Virtual Memory

- 13 - Address Spaces
- 14 - Memory API
- 15 - Address Translation
- 16 - Segmentation
- 17 - Free Space Management
- 18 - Paging
- 19 - Translation Lookaside Buffers
- 20 - Advanced Paging
- 21 - Swapping
- 22 - Swapping Policy

Who Handles The TLB Miss?

- Hardware handles the TLB miss entirely on CISC processors.
  - The hardware know where the page tables are located
  - … "walks" the page table, finding the correct entry and extracting the desired translation, update and retry instruction.
  - this is a hardware-managed TLB.

- RISC processors often manage TLBs in software.
  - On a TLB miss, the hardware raises an exception
    - Trap handler is code within the OS that is written with the express purpose of handling TLB misses.

TLB Control Flow algorithm (OS Handled)

- The hardware would do the following:

```
1:        VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2:        (Success, TlbEntry) = TLB_Lookup(VPN)
3:        if (Success == True) // TLB Hit
4:                if (CanAccess(TlbEntry.ProtectBits) == True)
5:                        Offset = VirtualAddress & OFFSET_MASK
6:                        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:                        Register = AccessMemory(PhysAddr)
8:                else
9:                        RaiseException(PROTECTION_FAULT)
10:       else // TLB Miss
11:               RaiseException(TLB_MISS)
```

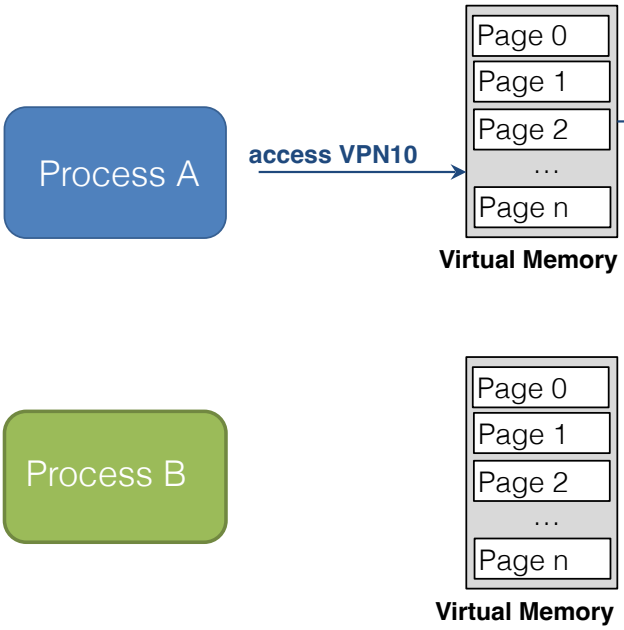
- But might be slow, why not just use the hardware approach?

TLB entry

- TLB entries are often *fully associative* (any entry for any mapping)
  - A typical TLB might have 32, 64, or 128 entries.
  - Hardware searches the TLB in parallel to find the translation.
  - other bits: valid, protection, address-space identifier, dirty bit

