

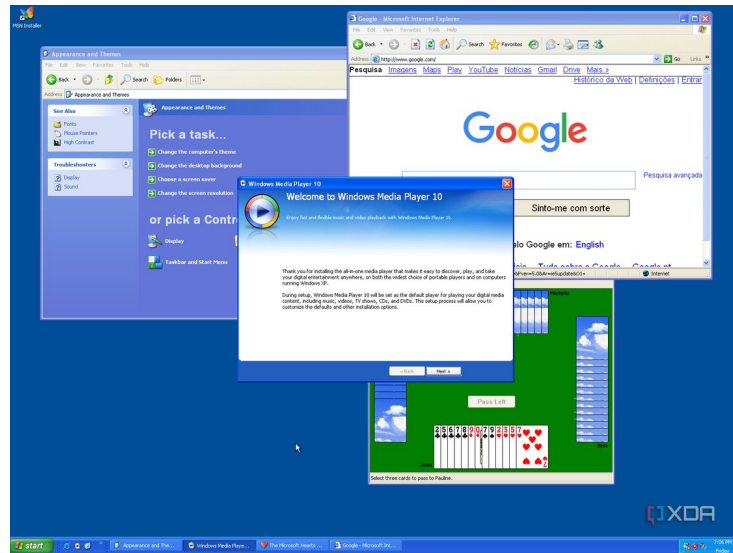
## Lecture 33

# Virtual Memory I

CS 61C, Fall 2024 @ UC Berkeley

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

Slides template credit: Josh Hug, Lisa Yan



## For Today: Pretend Caches Don't Exist

---

Today's topic is **virtual memory**.

- It's easier to understand if we temporarily forget that caches exist.
- For today, imagine that there are no caches.  
All loads/stores directly access memory.

Next time, we'll see how caches interact with virtual memory.

My personal opinion: Virtual memory is very similar to caches, but trying to compare them immediately will confuse you even further.

I suggest trying to understand VM first, forgetting about caches.  
Then, compare them only after you're comfortable with VM.

For this reason, these slides will never mention caches until the end.

# Running Multiple Programs

---

Lecture 33, CS 61C, Fall 2024

## Running Multiple Programs

Virtual and Physical Addresses

Page Tables

Memory Access Example

Virtual Memory Parameters

Memory Paging

VM Design Choices

Benefits of Virtual Memory

# Our Memory Model So Far

Our model so far: Each program has its own dedicated memory.

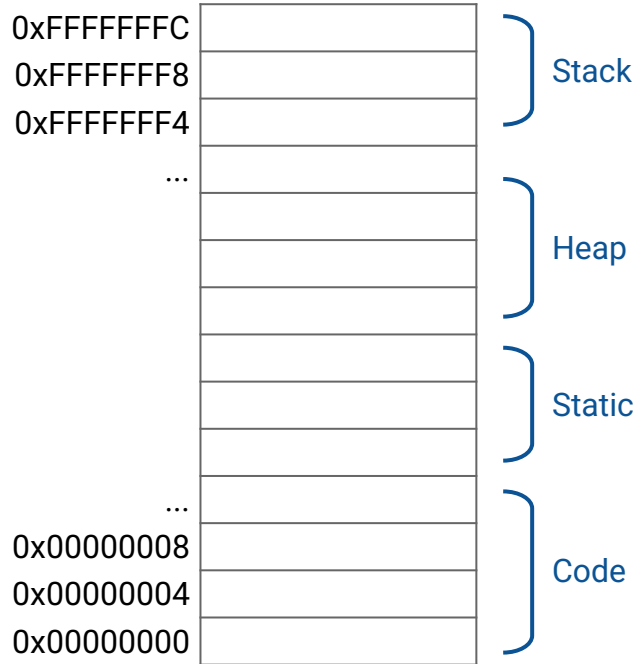
32-bit system: Array of  $2^{32}$  bytes, addressed from 0x00000000 to 0xFFFFFFFF.

classify.s

```
add s0 t5 t6  
sw s0 0(a0)  
lw s1 4(a1)  
...
```



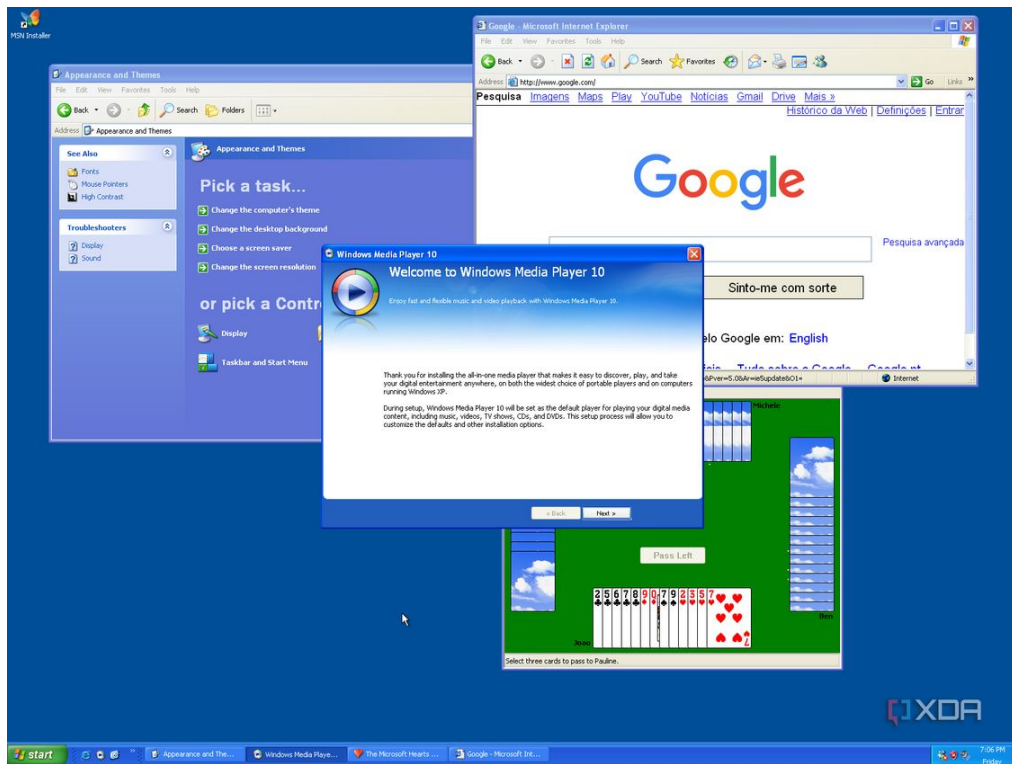
When your program  
loads or stores, it  
accesses memory.



# A Real Computer

When I run Venus, it only executes one program, and then stops.

Our computer runs many programs simultaneously!

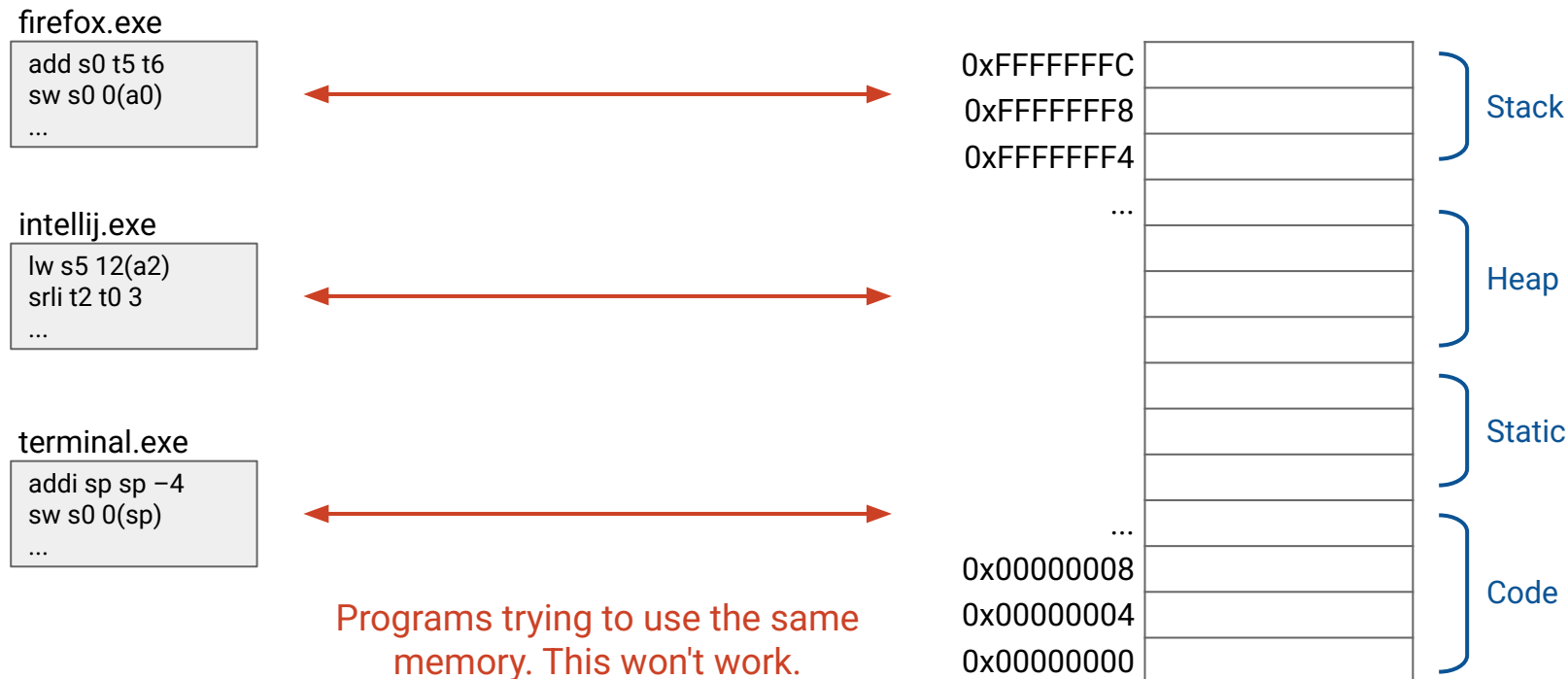


I miss Windows XP.

## Today's Goal: Virtual Memory

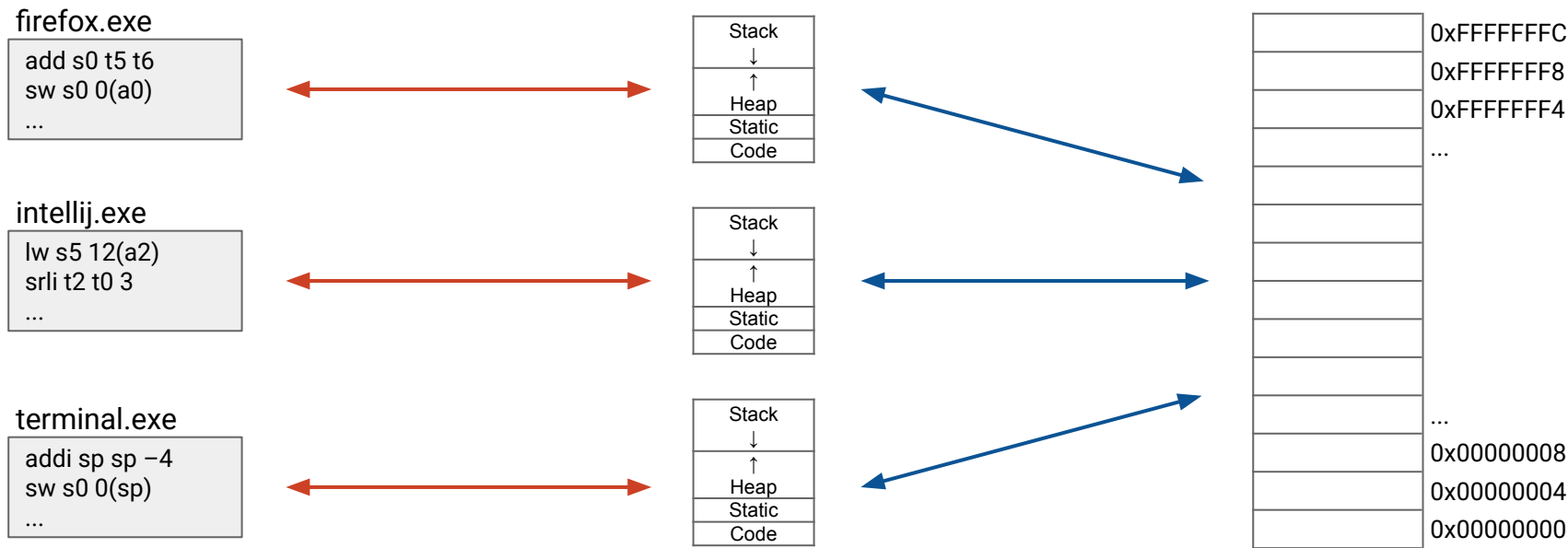
If multiple programs try to access the same memory, we'd have problems.

## What if firefox.exe and terminal.exe both want to store data at the same address?



# Today's Goal: Virtual Memory

Today's goal: Give each process the *illusion* of its own dedicated memory...  
...even though everyone is sharing the same memory on the computer.



Programs think they're accessing  
their own dedicated memory...

