CCA Collective Ports Specification

Jimbo “Speedo” Kohl
Oak Ridge National Laboratory
August 26, 1999

Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

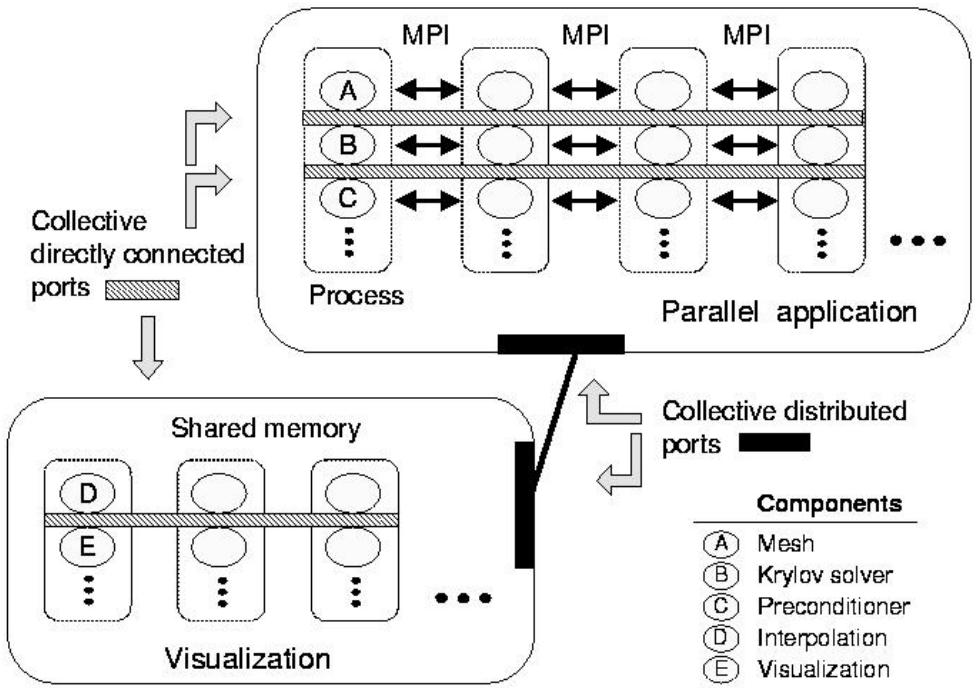
Collective Ports

- Extension of Basic Ports Model
  - ⇒ For Parallel Components, Distributed Data
- Extract & Assemble Subsets of Data Fields
  - ⇒ Online Visualization, Model Coupling
- Need Map of Data Layout / Decomposition
  - ⇒ Automate Data Collection (Hide Parallel Details)
  - ⇒ Support Interpolation or Re-Distribution of Data
  - ⇒ Improve Usability of Data (Data Semantics)

