15-213

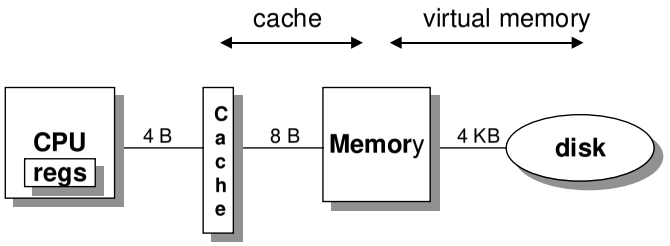
Virtual Memory

October 27, 1998

Topics

- Motivation
- Address Translation
- Accelerating with TLBs
- Alpha 21X64 memory system

Levels in Memory Hierarchy



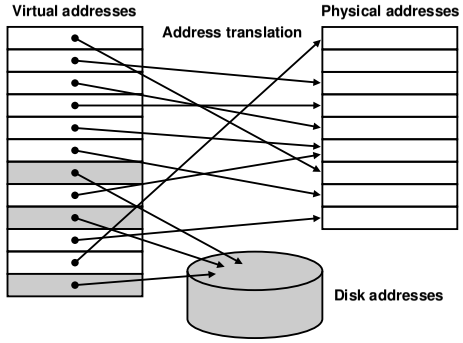
	Register	Cache	Memory	Disk Memory
size:	200 B	32 KB / 4MB	128 MB	20 GB
speed:	3 ns	6 ns	100 ns	10 ms
$/Mbyte:		$100/MB	$1.50/MB	$0.06/MB
block size:	4 B	8 B	4 KB	

larger, slower, cheaper

Virtual Memory

Classically

- Main memory acts as a cache for the secondary storage (disk)

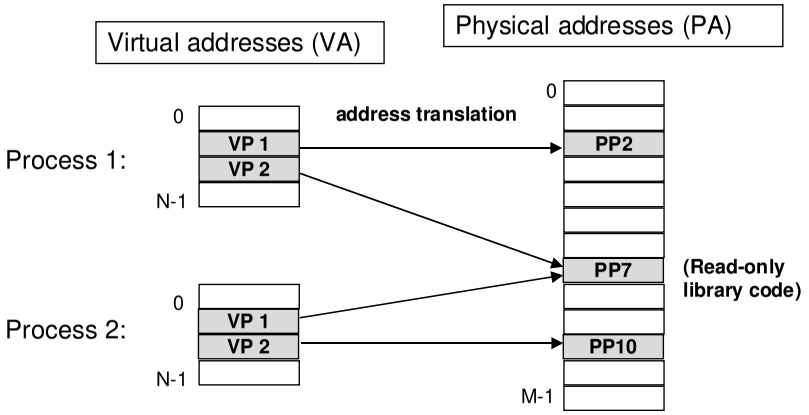


Increases Program-Accessible Memory

- address space of each job larger than physical memory
- sum of the memory of many jobs greater than physical memory

Address Spaces

- Virtual and physical address spaces divided into equal-sized blocks
  - "Pages" (both virtual and physical)
- Virtual address space typically larger than physical
- Each process has separate virtual address space



Other Motivations

Simplifies memory management

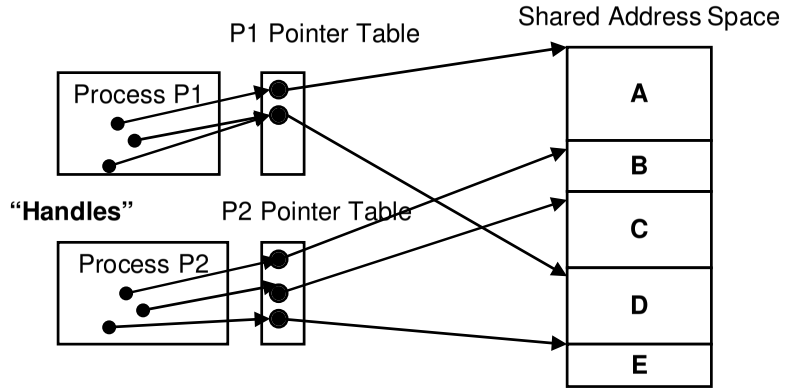
- **Main reason today**
- **Can have multiple processes resident in physical memory**
- **Their program addresses mapped dynamically**
  - Address `0x100` for process P1 doesn't collide with address `0x100` for process P2
- **Allocate more memory to process as its needs grow**

