EECS 370

Making Virtual Memory Fast

Virtual Memory Performance

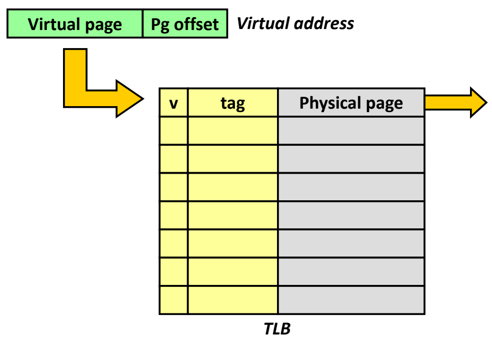
- To translate a virtual address into a physical address, we must first access the page table in physical memory
  - If it's an N-level page table, we must do N total loads before getting the physical page number
- Then we access physical memory again to get the data
  - A load instruction performs at least 2 memory reads
  - A store instruction performs at least 1 read and then a write
- Above lookups are **SLOW**

Translation look-aside buffer (TLB)

- We fix this performance problem by avoiding main memory in the translation from virtual to physical pages.
- Buffer common translations in a **Translation Look-aside Buffer (TLB)**, a fast cache memory dedicated to storing a small subset of valid V-to-P translations.
- 16-512 entries common.
- Generally has low miss rate (< 1%).

Poll: Why can we make TLBs smaller than the cache hierarchy?
a) Addresses are smaller than data
b) TLB is accessed less frequently than caches
c) Only need to store info about individual pages

Translation look-aside buffer (TLB)



Putting it all together

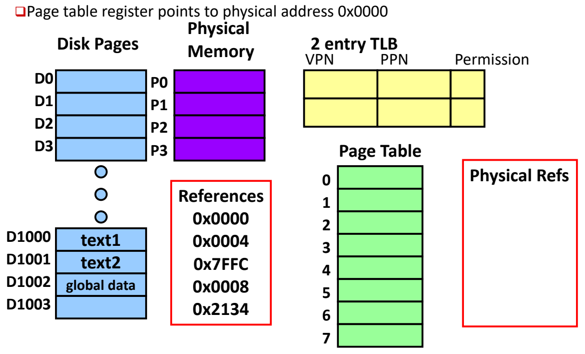
- Loading your program in memory
  - Ask operating system to create a new process
  - Construct a page table for this process
  - Mark all page table entries as invalid with a pointer to the disk image of the program
    - That is, point to the executable file containing the binary.
  - Run the program and get an immediate page fault on the first instruction.

Agenda

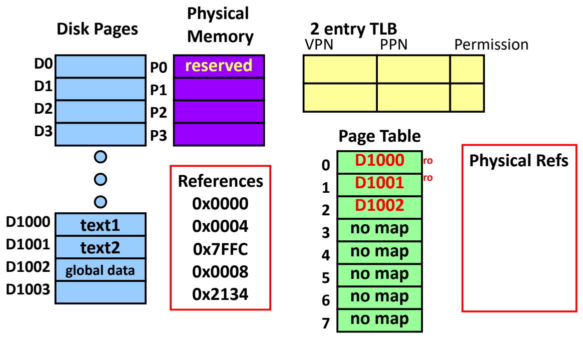
- Table Look-aside Buffers (TLB)
- **Virtual Memory Walkthrough**
- Cache Placement

Loading a program into memory

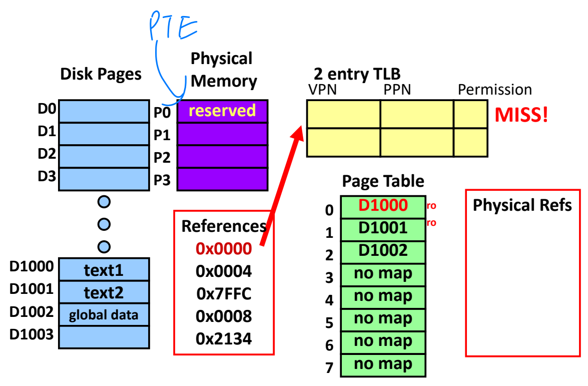
- Page size = 4 KB, Page table entry size = 4 B
- Page table register points to physical address 0x0000



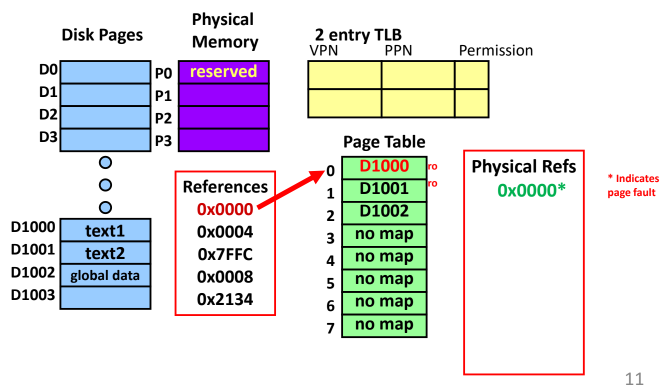
Step 1: Read executable header & initialize page table



