PHY 604: Computational Methods in Physics and Astrophysics II

Cyrus Dreyer
cyrus.dreyer@stonybrook.edu

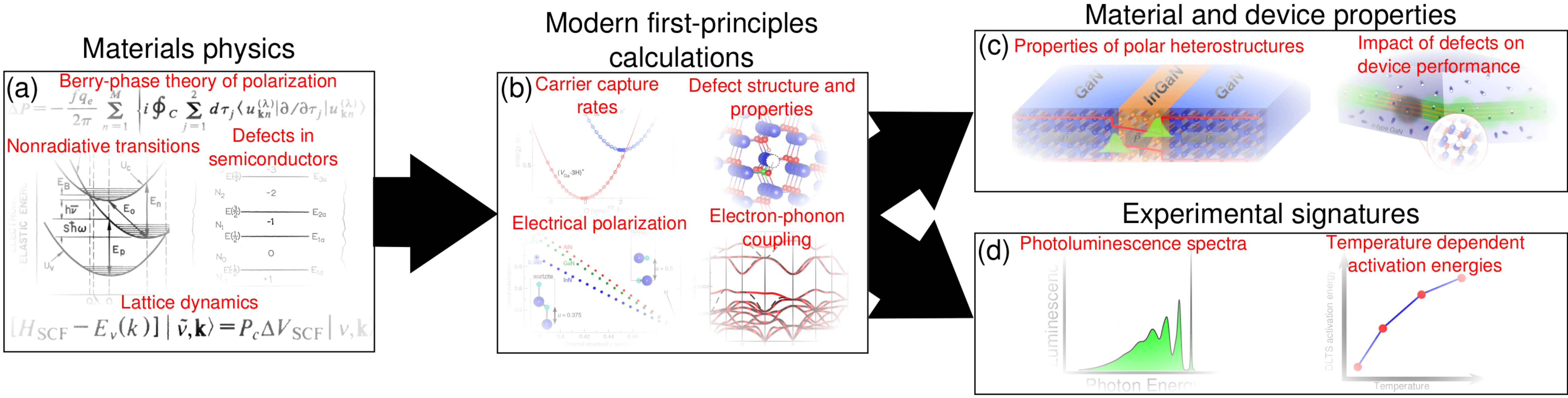
Fall 2023

My research interests: Computational condensed matter physics

Stony Brook University

FLATIRON INSTITUTE
Center for Computational Quantum Physics

https://www.simonsfoundation.org/flatiron/



https://you.stonybrook.edu/cdreyer/

Goals of the course:

- Learn how to solve problems in physics computationally
- Understand the limitations of numerical methods
- Have the ability to interpret numerical results presented in the literature
- Have exposure to computational tools
- Understand basic idea behind algorithms for performing common computational tasks

Technical points about the class: Programming Languages

- The assignments will involve writing computer programs
- You may use the programming language of your choice.* I would prefer:
  - Fortran
  - C++
  - Matlab
  - python
- * In general, and especially if your language is not on the list, you should provide some help for how to compile (if necessary) and run your code
- Examples will be given in fortran, C++, and python

Technical points about the class: Topics covered

- Basics of computation and programming constructions
- Good programming practices
- Numerical differentiation and integration
- Interpolation and root finding
- Ordinary differential equations
- Linear algebra
- Fast Fourier transforms
- Fitting
- Partial differential equations
- Monte Carlo techniques
- Genetic algorithms
- Parallel computing
- Machine learning

Technical points about the class: Class location

- Vote: Move the class to Physics building (likely B131)

Technical points about the class: Assignments

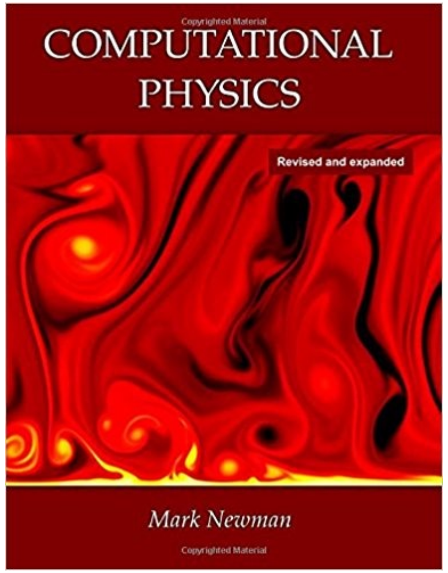
- Coding homework will be assigned roughly every two weeks
  - Homeworks will be 80% of the final grade
  - Will involve code and written analysis
  - **Recommendation** (not required)**: Use Jupyter notebooks**

