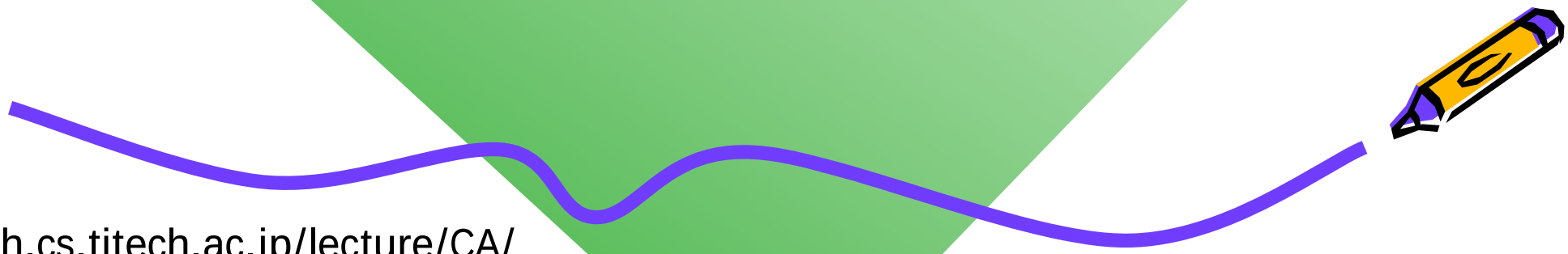
2018年度（平成30年度）版

Ver. 2018-10-10a

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

5. キャッシュ：セットアソシアティブ方式
Caches: Set-Associative

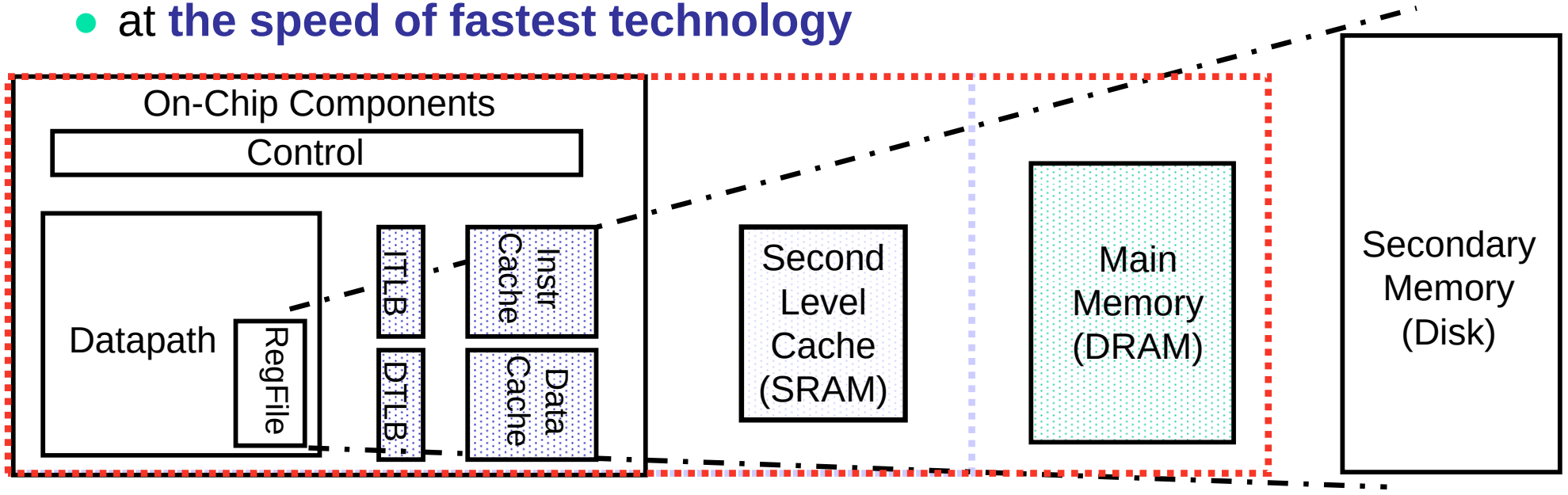


www.arch.cs.titech.ac.jp/lecture/CA/
Room No.W321
Tue 13:20-16:20, Fri 13:20-14:50

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

A Typical Memory Hierarchy

- By taking advantage of **the principle of locality**（局所性）
  - Present **much memory** in **the cheapest technology**
  - at **the speed of fastest technology**



Speed (%cycles):	½'s	1's	10's	100's	1,000's
Size (bytes):	100's	K's	10K's	M's	G's to T's
Cost:	highest				lowest