...but they're really sharing  
the same address space.

# Virtual Addresses and Physical Addresses

---

Lecture 33, CS 61C, Fall 2024

Running Multiple Programs

## **Virtual and Physical Addresses**

Page Tables

Memory Access Example

Virtual Memory Parameters

Memory Paging

VM Design Choices

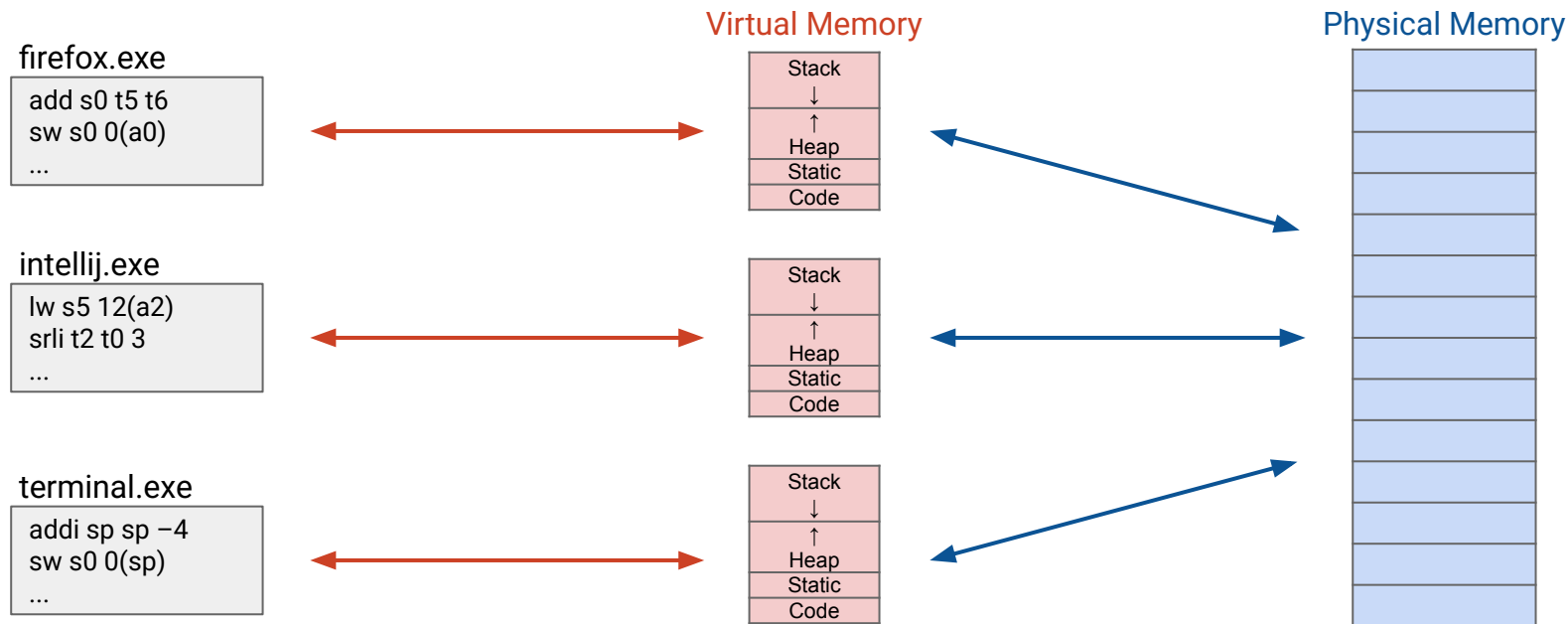
Benefits of Virtual Memory



# Virtual and Physical Addresses

**Virtual addresses:** The addresses the program is thinking about.

**Physical addresses:** The addresses that actually map to physical memory locations.



## Virtual and Physical Addresses

---

There are now two kinds of memory addresses:

**Virtual addresses:** The addresses the program is thinking about.

- Each program has its own dedicated virtual address space.

**Physical addresses:** The addresses that actually map to physical memory locations.

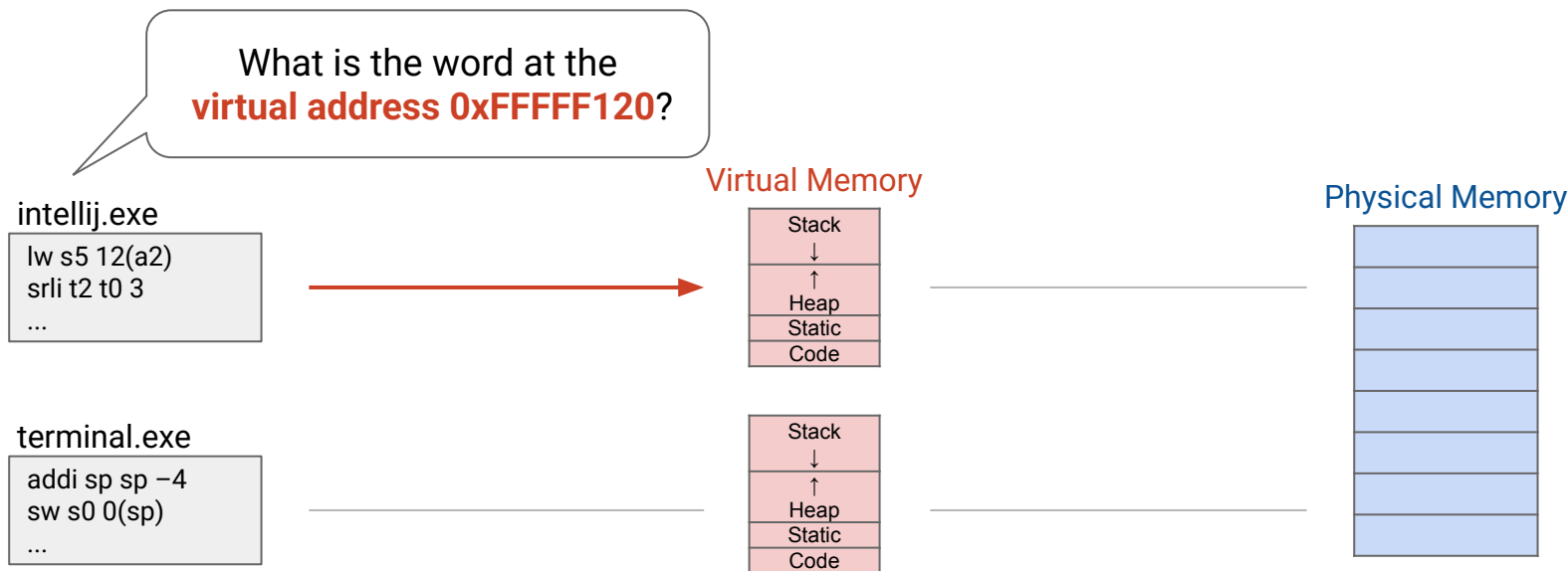
- Everyone shares the same physical address space.

The *memory manager* in the OS translates virtual addresses to physical addresses.

- Programs don't need to think about physical addresses!

# Virtual Address Translation (Conceptual)

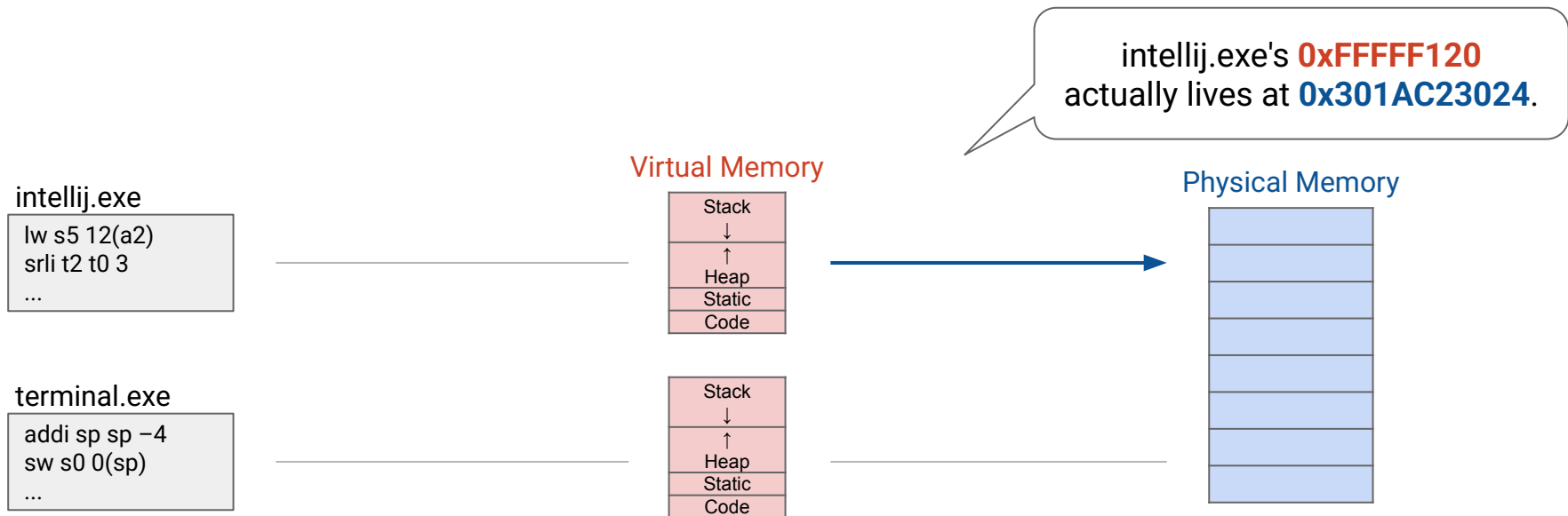
Step 1: The program wants to read/write from a **virtual address**.



## Virtual Address Translation (Conceptual)

Steps 2–4: OS translates the **virtual address** to its corresponding **physical address**.

The OS can now fetch the data at that **physical address**.



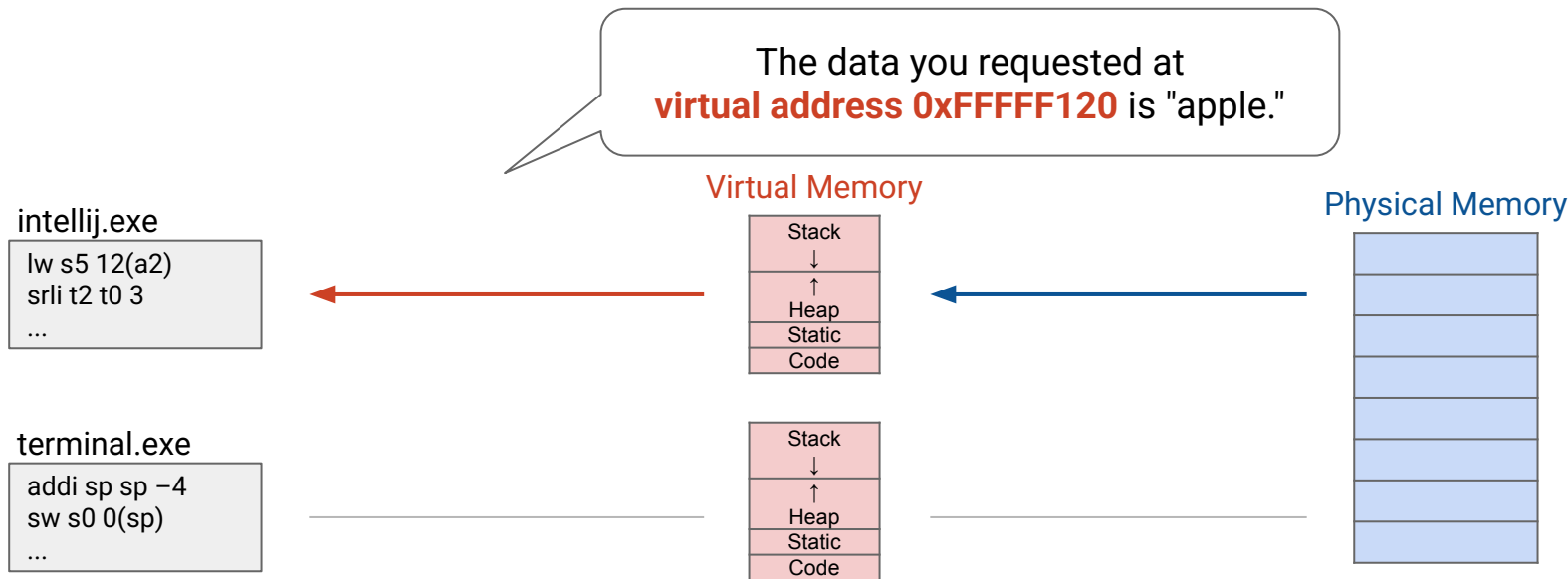
We will see why it's 3 steps (Steps 2–4) later.

10-digit = 40-bit physical address because we assume computer has  $2^{40}$  bits of physical memory. More on this later.

# Virtual Address Translation (Conceptual)

Step 5: The OS returns that data to the program.

