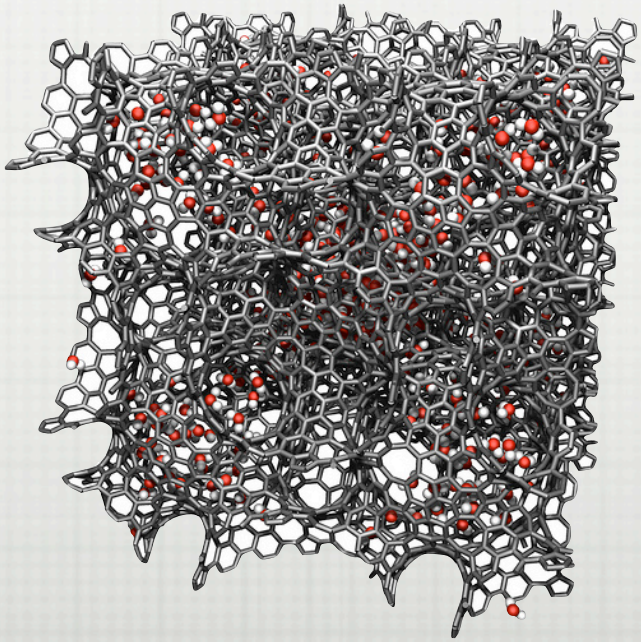
PHYS-4810
COMPUTATIONAL PHYSICS

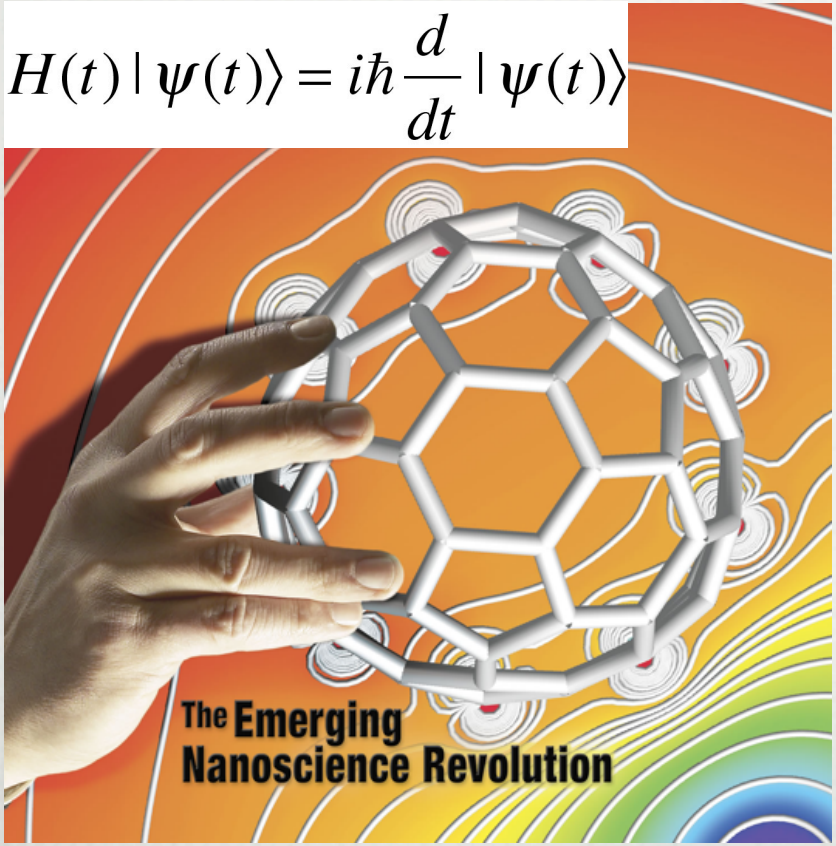
LECTURE 0: INTRODUCTION & OVERVIEW

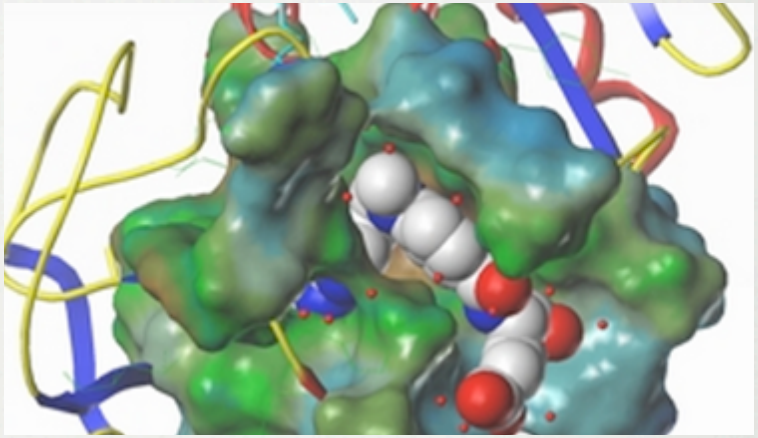


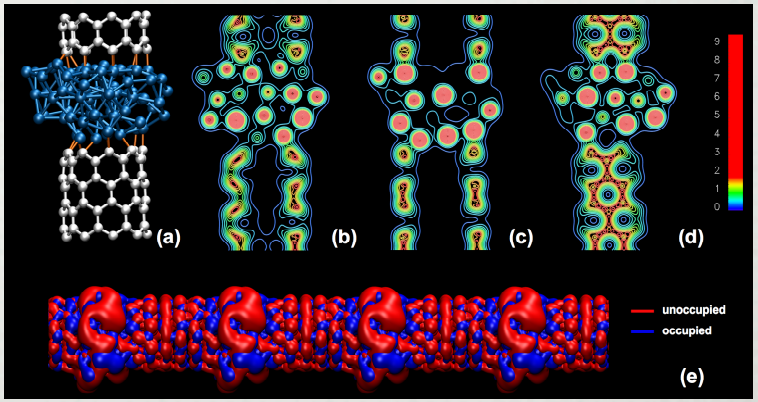
ABOUT THE INSTRUCTOR

- **COMPUTATIONAL PHYSICIST BY TRAINING**
- **JOINED RPI IN 2010**
- **HAS CREATED THIS COURSE IN SPRING 2011**
- **THIRD TIME THIS COURSE IS TAUGHT**
- **IF INTERESTED GO TO THE WEBSITE TO GET AN OVERVIEW OF ACTIVITIES**

