Summary of previous lectures

- introduction to memory hierarchy
  - locality principles
  - levels, blocks, hit, miss, miss penalty
  - direct-mapped cache, handling misses
- cache features and performance
  - review direct-mapped single-word and multi-word cache
  - cache performance, read/write stall cycles
  - multi-level caches

Associative caches

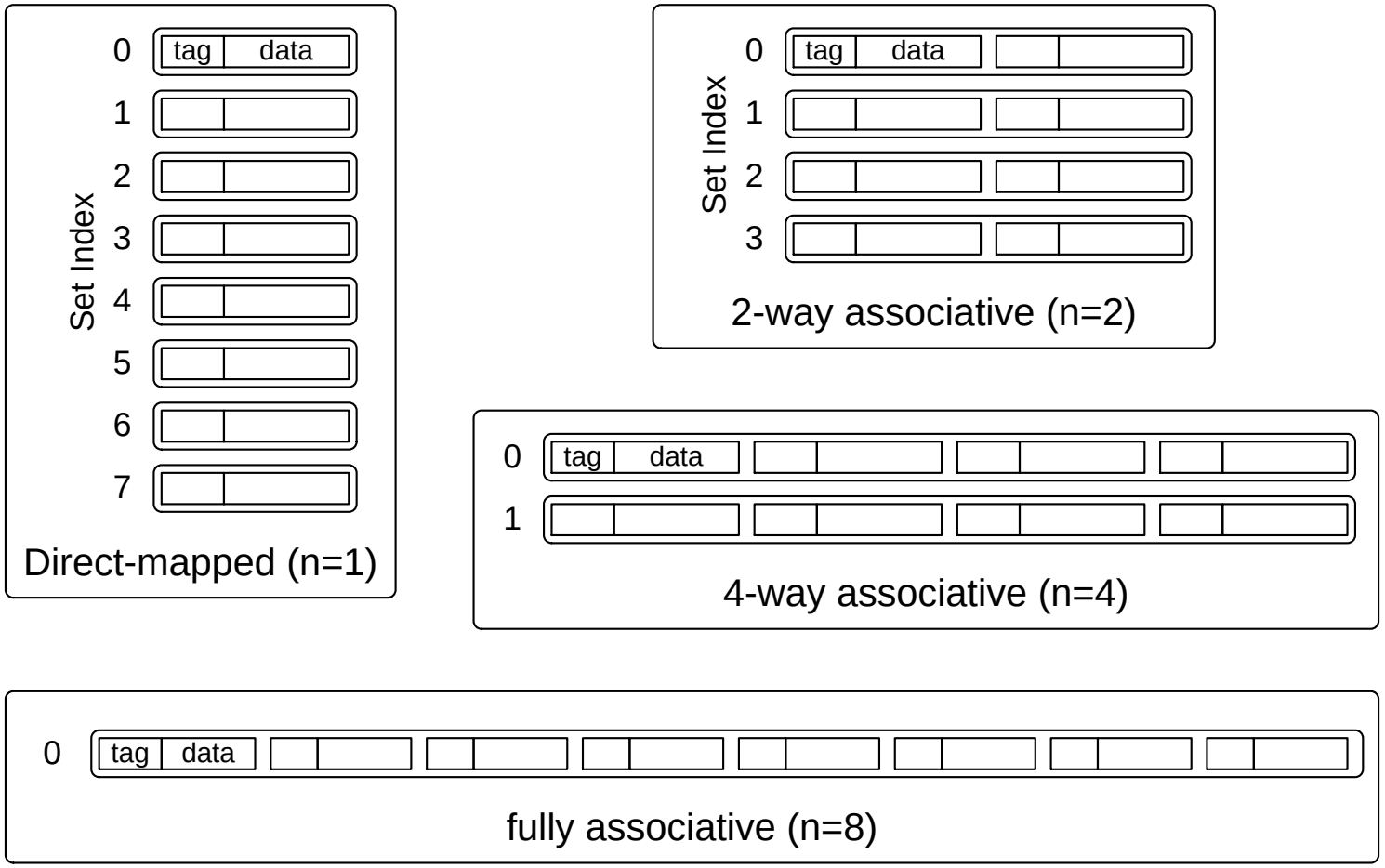
(3rd Ed: p.496-504, 4th Ed: 479-487)

