CSC 631: High-Performance Computer Architecture

Spring 2017
Lecture 9: Memory
Part I

Introduction

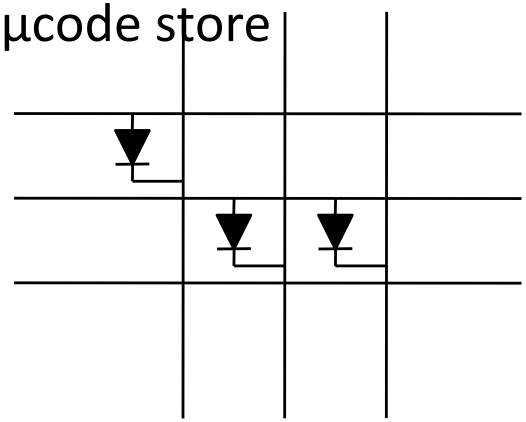
- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory

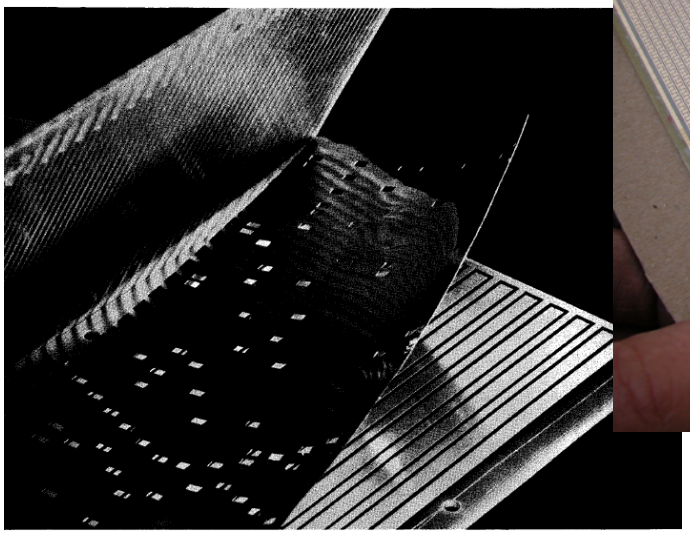
Early Read-Only Memory Technologies

Punched cards, From early 1700s through Jaquard Loom, Babbage, and then IBM

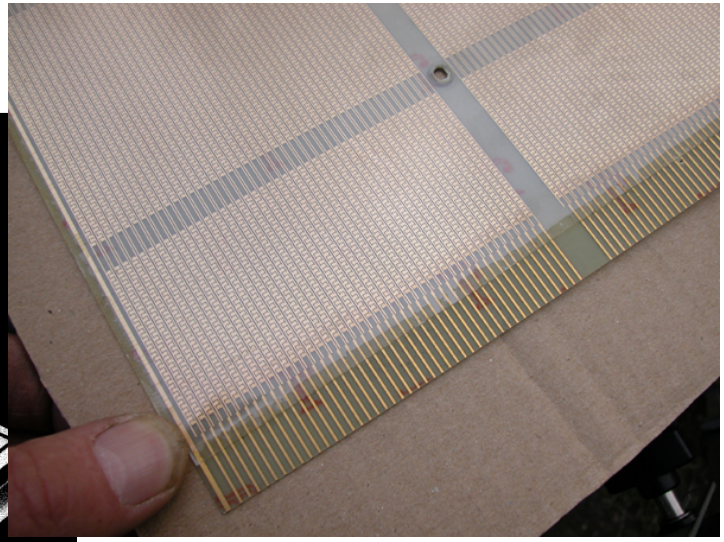
Punched paper tape, instruction stream in Harvard Mk 1

Diode Matrix, EDSAC-2 µcode store





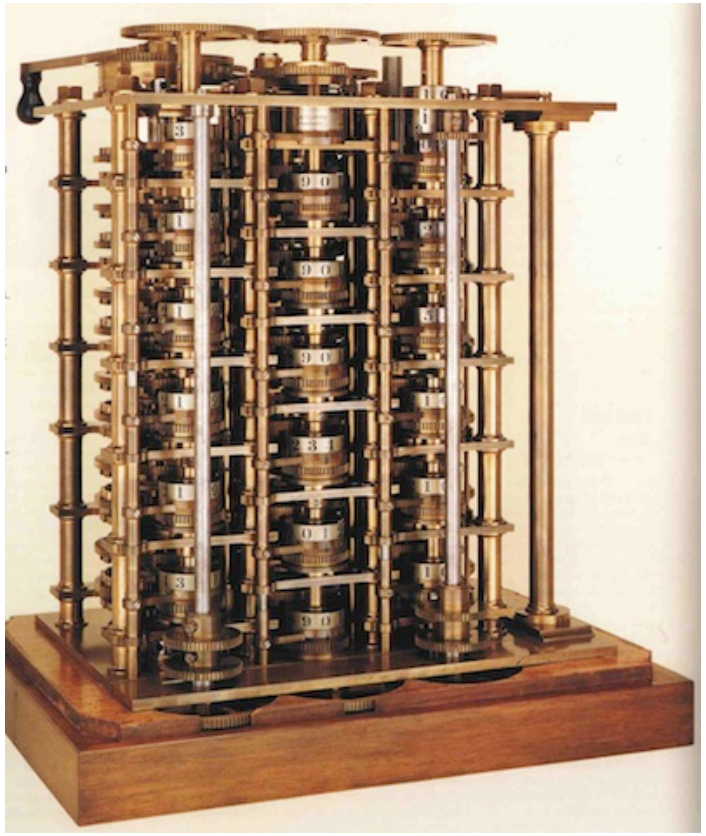
IBM Card Capacitor ROS

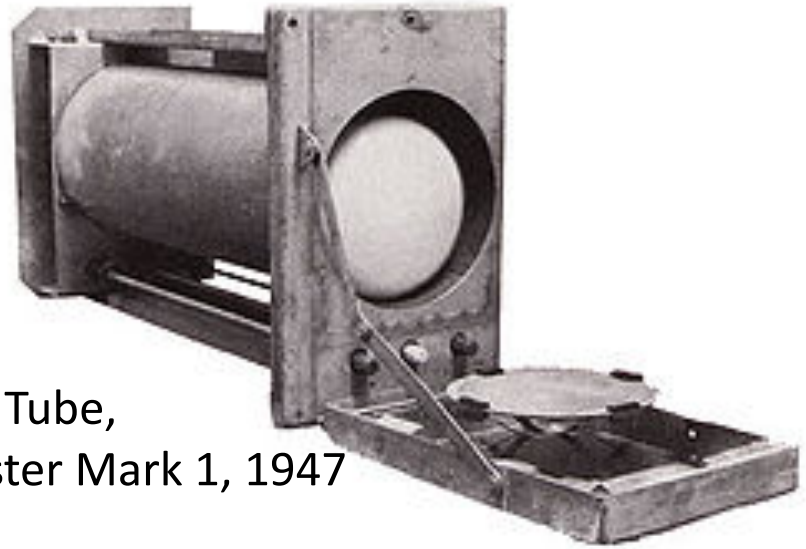


IBM Balanced Capacitor ROS

Early Read/Write Main Memory Technologies

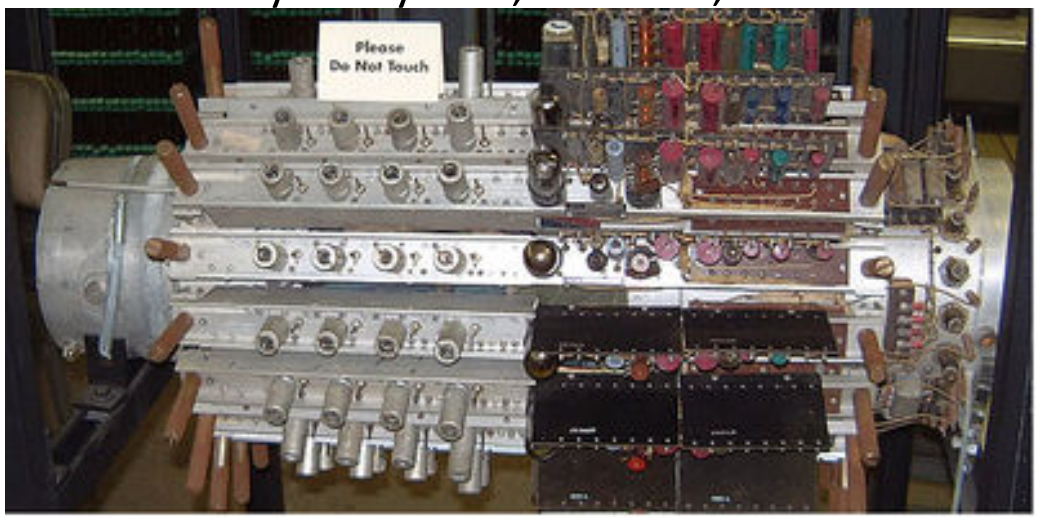
Babbage, 1800s: Digits stored on mechanical wheels





Williams Tube, Manchester Mark 1, 1947

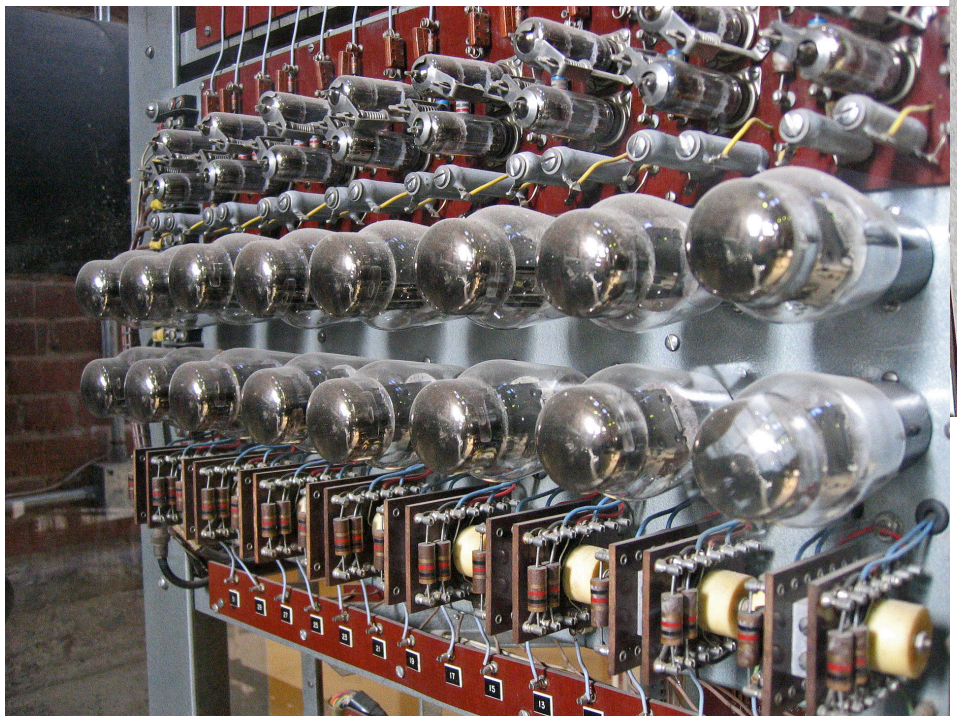
Mercury Delay Line, Univac 1, 1951

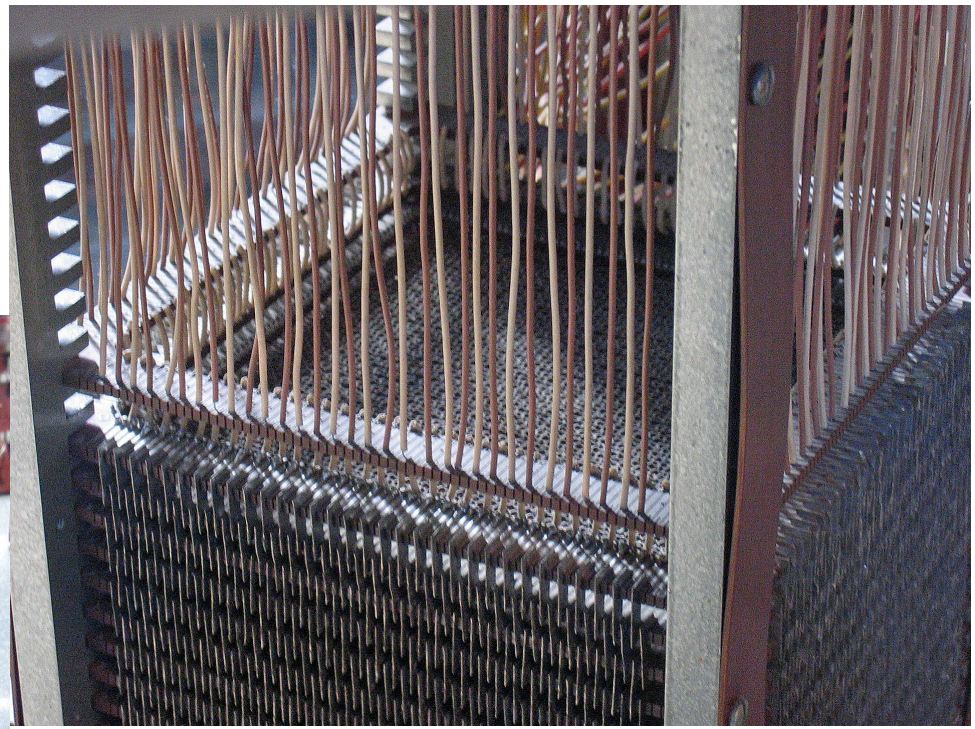


Also, regenerative capacitor memory on Atanasoff-Berry computer, and rotating magnetic drum memory on IBM 650

MIT Whirlwind Core Memory

2048 (2K) words of 16 bits each of random-access storage



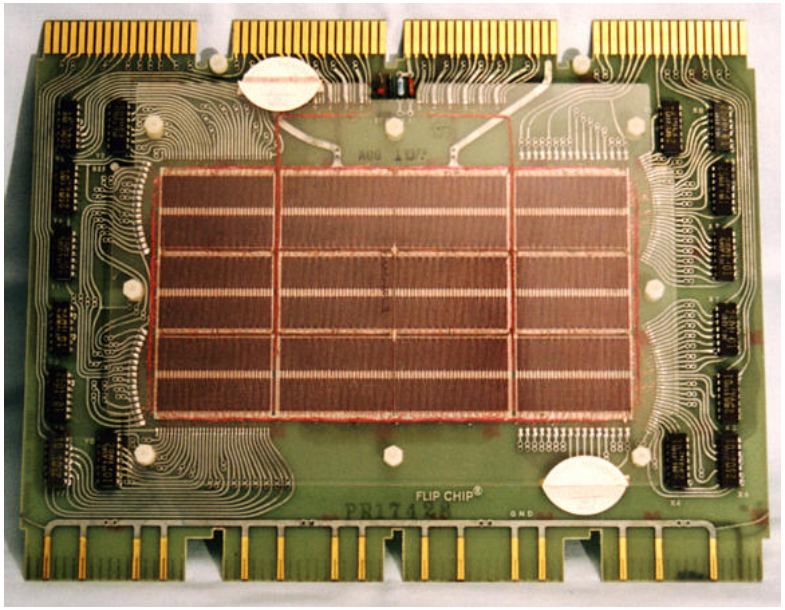


Memory technology used mercury delay lines and electrostatic storage

Core Memory

- Core memory was first large scale reliable main memory
  - invented by Forrester in late 40s/early 50s at MIT for Whirlwind project
- Bits stored as magnetization polarity on small ferrite cores threaded onto two-dimensional grid of wires
- Coincident current pulses on X and Y wires would write cell and also sense original state (destructive reads)

- Robust, non-volatile storage
- Used on space shuttle computers
- Cores threaded onto wires by hand (25 billion a year at peak production)
- Core access time ~ 1µs

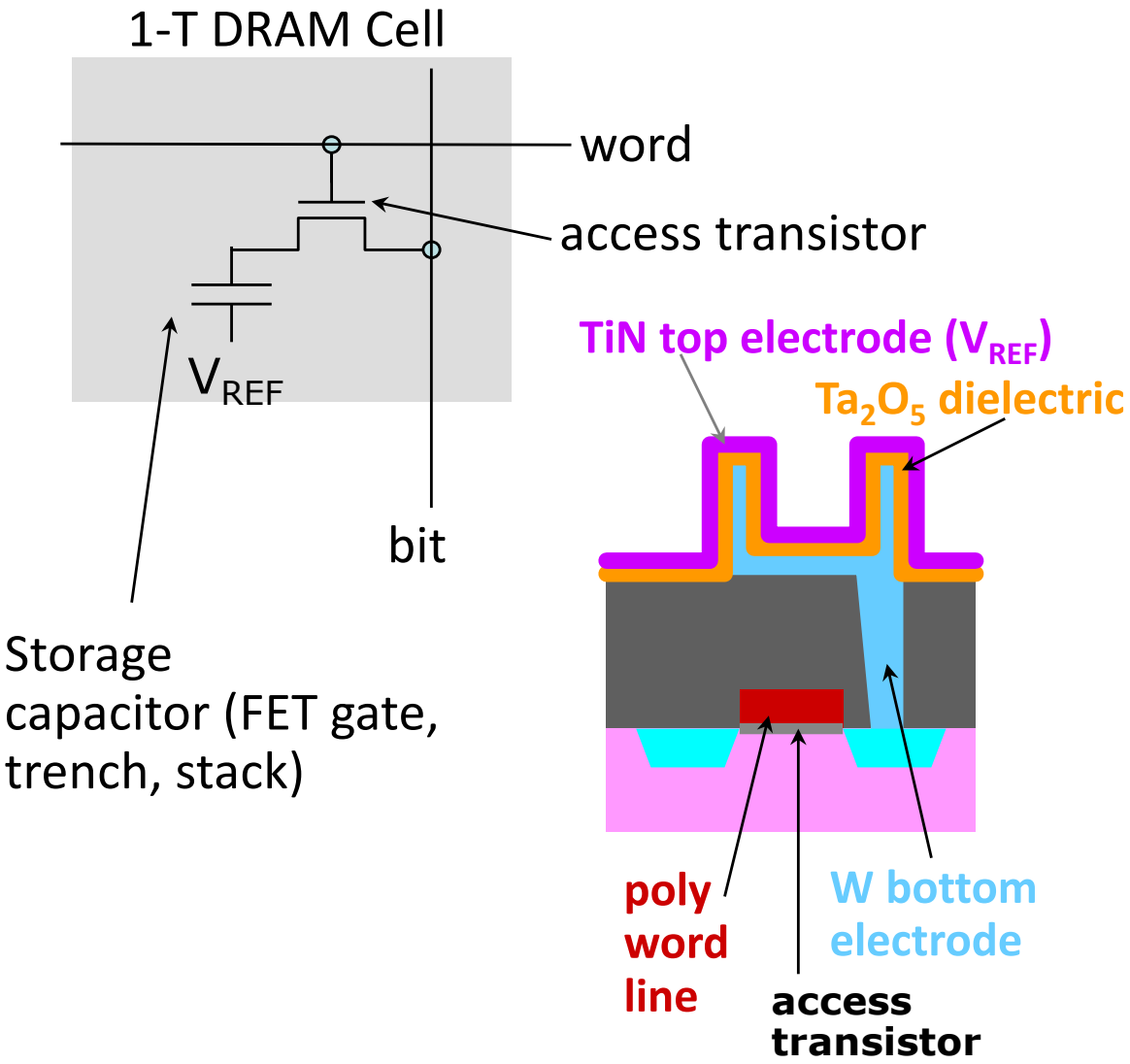


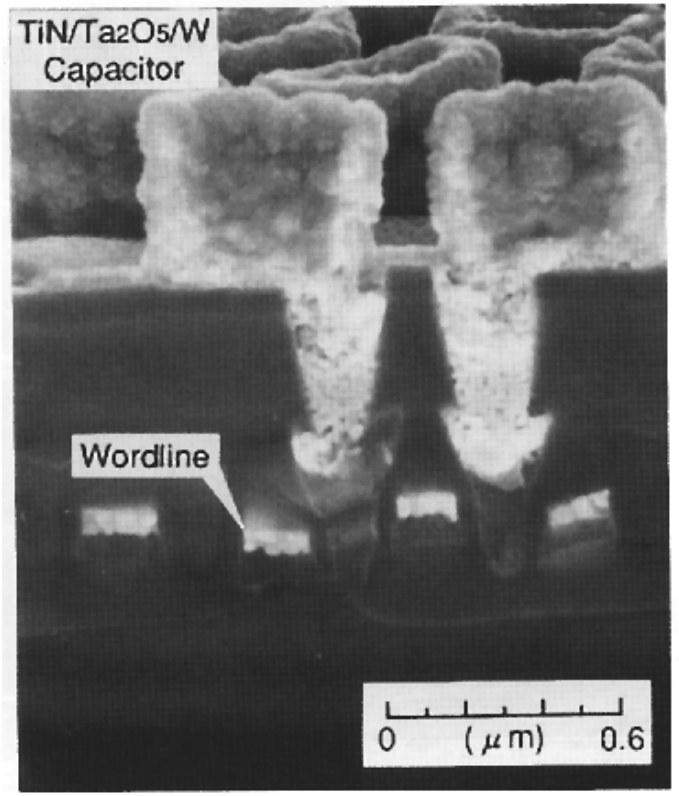
DEC PDP-8/E Board,
4K words x 12 bits, (1968)

