# Introduction to the Cactus Framework

The Cactus team

Jun 22 2009

# What is Cactus?

Cactus is

- a framework for developing portable, modular applications solving 3D PDE time evolutions

# What is Cactus?

Cactus is

- a framework for developing portable, modular applications solving 3D PDE time evolutions
- focusing, although not exclusively, on high-performance simulation codes

# What is Cactus?

Cactus is

- a framework for developing portable, modular applications solving 3D PDE time evolutions
- 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 thorn 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 thorn 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 the major programming languages

# Philosophy

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

# Cactus History

- Direct descendant of many years of code development in Ed Seidel's group of researchers at NCSA
- 1995, Paul Walker, Joan Masso, Ed Seidel, and John Shalf: Cactus 1.0
- Originally for numerical relativity
- Over the years generalized for use by scientists in other domains
- Current version: 4.0 beta 16

# Current Users and Developers

# Community

- Web: `http://www.cactuscode.org/`
- Email lists
  - `users@cactuscode.org`
  - `developers@cactuscode.org`
- Download: cvs (`cvs.cactuscode.org:/cactus`)
- Bug tracker

# Covers


COMMUNICATIONS
HIGH-PERFORMANCE NETWORKING
The Future of Theoretical Physics and Cosmology
MaxPlanckResearch
Sun, Man and Climate
Invisible U
National Science Foundation
Grav
from the grou
National Science Board
Science and Engineering Indicators 2006
Volume 1
Visualization and Processing of Tensor Fields
Chapman & Hall/CRC
Computational Science Series
Petascale Computing
Algorithms and Applications
EDITED BY
DAVID A. BADER
Chapman & Hall/CRC
Naturwissenschaftliche Rundschau
NR 689

# 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)
  - Europ. Commission (HPRN-CT-2000-00137, IST-2001-32133)
  - DFN-Verein (TK 6-2-AN 200)
  - DFG (TiKSL)
  - ONR (COMI)
  - DOE/BOR (OE DE-FG02-04ER46136, BOR DOE/LEQSF)

# The Cactus Computational Toolkit

Core parts (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)

# 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

# Visualization

Visualization is important, because

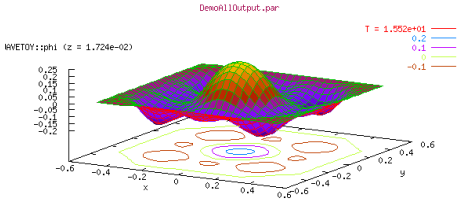
- can give essential insight into dynamics
- can detect errors much better than just looking at numbers
- necessary for publications
- can be good advertisement (cover pages ...)

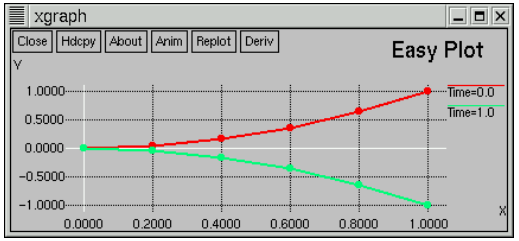
Different visualization types:

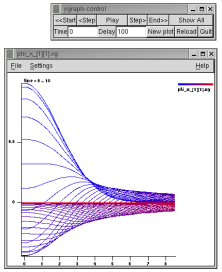
- 1D/2D: most of published pictures
- 3D: usually done for movies

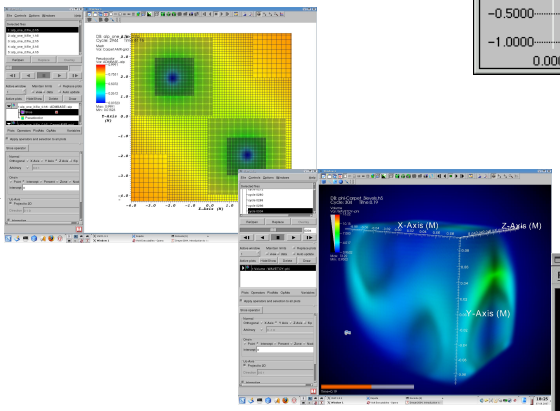
# Visualization Clients

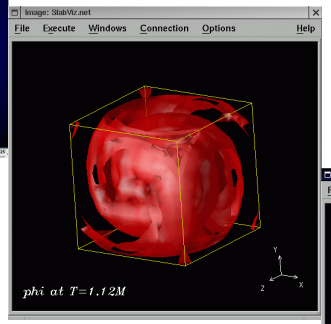
- Output can be visualized by many clients, e.g.:
  - 1D/2D
    - gnuplot
    - xgraph
    - ygraph
  - 3D
    - VisIt
    - OpenDX
    - Amira

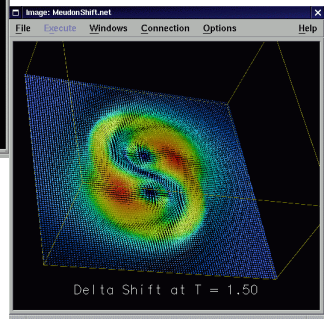


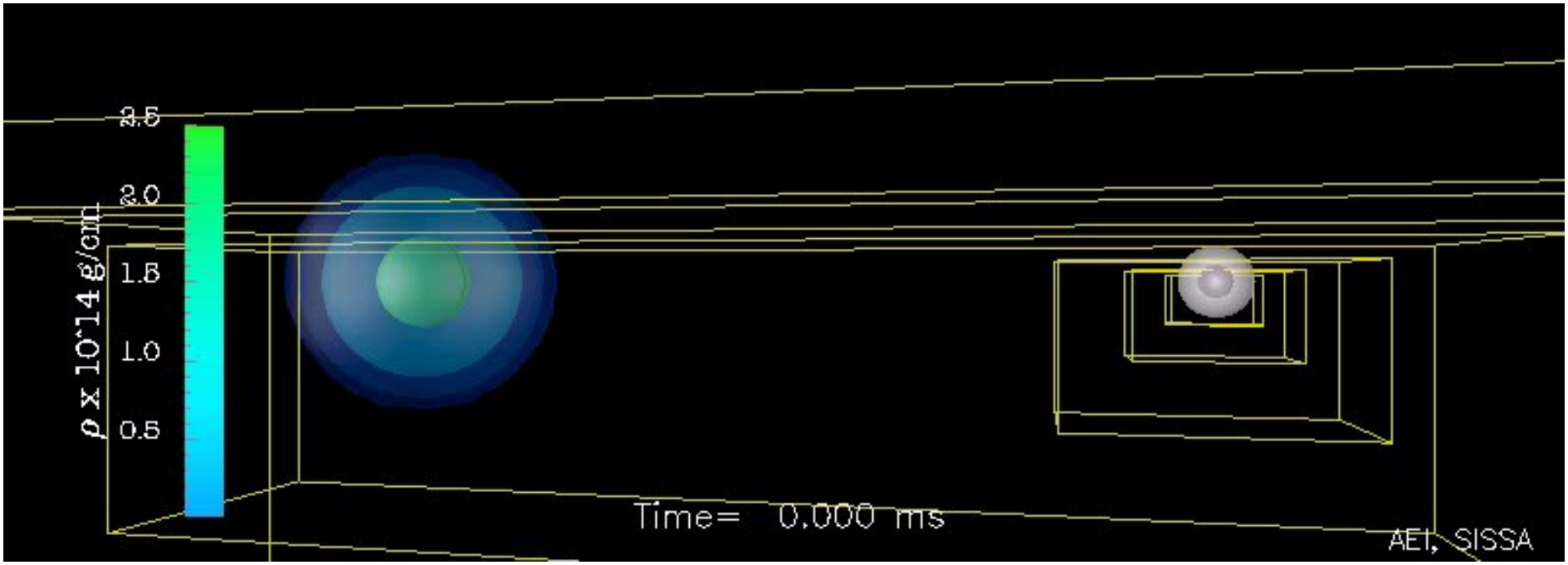





# Example Movie I




Merger of neutron star and black hole

# Example Movie II

Binary black hole inspiral, merger and ringdown

# Hands-on

Cactus Structure

# Framework - no prebuild executable

Cactus

- does not provide executable files
- provides infrastructure to create executables

Why?

- Problemspecific code not part of Cactus
- System libraries different on different systems
- Cactus is free software, but often problemspecific codes are not $\rightarrow$ non-distributable binary

# Structure Overview

Two fundamental parts:

- The Flesh
  - The core part of Cactus
  - Independent of other parts of Cactus
  - Acts as utility and service library

# Structure Overview

Two fundamental parts:

- The Flesh
  - The core part of Cactus
  - Independent of other parts of Cactus
  - Acts as utility and service library
- The Thorns
  - Separate libraries (modules) which encapsulate the implementation of some functionality
  - Can specify dependencies on other implementations

# Configuring Cactus

- Cactus knows about a lot of default system configurations
- However: often system specific configuration needed
- Example config file:

```
CC  = /usr/local/compilers/intel_cce_11.0.074.x86_64/bin/intel64/icc
CXX = /usr/local/compilers/intel_cce_11.0.074.x86_64/bin/intel64/icpc
F77 = /usr/local/compilers/intel_fce_11.0.074.x86_64/bin/intel64/ifort
F90 = /usr/local/compilers/intel_fce_11.0.074.x86_64/bin/intel64/ifort

CPPFLAGS = -openmp -DMPICH_IGNORE_CXX_SEEK
FPPFLAGS = -fopenmp -traditional
CFLAGS   = -openmp -g -debug all -align -std=c99 -U__STRICT_ANSI__
CXXFLAGS = -openmp -g -debug all -align -std=c++0x -restrict -D__builtin_isnan=::isnan
F77FLAGS = -openmp -g -debug all -align -pad -traceback -w95 -cm
F90FLAGS = -openmp -g -debug all -align -pad -traceback -w95 -cm

DEBUG          = yes
OPTIMISE       = yes
WARN           = yes

MPI       = MPICH
MPICH_DIR = /usr/local/packages/numrel/mpich-1.2.7p1

HDF5     = yes
HDF5_DIR = /usr/local/packages/numrel/hdf5-1.8.0
```

