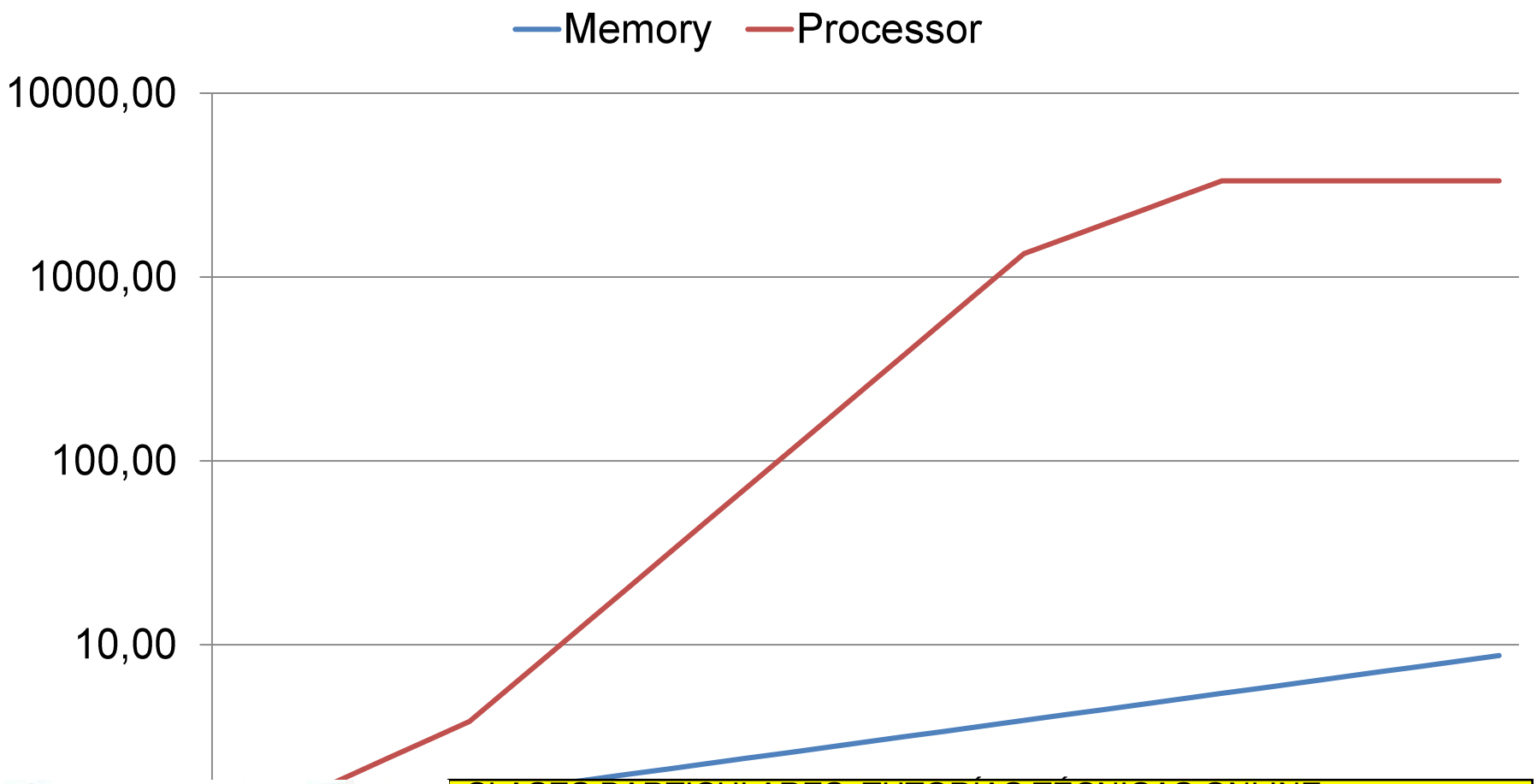
COMPUTER ARCHITECTURE

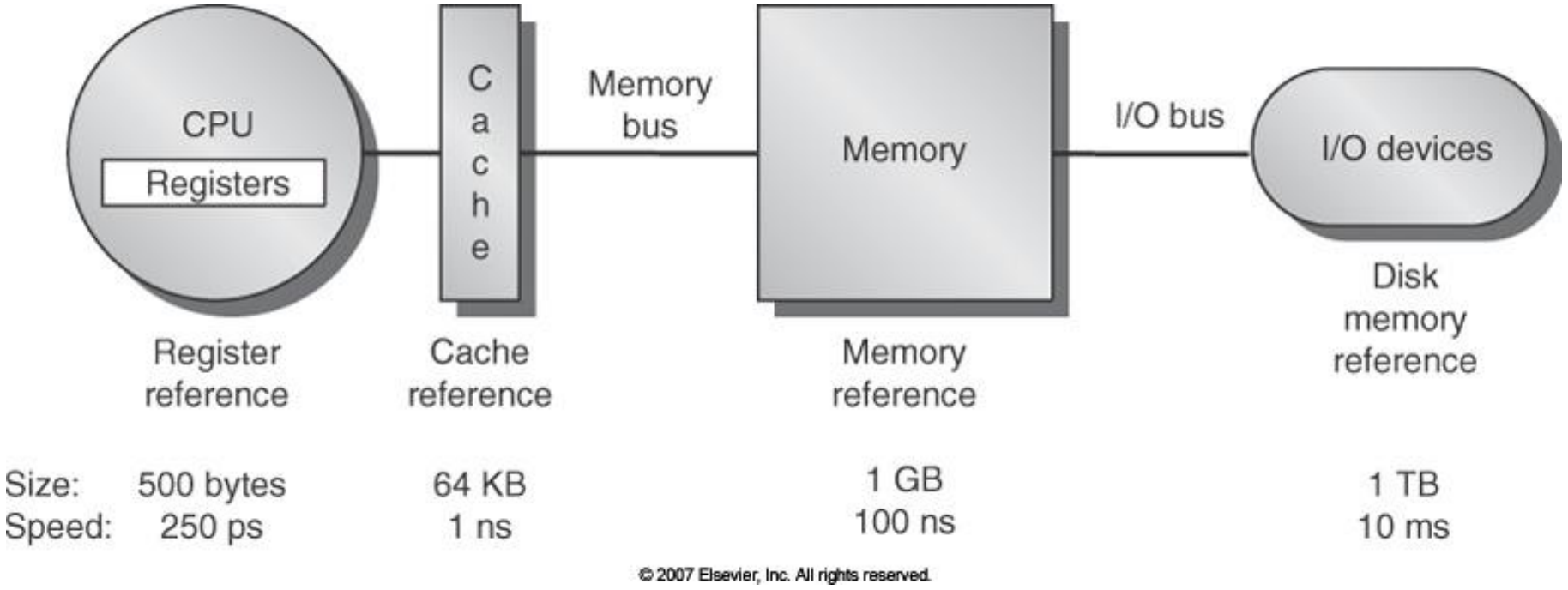
Advanced Cache Optimizations

Universidad Carlos III de Madrid
www.uc3m.es

Why caching?



Cache memory within memory hierarchy



Average access time to memory

- Level 1
  - Hit time + Miss rate x Miss Penalty
- Level 2
  - Hit time$_{L1}$ + Miss rate $_{L1}$ x (Hit time $_{L2}$ + Miss rate $_{L2}$ x Miss penalty $_{L2}$)
- …

Basic optimizations

- Larger block size.
- Larger cache size.
- Higher associativity.
- Multilevel caches.
- Prioritize read misses.
- Avoid address translation during indexing.

Metrics for advanced optimizations

- Metrics to be reduced:
  - **Hit time.**
  - **Miss rate.**
  - **Miss penalty.**
  - **Cache bandwidth.**

- All advanced optimizations try to improve some of this metrics.

