Title: Computational methods: (Now with data)

Date: Feb 09, 2012 11:15 AM

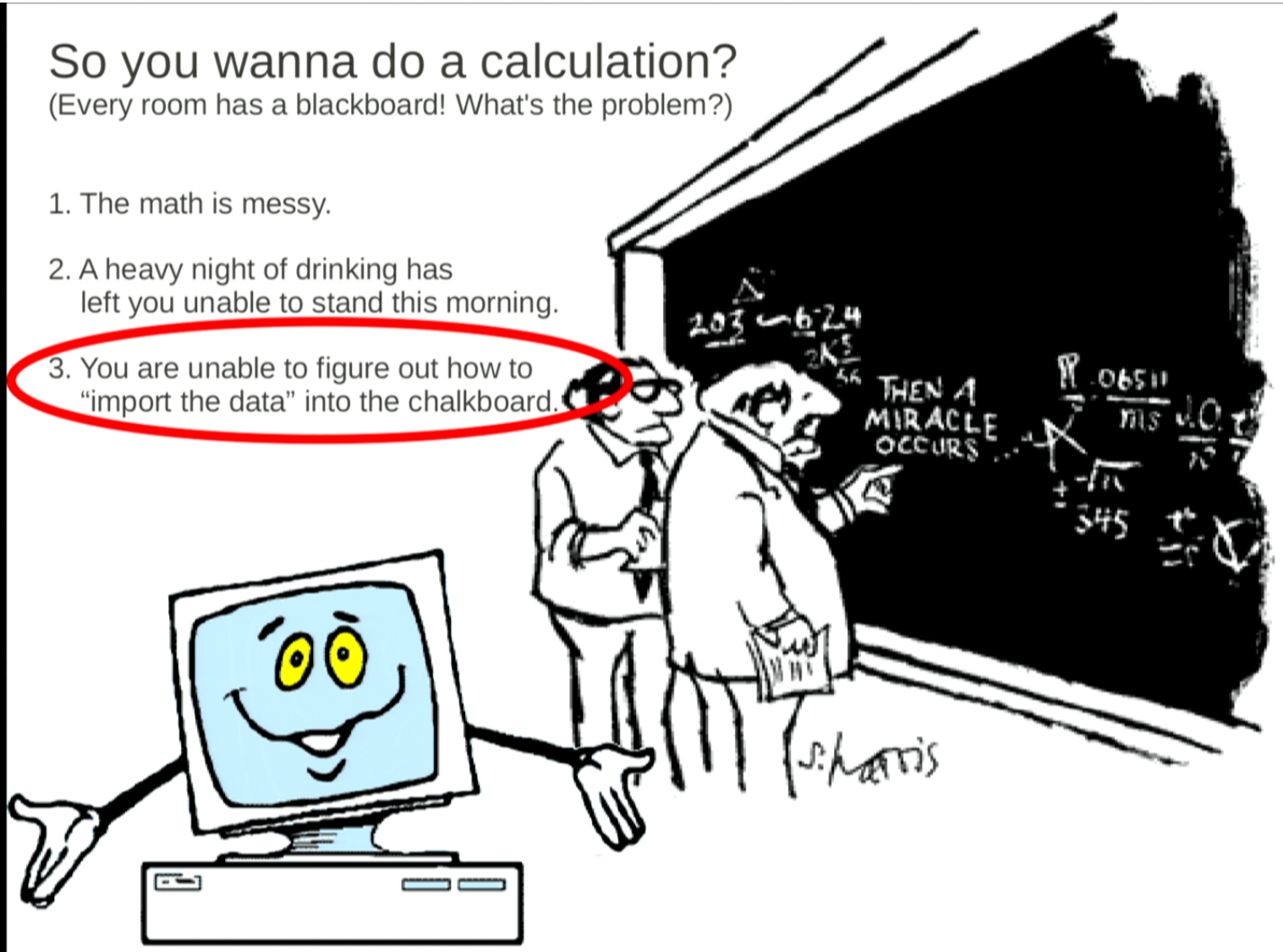
URL: http://pirsa.org/12020143

Abstract:

So you wanna do a calculation?

(Every room has a blackboard! What's the problem?)