Provides Protection

- **One process can't interfere with another**
  - Since operate in different address spaces
- **Process cannot access privileged information**
  - Different sections of address space have different access permissions

Contrast: Macintosh Memory Model

Does not Use Traditional Virtual Memory



All program objects accessed through "handles"

- **Indirect reference through pointer table**
- **Objects stored in shared global address space**

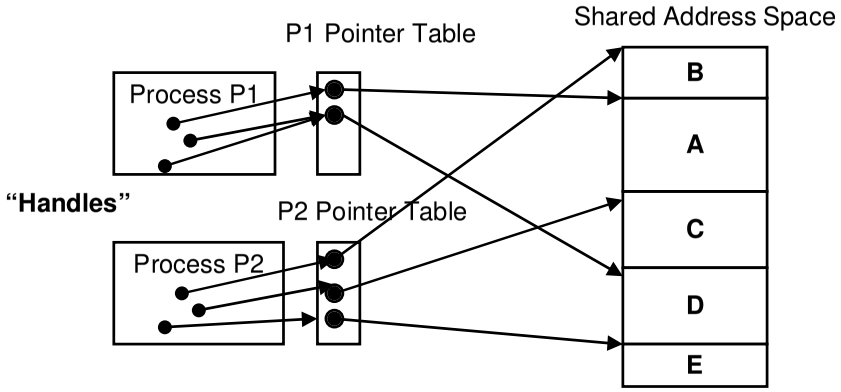
Macintosh Memory Management

Allocation / Deallocation

- **Similar to free-list management of malloc/free**

Compaction

- **Can move any object and just update the (unique) pointer in pointer table**



Macintosh vs. VM-based Mem. Mgmt

Both

- **Can allocate, deallocate, and move memory blocks**

Macintosh

- **Block is variable-sized**
  - May be very large or very small
- **Requires contiguous allocation**
- **No protection**
  - "Wild write" by one process can corrupt another's data

VM-Based

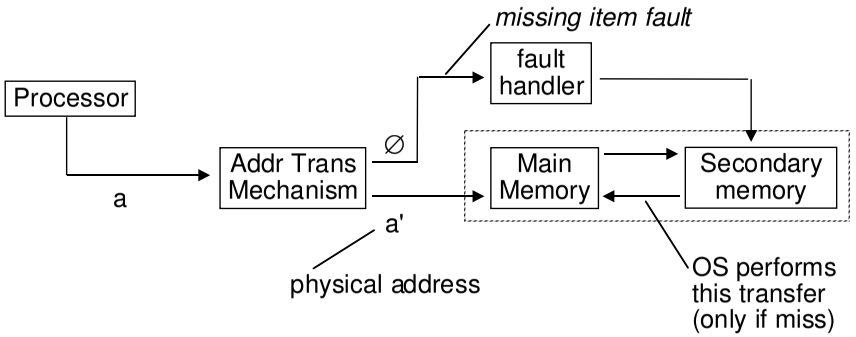
- **Block is fixed size**
  - Single page
  - Can map contiguous range of virtual addresses to disjoint ranges of physical addresses
- **Provides protection**
  - Between processes
  - So that process cannot corrupt OS information

VM Address Translation

V = {0, 1, . . . , N–1} virtual address space
P = {0, 1, . . . , M–1} physical address space
N > M

MAP: V → P U {∅} address mapping function

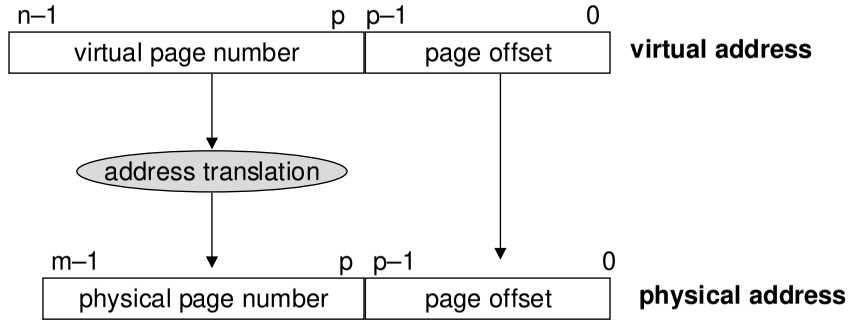
MAP(a) = a' if data at virtual address a is present at physical address a' in P
= ∅ if data at virtual address a is not present in P



