Virtual Memory
Process Abstraction, Part 2: Private Address Space

**Motivation:** why not direct physical memory access?
- Address translation with pages
- Extra benefits: sharing and protection

Memory as a contiguous array of bytes is a lie! Why?
Problems with physical addressing

Main memory

Physical address (PA)

CPU

Data

M-1:
Problem 1: memory management

Process 1
Process 2
Process 3
...
Process n

×

stack
heap
code
globals
...

Main memory

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)

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

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

(To scale with 64-bit address space,
you can't see it.)
Problem 3: protection

Physical main memory

Process i

Process j

Problem 4: sharing

Physical main memory

Process i

Process j
Solution: Virtual Memory  

(address indirection)

Virtual address space

Process 1

virtual addresses

virtual-to-physical mapping

Physical memory

physical addresses

data

Process n

virtual addresses

Virtual address space

Single physical address space managed by OS/hardware.
Indirection
(it's everywhere!)

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, or Butler Lampson

Another Wheeler quote? "Compatibility means deliberately repeating other people's mistakes."
Virtual addressing and address translation

Memory Management Unit
translates virtual address to physical address

CPU Chip

Virtual address
(VA)
4100

MMU

Physical address
(PA)
4

Main memory

Data

0:
1:
2:
3:
4:
5:
6:
7:
8:
...

M-1:
Page-based mapping

Virtual Address Space

Physical Address Space

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?
Cannot fit all virtual pages! Where are the rest stored?

1. On disk if used
2. Nowhere if not (yet?) used

virtual address space usually much larger than physical address space

Physical Memory Address Space

Virtual Memory Address Space

Virtual Page 2^n - 1

2^n - 1
Virtual memory: cache for disk?

Not drawn to scale!

Example system

SRAM
- L1 I-cache
  - 32 KB
  - Throughput: 16 B/cycle
  - Latency: 3 cycles

- L1 D-cache
  - Throughput: 8 B/cycle
  - Latency: 14 cycles

- L2 unified cache
  - Throughput: 2 B/cycle
  - Latency: 100 cycles

DRAM
- Main Memory
  - Throughput: 1 B/30 cycles
  - Latency: millions

Disk
- solid-state "flash" or spinning magnetic platter.

Cache miss penalty (latency): 33x
Memory miss penalty (latency): 10,000x
Address translation

CPU Chip

Virtual address (VA) 4100

MMU

Physical address (PA) 4

Main memory

0: 1: 2: 3: 4: 5: 6: 7: 8: ...

M-1:

Data
Page table

array of page table entries (PTEs)
mapping virtual page to where it is stored

Physical Page Number
or disk address

PTE 0
0  null
1
1
0
1
0
0

PTE 7
0
1

Valid

Physical pages
(Physical memory)

PP 0
VP 1
VP 2
VP 7
VP 4

PP 3

Swap space
(Disk)

VP 3
VP 6

VP 3
VP 6

null
null

Memory resident,
managed by HW (MMU), OS
Virtual memory benefits:
Simple address space allocation

Process needs private *contiguous* address space.

![Diagram showing virtual address spaces mapping to physical address space](image-url)
Virtual memory benefits:

**Protection:**
All 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).

![Diagram of virtual and physical address spaces](attachment:image.png)
Virtual memory benefits:
Memory permissions

Process 1:

<table>
<thead>
<tr>
<th>VP 0:</th>
<th>Valid</th>
<th>READ</th>
<th>WRITE</th>
<th>EXEC</th>
<th>Physical Page Num</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>PP 6</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>PP 4</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>PP 2</td>
</tr>
</tbody>
</table>

Process 2:

<table>
<thead>
<tr>
<th>VP 0:</th>
<th>Valid</th>
<th>READ</th>
<th>WRITE</th>
<th>EXEC</th>
<th>Physical Page Num</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>PP 9</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>PP 6</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>PP 11</td>
</tr>
</tbody>
</table>

MMU checks on every access.
Exception if not allowed.
Summary: virtual memory

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 (due to locality) by caching virtual memory pages
Simplifies memory management and sharing
Simplifies protection -- easy to interpose and check permissions
More goodies:
- Memory-mapped files
- Cheap \texttt{fork()} with copy-on-write pages (COW)