```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.
```

Lecture 34

Virtual Memory II

CS 61C, Fall 2024 @ UC Berkeley

Slides credit: Dan Garcia, Borivoje Nikolic, Lisa Yan, Peyrin Kao

Slides template credit: Josh Hug, Lisa Yan

TLB: Caching Address Lookups

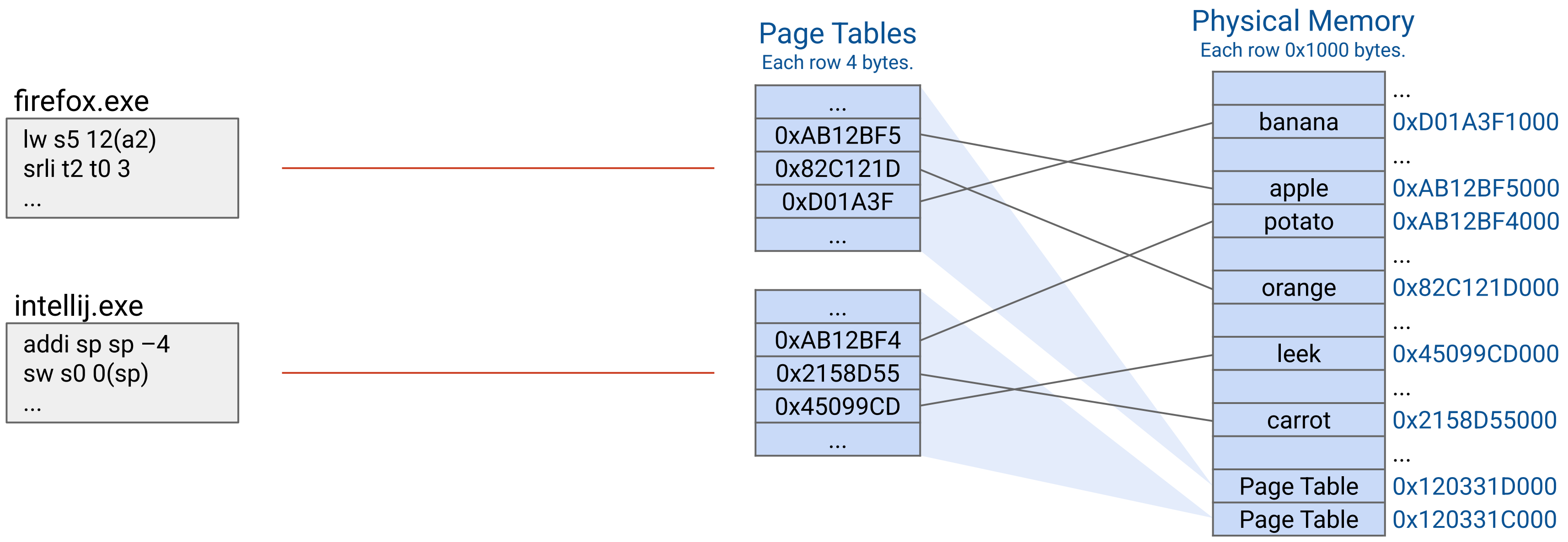
Lecture 34, CS 61C, Fall 2024

Review: Page Tables Live in Memory

From last time: Just like any other data, **page tables live in memory**.

This means that every load/store requires *two* memory accesses:

- First, access the page table to learn the physical address.
- Then, access the actual data at that physical address.



```
lw s5 12(a2)
srli t2 t0 3
...
```

```
addi sp sp −4
sw s0 0(sp)
...
```

Review: Steps of Address Translation

Step 1: The program wants to access memory at a given virtual address.

Step 2: Extract VPN and offset.

- Use parameters to decide # of VPN bits and # of offset bits.

Step 3: Translate VPN to PPN using table. ← We call this a *page table walk*.

- Look up the VPN'th index in the table to find the corresponding PPN.
- If page fault (page table entry is invalid), load the page from disk.

Step 4: Concatenate PPN and offset.

Step 5: Return data to the program.

Steps 3 and 5 both require accessing memory. (Analogy: Going to Sacramento.)

- Can we avoid accessing memory so often? Yes – caching!

TLB: Caching Address Lookups

The page table stores VPN/PPN mappings in memory. (Analogy: Sacramento.)

The **TLB** is a cache of VPN/PPN mappings. (Analogy: UC Berkeley campus.)

I just remember the last 3 VPN/PPN mappings I've seen.

The TLB (Cache)	
VPN	**PPN**
0x00004	0x60C25E6
0x00005	0x71DB139
0x00009	0x45099CD

I have every VPN/PPN mapping, from VPN 0x00000 to 0xFFFFF.

firefox.exe's Page Table (Memory)	
VPN	**PPN**
...	...
0x00004	0x60C25E6
0x00005	0x71DB139
0x00006	0xEC70DB7
0x00007	0xAB12BF4
0x00008	0x2158D55
0x00009	0x45099CD
...	...

TLB stands for "Translation Lookaside Buffer."
But it's really just a fancy name for a cache.

TLB Design

Technical details:

- 38–128 entries.
- Fully-associative.
- Replacement policy is FIFO or random.

Instead of thinking about associativity, tag/index/offset, etc., it's probably easiest to just think of the TLB as "the last few VPN/PPN mappings I've seen."

I just remember the last 3 VPN/PPN mappings I've seen.

The TLB (Cache)

VPN	PPN
0x00004	0x60C25E6
0x00005	0x71DB139
0x00009	0x45099CD

TLB Reach

TLB Reach tells us how many addresses can get immediately translated by the TLB (bypassing the page table in memory.)

$$\text{TLB Reach} = \#\text{ of TLB entries} \times \text{Page size}$$

I support instant lookup for up to $3 \times 2^{12} = 12288$ addresses.

The TLB (Cache)	
VPN	**PPN**
0x00004	0x60C25E6
0x00005	0x71DB139
0x00009	0x45099CD

Assuming 12-byte offset $= 2^{12}$-byte pages.

TLB Flushing

There's just one TLB on the computer. (Assuming no parallelism.)

By contrast, VPN/PPN mappings are specific to a program.

- firefox.exe and intellij.exe each have a virtual page numbered **0x00004**. Those virtual pages probably map to different physical pages.
- When the OS *context-switches* to run a different program, the OS must *flush* the TLB and invalidate all of its entries.

Flush me if you switch processes.

The TLB (Cache)	
VPN	**PPN**
0x00004	0x60C25E6
0x00005	0x71DB139
0x00009	0x45099CD

This says: firefox.exe's VPN 4 maps to PPN 0x60C25E6.

But if we switch to running intellij.exe, this entry isn't valid.