Semiconductor Memory

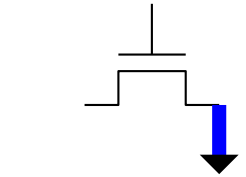
- Semiconductor memory began to be competitive in early 1970s
  - Intel formed to exploit market for semiconductor memory
  - Early semiconductor memory was Static RAM (SRAM).
- First commercial Dynamic RAM (DRAM) was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value
  - Value has to be regularly read and written back, hence dynamic
- Semiconductor memory quickly replaced core in ‘70s

One-Transistor Dynamic RAM [Dennard, IBM]

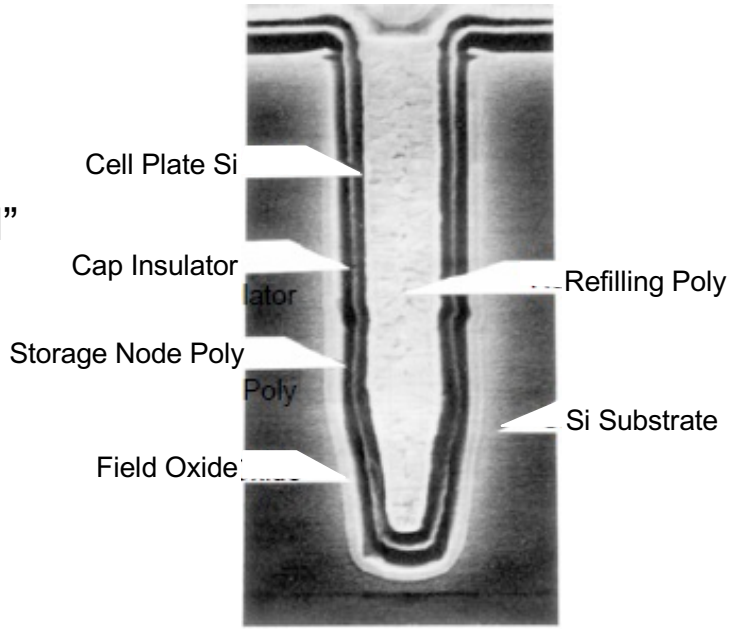


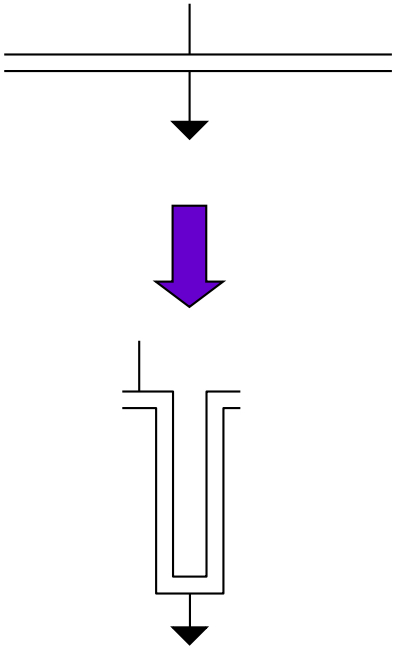


Implementing the Capacitor

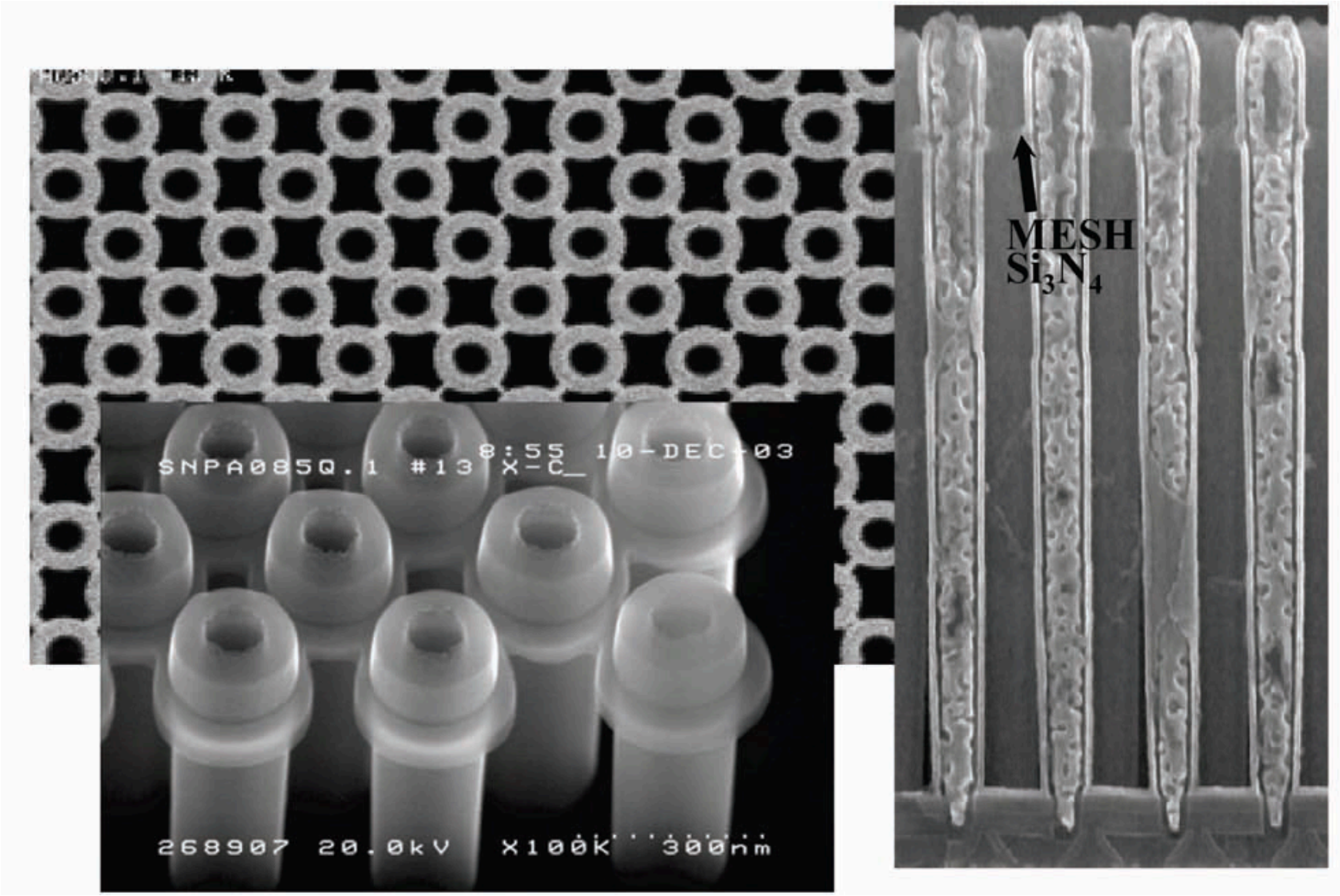


"Trench Cell"





Modern DRAM Cell Structure

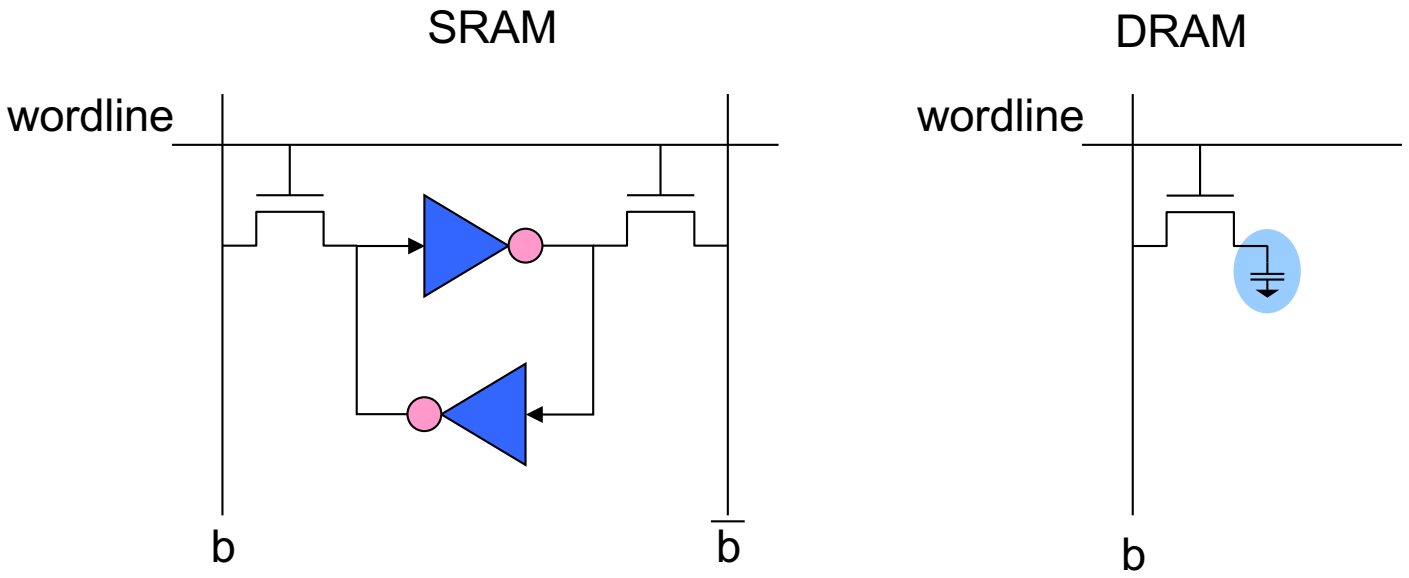


[Samsung, sub-70nm DRAM, 2004]

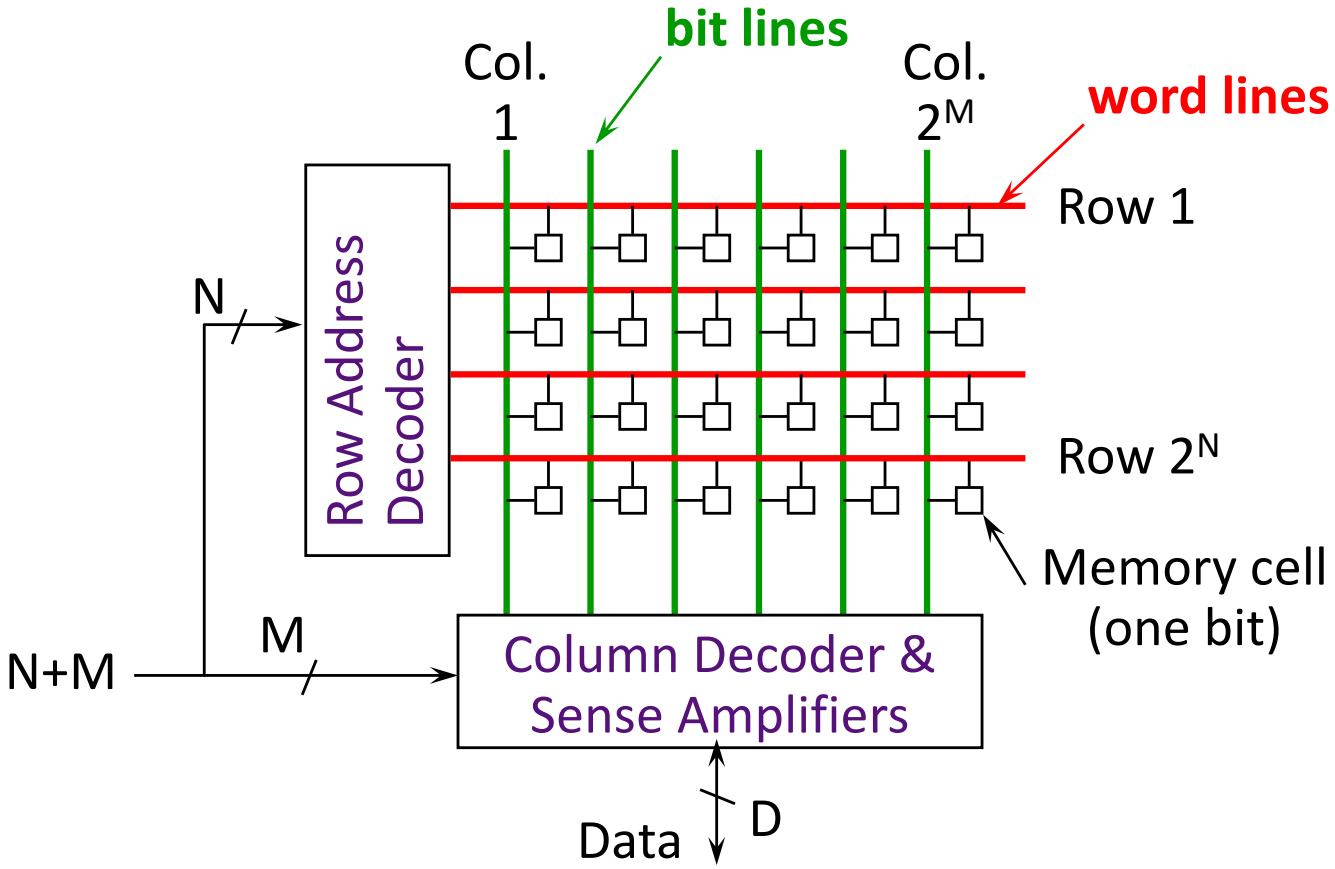
SRAM vs. DRAM

- DRAM = Dynamic RAM
- SRAM: 6T per bit
  - Requires low power to retain bit
  - Built with normal high-speed CMOS technology
- DRAM: 1T per bit
  - Built with special DRAM process optimized for density
  - Must be re-written after being read
  - Must also be periodically refreshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

Hardware Structures



DRAM Conceptual Architecture



- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4-8 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays

DRAM Physical Layout

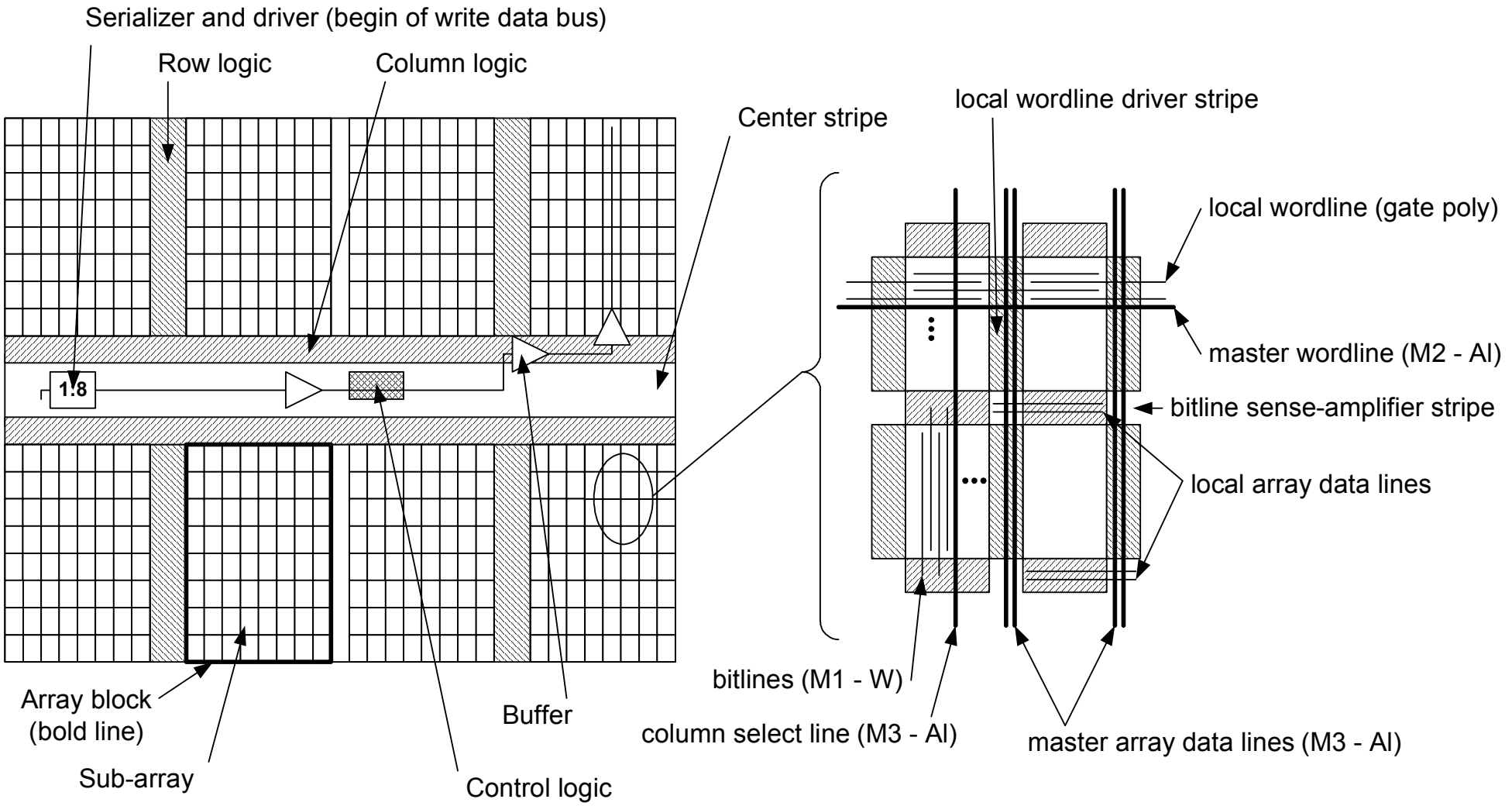
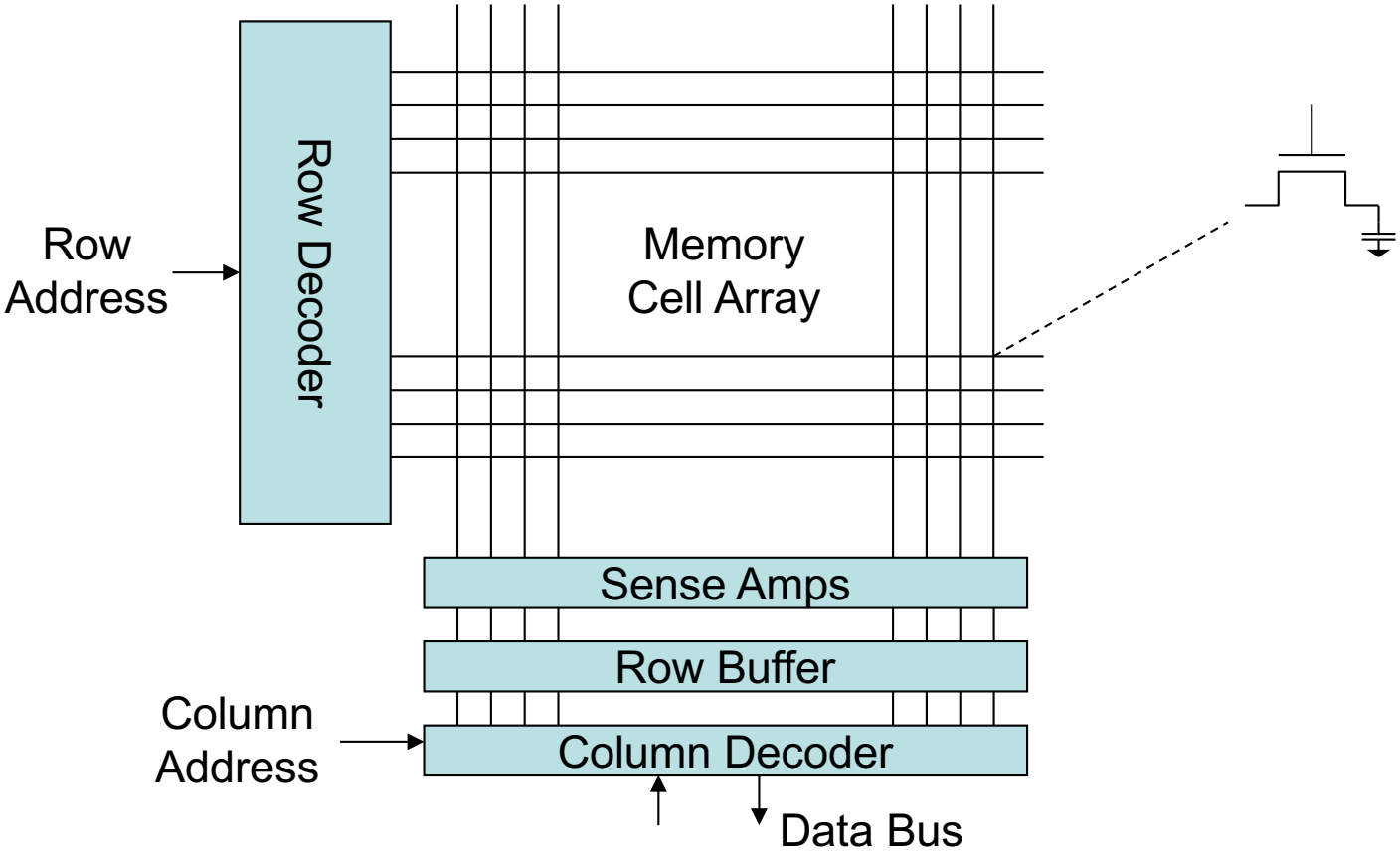


