AME Patterns

- Generic library for modeling, recognition, and synthesis of sequential patterns
  - Gestures, speech, text, numerical patterns
- Acknowledgements
  - Stjepan Rajko, Dhi Aurrahman, David Burri, Sashikanth Damaraju, Bo Peng, Arif Setiawan
  - School of Arts, Media and Engineering
  - This material is based upon work supported by the National Science Foundation under Grant No. 0403428. and Grant No. 0504647

Why BoostCon?

- Uses
  - Boost Graph Library
  - Boost.Range (+RangeEx)
  - Boost.Serialization
  - Boost.Fusion
  - Boost.Math
  - Boost.Random
  - Boost.Smart Pointers
  - Boost.Assignment
  - Boost.Spirit
  - Eigen (Boost.LA?)

Why BoostCon?

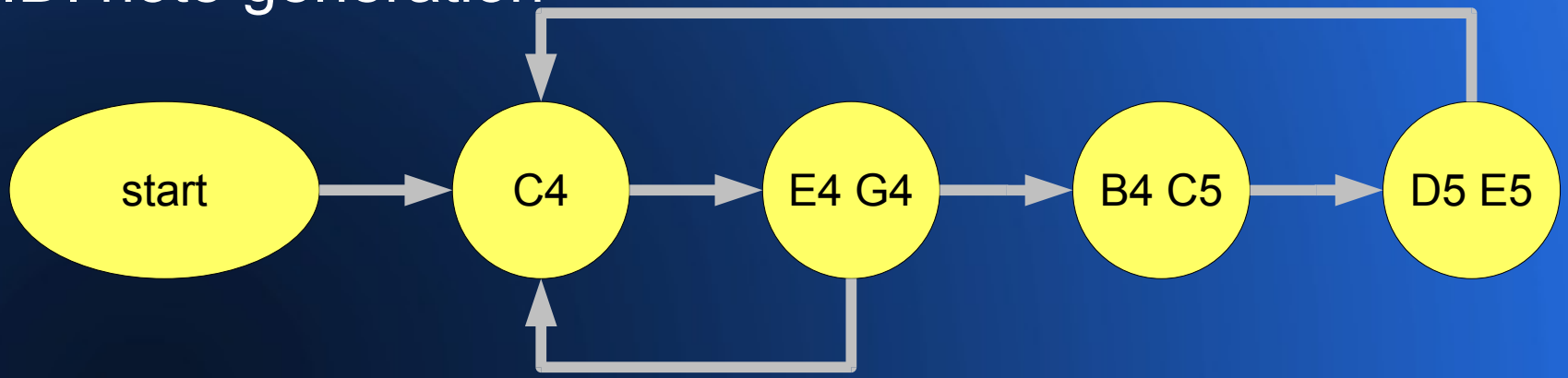
- Generic, concept-based design
- Boost-like build&test setup
  - Boost.Build
  - Boost.Test
- Multiple API levels
  - tradeoffs of programming complexity / flexibility
  - largely inspired by Bjarne Stroustrup's keynote @ BoostCon '08

Current Status

- Ain't done yet
- Has been successfully used for
  - Gesture Recognition
    - Marker-based Motion Capture
    - Video-based Motion Capture
    - Accelerometer data
    - Tangible Objects
    - Mouse / Touch / Multi-Touch data
  - Real-Time Movement Comparison
- Examples of other recognition / comparison / synthesis

