CSC 7700: Scientific Computing
Module C: Simulations and Application Frameworks
Lecture 1: Simulation Science Basics

Dr. Erik Schnetter

Goal

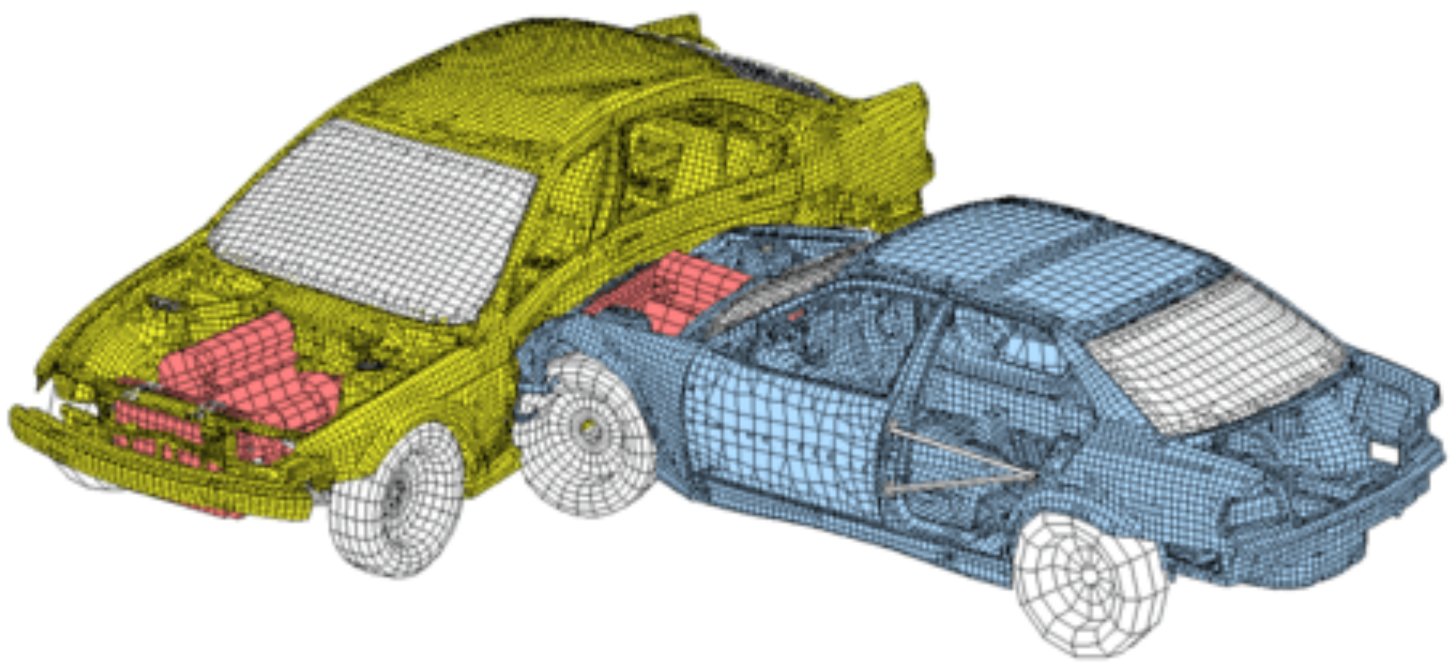
- This module *Simulations and Application Frameworks* will teach
  - how a typical simulation code looks like,
  - how it is used in practice (by physicists or engineers),
  - and what some of the major concerns in such a code are.