Figure 1. Physical floorplan of a DRAM. A DRAM actually contains a very large number of small DRAMs called sub-arrays.

[Vogelsang, MICRO-2010]

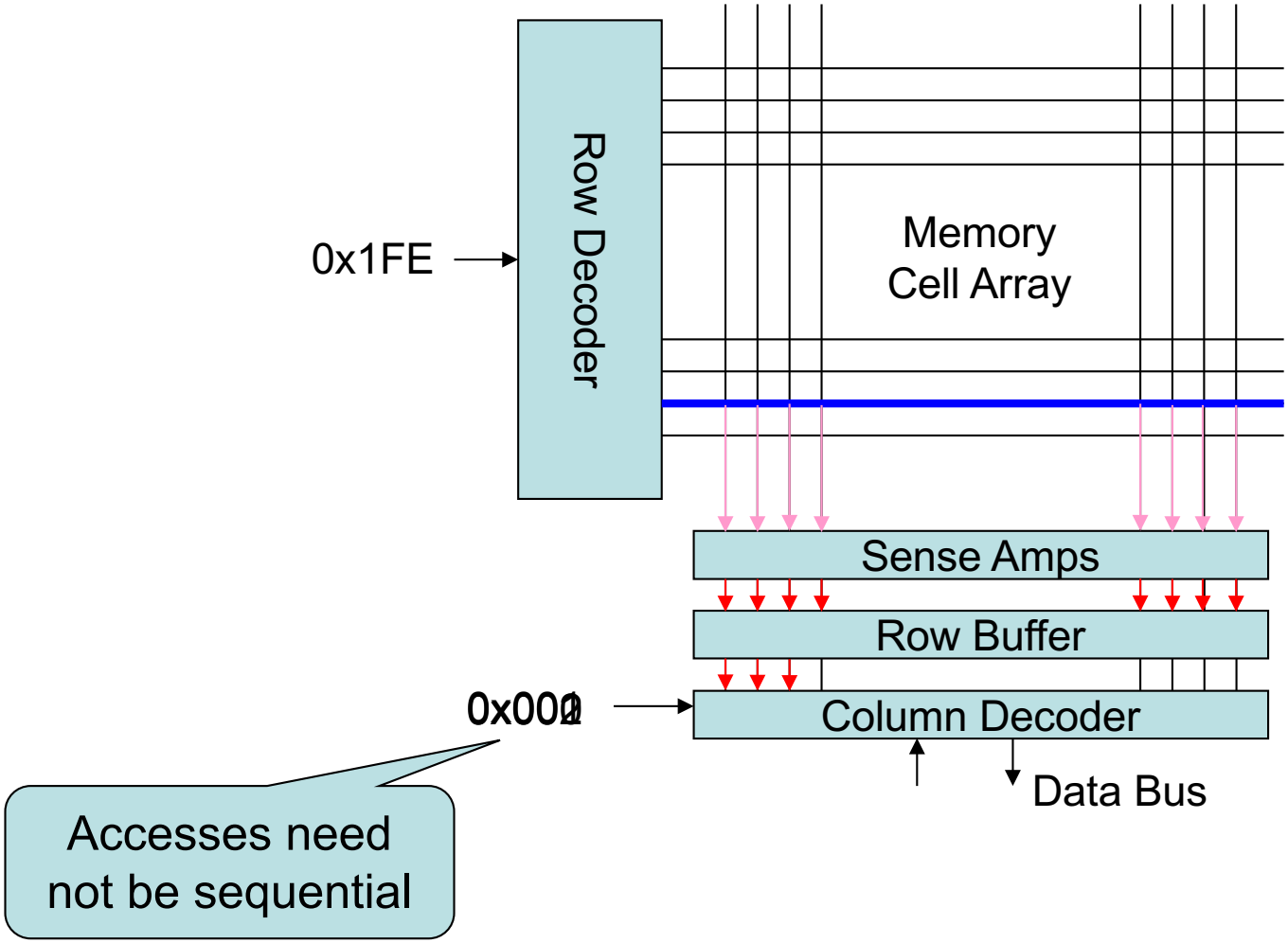
DRAM Chip Organization



DRAM Chip Organization (2)

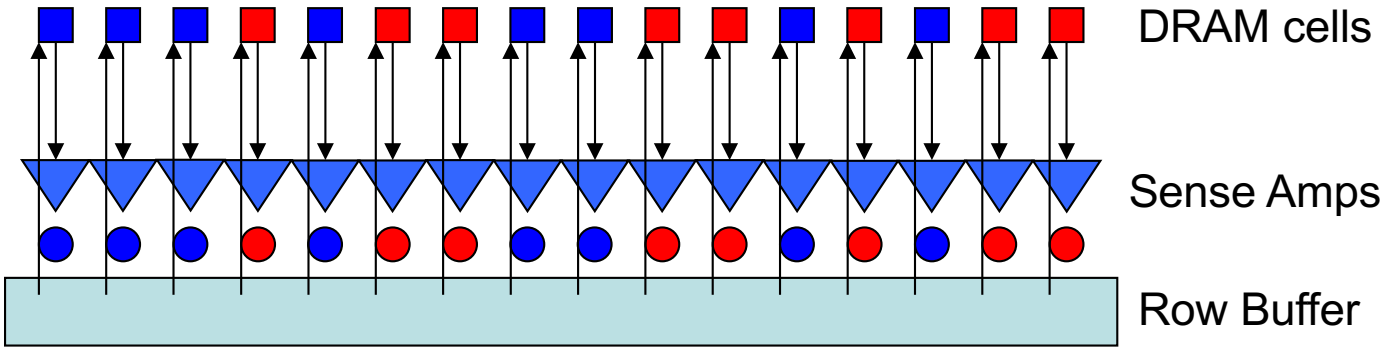
- Differences with SRAM
  - Reads are destructive: contents are erased after reading
  - Row buffer
  - Read lots of bits all at once, and then parcel them out based on different column addresses
    - similar to reading a full cache line, but only accessing one word at a time
  - “Fast-Page Mode” FPM DRAM organizes the DRAM row to contain bits for a complete page
    - row address held constant, and then fast read from different locations from the same page

DRAM Read Operation



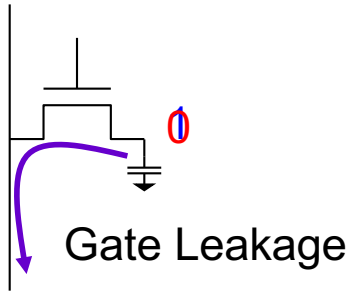
Refresh

- So after a read, the contents of the DRAM cell are gone
- The values are stored in the row buffer
- Write them back into the cells for the next read in the future



Refresh (2)

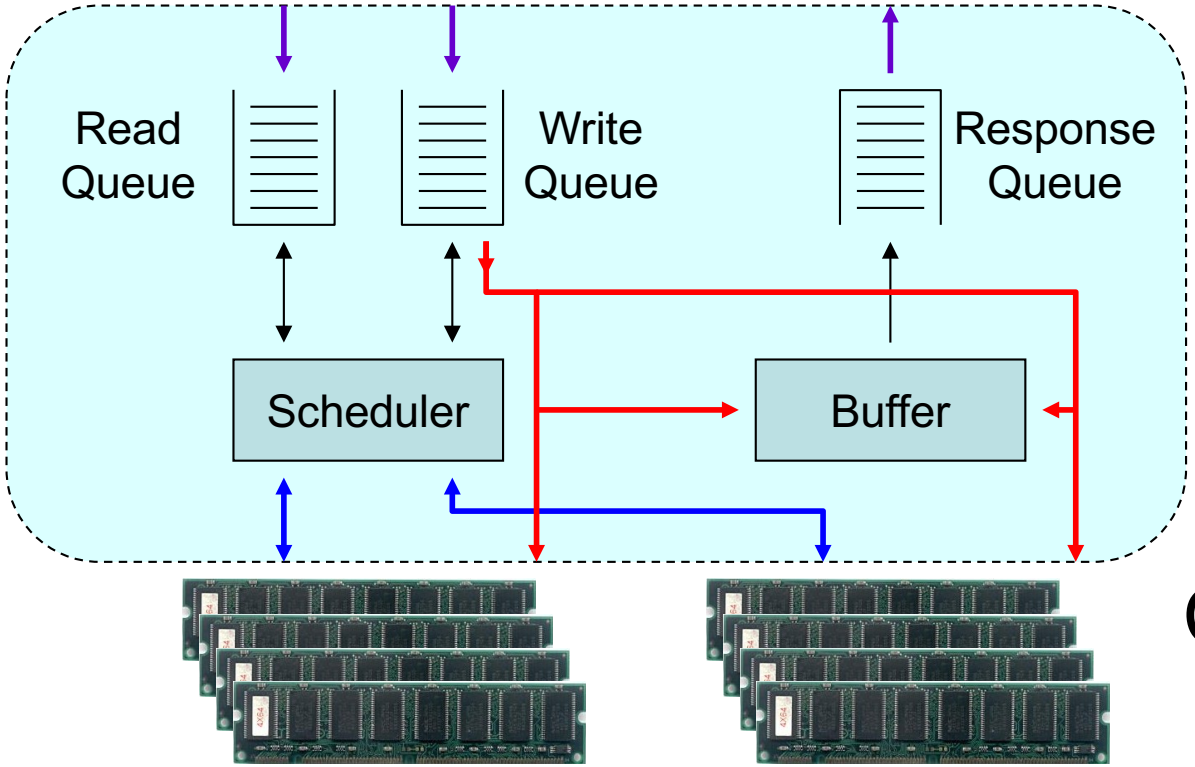
- Fairly gradually, the DRAM cell will lose its contents even if it's not accessed
  - This is why it's called "dynamic"
  - Contrast to SRAM which is "static" in that once written, it maintains its value forever (so long as power remains on)
- All DRAM rows need to be regularly read and re-written



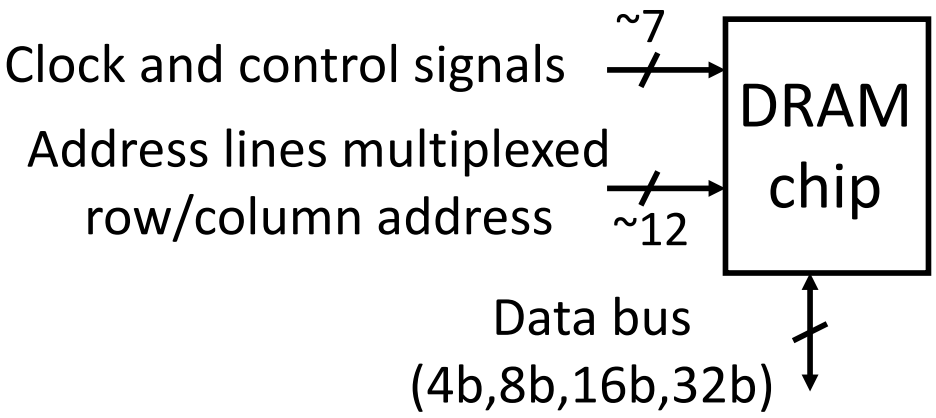
If it keeps its value even if power is removed, then it's "non-volatile" (e.g., flash, HDD, DVDs)

Memory Controller

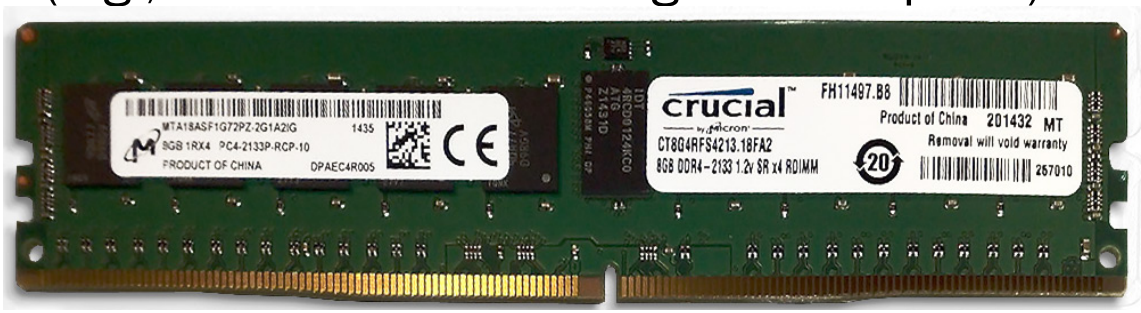
Like Write-Combining Buffer, Scheduler may coalesce multiple accesses together, or re-order to reduce number of row accesses



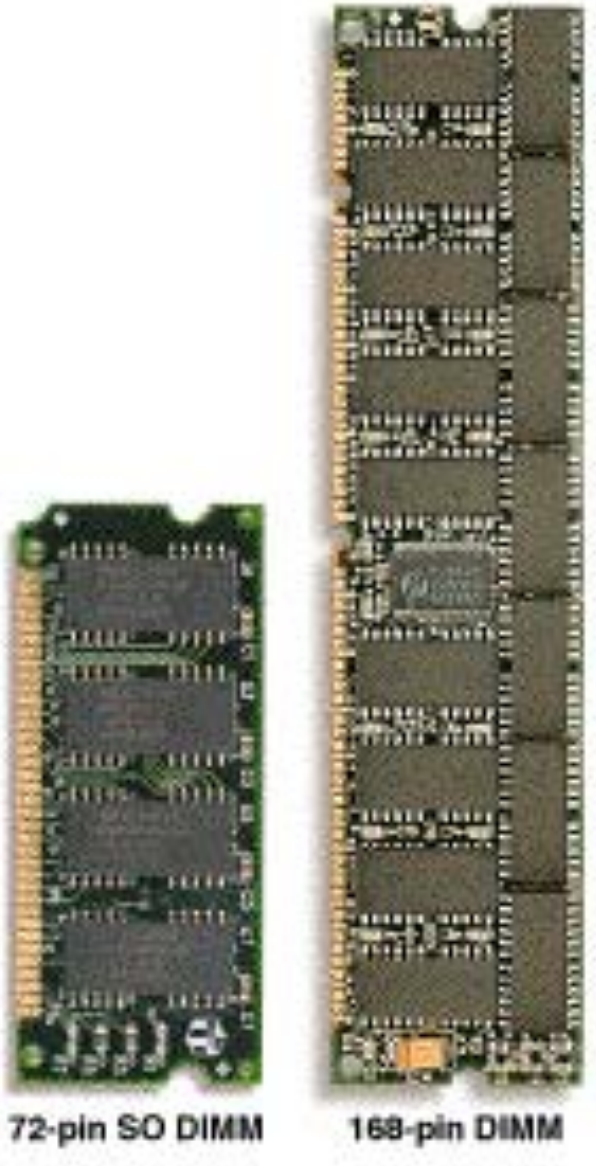
DRAM Packaging (Laptops/Desktops/Servers)



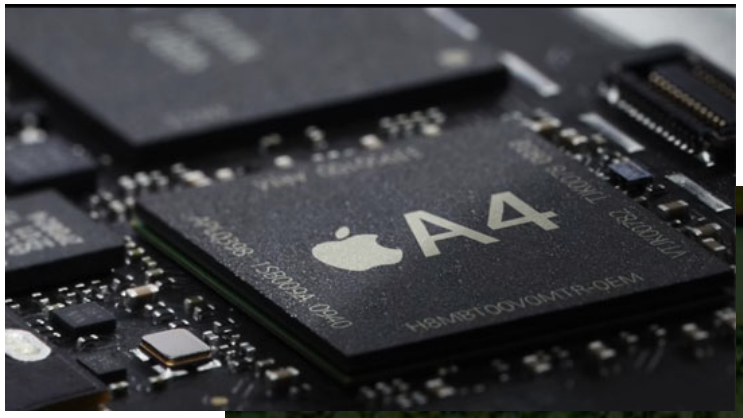
- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips)
- Data pins work together to return wide word (e.g., 64-bit data bus using 16x4-bit parts)