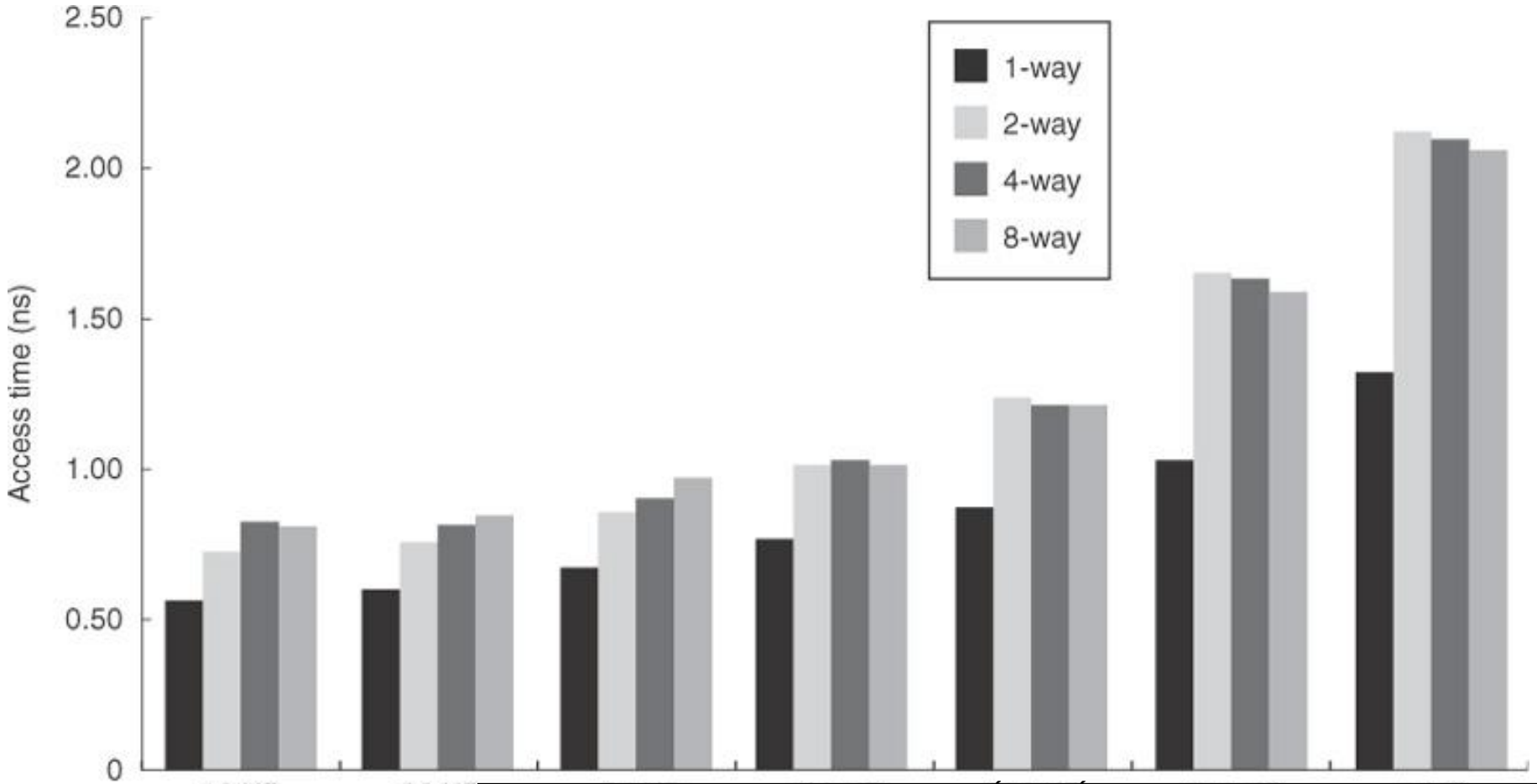
Advanced optimizations

- Small and simple caches.
- Way prediction.
- Pipelined access to caches.
- Non-blocking caches.
- Multi-bank caches.
- Critical word first and early restart.
- Write buffer merge.
- Compiler optimizations.
- Data and instruction hardware prefetching.

- **Lookup:**
  - Line selection using **index**.
  - Read **tag** from line.
  - Compare **tag** and **address** (tag field).

- **Lookup time** increases with **cache size**.
  - Lookup hardware simpler with smaller cache.
  - Cache fits into processor chip.

Access time and cache size



1: Smaller and simpler caches

- **Cache simplification.**
  - Using mapping mechanisms as simple as possible.
  - Direct mapping:
    - Allows to parallelize tag comparison and data transmission.

- Most modern processors more focused in using small caches than in simplifying them.

Intel Core i7

- L1 cache (1 per core)
  - 32 KB instructions
  - 32 KB data
  - Latency: 3 cycles
  - Associative 4(i), 8(d) ways.
- L2 cache (1 per core)
  - 256 KB
  - Latency: 9 cycles
  - Associative 8 ways.
- L3 cache (shared)
  - 8 MB
  - Latency: 39 cycles

2: Way prediction

- **Problem:**
  - Direct mapping → fast but many misses.
  - Set associative → less misses but slower.

- **Way prediction:**
  - Store additional bits to predict way to be selected in next access.
  - Block prefetching and compare to single tag.
    - On miss compare to other tags.

3: Pipelined access to cache

- **Goal:** Increase cache bandwidth.
- **Solution:** Pipeline cache access in several clock cycle.
- **Effects:**
  - Clock cycle can be shortened.
  - An access may be initiated in every clock cycle.
  - Cache bandwidth increases.
  - Latency increases.

