Design of Embedded Hardware and Firmware

Cache Memory

Andrea Guerrieri

HES-SO//Genève

andrea.guerrieri@hesge.ch

Motivation

- Ideal world: Data immediately available for the CPU

Motivation

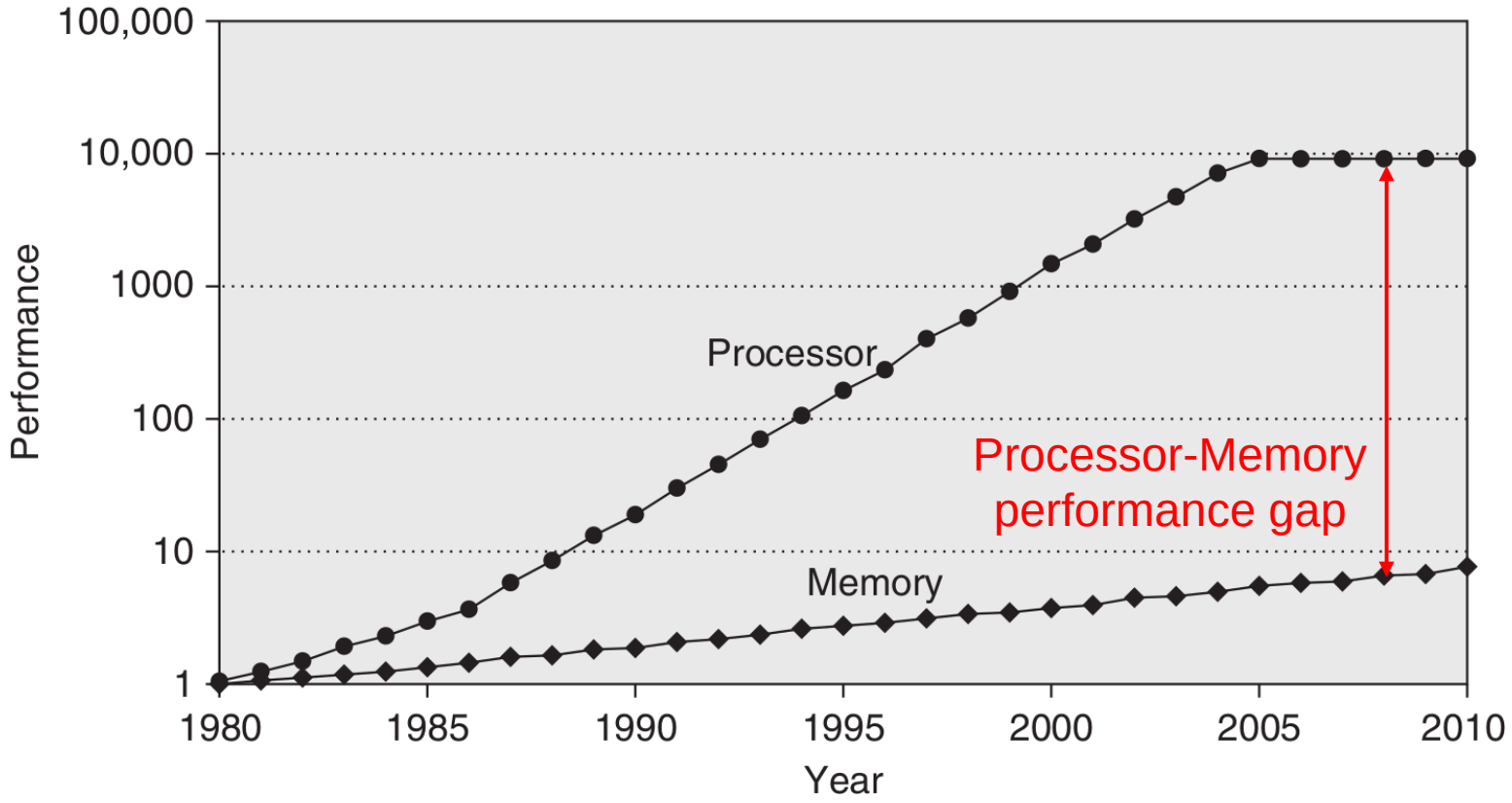
- For example…

```
        ldhu    r2,0(r2)
        sth     r2,-20(fp)
gray = (((rgb>>11)&0x1F)<<3)*21; // red part
        ldhu    r2,-20(fp)
        srli    r2,r2,11
        andi    r2,r2,65535
        slli    r2,r2,3
        andi    r2,r2,255
        muli    r2,r2,21
        stw     r2,-16(fp)
gray += (((rgb>>5)&0x3F)<<2)*72; // green part
        ldhu    r2,-20(fp)
        srli    r2,r2,5
        andi    r2,r2,65535
        add     r2,r2,r2
        add     r2,r2,r2
        andi    r2,r2,255
        muli    r2,r2,72
        ldw     r3,-16(fp)
        add     r2,r3,r2
        stw     r2,-16(fp)
gray += (((rgb>>0)&0x1F)<<3)*7; // blue part
        ldhu    r2,-20(fp)
        slli    r2,r2,3
        andi    r2,r2,255
        muli    r2,r2,7
        ldw     r3,-16(fp)
        add     r2,r3,r2
        stw     r2,-16(fp)
gray /= 100;
        ldw     r2,-16(fp)
        movi    r5,100
        mov     r4,r2
        call    22c8 <__divsi3>
        stw     r2,-16(fp)
```