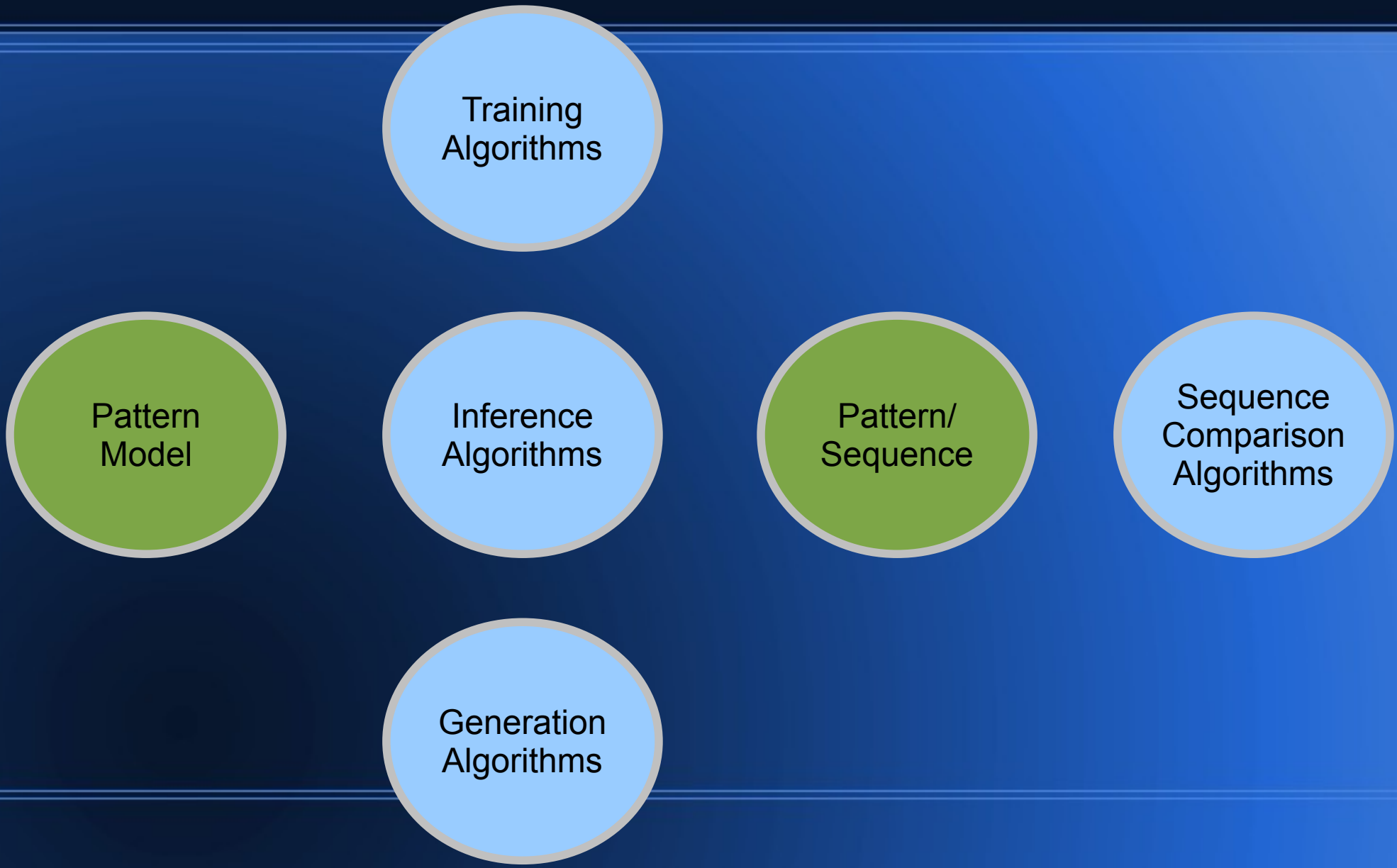
Motivating Examples

- Speech Recognition
- Mouse Gesture Classification
- Mouse Gesture On-Line Recognition
- MIDI note generation

