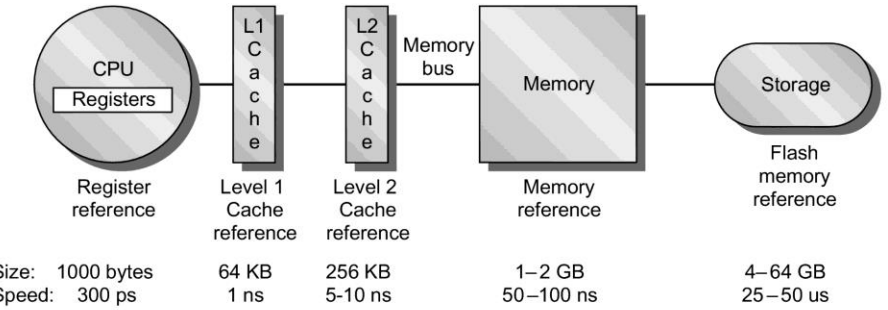
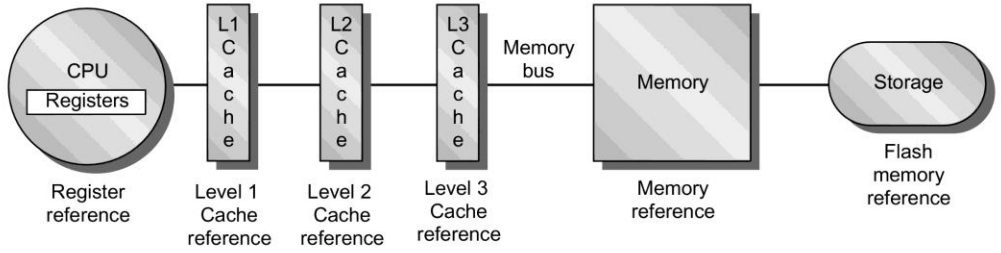
Memory Hierarchy Design

- Introduction
- Eleven Advanced Cache Optimizations
- Memory Technology and Optimizations
- Virtual Memory and Virtual Machines
- ARM Cortex-A53 Intel i7 Memory Hierarchy
- Conclusion

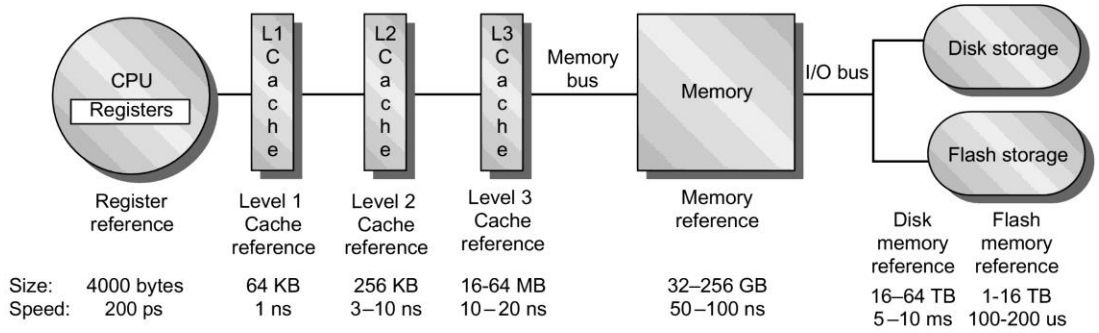
Memory Hierarchy



(A) Memory hierarchy for a personal mobile device

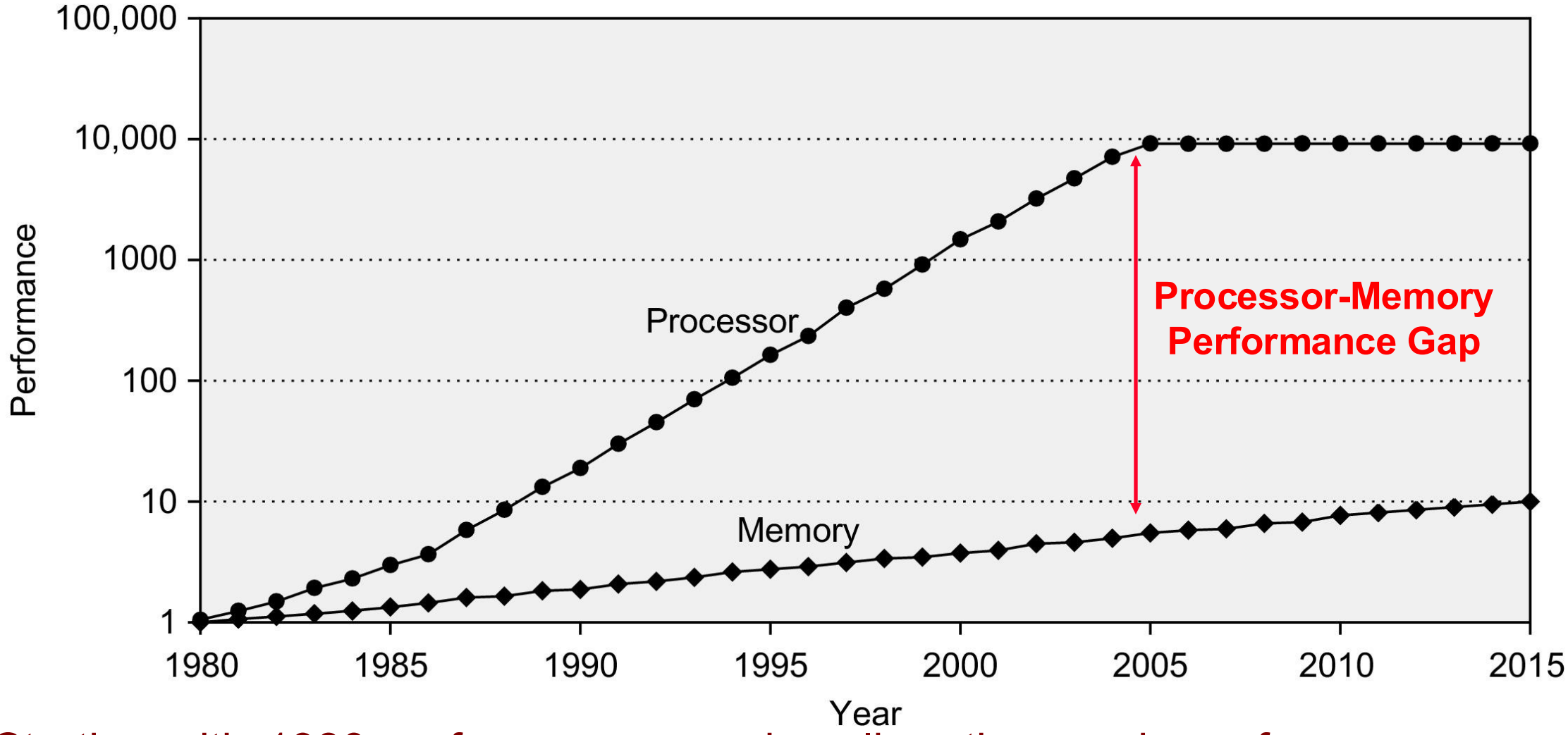


(B) Memory hierarchy for a laptop or a desktop



(C) Memory hierarchy for server

Why More on Memory Hierarchy?



Starting with 1980 performance as a baseline, the gap in performance, measured as the difference in the time between processor memory requests (for a single processor or core) and the latency of a DRAM access, is plotted.

Four Types of Misses

- Compulsory
  - During a program, the very first access to a block will not be in the cache (unless pre-fetched)
- Capacity
  - The working set of blocks accessed by the program is too large to fit in the cache
- Conflict
  - Unless cache is fully associative, sometimes blocks may be evicted too early because too many frequently-accessed blocks map to the same limited set of frames.
- Coherence
  - True and false sharing misses

Cache Performance Improvement

$$\text{(Average memory access time)} = \text{(Hit time)} + \underbrace{\text{(Miss rate)}\times\text{(Miss penalty)}}_{\text{“Amortized miss penalty”}}$$

- Reduce miss penalty:
  - Multilevel cache; Critical word first and early restart; priority to read miss; Merging write buffer; Victim cache
- Reduce miss rate:
  - Larger block size; Increase cache size; Higher associativity; Way prediction and Pseudo-associative caches; Compiler optimizations
- Reduce miss penalty/rate via parallelism:
  - Non-blocking cache; Hardware and Compiler-controlled prefetching
- Reduce hit time:
  - Small simple cache; Avoid address translation in indexing cache; Pipelined cache access; Trace caches

Six Basic Cache Optimizations

- Reducing hit time
  1. Giving Reads Priority over Writes
  2. Avoiding Address Translation during Cache Indexing
- Reducing Miss Penalty
  3. Multilevel Caches
- Reducing Miss Rate
  4. Larger Block size (Compulsory misses)
  5. Larger Cache size (Capacity misses)
  6. Higher Associativity (Conflict misses)

Memory Hierarchy Design

- Introduction
- Eleven Advanced Cache Optimizations
- Memory Technology and Optimizations
- Virtual Memory and Virtual Machines
- ARM Cortex-A53 Intel i7 Memory Hierarchy
- Conclusion

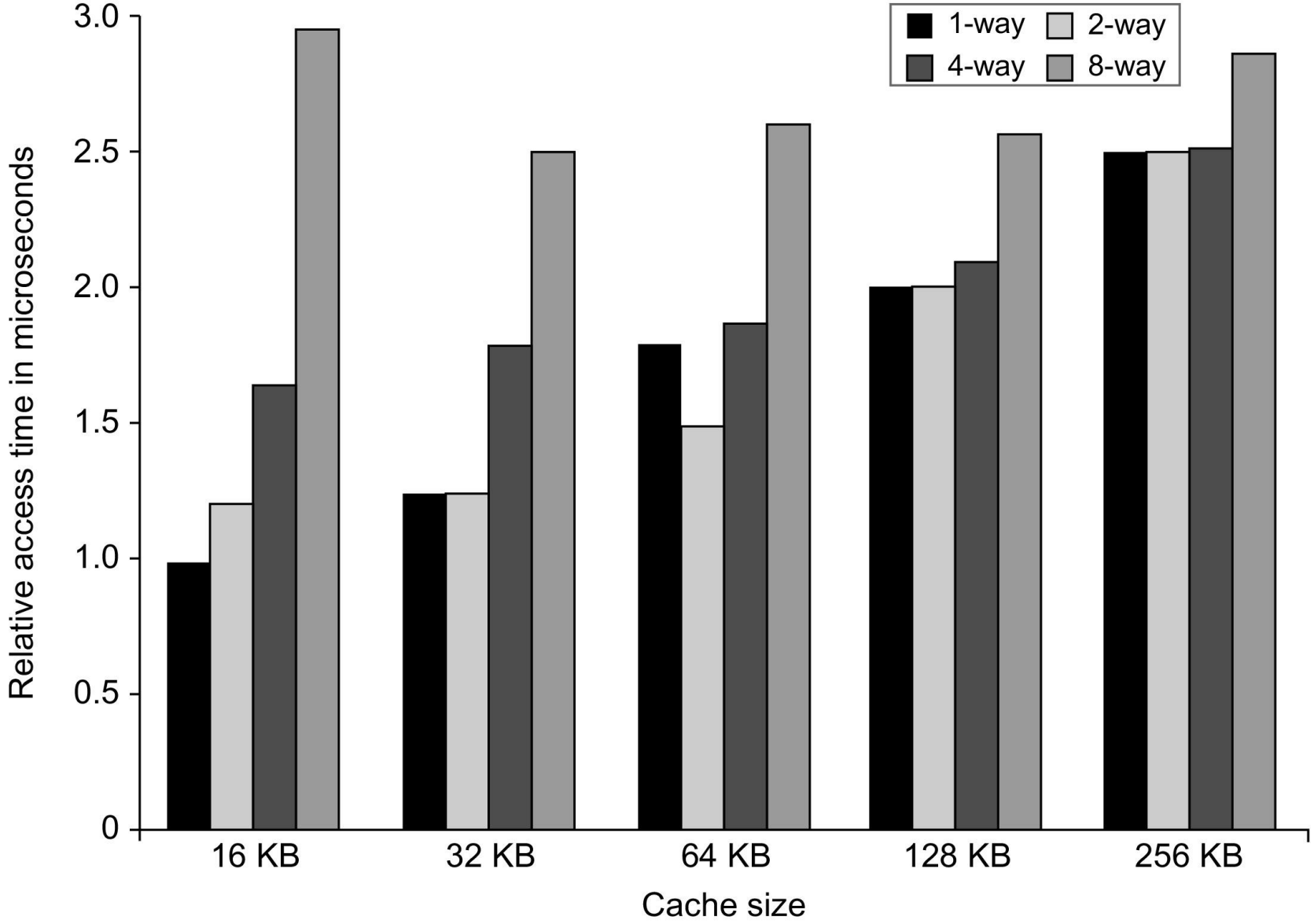
11 Advanced Cache Optimizations

- Reducing hit time
  1. Small and simple caches
  2. Way prediction
  3. Trace caches
- Increasing cache bandwidth
  4. Pipelined caches
  5. Multibanked caches
  6. Nonblocking caches
- Reducing Miss Penalty
  7. Critical word first
  8. Merging write buffers
- Reducing Miss Rate
  9. Compiler optimizations
- Reducing miss penalty or miss rate via parallelism
  10. Hardware prefetching
  11. Compiler prefetching

