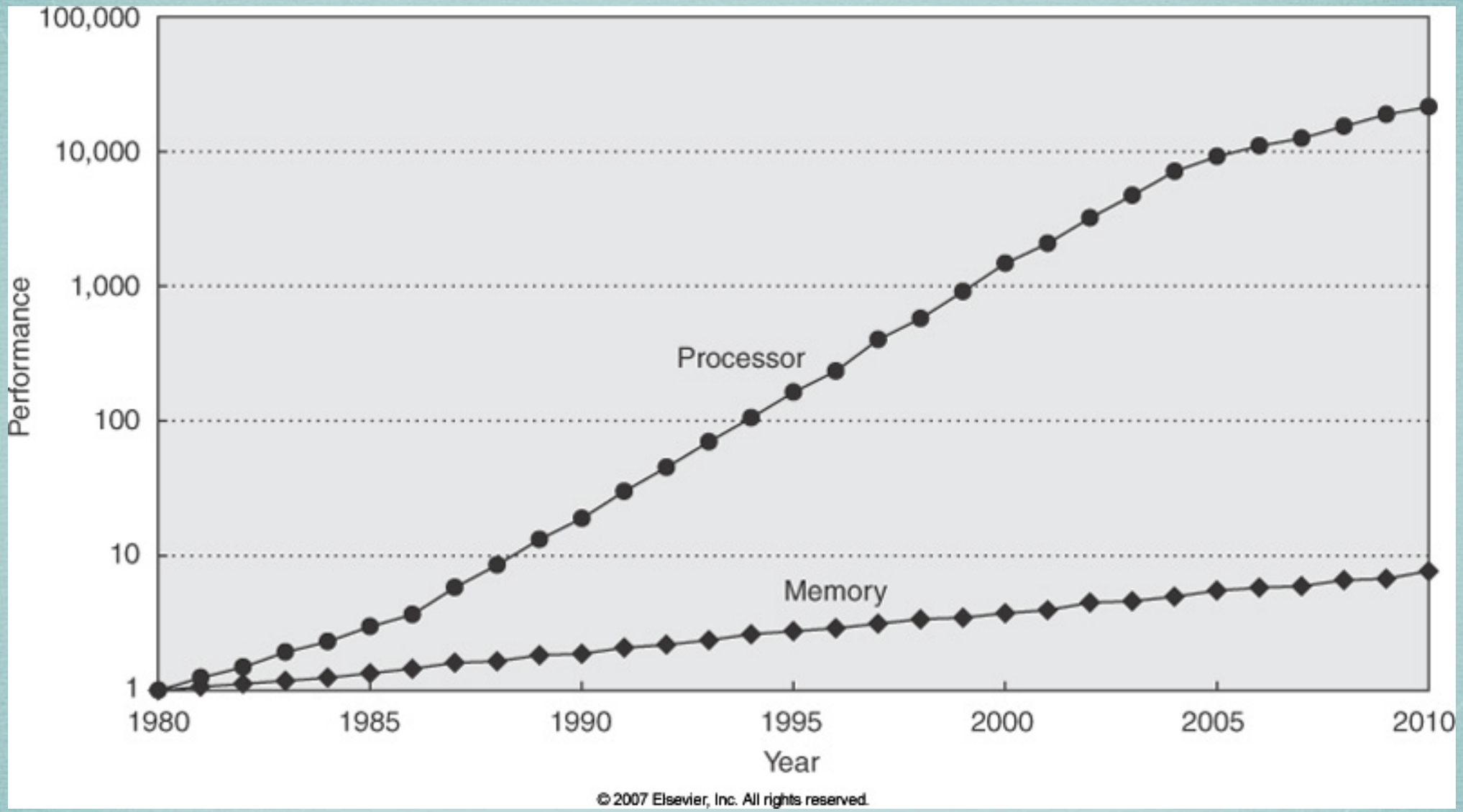
MEMORY HIERARCHY BASICS

B649

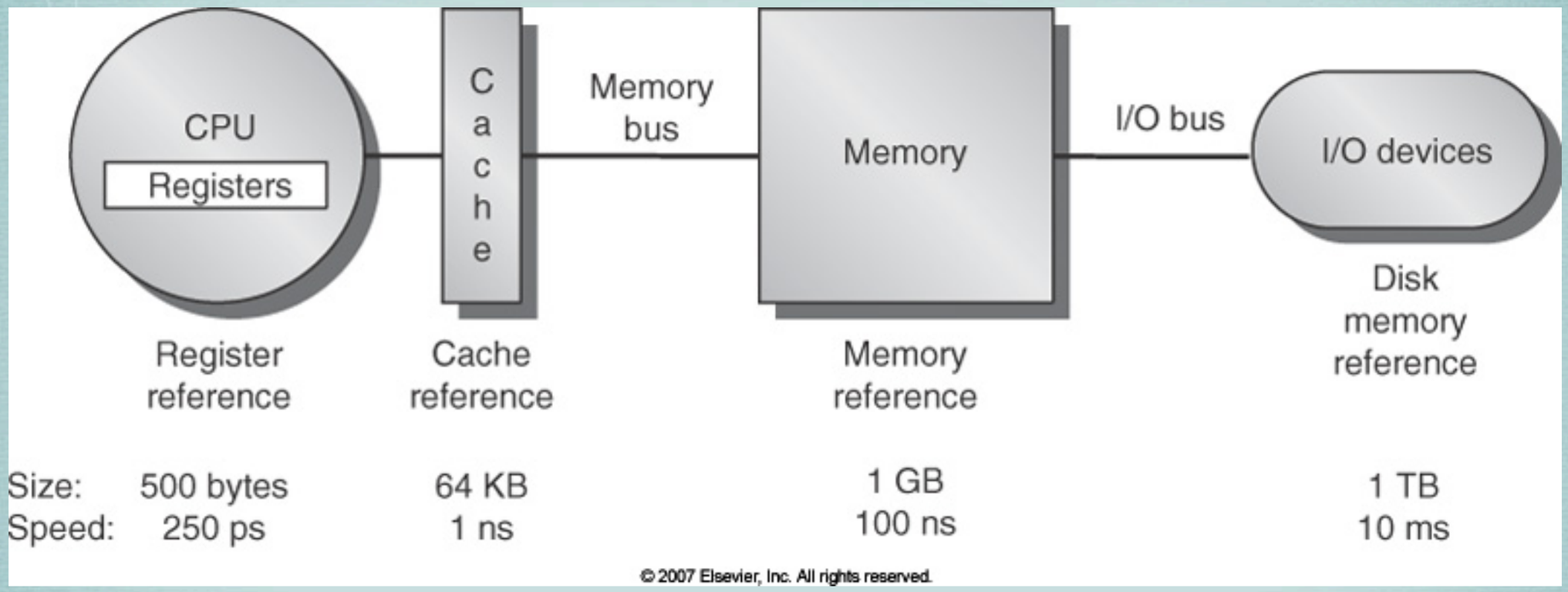
Parallel Architectures and Programming

BASICS

Why Do We Need Caches?



Overview



Terminology

cache
virtual memory
memory stall cycles
direct mapped
valid bit
block address
write through
instruction cache
average memory access time
cache hit
page
miss penalty
fully associative
dirty bit
block offset
write back
data cache
hit time
cache miss
page fault
miss rate
n-way set associative
least-recently used
tag field
write allocate
unified cache
misses per instruction
block
locality
address trace
set
random replacement
index field
no-write allocate
write buffer
write stall

Terminology

cache
virtual memory
memory stall cycles
direct mapped
valid bit
block address
write through
instruction cache
average memory access time
cache hit
page
miss penalty
fully associative
dirty bit
block offset
write back
data cache
hit time
cache miss
page fault
miss rate
n-way set associative
least-recently used
tag field
write allocate
unified cache
misses per instruction
block
locality
address trace
set
random replacement
index field
no-write allocate
write buffer
write stall

Four Memory-Hierarchy Questions

- Where can a block be placed in the upper level?
  - ⋆ block placement
- How is a block found if it is in the upper level?
  - ⋆ block identification
- Which block should be replaced on a miss?
  - ⋆ block replacement
- What happens on a write?
  - ⋆ write strategy

Where Can a Block Be Placed in a Cache?

- Only one place for each block
  - ⋆ direct mapped

(Block address) MOD (Number of blocks in cache)

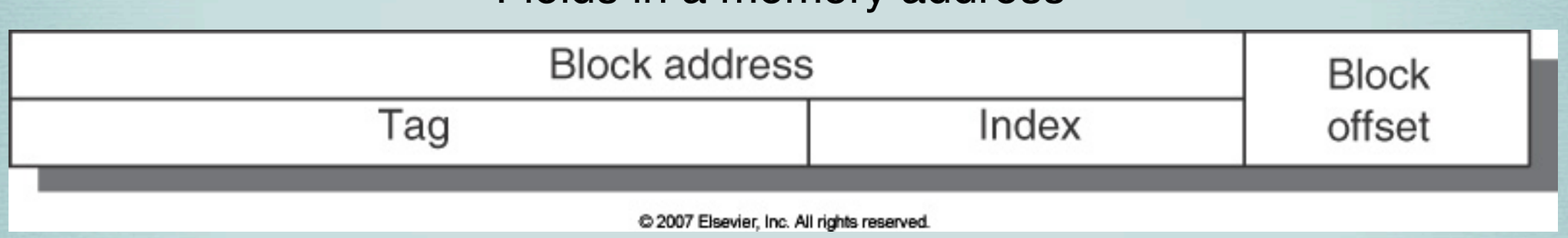
- Anywhere in the cache
  - ⋆ fully associative
- Restricted set of places
  - ⋆ set associative

(Block address) MOD (Number of sets in cache)

How is a Block Found if it is in Cache?

