10. The Memory Hierarchy

Need for Memory (Lots of it!)

- The trend: new software requires more memory
- Memory gets cheaper, and software developers always prefer to add extra features to software (that require more memory) than to keep software small just for the old machines.
- Some applications really do *need* lots of memory (e.g virtual PCs)
- In either case hardware needs to deliver bigger memories!

Early mainframe	64 words	6×10^2
High-end PC (1986)	640Kbytes	6×10^5
High-end PC (1996)	32Mbytes	3×10^7
High-end PC (now)	64Gbytes	6×10^{10}

Size vs. Speed

- Larger memory (more bytes) is inherently slower than a smaller one:
  - All else equal, larger physically, so signals must traverse longer wires
  - Addresses are bigger, so multiplexers for address decoding have more gate delays
  - Speed enhancements (e.g. multiple interfaces) affordable only for small memory
- Can identify hierarchy of memory in a typical computer
  - Registers (flip-flop based)
  - Cache RAM (flip-flop based)
  - Primary memory (Dynamic RAM based - see below)
  - Disk (magnetic surface, spinning disk)
  - Tape (magnetic surface, reeled tape)
- Static RAM (SRAM) is based on storing bits in flip-flops
- Dynamic RAM (DRAM) stores bits as isolated islands of charge within device
- DRAM has smaller cells than SRAM but cells are significantly slower
  - DRAM builds bigger but slower memories
  - DRAM almost universally used for primary memory
  - Implementation varies (DDR, DDR2, DDR3) but all based on same basic cell.

Cache Memory

- Modern CPUs invariably have a small amount of fast static RAM called a cache memory usually integrated with the CPU itself (same chip).
- The cache is *much faster* but *much smaller* than primary memory.
- When data or an instruction is fetched from primary memory for the first time it is also stored in the cache.
- Very often instructions and data that have been used once are used again soon afterwards.
  - This is called locality of reference.
  - If items are in the cache they can be retrieved faster than from primary memory.
- Hardware accesses the cache automatically when primary memory is referenced.
  - If the item is in the cache (a cache hit), primary memory access never takes place (fast)
  - If the item is not in the cache (a cache miss) primary memory is accessed anyway (slower)
- Since cache has limited space, need a way of deciding what to throw out when it becomes full, Various options: e.g. least recently used (LRU) item is eliminated.

Unassessed Exercise

- Consider the instruction sequence:

```
            lea     R15,10[R0]
            lea     R1,1[R0]
            lea     R2,0[R0]
loop        load    R4,array[R2]
            add     R4,R4,R4
            store   R4,array[R2]
            add     R2,R1,R2
            cmplt   R10,R2,R15
            jumpt   R10,loop[R0]
            trap    R0,R0,R0
```


  - What does this loop do?
  - How many memory cycles would this loop involve?
  - If a primary memory read or write takes 10ns, how long will this program take to execute with no cache? (You may assume that an instruction takes exactly as long as the time to conduct all its memory cycles).
  - If a cache is introduced with an access time of 1ns, how many cache hits are there?
  - If the cache read or write time is 1ns, how long will the program take to execute.

Address Space

- The maximum amount of physical primary memory that can be attached to a computer is its physical address space
- The physical address space size is determined by the number of bits that can be carried on the address bus.
- If the address space is large there may be unpopulated regions of it where there is no physical memory (or I/O interfaces) installed. In modern computers this is often the case.
- The opposite problem however occurs where the address space is too small and even when fully populated there is not enough memory. Two examples:
  1. A successful family of computers was the DEC PDP line manufactured by Digital Equipment Corporation between about 1965—1980. But physical addresses were only 18-bits so physical address space was only 256Kbytes. As larger memory became feasible to build, this entire product line was obsoleted for this reason alone.
  2. The original IBM PC (1981) allowed access to physical memory of only 640Kbytes. This became a serious limitation before the end of the 1980s.

