MxN or MxUse?

(Or just Cheesy?)

Jeeembo Kohl

Oak Ridge National Laboratory

CCA Meeting - SLC, Utah

May 22, 2000

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.

Purpose of MxN

- Share Data Among Parallel Applications
  - ⇒ Ocean & Atmosphere…
  - ⇒ Solver & Optimizer…
  - ⇒ Visualization (Mx1)
- MUST BE EASY TO USE
  - ⇒ Mapping & Interpolation of Parallel Data is **Hard**.
    - * Provide a Clear & Concise API for CCA Components.
  - ⇒ "Simple as Possible, But Not Simpler" ~ Einstein
  - ⇒ Several Approaches – **ALL** Could be Valid.

2 Main Approaches

- **Implicit**

  ⇒ Handled as Underlying CCA Framework Service

- **Explicit**

  ⇒ User Directly Invokes Collective Component Methods

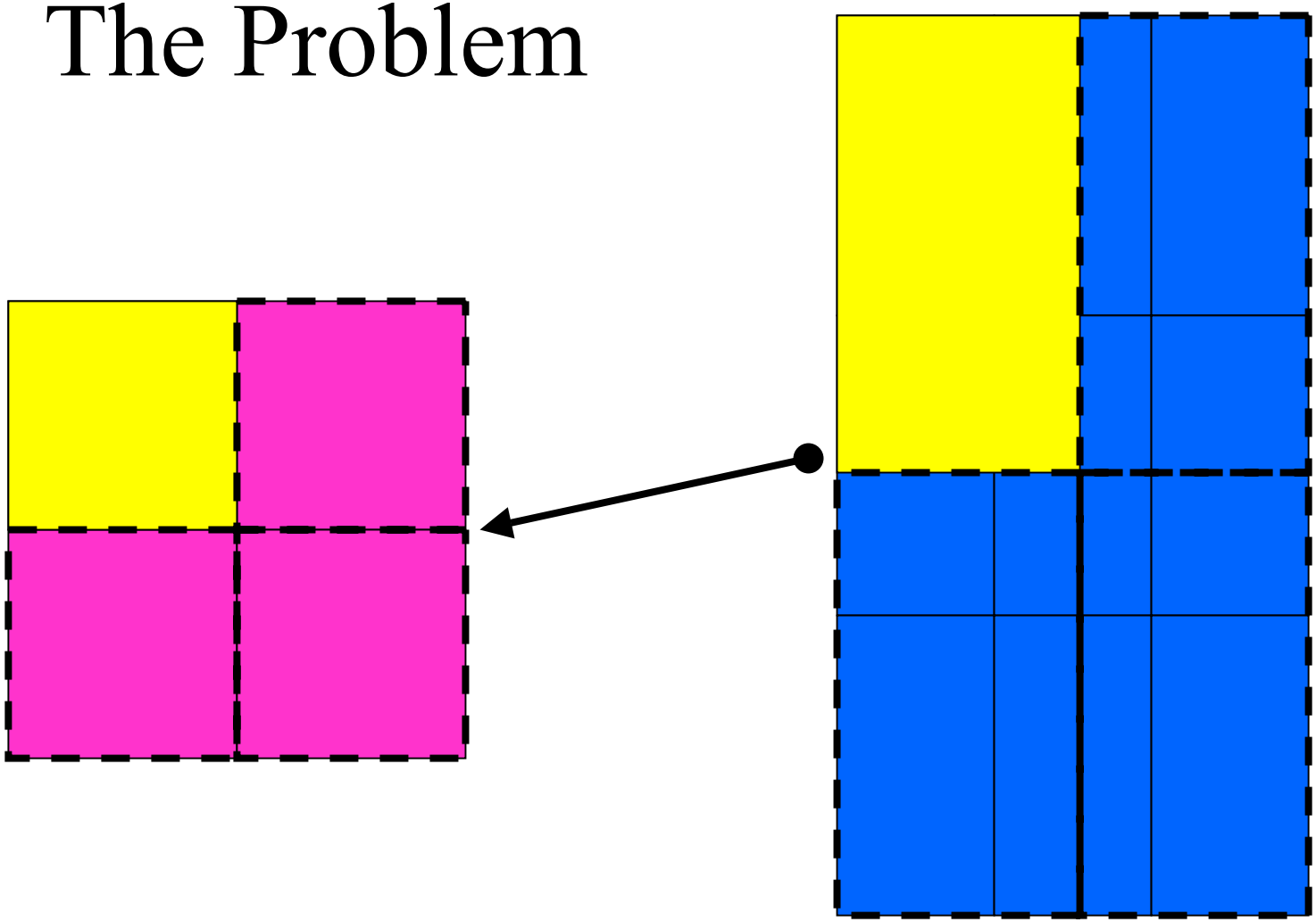
- Both Require Same Instrumentation Information

