Managing Application Complexity in the SAMRAI Framework

Rich Hornung

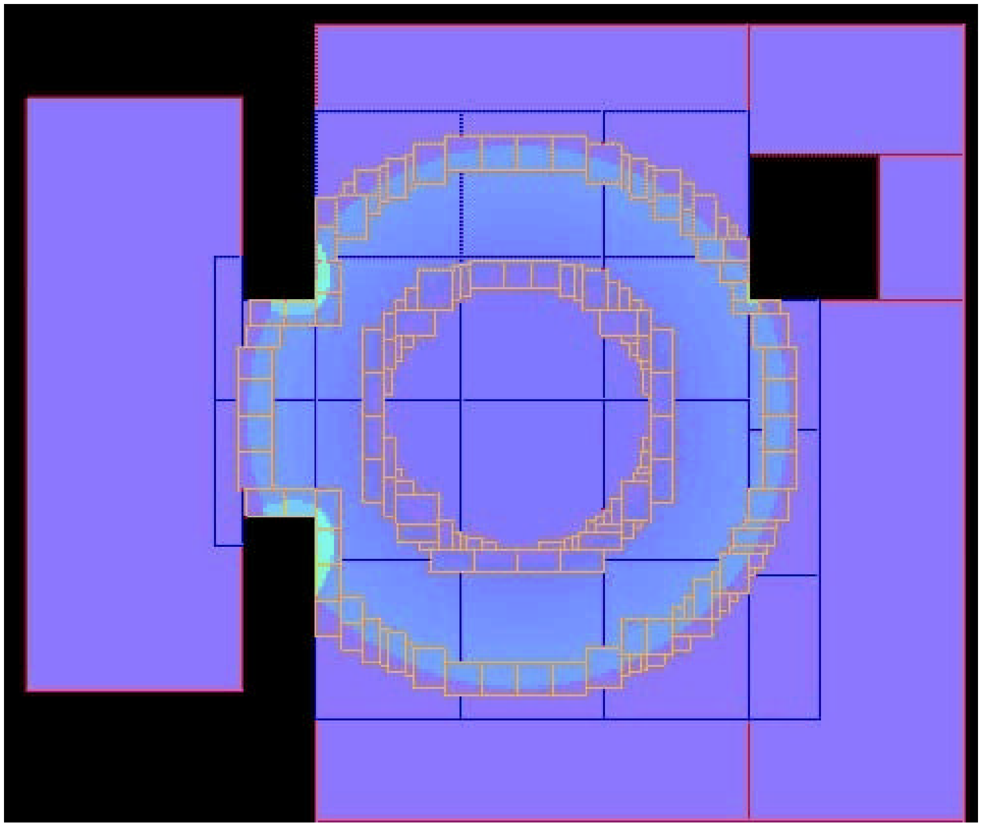
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

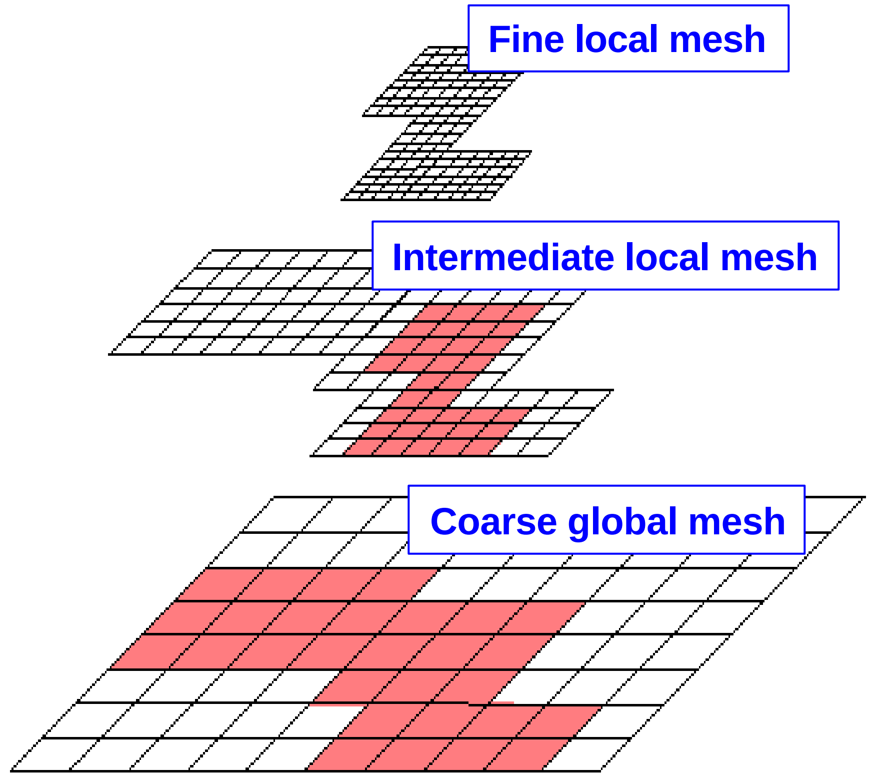
www.llnl.gov/CASC/SAMRAI
hornung@llnl.gov

Workshop on Object-Oriented and Component Technology for Scientific Computing
July 23-25, 2001

Center for Applied Scientific Computing

Structured AMR (SAMR) employs a dynamically adaptive "patch hierarchy"





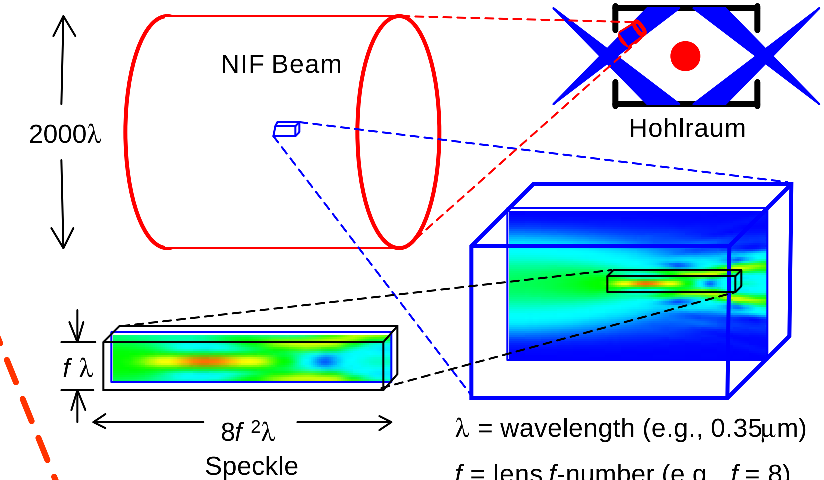
- **Hierarchy defines nested levels of varying mesh resolution (space & time)**
- **Data is stored on patches covering logically-rectangular regions in index space**

