Introducing

MPL.Graph

Gordon Woodhull

BoostCon 2011

Outline

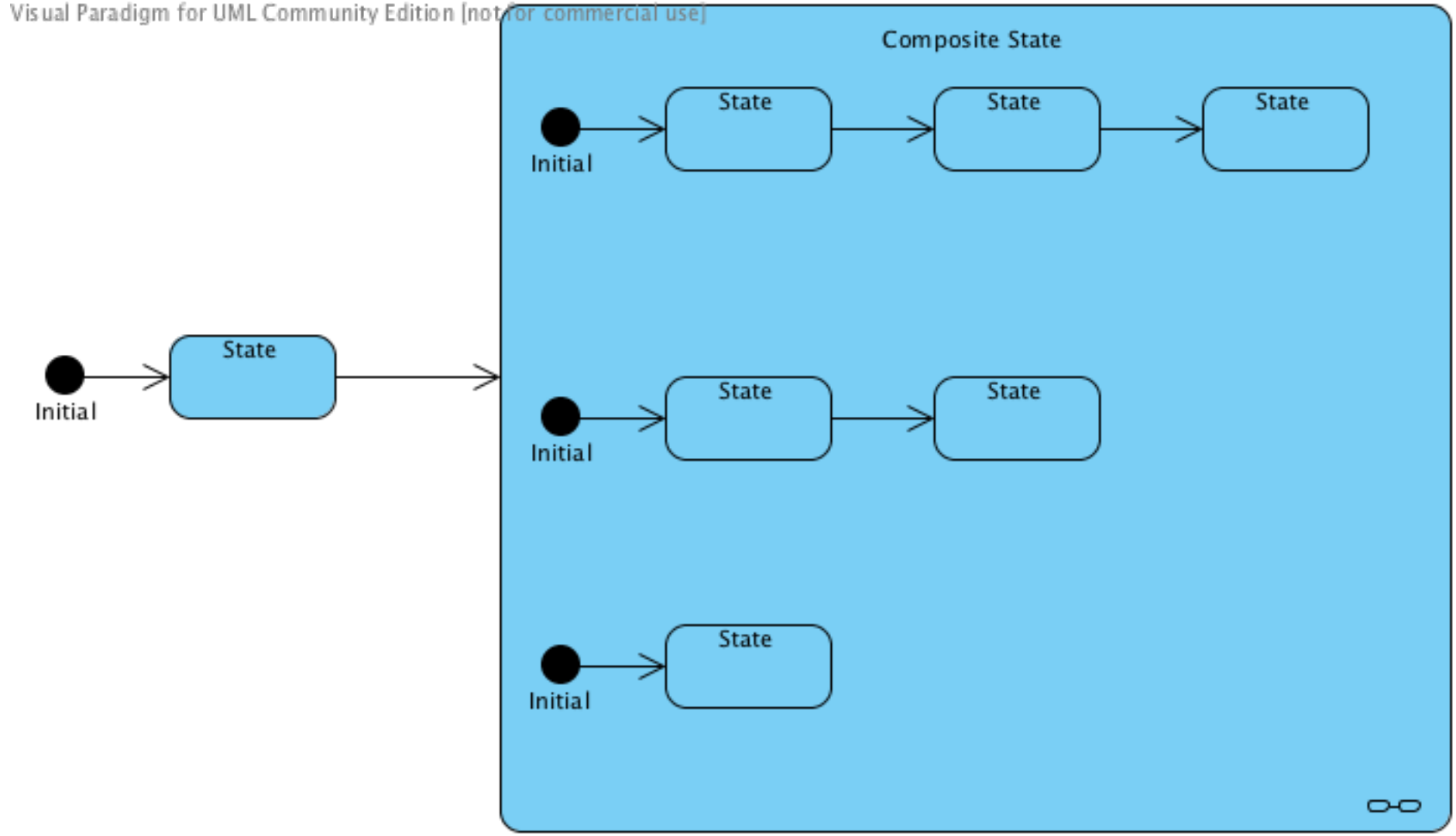
- MPL.Graph in Meta State Machine
- MPL.Graph Concepts & Algorithms
- MPL.Graph versus Meta Graph Library
- (prototype) Fusion.Graph
- Questions & Comments
- Fun Ideas