- flexible block placement schemes
- overview of set associative caches
- block replacement strategies
- associative cache implementation
- size and performance

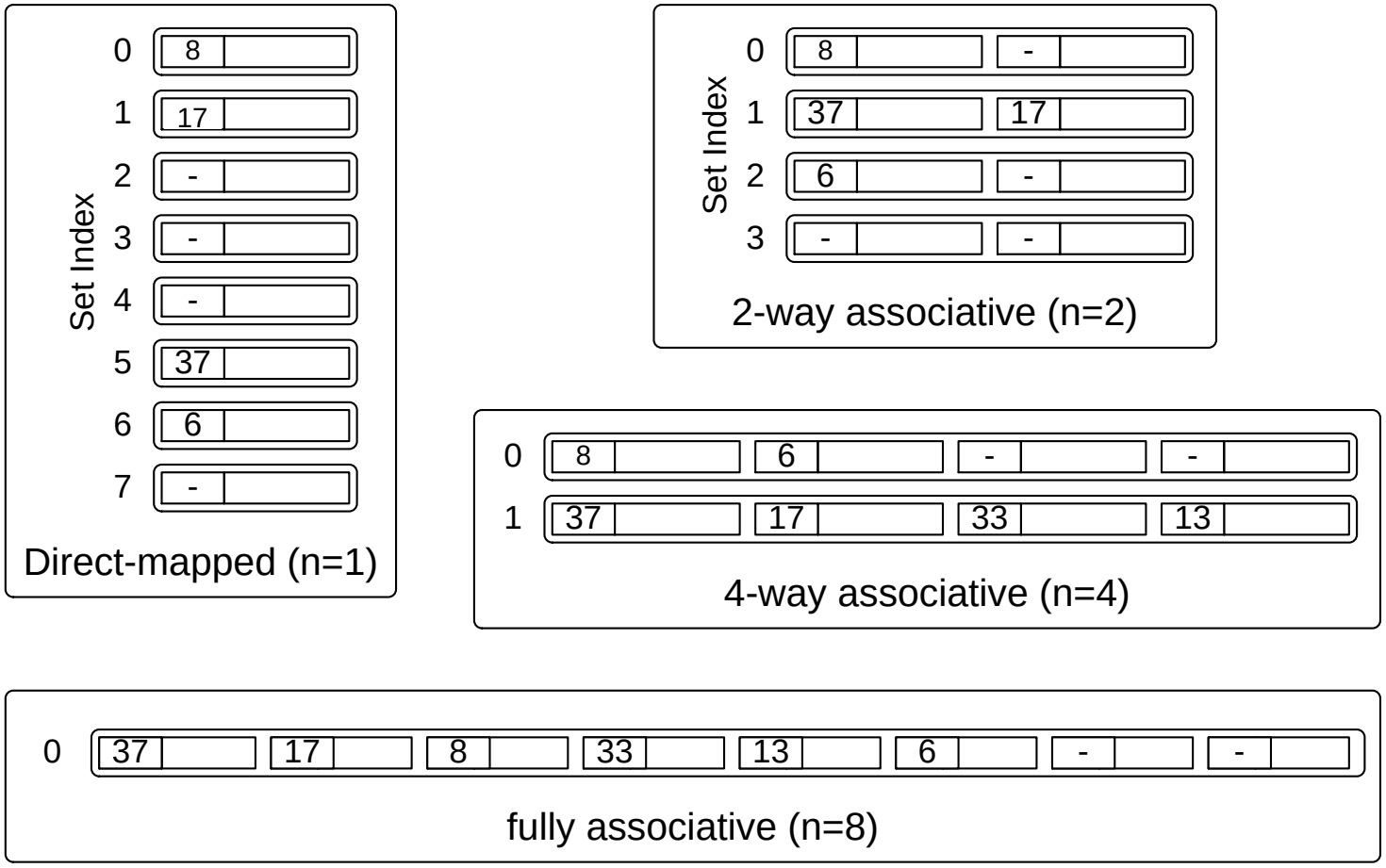
Placement schemes

- direct mapped: each address maps to one cache block
  - simple, fast access, high miss rate
- fully associative: each address maps to all *c* blocks
  - low miss rate, costly, slow (need to search everywhere)
- *n*-way set associative: each address maps to *n* blocks
  - each set contains *n* blocks; number of sets `s = (c div n)`
