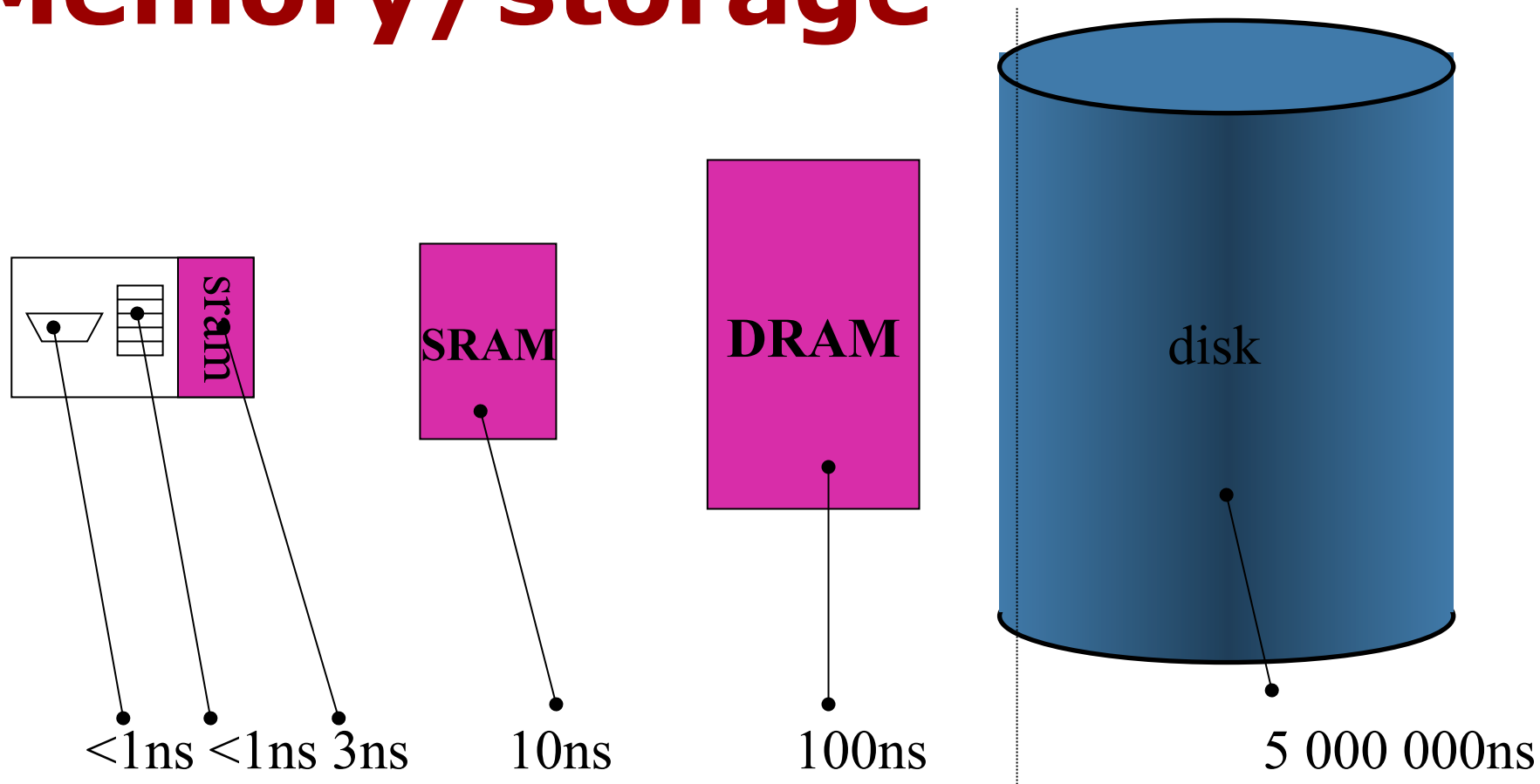
Caches and Memory System

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

Outline of these lectures

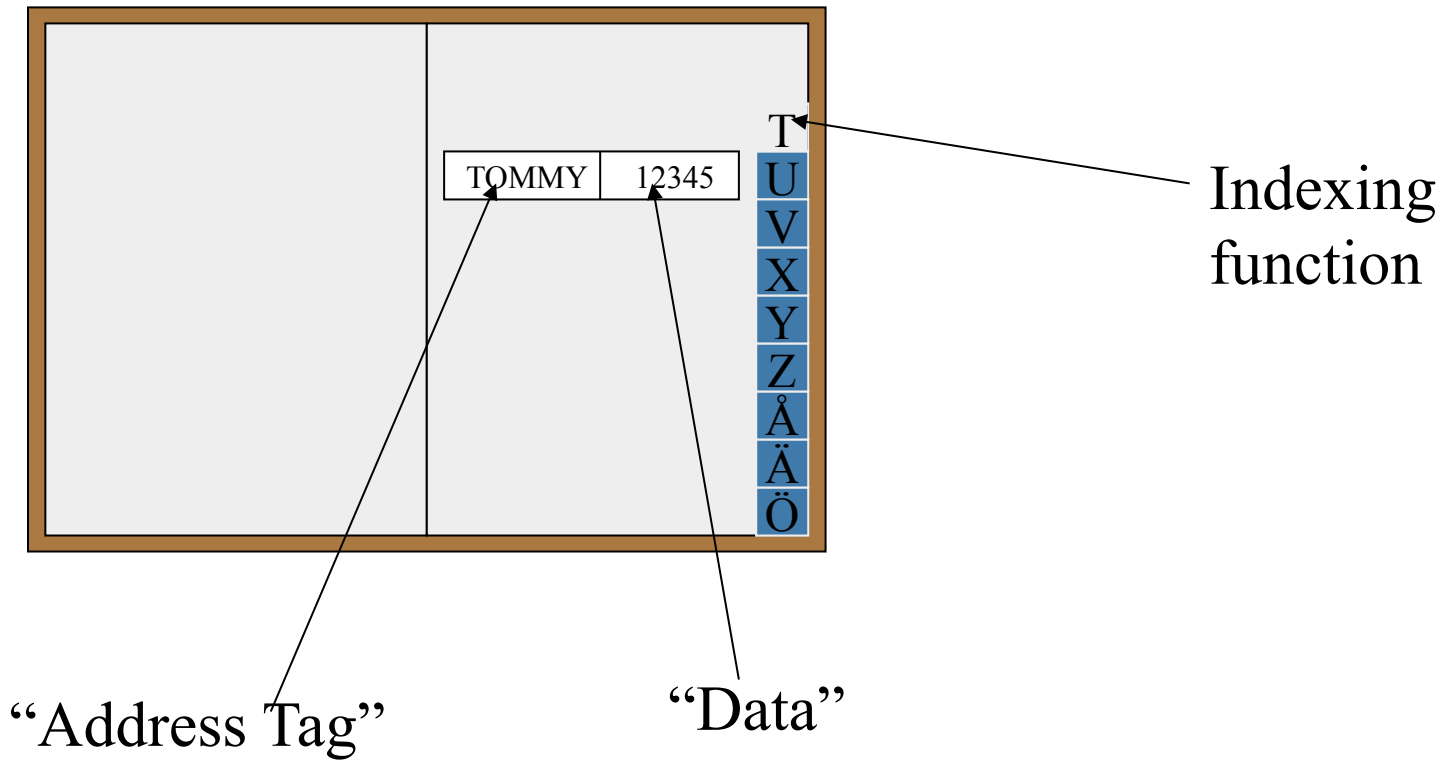
1. Processor implementations
2. **Caches and memory system**
3. Multiprocessors
4. HW optimizations
5. Multicore processors
6. SW optimizations

