Virtual Memory (section 5.7)

Main memory can act as a cache for the secondary storage (ex. disk)

- **Virtual addresses** (generated by the CPU)
- **Physical addresses space**
  - Physical memory (size depends on installed memory) -- Caches part of the virtual space.
  - Disk storage contains the virtual address space
  - Space not utilized

**Size of the virtual space is** $2^{32}$ bytes (4GB) in 32-bit machines (ex. MIPS).

In other architectures, the virtual address space may be $2^{48}$ bytes.

**Advantages:**
1) illusion of having more memory than what can fit in the physical memory
Virtual Memory

Virtual addresses for process 1

Virtual addresses for process 2

Physical memory

Some pages belong to process 2

Some pages belong to process 1

Some pages are shared

- Advantages:
  2) More than one process can share the same physical memory
  3) Allows page sharing among processors

Physical memory as a cache for the virtual memory

- Block (page) size = 1KB ~ 64KB (large because of huge miss penalty)
  - (as opposed to 16B~64B in L1/L2)
- Miss penalty (cost of a page fault) = 1M~10M cycles (to access hard drive)
  - (as opposed to 20~150 cycles (L2/memory)
- Hit time = 50 ~ 150 cycles
  - (as opposed to 1~3 in L1 and 6~12 in L2)
- Miss rate = 0.00001~0.001% (called page fault rate)
  - (as opposed to 0.1~10% in L1)
- Always write back (never write through)
  - (always write back in L2, and commonly in L1)
- Fully associative (to maximize hit rate)
  - (as opposed to 4-8 way set associative in L1/L2)
- Replacement implemented in OS
  - (as opposed to hardware implementation for L1/L2)
- Locate pages using page tables
  - (efficient way to implement full associativity)
Address translation through Page Tables

• Table lookup replaces tag comparison in L1/L2.

Address translation through Page Tables

• Table lookup replaces tag comparison in L1/L2.
Virtual to physical address translation

Assume 4KB pages (2\(^{12}\) bytes)

In this example, the physical memory is 1/4 the size of the virtual address space.

The page table is stored in memory starting at the address stored in the “Page Table Register”.

The Page Table is allocated in memory

- Each process has its own page table stored in memory starting at a specific address indicated in a page address register.
- A memory reference (if hits in main memory) requires two memory operations
- A page fault (main memory miss) results in a disk operation.
- The page table and page address register are part of the process context (along with the PC, stack pointer, registers …)
Example of address translation

**Examples:**
- Reference to 0011000 0101 (page 24), translates to 110 0101
- Reference to 0011001 0011 (page 25) causes a page fault

Virtual space = 128 pages
Page = 16 words

Physical memory = 8 pages
Page = 16 words

Page table (128 entries)

Multiple processes share the physical memory

CPU Executing Process i

Virtual memory address

Physical memory address
Multiple processes share the physical memory

How can we solve the problem of accessing the memory twice for each memory reference?