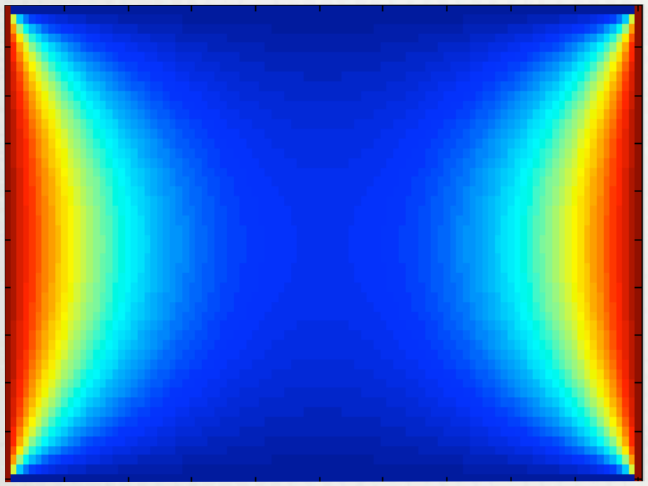
QUANTUM MECHANICS

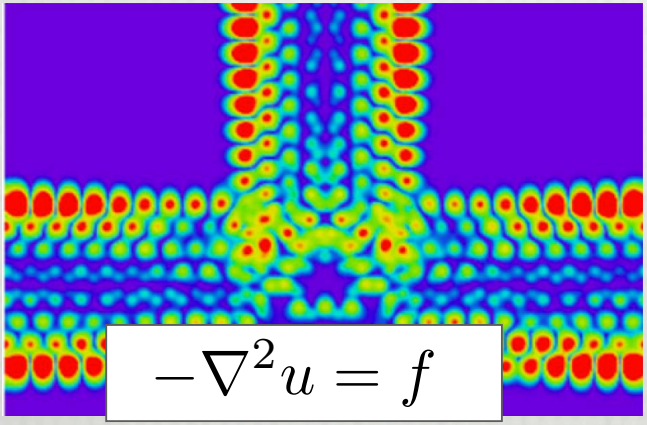




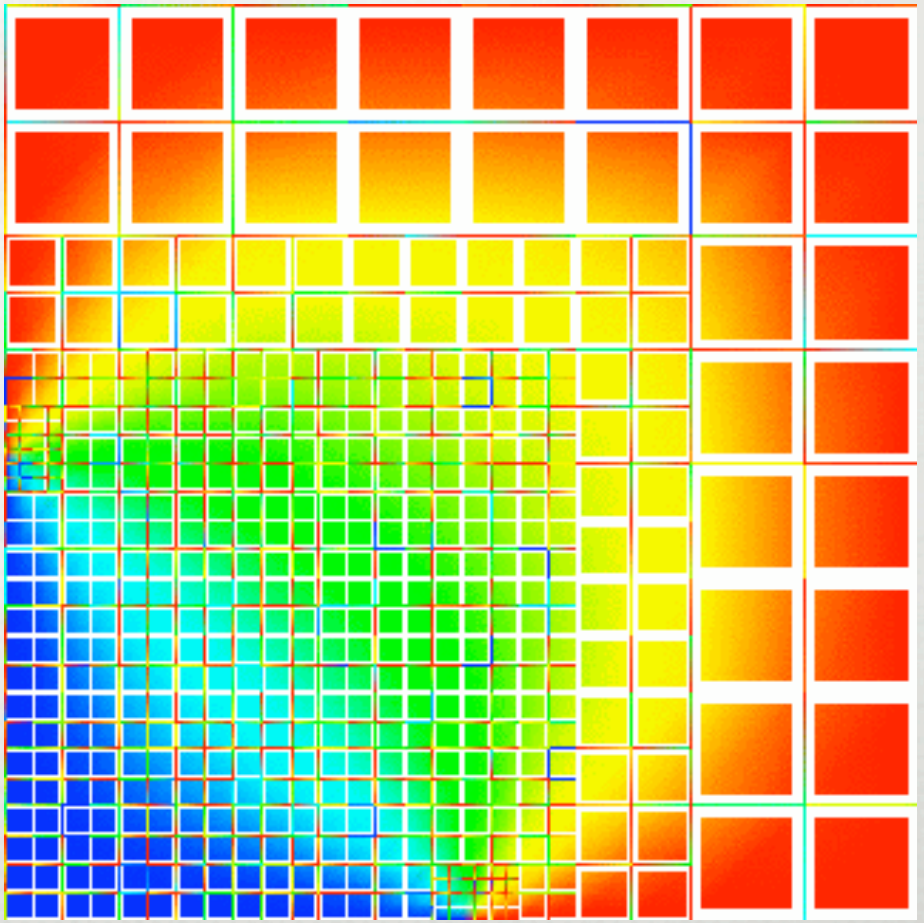


ELECTROSTATICS

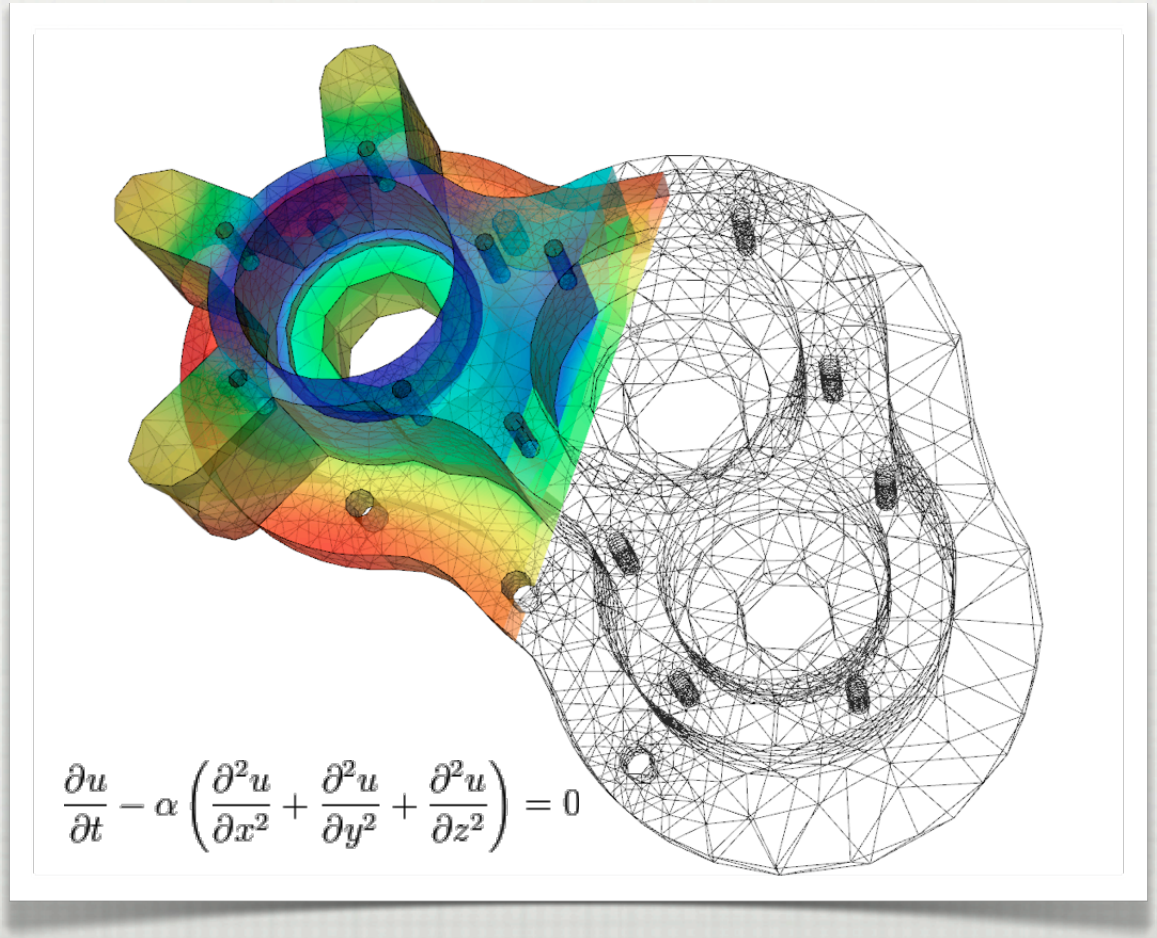




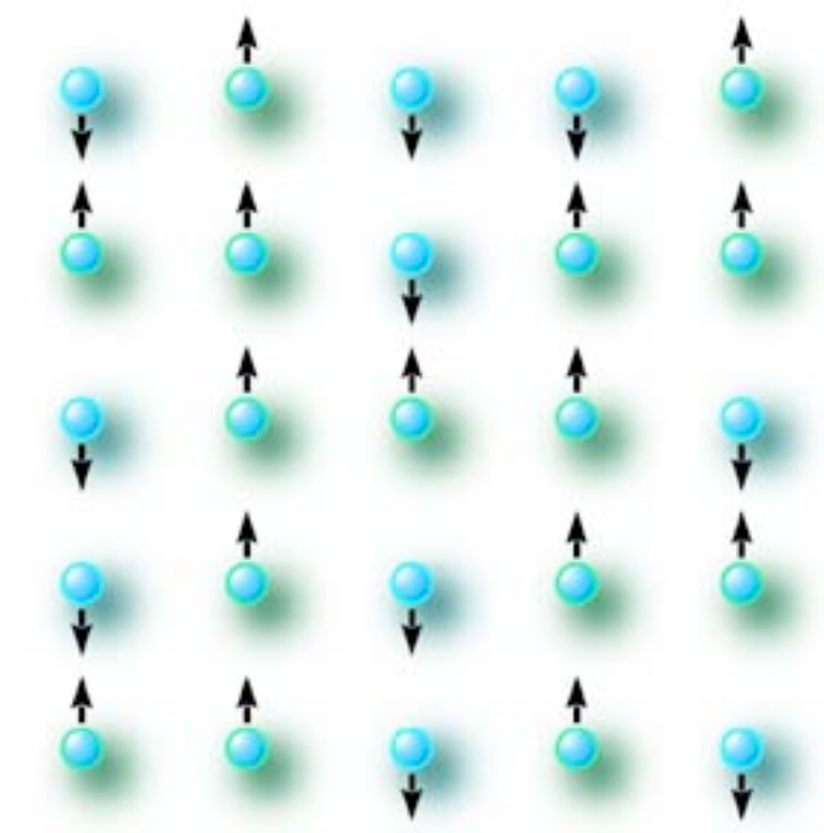
$$-\nabla^2 u = f$$

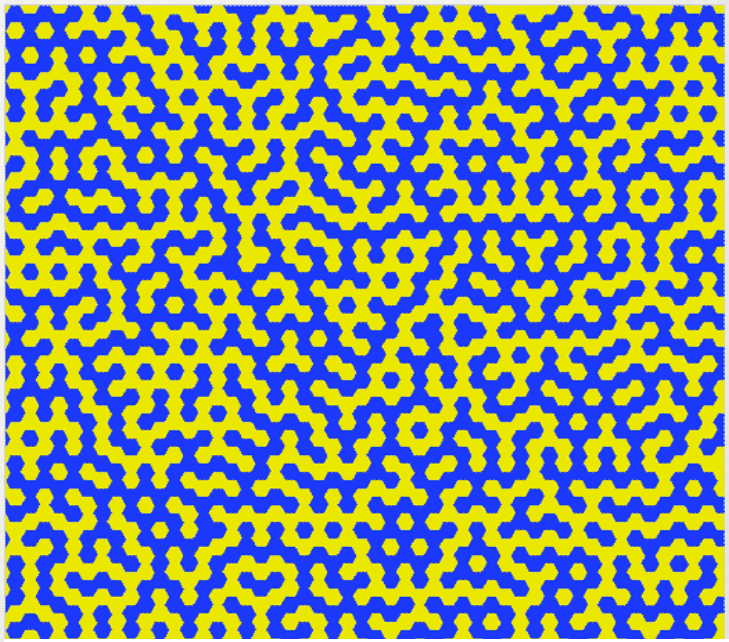


HEAT TRANSFER IN A PUMP



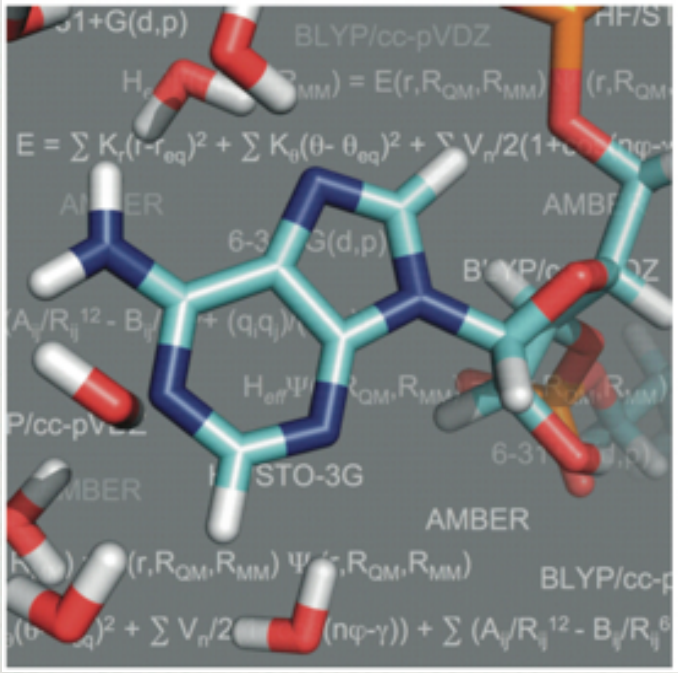
ISING MODEL: ANTIFERROMAGNETISM





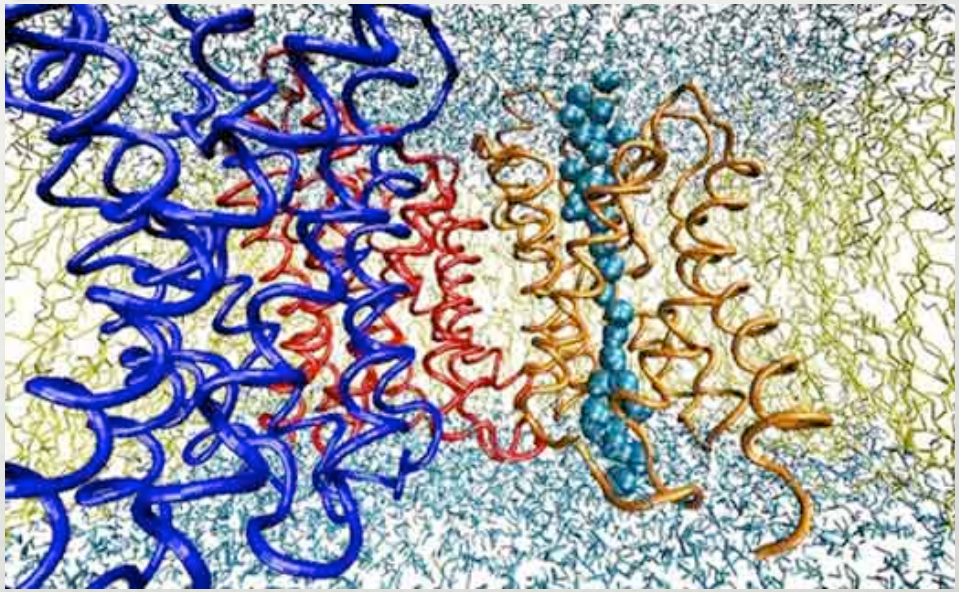
$$H = -t \sum_{\langle i,j\rangle,\sigma} (c^{+}_{i\sigma}\, c_{j\sigma} + c^{+}_{j\sigma}\, c_{i\sigma}) + U \sum_{i} (n_{i\uparrow} - \tfrac{1}{2}) \times (n_{i\downarrow} - \tfrac{1}{2}) - \mu \sum_{i} (n_{i\uparrow} + n_{i\downarrow}),$$

MOLECULAR DYNAMICS

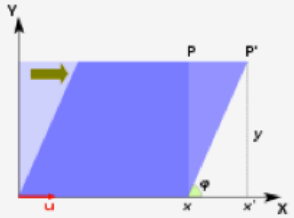
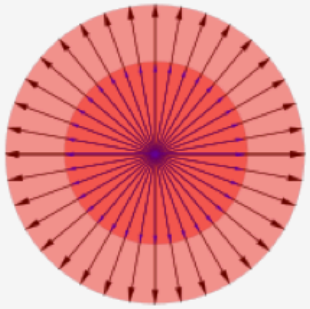
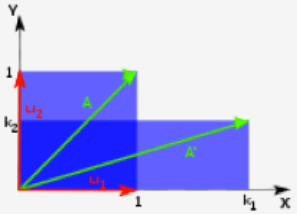
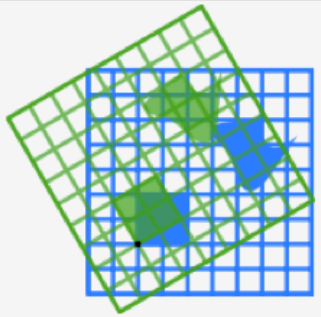


$$U = \sum_{bond} \frac{1}{2} K_b (b - b_0)^2 + \sum_{angle} \frac{1}{2} K_\theta (\theta - \theta_0)^2 + \sum_{dihedral} K_\phi [1 + \cos(n\phi - \delta)] + \sum_{nonbonded} 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r} \right)^{12} - \left(\frac{\sigma_{ij}}{r} \right)^{6} \right] + \frac{q_i q_j}{r}$$

$$m_i \frac{d^2 \vec{r}_i}{dt^2} = -\nabla_i [U(\vec{r}_1, \vec{r}_2, \ldots \vec{r}_N)]$$



LINEAR ALGEBRA AND MATRIX MANIPULATION

	horizontal shear	scaling	unequal scaling	counterclockwise rotation by φ
illustration				
matrix	$\begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix}$	$\begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix}$	$\begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}$
characteristic equation	$\lambda^2 - 2\lambda + 1 = (1-\lambda)^2 = 0$	$\lambda^2 - 2\lambda k + k^2 = (\lambda - k)^2 = 0$	$(\lambda - k_1)(\lambda - k_2) = 0$	$\lambda^2 - 2\lambda \cos\phi + 1 = 0$
eigenvalues λ_i	$\lambda_1 = 1$	$\lambda_1 = k$	$\lambda_1 = k_1, \lambda_2 = k_2$	$\lambda_{1,2} = \cos\phi \pm i \sin\phi = e^{\pm i\phi}$
algebraic and geometric multiplicities	$n_1 = 2, m_1 = 1$	$n_1 = 2, m_1 = 2$	$n_1 = m_1 = 1, n_2 = m_2 = 1$	$n_1 = m_1 = 1, n_2 = m_2 = 1$
eigenvectors	$\mathbf{u}_1 = (1,0)$	$\mathbf{u}_1 = (1,0), \mathbf{u}_2 = (0,1)$	$\mathbf{u}_1 = (1,0), \mathbf{u}_2 = (0,1)$	$\mathbf{u}_1 = \begin{bmatrix} 1 \\ -i \end{bmatrix}, \mathbf{u}_2 = \begin{bmatrix} 1 \\ i \end{bmatrix}.$

SUPERCOMPUTERS

Titan, the world's most powerful supercomputer for open science with a theoretical peak performance exceeding 20 petaflops (quadrillion calculations per second). That kind of computational capability is like each of the world's 7 billion people being able to carry out 3 million calculations per second

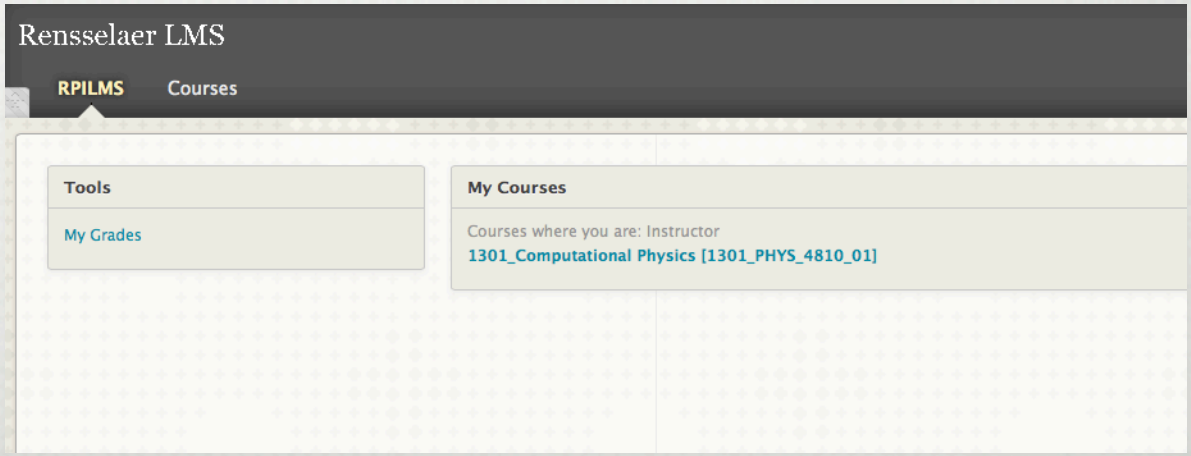
16 IBM Blue Gene/L racks, each with 1024 compute nodes and 32 I/O nodes

GOALS AND OBJECTIVES

THE PURPOSE OF THIS COURSE IS TO DEMONSTRATE HOW COMPUTATIONAL SCIENCES CAN ENABLE US TO BOTH BROADEN AND DEEPEN OUR UNDERSTANDING OF PHYSICS BY VASTLY INCREASING THE RANGE OF MATHEMATICAL CALCULATIONS WHICH WE CAN CONVENIENTLY PERFORM.