Small and Simple Caches

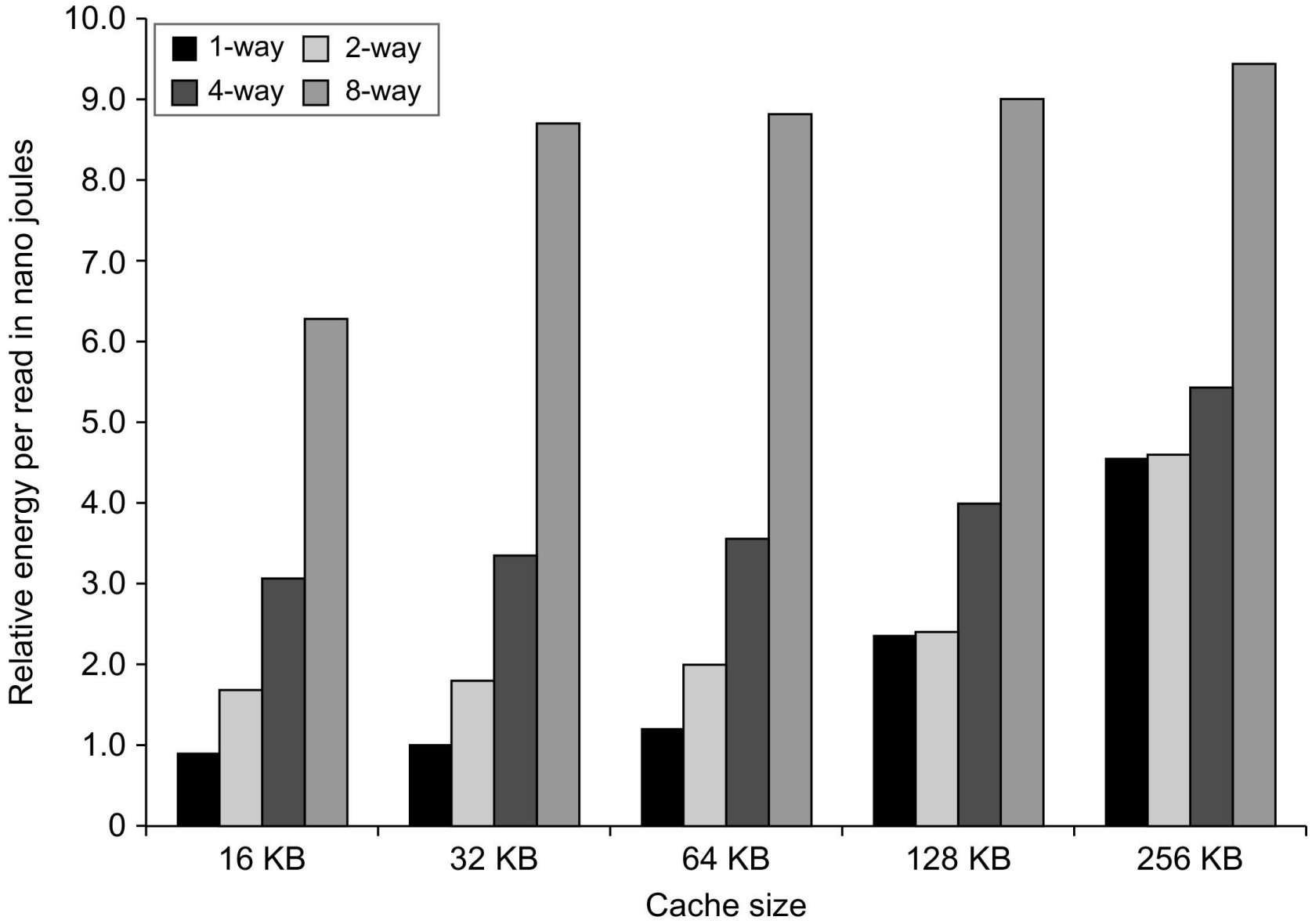
- Make cache smaller to improve hit time
  - Add a new & smaller L0 cache between existing L1 cache and CPU.
- Keep L1 cache on same chip as CPU
  - Physically close to functional units that access it
- Keep L1 design simple, e.g., direct-mapped
  - Avoids multiple tag comparisons
  - Tag can be compared after data cache fetch
    - Reduces effective hit time

Access Time in a SRAM Cache



Relative access times increase as cache size and associativity are increased.

Energy Consumption by SRAM Cache



Energy consumption increases as cache size and associativity are increased.

Way Prediction

- Keep in each set a way-prediction information to predict the block in each set will be accessed next
- Only one tag may be matched at the first cycle
  - If miss, other blocks need to be examined
- Beneficial in two aspects
  - Fast data access
    - Access data without knowing the tag comparison results
  - Low power
    - Match a single tag; if majority of the prediction is correct
- Different systems use variations of the concept

Trace Caches

- Supply enough instructions per cycle without dependencies
  - Finding ILP beyond 4 instructions per cycle
- Do not limit instructions in a cache block to spatial locality
  - Find dynamic sequence of instructions including taken branches
- NetBurst (P4) uses trace caches
  - Addresses are no longer aligned.
- Same instruction is stored more than once
  - If part of multiple traces

11 Advanced Cache Optimizations

- Reducing hit time
  1. Small and simple caches
  2. Way prediction
  3. Trace caches
- Increasing cache bandwidth
  4. Pipelined caches
  5. Multibanked caches
  6. Nonblocking caches
- Reducing Miss Penalty
  7. Critical word first
  8. Merging write buffers
- Reducing Miss Rate
  9. Compiler optimizations
- Reducing miss penalty or miss rate via parallelism
  10. Hardware prefetching
  11. Compiler prefetching

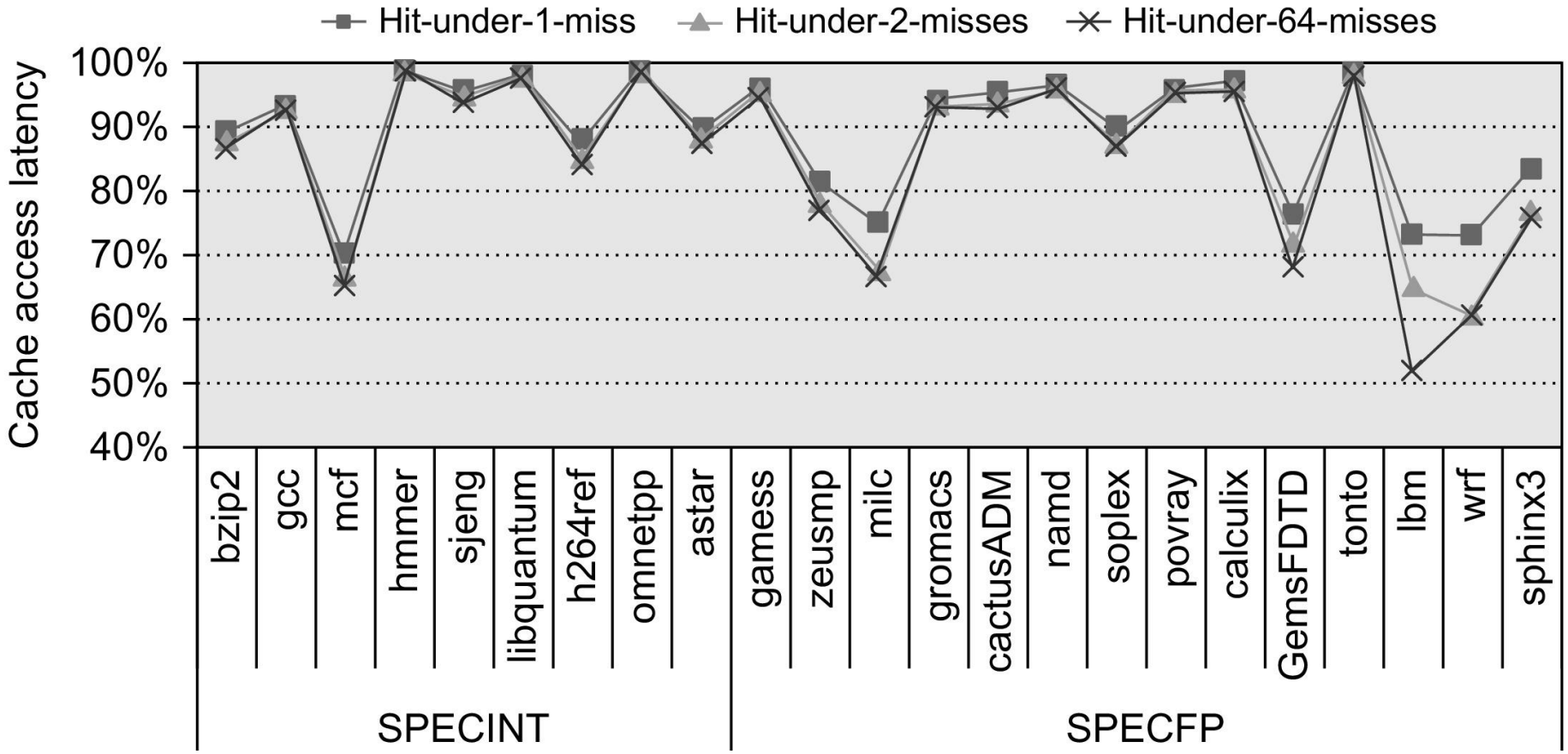
Pipelined Cache Access

- Pipeline cache access so that
  - Effective latency of first level cache hit can be multiple clock cycles
    - Fast cycle time and slow hits
    - Hit times: 1 for Pentium, 2 for P3, and 4 for P4
- Increases number of pipeline stages
  - Higher penalty on mispredicted branches
  - More cycles from issue of load to use of data
- In reality
  - Increases the bandwidth of instructions than decreasing the actual latency of a cache hit

Non-blocking Caches

- Known as lockup-free cache, hit under miss
- While a miss is being processed,
  - Allow *other* cache lookups to continue anyway
- Useful in dynamically scheduled CPUs
  - Other instructions may be in the load queue
- Reduces effective miss penalty
  - Useful CPU work fills the miss penalty “delay slot”
- Hit under multiple miss, miss under miss:
  - Extend technique to allow multiple misses to be queued up, while still processing new hits

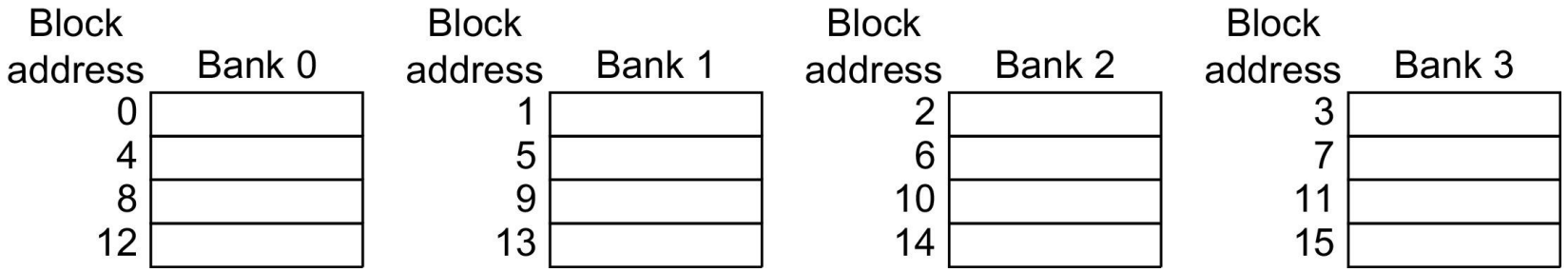
Non-blocking Caches



The data memory consists of a 32 KiB L1 with a 4-cycle access latency. The L2 is 256 KiB with a 10-clock cycle access latency. The L3 is 2 MiB and a 36-cycle access latency. All the caches are 8-way set associative and have a 64-byte block size. One hit under miss reduces the miss penalty by 9% for SPECINT and 12.5% for SPECFP. Allowing a second hit improves these results to 10% and 16%, allowing 64 results in little additional improvement.

