School of Engineering
InES Institute of Embedded Systems

Cache

Computer Engineering 2

Agenda

- **Principle of locality – The memory hierarchy**
- **Cache mechanics**
- **Cache organization**
  - Fully associative
  - Direct mapped
  - N-way set associative
- **Performance**
- **Replacement and write strategies**
- **The programmer's perspective**

Motivation

- **Situation**
  - Processor
    - Fast cycle time
  - Fast DRAM[1]
    - Slow cycle time (up to 100x)
    - Efficiently reads only in bursts
  - → Bridging the gap such that pipelining is effective!

- **Goal**
  - Access “slower” memory in bursts and maintain a fast cache for fast single accesses
  - But: Data consistency must be carefully managed, such that both, cache and main memory have the same data

[1] *Dynamic RAM*

Learning Objectives

- **At the end of this lesson you will be able**
  - to understand the principles of cache memory
  - to explain the principle of locality
  - to enumerate advantages and disadvantages of different cache models
    - Fully associative
    - Direct mapped
    - N-way set associative
  - to enumerate types of cache misses
  - to understand how cache size and cache hit rate are related
  - to name different replacement strategies

Principle of Locality

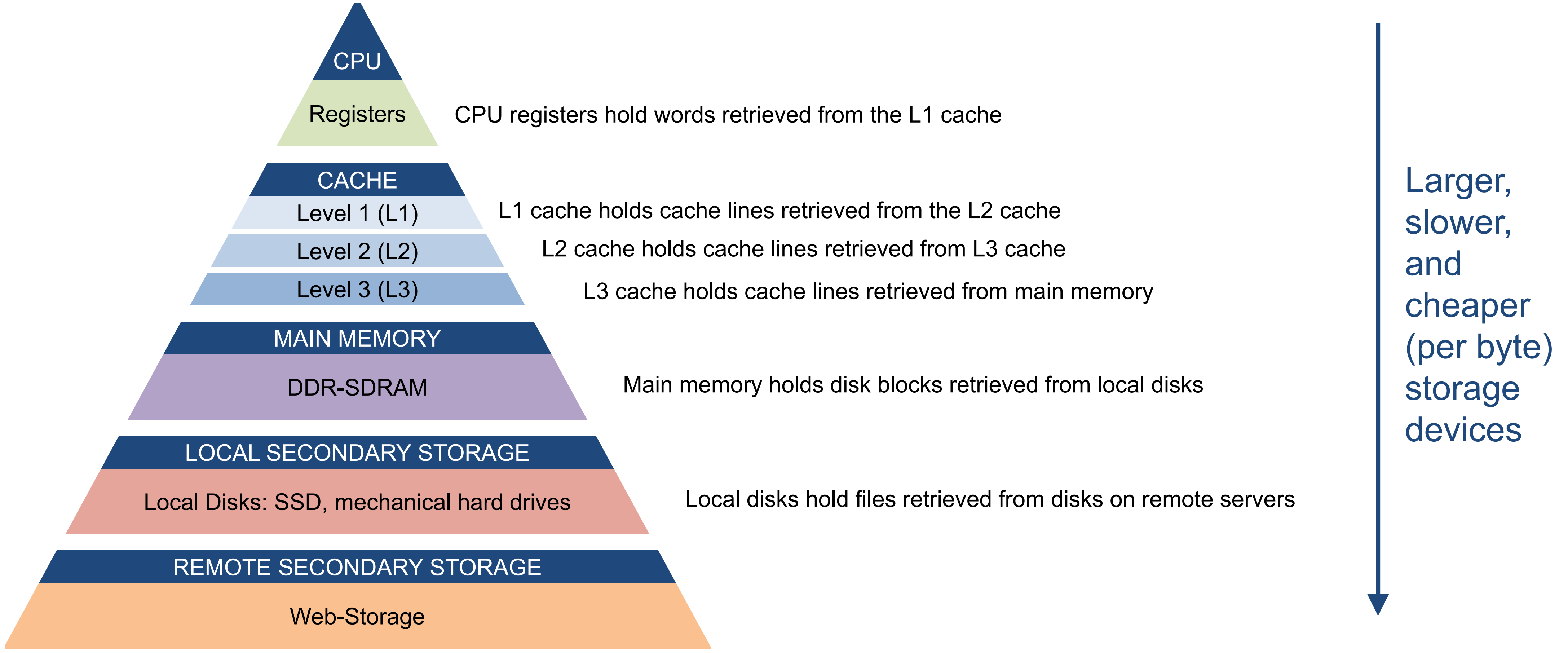
- **Principle of locality**

Programs usually access small regions of memory in a given interval of time

- **Spatial locality** → Current data location is likely being close to next accessed location
- **Temporal locality** → Current data location is likely being accessed again in near future

```
for(i = 0; i < 100000; i++) {      // incremental access
    a[i] = b[i];                   // → spatial locality
}

if (a[1234] == a[4321]) {          // → temporal locality
    a[1234] = 0;
}
```

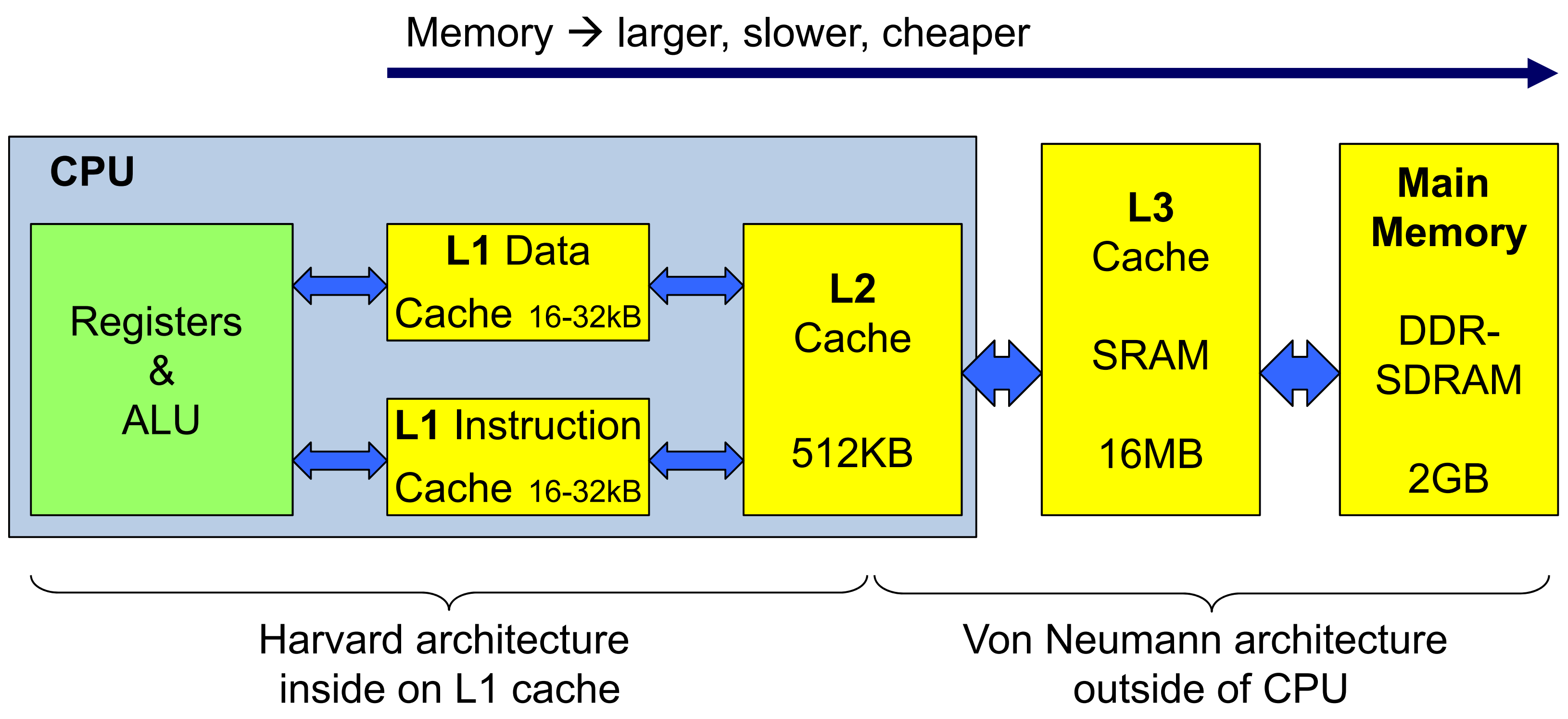
The Memory Hierarchy


CPU
Registers
CPU registers hold words retrieved from the L1 cache
CACHE
Level 1 (L1)
L1 cache holds cache lines retrieved from the L2 cache
Level 2 (L2)
L2 cache holds cache lines retrieved from L3 cache
Level 3 (L3)
L3 cache holds cache lines retrieved from main memory
MAIN MEMORY
DDR-SDRAM
Main memory holds disk blocks retrieved from local disks
LOCAL SECONDARY STORAGE
Local Disks: SSD, mechanical hard drives
Local disks hold files retrieved from disks on remote servers
REMOTE SECONDARY STORAGE
Web-Storage
Larger, slower, and cheaper (per byte) storage devices

Adapted from Bryant and O'Hallaron

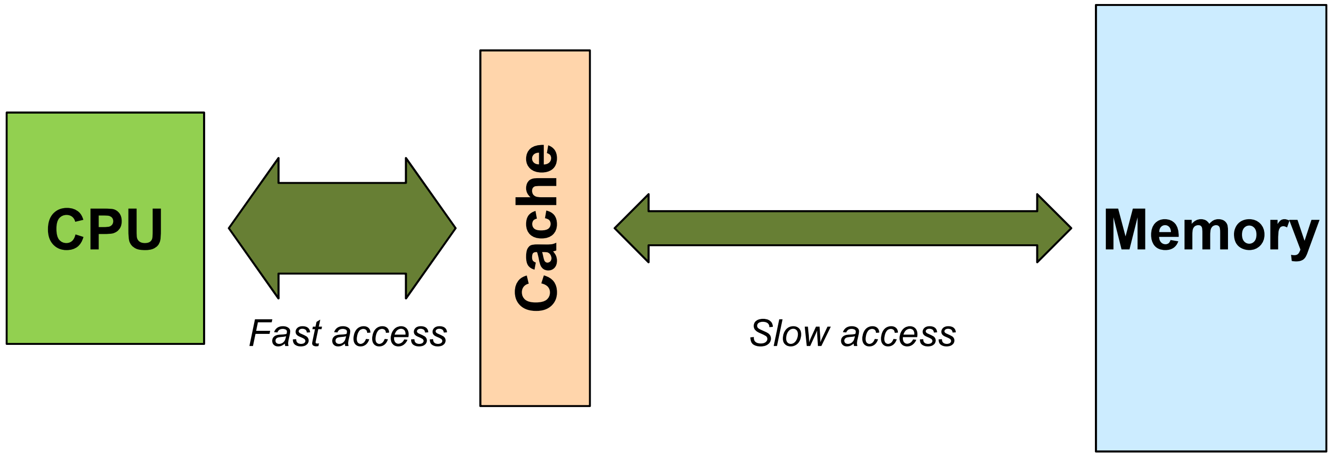
The Memory Hierarchy

- **Cache Levels**
  - Typical Cache Architecture


Memory → larger, slower, cheaper
CPU
Registers & ALU
L1 Data Cache 16-32kB
L1 Instruction Cache 16-32kB
L2 Cache 512KB
L3 Cache SRAM 16MB
Main Memory DDR-SDRAM 2GB
Harvard architecture inside on L1 cache
Von Neumann architecture outside of CPU

Cache Mechanics

- **Definition cache**
  - Computer memory with short access time
  - Storage of frequently or recently used instructions or data



CPU
Very small
Registers (Word)

