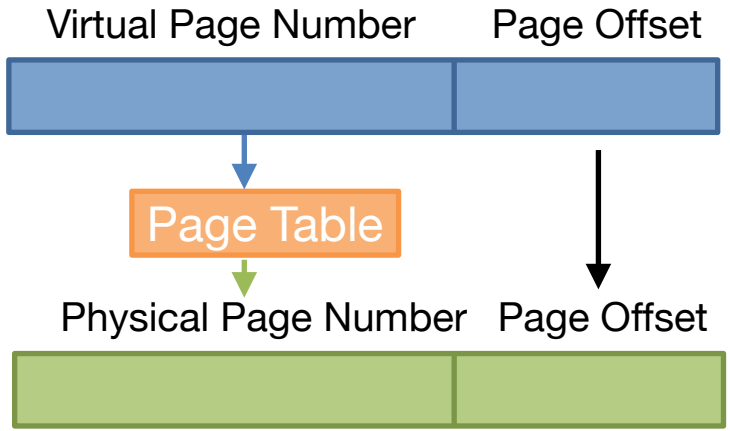
Virtual Memory

Special thanks to David Black-Schaffer

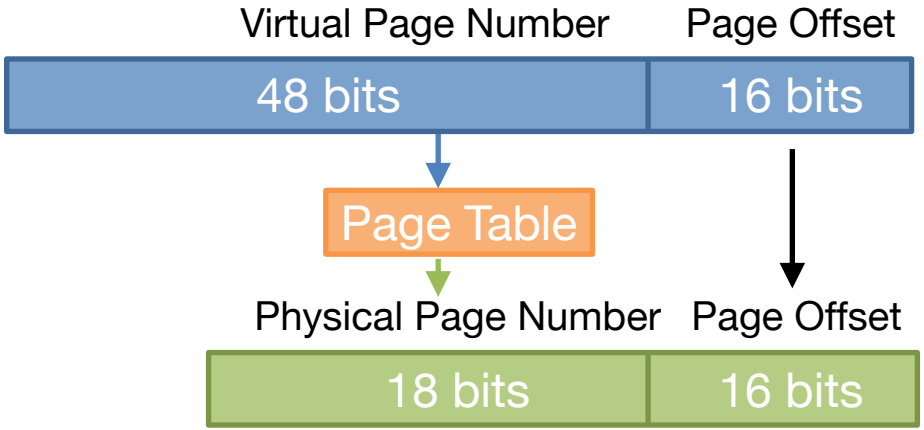
Warm-up

- What is the virtual and physical address breakdown of a 64 bit machine with 16 GB of RAM and 64KB pages?



Warm-up

- What is the virtual and physical address breakdown of a 64 bit machine with 16 GB of RAM and 64KB pages?
  - # Virtual address bits = 64
  - # Physical address bits = $\log_2(16GB) = \log_2(2^4 * 2^{30}) = 34$
  - # Page offset bits = $\log_2(16KB) = \log_2(2^6 * 2^{10}) = 16$
  - # Bits in VPN = 64 - 16 = 48
  - # Bits in PPN = 34 - 16 = 18



Warm-up

- Does the number of entries in the page table depend on the size of the virtual address space or the physical address space?
- How does the number of entries in the page table change as the size of the pages increases?
- How does the size of each entry change as the size of the pages increases?

Warm-up

- Does the number of entries in the page table depend on the size of the virtual address space or the physical address space?
  - Virtual Address Space
- How does the number of entries in the page table change as the size of the pages increases?
  - The number of entries decreases
- How does the size of each entry change as the size of the pages increases?
  - The size of each entry decreases

Warm-up

- How many virtual pages and physical pages would I have in a 32-bit system where I have a page size of 8KB and 8GB RAM?

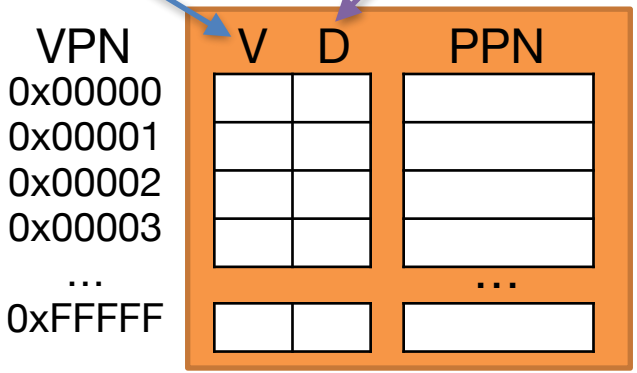
Warm-up

- How many virtual pages and physical pages would I have in a 32-bit system where I have a page size of 8KB and 8GB RAM?
  - Size of virtual address space: 2^{32}
  - Number of virtual pages: 2^{32} / 8K = 2^{32} / (2^3 * 2^{10}) = 2^{19}
  - Size of physical address space: 8G = (2^3 * 2^{30}) = 2^{33}
  - Number of physical pages: 2^{33} / 8K = 2^{33} / (2^3 * 2^{10}) = 2^{20}

Additional Page Table Metadata

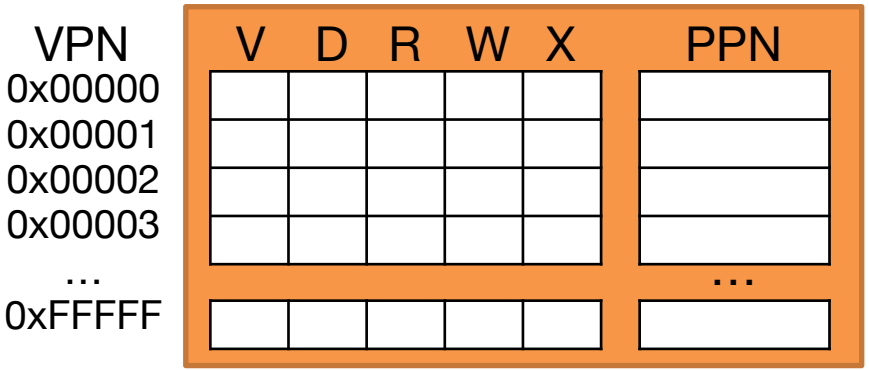
Valid
1 = page is in RAM and mapping is valid
0 = page is not in RAM

Dirty
1 = page on RAM is more up to date than page on disk
0 = page in RAM matches page on disk



Page Protection Bits

- We can specify how a process can use each page:
  - read, write, execute
- If you don't have permission to do what you are trying, then you get a memory protection fault or segmentation fault (you crash)



Memory Accesses

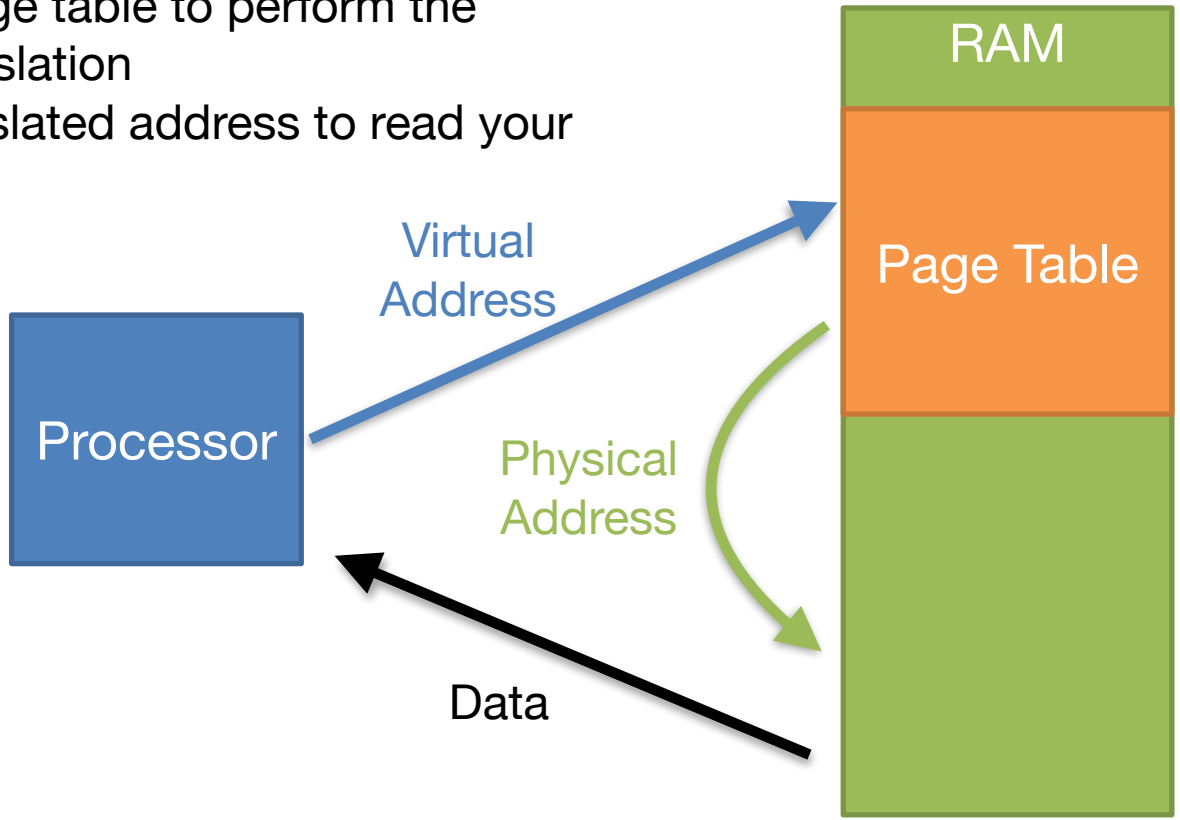
- Q: With virtual memory, how many memory accesses do you have to make each time you access a piece of data?

Memory Accesses

- Q: With virtual memory, how many memory accesses do you have to make each time you access a piece of data?
  - 2 accesses

Memory Accesses

1. Read the page table to perform the address translation
2. Use the translated address to read your data



Memory Accesses

- With virtual memory, how many memory accesses do you have to make each time you access a piece of data?
  - 2 accesses
- Q: How can we make this faster?

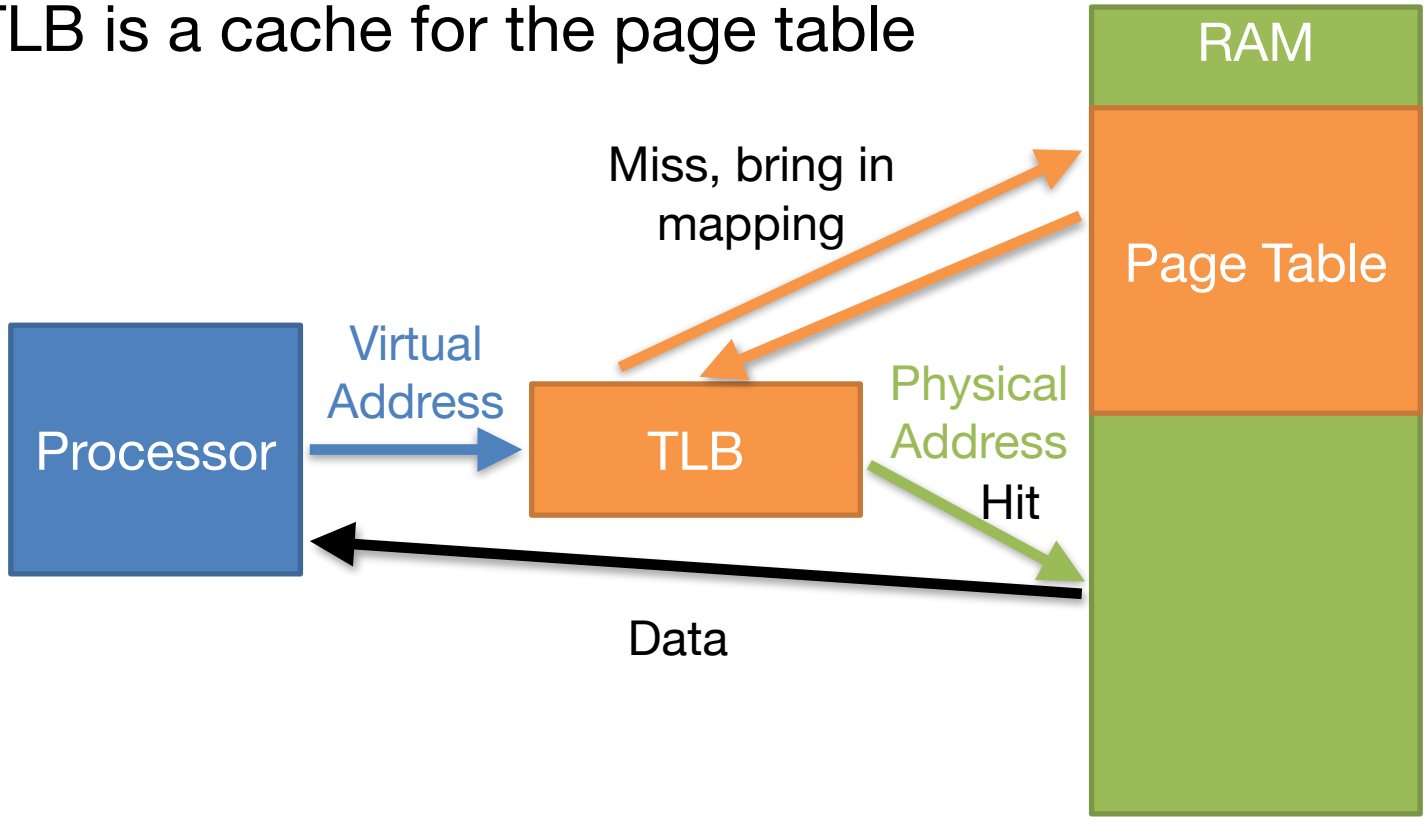
Memory Accesses

- With virtual memory, how many memory accesses do you have to make each time you access a piece of data?
  - 2 accesses
- Q: How can we make this faster?
  - With a cache!