1. The math is messy.
2. A heavy night of drinking has left you unable to stand this morning.
3. You are unable to figure out how to "import the data" into the chalkboard.



http://comtechsolutions.ca/sitebuildercontent/sitebuilderpictures/keepyourcomputerhappy.gif

https://www.ibm.com/developerworks/mydeveloperworks/blogs/InsideSystemStorage/resource/then-a-miracle-happens.gif

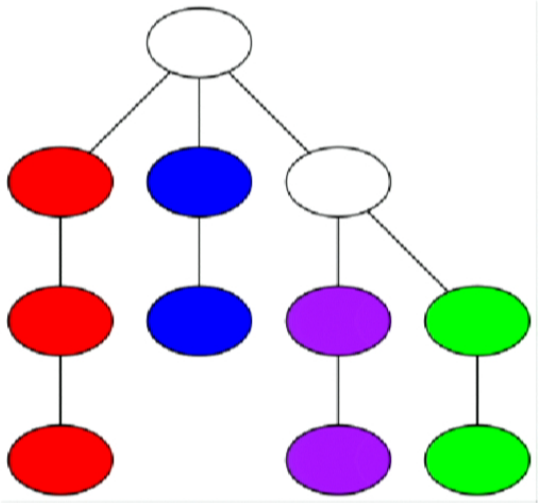
Lessons I have learned about analyzing data

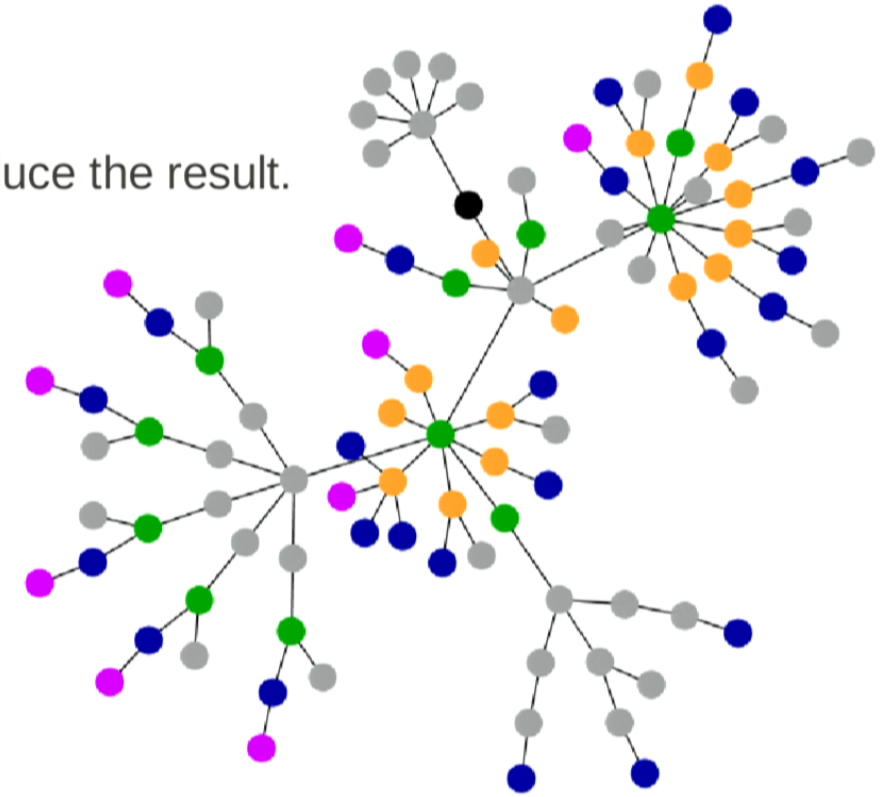
* Know the data.

* Know the scale.

* Know the software options.

* Know exactly how to reproduce the result.





Know the data

Trust but verify

Usually you can trust your data source, but that doesn't mean you can trust your data. Unintentional problems and errors can creep into datasets. Verify that they are correct.

1) Get an authoritative data set.

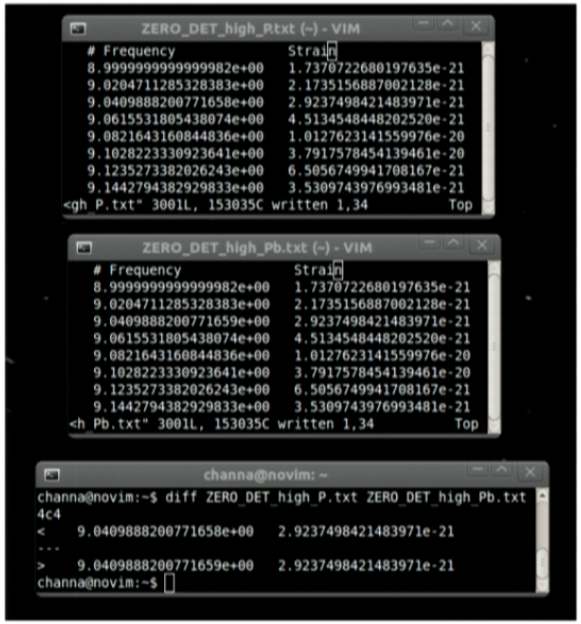
a) If small consider the retrieval of the data as a standard part of the analysis.
b) If large ask the provider for a checksum! (`wget, cksum, diff, cmp, ...`)

2) Data sets can be wrong for many reasons.

