Quincy Flint

Virtual Memory

EEL 3713C: Digital Computer Architecture

Quincy Flint

[Ionospheric Radio Lab in NEB]

Outline

1. Memory Problems
   - Not enough memory
   - Holes in address space
   - Programs overwriting
2. What is Virtual Memory?
   - Layer of indirection
   - How does indirection solve above
   - Page tables and translation
3. How do we implement VM?
   - Create and store page tables
   - Fast address translation
4. Virtual Memory and Caches
   - Prevent cache performance degradation when using VM

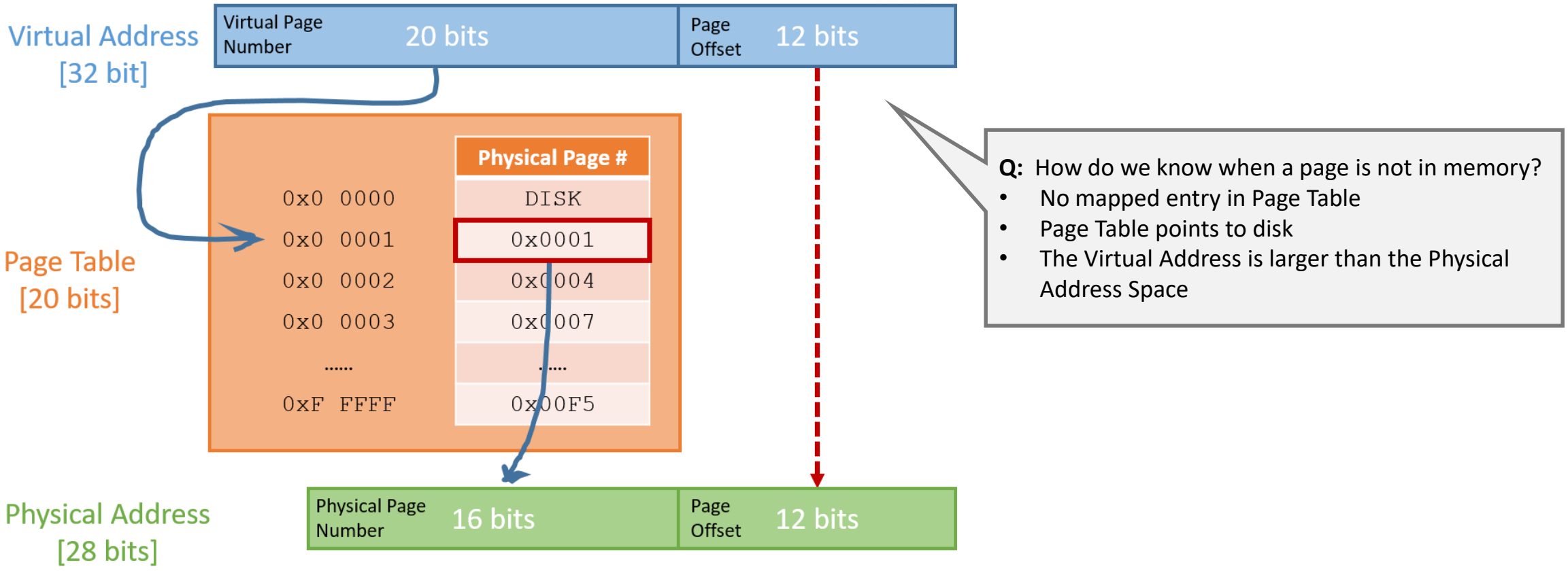
Quincy Flint

Page Faults

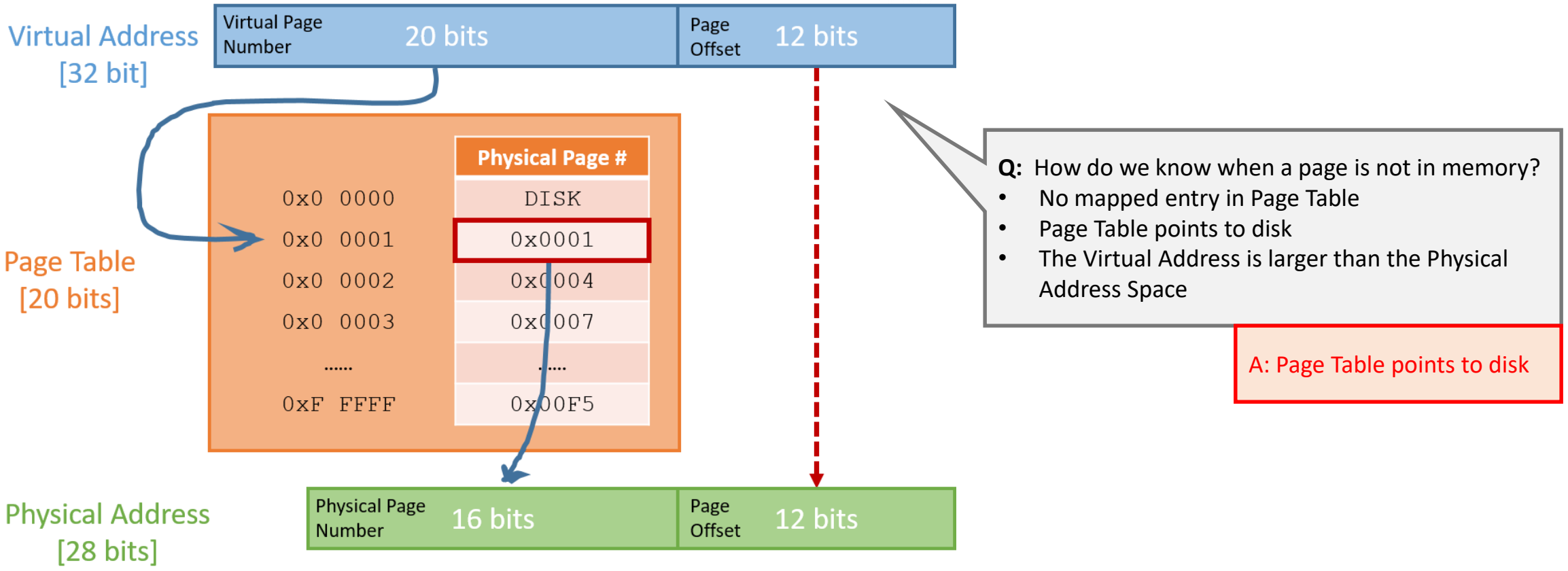
Page Faults

- A Page Fault occurs when we must access the disk to fetch data because it is not stored in memory.

What happens if a page is not in memory?



What happens if a page is not in memory?



What happens if a page is not in memory?

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults

Dirty Page: a page is "dirty" if it has been written to since it has been in the cache.

Dirty pages must be written back to disk before being evicted from memory.

A bit is set in the PT to flag dirty pages.

What happens if a page is not in memory?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults
- This takes a long time!

Dirty Page: a page is "dirty" if it has been written to since it has been in the cache.

Dirty pages must be written back to disk before being evicted from memory.

A bit is set in the PT to flag dirty pages.

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk.
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults
- This takes a long time!

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults
- This takes a long time!

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception ~100 cycles
  - Hardware passes control to O/S page fault handler
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults
- This takes a long time!

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception ~100 cycles
  - Hardware passes control to O/S page fault handler ~10,000 cycles
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite)
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults
- This takes a long time!

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception ~100 cycles
  - Hardware passes control to O/S page fault handler ~10,000 cycles
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite) ~40,000,000 cycles
    3. Update Page Table Entry for *old* page to reference disk
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults
- This takes a long time!

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception ~100 cycles
  - Hardware passes control to O/S page fault handler ~10,000 cycles
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite) ~40,000,000 cycles
    3. Update Page Table Entry for *old* page to reference disk ~1,000 cycles
    4. Bring in new page from disk to memory
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults
- This takes a long time!

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception ~100 cycles
  - Hardware passes control to O/S page fault handler ~10,000 cycles
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite) ~40,000,000 cycles
    3. Update Page Table Entry for *old* page to reference disk ~1,000 cycles
    4. Bring in new page from disk to memory ~40,000,000 cycles
    5. Update Page Table Entry for *new* page
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults
- This takes a long time!

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception ~100 cycles
  - Hardware passes control to O/S page fault handler ~10,000 cycles
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite) ~40,000,000 cycles
    3. Update Page Table Entry for *old* page to reference disk ~1,000 cycles
    4. Bring in new page from disk to memory ~40,000,000 cycles
    5. Update Page Table Entry for *new* page ~1,000 cycles
    6. Return control to faulting instruction
  - Architected hardware can also handle page faults
- This takes a long time!

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception ~100 cycles
  - Hardware passes control to O/S page fault handler ~10,000 cycles
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite) ~40,000,000 cycles
    3. Update Page Table Entry for *old* page to reference disk ~1,000 cycles
    4. Bring in new page from disk to memory ~40,000,000 cycles
    5. Update Page Table Entry for *new* page ~1,000 cycles
    6. Return control to faulting instruction ~10,000 cycles
  - Architected hardware can also handle page faults
- This takes a long time!

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception ~100 cycles
  - Hardware passes control to O/S page fault handler ~10,000 cycles
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite) ~40,000,000 cycles
    3. Update Page Table Entry for *old* page to reference disk ~1,000 cycles
    4. Bring in new page from disk to memory ~40,000,000 cycles
    5. Update Page Table Entry for *new* page ~1,000 cycles
    6. Return control to faulting instruction ~10,000 cycles
  - Architected hardware can also handle page faults
- This takes a long time! ~80,000,000 cycles

How long does a page fault take?

- If a page is not in memory, Page Table Entry says it is on disk. ~1 cycle
- Hardware generates a Page Fault Exception ~100 cycles
  - Hardware passes control to O/S page fault handler ~10,000 cycles
    1. The O/S chooses a page to replace in memory
    2. If the *old* page is **"dirty"**, write it to disk (if clean, we can overwrite) ~40,000,000 cycles
    3. Update Page Table Entry for *old* page to reference disk ~1,000 cycles
    4. Bring in new page from disk to memory ~40,000,000 cycles
    5. Update Page Table Entry for *new* page ~1,000 cycles
    6. Return control to faulting instruction ~10,000 cycles
  - Architected hardware can also handle page faults
- This takes a long time! ~80,000,000 cycles ~20 ms on a 4 GHz processor

Illustration from the textbook

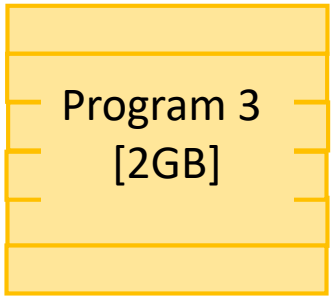
Memory technology	Typical access time	$ per GIB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

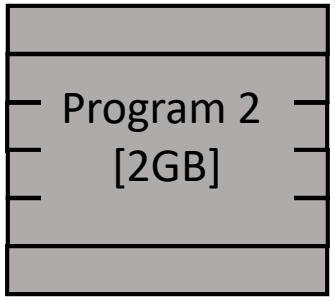
Quincy Flint

Memory Protection

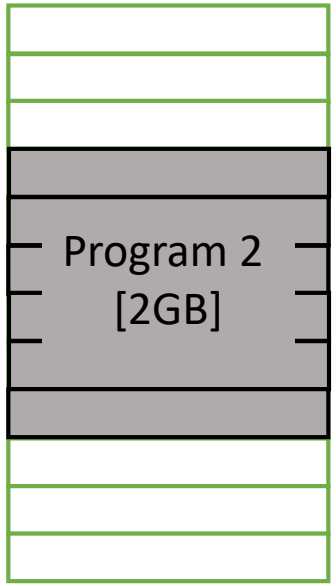
Virtual Memory Protects Applications *(Review)*

- Each program has its own Page Table. A program's Virtual Address is mapped to a unique Physical Address in memory.





4 GB [32-bit] RAM
Physical Address Space

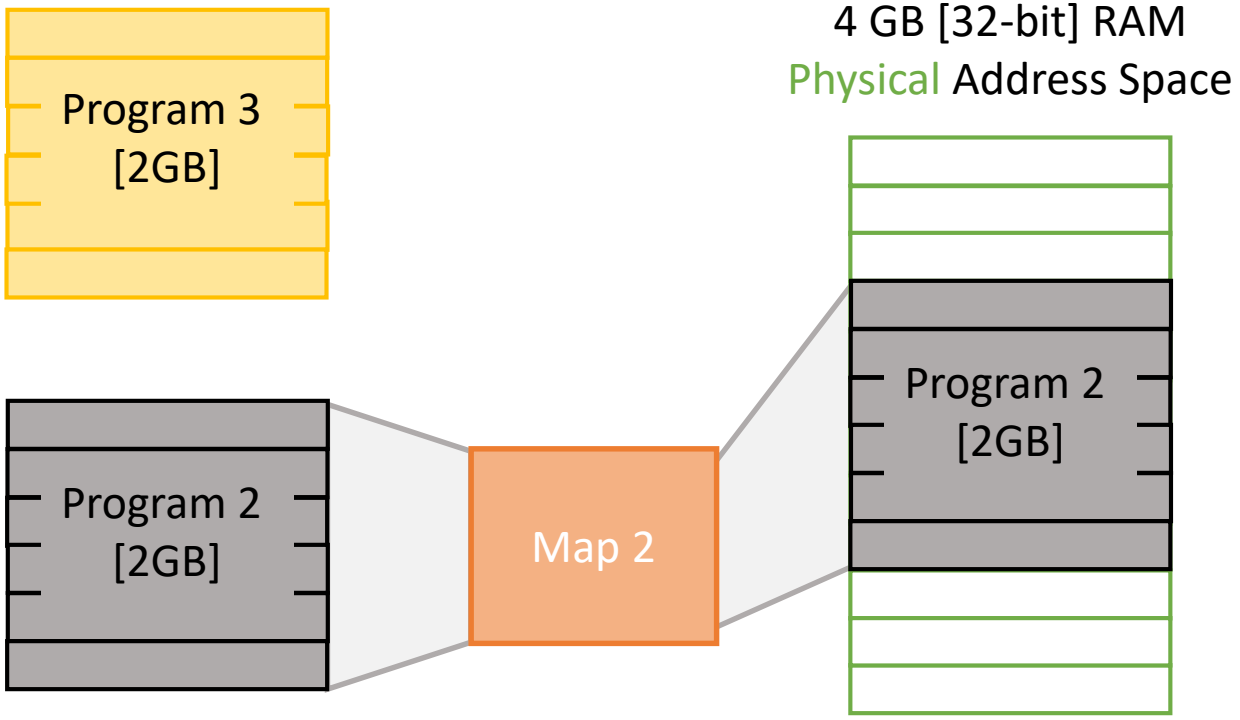


Program Sequence:

1. Run programs 1 and 2 [1 GB free]
2. Close program 1 [2 GB free]
3.

Virtual Memory Protects Applications *(Review)*

- Each program has its own Page Table. A program's Virtual Address is mapped to a unique Physical Address in memory.

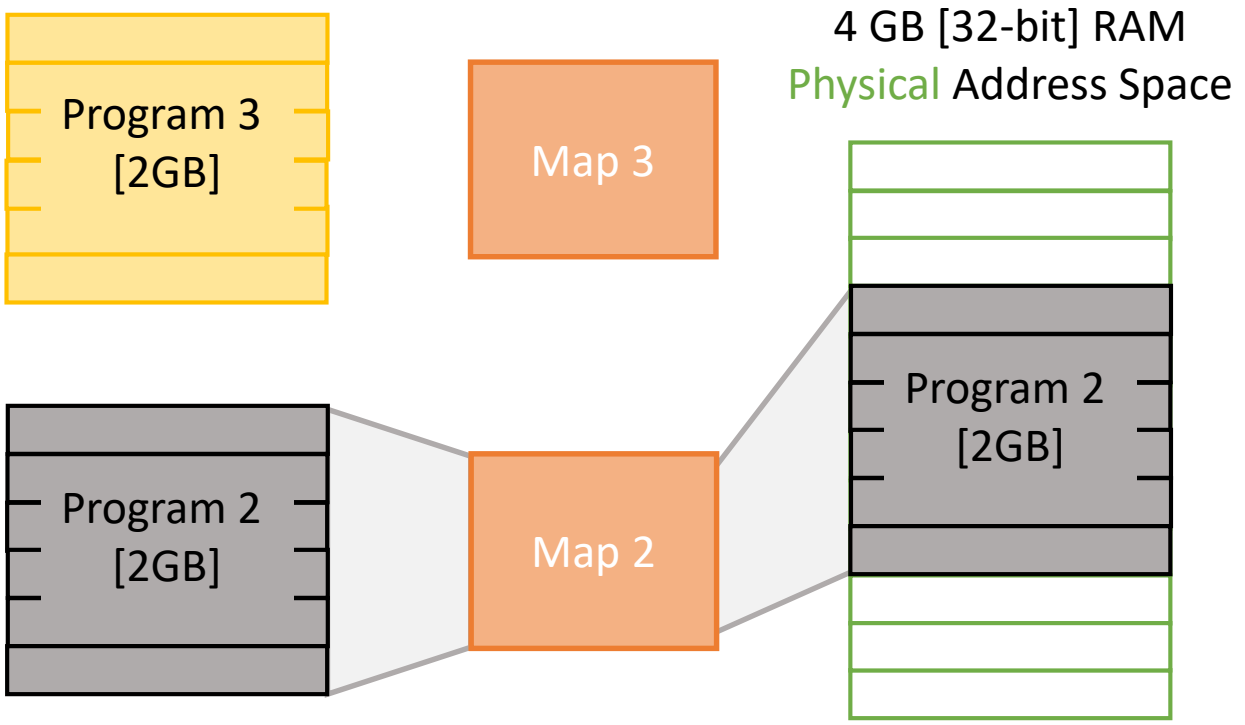


Program Sequence:

1. Run programs 1 and 2 [1 GB free]
2. Close program 1 [2 GB free]
3.

Virtual Memory Protects Applications *(Review)*

- Each program has its own Page Table. A program's Virtual Address is mapped to a unique Physical Address in memory.

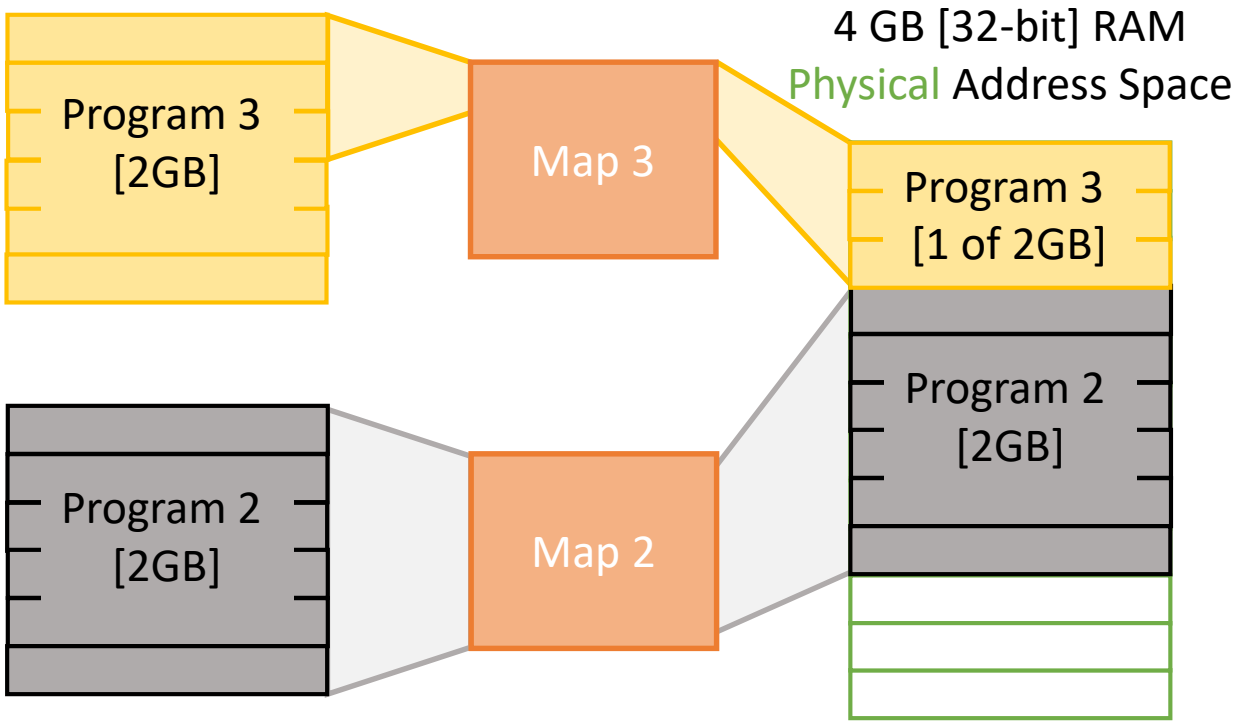


Program Sequence:

1. Run programs 1 and 2 [1 GB free]
2. Close program 1 [2 GB free]
3. Run program 3

Virtual Memory Protects Applications *(Review)*

- Each program has its own Page Table. A program's Virtual Address is mapped to a unique Physical Address in memory.

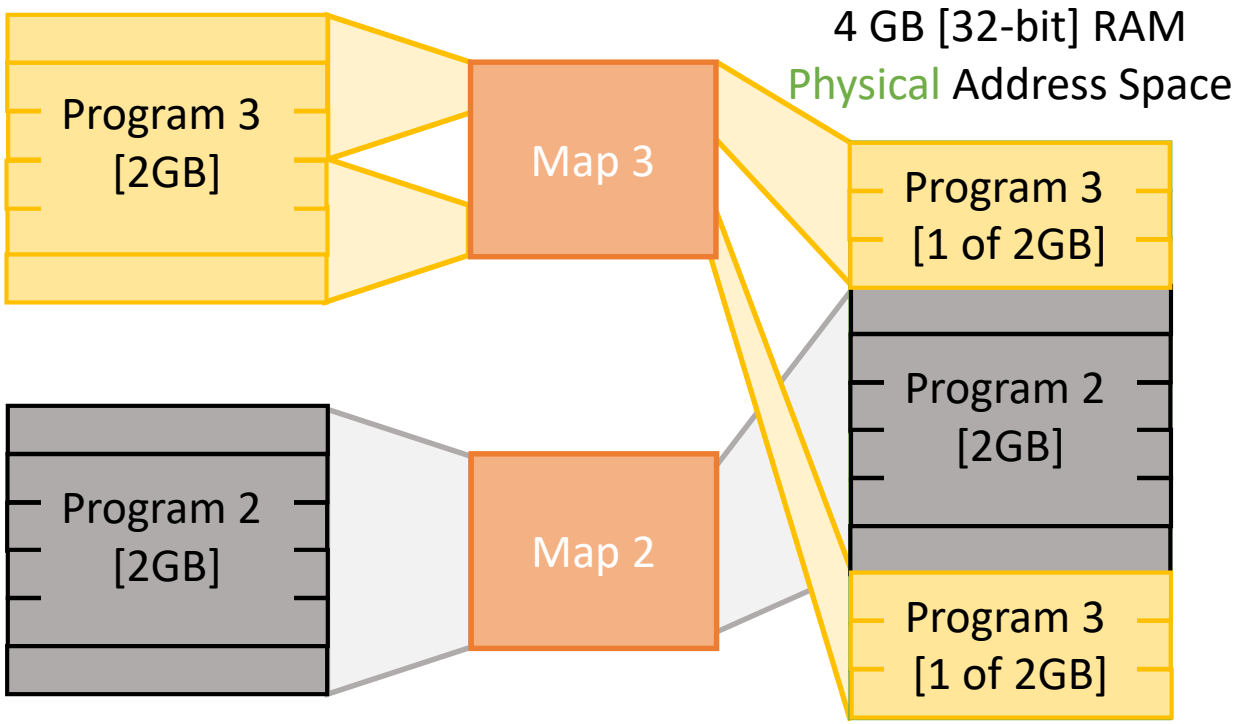


Program Sequence:

1. Run programs 1 and 2 [1 GB free]
2. Close program 1 [2 GB free]
3. Run program 3

Virtual Memory Protects Applications *(Review)*

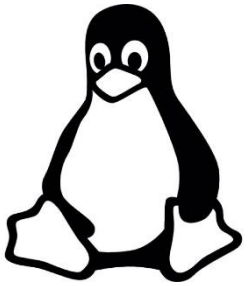
- Each program has its own Page Table. A program's Virtual Address is mapped to a unique Physical Address in memory.



