CS61C: Great Ideas in Computer Architecture (Machine Structure)

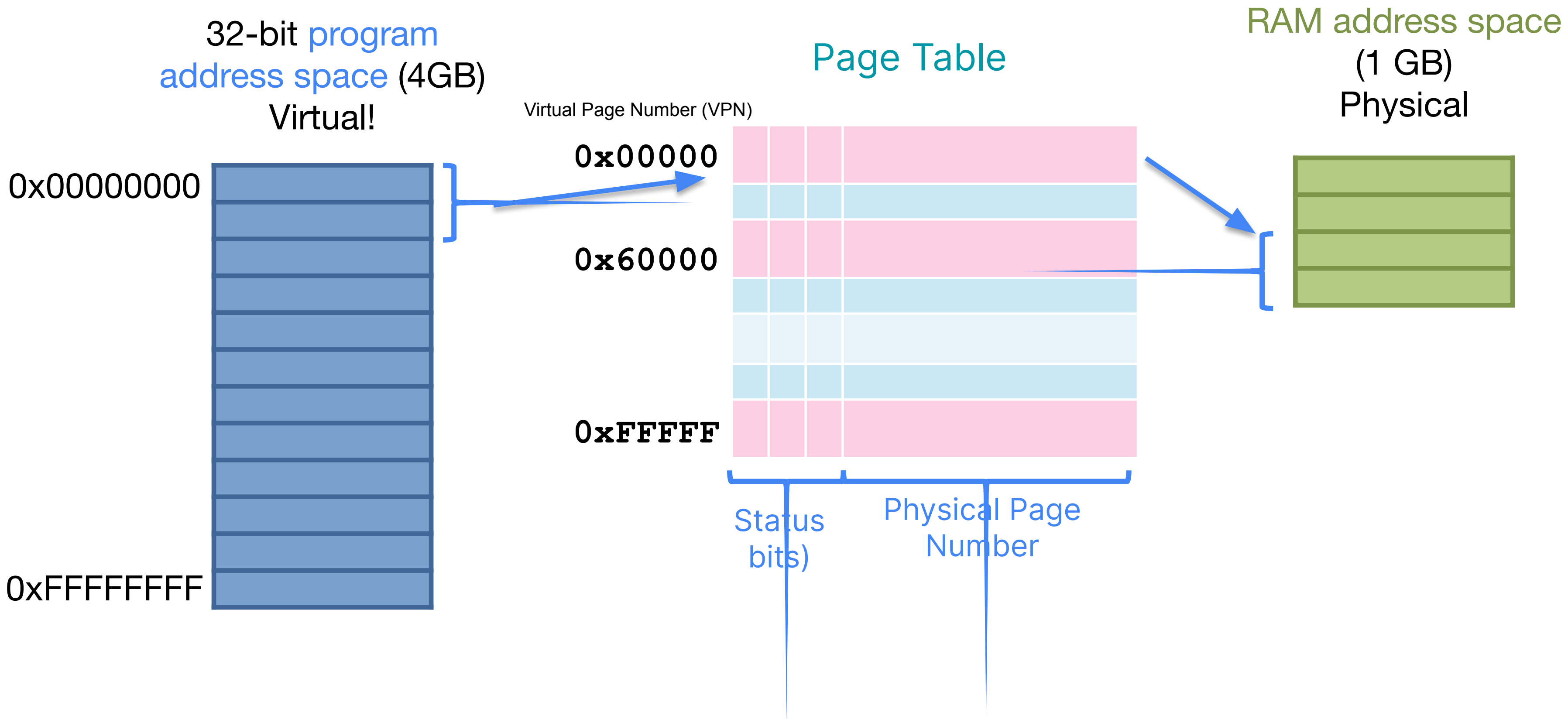
Instructors: Rosalie Fang, Charles Hong, Jero Wang