Meta State Machine

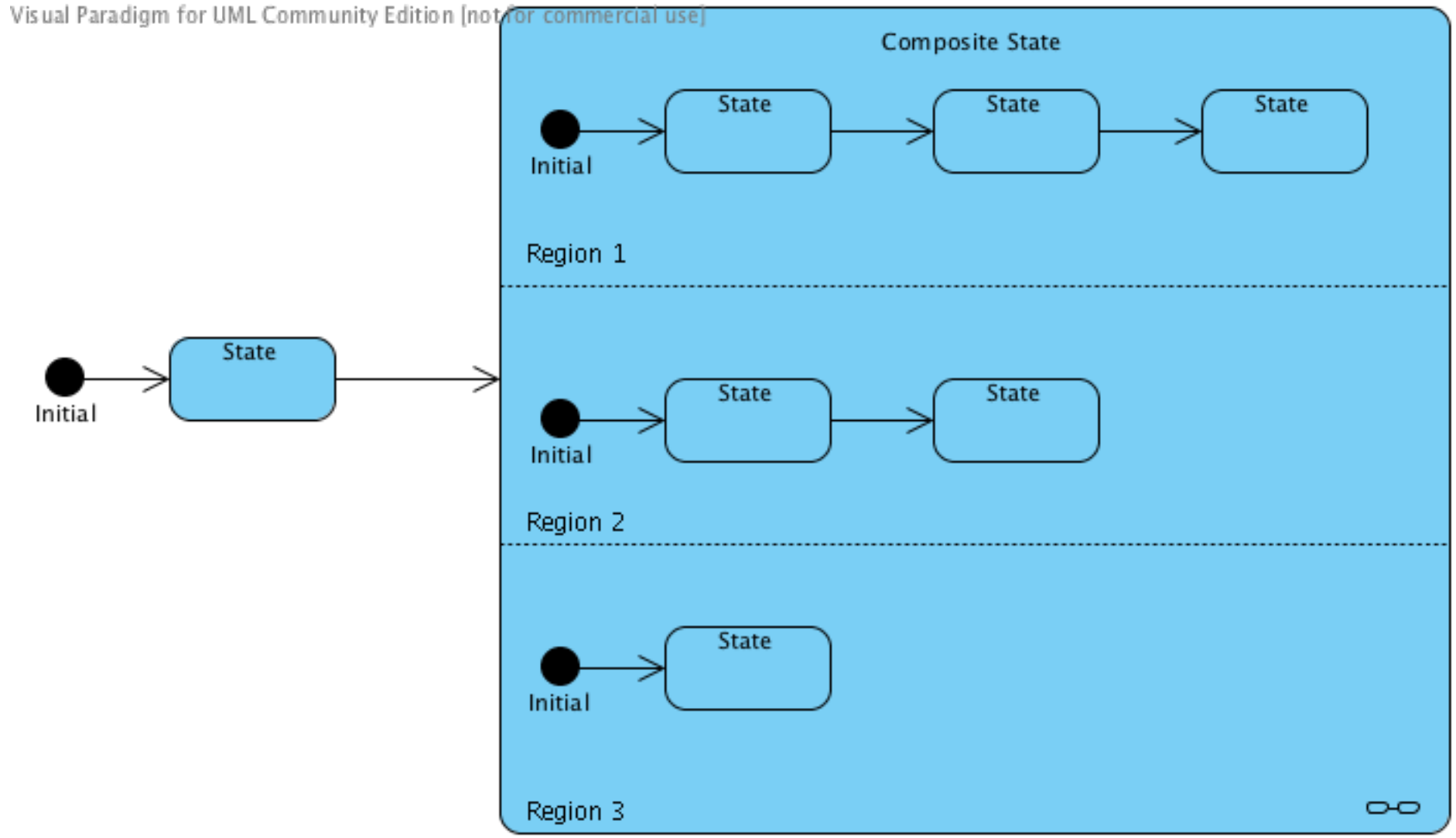
- Christophe Henry's implementation of UML state machines using metaprogramming

```
struct transition_table : mpl::vector<
   //    Start     Event         Next      Action   Guard
   //  +---------+-------------+---------+--------+-------+
  _row < Stopped , play        , Playing                   >,
  _row < Stopped , open_close  , Open                      >,
  _row < Stopped , stop        , Stopped                   >,
   //  +---------+-------------+---------+--------+-------+
  _row < Open    , open_close  , Empty                     >,
   //  +---------+-------------+---------+--------+-------+
  _row < Empty   , open_close  , Open                      >,
  _row < Empty   , cd_detected , Stopped                   >,
  _row < Empty   , cd_detected , Playing                   >,
   //  +---------+-------------+---------+--------+-------+
  _row < Playing , stop        , Stopped                   >,
  _row < Playing , pause       , Paused                    >,
  _row < Playing , open_close  , Open                      >,
   //  +---------+-------------+---------+--------+-------+
  _row < Paused  , end_pause   , Playing                   >,
  _row < Paused  , stop        , Stopped                   >,
  _row < Paused  , open_close  , Open                      >,
  _row < AllOk   , error_found , ErrorMode                 >
  //  +---------+-------------+---------+--------+-------+
  > {};
```