  ⇒ Same Internal Implementation, Too…

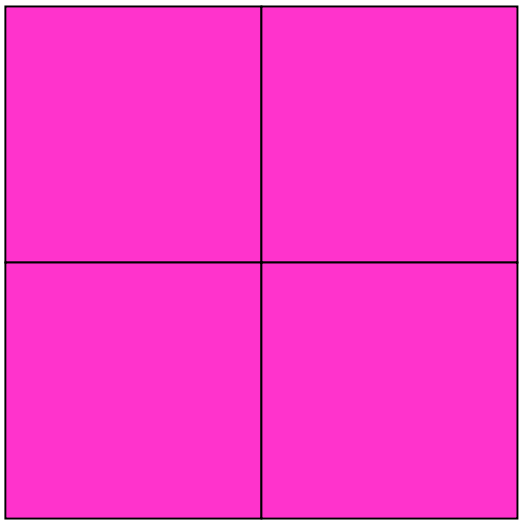
  ⇒ Perhaps CCA Can Support **BOTH** Approaches…

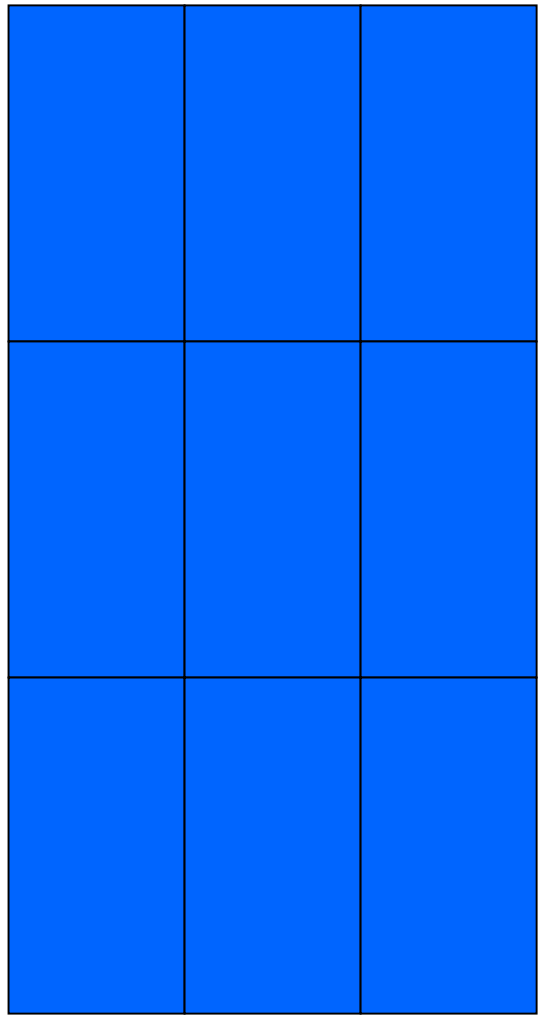
  * **IF** Collective Services Can Plug into Framework like Components!