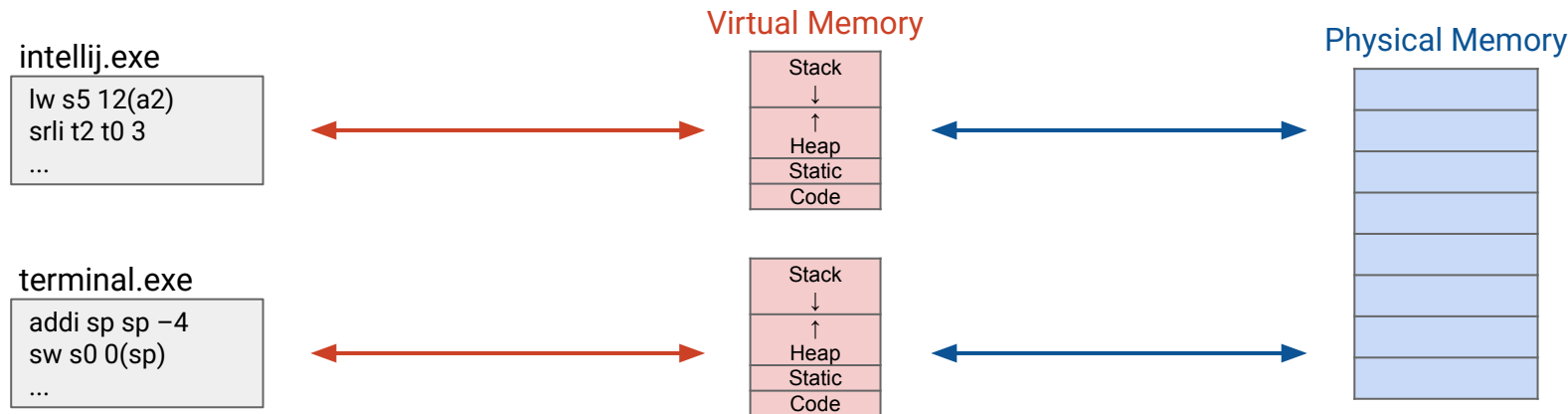
The program never thinks about physical addresses!



## Programs Using the Same Virtual Address

Notice: Two programs could use the same virtual address.

- intellij.exe could store "apple" at virtual address **0xFFFFF120**.
- terminal.exe could store "potato" at virtual address **0xFFFFF120**.
- The OS stores "apple" and "potato" at two different physical addresses.



# Page Tables

---

Lecture 33, CS 61C, Fall 2024

Running Multiple Programs

Virtual and Physical Addresses

## **Page Tables**

Memory Access Example

Virtual Memory Parameters

Memory Paging

VM Design Choices

Benefits of Virtual Memory

# Mapping Virtual to Physical Addresses (Naive)

How do we remember the mapping of virtual addresses to physical addresses?

- Idea: Let's use a table!
- We need **one table for each program**.

Because each program has its own virtual address space.

Naive approach: Map every virtual address to a corresponding physical address.

- Problem: These tables will be huge.

firefox.exe's Table (Naive)	
Virtual Address	Physical Address
...	...
0x00000400	0x60C25E65CE
0x00000401	0x1F96EC70D9
0x00000402	0xEC70DB7F19
0x00000403	0x69977241BB
...	...

intellij.exe's Table (Naive)	
Virtual Address	Physical Address
...	...
0x00000400	0xCD178731DD
0x00000401	0x128E05D21C
0x00000402	0x0C709FD626
0x00000403	0x937D13CF3E
...	...



## Mapping Virtual to Physical Addresses with Pages

Better approach: Group all memory (virtual and physical) into **pages**.

- Example: Each page could be 4 KiB = 0x1000 addresses.

The table now maps **virtual pages** to **physical pages**.

- **Virtual page:** A chunk of memory in the virtual address space.
- **Physical page:** A chunk of memory in the physical address space.

firefox.exe's Table	
Virtual Page	Physical Page
...	...
0x00004000 to 0x00004FFF	0x60C25E6000 to 0x60C25E6FFF
0x00005000 to 0x00005FFF	0x1F96EC7000 to 0x1F96EC7FFF
0x00006000 to 0x00006FFF	0xEC70DB7000 to 0xEC70DB7FFF
0x00007000 to 0x00007FFF	0x6997724000 to 0x6997724FFF
...	...

intellij.exe would have its own table (not pictured here).

# Mapping Virtual to Physical Addresses with Pages

Same address translation as before.  
We just grouped data into *pages*.

Note: Page size is the same in both  
physical and virtual worlds.

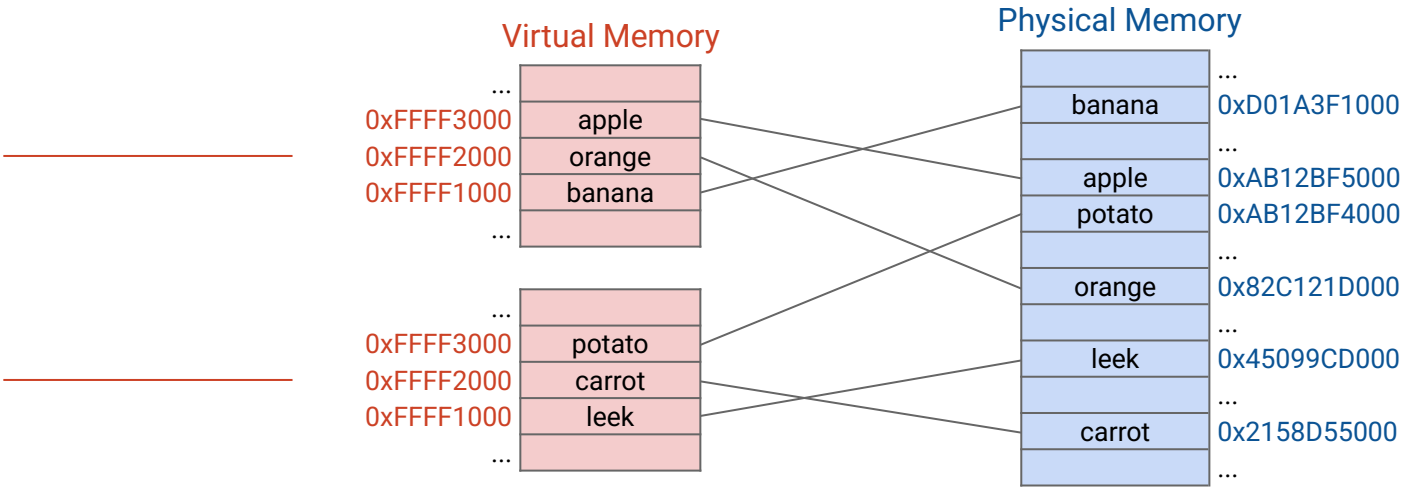
firefox.exe's Table	
Virtual Page	Physical Page
...	...
0xFFFF1000 to 0x00004FFF	0xD01A3F1000 to 0xD01A3FFFFF
0xFFFF2000 to 0x00005FFF	0x82C121D000 to 0x82C121DFFF
0xFFFF3000 to 0x00006FFF	0xAB12BF5000 to 0xAB12BF5FFF
...	...

firefox.exe

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

intellij.exe

```
addi sp sp -4
sw s0 0(sp)
...
```



Each English word here represents arbitrary data in a fixed-size page, e.g. imagine "potato" and "leek" are both 0x1000 bytes.

## Storing Page Tables – Page Numbers

The table maps **virtual pages** to **physical pages**.

firefox.exe's Table	
Virtual Page	Physical Page
...	...
0x00004000 to 0x00004FFF	0x60C25E6000 to 0x60C25E6FFF
0x00005000 to 0x00005FFF	0x1F96EC7000 to 0x1F96EC7FFF
0x00006000 to 0x00006FFF	0xEC70DB7000 to 0xEC70DB7FFF
0x00007000 to 0x00007FFF	0x6997724000 to 0x6997724FFF
...	...

We don't really need to store all this data.

- Instead of "0x00004000 to 0x00004FFF," just store "0x00004."
- Instead of "0x60C25E6000 to 0x60C25E6FFF," just store "0x60C25E6."
- We call these top bits the **page number**.

## Storing Page Tables – Page Numbers

The table maps **virtual pages** to **physical pages**.

firefox.exe's Table	
Virtual Page Number (VPN)	Physical Page Number (PPN)
...	...
<b>0x00004</b> <del>000 to 0x00004FFF</del>	<b>0x60C25E6</b> <del>000 to 0x60C25E6FFF</del>
<b>0x00005</b> <del>000 to 0x00005FFF</del>	<b>0x1F96EC7</b> <del>000 to 0x1F96EC7FFF</del>
<b>0x00006</b> <del>000 to 0x00006FFF</del>	<b>0xEC70DB7</b> <del>000 to 0xEC70DB7FFF</del>
<b>0x00007</b> <del>000 to 0x00007FFF</del>	<b>0x6997724</b> <del>000 to 0x6997724FFF</del>
...	...

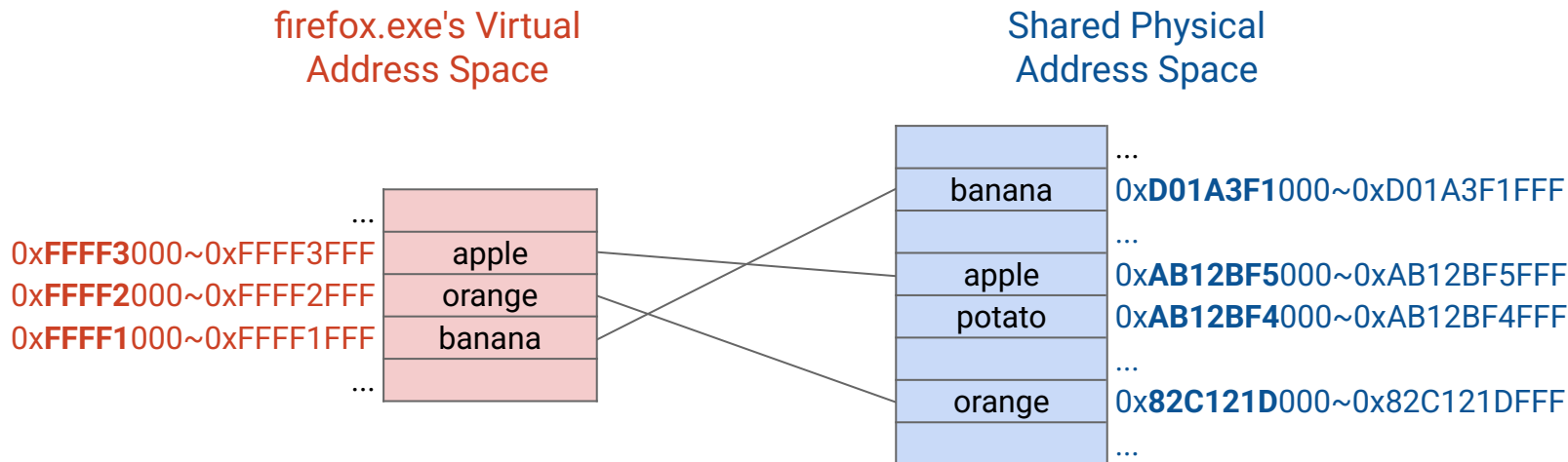
We don't really need to store all this data.

- Instead of "**0x00004**~~000 to 0x00004FFF~~," just store "**0x00004**."
- Instead of "**0x60C25E6**~~000 to 0x60C25E6FFF~~," just store "**0x60C25E6**."
- We call these top bits the **page number**.

# Storing Page Tables – Page Numbers

We can interpret the top bits of an address as a **page number**.

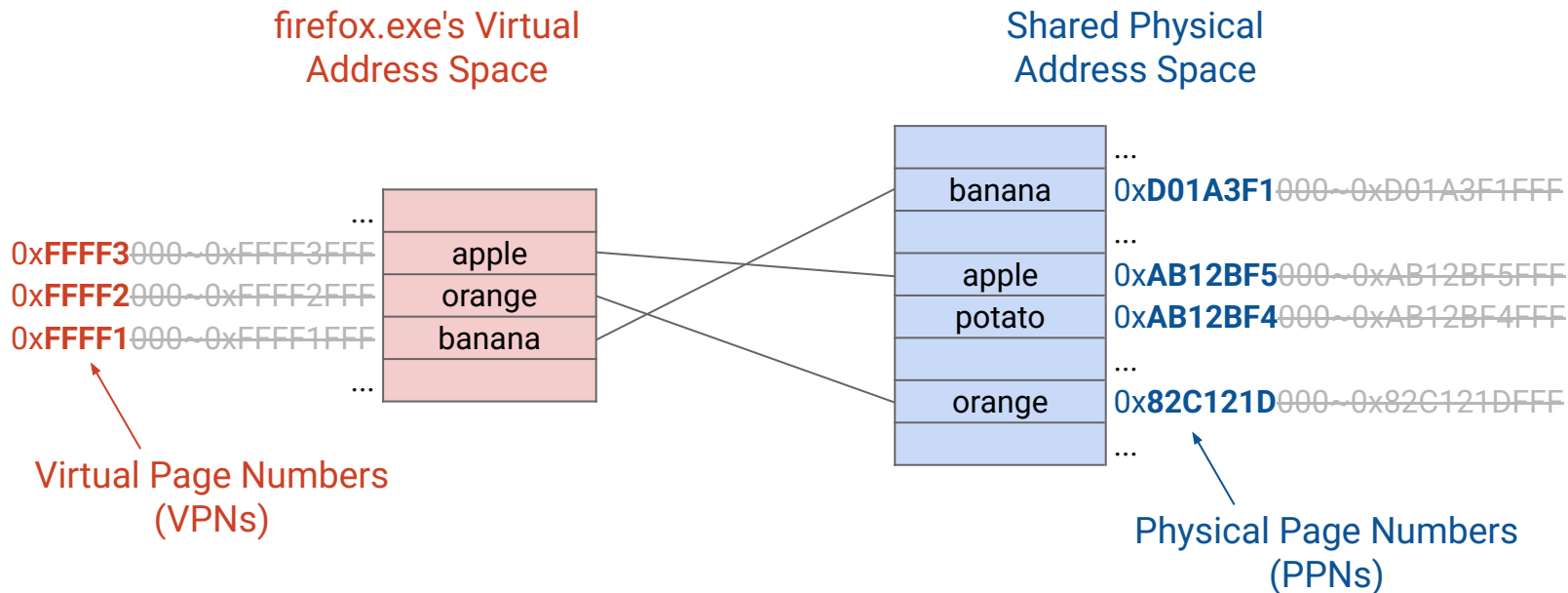
- Virtual page **0x00004** has addresses **0x00004000** to **0x00004FFF**.
- Physical page **0x60C25E6** has addresses **0x60C25E6000** to **0x60C25E6FFF**.



