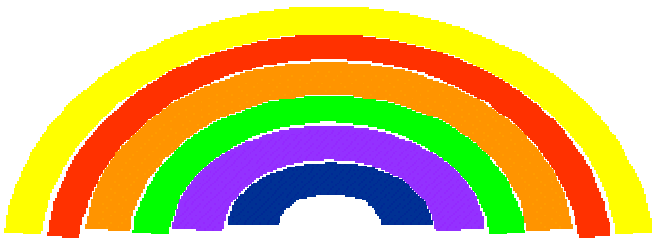
Drawn and Quartered: Turning MxN Inside-Out!

(Or "MxN Chum: The Greasy Details Float to the Surface" ☺)



October 2, 2001

James "Jeeembo" Kohl

Oak Ridge National Laboratory

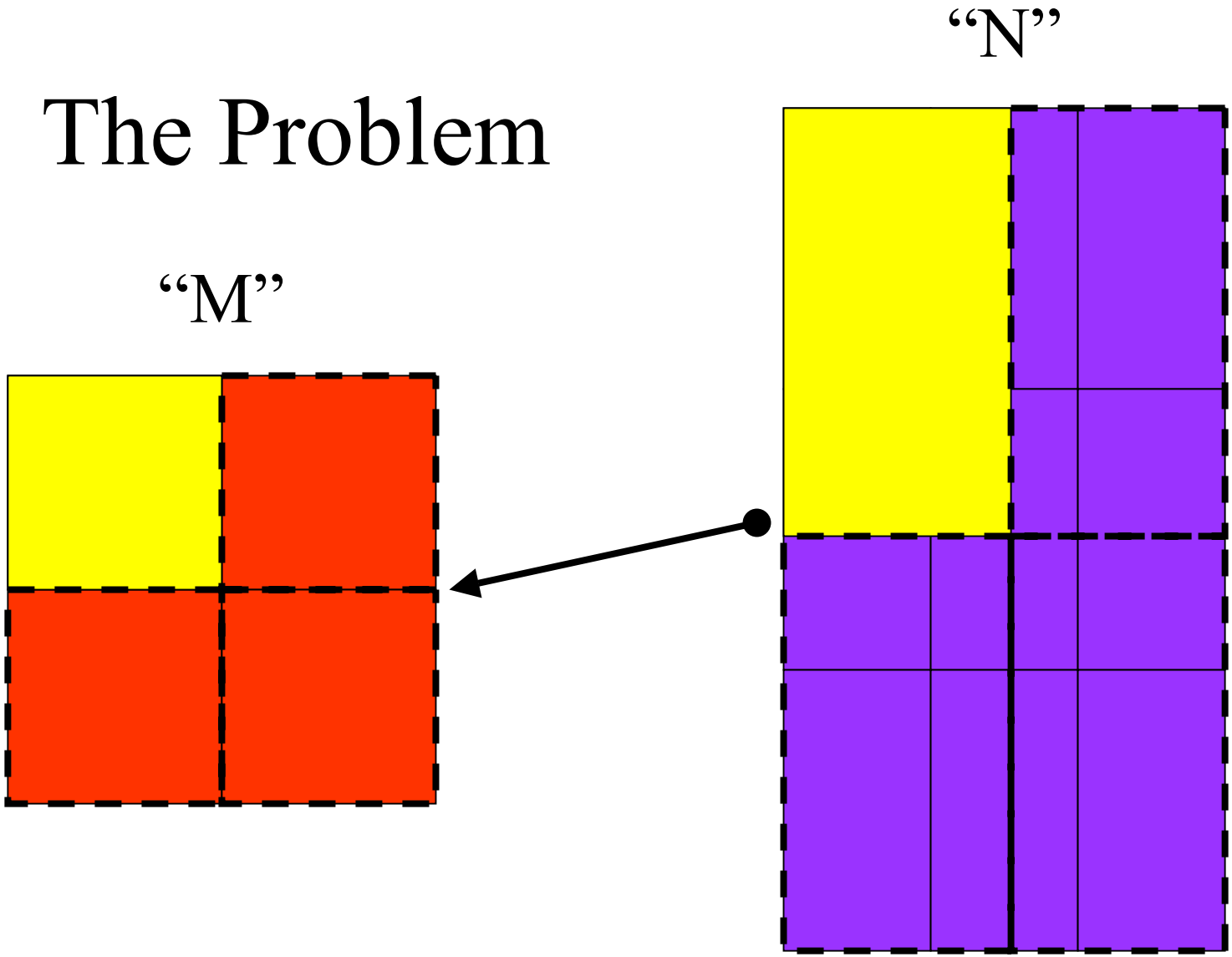
(and other assorted MxN-ers!)

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.