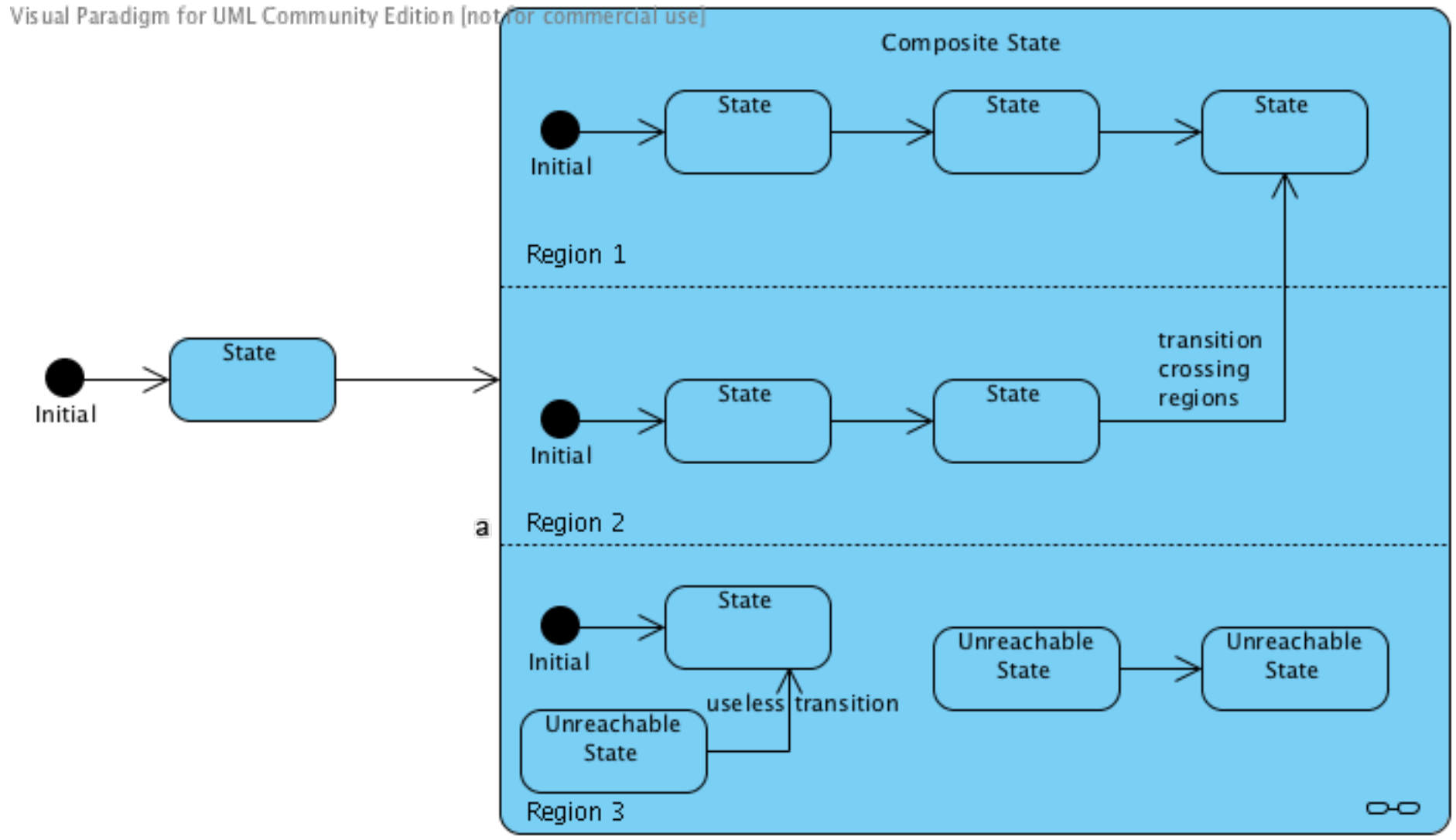
MSM – Orthogonal Regions



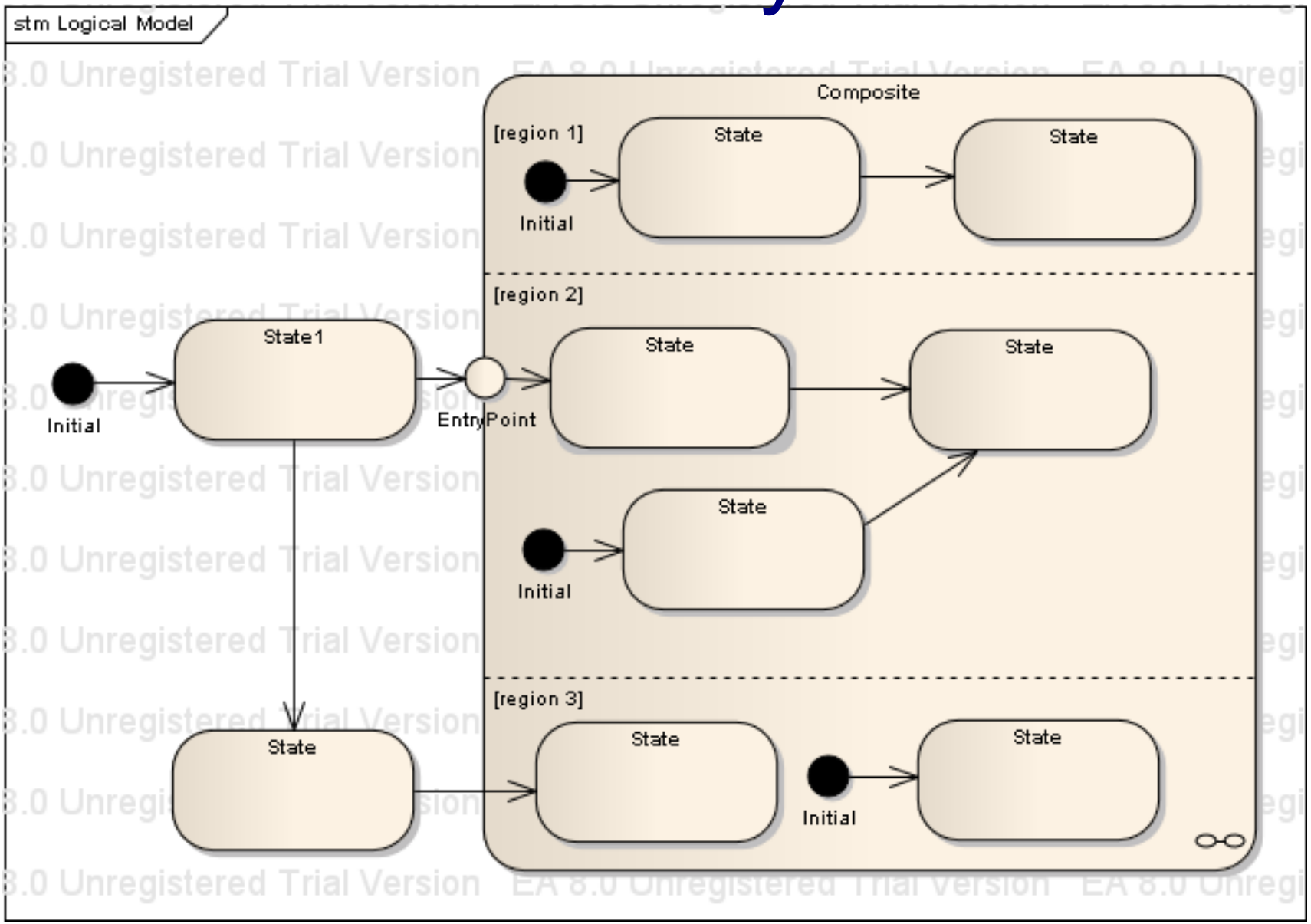
MSM – Orthogonal Regions



MSM – Orthogonality Errors



MSM – Entry Points



MPL.Graph

- Metafunction version of Boost.Graph
- All concepts except MutableGraph
- “Port” from BGL: “s/(/</”
- Lazy concepts: no need to choose implementation

MPL.Graph: BGL-like Concepts

```
// IncidenceGraph
template<typename Edge, typename Graph>
struct source : ...
{};
template<typename Edge, typename Graph>
struct target : ...
{};
template<typename Vertex, typename Graph>
struct out_edges : ...
{};
template<typename Vertex, typename Graph>
struct out_degree : ...
{};
```

MPL.Graph Algorithms

- Depth First Search
- Breadth First Search
- Algos take visitors

MPL.Graph: depth_first_search

```
template<typename Graph, typename VisitorOps, typename VisitorState,
         typename Vertex,
         typename ColorState = create_search_color_map::type >
struct depth_first_search {
    // enter vertex
    typedef typename VisitorOps::template
        discover_vertex<Vertex, Graph, VisitorState>::type
            discovered_state;
    typedef typename search_color_map_ops::template
        set_color<Vertex, search_colors::Gray, ColorState>::type
            discovered_colors;

    // loop over out edges
    typedef typename
        mpl::fold<typename mpl_graph::out_edges<Vertex, Graph>::type,
                  mpl::pair<discovered_state, discovered_colors>,
                  mpl::if_<boost::is_same<