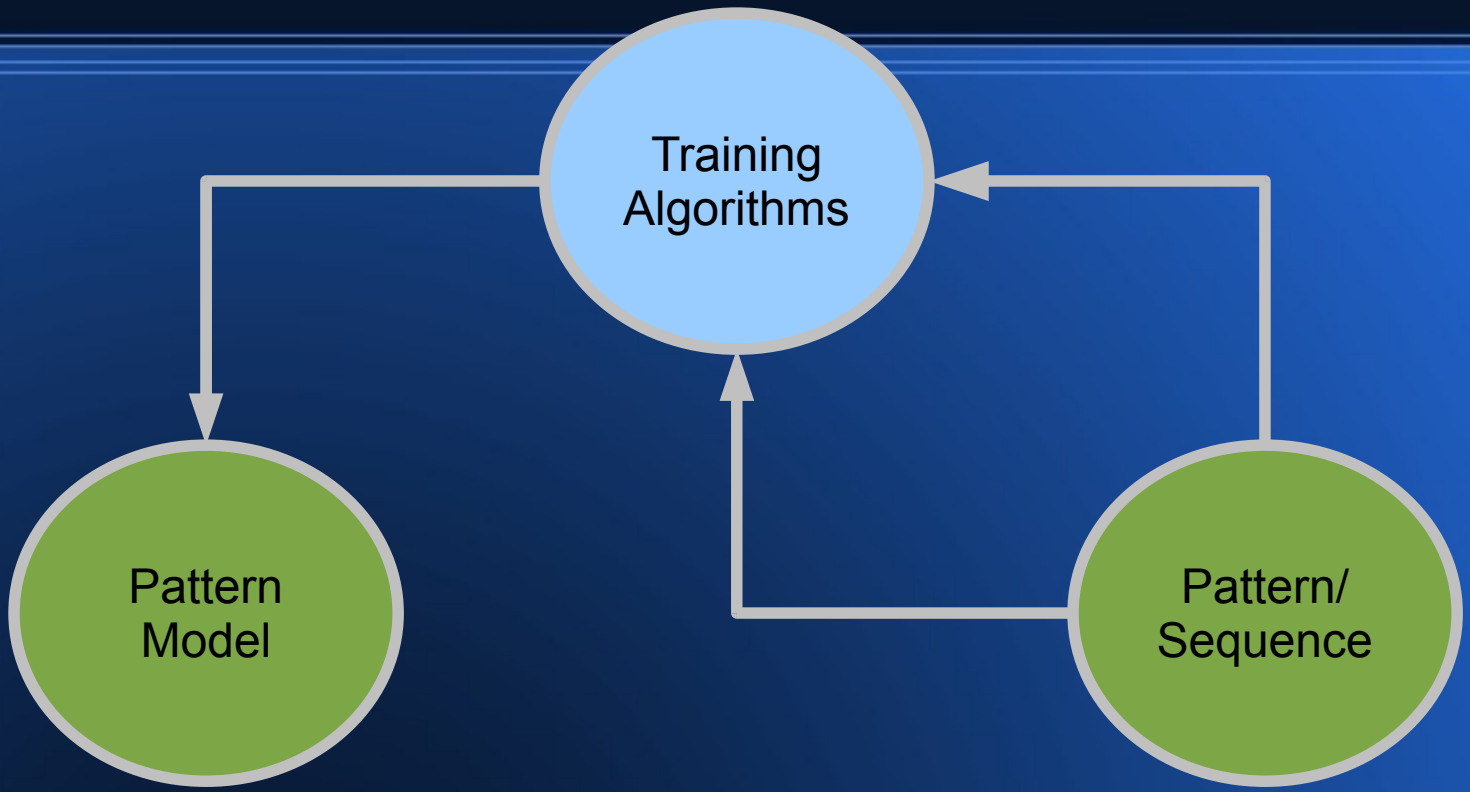


- Dynamic Time Warping

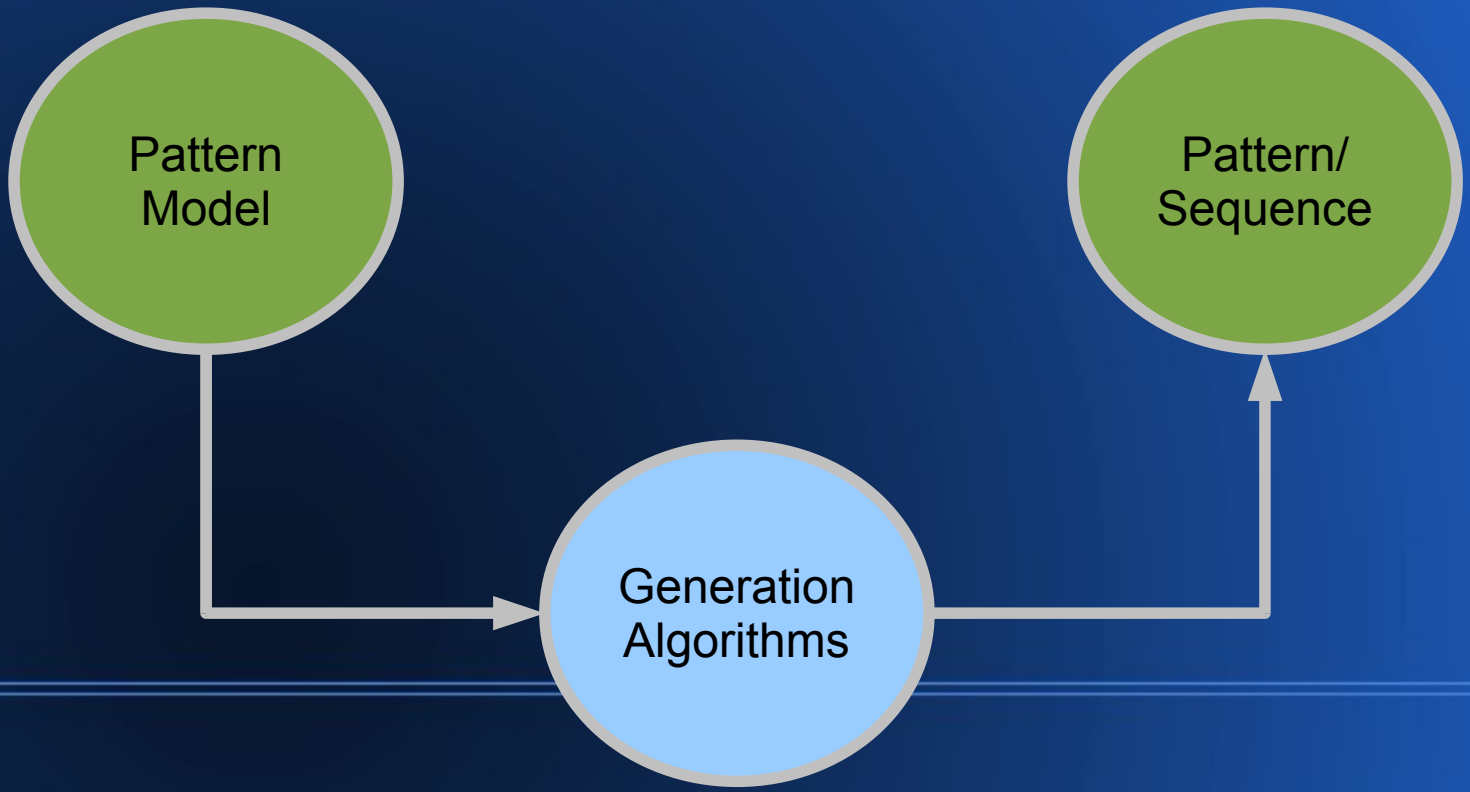
Library Design & Functionality



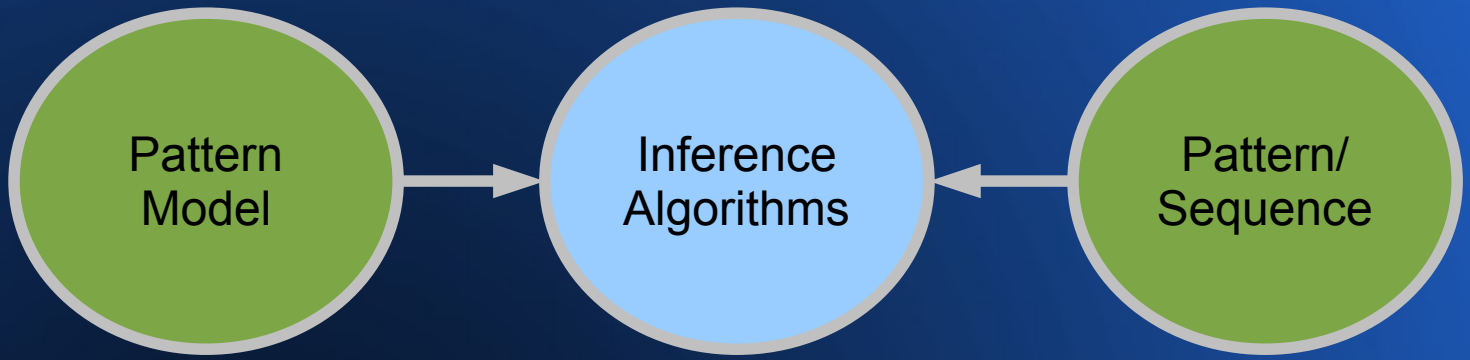
Library Design & Functionality



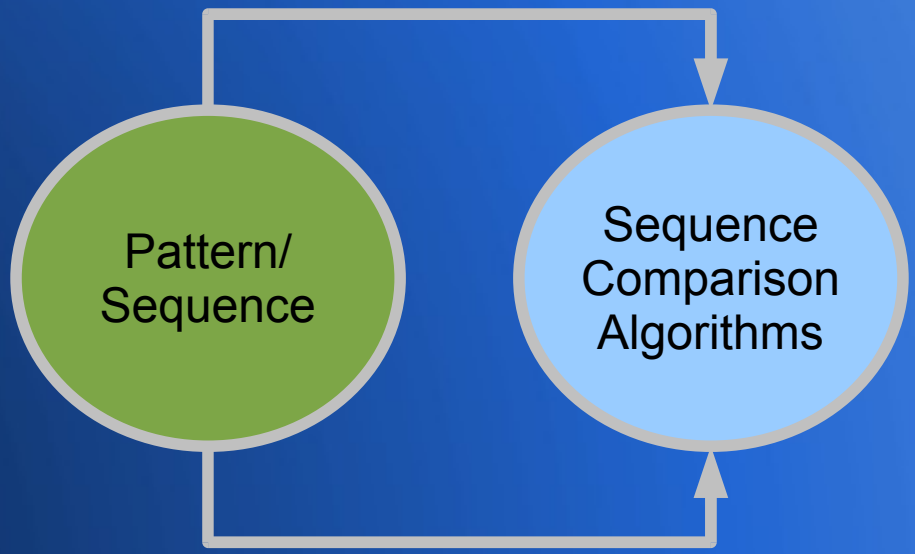
Library Design & Functionality



Library Design & Functionality



Library Design & Functionality

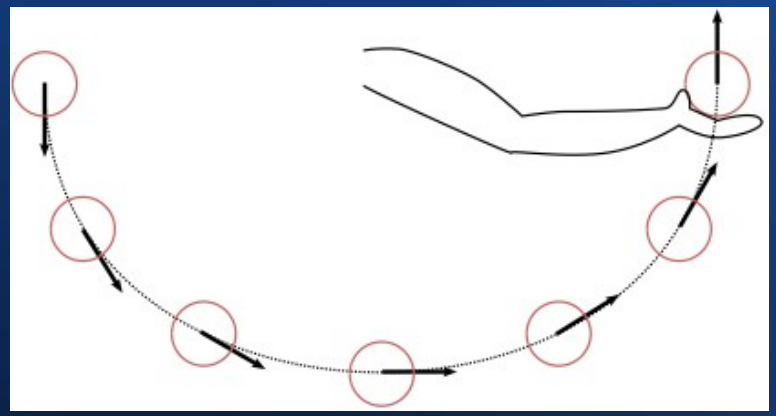


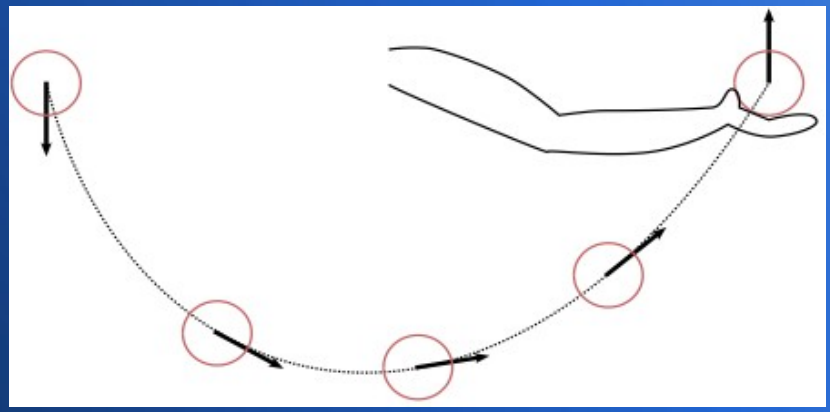
API Levels

- Object-Oriented precompiled library
  - ofxPatterns
  - Addon for openFrameworks
  - Supports common tasks for specific domains
  - Low barrier
- Task-oriented class templates