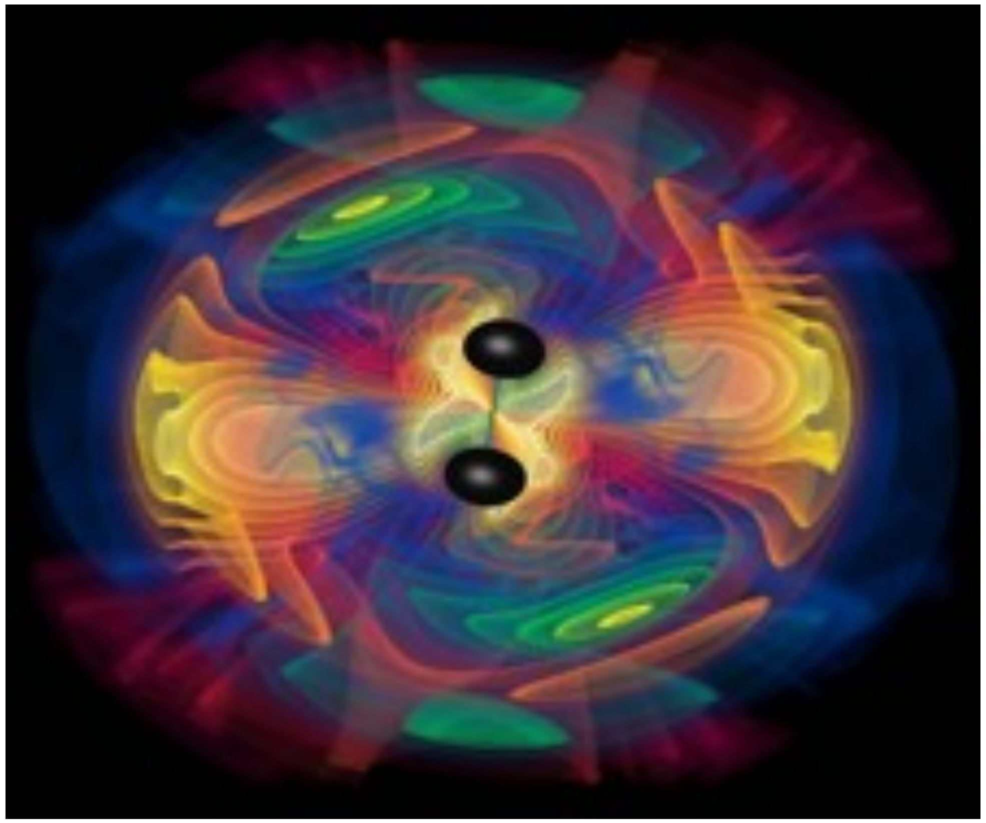
Literature

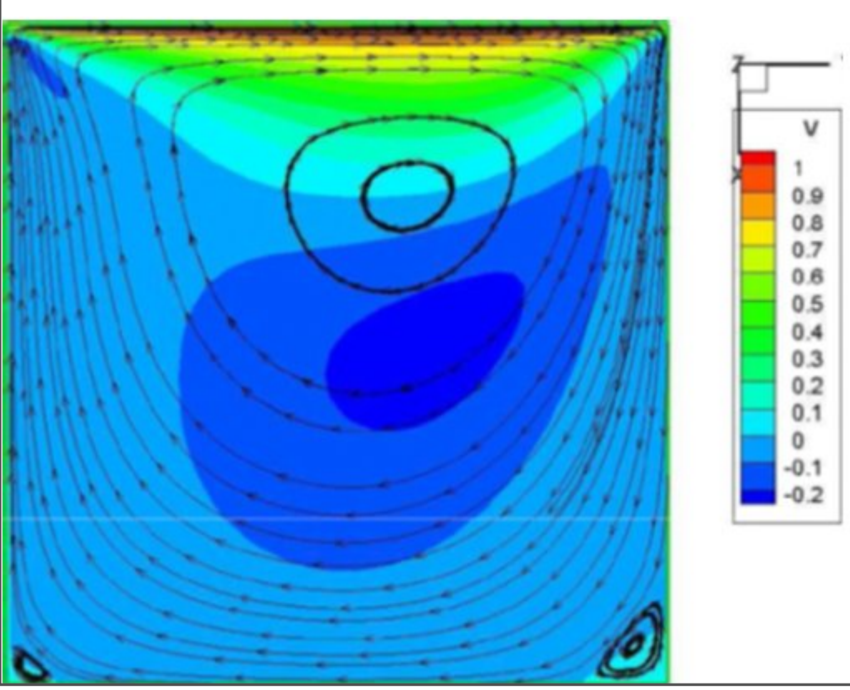
- Article: T. Goodale et al., *The Cactus Framework and Toolkit: Design and Applications*
- Book: M. T. Heath, *Scientific Computing: An Introductory Survey*
- Documentation: Cactus Users' Guide
- Tutorial: Einstein Toolkit Tutorial
    - Details see wiki page

