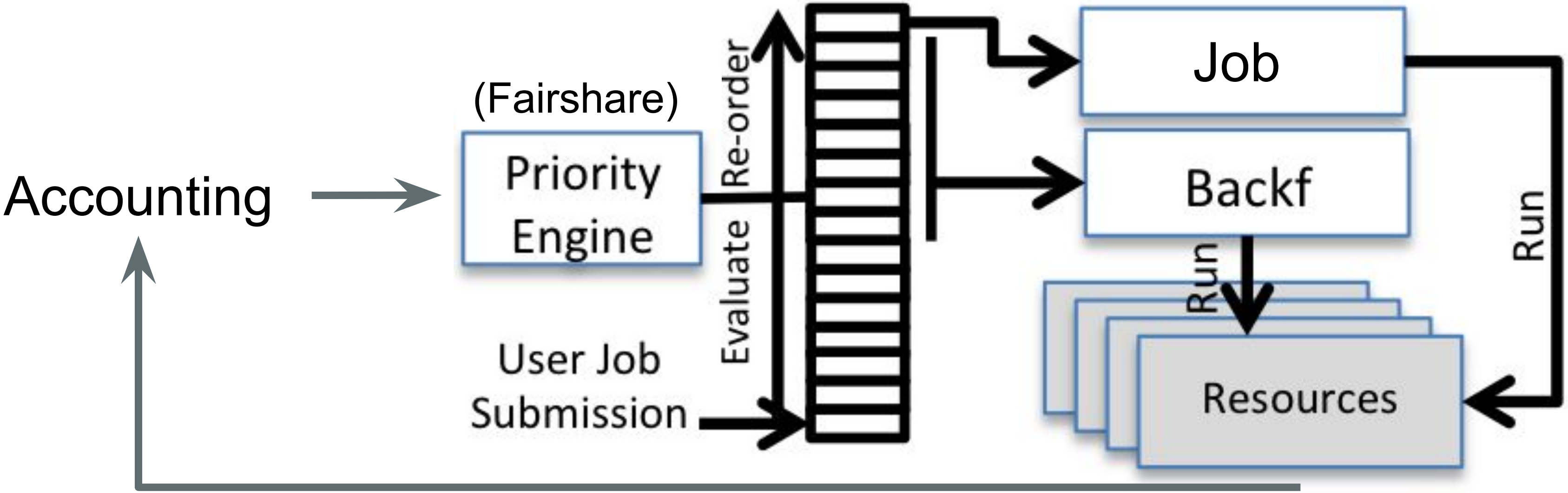
The Concept of Parallelism

The utilization challenge

- On a typical HCP system
  - ~1000 active users per year
  - Millions of submitted jobs per year
  - Many different shapes of jobs (single core to thousands, minutes to days)
  - Special requests / requirements
  - Maintenance & upgrades
- > 95% utilization
- Reasonable queue times

The utilization challenge

- Job scheduling system (@NSC = SLURM)



The utilization challenge

- Tip: It pays to understand the scheduling policy for the system you are using
  - e.g. Tetralith scheduling

The utilization challenge

- Remember: The queue times you experience are not solely dependent on you

The utilization challenge

- Tip: Monitor your individual and project resource usage (core hour and storage)
  - @NSC:
    - SUPR
    - Command line tool: `$ projinfo`

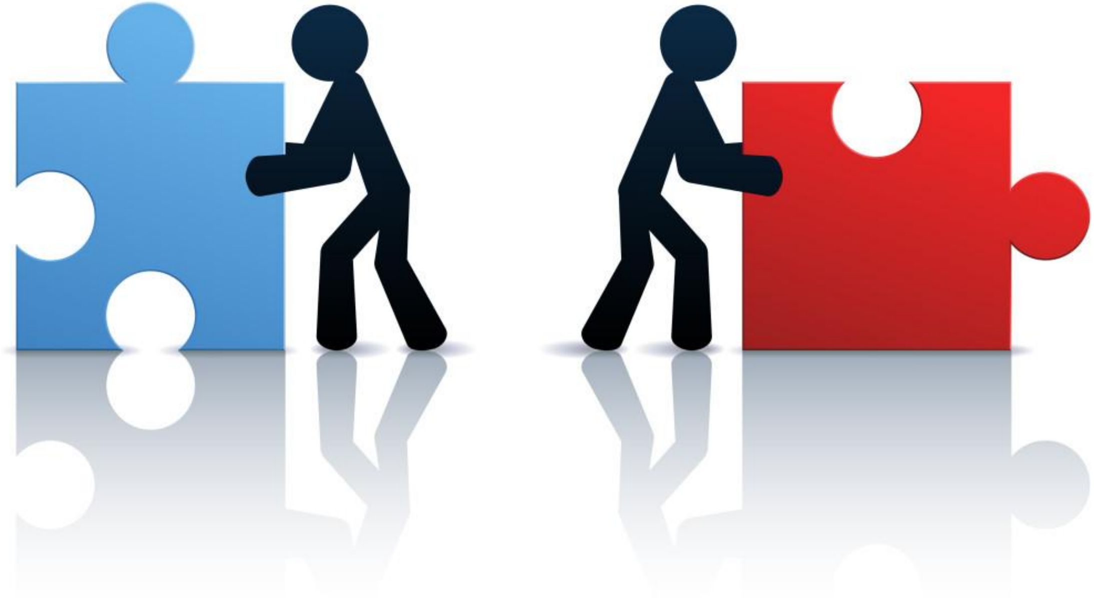
Parallelism models

Shared memory parallelism model

Shared memory parallelism

- Work is divided between multiple threads running on a single machine
- Each thread has access to common (shared) memory
  - e.g. OpenMP