SYLLABUS

- (SEE LMS)



SCHEDULE

- ☐ LECTURE 1: COMPUTING BASICS, ERROR AND UNCERTAINTIES
- ☐ LECTURE 2: NUMERICAL INTEGRATION & DIFFERENTIATION
- ☐ LECTURE 3: TRIALS AND ERRORS
- ☐ LECTURE 4: INTRODUCTION TO RANDOM NUMBERS
- ☐ LECTURE 5: MONTE CARLO APPLICATIONS
- ☐ LECTURE 6: METROPOLIS ALGORITHM AND ISING MODEL
- ☐ LECTURE 7: DATA FITTING
- ☐ LECTURE 8: ORDINARY DIFFERENTIAL EQUATIONS
- ☐ LECTURE 9: ODE AND 1D QUANTUM MECHANICS
- ☐ LECTURE 10: PARTIAL DIFFERENTIAL EQUATIONS (PDE)
- ☐ LECTURE 11: HEAT EQUATION AND PDES
- ☐ LECTURE 12: STRING, MEMBRANES, AND PDES
- ☐ LECTURE 13: MATRIX ALGEBRA PART 1: MATRIX INVERSION
- ☐ LECTURE 13B: MORE ON SPARSE MATRICES
- ☐ LECTURE 14: EIGENVALUE PROBLEM
- ☐ LECTURE 15: INTRODUCTION TO GENETIC ALGORITHMS
- ☐ LECTURE 16: FOURIER ANALYSIS
- ☐ LECTURE 17: FREQUENCY ANALYSIS: APPLICATION TO SOUNDS
- ☐ LECTURE 18: INTRODUCTION TO GPUS

TARGET AUDIENCE

THESE LECTURES ARE DESIGNED FOR A SENIOR UNDERGRADUATE OR GRADUATE AUDIENCE

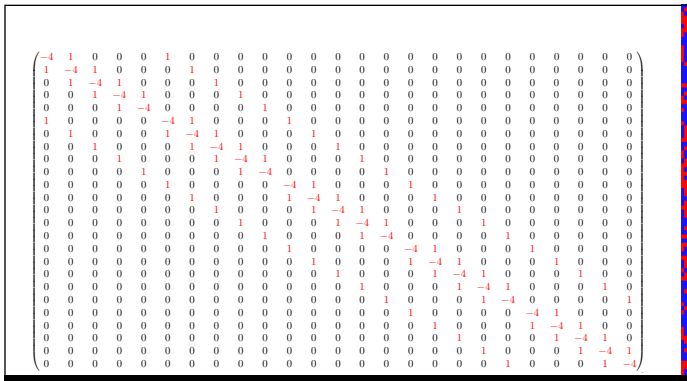
THIS COURSE IS NOT A PROGRAMMING LANGUAGE COURSE

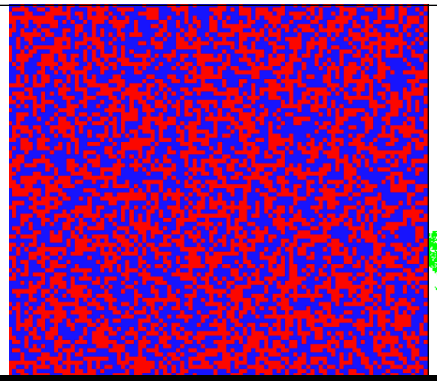
THIS COURSE IS A PHYSICS COURSE

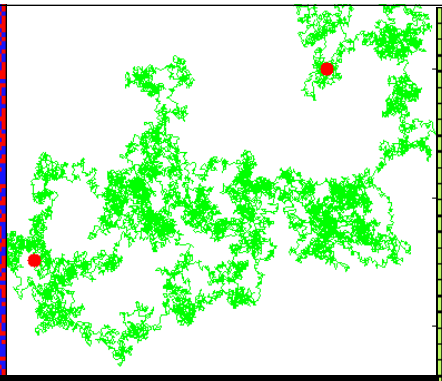
MY APPROACH

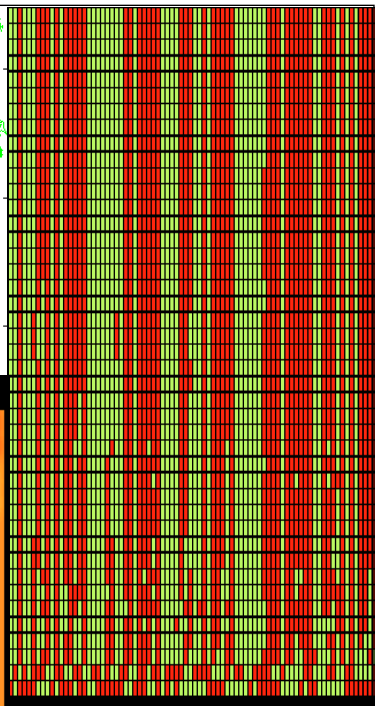
MY APPROACH TO COMPUTATIONAL PHYSICS IS TO WRITE SELF-CONTAINED PROGRAMS IN A HIGH-LEVEL SCIENTIFIC LANGUAGE SUCH AS EITHER FORTRAN OR C/C++.

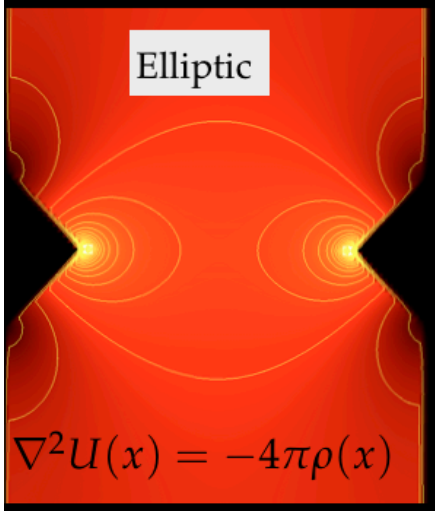
"Your son knows three computer languages, but he's flunking English."

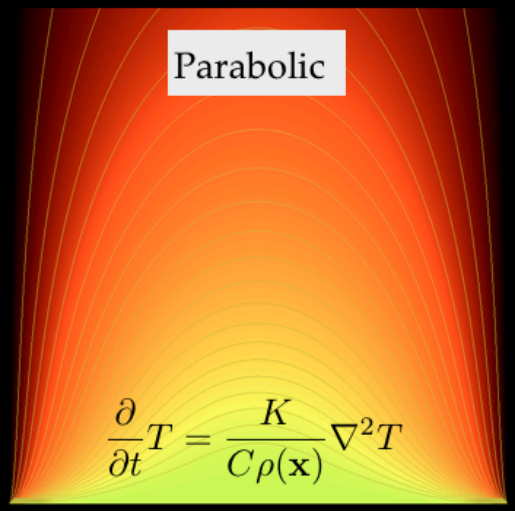


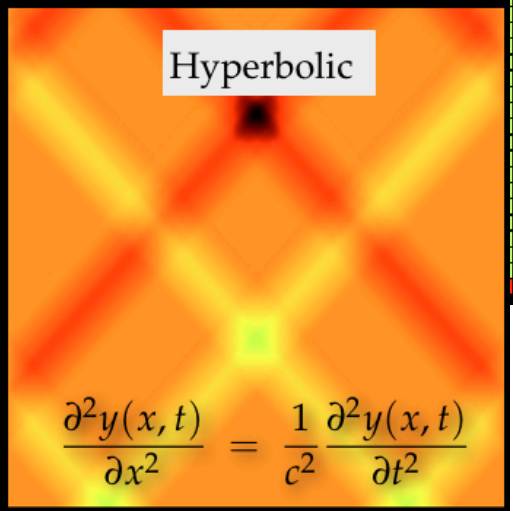


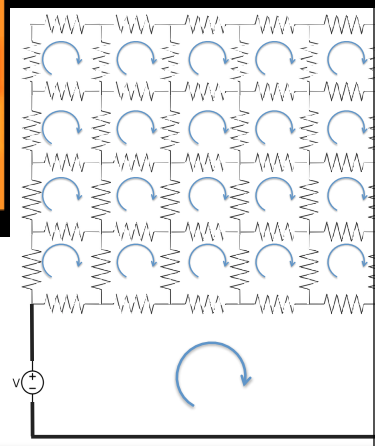


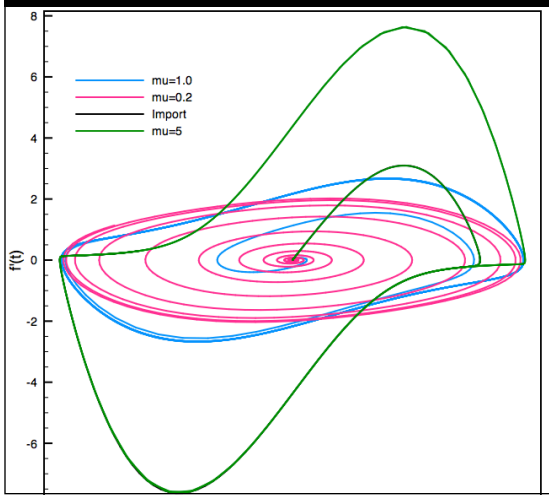


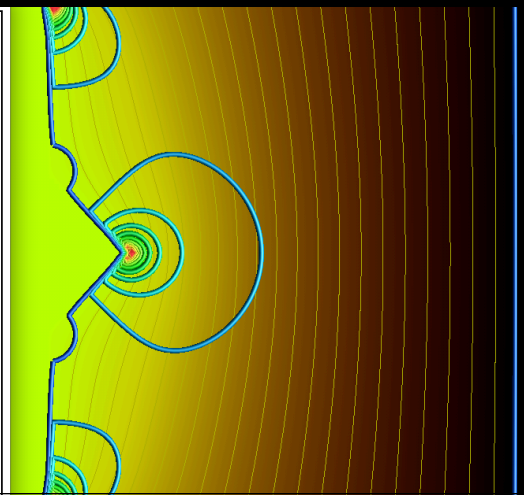

Elliptic
$\nabla^2 U(x) = -4\pi\rho(x)$

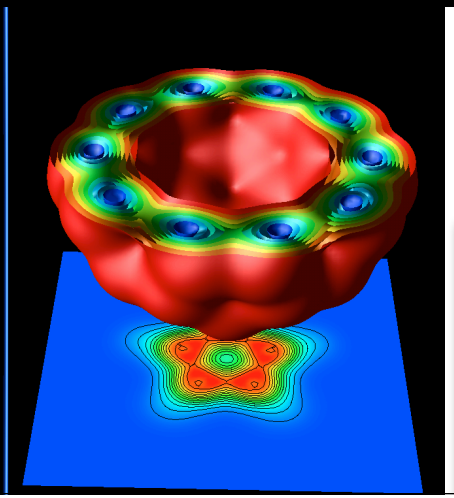

Parabolic
$\frac{\partial}{\partial t}T = \frac{K}{C\rho(\mathbf{x})}\nabla^2 T$

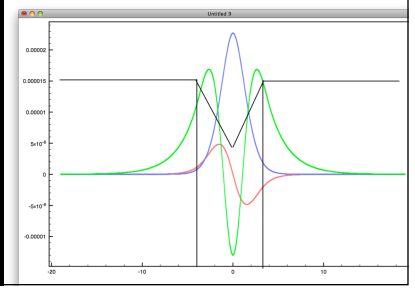

Hyperbolic
$\frac{\partial^2 y(x,t)}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 y(x,t)}{\partial t^2}$




mu=1.0
mu=0.2
Import
mu=5
f'(t)

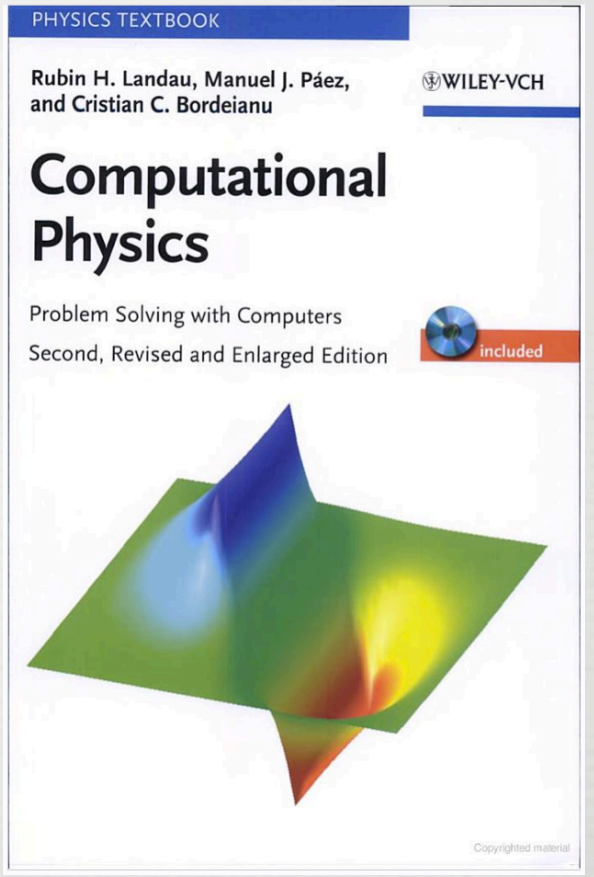






OPTIONAL TEXTBOOK

- **I WILL USE SOME ASPECTS OF THIS BOOK AND (MANY) OTHERS.**
- **SLIDES WILL BE AVAILABLE AHEAD OF LECTURE ON LMS**



HTTP://WWW.NRBOOK.COM/

HTTP://MATHWORLD.WOLFRAM.COM/

GRADING

- **50% FOR HOMEWORK**
- **50% FOR PROJECT (30 +20)**
- **NO EXAM**
- **GRADE MODIFIERS:**
  - **A=92 TO 100;**
  - **A-=89 TO 92;**
  - **B+=86 TO 89;**
  - **B=82 TO 86;**
  - **B-=79 TO 82;**
  - C+=76 TO 79;
  - C=72 TO 76;
  - C-=67 TO 72;
  - D=55 TO 67.

