

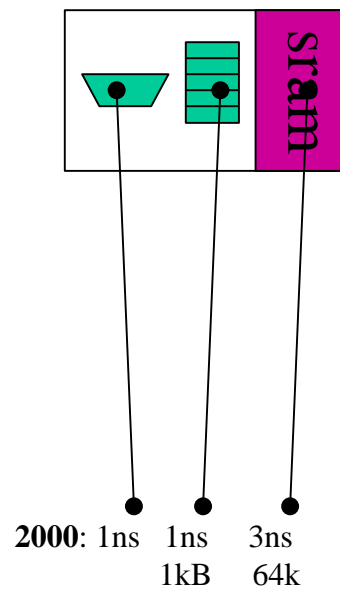
Caches and more caches
or
spam, spam, spam and spam

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

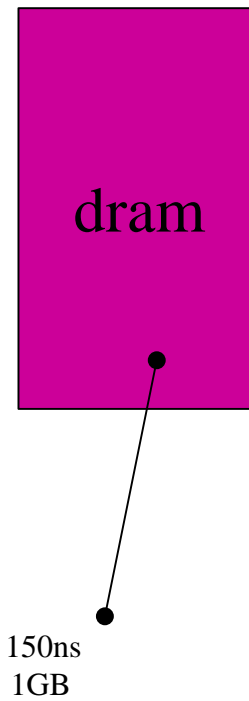
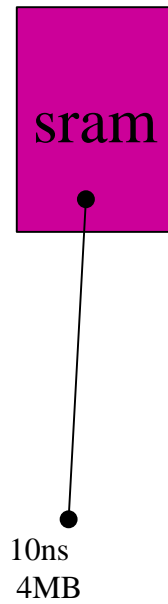
Cache knowledge useful when...

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

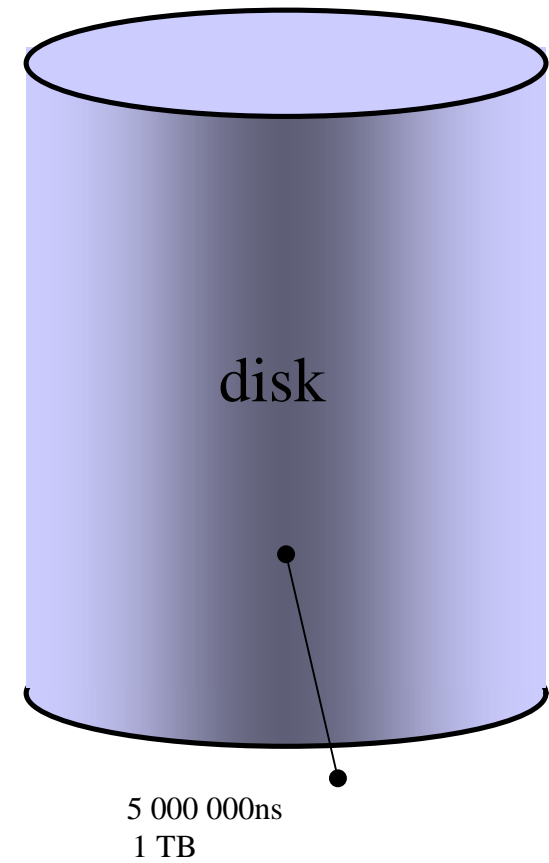
The development of memory systems



(1982: 200ns 200ns)



200ns



10 000 000ns)

Speed/Size/Cost tradeoff

| | Access time | #Entries | \$ per MB |
|----------|--------------------|-----------------|------------------|
| Disk | 10ms | 1000000000 | 0.1 |
| Memory | 150ns | 5 000 000 | 100 |
| Cache | 10ns | 50 000 | 10 |
| Register | 1ns | 64 | +++ |

Speed/Size/Cost tradeoff

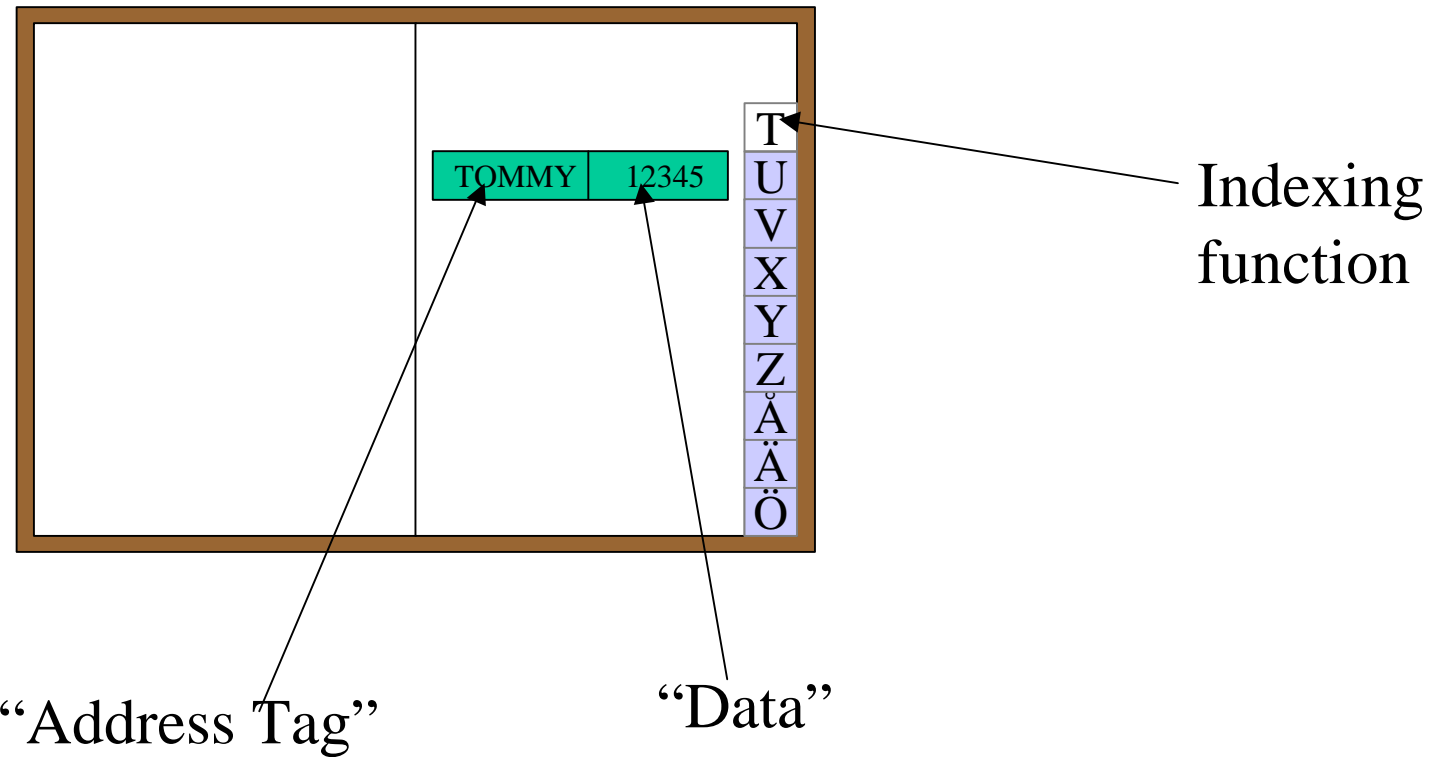
| | Access time | #Entries |
|---------------|------------------------|-----------------|
| Sweden | 300s | 5 000 000 |
| Uppsala | 60s | 100 000 |
| Family | 10s | 200 |
| Sweetheart(s) | 1s | 10 |

Webster about “cache”

