CSC 7700: Scientific Computing

Module D: Simulations and Application Frameworks

Lecture 1: Simulation Basics

Dr Peter Diener

Center for Computation and Technology
Louisiana State University, Baton Rouge, LA

November 15, 2013

1. Goals
2. Simulations
3. Supercomputers

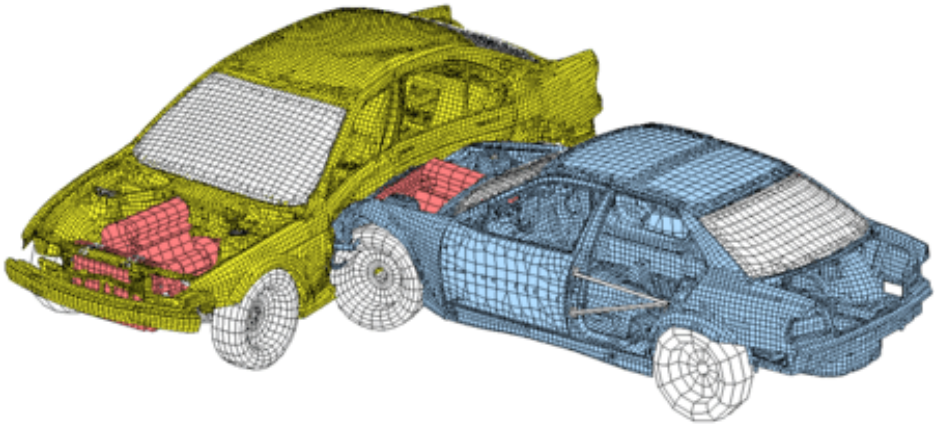
Goals

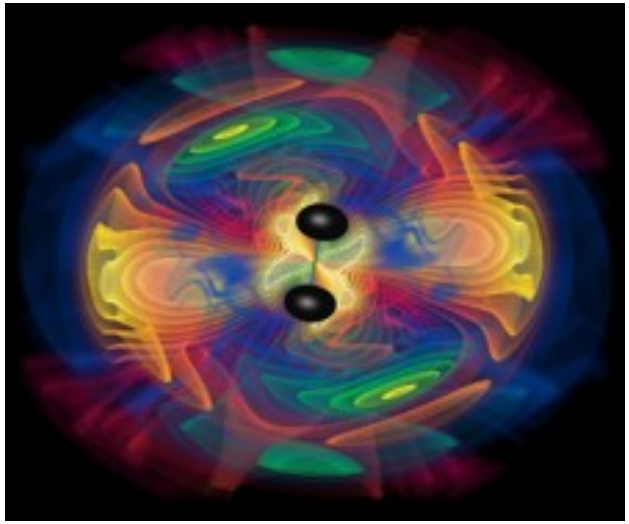
Goals

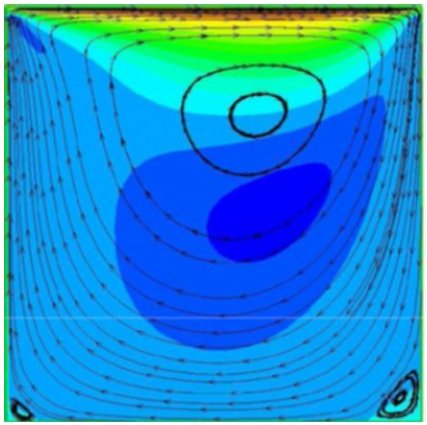
- The module *Simulations and Application Frameworks* will teach:
  - what a simulation is,
  - how a typical simulation code looks like,
  - how it is used in practice,
  - and what some of the major concerns in such a code are.
- We will use Cactus as an example of an Application Framework.