Typical TLB entry

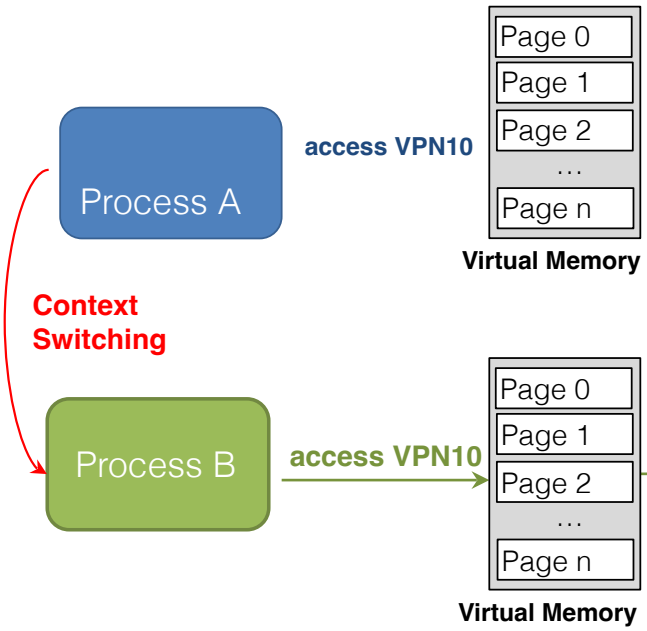
TLB Issue: Context Switching



TLB Table

VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
-	-	-	-
-	-	-	-

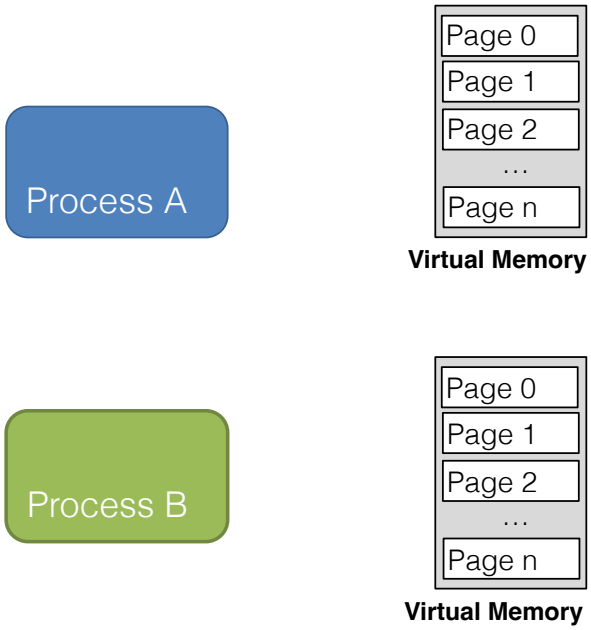
TLB Issue: Context Switching



TLB Table

VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
10	170	1	rwx
-	-	-	-

TLB Issue: Context Switching



TLB Table

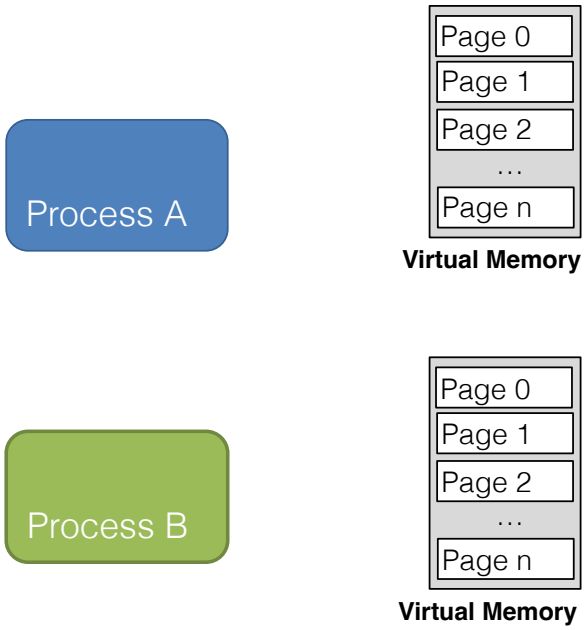
VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
10	170	1	rwx
-	-	-	-

Can't Distinguish which entry is meant for which process

Could just flush the TLB on every context switch…

Disambiguating Address Spaces

- Provide an address space identifier(ASID) field in the TLB.



TLB Table

VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
-	-	-	-	-
10	170	1	rwx	2
-	-	-	-	-

Another Case

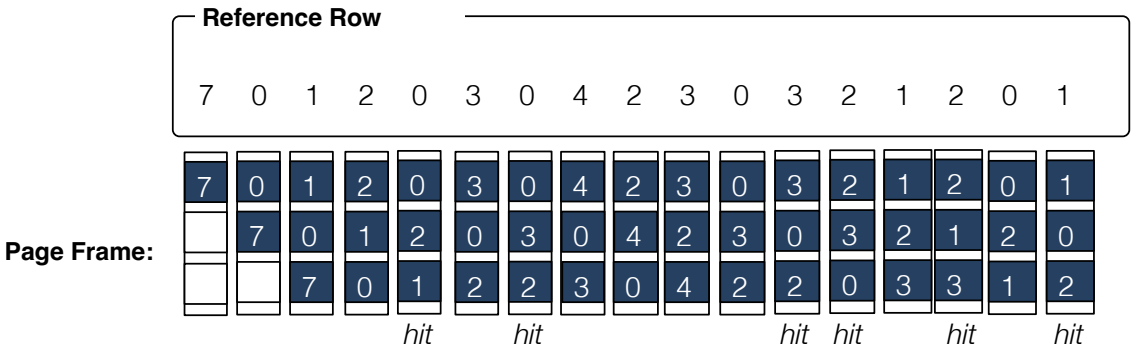
- Two processes share a page.
  - Process 1 is sharing physical page 101 with Process2.
  - P1 maps this page into the 10th page of its address space.
  - P2 maps this page to the 50th page of its address space.