“MxN” Parallel Data Redistribution

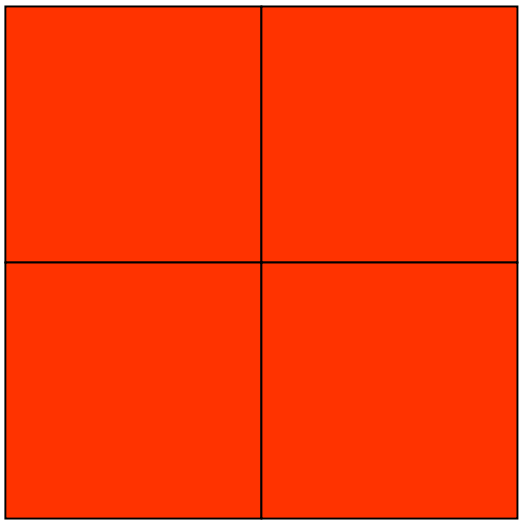
- Share Data Among Coupled Parallel Models
  - ⇒ Ocean & Atmosphere, Solver & Optimizer…
  - ⇒ Visualization (Mx1)
- Encapsulate Existing Technologies:
  - ⇒ CUMULVS (ORNL) & PAWS (LANL)
  - ⇒ Data Decompositions, Communication Schedules, Connections & Synchronization Protocols…
- General Coupling is Beyond Scope (Later…)
  - ⇒ Mapping & Interpolation of Parallel Data is **Hard**.
    - Different Grids, Spatial & Temporal, Units

The Problem

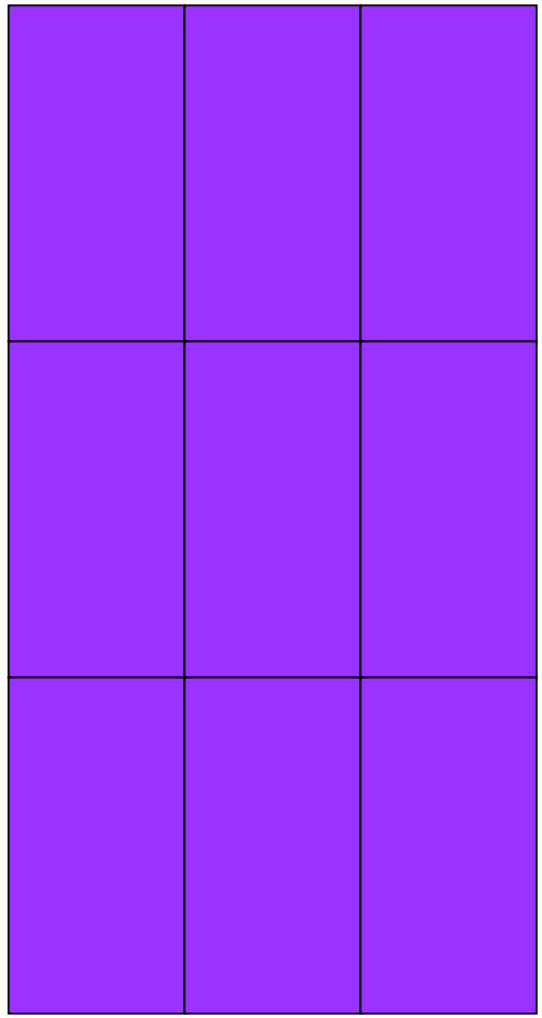


The Problem

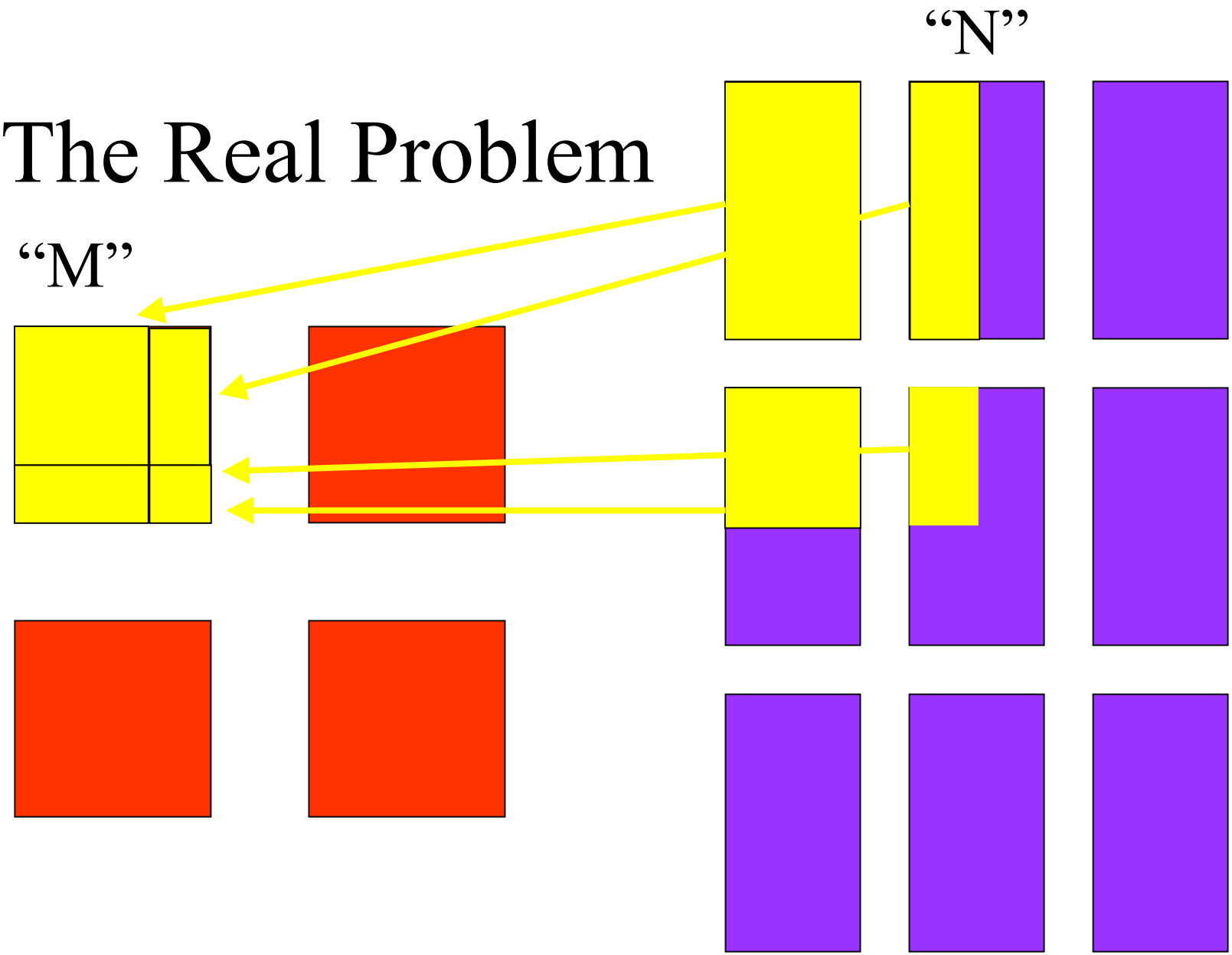
“M”



