Rosetta3.x as an library/API

With examples from an E2/E3 system

Suggested reading for the tutorial:
http://www.rosettacommons.org/manuals/rosetta3_user_guide/adv_overview.html
http://www.rosettacommons.org/manuals/rosetta3_user_guide/write_protocols.html

Steven Lewis
Kuhlman lab

Outline

- What is Mini / Rosetta3.x?
- Why should you learn to use it as an API?
  - Example system: E2/E3 tail binding
- What's in the code?
  - Developers' conventions
  - Major classes

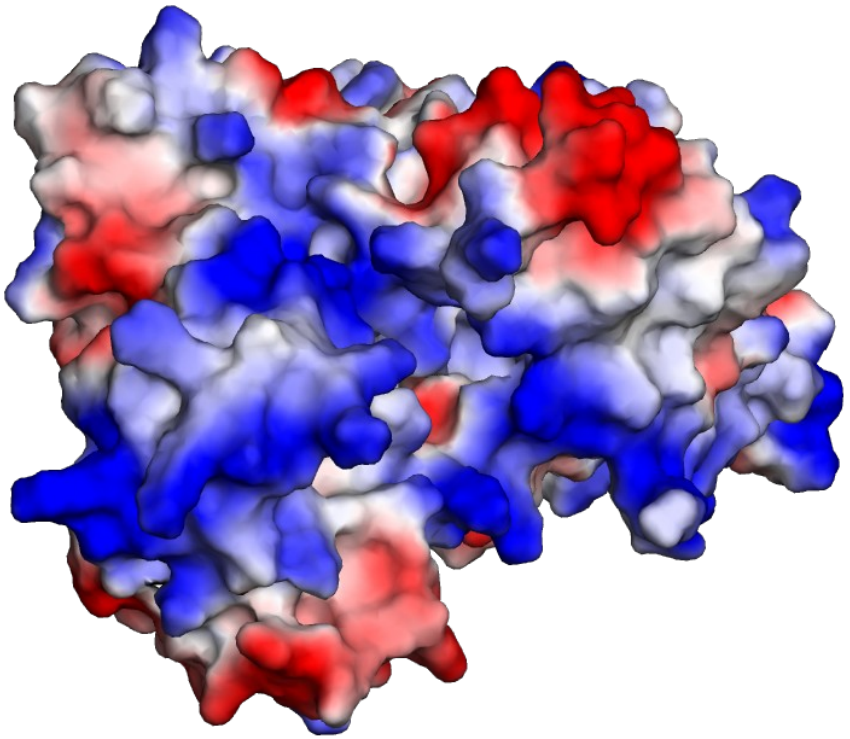
What is Mini / Rosetta3.x?

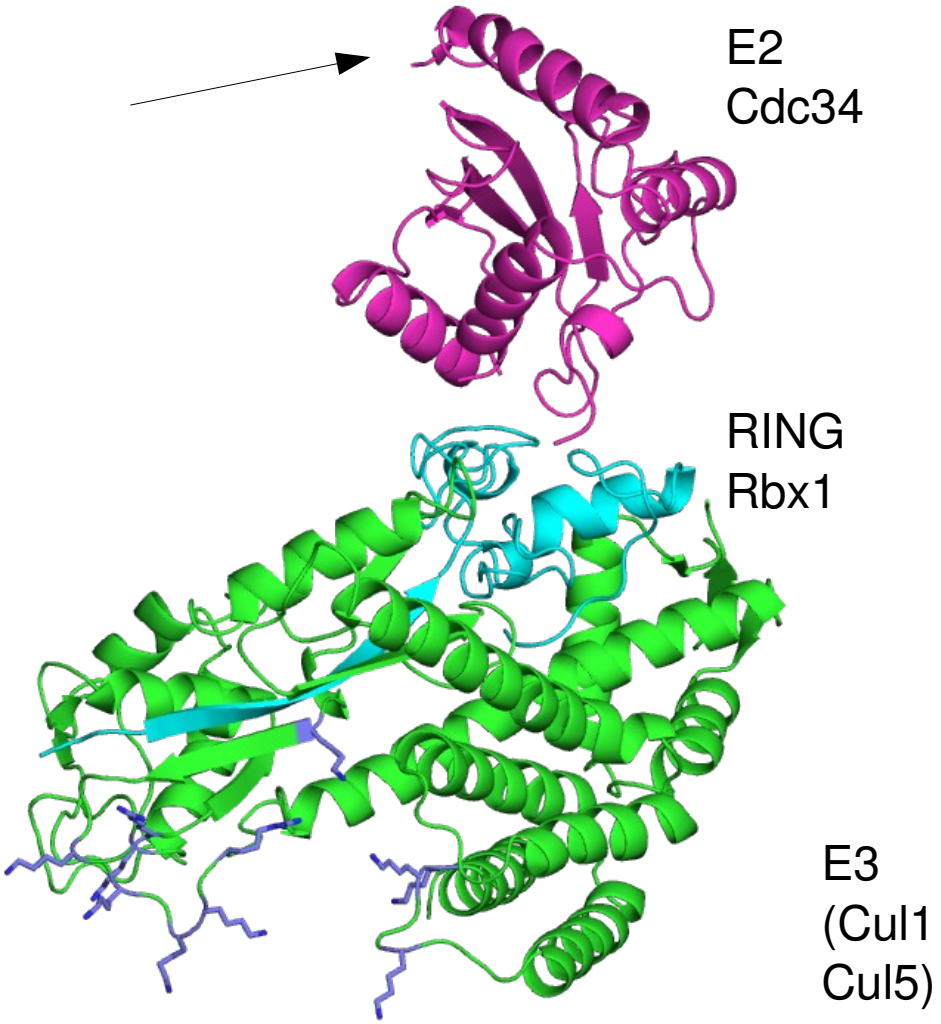
- Rosetta 3.0 was released 2/27/09 (3.1 is soon)
- The developers call it “Mini” or “Minirosetta”
- Represents a near-total reboot of codebase from Rosetta++ (2.x)
  - Organized into classes!
  - No more monolithic executable!
  - Allows use of the low-level code as a library/API
    - Easy to write new executables

So, we needed a new protocol...

- Collaborators had a question:
  - Long, flexible, acidic tail
  - Big, obvious basic patch on a binding partner
  - Mutational data implying the two go together
  - Some question of whether the tail is long enough to reach the proposed binding site
  - What might it look like...?

