Tammy Dahlgren, Tom Epperly,
Scott Kohn, & Gary Kumfert

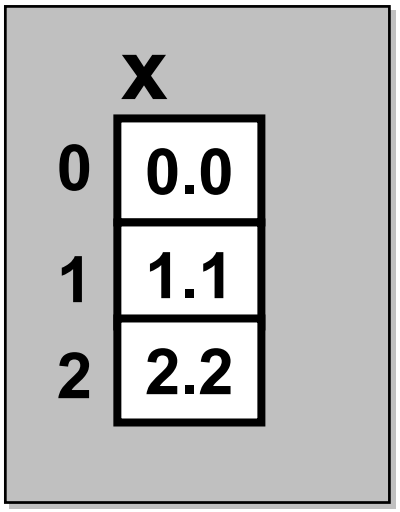
Format

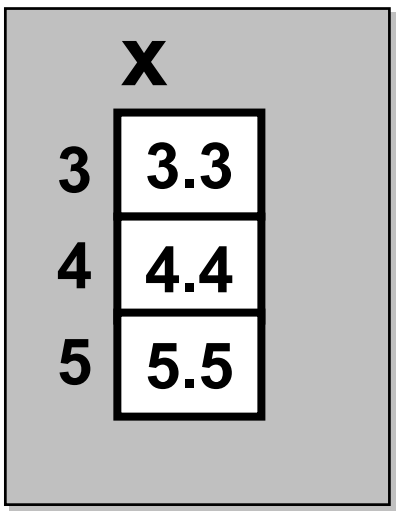
Recap of Gatlinburg Talk...

... only in Reverse...

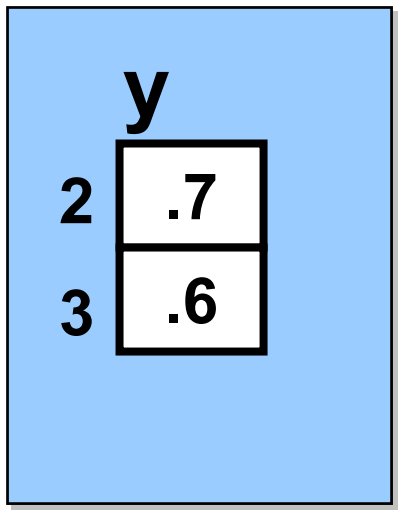
... Conceptually, not literally.

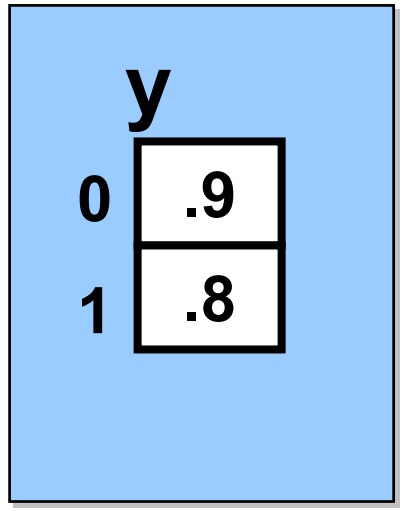
Example #1: 1-D Vectors

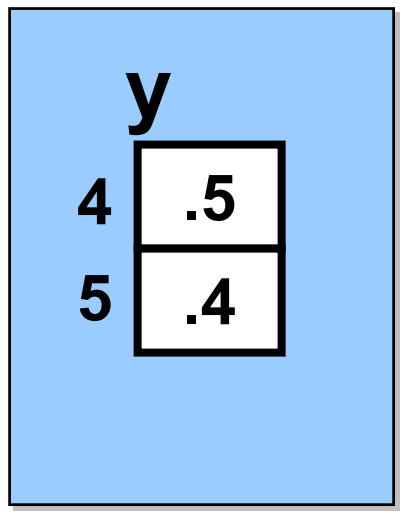




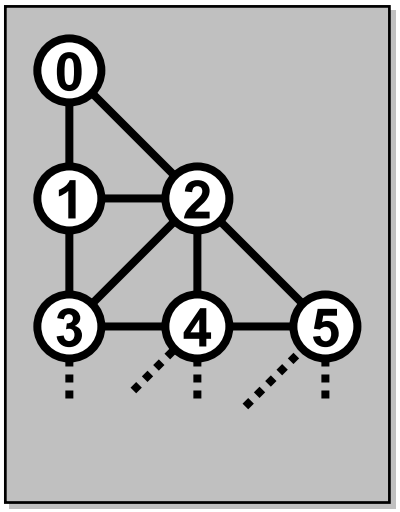

```
double d = x.dot( y );
```

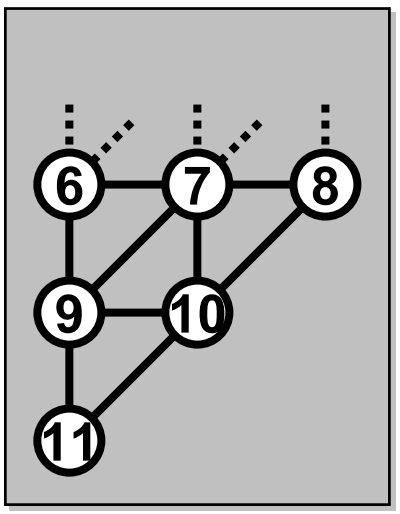


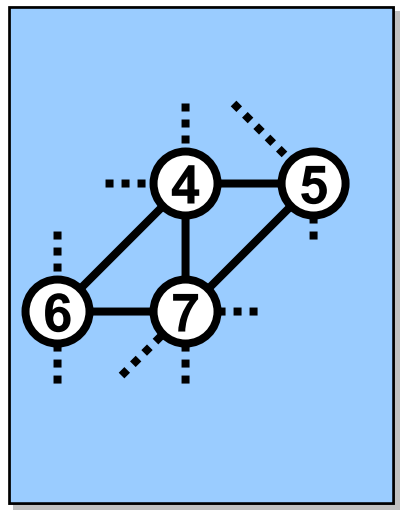


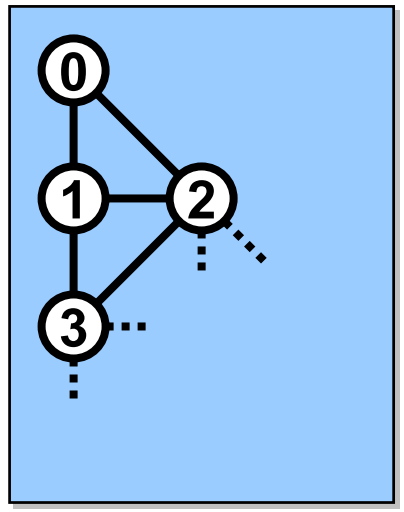


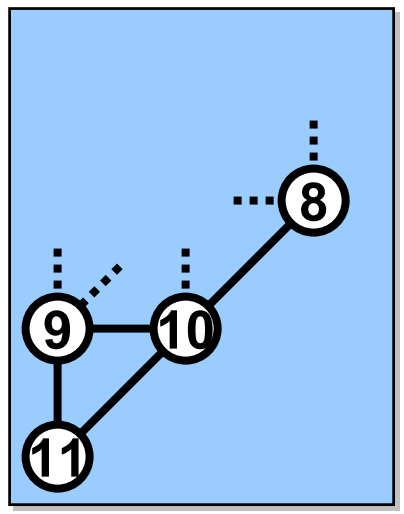
Example #2: Undirected Graph











Big Picture

- What is the largest set of MxN Redistributable components?
- How do we allow developers to add to this set?
- Generality over Performance

Other Complicated Tasks in CS

- GUI APIs
  - Java-AWT: must extend “Frame”
  - Lots of default behaviors inherited
- Threaded Apps
  - Java: class must implement Runnable
  - Must implement methods so JVM can interact with class correctly

Last few Rhetorical Questions

- Can we employ a similar strategy for MxN?