Hard Disks

- Hard disks are so-called secondary memory and are not directly addressable by the CPU.
  - Instead disks are treated as I/O devices and accessed via a special I/O interface called a disk controller.
- Electromechanical: data written on circular tracks on rotating disk. Track is divided into sectors (usually 512 bytes)
- To access data on a disk must first move the read head to the correct track. This is called seeking. Time taken is seek time and because of the mechanical nature it is slow.
  - Then wait for rotation of disk to bring requested sector under the head (rotational latency). Modern desktop disks spin at up to 7200 rpm.
  - At this point data can be streamed rapidly at a much faster data transfer rate. In a modern disk, rates of > 100MBytes/sec are possible
- Large modern disk size 6TB (6 x 10^{12} bytes), access time <10ms
- 1,000 times bigger than RAM, 1,000,000 times slower on access (perhaps 100 times slower on sustained transfer).
- Solid state disks (SSDs) are faster but lower capacity (<1TB) than mechanical drives.
  - Access times ~ 100 μs
  - DTR up to 600Mbytes/sec (still much slower than RAM)

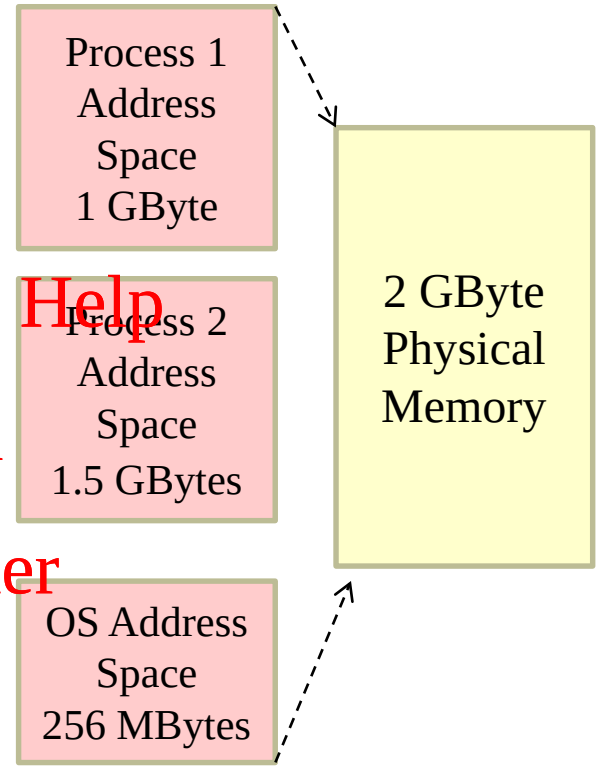
Above is a disk platter. Tracks are the concentric circles, each divided into sectors.

Tape Storage

- Tape storage systems are also treated by the computer as I/O devices and are another form of secondary memory.
- Common in large data centres, seldom used in small machines.
- Most common usage is for backup and for archiving of old data.
- In backup usage, all hard disks are copied to tape at regular intervals
  - Can mostly just copy changes (incremental backup)
  - Every so often do a full backup as a checkpoint
  - Sequence of full backups and incrementals is the backup schedule.
- A big tape installation has multiple slots that hold available tapes and a more limited number of tape drives. A robot arm selects the right tape and mounts it when needed
- Access time ~ minutes, capacity may be hundreds of TBs or more.

The Working Set

- Caching keeps instructions and data likely to be used again in fast memory local to the CPU.
  - The rest of the program and data for a process is normally in primary memory.
  - What if several processes are being run at the same time and the total physical primary memory is not enough to hold all their code and data (+ the OS)?
- Most (though not all) programs spend most of their time using a small portion of the address space available to them.
  - Programs have lots of loops and often operate repeatedly on a small group of variables and data structures. Often as little as 1% of the address space accounts for 99% of the accesses
  - This small portion of memory is the working set
  - The working set does change over time but usually not very quickly

