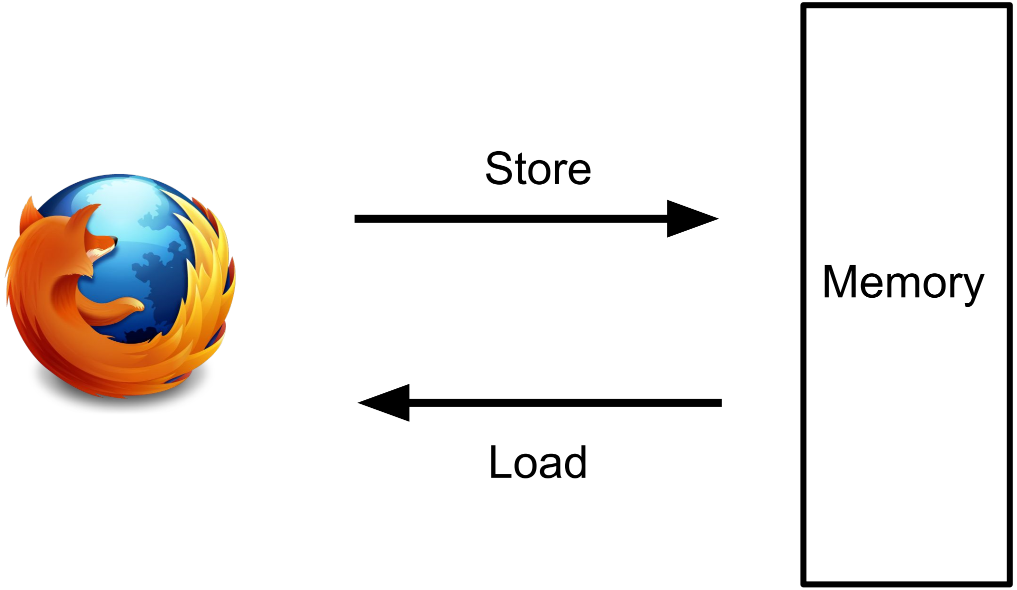
Virtual Memory for Security

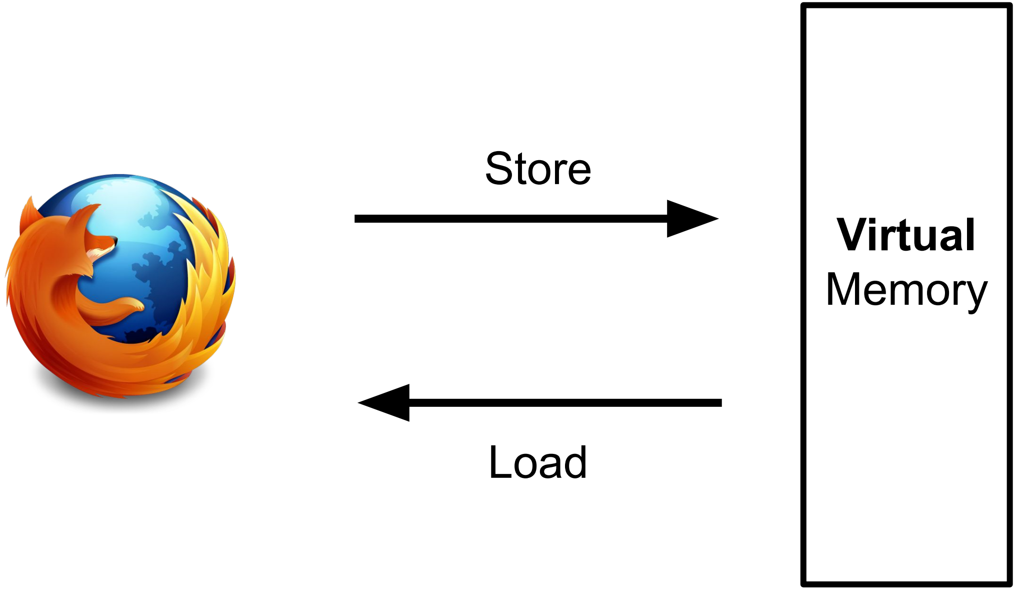
Noah Brown - slides adapted from Onur Mutlu and Don Porter

Virtual Memory

Process' View of Memory

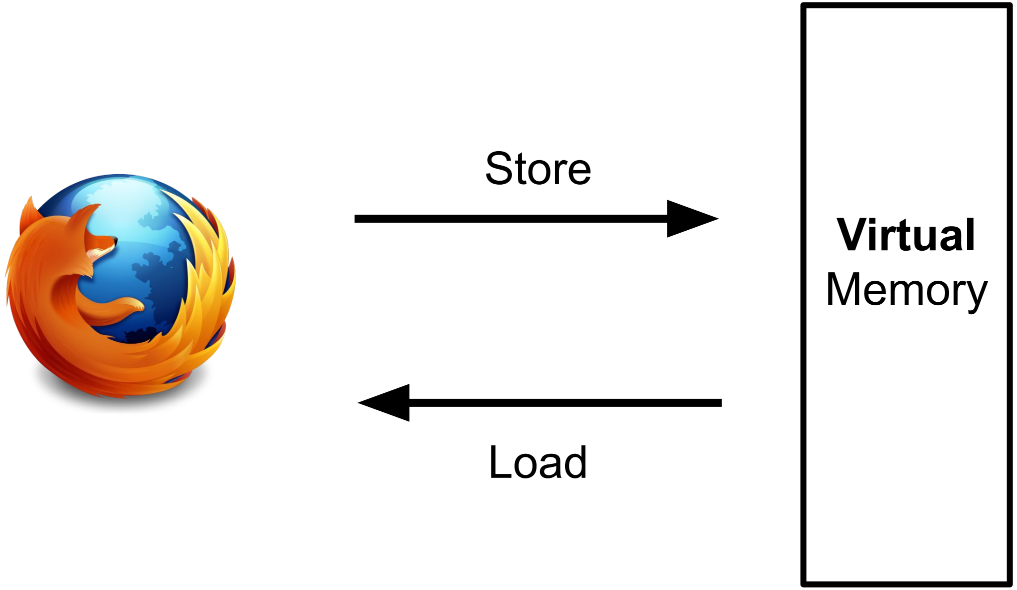


Process’ View of Memory



“Infinite” capacity

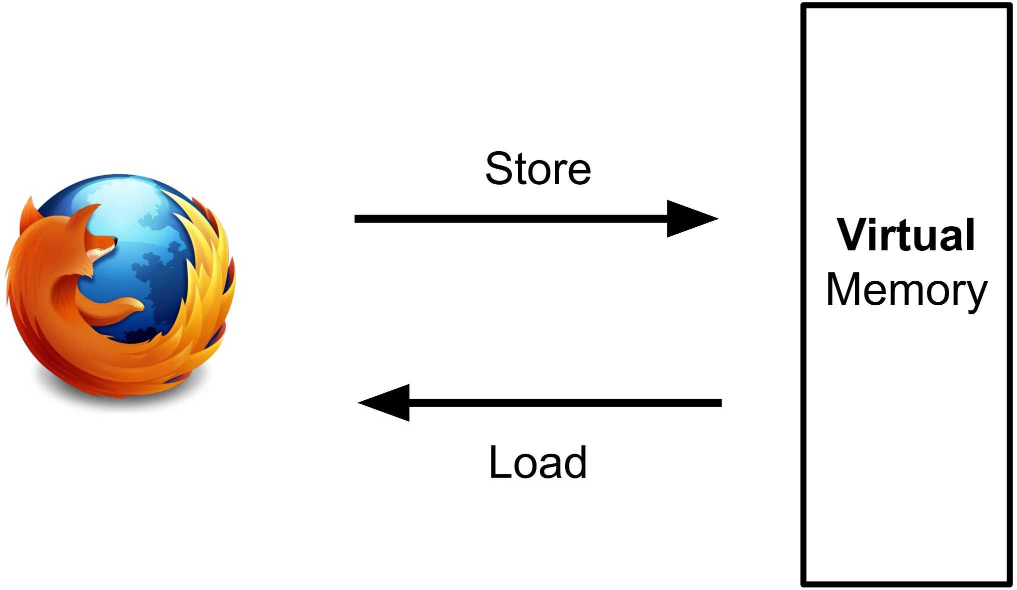
Process’ View of Memory



“Infinite” capacity

In practice, **physical** memory is much smaller

Process' View of Memory

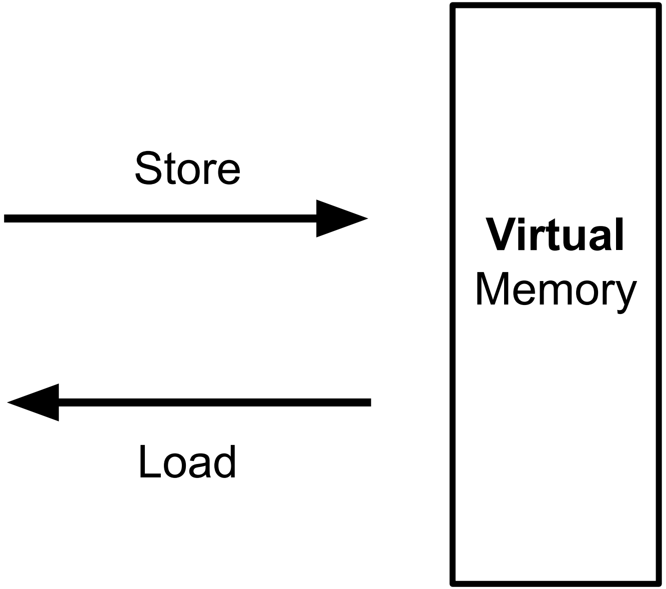


"Infinite" capacity

In practice, physical memory is much smaller

The OS maps virtual to physical transparently

Process’ View of Memory



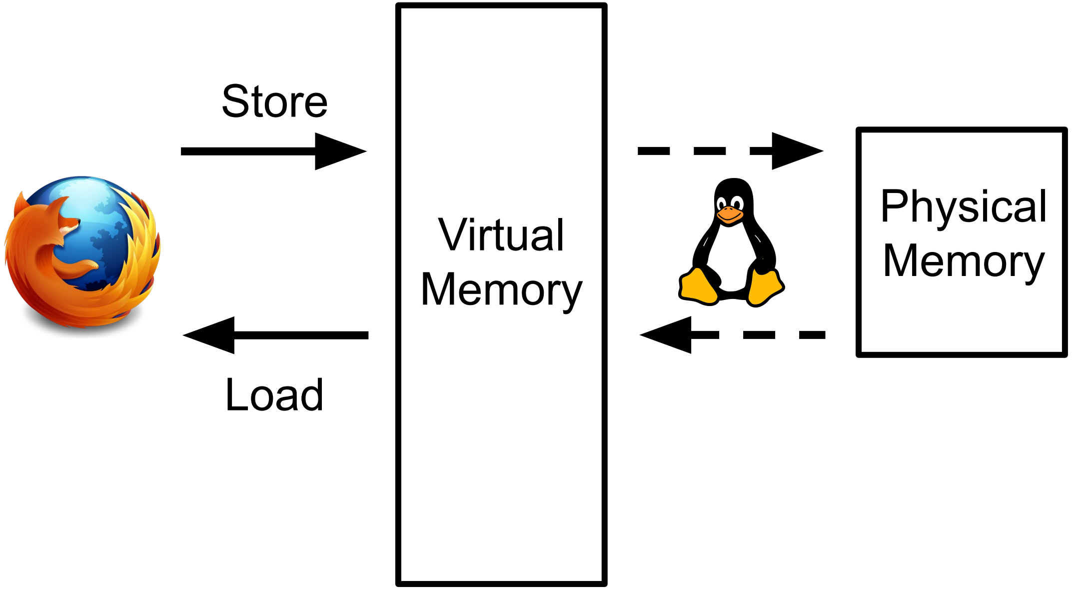
“Infinite” capacity

In practice, physical memory is much smaller

The OS maps virtual to physical transparently

How can we maintain this illusion?