# Storing Page Tables – Page Numbers

We can interpret the top bits of an address as a **page number**.

- Virtual page **0x00004** has addresses **0x00004000** to **0x00004FFF**.
- Physical page **0x60C25E6** has addresses **0x60C25E6000** to **0x60C25E6FFF**.



## Address Translation with Page Tables

---

The thing we just built is called the **page table**.

- Maps virtual page numbers (VPNs) to physical page numbers (PPNs).
- Remember: Every program needs its own page table.
- Notice: Page tables do not store actual program data! They only store addresses.

firefox.exe's Page Table	
Virtual Page Number (VPN)	Physical Page Number (PPN)
...	...
0x00004	0x60C25E6
0x00005	0x1F96EC7
0x00006	0xEC70DB7
0x00007	0x6997724
...	...

# Address Translation with Page Tables

---

Performing address translation with a page table:

Step 2: Split virtual address into VPN and offset.

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

Step 4: Concatenate PPN and offset.

firefox.exe's Page Table	
Virtual Page Number (VPN)	Physical Page Number (PPN)
...	...
0x00004	0x60C25E6
0x00005	0x1F96EC7
0x00006	0xEC70DB7
0x00007	0x6997724
...	...



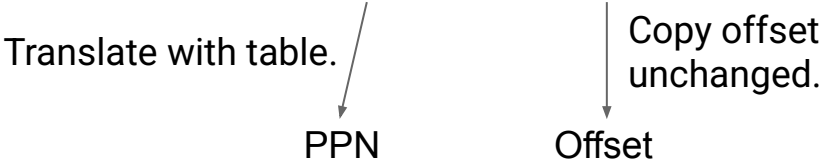
# Address Translation with Page Tables

Example: Virtual address **0x00004A3C** translates to physical address **0x60C25E6A3C**.

Step 2: Split virtual address into VPN and offset.

VPN	Offset
00004	A3C

Step 3: Translate VPN to PPN using table.



Step 4: Concatenate PPN and offset.

firefox.exe's Page Table	
Virtual Page Number (VPN)	Physical Page Number (PPN)
...	...
0x00004	0x60C25E6
0x00005	0x1F96EC7
0x00006	0xEC70DB7
0x00007	0x6997724
...	...

# Storing Page Tables – VPN as Index

Notice: The VPN column just says 0, 1, 2, 3, 4, 5, etc.

- Optimization: There's no need to store the VPN column.

The page table can be stored as an *array* of PPNs.

- The VPN can be used to *index* into the array to find the corresponding VPN.
- Example: To translate VPN 0x00005, get the 5th PPN in the table.
- Example: To translate VPN 0x721CD, get the 0x721CD'th PPN in the table.

firefox.exe's Page Table	
Virtual Page Number (VPN)	Physical Page Number (PPN)
...	...
0x00004	0x60C25E6
0x00005	0x1F96EC7
0x00006	0xEC70DB7
...	...



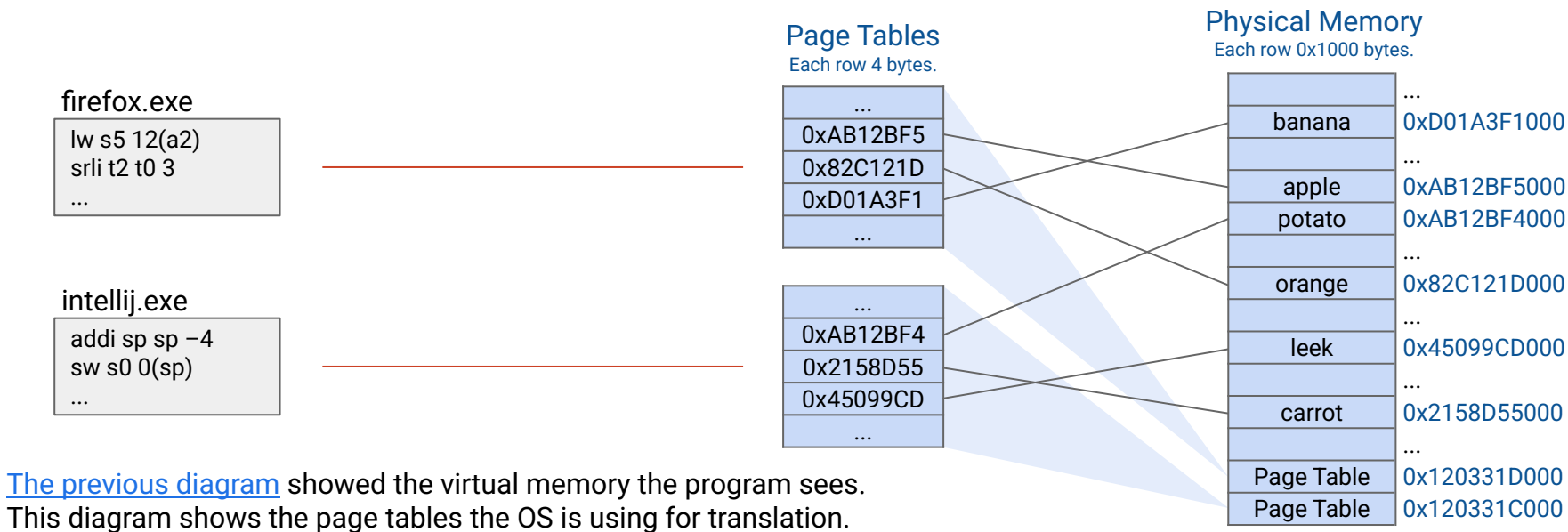
firefox.exe's Page Table (Optimized)	
Physical Page Number (PPN)	
...	
0x60C25E6	
0x1F96EC7	
0xEC70DB7	
...	

The slides will sometimes show unoptimized page tables for clarity.

# Page Tables Live in Memory

Just like any other data, **page tables live in physical 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.



# Memory Access Example

---

Lecture 33, CS 61C, Fall 2024

Running Multiple Programs

Virtual and Physical Addresses

Page Tables

## **Memory Access Example**

Virtual Memory Parameters

Memory Paging

VM Design Choices

Benefits of Virtual Memory

# Memory Access Example

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

What is the byte at the  
**virtual address 0x00004A3C?**

firefox.exe

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

intellij.exe

```
addi sp sp -4
sw s0 0(sp)
...
```

Page Tables  
Each row 4 bytes.

	...
VPN 3	0xAB12BF5
VPN 4	0x82C121D
VPN 5	0xD01A3F1
	...

Physical Memory  
Each row 0x1000 bytes.

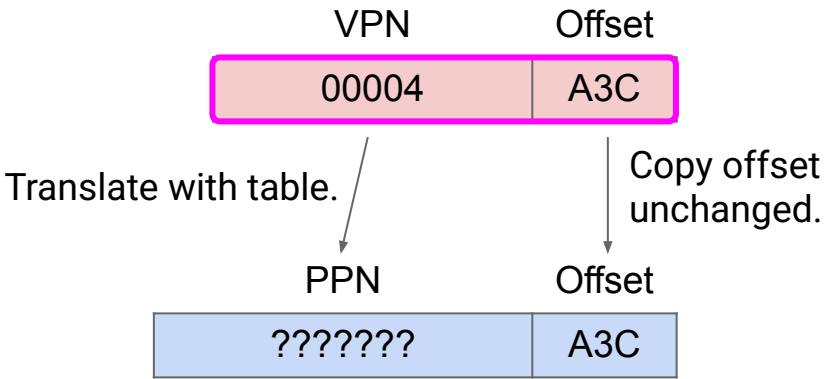
...	...
banana	0xD01A3F1000
...	...
apple	0xAB12BF5000
potato	0xAB12BF4000
...	...
orange	0x82C121D000
...	...
leek	0x45099CD000
...	...
carrot	0x2158D55000
...	...
Page Table	0x120331D000
Page Table	0x120331C000

# Memory Access Example

Step 2: Extract VPN and offset.

Use parameters to decide number of VPN bits and number of offset bits. (More on this later.)

What is the byte at the **virtual address 0x00004A3C?**



firefox.exe

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

intellij.exe

```
addi sp sp -4
sw s0 0(sp)
...
```

Page Tables  
Each row 4 bytes.

	...
VPN 3	0xAB12BF5
VPN 4	0x82C121D
VPN 5	0xD01A3F1
	...

Physical Memory  
Each row 0x1000 bytes.

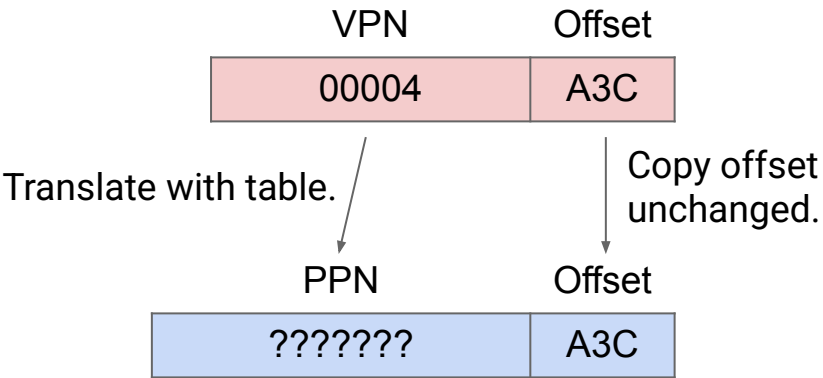
...	...
banana	0xD01A3F1000
...	...
apple	0xAB12BF5000
potato	0xAB12BF4000
...	...
orange	0x82C121D000
...	...
leek	0x45099CD000
...	...
carrot	0x2158D55000
...	...
Page Table	0x120331D000
Page Table	0x120331C000

# Memory Access Example

Step 3: Translate VPN to PPN using table.

Look up the VPN'th index in the table to find the corresponding PPN.

VPN is 4, so look up 4th PPN in the table.



firefox.exe

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

intellij.exe

```
addi sp sp -4
sw s0 0(sp)
...
```

## Page Tables

Each row 4 bytes.

...	...
VPN 3	0xAB12BF5
VPN 4	0x82C121D
VPN 5	0xD01A3F1
...	...

...	...
VPN 3	0xAB12BF4
VPN 4	0x2158D55
VPN 5	0x45099CD
...	...