  - `set_index = block_address mod s;`
- fixed number of blocks, increased associativity leads to
  - more blocks per set : increased *n*
  - fewer sets per cache : decreased *s*
  - more flexible, but more search overhead : *n* compares per lookup

Possible organisations of 8-block cache

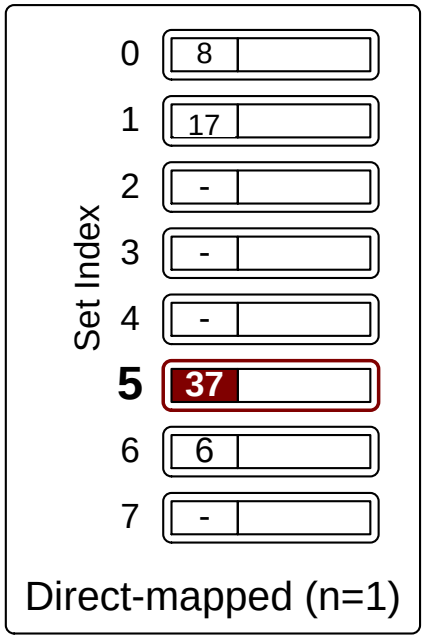


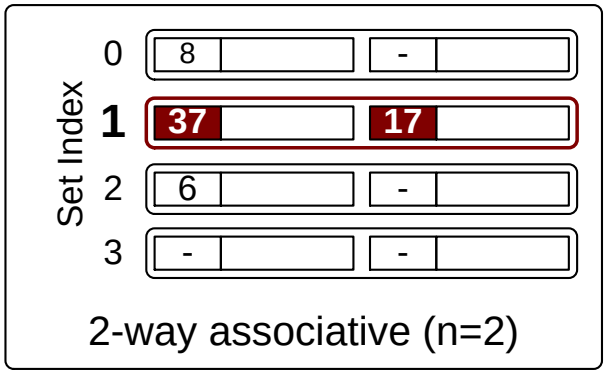
Placement flexibility vs need for search

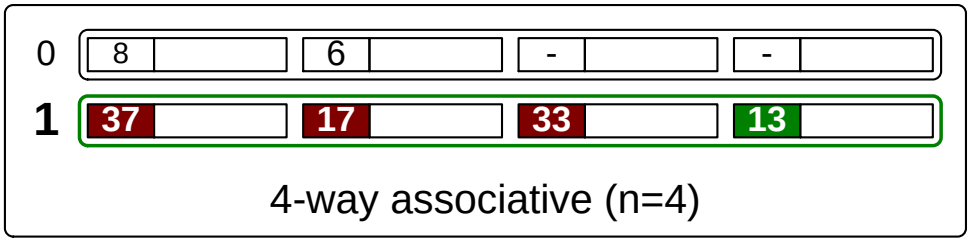


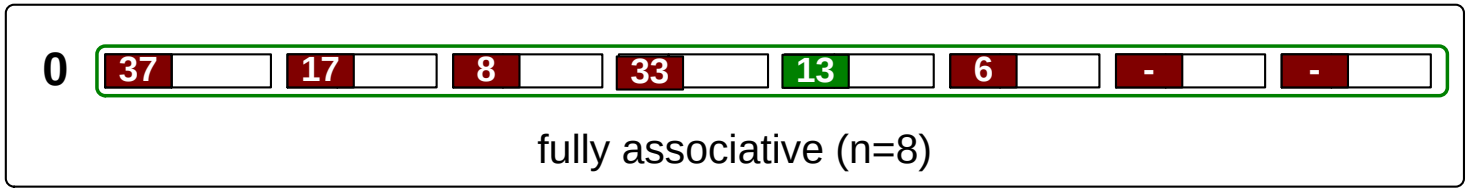
Previously accessed *block* addresses: 6, 13, 33, 8, 17, 37