TLB: Translation Lookaside Buffer

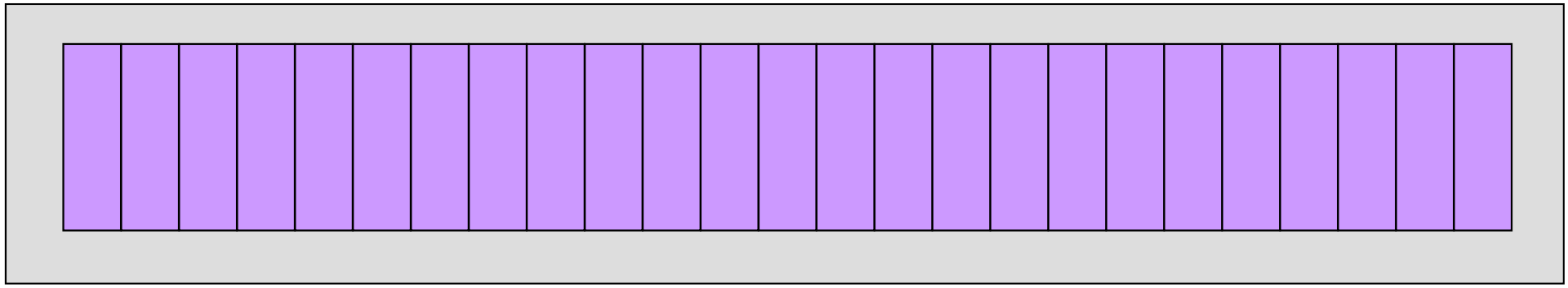
Sources of Cache Misses

- Compulsory (初期参照ミス, cold start or process migration, first reference):
  - First access to a block, "cold" fact of life, not a whole lot you can do about it
  - If you are going to run "millions" of instruction, compulsory misses are insignificant
- Conflict (競合性ミス, collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase **associativity**
- Capacity (容量性ミス):
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size

Reducing Cache Miss Rates, **Associativity**

- **Allow more flexible block placement**
  - In a **direct mapped cache** a memory block maps to exactly **one cache block**
  - At the other extreme, could allow a memory block to be mapped to any cache block – **fully associative cache**
  - A compromise is to divide the cache into **sets** each of which consists of **n "ways"** (**n-way set associative**).
    A memory block maps to a unique set and can be placed in any way of that set (so there are **n choices**)