Simple Interleaved Memory

- Adjacent words found in different memory banks
  - Banks can be accessed in parallel
  - Overlap latencies for accessing each word
- Can use narrow bus
  - To return accessed words sequentially
- Fits well with sequential access
  - *e.g.*, of words in cache blocks
- Independent memory banks
  - Allows multiple simultaneous cache misses
  - 4-way interleaved banks using block addressing



11 Advanced Cache Optimizations

- Reducing hit time
  1. Small and simple caches
  2. Way prediction
  3. Trace caches
- Increasing cache bandwidth
  4. Pipelined caches
  5. Multibanked caches
  6. Nonblocking caches
- Reducing Miss Penalty
  7. Critical word first
  8. Merging write buffers
- Reducing Miss Rate
  9. Compiler optimizations
- Reducing miss penalty or miss rate via parallelism
  10. Hardware prefetching
  11. Compiler prefetching

Early Restart, Critical Word First

- Early restart
  - Do not wait for entire block to fill
  - Resume CPU as soon as requested word is fetched
- Critical word first
  - Wrapped fetch, requested word first
  - Fetch the requested word from memory first
  - Resume CPU
  - Then transfer the rest of the cache block
- Most beneficial if block size is large
- Commonly used in all the processors

Early Restart, Critical Word First

Consider a cache with **32**-byte block (line) size and word size of **4** bytes. Assume that the bus size between the cache and main memory is **64** bits (**8** bytes). The time to transfer from the main memory to the cache is **16 cycles**. In other words, it takes **16 cycles** to transfer **8 bytes**. For the three pop-quiz questions, assume that the processor is requesting the **third word** in the cache block.

Merging Write Buffer

- A mechanism to help reduce write stalls
- On a write to memory, block address and data to be written are placed in a *write buffer*.
- CPU can continue immediately
  - Unless the write buffer is full.
- Write merging:
  - If the same block is written again before it has been flushed to memory, old contents are replaced with new contents.
  - Care must be taken to not violate memory consistency and proper write ordering

Write Merging Example

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	
	0		0		0		0	

The write buffer on top does not use write merging, the bottom one does. The four writes are merged into a single buffer entry with write merging; without it, the buffer is full even though three-fourths of each entry is wasted.

11 Advanced Cache Optimizations

- Reducing hit time
  1. Small and simple caches
  2. Way prediction
  3. Trace caches
- Increasing cache bandwidth
  4. Pipelined caches
  5. Multibanked caches
  6. Nonblocking caches
- Reducing Miss Penalty
  7. Critical word first
  8. Merging write buffers
- Reducing Miss Rate
  9. Compiler optimizations
- Reducing miss penalty or miss rate via parallelism
  10. Hardware prefetching
  11. Compiler prefetching

Compiler Optimizations

- Reorganize code to improve locality properties.
- “The hardware designer’s favorite solution.”
  - Requires no new hardware!
- Various techniques – Cache awareness:
  - Merging Arrays
  - Loop Interchange
  - Loop Fusion
  - Blocking (in multidimensional arrays)
  - Other source-to-source transformation techniques

Loop Interchange

- Original

```
for ( j = 0; j <100; j = j+1)
    for ( i = 0; i <5000; i = i+1)
        x[i][j] = 2 * x[i][j]
```

  - Skips through memory in strides of 100 words

- Modified

```
for ( i = 0; i <5000; i = i+1)
    for ( j = 0; j <100; j = j+1)
        x[i][j] = 2 * x[i][j]
```

  - Access all the words in a cache block before going to the next block.

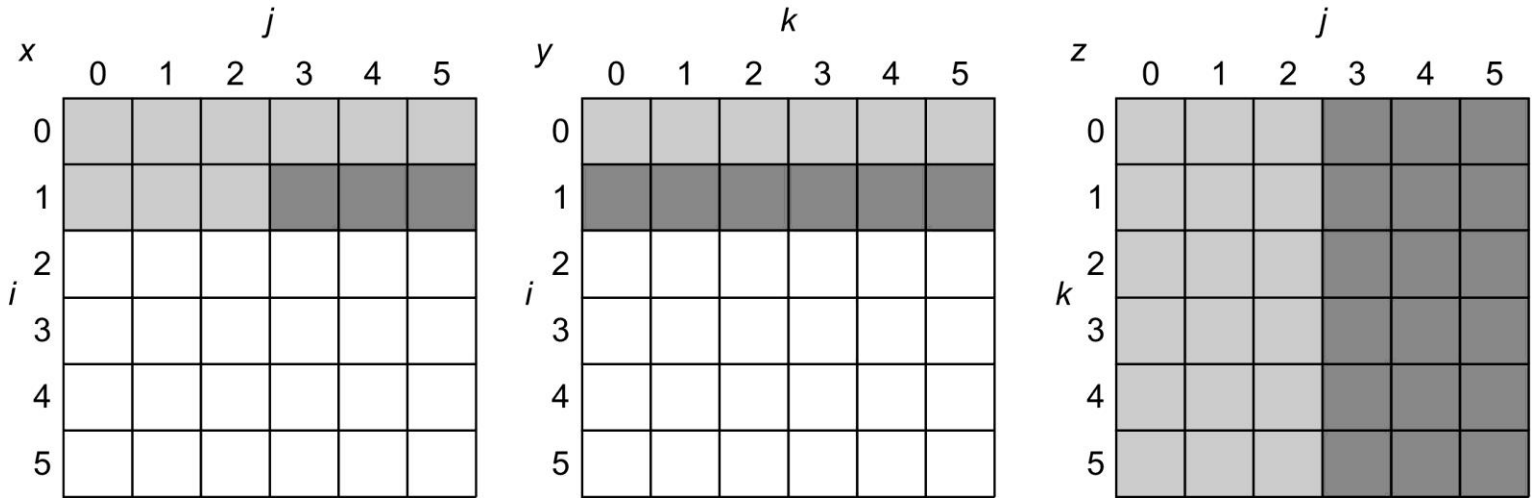
Blocking

- Loop interchange takes advantage of *row major* order (*column major* order) accesses.
  - What happens if we need to access both rows and columns in same iteration?

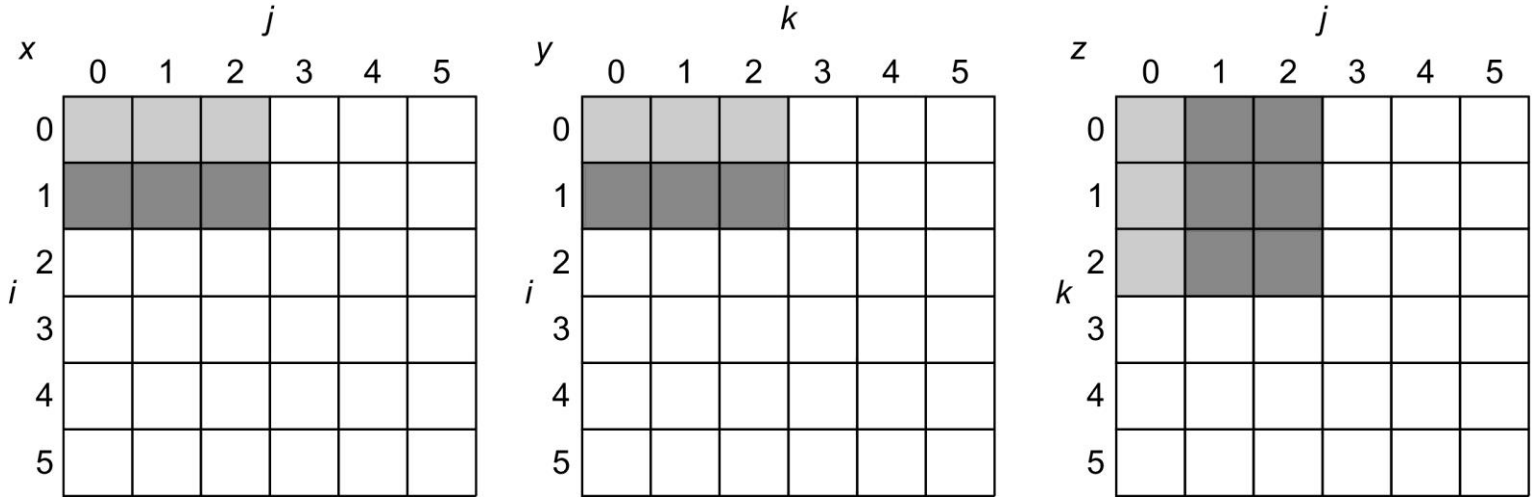
```
for ( i = 0; i <N; i = i+1)
    for ( j = 0; j <N; j = j+1)
    {
        r = 0;
      for (k = 0; k < N; k = k + 1)
        r = r + y[i][k] * z[k][j]
      x[i][j] = r;
     }
```

Loop Blocking – Matrix Multiply

Before (N=6 and i=1):



After (B=3):



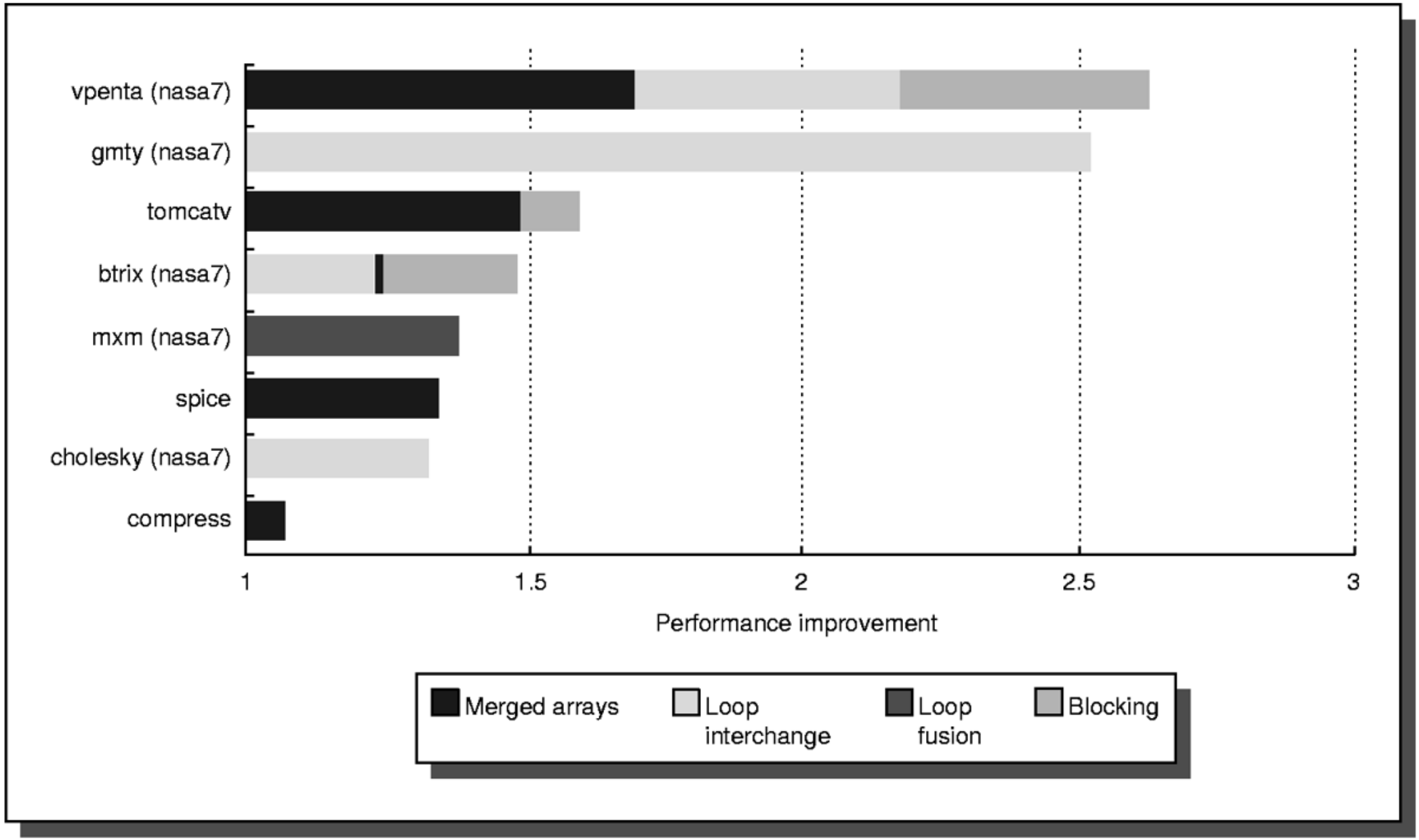
Code using Blocking

```
for ( jj = 0; jj <N; jj = jj+B)
for ( kk = 0; kk <N; kk = kk+B)
for ( i = 0; i <N; i = i+1)
     for ( j = jj; j < min (jj+B, N); j = j+1)
    {
         r = 0;
      for (k = kk; k < min(kk+B,N); k = k + 1)
         r = r + y[i][k] * z[k][j]
      x[i][j] = x[i][j] + r;
     }
```