Lecture feedback:
https://tinyurl.com/fyr-feedback

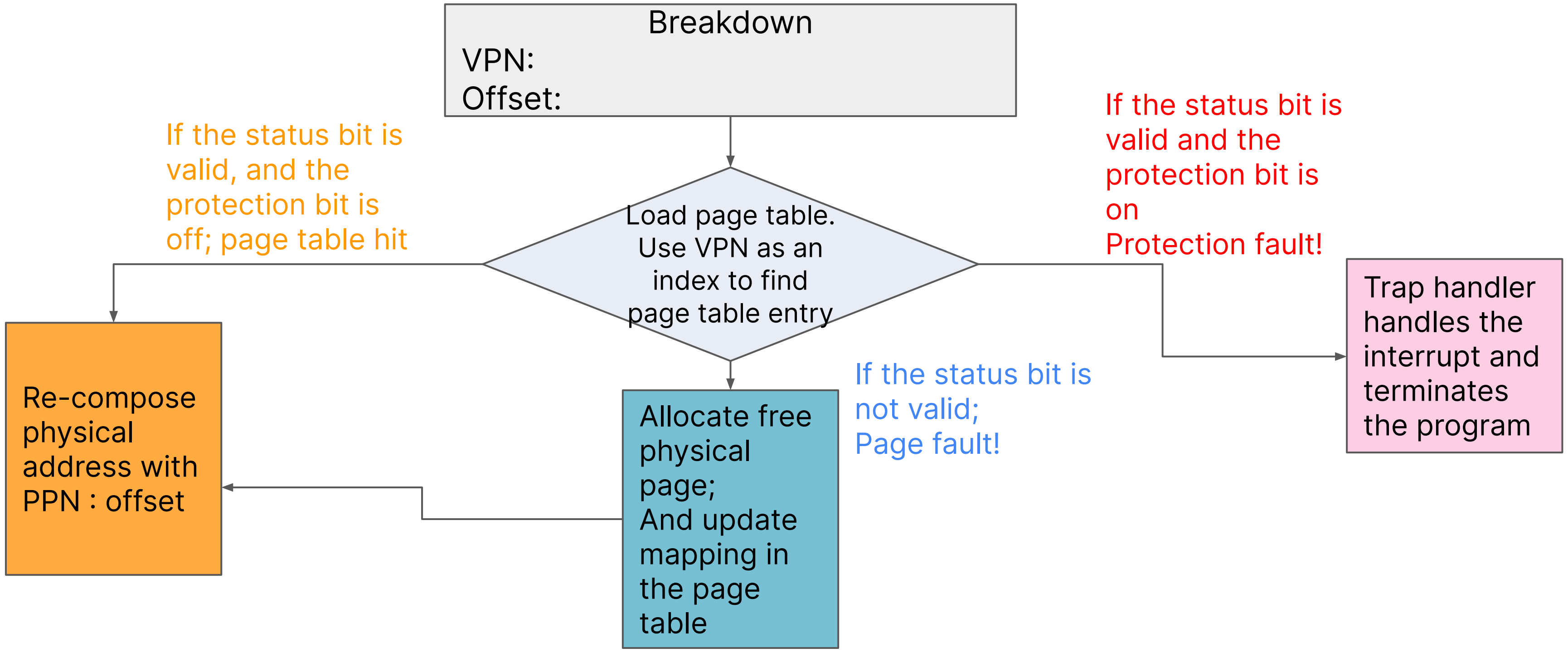
Come sit at the front 🥺🥺🥺

Review

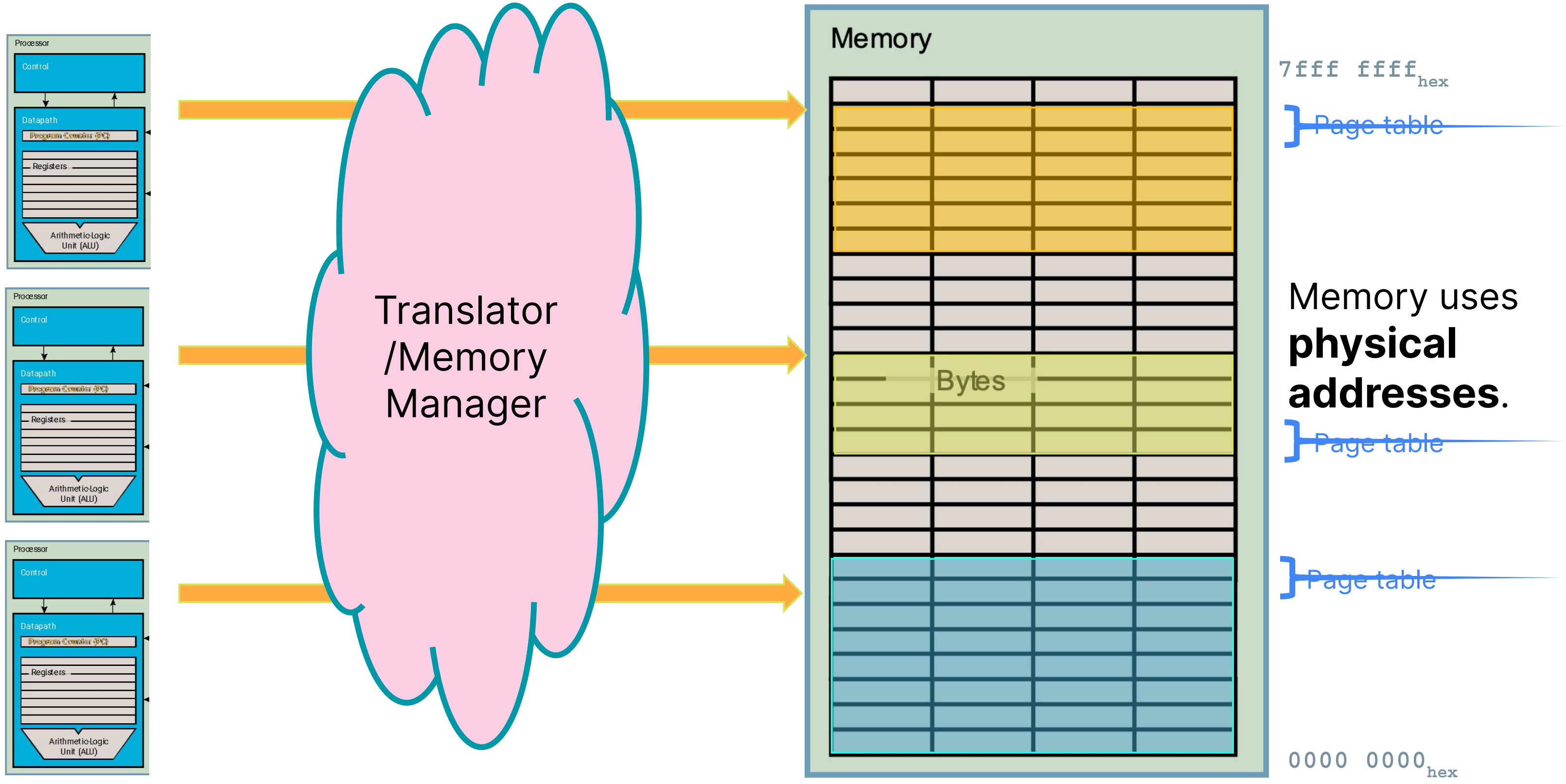
Virtual memory translation



VM Workflow so far given a virtual address


Breakdown
VPN:
Offset:
Load page table. Use VPN as an index to find page table entry
If the status bit is valid, and the protection bit is off; page table hit
Re-compose physical address with PPN : offset
If the status bit is not valid; Page fault!
Allocate free physical page; And update mapping in the page table
If the status bit is valid and the protection bit is on Protection fault!
Trap handler handles the interrupt and terminates the program

Conceptual Memory Manager in OS



Memory uses **physical addresses**.

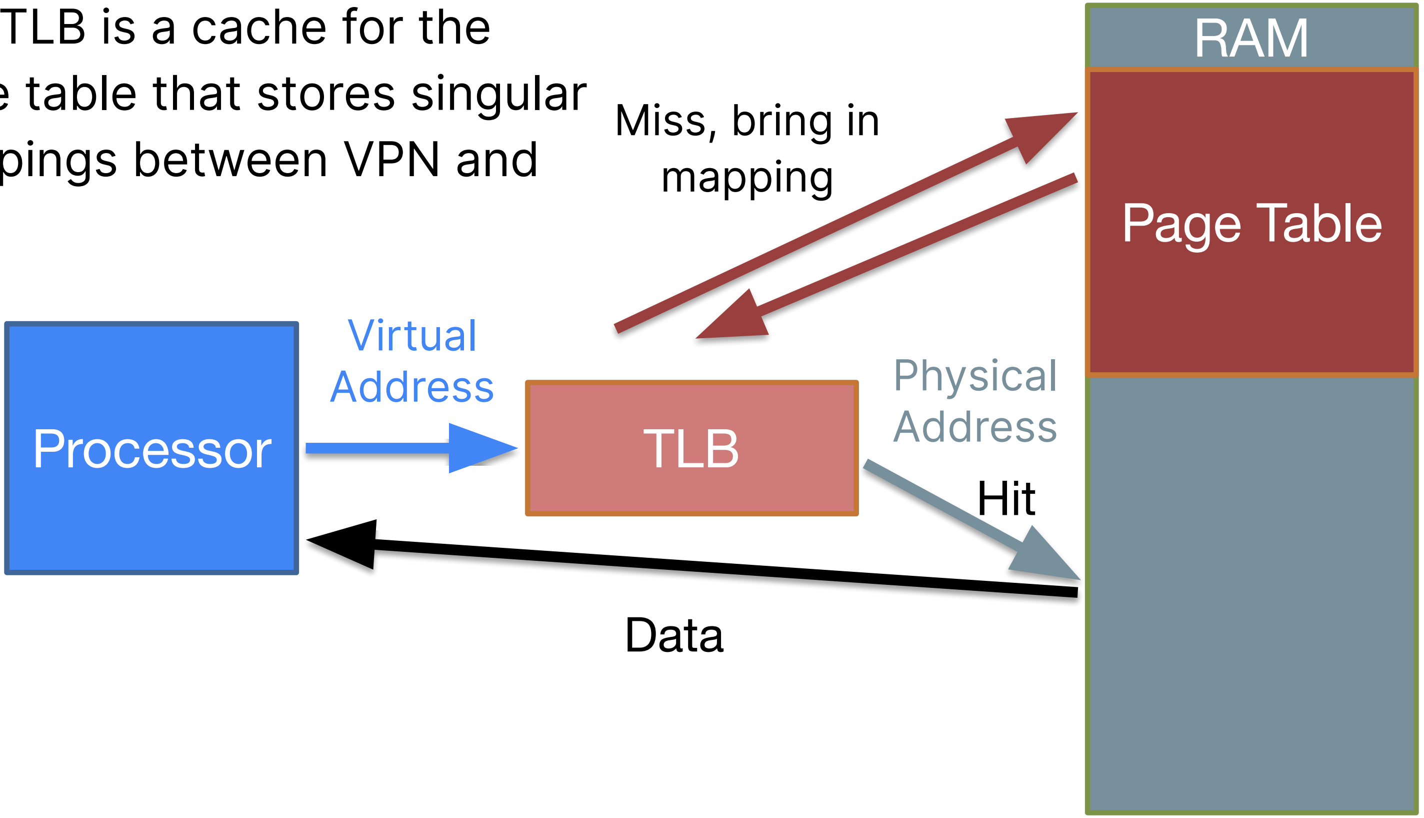
Problems with page tables being stored in memory

- With a virtual address space of 32 bits and a page size of 4KiB (2^{12} bytes)
  - We need 2^{20} page table entries.
  - If each entry is 4B, we need 2^{22} bytes of space, or 2^{10} pages
- Every single time we try to do an address translation, we have to load 1 page from the page table, check the translation, and load the translated physical page
- If there a way we can reduce access time?
  - Caches!