Memory/storage



Address Book Cache

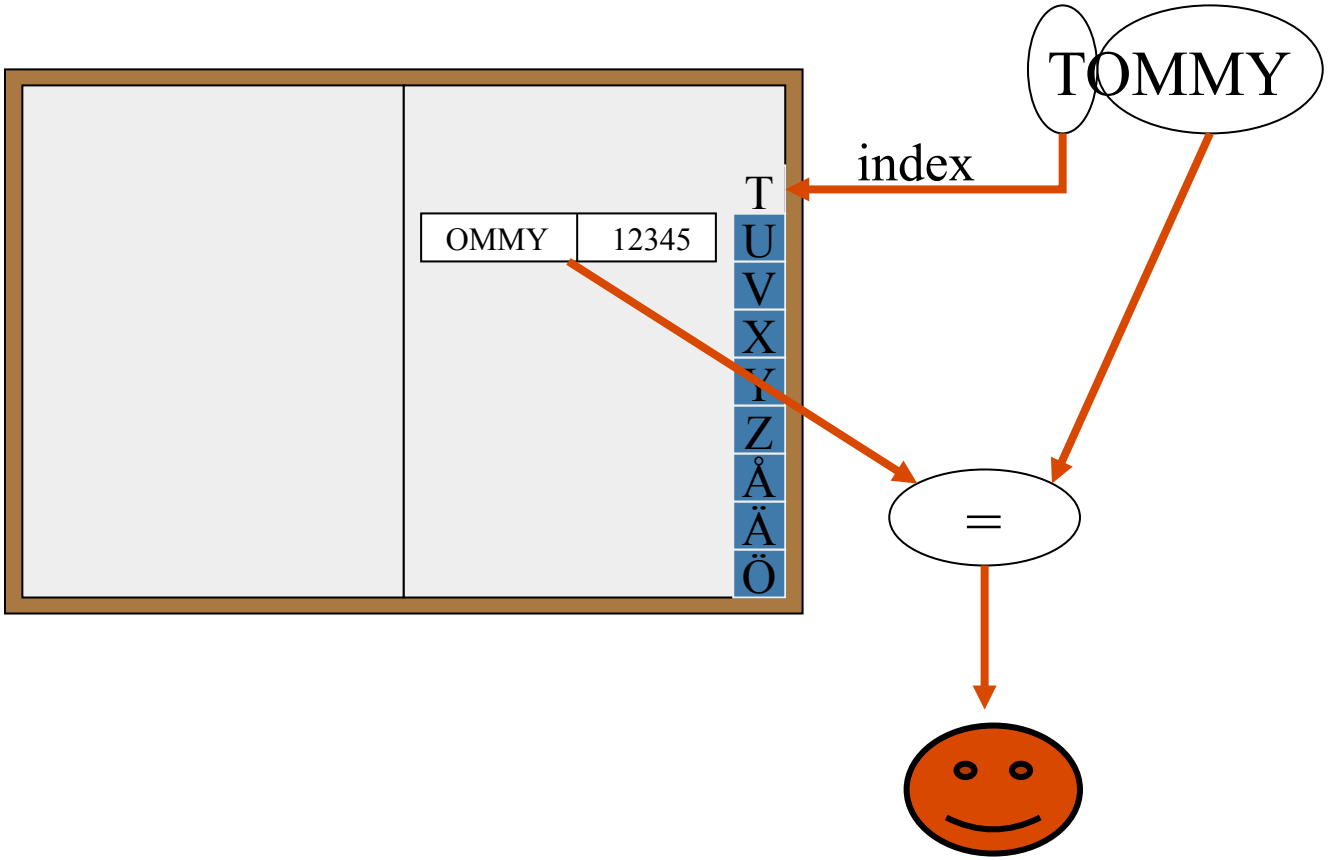
Looking for Tommy's Telephone Number



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

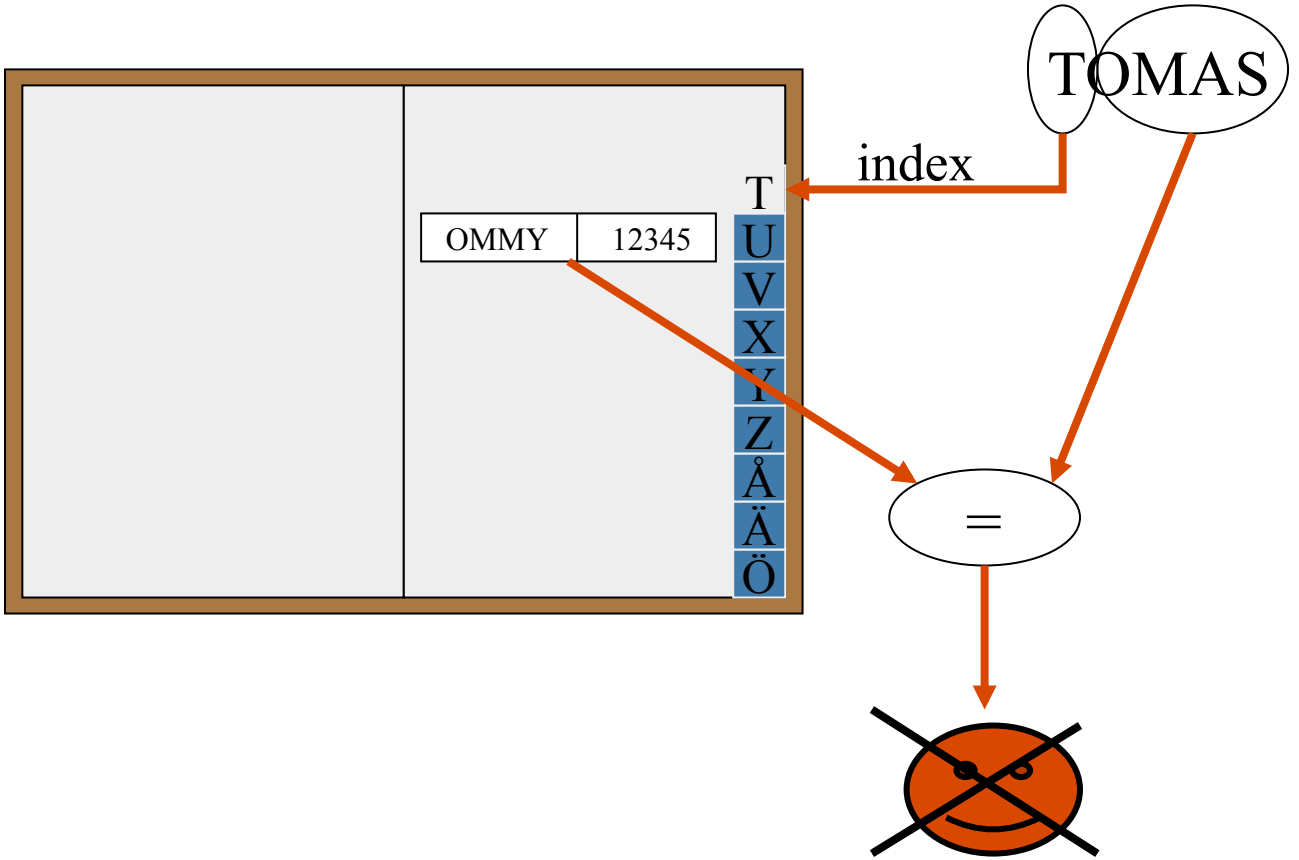
Address Book Cache

Looking for Tommy's Number



Address Book Cache

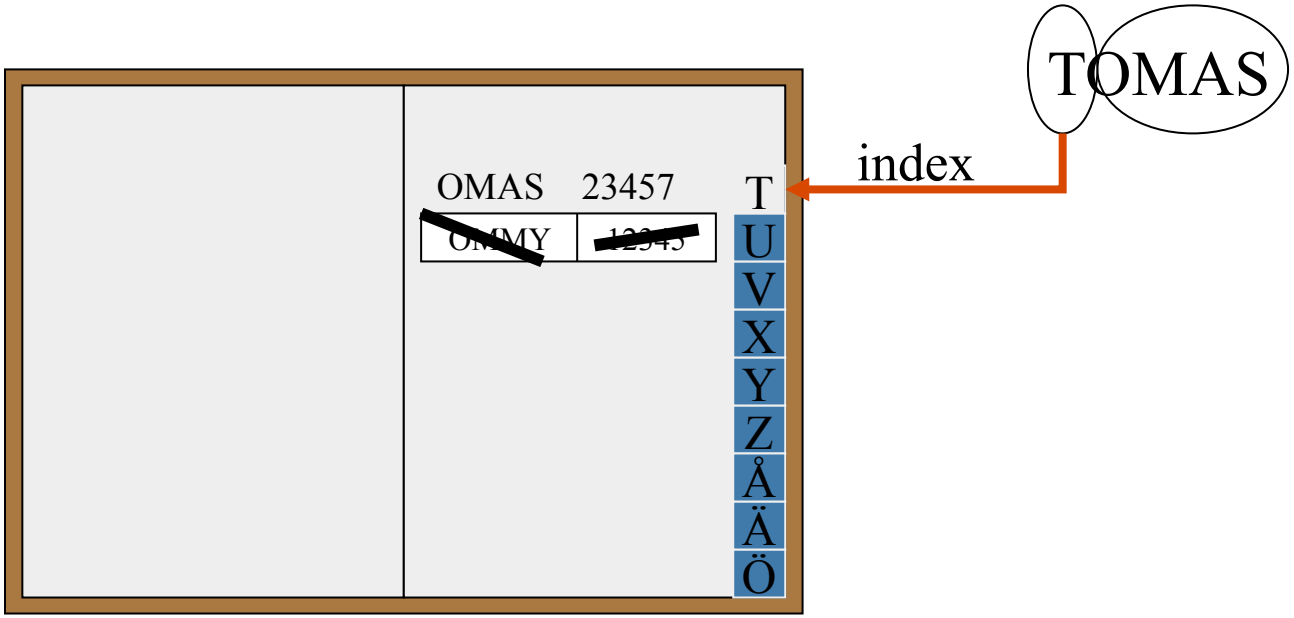
Looking for Tomas' Number



Miss!
Lookup Tomas' number in the telephone directory