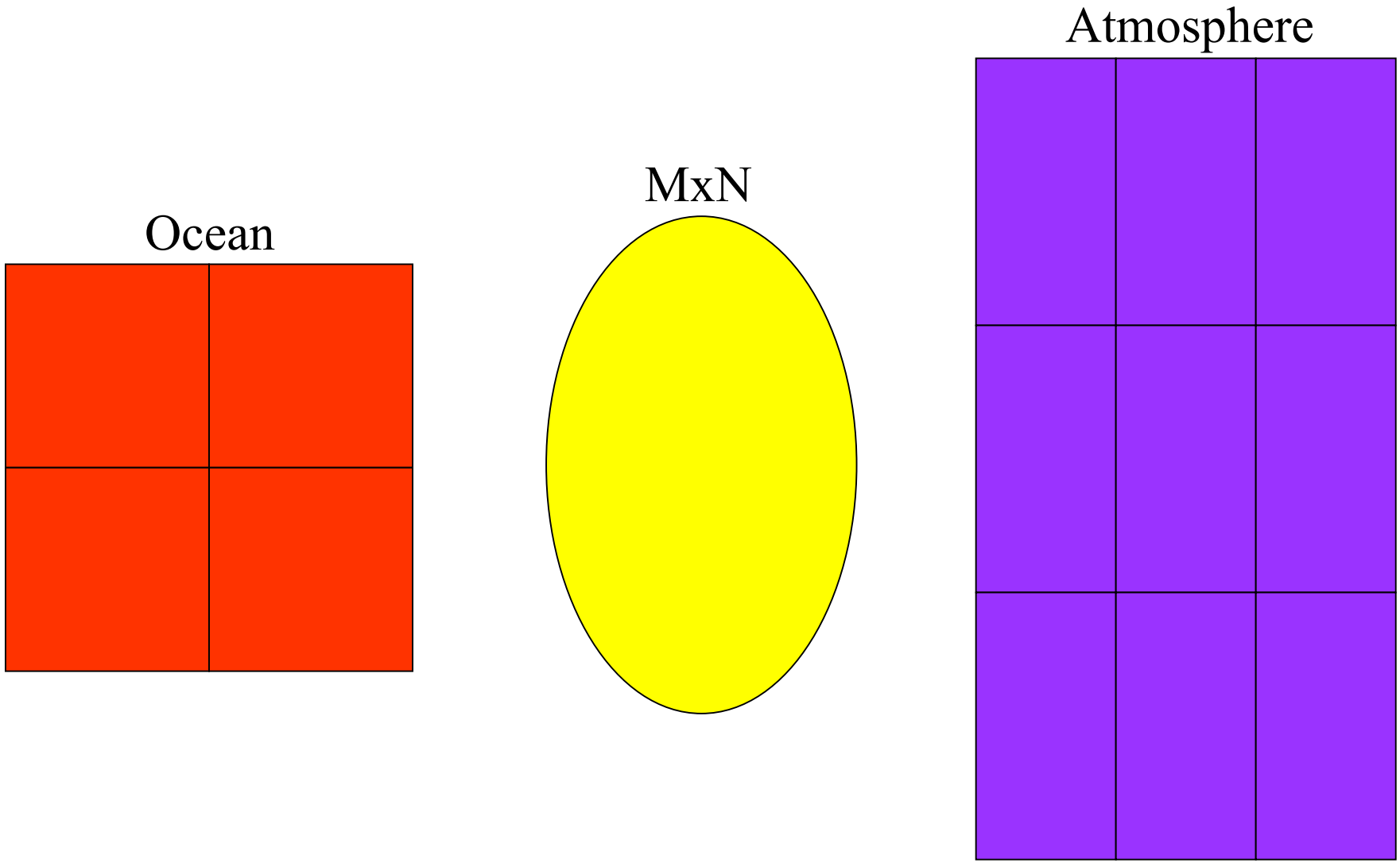
“N”