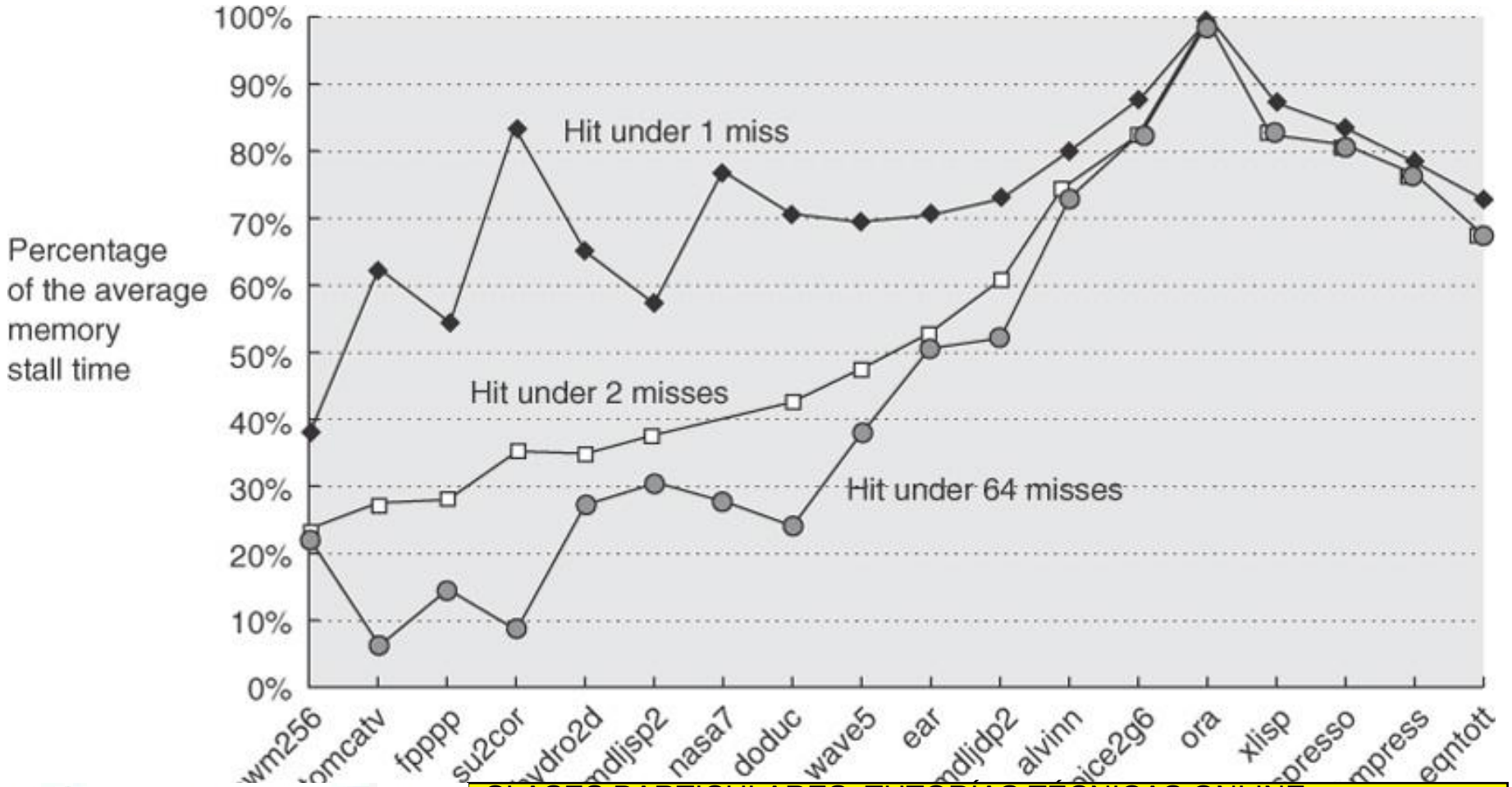
4: Non blocking caches

- **Problem:** Cache miss leads to stall until block is obtained.
- **Solution:** Out of order execution.
  - How do we access to cache while miss is solved?
- **Hit during miss.**
  - Allow access with hit while waiting.
  - Reduces miss penalty.
- **Hit during several misses / Miss during miss.**
  - Allow overlapped misses.