SAMRAI: Structured Adaptive Mesh Refinement Application Infrastructure

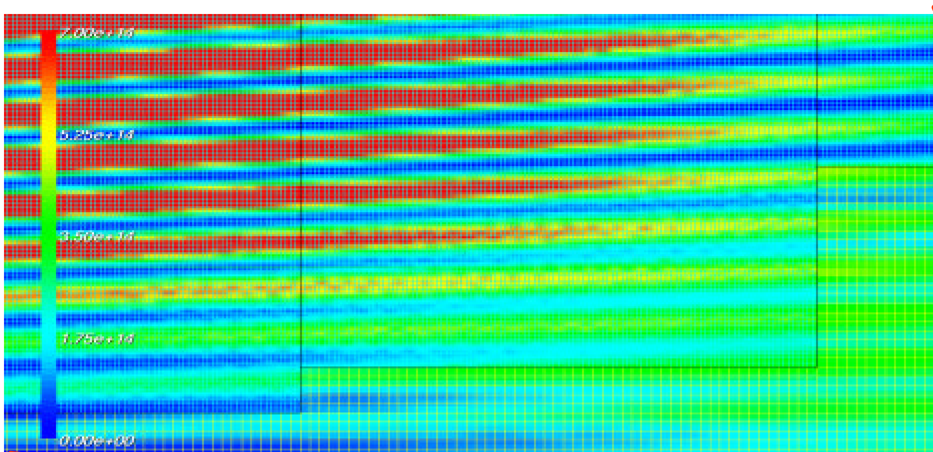
- **SAMRAI is an object-oriented (C++) software framework for SAMR multi-physics application development**
- **Design reflects survey of long-term LLNL AMR interests**
- **Research base for application, algorithm, numerical, software, parallel computing issues in SAMR**
- **Application-software feedback loop is main driver**
  - SAMRAI team: expertise in SAMR algorithms, numerics, software
  - app. collaborators: experts in numerical, physical,... problem issues
  - applications push SAMR technology into *new problem domains*
  - new capabilities are folded back into framework

ALPS is an advanced simulation tool for laser plasma instabilities

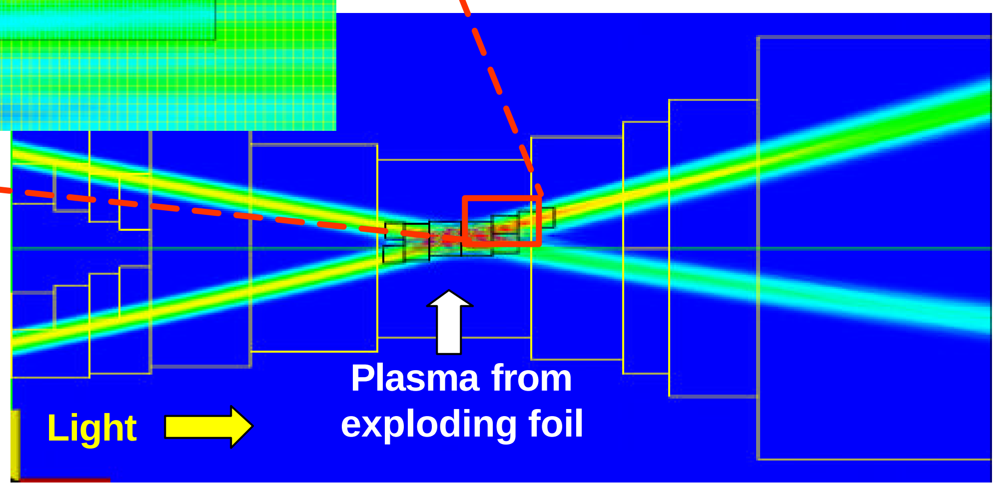
Understanding instabilities in laser-plasma interactions is critical in the design of plasma physics experiments



Numerical simulations need to accommodate multiple diverse scales



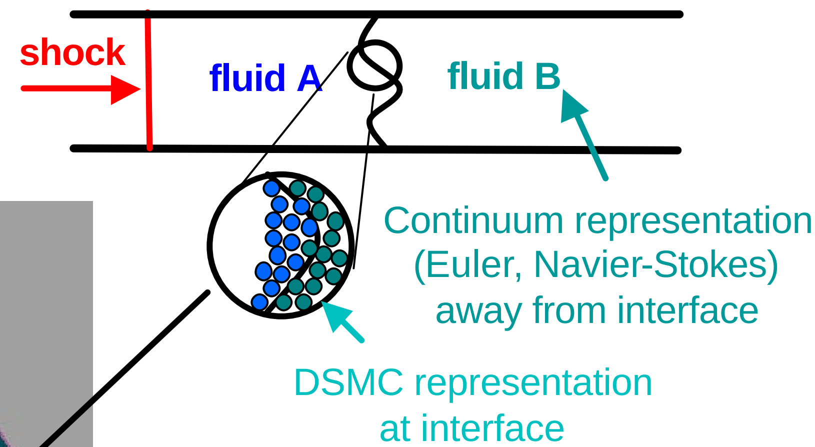
Locally refined grids resolve wave interaction where high accuracy is needed