  - Eases common tasks while keeping some flexibility
- Generic algorithms / data structures
  - The most flexibility

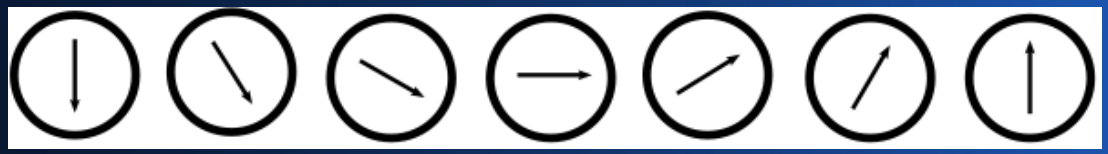
State-based Modeling of Patterns

- A *pattern* is perceived as a sequence of *observations*



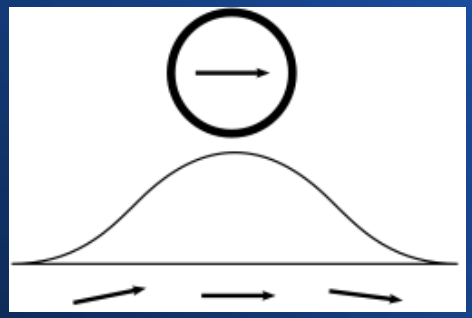


- A model such as HMM captures the behavior of the pattern in *states* – each state models a part of the pattern

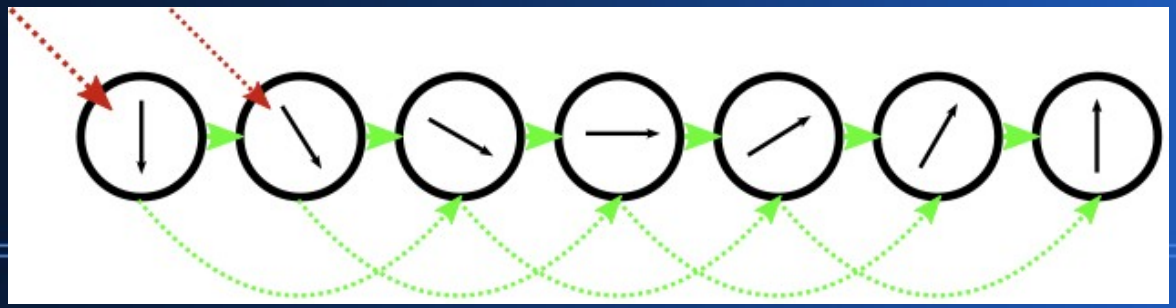


Two Main Aspects of Model

- How does each state relate to observations?
  (observation probability distribution)



- How do we transition through states?
  (initial state, transition probability distribution)