Translation Lookaside Buffer

Translation Lookaside Buffer (TLB)

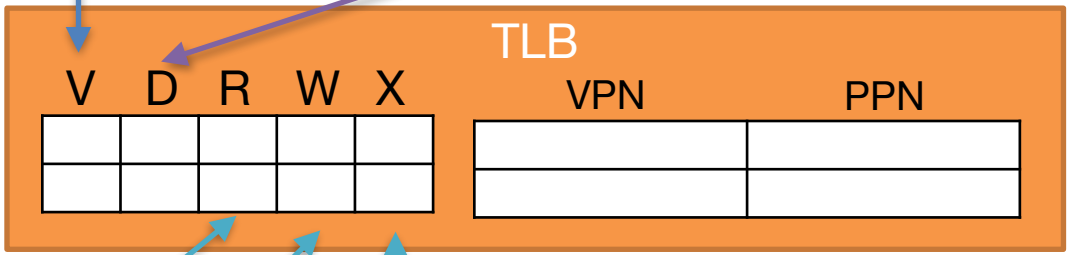
- The TLB is a cache for the page table



Translation Lookaside Buffer (TLB)

Valid
1 = page is in RAM and mapping is valid
0 = page is not in RAM

Dirty
1 = page on RAM is more up to date than page on disk
0 = page in RAM matches page on disk



Read, Write and Execute permissions

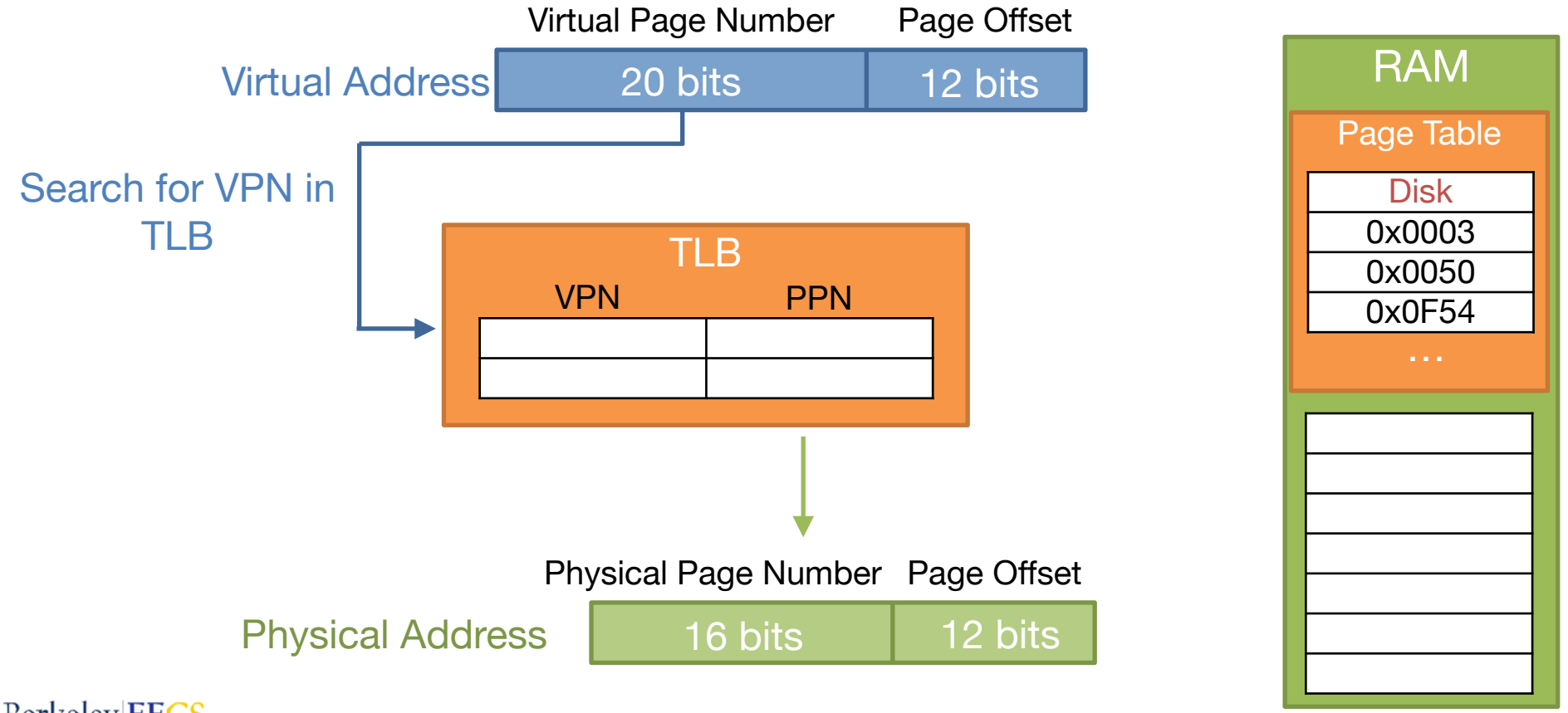
Translation Lookaside Buffer (TLB)

- To be fast, TLBs must be small
- Usually Fully Associative
- Typically 32-128 entries
- Each entry maps to a large page
  - Takes advantage of spatial and temporal locality
- Random or FIFO replacement policy

TLB Flush

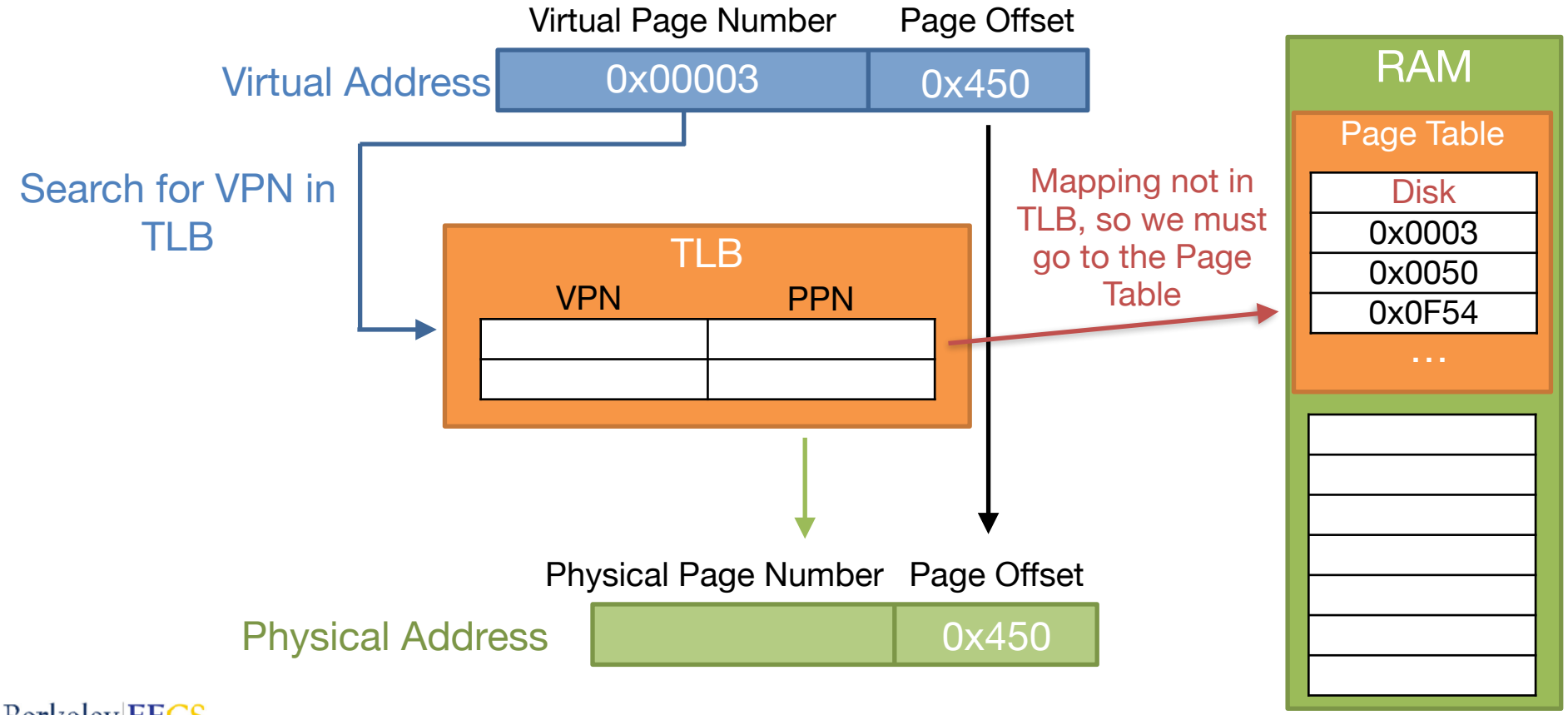
- Context switch - changing which thread is executing
- The entries in the TLB correspond to the currently active process
- On a context switch, the TLB is flushed (all entries are invalidated)

TLB Example


Virtual Page Number
Page Offset
Virtual Address
20 bits
12 bits
Search for VPN in TLB
TLB
VPN
PPN
Physical Page Number
Page Offset
Physical Address
16 bits
12 bits
RAM
Page Table
Disk
0x0003
0x0050
0x0F54
...