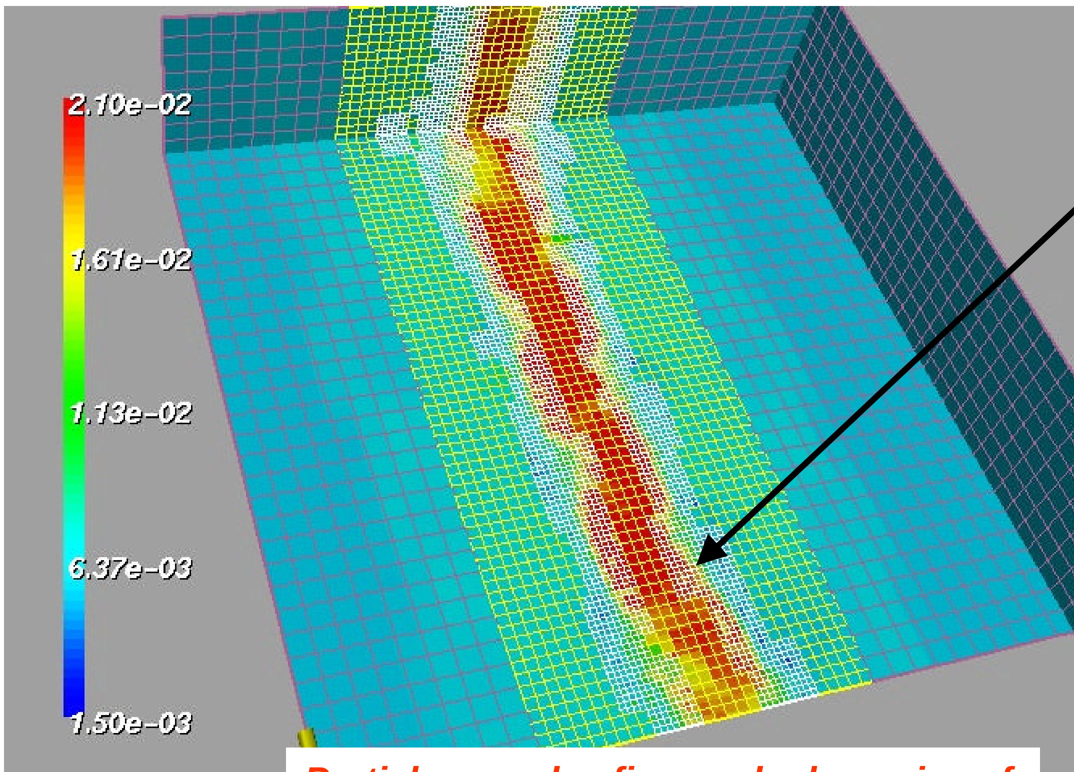


Center for Applied Scientific Computing

Hybrid continuum-DSMC AMR methods efficiently resolve interface dynamics

Interface instability problems (e.g., Richtmyer-Meshkov) involve coarse-scale hydrodynamic transport and fine-scale molecular diffusion





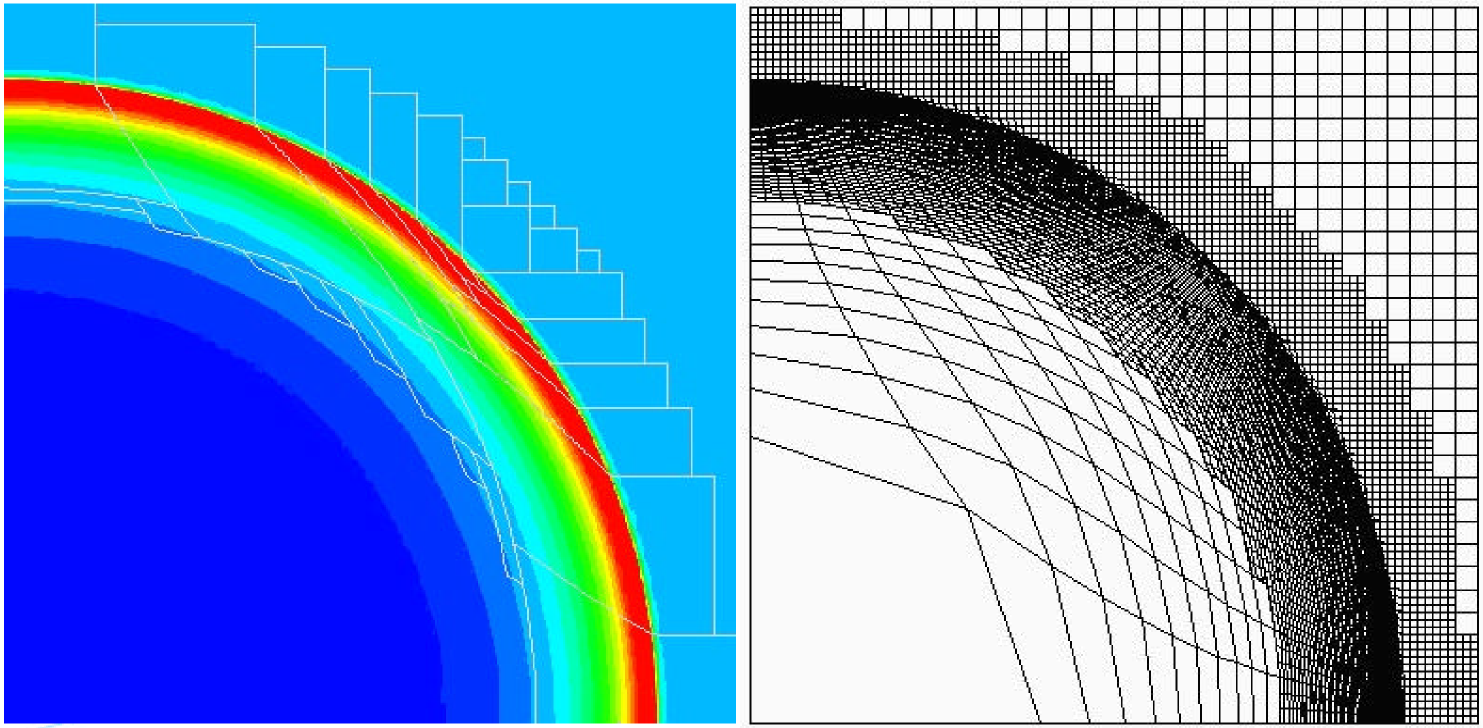
Particles resolve fine-scale dynamics of mixing region in an adaptive calculation