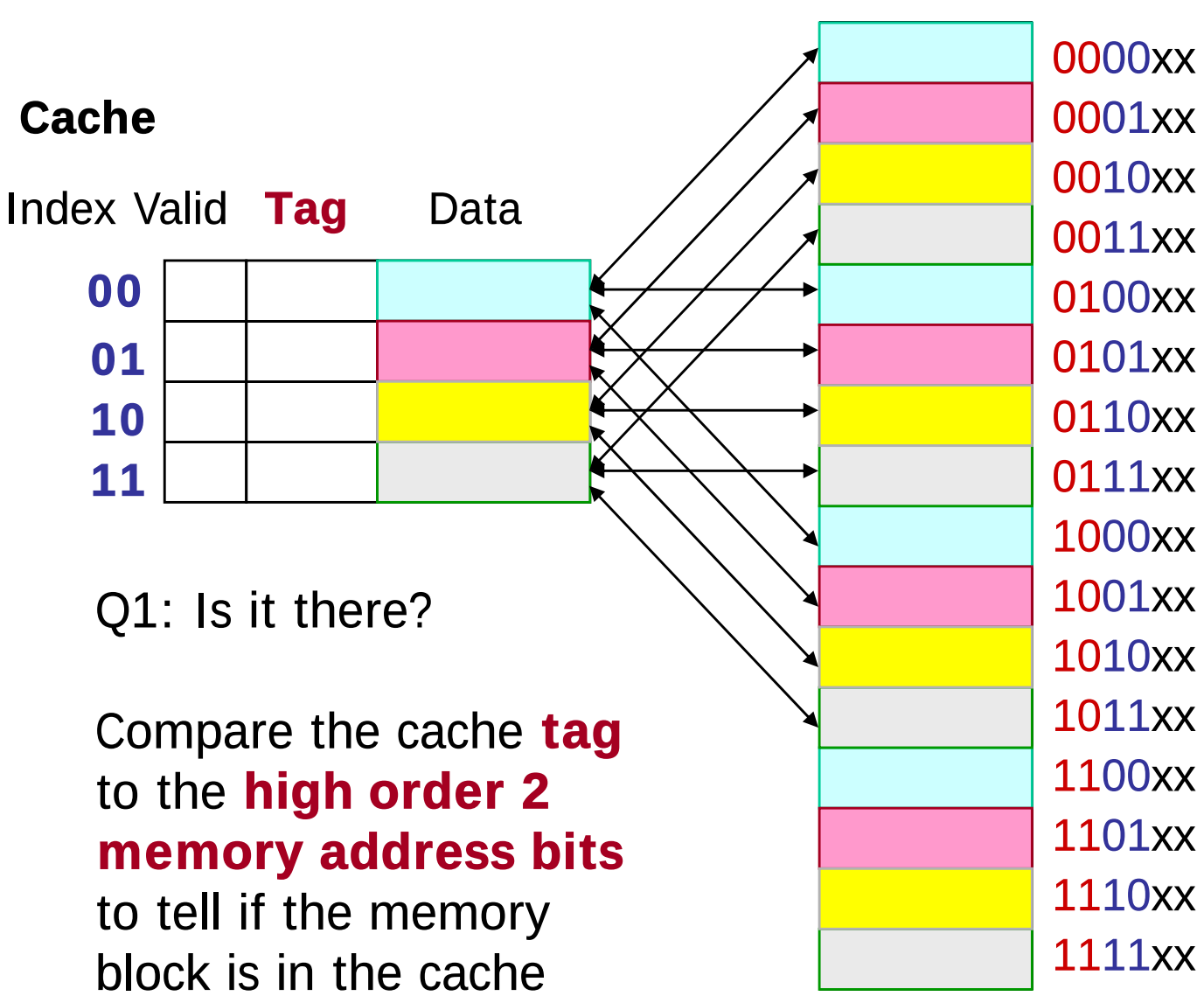
Cache Associativity



本棚

机

Caching: Direct mapped (First Example)



Q1: Is it there?

Compare the cache **tag** to the **high order 2 memory address bits** to tell if the memory block is in the cache

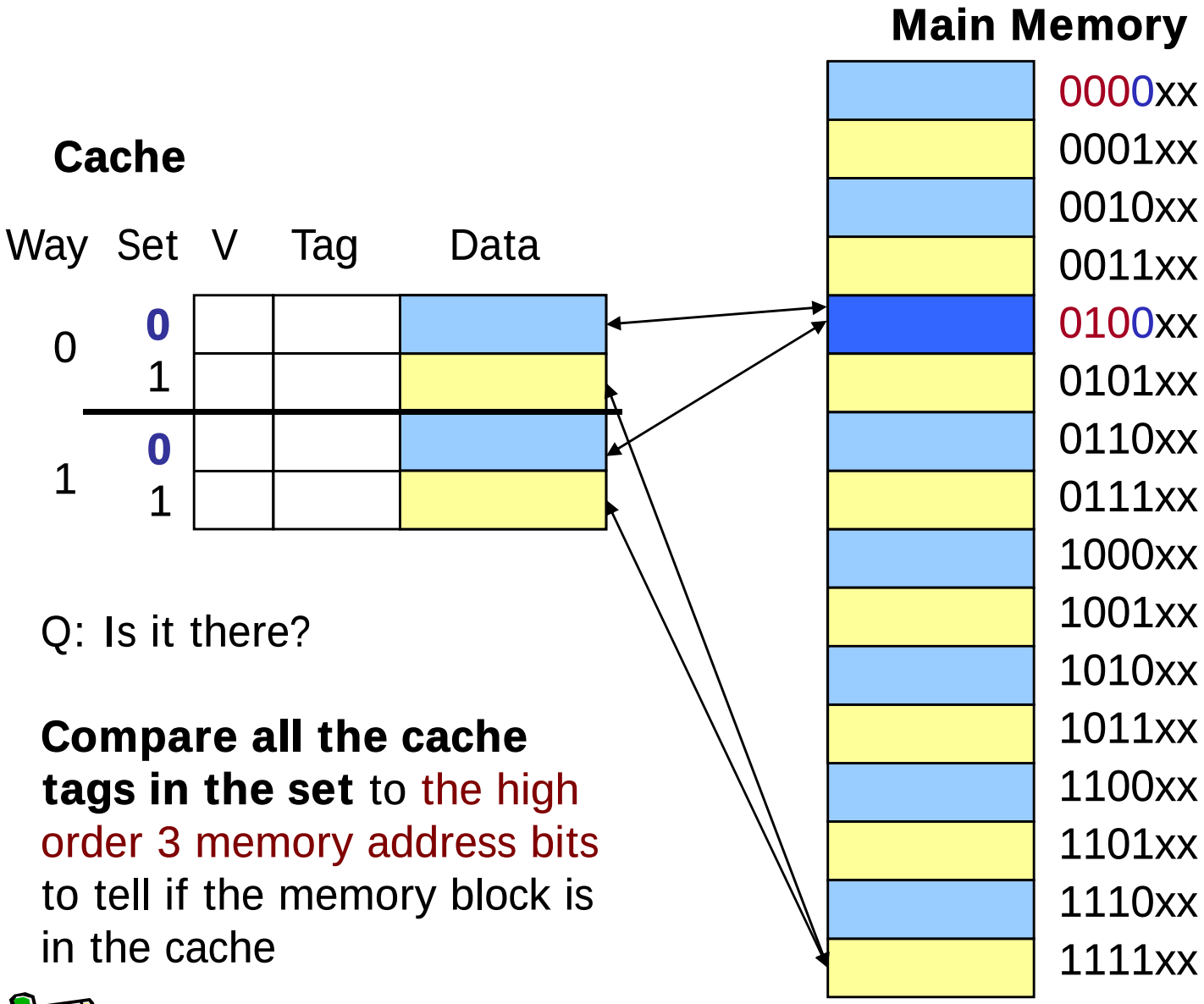
Two low order bits define the byte in the word (32-b words)