Motivation

- Ideal world: Data immediately available

But…



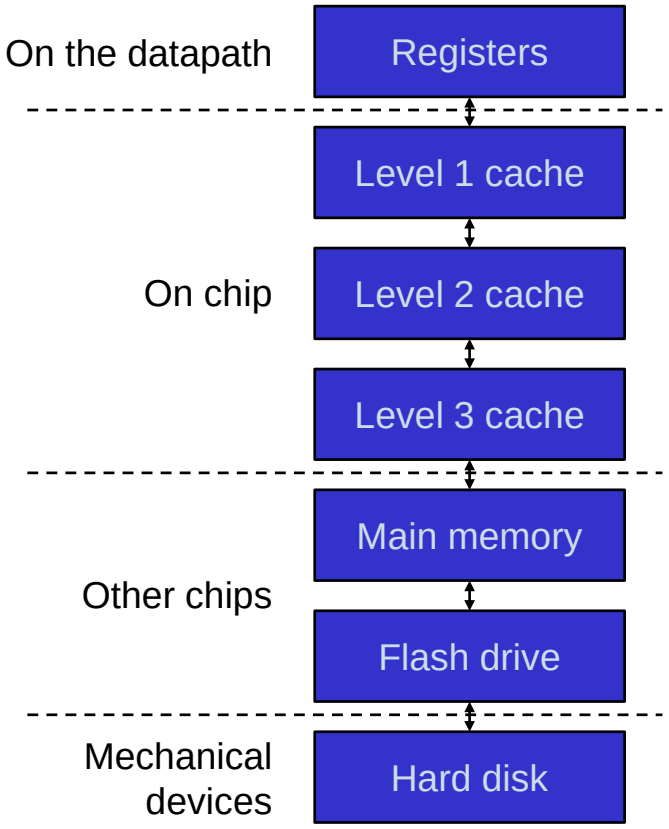
Memory Types

Technology	Typical access time
SRAM	0.5 – 2.5 ns
DRAM	50 – 70 ns
Flash	5 – 50 µs
Magnetic disk	5 – 20 ms

Memory Types

Technology	Typical access time	$ / Gb (2012)
SRAM	0.5 – 2.5 ns	$500 - $1000
DRAM	50 – 70 ns	$10 - $20
Flash	5 – 50 µs	$0.75 - $1
Magnetic disk	5 – 20 ms	$0.05 - $0.10

Memory Hierarchy



Access time	Capacity	Managed by
1 cycle	1 KB	Software/Compiler
2-4 cycles	32 KB	Hardware
10 cycles	256 KB	Hardware
40 cycles	10 MB	Hardware
200 cycles	10 GB	Software/OS
10-100 µs	100 GB	Software/OS
10 ms	1 TB	Software/OS

System Latencies

Table 2.2 Example Time Scale of System Latencies

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 µs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

Cache Memory

- Cache memories exploit the locality principle

Cache Memory

- Cache memories exploit the locality principle

- Locality principle

Cache Memory

- Cache memories exploit the locality principle

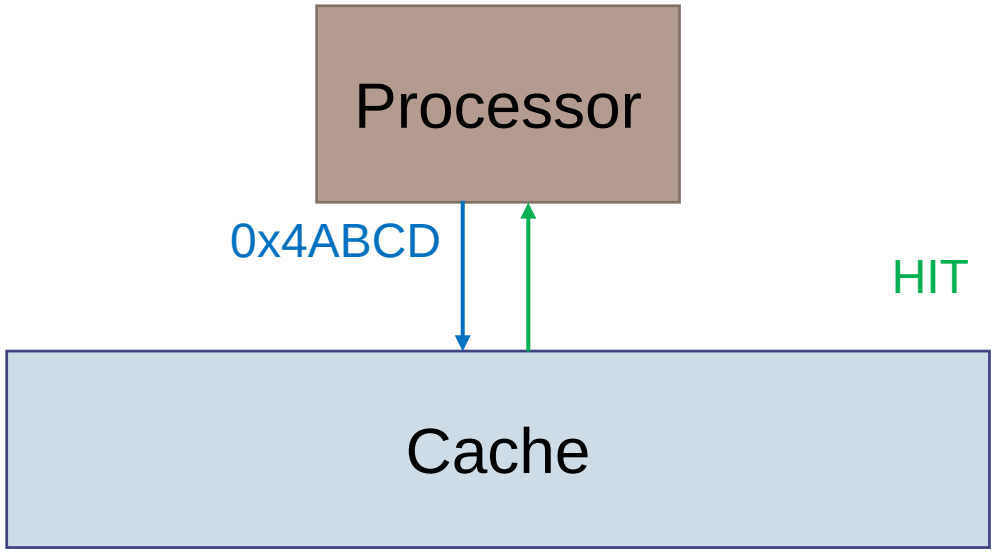
- Locality principle:
  - Temporal locality
    - Used data are probably reused again soon
      - Code: loops, functions…
      - Local variables

Cache Memory

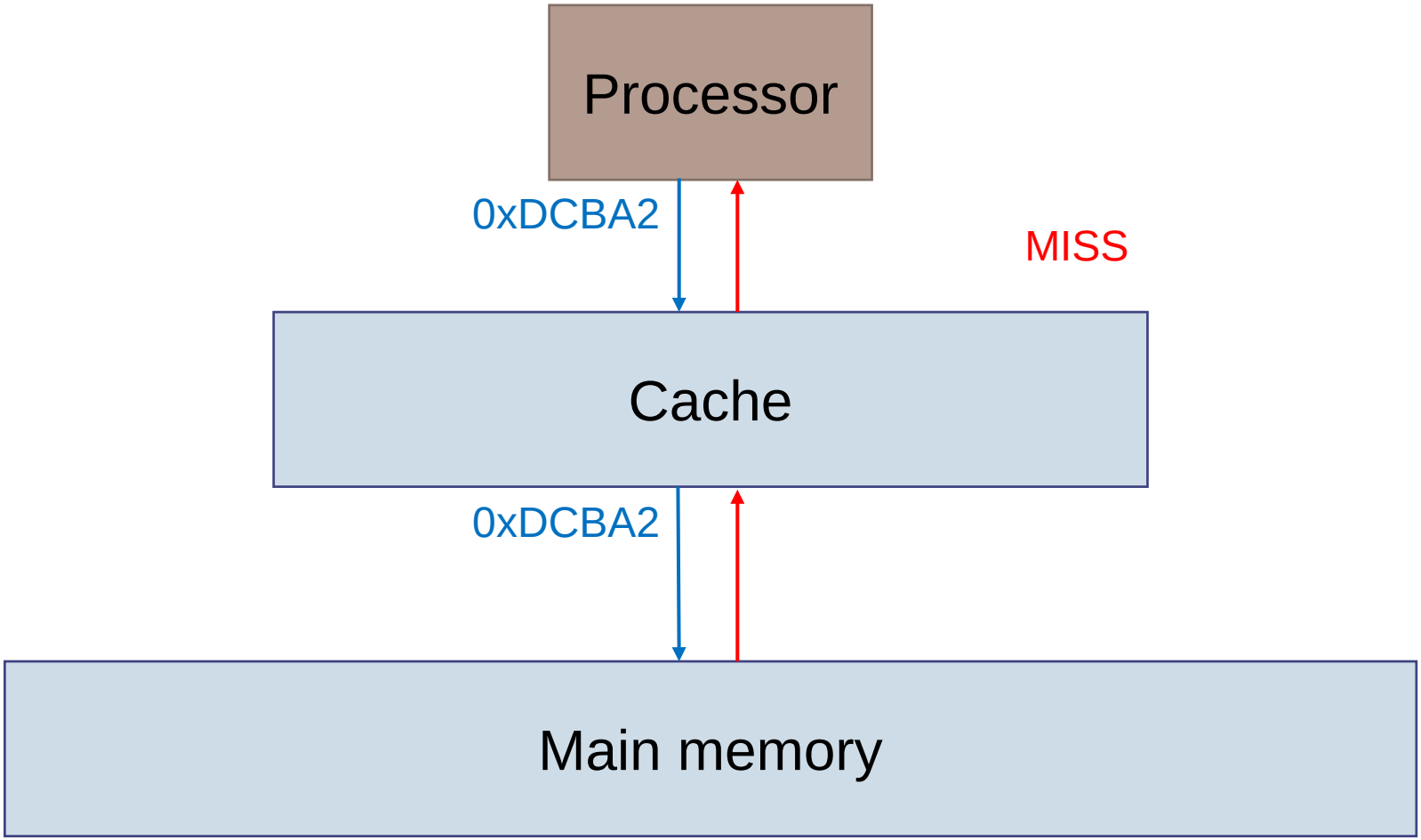
- Cache memories exploit the locality principle

