Lifting for Parallelism

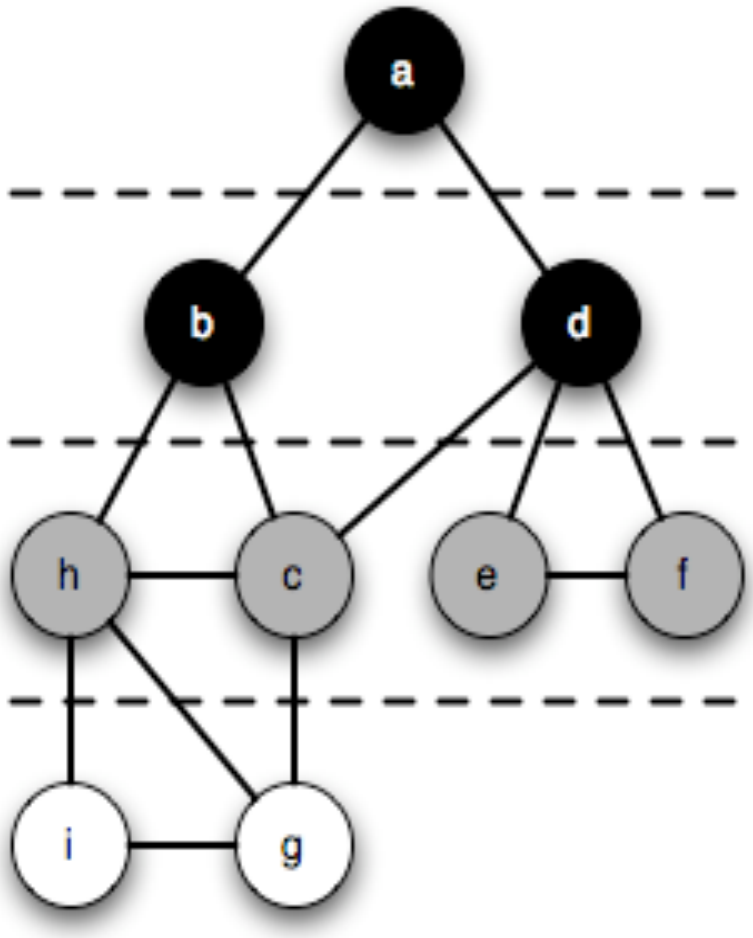
- Remove assumptions made by most sequential algorithms:
  - A single, shared address space.
  - A single “thread” of execution.

- Our goal: Build the Parallel BGL by lifting the sequential BGL.

pervasive technology labs
AT INDIANA UNIVERSITY

Breadth-First Search

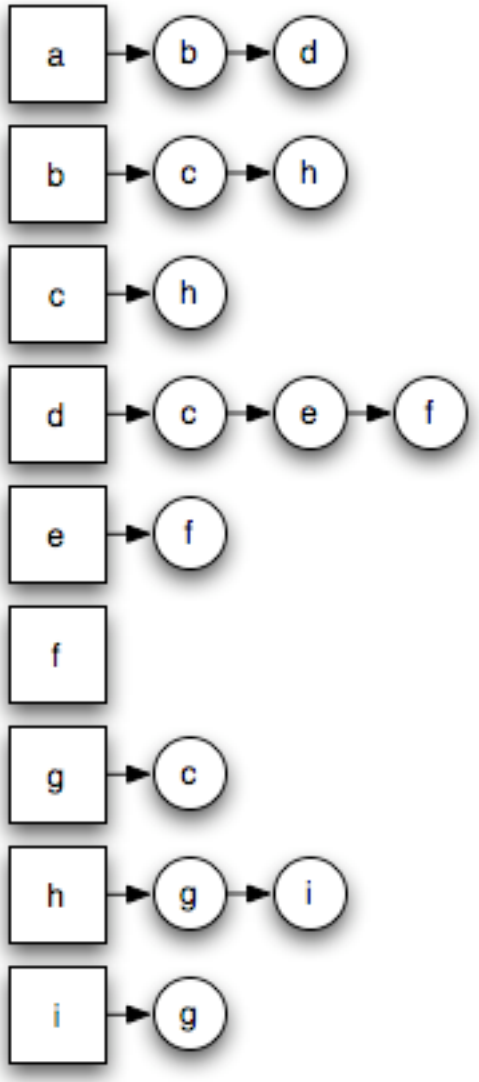
```
put(color, s, Color::gray());
Q.push(s);
while (! Q.empty()) {
  Vertex u = Q.top(); Q.pop();
  for (tie(ei, ei_end) = out_edges(u, g);
    Vertex v = target(*ei, g);
    ColorValue v_color = get(color, v);
    if (v_color == Color::white()) {
      put(color, v, Color::gray());
      Q.push(v);
    } else {
      if (v_color == Color::gray())
      else
    }
  }
  put(color, u, Color::black());
}
```



