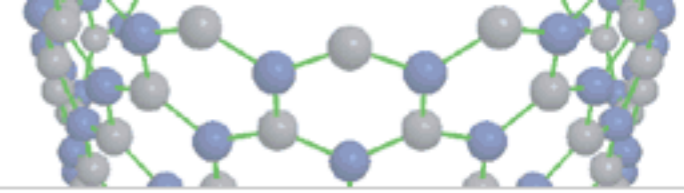


# Computational physics



UNIVERSITY OF  
ALBERTA

*Phys 420/580 Lecture 1*



## About me

Contact info  
CV

## Group

Research  
People  
Publications  
Recruiting

## Talks

Colloquium  
CMP seminar

## Courses

Phys 234  
Phys 308  
Phys 420/580  
Phys 499  
Grading  
Advice

## Guides

C++  
LAPACK  
REVTex  
EPS figures  
Journals

course  
webpages

technical  
info

## Welcome!

I'm a condensed matter theorist at the University of Alberta, working primarily in computational many-body physics. This site records some of my ongoing [research activities](#). It also includes several how-to guides and web pages for the courses I'm currently teaching.

$$\begin{aligned} F &= \text{[diagram 1]} + \text{[diagram 2]} + \text{[diagram 3]} + \text{[diagram 4]} + \text{[diagram 5]} + \text{[diagram 6]} + \text{[diagram 7]} \\ &+ \text{[diagram 8]} + \text{[diagram 9]} + \text{[diagram 10]} + \text{[diagram 11]} + \text{[diagram 12]} + \text{[diagram 13]} + \text{[diagram 14]} \\ &+ \text{[diagram 15]} + \text{[diagram 16]} + \text{[diagram 17]} + \text{[diagram 18]} + \text{[diagram 19]} + \text{[diagram 20]} + \text{[diagram 21]} \\ &+ \text{[diagram 22]} + \text{[diagram 23]} + \text{[diagram 24]} + \text{[diagram 25]} + \text{[diagram 26]} + \dots \end{aligned}$$

[^ Top](#)

© 2009 Kevin Beach. All rights reserved.

## Physics 420/580 – Fall 2010

Previous iterations of this course: [ [Fall 2008](#) : [Fall 2009](#) ]

links to previous  
years' pages

### Course outline

#### Computational Physics / Advanced Topics in Computational Physics

Lecture: CEB 123 MWF 13:00-13:50

Lab: CEB 114 R 14:00-16:50

Syllabus: [pdf](#)

Schedule: [pdf](#)

### Learning resources

detailed tutorials

- As you're getting started, I recommend that you work your way through these [introductory notes](#) and that you try solving some of the simple programming exercises.
- Check out these online tutorials for [C++](#) and [UNIX](#).
- Many of the lecture topics are covered in greater detail in various textbooks. I encourage you to at least peruse items 6–8 of this [book list](#). You can find them on reserve in [Cameron library](#).
- An old version of [Numerical Recipes](#), is available for free online. The newest edition (somewhat expensive) makes the jump from C to C++.
- A rather famous article: David Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," ACM Computing Surveys **23**, 5 (1991). [ [pdf](#) : [html](#) ]
- A short guide to 2D OpenGL graphics using the GLUT library [ [pdf](#) ]

lectures, labs,  
assignments all  
available online

### Exercises

These optional, ungraded problems touch on important issues that aren't addressed explicitly in the labs and assignments.

1. Floating-point numbers  
instructions [ [pdf](#) ], source [ [tar.gz](#) ]

### Lectures

Class starts on Wednesday, September 8.