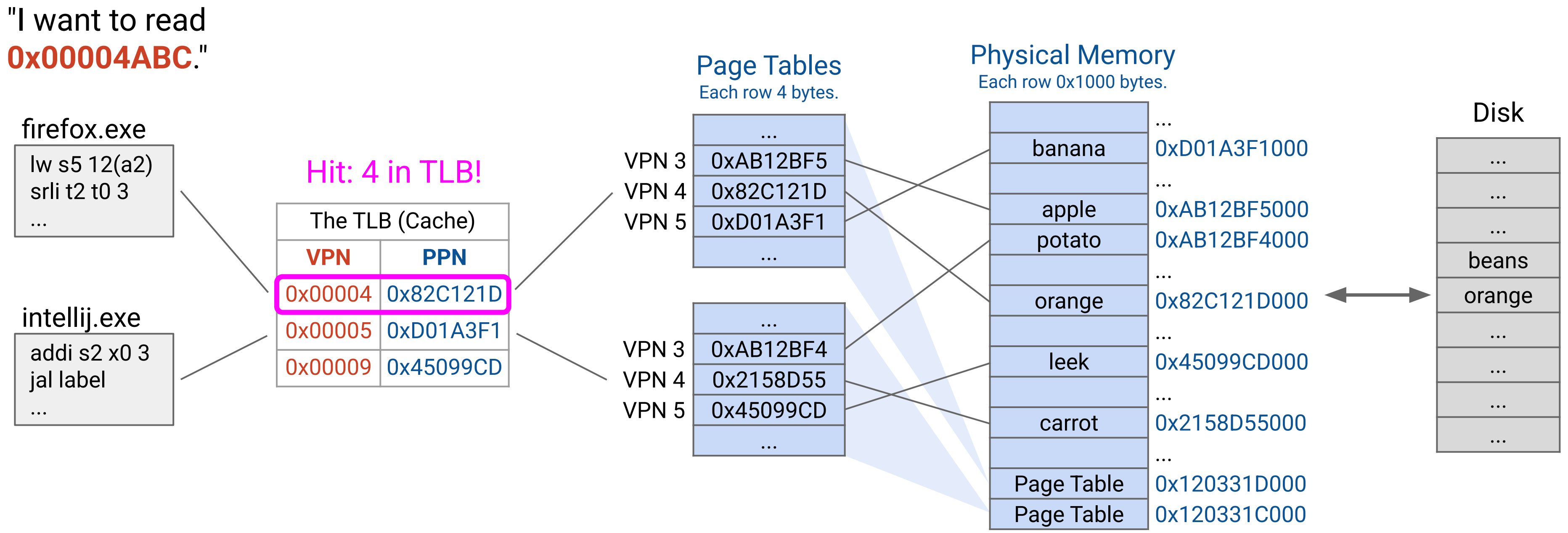
Address Translation Hit/Miss Cases

Lecture 34, CS 61C, Fall 2024

TLB Hits, Page Table Walk, Page Fault

Best case: TLB hit.

- Instant (~1 clock cycle) address translation! (Analogy: UC Berkeley campus.)

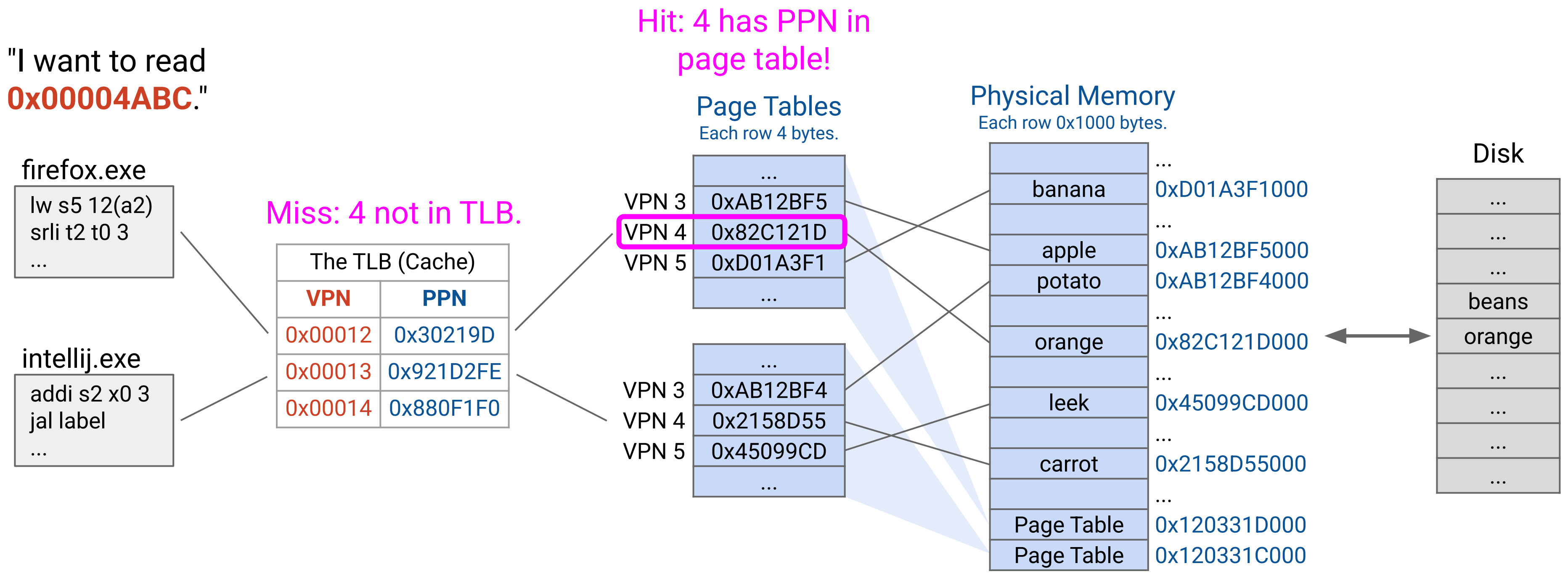


The TLB (Cache)	
VPN	PPN
0x00004	0x82C121D
0x00005	0xD01A3F1
0x00009	0x45099CD

TLB Hits, Page Table Walk, Page Fault

Worse case: TLB miss, page table walk.

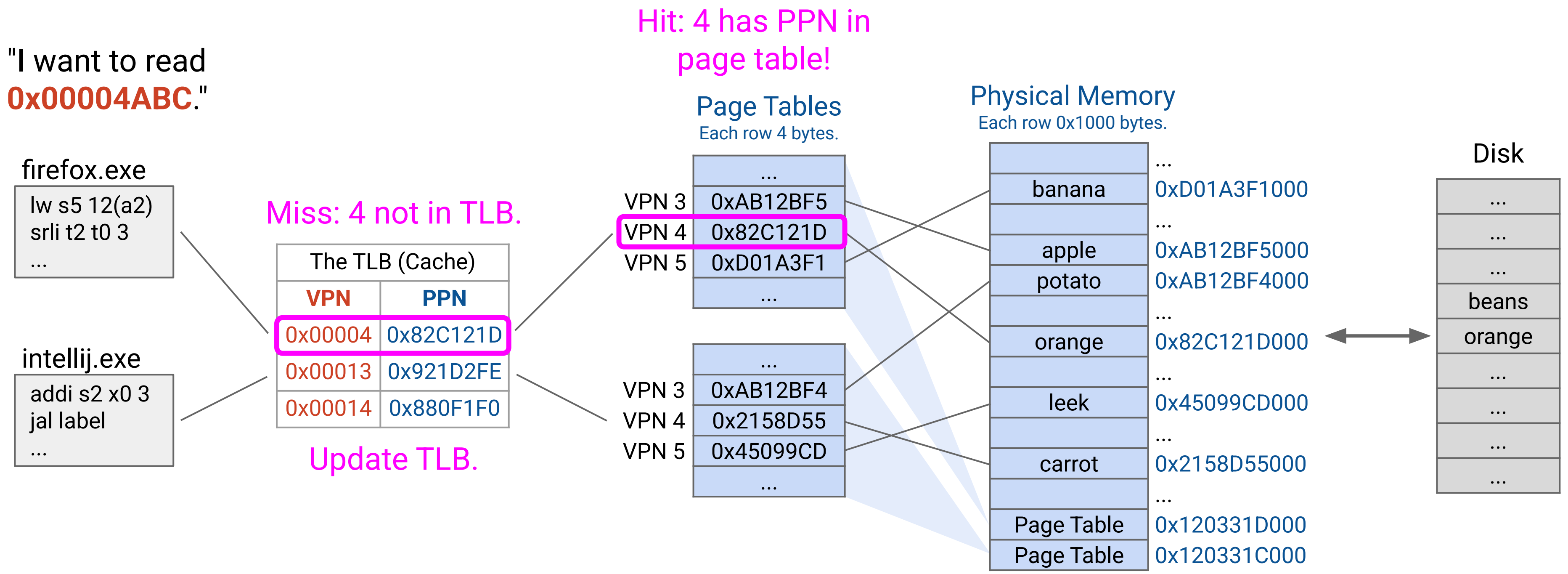
- Go to main memory to read page table. (Analogy: Sacramento.)