Q2: How do we find it?

Use **next 2 low order memory address bits** – the **index** – to determine which cache block

(block address) modulo (# of blocks in the cache)

Set Associative Cache Example



Q: Is it there?

Compare all the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Two low order bits define the byte in the word (32-b words)
One word blocks

Q: How do we find it?

Use next 1 low order memory address bit to determine which cache set

Another Reference String Mapping

- Consider the main memory word reference string
  - 0 4 0 4 0 4 0 4

0 miss

00	Mem(0)

4 miss

01 → 00	Mem(0) → 4

0 miss

00 → 01	Mem(4) → 0

4 miss

01 → 00	Mem(0) → 4

0 miss

00 → 01	Mem(4) → 0

4 miss

01 → 00	Mem(0) → 4

0 miss

00 → 01	Mem(4) → 0

4 miss

01 → 00	Mem(0) → 4

  - 8 requests, 8 misses
    - Ping pong effect due to **conflict** misses - two memory locations that map into the same cache block

Another Reference String Mapping

- Consider the main memory word reference string

0 4 0 4 0 4 0 4

Start with an empty cache –
all blocks initially marked as not valid

0 miss

000	Mem(0)

4 miss

000	Mem(0)
010	Mem(4)

0 hit

000	Mem(0)
010	Mem(4)

4 hit

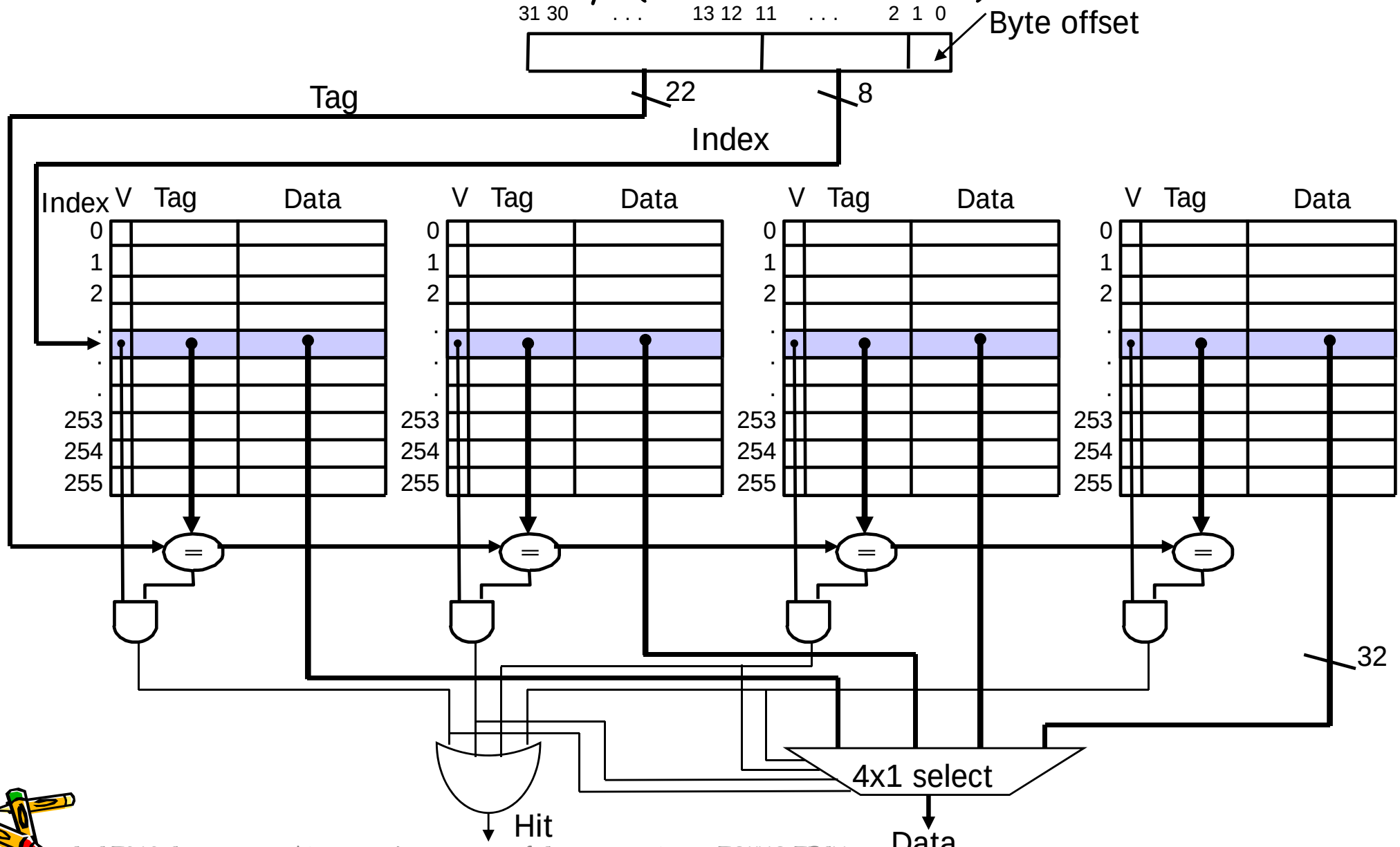
000	Mem(0)
010	Mem(4)