- Corollaries
  - If so, which one?
  - How?

Tale of two Customers

Developer

- Library Developer
- Familiar with Component Tech.
- Formal Software Training

User

- Application Programmer
- May not be aware of using Babel
- May have no formal SW training

Zen of Babel

- If its difficult...
  ... let Babel do it.
- If its impossible for Babel...
  ... let developer do it.
- If its difficult for user...
  ... shift burden to Babel or
  Developer if at all possible

Solution

- Babel provides a MUX in its runtime
- Parallel Distributed Components **MUST IMPLEMENT** the MxNRedistributable Interface
- All communication & data redistribution details hidden from user.

except performance, of course! ;-)

Ramifications

- User still programs in SPMD model
- User may not selectively exclude processes or threads from a parallel RMI
- User may not alter communication modes (synch v. asynch, etc.)
  - Determined by Babel
  - Modifiers may be available to developer in SIDL... e.g. “oneway” !

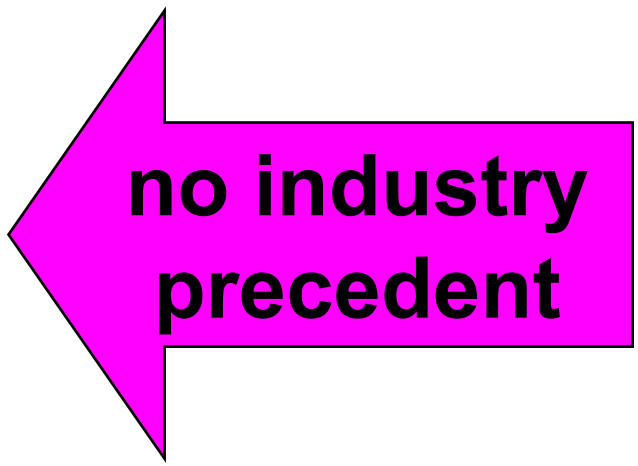
Ramifications (2)

- No MPI communicator connecting M to N
  - May not know M and N when driver is launched
- M = N, M<N, M>N, M=1, N=1, Okay
- Non-rectilinear Data, No Problem
- Owner Computes

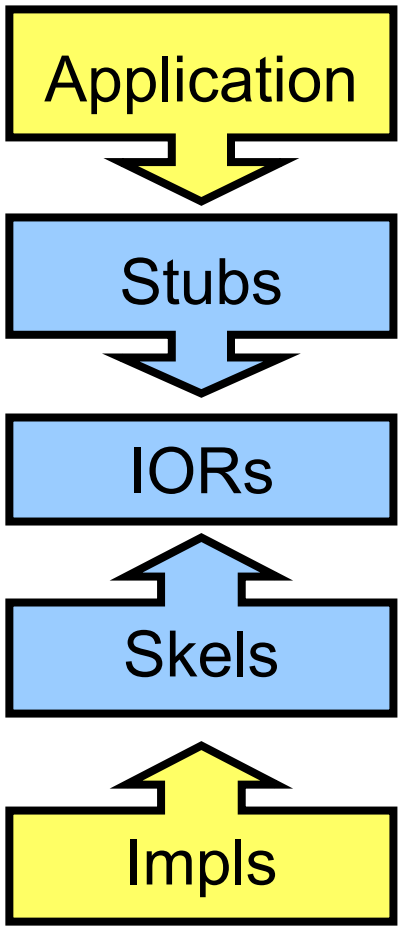
Now Drilling One Level Deeper...

Babel Inserts Code between User & Developer

- language interoperability
  - COMPLEX*(4) == complex<float>
- virtual function dispatch
  - call foo() on interface, dispatch to implementing class
- implement RMI
- remap distributed data

