BERKELEY LAB

LAWRENCE BERKELEY NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

Software Engineering and Architecting Scientific Codes

Anshu Dubey

ATPSEC 2013

Software Process Components

- For All Codes
  - Code Repository
  - Build Process
  - **Code Architecture**
  - Coding Standards
  - Verification Process
  - Maintenance Practices
- If Publicly Distributed code
  - Distribution Policies
  - Contribution Policies
  - Attribution Policies

Code Architecture : Important Questions

- What are the essential data structures
  - State data, meta data and scratch data
- What are the different ways in which the data structures are manipulated
  - Solver operations, housekeeping, being moved around
- How do various data structures interact with each other
  - What metadata needed to correctly change state data
  - How much scratch space is needed, where can it be reused
  - What are the data dependencies
- Where are the firewalls between who can use what data and how
  - Which part of the data can be modified by which solver
  - Which variables can only be modified by global state change
  - How should the data be scoped

Code Architecture : Considerations

- Identify logically separable functional units of computation
- Encode the logical separation (modularity) into a framework
- Separate what is exposed outside the module from what is private to the module
- Define interfaces through which the modules can interact with each other
- Devise control flow – the driver

While these are good principles to start with, they don't always work out easily. It may become difficult to untangle the data dependencies or modularity might dictate code replication. This is where design really becomes important.

FLASH Example

- Requirements for infrastructure support:
  - AMR, and also preferably Uniform Grid
  - Input runtime parameters
  - IO
  - Support for multiple species, physical constants etc
- Physics requirements
  - Shock hydrodynamics /MHD
  - Nuclear networks
  - Equation of state and other material properties
  - Time-stepping
  - Lagrangian particles

Classes of Units

- Infrastructure Units – do all the housekeeping
- Physics units – physics specific implementations
- Monitoring units – log the progress and performance statistics of the run
- Driver unit – implement time stepping and orchestrate the run
- Simulation unit – the most specialized unit
  - Define the application
  - Specify needed units and their specific implementations
  - Define initial conditions and provide boundary conditions if needed
  - Mechanism for customization

