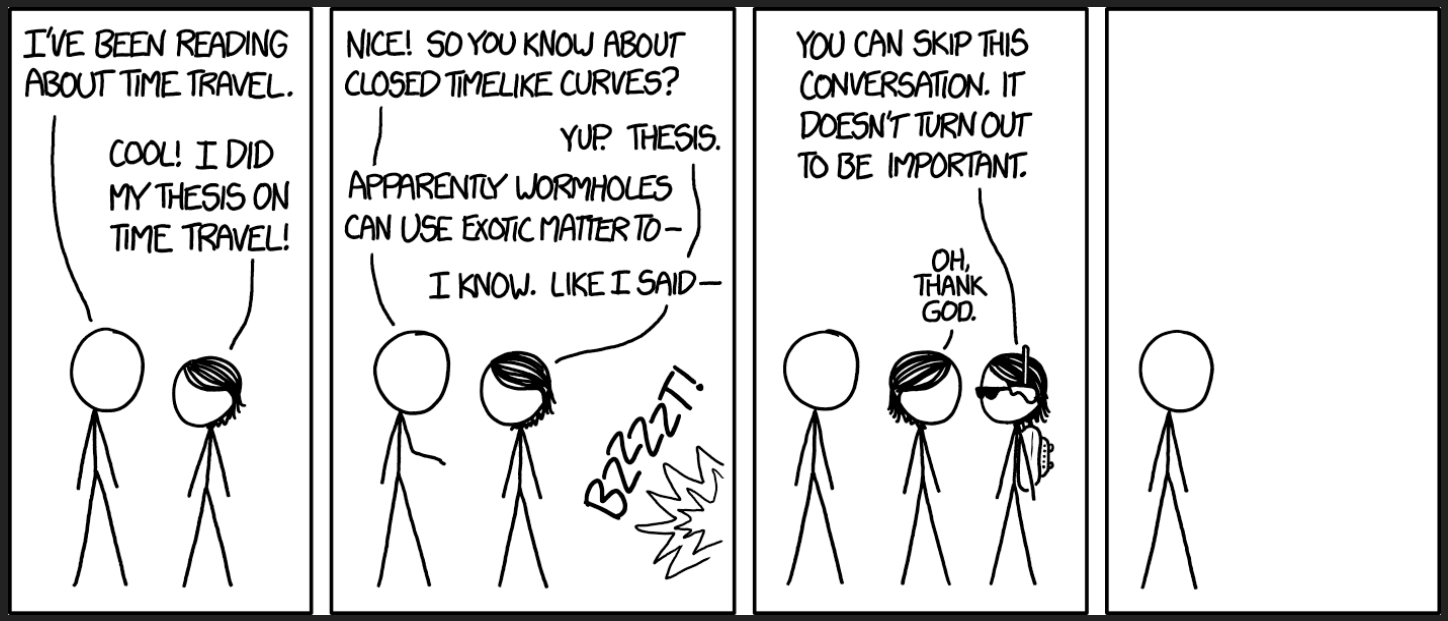


HEP ANALYSIS ECOSYSTEM WORKSHOP

VISION '25

Gerhard Raven / VU Amsterdam + Nikhef / LHCb

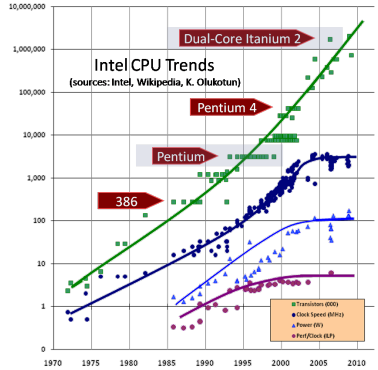
DISCLAIMER...

- Quote Andrei Alexandrescu – Declarative Control Flow [youtube]
  - *You* are (trying to shape) shaping how HEP analysis will look like!
  - Responsibility: What if I make a really bad suggestion and convince you?

- So: no answers / dictate – more of a wishlist / perspective

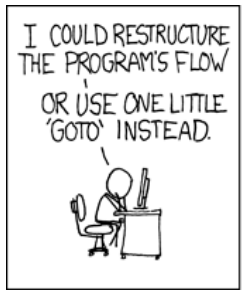
WE ARE STILL DOING THE SAME THING (!?)