Process’ View of Memory



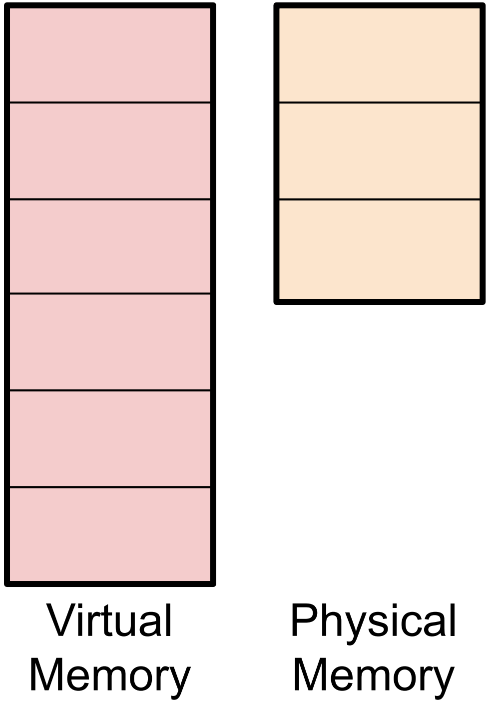
“Infinite” capacity

In practice, physical memory is much smaller

The OS maps virtual to physical transparently

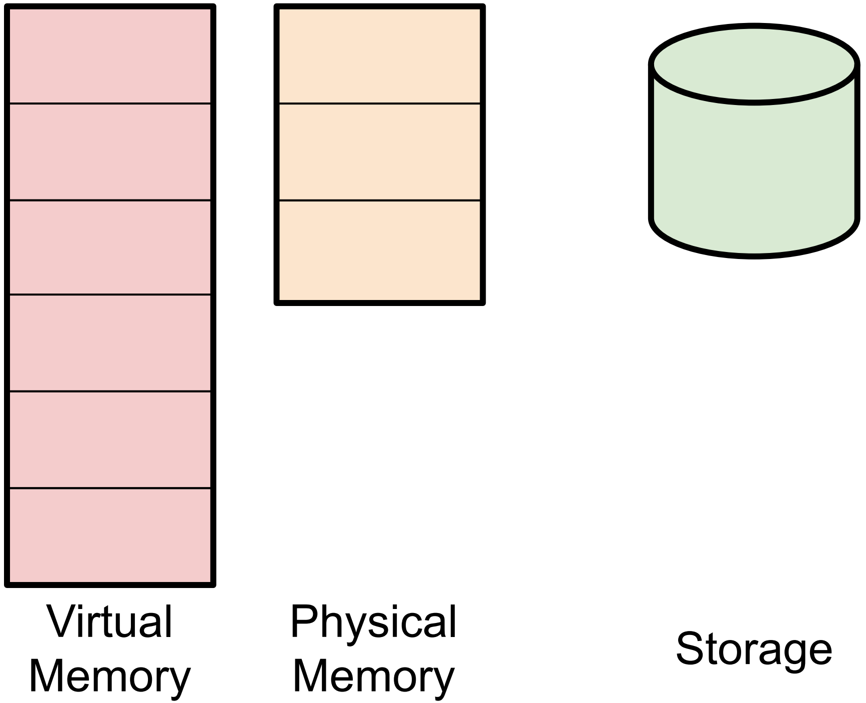
How can we maintain this illusion?

Virtualization of Memory



Segment virtual/physical memory into *pages* and *frames*

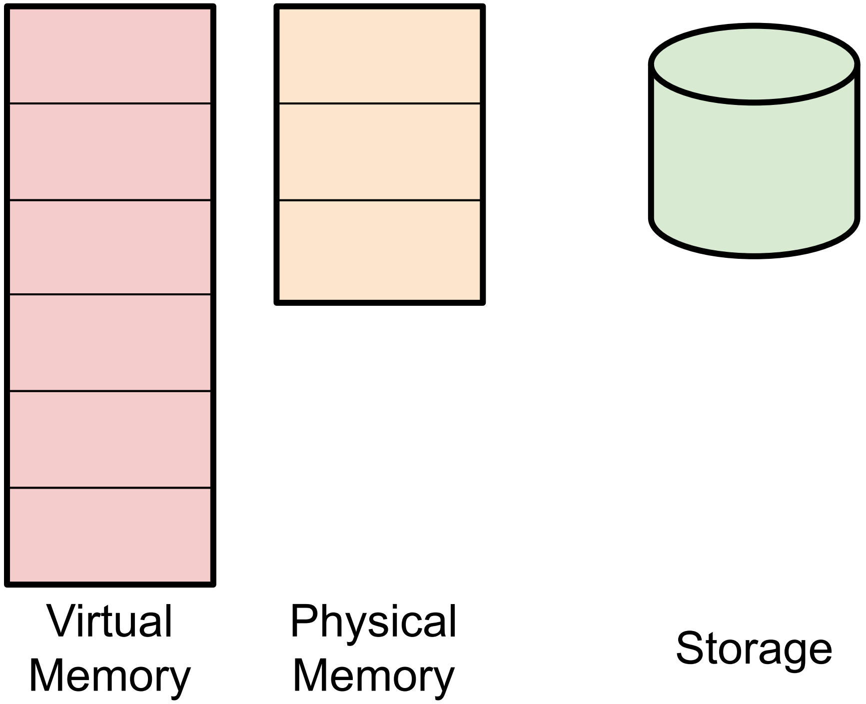
Virtualization of Memory



Segment virtual/physical memory into *pages* and *frames*

Pages are either in physical memory or out on disk

Virtualization of Memory



Segment virtual/physical memory into *pages* and *frames*

Pages are either in physical memory or out on disk

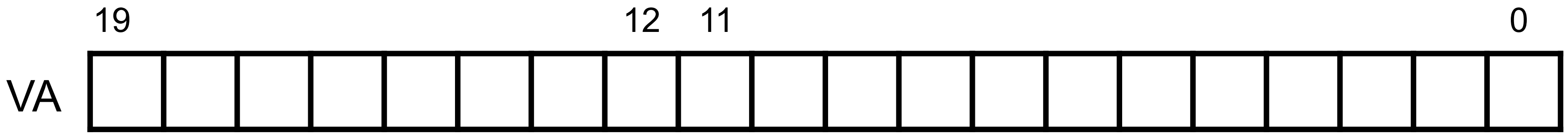
To the process, memory is contiguous and plentiful

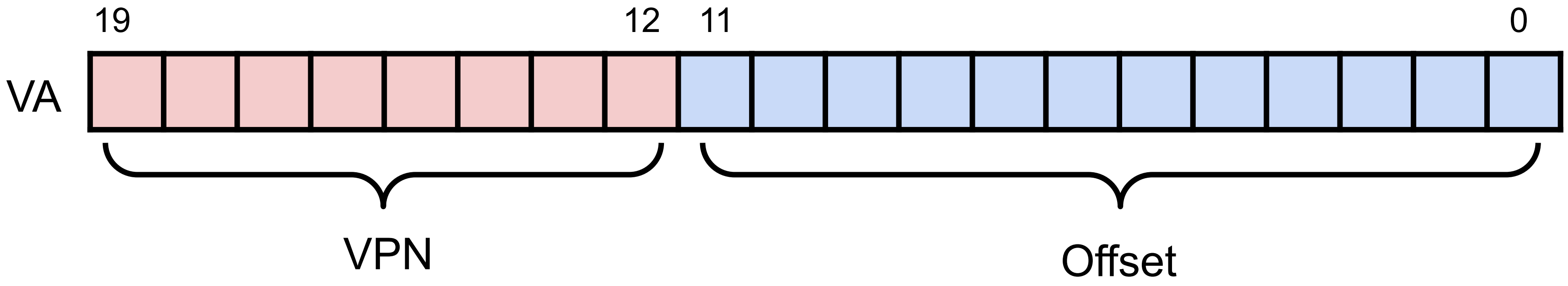
A Real World Analogy

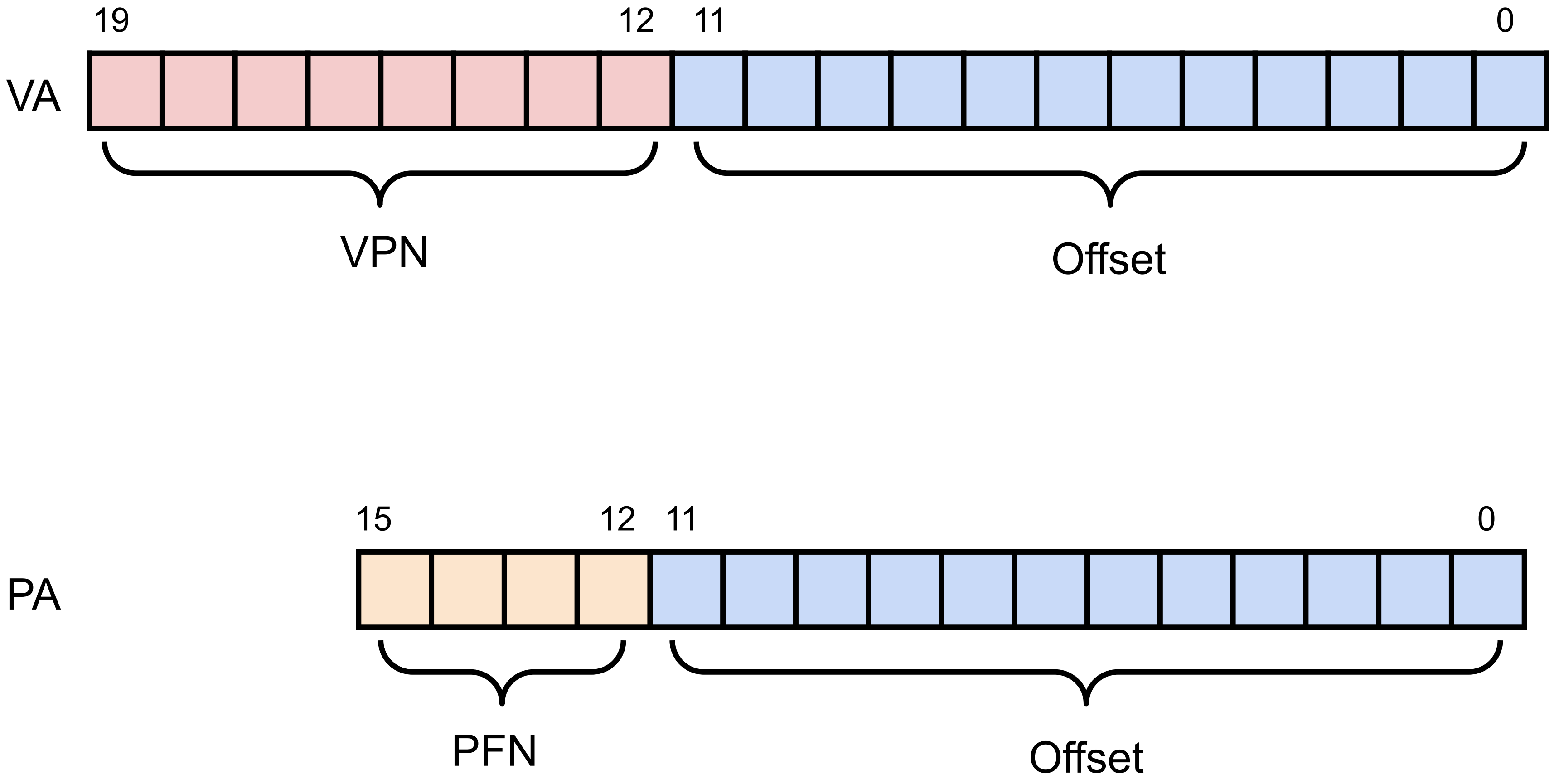
- Pieces of paper: physical pages
- All of you: processes “doing your thing”, no need for memory right now
- Disk: giant whiteboard in back of room
- Myself: the OS

