The Parallel Boost Graph Library

Trenton W. Ford
CSE 60742

Background: Boost Graph Library (BGL)

- The BGL is part of the Boost C++ Libraries (80+ individual libraries)
- Boost has been active since September 1st, 1999
- Boost has become a testing ground for some future C++ STL changes
- The BGL was started by **Lie-Quan Lee** during his graduate studies at Notre Dame, and **Jeremy Siek** while a Ph.D student at Indiana University @ Bloomington in 1999

Background: Parallel Boost Graph Library (PBGL)

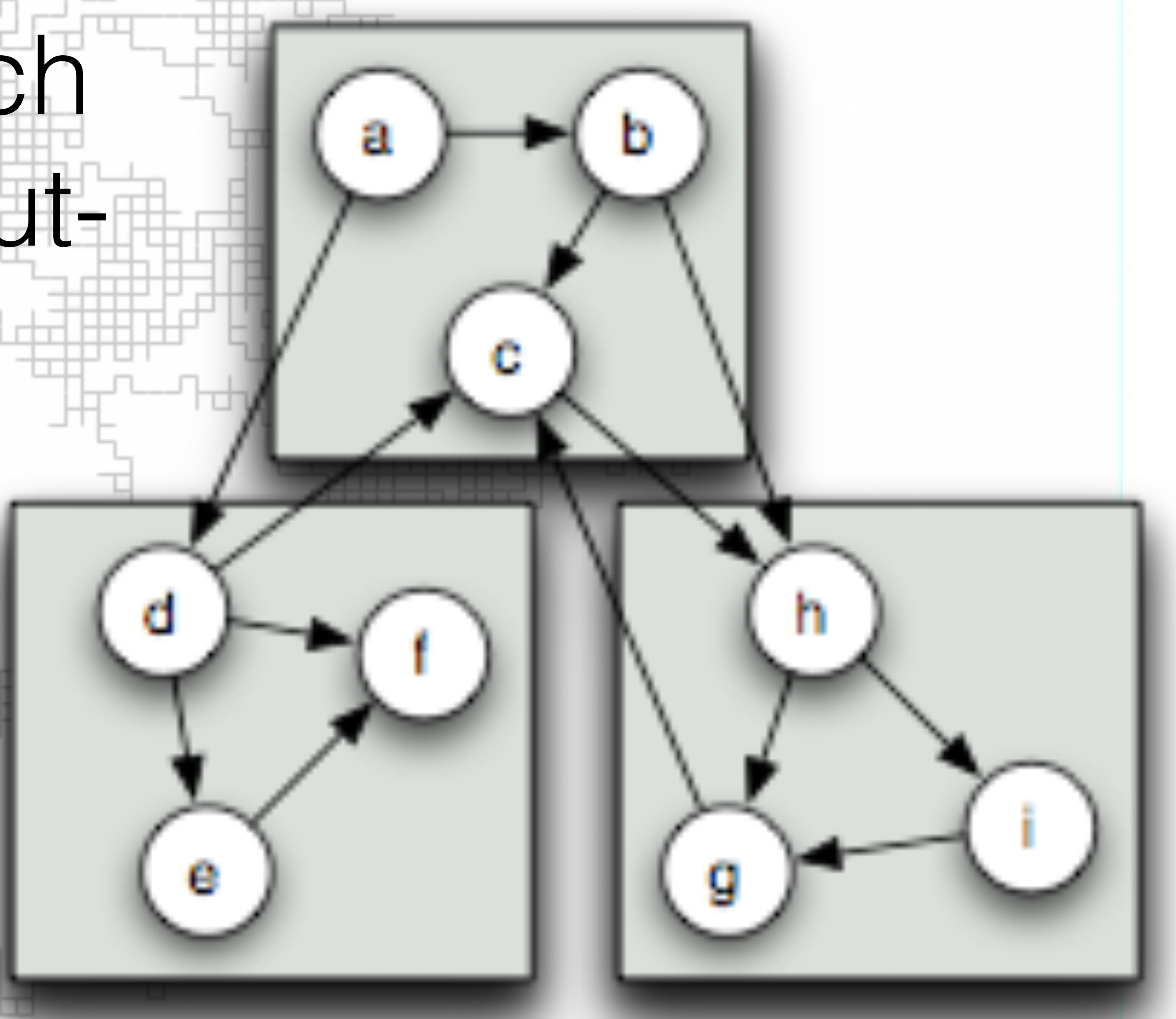
- PBGL has been available since 2008
- The Parallel BGL was developed by **Doug Gregor(Apple)** and **Andrew Lumsdaine(PNNL)** while post-docs at Indiana University @ Bloomington in 2008