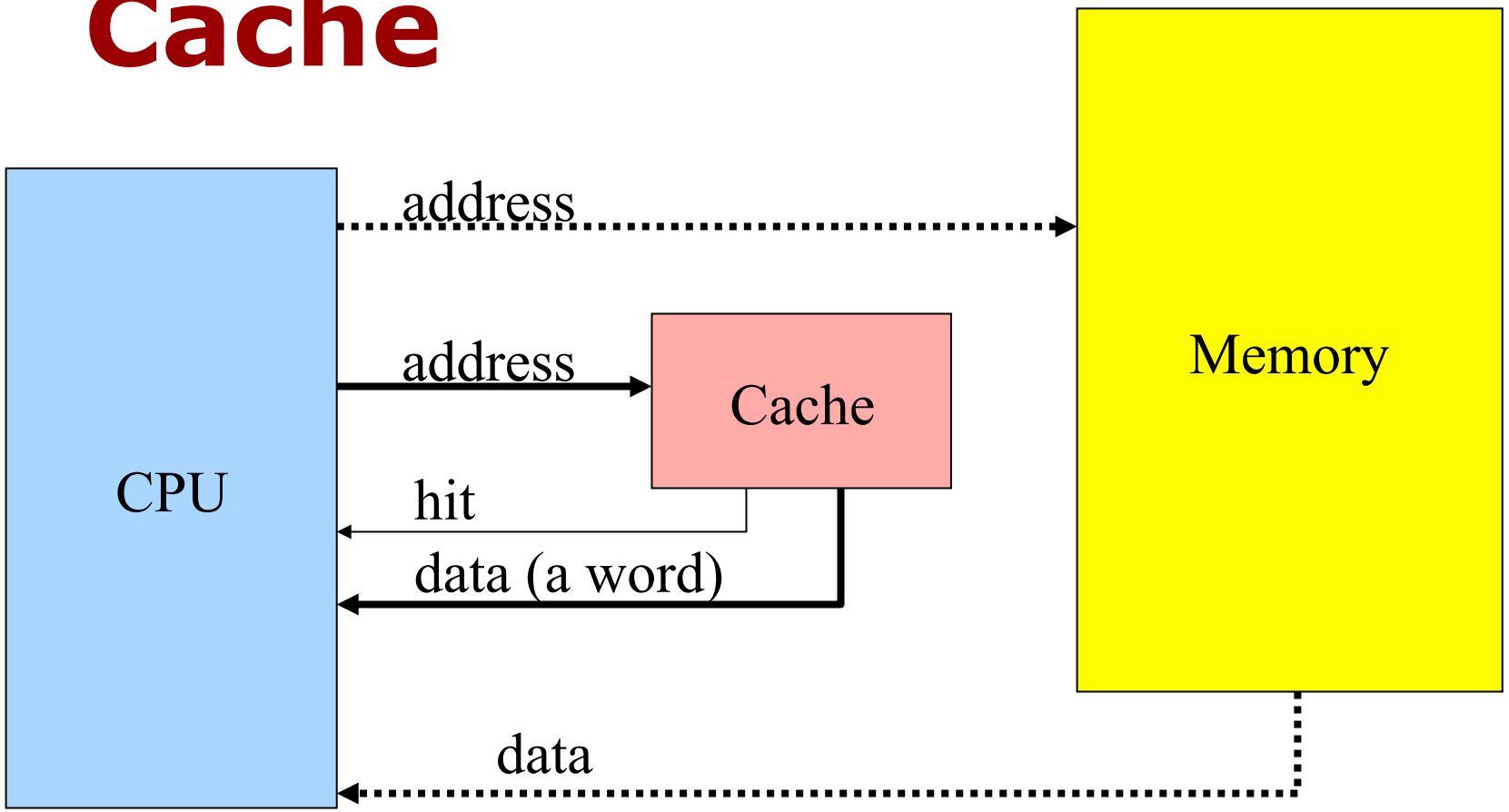
Address Book Cache

Looking for Tomas' Number

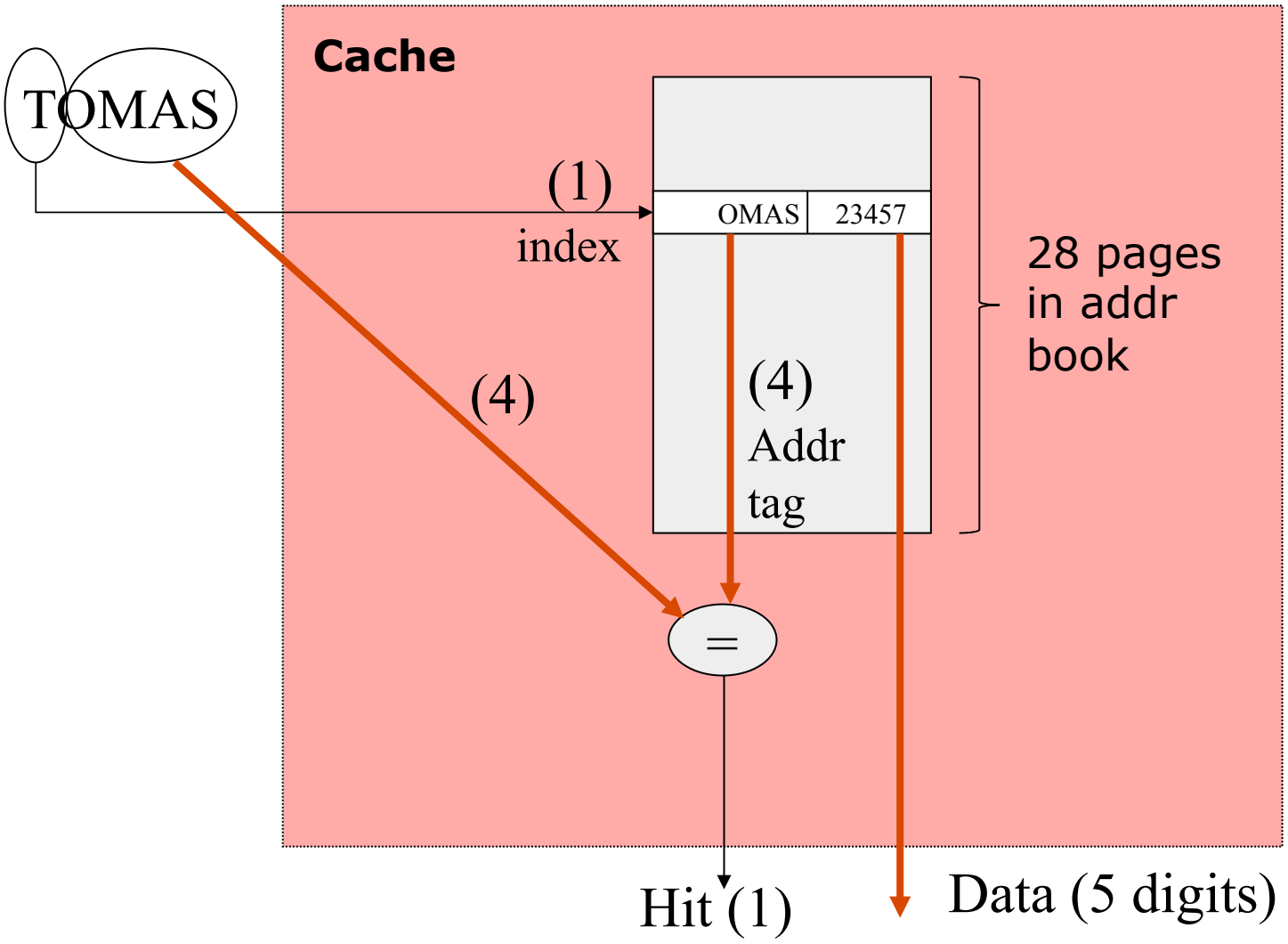


Replace TOMMY's data with TOMAS' data.
(Only one person per page = direct mapped cache)

Cache

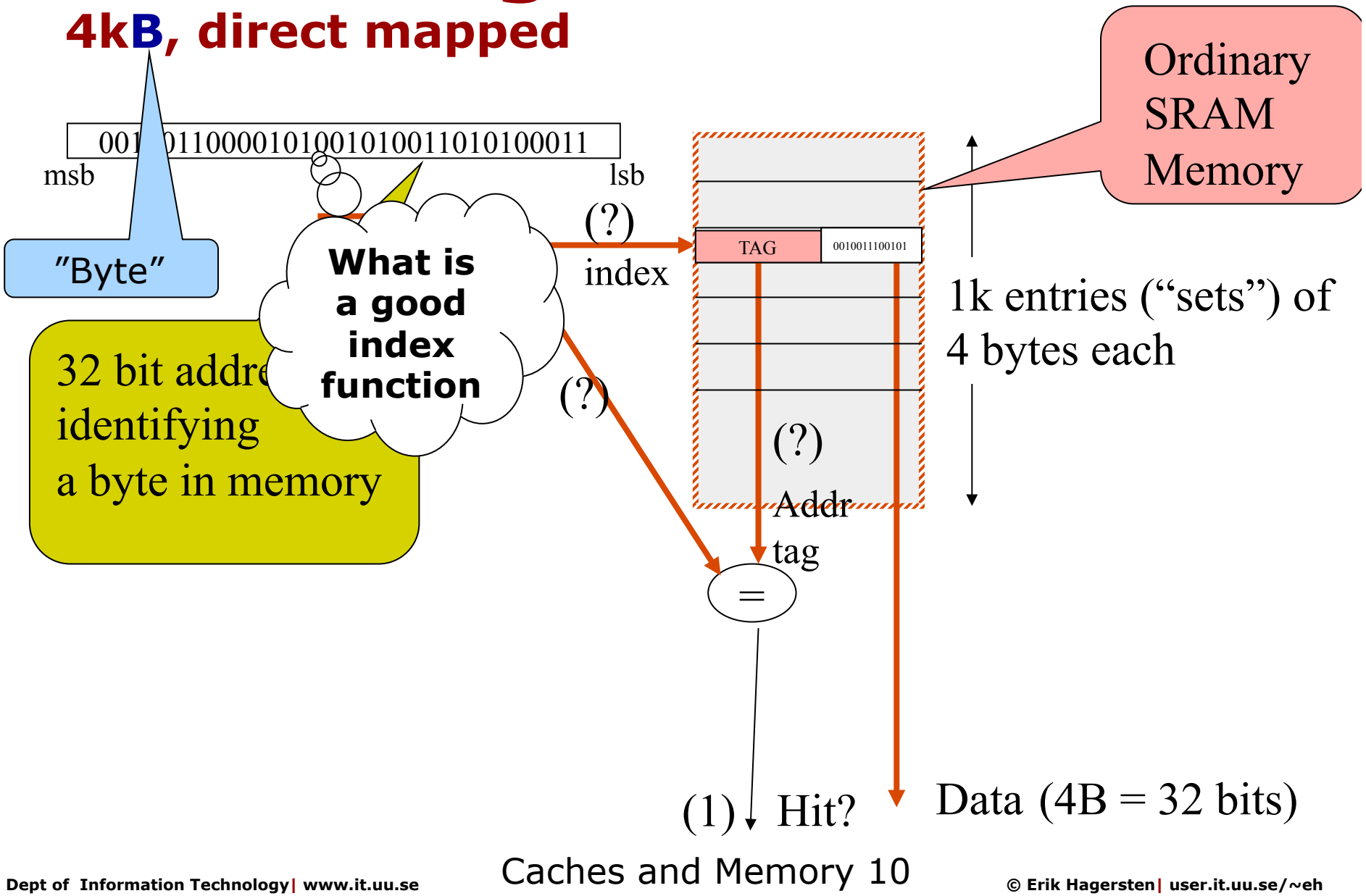


Cache Organization

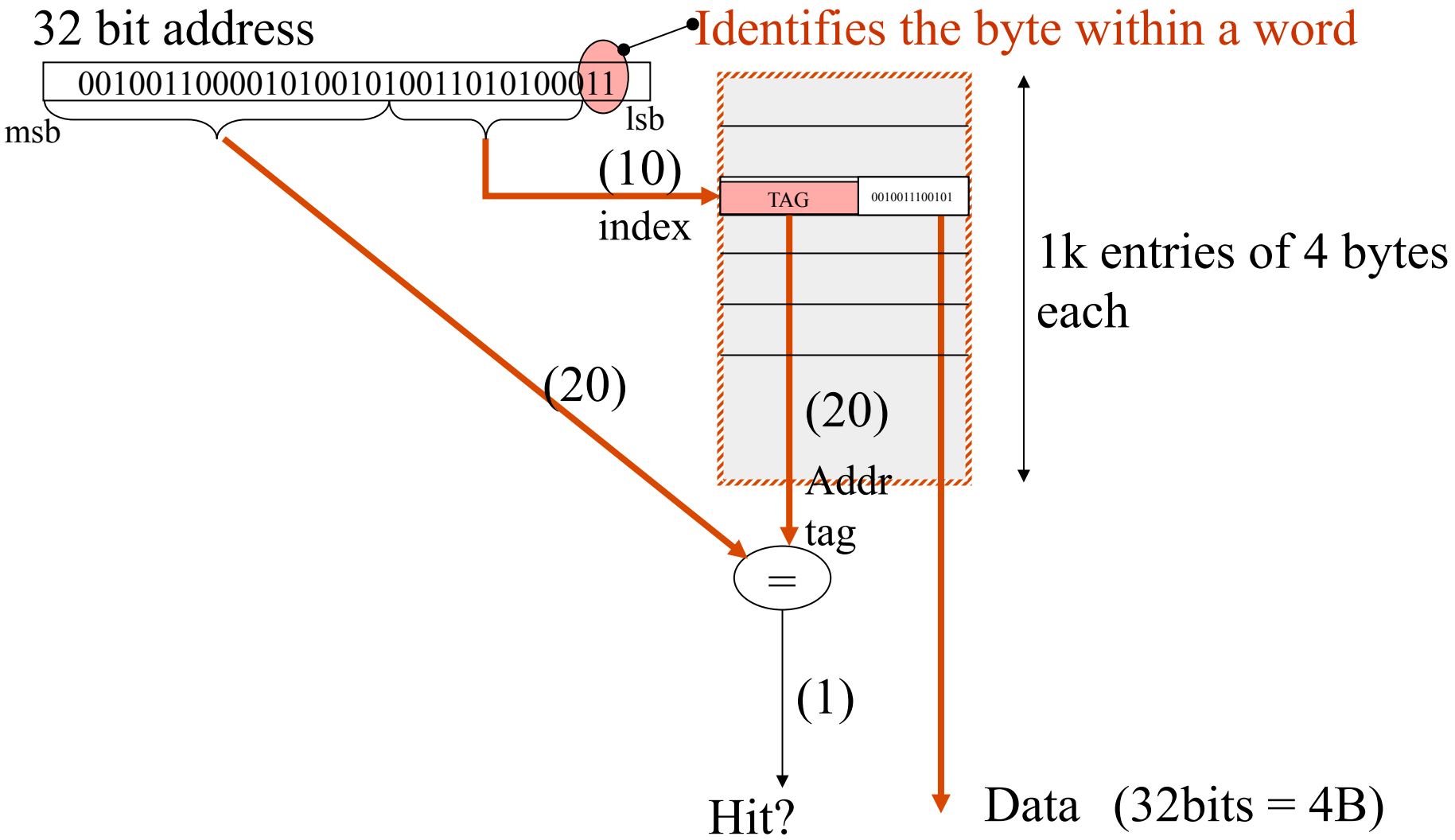


Cache Organization (really)