y benefits from spatial locality

z benefits from temporal locality

Effect of Compiler Optimizations



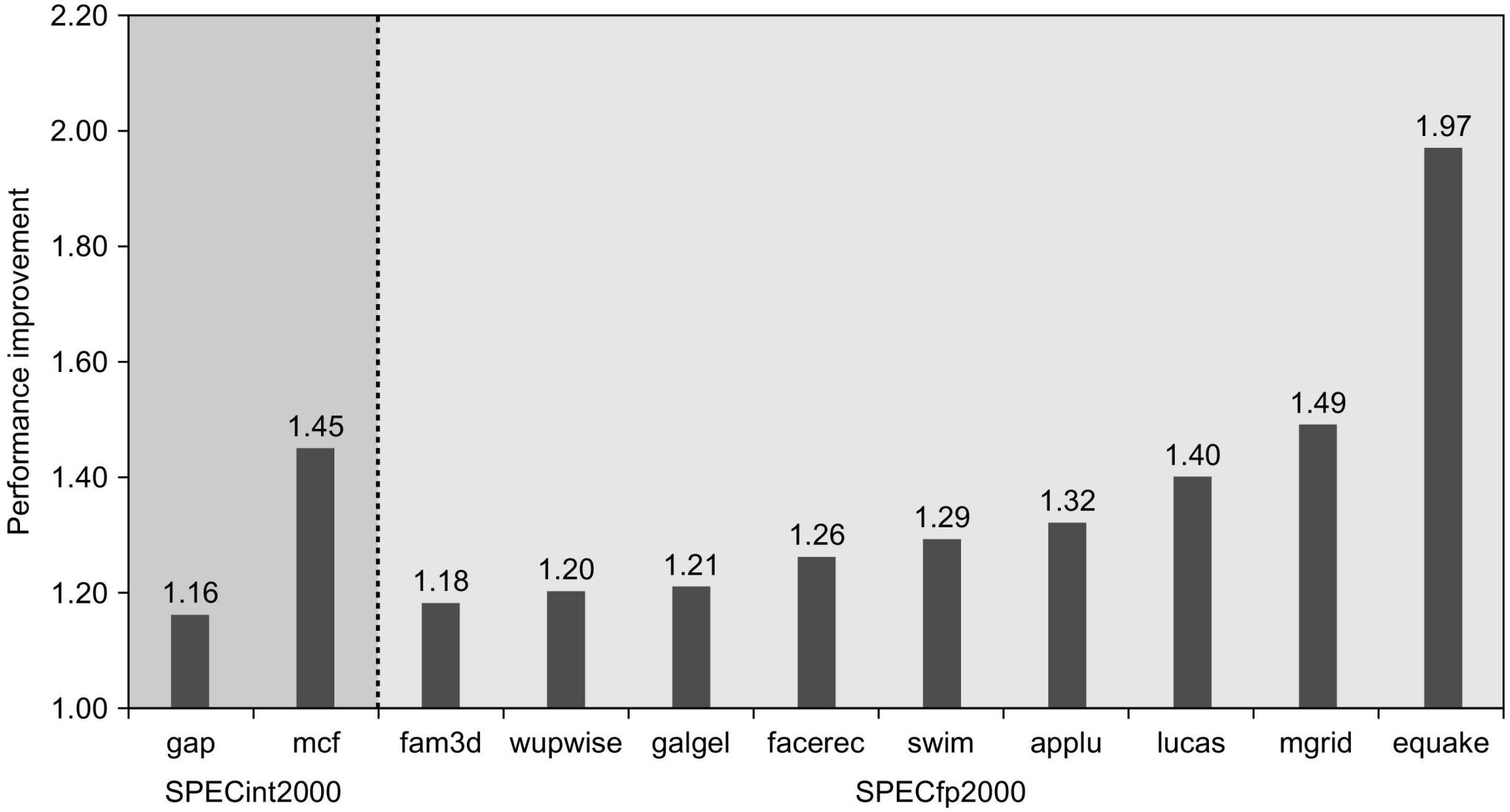
11 Advanced Cache Optimizations

- Reducing hit time
  1. Small and simple caches
  2. Way prediction
  3. Trace caches
- Increasing cache bandwidth
  4. Pipelined caches
  5. Multibanked caches
  6. Nonblocking caches
- Reducing Miss Penalty
  7. Critical word first
  8. Merging write buffers
- Reducing Miss Rate
  9. Compiler optimizations
- Reducing miss penalty or miss rate via parallelism
  10. Hardware prefetching
  11. Compiler prefetching

Hardware Prefetching

- When memory is idle, speculatively get some blocks *before* the CPU first asks for them!
  - Simple heuristic: Fetch 1 or more blocks that are consecutive to last one(s) fetched
  - Often, the extra blocks are placed in a special *stream buffer* so as not to conflict with actual active blocks in the cache, otherwise the prefetch may pollute the cache
- Pre-fetching can reduce misses considerably
- Speculative fetches should be low-priority
  - Use only otherwise-unused memory bandwidth
- Energy-inefficient (like all speculation)

Hardware Prefetching Speedup



Speedup of Intel Pentium 4 with hardware prefetching. Prefetching speeds up the remaining 15 SPEC CPU benchmarks by less than 15%.

Compiler-Controlled Prefetching

- Insert special instructions to load addresses from memory well before they are needed.
- Register vs. cache, faulting vs. nonfaulting
- Semantic invisibility, nonblocking-ness
- Can considerably reduce misses
- Can also *cause* extra conflict misses
  - Replacing a block before it is completely used
- Can also delay valid accesses (tying up bus)
  - Low-priority, can be pre-empted by real access

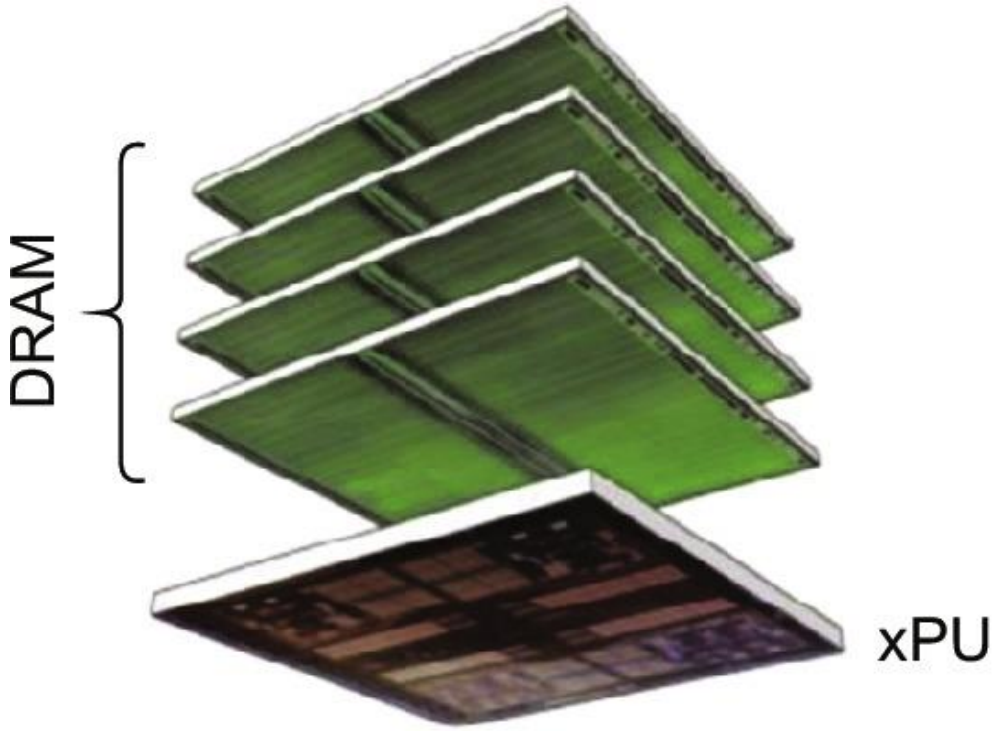
Summary of Memory Optimizations

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined & banked caches	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
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, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs
HBM as additional level of cache		+/–	–	+	+	3	Depends on new packaging technology. Effects depend heavily on hit rate improvements

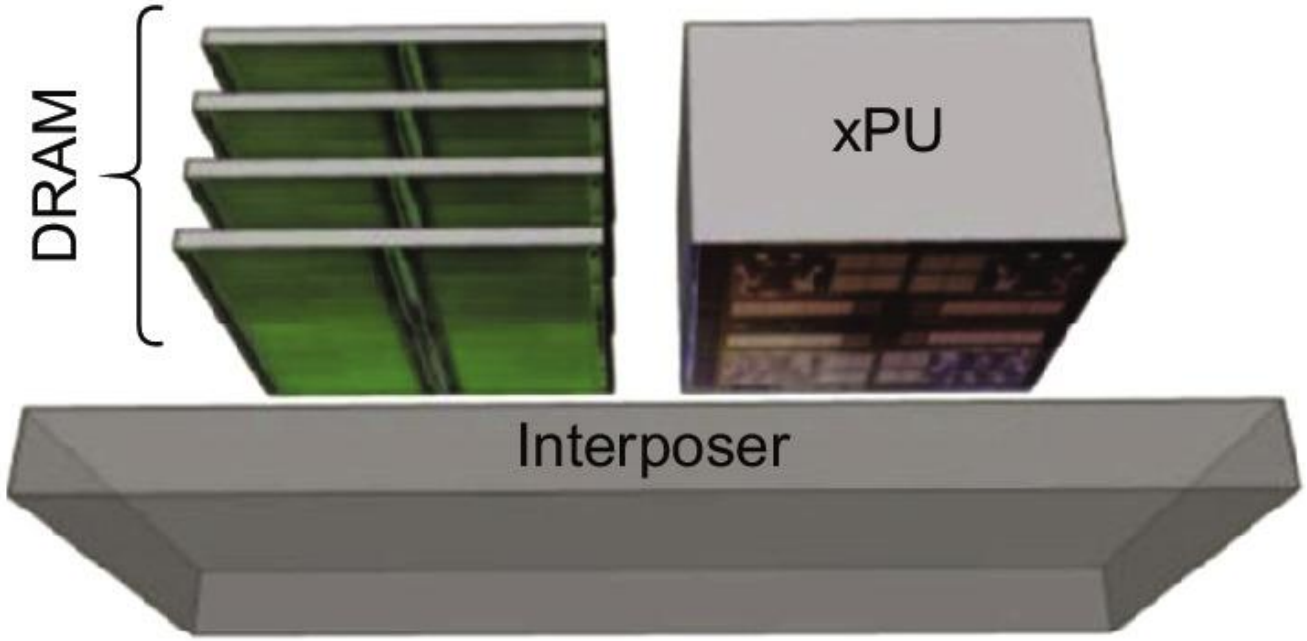
Memory Hierarchy Design

- Introduction
- Eleven Advanced Cache Optimizations
- Memory Technology and Optimizations
- Virtual Memory and Virtual Machines
- ARM Cortex-A53 Intel i7 Memory Hierarchy
- Conclusion

Two Forms of Stacking



Vertical stacking (3D)



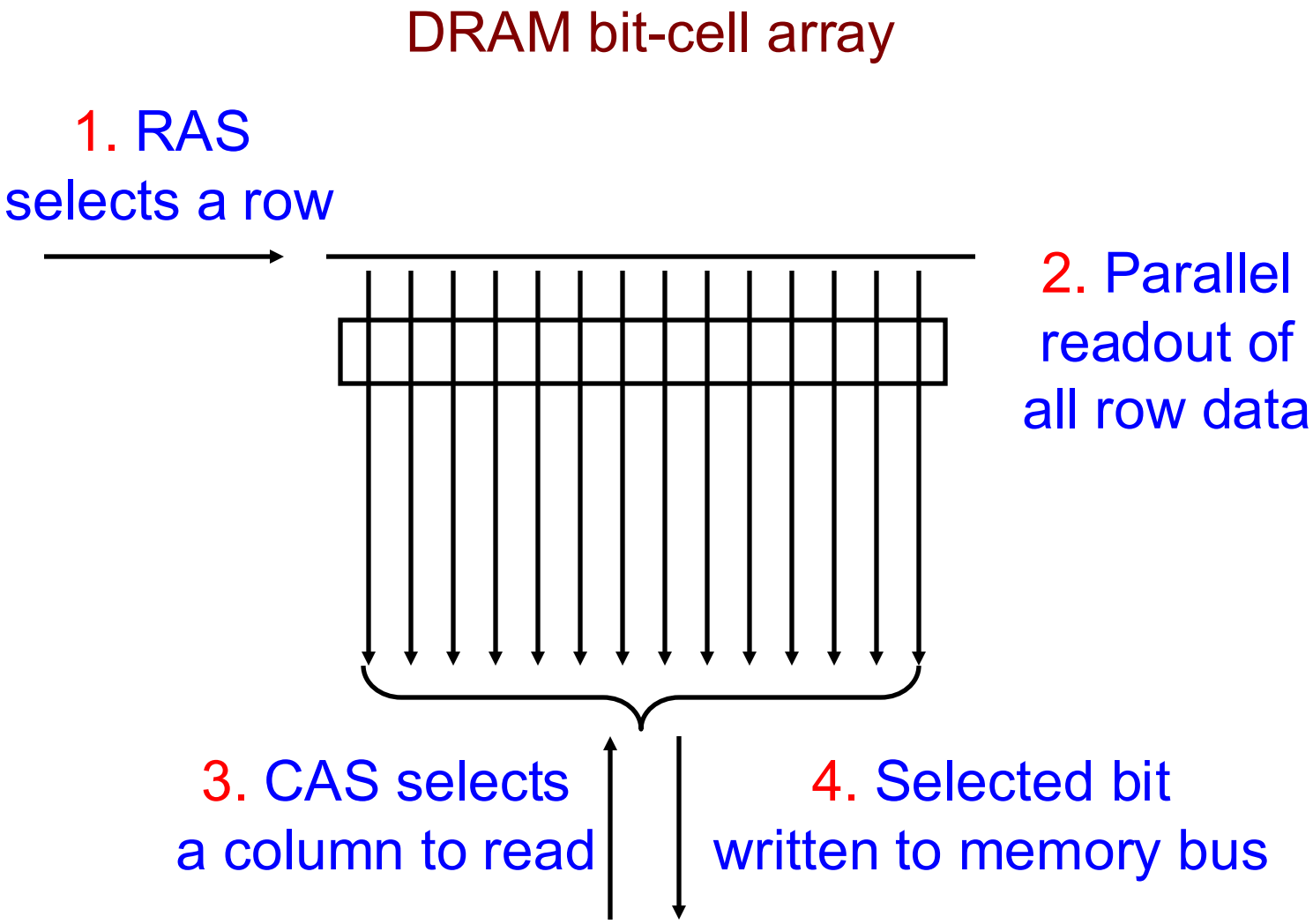
Interposer stacking (2.5D)

Two forms of die stacking. The 2.5D form is available now. 3D stacking is under development and faces heat management challenges due to the CPU.

