative cache: Victim cache



Victim Cache (VC): a small, fairly associative cache (~10s of entries)

Cache lookup: search cache and VC in parallel

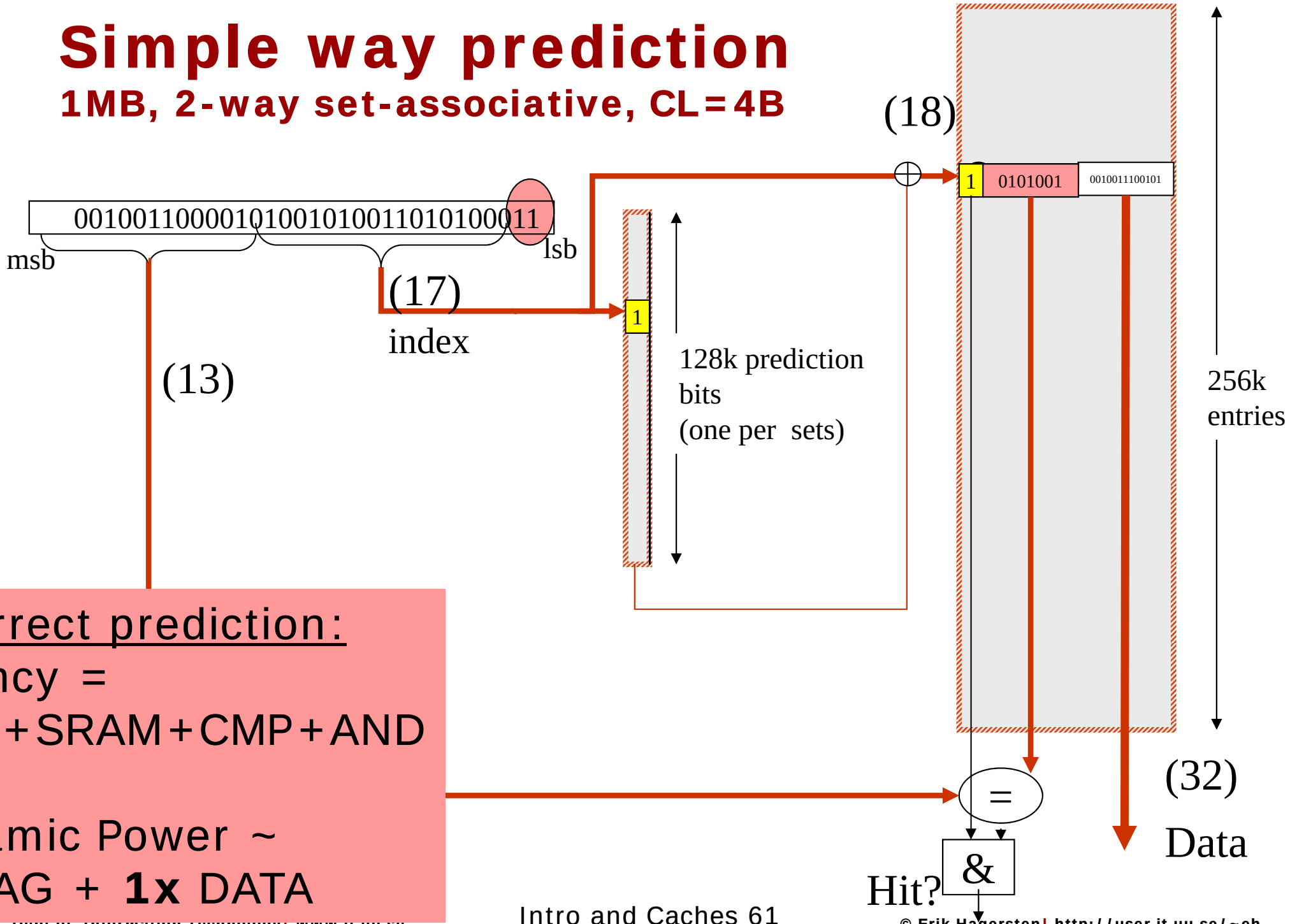
Cache replacement: move victim to the VC and replace from VC

VC hit: swap VC data with the corresponding data in Cache

"A second life ☺"