Arbitrary Data Distributions

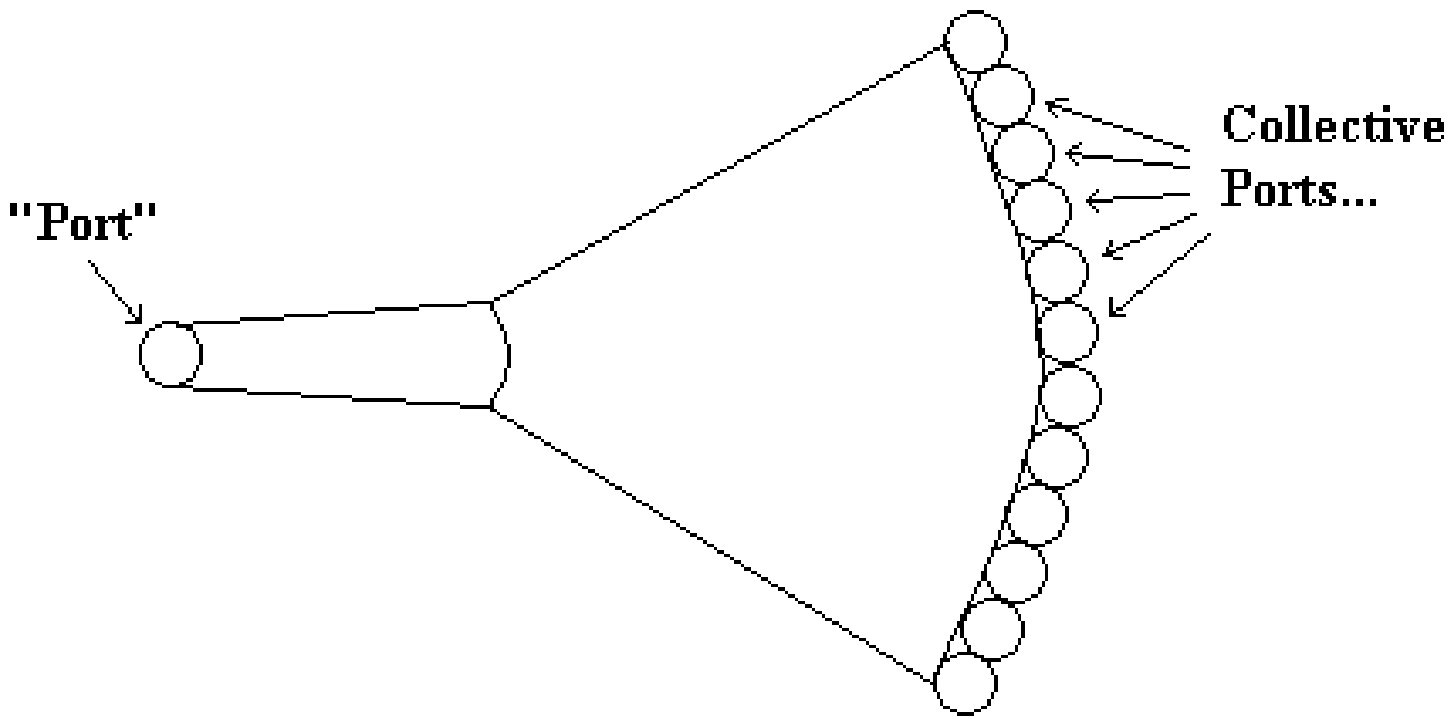
- Mixtures: NxN, 1xN, NxM, Dynamic

  ⇒ Varying Models: Distributed / Shared Memory



Two Views of a Collective Port

- Front-End View ~ Just Looks Like a “Port”
- Back-End View ~ Parallel Collection of Ports



Base Port Information

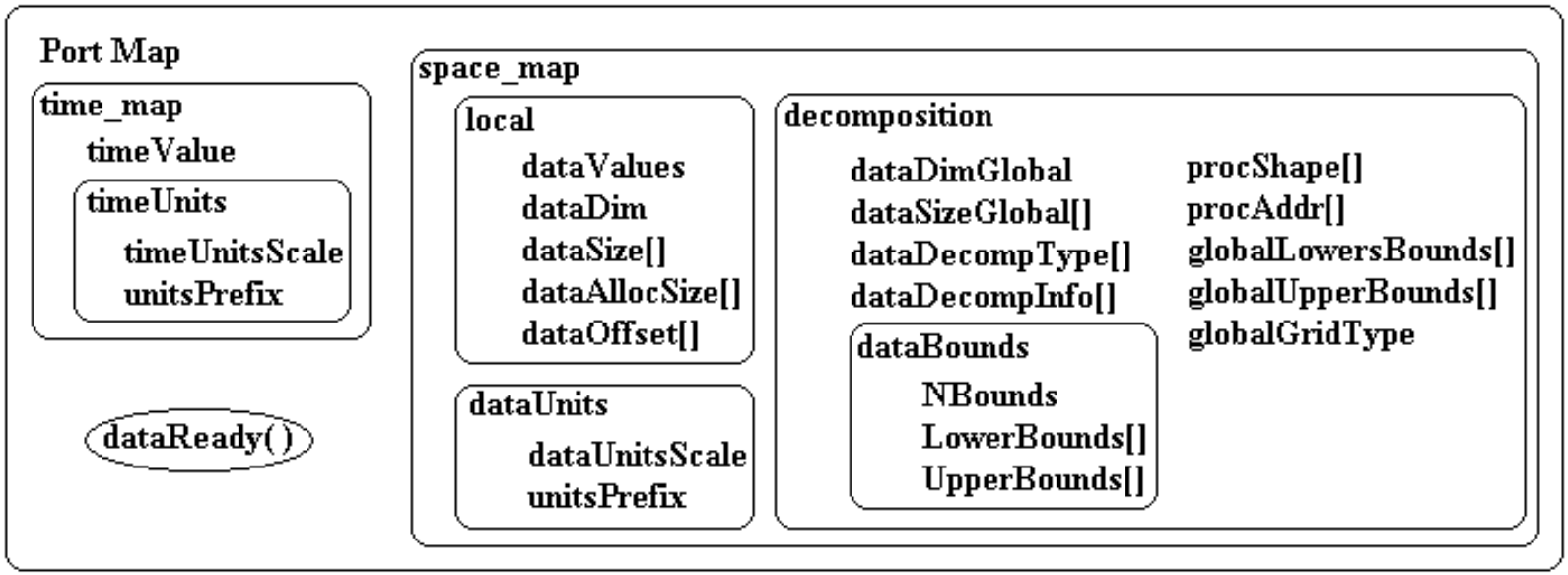
- ***Name*** ~ Semantic or Logical Description
- ***Data Type*** ~ Raw Data Format