        search_color_map_ops::get_color<mpl_graph::target<mpl::_2, Graph>,
                    mpl::second<mpl::_1> >,
        search_colors::White>,
        // unseen target: recurse
          depth_first_search<Graph, VisitorOps,
               typename VisitorOps::template tree_edge<mpl::_2, Graph,
                     mpl::first<mpl::_1> >,
               mpl_graph::target<mpl::_2, Graph>,
               mpl::second<mpl::_1> >,
```

depth_first_search ctd.

```
 // seen: back or forward edge
  mpl::pair<mpl::if_<boost::is_same<
  typename search_color_map_ops::template
   get_color<mpl_graph::target<mpl::_2, Graph>, mpl::second<mpl::_1 > >,
   search_colors::Gray>,
       typename VisitorOps::template back_edge<mpl::_2, Graph,
mpl::first<mpl::_1> >,
       typename VisitorOps::template forward_or_cross_edge<mpl::_2, Graph,
mpl::first<mpl::_1> > >,  // Black
     mpl::second<mpl::_1> > >
    >::type after_outedges;

    // leave vertex, and done!
    typedef mpl::pair<typename VisitorOps::template
        finish_vertex<Vertex, Graph, typename
                 mpl::first<after_outedges>::type >::type,
                      typename search_color_map_ops::template
                       set_color<Vertex, search_colors::Black, typename
                       mpl::second<after_outedges>::type>::type>
        type;
};
```

Meta Graph Library

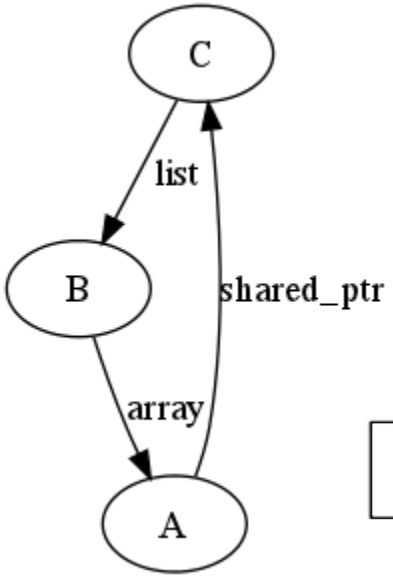
- Franz Alt's graph metaprogramming library
- Iterator-based design
- Better performance
- Strange metaprogramming style
- We will merge the best of both