# Creating a Configuration

- Configurations consist of
  - a particular list of thorns
  - particular configuration options
- Creating configuration:
  `make <NAME>-config`
- New source tree:

```
`-- Cactus
      :
      |-- Makefile
      |-- arrangements ...
      |-- doc
      |-- lib
      |-- src
      `-- configs
            `-- NAME
```

# Building and Running a Cactus binary

- Building new configuration:
  `make <NAME>`
- Executable in `Cactus/exe` subdirectory

```
`-- Cactus
    :
    |-- doc
    |-- lib
    |-- src
    |-- configs ...
    `-- exe
        |-- cactus_HelloWorld
        `-- cactus_wavetoy
```

- Execute Syntax: `<executable> [options] <parameter file>`
- Example: `./exe/cactus_HelloWorld HelloWorld.par`

# The Flesh

- Make system
  - Organizes builds as 'configurations'
  - Can build various documentation documents
  - Can update Cactus flesh and thorns
- Scheduling: Sophisticated scheduler which calls thorn-provided functions as and when needed
- API: Interface for functions to be callable from one thorn by another
- CCL: Configuration language which describes necessary details about thorns to the flesh

# The Driver

- Special thorn
- Only one active for a run, choose at starttime
- The only code (almost) dealing with parallelism
- Other thorns access parallelism via API
- Underlying parallel layer transparent to application thorns
- Examples
  - PUGH: unigrid, part of the Cactus computational toolkit
  - Carpet: mesh refinement, `http://www.carpetcode.org`

# Grid functions

Cactus provides methods to

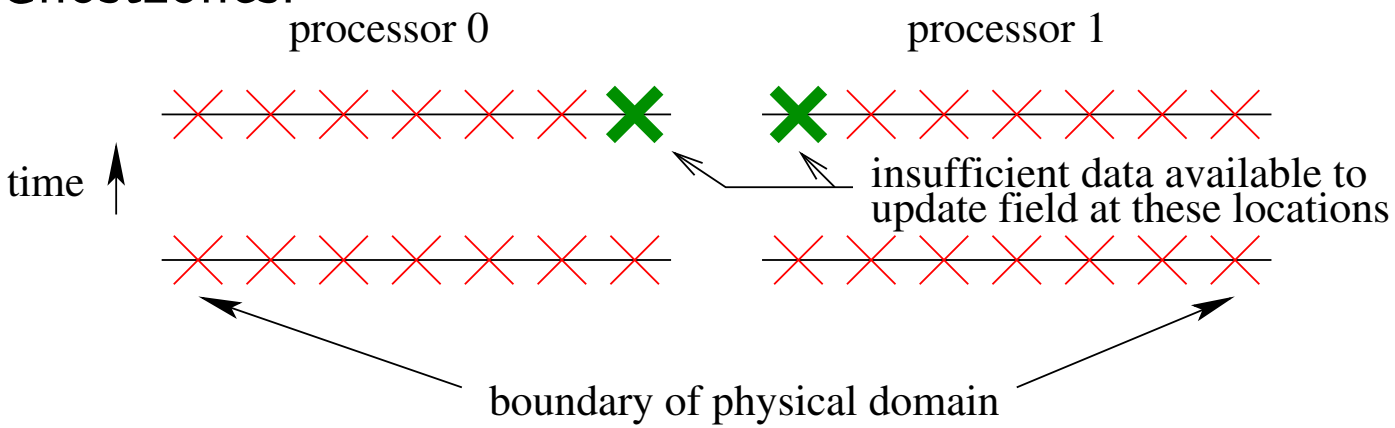
- Distribute variables across processes (grid function)
- Synchronize processor domain boundaries between processes
- Compute reductions across grid functions
- Actual implementation in driver thorn

# Ghost zones

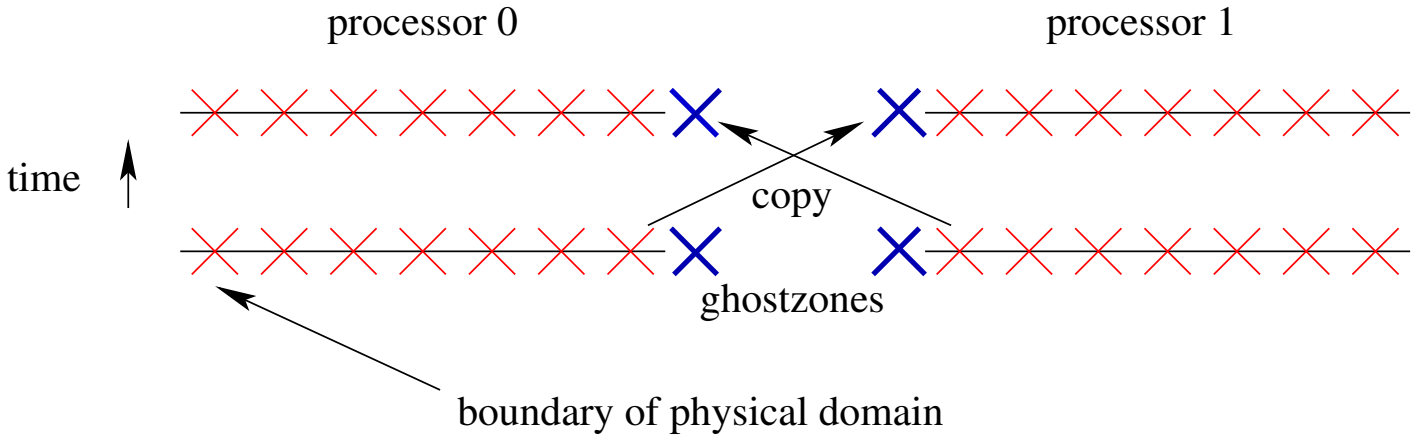
- Grid variables: distributed across processes
- Assumption: Most work done (quasi-) locally:
  True for hyperbolic and parabolic problems
- Split of computational domain into blocks
- Only communication at the borders (faces)
- At least stencil size many ghostzones

# Ghost zone example

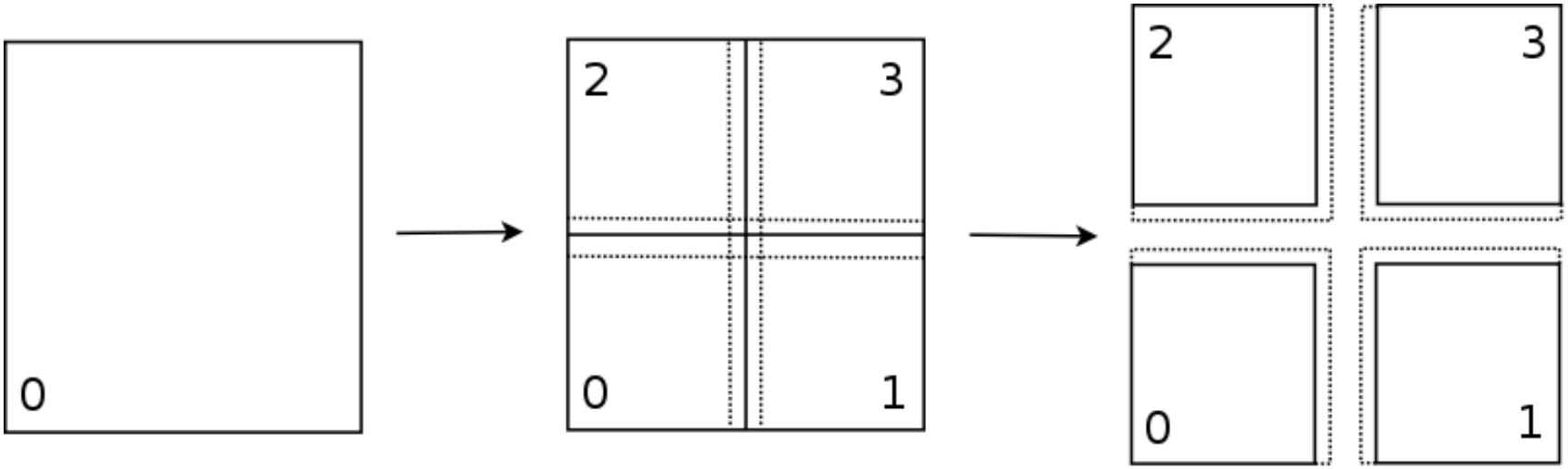
Without Ghostzones:



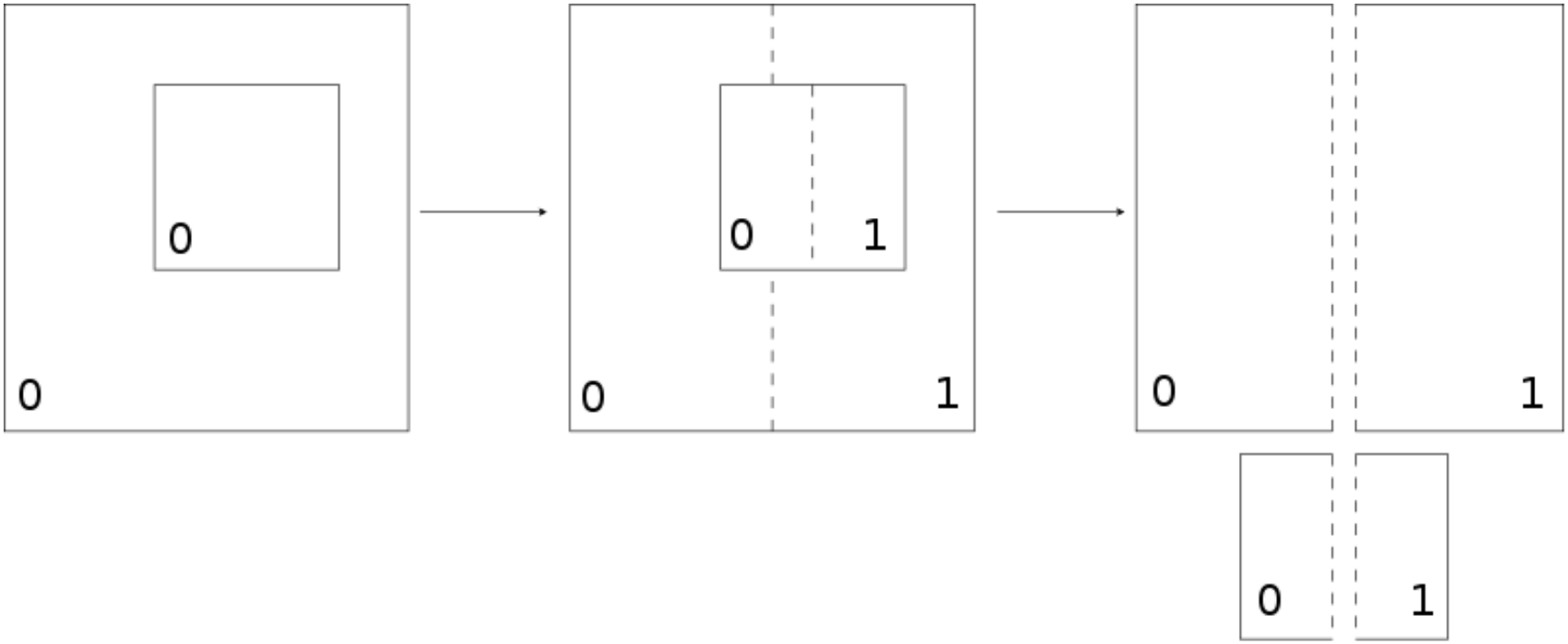

With Ghostzones:



# Domain decomposition



# Mesh refinement decomposition



# Arrangements

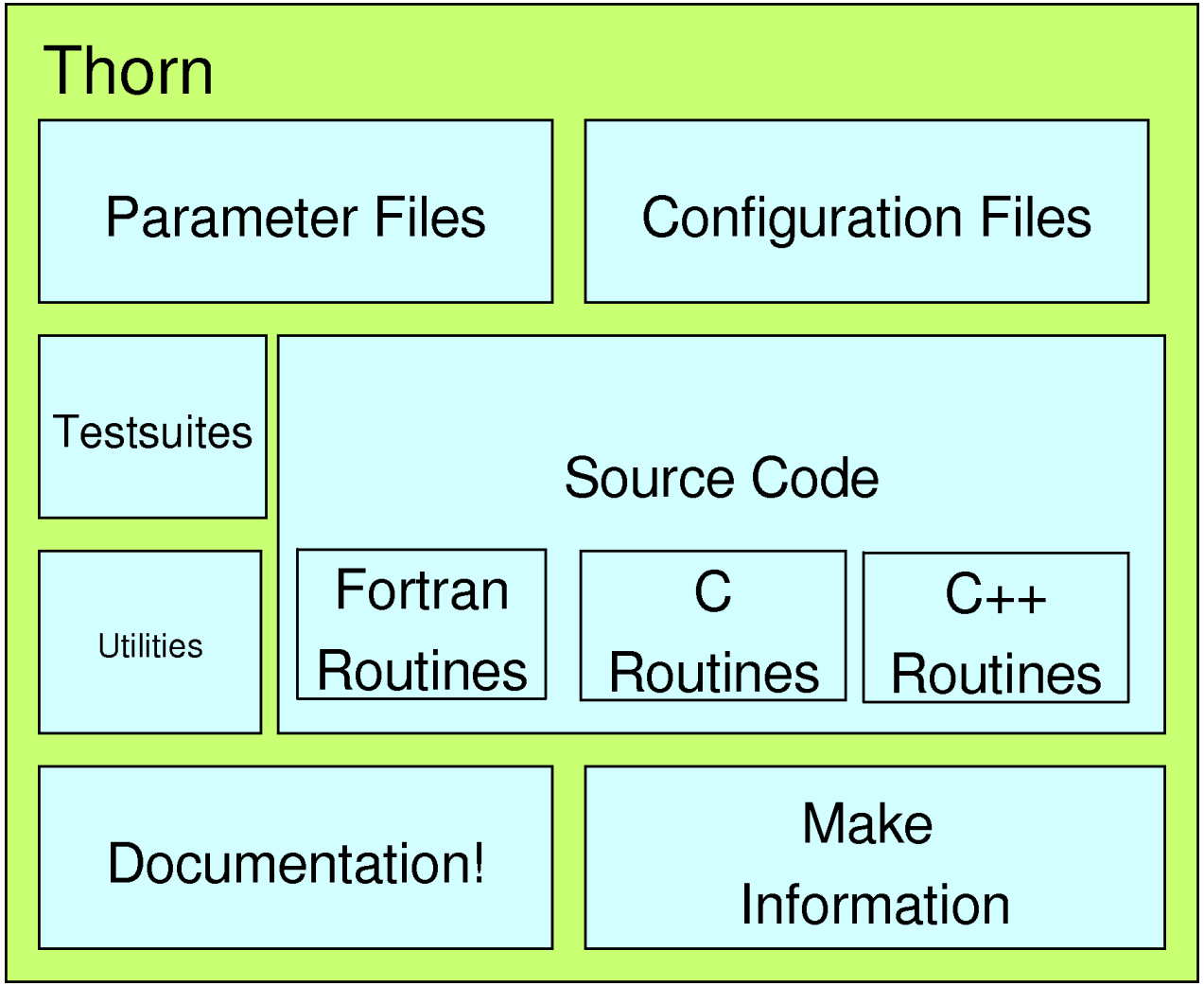
- Group of thorns
- Organizational convenience
- Something in common:
  - Related functionality
  - Same author(s), research group
- Toolkit: Collection of Arrangements

Directory structure:

```
Cactus
`-- arrangements
    |-- Arrangement-A
    |   |-- Thorn-A1
    |   `-- Thorn-A2
    `-- Arrangement-B
        |-- Thorn-B1
        `-- Thorn-B2
```

# Thorn Structure

Inside view of a plug-in module, or thorn for Cactus



# Thorn Structure

Directory structure:

```
Cactus
`-- arrangements
    `-- Introduction
        `-- HelloWorld
            |-- interface.ccl
            |-- param.ccl
            |-- schedule.ccl
            |-- README
            |-- doc
            |   `-- documentation.tex
            |-- src
            |   |-- HelloWorld.c
            |   `-- make.code.defn
            |-- test
            `-- utils
```

# Thorn Structure

Directory structure:

```
Cactus
`-- arrangements
    `-- Introduction
        `-- HelloWorld
            |-- interface.ccl
            |-- param.ccl
            |-- schedule.ccl
            |-- README
            |-- doc
            |   `-- documentation.tex
            |-- src
            |   |-- HelloWorld.c
            |   `-- make.code.defn
            |-- test
            `-- utils
```

# Thorn Specification

Three configuration files per thorn:

- interface.ccl declares:
  - an 'implementation' name
  - inheritance relationships between thorns
  - Thorn variables
  - Global functions, both provided and used

# Thorn Specification

Three configuration files per thorn:

- interface.ccl declares:
  - an 'implementation' name
  - inheritance relationships between thorns
  - Thorn variables
  - Global functions, both provided and used
- schedule.ccl declares:
  - When the flesh should schedule which functions
  - When which variables should be allocated/freed
  - Which variables should be syncronized when

# Thorn Specification

Three configuration files per thorn:

- interface.ccl declares:
  - an 'implementation' name
  - inheritance relationships between thorns
  - Thorn variables
  - Global functions, both provided and used
- schedule.ccl declares:
  - When the flesh should schedule which functions
  - When which variables should be allocated/freed
  - Which variables should be syncronized when
- param.ccl declares:
  - Runtime parameters for the thorn
  - Use/extension of parameters of other thorns

# interface.ccl

Variables:

- Flesh needs to know about thorn variables for which it has to care about allocation, parallelism, inter-thorn use
- Scopes: public, private
- Many different basic types (double, integer, string, ...
- Different 'group types' (grid functions, grid arrays, scalars, ...)
- Different 'tags' (not to be checkpointed, vector types, ...)

# Syntax of interface.ccl

```
IMPLEMENTS: <interface name>
INHERITS: <interface name> ...

[PUBLIC:|PRIVATE:]

[REAL|COMPLEX|INT] <group name> TYPE=[gf|array]
        TIMELEVELS=<number> [DIM=... SIZE=...]
{
  <variable name>
  ...
} <description>
```

# Syntax of interface.ccl cont.

```
[REAL|COMPLEX|INT|POINTER] FUNCTION <function name> (
        [REAL|COMPLEX|INT|STRING|POINTER] \
                [ARRAY] [IN|OUT] <argument name>,
        ...
        )

[USES|REQUIRES] FUNCTION <function name>

PROVIDES FUNCTION <function name>
         WITH <implementation name> LANG [C|FORTRAN]
```

# Example interface.ccl

```
IMPLEMENTS: wavetoy

INHERITS: grid

PUBLIC:

REAL scalarevolve TYPE=gf TIMELEVELS=3
{
  phi
} "The evolved scalar field"
```

# Example interface.ccl cont.

```
CCTK_INT FUNCTION Boundary_SelectVarForBC (
        CCTK_POINTER_TO_CONST IN cctkGH,
        CCTK_INT IN faces,
        CCTK_INT IN boundary_width,
        CCTK_INT IN options_handle,
        CCTK_STRING IN var_name,
        CCTK_STRING IN bc_name
        )

REQUIRES FUNCTION Boundary_SelectVarForBC
```