- Proposed office hours: Mondays, 3:00pm to 4:00pm; Thursdays, 10:00am to 1:00pm
  - Please feel free to come to me for help!

- There will be a final project at the end of the semester
  - Solve a physics problem computationally
  - Write up a short report, and present to the class
  - Final project is 20% of the final grade

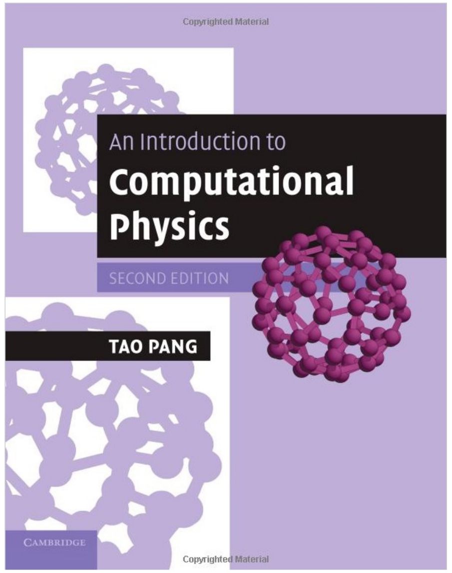
Technical points about the class: Textbooks

- **No textbook is required for this course**
  - Some recommended texts for further reading:



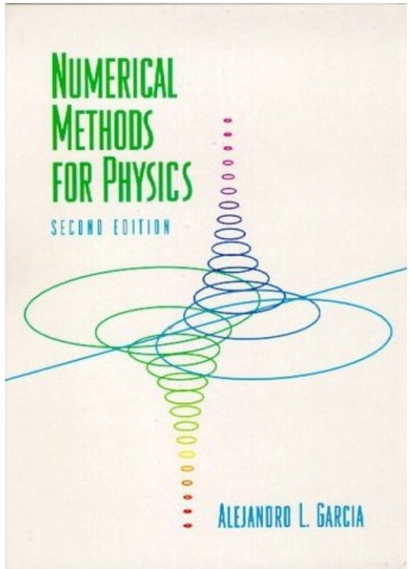
***Computational Physics*, by Mark Newman**

- Generally good coverage on most of the topics we'll discuss
- Lots of physics examples
- Inexpensive
- Main recommended book



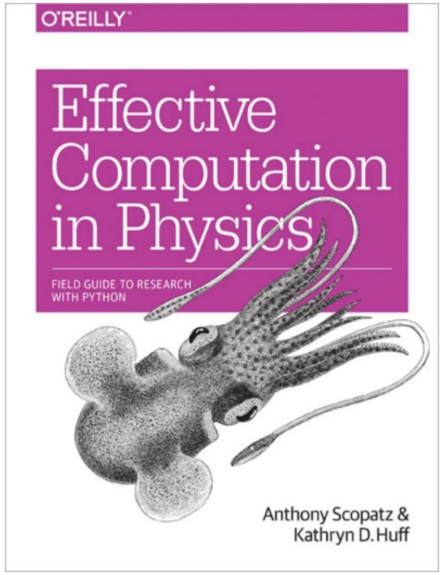
***An Introduction to Computational Physics*, by Tao Pang**

- Also good coverage of the topics (up to PDEs)
- Lots of physics examples
- Inexpensive



***Numerical Methods for Physics* by Alejandro Garcia**

- Broad coverage
- More PDE stuff than Pang



Effective Computation in Physics by Scopatz & Huff

- Introduces linux/unix shell
- Covers programming practices
- Introduces parallel programming

Why computation?