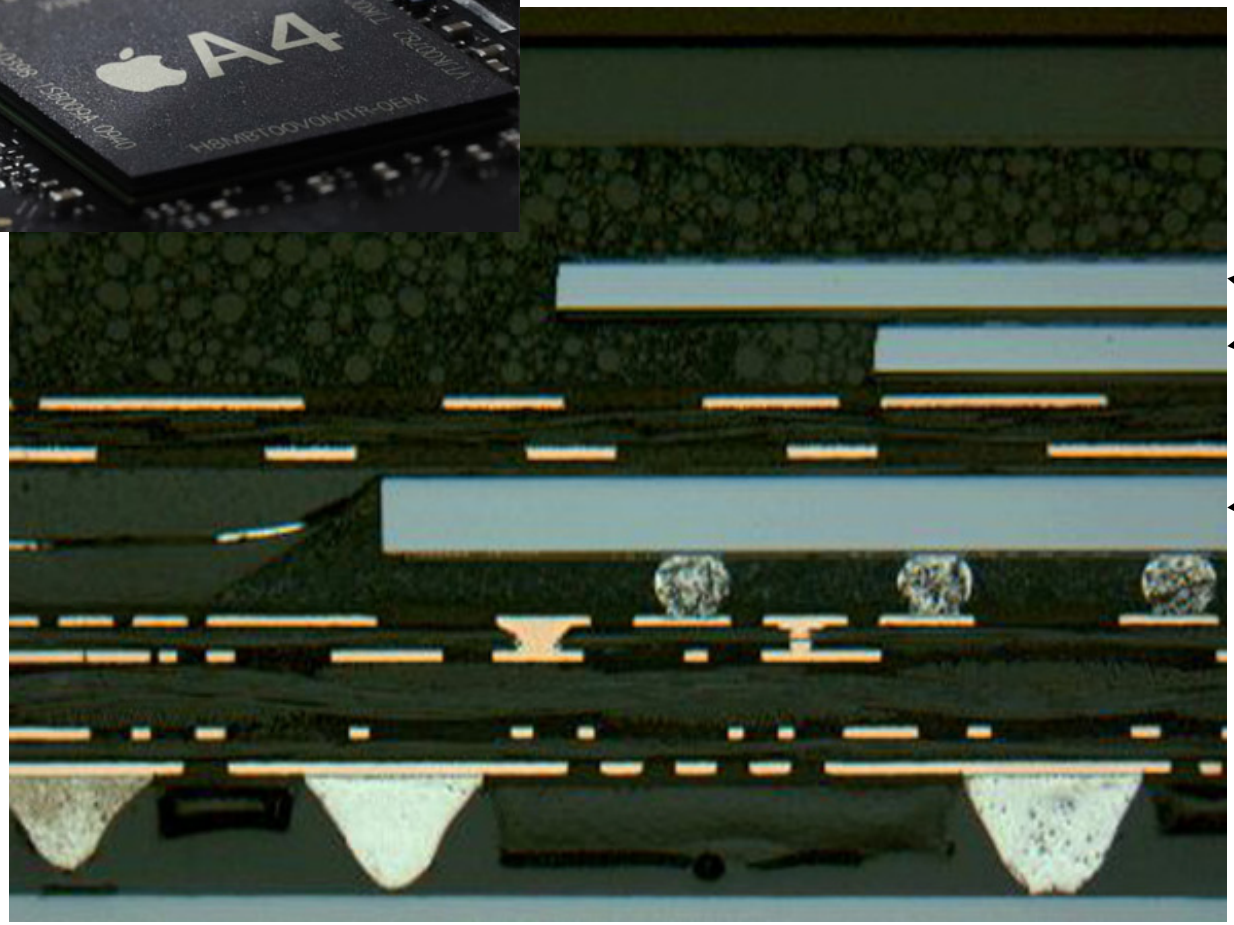
Two 8 GB DDR4-2133 288-pin ECC 1.2 V RDIMMs



DRAM Packaging, Mobile Devices



[Apple A4 package on circuit board]



Two stacked DRAM die

Processor plus logic die

[Apple A4 package cross-section, iFixit 2010]

DRAM Operation

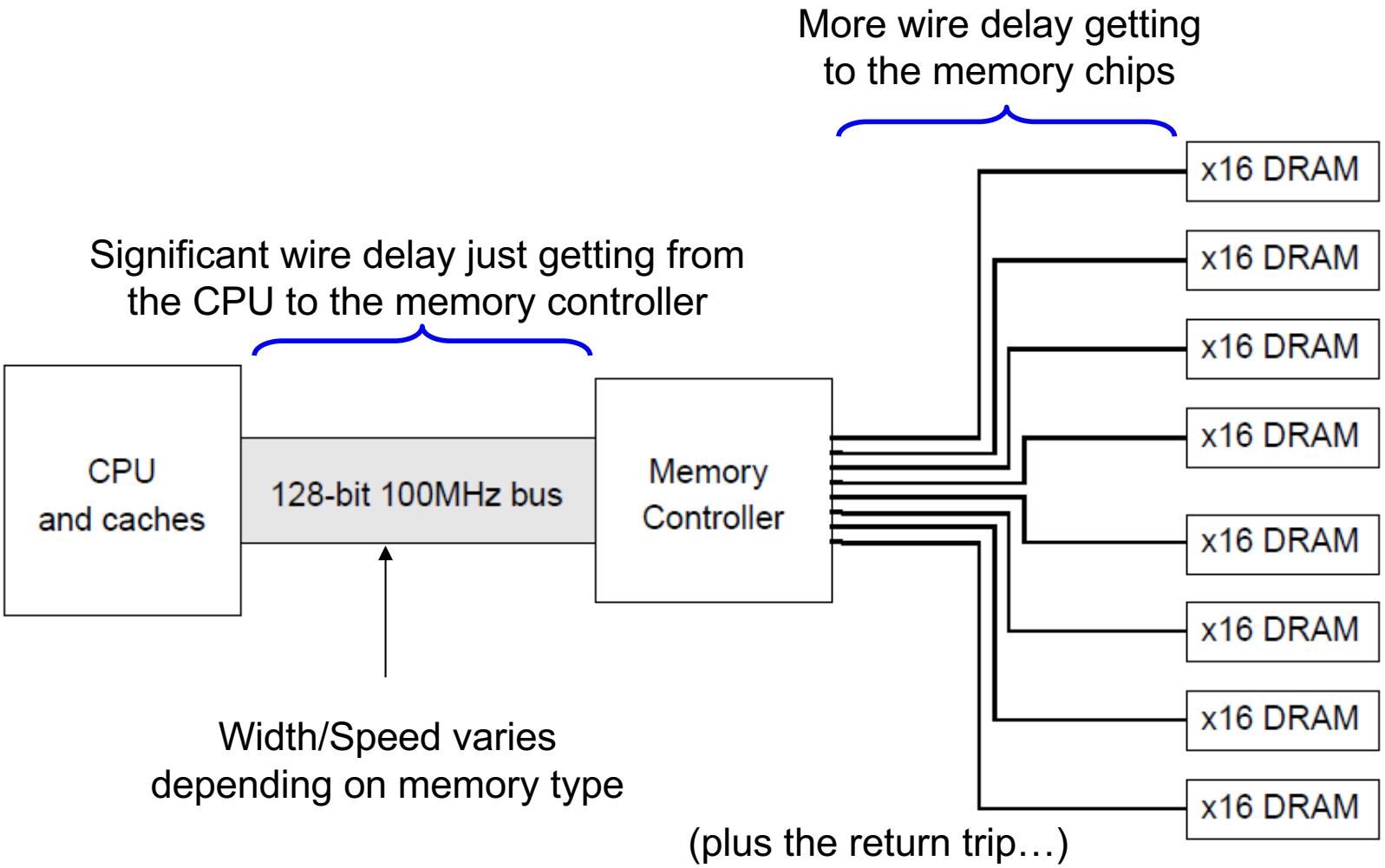
- Three steps in read/write access to a given bank
  - Row access (RAS)
    - decode row address, enable addressed row (often multiple Kb in row)
    - bitlines share charge with storage cell
    - small change in voltage detected by sense amplifiers which latch whole row of bits
    - sense amplifiers drive bitlines full rail to recharge storage cells
  - Column access (CAS)
    - decode column address to select small number of sense amplifier latches (4, 8, 16, or 32 bits depending on DRAM package)
    - on read, send latched bits out to chip pins
    - on write, change sense amplifier latches which then charge storage cells to required value
    - can perform multiple column accesses on same row without another row access (burst mode)
  - Precharge
    - charges bit lines to known value, required before next row access
- Each step has a latency of around 15-20ns in modern DRAMs
- Various DRAM standards (DDR, RDRAM) have different ways of encoding the signals for transmission to the DRAM, but all share same core architecture

Memory Parameters

- Latency
  - Time from initiation to completion of one memory read (e.g., in nanoseconds, or in CPU or DRAM clock cycles)
- Occupancy
  - Time that a memory bank is busy with one request
  - Usually the important parameter for a memory write
- Bandwidth
  - Rate at which requests can be processed (accesses/sec, or GB/s)

- All can vary significantly for reads vs. writes, or address, or address history (e.g., open/close page on DRAM bank)

More Latency

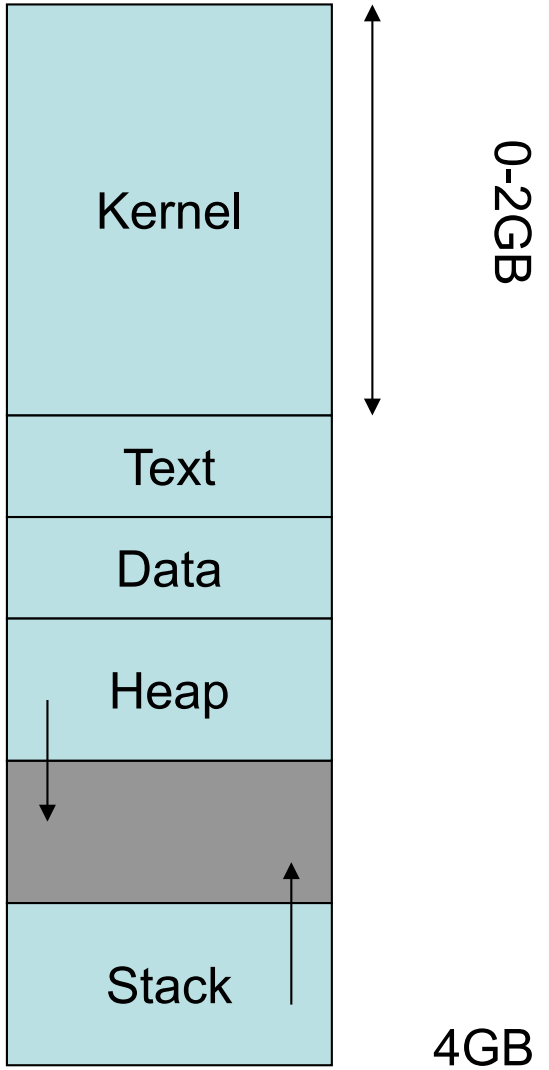


Views of Memory

- Real machines have limited amounts of memory
  - 640KB? 4GB? 16 GB?
- Programmer doesn't want to be bothered
  - Do you think, "oh, this computer only has 128MB so I'll write my code this way…"
  - What happens if you run on a different machine?

Programmer's View

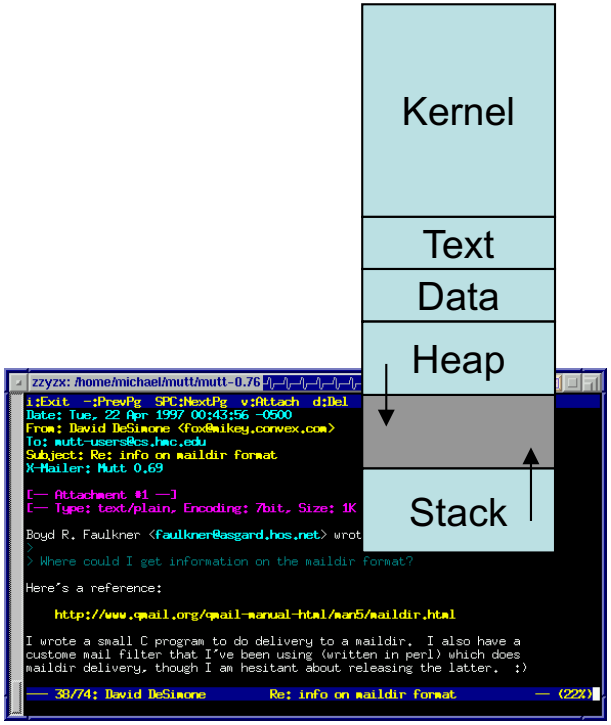
- Example 32-bit memory
  - When programming, you don't care about how much *real* memory there is
  - Even if you use a lot, memory can always be paged to disk

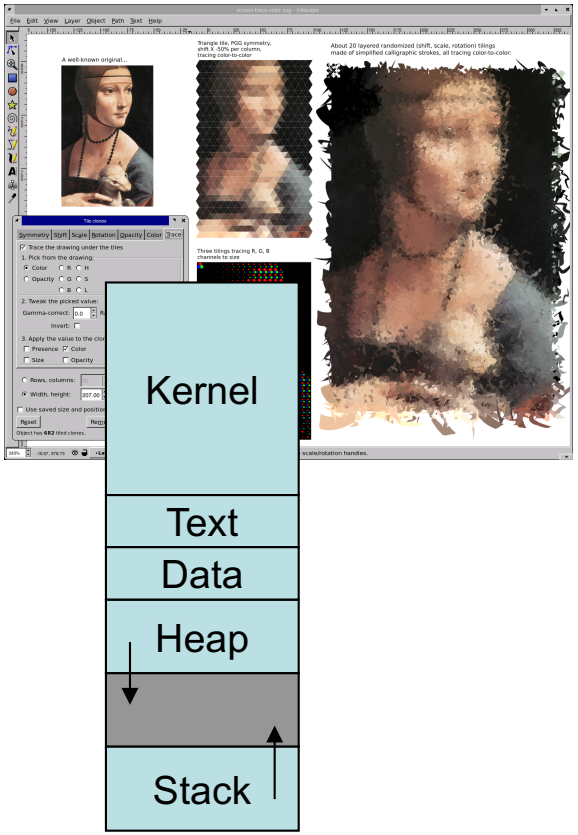


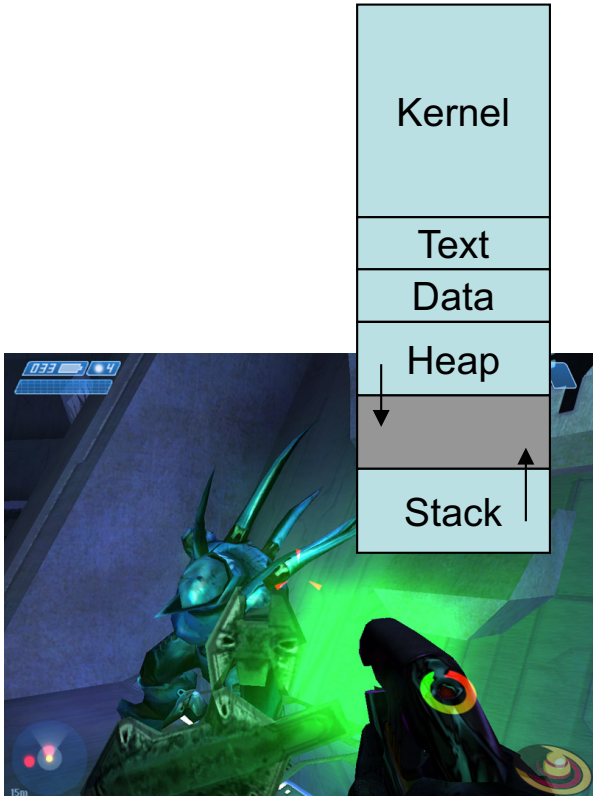
AKA Virtual Addresses

Programmer's View

- Really "Program's View"
- Each program/process gets its own 4GB space







CPU's View

- At some point, the CPU is going to have to load-from/store-to memory… all it knows is the real, A.K.A. ***physical*** memory