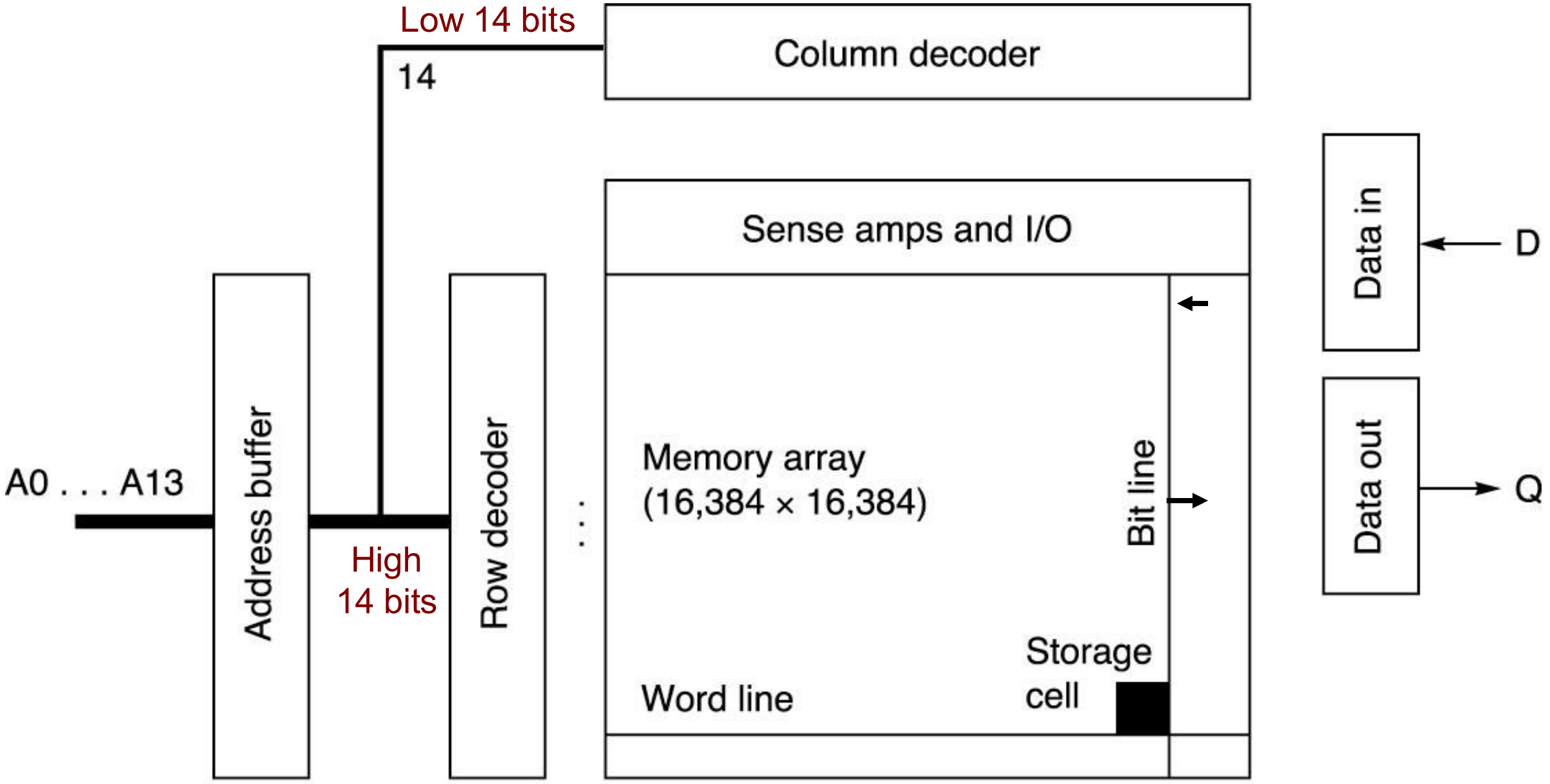
Main Memory

- Bandwidth: Bytes read or written per unit time
- Latency: Described by
  - *Access Time*: Delay between initiation/completion
    - For reads: Present address till result ready.
  - *Cycle time*: Minimum interval between separate requests to memory.
- Separate bus CPU ➔ Memory to carry addresses.
- RAS (Row Access Strobe)
  - First half of address, sent first.
- CAS (Column Access Strobe)
  - Second half of address, sent second.

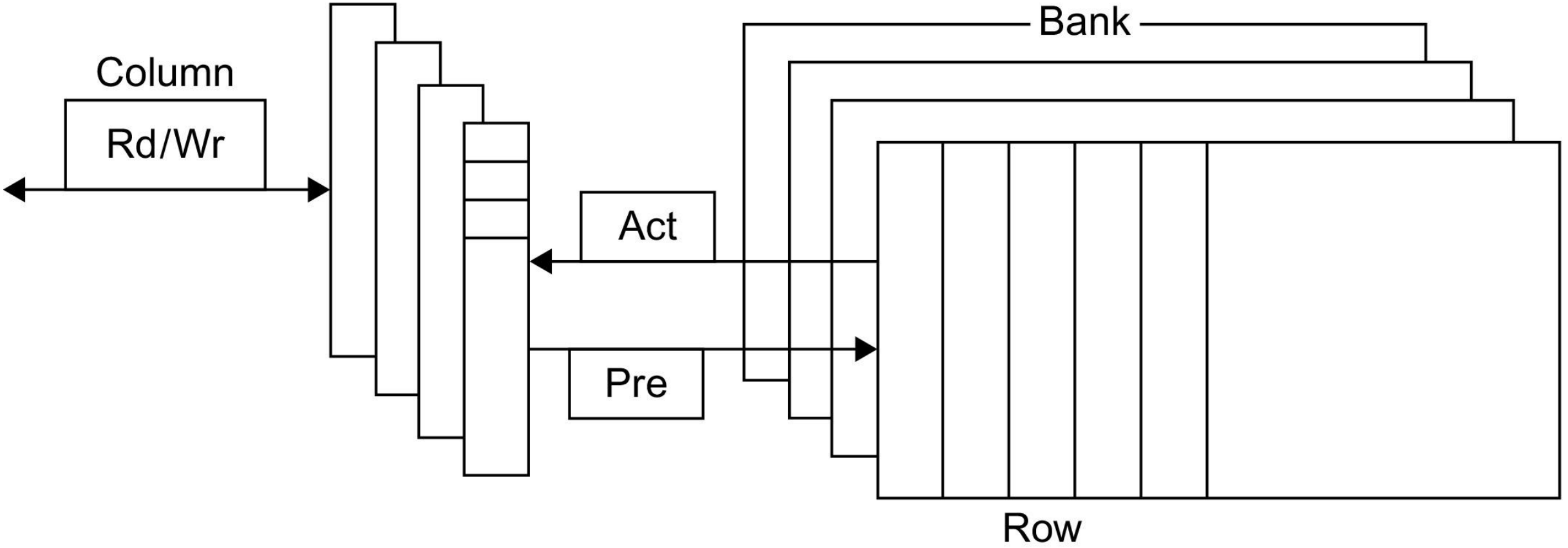
RAS vs. CAS


DRAM bit-cell array
1. RAS
selects a row
2. Parallel
readout of
all row data
3. CAS selects
a column to read
4. Selected bit
written to memory bus

Typical DRAM Organization



Internal Organization of a DRAM



Modern DRAMs are organized in banks, up to 16 for DDR4. Each bank consists of a series of rows. Sending an ACT (Activate) command opens a bank and a row and loads the row into a row buffer. When the row is in the buffer, it can be transferred by successive column addresses at whatever the width of the DRAM is (typically 4, 8, or 16 bits in DDR4) or by specifying a block transfer and the starting address. The Precharge command (PRE) closes the bank and row and readies it for a new access. Each command, as well as block transfers, are synchronized with a clock.

Types of Memory

- DRAM (Dynamic Random Access Memory)
  - Cell design needs only 1 transistor per bit stored.
  - Cell charges leak away and may *dynamically* (over time) drift from their initial levels.
  - Requires periodic refreshing to correct drift
    - e.g., every 8 ms
  - Time spent refreshing kept to <5% of bandwidth
- SRAM (Static Random Access Memory)
  - Cell voltages are *statically* (unchangingly) tied to power supply references. No drift, no refresh.
  - But needs 4-6 transistors per bit.
- DRAM: 4-8x larger, 8-16x slower, 8-16x cheaper/bit

DRAM Variations

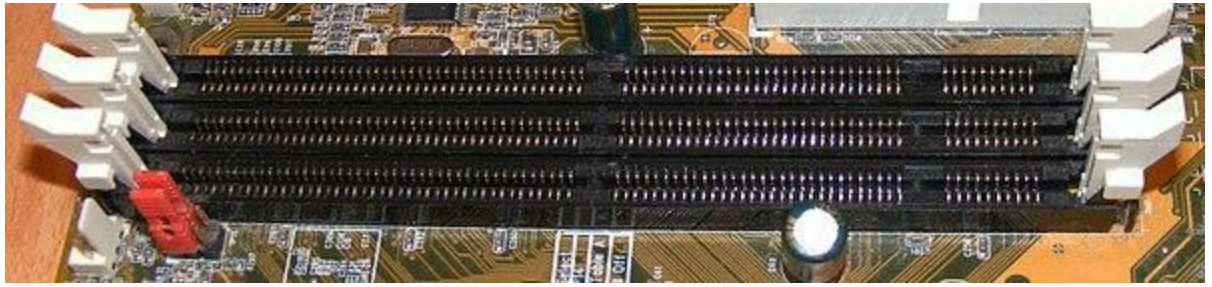
- SDRAM – Synchronous DRAM
  - DRAM internal operation synchronized by a clock signal provided on the memory bus
  - Double Data Rate (DDR) – uses both clock edges
- RDRAM – RAMBUS (Inc.) DRAM
  - Proprietary DRAM interface technology
    - on-chip interleaving / multi-bank technology
    - a high-speed *packet-switched* (*split-transaction*) bus interface
    - byte-wide interface, synchronous, dual-rate
  - Licensed to many chip & CPU makers
  - Higher bandwidth, costly than generic SDRAM
- DRDRAM – "Direct" RDRAM (2nd ed. spec.)
  - Separate row and column address/command buses
  - Higher bandwidth (18-bit data, more banks, faster clock)

Access Time for DDR SDRAMs

			Best case access time (no precharge)			Precharge needed
Production year	Chip size	DRAM type	RAS time (ns)	CAS time (ns)	Total (ns)	Total (ns)
2000	256M bit	DDR1	21	21	42	63
2002	512M bit	DDR1	15	15	30	45
2004	1G bit	DDR2	15	15	30	45
2006	2G bit	DDR2	10	10	20	30
2010	4G bit	DDR3	13	13	26	39
2016	8G bit	DDR4	13	13	26	39

Access time is for a random memory word and assumes a new row must be opened. If the row is in a different bank, we assume the bank is precharged. If the row is not open, then a precharge is required, and the access time is longer. As the number of banks has increased, the ability to hide the precharge time has also increased.

