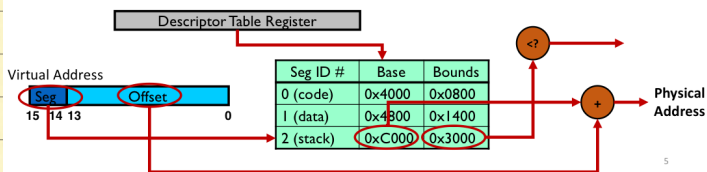


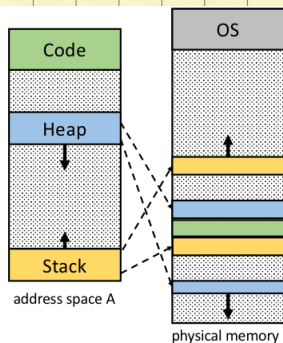
Solutions for Segmentation

- OS context switch must also save and restore all pairs of segment registers
 - Table of base/bounds stored in memory rather than registers
 - Pointer to the table stored in a register
 - Cons:** one more memory read per address translation



DTR中存的是table的物理地址

- A segment may grow, which may or may not be possible
 - Virtually continuous memory can be physically discontinuous



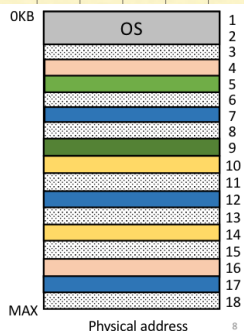
- Management of free spaces of physical memory with variable-sized segments

- Fixed-sized segments**
 - Physical memory carved into fixed sized chunks, **index by an integer**
 - Can use simple vector of bits to handle allocation (1-allocated, 0-free)

1101101011010110

- External fragmentation**

- Fixed-sized segments**
 - Each request is of the same fixed size - always fit, no external fragmentation



分成多个等大的chunk, 用数组维护.

Paging: Put All These Ideas Together

- Physical memory conceptually divided into fixed size
 - Each is called a **page frame**
 - Typical size 1KB to 16KB, most ISA uses 4KB
- Virtual address space conceptually divided into the same size
 - Each is called a **page**
- Page mapped to page frame
 - One to one mapping
 - Many to one mapping -> memory sharing
- One page table per process
 - Resides in physical memory
 - One entry for one virtual->physical translation

How big is one page (frame)?

- Too big -> internal fragmentation
- Too small -> page table too big

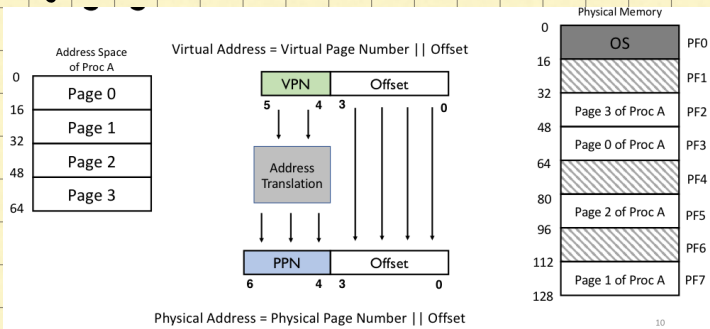
物理地址分为多个等大的页帧

虚拟地址分为多个等大的页

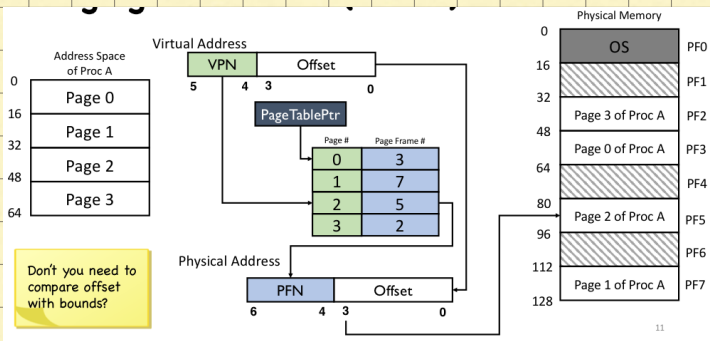
页映射到页帧 (体现于页表中)

一个进程有一个页表.