VPN	PFN	valid	prot	ASID
10	101	1	rwx	1
-	-	-	-	-
50	101	1	rwx	2
-	-	-	-	-

Sharing of pages is useful as it reduces the number of physical pages in use.

TLB Replacement Policy

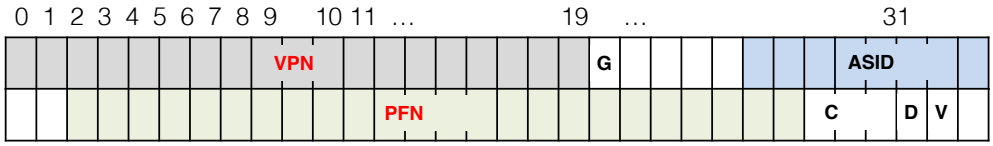
- LRU (Least Recently Used)
  - Evict an entry that has not recently been used.
  - Take advantage of *locality* in the memory-reference stream.



- 6 hits, 11 misses

A Real TLB Entry

64-bit MIPS R4000 TLB entry



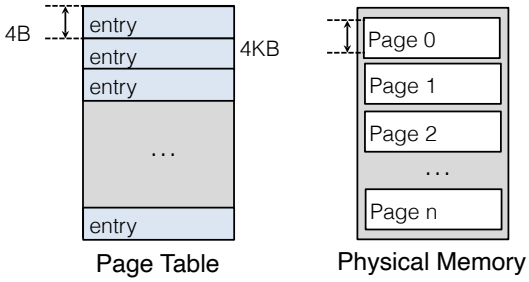
Flag	Content
19-bit VPN	The rest reserved for the kernel.
24-bit PFN	Systems can support with up to 64GB of main memory(pages).
Global bit(G)	Used for pages that are globally-shared among processes.
ASID	OS can use to distinguish between address spaces.
Coherence bit(C)	determine how a page is cached by the hardware.
Dirty bit(D)	marking when the page has been written.
Valid bit(V)	tells the hardware if there is a valid translation present in the entry.

Virtual Memory

- 13 - Address Spaces
- 14 - Memory API
- 15 - Address Translation
- 16 - Segmentation
- 17 - Free Space Management
- 18 - Paging
- 19 - Translation Lookaside Buffers
- 20 - Advanced Paging
- 21 - Swapping
- 22 - Swapping Policy

Paging: Linear (Single-Level) Tables

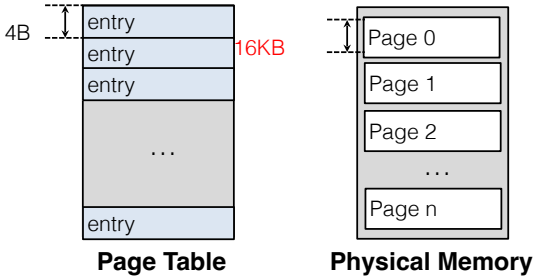
- Usually one page table for every process in the system.
- Example:
  - 32-bit address space, 4KB pages, 4-byte page-table entries



Page table size = $\frac{2^{32}}{2^{12}} * 4Byte = 4MByte$

Paging: Smaller Tables

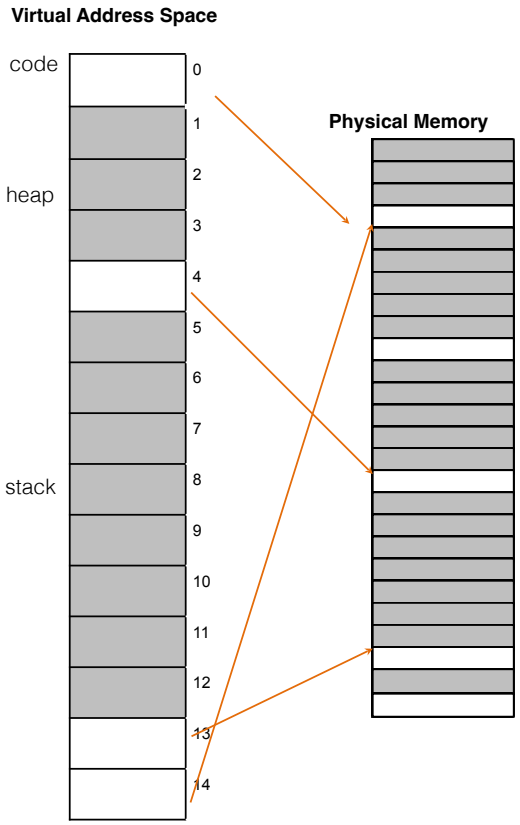
- Larger pages mean fewer entries
  - 32-bit address space, 16KB pages, 4-byte entries.