The Problem

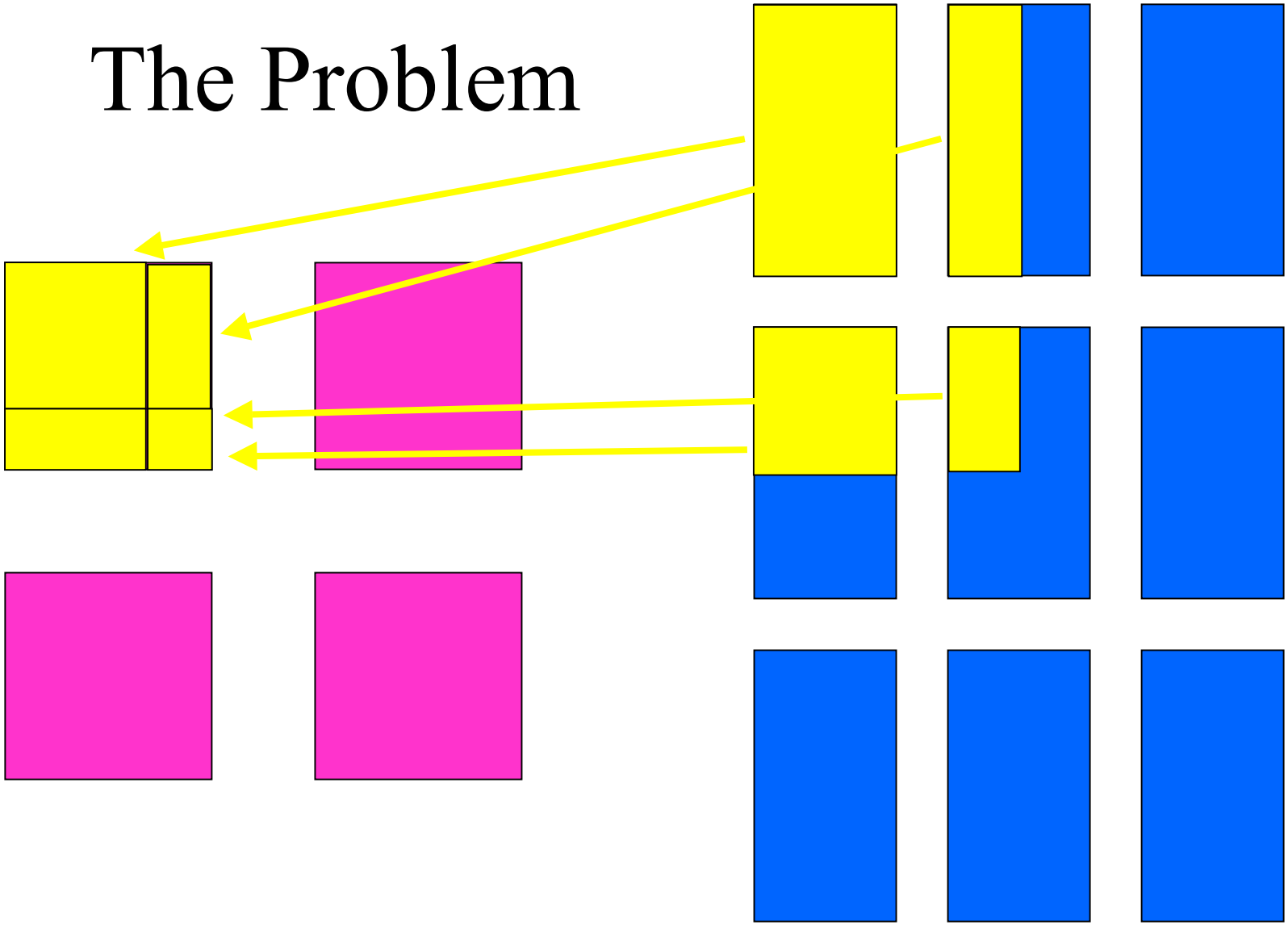


The Problem





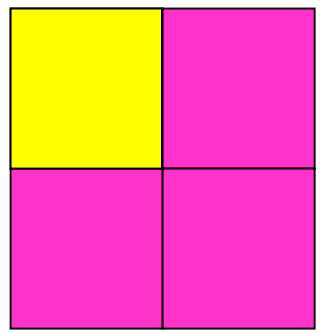
The Problem



Data Decomposition Specification

(Common to EITHER Approach)

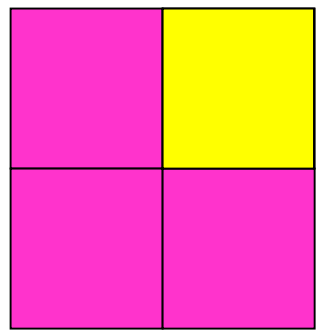
- FOO.Decomp.setDecompBounds(2, 30, 30);
- FOO.DecompInfo.setProperty(“Block Size”, 15);
- FOO.Decomp.setDecompAxis(0, “Block”, FOO.DecompInfo);
- FOO.Decomp.setDecompAxis(1, “Block”, FOO.DecompInfo);
- FOO.ProcArray.setProcShape(2, 2, 2);
- FOO.ProcArray.setProcAddress(2, 0, 0);
- FOO.DataField.setField(“Temperature”, datap);
- FOO.DataField.setDecomp(FOO.Decomp);
- FOO.DataField.setProcArray(FOO.ProcArray);