## Physical Memory

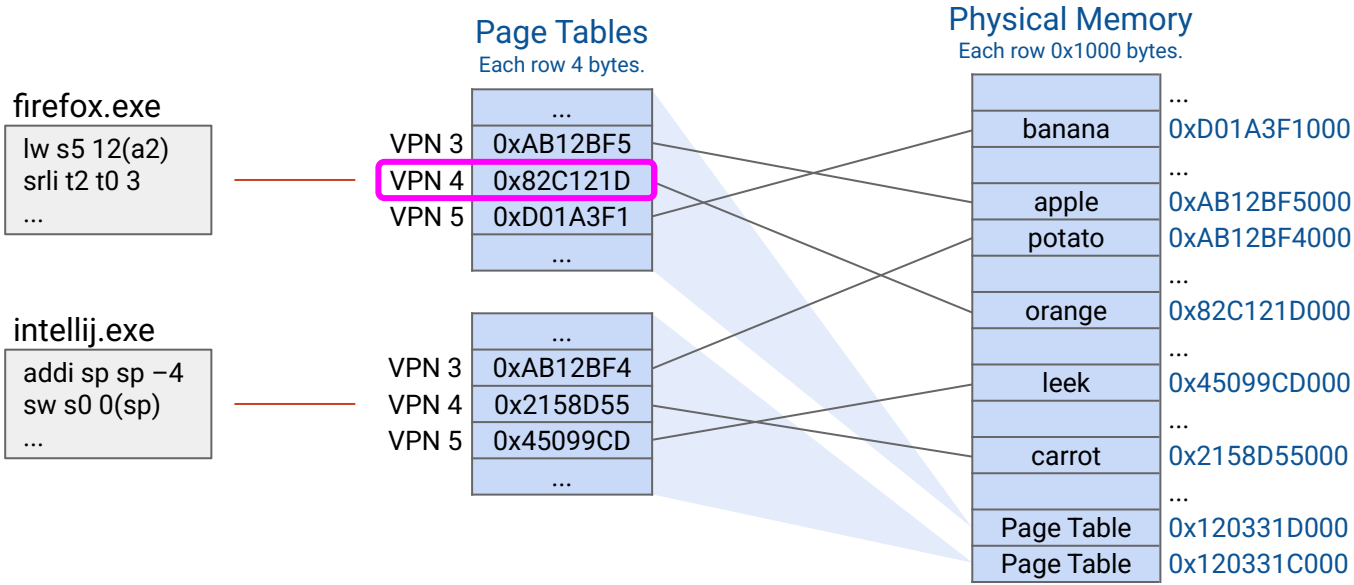
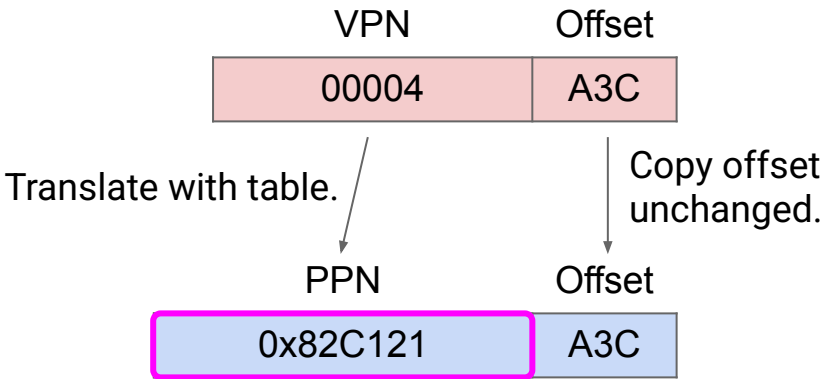
Each row 0x1000 bytes.

...	...
banana	0xD01A3F1000
...	...
apple	0xAB12BF5000
potato	0xAB12BF4000
...	...
orange	0x82C121D000
...	...
leek	0x45099CD000
...	...
carrot	0x2158D55000
...	...
Page Table	0x120331D000
Page Table	0x120331C000

# Memory Access Example

Step 4: Concatenate PPN and offset.

Physical address is **0x82C121A3C**.





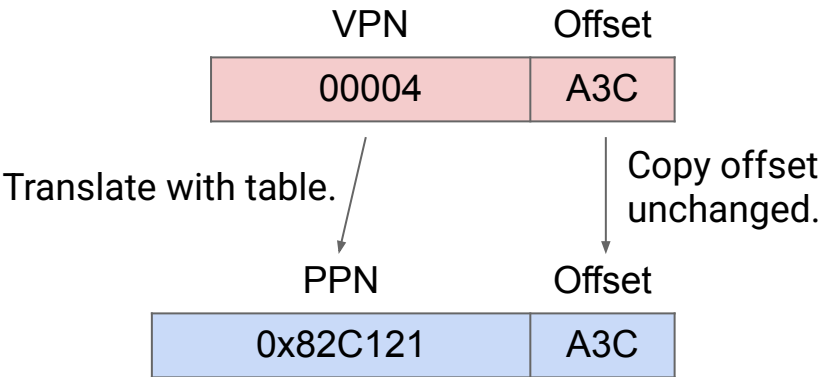
# Memory Access Example

Step 5: Return data to the program.

Requested page says "orange."

Return the 0xA3C'th byte on that page.

(Remember, we used "orange" to represent a page's worth of arbitrary data.)



firefox.exe

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

intellij.exe

```
addi sp sp -4
sw s0 0(sp)
...
```

## Page Tables

Each row 4 bytes.

...	
VPN 3	0xAB12BF5
VPN 4	0x82C121D
VPN 5	0xD01A3F1
...	

...	
VPN 3	0xAB12BF4
VPN 4	0x2158D55
VPN 5	0x45099CD
...	

## Physical Memory

Each row 0x1000 bytes.

...	
banana	0xD01A3F1000
...	
apple	0xAB12BF5000
potato	0xAB12BF4000
...	
orange	0x82C121D000
...	
leek	0x45099CD000
...	
carrot	0x2158D55000
...	
Page Table	0x120331D000
Page Table	0x120331C000

# Virtual Memory Parameters

---

Lecture 33, CS 61C, Fall 2024

Running Multiple Programs

Virtual and Physical Addresses

Page Tables

Memory Access Example

**Virtual Memory Parameters**

Memory Paging

VM Design Choices

Benefits of Virtual Memory

## Virtual Memory Parameters

---

There are certain "knobs" you can turn in a system.

These parameters determine what addresses look like.

Parameter to change:	This parameter affects:
Page size	# of offset bits.
Virtual memory size	# of bits in virtual address.
Physical memory size	# of bits in physical address.

—————→ Take  $\log_2(\text{size})$  to determine # of bits.

Take  $2^{\text{\# of bits}}$  to determine size.

←—————

# Virtual Memory Parameters

There are certain "knobs" you can turn in a system.  
These parameters determine what addresses look like.

Parameter to change:	This parameter affects:	These determine the number of VPN bits.
Page size	# of offset bits.	
Virtual memory size	# of bits in virtual address.	
Physical memory size	# of bits in physical address.	

$VPN\ bits = Virtual\ address\ bits - Offset\ bits.$   
You can think of page number as "leftover bits" after removing the offset bits.

# Virtual Memory Parameters

There are certain "knobs" you can turn in a system.

These parameters determine what addresses look like.

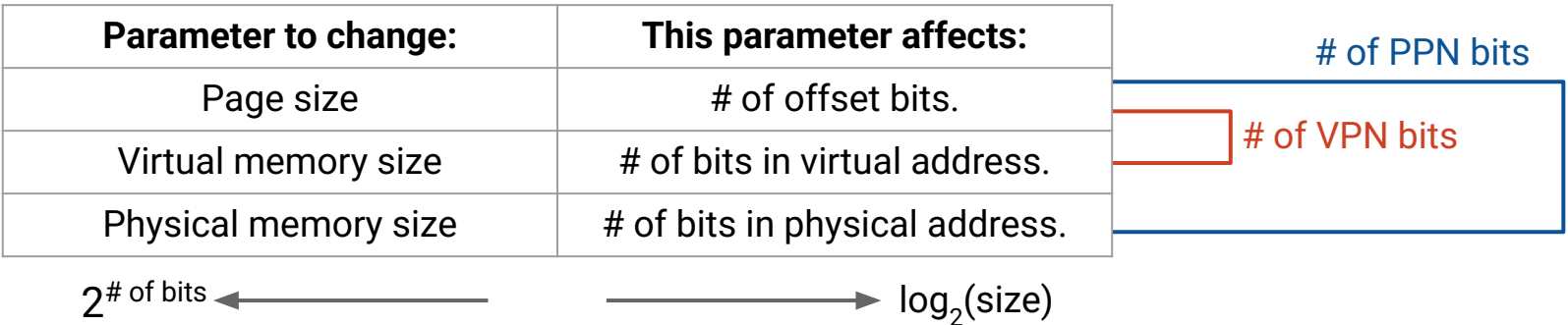
Parameter to change:	This parameter affects:
Page size	# of offset bits.
Virtual memory size	# of bits in virtual address.
Physical memory size	# of bits in physical address.

These determine the  
number of PPN bits.

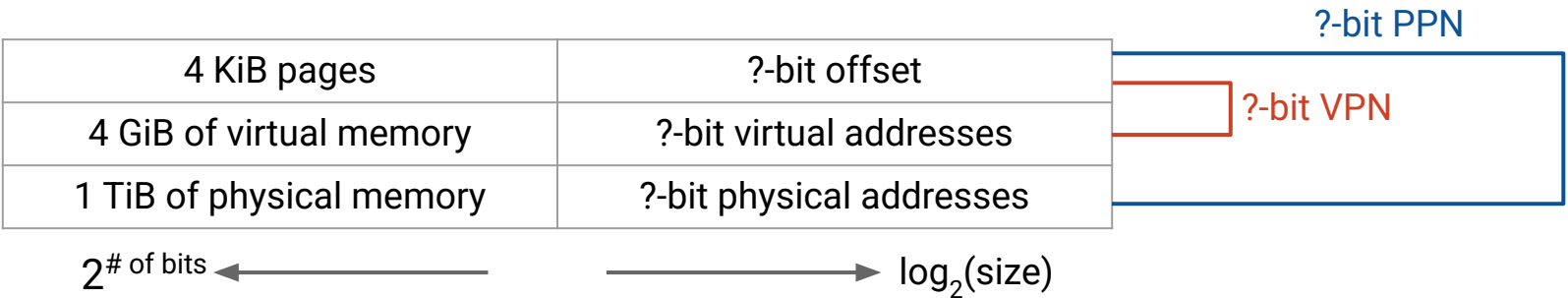
$\text{PPN bits} = \text{Physical address bits} - \text{Offset bits}.$

You can think of page number as "leftover bits" after removing the offset bits.

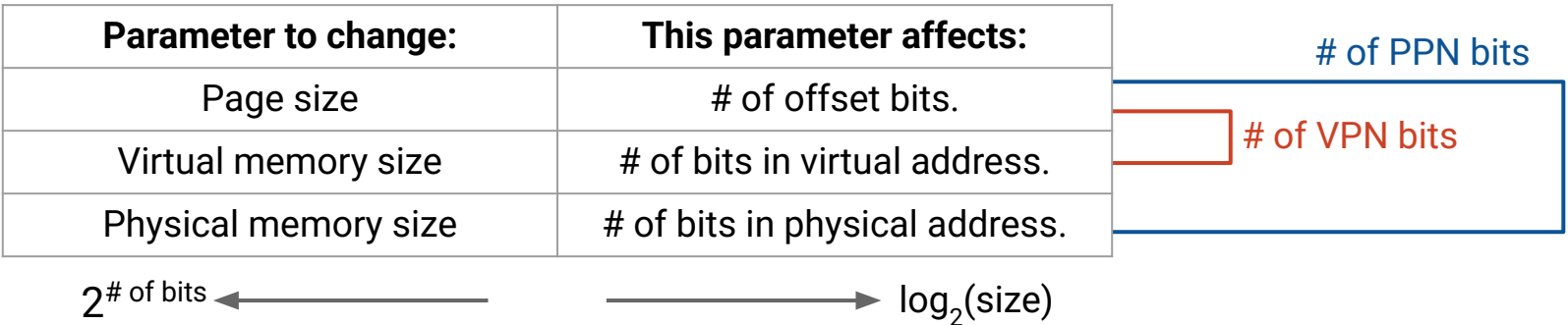
# Virtual Memory Parameters – Example



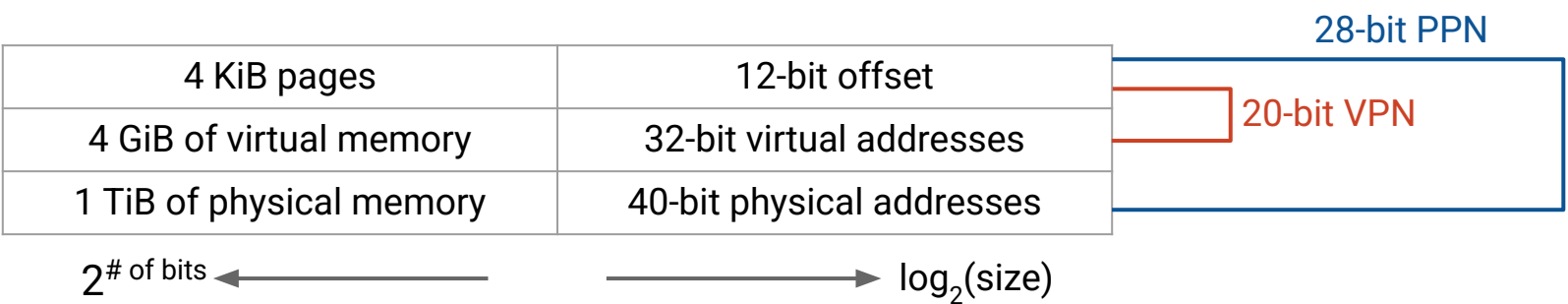
Example parameters:



# Virtual Memory Parameters – Example



Example parameters:



## Page Table Size

---

The approximate size of the page table can be computed from the other parameters.

$$\text{Size of page table} = \text{\# of PTEs} \times \text{Size per PTE}$$



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

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



$\text{\# of PPN bits}^*$

Because each entry is just a PPN value.

\*Additional status bits (e.g. valid bit, dirty bit) might increase size per PTE.  
More on this later.