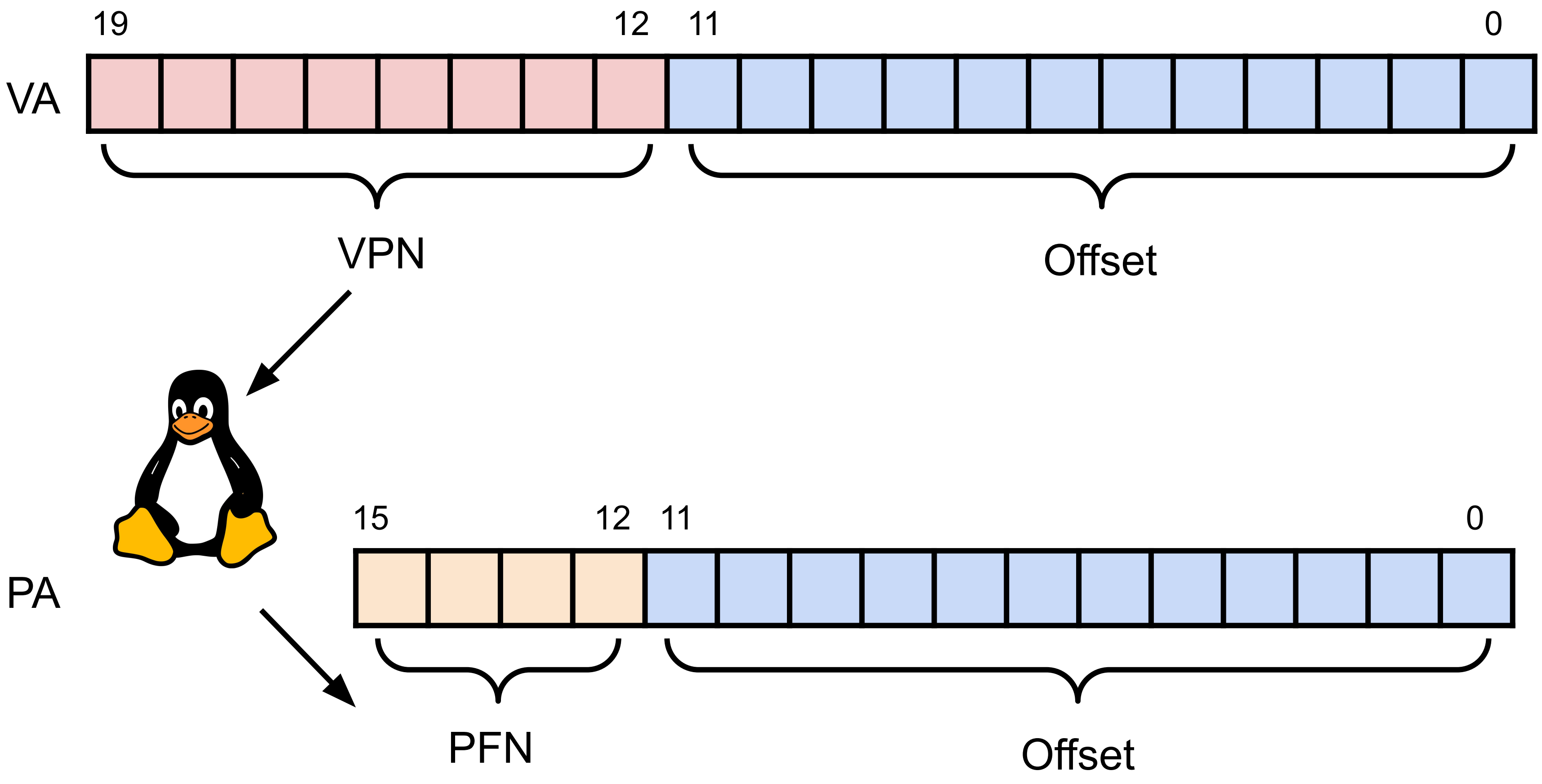
Virtual Addresses

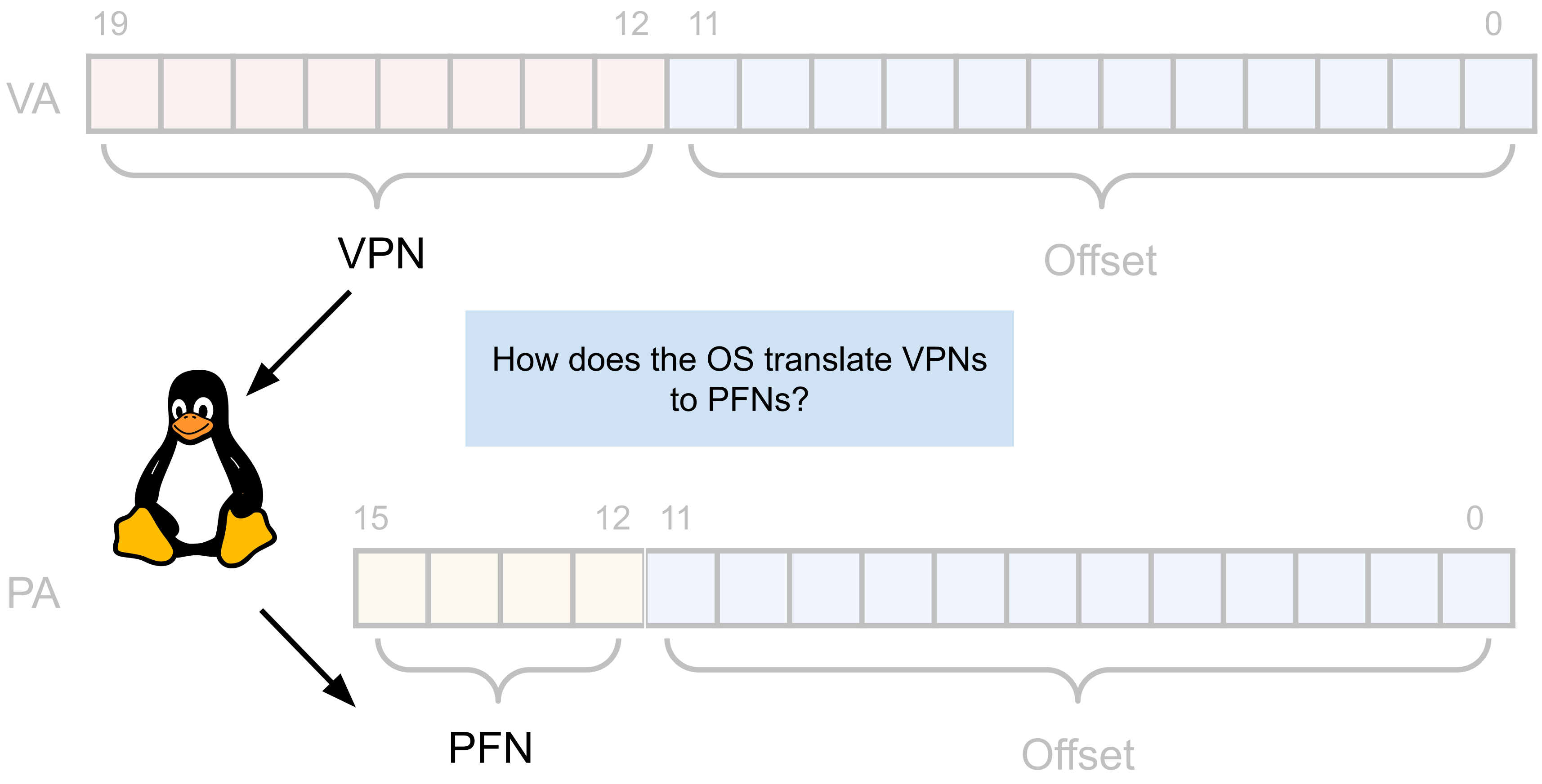
- Processes only ever see *virtual* addresses
  - No physical backing until a frame is mapped
- The OS handles the conversion of virtual to physical
- Example
  - 1 MB (20-bit) VAs
  - 64 KB (16-bit) PAs
  - 4 KB (12-bit) pages