pervasivetechnologylabs
AT INDIANA UNIVERSITY

Parallellizing BFS?

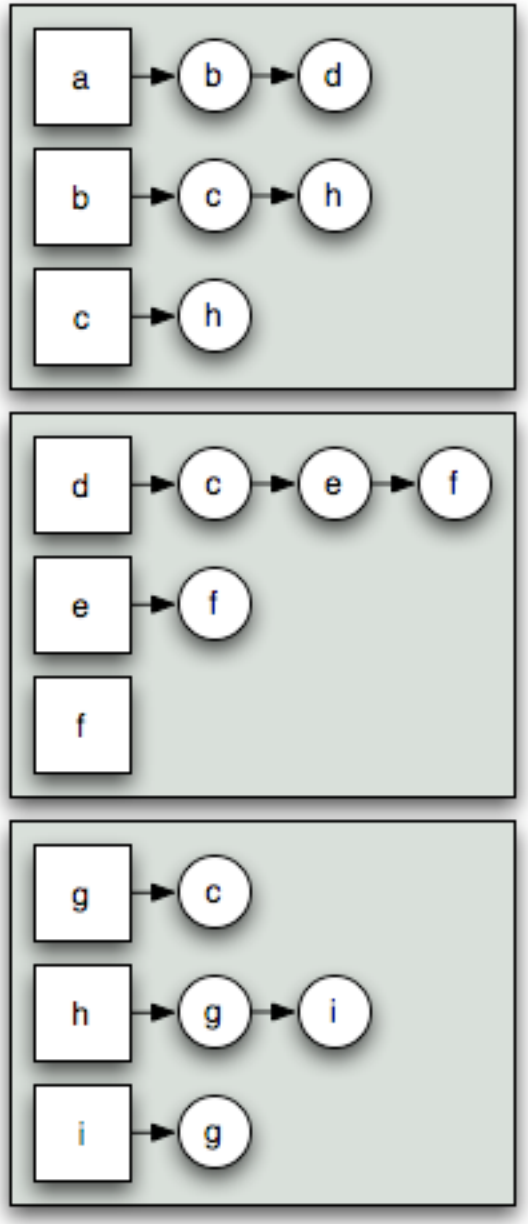
```
put(color, s, Color::gray());
Q.push(s);
while (! Q.empty()) {
  Vertex u = Q.top(); Q.pop();
  for (tie(ei, ei_end) = out_edges(u, g); ei != ei_er
    Vertex v = target(*ei, g);
    ColorValue v_color = get(color, v);
    if (v_color == Color::white()) {
      put(color, v, Color::gray());
      Q.push(v);
    } else {
      if (v_color == Color::gray())
      else
    }
  }
  put(color, u, Color::black());
}
```



