Memory Hierarchy

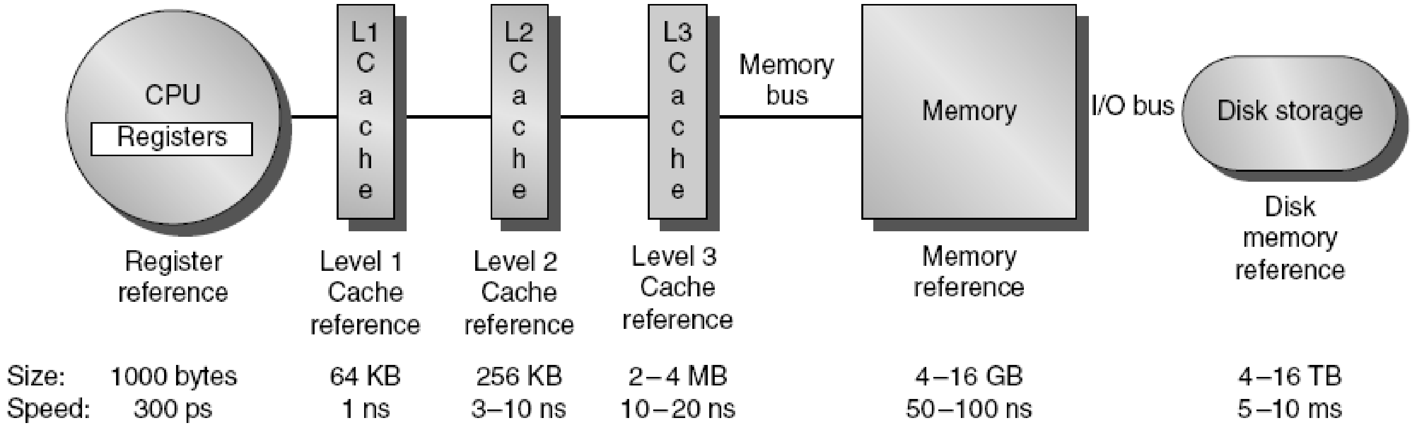
Memory Hierarchy Design

- Memory hierarchy
- Advanced optimizations
- Memory technology
- Virtual memory
- Address translations

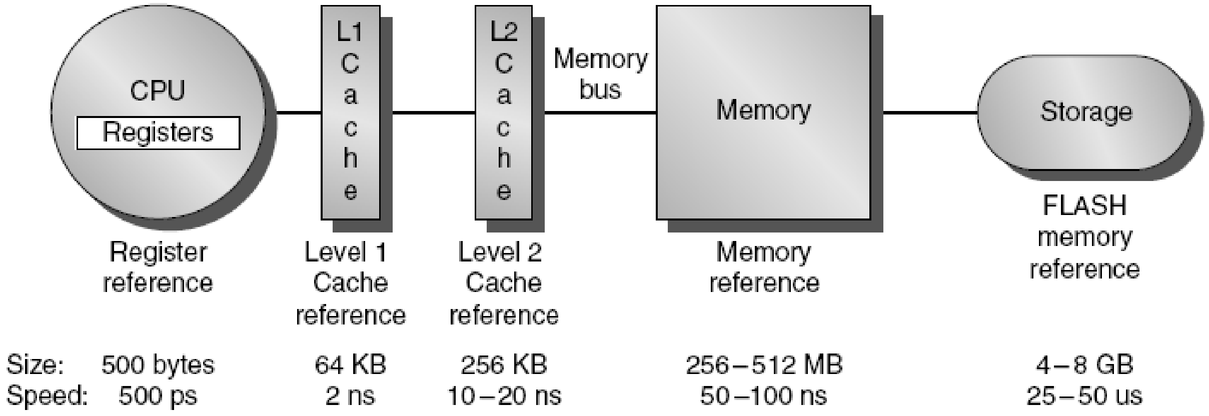
Introduction

- Programmers want unlimited amounts of memory with low latency.
- Fast memory technology is more expensive per bit than slower memory.
- Solution: 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 ensures that nearly all references can be found in smaller memories.
  - Gives the illusion of a large, fast memory being presented to the processor

Memory Hierarchy

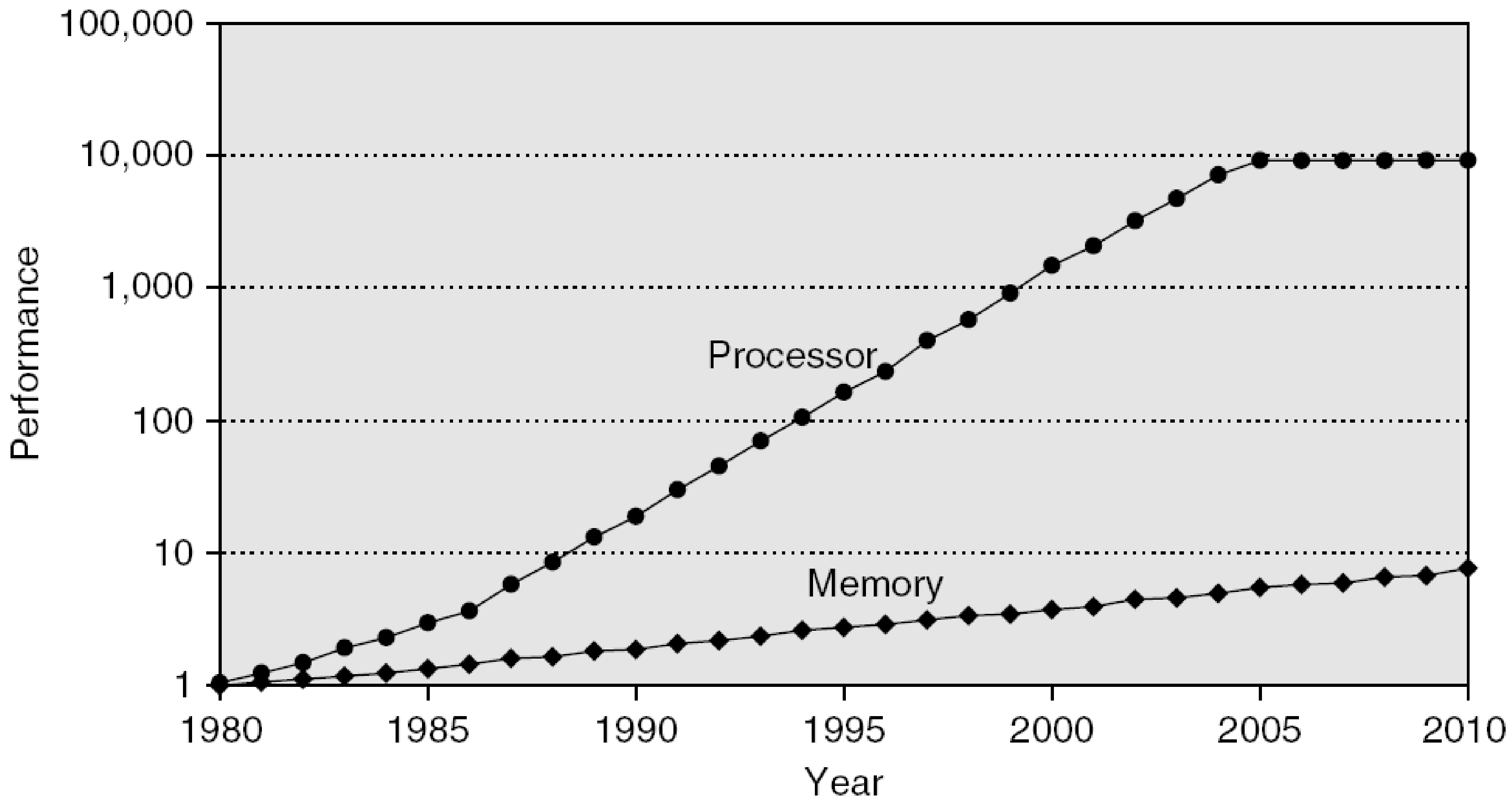


(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Memory Performance Gap



Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multicore processors.
  - Aggregate peak bandwidth grows with # cores.
    - Intel Core i7 can generate two references per core per clock.
    - Four cores and 3.2 GHz clock.
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references
      - = 409.6 GB/s!
    - DRAM bandwidth is only 6 percent of this (25 GB/s).
    - Requires:
      - Multiport, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

Performance and Power

- High-end microprocessors have >10 MB on-chip cache.
  - Consumes large amount of area and power budget

ENGINEERING@SYRACUSE

Cache Memory

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 (cont.)

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

- 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.

Memory Hierarchy Basics (cont.)

- 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.

Memory Hierarchy Basics (cont.)

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

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

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

Memory Hierarchy Basics (cont.)

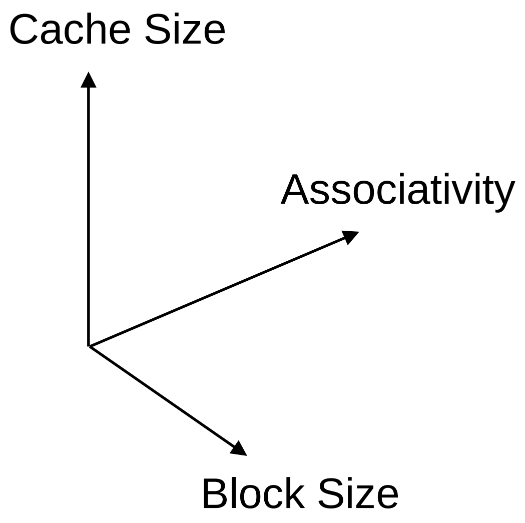
- Six basic cache optimizations.
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces 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

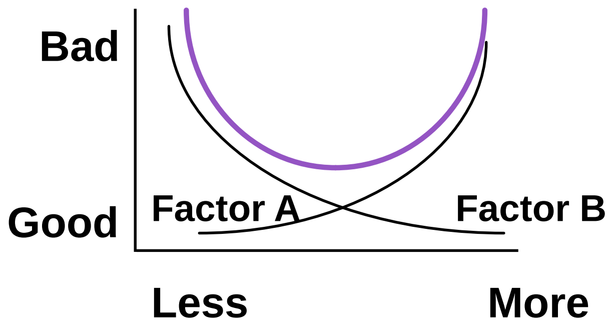
Cache Design Space

- The principle of locality
  - Program accesses a relatively small portion of the address space at any instant of time.
  - Temporal locality: locality in time
  - Spatial locality: locality in space
- Three major categories of cache misses
  - Compulsory misses: no escape
  - Capacity misses: increase cache size
  - Conflict misses: increase cache size and/or associativity.
- Write policy:
  - Write through vs. write back
- CPU time: function of (ops, cache misses) vs. just (ops)
  - Affects compilers, data structures, and algorithms

Cache Design Space (cont.)

- Several interacting dimensions:
  - Cache size
  - Block size
  - Associativity
  - Replacement policy
  - Write-through vs. write-back
  - Write allocation
- The optimal choice is a compromise.
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache, TLB)
  - Depends on technology/cost
- Simplicity often wins.





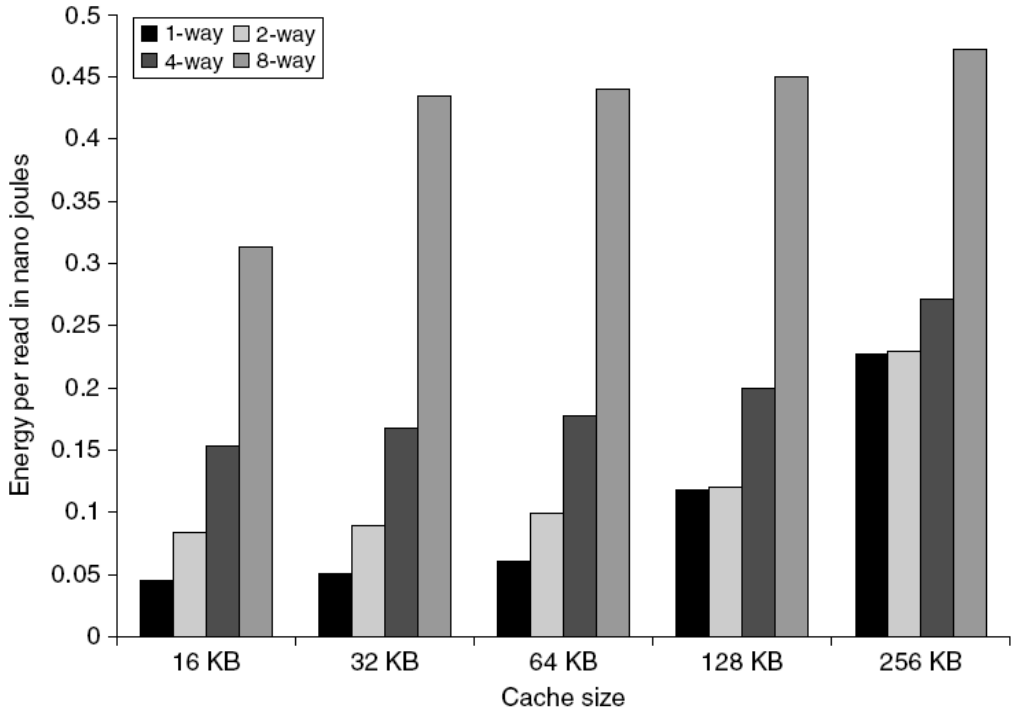
ENGINEERING@SYRACUSE

ENGINEERING@SYRACUSE

Advanced Optimizations

Small and Simple First-Level Cache

- Critical timing path:
  - Addressing tag → comparing tags → selecting correct set
- Direct mapped → overlap tag compare and transmission of data
- Lower associativity → fewer cache lines are accessed, lower power



ENGINEERING@SYRACUSE

Way Prediction

- Predict the way to preset mux → better hit time.
  - Misprediction gives longer hit time.
  - Prediction accuracy:
    - > 90 percent for two-way
    - > 80 percent for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-1990s.
  - Used on ARM Cortex-A8.
- Extend to predict block as well.
  - “Way selection”
  - Increases misprediction penalty