"Computational science now constitutes what many call the third pillar of the scientific enterprise, a peer alongside theory and physical experimentation."

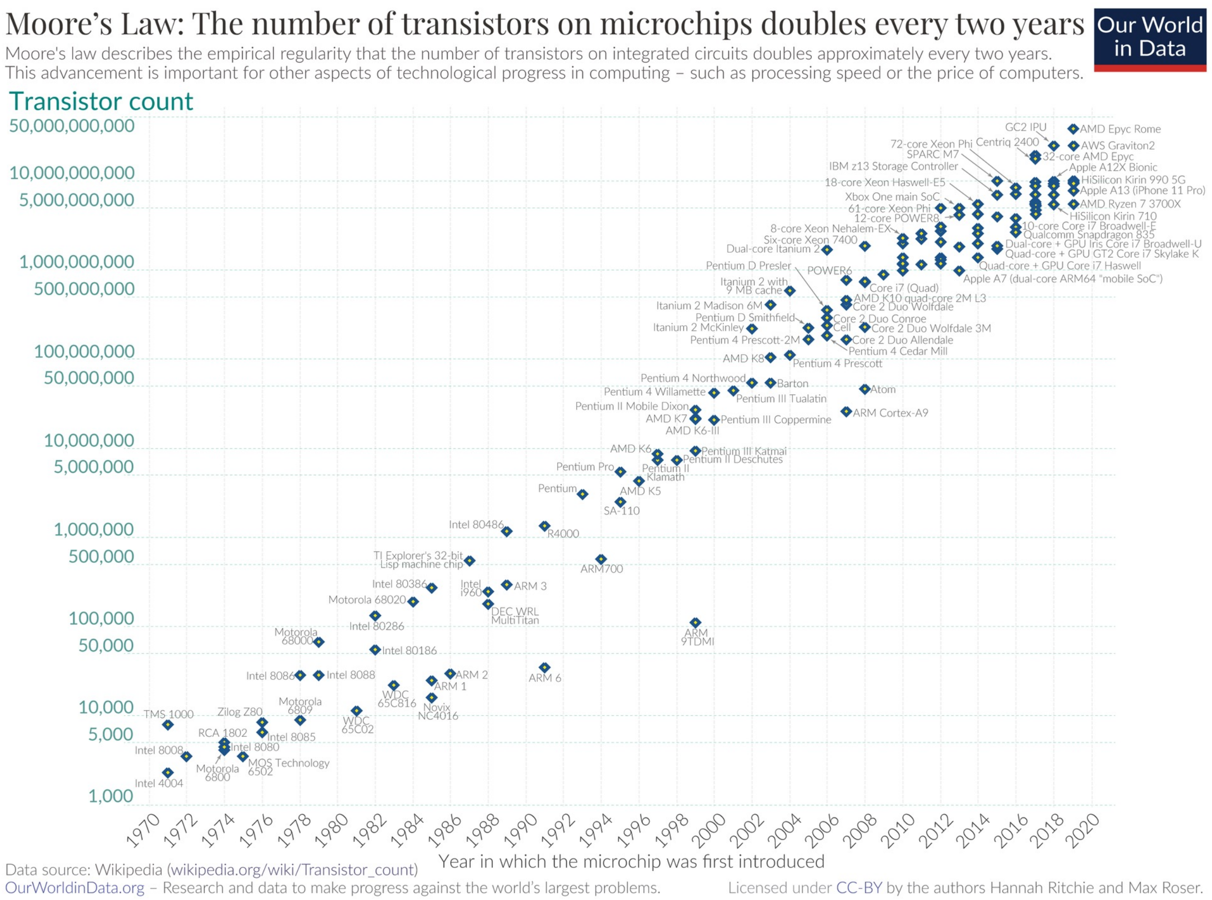
—President's information technology advisory committee (2005)

- Computation allows us to go beyond analytically solvable problems
- Computers allow us to perform repetitive tasks efficiently
- Computers allow us to generate and analyze large amounts of data

The two roles of computational in physics research

- Calculation: Using computers to solve well-defined problems
- Simulation: Use the computer to perform computational experiments