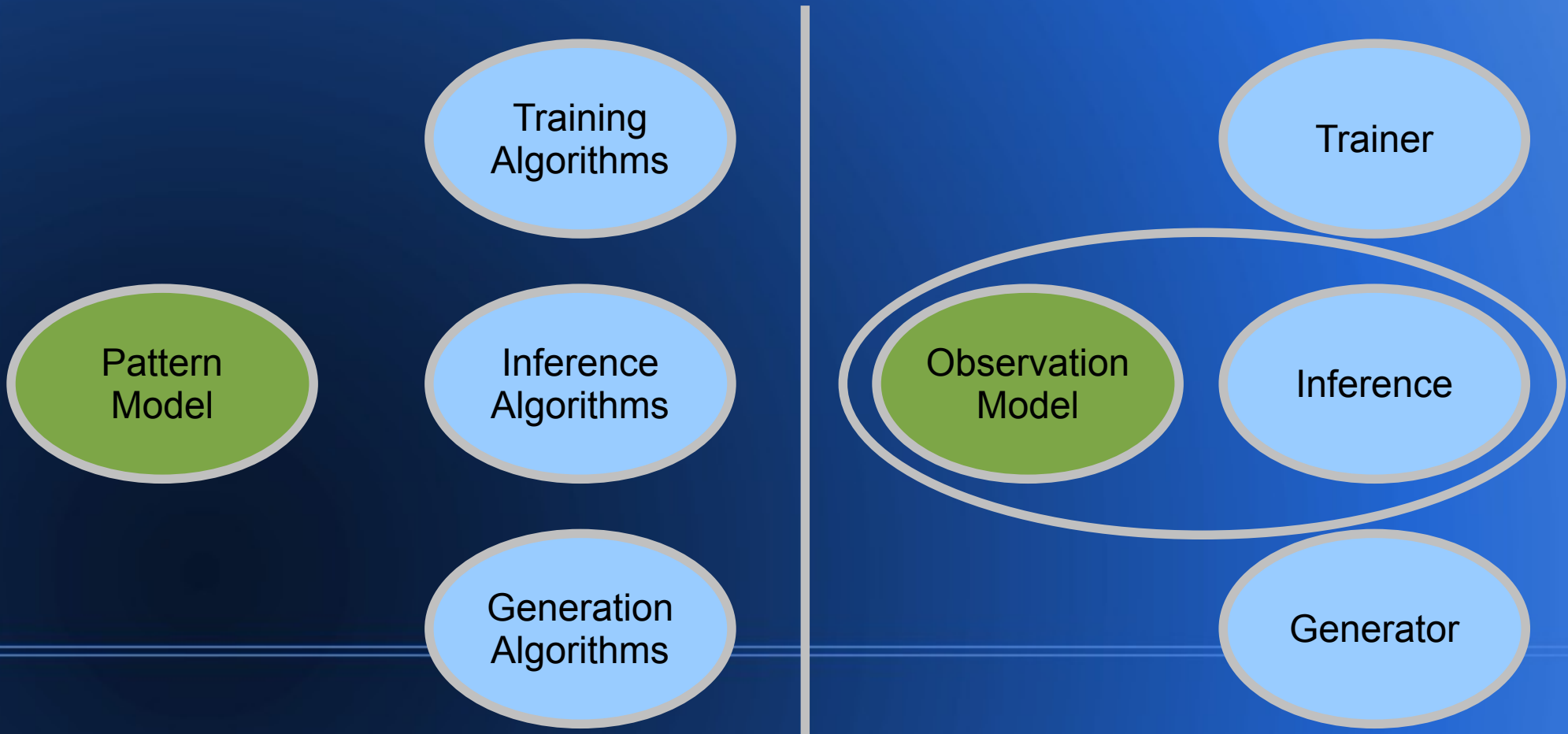


Pattern Model

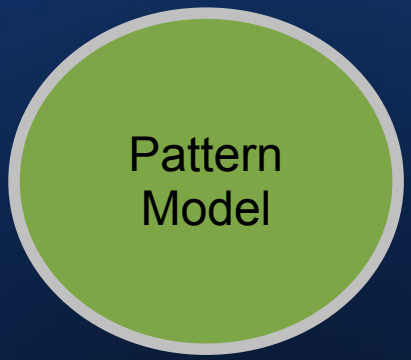
- Augmented Hidden Markov Model (AHMM)
  - States can be emitting or non-emitting
  - Begin and End states
  - Equivalence Classes of Transition Probabilities and Observation Distribution Probabilities (“parameter tying”)

Tutorial

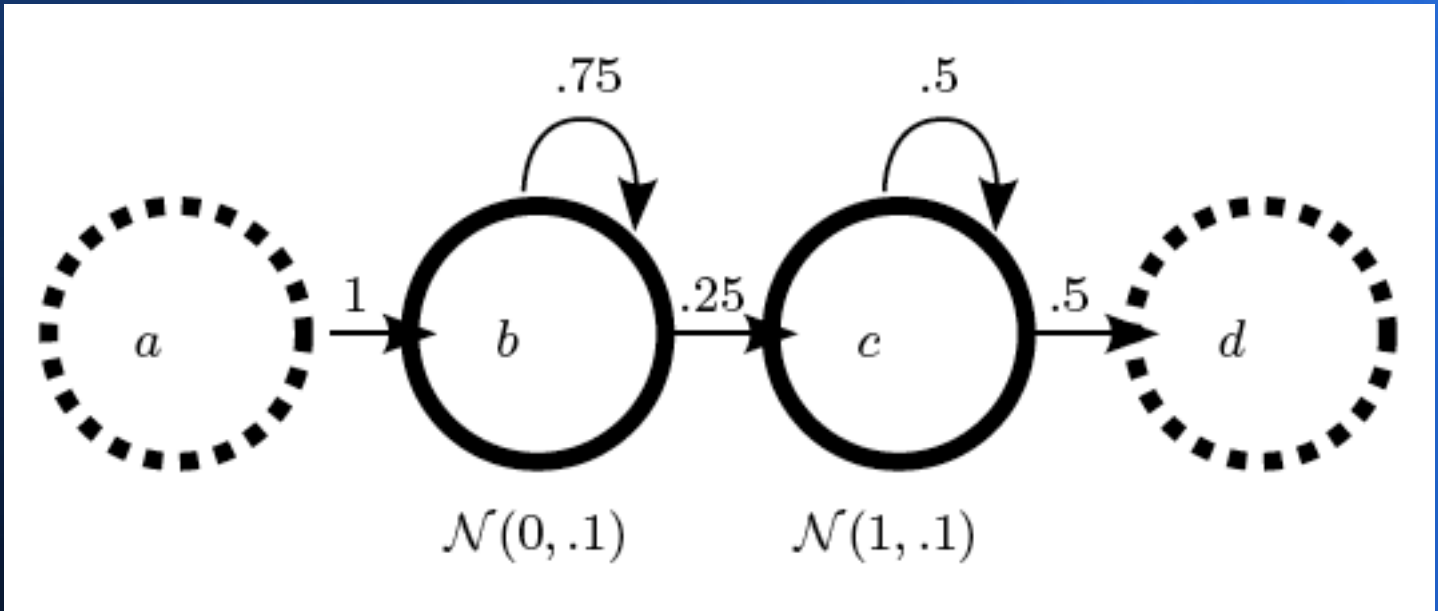
```
namespace patterns = ame::patterns;
namespace observations = ame::observations;
```