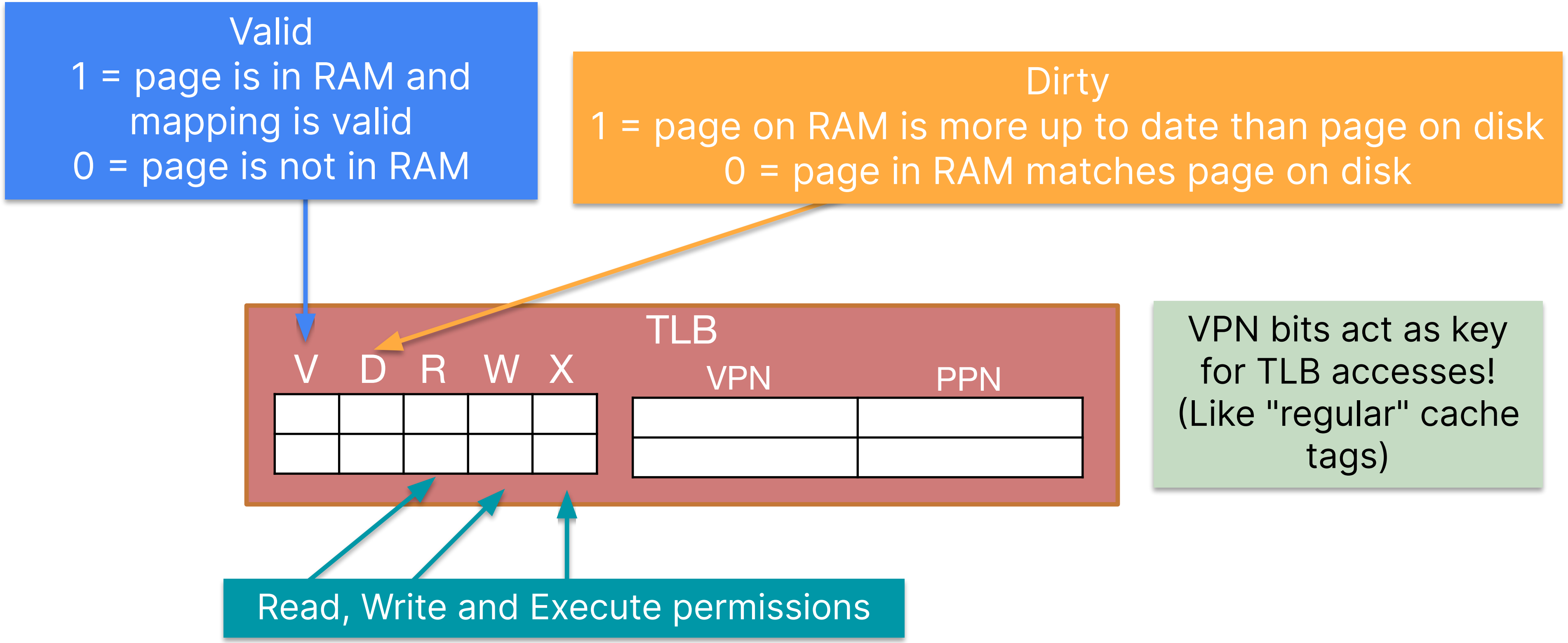
Translation Lookaside Buffer

Translation Lookaside Buffer (TLB)

- The TLB is a cache for the page table that stores singular mappings between VPN and PPN



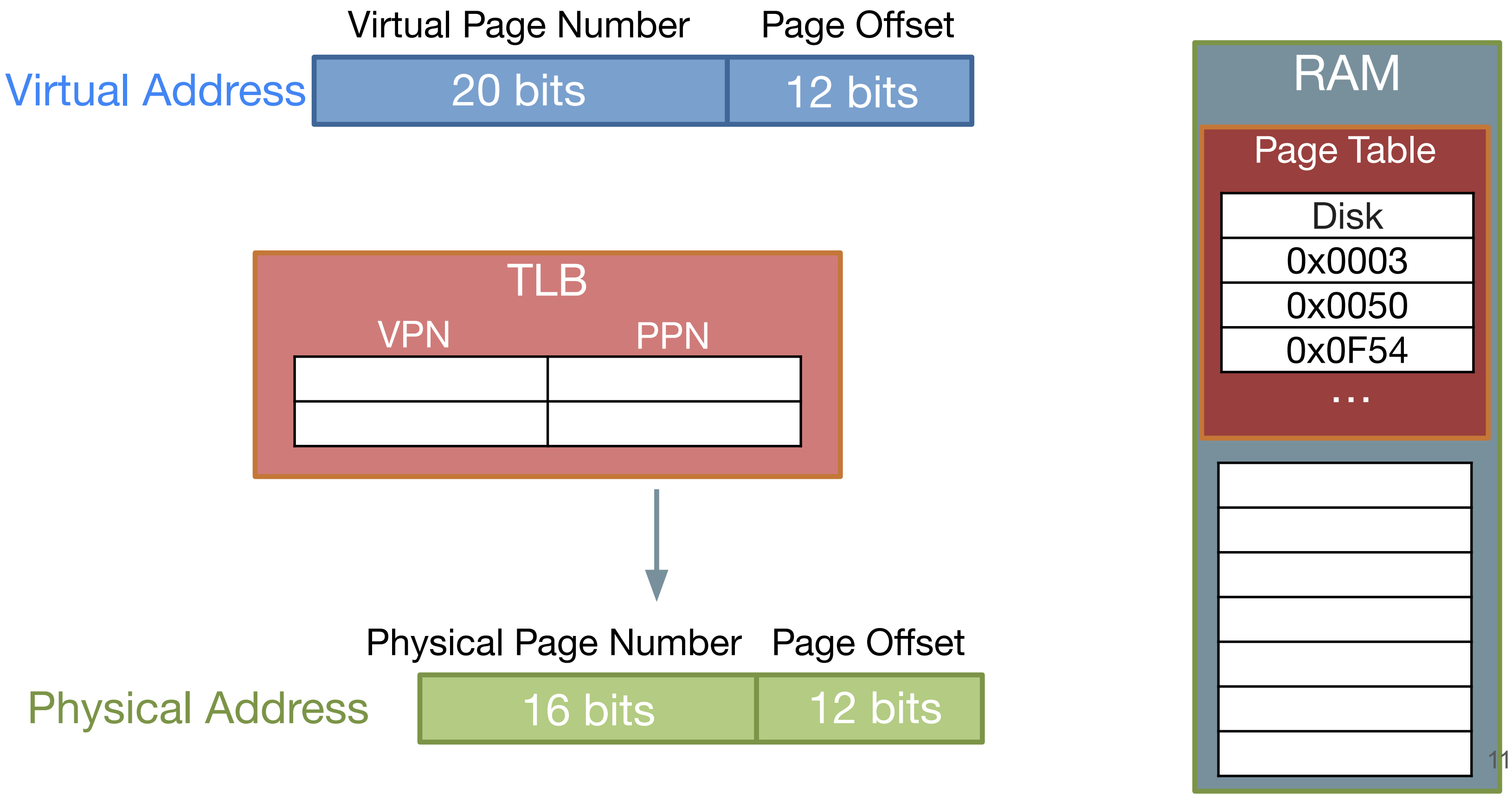
Translation Lookaside Buffer (TLB)


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
TLB
V
D
R
W
X
VPN
PPN
VPN bits act as key for TLB accesses! (Like "regular" cache tags)
Read, Write and Execute permissions