- **interface region grows and moves as instability evolves**
- **standard CFD simulation of turbulent mixing is limited by finest mesh scale**
- **particle resolve molecular behavior but are too expensive for large domains**

ALE-AMR combines structured grid ALE and AMR for shock hydrodynamics

Accurate ALE simulations (e.g., detonation fronts) require dynamic concentration of mesh points

- staggered mesh formulation is challenging in AMR
- coarse and fine meshes deform at different rates



Sedov blast wave density and Lagrangian mesh

CASC
Center for Applied Scientific Computing

SAMRAI is used in diverse adaptive mesh refinement application research efforts

- **Adaptive Laser-Plasma Methods (ALPS) -- Dorr, Garaizar, Hittinger (CASC/LLNL)**
- **Continuum-particle hybrid methods -- Hornung (CASC), Garcia (SJSU)**
- **ALE-AMR -- Pember, Anderson, Elliott (CASC/LLNL)**
- **Fractures in solids -- Garaizar, Hornung, et al. (CASC/LLNL)**
- **Radiation diffusion -- Kapfer, Woodward, P. Brown (CASC)**
- **Ocean current simulation -- Wickett, Hernstein (CASC/LLNL)**
- **Industrial fire simulation -- Smith, Rawat (Univ. Utah), Wissink (CASC)**
- **Global Geospace Circulation Model (GGCM) -- Raeder, Wang (UCLA)**

SAMRAI simplifies complexity management in SAMR applications

- Software must support evolving understanding of application and numerical issues (key: proper abstractions)

- **Application folks want to do certain things easily:**
  - quickly focus on numerical routines, solution algorithms
  - manage variables between coupled numerical models
  - manipulate data on dynamically changing, locally-refined mesh (data copying, coarsening, refining, time interpolation, …)

- **SAMRAI design goals:**
  - robust code base shared by diverse, complex applications (“infrastructure” common across apps. factored into framework)
  - flexible algorithmic framework to explore new solution methods
  - extensible parallel support for general dynamic data configurations (extensity *without recompilation*; e.g. via inheritance)

User view of SAMRAI is a “toolbox” of classes for application development

~570, 000 lines
~300 classes

Integration Algorithms
- Time Refinement Algorithm
- Hyperbolic Conservation Laws
- Method-of-lines Time Integrator
- Implicit Integrator

Solvers
- Package Interfaces (PETSc, KINSOL, PVODE)
- Vector Kernel Support
- Poisson Solver (FAC & *hypre*)

Vizamrai
(visualization tools)

Patch Data √
- Cell
- Node
- Face
- Edge
- IndexData
- …

Mesh Management
- Adaptive Mesh Generation
- Uniform/Nonuniform Load Balancing

Geometry
- Cartesian Grid
- Refine/Coarsen Operators
- Time Interpolation

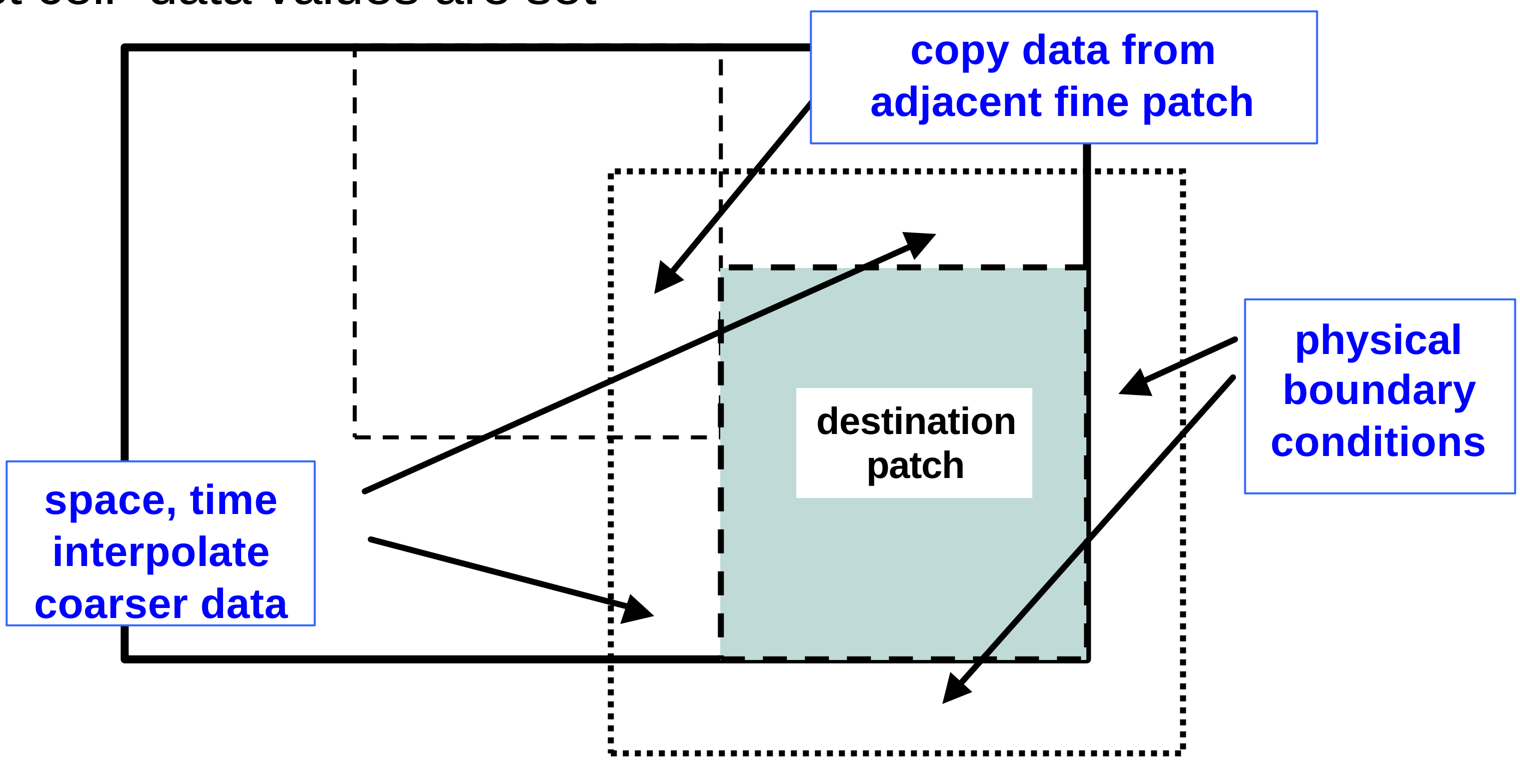
Transfer √
- Data Coarsen
- Data Refine & Boundary Fill
- Communication Schedules
- Message Streams