1. [Wednesday, September 8](#) — Computational physics
2. [Friday, September 10](#) — C++ language review
3. [Monday, September 13](#) — Discretization

# The C++ Programming Language

## Overview

In contrast with some older languages, such as FORTRAN 77 or COBOL, C++ is freeform in the sense that whitespace (spaces, tabs, returns) has no meaning. The code can be laid out as you wish, provided that each individual command is separated by a semicolon (;) and blocks of commands are delineated by braces ({ ... }). Explanatory remarks, called *comments*, can be inserted by signalling to the compiler that certain parts of the program file are to be ignored. Single line comments begin with a double slash (//) and extend to the end of the current line. Blocks of comments are enclosed by matching slash-star pairs (/\* ... \*/). Both the following code listings are interpreted identically by the compiler.

```
int main()
{
    return 0; // zero is the standard return value
             // when there are no errors to report
}
```

```
// C++ code is freeform and all whitespace is treated equally

int  main( ){
    ; ;
    ; ;
    return 0 ;}

/* Related blocks of code are enclosed in matching pairs
of braces. Each individual command is followed by a
semicolon. In the function above, there are four "do
nothing" commands and one return statement. */
```

These two examples are variations on the null program, `int main(){return 0;}`, which is the simplest possible. (It begins, does nothing, and ends.) Every valid C++ program must contain exactly one function called `main` that returns an integer value to the operating system. Program flow begins with the call to `main` and terminates when all the statements in `main` have been executed.

Roughly speaking, everything in the C++ language is either an object or a function. An object occupies memory (to store its data) and has a definite *type* associated with it. A function acts

### Table Of Contents

#### The C++ Programming Language

- [Overview](#)
- [Identifiers and keywords](#)
- [Types and literals](#)
- [Operations](#)
- [Type conversion](#)
- [Control structures](#)
  - [Branching](#)
  - [Looping](#)
  - [Logical skeletons](#)
- [Functions](#)
  - [Arguments to functions](#)
  - [Prototypes](#)
  - [Functions as arguments](#)
  - [Recursion](#)
  - [Templates](#)
  - [Standard Library](#)
- [Input/output](#)
  - [Command line arguments](#)
  - [Streams](#)
  - [Text files](#)
  - [Binary files](#)