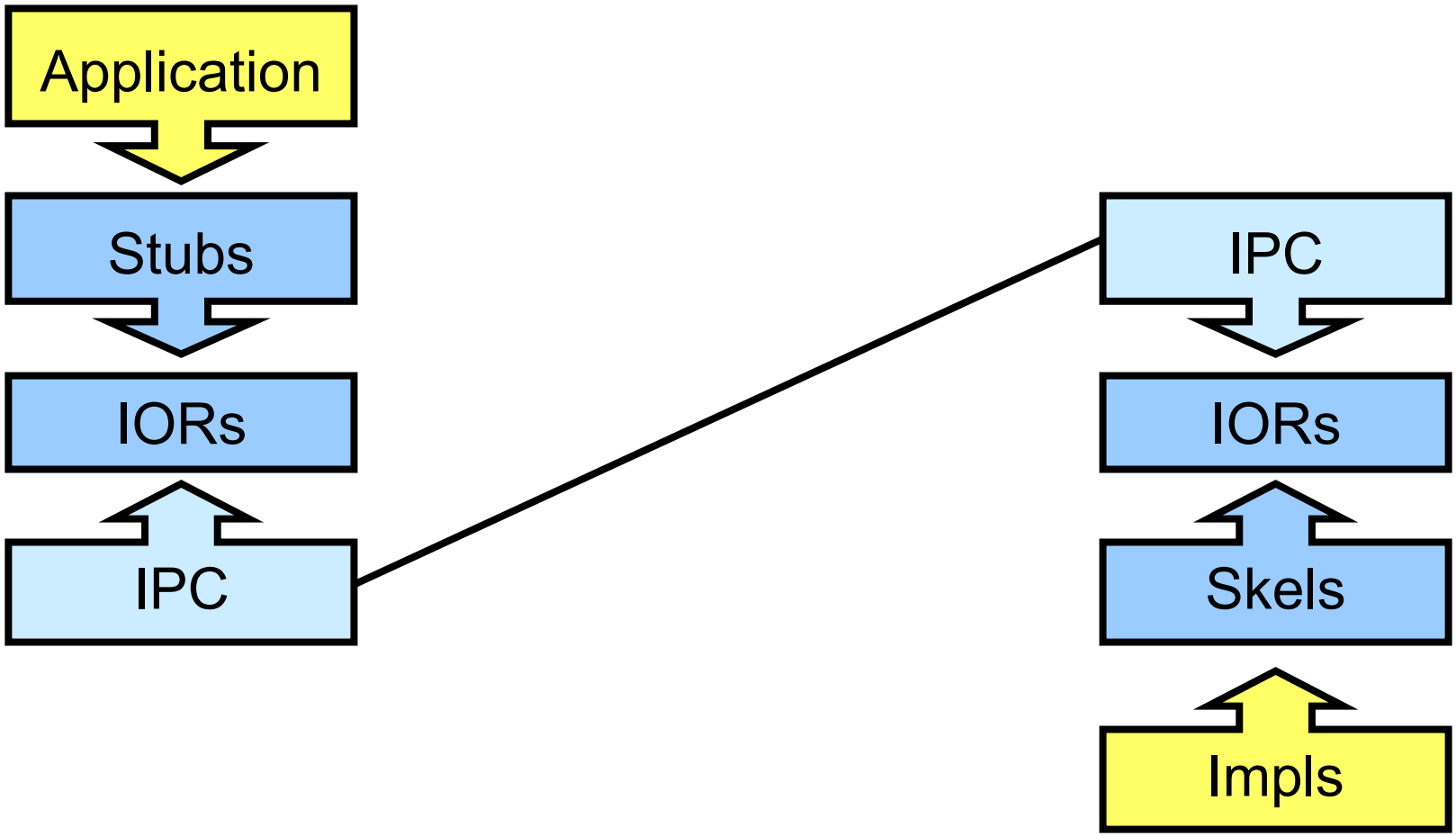


Impls and Stubs and Skels

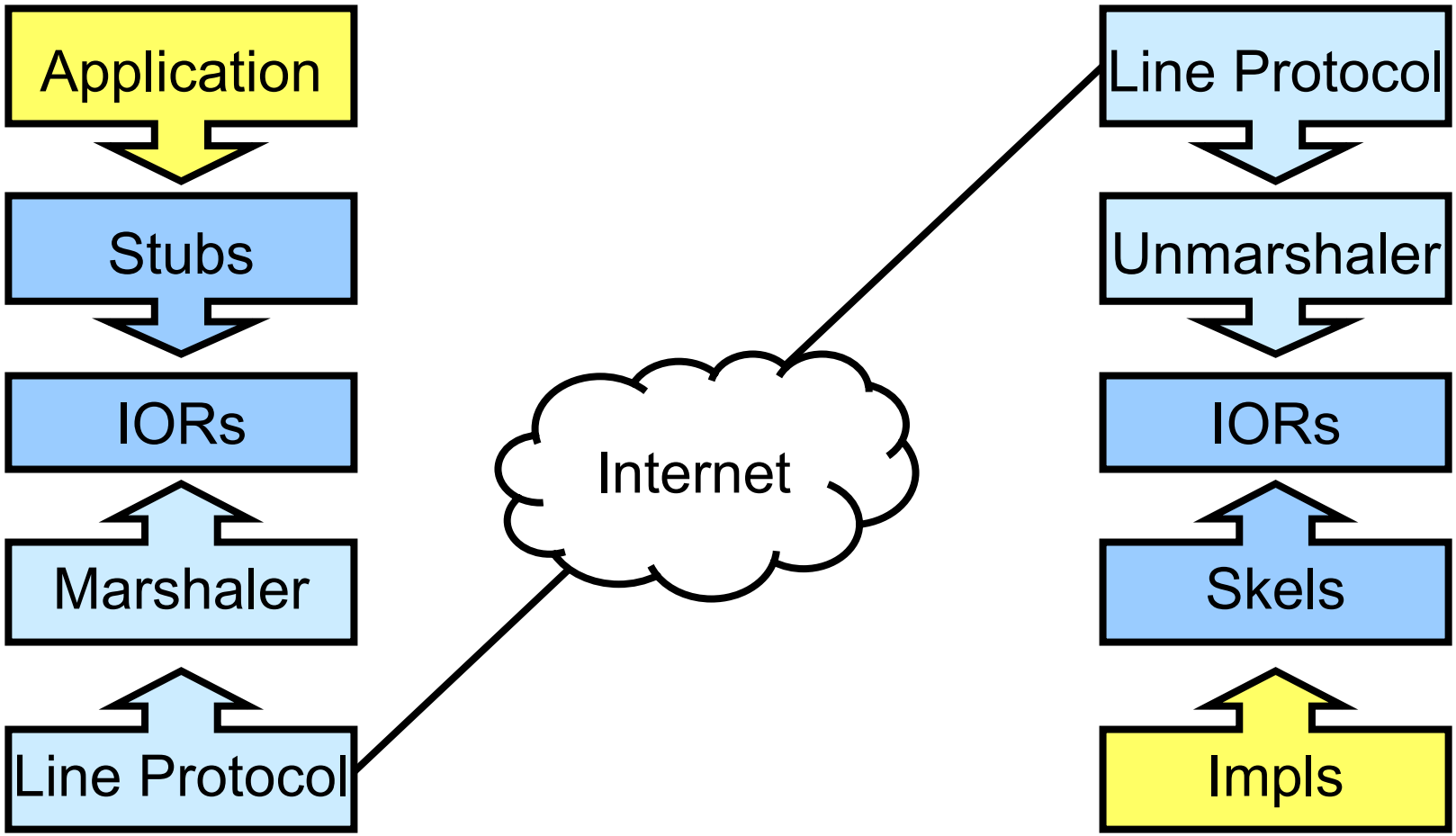


- **Application: uses components in user's language of choice**
- **Client Side Stubs: translate from application language to C**
- **Internal Object Representation: Always in C**
- **Server Side Skeletons: translates IOR (in C) to component implementation language**
- **Implementation: component developers choice of language. (Can be wrappers to legacy code)**

Out of Process Components



Remote Components



All MxN Components inherit one interface

- All Distributed Objects are Containers
- They are by nature subdivisible
- If an interface can make any container “look like” a 1-D vector

  then a MUX is little more than a KELP Mover

MxNRedistributable Interface

```
interface Serializable {

   store( in Stream s );
   load( in Stream s );
};

interface MxNRedistributable extends Serializable {

   int getGlobalSize();
   local int getLocalSize();
   local array<int,1> getLocal2Global();

   split ( in array<int,1> maskVector,
           out array<MxNRedistributable,1> pieces);

   merge( in array<MxNRedistributable,1> pieces);
};
```

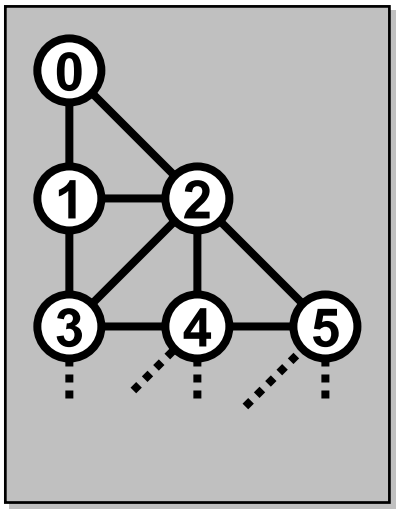
We saw something similar yesterday

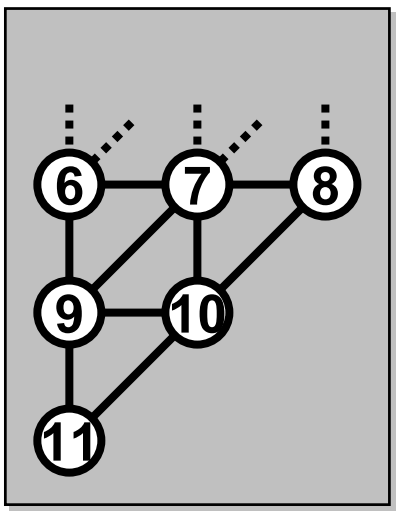
- Kelp uses callbacks for user to define copy, serialization, etc.
- I'm using an interface that user must implement
- Kelp uses rectilinear regions
- I'm using explicit local-global maps (for now)
- Kelp has a Mover
  - most of the "guts" of the solution
  - that depends on the user callbacks
- I'm calling it a MUX