$\frac{2^{32}}{2^{16}} * 4 = 1MB$ **per page table**

Big pages lead to internal fragmentation.

Problem

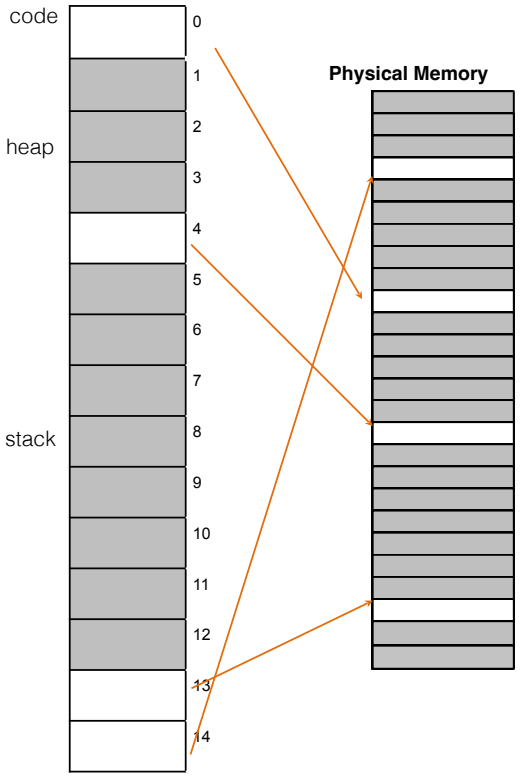


PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
…	…	…	…	…
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

Problem

Most of the page table is **unused**

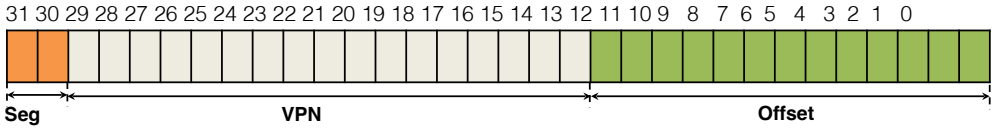


PFN	valid	prot	present	dirty
9	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
…	…	…	…	…
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

Hybrid: Page Table Per Segment

- Each process has three page tables associated with it.
  - Base register for each segment is physical address of its page table.



32-bit Virtual address space with 4KB pages

Seg value	Content
00	unused segment
01	code
10	heap
11	stack

GeekOS!