Dual In-Line Memory Module (DIMM)



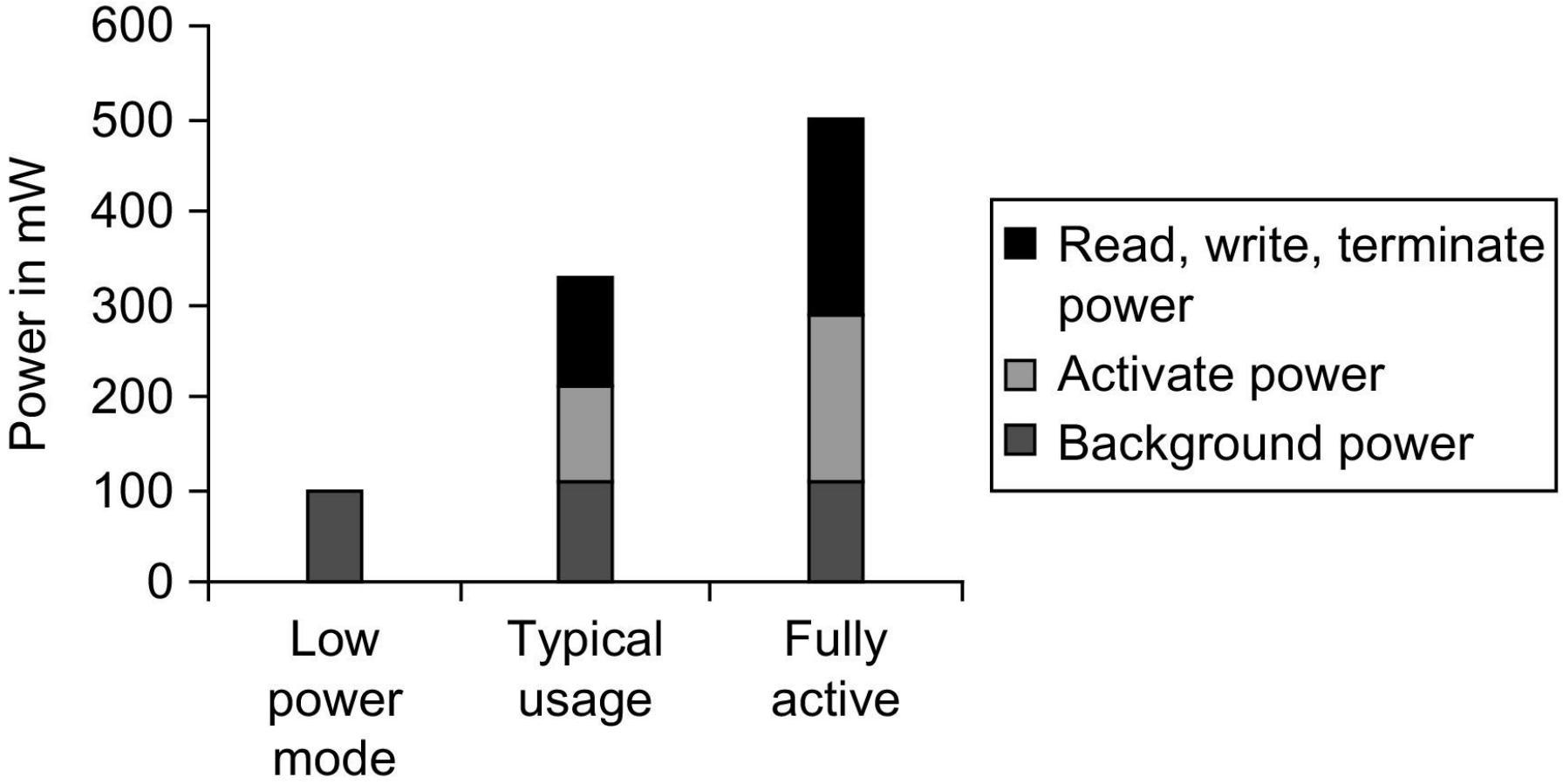
Clock Rates of DDR DRAMS in 2016

Standard	I/O clock rate	M transfers/s	DRAM name	MiB/s/DIMM	DIMM name
DDR1	133	266	DDR266	2128	PC2100
DDR1	150	300	DDR300	2400	PC2400
DDR1	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1333	2666	DDR4-2666	21,300	PC21300

Note the numerical relationship between the columns. The third column is twice the second, and the fourth uses the number from the third column in the name of the DRAM chip. The fifth column is eight times the third column, and a rounded version of this number is used in the name of the DIMM.

DDR4 is energy efficient than DDR3 – DDR4 operates at 1.2V while DDR3 operates at 1.35V or 1.5V.

Power Consumption for a DDR3 SDRAM



Power consumption for a DDR3 SDRAM operating under three conditions: low-power (shutdown) mode, typical system mode (DRAM is active 30% of the time for reads and 15% for writes), and fully active mode, where the DRAM is continuously reading or writing. Reads and writes assume bursts of eight transfers. These data are based on a Micron 1.5V 2GB DDR3-1066, although similar savings occur in DDR4 SDRAMs.

ROM and Flash

- ROM (Read-Only Memory)
  - Nonvolatile + protection
- Flash
  - Nonvolatile RAMs
  - NVRAMs require no power to maintain state
  - Reading flash is near DRAM speeds
  - Writing is 10-100x slower than DRAM
  - Frequently used for upgradeable embedded SW
- Used in Embedded Processors

Flash Memory

- Flash memory is a non-volatile computer memory that can be electrically erased (like EEPROM) and reprogrammed.
  - It does not need power to maintain the information stored in the chip.
  - It offers fast read access times and better kinetic shock resistance than hard disks. It is enormously durable, being able to withstand intense pressure, extremes of temperature and immersion in water.
- It is a specific type of EEPROM that is erased and programmed in large blocks; in early flash the entire chip had to be erased at once.
  - Examples of applications include PDAs, digital audio players, digital cameras and mobile phones.
- Flash memory stores information in an array of floating-gate transistors, called "cells". In multi-level cell (MLC) devices, it can store more than one bit per cell by choosing between multiple levels of electrical charge to apply to the floating gates of its cells.
  - NOR and NAND flash memories.

Memory Hierarchy Design

- Introduction
- Eleven Advanced Cache Optimizations
- Memory Technology and Optimizations
- Virtual Memory and Virtual Machines
- ARM Cortex-A53 Intel i7 Memory Hierarchy
- Conclusion

Protection via Virtual Memory

- Goal is to make computing systems secure
  - A process should not be able to observe/modify the content (e.g., credit card details) of another process
    - A process is a running program plus any state needed to continue running it (in case of context/process switch).
- Architecture and OS should join hands
  - Architecture must limit what a user process can access but allow an OS process greater access.
  - Architecture must support
    - At least two modes: user, kernel/supervisor
    - Allow user process some read-only area e.g., protection bits
    - Transfer from user mode to supervisor mode (system call)
    - Limit memory access to protect the memory state of a process without having to swap the process to disk on a context switch.

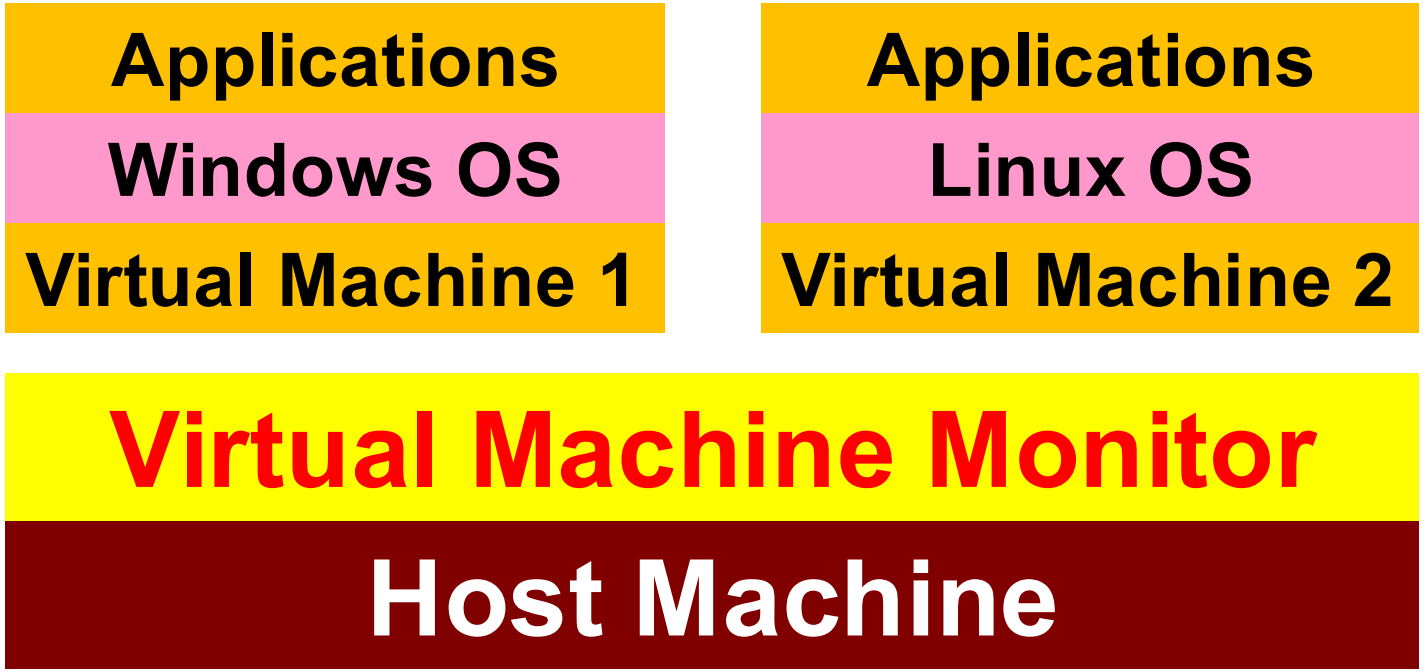
Protection via Virtual Memory

- Is it enough protection?
  - NO
- Large OS with too many possible bugs
  - Flaws in OS have led to vulnerabilities that are routinely exploited.
- Much smaller code base than OS
  - Virtual Machines (VM) for improving protection
  - VMs can also be used for
    - Managing software
    - Managing hardware

Introduction to Virtual Machines

- VMs developed in late 1960s
  - Remained important in mainframe computing over the years
  - Largely ignored in single user computers of 1980s and 1990s
- Recently regained popularity due to
  - increasing importance of isolation and security in modern systems,
  - failures in security and reliability of standard operating systems,
  - sharing of a single computer among many unrelated users,
  - and the dramatic increases in raw speed of processors, which makes the overhead of VMs more acceptable

Overview


Applications
Windows OS
Virtual Machine 1
Applications
Linux OS
Virtual Machine 2
Virtual Machine Monitor
Host Machine

What is a Virtual Machine (VM)?

- All emulation methods that provide a standard software interface, such as the Java VM
- "(Operating) System Virtual Machines" provide a complete system level environment at binary ISA
  - Here assume ISAs always match the native hardware ISA
  - e.g., IBM VM/370, VMware ESX Server, and Xen
- Present illusion that VM users have entire computer to themselves, including a copy of OS
- Single computer runs multiple VMs, and can support multiple, different OSes
  - On normal platform, single OS "owns" all HW resources
  - With a VM, multiple OSes all share HW resources
- Underlying HW platform is called the host, and its resources are shared among the guest VMs

Virtual Machine Monitors (VMMs)

- Virtual machine monitor (VMM) or hypervisor is a software that supports VMs
- VMM determines how to map virtual resources to physical resources
- Physical resource may be time-shared, partitioned, or emulated in software
- VMM is much smaller than a traditional OS;
  - isolation portion of a VMM is ≈ 10000 lines of code

VMM Overhead?

