Cactus

Einstein Toolkit

Beyond the Algorithm: Supporting Infrastructure for Large Scale Simulation Codes

---

Ian Hinder

Max Planck Institute for Gravitational Physics
(Albert Einstein Institute)
Potsdam, Germany

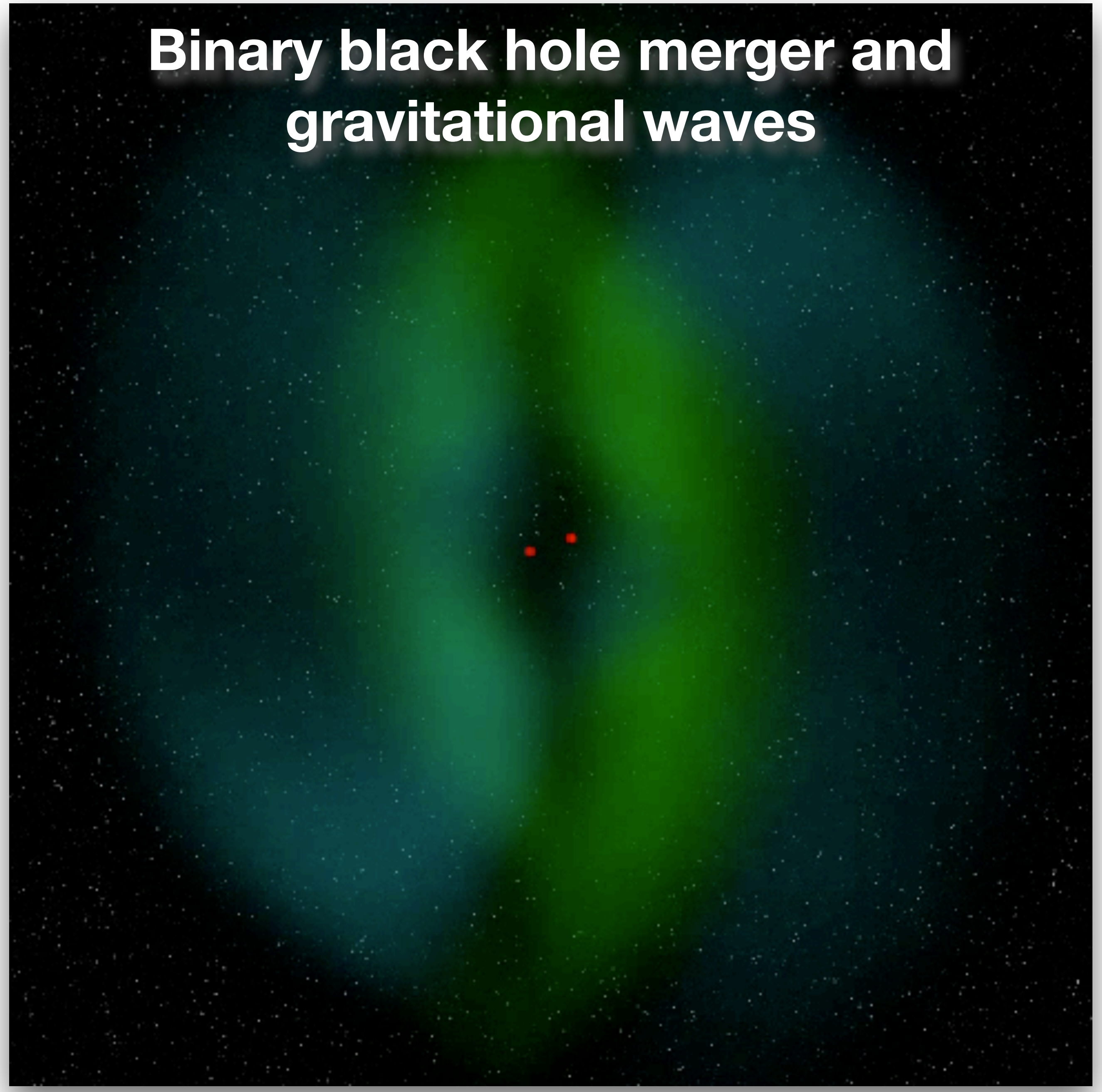
NCSA Software Seminar, Urbana-Champaign, July 2016