Pattern Model Construction



Manual Pattern Model Construction



Manual Pattern Model Construction

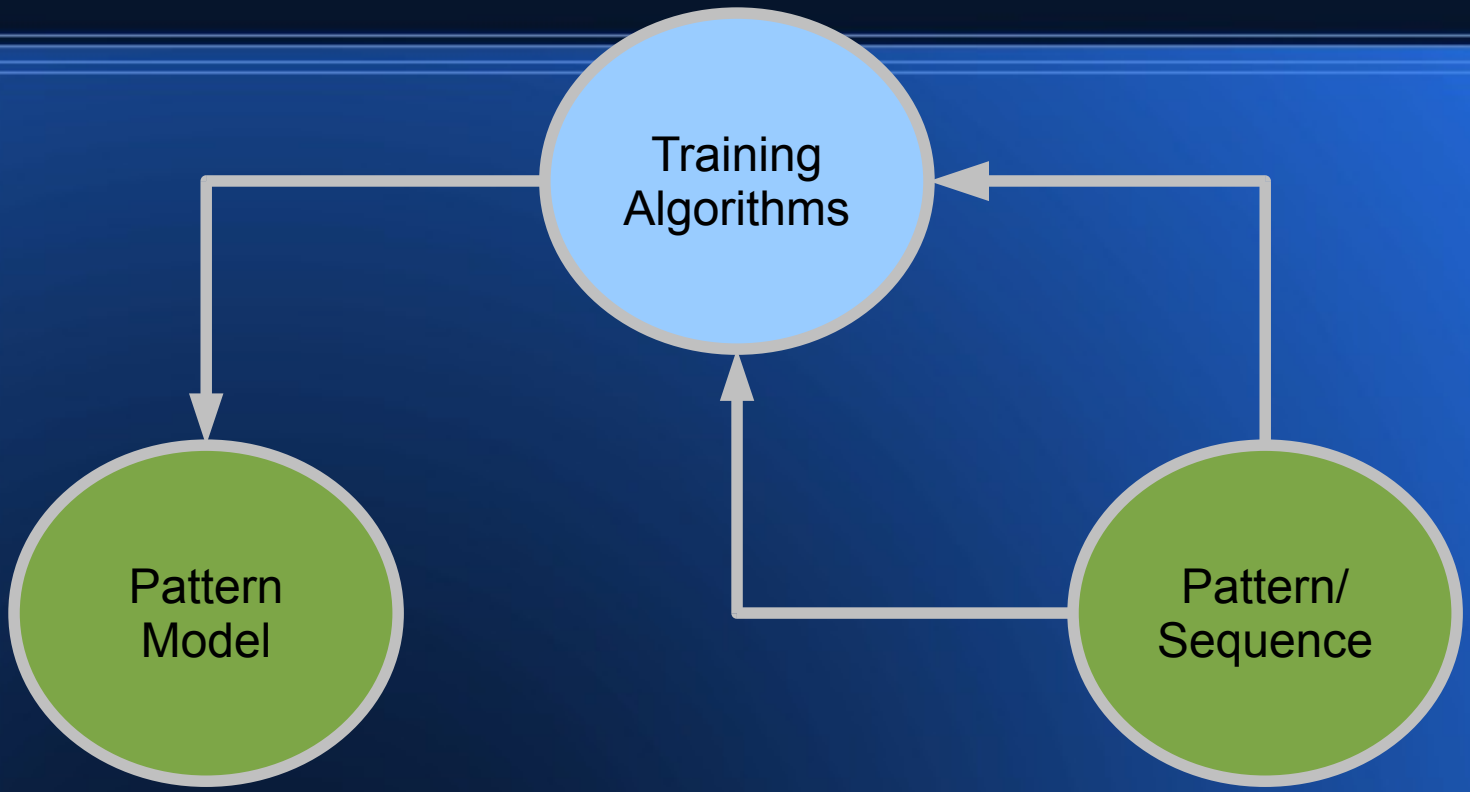
```
typedef patterns::model::ahmm<observations::normal>
   ahmm_type;

ahmm_type ahmm;
ahmm_type::vertex_descriptor a, b, c, d;

a = add_vertex(ahmm);
b = add_vertex(observations::normal(0.0, 0.1), ahmm);
c = add_vertex(observations::normal(1.0, 0.1), ahmm);
d = add_vertex(ahmm);

add_edge(a, b, ahmm);
add_edge(b, b, 0.75, ahmm);
add_edge(b, c, 0.25, ahmm);
add_edge(c, c, 0.5, ahmm);
add_edge(c, d, 0.5, ahmm);

ahmm.make_begin_vertex(a);
ahmm.make_end_vertex(d);
```

Observation Probability Distributions *(Tangent)*

```
typedef patterns::model::ahmm<observations::normal>
   ahmm_type;
```

- Normal (Gaussian)
- Multivariate Normal
- Discrete
- Static Vector
- Dynamic Vector
- Mixture (Homogenous)