### Previous topic

### UNIX Tools

### Next topic

### A Quick Review of C++

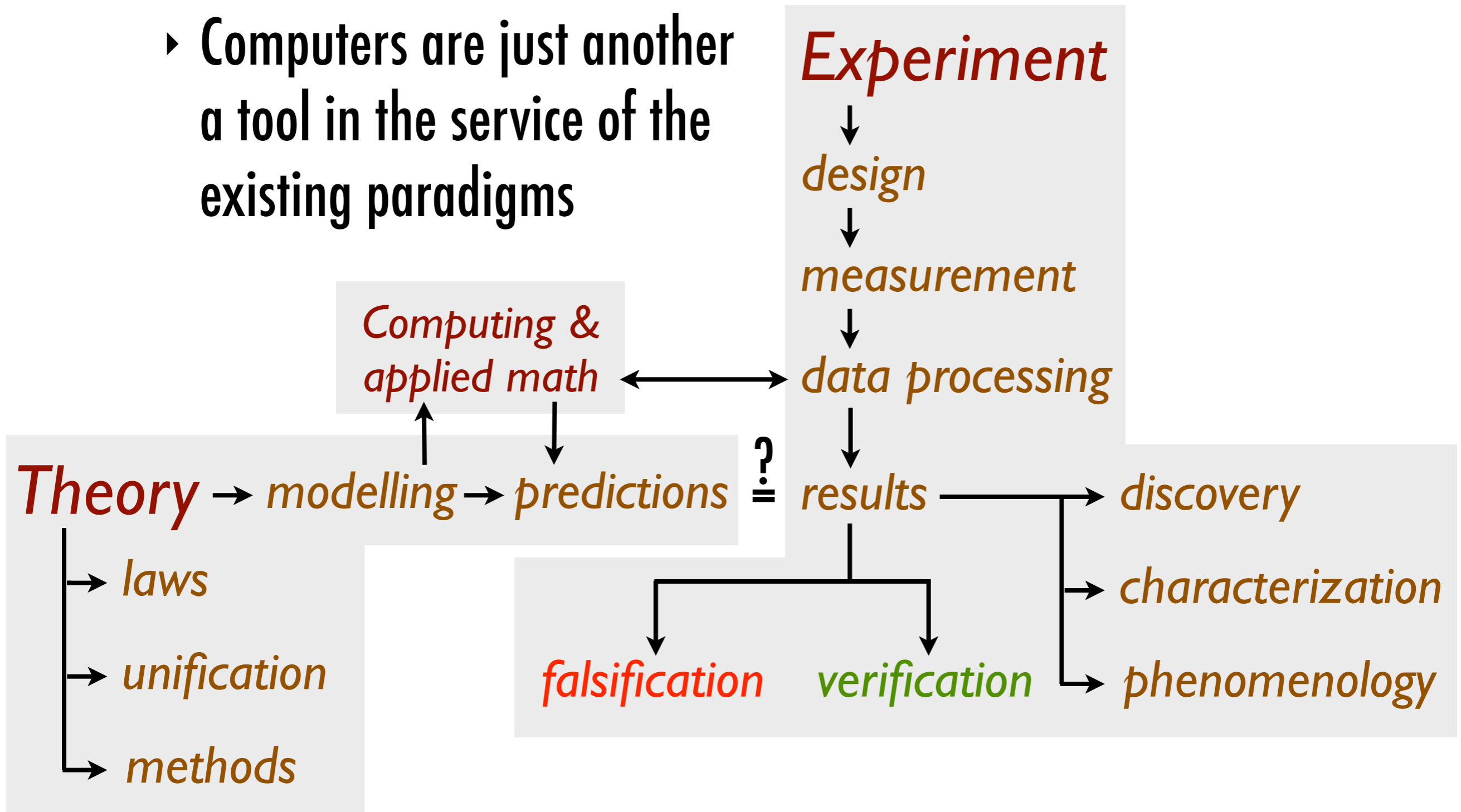
### Quick search

# What is computational physics?

- ▶ A **branch of physics** in its own right and an important bridge between theory and experiment
- ▶ Concerned with the development and implementation of numerical **algorithms** that can **simulate** complex physical behaviour
- ▶ Employed extensively in almost every physics discipline

# The purely instrumental view

- ▶ Computers are just another a tool in the service of the existing paradigms



# The purely instrumental view

- ▶ Contact between **theory** and **experiment** relies on our making **quantitative** comparisons
- ▶ The computer plays a supporting role:
  - ▶ treat by numerical methods models that cannot be solved analytically *can we solve any models by hand?!*
  - ▶ manipulate experimental data (background subtraction, Fourier analysis, etc.)  
*is it always necessary to perform an experiment?*

# A richer view

*Theory*

*Computational physics*

*Experiment*

*simulation*

*numerical experiment*

*discovery*

*hypothesis  
testing*

*engineering  
in silico*

*inaccessible  
regimes*

e.g., chaos,  
fractals, strange  
attractors

e.g., quark  
confinement  
in QCD

e.g., designer  
materials  
like graphene

e.g., metallic  
hydrogen at  
ultra-high pressure

# A richer view

- ▶ Computational physics seen as a **co-equal branch** of physics, **complementary** to theory and experiment
- ▶ Draws on lessons from computer science and applied mathematics but applies insights from physics
- ▶ Many problems can be addressed by no other means: Wigner crystals, plasma near fusion ignition, stellar interiors, colliding black holes, galaxy formation, undiscovered elements, climate models, ...

# Important issues

- ▶ How do we make physics come alive in the machine?
- ▶ How do we construct algorithms?
- ▶ What are the key considerations for efficiency of storage and execution time?
- ▶ How do we apply insights from physics?
- ▶ What degree of realism is necessary?

# Algorithmic scaling

- ▶ How do the storage requirements and execution time scale as the size of the simulation grows?

- ▶ For  $N$

- ▶ particles ...

- ▶ time steps ...