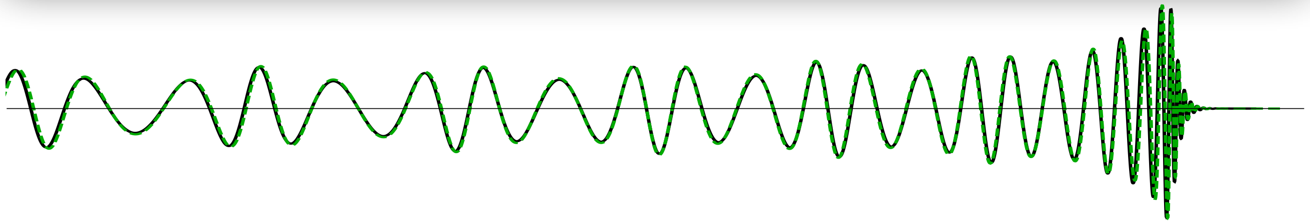
Overview

- The Einstein Toolkit and Cactus
- Automatic code generation
- Abstracting the machine
- Reproducibility
- Software quality control

The Einstein Toolkit and Cactus: Background

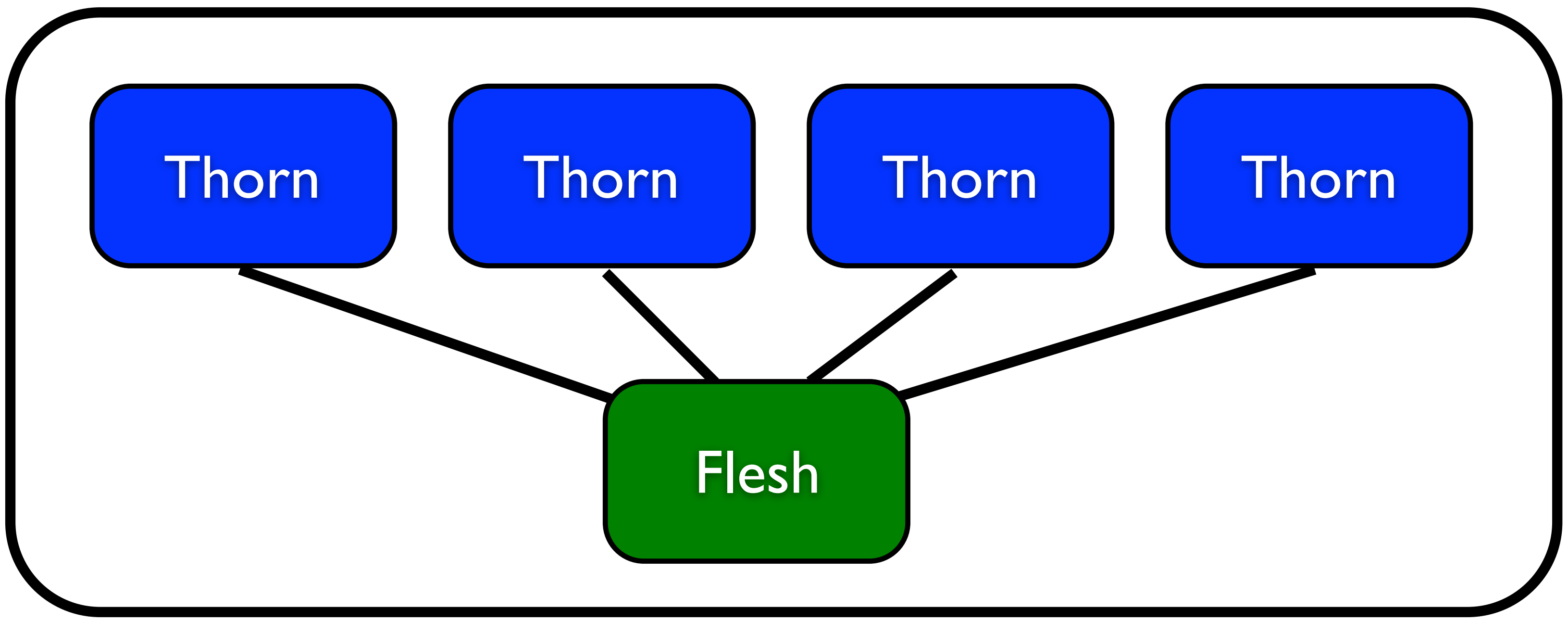
- Open code for **Numerical Relativity**
- einsteintoolkit.org
- Origin: **Ed Seidel's** group at AEI: binary black hole problem, 1995–
- **100** registered users in **56** different groups worldwide
- Einstein equations:
  - complicated **partial differential equations** solved with **finite difference methods**
- Based on **Cactus** framework
- OpenMP/MPI **parallelism** from 100s to 1000s of cores
- **Framework** vs library
  - large number of **components** ("**thorns**") plugged together
  - communication via well-defined **simple interfaces**