The patch; the tail; the distance





VKVPTTLAEYCVKTKAPAPDEGSDLFYD

DYYEDGEVEEEADSCFGDDEDDSGTEES

How to model this?

- Existing protocols not meant for purpose
  - COULD handle it, with some re-writing

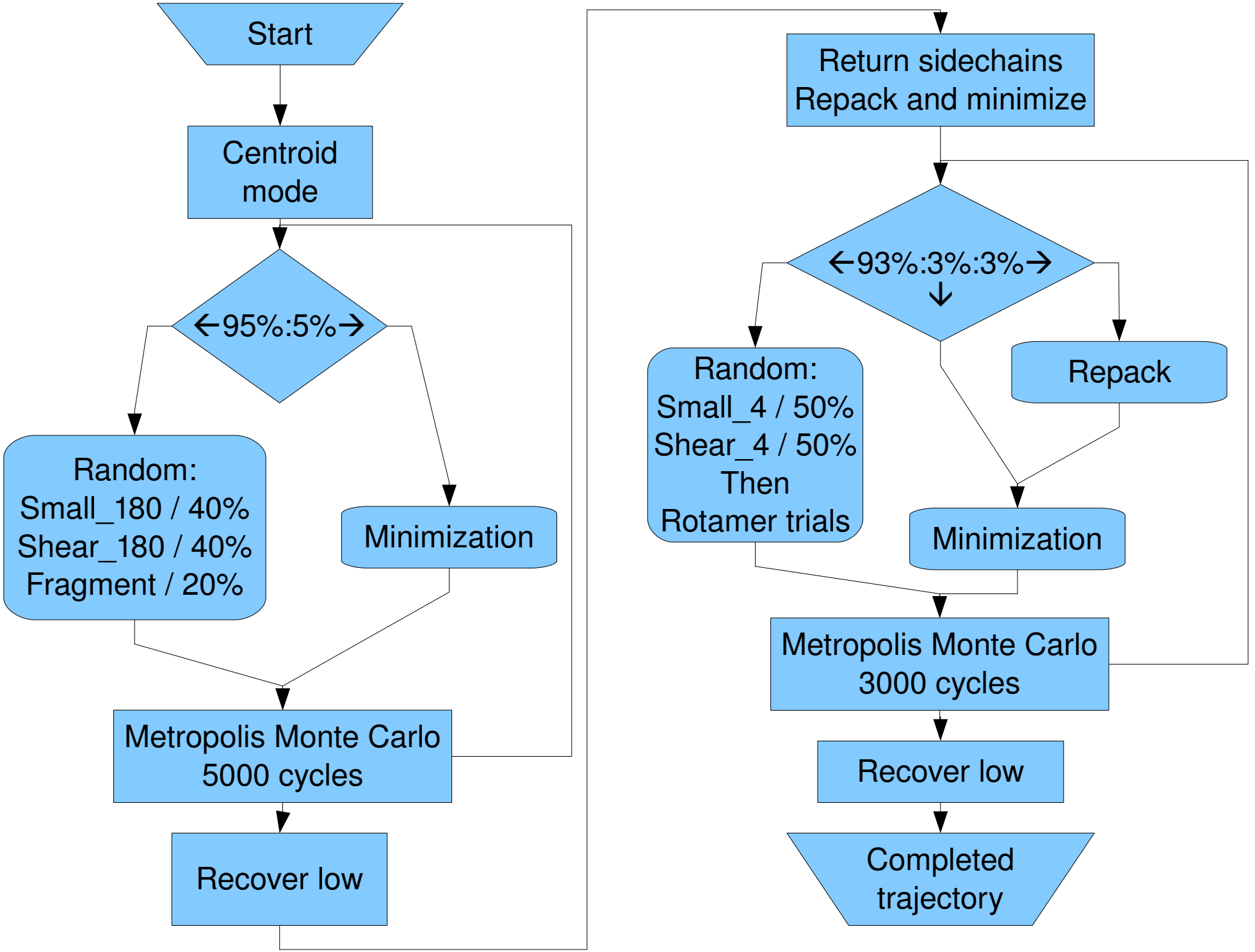
- Write new executable
  - Easy, because of modular library nature of mini
  - Simpler than special cases within major protocols

From zero to first results? DAYS.

- Planned:
  - Degrees of freedom to vary
  - How to vary them
- Protocol structure
  - Monte Carlo
  - How often to repack?
  - All fullatom, or some centroid?