3-CREDIT COURSE

- **PHYS-4810 IS A 3-CREDIT COURSE. IT IS LARGELY TAUGHT AS A 4-CREDIT COURSE (SIMILAR WORK LOAD)**
- **YOU CAN GET AN EXTRA CREDIT BY SIGNING UP FOR PHYS4910 (PROF. NAPOLITANO)**
  - **(SOME EXTRA WORKLOAD ASSOCIATED, SUCH AS STANDALONE PROJECT PRESENTATION)**

HOMEWORK ASSIGNMENTS

- **Homework**
  - **5 or 6 homework write-ups**
  - **worst grade will be dropped for final grading**
  - **ALL HW is due: a missing HW cannot be used as "worst grade"**
- **Project**
  - **Project report (individual) due the last week of class**
  - **Individual project interview (15 minutes)**

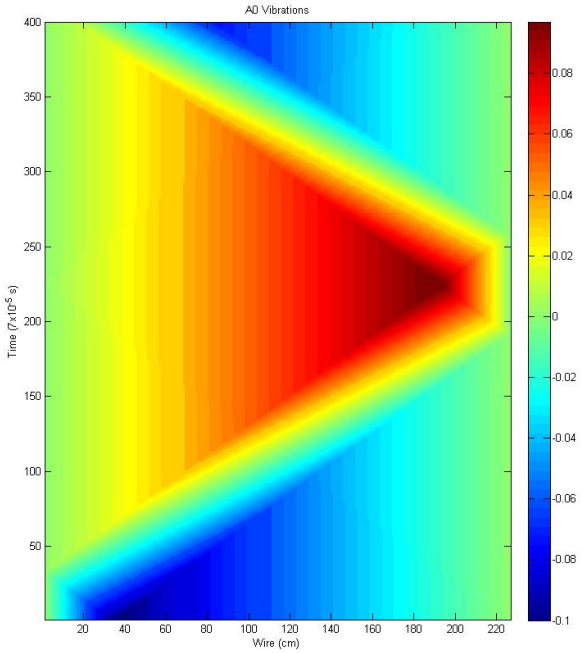
ORAL EXAMINATION

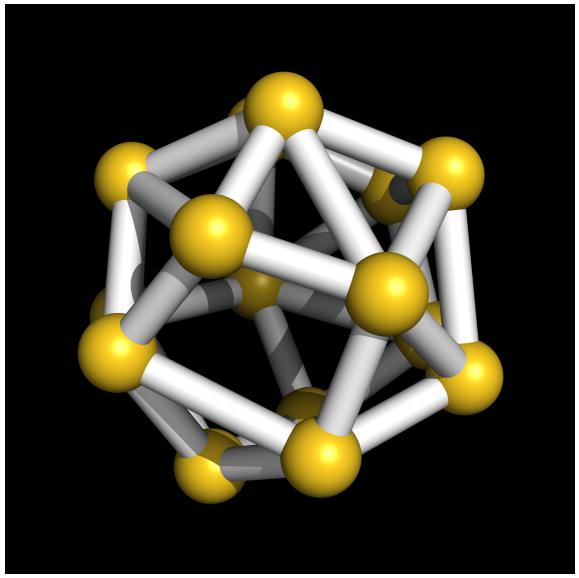
NOT THIS TYPE

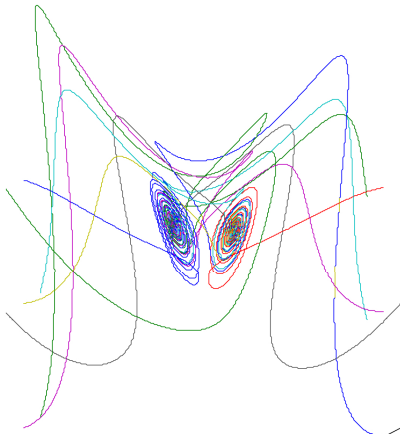
WHAT'S A PROJECT INTERVIEW?

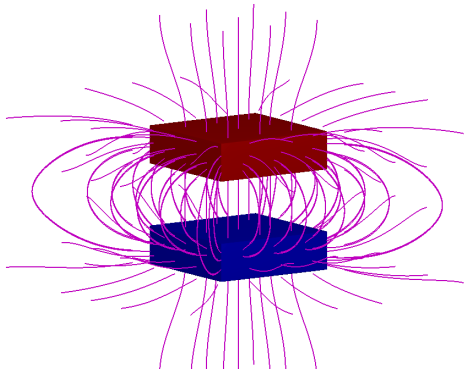
- **ONE-ON-ONE EXAM MOSTLY ON THE PROJECT**

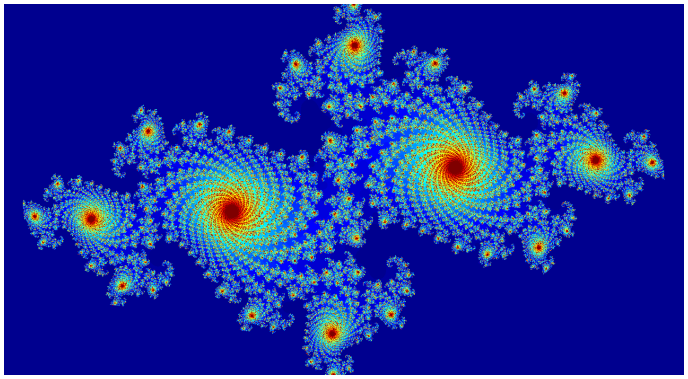
- **USE AT LEAST ONE OR TWO QUESTIONS FROM THE COURSE AS "GRADE MODIFIER"**

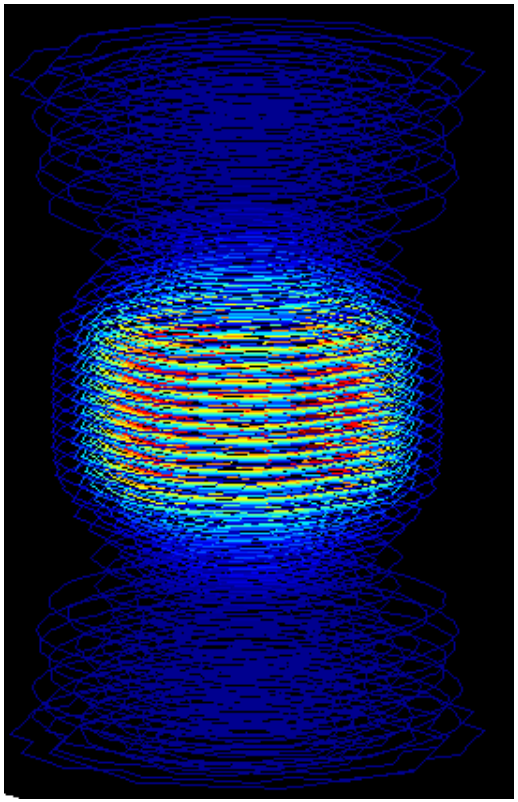












Examples of Projects (2012)

ATTENDANCE POLICY

THE WILLINGNESS TO ACCEPT RESPONSIBILITY FOR ONE'S OWN LIFE IS THE SOURCE FROM WHICH SELF-RESPECT SPRINGS.

~JOAN DIDION

REPORT AND HOMEWORK

Example:

SO_4^{2-}

$^{227}_{90}Th^{+}$

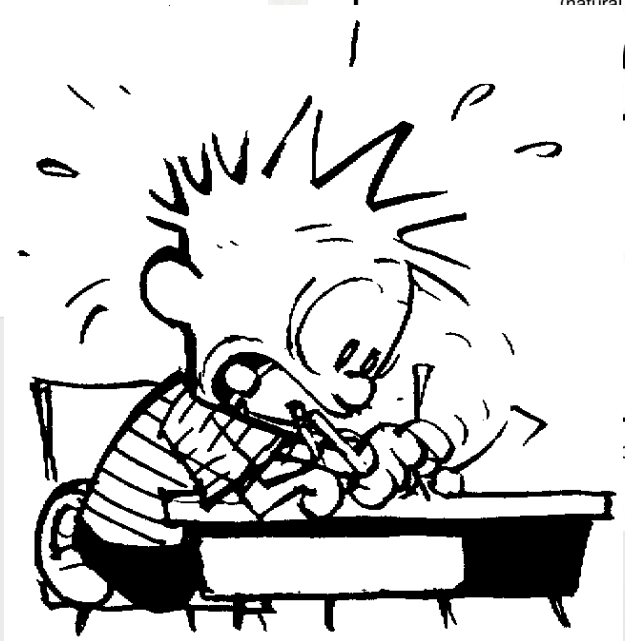
$A-B=C\equiv D$

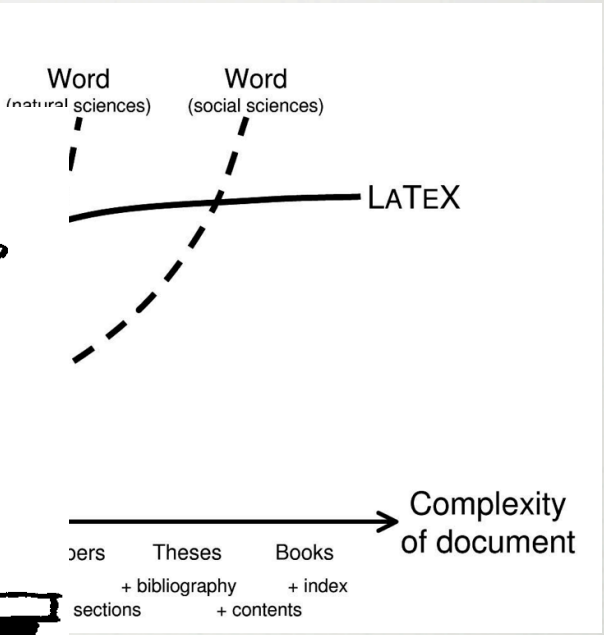
$CO_2 + C \longrightarrow 2\,CO$

$SO_4^{2-} + Ba^{2+} \longrightarrow BaSO_4\downarrow$

Source:

```
\documentclass{article}
\usepackage[version=3]{mhchem}
\parskip=0.1in
\begin{document}

\ce{SO4^2-}

\ce{^{227}_{90}Th+}

\ce{A\bond{-}B\bond{=}C\bond{#}D}

\ce{CO2 + C -> 2CO}

\ce{SO4^2- + Ba^2+ -> BaSO4 v}

\end{document}
```