TLB Hits, Page Table Walk, Page Fault

Worse case: TLB miss, page table walk.

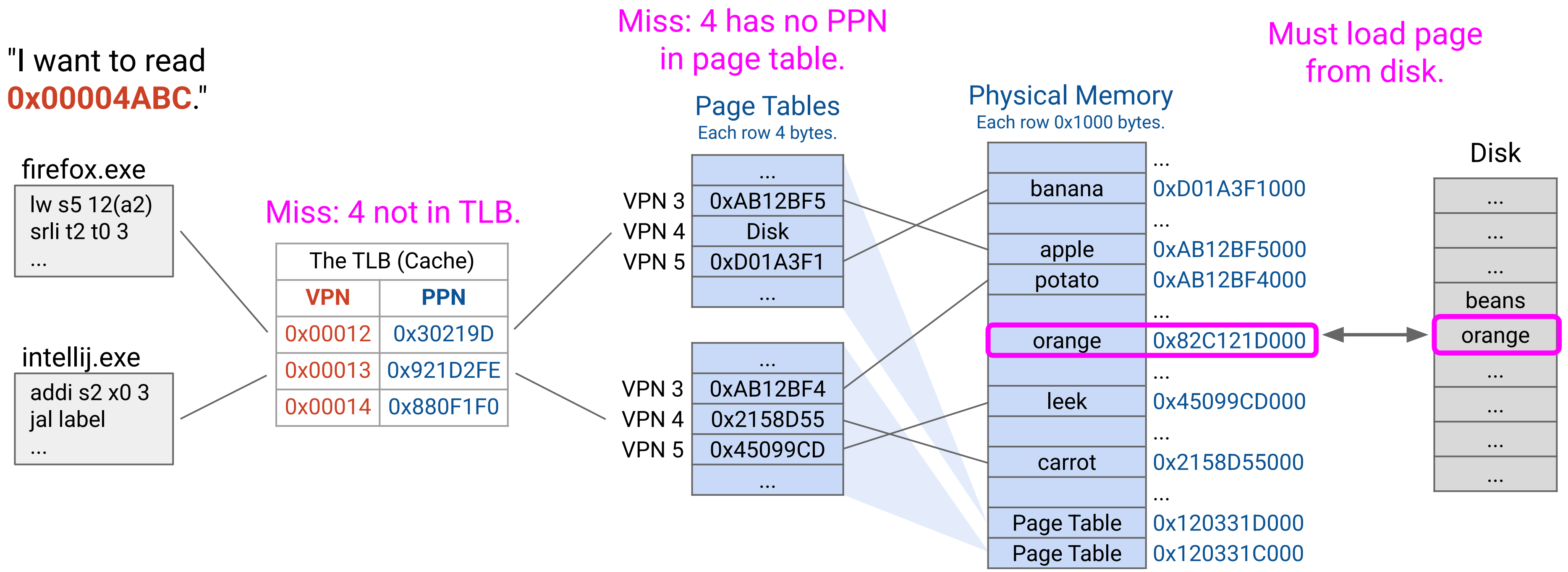
- Go to main memory to read page table. (Analogy: Sacramento.)
- Update TLB so we have this VPN/PPN mapping handy for next time.



TLB Hits, Page Table Walk, Page Fault

Worst case: TLB miss, page fault, go to disk.

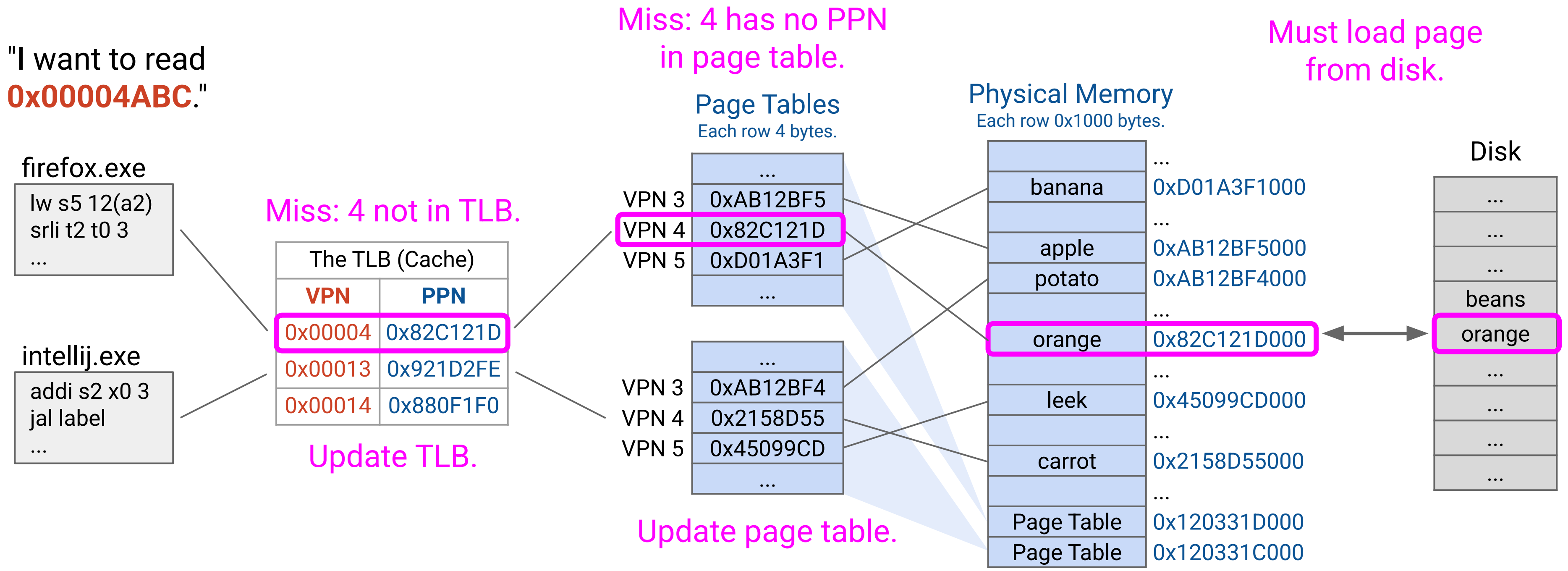
- Go to disk to load page. (Analogy: Pluto.)



TLB Hits, Page Table Walk, Page Fault

Worst case: TLB miss, page fault, go to disk.

- Go to disk to load page. (Analogy: Pluto.)
- Page fault causes page table to update with PPN of the newly-loaded page.
- Update TLB so we have this VPN/PPN mapping handy for next time.



TLB Hits, Page Table Walk, Page Fault

Three cases for an address translation:

	TLB hit?	Page table entry valid?
Best: TLB Hit	✔ Hit	Never visited
Worse: TLB Miss, Page Table Walk	✘ Miss	✔ Valid
Worst: TLB Miss, Page Fault	✘ Miss	✘ Invalid

The TLB remembers recent page table entries, so if the entry is in the TLB, it must also be in the page table.

