Introduction to Computational Physics

PHYS 250 (Autumn 2018) – Lecture 1

David Miller

Department of Physics and the Enrico Fermi Institute
University of Chicago

September 27, 2018

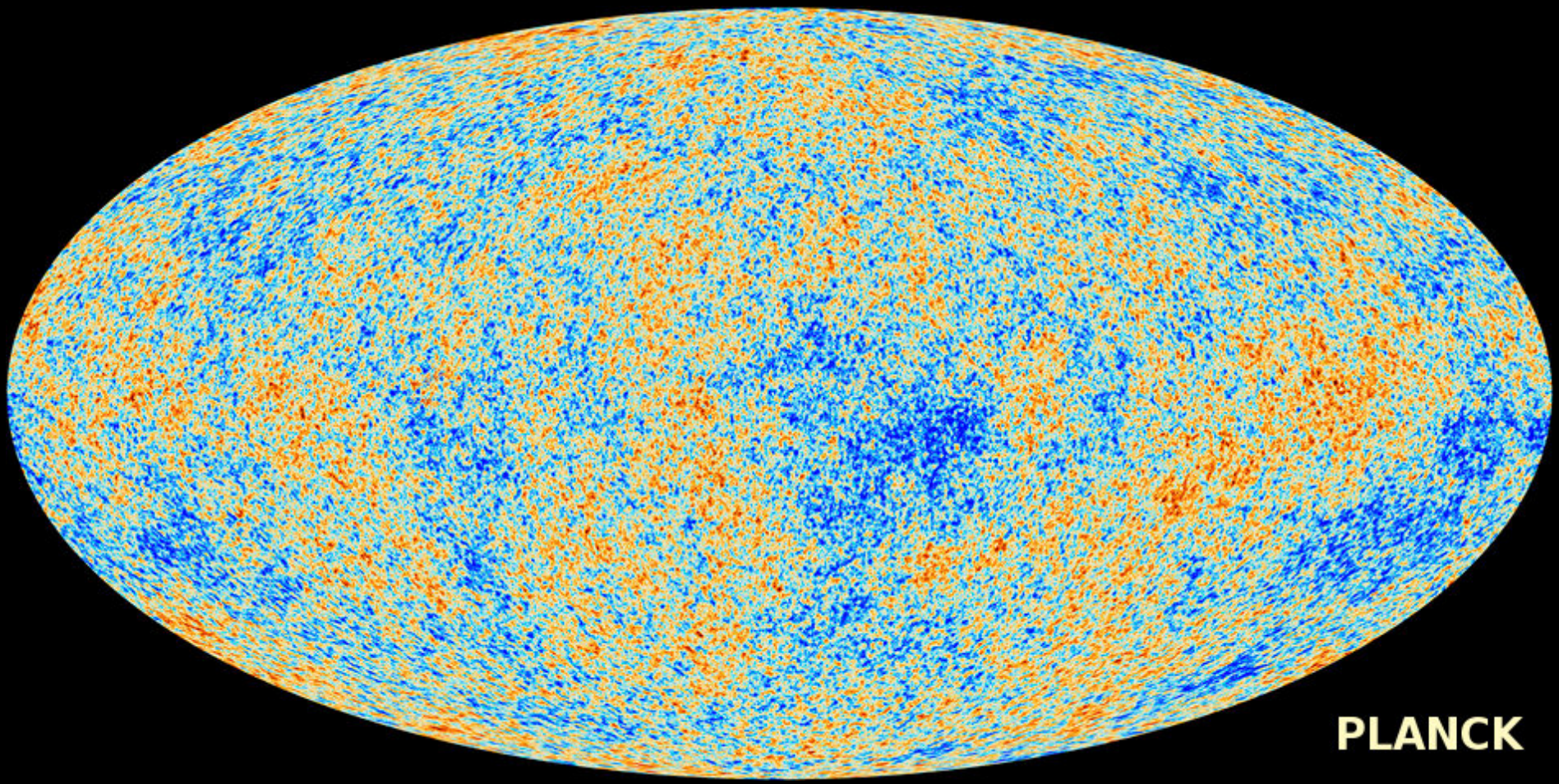
Outline

Computational Physics (PHYS 250)