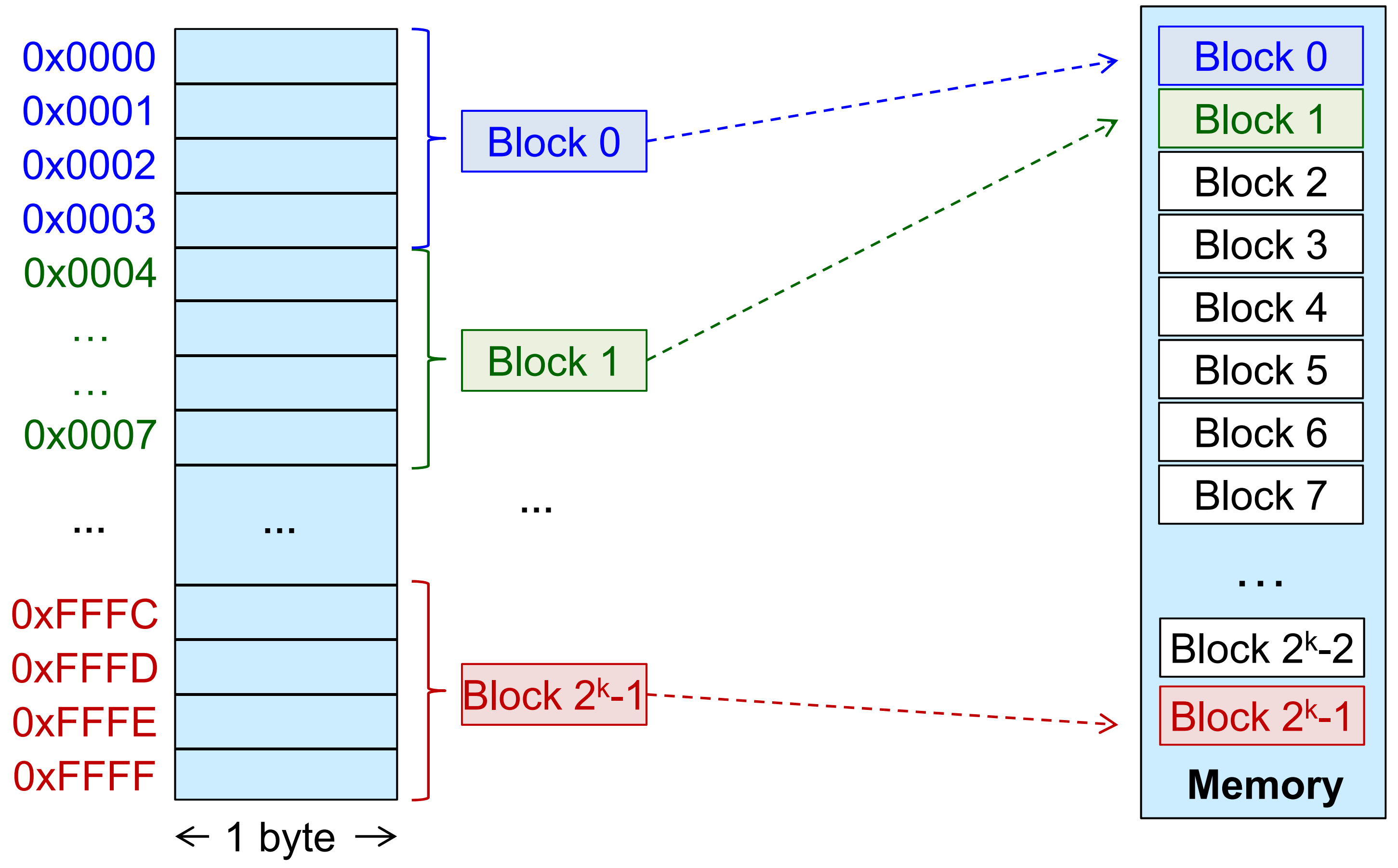
Cache
Small, fast, expensive
Caches a subset of memory blocks

Memory
Large, slow, cheap
Partitioned into memory blocks

Cache Mechanics

- **Memory blocks**
  - Address range is partitioned into memory blocks

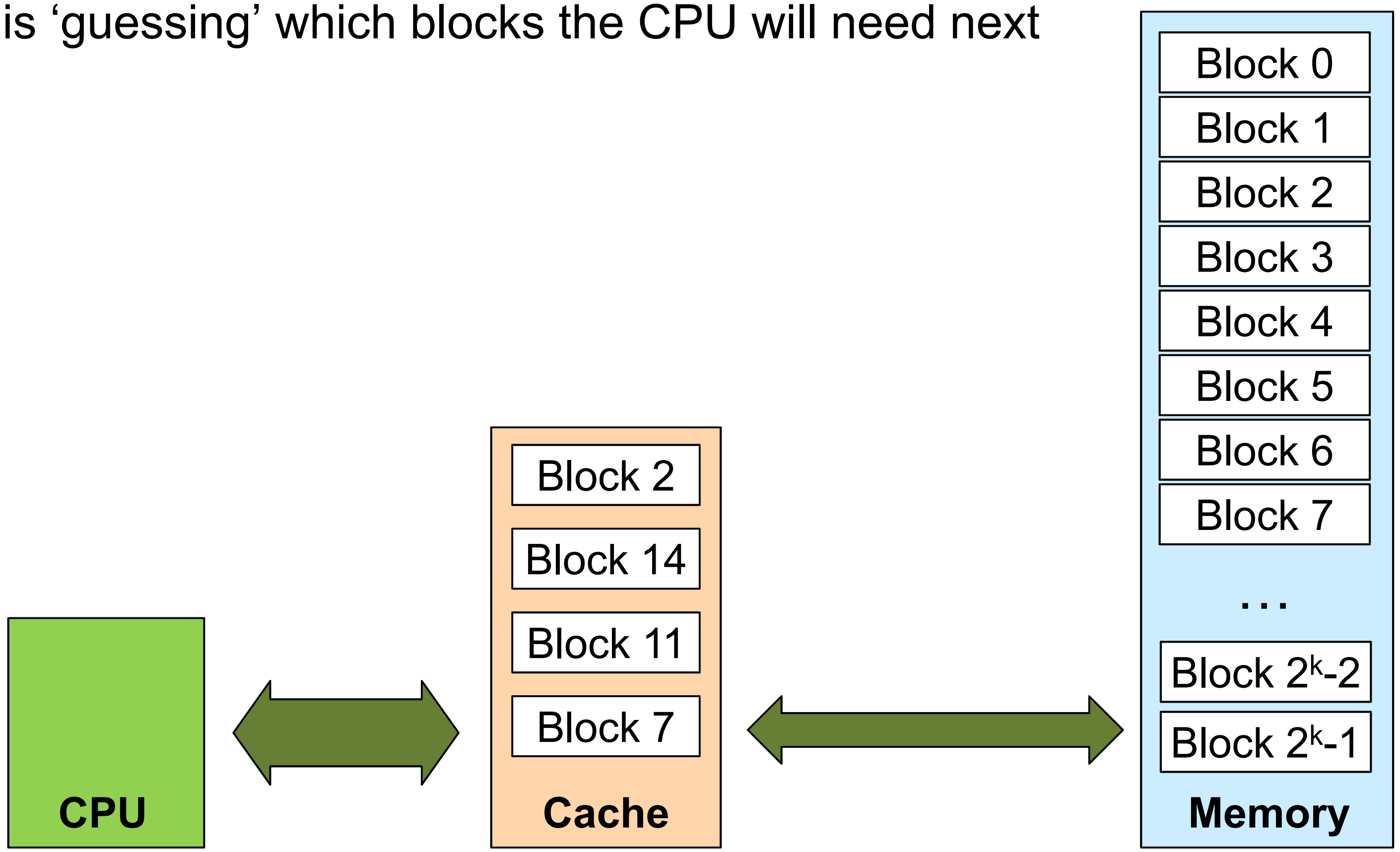
All examples given in this lecture are using
- ***hypothetical 16 bit wide addresses***
- ***blocks of 4 Bytes***



The number of Bytes per block is a design decision

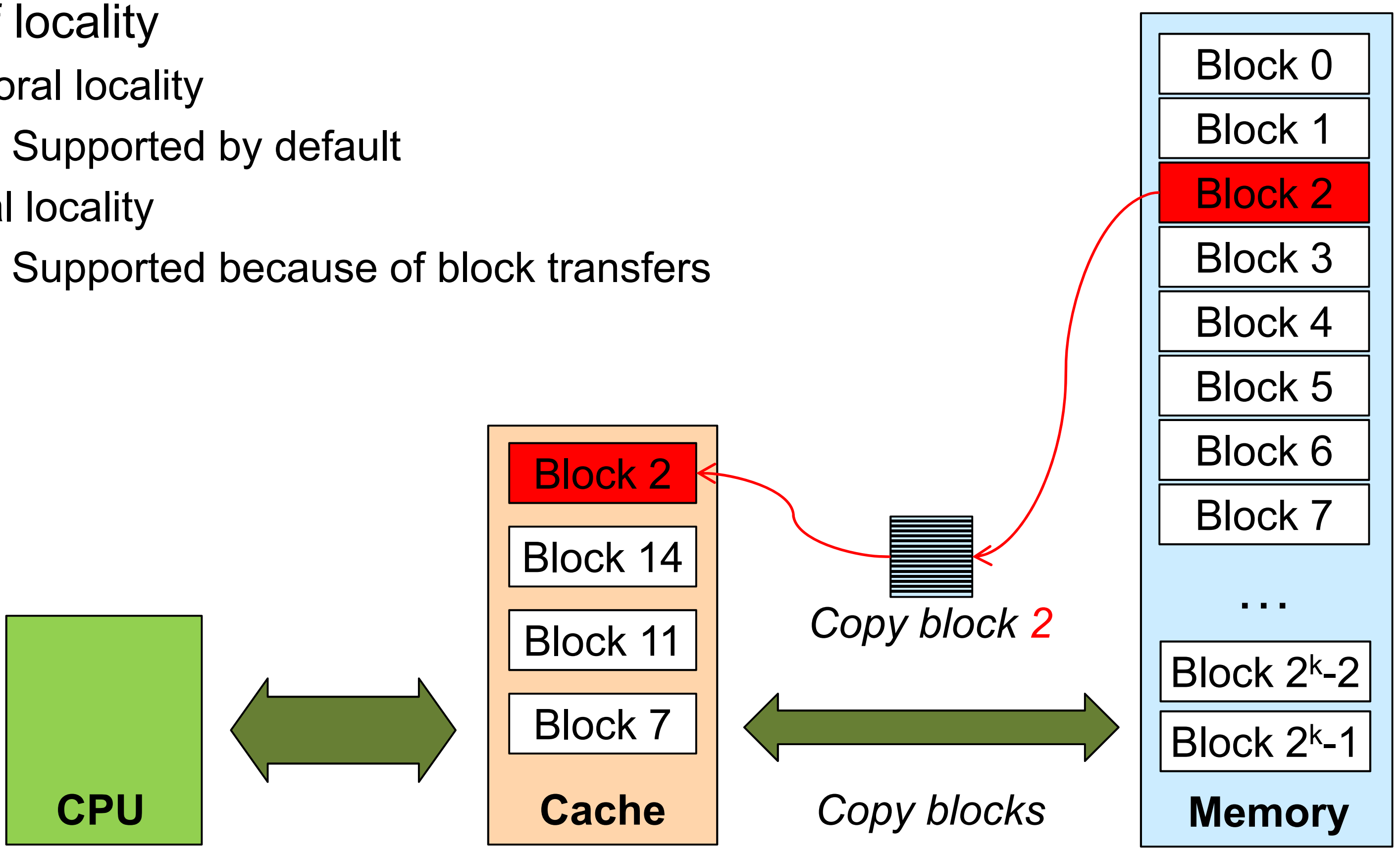
Cache Mechanics

- **Memory blocks**
  - Selected blocks of main memory copied to faster cache memory
  - The cache is ‘guessing’ which blocks the CPU will need next



