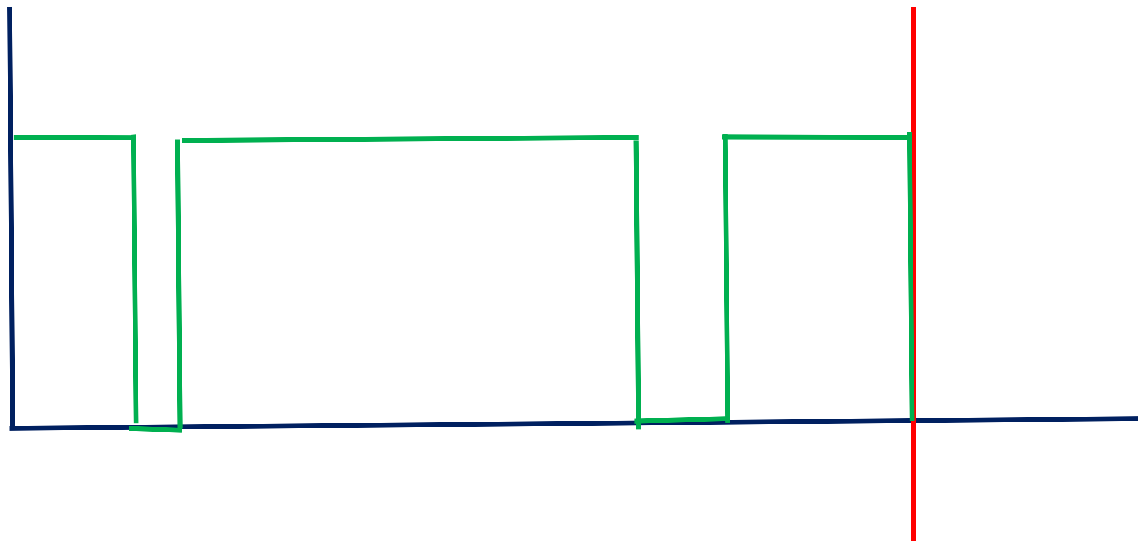
Performance and Reliability

Quality of Service (QoS)

- Reliability
- For example, for a system we are watching:



Performance

- Sometimes (often) the time it takes to do something will decide whether you will use a service or not, especially if there are alternatives
- For example:
  - Find/book air tickets
  - Buy things on-line
  - Search for information
  - …

Performance

- Performance (“speed”) is important
- We are all impatient

Performance

- Performance (“speed”) is important

- But, before we try to optimize “something” – a process,
- a method, an algorithm,

- We are reminded of Knuth’s “Law” of optimization…

Performance

- We are reminded of Knuth’s “Law” of optimization…

- Don’t

Performance

- We are reminded of Knuth’s “Law” of optimization…

- Actually.. He has said several variants of this,
- Usually it concerns “premature” optimization…
- Before you know if it really necessary

Performance

- First measure the performance
- Determine if you need to “optimize”
- Sometimes this isn’t necessary
- Or only a small improvement is needed

Performance

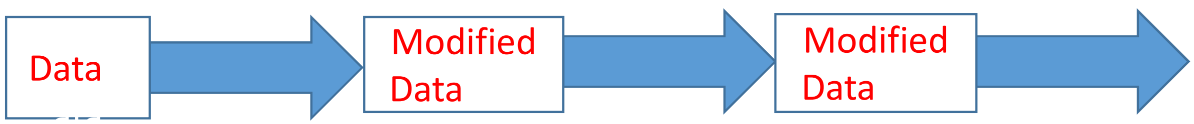
- From our perspective (very limited, for this class, now)

- We have a couple of paradigms, or methods, to exploit
- Parallelism or Concurrency

- Simple models:
  - Pipelining
  - Parallelism (here, “embarrassing parallelism”)

Performance

- We assume many CPUs, or “cores”, or even machines (computers)
- Pipelining
- Break a problem into pieces (parts) and do some work in one stage and pass results to the next stage



