CSEE 3827: Fundamentals of Computer Systems

Lecture 23

April 29, 2009

Martha Kim
martha@cs.columbia.edu

Caching

Memory Technology

Static RAM (SRAM) → 0.5ns – 2.5ns, $2000 – $5000 per GB

Dynamic RAM (DRAM) → 50ns – 70ns, $20 – $75 per GB

Magnetic disk → 5ms – 20ms, $0.20 – $2 per GB

Ideal memory = access time of SRAM + capacity and cost/GB of disk

Principle of Locality

Programs access a small proportion of their address space at any time

Temporal Locality:

Items accessed recently are likely to be accessed again soon

e.g., instructions in a loop, induction variables

Spatial locality:

Items near those accessed recently are likely to be accessed soon

E.g., sequential instruction access, array data

Taking Advantage of Locality

Organize memory hierarchically

Copy **more recently accessed (and nearby) items** from DRAM to smaller SRAM memory → cache attached to CPU

Copy **recently accessed (and nearby) items** to smaller DRAM memory → main memory

*Store **everything** on disk*

Canonical Memory Hierarchy

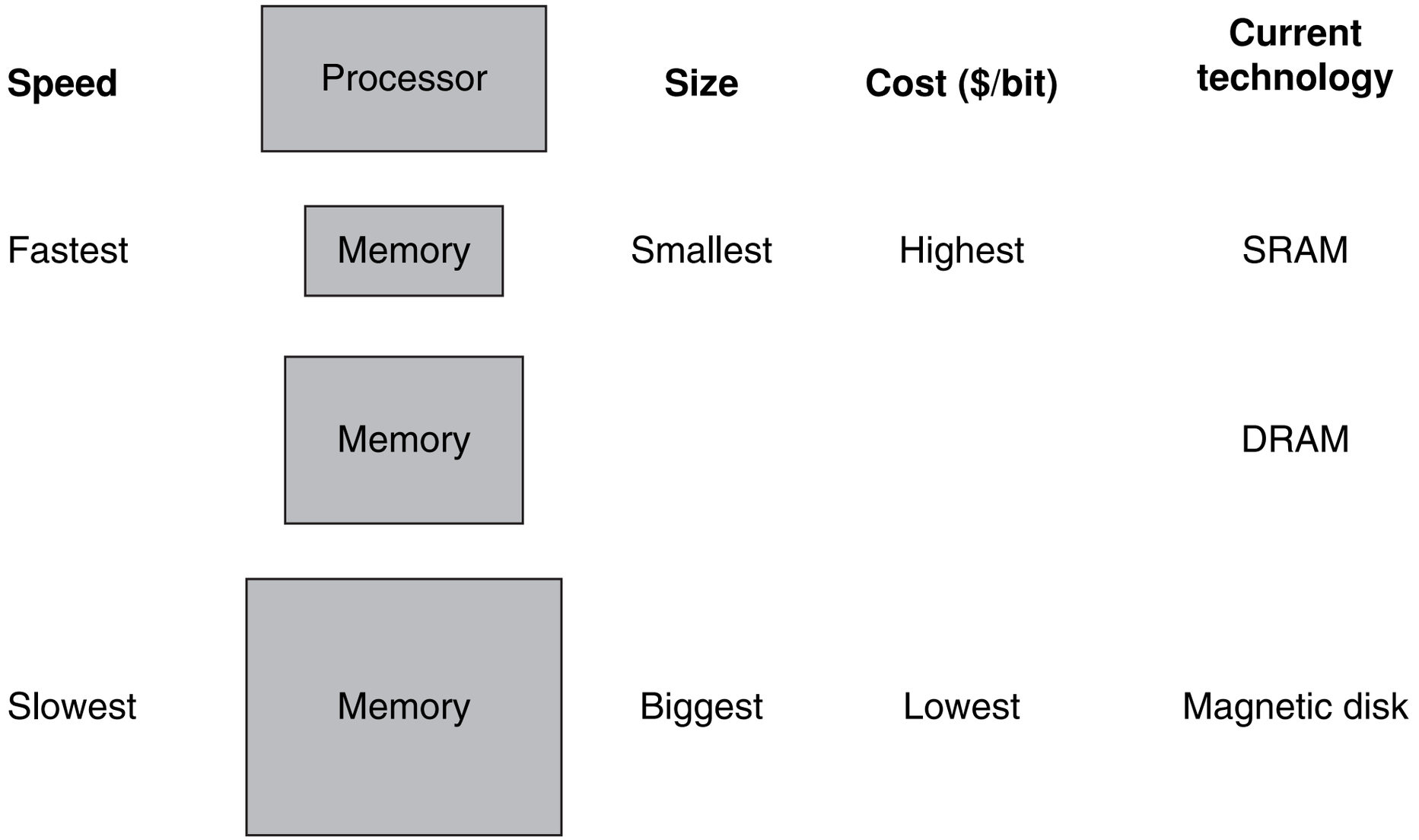
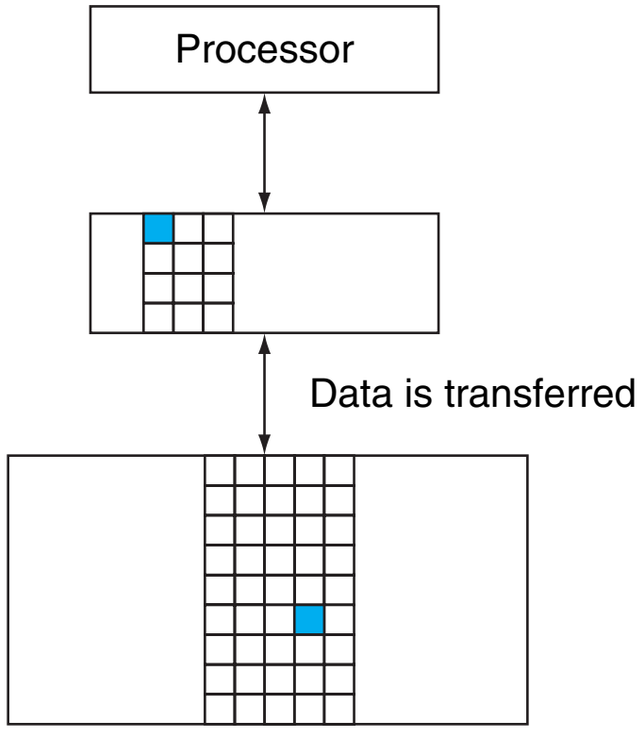


FIGURE 5.1 The basic structure of a memory hierarchy. By implementing the memory system as a hierarchy, the user has the illusion of a memory that is as large as the largest level of the hierarchy, but can be accessed as if it were all built from the fastest memory. Flash memory has replaced disks in many embedded devices, and may lead to a new level in the storage hierarchy for desktop and server computers; see Section 6.4. Copyright © 2009 Elsevier, Inc. All rights reserved.