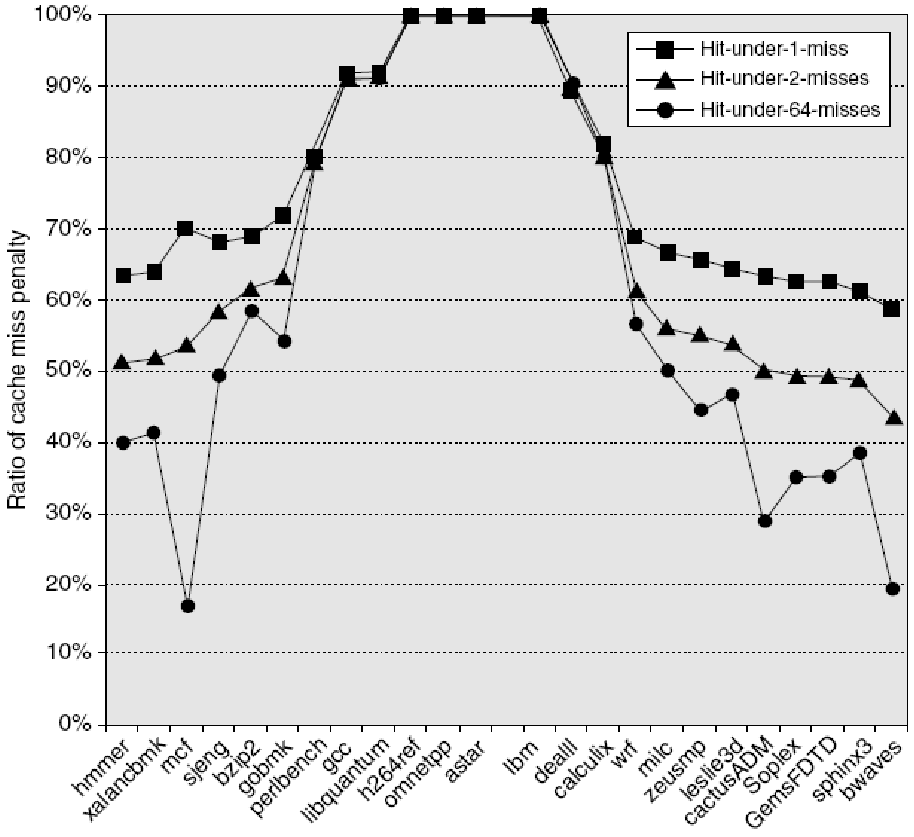
Pipelining Cache

- Pipeline cache access → better bandwidth
  - Examples:
    - Pentium: one cycle
    - Pentium Pro–Pentium III: two cycles
    - Pentium 4–Core i7: four cycles

- Increases branch misprediction 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 one to four banks for L2.
  - Intel i7 supports four banks for L1 and eight banks for L2.
- Interleave banks according to block address.

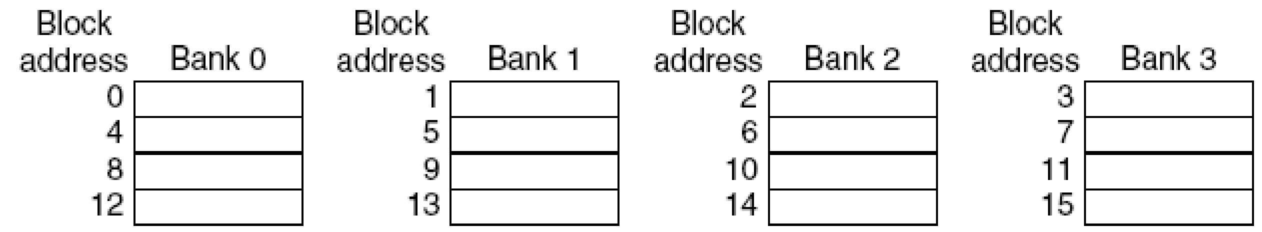


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

Critical Word First, Early Restart

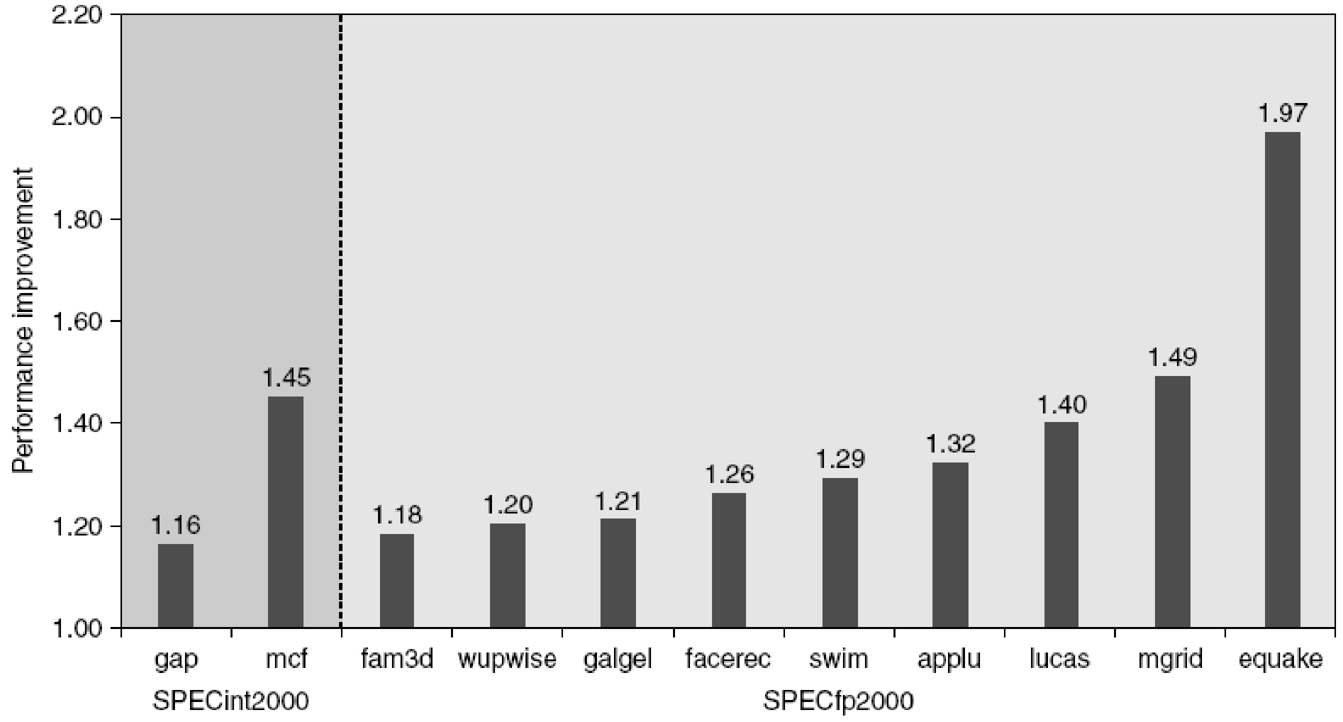
- Critical word first.
  - Request missed word from memory first.
  - Send it to the processor as soon as it arrives.
- Early restart.
  - Request words in normal order.
  - Send missed work to the processor as soon as it arrives.
- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched.

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 Prefetching

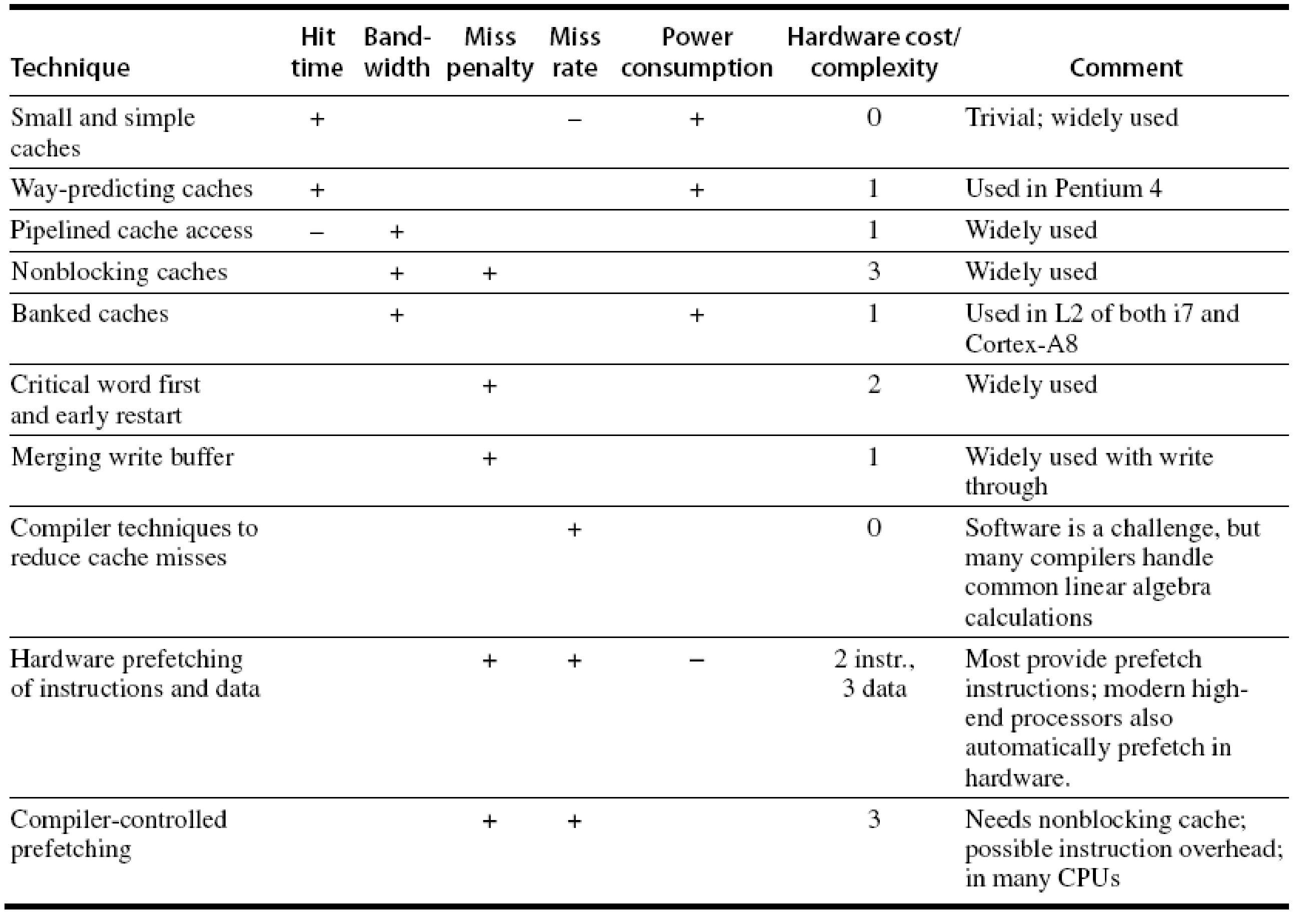
Compiler Prefetching

- Insert prefetch instructions before data are needed.
- Nonfaulting: 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.

ENGINEERING@SYRACUSE

Memory Technology

Memory Technology

- Performance metrics
  - Latency is concern of cache.
  - Bandwidth is concern of multiprocessors and I/O.
  - Access time:
    - Time between read request and when desired word arrives
  - Cycle time:
    - Minimum time between unrelated requests to memory
- DRAM used for main memory, SRAM used for cache.

Memory Technology (cont.)

- SRAM
  - Requires low power to retain bit
  - Requires six transistors/bit
- DRAM
  - Must be rewritten after being read
  - Must also be periodically refreshed
    - Every ~ 8 ms.
    - Each row can be refreshed simultaneously.
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

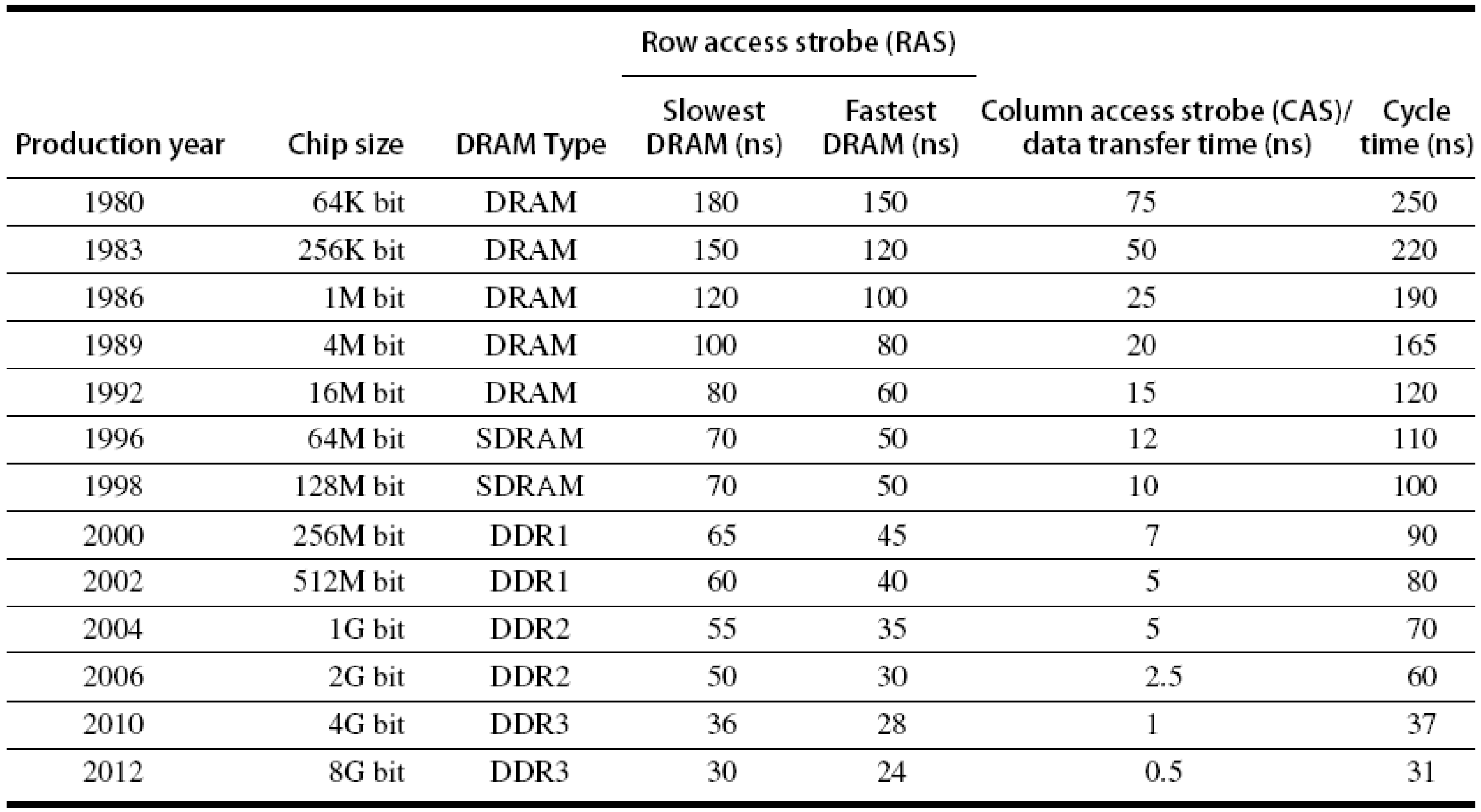
Memory Technology (cont.)