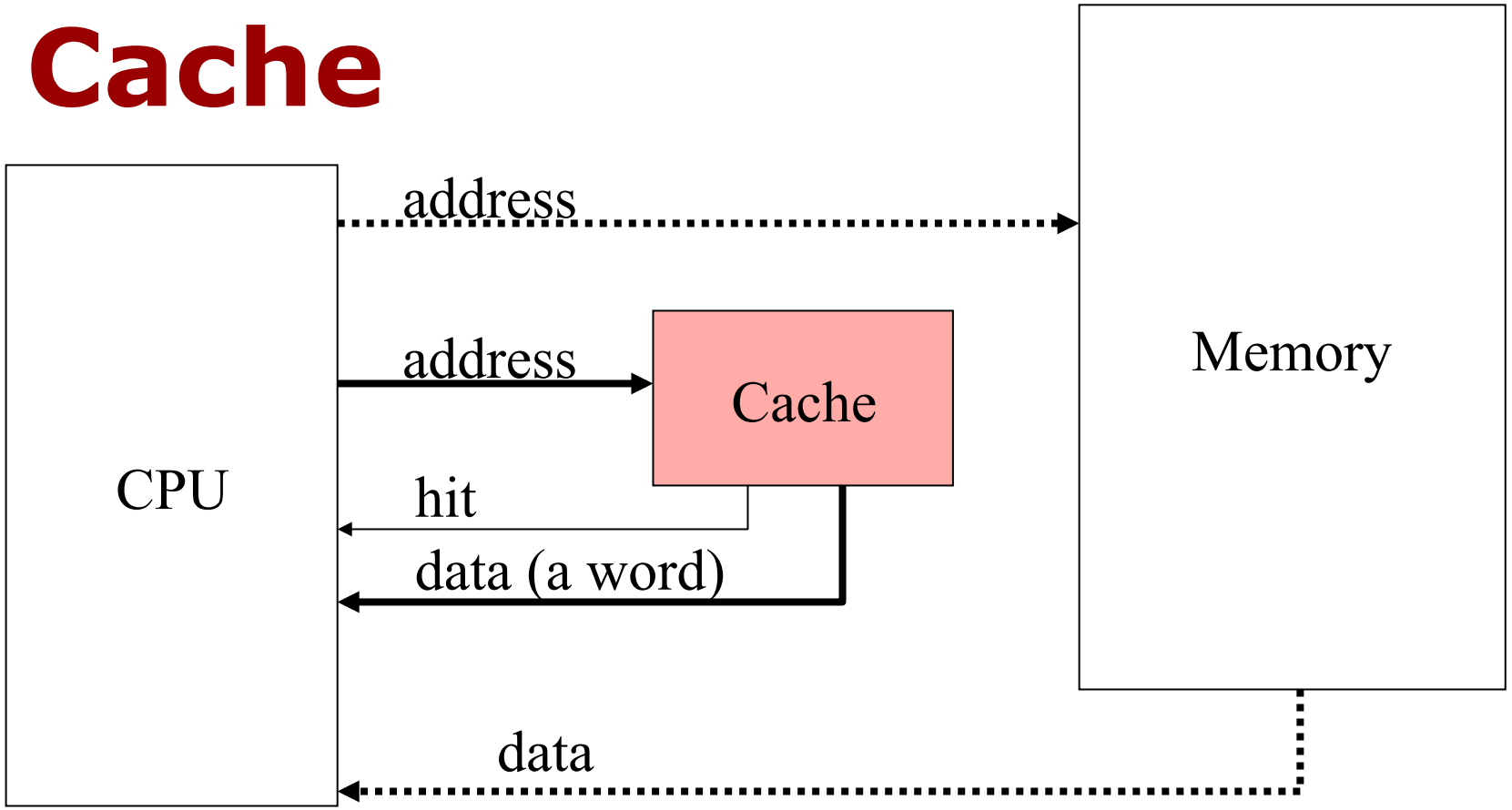
4kB, direct mapped



Cache Organization 4kB, direct mapped



Cache



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

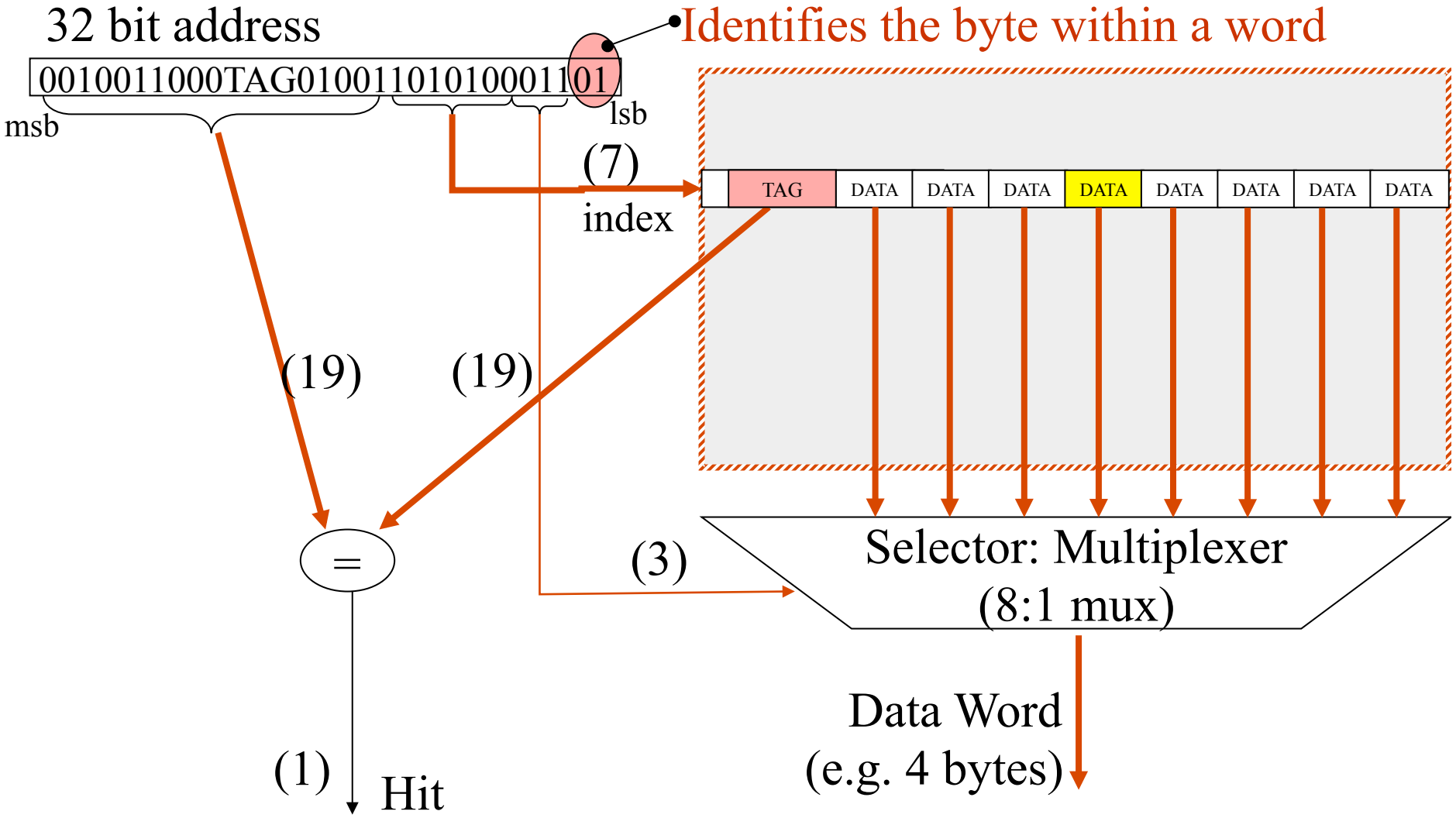
Why caches help:

- **Temporal locality:** Recently accessed data will likely be accessed again

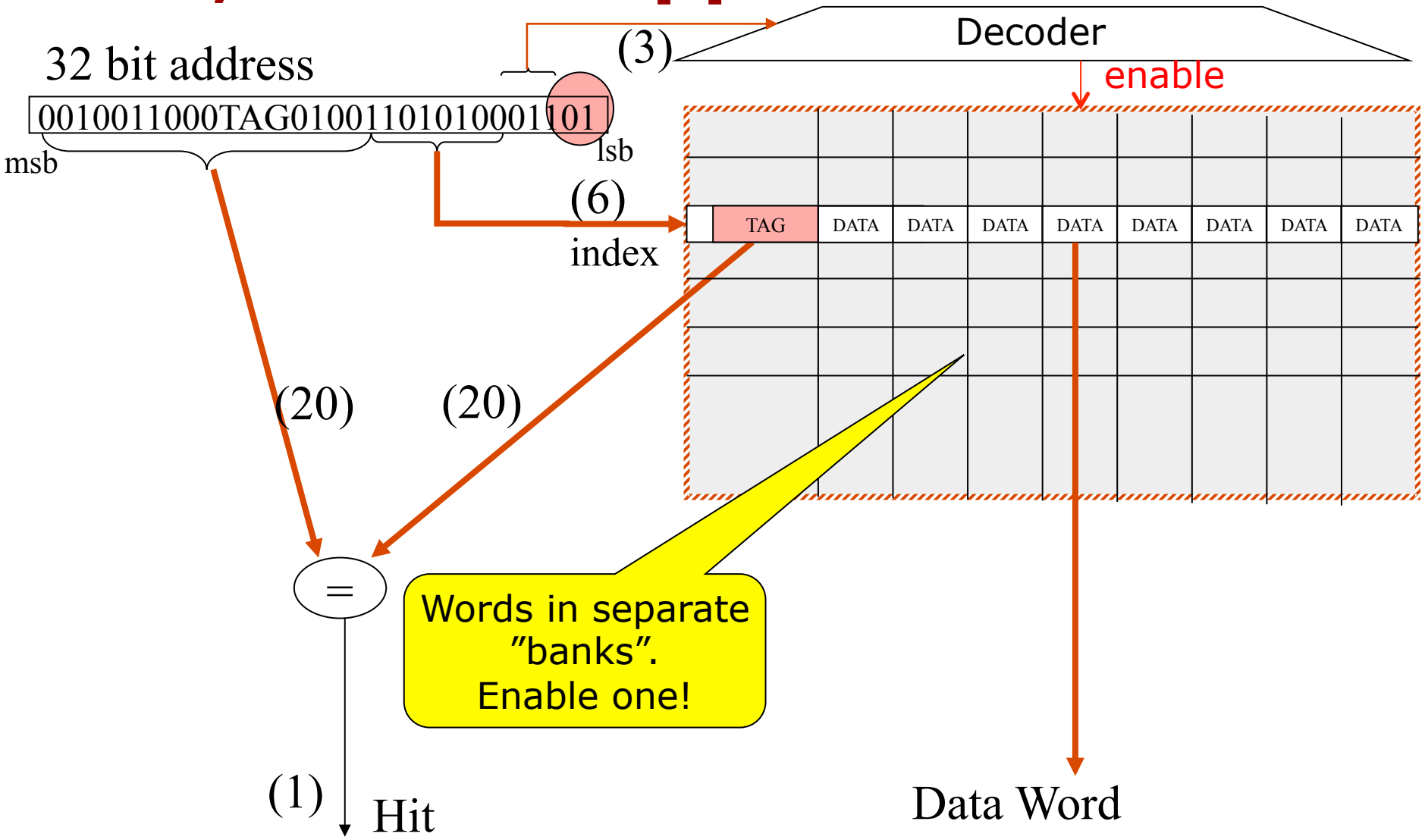
- **Spatial locality:** Data "near" recently accessed data will likely be accessed soon

➔ Data is moved to/from memory in large chunks (cachelines)