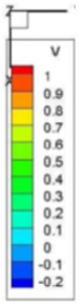
Simulations

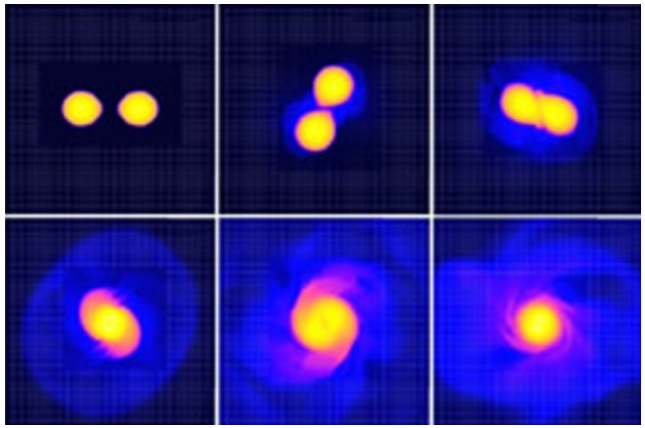
What is a simulation?

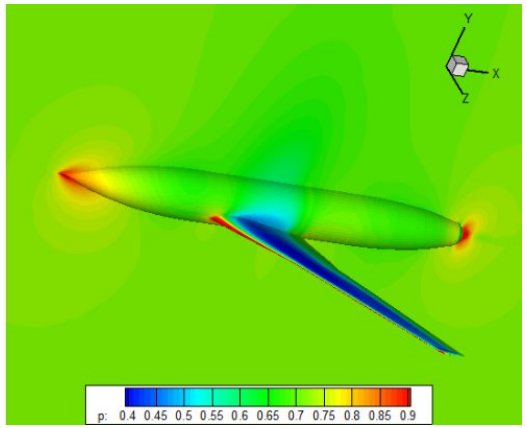








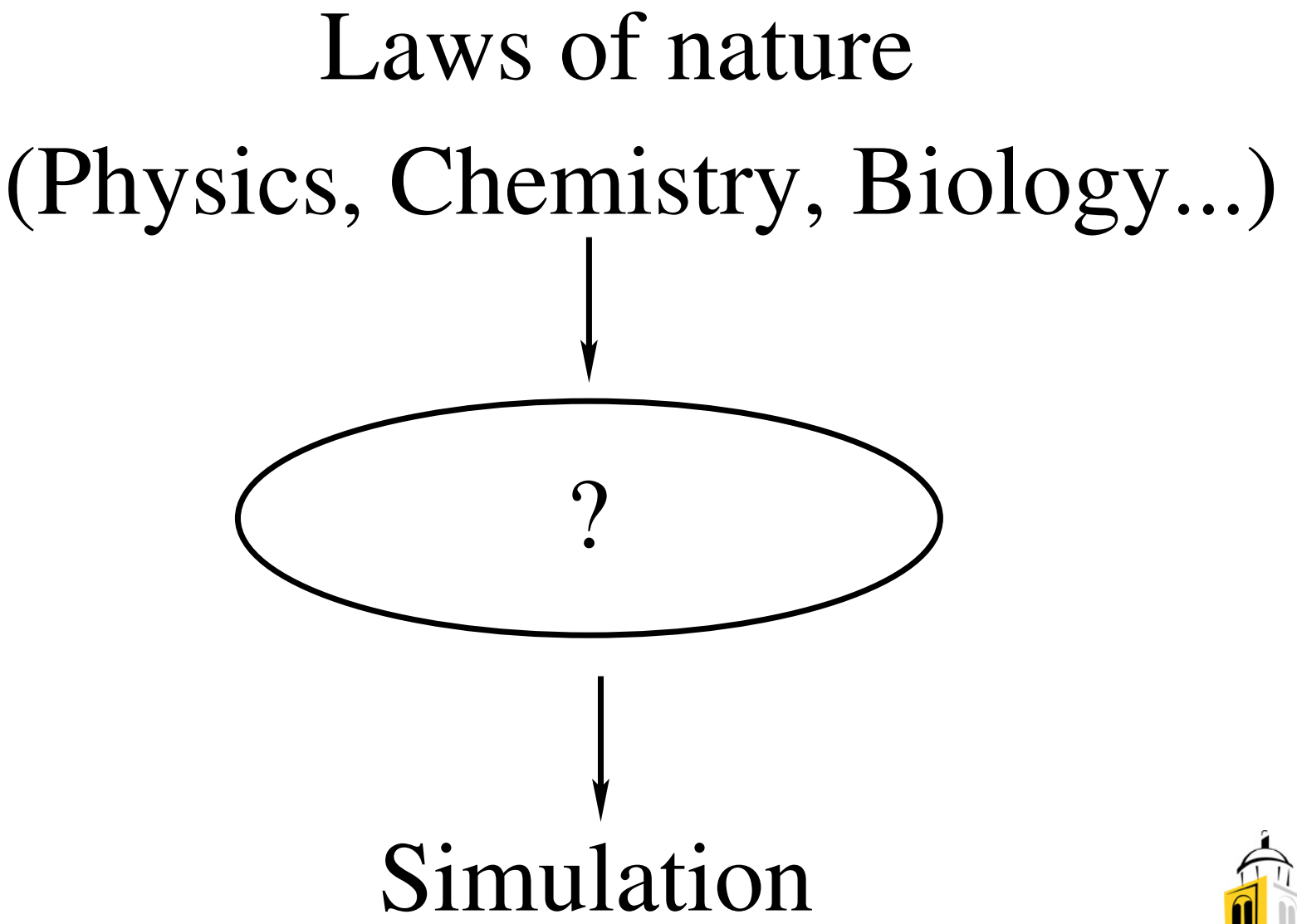




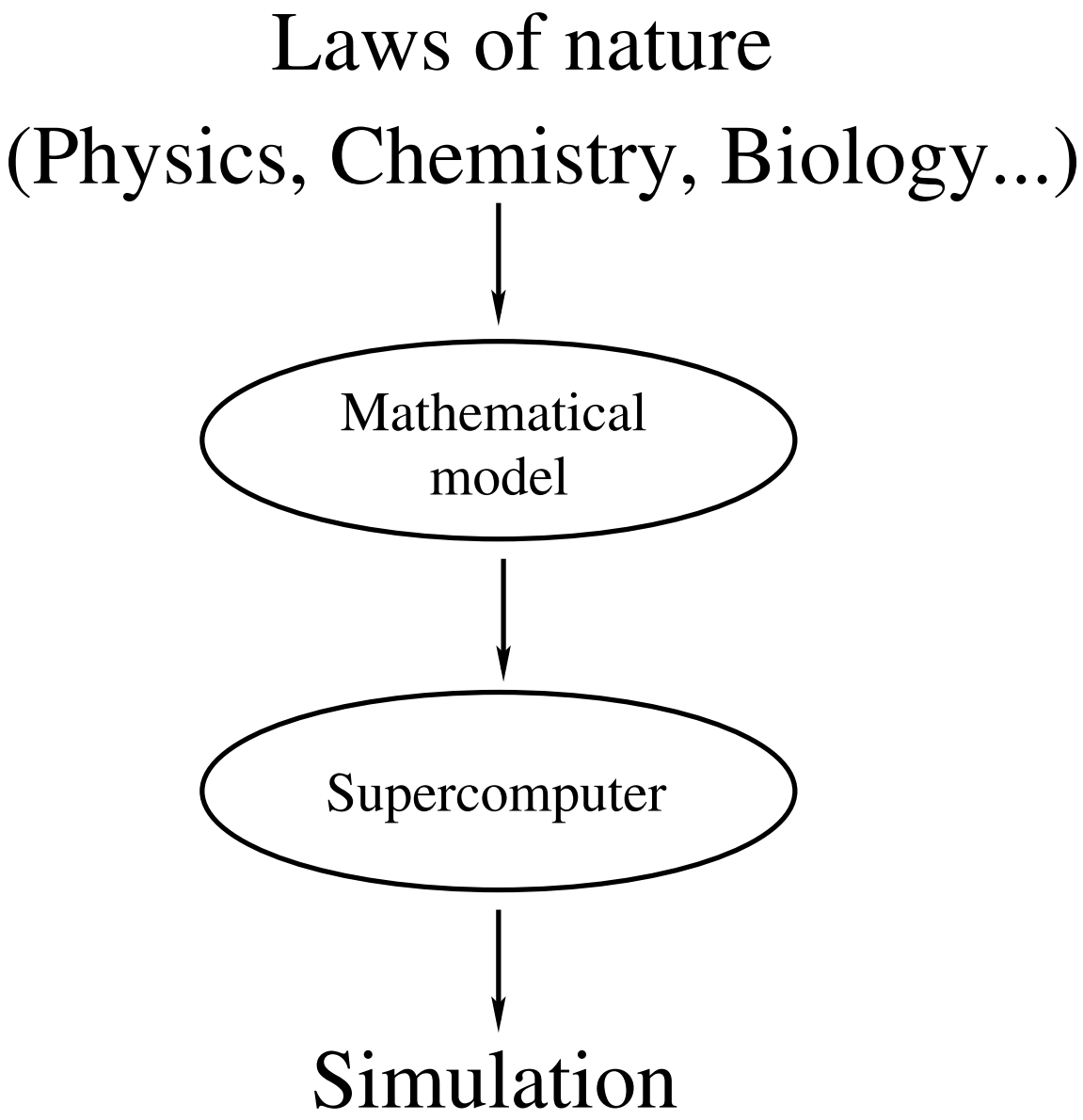
What is a simulation?

