Cactus is

- a framework for developing portable, modular applications
What is Cactus?

Cactus is

- a framework for developing portable, modular applications
- focusing, although not exclusively, on high-performance simulation codes
What is Cactus?

Cactus is

- a framework for developing portable, modular applications
- focusing, although not exclusively, on high-performance simulation codes
- designed to allow experts in different fields to develop modules based upon their experience and to use modules developed by experts in other fields with minimal knowledge of the internals or operation of the other modules
Cactus Goals

- Portable
  - Different development machines
  - Different production machines
Cactus Goals

- **Portable**
  - Different development machines
  - Different production machines

- **Modular**
  - Standard interfaces for module interaction for easier code interaction, writing and debugging
  - Interchangeable modules with same functionality
Cactus Goals

- **Portable**
  - Different development machines
  - Different production machines

- **Modular**
  - Standard interfaces for module interaction for easier code interaction, writing and debugging
  - Interchangeable modules with same functionality

- **Easy to use**
  - Good documentation
  - Try to let users program the way they are used to
  - Support all major (HPC) programming languages
Philosophy

- Open code base to encourage community contributions
Philosophy

- Open code base to encourage community contributions
- Strict quality control for base framework
Philosophy

- Open code base to encourage community contributions
- Strict quality control for base framework
- Development always driven by real user requirements
Philosophy

- Open code base to encourage community contributions
- Strict quality control for base framework
- Development always driven by real user requirements
- Support and develop for a wide range of application domains
What is Cactus for?

Assume:
- Computational problem
- Too large for single machine
- Distributed development
- Multiple programming languages
What is Cactus for?

Assume:

- Computational problem
- Too large for single machine
  - OpenMP
  - MPI
- Distributed development
- Multiple programming languages
What is Cactus for?

Assume:

- Computational problem
- Too large for single machine
  - OpenMP
  - MPI
- Distributed development
  - Modularize Problem
  - Versioning system(s)
- Multiple programming languages
What is Cactus for?

Assume:

- Computational problem
- Too large for single machine
  - OpenMP
  - MPI
- Distributed development
  - Modularize Problem
  - Versioning system(s)
- Multiple programming languages
  - Modularize Problem
  - Interfaces for inter-language communication
What is Cactus for?

Assume:

- Computational problem
- Too large for single machine
  - OpenMP
  - MPI
- Distributed development
  - Modularize Problem
  - Versioning system(s)
- Multiple programming languages
  - Modularize Problem
  - Interfaces for inter-language communication
Cactus is a direct descendant of many years of code development in Ed Seidel’s group of researchers at the National Center for Supercomputing Applications (NCSA) as the group wrestled to numerically solve Einstein’s Equations for General Relativity and thus model black holes, neutron stars, and boson stars.

In 1995, Ed Seidel and many of his group moved to the Max Planck Institute for Gravitational Physics (Albert Einstein Institute or AEI) in Potsdam, Germany. Frustrated by the difficulties of coordinating the development and use of several different codes across a large collaborative group, Paul Walker, Joan Massó, Ed Seidel, and John Shalf, designed and wrote the original version of Cactus, Cactus 1.0, which provided a collaborative parallel toolkit for numerical relativity.
Current Users and Developers

[Map showing current users and developers located around the world]
Cactus Funding

Organizations:
- Max-Planck-Gesellschaft
- Center for Computation & Technology at LSU
- National Center for Supercomputing Applications
- Lawrence Berkeley National Laboratory
- Washington University
- University of Tübingen

Grants:
- NSF (PHY-9979985, 0540374, 0653303, 0701491, 0701566, PIF-0904015, 0903973, 0903782)
- DFN-Verein (TK 6-2-AN 200)
- DFG (TiKSL)
- ONR (COMI)
- DOE/BOR (OE DE-FG02-04ER46136, BOR DOE/LEQSF)
Cactus Awards

IEEE SCALE09 Challenge Winner 2009
IEEE Sidney Fernback Award 2006
High-Performance Bandwidth Challenge SC2002
High-Performance Computing Challenge SC2002
Gordon Bell Prize for Supercomputing SC2001
HPC “Most Stellar” Challenge Award SC1998
Heinz Billing Prize for Scientific Computing 1998
The flesh is the central component of Cactus. It interfaces with modular components called thorns. The flesh provides:

- Variables & Data Types
- Parameters
- Functions for:
  - Parallelisation
  - Input/Output
  - Coordinates
  - Reduction
  - Interpolation
  - Information
- Staggering
  - Indexing
  - Ghostzones
Some thorns provide additional functionality, while others serve as applications.

Thorns are grouped into **arrangements** which supply some common functionality.

**Example thorns:**

- **CactusIO**  
  input and output operations
- **CactusIOJpeg**  
  JPEG image data compression and writing operations
- **CactusConnect**  
  networking
- **HTTPD**  
  starts the HTTP daemon for remote connections
- **PUGH**  
  unigrid driver + tools; reductions and interpolations
- **PUGH**  
  unigrid driver handles grid scalars, arrays and functions
The **Einstein Toolkit** is a collection of arrangements for computational relativity. The toolkit includes a vacuum spacetime solver (McLachlan), a relativistic hydrodynamics solver, along with thorns for initial data, analysis and computational infrastructure.