- Wrote code
- Tested with tiny experiments

- Your experience will vary
- Bug hunting and result-based refinement take time

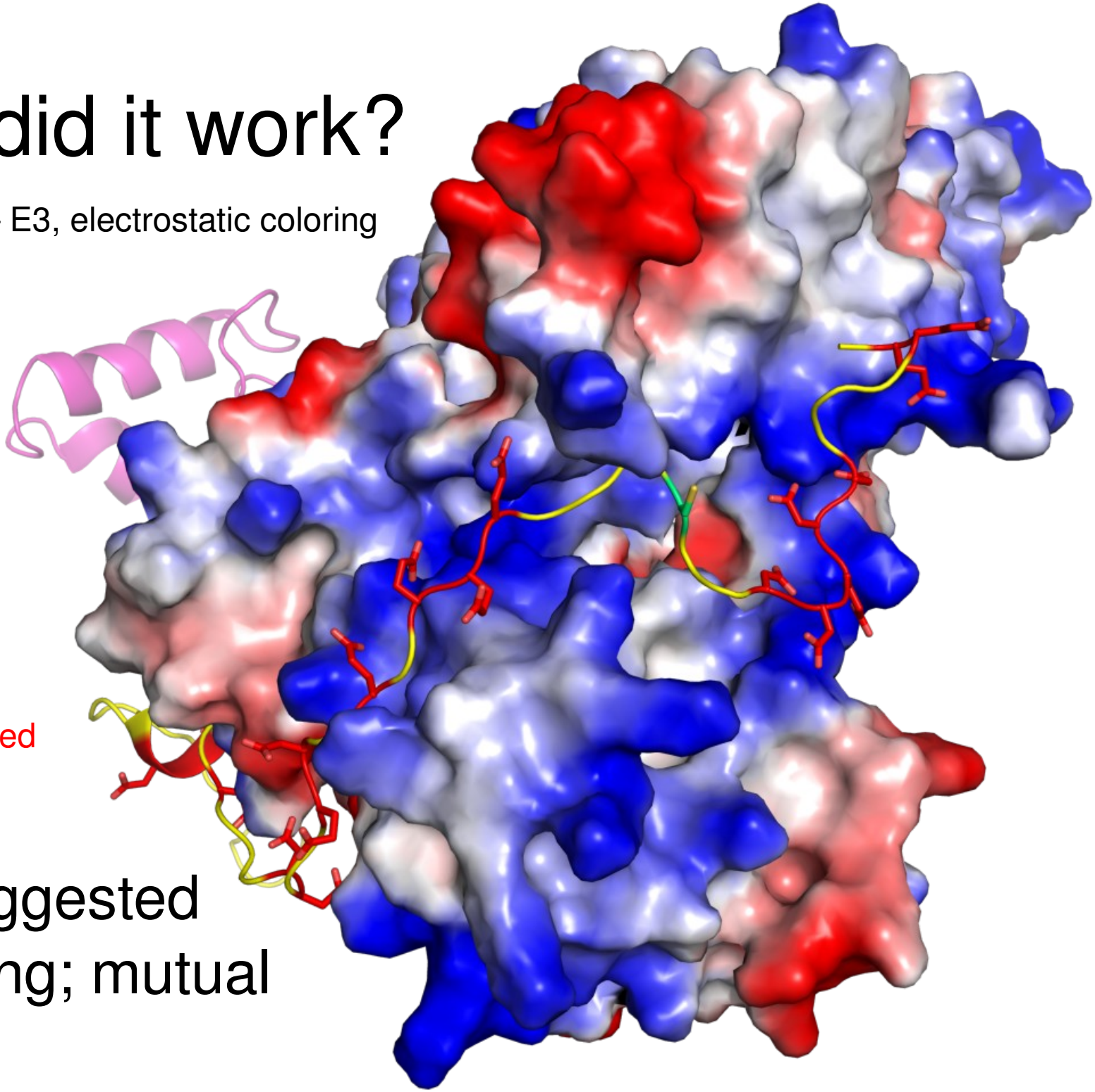

Start
Centroid mode
←95%:5%→
Random:
Small_180 / 40%
Shear_180 / 40%
Fragment / 20%
Minimization
Metropolis Monte Carlo
5000 cycles
Recover low
Return sidechains
Repack and minimize
←93%:3%:3%→
↓
Random:
Small_4 / 50%
Shear_4 / 50%
Then
Rotamer trials
Repack
Minimization
Metropolis Monte Carlo
3000 cycles
Recover low
Completed
trajectory

Soooo...did it work?

Surface – E3, electrostatic coloring

E2 core in fuchsia

E2 tail in yellow
acidic residues in red
cysteine in green



- Model suggested crosslinking; mutual support

Acknowledgements

- E2/E3 project
- Ray Deshaies
- Gary Kleiger
- Anjanabha Saha
- Brian Kuhlman
- Kuhlman lab

- Rosetta3.x / Mini
- All of the developers

- In review

Questions on science portion before we head into code tutorial?

Outline

- What is Mini / Rosetta3.x?
- Why should you learn to use it as an API?
  - Example system: E2/E3 tail binding
- What's in the code?
  - Developers' conventions
  - Major classes
- Interrupt with code questions at any time – this is a tutorial not just a talk!

Tutorial

- I assume:
  - You know some C++
  - You know what classes are, and inheritance
  - You've glanced at the codebase at least once
- We'll cover:
  - Philosophy
  - Structure of mini
  - Important conventions
  - Major classes
  - Protocol writing 101
    - Example in manual!
    - Code for 1st half is FloppyTail in 3.1

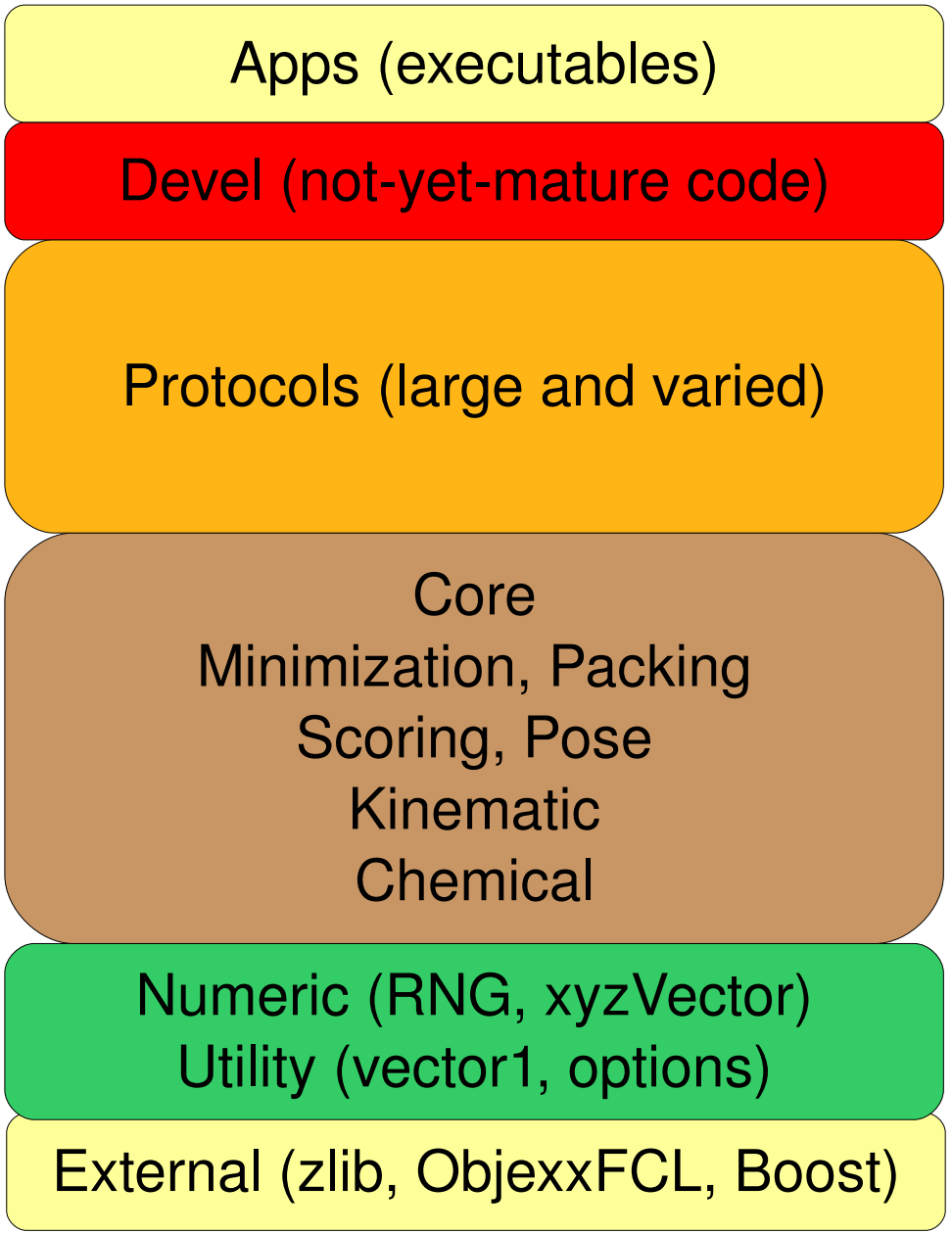
Tutorial

- Depth:
  - Raw C++
  - Most flexible
  - Most difficult (?)
- Other options:
  - Sarel's Parser
  - PyRosetta
  - Many executeables