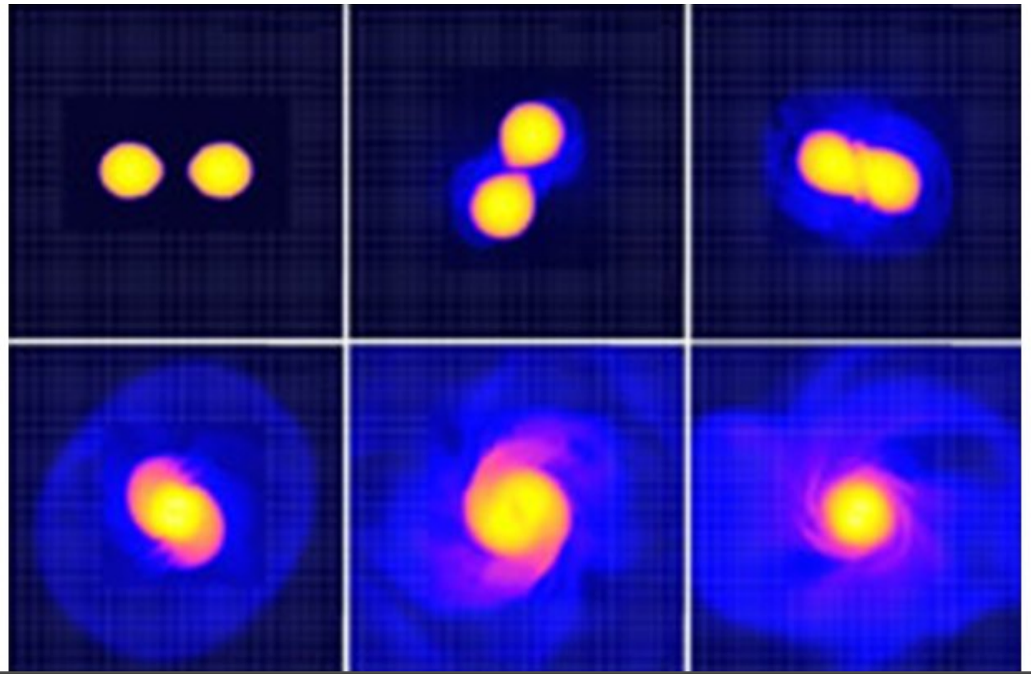
From Physics to Simulation

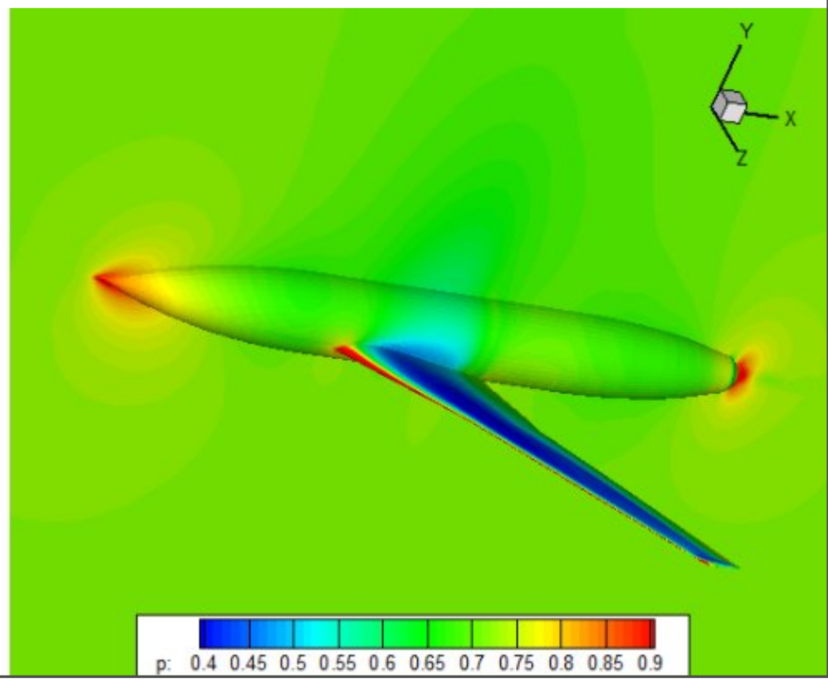
Simulations











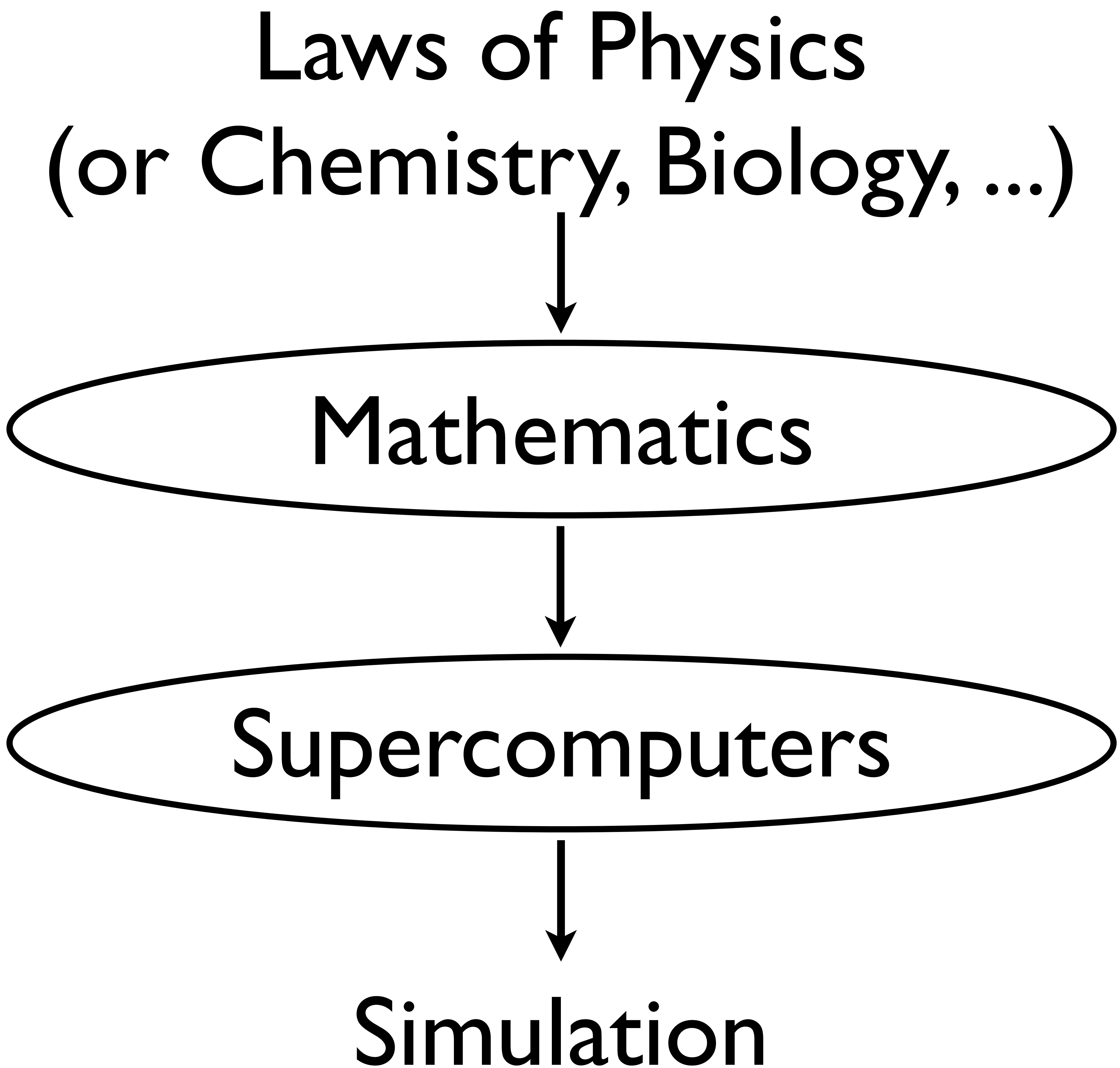
Why Use Simulations?

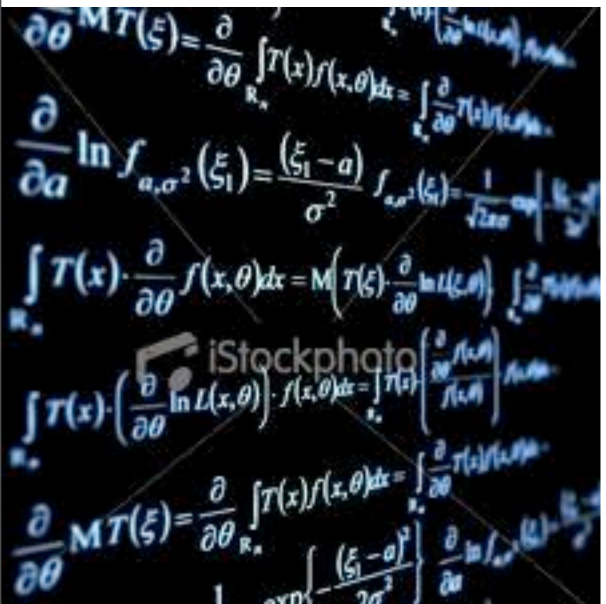
- Flame propagation in combustion engine: *understand* behaviour that is too fast or too small
- Hurricane modelling: *predict* behaviour
- Car crash testing: *engineer* better devices
- Video games: *create* a fantasy world similar to the real one

Laws of Physics
(or Chemistry, Biology, ...)

?

Simulation


Laws of Physics
(or Chemistry, Biology, ...)
Mathematics
Supercomputers
Simulation