- "Tags" in each cache block gives the block address
  - ⋆ all possible tags searched in parallel (associative memory)
  - ⋆ *valid bit* tells whether a tag match valid

Fields in a memory address



- No "index" field in fully associative caches

Which Block Should be Replaced on a Miss?

- Random
  - ⋆ easy to implement
- Least-recently used (LRU)
  - ⋆ idea: rely on the past to predict the future
  - ⋆ replace the block unused for the longest time
- First in, First out (FIFO)
  - ⋆ approximates LRU (*oldest*, rather than least recently used)
  - ⋆ simpler to implement

What Happens on a Write?

- Write strategy
  - ⋆ write through
    - ✳ write to cache block **and** to the block in the lower-level memory
  - ⋆ write back
    - ✳ write only to cache block, update the lower-level memory when block replaced
- Block allocation strategy
  - ⋆ write allocate
    - ✳ allocate a block on cache miss
  - ⋆ no-write allocate
    - ✳ do **not** allocate, no affect on cache

CACHE PERFORMANCE

Defining Performance

- Miss rate is attractive, but misleading

- Bottom line: CPU time
  - ★ assume in-order execution for now
  - ★ include hit clock cycles in memory cycles or execution cycles?

Defining Performance

- Miss rate is attractive, but misleading

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

- Bottom line: CPU time
  - ★ assume in-order execution for now
  - ★ include hit clock cycles in memory cycles or execution cycles?

Defining Performance

- Miss rate is attractive, but misleading

Average memory access time = Hit time + Miss rate × Miss penalty

- Bottom line: CPU time
  - ★ assume in-order execution for now

CPU time = (CPU execution clock cycles + Memory stall clock cycles) × Clock cycle time

  - ★ include hit clock cycles in memory cycles or execution cycles?

Example

- Assumptions
  - ⋆ in-order execution
  - ⋆ miss penalty = 200 cycles
  - ⋆ cycles per instruction (w/o cache misses) = 1 cycle
  - ⋆ average miss rate = 2%
  - ⋆ average memory references per instruction = 1.5
  - ⋆ cache misses per 1000 instructions = 30
- What is the impact of cache?

Impact of Cache

Assumptions
* in-order execution
* miss penalty = 200 cycles
* cycles per instruction (w/o cache misses) = 1 cycle
* average miss rate = 2%
* average memory references per instruction = 1.5
* cache misses per 1000 instructions = 30

CPU time = IC × (CPI + Memory stall cycle / instruction) × Clock cycle time

With cache = IC × (1.0 + (30/1000 × 200)) × Clock cycle time
= IC × 7.0 × Clock cycle time

Without cache = IC × (1.0 + 200 × 1.5) × Clock cycle time
= IC × 301 × Clock cycle time

Miss Penalty and Out-of-order Execution

- Some of the miss penalty is **hidden**

- Miss latency
  - ⋆ What is start and end of a memory operation (memory latency)?
  - ⋆ What is the start of overlap with the processor (latency overlap)?

Miss Penalty and Out-of-order Execution

- Some of the miss penalty is **hidden**

Memory stall cycle / Instruction = (Misses / Instruction) × (Total miss latency – Overlapped miss latency)

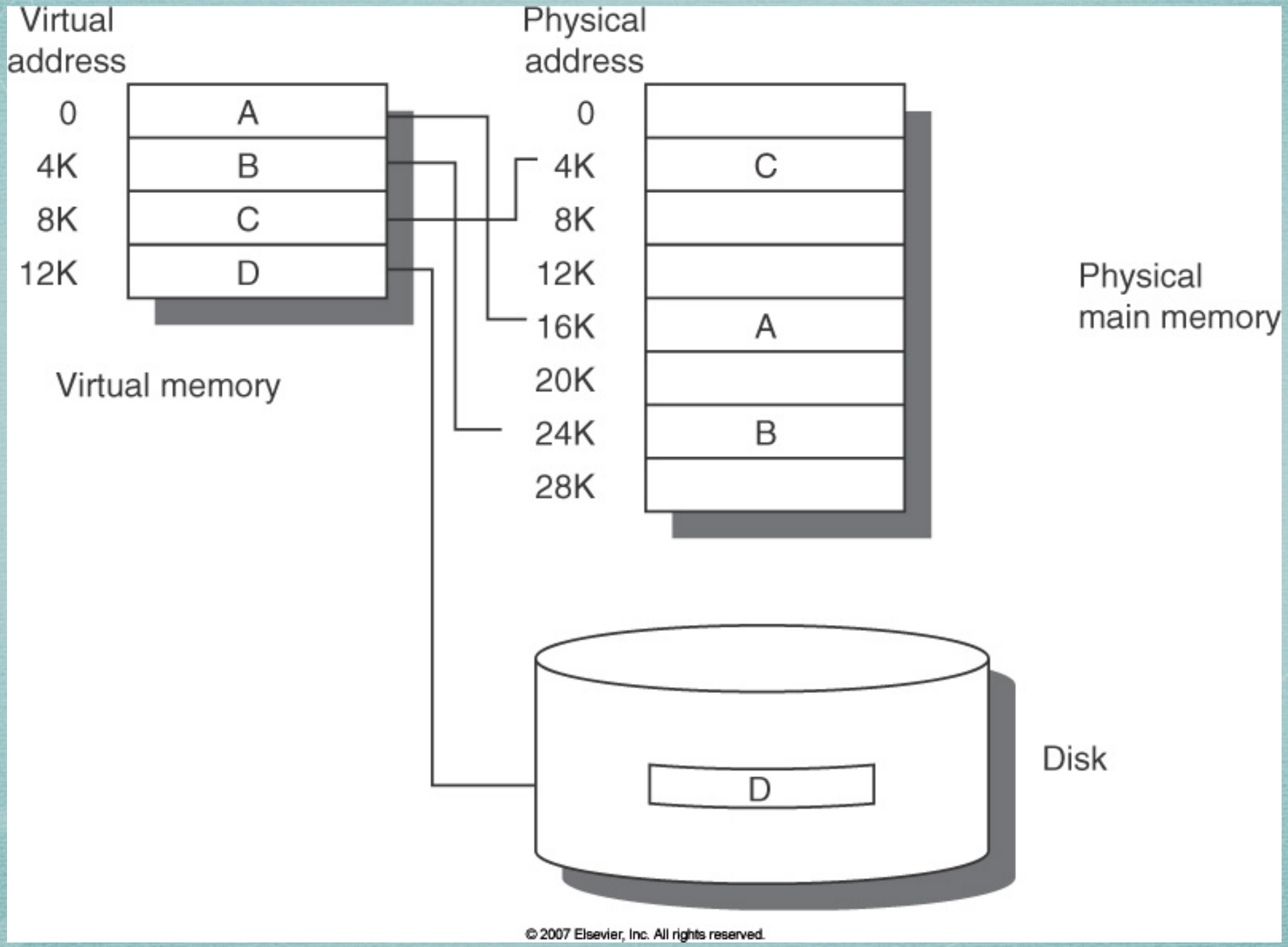
- Miss latency
  - ⋆ What is start and end of a memory operation (memory latency)?
  - ⋆ What is the start of overlap with the processor (latency overlap)?

VIRTUAL MEMORY

... a system has been devised to make the core drum combination appear to the programmer as a single level store, the requisite transfers taking place automatically.

Killburn et al. [1962]

Virtual Memory



Virtual Memory vs Cache

Parameter	First-level cache	Virtual memory
Block (page) size	16–128 bytes	4096–65,536 bytes
Hit time	1–3 clock cycles	100–200 clock cycles
Miss penalty	8–200 clock cycles	1,000,000–10,000,000 clock cycles
(access time)	(6–160 clock cycles)	(800,000–8,000,000 clock cycles)
(transfer time)	(2–40 clock cycles)	(200,000–2,000,000 clock cycles)
Miss rate	0.1–10%	0.00001–0.001%
Address mapping	25–45 bit physical address to 14–20 bit cache address	32–64 bit virtual address to 25–45 bit physical address

