Virtual Memory (VM)

Overview and motivation
VM as tool for caching
Address translation
VM as tool for memory management
VM as tool for memory protection

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

Problem 1: How Does Everything Fit?

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 dot compared to virtual memory.

1 virtual address space per process, with many processes...

Problem 2: Memory Management

Process 1
Process 2
Process 3
...
Process n

stack
heap
.text
data

Physical main memory

What goes where?

Problem 3: How To Protect

Physical main memory

Process i
Process j

Problem 4: How To Share?

Physical main memory

Process i
Process j
Indirection

“Any problem in computer science can be solved by adding another level of indirection.” - David Wheeler, inventor of the subroutine (a.k.a., procedure)

<table>
<thead>
<tr>
<th>Without Indirection</th>
<th>With Indirection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Thing</td>
<td>Thing</td>
</tr>
</tbody>
</table>

What if I want to move Thing?

Indirection

Indirection: the ability to reference something using a name, reference, or container instead of the value itself. A flexible mapping between a name and a thing allows changing the thing without notifying holders of the name.

<table>
<thead>
<tr>
<th>Without Indirection</th>
<th>With Indirection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Thing</td>
<td>Thing</td>
</tr>
</tbody>
</table>

Examples of indirection:
- Domain Name Service (DNS): translation from name to IP address
- phone system: cell phone number portability
- snail mail: mail forwarding
- 911: routed to local office
- Dynamic Host Configuration Protocol (DHCP): local network address assignment
- call centers: route calls to available operators, etc.

Indirection in Virtual Memory

Each process gets its own private virtual address space
Solves the previous problems

Address Spaces

Virtual address space: Set of $N = 2^n$ virtual addresses
\{0, 1, 2, ..., N-1\}

Physical address space: Set of $M = 2^m$ physical addresses (m >= n)
\{0, 1, 2, 3, ..., M-1\}

Every byte in main memory has:
- one physical address
- zero, one, or more virtual addresses
Mapping

A virtual address can be mapped to either physical memory or disk.

Physical Addressing

Used in "simple" systems with (usually) just one process: embedded microcontrollers in devices like cars, elevators, and digital picture frames.

Virtual Addressing

Physical addresses are invisible to programs.

VM and the Memory Hierarchy

Think of virtual memory as array of \( N = 2^n \) contiguous bytes. Pages of virtual memory usually stored in physical memory, sometimes spill to disk.

Page: unit of aligned memory (size is \( P = 2^p \) bytes)
Each virtual page can be stored in any physical page.
or: virtual memory as cache for disk

Physical main memory is a cache for the virtual memory array

Think of virtual memory as an array of $N = 2^n$ contiguous bytes stored on disk.

Virtual Memory Design Consequences

Large page size: typically 4-8 KB, sometimes up to 4 MB

Fully associative
Any virtual page can be placed in any physical page
Requires a "large" mapping function – different from CPU caches

Sophisticated, expensive replacement algorithms in OS
Too complicated and open-ended to be implemented in hardware

Write-back, not write-through

Address Translation

How do we perform the virtual -> physical address translation?
A page table is an array of page table entries (PTEs) that maps virtual pages to physical pages.

Address Translation: Page Tables

Page hit: reference to VM byte in physical memory

Page fault: reference to VM byte NOT in physical memory

How many page tables are in the system?
One per process

Valid bit = 0: page not in memory (page fault)

Common case: pure-hardware translation

Page table

Virtual page number (VPN) Virtual page offset (VPO)
Physical page number (PPN) Physical page offset (PPO)
Valid bit

Valid = 0: page not in memory (page fault)

Physical memory (DRAM)

Address Translation With a Page Table
Page Fault

Process accesses virtual memory location that's not in physical memory.

Returns to faulting instruction: sw is executed again!

Handling Page Fault

Page miss causes page fault (an exception)
Page fault handler selects a victim to be evicted (here VP 4)
Offending instruction re-executed: page hit!

Why can virtual memory be fast?

Locality.