- Locality principle:
  - Temporal locality
    - Used data are probably reused again soon
      - Code: loops, functions…
      - Local variables
  - Spatial locality
    - Data in adjacent addresses will probably be used soon
      - Code: Execution is mainly sequential
      - Arrays, tables, structures

Working Principle



Working Principle



Use of Cache

- Normally, multi-level cache systems
- Harvard architecture: Instructions and Data caches

Use of Cache

- Normally, multi-level cache systems
- Harvard architecture: Instructions and Data caches
- Cache metrics:
  - Hit ratio

$$HR = \frac{hits}{hits + misses} = 1 - MR$$

  - Miss ratio

$$MR = \frac{misses}{hits + misses} = 1 - HR$$

  - Average Memory Access Time

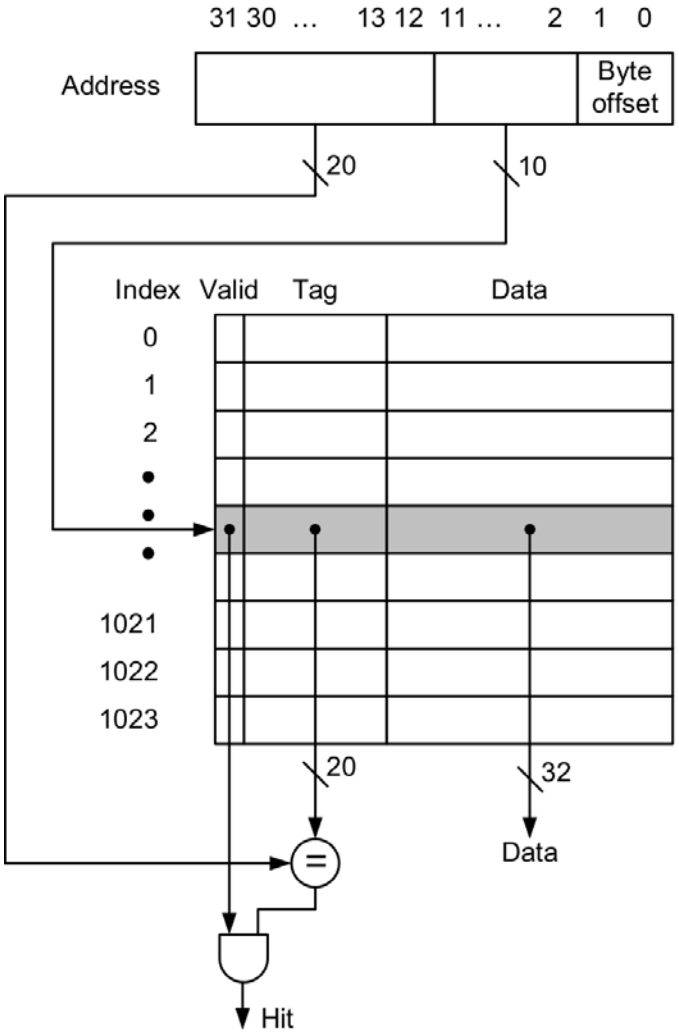
$$AMAT = HitTime + MissRatio \times MissPenalty$$

Types of Caches

Types of Caches

- Direct-mapped
  - One position in memory corresponds to one position in cache

Direct-mapped Cache



Types of Caches

- Direct-mapped
  - One position in memory corresponds to one position in cache
  - Easy to implement. Low usage of resources. Reduced tag size

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022
  - 0x0024
  - 0x0022
  - 0x0024
  - 0x0026
  - 0x0016
  - 0x0024
  - 0x0018

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022
  - 0x0024
  - 0x0022
  - 0x0024
  - 0x0026
  - 0x0016
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0			
1			
2			
3			
4			
5			
6			
7			

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022
  - 0x0024
  - 0x0022
  - 0x0024
  - 0x0026
  - 0x0016
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	0		
2	0		
3	0		
4	0		
5	0		
6	0		
7	0		

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - **0x0022**
  - 0x0024
  - 0x0022
  - 0x0024
  - 0x0026
  - 0x0016
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	0		
2	0		
3	0		
4	0		
5	0		
6	0		
7	0		

Bin: 0000 0000 0010 0010

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - **0x0022** → MISS
  - 0x0024
  - 0x0022
  - 0x0024
  - 0x0026
  - 0x0016
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	1	0x002	Mem(0x0022)
2	0		
3	0		
4	0		
5	0		
6	0		
7	0		

Bin: 0000 0000 0010 0010

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022 → MISS
  - **0x0024**
  - 0x0022
  - 0x0024
  - 0x0026
  - 0x0016
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	1	0x002	Mem(0x0022)
2	0		
3	0		
4	0		
5	0		
6	0		
7	0		

Bin: 0000 0000 0010 0100

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022 → MISS
  - **0x0024** → MISS
  - 0x0022
  - 0x0024
  - 0x0026
  - 0x0016
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	1	0x002	Mem(0x0022)
2	1	0x002	Mem(0x0024)
3	0		
4	0		
5	0		
6	0		
7	0		