Simple way prediction

1MB, 2-way set-associative, CL=4B



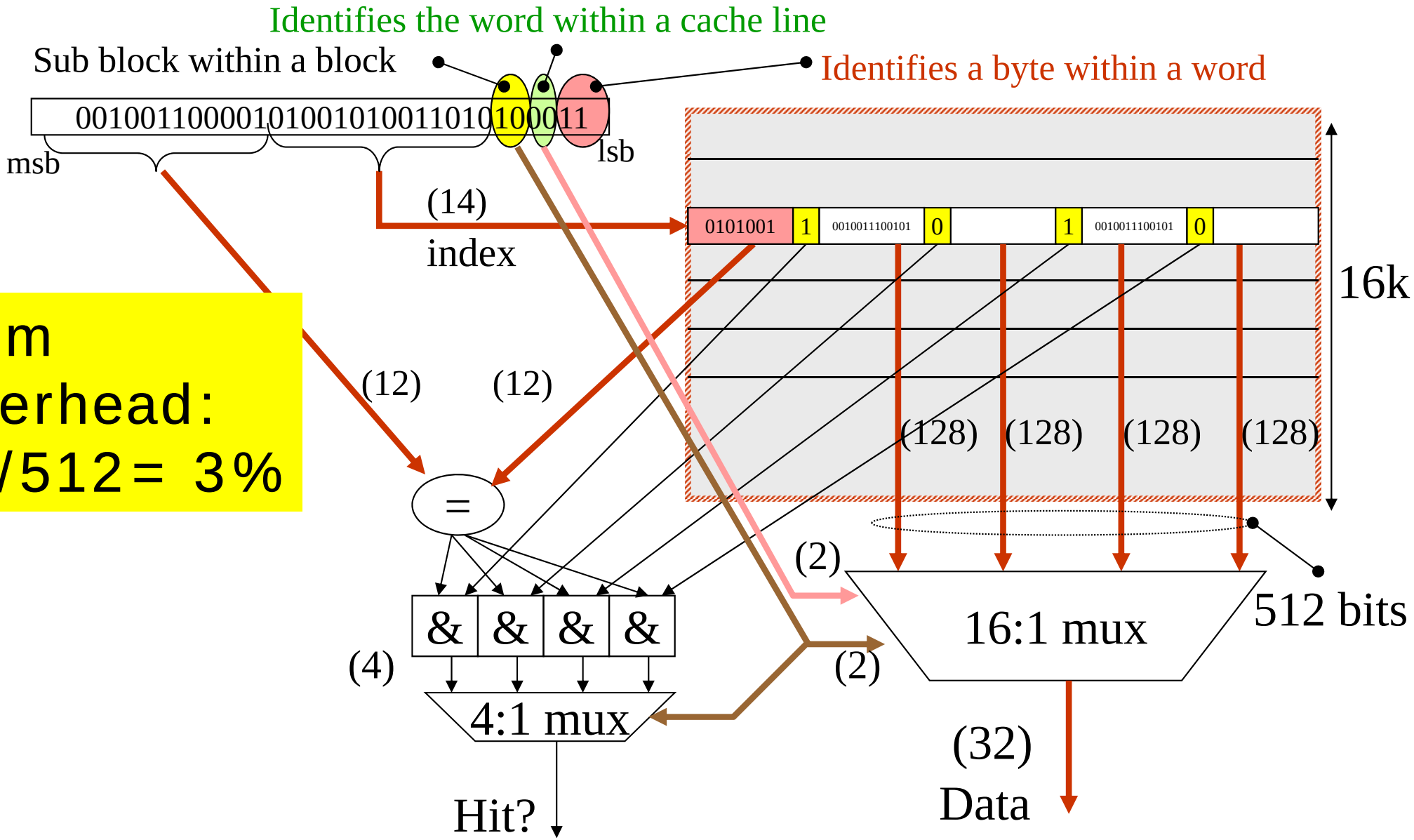
If correct prediction:

Latency = sram + SRAM + CMP + AND

Dynamic Power ~ **1x** TAG + **1x** DATA

4-way sub-blocked cache

1MB, direct mapped, Block=64B, sub-block=16B



Mem Overhead: 16/512= 3%

UPPSALA UNIVERSITET

Pros/Cons Sub-blocking

++ Lowers the memory overhead

++ (Avoids problems with false sharing -- MP)