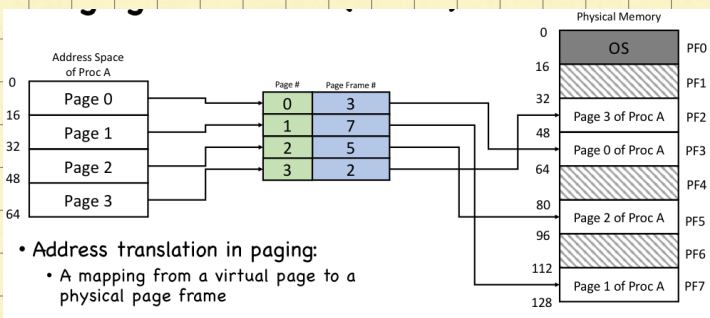
Paging Illustrated



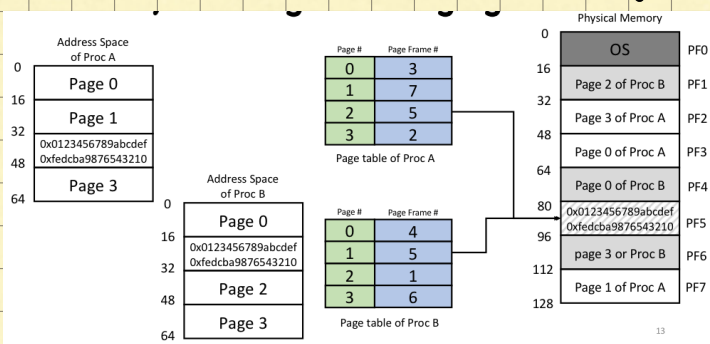
因为 offset 没有超出 bounds, 故直接拿过来.



这个翻译是CPU做的.

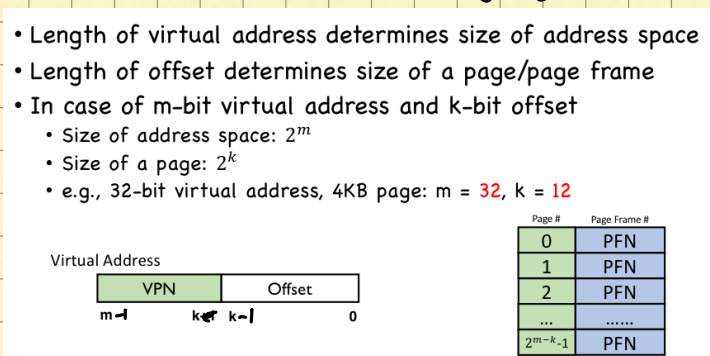


Memory Sharing with Paging



有同一个映射.

Virtual Address and Paging



Page# 实际上不在 PTE 中.

Page Table Entry

- An entry in the page table is called a page table entry (PTE).
- Besides **PFN**, PTE also contains a **valid bit**
 - Virtual pages with no valid mapping: valid bit = 0
 - Important for sparse address space (e.g., 2^{64} bytes)
- PTE also contains **protection bits**
 - Permission to read from or write, or execute code on this page
- PTE also contains an **access bit**, a **dirty bit**, a **present bit**
 - Present bit: whether this page is in physical memory or on disk
 - Dirty bit: whether the page has been modified since it was brought into memory
 - Access bit: whether a page has been accessed