Bin: 0000 0000 0010 0100

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022 → MISS
  - 0x0024 → MISS
  - **0x0022** → HIT
  - 0x0024
  - 0x0026
  - 0x0016
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	1	0x002	Mem(0x0022)
2	1	0x002	Mem(0x0024)
3	0		
4	0		
5	0		
6	0		
7	0		

Bin: 0000 0000 0010 0010

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022 → MISS
  - 0x0024 → MISS
  - 0x0022 → HIT
  - **0x0024** → HIT
  - 0x0026
  - 0x0016
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	1	0x002	Mem(0x0022)
2	1	0x002	Mem(0x0024)
3	0		
4	0		
5	0		
6	0		
7	0		

Bin: 0000 0000 0010 0100

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022 → MISS
  - 0x0024 → MISS
  - 0x0022 → HIT
  - 0x0024 → HIT
  - **0x0026** → MISS
  - 0x0016
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	1	0x002	Mem(0x0022)
2	1	0x002	Mem(0x0024)
3	1	0x002	Mem(0x0026)
4	0		
5	0		
6	0		
7	0		

Bin: 0000 0000 0010 0110

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022 → MISS
  - 0x0024 → MISS
  - 0x0022 → HIT
  - 0x0024 → HIT
  - 0x0026 → MISS
  - **0x0016** → MISS
  - 0x0024
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	1	0x002	Mem(0x0022)
2	1	0x002	Mem(0x0024)
3	1	0x001	Mem(0x0016)
4	0		
5	0		
6	0		
7	0		

Bin: 0000 0000 0001 0110

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022 → MISS
  - 0x0024 → MISS
  - 0x0022 → HIT
  - 0x0024 → HIT
  - 0x0026 → MISS
  - 0x0016 → MISS
  - **0x0024** → HIT
  - 0x0018

Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	1	0x002	Mem(0x0022)
2	1	0x002	Mem(0x0024)
3	1	0x001	Mem(0x0016)
4	0		
5	0		
6	0		
7	0		

Bin: 0000 0000 0010 0100

Direct-mapped Cache

- Example: Data/address: 16 bits. 8 blocks in cache
  - 0x0022 → MISS
  - 0x0024 → MISS
  - 0x0022 → HIT
  - 0x0024 → HIT
  - 0x0026 → MISS
  - 0x0016 → MISS
  - 0x0024 → HIT
  - **0x0018** → MISS

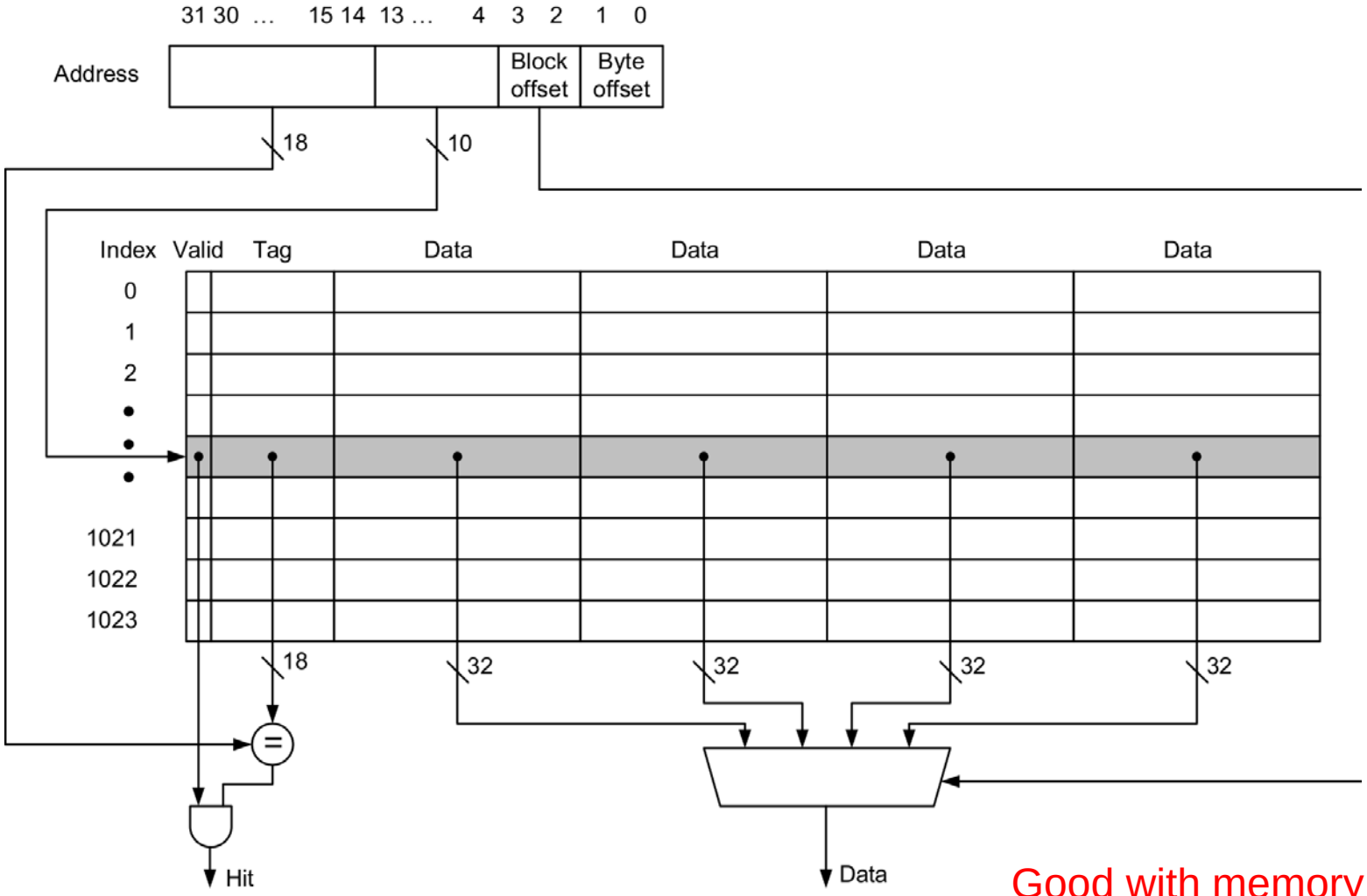
Block address	Valid (1 bit)	Tag (12 bits)	Data (16 bits)
0	0		
1	1	0x002	Mem(0x0022)
2	1	0x002	Mem(0x0024)
3	1	0x001	Mem(0x0016)
4	1	0x001	Mem(0x0018)
5	0		
6	0		
7	0		

Bin: 0000 0000 0001 1000

Types of Caches

- Direct-mapped
  - One position in memory corresponds to one position in cache
  - Easy to implement. Low usage of resources. Reduced tag size
  - Conflict addresses: high miss rate

Direct-mapped Cache (blocks)



Good with memory bursts!

