ECE 4750 Computer Architecture, Fall 2021

T03 Fundamental Memory Concepts

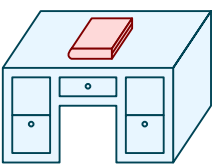
School of Electrical and Computer Engineering
Cornell University

revision: 2021-10-04-13-47

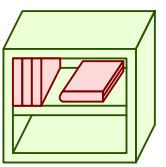
1. Memory/Library Analogy

Our goal is to do some research on a new computer architecture, and so we wish to consult the literature to learn more about past computer systems. The library contains most of the literature we are interested in, although some of the literature is stored off-site in a large warehouse. There are too many distractions at the library, so we prefer to do our reading in our doorm room or office. Our doorm room or office has an empty bookshelf that can hold ten books or so, and our desk can hold a single book at a time.

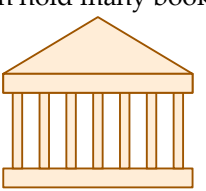
Desk
(can hold one book)



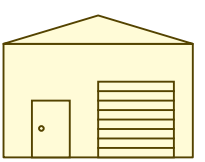
Book Shelf
(can hold a few books)



Library
(can hold many books)



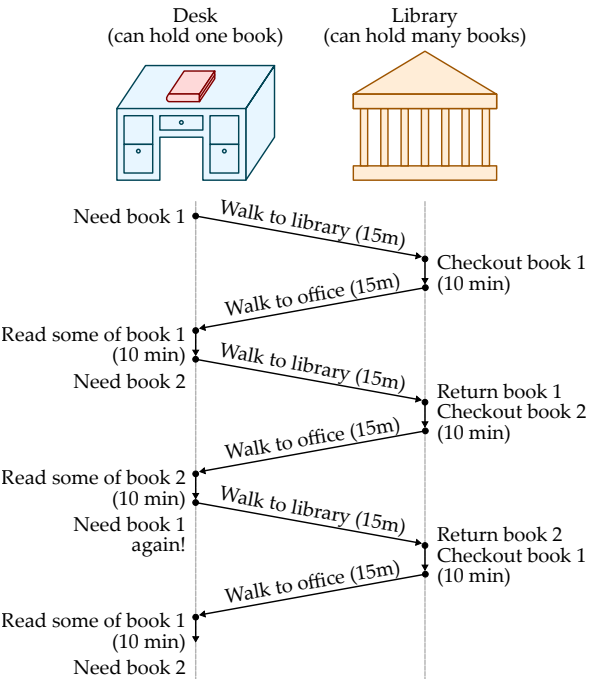
Warehouse
(long-term storage)



1.1. Three Example Scenarios

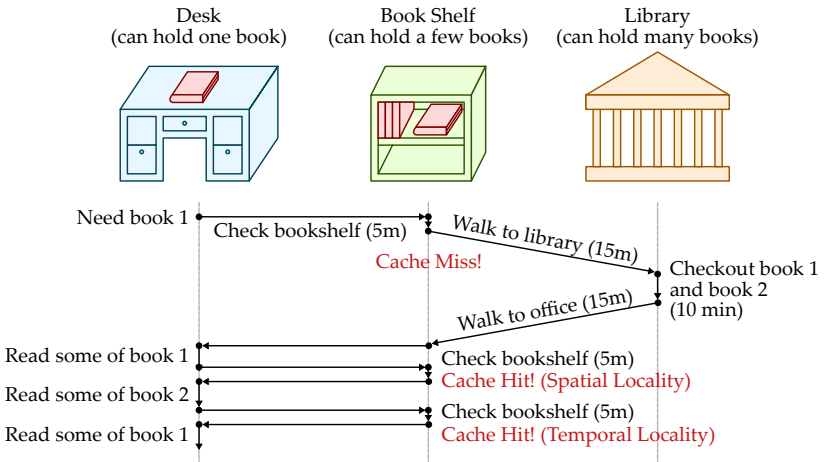
- Use desk and library
- Use desk, book shelf, and library
- Use desk, book shelf, library, and warehouse

Books from library with no bookshelf "cache"



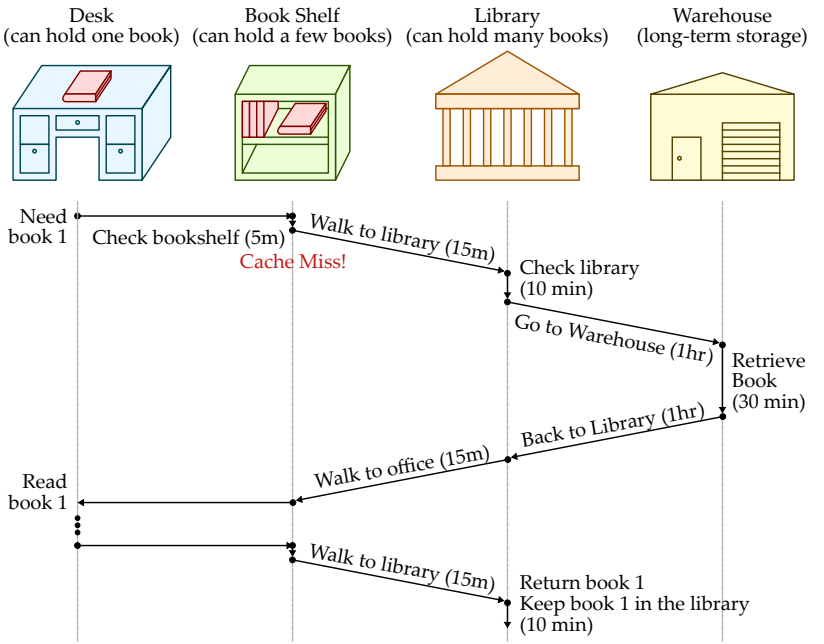
- Some inherent "translation" since we need to use the online catalog to translate a book author and title into a physical location in the library (e.g., floor, row, shelf)
- Average latency to access a book: 40 minutes
- Average throughput including reading time: 1.2 books/hour
- Latency to access library limits our throughput

Books from library with bookshelf "cache"



- Average latency to access a book: <20 minutes
- Average throughput including reading time: ≈2 books/hour
- Bookshelf acts as a small "cache" of the books in the library
  - Cache Hit: Book is on the bookshelf when we check, so there is no need to go to the library to get the book
  - Cache Miss: Book is not on the bookshelf when we check, so we need to go to the library to get the book
- Caches exploit structure in the access pattern to avoid the library access time which limits throughput
  - Temporal Locality: If we access a book once we are likely to access the same book again in the near future
  - Spatial Locality: If we access a book on a given topic we are likely to access other books on the same topic in the near future

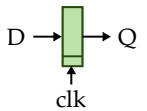
Books from warehouse

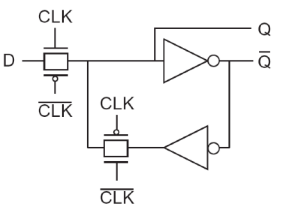


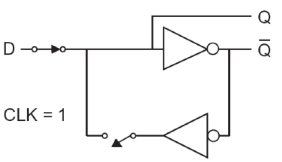
- Keep very frequently used books on book shelf, but also keep books that have recently been checked out in the library before moving them back to long-term storage in the warehouse
- We have created a "book storage hierarchy"
- Book Shelf : low latency, low capacity
- Library : high latency, high capacity
- Warehouse : very high latency, very high capacity