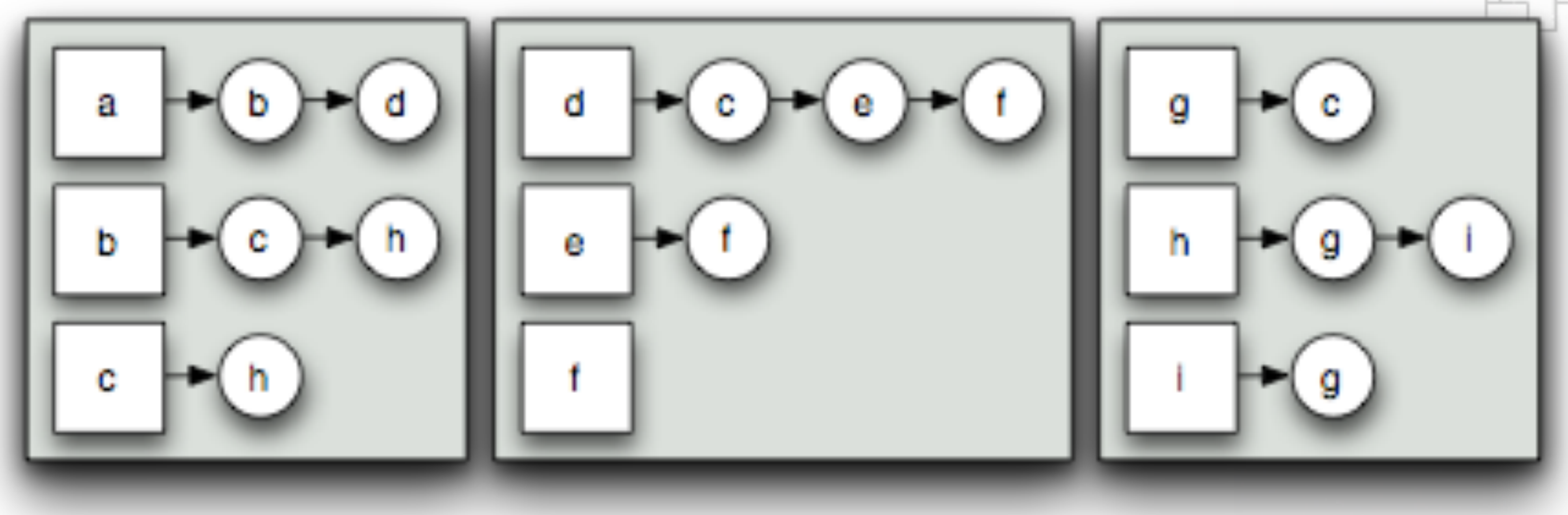
Parallel BGL Paradigms

- PBGL applies the paradigm of generic programming to provide a library that allows **distributed computation** on graphs

- PBGL does not natively support sharing graph data structures across compute resources, but each resource works on their own graph data structure

- Graphs are stored as distributed adjacency lists where each compute resource is given ownership of a vertex and its out-going edges. The distribution is normally arbitrary.