If the TLB hits, though, we won't visit the page table.

Caches vs. Virtual Memory

Lecture 34, CS 61C, Fall 2024

Units of Memory

There are many different units of memory:

- **Byte**: Smallest addressable unit of memory.
- **Word**: Size of an address. *(4 bytes or 8 bytes.)*
- **Blocks**: Caches read one block of memory at a time. *(64 bytes.)*
- **Pages**: Virtual memory reads one page of memory at a time. *(4096 bytes.)*

All of these are just units of measurement. *(Analogy: kilometer, meter, centimeter.)*

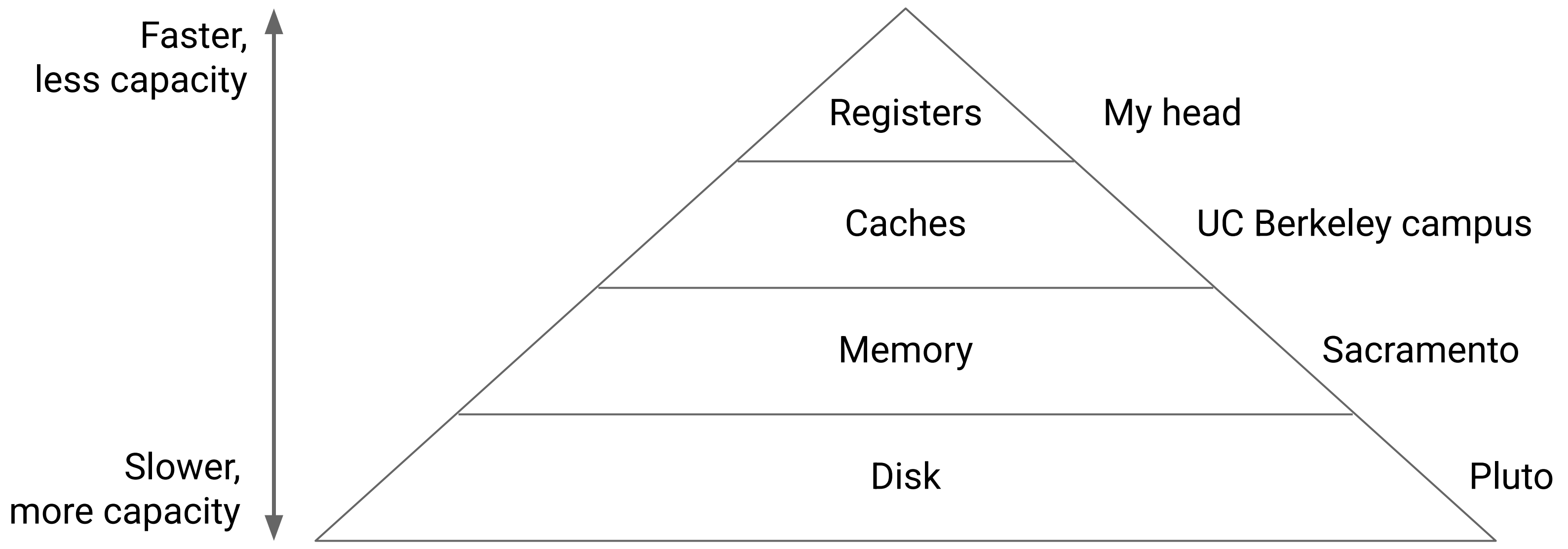
Caches and VM both deal in chunks of memory: Blocks or pages.

(Number in parentheses = size on modern systems.)

Cache Paradigm

Cache paradigm: Data at each level is a "quick-access" copy of data at a lower level.

- Cache: "I remember recently-accessed blocks from memory."
- Memory: "I remember recently-accessed pages from disk."

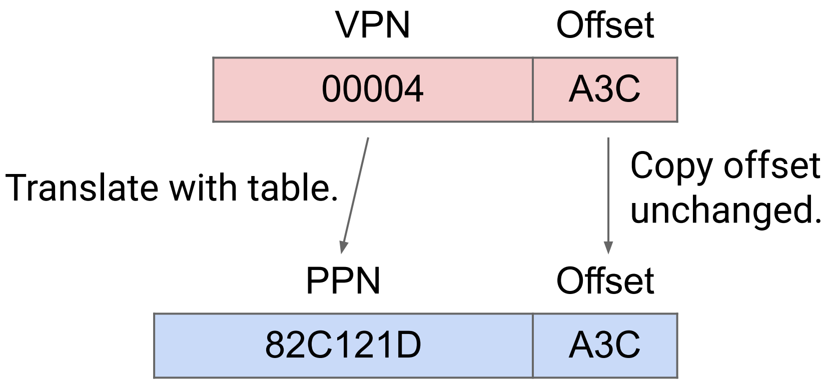


Caches vs. Virtual Memory

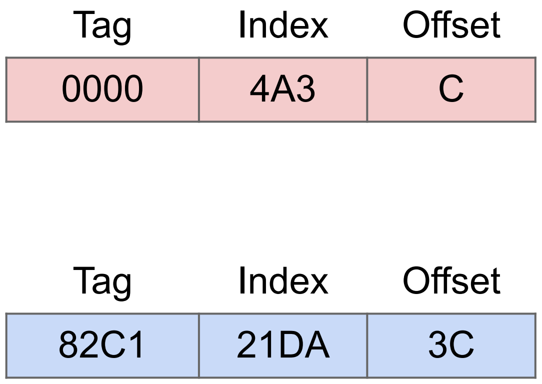
	Caches	**Virtual Memory**
Memory hierarchy position:	Caches ↔ Memory	Memory ↔ Disk
Memory unit:	Block (~64 bytes)	Page (~4096 bytes)
Miss:	Cache Miss	Page Fault
Associativity:	Direct-mapped, N-way set associative, fully associative	Fully associative (pages can go anywhere in memory)
Replacement policy:	Least-recently-used (LRU) or random	LRU (most common), FIFO, or random
Write policy:	Write-through or write-back	Write-back

Splitting Addresses

Caches and VMs split address in different, totally unrelated ways.



VM: Page number, offset.



Caches: Tag, index, offset.

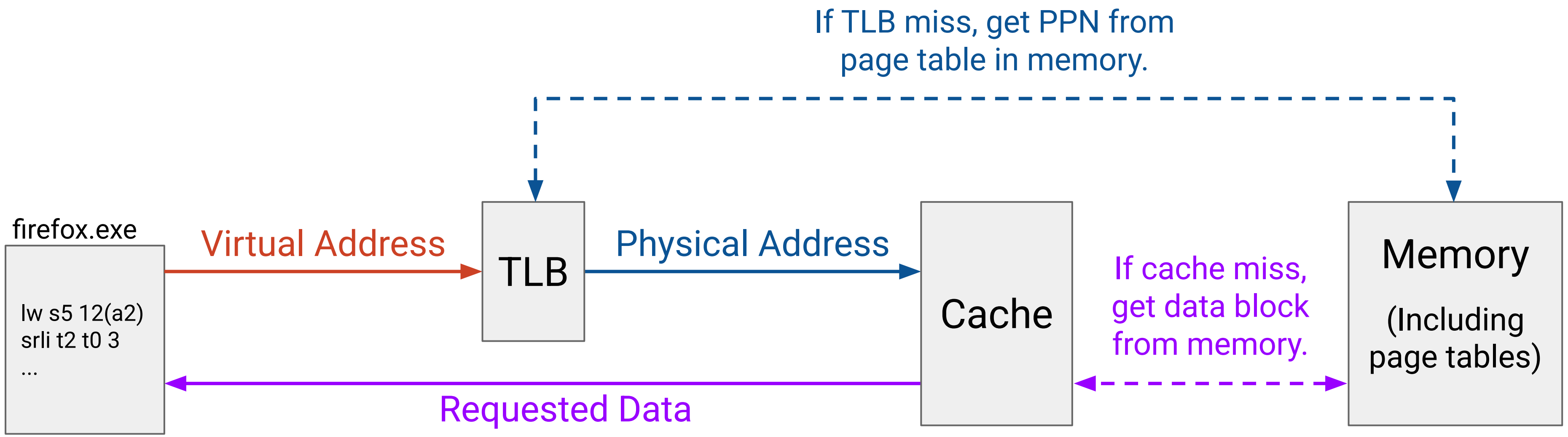
We could do TIO breakdowns on either physical or virtual addresses. More on the next slide.

