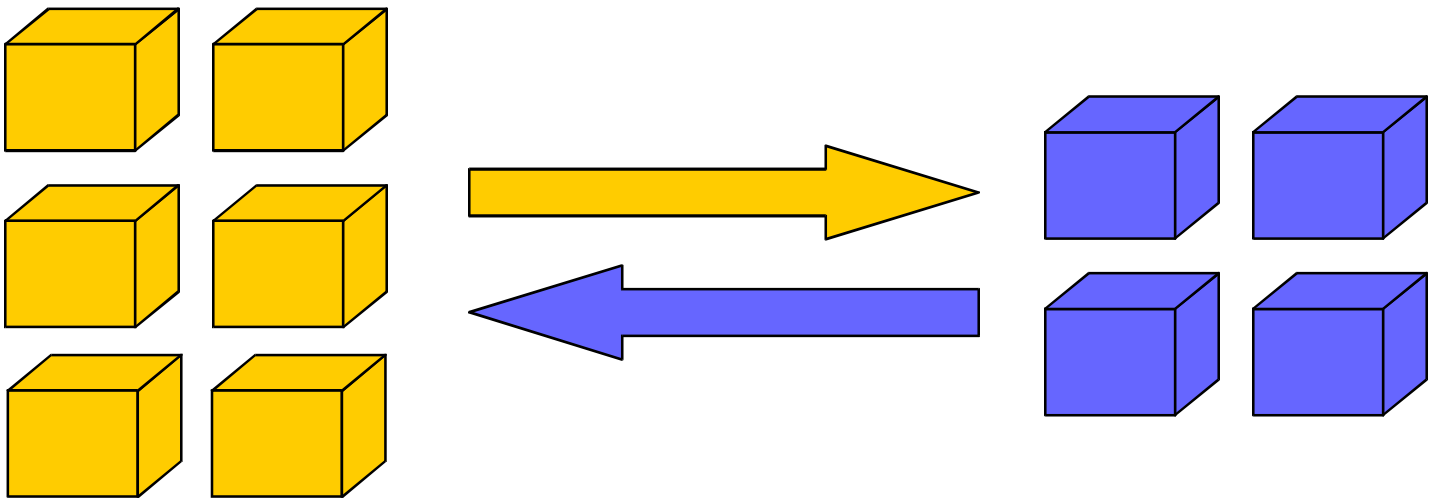
Efficient Coupling of Parallel Scientific Applications Using PAWS

Pat Fasel, Sue Mniszewski,
Craig Rasmussen

Advanced Computing Laboratory
Los Alamos National Laboratory

Primary Goal of PAWS

- Provide the ability to share data between two separate, parallel applications, using the maximum bandwidth available