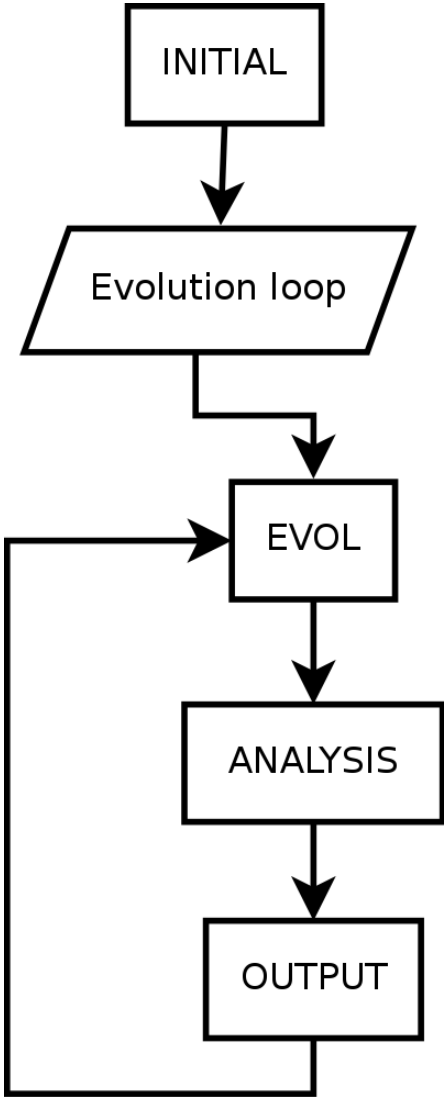
# schedule.ccl

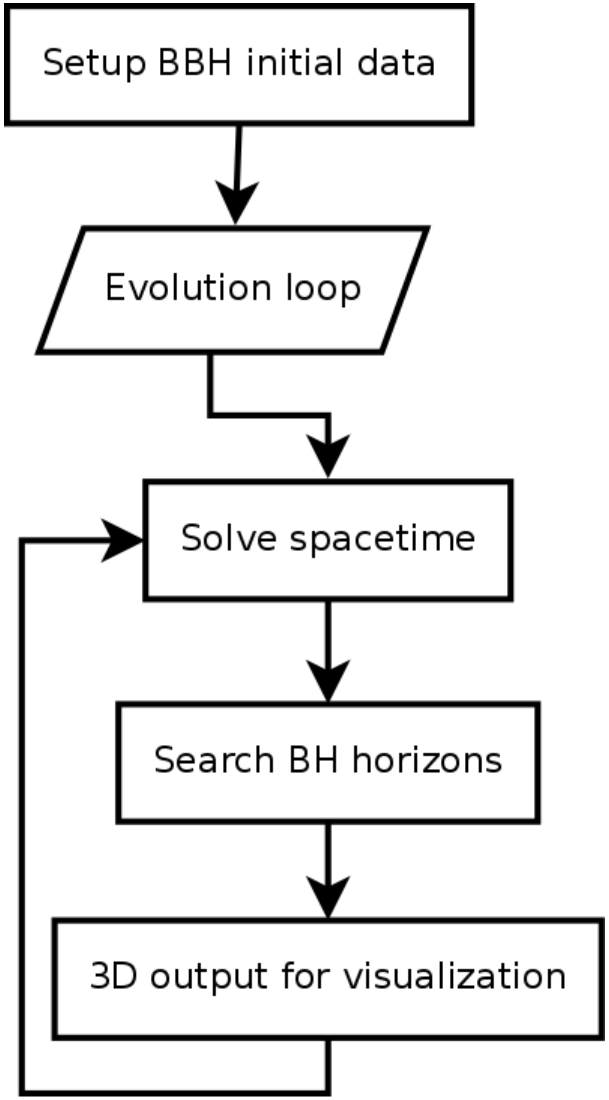
- Flesh contains a flexible rule based scheduler
- Order is prescribed in `schedule.ccl`
- Scheduler also handles when variables are allocated, freed or syncronized between parallel processes
- Functions or groups of functions can be
  - grouped and whole group scheduled
  - scheduled before or after each other
  - scheduled depending on parameters
  - scheduled while some condition is true
- Flesh scheduler sorts all rules and flags an error for inconsistent schedule requests.

# schedule.ccl cont.

- Hierarchy: using schedule groups leads to a *schedule tree*
- Execution order: can schedule `BEFORE` or `AFTER` other items
  e.g.: take time step after calculating RHS
- Loops: can schedule `WHILE` a condition is true
  e.g.: loop while error is too large
- Conditions: can schedule `if` a parameter is set
  e.g.: choose between boundary conditions
- Perform analysis at run time: `TRIGGERS` statements: call routine only if result is needed for I/O

# Example scheduling tree



# Syntax of schedule.ccl

```
SCHEDULE <function name> [AT <schedule bin>|
                          IN <schedule group>]
{
  LANG: [C|Fortran]
  SYNC: <group name> ...
} <description>

SCHEDULE GROUP <name> [AT <schedule bin>|
                       IN <schedule group>]
{
} <description>}

STORAGE: <group name>[timelevels] ...
```

# Example schedule.ccl

```
SCHEDULE WaveToyC_Evolution AT evol
{
  LANG: C
} "Evolution of 3D wave equation"

SCHEDULE GROUP WaveToy_Boundaries AT evol \
               AFTER WaveToyC_Evolution
{
} "Boundaries of 3D wave equation"

STORAGE: scalarevolve[3]
```

# param.ccl

- Definition of parameters
- Scopes: Global, Restricted, Private
- Thorns can use and extened each others parameters
- Different types (double, integer, string, keyword, . . . )
- Range checking and validation
- Steerablility at runtime

# Syntax of param.ccl

```
[SHARES: <implementation>]

[PUBLIC:|RESTRICTED:|PRIVATE:]

[BOOLEAN|KEYWORD|INT|REAL|STRING]
    <parameter name> <description> [STEERABLE=...]
{
    <allowed value>                :: <description>
    <lower bound>:<upper bound> :: <description>
    <pattern>                      :: <description>
    ...
} <default value>
```

# Example param.ccl

```
SHARES: grid
USES KEYWORD type

PRIVATE:
KEYWORD initial_data "Type of initial data"
{
    "plane"    :: "Plane wave"
    "gaussian" :: "Gaussian wave"
} "gaussian"

REAL radius "The radius of the gaussian wave"
{
    0:* :: "Positive"
} 0.0
```

# Hands-on

# Examples

# Hello World, Standalone

Standalone in C:

```
#include <stdio.h>
int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

# Hello World Thorn

- interface.ccl:

```
implements: HelloWorld
```

- schedule.ccl:

```
schedule HelloWorld at CCTK_EVOL
{
  LANG: C
} "Print Hello World message"
```

- param.ccl: empty
- REAME:

```
Cactus Code Thorn HelloWorld
Author(s)    : Frank Löffler <knarf@cct.lsu.edu>
Maintainer(s): Frank Löffler <knarf@cct.lsu.edu>
Licence      : GPL
--------------------------------------------------------------------------

1. Purpose

Example thorn for tutorial Introduction to Cactus
```

# Hello World Thorn cont.

- src/HelloWorld.c:

```
#include "cctk.h"
#include "cctk_Arguments.h"

void HelloWorld(CCTK_ARGUMENTS)
{
  DECLARE_CCTK_ARGUMENTS
  CCTK_INFO("Hello World!");
  return;
}
```

- make.code.defn:

```
SRCS = HelloWorld.c
```

# Hello World Thorn

- parameter file:

```
ActiveThorns = "HelloWorld"
Cactus::cctk_itlast = 10
```

- run: `[mpirun] <cactus executable> <parameter file>`

# Hello World Thorn

- Screen output:

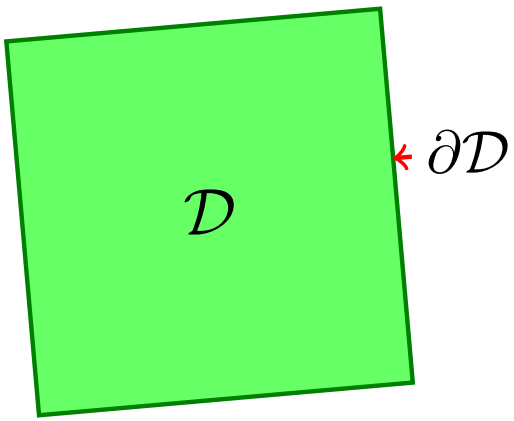
```
      10
  1   0101       ************************
  01  1010 10      The Cactus Code V4.0
 1010 1101 011      www.cactuscode.org
  1001 100101    ************************
    00010101
     100011     (c) Copyright The Authors
      0100      GNU Licensed. No Warranty
      0101

Cactus version:    4.0.b17
Compile date:      May 06 2009 (13:15:01)
Run date:          May 06 2009 (13:15:54-0500)
[...]