Memory Hierarchy Levels

Block (aka line): unit of copying, may be multiple words



→ If accessed data is present in upper level
- **Hit:** access satisfied by upper level
- **Hit ratio:** hits/accesses

→ If accessed data is absent
- **Miss:** block copied from lower level
- **Time taken:** miss penalty
- **Miss ratio:** misses

→ Accessed data then supplied from upper level

FIGURE 5.2 Every pair of levels in the memory hierarchy can be thought of as having an upper and lower level. Within each level, the unit of information that is present or not is called a *block* or a *line*. Usually we transfer an entire block when we copy something between levels. Copyright © 2009 Elsevier, Inc. All rights reserved.

How do we know if the data is present?
Where do we look?

A General Memory Hierarchy

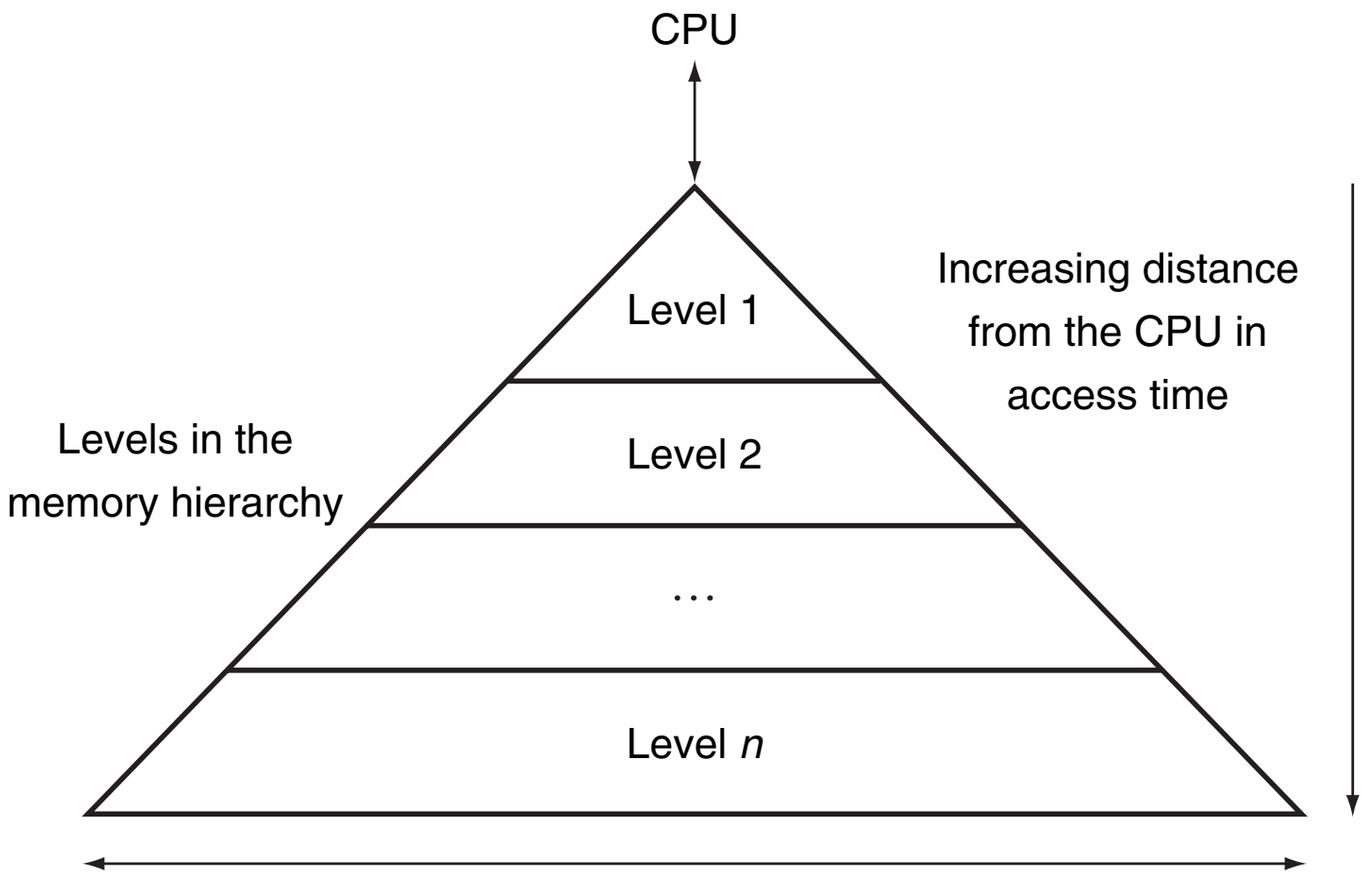
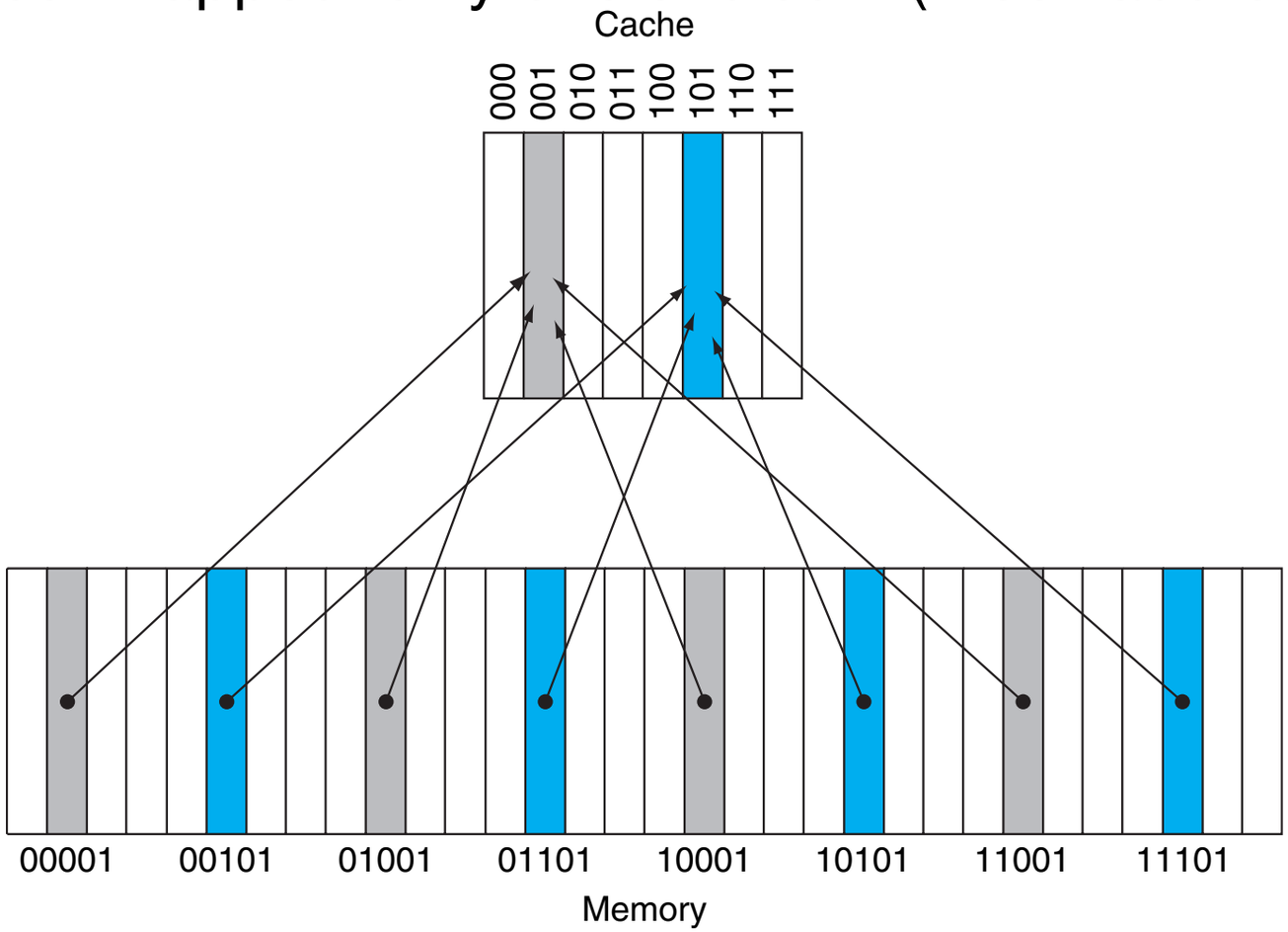