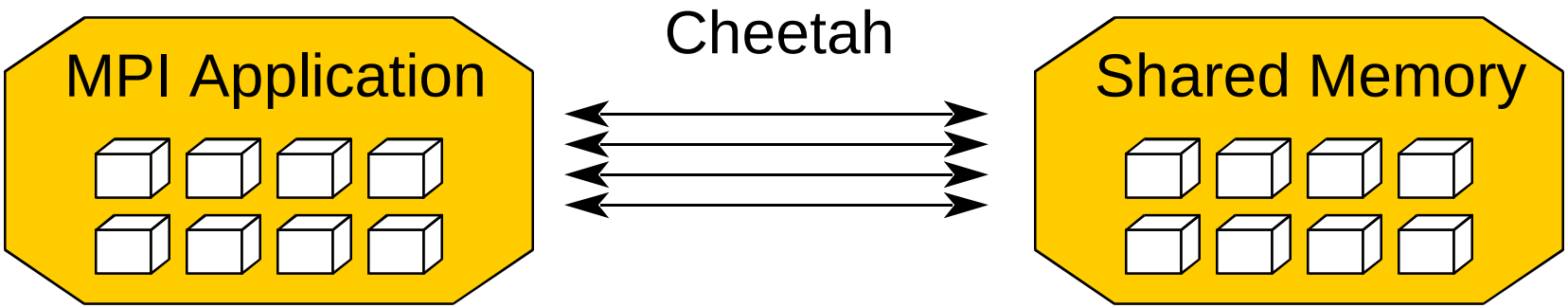


Challenges

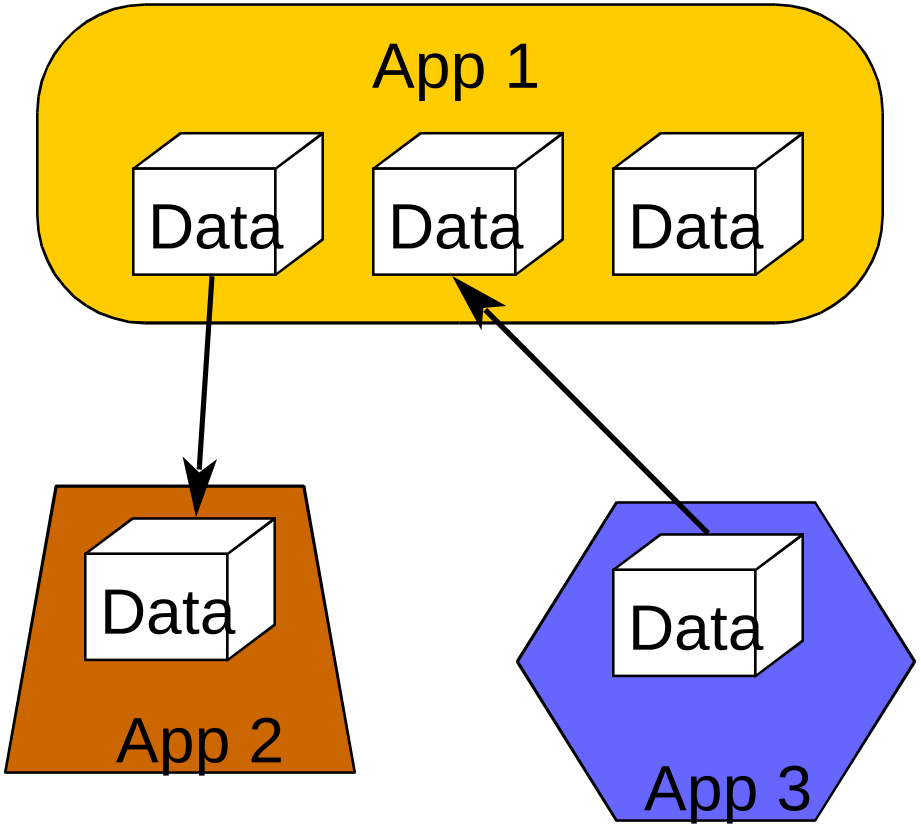
- Dynamic coupling of parallel applications
- Avoid serialization
- Be as non−invasive as possible
- Connect parallel applications with unequal numbers of processes
- Support different programming languages, Fortran, C, and C++

Interprocess Communication

- Currently uses NEXUS/GLOBUS from Argonne National Laboratory
- Will use Cheetah (MPI, shared memory, HiPPI)
- Used for inter−application communication
  - Applications can use other communication methods