- Amdahl:
  - Memory capacity should grow linearly with processor speed.
  - Unfortunately, memory capacity and speed have 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

ENGINEERING@SYRACUSE

Main Memory

Memory Optimizations



			Row access strobe (RAS)			
Production year	Chip size	DRAM Type	Slowest DRAM (ns)	Fastest DRAM (ns)	Column access strobe (CAS)/ data transfer time (ns)	Cycle time (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 (cont.)

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 (cont.)

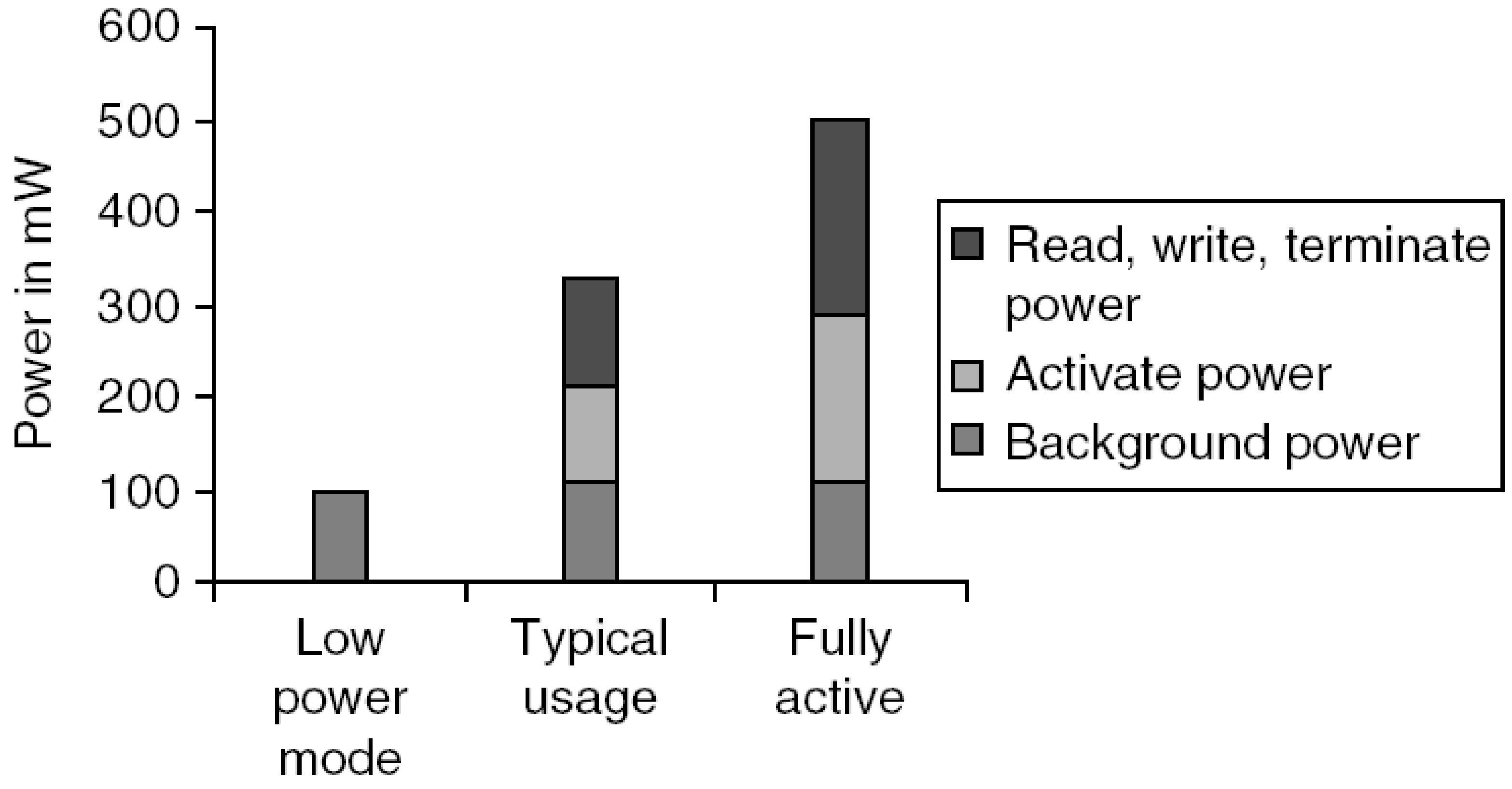
- DDR:
  - DDR2
    - Lower power (2.5 V → 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz
  - DDR5
    - Specs by December 2016, and deployed in 2020

- GDDR5 is graphics memory based on DDR3/

Memory Optimizations (cont.)

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

Memory Power Consumption



ENGINEERING@SYRACUSE

Flash Memory

- Type of EEPROM
- Must be erased (in blocks) before being overwritten
- Nonvolatile
- 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.

ENGINEERING@SYRACUSE

Virtual Memory

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

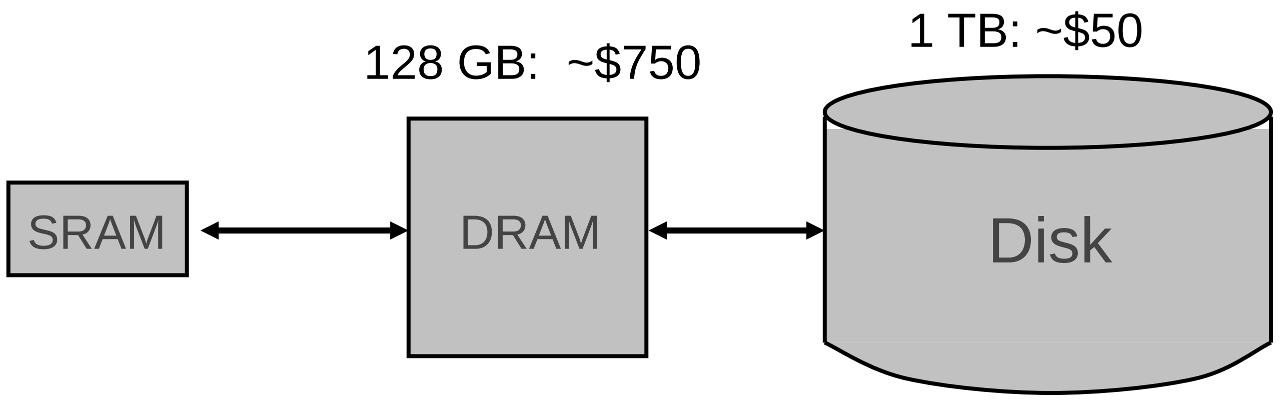
- Each guest OS maintains its own set of page tables.
  - VMM (VM monitor) adds a level of memory between physical and virtual memory called “real memory.”
  - VMM maintains a shadow page table that maps guest virtual addresses to physical addresses.
    - Requires VMM to detect guest’s changes to its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

Reasons for Virtual Memory

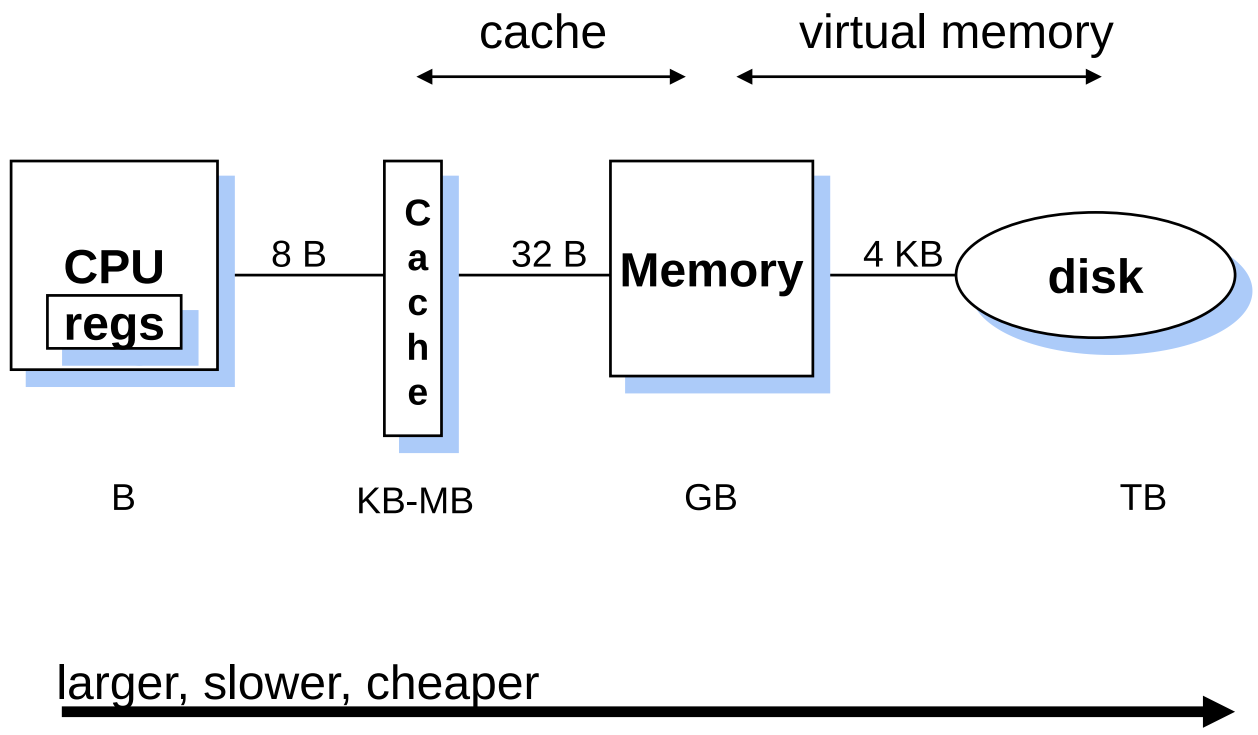
1. Use physical DRAM as a cache for the disk.
    - Address space of a process can exceed physical memory size.
    - Sum of address spaces of multiple processes can exceed physical memory.
2. Simplify memory management
    - Multiple processes resident in main memory.
        - Each process with its own address space
    - Only “active” code and data are actually in memory.
        - Allocate more memory to process as needed.
3. Provide protection
    - One process can’t interfere with another.
        - Because they operate in different address spaces
    - User process cannot access privileged information.
        - Different sections of address spaces have different permissions.

DRAM a “Cache” for Disk

- Full address space is quite large:
  - 32-bit addresses: ~4,000,000,000 (4 billion) bytes
  - 64-bit addresses: ~16,000,000,000,000,000,000 (16 quintillion) bytes
- Disk storage is ~100X cheaper than DRAM storage.
- 1 TB of DRAM: ~ $5,000
- 1 TB of disk: ~ $50
- To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk.



Levels in Memory Hierarchy



DRAM vs. SRAM as a “Cache”

- DRAM vs. disk is more extreme than SRAM vs. DRAM.
- Access latencies:
  - DRAM ~10X slower than SRAM
  - Disk ~100,000X slower than DRAM
- Importance of exploiting spatial locality:
  - First byte is ~100,000X slower than successive bytes on disk vs. ~4X improvement for page-mode vs. regular accesses to DRAM.
- Bottom line:
  - Design decisions made for DRAM caches driven by enormous cost of misses.

Impact of These Properties on Design

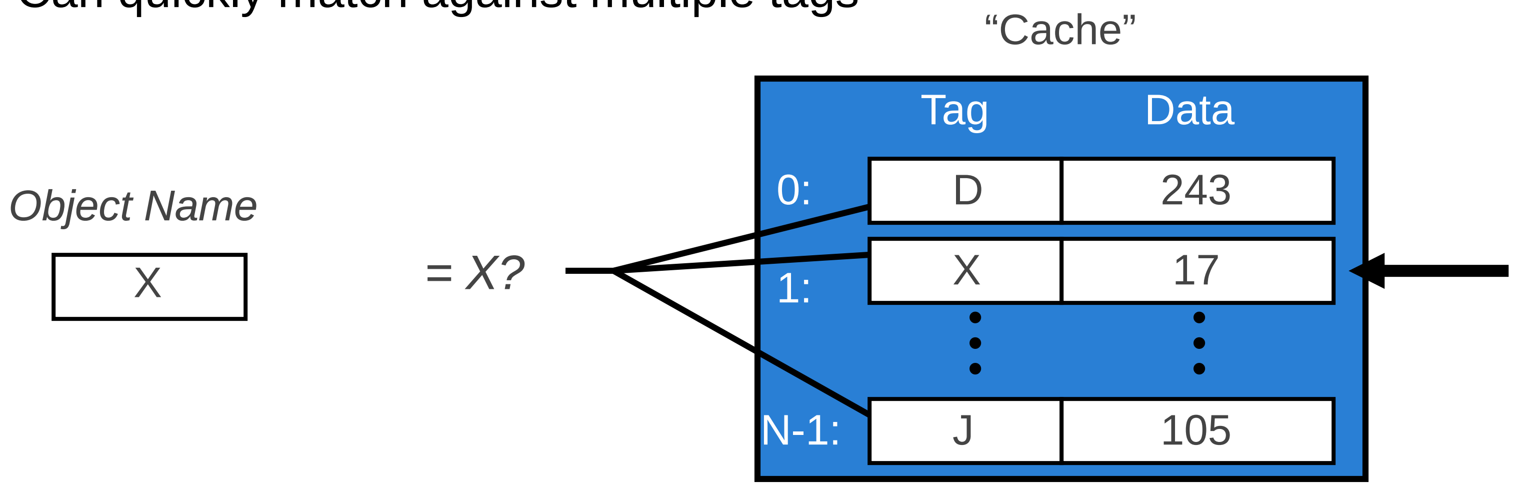
- If DRAM were to be organized similar to an SRAM cache, how would we set the following design parameters?
  - Line size? Large, because disk is better at transferring large blocks.
  - Associativity? High, to minimize miss rate.
  - Write through or write back? Write back, because can't afford to perform small writes to disk.
- What would the impact of these choices be on:
  - Miss rate: extremely low. << 1 percent
  - Hit time: must match cache/DRAM performance
  - Miss latency (penalty): very high. ~20 ms
  - Tag storage overhead: low, relative to block size