Computational science is driven by **more powerful computers**



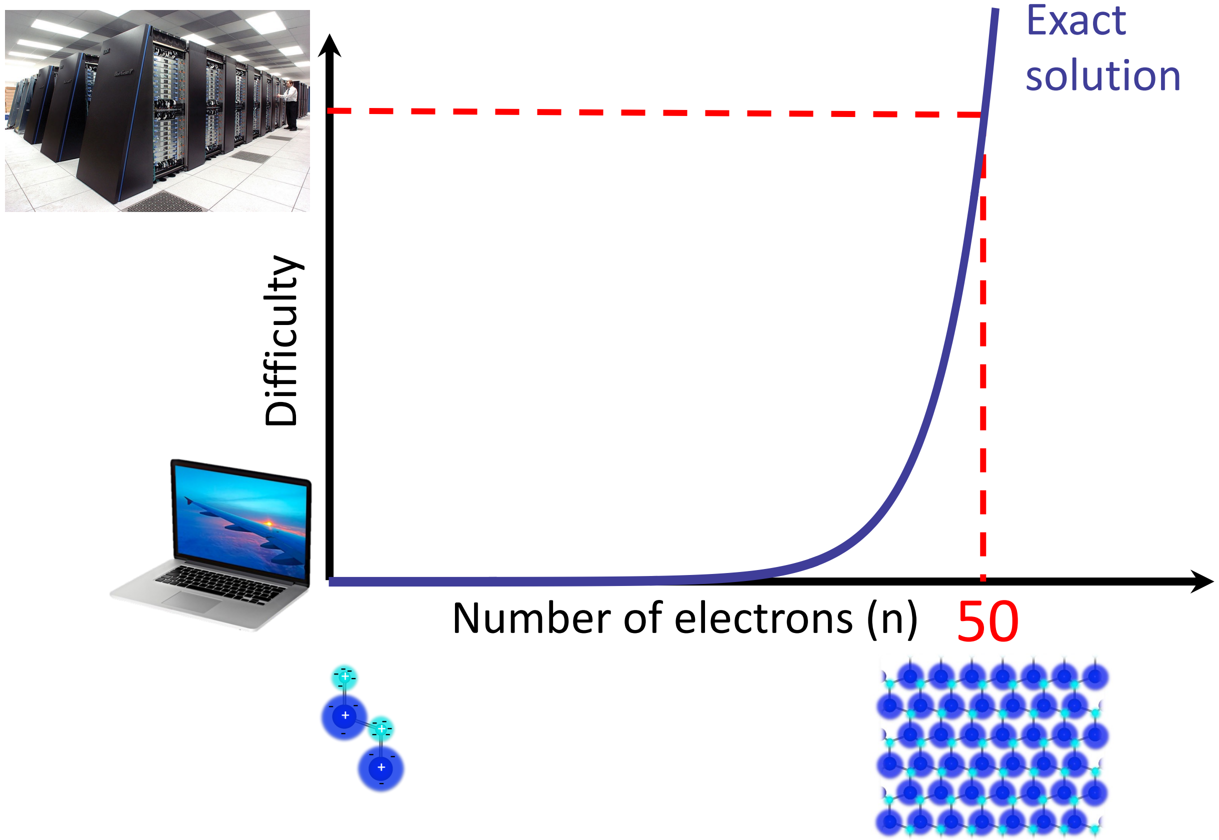
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096

Computational science is driven by **better methods/algorithms**

$$\hat{H} = -\frac{\hbar}{2m_e}\sum_i \nabla_i^2 - \sum_{i,I}\frac{Z_I e^2}{|\mathbf{r}_i - \mathbf{R}_I|} + \frac{1}{2}\sum_{i\neq j}\frac{e^2}{|\mathbf{r}_i - \mathbf{r}_j|} - \sum_I \frac{\hbar^2}{2M_I}\nabla_I^2 + \frac{1}{2}\sum_{I\neq J}\frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|}$$