  - 8 requests, 2 misses

- Solves the **ping pong effect** in a direct mapped cache due to conflict misses

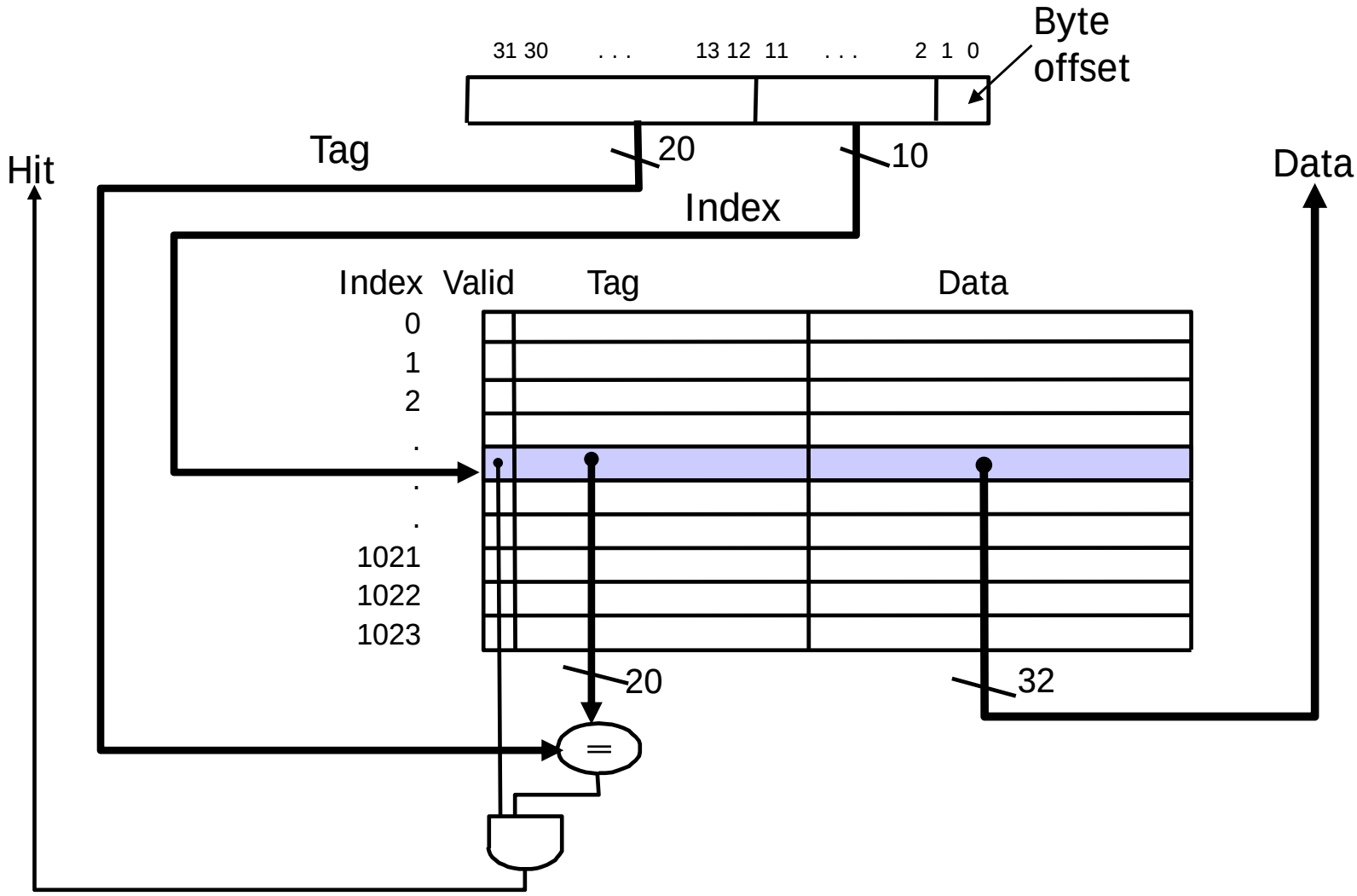
Four-Way Set Associative Cache

- 2^8 = **256 sets** each with four ways (each with one block)