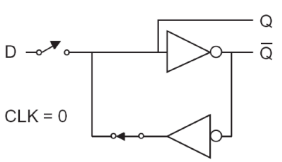
1.2. Memory Technology

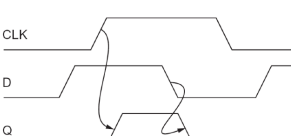
Level-High Latch



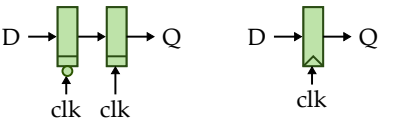


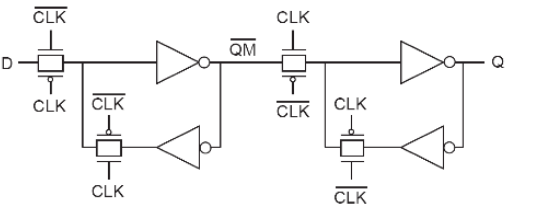


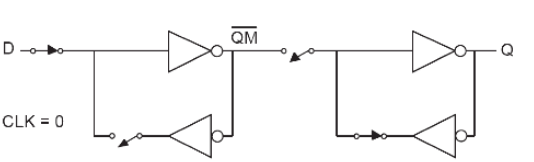


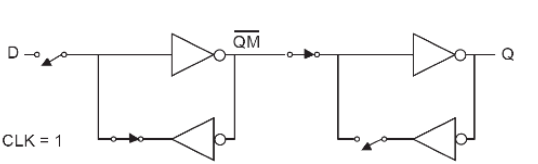


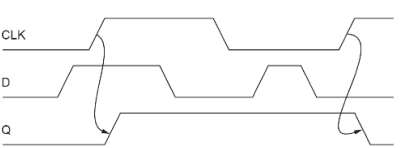
Positive Edge-Triggered Register



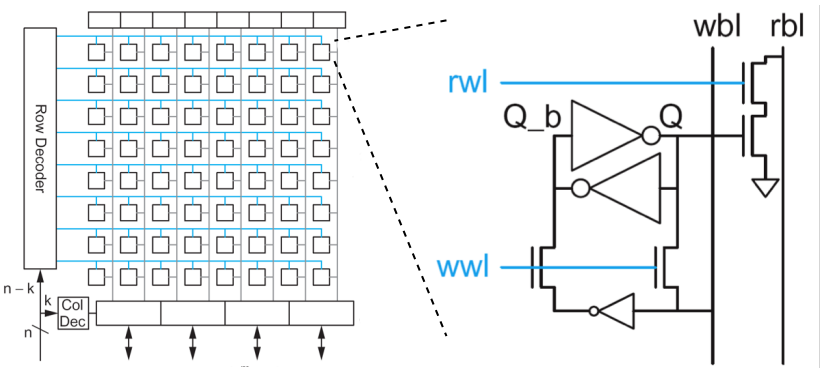




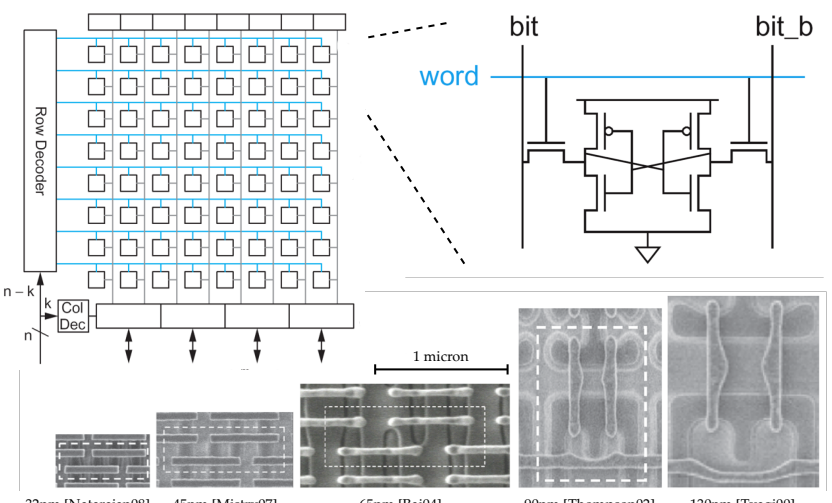




Memory Arrays: Register Files

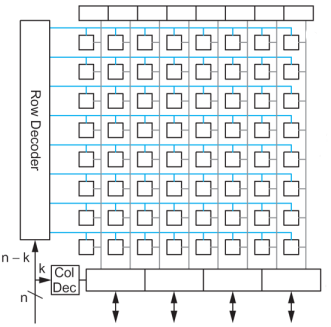


Memory Arrays: SRAM

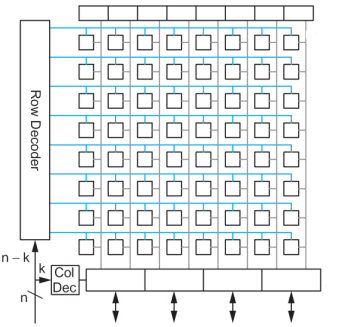


32nm [Natarajan08] 45nm [Mistry07] 65nm [Bai04] 90nm [Thompson02] 130nm [Tyagi00]