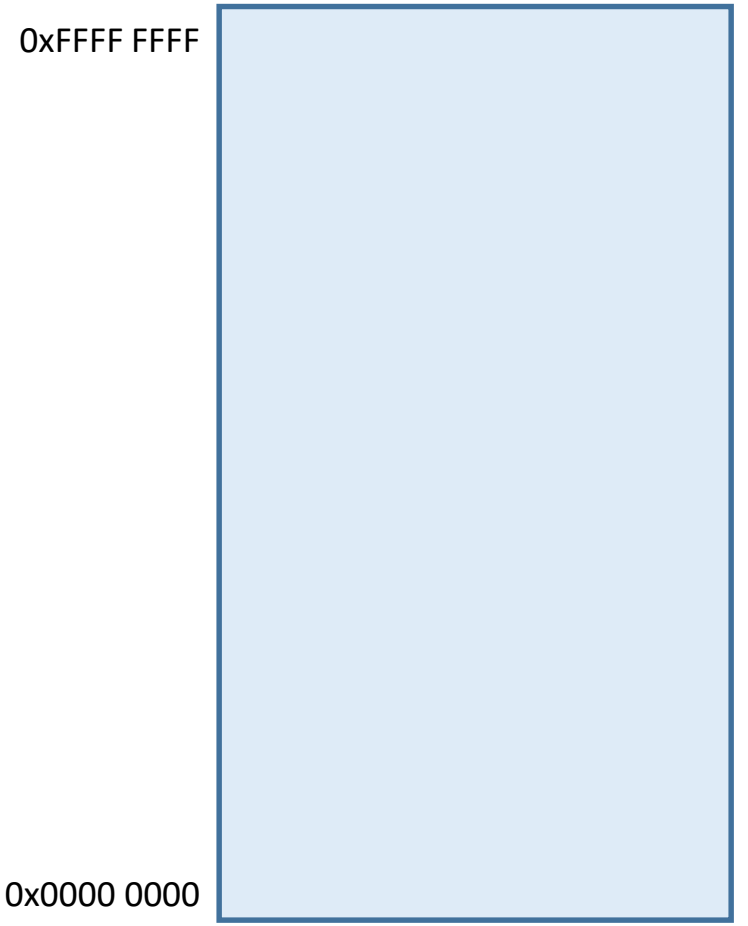
Program Sequence:

1. Run programs 1 and 2 [1 GB free]
2. Close program 1 [2 GB free]
3. Run program 3 [CAN DO!]

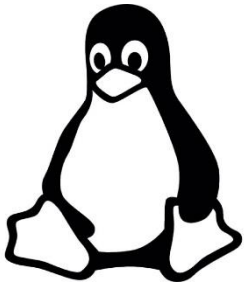
Linux Virtual Address Space



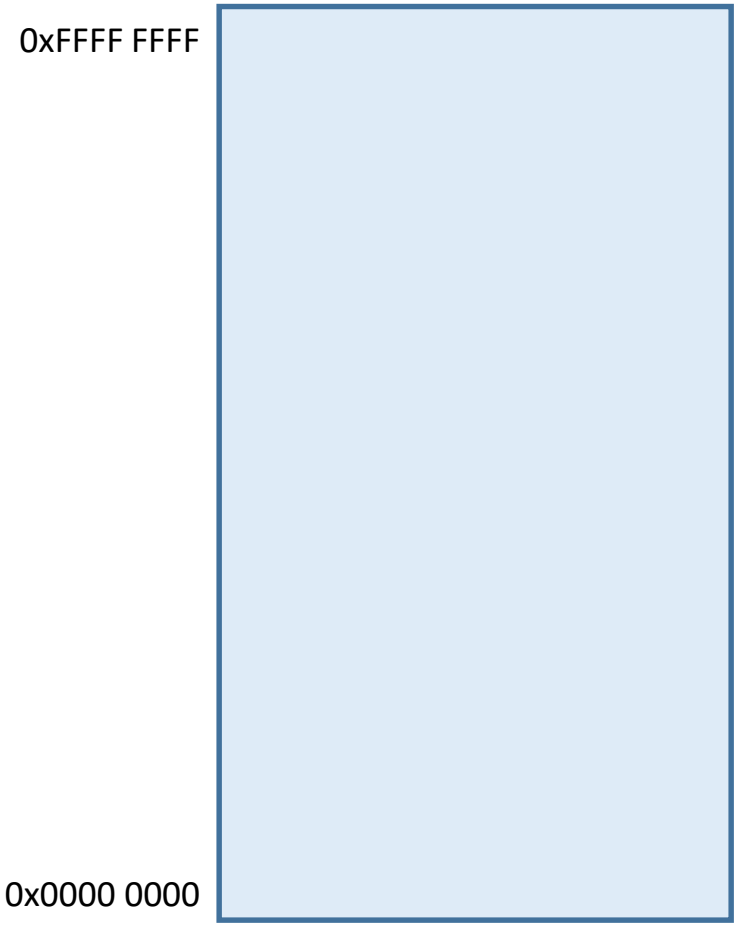
- Consider a 32-bit address space



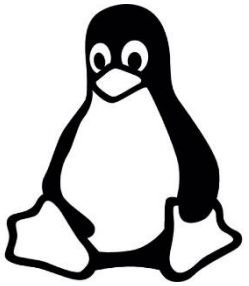
Linux Virtual Address Space



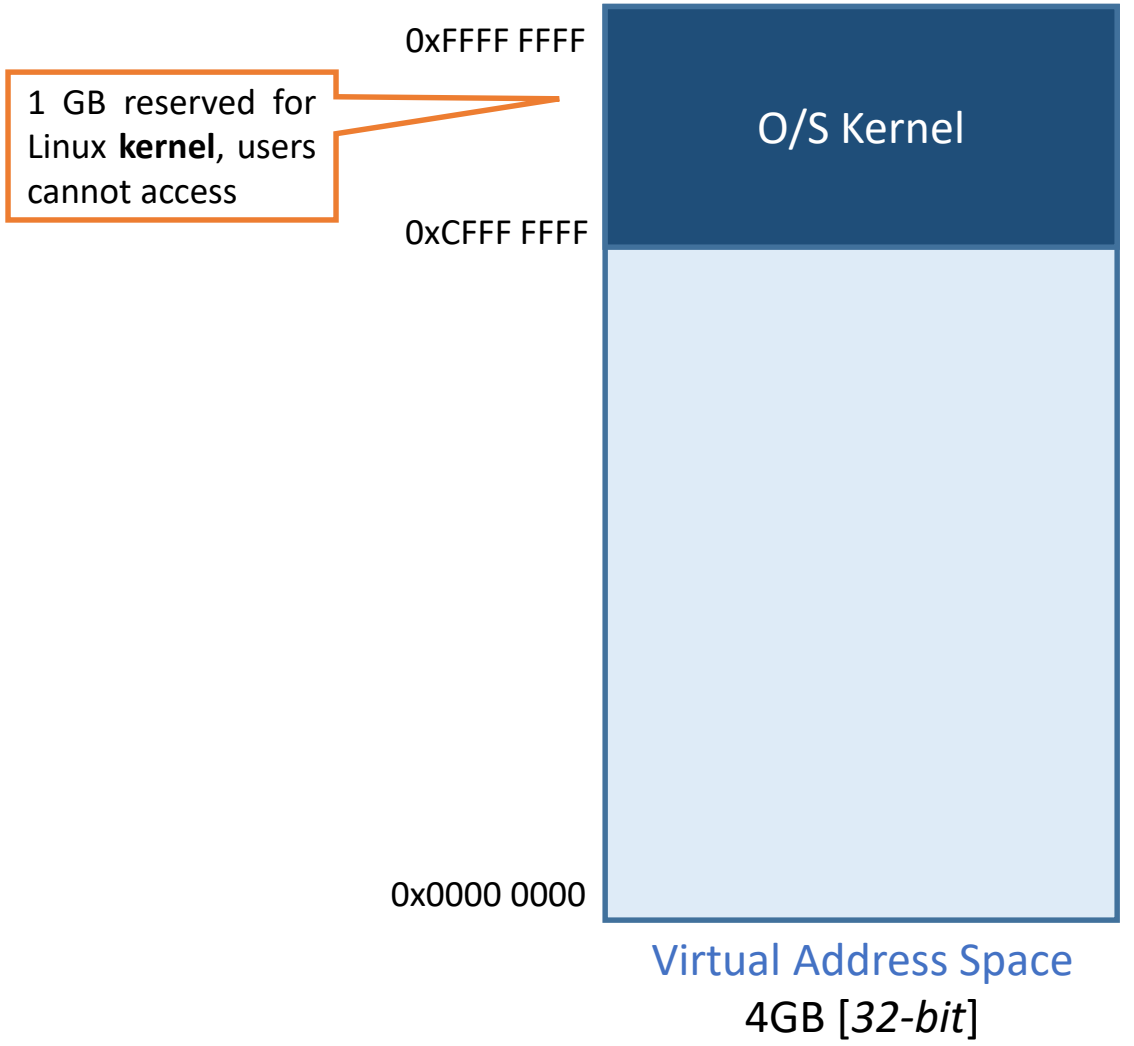
- Consider a 32-bit address space
- The Linux Address Space -->



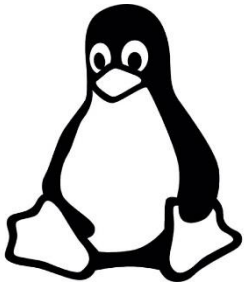
Linux Virtual Address Space



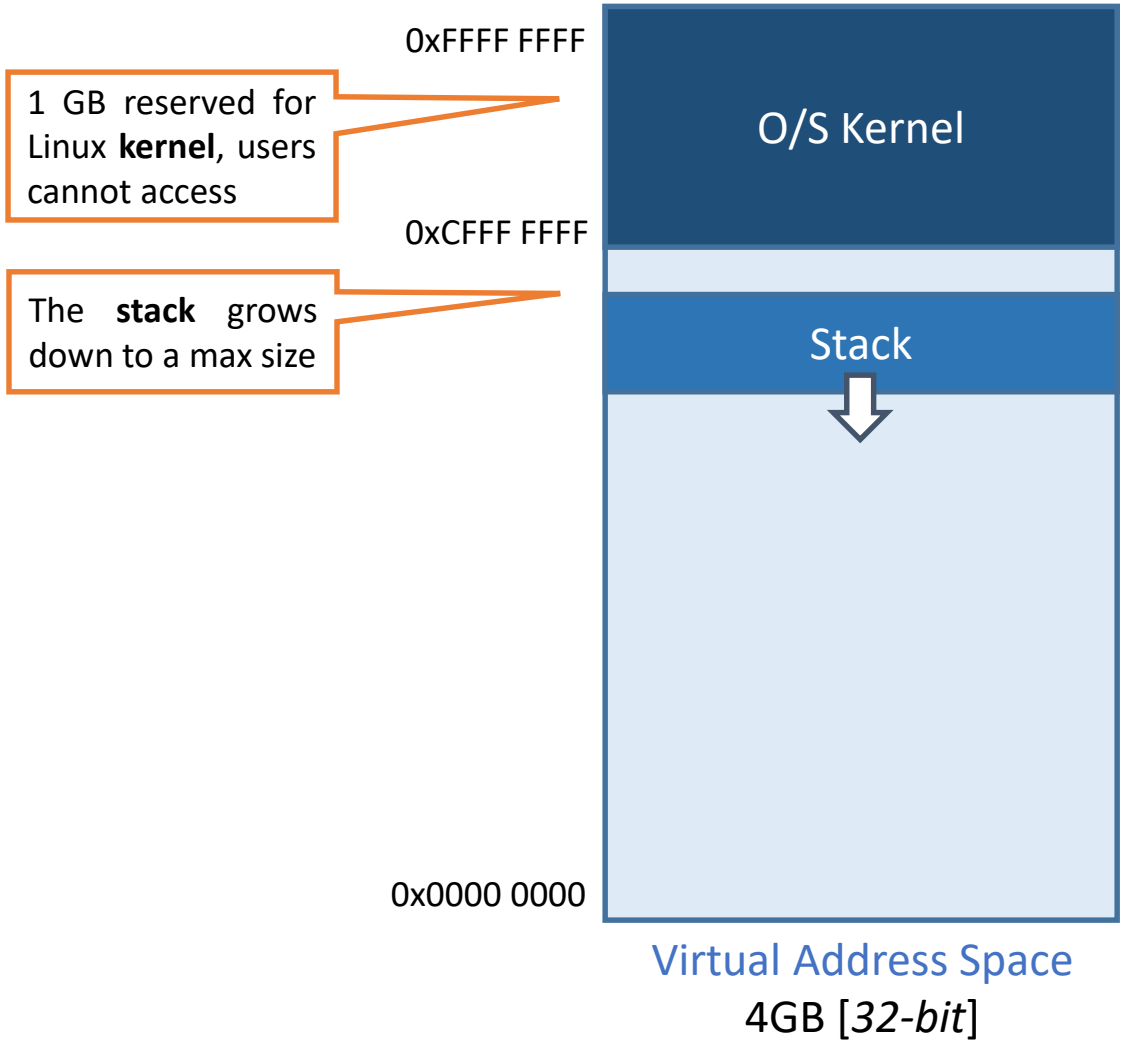
- Consider a 32-bit address space
- The Linux Address Space -->



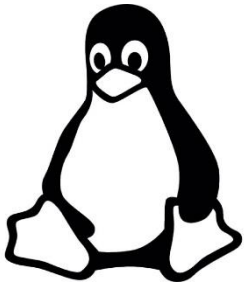
Linux Virtual Address Space



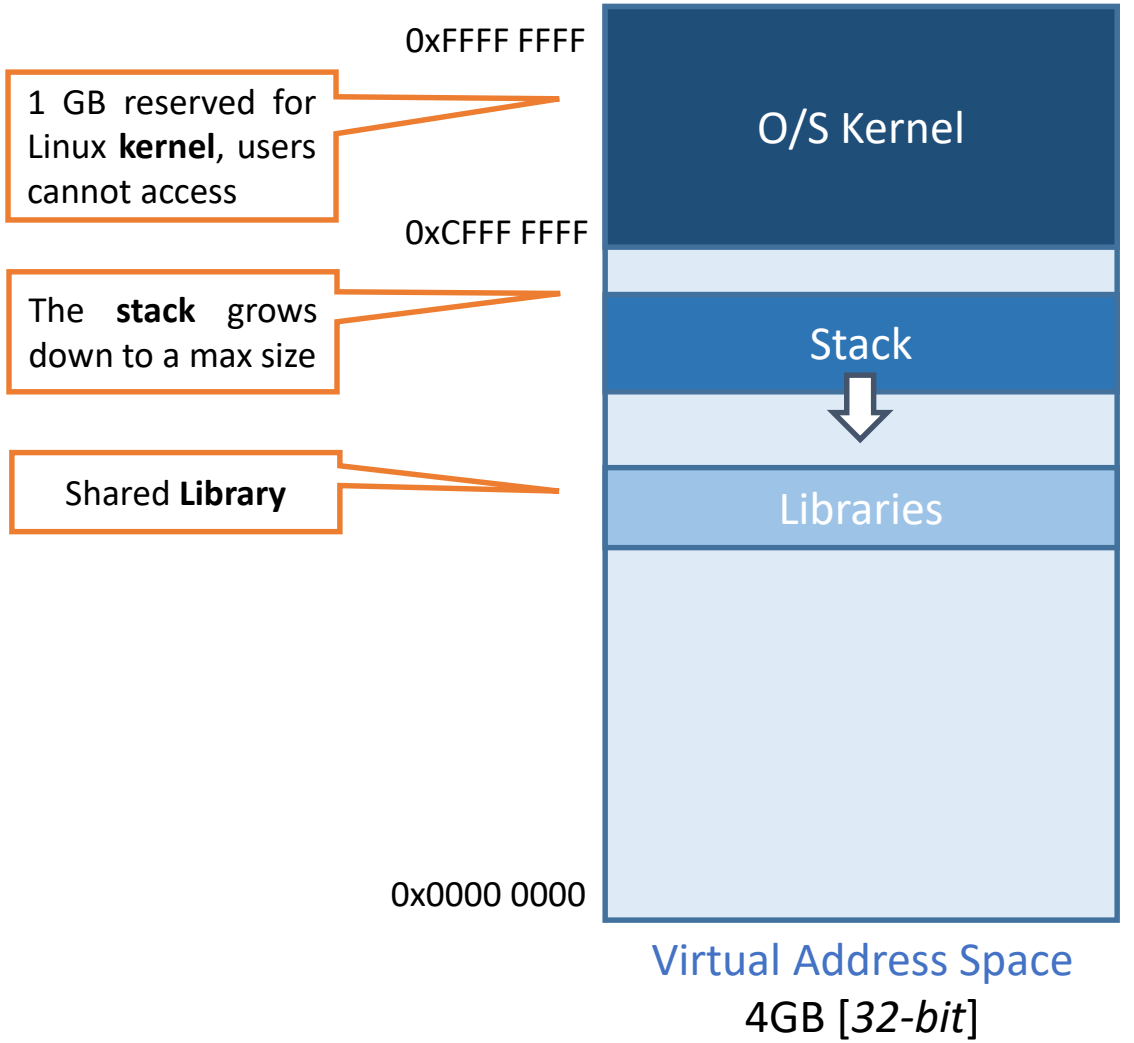
- Consider a 32-bit address space
- The Linux Address Space -->



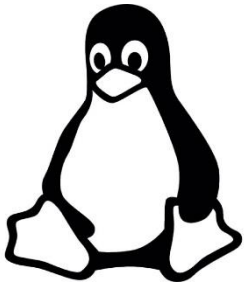
Linux Virtual Address Space



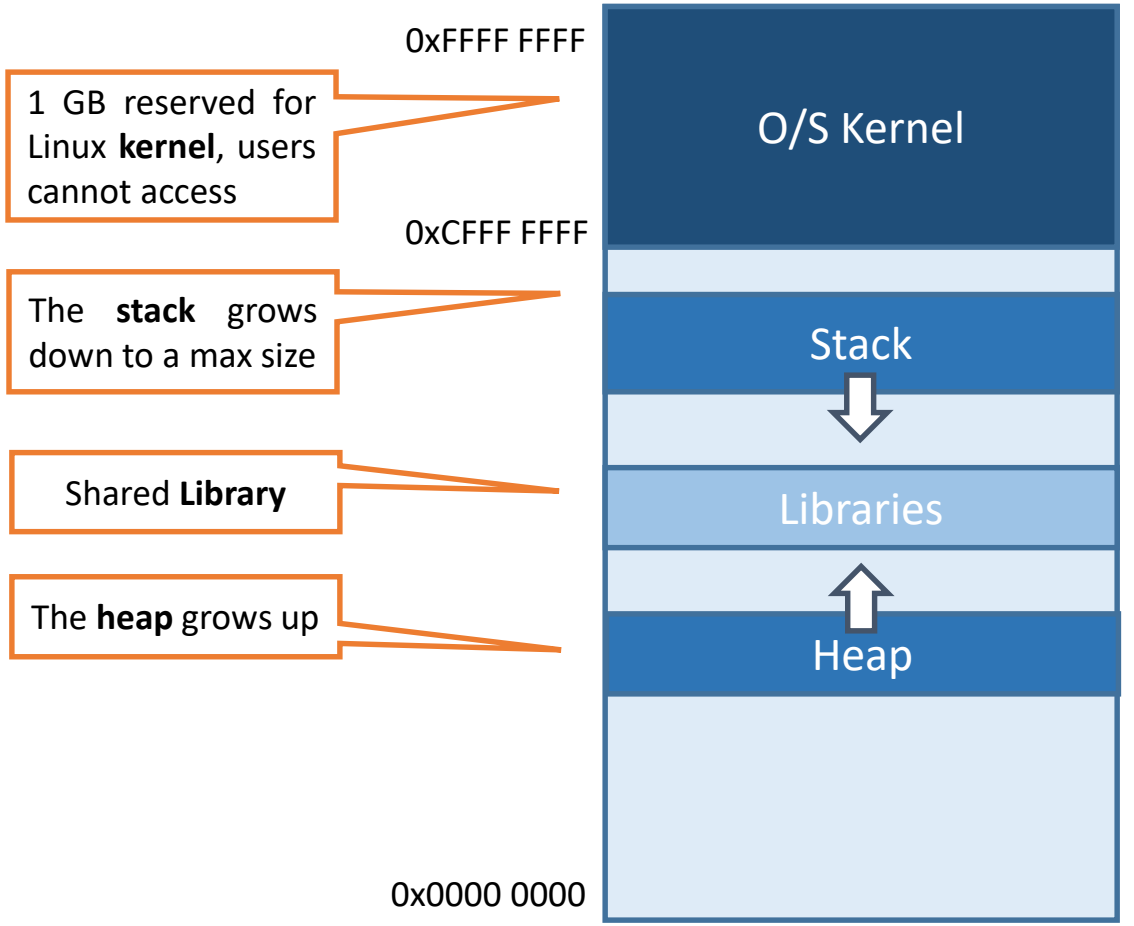
- Consider a 32-bit address space
- The Linux Address Space -->



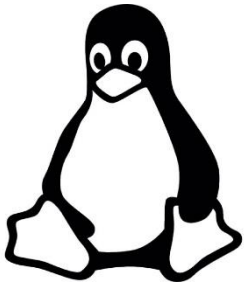
Linux Virtual Address Space



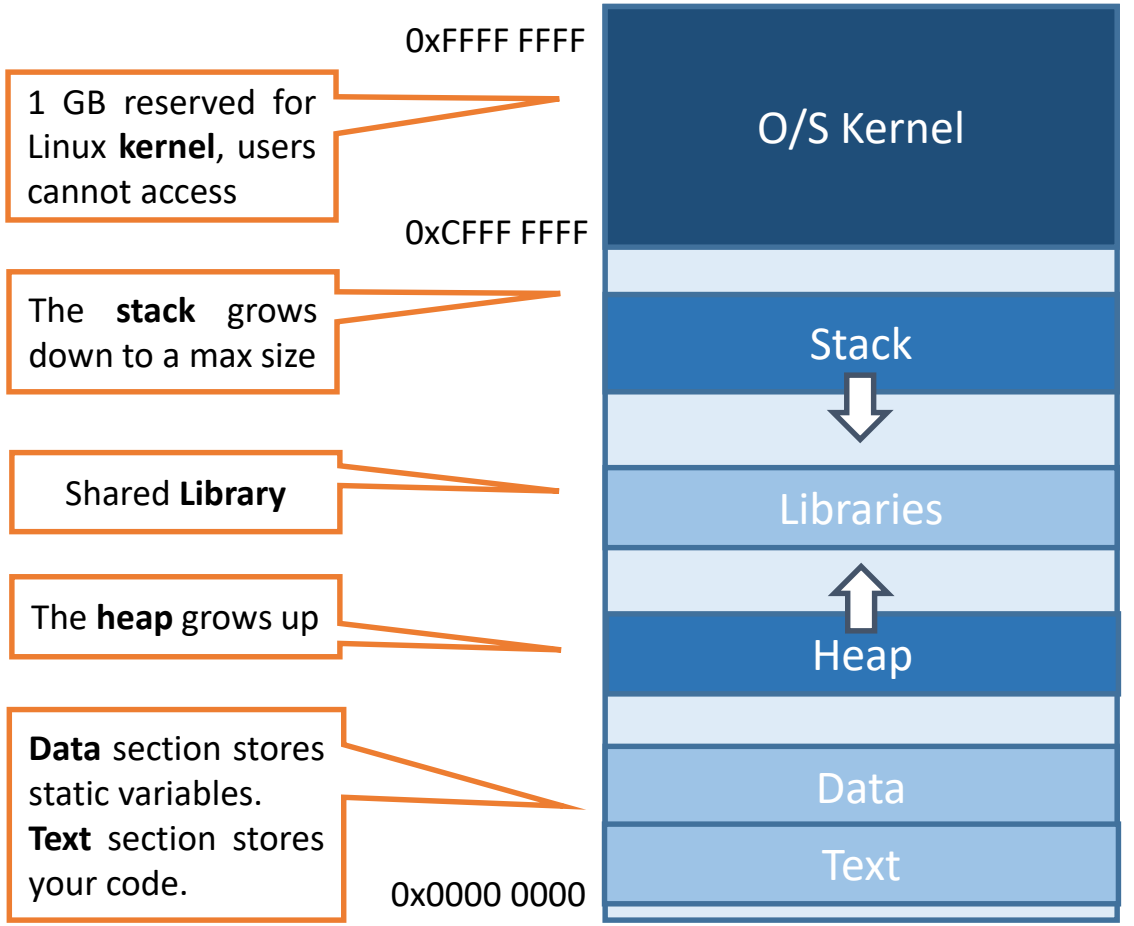
- Consider a 32-bit address space
- The Linux Address Space -->