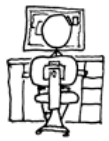
- Take a LEP-era physicist
  - would be comfortable with an LHC analysis – after being amazed about the growth of data, computing & complexity
- The growth in computing since the early 90-ies to late 00-ies has allowed us to be ~ conservative

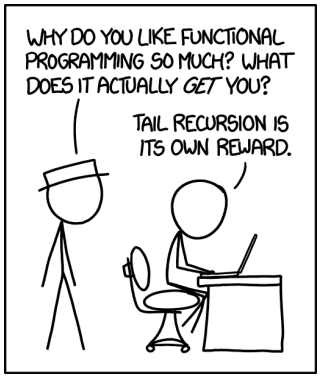


EVOLVING CODE IS TAKING AWAY — AND INTRODUCE ABSTRACTIONS

- Structured programming – Dijkstra: GOTO considered harmful
  - use loop constructs (for, while) instead
- Procedural programming – modularization
  - (local) scope
- Object-Oriented programming – dependency inversion
  - takes away 'void *', use VTBL instead – allows to call lower-level code 'not yet written'
  - hide state
- Functional programming – takes away (mutable) state
  - powerful type systems, referential transparency
- Declarative programming – takes away control flow







PROGRAMMING PARADIGMS

- Why does HEP love 'imperative programming'?
  - (the illusion of) control!
  - not a black box (erhm.. really?)
- The problem with 'imperative' code' : it over-specifies!

- What is the alternative?
  - 'declarative' / 'functional'
  - express the logic *without* specifying the control flow
    - eg. Makefile / SQL / Wolfram Language / C++ destructors

IT IS TIME TO BE MORE ABSTRACT, AND LET GO OF (BORING) DETAILS

High-level and speed are not antithetical

diana hep

Code like

```
bestmuon =
  muons.filter(m => m.iso > 10)
       .maxBy(m => m.pt)
```

does *not* need to create function objects or muon objects at runtime!

It need not be "taken literally."

Another possible execution plan:

1. Start with all `muon.iso` values in one array, all `muon.pt` values in another array, and a "repetition level" to specify where events begin and end.
2. Apply the contents of the filter function to make a mask.
3. Use the mask and repetition level to compact the `muon.pt` into zero or one results per event.

Jim Pivarski

Improving on current interfaces

```
TTreeReader data(tree);
TTreeReaderValue<A> x(data, "x");
TTreeReaderValue<B> y(data, "y");
TTreeReaderValue<C> z(data, "z");

while (reader.Next()) {
   if (IsGoodEvent(x, y, z))
      DoStuff(x, y, z);
}
```

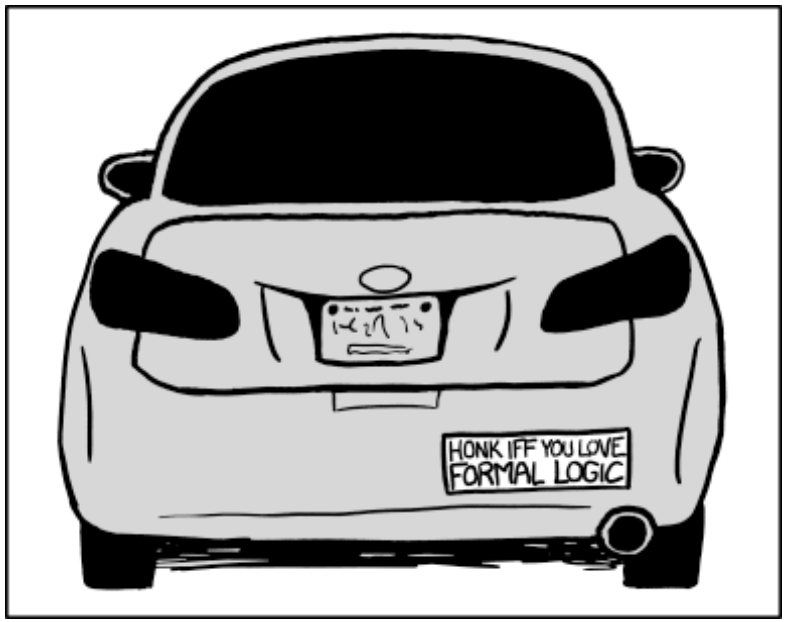
```
ROOT::EnableImplicitMT();
TDataFrame data(tree, {"x","y","z"});


data.Filter(IsGoodEvent)
    .Foreach(DoStuff);
```

- users have full control over the event-loop
- ✔ needs some boilerplate
- ✔ running the event-loop in parallel is not trivial
- ✔ users implement trivial operations again and again