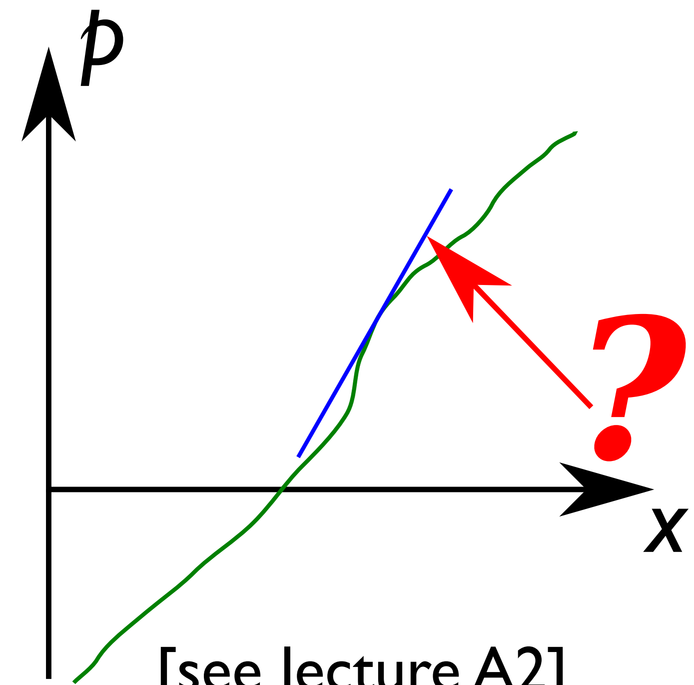
- The physics that is to be simulated is expressed in "the language of Mathematics"
  - Called *Scientific Computing* or *Numerical Analysis*

- The resulting systems of equations are solved on large computers
  - Called *Supercomputers* because they are as large and awkward as a supertanker

Systems and Equations

- The state of a system is described via variables (density, velocity, pressure, etc.)
- Laws of Physics can then often be described via *PDEs* (Partial Differential Equations)
- A PDE describes how a system is *changing* depending on its current *state*

PDE Example



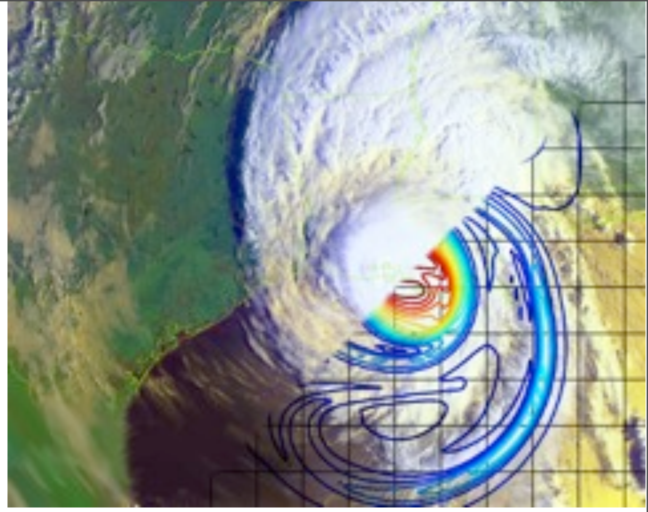
Euler equation: (hydrodynamics) $$\frac{Dv}{Dt} = -\frac{1}{\rho}\frac{\partial p}{\partial x}$$

- ρ: density, *p*: pressure, *v*: velocity; *t*: time, *x*: position
- Interpretation:
  Consider a small chunk of matter. If the pressure to its left and right are different (∂p), then it will be accelerated (Dv).
- This assumes a co-moving coordinate system (Lagrangian picture). In a fixed coordinate system, additional terms will appear in the Euler equation.

PDE Interpretation

- A *PDE* (partial differential equation) tells us how a system is changing, if we know the state of the system
- Starting from an *Initial Condition*, we can thus simulate the behaviour of a system by meticulously tracking how the system is changing

Discretisation



- PDEs describe continuum systems (car body, water, air); these have infinitely many degrees of freedom
- Reduce complexity by approximation via a *discrete system* instead
- Compare e.g. pixels on a TV screen, surface triangulation for visualisation

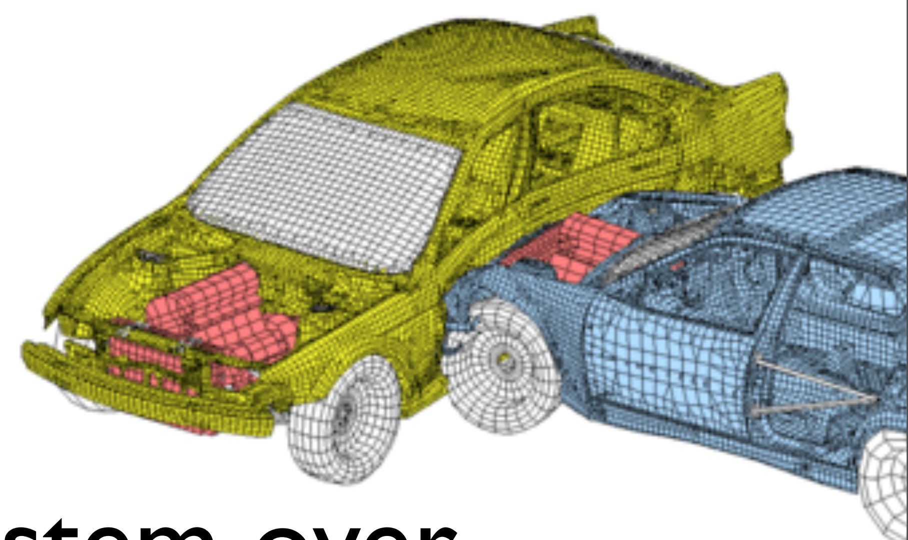
- Many possibilities:
- finite elements (e.g. small rigid triangles)
- finite volumes (e.g. small cubes)
- finite differences (sample solution on regular grid)
- particles (small chunks of matter)
- many more...