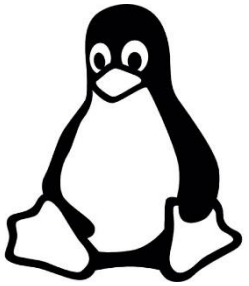
Linux Virtual Address Space



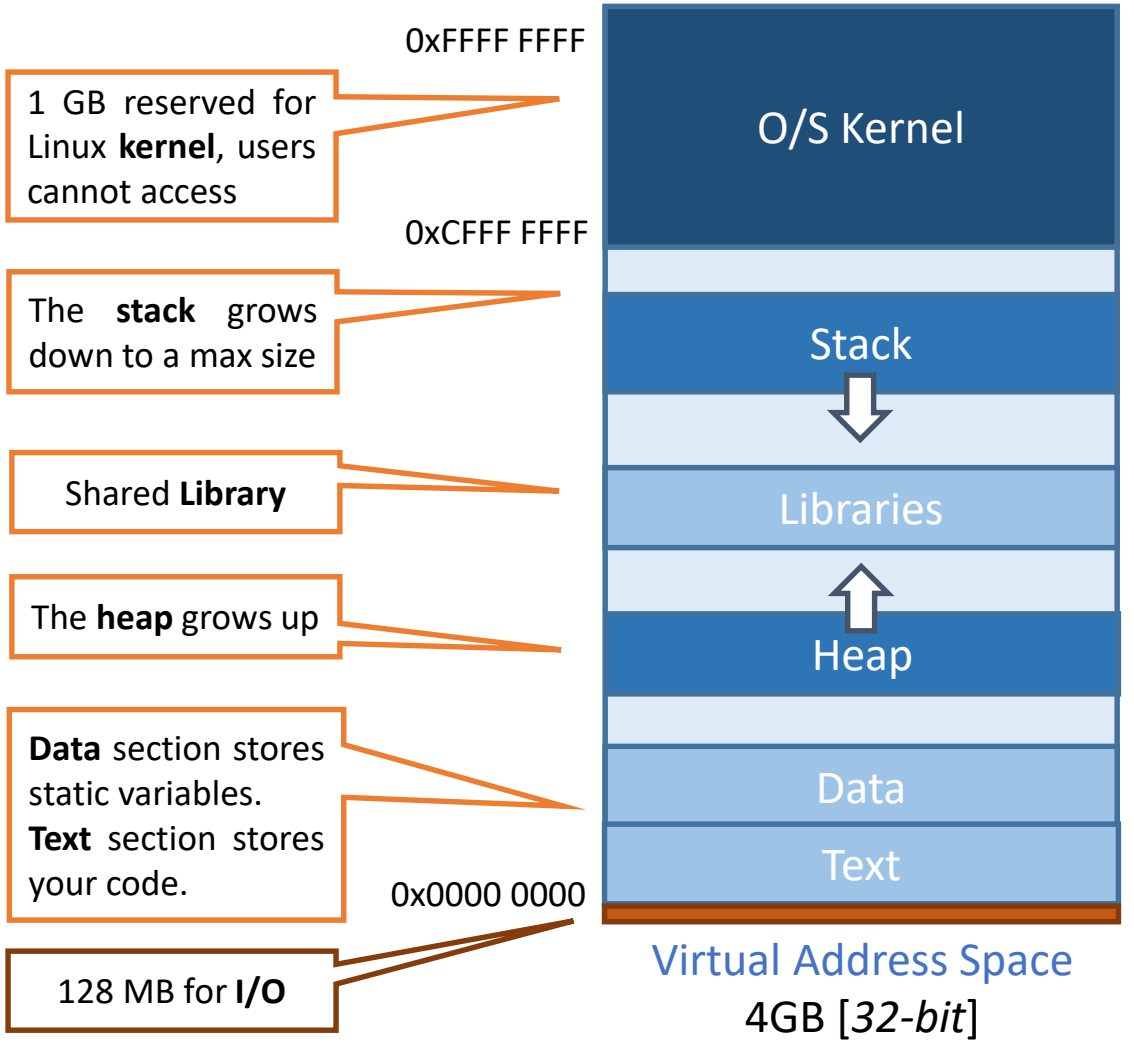
- Consider a 32-bit address space
- The Linux Address Space -->



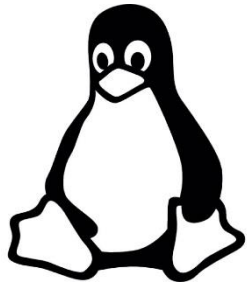
Linux Virtual Address Space



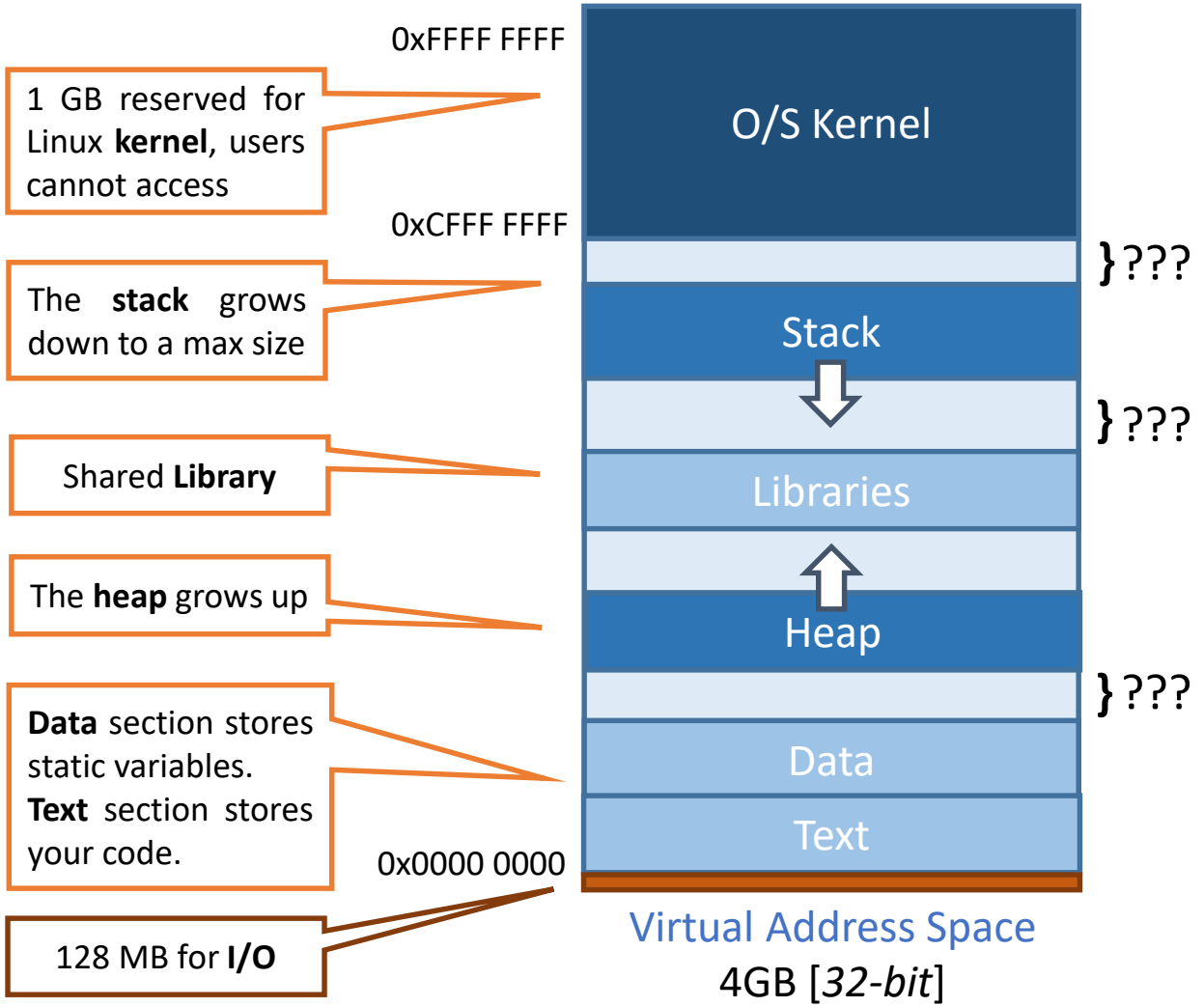
- Consider a 32-bit address space
- The Linux Address Space -->



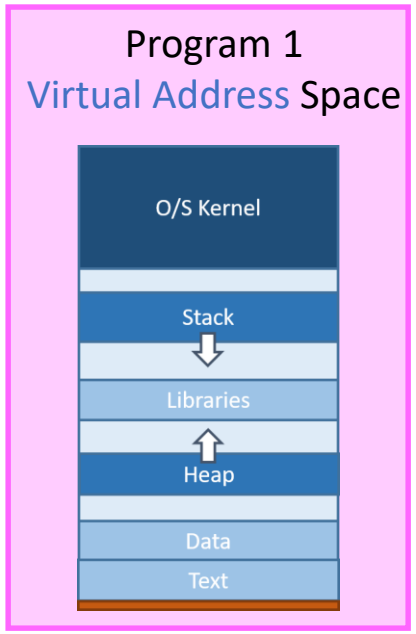
Linux Virtual Address Space

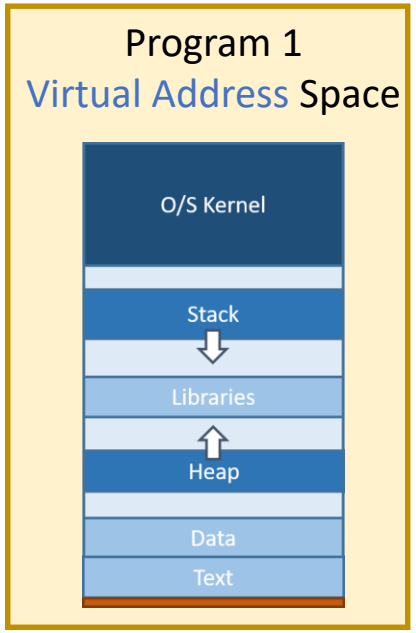


- Consider a 32-bit address space
- The Linux Address Space -->
- Random offsets for security
  - Never know where code is…