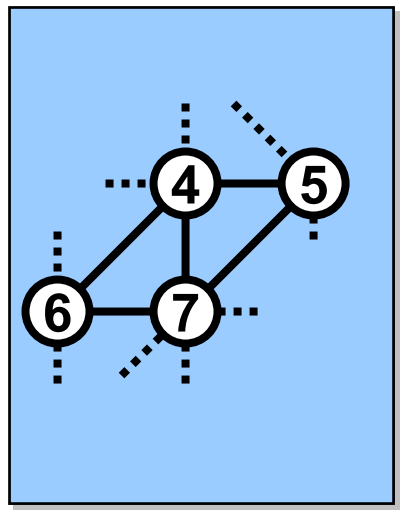
MxNRedistributable Interface in action

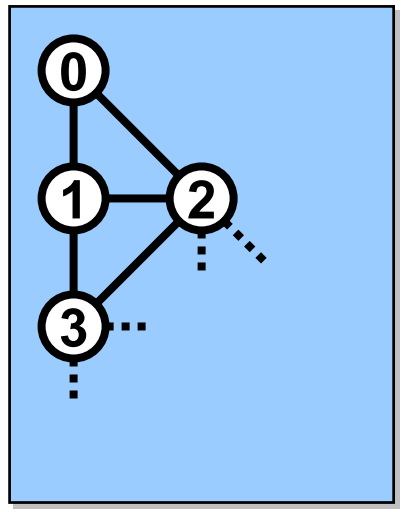
- Last Time
  - I started with object creation
  - I handled problems as they aroze
- This time
  - I'll just do the MxN stuff
  - I'll gloss over details altogether

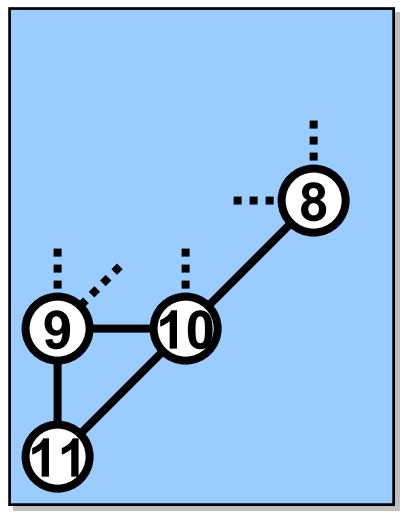
Example #2: Undirected Graph



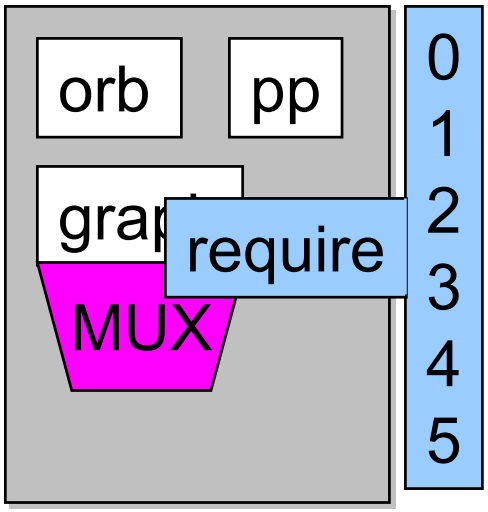


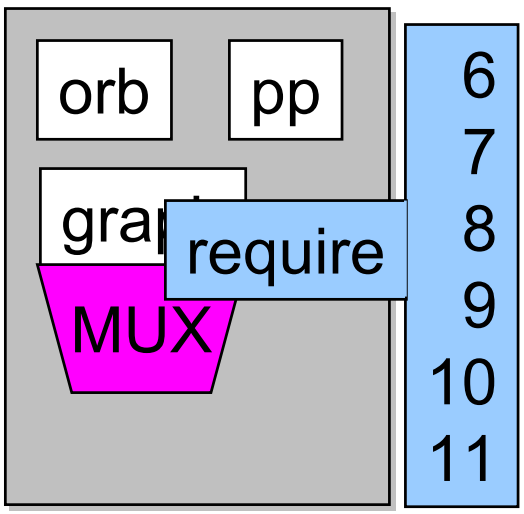




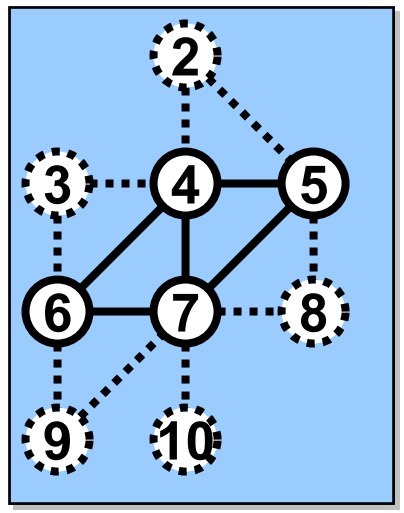


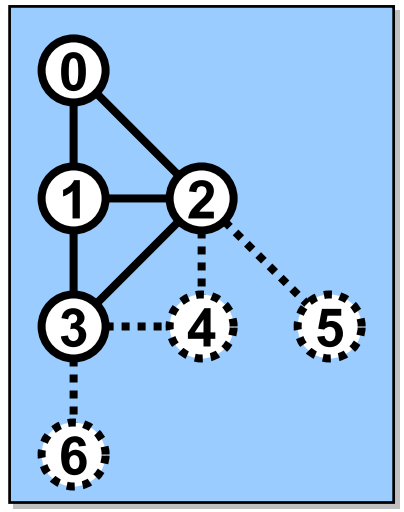
pp->minSpanTree(graph);