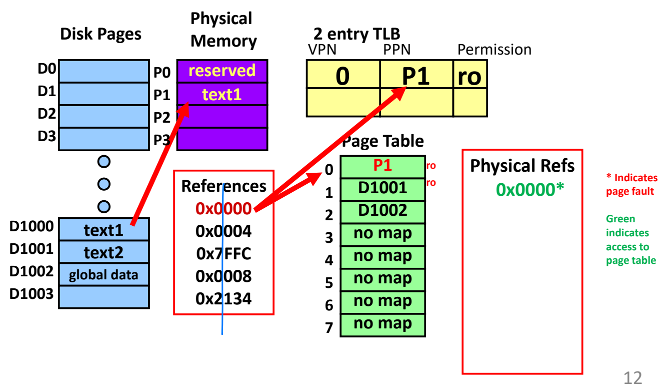
Step 2: Load PC from header & start execution



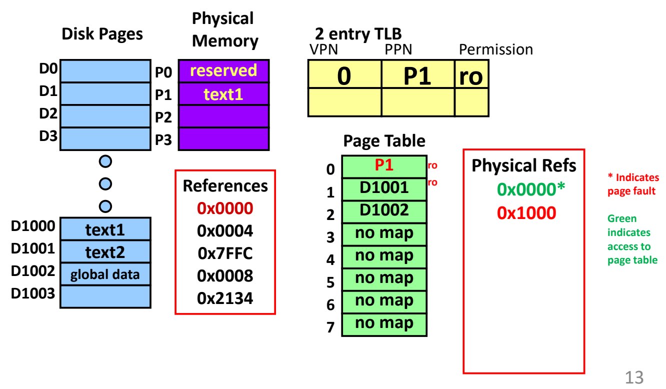
Fetching instruction 0000



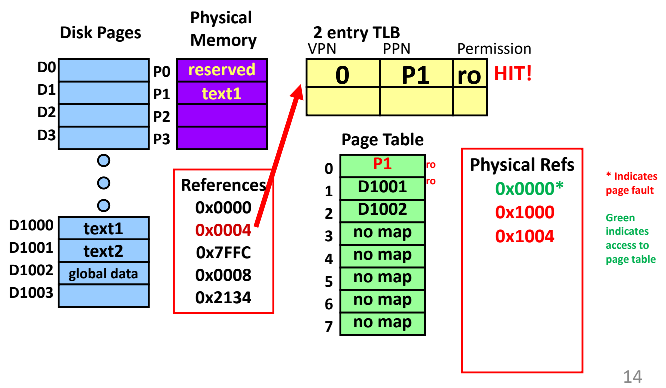
Fetching instruction 0000



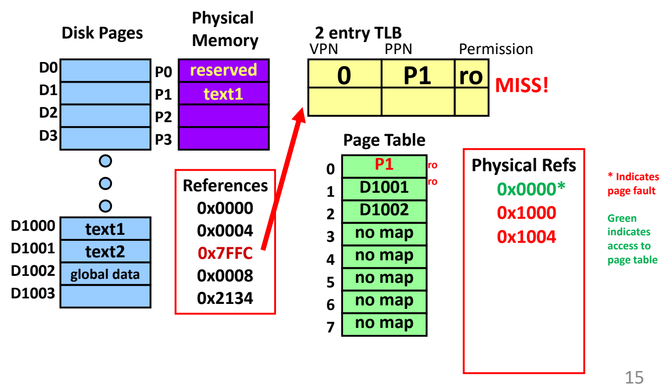
Fetching instruction 0000



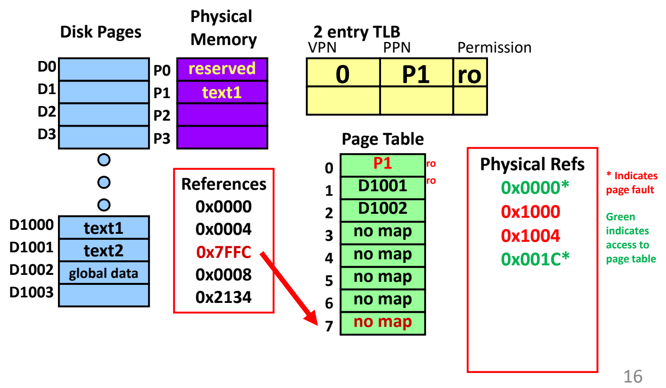
Fetching instruction 0004



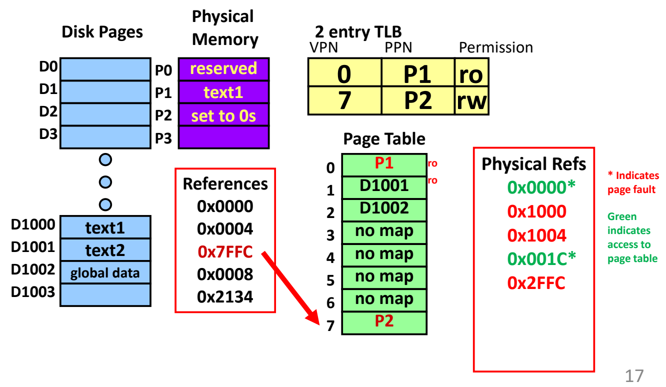
Reference 7FFC



Reference 7FFC



Reference 7FFC



Fetching instruction 0008

Disk Pages	
D0	
D1	
D2	
D3	
D1000	text1
D1001	text2
D1002	global data
D1003	

Physical Memory	
P0	reserved
P1	text1
P2	set to 0s
P3	

2 entry TLB

VPN	PPN	Permission	
0	P1	ro	HIT!
7	P2	rw	

Page Table

0	P1	ro
1	D1001	ro
2	D1002	
3	no map	
4	no map	
5	no map	
6	no map	
7	P2	

References
0x0000
0x0004
0x7FFC
0x0008
0x2134

Physical Refs
0x0000*
0x1000
0x1004
0x001C*
0x2FFC
0x1008

* Indicates page fault

Green indicates access to page table

Reference 2134

Disk Pages	
D0	
D1	
D2	
D3	
D1000	text1
D1001	text2
D1002	global data
D1003	

Physical Memory	
P0	reserved
P1	text1
P2	set to 0s
P3	

2 entry TLB

VPN	PPN	Permission	
0	P1	ro	MISS!
7	P2	rw	

Page Table

0	P1	ro
1	D1001	ro
2	D1002	
3	no map	
4	no map	
5	no map	
6	no map	
7	P2	

References
0x0000
0x0004
0x7FFC
0x0008
0x2134

Physical Refs
0x0000*
0x1000
0x1004
0x001C*
0x2FFC
0x1008

* Indicates page fault

Green indicates access to page table

