18-344 Recitation 7

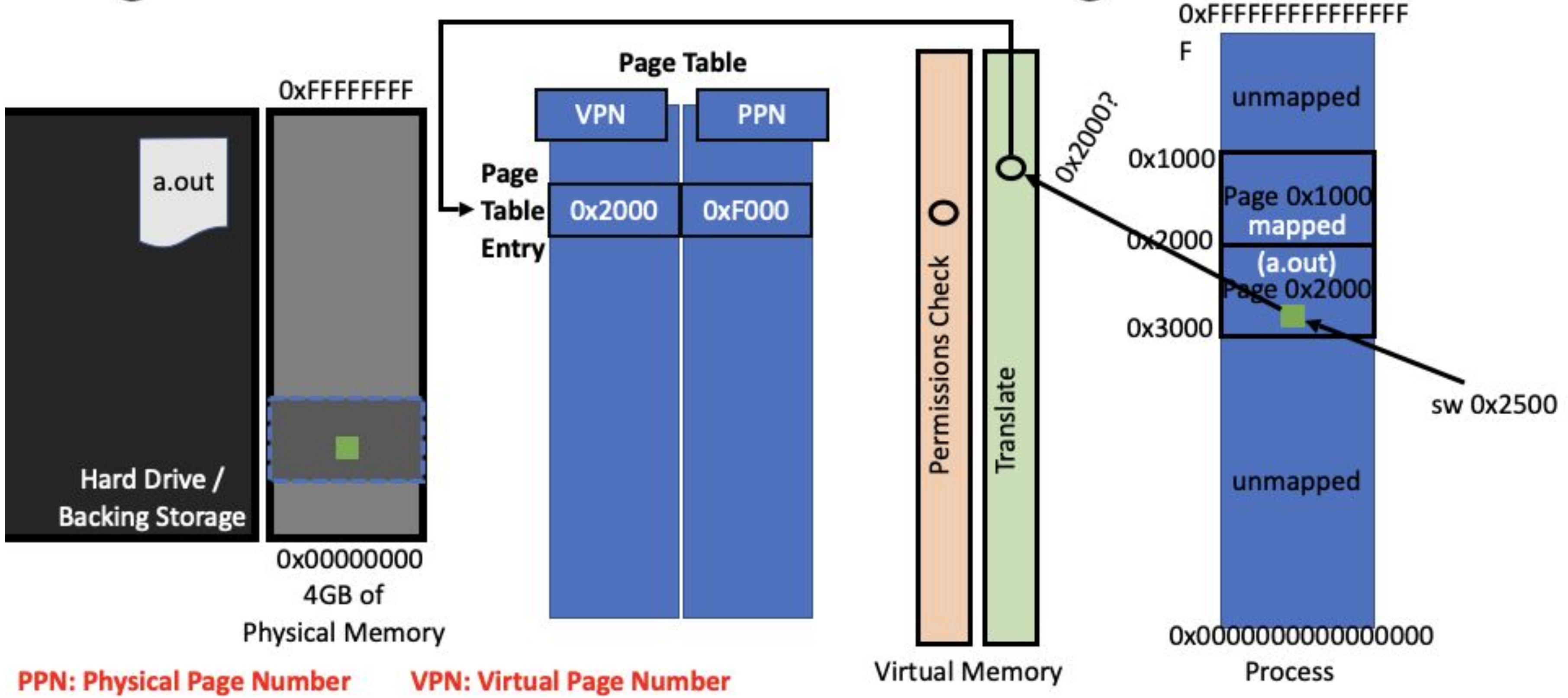
Lab3 - Virtual Memory

Logistical Notes

- HW 6 Released (Due Nov 2nd)
- Lab 3 Released (Due Nov 7th)
- Lab 2 Grades released
- Midterm Grades released (graded out of 61)