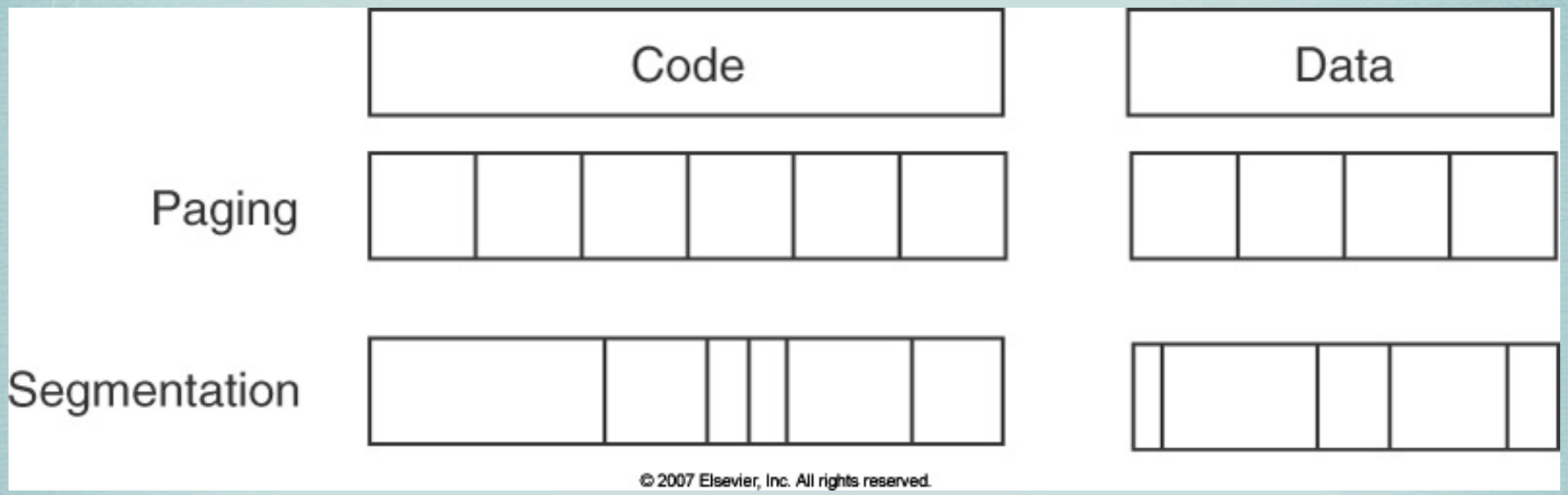
Why Virtual Memory?

- Ability to large programs with large data
  - ⋆ earlier, users would need to use *overlays* manually
- Ability to share a machine among multiple processes, with protection
  - ⋆ very early Windows systems did not have this protection!
- Ease of relocation
  - ⋆ alternatively, would need to use a special relocation register, or do this is software

Virtual Memory: Terminology

- Cache block ⇒ page or segment
- Cache miss ⇒ page fault or address fault
- Address translation or memory mapping
  - ⋆ *virtual* address to *physical* address
- Differences between VM and caches
  - ⋆ replacement controlled by software in VM
  - ⋆ processor address determines the size of VM
  - ⋆ secondary storage shared with file system
- Two flavors of VM: paged and segmented

Segmented vs Paged



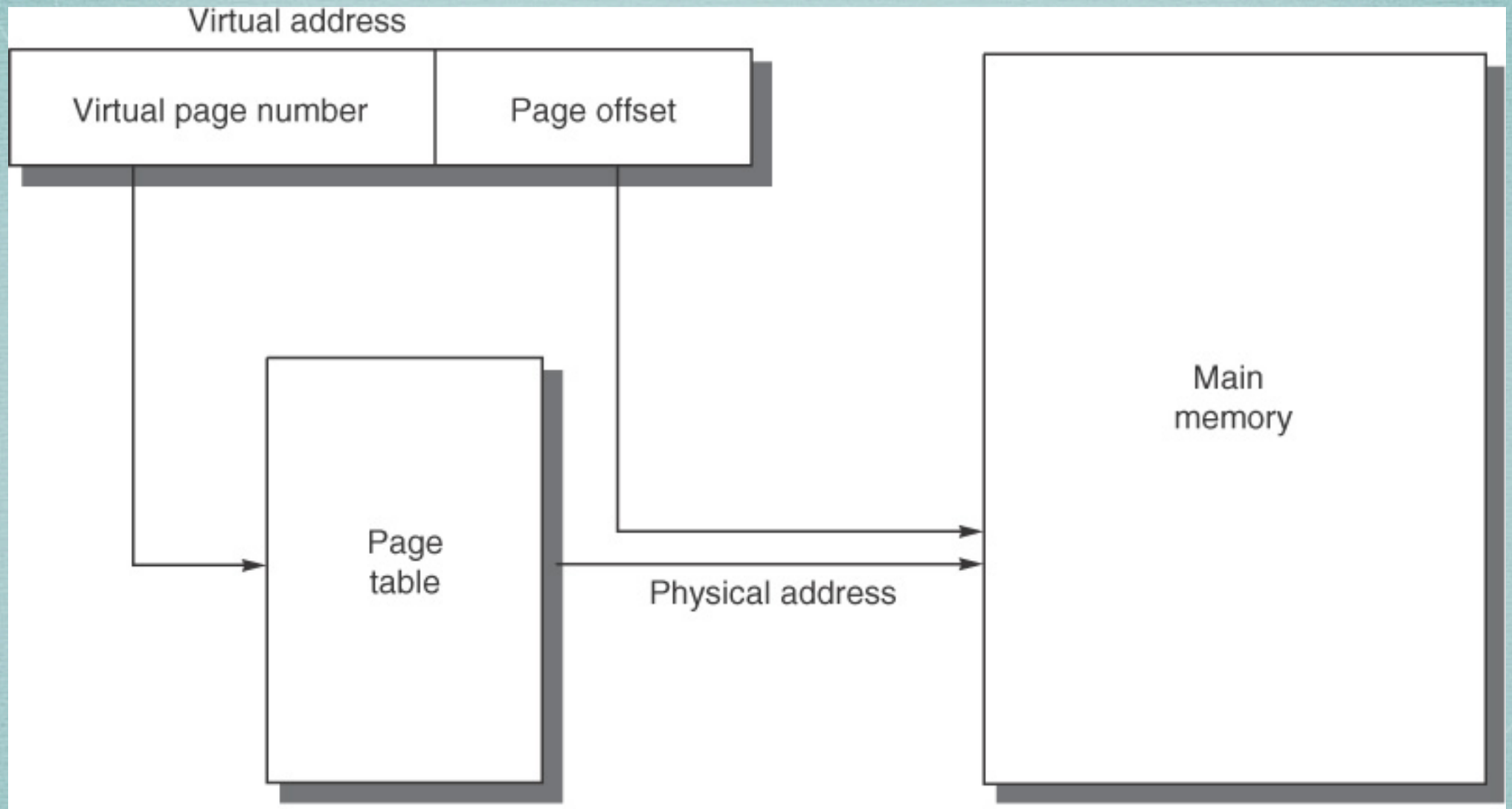
Four Memory Hierarchy Questions

- Where can a block be placed in memory?
  - ⋆ fully associative

Four Memory Hierarchy Questions

- Where can a block be placed in memory?
  - ⋆ fully associative
- How is a block found if it is in main memory?
  - ⋆ using a page table

Page Table



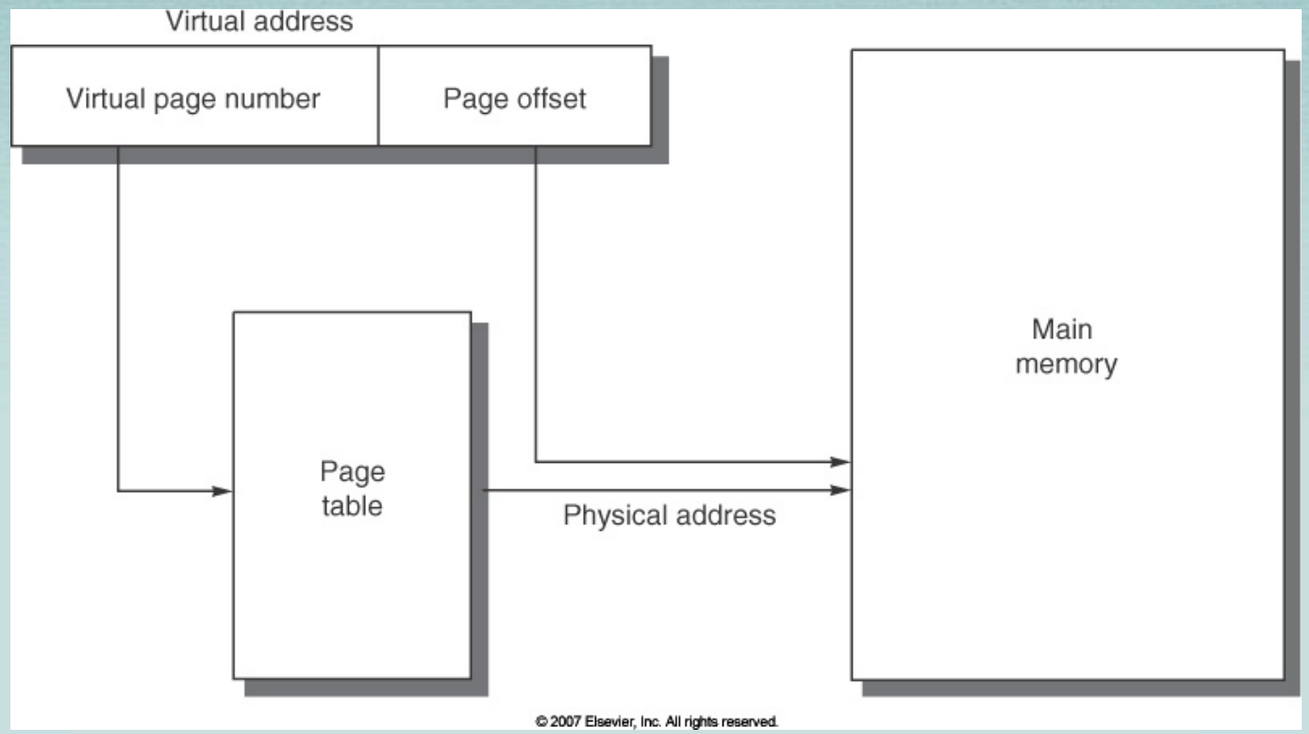
Four Memory Hierarchy Questions

- Where can a block be placed in memory?
  - ⋆ fully associative
- How is a block found if it is in main memory?
  - ⋆ using a page table
- Which block should be replaced on a virtual memory miss?
  - ⋆ LRU policy, implemented with a *use bit* or *reference bit*