# Memory Paging

---

Lecture 33, CS 61C, Fall 2024

Running Multiple Programs

Virtual and Physical Addresses

Page Tables

Memory Access Example

Virtual Memory Parameters

## **Memory Paging**

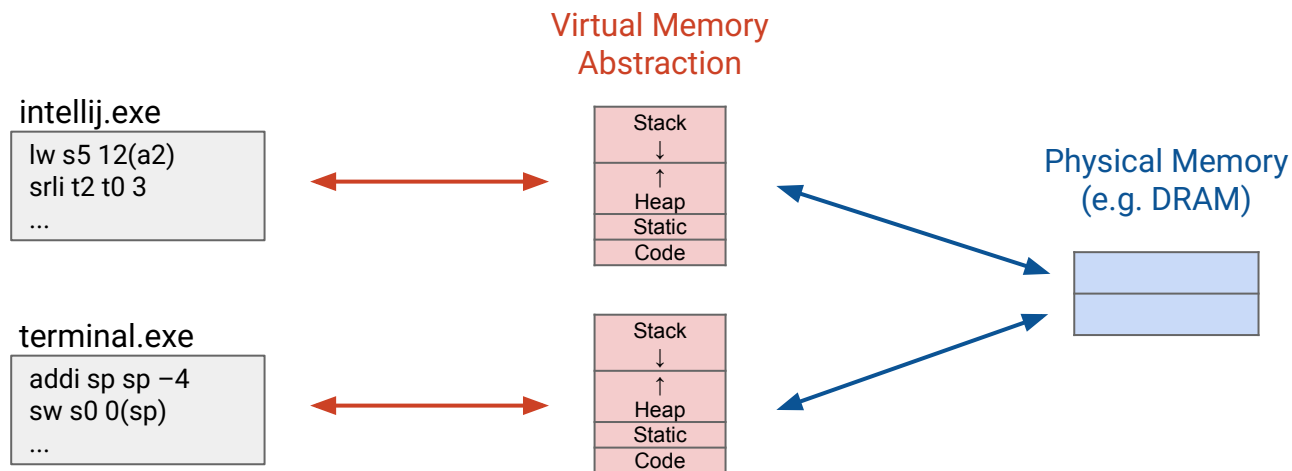
VM Design Choices

Benefits of Virtual Memory

## Problem: Exceeding Physical Memory

What if we don't have enough physical memory?

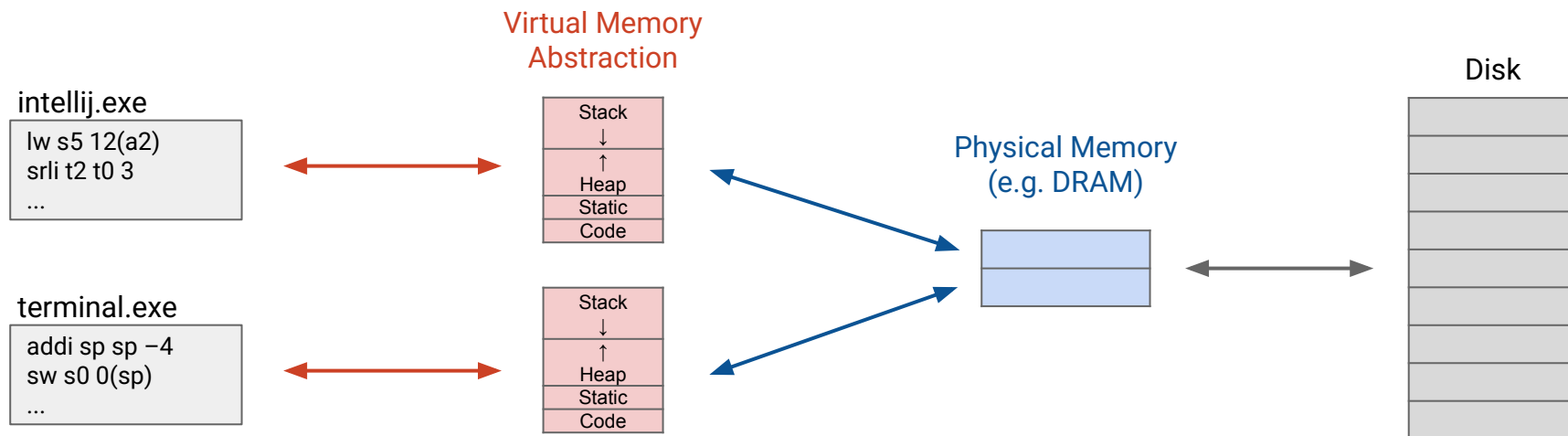
- Example: Virtual memory size > physical memory size.
- Example: Lots of programs. Not enough physical pages to support all programs.



## Solution: Additional Storage on Disk

What if we don't have enough physical memory?

- Solution: Store additional pages on the *disk*. (Or some other secondary storage, e.g. tape, DVD.)
- Only store a subset of pages in memory.
- Gives the illusion of a larger physical memory.



Now, a page table entry (PTE) might say "this page is on disk," instead of PPN.

If a program tries to access a page that's not in memory, we get a **page fault**.

- The OS looks at the page table, and realizes the page is on disk.
- The OS loads the requested page from disk into memory.  
This might *evict* another page, kicking it out of memory.
- The OS updates the page table with the location (PPN) of the newly-loaded page.
- The OS returns the requested data to the program.

firefox.exe's Page Table	
VPN	PPN
...	...
0x00004	0x60C25E6
0x00005	Disk
0x00006	0xEC70DB7
...	...

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

What is the byte at the **virtual address 0x00004A3C**?

firefox.exe

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

intellij.exe

```
addi sp sp -4
sw s0 0(sp)
...
```

Page Tables  
Each row 4 bytes.

	...
VPN 3	0xAB12BF5
VPN 4	Disk
VPN 5	0xD01A3F1
	...

	...
VPN 3	0xAB12BF4
VPN 4	0x2158D55
VPN 5	0x45099CD
	...

Physical Memory  
Each row 0x1000 bytes.

	...
banana	0xD01A3F1000
	...
apple	0xAB12BF5000
potato	0xAB12BF4000
	...
beans	0x82C121D000
	...
leek	0x45099CD000
	...
carrot	0x2158D55000
	...
Page Table	0x120331D000
Page Table	0x120331C000

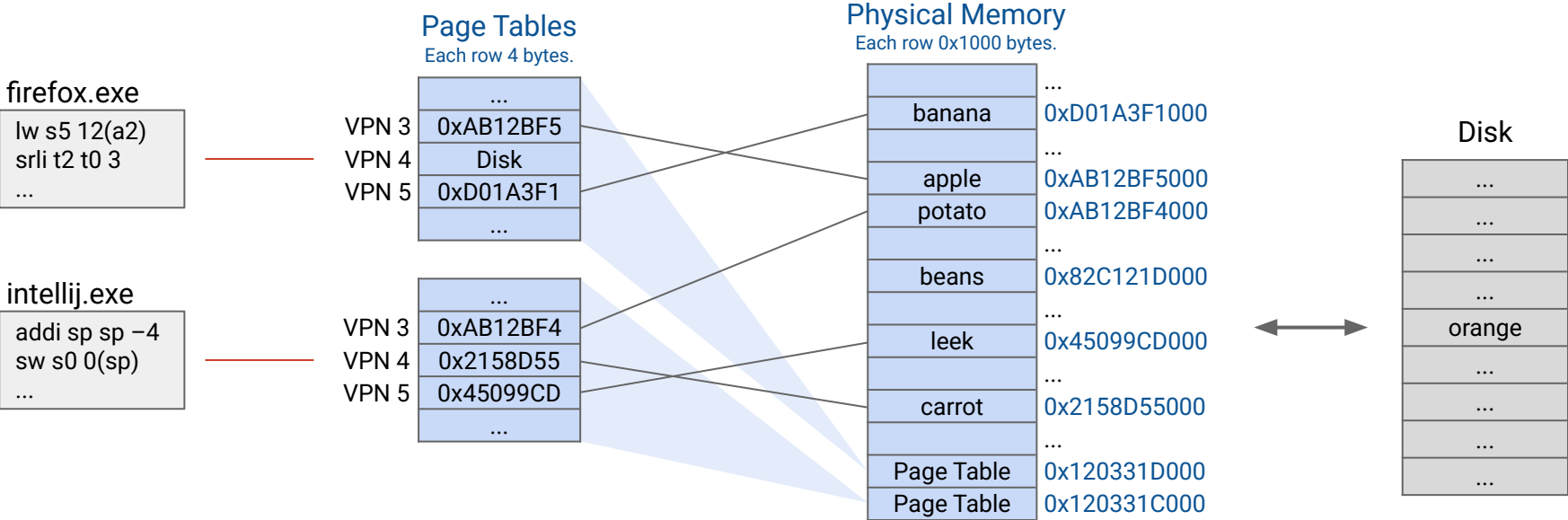
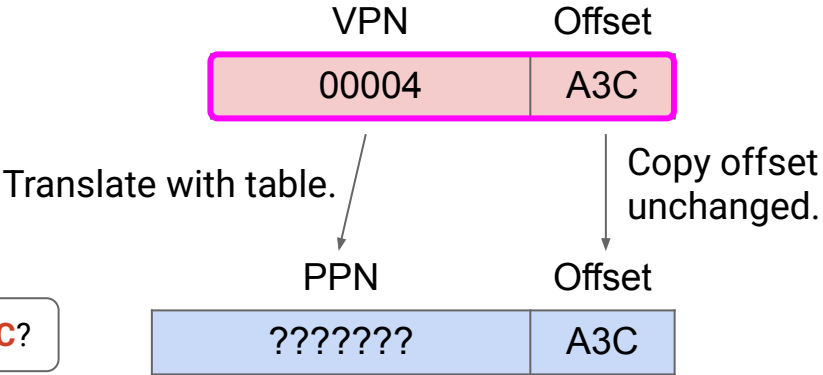
Disk

...
...
...
...
orange
...
...
...

Step 2: Extract VPN and offset.

Use system parameters (page size, virtual memory size) to decide number of VPN bits and number of offset bits.

What is the byte at the **virtual address 0x00004A3C**?

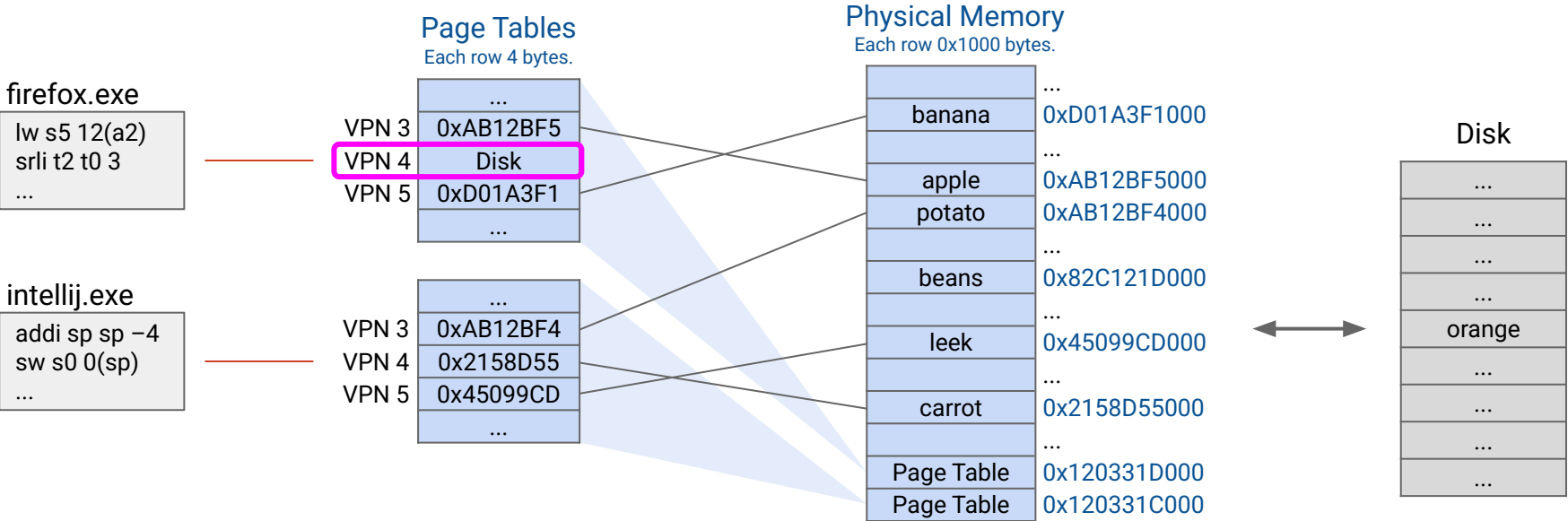
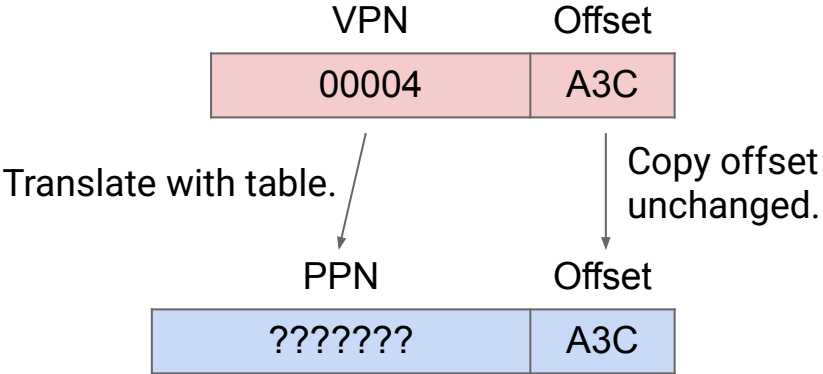


# Page Fault

Step 3: Translate VPN to PPN using table.