1. cache \ˈkash\ 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

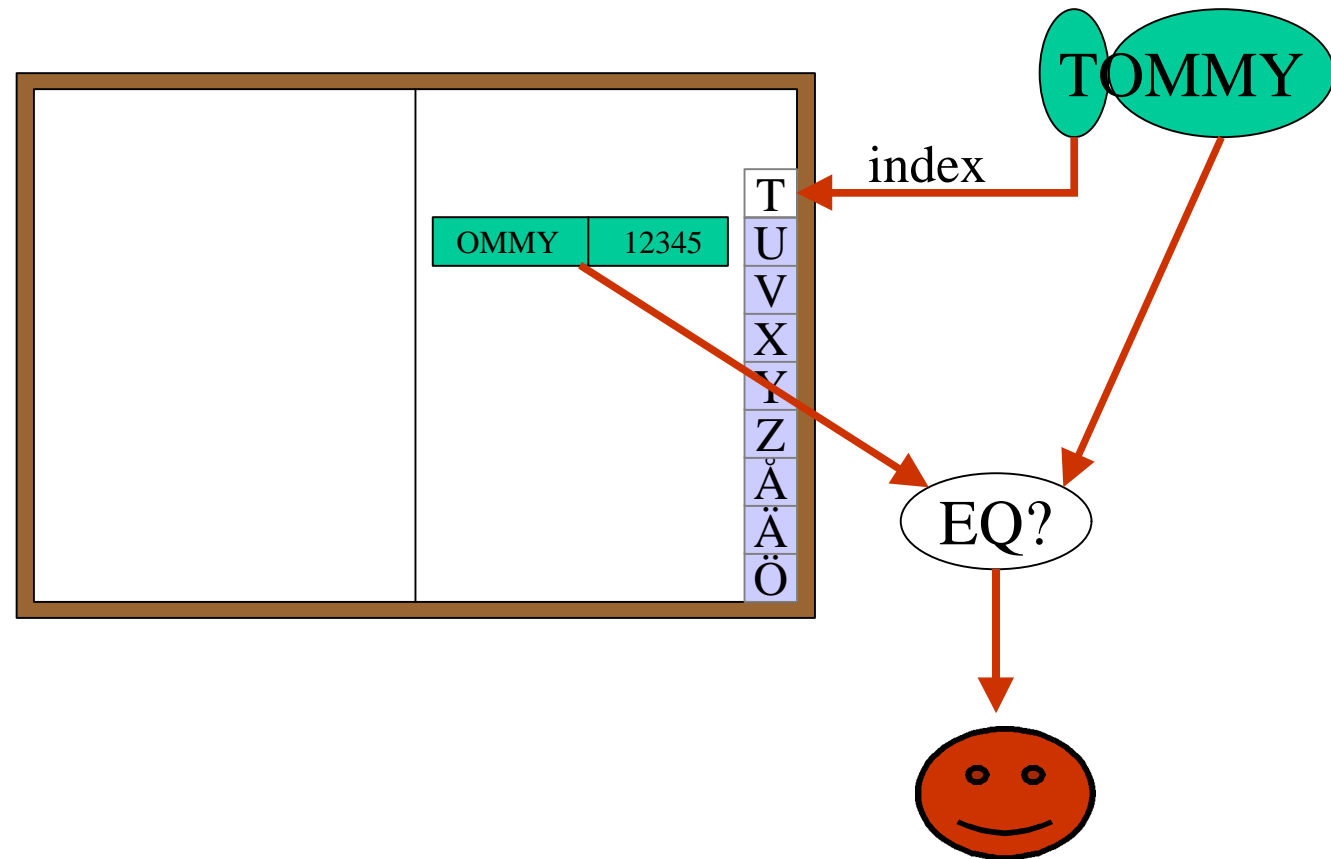


One entry per page =>

Direct-mapped caches with 28 entries

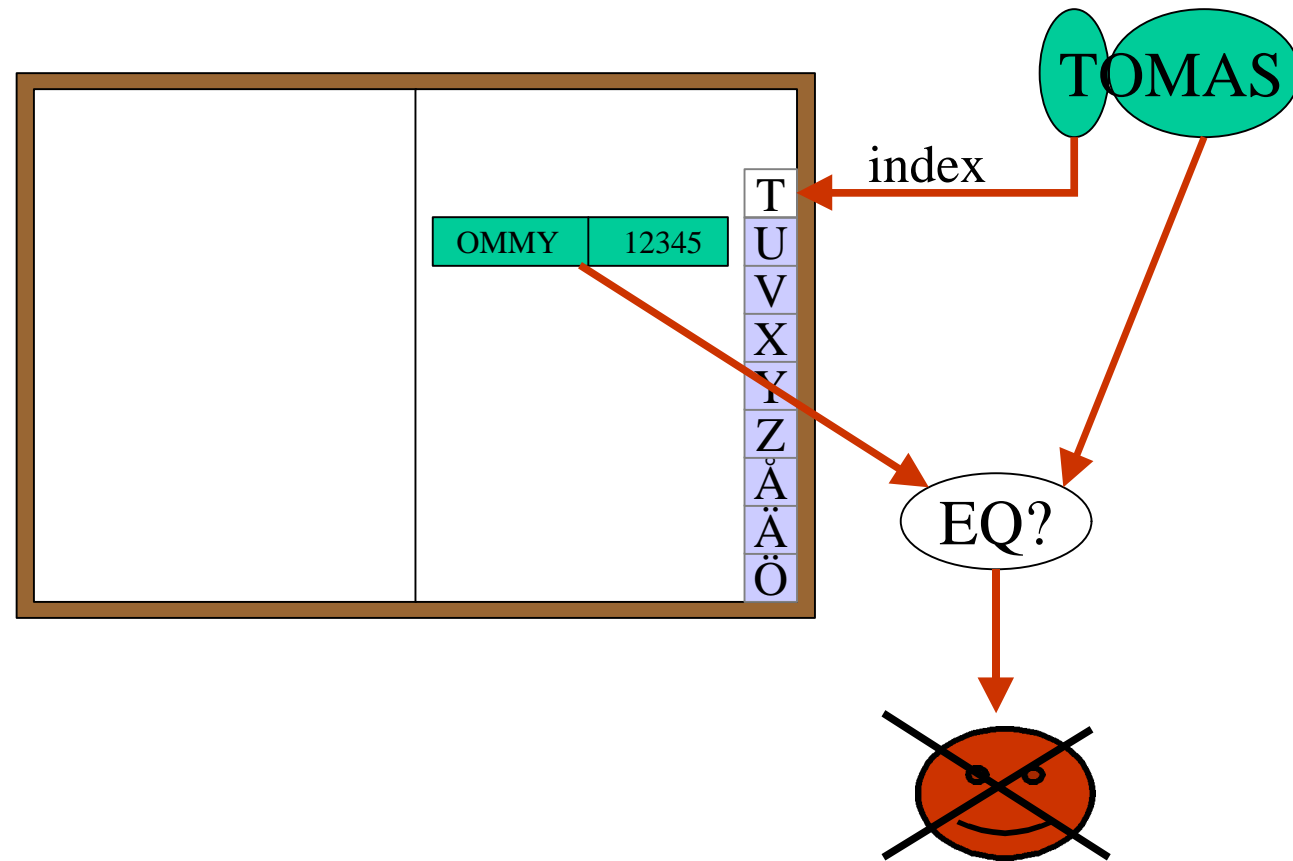
Address Book Cache

Looking for Tommy's Number



Address Book Cache

Looking for Tomas' Number

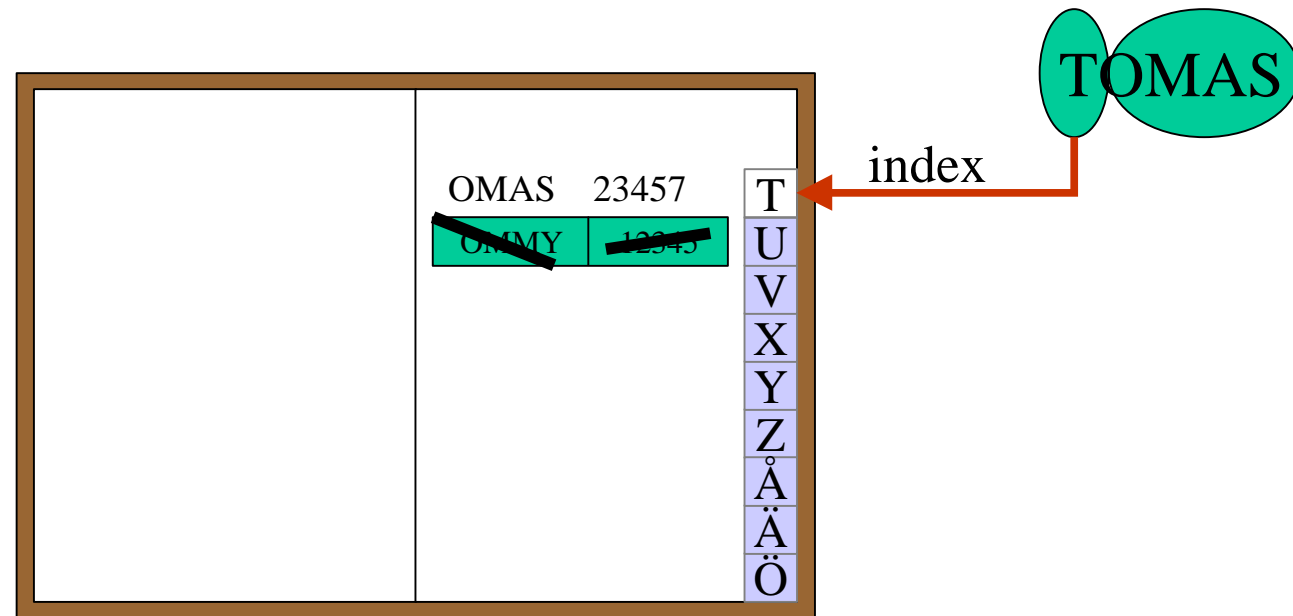


Miss!

Lookup Tomas' number in
the telephone directory

Address Book Cache

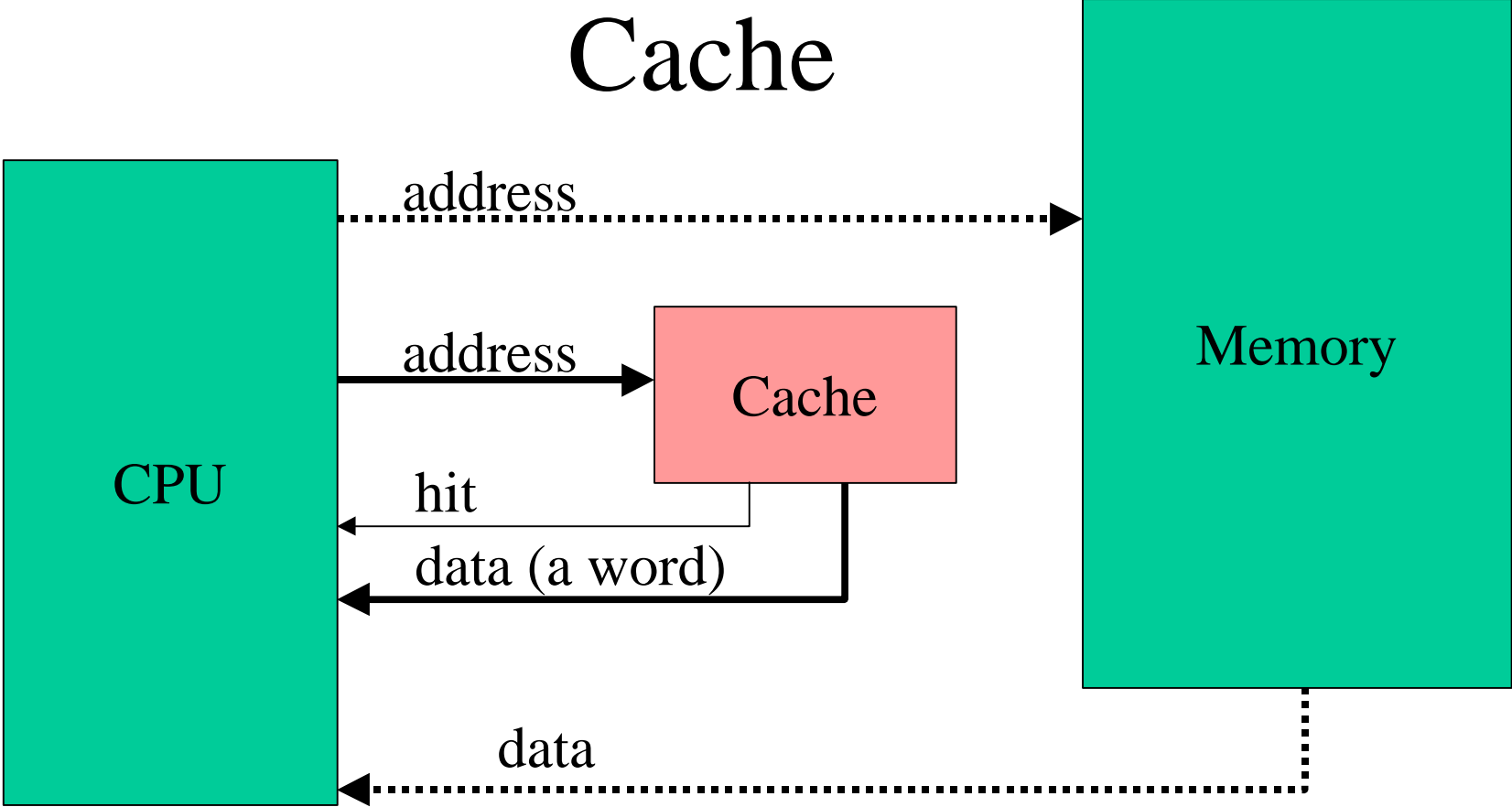
Looking for Tomas' Number



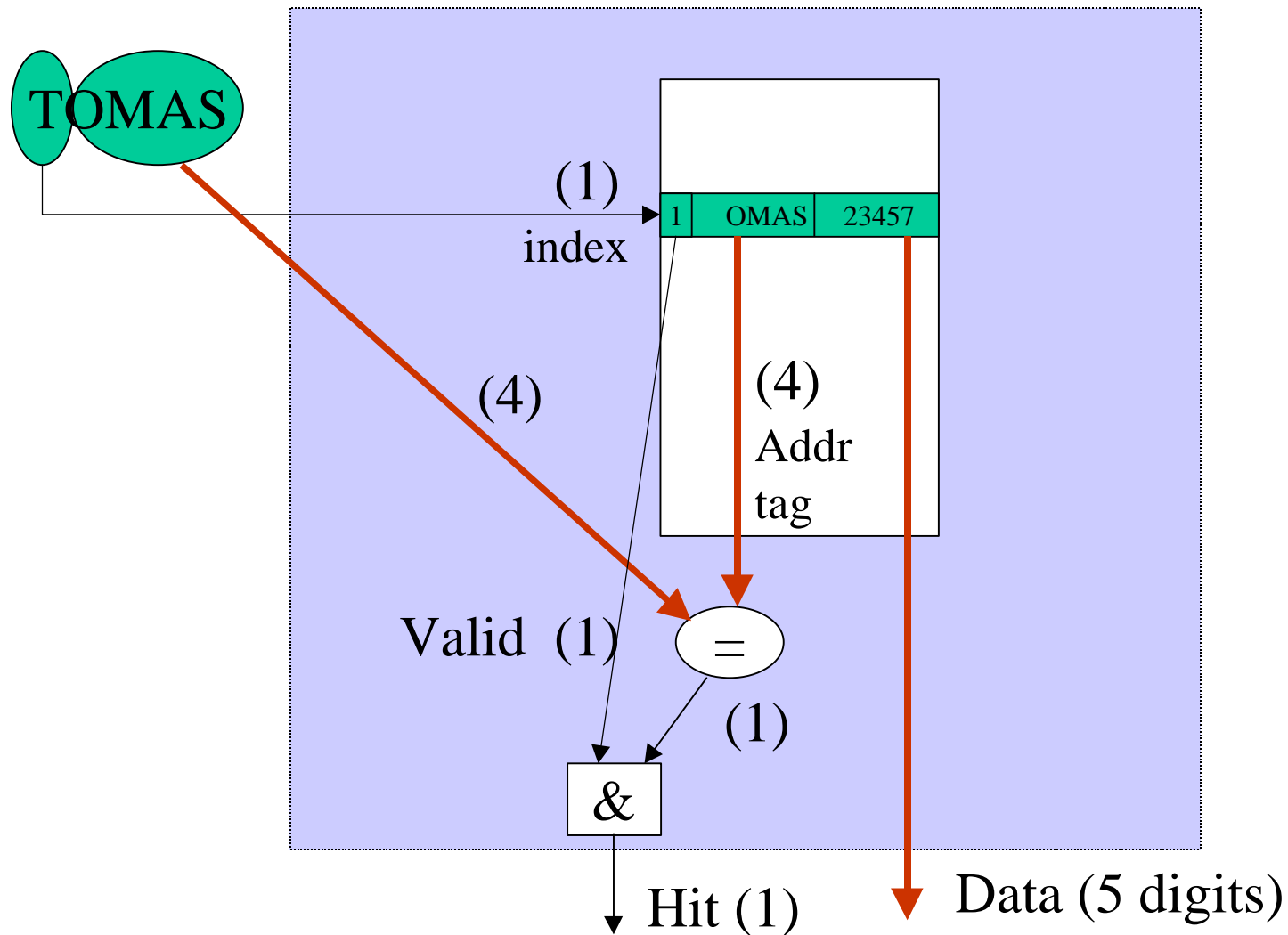
Replace TOMMY's data
with TOMAS' data.