Cache Mechanics

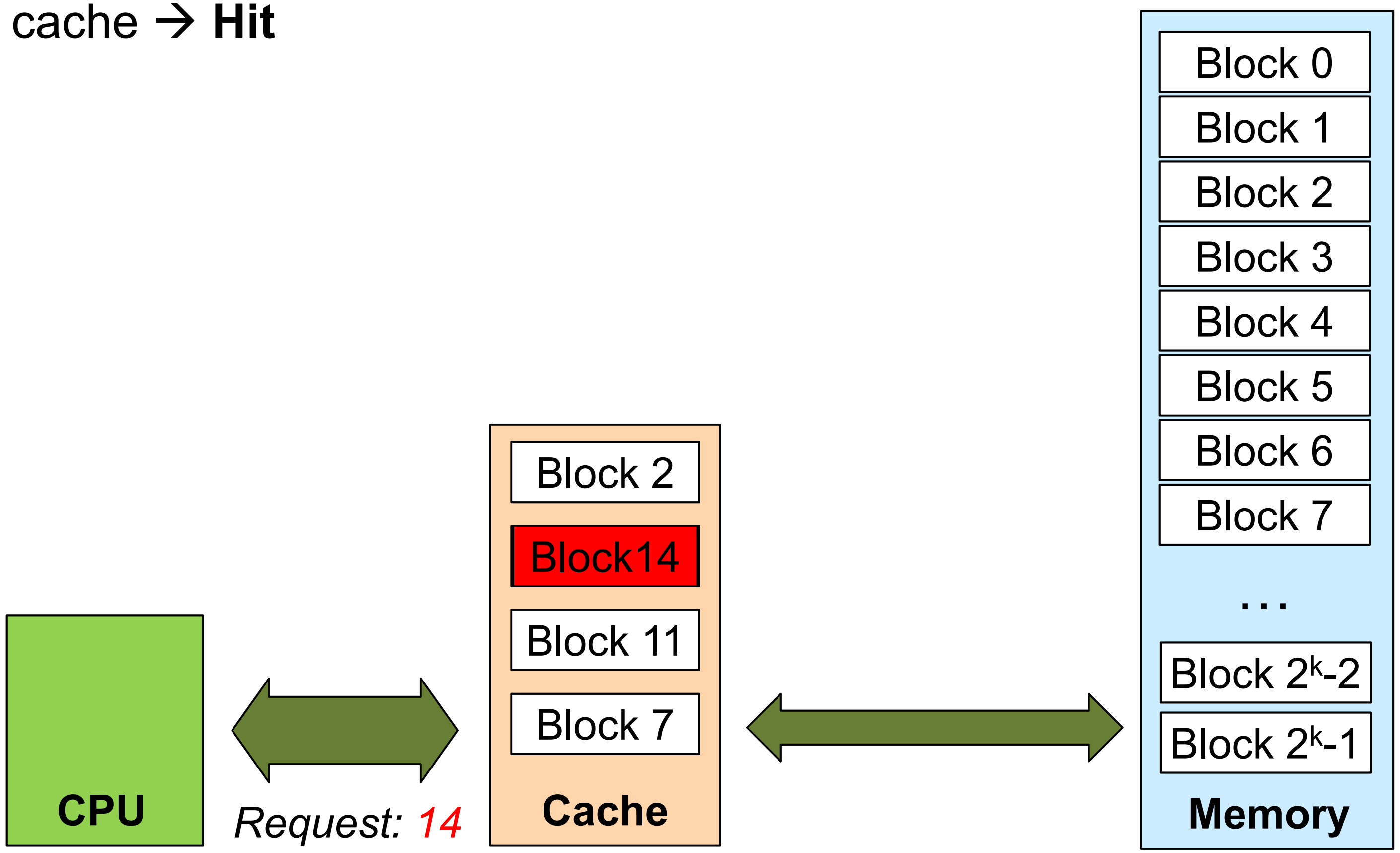
- **Memory blocks**
  - Blocks of main memory copied to faster cache memory
  - Principle of locality
    - Temporal locality
      - ▶ Supported by default
    - Spatial locality
      - ▶ Supported because of block transfers



Animation

Cache Mechanics

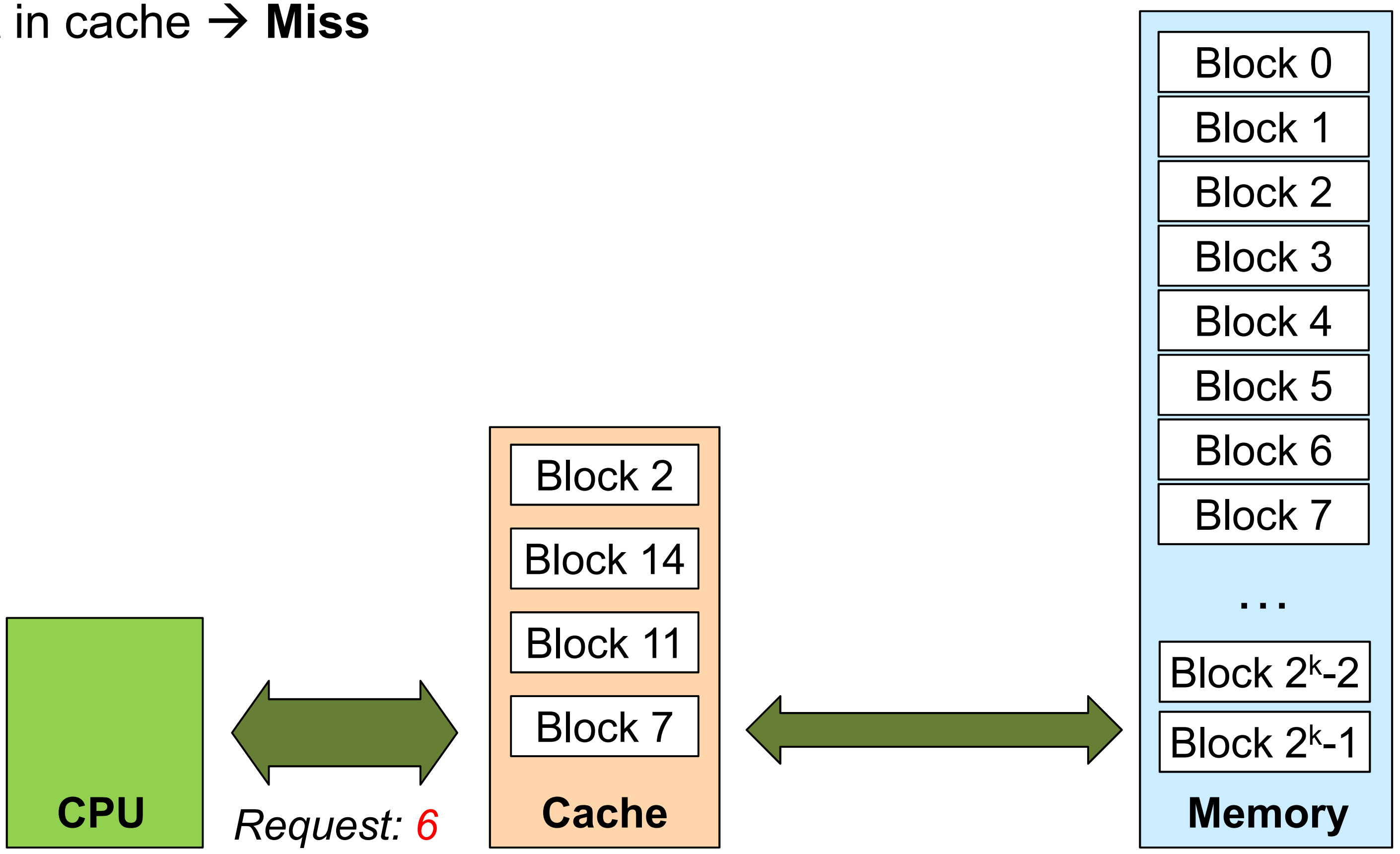
- **Cache hit**
  - Data in block 14 is needed
  - Block 14 in cache → **Hit**



Animation

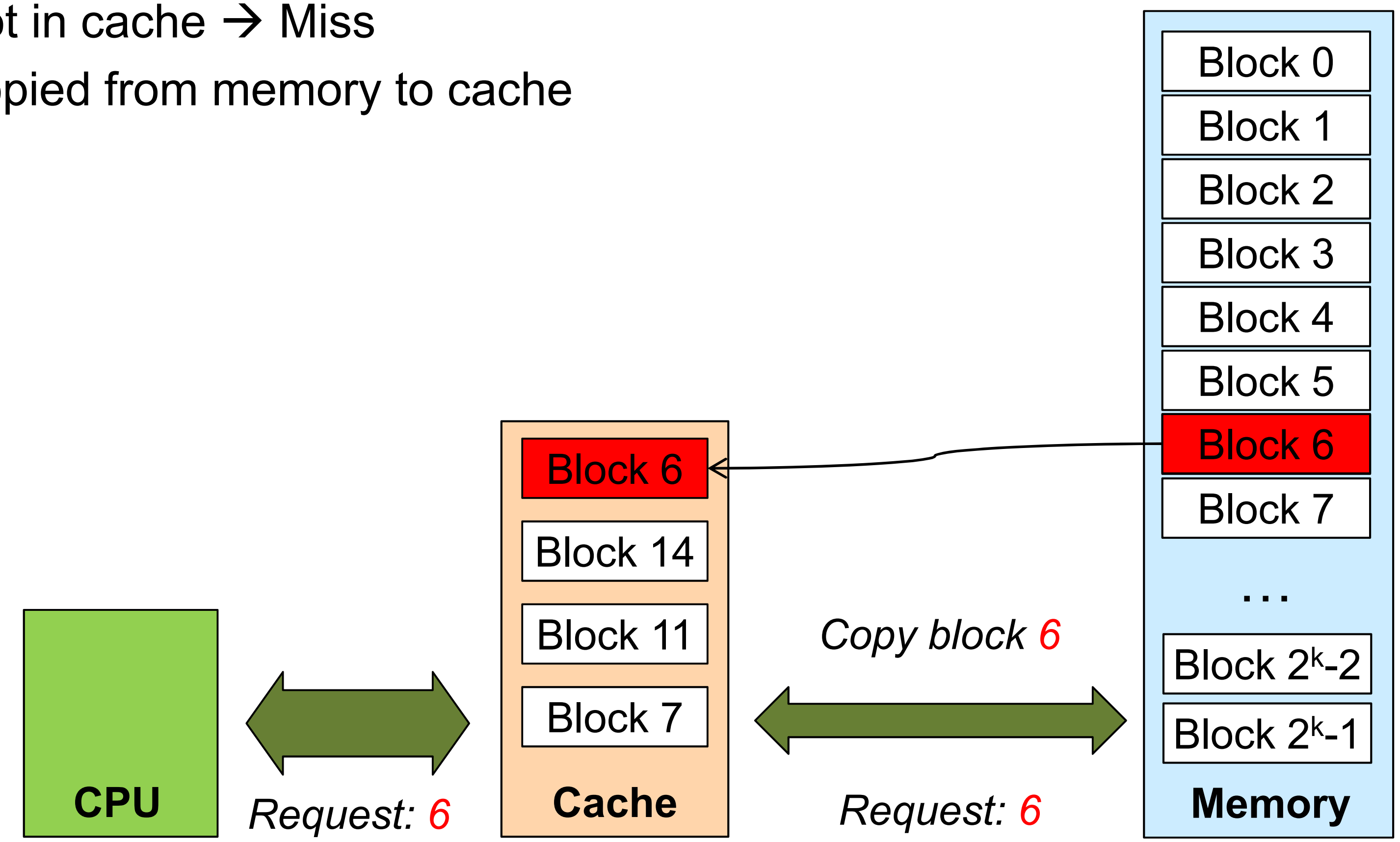
Cache Mechanics

- **Cache miss (I)**
  - Data in block 6 is needed
  - Block 6 not in cache → **Miss**



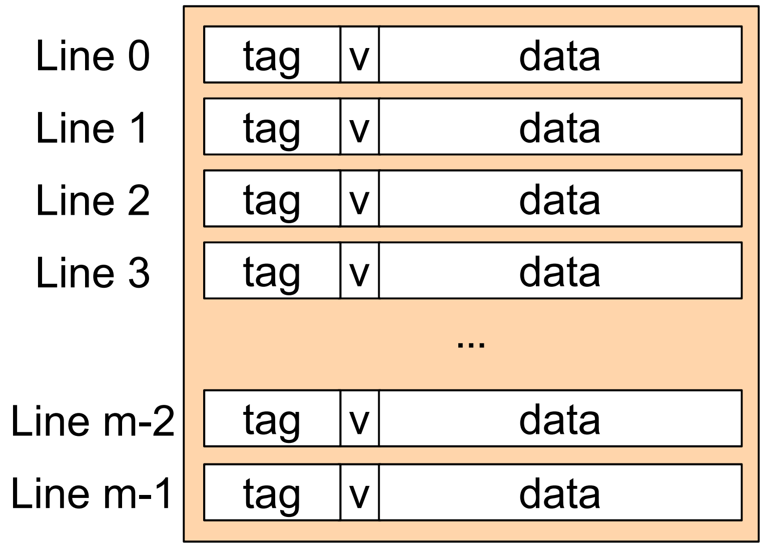
Cache Mechanics

- **Cache miss (II)**
  - Data in block 6 is needed
  - Block 6 not in cache → Miss
  - Block 6 copied from memory to cache