Activating thorn Cactus...Success -> active implementation Cactus
Activation requested for
--->HelloWorld<---
Activating thorn HelloWorld...Success -> active implementation HelloWorld
--------------------------------------------------------------------------------
INFO (HelloWorld): Hello World!
INFO (HelloWorld): Hello World!
[...] 8x
--------------------------------------------------------------------------------
Done.
```

# WaveToy Thorn: Wave Equation

For a given source function $S(x, y, z, t)$ find a scalar wave field $\varphi(x, y, z, t)$ inside the domain $\mathcal{D}$ with a boundary condition:




- inside $\mathcal{D}$:

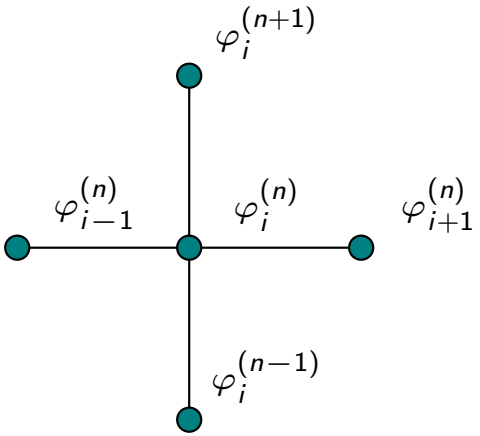
$$\frac{\partial^2 \varphi}{\partial t^2} = c^2 \Delta \varphi + S$$

- on the boundary $\partial\mathcal{D}$:

$$\varphi|_{\partial\mathcal{D}} = \varphi(t = 0)$$

# WaveToy Thorn: Discretization

Discretization:
approximating continuous function $\varphi(x,t)$ with a grid function $\varphi_i^{(n)}$:




$$\frac{\partial^2\varphi}{\partial t^2} = c^2(\partial_x^2\varphi) + S$$

$$\Downarrow (c \equiv 1)$$

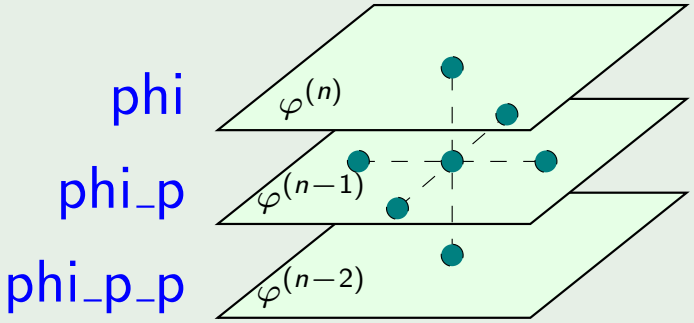
$$\frac{\varphi_i^{(n+1)} - 2\varphi_i^{(n)} + \varphi_i^{(n-1)}}{2\Delta t^2} = \frac{\varphi_{i+1}^{(n)} - 2\varphi_i^{(n)} + \varphi_{i-1}^{(n)}}{2\Delta x^2} + S_i^{(n)}$$

# WaveToy Thorn

Thorn structure:

## interface.ccl

- grid function phi[3]:




- Boundary_SelectVarForBC

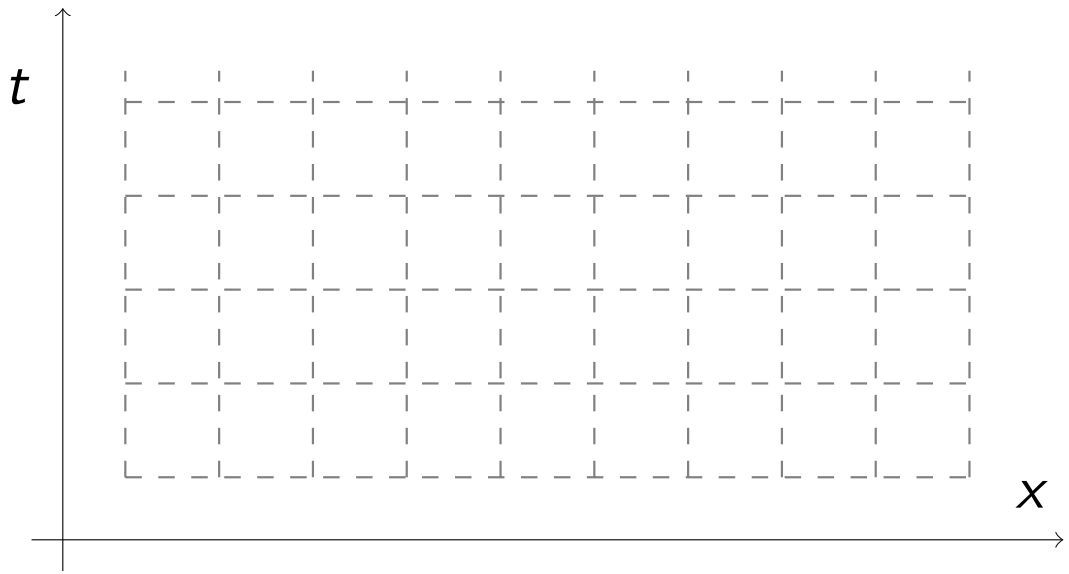
## param.ccl

- Parameters of initial Gaussian pulse: amplitude $A$, radius $R$, width $\sigma$

## schedule.ccl

- WaveToyC_InitialData
- WaveToyC_Evolution
- WaveToyC_Boundaries

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

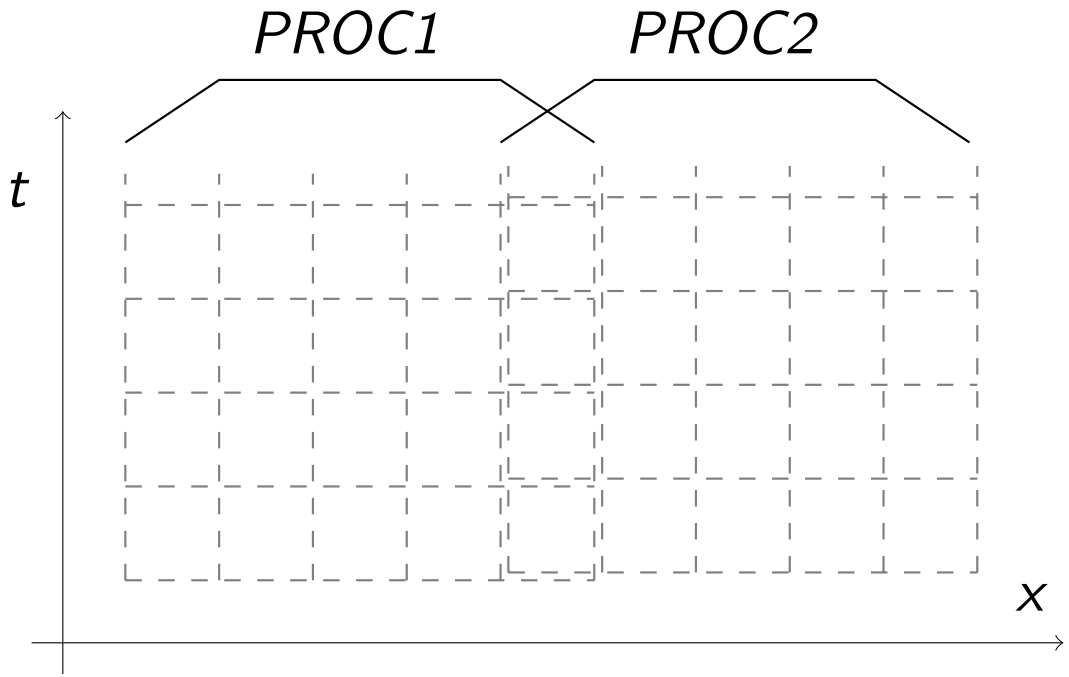
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

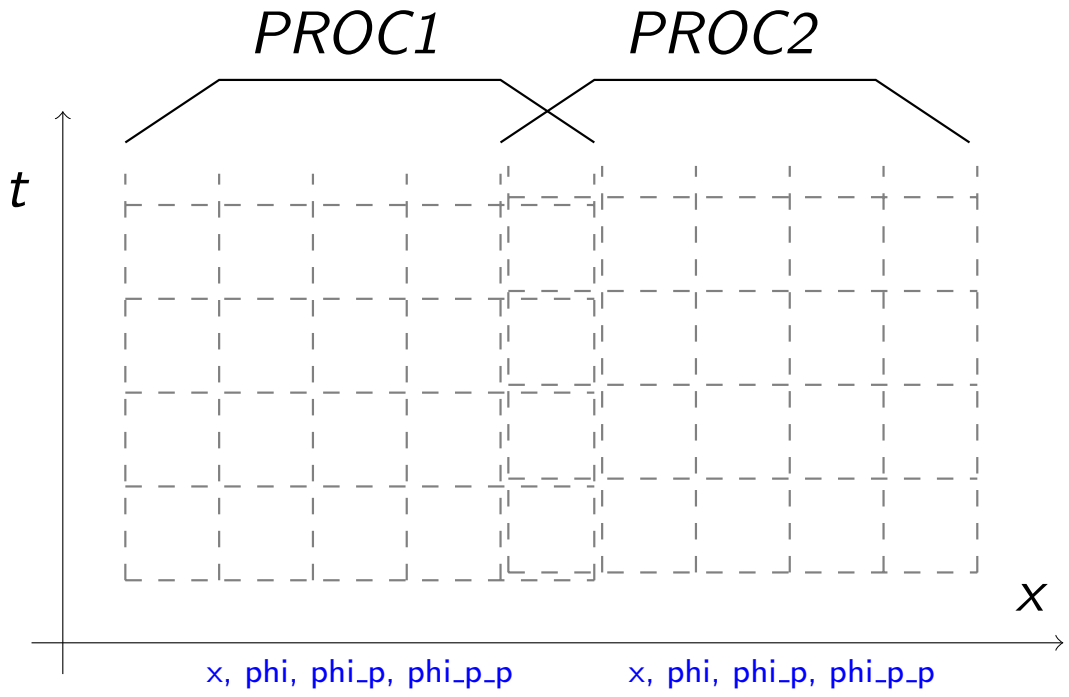
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

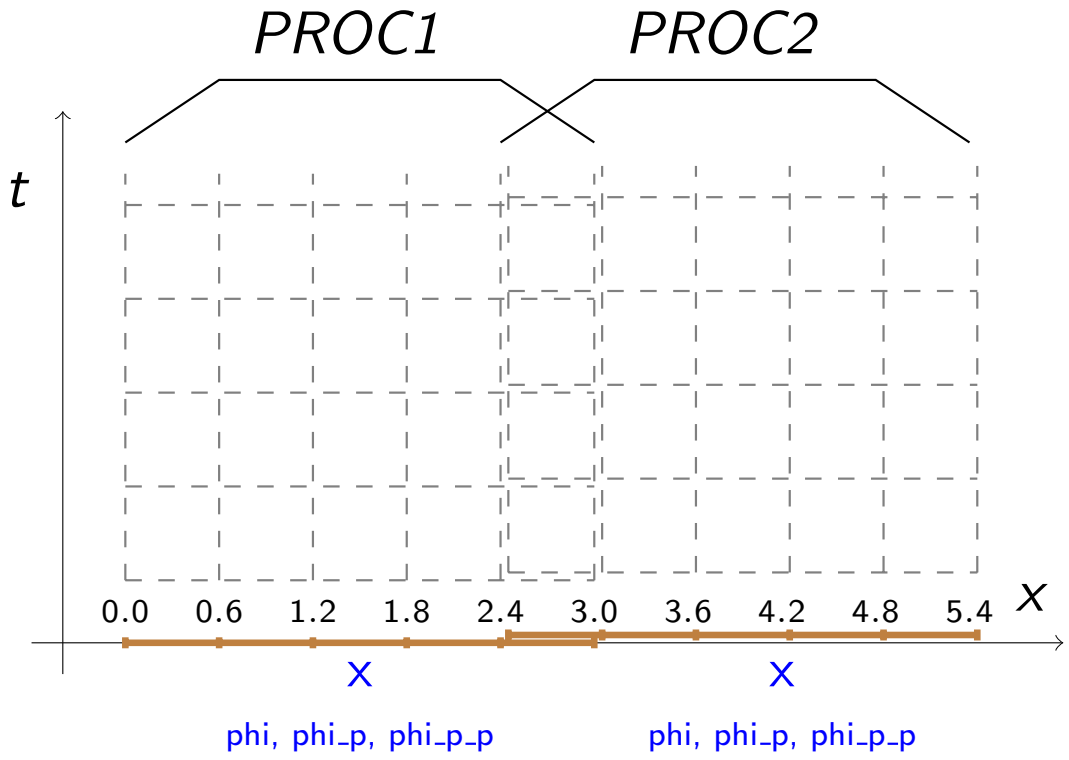
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