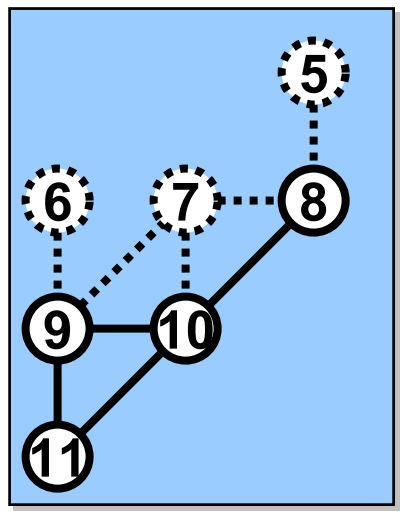




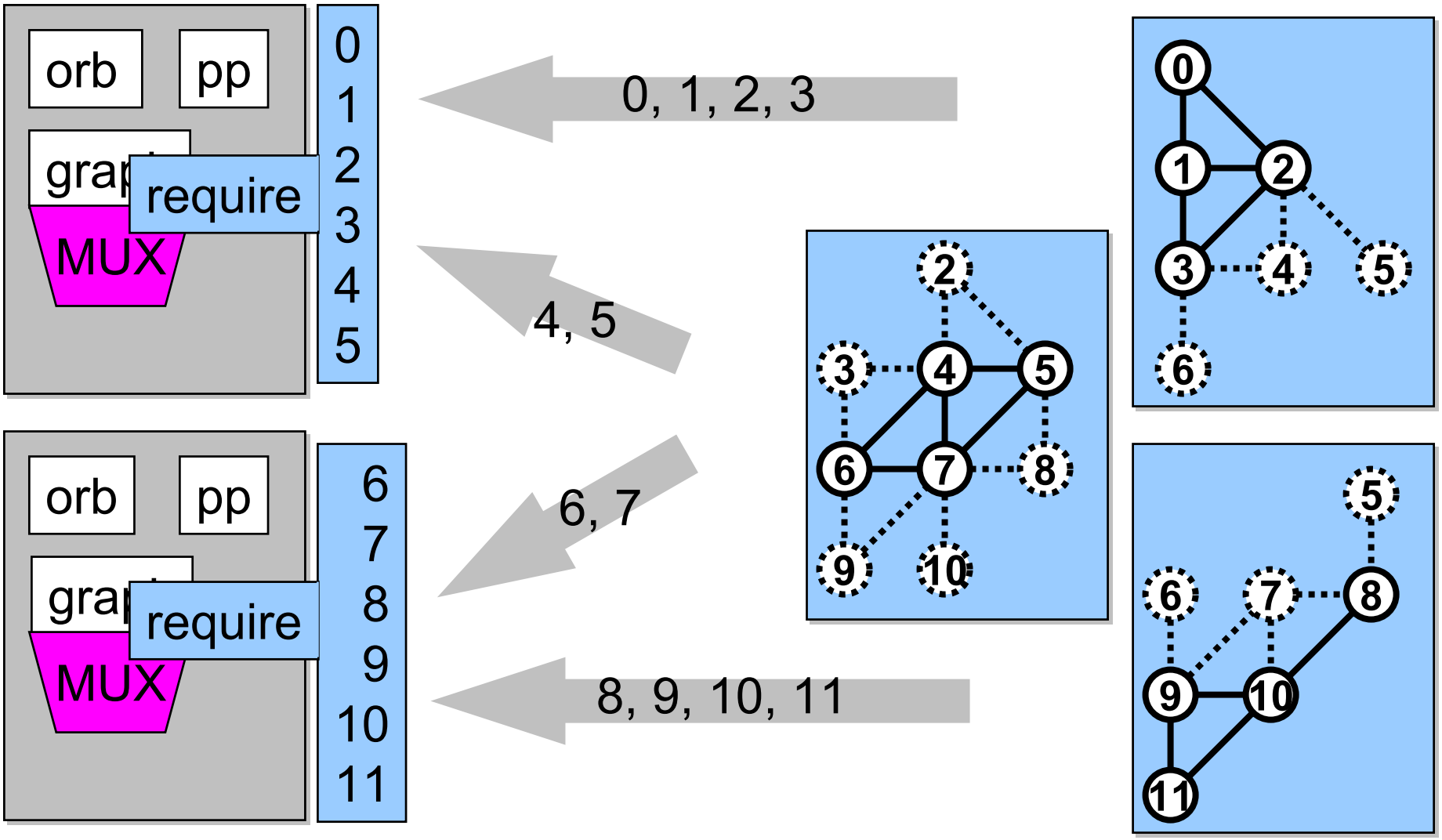
- MUX queries graph for global size (12)
- Graph determines particular data layout (blocked)
- MUX is invoked to guarantee that layout before render implementation is called



