NERSC

BERKELEY LAB

Lawrence Berkeley National Laboratory / National Energy Research Supercomputing Center

Frameworks for Complex Multiphysics HPC Applications

CS267 – Spring 2011

John Shalf

With contributions from: *Gabrielle Allen, Tom Goodale, Eric Schnetter, Ed Seidel (AEI/LSU)*

Phil Colella, Brian Van Straalen (ANAG/LBNL)

April 12, 2011

Office of Science
U.S. DEPARTMENT OF ENERGY

Application Code Complexity

- **Application Complexity has Grown**
  - Big Science on leading-edge HPC systems is a multi-disciplinary, multi-institutional, multi-national efforts! *(and we are not just talking about particle accelerators and Tokamaks)*
  - Looking more like science on atom-smashers

- **Advanced Parallel Languages are Necessary, but NOT Sufficient!**
  - Need higher-level organizing constructs for teams of programmers
  - Languages must work together with frameworks for a complete solution!

Application Code Complexity

Large-Scale Electronic Structure Calculations of High-Z Metals on the BlueGene/L Platform

Francois Gygi
Department of Applied Science
University of California, Davis
Davis, CA 95616
530-752-4042
fgygi@ucdavis.edu

Erik W. Draeger, Martin Schulz,
Bronis R. de Supinski
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551
{draeger1,schulz6,bronis}@llnl.gov

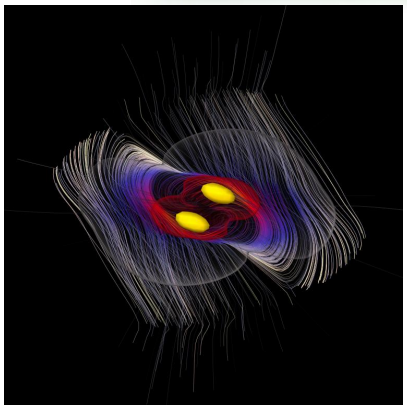
John A.Gunnels, Vernon Austel,
James C. Sexton
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
{gunnels,austel,sextonjc}@us.ibm.com

Franz Franchetti
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
franzf@ece.cmu.edu

Stefan Kral, Christoph W. Ueberhuber, Juergen Lorenz
Institute of Analysis and Scientific Computing
Vienna University of Technology, Vienna, Austria
skral@mips.complang.tuwien.ac.at, c.ueberhuber@tuwien.ac.at
juergen.lorenz@aurora.anum.tuwien.ac.at

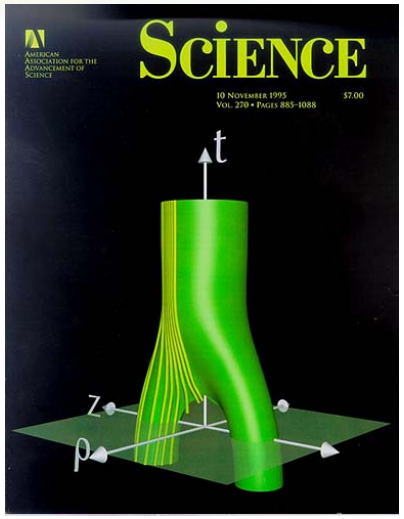
- **HPC is looking more and more like traditional "big science" experiments.**
- **QBox: Gordon Bell Paper title page**
  - Its just like particle physics papers!
  - *Looks like discovery of the Top Quark!*

Example: Grand Challenge Simulation Science



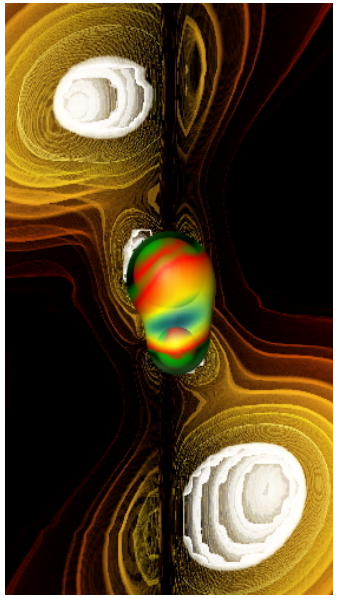
NASA Neutron Star Grand Challenge

- 5 US Institutions
- Towards colliding neutron stars



NSF Black Hole Grand Challenge

- 8 US Institutions, 5 years
- Towards colliding black holes



Gamma Ray Busts Core Collapse Supernova

- 10 countries x 12 institutions x 5 years
- *Multiple disciplines*
  - *GR*
  - *Hydro*
  - *Chemistry*
  - *Radiation Transp*
  - *Analytic Topology*

Examples of Future of Science & Engineering

- ***Require Large Scale Simulations, at edge of largest computing sys***
- ***Complex multi-physics codes with millions of lines of codes***
- ***Require Large Geo-Distributed Cross-Disciplinary Collaborations***

Community Codes & Frameworks

(hiding complexity using good software engineering)

- **Clearly separate roles and responsibilities of your expert programmers from that of the domain experts/scientist/users (productivity layer vs. performance layer)**
- **Define a *social* contract between the expert programmers and the domain scientists**
- **Enforces and facilitates SW engineering style/discipline to ensure correctness**
- **Hides complex domain-specific parallel abstractions from scientist/users to enable performance (hence, most effective when applied to community codes)**
- **Allow scientists/users to code nominally serial plug-ins *(C++ or Fortran)* that are invoked by a parallel "driver" *(either as DAG or constraint-based scheduler)* to enable productivity**

Framework: Developer Expertise

Developer Roles	Domain Expertise	CS/Coding Expertise	Hardware Expertise
***Application**: Assemble solver modules to solve science problems. (eg. combine hydro+GR +elliptic solver w/MPI driver for Neutron Star simulation)*	Einstein *(expert)*	Elvis *(can dance)*	Mort *(novice)*
***Solver**: Write solver modules to implement algorithms. Solvers use driver layer to implement "idiom for parallelism". (e.g. an elliptic solver or hydrodynamics solver)*	Elvis	Einstein	Elvis
***Driver**: Write low-level data allocation/placement, communication and scheduling to implement "idiom for parallelism" for a given "dwarf". (e.g. PUGH)*	Mort	Elvis	Einstein

User/Developer Roles

Developer Roles	Conceptual Model	Instantiation
***Application**: Assemble solver modules to solve science problems.*	Neutron Star Simulation: Hydrodynamics + GR Solver using Adaptive Mesh Refinement (AMR)	BSSN GR Solver + MoL integrator + Valencia Hydro + Carpet AMR Driver + Parameter file (params for NS)
***Solver**: Write solver modules to implement algorithms. Solvers use driver layer to implement "idiom for parallelism".*	Elliptic Solver	PETSC Elliptic Solver pkg. (in C) BAM Elliptic Solver (C++ & F90) John Town's custom BiCG-Stab implementation (in F77)
***Driver**: Write low-level data allocation/placement, communication and scheduling to implement "idiom for parallelism" for a given "dwarf".*	Parallel boundary exchange idiom for structured grid applications	Carpet AMR Driver SAMRAI AMR Driver GrACE AMR driver PUGH (MPI unigrid driver) SHMUGH (SMP unigrid driver)

Enabling Collaborative Development!

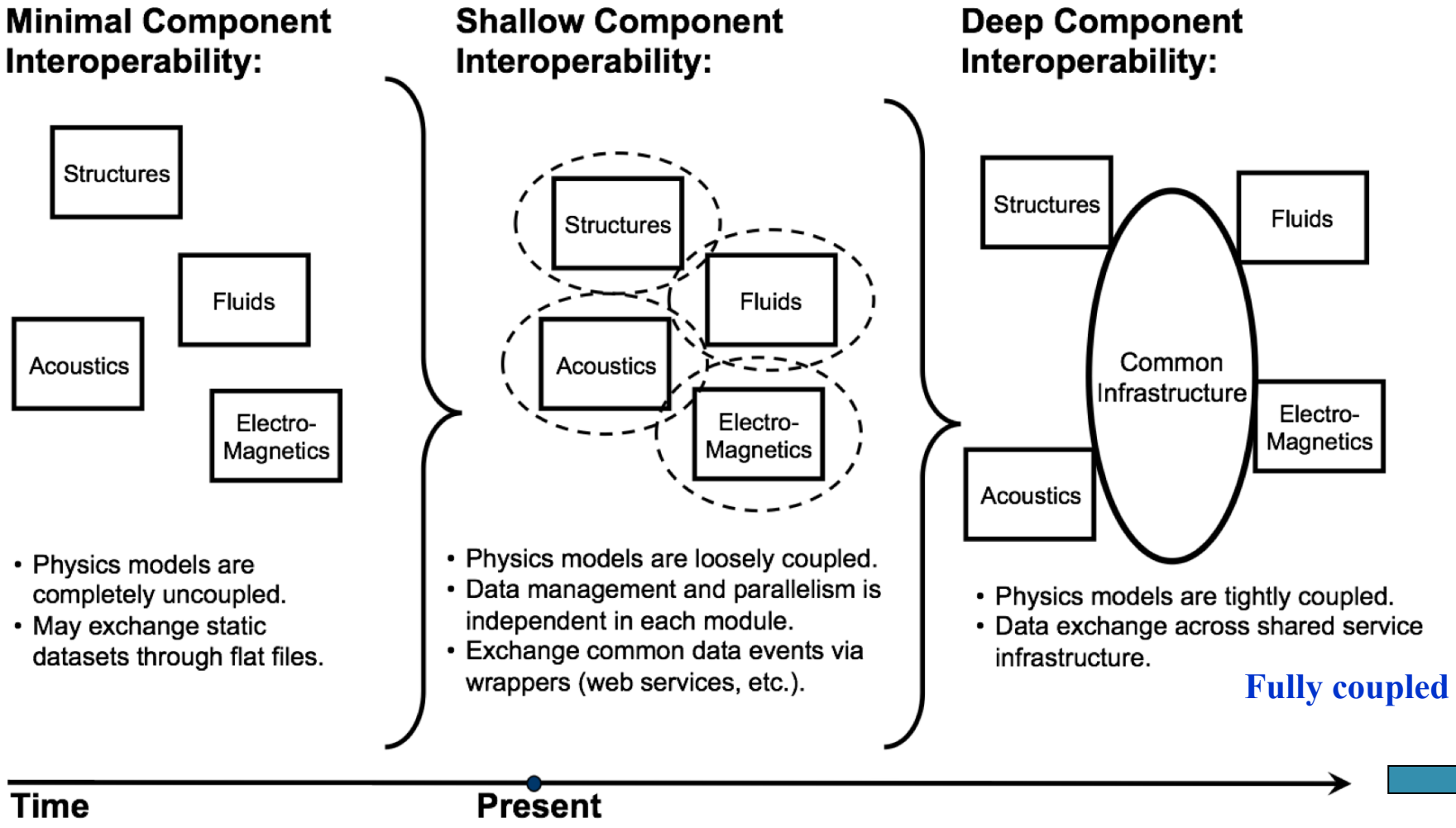
- **They enable computer scientists and computational scientists to play nicely together**
  - No more arguments about C++ vs. Fortran
  - Easy **unit-testing** to reduce finger pointing (are the CS weenies "tainting the numerics") (also good to accelerate V&V)
  - Enables multidisciplinary collaboration (domain scientists + computer jocks) to enables features that would not otherwise emerge in their own codes!
    - *Scientists write code that seem to never use "new" features*
    - *Computer jocks write code that no reasonable scientist would use*
- **Advanced CS Features are trivially accessible by Application Scientists**
  - Just list the name of the module and it is available
  - Also trivially unit-testable to make sure they don't change numerics
- **Also enables sharing of physics modules among computational scientists**
  - The hardest part is agreeing upon physics interfaces (there is no magic!)
  - *Nice, but not actually not as important as the other benefits (organizing large teams of programmers along the lines of their expertise is the*

Framework vs. Libraries

- **Library**
  - User program invokes library *(imperative execution model offers limited scheduling freedom)*
  - User defines presents data layout to library *(compiler and system has limited freedom to reorganize to match physical topology of underlying system hardware)*

- **Framework**
  - Framework invokes user plug-in *(declarative execution model)*
  - Only operation on data given *(well defined scope for side-effects.*
  - ***Functional semantics provide more scheduling freedom***

Framework Taxonomy



- Integration is invasive: *how much will you put up with?*

Observations on Domain-Specific Frameworks

- **Frameworks and domain-specific languages**
  - enforce coding conventions for big software teams
  - Encapsulate a domain-specific “idiom for parallelism”
  - Create familiar semantics for domain experts (more productive)
  - *Clear separation of concerns (separate implementation from specification)*

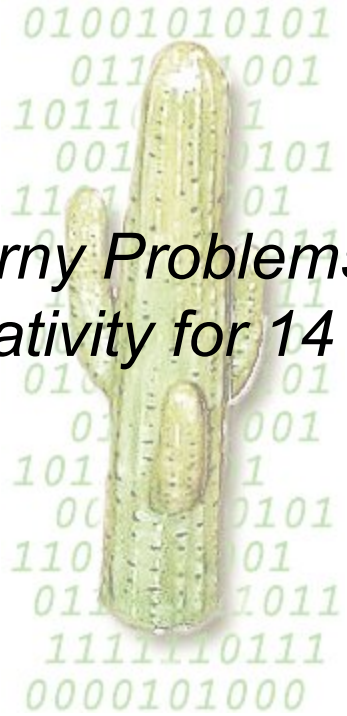
- **Common design principles for frameworks from SIAM CSE07 and DARPA Ogden frameworks meeting**
  - Give up main(): *schedule controlled by framework*
  - Stateless: *Plug-ins only operate on state passed-in when invoked*
  - Bounded (or well-understood) side-effects: *Plug-ins promise to restrict memory touched to that passed to it (same as CILK)*

NERSC

BERKELEY LAB

Lawrence Berkeley National Laboratory / National Energy Research Supercomputing Center

Examples: CACTUS



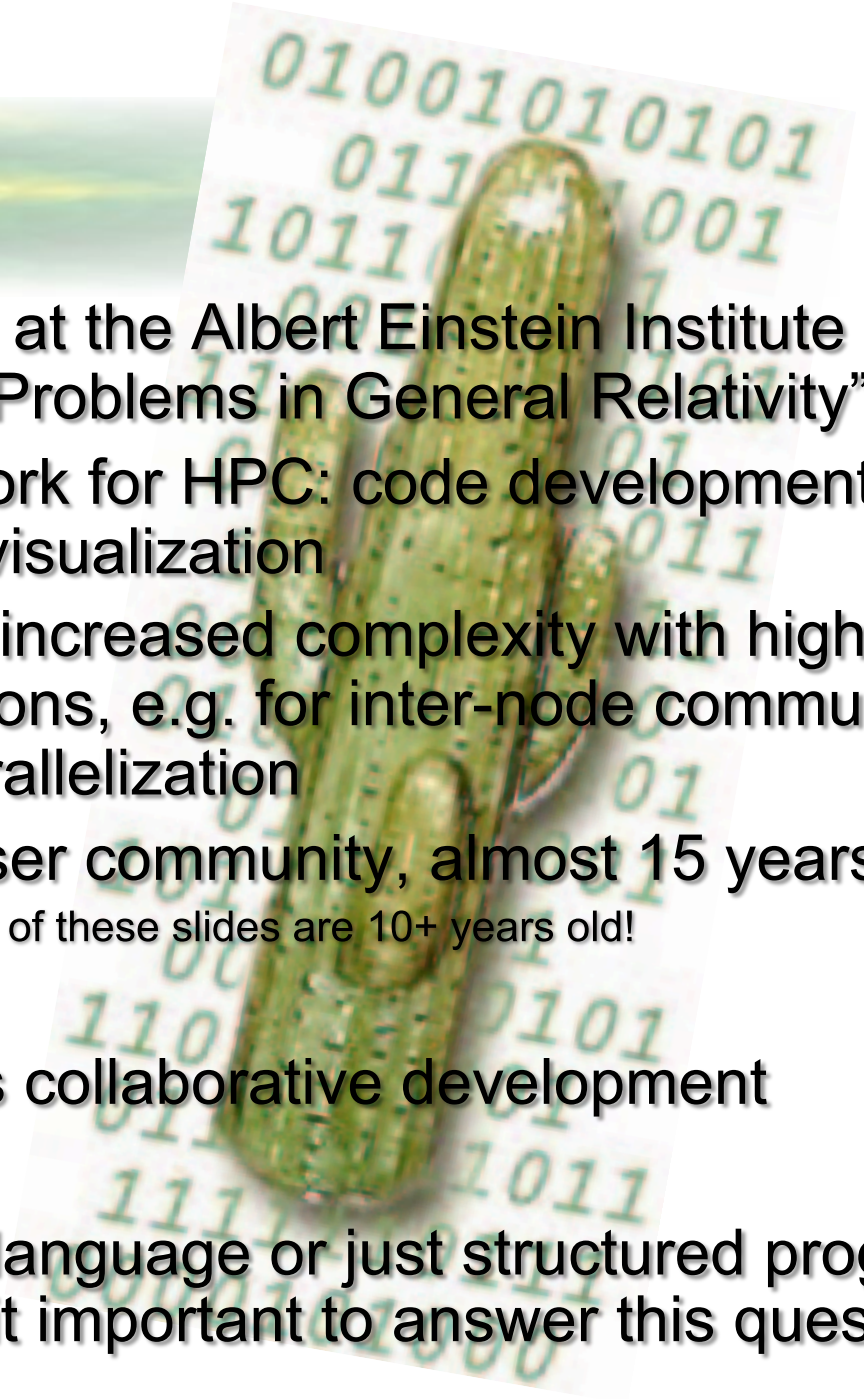
Solving Thorny Problems in Numerical Relativity for 14 years!

Office of Science
U.S. DEPARTMENT OF ENERGY

www.CactusCode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Cactus



- Invented at the Albert Einstein Institute in 1997 to solve "Thorny Problems in General Relativity"
- Framework for HPC: code development, simulation control, visualization
- Manage increased complexity with higher level abstractions, e.g. for inter-node communication, intra-node parallelization
- Active user community, almost 15 years old
  - Many of these slides are 10+ years old!
- Supports collaborative development
- Is this a language or just structured programming? (Why is it important to answer this question?)

Cactus User Community

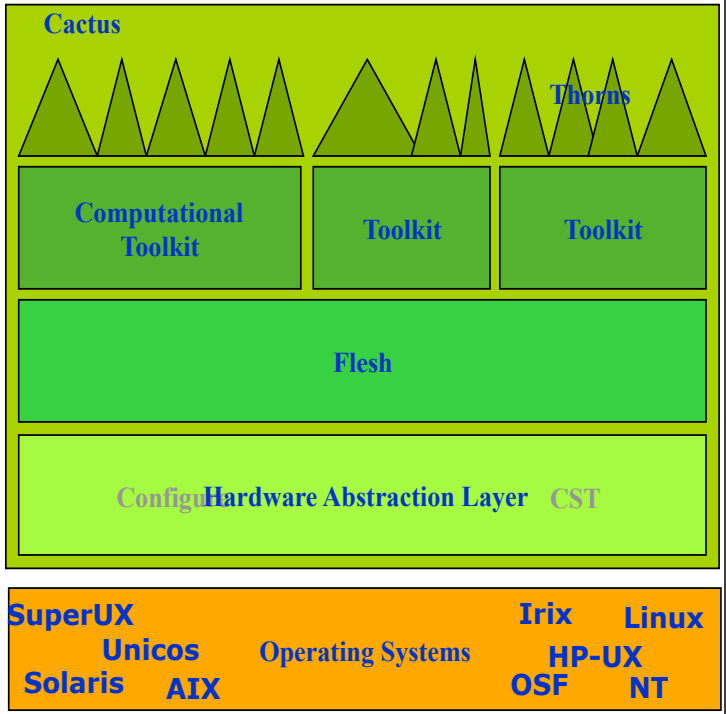
- **General Relativity**
  - LSU(USA),AEI(Germany),UNAM (Mexico), Tuebingen(Germany), Southampton (UK), Sissa(Italy), Valencia (Spain), University of Thessaloniki (Greece), MPA (Germany), RIKEN (Japan), TAT(Denmark), Penn State (USA), University of Texas at Austin (USA), University of Texas at Brwosville (USA), WashU (USA), University of Pittsburg (USA), University of Arizona (USA), Washburn (USA), UIB (Spain), University of Maryland (USA), Monash (Australia)
- **Astrophysics**
  - Zeus-MP MHD ported to Cactus (Mike Norman: NCSA/UCSD)
- **Computational Fluid Dynamics**
  - KISTI
  - DLR: (turbine design)
- **Chemistry**
  - University of Oklahoma: (Chem reaction vessels)
- **Bioinformatics**
  - Chicago

Cactus Features

- **Scalable Model of Computation**
  - Cactus provides ‘idiom’ for parallelism
    - Idiom for Cactus is parallel boundary exchange for block structured grids
    - Algorithm developers provide nominally “serial” plug-ins
    - Algorithm developers are shielded from complexity of parallel implementation
  - Neuron uses similar approach for scalable parallel idiom
- **Build System**
  - User does not see makefiles *(just provides a list of source files in a given module)*
  - “known architectures” used to store accumulated wisdom for multi-platform builds
  - Write once and run everywhere (laptop, desktop, clusters, petaflop HPC)
- **Modular Application Composition System**
  - This is a system for composing algorithm and service components together into a complex composite application
  - Just provide a list of “modules” and they self-organize according to constraints *(less tedious than explicit workflow)*
  - *Enables unit testing for V&V of complex multiphysics applications*
- **Language Neutrality**
  - Write modules in any language *(C, C++, F77, F90, Java, etc…)*
  - Automatically generates bindings (also hidden from user)
  - Overcomes age-old religious battles about programming languages

