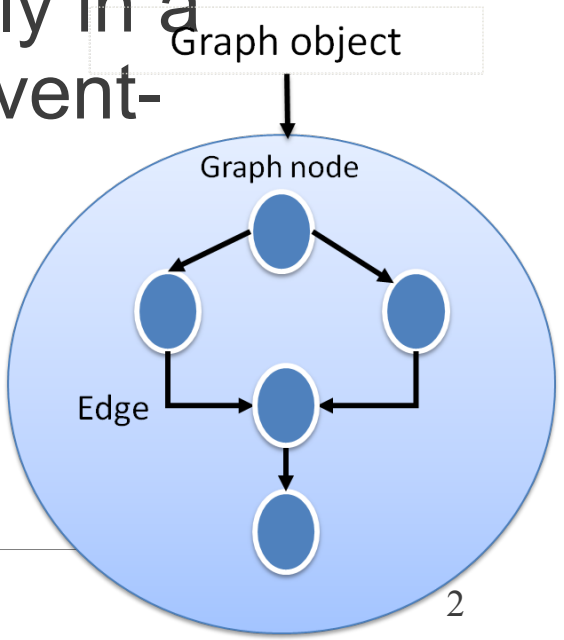
TBB investigation for SuperB

Francesco Giacomini – INFN

Forum on Concurrent Programming Models and Frameworks
2012-01-18

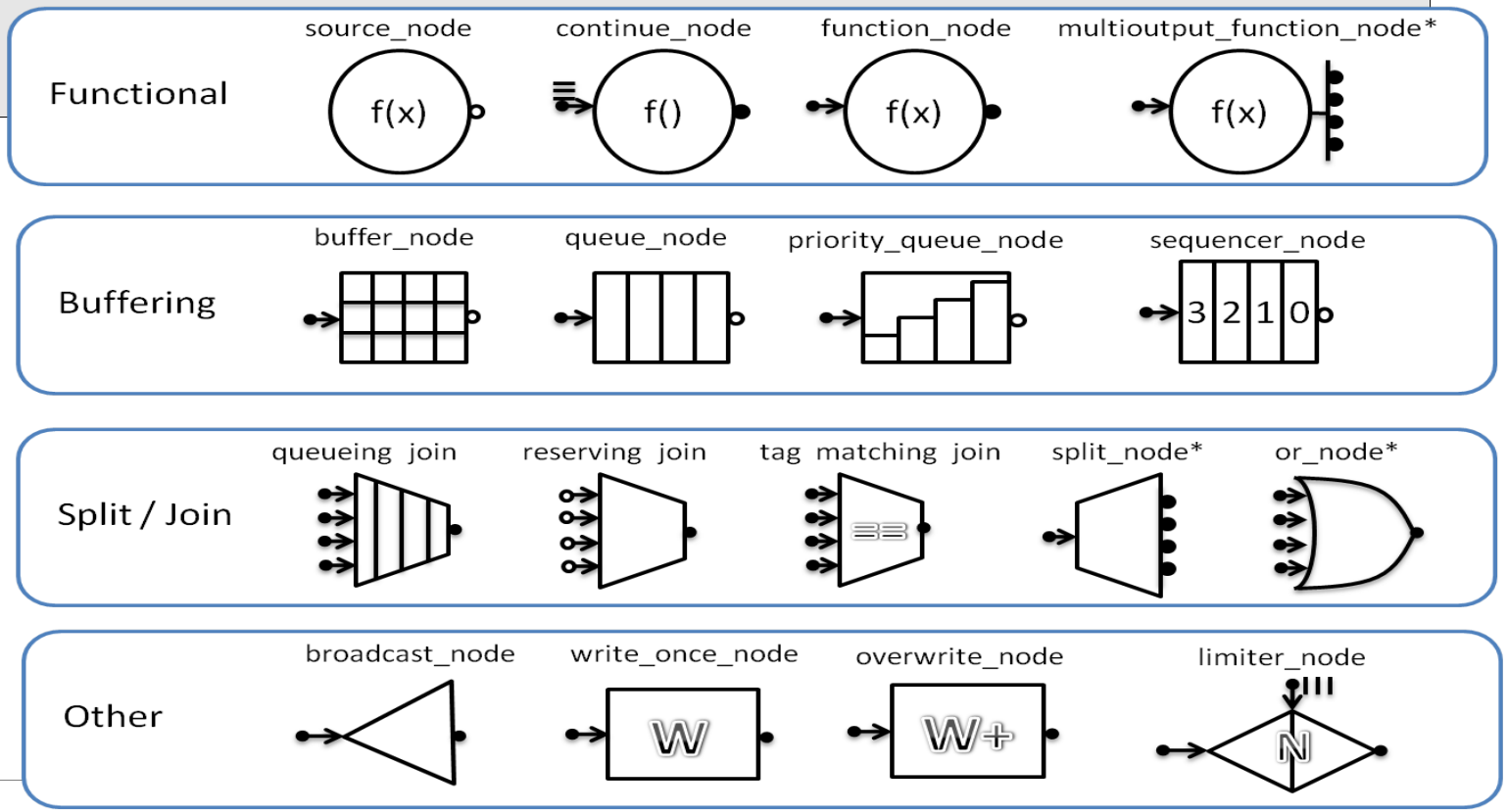
What is TBB?