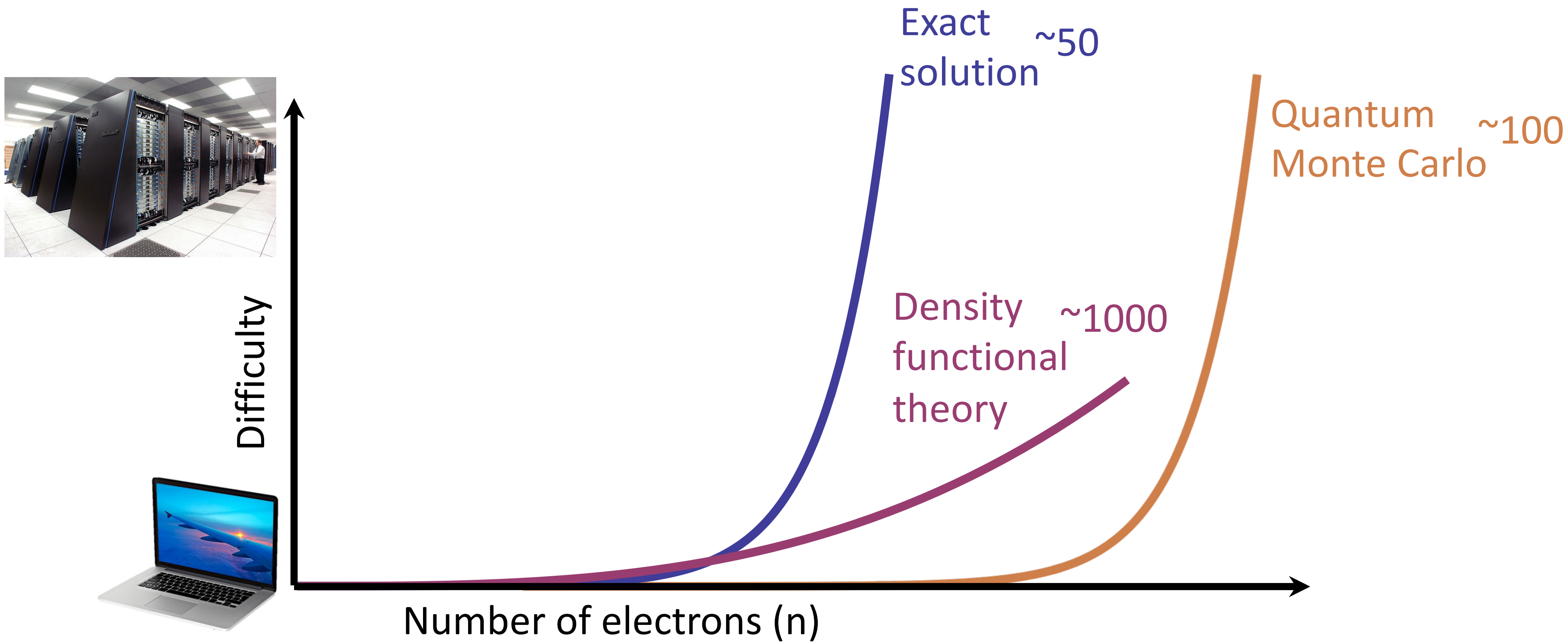
Computational science is driven by **better methods/algorithms**

$$\hat{H} = -\frac{\hbar}{2m_e}\sum_i \nabla_i^2 - \sum_{i,I}\frac{Z_I e^2}{|\mathbf{r}_i - \mathbf{R}_I|} + \frac{1}{2}\sum_{i\neq j}\frac{e^2}{|\mathbf{r}_i - \mathbf{r}_j|} - \sum_I \frac{\hbar^2}{2M_I}\nabla_I^2 + \frac{1}{2}\sum_{I\neq J}\frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|}$$



Computational science is driven by **better methods/algorithms**

$$\hat{H} = -\frac{\hbar}{2m_e}\sum_i \nabla_i^2 - \sum_{i,I}\frac{Z_I e^2}{|\mathbf{r}_i - \mathbf{R}_I|} + \frac{1}{2}\sum_{i\neq j}\frac{e^2}{|\mathbf{r}_i - \mathbf{r}_j|} - \sum_I \frac{\hbar^2}{2M_I}\nabla_I^2 + \frac{1}{2}\sum_{I\neq J}\frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|}$$