Types of Caches

- Fully-associative
  - Memory positions can be mapped anywhere in the cache

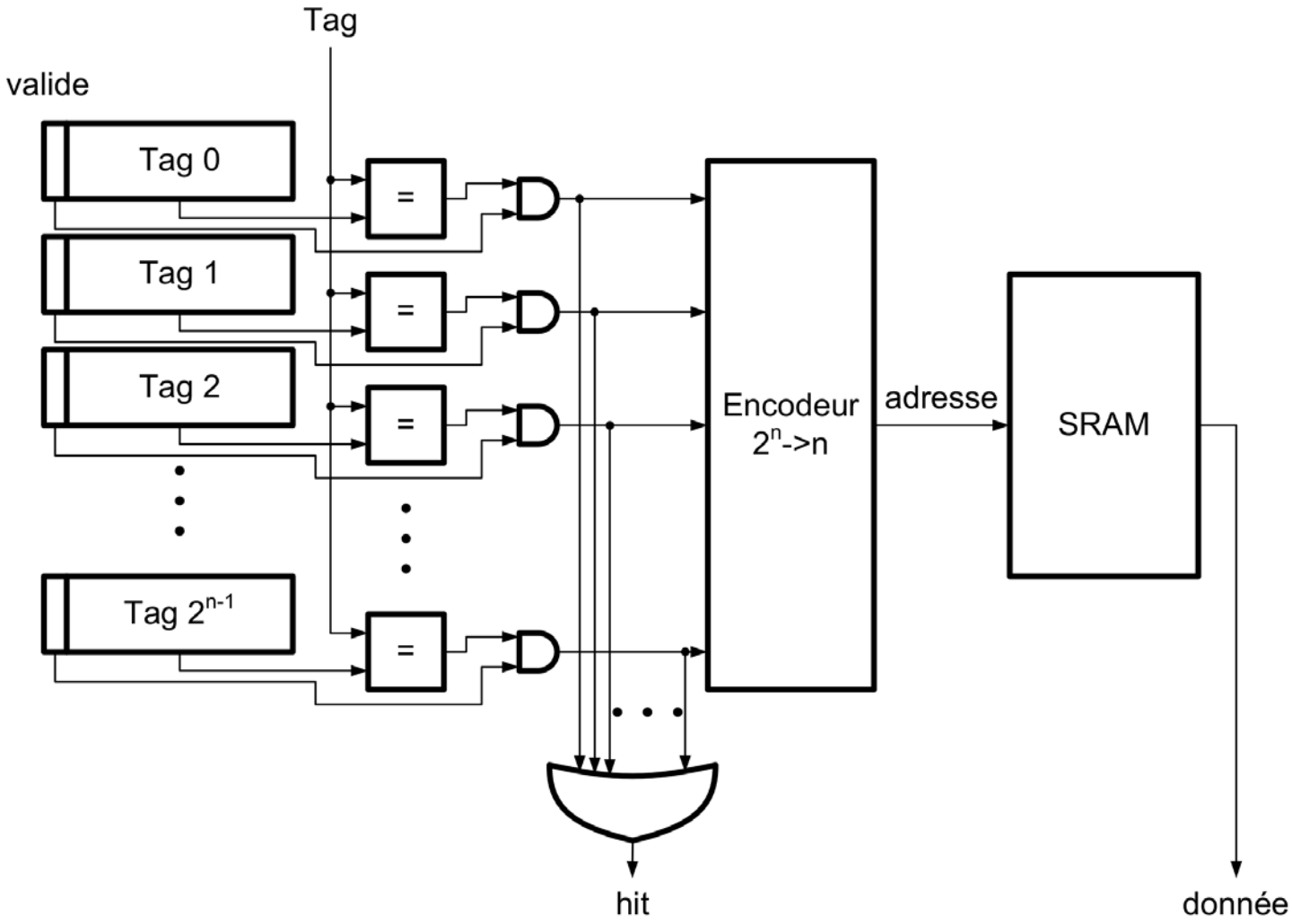
Types of Caches

- Fully-associative
  - Memory positions can be mapped anywhere in the cache
  - Low miss rate

Types of Caches

- Fully-associative
  - Memory positions can be mapped anywhere in the cache
  - Low miss rate
  - One comparator and full tag in each line of cache