Cactus Architecture and Terminology

- **Flesh:** ***Services (the glue) that ties everything together***
  - Supports composition of modules into applications
  - Invokes modules in correct order (baseline scheduling)
  - Implements code build system (get rid of makefiles)
  - Implements parameter file parsing
  - Generates bindings for any language (Fortran, C++, Java)
- **Driver:** ***Implements idiom for parallelism***
  - Implements "dwarf-specific" composite datatypes
  - Handles data allocation, placement, domain decomposition
  - Implements communication pattern for "idiom for parallelism"
  - Implements thread-creation and scheduling for parallelism
- **Thorn:** ***Modular Component implementing a solver or service***
  - Can be written in any language (bindings auto-generated)
  - Implementation of parallelism externalized, so developer writes nominally serial code with correct idiom. Parallelism handled by the "driver".
  - Thorns implementing same functionality derived from same 'abstract class' of functionality such as "*elliptic solver*" (can have many implementations of *elliptic solve* that can be selected at compile time or at runtime)
- **Hardware Abstraction Layer: Eliminate "makefile" and *make differences between systems go away***

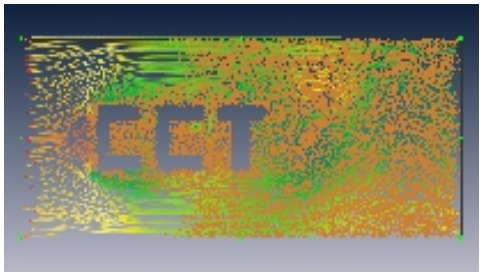


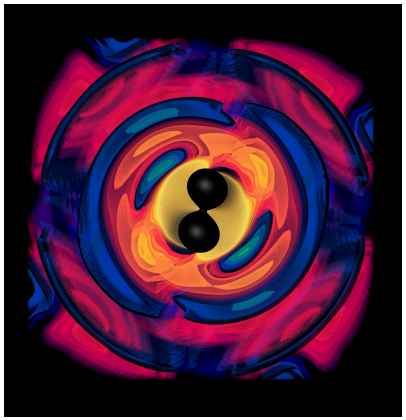
Benefits

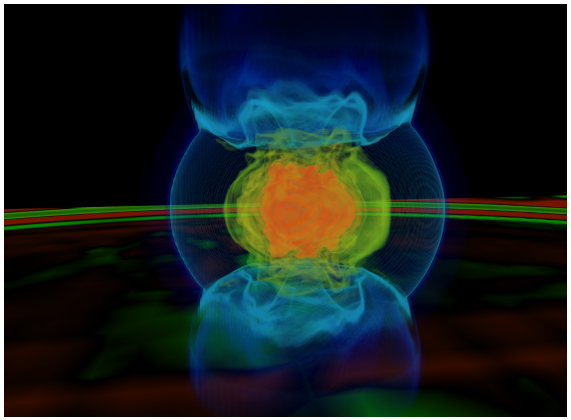
- **Other "frameworks" that use same organizing principles (and similar motivation)**
  - NEURON (parallel implementation of Genesis neurodyn)
  - SIERRA (finite elements/structural mechanics)
  - UPIC and TechX (generized code frameworks for PIC codes)
  - Chombo: AMR on block-structured grids (its hard)
  - Common feature is that computational model is well understood and broadly used (seems to be a good feature for workhorse "languages")
- **Common benefits (and motivations) are**
  - **Modularity** (composition using higher-level semantics)
  - **Segmenting expertise**
  - **Unit Testing:** This was the biggest benefit
  - **Performance analysis** (with data aggregated on reasonable semantic boundaries)
  - **Correctness testing** (on reasonable semantic boundaries)
  - **Enables reuse of "solver" components**, but can replace "driver" if you have a different hardware platform.

Cactus in the Real World

- Numerical Relativity (Black Holes)
- Relativistic Astrophysics (Gamma Ray Bursts)
- CFD (Toolkit)



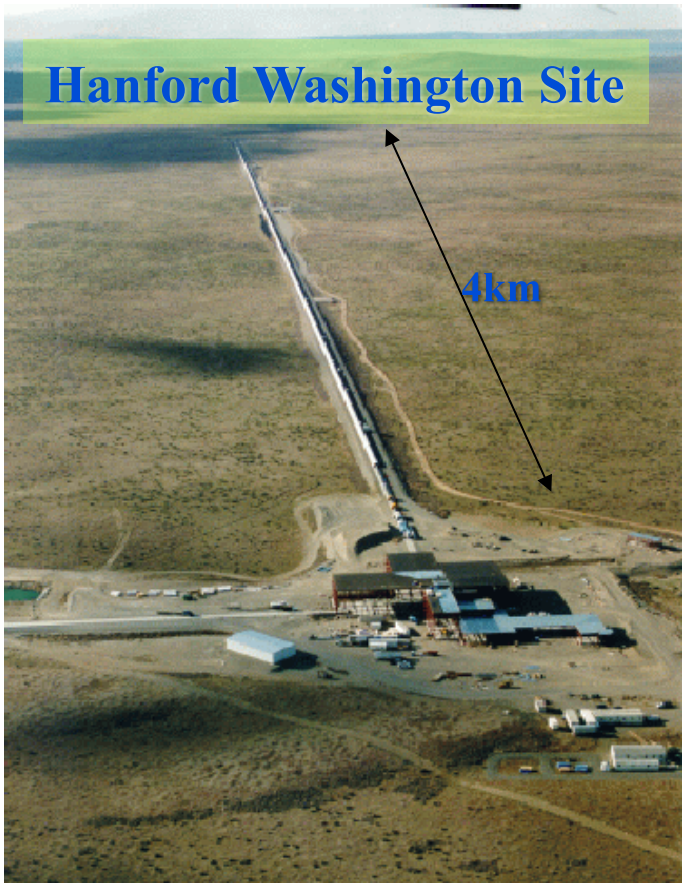


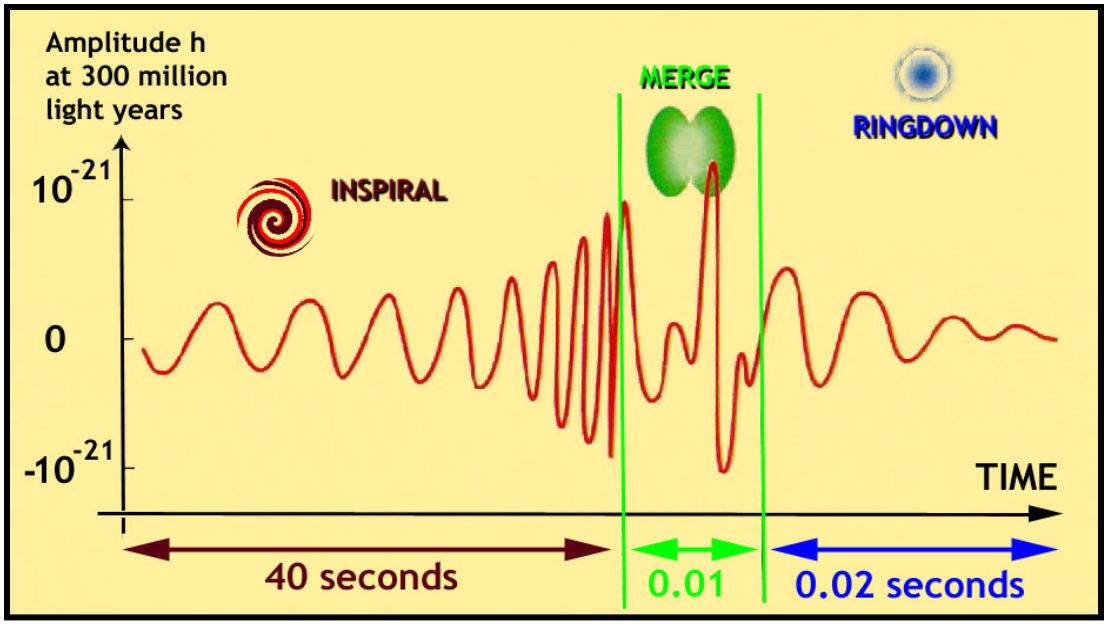


Detecting Gravitational Waves

Will uncover fundamentally new information about the universe

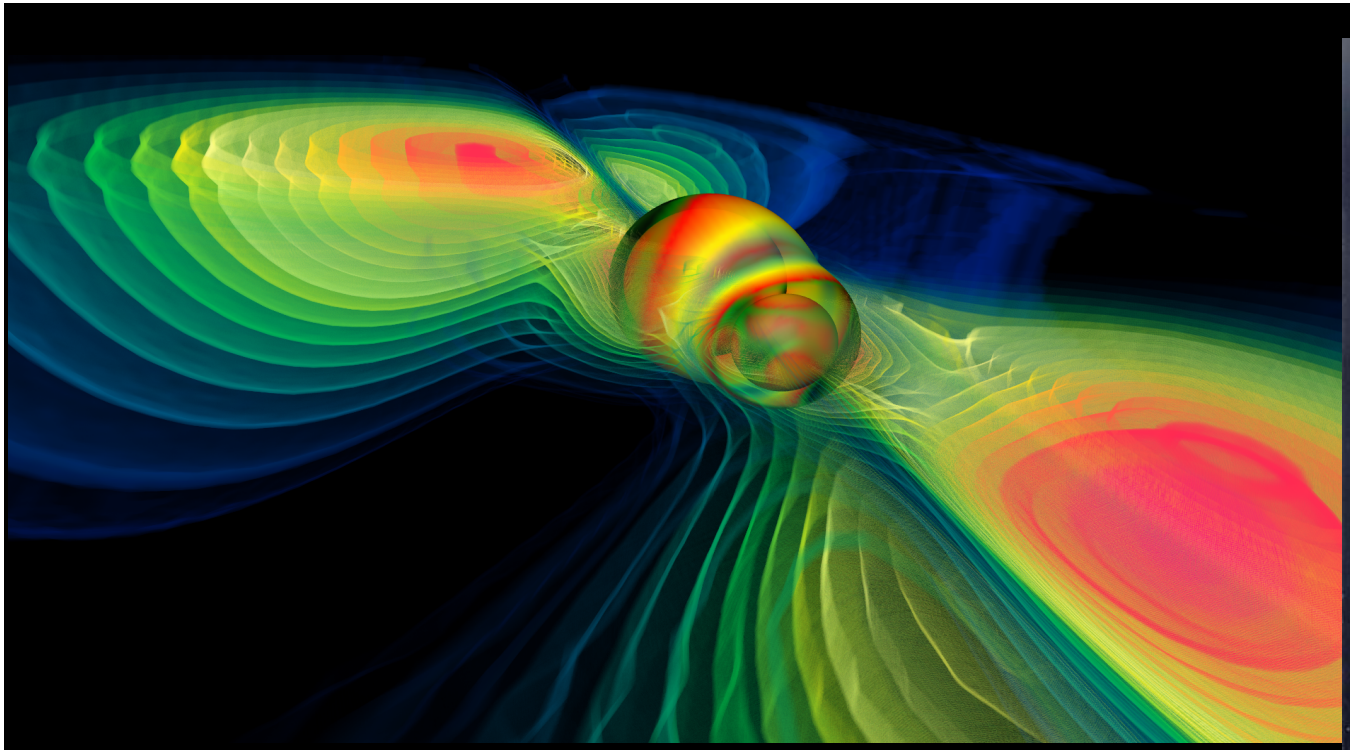
- **LIGO, VIRGO (Pisa), GEO600,… $1 Billion Worldwide**
- **Was Einstein right? 5-10 years, we'll see!**

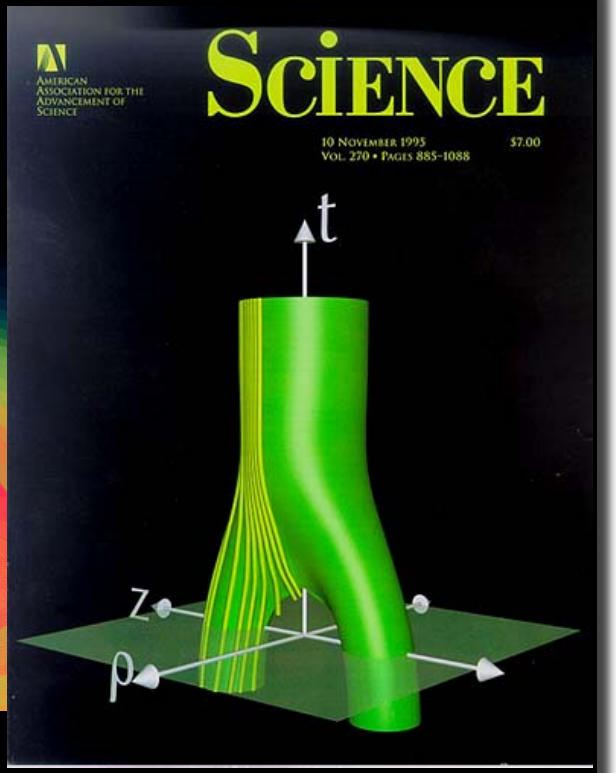


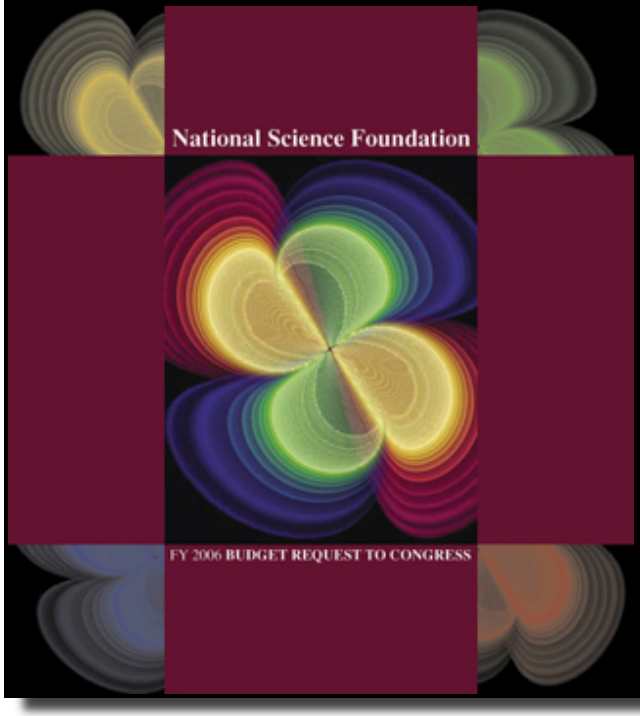


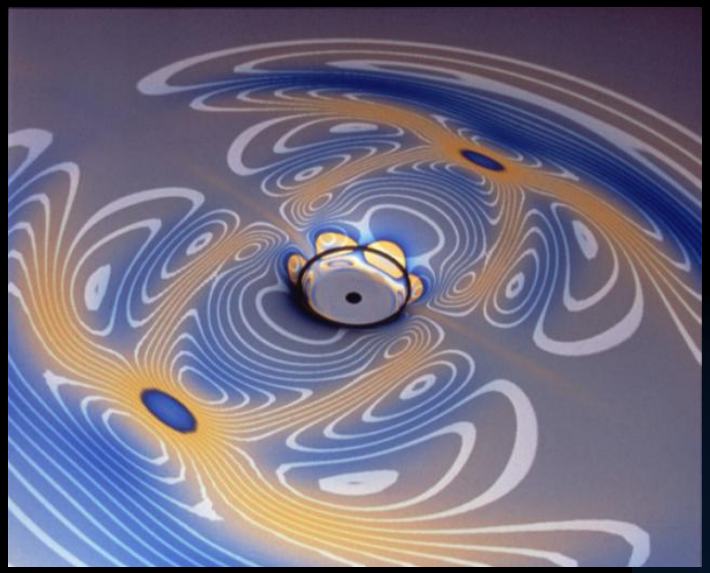
GR requires solution of dozens of coupled, nonlinear hyperbolic-elliptic equations with 1000's of terms (barely have the capability to solve after a century of development)

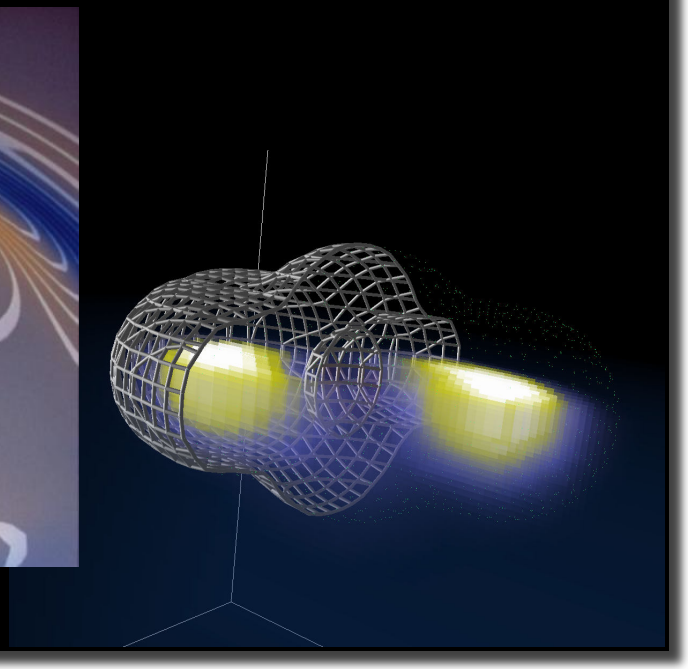
- **Detect GR Waves…pattern matching against numerical templates to enhance signal/noise ratio**
- **Understand them…just what are the waves telling us?**




American Association for the Advancement of Science
SCIENCE
10 November 1995
Vol. 270 • Pages 885–1088
$7.00
t
z
ρ


National Science Foundation
FY 2006 BUDGET REQUEST TO CONGRESS





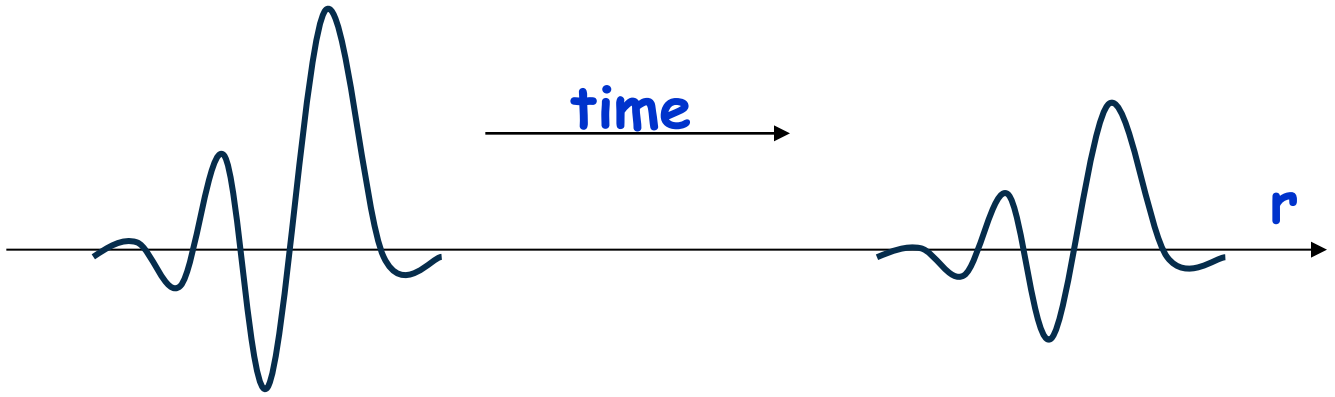
How Does Cactus Work?