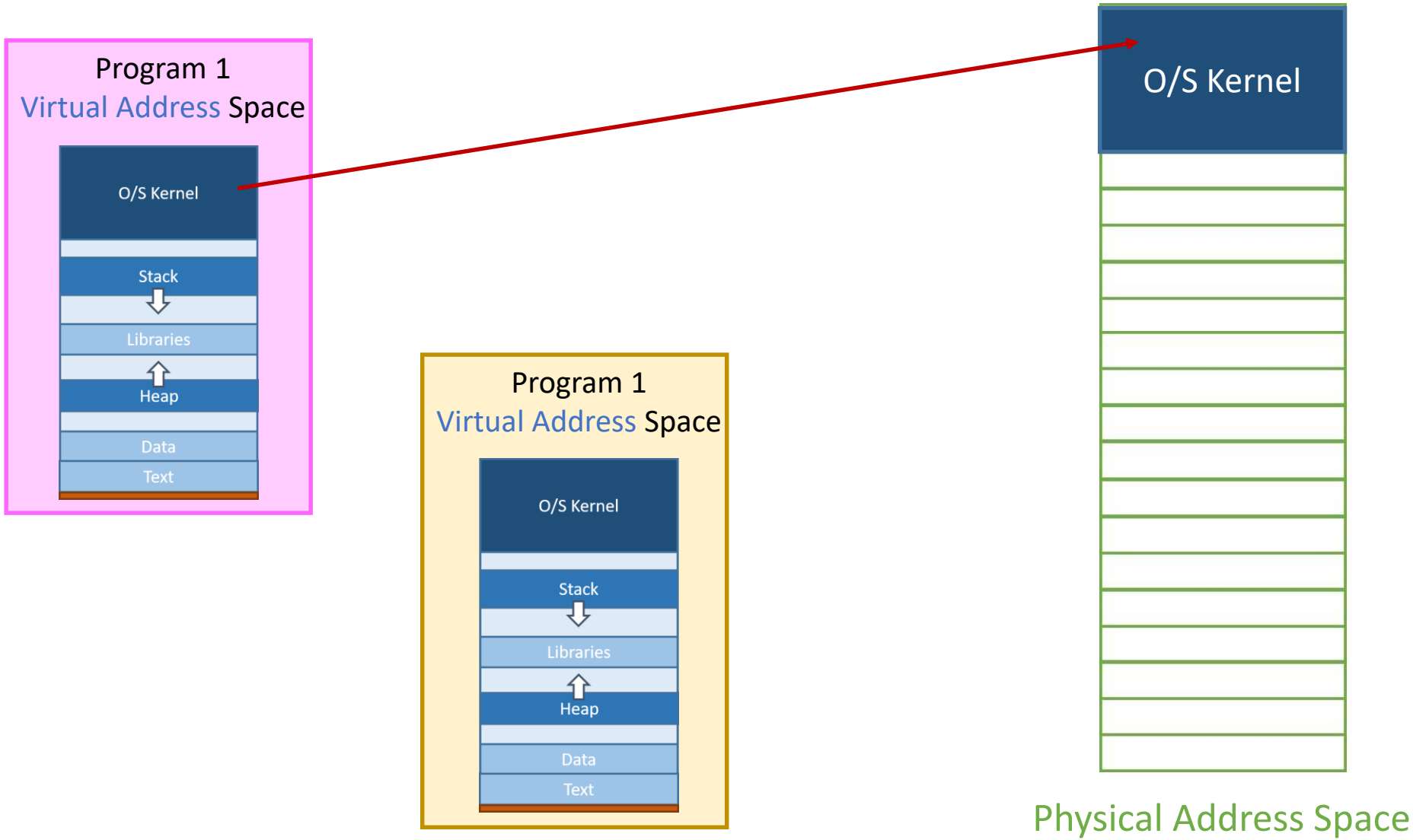
Linux Virtual Address Space



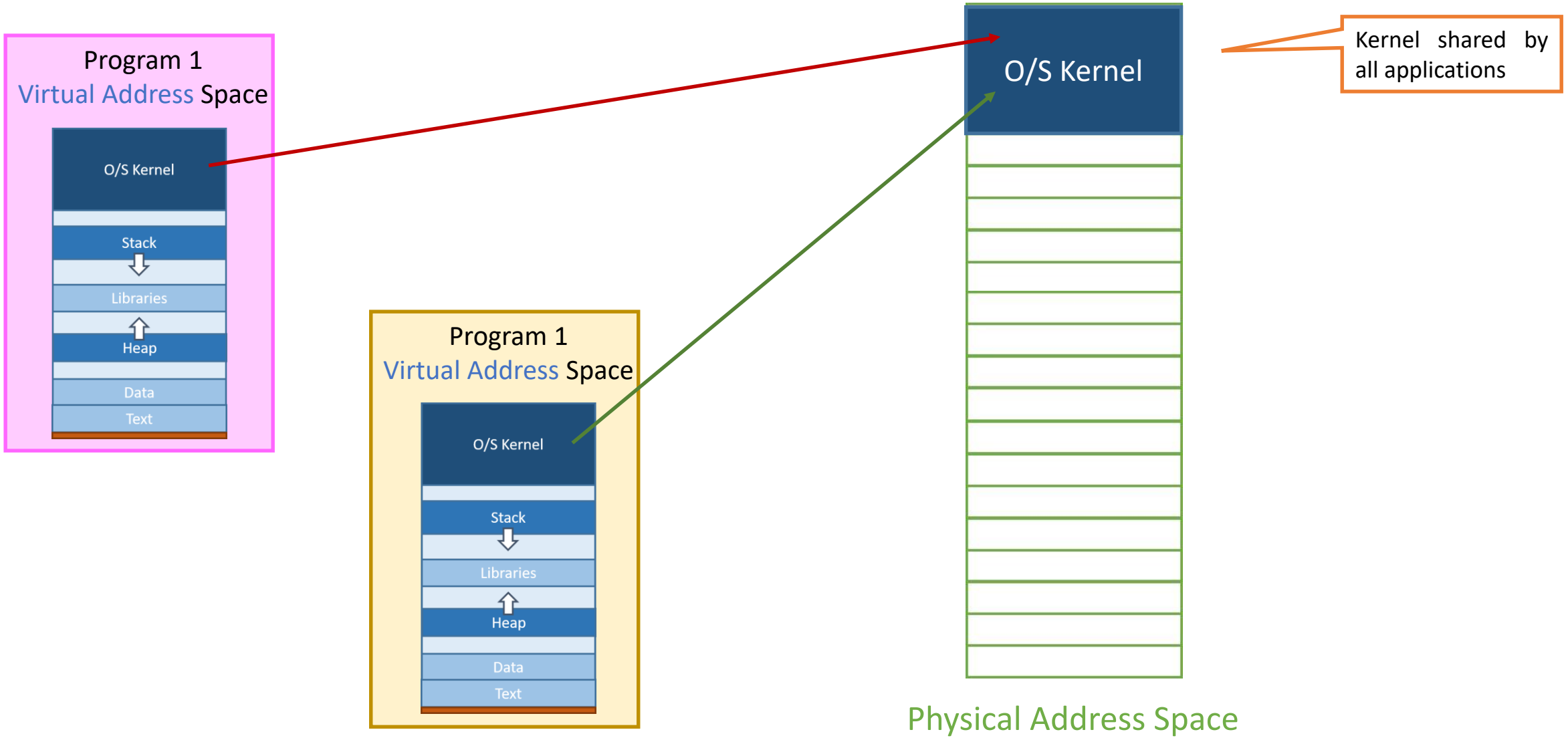


Physical Address Space

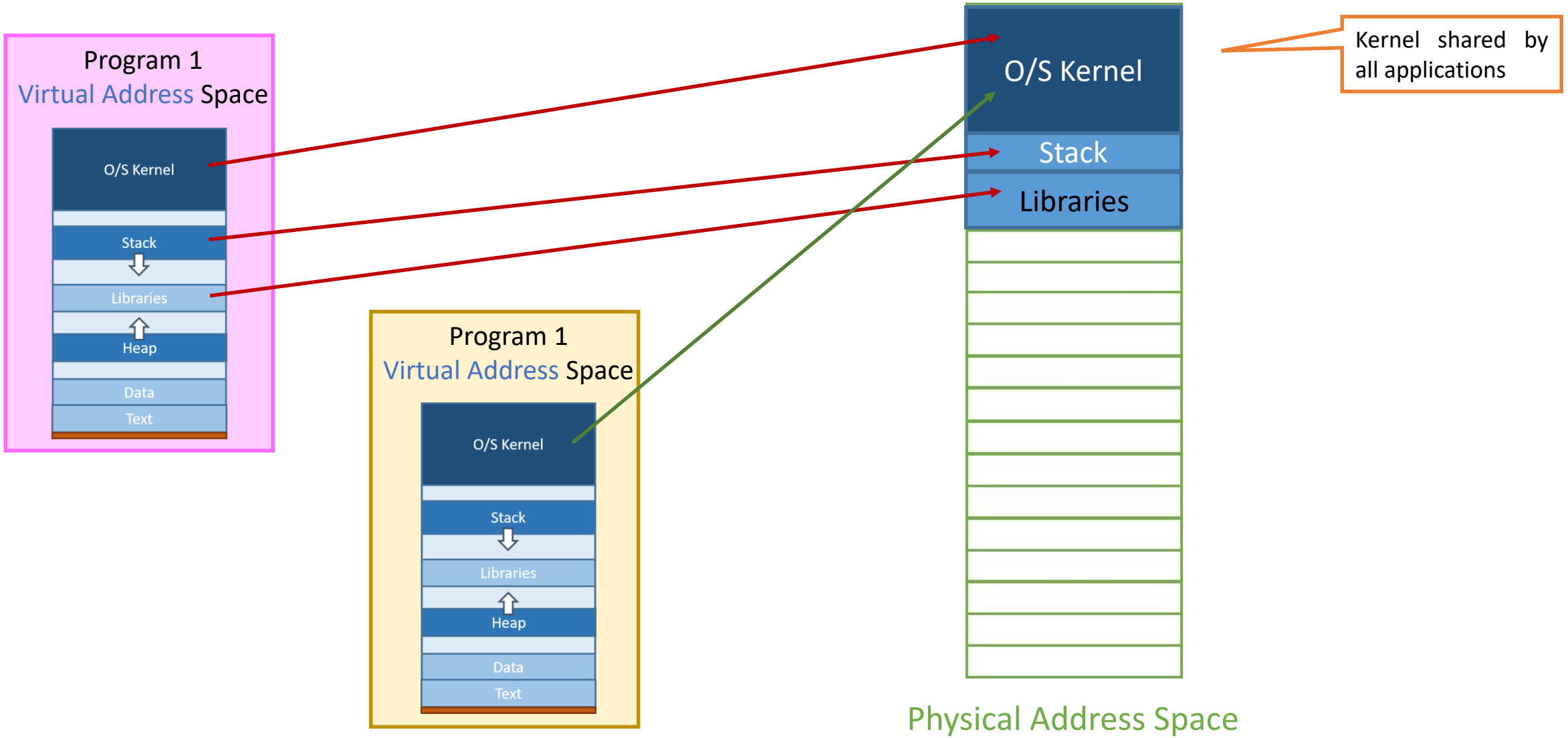
Linux Virtual Address Space



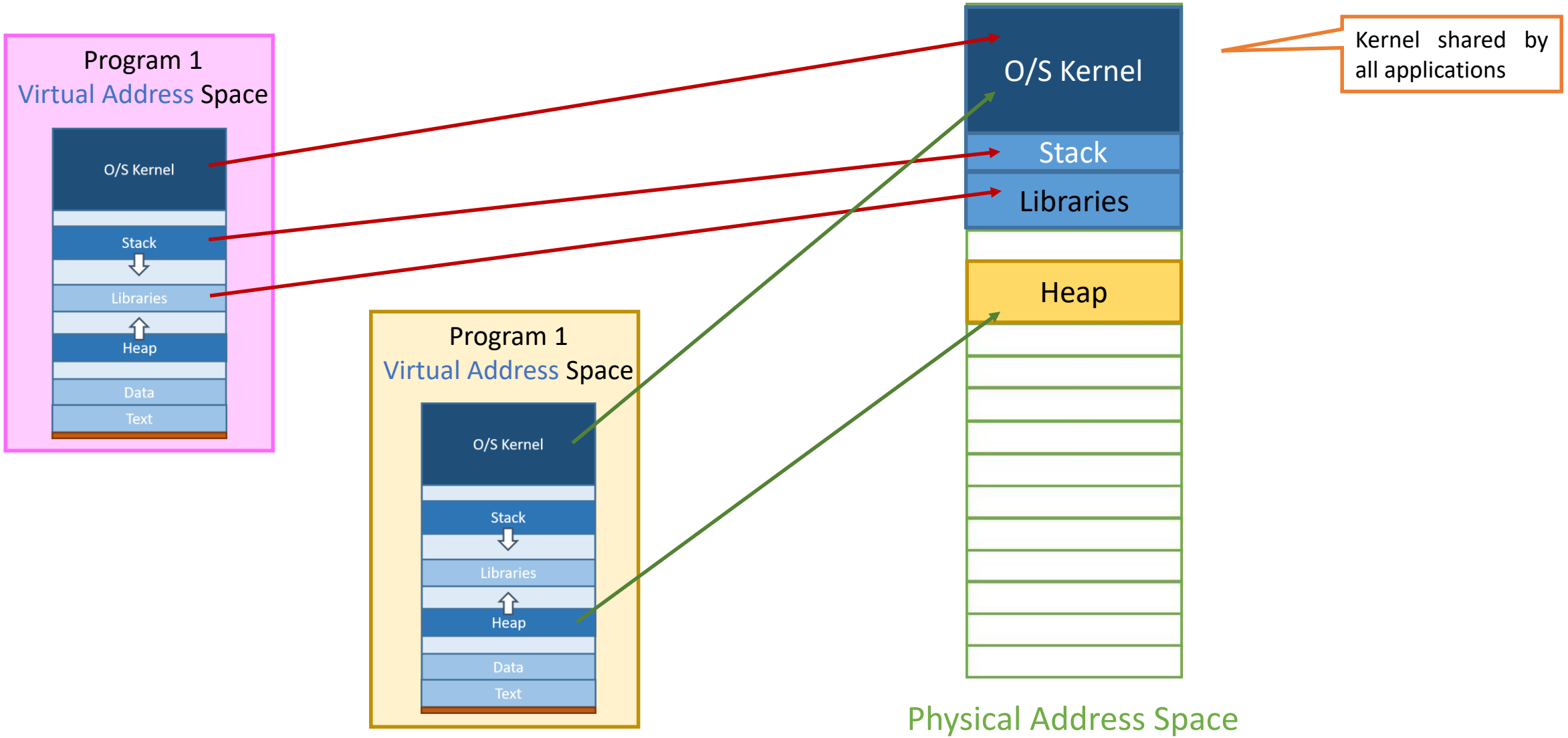
Linux Virtual Address Space



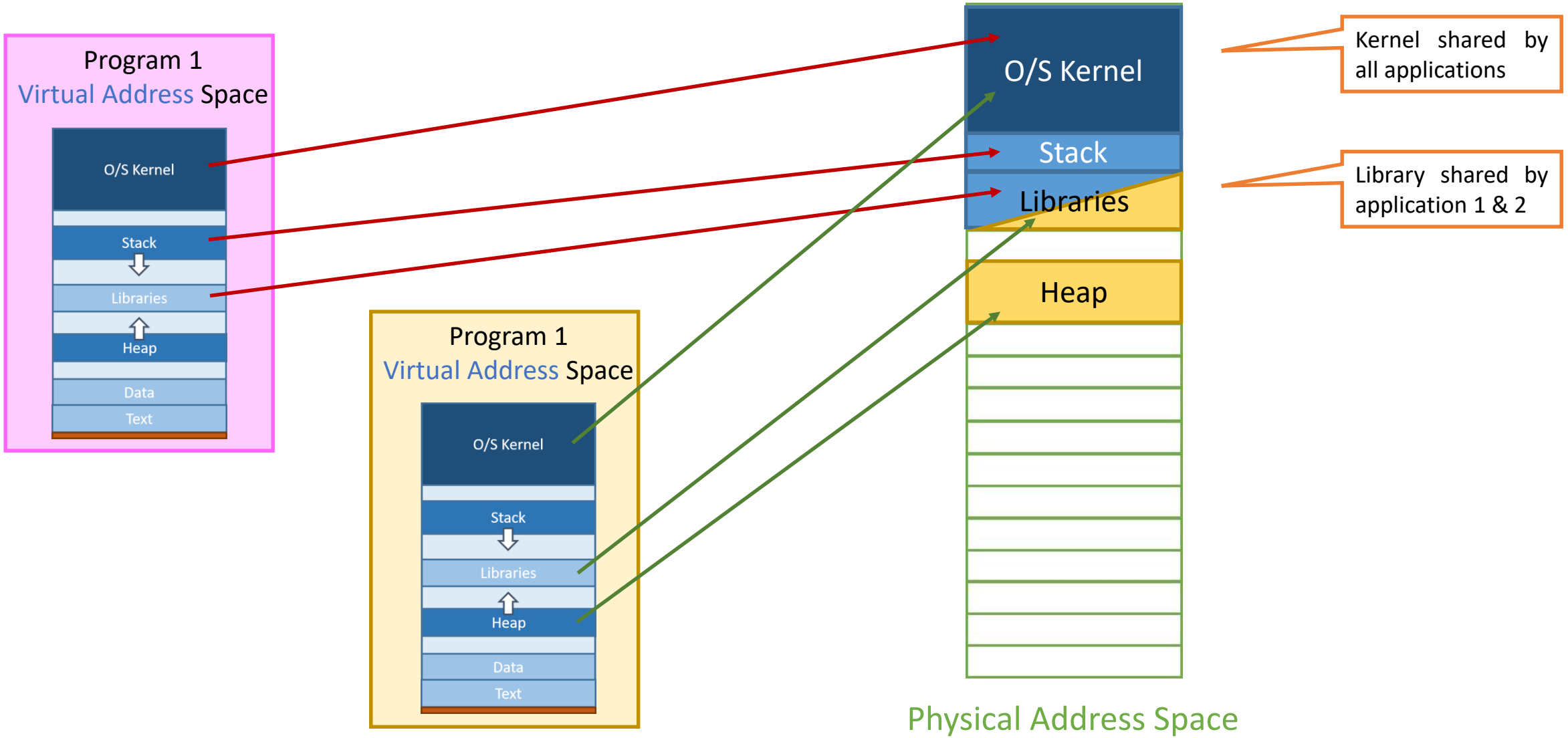
Linux Virtual Address Space



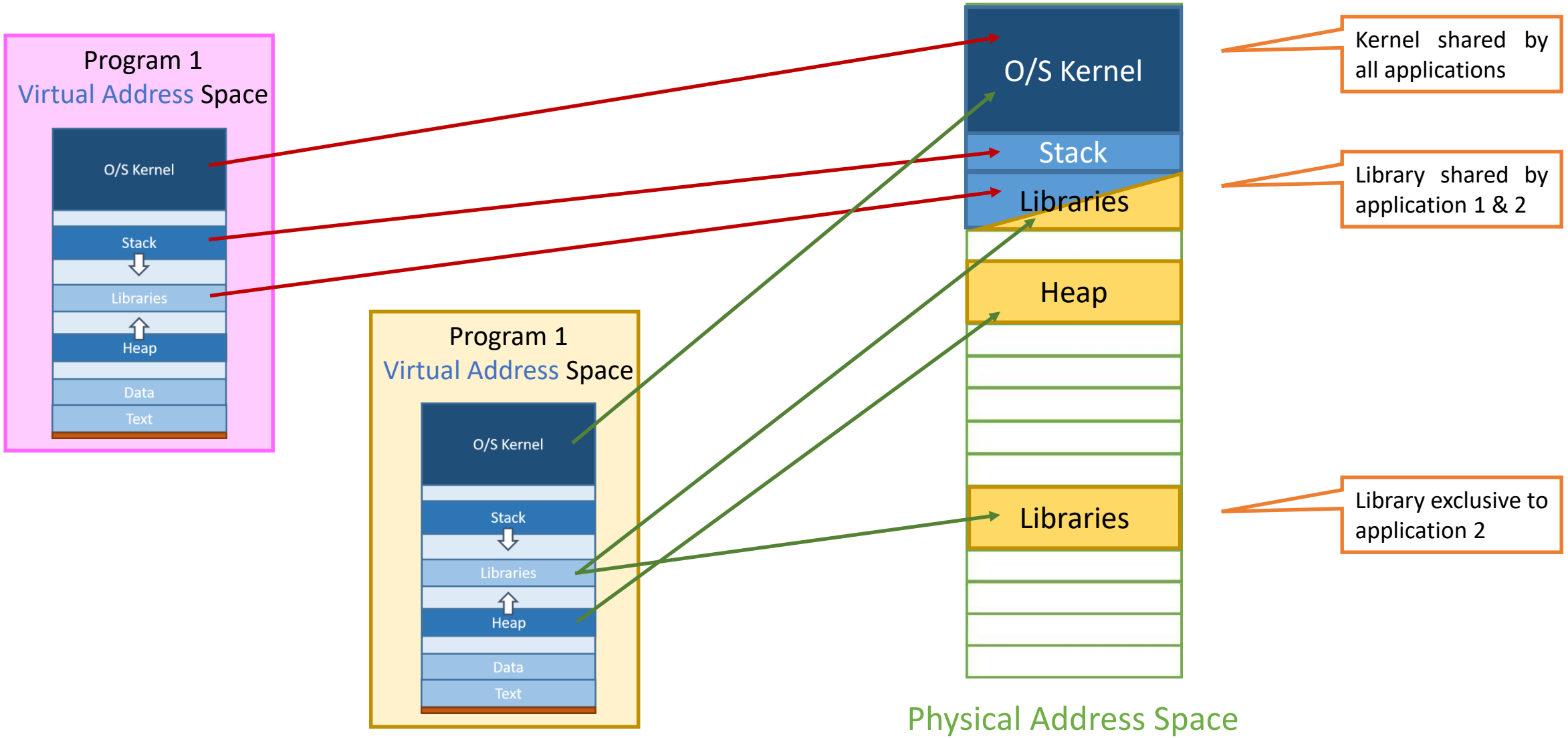
Linux Virtual Address Space



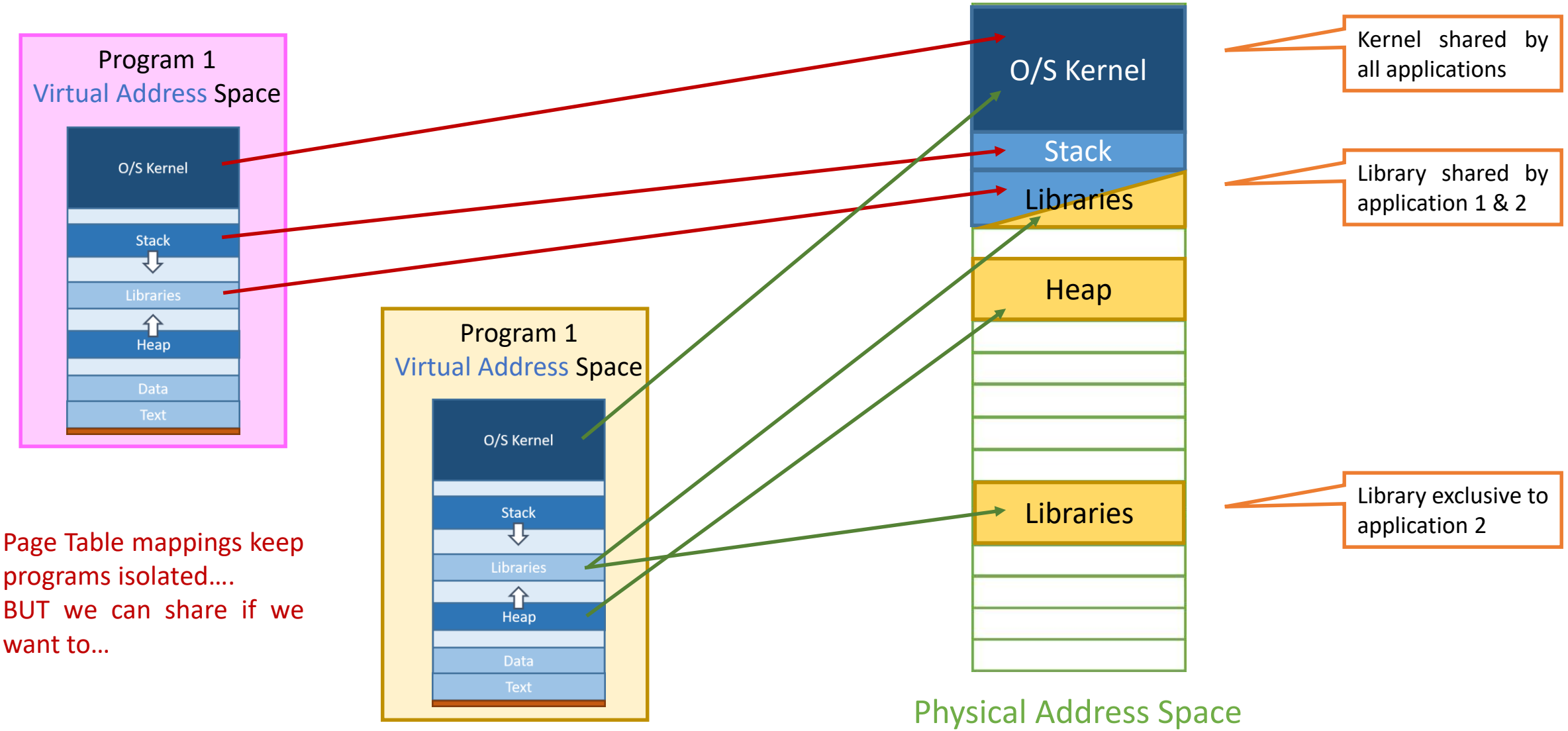
Linux Virtual Address Space



Linux Virtual Address Space

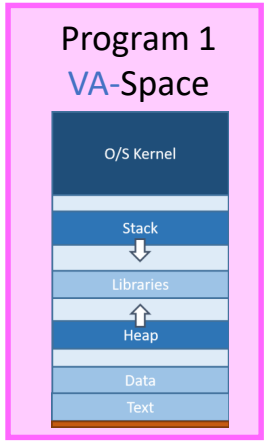


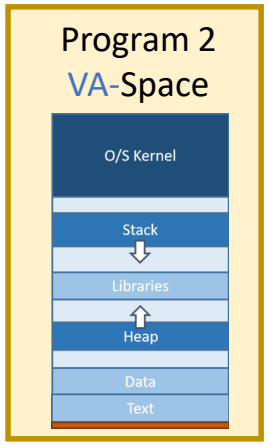
Linux Virtual Address Space

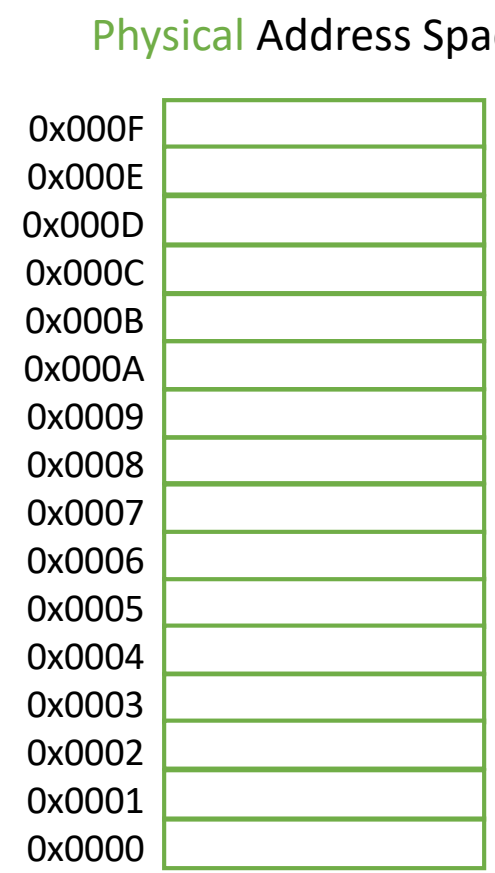


Page Table mappings keep programs isolated….
BUT we can share if we want to…

Linux Mapping Separate Address Spaces

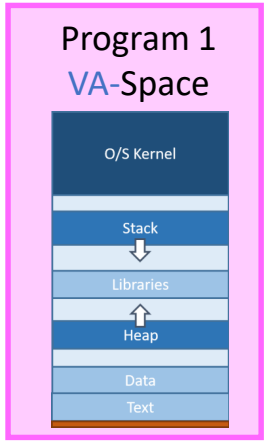




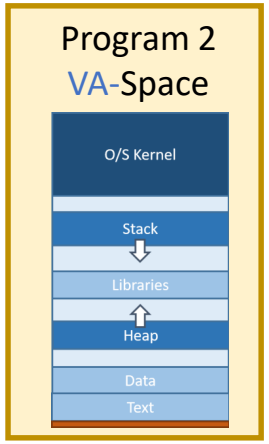


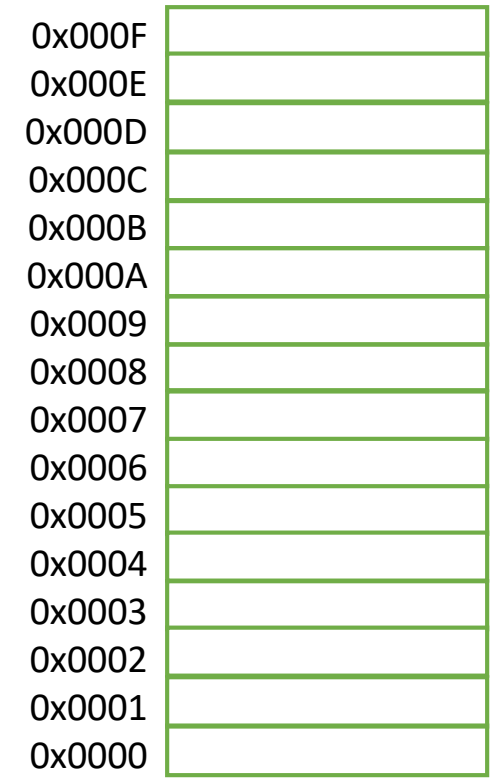
Linux Mapping Separate Address Spaces

LD $R2, 3($R0)



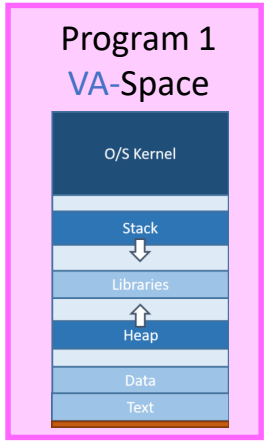
LD $R2, 3($R0)



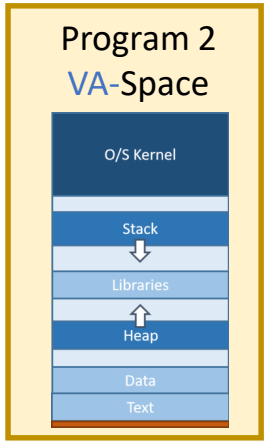


Linux Mapping Separate Address Spaces

LD $R2, 3($R0)

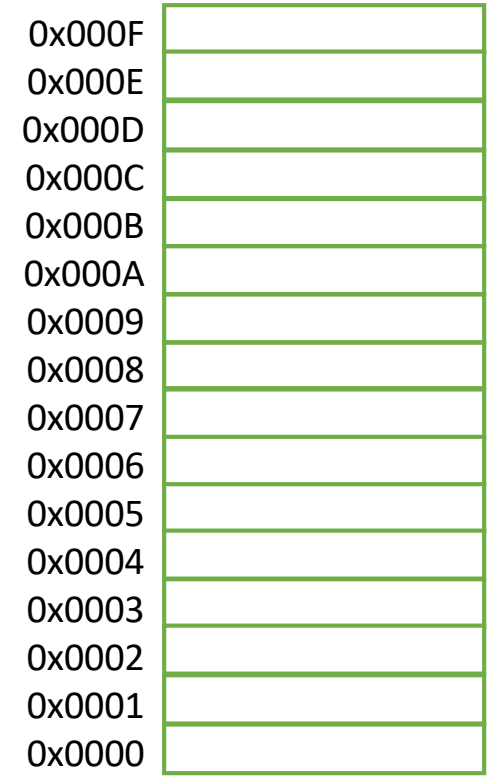


LD $R2, 3($R0)



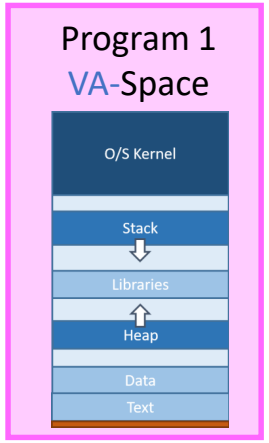
	Physical Page #
0x0 0000	0x0000
0x0 0001	0x0001
0x0 0002	0x0004
0x0 0003	0x0007
.....	.....
0xF FFFF	0x000E

Physical Address Space



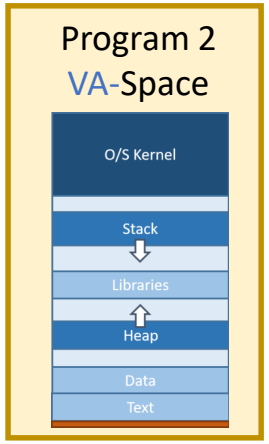
Linux Mapping Separate Address Spaces

LD $R2, 3($R0)



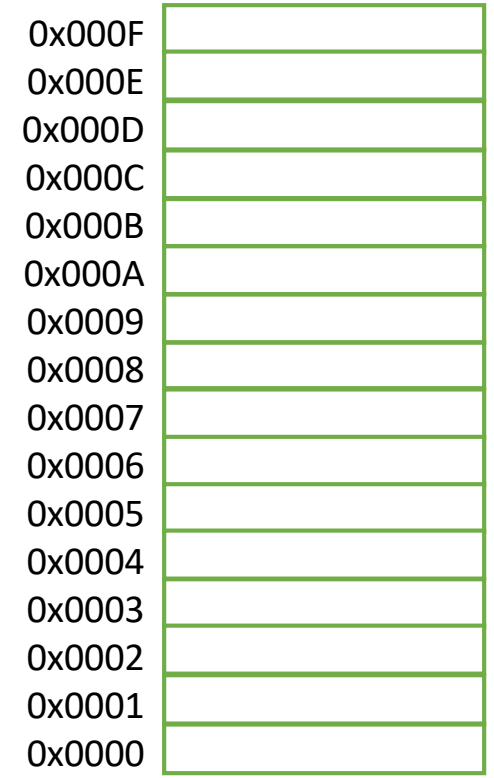
	Physical Page #
0x0 0000	0x0000
0x0 0001	0x0001
0x0 0002	0x0004
0x0 0003	0x0007
.....	.....
0xF FFFF	0x000E

LD $R2, 3($R0)



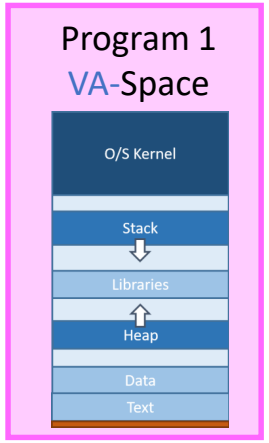
	Physical Page #
0x0 0000	0x000A
0x0 0001	0x0009
0x0 0002	0x0008
0x0 0003	0x0003
.....	.....
0xF FFFF	0x000E

Physical Address Space

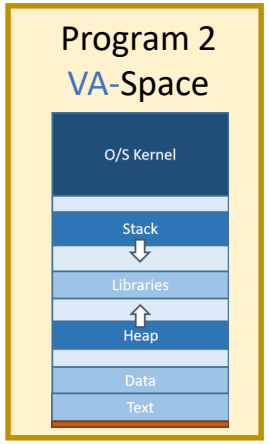


Linux Mapping Separate Address Spaces

LD $R2, 3($R0)



LD $R2, 3($R0)



Each process gets its own Page Table

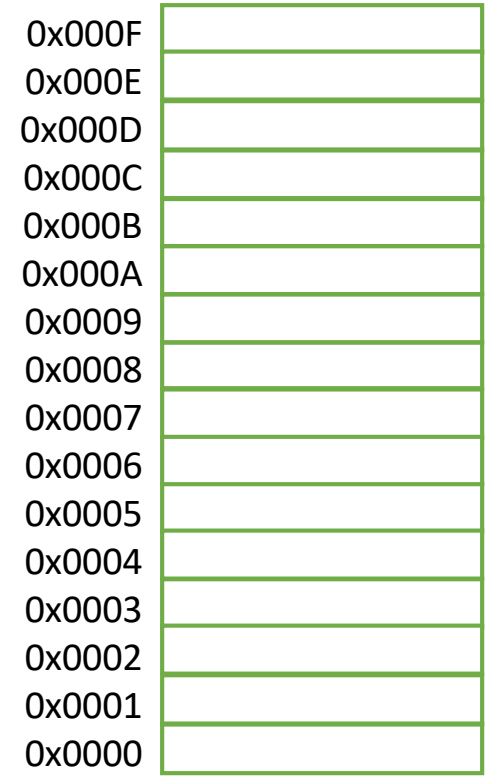
Program 1 Page Table

	Physical Page #
0x0 0000	0x0000
0x0 0001	0x0001
0x0 0002	0x0004
0x0 0003	0x0007
.....	.....
0xF FFFF	0x000E

Program 2 Page Table

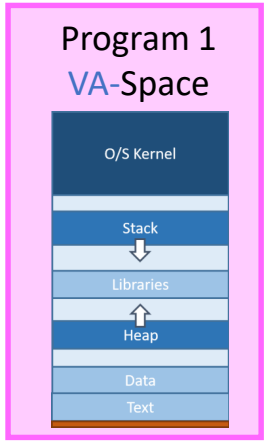
	Physical Page #
0x0 0000	0x000A
0x0 0001	0x0009
0x0 0002	0x0008
0x0 0003	0x0003
.....	.....
0xF FFFF	0x000E

Physical Address Space

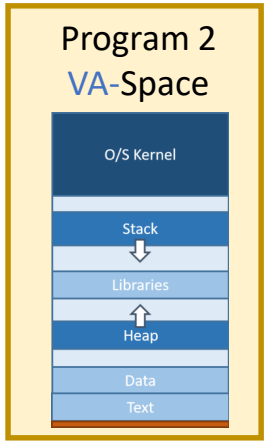


Linux Mapping Separate Address Spaces

LD $R2, 3($R0)



LD $R2, 3($R0)



Each process gets its own Page Table

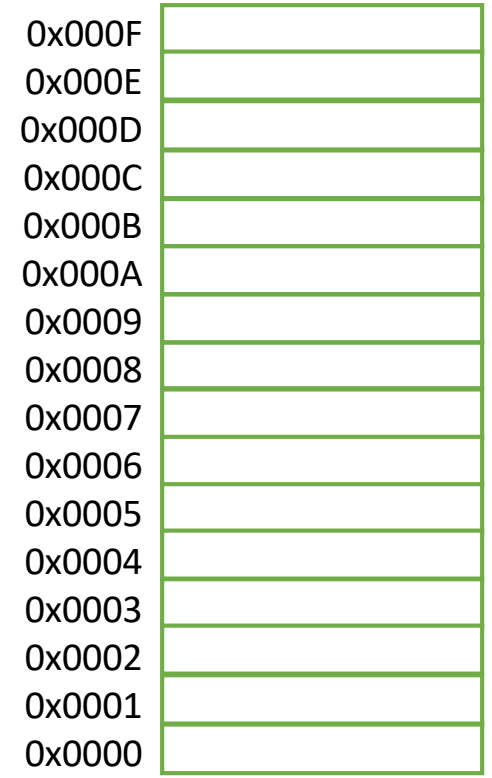
Program 1 Page Table

	Physical Page #
0x0 0000	0x0000
0x0 0001	0x0001
0x0 0002	0x0004
0x0 0003	0x0007
.....	.....
0xF FFFF	0x000E

Program 2 Page Table

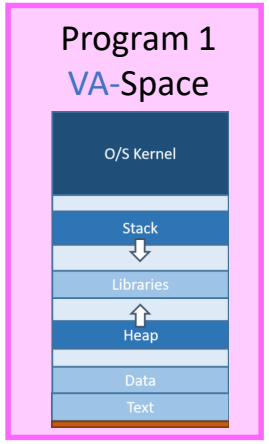
	Physical Page #
0x0 0000	0x000A
0x0 0001	0x0009
0x0 0002	0x0008
0x0 0003	0x0003
.....	.....
0xF FFFF	0x000E

Physical Address Space



Linux Mapping Separate Address Spaces

LD $R2, 3($R0)

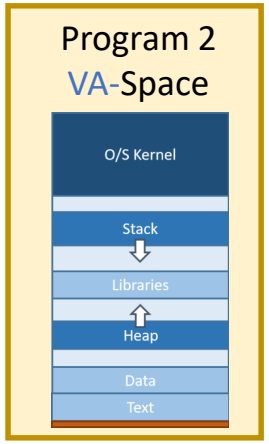


VA=3

Program 1 Page Table

	Physical Page #
0x0 0000	0x0000
0x0 0001	0x0001
0x0 0002	0x0004
0x0 0003	0x0007
.....	.....
0xF FFFF	0x000E

LD $R2, 3($R0)



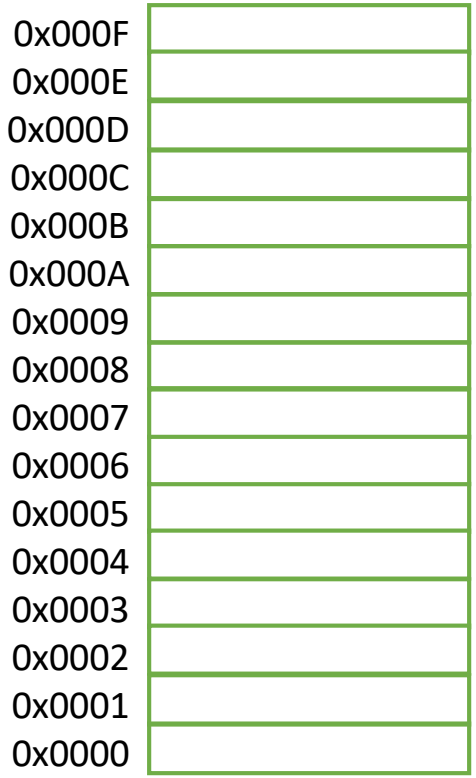
VA=3

Program 2 Page Table

	Physical Page #
0x0 0000	0x000A
0x0 0001	0x0009
0x0 0002	0x0008
0x0 0003	0x0003
.....	.....
0xF FFFF	0x000E

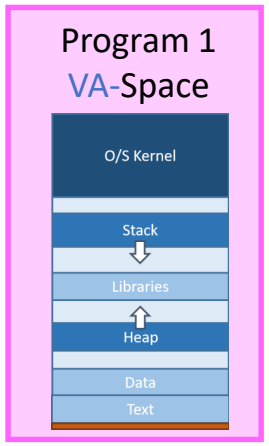
Each process gets its own Page Table

Physical Address Space



Linux Mapping Separate Address Spaces

LD $R2, 3($R0)