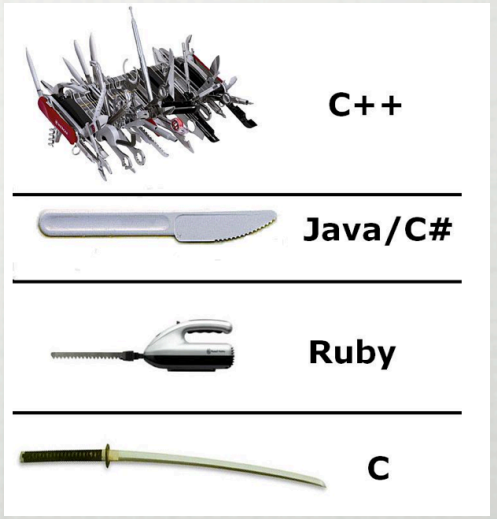


LaTeX 2ε

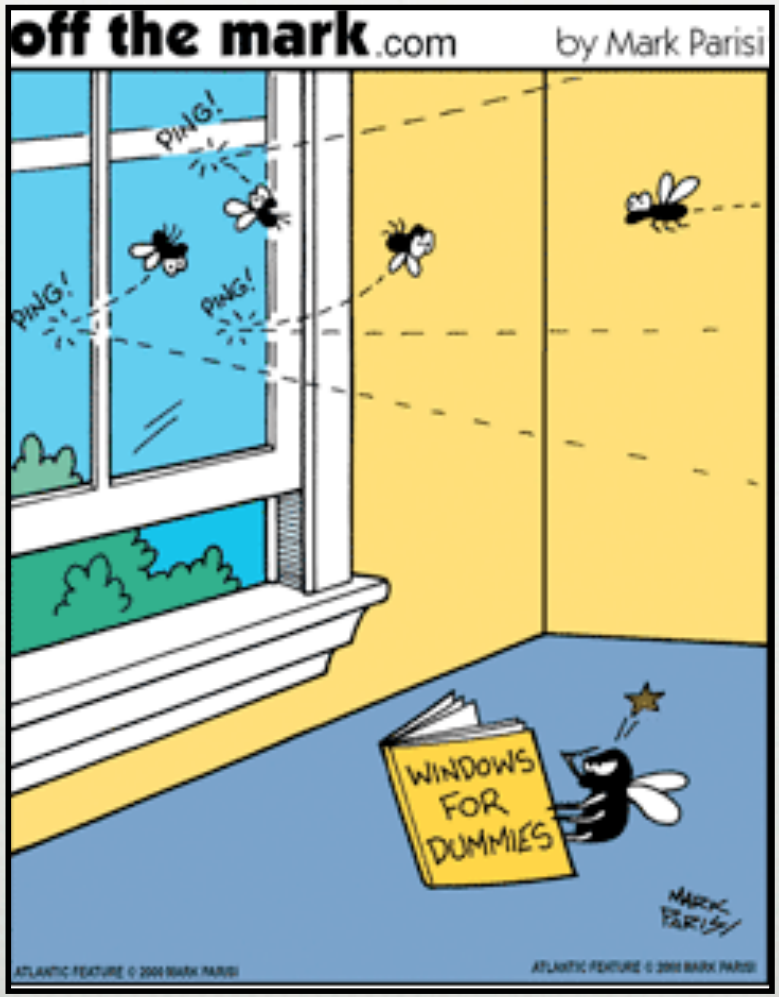
WHICH LANGUAGE?

FORTRAN 90:
This language is a major extension to FORTRAN 77 which does away with many of the latter language's objectionable features. In addition, many ``modern'' features, such as dynamic memory allocation, are included in the language for the first time.

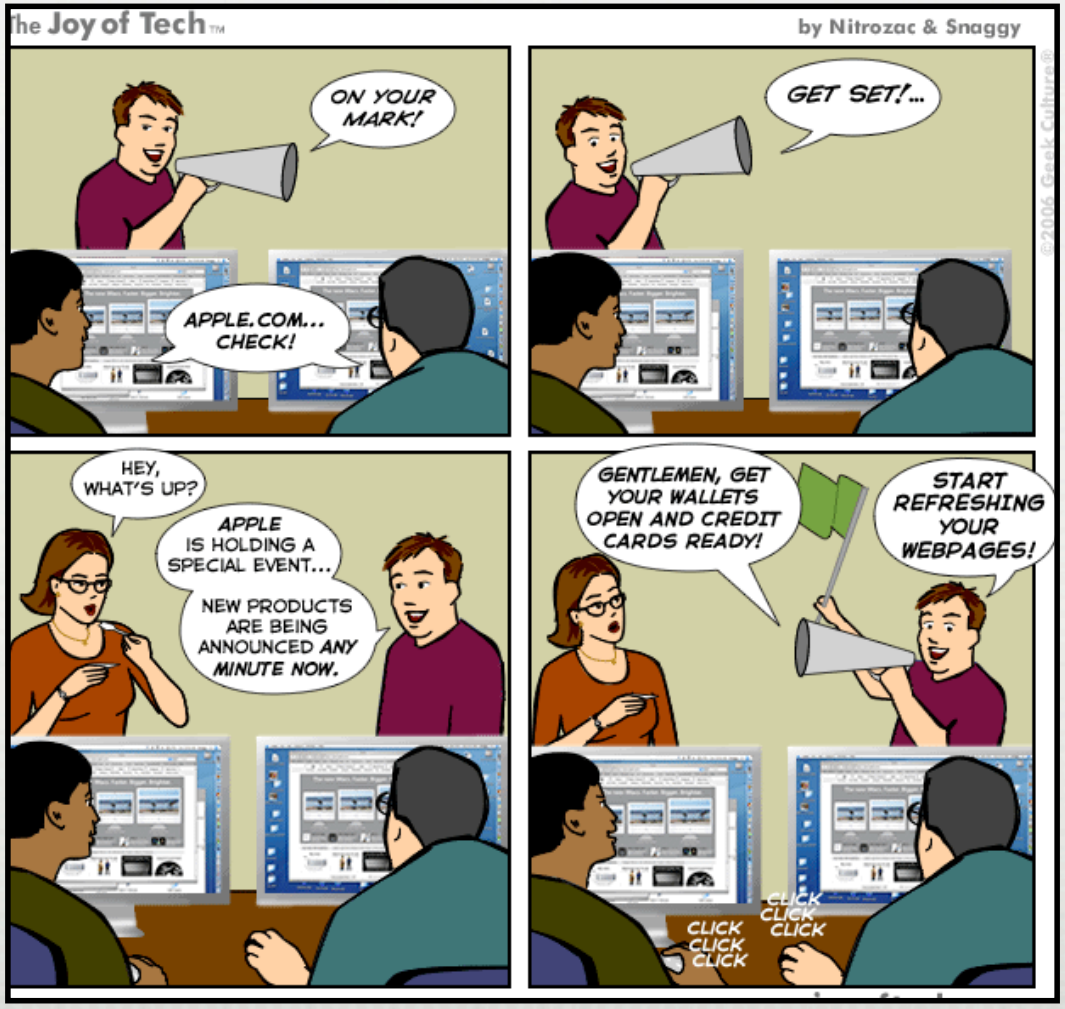
C++:
This language is a major extension of C whose main aim is to facilitate object-orientated programming. Object-orientation is a completely different approach to programming than the more traditional procedural approach. However, object-orientation represents a large, and somewhat unnecessary, overhead for the type of straightforward, single person programming tasks considered in this course. Note, however, that C++ incorporates some non-object-orientated extensions to C which are extremely useful.



NOT A FAN OF WINDOWS



MAC (MAYBE) BETTER BUT...

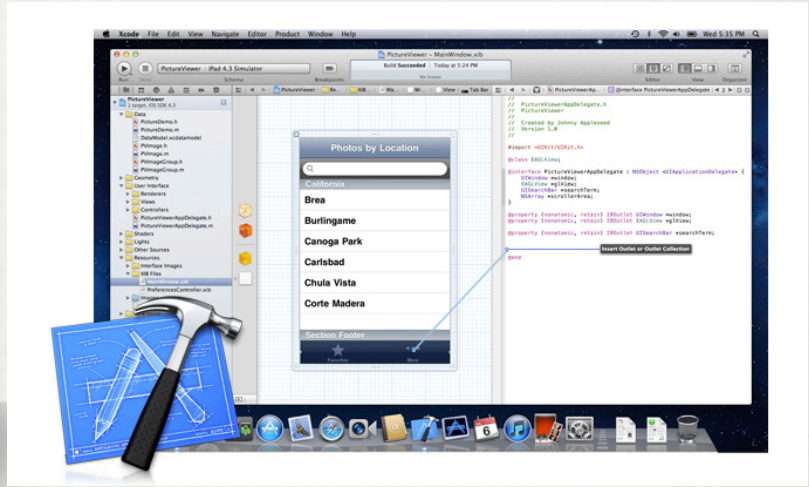


WHICH OPERATING SYSTEM TO CHOOSE?

- **SHORT ANSWER: IT DOES NOT MATTER (TOO MUCH) BUT I'D ADVISE LINUX.**

WHICH EDITOR TO CHOOSE?

- **SHORT ANSWER: IT DOES NOT MATTER (TOO MUCH)**



GETTING HELP

"They said he had to post his office hours, but they didn't say where."

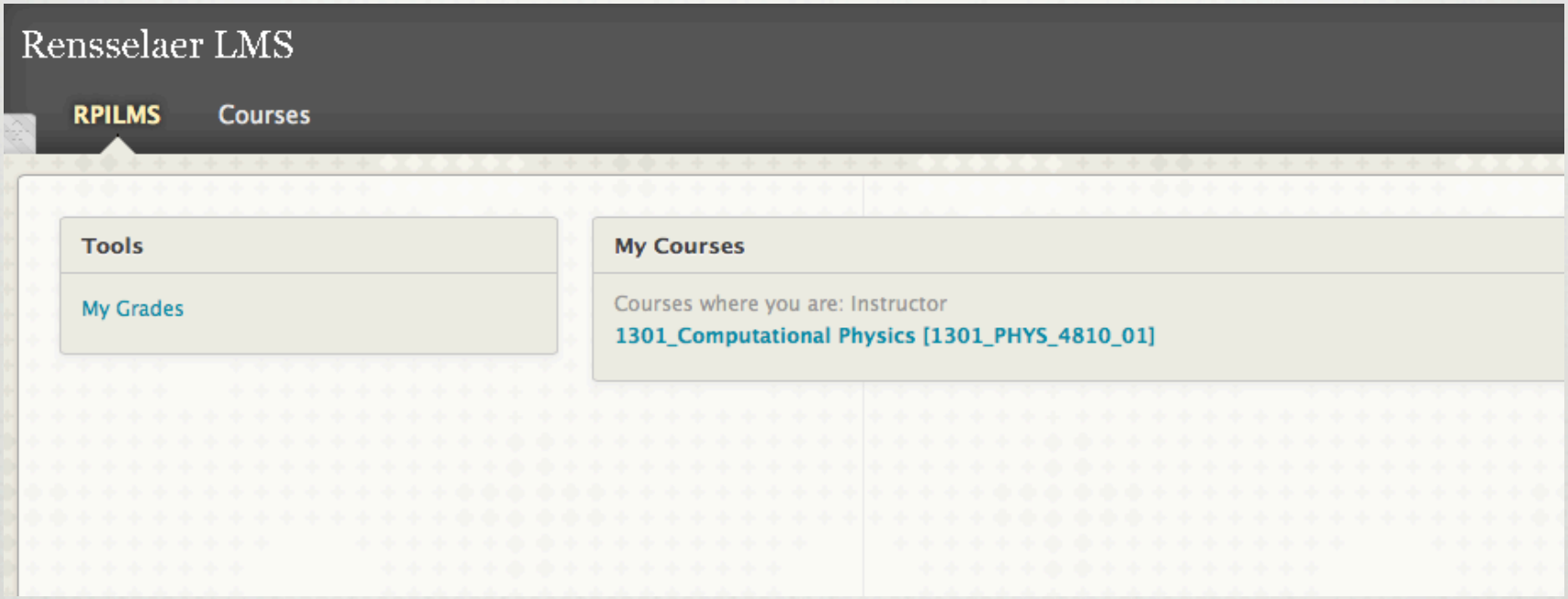
OFFICE HOURS

OFFICE LOCATION: CII 4123
E-MAIL ADDRESS:
MEUNIV@RPI.EDU
OFFICE HOURS: ANYTIME WITH APPOINTMENT

TEACHING ASSISTANT: JONATHAN OWENS
OFFICE HOURS: EVERY XXX

SEEK HELP (NO CASUALTY REPORTED -YET- IN ANY OF MY CLASSES

HOMEWORK, SCHEDULES, ETC



"You should check your e-mails more often. I fired you over three weeks ago."

Cell Phone policy

PHYS-4810
LECTURE 1

COMPUTING BASICS, ERROR AND UNCERTAINTIES

ON THE MENU TODAY

- ☐ **BASICS OF NUMBER REPRESENTATION**
- ☐ **ERRORS AND UNCERTAINTIES**

PART I:
BASICS OF NUMBER REPRESENTATION

NUMBER REPRESENTATION: EXAMPLE

- IN BASE-10 (I.E. THE FAMILIAR DECIMAL NOTATION) :
  - THE NUMBER 152853.5047,
    - HAS TEN DECIMAL DIGITS OF PRECISION,
    - IS REPRESENTED AS THE SIGNIFICAND 1528535047 (S) AND
    - AN EXPONENT OF 5 (IF THE IMPLIED POSITION OF THE RADIX POINT IS AFTER THE FIRST MOST SIGNIFICANT DIGIT, HERE 1).
- TO RECOVER THE ACTUAL VALUE, A DECIMAL POINT IS PLACED AFTER THE FIRST DIGIT OF THE SIGNIFICAND AND THE RESULT IS MULTIPLIED BY 10^5 TO GIVE 1.528535047×10^5, OR 152853.5047.

$$s \times b^e$$

OTHER EXAMPLE

- **3.14159**
  - **6 DECIMAL DIGITS**
  - **BASE: 10**
  - **SIGNIFICAND: S=314159**
  - **EXPONENT: E=0 (WE SUPPOSE THE RADIX IS AFTER THE FIRST DIGIT)**

$$s \times b^e$$

NUMBER REPRESENTATION: BASES

- HISTORICALLY, DIFFERENT BASES HAVE BEEN USED FOR REPRESENTING FLOATING-POINT NUMBERS,
  - WITH BASE 2 (BINARY) BEING THE MOST COMMON,
  - FOLLOWED BY BASE 10 (DECIMAL),
  - AND OTHER LESS COMMON VARIETIES SUCH AS BASE 16 (HEXADECIMAL NOTATION).