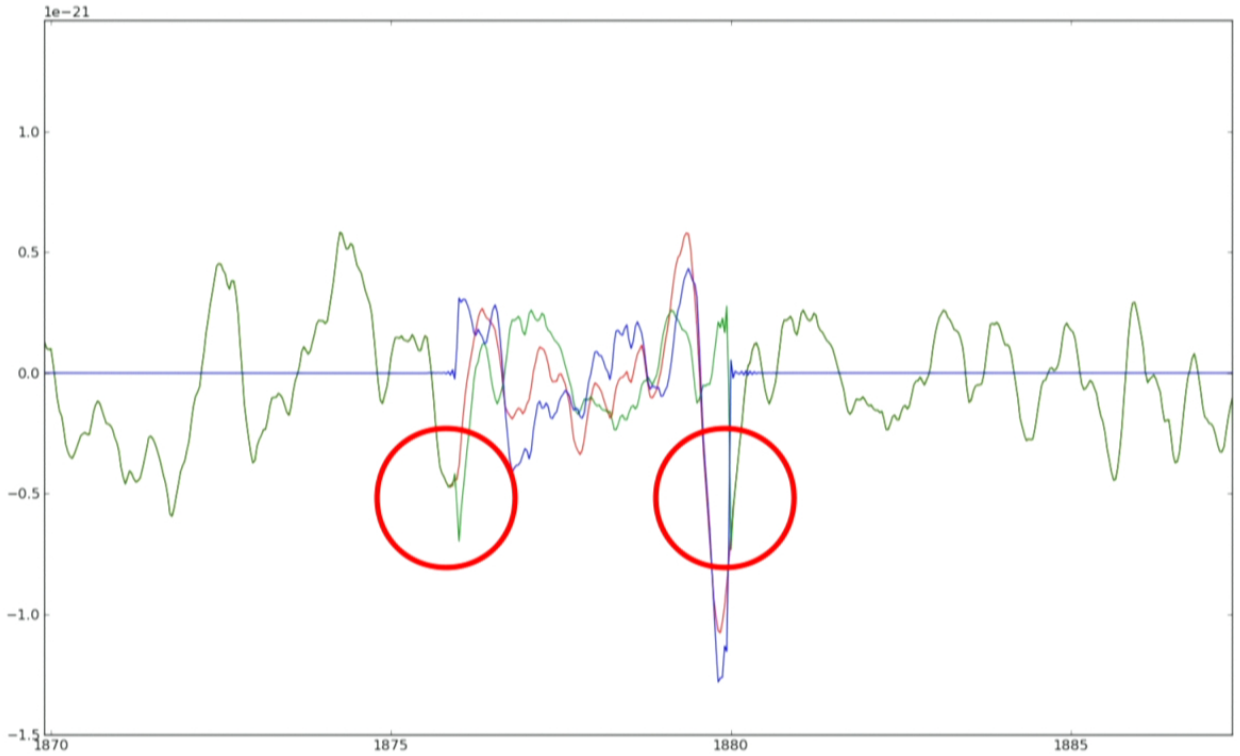
a) There was an error creating the authoritative set. Check back often for "fixed" data!
b) Somewhere between the authority and you the data has been corrupted.

3) When maintaining your own data be an authoritative source for yourself and your colleagues.

a) Put your data in version control. (`svn, cvs, git`)
b) Provide checksums or other verification.