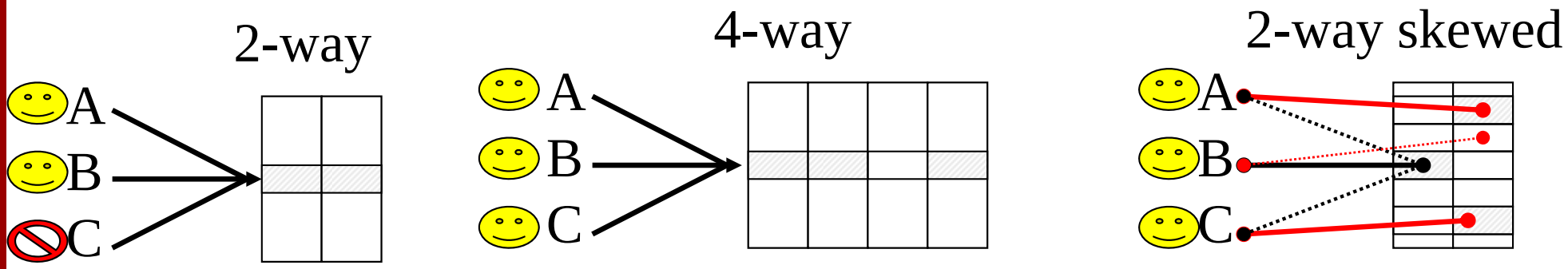
-- Will not explore as much spatial locality

-- Still poor utilization of SRAM (fewer sparse "things" allocated)

AVDARK 2013

Skewed Associative Cache

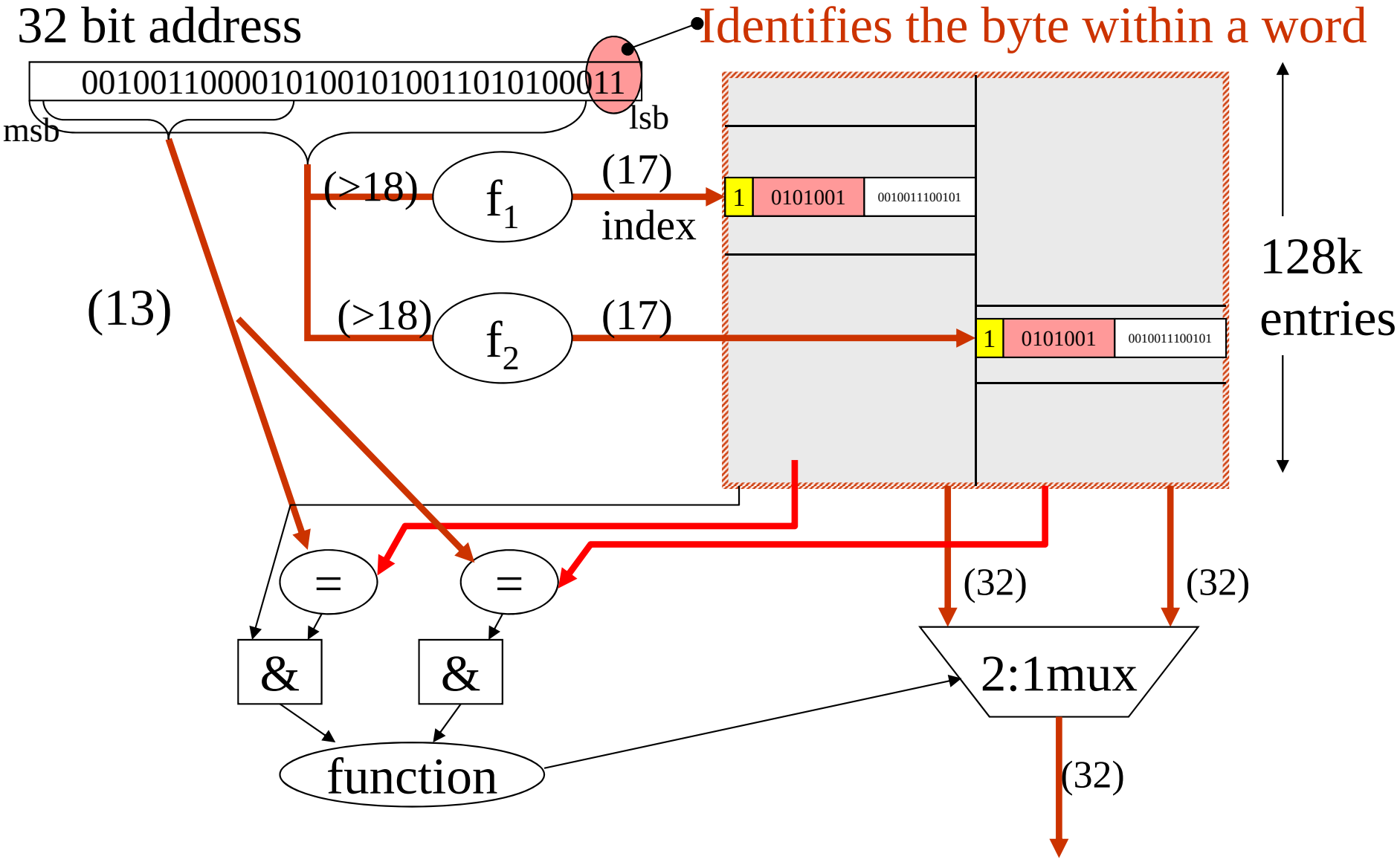
Example: A, B and C have a three-way conflict