Cache: Logic Organization 4kB, CacheLine = 32B



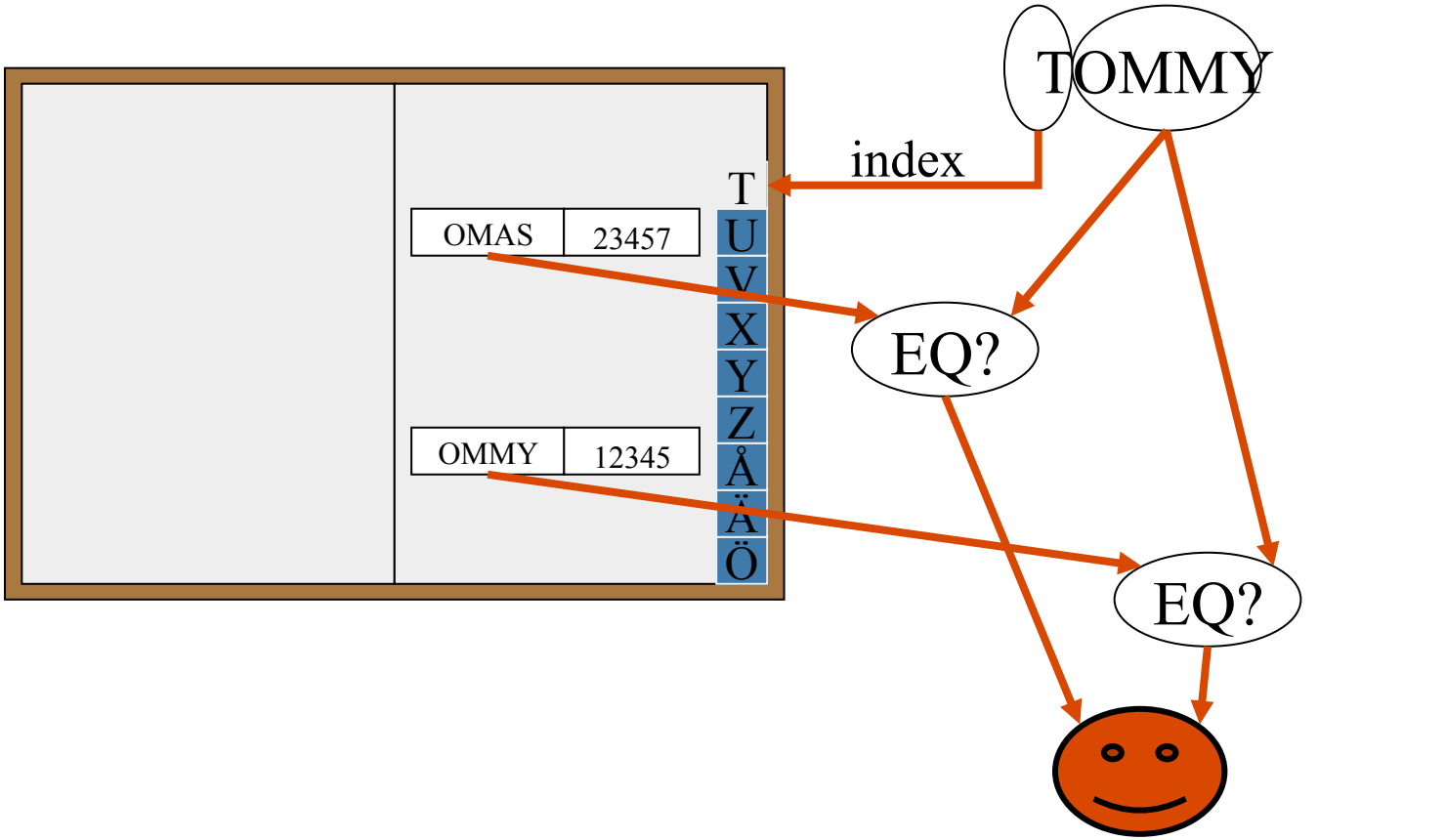
Cache Physical Organization 4kB, direct mapped



Address Book Analogy

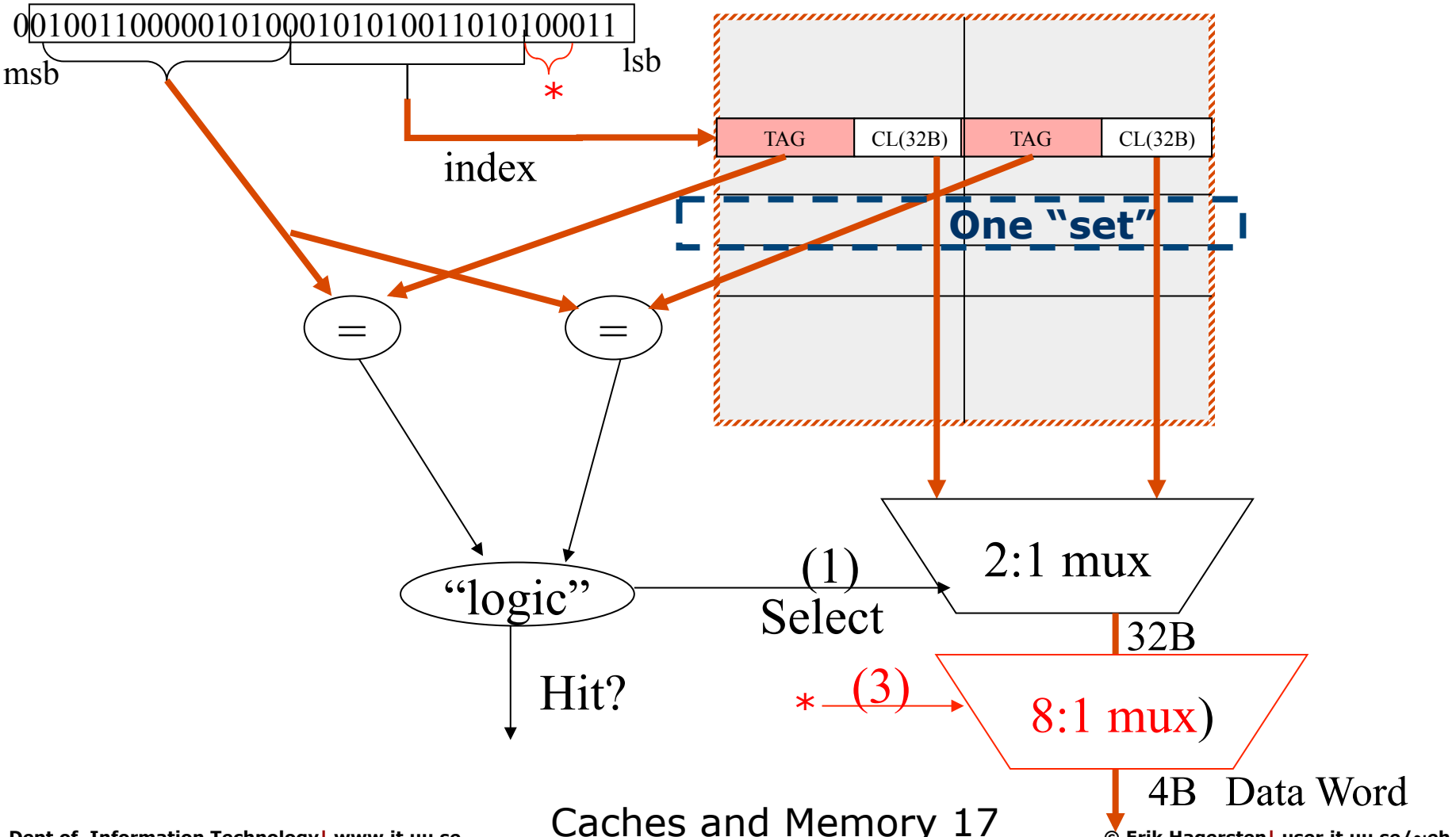
"2-way set-associative cache"

Two names per page: index first, then search.



Select the second entry!

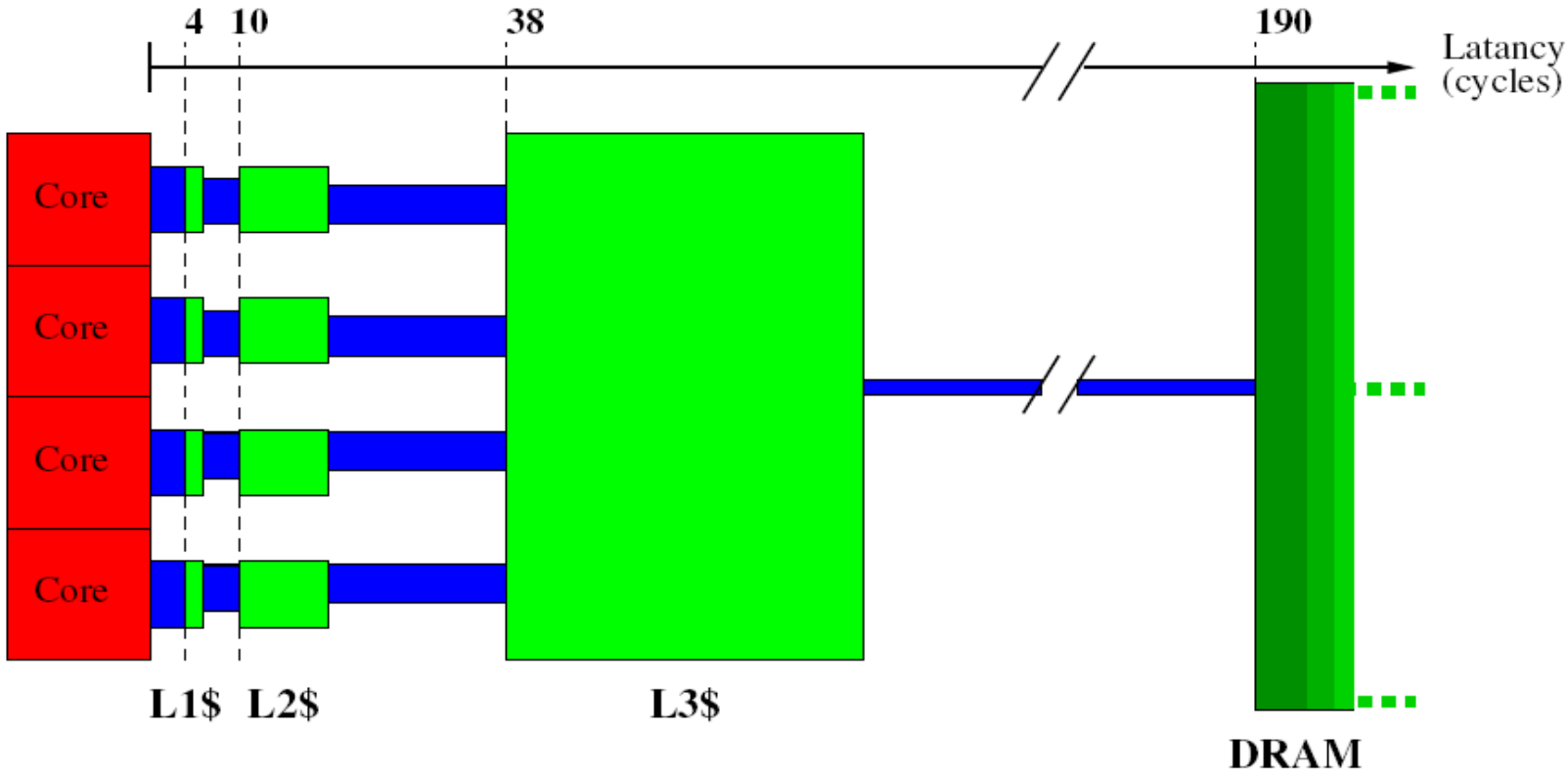
Avoiding conflict: Associativity



Who to replace?
Picking a "victim"

- Least-recently used (LRU)
  - Considered the best algorithm
  - Only practical up to 4-way (16 bits/CL)
- Pseudo-LRU
  - Course Time stamps, used in the VM system
  - Tree of binary LRU pointers
- Random replacement
  - Can't continuously have "bad luck...
- ...