**working set**: all virtual pages that a process is "actively" accessing at a point in time

better temporal locality = smaller working set

If (working set size of one process < main memory size):

Good performance for one process after compulsory misses

But if

SUM(working set sizes of all processes) > main memory size:

**Thrashing**: Performance meltdown where pages are swapped (copied) between memory and disk continuously. CPU always waiting or paging.

Full quote from David Wheeler: "Every problem in computer science can be solved by adding another level of indirection, but that usually will create another problem."

It’s not fast yet...
VM for allocation

Private contiguous address space.

Fully associative: Contiguous virtual pages can live anywhere in physical memory.

VM for sharing and protection

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

Protection:
Impossible to access physical memory not mapped in virtual address space. Cannot conjure address. (Process 2 can’t access PP 2.)

Memory Protection Within a Single Process

Extend page table entries with permission bits
MMU checks permission bits on every memory access

If violated, raises exception

Terminology

context switch
Switch 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.

thrash
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 Hit

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

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

Hmm... 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?

Speeding up Translation with a TLB

Solution: add another cache!

Translation Lookaside Buffer (TLB):
Small hardware cache in MMU
Maps virtual page numbers to physical page numbers
Contains complete page table entries for small number of pages
Modern Intel processors: 128 or 256 entries in TLB
Much faster than a page table lookup in cache/memory
A TLB hit eliminates a memory access

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

Simple Memory System Page Table

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

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?

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>02</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Simple Memory System Cache

16 lines, 4-byte block size  
Physically addressed  

Direct mapped

<table>
<thead>
<tr>
<th>Cache Index</th>
<th>Cache Tag</th>
<th>Physical Page Number</th>
<th>Address Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00 00 00 00 00 00 00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>00 00 00 00 00 00 00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00 00 00 00 00 00 00</td>
<td>2</td>
</tr>
</tbody>
</table>

---

### Address Translation Example #1

Virtual Address: 0x03D4

<table>
<thead>
<tr>
<th>TLB Index</th>
<th>TLB Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00 00 00 00 00 00</td>
<td>00 00</td>
</tr>
</tbody>
</table>

Physical Address: 0x3456

<table>
<thead>
<tr>
<th>Physical Page Number</th>
<th>Physical Page Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00 00 00 00 00 00</td>
<td>00 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>
### Address Translation Example #2

**Virtual Address:** 0x0BBF

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>VPN</th>
<th>Valid</th>
<th>Top</th>
<th>PPN</th>
<th>Invalid</th>
<th>Top</th>
<th>PPN</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**TLB**

<table>
<thead>
<tr>
<th>virtual page</th>
<th>TLB index</th>
<th>TLB Hit?</th>
<th>Page Fault?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0BBF</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Address Translation Example #3

**Virtual Address:** 0x0020

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>VPN</th>
<th>Valid</th>
<th>Top</th>
<th>PPN</th>
<th>Invalid</th>
<th>Top</th>
<th>PPN</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**TLB**

<table>
<thead>
<tr>
<th>virtual page</th>
<th>TLB index</th>
<th>TLB Hit?</th>
<th>Page Fault?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0020</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Address Translation Example #3

**Virtual Address:** 0x0020

**Cache**

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>VPN</th>
<th>Valid</th>
<th>Tag</th>
<th>VPN</th>
<th>Valid</th>
<th>Tag</th>
<th>VPN</th>
<th>Valid</th>
<th>Tag</th>
<th>VPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Physical Address:** 0xA20

<table>
<thead>
<tr>
<th>cache tag</th>
<th>cache index</th>
<th>cache offset</th>
<th>physical page number</th>
<th>physical page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 by providing a convenient interpositioning point to check permissions

Memory System Summary

L1/L2 Memory 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 functions (not in 351)
Servicing a Page Fault

1. Processor signals disk controller
   - Read block of length P starting at disk address X
   - Store starting at memory address Y

2. Read occurs
   - Direct Memory Access (DMA)
   - Under control of I/O controller

3. Controller signals completion
   - Interrupts processor
   - OS resumes suspended process