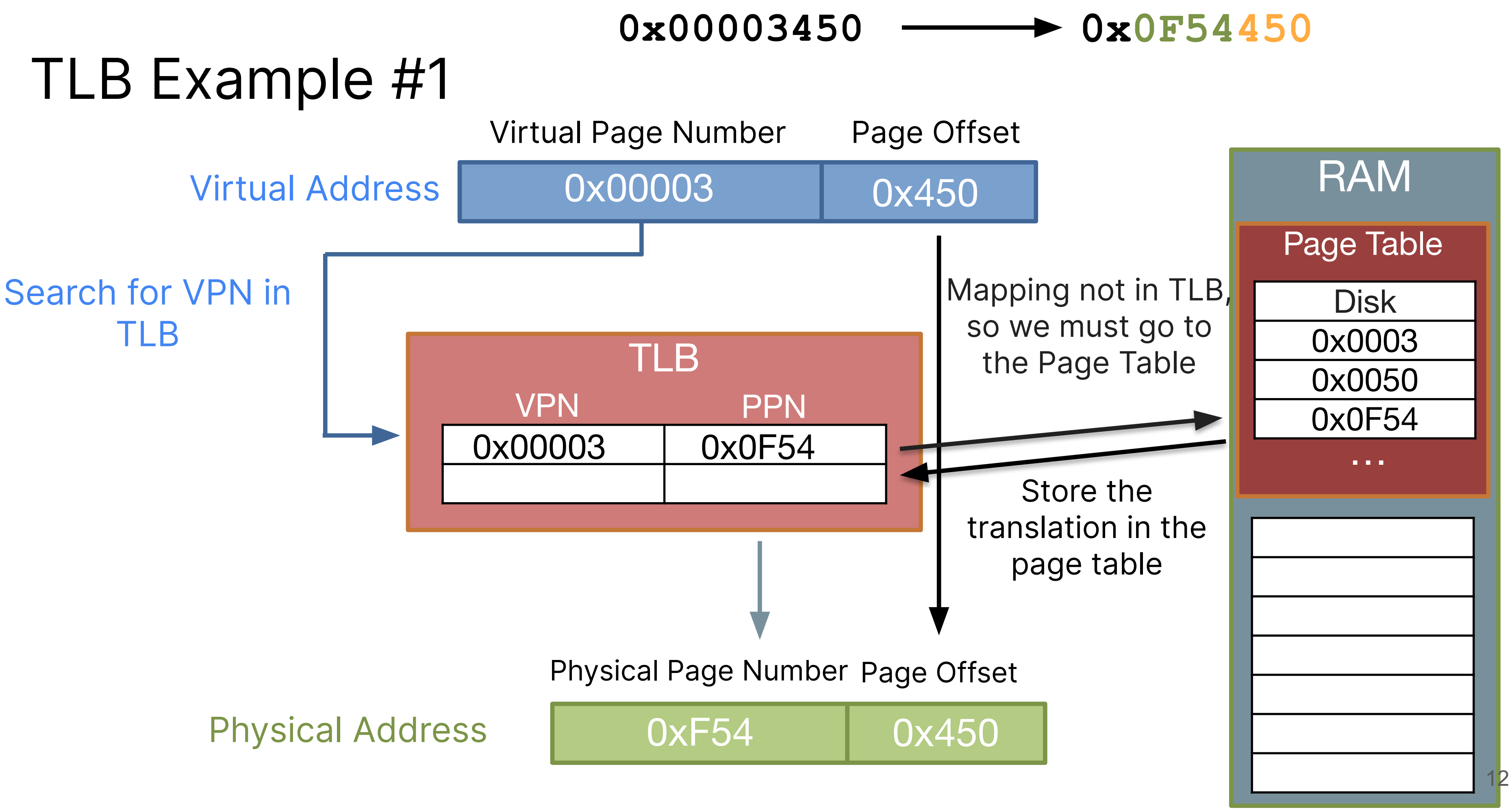
Translation Lookaside Buffer (TLB)

- 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
- Context switches - 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

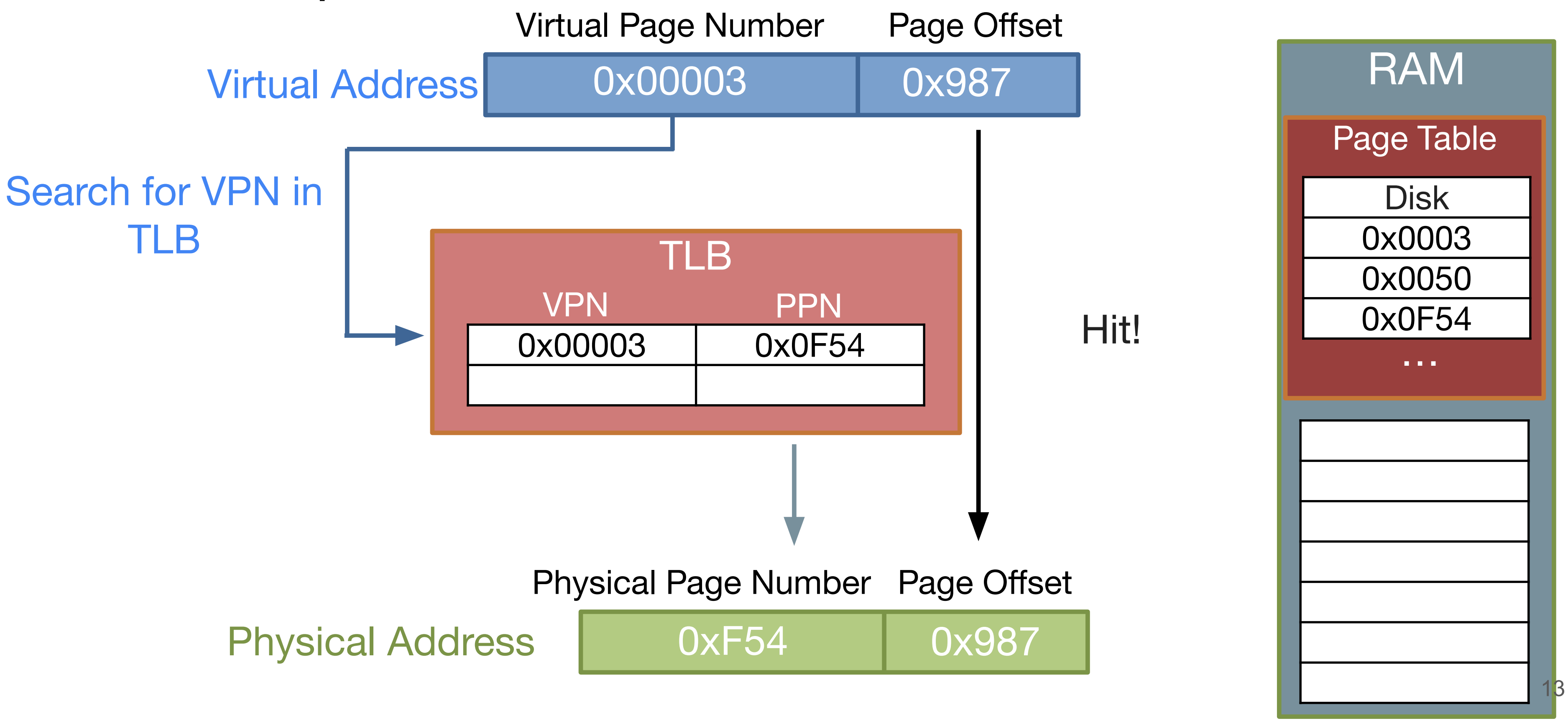

Virtual Page Number
Page Offset
Virtual Address
20 bits
12 bits
TLB
VPN
PPN
Physical Page Number
Page Offset
Physical Address
16 bits
12 bits
RAM
Page Table
Disk
0x0003
0x0050
0x0F54
...