Primer on PDE Solvers on Block Structured Grids

Scalar Wave Model Problem

Scalar waves in 3D are solutions of the hyperbolic wave equation: $-\phi_{,tt} + \phi_{,xx} + \phi_{,yy} + \phi_{,zz} = 0$

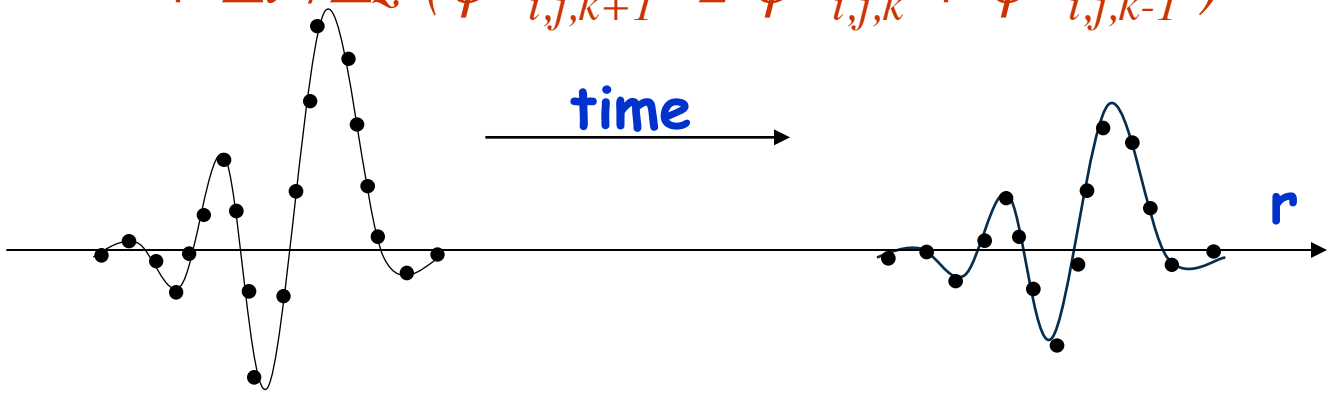
Initial value problem: given data for ϕ and its first time derivative at initial time, the wave equation says how it evolves with time



Numerical Method

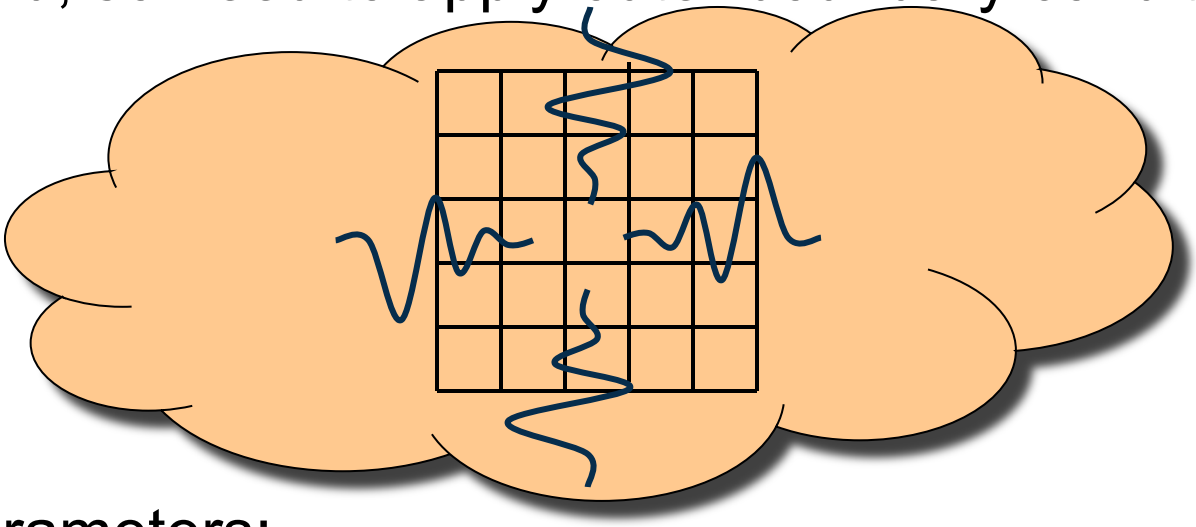
Numerical solve by discretising on a grid, using explicit *finite differencing* (centered, second order)

$$\phi^{n+1}_{i,j,k} = 2\phi^{n}_{i,j,k} - \phi^{n-1}_{i,j,k}$$
$$+ \Delta t^2/\Delta x^2(\phi^{n}_{i+1,j,k} - 2\,\phi^{n}_{i,j,k} + \phi^{n}_{i-1,j,k})$$
$$+ \Delta t^2/\Delta y^2(\phi^{n}_{i,j+1,k} - 2\,\phi^{n}_{i,j,k} + \phi^{n}_{i,j-1,k})$$
$$+ \Delta t^2/\Delta z^2(\phi^{n}_{i,j,k+1} - 2\,\phi^{n}_{i,j,k} + \phi^{n}_{i,j,k-1})$$



Numerical Method

- Finite grid, so need to apply outer boundary conditions



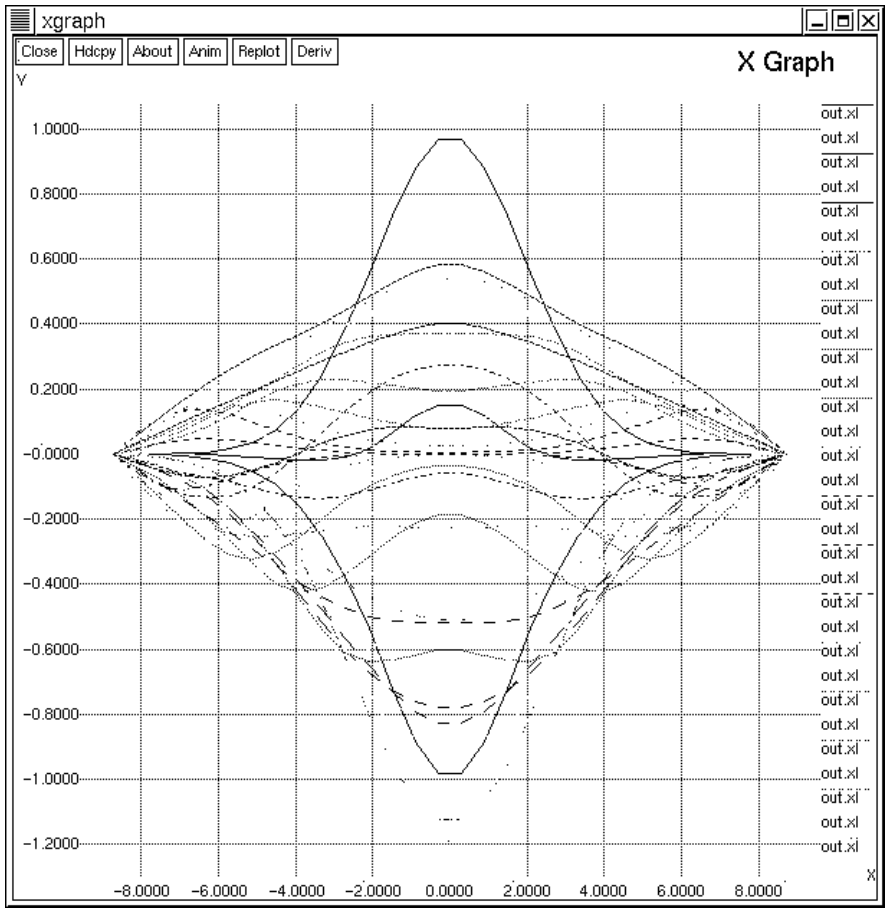
- Main parameters:
  - grid spacings: Δt, Δx, Δy, Δz, which coords?, which initial data?
- Simple problem, analytic solutions, but contains many features needed for modelling more complex problems

Example Stand Alone Code: Main.f

```
c     ================================
      program WaveToy
c     ================================
c     Fortran 77 program for 3D wave equation.
c     Explicit finite difference method.
c     ================================

c     Global variables in include file
      include "WaveToy.h"
      integer i,j,k

c     SET UP PARAMETERS
      nx = 30
      [MORE PARAMETERS]

c     SET UP COORDINATE SYSTEM AND GRID
      x_origin = (0.5 - nx/2)*dx
      y_origin = (0.5 - ny/2)*dy
      z_origin = (0.5 - nz/2)*dz

      do I=1,nx
        do j=1,ny
          do k=1,nz
            x(i,j,k) = dx*(i-1) + x_origin
            y(i,j,k) = dy*(j-1) + y_origin
            z(i,j,k) = dz*(k-1) + z_origin
           r(i,j,k) = sqrt(x(i,j,k)**2+y(i,j,k)**2+z(i,j,k)**2)
          end do
        end do
      end do

c     OPEN OUTPUT FILES
      open(unit=11,file="out.xl")
      open(unit=12,file="out.yl")
      open(unit=13,file="out.zl")

c     SET UP INITIAL DATA
      call InitialData
      call Output

c     EVOLVING
      do iteration = 1, nt
        call Evolve
        if (mod(iteration,10).eq.0) call Output
      end do

      stop
      end
```

Standalone Serial Program

Setting up parameters
Setting up grid and coordinate system
Opening output files
Setting up initial data
Performing iteration 10
Performing iteration 20
Performing iteration 30
Performing iteration 40
Performing iteration 50
Performing iteration 60
Performing iteration 70
Performing iteration 80
Performing iteration 90
Performing iteration 100
Done



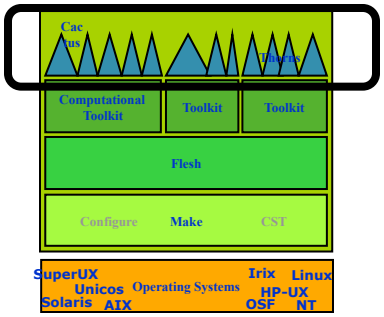
Making a "Thorn" *(a Cactus Module)*

```
c     ================================
      program WaveToy
c     ================================
c     Fortran 77 program for 3D wave equation
c     Explicit finite difference method.
c     ================================

c     Global variables in include file
      include "WaveToy.h"
      integer i,j,k


c     SET UP P
      nx = 30


c     SET UP
      x_or
      y_
      z_origin = (0.5 - nz
```

```
do I=1,nx
  do j=1,ny
    do k=1,nz
      x(i,j,k) = dx*(i-1) + x_origin
      y(i,j,k) = dy*(j-1) + y_origin
      z(i,j,k) = dz*(k-1) + z_origin
     r(i,j,k) = sqrt(x(i,j,k)**2+y(i,j,k)**2+z(i,j,k)**2)
    end do
  end do
end do

c     PUT FILES
      11 file="
      ")

      n
      olve
      (iteration,10).eq      all Output
      e


      stop
      end
```

Throw the rest of this stuff away
(less writing)
And get parallelism, modularity, and portability for free

Thorn Architecture



Thorn

Interface.ccl

Parameter Files and Testsuites

Param.ccl

Source Code

Fortran Routines

C Routines

C++ Routines

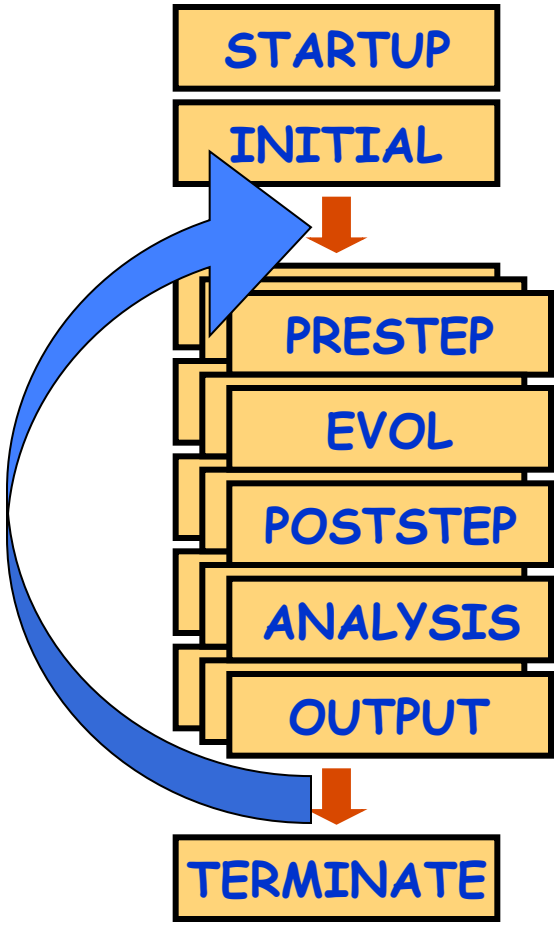
Schedule.ccl

Documentation!

Make Information

IDScalarWave: schedule.ccl

```
# Schedule definitions for scalarwave

STORAGE: scalarevolve


schedule WaveToyF77_Evolution as
    WaveToy_Evolution at EVOL
{
  LANG: Fortran
  SYNC: scalarevolve
} "Evolution of 3D wave equation”
```



IDScalarWave: param.ccl

- **Parameters are stored in runtime database that is used to**
  - **Automate creation of parameter file parsers**
  - **Enable introspection for support of remote-steering and monitoring**

```
# Parameter definitions


REAL radius "The radius of the gaussian wave"
{
 0:* :: “Radius must be positive”
} 0.0
```

WaveToyF77: param.ccl

```
# Parameter definitions for thorn WaveToyF77

private: # other options are public or inherit

KEYWORD bound "Type of boundary condition to use"
{
  "none"          :: "No boundary condition"
  "flat"          :: "Flat boundary condition"
  "static"        :: "Static boundary condition"
  "radiation"     :: "Radiation boundary condition"
  "robin"         :: "Robin boundary condition"
  "zero"          :: "Zero boundary condition"
} "none"
```

Parameter File

```
ActiveThorns = “qft time pugh pughreduce
        pughslab cartgrid3d ioutil iobasic”

time::dtfac = 0.1
pugh::periodic= “yes”
grid::type = “BySpacing”
grid::domain = “full”
grid::dxyz = 1.0

qft::lambda = 1.0
qft::smooth = 100
qft::damp = 0.5
```

ThornList

- Just a **list** of the modules you want compile into your application
  - The modules self-configure using constraints from Schedule.ccl *(You don't explicitly wire them together… that would be tedious)*
  - Some modules provide duplicate functionality *(This simply makes them available to you for runtime. The parameter file actually selects the module)*

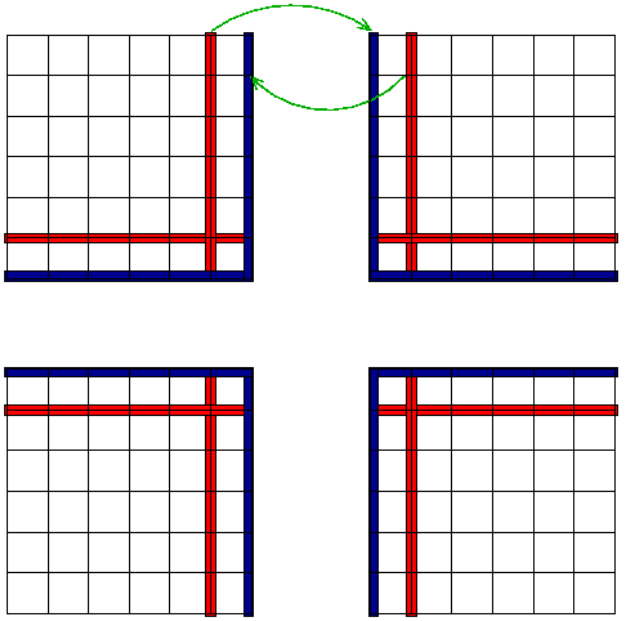
```
# arrangement/thorn              # implements (inherits) [friend] {shares}
#
CactusBase/Boundary               # boundary ( ) [ ] { }
CactusBase/CartGrid3D             # grid (coordbase) [ ] {driver}
CactusBase/CoordBase              # CoordBase ( ) [ ] { }
CactusBase/IOASCII                # IOASCII ( ) [ ] {IO}
CactusBase/IOBasic                # IOBasic (IO) [ ] {IO}
CactusBase/IOUtil                 # IO ( ) [ ] { }
CactusBase/Time                   # time ( ) [ ] {cactus}
CactusPUGH/PUGH                   # Driver ( ) [ ] {cactus}
CactusWave/IDScalarWave           # idscalarwave (wavetoy,grid) [ ] {grid}
CactusWave/WaveToyF77             # wavetoy (grid) [ ] { }
CactusWave/WaveToyCXX             # wavetoy (grid) [ ] { }
```

Parallelizing PDE Solvers (the driver)

- Central Idiom for Parallelism in Cactus: Decompose the grid across processors and exchange ghost zone information
  - Data layout and data decomposition can also be virtualized
  - Ghost-cell exchange can be hidden as “sync_grids()” operation
  - Mechanism for exchange also virtualized (MPI, copy shared memory, UPC, etc.)
  - *this exchange can be presented with a standard interface, independent of the stencil method*

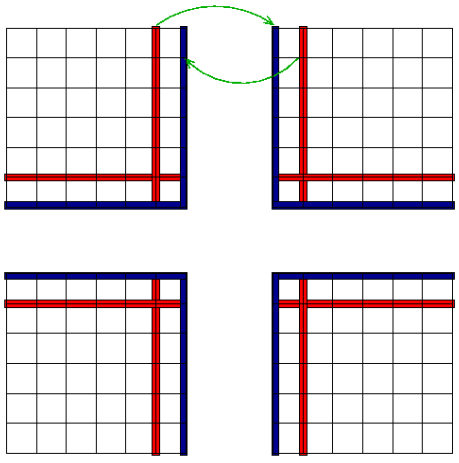
- **Standard driver distributed with Cactus (PUGH) is for a parallel unigrid and uses MPI for the communication layer**

- **PUGH can do custom processor decomposition and static load balancing**

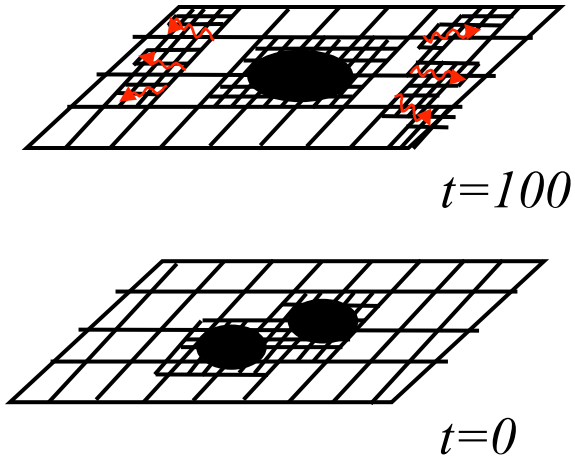


Idiom for Parallelism

- **Even the "driver" is a module – enables modular inclusion of advanced features**
  - New fault recovery mechanisms
  - Advanced static and dynamic load balancers and introspective auto-tuning systems
  - New communication mechanisms (e.g. Gemini or GAS drivers)
- **Same idiom also works for Adaptive Mesh Refinement**
  - Can switch to AMR method without any direct changes to solver
  - *Carpet (Erik Schnetter's AMR driver)*
  - *DAGH/GrACE driver for Cactus*
  - *SAMRAI driver for Cactus*



Unigrid

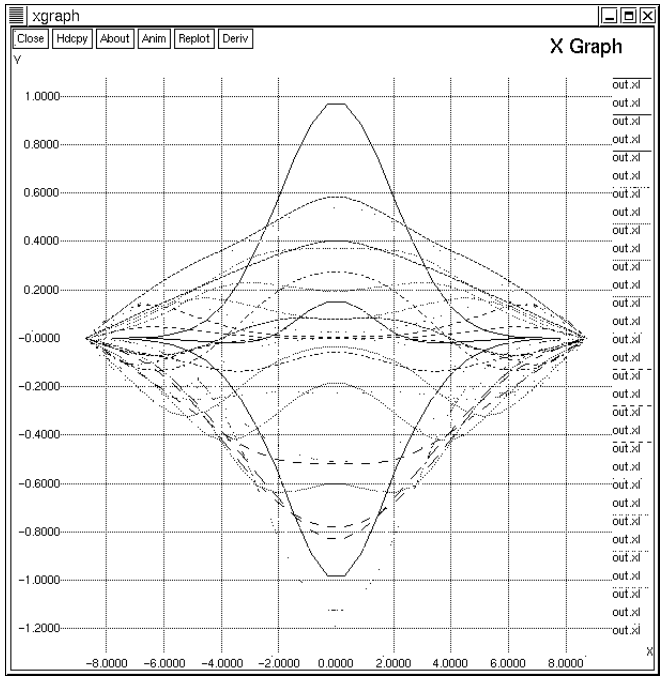


AMR

Finally an application

- make *myappname* *(on any platform!)*
- ./cactus_*myappname myoparamfile*.par
  - Congratulations, you have a Cactus application!

```
---------------------------------------------------
       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.b12
Compile date:   Jun 10 2002 (10:26:04)
Run date:       Jun 10 2002 (10:42:28)
Run host:       10:42:28
Executable:     c:\home\cactus_hello.exe
Parameter file: HelloWorld.par
---------------------------------------------------
Activating thorn Cactus...Success
Activation requested for
--->HelloWorld<---
Activating thorn HelloWorld...Success
---------------------------------------------------
do loop over timesteps
    iteration = iteration + 1
    t = t+dt
    HelloWorld: Print message to screen
enddo
---------------------------------------------------
```



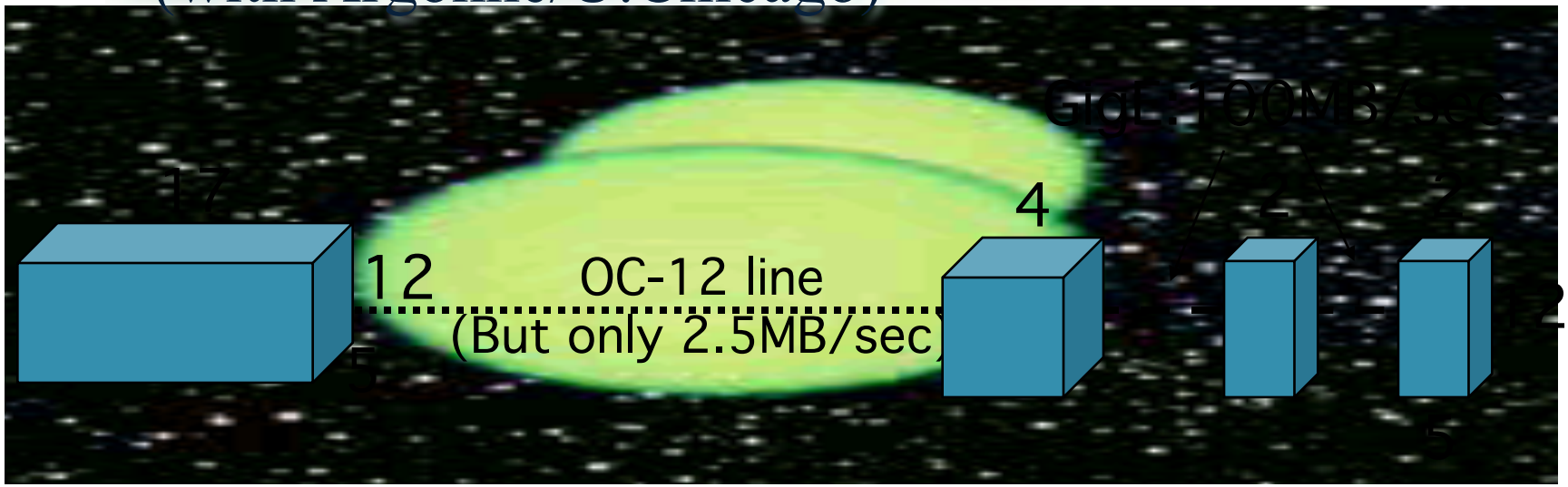
Cool Tricks with Cactus

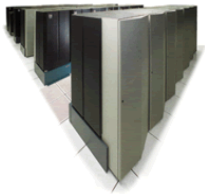
Once you have a framework, many more interesting possibilities emerge

Abstraction Enables **Auto-Tuning**

- The following example shows how the framework abstractions enable auto-tuning of the parallel performance of a code without any change to the higher-levels of the framework
  - Normally people accuse abstractions of reducing performance
  - Framework abstractions ***enable*** performance tuning!!!