Performance

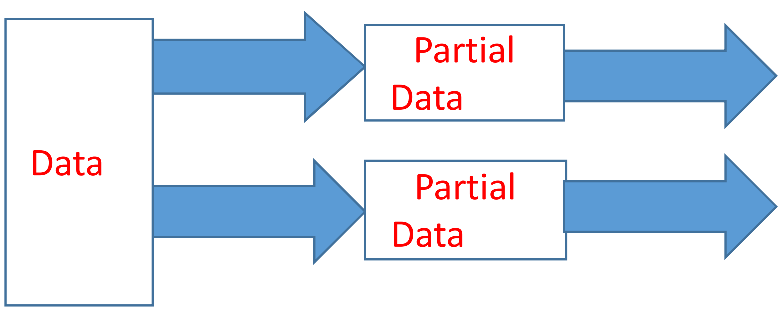
- We assume many CPUs, or “cores”, or even machines (computers)
- Pipelining
- For example
- There are seismographs all over the world collecting earthquake information. Each has a networked computer connected to it. If one, or several, detect a sufficiently strong movement, that information is passed on to another computer.
- That computer does some more processing and then send its information to the next computer, that will record it
- The next computer will sort, annotate, and record those events

Performance

- We assume many CPUs, or “cores”, or even machines (computers)
- Pipelining
- Each part of the processing can be done on a separate CPU, or computer, or core

Performance

- We assume many CPUs, or “cores”, or even machines (computers)
- Parallelism
- Break a problem into pieces (parts) and do some identical work on pieces (parts) or the data then possibly collect the results

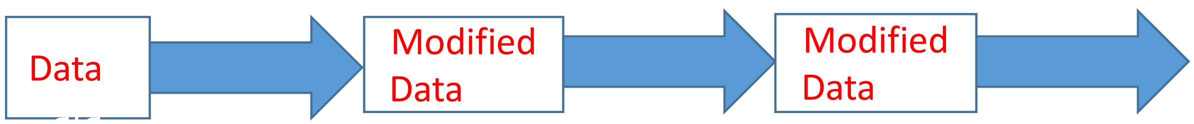


Performance

- We assume many CPUs, or “cores”, or even machines (computers)
- Parallelism
- For example
- We have millions of data values (for example, earthquakes)
- We want to find the largest one
- Split the data into two parts, use two computers or cores to find
- Maximum in each part,
- “Merge” results (largest of each part)

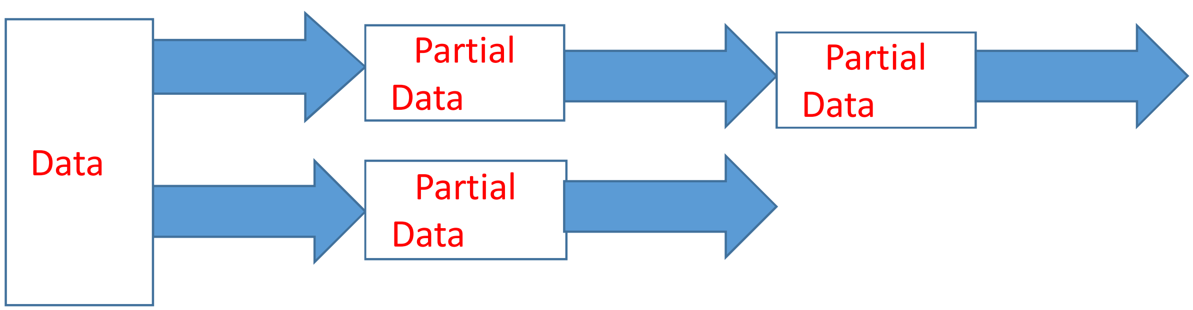
Performance

- We assume many CPUs, or “cores”, or even machines (computers)
- Pipelining and Parallelism
- Most problems can have both applied



Performance

- We assume many CPUs, or “cores”, or even machines (computers)
- Parallelism and Pipelining
- Most problems can benefit from both applied



Performance

- When interactions between data is more complex, finding and exploiting pipelining and parallelism becomes more complicated but usually (often) still worth time invested
- Need to carefully coordinate or synchronize processing activities so “next step” not done until previous step is complete (or “enough” complete)

Last

•