TLB Example #1



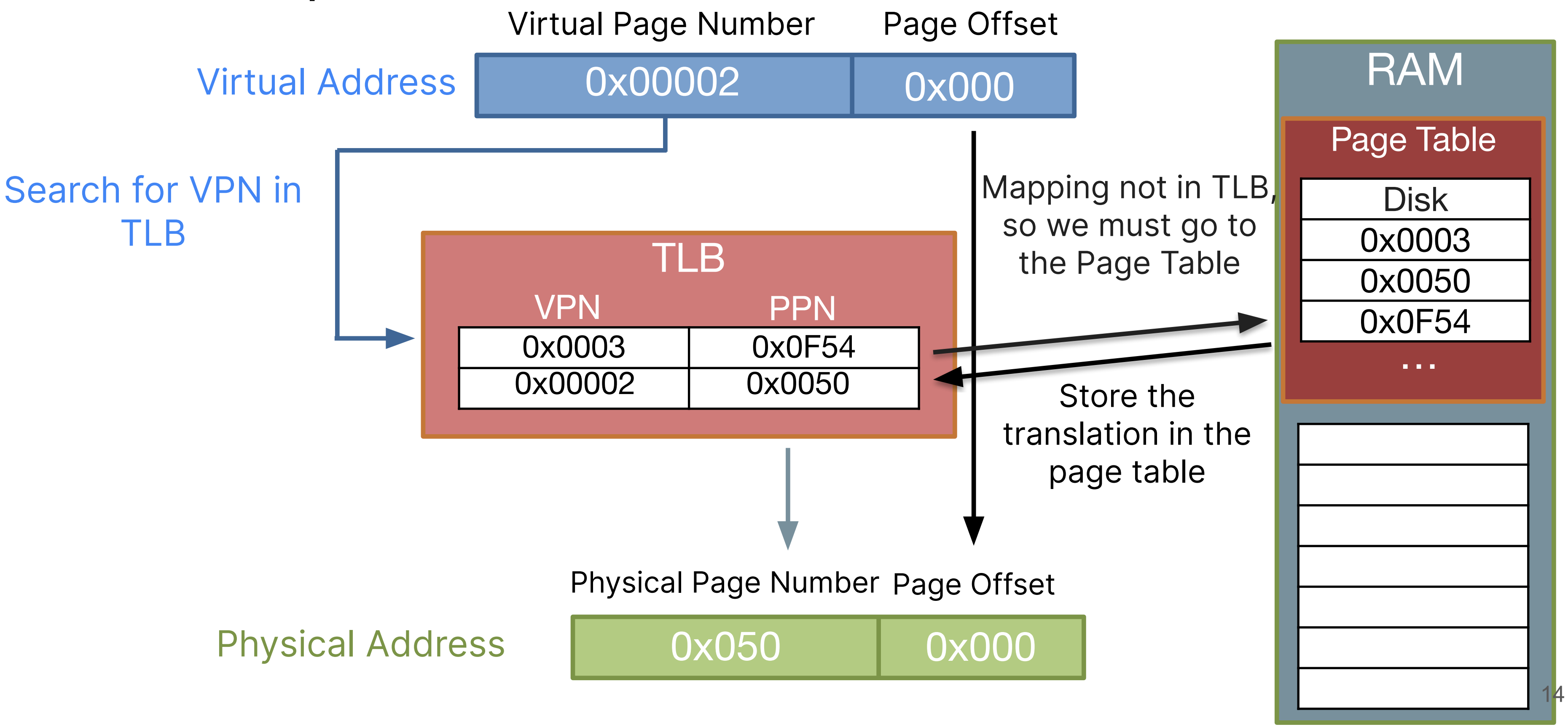
TLB Example #2

0x00003987 → 0x0F54987

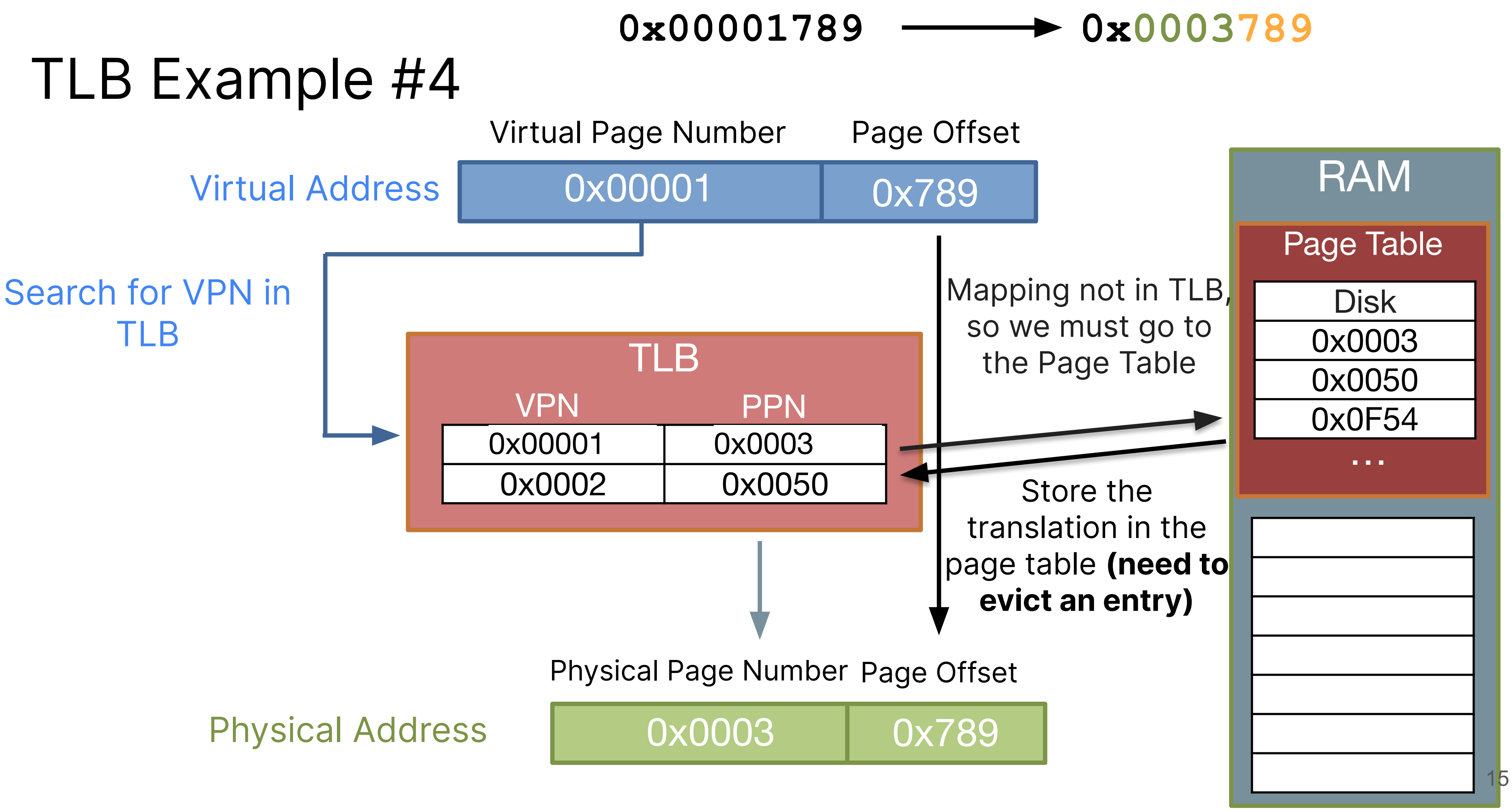


TLB Example #3

0x00002000 → 0x0050000

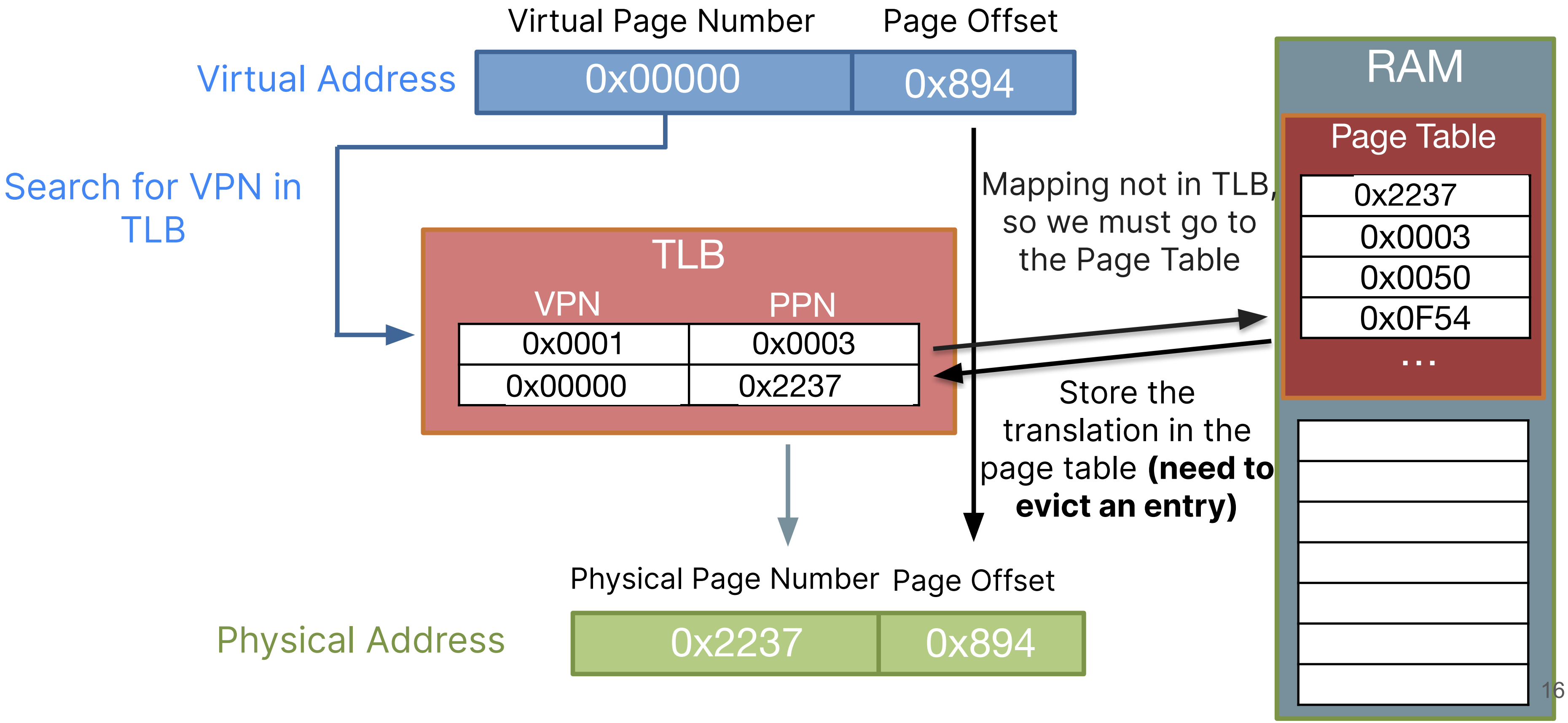


TLB Example #4

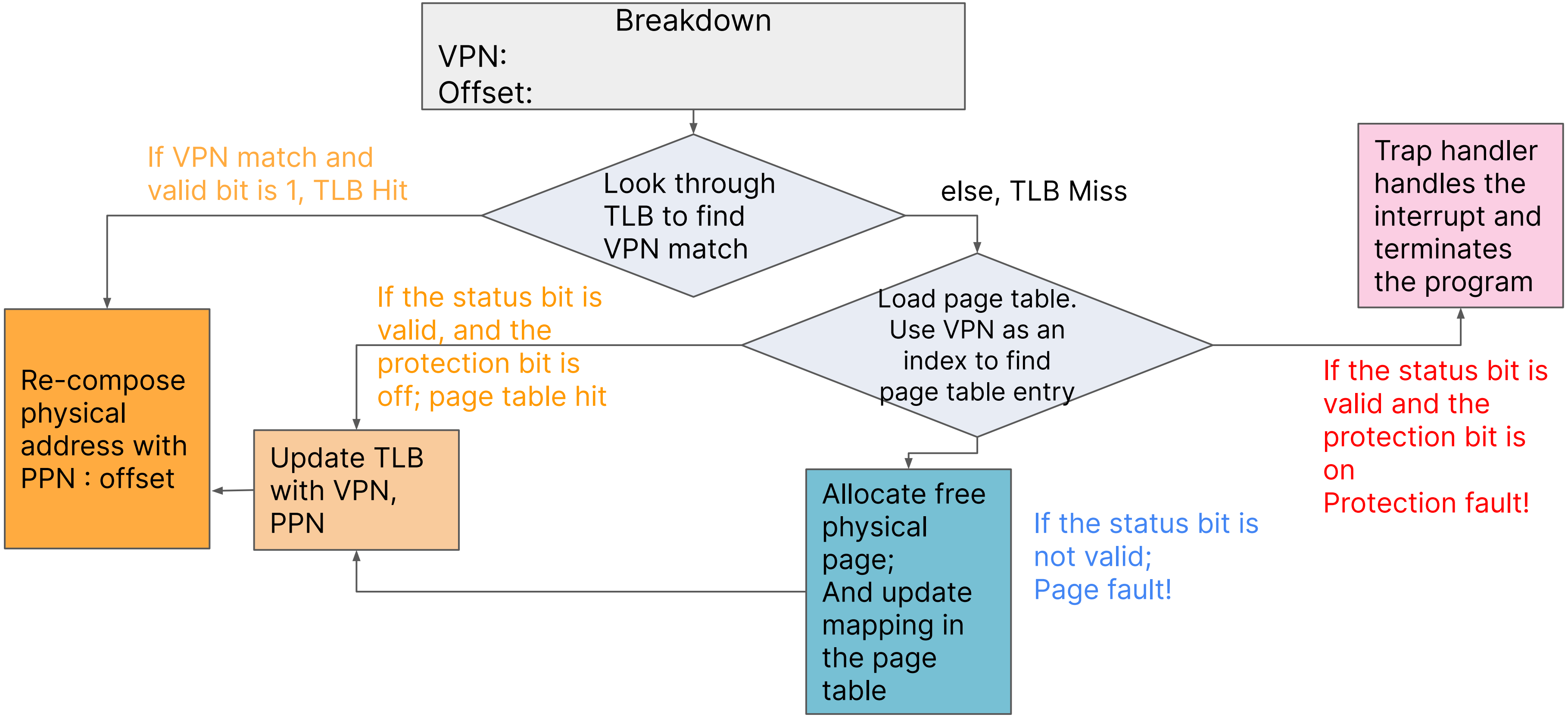


TLB Example #5

`0x00000894` → `0x2237894`



Summary... given a virtual address



Breakdown
VPN:
Offset:
Look through TLB to find VPN match
If VPN match and valid bit is 1, TLB Hit
else, TLB Miss
Re-compose physical address with PPN : offset
Load page table. Use VPN as an index to find page table entry
