Both are networked

Parallel Computing vs. Distributed Computing

- Everyone is aware of each other
- LAN
- Operate as one unit
- tightly coupled
- If one node fails the entire framework will need to reset its hierarchy
- Computation will be much faster than Distributed

- Asynchronous
- Less coordination (Ad hoc)
- Larger latencies
- Can have great distances
- Not everybody knows everybody else
- WAN
- loosely coupled
- Code needs to be fault resistant
  Nodes can come on and offline
  ↳ Cannot assume all are persistent
- Often has no hierarchy due to the above observation.

Generic Metric to measure "power" of a computer (see top500.org)

↳ FLOPs => Floating Point Operations per second.

↳ only capture aspects of a machine's computation power

↳ IOPS => Integer Operations

↓

Can give you data on the machines themselves too CPU, framework, I/O etc

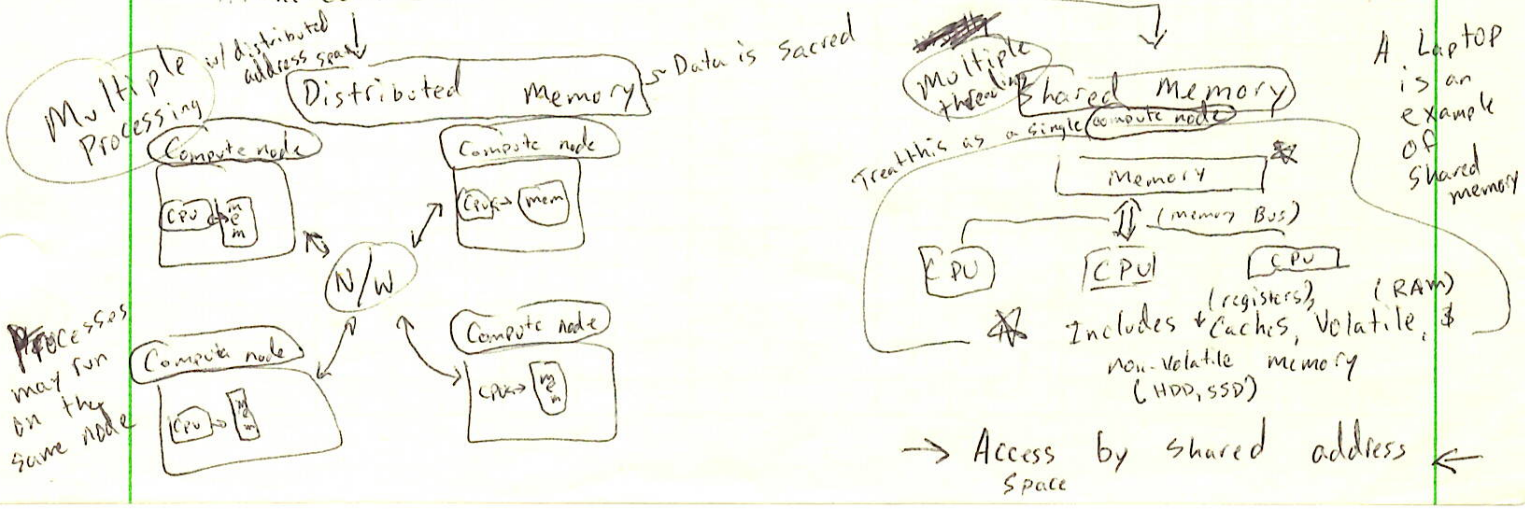
LINPACK → Dev'd in 1980s

↳ Linear Algebra, matrices, vectors etc

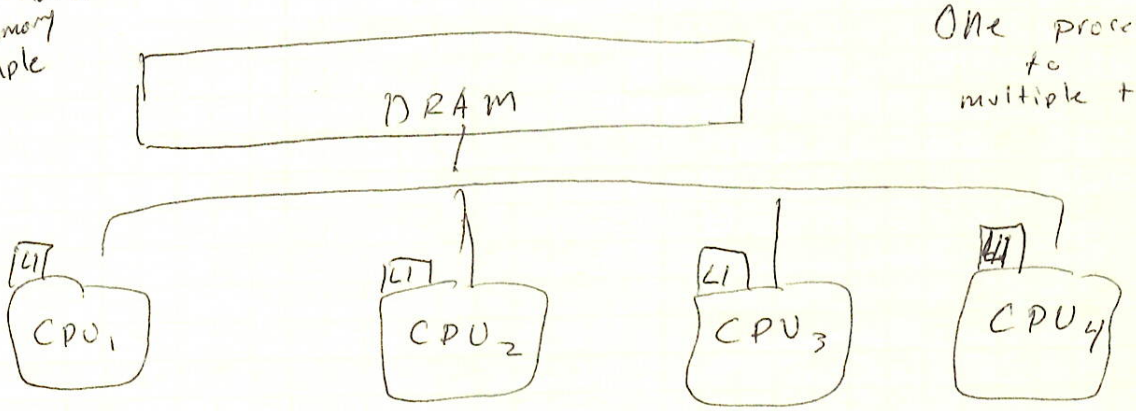
↳ Code to run on machines in order to rank the super computers