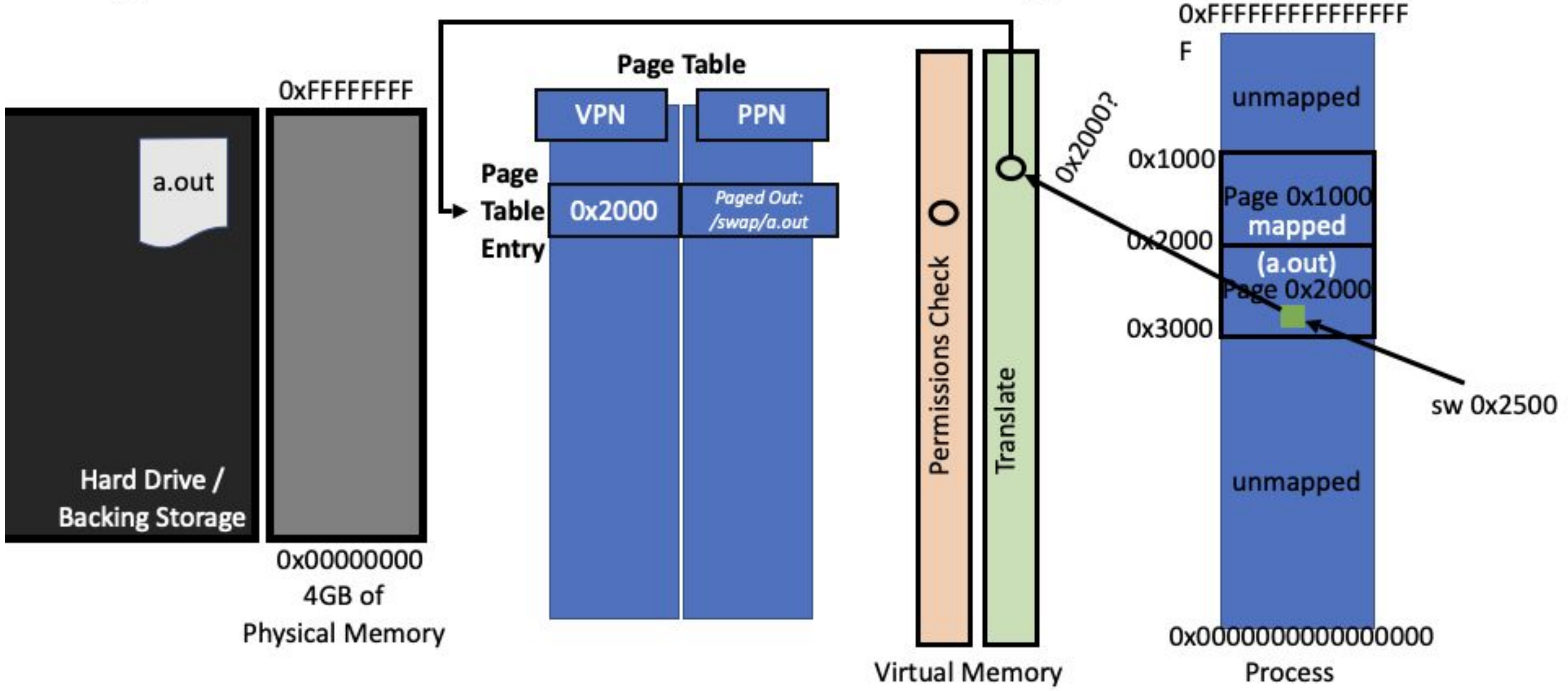
Virtual Memory Overview

Virtual Memory: The Translation Function
Page Table Stores Translation for Paged-In Data