If the status bit is valid, and the protection bit is off; page table hit
Update TLB with VPN, PPN
Trap handler handles the interrupt and terminates the program
If the status bit is valid and the protection bit is on
Protection fault!
Allocate free physical page; And update mapping in the page table
If the status bit is not valid;
Page fault!

Problem with Page Table Size

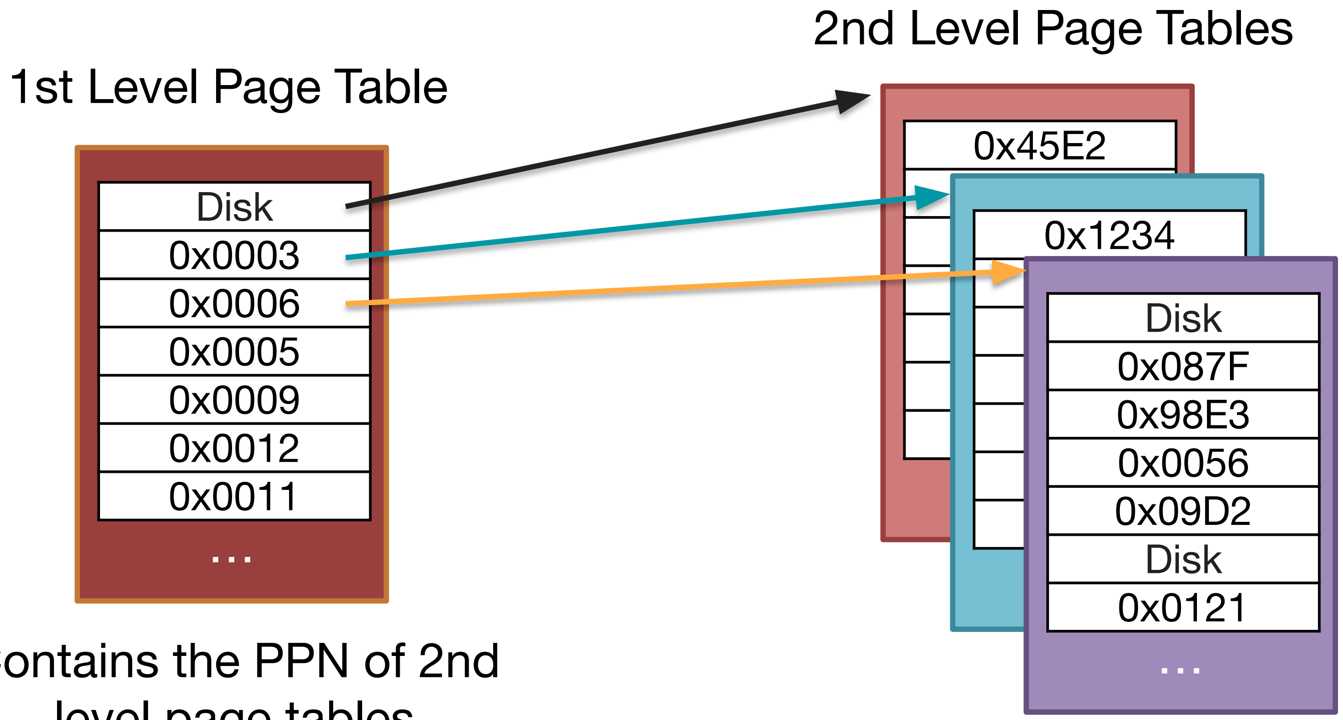
- For 32-bit machine with 4GB RAM, 4kB pages we need:
  - 1M Page Table Entries (32 bits – 12 bits for page offset = 20 bits, 2^{20}=1M)
  - Each PTE is about 4 bytes (20 bits for physical page + status bits)
  - 4MB total
- Not bad…