Compiling

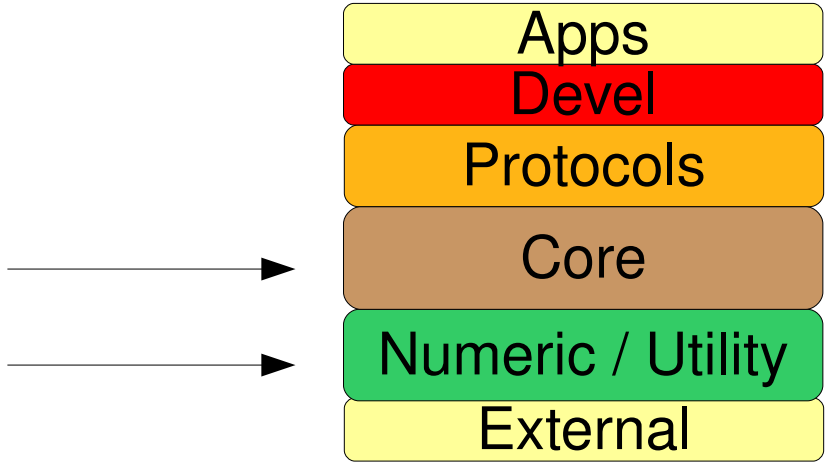
- http://www.rosettacommons.org/manuals/rosetta3_user_guide/build.html
- Scons is included with rosetta (python)
- external/scons.py is executeable
- “scons bin” builds code
- “scons bin mode=release” for the optimizations
- “scons bin -j#” where # is processors you've got

Mini has layers



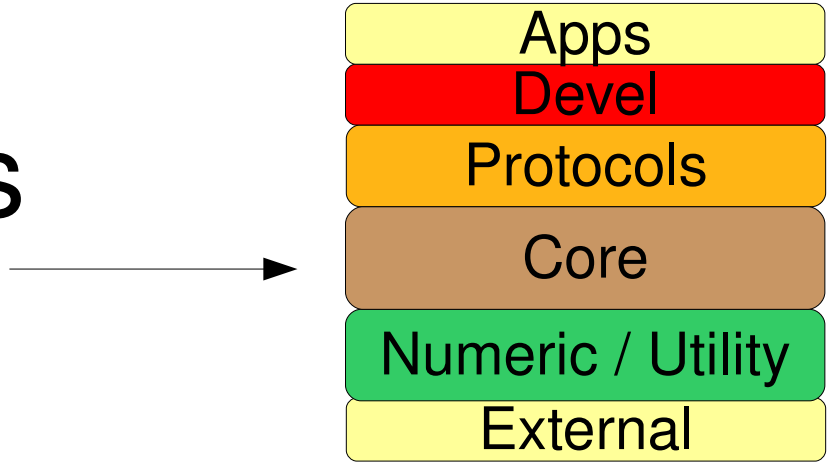
- Layers = libraries
- Enforcement:
  - Directory structure
  - Namespacing
  - SCons building
  - Public shaming

Basic conventions



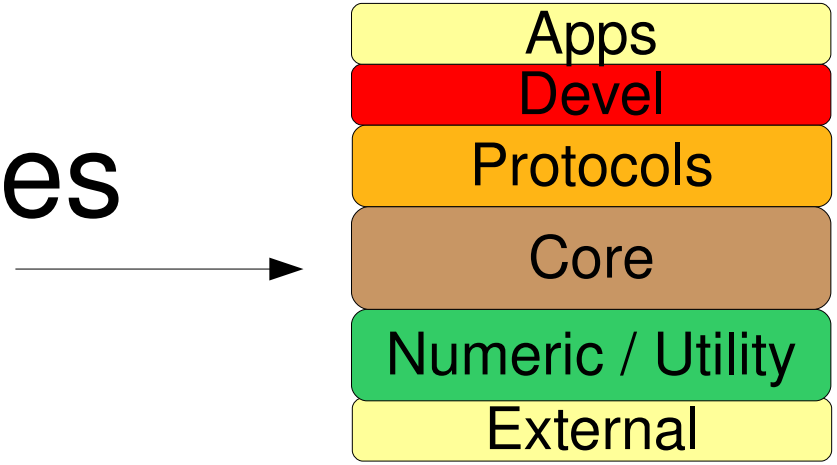
- Owning pointers
  - OP, COP
  - Inherit from `ReferenceCount`
- `vector1`
  - Index from 1, not 0
  - Bounds check <=
- `Core::Size` (uint)
- `Core::Real` (double)
- `Tracer`
  - Replaces `std::cout`
  - Labels output
  - Provides mute control

Core::chemical classes



- Abstract representations
- `AtomType`
- `ResidueType`
  - Defines what atoms are in a residue (or ligand)
  - How they connect internally
- Variant system
  - C-terminal OXT atom
- `ResidueTypeSet`
- `ChemicalManager`
  - Singleton
  - Single read of database