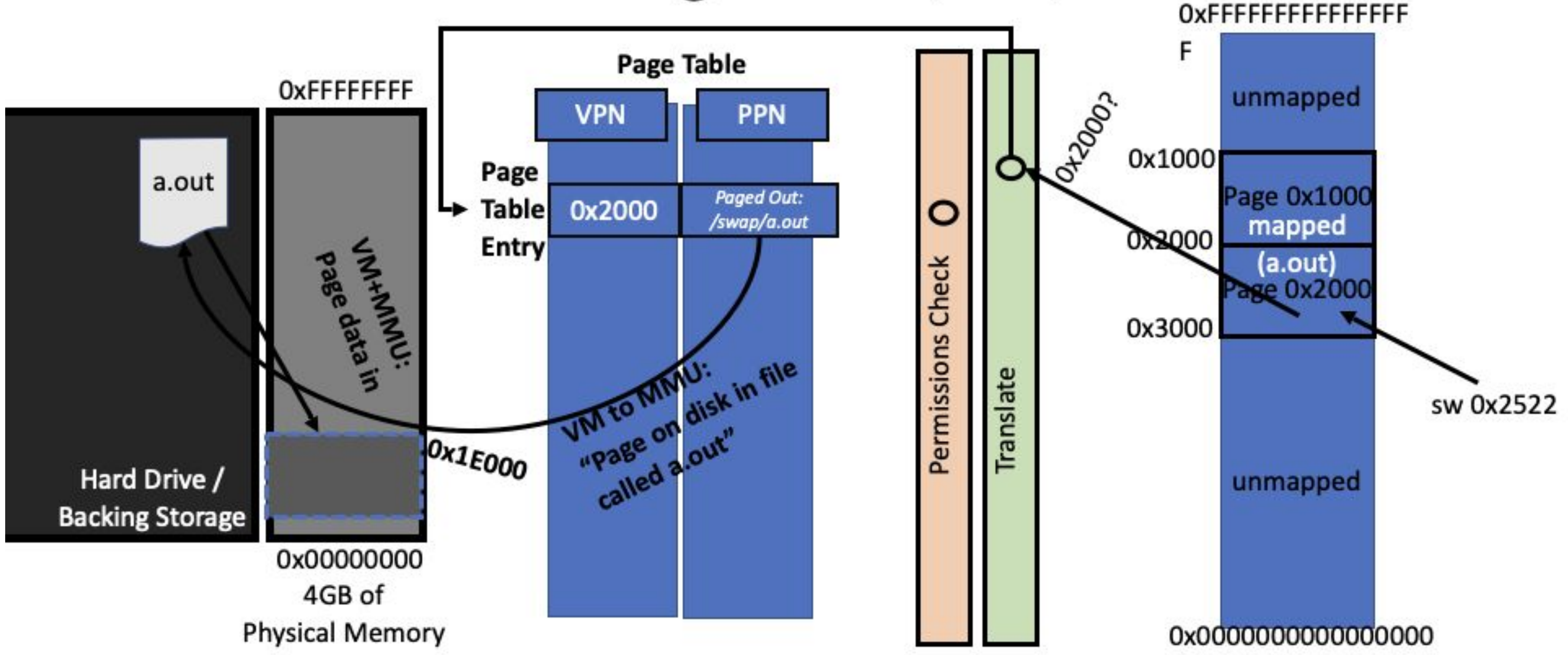


PPN: Physical Page Number **VPN: Virtual Page Number**

Virtual Memory: The Translation Function Page Table Holds Disk Location for Paged-Out Data

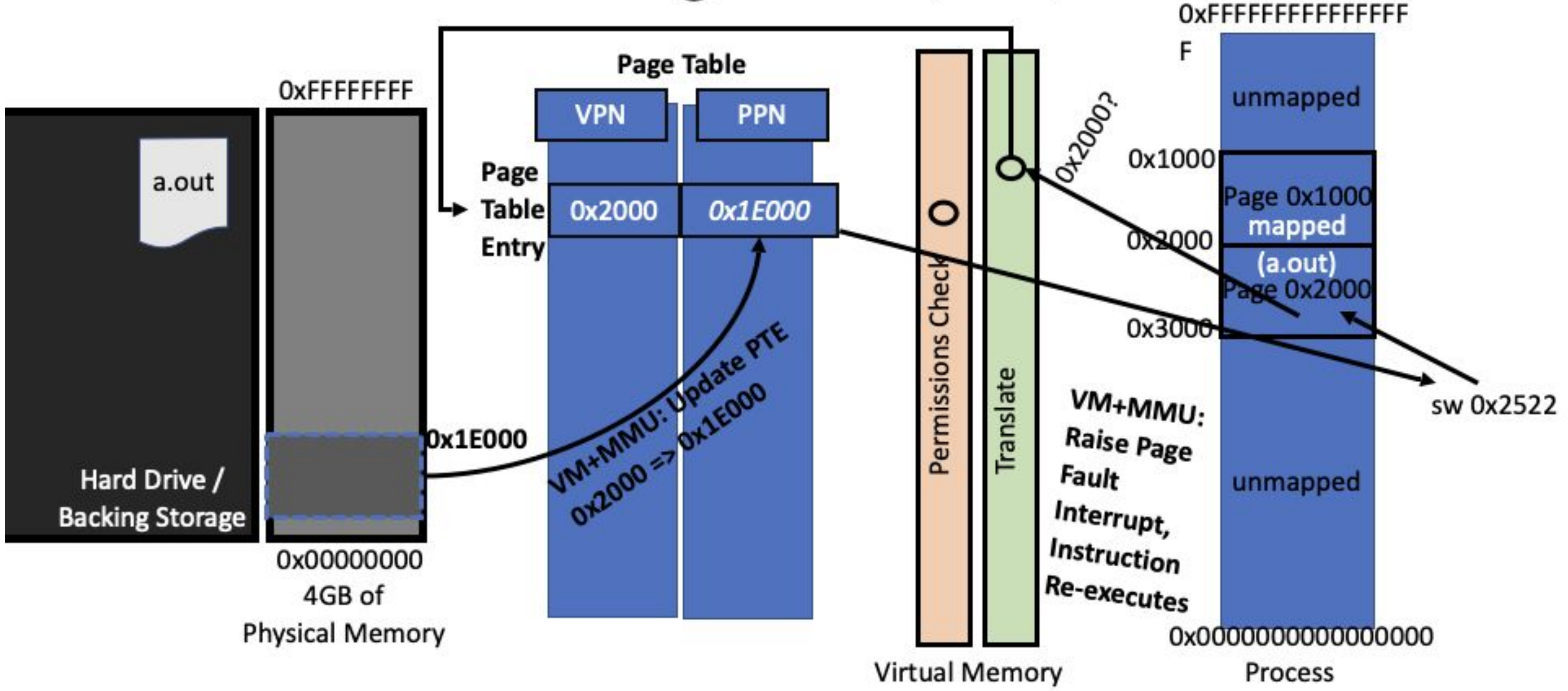


Physical Memory as a Cache of Data on Disk: Cache Miss Means Page Fault (1/2)