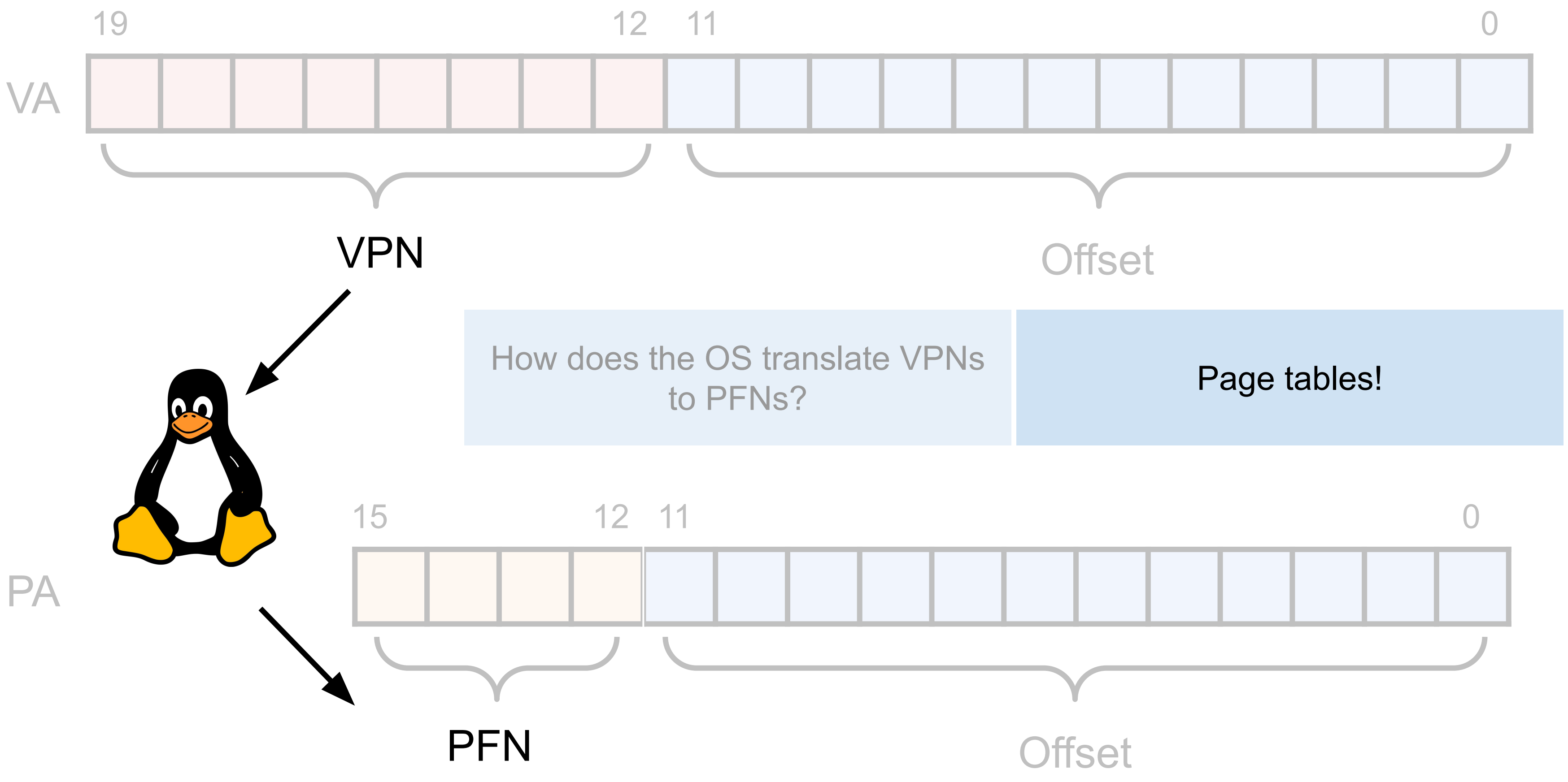

19
12
11
0
VA


19
12
11
0
VA
VPN
Offset


19
12
11
0
VA
VPN
Offset
15
12
11
0
PA
PFN
Offset


19
12
11
0
VA
VPN
Offset
15
12
11
0
PA
PFN
Offset


19
12
11
0
VA
VPN
Offset
How does the OS translate VPNs to PFNs?
15
12
11
0
PA
PFN
Offset


19
12
11
0
VA
VPN
Offset
How does the OS translate VPNs to PFNs?
Page tables!
15
12
11
0
PA
PFN
Offset

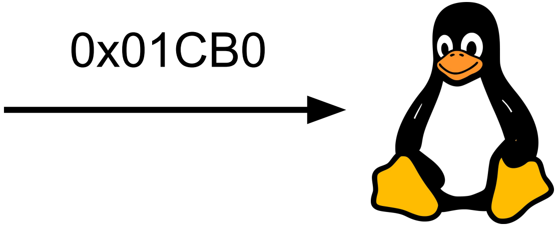
Page Tables

Page Tables

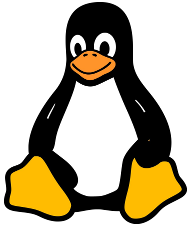
- 1 entry for each *virtual* page
- Each page table entry (PTE) has:
  - Valid bit - whether page is in memory
  - Physical page number - where page is in memory
  - A bunch of other “flags”

Page Tables

- The VPN says *where* to look in the page table
- Example:
  - VA: 0x04450
  - The page table translates page 04 to frame 6
    - Looks at the 04th entry in the table
  - PA: 0x6450

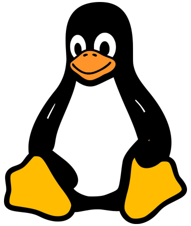

0x01CB0

0x01CB0



PFN	Valid
0xB	0
0xE	1
0x1	1
0x3	0
0x4	1
…	…
0x9	1
0xA	1
0x2	1
0x5	1
0x6	1

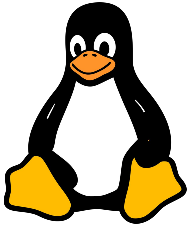
0x01CB0



PFN	Valid
0xB	0
0xE	1
0x1	1
0x3	0
0x4	1
…	…
0x9	1
0xA	1
0x2	1
0x5	1
0x6	1

0xECB0

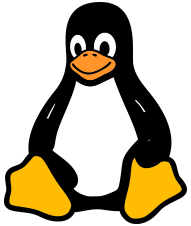
0x02B43



PFN	Valid
0xB	0
0xE	1
0x1	1
0x3	0
0x4	1
…	…
0x9	1
0xA	1
0x2	1
0x5	1
0x6	1

0x???

0x03FFE



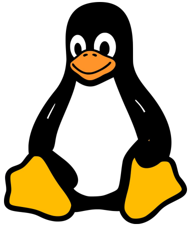
PFN	Valid
0xB	0
0xE	1
0x1	1
0x3	0
0x4	1
…	…
0x9	1
0xA	1
0x2	1
0x5	1
0x6	1

0x03FFE

PFN	Valid
0xB	0
0xE	1
0x1	1
0x3	0
0x4	1
…	…
0x9	1
0xA	1
0x2	1
0x5	1
0x6	1

0x?FFE

0x03FFE

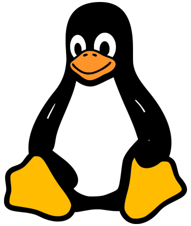


PFN	Valid
0xB	0
0xE	1
0x1	1
0x3	0
0x4	1
…	…
0x9	1
0xA	1
0x2	1
0x5	1
0x6	1

0x?FFE

If the valid bit isn’t set, then the PFN is meaningless

0x03FFE



PFN	Valid
0xB	0
0xE	1
0x1	1
0x3	0
0x4	1
…	…
0x9	1
0xA	1
0x2	1
0x5	1
0x6	1

0x?FFE

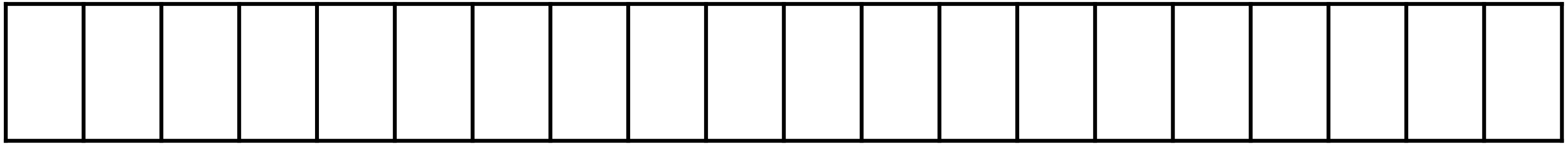
If the valid bit isn’t set, then the PFN is meaningless

If the page is out in storage, how do we update the page table?

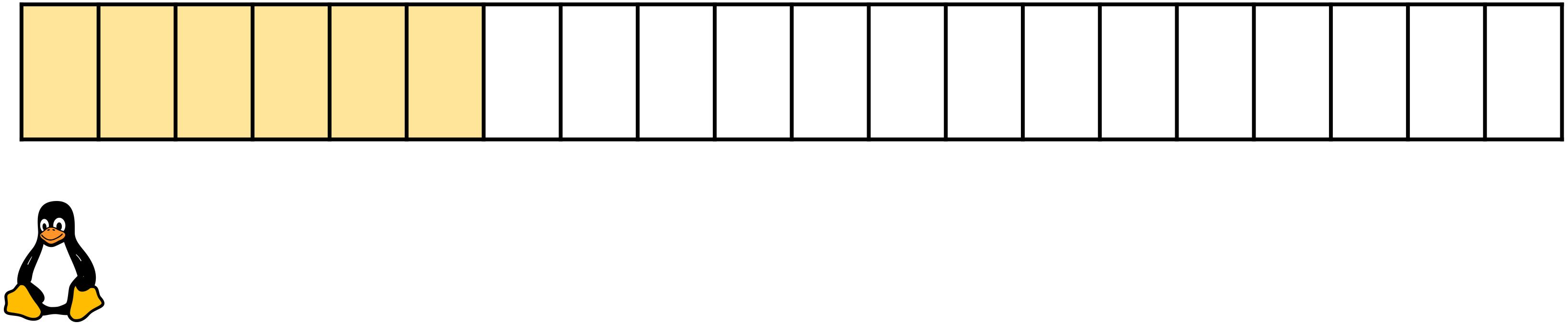
Page Faulting

- When data must be retrieved from storage first
- Then, have to find a free spot in memory for that data
- **Very slow** operation - order of ~ms
- If memory is full, have to **evict** something