Dynamic Adaptive Distributed Computation (with Argonne/U.Chicago)





SDSC IBM SP
1024 procs
5x12x17 =1020

NCSA Origin Array
256+128+128
5x12x(4+2+2) =480

This experiment:

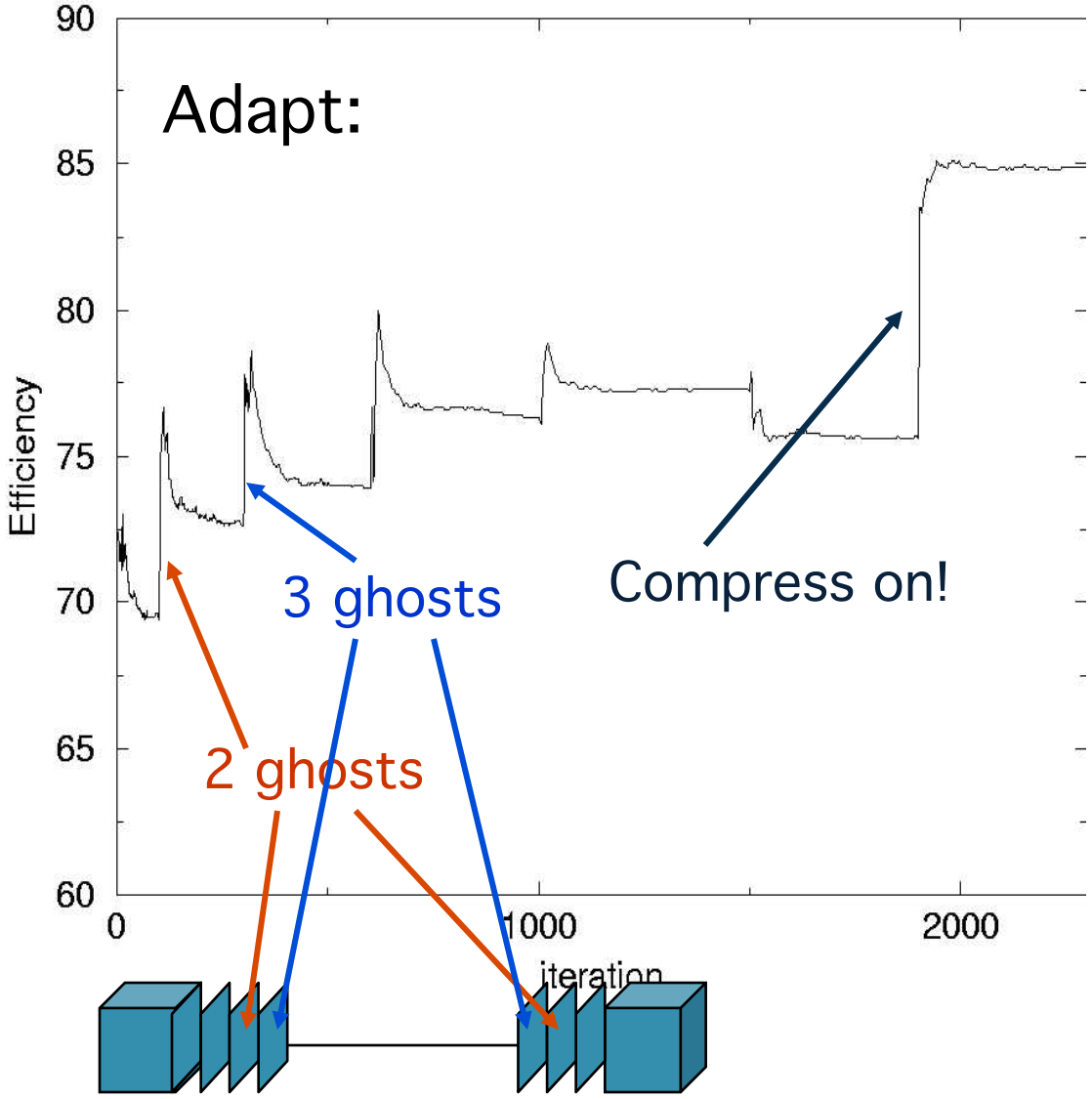
- Einstein Equations (but could be any Cactus application)

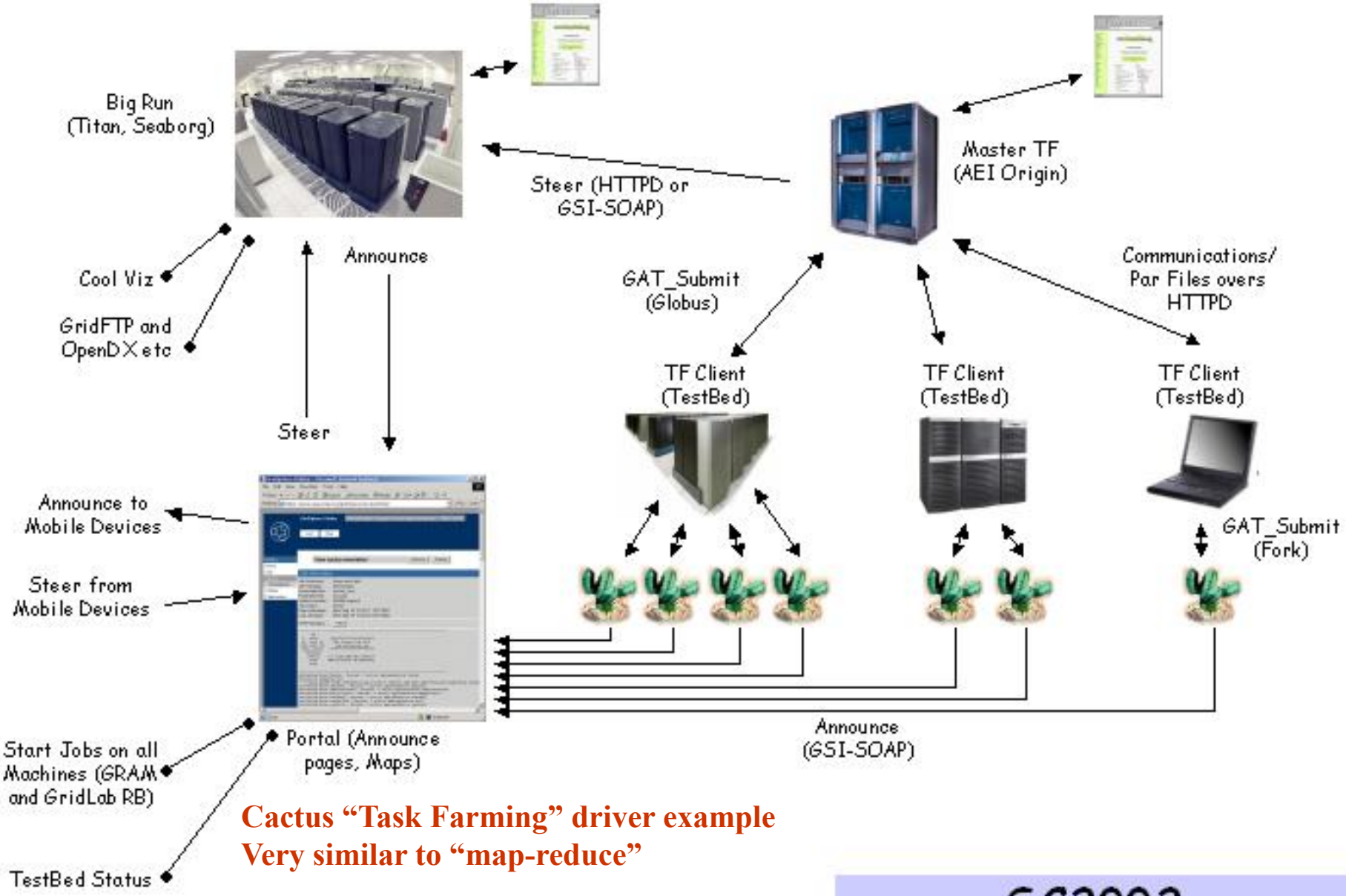
Achieved:

- First runs: 15% scaling
- With new techniques: 70-85% scaling, ~ 250GF

Dynamic Adaptation (auto-tuning)

- Automatically adapt to bandwidth latency issues
- Application has NO KNOWLEDGE of machines(s) it is on, networks, etc
- Adaptive techniques make NO assumptions about network
- **Adaptive MPI unigrid driver required NO changes to the physics components of the application!! (plug-n-play!)**
- Issues:
  - More intellegent adaption algorithm
  - Eg if network conditions change faster than adaption…





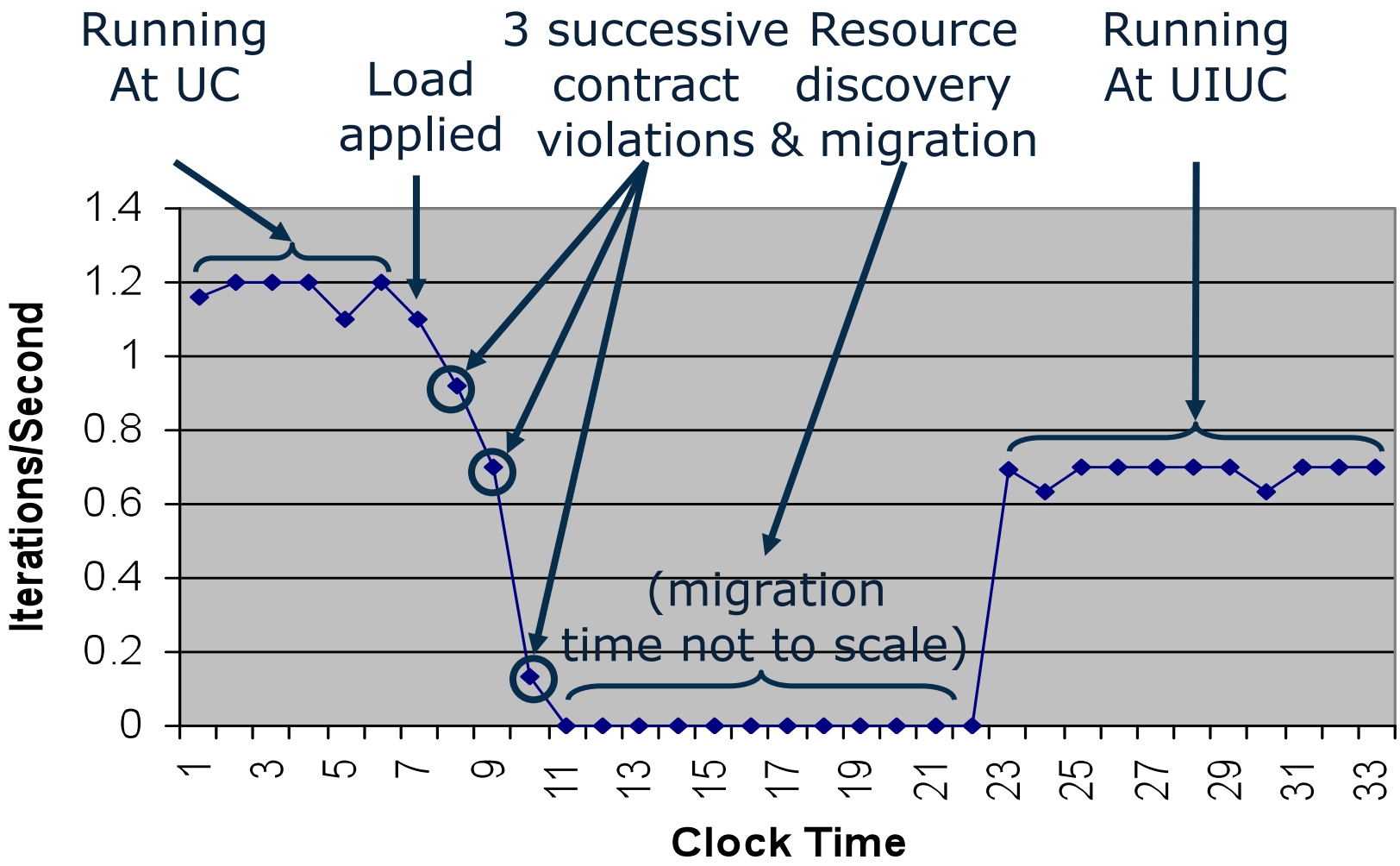
Cactus "Task Farming" driver example
Very similar to "map-reduce"

This example was used to farm out Smith-Waterman DNA sequence mapping calculations

SC2002
Demonstrations

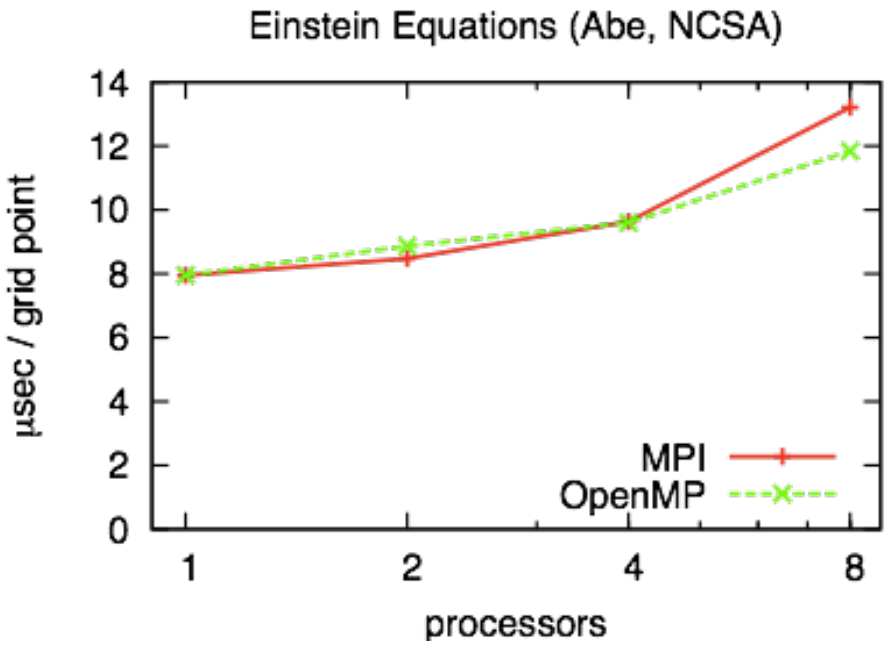
Nomadic Application Codes

(Foster, Angulo, Cactus Team…)



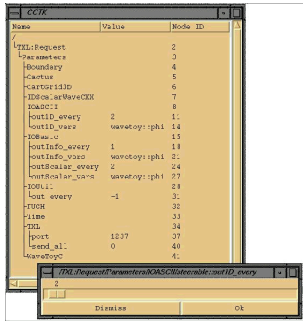
Hybrid Communication Models

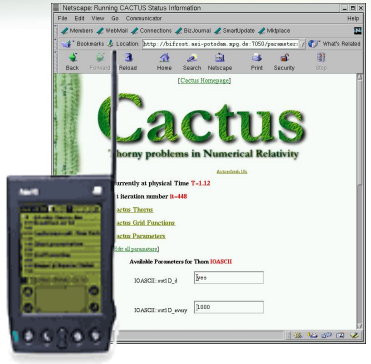
- New “multicore” driver required no changes to physics components!
- Use MPI between nodes, OpenMP within nodes



- Common address space enables more cache optimisations
- Cactus framework offers abstraction layer for parallelisation: basic OpenMP features work as black box *(central idiom)*