ENGINEERING@SYRACUSE

Locating an Object

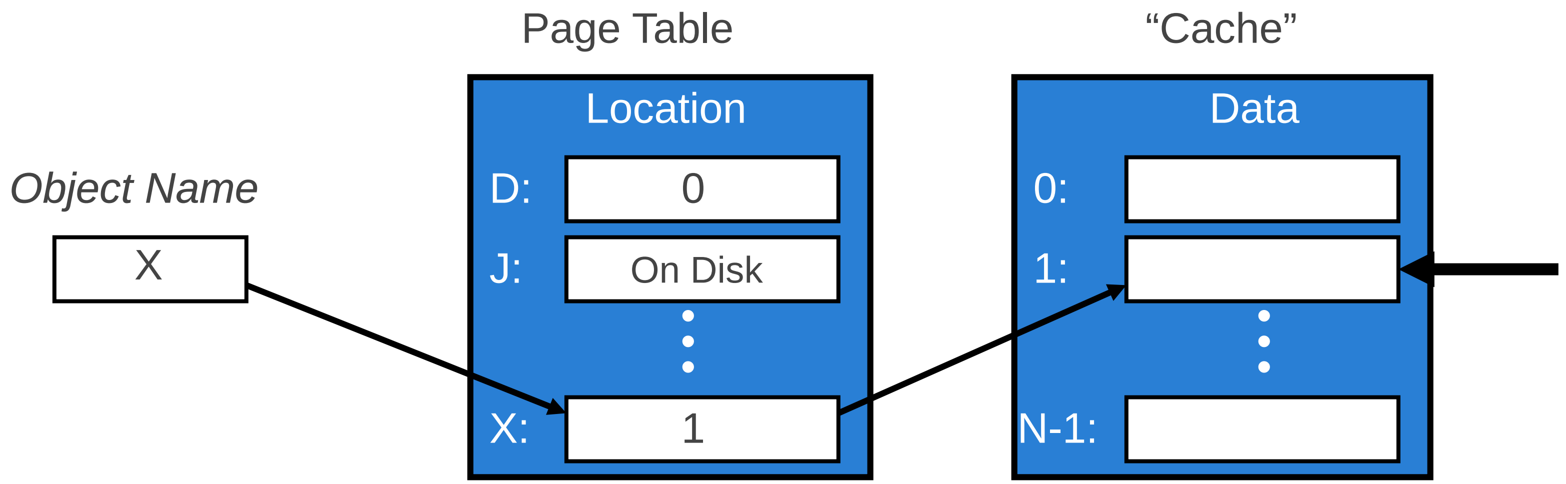
Locating an Object in a “Cache”

- SRAM cache
  - Tag stored with cache line.
  - Maps from cache block to memory blocks.
    - From cached to uncached form
  - No tag for block not in cache.
  - Hardware retrieves information.
    - Can quickly match against multiple tags



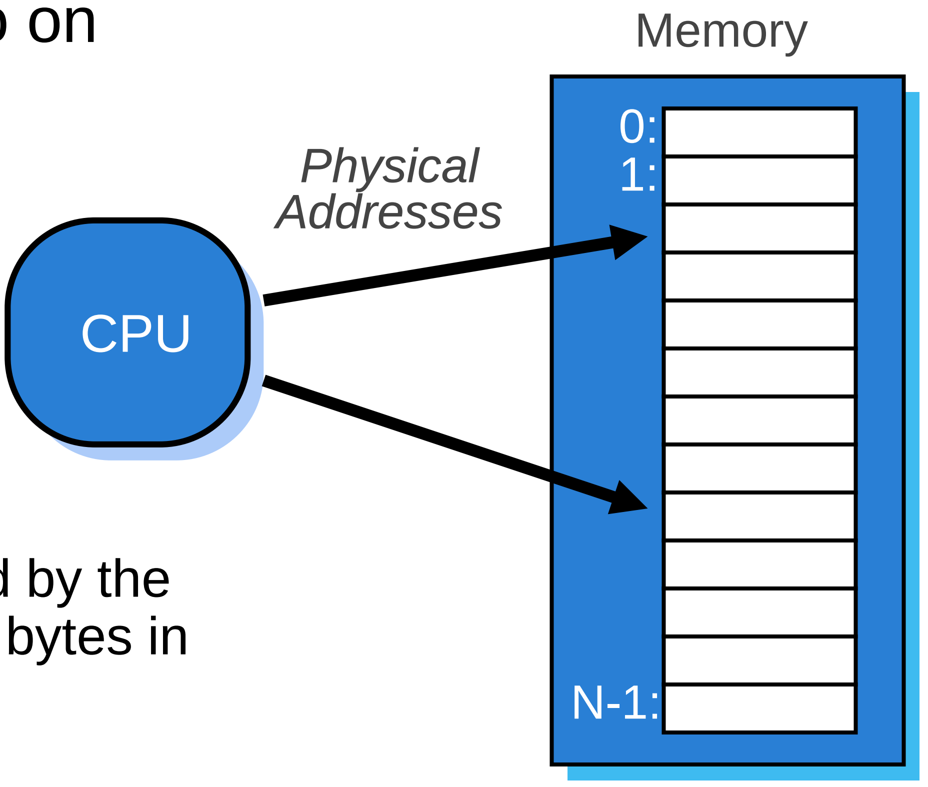
Locating an Object in a "Cache" (cont.)

- DRAM cache
  - Each allocated page of virtual memory has entry in *page table.*
  - Mapping from virtual pages to physical pages.
    - From uncached form to cached form
  - Page table entry even if page not in memory.
    - Specifies disk address
  - OS retrieves information.



Systems with Physical Memory Only

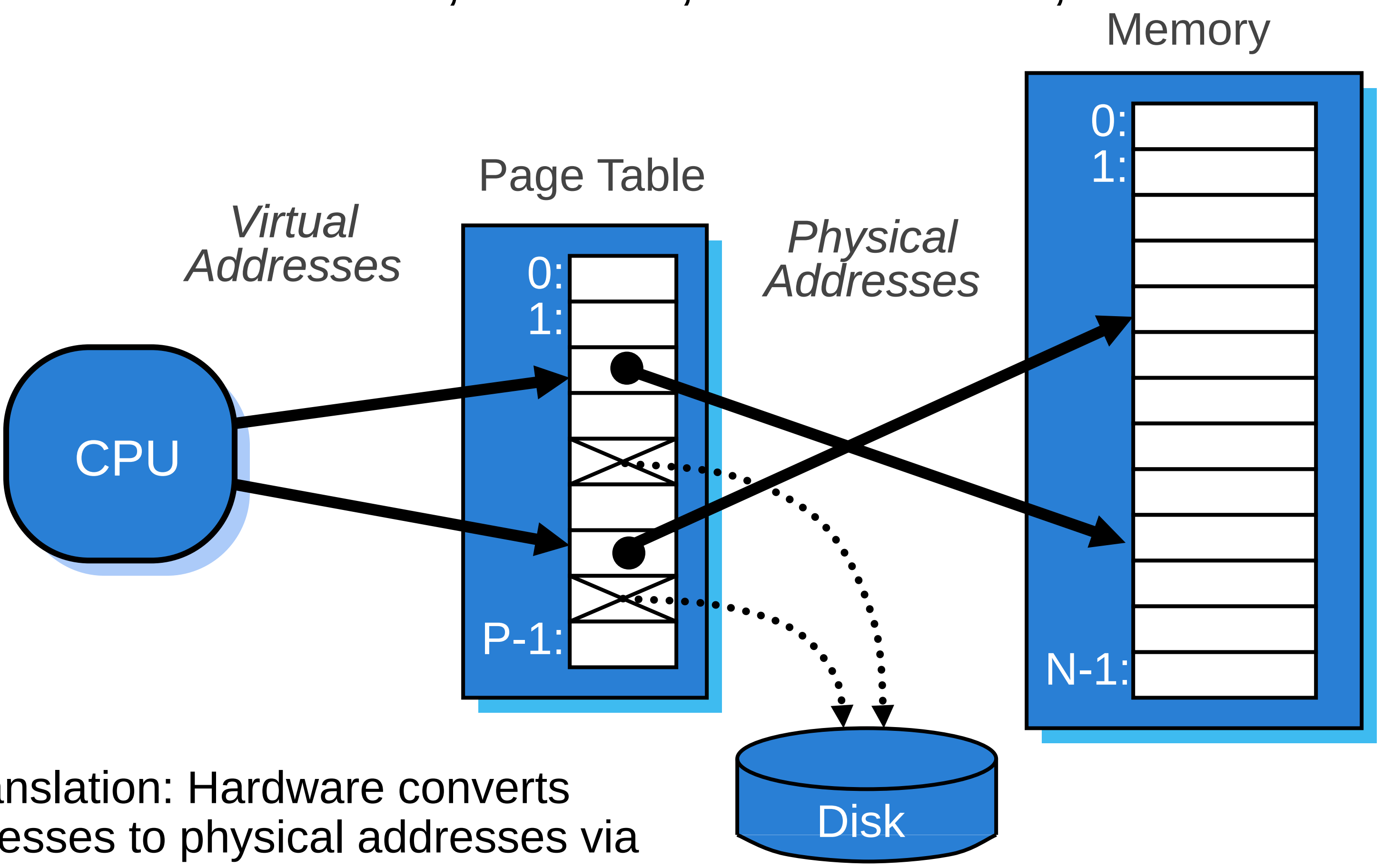
- Examples
  - Most Cray machines, early PCs, nearly all embedded systems, and so on



Addresses generated by the CPU point directly to bytes in physical memory.

A System with Virtual Memory

Examples: workstations, servers, modern PCs, and so on

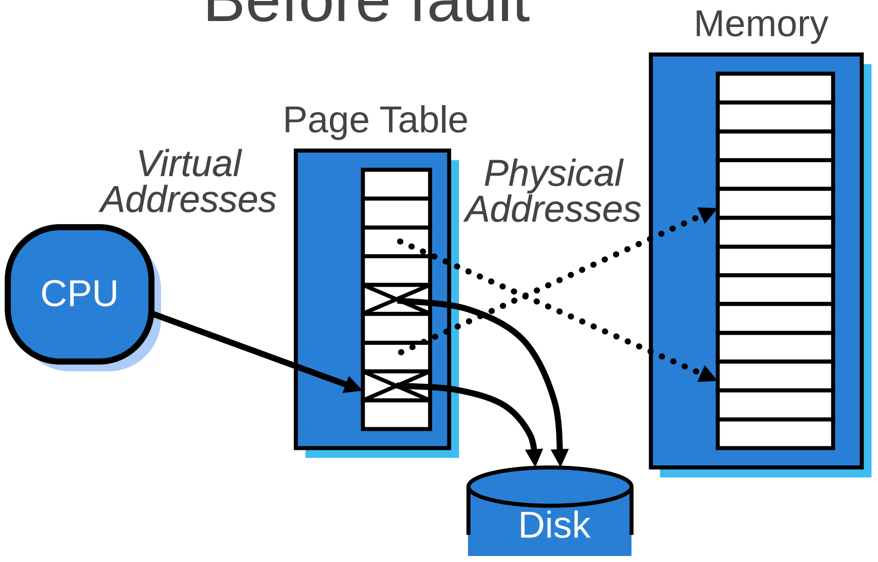


Address translation: Hardware converts virtual addresses to physical addresses via an OS-managed lookup table (page table).

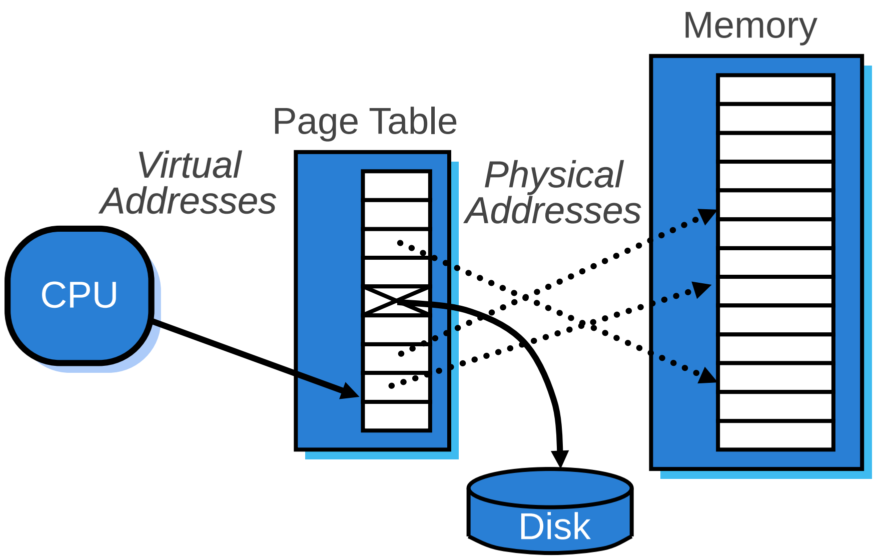
Page Fault (Similar to Cache Miss)

- What if an object is on disk rather than in memory?
  - Page table entry indicates virtual address not in memory.
  - OS exception handler invoked to move data from disk into memory.
    - Current process suspends; others can resume.
    - OS has full control over placement, and so on.

Before fault



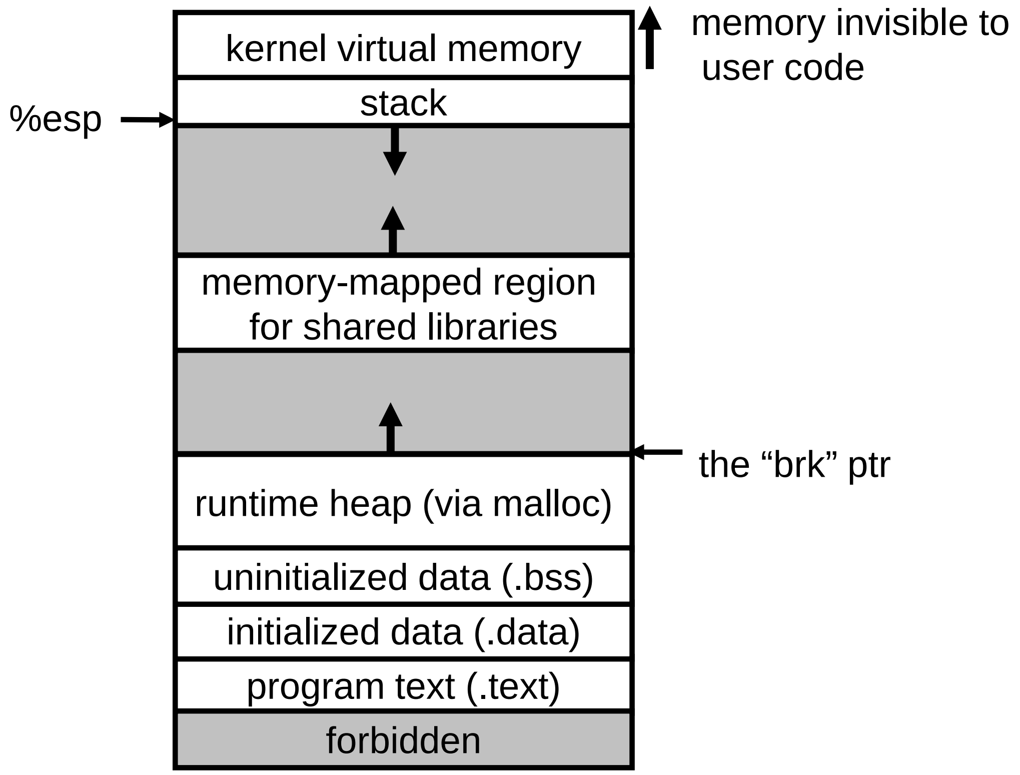
After fault



Memory Management

- Multiple processes can reside in physical memory.
- How do we resolve address conflicts?
  - What if two processes access something at the same address?