FIGURE 5.3 This diagram shows the structure of a memory hierarchy: as the distance from the processor increases, so does the size. This structure, with the appropriate operating mechanisms, allows the processor to have an access time that is determined primarily by level 1 of the hierarchy and yet have a memory as large as level *n*. Maintaining this illusion is the subject of this chapter. Although the local disk is normally the bottom of the hierarchy, some systems use tape or a file server over a local area network as the next levels of the hierarchy.

Direct Mapped Cache

- Location determined by address

- Direct mapped: only one choice = (Block address) modulo (#Blocks in cache)



If a power of two, use low order address bits

FIGURE 5.5 A direct-mapped cache with eight entries showing the addresses of memory words between 0 and 31 that map to the same cache locations. Because there are eight words in the cache, an address X maps to the direct-mapped cache word X modulo 8. That is, the low-order $\log_2(8) = 3$ bits are used as the cache index. Thus, addresses 00001_{two}, 01001_{two}, 10001_{two}, and 11001_{two} all map to entry 001_{two} of the cache, while addresses 00101_{two}, 01101_{two}, 10101_{two}, and 11101_{two} all map to entry 101_{two} of the cache.

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
    - Store block address as well as the data
    - Actually, only need the high-order bits
    - Called the tag
- What if there is no data in a location?
    - Valid bit: 1 = present, 0 = not present
    - Initially 0

Address Subdivision

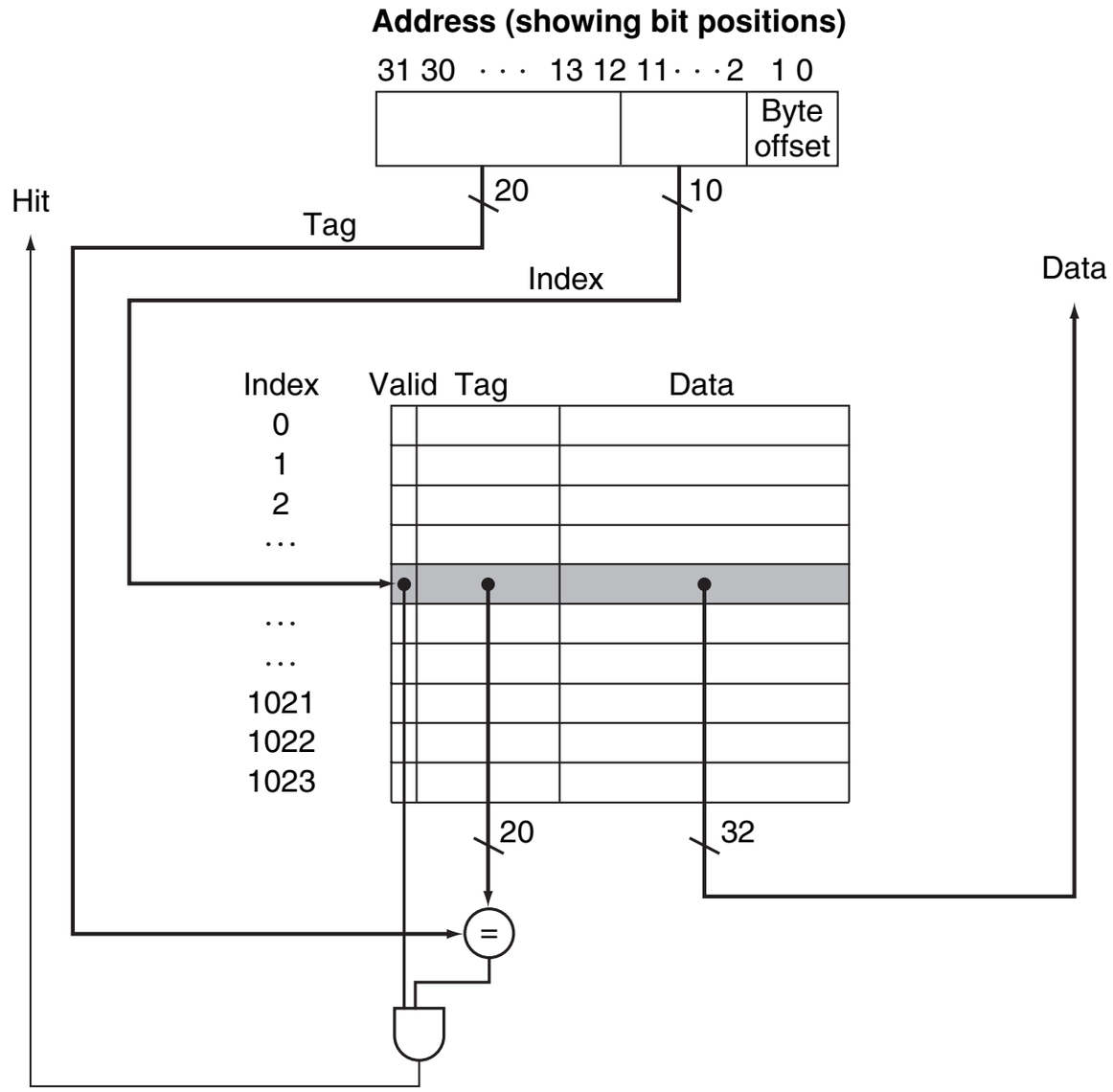


FIGURE 5.7 For this cache, the lower portion of the address is used to select a cache entry consisting of a data word and a tag. This cache holds 1024 words or 4 KB. We assume 32-bit addresses in this chapter. The tag from the cache is compared against the upper portion of the address to determine whether the entry in the cache corresponds to the requested address. Because the cache has 2^{10} (or 1024) words and a block size of one word, 10 bits are used to index the cache, leaving $32 - 10 - 2 = 20$ bits to be compared against the tag. If the tag and upper 20 bits of the address are equal and the valid bit is on, then the request hits in the cache, and the word is supplied to the processor. Otherwise, a miss occurs.