$\Rightarrow$ For Generic Handling of Transfers

$\Rightarrow$ No Knowledge of Parallel Context Required

Collective Port Map

- ***Time Map*** ~ Time Scale of Data Snapshots
- ***Space Map*** ~ Spatial Organization of Data



Time Map

- Ports Produce Sequence of Data Snapshots
  - ⇒ Each Frame Has Associated "Time"…
    - → Explicit or Implied…
- Scientific Simulations
  - ⇒ Passage of Simulated Time, Time Steps
- Iterative Computations
  - ⇒ Simple Iteration Counts, Toward Convergence
- Static Data Sources
  - ⇒ Effective Time Scale of Zero.

Specifying Time Values

- Time Value with Units and Scale Factor

  ⇒ Immense Range of Times, High Precision, Good Redundancy for Convenience…

```
struct  time_map {
    float  timeValue;
    struct  timeUnits {
        enumerated  timeUnitsScale =
            { “seconds,” “minutes,” “hours,” “days,”
                “weeks,” “years,” “decades,” “centuries” };
        enumerated  unitsPrefix =
            { “pico,” “nano,” “micro,” “milli,” “centi,” “deci,”
                “deca,” “kilo,” “mega,” “giga,” “tera,” “peta” };
    };
};
```

When Can Data Be Extracted?

- Snapshots More Often Than Whole Iteration?
  - ⇒ Debugging, Sub-Steps of Computation…
- Component Must Choose Minimal Time Scale
  - ⇒ Nested Components?
- “Consistent” Data Snapshots?
  - ⇒ Application Must Define “When” & “Where”...

Instrument Computation

- Invoke ***dataReady***( ) Method for Port.
  - ⇒ Beginning or End of Primary Loops
  - ⇒ Other "Logical" Points in Computation…
- Bothersome and Intrusive…
  - ⇒ Though Simple and Direct!
- Convenient Hook for Non-Threaded Apps

What if the Time Scale is Dynamic?!