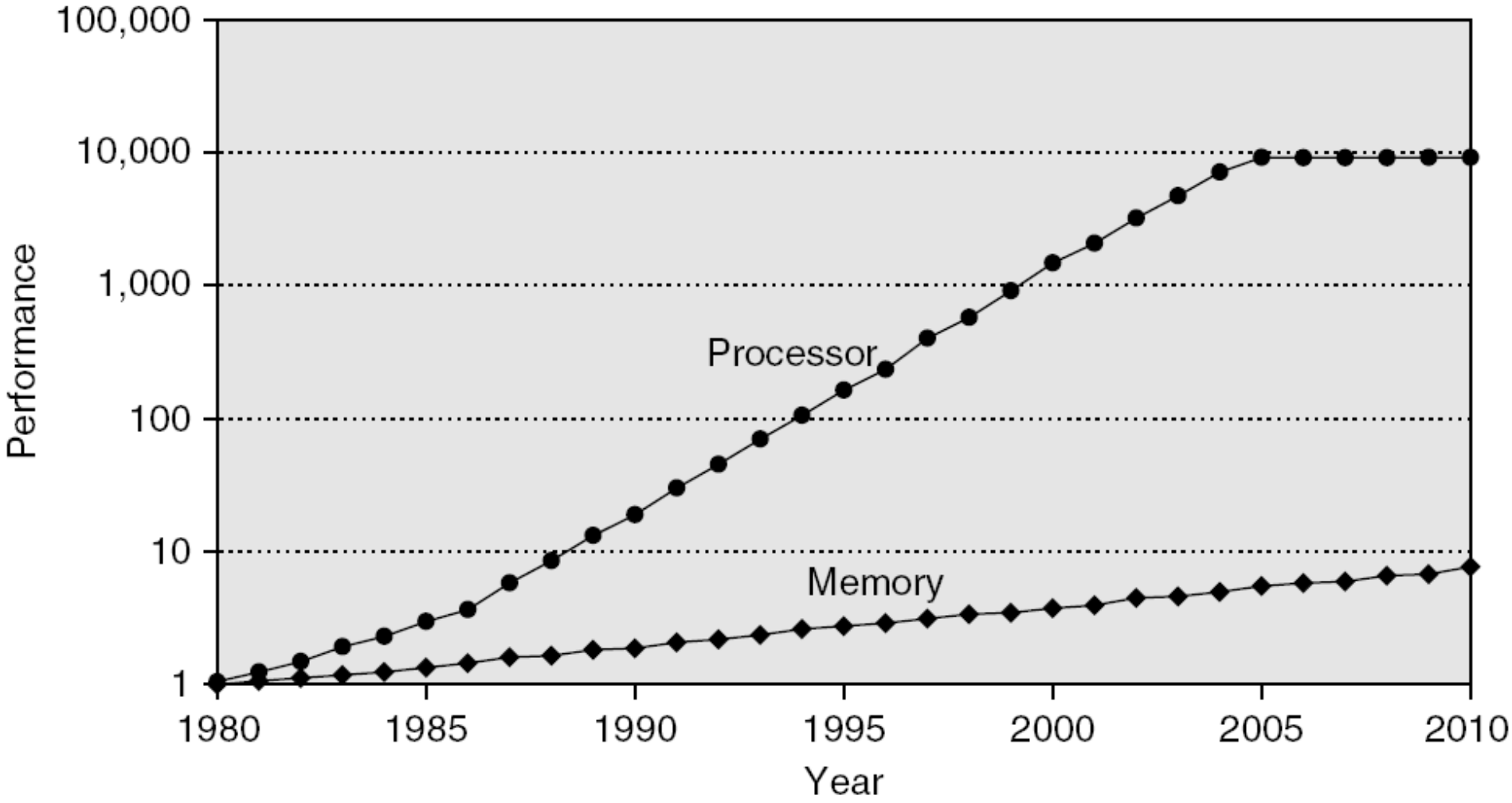
- … which these days

  between 4-32 GB

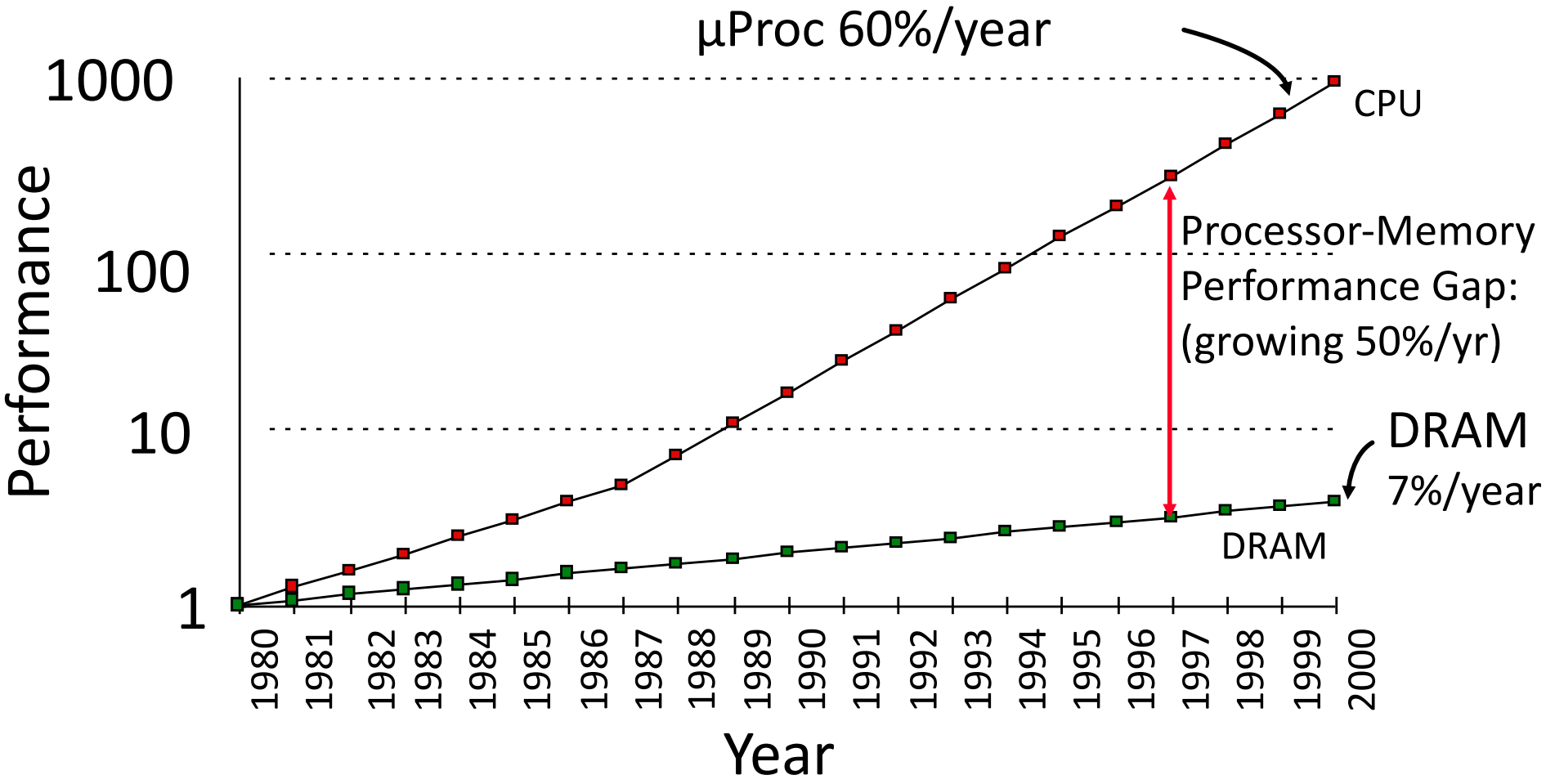
Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors
- Some optimizations:
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device

Processor-DRAM Gap (latency)



Processor-DRAM Gap (latency)



Four-issue 3GHz superscalar accessing 100ns DRAM could execute 1,200 instructions during time for one memory access!

Memory Optimizations

Production year	Chip size	DRAM Type	Row access strobe (RAS)		Column access strobe (CAS)/ data transfer time (ns)	Cycle time (ns)
			Slowest DRAM (ns)	Fastest DRAM (ns)		
1980	64K bit	DRAM	180	150	75	250
1983	256K bit	DRAM	150	120	50	220
1986	1M bit	DRAM	120	100	25	190
1989	4M bit	DRAM	100	80	20	165
1992	16M bit	DRAM	80	60	15	120
1996	64M bit	SDRAM	70	50	12	110
1998	128M bit	SDRAM	70	50	10	100
2000	256M bit	DDR1	65	45	7	90
2002	512M bit	DDR1	60	40	5	80
2004	1G bit	DDR2	55	35	5	70
2006	2G bit	DDR2	50	30	2.5	60
2010	4G bit	DDR3	36	28	1	37
2012	8G bit	DDR3	30	24	0.5	31

Figure 2.13 Times of fast and slow DRAMs vary with each generation. (Cycle time is defined on page 95.) Performance improvement of row access time is about 5% per year. The improvement by a factor of 2 in column access in 1986 accompanied the switch from NMOS DRAMs to CMOS DRAMs. The introduction of various burst transfer modes in the mid-1990s and SDRAMs in the late 1990s has significantly complicated the calculation of access time for blocks of data; we discuss this later in this section when we talk about SDRAM access time and power. The DDR4 designs are due for introduction in mid- to late 2012. We discuss these various forms of DRAMs in the next few pages.

Memory Optimizations

Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	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	1066–1600	2133–3200	DDR4-3200	17,056–25,600	PC25600

Figure 2.14 Clock rates, bandwidth, and names of DDR DRAMS and DIMMs in 2010. 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. Although not shown in this figure, DDRs also specify latency in clock cycles as four numbers, which are specified by the DDR standard. For example, DDR3-2000 CL 9 has latencies of 9-9-9-28. What does this mean? With a 1 ns clock (clock cycle is one-half the transfer rate), this indicate 9 ns for row to columns address (RAS time), 9 ns for column access to data (CAS time), and a minimum read time of 28 ns. Closing the row takes 9 ns for precharge but happens only when the reads from that row are finished. In burst mode, transfers occur on every clock on both edges, when the first RAS and CAS times have elapsed. Furthermore, the precharge in not needed until the entire row is read. DDR4 will be produced in 2012 and is expected to reach clock rates of 1600 MHz in 2014, when DDR5 is expected to take over. The exercises explore these details further.

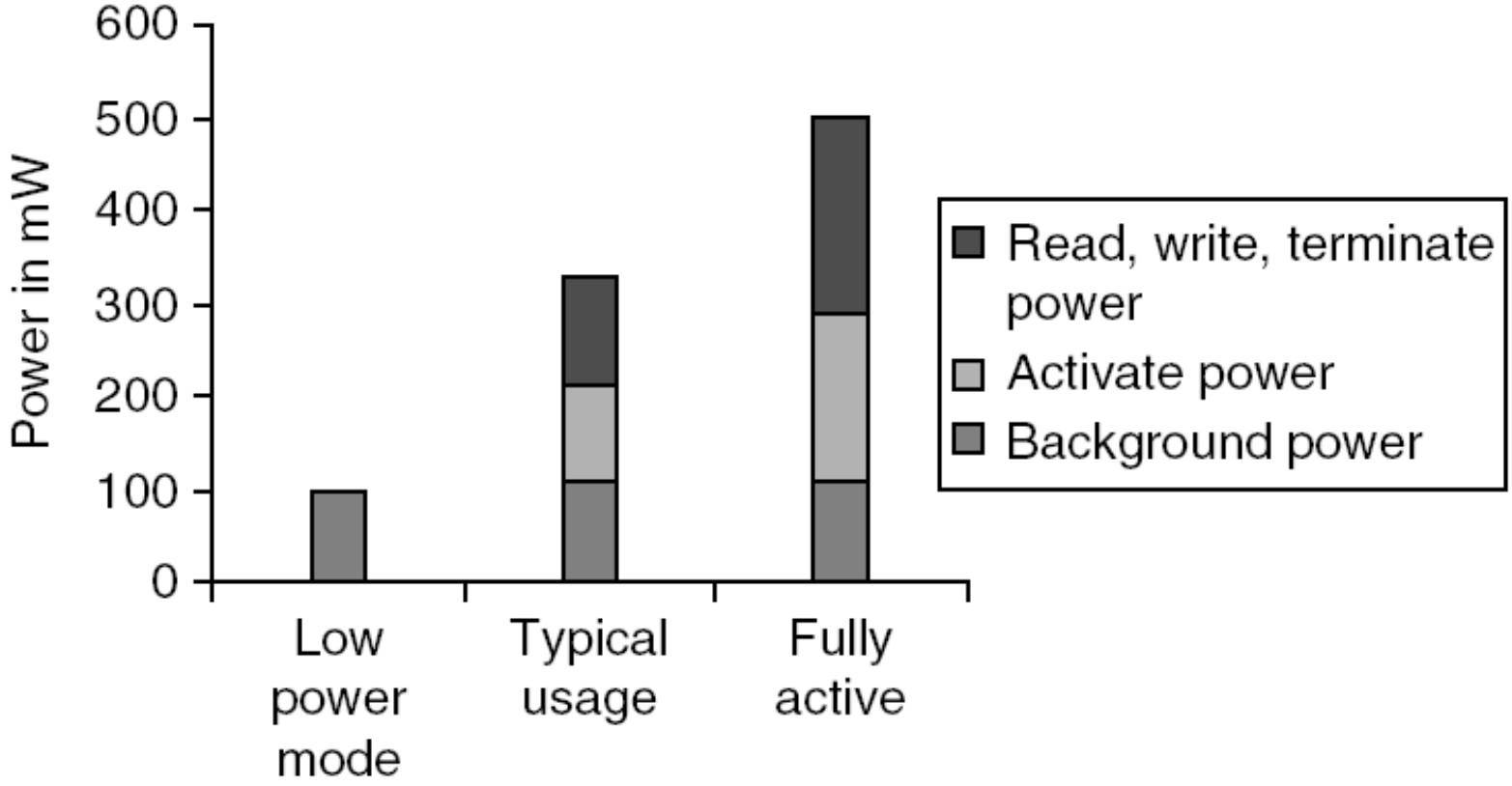
Memory Optimizations

- DDR:
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz
- GDDR5 is graphics memory based on DDR3

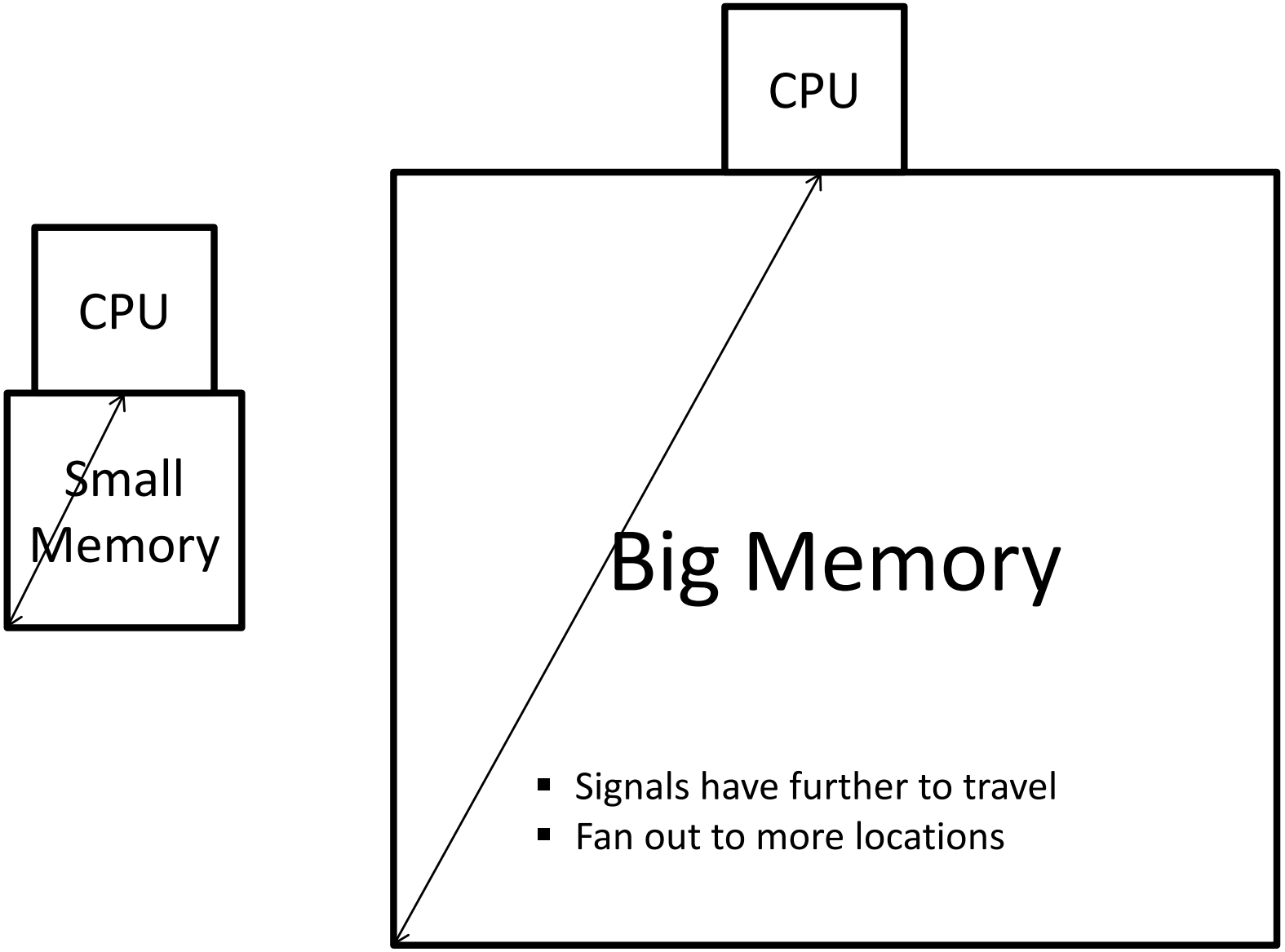
Memory Optimizations

- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketted DIMM modules
- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

Memory Power Consumption

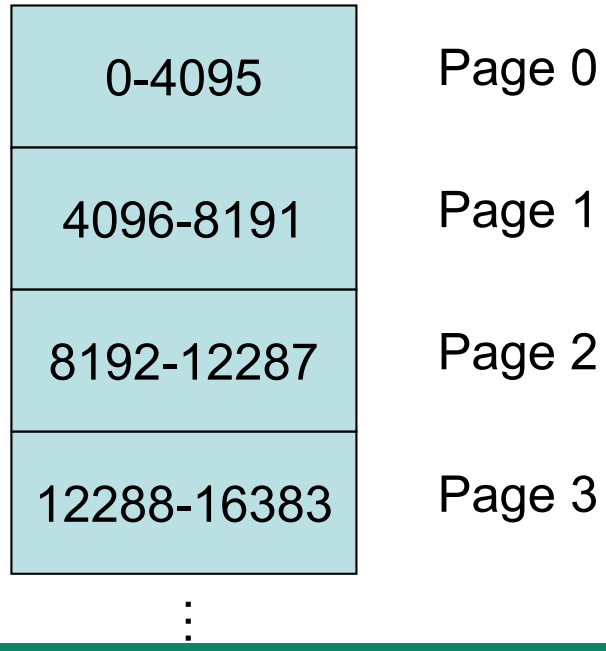


Physical Size Affects Latency



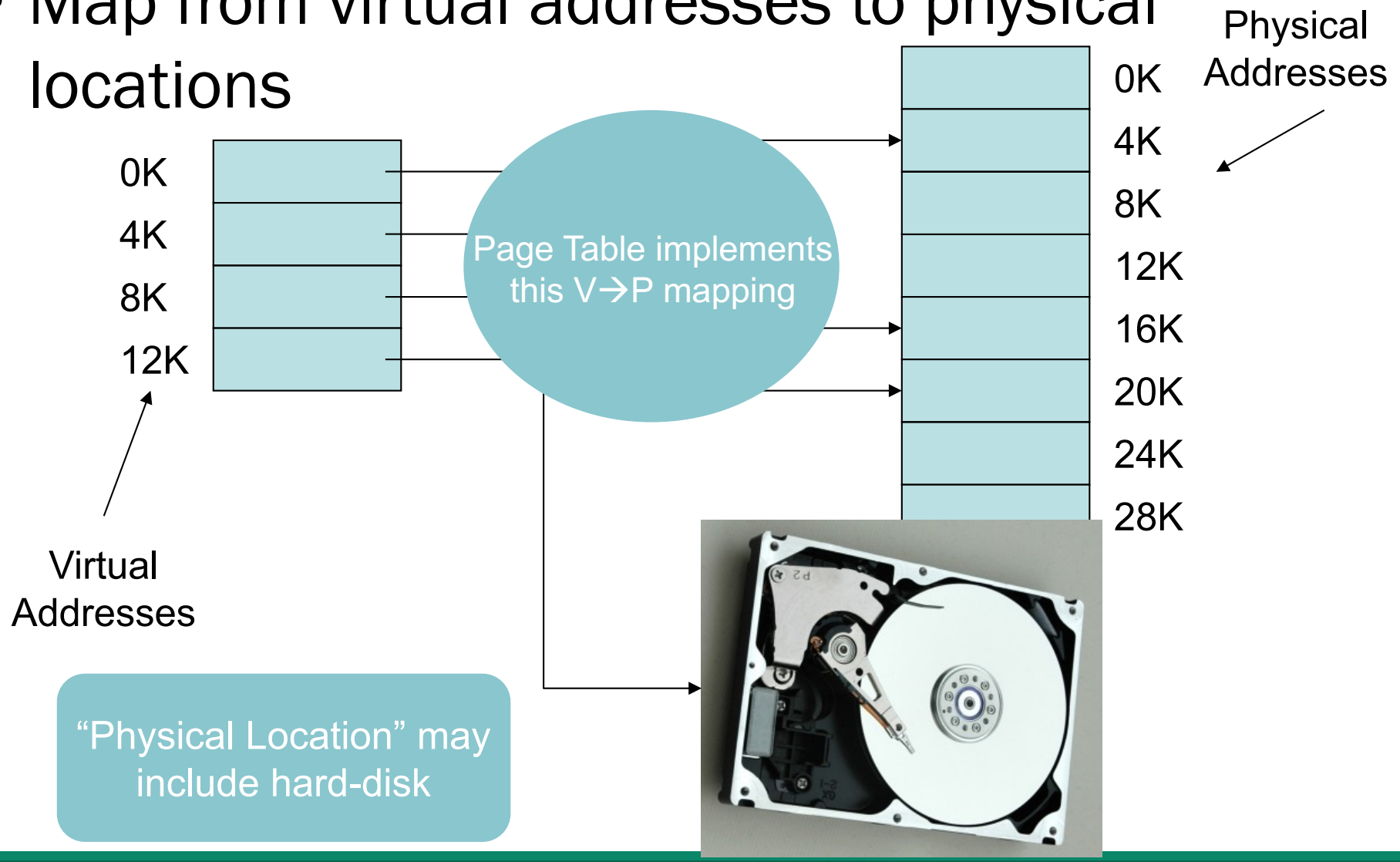
Pages

- Memory is divided into *pages*, which are nothing more than fixed sized and aligned regions of memory
  - Typical size: 4KB/page (but not always)