There is no other choice
(direct mapped)

Cache



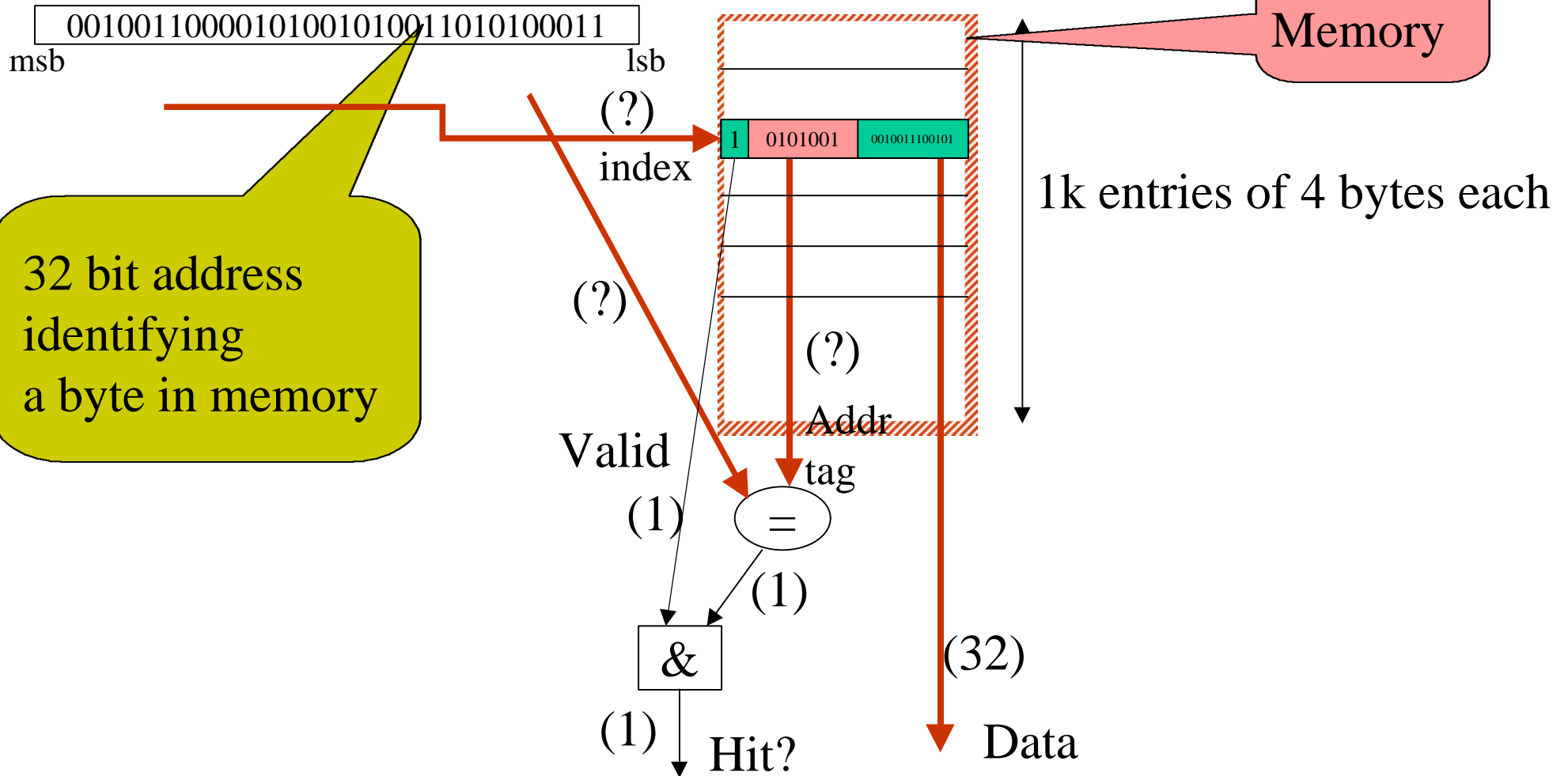
Cache Organization



Cache Organization (really)

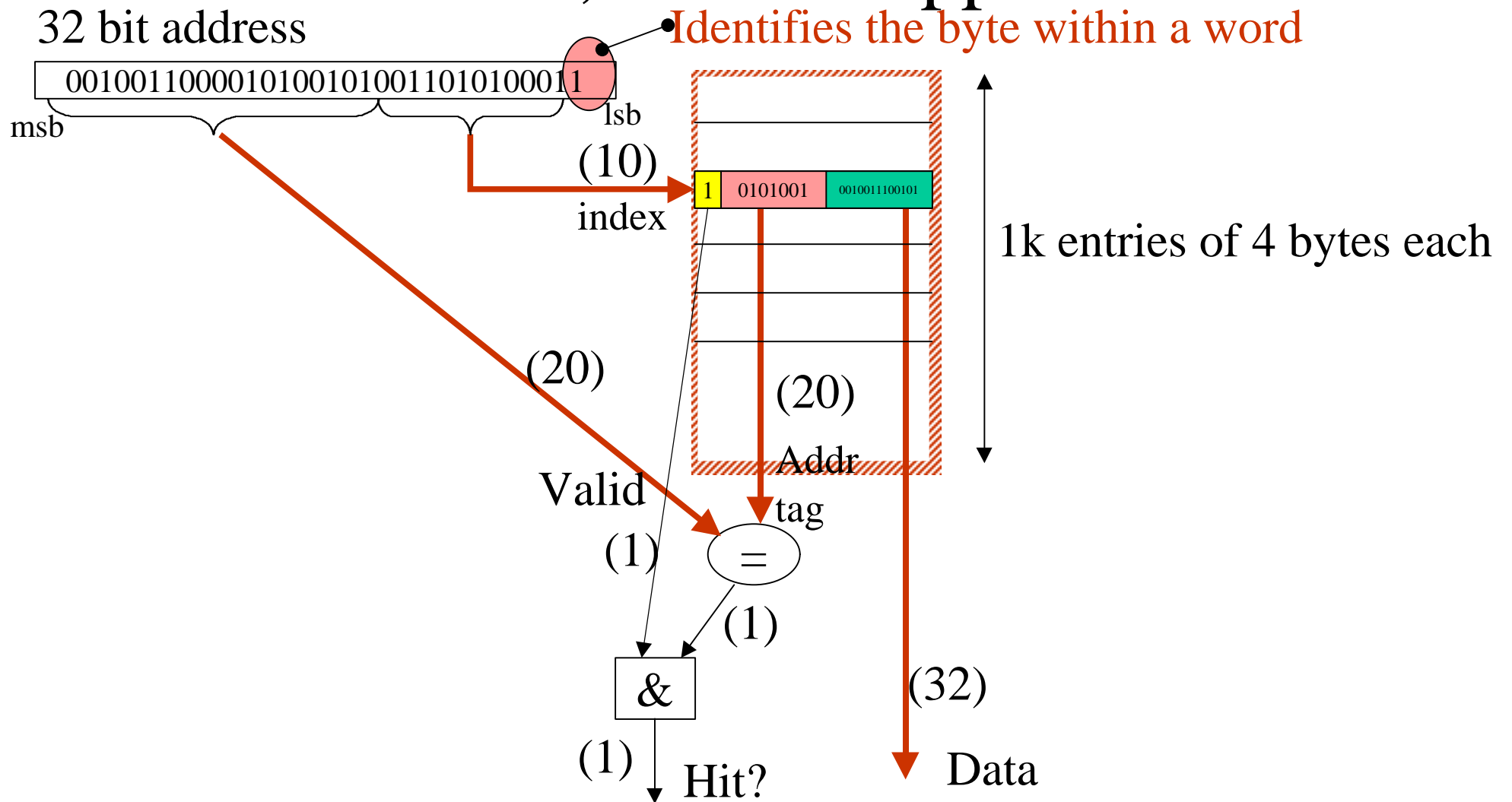
B="byte"