- It has been shown that 2-way skewed performs roughly the same as 4-way caches

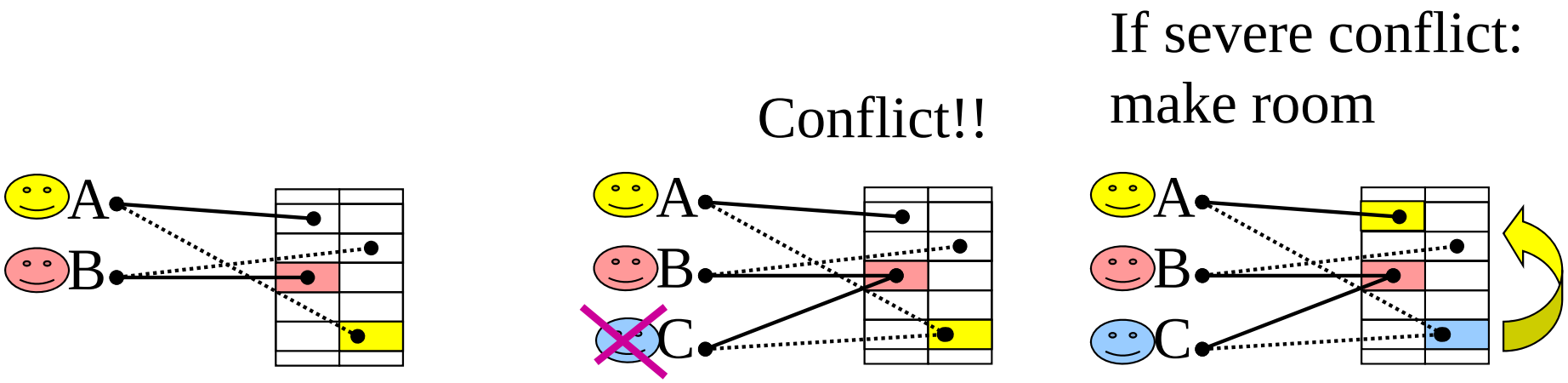
- Uses less power than 4-way

Skewed-associative cache:
Different indexing functions



UART: Elbow cache (Zcache)

Increase "associativity" when needed



Performs roughly the same as an 8-way cache

Slightly faster

Uses much less dynamic power!!

UPPSALA UNIVERSITET

How are we doing?

- Increase clock frequency
- 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)
- Speculative execution
  a) Out-of-order execution
  b) Branch prediction
  c) Prefetching

UPPSALA UNIVERSITET

Goals for this course

- Understand **how and why** modern computer systems are designed the way the are:
  - pipelines
  - memory organization
  - virtual/physical memory …
- Understand **how and why** multiprocessors are built
  - Cache coherence
  - Memory models
  - Synchronization…
- Understand **how and why** parallelism is created and leveraged
  - Instruction-level parallelism
  - Memory-level parallelism
  - Thread-level parallelism…
- Understand **how and why** multiprocessors of combined SIMD/MIMD type are built
  - GPU
  - Vector processing…
- Understand **how** computer systems are adopted to different usage areas
  - General-purpose processors
  - Embedded/network processors…
- Understand the physical limitation of modern computers
  - Bandwidth
  - Energy
  - Cooling…

AVDARK 2013