pervasivetechnologylabs
AT INDIANA UNIVERSITY

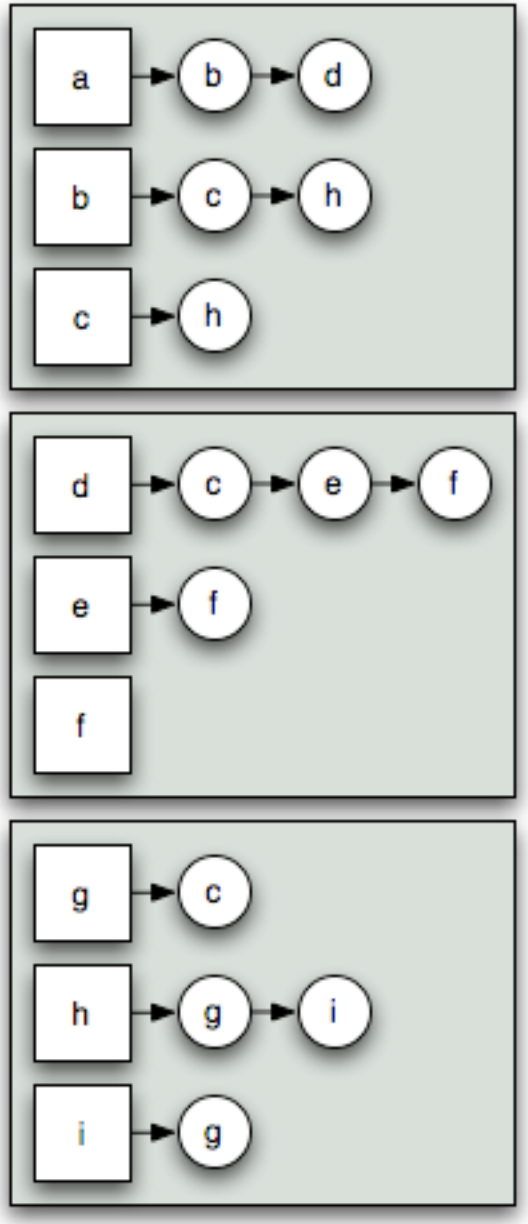
Parallellizing BFS?

```
put(color, s, Color::gray());
Q.push(s);
while (! Q.empty()) {
  Vertex u = Q.top(); Q.pop();
  for (tie(ei, ei_end) = out_edges(u, g); ei != ei_e
    Vertex v = target(*ei, g);
    ColorValue v_color = get(color, v);
    if (v_color == Color::white()) {
      put(color, v, Color::gray());
      Q.push(v);
    } else {
      if (v_color == Color::gray())
      else
    }
  }
  put(color, u, Color::black());
}
```



Distributed Graph

- One fundamental operation:
  - Enumerate out-edges of a given vertex
- Distributed adjacency list:
  - Distribute vertices
  - Out-edges stored with the vertices



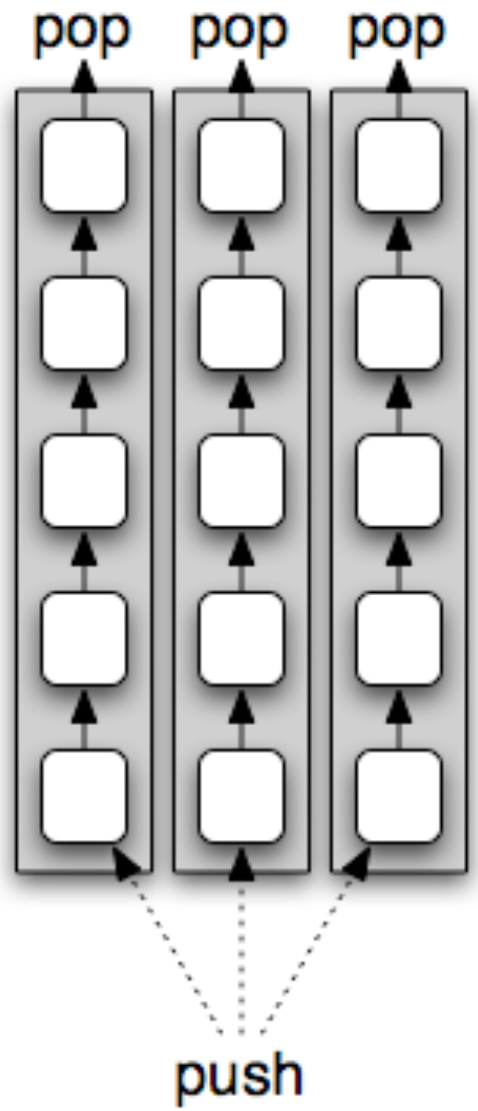
Parallellizing BFS?

```
put(color, s, Color::gray());
Q.push(s);
while (! Q.empty()) {
  Vertex u = Q.top(); Q.pop();
  for (tie(ei, ei_end) = out_edges(u, g); ei != ei_end; ++ei) {
    Vertex v = target(*ei, g);
    ColorValue v_color = get(color, v);
    if (v_color == Color::white()) {
      put(color, v, Color::gray());
      Q.push(v);
    } else {
      if (v_color == Color::gray())
      else
    }
  }
  put(color, u, Color::black());
}
```