Putting Caches and VM Together

Physically Indexed, Physically Tagged (PIPT) caches:

- Translate address first.
- Then use the physical address to access the cache.

Other, more complicated designs exist.



Optional: Multi-Level Page Tables

Lecture 34, CS 61C, Fall 2024

Why Is This Optional?

Multi-level page tables used to be in 61C (pre-2023), but is optional now.

Trying to understand this section without the fundamentals will confuse you further.

- If the rest of VM doesn't make sense to you, we suggest coming back to this later, after digesting the rest of the unit.

You'll (probably) see this topic if you take CS 162 (Operating Systems) at UC Berkeley.

- TAs report that it's useful to at least see this topic once before then.

Recall: Page Table Size

Recall: We can compute the approximate size of the page table as follows:

Size of page table = # of PTEs × Size per PTE

$2^{\text{\# of VPN bits}}$

Because we need one entry per VPN, and this is the number of VPNs.

# of PPN bits

Because each entry is just a PPN value.

Measured in bytes, e.g. 32 bits = 4 bytes.

Problem: Page Tables Are Memory-Intensive

Page tables take up a lot of space in memory. Example parameters:

● 4 KiB pages	=	12-bit offset.
● 4 GiB virtual address space	=	32-bit virtual address.
● 4 GiB physical address space	=	32-bit physical address.

Estimating page table size:

- # of PPN bits = 32 – 12 = 20 bits ≈ 4 bytes. (Rounding up to include status bits.)
- # of VPN bits = 32 – 12 = 20 bits.
- Page table size = 2^{20} PTEs × 4 bytes per PTE = 4 MiB.

If we have 256 processes (each needs its own page table):

- 256 × 4 MiB = 1 GiB.
- That's 25% of physical memory used just to store the page tables!

Problem: Page Tables Are Memory-Intensive

Page tables on 64-bit systems take up even more space in memory.

Unrelated note: 64-bit systems usually only use 48-bit virtual address spaces.

- Top 16 bits of every address are all zeroes.
- Because 2^{48} bytes = 256 TiB is enough for almost any program.

Problem: Page Tables Are Memory-Intensive

Page tables take up a lot of space in memory. Example parameters:

● 4 KiB pages	=	12-bit offset.
● 256 TiB virtual address space	=	48-bit virtual address.
● 4 GiB physical address space	=	32-bit physical address.

Estimating page table size:

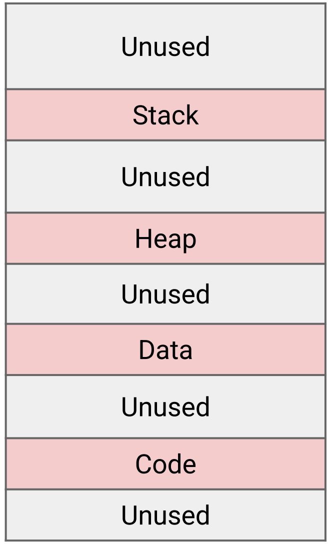
- # of PPN bits = 32 – 12 = 20 bits ≈ 4 bytes.
- # of VPN bits = 48 – 12 = 36 bits.
- Page table size = 2^{36} PTEs × 4 bytes per PTE = 256 GiB.

A single page table doesn't even fit in physical memory!

Page Tables Are Sparse

Recall: Small programs leave most memory unused.

- This means most VPNs are never even accessed.
- The vast majority of the page table is invalid entries.
- We're wasting tons of memory storing invalid PTEs.



firefox.exe's Page Table

Valid?	VPN	PPN
0	0	garbage
0	1	garbage
0	2	garbage
0	3	garbage
0	4	garbage
0	5	garbage
0	6	garbage
0	7	garbage
0	8	garbage
0	9	garbage
0	10	garbage
0	11	garbage
0	12	garbage
1	13	0x45099CD
0	14	garbage
0	15	garbage

Solution: Multi-Level Page Tables

Level 1 page table maps *ranges of VPNs* to Level 2 page tables.

Level 2 page table maps individual VPNs to PPNs.

Level 1 Page Table

Valid?	VPNs	L2 Page Table
0	0~3	garbage
0	4~7	garbage
0	8~11	garbage
1	12~15	0x8103D32

Benefit: A single Level 1 PTE can indicate that a range of VPNs are all invalid.

This page table lives at page number 0x8103D32 in physical memory.

Level 2 Page Table

Valid?	VPN	PPN
0	12	garbage
1	13	0x45099CD
0	14	garbage
0	15	garbage

Multi-Level Page Table Example

Consider a 16-bit VPN.

- 0x10000 = 2^{16} page table entries (PTEs). 0x0000 ~ 0xFFFF.
- The vast majority are invalid.
- Suppose only 3 are valid.

firefox.exe's Page Table

Valid?	VPN	PPN
0	0x0000	garbage
0	0x0001	garbage
1	0x0002	0x8932D13
1	0x0003	0x903B9CD
0	0x0004	garbage
0	0x0005	garbage
0	0x0006	garbage
0	0x0007	garbage
0	0x0008	garbage
...	...	...
0	0xFFFA	garbage
0	0xFFFB	garbage
0	0xFFFC	garbage
1	0xFFFD	0x45099CD
0	0xFFFE	garbage
0	0xFFFF	garbage

Multi-Level Page Table Example

L1 page table maps *ranges of VPNs* to L2 page tables.

L2 page table maps individual VPNs to PPNs.

Level 1 Page Table

Valid?	VPNs	L2 Page Table
1	0x0000 ~ 0x00FF	0x8103D32
0	0x0100 ~ 0x01FF	garbage
0	0x0200 ~ 0x02FF	garbage
0	0x0300 ~ 0x03FF	garbage
0	0x0400 ~ 0x04FF	garbage
...	...	...
0	0xFD00 ~ 0xFDFF	garbage
0	0xFE00 ~ 0xFEFF	garbage
1	0xFF00 ~ 0xFFFF	0x12DB93A

Page table living at PPN 0x8103D32.

Level 2 Page Table

Valid?	VPN	PPN
0	0x0000	garbage
0	0x0001	garbage
1	0x0002	0x8932D13
1	0x0003	0x903B9CD
0	0x0004	garbage
...	...	...
0	0x00FF	garbage

Page table living at PPN 0x12DB93A.

Level 2 Page Table

Valid?	VPN	PPN
0	0xFF00	garbage
0	0xFF01	garbage
...	...	...
0	0xFFFC	garbage
1	0xFFFD	0x45099CD
0	0x00FE	garbage
0	0x00FF	garbage

Multi-Level Page Table Example

Top 8 bits (2 hex digits) indexes into the L1 page table.

Bottom 8 bits indexes into the L2 page table.