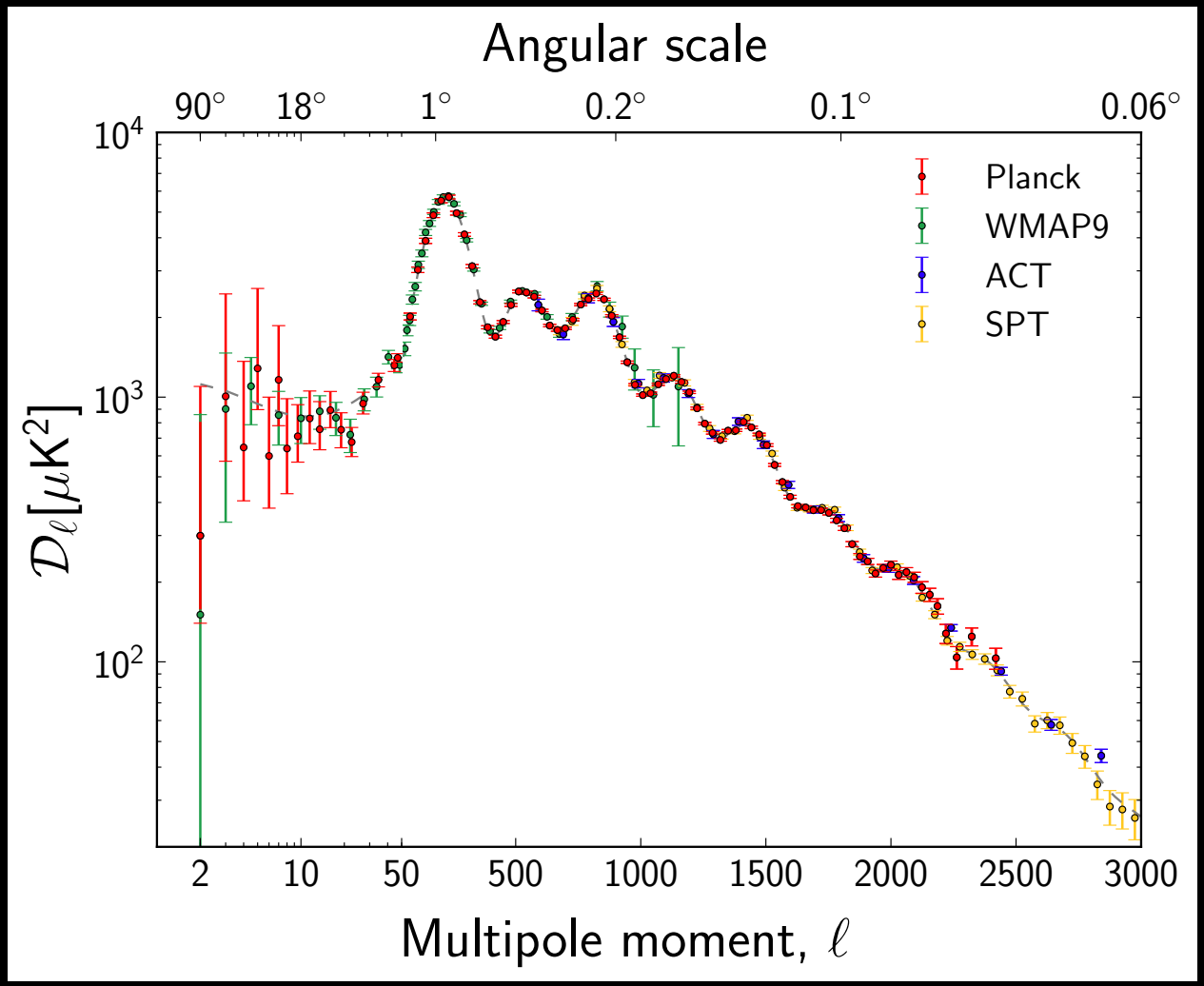
Course Description PHYS 250 (link to Course Catalog)

This course introduces the use of computers in the physical sciences. After an introduction to programming basics, we cover numerical solutions to fundamental types of problems, including cellular automatons, artificial neural networks, computer simulations of complex systems, and finite element analysis. Additional topics may include an introduction to graphical programming, with applications to data acquisition and device control.