Cactus framework

- Cactus modules called **thorns**, all talk to the **flesh**



Executable

cactuscode.org

- Each thorn has:
  - **Metadata** files (interfaces, parameters, scheduling)
  - **Source** files (C/C++/Fortran: physics equations, algorithms, infrastructure)
- The flesh:
  - Defines APIs for **communication** between thorns
- Intent: many groups can **independently develop** public and private codes which all **work together**

The Einstein Toolkit

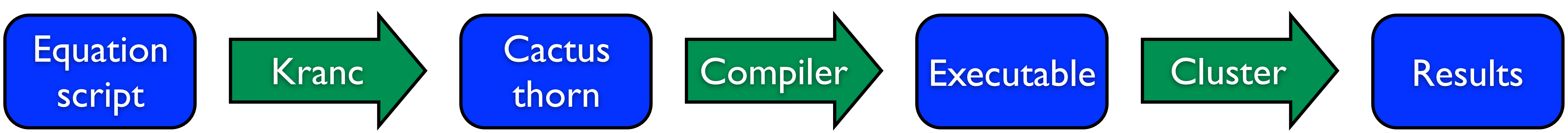
- **Cactus:**
  - flesh
  - support thorns
- **Physics** thorns
- **Mesh refinement** thorns
- **Numerical methods** thorns (interpolator, time integrator, etc)
- **Infrastructure** thorns (3D output, base interfaces, termination management etc)
- **226** Cactus thorns in total
- **Kranc** automatic code generator: generates Cactus thorns
- **SimFactory**: manage simulations/compilation across diverse HPC machines

Automatic code generation

Kranc

Kranc Assembles Numerical Code

- Sascha Husa, IH, Christiane Lechner, 2004
- **High level** description of equations, including tensorial
- "Compiled" to **complete Cactus thorn**
- Application developer sees **equations**, not code
- Almost all Cactus **boilerplate** hidden
- Implemented in **Mathematica**
- kranccode.org



Automatic code generation: Example wave equation

```
initialSineCalc =
{
  Name -> "initial_sine",
  Schedule -> {"AT INITIAL"},
  Equations ->
  {
    phi -> Sin[2 Pi (x - t)],
    pi -> -2 Pi Cos[2 Pi (x - t)]
  }
};
```

```
evolveCalc =
{
  Name -> "calc_rhs",
  Schedule -> {"in MoL_CalcRHS"},
  Equations ->
  {
    dot[phi] -> pi,
    dot[pi] -> Euc[ui,uj] PD[phi,li,lj]
  }
};
```

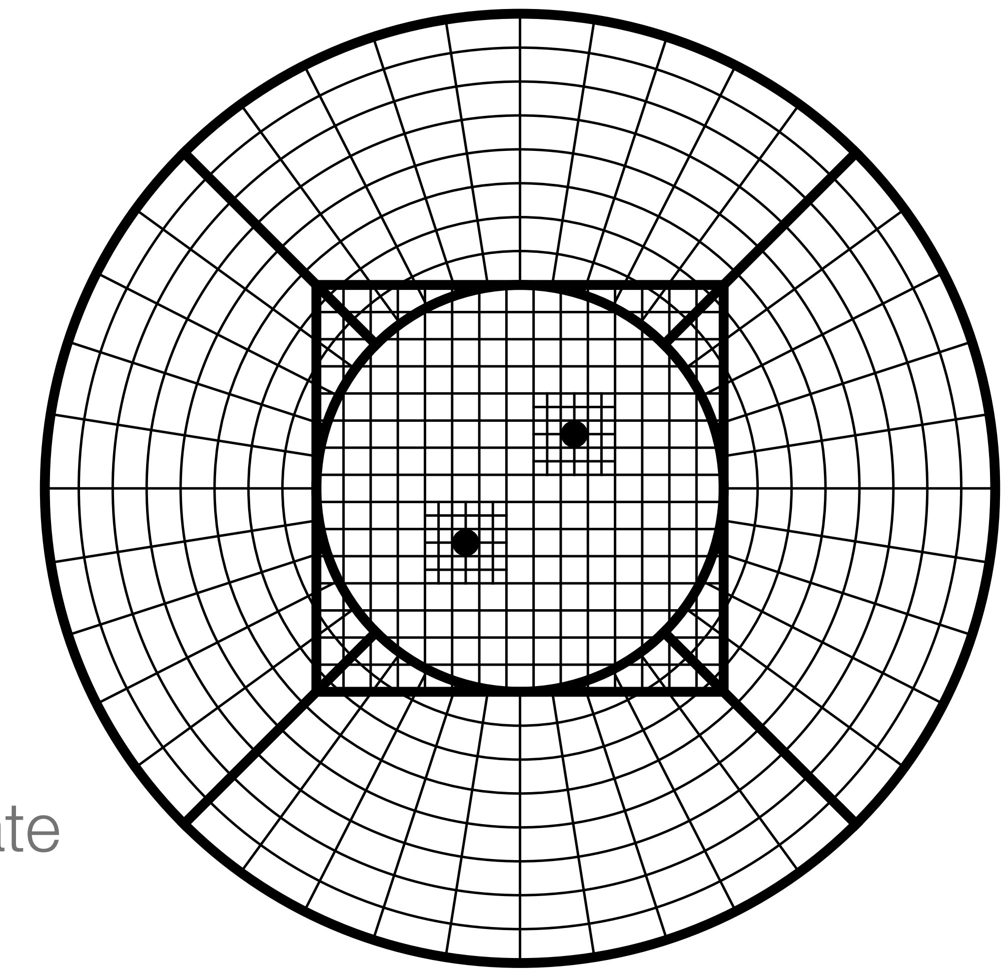
```
CreateKrancThornTT[groups, ".", "SimpleWave",
 Calculations -> {initialSineCalc, evolveCalc},
 PartialDerivatives -> derivatives, DeclaredGroups -> {"evolved_group"}];
```