Level 1 Page Table

Valid?	VPNs	L2 Page Table
1	0x**00**00 ~ 0x00FF	0x8103D32
0	0x**01**00 ~ 0x01FF	garbage
0	0x**02**00 ~ 0x02FF	garbage
0	0x**03**00 ~ 0x03FF	garbage
0	0x**04**00 ~ 0x04FF	garbage
...	...	...
0	0x**FD**00 ~ 0xFDFF	garbage
0	0x**FE**00 ~ 0xFEFF	garbage
1	0x**FF**00 ~ 0xFFFF	0x12DB93A

Page table living at PPN 0x8103D32.

Level 2 Page Table

Valid?	VPN	PPN
0	0x00**00**	garbage
0	0x00**01**	garbage
1	0x00**02**	0x8932D13
1	0x00**03**	0x903B9CD
0	0x00**04**	garbage
...	...	...
0	0x00**FF**	garbage

Page table living at PPN 0x12DB93A.

Level 2 Page Table

Valid?	VPN	PPN
0	0xFF**00**	garbage
0	0xFF**01**	garbage
...	...	...
0	0xFF**FC**	garbage
1	0xFF**FD**	0x45099CD
0	0x00**FE**	garbage
0	0x00**FF**	garbage

Multi-Level Page Table Example

Example: Translate VPN **0x0391**.

Level 1 Page Table

Valid?	VPNs	L2 Page Table
1	0x**00**00 ~ 0x00FF	0x8103D32
0	0x**01**00 ~ 0x01FF	garbage
0	0x**02**00 ~ 0x02FF	garbage
0	0x**03**00 ~ 0x03FF	garbage
0	0x**04**00 ~ 0x04FF	garbage
...	...	...
0	0x**FD**00 ~ 0xFDFF	garbage
0	0x**FE**00 ~ 0xFEFF	garbage
1	0x**FF**00 ~ 0xFFFF	0x12DB93A

All VPNs starting with 0x03 don't exist, so page fault right away.

Page table living at PPN 0x8103D32.

Level 2 Page Table

Valid?	VPN	PPN
0	0x00**00**	garbage
0	0x00**01**	garbage
1	0x00**02**	0x8932D13
1	0x00**03**	0x903B9CD
0	0x00**04**	garbage
...	...	...
0	0x00**FF**	garbage

Page table living at PPN 0x12DB93A.

Level 2 Page Table

Valid?	VPN	PPN
0	0xFF**00**	garbage
0	0xFF**01**	garbage
...	...	...
0	0xFF**FC**	garbage
1	0xFF**FD**	0x45099CD
0	0x00**FE**	garbage
0	0x00**FF**	garbage

Multi-Level Page Table Example

Example: Translate VPN **0x0001**.

Level 1 Page Table

Valid?	VPNs	L2 Page Table
1	0x0000 ~ 0x00FF	0x8103D32
0	0x0100 ~ 0x01FF	garbage
0	0x0200 ~ 0x02FF	garbage
0	0x0300 ~ 0x03FF	garbage
0	0x0400 ~ 0x04FF	garbage
...	...	...
0	0xFD00 ~ 0xFDFF	garbage
0	0xFE00 ~ 0xFEFF	garbage
1	0xFF00 ~ 0xFFFF	0x12DB93A

L1 page table points us to an L2 page table for VPNs starting with 0x00.

Garbage at L2 page table. Page fault.

Page table living at PPN 0x8103D32.

Level 2 Page Table

Valid?	VPN	PPN
0	0x0000	garbage
0	0x0001	garbage
1	0x0002	0x8932D13
1	0x0003	0x903B9CD
0	0x0004	garbage
...	...	...
0	0x00FF	garbage

Page table living at PPN 0x12DB93A.

Level 2 Page Table

Valid?	VPN	PPN
0	0xFF00	garbage
0	0xFF01	garbage
...	...	...
0	0xFFFC	garbage
1	0xFFFD	0x45099CD
0	0x00FE	garbage
0	0x00FF	garbage

Multi-Level Page Table Example

Example: Translate VPN **0xFFFD**.

Level 1 Page Table

Valid?	VPNs	L2 Page Table
1	0x**00**00 ~ 0x00FF	0x8103D32
0	0x**01**00 ~ 0x01FF	garbage
0	0x**02**00 ~ 0x02FF	garbage
0	0x**03**00 ~ 0x03FF	garbage
0	0x**04**00 ~ 0x04FF	garbage
...	...	...
0	0x**FD**00 ~ 0xFDFF	garbage
0	0x**FE**00 ~ 0xFEFF	garbage
1	0x**FF**00 ~ 0xFFFF	0x12DB93A

L1 page table points us to an L2 page table for VPNs starting with 0xFF.

Page table living at PPN 0x8103D32.

Level 2 Page Table

Valid?	VPN	PPN
0	0x00**00**	garbage
0	0x00**01**	garbage
1	0x00**02**	0x8932D13
1	0x00**03**	0x903B9CD
0	0x00**04**	garbage
...	...	...
0	0x00**FF**	garbage

Page table living at PPN 0x12DB93A.

Level 2 Page Table

Valid?	VPN	PPN
0	0xFF**00**	garbage
0	0xFF**01**	garbage
...	...	...
0	0xFF**FC**	garbage
1	0xFF**FD**	0x45099CD
0	0x00**FE**	garbage
0	0x00**FF**	garbage

Found PPN in L2 page table!

Multi-Level Page Table Example

3 page tables. Each has 0x100 entries (0x00 ~ 0xFF).

Total of 0x300 entries.

Level 1 Page Table		
Valid?	**VPNs**	**L2 Page Table**
1	0x**00**00 ~ 0x00FF	0x8103D32
0	0x**01**00 ~ 0x01FF	garbage
0	0x**02**00 ~ 0x02FF	garbage
0	0x**03**00 ~ 0x03FF	garbage
0	0x**04**00 ~ 0x04FF	garbage
...	...	...
0	0x**FD**00 ~ 0xFDFF	garbage
0	0x**FE**00 ~ 0xFEFF	garbage
1	0x**FF**00 ~ 0xFFFF	0x12DB93A

Page table living at PPN 0x8103D32.

Level 2 Page Table		
Valid?	**VPN**	**PPN**
0	0x00**00**	garbage
0	0x00**01**	garbage
1	0x00**02**	0x8932D13
1	0x00**03**	0x903B9CD
0	0x00**04**	garbage
...	...	...
0	0x00**FF**	garbage

Page table living at PPN 0x12DB93A.

Level 2 Page Table		
Valid?	**VPN**	**PPN**
0	0xFF**00**	garbage
0	0xFF**01**	garbage
...	...	...
0	0xFF**FC**	garbage
1	0xFF**FD**	0x45099CD
0	0x00**FE**	garbage
0	0x00**FF**	garbage

Much better than single-level (0x10000 entries).

Multi-Level Page Table Example

Just like in single-level tables, storing the VPN column is unnecessary. It just says 0, 1, 2, 3, 4, 5, etc.

Level 1 Page Table

Valid?	VPNs	L2 Page Table
1	0x0000 ~ 0x00FF	0x8103D32
0	0x0100 ~ 0x01FF	garbage
0	0x0200 ~ 0x02FF	garbage
0	0x0300 ~ 0x03FF	garbage
0	0x0400 ~ 0x04FF	garbage
...	...	...
0	0xFD00 ~ 0xFDFF	garbage
0	0xFE00 ~ 0xFEFF	garbage
1	0xFF00 ~ 0xFFFF	0x12DB93A

Page table living at PPN 0x8103D32.