Discretisation Error

- Discretising is an approximation and thus leads to an error
- Can use a finer discretisation (higher *resolution*) to reduce this error
- *Order of Accuracy* describes how this error scales with the resolution , e.g.
  fourth order: $E = O(h^4)$
  doubling resolution reduces error by 16

Simulation Procedure

- Choose PDE that describes system well
- Discretise PDE
- Set up initial condition
- Follow each element of the system over many many tiny steps
- A simulation can have billions of elements with millions of steps, taking weeks of computing time



Caveat

- Some systems are described not by PDEs but otherwise (e.g. coupled ODEs, discrete transitions)
- Sometimes not time evolution is interesting, but e.g. equilibrium configuration
- Usually (in real life), PDEs and initial conditions are only *approximations or guesses*, and simulation results *may not be reliable*

Garbage In,
Garbage Out

Connection to Other Modules

- Some systems are not described by PDEs: **Distributed Scientific Computing**
- Simulations produce large output files (billions of elements, millions of steps): **Networks and Data**
- To understand results, need to "undo" formulation as PDE and discretisation: **Scientific Visualisation**

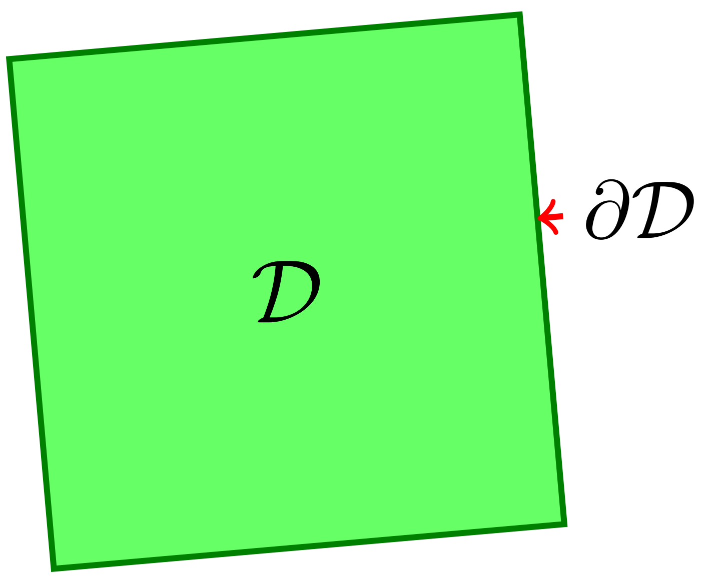
Ingredients of a Simulation

Key Concepts in a Numerical Simulation

- Simulation Domain: the part of the world that is simulated, often just a small box
- Resolution: the accuracy of the discretisation; higher is better (and more expensive)
- Evolution System: (discretised) PDE system
- Initial Condition: initial state
- Boundary Condition: what to do at the (artificial?) domain boundaries, often also PDEs
- Output Variables: which part of the solution should be output -- it is often too expensive to output everything

WaveToy Thorn: Wave Equation

For a given source function $S(x, y, z, t)$ find a scalar wave field $\varphi(x, y, z, t)$ inside the domain $\mathcal{D}$ with a boundary condition:



- inside $\mathcal{D}$:

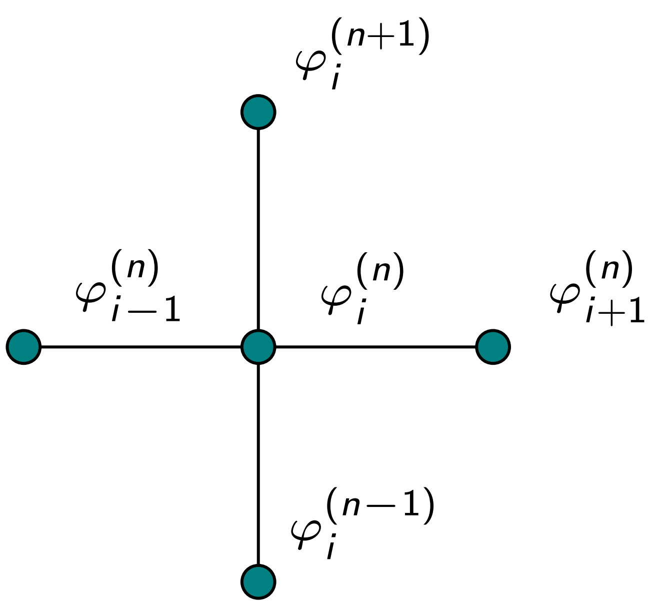
$$\frac{\partial^2 \varphi}{\partial t^2} = c^2 \Delta\varphi + S$$