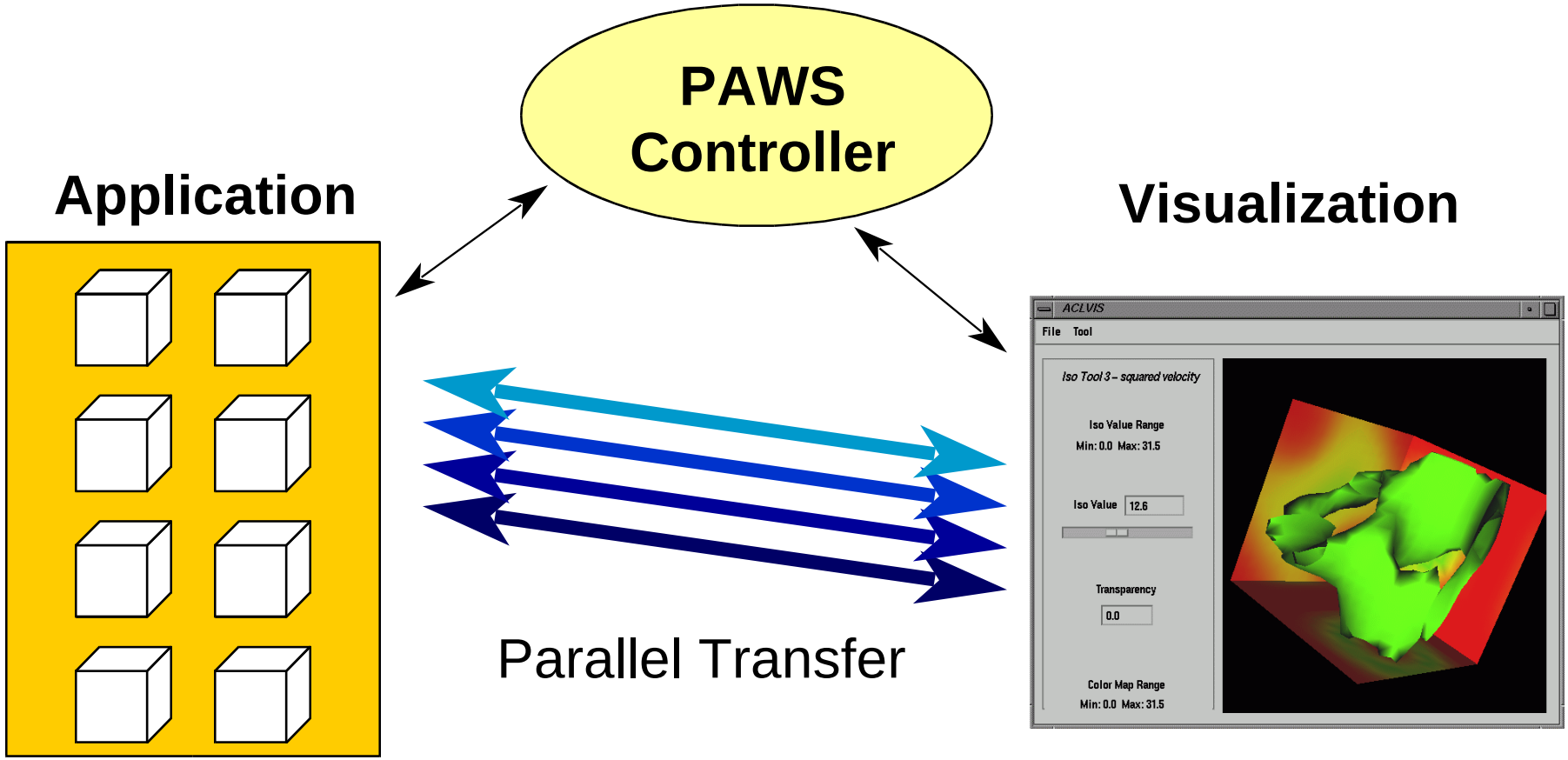


Data−Centric View

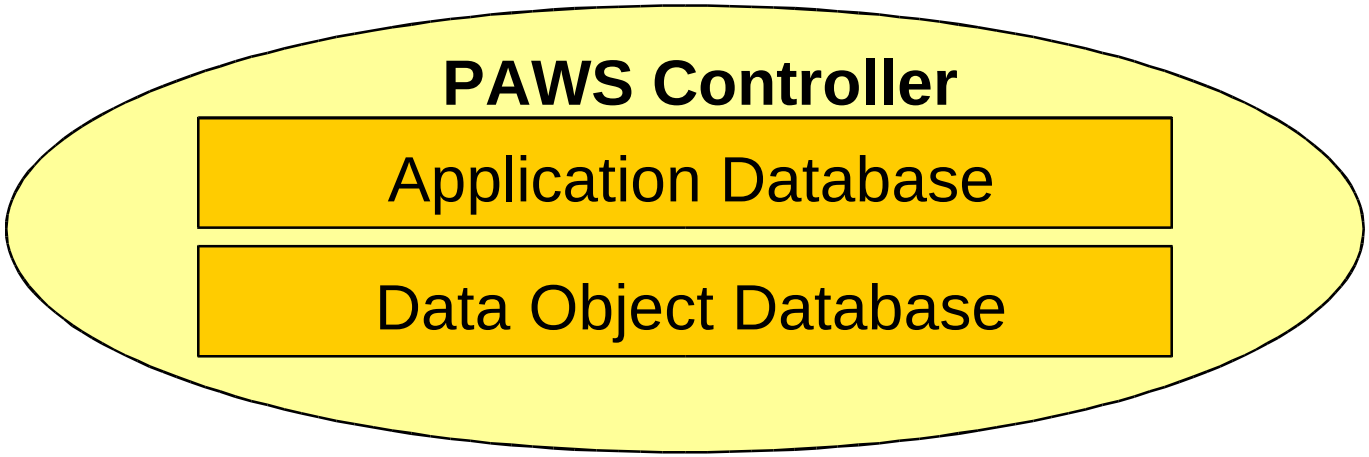


- PAWS views world as applications with data objects. API used to:
  - Register applications
  - Register data objects
  - Query database about registered items
  - Connect <u>data objects</u> between apps

Sample PAWS Environment



PAWS Controller



- Repository with registered applications
- Repository with registered data objects
- Initialize connection between apps, data objects
- Can launch apps or use dynamic attach/detach

PAWS Application

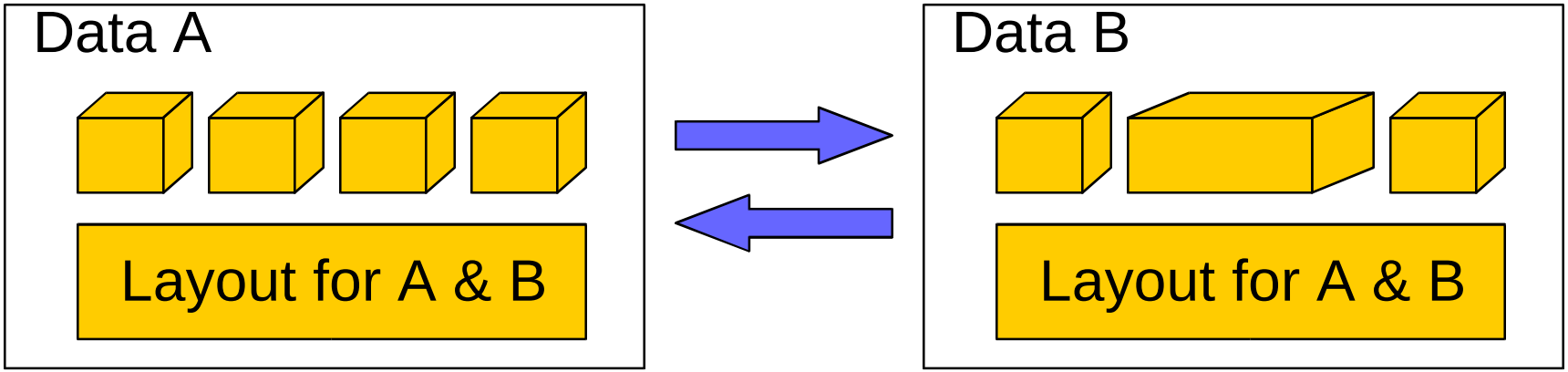
- Must be told where to find the Controller
- Can be written in Fortran, C, or C++
- Use PAWS API to:
  - Register app and data with Controller
  - Query Controller information repositories
  - Make connections to other apps and data
  - Send/Receive data
  - Disconnect data or app from Controller

What Data Can Be Shared

- Fixed−size, regular, parallel multidimensional arrays
  - Arbitrary number of sub−blocks across nodes
- Variable−size parallel multidimensional arrays
  - May change in total size or parallel distribution
- Extensible to other parallel data structures
  - POOMA framework includes extensions to share multidimensional data with other apps using PAWS