Level 2 Page Table

Valid?	VPN	PPN
0	0x0000	garbage
0	0x0001	garbage
1	0x0002	0x8932D13
1	0x0003	0x903B9CD
0	0x0004	garbage
...	...	...
0	0x00FF	garbage

Page table living at PPN 0x12DB93A.

Level 2 Page Table

Valid?	VPN	PPN
0	0xFF00	garbage
0	0xFF01	garbage
...	...	...
0	0xFFFC	garbage
1	0xFFFD	0x45099CD
0	0x00FE	garbage
0	0x00FF	garbage

Think of the VPN as indexing into the page tables.
Top bits index into L1 table, and bottom bits index into L2 table.

These slides are more high-level than the rest of VM, so we put them at the end.

If we run out of time, you can read these slides on your own time.

OS Basics

Lecture 34, CS 61C, Fall 2024

What does the OS do?

The OS is the first thing that runs when computer starts.

- Starts dozens of services: File system, network stack (Wi-Fi), keyboard, etc.

The OS provides interaction with the outside world.

- Finds and controls all devices on the machine in a general way.
- Relies on hardware-specific "device drivers."

The OS loads, manages, and runs programs.

- Isolation: Give each program the illusion of its own dedicated world.
- Resource-sharing: Allow multiple programs to share resources.
  - Memory.
  - I/O devices: Disk, keyboard, display, network, etc.
- Time-sharing: Processor/CPU runs multiple processes.

VM helped us achieve these. See optional lectures for the other topics on this slide.

Multiprogramming at a High Level

The OS manages **multiprogramming**, which is running multiple applications on one CPU, switching between them as needed.

- Different from multiprocessing: running processes simultaneously on different CPUs. The OS would also manage this.

The OS achieves this with **context switches** between processes:

- Happens very quickly (a few microseconds).
- Save current process state: program counter, registers, etc.
- Load next process state. ← Also, flush TLB.
- Don't switch out data between memory and disk! Too costly.

Don't evict current process pages out of memory!
Just leave them in memory.

Supervisor Mode vs. User Mode

If a program goes wrong (or is evil, e.g. malware), it could crash the whole machine!

CPUs have a hardware **supervisor mode**, also called **kernel mode**.

- Set by a status bit in a special register.
- An OS process runs in supervisor mode and enforces things like memory access.
- Can think of supervisor mode like the "admin" user.
- Errors in supervisor mode are very dangerous, e.g. disk corrupted, OS stops working and shows "blue screen of death."

By contrast, in **user mode**, a program can only access a subset of instructions and physical memory.

- Can change *out* of supervisor mode using a special instruction.
- Cannot change *into* supervisor mode directly. Instead, hardware interrupt.

OS mostly runs in user mode! Supervisor mode is used sparingly.

Exceptions and Interrupts

Exceptions:

- Caused by an event *during* execution of current program.
- *Synchronous*. Must be handled immediately.

Interrupts:

- Caused by an event *external* to the current program.
- *Asynchronous* to current program. Does not need to be handled immediately (but should be soon).

Examples of exceptions:

- Illegal instruction.
- Divide by zero.
- Page fault.
- Write protection violation.

Examples of interrupts:

- Keyboard press.
- Disk read/write.

Traps Handle Exceptions and Interrupts

The **trap handler** is code that services interrupts and executions.

If an exception/interrupt occurs:

1. Complete all instructions before the faulting instruction.
2. Flush all instructions after the faulting instruction.
    - Like pipeline hazard: Convert to no-ops.
    - Also flush the faulting instruction.
3. Transfer execution to the trap handler in *supervisor mode*.
    - Optionally, return to original program and re-execute instruction.

If the trap handler returns, then from the program's point of view, it must look like nothing has happened!

The Trap Handler

1. Save the state of the current program, e.g. all registers.
2. Determine what caused the exception/interrupt.
3. Handle the exception/interrupt, then do one of two things:

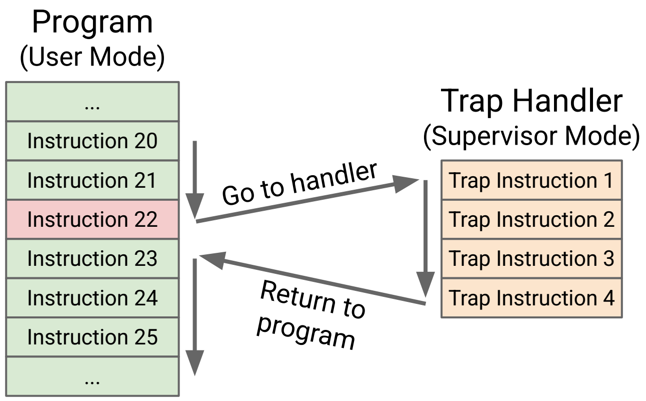
Continue program execution:

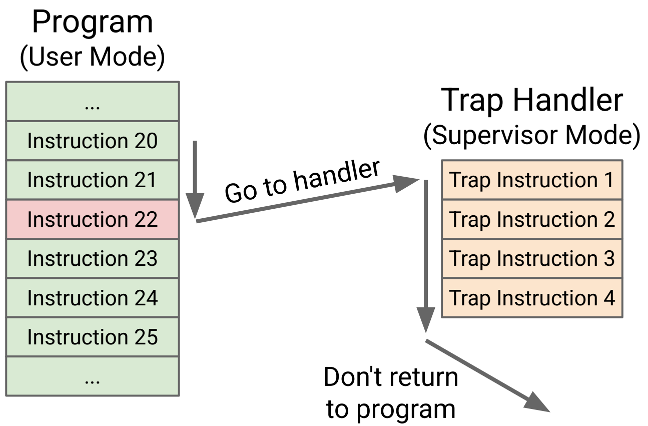
- Restore program state.
- Return control to program.

Terminate the program:

- Free program resources.
- Schedule a new program.

Assume exception/interrupt triggered at Instruction 22.



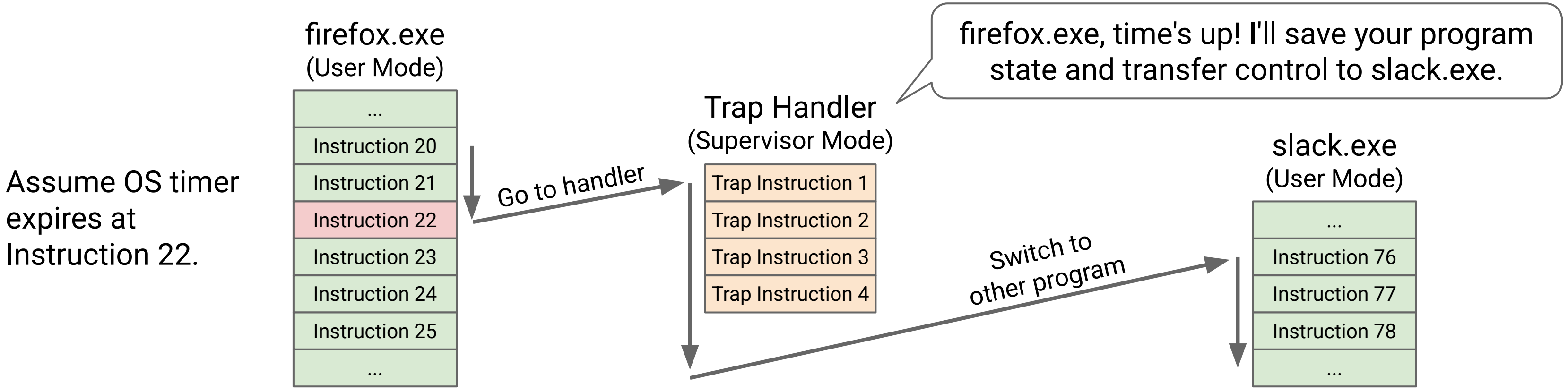


Handling Context Switches

Recall: Context switches allow the OS to switch between programs.