Universidad Carlos III de Madrid
www.uc3m.es

Stall rate versus maximum number of misses

ARCOS



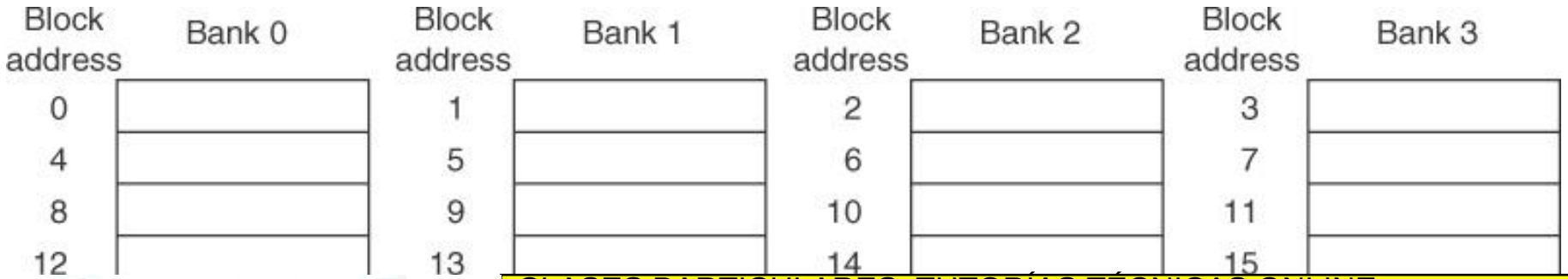
5: Multi-bank caches

- **Goal:** Allow simultaneous accesses to different cache locations.
- **Solution:** Divide memory into independent banks.
- **Effect:** Increases bandwidth.
- **Example:** Sun Niagara
  - L2: 4 banks

5: Multi-bank caches

- To **improve bandwidth** it is necessary that accesses are **distributed across banks**.

- **Simple approach:** Sequential interleaving.
  - Round-robin of blocks across banks.



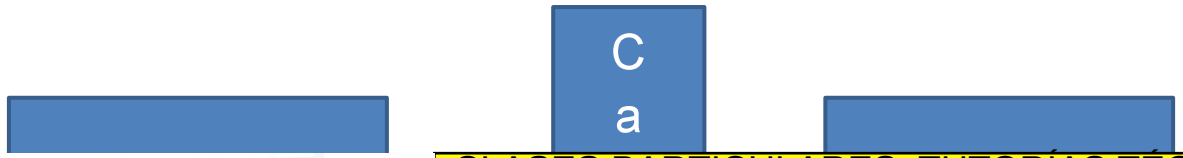
6: Critical word first and early restart

- **Observation:** Processor usually needs a single word to proceed.
- **Solution:** Do not wait to get the whole block from memory.
- **Alternatives:**
  - **Critical word first:** Block received ordered by word needed by processor first.
  - **Early restart:** Block received without reordering.
    - As soon as word of interest is received, CPU proceeds.

7: Write buffer merge

- Write buffer allows to decrease miss penalty.
  - When data written to buffer processor considers write done.
  - Simultaneous writes on memory more efficient than single write.

- **Uses:**
  - **Write-through:** In every write.
  - **Write-back:** When block is replaced



7: Write buffer merge

- When buffer contains modified blocks, check addresses and overwrite if possible.

- **Effects:**
  - Reduces number of memory writes.
  - Reduces stalls due to full buffer.

7: Write buffer merge

Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	

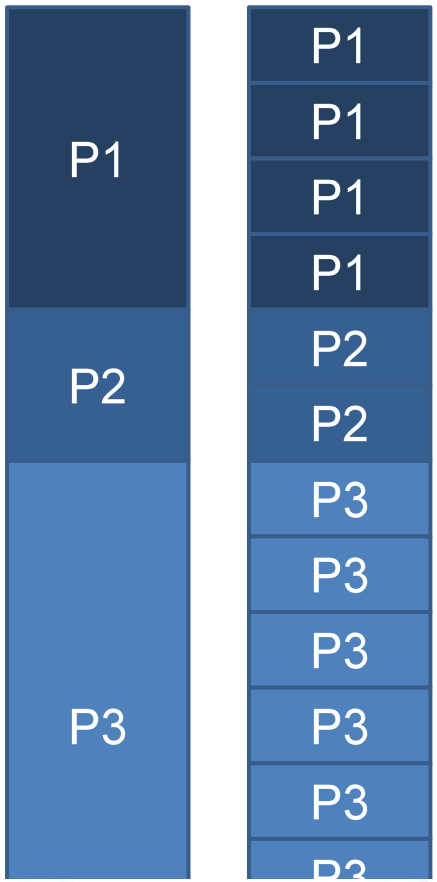
8: Compiler optimizations

- **Goal**: Generate code leading to less instruction and data misses.

- **Instructions:**
  - Procedure reordering.
  - Align code blocks to cache line start.
  - Linearization.

- **Data:**
  - Array merge.
  - Loop exchange.
  - Loop merge.

8.1: Procedure reordering



- **Goal:**
  - Reduce conflict misses due to two procedures overlapping in time and mapped to the same cache line.
- **Technique:**
  - Reorder procedures in memory.