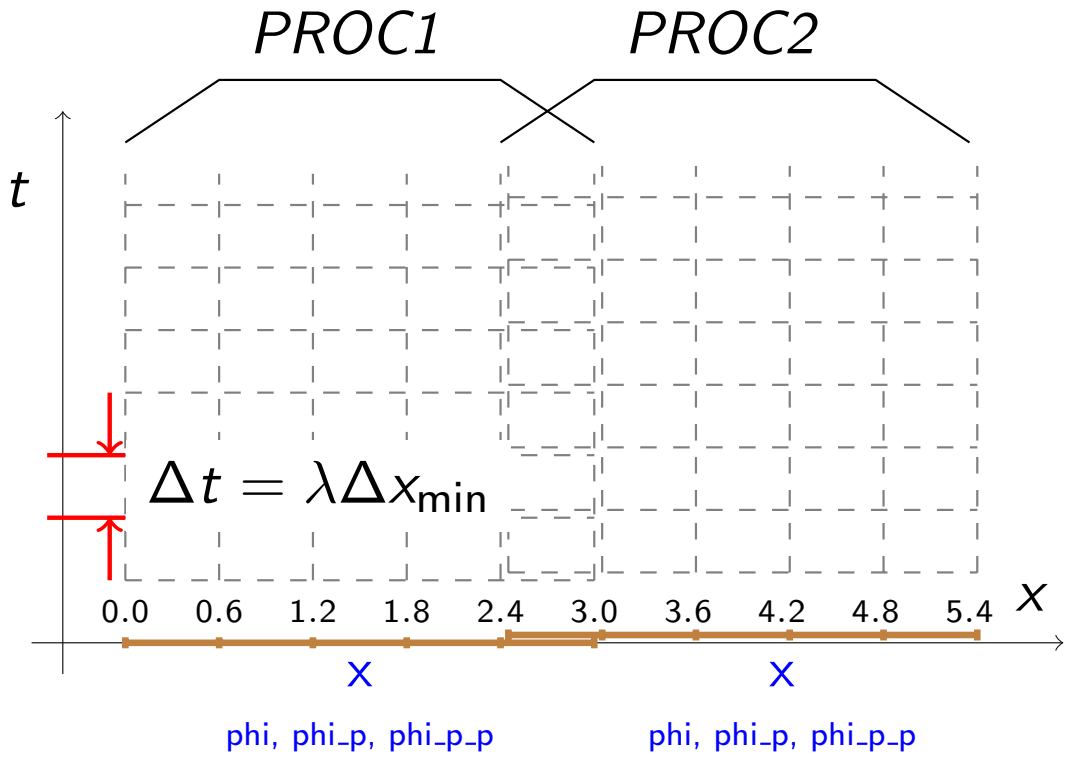
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

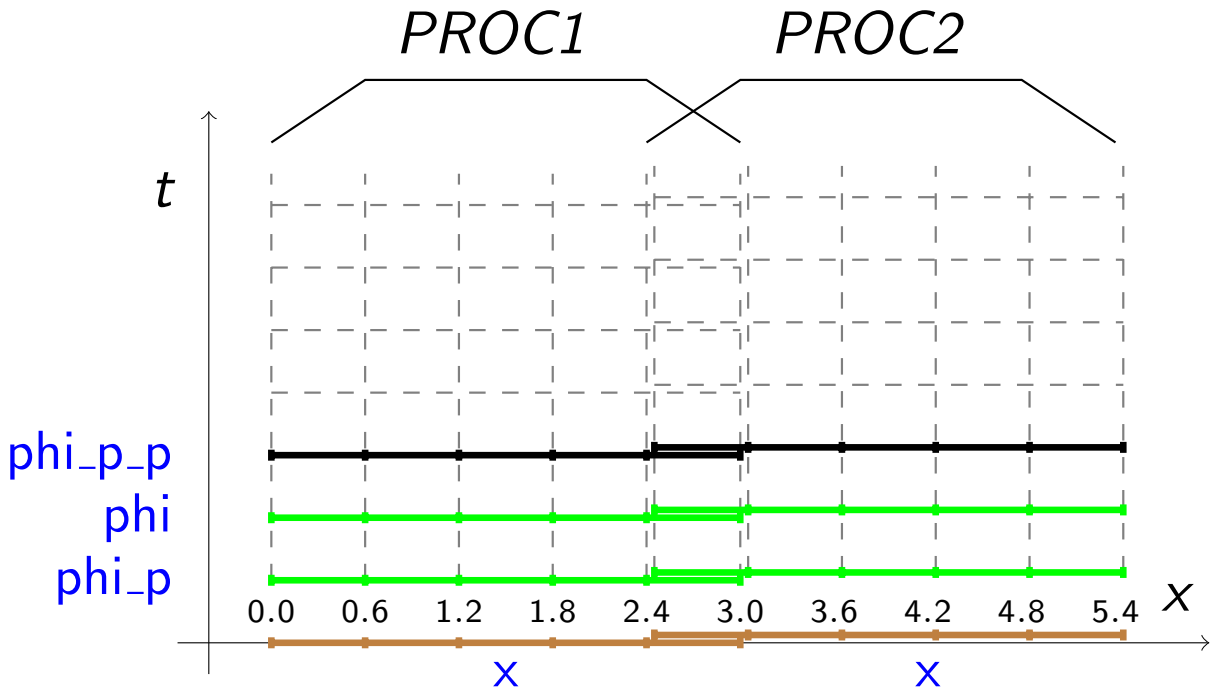
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

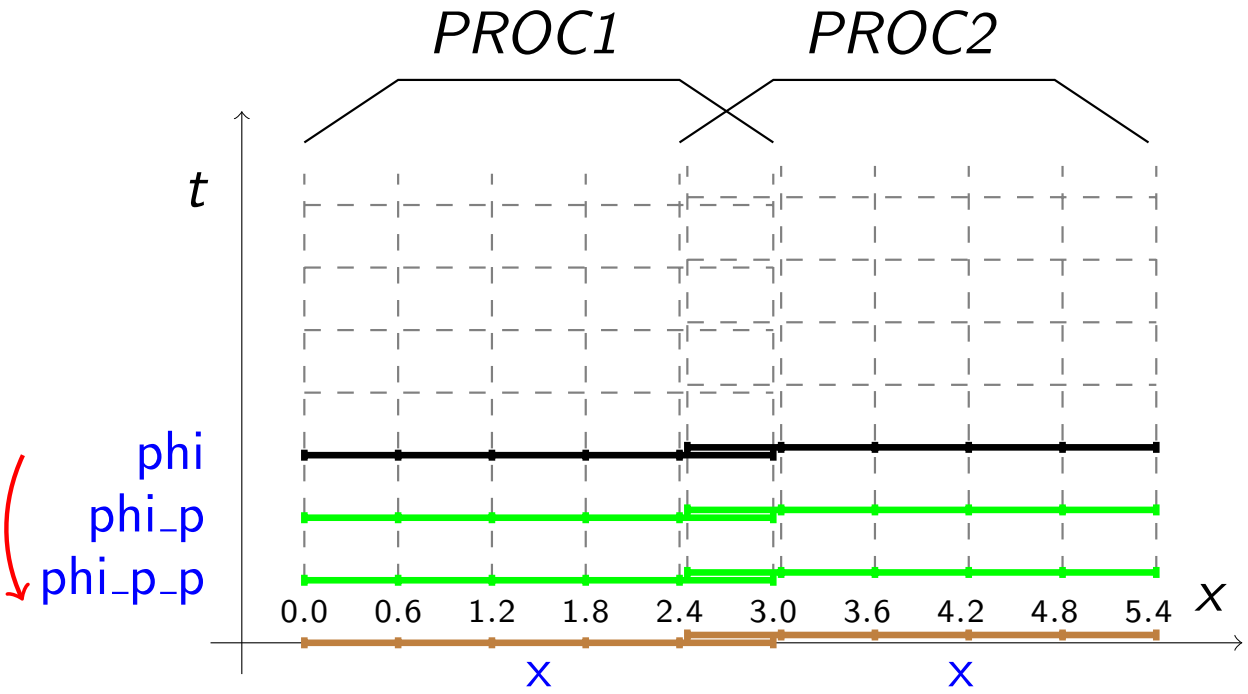
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

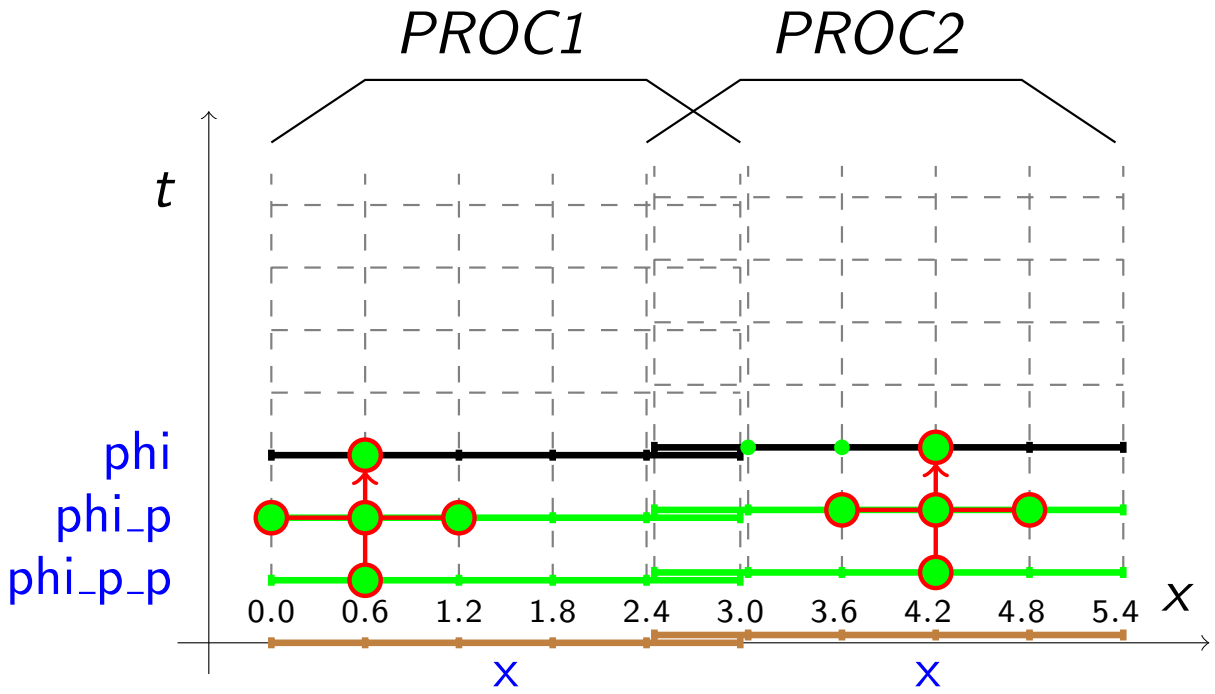
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

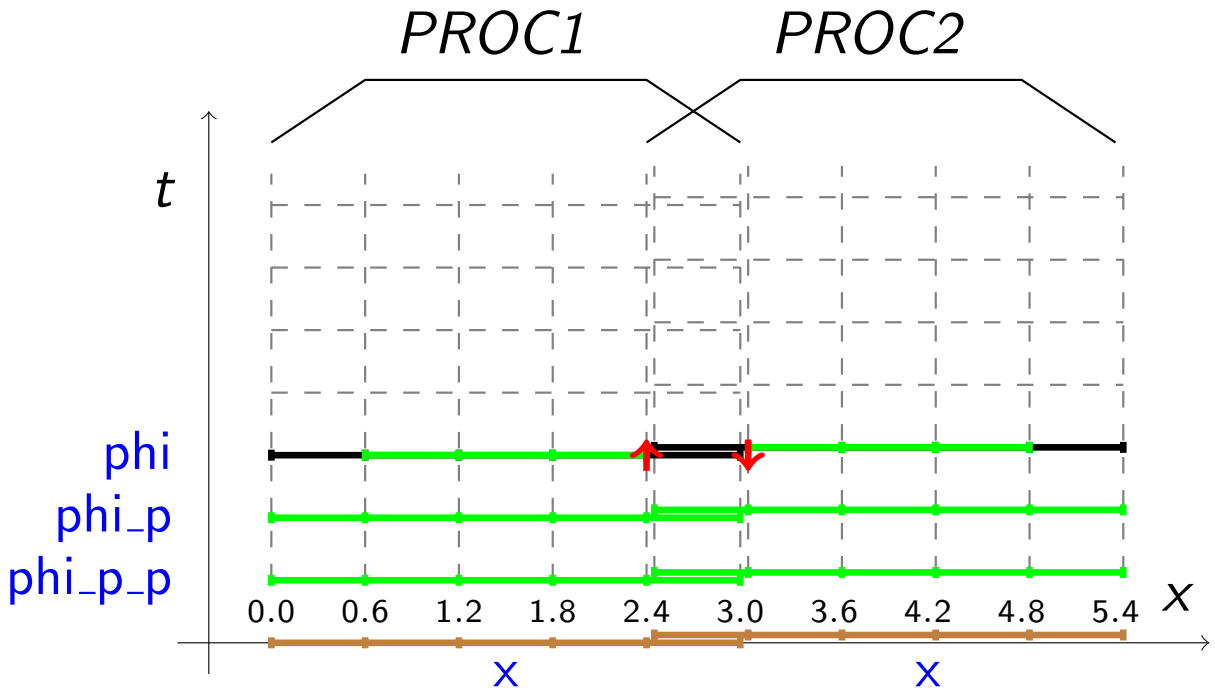
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

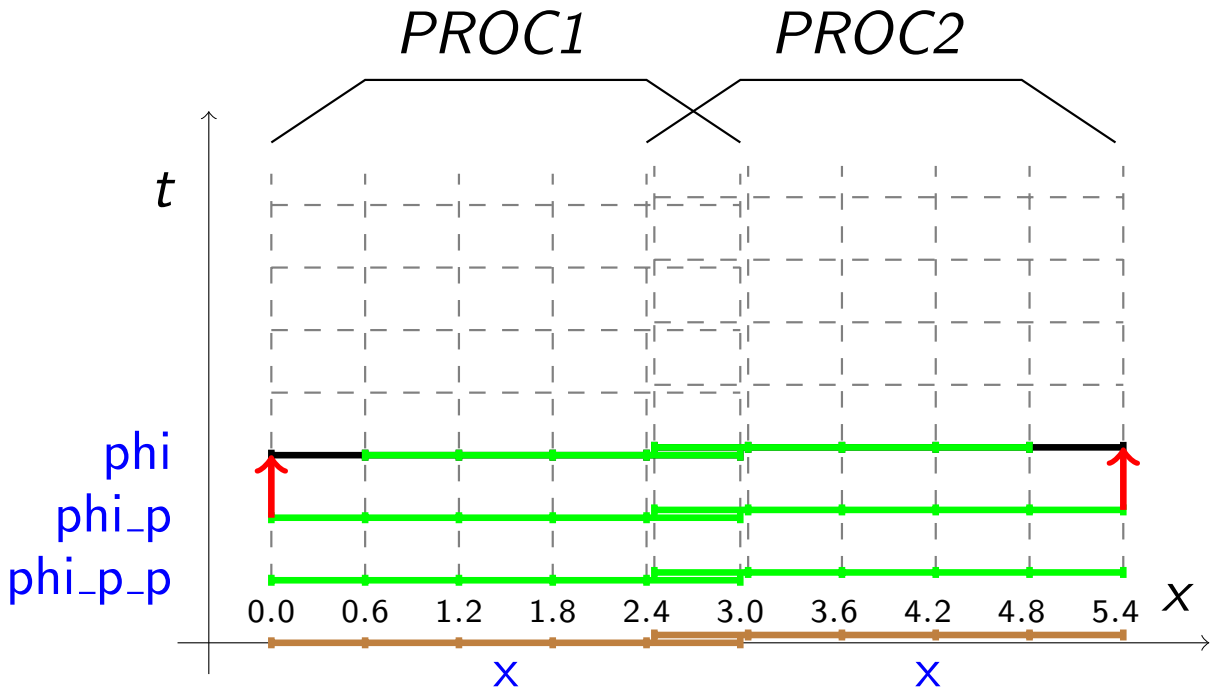
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

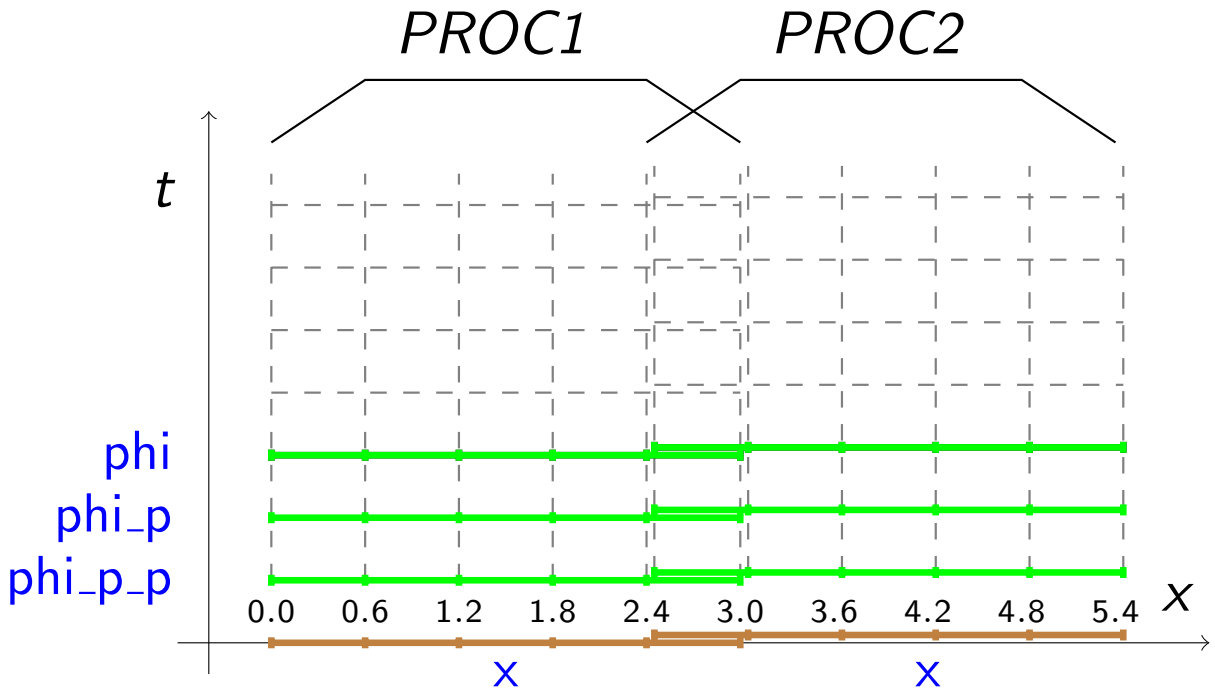
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

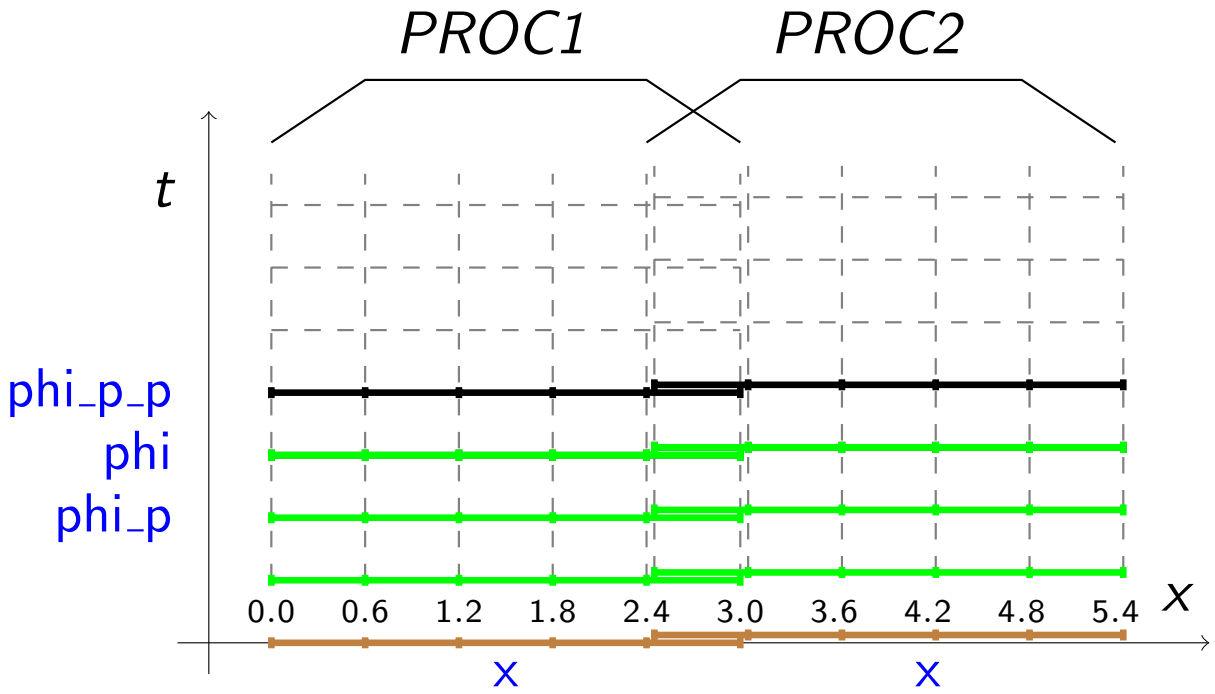
↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn: Algorithm Illustration




Grid structure
GF allocation

↓

Set up coords
Compute $\Delta t$
Initial data

↓

Rotate timelevels
Evolve GF, sync
Apply BCs
Output data

# WaveToy Thorn

Directory structure:

```
WaveToy/
|-- COPYRIGHT
|-- README
|-- configuration.ccl
|-- doc
|   `-- documentation.tex
|-- interface.ccl
|-- schedule.ccl
|-- param.ccl
`-- src
    |-- WaveToy.c
    `-- make.code.defn
```

# WaveToy Thorn

Directory structure:

```
WaveToy/
|-- COPYRIGHT
|-- README
|-- configuration.ccl
|-- doc
|    `-- documentation.tex
|-- interface.ccl
|-- schedule.ccl
|-- param.ccl
`-- src
     |-- WaveToy.c
     `-- make.code.defn
```

# WaveToy Thorn

- `interface.ccl`:

```
IMPLEMENTS: wavetoy_simple
INHERITS: grid

PUBLIC:

CCTK_REAL scalarevolve TYPE=gf TIMELEVELS=3
{
  phi
} "The evolved scalar field"

CCTK_INT FUNCTION Boundary_SelectVarForBC( \
  CCTK_POINTER_TO_CONST IN GH, CCTK_INT IN faces, \
  CCTK_INT IN boundary_width, CCTK_INT IN table_handle, \
  CCTK_STRING IN var_name, CCTK_STRING IN bc_name)

REQUIRES FUNCTION Boundary_SelectVarForBC
```

# WaveToy Thorn cont.

- `schedule.ccl`:

```
STORAGE: scalarevolve[3]

SCHEDULE WaveToy_InitialData AT CCTK_INITIAL
{
  LANG: C
} "Initial data for 3D wave equation"

SCHEDULE WaveToy_Evolution AT CCTK_EVOL
{
  LANG: C
  SYNC: scalarevolve
} "Evolution of 3D wave equation"

SCHEDULE WaveToy_Boundaries AT CCTK_EVOL AFTER WaveToy_Evolution
{
  LANG: C
} "Select boundary conditions for the evolved scalar"

SCHEDULE GROUP ApplyBCs as WaveToy_ApplyBCs AT CCTK_EVOL AFTER WaveToy_Boundaries
{
} "Apply boundary conditions"
```

# WaveToy Thorn cont.

- param.ccl:

```
CCTK_REAL amplitude "The amplitude of the waves"
{
 *:* :: "Anything"
} 1.0

CCTK_REAL radius "The radius of the gaussian wave"
{
 0:* :: "Positive"
} 0.0

CCTK_REAL sigma "The sigma for the gaussian wave"
{
 0:* :: "Positive"
} 0.1
```

# WaveToy Thorn cont.

- Example parameter file:

```
Cactus::cctk_run_title = "Simple WaveToy"

ActiveThorns = "time boundary Carpet CarpetLib CartGrid3D"
ActiveThorns = "CoordBase ioutil CarpetIOBasic CarpetIOASCII"
ActiveThorns = "CarpetIOHDF5 SymBase wavetoy"

cactus::cctk_itlast = 10000
time::dtfac = 0.5

IO::out_dir              = $parfile
IOBasic::outInfo_every   = 1
IOASCII::out1D_every     = 1
IOASCII::out1D_vars      = "wavetoy_simple::phi"

iohdf5::out_every = 10
iohdf5::out_vars         = "grid::coordinates{out_every=10000000} wavetoy_simple::phi"
```

# Summary

- Cactus is a powerful framework for developing portable applications, particularly suited to large collaboration.
- Cactus is currently used by many groups around the world, in several fields.
- For more information: `http://www.cactuscode.org`, Users' guide (available online, also distributed with Cactus)

# Thanks

Thanks goes to

- TeraGrid for hosting the tutorial
- LONI for providing support for the hands-on part on QueenBee
- LSU for support at CCT
- Cactus developers...

# Hands-on

Hands - On

# Hands-on overview

You will

- Download, configure and build Cactus executable
- Run a small Cactus application with WaveToy example parameter file using...
  - the queueing system on QueenBee
  - the WaveToy thorn of the Examples section
  - other thorns like HTTPD, IOJpeg, Twitter
- Run a larger Cactus application using multiple nodes of QueenBee
- Run WaveBinary simulation to create more interesting data
- Visualize these data remotely using VisIt

# WaveToy Thorn – Hands-on part

- Login to `queenbee.loni.org` using your `trainXX` account
- Download `GetCactus` script

```
wget http://www.cactuscode.org/download/GetCactus
chmod 755 GetCactus
```

- Get tutorial example files, e.g. a thornlist

```
svn co https://svn.cct.lsu.edu/repos/cactus/tutorials/
    introduction/examples/Cactus/misc
```

- Get Cactus flesh and thorns

```
./GetCactus -repository=development ./misc/Thorns.th
```

- Build executable

```
cd Cactus
make wavetoy-config THORNLIST=../misc/Thorns.th
    options=../misc/queenbee.config
make wavetoy
```

# WaveToy Thorn – Hands-on part

- Submit job to queue

```
qsub misc/queenbee.qsub
```

- Status of your job in queue

```
qstat -a | grep trainXX
```

- Delete (abort) job before usual end

```
qdel <ID>
```

- Output directory:

```
ls simulations/WaveToy
```

- Simulation browser URL announced on Twitter:

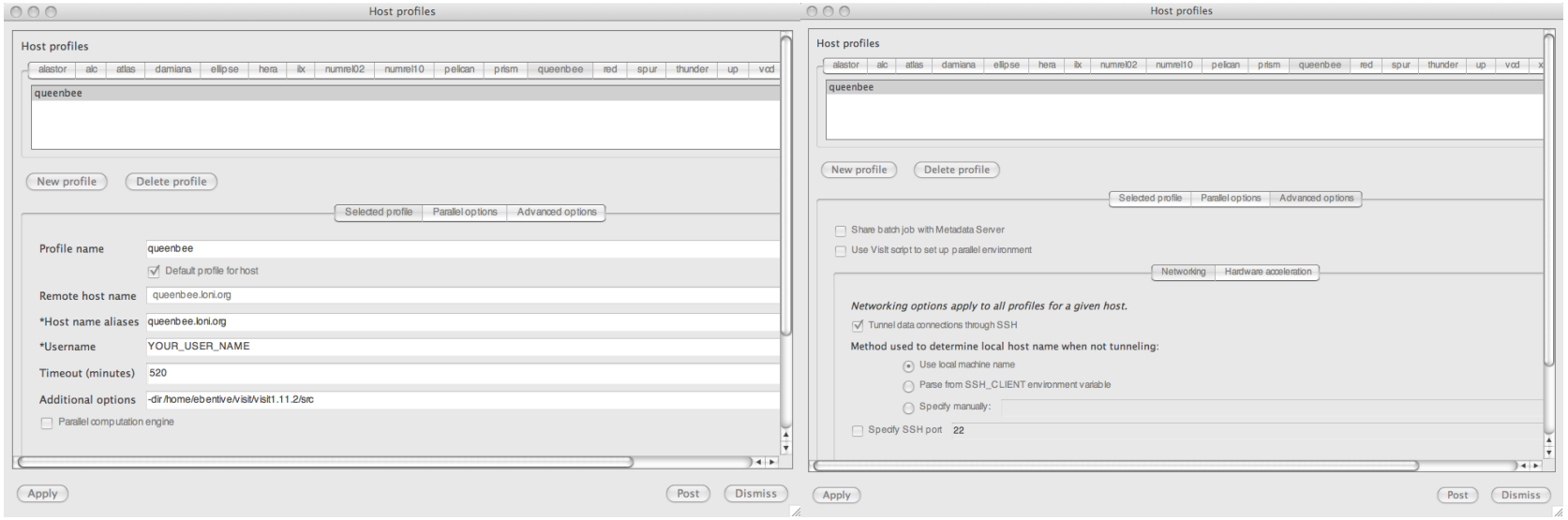
```
http://twitter.com/numrel
```

- To run other parameter file: copy & edit `queenbee.qsub`

# WaveToy Thorn – Hands-on part

**On your laptop:**

- Launch version 1.11.x of VisIt
- Create a host profile for queenbee:




- Open data files under `QueenBee:simulations/WaveToy/`

# The End

The End

# What is a framework?

A framework is

- a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact

# What is a framework?

A framework is

- a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact
- the skeleton of an application that can be customized by an application developer

# What is a framework?

A framework is

- a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact
- the skeleton of an application that can be customized by an application developer
- an architecture, implementation and documentation that captures the intended use of the framework for building applications

# What is a framework?

A framework is

- a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact
- the skeleton of an application that can be customized by an application developer
- an architecture, implementation and documentation that captures the intended use of the framework for building applications
- community building