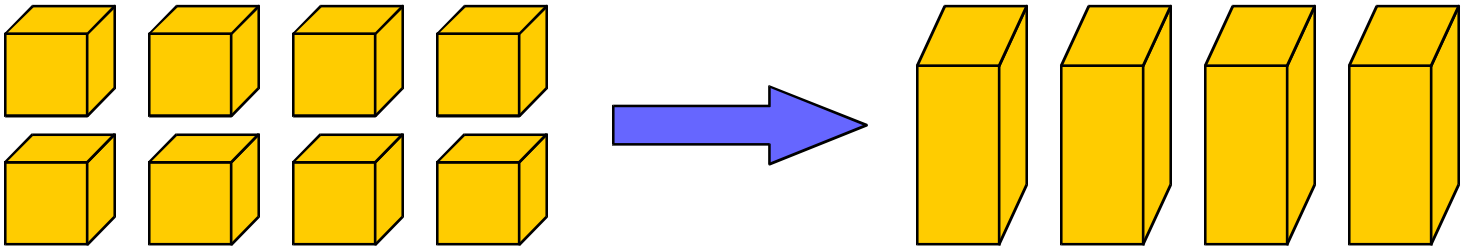
How Do You Connect Data

- Step 1: applications register with controller
- Step 2: applications register data
  - Must provide data type and parallel layout info
- Step 3: applications are told to connect data

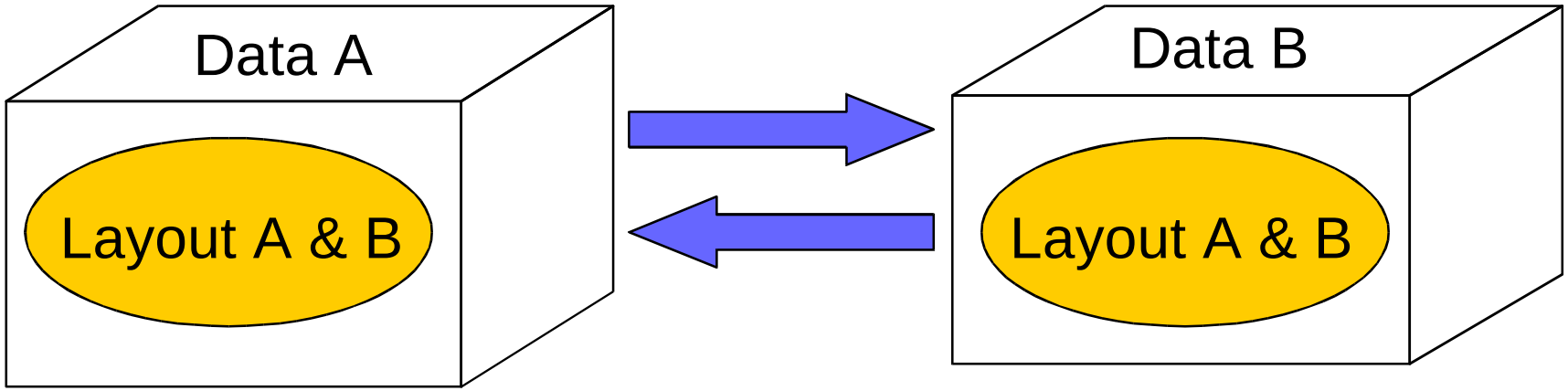


Parallel Redistribution

- Each application can partition data differently
  - Different numbers of processes
  - Different parallel layouts



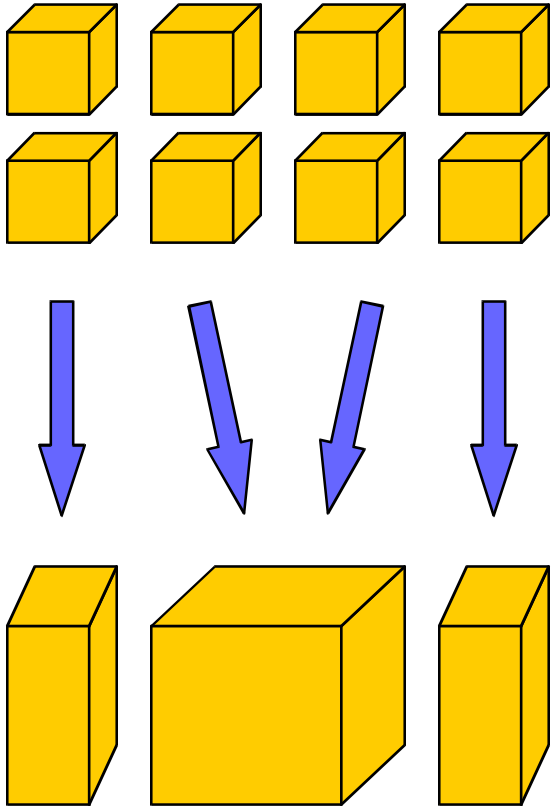
Parallel Redistribution (cont)