Cache Example 1

8-block cache, 1 word/block, 32B memory

Index	V	Tag	Data
0 0 0	0		
0 0 1	0		
0 1 0	0		
0 1 1	0		
1 0 0	0		
1 0 1	0		
1 1 0	0		
1 1 1	0		

Initial state after power on

Cache Example 2

Index	V	Tag	Data
0 0 0	0		
0 0 1	0		
0 1 0	0		
0 1 1	0		
1 0 0	0		
1 0 1	0		
1 1 0	1	1 0	Mem[10110]
1 1 1	0		

After handling miss of address 10110

Cache Example 3

Index	V	Tag	Data
0 0 0	0		
0 0 1	0		
0 1 0	1	1 1	Mem[11010]
0 1 1	0		
1 0 0	0		
1 0 1	0		
1 1 0	1	1 0	Mem[10110]
1 1 1	0		

After handling miss of address 11010

Cache Example 4

Index	V	Tag	Data
0 0 0	1	1 0	Mem[10000]
0 0 1	0		
0 1 0	1	1 1	Mem[11010]
0 1 1	0		
1 0 0	0		
1 0 1	0		
1 1 0	1	1 0	Mem[10110]
1 1 1	0		

After handling miss of address 10000

Cache Example 5

Index	V	Tag	Data
0 0 0	1	1 0	Mem[10000]
0 0 1	0		
0 1 0	1	1 1	Mem[11010]
0 1 1	1	0 0	Mem[00011]
1 0 0	0		
1 0 1	0		
1 1 0	1	1 0	Mem[10110]
1 1 1	0		

After handling miss of address 00011

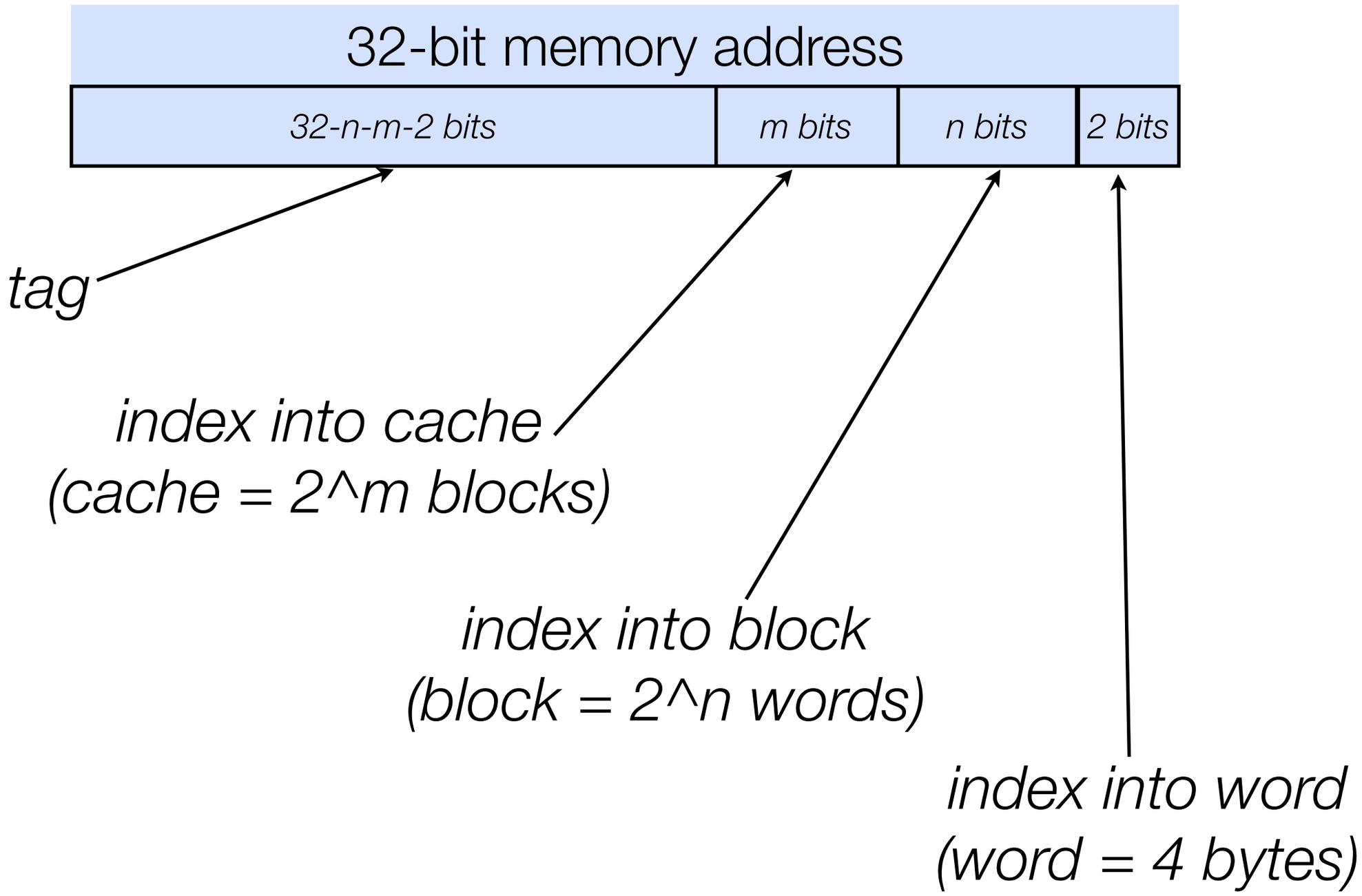
Cache Example 6

Index
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

V	Tag	Data
1	1 0	Mem[10000]
0		
1	1 0	Mem[10010]
1	0 0	Mem[00011]
0		
0		
1	1 0	Mem[10110]
0		

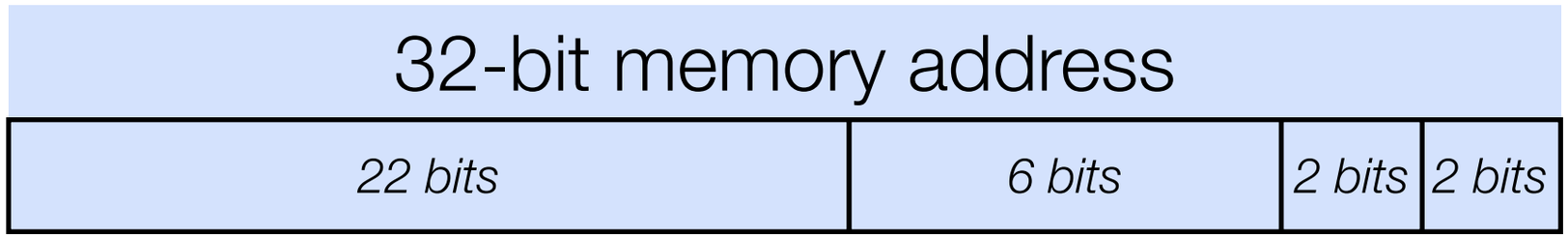
After handling miss of address 10010

Multi-word Cache Blocks



Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
- Block address = floor(1200/16) = 75
- Block number = 75 modulo 64 = 11