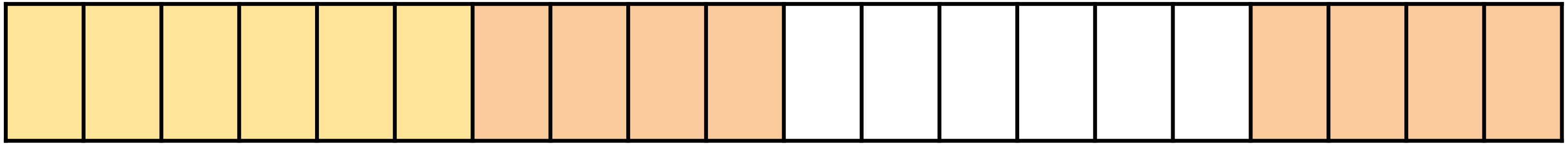
Physical Memory



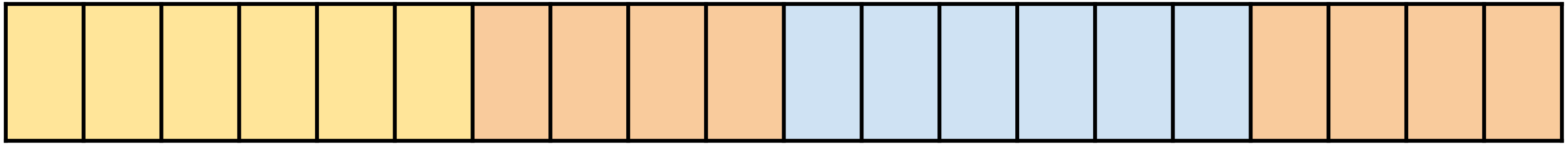
Physical Memory



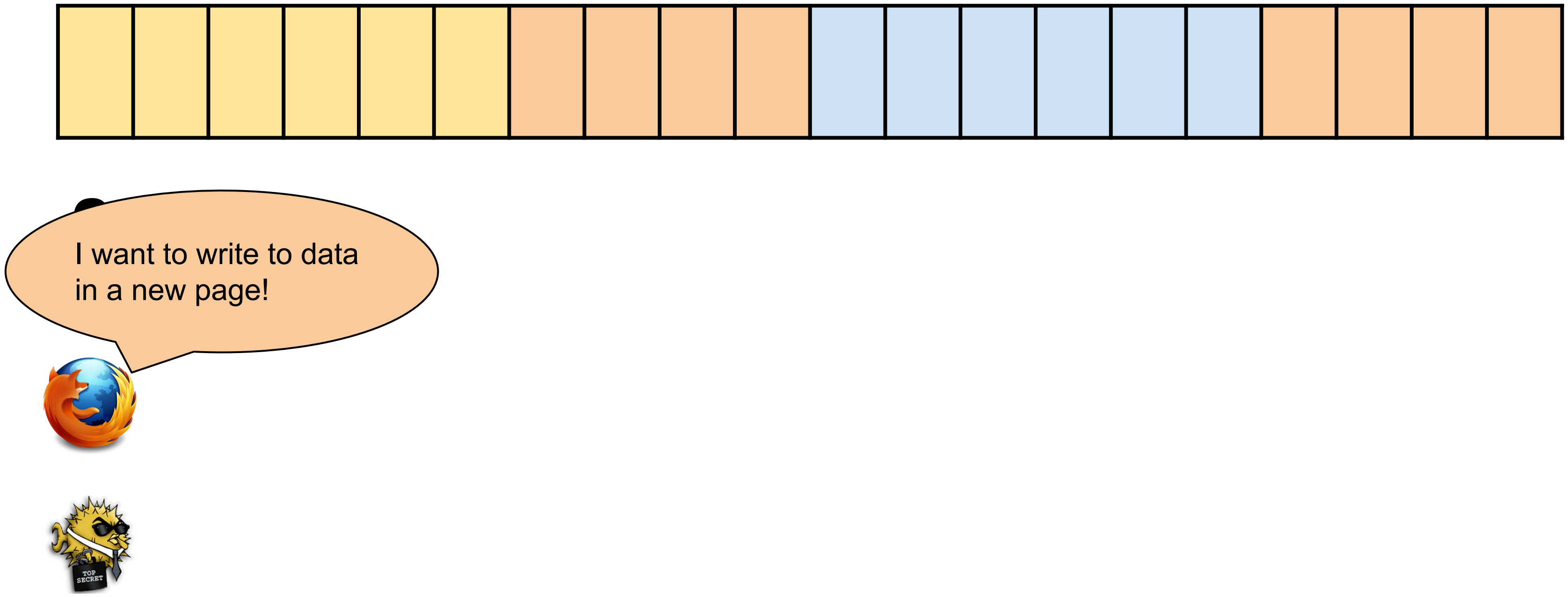
Physical Memory



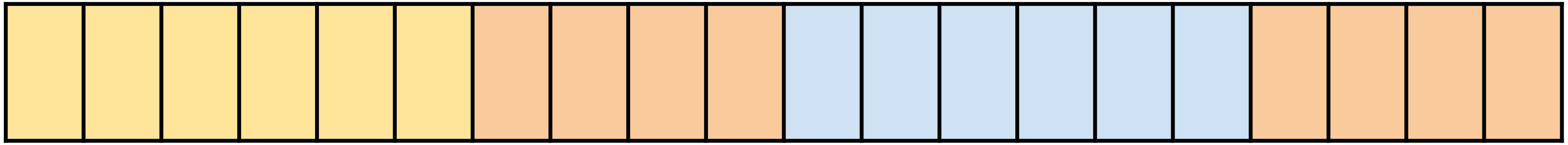
Physical Memory



Physical Memory



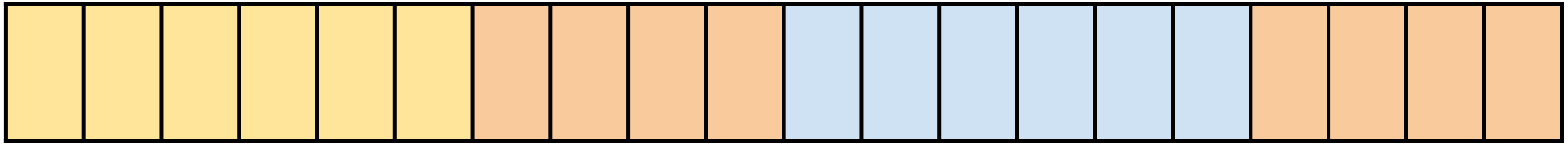
Physical Memory



I want to write to data
in a new page!

Memory is full…what do we do?

Physical Memory

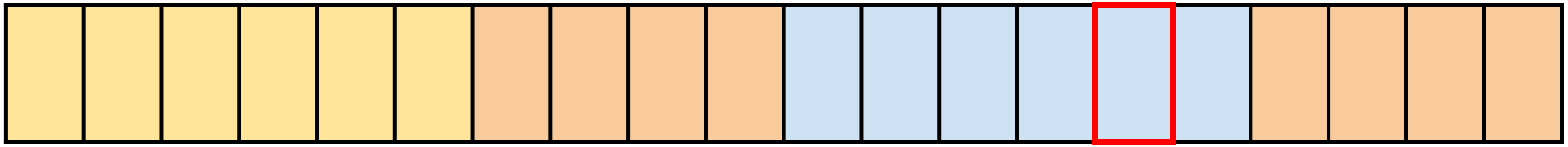


I want to write to data in a new page!

Memory is full…what do we do?

Need to pick a page to evict!

Physical Memory



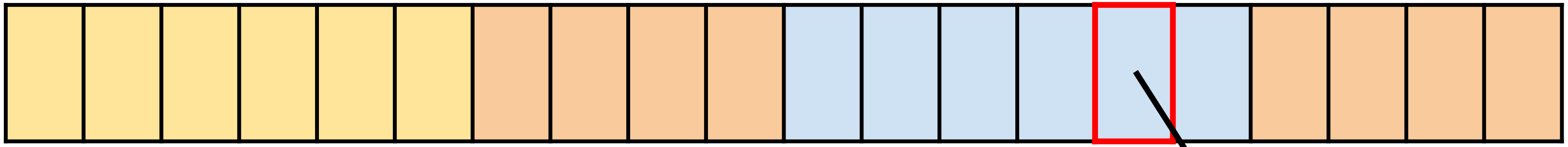
I want to write to data in a new page!

Memory is full…what do we do?

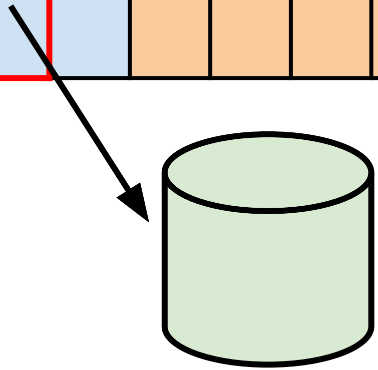
Need to pick a page to evict!

What do we need to do before handing the page over?

Physical Memory



I want to write to data in a new page!



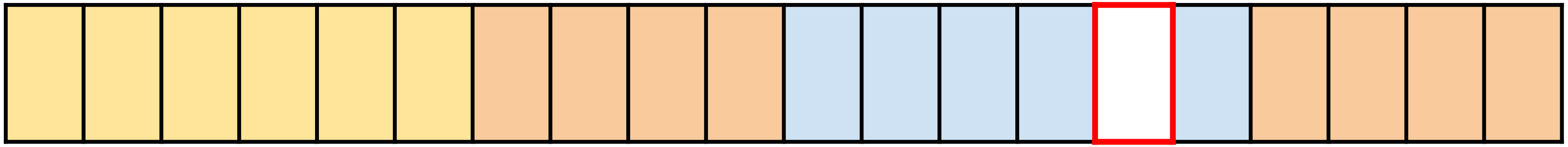
Memory is full…what do we do?

Need to pick a page to evict!

What do we need to do before handing the page over?

Write page out to disk!

Physical Memory



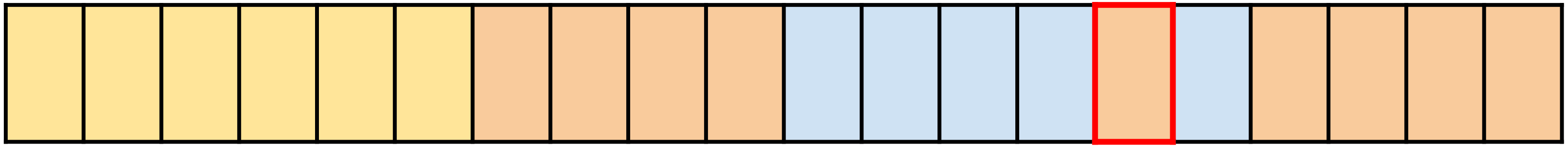
Memory is full…what do we do?

Need to pick a page to evict!

What do we need to do before handing the page over?

Write page out to disk!

Physical Memory



I want to write to data in a new page!

Memory is full…what do we do?

Need to pick a page to evict!

What do we need to do before handing the page over?

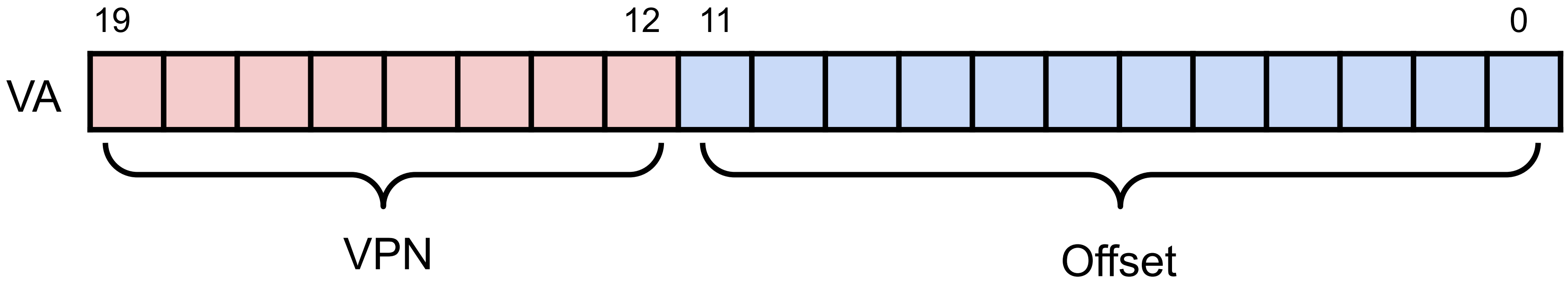
Write page out to disk!