- Registered data objects exchange layout information when connected or changed
- PAWS computes communication schedule for data redistribution

Communication Schedule

- Computed during connection
- Recalculated when layouts change
- Use maximum available channels to avoid serialization



When Do You Transfer Data

- Applications must indicate when data is ready to be sent or received
  - At start or end of computation
  - At each iteration
- PAWS API used to indicate when data is ready to be transferred
- Connections can be either synchronous (with or without timeout) or asynchronous

PAWS Code Example

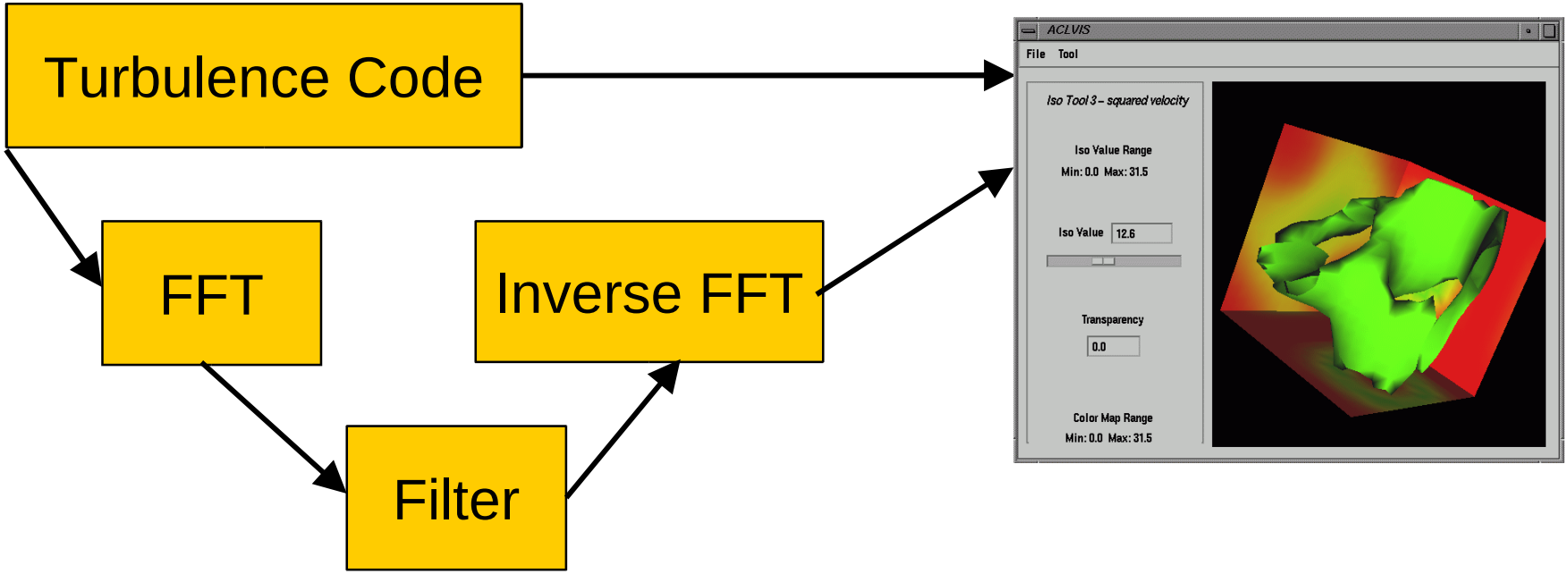
```
paws_init("diffusion", ...);                    // initialize app for PAWS use

paws_data("diff_A", A, ...);                    // register data

paws_ready();                                   // rendezvous with other apps

do n times {
    // perform some calculation
    paws_send("diff_A");                        // send data to other app
}

paws_finalize();                                // shutdown
```

Common Component Architecture (CCA)

- Multi−lab effort to define a component model for high−performance applications