pervasivetechnologylabs
AT INDIANA UNIVERSITY

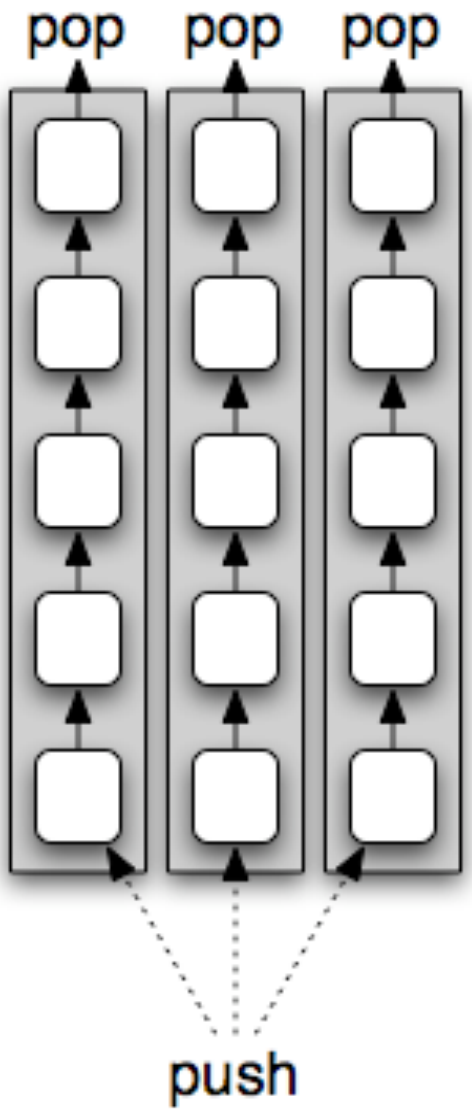
Parallellizing BFS?

```
put(color, s, Color::gray());
Q.push(s);
while (! Q.empty()) {
  Vertex u = Q.top(); Q.pop();
  for (tie(ei, ei_end) = out_edges(u, g); ei != ei_end;
    Vertex v = target(*ei, g);
    ColorValue v_color = get(color, v);
    if (v_color == Color::white()) {
      put(color, v, Color::gray());
      Q.push(v);
    } else {
      if (v_color == Color::gray())
      else
    }
  }
  put(color, u, Color::black());
}
```



Distributed Queue

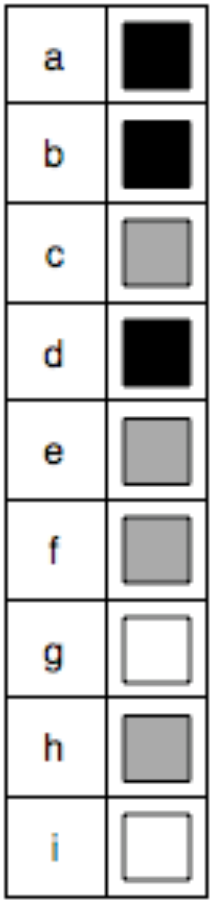
- Three fundamental operations:
  - ***top/pop*** retrieves from queue
  - ***push*** operation adds to queue
  - ***empty*** operation signals termination
- Distributed queue:
  - Separate, local queues
  - ***top/pop*** from local queue
  - ***push*** sends to a remote queue
  - ***empty*** waits for remote sends



pervasivetechnologylabs
AT INDIANA UNIVERSITY

Parallellizing BFS?