Automatic code generation

- Solve **time evolution** PDEs in 3D
- **High performance** parallel codes
- End users can treat as **black box**
- Arbitrary order finite differencing
- Existing codes benefit from **new Kranc features**

Automatic code generation: Features

- Arbitrary order finite differencing
- **Mesh refinement** and **multi-block** grids
- **OpenMP**
- High level optimisations:
  - Common **subexpression** elimination
  - **Loop** splitting and joining
- Floating point instruction **vectorisation**: generate compiler intrinsics for all operations
- Experimental support for **CUDA**/**OpenCL**, as well as **Xeon Phi**
- Can implement many of these algorithms at a very **high level** (in Mathematica)



Abstracting the machine: The Simulation Factory

- **Manage simulations:** uniform interface across supercomputers
- by **Erik Schnetter**
- Hide **low-level** cluster-specific details
- Machine database: many **XSEDE** and institution clusters
- Enforce/encourage best practices and avoid common **mistakes**
- **New version** under development (IH, Barry Wardell, Erik Schnetter)
  - **Code-agnostic**; not specific to Cactus
  - Want to try it out with **other codes**
  - **Mostly working**
  - http://simfactory.org

```
sim setup

sim build --thornlist
thornlists/mythorns.th

sim submit mysim --
parfile par/mysim.par
--procs 128 --walltime
12:00:00

sim list-simulations

sim stop mysim
```

Abstracting the machine:
Source code and building

- Centralisation:
  - "*Which machine did I fix that bug on?*"
  - Keep source tree in single **central location**
  - **Sync** code to remote clusters (rsync)
- Building:
  - Database of Cactus "**optionlists**" for each machine
  - **Environment** setup commands: module etc

```
sim sync bluewaters

sim --remote bluewaters build
```

Abstracting the machine: Simulations

- Submitting a simulation to the queue:
  - **Submit scripts** for each machine in machine database
  - Specify parameter file, number of cores, walltime
  - Also: undersubscribing, OpenMP threads, more

```
sim --remote bluewaters submit mysim
  --parfile par/mysim.par --procs 128 --walltime 12:00:00
```

- Simulation lifecycle management: simulation **states**: active(running,queued)/inactive

Abstracting the machine: The Simulation Factory

- Long simulations split into **segments**
- One segment per **job walltime**, e.g. 24 h
- Best practice:
  - No job should **overwrite data** from a previous job
- Checkpoint files hardlinked between segments
- Simulations automatically **submit the next segment** (new version)
- **Termination conditions** defined by regexps
  - FinalTime, EndOfWalltime, DiskQuotaExceeded, UnknownError
- **Termination actions**: Continue, Error, Email
- sim pause, sim continue - request immediate **checkpoint**/restart

Reproducibility

- Copy of **source code** preserved in every simulation (tar.gz archive)
  - Always know **what code was run**, even 10 years later
  - But: difficult to relate to **version controlled** commits
- Multiple components: **multiple repositories**
  - Difficult to identify a single **revision** of "the code"
  - Experimental use of **git submodules** to pull everything together