8.2: Basic block alignment

- **Definition:** A basic block is a set of instructions sequentially executed (no branches contained).
- **Goal:** Reduce cache misses possibility for sequential code.
- **Technique:** Align first instructions in basic block to first word in cache line.

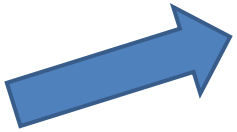
8.3: Linearization

- **Goal:** Reduce cache misses due to conditional branching.

- **Technique:** If compiler knows it is likely to take a branch, it can invert the condition and interchange both alternatives basic blocks.
  - Most likely basic block likely not to cause a miss.

8.4: Array merging


```
vector<int> key{max};
vector<int> value{max};
…
for (int i=0;i<max;++i) {
  cout << key[i] << " , "
    << value[i] << endl;
}
```



```
struct entry {
  int key;
  int value;
};
vector<entry> v{max};
…
for (auto & e : v) {
  cout << e.key << " , "
    << e.value << endl;
}
```

Conflict reduction

Improve spatial locality

8.5 Loop interchange


```
for (j=0; j<100; j++) {
  for (i=0;i<5000; i++) {
    v[i][j] = 2 * v[i][j];
  }
}
```

Striped accesses

```
for (i=0; i<5000; j++) {
  for (j=0;i<100; i++) {
    v[i][j] = 2 * v[i][j];
  }
}
```

Sequential accesses

Better spatial locality

8.6: Loop merge


```
for (i=0; i<N; i++) {
  for (j=0;j<N; j++) {
    a[i][j] = b[i][j] * c[i][j];
  }
}
for (i=0; i<N; i++) {
  for (j=0;j<N; j++) {
    d[i][j] = a[i][j] + c[i][j];
  }
}
```

```
for (i=0; i<N; i++) {
  for (j=0;j<N; j++) {
    a[i][j] = b[i][j] * c[i][j];
    d[i][j] = a[i][j] + c[i][j];
  }
}
```