Reference 2134

Disk Pages	
D0	
D1	
D2	
D3	
D1000	text1
D1001	text2
D1002	global data
D1003	

Physical Memory	
P0	reserved
P1	text1
P2	set to 0s
P3	

2 entry TLB

VPN	PPN	Permission
0	P1	ro
7	P2	rw

Page Table

0	P1	ro
1	D1001	ro
2	D1002	
3	no map	
4	no map	
5	no map	
6	no map	
7	P2	

References
0x0000
0x0004
0x7FFC
0x0008
0x2134

Physical Refs
0x0000*
0x1000
0x1004
0x001C*
0x2FFC
0x1008
0x0008*
0x3134

* Indicates page fault

Green indicates access to page table

Reference 2134

Disk Pages	
D0	
D1	
D2	
D3	
D1000	text1
D1001	text2
D1002	global data
D1003	

Physical Memory	
P0	reserved
P1	text1
P2	set to 0s
P3	global data

2 entry TLB

VPN	PPN	Permission
0	P1	ro
2	P3	rw

Page Table

0	P1	ro
1	D1001	ro
2	P3	
3	no map	
4	no map	
5	no map	
6	no map	
7	P2	

Virt Addr
0x0000
0x0004
0x7FFC
0x0008
0x2134

Physical Refs
0x0000*
0x1000
0x1004
0x001C*
0x2FFC
0x1008
0x0008*
0x3134

* Indicates page fault

Green indicates access to page table

Agenda

- Table Look-aside Buffers (TLB)
- Virtual Memory Walkthrough
- **Cache Placement**

Next topic: Placing Caches in a VM System

- VM systems give us two different addresses:
  - virtual and physical
- Which address should we use to access the data cache?
  - Physical address (after VM translations).
    - We have to wait for the translation; slower.
  - Virtual address (before VM translation).
    - Faster access.
    - More complex.