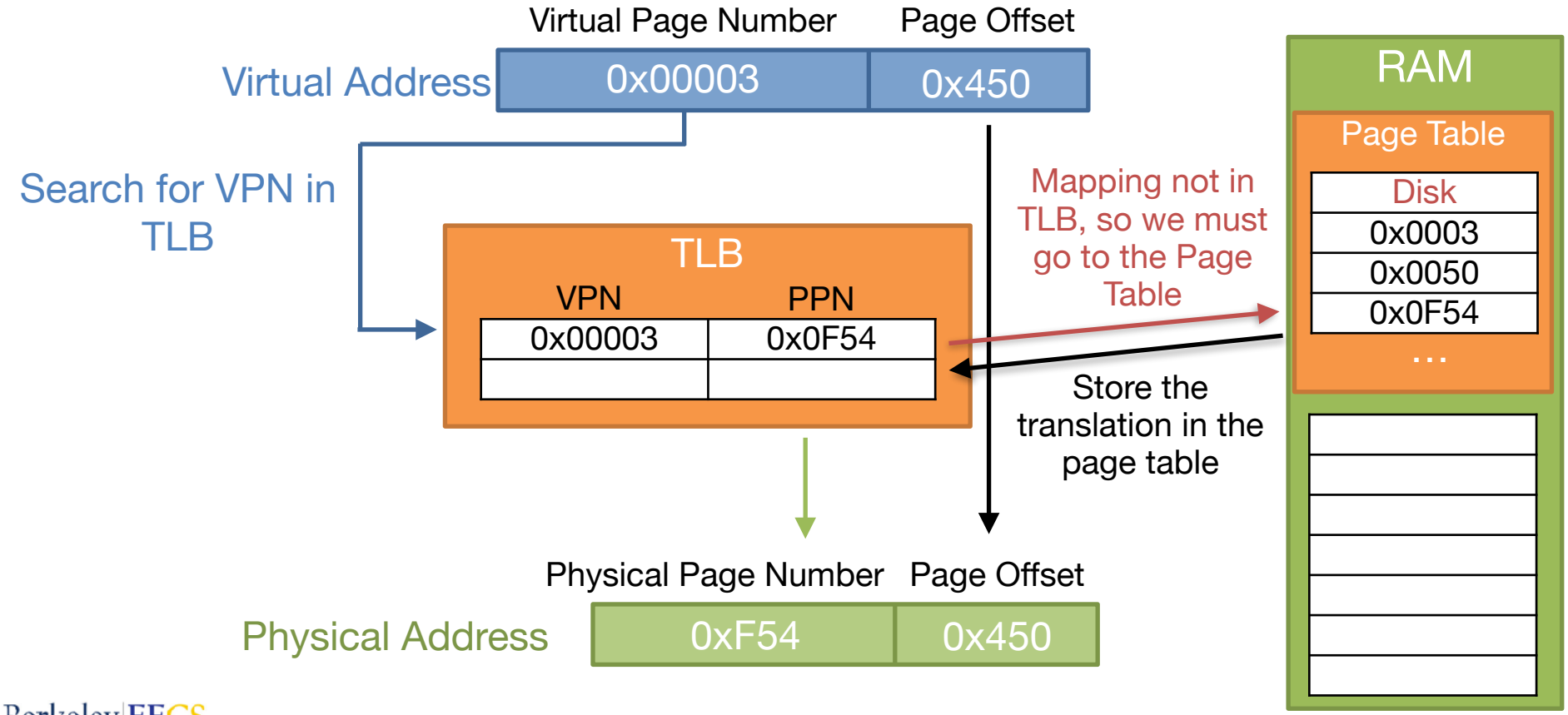
TLB Example #1

0x00003450



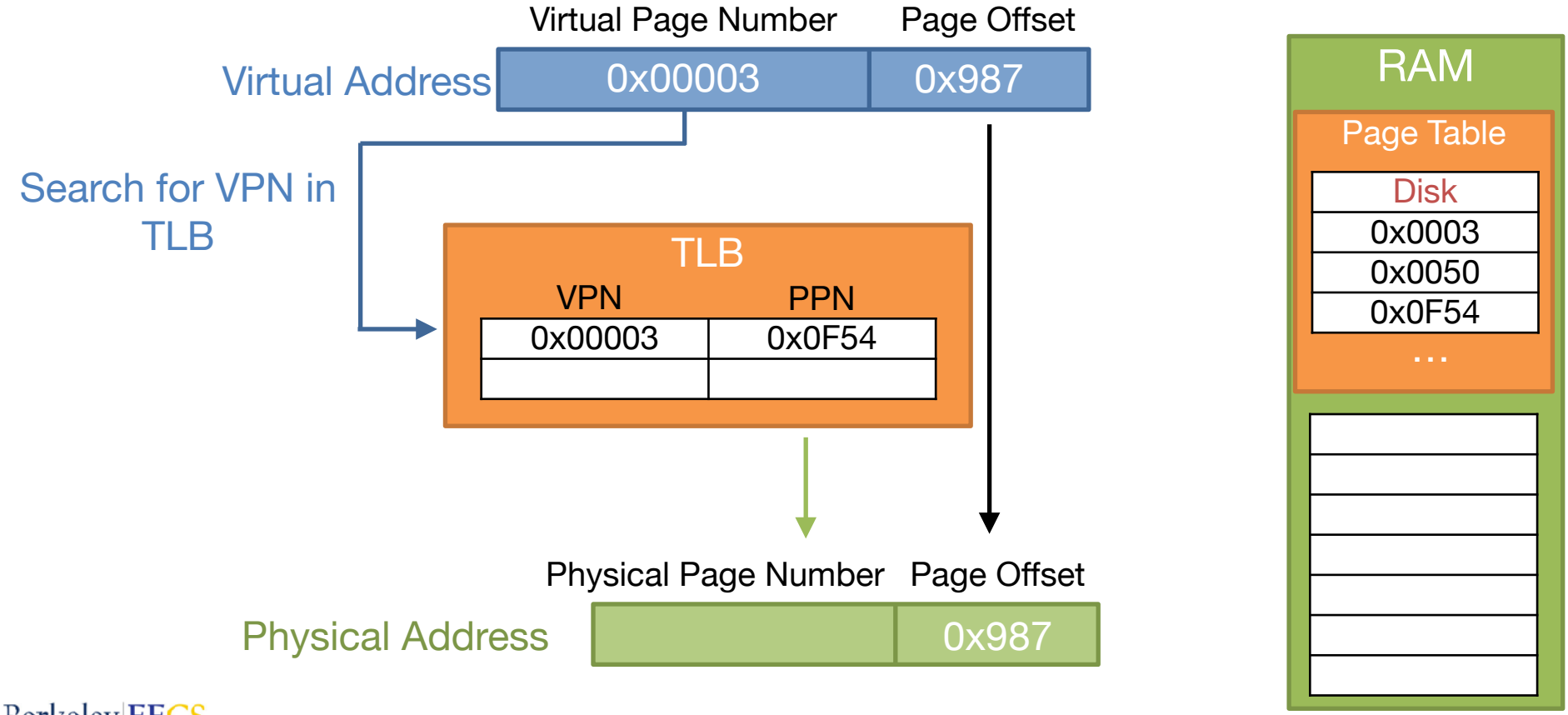
TLB Example #1

0x00003450 ⟶ 0x0F54450



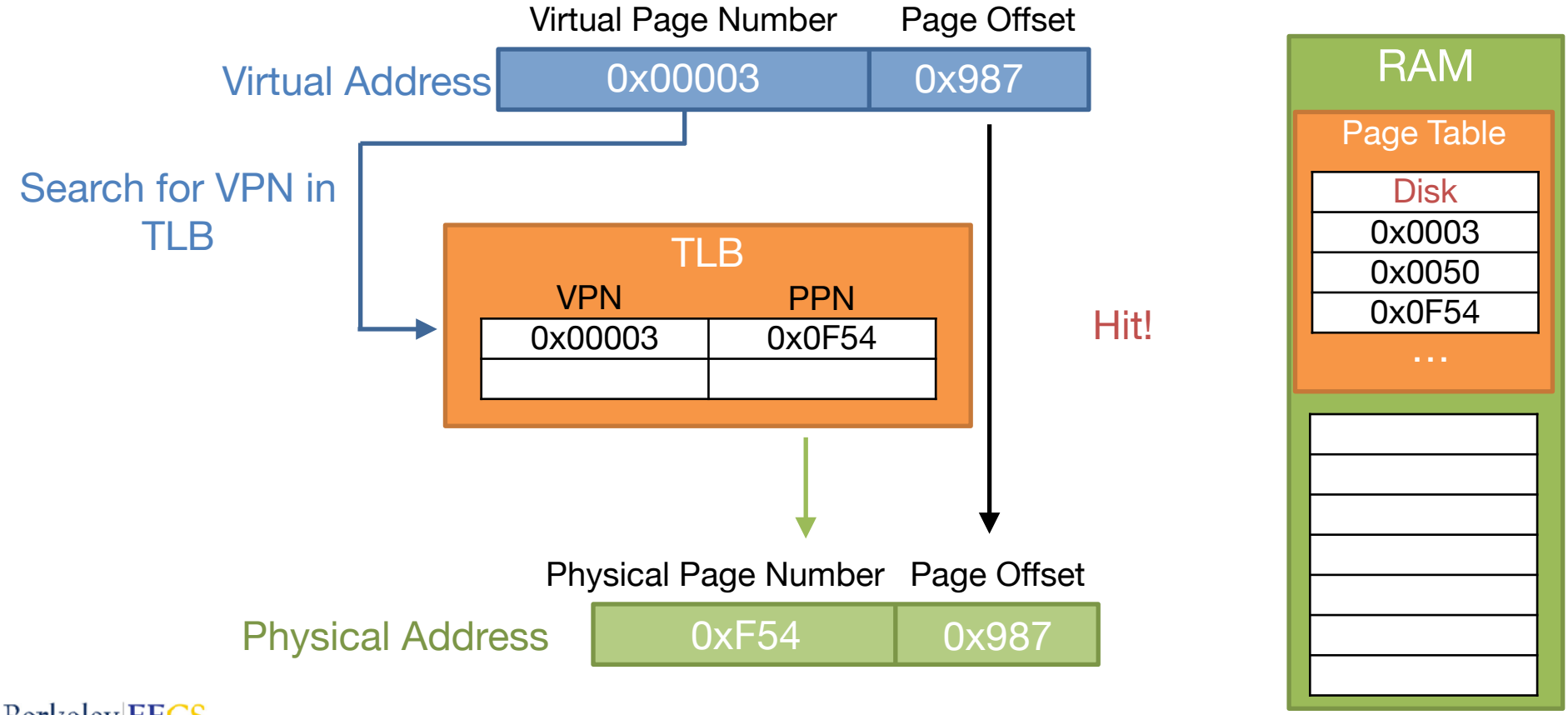
TLB Example #2

0x00003987



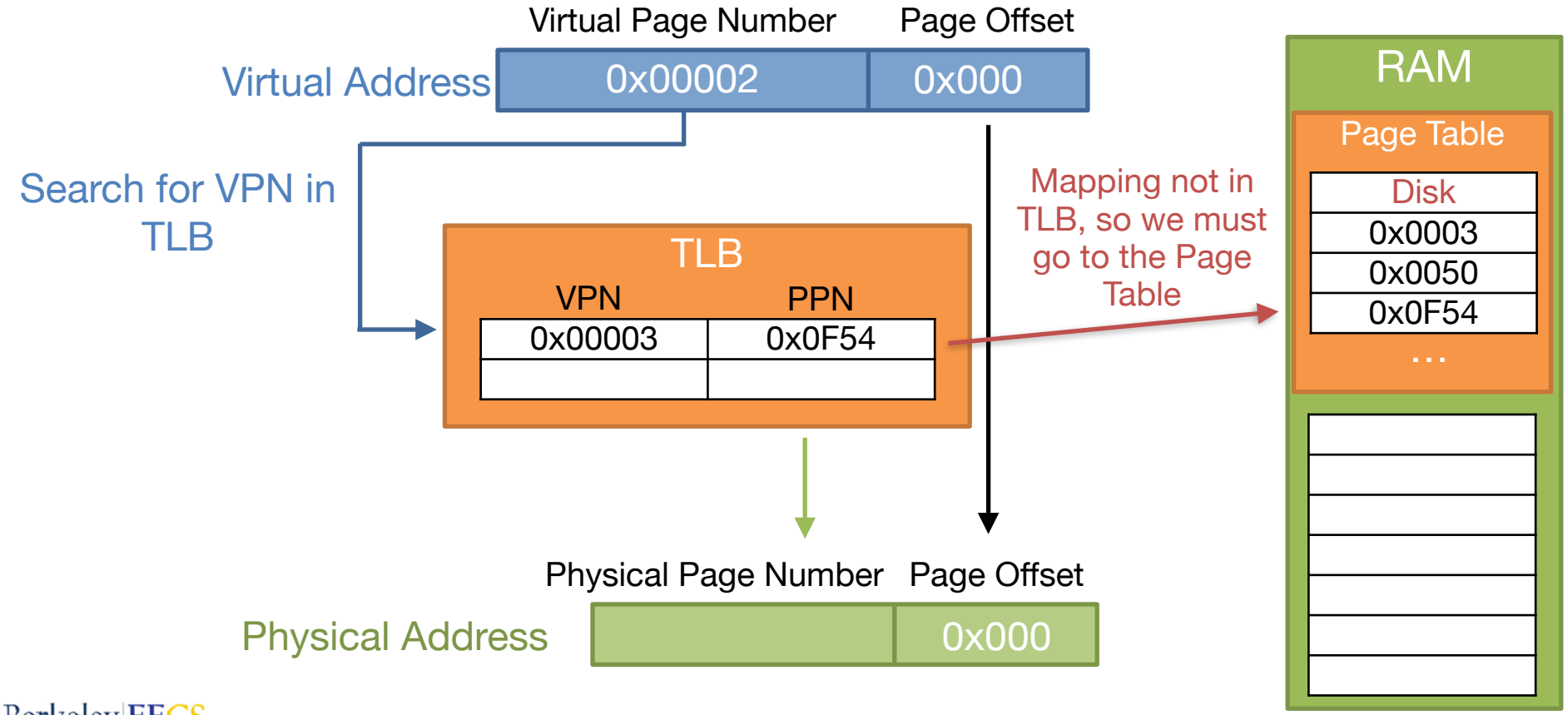
TLB Example #2

0x00003987 ⟶ 0x0F54987



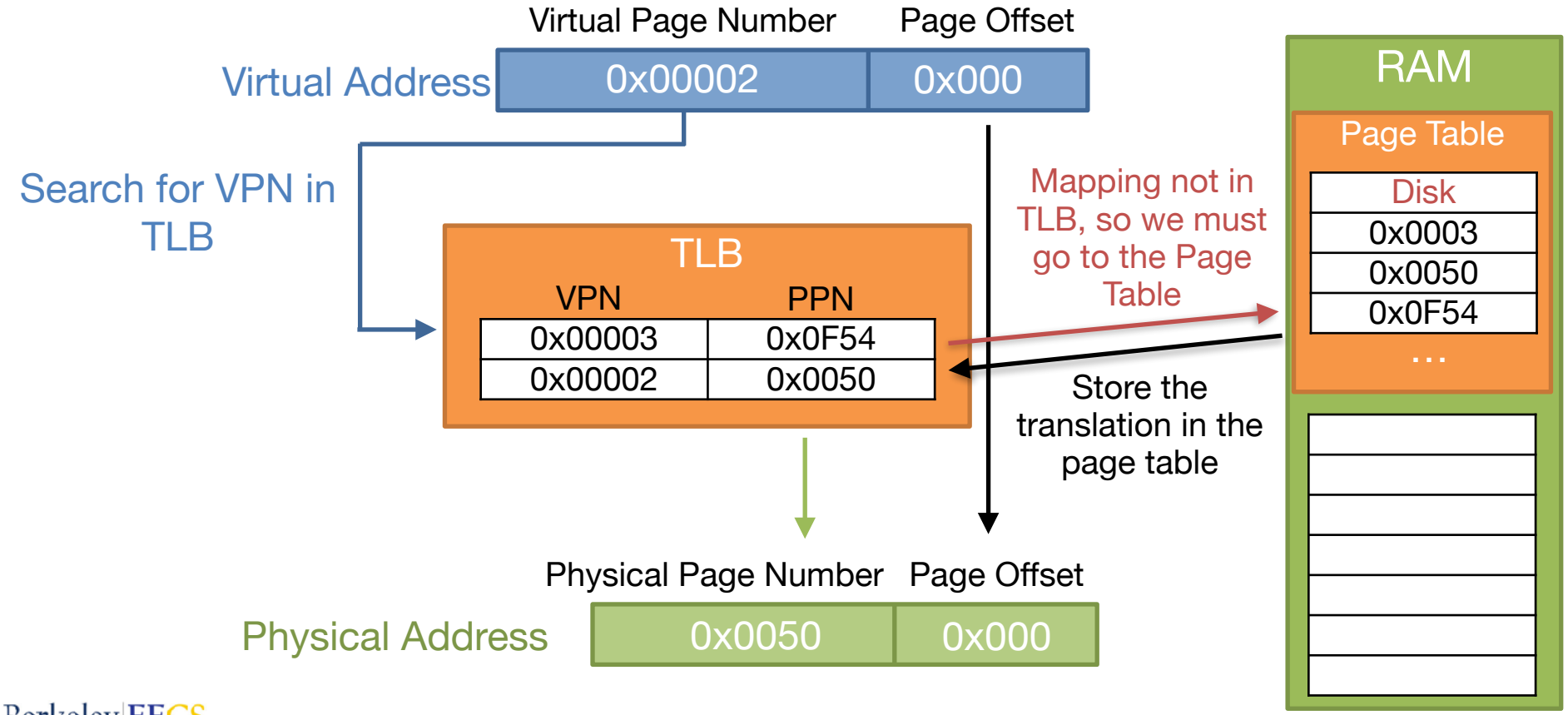
TLB Example #3

0x00002000