protection bits有三个bit, 分别给读、写、执行。

物理内存不够用时, page会映射到disk上. 此时 present bit 为 0

How Big are Page Tables

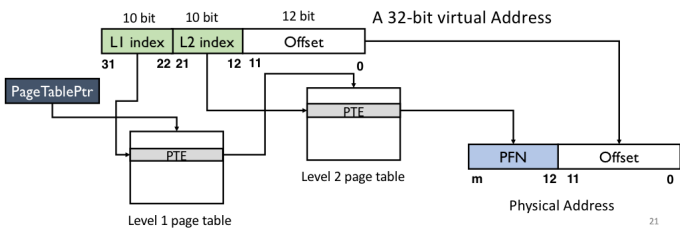
- Assume 32-bit machine and 4KB pages, and each PTE takes 4 bytes (PFN plus status bits)
 - Number of PTEs: $2^{32-12} = 2^{20}$
 - Size of page table: 4MB
 - What if 100 processes running? 400MB
- Page tables are stored in memory and context switch only changes the pointer to page table (e.g., CR3 register)

PageTablePtr		
Page #	Page Frame #	status bits
0	PFN	V,WR
1	PFN	V,E
2	PFN	V,R
...
$2^m - 1$	PFN	I

页表太大会存不下, 需要压缩.

Two-level Page Tables

- Use two levels of page tables to save space

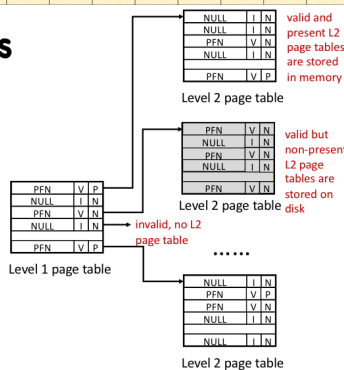


这里一个 page table 大小为 一个 page (2^{12} byte) 刚好有 2^{10} 个 PTE (因为一个 PTE 为 4 byte 大小) index 长为 10 bit 刚好分完一个 page table.

A Tree of Page Tables

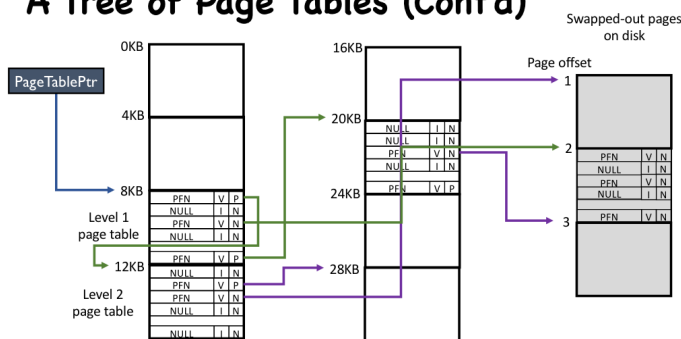
A Tree of Page Tables

- A page table also occupies a page frame
 - e.g., 4KB page and 4-byte PTE: 1024 entries per table
- Two-level page table forms a tree of page tables
 - In theory 1024 level-2 page tables, but only a subset are valid
 - A subset of valid page tables are stored in memory (others on disk)



二级页表是树型结构. 当第一级中 entry 无效时不会指向第二级页表.

A Tree of Page Tables (Cont'd)



Page Table Structures

- Hierarchical page tables
 - 2-level page tables
 - 3-level page tables
 - 4-level page tables
- Hashed Page Tables
- Inverted Page Tables

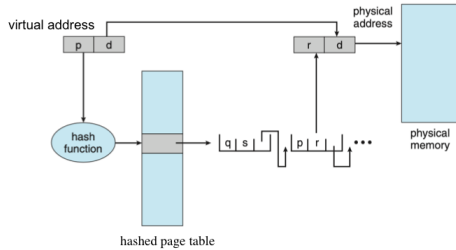
Hashed Page Tables

• Hash function

- input: VPN
- output: index in the hashed page table

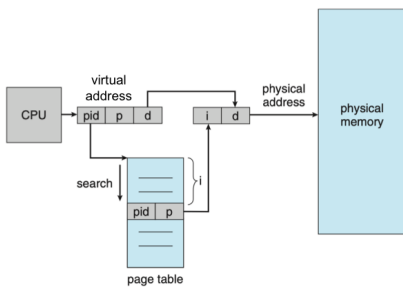
• Collision handling

- A linked list
- Each element consists of three fields: (1) VPN, (2) PFN, and (3) pointer to the next element



Inverted Page Tables

- One page table for the whole system
 - Used in 64-bit UltraSPARC and PowerPC
- Each entry corresponds to one physical page frame
 - Process ID and VPN
- Page table lookup requires linear search of the entire table
- Memory sharing is hard



一个系统一个 inverted Page Table 就可以了。
进程号和VPN定义一个entry.