Parallel BGL Paradigm: Generics

PBGL applies the paradigm of generic programming:

template function parameterized on InputIterator and T.

return type T

```
template< typename InputIterator, typename T>
T sum( InputIterator first, InputIterator last, T s )
{
```

```
vector <int> y(10);
int result =  sum( y.begin(), y.end(), 0 );
```

Parallel BGL Paradigm: Nodes and Vertices

Basic Graph Objects and Functions:

Expression	Return Type or Description
`boost::graph_traits<G>::vertex_descriptor`	The type for vertex representative objects.
`boost::graph_traits<G>::edge_descriptor`	The type for edge representative objects.
`add_edge(u, v, g)`	`std::pair<edge_descriptor, bool>`
`add_vertex(vp, g)`	`vertex_descriptor`

Parallel BGL Paradigm: Nodes and Vertices

Selecting Individual Nodes:

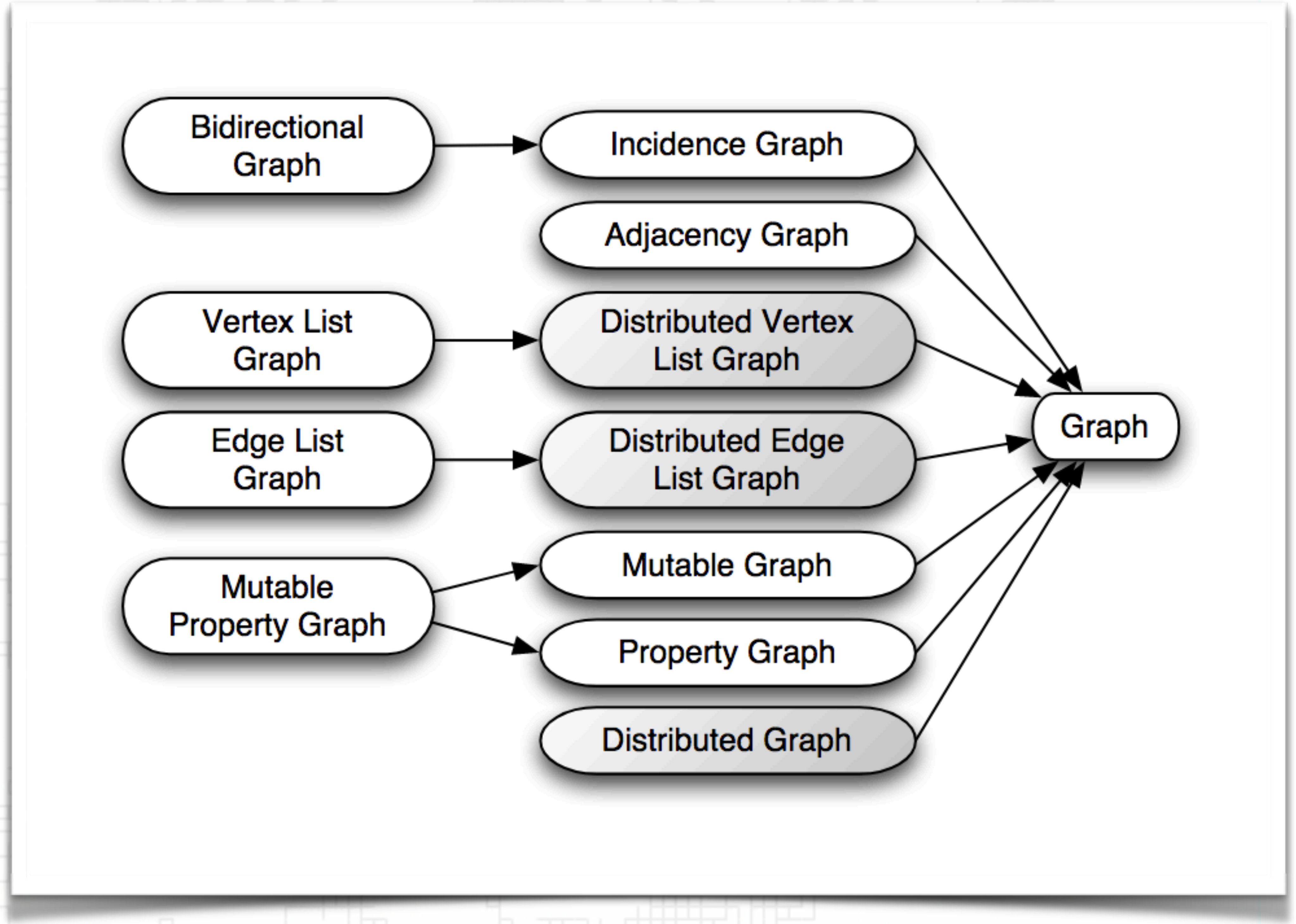
```
boost::graph_traits<UndirectedGraph>::vertex_descriptor u, v;
u = vertex(0, undigraph);
v = vertex(1, undigraph);
```