Data Decomposition Specification

(Common to EITHER Approach)

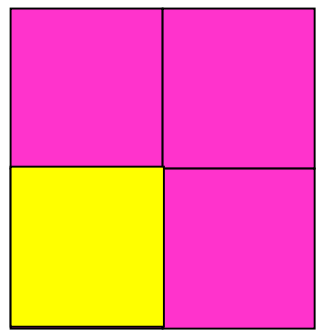
- FOO.Decomp.setDecompBounds(2, 30, 30);
- FOO.DecompInfo.setProperty(“Block Size”, 15);
- FOO.Decomp.setDecompAxis(0, “Block”, FOO.DecompInfo);
- FOO.Decomp.setDecompAxis(1, “Block”, FOO.DecompInfo);
- FOO.ProcArray.setProcShape(2, 2, 2);
- FOO.ProcArray.setProcAddress(2, 0, 1);
- FOO.DataField.setField(“Temperature”, datap);
- FOO.DataField.setDecomp(FOO.Decomp);
- FOO.DataField.setProcArray(FOO.ProcArray);



Data Decomposition Specification

(Common to EITHER Approach)

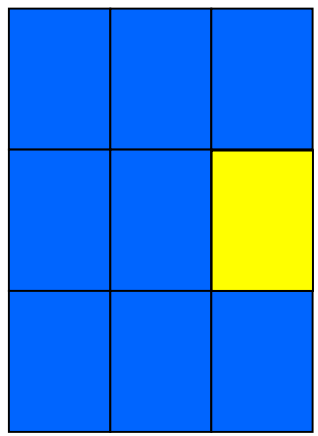
- FOO.Decomp.setDecompBounds(2, 30, 30);
- FOO.DecompInfo.setProperty(“Block Size”, 15);
- FOO.Decomp.setDecompAxis(0, “Block”, FOO.DecompInfo);
- FOO.Decomp.setDecompAxis(1, “Block”, FOO.DecompInfo);
- FOO.ProcArray.setProcShape(2, 2, 2);
- FOO.ProcArray.setProcAddress(2, 1, 0);
- FOO.DataField.setField(“Temperature”, datap);
- FOO.DataField.setDecomp(FOO.Decomp);
- FOO.DataField.setProcArray(FOO.ProcArray);

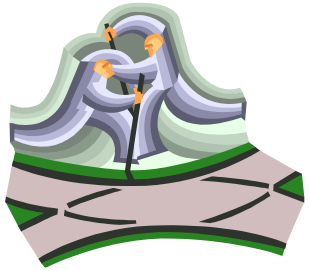


Data Decomposition Specification

(Common to EITHER Approach)

- FOO.Decomp.setDecompBounds(2, 30, 30);
- FOO.DecompInfo.setProperty(“Block Size”, **10**);
- FOO.Decomp.setDecompAxis(0, “Block”, FOO.DecompInfo);
- FOO.Decomp.setDecompAxis(1, “Block”, FOO.DecompInfo);
- FOO.ProcArray.setProcShape(2, **3, 3**);
- FOO.ProcArray.setProcAddress(2, **1, 2**);
- FOO.DataField.setField(“Temperature”, datap);
- FOO.DataField.setDecomp(FOO.Decomp);
- FOO.DataField.setProcArray(FOO.ProcArray);





And Now The Grudge Match…

- So What Next?

  ⇒ Given Each Local Data Field Decomposition, How Do Users Invoke the "Collective"…?

- Simplest Case

  ⇒ Framework Just Does It.

- Flexible, Direct Case

  ⇒ User Invokes "Collective Component".

- Each Case Has Its Benefits…

Collective Framework Service

(FOO = CCAServices.Collective)

Ocean.CorrelateTemperature(Atmosphere.Temperature);

- CCAServices.Collective Service ~ Automatic Handling
  - ⇒ Checks Data Field Property to Determine Data Parallelism (Atmosphere.Temperature)
  - ⇒ Resolves Data Extraction Per "Caller Decomposition"?
    - * Probably Need to Set Collective Properties for METHODS, too… (Ocean.CorrelateTemperature( ))

Collective Framework Service

(FOO = CCAServices.Collective)

- Pros:
  - ⇒ VERY Easy API for Users, Scientists, Anyone…
- Cons:
  - ⇒ Burdens Each Framework Developer with MxN Problem.
    - * Circumvented if a Pluggable Service is Created for Collective!
  - ⇒ Minor Overhead on Each Invocation to Check Field Property.
  - ⇒ Lacks Flexibility ~ Eliminates Alternative Semantics
    - * What if the 2 Data Fields Don't Quite Match? Interpolation?!
    - * What if You Want a Copy of the **Whole** Data Field Somewhere?