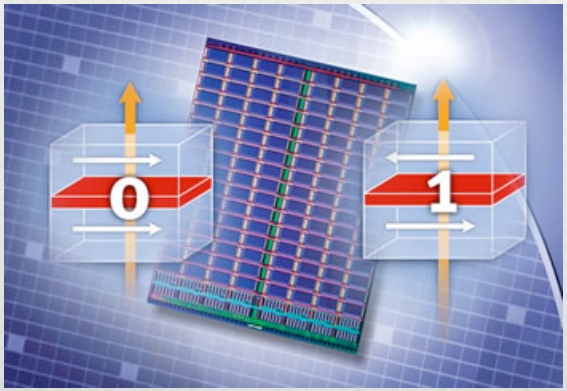
$0_{hex} = 0_{dec} = 0_{oct}$	0	0	0	0
$1_{hex} = 1_{dec} = 1_{oct}$	0	0	0	1
$2_{hex} = 2_{dec} = 2_{oct}$	0	0	1	0
$3_{hex} = 3_{dec} = 3_{oct}$	0	0	1	1
$4_{hex} = 4_{dec} = 4_{oct}$	0	1	0	0
$5_{hex} = 5_{dec} = 5_{oct}$	0	1	0	1
$6_{hex} = 6_{dec} = 6_{oct}$	0	1	1	0
$7_{hex} = 7_{dec} = 7_{oct}$	0	1	1	1
$8_{hex} = 8_{dec} = 10_{oct}$	1	0	0	0
$9_{hex} = 9_{dec} = 11_{oct}$	1	0	0	1
$A_{hex} = 10_{dec} = 12_{oct}$	1	0	1	0
$B_{hex} = 11_{dec} = 13_{oct}$	1	0	1	1
$C_{hex} = 12_{dec} = 14_{oct}$	1	1	0	0
$D_{hex} = 13_{dec} = 15_{oct}$	1	1	0	1
$E_{hex} = 14_{dec} = 16_{oct}$	1	1	1	0
$F_{hex} = 15_{dec} = 17_{oct}$	1	1	1	1

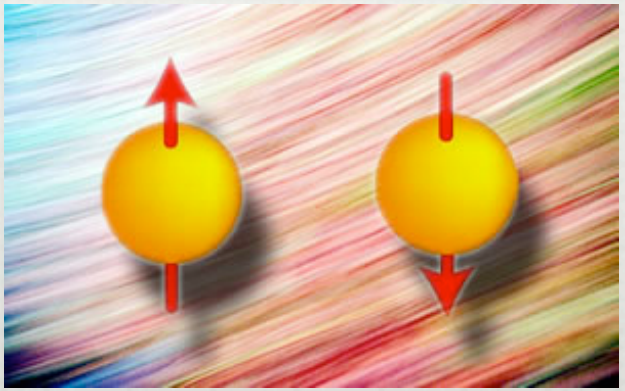
THEORY OF NUMBER REPRESENTATION

1. **COMPUTER MEMORIES ARE BASED ON THE MAGNETIC OR ELECTRONIC REALIZATION OF A SPIN POINTING UP OR DOWN**

2. **THE MOST ELEMENTARY UNITS OF COMPUTER MEMORY ARE THE TWO BITS (BINARY INTEGERS) 0 AND 1.**

3. **THIS MEANS THAT ALL NUMBERS ARE STORED IN MEMORY IN BINARY FORM, THAT IS, AS LONG STRINGS OF ZEROS AND ONES.**





THEORY OF NUMBER REPRESENTATION

AS A CONSEQUENCE, N BITS CAN STORE INTEGERS IN THE RANGE [0, 2^N-1]

BECAUSE THE SIGN OF THE INTEGER IS REPRESENTED BY THE FIRST BIT (A ZERO BIT FOR POSITIVE NUMBERS);

THE ACTUAL RANGE DECREASES TO [$-2^{N-1}+1$, $2^{N-1}-1$].

EXAMPLE:

1. IF N=2: WE HAVE

   00; 01; 10; 11 => (0_{10}, 1_{10}, 2_{10}, 3_{10})

2. WITH SIGN:

   00; 01; 10; 11 => (0_{10}, 1_{10}, -0_{10}, -1_{10})

- Long strings of zeros and ones are fine for computers, but are awkward for people.
- Binary strings are converted to octal, decimal, or hexadecimal numbers before results are communicated to people.
- Octal and hexadecimal numbers are nice because **the conversion loses no precision**, but not all that nice because **our decimal rules of arithmetic do not work** for them.
- **Converting to decimal numbers makes the numbers easier for us to work with, but unless the number is a power of 2, it leads to a decrease in precision.**

- A DESCRIPTION OF A PARTICULAR COMPUTER SYSTEM WILL NORMALLY STATE THE WORD LENGTH, THAT IS, THE NUMBER OF BITS USED TO STORE A NUMBER. THE LENGTH IS OFTEN EXPRESSED IN BYTES WITH

  - 1BYTE ≡ 1B = 8BITS (1 OCTET)

ONE BYTE:

01101101

- MEMORY AND STORAGE SIZES ARE MEASURED IN BYTES, KILOBYTES, MEGABYTES, GIGABYTES, AND TERABYTES.

- !!!! SOME CARE IS NEEDED HERE FOR THOSE WHO CHOSE TO COMPUTE SIZES IN DETAIL BECAUSE NOT EVERYONE MEANS THE SAME THING BY THESE UNITS.!!!

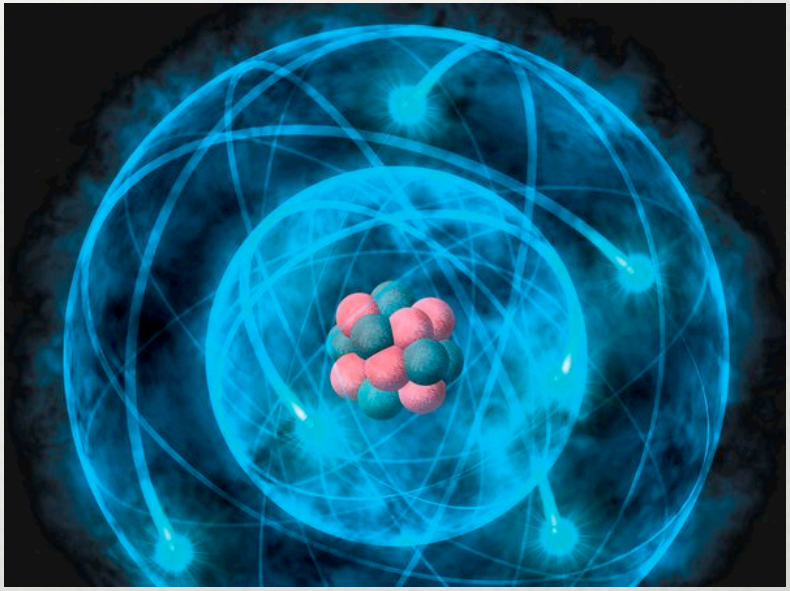
  - 1 KB = 1024 BYTES

- CONVENIENTLY, 1 BYTE IS ALSO THE AMOUNT OF MEMORY NEEDED TO STORE A SINGLE CHARACTER, SUCH AS THE LETTER “A” OR “B” THIS ADDS UP TO A TYPICAL TYPED PAGE REQUIRING ~3 KB OF STORAGE.

- THE MEMORY CHIPS IN SOME OF THE OLDER PERSONAL COMPUTERS USED 8-BIT WORDS. THIS MEANT THAT THE MAXIMUM INTEGER WAS $2^7 - 1 = 127$ (7 BECAUSE ONE BIT IS USED FOR THE SIGN).

- AT PRESENT MOST SCIENTIFIC COMPUTERS USE 64 BITS FOR AN INTEGER, WHICH MEANS THAT THE MAXIMUM INTEGER IS $2^{63} - 1 \simeq 10^{19}$. WHILE AT FIRST THIS MAY SEEM TO BE A LARGE RANGE FOR NUMBERS, IT REALLY IS NOT WHEN COMPARED TO THE RANGE OF SIZES ENCOUNTERED IN THE PHYSICAL WORLD.

- COMPARE 10^{19} TO THE RATIO OF THE SIZE OF THE UNIVERSE TO THE SIZE OF A PROTON WHICH IS APPROXIMATELY 10^{41}.



HOW TO CONVERT AN INTEGER FROM BASE 10 TO BASE 2?

- **DIVIDE THE NUMBER BY 2 AND THE REMAINDER IS THE LEAST SIGNIFICANT BIT (THE RIGHTMOST)**
- **REPEAT ON THE RESULT UNTIL THE RESULT OF DIVISION IS ZERO**
- **EXAMPLE: 10_{10}**
  - **STEP 1: 10/2: RESULT 5, REMAINDER 0**
  - **STEP 2: 5/2: RESULT 2, REMAINDER 1**
  - **STEP 3: 2/2: RESULT 1, REMAINDER 0**
  - **STEP 4: 1/2 : RESULT 0, REMAINDER 1**
  - **THEREFORE =>$10_{10}=1010_2$**

HOW TO CONVERT AN INTEGER FROM BASE 2 TO BASE 10?

- **THE BITS OF THE BINARY NUMBER ARE USED ONE BY ONE, STARTING WITH THE MOST SIGNIFICANT (LEFTMOST) BIT.**
- **BEGINNING WITH THE VALUE 0, REPEATEDLY DOUBLE THE PRIOR VALUE AND ADD THE NEXT BIT TO PRODUCE THE NEXT VALUE.**
- **EXAMPLE**: 11010_2
  - $(((0*2+1)*2+1)*2+0)*2+1)*2+0 = 26_{10}$

- CHECK USING PREVIOUS METHOD:
  - 26/2; R=13, R=0
  - 13/2; R=6, R=1
  - 6/2; R=3, R=0
  - 3/2; R=1, R=1
  - 1/2; R=0, R=1
- 11010_2

EXERCISE

- **WHAT IS 1011010_2 ?**
  - **(((((((0*2+1)*2+0)*2+1)*2+1)*2+0)*2+1)*2+0)=**
  - **90_{10}**
- **CHECK:**
  - **90/2 =45 + R0**
  - **45/2 =22 +R1**
  - **22/2 = 11 + R0**
  - **11/2 = 5 +R1**
  - **5/2 = 2 +R1**
  - **2/2 = 1 +0**
  - **1/2= 0+R1**
  - **1011010**

HOW TO CONVERT THE FRACTIONAL PART OF A NUMBER FROM BASE 10 TO BASE 2 ?

- **START WITH 0.**
- **REPEATEDLY DOUBLE THE NUMBER TO BE CONVERTED, RECORD IF THE RESULT IS AT LEAST 1, AND THEN THROW AWAY THE INTEGER PART.**
- **EXAMPLE:** $(1/3)_{10}$
  - **STEP 1: 1/3*2 < 1 -> 0**
  - **STEP 2: 2/3*2 >1 -> 1 (REMOVE 1, WORK ON 1/3)**
  - **STEP 3 : 1/3*2 < 1 -> 0**
  - **...**
  - $(1/3)_{10}=(0.0101010101010101...)_2$

0.01010

- **EXAMPLE:** $(0.1)_{10}$
- **DO IT!**

SOLUTION: 0.1 FROM BASE 10 TO BASE 2

- START WITH 0.
- 0.1 *2 < 1 => 0
- 0.2*2 < 1 => 0
- 0.4*2 < 1 => 0
- 0.8*2 > 1 => 1
- 0.6 *2 > 1 => 1
- 0.2 *2 < 1 => 0

0.0001100110011001100110011001100110011001100110011001100110011001

HOW TO MULTIPLY A BINARY NUMBER BY 2 ?

- **HOW TO MULTIPLY A DECIMAL NUMBER BY 10?**
- **SAME FOR DECIMAL: IF YOU DIVIDE (OR MULTIPLY) BY THE BASIS YOU SHIFT DECIMAL POINT LEFT (RIGHT)**
  - **10_2 DIVIDED BY 2 = 1_2**
  - **1_2 DIVIDED BY 2 = 0.1_2**

REPRESENTING REAL NUMBERS

- NON-INTEGERS CAN BE REPRESENTED BY USING NEGATIVE POWERS, WHICH ARE SET OFF FROM THE OTHER DIGITS BY MEANS OF A RADIX POINT (CALLED A DECIMAL POINT IN THE DECIMAL SYSTEM).
- FOR EXAMPLE, THE BINARY NUMBER 11.01_2 THUS MEANS:
  - 1×2^1 ($1 \times 2 = 2$) PLUS
  - 1×2^0 ($1 \times 1 = 1$) PLUS
  - 0×2^{-1} ($0 \times ½ = 0$) PLUS
  - 1×2^{-2} ($1 \times ¼ = 0.25$)
- $= 3.25_{10}$
- YOU CAN USE A TRICK TO MAKE THINGS EASIER (FIRST SHIFT THE RADIX BY TWO POSITIONS TO THE RIGHT, I.E. MULTIPLY BY 2^2, WE GET: $11.01_2=1101_2/4_{10}=(((2+1)*2+0)*2+1)/4=13/4=3.25$

NOTION OF FLOATING NUMBERS

- BY ALLOWING THE RADIX POINT (= THE "." DOT) TO BE ADJUSTABLE, FLOATING-POINT NOTATION ALLOWS CALCULATIONS OVER A WIDE RANGE OF MAGNITUDES, USING A FIXED NUMBER OF DIGITS, WHILE MAINTAINING GOOD PRECISION.
- FOR EXAMPLE, IN A DECIMAL FLOATING-POINT SYSTEM WITH THREE DIGITS, THE MULTIPLICATION THAT HUMANS WOULD WRITE AS
  - $0.12 \times 0.12 = 0.0144$ WOULD BE EXPRESSED AS
  - $(1.2\times10^{-1}) \times (1.2\times10^{-1}) = (1.44\times10^{-2})$.
- IN A FIXED-POINT SYSTEM WITH THE DECIMAL POINT AT THE LEFT, IT WOULD BE
  - $0.120 \times 0.120 = 0.014$.
- **A DIGIT OF THE RESULT WAS LOST BECAUSE OF THE INABILITY OF THE DIGITS AND DECIMAL POINT TO 'FLOAT' RELATIVE TO EACH OTHER WITHIN THE DIGIT STRING.**

FLOATING POINT

- There are actually a number of components in the IEEE standard, and different computer or chip manufacturers may adhere to only some of them. Normally a floating-point number x is stored as

$$x_{\text{float}} = (-1)^s \times 1.f \times 2^{\text{e - bias}}$$

  - with separate entities for the sign s,
  - the fractional part of the (significant) mantissa f , and
  - the exponential field e.