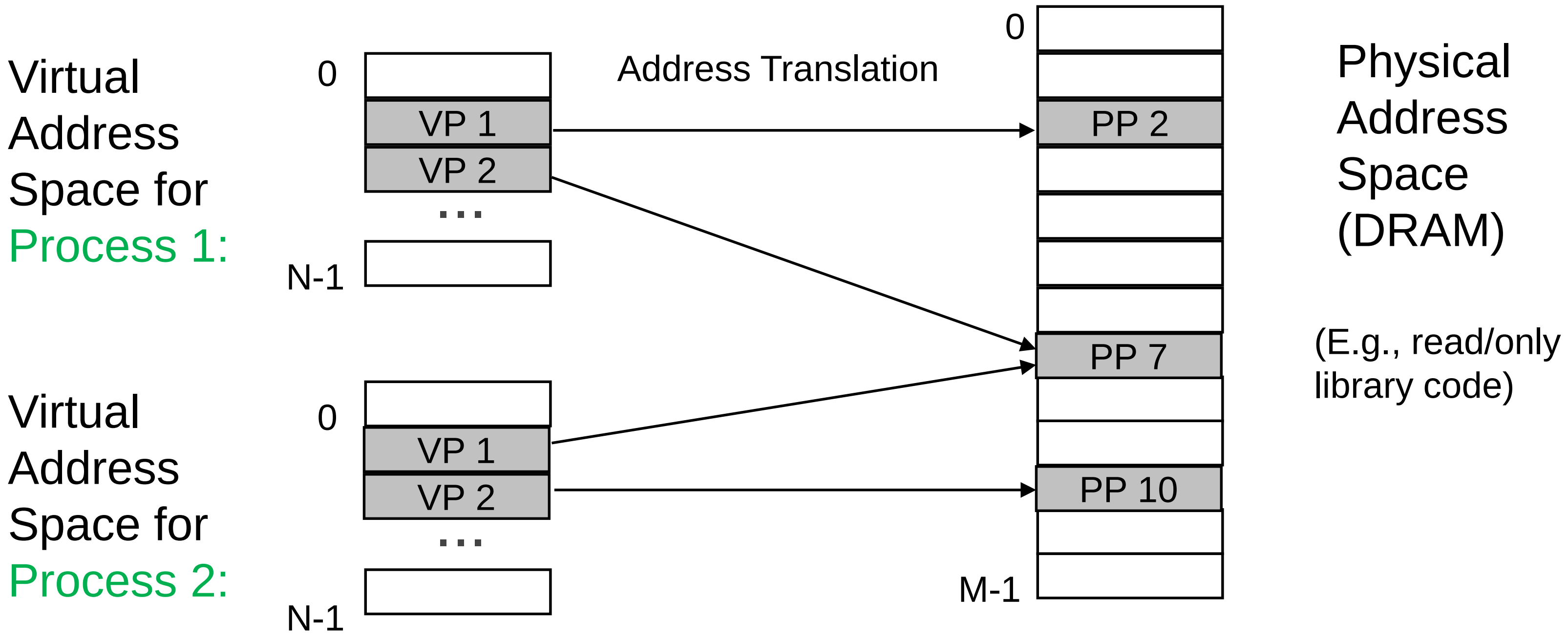
Linux/x86 process memory image



memory invisible to user code
kernel virtual memory
stack
%esp
memory-mapped region for shared libraries
the "brk" ptr
runtime heap (via malloc)
uninitialized data (.bss)
initialized data (.data)
program text (.text)
forbidden

Separate Virtual Address Spaces

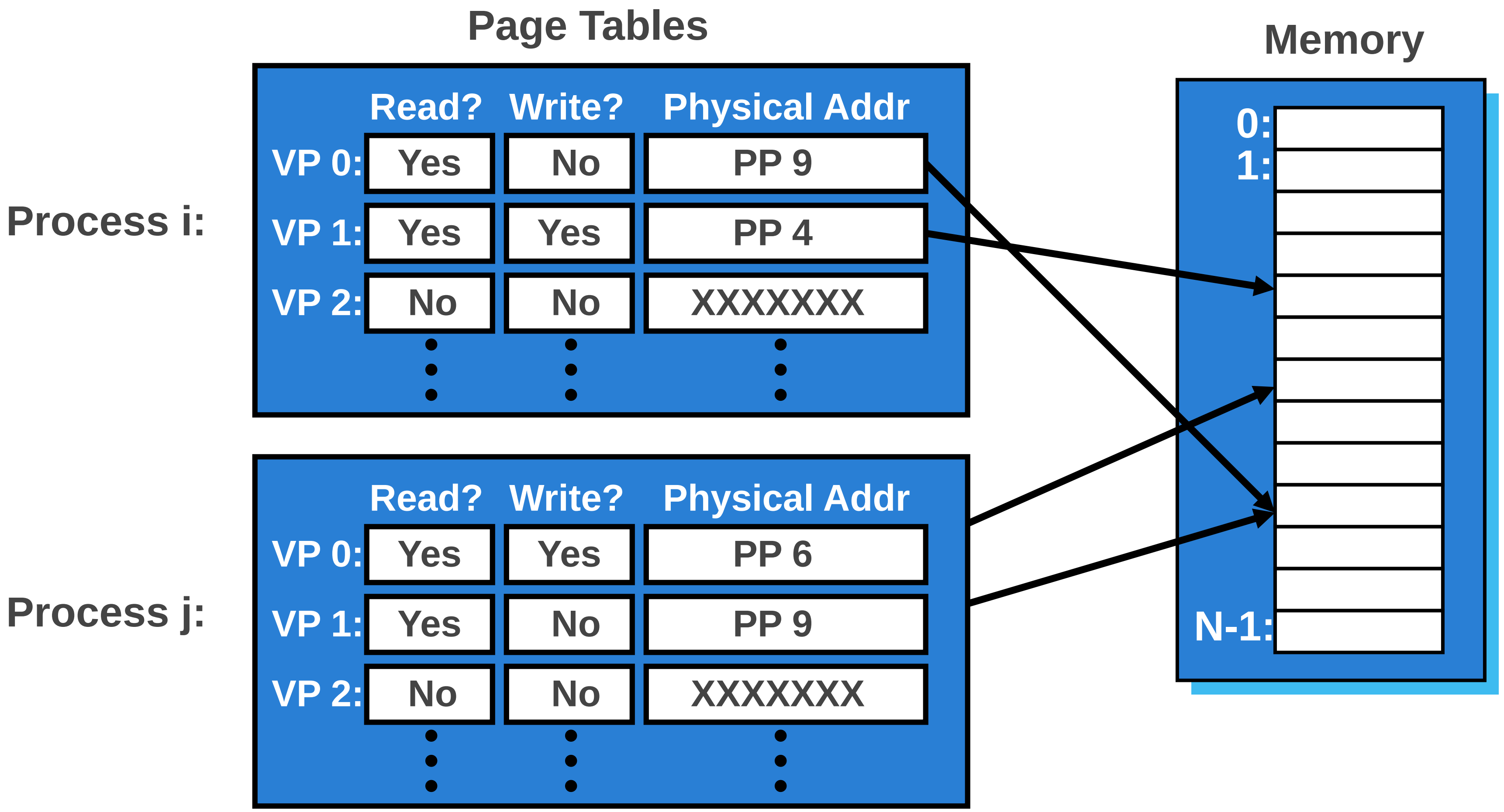
- Virtual and physical address spaces divided into equal-sized blocks.
  - Blocks are called “pages” (both virtual and physical).
- Each process has its own virtual address space.
  - Operating system controls how virtual pages as assigned to physical memory.
  - Every program can start at the same address (virtual address)!


Virtual Address Space for Process 1:
0
VP 1
VP 2
...
N-1
Address Translation
0
PP 2
PP 7
PP 10
M-1
Physical Address Space (DRAM)
(E.g., read/only library code)
Virtual Address Space for Process 2:
0
VP 1
VP 2
...
N-1

ENGINEERING@SYRACUSE

Protection

- Page table entry contains access rights information.
  - Hardware enforces this protection (trap into OS if violation occurs).



Process i:

	Read?	Write?	Physical Addr
VP 0:	Yes	No	PP 9
VP 1:	Yes	Yes	PP 4
VP 2:	No	No	XXXXXXX
	⋮	⋮	⋮

Process j:

	Read?	Write?	Physical Addr
VP 0:	Yes	Yes	PP 6
VP 1:	Yes	No	PP 9
VP 2:	No	No	XXXXXXX
	⋮	⋮	⋮

ENGINEERING@SYRACUSE

ENGINEERING@SYRACUSE

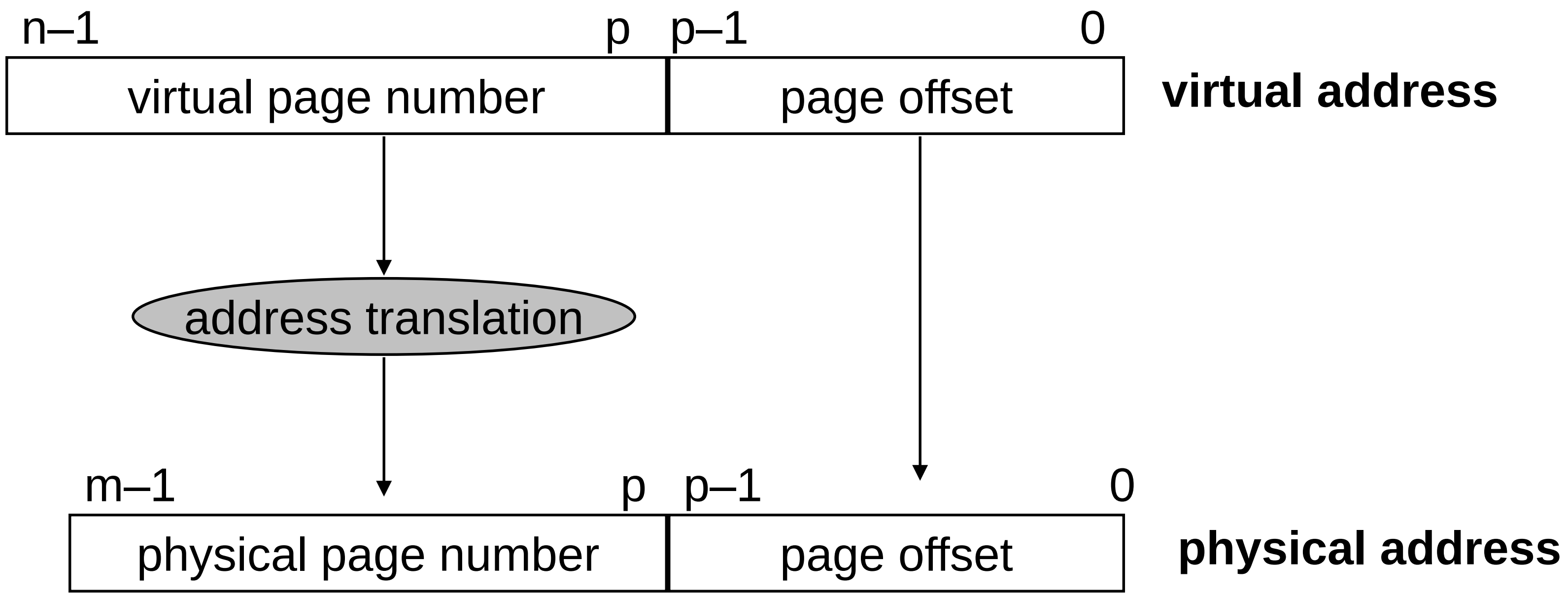
Address Translations

Address Translation

- Address translation must be fast.
  - It happens on every memory access.
- We need a fully associative placement policy.
  - Miss penalty is huge!
- We can’t afford to go looking at every virtual page to find the right one.
  - We don’t use the tag bits approach.