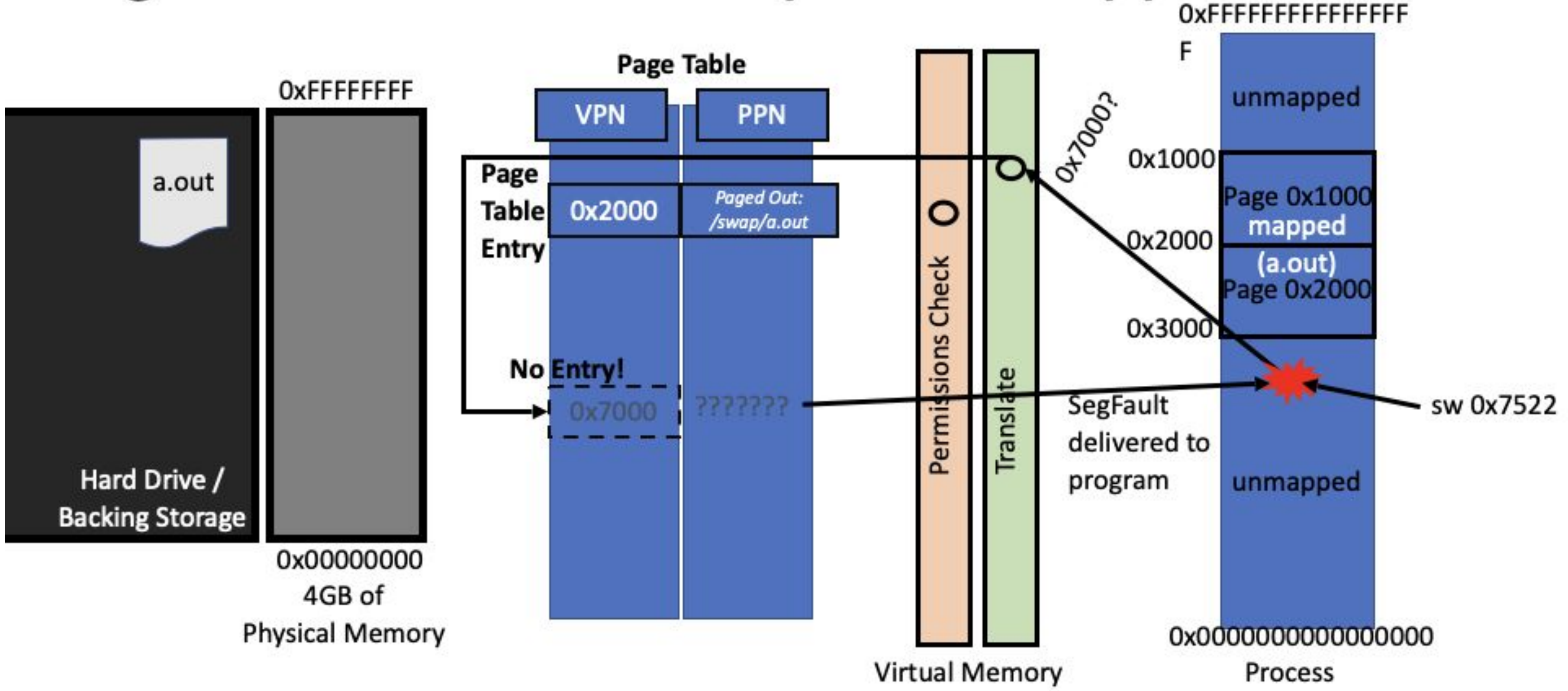


"Demand Paging" – bring into physical memory on first access

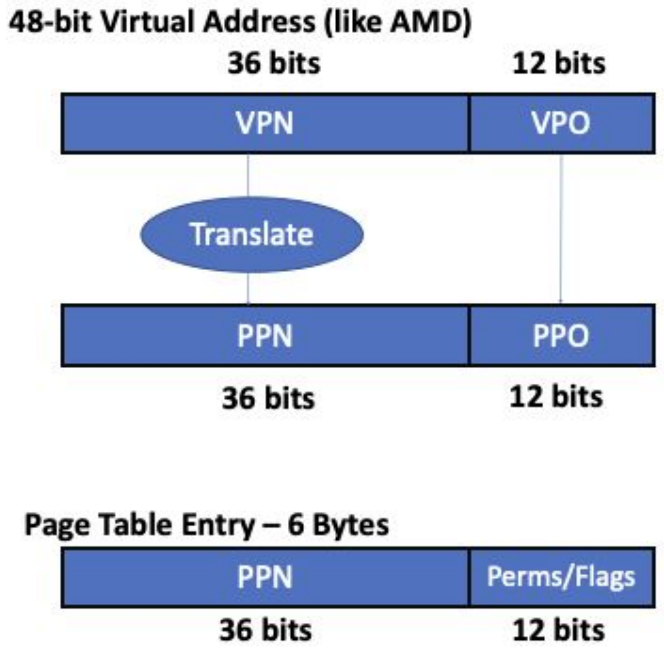
Physical Memory as a Cache of Data on Disk: Cache Miss Means Page Fault (2/2)



Virtual Memory: The Translation Function Page Table Holds No Entry for Unmapped Data