VM Address Translation

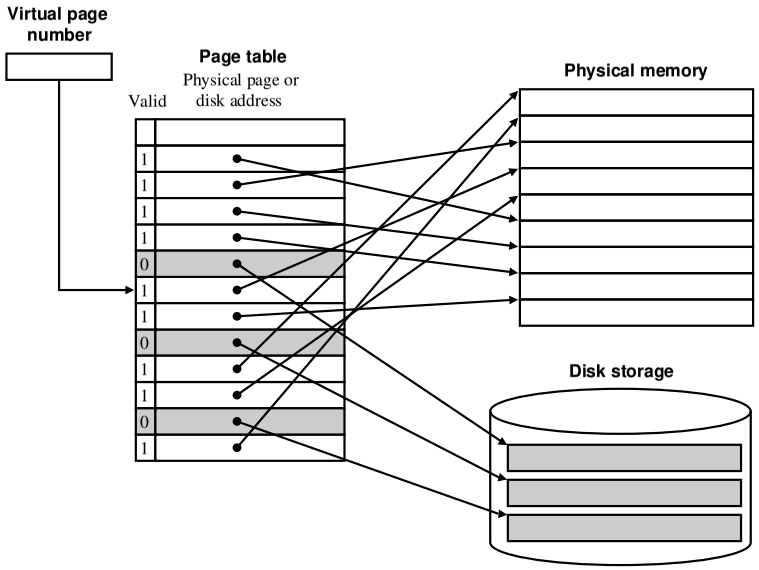
Parameters

- **P = 2^p = page size (bytes). Typically 1KB–16KB**
- **N = 2^n = Virtual address limit**
- **M = 2^m = Physical address limit**

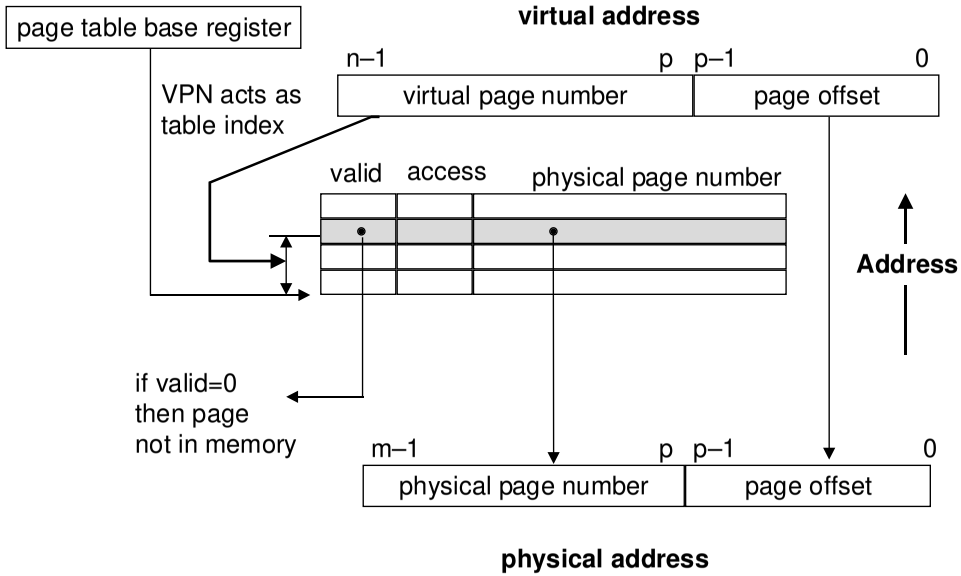


Notice that the page offset bits don't change as a result of translation

Page Tables



Address Translation via Page Table



Page Table Operation

Translation

- **Separate (set of) page table(s) per process**
- **VPN forms index into page table**

Computing Physical Address

- **Page Table Entry (PTE) provides information about page**
  - Valid bit = 1 ==> page in memory.
    - » Use physical page number (PPN) to construct address
  - Valid bit = 0 ==> page in secondary memory
    - » Page fault
    - » Must load into main memory before continuing

Checking Protection

- **Access rights field indicate allowable access**
  - E.g., read-only, read-write, execute-only
  - Typically support multiple protection modes (e.g., kernel vs. user)
- **Protection violation fault if don't have necessary permission**

VM design issues

Everything Driven by Enormous Cost of Misses:

- **Hundreds of thousands to millions of clocks.**
  - vs units or tens of clocks for cache misses.
- **Disks are high latency**
  - Typically 10 ms access time
- **Moderate disk to memory bandwidth**
  - 10 MBytes/sec transfer rate