- Depends on the workload
- User-level processor-bound programs (e.g., SPEC) have zero-virtualization overhead
  - Runs at native speeds since OS rarely invoked
- I/O-intensive workloads $\Rightarrow$ OS-intensive
  - execute many system calls and privileged instructions
  - can result in high virtualization overhead
  - For System VMs, goal of architecture and VMM is to run almost all instructions directly on native hardware
- If I/O-intensive workload is also I/O-bound
  - low processor utilization since waiting for I/O
  - processor virtualization can be hidden
  - low virtualization overhead

Other Uses of VMs

- Focus so far was on protection
- Two other commercially important uses of VMs

1. Managing Software
   - VMs provide an abstraction that can run the complete SW stack, even including old OSes like DOS
   - Some VMs running legacy OSes, many running current stable OS release, few testing next OS release
2. Managing Hardware
   - VMs allow separate SW stacks to run independently yet share HW, thereby consolidating number of servers
     - Some run each application with compatible version of OS on separate computers, as separation helps dependability
   - Migrate running VM to a different computer
     - Either to balance load or to evacuate from failing HW

Example: Xen VM

- Xen: Open-source System VMM for 80x86 ISA
  - Project started at University of Cambridge, GNU license model
- Original version of VM is running unmodified OS
  - Significant wasted effort just to keep guest OS happy
- "paravirtualization" - small modifications to guest OS to simplify virtualization

Three Examples of paravirtualization in Xen:

1. To avoid flushing TLB when invoke VMM, Xen mapped itself into upper 64 MB of address space of each VM
2. Guest OS allowed to allocate pages, check that didn't violate protection restrictions
3. To protect the guest OS from user programs in VM, Xen takes advantage of 4 protection levels available in 80x86
   - Most OSes for 80x86 keep everything at privilege levels 0 or at 3.
   - Xen VMM runs at the highest privilege level (0)
   - Guest OS runs at the next level (1)
   - Applications run at the lowest privilege level (3)

Xen Performance

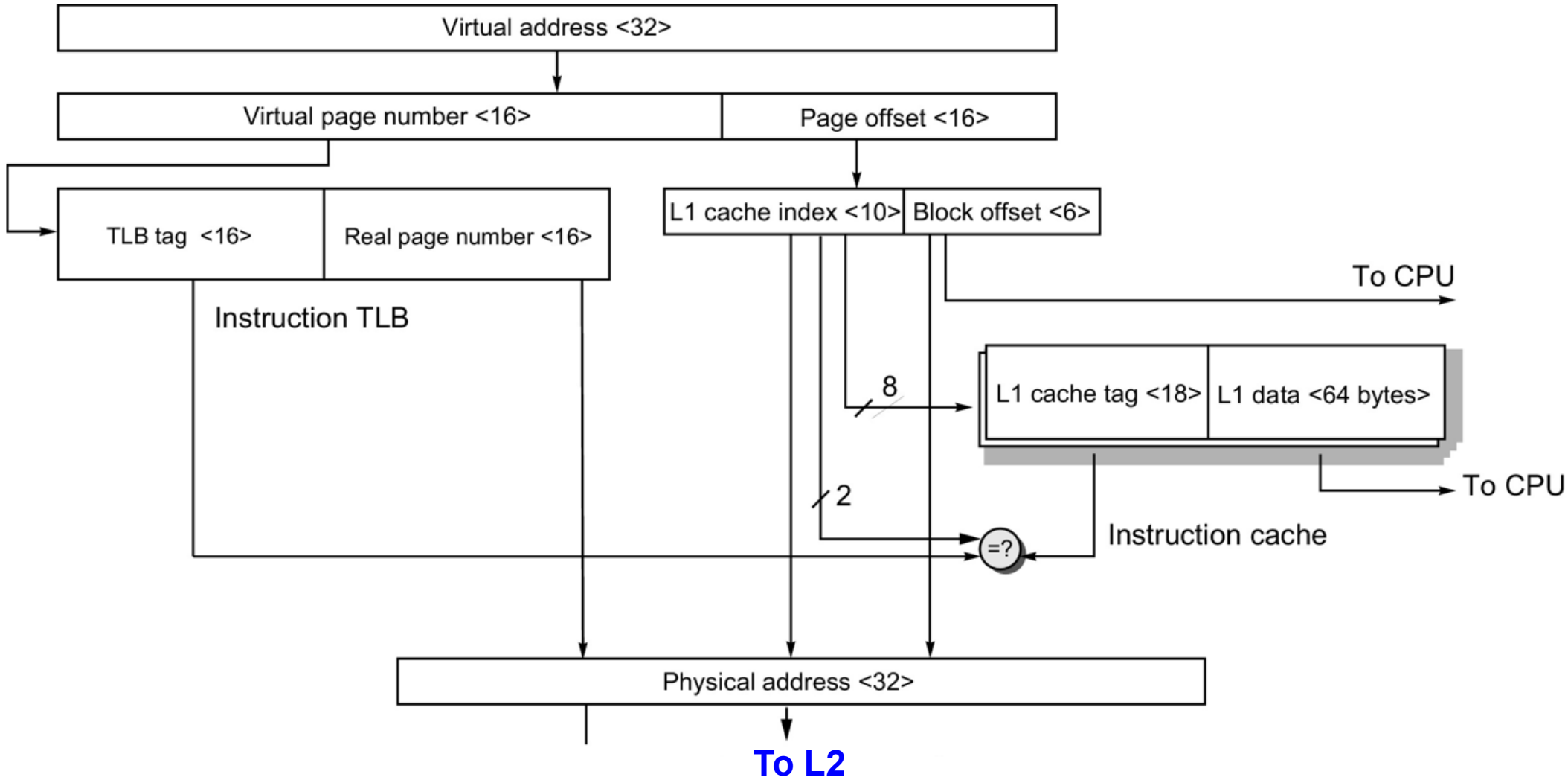
1. **> 2X instructions**: page remapping and page transfer between driver and guest VMs and due to communication between the VMs
2. **4X L2 cache misses**: Linux uses zero-copy network interface that depends on ability of NIC to do DMA from different locations in memory
   - Since Xen does not support "gather DMA" in its virtual network interface, it can't do true zero-copy in the guest VM
3. **12X – 24X Data TLB misses**: 2 Linux optimizations
   - Superpages for part of Linux kernel space, and 4MB pages lowers TLB misses versus using 1024 4 KB pages. Not in Xen.
   - PTEs marked global are not flushed on a context switch, and Linux uses them for its kernel space. Not in Xen.

Future Xen may address the problems in 2 and 3, but 1 is inherent.

Memory Hierarchy Design

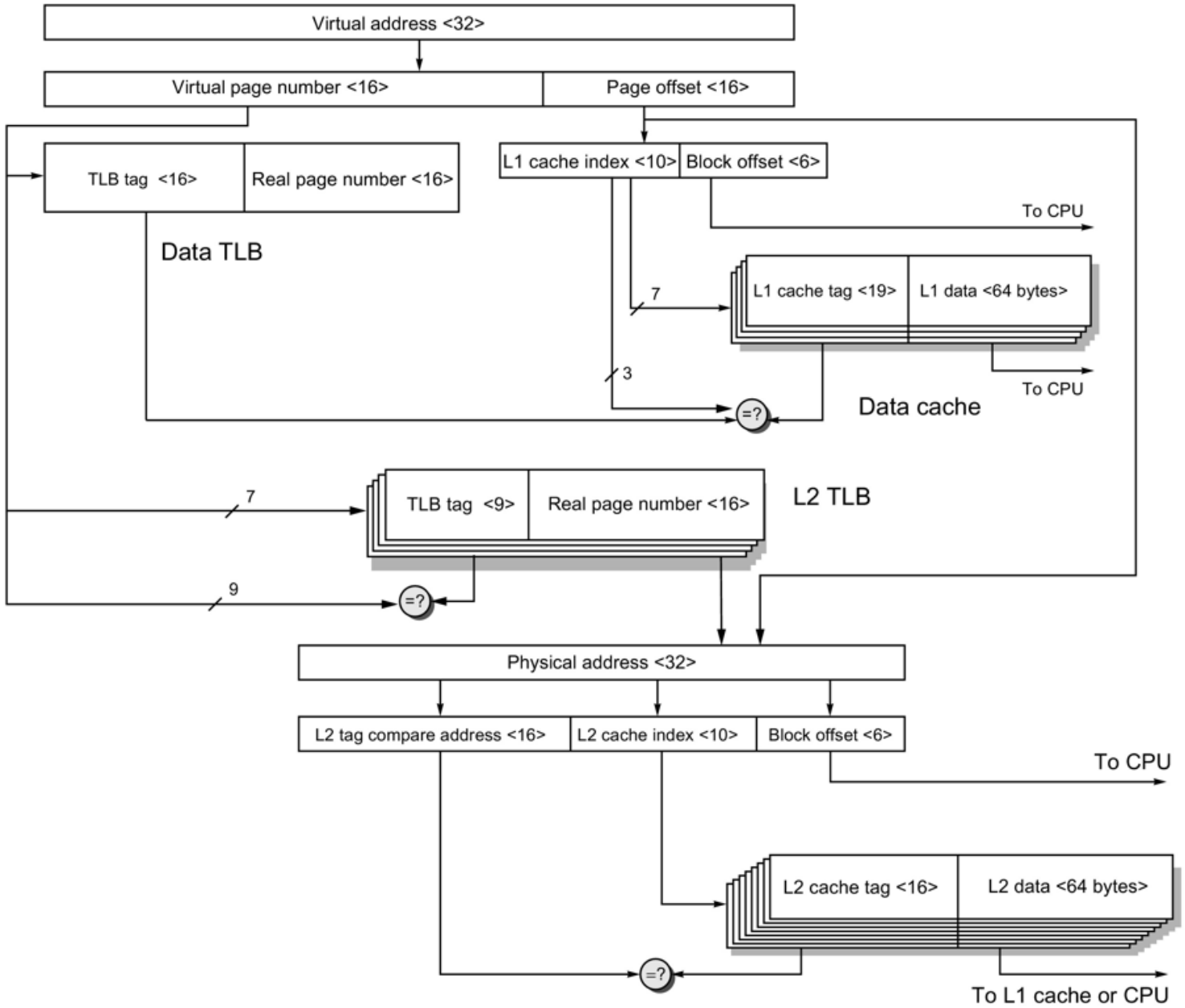
- Introduction
- Eleven Advanced Cache Optimizations
- Memory Technology and Optimizations
- Virtual Memory and Virtual Machines
- ARM Cortex-A53 Intel i7 Memory Hierarchy
- Conclusion

ARM Cortex-A53 Instruction Access Path



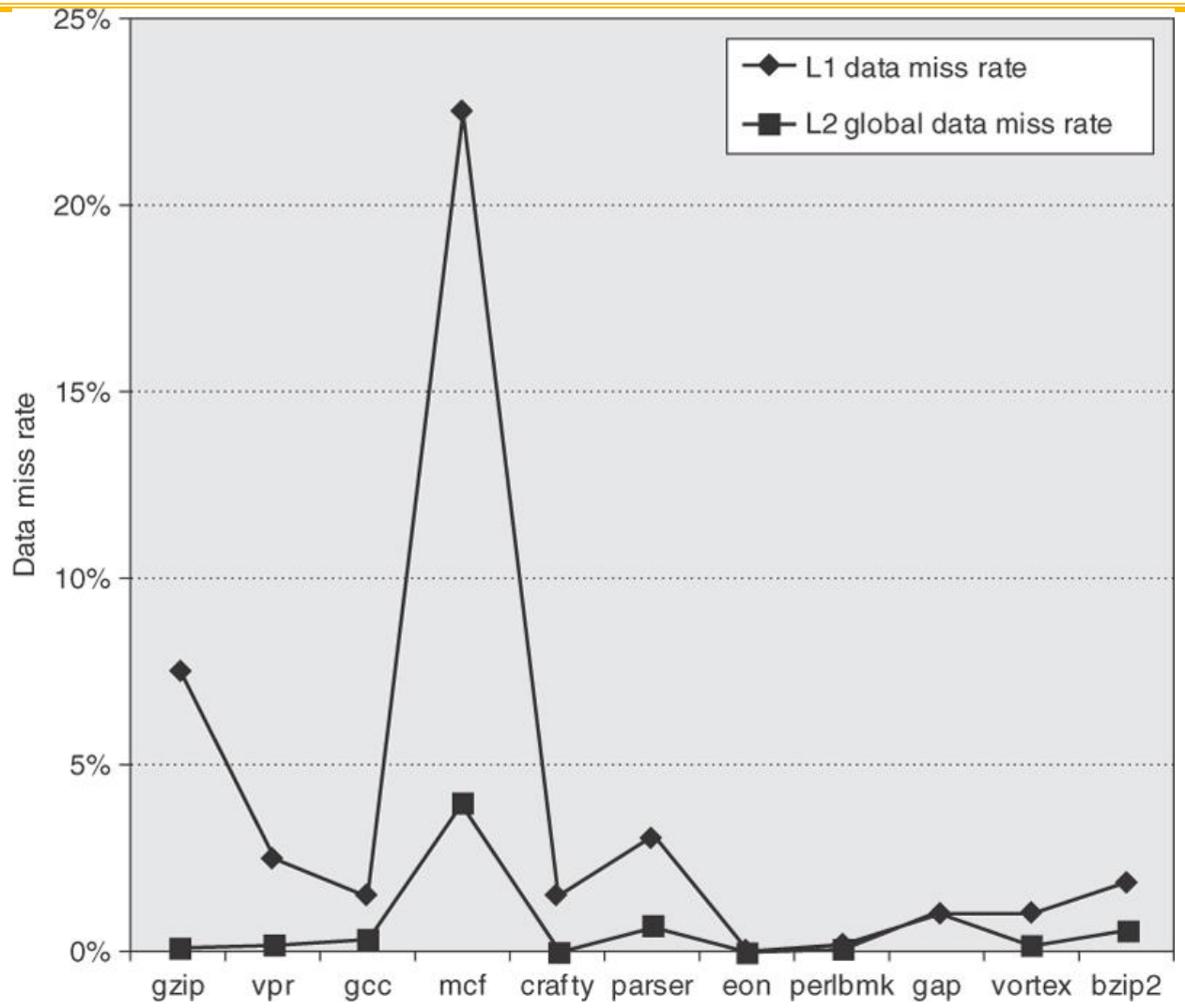
The TLB (instruction or data) is fully associative each with 10 entries, using a 64 KB page. The L1 I-cache is 32KB, 2-way set associative, with 64-byte blocks. The L1 D-cache is 32 KB, 4-way set associative, and 64-byte blocks. The L2 TLB is 512 entries and 4-way set associative. The L2 cache is 1MB, 16-way set associative with 64-byte blocks.