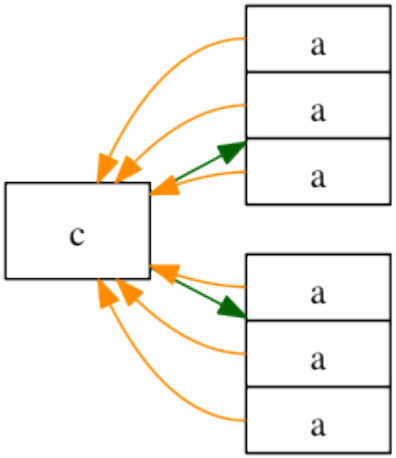
Objects Pointing at Objects

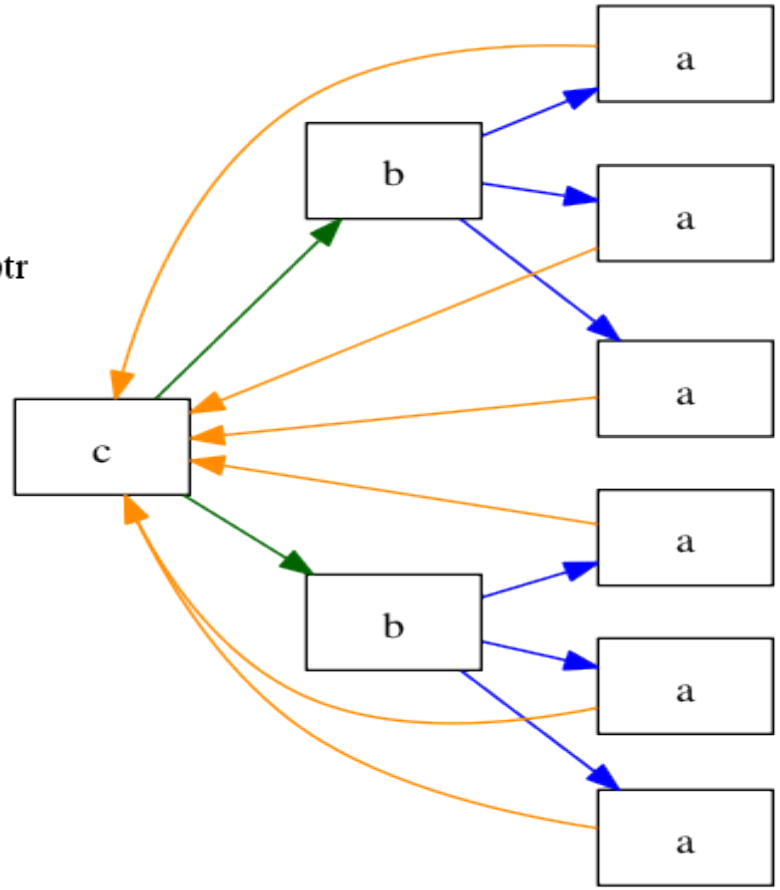
- “Graphlike” or “graphy”
- Type relations known at compile time
- Any system of objects = heterogeneous graph
- What if we want to generate from metadata?
- Containment = Tree

Objects Pointing at Objects

```
class A;
class B;
class C {
    list<B> b_list;
};
class B {
    array<A,5> a_array;
};
class A {
    shared_ptr<C> c_sp;
};
```







Fusion.Graph
(in progress)

- As with BGL & MPL.Graph:
  1. Concepts
  2. One general-purpose implementation
  3. Algorithms

Fusion.Graph

```
template<typename InputGraph>
struct make_fusion_graph {
    struct type {
        template<typename VertexTag> struct vertex_impl;
        template<typename EdgeTag>
        struct edge_impl :
            EdgeTag::template link_container<typename vertex_impl<typename
                    mpl_graph::target<EdgeTag,InputGraph>::type>::type> {};
        template<typename VertexTag>
        struct vertex_impl {
            struct type {
                typedef typename mpl::transform<typename
                        mpl_graph::out_edges<VertexTag,InputGraph>::type,
                                    fusion::pair<mpl::_1,
                                                edge_impl<mpl::_1> >
                                   >::type tag_n_impl_sequence;
                typedef typename mpl::joint_view<tag_n_impl_sequence,typename
                        VertexTag::data_type>::type all_vertex_data;
                typedef typename boost::fusion::result_of::as_map<all_vertex_data>::type
                                data_type;
                data_type data;
        }; }; };
```

MPL.Graph Status

- Performance: measure, improve
- More Algorithms
  - Topological Sort
  - Connected Components
  - Dijkstra's Shortest Paths?
- Another Application or Two
- Review