Block Size Considerations

- Larger blocks should reduce miss rate, due to spatial locality
- But in a fixed-sized cache
  - Larger blocks → fewer of them → more competition → increased miss rate
  - Larger blocks → pollution
- Larger miss penalty, which could override benefit of reduced miss rate

Handling Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - After Instruction cache miss, restart instruction fetch
  - After data cache miss, complete data access

Handling Writes: Write Through

- On data-write hit, could just update the block in cache, but then cache and memory would be inconsistent

- **Write through**: on write, update memory as well as cache

  - Makes writes take longer (e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles, effective CPI = 1 + 0.1×100 = 11)

  - Solution: write buffer which holds data waiting to be written to memory. CPU can now continue immediately, stalling only if write buffer is full.

Handling Writes: Write Back

- An alternative to write through
- **Write through**: on data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
  - When a dirty block is replaced, write it back to memory
  - Can use a write buffer to allow replacing block to be read first

Example: Intrinsity FastMATH

- Embedded MIPS processor
    - 12-stage pipeline
    - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
- Each 16KB: 256 blocks × 16 words/block
- SPEC2000 miss rates
    - I-cache: 0.4%
    - D-cache: 11.4%
    - Weighted average: 3.2%

Computer Architecture, Then and Now and Here

History of Processor Performance

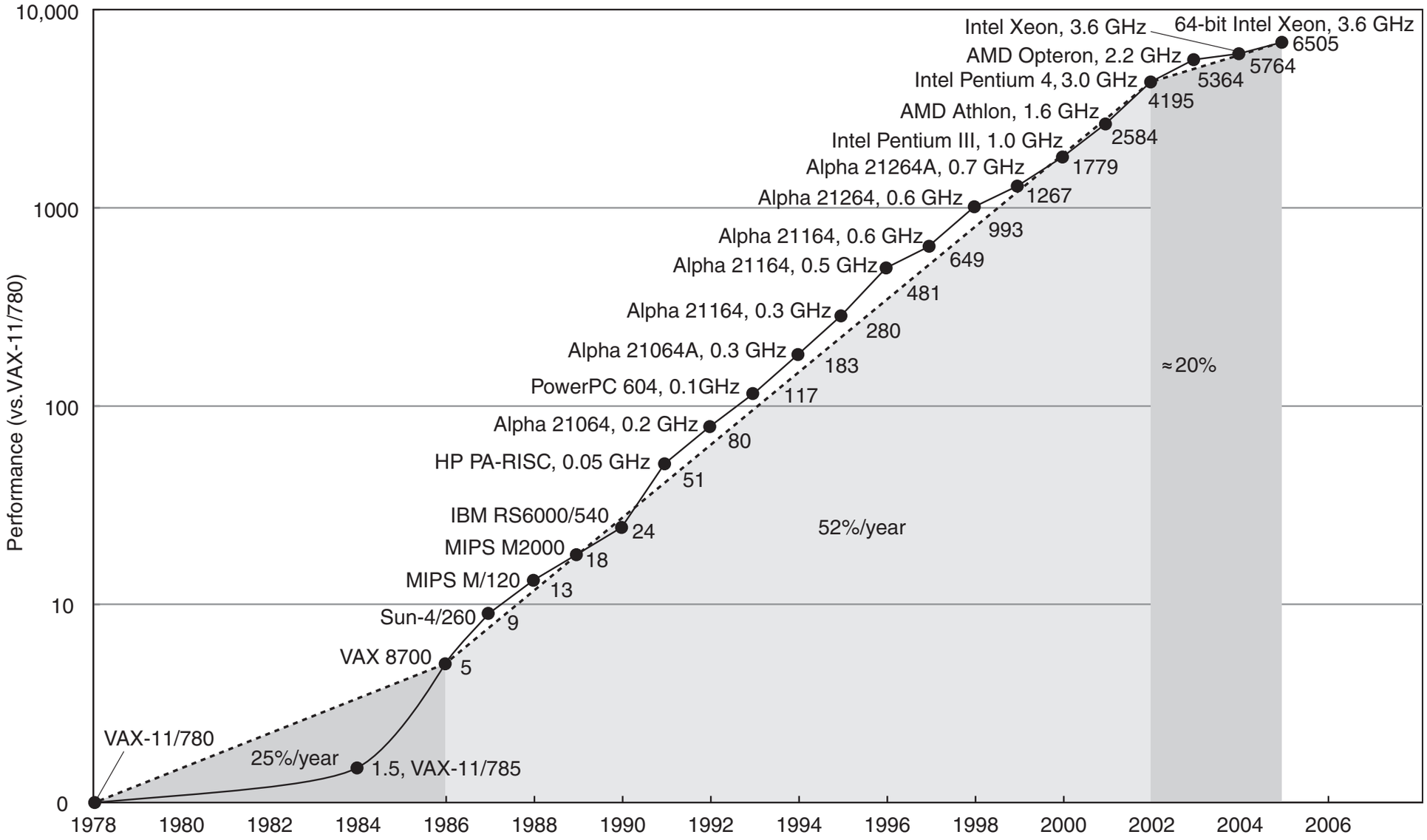


FIGURE 1.16 Growth in processor performance since the mid-1980s. This chart plots performance relative to the VAX 11/780 as measured by the SPECint benchmarks (see Section 1.8). Prior to the mid-1980s, processor performance growth was largely technology-driven and averaged about 25% per year. The increase in growth to about 52% since then is attributable to more advanced architectural and organizational ideas. By 2002, this growth led to a difference in performance of about a factor of seven. Performance for floating-point-oriented calculations has increased even faster. Since 2002, the limits of power, available instruction-level parallelism, and long memory latency have slowed uniprocessor performance recently, to about 20% per year.