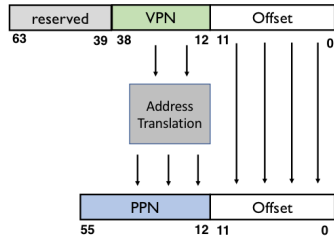
Virtual Memory on RISC-V

• RISC-V supports multiple MMU

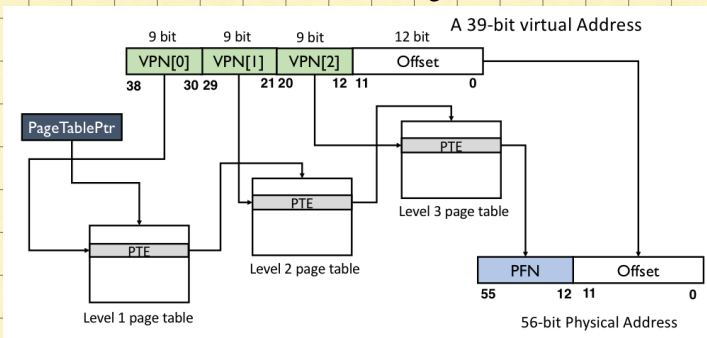
- For RV32: SV32
- For RV64: SV39 and SV48

• Here we introduce SV39

- Page size: 4KB
- Virtual address: 39 bits
 - remaining bits of total 64 bits reserved
- Physical address: 56 bits



SV39: Three Levels of Page Tables



- 一个PTE 8byte (64 bit)

page size: 4KB

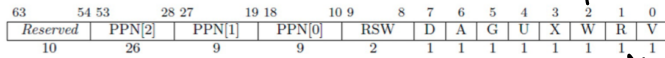
page table 与 page 等大.

那么共有 $\frac{4KB}{8byte} = 2^9$ 个PTE

SV48比SV39多了一个VPN[3]

SV39: Page Table Entries

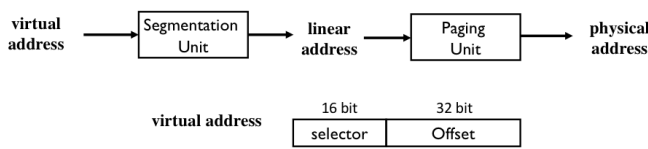
- A PTE in SV39 takes 8 bytes (64 bits)
 - Bit 8-0 status bits
 - Bit 53-10 is PFN (or physical page number, PPN)
 - Bits 63-54 reserved
- The 9 status bits
 - D (dirty), A (accessed), V (valid)
 - G (Global): G=1 the page is mapped in all address spaces
 - U (User): U-mode code may access this page
 - RSW: reserved for s-mode



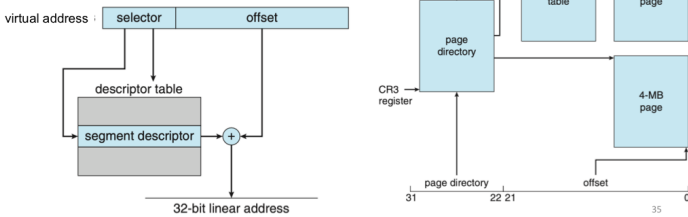
writable
↓
executable
↓
readable

Virtual Memory on IA-32

- Intel's 32-bit CPU (IA-32) uses two stage address translation: segmentation + paging
- A virtual address contains a 16-bit segment selector and 32-bit offset



- Two descriptor tables: GDT & LDT
- Six segment register to cache segment base addresses

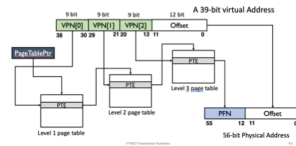


有的软件要求较大的连续物理内存,此时就需要分配一个 huge page.