Four Memory Hierarchy Questions

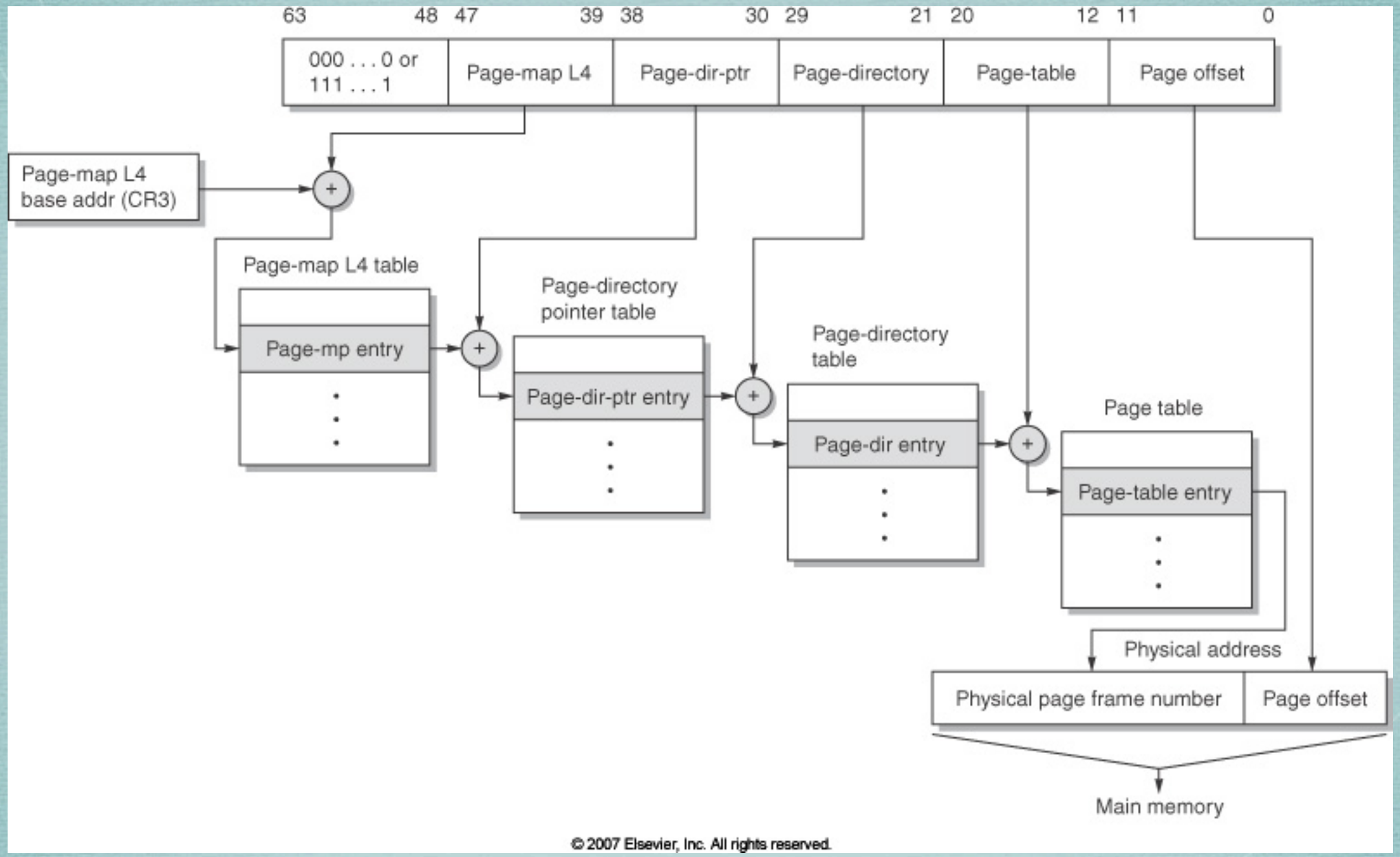
- Where can a block be placed in memory?
  - ⋆ fully associative
- How is a block found if it is in main memory?
  - ⋆ using a page table
- Which block should be replaced on a virtual memory miss?
  - ⋆ LRU policy, implemented with a *use bit* or *reference bit*
- What happens on a write?
  - ⋆ **always** write back

Optimizing Page Table



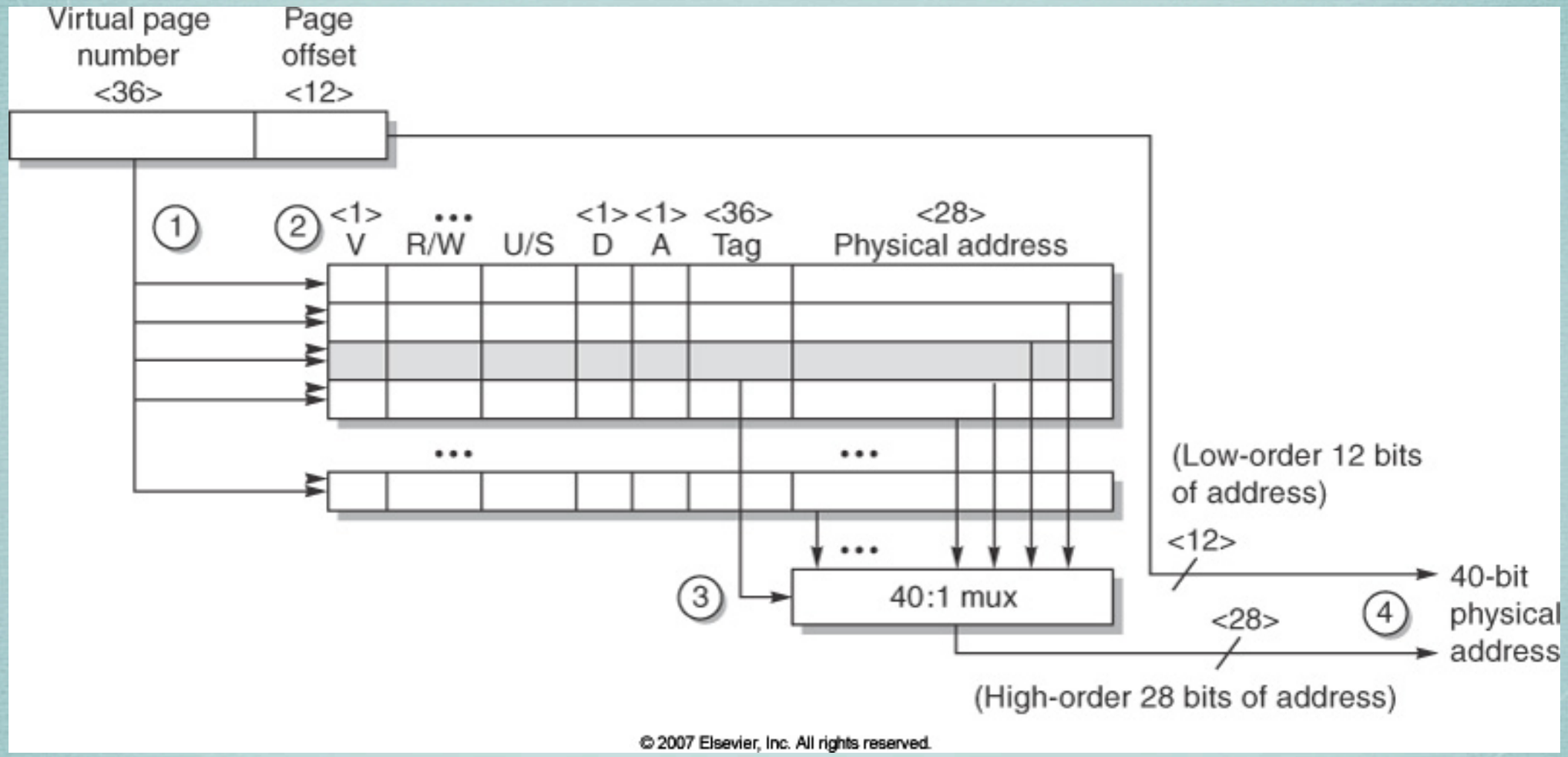
- Virtualized page tables
- Inverted page tables
  - ⋆ hash tables, as many entries as the number of allocated pages
- Multilevel page tables