- Multi-Phase, Multi-Level or Non-Traditional
- Each Data Frame Needs Unique Time Map

  ⇒ Pass Optional "***struct time_map***" Argument to ***dataReady***( ) …
- Gives Sequence of Time Values

  ⇒ Use to Decode and Interpolate Data Frames (As Needed)

Spatial Mapping

- ***Local*** ~ Allocation & Layout of Local Data
  - ⇒ Some Information Obtainable Automatically…
- ***Decomposition*** ~ Context in Global Array
  - ⇒ Semantics of Locally-Stored Portion
  - ⇒ Maps Each Local Element to a Global Element
  - ⇒ Usually Only *Implied* By Data Usage…
- Full Specification of Both Required
  - ⇒ For Framework to Handle Extraction / Collection

Local Specification

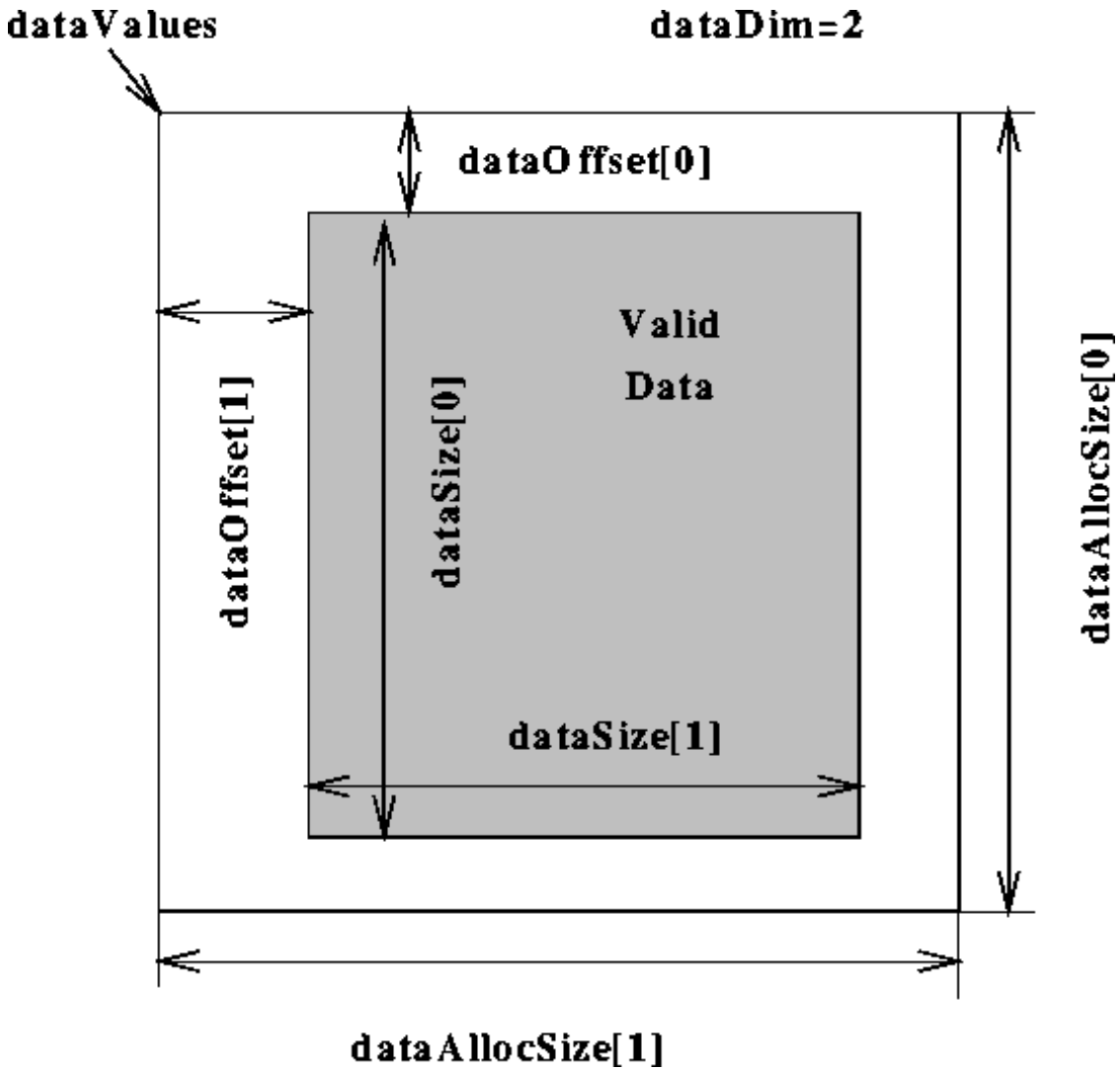
- For Locating Specific Local Data Elements
- Pointer to Actual ***dataValues***
- Local Array Size Information

  ⇒ ***dataDim***, ***dataSize***[ ]…
- Local Allocation Can Be Oversized!