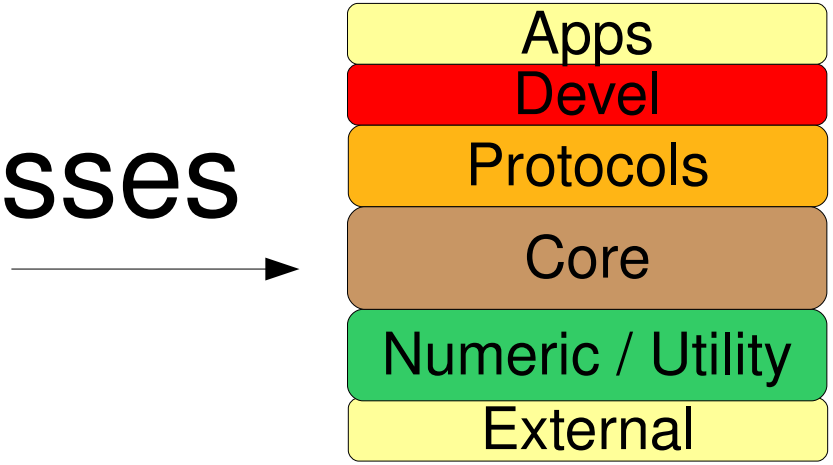
Core::kinematics classes



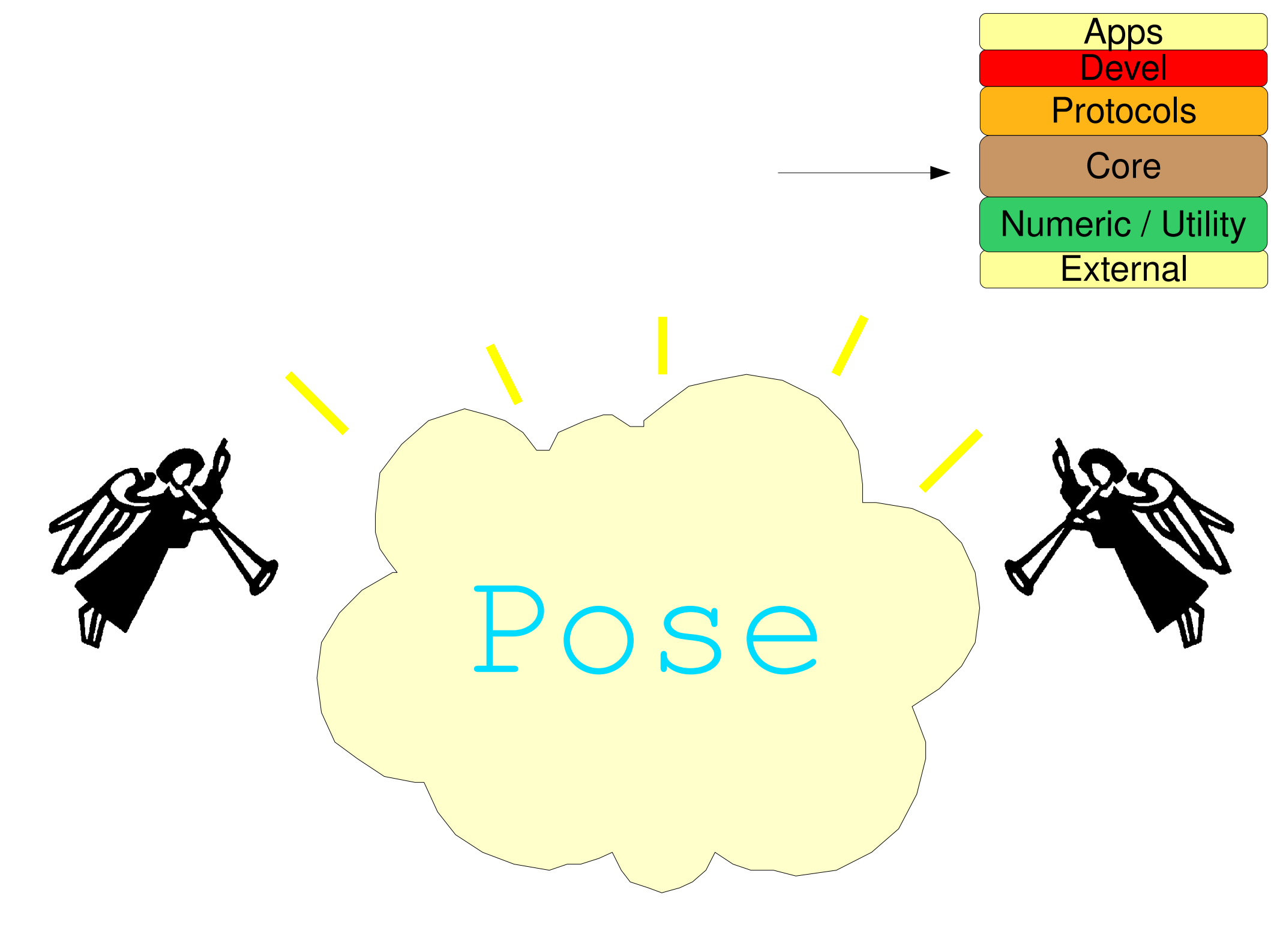
- `AtomTree`
  - Defines atomic connectivity
  - Internal → (x, y, z)
- `FoldTree`
  - Defines residue connectivity
  - Human interface

- `MoveMap`
  - Contains lists of mobile, immobile degrees of freedom

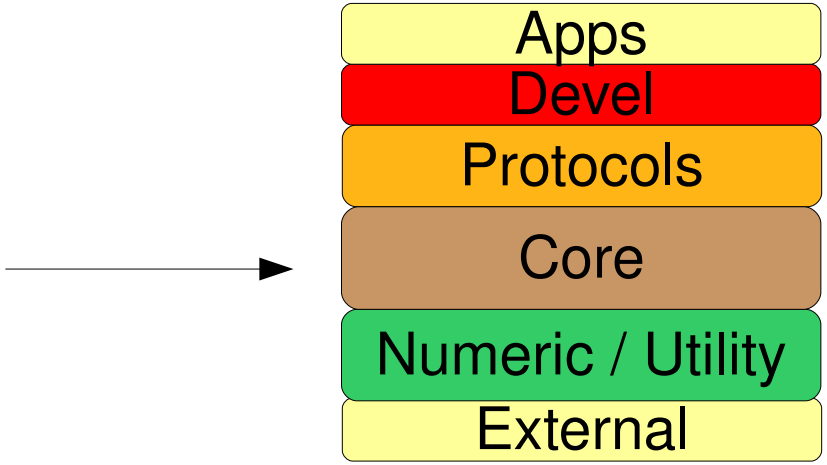
Core::conformation classes



- Abstract chemical + kinematic layers
  - Concrete, distinct
- `Atom`
  - Gives an (x, y, z) to an `AtomType`
- `Residue`
  - Puts `Atom` objects on `ResidueType` skeleton
- `Conformation`
  - Contains `Residue` objects
  - Linked by kinematic layer to describe internal-coordinate folding