Pattern Model Training



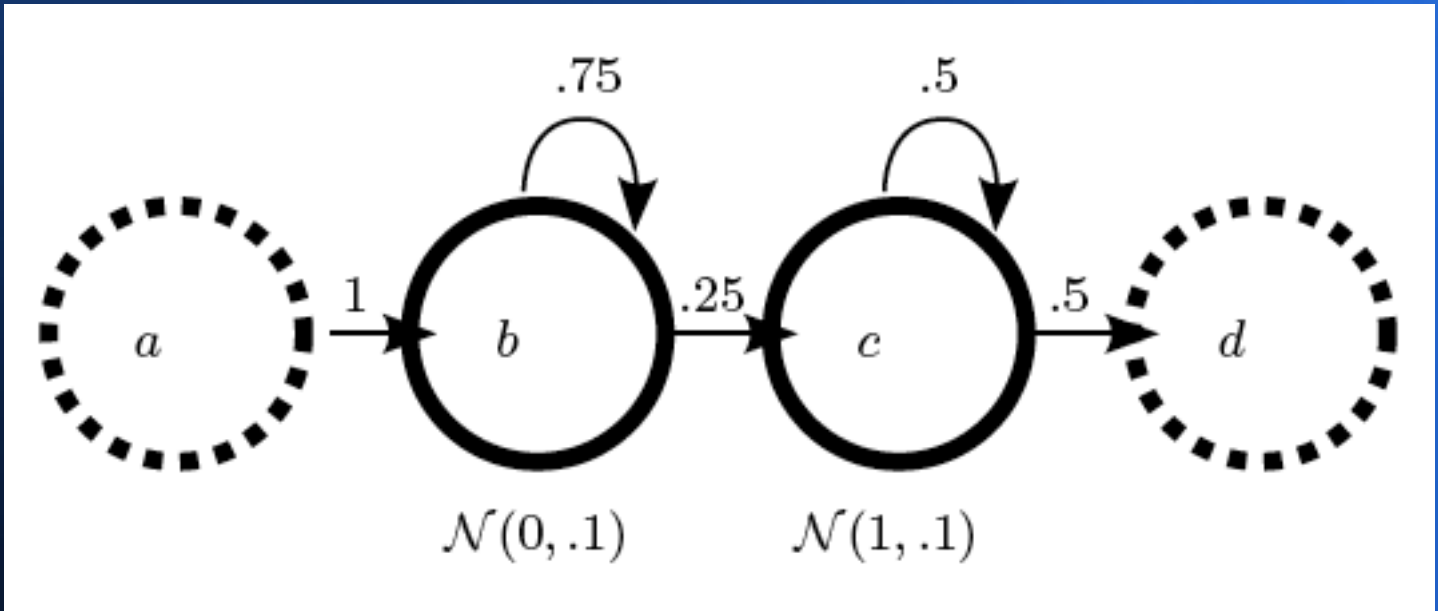
Pattern Model Training

- Given a number of examples of pattern (each a sequence)
- Initialize Pattern Model (somehow)
- While some condition
  - Run inference on Pattern Model & each example
  - Use inference results to improve Pattern Model

Pattern Model Training

```
std::vector<std::vector<double> > examples(2);
using namespace boost::assign;
examples[0] += 0, 0.1, -0.1, 1.1, 0.9, 1.0;
examples[1] += 0.1, -0.1, 0.1, -0.1, 0, 1.0;

typedef patterns::model::chain_hmm<observations::normal>
    ahmm_type;

ahmm_type ahmm(2);
patterns::training::naive_alignment(ahmm, examples);

typedef patterns::inference::best_match<ahmm_type,
        ame::selectors::vector>
    inference_type;

patterns::training::best_match_maximization<inference_type>
   (ahmm, examples, 1, 0.0);
```