Multilevel Page Tables



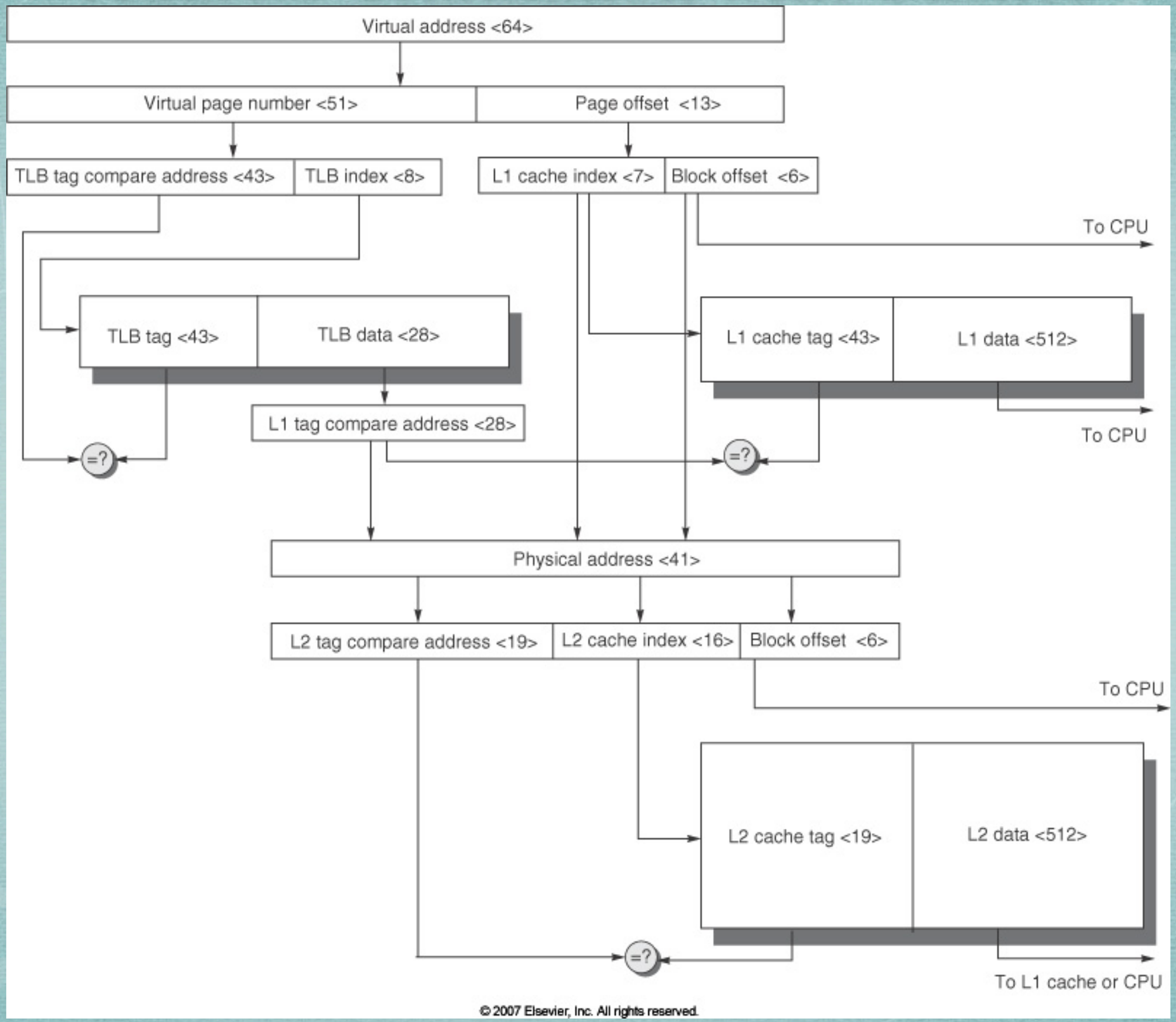
Virtual address translation on Opteron

Translation Look-aside Buffers (TLBs)



TLB uses fully associative placement

Address Translation with Caches



SIX BASIC CACHE OPTIMIZATIONS

Six Ideas

Average memory access time = Hit time + Miss rate × Miss penalty

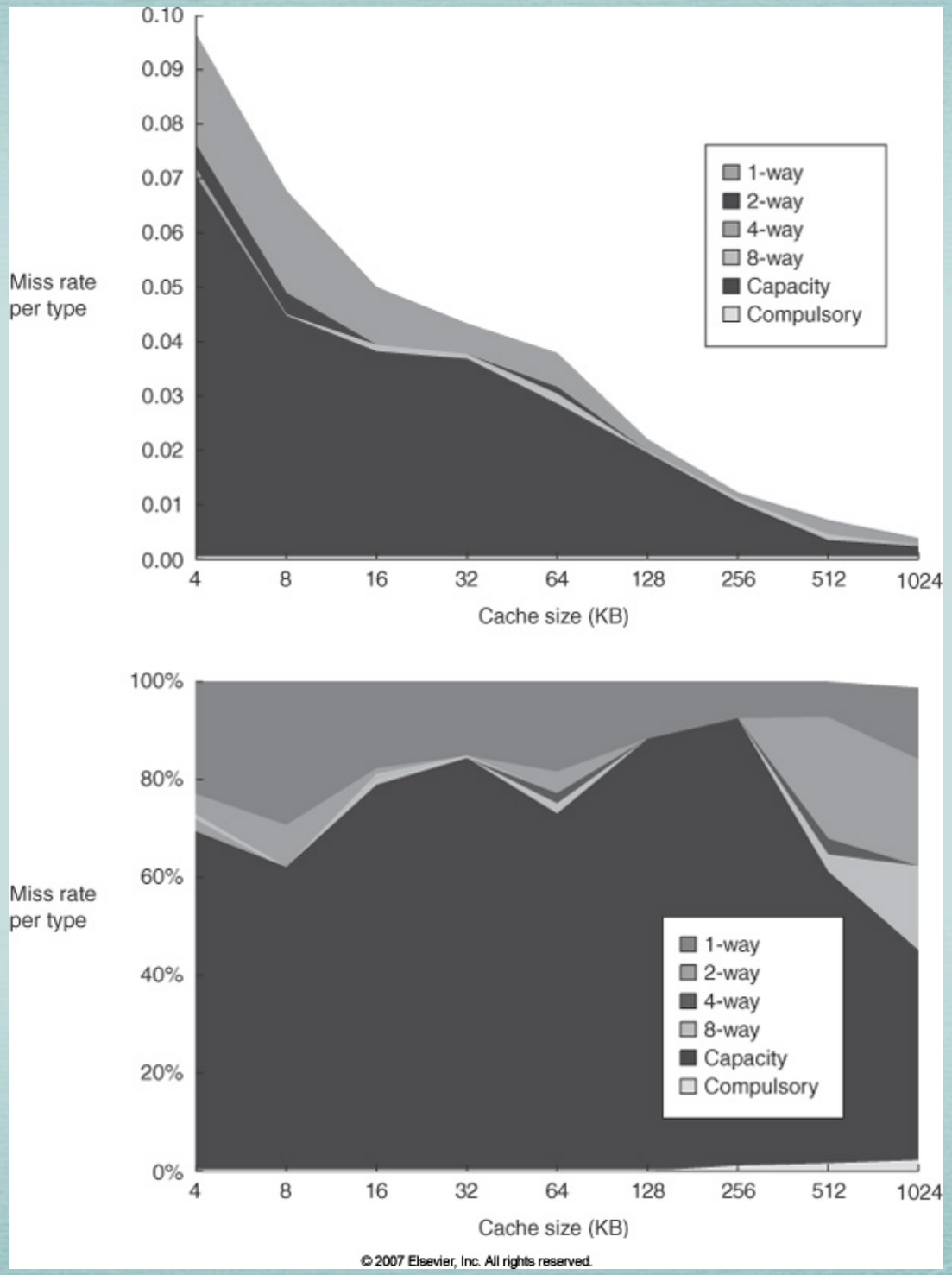
- Reducing the miss rate
  - ⋆ larger block size
  - ⋆ larger cache size
  - ⋆ higher associativity
- Reducing the miss penalty
  - ⋆ multilevel caches
  - ⋆ prioritize read misses over writes
- Reducing the time to hit in cache
  - ⋆ avoid address translation when indexing the cache

Type of Cache Misses

- Compulsory
  - ⋆ caused by the very first access to the block
- Capacity
  - ⋆ if the cache cannot contain all the needed blocks, misses (in addition to the compulsory misses) due to blocks being discarded and retrieved later
- Conflict
  - ⋆ if the block placement is not fully associative, misses (in addition to the above two kinds) due to blocks being discarded and later retrieved due to too many blocks being mapped to the same set
- Coherency