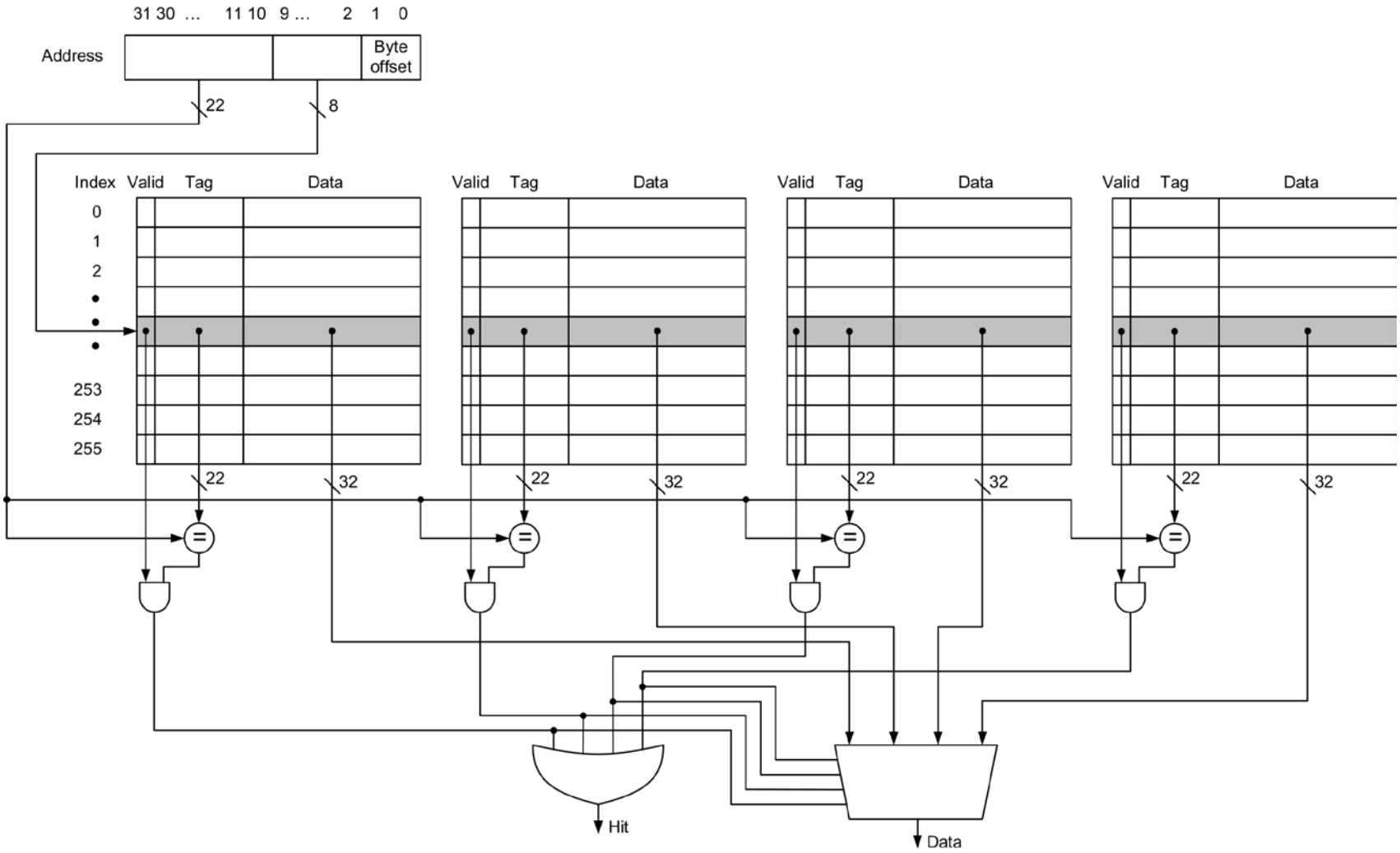
Fully-associative Cache



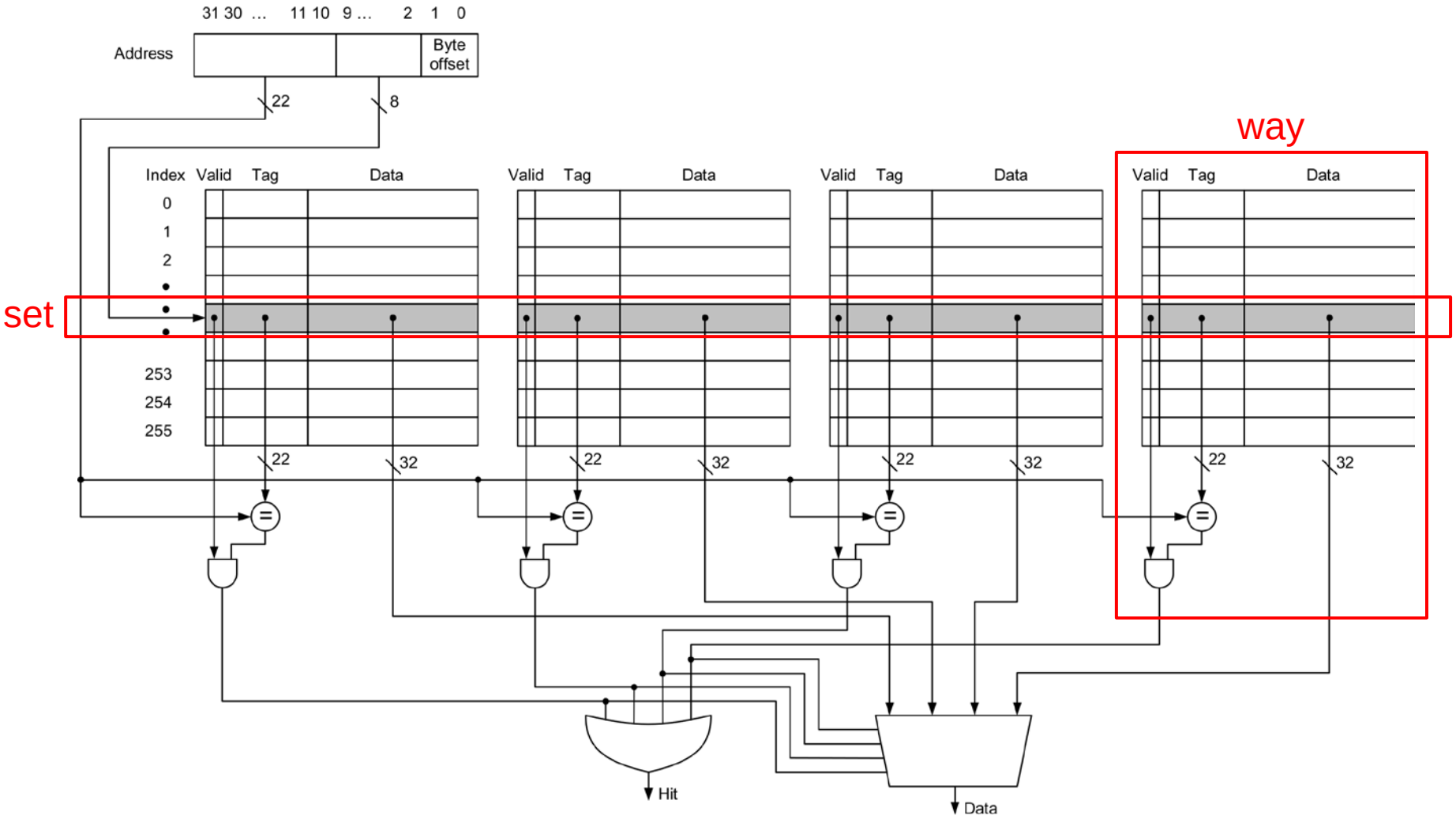
Types of Caches

- Set-associative
  - In between the two other options. Can be seen as N direct-mapped sub-caches in parallel
  - Good compromise between the two options