I AM NOT MAKING THIS STUFF UP

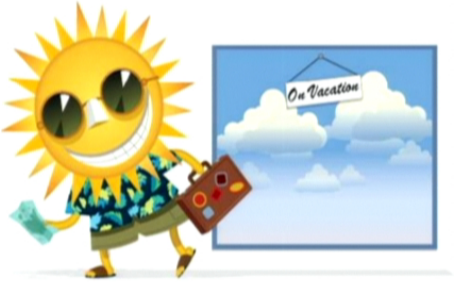


Know the scale

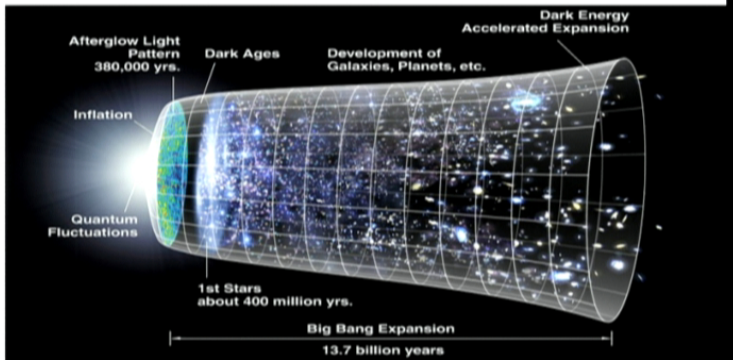
* Can I get a cup of coffee while this finishes?
* Can I go to lunch while this finishes?
* Should I take a vacation while this finishes?
* Is this best left for future generations or alien civilizations?

Odds are that if it is any but the first one you need parallelization.

* Will this data fit on a thumb drive
* Will it fit on my hard drive?
* Will I need to ask Troy for more hard drives?



Storage of data is not as much of an issue as efficient access.



Know the scale

Often times parallelization can be “embarrassingly” simple.

Computers typically have 2-8 processing cores. Parallelization might be as simple as running 8 versions of your analysis at the same time. High level languages (like Python) support simple multi threading that can make a job go from lunch time scale to coffee time scale.

(HPC facilities often have 32 and 64 core machines = vacation time → lunch time scales)

Consider the following options only after you are sure that you cannot gain from very simple parallelization techniques:

* Open MP
* direct use of POSIX thread libraries
* GPU acceleration (Tempting, I know! But a lot of work and still not extremely portable)

Know the scale

Often times parallelization can be “embarrassingly” simple.

Computers typically have 2-8 processing cores. Parallelization might be as simple as running 8 versions of your analysis at the same time. High level languages (like Python) support simple multi threading that can make a job go from lunch time scale to coffee time scale.

(HPC facilities often have 32 and 64 core machines = vacation time → lunch time scales)

Consider the following options only after you are sure that you cannot gain from very simple parallelization techniques:

* Open MP
* direct use of POSIX thread libraries
* GPU acceleration (Tempting, I know! But a lot of work and still not extremely portable)

Know the scale

Don't let data access slow you down.

Big data sets cannot simply be read it into memory.

It is easier to learn about nifty file formats and tools than to try to invent it yourself.

* If your data is best described as large arrays...

Consider HDF5 to store the data. Many big science collaborations already use this format. It is possible to read and write HDF5 files in C, Python, Matlab, …

* If your data is table like in nature...

Avoid just using the old standard of space delimited ASCII files! It doesn't scale and it will cause you nothing but pain in the long run. Relational databases have existed since the 1970s to solve the problem of storing, querying and manipulating relational data. It doesn't have to be hard. Check out sqlite. It is a simple file based relational database with support in all sorts of languages.

Know the software

There are two extremes where folks tend to go wrong in choosing software for a project.

1. “I can do this myself, I don't need your stinking software.”

You are smart. You probably can do it yourself. But if you have never tried you will grossly underestimate the time it takes to produce an acceptably bug free software stack from scratch. Sometimes this is necessary. And sometimes it is good to learn (or teach) by doing something yourself. But... it is rarely a good idea for critical projects unless you are sure it is the only option.

2. “This does what I want. Seems simple. I will just use it.”

Often people choose what is easy for them right now and just move on. Consider if your choice is portable. Does it have licensing restrictions? Will it allow you to scale up to a larger more complete version of the analysis you are trying to do (maybe HPC)?

There are a lot of open source scientific computing resources that are very friendly to use on many different types of machines / operating systems. HPC centers don't have to worry about licenses. Consider:

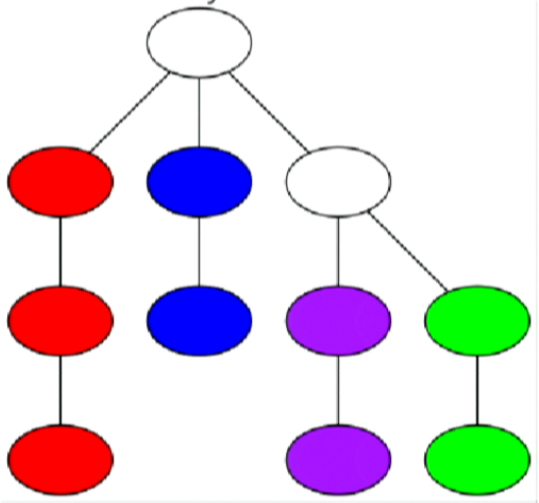
Python (scipy and numpy modules)
Octave (Matlab like syntax without the license)
GSL (C library for numerical work)
FFTW (Open FFT platform, and the best as far as I know. Really, don't use anything else)

