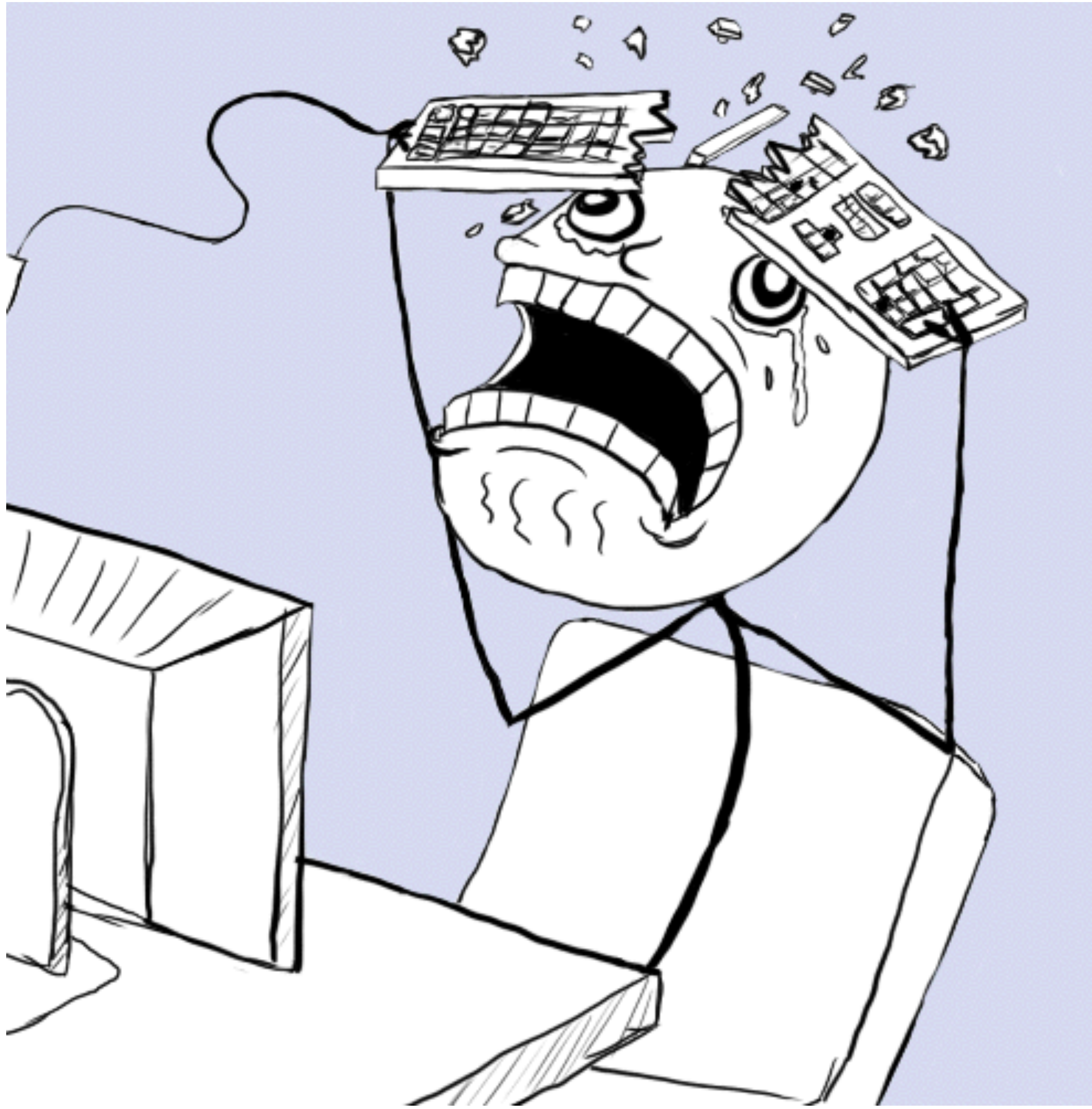
D is for Science

John Colvin

What is scientific programming?

You want to do science, using a computer



but the existing software isn't up to the task

Scientific Programming

- Simulations
- Data Analysis
- Visualisations
- Control

Simulation

Start with a system and some rules. Numerically, calculate the outcome. E.g.:

- Climate models
- Aerofoil models
- Epidemics

Data Analysis

You've got some data, work out who/what you're looking at, E.g.

- Calculating summary statistics of a social network like the mean degree.

- Identifying different types of oscillations in/of coronal loops.

- Does that star have a planet?

- How much of that weapons grade uranium is now actually lead?