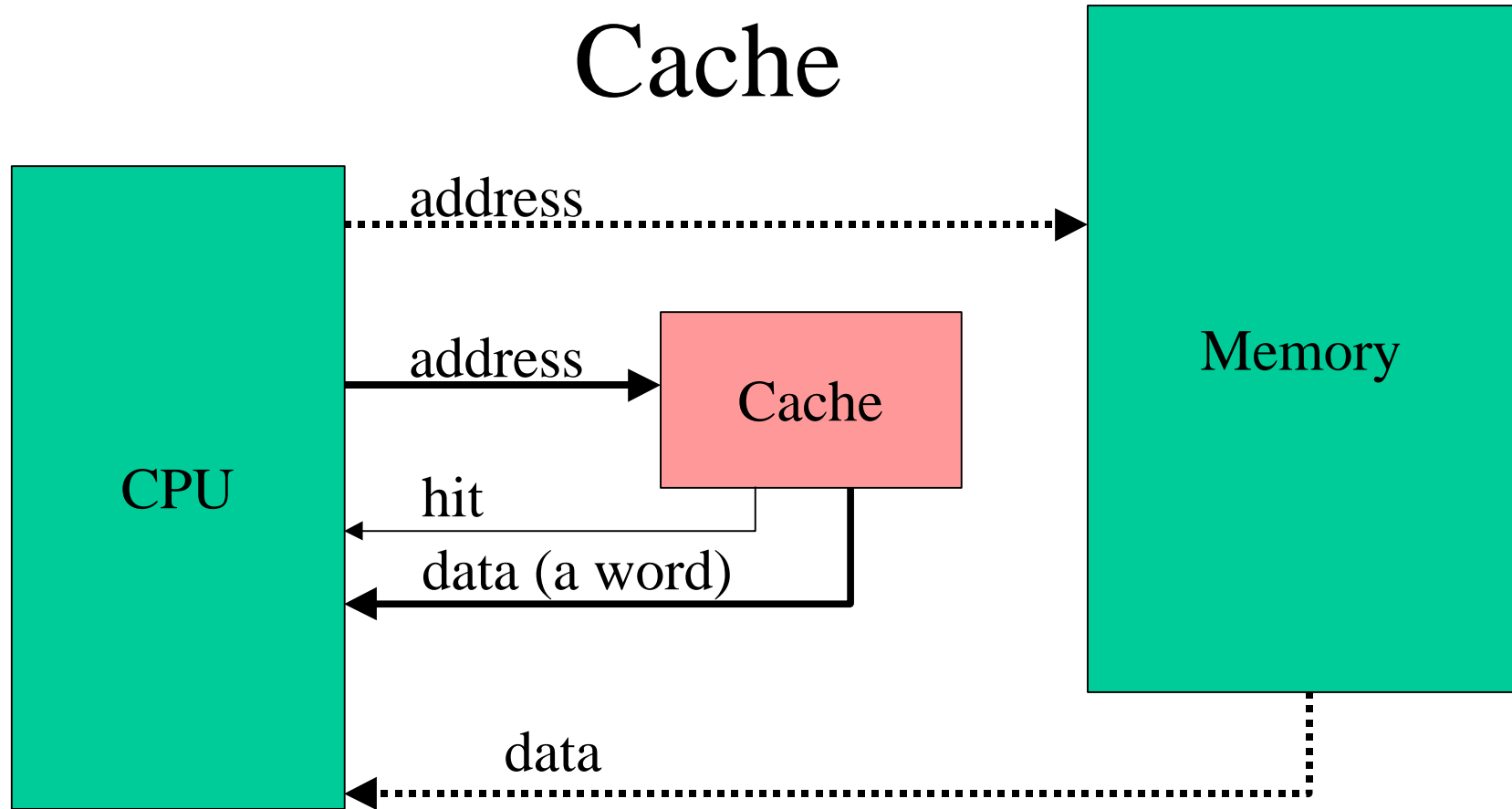
4kB, direct mapped



Cache Organization

4kB, direct mapped





Hit: Use the data provided from the cache

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

Cache performance parameters

- Cache “hit rate” [%]
 - Cache “miss rate” [%] ($= 1 - \text{hit_rate}$)
 - Hit time [CPU cycles]
 - Miss time [CPU cycles]
- Hit bandwidth
 - Miss bandwidth
 - Write strategy
 -

Cache performance example

Assumption:

Infinite bandwidth

A perfect 1.0 CyclesPerInstruction (CPI) CPU

100% instruction cache hit rate

NumberOfExecutionCycles =

#Instr. * (non_mem_ratio * 1 + mem_ratio * mem_avg_accesstime) =

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

CPI = NumberExecutionCycles / #Instructions =

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

Example Numbers

$$\text{CPI} = 1 - \text{mem_ratio} + \frac{\text{mem_ratio} * (\text{hit_rate} * \text{hit_time})}{\text{mem_ratio} * (1 - \text{hit_rate}) * \text{miss_time}}$$

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

$$\text{CPI} = 0.75 + 0.25 * 0.85 * 3 + 0.25 * 0.15 * 100 =$$

$$\frac{0.75}{\text{CPU}} + \frac{0.64}{\text{HIT}} + \frac{3.75}{\text{MISS}} = 5.14$$

What if ...

$$\text{CPI} = 1 - \text{mem_ratio} + \text{mem_ratio} * (\text{hit_rate} * \text{hit_time}) + \text{mem_ratio} * (1 - \text{hit_rate}) * \text{miss_time}$$

$$\text{mem_ratio} = 0.25$$

$$\text{hit_rate} = 0.85$$

$$\text{hit_time} = 3$$

$$\text{miss_time} = 100$$

$$\begin{array}{r} \text{CPU} \quad \text{HIT} \quad \text{MISS} \\ \Rightarrow 0.75 + 0.64 + 3.75 = 5.14 \end{array}$$

$$\text{Twice as fast CPU} \Rightarrow 0.37 + 0.64 + 3.75 = 4.77$$

$$\text{Faster memory (70c)} \Rightarrow 0.75 + 0.64 + 2.62 = 4.01$$

$$\text{Improve hit_rate (0.95)} \Rightarrow 0.75 + 0.71 + 1.25 = 2.71$$

How to get more effective caches:

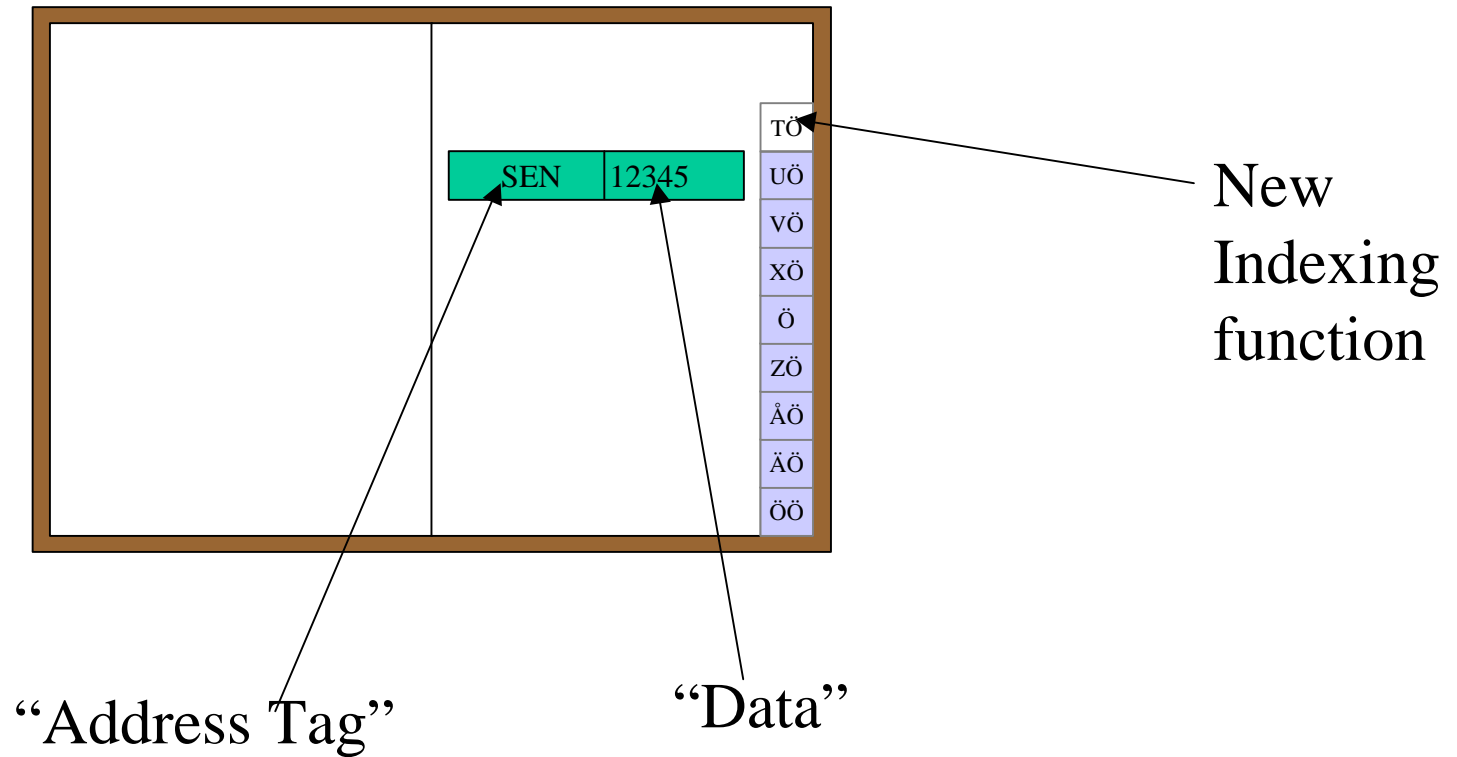
- Larger cache (more capacity)
- Cache block size (larger cache lines)
- More placement choice (more associativity)
- Innovative caches (victim, skewed, ...)
- Cache hierarchies (L1, L2, L3, CMR)
- Latency-hiding (weaker memory models)
- Latency-avoiding (prefetching)
- Cache avoiding (cache bypass)

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 (imperfect cache implementation)
- (Multiprocessors)
 - Communication (imposed by communication)
 - False sharing (side-effect from large cache blocks)

Huge Address Book

Lots of pages. One entry per page.

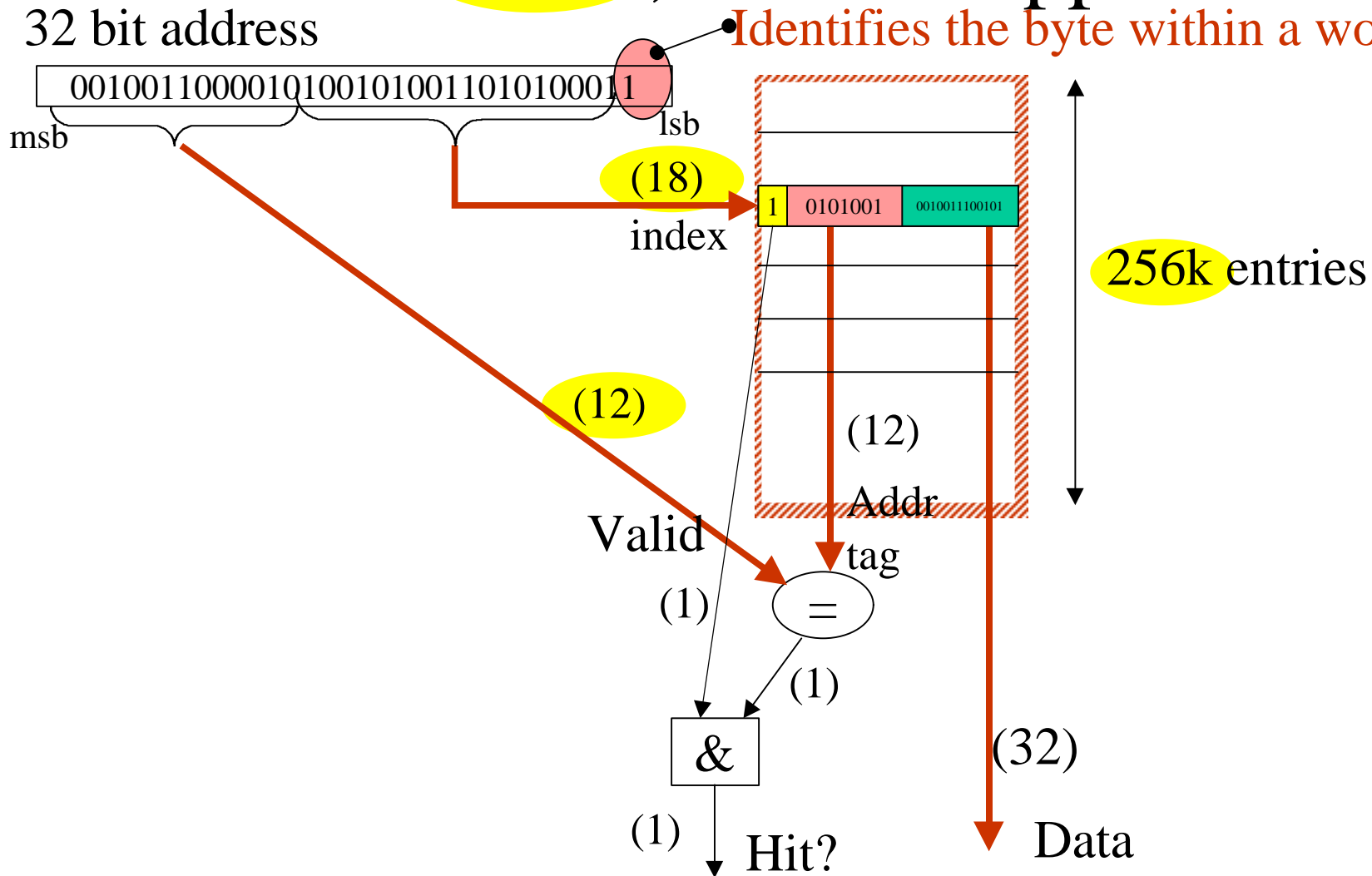


One entry per page =>
Direct-mapped caches with 28 entries

Cache Organization

1MB, direct mapped

Identifies the byte within a word



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)