The **Cactus Computational Toolkit** is a collection of arrangements that provides general computational infrastructure.
I/O Capabilities

Usual I/O and checkpointing in different formats:

- Screen output
- ASCII file output
- HDF5 file in-/output
- Online Jpeg rendering
- Online VisIt visualization
More Capabilities: Grids, Boundaries, Symmetries

- **Grids**
  - Only structured meshes (at the moment)
  - Unigrid (PUGH)
  - Adaptive Mesh Refinement (Carpet)

- **Boundaries / Symmetries**
  - Periodic
  - Static
  - Mirror symmetries
  - Rotational symmetries
  - Problemspecific boundaries
The Cactus Computational Toolkit

Core modules (thorns) providing many basic utilities:

- I/O methods
- Boundary conditions
- Parallel unigrid driver
- Reduction and Interpolation operators
- Interface to external elliptic solvers
- Web-based interaction and monitoring interface
- Simple example thorns (wavetoy)
Many arrangements with many modules...

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CactusBase</td>
<td>Basic utility and interface thorns</td>
</tr>
<tr>
<td>CactusBench</td>
<td>Benchmark utility thorns</td>
</tr>
<tr>
<td>CactusConnect</td>
<td>Network utility thorns</td>
</tr>
<tr>
<td>CactusElliptic</td>
<td>Elliptic solvers / interface thorns</td>
</tr>
<tr>
<td>CactusExamples</td>
<td>Example thorns</td>
</tr>
<tr>
<td>CactusExternal</td>
<td>External library interface thorns</td>
</tr>
<tr>
<td>CactusIO</td>
<td>General I/O thorns</td>
</tr>
<tr>
<td>CactusNumerical</td>
<td>General numerical methods</td>
</tr>
<tr>
<td>CactusPUGH</td>
<td>Cactus Unigrid Driver thorn</td>
</tr>
<tr>
<td>CactusPUGHIO</td>
<td>I/O thorns specifix for PUGH driver</td>
</tr>
<tr>
<td>CactusTest</td>
<td>Thorns for Cactus testing</td>
</tr>
<tr>
<td>CactusUtils</td>
<td>Misc. utility thorns</td>
</tr>
<tr>
<td>CactusWave</td>
<td>Wave example example thorns</td>
</tr>
</tbody>
</table>
A Cactus Thorn will contain three definition files and a folder with the source code and compilation instructions.
Cactus Configuration Language Files

The interaction of a thorn with the Flesh and other thorns is controlled by certain configuration files:

- **interface.ccl** defines the implementation (Section B2.3) the thorn provides, and the variables the thorn needs, along with their visibility to other implementations.

- **schedule.ccl** defines which functions are called from the thorn and when they are called; it also handles memory and communication assignment for grid variables.

- **param.ccl** defines the parameters that are used to control the thorn, along with their visibility to other implementations.

These are called *Cactus Configuration Language* files.
param.ccl:

:  
# Parameter definitions for thorn HTTPD  
# $Header$

shares: Cactus

# cctk variables used  
USES REAL cctk_final_time  
USES REAL cctk_initial_time  
USES INT cctk_itlast  
# time of last iteration

private:

# sets the port number  
INT port "HTTP port number to use"  
{
    # ports 1-65535 are valid  
    1:65535 :: "Any valid port"  
} 5555  
# 5555 is the default port

# if true, starts the simulation paused  
BOOLEAN pause "Pause ?" STEERABLE = ALWAYS  
{
    } "no"  
# do not start simulation paused by default

# username and password for controlling cactus  
STRING user "The username for Cactus Control "  
{
    # any non-empty string is valid
interface.ccl:

# Interface definition for thorn HTTPD
# $Header$

# state what thorn is implemented
Implements: HTTPD
# inherit all the functions of Socket
Inherits: Socket

# name of the include header
USES INCLUDE HEADER: SocketUtils.h

INCLUDE HEADER: Auth.h in http_Auth.h
INCLUDE HEADER: Cookies.h in http_Cookies.h
INCLUDE HEADER: Steer.h in http_Steer.h
INCLUDE HEADER: Content.h in http_Content.h

CCTK_INT FUNCTION Send_Twitter_Msg(CCTK_STRING IN msg)
USES FUNCTION Send_Twitter_Msg
schedule.ccl:

# Schedule definitions for thorn HTTPD
# $Header$

# perform these functions at startup
SCHEDULE GROUP HTTP_Startup AT startup
{
    OPTIONS: GLOBAL
} "HTTP daemon startup group"

SCHEDULE HTTP_StartServer in HTTP_Startup
{
    LANG: C
    OPTIONS: GLOBAL
} "Start HTTP server"

# during startup but after HTTP_SetupPages, do HTTP_FirstServ
SCHEDULE HTTP_FirstServ in HTTP_Startup AFTER HTTP_SetupPages
{
    LANG: C
    OPTIONS: GLOBAL
} "Serve first pages at startup"

# poststep is after startup and after every evolution step
# HTTP_Work at poststep but before IOUtil_UpdateParFile
SCHEDULE HTTP_Work AT poststep BEFORE IOUtil_UpdateParFile
{
    LANG: C
    OPTIONS: GLOBAL
} "Working routine"

if(provide_pages)
{
    SCHEDULE HTTP_ContentWork AT poststep
    {
        LANG: C
        OPTIONS: GLOBAL
    } "Content Working routine"
}