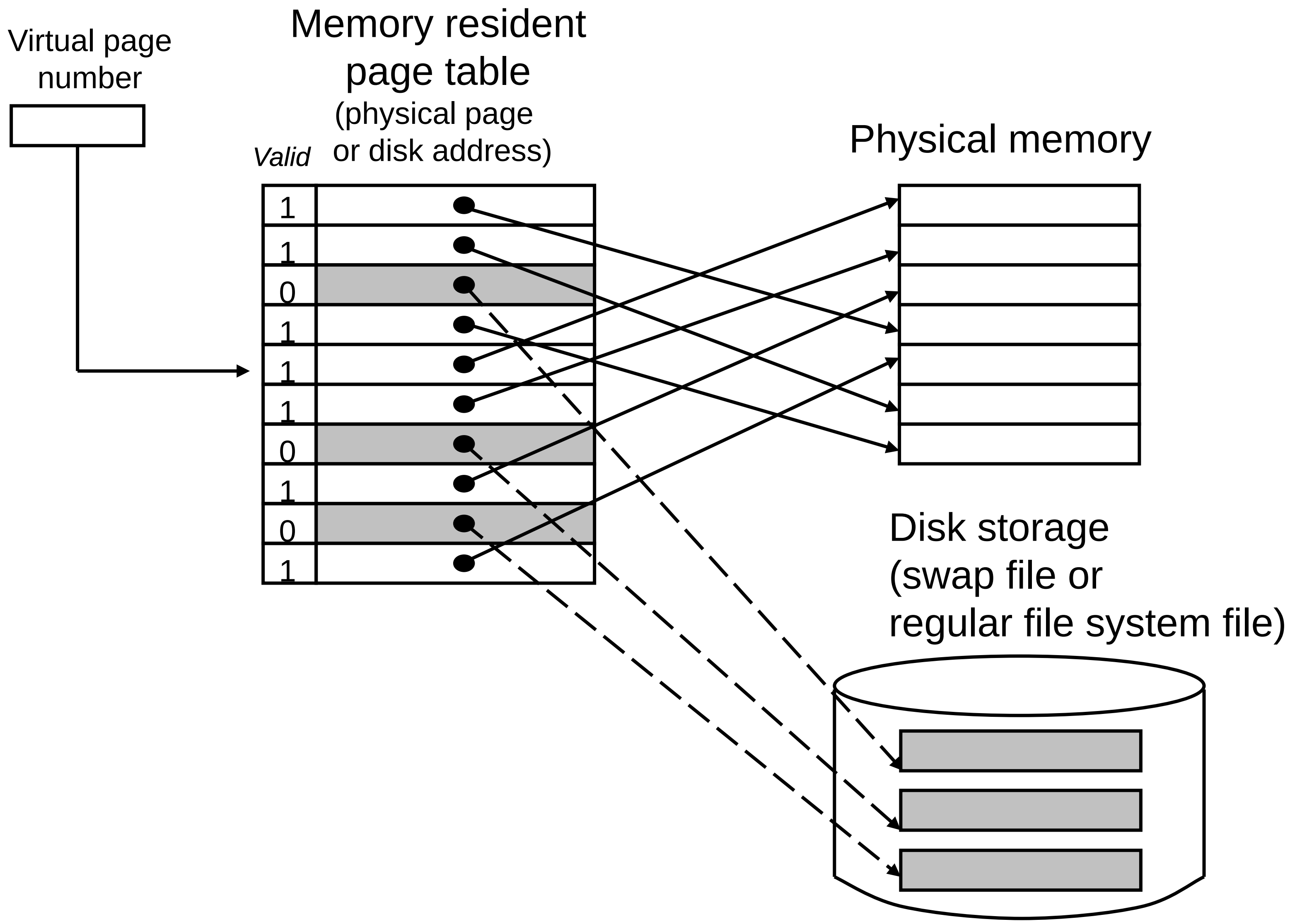
VM Address Translation

- Parameters
  - $P = 2^p$ = page size (bytes)
  - $N = 2^n$ = virtual address limit
  - $M = 2^m$ = physical address limit

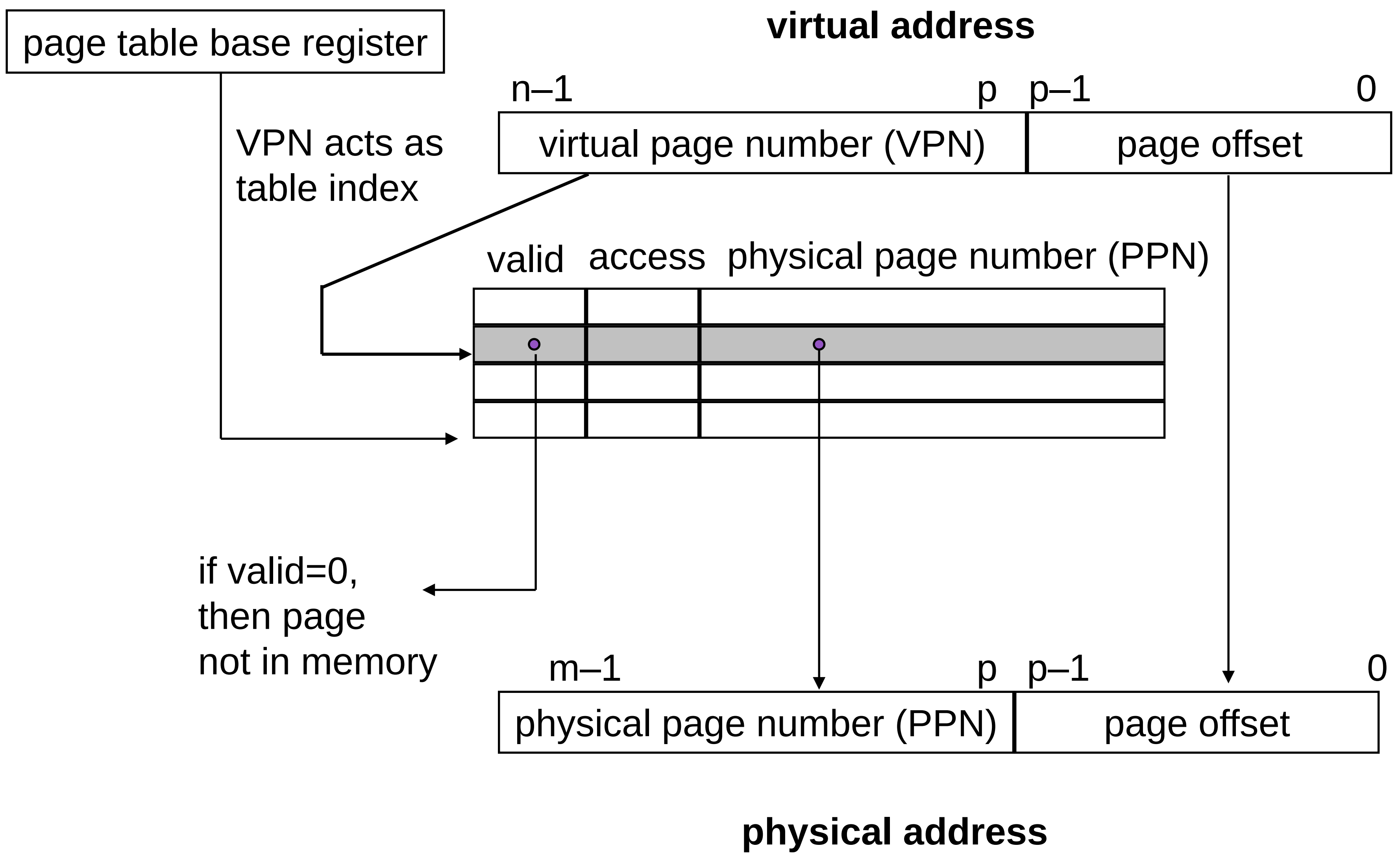


Notice that the page offset bits don't change as a result of translation.

Page Tables


Virtual page number
Memory resident page table (physical page or disk address)
Valid
1
1
0
1
1
1
0
1
0
1
Physical memory
Disk storage (swap file or regular file system file)

Address Translation via Page Table



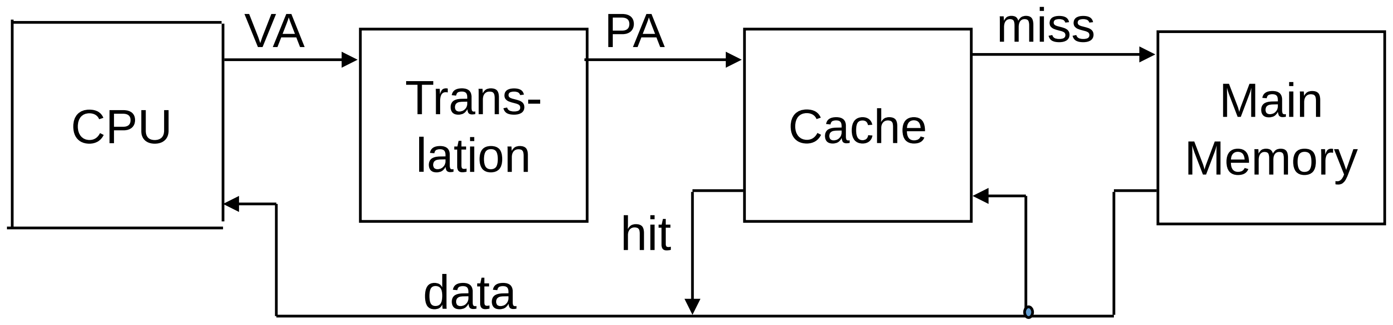
ENGINEERING@SYRACUSE

Page Table Operation

- Translation
  - Separate (set of) page table(s) per process.
  - VPN forms index into page table (points to a page table entry).
- Computing physical address
  - Page table entry (PTE) provides information about page.
  - if (valid bit = 1), then the page is in memory.
    - Use physical page number (PPN) to construct address.
  - if (valid bit = 0), then the page is on disk.
    - Page fault
    - Must load page from disk into main memory before continuing
- Checking protection
  - Access rights field indicate allowable access.
    - E.g., read-only, read-write, execute-only
    - Typically support multiple protection modes (e.g., kernel vs. user)
  - Protection violation fault if user doesn't have necessary permission.

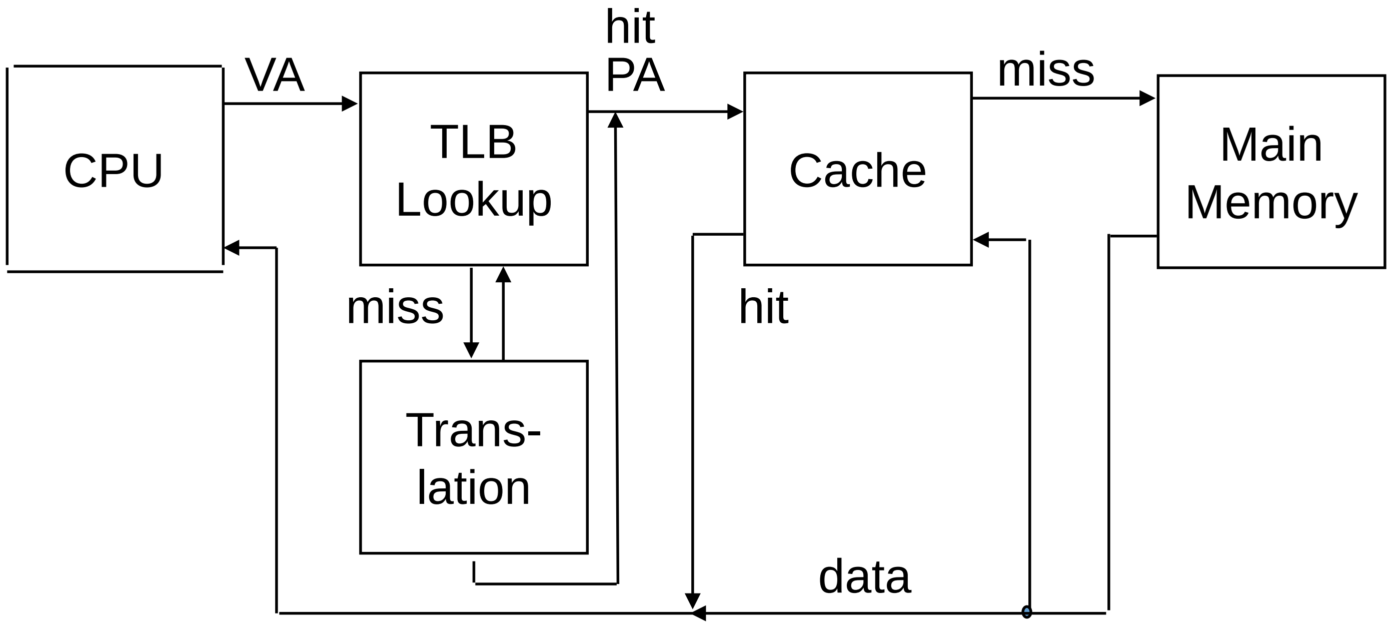
Integrating VM and Cache

- Most caches "physically addressed":
  - Accessed by physical addresses.
  - Allows multiple processes to have blocks in cache at same time.
  - Allows multiple processes to share pages.
  - Cache doesn't need to be concerned with protection issues.
    - Access rights checked as part of address translation.
- Perform address translation before cache lookup
  - But this could involve a memory access itself (of the PTE).
  - Of course, page table entries can also become cached.

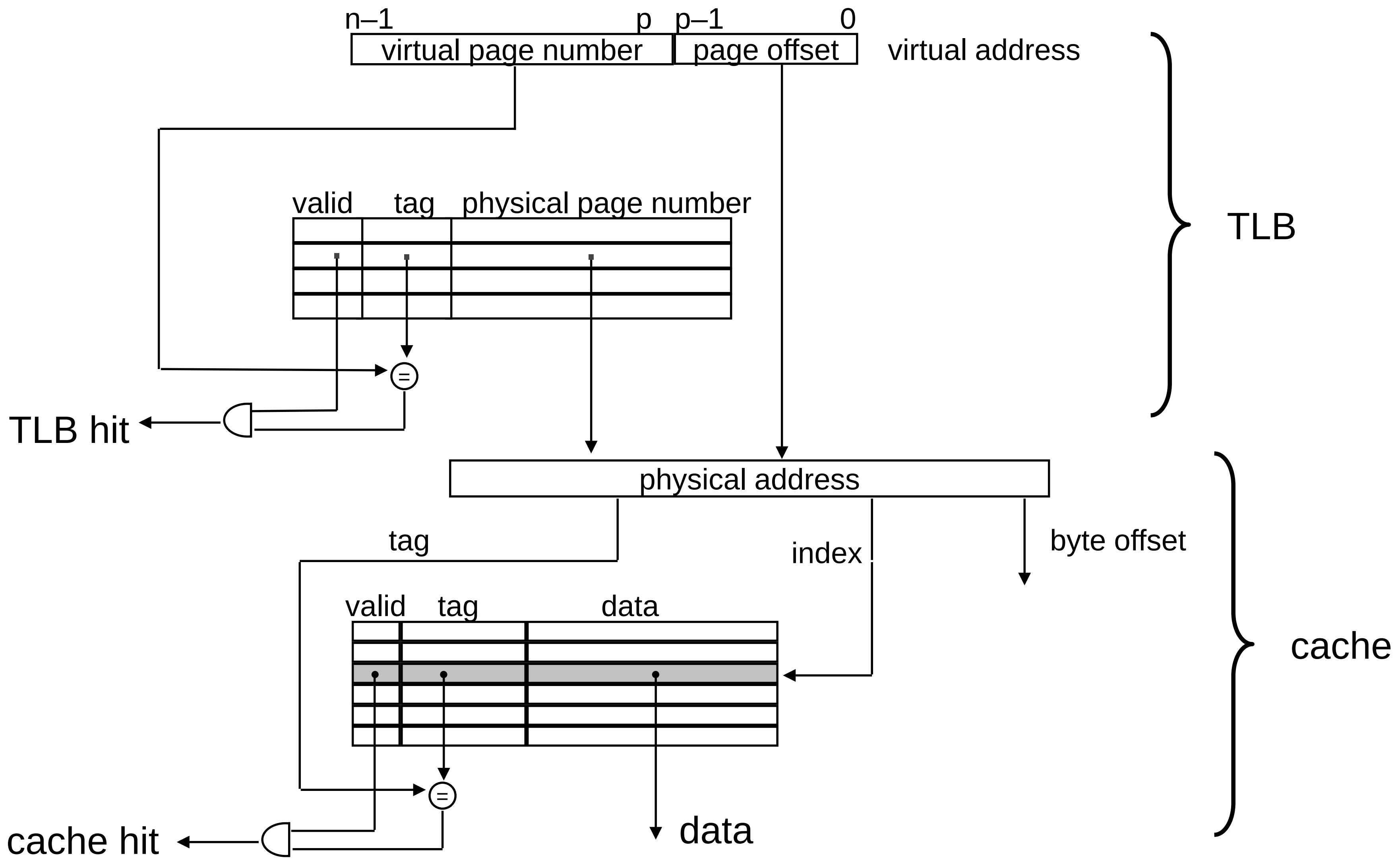


Faster Translations

- "Translation lookaside buffer" (TLB)
  - Small hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages



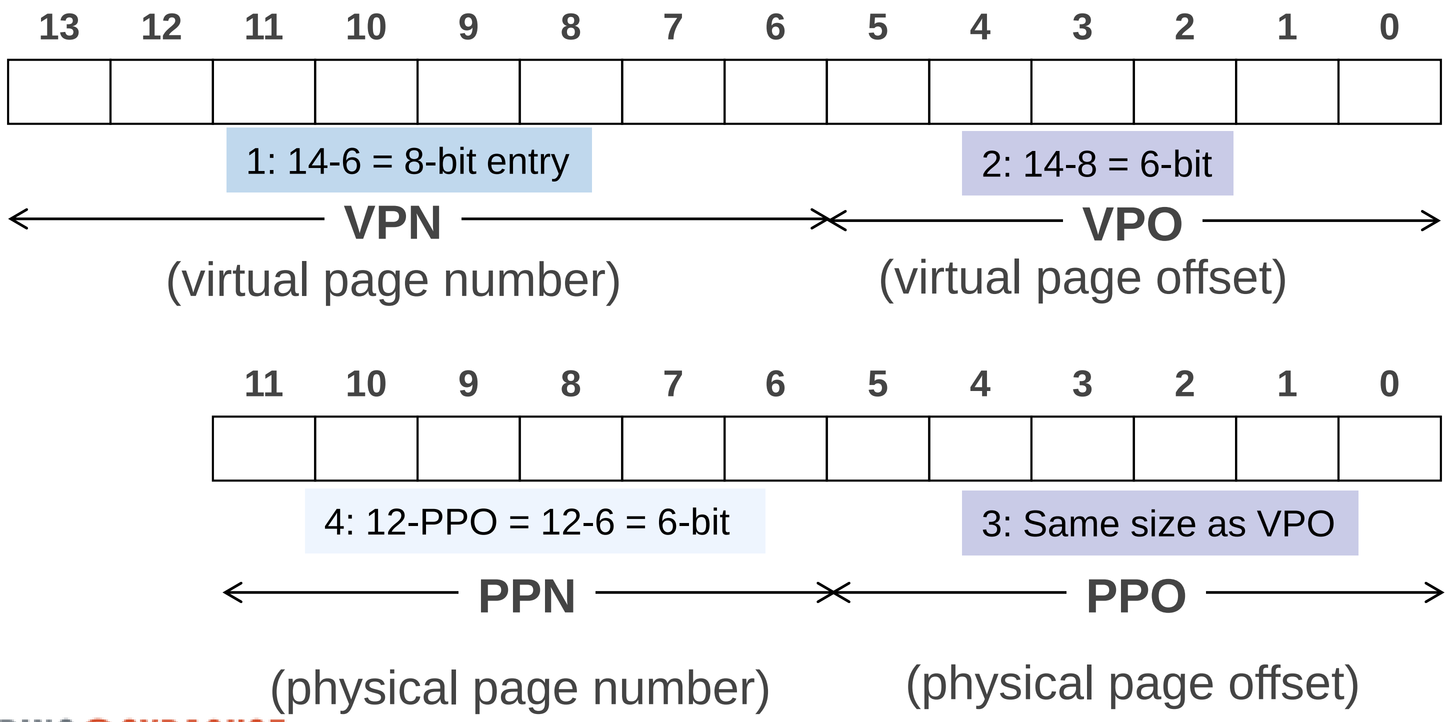
Address Translation with a TLB



ENGINEERING@SYRACUSE

Simple Memory System Example

- Addressing
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes (6-bit)



ENGINEERING@SYRACUSE

Simple Memory System Page Table

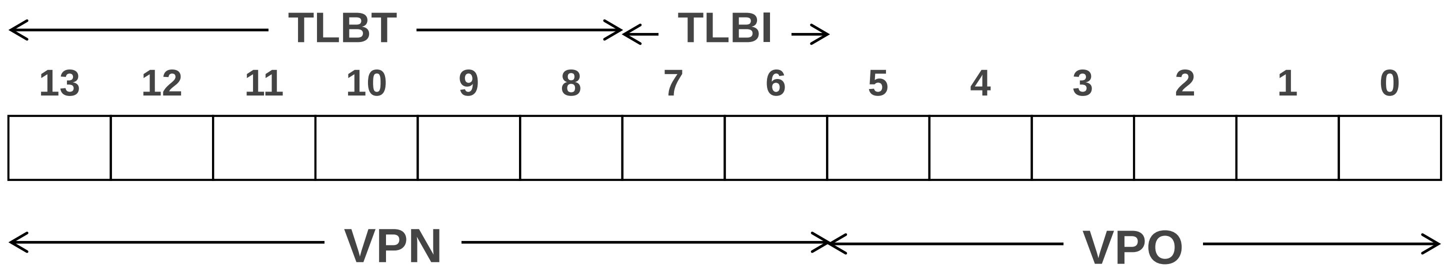
VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	–	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	–	0
04	–	0	0C	–	0
05	16	1	0D	2D	1
06	–	0	0E	11	1
07	–	0	0F	0D	1

0D = 0000 1101

- Showing only first 16 entries

Simple Memory System TLB

- TLB
  - 16 entries
  - Four-way associative

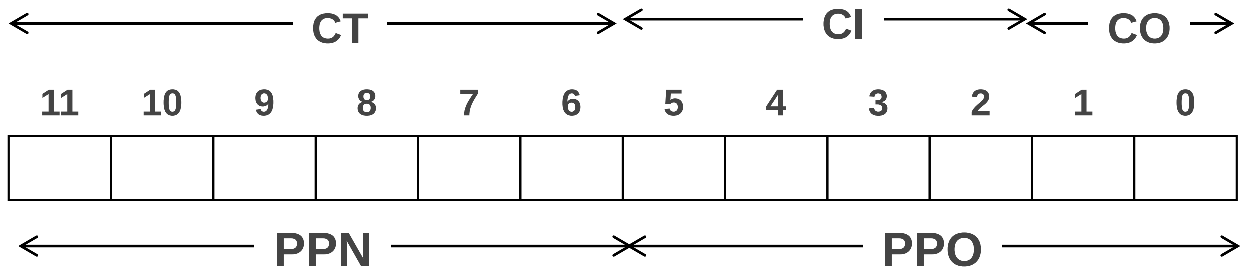


VPN = 0D = 000011 01 ➔ TAG=03, SET 1

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Simple Memory System Cache

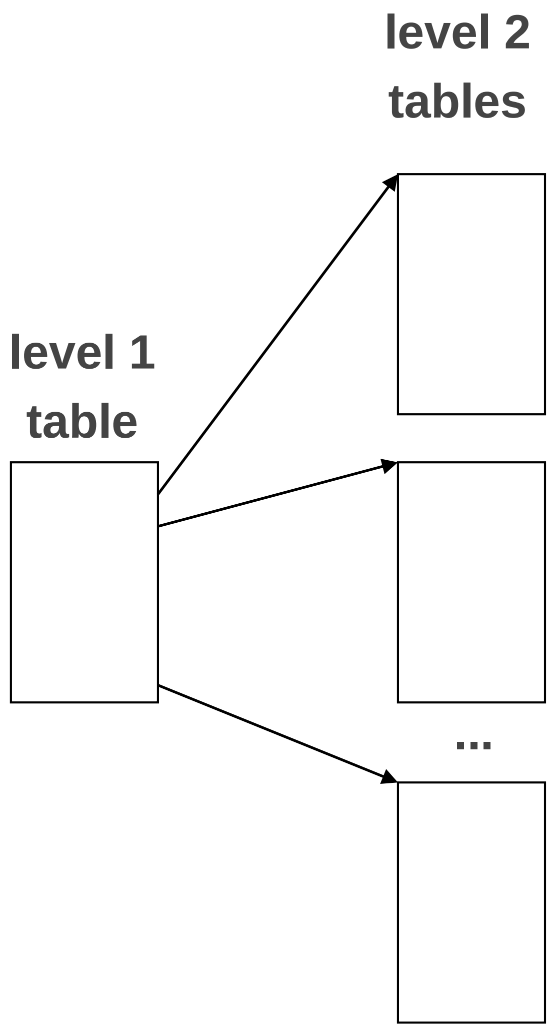
- Cache
  - 16 lines
  - 4-byte line size
  - Direct mapped



Idx	Tag	Valid	B0	B1	B2	B3	Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11	8	24	1	3A	00	51	89
1	15	0	–	–	–	–	9	2D	0	–	–	–	–
2	1B	1	00	02	04	08	A	2D	1	93	15	DA	3B
3	36	0	–	–	–	–	B	0B	0	–	–	–	–
4	32	1	43	6D	8F	09	C	12	0	–	–	–	–
5	0D	1	36	72	F0	1D	D	16	1	04	96	34	15
6	31	0	–	–	–	–	E	13	1	83	77	1B	D3
7	16	1	11	C2	DF	03	F	14	0	–	–	–	–

Multilevel Page Tables

- Given:
  - 4KB (2^{12}) page size
  - 32-bit address space
  - 4-byte PTE (to address memory)
- Problem:
  - Would need a 4 MB page table!
    - 2^{20} *4 bytes
- Common solution
  - Multilevel page tables
  - E.g., two-level table (P6)
    - Level 1 table: 1024 entries, each of which points to a level 2 page table.
    - Level 2 table: 1024 entries, each of which points to a page.



ENGINEERING@SYRACUSE

Virtual Memory

- Programmer’s view
  - Large “flat” address space.
    - Can allocate large blocks of contiguous addresses
  - Processor “owns” machine.
    - Has private address space, unaffected by other processes
- System view
  - User virtual address space created by mapping to set of pages.
    - Need not be contiguous, allocated dynamically
    - Enforce protection during address translation
  - OS manages many processes simultaneously.
    - Continually switching among processes
    - Especially when one must wait for resource
      - E.g., disk I/O to handle page fault

Virtual Memory

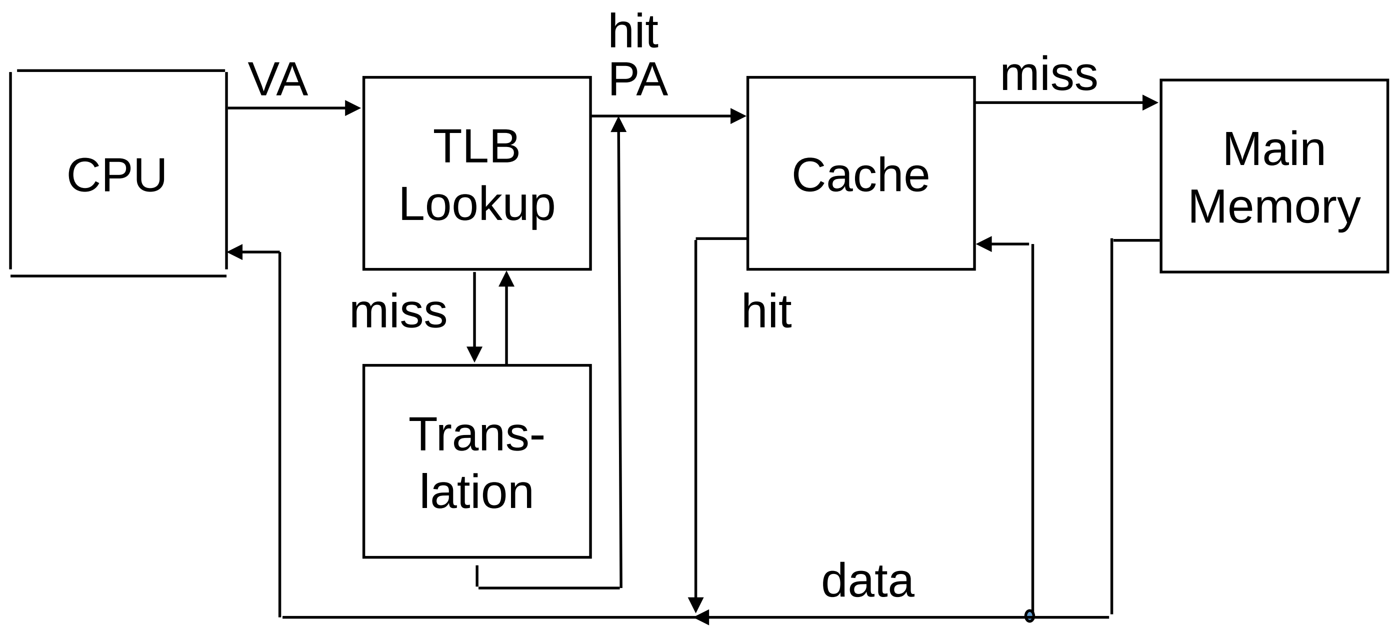
- Programmer's view
  - Large "flat" address space.
    - Can allocate large blocks of contiguous addresses
  - Processor "owns" machine.
    - Has private address space
    - Unaffected by behavior of other processes
- System view
  - User virtual address space created by mapping to set of pages.
    - Need not be contiguous
    - Allocated dynamically
    - Enforce protection during address translation
  - OS manages many processes simultaneously.
    - Continually switching among processes
    - Especially when one must wait for resource
      - E.g., disk I/O to handle page fault