Remote Visualization and Steering





Any Viz Client

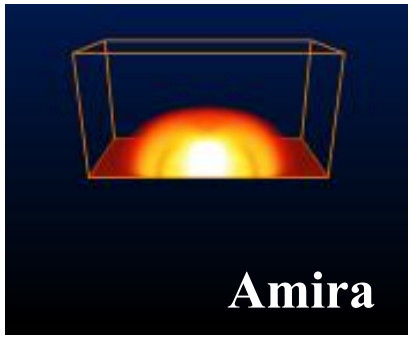
XML

HTTP

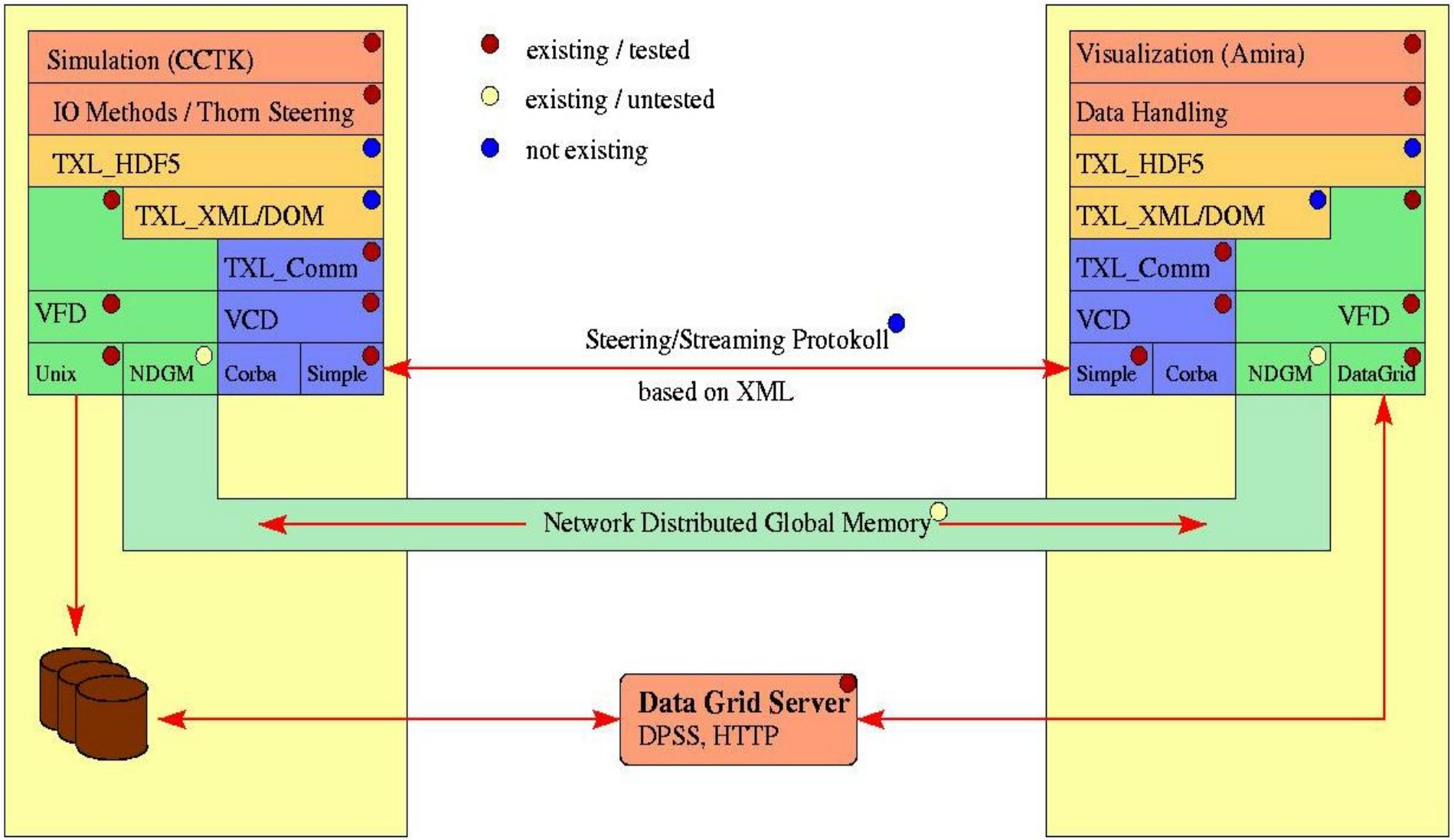
Remote
Viz data

HDF5

Remote
Viz data

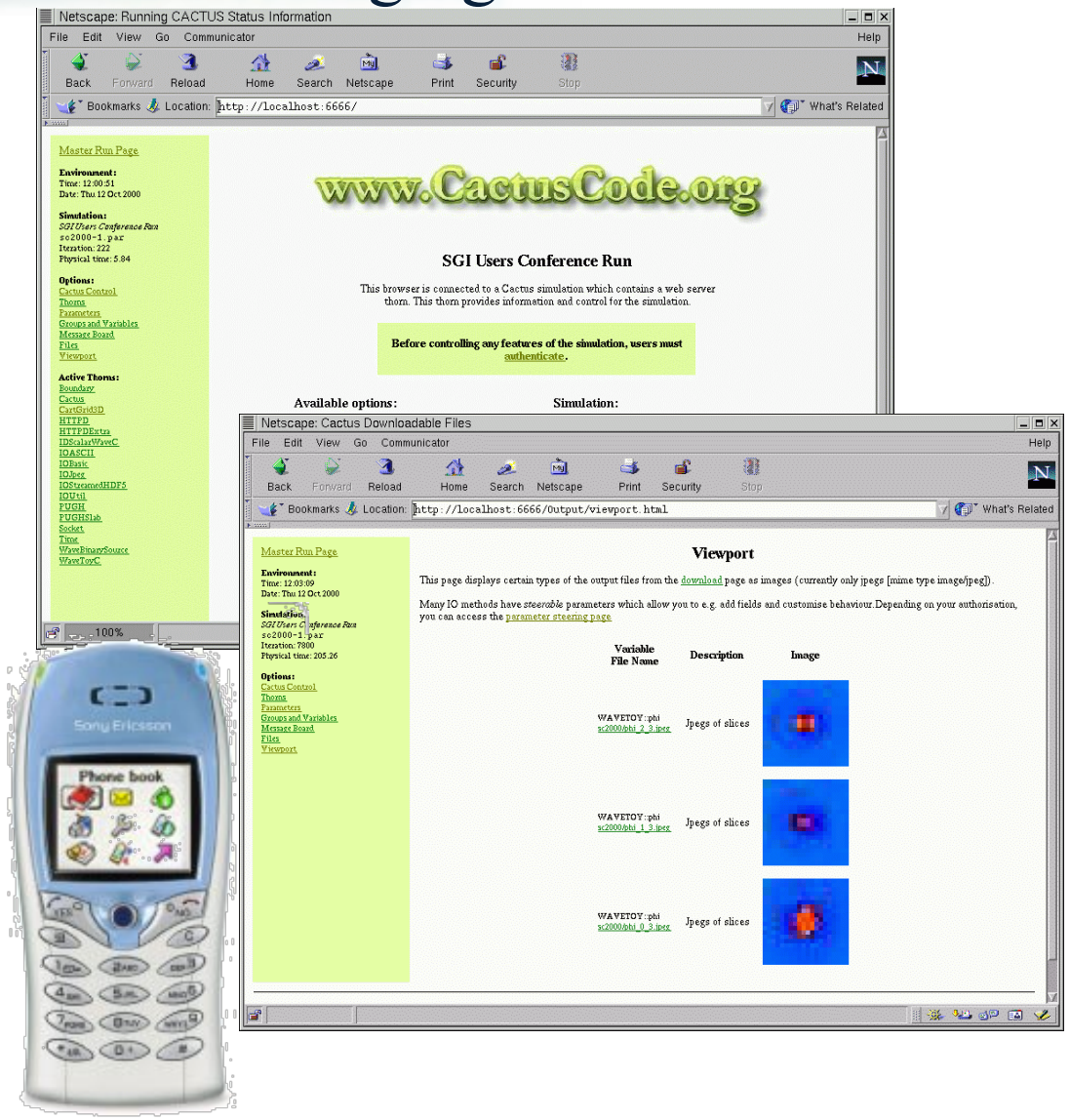


Remote Steering/Visualization Architecture

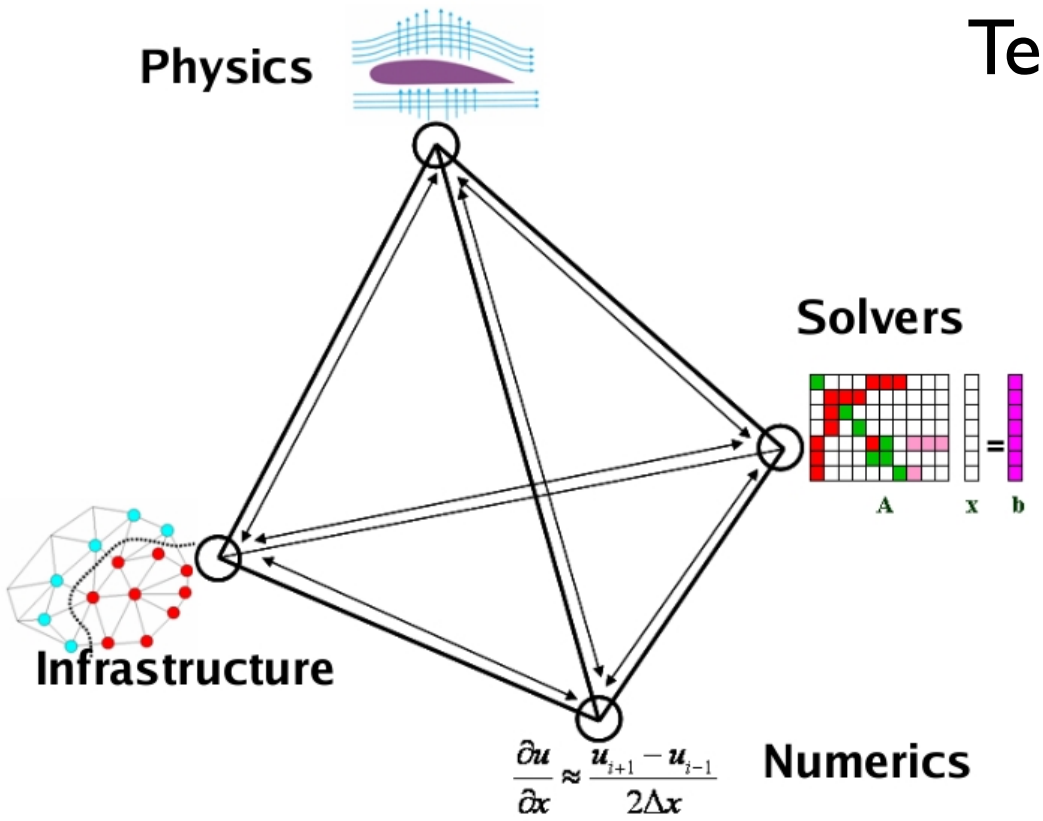


Remote Monitoring/Steering:
Thorn HTTPD and SMS Messaging

- Thorn which allows simulation any to act as its own web server
- Connect to simulation from any browser anywhere … **collaborate**
- Monitor run: parameters, basic visualization, ...
- Change *steerable* parameters
- See running example at www.CactusCode.org
- Get Text Messages from your simulation or chat with it on IM!



(future directions)CFD Toolkit



Toolkit for both Research and Teaching (on-going development)

Abstractions for physics, discretisation, solvers, and computational infrastructure

Considerations for CS267

- Are you familiar with Phil Colella's 7 dwarves?
- Perhaps there should be a canonical "framework per dwarf"
- Would be an interesting class project
  - FYI: we are also looking for summer interns to work on this idea

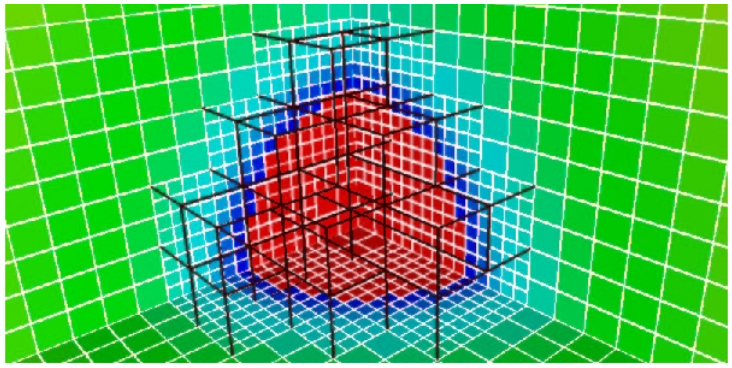
NERSC

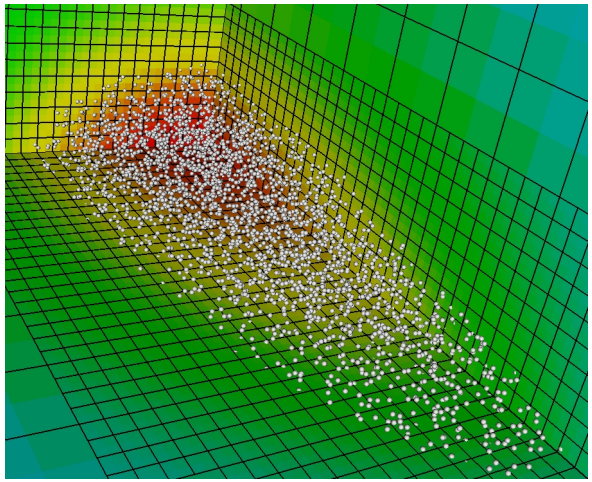
BERKELEY LAB

Examples:
Chombo AMR

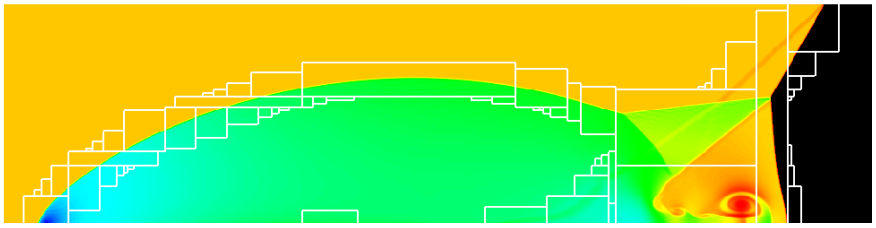
Block-Structured Local Refinement

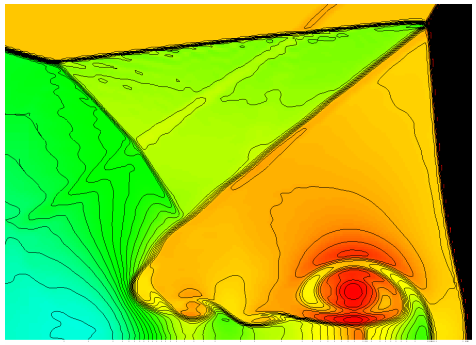
- Refined regions are organized into rectangular patches.



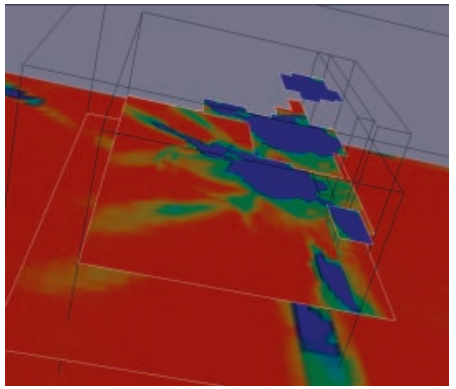


- Refinement in time as well as in space for time-dependent problems.
- Local refinement can be applied to any structured-grid data, such as bin-sorted particles.



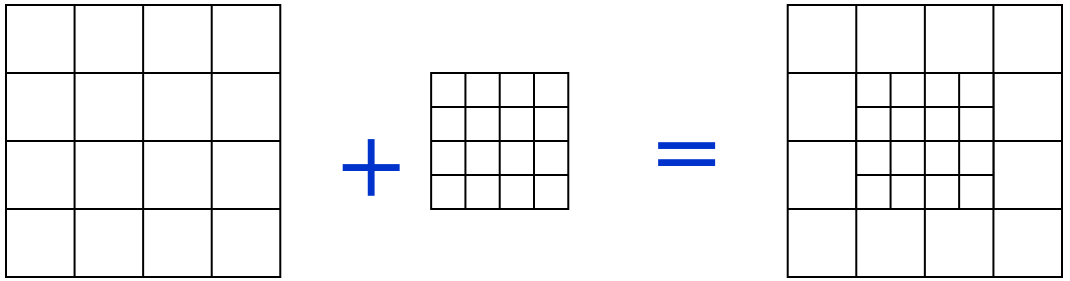


The Berger & Oliger AMR Method

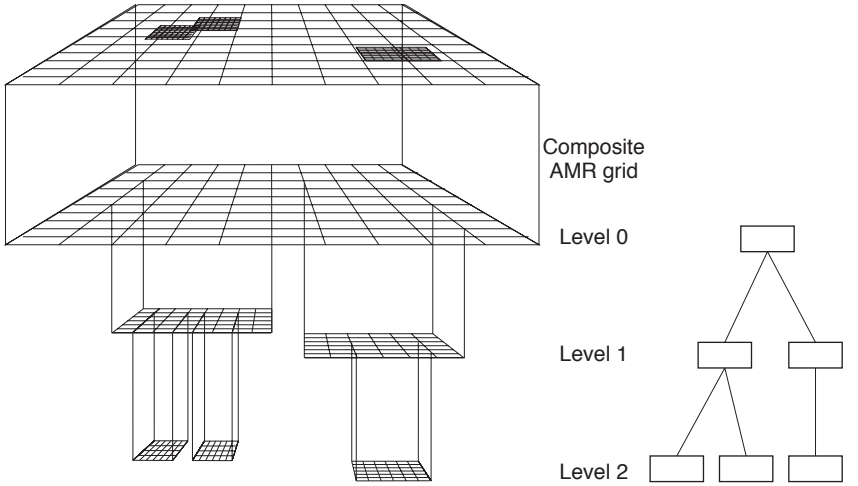


Berger & Oliger, *J. Comp. Phys.* 53, 1984

- AMR via multiple, uniform rectangular grids w/ different resolutions



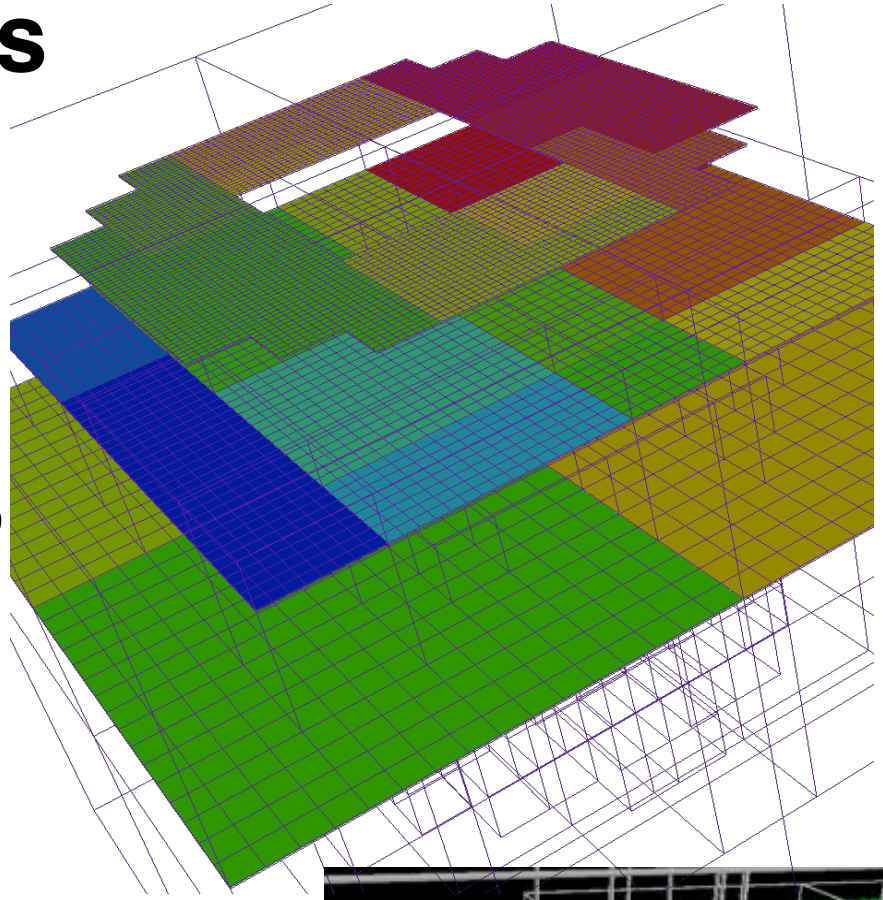
- Grids are distinct domains, made to 'communicate' along boundaries, & via *prolongation* (interpolation) & *restriction*
- Recursive algorithm, evolve level L=0 (coarsest) first, call again for level L+1...

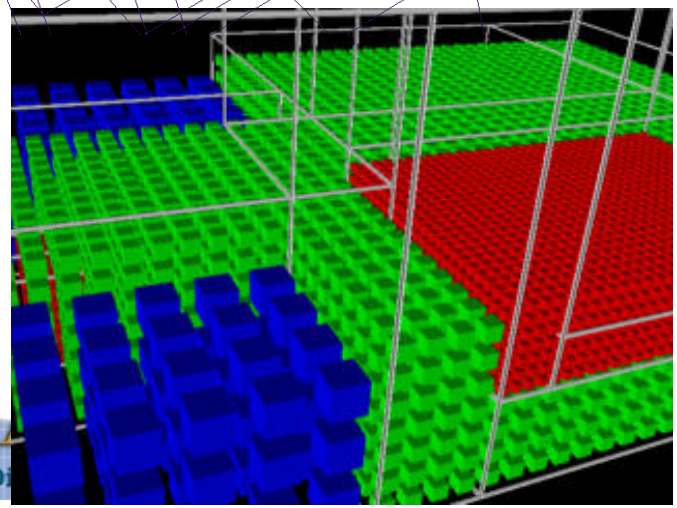


AMR ingredients

For each level subcycle:

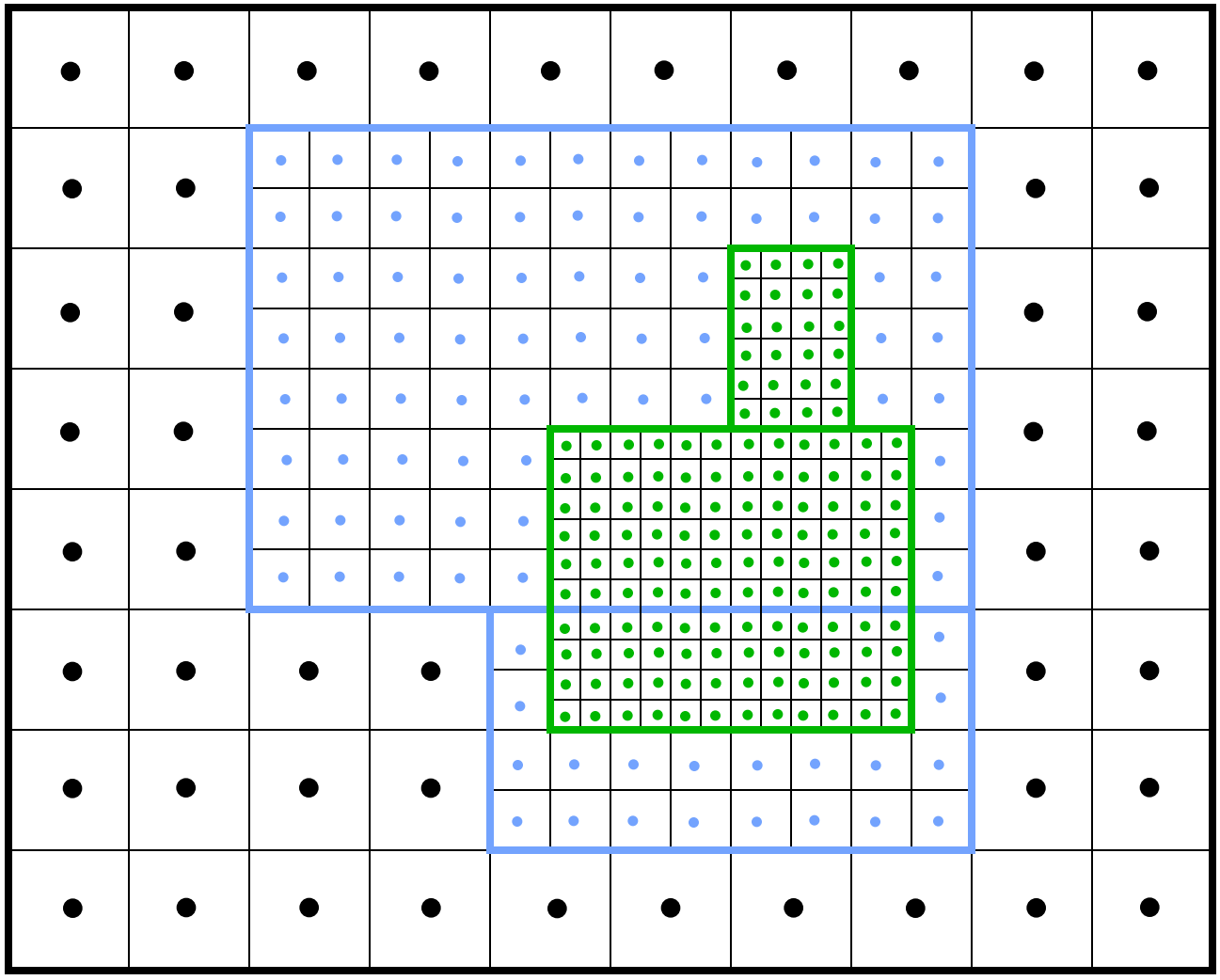
1. tag cells
2. create covering box set hierarchy
3. load balance (assign boxes to processors)
4. propagate current solution to new grids
5. advance solution:
   - hyperbolic explicit steps
   - elliptic implicit steps
   - elliptic inter-level synchronization



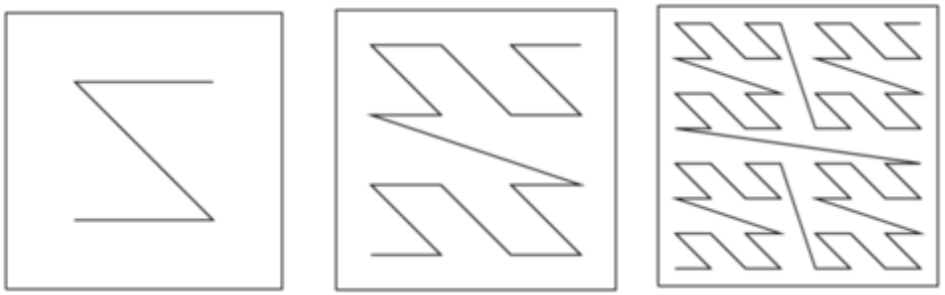


AMR requires complicated boundary updates

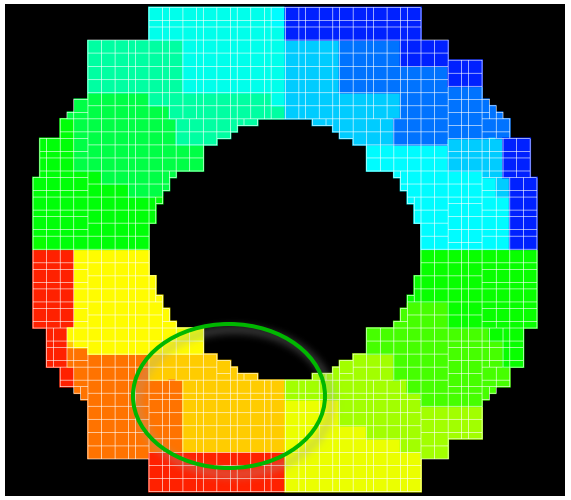
(Motivates need for Chombo Framework to manage complexity)



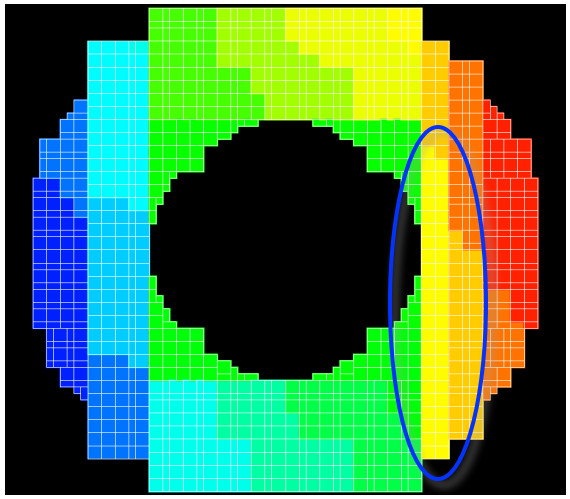
Minimizing Communications Costs



- Distributing patches to processors to maximize locality. Sort the patches by Morton ordering, and divide into equal-sized intervals.
- Overlapping local copying and MPI communications in exchanging ghost-cell data (only has an impact at 4096, 8192).
- Exchanging ghost-cell data less frequently in point relaxation.