MIPS Direct Mapped Cache Example

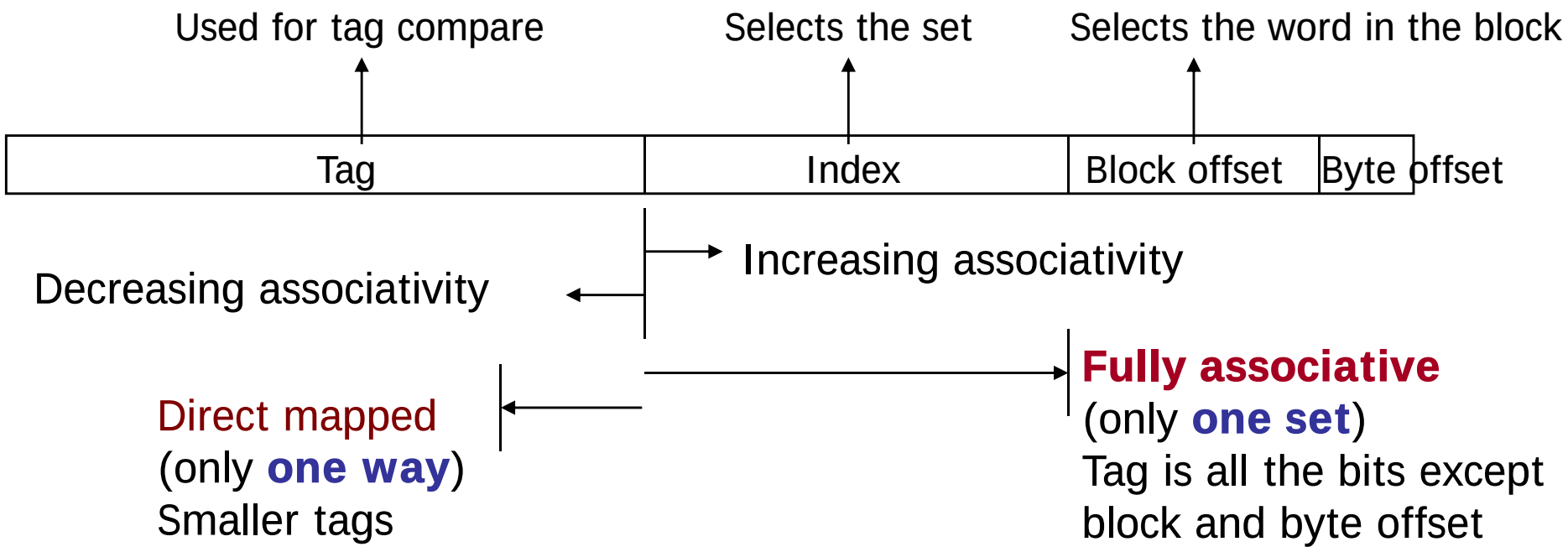
- One word/block, cache size = 1K words



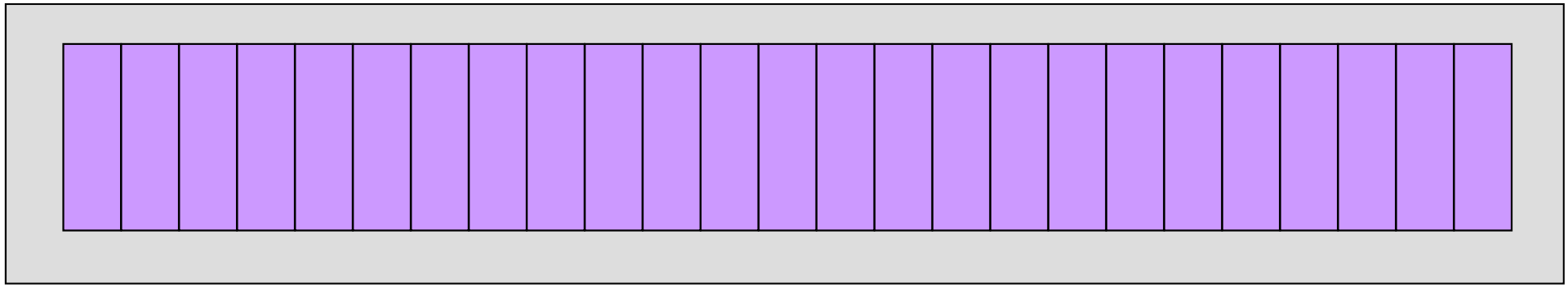
What kind of locality are we taking advantage of?