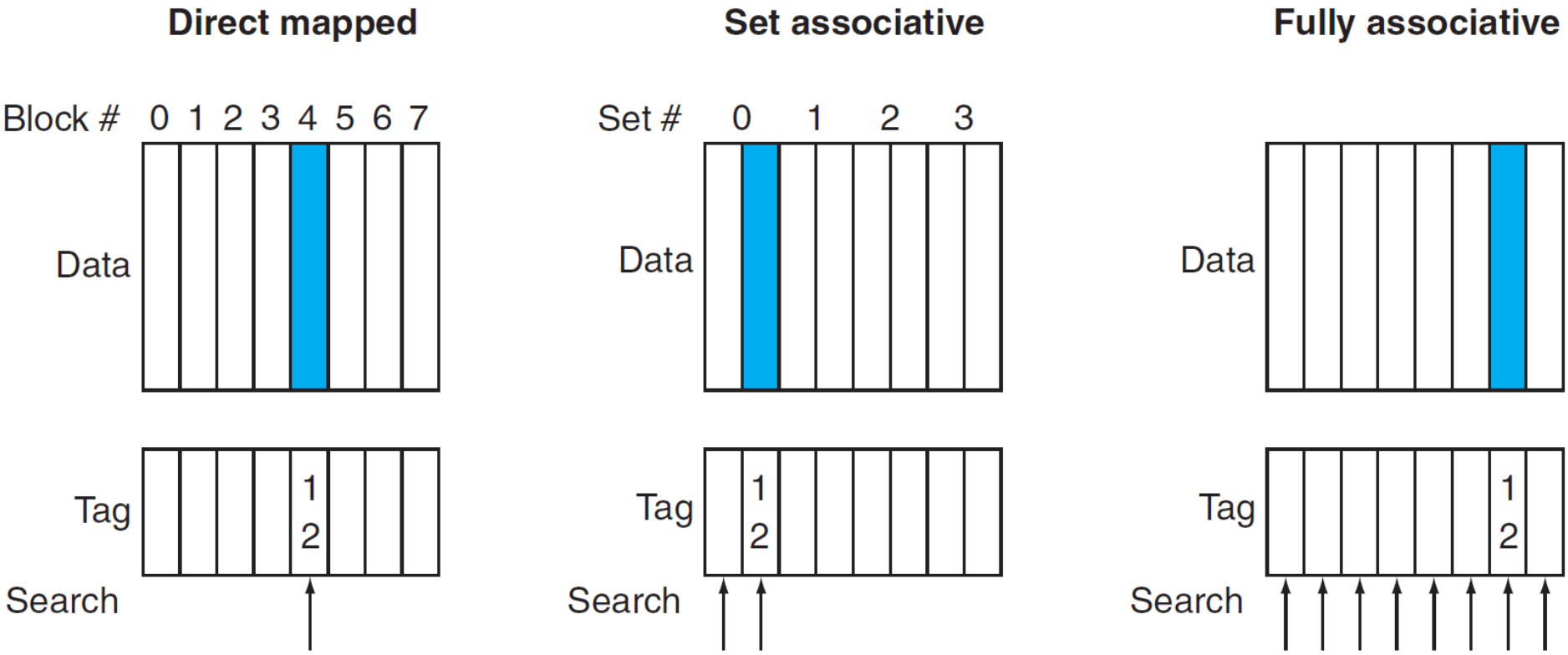
4-way (256-set) set-associative cache



4-way (256-set) set-associative cache



Types of Caches



Types of Caches

- Direct-mapped
  - One position in memory corresponds to one position in cache
  - Easy to implement. Low usage of resources. Reduced tag size
  - Conflict addresses: high miss rate (partial solution: increase block size)
- Fully-associative
  - Memory positions can be mapped anywhere in the cache
  - Low miss rate
  - One comparator and full tag in each line of cache
- Set-associative
  - In between the two other options. Can be seen as N direct-mapped sub-caches in parallel
  - Good compromise between the two options

Replacement Principles

- Only for associative caches (any block or by sets)

Replacement Principles

- Only for associative caches (any block or by sets)
- The optimal choice would be the block accessed further in the future (or never)

Replacement Principles

- Only for associative caches (any block or by sets)
- The optimal choice would be the block accessed further in the future (or never)
- Predicting the future: by looking at the past (temporal locality)

Replacement Principles

- Only for associative caches (any block or by sets)
- The optimal choice would be the block accessed further in the future (or never)
- Predicting the future: by looking at the past (temporal locality)

- Options:
  - Random choice
  - Not Last Used (NLU)
    - Requires storing last used
  - Least Recently Used (LRU)
    - Requires storing (and updating!) usage order

Write policies (any cache type)

- **Write-through**
  - Data are written immediately to the main memory
  - Simple. Avoid consistency problems
  - Inefficient. CPU waiting for every write operation

Write policies (any cache type)

- **Write-through**
  - Data are written immediately to the main memory
  - Simple. Avoid consistency problems
  - Inefficient. CPU waiting for every write operation
- **Write-back**
  - Changing data in main memory only when removed from cache
  - A **dirty bit** is used to know if the value in cache and memory are consistent
  - Write to main memory only with a cache miss and dirty bit active

Write policies (any cache type)

- **Write-through**
  - Data are written immediately to the main memory
  - Simple. Avoid consistency problems
  - Inefficient. CPU waiting for every write operation
- **Write-back**
  - Changing data in main memory only when removed from cache
  - A **dirty bit** is used to know if the value in cache and memory are consistent
  - Write to main memory only with a cache miss and dirty bit active
- To avoid blocking the CPU while executing the write, we can use a **write buffer** (with multiple entries) from the cache to the main memory in any of the previous policies

Write policies (any cache type)

- **Write-through**
  - Data are written immediately to the main memory
  - Simple. Avoid consistency problems
  - Inefficient. CPU waiting for every write operation
- **Write-back**
  - Changing data in main memory only when removed from cache
  - A **dirty bit** is used to know if the value in cache and memory are consistent
  - Write to main memory only with a cache miss and dirty bit active
- To avoid blocking the CPU while executing the write, we can use a **write buffer** (with multiple entries) from the cache to the main memory in any of the previous policies
- **Write allocate** policy is typically used: Blocks are always allocated on a write miss (as with a read miss)