Improves temporal locality

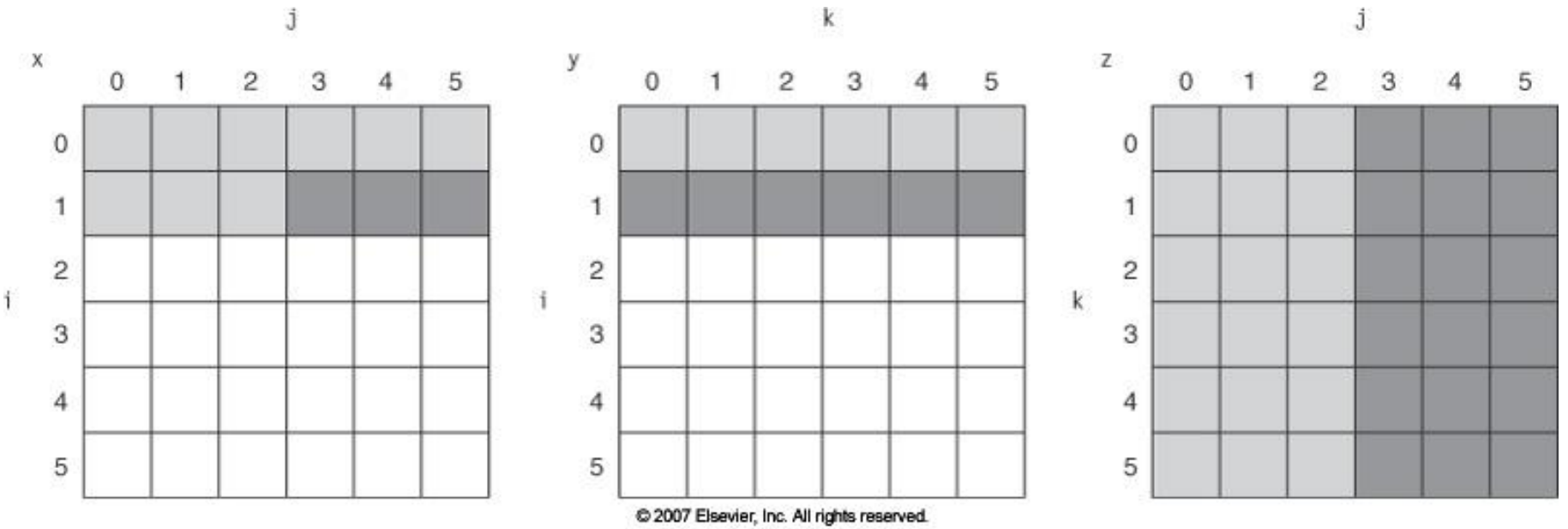
Beware: It could reduce spatial

8.7: Blocked access

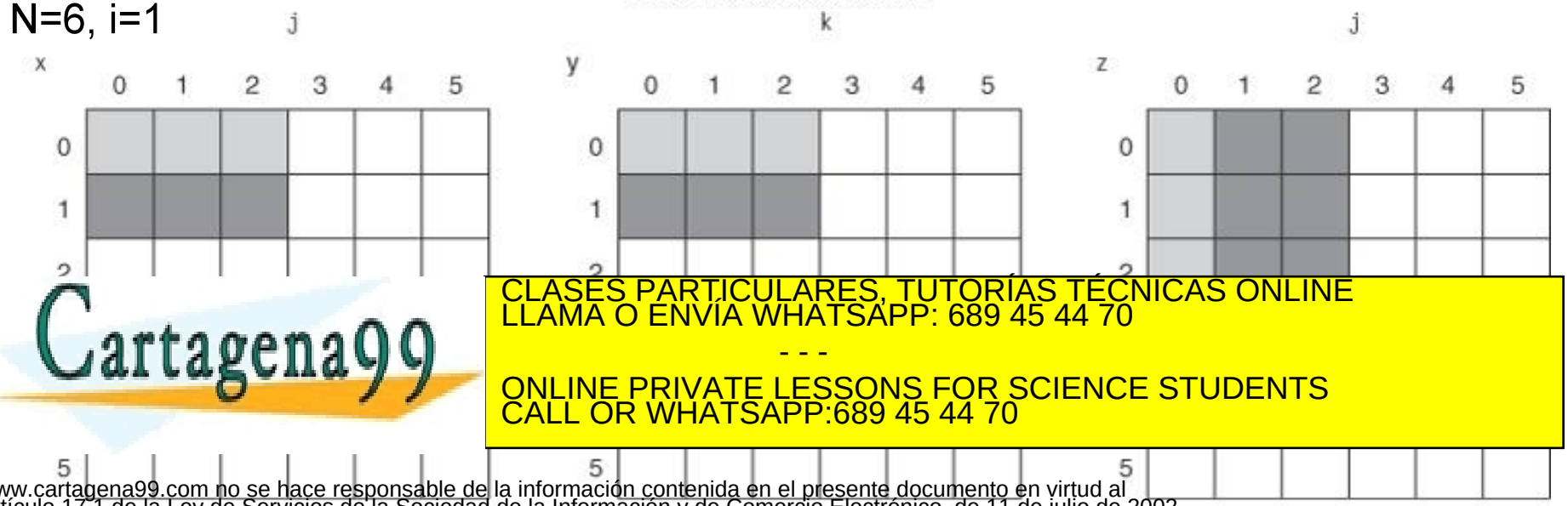

```
for (i=0;i<N;i++) {
  for (j=0; j<N; j++) {
    r=0;
    for (k=0; k<N; k++) {
      r+= y[i][k] * z[k][j];
    }
    x[i][j] = r;
}
```

```
for (bj = 0; bj<N; bj+=B) {
  for (bk=0; bk<N; bk+=B) {
    for (i=0; i<N; i++) {
      for (j=bj; j<min(bj+B,N); j++) {
        r=0;
        for (k=bk;k<min(bk+B,N); k++) {
          r += y[i][k] * z[k][j];
        }
        x[i][j] += r;
      }
    }
}
```