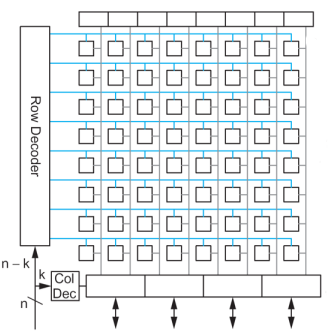
Full-Word Write



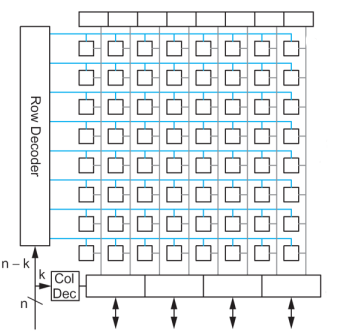
Partial-Word Write



Combinational-Read

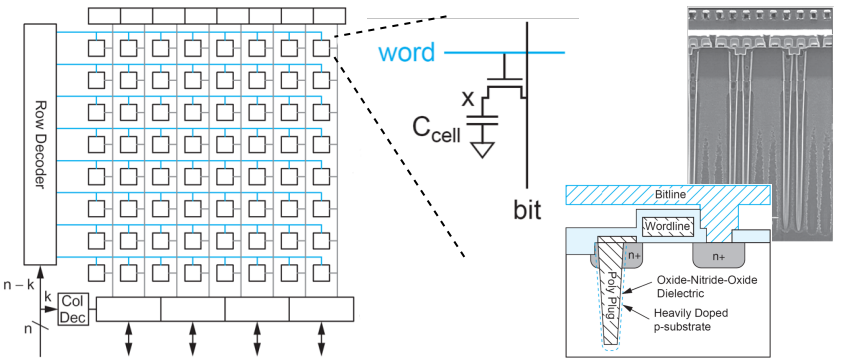


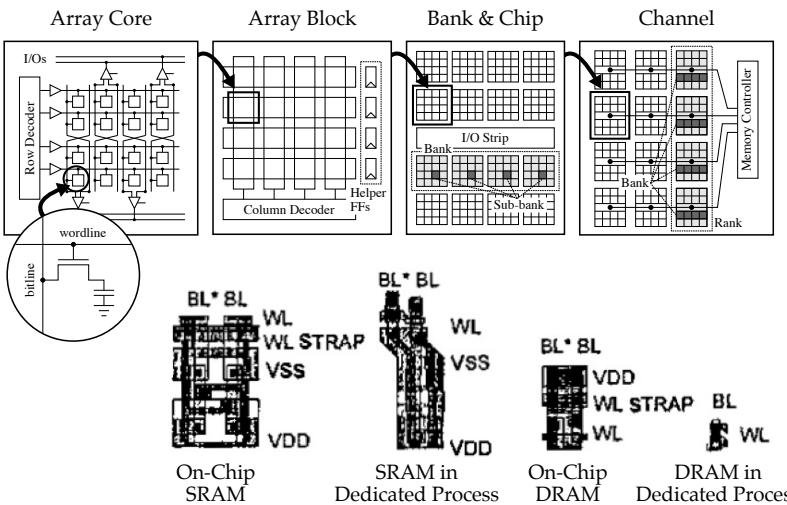
Synchronous-Read



No such thing as a "combinational write"!

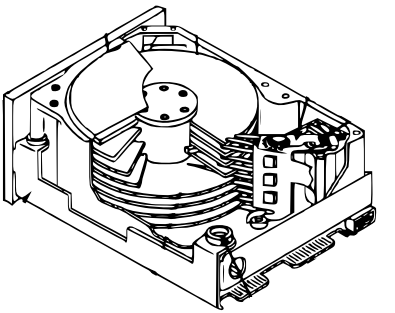
Memory Arrays: DRAM





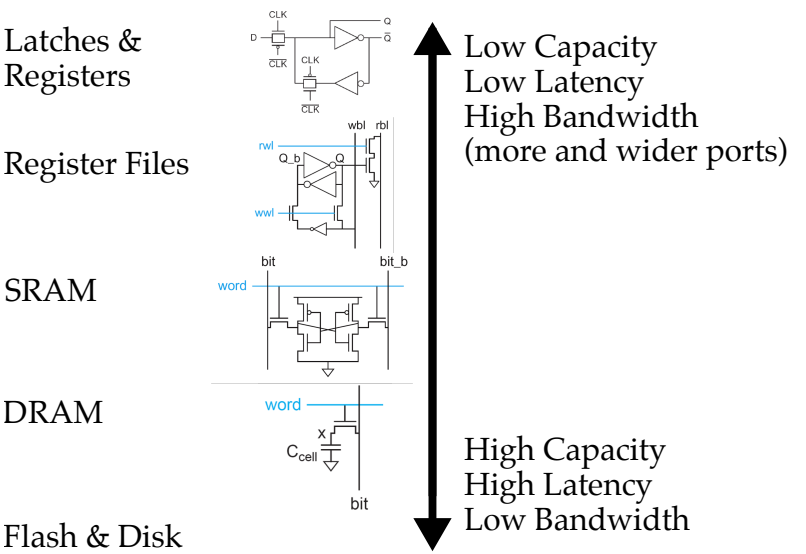
Adapted from [Foss, "Implementing Application-Specific Memory." ISSCC'96]

Flash and Disk



- Magnetic hard drives require rotating platters resulting in long random acess times which have hardly improved over several decades
- Solid-state drives using flash have 100× lower latencies but also lower density and higher cost

Memory Technology Trade-Offs

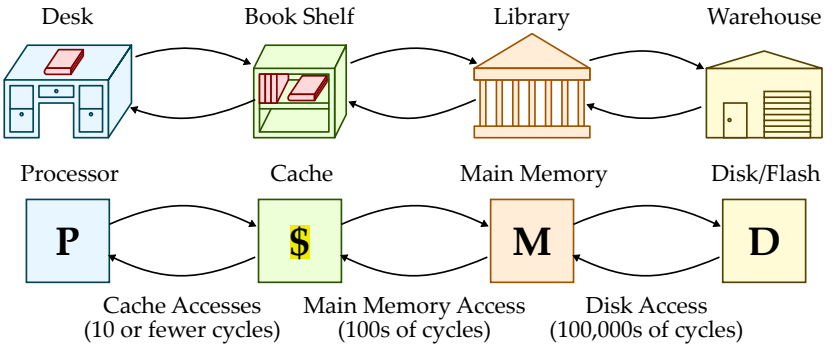


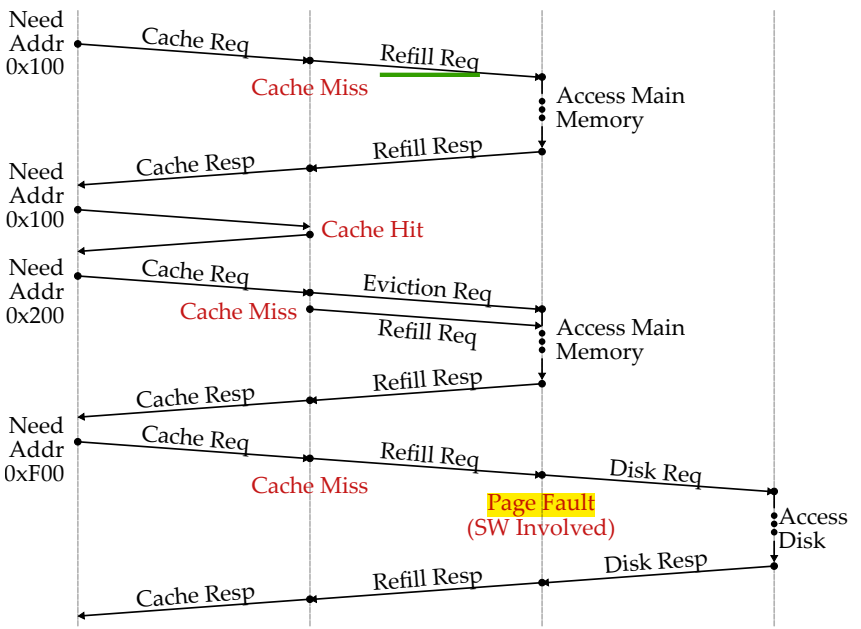
Latency numbers every programmer (architect) should know

L1 cache reference	1 ns
Branch mispredict	3 ns
L2 cache reference	4 ns
Mutex lock/unlock	17 ns
Main memory reference	100 ns
Send 2KB over commodity network	250 ns
Compress 1KB with zip	2 us
Read 1MB sequentially from main memory	9 us
SSD random read	16 us
Read 1MB sequentially from SSD	156 us
Round trip in datacenter	500 us
Read 1MB sequentially from disk	2 ms
Disk random read	4 ms
Packet roundtrip from CA to Netherlands	150 ms