Look up the VPN'th index in the table to find the corresponding PPN.

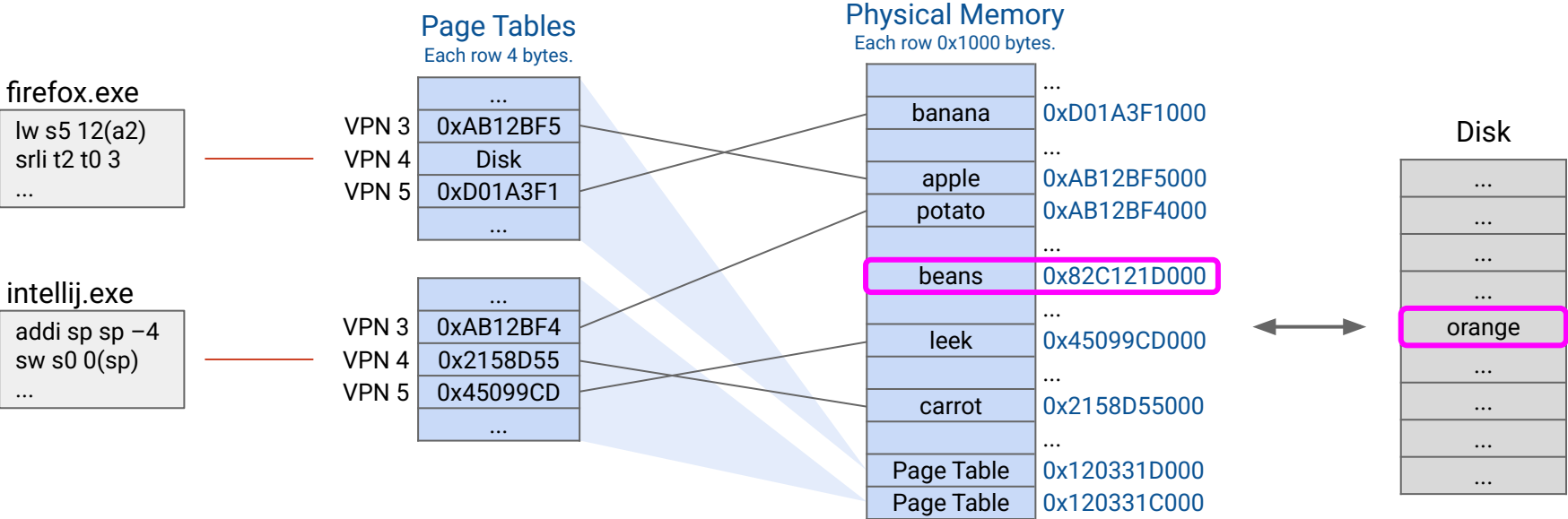
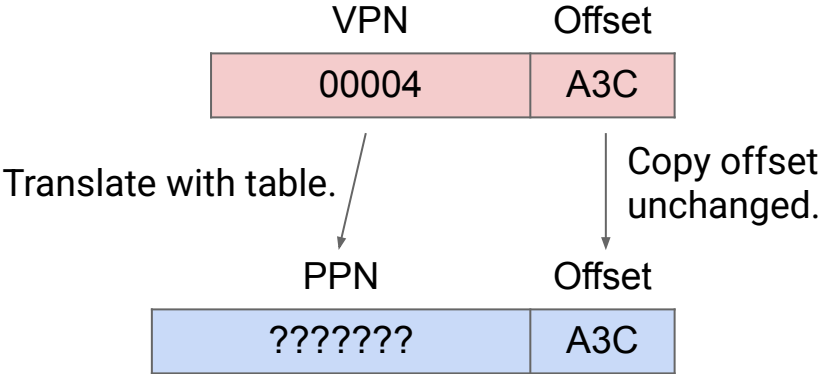
In this case, the entry says Disk, so we have a page fault!



Page fault!

Step 3.1: Load the page from disk into memory.

This might *evict* another page, kicking it out of memory.

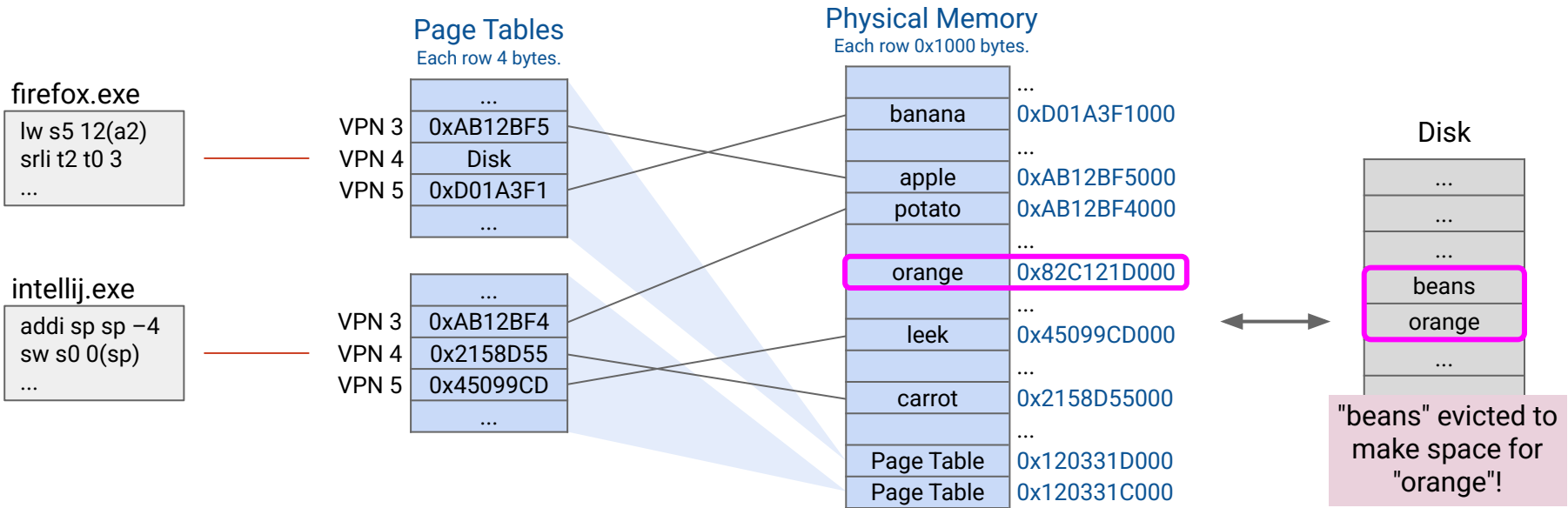
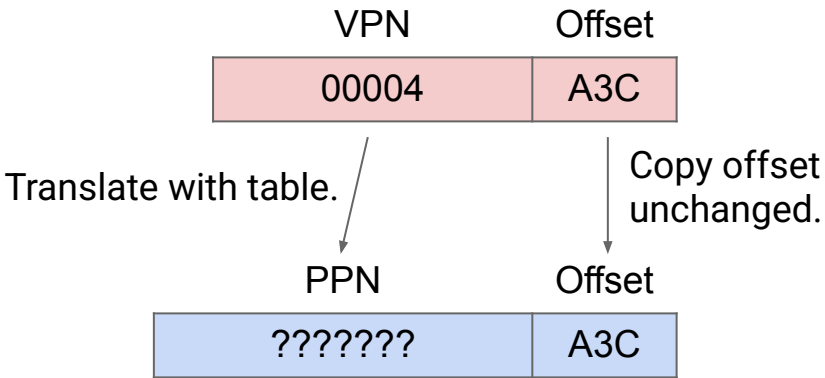




Page fault!

Step 3.1: Load the page from disk into memory.

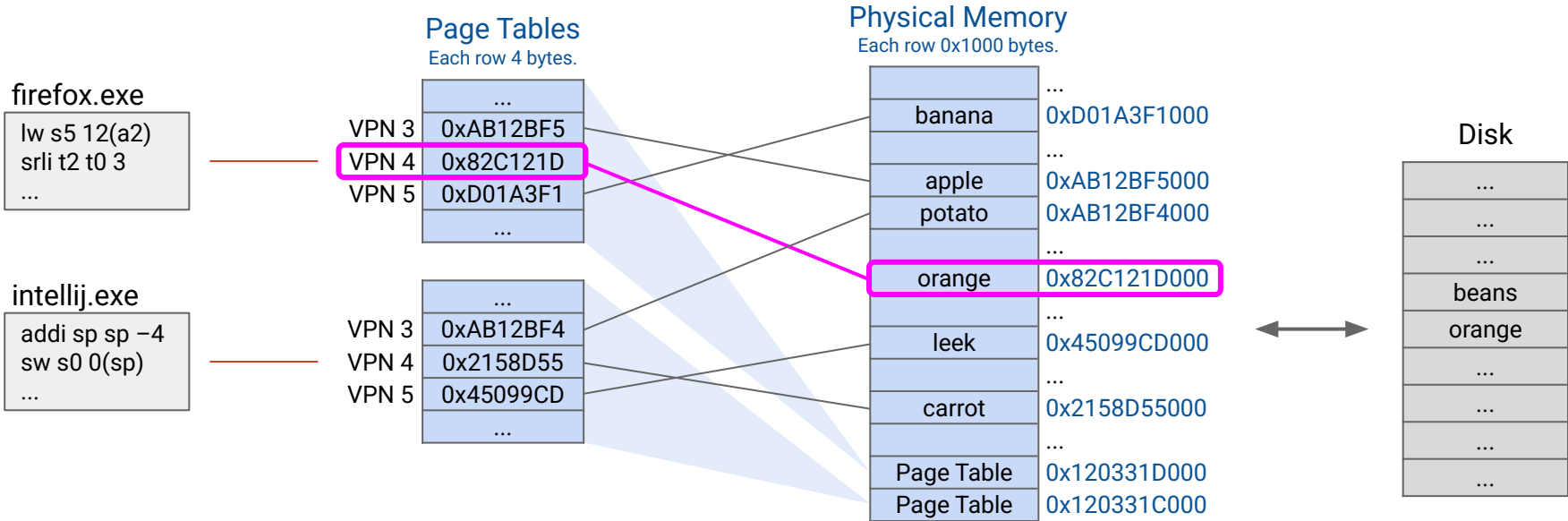
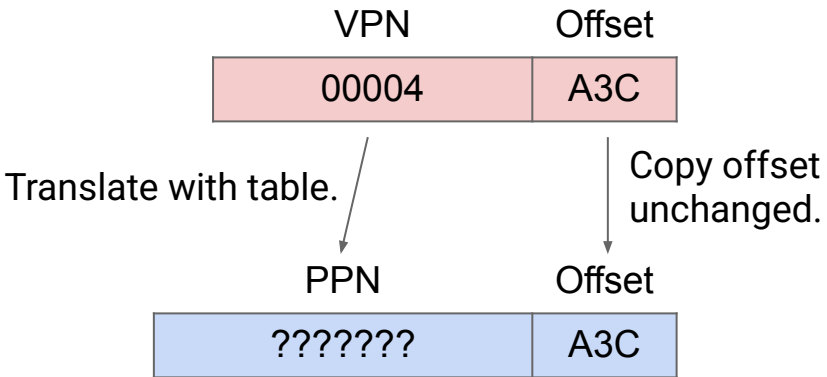
This might *evict* another page, kicking it out of memory.



Page fault!

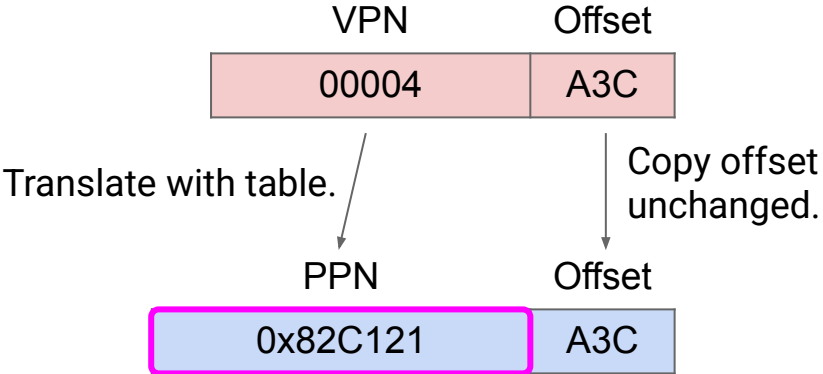
Step 3.2: Update the page table with the location (PPN) of the newly-loaded page.

Page fault resolved. Back to the usual steps.



Step 4: Concatenate PPN and offset.

Physical address is **0x82C121A3C**.



firefox.exe

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

intellij.exe

```
addi sp sp -4
sw s0 0(sp)
...
```

Page Tables

Each row 4 bytes.

...	...
VPN 3	0xAB12BF5
VPN 4	0x82C121D
VPN 5	0xD01A3F1
...	...

...	...
VPN 3	0xAB12BF4
VPN 4	0x2158D55
VPN 5	0x45099CD
...	...

Physical Memory

Each row 0x1000 bytes.