Danillo Piparo

FRONTEND VS. BACKEND

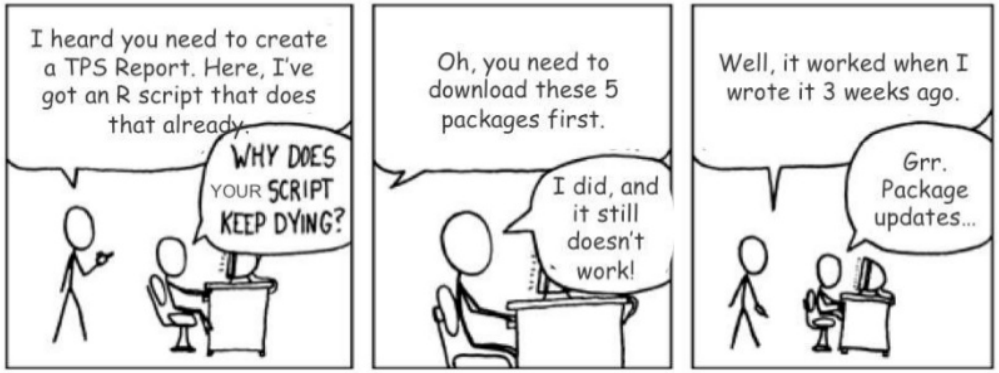
- Separate ‘physics’ configuration / logic (‘what’) / logic from the ‘compute’ implementation (‘how’)
  - eg. RooFit provides ‘declarative’ hooks for evaluation & normalization – and does constant folding, caching, hybrid numerical/analytic integration, interpolation, …
- Must allow backends the freedom to evolve!



- black box risk: (further) split between ‘analysis’ and ‘computing’ knowledge / people…

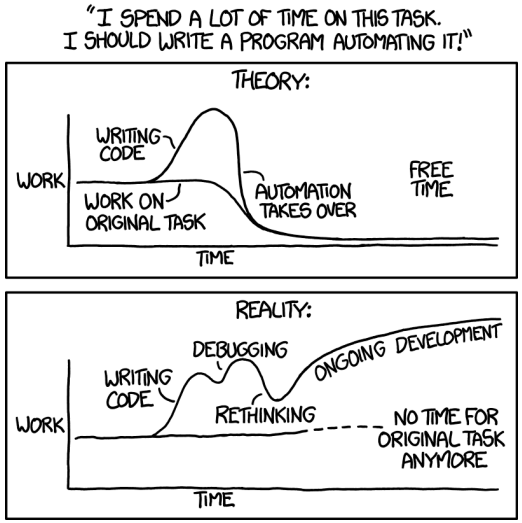
ESTABLISH PROVENANCE / VOCABULARY

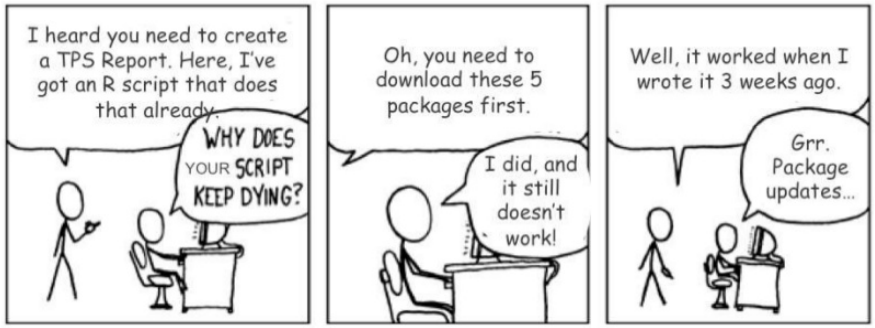
- one of my greatest ()#$*) with NTuples / TTrees (key-value stores),
  - How do I know whether those keys really correspond to the right observables?
- Need first-class provenance!
  - Links back from 'keys' to the code that produced the 'values'
  - git-like versioning for data
    - redo an analysis on the 'previous version' of the data (can we afford to do that? storage is already a problem!)
  - dependency tracking
    - updates to observables when eg. calibrations are updated
  - ability to add (forgotten) observables without redo-ing everything…