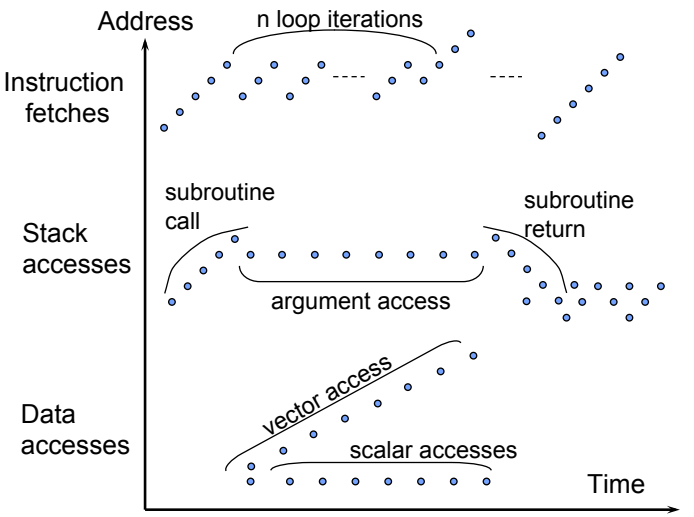
`http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html`

1.3. Cache Memories in Computer Architecture





Cache memories exploit temporal and spatial locality



Understanding locality for assembly programs

Examine each of the following assembly programs and rank each program based on the level of temporal and spatial locality in both the instruction and data address stream on a scale from 0 to 5 with 0 being no locality and 5 being very significant locality.

	Inst Temp	Inst Spat	Data Temp	Data Spat

```
loop:
  lw   x1, 0(x2)
  lw   x3, 0(x4)
  add  x5, x1, x3
  sw   x5, 0(x6)
  addi x2, x2, 4
  addi x4, x4, 4
  addi x6, x6, 4
  addi x7, x7, -1
  bne  x7, x0, loop
```

```
loop:
  lw   x1, 0(x2)
  lw   x3, 0(x1) # random ptrs
  lw   x4, 0(x3) # random ptrs
  addi x4, x4, 1
  addi x2, x2, 4
  addi x7, x7, -1
  bne  x7, x0, loop
```

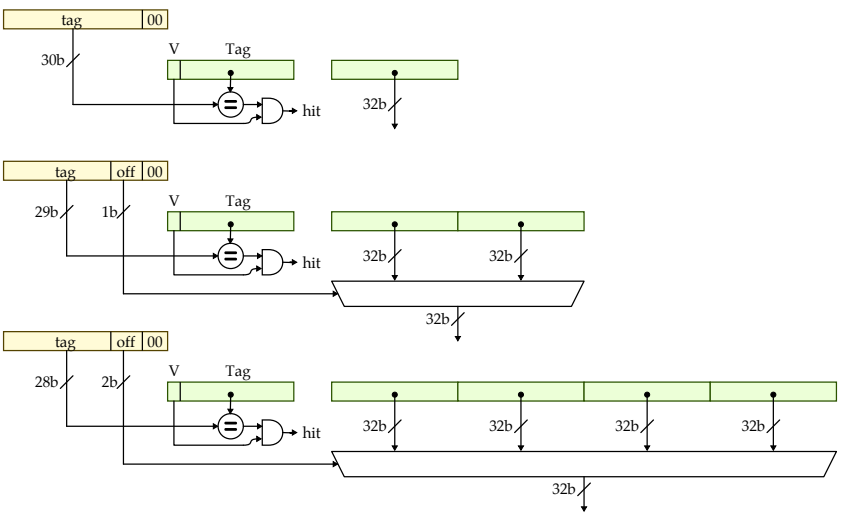
```
loop:
  lw   x1, 0(x2) # many diff
  jalr x1        # func ptrs
  addi x2, x2, 4
  addi x7, x7, -1
  bne  x7, x0, loop
```

2. Cache Concepts

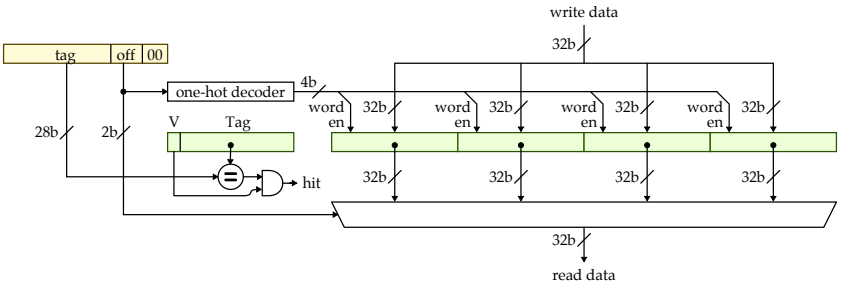
- Single-line cache
- Multi-line cache
- Replacement policies
- Write Policies
- Categorizing Misses

2.1. Single-Line Cache

Consider only 4B word accesses and only the read path for three single-line cache designs:



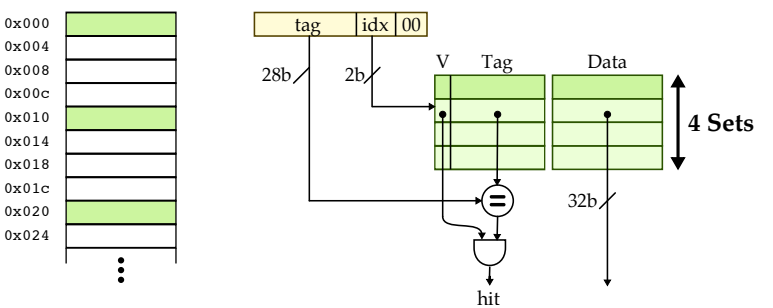
What about writes?



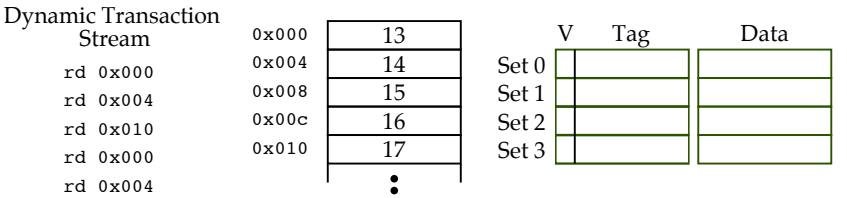
- Spatial Locality: Refill entire cache line at once
- Temporal Locality: Reuse word multiple times

2.2. Multi-Line Cache

Consider a four-line direct-mapped cache with 4B cache lines



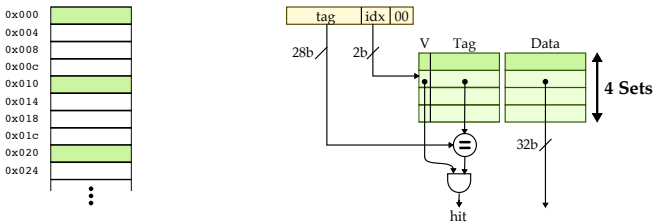
Example execution worksheet and table for direct-mapped cache



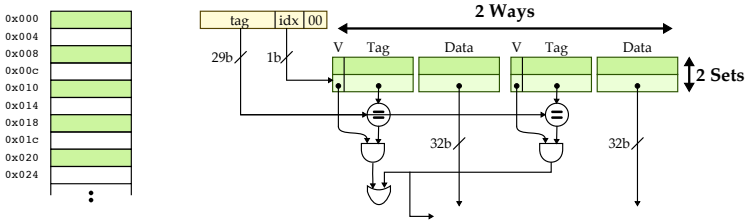
				Set			
	tag	**idx**	**h/m**	**0**	**1**	**2**	**3**
`rd 0x000`							
`rd 0x004`							
`rd 0x010`							
`rd 0x000`							
`rd 0x004`							
`rd 0x020`							