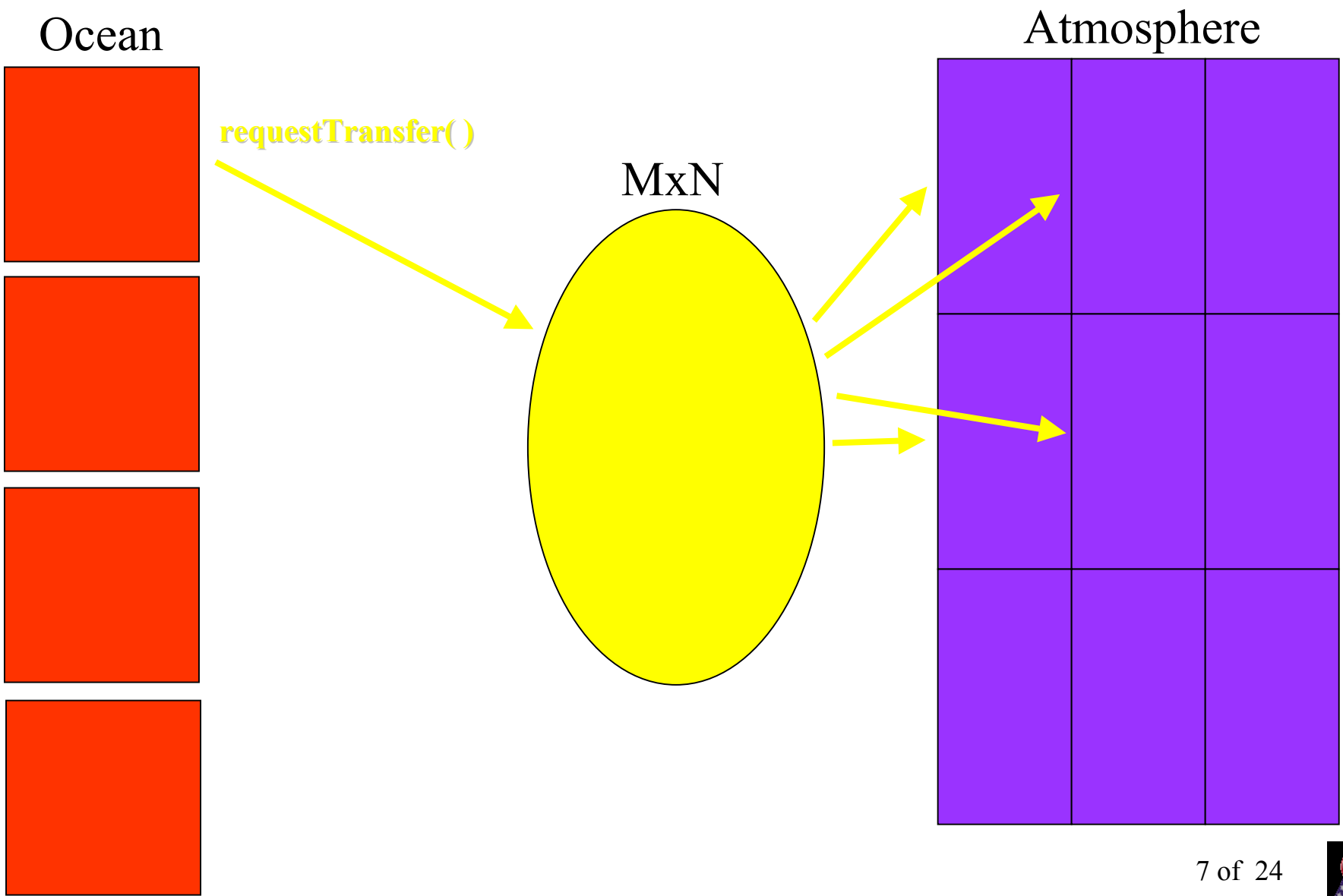
The Real Problem



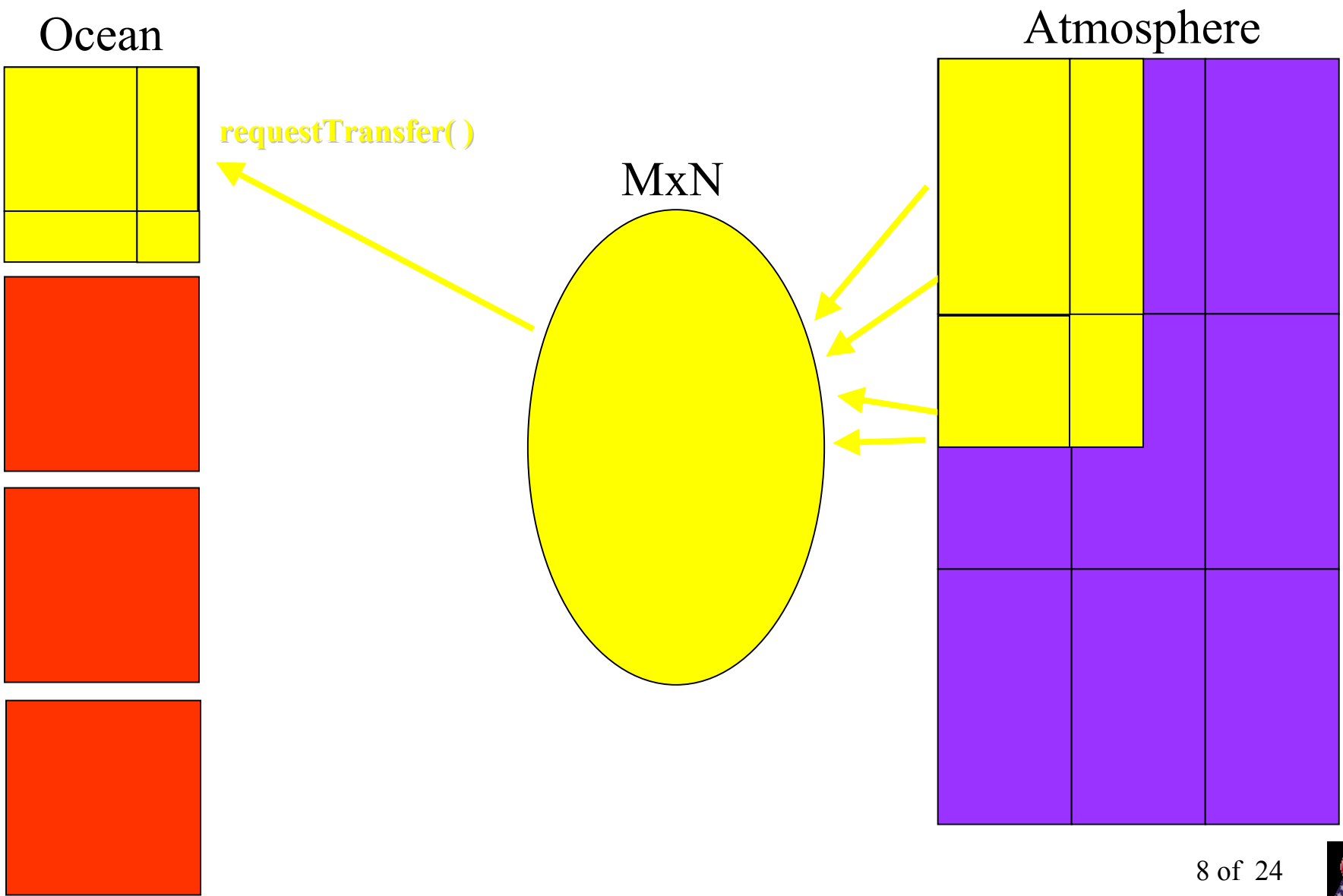
MxN Component Example



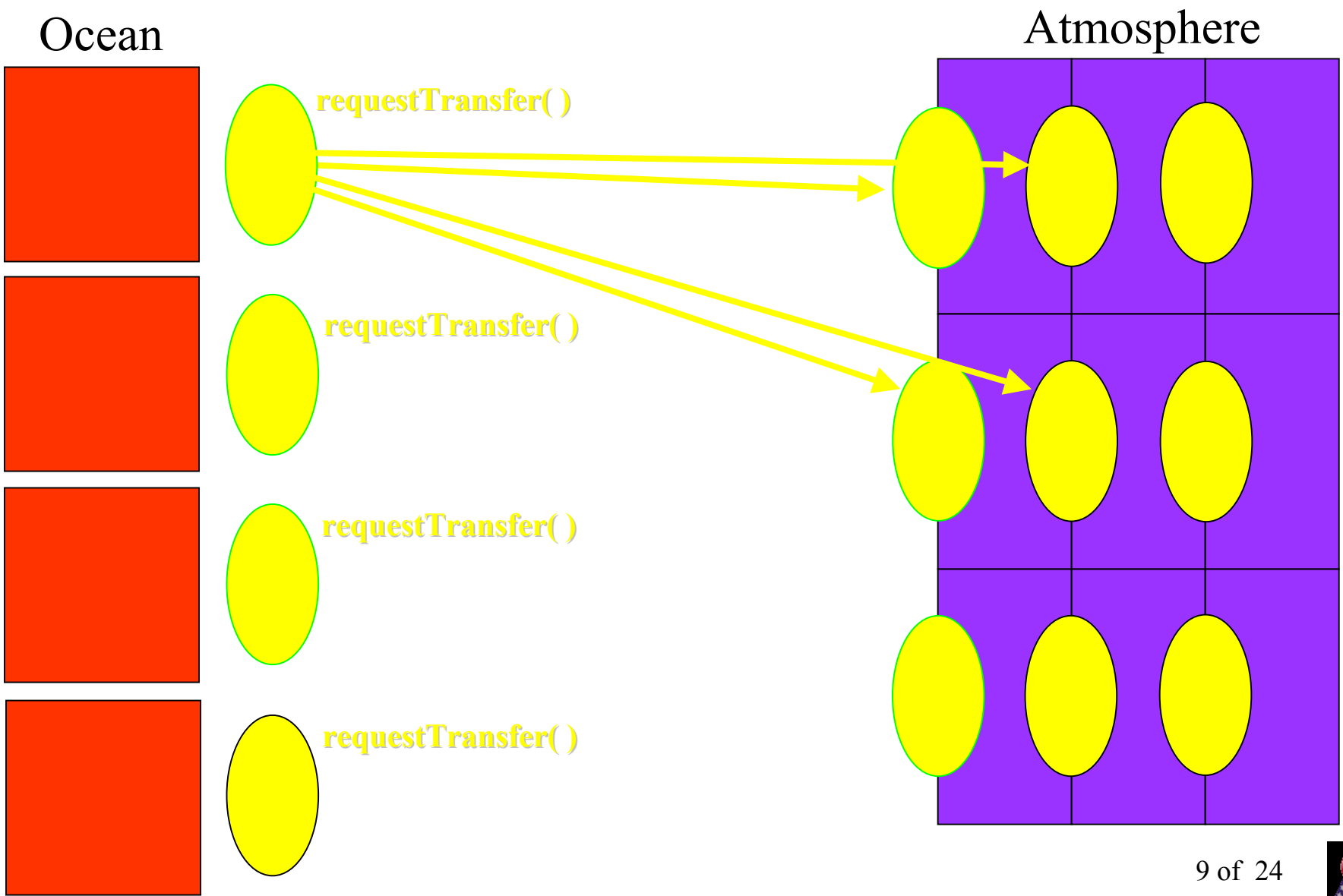
MxN Component Example



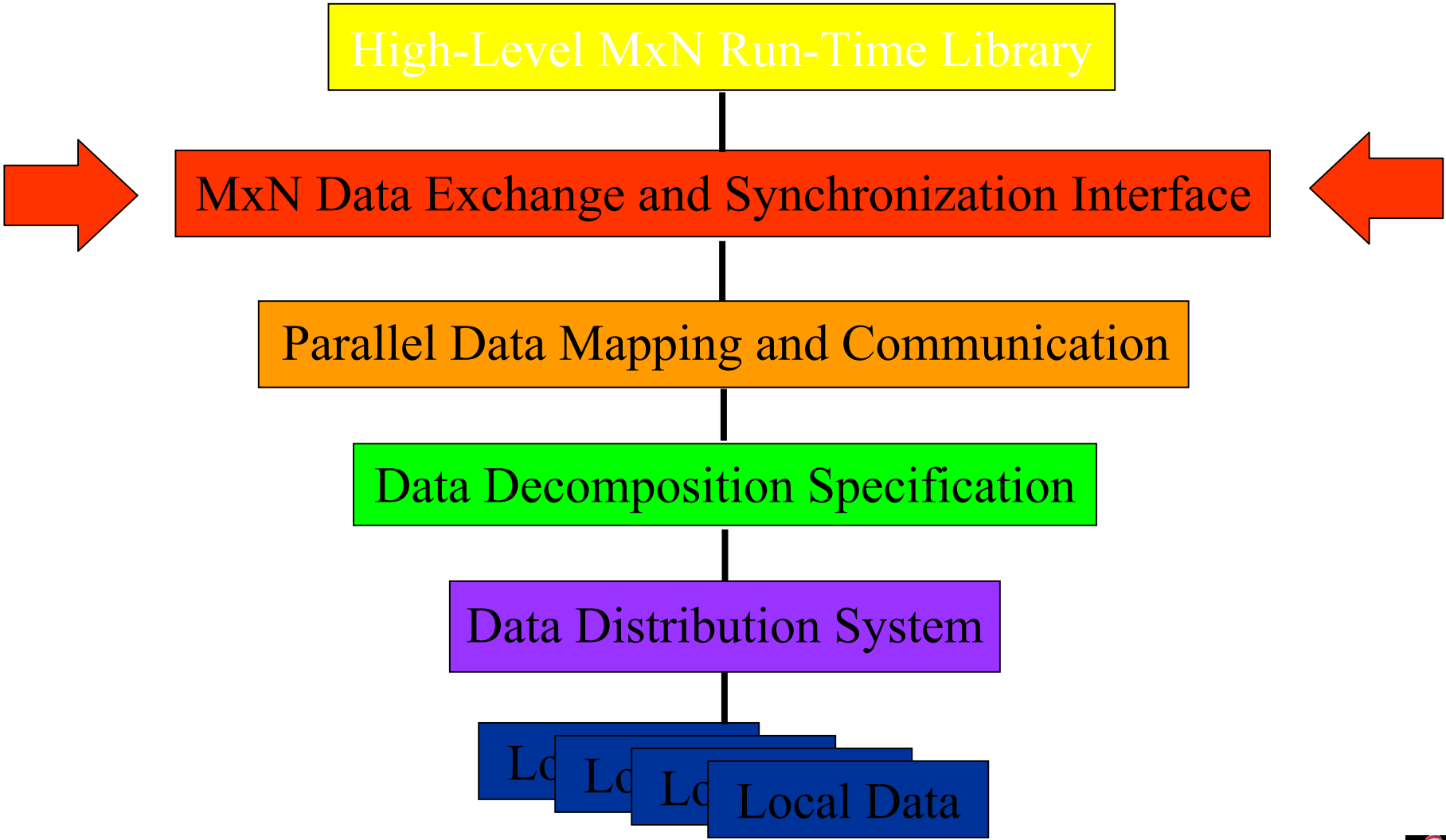
MxN Component Example



Scalable MxN Component Example



Unified MxN Internal Structure



Original MxN Port Interface

```
class MxN : Port {
    virtual void registerData( Data A );
    virtual void unregisterData( Data A );
    // Data Properties: synch
    virtual int getData( Data A, Data B );
    virtual int getDataNonBlocking( A, B );
    virtual void waitData( A, B );
    virtual void releaseData( A, B );
    // Port Properties: freq, init_synch, run_synch…
}
```

“NEW & IMPROVED” MxN Port Interface Exposed!

```
class MxN : public virtual gov::cca:Port {
        enum AccessMode { ReadOnly, WriteOnly, ReadWrite };