PROVENANCE REQUIRES INTEGRATED/AUTOMATED WORKFLOW

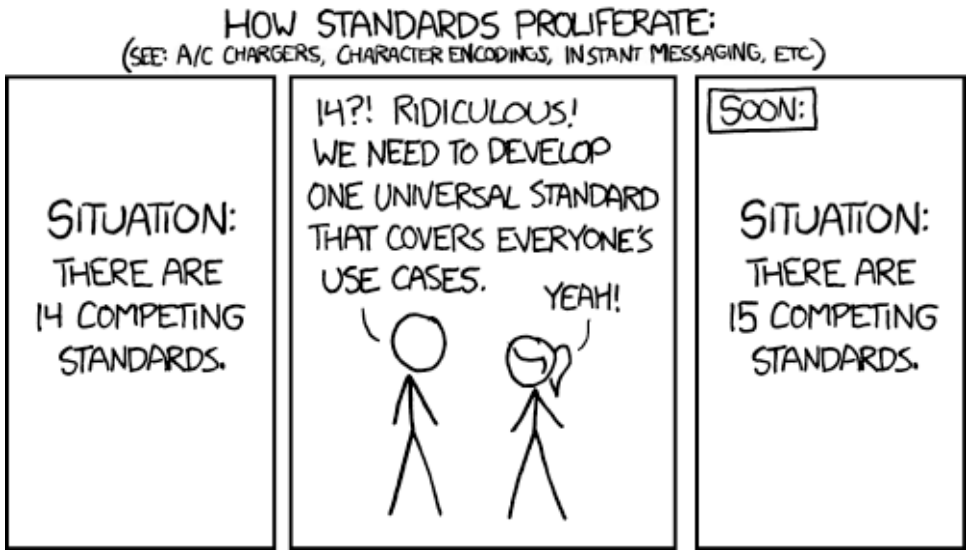
- producing large scale NTuples is a)#($*@)_
  - trigger → MDF (bytestream) → reco → DST → stripping (skimming) → uDST → NTPL "A" → NTPL "B" → RooFit
- Why hasn't uDST / AOD taken over?
  - Need to link with 'event model'
  - Not invented here syndrome
  - Toolkits vs. frameworks / straightjackets
- Automated pipelines & continuous integration – first steps towards reproducible analysis
  - Universities / Funding Agencies plan audits!





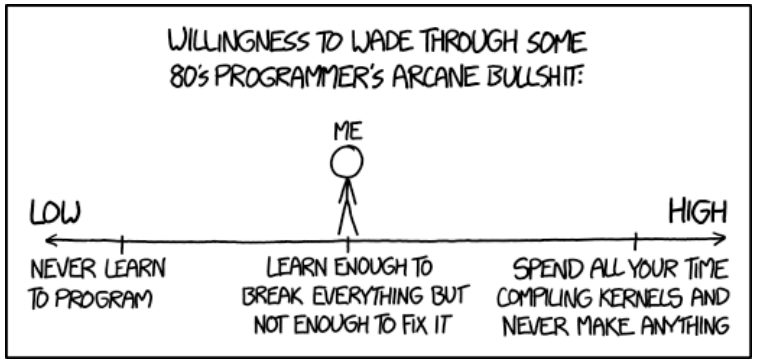
HOW TO BENEFIT FROM THE WORLD OUT THERE

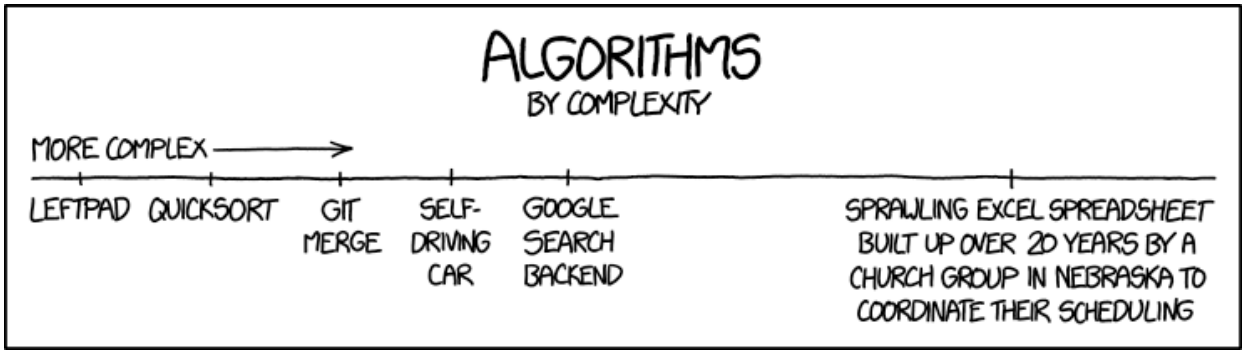
- Industry has grown a LOT
- Google/Facebook/Amazon/Microsoft/Apple employ a lot of very clever people
- Doing your own bare-bones 'GPU' framework will not keep up
- Better to re-use/interop with eg. TensorFlow / SPARK (or lower level like Thrust) and focus on how to leverage those
- But what if you pick the "wrong" standard, and it dead-ends?
  - Major reason why in the past we 'did it ourselves'....
  - Contribute back (eg. to standards)