Increasing cache associativity

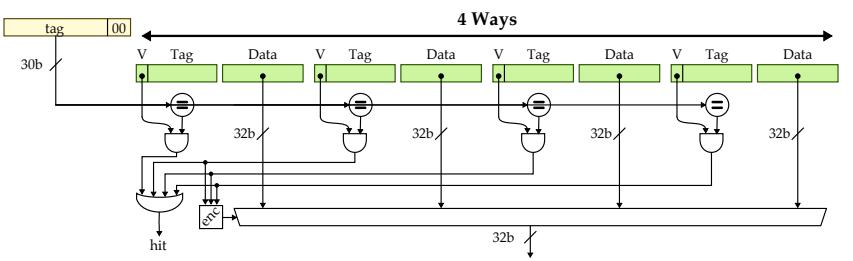
Four-line direct-mapped cache with 4B cache lines



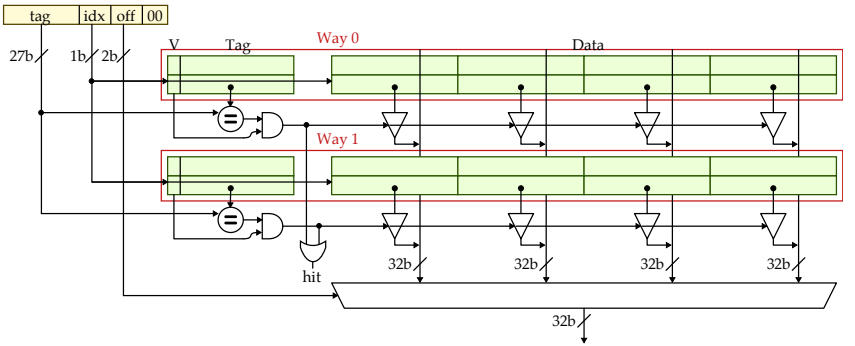
Four-line two-way set-associative cache with 4B cache lines



Four-line fully-associative cache with 4B cache lines



Combining associativity with longer cache lines

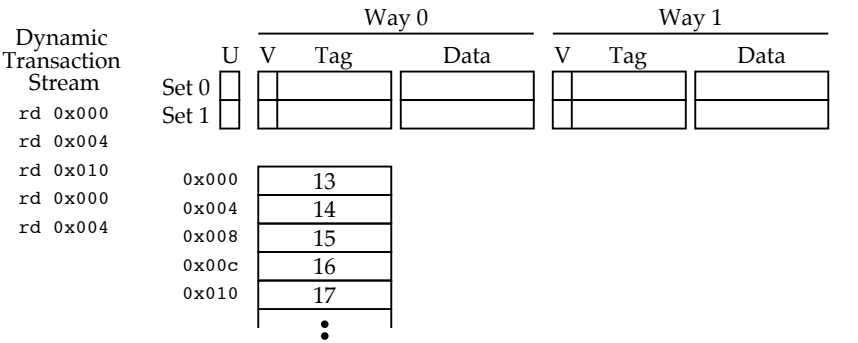


- Spatial Locality: Refill entire cache line + simple indexing to find set
- Temporal Locality: Reuse word multiple times + replacement policy

2.3. Replacement Policies

- No choice in a direct-mapped cache
- Random
  - Good average case performance, but difficult to implement
- Least Recently Used (LRU)
  - Replace cache line which has not been accessed recently
  - LRU cache state must be updated on every access which is expensive
  - True implementation only feasible for small sets
  - Two-way cache can use a single “last used bit”
  - Pseudo-LRU uses binary tree to approximate LRU for higher associativity
- First-In First-Out (FIFO, Round Robin)
  - Simpler implementation, but does not exploit temporal locality
  - Potentially useful in large fully associative caches

Example execution worksheet and table for 2-way set associative cache



	tag	idx	h/m	Set 0			Set 1		
				U	Way 0	Way 1	U	Way 0	Way 1
rd 0x000									
rd 0x004									
rd 0x010									
rd 0x000									
rd 0x004									
rd 0x020									

2.4. Write Policies

Write-Through with No Write Allocate

- On write miss, write memory but do not bring line into cache
- On write hit, write both cache and memory
- Requires more memory bandwidth, but simpler to implement

				Set				write
	tag	idx	h/m	0	1	2	3	mem?
`rd 0x010`								
`wr 0x010`								
`wr 0x024`								
`rd 0x024`								
`rd 0x020`								

Assume 4-line direct-mapped cache with 4B cache lines

Write-Back with Write Allocate

- On write miss, bring cache line into cache then write
- On write hit, only write cache, do not write memory
- Only update memory when a dirty cache line is evicted
- More efficient, but more complicated to implement

				Set				write
	tag	idx	h/m	0	1	2	3	mem?
`rd 0x010`								
`wr 0x010`								
`wr 0x024`								
`rd 0x024`								
`rd 0x020`								

Assume 4-line direct-mapped cache with 4B cache lines

2.5. Categorizing Misses: The Three C's

- Compulsory : first-reference to a block
- Capacity : cache is too small to hold all of the data
- Conflict : collisions in a specific set

Classifying misses in a cache with a target capacity and associativity as a sequence of three questions:

- Q1) Would this miss occur in a cache with infinite capacity? If the answer is yes, then this is a compulsory miss and we are done. If the answer is no, then consider question 2.
- Q2) Would this miss occur in a *fully associative* cache with the desired capacity? If the answer is yes, then this is a capacity miss and we are done. If the answer is no, then consider question 3.
- Q3) Would this miss occur in a cache with the desired capacity and associativity? If the answer is yes, then this is a conflict miss and we are done. If the answer is no, then this is not a miss – it is a hit!

Example 1 illustrating categorizing misses

Assume we have a direct-mapped cache with two 16B lines, each with four 4B words for a total cache capacity of 32B. We will need four-bits for the offset, one bit for the index, and the remaining bits for the tag.

	tag	idx	h/m	type	Set 0	Set 1
`rd 0x000`						
`rd 0x020`						
`rd 0x000`						
`rd 0x020`						

Q1. Would the cache miss occur in an infinite capacity cache? For the first two misses, the answer is yes so they are compulsory misses. For the last two misses, the answer is no, so consider question 2.

Q2. Would the cache miss occur in a fully associative cache with the target capacity (two 16B lines)? Re-run address stream on such a fully associative cache. For the last two misses, the answer is no, so consider question 3.

	tag	h/m	Way 0	Way 1
`rd 0x000`				
`rd 0x020`				
`rd 0x000`				
`rd 0x020`				

3. Would the cache miss occur in a cache with the desired capacity and associativity? For the last two misses, the asnwer is yes, so these are conflict misses. There is enough capacity in the cache; the limited associativity is what is causing the misses.