        virtual int registerData( Data data, Boolean synch, AccessMode access, mxnDataHandle *dhandle ) = 0;
        virtual int unregisterData( mxnDataHandle dhandle ) = 0;
        virtual int reregisterData( mxnDataHandle dhandle, Data data, Boolean synch, AccessMode access ) = 0;
        virtual int createCommSched( Decomp dst_decomp, Decomp src_decomp, mxnCommSchedHandle *cshandle ) = 0;
        virtual int freeCommSched( mxnCommSchedHandle cshandle ) = 0;
        enum mxnSynch { Default, Synch, NoSynch };
        virtual int makeConnection( mxnDataHandle dst, mxnDataHandle src, mxnCommSchedHandle cshandle, mxnSynch dst_synch,
                mxnSynch src_synch, Time dst_freq, Time src_freq, mxnConnHandle *chandle ) = 0;
        virtual int releaseConn( mxnConnHandle chandle ) = 0;
        virtual int modifyConn( mxnConnHandle chandle, mxnCommSchedHandle cshandle, mxnSynch dst_synch, mxnSynch src_synch,
                Time dst_freq, Time src_freq ) = 0;
        virtual int postData( mxnDataHandle dhandle ) = 0;
        virtual int releaseData( mxnDataHandle dhandle ) = 0;
        virtual int requestTransfer( mxnConnHandle chandle,  mxnTransferHandle *thandle ) = 0;
        virtual int waitTransfer( mxnTransferHandle thandle ) = 0;
        virtual int testTransfer( mxnTransferHandle thandle ) = 0;
        int dataReady( mxnDataHandle dhandle, Boolean update_time );
        int dataReadyBegin( mxnDataHandle dhandle, Boolean update_time );
        int dataReadyEnd( mxnDataHandle dhandle );
}
```

All MxN! All The Time! ☺

Data "Registration"

- Provides "MxN-er" With Access Mode and Info
  - ⇒ Use Generic "Data" Interface Handle
    - Decomposition & Allocation Details (Where Elements Are)
  - ⇒ Synchronization Requirements
    - Periodic Consistency (True) vs. Embarrassingly Parallel (False)
  - ⇒ Set Allowed Access Modes (Read & Write Elements)

```
enum AccessMode { ReadOnly, WriteOnly, ReadWrite };

virtual int registerData( Data data, Boolean synch,
        AccessMode access, mxnDataHandle *dhandle ) = 0;

virtual int unregisterData( mxnDataHandle dhandle ) = 0;
```

Communication Schedules

- Coordinates Communication Mapping Between Two Parallel Data Instances
  - ⇒ Only Uses Data *Distribution* Templates
    - No Actual Data Needed…
  - ⇒ Returns Reusable Comm Schedule Handle
    - AVI For Comm Schedules ~ Operate On / Optimize…

```
virtual int createCommSched( Decomp dst, Decomp src,
        mxnCommSchedHandle *cshandle ) = 0;

virtual int freeCommSched( mxnCommSchedHandle cshandle ) = 0;
```

“Connections”