Distributed memory parallelism model



Distributed memory parallelism

- A set of tasks (or processes) that use their own local memory during computation.
- Multiple tasks can reside on the same physical machine or across an arbitrary number of machines.
- Tasks exchange data through communications by sending and receiving messages.
  - MPI is the industry standard for message passing

Parallel programming models

- Remember:
  - Threads = shared memory (OpenMP)
  - Tasks (processes) = distributed memory (MPI)

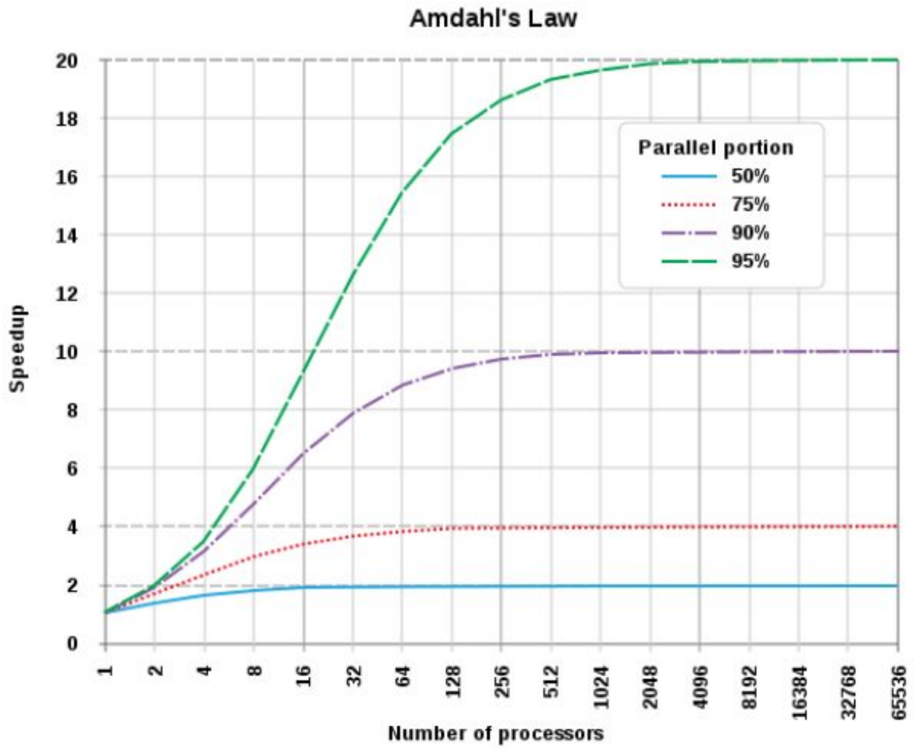
- Other parallel programming models exist, e.g.
  - MPI + OpenMP (hybrid)
  - MPI + Cuda (CPU + GPU)
  - Data parallel model
  - …

Amdahl's Law

Obey Amdahl – it's the law.

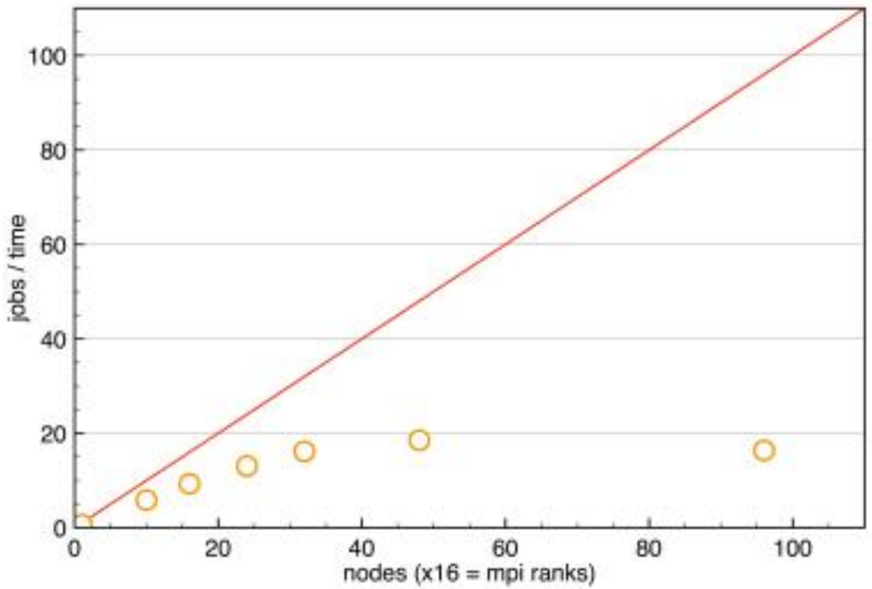
Amdahl's Law

Predicts the theoretical speedup when using multiple processors.



Practical example: VASP

- VASP (Vienna ab initio Simulation Package) scaling example
- Scaling characteristics depend on:
  - Problem size
  - Model configuration
  - HPC system
  - ...



Scenario 1: Starting new

Scenario 1: Starting new

- Before starting with a new job type or new model
  a. Test run the software
     - Verify the output: does it produce scientifically reasonable results
     - Verify the job launches correctly and uses the allocated resources in a sensible way
  b. Perform (at least) a simple scaling analysis

Simple scaling analysis

- A minimal scaling analysis can save you vast amounts of core hours
    a. Tool your runscript to time your simulation
    b. Run an initial best guess number of cores (n)
    c. Run the same test on half the number of cores (n/2)
    d. S = (time: n/2 cores) / (time: n cores)

S=2.0 👍 S=1.5 🤔 S=1.0 😢

Scenario 2: Inheriting

Scenario 2: Inheriting

- If you inherit code, scripts or configuration files from someone else
    a. Follow scenario 1