B = Blocking factor



N=6, i=1



9: Data and instruction hardware prefetching

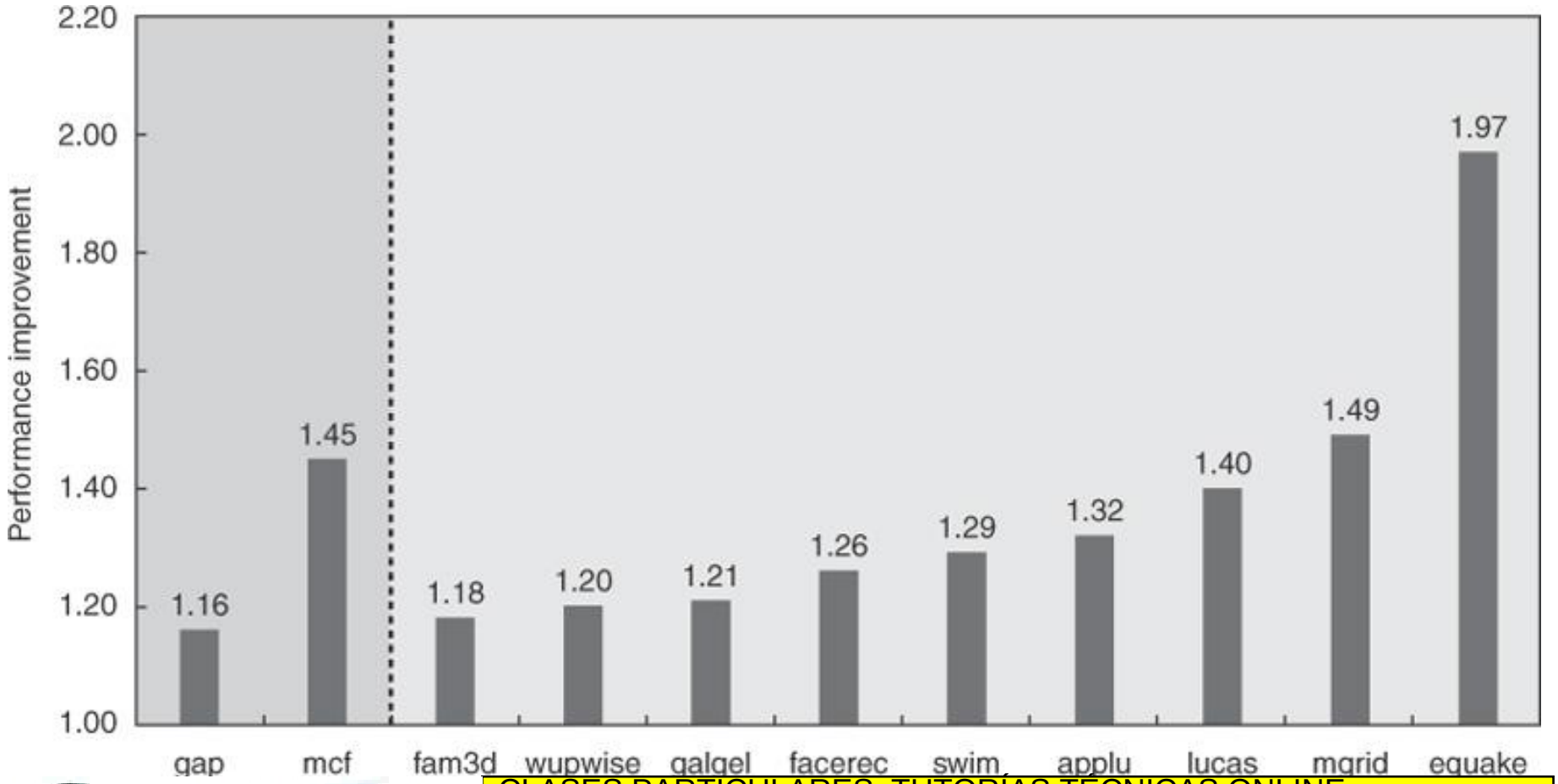
Instruction pefetching

- Instruction high spatial locality.
- Read **two blocks** on miss.
  - Block causing miss.
  - Next block.
- Location:
  - Block causing miss Instruction cache.
  - Next block Instruction

Data prefetching

- Example: Pentium 4.
- Allows 4KB prefetching to L2 cache.
- Prefetching invoked if:
  - 2 misses on L2 due to same page.
  - Distance between misses

Prefetching speedup



10: Compiler controlled prefetching

- Compiler generates prefetching instructions.
  - Register prefetch → Load value into register.
  - Cache prefetch → Load block into cache.

- Types:
  - Faulting → Can generate exception in VM.
  - Non-faulting → Cannot generate exception in VM.

Optimizations Summary

Technique	Hit Time	Band-width	Miss penalty	Miss rate	HW cost/ complexity	Comment
Small and simple caches	+			–	0	Trivial; widely used
Way-predicting caches	+				1	Used in Pentium 4
Pipelined cache access	–	+			1	Widely used
Nonblocking caches		+	+		3	Widely used
Banked caches		+			1	Used in L2 of i7 and Cortex A8
Critical word first and early restart			+		2	Widely used
Merging write buffer			+		1	Widely used with write through
Compiler techniques to reduce cache misses				+	0	Software is a challenge; some computers have compiler option
Hardware prefetching of instructions and data						Many prefetch instructions; AMD Opteron

Reference

- **Computer Architecture. A Quantitative Approach.**
  Fifth Edition.
  Hennessy y Patterson.
  Sections: 2.1, 2.2

- Exercises: 2.1, 2.2, 2.3, 2.8, 2.9, 2.10, 2.11, 2.12