Placement flexibility vs need for search









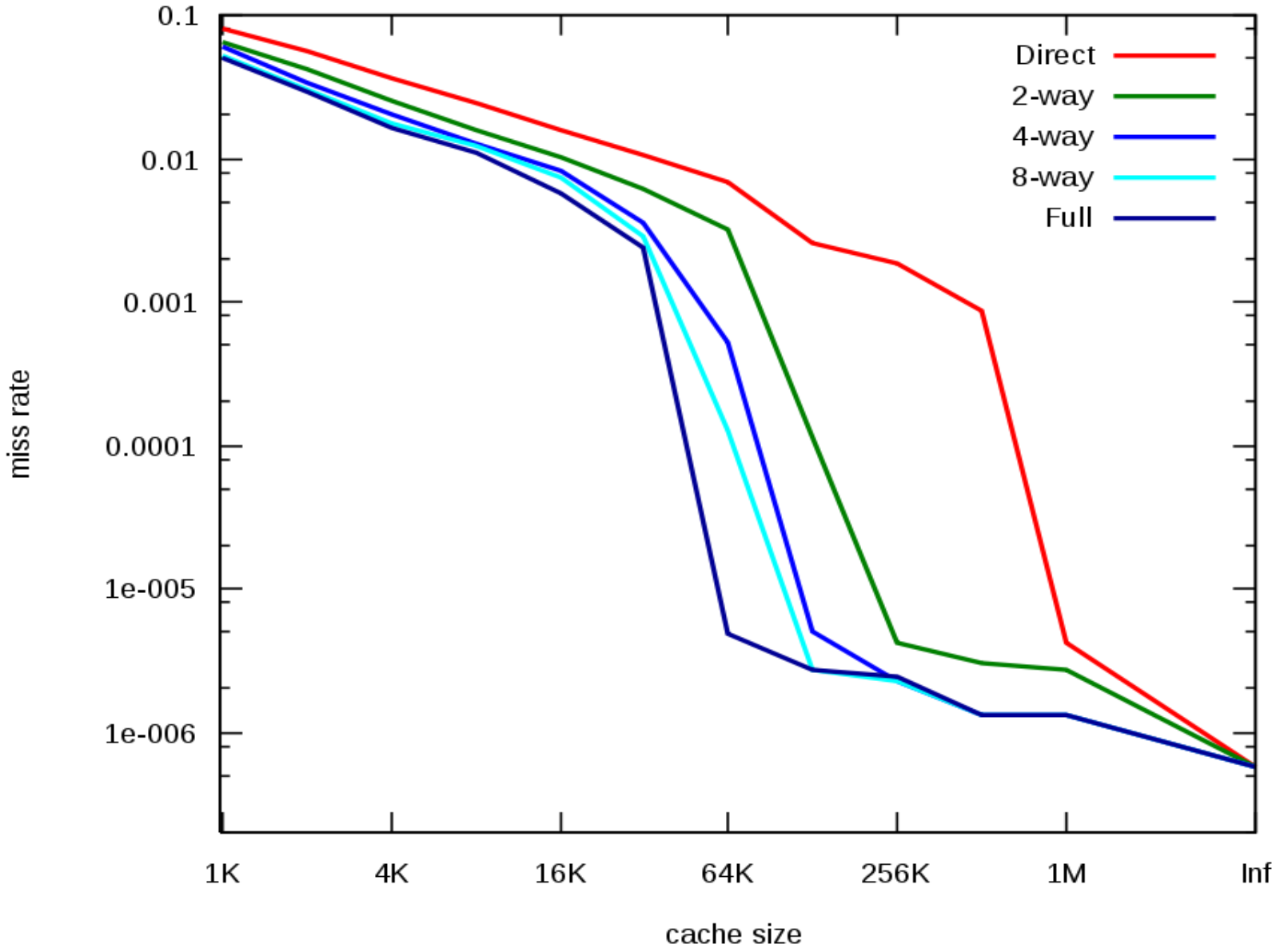
Previously accessed *block* addresses: 6, 13, 33, 8, 17, 37

Now look for block address 13...