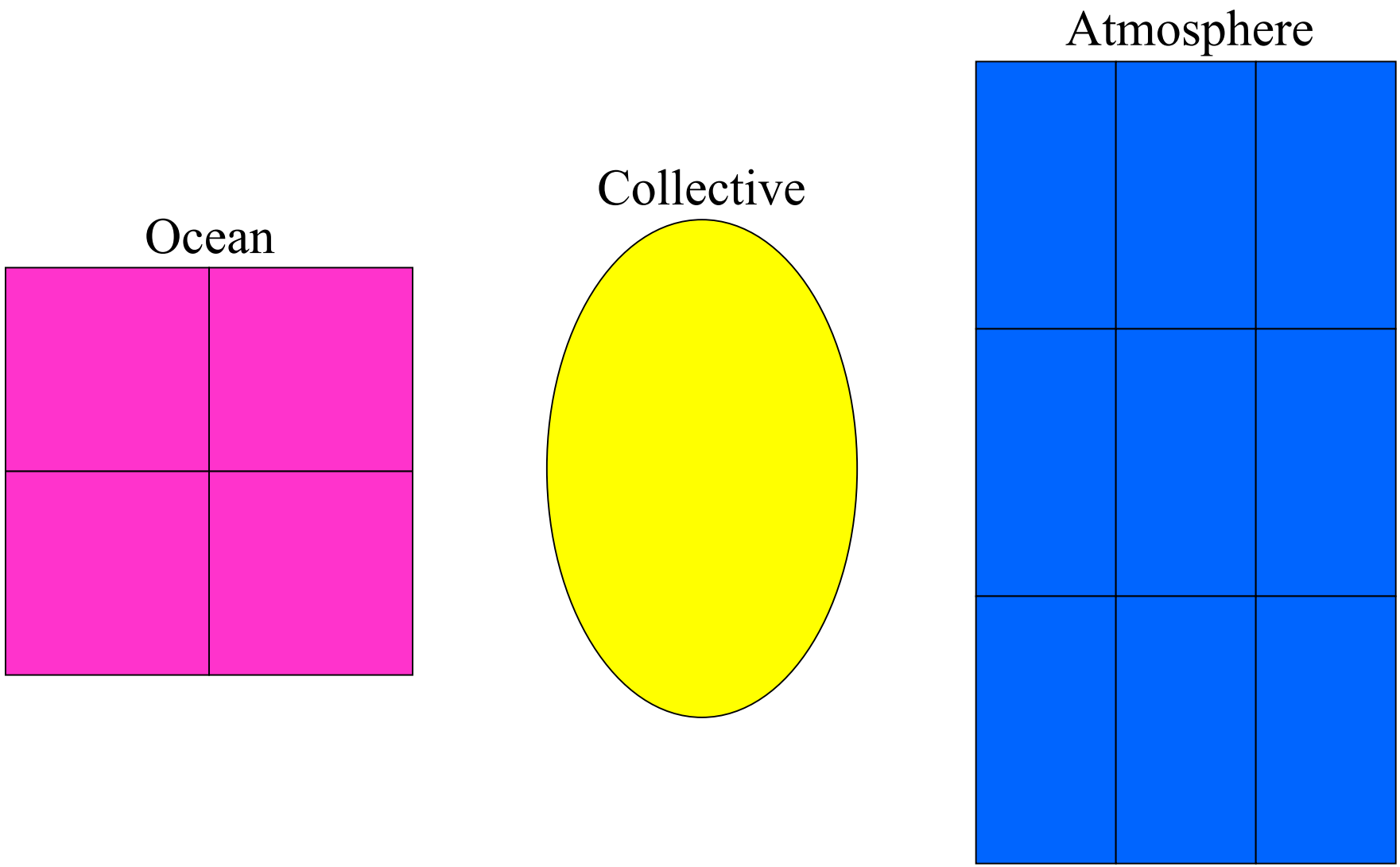
Collective Component

(FOO = CCACollective)