Cache Capacity/Latency/BW



Cache implementation
"8-way set-associative cache"

Caches at all level roughly work like this:

L3 € 24MB

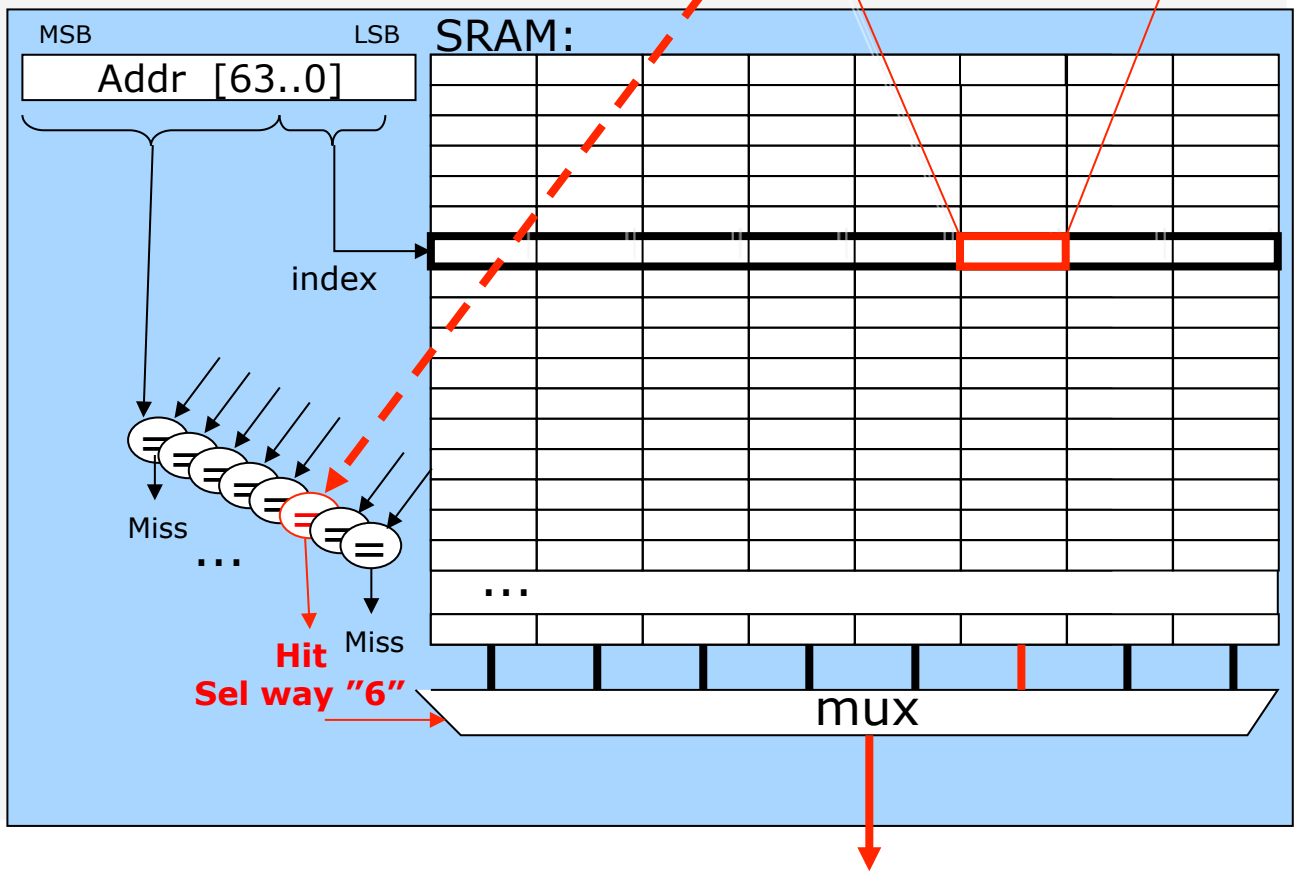
L2 $ 256kB

D1 ¢ 64kB

I1 ¢ 64kB

Cacheline, here 64B:

Generic Cache:



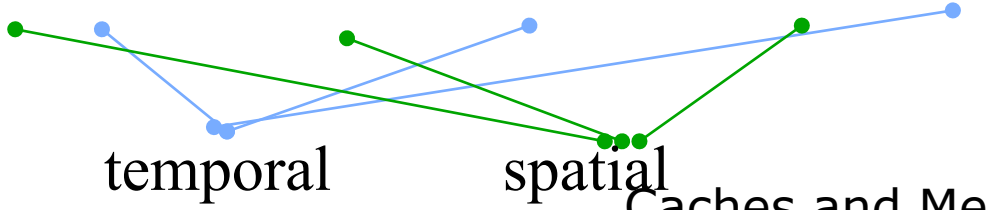
Cache lingo

- **Cacheline:** Data chunk moved 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

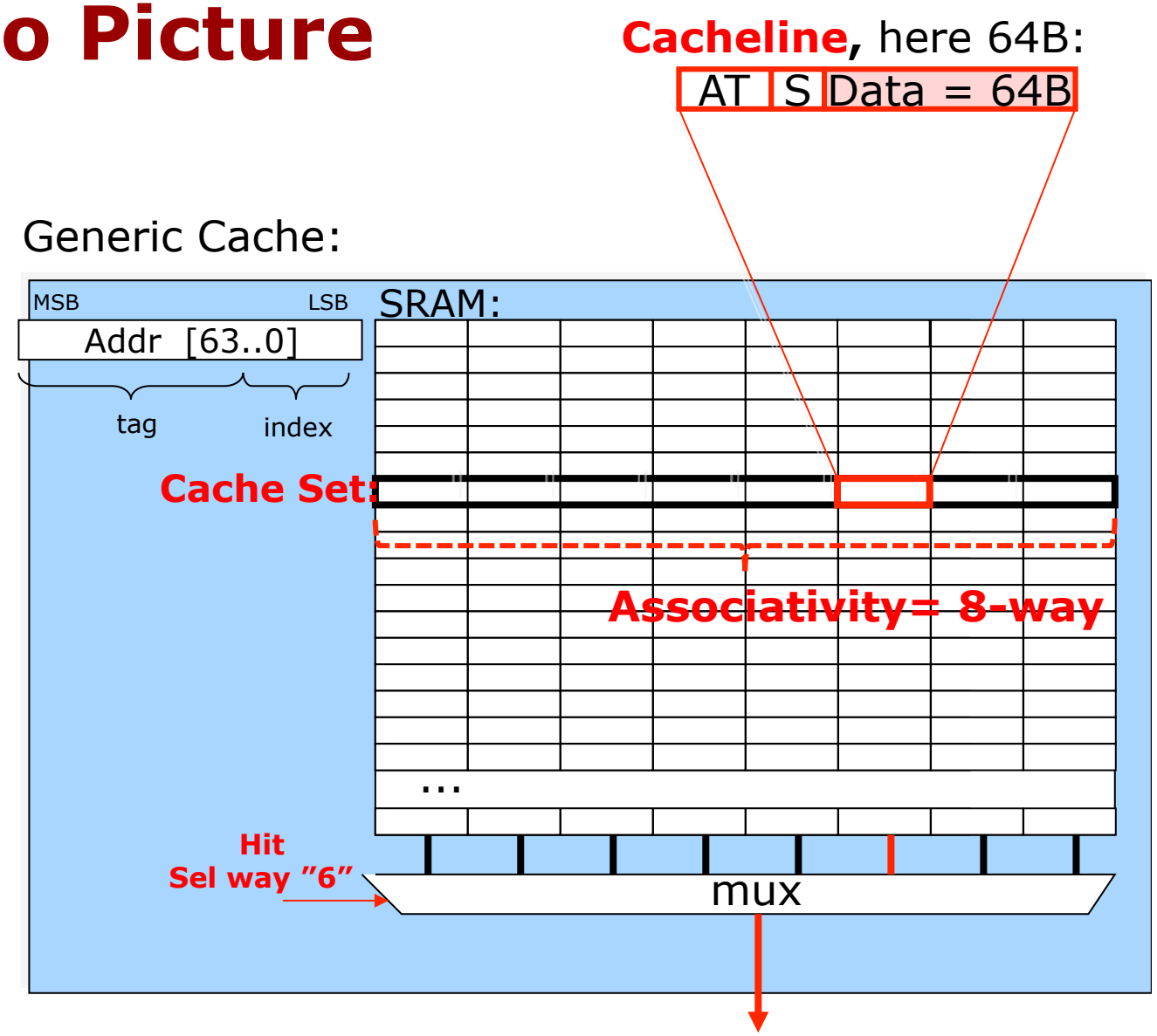
Typical access pattern:

(inner loop stepping through an array)

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



Cache Lingo Picture



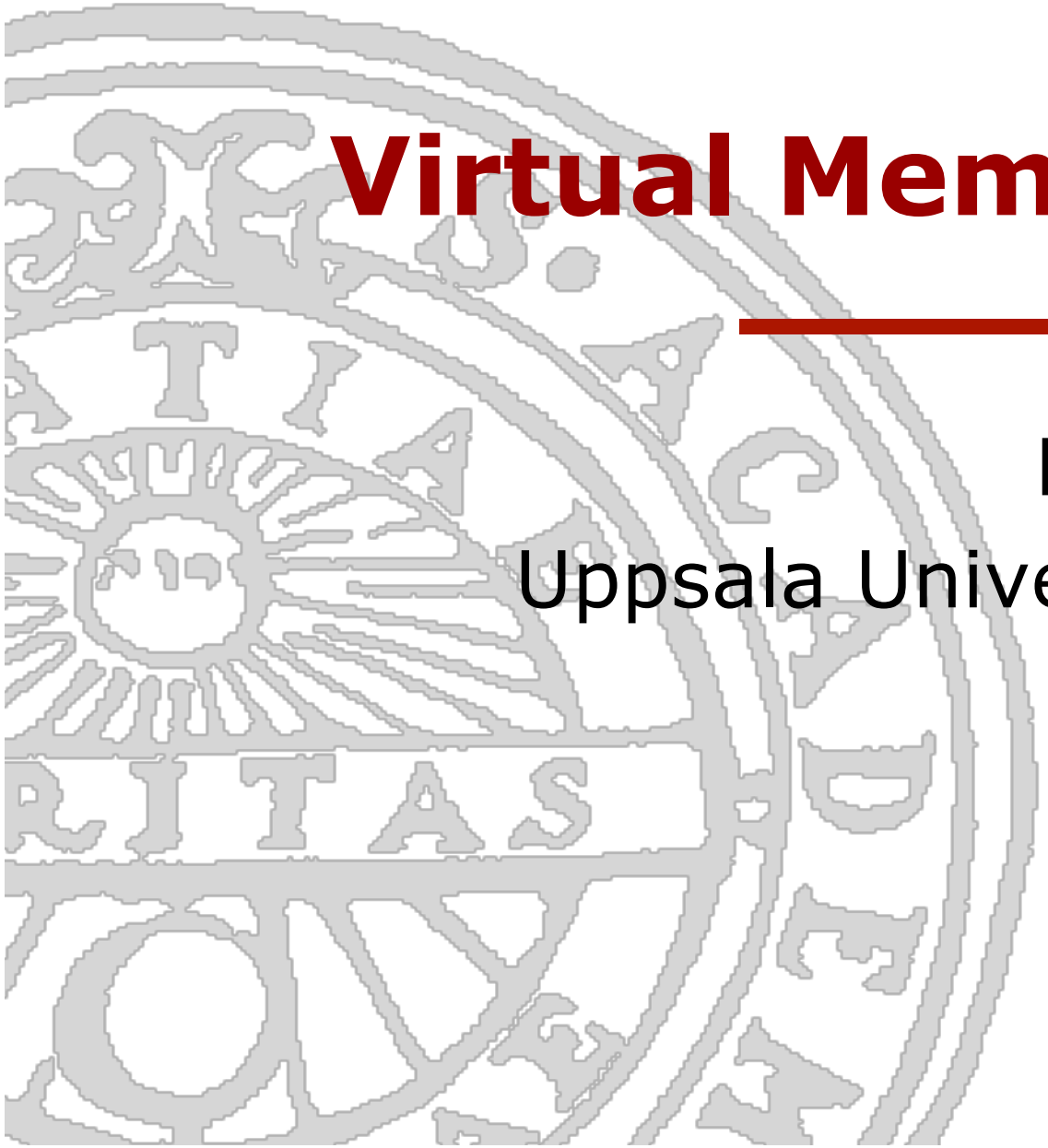
Why do you miss in a cache

- 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/Coherence (imposed by coherence)
  - False sharing (side-effect from large cache blocks)