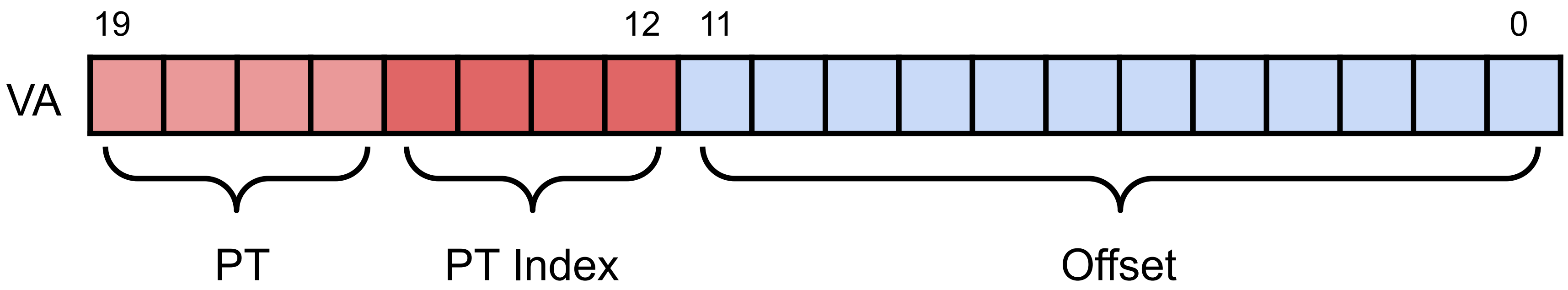
A Real World Analogy

- I have limited pieces of paper
- If somebody needs one, I have to take a page from somebody else
- **I can’t lose this person’s data!**
- So I write it down somewhere on the whiteboard

Further Complications

- Page tables are *big*
- Solution: break page table into many *smaller* tables


19
12
11
0
VA
VPN
Offset


19
12
11
0
VA
PT
PT Index
Offset

Further Complications

- Modern systems use 4 or 5-level paging
- Problem: now have to access 4 or 5 spots in memory…just to access data in memory (5-6x slowdown!)
- Solution: Translation Lookaside Buffer (TLB)
    - Acts as a “cache” to store most recently translated addresses

A Real World Analogy

- Keep a sticky note with me as I walk around the room
- Whenever somebody needs a page:
  - Check if its location is on sticky note
  - If not, *then* go to front pages and start figuring where it is

Page Size Trade-Offs

- Larger pages:
    - More frequent TLB hits
    - Smaller page tables
- Smaller pages:
    - Reduced wasting of memory

Page Size Trade-Offs

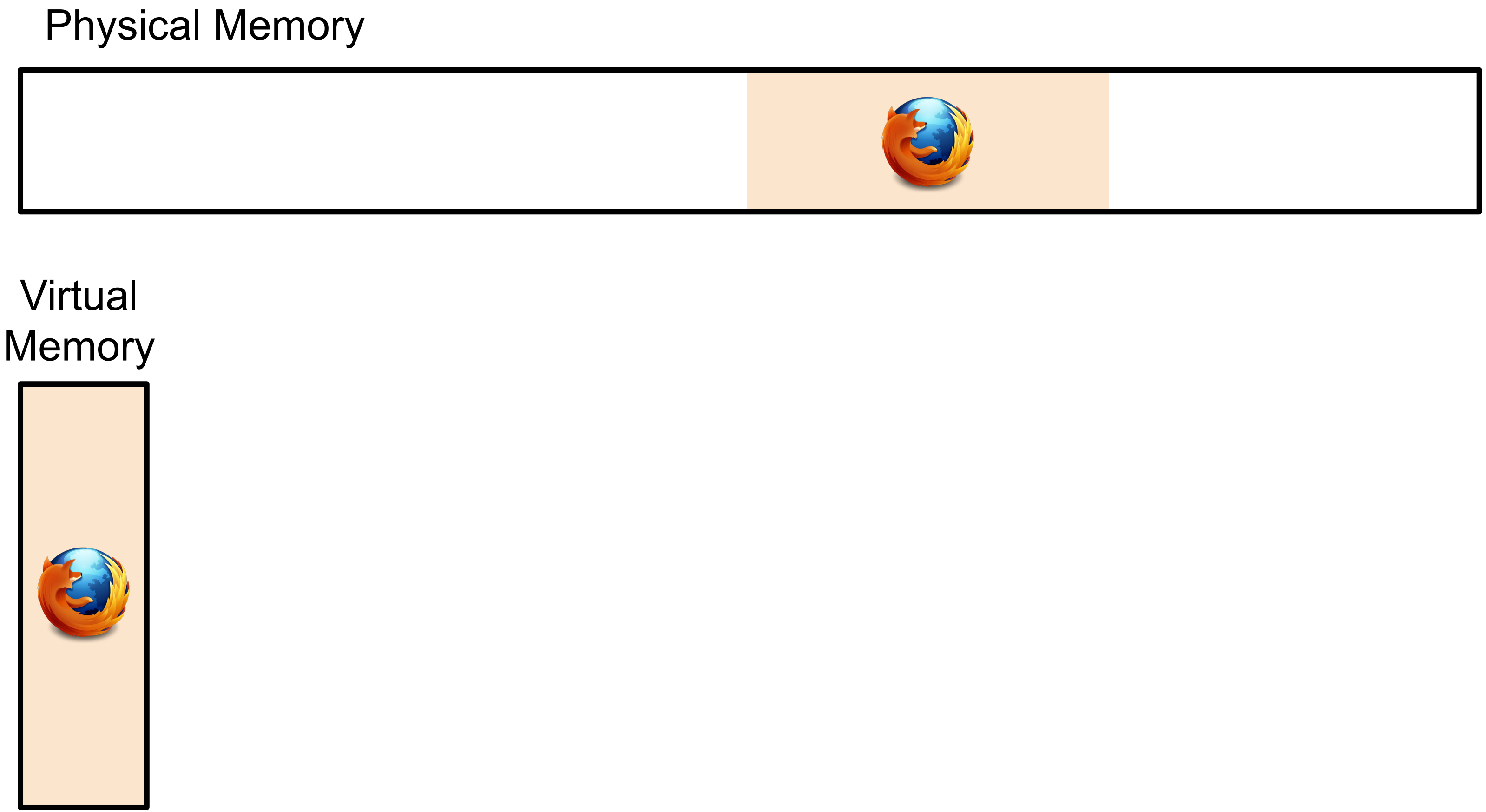
- In terms of *generating eviction sets*, remember that we assume we can't figure out the virtual-to-physical mapping
- Are bigger or smaller pages more useful for generating eviction sets?
- **Bigger** pages are helpful; more bits remain the same (offset doesn't change)

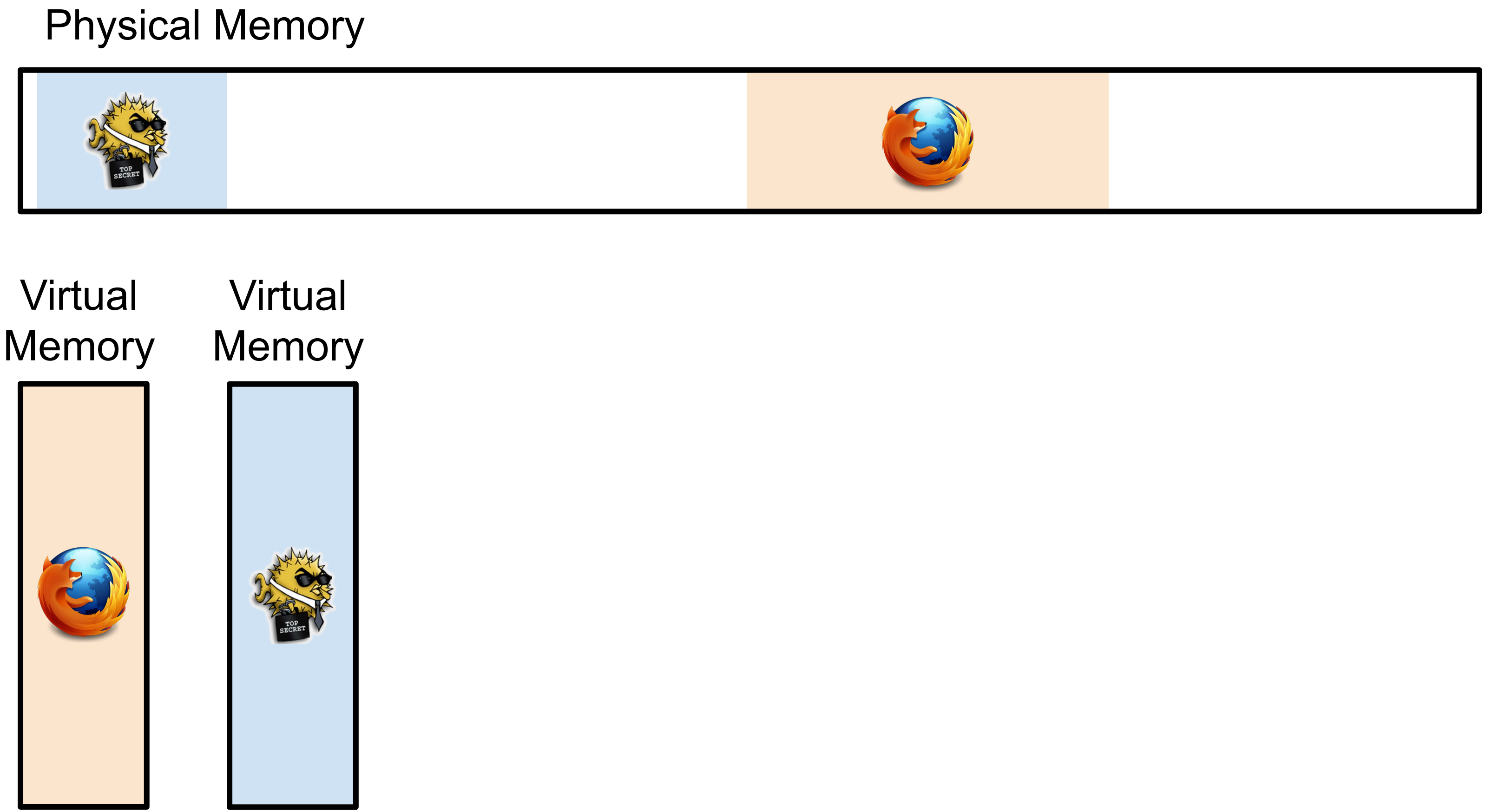
Memory Protection

Memory Protection

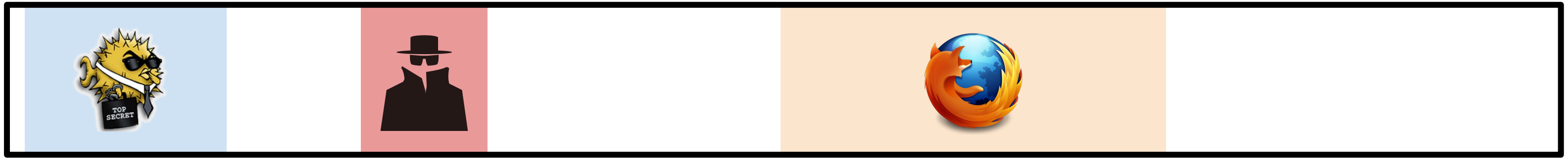
- Multiple processes can run simultaneously
  - The OS has a page table for each process
- A process can only access physical pages in its own page table