There are an infinite number of paths that we might follow and still not deviate from this description. I therefore would like to lay out some of the principles that will guide me, and us, in how we navigate through those many possibilities.



The Nature of Science: Science is all about models. We look at something in real life and try to make a model of it. We can use this model to predict future (or new) events in real life. If the model doesn't agree with real data, we change the model. Repeat forever. [From Wired]



The Nature of Science: Science is all about models. We look at something in real life and try to make a model of it. We can use this model to predict future (or new) events in real life. If the model doesn't agree with real data, we change the model. Repeat forever. [From Wired]

Theory and Practice (or...Theory and Experiment)

We often divide the world of science (and perhaps even moreso, Physics) into two deeply related and intertwined but often distinct categories of activities and research:

- **Theoretical:** building models
- **Experimental:** testing models

We use computers to do *both* of these crucial activities!

In doing so, we often blur the distinction between the two, as well.

Overarching Learning Goals for this Quarter

- **Identify models** that benefit from or require **computational/numerical approaches** and tools to either develop and understand or to evalaute.
- Develop an **algorithmic approach** to addressing those problems computationally.
- Familiarity with **common computational models and algorithms** and numerical approaches to typical problems.
- Use **modern, high-level programming languages** to implement the computational algorithms.
- Use modern software tools for **developing, preserving, disseminating, and expanding** on those solutions.
- **Apply computational tools** to contemporary physics questions.

The role of the physicist

"The computer is incredibly fast, accurate, and stupid. Man is incredibly slow, inaccurate, and brilliant. The marriage of the two is a force beyond calculation."

→ Maybe this is a quote from Albert Einstein, maybe it is from Leo Cherne...I'm not sure, but it's certainly true.

Outline

Algorithmic and process-based thinking

You probably already learned in your first year courses that certain approaches to problem solving in physics are important to efficiently identifying solutions and paths through a particular problem.

Before we get into any code, let's discuss the thought process behind developing a computational model or approach to problem solving.

Thinking like a machine

From the Kinder & Nelson text

Human-readable (high level) instructions to start a car

```
Put the key in the ignition
Turn the key until the engine starts,
    then let go
Push the button on the shifter and move it
    to REVERSE
...
```

Bug!!!

```
Press down the left pedal # Needed for some cars
```

Machine-readable (low level) instructions

```
Grab the wide end of the key
Insert pointed end of key into slot on lower
    right side of steering column
Rotate key clockwise about long axis when viewed
    toward the pointed end
```

Thinking like a machine

From the Kinder & Nelson text

Human-readable (high level) instructions to start a car

```
Put the key in the ignition
Turn the key until the engine starts,
    then let go
Push the button on the shifter and move it
    to REVERSE
...
```

Bug!!!

```
Press down the left pedal # Needed for some cars
```

Machine-readable (low level) instructions

```
Grab the wide end of the key
Insert pointed end of key into slot on lower
    right side of steering column
Rotate key clockwise about long axis when viewed
    toward the pointed end
```

Thinking like a machine

From the Kinder & Nelson text

Human-readable (high level) instructions to start a car

```
Put the key in the ignition
Turn the key until the engine starts,
    then let go
Push the button on the shifter and move it
    to REVERSE
...
```

Bug!!!

```
Press down the left pedal # Needed for some cars
```

Machine-readable (low level) instructions

```
Grab the wide end of the key
Insert pointed end of key into slot on lower
    right side of steering column
Rotate key clockwise about long axis when viewed
    toward the pointed end
```

Thinking like a machine

From the Kinder & Nelson text

Human-readable (high level) instructions to start a car

```
Put the key in the ignition
Turn the key until the engine starts,
    then let go
Push the button on the shifter and move it
    to REVERSE
...
```