- on the boundary $\partial\mathcal{D}$:

$$\varphi|_{\partial\mathcal{D}} = \varphi(t = 0)$$

WaveToy Thorn: Discretization

Discretization:
approximating continuous function $\varphi(x,t)$ with a grid function $\varphi_i^{(n)}$:



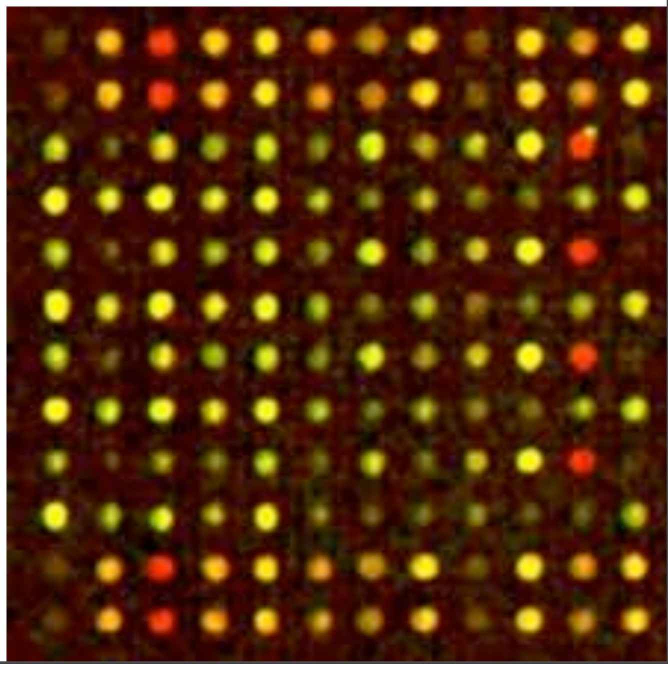
$$\frac{\partial^2 \varphi}{\partial t^2} = c^2(\partial_x^2 \varphi) + S$$
$$\Downarrow (c \equiv 1)$$
$$\frac{\varphi_i^{(n+1)} - 2\varphi_i^{(n)} + \varphi_i^{(n-1)}}{2\Delta t^2} = \frac{\varphi_{i+1}^{(n)} - 2\varphi_i^{(n)} + \varphi_{i-1}^{(n)}}{2\Delta x^2} + S_i^{(n)}$$

Basic Structure of a Simulation Code

- State (solution) stored in large "vectors"
- Routine to set up initial condition
- Routine to perform many identical steps (applying discretised PDE or similar)
- I/O methods to write solution to disk
- Run as batch job without user interaction

Storing the Solution

- After discretising a PDE, one obtains many (...billions...) very similar elements (cells, points, particles, ...)
- Best handled in efficient container structure: Fortran array, C++ vector, etc. (maybe also tree structure)
- Code contains many constructs that iterate over these elements



Initial Condition

- Often generated by external method, then read in from file
- Can also checkpoint, and then restart where previous simulation left off
- Initial data can be large; example:
  1 billion elements,
  5 variables/element,
  8 Byte/variable: total 40 GByte

Parallel Computing

- Cannot store solution on a single node; parallel programming via MPI is a must
- These days, only Fortran, C and C++ are viable languages for programming a supercomputer
- There is research in other, simpler ways, e.g. Unified Parallel C, Co-Array Fortran, or ParalleX (here at LSU)

(Time) Stepping

- Performing many identical steps to arrive at the solution
- Simulations can take long; example:
  1 billion elements,
  1000 Flop/element per step,
  1 million steps,
  CPU speed 10 GFlop/sec:
  total 28,000 CPU hours (3.2 CPU years),
  or 12 days when running on 100 CPUs

Batch Processing

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

Batch Processing 2

- Need to *plan* simulations carefully ahead of time, then let them run automatically...
- ...so that each error is only discovered weeks later!
- Using a supercomputer thus requires much expertise and experience, patience, and a high tolerance for frustration
- **This points to a large problem in supercomputer usability these days**

Ingredients of a Simulation

- Many simulation programs have a similar structure
- This structure is determined by the physics description (PDEs and discretisation)
- A simulation handles many small elements of data, and iterates over them many times