Physical Memory


Physical Memory
Virtual Memory


Physical Memory
TOP SECRET
Virtual Memory
Virtual Memory
TOP SECRET

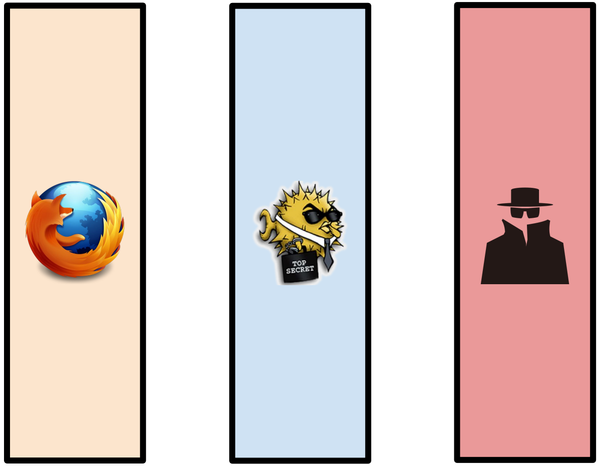
Physical Memory

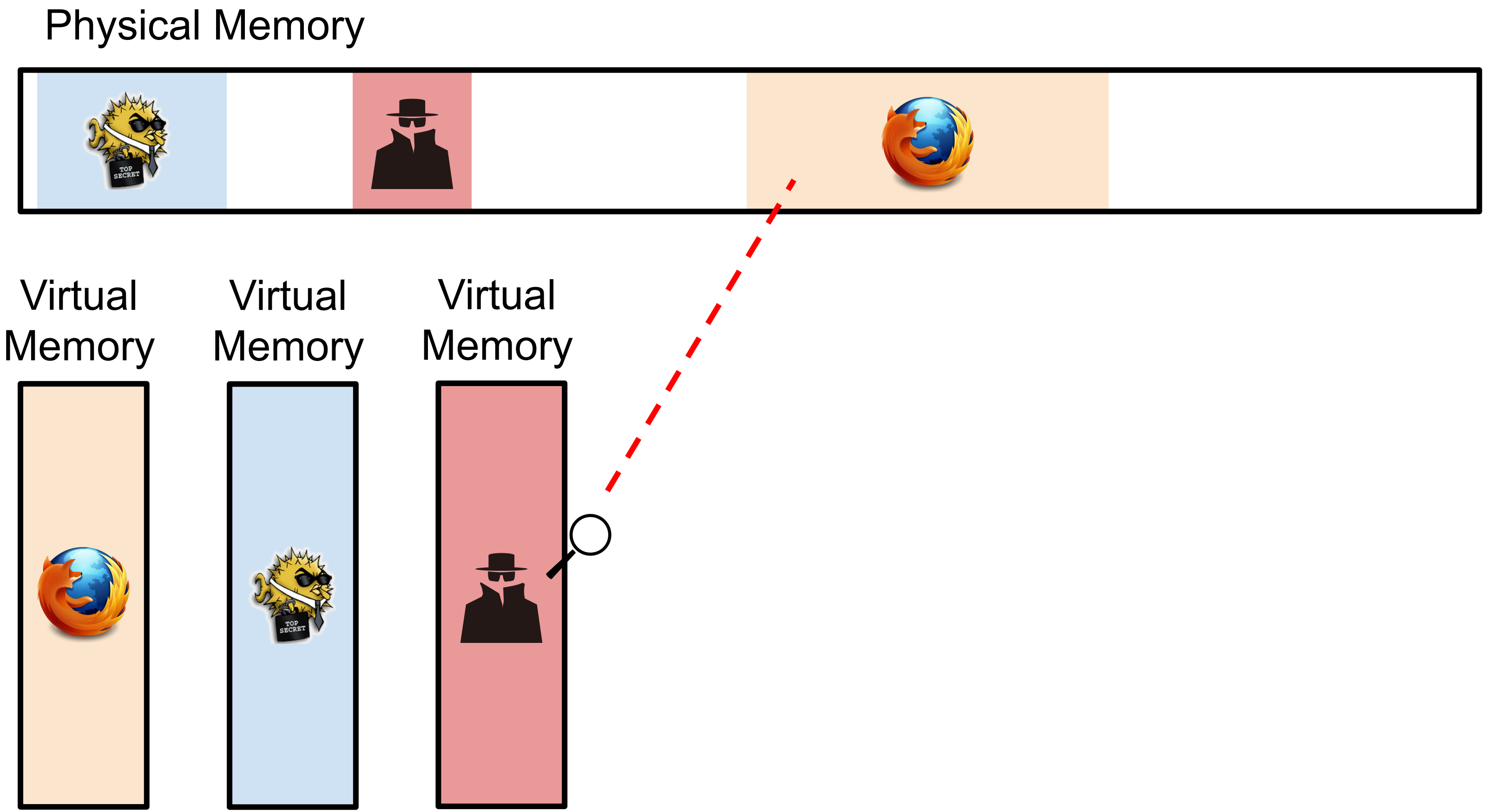


Virtual Memory

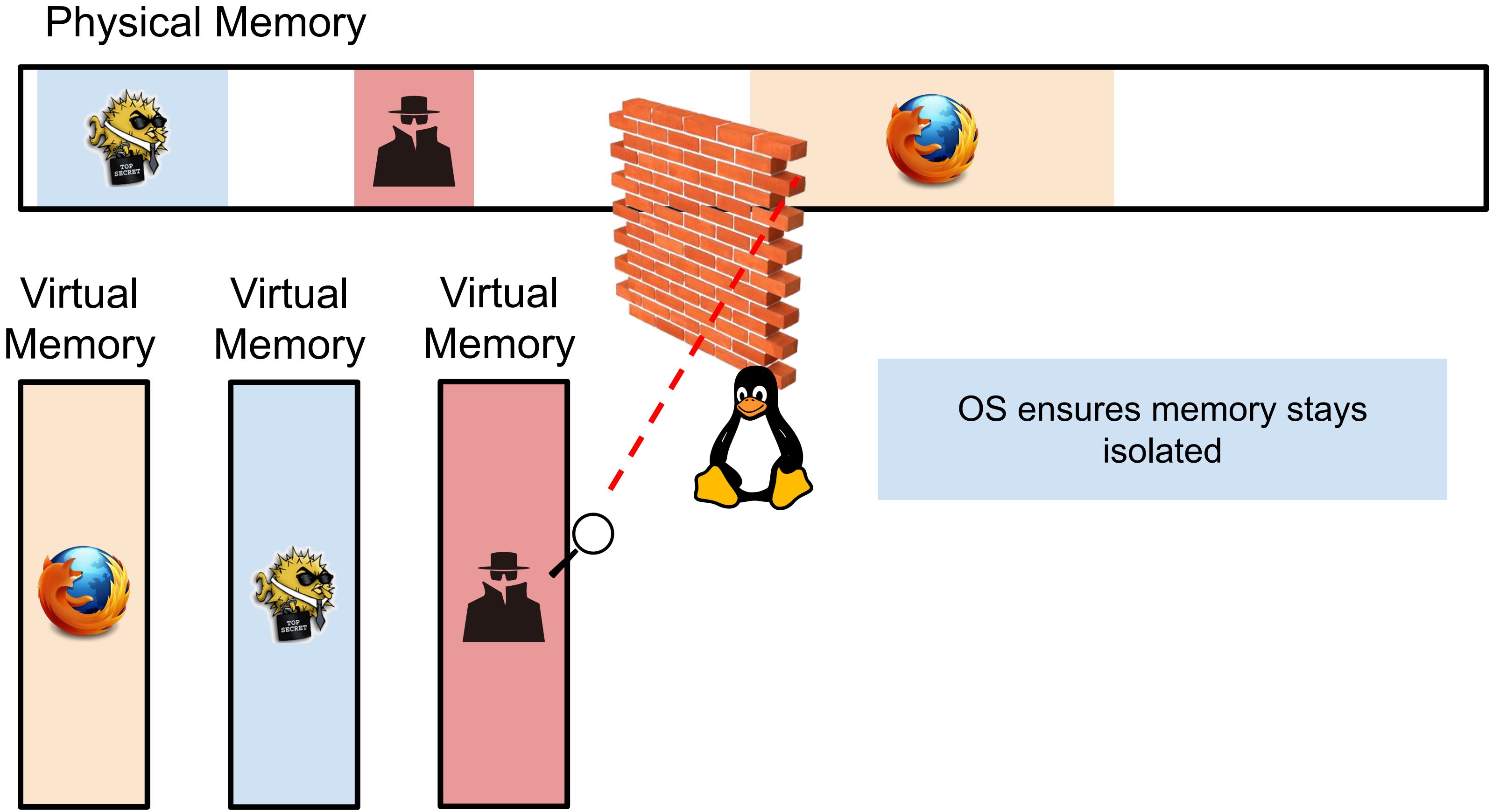
Virtual Memory

Virtual Memory





Physical Memory
TOP SECRET
Virtual Memory
Virtual Memory
Virtual Memory
TOP SECRET



OS ensures memory stays isolated

Memory Protection

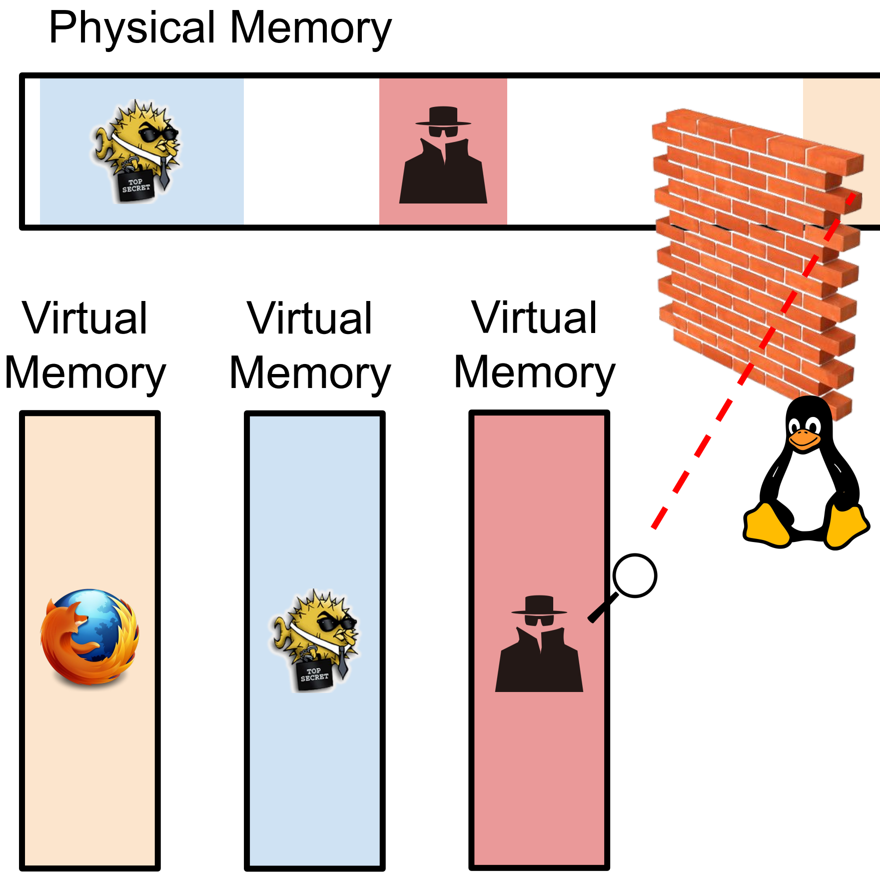
- Recall that a page table entry has:
    - Valid bit - whether page is in memory
    - Physical page number - where page is in memory
    - A bunch of other “flags”