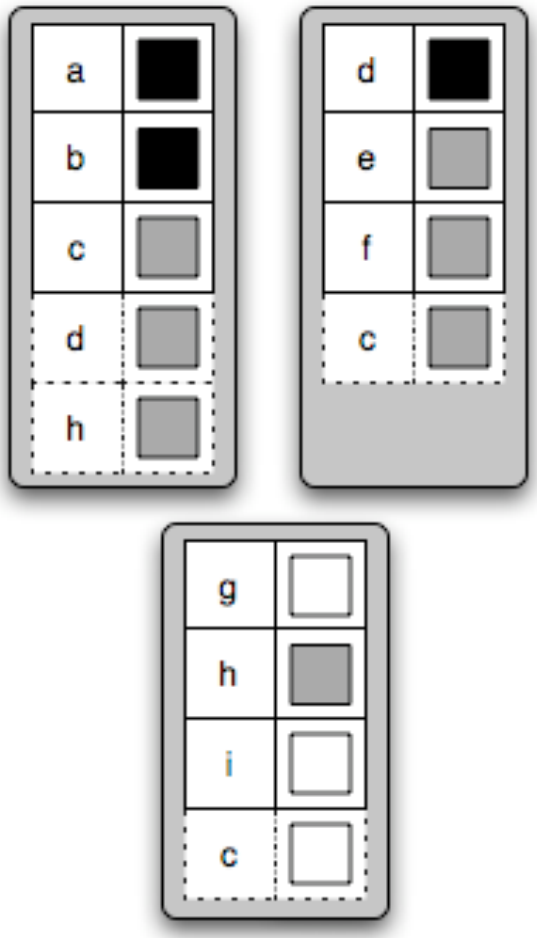
```
put(color, s, Color::gray());
Q.push(s);
while (! Q.empty()) {
  Vertex u = Q.top(); Q.pop();
  for (tie(ei, ei_end) = out_edges(u, g); ei != ei_end; ++ei) {
    Vertex v = target(*ei, g);
    ColorValue v_color = get(color, v);
    if (v_color == Color::white()) {
      put(color, v, Color::gray());
      Q.push(v);
    } else {
      if (v_color == Color::gray())
      else
    }
  }
  put(color, u, Color::black());
}
```



pervasivetechnologylabs
AT INDIANA UNIVERSITY

Parallellizing BFS?

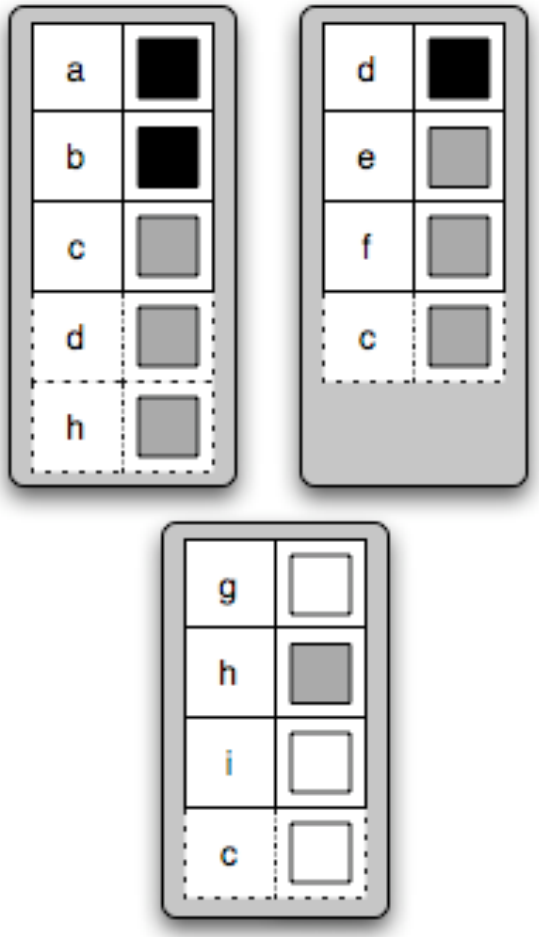
```
put(color, s, Color::gray());
Q.push(s);
while (! Q.empty()) {
  Vertex u = Q.top(); Q.pop();
  for (tie(ei, ei_end) = out_edges(u, g); ei != ei_er
    Vertex v = target(*ei, g);
    ColorValue v_color = get(color, v);
    if (v_color == Color::white()) {
      put(color, v, Color::gray());
      Q.push(v);
    } else {
      if (v_color == Color::gray())
      else
    }
  }
  put(color, u, Color::black());
}
```