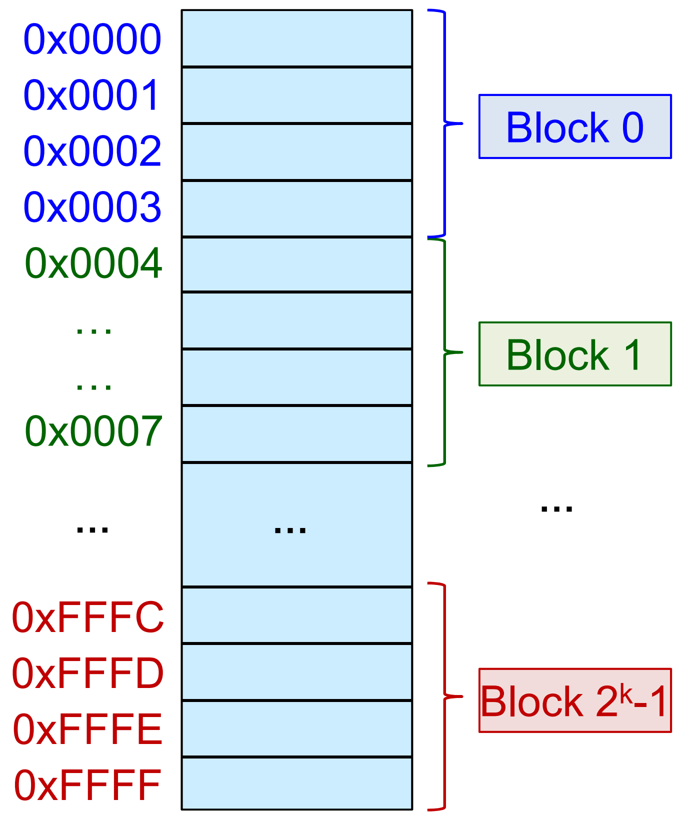
Cache Organization

- **Organized in lines**
  - Valid bit v → indicates that line contains valid data
  - Tag → unique identifier for memory location
  - Data → data of exactly one memory block
  - m = overall number of cache lines



Cache Organization

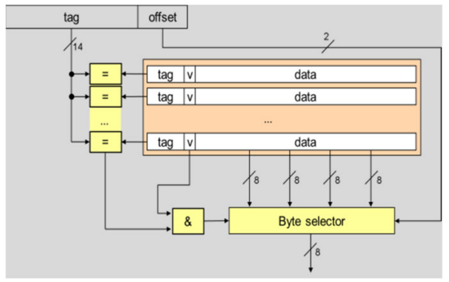
- **Addressing**

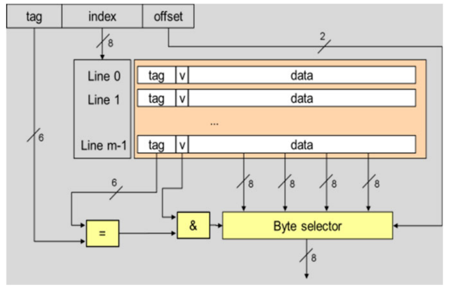


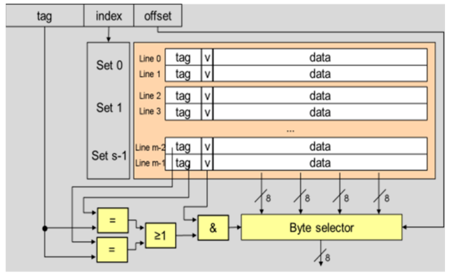
Block	Address	Block identification bits	offset
Block 0	0x0000	0000 0000 0000 00	00
	0x0001	0000 0000 0000 00	01
	0x0002	0000 0000 0000 00	10
	0x0003	0000 0000 0000 00	11
Block 1	0x0004	0000 0000 0000 01	00
	0x0005	0000 0000 0000 01	01
	0x0006	0000 0000 0000 01	10
	0x0007	0000 0000 0000 01	11
...			
Block 2k -1	0xFFFC	1111 1111 1111 11	00
	0xFFFD	1111 1111 1111 11	01
	0xFFFE	1111 1111 1111 11	10
	0xFFFF	1111 1111 1111 11	11

Cache Organization

- **Three different cache models**
  - Fully associative
  - Direct mapped
  - N-way set associative



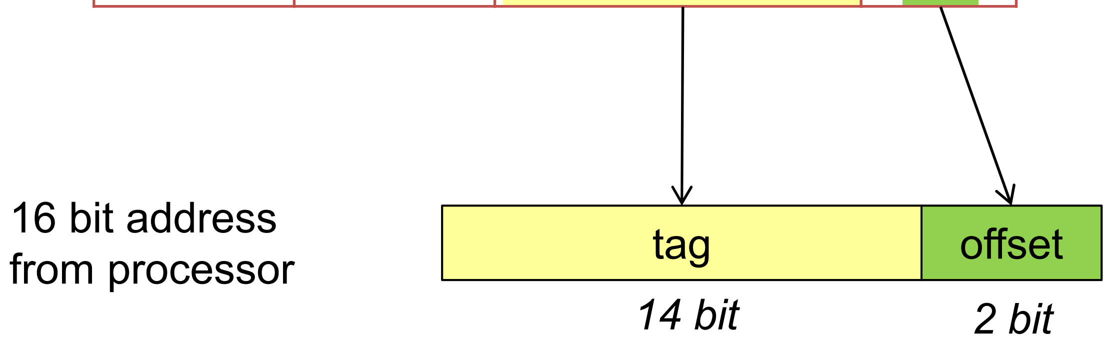




Fully Associative

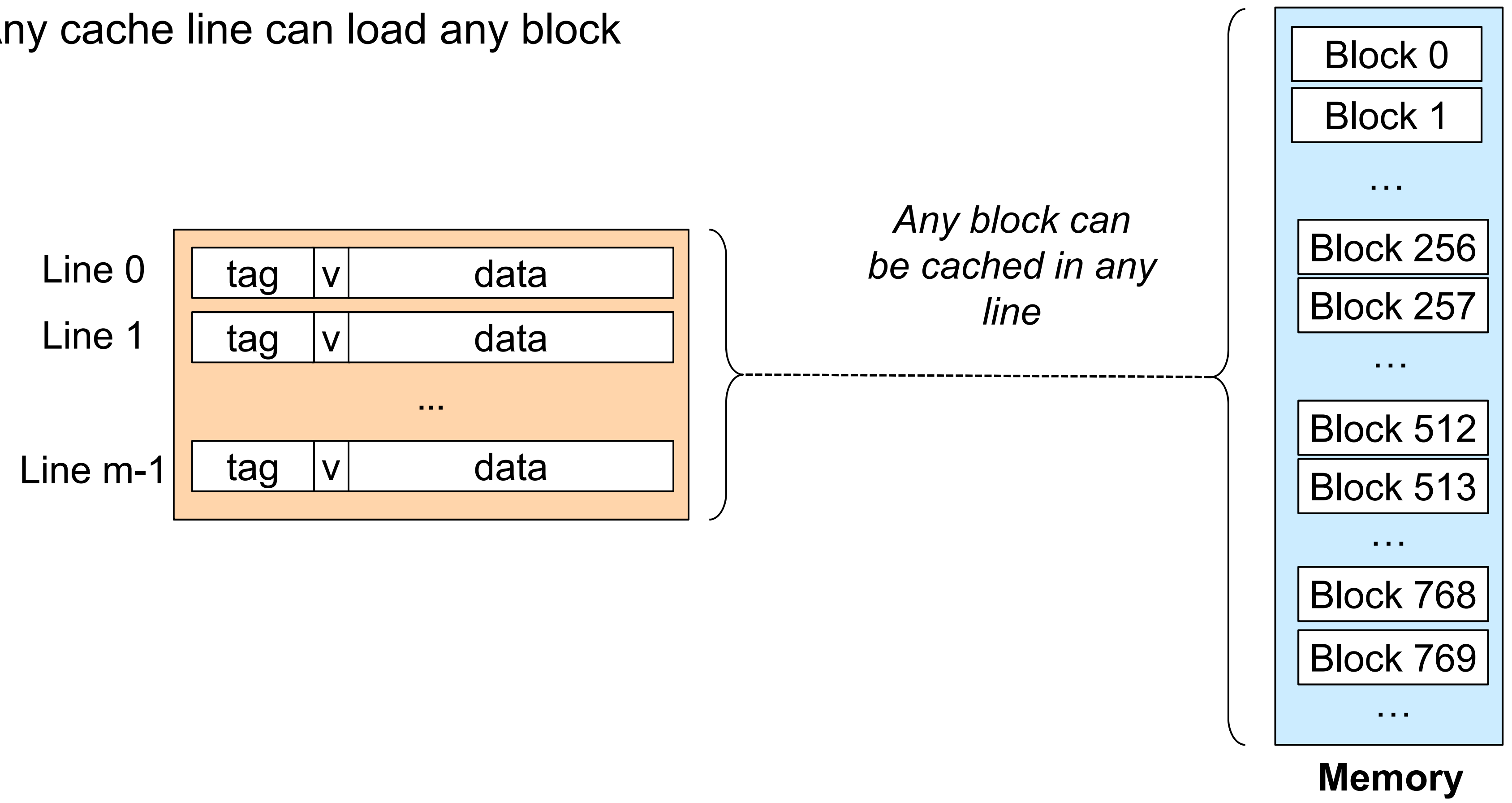
- **Addressing**
  - Tag contains complete block identification

Block	Address	Block identification bits	offset
Block 1	0x0004	0000 0000 0000 01	00
	0x0005	0000 0000 0000 01	01
	0x0006	0000 0000 0000 01	10
	0x0007	0000 0000 0000 01	11



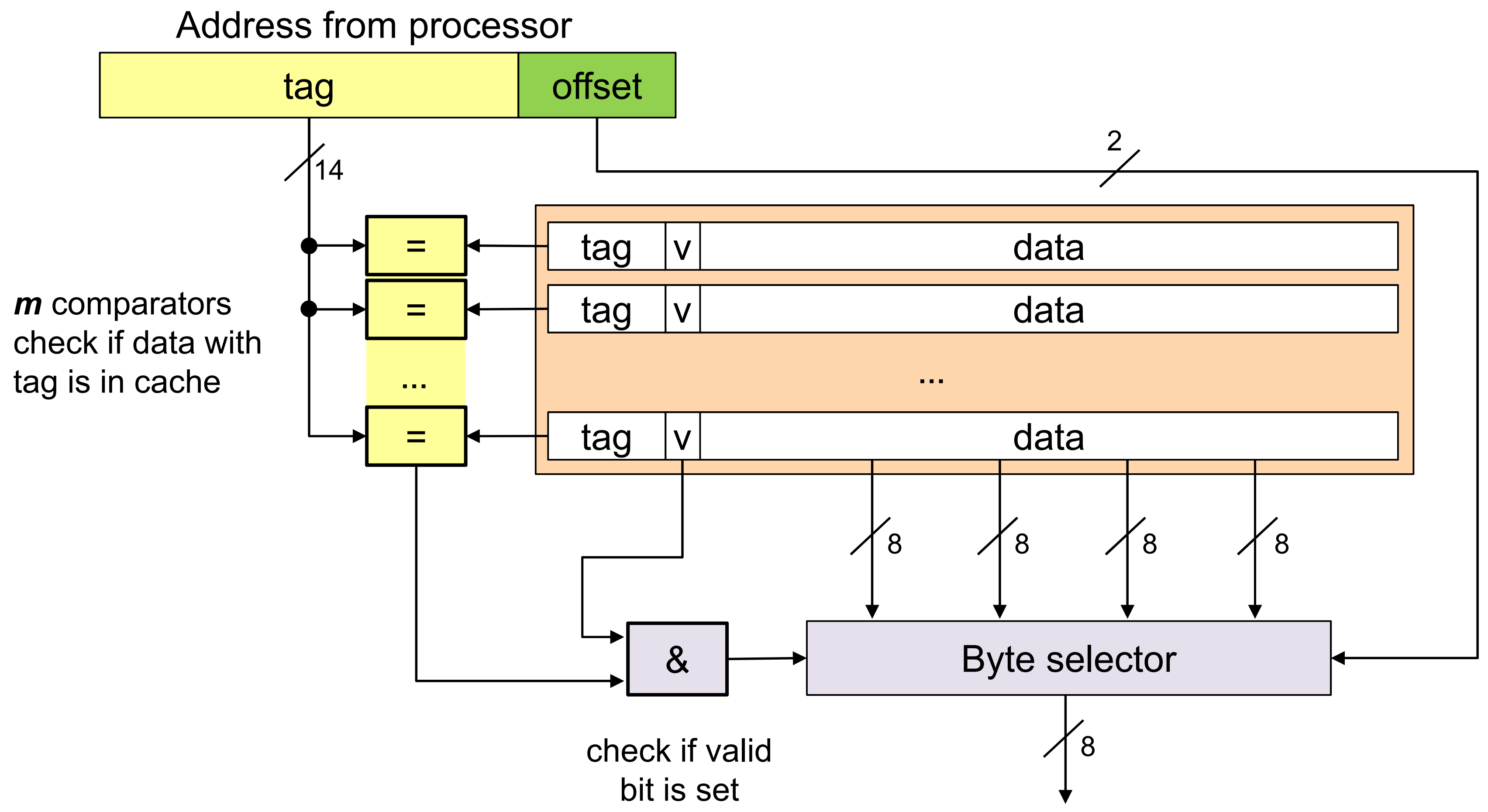
Fully Associative

- **Organization**
  - Tag contains complete block identification
  - Any cache line can load any block