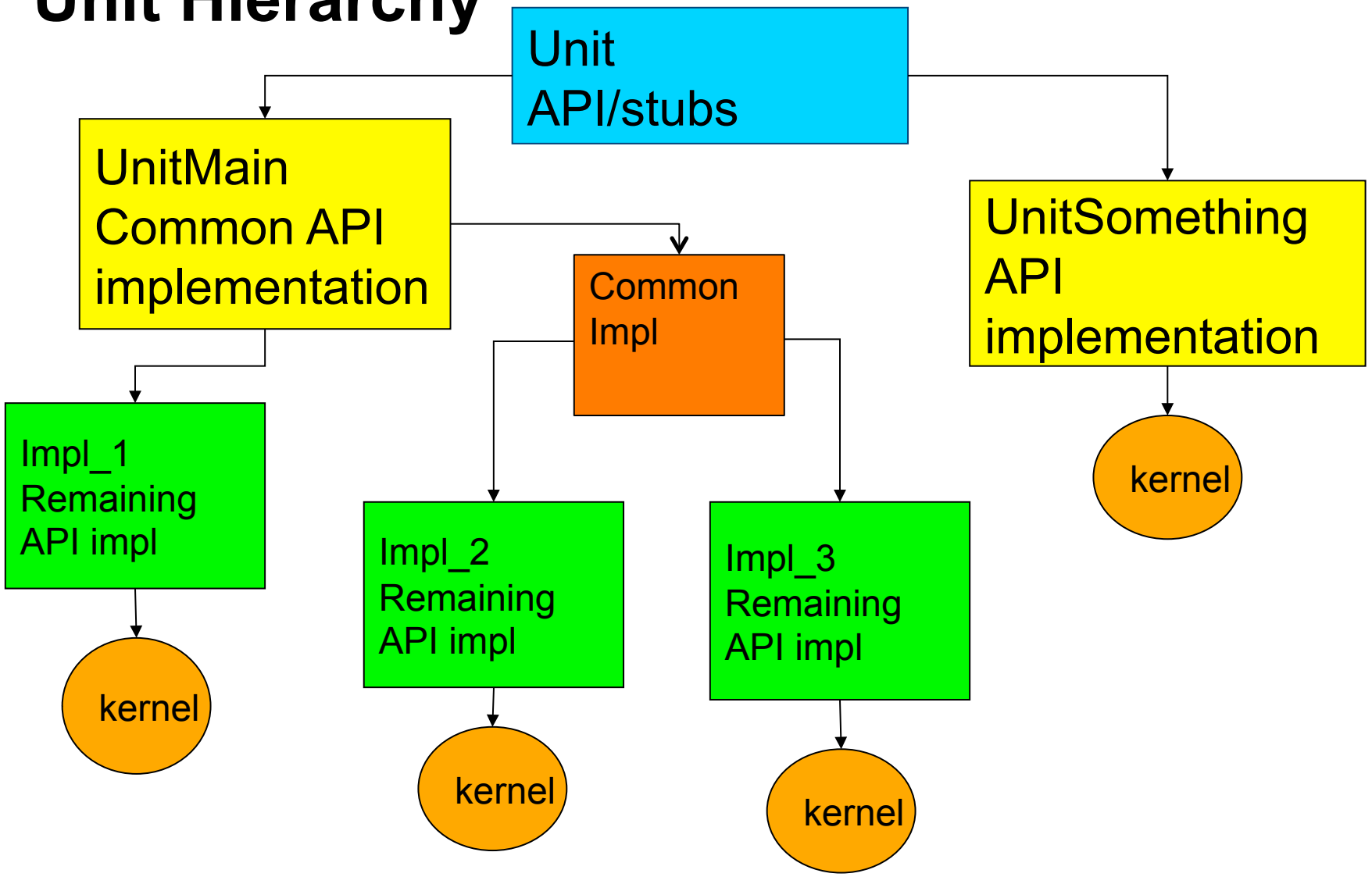
Architecture : Unit

- FLASH basic architecture unit
  - Different combinations of units are used for particular problem setups
  - Publishes a public interface (API) for other units' use.
  - Ex: Driver, Grid, Hydro, IO etc
  - Unit can have subunits
- Interaction between units governed by the Driver
- Not all units are included in all applications
  - Not all subunits of an included unit need to be included in all applications