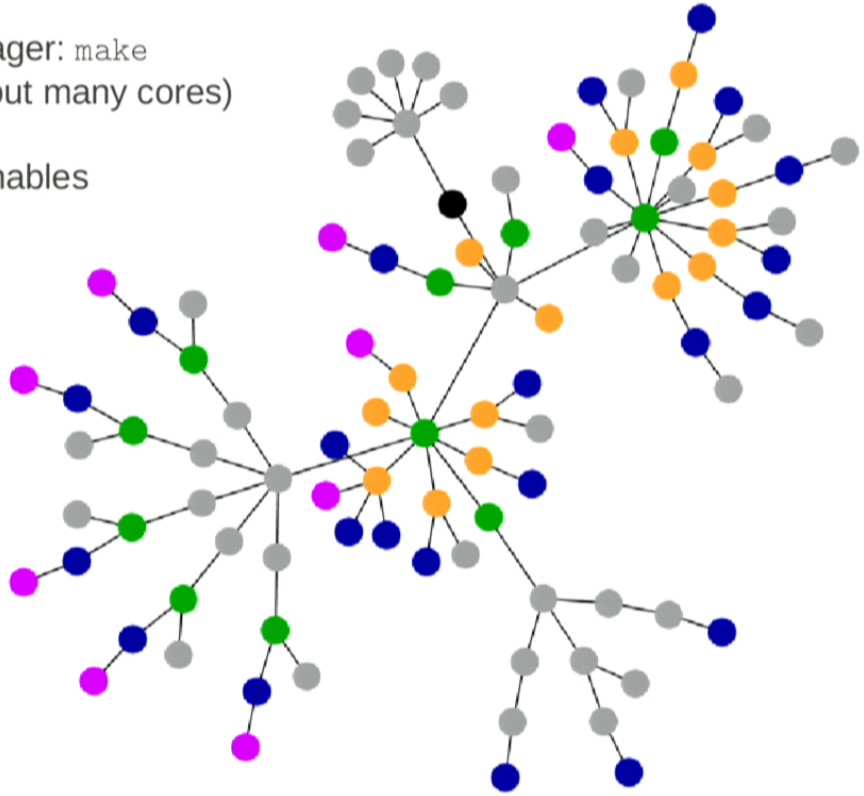
Be able to reproduce the results

Analyses often evolve organically. The steps typically do not commute even if you think they do. If you finished the analysis once it means that you should be able to express it as a directed acyclic graph (DAG). Great! Now implement a workflow that repeats it start to finish. Better yet decide to do this from the beginning.

Simple, often overlooked workflow manager: `make`
Great for a single machine (with or without many cores)

HPC compatible batch processor that enables the implementation of DAGS: `condor`
Great for many machines