Adding Individual Edges:

```
add_edge(undigraph, u, v, Weight(3.1));
```

Parallel BGL Paradigm: Generics and Concept Taxonomies.

PBGL generics form hierarchies where children have **more strict** requirements than their parents



Generic Graph Requirements:
Must have associated types that name vertices and edges (called vertex and edge descriptors), along with some additional identification information

Distributed Edge List Graph Requirements:
Requires that the set of edges local to a process be accessible in constant time. The union of the edge sets returned on all processes must be the set of all edges and the pairwise intersection of these edge sets must be the empty set.

Parallel BGL Paradigm: From Generics to Graphs

PBGL generics form hierarchies where children have **more strict** requirements than their parents

Graph Type (adjacency_list):

listS & vecS create a linked list as storage for edge and node information

```
typedef adjacency_list< /*edge storage =*/listS,
                        /*vertex storage =*/vecS,
                        /*directedness =*/bidirectionalS,
                        property<vertex_distance_t, double>,
                        property<edge_weight_t, double>> Graph;
```

property maps attach properties to each vertex, edge, or graph

adding weight property, of datatype double to all edges

Parallel BGL Paradigm: From Generics to Graphs

PBGL generics form hierarchies where children have **more strict** requirements than their parents

Graph Type (distributed_adjacency_list):

the only change is the storage container used for vertices

```
typedef adjacency_list< /*edge storage =*/listS,
                        /*vertex storage =*/distributedS<mpi::bsp_process_group, vecS>,
                        /*directedness =*/bidirectionalS,
                        property<vertex_distance_t, double>,
                        property<edge_weight_t, double>> DistGraph;
```

property maps attach properties to each vertex, edge, or graph

adding weight property, of datatype double to all edges

Parallel BGL Paradigm: When did MPI get here?

```
typedef adjacency_list< /*edge storage =*/listS,
                        /*vertex storage =*/distributedS<mpi::bsp_process_group, vecS>,
                        /*directedness =*/bidirectionalS,
                        property<vertex_distance_t, double>,
                        property<edge_weight_t, double>> DistGraph;
```

Process groups abstract the notion of several processes cooperating to perform some computation.