- Flame propagation in combustion engine: *understand* behavior that is too fast or too small.
- Hurricane modeling: *predict* behavior.
- Car crash testing: *engineer* better devices.
- Video games: *create* a fantasy world.
- Black hole collisions: *test* the basic laws of nature.

What is a simulation?



What is a simulation?



Ingredients of a simulation

- From the laws of nature derive a mathematical model relating the variables describing the system you want to simulate. This often involves approximations.
- Decide on a discretisation scheme to use for your variables and your mathematical equations (more later).
- Write a computer code to solve the discretised equations.
- Setup initial data and run the simulation.
- Analyze and visualize the results.

Example: The wave equation

Consider a fluid in 1D described at time t and position x by the density ρ, pressure p and velocity u.
From Newton's first law ($ma = F$) we have

$$\rho \frac{\partial u}{\partial t} = -\frac{\partial P}{\partial x}.$$

On the other hand if the velocity u varies with position x the pressure will change

$$\frac{\partial P}{\partial t} = -K \frac{\partial u}{\partial x},$$

where K is the incompressibility of the fluid.
Assuming that K and ρ does not vary with position we can combine the equations into one equation for the pressure P

$$\frac{\partial^2 P}{\partial t^2} = \frac{K}{\rho} \frac{\partial^2 P}{\partial x^2}.$$

The Initial Value Problem

The wave equation is a very simple example of a very large class of computational problems encountered in physics, chemistry and biology: The Initial Value Problem (IVP).

- Typically expressed in terms of a set of PDEs (partial differential equations) that tells us how a system is changing given a known state of the system.
- Starting from a set of *Initial Conditions* we can then simulate the behavior of the system by evolving from one state to another in finite timesteps.

Discretisation

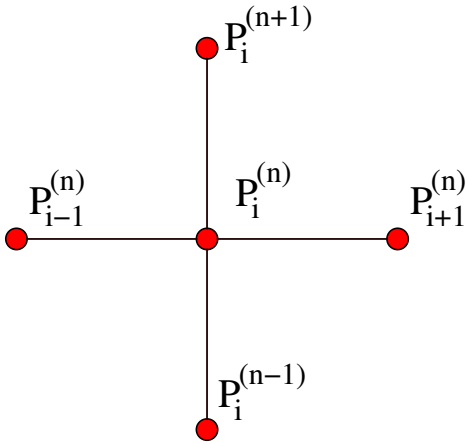
- It would require an infinite amount of information to describe the complete state of continuum system (air, water, car body, etc.).
- Instead we select a finite set of discrete points where we assign values to the field variables in order to reduce the number of degrees of freedom.
- There a many different ways to do this
  - Finite differences (sample solution on a regular grid)
  - Finite elements (small rigid triangles or tetrahedra)
  - Finite volumes (small squares or cubes)
  - Expansion of fields in a finite number of basis functions.
  - Particles

Discretisation

- Using a finite (instead of infinite) number of degrees of freedom introduces an approximation error.
- The accuracy of the approximation depends on the discretisation scheme and the resolution (how closely is the continuum sampled).
- For any discretisation scheme we can establish the order of accuracy. E.g. fourth order: $\epsilon(h) = O(h^4)$.
- Validation of simulation results requires careful convergence studies.
- The order of accuracy only gives a reliable measure of the error when the resolution is high enough (the convergent regime).