Apps
Devel
Protocols
Core
Numeric / Utility
External
Pose

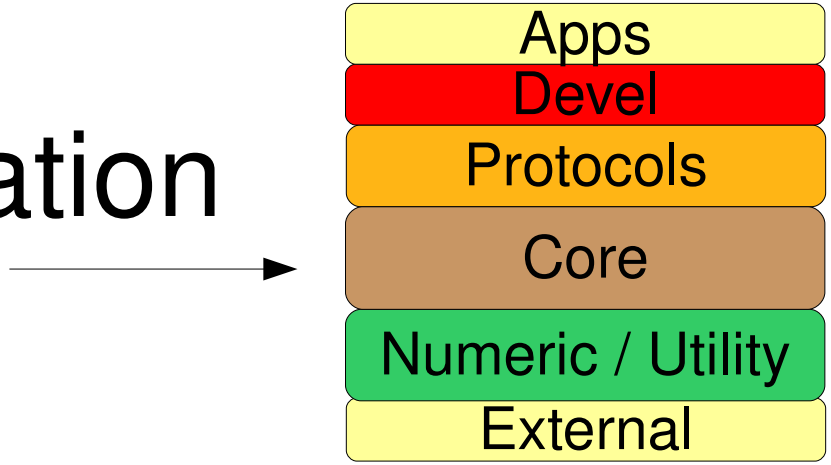
Core::scoring classes



- `Energies`
  - Caches scores, lives in `Pose`
- `ScoreFunction`
  - Scores
  - OPs to `EnergyMethod`s
- `EnergyMethod`
  - Scoring terms
  - Many & varied

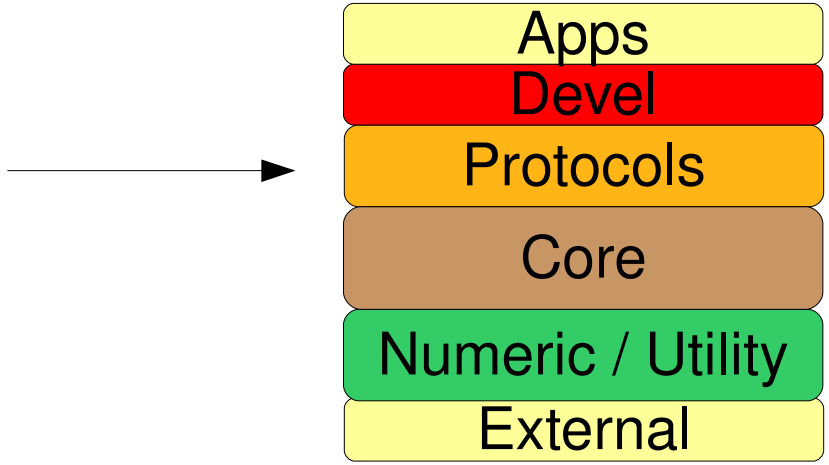
- `ScoringManager`
  - Singleton!
  - Single read of database

Core::pack & ::optimization

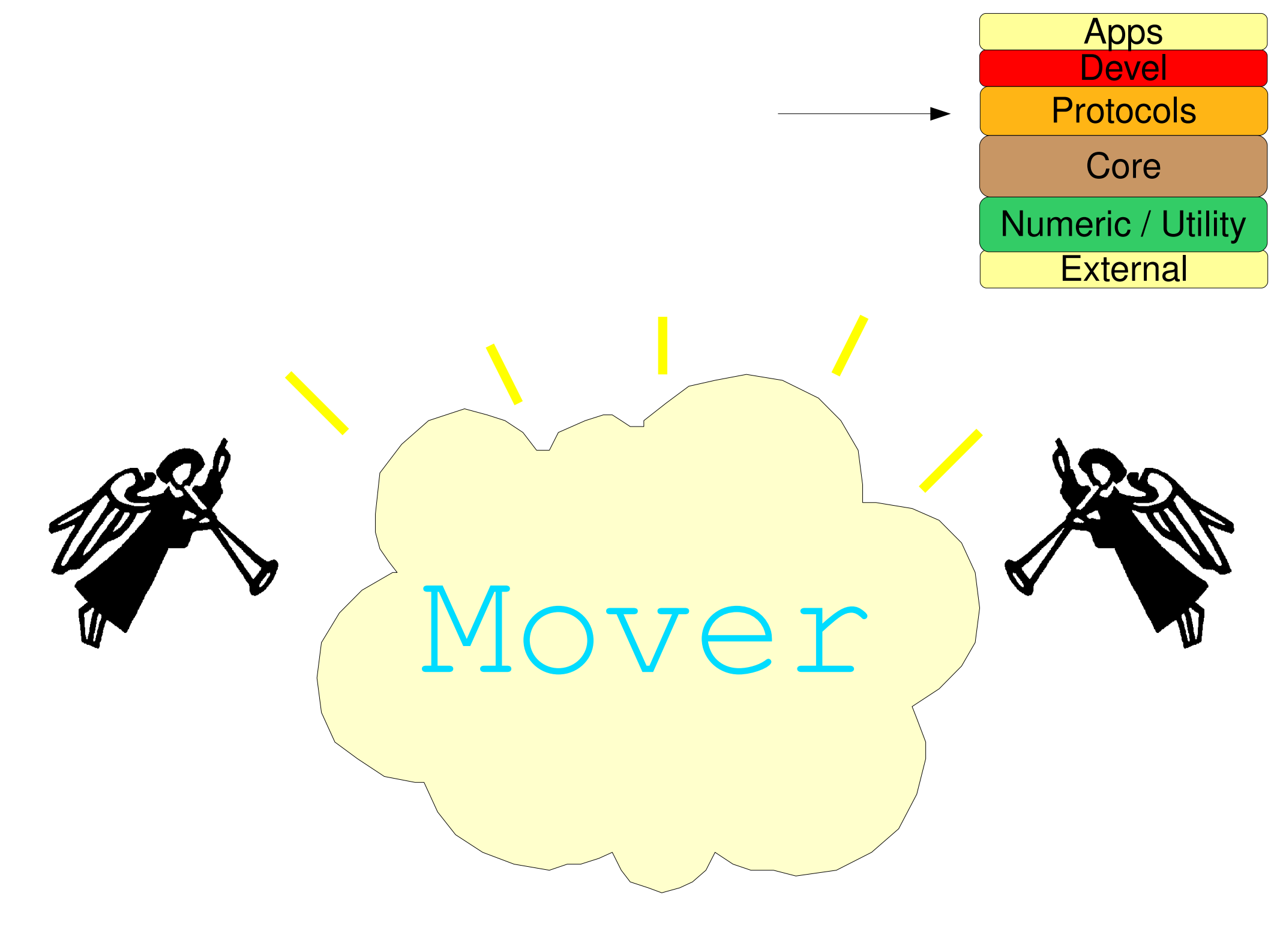


- Guts of packing and minimization
- Very little direct use – almost all through protocols layer
- `PackerTask`
  - Set up what's allowed in packing
  - Disposable
- `TaskFactory`
  - Set up new `PackerTask`s as needed
  - uses `TaskOperation`s