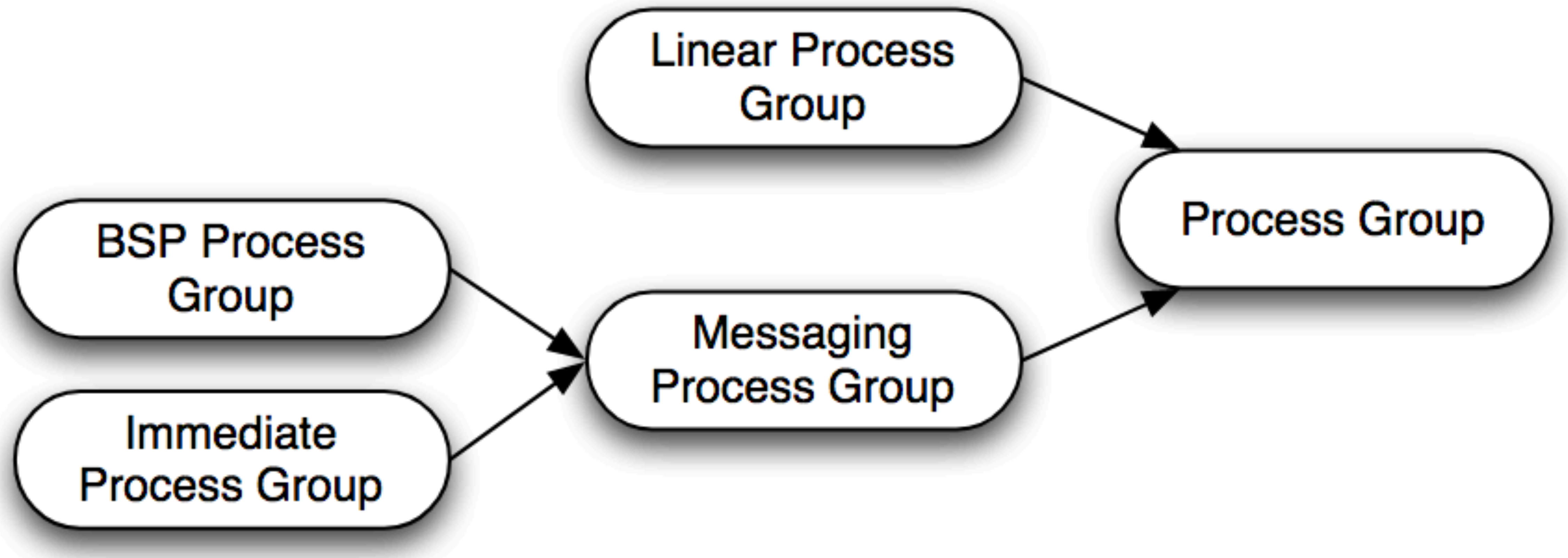
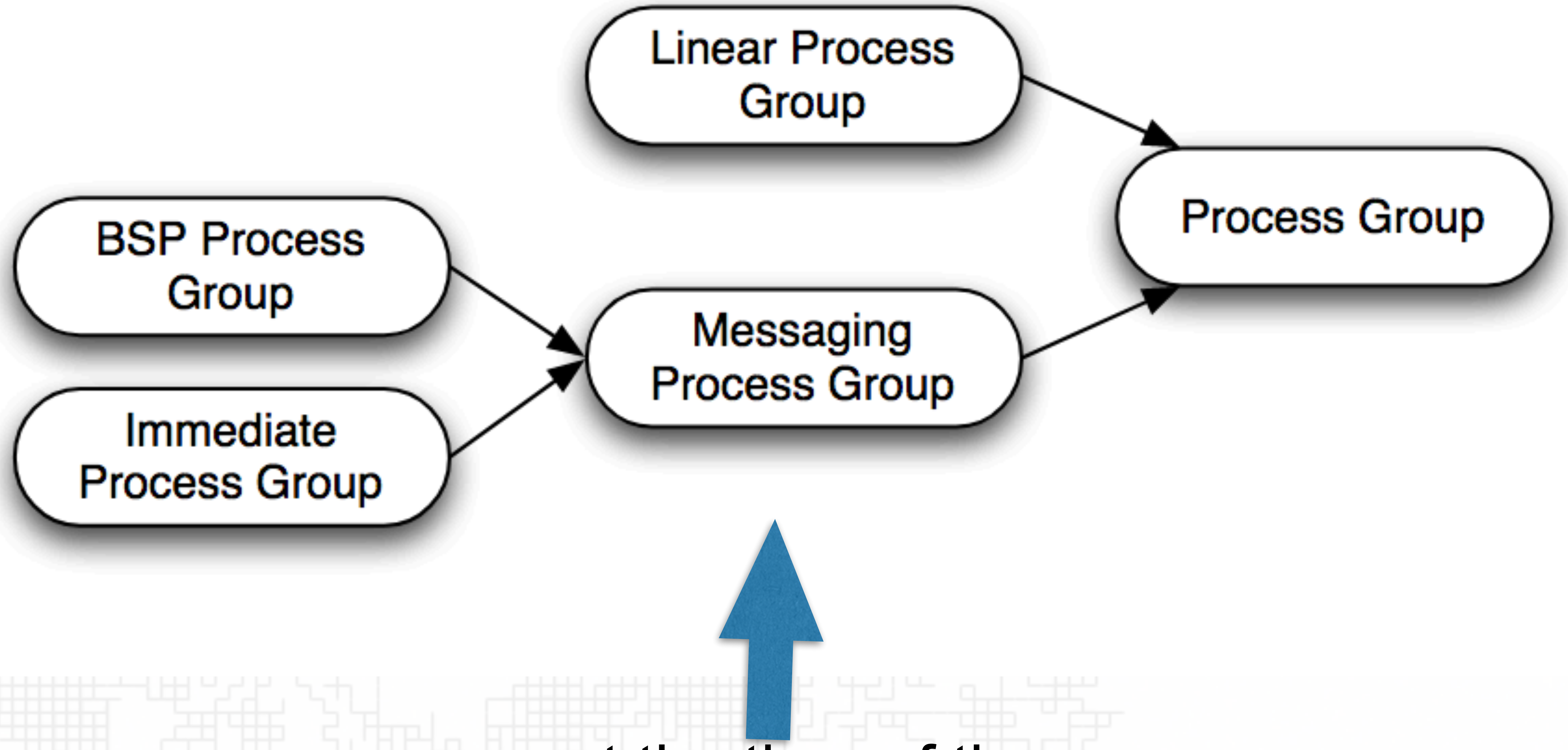