TLB miss on Hybrid Approach

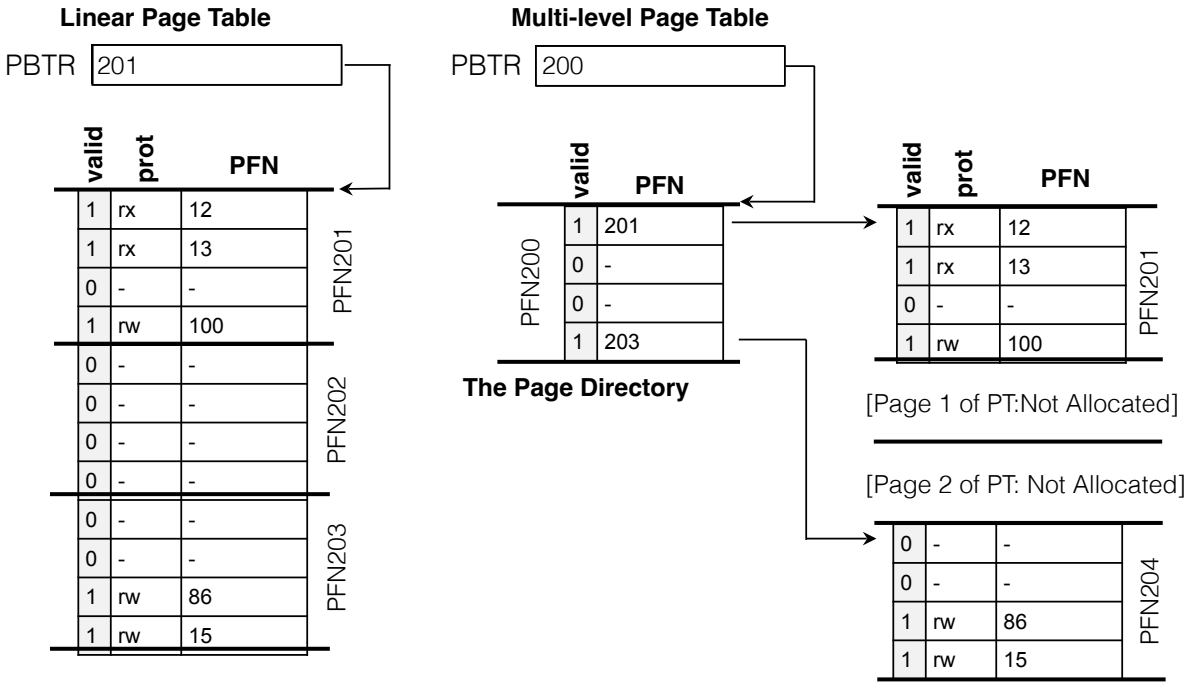
- Need physical address of entry from page table.
  - Segment bits (SN) determine which base and bounds pair
  - Hardware combines physical address therein and the VPN to form the address of the page table entry (PTE) .

```
01:      SN = (VirtualAddress & SEG_MASK) >> SN_SHIFT
02:      VPN = (VirtualAddress & VPN_MASK) >> VPN_SHIFT
03:      AddressOfPTE = Base[SN] + (VPN * sizeof(PTE))
```

Multi-level Page Tables

- Hybrid Approach is not without problems
  - Sparsely-used heap still leads to external fragmentation
- Turns the linear page table into something like a tree
  - Page the page table
  - Allocate page-table pages as needed
  - Track valid page table pages with *page directory*

Multi-level Page Tables: Page directory



Linear (Left) And Multi-Level (Right) Page Tables

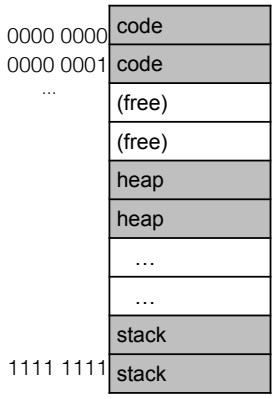
Multi-level Page Tables

- Page directory has:
  - one page directory entry (PDE) per page of the page table
  - Valid bit and page frame number (PFN)

- Advantages
  - Page-table space in proportion to used address space
  - OS can lazily allocate new pages as need
  - *Indirection* can disperse page-table pages through memory

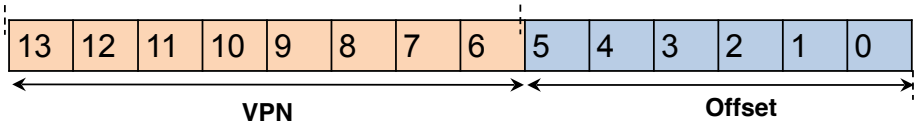
- Disadvantages
  - Time and space tradeoff
  - Complexity

A Detailed Multi-Level Example



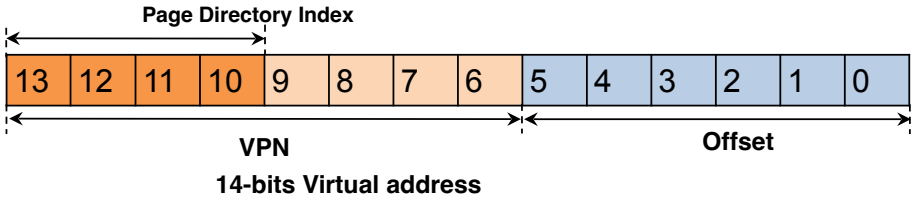
Flag	Detail
Address space	16 KB
Page size	64 byte
Virtual address	14 bit
Num pages	$2^{14}/2^6 = 2^8 = 256$ pages
VPN	8 bit
Offset	6 bit
Page table entry	4 bytes