Discretisation example: The wave equation

Approximating the pressure $P(x,t)$ with a grid function $P_i^{(n)}$.



$$\frac{\partial^2 P}{\partial t^2} = v^2 \frac{\partial^2 P}{\partial x^2}$$

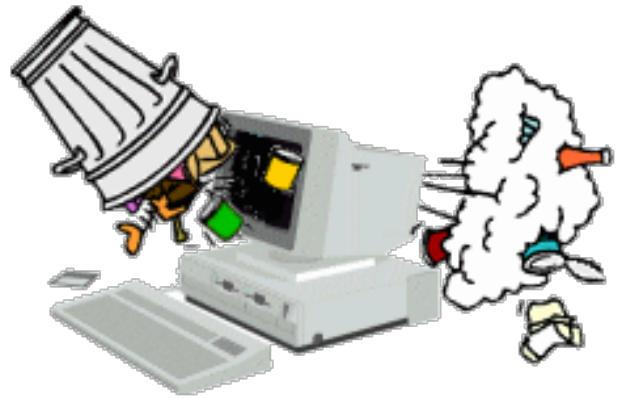
$$\Downarrow$$

$$\frac{P_i^{(n+1)} - 2P_i^{(n)} + P_i^{(n-1)}}{\Delta t^2} = v^2 \frac{P_{i+1}^{(n)} - 2P_i^{(n)} + P_{i-1}^{(n)}}{\Delta x^2}$$

The error from this time and space discretisation is $O(\Delta x^2)$.

Caveat

- Some systems are not described by PDEs but other types of mathematical relations.
- Sometimes it is the end state of a system that is interesting (like an equilibrium configuration) and not how it got there.
- If the initial data is an approximation or a guess, the result of a simulation may not be reliable.



Simulation Overview

- Derive a mathematical model for the system you want to simulate.
- Discretise the resulting PDEs (or whatever other form they are in).
- Decide on the shape and size of the computational domain (normally it's impossible to simulate the whole universe).
- Decide on the resolution.
- Decide on how to handle the boundaries of the computational domain.
- Set up initial conditions.
- Evolve using many small timesteps.
- Output data along the way.
- Analyse and visualize the data.
- Write the paper.

Structure of a Simulation Code

- Variables (solution) stored in large "state vectors"
  - There can be many (e.g. billions) elements in the state vector
  - Best handled in efficient container structure like a Fortran array, C or C++ vector or a tree structure.
  - The code needs constructs to iterate over these elements or subsets of elements.
- Routines to set up the initial condition.
- Routines to perform many identical time steps.
- I/O methods to write solution to disk and optionally checkpoint the simulation.
- Should be able to run as batch job without user intervention.

Initial Data

- Analytical initial data can just be calculated when the simulation starts.
- Numerical initial data may be solved internally at the beginning or be read in from file.
- Can be set up from reading in a checkpoint file (continuation run).
- Initial data can be large:
  1 billion elements,
  25 variables/element,
  8 Byte/variable: total 200 GByte.

Parallel Computing

- Cannot store the solution on a single node; parallel programming using MPI is necessary.
- At the moment only Fortran, C and C++ are viable languages for programming a supercomputer.
- Research is underway in other (maybe simpler) ways like Unified Parallel C, Co-Array Fortran or HPX (here at LSU)
- Simulations take long:
  1 billion elements,
  1000 Flop/element/step,
  1 million steps,
  CPU speed 10 GFlop/s: total 28,000 CPU hours (3.2 CPU years), or 12 days when running on 100 CPUs.

Batch Processing

- Since simulations take so long we cannot supervise them manually
  - Cannot be awake at all times.
  - A user error can destroy weeks of data.
  - Supercomputers are expensive; cannot waste valuable resources waiting for the next user input.
- Need to *plan* simulations carefully ahead of time, then let them run automatically.