Range of Set Associative Caches

- For a fixed size cache



Cache Associativity



本棚

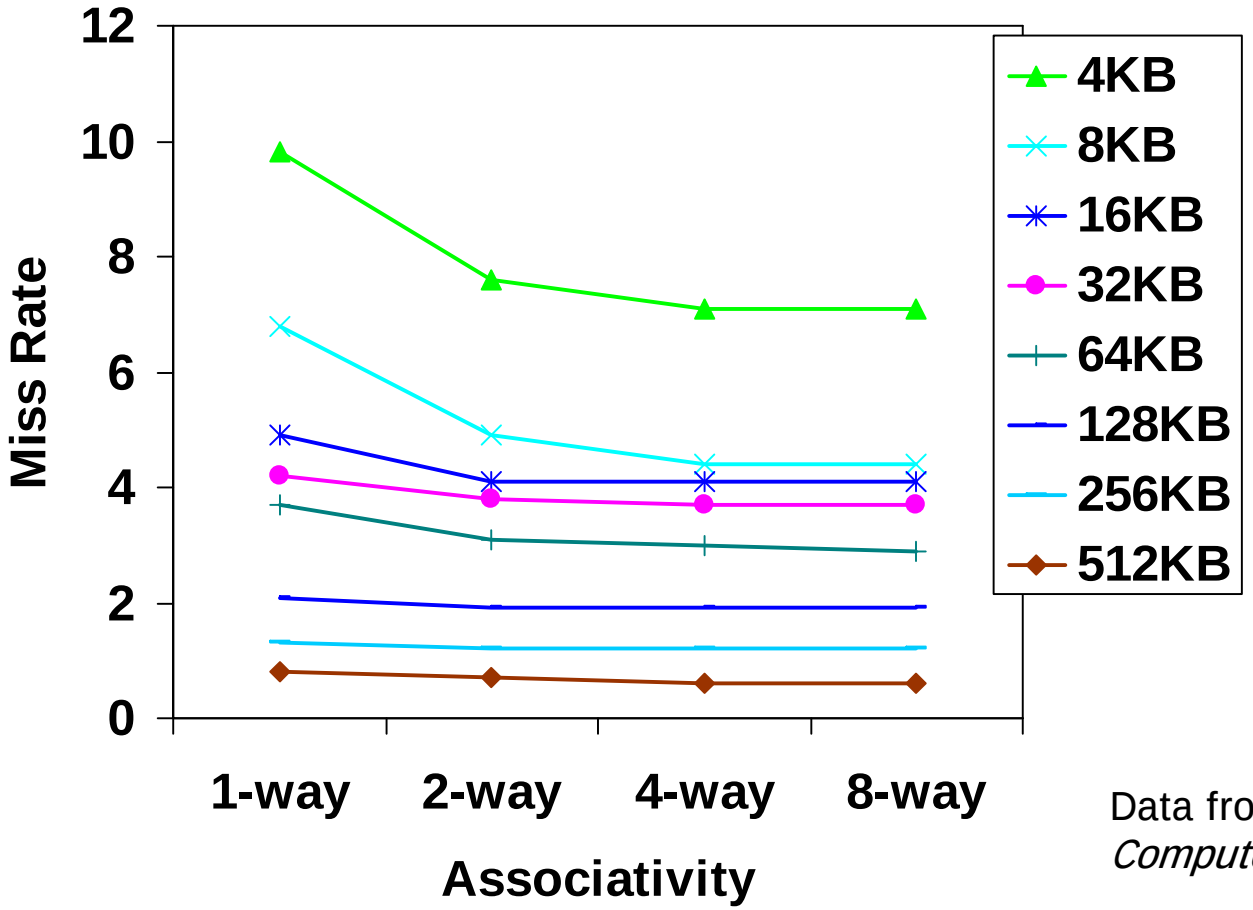
机

Costs of Set Associative Caches

- **N-way set associative** cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available **after** set selection and Hit/Miss decision.
- **When a miss occurs**,
  which way's block do we pick **for replacement** ?
  - **Least Recently Used (LRU):**
    the block replaced is the one that has been unused for the longest time
    - Must have hardware to keep track of when each way's block was used
    - For **2-way set associative**, takes **one bit per set** →
      set the bit when a block is referenced
      (and reset the other way's bit)
  - **Random**