Large Block Sizes:

- **Typically 1KB–16 KB**
- **Amortize high access time**
- **Reduce miss rate by exploiting spatial locality**

Perform Context Switch While Waiting

- **Memory filled from disk by direct memory access**
- **Meanwhile, processor can be executing other processes**

Disk / System Interface

Processor Signals Controller

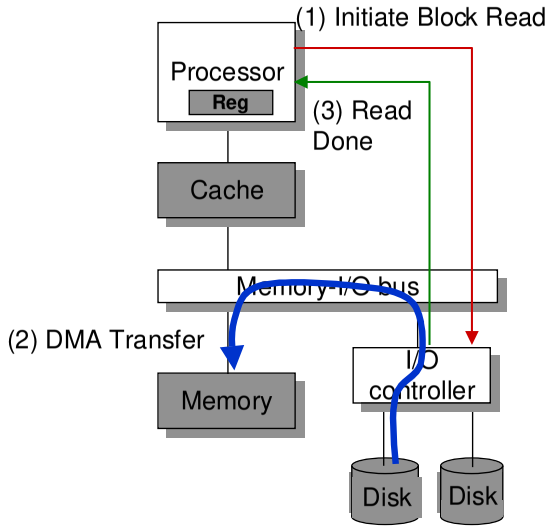
- **Read block of length P starting at disk address X and store starting at memory address Y**

Read Occurs

- **Direct Memory Access**
- **Under control of I/O controller**

I / O Controller Signals Completion

- **Interrupt processor**
- **Can resume suspended process**



VM design issues (cont)

Fully Associative Page Placement

- **Eliminates conflict misses**
- **Every miss is a killer, so worth the lower hit time**

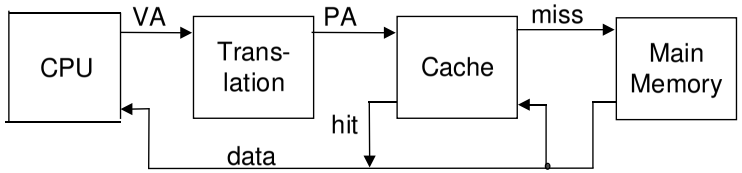
Use Smart Replacement Algorithms

- **Handle misses in software**
  - Plenty of time to get job done
  - Vs. caching where time is critical
- **Miss penalty is so high anyway, no reason to handle in hardware**
- **Dmall improvements pay big dividends**

Write Back Only

- **Disk access too slow to afford write through**

Integrating VM and cache



Most Caches "Physically Addressed"

- **Accessed by physical addresses**
- **Allows multiple processes to have blocks in cache at same time**
- **Allows multiple processes to share pages**
- **Cache doesn't need to be concerned with protection issues**
  - Access rights checked as part of address translation

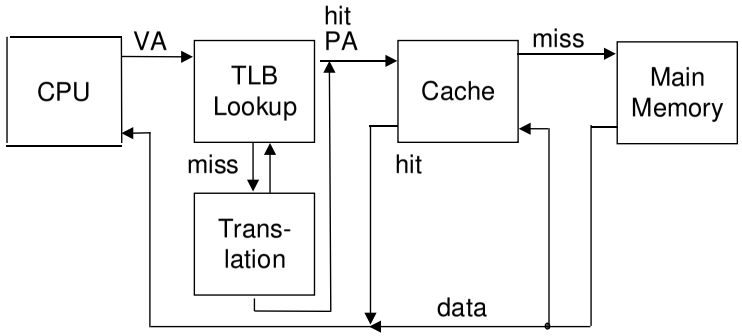
Perform Address Translation Before Cache Lookup

- **But this could involve a memory access itself**
- **Of course, page table entries can also become cached**