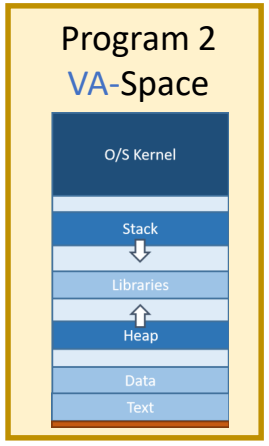


VA=3

Program 1 Page Table

	Physical Page #
0x0 0000	0x0000
0x0 0001	0x0001
0x0 0002	0x0004
0x0 0003	0x0007
.....	.....
0xF FFFF	0x000E

LD $R2, 3($R0)



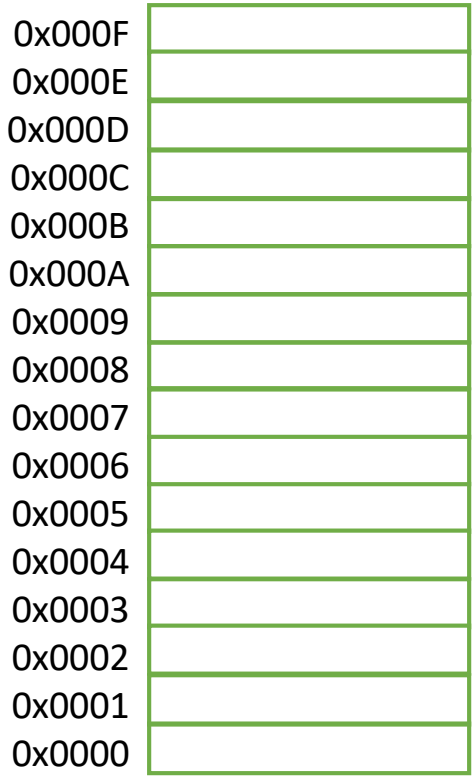
VA=3

Program 2 Page Table

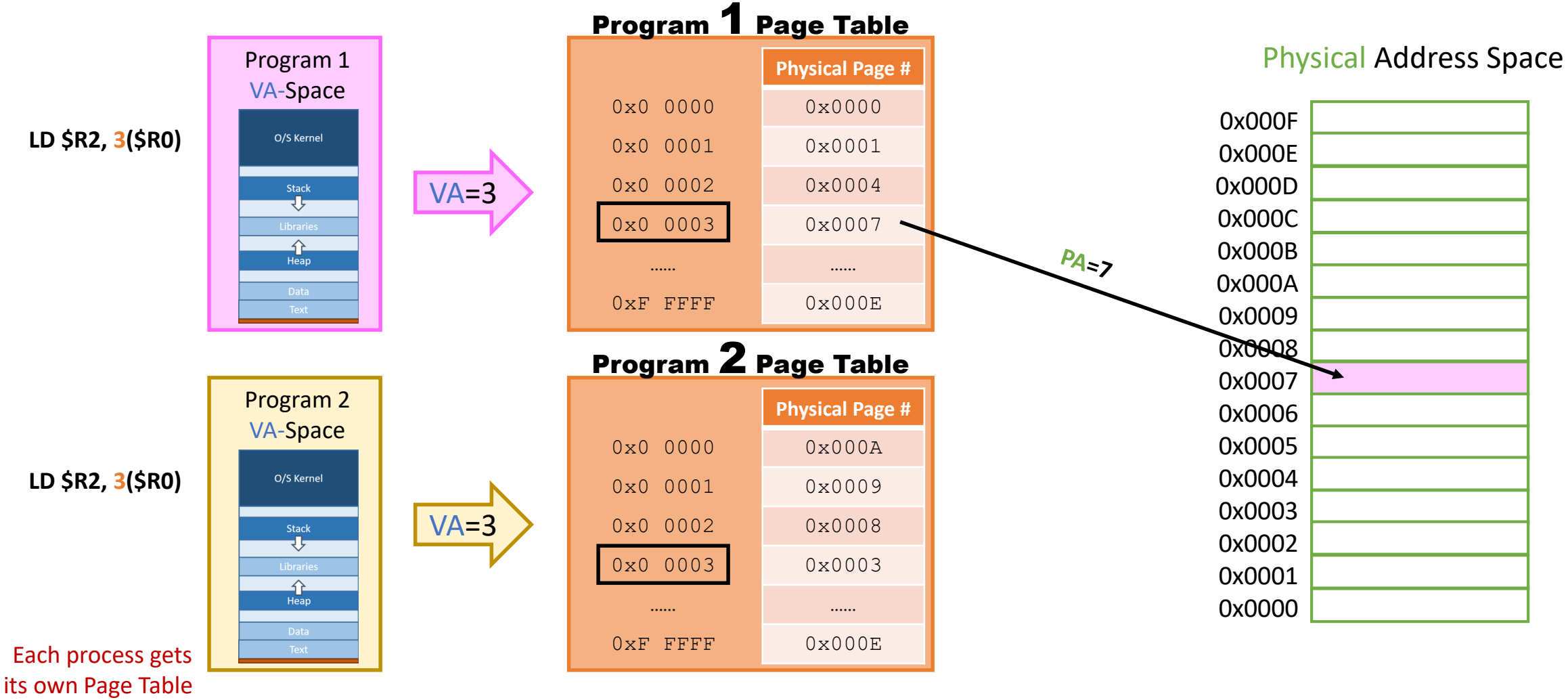
	Physical Page #
0x0 0000	0x000A
0x0 0001	0x0009
0x0 0002	0x0008
0x0 0003	0x0003
.....	.....
0xF FFFF	0x000E

Each process gets its own Page Table

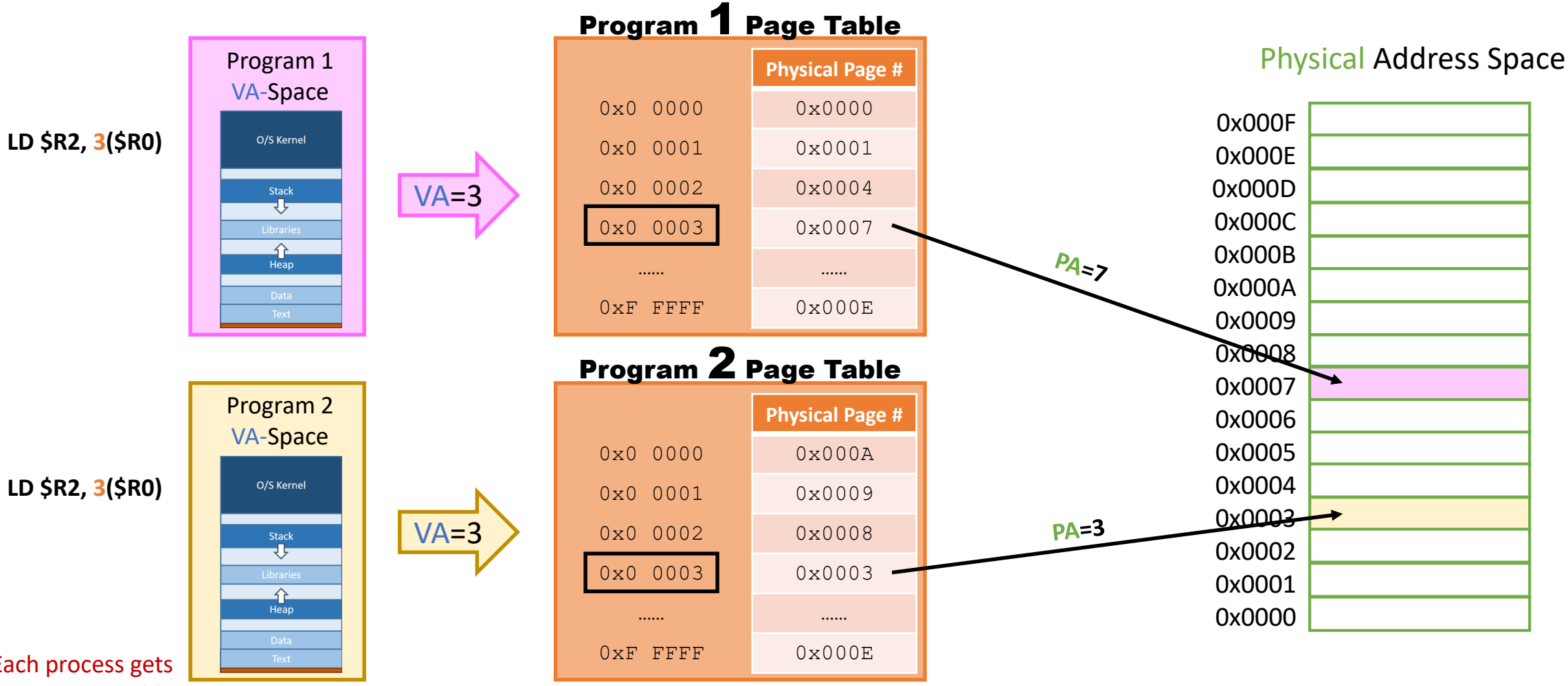
Physical Address Space



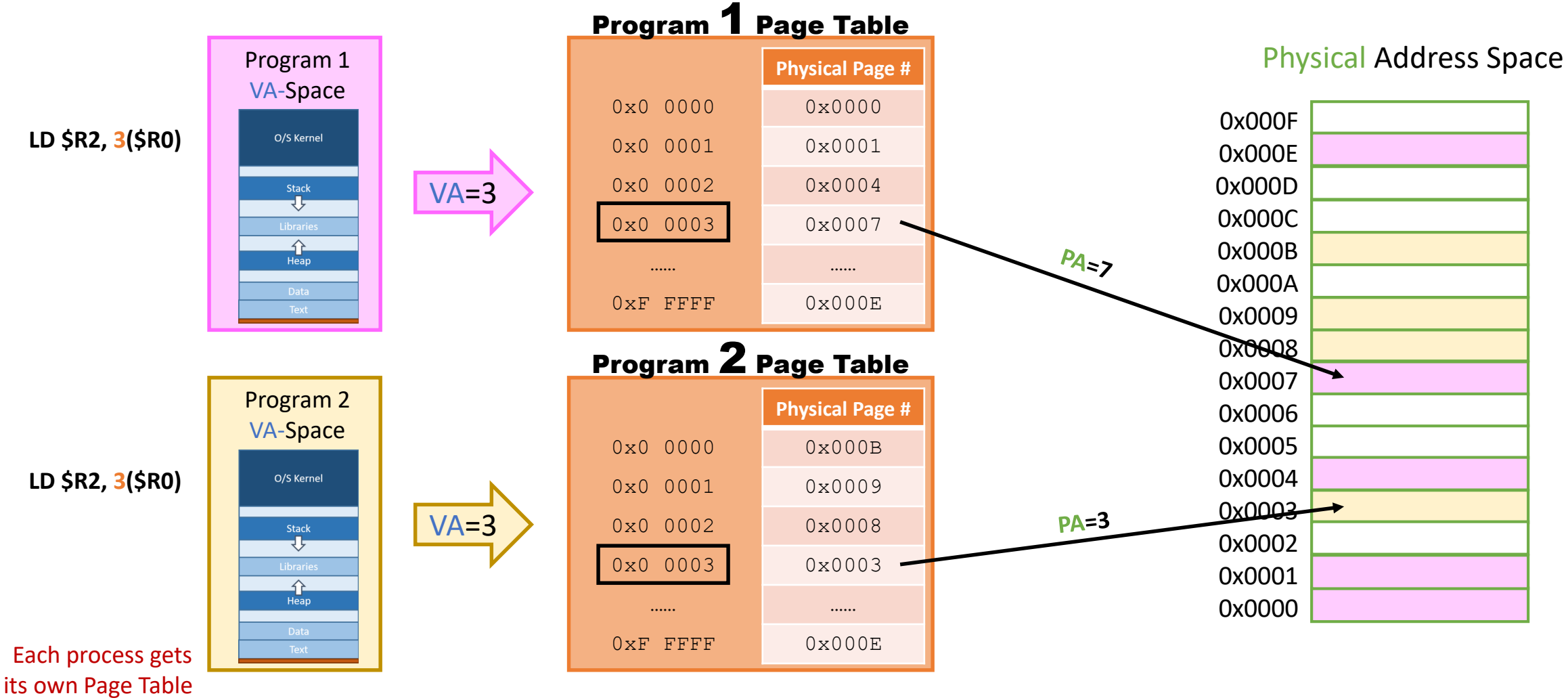
Linux Mapping Separate Address Spaces



Linux Mapping Separate Address Spaces



Linux Mapping Separate Address Spaces



Quiz: Memory Protection

Q: Which of the following Page Table Entries can cause data corruption?

- **Program 1** 0x00003, **Program 2** 0x00003
- **Program 1** 0x00002, **Program 2** 0x00000
- **Program 1** 0xFFFFF, **Program 2** 0xFFFFF
- None of these

Program 1 Page Table

	Physical Page #
0x0 0000	0x0000
0x0 0001	0x0001
0x0 0002	0x0004
0x0 0003	0x0007
.....	.....
0xF FFFF	0x000E

Program 2 Page Table

	Physical Page #
0x0 0000	0x0004
0x0 0001	0x0006
0x0 0002	0x000C
0x0 0003	0x000D
.....	.....
0xF FFFF	0x00FF

Quiz: Memory Protection

Q: Which of the following Page Table Entries can cause data corruption?

- **Program 1** 0x00003, **Program 2** 0x00003
- **Program 1** 0x00002, **Program 2** 0x00000
- **Program 1** 0xFFFFF, **Program 2** 0xFFFFF
- None of these

A: Program 1 0x00002, Program 2 0x00000.

These Virtual Addresses point to the same Physical Address. This can cause data corruption if care is not taken. These programs can safely share data, however.

Program 1 Page Table

	Physical Page #
0x0 0000	0x0000
0x0 0001	0x0001
0x0 0002	0x0004
0x0 0003	0x0007
.....	.....
0xF FFFF	0x000E

Program 2 Page Table

	Physical Page #
0x0 0000	0x0004
0x0 0001	0x0006
0x0 0002	0x000C
0x0 0003	0x000D
.....	.....
0xF FFFF	0x00FF

Quincy Flint

Making VM Fast

Quiz: Memory Access under VM

Q: Which of the following occur for *each* memory access under Virtual Memory? Select all that apply…

I. Translate the address
II. Load data from disk
III. Update the cache
IV. Reference the Page Table
V. Update the Page Table
VI. Access data in RAM

Quiz: Memory Access under VM

Q: Which of the following occur for *each* memory access under Virtual Memory? Select all that apply…

I. Translate the address
II. Load data from disk
III. Update the cache
IV. Reference the Page Table
V. Update the Page Table
VI. Access data in RAM

A:

I. Translate the address
IV. Reference the Page Table
VI. Access data in RAM

The others can occur, but do not happen on every memory access.

Making Virtual Memory Fast

Quincy Flint

Making Virtual Memory Fast

- Virtual Memory solves our 3 memory problems
  - “unlimited” memory, data fragmentation, data corruption

Making Virtual Memory Fast

- Virtual Memory solves our 3 memory problems
  - “unlimited” memory, data fragmentation, data corruption

- Virtual Memory is very costly
  - Each memory access must be translated using the Page Table before fetching

Making Virtual Memory Fast

- Virtual Memory solves our 3 memory problems
  - “unlimited” memory, data fragmentation, data corruption
- Virtual Memory is very costly
  - Each memory access must be translated using the Page Table before fetching
- We need to make the Page Table look-up very fast
  - If not, VM is not tenable…

Making Virtual Memory Fast

- Virtual Memory solves our 3 memory problems
  - “unlimited” memory, data fragmentation, data corruption
- Virtual Memory is very costly
  - Each memory access must be translated using the Page Table before fetching
- We need to make the Page Table look-up very fast
  - If not, VM is not tenable…
  - Cannot do this in software (this adds 10’s of instructions)

Making Virtual Memory Fast

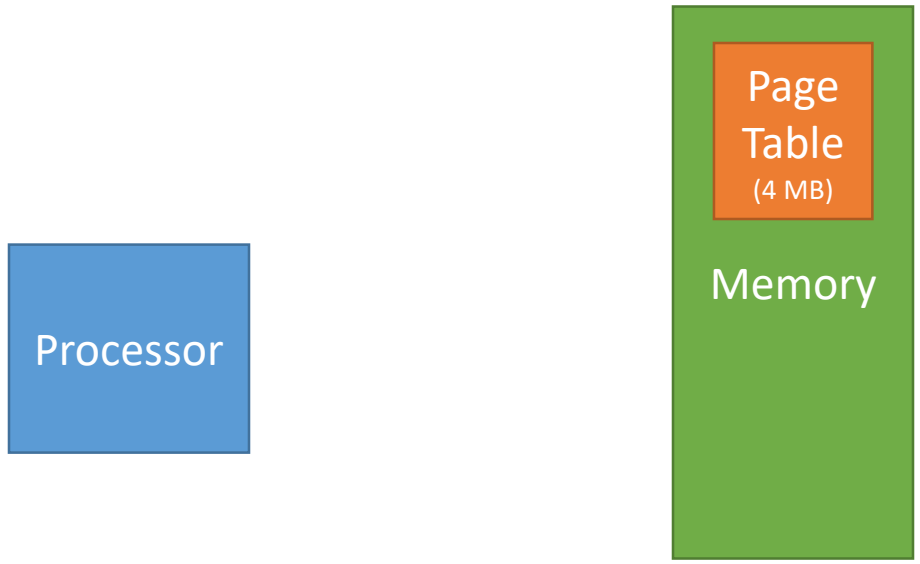
- Virtual Memory solves our 3 memory problems
  - “unlimited” memory, data fragmentation, data corruption
- Virtual Memory is very costly
  - Each memory access must be translated using the Page Table before fetching
- We need to make the Page Table look-up very fast
  - If not, VM is not tenable…
  - Cannot do this in software (this adds 10’s of instructions)
  - Must do this in hardware… use another layer of **cache**

Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast

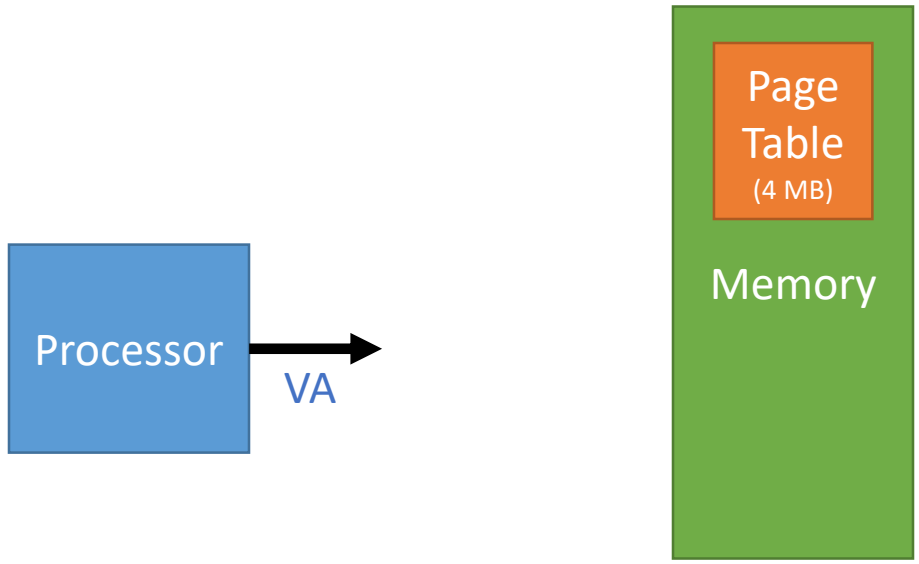
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



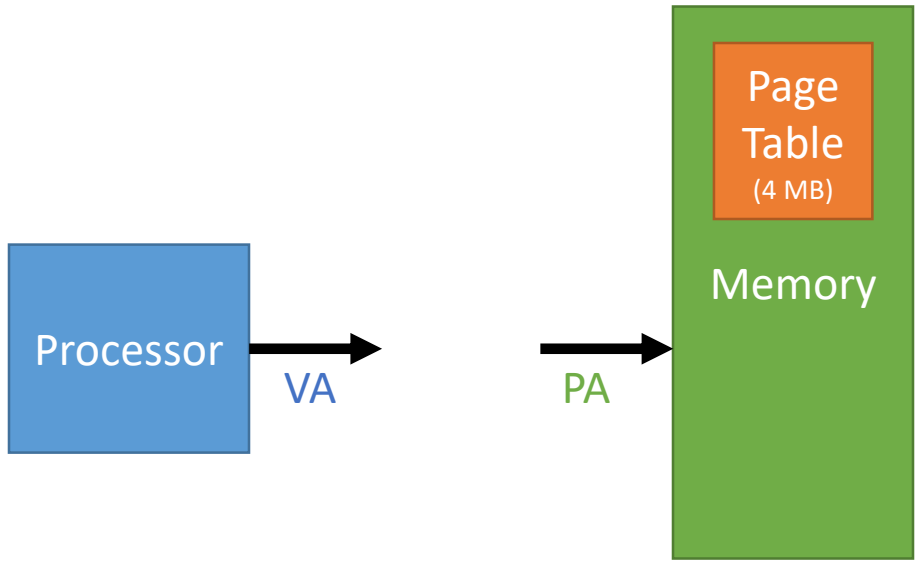
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



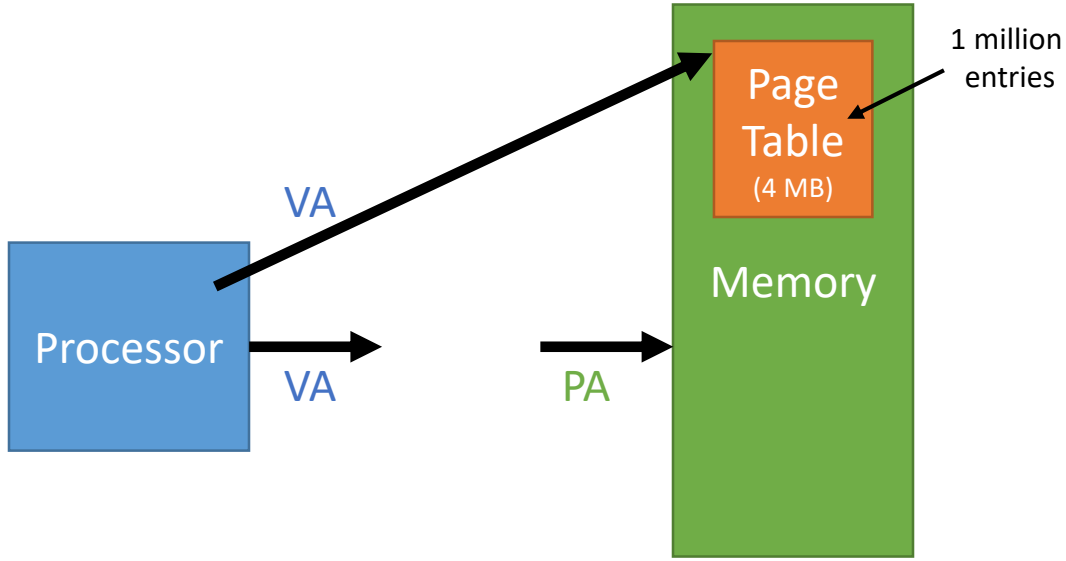
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



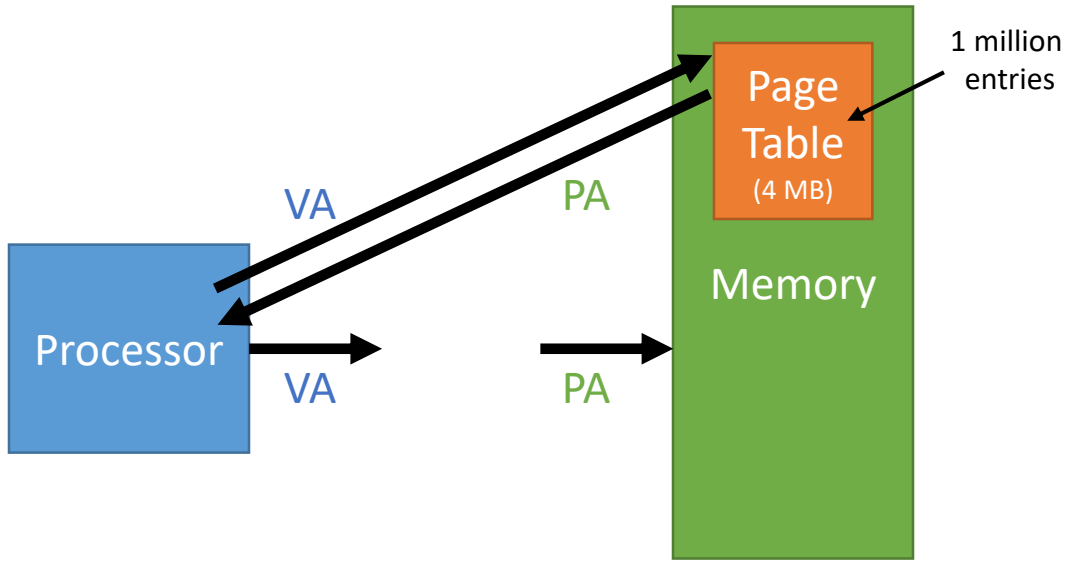
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