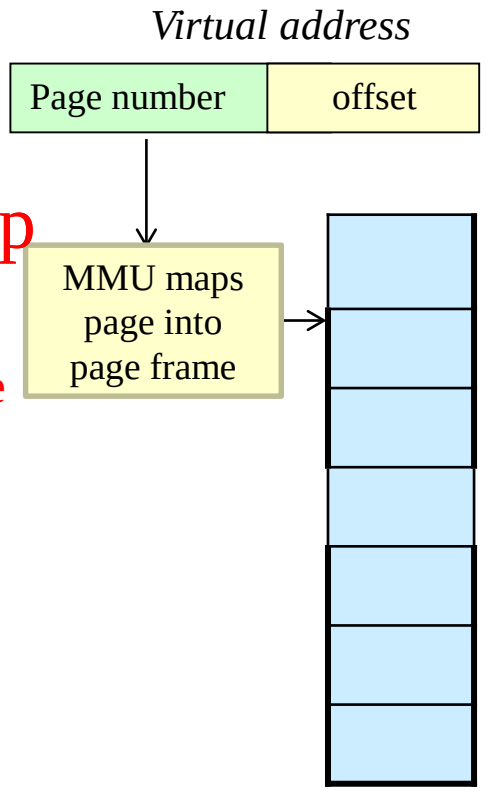


Virtual Memory

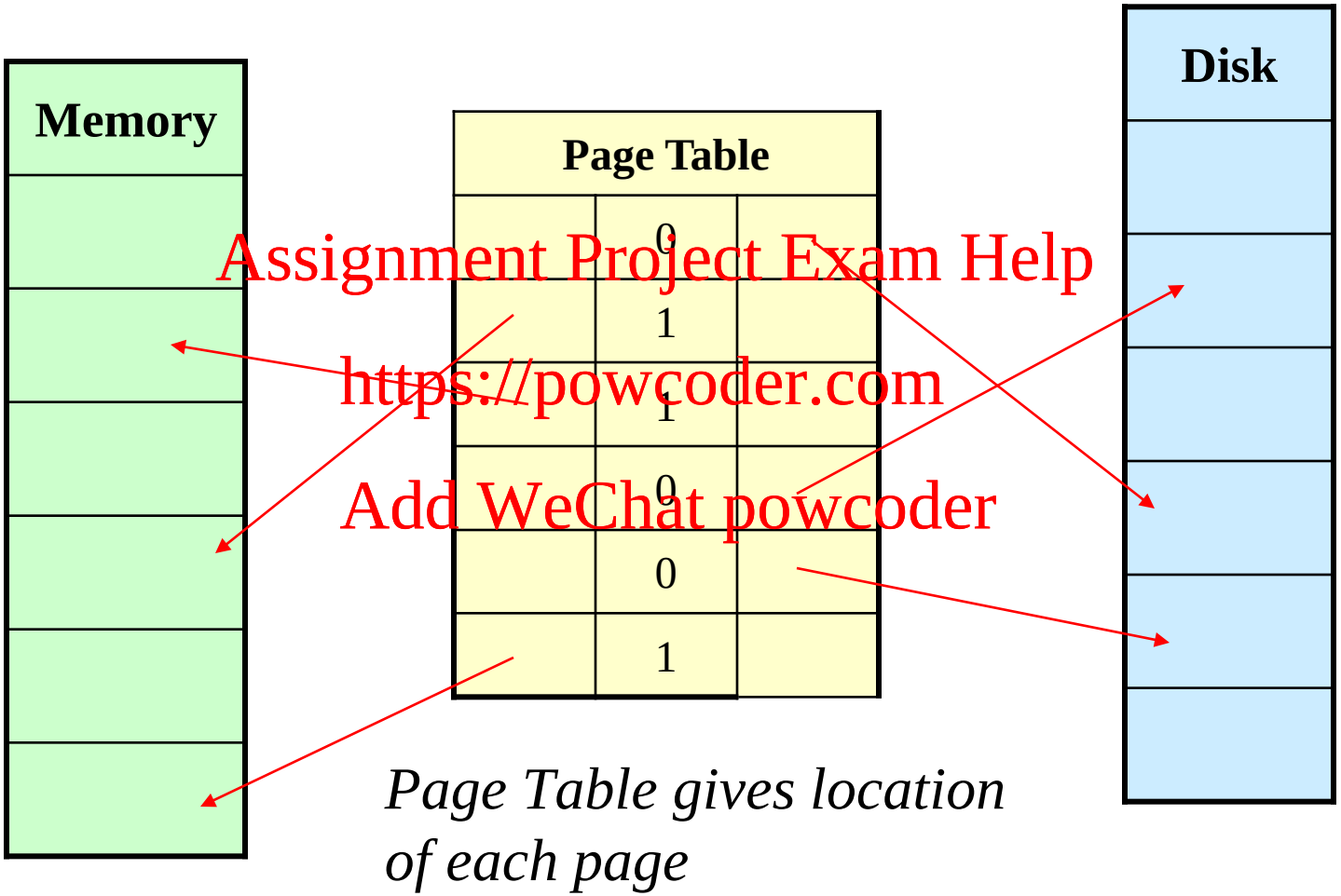
- Idea: keep the working set items in primary (physical) memory for all running processes, keep inactive code and data on disk
  - Transfer code data between memory and disk using I/O whenever necessary
  - While this could be done explicitly by each program it is better if it is automatic.
- In a virtual memory system, when a process needs a code or data item not in primary memory, process is suspended while the OS organises the transfer.
  - The idea is that the transfer is transparent to the process (performance hit though)
  - With virtual memory, a process can seem to have a fully populated address space even when there is not enough physical memory to make this possible.
- When the process tries to access an item (instruction or data) that is not in primary memory this is called a memory fault or a page fault.
- Memory faults must be detected by hardware (usually the memory management unit) and the access in question aborted.
- The OS must now get missing items from disk before letting process resume.