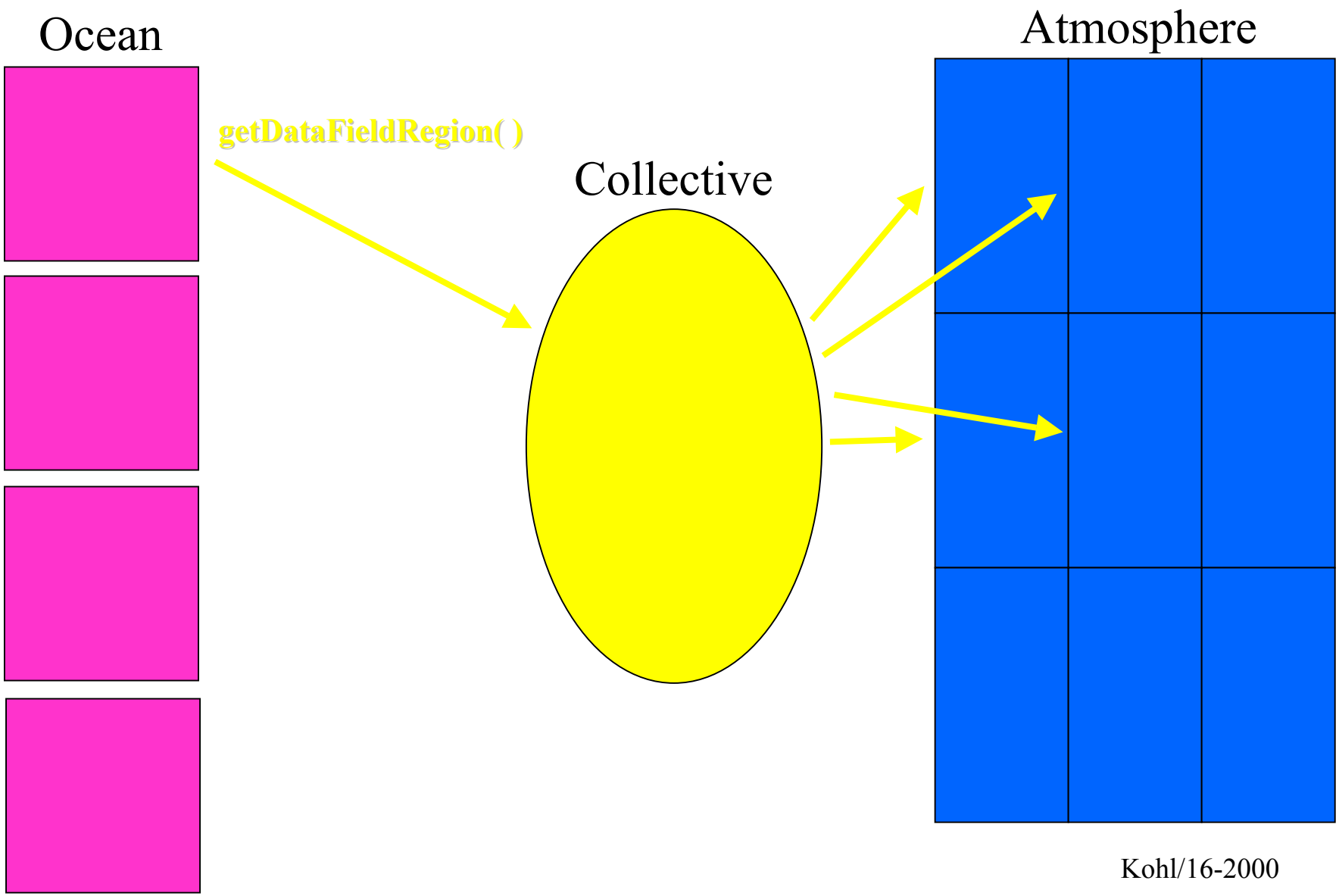
Ocean.CorrelateTemperature(
CCACollective.getDataFieldRegion(Atmosphere.Temperature,
Ocean.Temperature.Decomp, Ocean.ProcArray));

- CCACollective Component Resolves Decompositions
  - ⇒ Explicit Method **getDataFieldRegion( )** Used to Translate
  - ⇒ Caller's Data Decomposition used as Template for Extraction
    - * Specific Address in Processor Array Used to Index Decomposition
  - ⇒ Source Data's Decomposition is Known
    - * Source isn't the Tricky Part, it's the Destination!