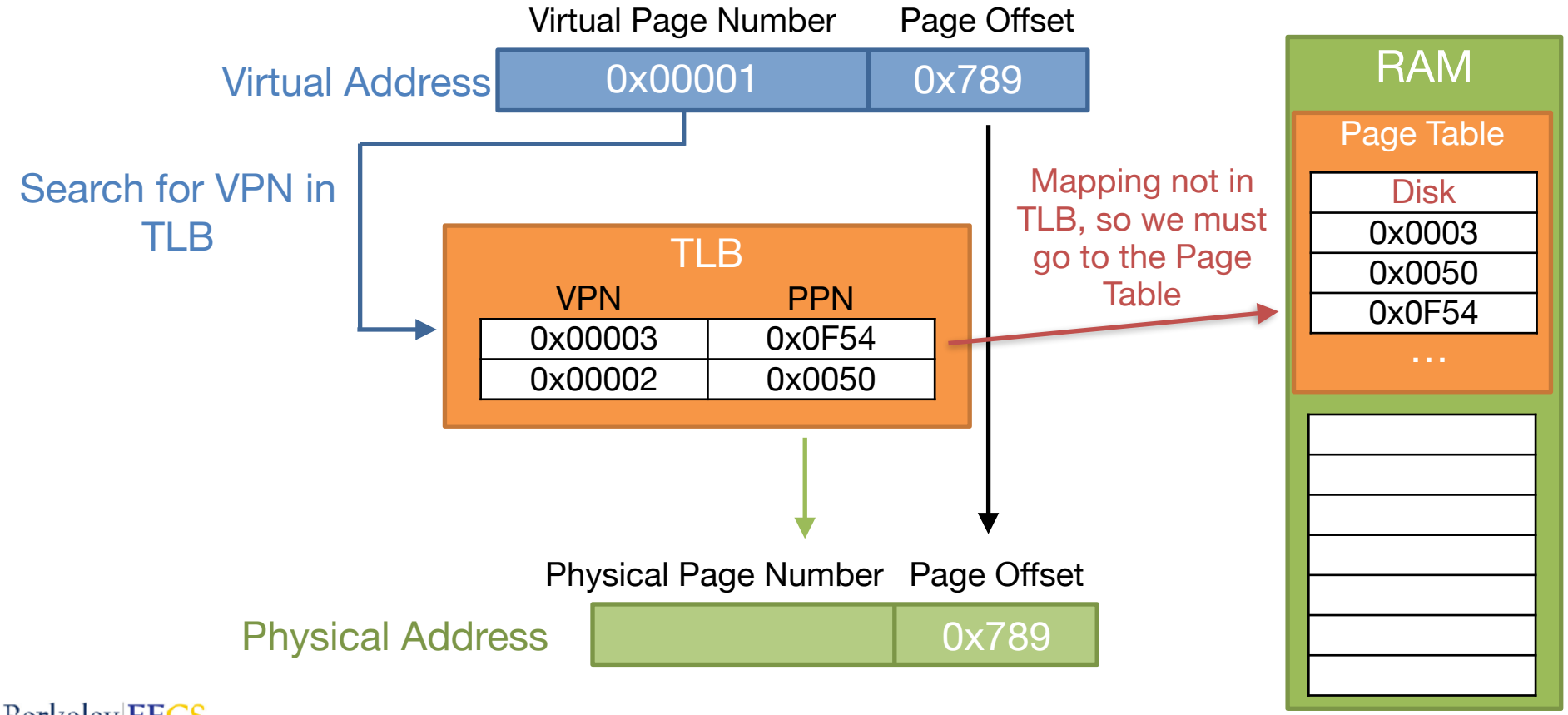
TLB Example #3

0x00002000 ⟶ 0x0050000



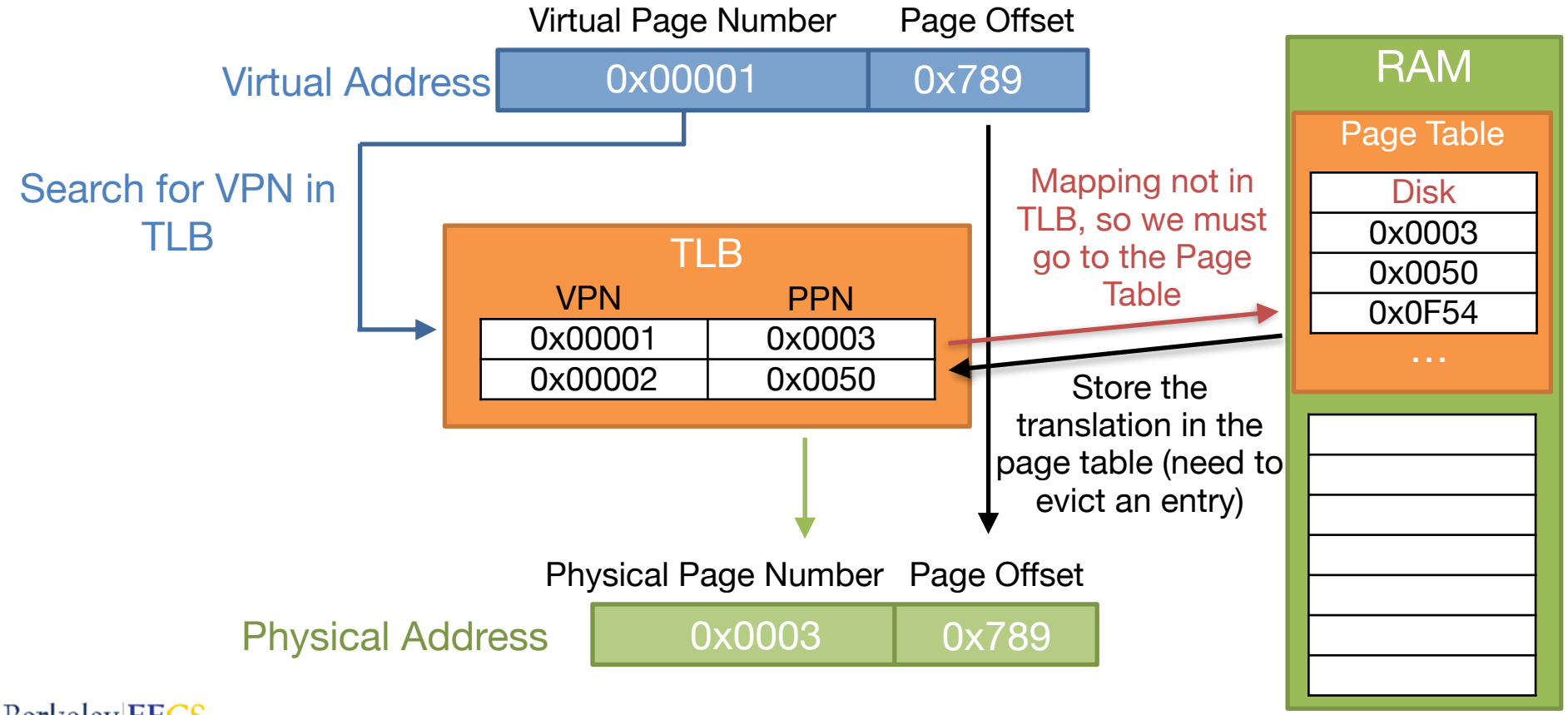
TLB Example #4

0x00001789



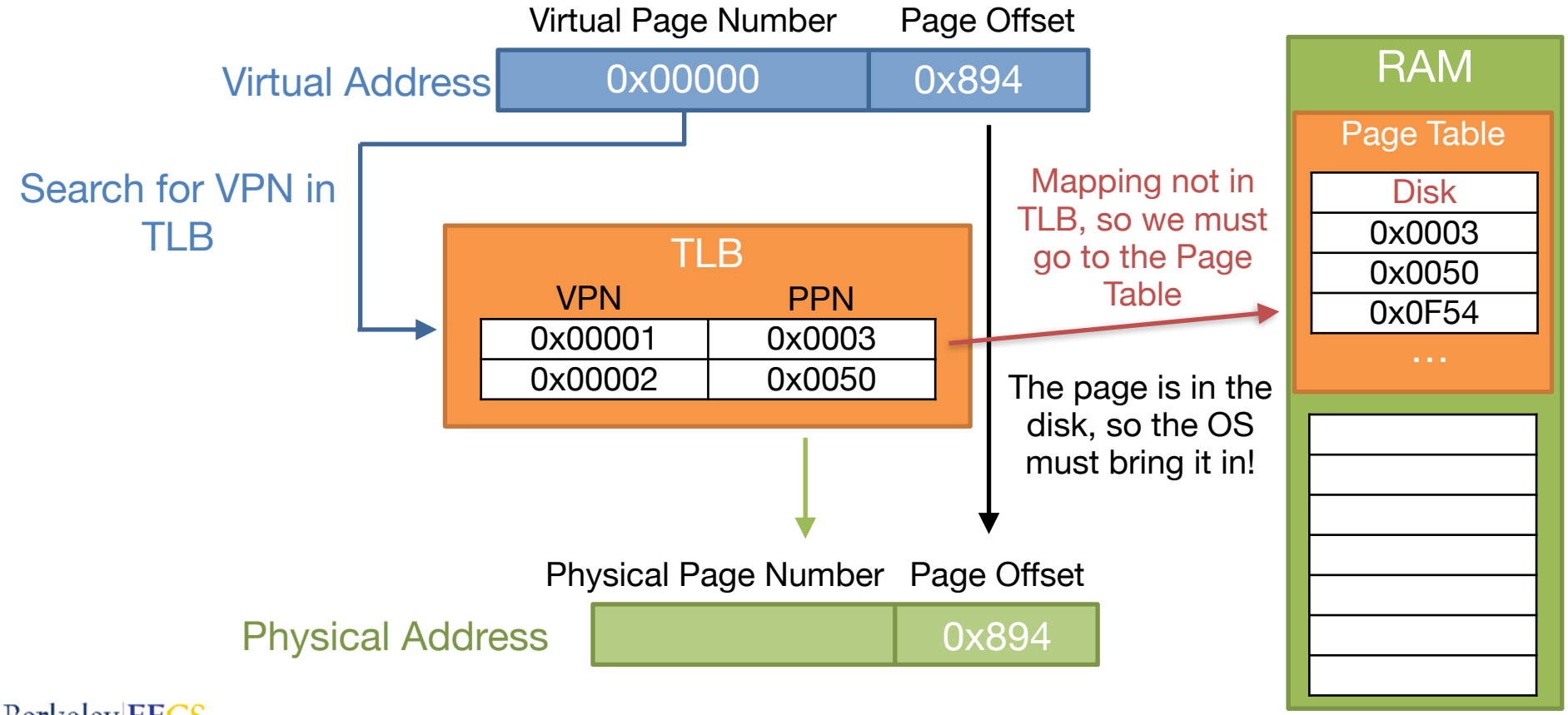
TLB Example #4

0x00001789 ⟶ 0x0003789



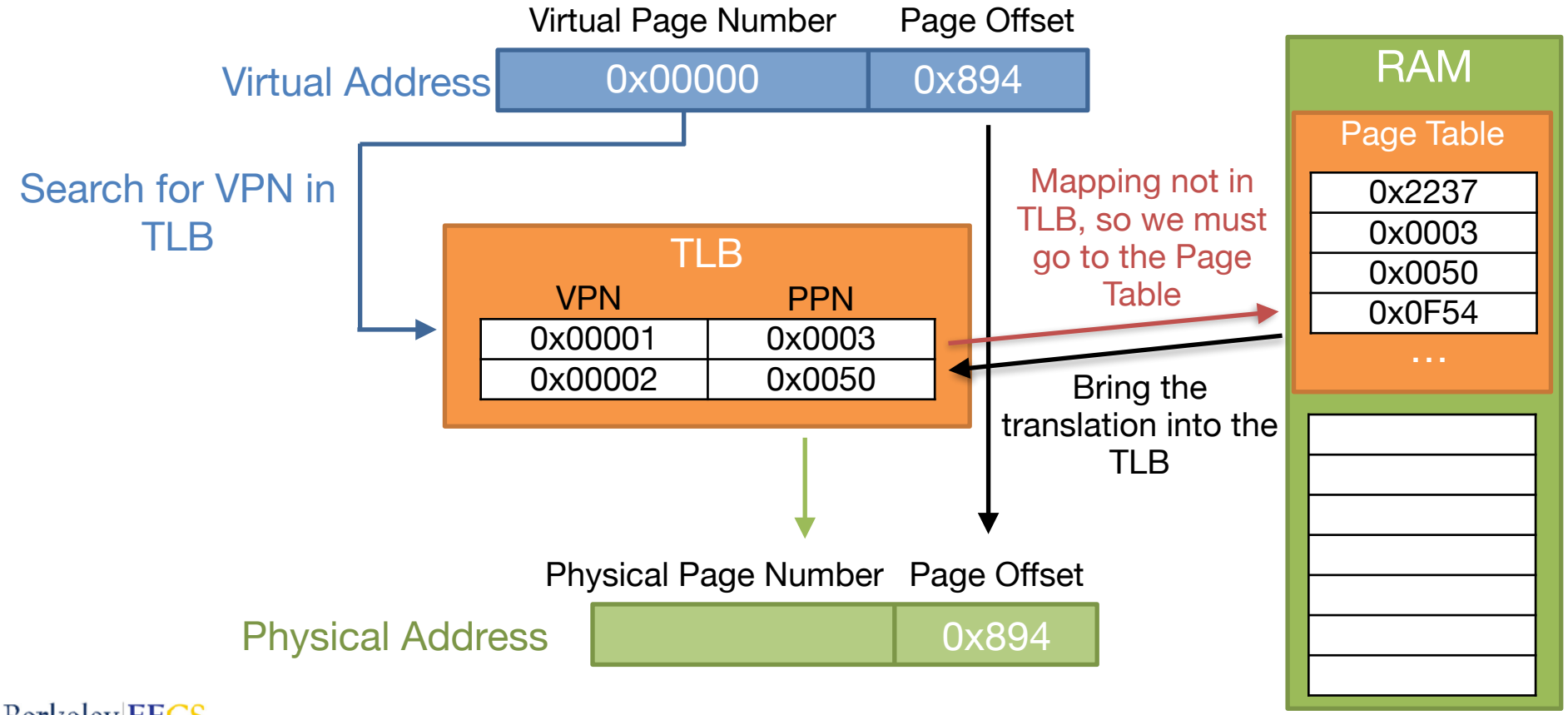
TLB Example #5

0x00000894



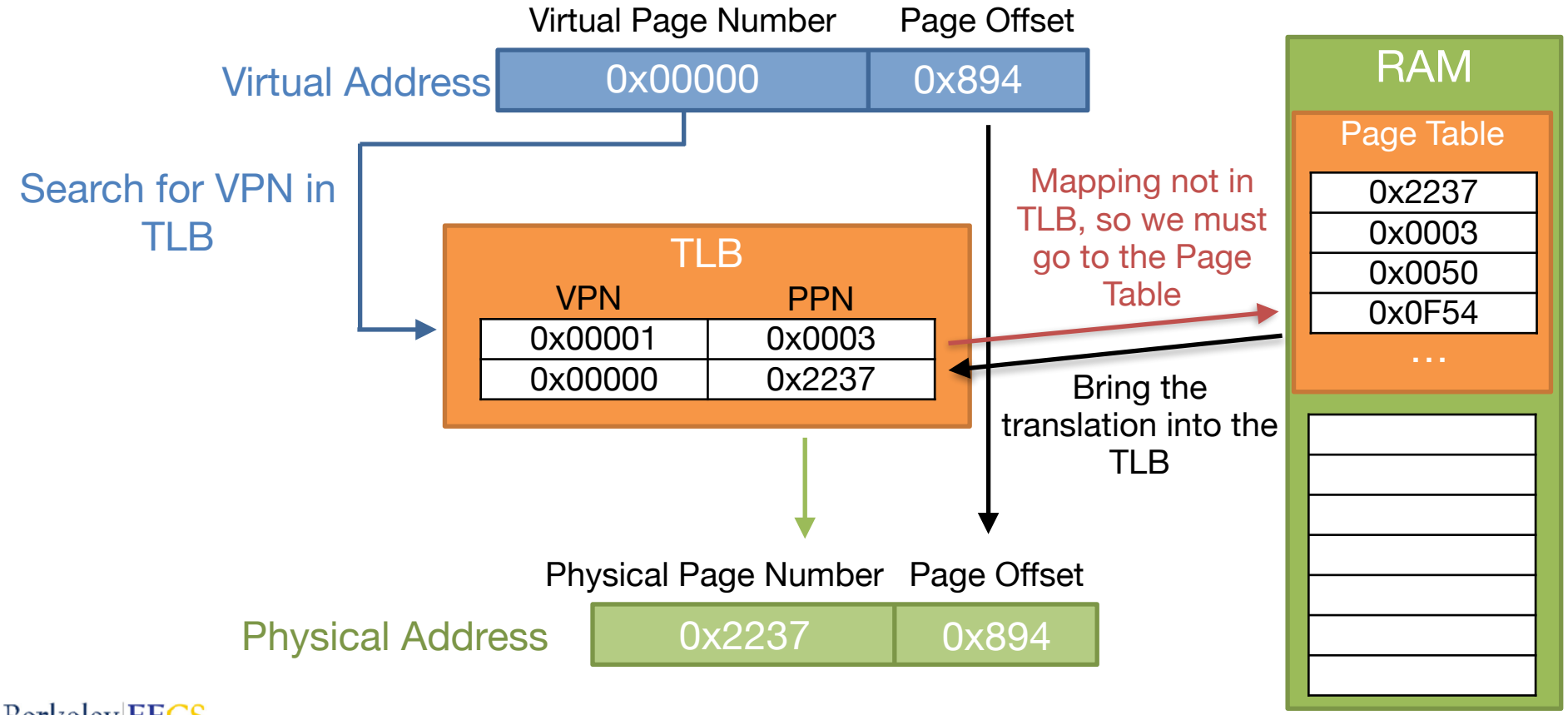
TLB Example #5