Hierarchy √
- Boxes
- Patches
- Variables
- Variable Database

Tool Box
- Input Database
- Restart Database
- Performance Timers
- Smart Pointers
- Container Classes

Data manipulation is dictated by solution algorithm and application needs

For example, before performing numerical operations on a patch, "ghost cell" data values are set



Typical SAMR data movement involves arbitrary combinations of variable quantities and operations

SAMRAI abstractions capture application features and simplify data management

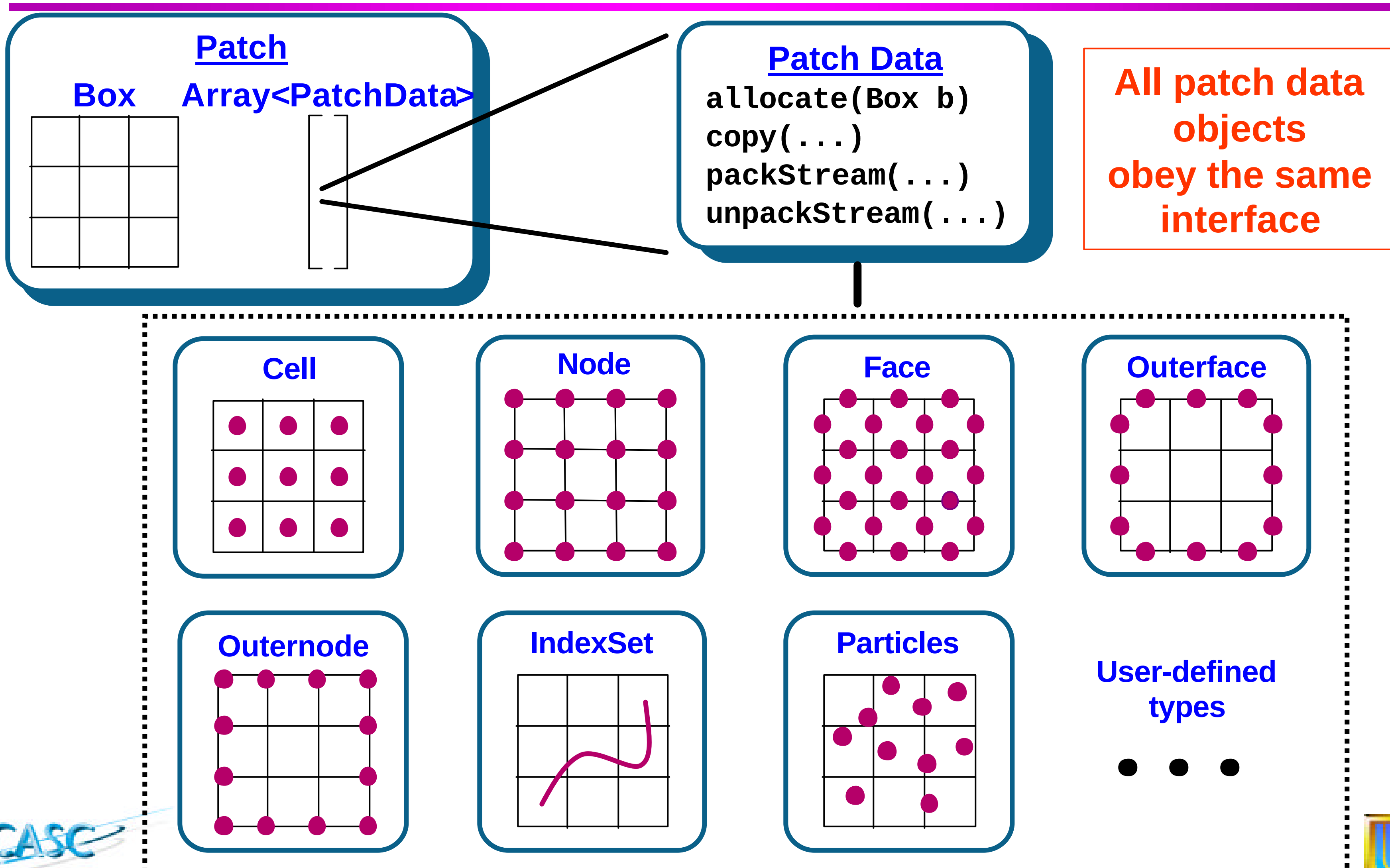
Solution algorithms tend to be <u>static</u>

- Variable
  - — defines a data quantity; centering, type, …
  - — creates data object instances (*abstract factory*)
- Communication algorithm
  - — describes data transfer phase of a computation
  - — expressed using variables, operators, …
  - — independent of mesh

Mesh and data objects tend to be <u>dynamic</u>

- Patch data
  - — represents data on a “box”
  - — interface for data communication (*strategy*)
  - — created by factory defined by variable
- Communication schedule
  - — manages details of data movement
  - — depends on mesh
  - — created by communication algorithm