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

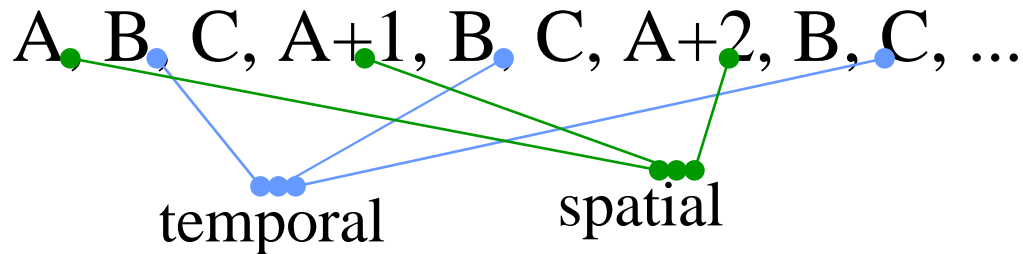
Typical access pattern:

(inner loop stepping through an array)

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

temporal

spatial

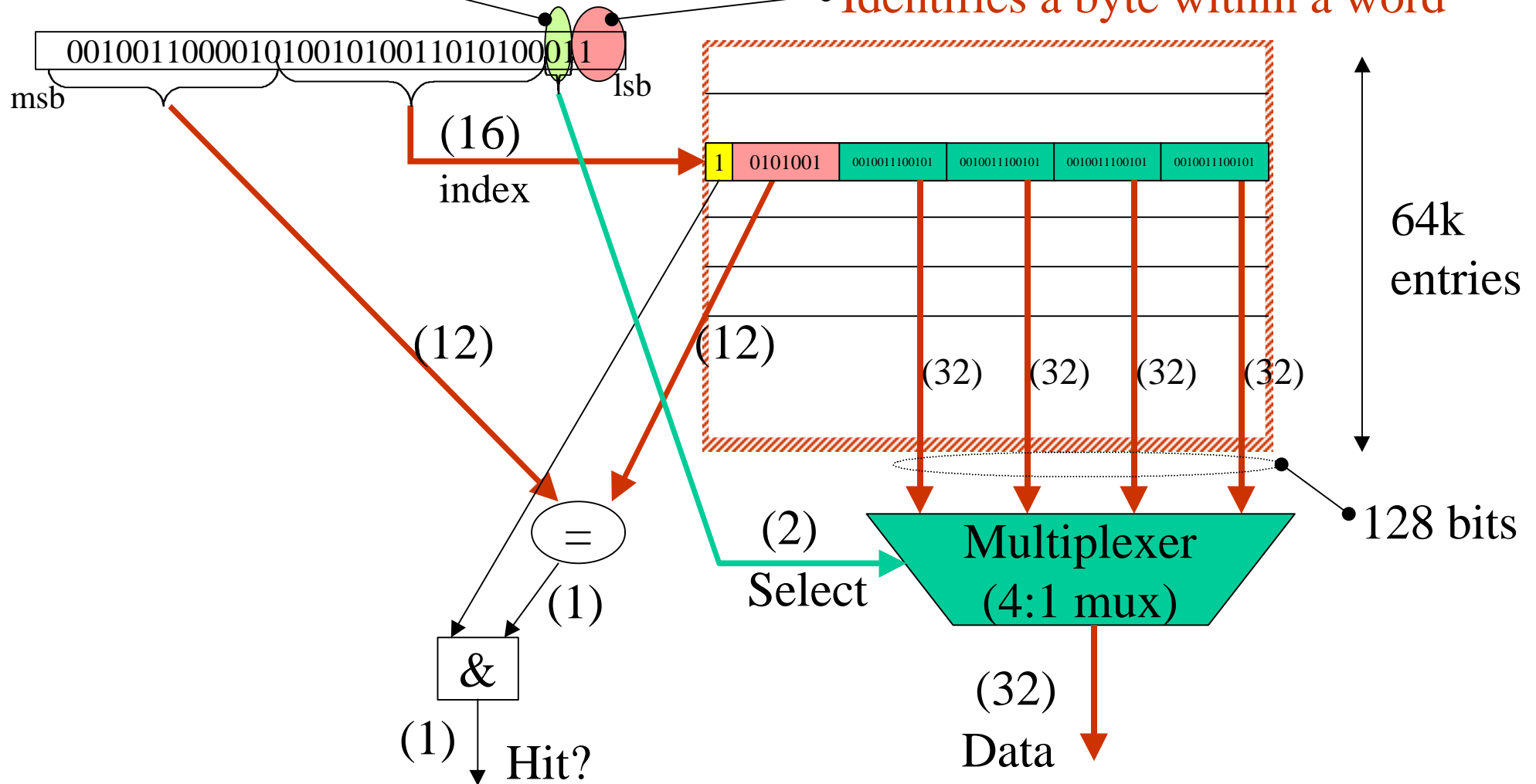


Fetch more than a word: cache blocks

1MB, direct mapped, CacheLine=16B

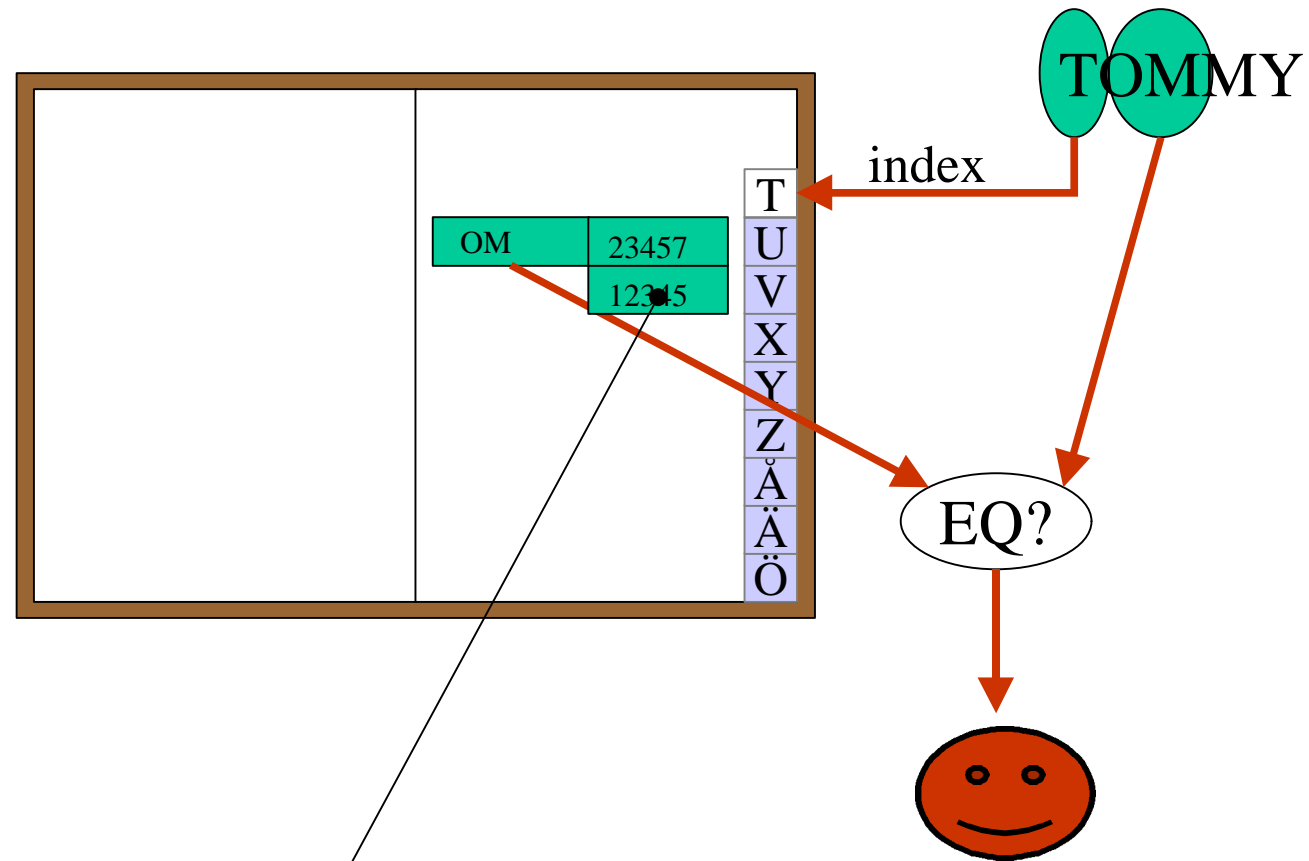
Identifies the word within a cache line

Identifies a byte within a word



Address Book Cache

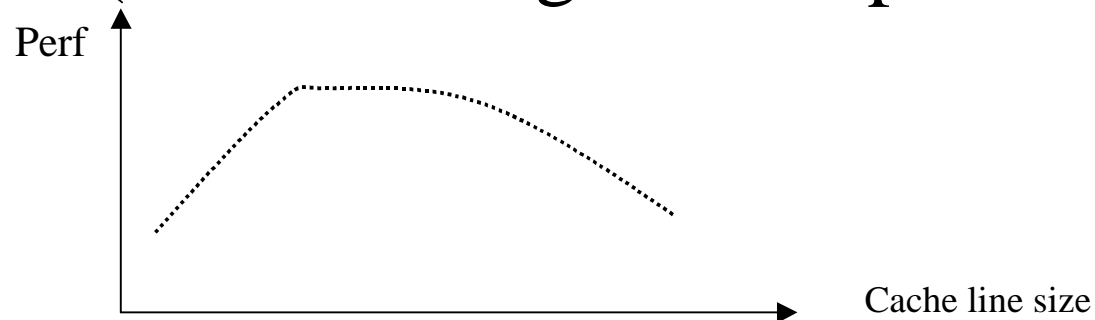
Looking for Tommy's Number



“Tommy’s number is the second starting on TOM”

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)