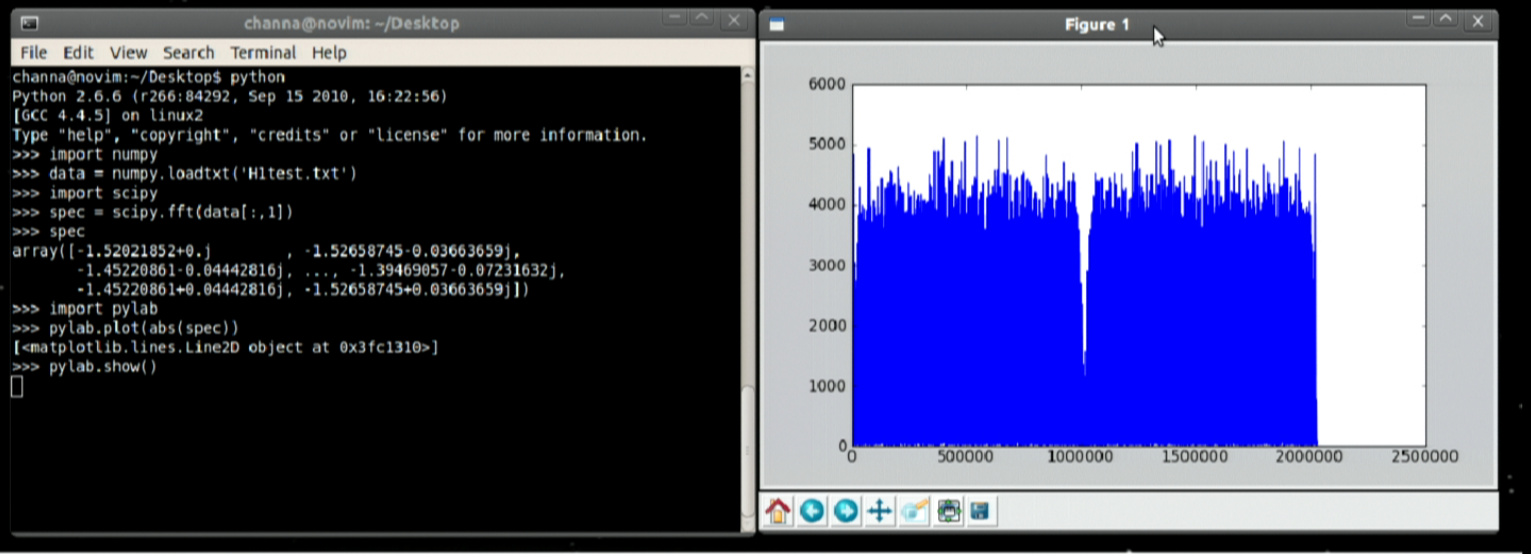




Be able to reproduce the results

Avoid interactive sessions!

Everything you type into a terminal is software. Interactive sessions can often be replaced with scripts. You will want a recorded history of what you do and a push button way to repeat it. Write scripts and put them into version control.



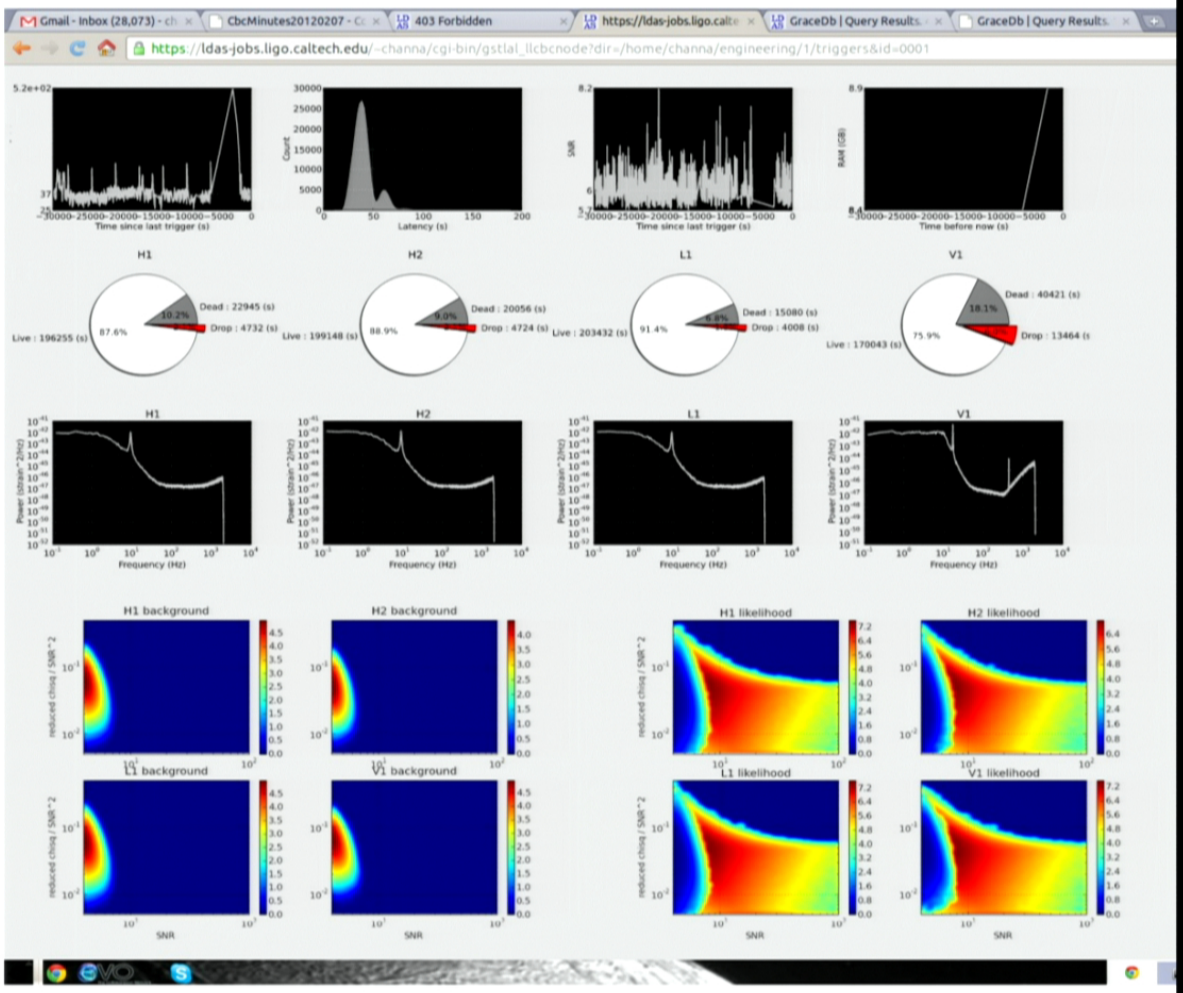
Shameless plug for my research (with Kipp Cannon (CITA) Drew Keppel (AEI))