Impact of associativity on miss rate

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

(from 10 SPEC2000 programs)


0.1
0.01
0.001
0.0001
1e-005
1e-006
miss rate
1K
4K
16K
64K
256K
1M
Inf
cache size
Direct
2-way
4-way
8-way
Full

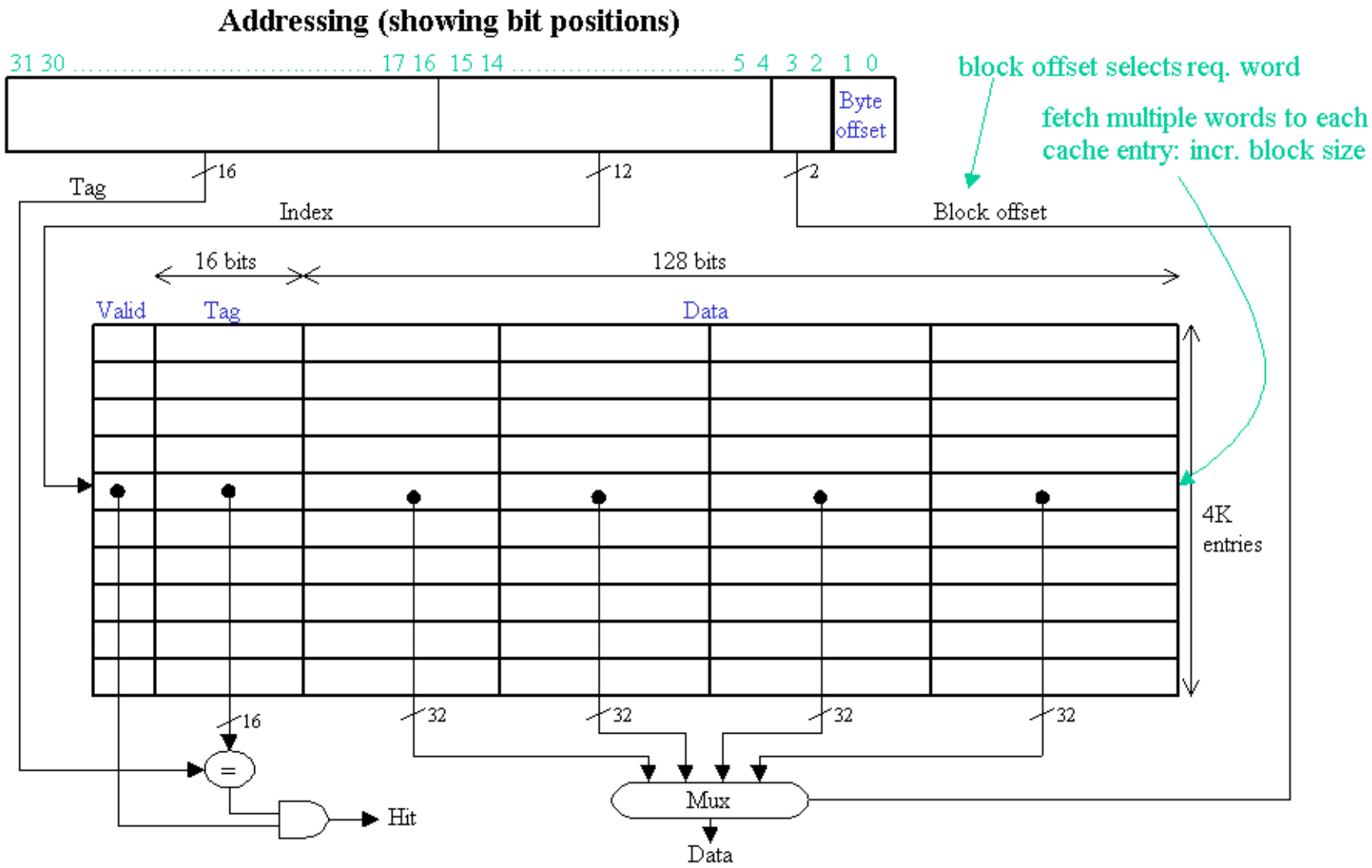
Real-world caches

Feature	Intel P4	AMD Opteron
L1 instruction	96KB	64KB
L1 data	8KB	64KB
L1 associativity	4-way set assoc.	2-way set assoc.
L2	512KB	1024KB
L2 associativity	8-way set assoc.	16-way set assoc.

n-way associative: lookup algorithm