pervasivetechnologylabs
AT INDIANA UNIVERSITY

Distributed Property Maps

- Two fundamental operations:
  - ***put*** sets the value for a vertex/edge
  - ***get*** retrieves the value
- Distributed property map:
  - Store data on same processor as vertex or edge
  - ***put/get*** send messages
  - Ghost cells cache remote values
  - Resolver combines **puts**



pervasivetechnologylabs
AT INDIANA UNIVERSITY

“Implementing” Parallel BFS

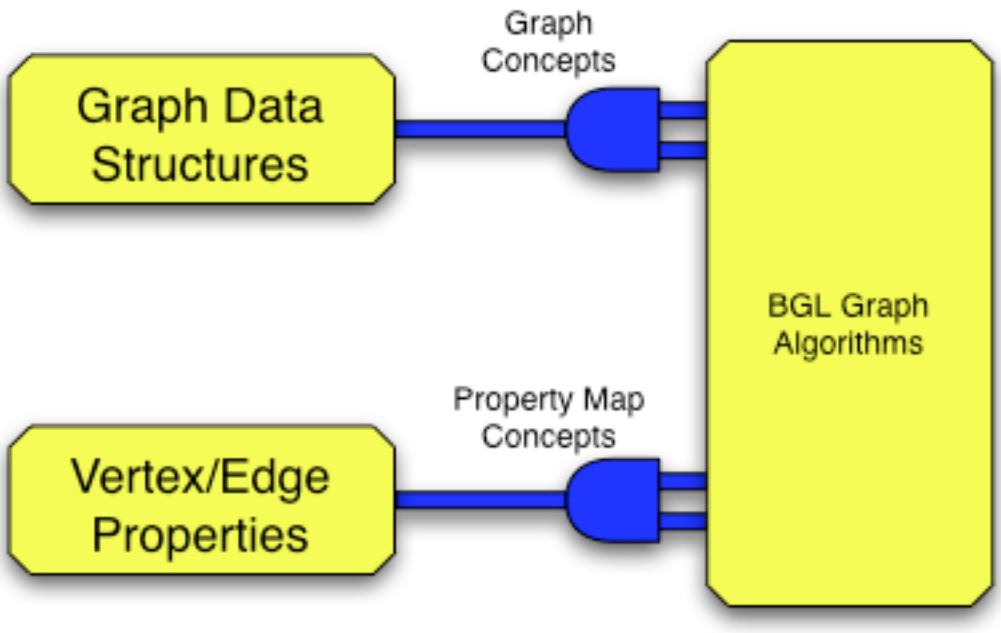
- Generic interface from the Boost Graph Library

```
template<class IncidenceGraph, class Queue, class BFSVisitor,
         class ColorMap>
void breadth_first_search(const IncidenceGraph& g,
                          vertex_descriptor s, Queue& Q,
                          BFSVisitor vis, ColorMap color);
```

- Effect parallelism by using appropriate types:
  - Distributed graph
  - Distributed queue
  - Distributed property map
- Our sequential implementation is also parallel!
  - Parallel BGL can just “wrap up” sequential BFS