- Create Communication “Channel” Between Two Parallel Data Instances
  - ⇒ Synchronize Each Parallel Cohort (As Needed)
  - ⇒ Set Up Frequency of Transfers (Else “One-Shot”)
  - ⇒ No Actual Data Transfers Yet!
    - Use Connection Handle to Invoke Each Transfer…

```
enum mxnSynch { Default, Synch, NoSynch };

virtual int makeConnection( mxnDataHandle dst, mxnDataHandle src,
    mxnCommSchedHandle cshandle, mxnSynch dst_synch, mxnSynch src_synch,
    Time dst_freq, Time src_freq, mxnConnHandle *chandle ) = 0;

virtual int releaseConn( mxnConnHandle chandle ) = 0;
```

MxN Transfers

- Actually Transfer Elements from Src → Dst
  - ⇒ Non-Blocking Request Returns "Transfer Handle"
    - Test / Wait For Transfer Completion
  - ⇒ Operation Proceeds Asynchronously On Each Side
    - As Each Parallel Instance Indicates "Data Ready"…

```
virtual int requestTransfer( mxnConnHandle chandle,
        mxnTransferHandle *thandle ) = 0;

virtual int waitTransfer( mxnTransferHandle thandle ) = 0;

virtual int testTransfer( mxnTransferHandle thandle ) = 0;
```

“Data Ready”

- Parallel Instance’s Local Data is Consistent
  - ⇒ Ready for Any Pending Transfers
  - ⇒ Use Blocking Or Begin/End Versions
  - ⇒ Can Automatically Update Data “Time”…

```
virtual int dataReady( mxnDataHandle dhandle,
        Boolean update_time ) = 0;

virtual int dataReadyBegin( mxnDataHandle dhandle,
        Boolean update_time ) = 0;

virtual int dataReadyEnd( mxnDataHandle dhandle ) = 0;
```

MxN Connection / Transfer Example

Component “A”:

```
registerData( A, True, ReadOnly, &Ahandle );
...
dataReady( Ahandle, True );
```

Component “B”:

```
registerData( B, True, WriteOnly, &Bhandle );

createCommSched( A.decomp, B.decomp, &cshandle );
makeConnection( Bhandle, Ahandle, cshandle, Default, Default,
    Bfreq, Afreq, &chandle );

requestTransfer( chandle, &thandle );
...
dataReady( Bhandle, True );
...
waitTransfer( thandle );
```

Issue #1: "Application Discovery"

- Who Decides Which Data Gets Exchanged?
- How Do They Get The MxN Data Handles?!

1. "Magic" ~ Well-Known, Hard-Coded Names?
2. Programmatic Control ~ Port Parameters?
3. Composition Time Cues?!

Either Way:

→ Add "Name" to **registerData**( )

→ Add: mxnDataHandle **getRegisteredData**("Name")

Issue #2: Dynamic MxN Connections

Problem: Dynamic Data Decompositions

⇒ Adaptive Mesh Refinement? (Not Necess…)

⇒ Load Balancing… (Real Life Examples)

- Need Way To Invalidate MxN Connections
- Fundamental Choices:

⇒ Modify Existing Connection

or

⇒ Break Connection & Re-Initiate

Modify Existing Connection…

- CON: Adds Complexity to Interface
  - ⇒ Also Implementation…
- PRO: Maintains Synchronization!

```
virtual int reregisterData( mxnDataHandle dhandle,
        Data data, Boolean synch, AccessMode access ) = 0;

virtual int modifyConn( mxnConnHandle chandle,
        mxnCommSchedHandle cshandle,
        mxnSynch dst_synch, mxnSynch src_synch,
        Time dst_freq, Time src_freq ) = 0;
```

Break Connection & Re-Initiate…

- CON: Broken Conn Loses Synchronization!
  - ⇒ Need a Way to "Hold" One / Both Parallel Cohorts
- PRO: Little Additional Interface Complexity
  - ⇒ Extra Method Could Be Generally Useful:

```
virtual int holdData( mxnDataHandle dhandle ) = 0;

virtual int releaseData( mxnDataHandle dhandle ) = 0;
```

Conn Hold / Re-Initialize Example

```
requestTransfer( chandle, &thandle );
...
waitTransfer( thandle );

<Data Decomposition Changes…>

holdData( Ahandle );
holdData( Bhandle );

releaseConn( chandle );
createCommSched( A.decomp, B.decomp, &cshandle );
makeConnection( Bhandle, Ahandle, cshandle, Default, Default,
    Bfreq, Afreq, &chandle );

releaseData( Ahandle );
releaseData( Bhandle );

requestTransfer( chandle, &thandle );
...
```

Hold
Sync

MxN Status

- BIG PLANS for SC2001 Demo… ☺
  - ⇒ Concurrent Efforts:
    - CUMULVS @ ORNL
    - PAWS @ LANL
  - ⇒ New MxN Spec Is General and "Complete"…

    (So General That Neither CUMULVS or PAWS

    Can Currently Fully Implement It! ☺)
  - ⇒ Need Generalized Data Object to Drive It Home…