Page Translation and Its Implementation



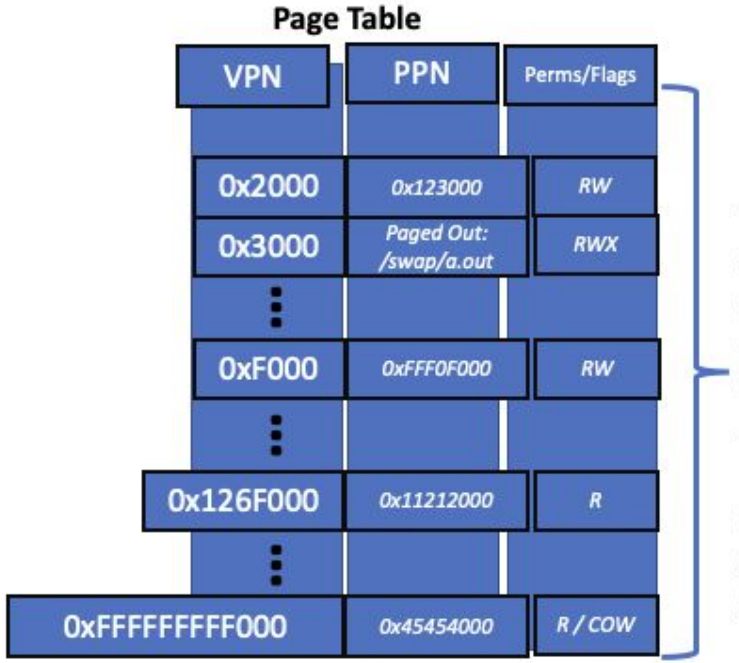
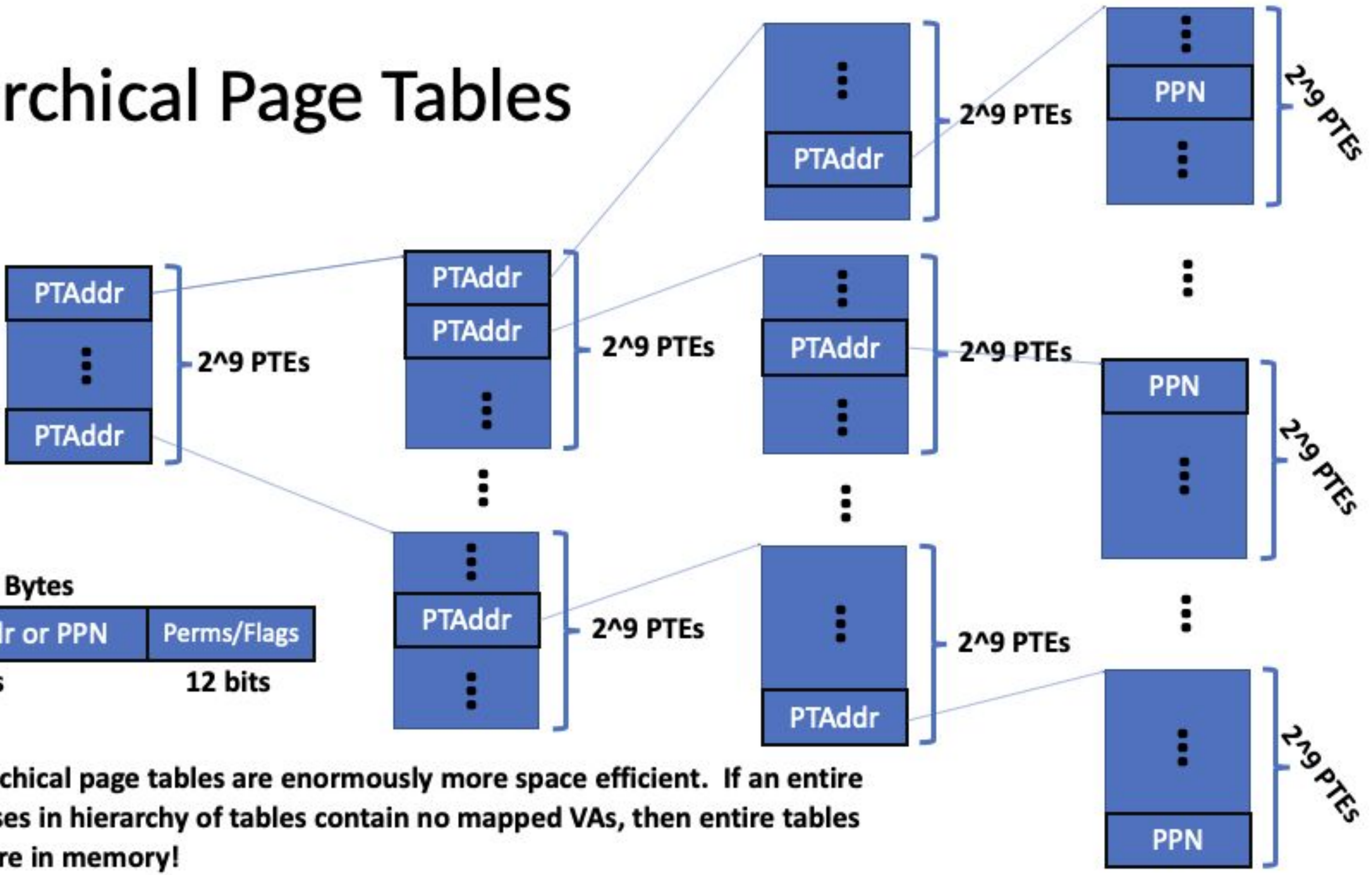


Table stores 2^36 entries in it for virtual pages 0x000000000000 up to 0xFFFFFFFFF000 which span the entire 48-bit address space.