Issues of Paging

- Time complexity
 - Extra memory references during address translation
 - Three-level page tables requires 3 additional memory reads
 - If every memory reference needs 4 memory reads

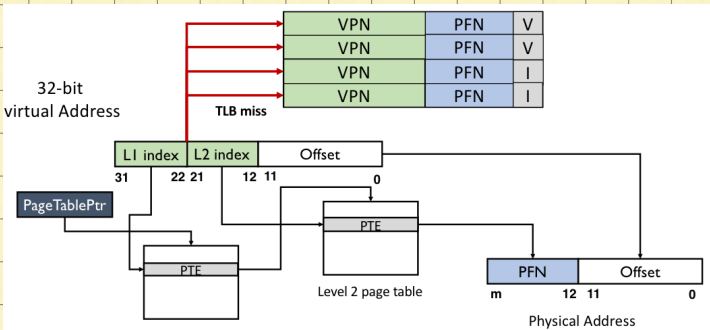
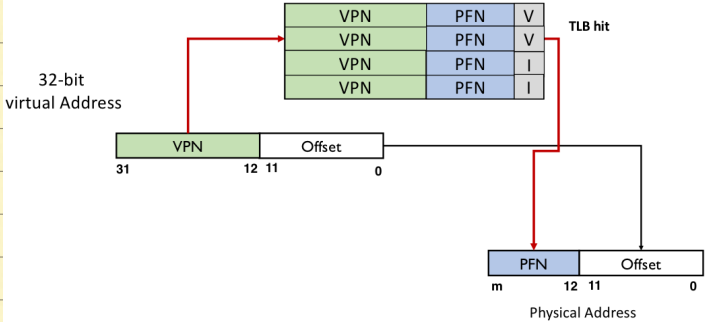
Question:
• How to speed up address translation and avoid extra memory reads?



Translation Lookaside Buffer

- A translation lookaside buffer (TLB) is a hardware cache that is part of the MMU
 - A cache for the PTEs: holding a translation likely to be re-used
 - Replacement policy: LRU, FIFO, random
 - Each entry holds mapping of a virtual address to a physical address
- Before a virtual-to-physical address translation is to be performed, TLB is looked up using VPN
 - TLB hit: VPN is found, and the PFN of the same entry used
 - TLB miss: VPN not found, page table walk

TLB是PTE的cache



Why Does TLB Work?

Why Does TLB Work?

- Ideally one page table walk for the entire page
 - TLB is smaller than page table, but
- Spatial locality
 - Sequentially executed instructions, local variables (on stack), arrays (on heap) likely on the same page
 - E.g., accessing a[0] to a[9]
 - m, h, h, m, h, h, h, m, h, h → 70% hit rate
 - Large page size increases hit rate
- Temporal locality
 - Accesses to the same page tend to be close in time

VPN	Offset
VPN = 00	
VPN = 01	
VPN = 02	
VPN = 03	
VPN = 04	
VPN = 05	
VPN = 06	a[0] a[1] a[2]
VPN = 07	a[3] a[4] a[5] a[6]
VPN = 08	a[7] a[8] a[9]
VPN = 09	
VPN = 10	
VPN = 11	
VPN = 12	
VPN = 13	
VPN = 14	
VPN = 15	

Issues with Context Switch

- Two process may use the same virtual address
 - P1: 100 → 110
 - P2: 100 → 170
- Solutions
 - Flush TLB upon context switch
 - Invalidate all entries: V → I
 - Extending TLB with address space ID
 - No need to flush tlb

VPN	PFN	valid
-	-	I
100	110	V
-	-	I
100	170	V

VPN	PFN	valid	ASID
-	-	I	-
100	110	V	1
-	-	I	-
100	170	V	2