Take-away message: Caches

- Caches are fast but small
- Cache data travels in cache-line chunks (e.g., ~64bytes)
- Least significant 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

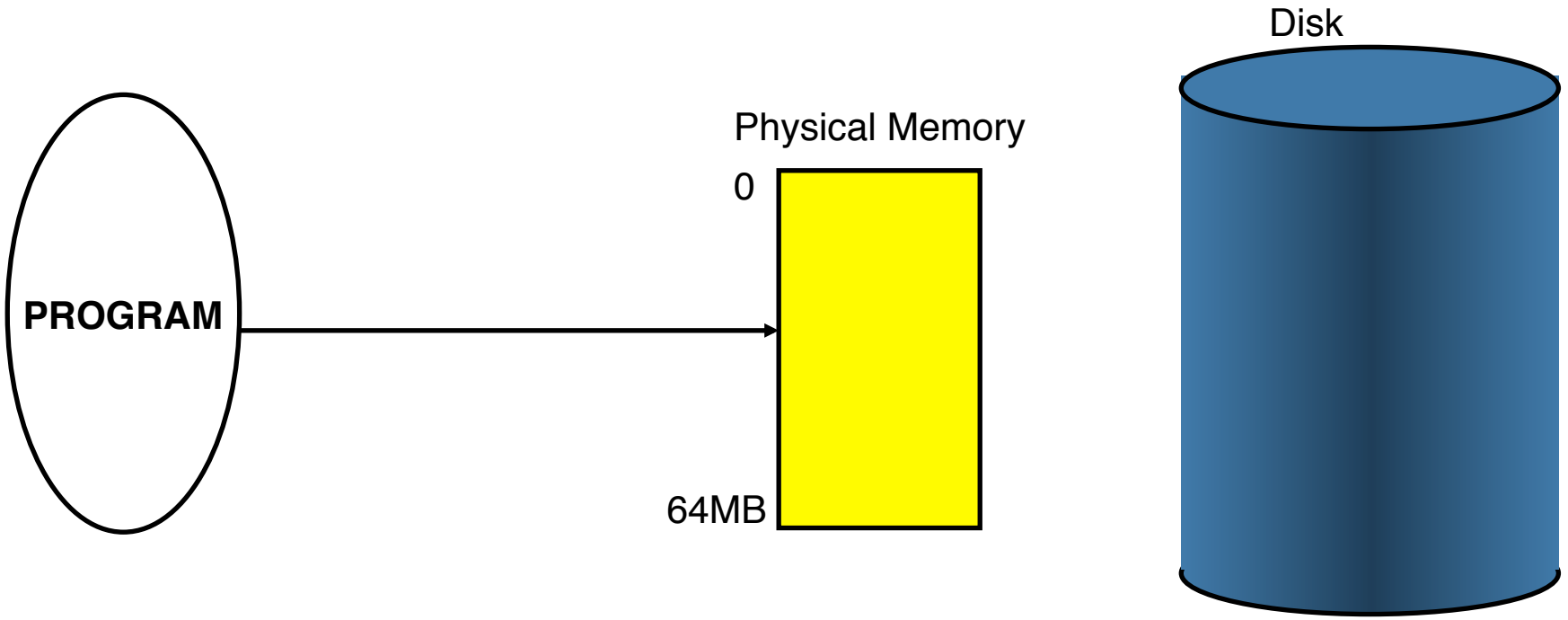
- Caches are most important target for optimizations