A 16-KB Address Space With 64-byte Pages



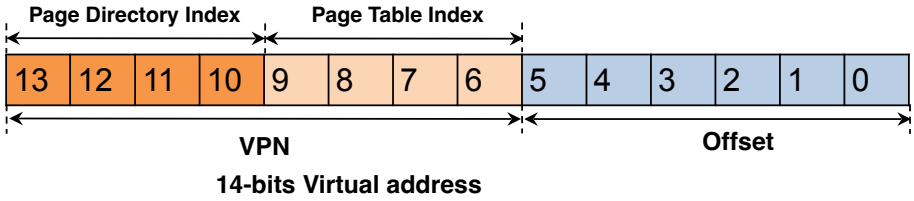
Detailed Example

- Page directory has one entry per page of the page table
  - 256 pages, 4 bytes for PTE, 64 byte pages
    - $256 \times 4/64 = 16$ pages
- Accessing invalid page-directory entry raises exception



Detailed Example

- If PDE is valid:
  - Fetch the page table entry (PTE) from the page of the page table pointed to by this page-directory entry.
- This page-table index can then be used to index into the page table itself.



The Translation Process: Remember the TLB

```
18:      PTIndex = (VPN & PT_MASK) >> PT_SHIFT
19:      PTEAddr = (PDE.PFN << SHIFT) + (PTIndex * sizeof(PTE))
20:      PTE = AccessMemory(PTEAddr)
21:      if(PTE.Valid == False)
22:               RaiseException(SEGMENTATION_FAULT)
23:      else if(CanAccess(PTE.ProtectBits) == False)
24:               RaiseException(PROTECTION_FAULT);
25:      else
26:               TLB_Insert(VPN, PTE.PFN , PTE.ProtectBits)
27:               RetryInstruction()
```


Inverted Page Tables

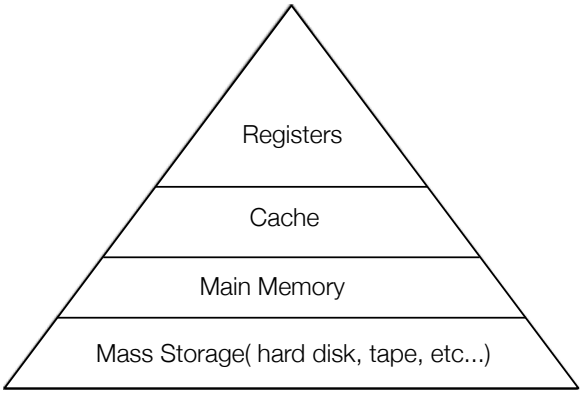
- Keeping only a single page table that has
  - an entry for each physical page of the system
- The entry tells us
  - which process is using this page, and
  - which virtual page that maps to this physical page
- Finding translating a virtual address now requires a search!
  - But can use a hash (PowerPC)

Virtual Memory

- 13 - Address Spaces
- 14 - Memory API
- 15 - Address Translation
- 16 - Segmentation
- 17 - Free Space Management
- 18 - Paging
- 19 - Translation Lookaside Buffers
- 20 - Advanced Paging
- 21 - Swapping and Demand Paging
- 22 - Swapping Policy

Beyond Physical Memory: Mechanisms

- Require an additional level in the memory hierarchy.
  - OS need a place to stash away portions of address space that currently aren't in great demand.
  - In modern systems, this role is usually served by a hard drive