Bug!!!

```
Press down the left pedal # Needed for some cars
```

Machine-readable (low level) instructions

```
Grab the wide end of the key
Insert pointed end of key into slot on lower
    right side of steering column
Rotate key clockwise about long axis when viewed
    toward the pointed end
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                  # Input
calculate area of circle     # Numeric
print area                   # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                  # Input
calculate area of circle     # Numeric
print area                   # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                 # Input
calculate area of circle    # Numeric
print area                  # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                 # Input
calculate area of circle    # Numeric
print area                  # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                  # Input
calculate area of circle     # Numeric
print area                   # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                  # Input
calculate area of circle     # Numeric
print area                   # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")   # Input
pi     = 3.141593                  # Set constant
area   = pi  * radius^2            # Algorithm
print("Area = " + area)            # Output
```

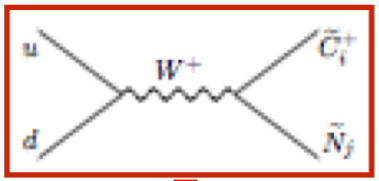
Translate algorithmic thinking into a development process

We can adapt to the need for algorithmic thinking by adopting a process for developing algorithms, computational approaches, and software generally.

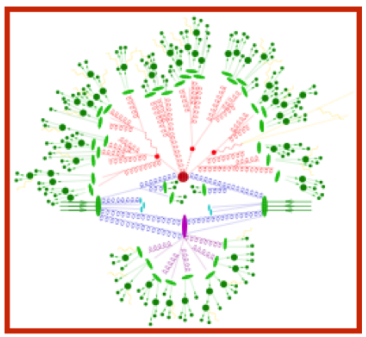
Step 1: write the algorithm down on paper
Step 2: think
    If: you don't understand everything
    Then: `goto` Step 1
    Else: `continue`
Step 3: write pseudocode
Step 4: think
    If: you don't understand everything
    Then: `goto` Step 3
    Else: `continue`
Step 5: write actual code
Step 6: test code with unit tests
    If: code does not unit checks
    Then: `goto` Step 5
    Else: `continue`
Step 7: publish!

Alas, sometimes, it's a bit more complicated than that...

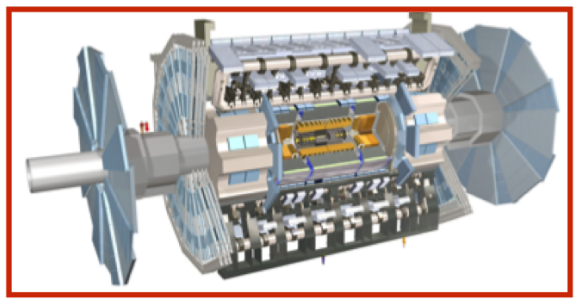
Simulation of hard-scatter for signal + SM background



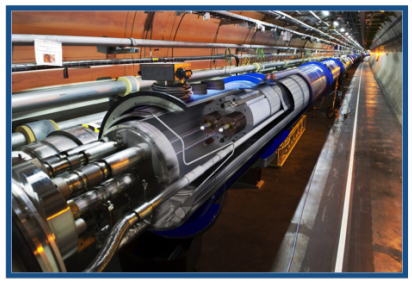
Simulation of "soft physics" (shower + hadronization)