Scales

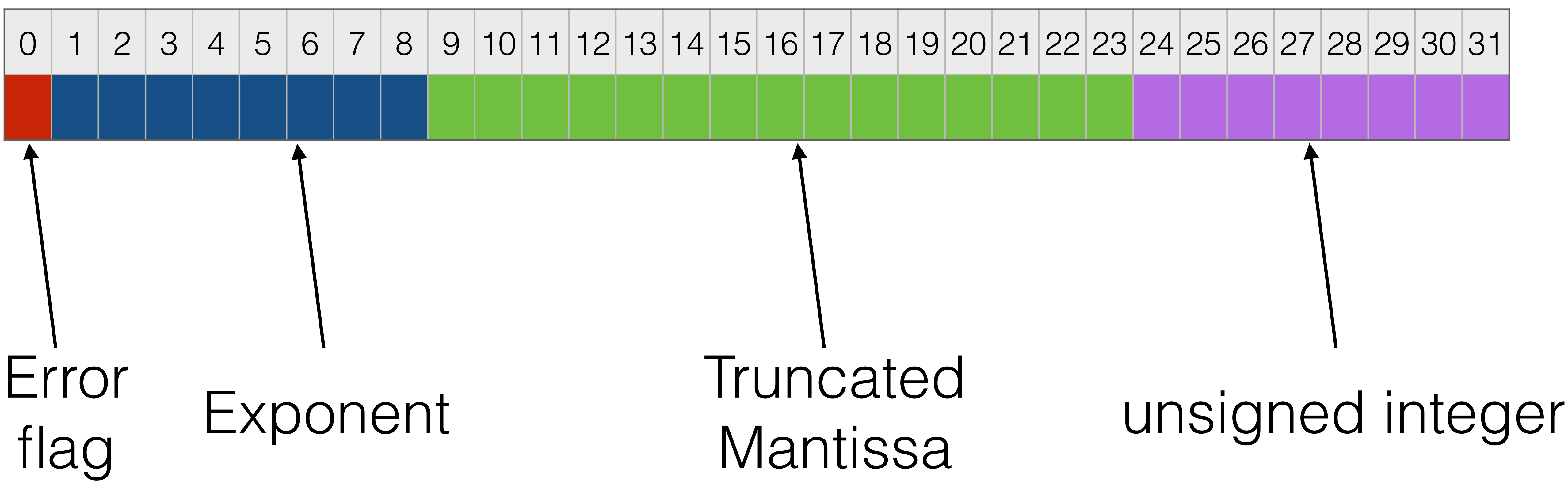
Every Scale

- DIY instruments with microcontrollers.
- Data cleaning scripts on laptops
- >10^6 core simulations running for weeks or even months.

Example:

"I just want to get some summary stats!"

A weird data layout



Want to calculate the mean and standard deviation of the floating point number for each value of the integer, with and without erroneous data removed

Large scale simulations

The status quo

- Fortran is still well used and well liked.
- C is also common
- C++ is used by some high-profile projects (e.g. OpenFOAM), but isn't widely liked.
- Small teams of developers with some peripheral scientists, at best.

Why Fortran?

- It has proper multidimensional arrays
- You can do most of what you need without pointers
- Hard to mess up due to restrictive semantics
- Easy for optimisers
- Totally familiar to 2 generations worth of scientific programmers

Large simulations

Language requirements

- Ease of development for people aren't that hugely in to programming.
- No compromises performance
- MPI support. Any competing system needs to have equally good support from high performance network hardware/drivers e.g. infiniband
- Great linux support

Simulations in D

Multidimensional arrays	✔
No compromises performance	✔
MPI support	✘
Linux support	✔
Foot-shooting protection	✔
Familiarity	✘
Proven track record	✘
Flexibility	✔

Data Analysis

The status quo

- Matlab, Python, IDL, shell scripts
- Dropping down to C / Fortran for when performance becomes a problem
- Often done by people who aren’t really interested in programming, even if it’s 90% of their job in practice
- Mostly individuals working independently, or large infrastructure projects set up for specific experiments.

Why Python/Matlab?

- Get work done, fast.
- Libraries
- No segfaults
- REPLs/notebooks (for everything, or sometimes just as command and control for scripts). Persistent state is great.
- Fast enough most of the time
- Familiarity and existing tools. Huge piles of specialised little functions that whole workflows are wedded to
- Libraries. Libraries. Libraries.

Data Analysis

Language requirements

- Ease of development
- Fast enough. People are used to slow
- OS X, Linux and Windows support all necessary, although Windows less so
- Visualisation
- Quick Edit-Run cycle and a stateful UI. No one wants to wait 60s compiling code and reloading datasets just to slightly change a line weight on a plot!

Data Analysis in D

Multidimensional arrays	✗
Good enough performance	✔
Linux support	✔
OS X support	✗
REPL / notebook	✗
Familiarity	✗
Interoperability	✔
Flexibility	✔

Numerical precision

- x87 is really slow for a variety of reasons
- Assuming you do actually need 80 bit floats:
- Either you know what you are doing and need choice or
- You don’t know what you’re doing, a few extra bits isn’t going to solve your numerical problems.
- A lot of high performance work is not hyper-precision-sensitive, by design.

Aside: GPGPU

- There’s a lot of scientific calculation that is implicitly parallel and that works well on GPUs

- On the other hand, despite improvements in libraries and tooling, the barrier to entry remains high enough to put off most

- I have my own attempt to give D great GPGPU support, but it is still work in progress

D-OpenCL architecture

- Layer 1: Take the OpenCL C API and make it strictly typed
- Layer 2: Take this strictly typed layer and layer a more D-appropriate API on top
- Layer 3: Go to town. High level abstractions to enable people to easily execute code on co-processors. Based on Layer 2.

DlangScience

- https://github.com/DlangScience
- A focal point for both developers and users
- Vetted, Tested and eventually, hopefully, comprehensive.
- Currently just me :-(

Fin

Questions?