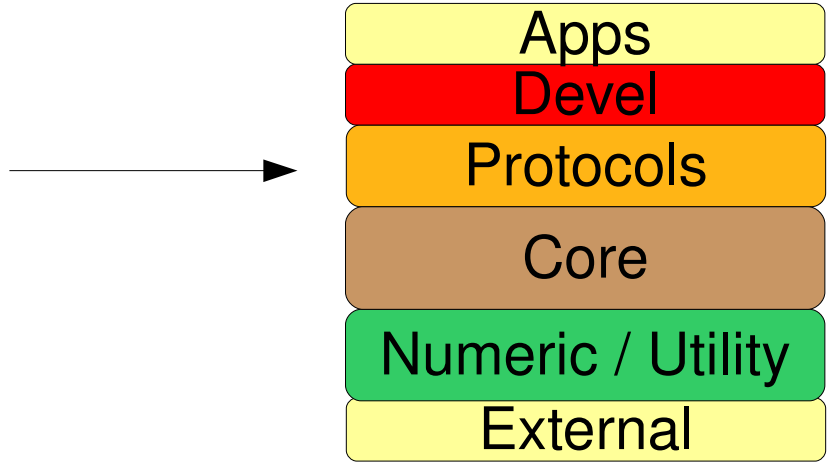
Protocols layer



- `MonteCarlo`
  - Tracks trajectory
- Job distribution
  - 1UBQ_0001.pdb ...
  - Communication layer for MPI
  - New system in Rosetta 3.1 from 3.0
- Gobs of protocols
  - Abinitio
  - Loops
  - Relax
  - Fixbb
- `Mover`...


Apps
Devel
Protocols
Core
Numeric / Utility
External
Mover

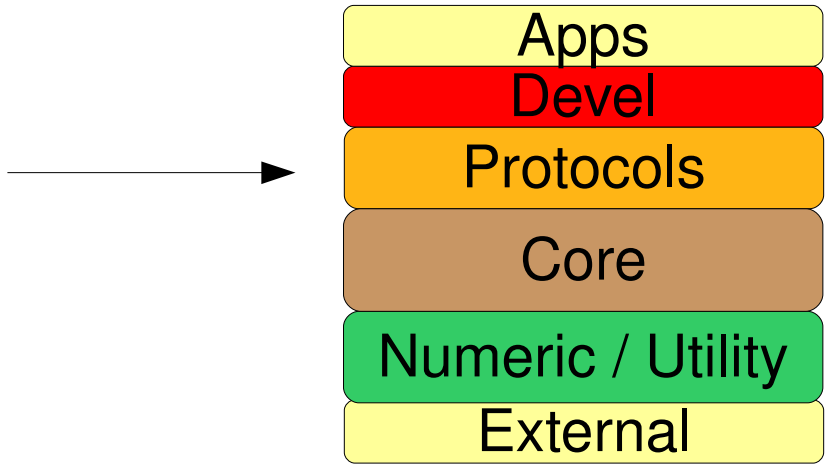
Movers



- `Mover`...
  - Beloved workhorse
  - Centralizes `Pose` alteration
  - Movers can call other movers
    - A protocol is born
  - `virtual void apply( Pose & )`

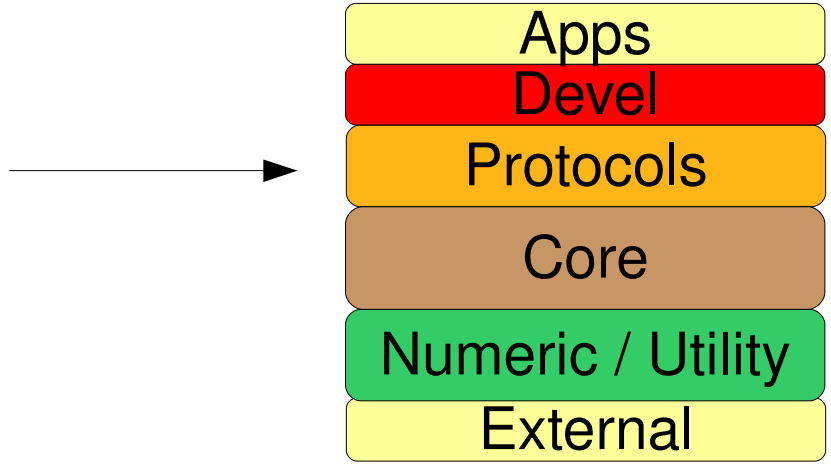
- Simple modifiers
- Empty boxes
- Not-really-a-Mover
- Whole protocols

Simple Movers



- Traditional R++ functions
- Packing
  - `PackRotamersMover`
  - `RotamerTrialsMover`
- Backbone movement
  - `SmallMover`
  - `ShearMover`
  - Fragment movers
- Minimization
  - `MinMover`

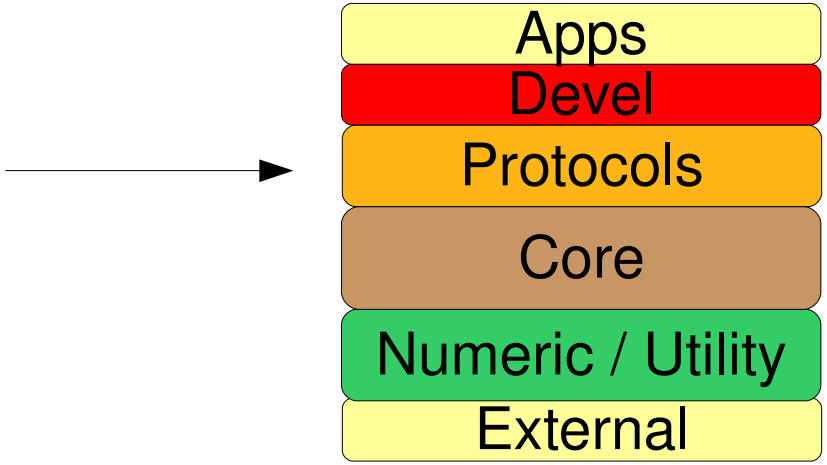
Empty box Movers



- Why empty?
  - Power of polymorphism
  - `MoverOP` argument common
  - Package many movers into one!