Benefits of Set Associative Caches

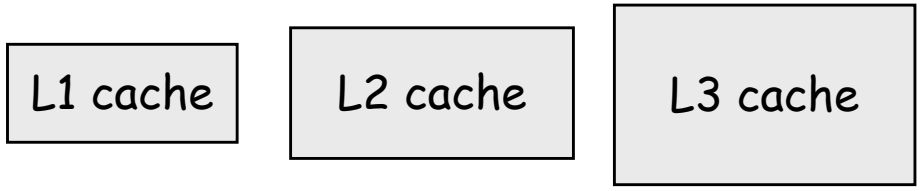
- The choice of direct mapped or set associative depends on **the cost of a miss** versus **the cost of implementation**



Data from Hennessy & Patterson, *Computer Architecture*, 2003

- Largest gains are in going from **direct mapped** to **2-way**

Reducing Cache Miss Rates by **multiple levels**



- Enough room on the die for **bigger L1 caches** *or* for a **second level of caches** – normally a **unified** L2 cache (i.e., it holds both instructions and data) and in some cases even a **unified L3 cache**
- For our example,
  - CPI_{ideal} of 2,
  - 100 cycle miss penalty (to main memory),
  - 36% load/stores,
  - a 2% (4%) L1I$ (D$) miss rate,
  - **add a UL2$ that has a 25 cycle miss penalty and a 0.5% miss rate**

$$CPI_{stalls} = 2 + .02 \times \mathbf{25} + .36 \times .04 \times \mathbf{25} + \mathbf{.005 \times 100} + \mathbf{.36 \times .005 \times 100} = 3.54$$

(as compared to **5.44** with no L2$)

Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are very different
  - Primary cache should focus on **minimizing hit time** in support of a shorter clock cycle
  - Secondary cache should focus on **reducing miss rate** to reduce the penalty of long main memory access times

- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate
- For the L2 cache, hit time is less important than miss rate
  - The L2$ hit time determines L1$'s miss penalty

Key Cache Design Parameters

	L1 typical	L2 typical
Total size (blocks)	250 to 2000	4000 to 250,000
Total size (KB)	16 to 64	500 to 8000
Block size (B)	32 to 64	32 to 128
Miss penalty (clocks)	10 to 25	100 to 1000
Miss rates	2% to 5%	0.1% to 2%

Two Machines' Cache Parameters

	Intel P4	AMD Opteron
L1 organization	Split I$ and D$	Split I$ and D$
L1 cache size	8KB for D$, 96KB for trace cache (~I$)	64KB for each of I$ and D$
L1 block size	64 bytes	64 bytes
L1 associativity	4-way set assoc.	2-way set assoc.
L1 replacement	~ LRU	LRU
L1 write policy	write-through	write-back
L2 organization	Unified	Unified
L2 cache size	512KB	1024KB (1MB)
L2 block size	128 bytes	64 bytes
L2 associativity	8-way set assoc.	16-way set assoc.
L2 replacement	~LRU	~LRU
L2 write policy	write-back	write-back

OPT: Optimal Replacement Policy

The Optimal Replacement Policy

1. Replacement Candidates : On a miss any replacement policy could either choose to replace any of the lines in the cache or choose not to place the miss causing line in the cache at all.
2. Self Replacement : The latter choice is referred to as a self-replacement or a cache bypass

Optimal Replacement Policy

On a miss replace the candidate to which an access is least imminent [Belady1966,Mattson1970,McFarling-thesis]

3. Lookahead Window : Window of accesses between miss causing access and the access to the least imminent replacement candidate. Single pass simulation of OPT make use of lookahead windows to identify replacement candidates and modify current cache state [Sugumar-SIGMETRICS1993]

OPT: あまり切迫していないものを置き換える.

Example Behavior of Optimal Replacement Policy

Understanding OPT

Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		0		1		2	3	4				
OPT order for A_6				0	1	2	3				4	

- Consider 4 way associative cache with one set initially containing lines (A_1, A_2, A_3, A_4), consider the access stream shown in table
- Access A_5 misses, replacement decision proceeds as follows
  1. Identify replacement candidates : $(A_1, A_2, A_3, A_4, A_5)$
  2. Lookahead and gather imminence order : shown in table, lookahead window circled
  3. Make replacement decision : A_5 replaces A_2
- A_6 self-replaces, lookahead window and imminence order in table

The Cache Design Space

- **Several interacting dimensions**
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation
- **The optimal choice is a compromise**
  - depends on access characteristics
    - workload
    - I-cache, D-cache
  - depends on technology / cost
- **Simplicity** often wins