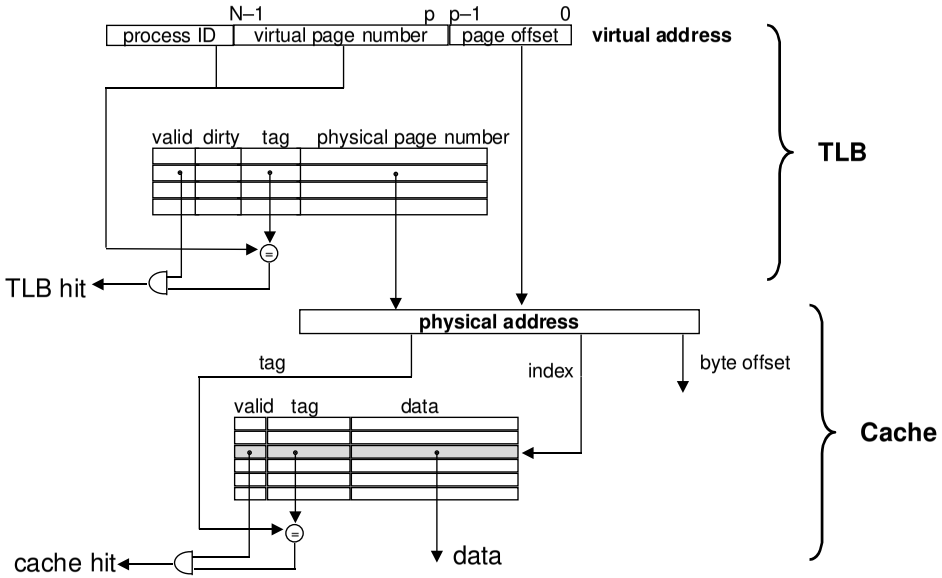
Speeding up Translation with a TLB

Translation lookaside buffer (TLB)

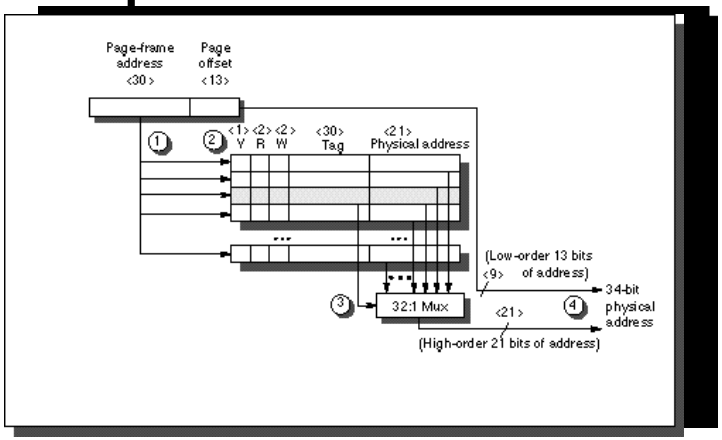
- **Small, usually fully associative cache**
- **Maps virtual page numbers to physical page numbers**
- **Contains complete page table entries for small number of pages**



Address translation with a TLB



Alpha AXP 21064 TLB



- page size: **8KB**
- hit time: **1 clock**
- miss penalty: **20 clocks**
- TLB size: **ITLB 8 PTEs, DTLB 32 PTEs**
- placement: **Fully associative**

TLB-Process Interactions

TLB Translates Virtual Addresses

- But virtual address space changes each time have context switch

Could Flush TLB

- Every time perform context switch
- Refill for new process by series of TLB misses
- ~100 clock cycles each

Could Include Process ID Tag with TLB Entry

- Identifies which address space being accessed
- OK even when sharing physical pages

Alpha Physical Addresses

Model	Bits	Max. Size
• 21064	34	16GB
• 21164	40	1TB
• 21264	44	16TB

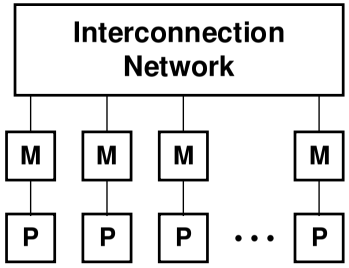
Why a 1TB (or More) Address Space?

- At $1.00 / MB, would cost $1 million for 1 TB
- Would require 131,072 memory chips, each with 256 Megabits
- Current uniprocessor models limited to 2 GB

Massively-Parallel Machines

Example: Cray T3E

- Up to 2048 Alpha 21164 processors
- Up to 2 GB memory / processor
- 8 TB physical address space!



Logical Structure

- Many processors sharing large global address space
- Any processor can reference any physical address
- VM system manages allocation, sharing, and protection among processors

Physical Structure

- Memory distributed over many processor modules
- Messages routed over interconnection network to perform remote reads & writes

Alpha Virtual Addresses

Page Size

- Currently 8KB

Page Tables

- Each table fits in single page
- Page Table Entry 8 bytes
  - 4 bytes: physical page number
  - Other bytes: for valid bit, access information, etc.
- 8K page can have 1024 PTEs

