Uh-Oh (2 of 2)

The CPU is idle most of the time! (picture not to scale)

Queries don’t run until earlier queries finish

Only one I/O request at a time is “in flight”

Queries 1, 2, and 3:
- CPU 1
- I/O 1
- CPU 1
- I/O 1
- CPU 1
- I/O 1
- CPU 1
- I/O 1
- CPU 1
- I/O 1
- CPU 2
- I/O 2
- CPU 2
- I/O 2
- CPU 2
- I/O 2
- CPU 3
- I/O 3
- CPU 3
- I/O 3
- CPU 3
- I/O 3

Sequential Can Be Inefficient

- Only one query is being processed at a time
  - All other queries queue up behind the first one
  - And clients queue up behind the queries ...

- Even while processing one query, the CPU is idle the vast majority of the time
  - It is blocked waiting for I/O to complete
    - Disk I/O can be very, very slow (10 million times slower ...)

- At most one I/O operation is in flight at a time
  - Missed opportunities to speed I/O up
    - Separate devices in parallel, better scheduling of a single device, etc.

Concurrency

- Our search engine could run concurrently:
  - Example: Execute queries one at a time, but issue I/O requests against different files/disk simultaneously
    - Could read from several index files at once, processing the I/O results as they arrive
  - Example: Our web server could execute multiple queries at the same time
    - While one is waiting for I/O, another can be executing on the CPU

- Concurrency != parallelism
  - Concurrency is doing multiple tasks at a time
  - Parallelism is executing multiple CPU instructions simultaneously

Threads vs. Processes

- In most modern OS’s:
  - A Process has a unique: address space, OS resources, & security attributes
  - A Thread has a unique: stack, stack pointer, program counter, & registers

  - Threads are the unit of scheduling and processes are their containers; every process has at least one thread running in it