Goal:

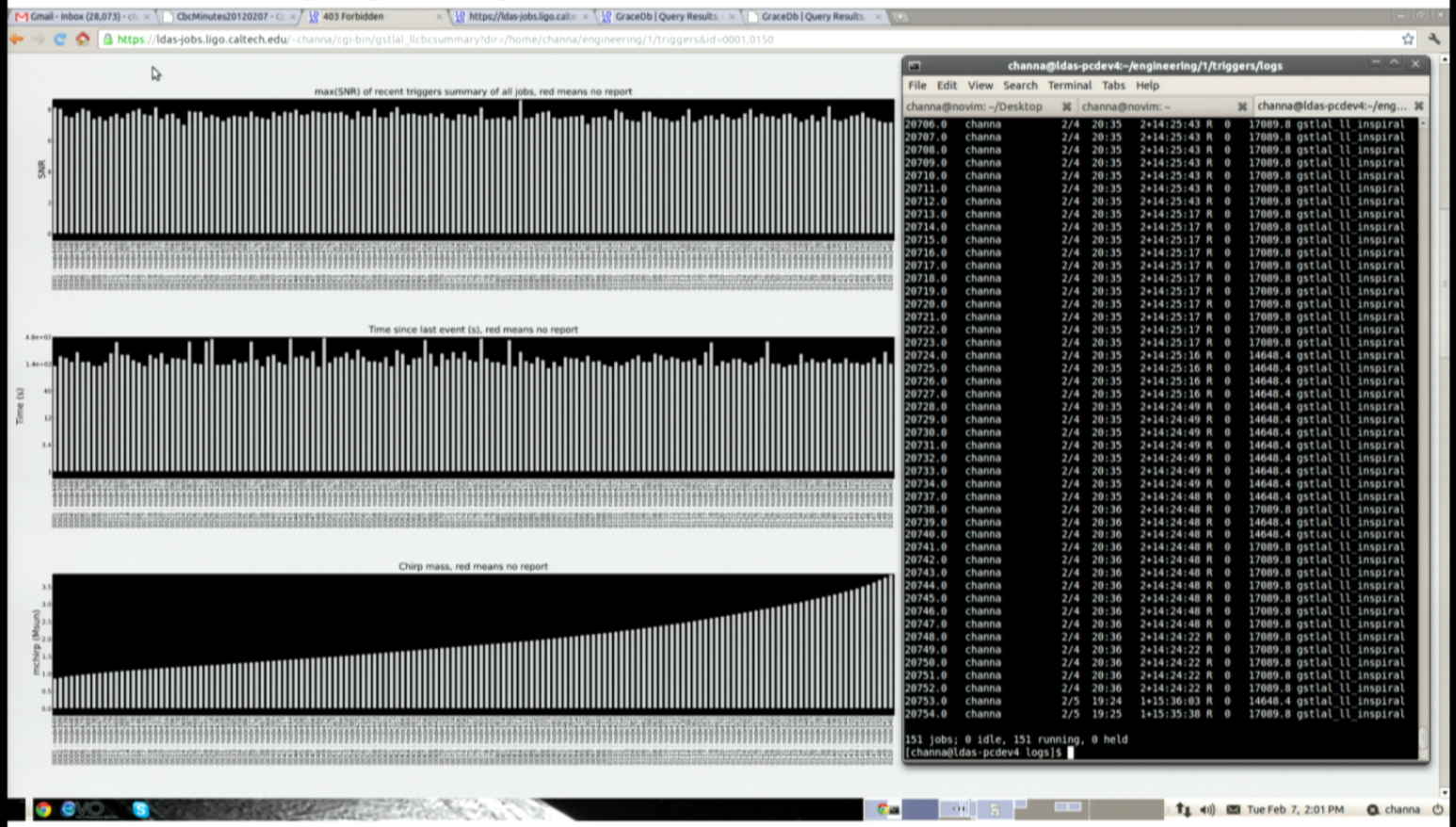
Realtime detection and localization of gravitational waves from merging neutron stars from a global network of GW detectors.