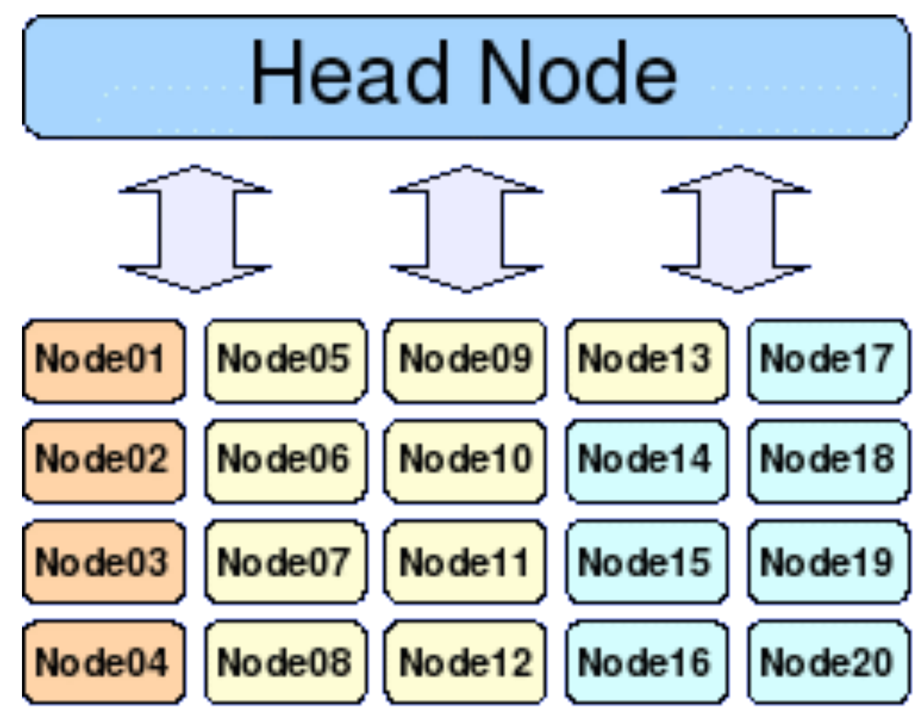
Ingredients of a Supercomputer

Fast vs. Large

- Supercomputers are not fast, they are large
- They are not interactive (like a notebook or workstation), they operate in batch mode
- Their hardware is complex -- I am going to describe the user's point of view only here

Remote Access

- Supercomputers are located in far away places, 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 file systems to store simulation data, often many 100 TByte
- For management and performance reasons, usually split into different parts with different properties
- Different on each supercomputer -- read documentation!

- <u>Home directory:</u> GBytes per user, many small files, backed up
- <u>Data directory:</u> TBytes per user, few large files, backed up, tape backend
- <u>Scratch directory:</u> no quota, few large files, often automatically deleted

Compute Nodes, Interconnect

- Most supercomputers have a *cluster* architecture with many compute nodes
- Each node has (4 to 32?) cores, similar to a large workstation
- Nodes are connected via a low-latency *communication network* (e.g. Infiniband)
- Overall system has (128 to >8,000?) nodes, or up to 100k cores
- My personal scale:
  <1k cores: small,
  <10k cores: medium
  >10k cores: large

Batch System

- Cannot (or should not) use compute nodes directly
- Need to submit *job* to *batch system*, requesting *N* nodes...
- ... wait (a few days?) ...
- ... then the job runs
- (... and then one discovers one's errors)

- There is a run time limit, often 24h or 48h
- ... which is inconvenient if one needs to run for 2 weeks: checkpoint/restart
- Batch systems ensure that a supercomputer is not idle; there are always jobs waiting to be executed

Allocations

- Need to ensure fair use of supercomputer, prevent individual users from monopolising it
- Typically, an *allocation process* decides who can use how much of a supercomputer's time during a year (similar to writing a grant proposal)
- 1 CPU hour costs about 5 cents (10 cents on Amazon ECC)
- With this metric, Queen Bee produces about $270 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 manually (or ask administrators to do that)
- HPC developers often prefer command line tools, don't use GUIs (which may not be available)
- (But: Eclipse and PTP may change this)

Ingredients of a Supercomputer

- Obtaining an *Allocation*,
- Logging in to a *Front End*,
- Submitting jobs to a *Batch System*,
- Simulation executes on *Compute Nodes* connected via a *Communication Network*,
- Storing data in various *File Systems*.

Sample Session: Einstein Toolkit

Tutorial

- The tutorial instructions are at http://docs.einsteintoolkit.org/et-docs/Tutorial_for_New_Users
- Note: These instructions require an *account* and an *allocation* on Queen Bee. Obtaining these may take several business days!

Homework

- Follow these instructions. Skip the "Additional ..." parts.
- Write a report detailing how many cores the simulation used, how much CPU time it required, and how much disk space the simulation output occupies.
- State which allocation you used, and how long you needed to wait in the queue.
- Include the gnuplot graphs in your report.