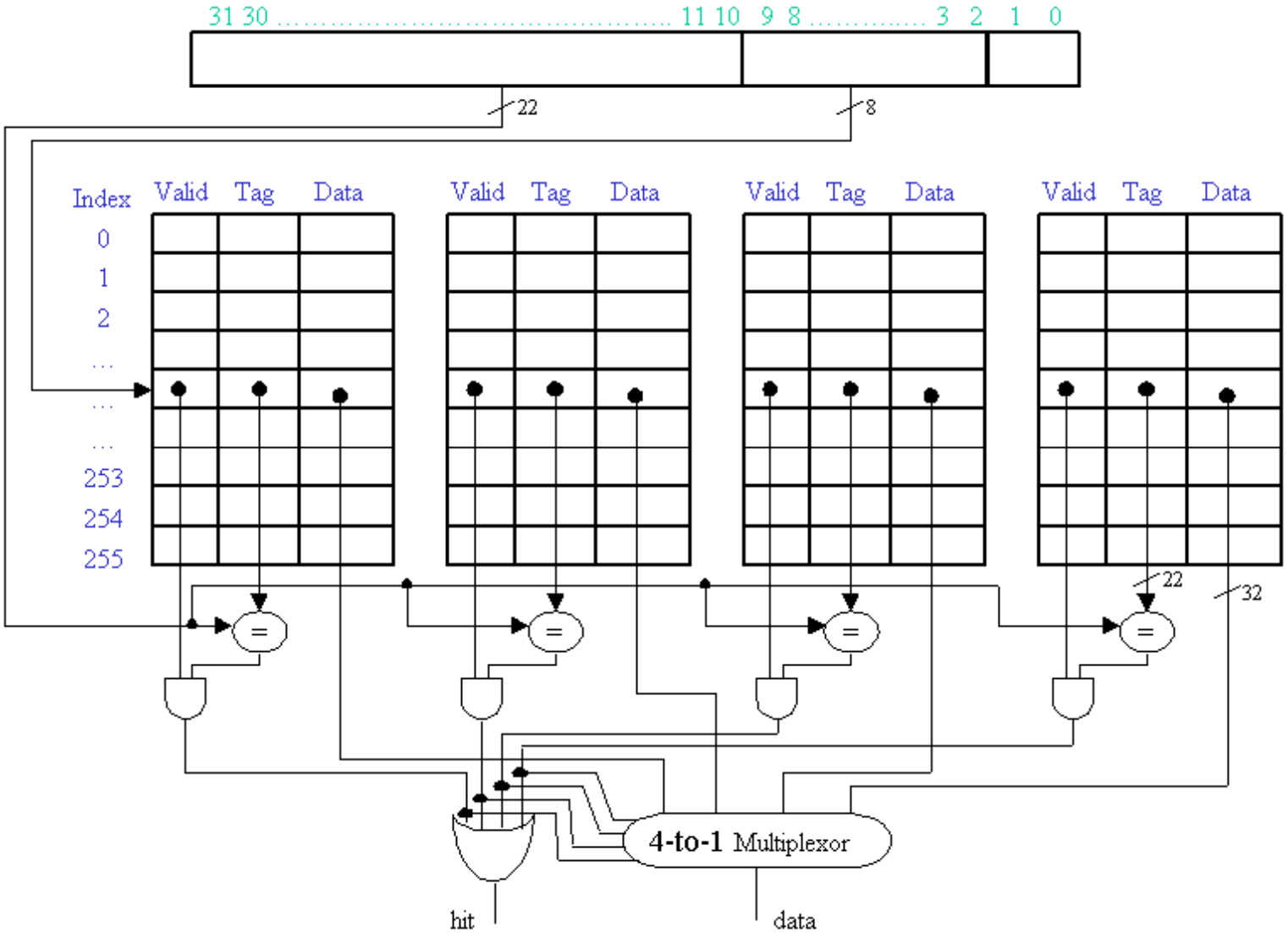
```
type block_t is {
  tag_t tag; bool valid; word_t data[k];
};
type set_t   is block_t[n];
type cache_t is set_t[s];

cache_t cache;

1.   uint block_address = (word_address div k);
2.   uint block_offset  = (word_address mod k);
3.   uint set_index     = (block_address mod s );
4.   set_t set   = cache[set_index];
5.   parallel_for(i in 0..n-1){
       if( set[i].tag = block_address and set[i].valid )
         return set[i].data[block_offset];
     }
1.   MISS! ...
```