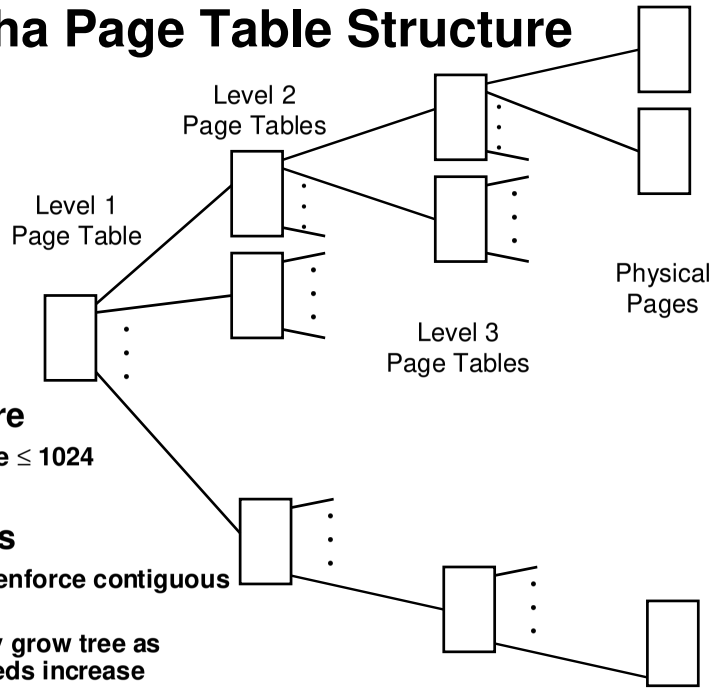
Alpha Virtual Address

- Based on 3-level paging structure

level 1	level 2	level 3	page offset
10	10	10	13

- Each level indexes into page table
- Allows 43-bit virtual address when have 8KB page size

Alpha Page Table Structure



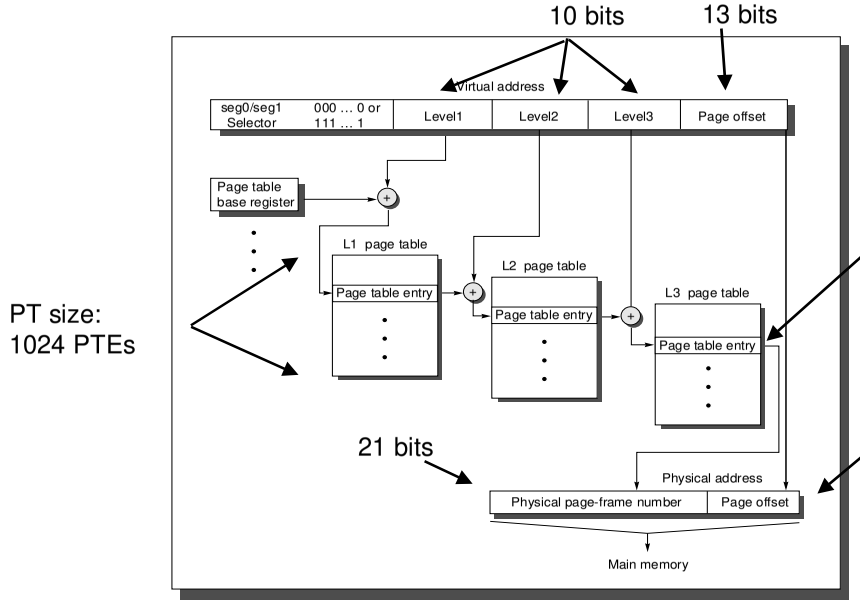
Tree Structure

- **Node degree ≤ 1024**
- **Depth 3**

Nice Features

- **No need to enforce contiguous page layout**
- **Dynamically grow tree as memory needs increase**

Mapping an Alpha 21064 Virtual Address



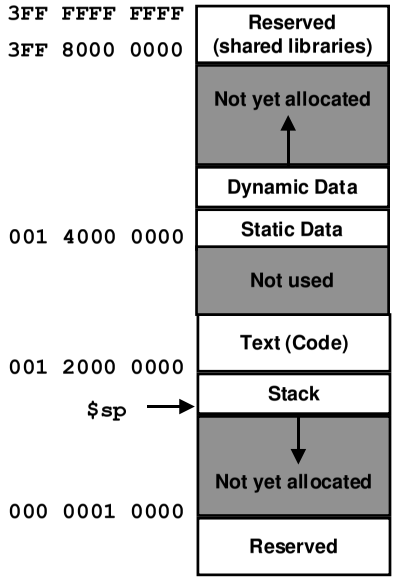
Virtual Address Ranges

Binary Address	Segment	Purpose
`1…1 11 xxxx…xxx`	**seg1**	**Kernel accessible virtual addresses**
– Information maintained by OS but not to be accessed by user		
`1…1 10 xxxx…xxx`	**kseg**	**Kernel accessible physical addresses**
– No address translation performed		
– Used by OS to indicate physical addresses		
`0…0 0x xxxx…xxx`	**seg0**	**User accessible virtual addresses**
– Only part accessible by user program		