History of Processor Performance

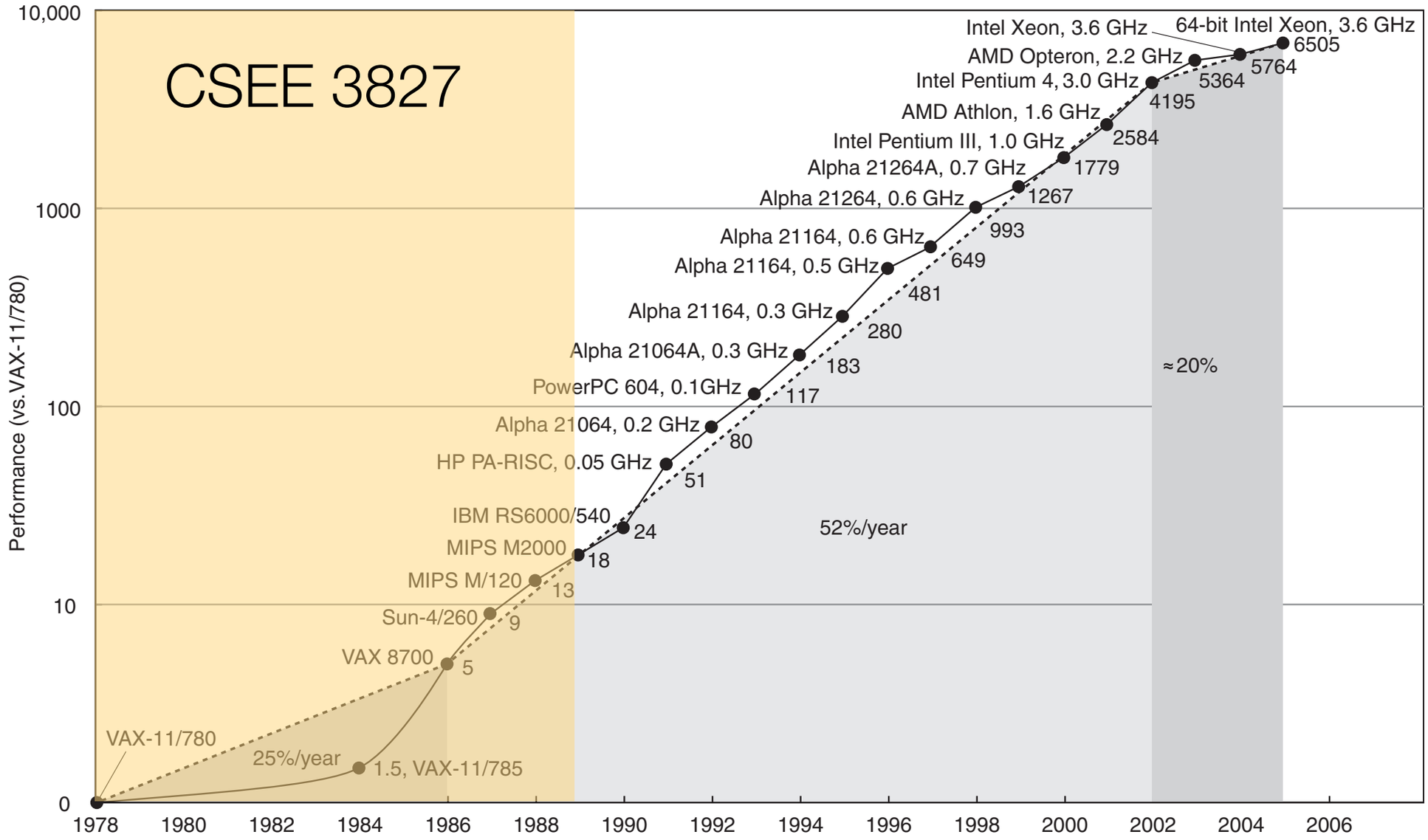


FIGURE 1.16 Growth in processor performance since the mid-1980s. This chart plots performance relative to the VAX 11/780 as measured by the SPECint benchmarks (see Section 1.8). Prior to the mid-1980s, processor performance growth was largely technology-driven and averaged about 25% per year. The increase in growth to about 52% since then is attributable to more advanced architectural and organizational ideas. By 2002, this growth led to a difference in performance of about a factor of seven. Performance for floating-point-oriented calculations has increased even faster. Since 2002, the limits of power, available instruction-level parallelism, and long memory latency have slowed uniprocessor performance recently, to about 20% per year.