ENGINEERING@SYRACUSE

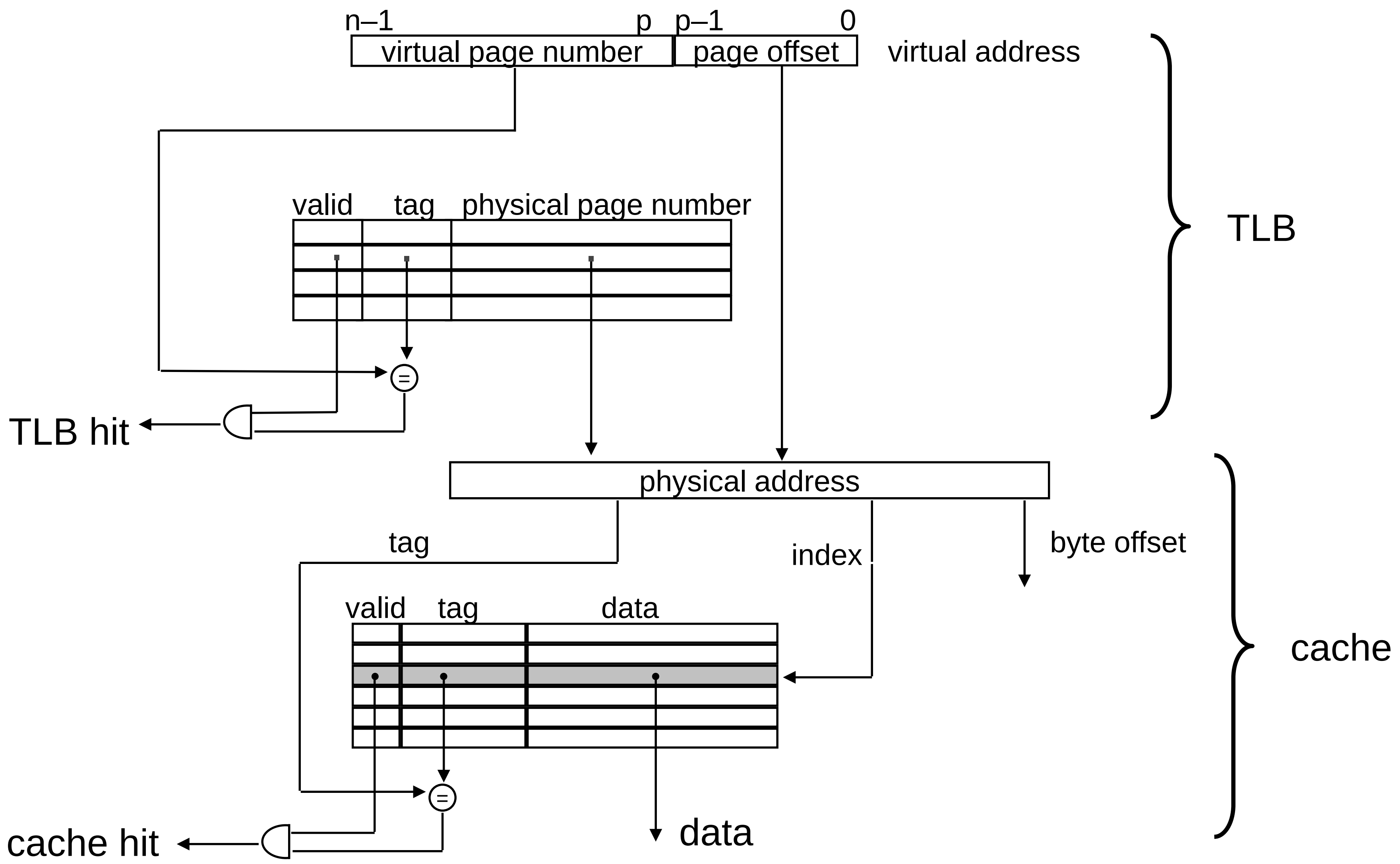
Address with Translations

Faster Translations

- “Translation lookaside buffer” (TLB)
  - Small hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages



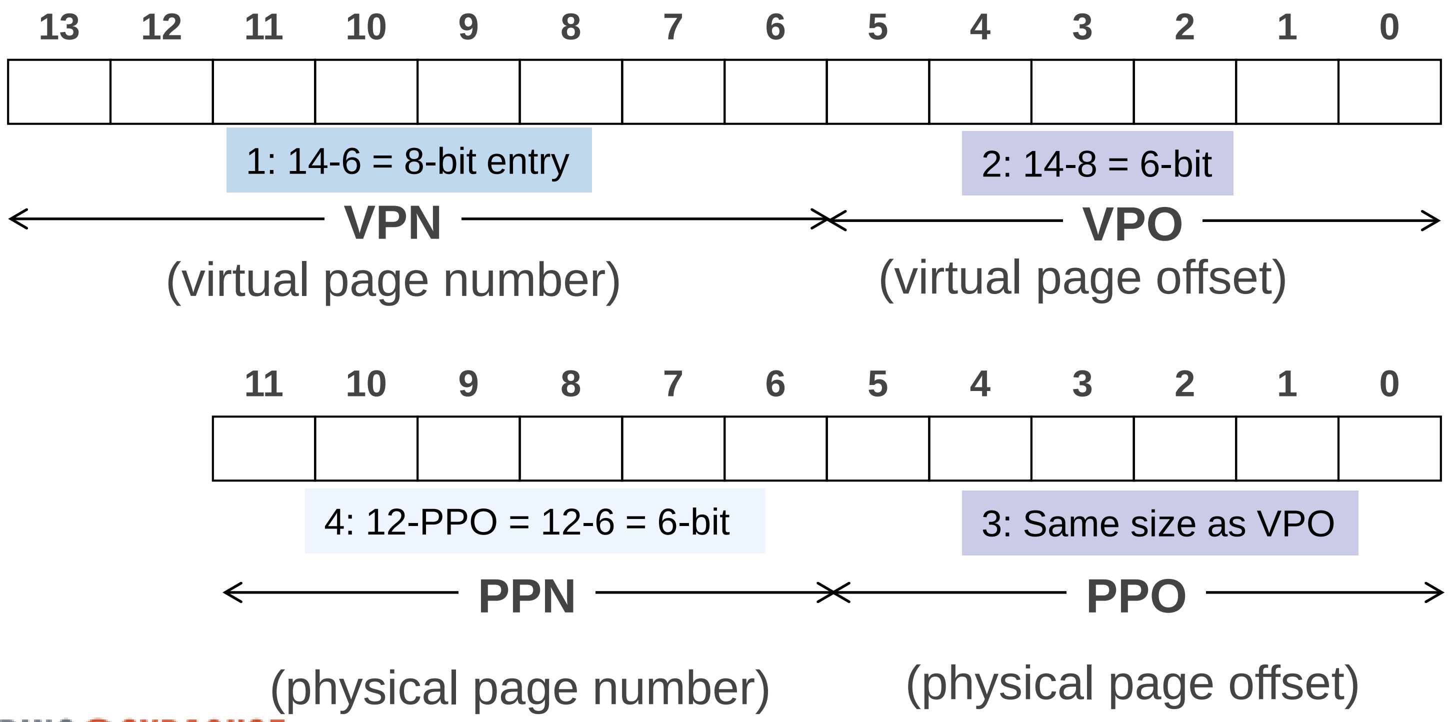
Address Translation with a TLB



ENGINEERING@SYRACUSE

Simple Memory System Example

- Addressing
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes (6-bit)



ENGINEERING@SYRACUSE

Simple Mem System Page Table

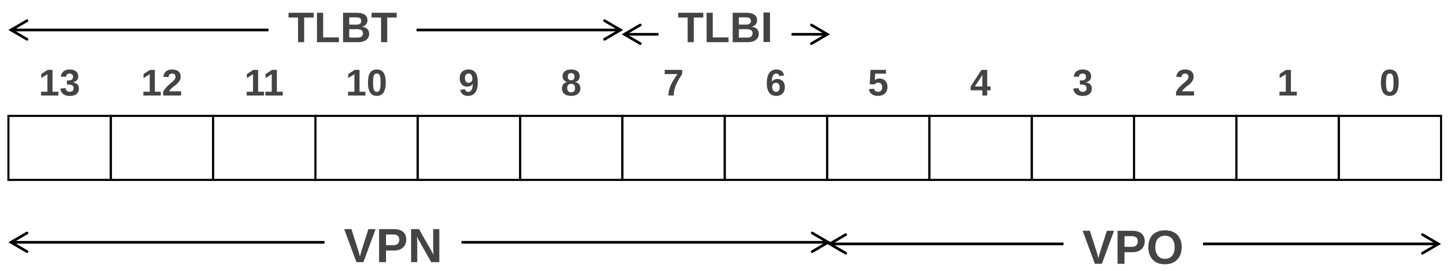
VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	–	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	–	0
04	–	0	0C	–	0
05	16	1	0D	2D	1
06	–	0	0E	11	1
07	–	0	0F	0D	1

0D = 0000 1101

- Showing only first 16 entries

Simple Memory System TLB

- TLB
  - 16 entries
  - Four-way associative

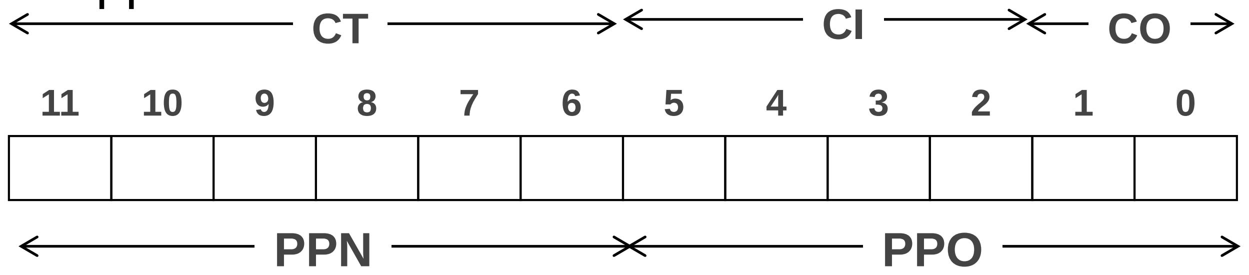


VPN = 0D = 000011 01 → TAG=03, SET 1

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Simple Memory System Cache

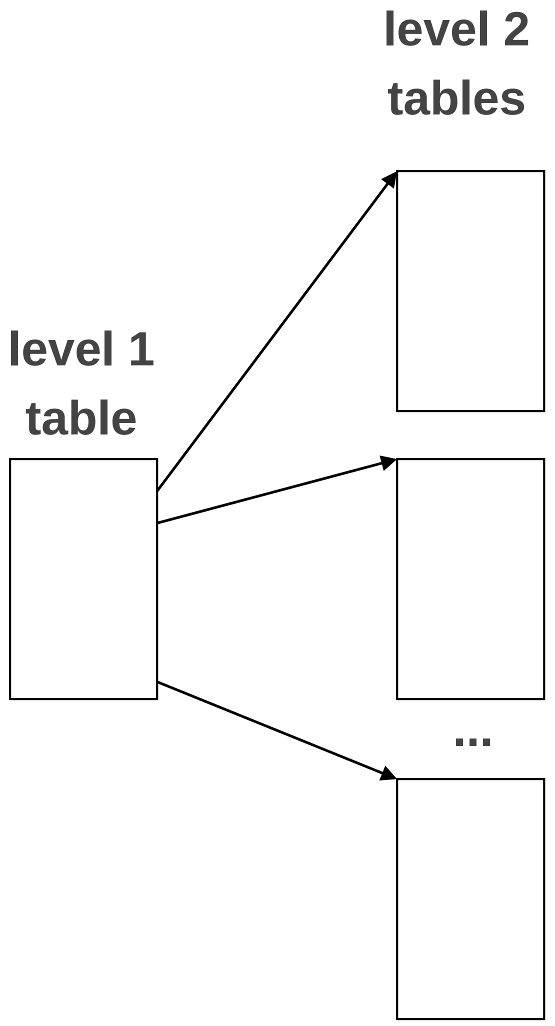
- Cache
  - 16 lines
  - 4-byte line size
  - Direct mapped



Idx	Tag	Valid	B0	B1	B2	B3	Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11	8	24	1	3A	00	51	89
1	15	0	–	–	–	–	9	2D	0	–	–	–	–
2	1B	1	00	02	04	08	A	2D	1	93	15	DA	3B
3	36	0	–	–	–	–	B	0B	0	–	–	–	–
4	32	1	43	6D	8F	09	C	12	0	–	–	–	–
5	0D	1	36	72	F0	1D	D	16	1	04	96	34	15
6	31	0	–	–	–	–	E	13	1	83	77	1B	D3
7	16	1	11	C2	DF	03	F	14	0	–	–	–	–

Multilevel Page Tables

- Given:
  - 4KB (2^{12}) page size
  - 32-bit address space
  - 4-byte PTE (to address memory)
- Problem:
  - Would need a 4 MB page table!
    - 2^{20} *4 bytes
- Common solution
  - Multilevel page tables
  - E.g., two-level table (P6)
    - Level 1 table: 1024 entries, each of which points to a level 2 page table.
    - Level 2 table: 1024 entries, each of which points to a page.



Virtual Memory

- Programmer’s view
  - Large “flat” address space.
    - Can allocate large blocks of contiguous addresses
  - Processor “owns” machine.
    - Has private address space, unaffected by other processes
- System view
  - User virtual address space created by mapping to set of pages.
    - Need not be contiguous, allocated dynamically
    - Enforce protection during address translation
  - OS manages many processes simultaneously.
    - Continually switching among processes
    - Especially when one must wait for resource
      - E.g., disk I/O to handle page fault

Virtual Memory

- Programmer's view
  - Large "flat" address space.
    - Can allocate large blocks of contiguous addresses
  - Processor "owns" machine.
    - Has private address space
    - Unaffected by behavior of other processes
- System view
  - User virtual address space created by mapping to set of pages.
    - Need not be contiguous
    - Allocated dynamically
    - Enforce protection during address translation
  - OS manages many processes simultaneously.
    - Continually switching among processes
    - Especially when one must wait for resource
      - E.g., disk I/O to handle page fault

ENGINEERING@SYRACUSE

Conclusions

In Conclusion

- Memory hierarchy
  - Cache, main memory, storage
- Advanced optimizations
  - Caches, TLBs, virtual memory
  - 4Qs: where, how, which, what
- Memory technology
  - SRAM, DRAM
- Virtual memory
  - DRAM as cache, memory management, protection
- Address translations
  - Page table, table lookaside buffer

ENGINEERING@SYRACUSE