  ⇒ Boundary Regions for Neighbor Communication

  ⇒ Only Use Portion of Larger Static Allocation
- So…. ***dataAllocSize***[ ] & ***dataOffset***[ ]…

Fully Specified Local Storage



Decomposition Specification

☙ Global Context of Local Data Elements ❧

- Global Array Size, Shape and Coordinates
- Decomposition Types for Each Axis
  - ⇒ With Detail Information
- Process / Thread Topology & Local Position
- Units of Actual Data

Global Array Size, Shape, Coords

- ***dataDimGlobal*** ~ Dimension of Global Array
- ***dataSizeGlobal***[ ] ~ Size Along Each Axis
- Coordinate Labels of Global Array

  ⇒ Identify Subregions for Collection

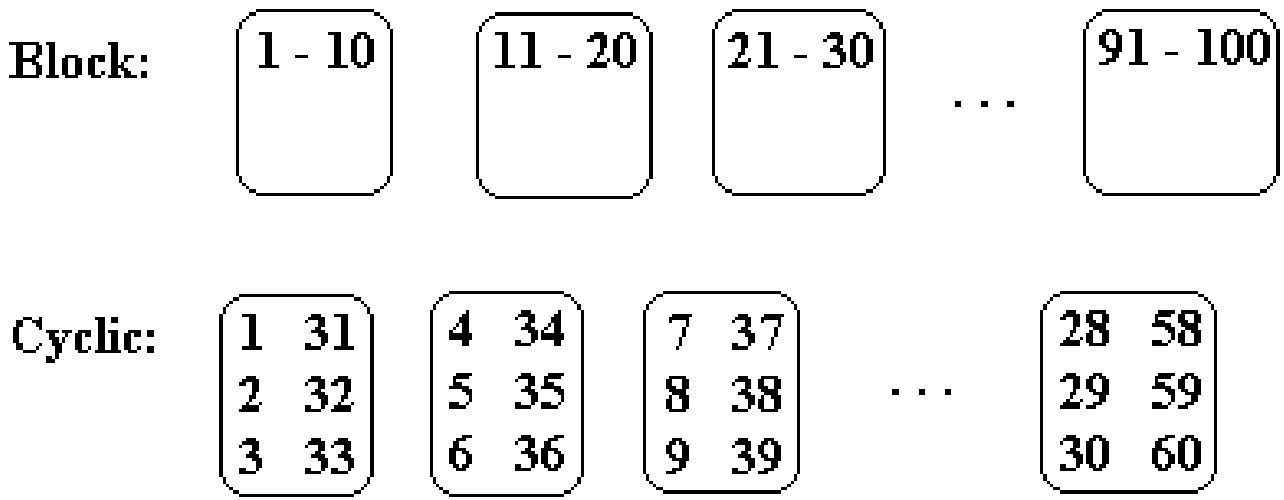
  ⇒ Frame of Reference for Sharing Data

  ⇒ ***globalLowerBounds***[ ] & ***globalUpperBounds***[ ]
- Global Grid Type

  ⇒ ***string globalGridType*** ~ polar, spectral, rect…

Decomposition Types

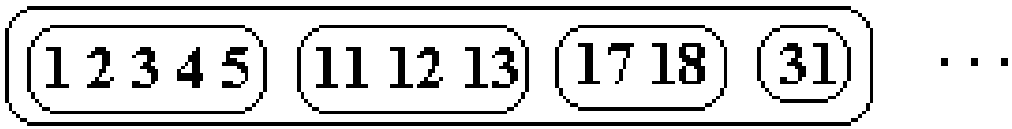
- ***enumerated dataDecompType***[ ]:
  - ⇒ "Standard" HPF-like Definitions
  - ⇒ ***Block, Cyclic, Collapse, Explicit***
- ***dataDecompInfo***[ ] ~ Detail for Block/Cyclic