- All parts are stored in binary form and occupy adjacent segments of a single 32-bit word for singles, or two adjacent 32-bit words for doubles.
  - The sign s is stored as a single bit, with s = 0 or 1 for positive or negative signs.
  - Eight bits are used to store the exponent e, which means that e can be in the range $0 \leq e \leq 255$. The end points e = 0 and e = 255 are special cases.
  - Normal numbers have $0 < e < 255$, and with them, the convention is to assume that the mantissa's first bit is a 1, and so only the fractional part f after the binary point is stored.

FLOATING POINTS

THE IEEE REPRESENTATIONS ENSURE THAT ALL NORMAL FLOATING-POINT NUMBERS HAVE THE SAME RELATIVE PRECISION.

BECAUSE THE FIRST BIT IS ASSUMED TO BE 1, IT DOES NOT HAVE TO BE STORED, AND COMPUTER DESIGNERS NEED ONLY RECALL THAT THERE IS A PHANTOM BIT THERE TO OBTAIN AN EXTRA BIT OF PRECISION.

DURING THE PROCESSING OF NUMBERS IN A CALCULATION, THE FIRST BIT OF AN INTERMEDIATE RESULT MAY BECOME ZERO, BUT THIS WILL BE CORRECTED BEFORE THE FINAL NUMBER IS STORED.

IN SUMMARY, FOR NORMAL CASES, THE ACTUAL MANTISSA (F IN BINARY NOTATION) CONTAINS AN IMPLIED 1 PRECEDING THE BINARY POINT.

SINGLES AND DOUBLES

- "SINGLES" OR FLOATS IS SHORTHAND FOR SINGLE PRECISION FLOATING-POINTS
- "DOUBLES" IS SHORTHAND FOR DOUBLE PRECISION FLOATING-POINT NUMBERS.
- SINGLES OCCUPY 32 BITS OVERALL, WITH 1 BIT FOR THE SIGN, 8 BITS FOR THE EXPONENT, AND 23 BITS FOR THE FRACTIONAL MANTISSA (WHICH GIVES 24-BIT PRECISION WHEN THE PHANTOM BIT IS INCLUDED).
- DOUBLES OCCUPY 64 BITS OVERALL, WITH ONE BIT FOR THE SIGN, 10 FOR THE EXPONENT, AND 53 FOR THE FRACTIONAL MANTISSA (FOR 54-BIT PRECISION).
- THIS MEANS THAT THE EXPONENTS AND MANTISSAS FOR DOUBLES ARE NOT SIMPLY DOUBLE THOSE OF FLOATS.

SINGLE PRECISION: SUMMARY

Number name	Values of s, e, and f	Value of single
Normal	$0 < e < 255$	$(-1)^s \times 2^{e-127} \times 1.f$
Subnormal	$e = 0,\ \ f \neq 0$	$(-1)^s \times 2^{-126} \times 0.f$
Signed Zero (± 0)	$e = 0,\ \ f = 0$	$(-1)^s \times 0.0$
$+\infty$	$s = 0,\ \ e = 255,\ \ f = 0$	`+INF`
$-\infty$	$s = 1,\ \ e = 255,\ \ f = 0$	`-INF`
Not a number	$s = u,\ \ e = 255,\ \ f \neq 0$	`NaN`

$$x_{\text{float}} = (-1)^s \times 1.f \times 2^{\text{e - bias}}$$

SINGLE PRECISION

- **SINGLE-PRECISION (32 BIT OR 4-BYTE) NUMBERS HAVE 6–7 DECIMAL PLACES OF SIGNIFICANCE AND MAGNITUDES IN THE RANGE $1.4 \times 10^{-45} \leq$ SINGLE PRECISION $\leq 3.4 \times 10^{38}$.**

$$x_{\text{float}} = (-1)^s \times 1.f \times 2^{\text{e - bias}}$$

Number	s	e	f
+0	0	0000 0000	0000 0000 00000 00000 00000 000
-0	1	0000 0000	0000 0000 00000 00000 00000 000
$+\infty$	0	1111 1111	0000 0000 00000 00000 00000 000
$-\infty$	1	1111 1111	0000 0000 00000 00000 00000 000

DOUBLE PRECISION

- Doubles are stored as two 32-bit words, for a total of 64 bits (8B) overall. The sign occupies one bit, the biased exponent e 11 bits, and the fractional mantissa 52 bits:

	s	e	f	f (cont)
Bit position:	63	62 52	51 32	31 0

- The fields are stored contiguously, with part of the mantissa f stored in separate 32-bit words. The order of these words, and whether the second word with f is the most-, or least-significant part of the mantissa, is machine dependent. For doubles, the bias is quite a bit larger than for singles,
  - bias = $1111111111_2 = 1023_{10}$
  - so the actual exponent p = e − 1023.

Number name	Values of s, e, and f	Value of double
Normal	$0 \le e \le 2047$	$(-1)^s \times 2^{e-1023} \times 1.f$
Subnormal	$e = 0,\ f \ne 0$	$(-1)^s \times 2^{-1022} \times 0.f$
Signed zero	$e = 0,\ f = 0$	$(-1)^s \times 0.0$
$+\infty$	$s = 0,\ e = 2047,\ f = 0$	`+INF`
$-\infty$	$s = 1,\ e = 2047,\ f = 0$	`-INF`
Not a number	$s = u,\ e = 2047,\ f \ne 0$	`NaN`

INTERNAL REPRESENTATION IEEE 754 BINARY

Type	Sign	Exponent	Significand	Total bits		Exponent bias	Bits precision
Half (IEEE 754-2008)	1	5	10	16		15	11
Single	1	8	23	32		127	24
Double	1	11	52	64		1023	53
Quad	1	15	112	128		16383	113

TERMINATIONS

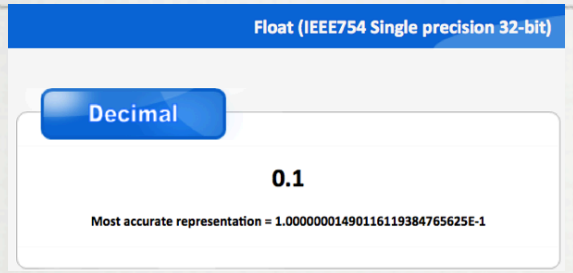
- WHETHER OR NOT A RATIONAL NUMBER HAS A TERMINATING EXPANSION DEPENDS ON THE BASE.
- FOR EXAMPLE, IN BASE-10 THE NUMBER 1/2 HAS A TERMINATING EXPANSION (0.5) WHILE THE NUMBER 1/3 DOES NOT (0.333...).
- IN BASE-2 ONLY RATIONALS WITH DENOMINATORS THAT ARE POWERS OF 2 (SUCH AS 1/2 OR 3/16) ARE TERMINATING.
- ANY RATIONAL WITH A DENOMINATOR THAT HAS A PRIME FACTOR OTHER THAN 2 WILL HAVE AN INFINITE BINARY EXPANSION.
- THIS MEANS THAT NUMBERS WHICH APPEAR TO BE SHORT AND EXACT WHEN WRITTEN IN DECIMAL FORMAT MAY NEED TO BE APPROXIMATED WHEN CONVERTED TO BINARY FLOATING-POINT.

EXAMPLE OF 0.1

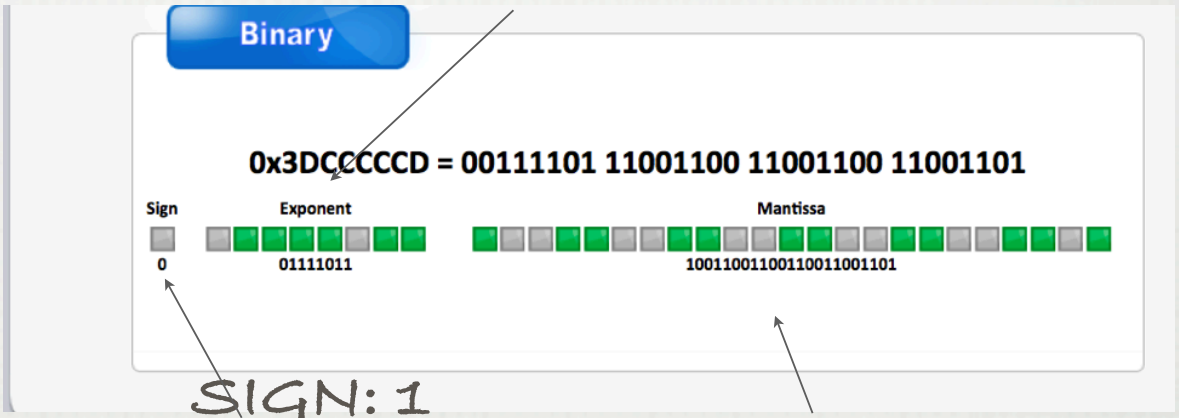
- For example, the decimal number 0.1 is not representable in binary floating-point of any finite precision;
- The exact binary representation would have a "1100" sequence continuing endlessly:
  - e = −4; s = 1100110011001100110011001100110011...,
- When rounded to 24 bits this becomes
  - e = −4; s = 110011001100110011001101
- which is actually 0.100000001490116119384765625 in decimal

CONVERSION SINGLE PRECISION

http://www.binaryconvert.com/



EXPONENT: 8 DIGITS : 123_{10}



SIGN: 1 DIGIT

MANTISSA: 23 DIGITS

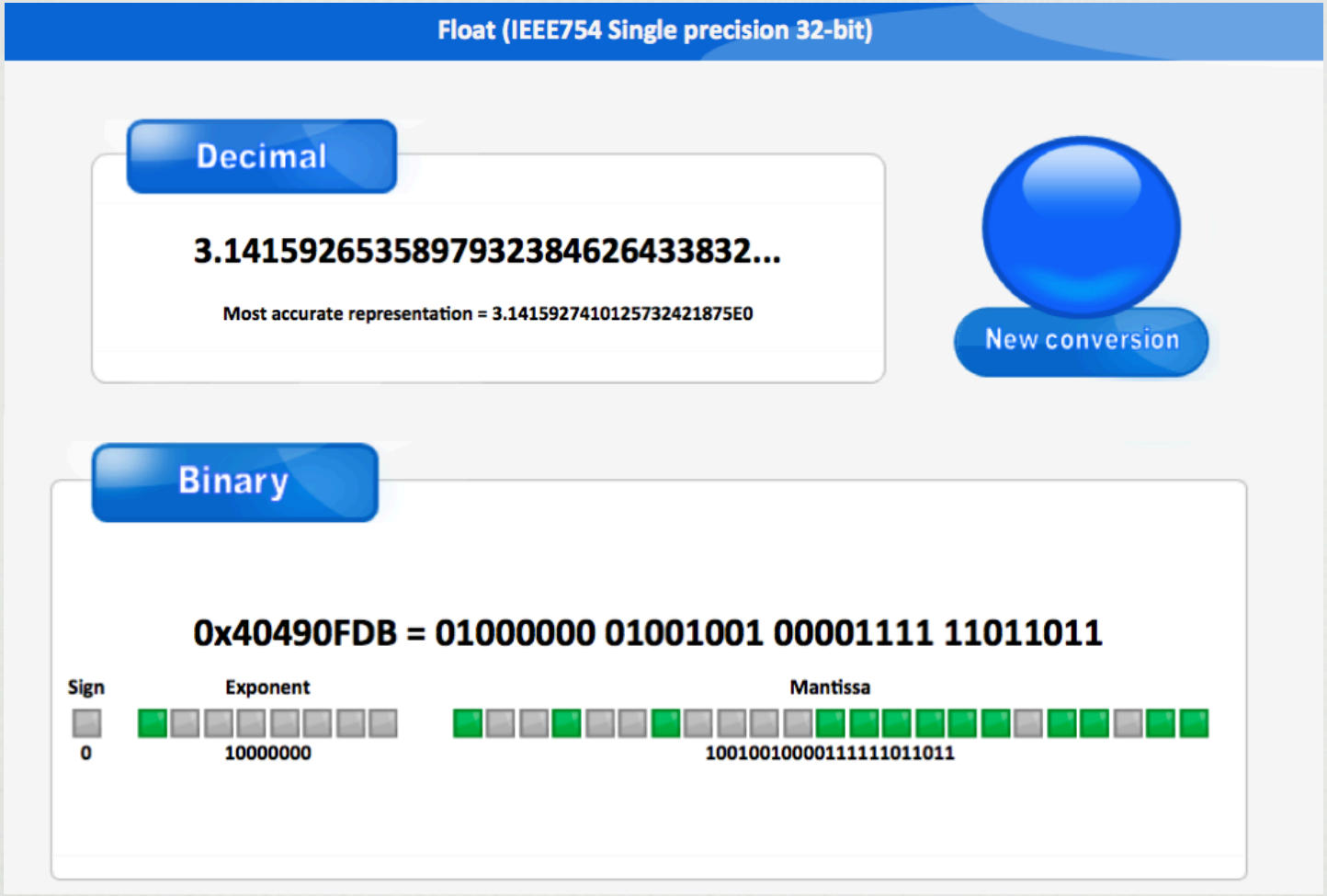
Exponent: 123_{10} with bias 127_{10} : -4 POSITIONS
Therefore: shift by -4 positions, adding the phantom digit, we get:
$0.0001100110011001100110011101_2$