Example 2 illustrating categorizing misses

Assume we have a direct-mapped cache with two 16B lines, each with four 4B words for a total cache capacity of 32B. We will need four-bits for the offset, one bit for the index, and the remaining bits for the tag.

	tag	idx	h/m	type	Set 0	Set 1
`rd 0x000`						
`rd 0x020`						
`rd 0x030`						
`rd 0x000`						

Q1. Would the cache miss occur in an infinite capacity cache? For the first three misses, the answer is yes so they are compulsory misses. For the last miss, the answer is no, so consider question 2.

Q2. Would the cache miss occur in a fully associative cache with the target capacity (two 16B lines)? Re-run address stream on such a fully associative cache. For the last miss, the answer is yes, so this is a capacity miss.

	tag	h/m	Way 0	Way 1
`rd 0x000`				
`rd 0x020`				
`rd 0x030`				
`rd 0x000`				

Categorizing misses helps us understand how to reduce miss rate. Should we increase associativity? Should we use a larger cache?

3. Memory Translation, Protection, and Virtualization

Memory Management Unit (MMU)

- Translation : mapping of virtual addresses to physical addresses
- Protection : permission to access address in memory
- Virtualization : transparent extension of memory space using disk

Most modern systems provide support for all three functions
with a single paged-based MMU

3.1. Memory Translation

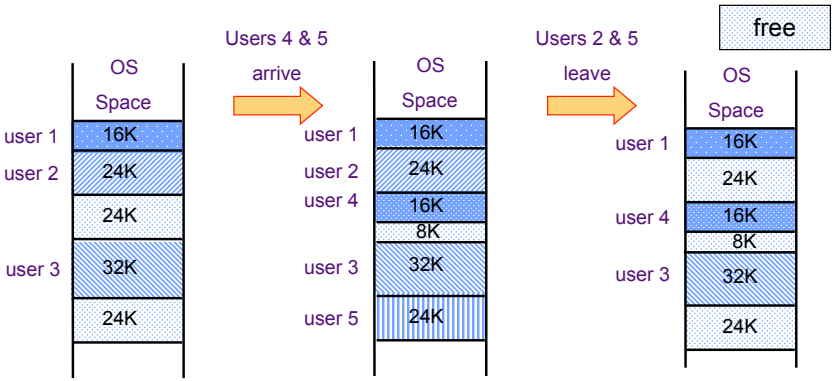
Mapping of virtual addresses to physical addresses

Why memory translation?

- Enables using full virtual address space with less physical memory
- Enables multiple programs to execute concurrently
- Can facilitate memory protection and virtualization

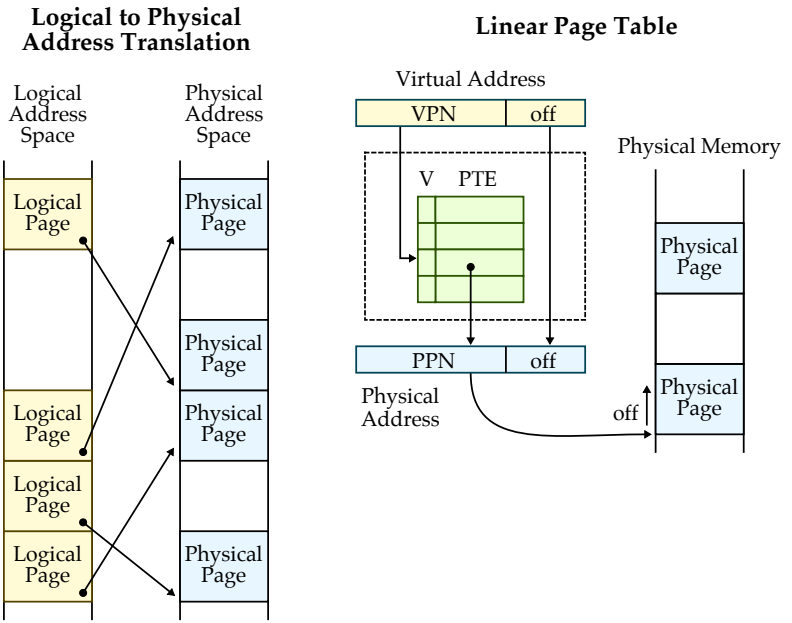
Simple base-register translation

Memory fragmentation

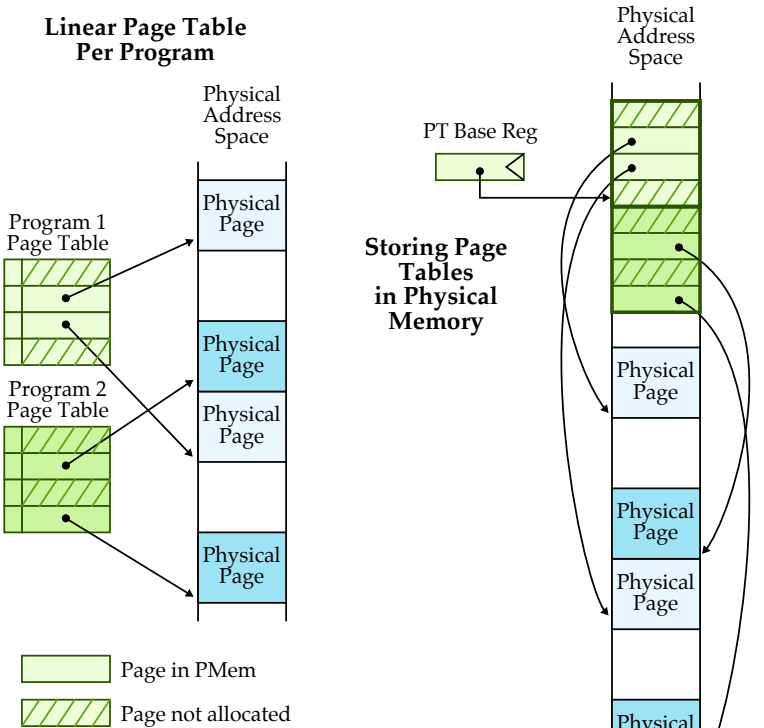


As users come and go, the storage is “fragmented”. Therefore, at some stage programs have to be moved around to compact the storage.

Linear-page table tranlsation



- Logical address can be interpreted as a page number and offset
- Page table contains the physical address of the base of each page
- Page tables make it possible to store the pages of a program non-contiguously