Fully Associative

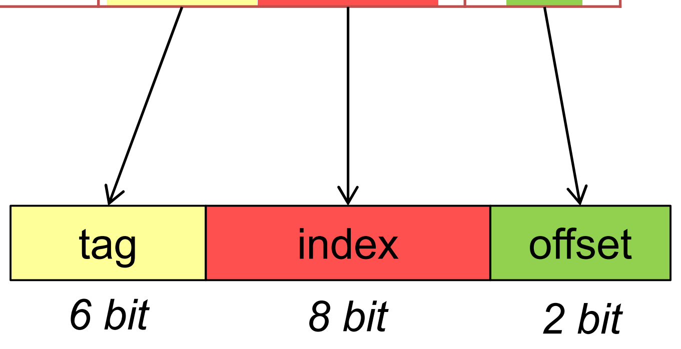
- **Architecture**



Direct Mapped

- **Addressing**
  - Block identification split into tag and index

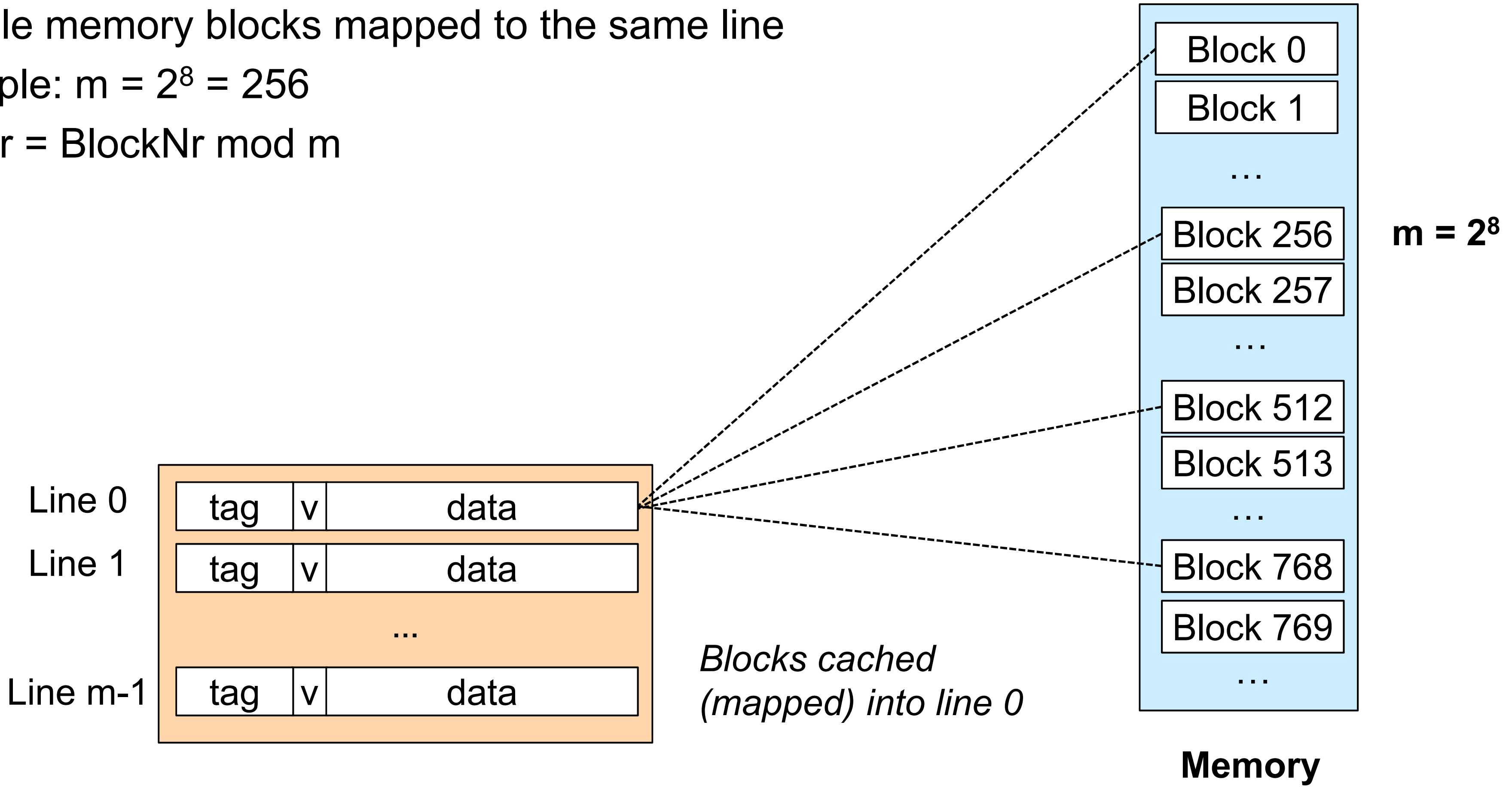
Block	Address	Block identification bits	offset
Block 1	0x0004	0000 0000 0000 01	00
	0x0005	0000 0000 0000 01	01
	0x0006	0000 0000 0000 01	10
	0x0007	0000 0000 0000 01	11



Bit sizes are an example

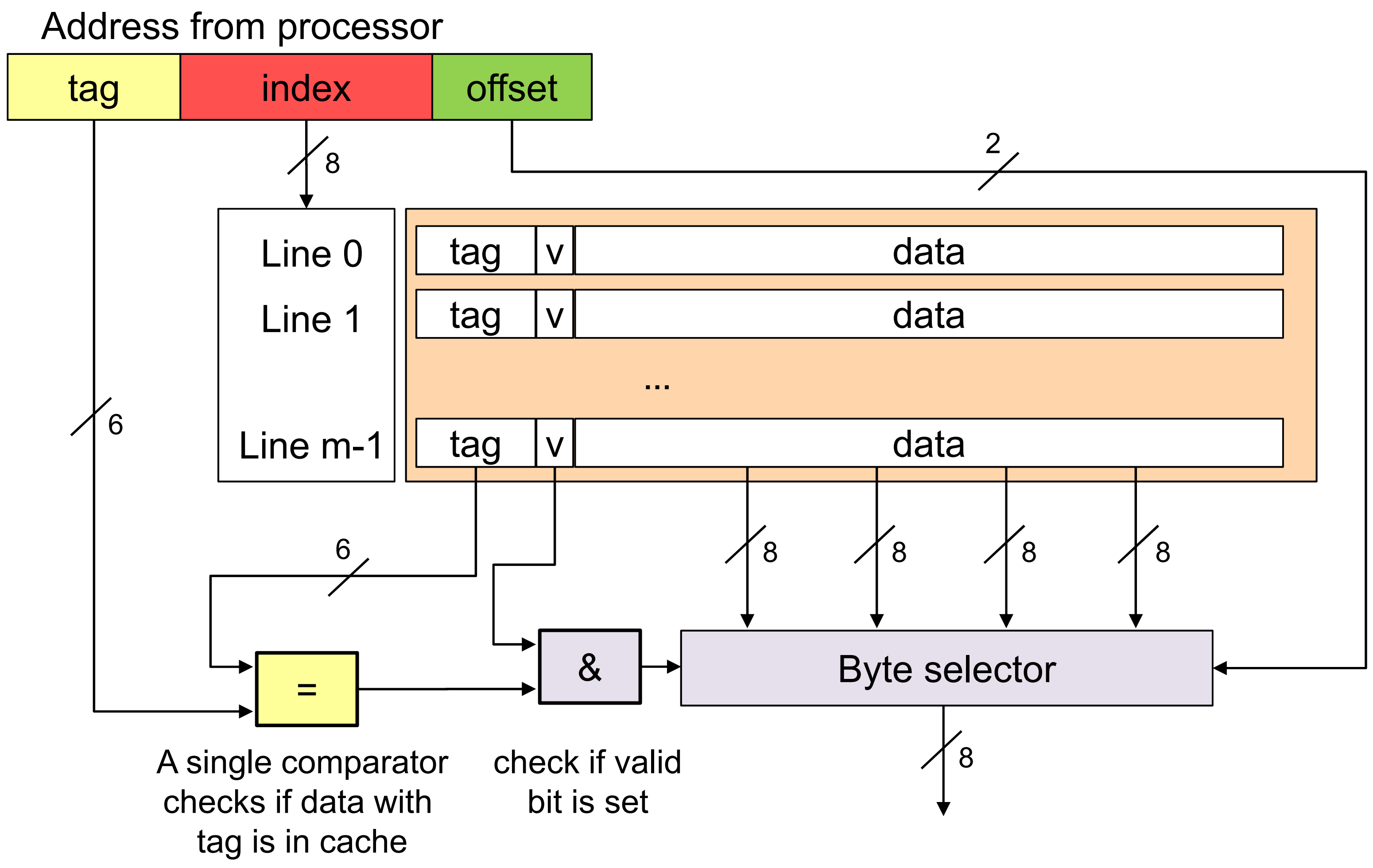
Direct Mapped

- **Organization**
  - Each memory block is mapped to exactly one cache line
  - Multiple memory blocks mapped to the same line
  - Example: m = 2^8 = 256
  - LineNr = BlockNr mod m



Direct Mapped

- **Architecture**

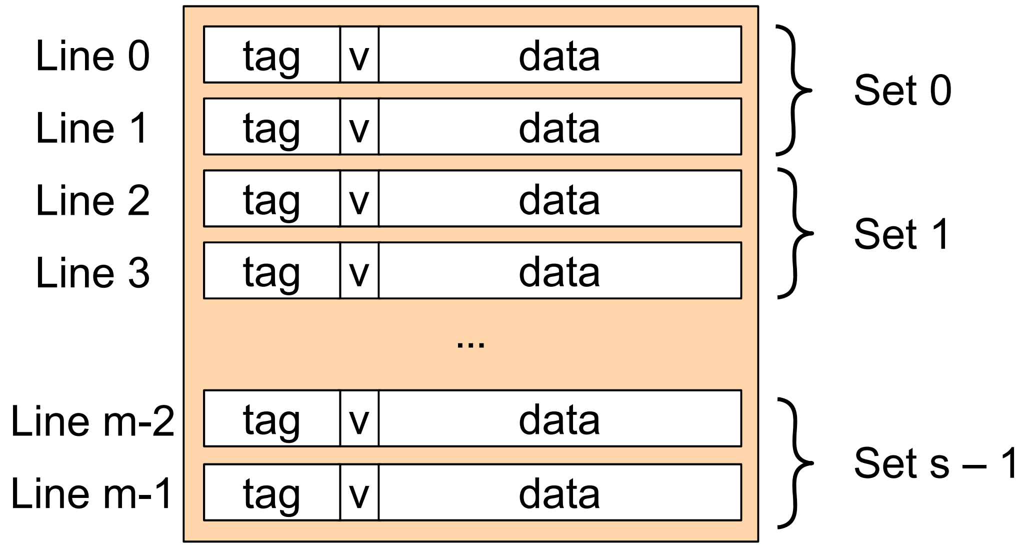


N-Way Set Associative

- **Organization**
  - Partition into sets
    - s = m/n number of sets
    - n lines per set („N-way“)
    - b Bytes per line
  - s x n x b data Bytes

N-Way Set Associative
Finding a compromise between simple logic (Direct Mapped) and high hit rates (Fully Associative)