ARM Cortex-A53 Data Access Path



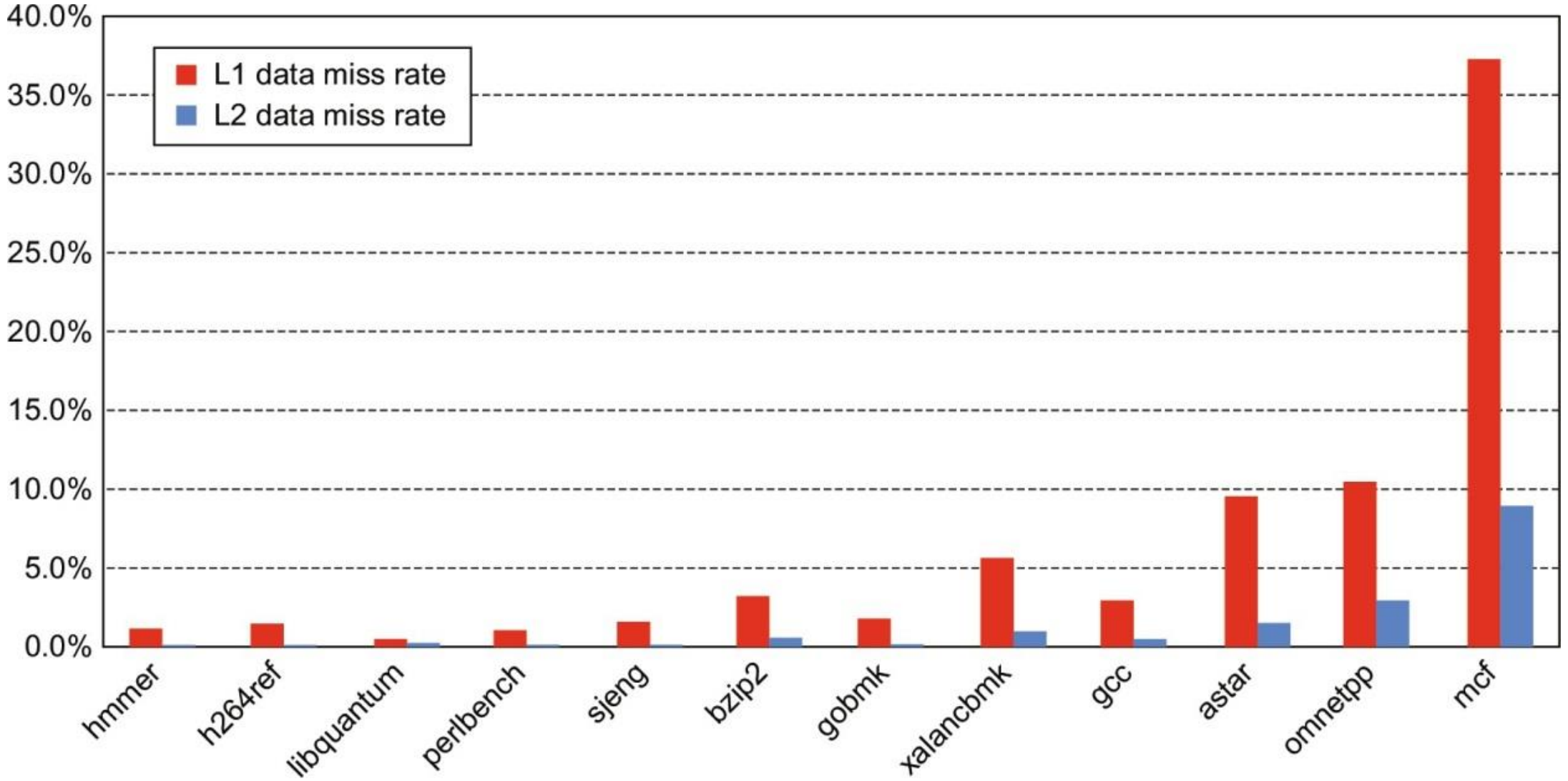
- The TLB (instruction or data) is fully associative each with 10 entries, using a 64 KB page.
- The L1 I-cache is 32KB, 2-way set associative, with 64-byte blocks.
- The L1 D-cache is 32 KB, 4-way set associative, and 64-byte blocks.
- The L2 TLB is 512 entries and 4-way set associative.
- The L2 cache is 1MB, 16-way set associative with 64-byte blocks.

Data Miss Rate in ARM Cortex-A8



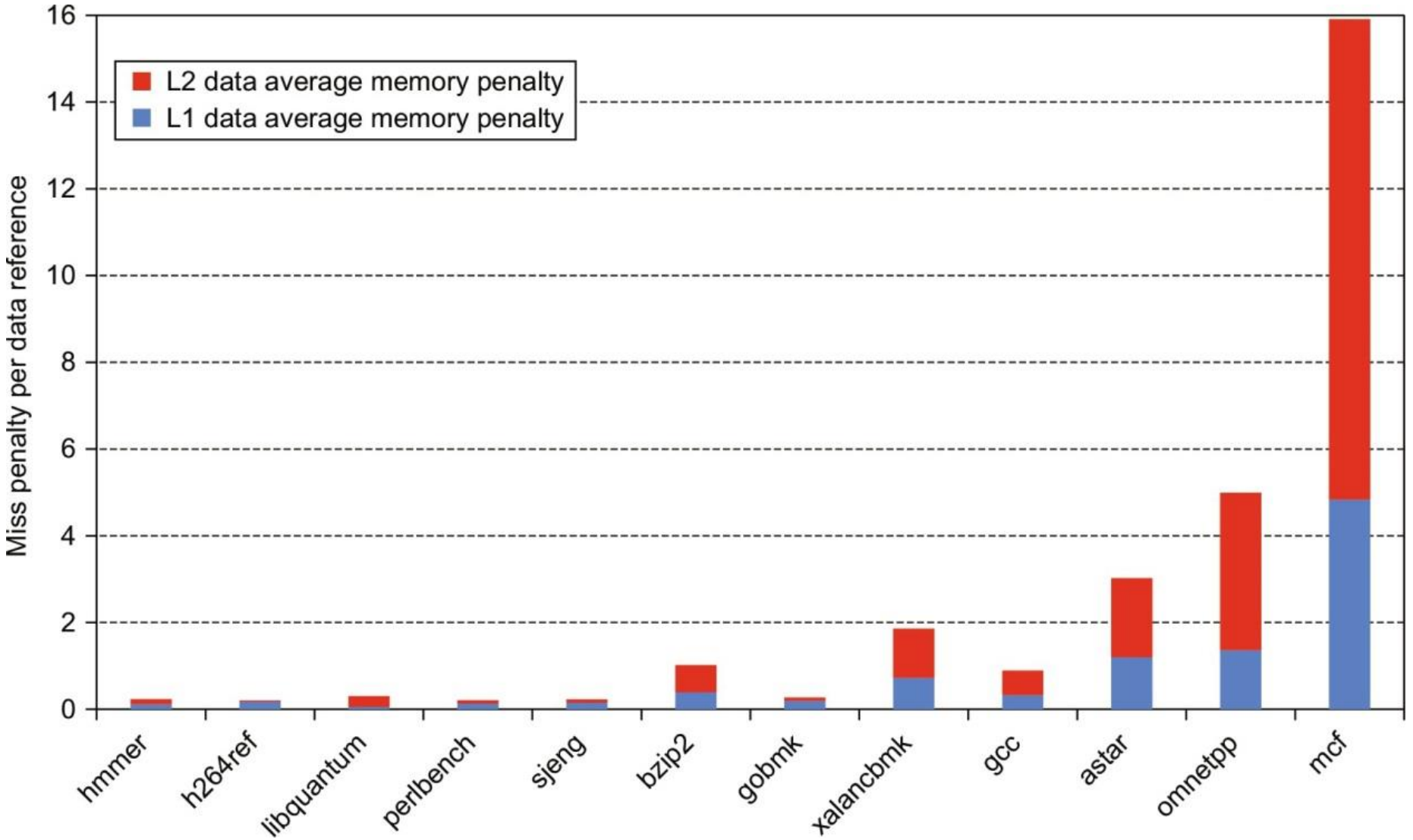
Applications with larger memory footprints tend to have higher miss rates in both L1 and L2. Note that the L2 rate is the global miss rate, that is counting all references, including those that hit in L1. **mcf** is known as a cache buster.

Data Cache Miss Rate



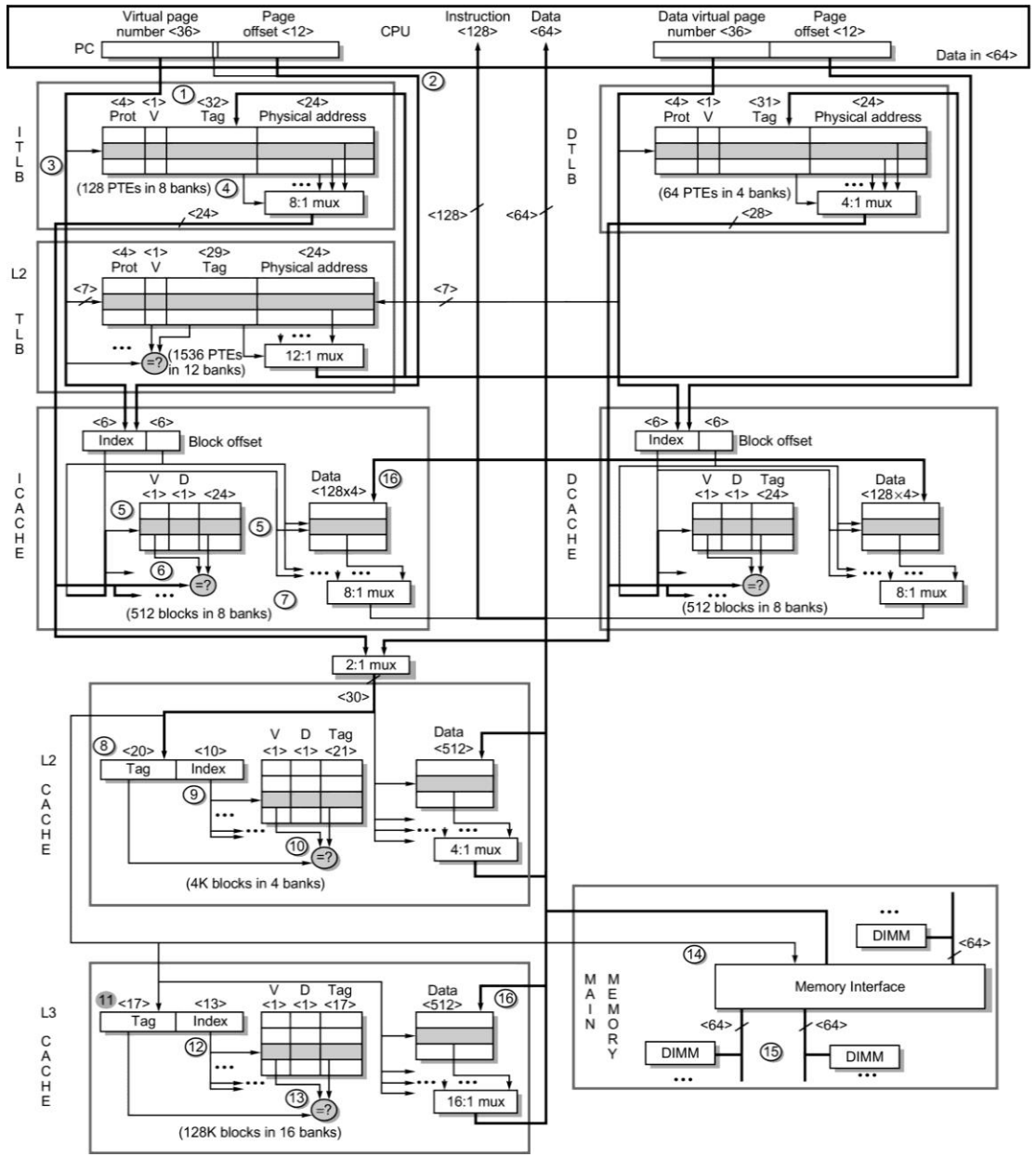
Cortex-A53 running SPECInt2006: Applications with larger memory footprints tend to have higher miss rates in both L1 and L2. L2 rate is the global miss rate. MCF is known as a cache buster.

Average Memory Access Penalty



Cortex-A53 running SPECInt2006: Although the miss rates for L1 are significantly higher, the L2 miss penalty, which is more than five times higher, means that the L2 misses can contribute significantly.

Intel Core i7 6700 Memory Hierarchy



Conclusion

- Eleven Advanced Cache Optimizations
  - Improvement of cache performance using way prediction, prefetching, nonblocking caches, etc.
- Memory Technology and Optimizations
  - SRAM, DRAM with innovations (fast page mode, synchronous, double data rate, etc.), ROM, Flash
- Protection using Virtual Machine
  - Overcome security flaws of large OSes
  - Manage software, manage hardware, etc.
- ARM Cortex-A53 and Intel i7 Memory Hierarchy
- Please read Section 2.1 – 2.4
- We review multiprocessors next.