Virtual Memory

Motivation: why not direct physical memory access?
Address translation with pages
Optimizing translation: translation lookaside buffer
Extra benefits: sharing and protection

Memory as a contiguous array of bytes is a lie! Why?

Problem 1: Memory Management

Process 1
Process 2
Process 3
...
Process n

What goes where?

Also:
Context switches must swap out entire memory contents.
Isn't that expensive?

Problem 2: Capacity

64-bit addresses can address several exabytes
(18,446,744,073,709,551,616 bytes)

Physical main memory offers a few gigabytes
(e.g. 8,589,934,592 bytes)

Actually, it's smaller than that.

1 virtual address space per process, with many processes...
Problem 3: Protection

Problem 4: Sharing

Solution: Virtual Memory (address indirection)

Indirection

Direct naming

Indirect naming

What if we move Thing?

Tangent: Indirection everywhere

- Pointers
- Constants
- Procedural abstraction
- Domain Name Service (DNS)
- Dynamic Host Configuration Protocol (DHCP)
- Phone numbers
- 911
- Call centers
- Snail mail forwarding
- ...

"Any problem in computer science can be solved by adding another level of indirection"

- David Wheeler, inventor of the subroutine (a.k.a. procedure), or Butler Lampson

Another Wheeler quote? "Compatibility means deliberately repeating other people's mistakes."
Virtual Addressing

Memory Management Unit translates virtual address to physical address

Physical addresses are invisible to programs.

Page-based Mapping

both address spaces divided into fixed-size, aligned pages
page size = power of two

Map virtual pages onto physical pages.

Some virtual pages do not fit! Where are they stored?

virtual address space usually much larger than physical address space

Virtual Memory: cache for disk?

Not drawn to scale

SRAM

DRAM

Disk

Memory miss penalty (latency): 33x

Memory miss penalty (latency): 10,000x
Design for a Slow Disk: Exploit Locality

- **Large page size**
  - Usually 4KB, up to 2-4MB

- **Fully associative**
  - Store any virtual page in any physical page
  - Large mapping function

- **Sophisticated replacement policy**
  - Not just hardware

- **Write-back**

Address Translation

- **Virtual address (VA)**
- **Physical address (PA)**
- **CPU Chip**
- **MMU**

- **Main memory**
  - 0: 1: 3: 4: 5: 6: 7: 8: M-1:

- **What happens in here?**

Page Table

- Array of page table entries (PTEs)
- Mapping virtual page to where it is stored

- **Physical Page Number or disk address**
- **Valid**

- **Swap space (Disk)**

Address Translation with a Page Table

- **Virtual address (VA)**
- **Virtual page number (VPN)**
- **Virtual page offset (VPO)**
- **Physical address (PA)**

- **Page table**

- **Base address of current process's page table**

- **Virtual page mapped to physical page?**

- **How many page tables are in the system?**
**Page Hit:** virtual page in memory

- Virtual Page Number
- Physical Page Number or disk address
- Valid
- Physical pages (Physical memory)
- PTE 0
- PTE 7

**Page Fault:** exceptional control flow

- Process accessed virtual address in a page that is not in physical memory.
- Exception: page fault
- OS exception handler
- Create page and load into memory
- Returns to faulting instruction: `movl` is executed again!

**Page Fault: 1. page not in memory**

- Exception!
- What now? OS handles fault

**Page Fault: 2. OS evicts another page.**

- "Page out"
**Page Fault:** 2. OS loads needed page.

1. Processor sends virtual address to MMU (memory management unit)
2-3) MMU fetches PTE from page table in cache/memory
4) MMU sends physical address to cache/memory
5) Cache/memory sends data word to processor

Terminology

- **context switch**
  Switch control between processes on the same CPU.

- **page in**
  Move pages of virtual memory from disk to physical memory.

- **page out**
  Move pages of virtual memory from physical memory to disk.

- **swap**
  Total working set size of processes is larger than physical memory.
  Most time is spent paging in and out instead of doing useful computation.

(I find all these terms useful when talking to other computer scientists about my brain...)

**Address Translation: Page Fault**

1) Processor sends virtual address to MMU
2-3) MMU fetches PTE from page table in cache/memory
4) Valid bit is zero, so MMU triggers page fault exception
5) Handler identifies victim (and, if dirty, pages it out to disk)
6) Handler pages in new page and updates PTE in memory
7) Handler returns to original process, restarting faulting instruction
Translation sounds slow!

Each access = 2 accesses: load PTE, access requested address
PTEs may be cached, but may be evicted.
L1 cache hit still requires 1-3 cycles

What can we do to make this faster?

Translation Lookaside Buffer (TLB)

Small hardware cache in MMU just for page table entries
Modern Intel processors: 128 or 256 entries in TLB

Much faster than a page table lookup in cache/memory

In the running for "classiest name of a thing in CS"

TLB Hit

A TLB hit eliminates a memory access

TLB Miss

A TLB miss incurs an additional memory access (the PTE)
Fortunately, TLB misses are rare. Does a TLB miss require disk access?
### Simple Memory System Example (small)

**Addressing**
- 14-bit virtual addresses
- 12-bit physical address
- Page size = 64 bytes

#### Virtual Page Number (VPN) and Virtual Page Offset (VPO)

<table>
<thead>
<tr>
<th>13 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VPN</strong></td>
<td><strong>VPO</strong></td>
</tr>
</tbody>
</table>

#### Physical Page Number (PPN) and Physical Page Offset (PPO)

<table>
<thead>
<tr>
<th>13 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PPN</strong></td>
<td><strong>PPO</strong></td>
</tr>
</tbody>
</table>

### Simple Memory System Page Table

Only showing first 16 entries (out of 256 = $2^8$)

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td>–</td>
<td>0</td>
</tr>
</tbody>
</table>

What about a real address space? Read more in the book...