Classical Approach: Reduce Miss Rate

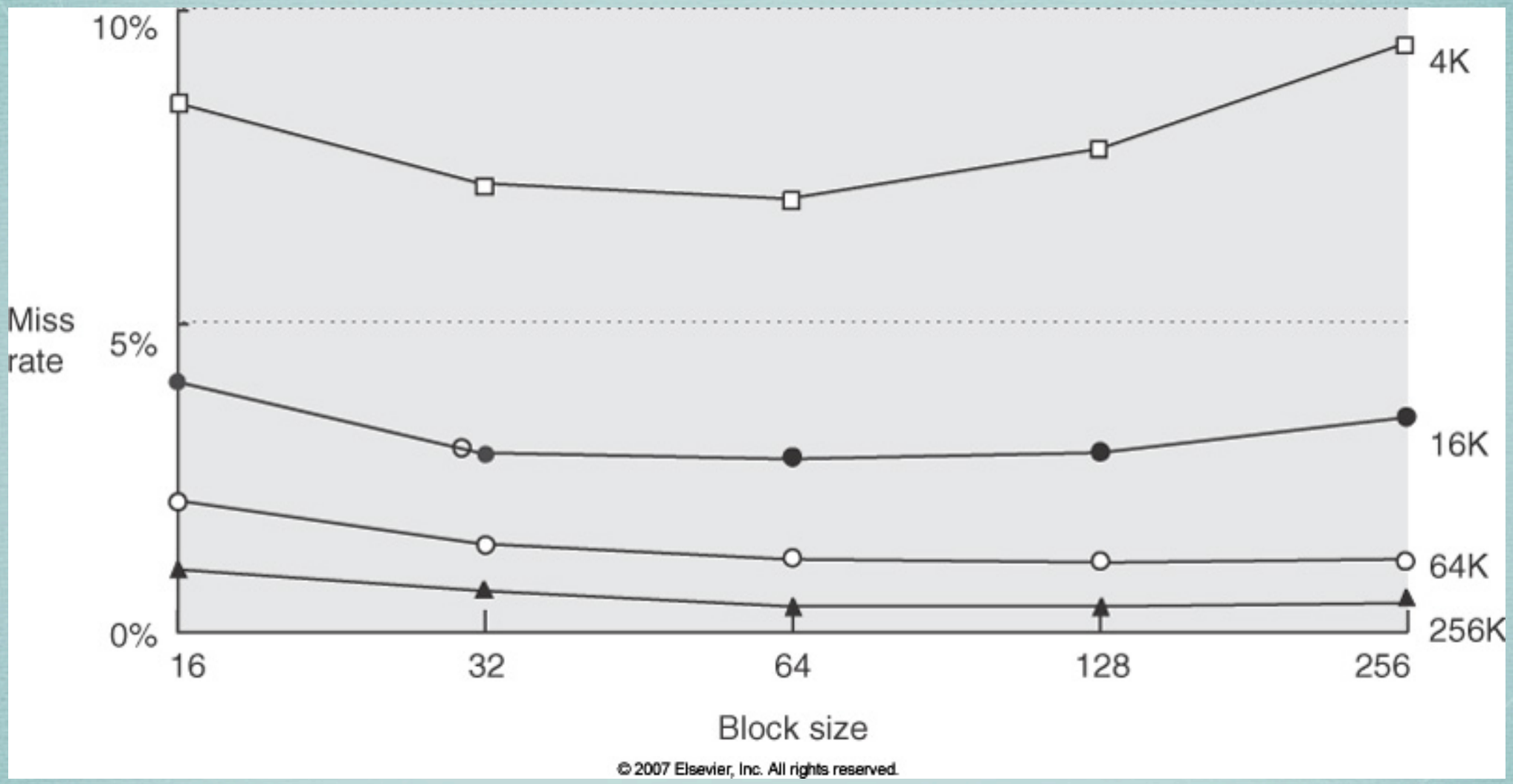
SPEC2000 benchmarks



Optimizations

- Larger block size to reduce miss rate

Increasing Block Size



Example

- Memory access:
  - ⋆ latency = 80 clock cycles
  - ⋆ bandwidth: 16 bytes each 2 cycles, thereafter
  - ⋆ thus, 16 bytes in 82 cycles, 32 bytes in 84 cycles, ...

	Cache size			
Block size	4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

Example

- Memory access:
  - ⋆ latency = 80 clock cycles
  - ⋆ bandwidth: 16 bytes each 2 cycles, thereafter
  - ⋆ thus, 16 bytes in 82 cycles, 32 bytes in 84 cycles, ...

	Cache size			
Block size	4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

Average memory access time = Hit time + Miss rate × Miss penalty
For 16-byte block, 4K size cache = 1 + (8.57% × 82) = 8.027 clock cycles
For 256-byte block, 256K size cache = 1 + (0.49% × 112) = 1.549 clock cycles

Average Memory Access Times

Block size	Miss penalty	Cache size			
		4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.160	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

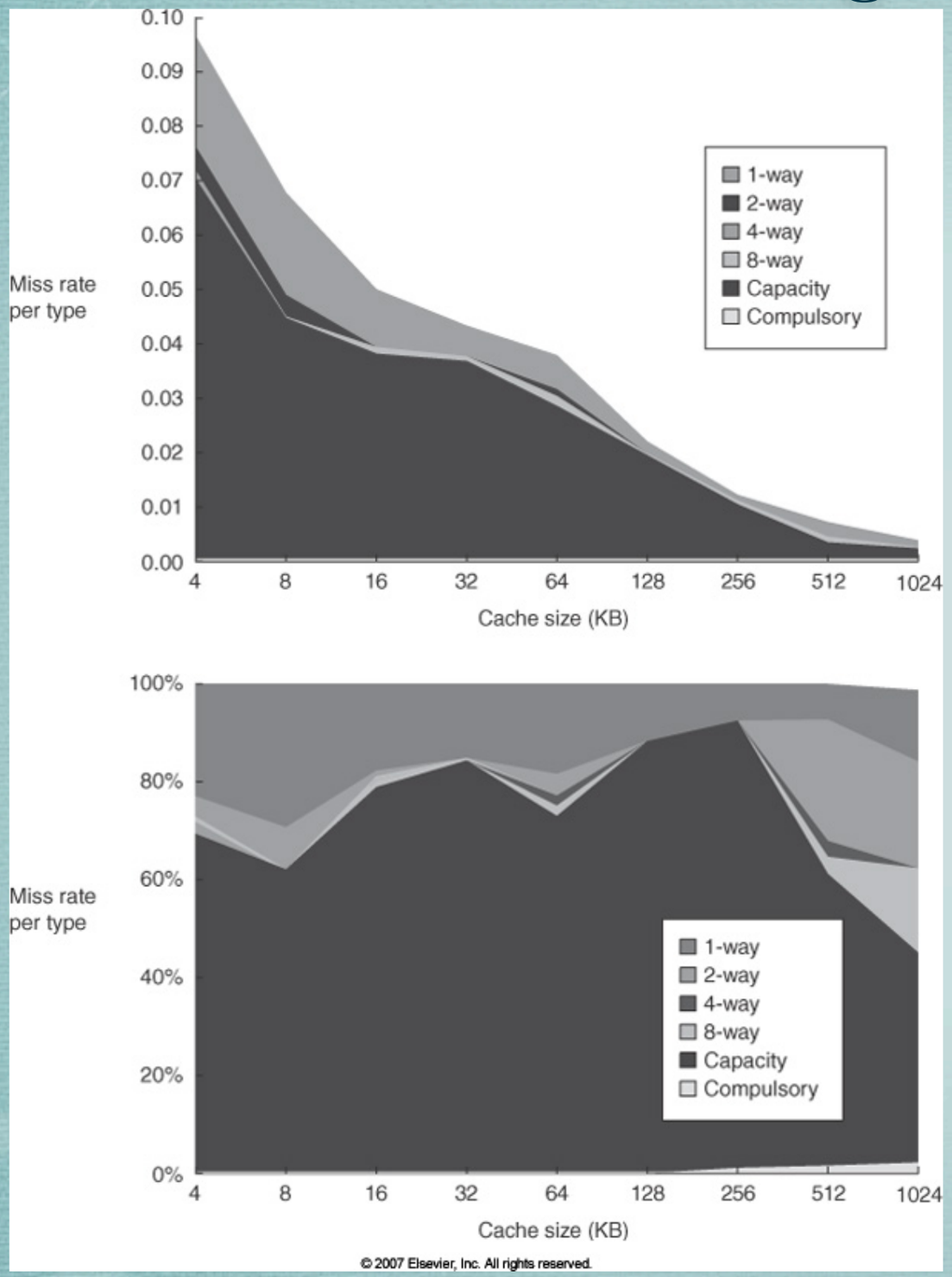
Optimizations

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate

Optimizations

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate
- Higher associativity to reduce miss rate

Increasing Associativity



Lessons:

1. 8-way is as good as fully associative

2. *2:1 cache rule of thumb*: direct mapped cache of size N has about the same miss rate as 2-way of size N/2

Example

Clock cycle time$_2$ = 1.36 × Clock cycle time$_1$
Clock cycle time$_4$ = 1.44 × Clock cycle time$_2$
Clock cycle time$_8$ = 1.52 × Clock cycle time$_4$
Hit time = 1 clock cycle
Miss penalty to L2 = 25 cycles

Example

$\text{Clock cycle time}_2 = 1.36 \times \text{Clock cycle time}_1$
$\text{Clock cycle time}_4 = 1.44 \times \text{Clock cycle time}_2$
$\text{Clock cycle time}_8 = 1.52 \times \text{Clock cycle time}_4$
Hit time = 1 clock cycle
Miss penalty to L2 = 25 cycles

$$\text{Average memory access time}_8 = \text{Hit time}_8 + \text{Miss rate}_8 \times \text{Miss penalty}_8$$
$$= 1.52 + \text{Miss rate}_8 \times 25$$
$$\text{Average memory access time}_4 = 1.44 + \text{Miss rate}_4 \times 25$$
$$\text{Average memory access time}_2 = 1.36 + \text{Miss rate}_2 \times 25$$
$$\text{Average memory access time}_1 = 1.00 + \text{Miss rate}_1 \times 25$$

Average Memory Access Times

Cache size (KB)	Associativity			
	One-way	Two-way	Four-way	Eight-way
4	3.44	3.25	3.22	**3.28**
8	2.69	2.58	2.55	**2.62**
16	2.23	**2.40**	**2.46**	**2.53**
32	2.06	**2.30**	**2.37**	**2.45**
64	1.92	**2.14**	**2.18**	**2.25**
128	1.52	**1.84**	**1.92**	**2.00**
256	1.32	**1.66**	**1.74**	**1.82**
512	1.20	**1.55**	**1.59**	**1.66**