Explicit Decompositions

- "Catch All" for Non-Standard Decompositions

  ⇒ Multiple Ranges Along Given Axis




```
struct  dataBounds[ dataDimGlobal ]  {
        int  NBounds;
        int  LowerBounds[ NBounds ];
        int  UpperBounds[ NBounds ];
};
```

Topology & Position

- ***procShape***[ ] ~ Organization of Parallel Tasks
  - ⇒ Number of Tasks Participating for Each Axis
- ***procAddr***[ ] ~ Local Task's Position
  - ⇒ Index Along Each Decomposition Axis
  - ⇒ Determines Local Portion of Global Array

Units of Actual Data

- Similar to Time Units…
- ***string dataUnitsScale*** ~ Units Description
  - ⇒ Immense Set of Possible Values
  - ⇒ Defined By Each Application Domain
- ***enumerated unitsPrefix*** ~ Same as Time
  - ⇒ “pico,” “nano,” … “tera,” “peta”

Dynamic Spatial Mappings

- Changing Decomposition or Grid
  - ⇒ E.g. Adaptive Mesh Refinement
- Optional "***struct space_map***" to ***dataReady***( )
  - ⇒ Just Like Dynamic Time Scale…
  - ⇒ Each Data Snapshot ~ Distinct Spatial Layout

Front-End Port...

- What the Outside World Sees…
- Includes:
  - ⇒ Time Map
    - → ***timeValue***, { ***timeUnitsScale***, ***unitsPrefix*** }
  - ⇒ Global Array Size
    - → ***globalLowerBounds***[ ], ***globalUpperBounds***[ ]
    - → ***globalGridType***
  - ⇒ Data Units
    - → ***dataUnitsScale***, ***unitsPrefix***
- Optional Subregion Bounds in ***getPort***()…

Particle-Based Data

- Non-Contiguous Data Distribution
  - ⇒ Specialized Collective Port Needed
  - ⇒ Replace ***decomposition*** with ***particle***…
  - ⇒ A ***particle*** Still Has:
    - → ***dataDimGlobal***
    - → ***globalLowerBounds***[ ] & ***globalUpperBounds***[ ]
    - → ***globalGridType***
  - ⇒ Describe Coordinate Space for Particles
  - ⇒ Subregion in Space Selects Particle Subsets

Particle Data Fields

- ***NFields*** ~ Number of Data Fields in Particle

  ⇒ Simple Data Fields/Arrays, No Decompositions

```
struct  partFields[ NFields ] {
        string  Name;
        enumerated  dataType = { . . . };
        int  dataDim;
        int  dataSize[ dataDim ];
};
```

Extracting Particle Data

- No Well-Defined /Accepted Particle Format

  ⇒ User-Defined Accessor Methods

- ***particleGet*( )** ~ Return Particle "Handle" List

  ⇒ User-Defined, (void *) pointers.

```
void  particleGet (
        int  regionLowerBounds[ dataDimGlobal ].
        int  regionUpperBounds[ dataDimGlobal ],
        void  *parts[ nparts ],
        int  coords[ nparts ][ dataDimGlobal ],
        int  *nparts );
```

Interpolation in Time & Space

- If Port's Data Not In Form Desired
  - ⇒ Either Temporally or Spatially (or Both!)…
- Visualization? No, Leave Data As Is…
- Model Coupling? A Necessity!
- Too Many Interpolation Techniques… :-Q
  - ⇒ Provide Hooks for User-Defined Methods
- Part of "Uses" Port Interface
  - ⇒ "Provider" Just Doesn't Wanna Know… :-)

Temporal Interpolation

- Goal: Process Sequence of Data Snapshots for Different Time Steps to Produce a New Snapshot for Some Other Target Time.