msm/mpl_graph

dynagraph.org/mpl_graph/

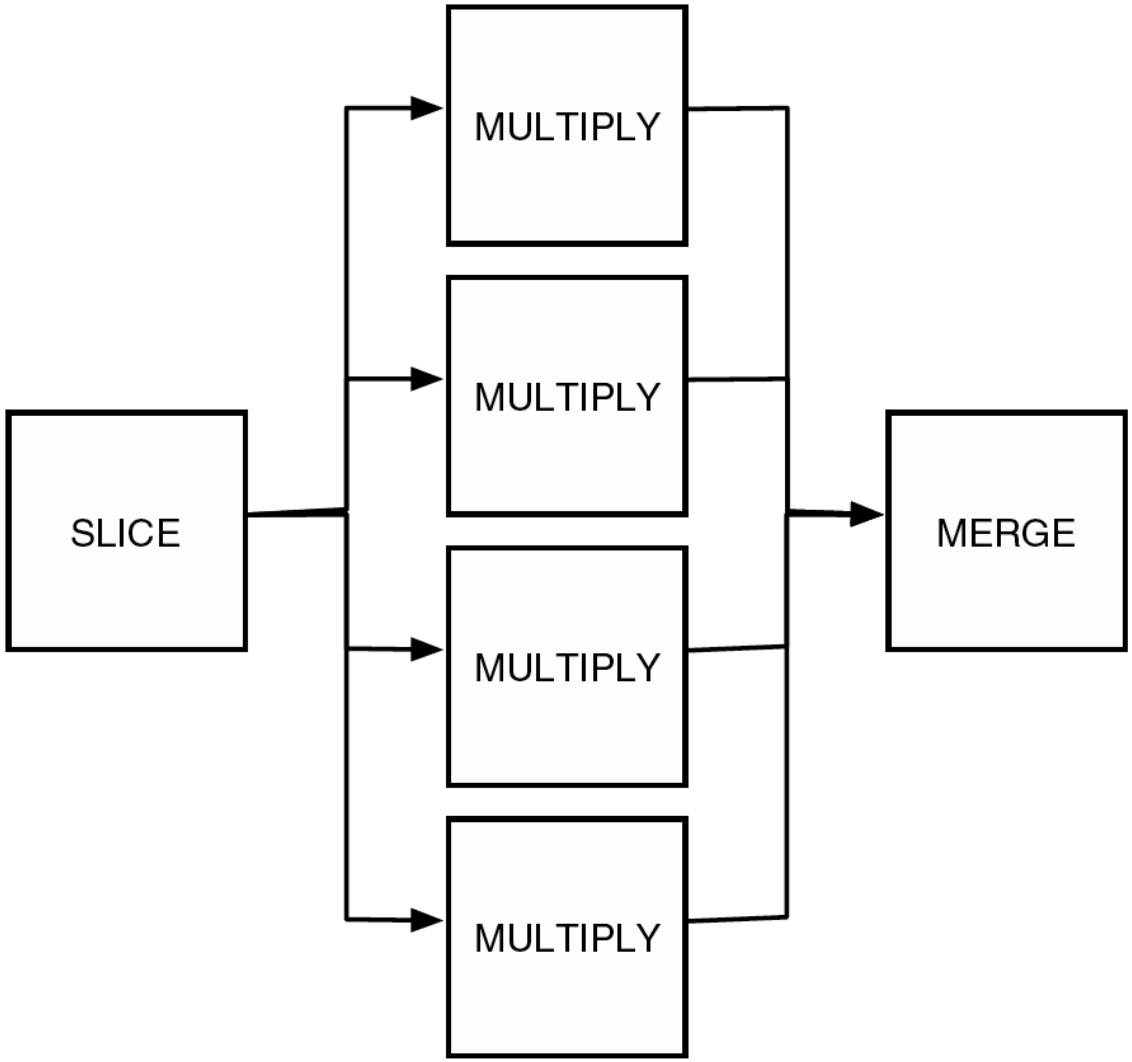
Questions? Comments?

And onward!

WARNING: Everything after this point in the presentation is speculative. I think it's all possible! I'm working on some things & hoping others will work on other parts.

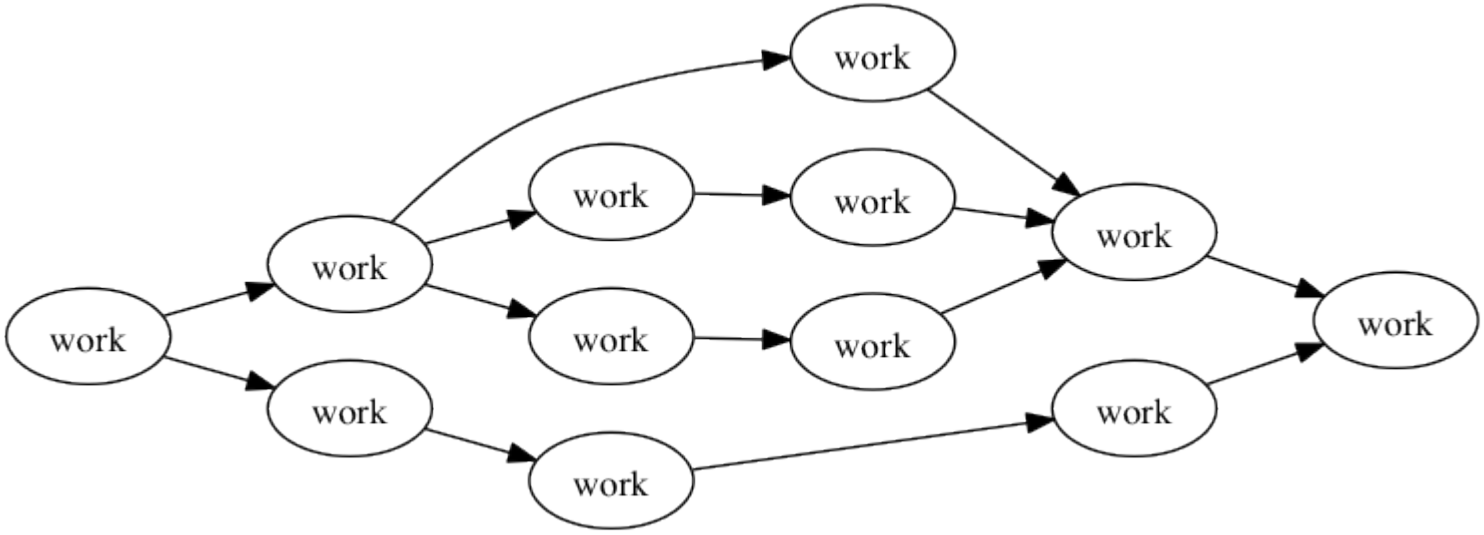
Quaff

```
// Skeleton definition
typedef scm<slice, worker< repeat<4,mul> >, merge> app;
```



Quaff

- Go beyond Series-Parallel?