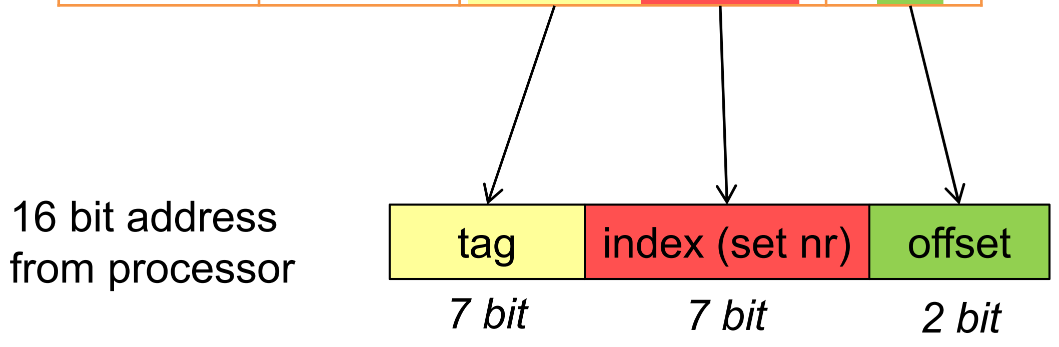
Example: k = 2



N-Way Set Associative

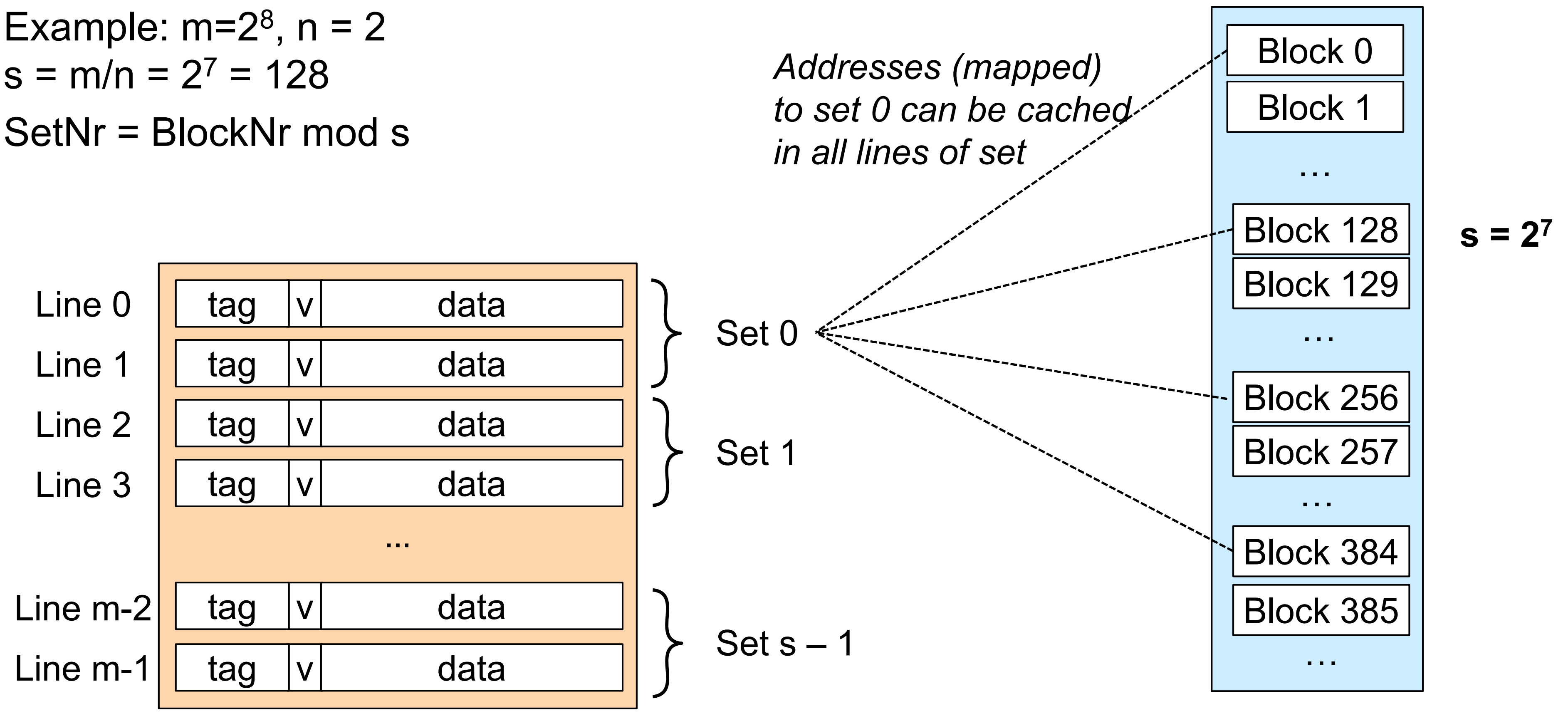
- **Addressing**
  - Example: n = 2
    - Maximum index corresponds to number of sets (s = m/n)

Block	Address	Block identification bits	offset
Block 1	0x0007	0000 0000 0000 01	11
	0x0006	0000 0000 0000 01	10
	0x0005	0000 0000 0000 01	01
	0x0004	0000 0000 0000 01	00



N-Way Set Associative

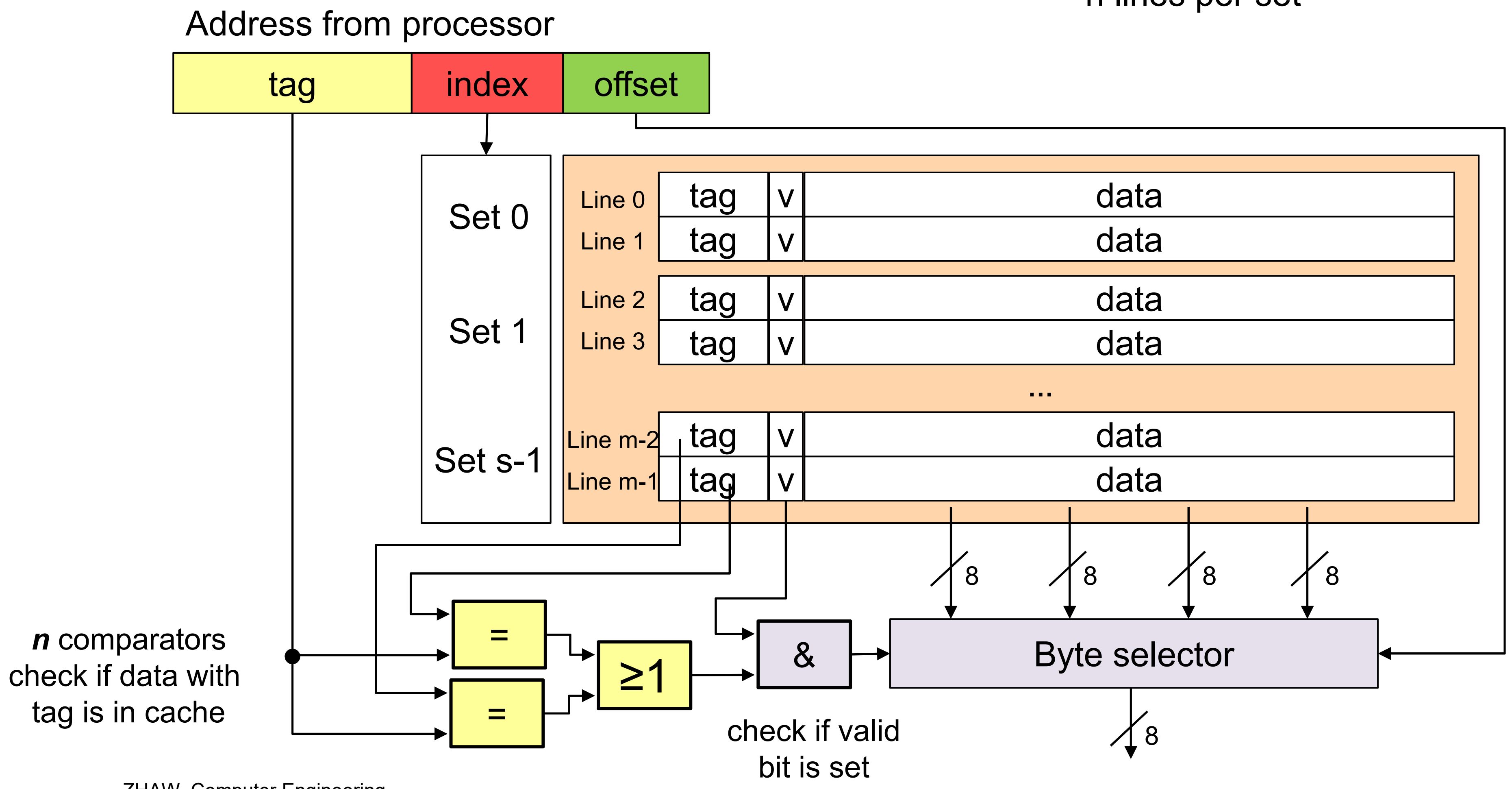
- **Organization**
  - Partition into sets
  - Example: $m=2^8$, $n = 2$
    $s = m/n = 2^7 = 128$
  - SetNr = BlockNr mod s



N-Way Set Associative

- **Architecture**

Example: n = 2
s = m/n number of sets
n lines per set



Cache Organization

- **Comparison**

Organization	Fully associative	Direct mapped	N-way set associative
Number of sets	1	m	m/n
Associativity	m(=n)	1	n
Advantages	• Fast, flexible • Highest hit rates • Advanced replacement strategies	• Simple logic • Replacement strategy defined by organization	Combination of both other concepts to combine advantages and to compensate disadvantages
Disadvantages	• Complex logic: one comparator per line • Requires large area on silicon • Replacement can be complex	• Lower hit rates	

Performance – Cache Misses

- **Cold miss**
  - First access to a block
- **Capacity miss**
  - Working set larger than cache
- **Conflict miss**
  - Multiple data objects map to same slot

Performance

- **Hit rate / miss rate**
  - Fraction of memory references found / not found in cache
    - hit rate = nr_of_hits / nr_of_accesses
    - miss rate = nr_of_misses / nr_of_accesses = 1 – hit rate
- **Hit time**
  - Time to deliver a block in the cache to the processor
- **Miss penalty**
  - Additional time required to fetch data from memory because of a cache miss

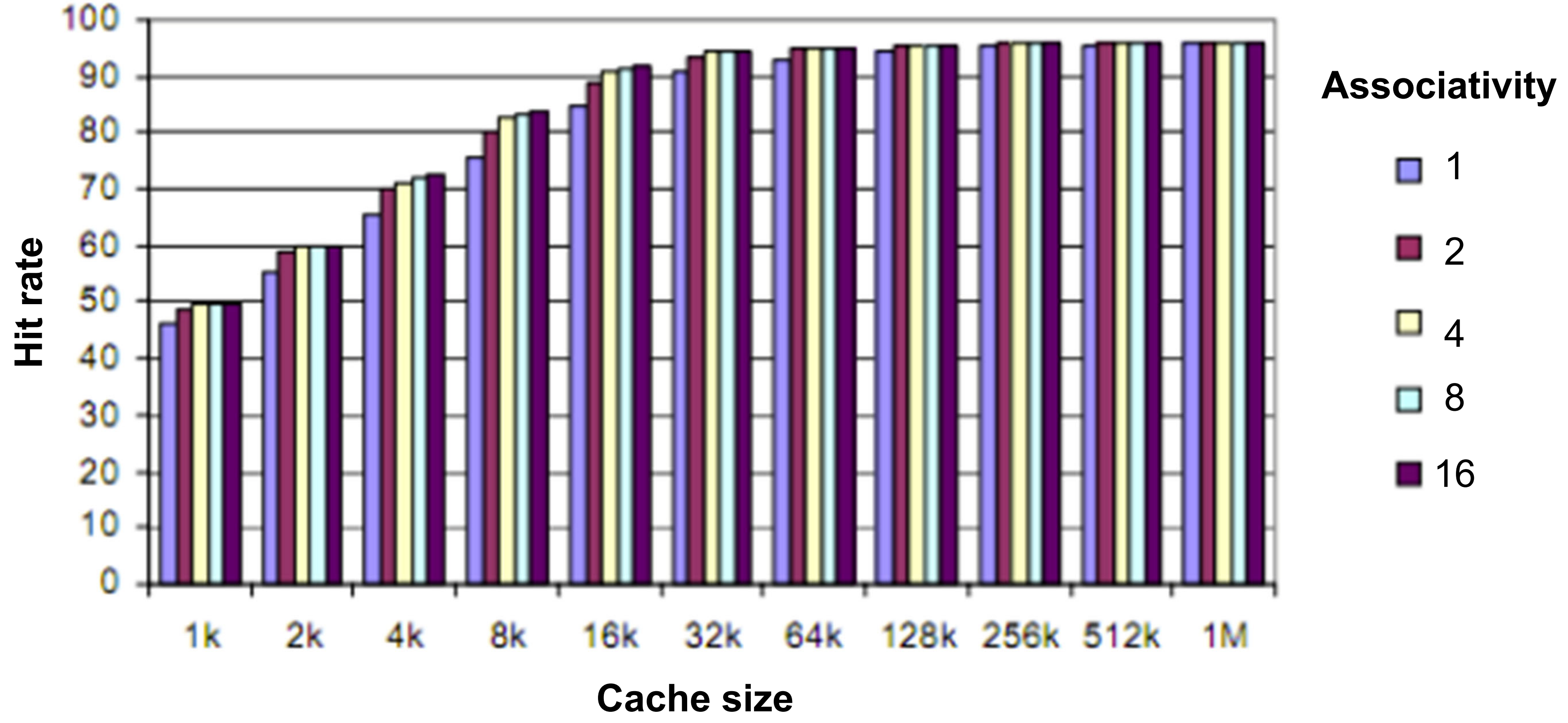
Performance

- **High cache hit rate is important!**
  - 99% hit rate can be twice as fast as 97%