Nios-II cache memories

Instruction Cache

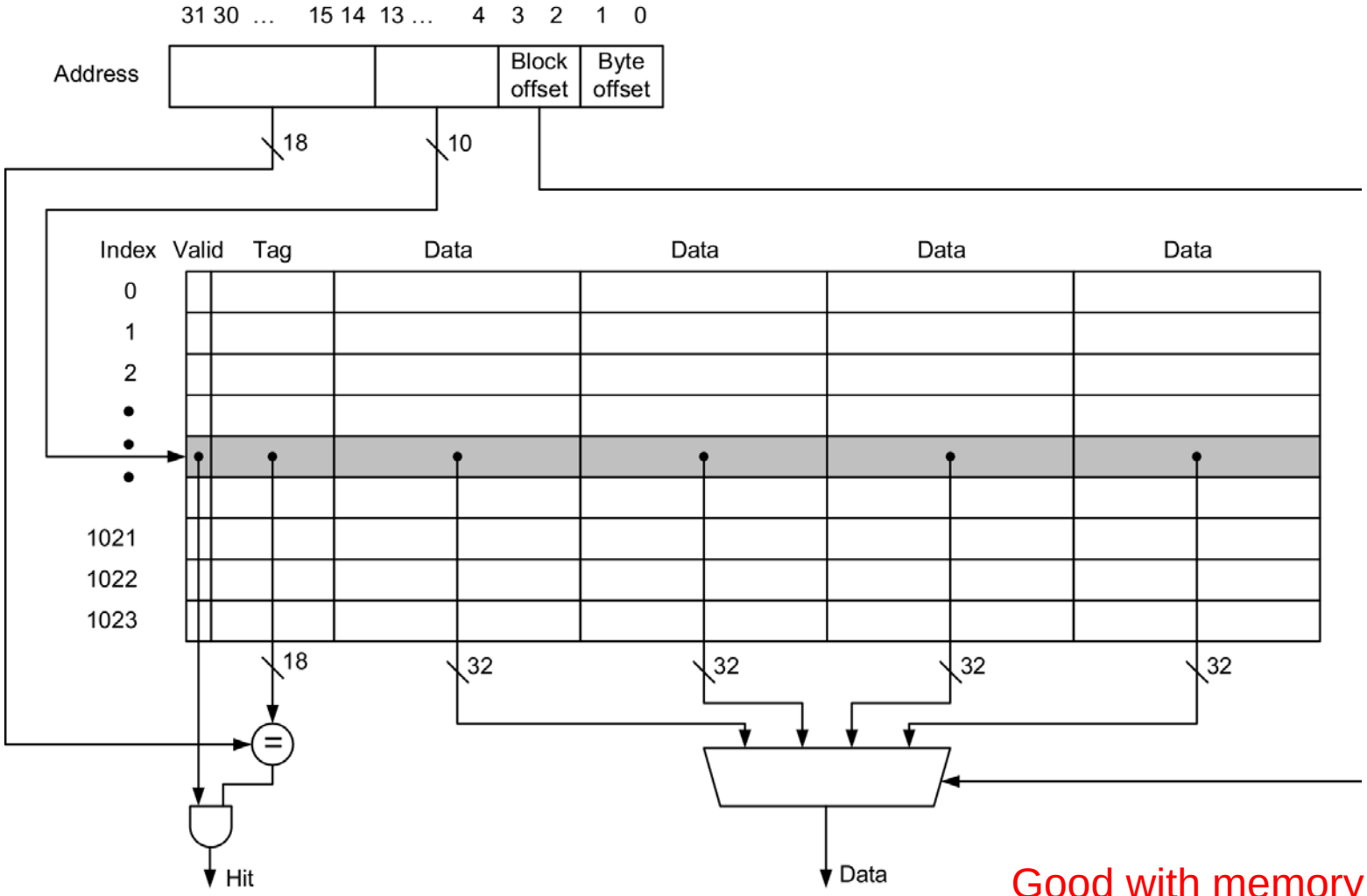
The instruction cache memory has the following characteristics:

- Direct-mapped cache implementation.
- 32 bytes (8 words) per cache line.
- The instruction master port reads an entire cache line at a time from memory, and issues one read per clock cycle.
- Critical word first.
- Virtually-indexed, physically-tagged, when MMU present.

Data Cache

- Direct-mapped cache implementation
- Line size of 32-bytes
- The data master port reads an entire cache line at a time from memory, and issues one read per clock cycle.
- Write-back
- Write-allocate (i.e., on a store instruction, a cache miss allocates the line for that address)
- Virtually-indexed, physically-tagged, when MMU present

Direct-mapped cache (blocks)



Good with memory bursts!

Nios-II cache memories

- Cache Byte Address Fields

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
tag													line		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
line											offset				

- Offset length fixed to 5 bits
- Line length depends on cache size
- Tag length depends on cache and main memory sizes
- Without MMU, address size is 31 bits
- MSB (bit-31) may be used to cache bypass (if enabled)

Nios-II cache memories

- The cache memories are optional. The need for higher memory performance (the need for cache memory) is application dependent. The inclusion of cache memory does not affect the functionality of programs, but only the speed at which the processor fetches instructions and reads/writes data.
- Many applications require the smallest possible processor core, and can trade-off performance for size. A Nios II might include one, both, or neither of the cache memories.
- Furthermore, the sizes of the cache memories are user-configurable.

Nios-II cache memories

The effectiveness of cache memory to improve performance is based on the following premises:

1. Regular memory is located off-chip, and access time is long compared to on-chip memory
2. The largest, performance-critical instruction loop is smaller than the instruction cache
3. The largest block of performance-critical data is smaller than the data cache

Nios-II cache memories

Optimal cache configuration is application specific. It's up to the HW designer to take the right decisions that are effective across the use case applications…

For example, if a Nios II processor system includes only on-chip memory and it never needs to access to off-chip memory, an instruction or data cache is unlikely to offer any performance gain.

Nios-II cache memories

As another example, if the critical loop of a program is 2 kilobytes (KB), but the size of the instruction cache is 1 KB, an instruction cache does not improve execution speed. In fact, an instruction cache may degrade performance in this situation.

If an application always requires certain data or sections of code to be located in cache memory for performance reasons, the tightly-coupled memory (TCM) feature might provide a more appropriate solution.