Collective Component Example

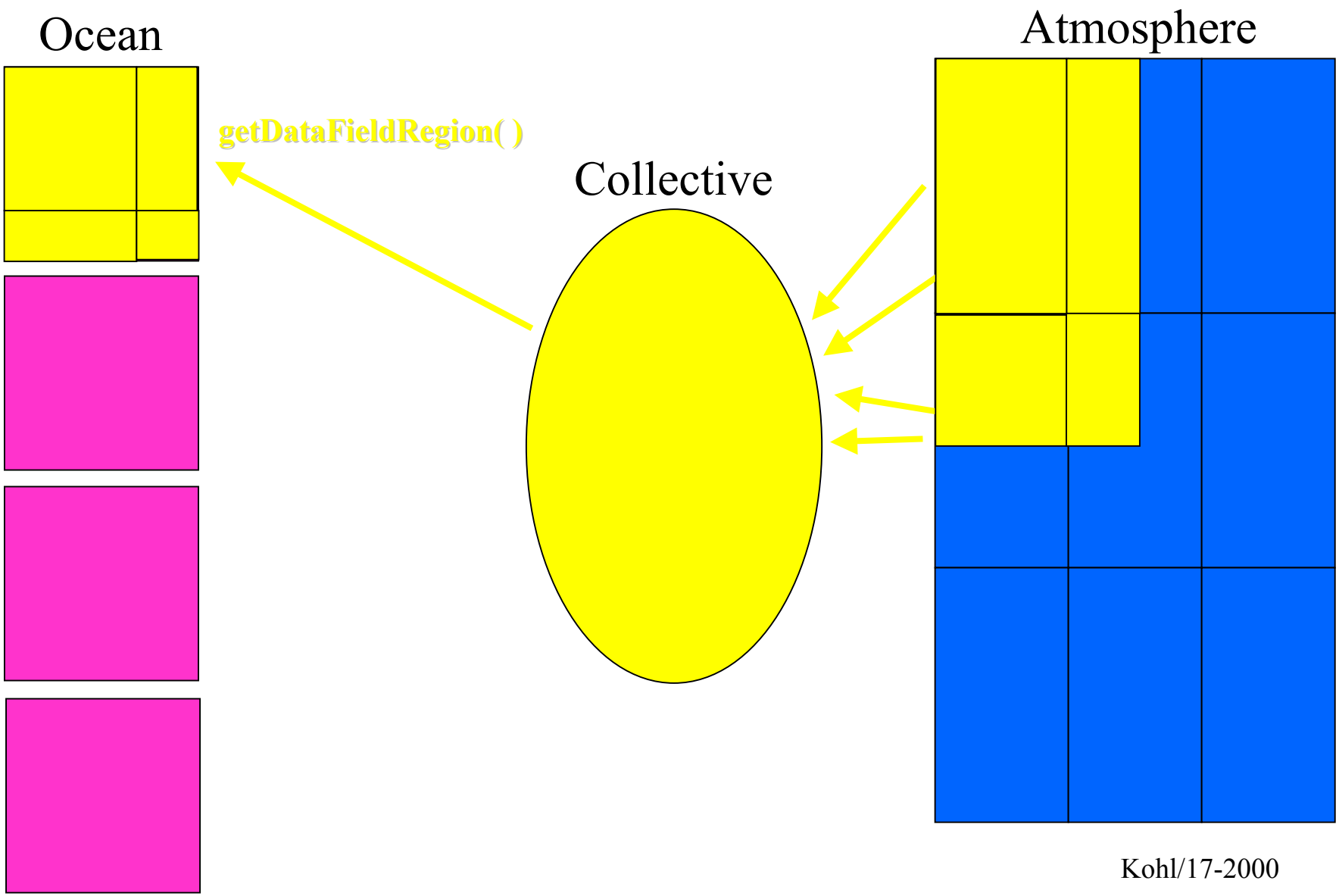


Collective Component Example

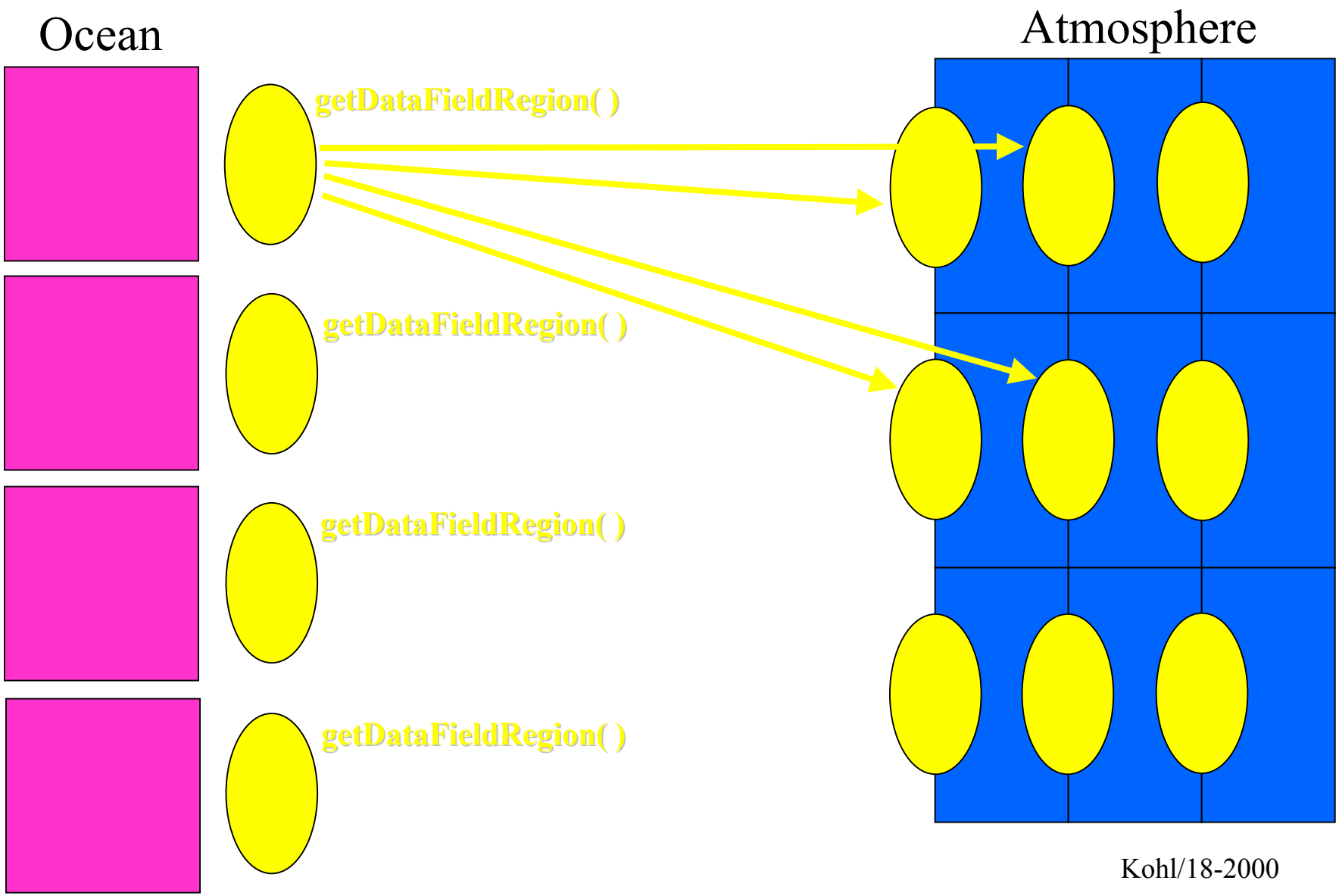


Kohl/16-2000

Collective Component Example



Collective Component Example



Collective Component

(FOO = CCACollective)

- Pros:
  - ⇒ Single, Portable Component ~ One-Stop MxN Shopping!
  - ⇒ Precise Functionality Defined by Method Semantics
    - * Small Set of Methods Covers Wide Range of Functions
    - * Scatter, Gather, Subregions, Time & Space Interpolation
  - ⇒ (No Unnecessary Overheads for Scalar Data Case)
- Cons:
  - ⇒ User Has to Know What They Want (& How to Get It)
    - * Requires Basic Knowledge of Data Decompositions & Functions…
    - (This should be true for framework service case, too… :-)

Conclusion?

- Both Approaches Valid
  - ⇒ Both Have Merit, Pros/Cons…
  - ⇒ Why Not Support Both?
- Real Key:
  - ⇒ Implement Full MxN Guts Once
  - ⇒ Wrap With Multiple APIs ~ Use For Both
  - ⇒ Can't We Just All Get Along?! ☺