Dense, linear table stores 2^36 * 6B PTEs: 550.8GB of Page Tables

Most PTEs Empty!!!

Hierarchical Page Tables

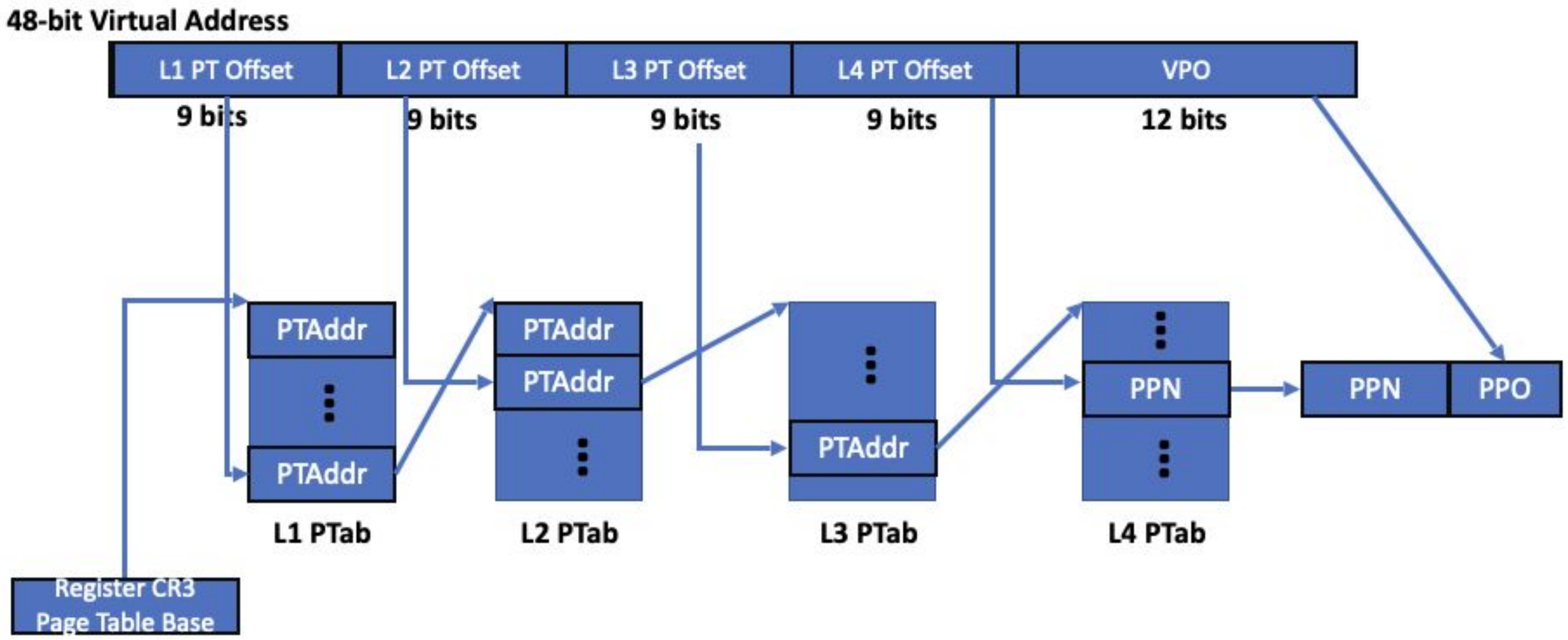


Page Table Entry – 6 Bytes

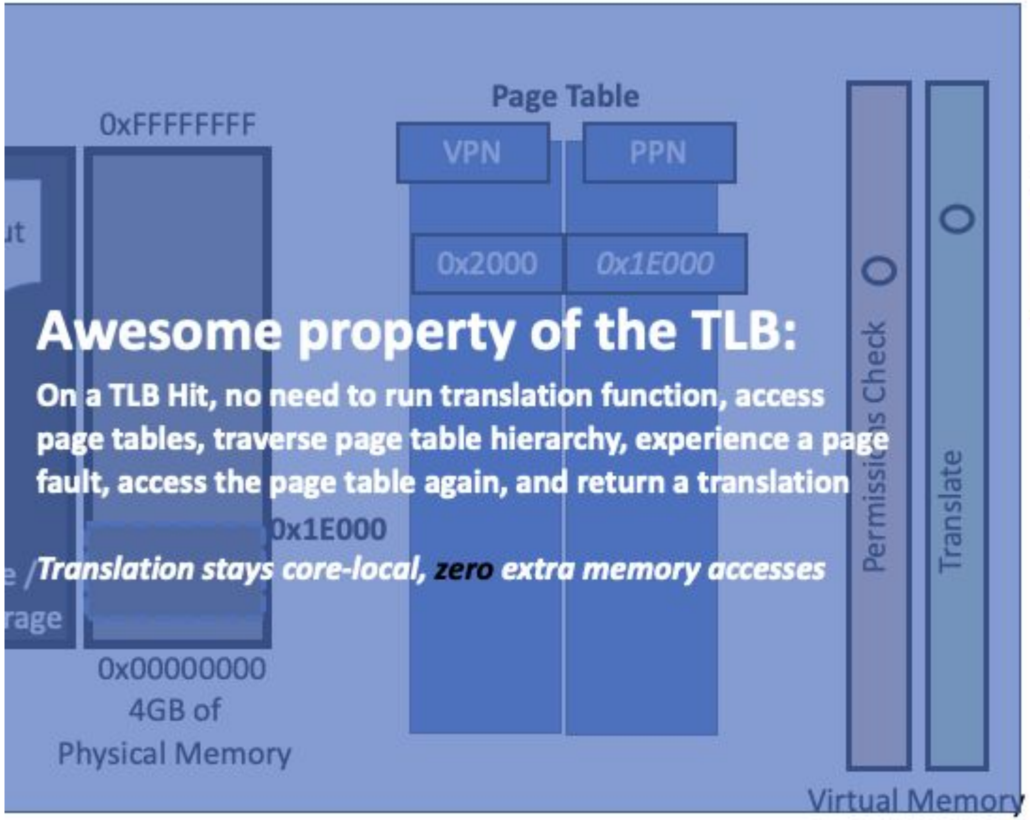
Page Table Addr or PPN	Perms/Flags
36 bits	12 bits