- **Example**
  - Cache hit time: 1 processor cycle
  - Miss penalty: 100 processor cycles
  - Average access time is:
    - 97% hits: 1 cycle + 0.03 * 100 cycles = 4 cycles average
    - 99% hits: 1 cycle + 0.01 * 100 cycles = 2 cycles average

Performance

- **Cache size versus hit rate**
  - Typical hit rate for Associativity 1, 2, 4, 8, 16 and cache size



http://www.extremetech.com/extreme/188776-how-l1-and-l2-cpu-caches-work-and-why-theyre-an-essential-part-of-modern-chips

Replacement Strategies

- **Selecting cache line to replace by**
  - LRU: Least recently used
  - LFU: Least frequently used
  - FIFO: First In–First Out → oldest
  - Random Replace: randomly chosen

$\left.\begin{array}{l}\text{LRU}\\ \text{LFU}\\ \text{FIFO}\end{array}\right\}$ *additional information needed in cache*

$\left.\text{Random Replace}\right\}$ *simple, but still good performance*

→ Only relevant for Fully- / N-way associative caches
→ Hard coded in cache implementation (=hardware)

Write Strategies

- **What to do on a write hit*?**
  - Write-through
    - Write immediately to memory
  - Write-back
    - Delay write to memory until replacement of line (needs a valid bit)
- **What to do on a write miss**?**
  - Write-allocate
    - Load line into cache (from memory) and update line in cache
  - No-write-allocate
    - Writes immediately to memory

* *Write hit: data already in cache*
** *Write miss: data not in cache*

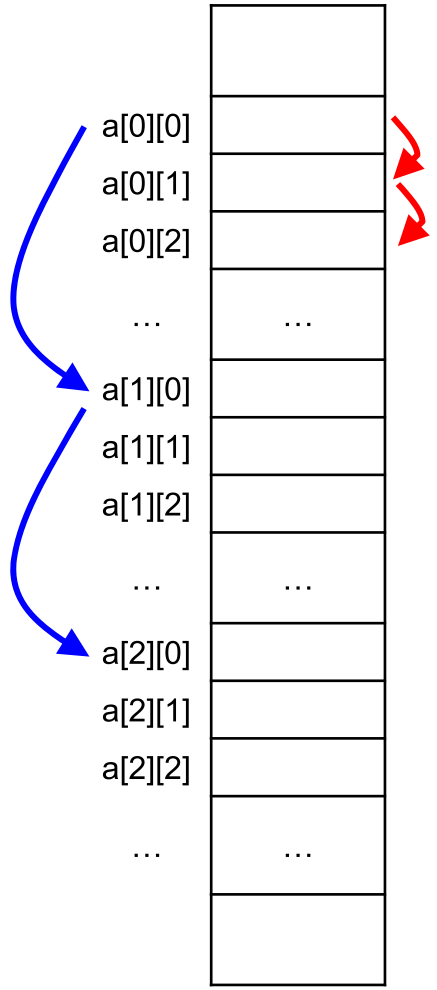
Animation

The Programmer’s Perspective

- **For loops over multi-dimensional arrays**
  - Example: matrices (2-dim arrays)
- **Change order of iteration to match layout**
  - Gets better spatial locality
  - Layout in C: last index changes first!

```
for(j = 0; j < 10000; j++){
  for(i = 0; i < 40000; i++){
    c[i][j] = a[i][j] + b[i][j];
  }
}

// a[i][j] and a[i+1][j]
// are 10’000 elements apart
```

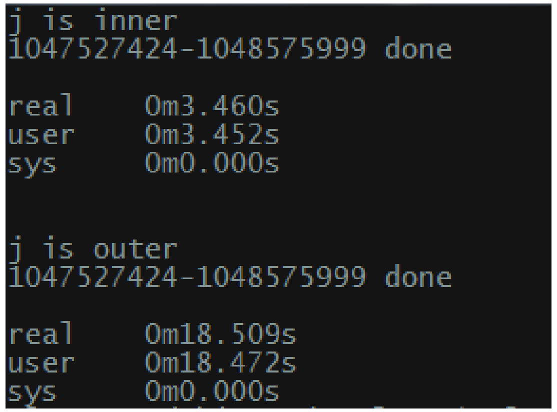
```
for(i = 0;i < 40000; i++){
  for(j = 0;j < 10000; j++){
    c[i][j] = a[i][j] + b[i][j];
  }
}

// a[i][j] and a[i][j+1]
// are next to each other
```



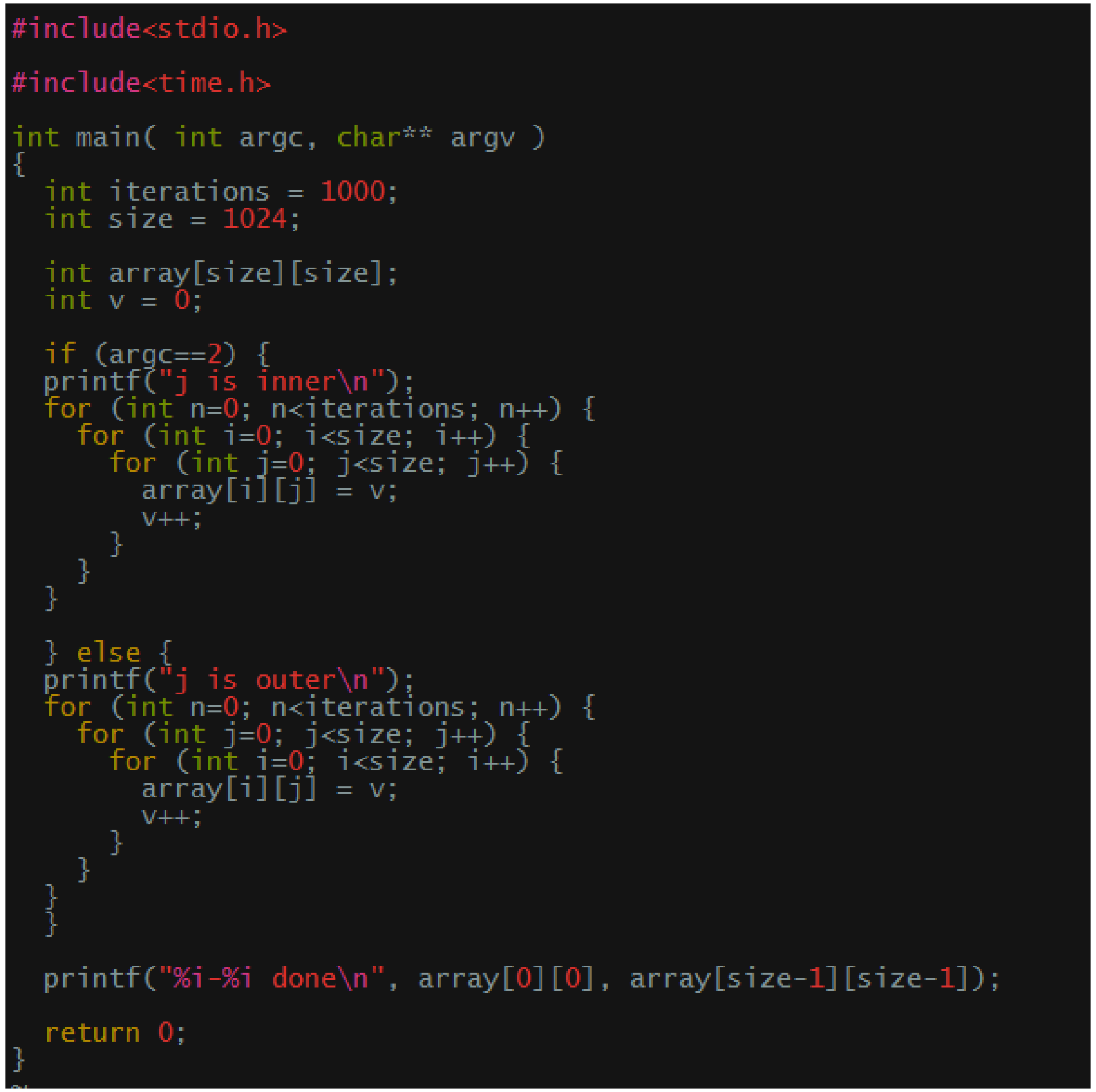
Animation

The Programmer's Perspective

- **Change order of iteration to match layout**



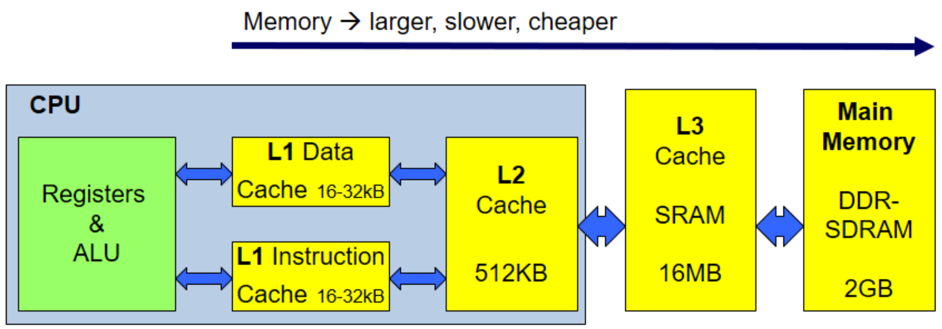
```
j is inner
1047527424-1048575999 done

real    0m3.460s
user    0m3.452s
sys     0m0.000s


j is outer
1047527424-1048575999 done

real    0m18.509s
user    0m18.472s
sys     0m0.000s
```