Parallel Computing Models

- Two different Architectures (in Parallel Computing)



Shared Memory Example



One process to multiple threads

For this model, when a given CPU wants to manipulate a variable and it's not found in any of its local caches it must retrieve the variable from main memory and then ~~then~~ it may manipulate the variable

An Aside Note

In an example for an Intel CPU, when the above event happens

Intel has a directory set up to flag the particular variable that it's being written such that when another CPU wants to read from the variable being manipulated, the directory will force the CPU working on that variable to flush its current variable in question to the main memory so the other CPU may read the variable.

Distributed memory Architecture

Incur overhead of communication between computed nodes.

Shared Memory Architecture

Risk the danger of corrupting other cpu's Data

Process & Thread are both software entities

ie. to do threading/process on a single CPU, you will incur timesharing

Parallel programming models.

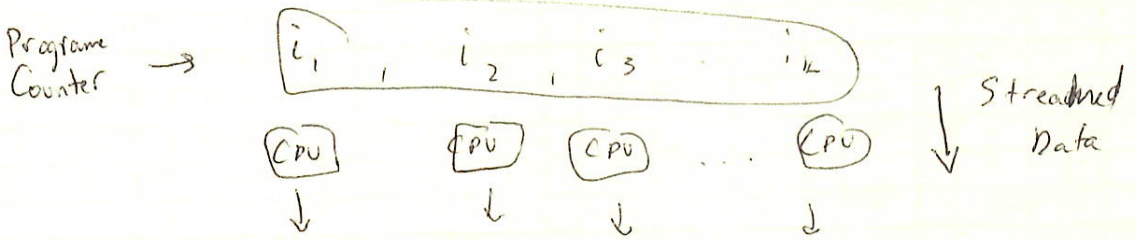
(1) Single Instruction (PC) Single Data (SISD) (Serial)

No concurrency

(2) Single Instruction multiple Data (SIMD) (Data Parallelism)

Think quicksort! (Divide and conquer on big data set)

→ Always assume that you have the most CPU's possible

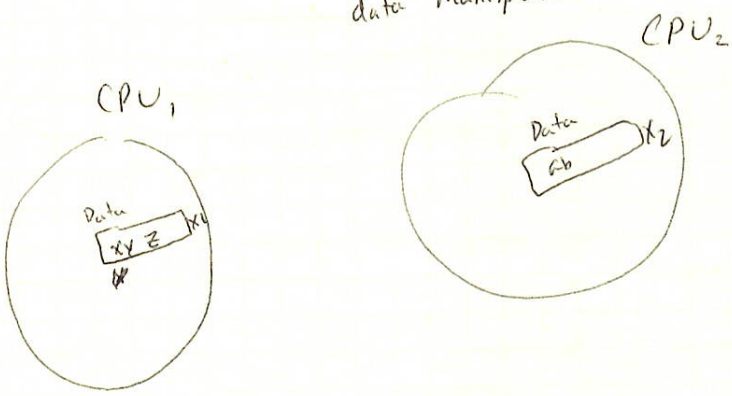


"Vector processing" Architecture ⟹ Mapping Data to processes

(3) Multiple Instruction multiple Data (MIMD)

You will have as many Instruction pointers as CPU's

Rate of execution will be variable (and data manipulation)



asynchronous execution. (varying from clock tick to clock tick)

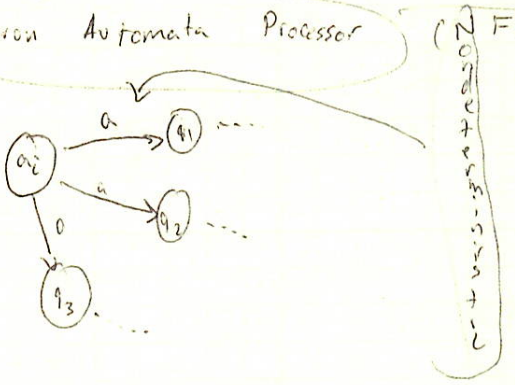
multiprocessing

Parallel Programming Models

Multiple Instruction Single Data (MISD)

No real good use case of these

Micron Automata Processor (NFAs)



Read one input symbol executed different paths

SISD
SIMD
MIMD
MISD

} 4 ways to write a parallel program

Combine

[Shared Memory / Distributed Memory] X [SISD, SIMD, MIMD, MISD]

- SISD, SIMD → maps both shared memory
- MIMD → maps both
- MISD → distributed memory but not good

Software!!!

Process vs Thread

Can hold multiple threads

- Independent
- Can see what other threads (within same CPU) are doing to its local memory (shared memory)

# of threads should equal number of CPUs

CPU's are simply given processes or threads, it doesn't care.