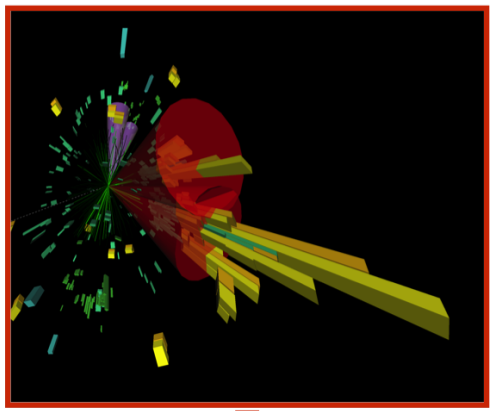
Detector simulation



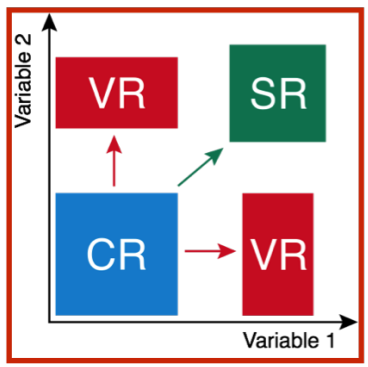
LHC real data



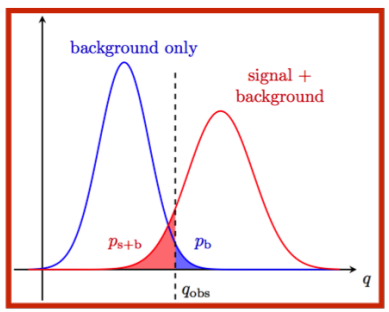
Reconstruction of physical observables



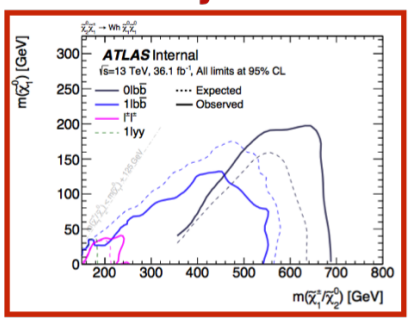
Event selection and background estimation



Hypothesis testing



Discovery or exclusion



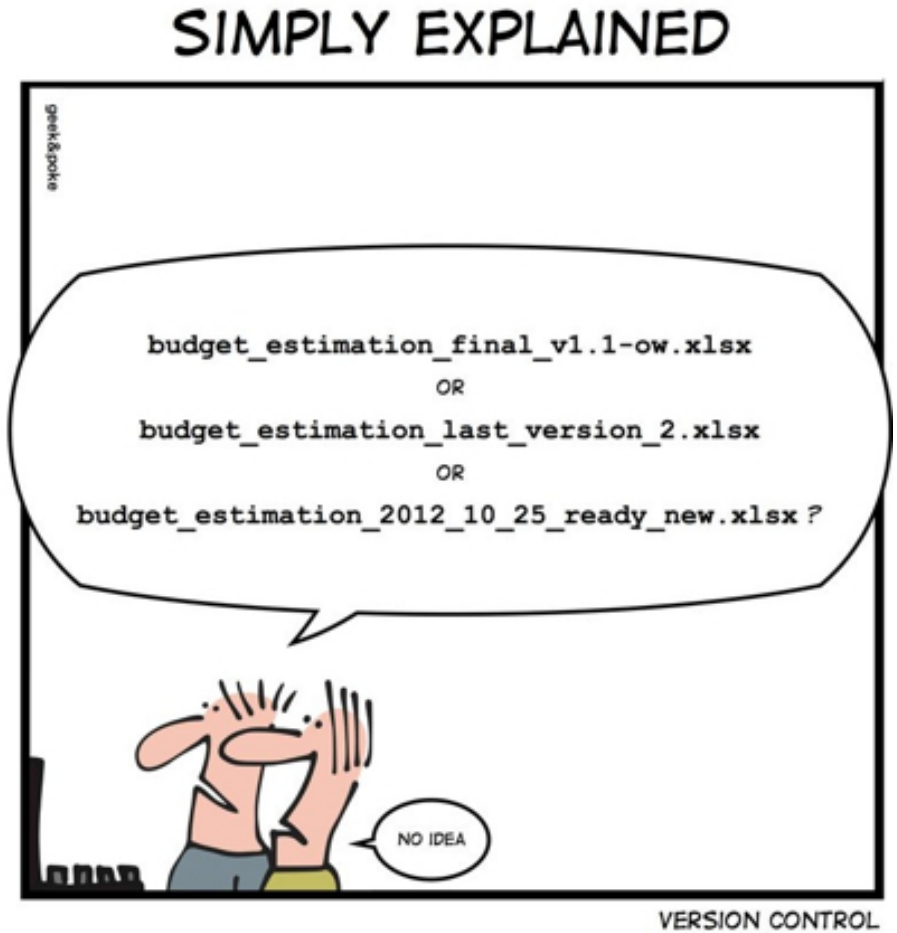
Version control

- The most important message of thisslide is simple...**Use a software revision control system for all of your code**
  - And that means now...not tomorrow or next week
  - Because if you wait until you need it, it will be too late