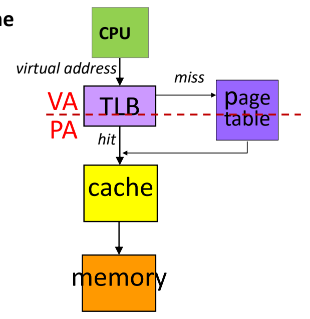
Poll: Which would be faster to access?
a) Address cache with virtual address
b) Address cache with physical address

Cache & VM Organization: Option 1

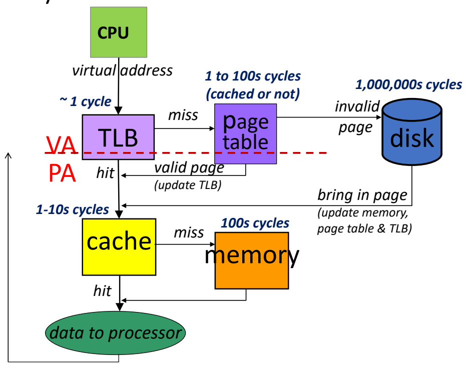
Physically-addressed Cache

✗ Slower

✓ Low complexity



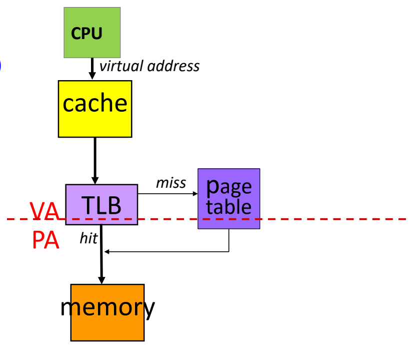
Physically addressed caches: detailed flow



Cache & VM Organization: option 2

Virtually-addressed Cache

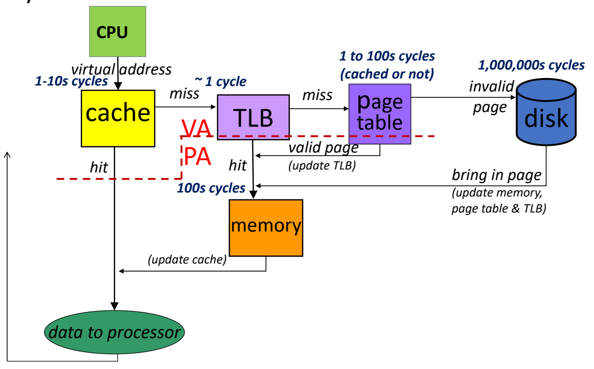
- ❌ High complexity (aliasing)
- ✓ Faster



Virtually addressed caches

- Cache uses bits from the virtual address to access cache (tag, set index, and block offset)
- Perform the TLB only if the cache gets a miss.
- Problems:
  - Aliasing: Two processes may refer to the same physical location with different virtual addresses (synonyms)
  - Two processes may have same virtual addresses with different physical addresses (homonyms)
  - When switching processes, TLB must be invalidated, dirty cache blocks must be written back to memory, and cache must be invalidated to solve homonym problem

Virtually addressed caches: detailed flow



OS Support for Virtual Memory

- It must be able to modify the page table register, update page table values, etc.
- To enable the OS to do this, **BUT** not the user program, we have different execution modes for a process.
  - Executive (or supervisor or kernel level) permissions and
  - User level permissions.

Exercise using previous multi-level VM

0x 0000 000|0 0000 1111|0000 1100

12

Virtual Address	1st level	2nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0	7	0x10C	Y	0	010C
0x001F0C						
0x020F0C						

1st level = 7b	2nd level = 8b	Page offset = 9b

Virtual address = 24b

Physical page number = 9b	Page offset = 9b

Physical address = 18b

Assume memory for page tables are "somewhere else" in memory

Exercise using previous multi-level VM

Virtual Address	1st level	2nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C						
0x020F0C						

1st level = 7b	2nd level = 8b	Page offset = 9b

Virtual address = 24b

Physical page number = 9b	Page offset = 9b

Physical address = 18b

Assume memory for page tables are "somewhere else" in memory

Exercise using previous multi-level VM

Virtual Address	1st level	2nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C	0x00	0x0F	0x10C	Y	0x001	0x0030C
0x020F0C						

1st level = 7b	2nd level = 8b	Page offset = 9b

Virtual address = 24b

Physical page number = 9b	Page offset = 9b

Physical address = 18b

Assume memory for page tables are "somewhere else" in memory

Exercise using previous multi-level VM

Virtual Address	1st level	2nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C	0x00	0x0F	0x10C	Y	0x001	0x0030C
0x020F0C	0x01	0x07	0x10C	Y	0x002	0x0050C

1st level = 7b	2nd level = 8b	Page offset = 9b

Virtual address = 24b

Physical page number = 9b	Page offset = 9b

Physical address = 18b

Assume memory for page tables are "somewhere else" in memory