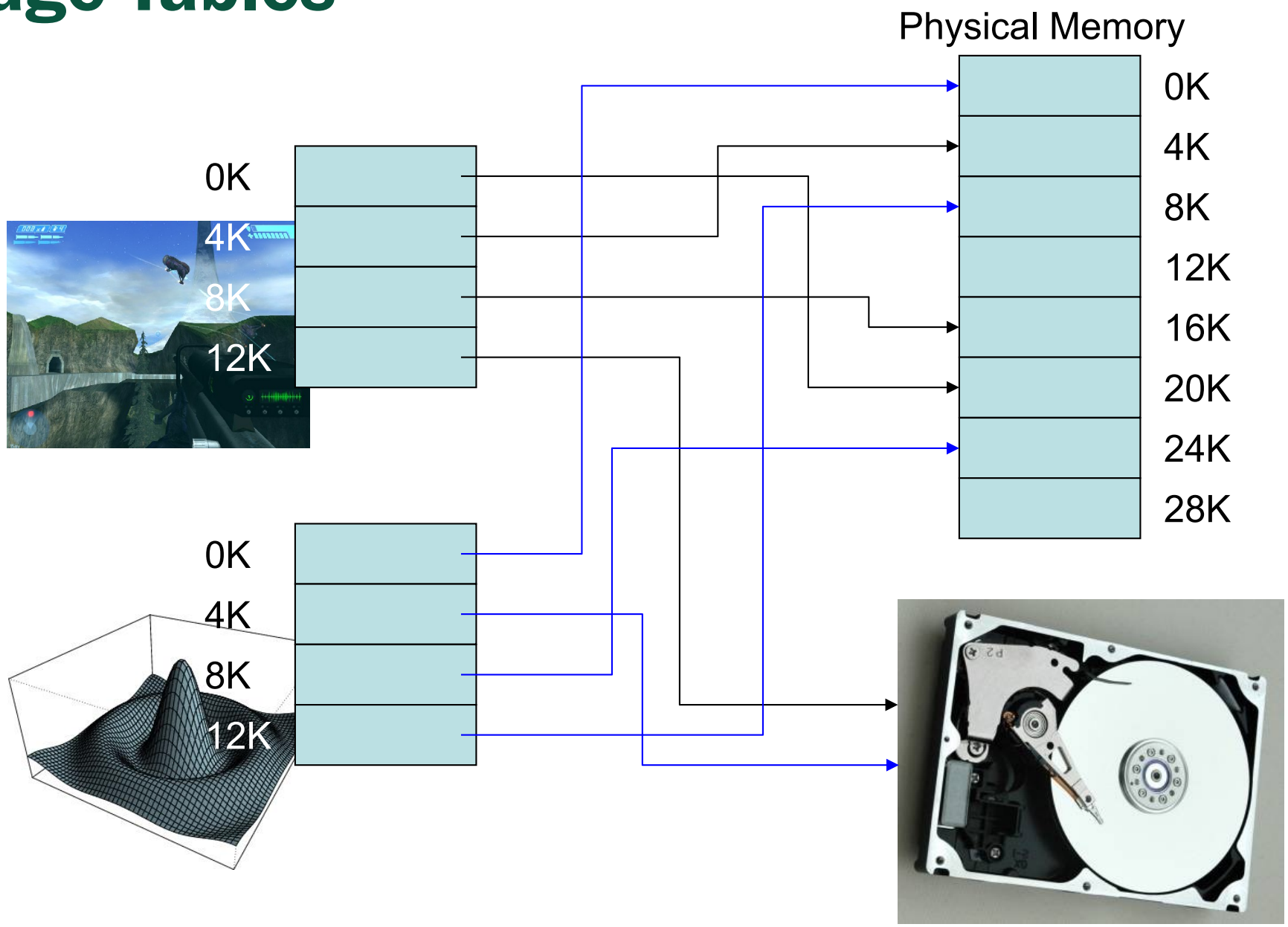


Page Table

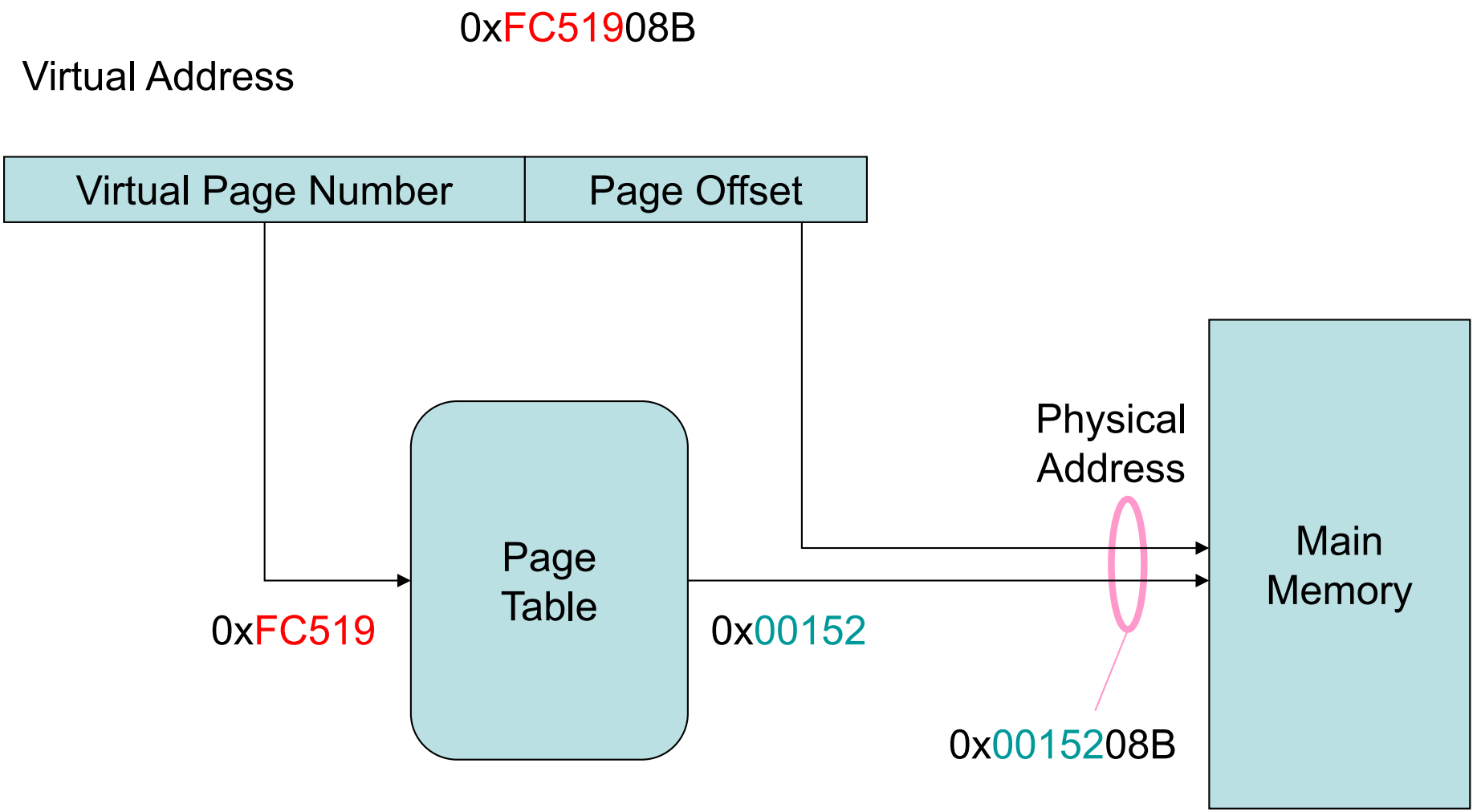
- Map from virtual addresses to physical locations



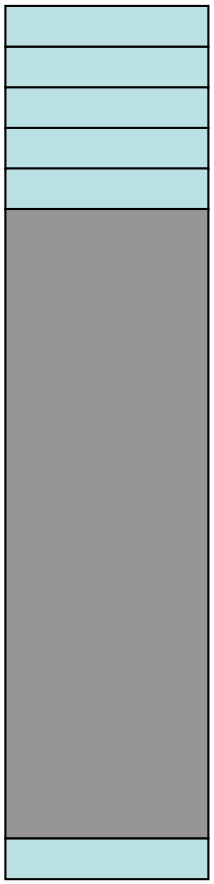
Page Tables


Physical Memory
0K
4K
8K
12K
16K
20K
24K
28K
0K
4K
8K
12K
0K
4K
8K
12K

Need for Translation



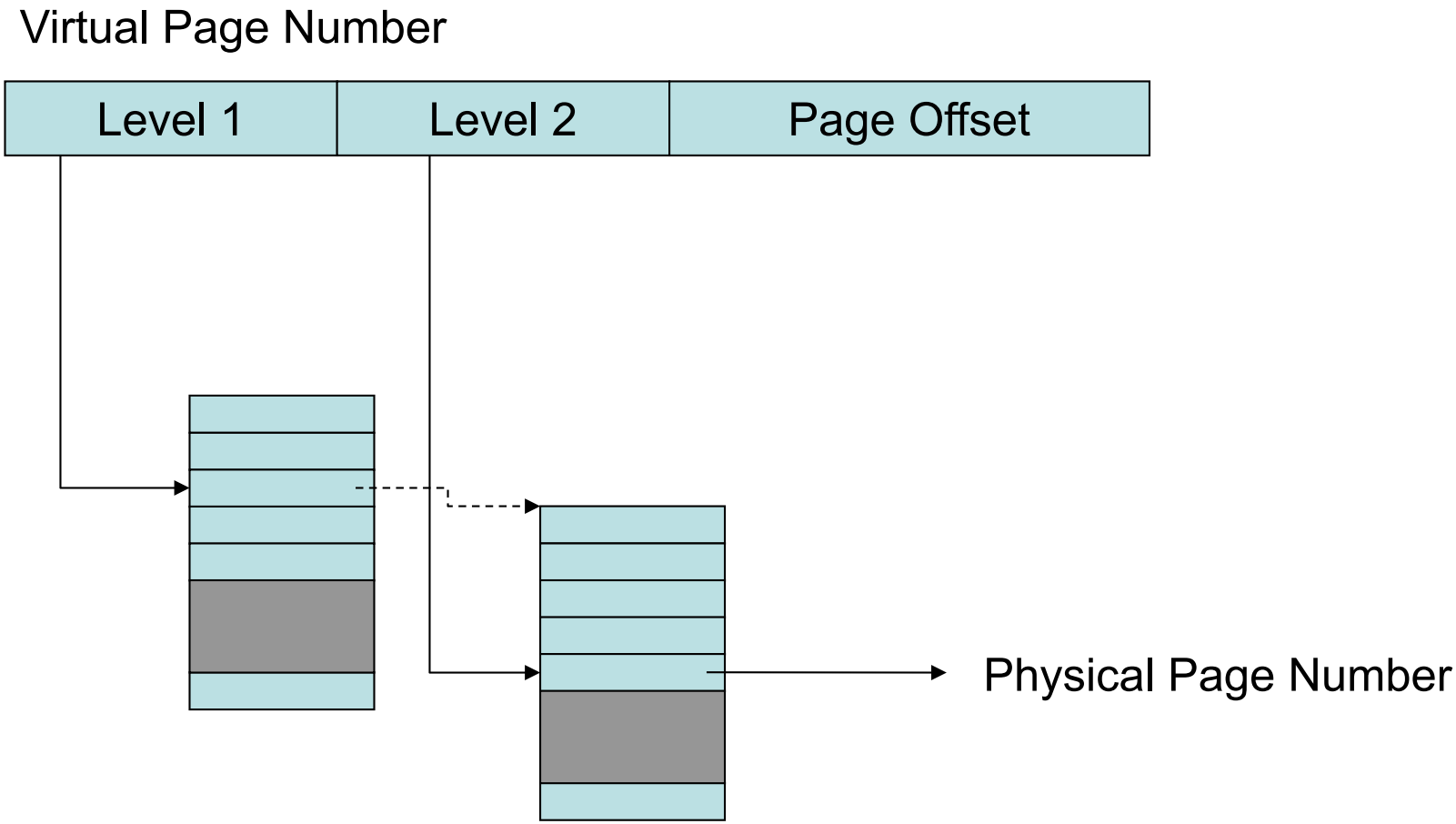
Simple Page Table



One entry per page

- Flat organization
  - One entry per page
  - Entry contains physical page number (PPN) or indicates page is on disk or invalid
  - Also meta-data (e.g., permissions, dirtiness, etc.)

Multi-Level Page Tables



Choosing a Page Size

- Page size inversely proportional to page table overhead
- Large page size permits more efficient transfer to/from disk
  - vs. many small transfers
  - Like downloading from Internet
- Small page leads to less fragmentation
  - Big page likely to have more bytes unused

CPU Memory Access

- Program deals with virtual addresses
  - "Load R1 = 0[R2]"
- On memory instruction
  1. Compute virtual address (0[R2])
  2. Compute virtual page number
  3. Compute physical address of VPN's page table entry
  4. Load* mapping
  5. Compute physical address
  6. Do the actual Load* from memory

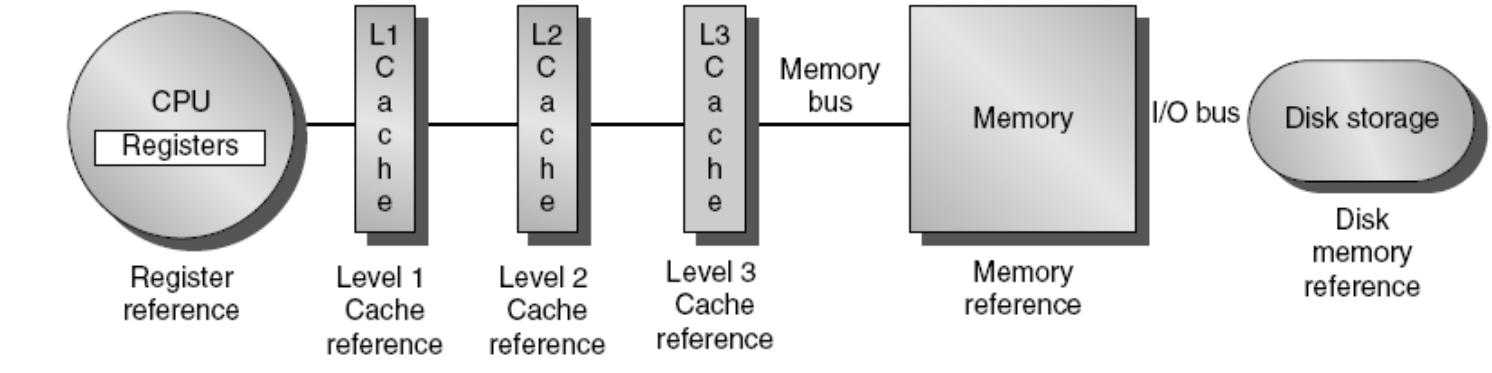
Could be more depending On page table organization

Impact on Performance?

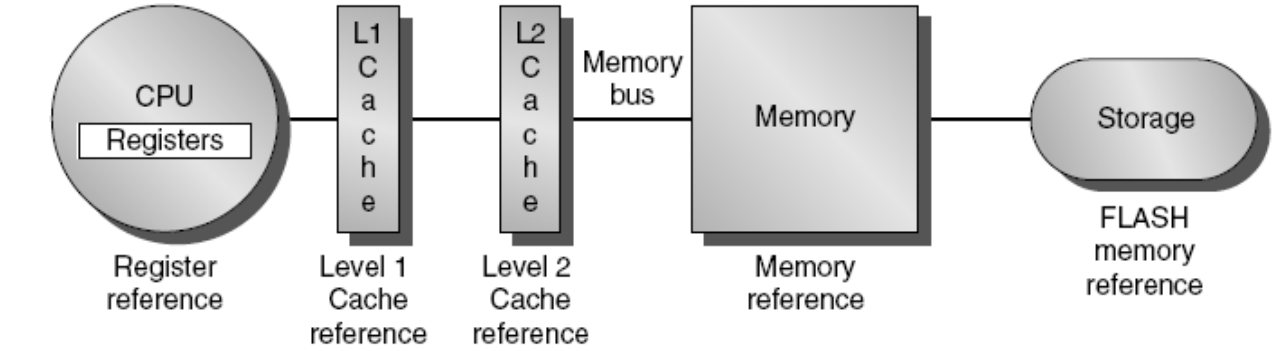
- Every time you load/store, the CPU must perform two (or more) accesses!
- Even worse, every *fetch* requires translation of the PC!

- Observation:
  - Once a virtual page is mapped into a physical page, it'll likely stay put for quite some time

Idea: Memory Hierarchy



(a) Memory hierarchy for server



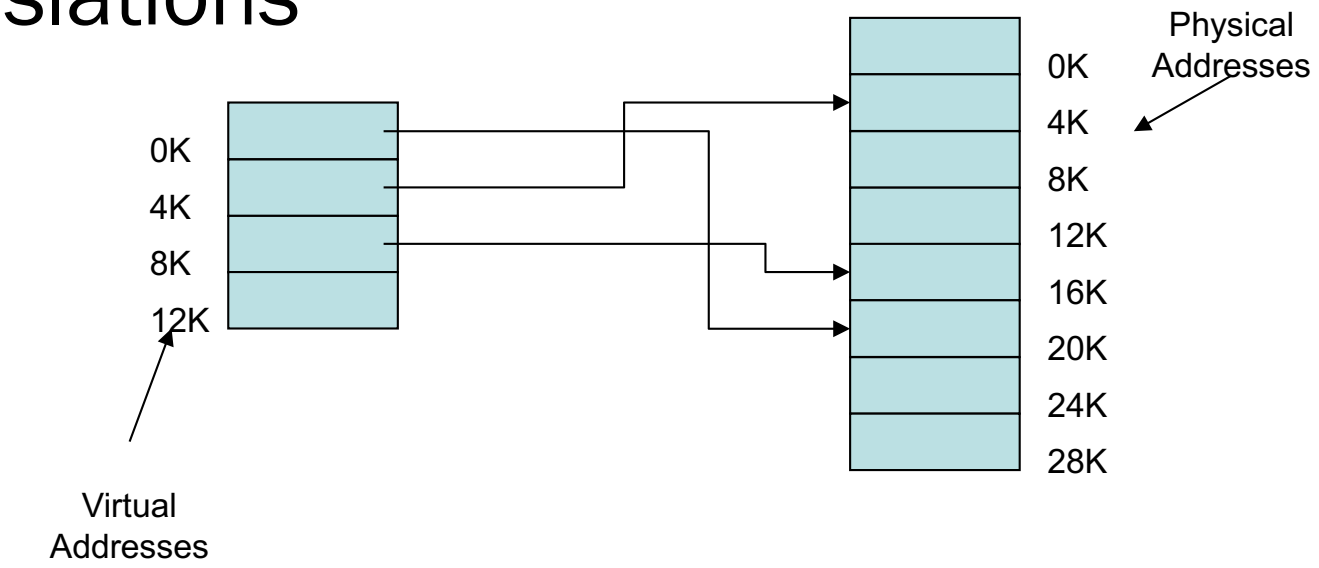
(b) Memory hierarchy for a personal mobile device

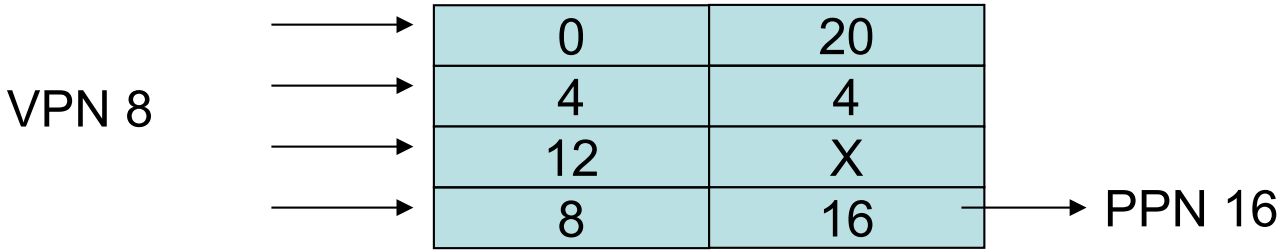
Idea: Caching!

- Organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

Idea: Caching!

- Not caching of data, but caching of translations





Memory Reference Scheduling

- Just like registers, need to enforce RAW, WAW, WAR dependencies
- No “memory renaming” in memory controller, so enforce all three dependencies

- Like everything else, still need to maintain appearance of sequential access
  - Consider multiple read/write requests to the same address

So what do we do about it?