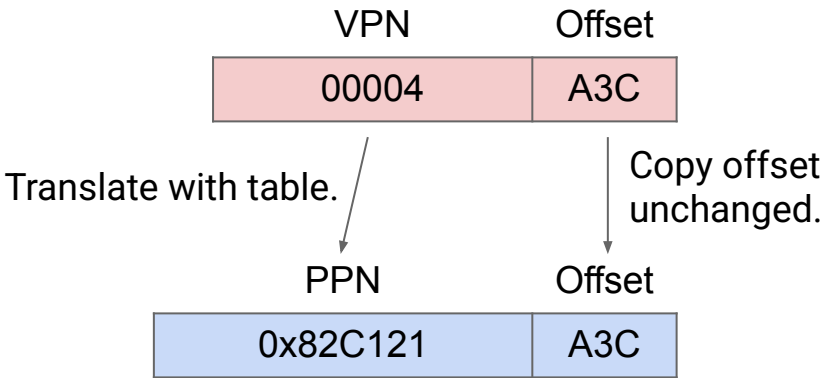
...	...
banana	0xD01A3F1000
...	...
apple	0xAB12BF5000
potato	0xAB12BF4000
...	...
orange	0x82C121D000
...	...
leek	0x45099CD000
...	...
carrot	0x2158D55000
...	...
Page Table	0x120331D000
Page Table	0x120331C000

Disk

...
...
...
beans
orange
...
...
...
...

Step 5: Return data to the program.  
Requested page says "orange."  
Return the 0xA3C'th byte on that page.

(Remember, we used "orange" to represent a page's worth of arbitrary data.)



firefox.exe

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

intellij.exe

```
addi sp sp -4
sw s0 0(sp)
...
```

Page Tables

Each row 4 bytes.

	...
VPN 3	0xAB12BF5
VPN 4	0x82C121D
VPN 5	0xD01A3F1
	...

Physical Memory

Each row 0x1000 bytes.

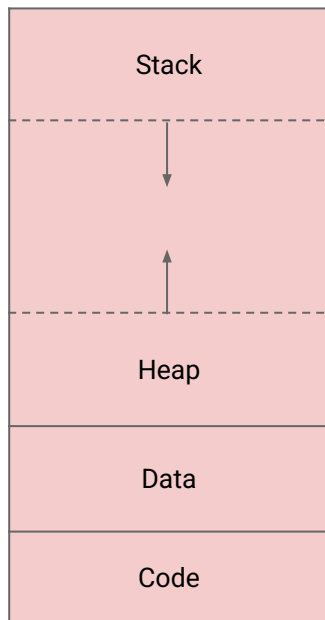
...	...
banana	0xD01A3F1000
...	...
apple	0xAB12BF5000
potato	0xAB12BF4000
...	...
orange	0x82C121D000
...	...
leek	0x45099CD000
...	...
carrot	0x2158D55000
...	...
Page Table	0x120331D000
Page Table	0x120331C000

Disk

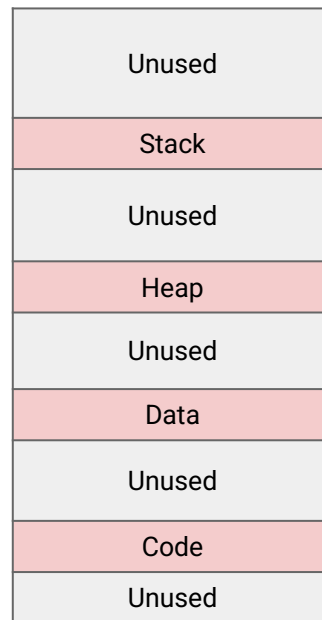
...
...
...
beans
orange
...
...
...
...

Loading a ton of pages for every new program is wasteful.

- What if the program is just "hello world"? Most pages are never used.



Our VM abstraction allows a program to use the full virtual address space...



...but some programs use only a tiny amount of memory.

## Demand Paging

---

Loading a ton of pages for every new program is wasteful.

- What if the program is just "hello world"? Most pages are never used.

Solution: **Demand paging**.

- Only load a page into memory if the user requests it.
- When a program starts, the page table says "Disk" for every entry.

firefox.exe's Page Table	
VPN	PPN
0x00001	Disk
0x00002	Disk
0x00003	Disk
0x00004	Disk
...	...

## Valid Bit

Instead of writing "Disk" in the table, we use a **valid bit**.

- Valid bit = 1: The page is in memory.  
Read the PPN to learn the corresponding physical address.
- Valid bit = 0: The page is on disk. Page fault!  
The PPN value is garbage.

firefox.exe's Page Table	
VPN	PPN
...	...
0x00004	0x60C25E6
0x00005	Disk
0x00006	0xEC70DB7
...	...



firefox.exe's Page Table		
Valid?	VPN	PPN
...	...	...
1	0x00004	0x60C25E6
0	0x00005	0x8173023
1	0x00006	0xEC70DB7
..	...	...

Garbage value.

# VM Design Choices

---

Lecture 33, CS 61C, Fall 2024

Running Multiple Programs

Virtual and Physical Addresses

Page Tables

Memory Access Example

Virtual Memory Parameters

Memory Paging

## **VM Design Choices**

Benefits of Virtual Memory



# Eviction Policies

On a page fault, if memory is full, we need to *evict* a page.

- Recall: Page fault = load a page from disk into memory.
- Recall: Evict = kick a page out of memory into disk.

How does the OS select which page to evict?

- LRU: Least Recently Used.
- FIFO: First-In, First-Out.
- Random.

## Physical Memory

Each row 0x1000 bytes.

	...
banana	0xD01A3F1000
	...
apple	0xAB12BF5000
potato	0xAB12BF4000
	...
orange	0x82C121D000
	...
leek	0x45099CD000
	...
carrot	0x2158D55000
	...
Page Table	0x120331D000
Page Table	0x120331C000



## Disk

...
...
...
beans
orange
...

"beans" evicted to  
make space for  
"orange"!

What if the user wants to write data?

Option 1: **Write-through.**

- Update the page in memory. Also, update the page on disk.
- Pro: Simple. Pages are always in sync.
- Con: Expensive. Disk update on every write.

Option 2: **Write-back.**

- Update the page in memory. Don't update the page on disk immediately.
- When the page is evicted from memory, update the page on disk.
- Pro: Cheap. Don't need to update disk every time.
- Con: Complicated. Memory and disk can get out of sync.

All VM systems use **write-back**.

- Disk accesses take too long. Disk update on every write is way too slow.
- Recall memory analogy: Disk is like going to Pluto.

Write-back requires a dirty bit to keep track of pages that have an unsynced write.

- When evicting a page: If dirty bit = 1, update disk.

When physical page 0x8173023 is evicted,  
we need to update disk.



firefox.exe's Page Table			
Dirty?	Valid?	VPN	PPN
...	...	...	...
0	1	0x00004	0x60C25E6
1	1	0x00005	0x8173023
0	1	0x00006	0xEC70DB7
...	..	...	...

# Benefits of Virtual Memory

---

Lecture 33, CS 61C, Fall 2024

Running Multiple Programs

Virtual and Physical Addresses

Page Tables

Memory Access Example

Virtual Memory Parameters

Memory Paging

VM Design Choices

**Benefits of Virtual Memory**

## Benefits of Virtual Memory

---

We can run programs that require more memory than the computer has.

- Memory paging gives us the illusion of a larger memory.

With **demand paging**, smaller programs don't waste memory.

- Only the pages that actually get used are brought into memory.

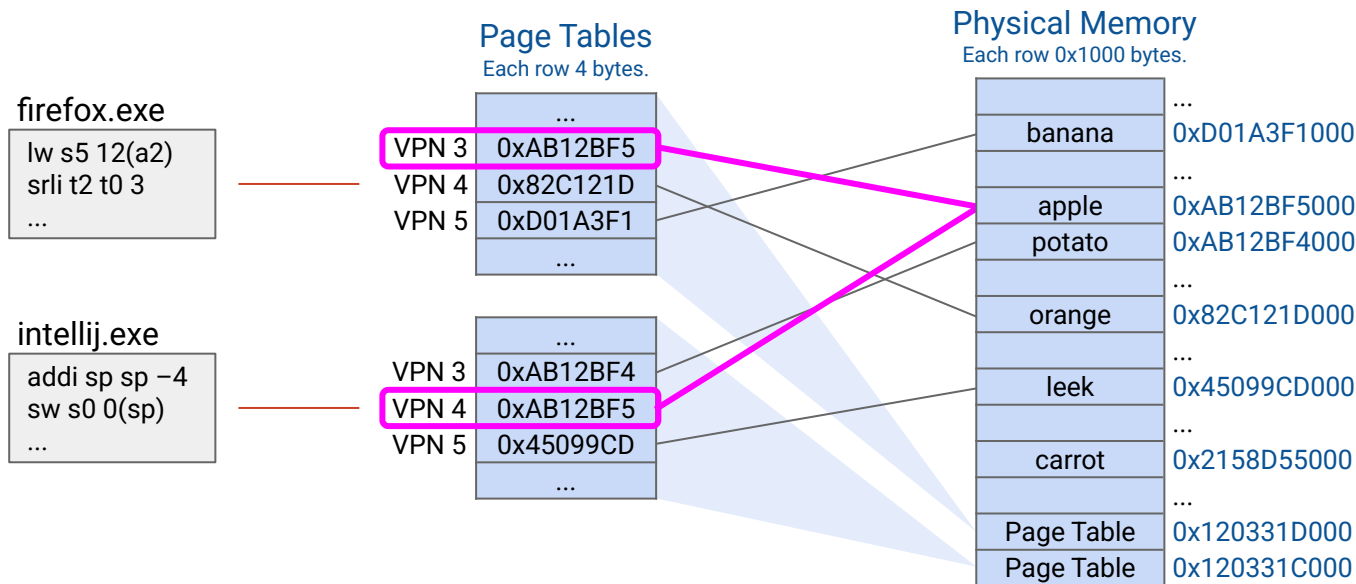
Allows OS to enforce **isolation** between programs.

- Every program has the illusion of its own dedicated (virtual) address space.
- firefox.exe cannot access the memory of intellij.exe.
- Even if both programs access the same virtual address, they map to different physical addresses.
- Errors in one program won't corrupt the memory of other programs.

# Memory Sharing

Multiple programs could share memory.

- Just direct both programs to the same physical page.
- Example: Programs might need the same library code, e.g. `stdlib.h`.



## Write Protection

Allows OS to **protect** certain pages.

- OS can mark specific pages as read-only by setting a bit in the page table.
- Writing to a protected page triggers an exception.  
Exceptions are handled by the OS (more later).
- Example: We could mark the code section as read-only.  
Stops the program from overwriting itself while it's executing.

If a user tries to write to VPN 0x00004, —————→  
the OS will not allow it.

firefox.exe's Page Table				
Read-only?	Dirty?	Valid?	VPN	PPN
...	...	...	...	...
1	0	1	0x00004	0x60C25E6
0	1	1	0x00005	0x8173023
0	0	1	0x00006	0xEC70DB7
...	...	..	...	...

# Summary: Virtual Memory I

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

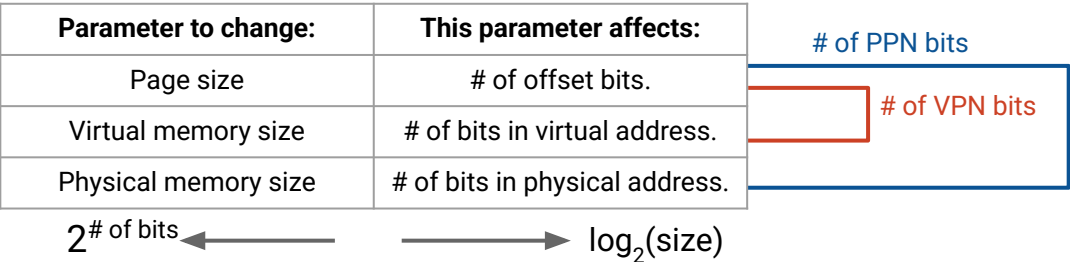
Step 2: Extract VPN and offset. Use parameters to decide # of VPN bits and # of offset bits.

Step 3: Translate VPN to PPN using table.  
Look up the VPN'th index in the table to find the corresponding PPN.  
If page fault (entry is invalid), do Steps 3.1–3.2.

Step 4: Concatenate PPN and offset.

Step 5: Return data to the program.

Step 3.1: Load page from disk into memory. This might evict another page (kick it out of memory).  
Step 3.2: Update the page table with the location (PPN) of the newly-loaded page.



firefox.exe

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

intellij.exe

```
addi sp sp -4
sw s0 0(sp)
...
```

Page Tables  
Each row 4 bytes.

...	...
VPN 3	0xAB12BF5
VPN 4	0x82C121D
VPN 5	0xD01A3F1
...	...

Physical Memory  
Each row 0x1000 bytes.

...	...
banana	0xD01A3F1000
...	...
apple	0xAB12BF5000
potato	0xAB12BF4000
...	...
orange	0x82C121D000
...	...
leek	0x45099CD000
...	...
carrot	0x2158D55000
...	...
Page Table	0x120331D000
Page Table	0x120331C000

Page table lives in physical memory!

Size of page table = # of PTEs × Size per PTE  
=  $2^{\text{\# of VPN bits}}$  × # of PPN bits

Disk

beans
orange
...