- http://threadingbuildingblocks.org/
- Intel Threading Building Blocks (TBB) is a library offering a rich approach to expressing parallelism in a C++ program
- It represents a high-level, task-based parallelism that abstracts platform details and threading mechanisms
- For the moment we are interested mainly in a feature called “flow graph”, an API for event-driven/reactive programming models
  - Nodes would be application modules edges would be deps among them



TBB Example

```
tbb::flow::graph g;
tbb::flow::source_node<Event> source(g, GenerateEvent(n_of_events), false);
tbb::flow::function_node<Event, Event> a(g, tbb::flow::unlimited, Body("A"));
tbb::flow::function_node<Event, Event> b(g, tbb::flow::unlimited, Body("B"));
tbb::flow::join_node<std::tuple<Event, Event>, tbb::flow::tag_matching> j(g, tag, tag);
tbb::flow::function_node<std::tuple<Event, Event>> sink(g, tbb::flow::serial, OutputEvent);
make_edge(source, a);
make_edge(source, b);
make_edge(a, std::get<0>(j.inputs()));
make_edge(b, std::get<1>(j.inputs()));
make_edge(j, sink);
source.activate();
g.wait_for_all();
```

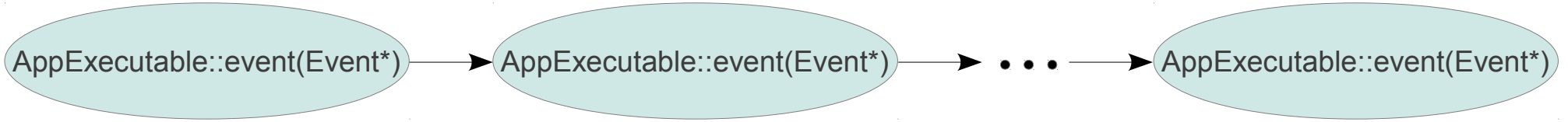


Many node types are available

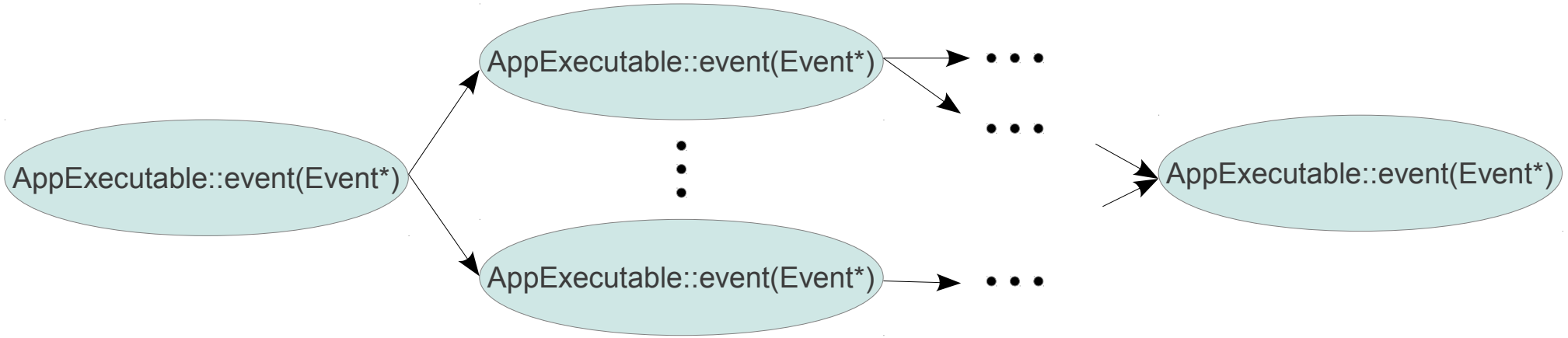
TBB Example in SB Fastsim

(slide shown at the SuperB Collaboration Meeting @LNF last December)

- Modified the framework so that the loop executing the modules in a sequence has been replaced by a graph with the same modules



- Inefficient way of doing the same thing
  - Proof of concept
- But what about the following?



How to express dependencies

- Contrasting guidelines → Challenge
  - Expressivity in the application domain
    - Syntax-friendly to the framework user
  - Efficient mapping to hardware resources available at run-time
    - Leave enough flexibility to the framework developer
      - e.g. replace TBB with libdispatch
- Leverage the compiler as much as possible
  - Profit from C++11 features
  - Do as much as possible at compile time
    - Hashed string literals can be used as template parameters to identify modules and products

How to express dependencies /2

- Basic idea: each module declares which products it needs and which products it provides

```
class M4: public Module<module_four,
                        Requires<product_one>,
                        Provides<product_three, product_four>
                       >
{
  bool operator()(Event& e) const { /* ... */ }
};
```

- *Requires*<> and *Provides*<> provide “safe” *get()* and *put()* of products from/to an event
- Module instantiation causes the population of hidden (to the user) data structures with relationships between modules and their respective products
  - Modules cannot be run in the wrong order
- Leave the door open to more flexibility at run-time
  - Need requirements

Next steps

- Implement a small system integrating the basic idea with TBB
- Collect requirements in the SB community