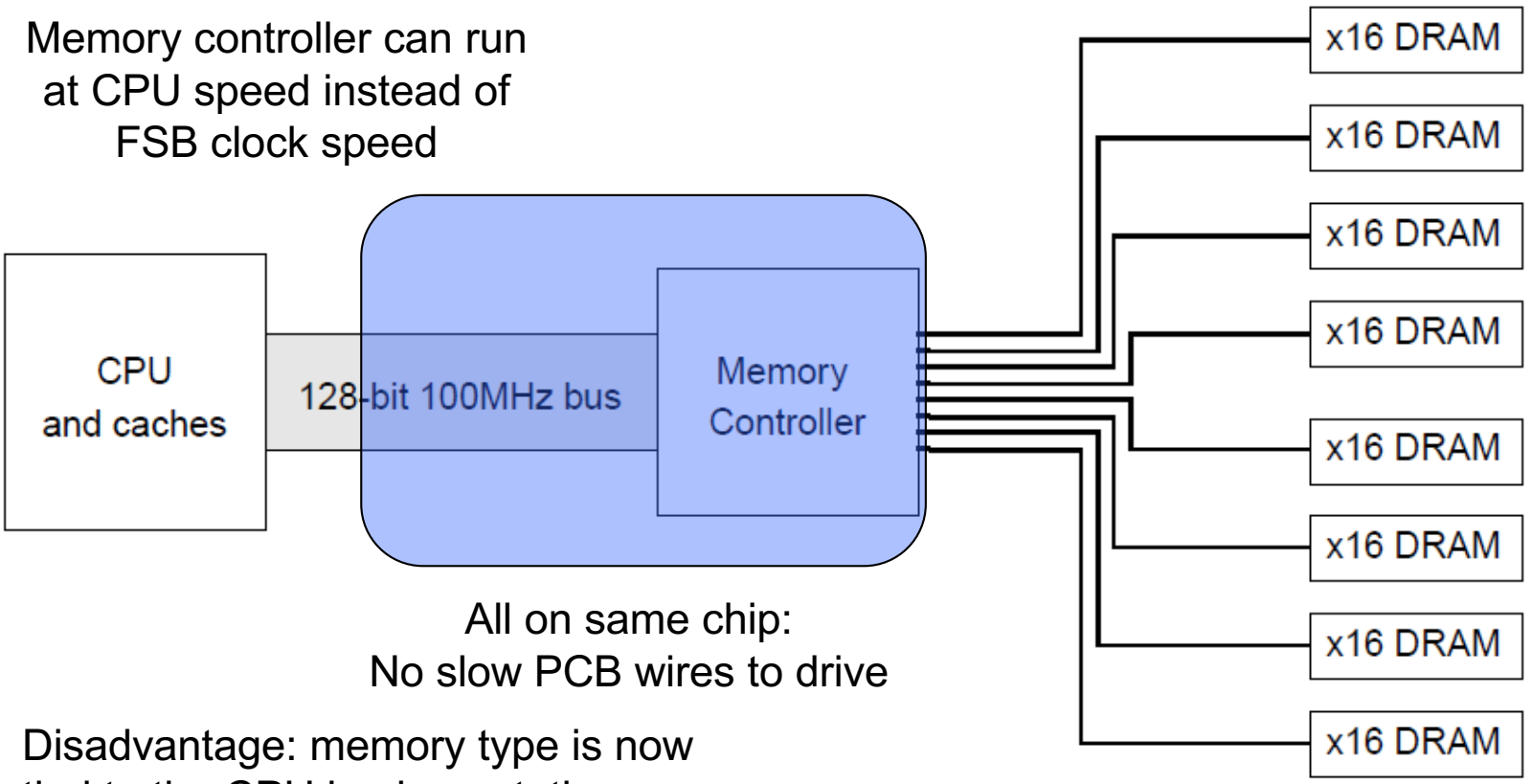
- Caching
  - reduces average memory instruction latency by avoiding DRAM altogether

- Limitations
  - Capacity
    - programs keep increasing in size
  - Compulsory misses

On-Chip Memory Controller

Also: more sophisticated memory scheduling algorithms

Memory controller can run at CPU speed instead of FSB clock speed



All on same chip:
No slow PCB wires to drive

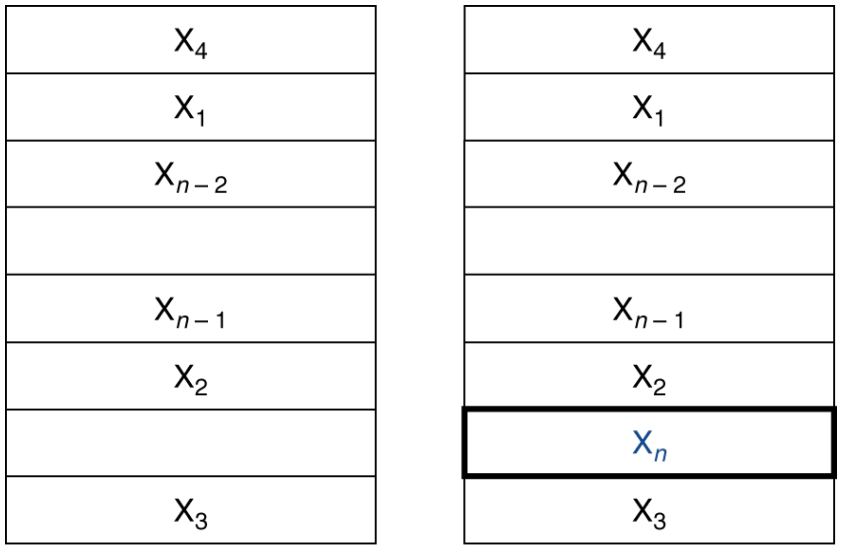
Disadvantage: memory type is now tied to the CPU implementation

Idea: Memory Hierarchy Basics

- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets

Memory Hierarchy Basics

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses X1, ..., Xn–1, Xn



a. Before the reference to X_n b. After the reference to X_n

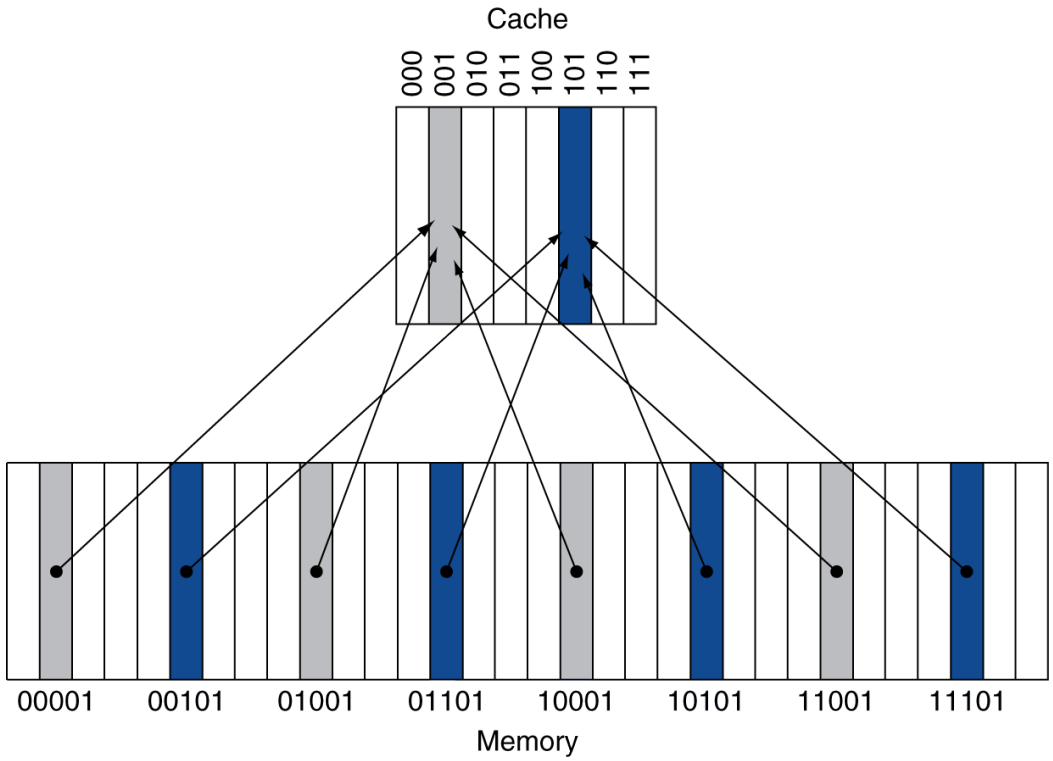
- How do we know if the data is present?
- Where do we look?

Memory Hierarchy Basics

- *n* sets => *n-way set associative*
  - *Direct-mapped cache* => one block per set
  - *Fully associative* => one set

Direct Mapped Cache

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

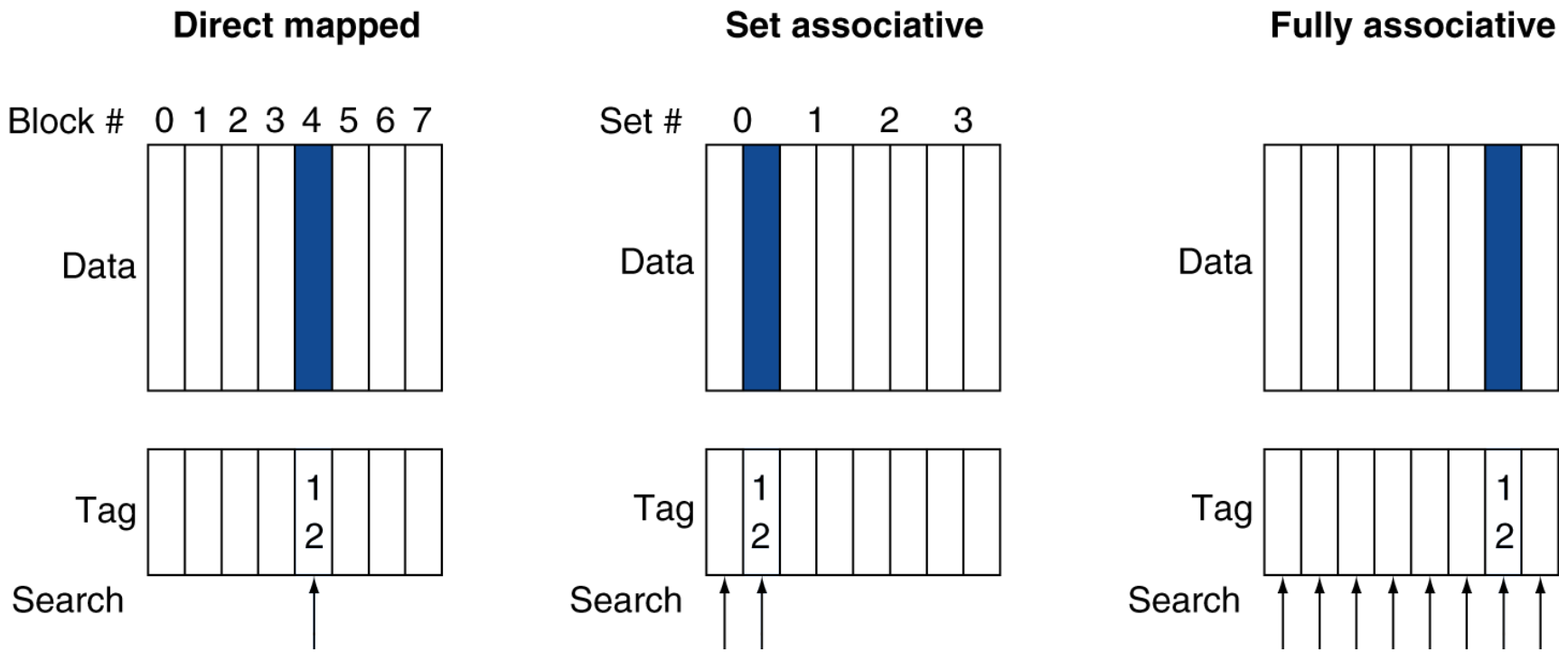


- #Blocks is a power of 2
- Use low-order address bits

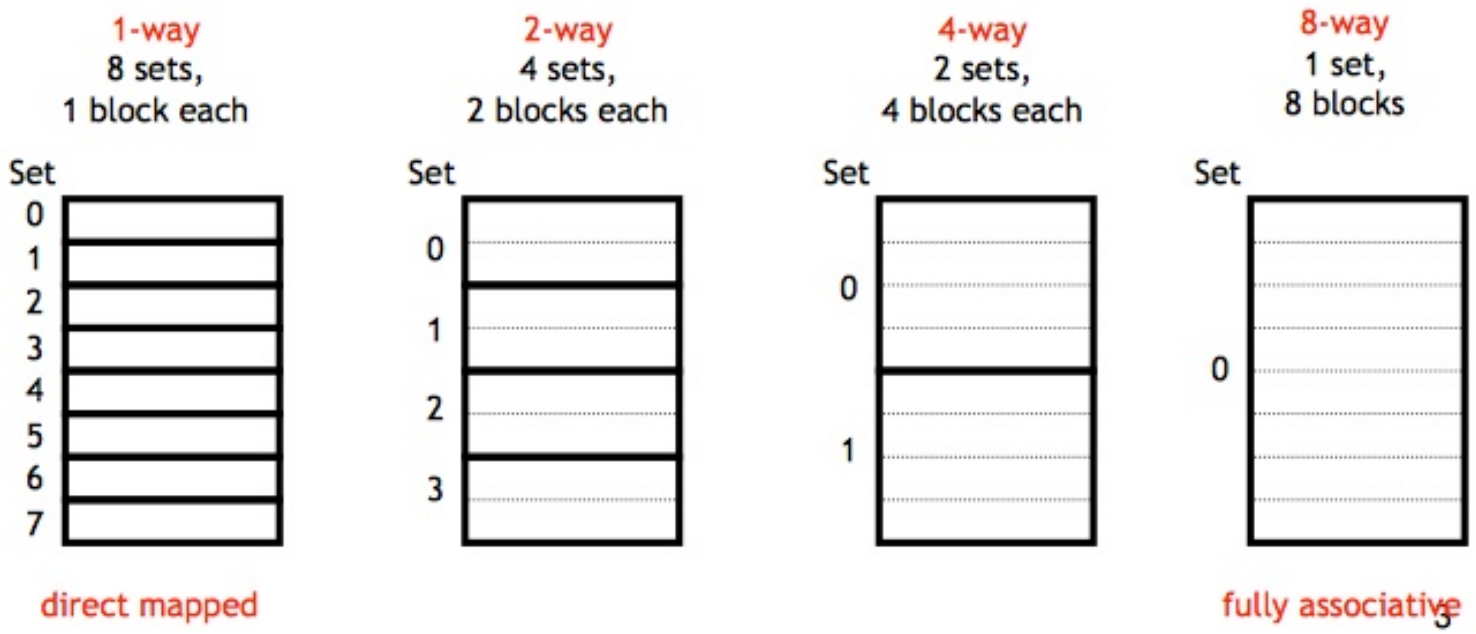
Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Associative Cache Example