- ▶ lattice spacings ...

algorithmic efficiency ↑

$O(\log N)$

$O(N)$

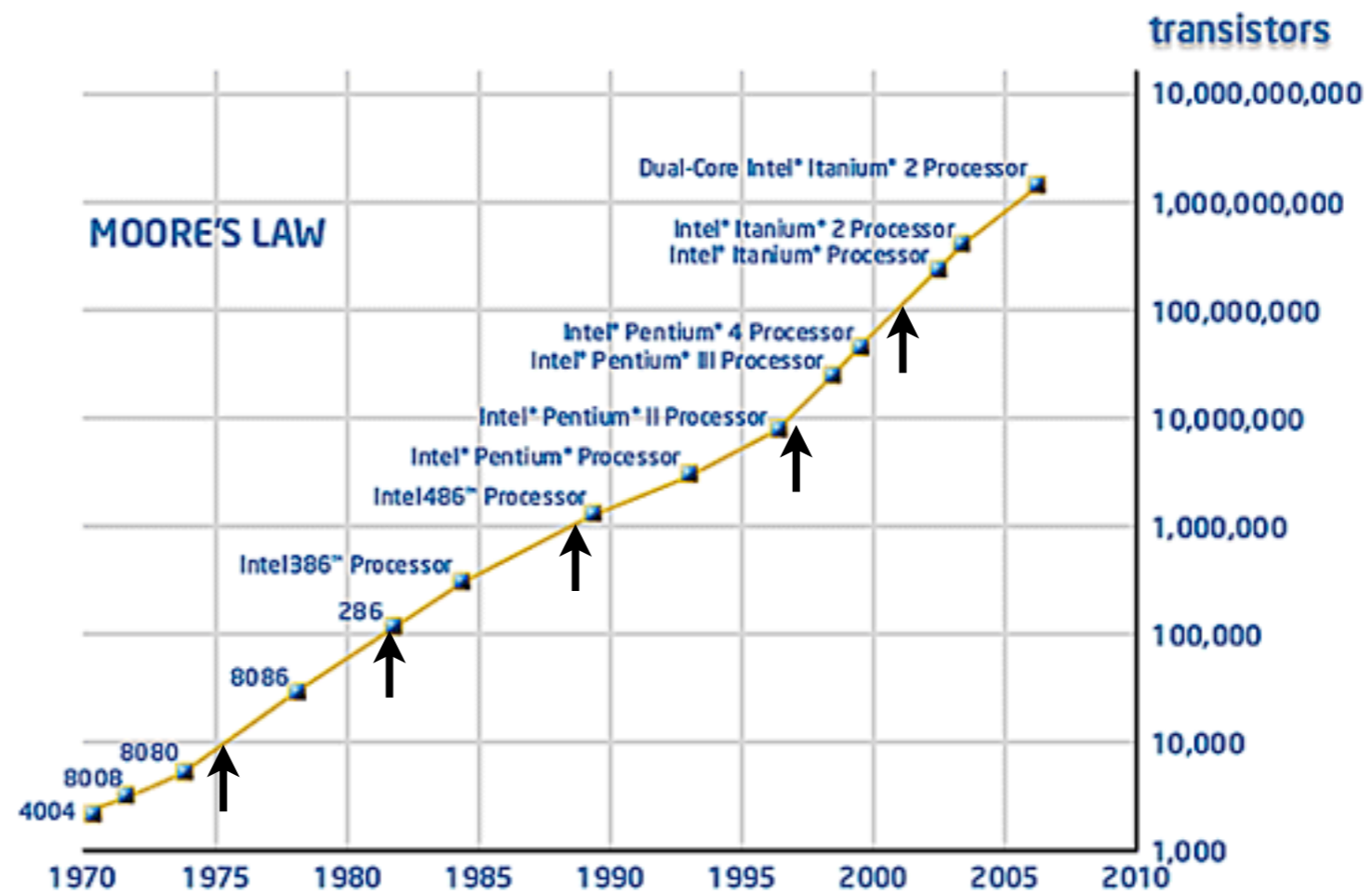
$O(N \log N)$

$O(N^2)$

$O(2^N)$

# Algorithmic scaling

<i>good</i>	$O(N \log N)$	$N=10$	$N=80$	$N=667$	$N=5720$	$N=50000$
<i>bad</i>	$O(N^2)$	$N=10$	$N=32$	$N=100$	$N=316$	$N=1000$
<i>ugly</i>	$O(2^N)$	$N=10$	$N=12.5$	$N=15$	$N=17.5$	$N=20$



# Degree of realism