History of Processor Performance

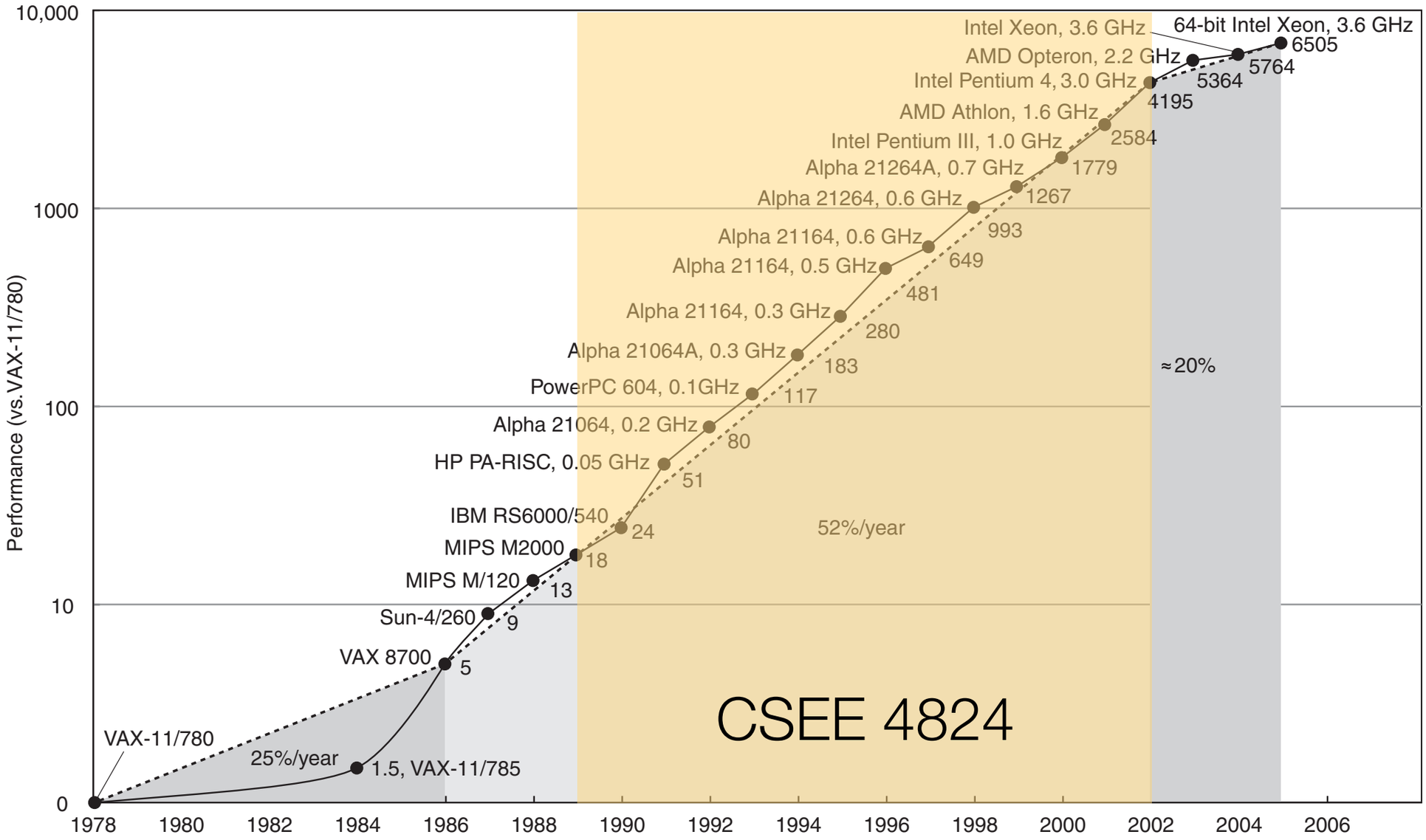


FIGURE 1.16 Growth in processor performance since the mid-1980s. This chart plots performance relative to the VAX 11/780 as measured by the SPECint benchmarks (see Section 1.8). Prior to the mid-1980s, processor performance growth was largely technology-driven and averaged about 25% per year. The increase in growth to about 52% since then is attributable to more advanced architectural and organizational ideas. By 2002, this growth led to a difference in performance of about a factor of seven. Performance for floating-point-oriented calculations has increased even faster. Since 2002, the limits of power, available instruction-level parallelism, and long memory latency have slowed uniprocessor performance recently, to about 20% per year.

The Power Wall

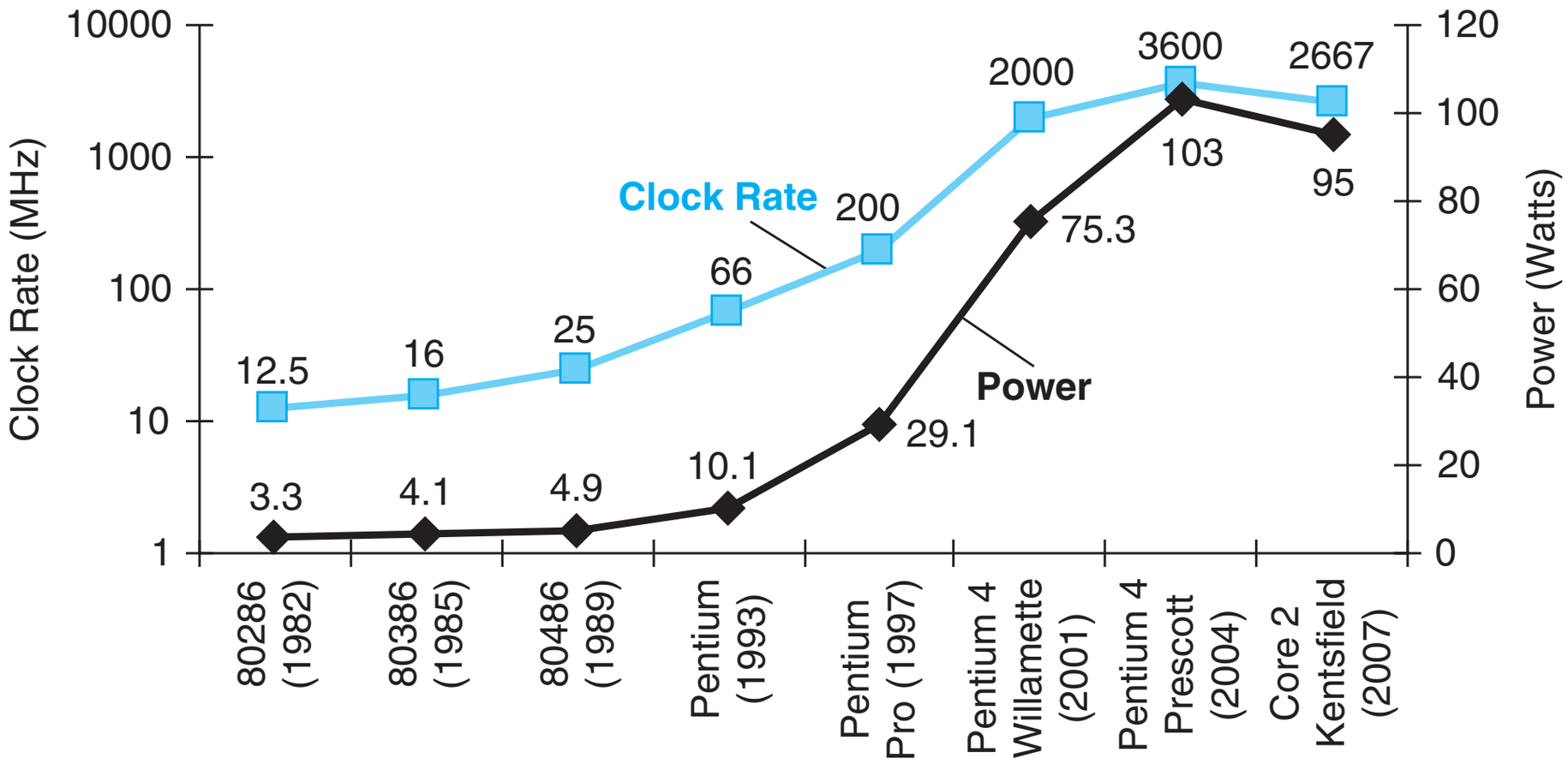


FIGURE 1.15 Clock rate and Power for Intel x86 microprocessors over eight generations and 25 years. The Pentium 4 made a dramatic jump in clock rate and power but less so in performance. The Prescott thermal problems led to the abandonment of the Pentium 4 line. The Core 2 line reverts to a simpler pipeline with lower clock rates and multiple processors per chip.