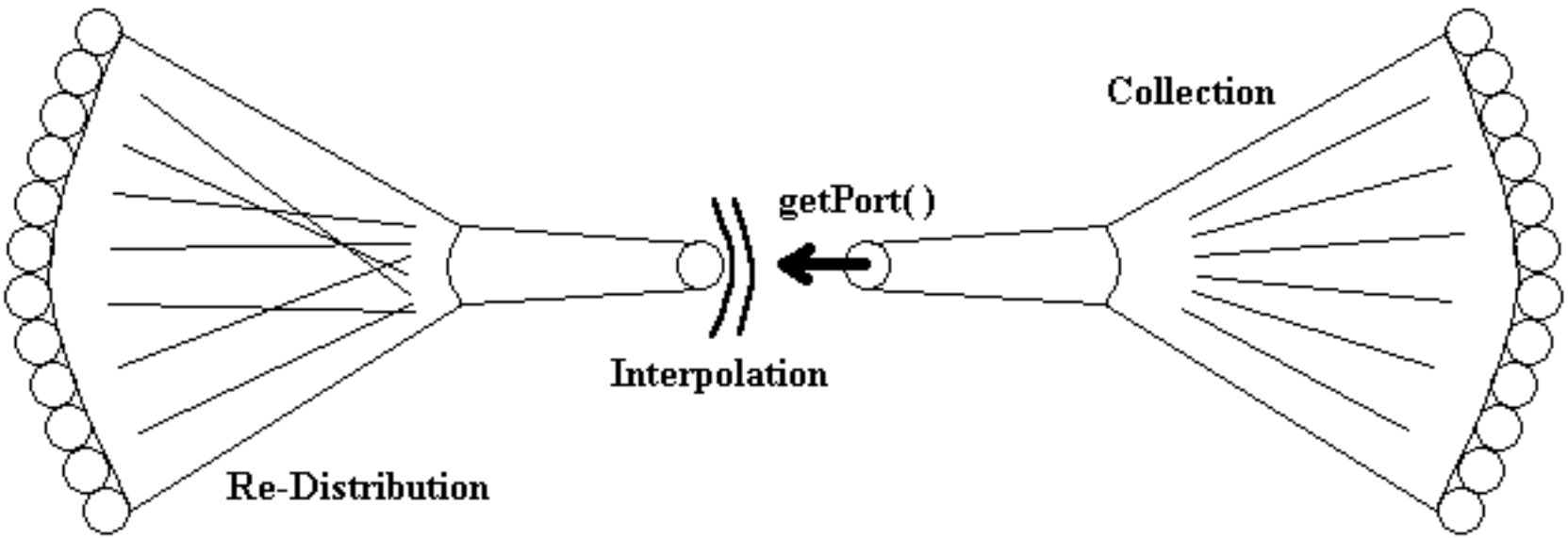
```
void dataInterpolateTime (
        struct space_map smap;
        struct time_map intimes[ nvals ];
        void *indata[ nvals ];
        int nvals;
        struct time_map outtime;
        void *outdata );
```

Spatial Interpolation

- Goal: Translate a Data Array into a Different Spatial Map.

```
void  dataInterpolateSpace (
        struct  space_map  insmap;
        void  *indata;
        struct  space_map  outsmap;
        void  *outdata );
```

Interpolation vs. Re-Distribution



Conclusion

- Some Portions of Collective Map Generalize
  - ⇒ Apply Equally Well to Serial Data
- Basic Port ~ Special Case of Collective
  - ⇒ Without Decomposition / Particle Specification
- To Integrate Into Basic Port Interface:
  - ⇒ ***time_map***
  - ⇒ ***space_map***: ***local***, ***global*** * and ***dataUnits***
  - ⇒ ***dataReady***( )

Oh Yeah, THE DEMO…

- "Viz" Side…
  - ⇒ Porting CUMULVS to Nexus / Globus
  - ⇒ Representative CUMULVS "Kernel" Works
- To Do Next:
  - ⇒ Finesse Nexus Model Into CUMULVS Comm
  - ⇒ Clean Up CUMULVS Message Handlers
- Next BIG Task:
  - ⇒Wrap CUMULVS for CCA ~ Interface to Ports