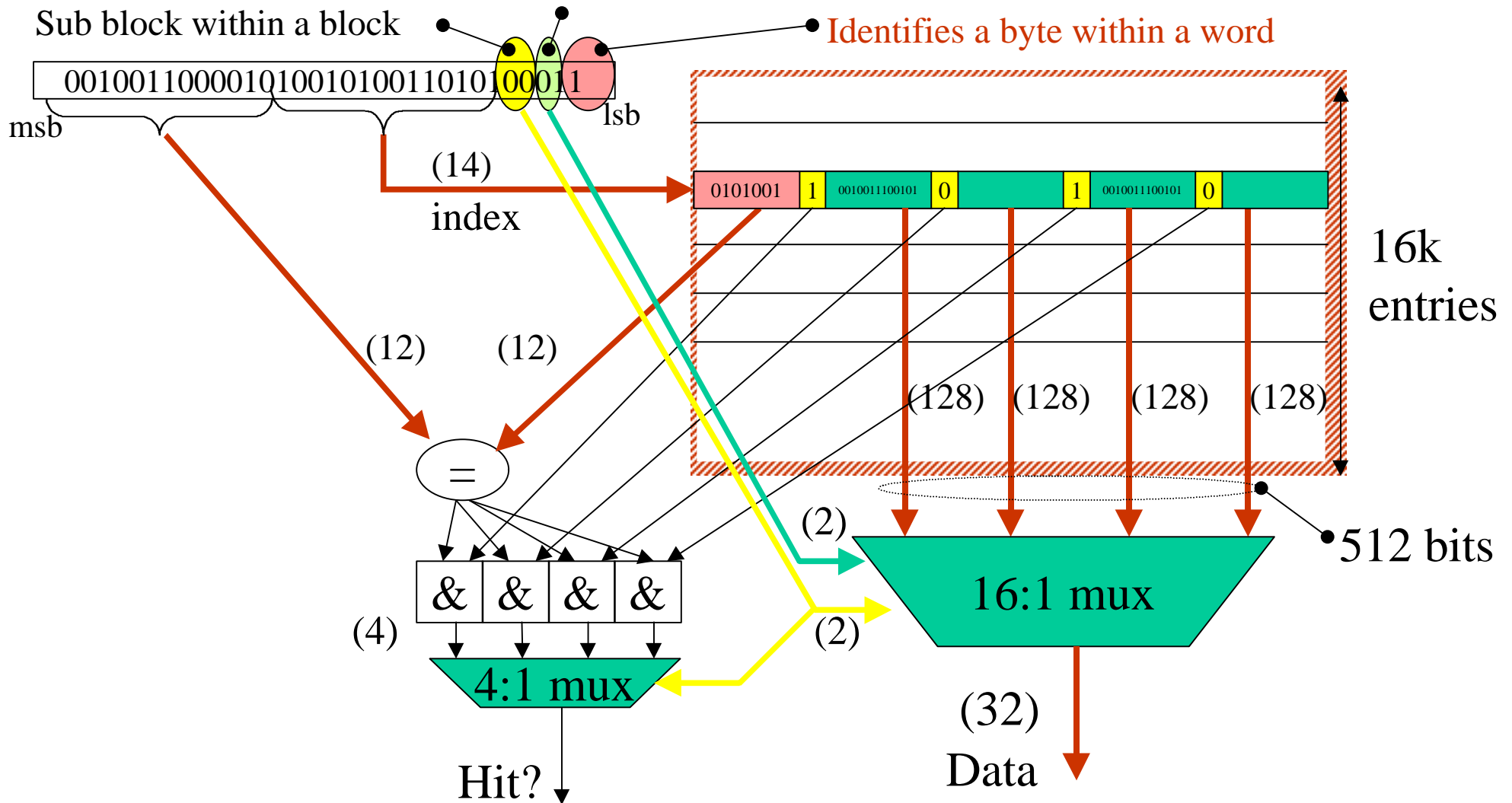


4-way sub-blocked cache

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

Identifies the word within a cache line

Identifies a byte within a word



Pros/Cons Sub-blocking

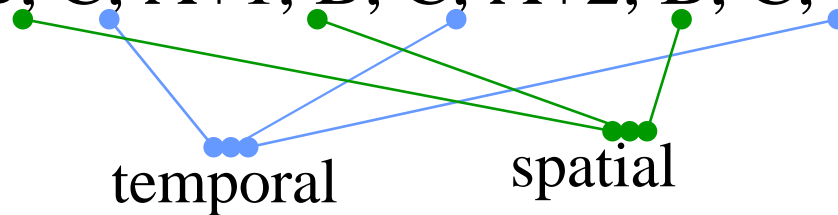
- ++ Needs much less address tags
- ++ Avoids problems with false sharing
- ++ Avoids problems with bandwidth waist
- Will not explore as much spatial locality
- Still poor utilization of SRAM
- Fewer sparse “things”

Why do you hit in a cache

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

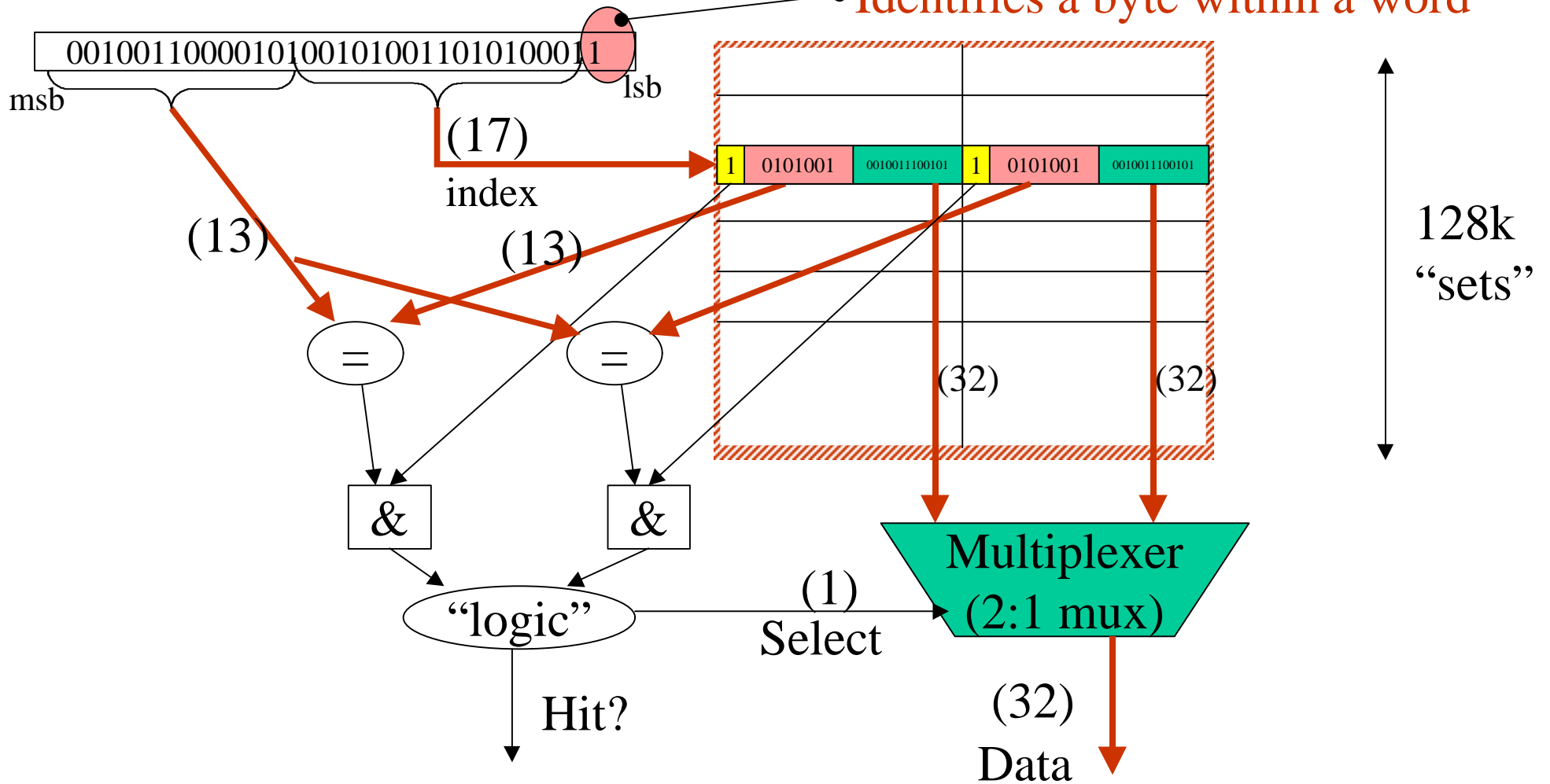
Conflict misses -- big time!

Potential performance loss 10-100x

Avoiding conflict: More associativity

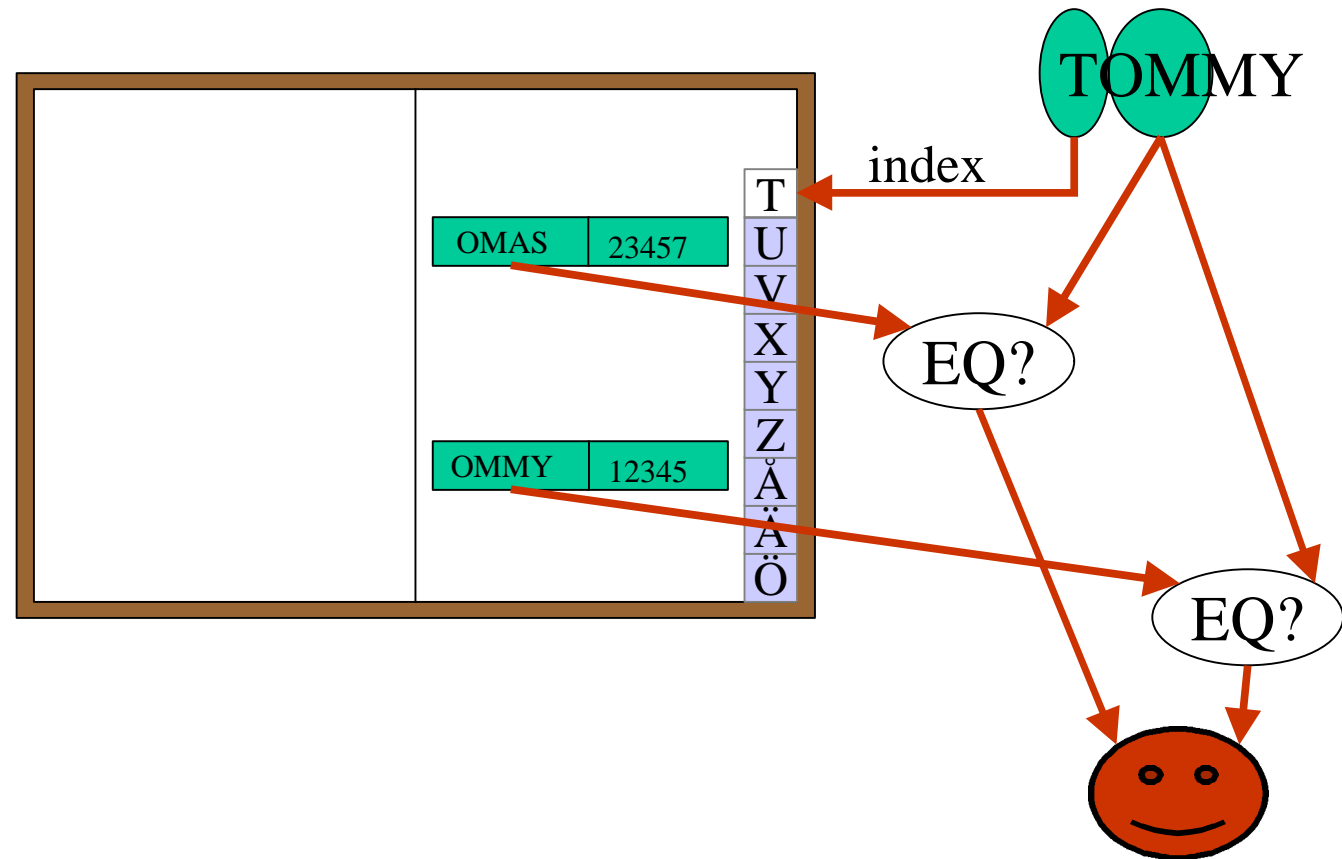
1MB, 2-way, CL=4B

Identifies a byte within a word



Address Book Cache

Two names per page: First index then search.