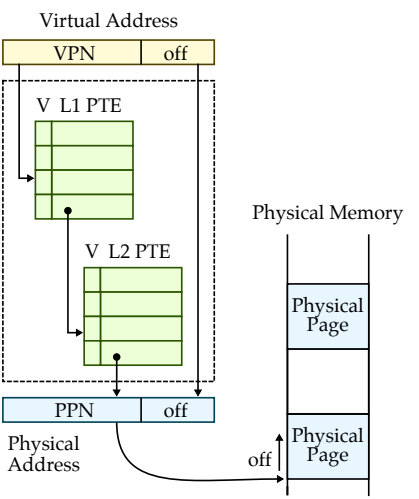


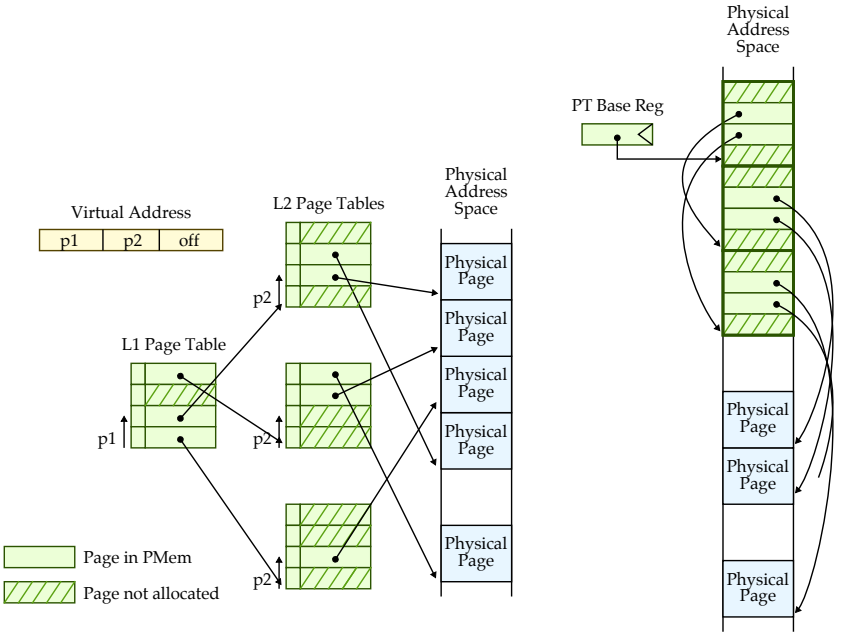
- Not all page tabele entries (PTEs) are valid
- Invalid PTE means the program has not allocated that page
- Each program has its own page table with entry for each logical page
- Where should page tables reside?
  - Space required by page table proportional to address space
  - Too large too keep in registers
  - Keep page tables in memory
  - Need one mem access for page bage address, and another for actual data

Size of linear page table?

- With 32-bit addresses, 4KB pages, and 4B PTEs
  - Potentially 4GB of physical memory needed per program
  - 4KB page means VPN is 20 bits and offset is 12 bits
  - 2^{20} PTEs, which means 4MB page table overhead per program
- With 64-bit addresses, 1MB pages, and 8B PTEs
  - 1MB pages means VPN is 44 bits and offset is 20 bits
  - 2^{44} PTEs, which means 140TB page table overhead per program
- How can this possible ever work? Exploit program structure, i.e., sparsity in logical address usage

Two-level table translation

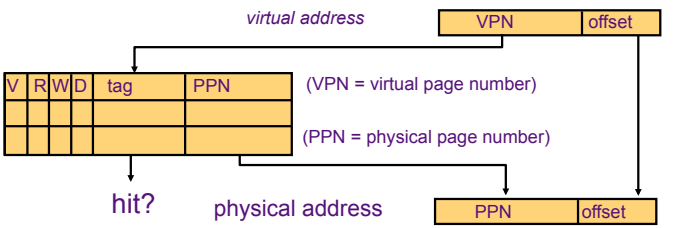




- Again, we store page tables in physical memory
- Space requirements are now much more modest
- Now need three memory accesses to retieve one piece of data

Translation lookaside buffers

- Address translation is very expensive
- Every reference requires multiple memory accesses
- Solution: Cache translations in a translation lookaside buffer
  - TLB Hit: Single-cycle translation
  - TLB Miss: Page table walk to refill TLB

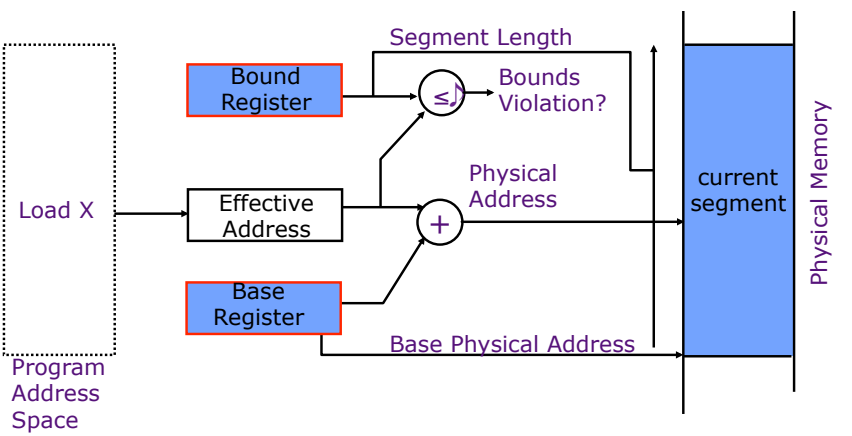


- Typically 32-128 entries, usually fully associative
  - Each entry maps large number of consecutive addresses so most spatial locality within page as opposed to across pages -> More likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-8 way set-associative
  - Larger systems sometimes have multi-level (L1 and L2) TLBs
- Random or FIFO replacement policy
- Usually no program identifier in the TLB
  - Flush TLB on program context switch
- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB
  - Example: 64 TLB entries, 4KB pages, one page pery entry
  - TLB Reach: 64 entries * 4 KB = 256 KB (if contiguous)

- Handling a TLB miss in software (MIPS, Alpha)
  - TLB miss causes an exception and the operating system walks the page tables and reloads TLB. A privileged "untranslated" addressing mode used for walk
- Handling a TLB miss in hardware (SPARCv8, x86, PowerPC)
  - The memory management unit (MMU) walks the page tables and reloads the TLB, any additional complexities encountered during walk causes MMU to give up and signal an exception

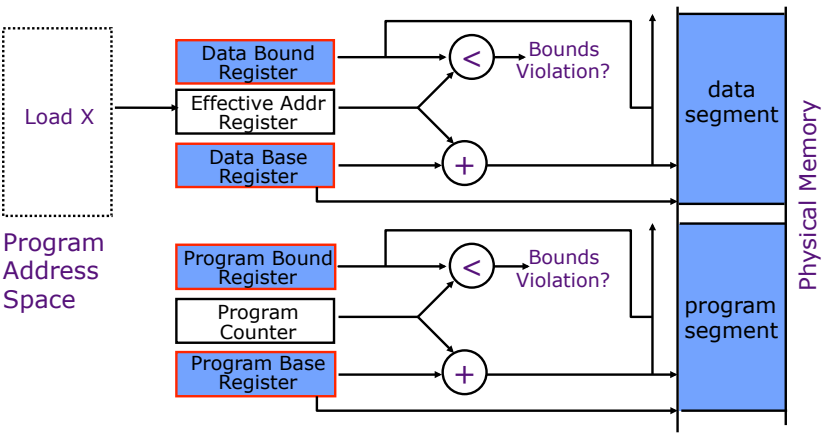
3.2. Memory Protection

Base-and-bound protection



Base and bounds registers are visible/accessible only when processor is running in the *supervisor mode*

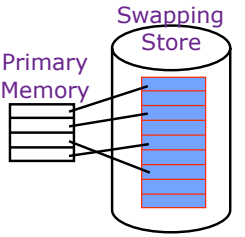
Separate areas for program and data



Page-based protection

- We can store protection information in the page-tables to enable page-level protection
- Protection information prevents two programs from being able to read or write each other's physical memory space