Memory Hierarchy in modern system

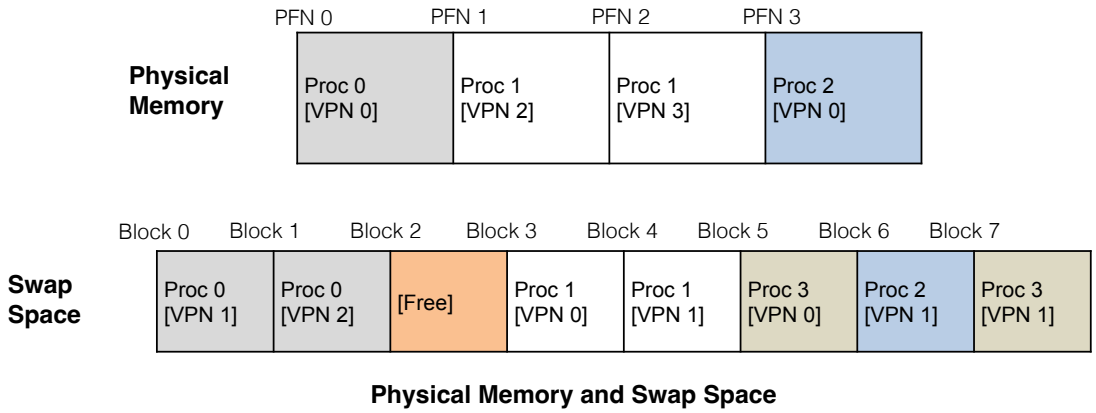
Single large address for a process

- Need to arrange for the code or data to be in memory before calling a function or accessing data.

- Beyond just a single process.
  - The addition of swap space allows the OS to support the illusion of a large virtual memory for multiple concurrently-running process

Swap Space

- Reserve some space on the disk for swapping pages



Physical Memory and Swap Space

Present Bit

- Add some machinery higher up in the system in order to support swapping pages to and from the disk.
  - When the hardware looks in the PTE, it may find that the page is not present in physical memory.

Value	Meaning
1	page is present in physical memory
0	The page is not in memory but rather on disk.

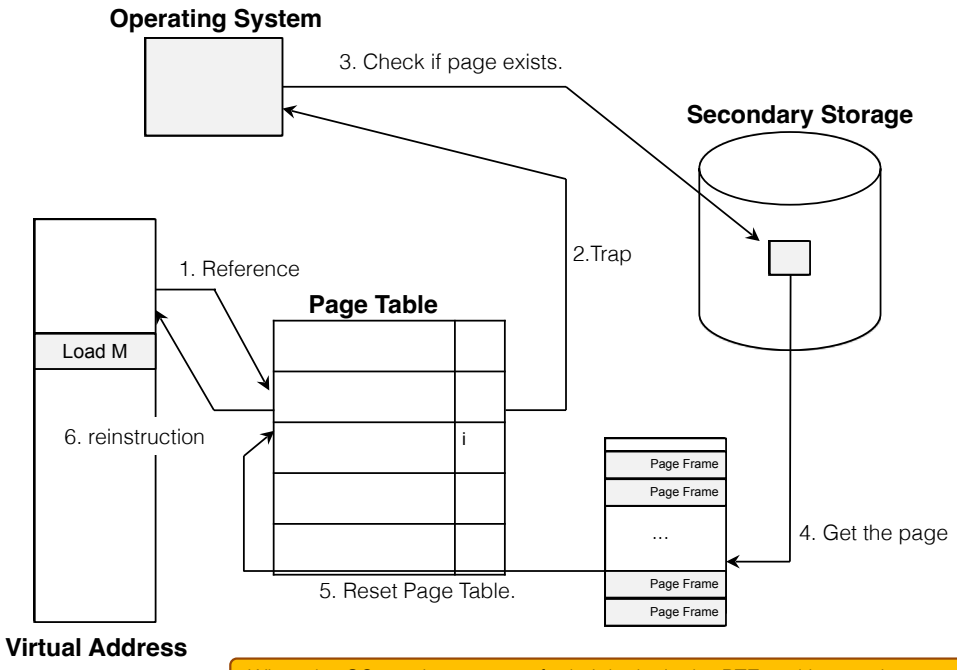
- OS often needs to make room for the new pages
  - The process of picking a page to replace is known as page-replacement or victim-selection policy

The Page Fault

- Accessing page that is not in physical memory.
  - A page with *false* present bit has either:
    - never been in-core (lazily loaded), or
    - has been swapped out to disk

Page Fault Control Flow

- PTE used for data such as the PFN of the page for a disk address.



When the OS receives a page fault, it looks in the PTE and issues the request to disk.

When Replacements Really Occur

- Wait until memory entirely full?
  - No, proactively try to keep small portion of memory free.
- Swap or Page Daemon
  - Frees/evicts page frames if fewer than a low-water threshold available.
  - …until a high-water threshold pages available.

Virtual Memory

- 13 - Address Spaces
- 14 - Memory API
- 15 - Address Translation
- 16 - Segmentation
- 17 - Free Space Management
- 18 - Paging
- 19 - Translation Lookaside Buffers
- 20 - Advanced Paging
- 21 - Swapping
- 22 - Swapping Policy

Beyond Physical Memory: Policies

- Memory pressure forces the OS to start paging out pages to make room for actively-used pages.
- Deciding which page to evict is encapsulated within the replacement policy of the OS.