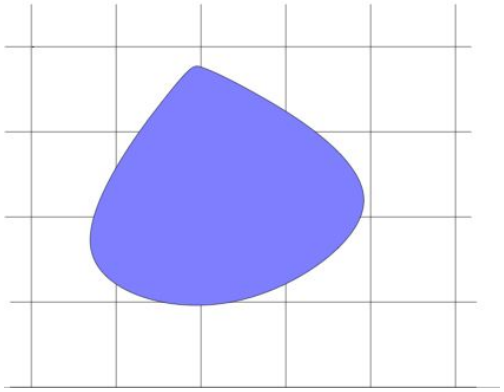
Morton-ordered load balancing (slice through 3D grids).

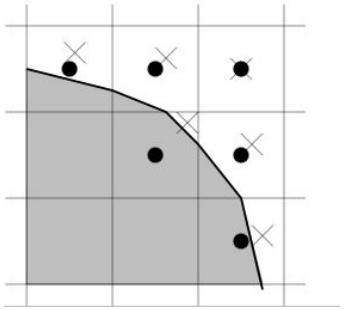


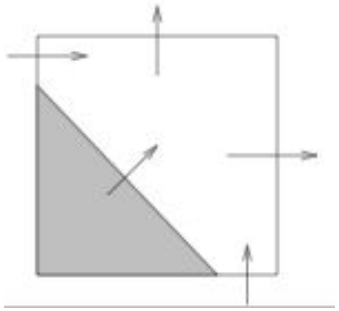
Berger-Rigoutsos + recursive bisection.

Cartesian Grid Representation of Irregular Boundaries

Based on nodal-point representation (Shortley and Weller, 1938) or finite-volume representation (Noh, 1964).







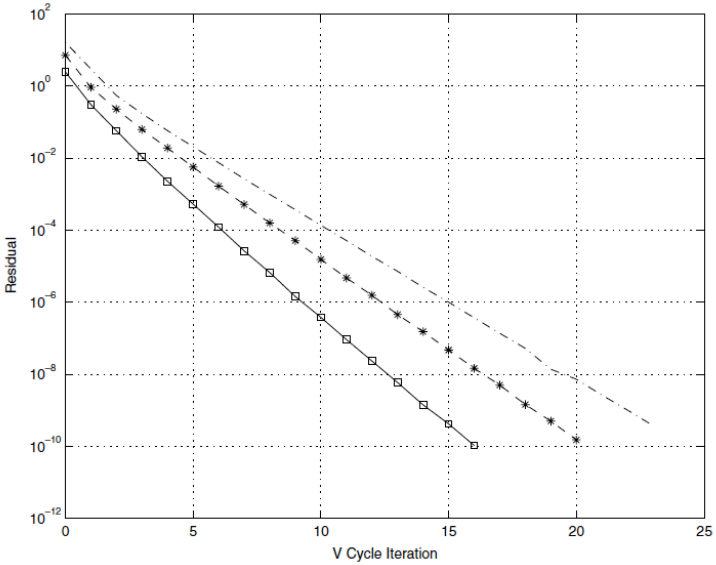
$$\nabla \cdot \vec{F} \approx \frac{1}{\kappa h^d} \int \nabla \cdot \vec{F} dx = \frac{1}{\kappa h} \sum \alpha_s \vec{F}_s \cdot \vec{n}_s + \alpha_B \vec{F} \cdot \vec{n}_B \equiv D \cdot \vec{F}$$

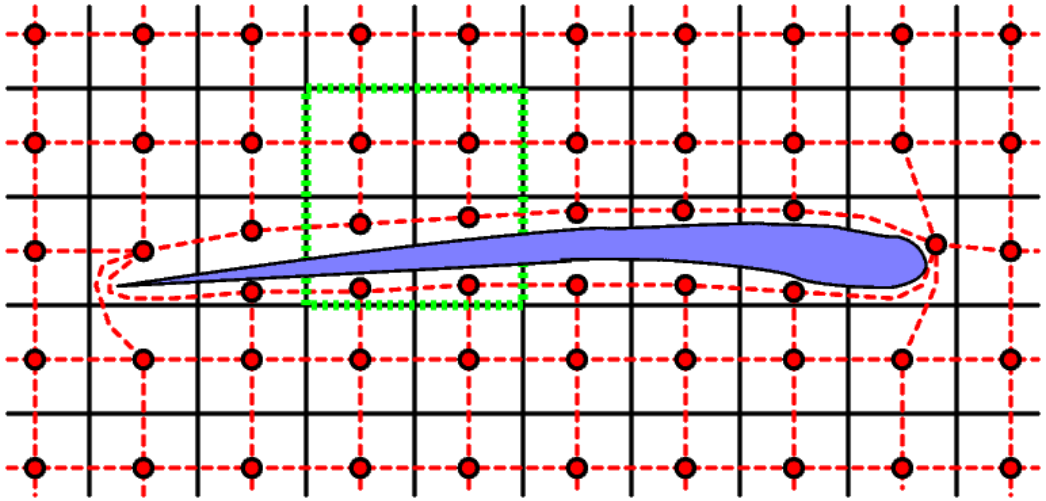
Advantages:

- Grid generation is easy.
- Good discretization technology (e.g. finite differences on rectangular grids, geometric multigrid)
- Straightforward coupling to AMR (in fact, AMR is essential).

Efficient Embedded Boundary Multigrid Solvers

- In the EB case, the matrices are not symmetric, but they are sufficiently close to M-matrices for multigrid to work (nontrivial to arrange this in 3D).
- A key step in multigrid algorithms is *coarsening*. In the non-EB case, computing the relationship between the locations of the coarse and fine data involves simple integer arithmetic. In the EB case, both the data access and the averaging operations are more complicated.
- It is essential that coarsening a geometry preserves the *topology* of the finer EB representation.





Chombo AMR Software Design Principles

The Chombo Framework is Designed to Maximize Reuse of Datastructures, Basic Operators, Algorithms, Tools, and Complete Applications

A Software Framework for Structured-Grid Applications

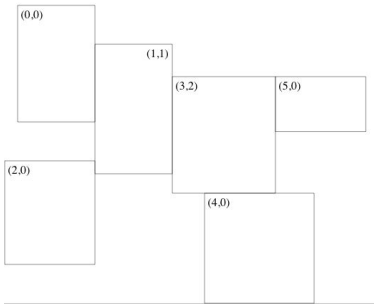
The empirical nature of multiphysics code development places a premium on the availability of a diverse and agile software toolset that enables experimentation. We accomplish this with a software architecture made up of reusable tested components organized into layers.

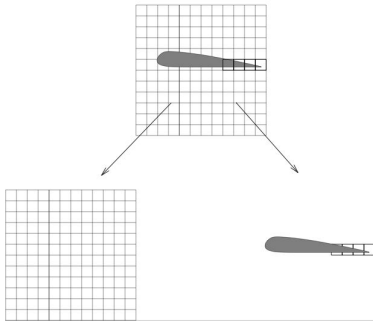
- **Layer 1**: Data and operations on unions of rectangles - *set calculus, rectangular array library (with interface to Fortran). Data on unions of rectangles, with SPMD parallelism implemented by distributing boxes to processors. Load balancing tools (e.g., SFC).*
- **Layer 2:** Tools for managing interactions between different levels of refinement in an AMR calculation - *interpolation, averaging operators, coarse-fine boundary conditions.*
- **Layer 3:** Solver libraries - *multigrid solvers on unions of rectangles, AMR hierarchies; hyperbolic solvers; AMR time stepping.*
- **Layer 4:** Complete parallel applications.
- **Utility Layer:** Support, interoperability libraries - *API for HDF5 I/O, AMR data alias.*

Mechanisms for Reuse

• **Algorithmic reuse.** Identify mathematical components that cut across applications. Easy example: solvers. Less easy example: Layer 2.

• **Reuse by templating data holders.** Easy example: rectangular array library - array values are the template type. Less easy example: data on unions of rectangles - "rectangular array" is a template type.





• **Reuse by inheritance.** Control structures (Iterative solvers, Berger-Oliger timestepping) are independent of the data, operations on that data. Use inheritance to isolate the control structure from the details of what is being controlled (interface classes).

Examples of Layer 1 Classes (BoxTools)

- `IntVect` $i \in Z^d$. Can translate $i_1 + i_2$, coarsen i / s, refine $i * s$.

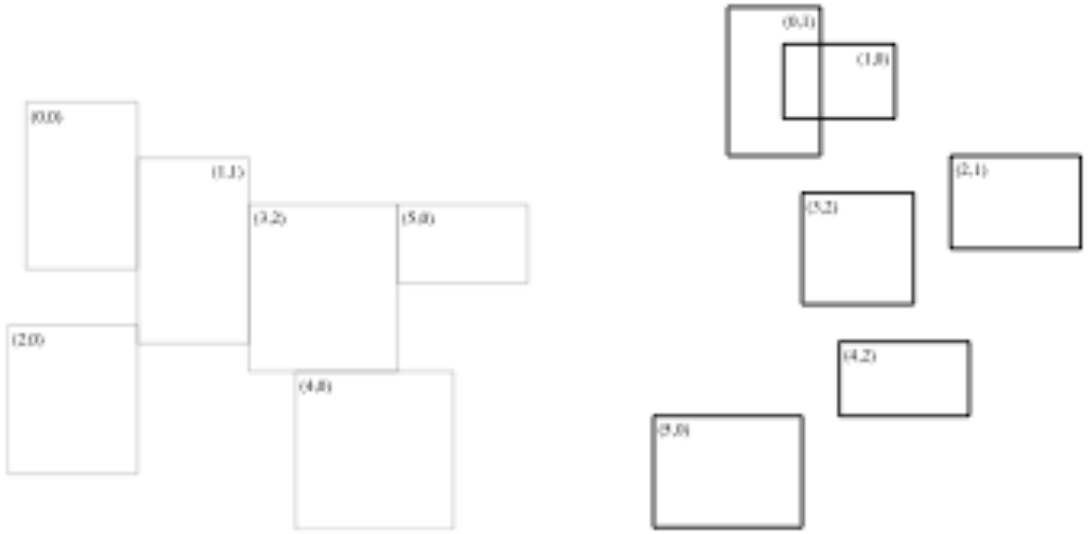
- `Box` $B \subset Z^d$ is a rectangle: $B = [i_{low}, i_{high}]$. B can be translated, coarsened, refined. Supports different centerings (node-centered vs. cell-centered) in each coordinate direction.

- `IntVectSet` $I \subset Z^d$ is an arbitrary subset of Z^d. I can be shifted, coarsened, refined. One can take unions and intersections, with other `IntVectSets` and with `Boxes`, and iterate over an `IntVectSet`.

- `FArrayBox A(Box B, int nComps)`: multidimensional arrays of doubles or floats constructed with `B` specifying the range of indices in space, `nComp` the number of components.
  - `Real* FArrayBox::dataPtr` returns the pointer to the contiguous block of data that can be passed to Fortran.

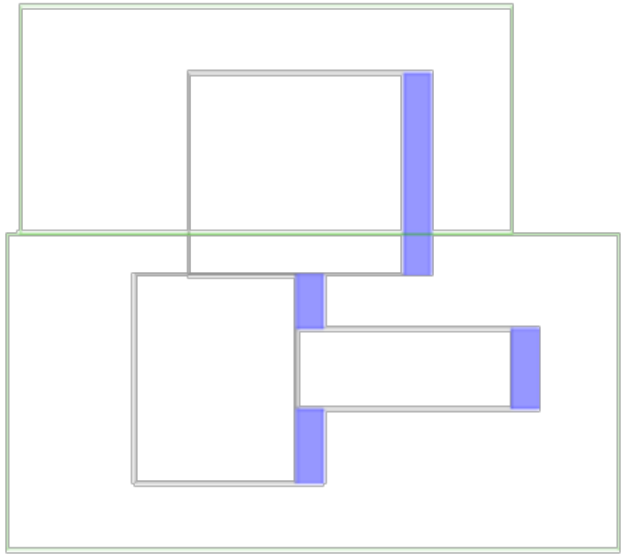
Layer 1 Reuse: Distributed Data on Unions of Rectangles

Provides a general mechanism for distributing data defined on unions of rectangles onto processors, and communication between processors.



- `BoxLayout` is a collection of `Boxes` and processor assignments:
  - Metadata of which all processors have a copy: $\{B_k, p_k\}_{k=1}^{nGrids}$.
- `template <class T> LevelData<T>` and other container classes hold data distributed over multiple processors. Straightforward API's for copying, exchanging ghost cell data, iterating over the arrays on your processor in a SPMD manner.
  - `LevelData<T>::exchange()`: obtains ghost cell data from valid regions on other patches
  - `DataIterator`: iterates over the patches that are owned by current processor.

Layer 2 Reuse: algorithmic components



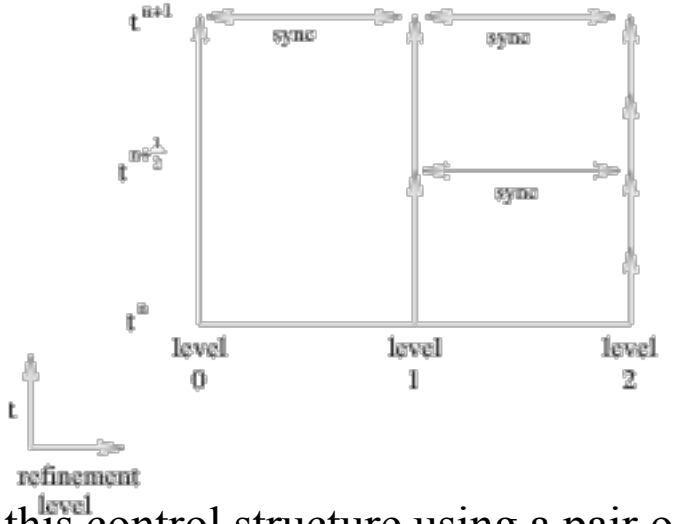
$$U^c := U^c + \frac{\Delta t^c}{h}(F^{c,s}_{i^c-\frac{1}{2}e} - \frac{1}{Z}\sum_{if} F^{f,s}_{if-\frac{1}{2}e})$$

The coarse and fine fluxes are computed at different times in the program, and on different processors. We rewrite the processes in the following step.

$$\delta F = 0$$
$$\delta F := \delta F - \Delta t^c F^c$$
$$\delta F := \delta F + \Delta t^f < F^f >$$
$$U^c := U^c + D_R(\delta F)$$

- **`LevelFluxRegister::setToZero()`**
- **`LevelFluxRegister::incrementCoarse`:** given a flux in a direction for one of the patches at the coarse level, increment the flux register for that direction.
- **`LevelFluxRegister::incrementFine`:** given a flux in a direction for one of the patches at the fine level, increment the flux register with the average of that flux onto the coarser level for that direction.
- **`LevelFluxRegister::reflux`:** given the data for the entire coarse level, increment the solution with the flux register data for all of the coordinate directions.

Layer 3: Sample AMR / AMRLevel interface for Berger-Oliger timestepping



Chombo implements this control structure using a pair of classes.

`class AMR:` manages the Berger-Oliger time-stepping process.

`class AMRLevel:` collection of virtual functions called by an AMR object that perform the operations on the data at a level, e.g.:

- **`virtual void AMRLevel::advance()=0`** advances the data at a level by one time step.
- **`virtual void AMRLevel::postTimeStep()=0`** performs whatever synchronization operations required after all the finer levels have been updated.

AMR Utility Layer

- **API for HDF5 I/O.**
- **Interoperability tools**. We have developed a framework-neutral representation for pointers to AMR data, using opaque handles. This will allow us to wrap Chombo classes with a C interface and call them from other AMR applications.
- **Chombo Fortran** - a macro package for writing dimension-independent Fortran and managing the Fortran / C interface.
- **`Parmparse` class** from BoxLib for handling input files.
- **Visualization** and analysis tools
  - (VisIt and ChomboVis).

Spiral Design Approach to Software Development

Scientific software development is inherently high-risk: multiple experimental platforms, algorithmic uncertainties, performance requirements at the highest level. The Spiral Design approach allows one to manage that risk, by allowing multiple passes at the software and providing a high degree of schedule visibility.

Software components are developed in phases.

1. **Design and implement a basic framework for a given algorithm domain (EB, particles, etc.), implementing the tools required to develop a given class of applications.**
2. **Implement one or more prototype applications as benchmarks.**
3. **Use the benchmark codes as a basis for measuring performance and evaluating design space flexibility and robustness. Modify the framework as appropriate.**
4. **The framework and applications are released, with user documentation, regression testing, and configuration for multiple platforms.**

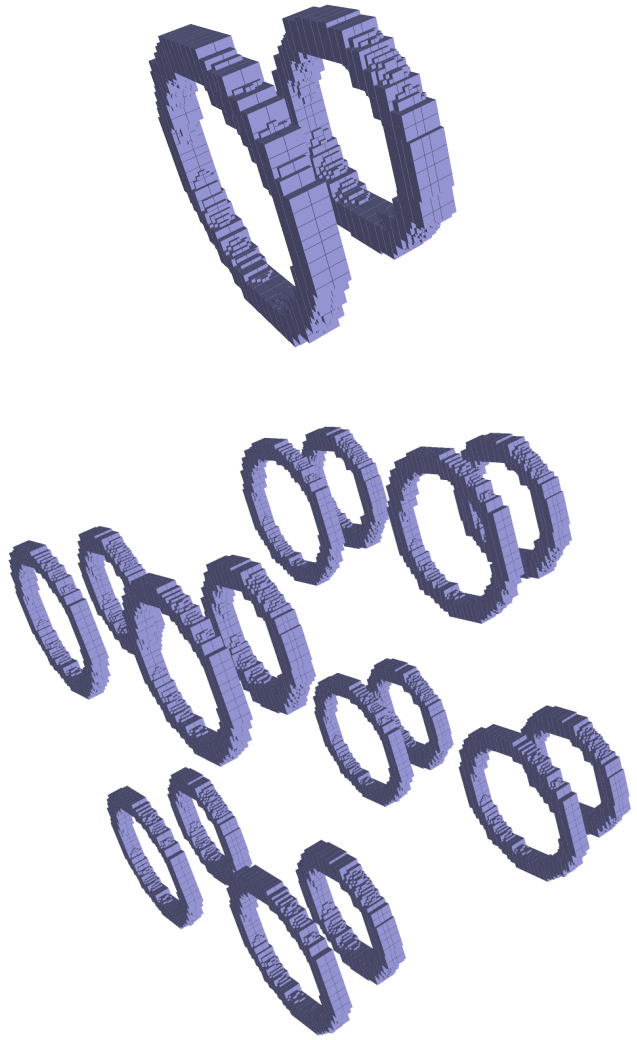
Software Engineering Plan

- **All software is open source:**
  - `http://seesar.lbl.gov/anag/software.html`.
- **Documentation: algorithm, software design documents;**
  - ***Doxygen* manual generation; users' guides.**
- **Implementation discipline:**
  - **CVS source code control**
  - **coding standards.**
- **Portability and robustness:**
  - **flexible make-based system, regression testing.**
- **Interoperability:**
  - **C interfaces, opaque handles, permit interoperability across a variety of languages (C++, Fortran 77, Python, Fortran 90).**
  - **Adaptors for large data items a serious issue, must be custom-designed for each application.**

Chombo Performance and Scalability

Replication Scaling Benchmarks

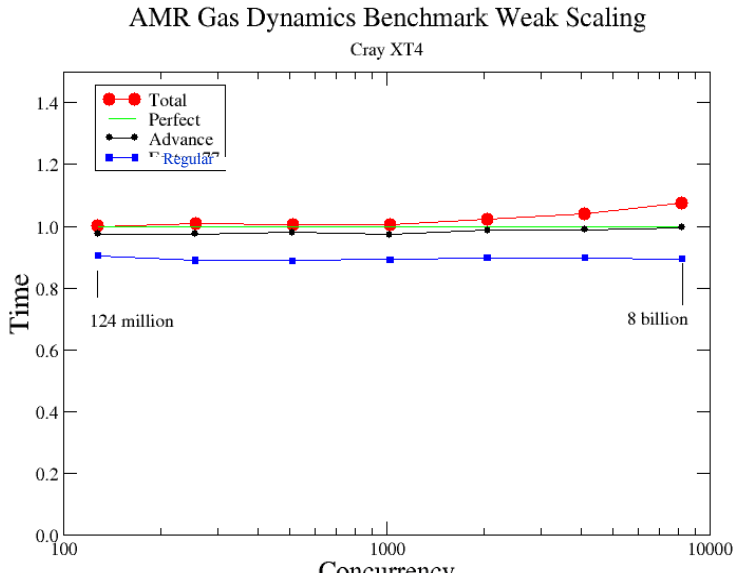
- Take a single grid hierarchy, and scale up the problem by making identical copies. Full AMR code (processor assignment, remaining problem setup) is done without knowledge of replication.
  - Good proxy for some kinds of applications scaleup.
  - Tests algorithmic weak scalability and overall performance.
  - Avoids problems with interpreting scalability of more conventional mesh refinement studies with AMR.