Context switches at a high level:

- OS sets a timer. When it expires, perform a *hardware interrupt*.
- Trap handler saves registers, including:
    - Program counter.
    - Page table register: Holds memory address of current program's page table.
- Trap handler then loads in next program's registers and returns to user mode.



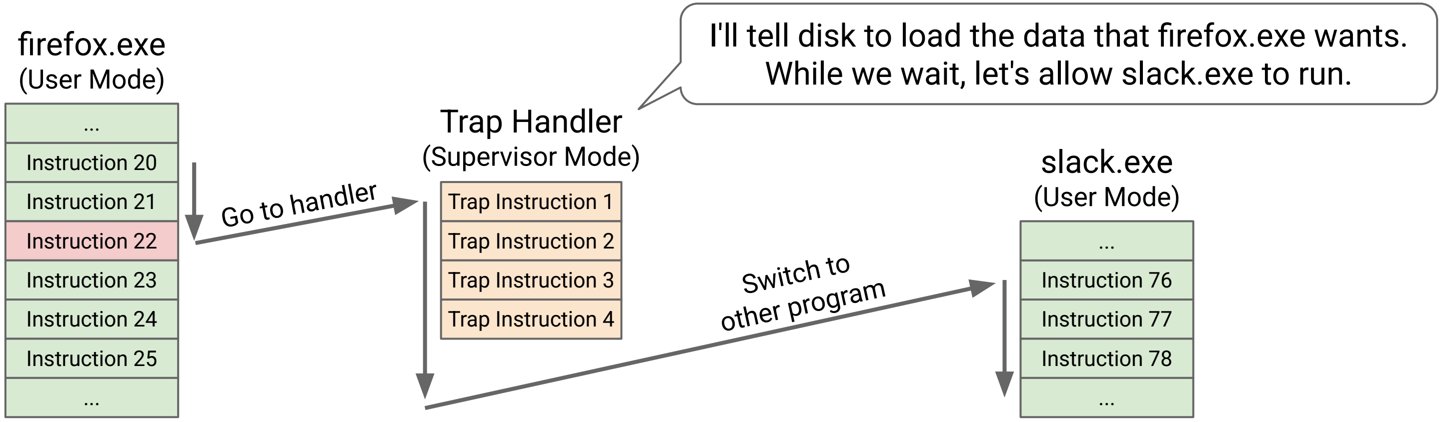
Handling Page Faults

Recall page faults: Program requests a page that's on disk, not in memory.

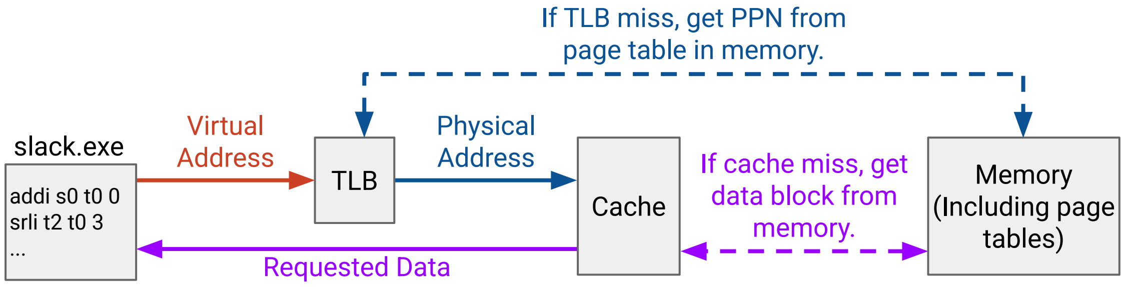
Page faults are handled by the trap handler.

- The *page fault exception handler* initiates transfers to/from disk, and performs any page table updates.
- If pages need to be swapped from disk, perform a context switch so that another process can use the CPU in the meantime.
- After the page fault, re-execute the faulting instruction.

Assume page fault at Instruction 22, e.g. lw data on disk.

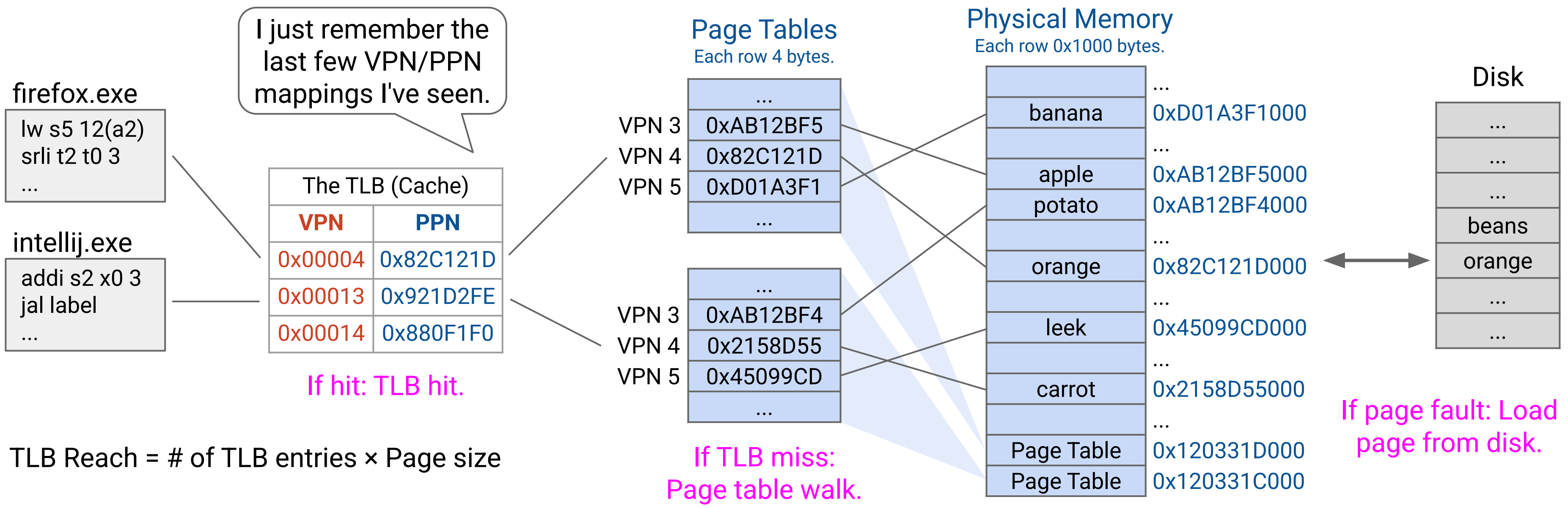


Summary: Virtual Memory II


If TLB miss, get PPN from page table in memory.
slack.exe
Virtual Address
TLB
Physical Address
Cache
If cache miss, get data block from memory.
Memory (Including page tables)
Requested Data

```
addi s0 t0 0
srli t2 t0 3
...
```

	TLB hit?	PTE valid?
Best: TLB Hit	✔ Hit	Never visited
Worse: TLB Miss, Page Table Walk	✘ Miss	✔ Valid
Worst: TLB Miss, Page Fault	✘ Miss	✘ Invalid



firefox.exe

```
lw s5 12(a2)
srli t2 t0 3
...
```

intellij.exe

```
addi s2 x0 3
jal label
...
```

The TLB (Cache)

VPN	PPN
0x00004	0x82C121D
0x00013	0x921D2FE
0x00014	0x880F1F0

TLB Reach = # of TLB entries × Page size