…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

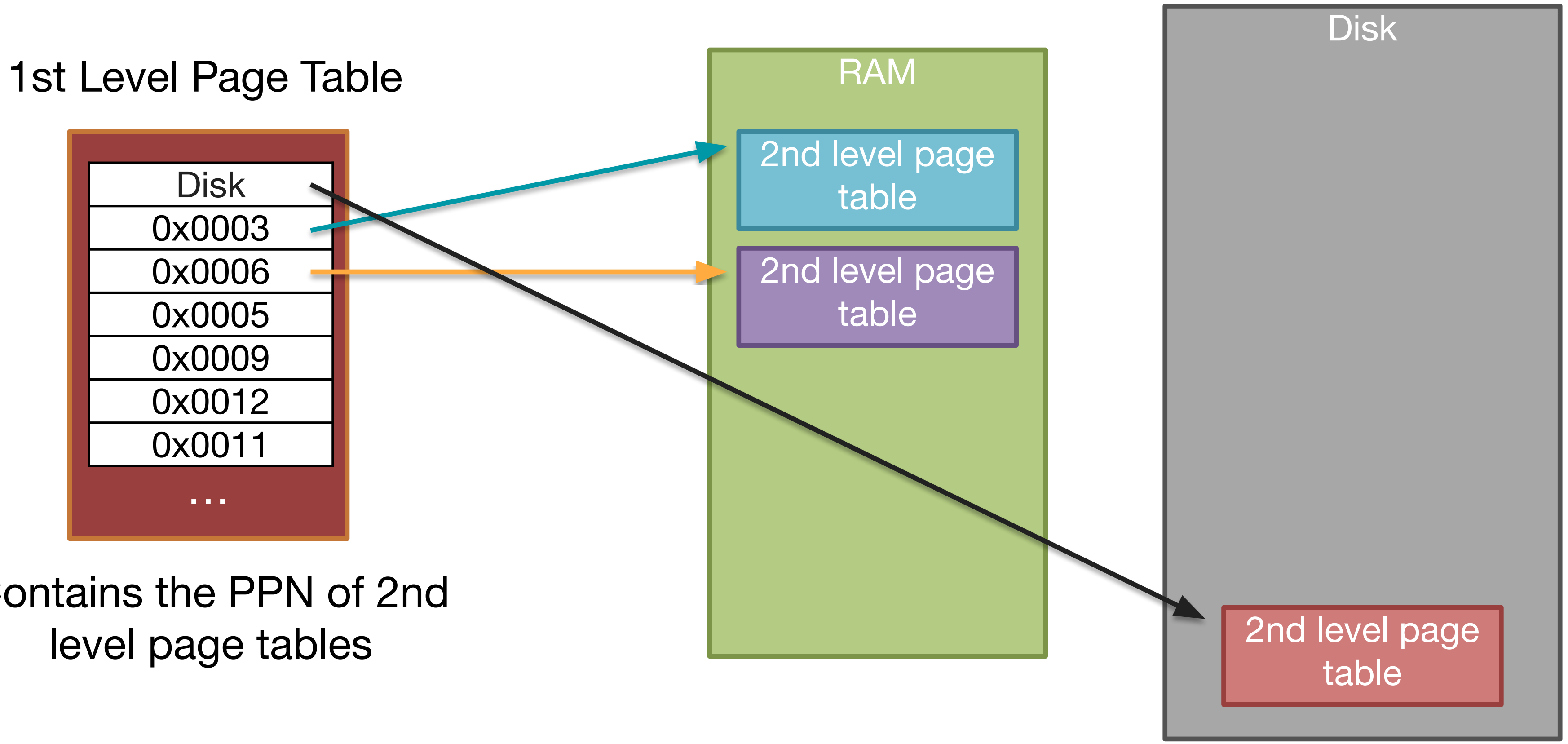
Multi-level Page Tables



Contains the PPN of 2nd level page tables

Contains the PPN of the data we want

Multi-level Page Tables



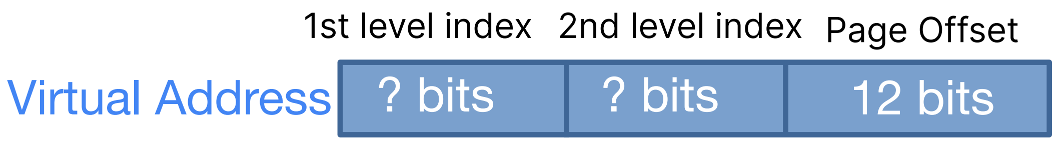
Contains the PPN of 2nd level page tables

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
- Why do we only need the PPN of the 2nd level page tables in real-life implementations?
    - Because in real life, each page in memory typically fits one page table and thus, the rest of the physical address for the start of a 2nd level page table can be inferred!