Sea Change in Architecture: Multicore

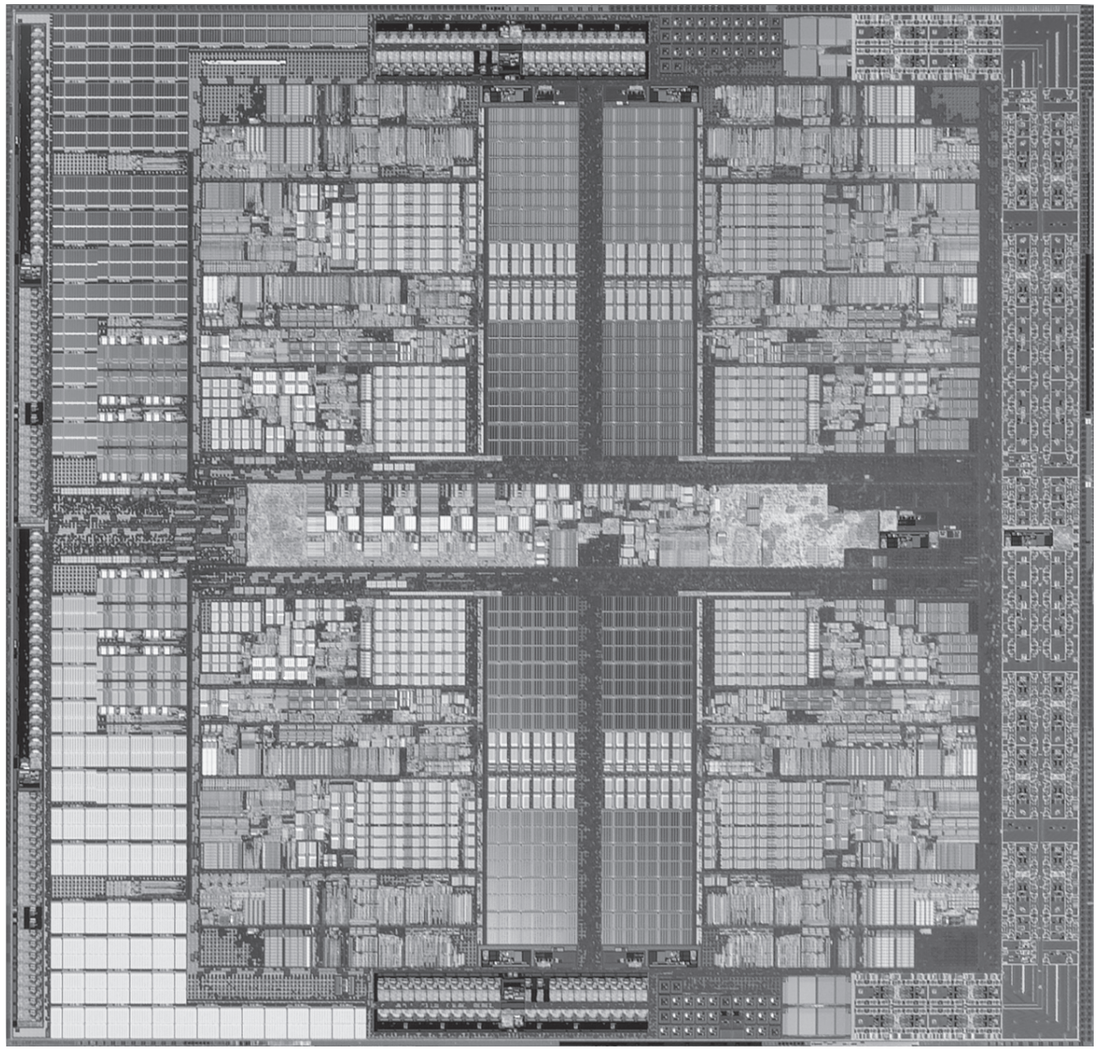


FIGURE 1.9 Inside the AMD Barcelona microprocessor. The left-hand side is a microphotograph of the AMD Barcelona processor chip, and the right-hand side shows the major blocks in the processor. This chip has four processors or "cores". The microprocessor in the laptop in Figure 1.7 has two cores per chip, called an Intel Core 2 Duo.

Modern Processor Performance

While single threaded performance has leveled, multithreaded performance potential scaling.

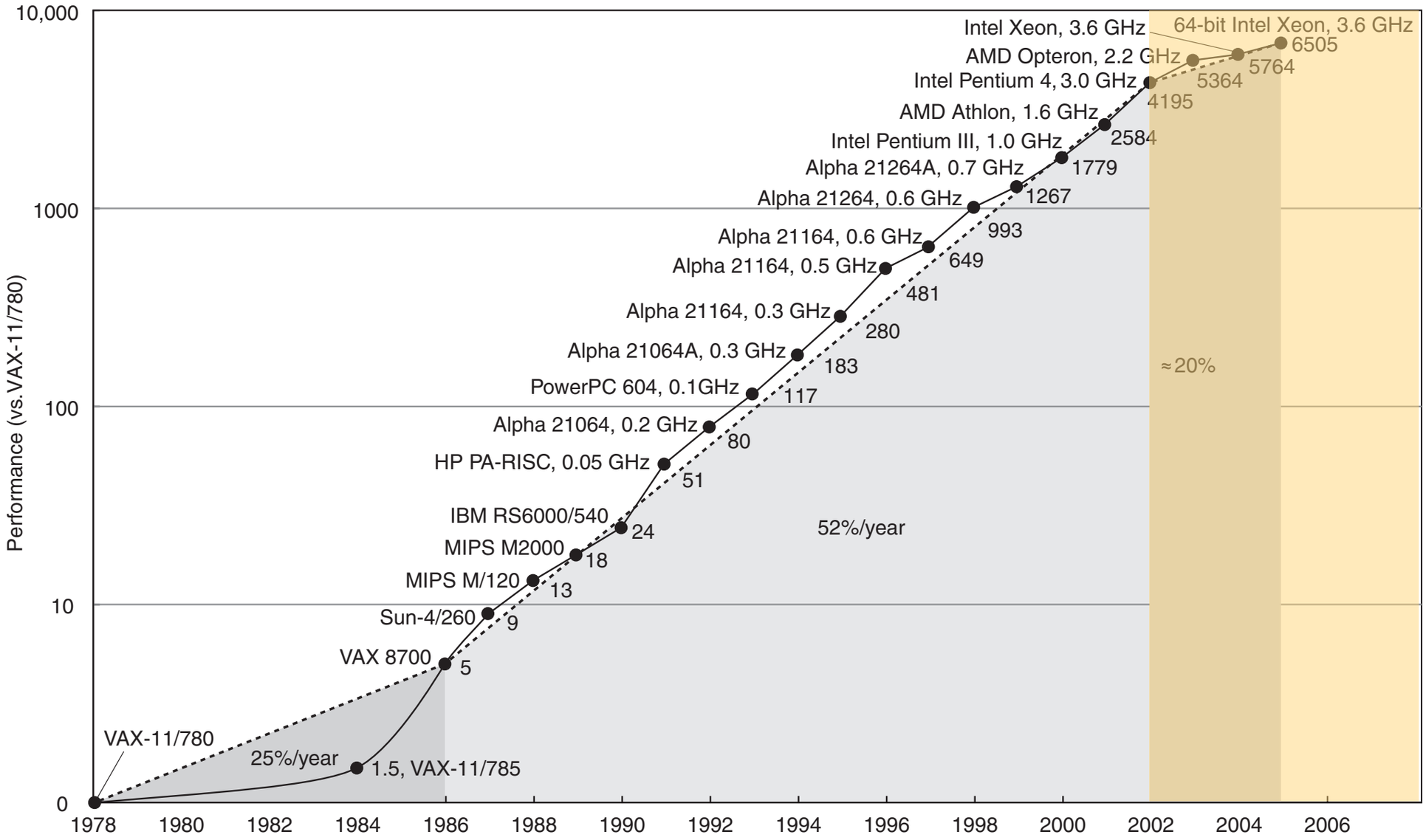


FIGURE 1.16 Growth in processor performance since the mid-1980s. This chart plots performance relative to the VAX 11/780 as measured by the SPECint benchmarks (see Section 1.8). Prior to the mid-1980s, processor performance growth was largely technology-driven and averaged about 25% per year. The increase in growth to about 52% since then is attributable to more advanced architectural and organizational ideas. By 2002, this growth led to a difference in performance of about a factor of seven. Performance for floating-point-oriented calculations has increased even faster. Since 2002, the limits of power, available instruction-level parallelism, and long memory latency have slowed uniprocessor performance recently, to about 20% per year.