Pages

- In a virtual memory system, a process' address space is called a virtual address space. The actual address space containing the real memory is the physical address space.
- The virtual space of each process is divided into fixed size contiguous chunks called pages (~2Kbytes or 4Kbytes each)
  - Divide each virtual address into two fields: most significant is a page number, least significant an offset into the page
- The physical address space is divided up into blocks called page frames each able to hold one page
  - Some pages for a process need not be in primary memory even when that process is running
  - When page is loaded into memory it is placed in a free frame.
- OS (and MMU) must keep track of which page is in which frame (or where on disk)
  - Maintain page table of mappings to page frames or disk storage.
  - Each process needs a page table of its own.



The Page Table



Page Table gives location of each page

The Memory Management Unit

- The MMU sits between the CPU and the memory. It examines the page number in every virtual address and consults the page table. There are two possibilities
  1. *The page is in primary memory.* The MMU translates the virtual address to a physical one by replacing its page number with the number of the page frame containing it. This dynamic address translation is done by the MMU hardware and so fast that the memory cycle is not usually slowed at all.
  2. *The page is not in primary memory* and must be copied from disk before it can be accessed. This is a page fault.
- On a page fault, MMU sends a hardware memory fault signal to the CPU which aborts the memory cycle and jumps to a memory fault handler routine in the OS.
  - The handler routine transfers the required data from disk to memory and then allows the process to resume.
  - Very like an interrupt but memory faults occur in the middle of instructions. CPU must save all of its state (not just programmer's model) if process is to resume.
- By keeping virtual address spaces separate, MMU protects one process from another
  - However, if desired a page can be shared between two processes.
  - The OS can manipulate and bypass the MMU. It can therefore access anything.

Exceptions

- Interrupts and memory faults have much in common. The idea can be generalised as follows.
- An exception is a hardware or software event which causes the CPU to stop executing the current process and jump to an exception handler routine, usually in the OS.
- In addition to interrupts and memory faults other familiar exceptions include system reset and trap instructions.
- Every exception has its own handler. Since this is in the OS, exceptions force a control change to the OS. Traps are used to request OS services (including I/O).
- When the handler is finished, the suspended process may resume if the CPU state has been saved.
  - However, other processes may be allowed to run first
  - Some exceptions (e.g. Reset, some memory faults) do not allow a resume and abort any running process.