Batch Processing

- Need to *plan* simulations carefully ahead of time, then let them run automatically...
- ...so that each error is only discovered days or weeks later!
- Using a supercomputer thus requires much expertise, experience, patience and a high tolerance for frustration.
- *Using supercomputers are more difficult than it really should be.*

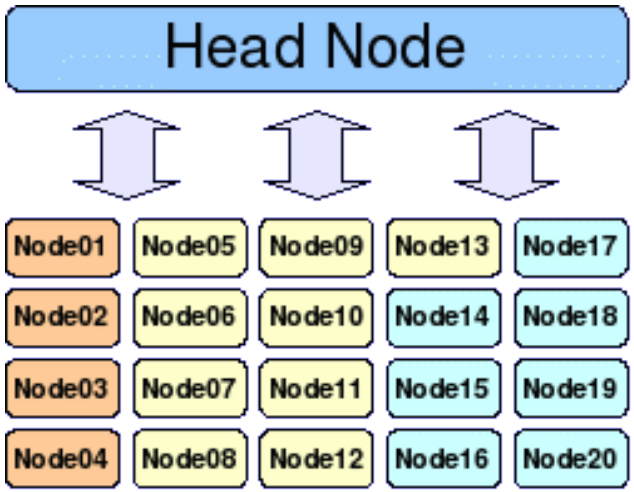
Supercomputers

Fast vs. Large

- Supercomputers are not so much fast as they are large.
- They are not interactive (like a notebook or workstation), they operate in batch mode.
- Their hardware is complex.

Remote Access

- Supercomputers are located far away, need to use ssh/gsissh to access.
- Log in is to *front end (head node)* only, usually a large workstation.
- Cannot (or should not) use front end to run simulations.



File Systems

- Supercomputers need large and fast file systems to store simulation data often many 100 TByte or PByte.
- For management and performance reasons the file systems are usually split into different parts with different properties.

  Home directory GBytes per user, many small files, backed up.

  Data directory TBytes per user, few large files, backed up, tape backend.

  Scratch directory No quota, few large files, often automatically deleted.
- This is done differently on each supercomputer – read documentation!

Compute Nodes and Interconnect

- Most supercomputers have a *cluster* architecture with many interconnected compute nodes.
- Each node may have multiple cores (4 to 64), similar to a large workstation typically with 1GByte to 4GByte of memory per core.
- Nodes may additionally have GPU's or Intel Xeon Phi's.
- Nodes are connected via a low-latency *communication network* (e.g. Infiniband or other proprietary technologies).
- Overall system has from a few up to 10,000 nodes. The largest Supercomputer in the world currently has 3,120,000 cores in 16,000 nodes (Intel Xeon IvyBridge processors and Xeon Phi).
- My personal scale:

$$\begin{aligned} \text{small} &< \text{2k cores} \\ \text{medium} &< \text{20k cores} \\ \text{large} &> \text{20k cores} \end{aligned}$$

Batch System

- Cannot (or should not) use compute nodes directly.
- Need to submit *job* to *batch system*, requesting N nodes for T amount of time.
  - ...wait (a few days?)...
  - ...then the job start to run (most likely while you're asleep)...
  - ...and then you discover your errors...
- There is always a run time limit (e.g. 8, 12, 24, 48, 72 hours)
  - ...which is inconvenient when you need to run for 2 weeks: checkpoint/restart is necessary.
- The batch system ensures that jobs get run in an order that keeps the supercomputer as busy as possible at all times.

Allocations

- Need to ensure fair use of supercomputers and prevent individual users from monopolising it.
- Need to ensure that supercomputer time are used for important research and by codes that can run efficiently.
- Allocation proposals for the national supercomputers are therefore peer reviewed.
- The allocation review panel then decides how much time each research project will be allowed to use in the coming allocation period.
- 1 CPU hour costs about 5 cents (10 cents on Amazon ECC).
- With this metric, Super Mike II produces about $352 worth of CPU time every hour.

Software

- Installed/available software is system dependent, not just standard Unix systems.
- Therefore cannot just install binaries, need to build software for each supercomputer.
- Installed software typically consists of compilers, MPI libraries, Scientific libraries (like GSL, BLAS/LAPACK, PETSC, etc.), perl, python, etc.
- Most interactions with a supercomputer takes place on the command line. Some GUIs, Portals exist, but typically limited in functionality.

Supercomputer Steps

- Obtain an *Allocation*.
- Log in to the *Front End*.
- Compile your code on the *Front End*.
- Submit your jobs on the *Front End* to the *Batch Queue*.
- Simulations then execute on the *Compute Nodes* connected via a *Communication Network*.
- Data stored in *File Systems* can then be analyzed on the supercomputer itself or transferred to other places for analysis.
- Data can be stored permanently in *Tape Archives*.