3.3. Memory Virtualization



- More than just translation and protection
- Use disk to extend apparent size of mem
- Treat DRAM as cache of disk contents
- Only need to hold active working set of processes in DRAM, rest of memory image can be swapped to disk
- Inactive processes can be completely swapped to disk (except usually the root of the page table)
- Hides machine configuration from software
- Implemented with combination of hardware/software

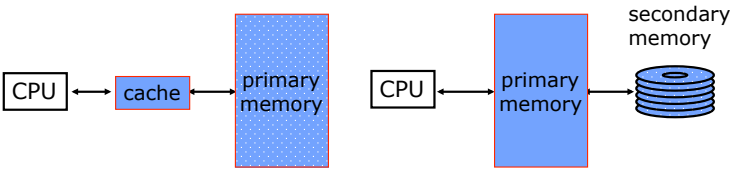
Page Fault Handler

- When the referenced page is not in DRAM:
  - The missing page is located (or created)
  - It is brought in from disk, and page table is updated

    Another job may be run on the CPU while the first job waits for the requested page to be read from disk
  - If no free pages are left, a page is swapped out

    Pseudo-LRU replacement policy
- Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the OS
  - Untranslated addressing mode is essential to allow kernel to access page tables

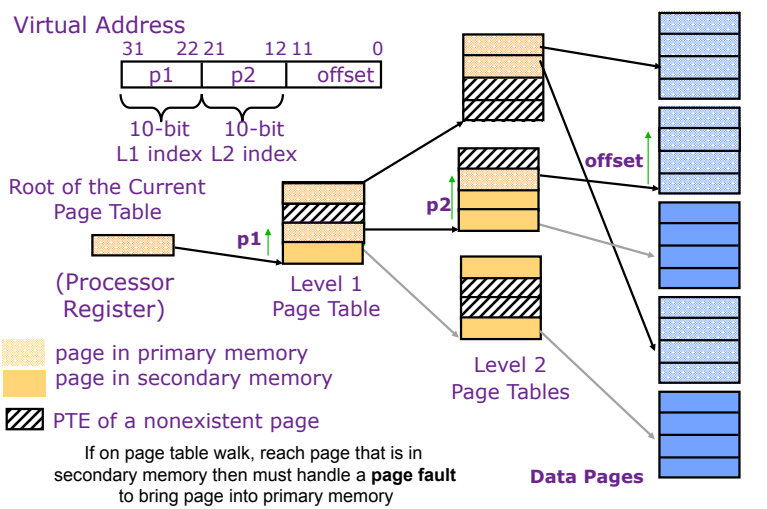
Caching vs. Demand Paging

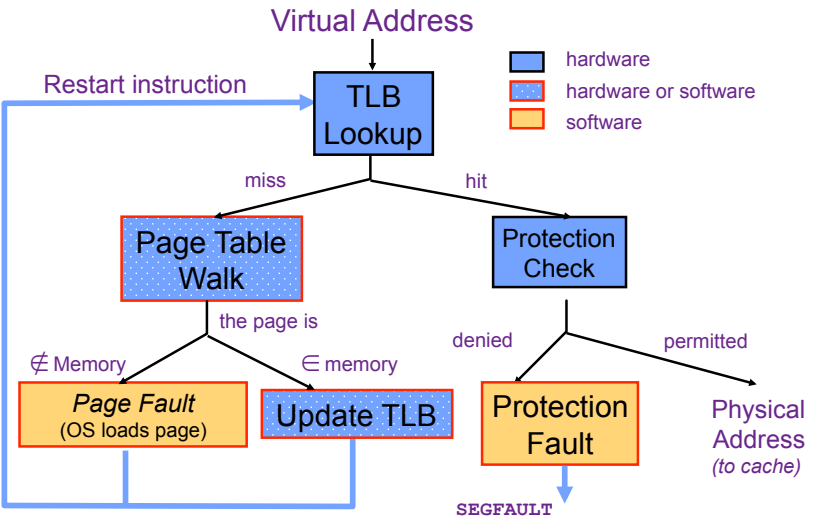


Caching
- cache entry
- cache block (~32 bytes)
- cache miss rate (1% to 20%)
- cache hit (~1 cycle)
- cache miss (~100 cycles)
- a miss is handled in *hardware*

Demand paging
- page frame
- page (~4K bytes)
- page miss rate (<0.001%)
- page hit (~100 cycles)
- page miss (~5M cycles)
- a miss is handled mostly in *software*

Hierarchical Page Table with VM




Virtual Address
hardware
hardware or software
software
Restart instruction
TLB Lookup
miss
hit
Page Table Walk
Protection Check
the page is
denied
permitted
∉ Memory
∈ memory
Page Fault
(OS loads page)
Update TLB
Protection Fault
Physical Address
(to cache)
SEGFAULT

4. Analyzing Memory Performance

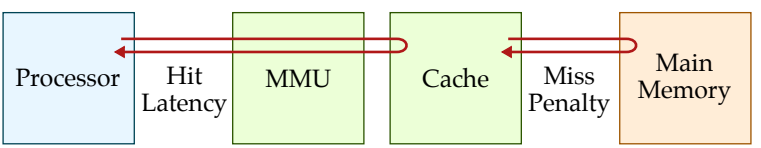
$$\frac{\text{Time}}{\text{Mem Access Sequence}} = \frac{\text{Mem Accesses}}{\text{Sequence}} \times \frac{\text{Avg Cycles}}{\text{Mem Access}} \times \frac{\text{Time}}{\text{Cycle}}$$

$$\frac{\text{Avg Cycles}}{\text{Mem Access}} = \frac{\text{Avg Cycles}}{\text{Hit}} + \left(\frac{\text{Num Misses}}{\text{Num Accesses}} \times \frac{\text{Avg Extra Cycles}}{\text{Miss}} \right)$$

- Mem access / sequence depends on program and translation
- Time / cycle depends on microarchitecture and implementation

- Also called the average memory access latency (AMAL)
- Avg cycles / hit is called the hit latency
- Number of misses / number of accesses is called the miss rate
- Avg extra cycles / miss is called the miss penalty

- Avg cycles per hit depends on microarchitecture
- Miss rate depends on microarchitecture
- Miss penalty depends on microarchitecture, rest of memory system



Microarchitecture	Hit Latency	Extra Accesses for Translation
FSM Cache	>1	1+
Pipelined Cache	≈1	1+
Pipelined Cache + TLB	≈1	≈0

4.1. Estimating AMAL

Consider the following sequence of memory acceses which might correspond to copying 4 B elements from a source array to a destination array. Each array contains 64 elements. Assume two-way set associative cache with 16 B cache lines, hit latency of 1 cycle and 10 cycle miss penalty. What is the AMAL in cycles?

```
rd 0x1000
wr 0x2000
rd 0x1004
wr 0x2004
rd 0x1008
wr 0x2008
...
rd 0x1040
wr 0x2040
```

Consider the following sequence of memory acceses which might correspond to incrementing 4 B elements in an array. The array contains 64 elements. Assume two-way set associative cache with 16 B cache lines, hit latency of 1 cycle and 10 cycle miss penalty. What is the AMAL in cycles?

```
rd 0x1000
wr 0x1000
rd 0x1004
wr 0x1004
rd 0x1008
wr 0x1008
...
rd 0x1040
wr 0x1040
```