Address Patterns

- **Must have high order bits all 0's or all 1's**
  - Currently 64–43 = 21 wasted bits in each virtual address
- **Prevents programmers from sticking in extra information**
  - Could lead to problems when want to expand virtual address space in future

Alpha Seg0 Memory Layout



Regions

- **Data**
  - Static space for global variables
    - » Allocation determined at compile time
    - » Access via `$gp`
  - Dynamic space for runtime allocation
    - » E.g., using `malloc`
- **Text**
  - Stores machine code for program
- **Stack**
  - Implements runtime stack
  - Access via `$sp`
- **Reserved**
  - Used by operating system
    - » shared libraries, process info, etc.

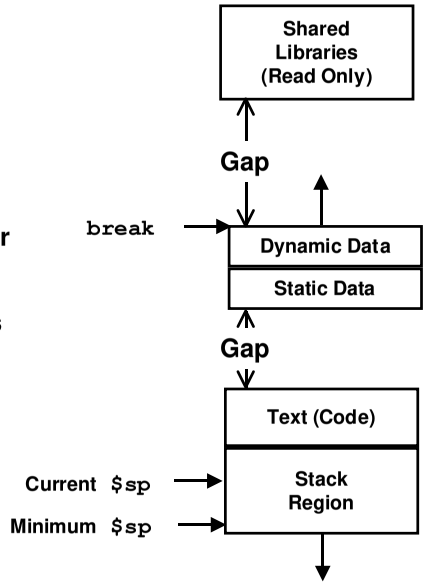
Alpha Seg0 Memory Allocation

Address Range

- **User code can access memory locations in range `0x0000000000010000` to `0x000003FFFFFFFFFF`**
- **Nearly $2^{42} \approx 4.3980465 \times 10^{12}$ byte range**
- **In practice, programs access far fewer**

Dynamic Memory Allocation

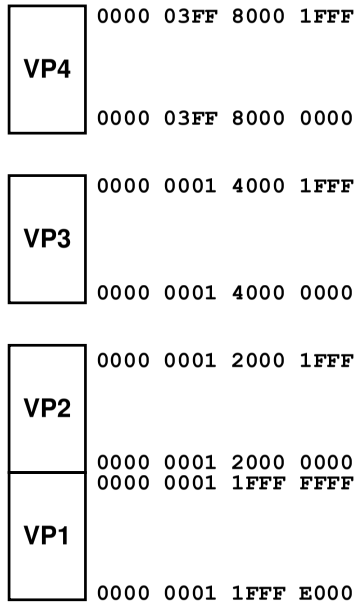
- **Virtual memory system only allocates blocks of memory as needed**
- **As stack reaches lower addresses, add to lower allocation**
- **As break moves toward higher addresses, add to upper allocation**
  - Due to calls to `malloc`, `calloc`, etc.



Minimal Page Table Configuration

User-Accessible Pages

- **VP4: Shared Library**
  - Read only to prevent undesirable interprocess interactions
  - Near top of Seg0 address space
- **VP3: Data**
  - Both static & dynamic
  - Grows upward from virtual address `0x140000000`
- **VP2: Text**
  - Read only to prevent corrupting code
- **VP1: Stack**
  - Grows downward from virtual address `0x120000000`