Optimizations

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate
- Higher associativity to reduce miss rate
- Multilevel caches to reduce miss penalty

Computing Memory Access Time

$$\text{Average memory access time} = \text{Hit time}_1 + \text{Miss rate}_1 \times \text{Miss penalty}_1$$

$$\text{Miss penalty}_1 = \text{Hit time}_2 + \text{Miss rate}_2 \times \text{Miss penalty}_2$$

$$\text{Average memory access time} = \text{Hit time}_1 + \text{Miss rate}_1 \times (\text{Hit time}_2 + \text{Miss rate}_2 \times \text{Miss penalty}_2)$$

Computing Memory Access Time

$$\text{Average memory access time} = \text{Hit time}_1 + \text{Miss rate}_1 \times \text{Miss penalty}_1$$

$$\text{Miss penalty}_1 = \text{Hit time}_2 + \text{Miss rate}_2 \times \text{Miss penalty}_2$$

$$\text{Average memory access time} = \text{Hit time}_1 + \text{Miss rate}_1 \times (\text{Hit time}_2 + \text{Miss rate}_2 \times \text{Miss penalty}_2)$$

- Note: L2 misses are on leftovers from L1

Two Types of Miss Rates

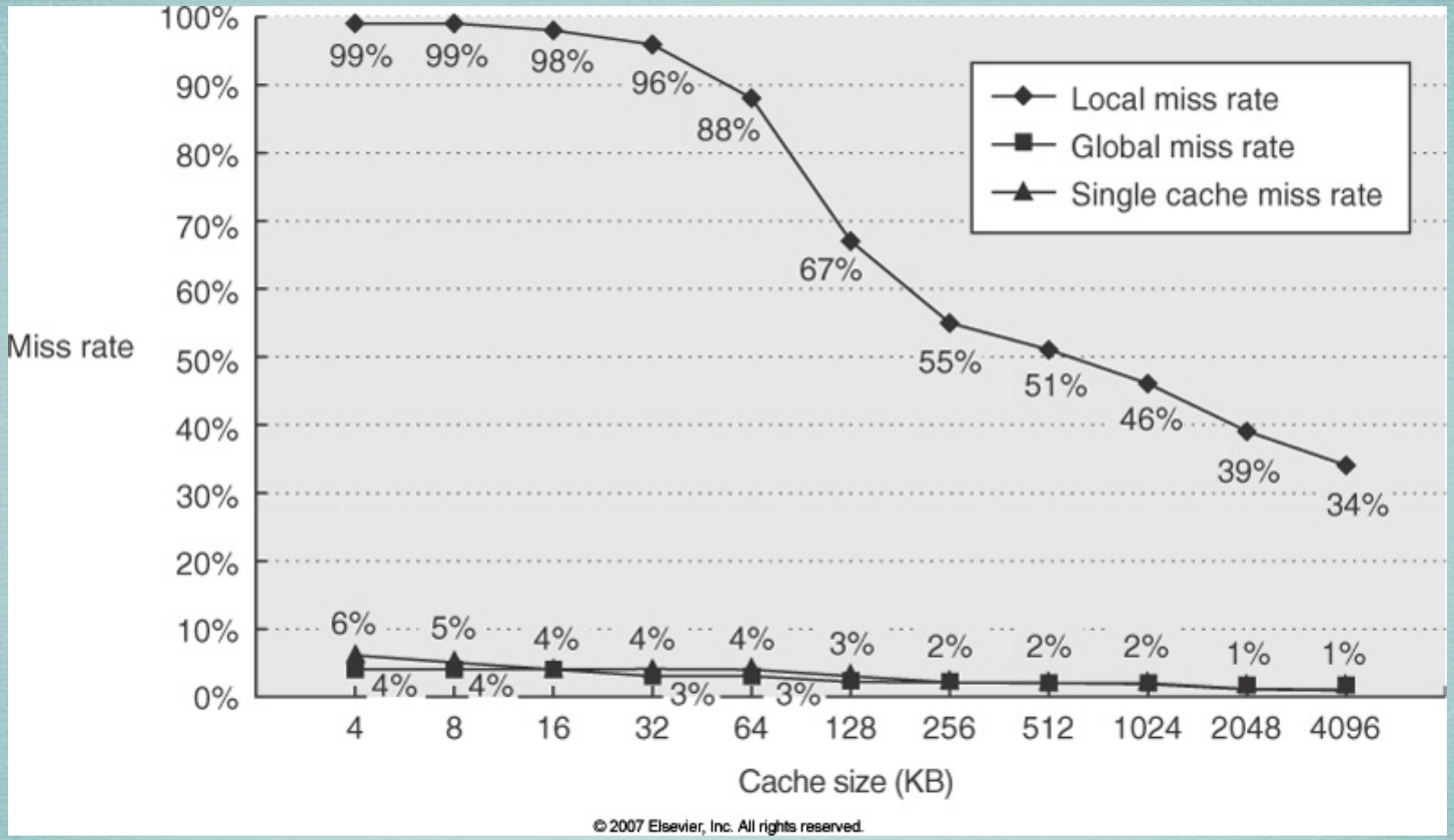
- Local Miss Rate
  - ⋆ number of misses / total number of memory accesses to this cache
- Global Miss Rate
  - ⋆ number of misses / total number of memory accesses generated by the processor
- Local miss rate is large for second-level cache
  - ⋆ Why?

Two Types of Miss Rates

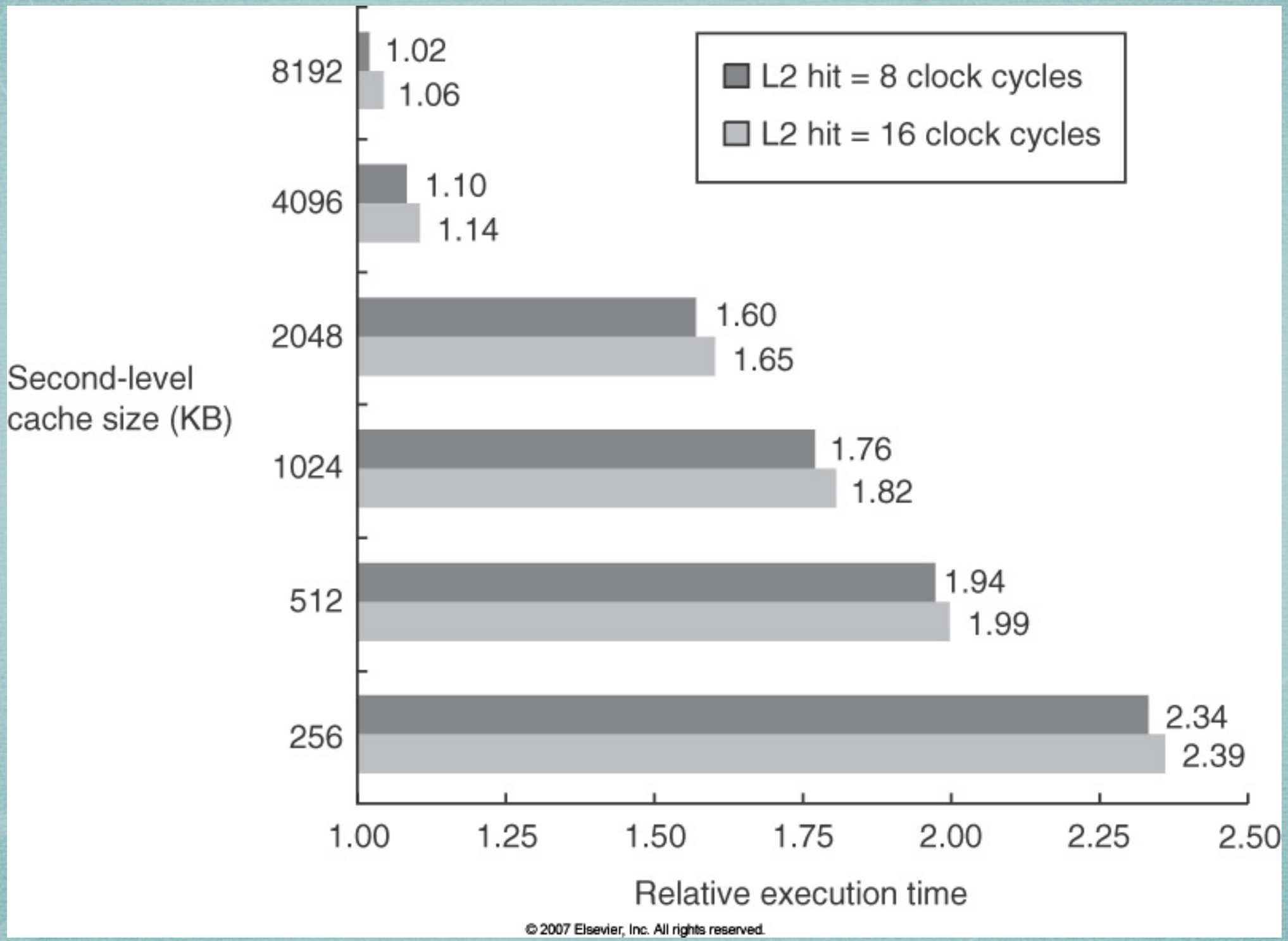
- Local Miss Rate
  - ⋆ number of misses / total number of memory accesses to this cache
- Global Miss Rate
  - ⋆ number of misses / total number of memory accesses generated by the processor
- Local miss rate is large for second-level cache
  - ⋆ Why?