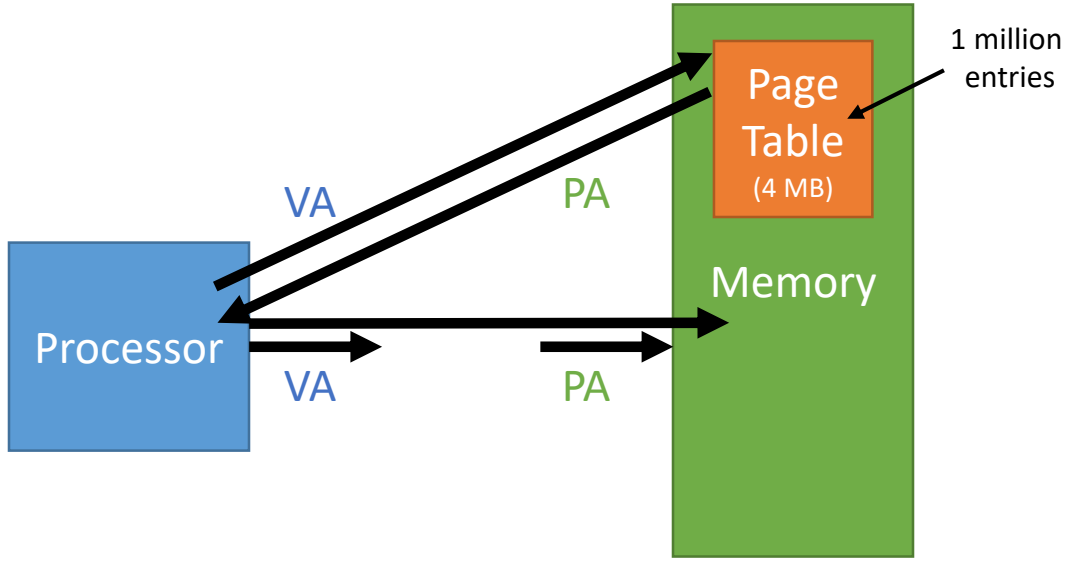
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



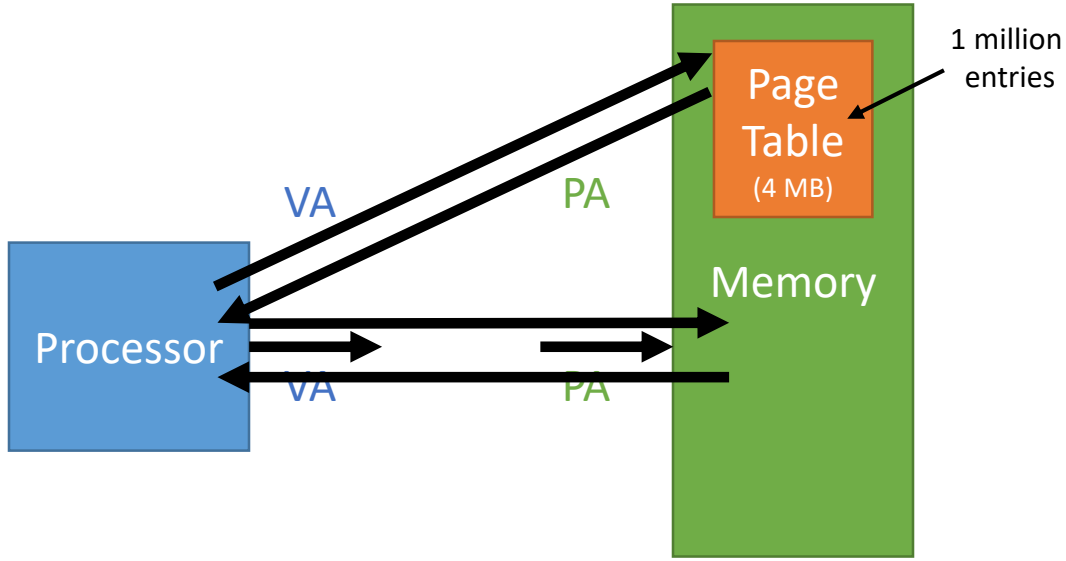
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



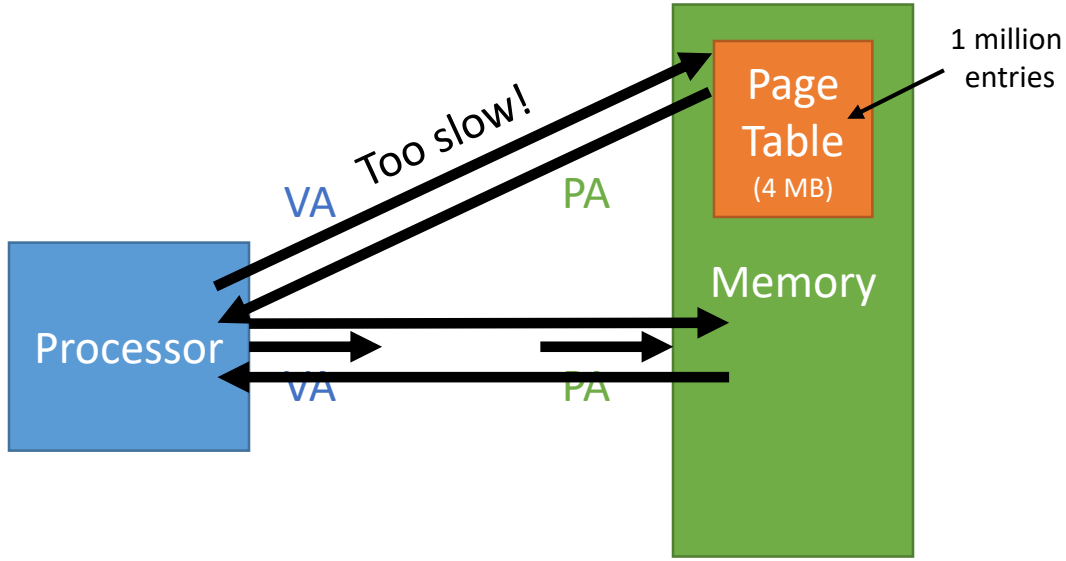
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



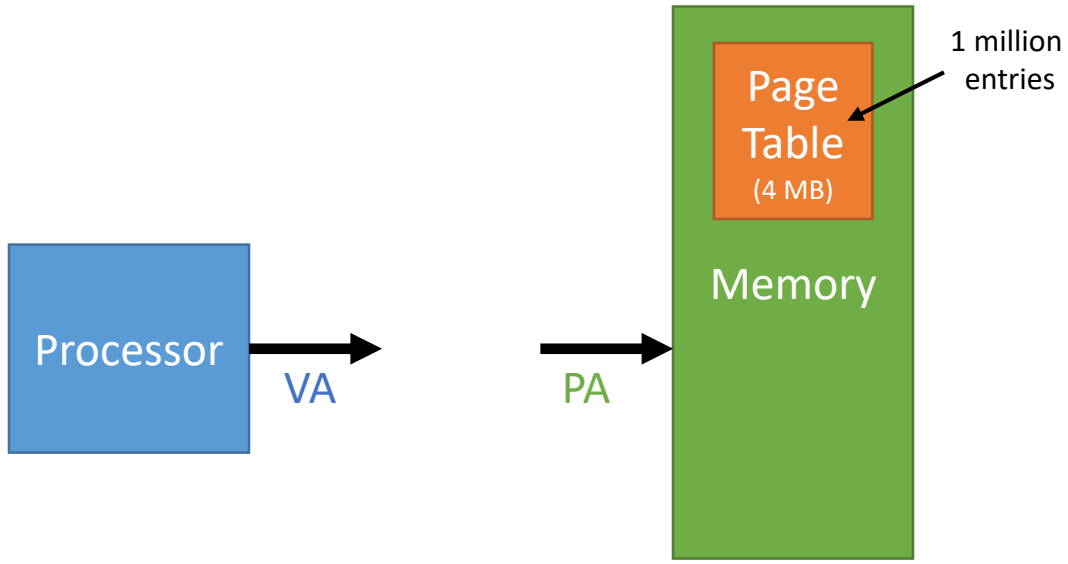
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



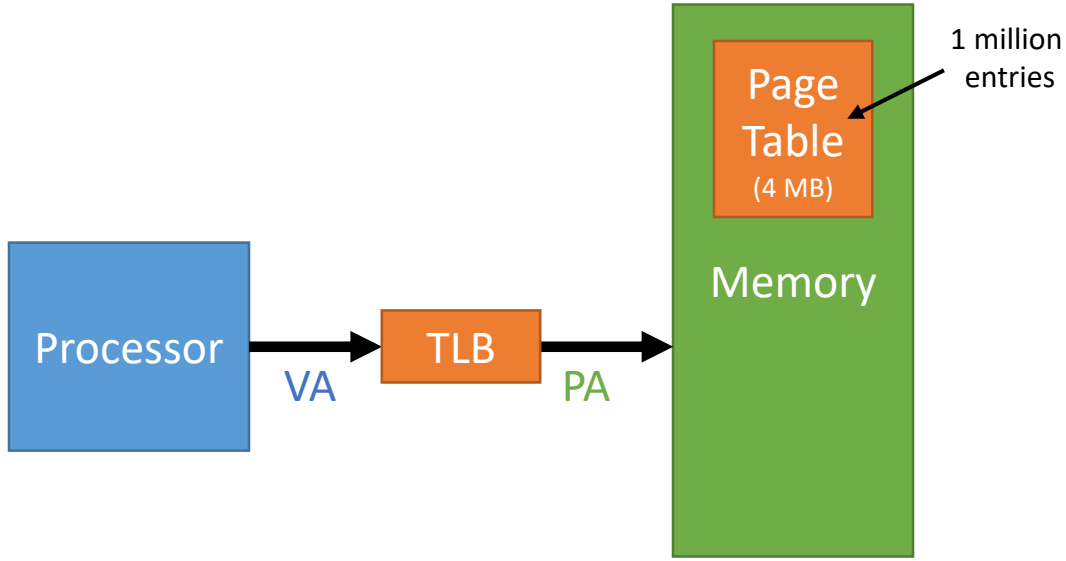
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



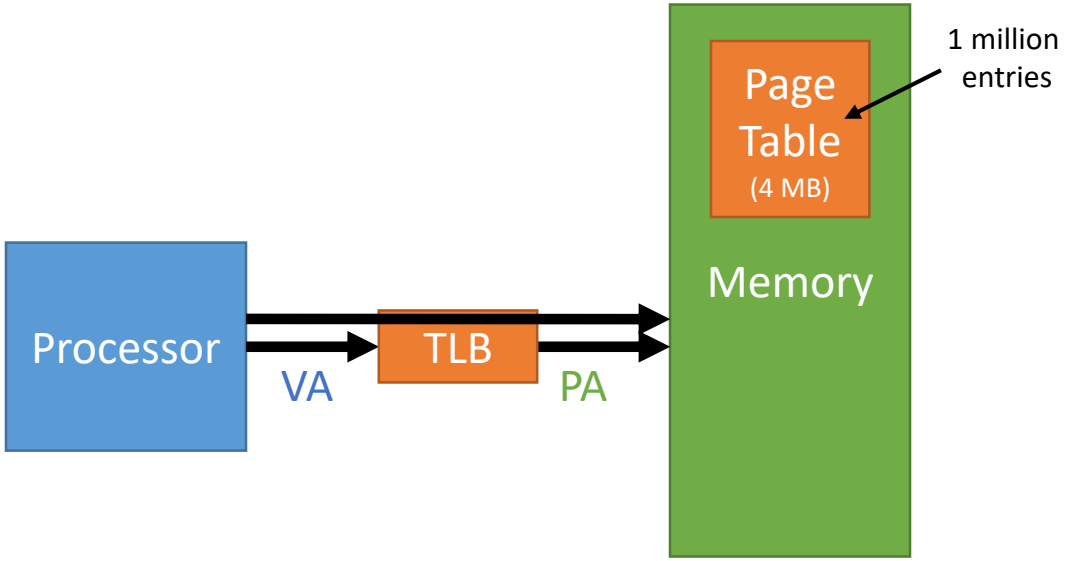
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



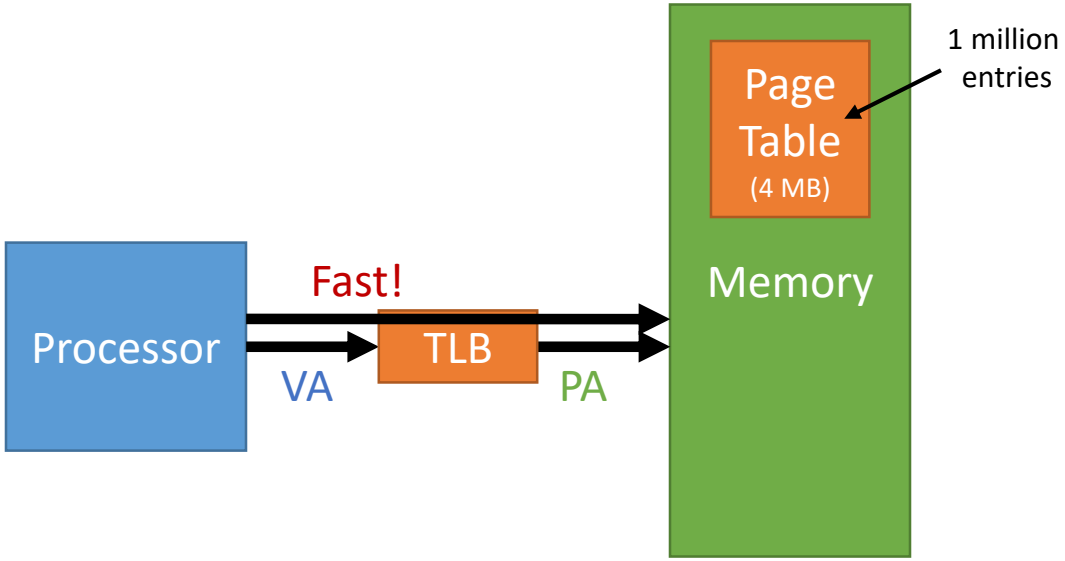
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast



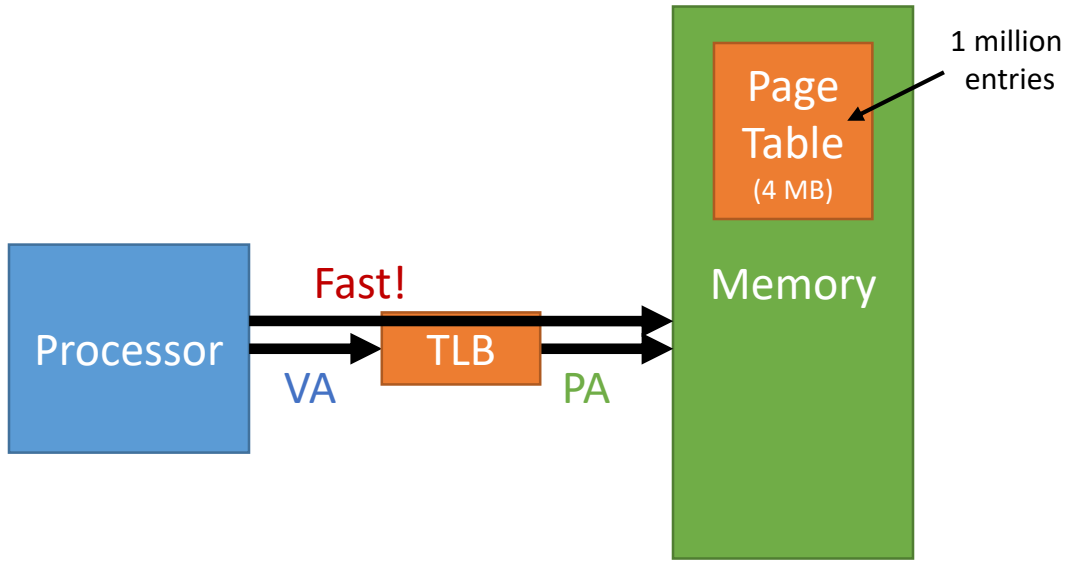
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast
  - Fast: less than 1 cycle access time



Making Virtual Memory Fast: TLB

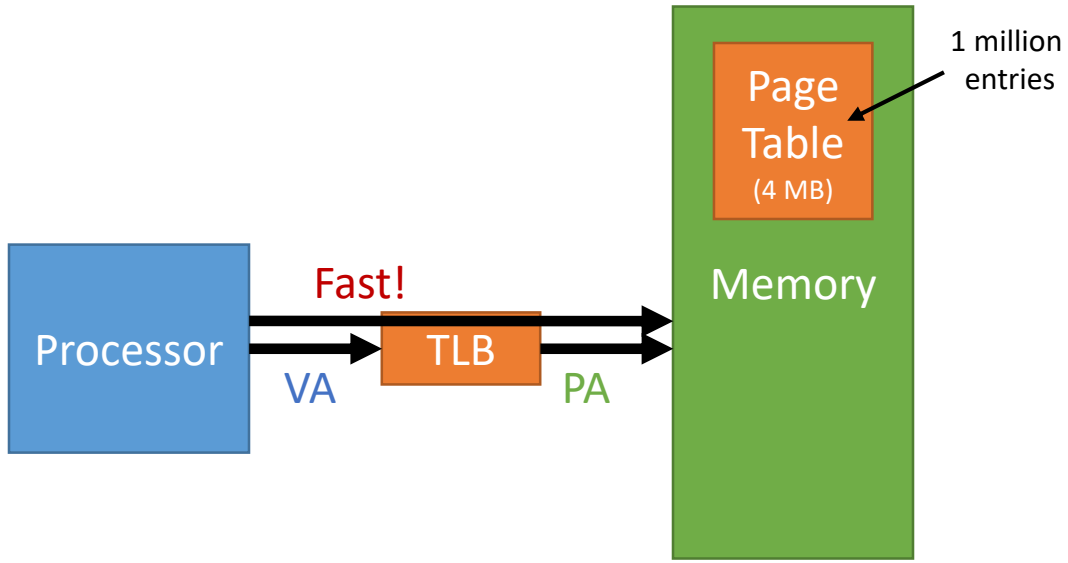
- Translation Lookaside Buffer (TLB): special page table cache to make VM fast
  - Fast: less than 1 cycle access time
  - Small: only stores a few Page Table Entries



Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast
  - Fast: less than 1 cycle access time
  - Small: only stores a few Page Table Entries

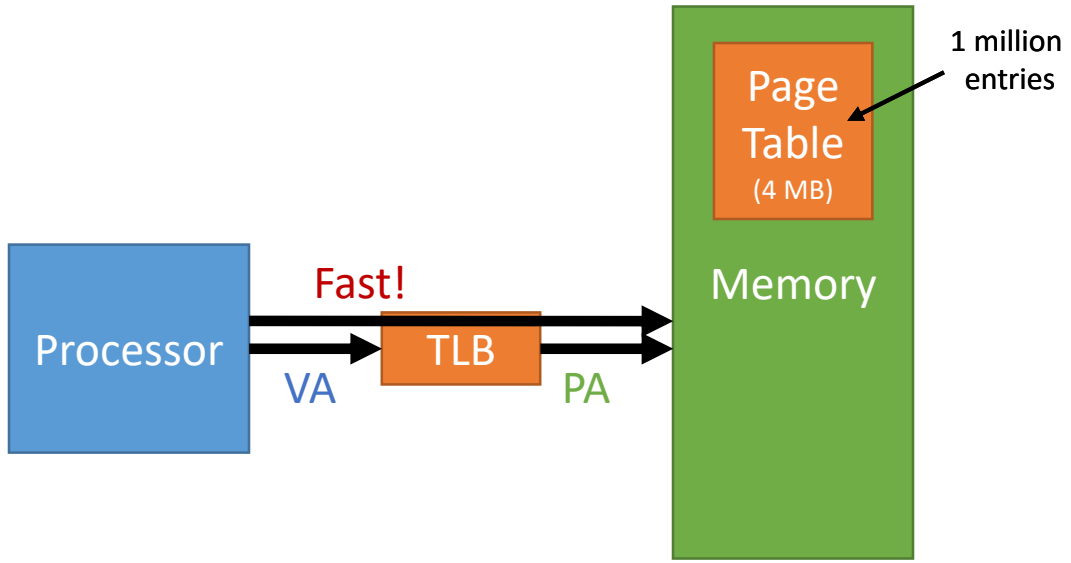
- To be **fast**, the TLB must be **small**



Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast
  - Fast: less than 1 cycle access time
  - Small: only stores a few Page Table Entries

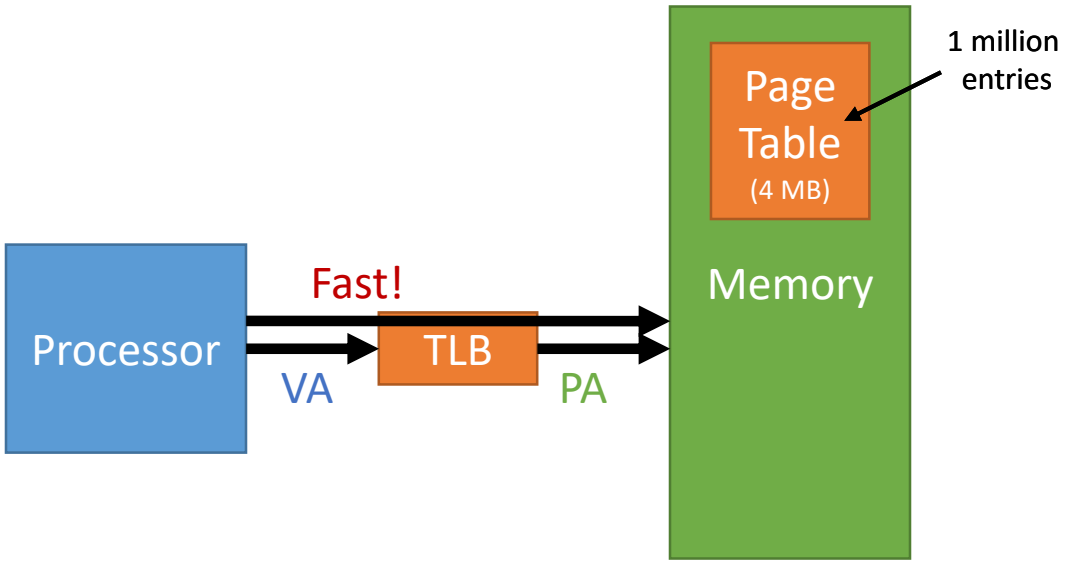
- To be **fast**, the TLB must be **small**
  - Separate TLBs for *instructions* and *data*



Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast
  - Fast: less than 1 cycle access time
  - Small: only stores a few Page Table Entries

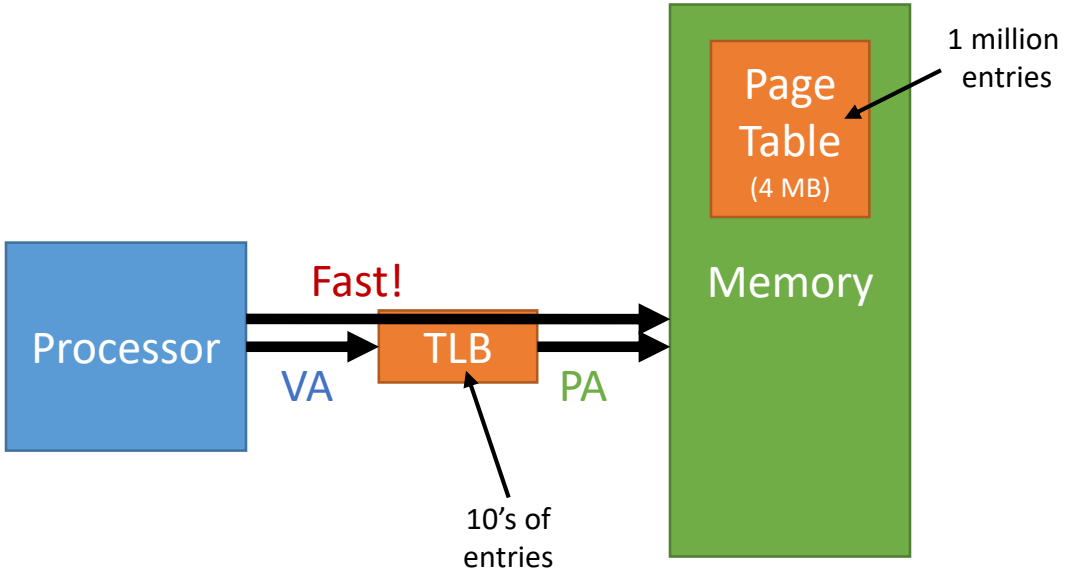
- To be **fast**, the TLB must be **small**
  - Separate TLBs for *instructions* and *data*
  - **4 kB Pages**: 64 entries, 4-way set associative



Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast
  - Fast: less than 1 cycle access time
  - Small: only stores a few Page Table Entries

- To be **fast**, the TLB must be **small**
  - Separate TLBs for *instructions* and *data*
  - **4 kB Pages**: 64 entries, 4-way set associative
  - **2 MB Pages**: 32 entries, 4-way set associative



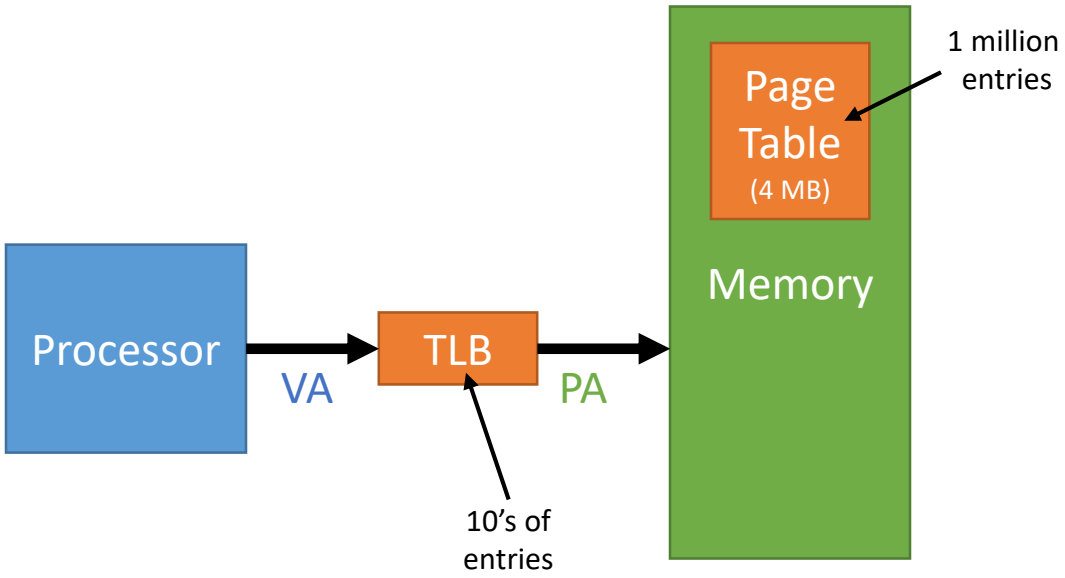
Making Virtual Memory Fast: TLB

- Translation Lookaside Buffer (TLB): special page table cache to make VM fast
  - Fast: less than 1 cycle access time
  - Small: only stores a few Page Table Entries

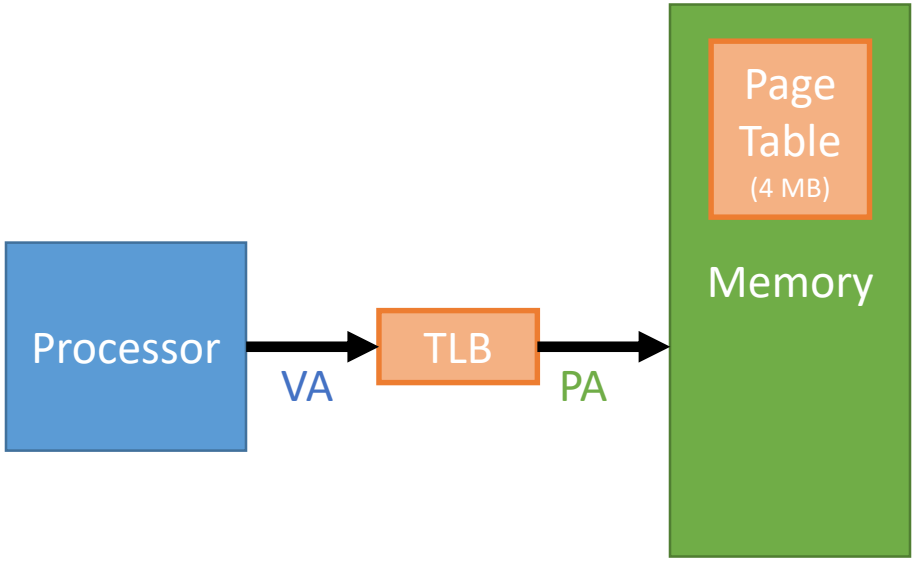
- To be **fast**, the TLB must be **small**
  - Separate TLBs for *instructions* and *data*
  - **4 kB Pages**: 64 entries, 4-way set associative
  - **2 MB Pages**: 32 entries, 4-way set associative

Page Table has 1 million entry, TLB only has 10's of entries??