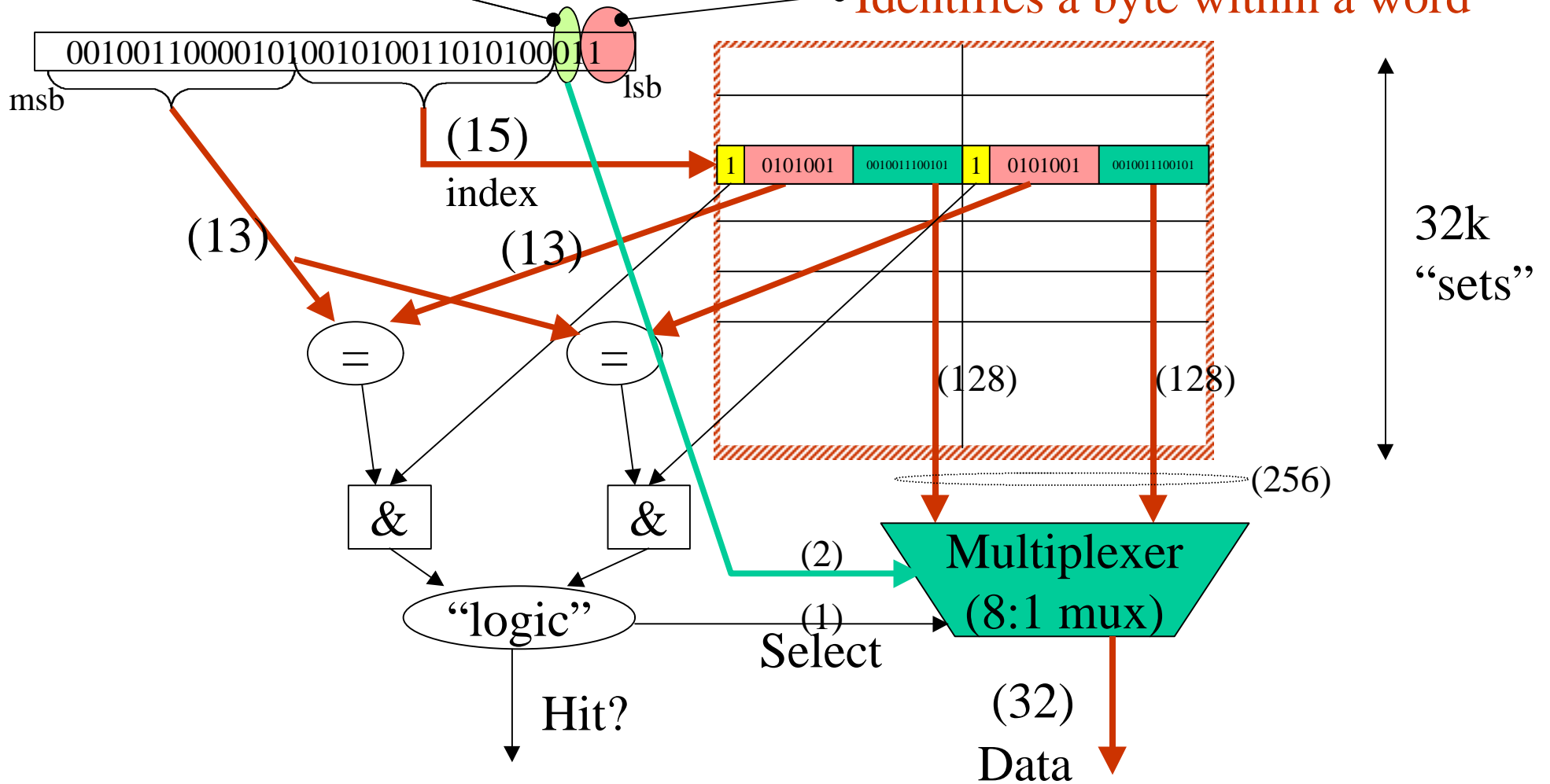


A combination there off

1MB, 2-way, CL=16B

Identifies the word within a cache line

Identifies a byte within a word

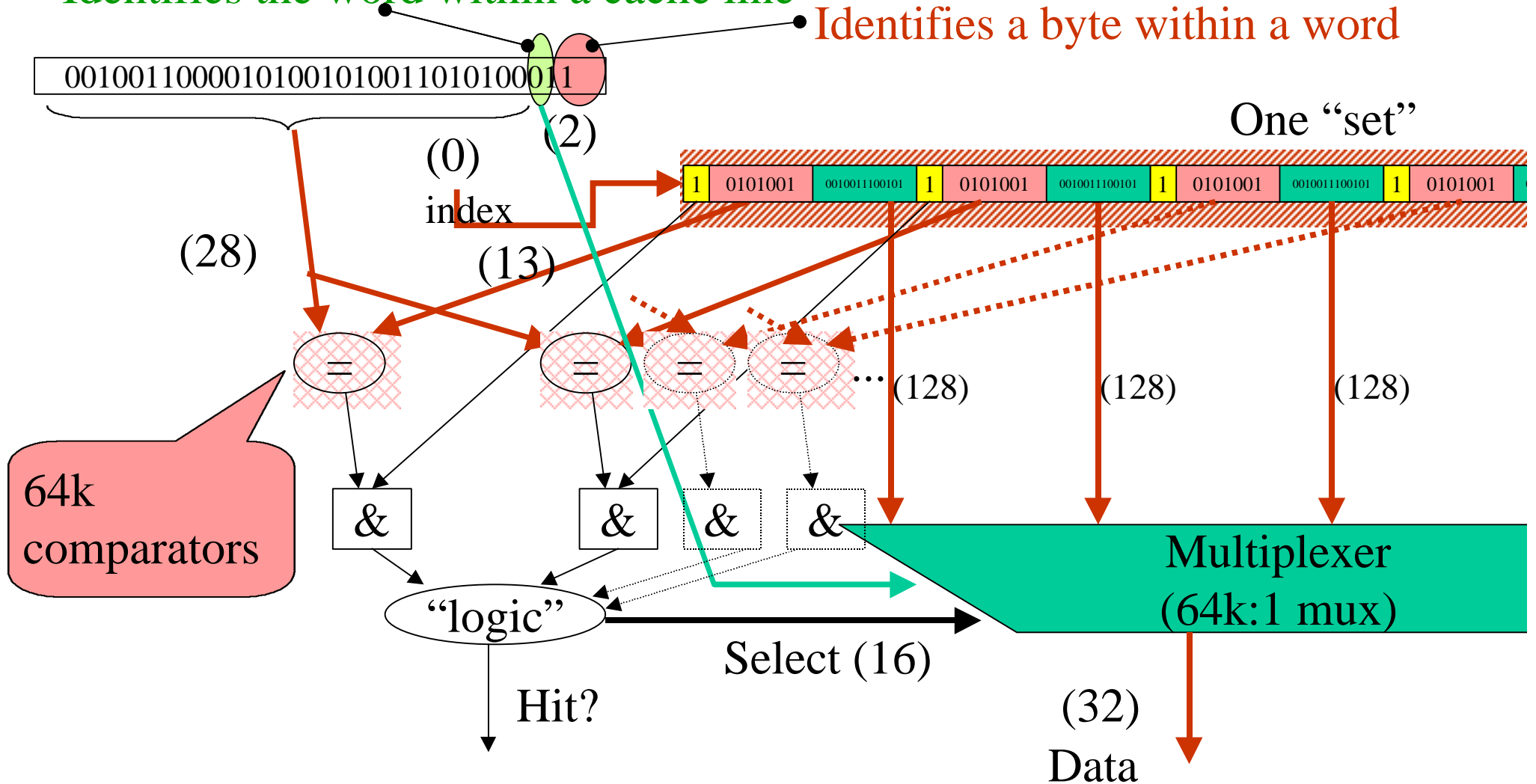


Going all the way...

1MB, fully associative, CL=16B

Identifies the word within a cache line

Identifies a byte within a word



Pros/Cons Associativity

- ++ Avoids conflict misses
- Slower access time
- More complex implementation
comparators, muxes, ...
- Requires more pins (for external SRAM...)

Who to replace?

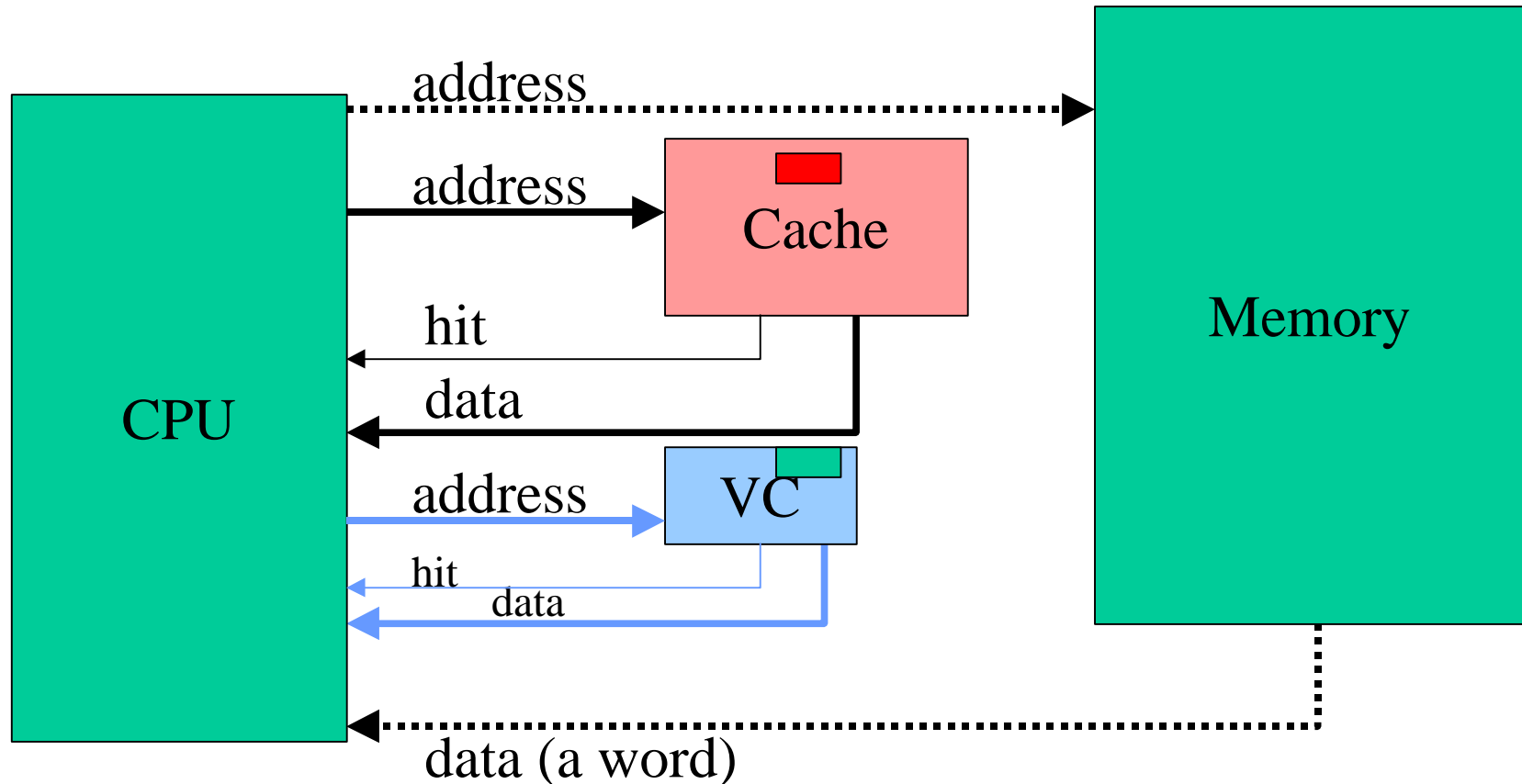
Picking a “victim”

- Least-recently used
 - 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...”