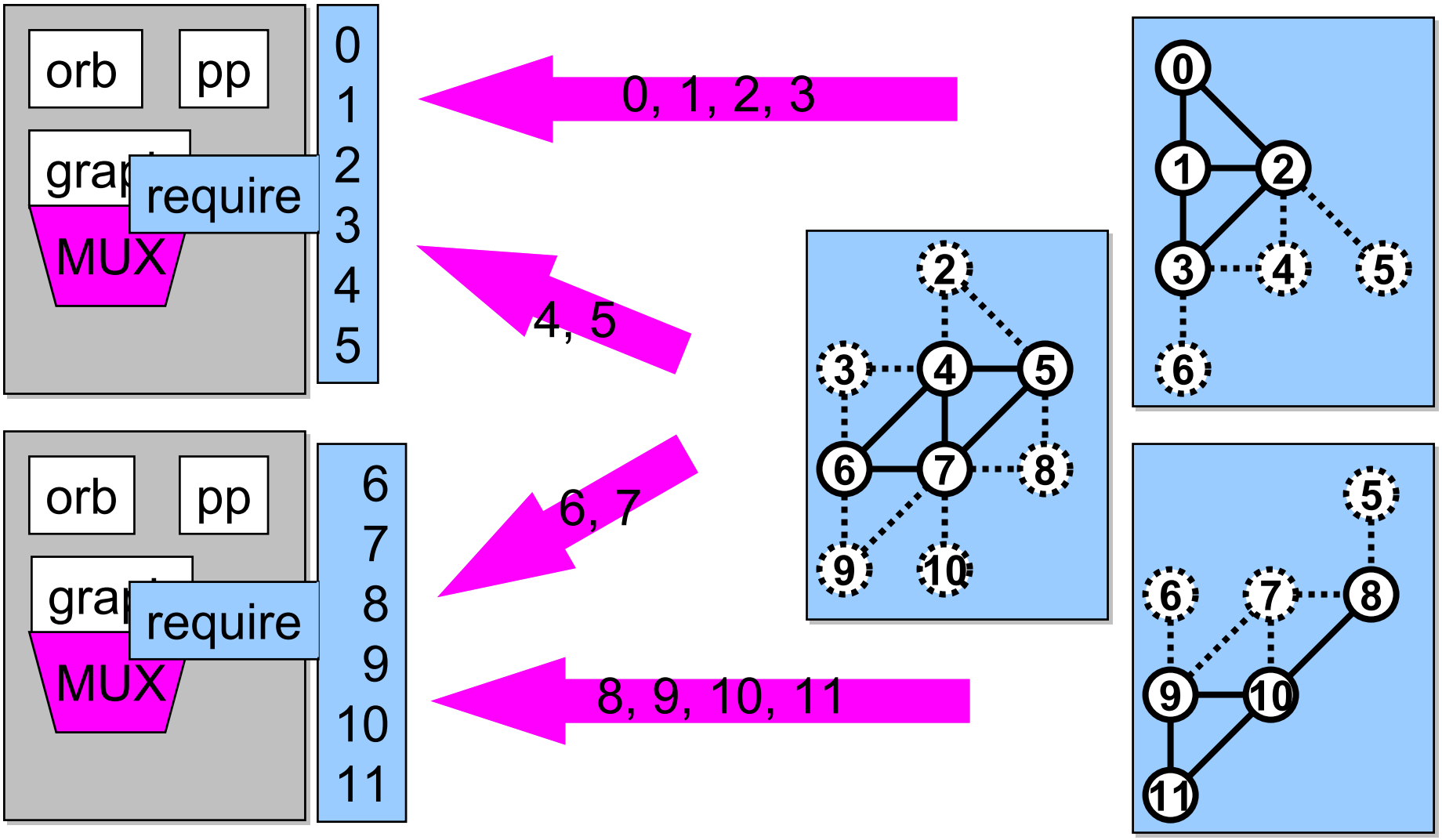




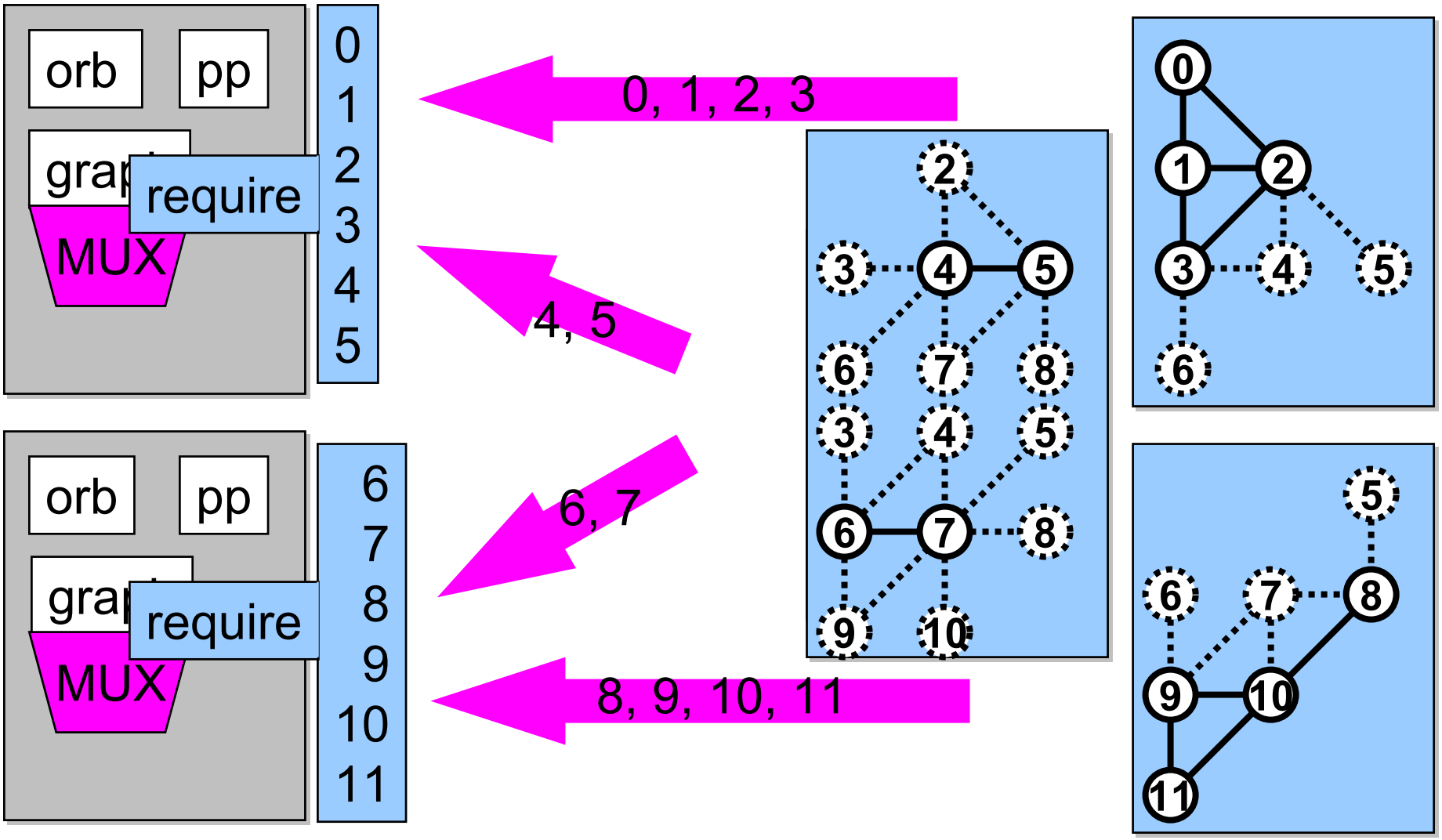
MUX generates communication schedules (client & server)