Memory Protection

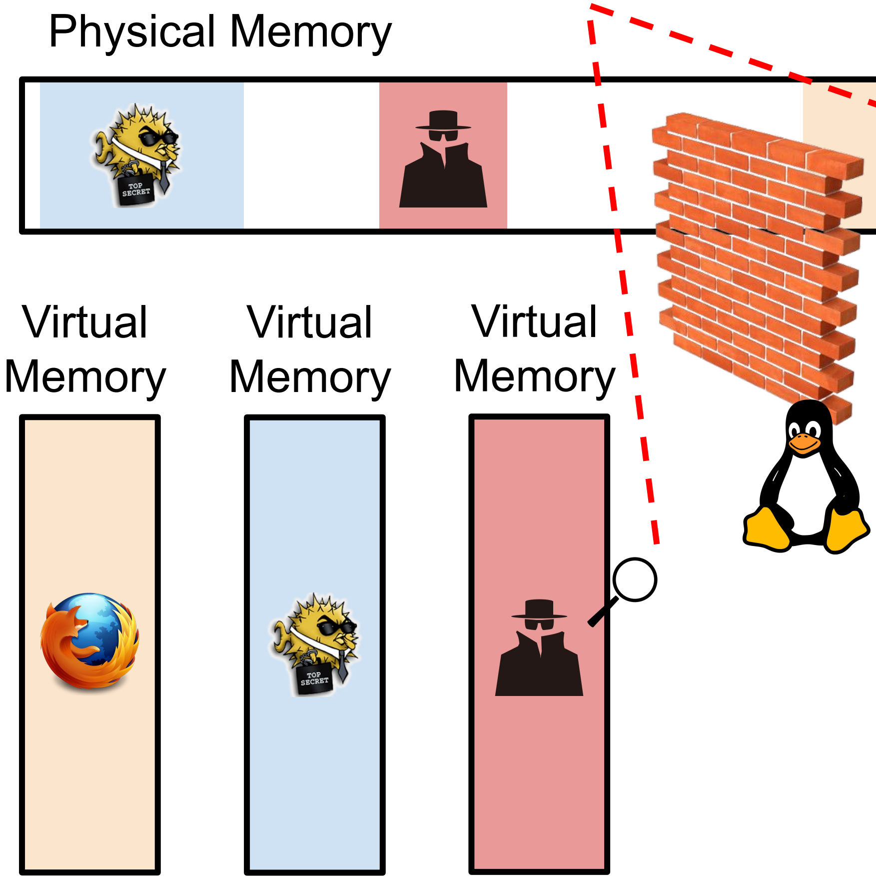
- Recall that a page table entry has:
  - Valid bit - whether page is in memory
  - Physical page number - where page is in memory
  - **A bunch of other "flags"**

Memory Protection

- These flags can include:
  - Whether the page is readable/writable
  - Whether a user can access the page (or only kernel)
- By enforcing these flags (bits), the OS can prevent processes from tampering with other processes’ data
- …but what happens when security fails?



OS ensures memory stays isolated



OS ensures memory stays isolated

What if an attacker can break this isolation?

Can Security Fail?

- What if your hardware is unreliable?
- An attacker can induce bit errors in many commodity DRAM memory chips
- Many different memory vulnerabilities; will look at some in coming weeks

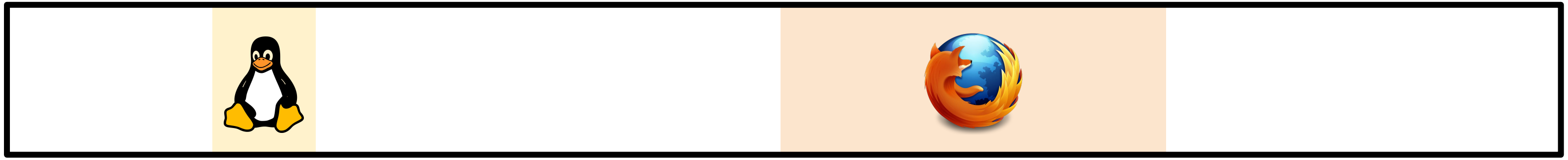
Can Security Fail?

- For now, we just assume that memory isn't 100% reliable
- In other words, assume we can cause a small fraction of bits in physical memory to flip
- What's the damage?

Can Security Fail?

- Recall that a page table entry has:
  - Valid bit - can validate an old mapping/invalidate a current one
  - Read/write bit - can make a page writable that wasn't before
  - Physical page number - change where a VA maps to

Physical Memory



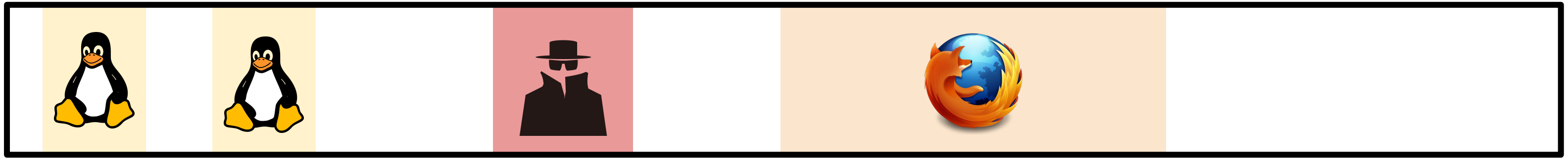
Some physical memory is used to store page tables

Physical Memory

VA	PFN	R/W
0x10	0x9	W
0x20	0xA	W
0x30	0xB	R
0x40	0xC	W
0x50	0xD	R
…	…	…
0xB0		
0xC0		
0xD0		
0xE0		
0xF0		

Some physical memory is used to store page tables

Physical Memory

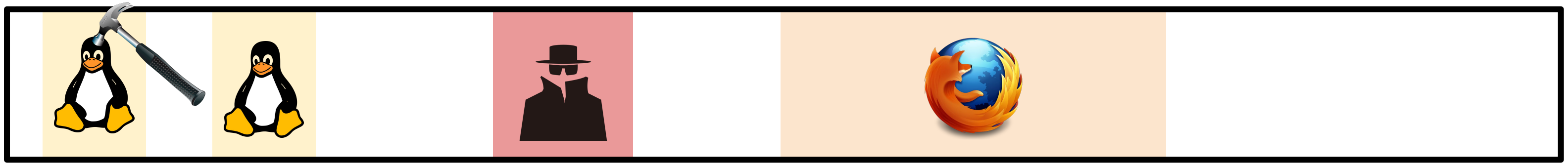


VA	PFN	R/W
0x10	0x9	W
0x20	0xA	W
0x30	0xB	R
0x40	0xC	W
0x50	0xD	R
…	…	…
0xB0		
0xC0		
0xD0		
0xE0		
0xF0		

VA	PFN	R/W
0x10	0x5	R
0x20	0x6	W
0x30	0x7	W
0x40		
0x50		
…	…	…
0xB0		
0xC0		
0xD0		
0xE0		
0xF0		

Some physical memory is used to store page tables

Physical Memory



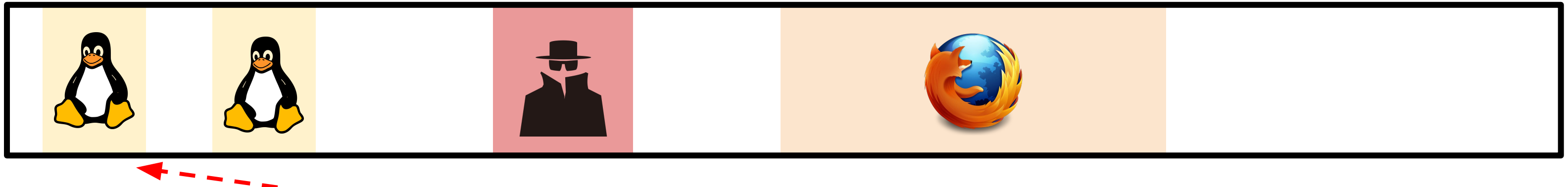
VA	PFN	R/W
0x10	0x9	W
0x20	0xA	W
0x30	0xB	R
0x40	0xC	W
0x50	0xD	R
…	…	…
0xB0		
0xC0		
0xD0		
0xE0		
0xF0		

VA	PFN	R/W
0x10	0x5	R
0x20	0x**2**	W
0x30	0x7	W
0x40		
0x50		
…	…	…
0xB0		
0xC0		
0xD0		
0xE0		
0xF0		

Some physical memory is used to store page tables

A memory attack can cause a corruption in a page table…

Physical Memory



VA	PFN	R/W
0x10	0x9	W
0x20	0xA	W
0x30	0xB	R
0x40	0xC	W
0x50	0xD	R
…	…	…
0xB0		
0xC0		
0xD0		
0xE0		
0xF0		

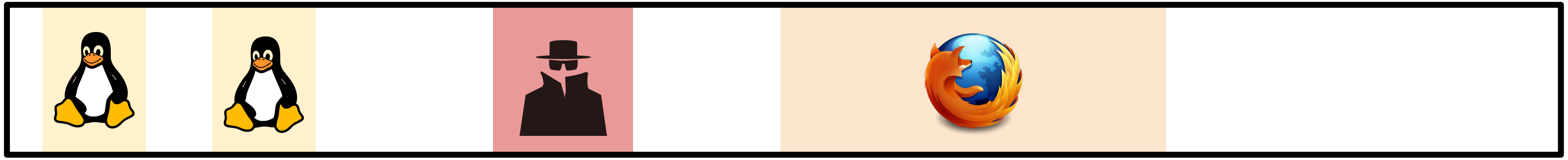
VA	PFN	R/W
0x10	0x5	R
0x20	**0x2**	**W**
0x30	0x7	W
0x40		
0x50		
…	…	…
0xB0		
0xC0		
0xD0		
0xE0		
0xF0		

Some physical memory is used to store page tables

A memory attack can cause a corruption in a page table…

…which can give a process write access to its **own** page table!

Physical Memory



VA	PFN	R/W
0x10	0x9	W
0x20	0xA	W
0x30	0xB	R
0x40	0xC	W
0x50	0xD	R
…	…	…
0xB0		
0xC0		
0xD0		
0xE0		
0xF0		

VA	PFN	R/W
0x10	0x5	R
0x20	**0x2**	**W**
0x30	0x7	W
0x40	**0xA**	**W**
0x50		
…	…	…
0xB0		
0xC0		
0xD0		
0xE0		
0xF0		

Some physical memory is used to store page tables

A memory attack can cause a corruption in a page table…

…which can give a process write access to its **own** page table!

The Bottom Line

- There's an inherent *contract* between virtual and physical memory
- Hardware faults violate this contract
- Further work on *defending* physical memory is crucial