Each Page maps 4k addresses, exploit principal of locality!

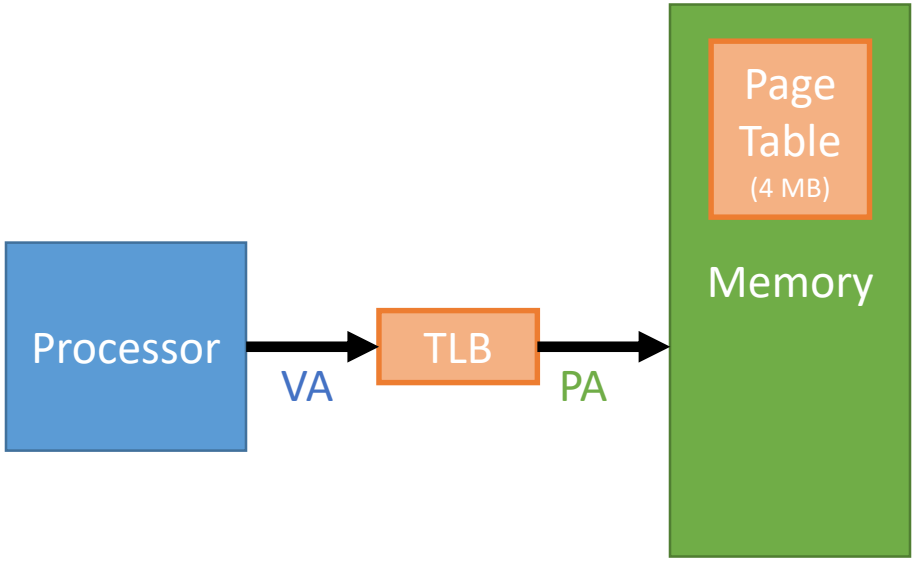


What happens when we access memory?



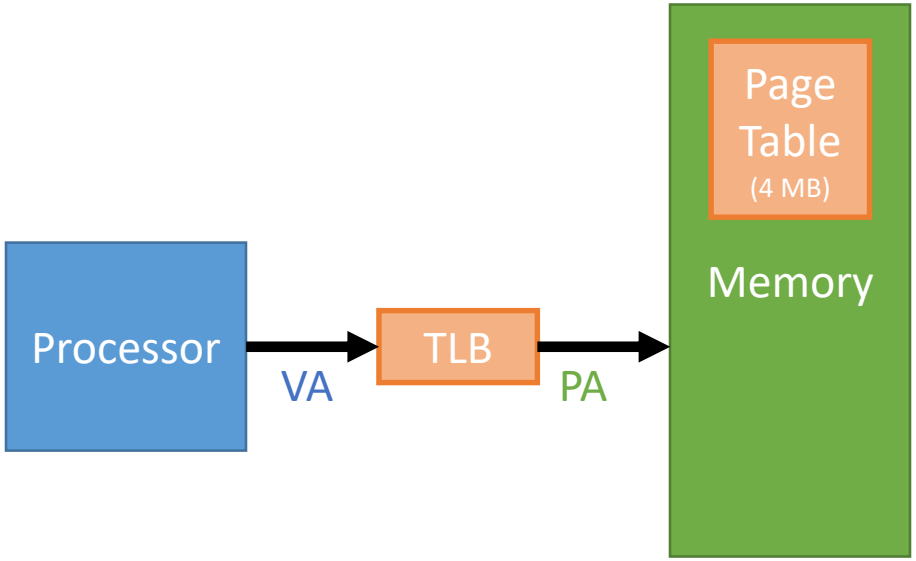
What happens when we access memory?

- Page is in RAM [Good]



What happens when we access memory?

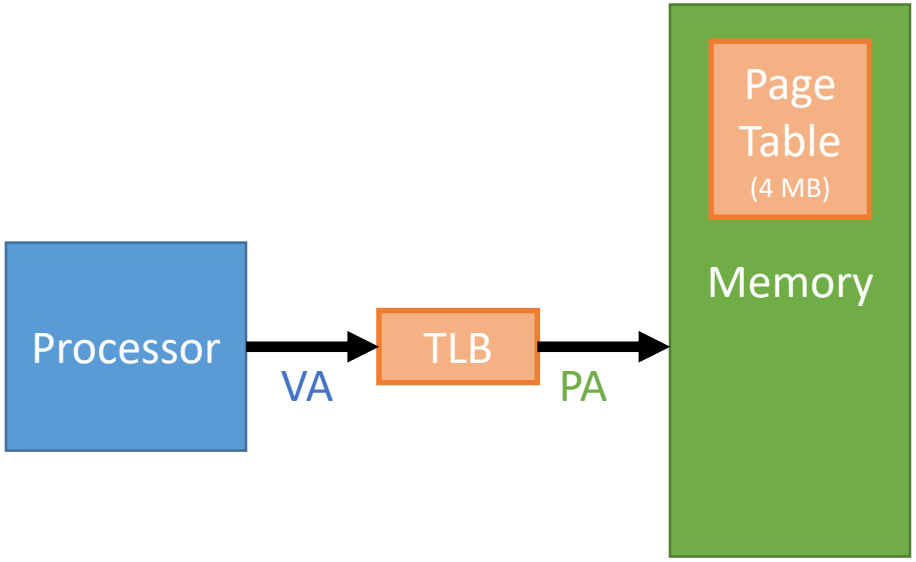
- Page is in RAM [Good]
- Page is *not* in RAM [Bad]



What happens when we access memory?

- Page is in RAM [Good]
  - Page Table Entry in the TLB
    - Best performance

- Page is *not* in RAM [Bad]

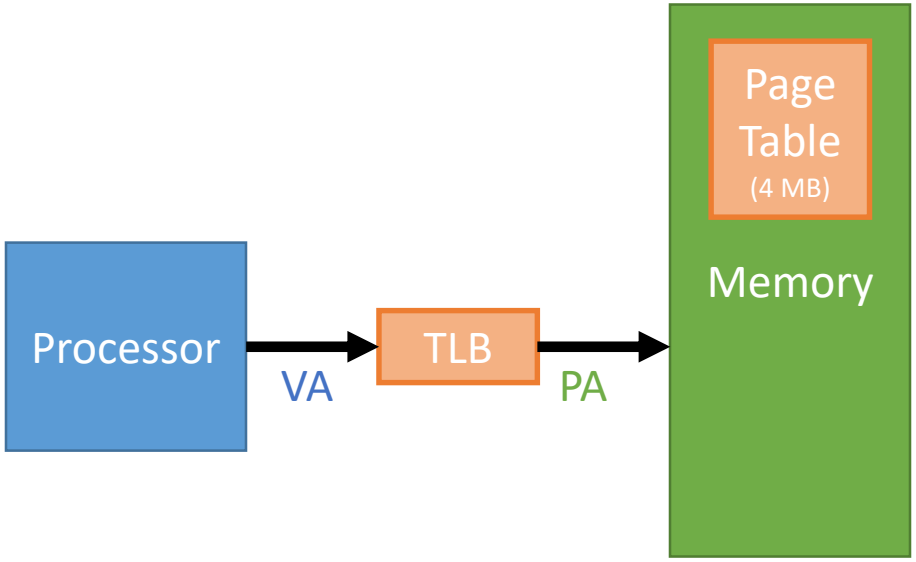


What happens when we access memory?

- Page is in RAM [Good]
  - Page Table Entry in the TLB
    - Best performance

~ 1 cycle to translate, then can go to RAM

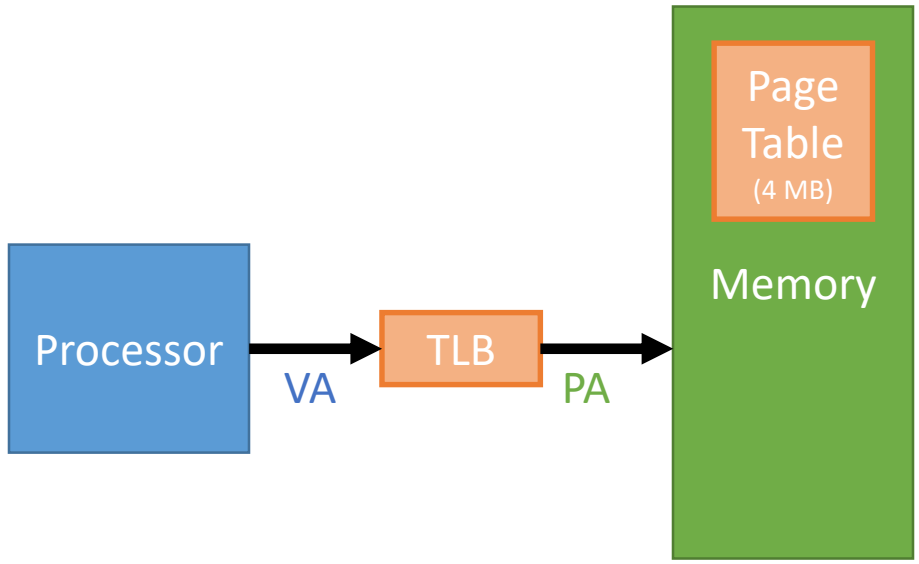
- Page is *not* in RAM [Bad]



What happens when we access memory?

- Page is in RAM [Good]
  - Page Table Entry in the TLB
    - Best performance
  - Page Table Entry *not* in the TLB
    - Poor performance
- Page is *not* in RAM [Bad]

~ 1 cycle to translate, then can go to RAM

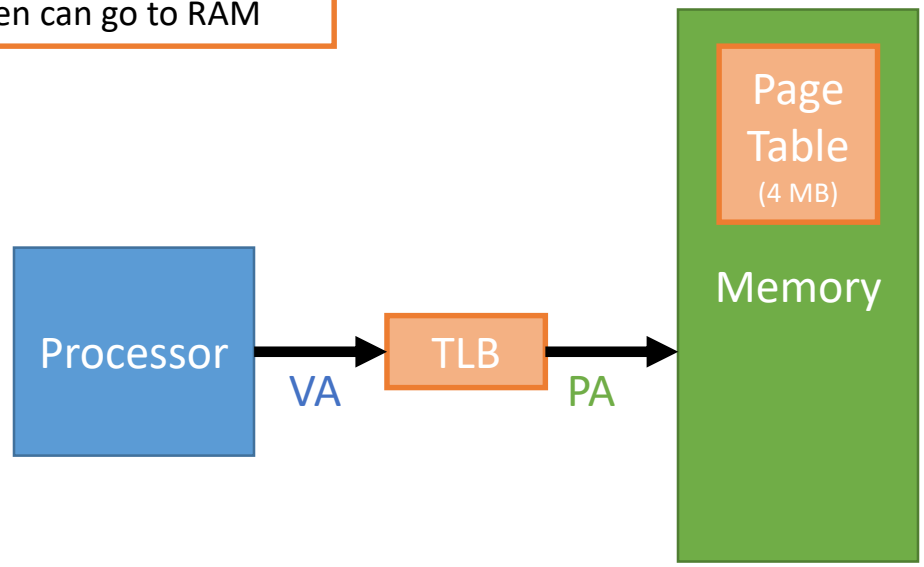


What happens when we access memory?

- Page is in RAM [Good]
  - Page Table Entry in the TLB
    - Best performance
  - Page Table Entry *not* in the TLB
    - Poor performance
- Page is *not* in RAM [Bad]

~ 1 cycle to translate, then can go to RAM

~1,000 cycles to load PTE to TLB, then can go to RAM

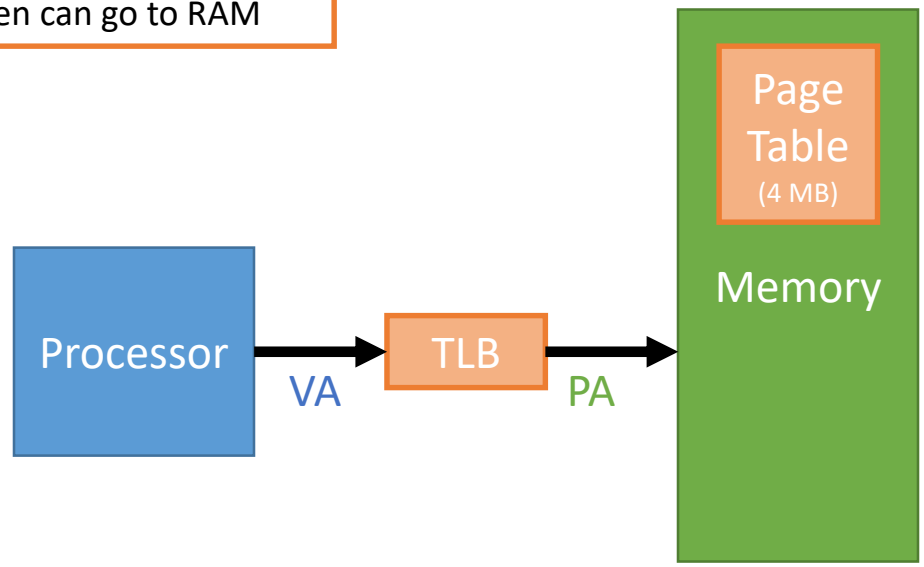


What happens when we access memory?

- Page is in RAM [Good]
  - Page Table Entry in the TLB
    - Best performance
  - Page Table Entry *not* in the TLB
    - Poor performance
- Page is *not* in RAM [Bad]
  - Page Table Entry in the TLB
    - Very poor performance

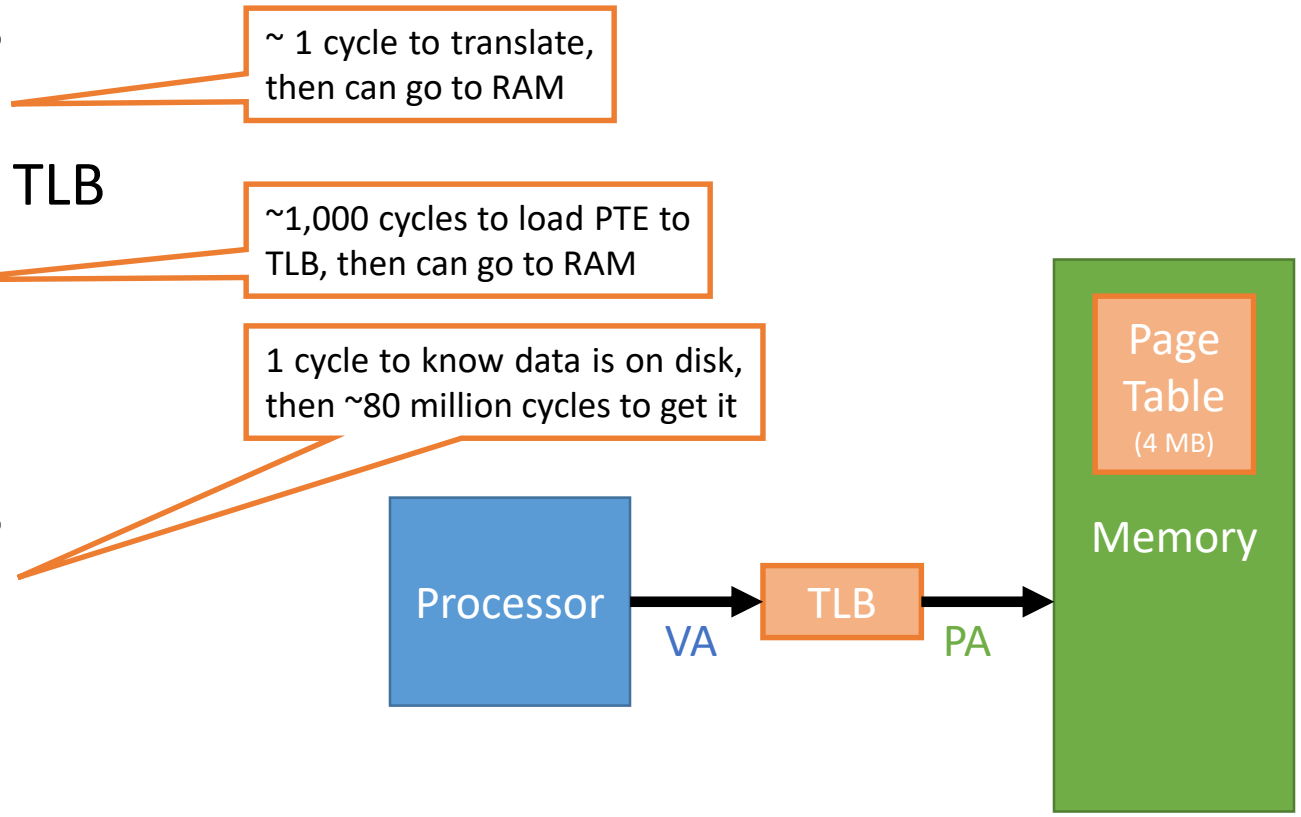
~ 1 cycle to translate, then can go to RAM

~1,000 cycles to load PTE to TLB, then can go to RAM



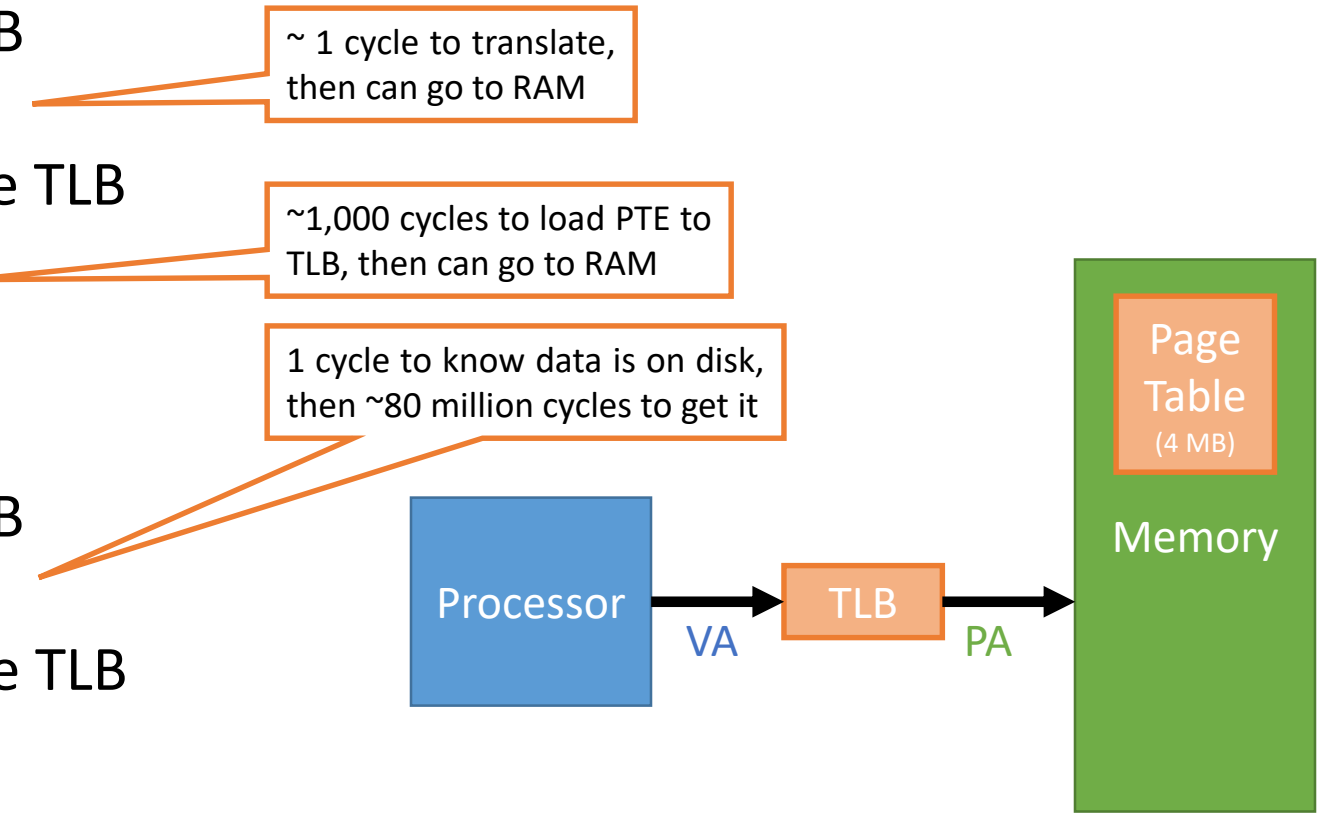
What happens when we access memory?

- Page is in RAM [Good]
  - Page Table Entry in the TLB
    - Best performance
  - Page Table Entry *not* in the TLB
    - Poor performance
- Page is *not* in RAM [Bad]
  - Page Table Entry in the TLB
    - Very poor performance



What happens when we access memory?

- Page is in RAM [Good]
  - Page Table Entry in the TLB
    - Best performance
  - Page Table Entry *not* in the TLB
    - Poor performance
- Page is *not* in RAM [Bad]
  - Page Table Entry in the TLB
    - Very poor performance
  - Page Table Entry not in the TLB
    - Worst performance



What happens when we access memory?