### Simple Memory System TLB

16 entries
- 4-way associative
- TLB ignores page offset. Why?

#### TLB Tag, TLB Index, Virtual Page Number, Virtual Page Offset

<table>
<thead>
<tr>
<th>13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TLB tag</strong></td>
<td><strong>TLB index</strong></td>
</tr>
</tbody>
</table>

### Simple Memory System Cache

16 lines, 4-byte block size
- Physically addressed
- Direct mapped

#### Cache Tag, Cache Index, Cache Offset, Physical Page Number, Physical Page Offset

<table>
<thead>
<tr>
<th>13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cache tag</strong></td>
<td><strong>cache index</strong></td>
</tr>
</tbody>
</table>

---

**Valid**

- PPN
- VPN
- Set
- Tag
### Address Translation Example #1

**Virtual Address:** 0x03D4

<table>
<thead>
<tr>
<th>TLB</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>02</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2D</td>
<td>1</td>
<td>02</td>
<td>04</td>
<td>-</td>
<td>O</td>
<td>03</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>-</td>
<td>-</td>
<td>06</td>
<td>-</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>-</td>
<td>-</td>
<td>03</td>
<td>00</td>
<td>0A</td>
<td>34</td>
<td>1</td>
<td>02</td>
</tr>
</tbody>
</table>

**Physical Address:** 0x354

<table>
<thead>
<tr>
<th>Cache</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>02</td>
<td>-</td>
<td>-</td>
<td>01</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2D</td>
<td>1</td>
<td>02</td>
<td>04</td>
<td>-</td>
<td>O</td>
<td>02</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>-</td>
<td>-</td>
<td>06</td>
<td>-</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>-</td>
<td>-</td>
<td>03</td>
<td>00</td>
<td>0A</td>
<td>34</td>
<td>1</td>
<td>02</td>
</tr>
</tbody>
</table>

### Address Translation Example #2

**Virtual Address:** 0x0B8F

<table>
<thead>
<tr>
<th>TLB</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>02</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2D</td>
<td>1</td>
<td>02</td>
<td>04</td>
<td>-</td>
<td>O</td>
<td>03</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>-</td>
<td>-</td>
<td>06</td>
<td>-</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>-</td>
<td>-</td>
<td>03</td>
<td>00</td>
<td>0A</td>
<td>34</td>
<td>1</td>
<td>02</td>
</tr>
</tbody>
</table>

### Address Translation Example #3

**Virtual Address:** 0x0020

<table>
<thead>
<tr>
<th>TLB</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>02</td>
<td>-</td>
<td>-</td>
<td>01</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2D</td>
<td>1</td>
<td>02</td>
<td>04</td>
<td>-</td>
<td>O</td>
<td>02</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>-</td>
<td>-</td>
<td>06</td>
<td>-</td>
<td>03</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>-</td>
<td>-</td>
<td>03</td>
<td>00</td>
<td>0A</td>
<td>34</td>
<td>1</td>
<td>02</td>
</tr>
</tbody>
</table>
**Address Translation Example #3**

Virtual Address: 0x0020

<table>
<thead>
<tr>
<th>Virtual page #</th>
<th>TLB index</th>
<th>TLB tag</th>
<th>Page #?</th>
<th>Fault?</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 09 13 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 09 17 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02 04 09 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03 06 11 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04 06 11 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05 06 11 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07 06 11 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Virtual pages can live anywhere in physical memory.

**Easy cached access to storage > memory**

Good temporal locality + small working set = mostly page hits

Great if working sets < physical memory, even if total data > physical memory

If combined working sets of all processes) > physical memory:

Thrashing: Performance meltdown. CPU always waiting or paging.

Full indirection quote: **"Every problem in computer science can be solved by adding another level of indirection, but that usually will create another problem."**
**Easy protection and sharing**

**Protection:**
All user accesses go through translation.
Impossible to access physical memory not mapped in virtual address space.

**Sharing:**
Map virtual pages in separate address spaces to same physical page (PP 6).

---

**Summary**

**Programmer’s view of virtual memory**
Each process has its own private linear address space
Cannot be corrupted by other processes

**System view of virtual memory**
Uses memory efficiently by caching virtual memory pages
Efficient only because of locality
Simplifies memory management and sharing
Simplifies protection – easy to interpose and check permissions

---

**Easy protection**
All user accesses go through translation.
Extend page table entries with permission bits.
MMU checks permission bits on every memory access
If not allowed, raise exception.

---

**Memory System Summary**

**L1/L2/L3 Cache**
Purely a speed-up technique
"Invisible" to application programmer and OS
Implemented totally in hardware

**Virtual Memory**
Supports processes, memory management
Operating System (software)
Allocates physical memory
Maintains page tables and memory metadata
Handles exceptions, fills tables used by hardware
Hardware
Translates virtual addresses via mapping tables, enforces permissions
Accelerates mapping via translation cache (TLB)
Memory System Summary

L1/L2/L3 Cache
- Controlled by hardware
- Programmer cannot control it
- Programmer can write code in a way that takes advantage of it

Virtual Memory
- Controlled by OS and hardware
- Programmer cannot control mapping to physical memory
- Programmer can control sharing and some protection via OS system calls