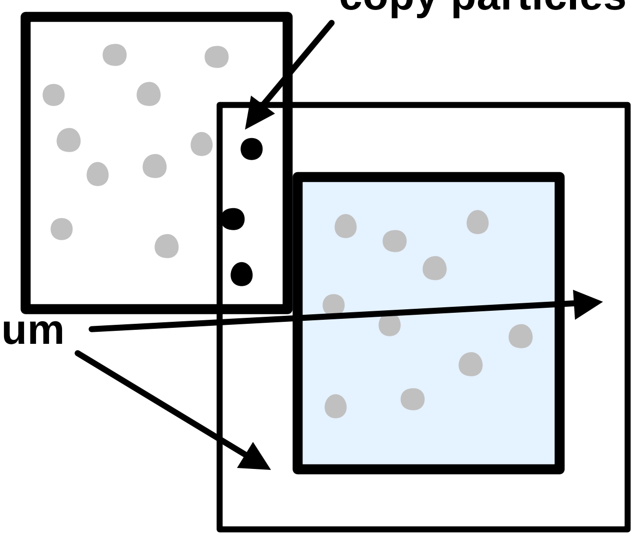
SAMRAI "patches" contain all data living on a region of the computational mesh



Center for Applied Scientific Computing

Communication algorithms describe data transfers needed for solution method

For example, integration of particle regions requires both continuum and particle boundary data for each patch



- **Create algorithm to fill data**

```
RefineAlgorithm  fill_alg;
```

- **Register variable operations with algorithm:**
  - **density refined from coarser, copied from fine, BCs set**

```
fill_alg.registerRefine(rho_old,                  // destination
                        rho_old, rho_new,         // sources
                        ..., "CONSERVATIVE_INTERP");
```

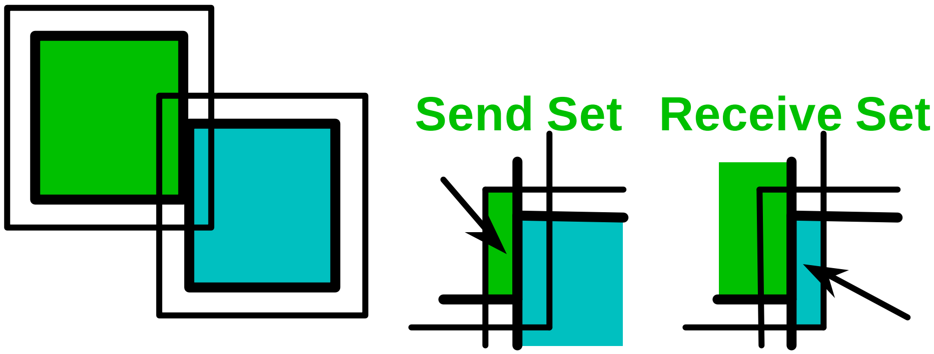
  - **particles copied from neighboring patches**

```
fill_alg.registerRefine(particles,   // destination
                        particles,   // source
                        ...);
```

Communication schedules create and store data dependencies

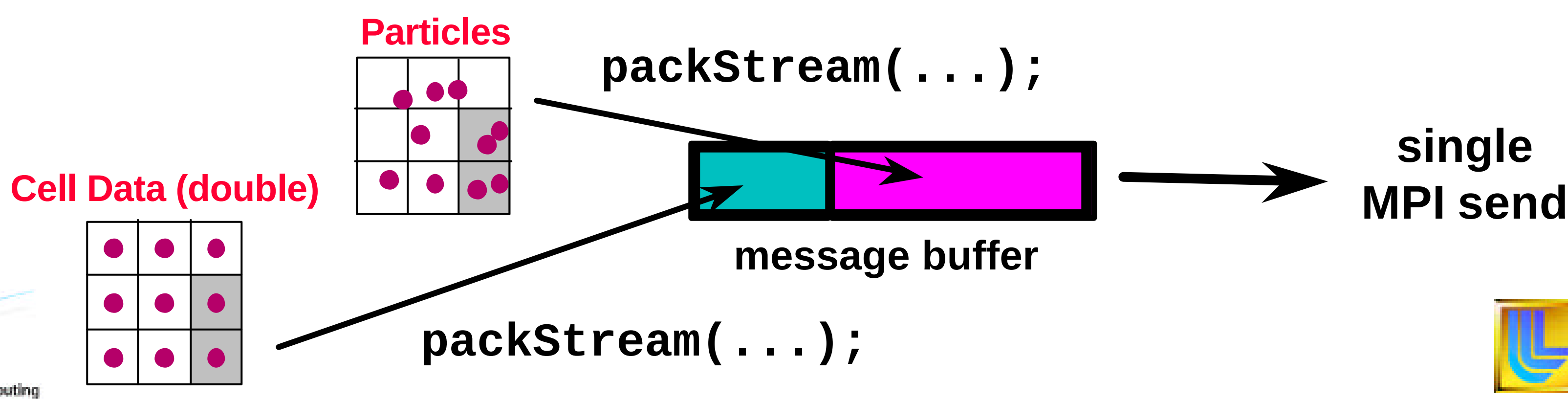
- Amortize cost of creating send/receive sets over multiple communication cycles

  - Create schedule to fill data
```
RefineSchedule fill_sched =
  fill_alg.createSchedule(
      hierarchy, level, ...);
```