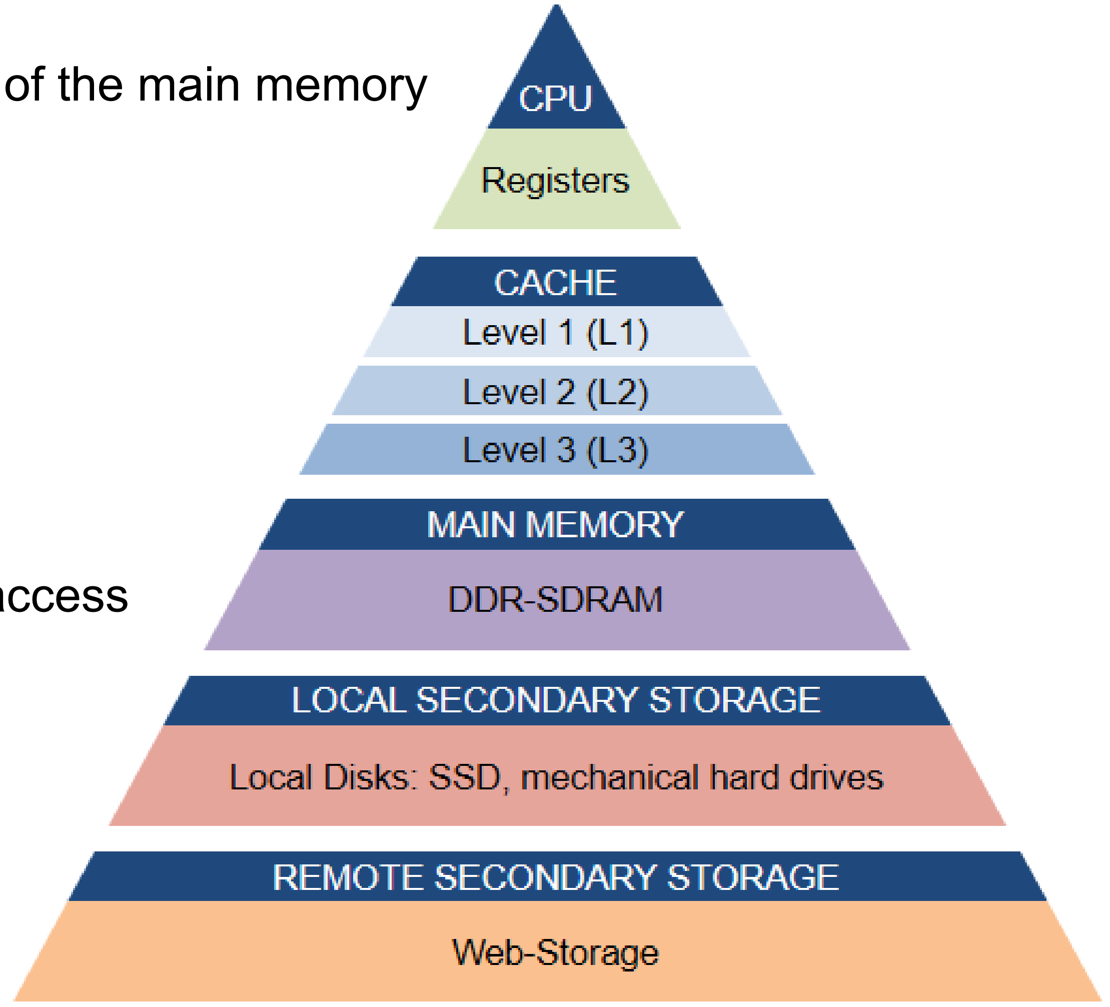


```
#include<stdio.h>

#include<time.h>

int main( int argc, char** argv )
{
  int iterations = 1000;
  int size = 1024;

  int array[size][size];
  int v = 0;

  if (argc==2) {
  printf("j is inner\n");
  for (int n=0; n<iterations; n++) {
    for (int i=0; i<size; i++) {
      for (int j=0; j<size; j++) {
        array[i][j] = v;
        v++;
      }
    }
  }

  } else {
  printf("j is outer\n");
  for (int n=0; n<iterations; n++) {
    for (int j=0; j<size; j++) {
      for (int i=0; i<size; i++) {
        array[i][j] = v;
        v++;
      }
    }
  }
  }

  printf("%i-%i done\n", array[0][0], array[size-1][size-1]);

  return 0;
}
```

Conclusion

- **Cache**
  - Cache allows fast data access to selected parts of the main memory which are mirrored in the cache
    - Spatial and temporal locality
  - Different cache models
    - Fully associative
    - Direct mapped
    - N-way associative
  - Replacement / Write strategies
  - Reducing cache misses by optimizing memory access


Memory → larger, slower, cheaper
CPU
Registers & ALU
L1 Data Cache 16-32kB
L1 Instruction Cache 16-32kB
L2 Cache 512KB
L3 Cache SRAM 16MB
Main Memory DDR-SDRAM 2GB



Driver for Cache and Pipelining

ACAT2013 IOP Publishing

Journal of Physics: Conference Series **523** (2014) 012002 doi:10.1088/1742-6596/523/1/012002

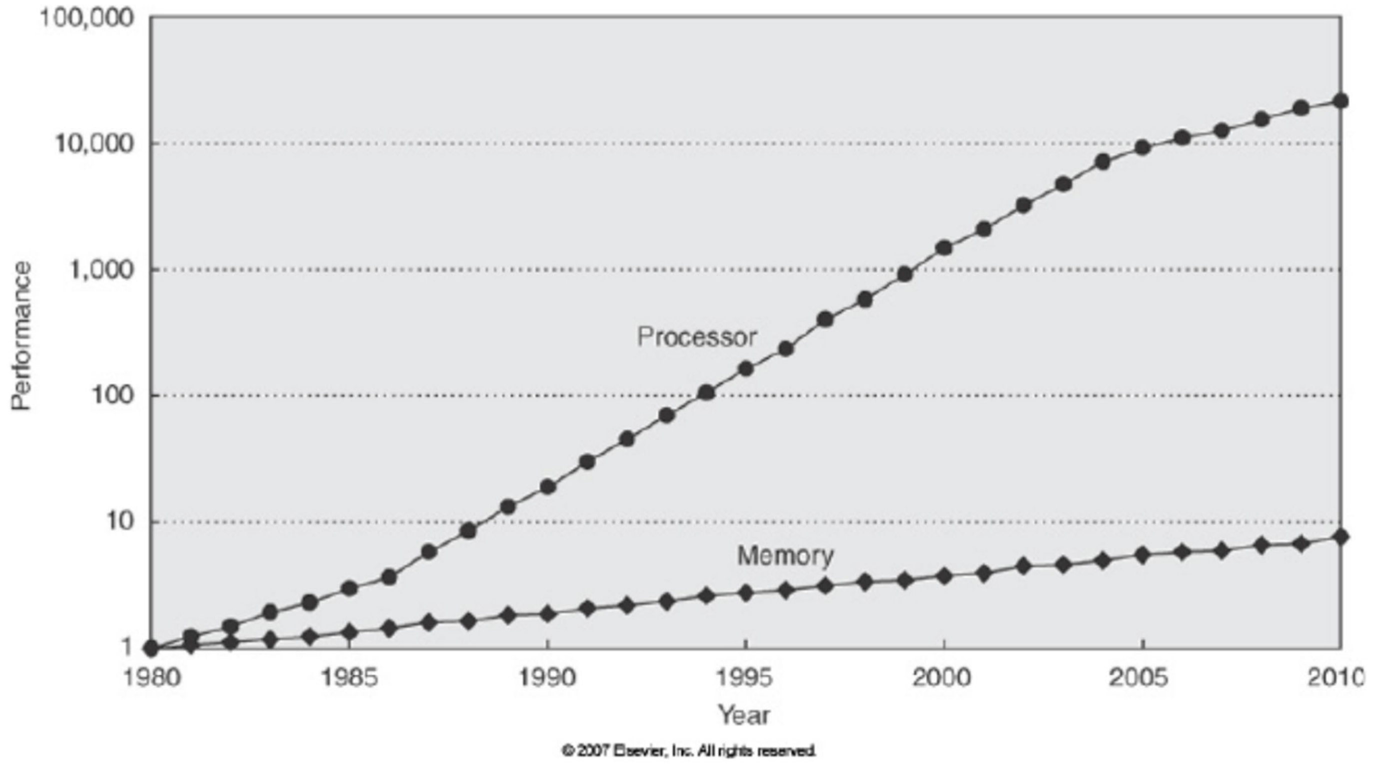


Figure 1: CPU-memory performance gap. Modelled after "Computer Architecture": Hennessy, John L.; Patterson, David A.

https://www.researchgate.net/publication/273029990_Opportunities_and_choice_in_a_new_vector_era

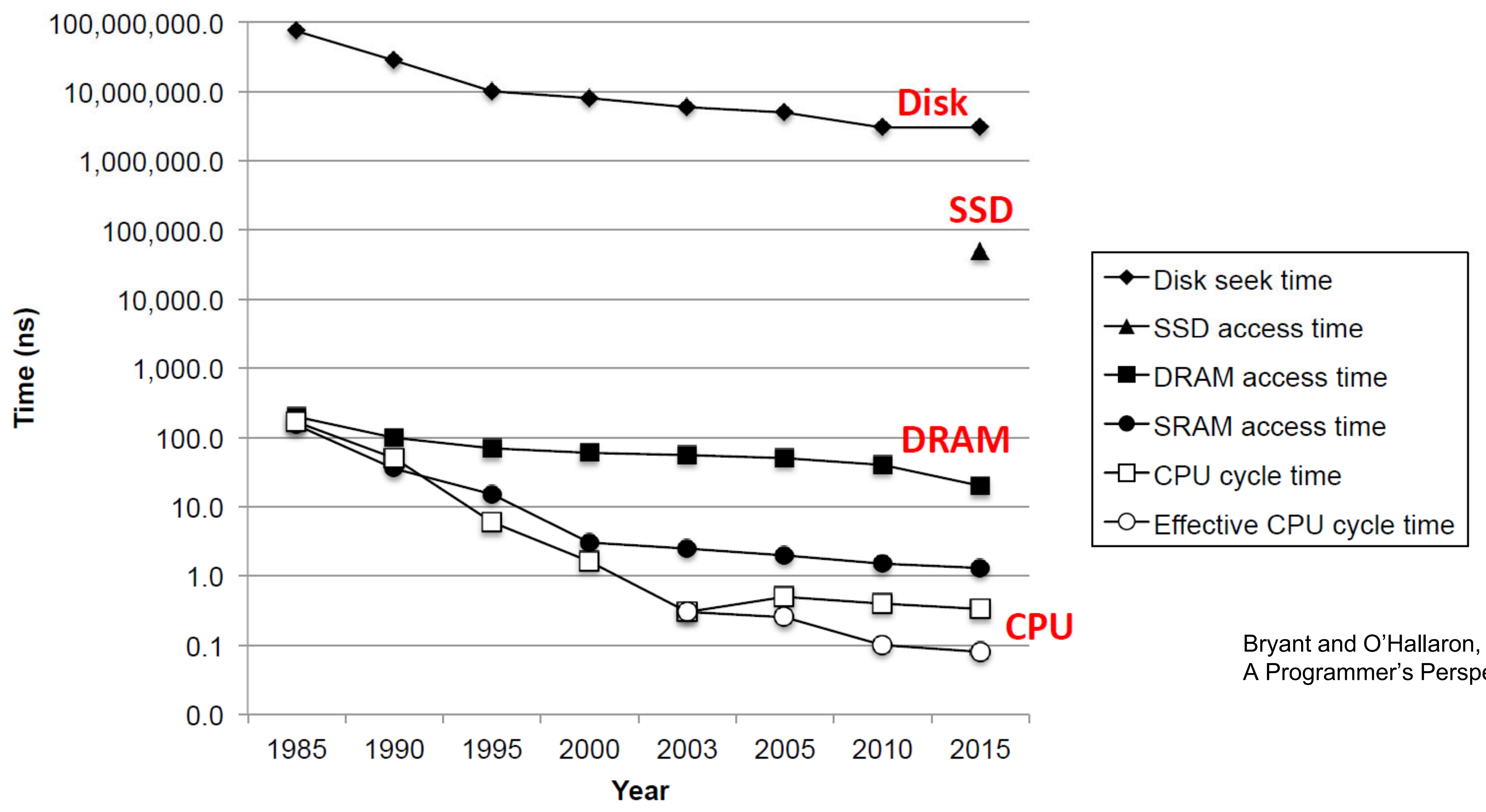
Cache Architecture Comparison

	Characteristic	ARM Cortex-A8	Intel Nehalem
L1	LI cache organization	Split instruction and data caches	Split instruction and data caches
	LI cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
	LI cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
	LI replacement	Random	Approximated LRU
	LI block size	64 bytes	64 bytes
	LI write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
	LI hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2	L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
	L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
	L2 cache associativity	8-way set associative	8-way set associative
	L2 replacement	Random(?)	Approximated LRU
	L2 block size	64 bytes	64 bytes
	L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
	L2 hit time	11 clock cycles	10 clock cycles
L3	L3 cache organization		Unified (instruction and data)
	L3 cache size		8 MiB, shared
	L3 cache associativity		16-way set associative
	L3 replacement		Approximated LRU
	L3 block size		64 bytes
	L3 write policy		Write-back, Write-allocate
	L3 hit time		35 Clock Cycles

Source: Patterson / Hennesey: Computer Organization and Design, 5th edition, Elsevier

The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.



Bryant and O'Hallaron, Computer Systems:
A Programmer's Perspectve, Third Editon