Average memory stalls per instruction =
L1 misses per instruction × L2 hit time +
L2 misses per instruction × L2 miss penalty

Miss Rates vs Cache Sizes



Relative Execution Time by L2 Size



Questions to Consider

- Size of L2 cache
- Set-associativity
- Multi-level inclusion
  - ⋆ different block sizes cause problems
  - ⋆ what if size of L2 is only slightly bigger than L1?
  - ⋆ AMD Opteron follows exclusion property
    - ✳ swap L1 and L2 blocks on L1 miss

Optimizations

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate
- Higher associativity to reduce miss rate
- Multilevel caches to reduce miss penalty
- Prioritizing read misses over writes to reduce miss penalty

Example

```
SW R3, 512(R0)     ;M[512]←R3   (cache index 0)
LW R1, 1024(R0)    ;R1←M[1024]  (cache index 0)
LW R2, 512(R0)     ;R2←M[512]   (cache index 0)
```

Example

```
SW R3, 512(R0)     ;M[512]←R3   (cache index 0)
LW R1, 1024(R0)    ;R1←M[1024]  (cache index 0)
LW R2, 512(R0)     ;R2←M[512]   (cache index 0)
```

- RAW data hazard
- Write through cache
  - ⋆ write buffer can cause a problem
  - ⋆ either let write finish or check the buffer and let read miss proceed (preferred)
- Write back cache
  - ⋆ either let writes of replaced dirty blocks to finish, or buffer writes and let read misses proceed after checking the buffer (preferred)

Optimizations

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate
- Higher associativity to reduce miss rate
- Multilevel caches to reduce miss penalty
- Prioritizing read misses over writes to reduce miss penalty
- Avoiding address translation during indexing of the cache to reduce hit time

Virtual Caches

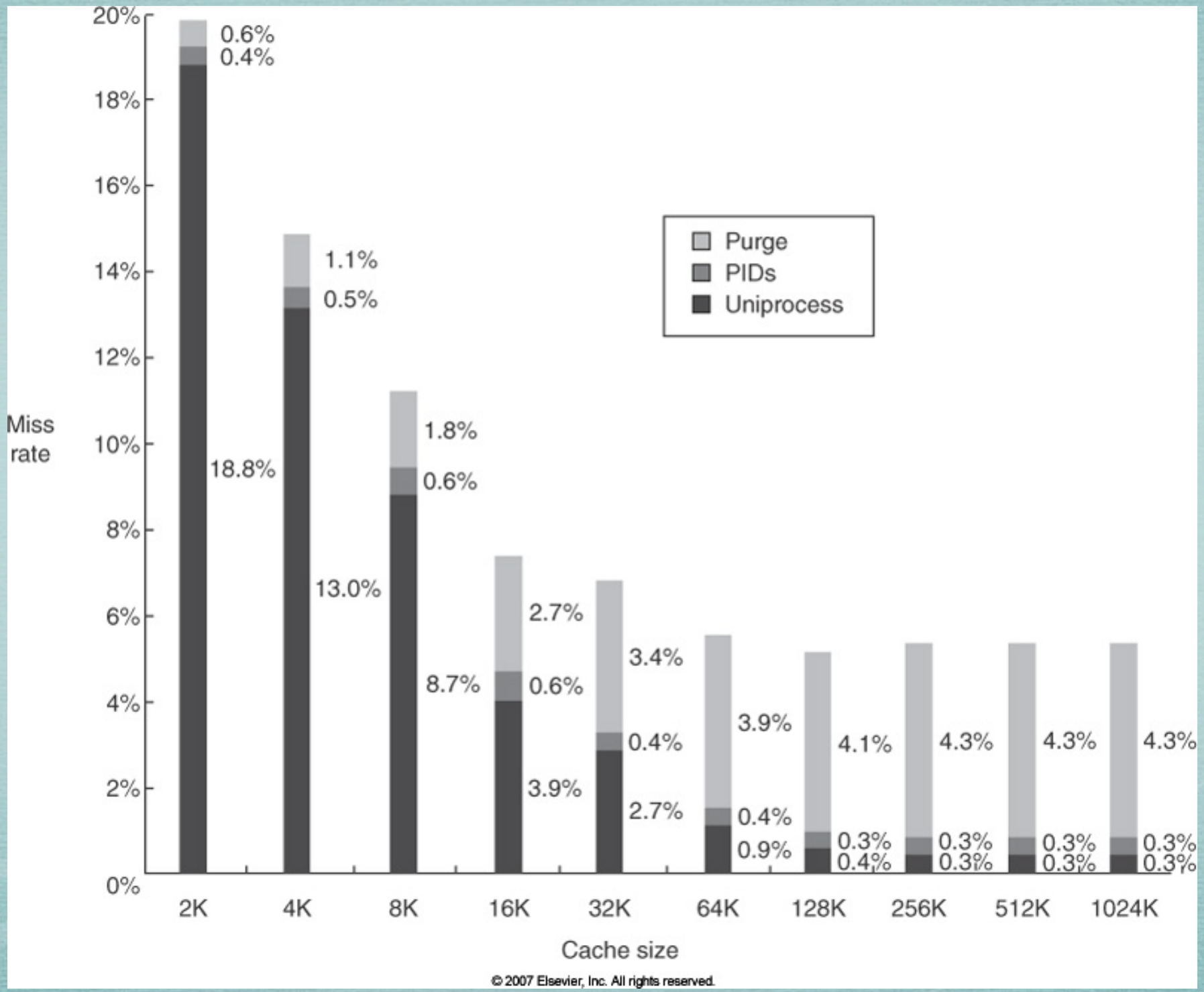
Virtual Caches

- Use virtual address in caches to avoid translation

Virtual Caches

- Use virtual address in caches to avoid translation
- Reasons against:
  - ⋆ Page-level protection (checked at address translation time)
  - ⋆ Process switches might force flushes; or expand the cache address tag with process-identifier (PID) tag

Miss Rate of Virtually Addressed Caches

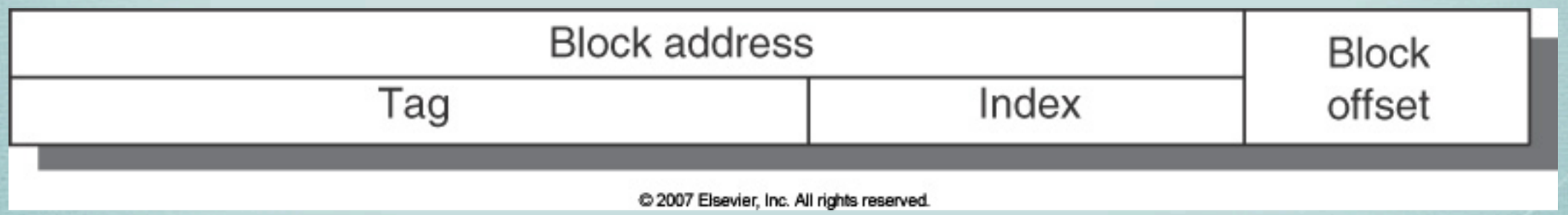


Virtual Caches

- Use virtual address in caches to avoid translation
- Reasons against:
  - ⋆ Page-level protection (checked at address translation time)
  - ⋆ Process switches might force flushes; or expand the cache address tag with process-identifier (PID) tag
  - ⋆ Multiple virtual addresses might map to same physical address (*aliases* or *synonyms*)
    - ✳ need special hardware to ensure unique copy (e.g., Opteron has 64KB instruction cache, 2-way set associative, 4KB page size)
    - ✳ *page coloring* simplifies the hardware (e.g., Sun OS required all aliases to be identical in last 18 bits)
  - ⋆ I/O often uses physical addresses

Best of Both Worlds

- Idea: Use part of page offset to index the cache
  - ⋆ recall that page offset is identical in both virtual and physical addresses
- Virtually indexed, physically tagged
- Limitation: direct-mapped cache size may not be bigger than page size



  - ⋆ increase associativity to get larger caches (e.g., IBM 3033 has 16-way set-associative cache to get over the limit of 4KB page size)

6 Optimizations: Recap

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate
- Higher associativity to reduce miss rate
- Multilevel caches to reduce miss penalty
- Prioritizing read misses over writes to reduce miss penalty
- Avoiding address translation during indexing of the cache to reduce hit time

Watch Out for OS Performance

	Misses		Time							
			% time due to application misses		% time due directly to OS misses					% time OS misses and application conflicts
Workload	% in applications	% in OS	Inherent application misses	OS conflicts with applications	OS instruction misses	Data misses for migration	Data misses in block operations	Rest of OS misses	
Pmake	47%	53%	14.1%	4.8%	10.9%	1.0%	6.2%	2.9%	25.8%
Multipgm	53%	47%	21.6%	3.4%	9.2%	4.2%	4.7%	3.4%	24.9%
Oracle	73%	27%	25.7%	10.2%	10.6%	2.6%	0.6%	2.8%	26.8%

NEXT: MORE ON MEMORY HIERARCHY