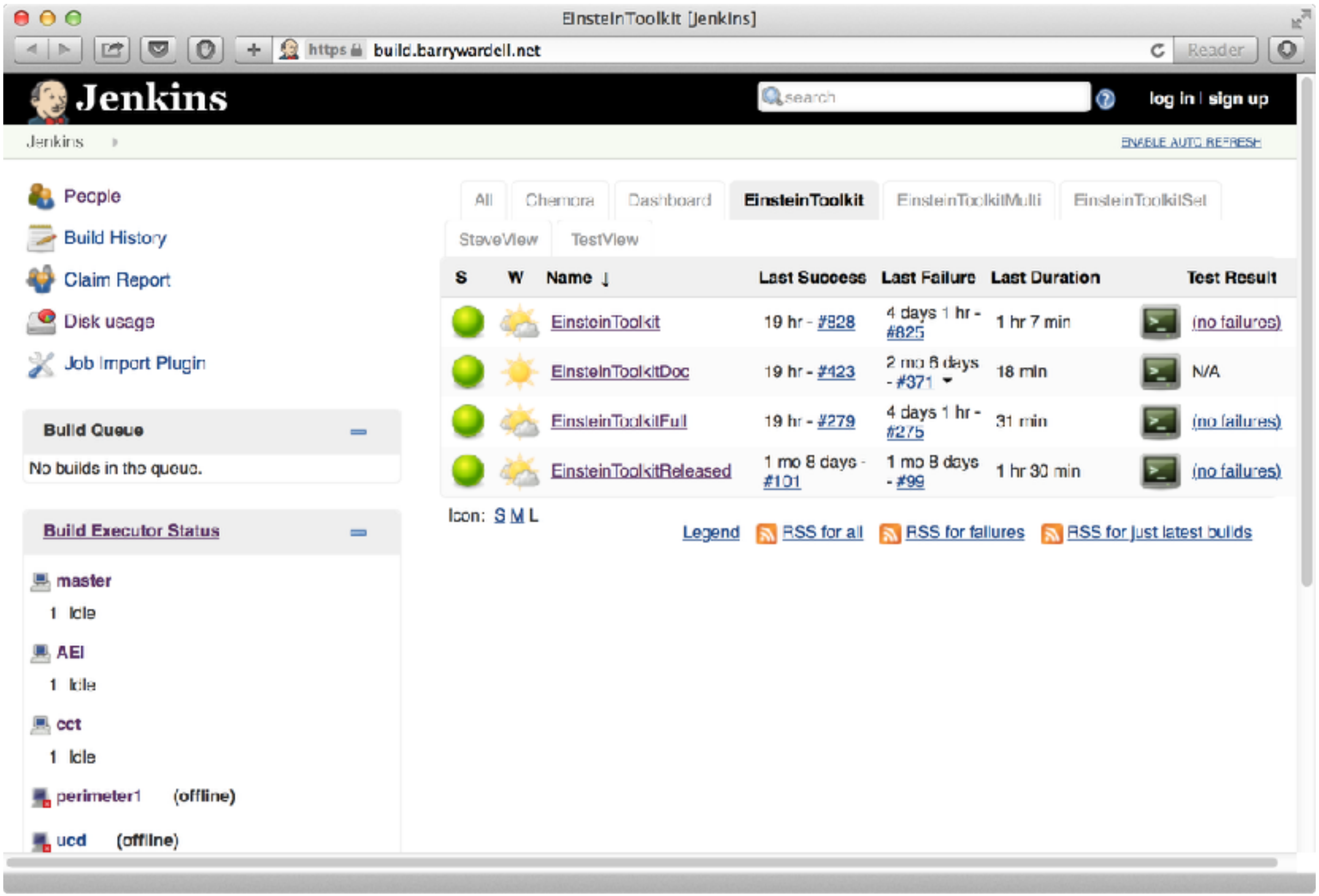
Releases

- Every **6 months**
- **Run tests** on all supported HPC systems, track down and fix problems
- Commit to **backporting** serious issues to last release
- "**Known good**" version that people can use

Testing

- Cactus has standard mechanism for **test cases**
- Mostly **regression tests**: "does this parameter file lead to the same results as the reference data?"
- **Problem**: developers don't run them
- **Solution**: Tests run **after each commit** on a central server
- **Jenkins** web application to manage
- Distributed build nodes
- Integrating with HPC systems is a problem

Testing: Jenkins web application



Tickets and review

- **Ticket system** (TRAC) where people can report problems and track discussion and patches
- Changes **discussed** in a ticket and fixes or enhancements **reviewed** by someone else
- 2nd pair of eyes
- Not always applicable

Thank you!

- Room 4018, leaving tomorrow
- ian.hinder@aei.mpg.de
- https://members.aei.mpg.de/ianhin/

einsteintoolkit.org

cactuscode.org

simfactory.org

kranccode.org

build.barrywardell.net