CPU, Memory & Disk: Limitations

* Status quo
  + CPUs execute an endless stream of instructions in memory
  + All system memory is in a contiguous physical address space
  + The disk is a finite set of blocks
  + All instructions execute in privileged mode
* To handle concurrent programs, the OS must separate the execution of different programs, providing the illusion to programs that each program is the only running program

Processes

* Definition: a process is an instance of the code we are running (the program)
  + One of the most profound ideas in CS
  + Not the same as “program” or “processor”
* Processes provide each program with two key abstractions
  + Logical control flow
    - Each program appears to have exclusive use of the CPU
    - Context switching
  + Private address space
    - Each program seems to have exclusive use of main memory
    - Virtual memory

Concurrent Processes

* Each process is a logical control flow
* Two processes run concurrently if their flows overlap in time
* Otherwise, they are sequential

Context switching

* Processes are managed by a shared chunk of memory-resident OS code called the kernel
  + The kernel is not a separate process but rather runs as part of some existing process
* Control flow passes from one process to another via context switching
* A context switch is a mechanism that allows the OD to store the current process state and switch to some other, previously stored context
* Reasons for a context switch:
  + The process completes/exits
  + The process is not using any CPU, so instead of waiting for it to finish, give the CPU to another process
  + The hardware requires OS help
  + The OS decides to stop the currently running process and start another to ensure all processes are getting started In a fair way

Process scheduling

* Scheduling has two aspects:
  + How to switch from one process to another
  + What process should run next
* How des the kernel switch from one process to another?
  + System has several ready processes
* How does the kernel stay in control?
  + Processes may yield() or execute I/O
  + Configurable timer interrupts let OS take control

Preemption

* If a task never gives up control, exits, or performs I/O, it could run forever ad the OS could not gain control
* Getting the control back from CPU is not easy
* The OS therefore sets a timer before scheduling a process
  + If the timer expires, the hardware interrupts the execution of the process and switches to the kernel
  + The kernel then decides if the process should continue

Process states

* Running, ready, blocked
* When a process becomes ready, it is scheduled to run, it does not run instantly because the CPU may be used
* Enters blocked state when it is waiting for something, and not actually using the CPU

Scheduling policy

* The scheduling policy determines which process should run next