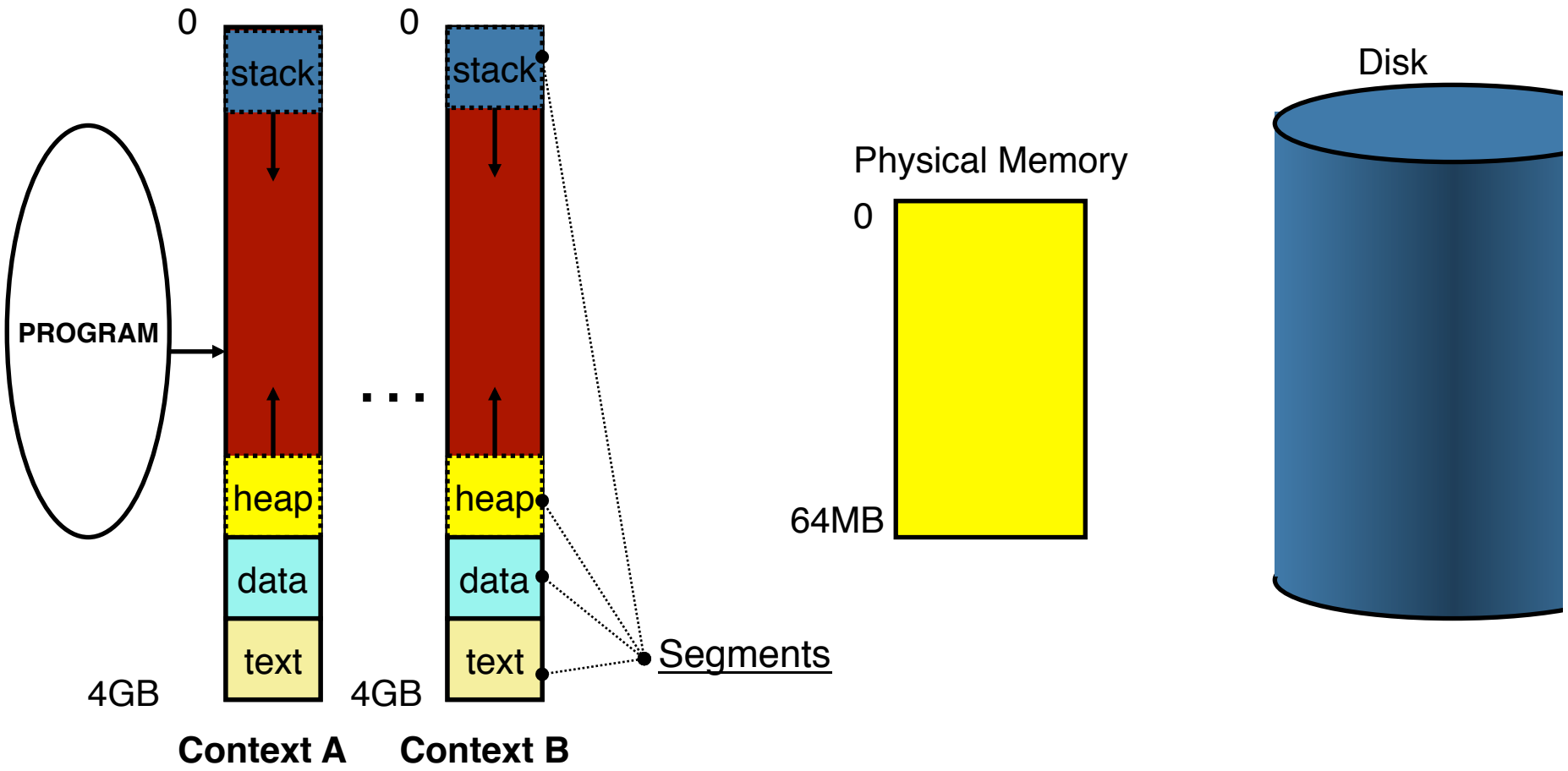
Virtual Memory System

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

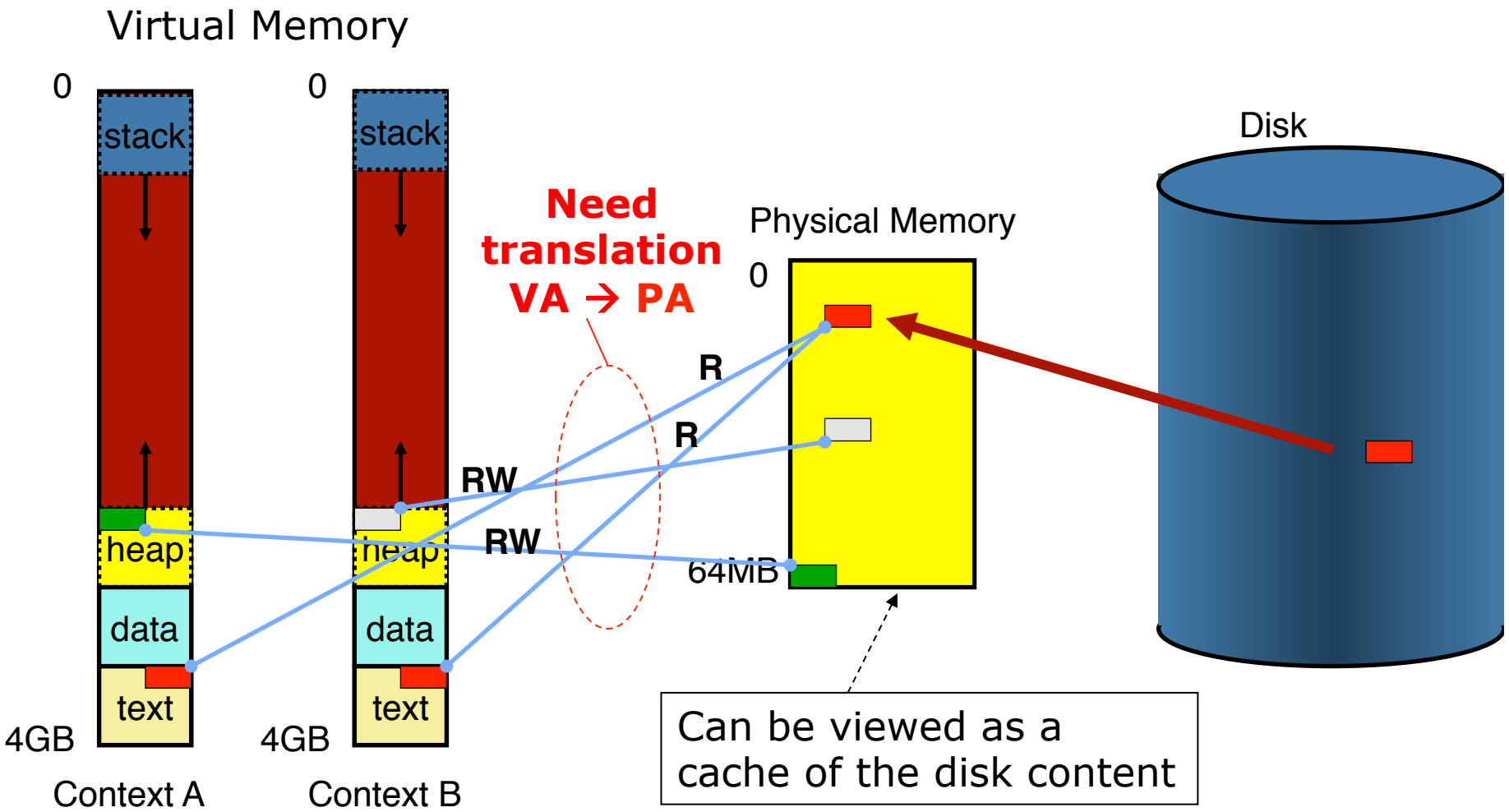
Physical Memory



Virtual and Physical Memory



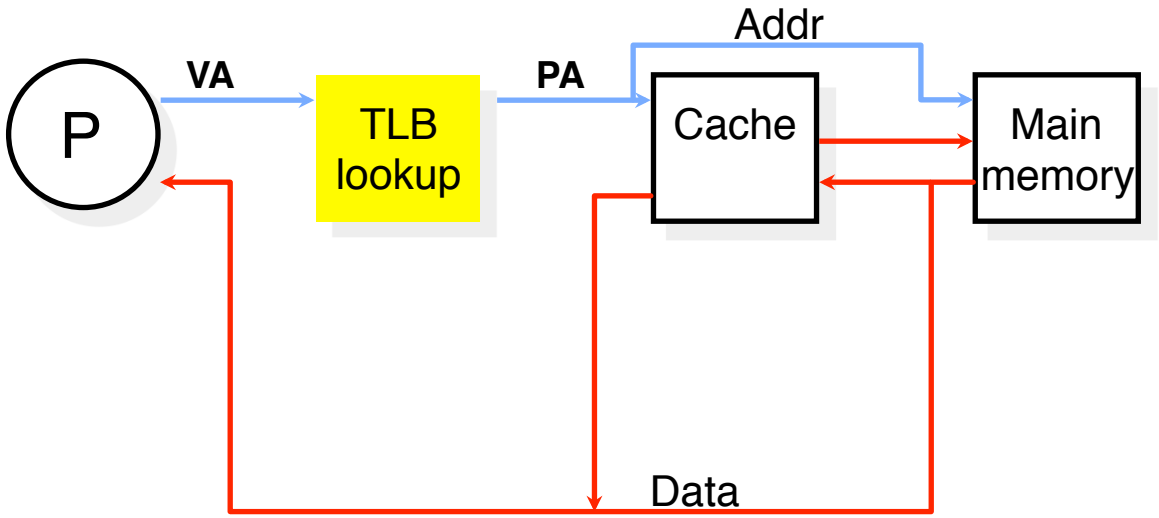
Translation & Protection



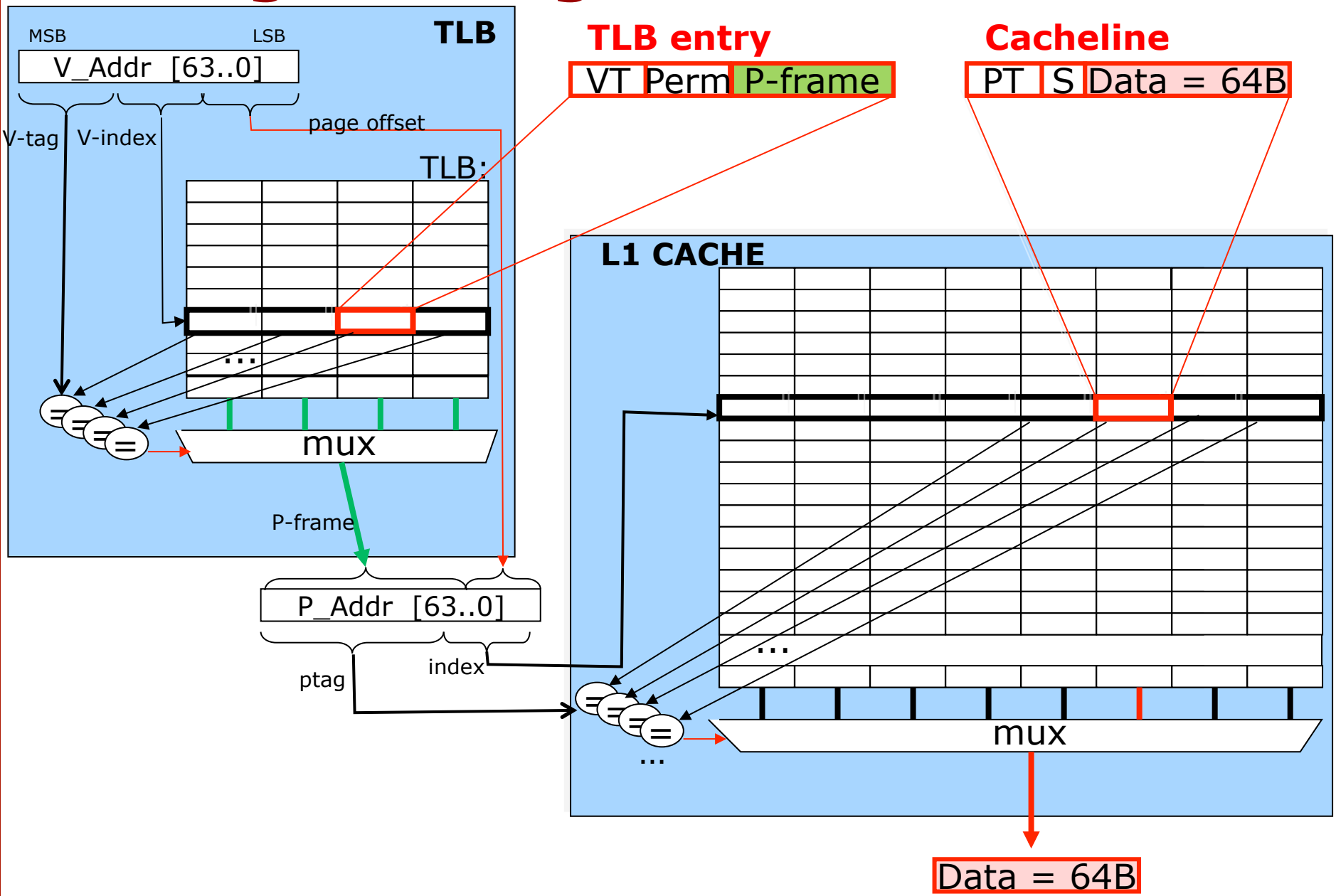
Fast address translation

How to quickly and cheaply find the right physical page in the [physical memory "cache"?

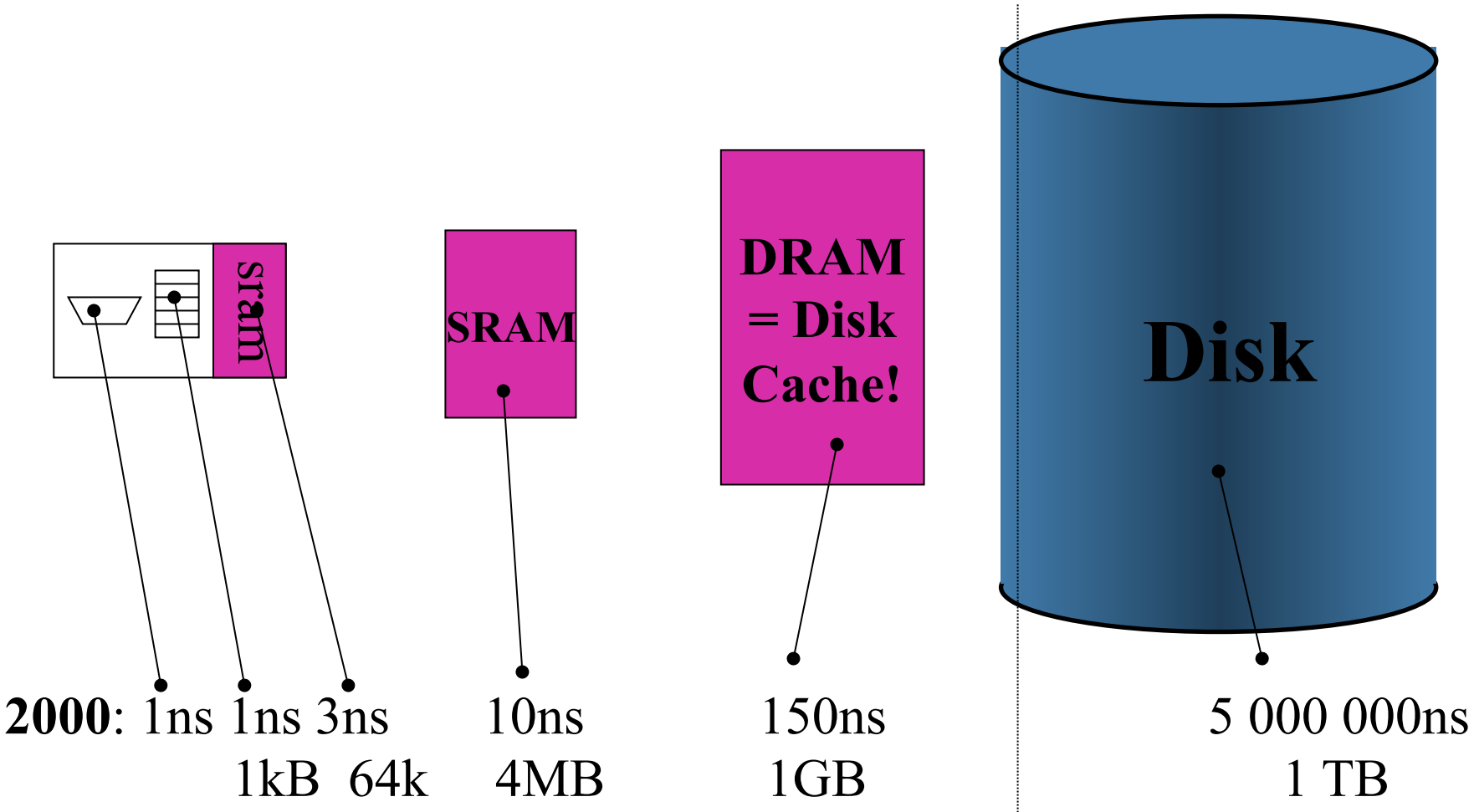
- Store the most recently used address translations in a **SMALL cache**—***Translation Look-aside Buffer*** (TLB)
  ==> The caches rears their ugly faces again!



Putting it all together:



Memory/storage



VM dictionary

Virtual Memory System	The ~~c~~ache“ languge
Virtual address	~Cache address
Physical address	~Cache location, ~~w~~ay info“
Page	~Huge cache block
Page fault	~Extremely painfull $miss
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 predictor
- HW Prefetch algorithms
- Directory cache (coherence)
- …

How are we doing?

- Create and explore locality:
  - ✓ a) Spatial locality
  - ✓ b) Temporal 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