- Data from multiple sources is packed into one message stream

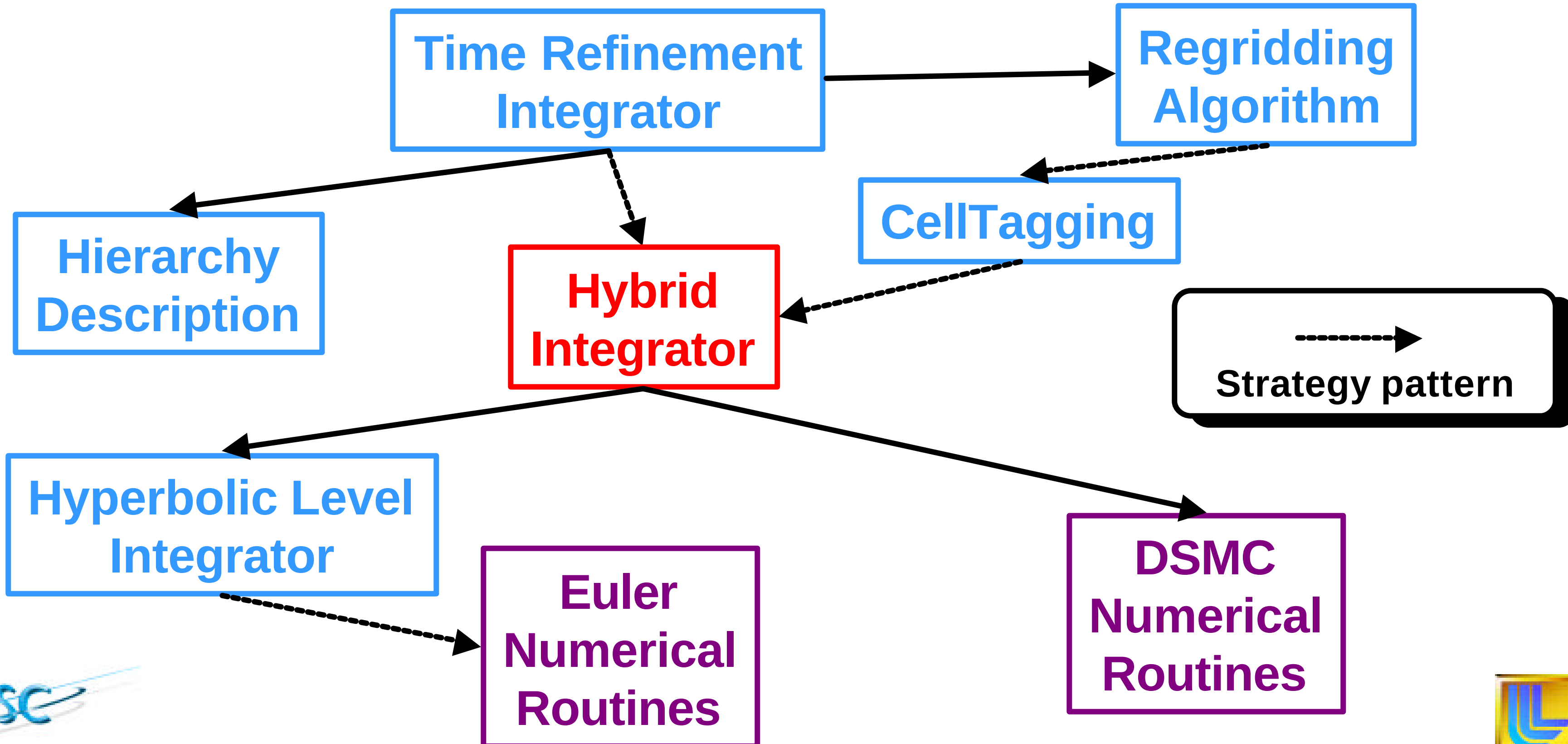
  - Invoke data fill operations
```
fill_sched.fillData(time, ...);
```



Analysis of SAMRAI OO design features...

- **Uncoupling variable, data, communication, mesh provides a lot of flexibility**

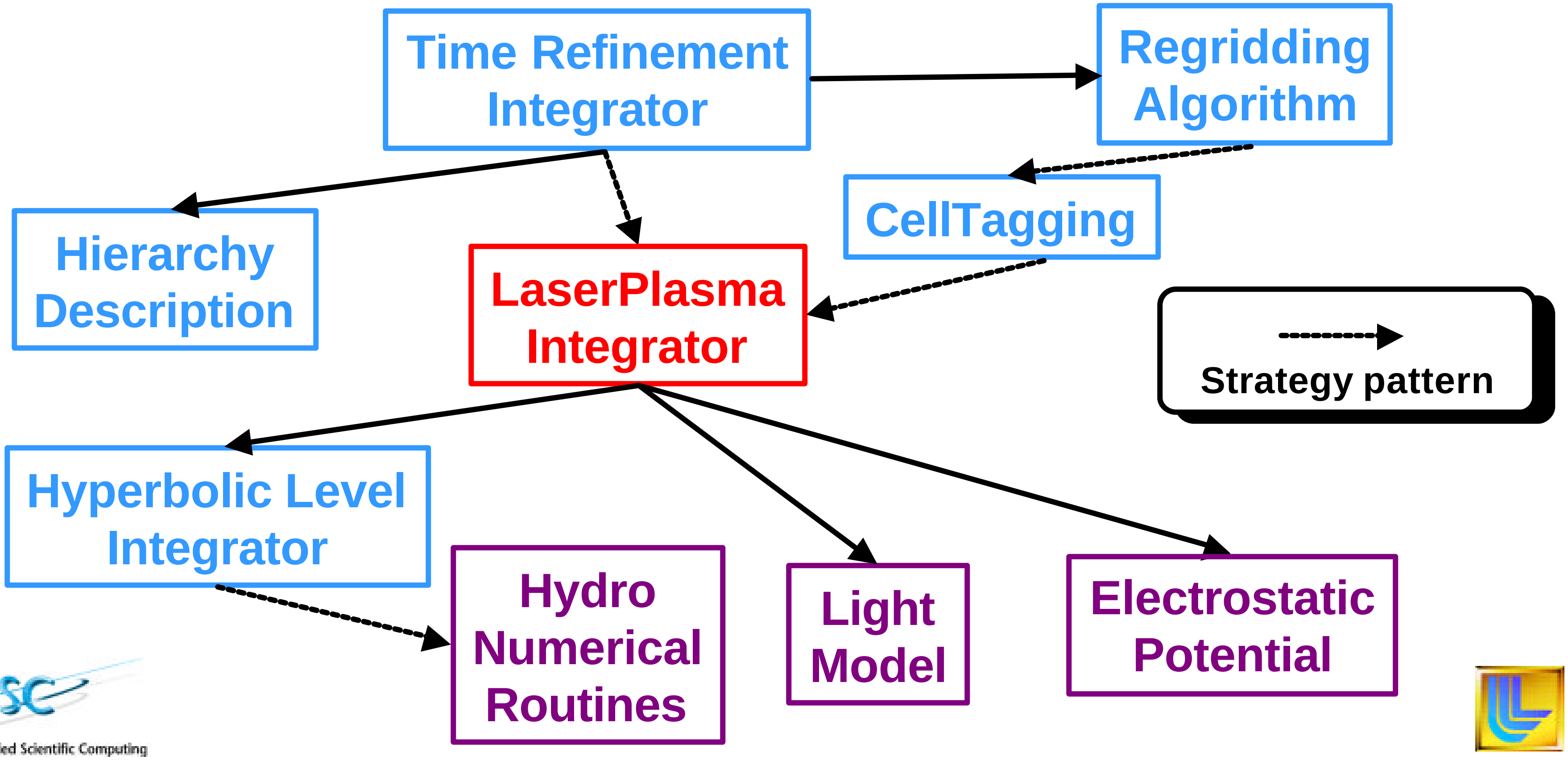
  "Strategy" supports extensible, specializable algorithms



Analysis of SAMRAI OO design features...