Multi-level / hierarchical page tables are enormously more space efficient. If an entire sub-tree of addresses in hierarchy of tables contain no mapped VAs, then entire tables not stored anywhere in memory!

Translation Using Hierarchical Page Tables



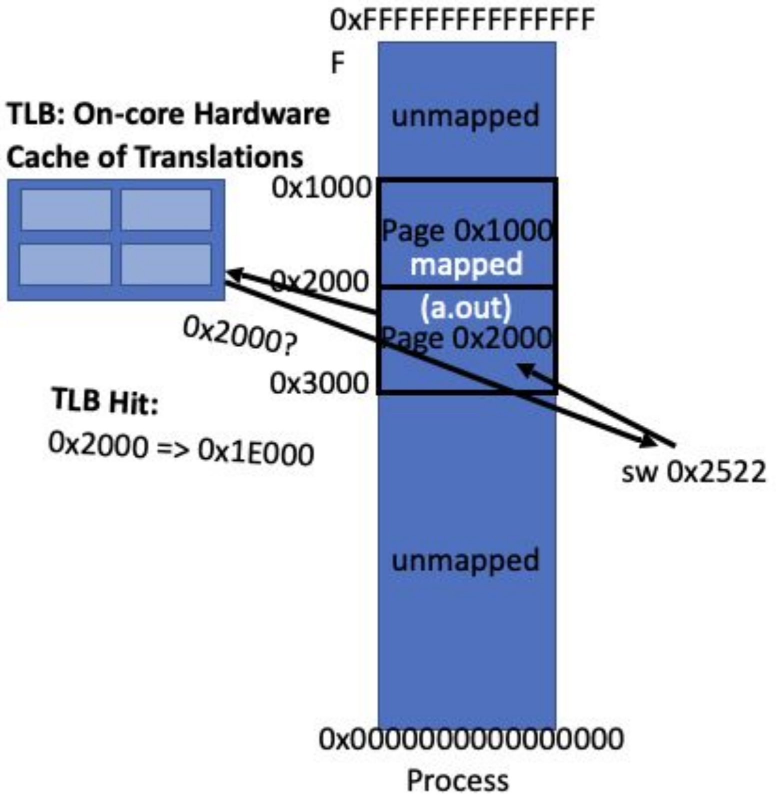
Translation Lookaside Buffer: Basic Idea (Hit)