EXAMPLES OF NON-REPRESENTABILITY

- IN 24-BIT (SINGLE PRECISION) REPRESENTATION, 0.1 (DECIMAL) WAS GIVEN PREVIOUSLY AS
  - E = −4; S = 11001100110011001100101, WHICH IS
  - 0.100000001490116119384765625 EXACTLY.
- SQUARING THIS NUMBER GIVES
  - 0.010000000298023226097399174250313080847263336181640625 (EXACT)
- SQUARING IT WITH SINGLE-PRECISION FLOATING-POINT HARDWARE (WITH ROUNDING) GIVES
  - 0.010000000707805156707763671875 EXACTLY.
- BUT THE REPRESENTABLE NUMBER CLOSEST TO 0.01 IS
  - 0.009999999776482582092285156250 EXACTLY.

EXAMPLE OF PI

- As a further example, the real number π, represented in binary as an infinite series of bits is
  - 11.00100100001111110110101010001000100001011010001100001000110100 11...
- but when approximated by rounding to a precision of 24 bits:
  - 11.00100100001111110110 11
- In binary single-precision floating-point, this is represented as
  - s = 1.00100100001111110110 11 with e = 1. (with shift of 127)
- This has a decimal value of 3.14159274101257324218 75
  - (compare with π = ~ 3.14159265358979323846264338327 95)



Exponent: 128_{10} with bias: 1_{10},
Therefore: shift by one position, adding the phantom digit, we get:
$11.00100100001111111011011_2$

IS ADDITION A COMMUTATIVE/ ASSOCIATIVE OPERATION?

- **WE WORK WITH 7 DIGITS**
- **A = 1234.567, B = 45.67834, C = 0.0004**
- **(A + B) + C:**
  - **1234.567 (A) + 45.67834 (B) = 1280.24534 ROUNDS TO 1280.245**
  - **1280.245 (A + B) + 0.0004 (C) = 1280.2454 ROUNDS TO 1280.245 <--- (A + B) + C**
- **A + (B + C):**
  - **45.67834 (B) + 0.0004 (C) = 45.67874**
  - **45.67874 (B + C) + 1234.567 (A) = 1280.24574 ROUNDS TO 1280.246 <--- A + (B + C)**
- **OOPS!**

MACHINE PRECISION: DEFINITION

- THE PRECEDING LOSS OF PRECISION IS CATEGORIZED BY DEFINING THE MACHINE PRECISION E_M AS THE MAXIMUM POSITIVE NUMBER THAT, ON THE COMPUTER, CAN BE ADDED TO THE NUMBER STORED AS 1 WITHOUT CHANGING THAT STORED 1:

$$1_c + \epsilon_m \quad \stackrel{\text{def}}{=} \quad 1_c$$

- WHERE THE SUBSCRIPT C IS A REMINDER THAT THIS IS A COMPUTER REPRESENTATION OF 1.

TAKE-HOME

- ALTHOUGH SOME NUMBERS ARE REPRESENTED EXACTLY (POWERS OF 2), **WE SHOULD ASSUME THAT ALL SINGLE-PRECISION NUMBERS CONTAIN AN ERROR IN THEIR SIXTH DECIMAL PLACE, AND THAT ALL DOUBLES HAVE AN ERROR IN THEIR 15TH PLACE**.
- AND AS IS ALWAYS THE CASE WITH ERRORS, THERE IS NO WAY TO KNOW AHEAD OF TIME WHAT THE ERROR IS, FOR IF WE COULD, THEN WE WOULD GET RID OF THE ERROR! CONSEQUENTLY, THE ARGUMENTS WE PUT FORTH REGARDING ERRORS ARE ALWAYS APPROXIMATE, AND THAT IS THE BEST WE CAN DO.

MACHINE PRECISION

- A MAJOR CONCERN OF COMPUTATIONAL SCIENTISTS IS THAT THE FLOATING-POINT REPRESENTATION USED TO STORE NUMBERS IS OF LIMITED PRECISION.
- IN GENERAL FOR A 32- BIT WORD MACHINE, SINGLE-PRECISION NUMBERS USUALLY ARE GOOD TO 6–7 DECIMAL PLACES WHILE DOUBLES ARE GOOD TO 15–16 PLACES.

IEEE 754 - 2008	Common name	C++ data type	Base b	*Fractional* digits p	Machine epsilon $b^{-p}/2$	C++ or Python formula	Value
binary16	half precision	not available	2	10	2^{-11}	pow (2, - 11)	4.88e-04
binary32	single precision	float	2	23	2^{-24}	pow (2, - 24)	5.96e-08
binary64	double precision	double	2	52	2^{-53}	pow (2, - 53)	2.22e-16
binary128	quad(ruple) precision	long double	2	112	2^{-113}	pow (2, - 113)	9.63e-35

TO SEE HOW LIMITED PRECISION AFFECTS CALCULATIONS, CONSIDER THE SIMPLE COMPUTER ADDITION OF TWO SINGLE-PRECISION WORDS:

$$7 + 1.0 \times 10^{-7} = ?$$

- 7 = 0 10000001 1100 0000 0000 0000 0000 000 (MISTAKE IN BOOK)
  - e=128+1=129, bias=127 => 1.11 to move 2 places: $111=7_{10}$
- 10^{-7} = 0 01100111 1010 1101 0111 1111 0010 101
- Because the exponents are different, it would be incorrect to add the mantissas, and so the exponent of the smaller number is made larger while progressively decreasing the mantissa by shifting bits to the right (inserting zeros), until both numbers have the same exponent, ADDING ZEROS
- 10^{-7} = 0 01100111 1010 1101 0111 1111 0010 101
  0 10000001 0000 0000 0000 0000 0000 000(1)
- $\Rightarrow$ $7 + 10^{-7}=7$

ERRORS

TYPES OF ERRORS

- **Blunders or bad theory:**
  - Errors entered with your program or data, having a fault in your reasoning (theory), using the wrong data file,...
- **Random errors:**
  - Errors caused by events such as fluctuation in electronics due to power surges, cosmic rays, ...
- **Approximation errors:**
  - Errors arising from simplifying the mathematics so that a problem can be solved or approximated on the computer.
- **Roundoff errors:**
  - Imprecisions arising from the finite number of digits used to store floating-point numbers.
  - These “errors” are analogous to the uncertainty in the measurement of a physical quantity The overall error arising from using a finite number of digits to represent numbers ACCUMULATES DURING RUN...
  - In some cases, the roundoff error may become the major component in your answer, leading to what computer experts call garbage.

MODEL FOR DISASTER (1)

- An operation performed on a computer usually only approximates the analytic answer because the numbers are stored only approximately.
- To demonstrate the effect of this type of uncertainty, let us call x_c the computer representation of the exact number x. The two are related by

$$x_c \simeq x(1+\epsilon_x)$$

- where ε_x is the relative error in x_c, which we expect to be of a similar magnitude to the machine precision ε_m.
- Let's now apply this notation to the simple subtraction $a = b - c$, we obtain

MODEL FOR DISASTER (2)

- If we apply this notation to the simple subtraction $a = b - c$, we obtain

$$a = b - c \quad \Rightarrow \quad a_c \simeq b_c - c_c \simeq b(1+\epsilon_b) - c(1+\epsilon_c)$$
$$\Rightarrow \quad \frac{a_c}{a} \simeq 1 + \epsilon_b \frac{b}{a} - \frac{c}{a}\epsilon_c$$

- The resulting error in *a* is essentially a weighted average of the errors in *b* and *c*, with no assurance that the terms will cancel.

- Of special importance here is to observe that the error in the answer *a* increases when we subtract two nearly equal numbers (b ≈ c), because then we are subtracting off the most significant parts of both numbers and leaving the error-prone least significant parts.

- If you subtract two large numbers and end up with a small one, there will be less significance in the small one.

- In terms of our error analysis, if the answer from your subtraction a is small, it must mean b ≃ c and so

$$\frac{a_c}{a} \overset{\text{def}}{=} 1+\epsilon_a \simeq 1+\frac{b}{a}(\epsilon_b-\epsilon_c) \simeq 1+\frac{b}{a}\max(|\epsilon_b|,|\epsilon_c|)$$

- This shows that even if the relative errors in b and c may cancel somewhat, they are multiplied by the large number b/a, which can significantly magnify the error. **Because we cannot assume any sign for the errors, we must assume the worst**

OTHER EXAMPLE OF DISASTER

$$(x+y)(x-y) = x^2 - y^2$$

- Suppose x, y have 4 digits precision with *x=9.876; y=1.234*
- *x+y=11.11; x-y=8.642*
  - *(x+y)*(x-y)=96.01*
- *x^2=97.54; y^2=1.523*
  - *x^2-y^2=96.02* (exact => *96.01262*)

EXAMPLE OF ACCUMULATED ROUNDOFF ERROR

- Error from a single multiplication of the computer representation of two numbers:

$$a = b \times c \quad \Rightarrow \quad a_c = b_c \times c_c = b(1+\epsilon_b) \times c(1+\epsilon_c)$$
$$\Rightarrow \quad \frac{a_c}{a} = (1+\epsilon_b)(1+\epsilon_c) \simeq 1+\epsilon_b+\epsilon_c$$

where we ignore very small ε^2 terms.

- This is just the basic rule of error propagation from elementary physics or engineering laboratory: you add the uncertainties in each quantity involved in an analysis in order to determine the overall uncertainty.

- As before, the safest assumption is that there is no cancellation of error, that is, that the errors add in absolute value.

ROUNDOFF ERROR ACCUMULATION: RANDOM WALK MODEL

- We view the error in each step as a literal "step" in a random walk, that is, a walk for which each step is in a random direction.

$$R \approx \sqrt{N}\, r$$

- The total distance covered in *N* steps of length *r*, is, on average,

$$\epsilon_{\mathrm{ro}} \approx \sqrt{N}$$

- By analogy, the total relative error ε_{ro} arising after *N* computing steps each with the machine precision error εm, is, on average,

NOT ALWAYS A MODEL FOR ROE

- **IF THE ROUNDOFF ERRORS IN A PARTICULAR ALGORITHM DO NOT ACCUMULATE IN A RANDOM MANNER, THEN A DETAILED ANALYSIS IS NEEDED TO PREDICT THE DEPENDENCE OF THE ERROR ON THE NUMBER OF STEPS N.**
- **IN SOME CASES THERE MAY BE NO CANCELLATION AND THE ERROR MAY INCREASE LIKE NEM.**
- **EVEN WORSE, IN SOME RECURSIVE ALGORITHMS, WHERE THE PRODUCTION OF ERRORS IS COHERENT, THE ERROR INCREASES LIKE N!**

HOW TO MINIMIZE ERRORS?

- **Naive use of floating-point arithmetic can lead to many problems. Good understanding of numerical analysis is essential.**
- **Small errors in floating-point arithmetic can grow when mathematical algorithms perform operations an enormous number of times.**
  - **examples: matrix inversion, eigenvector computation, and differential equation solving.**
- **Expectations from mathematics may not be realized in the field of floating-point computation.**
- **The use of the equality test (*if (x==y)* ...) is usually not recommended when dealing with floating point numbers.**

GOOD PROGRAMMING PHILOSOPHY

- **TEST TEST TEST TEST TEST.....**

EXERCISES:

- Determine your machine precision
- Error assessment: how many steps do you need to compute this infinite series?

$$e^{-x} \simeq \sum_{n=0}^{N} \frac{(-x)^n}{n!}$$

  - Compare with built-in function
  - For x<1, should you start the summation from large numbers (increasing n) or small numbers (decreasing n)?
- Using pencil/paper, find the binary representation of number 1.252011; what is its IEEE single precision representation on the computer?

LECTURE 1: SUMMARY

- Numerical errors are parts of computational experiments, because of the way numbers can be represented numerically
- Other errors must be "lived with", such as approximation in algorithms, and random errors
- Some errors cannot be "lived with", such as blunders in theory, or naive programming practice.
- Test your code and do not trust compilers blindly!
- Remember a computer does exactly as being told to do...it is not your friend nor your enemy...

NEXT WEEK:
NUMERICAL INTEGRATION & DIFFERENTIATION