Associative Cache Example

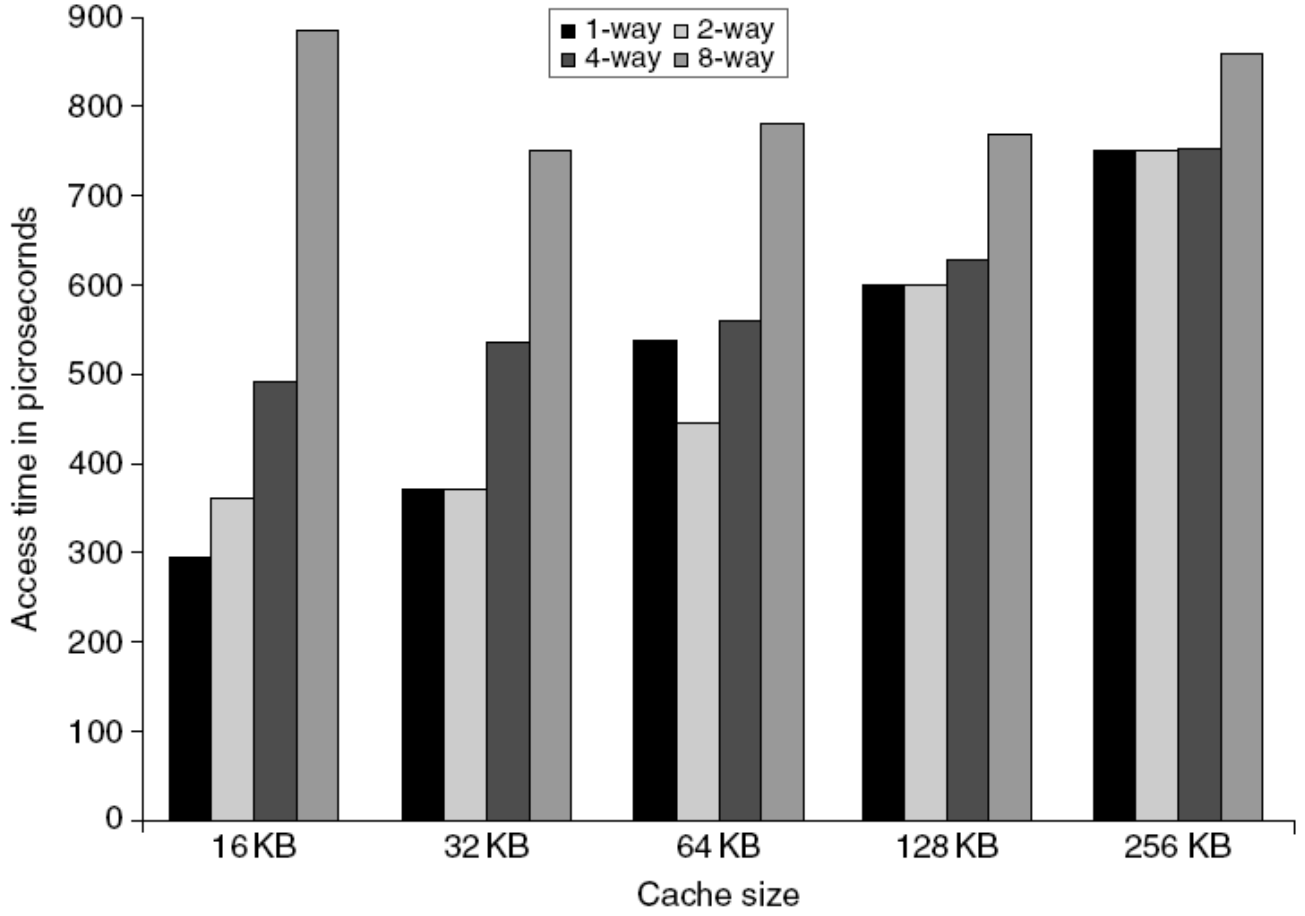


How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

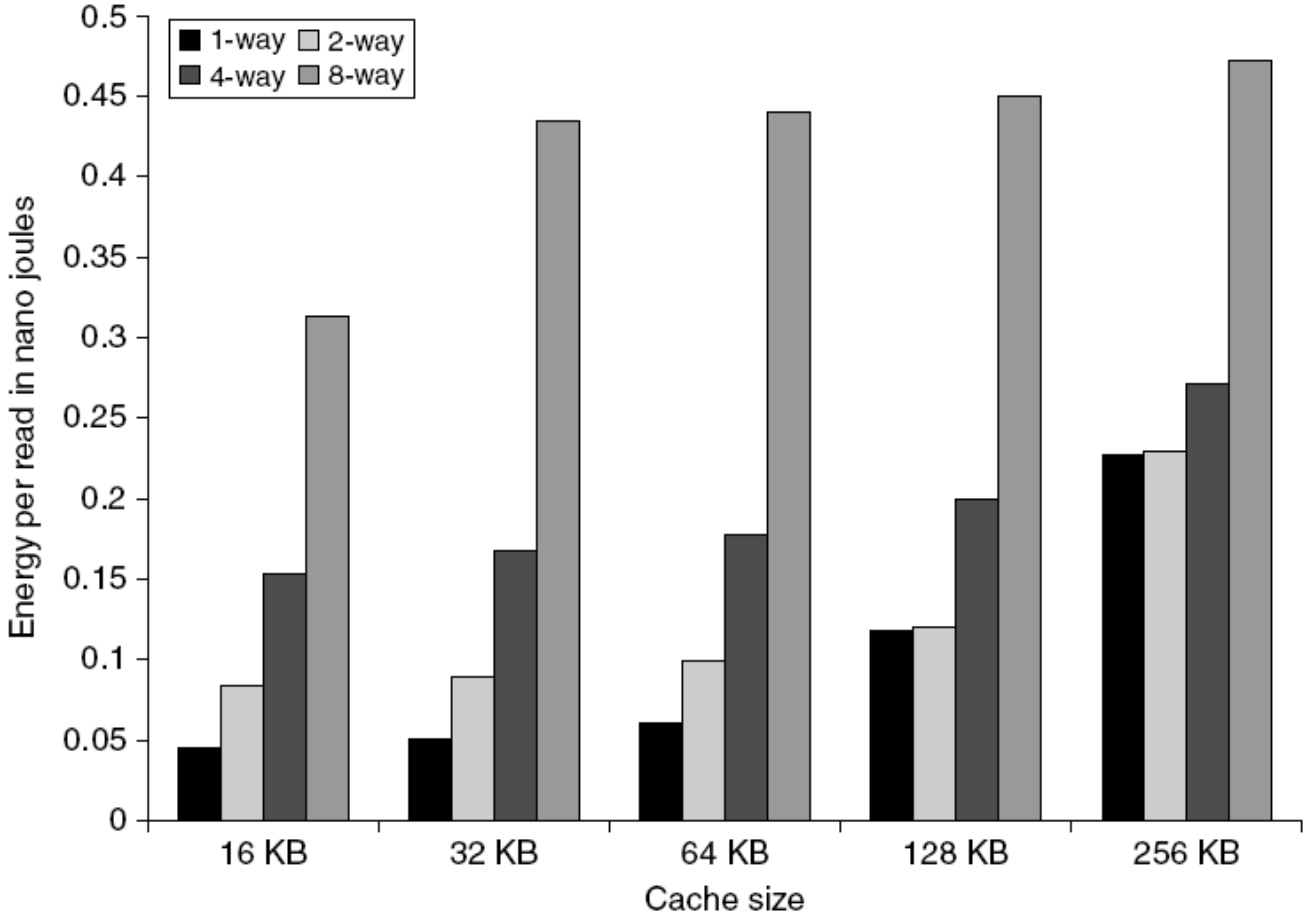
2-way set associative = 2 blocks in set

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

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

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

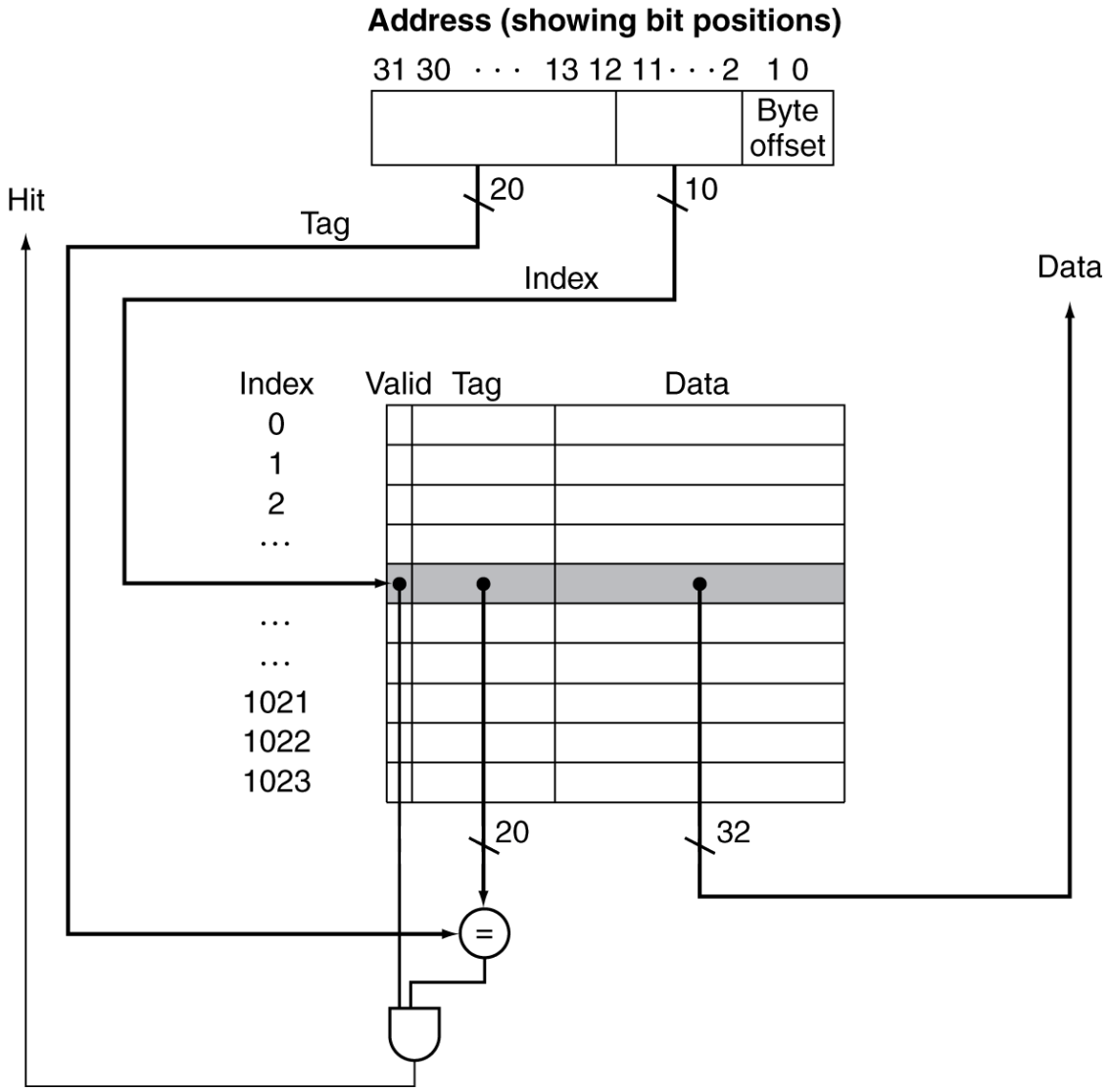
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

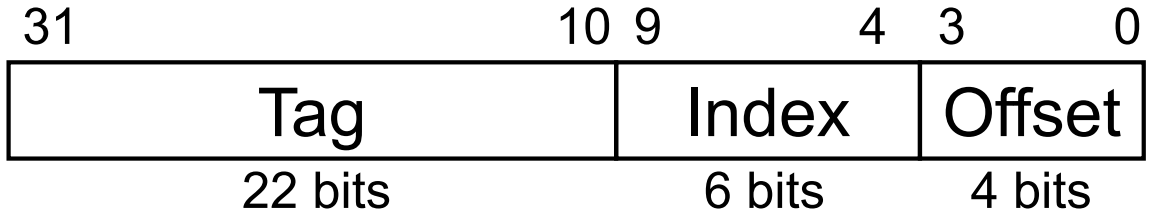
Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Address Subdivision



Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$
- 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 $\Rightarrow$ fewer of them
    - More competition $\Rightarrow$ increased miss rate
  - Larger blocks $\Rightarrow$ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

Memory Hierarchy Basics

- Writing to cache: two strategies
  - *Write-through*
    - Immediately update lower levels of hierarchy
  - *Write-back*
    - Only update lower levels of hierarchy when an updated block is replaced
  - Both strategies use *write buffer* to make writes asynchronous

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But 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
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back

- Alternative: 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

Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don't fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Cache Misses

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

Memory Hierarchy Basics

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

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

- Note that speculative and multithreaded processors may execute other instructions during a miss
  - Reduces performance impact of misses

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

Way Prediction

- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - “Way selection”
  - Increases mis-prediction penalty

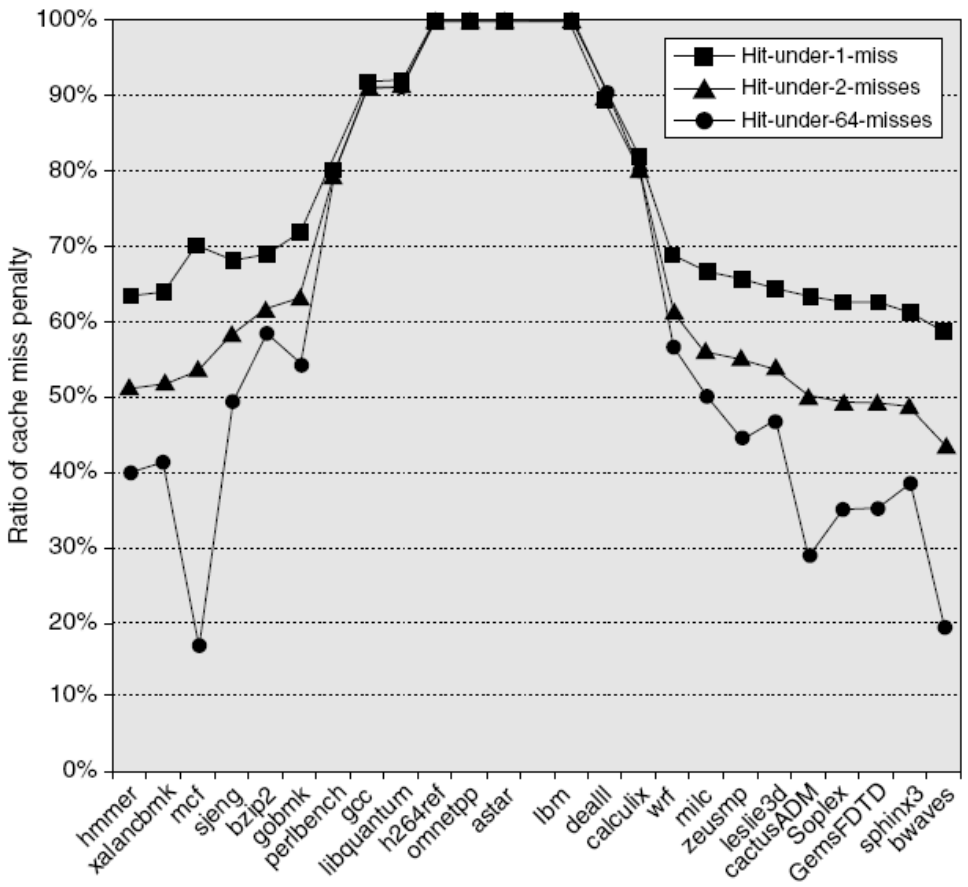
Pipelining Cache

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles

- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

Nonblocking Caches

- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



Multibanked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2

- Interleave banks according to block address

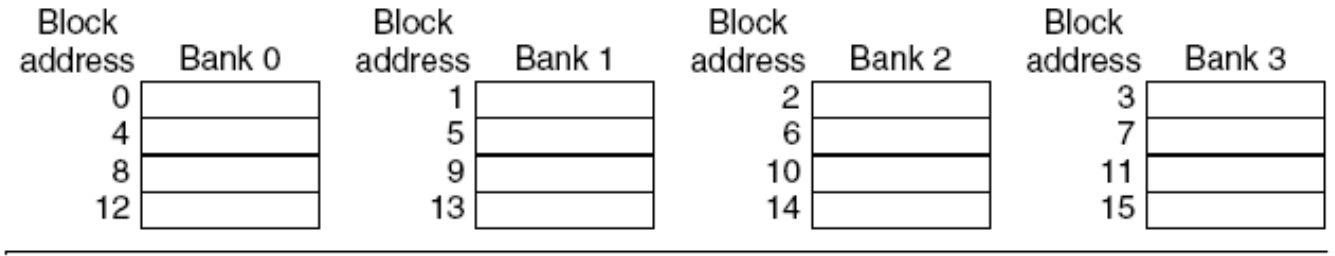


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

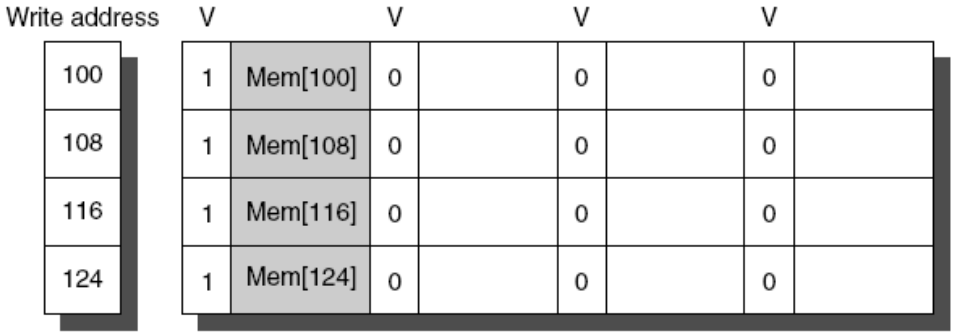
Critical Word First, Early Restart

- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- Early restart
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives

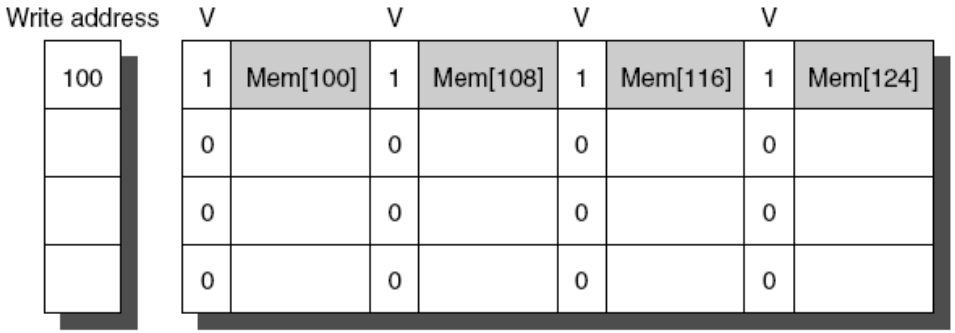
- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses



No write buffering



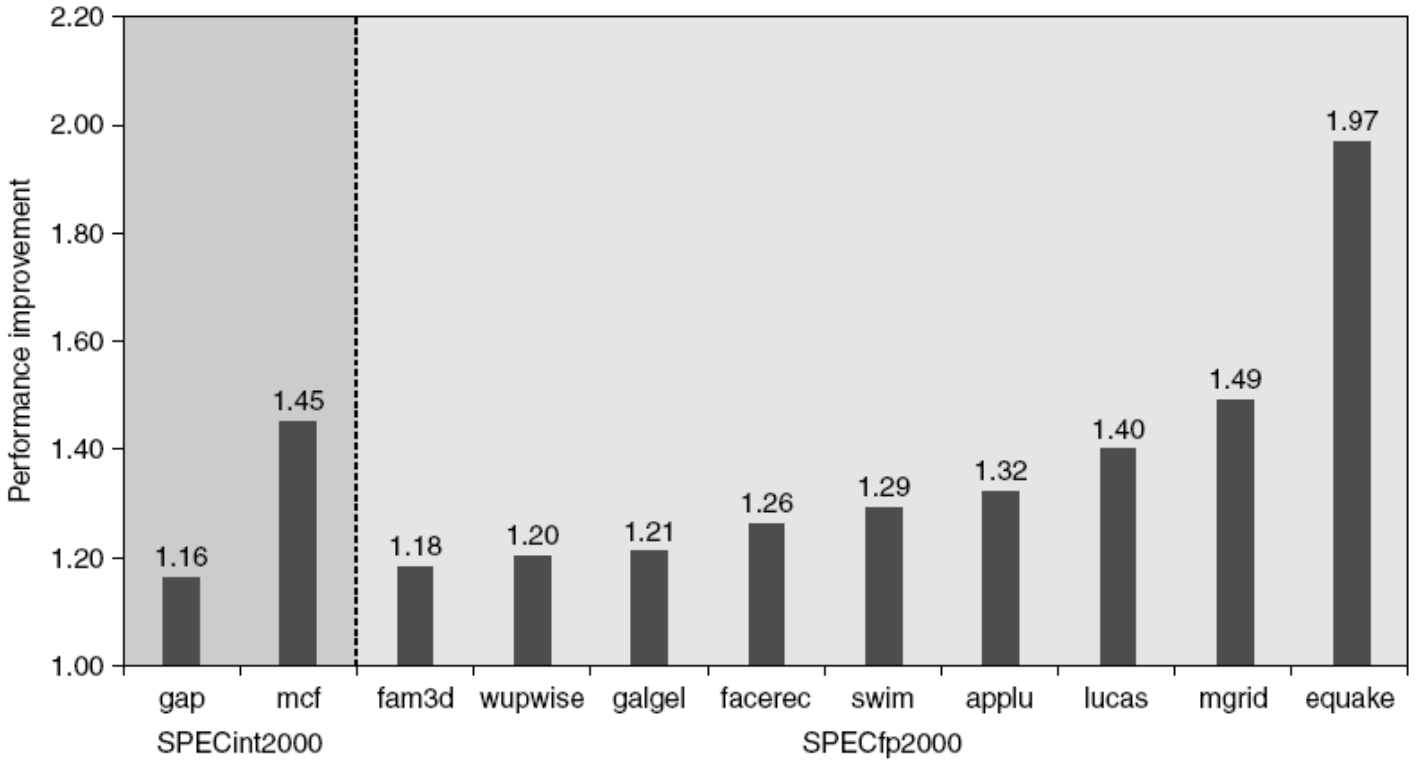
Write buffering

Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)



Pentium 4 Pre-fetching

Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn't cause exceptions

- Register prefetch
  - Loads data into register
- Cache prefetch
  - Loads data into cache

- Combine with loop unrolling and software pipelining

Summary

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 cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both 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, 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

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Ten Advanced Optimizations

- Small and simple first level caches
  - Critical timing path:
    - addressing tag memory, then
    - comparing tags, then
    - selecting correct set
  - Direct-mapped caches can overlap tag compare and transmission of data
  - Lower associativity reduces power because fewer cache lines are accessed

Flash Memory

- Type of EEPROM
- Must be erased (in blocks) before being overwritten
- Non volatile
- Limited number of write cycles
- Cheaper than SDRAM, more expensive than disk
- Slower than SRAM, faster than disk

Memory Dependability

- Memory is susceptible to cosmic rays
- Soft errors: dynamic errors
  - Detected and fixed by error correcting codes (ECC)
- Hard errors: permanent errors
  - Use sparse rows to replace defective rows
- Chipkill: a RAID-like error recovery technique

Virtual Memory

- Protection via virtual memory
  - Keeps processes in their own memory space

- Role of architecture:
  - Provide user mode and supervisor mode
  - Protect certain aspects of CPU state
  - Provide mechanisms for switching between user mode and supervisor mode
  - Provide mechanisms to limit memory accesses
  - Provide TLB to translate addresses

Virtual Machines

- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable
- Allows different ISAs and operating systems to be presented to user programs
  - "System Virtual Machines"
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs"

Impact of VMs on Virtual Memory

- Each guest OS maintains its own set of page tables
  - VMM adds a level of memory between physical and virtual memory called “real memory”
  - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
    - Requires VMM to detect guest’s changes to its own page table
    - Occurs naturally if accessing the page table pointer is a privileged operation