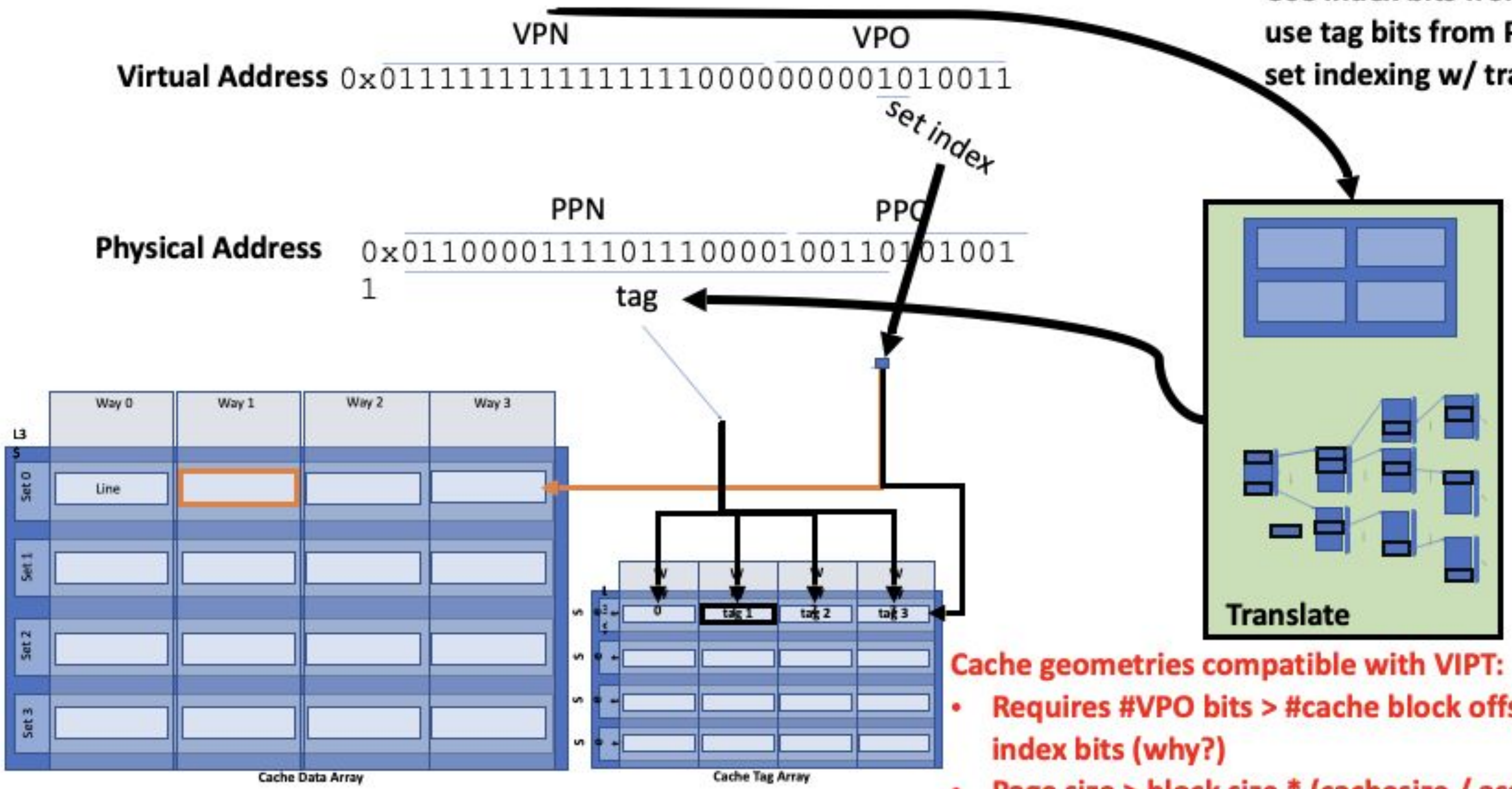
Awesome property of the TLB:

On a TLB Hit, no need to run translation function, access page tables, traverse page table hierarchy, experience a page fault, access the page table again, and return a translation

Translation stays core-local, zero extra memory accesses



Virtually Indexed, Physically Tagged Caches



Use index bits from VPO and use tag bits from PPN. Overlap set indexing w/ translation

Cache geometries compatible with VIPT:

- **Requires #VPO bits > #cache block offset + #cache set index bits (why?)**
- **Page size > block size * (cachesize / assoc = #sets)**

Virtual Memory Lab

Task 1: Implement a Page Table

- Page Table should have a 4-level hierarchy like discussed in class, with 512 entries per table (similar to Intel Core i7).
- You will be responsible for implementing memory mapping, page fault handling and TLB implementation

Task 1: Implement a Page Table

vm-student.cpp/.h: You have to implement three functions:

- vmMap(vaddr, size)
  - Update Page Table to map \`size\` bytes at address \`vaddr\`; Could span multiple pages; create Page Tables or PTEs here if not exist
- vmTranslate(vaddr)
  - TODOs in the handout; Return the translated physical address
- vmPageFaultHandler(*pte)
  - Handle Page Faults (page not residing in memory) here; if there exist free physical pages in memory, allocate them with bumpAllocate() function; if you run out of free pages, replace an existing page with the replacePage() function. These functions return the PPN to store in your PTE.

*You will not be writing the allocation or replacement logic

Task 2: Implement a TLB

tlb.cpp: Implement a TLB structure, with your choice of organization (size, ways, replacement). You will also need two functions: lookup() and update()

- lookup(vaddr, &PPN)
  - Attempt to assign cached-PPN to the argument &PPN, and return true for hits
- update(vaddr, new_PPN)
  - Update the TLB entries with this new mapping. This may evict an existing entry, your TLB organization should account for replacement.

Reporting the results

- Page Table implementation should report #page_faults, #num_accesses and #tlb_hits
- You will justify your TLB organization with a quantitative analysis of these three parameters.
- Show a plot of TLB misses vs TLB configurations
- Reason about the reduction in Page Table Walks due to your TLB
- Your code should also be robust to handle exceptions*: page faults and seg. faults (unmapped accesses).
- Turn in your code, writeup and test traces!