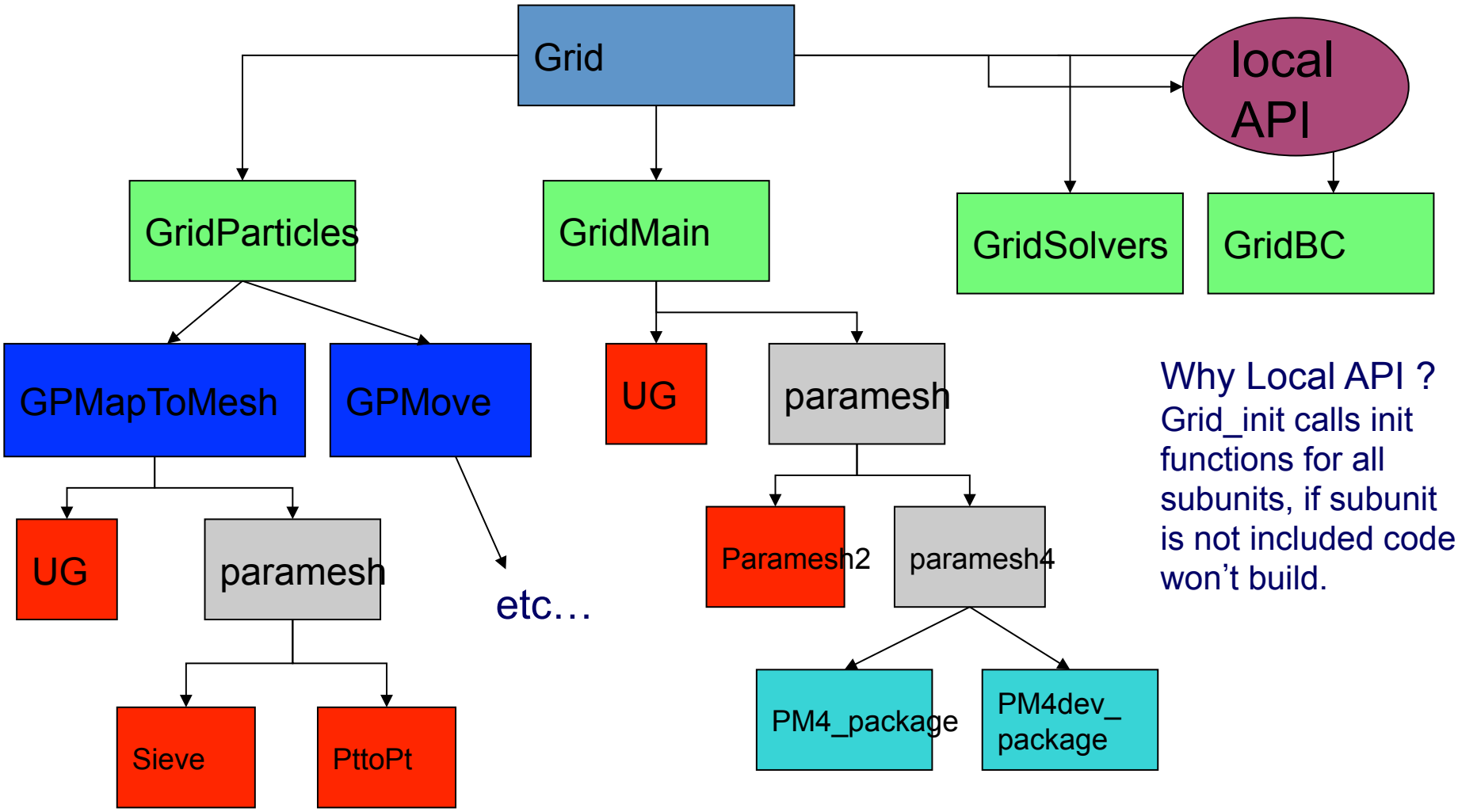
Data Management

- Defined constants for globally known quantities
- Data ownership by individual units
  - Arbitration on data shared by two or more units
- Definition of scope for groups of data
  - Unit scope data module, one per implementation of the unit
  - Subunit scope data module, one per implementation of the subunit
  - All other data modules follow the general FLASH inheritance
    - The directory in which the module exists, and all of its subdirectories have access to the data modules
- Other units can access data through available accessor functions
- For large scale manipulations of data residing in two or more units, runtime control transfers back and forth between units
  - Avoids lateral transfer of large amounts of data
  - Avoids performance degradation

Unit Hierarchy



Example of a Unit – Grid (simplified)



Why Local API ?
Grid_init calls init functions for all subunits, if subunit is not included code won't build.

Example of Unit Design

- Non trivial to design several of the physics units in ways that meet modularity and performance constraints.
- Eos (equation of state) unit is a good example
  - Individual mesh points are independent of each other
  - There are several reusable calculations
  - Other physics units demand great flexibility from it
    - single grid point
    - only the interior cells, or only the ghost cells
    - a row at a time, a column at a time or the entire block at once
    - different grid data structures, and different modes at different times
  - Implementations range from simple ideal gas law to table look up and iterations for degenerate matter and plasma, with widely differing relative contribution in the overall execution time
  - Relative values of overall energy and internal energy play role in accuracy of results
  - Sometimes several derivative quantities are desired as output

EOS interface Design

- Hierarchy in complexity of interfaces
  - For single point calculation scalar input and output
  - For sections of a block or full block vectorized input and output
    - wrappers to vectorize and configure the data
    - returning derivative quantities if desired
- Different levels in the hierarchy give different degrees of control to the client routines
  - Most of the complexity is completely hidden from casual users
  - More sophisticated users can bypass the wrappers for greater control
- Done with elaborate machinery of masks and defined constants

Functional Component in Multiple Units

- Example Particles
  - Position initialization and time integration in Particles unit
  - Data movement in Grid unit
  - Mapping divided between Grid and Particles
- Solve the problem by moving control back and forth between units