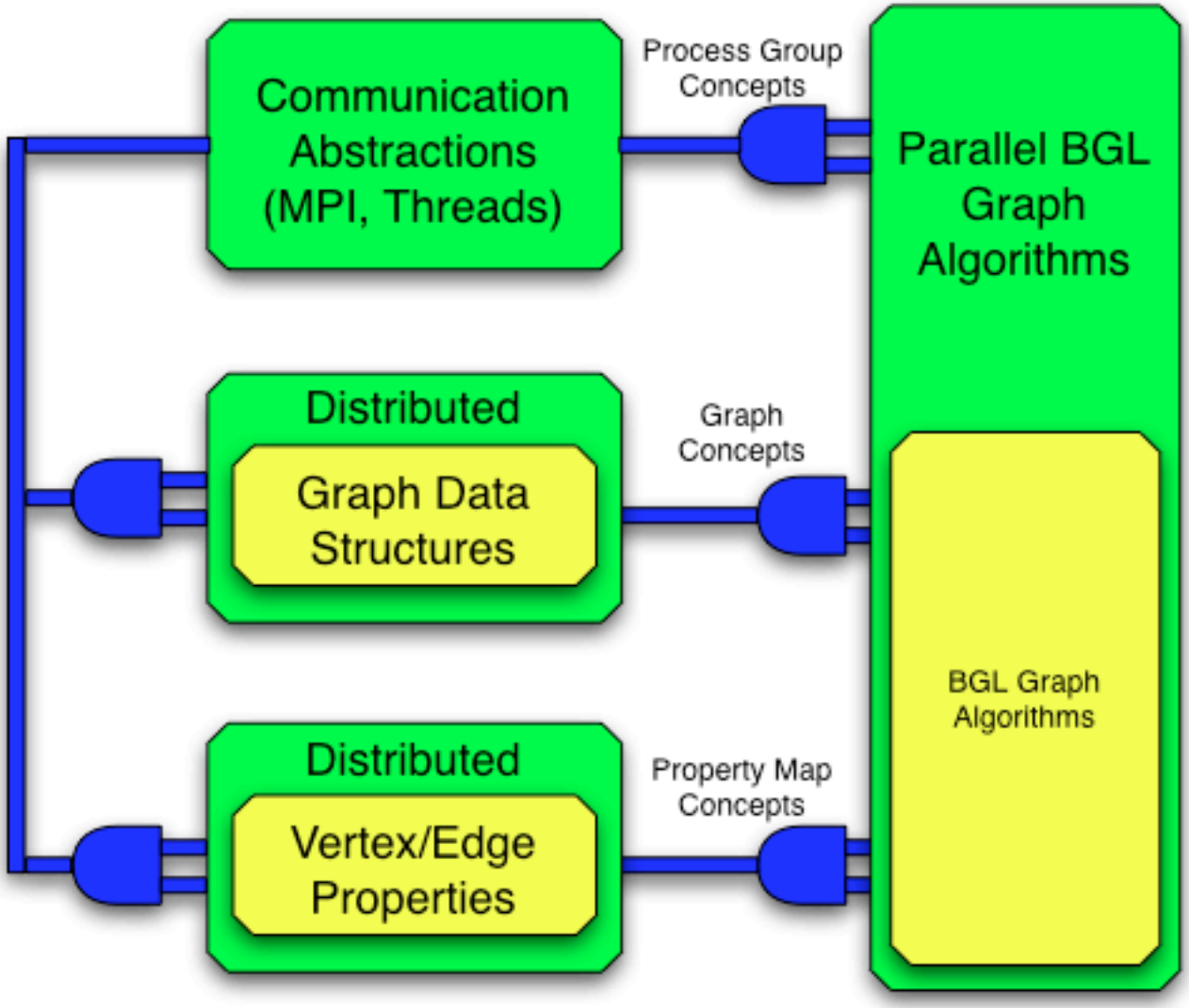
pervasive technology labs
AT INDIANA UNIVERSITY

BGL Architecture



pervasivetechnologylabs
AT INDIANA UNIVERSITY

Parallel BGL Architecture



pervasive technology labs
AT INDIANA UNIVERSITY