Goals for (the rest of) this lecture

- Representing numbers on the computer
  - Types
  - Finite precision of floating points
  - Comparing real numbers

Information in computer programs categorized by “**Type**”

C++ Type	**Fortran Equivalent**	**Description**	**Example**
`short` (also called `short int`)	`integer(4)`	Positive or negative number with no decimal places.	`56478, 3, -278`
`int`	`integer`		
`long` (also called `long int`)	`integer(8)`		
`float`	`real`	Positive or negative number with decimal places.	`3.0, 1.67e10, -3.2234e-20`
`double`	`real(8)`		
`long double`	`real(16)`		
`char`	`character(1)`	Single or multiple letters, numbers, symbols with no special interpretation	`a, abj3a, gh_&w`
`string` (string type implemented as a container in C++ standard library)	`character(len=*)` (as of Fortran 2008 standard)		
`bool`	`logical`	True or False	`.True., False`
`complex` (complex type implemented as a Template class in C++ standard library)	`complex`	Complex numbers	`3.0+5.6i`

All information in a computer stored as **bits**

- Basic unit of information in a computer is a bit: **0 or 1**
  - 8 bits = 1 byte

- All types must be converted into some number of bytes

- Finite storage limits, e.g., the size or precision of a number

Binary data representation

- "Human" representation: Base ten (decimal)
  - Each digit multiplies a power of 10

10^0 = 1's place

10^1 = 10's place

10^2 = 100's place

$$137_{10} = 1x10^2 + 3x10^1 + 7x10^0$$

- "Computer" representation: Base two (binary)
  - Each digit multiplies a power of 2:

2^0 = 1's place

2^1 = 2's place

2^2 = 4's place

$$101_2 = 1x2^2 + 0x2^1 + 1x2^0$$

$= 5_{10}$ (in base 10 representation)

The amount of memory allocated to an integer determines largest number that can be stored

- E.g., 1 byte: | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $= 1x2^6 + 1x2^5 + 1x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 1x2^0 = 127_{10}$

  Sign bit; 0 if positive integer
  1 if negative integer

- 2-byte:
  - This can store 2^{15}-1 distinct values: -32,768 to 32,767 (signed)
  - Or it can store 2^{16} values: 0 to 65,535 (unsigned)

- Standard in many languages is 4-bytes
  - This can store 2^{31}-1 distinct values: -2,147,483,648 to 2,147,483,647 (signed)
    - C/C++: int (usually) or int32_t
    - Fortran: integer or integer(4)
  - Or it can store 2^{32} distinct values : 0 to 4,294,967,295 (unsigned)
    - C/C++: uint or uint32_t
    - Fortran (as of 95): unsigned

- For very big integers, 8-byte allows for 2^{64}
  - Fotran: integer(8)
  - C++: long

Overflow: Trying to put more information in a type than will fit

- What happens when you try to store an integer that too large for the memory allocated?
  - **Depends on the language!**

- Fortran: Just gives you the wrong result

- Python: Allows the size of the integer to scale with the size of the number

Another aspect of integers to keep in mind: **Integer division**

- Multiplication of integers results in an integer; addition/subtraction of integers result in an integer; division of integers does not always result in an integer!

- What happens if we divide two integers like: `1/2`?
    - In some codes, 1/2 gives 0, in others it converts to real and give 0.5
    - Common source of bugs!!