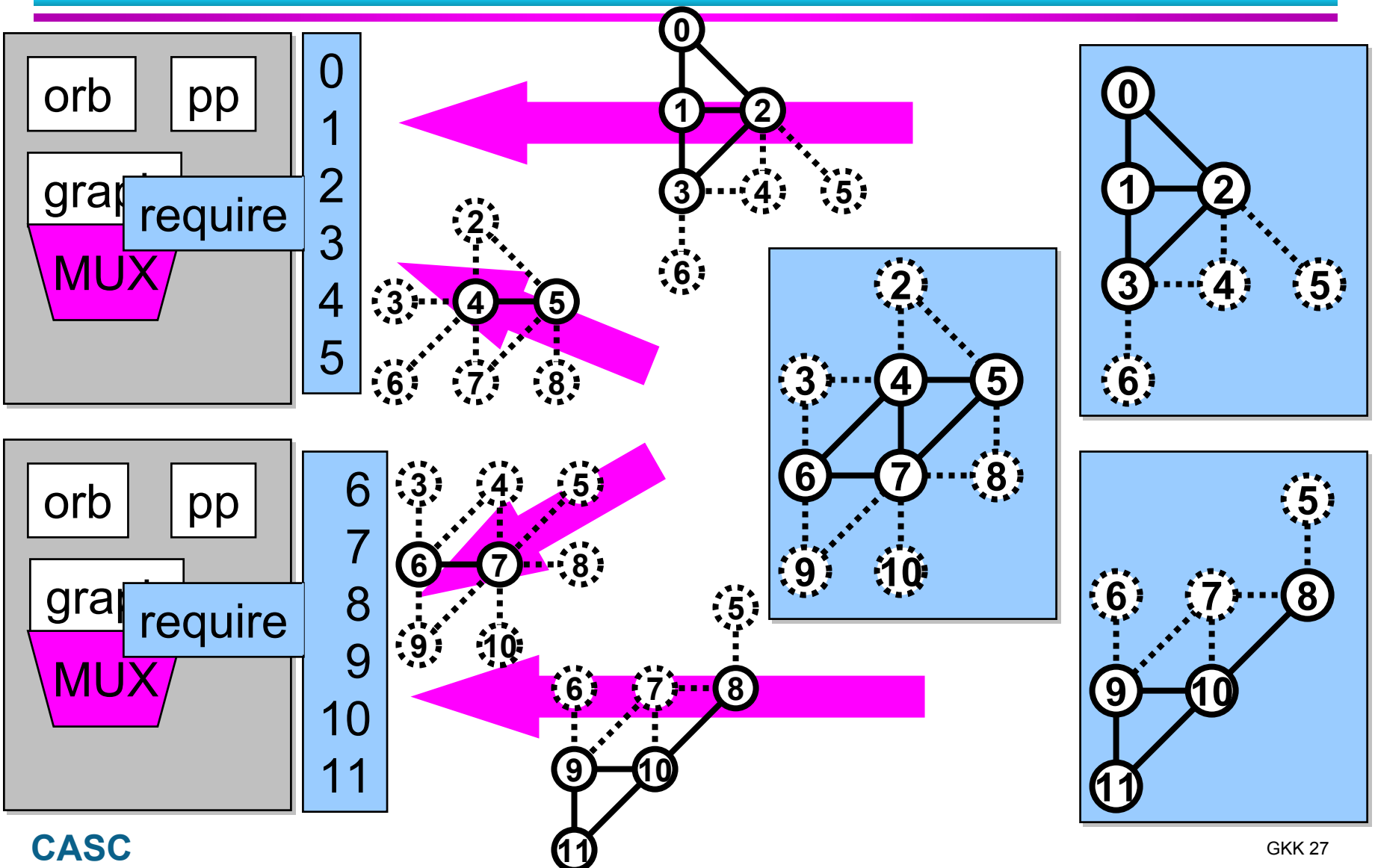
MUX opens communication pipes



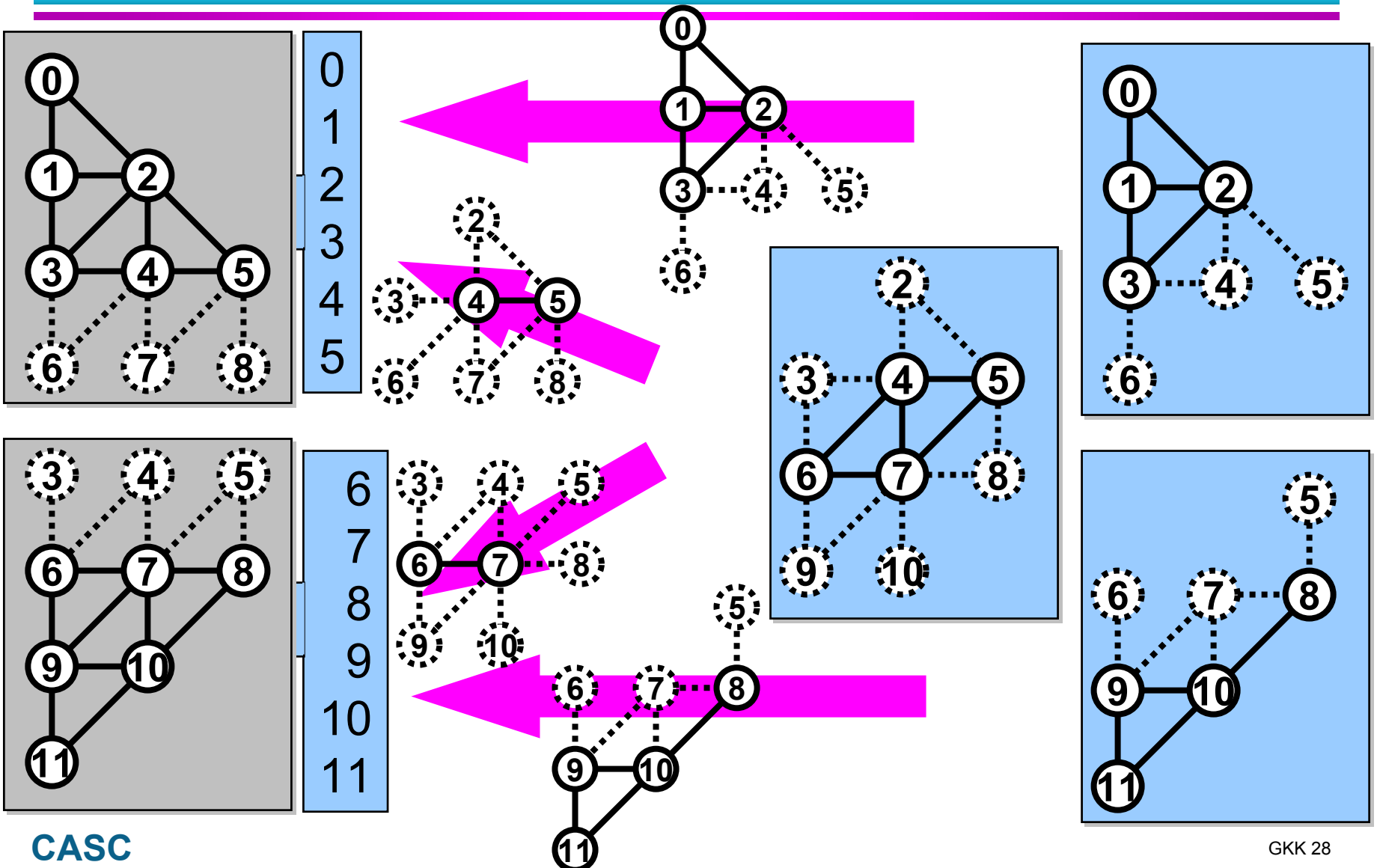
MUX splits graphs with multiple destinations (server-side)



MUX sends pieces through communication pipes (persistance)



MUX receives graphs through pipes & assembles them (client side)



Summary

- All distributed components are containers and subdivisable
- The smallest globally addressable unit is an atom
- MxNRedistributable interface reduces general component MxN problem to a 1-D array of ints
- MxN problem is a special case of the general problem N handles to M instances
- Babel is uniquely positioned to contribute a solution to this problem

Tentative Research Strategy

Fast Track

- Java only, no Babel
  - serialization & RMI built-in
- Build MUX
- Experiment
- Write Paper

Sure Track

- Finish 0.5.x line
- add serialization
- add RMI
- Add in technology from Fast Track