Replacing dirty cache lines

- Write-back
 - A “dirty bit” indicates an altered cache line
 - Write dirty data back to memory (next level) at replacement
- Write-through
 - Always write through to the next level (as well)
 - Never dirty data to write back

Innovative cache: Victim cache



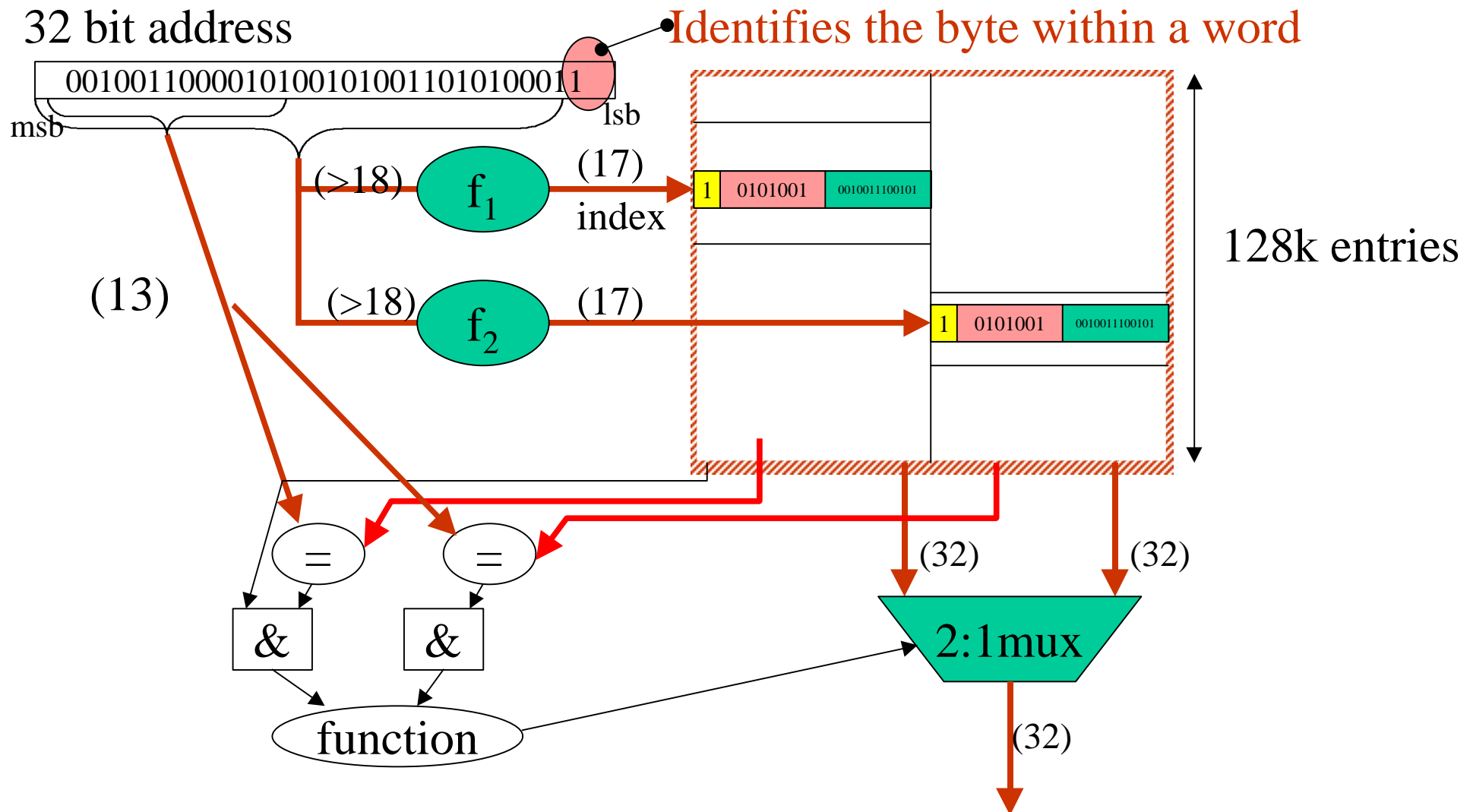
Victim Cache (VC): a small, fairly associative cache

Lookup: search cache and VC in parallel

Cache replacement: move victim to the VC

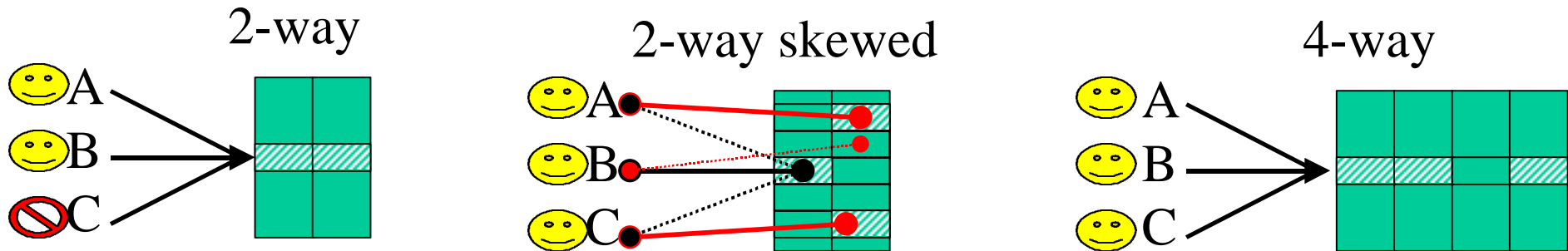
VC hit: swap VC data with the corresponding data in Cache

Skewed-associative cache: Different indexing functions



Skewed Associative Cache

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

Summing up Innovative caches

Victim cache:

- removes conflict misses

- cheap complement to direct mapped

Skewed associative:

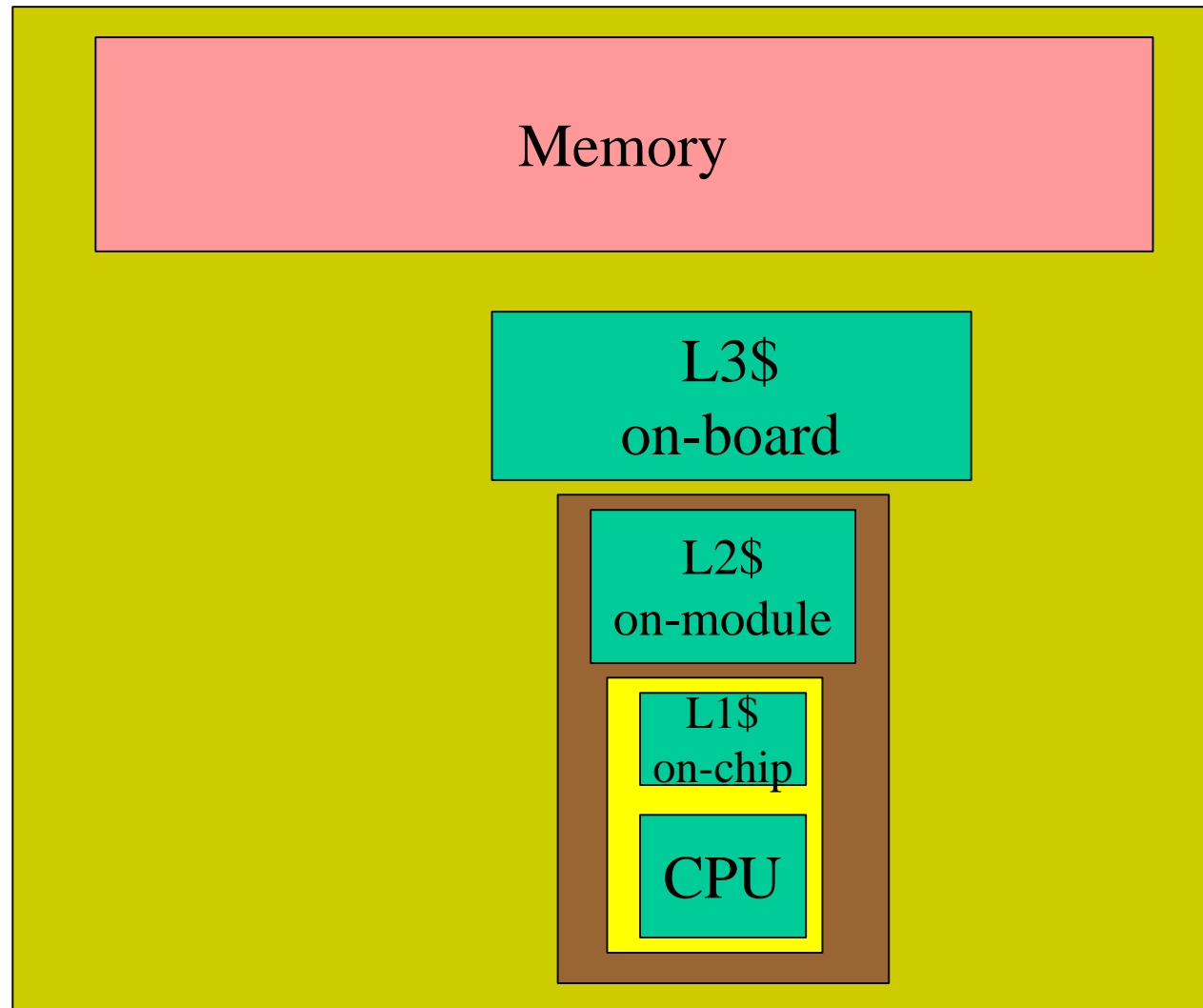
- great idea, never been commercially used

Cache Hierarchy

- On-chip cache is fast but small
- VIPT caches make adds constraints (later)
- Database application with huge datasets
- Latency 50:1 between on-chip SRAM - DRAM

==> Cache hierarchies

Cache Hierarchy



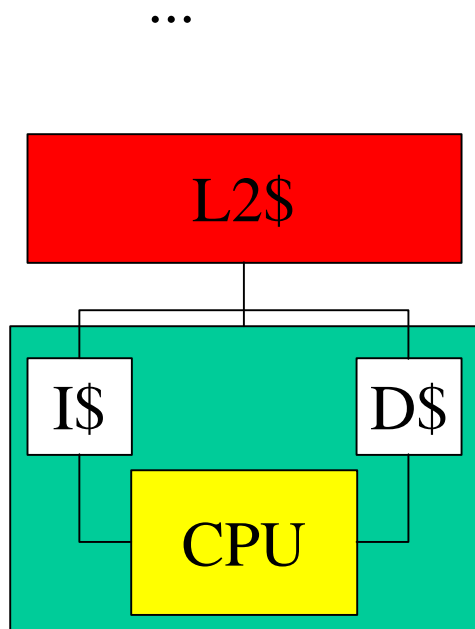
Topology of caches: Harvard Arch

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

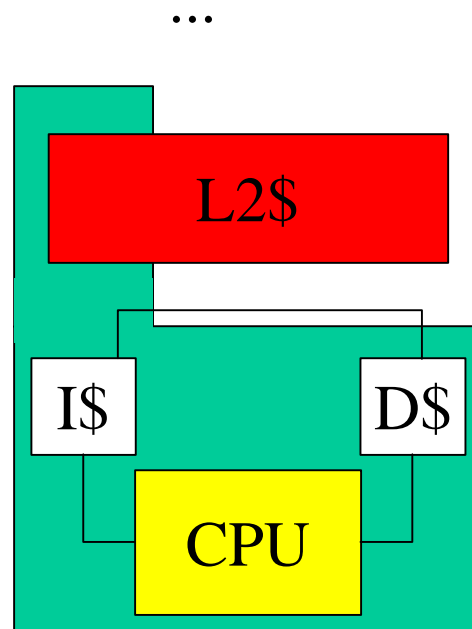
==> Separate D and I first level cache

==> Unified 2nd and 3rd level caches

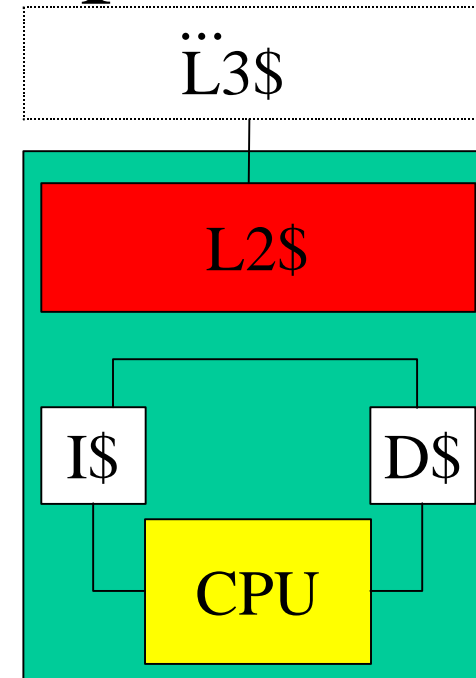
On-chip/Off-chip



L2 off chip
 ++ size
 -- associativity



L2 A-tags on chip
 ++ size,
 ++ miss time
 ?? associativity
 ++ snooping MP



L2 on chip
 -- size
 ++ hit time
 ++ associativity
 ++ snooping MP