Partitioning Addresses

Address `0x001 2000 0000`

0000	0000	0001	0010	0000	0000	0000	0000	0000	0000	0000

0000000000	100100000	0000000000	0000000000000

- **Level 1: 0 Level 2: 576 Level 3: 0**

Address `0x001 4000 0000`

0000	0000	0001	0100	0000	0000	0000	0000	0000	0000	0000

0000000000	101000000	0000000000	0000000000000

- **Level 1: 0 Level 2: 640 Level 3: 0**

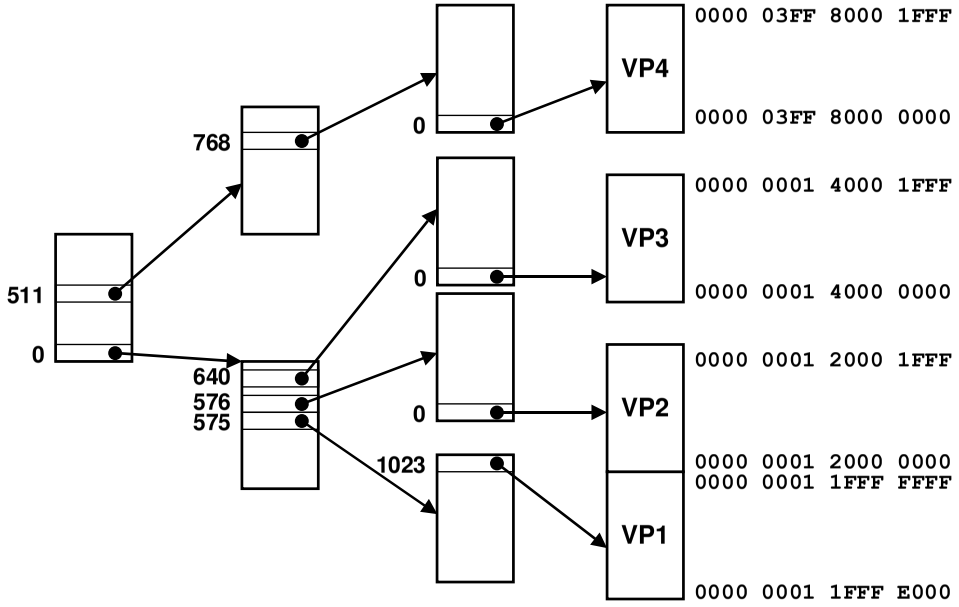
Address `0x3FF 8000 0000`

0011	1111	1111	1000	0000	0000	0000	0000	0000	0000	0000

0111111111	110000000	0000000000	0000000000000

- **Level 1: 511 Level 2: 768 Level 3: 0**

Mapping Minimal Configuration



Increasing Heap Allocation

Without More Page Tables

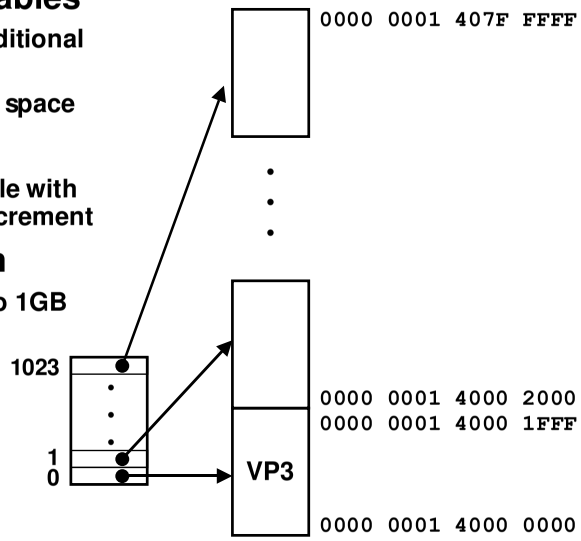
- **Could allocate 1023 additional pages**
- **Would give ~8MB heap space**

Adding Page Tables

- **Must add new page table with each additional 8MB increment**

Maxiumum Allocation

- **Our Alphas limit user to 1GB data segment**
- **Limit stack to 32MB**



Expanding Alpha Address Space

Increase Page Size

- **Increasing page size 2X increases virtual address space 16X**
  - 1 bit page offset, 1 bit for each level index

level 1	level 2	level 3	page offset
10+k	10+k	10+k	13+k

Physical Memory Limits

- **Cannot be larger than kseg**
  VA bits –2 ≥ PA bits
- **Cannot be larger than 32 + page offset bits**
  - Since PTE only has 32 bits for PPN

Configurations

• **Page Size**	**8K**	**16K**	**32K**	**64K**
• **VA Size**	**43**	**47**	**51**	**55**
• **PA Size**	**41**	**45**	**47**	**48**

Main Theme

Programmer's View

- **Large "flat" address space**
  - Can allocate large blocks of contiguous addresses
- **Processor "owns" machine**
  - Has private address space
  - Unaffected by behavior of other processes

System View

- **User virtual address space created by mapping to set of pages**
  - Need not be contiguous
  - Allocated dynamically
  - Enforce protection during address translation
- **OS manages many processes simultaneously**
  - Continually switching among processes
  - Especially when one must wait for resource
    - » E.g., disk I/O to handle page fault