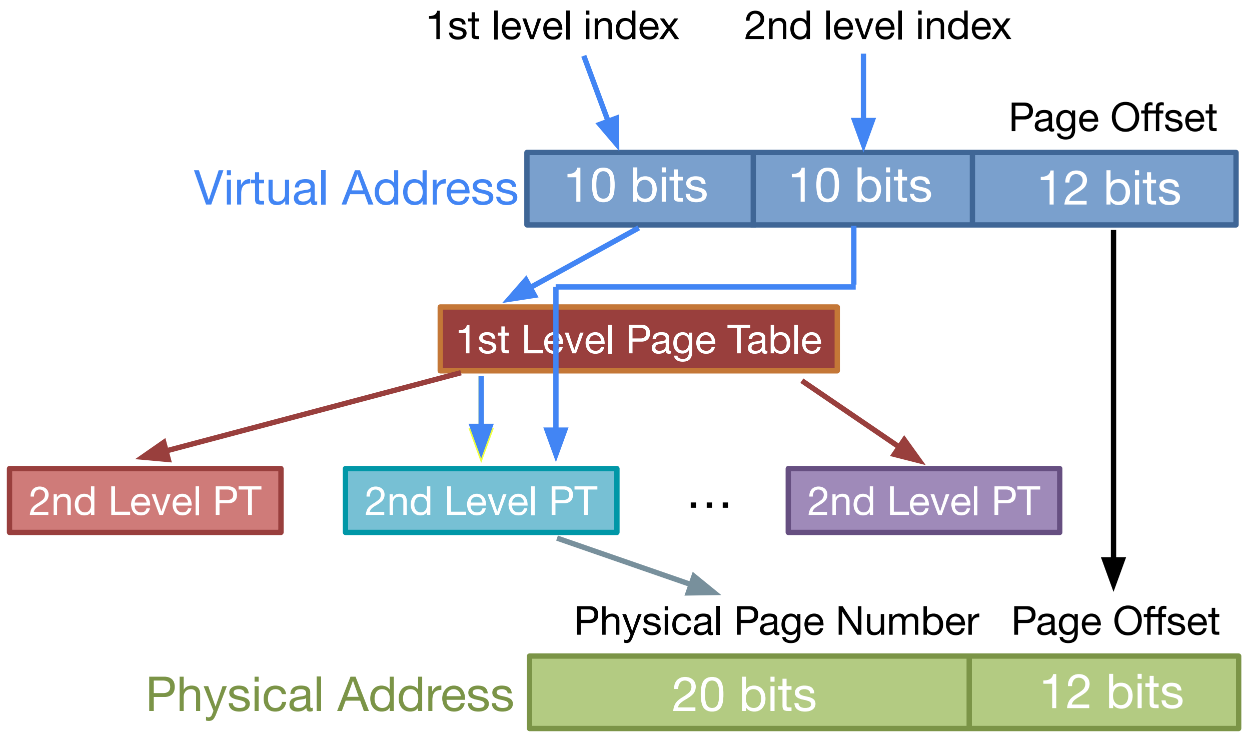
Multi-level Page Table Translation

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

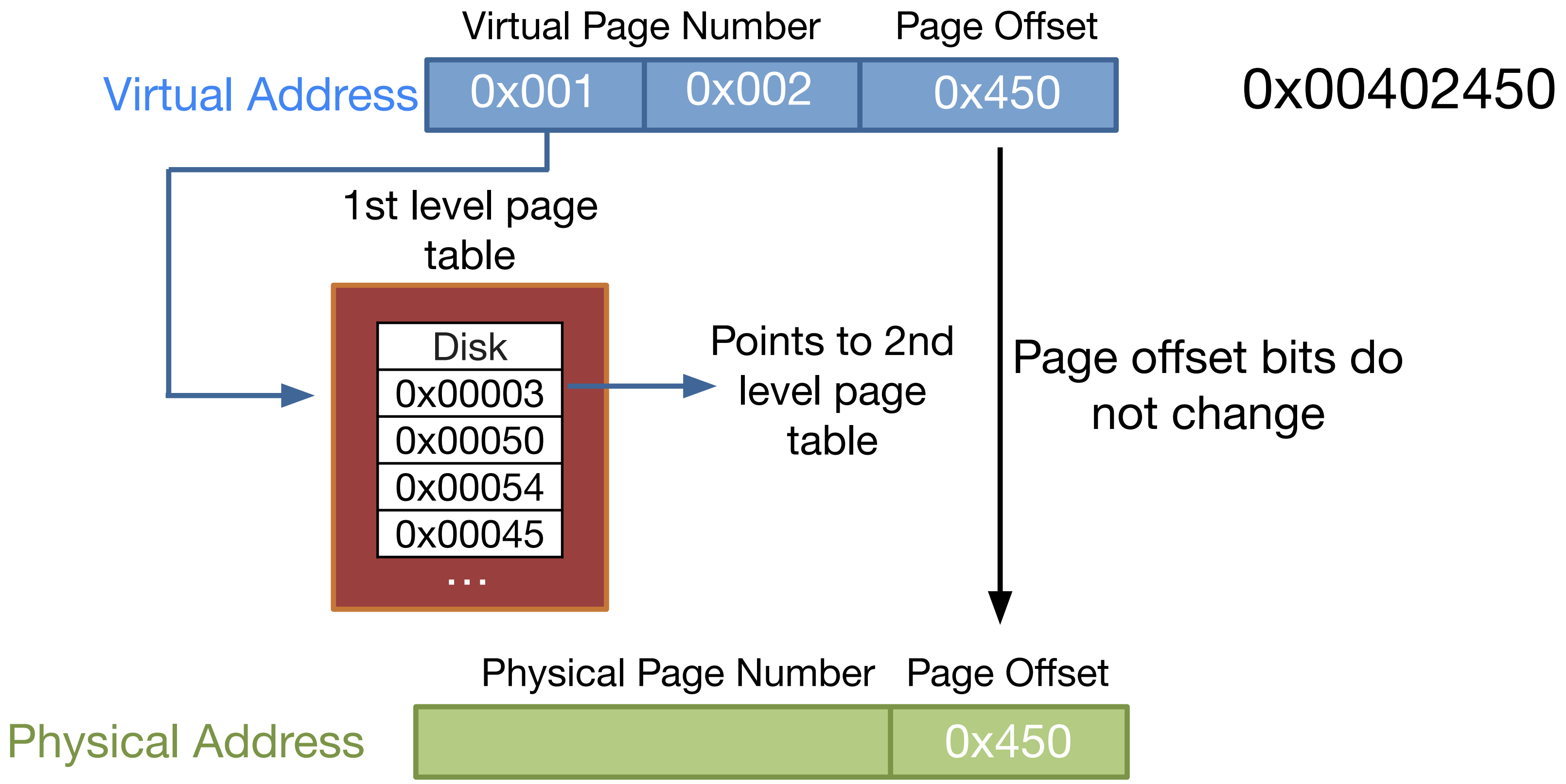


Multi-level Page Table Translation

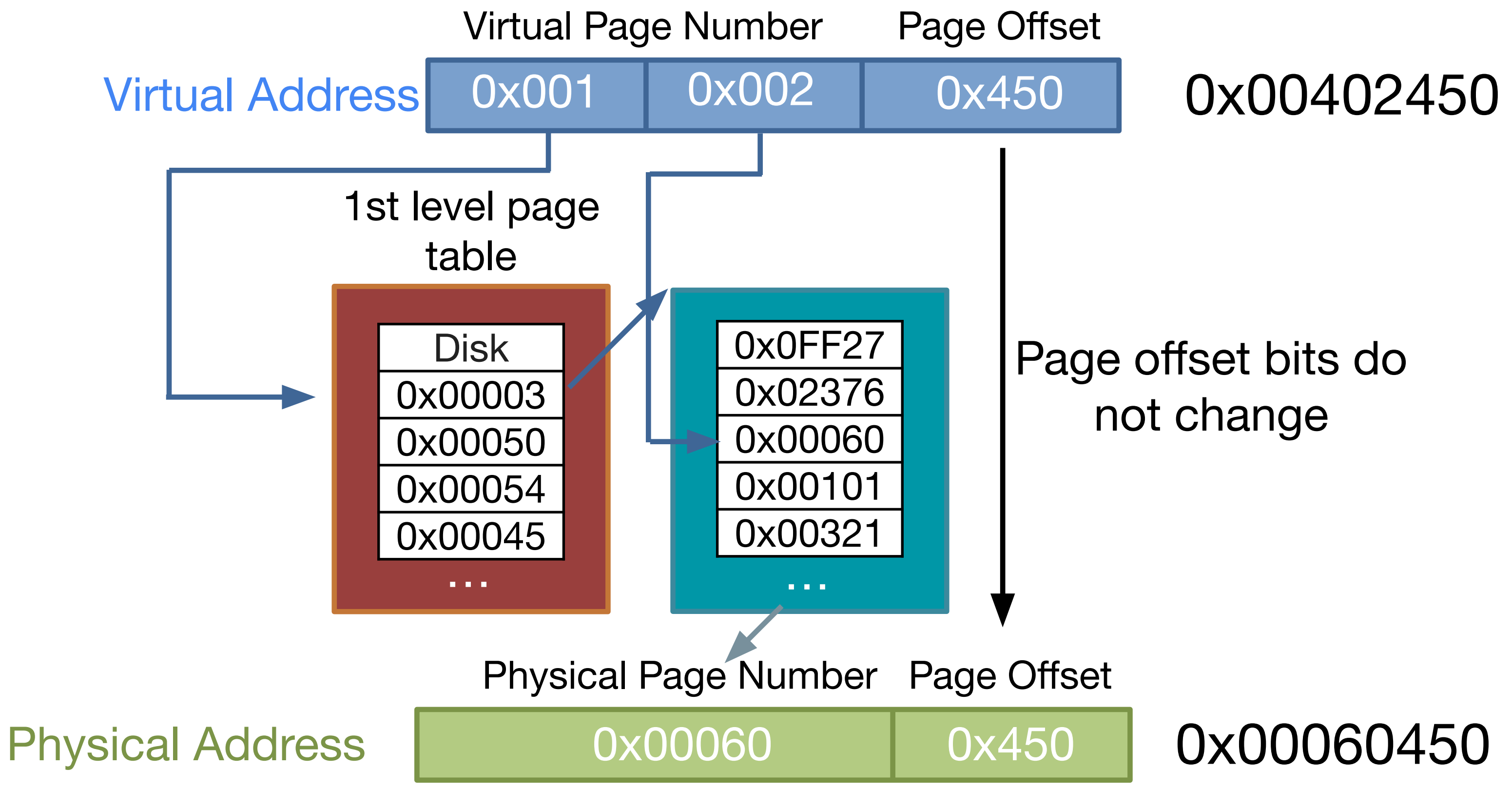
32 bit machine with 4 GB of RAM and 4KB pages



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)
  - 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)
    - # 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

Summary

- Virtual memory's functionalities
    - Give each program the illusion that they have the full address space to themselves
        - Virtual memory can appear to be larger than RAM, with some content actually stored on disk
    - Enables protection: isolated memory between processes
- Ways to increase efficiency for memory translation
    - TLB: a cache for page tables
    - Multi-level page tables: enable some 2nd level page tables to be stored on the disk instead of RAM