0x00000894



TLB Example #5

0x00000894 ⟶ 0x2237894



Multi-level Page Tables

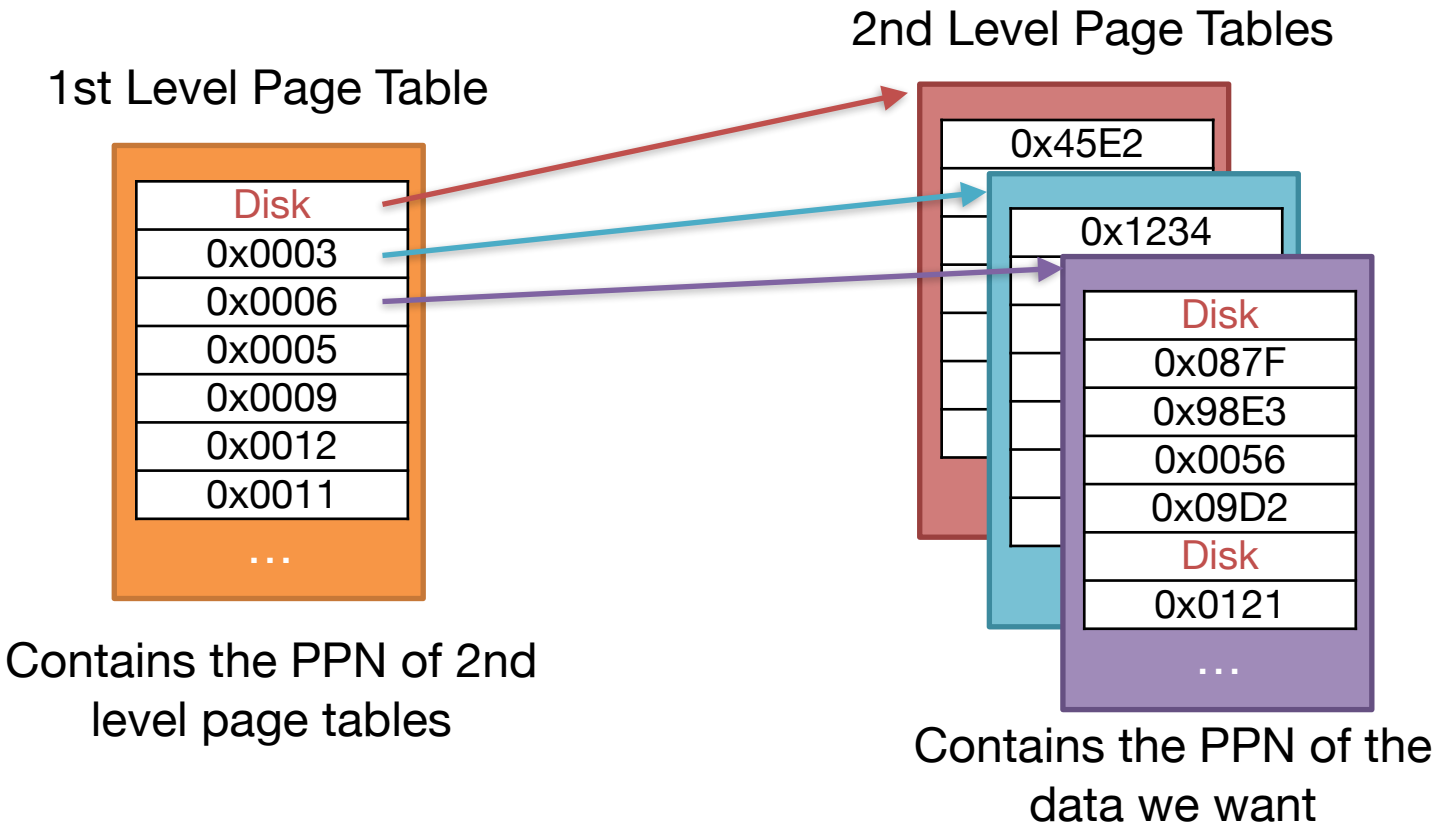
Page Table Size

- For 32-bit machine with 4GB RAM, 4kB pages we need:
  - 1M Page Table Entries (32 bits – 12 bits for page offset = 20 bits, 2^{20}=1M)
  - Each PTE is about 4 bytes (20 bits for physical page + status bits)
  - 4MB total
- Not bad…