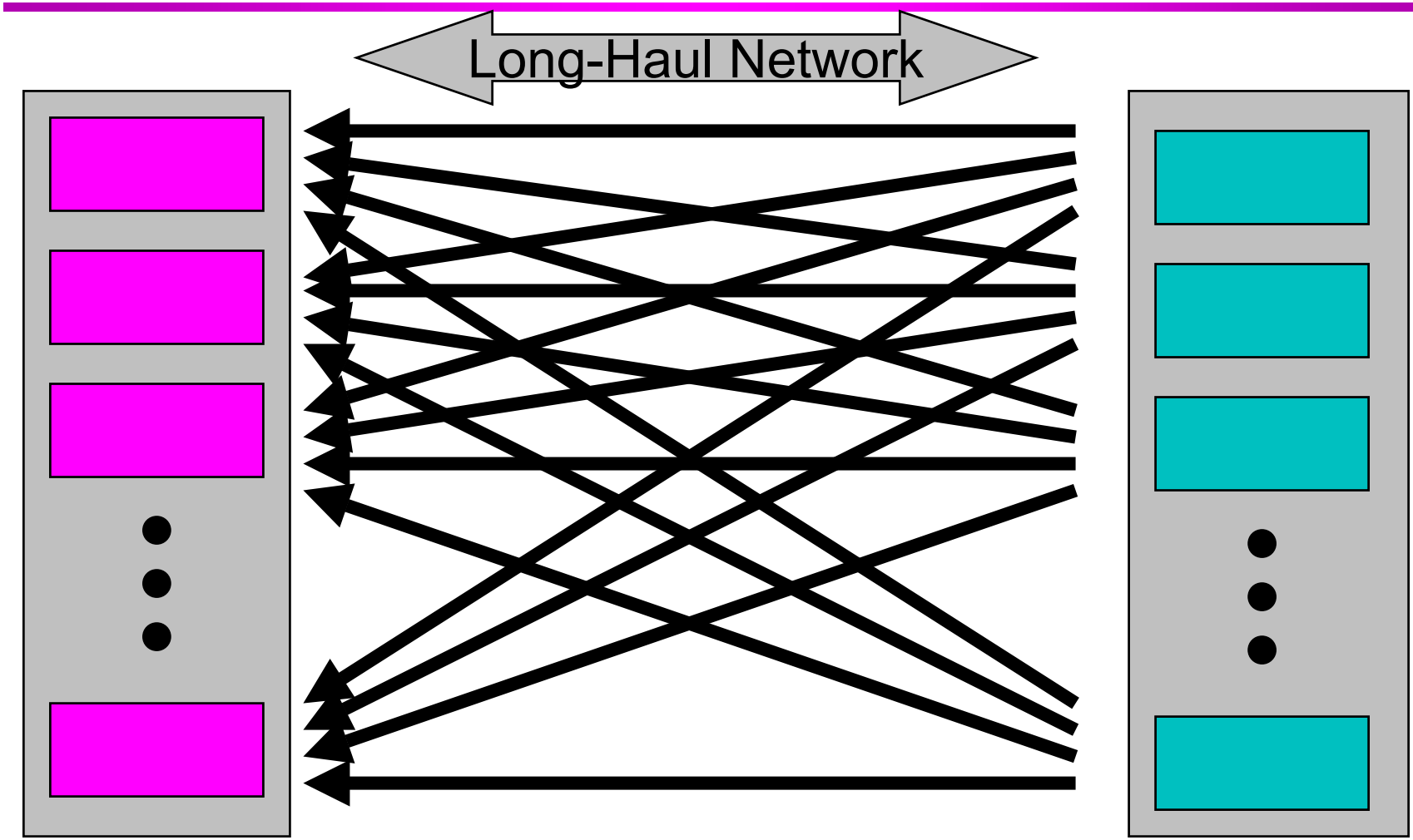
Closing Remarks

- User calls without regard to data distribution
- Developers code assuming data distributed appropriately
- Babel does all the redistribution between the two
  - Requires Developers implementing an interface (a.k.a. callbacks)

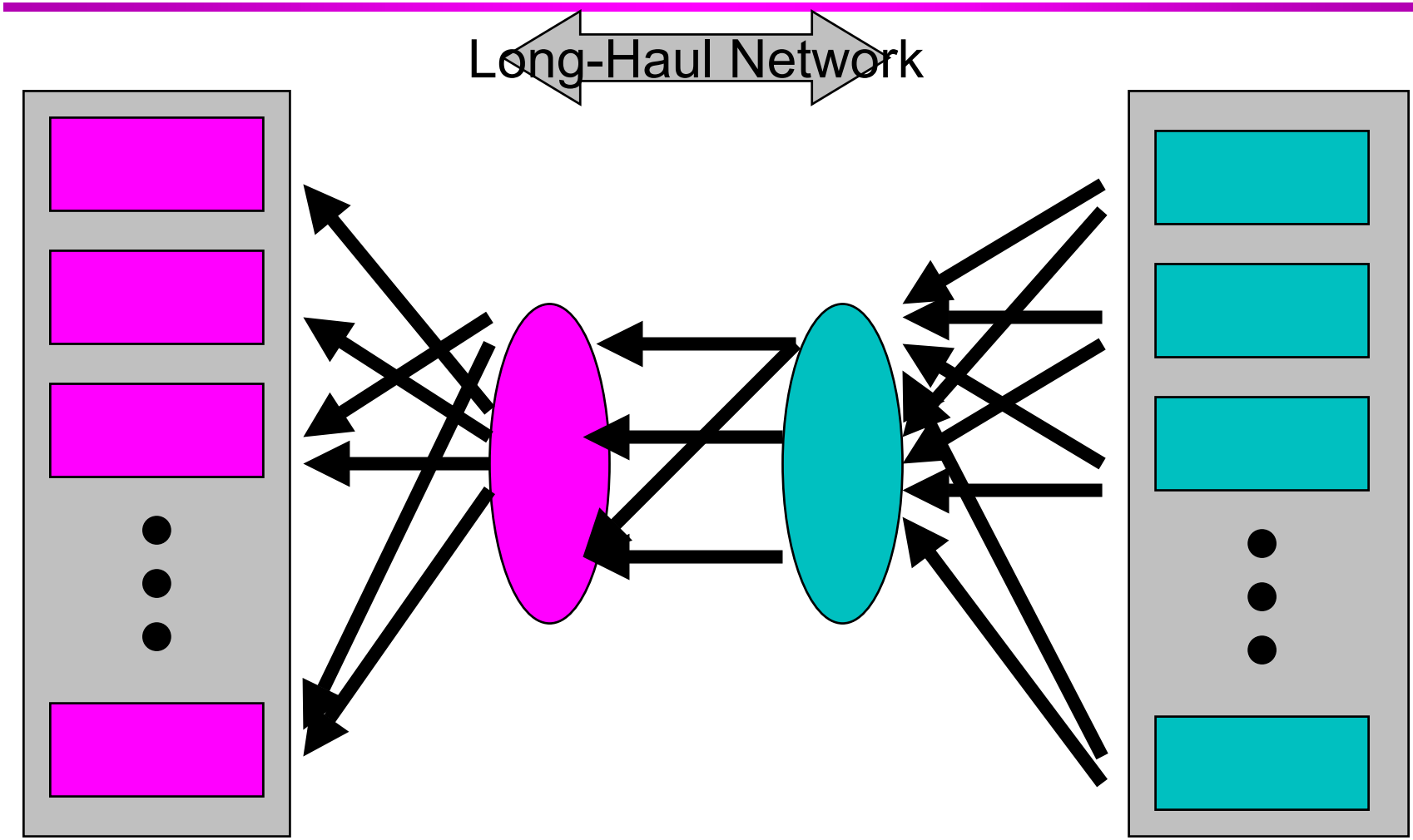
Open Questions

- Non-general, Optimized Solutions
- Client-side Caching issues
- Fault Tolerance
- Subcomponent Migration
- Inter vs. Intra component communication
- MxN , MxP, or MxPxQxN

MxPxQxN Problem



MxPxQxN Problem



The End

Work performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48