- Variety of types
  - `SequenceMover`
  - `RandomMover`
  - `CycleMover`
  - etc...

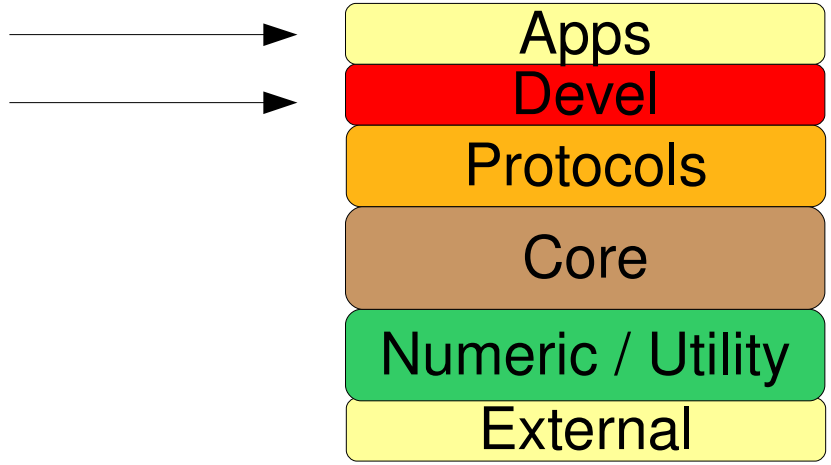
Other Movers



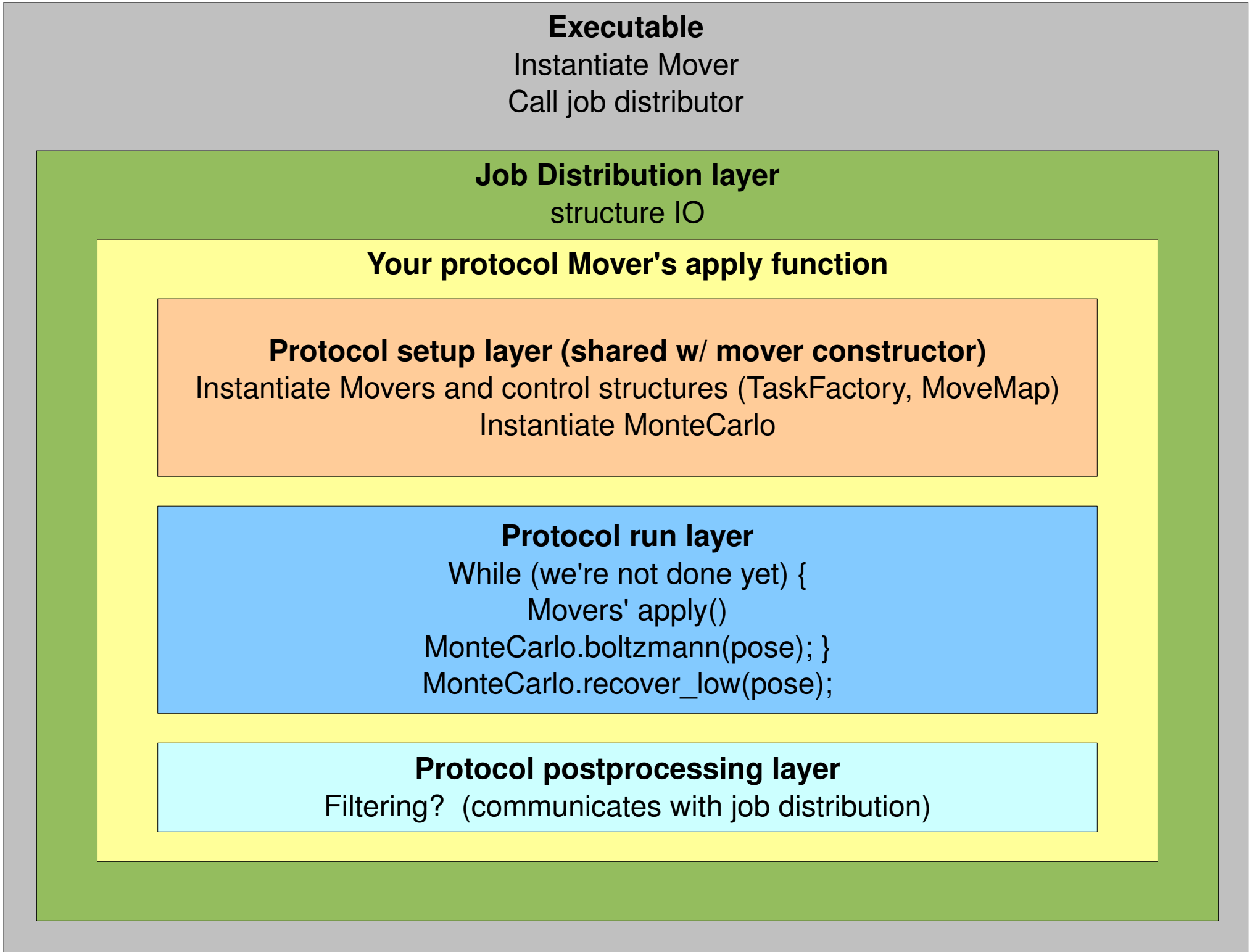
- Whole protocols
  - Job distributor runs a `MoverOP`
  - Allows your protocol to call others
  - Pushes complexity out of Apps layer, into Protocols (reuse)

- Assorted Movers
  - Often do not modify pose
  - Print data
  - Output `Pose` to disk for debugging
  - Ramp `ScoreFunction` weights

Upper layers



- Devel
  - Not present in public releases
  - Code under development, not for general use
- Apps
  - Not a library: just executeables
  - Pilot not released (under development)
  - Public released


Executable
Instantiate Mover
Call job distributor
Job Distribution layer
structure IO
Your protocol Mover's apply function
Protocol setup layer (shared w/ mover constructor)
Instantiate Movers and control structures (TaskFactory, MoveMap)
Instantiate MonteCarlo
Protocol run layer
While (we're not done yet) {
Movers' apply()
MonteCarlo.boltzmann(pose); }
MonteCarlo.recover_low(pose);
Protocol postprocessing layer
Filtering? (communicates with job distribution)

Questions?