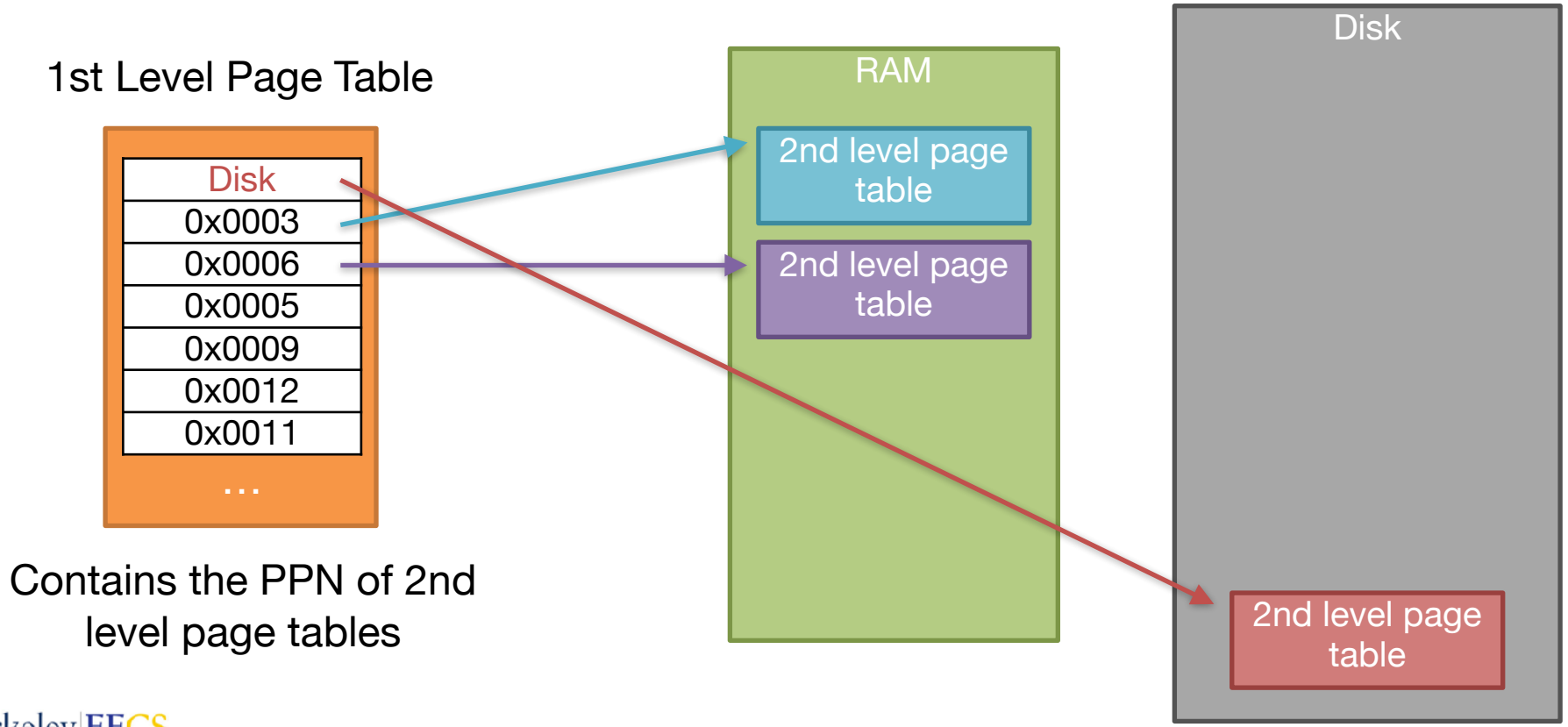
…except each program needs its own page table…

  - If we have 100 programs running, we need 400MB of Page Tables!
- And here's the tough part:
  - We can't swap the page tables out to disk
  - If the page table is not in RAM, we have no way to access it to find it!
- How can we fix this?
  - Just add more indirection…

Multi-level Page Tables



Multi-level Page Tables



Multi-level Page Tables

- The 1st level page table must ALWAYS be in RAM
- The 2nd level page tables can be paged out to disk because we can find them through the 1st level page table

Multi-level Page Table Translation

- We want a page table to be equal to the size of one page
  - In this case, its 4KB
- Q: How many entries can I fit in my 1st level page table? (Each PTE is about 4 bytes -> PPN + status bits)

Multi-level Page Table Translation

- We want a page table to be equal to the size of one page
  - In this case, its 4KB
- Q: How many entries can I fit in my 1st level page table? (Each PTE is about 4 bytes -> PPN + status bits)
  - 1024
- Q: How many bits do I need to index a page table with 1024 entries?

Multi-level Page Table Translation

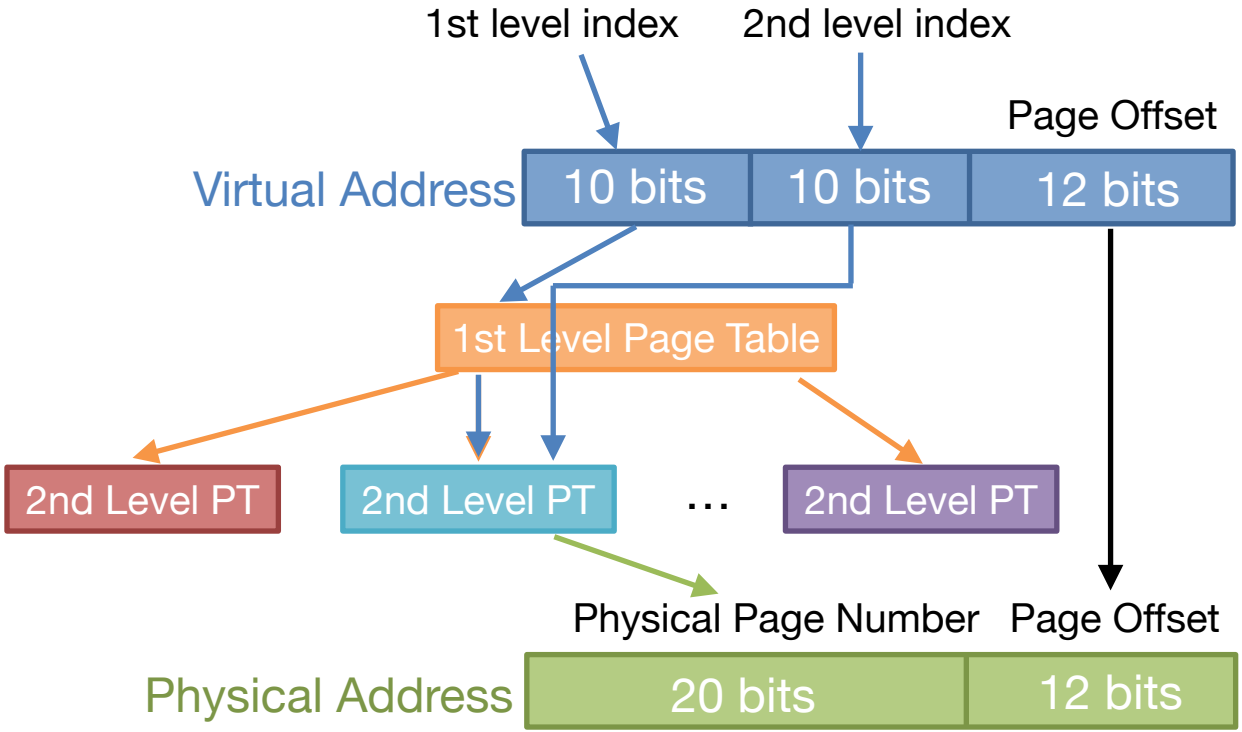
- We want a page table to be equal to the size of one page
  - In this case, its 4KB
- Q: How many entries can I fit in my 1st level page table? (Each PTE is about 4 bytes -> PPN + status bits)
  - 1024
- Q: How many bits do I need to index a page table with 1024 entries?
  - 10
- Q: How many entries can I fit in my 2nd level page table? (Each PTE is about 4 bytes - PPN + status bits)

Multi-level Page Table Translation

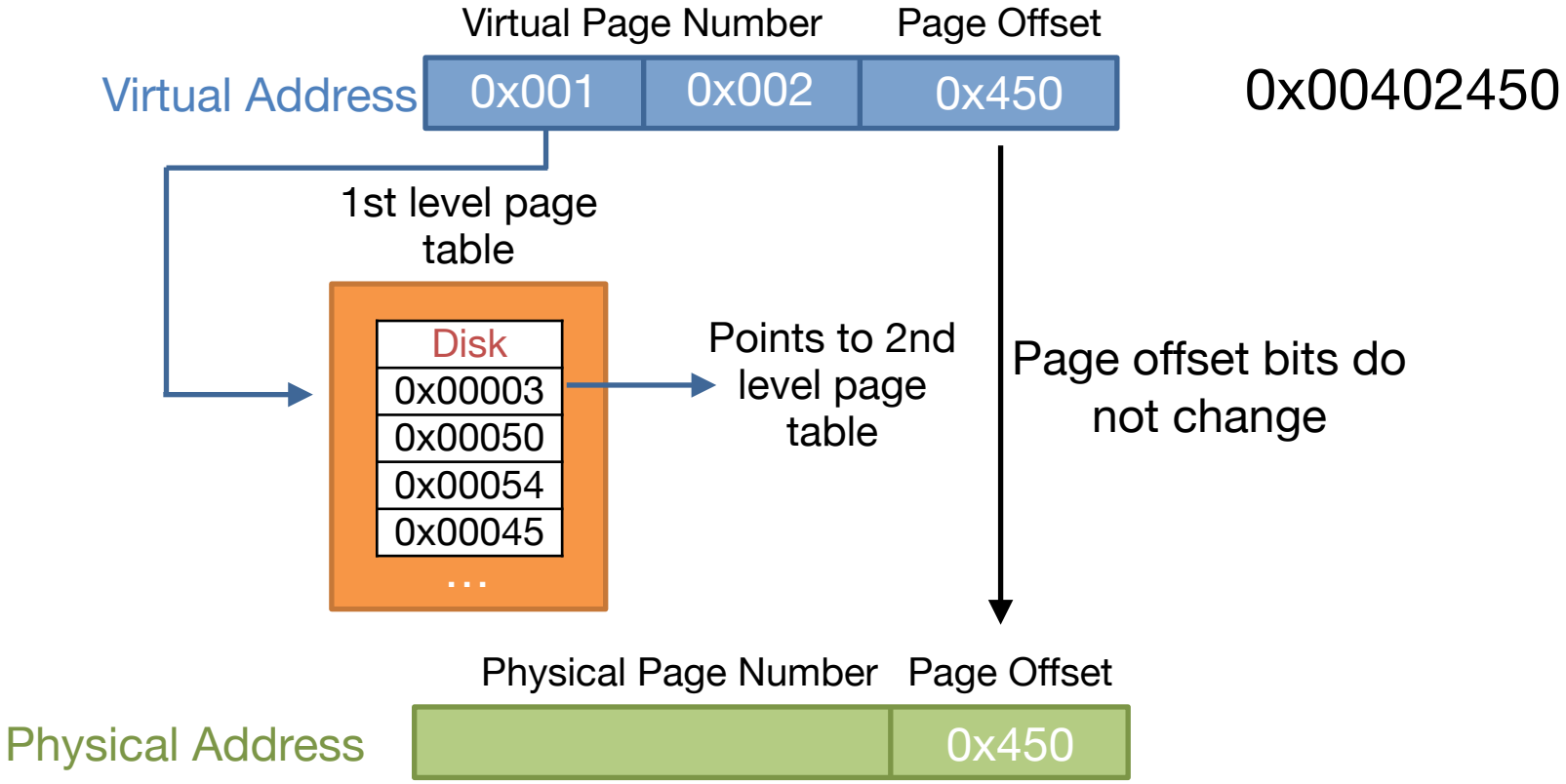
- We want a page table to be equal to the size of one page
  - In this case, its 4KB
- Q: How many entries can I fit in my 1st level page table? (Each PTE is about 4 bytes -> PPN + status bits)
  - 1024
- Q: How many bits do I need to index a page table with 1024 entries?
  - 10
- Q: How many entries can I fit in my 2nd level page table? (Each PTE is about 4 bytes - PPN + status bits)
  - 1024