POSIX threaded programing on multiple cores.

```
gsl
FFTW
gstreamer
scipy / numpy
sqlite
```

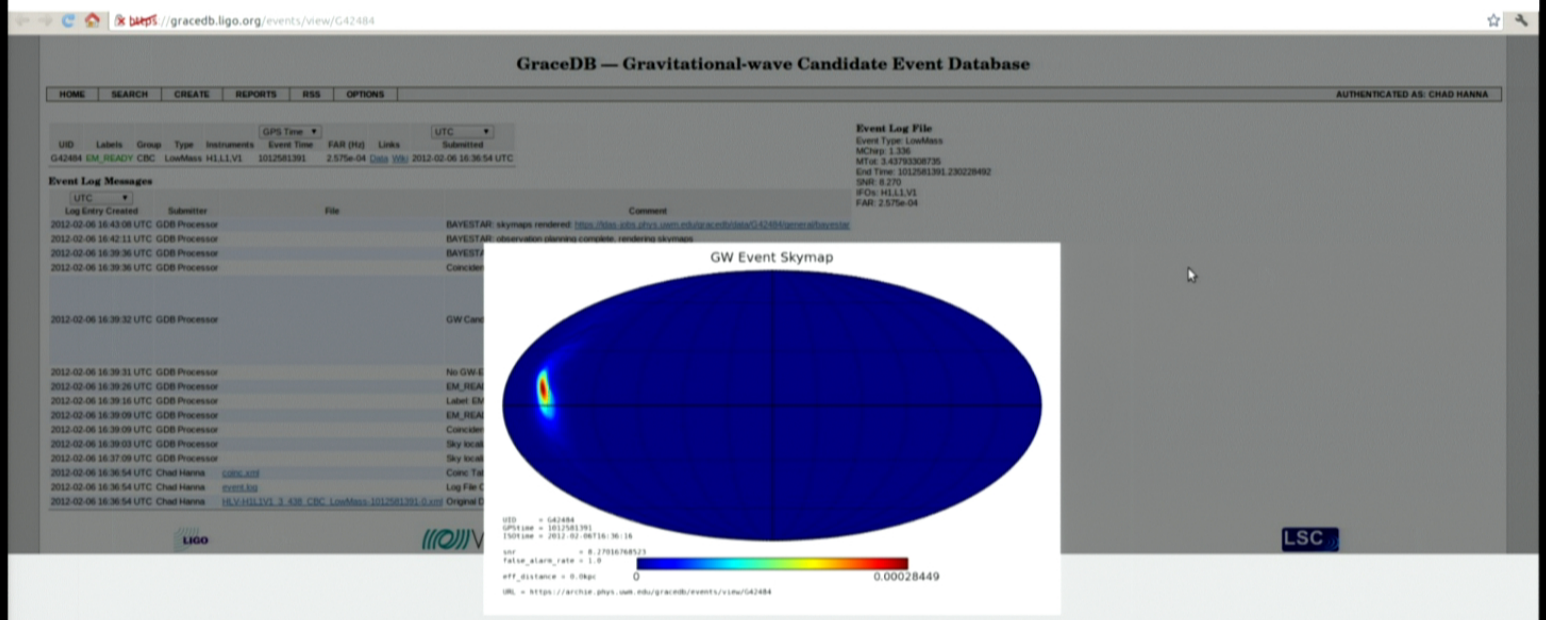


Shameless plug for my research



`condor` manages the embarrassingly parallel aspects of splitting up the parameter space. Needs 150 8-core machines. Configuration is handled by `make`

Shameless plug for my research



Identify candidate GW events within ~30 seconds. Plan immediate, coordinated electromagnetic follow up campaign.

[The current engineering run uses simulated data but is otherwise the real deal. If this really was a gravitational wave candidate event I wouldn't be allowed to show you anyway]