Direct-mapped multi-word cache



w = bytes/word	k = words/block	s = blocks/set	c = bytes/cache

4-way set associative cache



w = bytes/word	*k* = words/block	*s* = blocks/set	*c* = bytes/cache

Block replacement policy

- for fully associative or set associative caches
- random selection
  - simple: just evict a random block
  - possible hardware support
- LRU - replace block unused for the longest time
- ↑ cache size:
  - ↓ miss rate for both policies
  - ↓ advantage of LRU over random

Compare number of misses: LRU replacement

- block addresses accessed: 0,8,0,6,8
  (four 1-word blocks)
- direct-mapped: (address→block) 0→0, 6→2, 8→0
  cache content: M_0 _ _ _, M_8 _ _ _, M_0 _ _ _, M_0 _ M_6 _, M_8 _ M_6 _.
  5 misses
- 2-way set assoc.: (address→set) 0→0, 6→0, 8→0
  cache content: M_0 _ _ _, M_0M_8 _ _, M_0M_8 _ _, M_0M_6 _ _, M_8M_6 _ _.
  4 misses, 1 hit
- fully associative:
  cache content: M_0 _ _ _, M_0M_8 _ _, M_0M_8 _ _, $M_0M_8M_6$ _, $M_0M_8M_6$ _.
  3 misses, 2 hits

Associative cache: size and performance

- resources required
  - storage
  - processing
- performance
  - miss rate
  - hit time
  - clock speed
- effect of increasing associativity on
  - resources?
  - performance?

Average Memory Access Time

- want the Average Memory Access Time (AMAT)
  - take into account all levels of the hierarchy
  - calculate MT_{cpu} : AMAT for ISA-level accesses
  - follow the abstract hierarchy

$AMAT_{CPU} = AMAT_{L1}$

$AMAT_{L1} = HitTime_{L1} + MissRate_{L1} * AMAT_{L2}$

$AMAT_{L2} = HitTime_{L2} + MissRate_{L2} * AMAT_{M}$

$AMAT_{M} = constant$

$AMAT_{CPU} = HitTm_{L1} + MissRt_{L1}(HitTm_{L2} + MissRt_{L2}\ AMAT_{M})$

Example

- assume
  - L1 hit time = 1 cycle
  - L1 miss rate = 5%
  - L2 hit time = 5 cycles
  - L2 miss rate = 15% (% L1 misses that miss)
  - L2 miss penalty = 200 cycles
- L1 miss penalty = 5 + 0.15 x 200 = 35 cycles
- AMAT = 1 + 0.05 x 35
  = 2.75 cycles