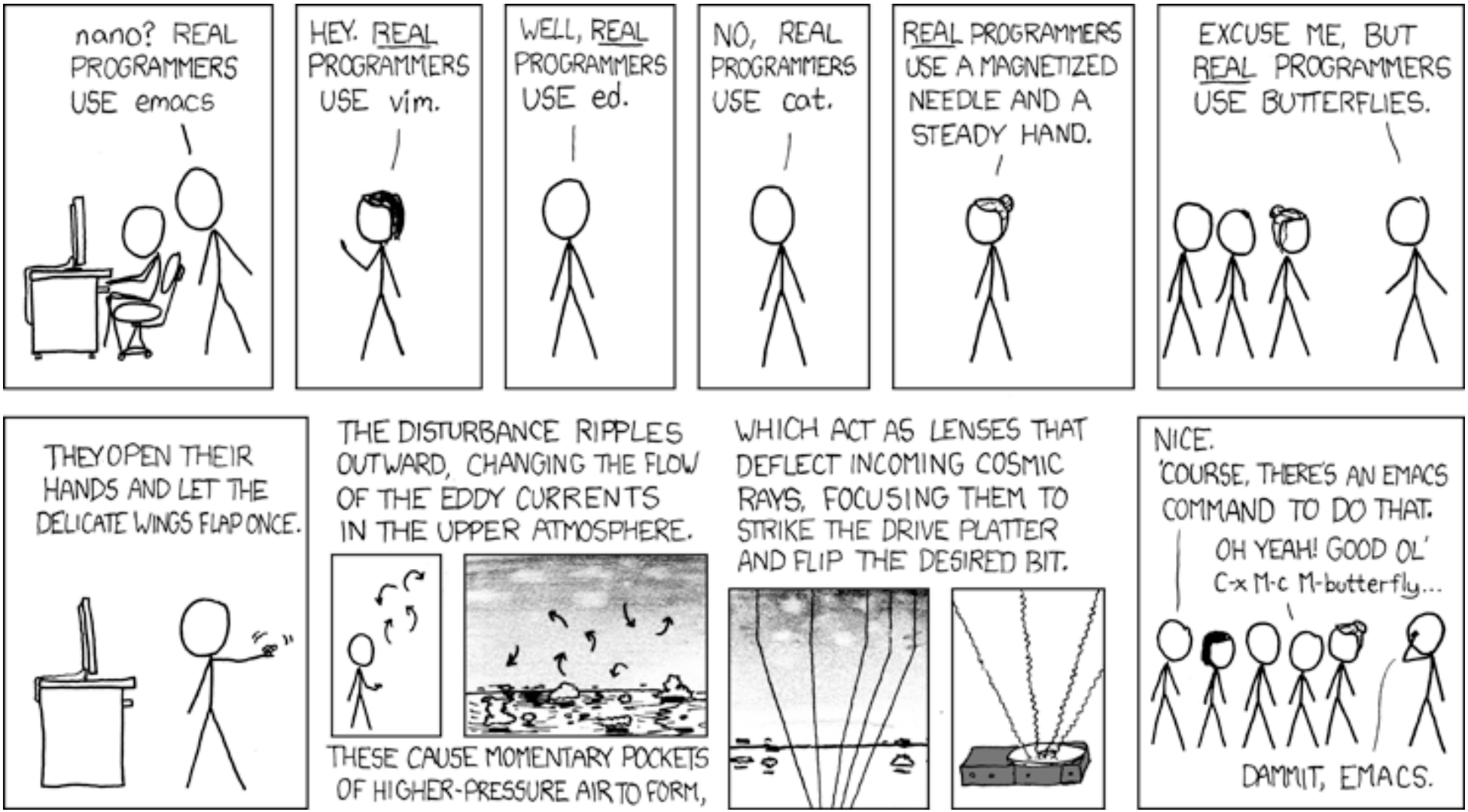
PERCEIVED/UNNECESSARY COMPLEXITY

- “People can not contribute because computing nowadays is too complex”





- Need several full analysis chains which demonstrate “the new way” is better / easier / more performant / ....


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

Vision::~Vision()