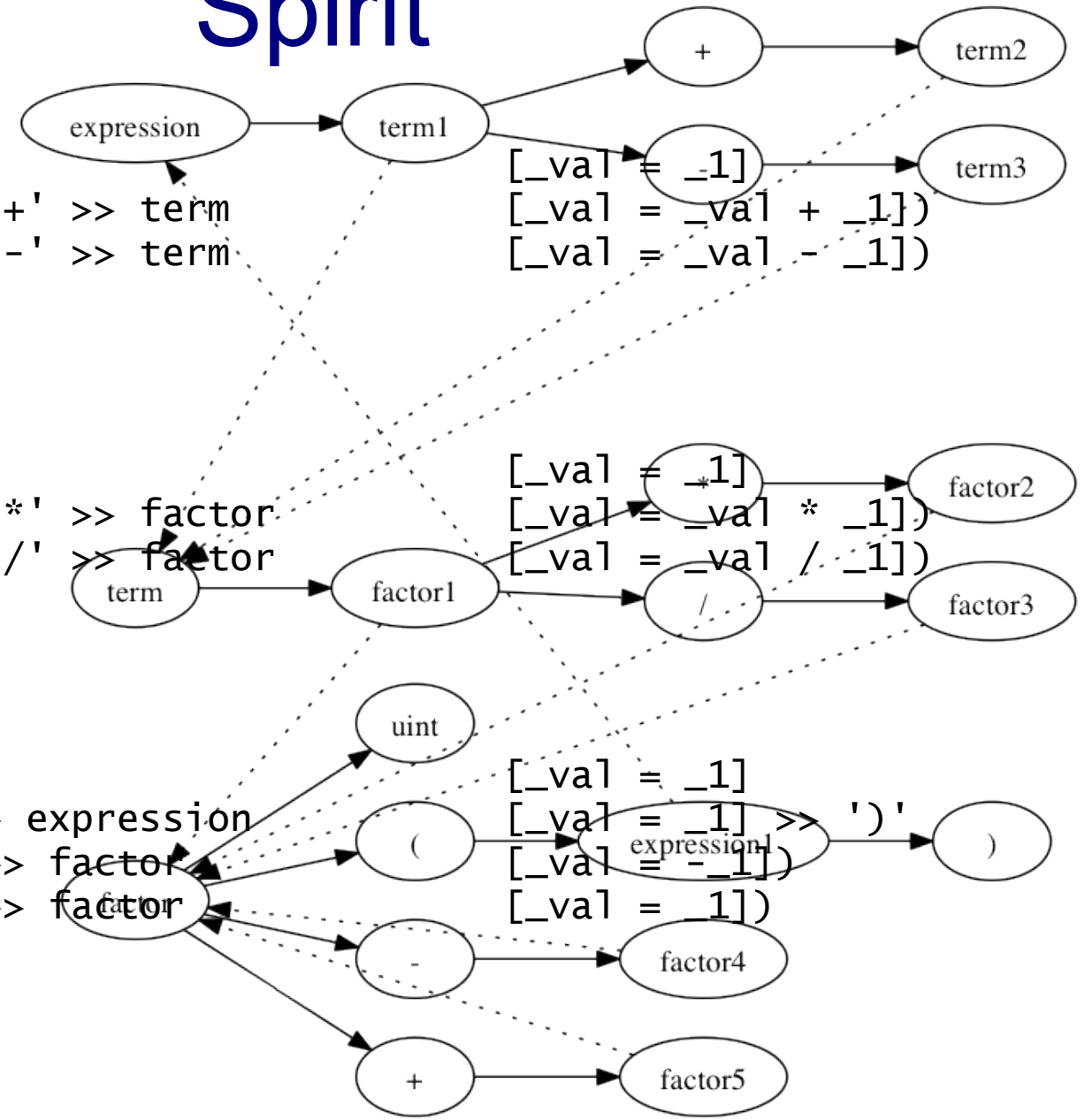
- Also: optimize scheduling?