- Page is in RAM [Good]
  - Page Table Entry in the TLB
    - Best performance
  - Page Table Entry *not* in the TLB
    - Poor performance
- Page is *not* in RAM [Bad]
  - Page Table Entry in the TLB
    - Very poor performance
  - Page Table Entry not in the TLB
    - Worst performance

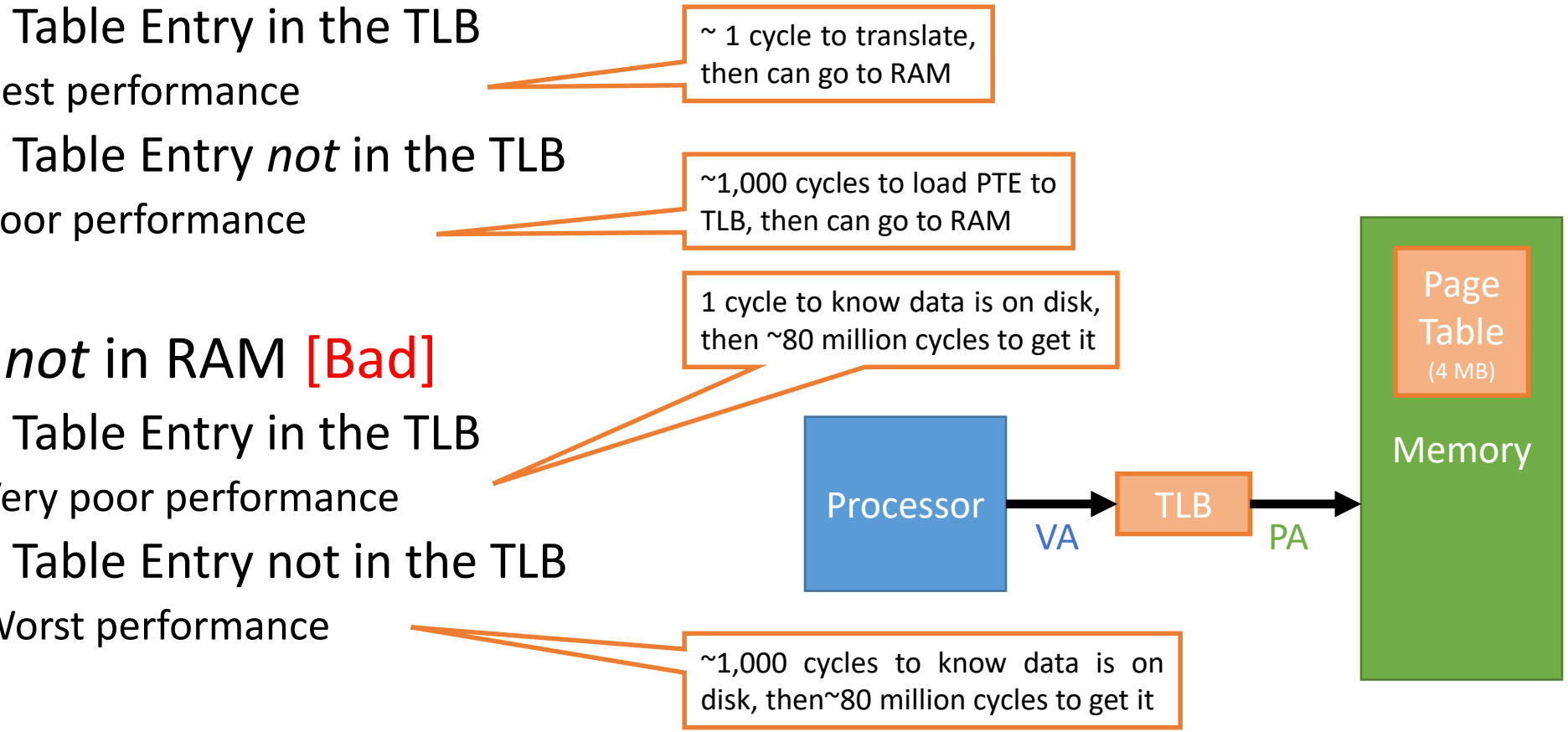


Illustration from the textbook

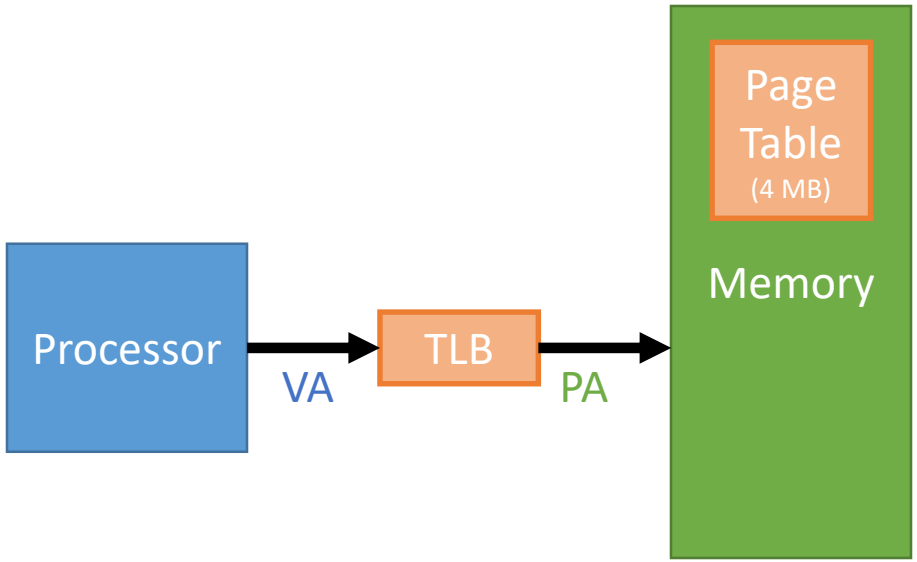
TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

FIGURE 5.32 The possible combinations of events in the TLB, virtual memory system, and cache. Three of these combinations are impossible, and one is possible (TLB hit, virtual memory hit, cache miss) but never detected.

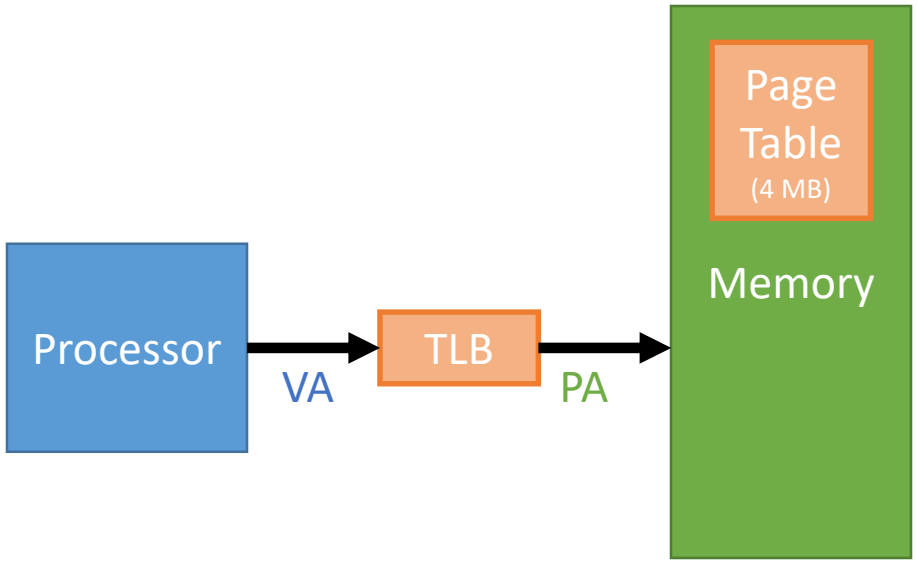
How do we make the TLB *seem* larger?

Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size



How do we make the TLB *seem* larger?



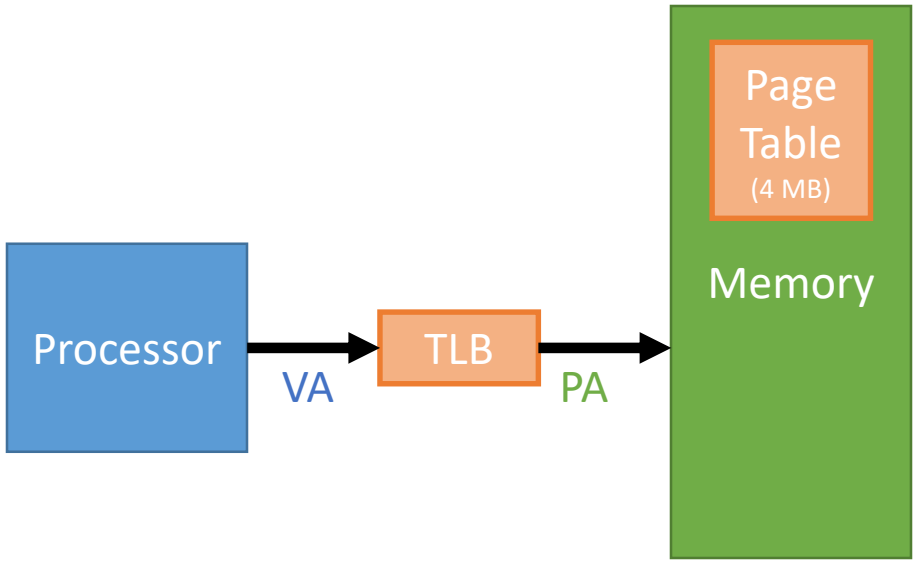
Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

II. Increase page size

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

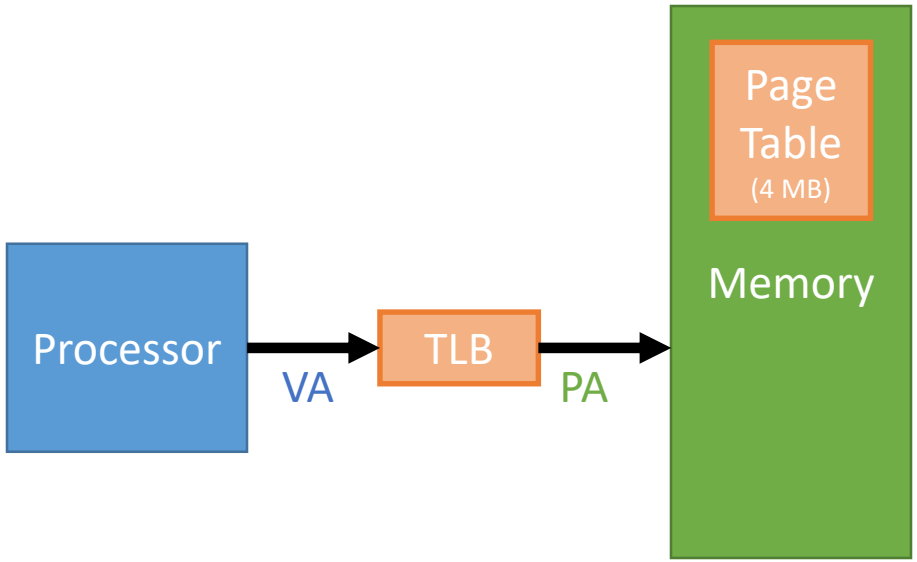
I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

II. Increase page size

64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

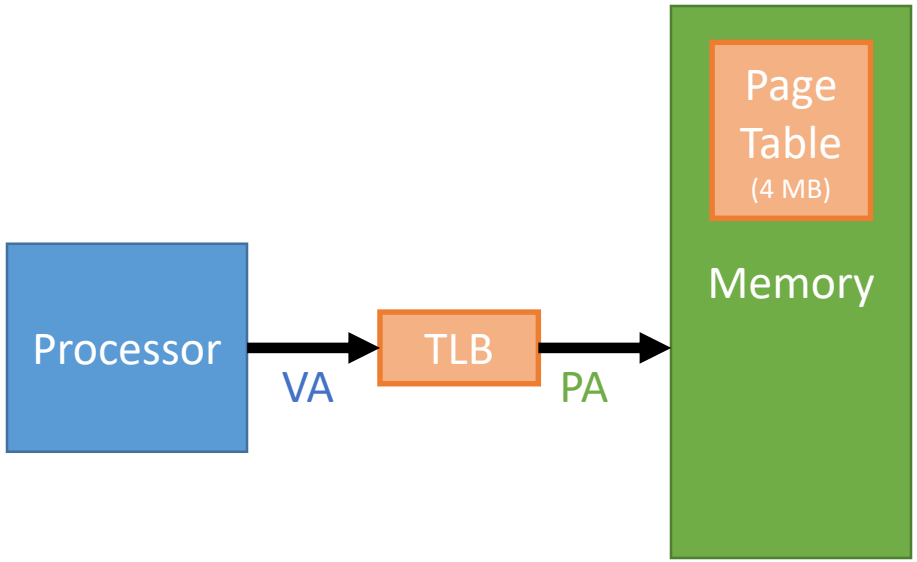
A:

II. Increase page size

64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

We can reduce the number of TLB misses by using larger page tables.
We can address more memory with the same number of PTEs.

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

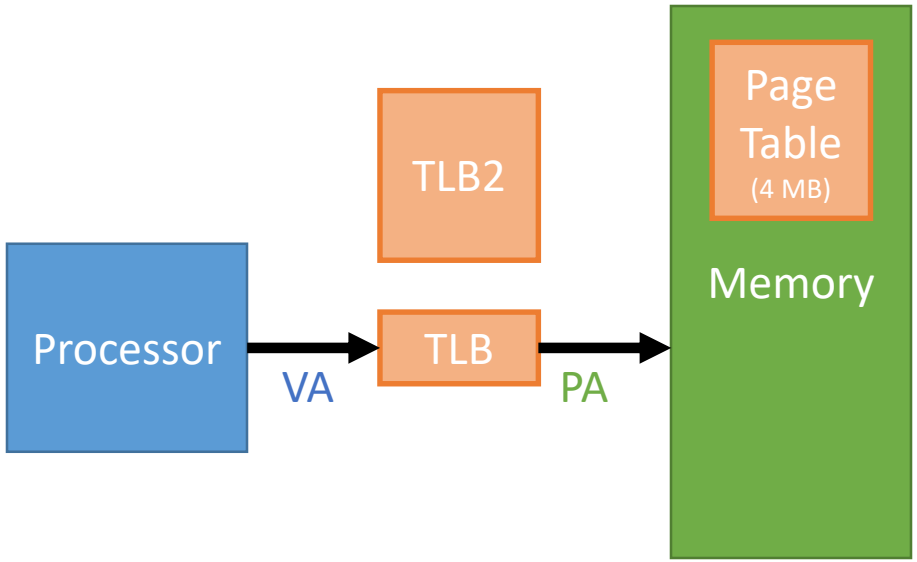
II. Increase page size

64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

We can reduce the number of TLB misses by using larger page tables.
We can address more memory with the same number of PTEs.

III. Add another TLB level

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

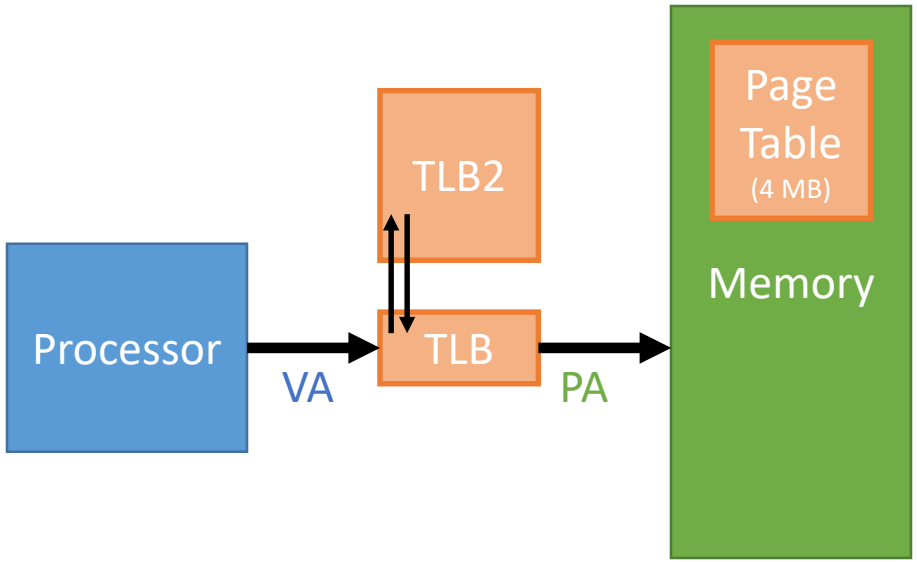
II. Increase page size

64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

We can reduce the number of TLB misses by using larger page tables.
We can address more memory with the same number of PTEs.

III. Add another TLB level

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

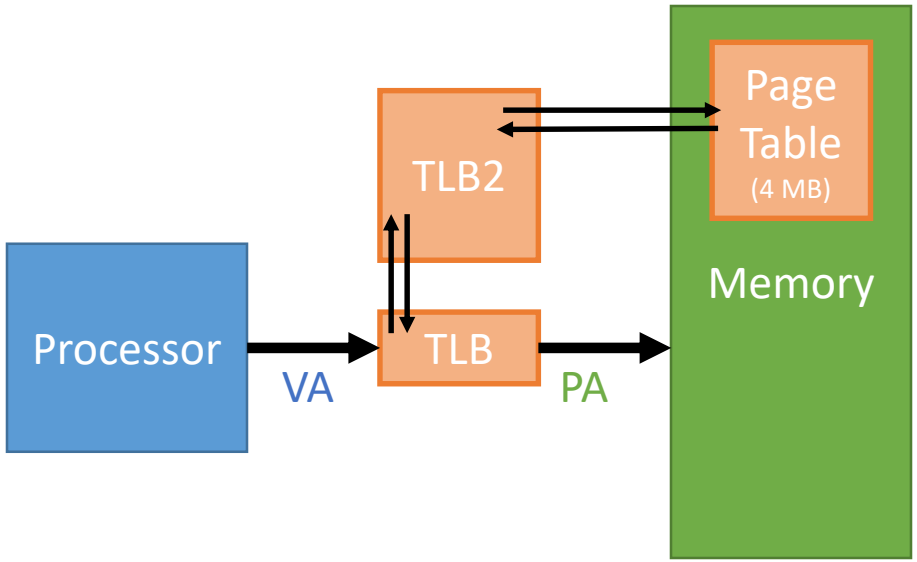
II. Increase page size

64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

We can reduce the number of TLB misses by using larger page tables.
We can address more memory with the same number of PTEs.

III. Add another TLB level

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

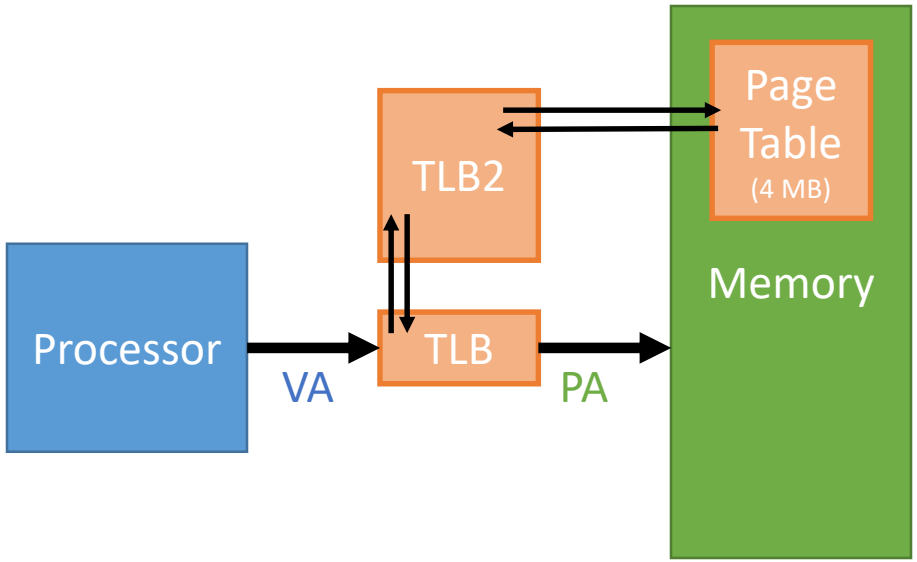
II. Increase page size

64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

We can reduce the number of TLB misses by using larger page tables.
We can address more memory with the same number of PTEs.

III. Add another TLB level

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

II. Increase page size

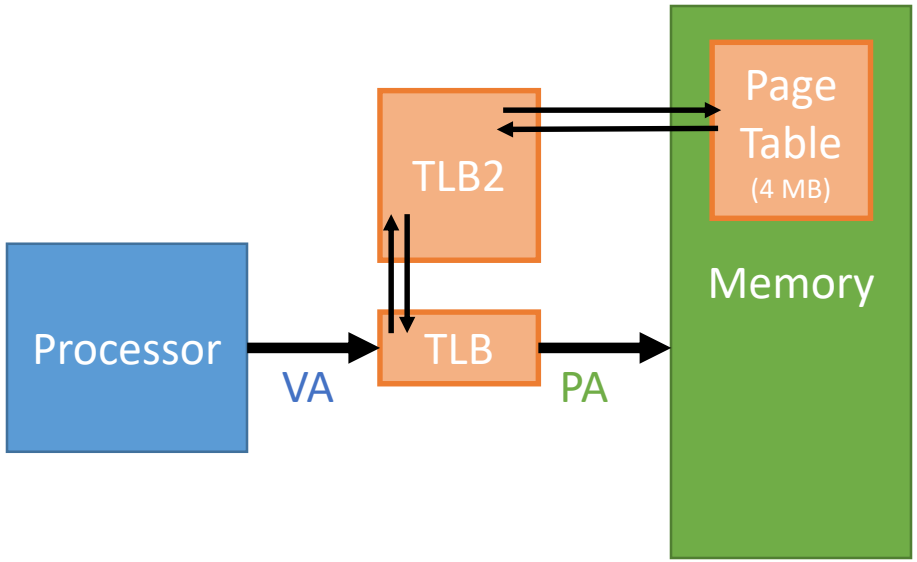
64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

We can reduce the number of TLB misses by using larger page tables.
We can address more memory with the same number of PTEs.

III. Add another TLB level

Add 2nd-level TLB that is larger, but slower.

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

II. Increase page size

64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

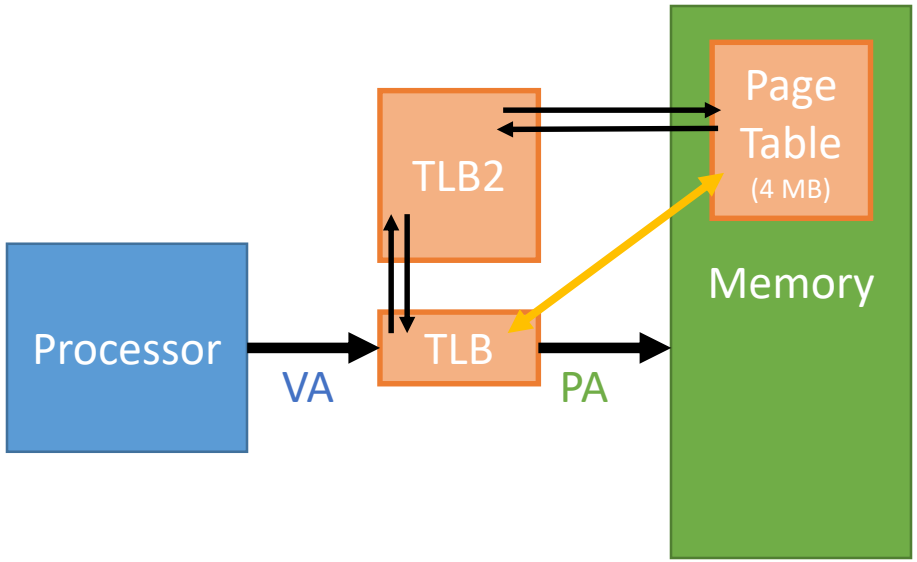
We can reduce the number of TLB misses by using larger page tables.
We can address more memory with the same number of PTEs.

III. Add another TLB level

Add 2nd-level TLB that is larger, but slower.

IV. Have HW manage TLB misses.

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

II. Increase page size

64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

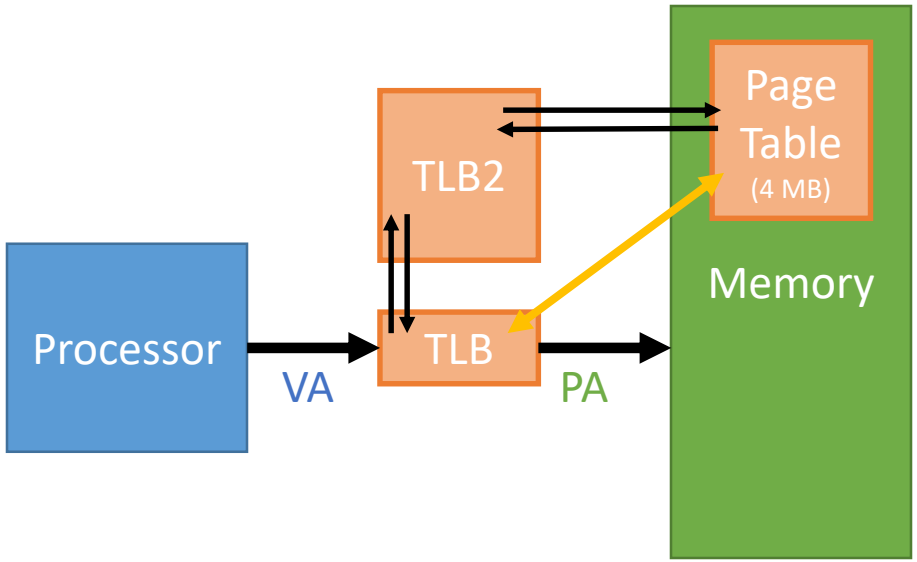
We can reduce the number of TLB misses by using larger page tables.
We can address more memory with the same number of PTEs.

III. Add another TLB level

Add 2nd-level TLB that is larger, but slower.

IV. Have HW manage TLB misses.

How do we make the TLB *seem* larger?



Q: How can we make the TLB appear larger without reducing performance?

I. Store more PTEs in the TLB
II. Increase page size
III. Add another TLB level
IV. Have HW manage TLB misses, not O/S
V. Decrease page size

A:

II. Increase page size

64 4kB pages = 256kB of data
32 2MB pages = 64 MB of data

We can reduce the number of TLB misses by using larger page tables.
We can address more memory with the same number of PTEs.

III. Add another TLB level

Add 2nd-level TLB that is larger, but slower.

IV. Have HW manage TLB misses.

Hardware can do a page table walk to replace a page in the TLB.

Quincy Flint

Quincy Flint

References

- David Black-Schaffer: Lecture Series on Virtual Memory
- Patterson, Hennessy: Computer Organization and Design: the Hardware/Software Interface
- Intel Hardware Data-Sheets
- Linux: Anatomy of a Program in Memory