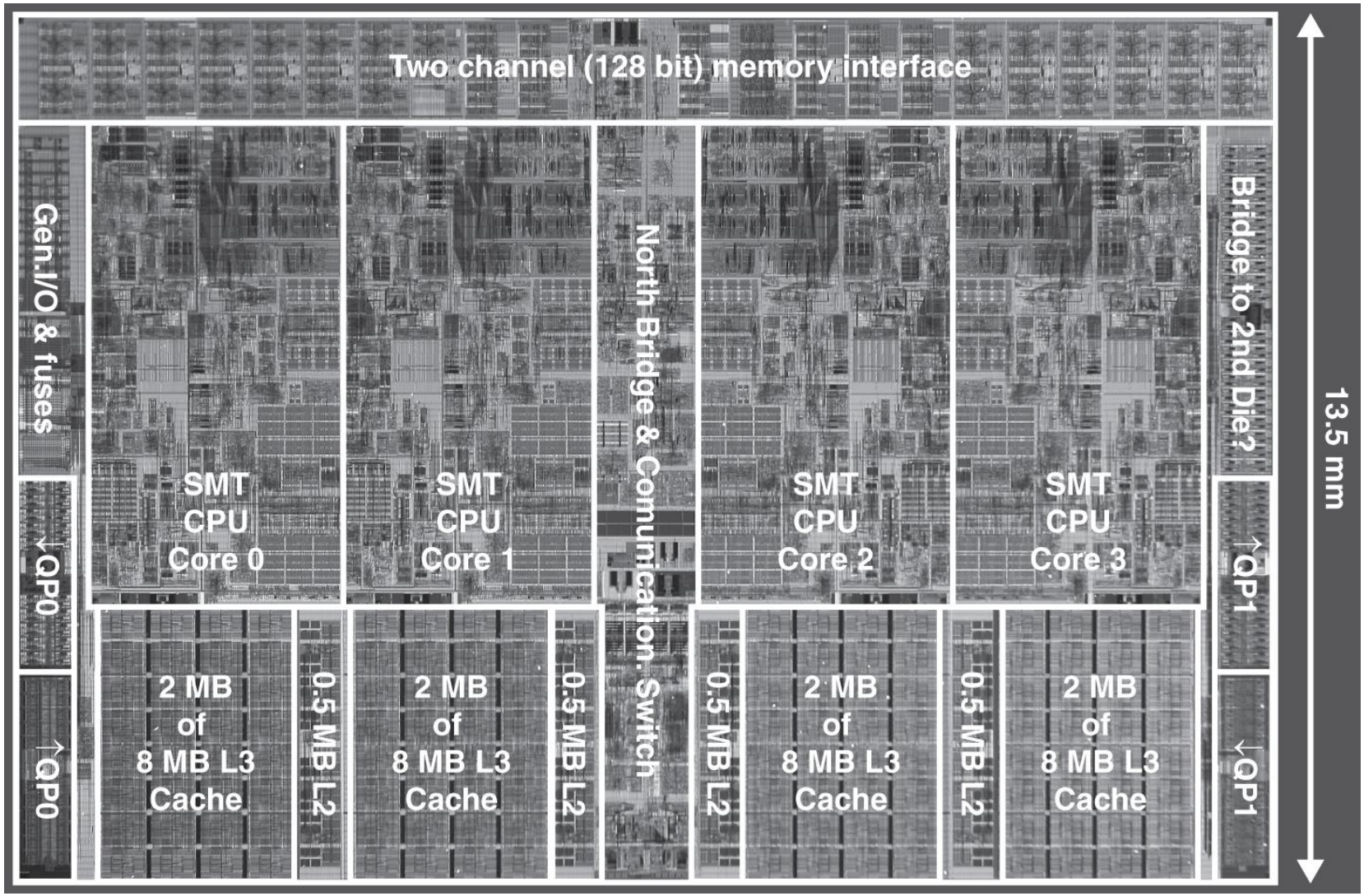
Example: without L2 cache

- assume
  - L1 hit time = 1 cycle
  - L1 miss rate = 5%
  - L1 miss penalty = 200 cycles
- AMAT = 1 + 0.05 x 200
  = 11 cycles

- 4 times faster with L2 cache! (2.75 versus 11)

Multi-level on-chip caches

Intel Nehalem - per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache



3-level cache organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available