Real/Floating point numbers are more complicated

- Infinite real numbers on the number line need to be represented by a finite number of bits

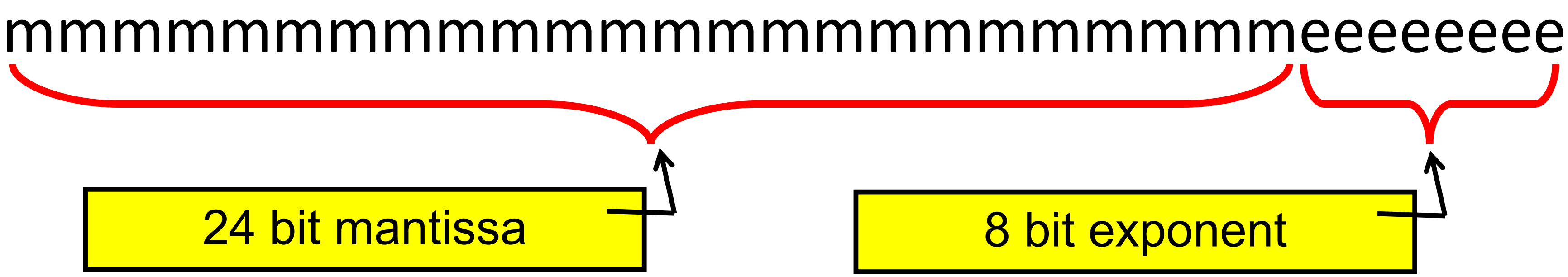
- Finite memory results in limited **size and precision** of floating point numbers
  - Not all real numbers (even simple ones) can be stored in a finite number of digits in a base-2 representation
  - Example: $1/10=0.1_{10} = 0.0001100110011..._2$ does not have a finite representation in base 2 just as $1/3=0.333333..._{10}$ has no finite representation in base 10

- This means that even simple floating point numbers are often approximated with some small error
  - This means that floating point arithmetic is not exact! (on all computers and programming languages)

- Errors can compound if not treated carefully!

Real (a.k.a. floating point) data

- IEEE 754 mantissa-exponent form:



- Value = mantissa x 2 $^{\text{exponent}}$
- Single precision:
  - Sign: 1 bit; exponent: 8 bits; significand: 24 bits (23 stored) = 32 bits
  - Range: 2^7-1 in exponent (because of sign) = 2^{127} multiplier ~ 10^{38}
  - Decimal precision: ~6 significant digits
- Double precision:
  - Sign: 1 bit; exponent: 11 bits; significand: 53 bits (52 stored) = 64 bits
  - Range: 2^{10}-1 in exponent = 2^{1023} multiplier ~ 10^{308}
  - Decimal precision: ~15 significant digits

Finite precision of floating points

- This means that most real numbers do not have an exact representation on a computer.
  - Spacing between numbers varies with the size of numbers
  - Relative spacing is constant

$$\text{relative roundoff error} = \frac{|\text{true number} - \text{computer number}|}{|\text{true number}|} \leq \epsilon$$

Overflows/underflows with reals

- Overflows and underflows can still occur when you go outside the representable range.
  - The floating-point standard will signal these (and compilers can catch them)
- Some special numbers:
  - `NaN` = 0/0 or $\sqrt{-1}$
  - `Inf` is for overflows, like 1/0
  - Both of these allow the program to continue, and both can be trapped (and dealt with)
- `-0` is a valid number, and `-0 = 0` in comparison
- Floating point is governed by an IEEE standard
  - Ensures all machines do the same thing
  - Aggressive compiler optimizations can break the standard

A result of finite precision: Need to be careful when comparing floats/reals

- Floating point numbers involve rounding and imprecision, which propagate in different ways under different operations
- Mathematically analogous expressions may yield slightly (or significantly as we will see!) different results
- In principle, this can be accounted for since floating point operations follow specific rules
  - see reading "What Every Computer Scientist Should Know About Floating-Point Arithmetic," by David Goldberg
- In practice, it best to do an "epsilon check"

Epsilon check for comparing floats

- Take two real numbers `a` and `b`
- We take `a==b` if `abs(a-b) < epsilon`

- Have to be very careful with this!!! We should think about:
  - The choice of `epsilon` based on the precision we require/expect for `a` and `b`
  - The choice of `epsilon` based on the magnitude of `a` and `b`
  - What will happen in special cases (`0, NaN, inf`)
  - …

After class tasks

- Readings:
  - What every computer scientist should know about floating-point arithmetic
  - Wikipedia page on the Floating Point
  - Wikipedia page on the Kahan Summation Algorithm