Multi-level Page Table Translation

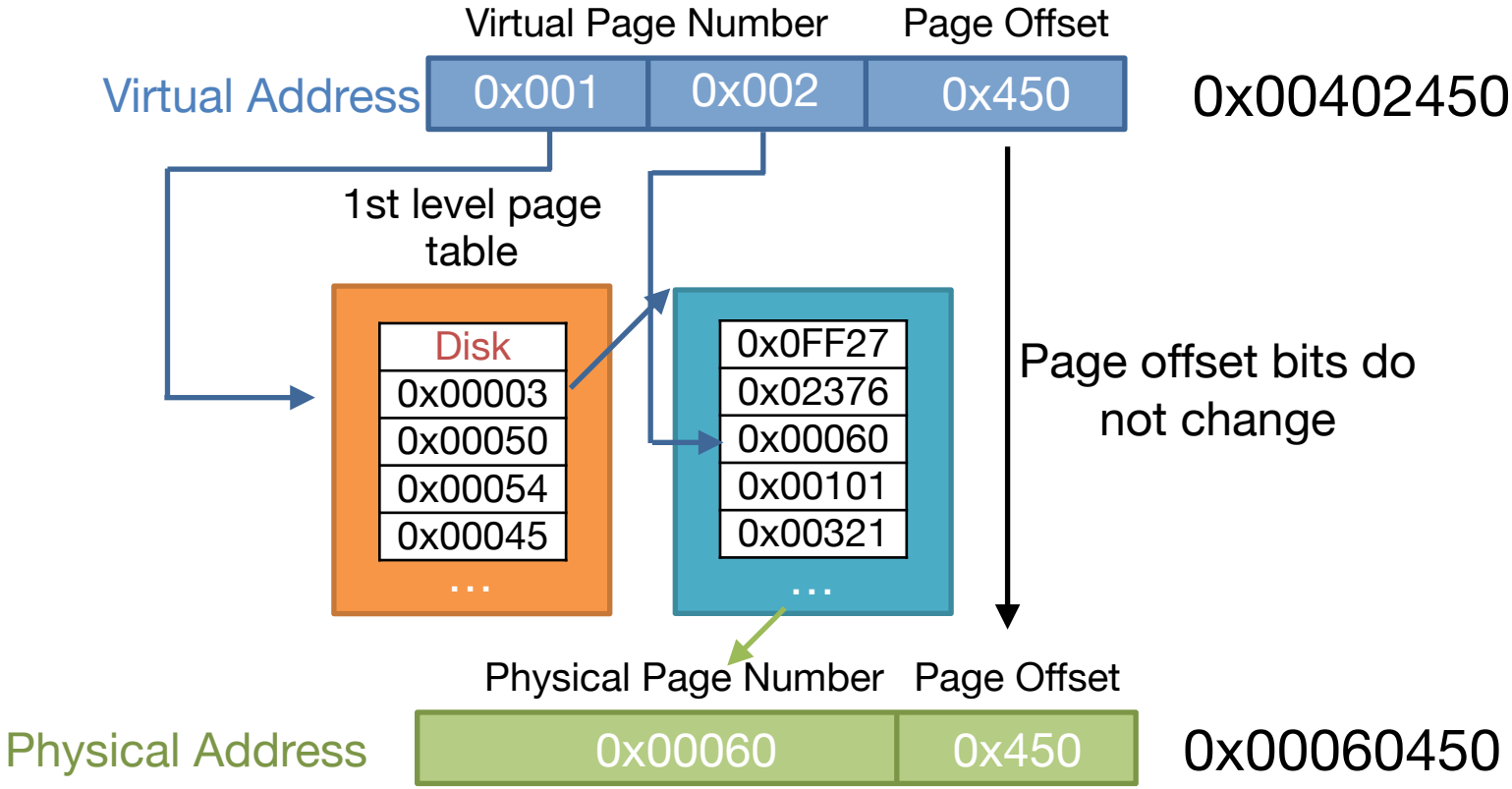
32 bit machine with 4 GB of RAM and 4KB pages



Multi-level Page Table Translation



Multi-level Page Table Translation



1-level vs 2-level Page Tables

- Q: If I'm running 10 applications on my 32-bit computer with 4GB RAM, 4KB pages and 1-level page tables, how much of my RAM is consumed by page tables? (size of PTE is 4 bytes)

1-level vs 2-level Page Tables

- Q: If I'm running 10 applications on my 32-bit computer with 4GB RAM, 4KB pages and 1-level page tables, how much of my RAM is consumed by page tables? (size of PTE is 4 bytes)
  - VA size = 32 bits
  - PA size = $\log_2(4\text{ GB}) = \log_2(2^2 * 2^{30}) = 32$ bits
  - # bits in page offset = $\log_2(4\text{ KB}) = \log_2(2^2 * 2^{10}) = 12$
  - # bits in VPN = 32 - 12 = 20
  - # entries in page table = 2^{20}
  - size of each entry is ~4 bytes
  - size of one page table = $2^{20} * 2^2 = 2^{22}$
  - total RAM consumed by pages tables = $10 * 2^{22}$ bytes = 40 MB

1-level vs 2-level Page Tables

- Q: If I'm running 10 applications on my 32-bit computer with 4GB RAM, 4KB pages and a 2-level page table, how much of my RAM is consumed by 1st level page tables? (size of PTE is 4 bytes)

1-level vs 2-level Page Tables

- Q: If I'm running 10 applications on my 32-bit computer with 4GB RAM, 4KB pages and a 2-level page table, how much of my RAM is consumed by 1st level page tables? (size of PTE is 4 bytes)
  - # bits in VPN = 32 - 12 = 20 = 10 bits for level 1 + 10 bits for level 2
  - Size of 1st level page table = page size = 4KB
  - total RAM consumed by 1st level pages tables = 10 * 2^{12} bytes = 40KB

Multi-level Page Tables

- While actively running a program, what is the minimum number of page table pages would I need in the RAM to access data?

Multi-level Page Tables

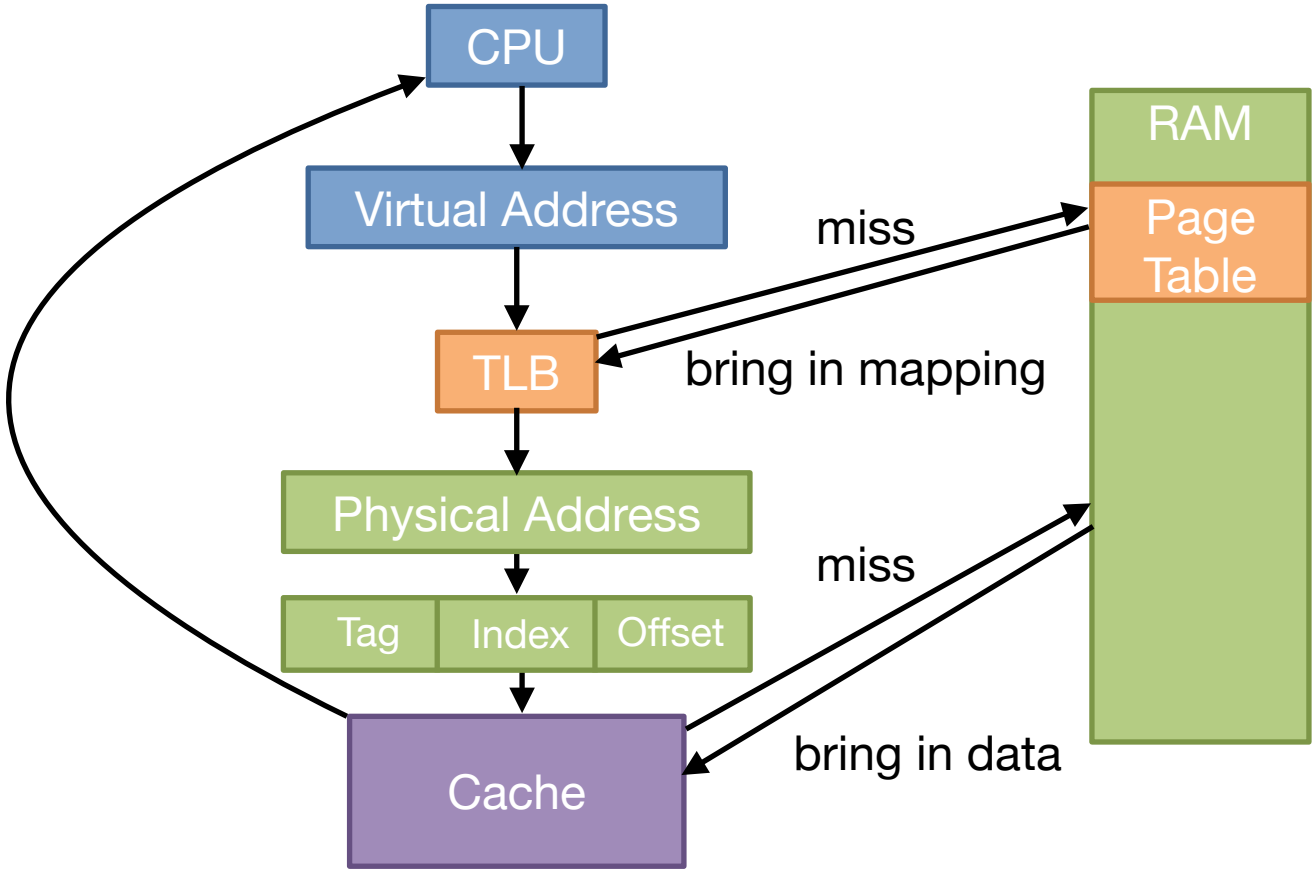
- While actively running a program, what is the minimum number of page table pages would it have in the RAM to access data?
  - 2: The first level page table is always in the RAM. There will be at least one 2nd level page table in the RAM in order to access the data pages

Do multi-level page tables alter the TLB?

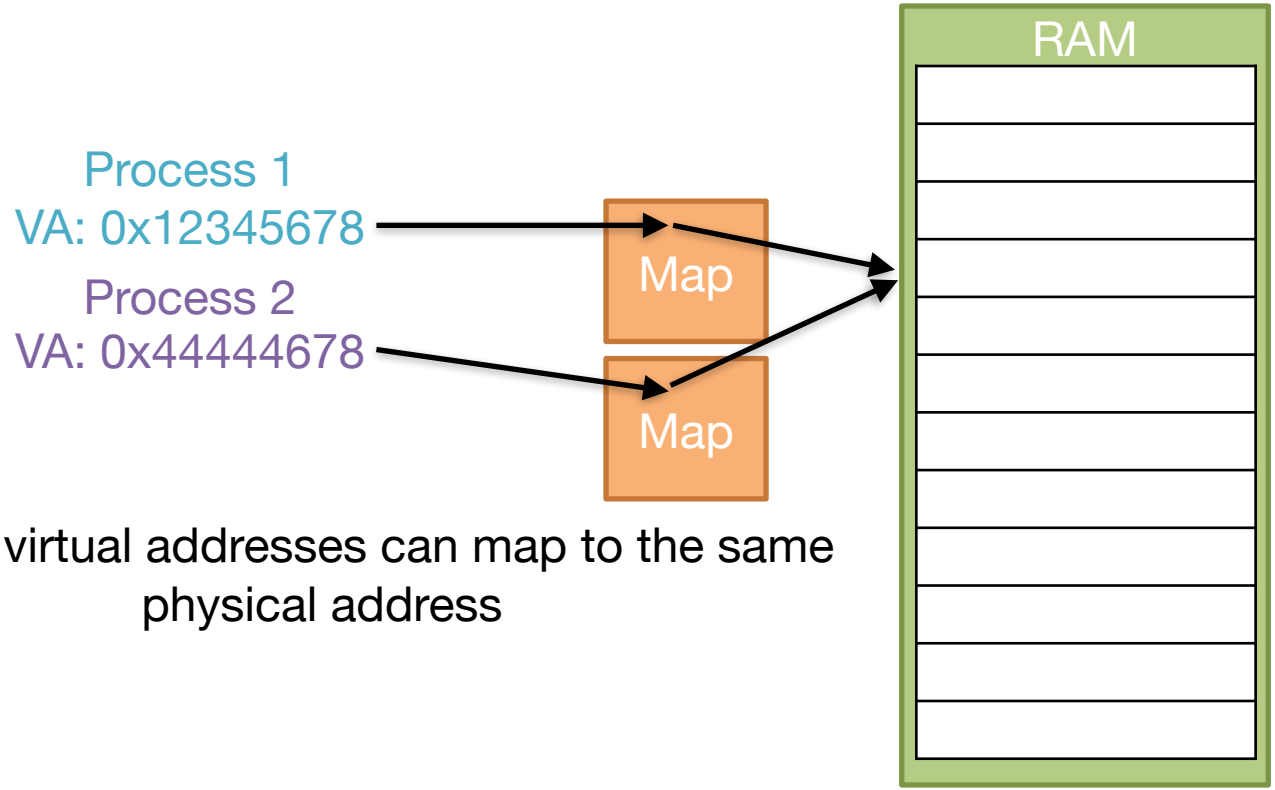
- No, the TLB still contains a mapping from the VPN to the final PPN
- The multilevel page table just makes it take longer for us to find the mapping when it is not stored in the TLB