Figure 4: Partial Parallel BGL process group concept taxonomy.

Parallel BGL Paradigm: When did MPI get here?

To handle the Message Process Group, several message types were required.

•**send**(pg,dest,tag,value)
•**receive**(pg,source,tag,value)
•**probe**(pg)
•**synchronize**(pg)



at the time of the reference paper, the authors were mainly focused on the MPG

Parallel BGL Paradigm: Message Passing Commands

- **send**(pg,dest,tag,value):Send the given value in a message marked with the given numerical tag to the process with identifier dest. Messages with a given (source, dest) pair are guaranteed to be received in the order sent.
- **receive**(pg,source,tag,value): Receive a message containing value from process source with the given tag.
- **probe**(pg):Immediately returns a (source, tag)pair if a message is available, or a no-message indicator.
- **synchronize**(pg):Collectively waits until all messages sent by any process are stored in a buffer at their destinations. All messages sent prior to synchronization may be immediately received after synchronization.

Parallel BGL: Using an Algorithm on a Graph

Parallel BGL: Using an Algorithm on a Graph

8.Algorithms

- Distributed algorithms
  - Breadth-first search
  - Dijkstra's single-source shortest paths
    - Eager Dijkstra shortest paths
    - Crauser et al. Dijkstra shortest paths
    - Delta-Stepping shortest paths
  - Depth-first search
  - Minimum spanning tree
    - Boruvka's minimum spanning tree
    - Merging local minimum spanning forests
    - Boruvka-then-merge
    - Boruvka-mixed-merge
  - Connected components
    - Connected components
    - Connected components parallel search
    - Strongly-connected components
  - PageRank
  - Boman et al. Graph coloring
  - Fruchterman Reingold force-directed layout
  - s-t connectivity
  - Betweenness centrality
  - Non-distributed betweenness centrality
  -

PBGL References

- **Paper written by Stroustrup, that give a framework for generic programming methodology in C++.** (http://www.stroustrup.com/oopsla06.pdf)

- **Link to the original PBGL Paper.** (https://people.csail.mit.edu/jshun/papers/PBGL.pdf)

- **Boost's Graph Library Online User Guide** (https://www.boost.org/doc/libs/1_68_0/libs/graph/doc/)

- **METIS Graph Format Explanation** (https://people.sc.fsu.edu/~jburkardt/data/metis_graph/metis_graph.html)

Questions?