Replication Scaling of AMR: Cray XT4 Results

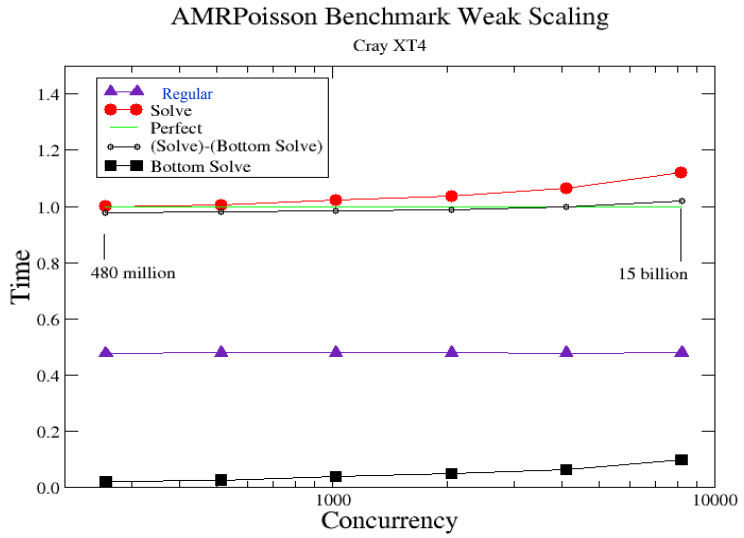
PPM gas dynamics solver:

- 97% efficient scaled speedup over range of 128-8192 processors (176-181 seconds).
- Fraction of operator peak: 90% (480 Mflops / processor).
- Adaptivity Factor: 16.



AMR-multigrid Poisson solver:

- 87% efficient scaled speedup over range of 256-8192 processors (8.4-9.5 seconds).
- Fraction of operator peak: 45% (375 Mflops / processor).
- Adaptivity factor: 48.



Embedded Boundary Performance Optimization and Scaling

- **Aggregate stencil operations, which use pointers to data in memory and integer offsets, improve serial performance by a factor of 100.**
- **Template design**

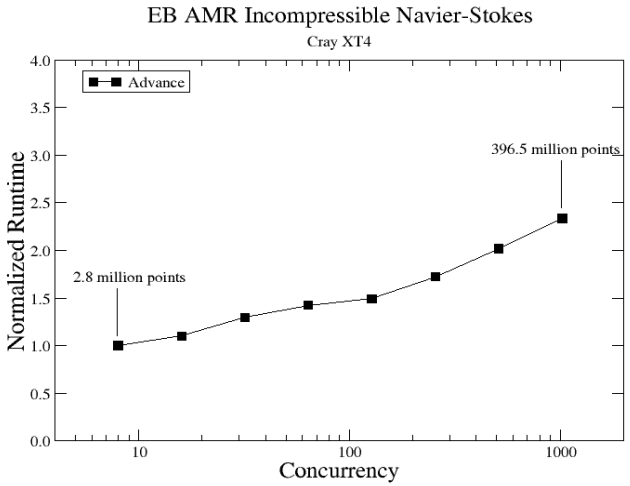
  Implement AMRMultigrid once and re-use across multiple operators
- **Operator-dependent load balancing**
- **space-filling curve algorithm to order boxes (Morton)**

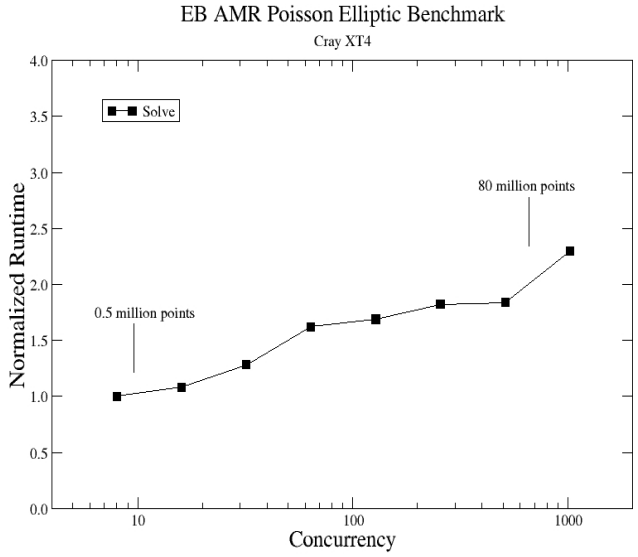
  Minimization of communication
- **Relaxing about relaxation**

  gsrb vs. multi-color.

  edge and corner trimming of boxes

- **And many many more …**





NERSC

BERKELEY LAB

Lawrence Berkeley National Laboratory / National Energy Research Supercomputing Center

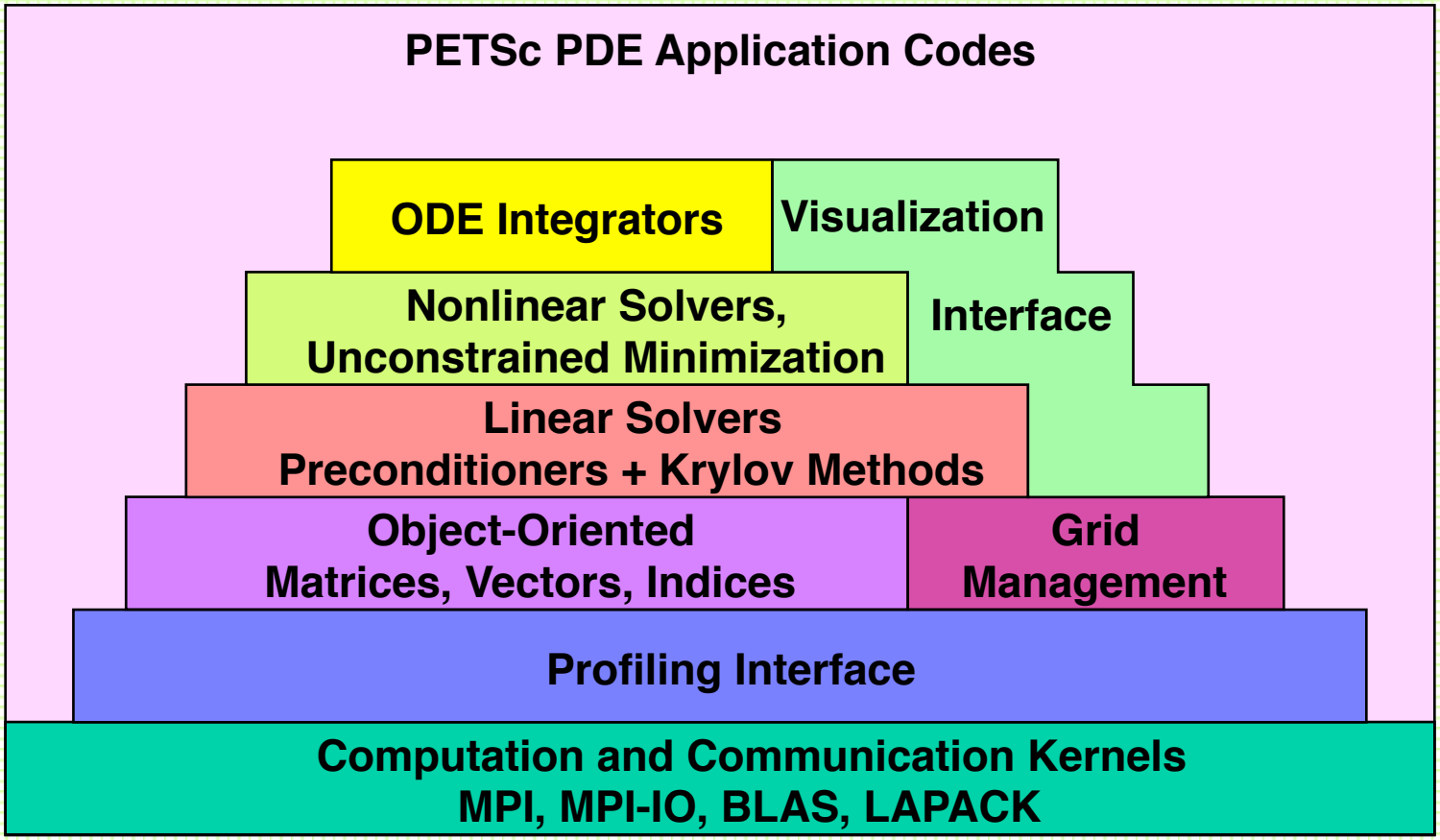
Example: PETSc

Portable, Extensible Toolkit for Scientific Computation

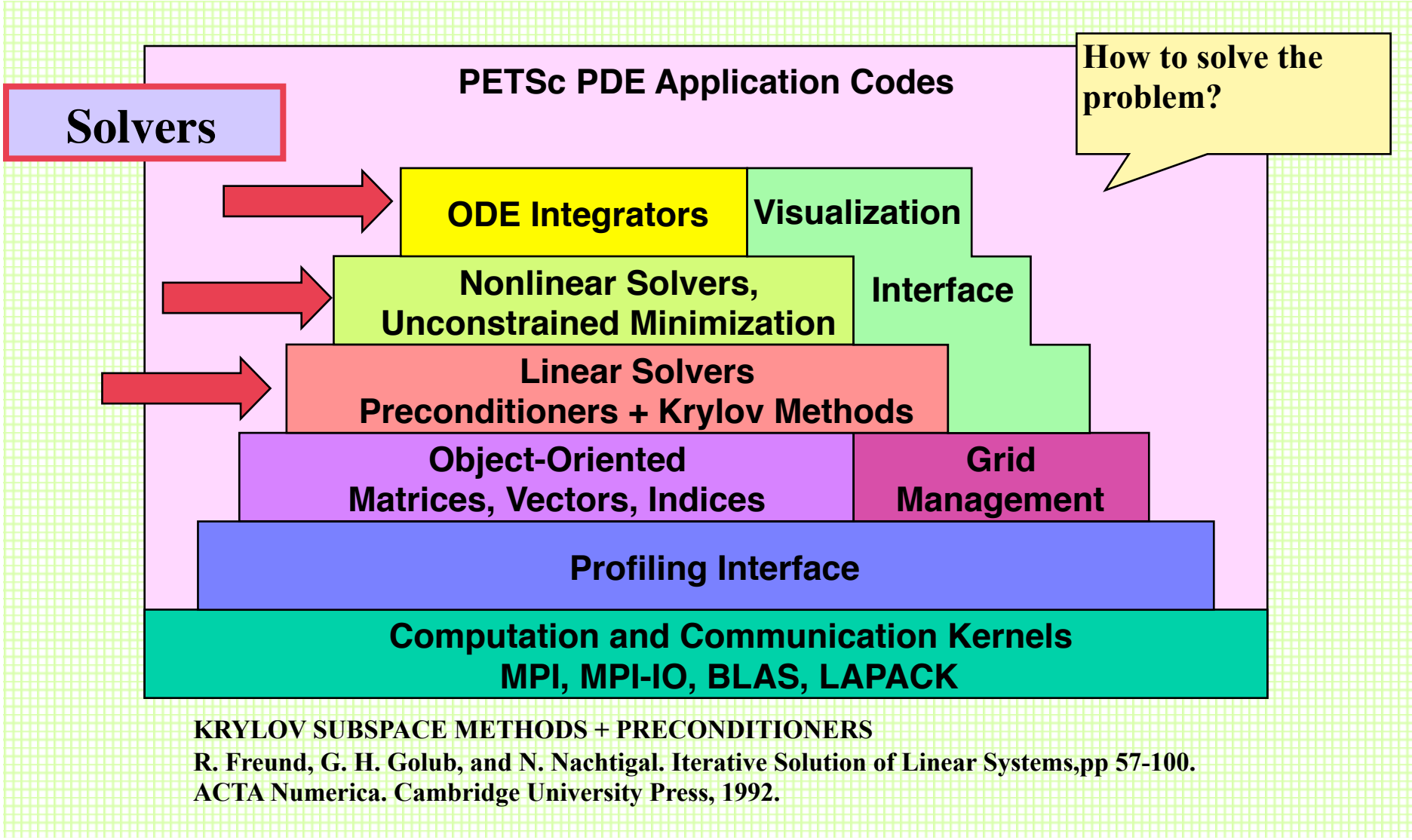
Office of Science
U.S. DEPARTMENT OF ENERGY

Argonne NATIONAL LABORATORY

PETSc Software Interfaces and Structure

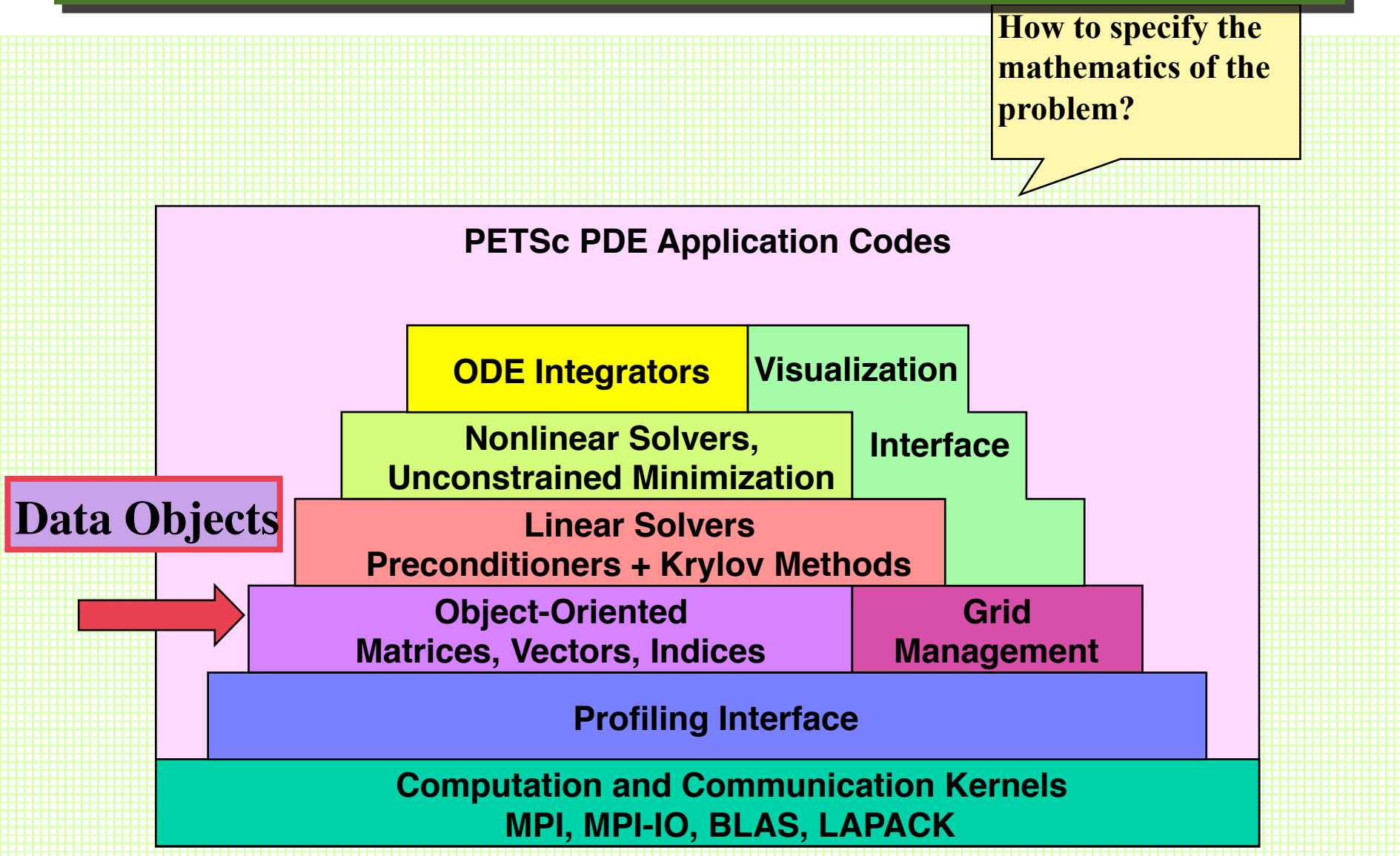


PETSc Software Interfaces and Structure

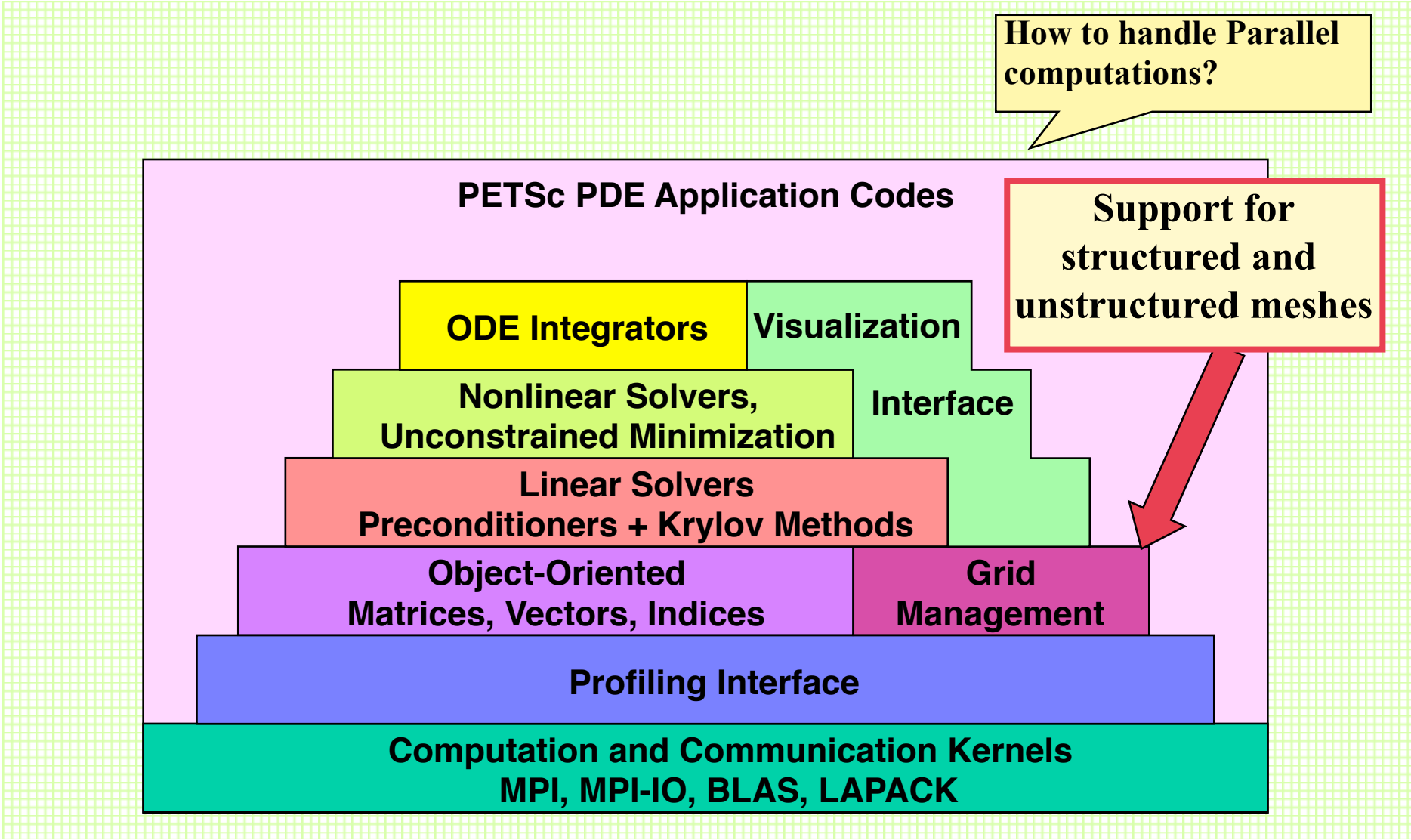


KRYLOV SUBSPACE METHODS + PRECONDITIONERS
R. Freund, G. H. Golub, and N. Nachtigal. Iterative Solution of Linear Systems,pp 57-100. ACTA Numerica. Cambridge University Press, 1992.