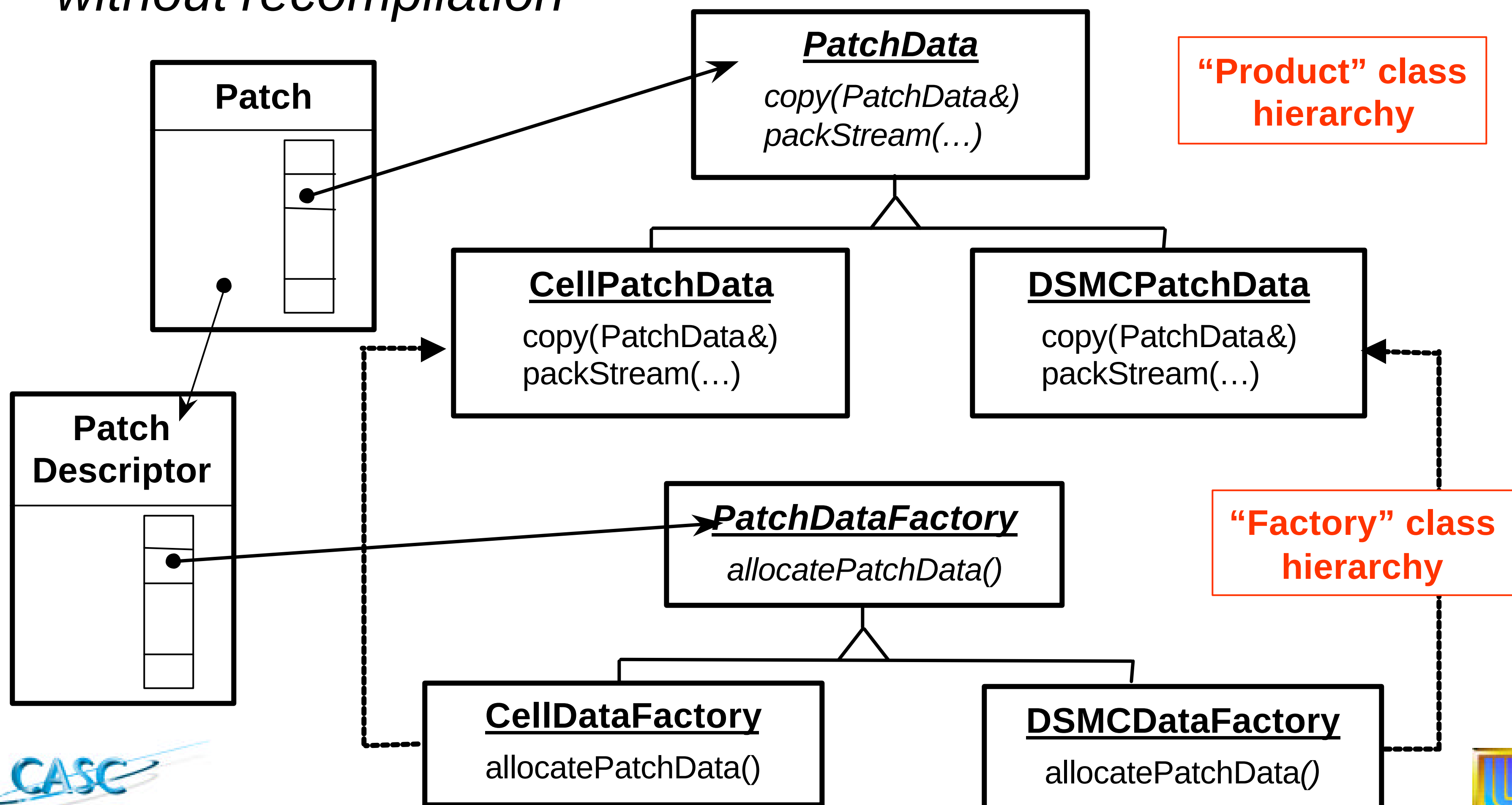
- **Uncoupling variable, data, communication, mesh provides a lot of flexibility**

  "Strategy" supports extensible, specializable algorithms



Analysis of SAMRAI OO design features...

"Factory" pattern supports user-defined data types without recompilation



Center for Applied Scientific Computing

SAMRAI parallel framework supports new patch data types without recompilation

- Create a DSMCData subclass and provide virtual functions

```
class DSMCData : public PatchData
{
      void copy(...);
      void packStream(...);
      int getDataStreamSize(...)
      . . .
};
```

- Create a DSMCFactory subclass to allocate DSMCData objects

```
class DSMCFactory : public PatchDataFactory
{
      Pointer<PatchData> allocate(...);
      . . .
};
```

- Create DSMCVariable subclass to create DSMCFactory objects

```
class DSMCVariable : public Variable
{
      Pointer<PatchDataFactory> getPatchDataFactory(...);
      . . .
};
```

Analysis of SAMRAI OO design features...

- **Design patterns, other OO techniques useful to manage complexity at a high level**
  - software architecture easier to understand (e.g., *strategy*, *abstract factory* ubiquitous)
  - inheritance model straightforward for users
- **Flexibility comes at a cost**
  - learning curve: pattern-based design at application level
  - uncoupling requires extra indirection
    - keep OO abstractions out of computation
    - need to understand where things are in library
  - early prototyping, user feedback essential

Auspices Statement

- This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.
- Document UCRL-PRES-144527