The Optimal Replacement Policy

- Leads to the fewest number of misses overall
  - Replaces the page that will be accessed furthest in the future
  - Resulting in the fewest-possible cache misses
- Serve only as a comparison point, to know how close we are to perfect

Tracing the Optimal Policy

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	2	0,1,3
0	Hit		0,1,3
3	Hit		0,1,3
1	Hit		0,1,3
2	Miss	3	0,1,2
1	Hit		0,1,2

Reference Row

0 1 2 0 1 3 0 3 1 2 1

Hit rate is $\frac{Hits}{Hits + Misses} = \mathbf{54.6\%}$

A Simple Policy: FIFO

- Pages were placed in a queue when they enter the system.
- When a replacement occurs, the page on the tail of the queue(the "**First-in**" pages) is evicted.
  - It is simple to implement, but can't determine the importance of blocks.

Tracing the FIFIO Policy

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	0	1,2,3
0	Miss	1	2,3,0
3	Hit		2,3,0
1	Miss		3,0,1
2	Miss	3	0,1,2
1	Hit		0,1,2

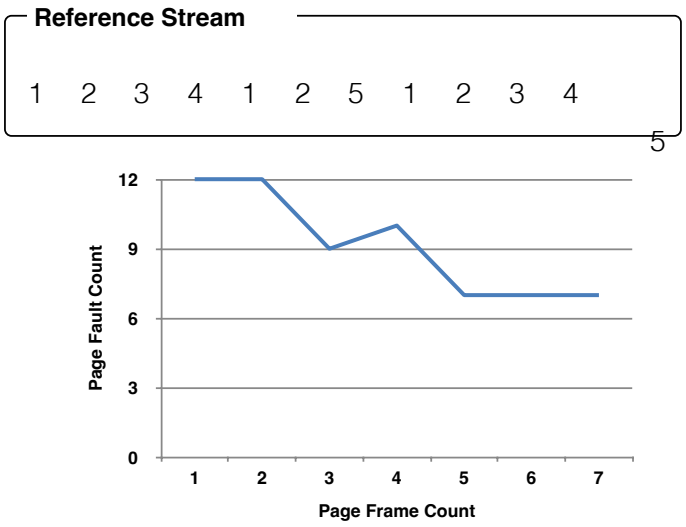
Reference Stream

0 1 2 0 1 3 0 3 1 2 1

Hit rate is $\frac{Hits}{Hits + Misses} = 36.4\%$

BELADY'S ANOMALY

- We would expect the cache hit rate to increase when the cache gets larger. But with FIFO, it gets worse:



- FIFO does not have stack policy
  - i.e. pages with *n* frames always subset of pages with *n+1* frames

Using History

- Lean on the past and use **history**.
  - Two type of historical information.

Historical Information	Meaning	Algorithms
recency	The more recently a page has been accessed, the more likely it will be accessed again	LRU
frequency	If a page has been accessed many times, It should not be replcaed as it clearly has some value	LFU

Using History : LRU

- Replaces the least-recently-used page.

Reference Stream

0 1 2 0 1 3 0 3 1 2 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		1,2,0
1	Hit		2,0,1
3	Miss	2	0,1,3
0	Hit		1,3,0
3	Hit		1,0,3
1	Hit		0,3,1
2	Miss	0	3,1,2
1	Hit		3,2,1

The Exam

- GeekOS
- Synchronization
  - Deadlock mitigation
  - Trylocks, TestAndSet
  - Writing code
  - Semantics
- Queueing
  - Characteristics
  - Deriving queue lengths
  - changing my terminology: turnaround time, not response
- Paging and memory systems
  - segmentation
  - paging
  - multi-level page tables
  - victim-replacement policies: LRU, FIFO, MIN

The Exam (all point totals approximate)

- GeekOS: 5 pts
- Synchronization: 45 pts
  - Deadlock mitigation
  - Trylocks, TestAndSet
  - Writing code
  - Semantics
- Queueing: 25 pts
  - Characteristics
  - Deriving queue lengths
  - changing my terminology: turnaround time, not response
- Paging and memory systems: 25 pts
  - segmentation
  - paging
  - multi-level page tables
  - victim-replacement policies: LRU, FIFO, MIN