Version control

- The most important message of thisslide is simple...**Use a software revision control system for all of your code**
  - And that means now...not tomorrow or next week
  - Because if you wait until you need it, it will be too late

GitHub



A brief history of version control

- **The first RCS were designed to be used on large systems where everyone logged into the same machine**
  - They tracked code on the same filesystem where it lived (e.g., in a subdirectory)
  - SCCS and RCS are examples
- **Then client-server systems were developed, so that developers could work on their own machines**
  - Checking code into a central server to share and collaborate
  - CVS and SVN are examples
- **More recently distributed version control systems have arisen**
  - These are decentralised, so everyone has a complete copy of the repository
  - Gives a lot of freedom to developers to share and merge as they like, so liked very much by the open source community
  - git, mercurial and bit keeper are examples

`git` & `GitHub`

`https://git-scm.com` and `https://github.com`

- **`git` is the most popular open source version control system**
  - can host huge projects (Linux Kernel, LHC experiment software, etc)
  - scales very well and it?s extremely fast and powerful
  - very flexible (= complex)
- **Distributed version control systems (`git`) are great, but they are made even better by using a social coding website (`GitHub`)**
- **These sites allow developers:**
  - browse code easily
  - compare different versions
  - take copies (a.k.a. fork)
  - offer patches back to upstream repositories
    - And discuss and review these patches before acceptance
  - even build websites
- **The best known social coding site is GitHub, but there are others, e.g. BitBucket and GitLab**

ATLAS Experiment analysis software package on **GitHub**

ATLAS Run II analysis framework for AnalysisTop and AnalysisBase for proton-smashing physics https://ucatlas.github.io/xAODAnaHelp...

physics-analysis | high-energy-physics | analysis-framework

1,911 commits | **16** branches | **130** releases | **40** contributors

Branch: **master** ▾ | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download** ▾

kratsg Custom jvt cuts need to use detector-eta (#1250) — Latest commit `ec9c86b` 17 hours ago

Root	Custom jvt cuts need to use detector-eta (#1250)	17 hours ago
ci	deploy to multiple repos instead (#1126)	9 months ago
cmake	Add bash command line completion for xAH_run.py in CMake (#1038)	a year ago
data	A few more changes to autoconfigure PRW (#1223)	3 months ago
docs	Add TauJet track matching (#1242)	18 days ago
python	remove is_release20. resolves #1243 (#1246)	4 days ago
scripts	remove is_release20. resolves #1243 (#1246)	4 days ago
xAODAnaHelpers	Keep OR of triggers together for extra chains (#1186)	17 days ago
.gitignore	large-R area info (#736)	2 years ago
.travis.yml	Fixing compilation warning for deprecated jet jvt efficiency tool hea... (	2 months ago
CMakeLists.txt	Fixing compilation warning for deprecated jet jvt efficiency tool hea... (	2 months ago
CONTRIBUTING.md	Add analysis top (#1125)	9 months ago
README.md	Remove RootCore functionality (#1124)	9 months ago

Hello world!

Code example

```
import numpy as np
import matplotlib.pyplot as plt

def p(x):
    return np.exp(-x**2)

#let's plot it
x = np.linspace(-3,3,100)
y = p(x)
plt.plot(x,y)
```

Hello world!

```
import numpy as np
import matplotlib.pyplot as plt

def p(x):
    return np.exp(-x**2)

#let's plot it
x = np.linspace(-3,3,100)
y = p(x)
plt.plot(x,y)
```

Outline

Conclusions

- Something