Caches and Virtual Memory

Physically Indexed, Physically Tagged Cache

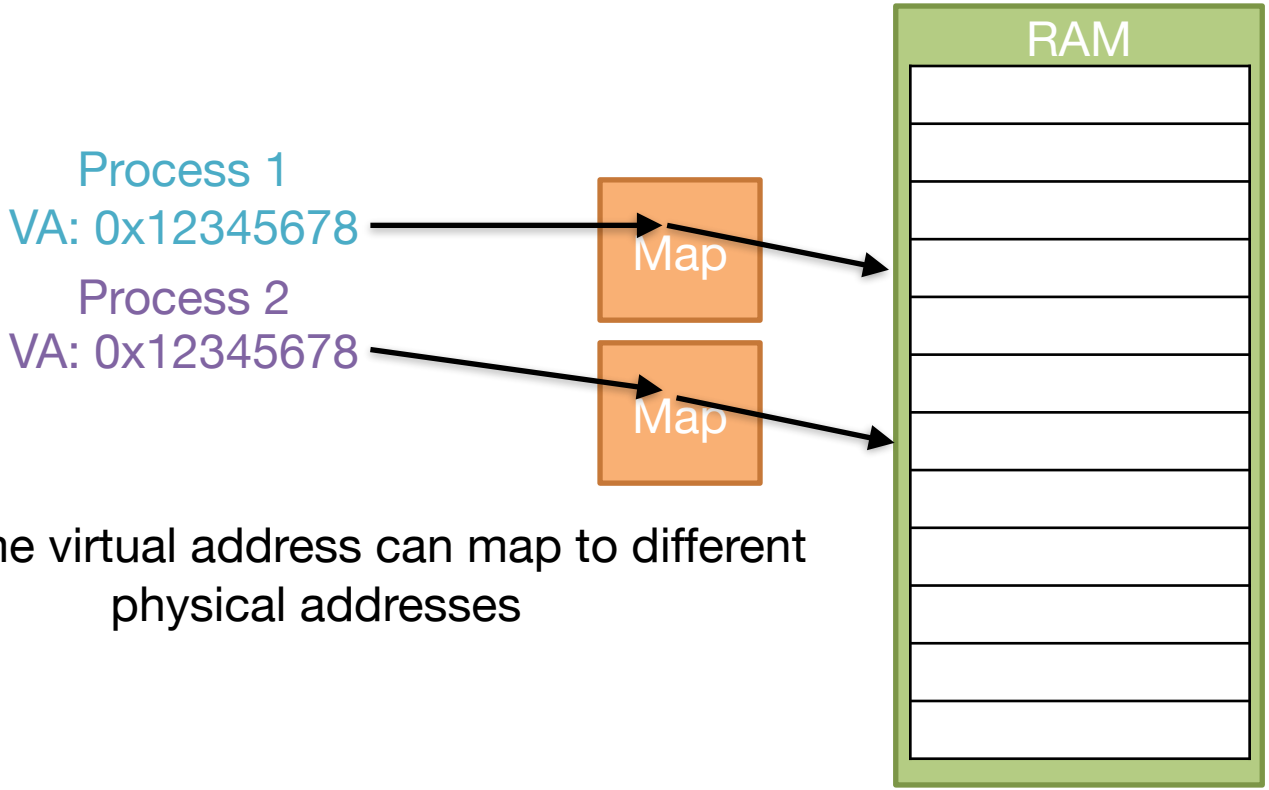


Synonyms



Different virtual addresses can map to the same physical address

Homonyms

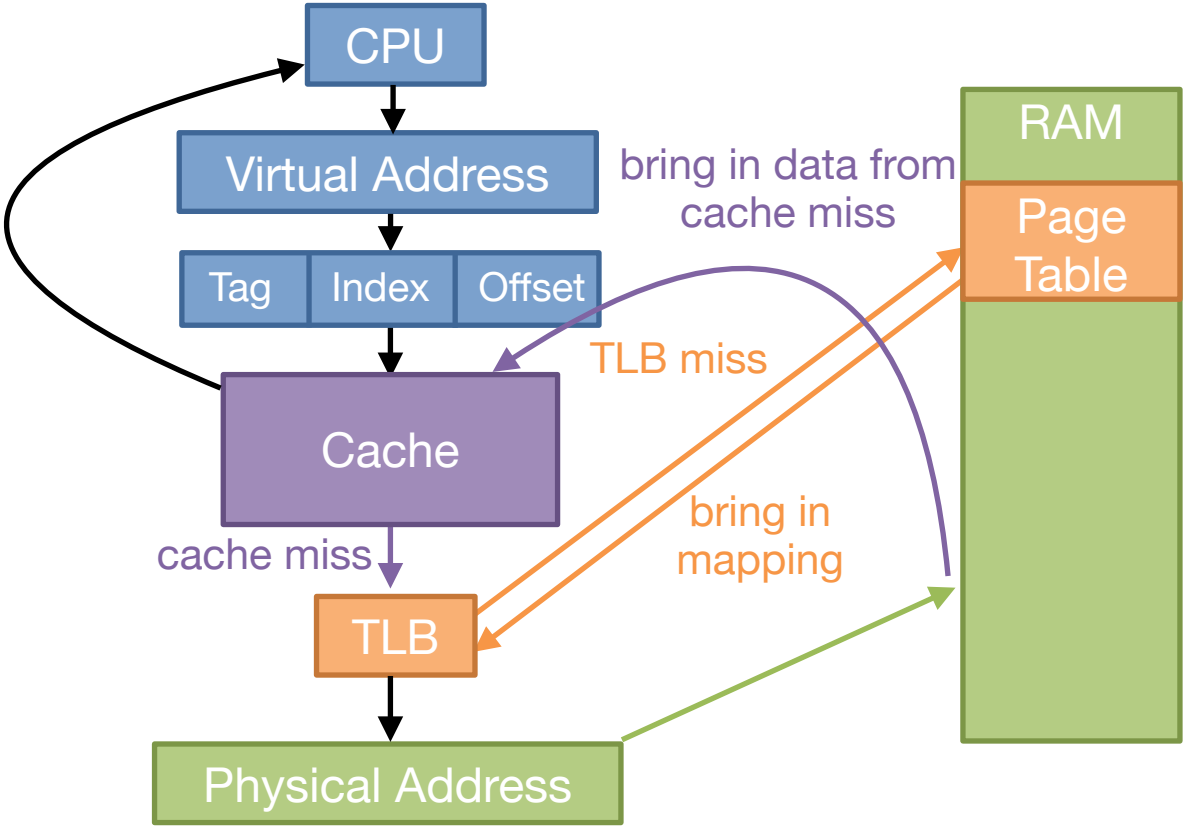


The same virtual address can map to different physical addresses

Physically Indexed, Physically Tagged Cache

- Can we use the same cache for multiple processes?
  - Yes! If the cache is physically indexed, physically tagged, we can use the same cache
- Downside
  - Must translate address before accessing the cache

Virtually Indexed, Virtually Tagged Caches



Virtually Indexed, Virtually Tagged Caches

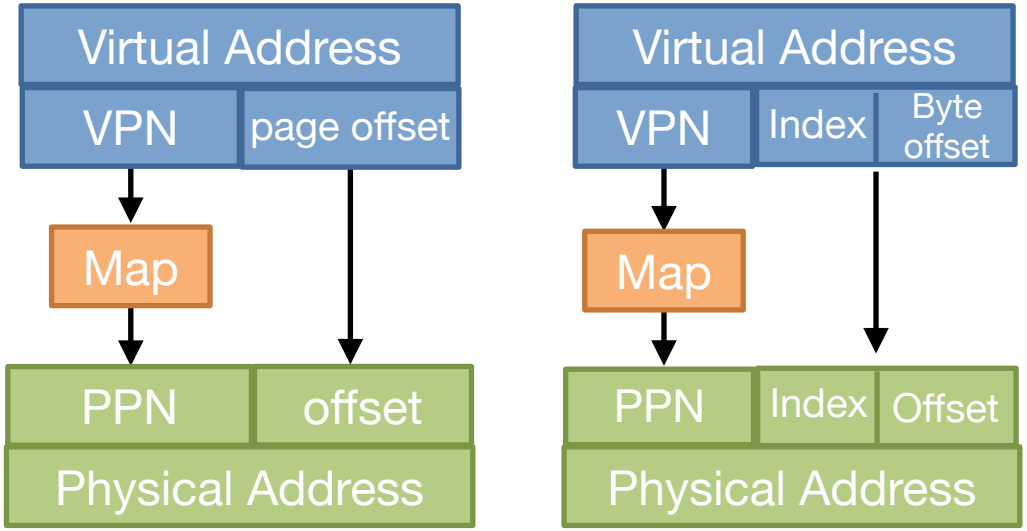
- Can we use the same virtual cache for different processes?
  - No because of synonyms and homonyms
- How do we solve this issue?
  - Flushing the cache on a context switch

Best of Both Worlds (Virtually Indexed, Physically Tagged Caches)

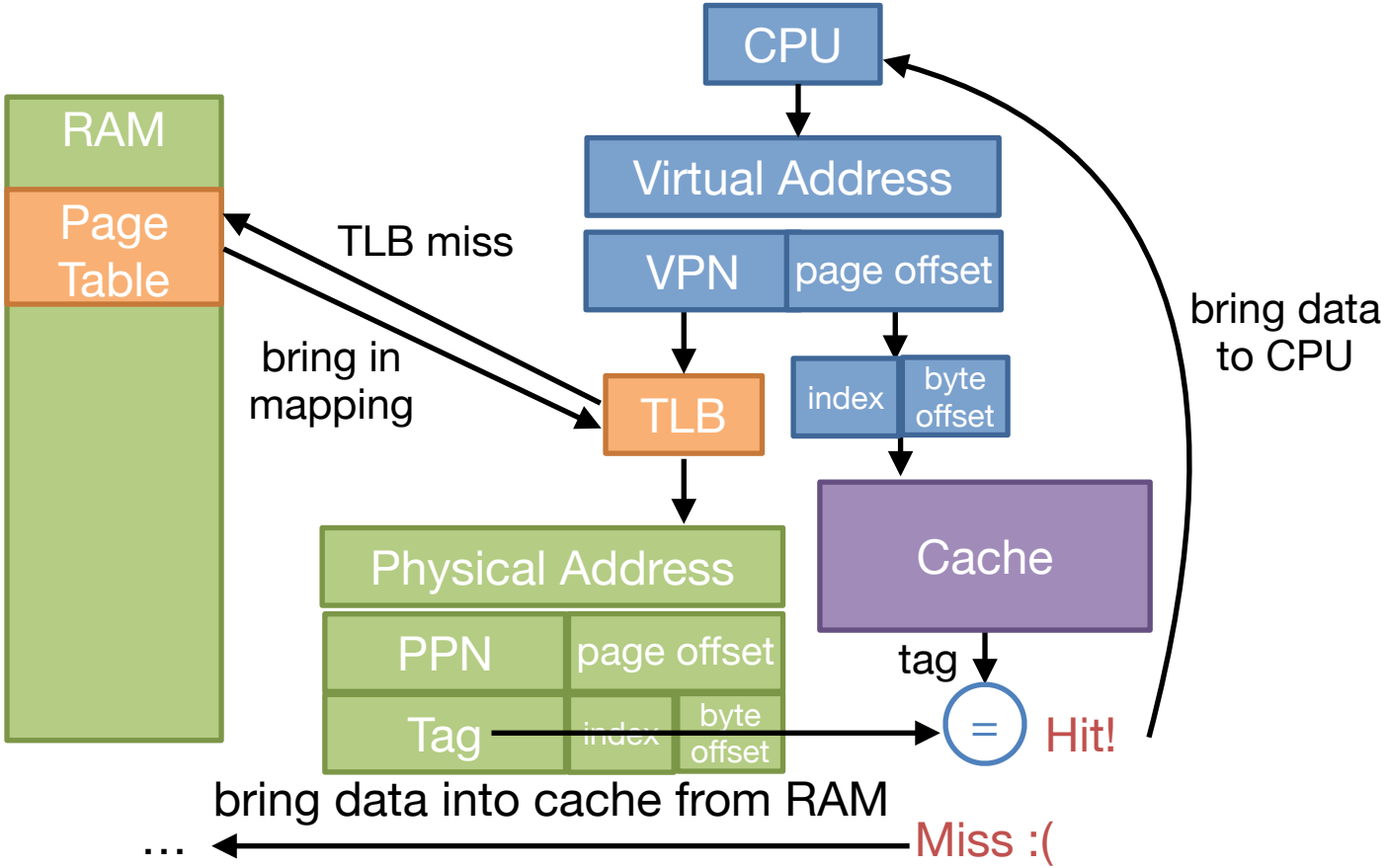
- We want to use the physical address to access the cache
  - To avoid synonym and homonym problem
- Can we perform the TLB lookup and cache index lookup in parallel?
- How can we get part of the physical address from the virtual address without going through the TLB?
  - Some of the bits remain the same after translation

How to make this work?

- Which bits remain the same after the translation?
  - The page offset bits



Virtually Indexed, Physically Tagged (VIPT) Caches



VIPT Cache Size

- The size of our cache is limited by the number of page offset bits
- Q: With 4kB pages, how many bytes can a direct-mapped (1-way) VIPT cache store?

VIPT Cache Size

- The size of our cache is limited by the number of page offset bits
- Q: With 4kB pages, how many bytes can a direct-mapped (1-way) VIPT cache store?
  - 4kB
  - We can only use the page offset bits (12 bits for 4kB pages) to index into the cache. So the index can only address 12 bits of address, or 4kB of data

VIPT Cache Size

- The size of our cache is limited by the number of page offset bits
- Q: With 4kB pages, how many bytes can a direct-mapped (1-way) VIPT cache store?
  - 4kB
  - We can only use the page offset bits (12 bits for 4kB pages) to index into the cache. So the index can only address 12 bits of address, or 4kB of data
- How can we make our cache larger with the same page size?

VIPT Cache Size

- The size of our cache is limited by the number of page offset bits
- Q: With 4kB pages, how many bytes can a direct-mapped (1-way) VIPT cache store?
  - 4kB
  - We can only use the page offset bits (12 bits for 4kB pages) to index into the cache. So the index can only address 12 bits of address, or 4kB of data
- How can we make our cache larger with the same page size?
  - Increase the associativity