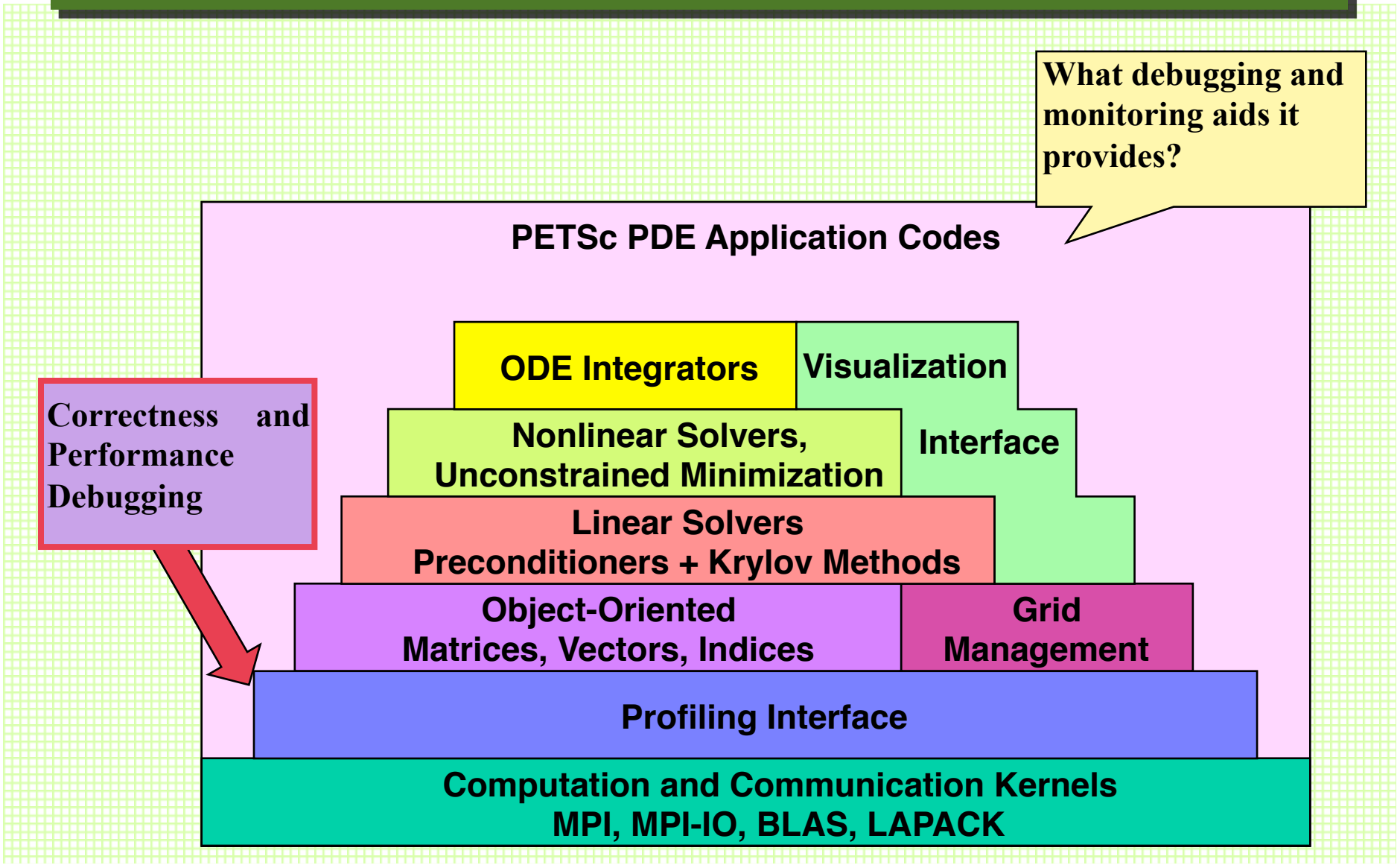
PETSc Software Interfaces and Structure



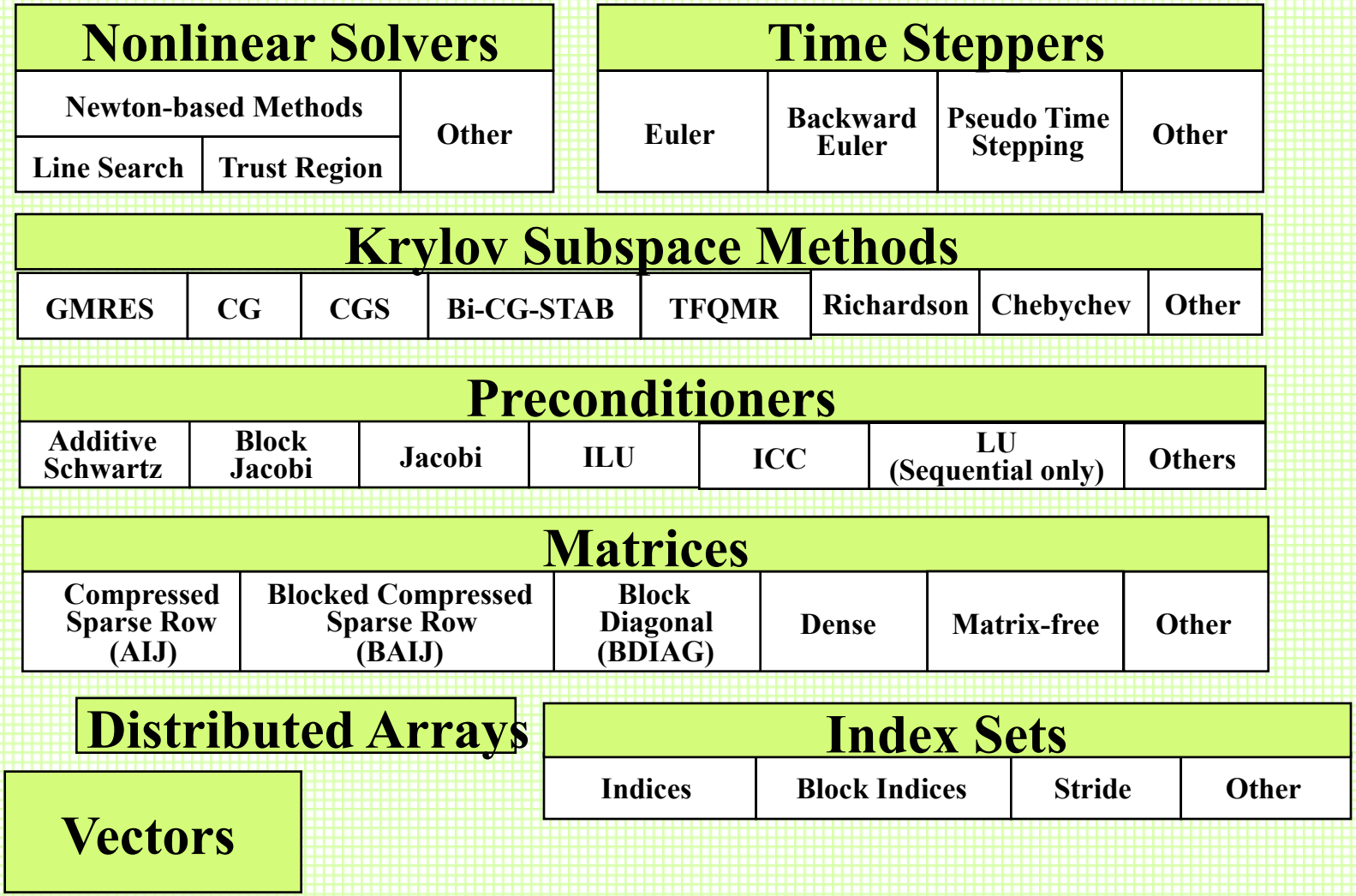
PETSc Software Interfaces and Structure



PETSc Software Interfaces and Structure



Some Algorithmic Implementations in PETSc



Basic Program setup in PETSc (In C or in Fortran)

```
•     program main
•     integer ierr, rank
•#include "include/finclude/petsc.h"
•     call PetscInitialize( PETSC_NULL_CHARACTER, ierr )
•     call MPI_Comm_rank( PETSC_COMM_WORLD, rank, ierr )
•     if (rank .eq. 0) then
•         print *, 'Hello World'
•     endif
•     call PetscFinalize(ierr)
•     end
```

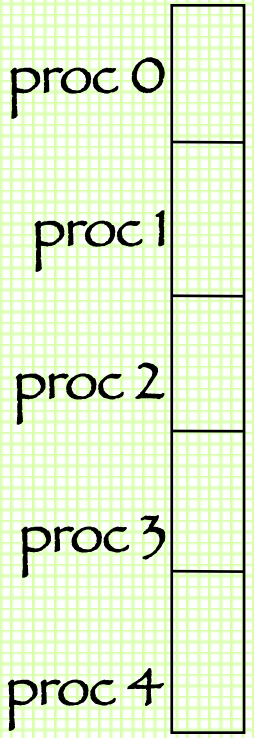
Vectors and Matrices in PETSc

VECTORS
Fundamental objects to store fields, right-hand side vectors, solution vectors, etc. . .

Matrices
Fundamental Objects to store Operators

PETSC: Some Basic Vector Operations

- PETSc vectors can be sequential (full vector is created in every process) or parallel (every process contains a part of the vector)



– Create a PETSc Vector

```
VecCreate(MPI_Comm Comm,Vec * v)
```

- comm ~ MPI_Comm parallel processes
- v ≈ vector

–Set the PETSc Vector type:

```
VecSetType(Vec,VecType)
```

- Vector Types can be:
  - **`VEC_SEQ`**, **`VEC_MPI`**, or **`VEC_SHARED`**

– Set the PETSc vector size:

```
VecSetSizes(Vec *v,int n, int N)
```

- Where n or N (not both) could be PETSC_DECIDE

– Destroy a PETSc Vector (Important for storage)

```
VecDestroy(Vec *)
```

PETSC: Some Basic Vector Operations

Function Name	Operation
VecAXPY(Scalar *a, Vec x, Vec y)	y = y + a*x
VecAYPX(Scalar *a, Vec x, Vec y)	y = x + a*y
VecWAXPY(Scalar *a, Vec x, Vec y, Vec w)	w = a*x + y
VecScale(Scalar *a, Vec x)	x = a*x
VecCopy(Vec x, Vec y)	y = x
VecPointwiseMult(Vec x, Vec y, Vec w)	w_i = x_i *y_i
VecMax(Vec x, int *idx, double *r)	r = max x_i
VecShift(Scalar *s, Vec x)	x_i = s+x_i
VecAbs(Vec x)	x_i = |x_i |
VecNorm(Vec x, NormType type , double *r)	r = ||x||

PETSC: Some Basic Matrix Operations

- Create a PETSc Matrix

  `MatCreate(MPI_Comm comm, Mat *A)`

- Set the PETSc Matrix type

  `MatSetType(Mat *A, MatType matype)`

  - default sparse AIJ (generic), MPIAIJ (parallel), SEQAIJ (sequential)
  - block sparse AIJ (for multi-component PDEs): MPIAIJ, SEQAIJ
  - symmetric block sparse AIJ: MPISBAIJ, SAEQSBAIJ
  - block diagonal: MPIBDIAG, SEQBDIAG
  - dense: MPIDENSE, SEQDENSE

- Set the PETSc Matrix sizes

  `MatSetSizes(Mat *A, PetscInt m, PetscInt n, PetscInt M, PetscInt N )`

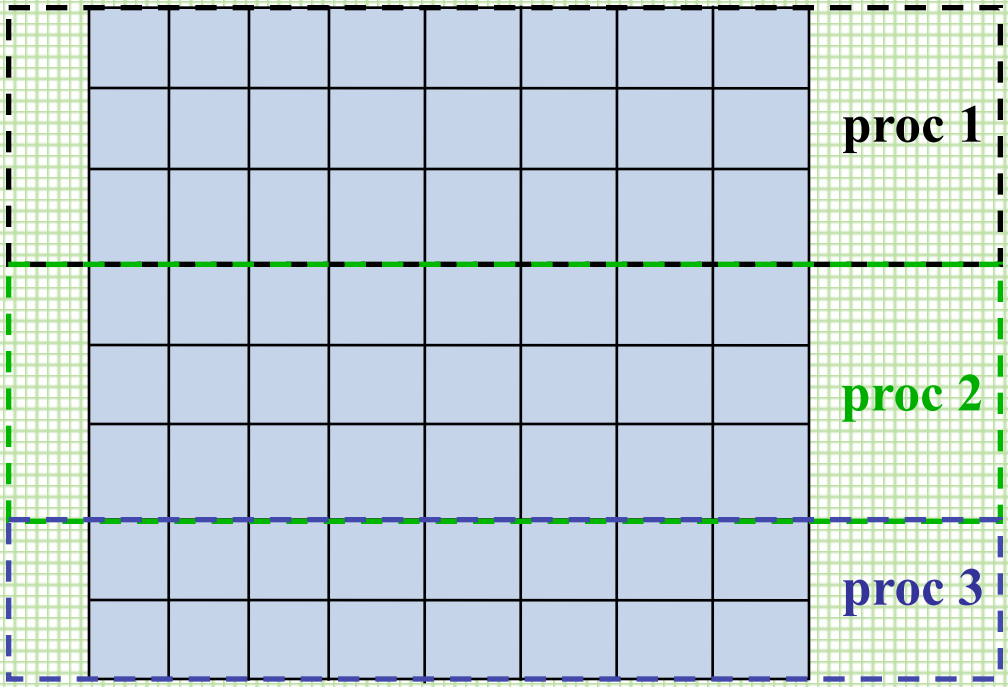
  - where m, n are the dimensions of local sub-matrix. M, N are the dimensions of the global matrix A

- Destroy a PETSc Matrix

  `MatDestroy(Mat *A)`

PETSC: Some Basic Vector Operations

Every process will receive a set of consecutive and non-overlapping rows, the columns are determined by the matrix non-zero structure ($\max(n_i) = N$)



$M=8,N=8,m_1=3,n_1=k_1$
rstart=0,rend=4

$M=8,N=8,m_2=3,n_2=k_2$
rstart=3,rend=6

$M=8,N=8,m_3=2,n_3=k_3$
rstart=6,rend=8

PETSC: Some Basic Viewer Operations

- VIEWERS provide information on any PETSc conceptual Object
- VIEWERS can be setup inside the program or at execution time
- VIEWERS provide an interface for extracting data and making it available to other tools and libraries
  - vector fields, matrix contents
  - various formats (ASCII, binary)
- Visualization
  - simple graphics created with X11.

PETSC: Some Basic Viewer Operations

```
MatView(Mat A, PetscViewer v);
```

With **PETSC_VIEWER_DRAW_WORLD**

~ Other useful viewers can be set through PETScViewerSetFormat:

- **PETSC_VIEWER_ASCII_MATLAB**
- **PETSC_VIEWER_ASCII_DENSE**
- **PETSC_VIEWER_ASCII_INFO**
- **PETSC_VIEWER_ASCII_INFO DETAILED**

Linear Systems in PETSc

- PETSc Linear System Solver Interface (KSP)
- Solve: Ax=b,
- Based on the Krylov subspace methods with the use of a preconditioning technique to accelerate the convergence rate of the numerical scheme.

KRYLOV SUBSPACE METHODS + PRECONDITIONERS
R. Freund, G. H. Golub, and N. Nachtigal. ***Iterative Solution of Linear Systems,pp 57-100. ACTA Numerica. Cambridge University Press, 1992.***

$$(M_L^{-1} A M_R^{-1})(M_R x) \quad M_L^{-1} b,$$

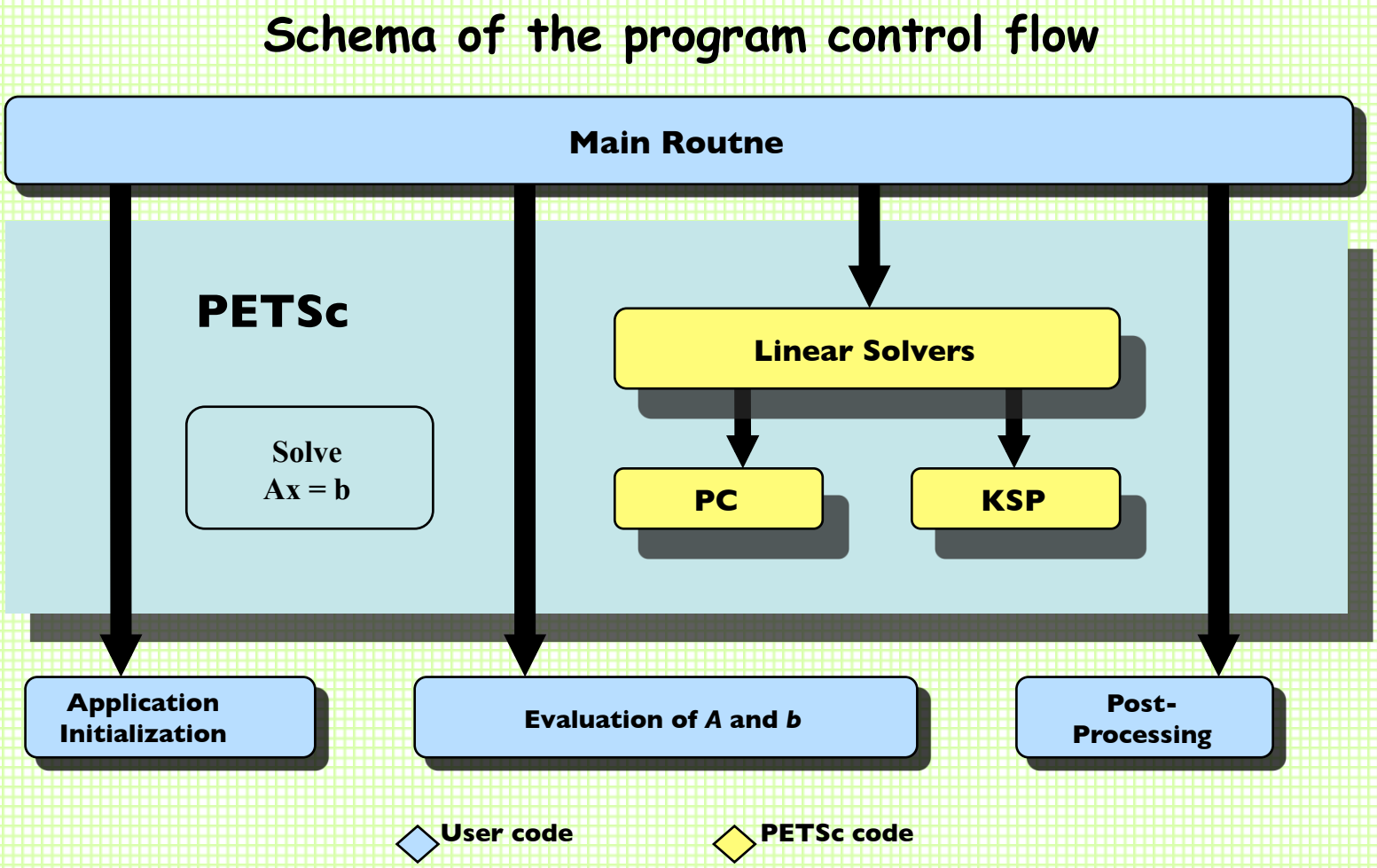
- For left and right preconditioning matrices, M_L and M_R, respectively

For $\mathbf{M_R = I}$

$$r_L \quad M_L^{-1} b \quad M_L^{-1} A x \quad M_L^{-1} r$$

PETSC Default

Linear Systems in PETSc



Linear Systems in PETSc

- To solve a Linear System, Ax = b in PETSc, one needs:
- Declare x, b as PETSc vectors, and set the RHS b
- Declare the matrix A, and explicitly set the matrix A when appropriate
- Set the Solver KSP:
  - Option 1:
    - Select the base Krylov subspace based solver
    - Select the preconditioner (**`Petsc PC`**)
  - Option 2:
    - Set the solver to use a solver from an external library

PETSc: Linear Solver - KSP Interface

- Create a KSP Object

```
KSPCreate(MPI_Comm comm, KSP *ksp)
```

- Set KSP Operators

```
KSPSetOperators(KSP *ksp, Mat Amat, Mat Pmat,
                MatStructure flag)
```

- Solve Linear System

```
KSPSolve(KSP *ksp, Vec b, Vec x)
```

- Get Iteration Number

```
KSPSolve(KSP *ksp, int *its)
```

- Destroy Solver

```
KSPDestroy(KSP *ksp)
```

PETSc: Linear Solver - KSP Interface

KSP Object:

- Is the key element to manipulate linear solver
- Stores the state of the solver and other relevant information like:
  - Convergence rate and tolerance
  - Number of iteration steps
  - Preconditioners

PETSc: Linear Solver - KSP Interface

- Create a KSP Object

```
KSPCreate(MPI_Comm comm, KSP *ksp)
```

- Set KSP Operators

```
KSPSetOperators(KSP *ksp, Mat Amat, Mat Pmat,
                MatStructure flag)
```

Amat: is the original matrix from Ax=b

Pmat: is the place holder for the preconditioning matrix (can be the same as A)

flag: saves work while repeatedly solving linear systems of the same size using the same preconditioners. Possible values:

- **SAME_NONZERO_PATTERN** (same pattern for Pmat)
- **DIFFERENT_NONZERO_PATTERN** (different pattern for Pmat)
- **SAM_PRECONDITIONER** (identical Pmat)

PETSc: Linear Solver - KSP Interface

- Set the type PETSc KSP solver

```
KSPSetType(KSP *ksp, KSPType method)
```

Method	KSPType	Options Database Name	Default Convergence Monitor†
Richardson	KSPRICHARDSON	richardson	true
Chebychev	KSPCHEBYCHEV	chebychev	true
Conjugate Gradient [11]	KSPCG	cg	true
BiConjugate Gradient	KSPBICG	bicg	true
Generalized Minimal Residual [15]	KSPGMRES	gmres	precond
BiCGSTAB [18]	KSPBCGS	bcgs	precond
Conjugate Gradient Squared [17]	KSPCGS	cgs	precond
Transpose-Free Quasi-Minimal Residual (1) [7]	KSPTFQMR	tfqmr	precond
Transpose-Free Quasi-Minimal Residual (2)	KSPTCQMR	tcqmr	precond
Conjugate Residual	KSPCR	cr	precond
Least Squares Method	KSPLSQR	lsqr	precond
Shell for no KSP method	KSPPREONLY	preonly	precond

†true - denotes true residual norm, precond - denotes preconditioned residual norm

Table 3: KSP Defaults. All methods use left preconditioning by default.

PETSc: Linear Solver - KSP Interface

- Setting up the Preconditioners

```
KSPGetPC(KSP ksp,PC *pc);
PCSetType(PC *pc, const PCType type)
```

Method	PCType	Options Database Name
Jacobi	PCJACOBI	jacobi
Block Jacobi	PCBJACOBI	bjacobi
SOR (and SSOR)	PCSOR	sor
SOR with Eisenstat trick	PCEISENSTAT	eisenstat
Incomplete Cholesky	PCICC	icc
Incomplete LU	PCILU	ilu
Additive Schwarz	PCASM	asm
Linear solver	PCKSP	ksp
Combination of preconditioners	PCCOMPOSITE	composite
LU	PCLU	lu
Cholesky	PCCholesky	cholesky
No preconditioning	PCNONE	none
Shell for user-defined PC	PCSHELL	shell

Table 4: PETSc Preconditioners

Summary

- Computational Science is increasingly carried out in large teams formed around applications frameworks
- Frameworks enable large and diverse teams to collaborate by organizing teams according to their capabilities
- Frameworks are modular, highly configurable, and extensible
- Isolation of applications, solver, and driver layers enables re-use in different applications domains, and scalability on new parallel architectures

More Information

- NERSC Advanced technology Group
  - http://www.nersc.gov/projects/SDSA/
- Cactus: Numerical Relativity and the Cactus Community Code
  - http://www.cactuscode.org/
  - http://dsc.discovery.com/schedule/episode.jsp?episode=23428000
- Chombo: The Applied Numerical Algorithms Group (ANAG)
  - https://seesar.lbl.gov/anag/chombo/
- PETSc: Portable Extensible Toolkit for Scientific Computation
  - http://www.mcs.anl.gov/petsc/petsc-as/