Spirit

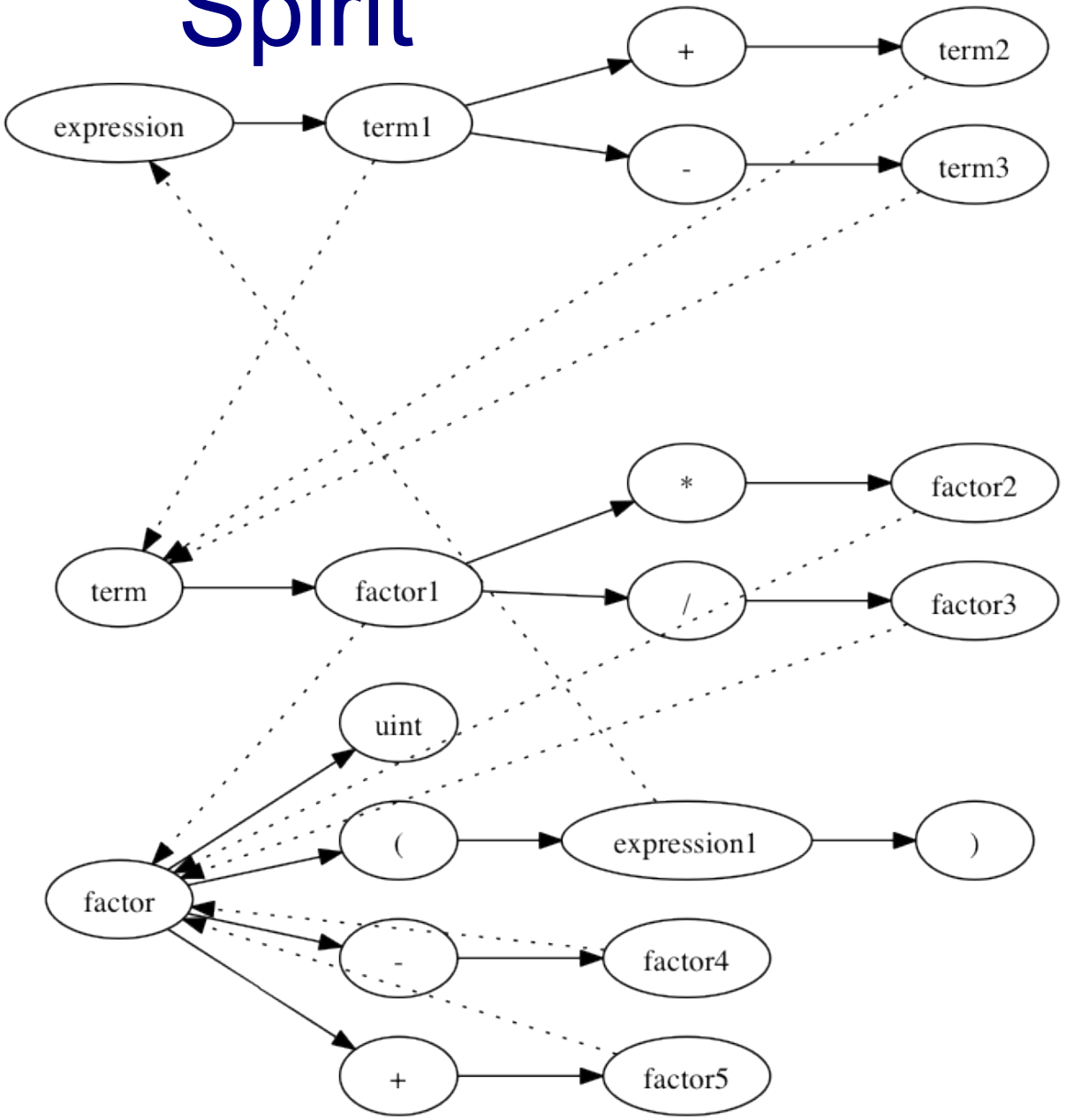
```
expression =
    term                                [_val = _1]
    >> *(   ('+' >> term                [_val = _val + _1])
        |   ('-' >> term                [_val = _val - _1])
        )
    ;

term =
    factor                              [_val = _1]
    >> *(   ('*' >> factor              [_val = _val * _1])
        |   ('/' >> factor              [_val = _val / _1])
        )
    ;

factor =
    uint_                               [_val = _1]
    |   '(' >> expression               [_val = _1] >> ')'
    |   ('-' >> factor                  [_val = -_1])
    |   ('+' >> factor                  [_val = _1])
    ;
```

Spirit

```
expression =
    term
    >> *(   ('+' >> term                [_val = _val + _1])
        |   ('-' >> term                [_val = _val - _1])
        )
    ;

term =
    factor                              [_val = _1]
    >> *(   ('*' >> factor              [_val = _val * _1])
        |   ('/' >> factor              [_val = _val / _1])
        )
    ;

factor =
    uint_                               [_val = _1]
    |   '(' >> expression               [_val = _1] >> ')'
    |   ('-' >> factor                  [_val = -_1])
    |   ('+' >> factor                  [_val = _1])
    ;
```



Spirit


expression
term1
+
-
term2
term3
term
factor1
*
/
factor2
factor3
factor
uint
(
expression1
)
-
+
factor4
factor5

Spirit

- Analyze grammars?
- Abstract Syntax Tree nodes are generated from (simplified) grammar graph
- Thus nodes are a Fusion Graph already, without Fusion.Graph

An EDSL for Graphs

- Base it on GraphViz *dot* syntax
- For CT graph, need unique terminal types

Something like:

```
g = (a >= b [_attr = value],
        b >= c >= d);
```

- General problem: turn expression tree into graph by finding common references

Call Graphs

- Can generate more complex types forward
- Branches represented as.. branches
- Configuration

Schemas

- XML
- DB
- C++?

Memory Management

- Once again, type graph describes object graph
- Find cycles in reference-counted pointers automatically
- Or something more sophisticated?

History

- 1997-2003 AT&T Dynagraph LGraph
- 2006 Johns Hopkins Dyna
- 2007 1st BoostCon
- 2008 Started Metagraph library
- 2009 MSM and Quaff @ BoostCon
- 2010 Split MPL.Graph and put in MSM

The Metagraph

- Graphs of graphs, graphs at levels of detail, nested graphs, subgraphs, n-dimensional graphs
- Build on Fusion.Graph
- Overlaid patterns refine/restrict
- Intrusive data structures - multi-belonging