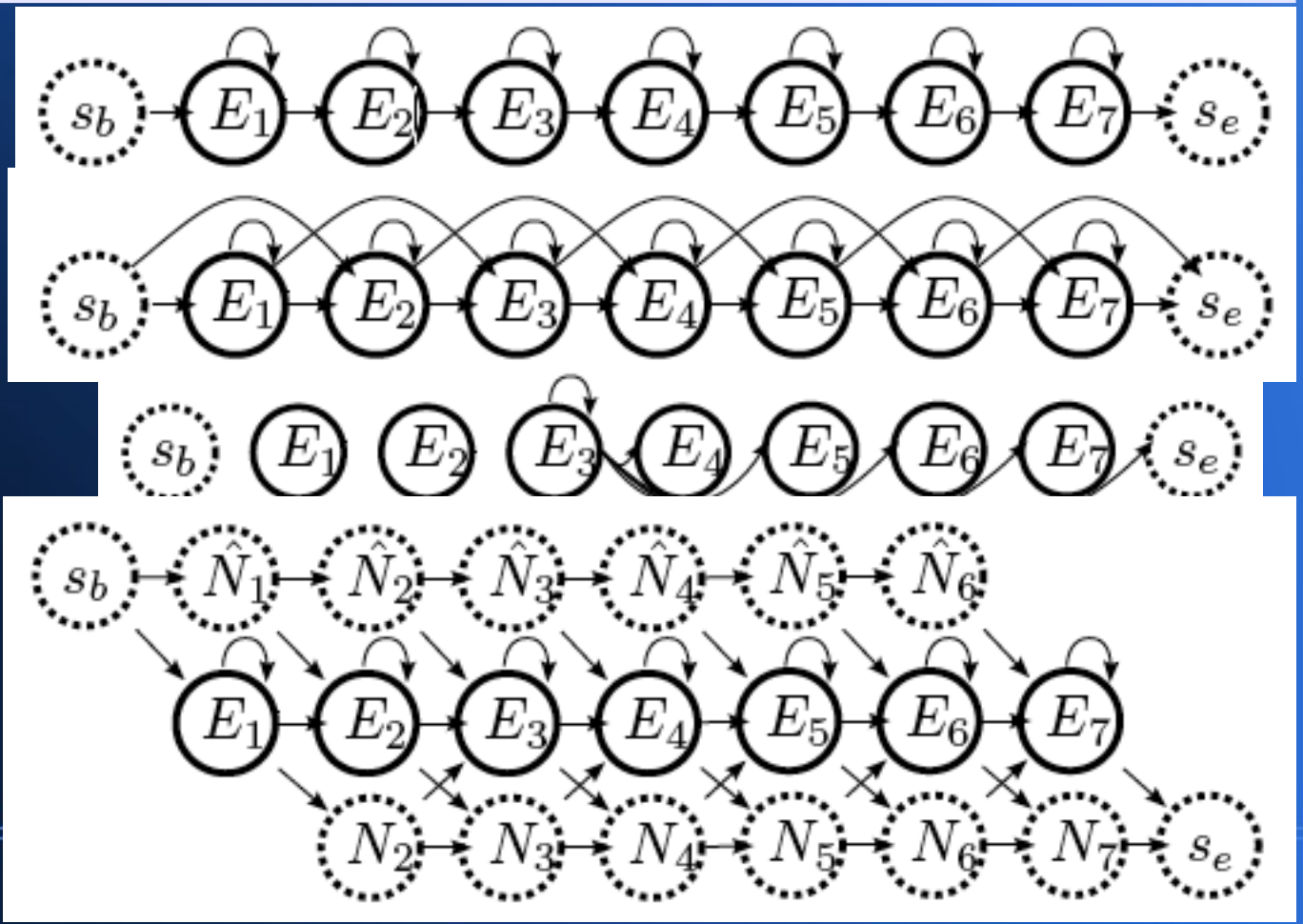
Pattern Model Training



AHMM Topologies *(Tangent)*

```
typedef patterns::model::chain_hmm<observations::normal>
   ahmm_type;
```

- Chain
- Chain Skip
- L-to-R
- Reduced



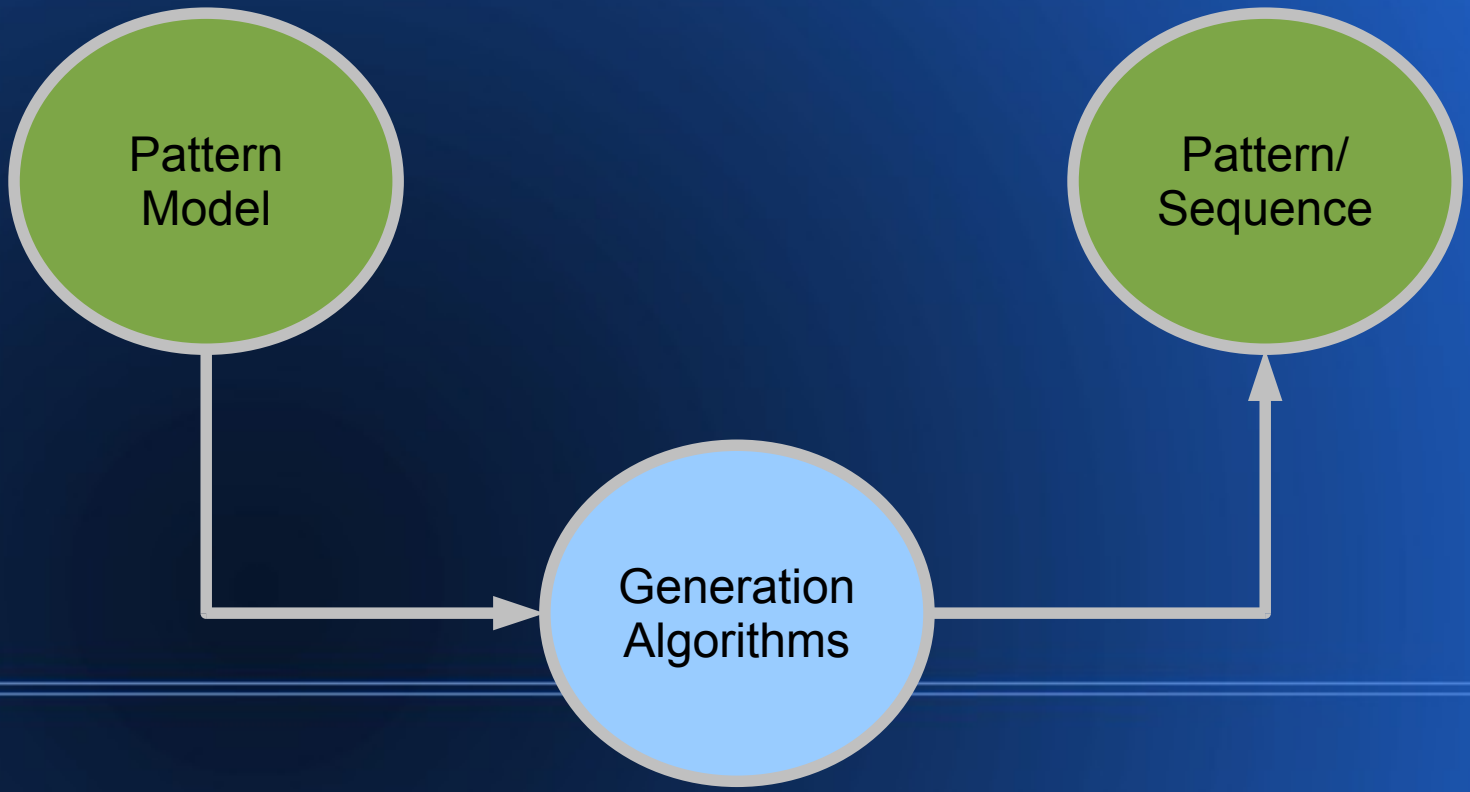
Pattern Model Training

```
std::vector<std::vector<double> > examples(2);
using namespace boost::assign;
examples[0] += 0, 0.1, -0.1, 1.1, 0.9, 1.0;
examples[1] += 0.1, -0.1, 0.1, -0.1, 0, 1.0;

typedef patterns::model::chain_hmm<observations::normal>
    ahmm_type;

ahmm_type ahmm(2);
patterns::training::naive_alignment(ahmm, examples);

typedef patterns::inference::best_match<ahmm_type,
        ame::selectors::vector>
    inference_type;

patterns::training::best_match_maximization<inference_type>
   (ahmm, examples, 1, 0.0);
```

Pattern Model Training (Task API)

```
std::vector<std::vector<double> > examples(2);
using namespace boost::assign;
examples[0] += 0, 0.1, -0.1, 1.1, 0.9, 1.0;
examples[1] += 0.1, -0.1, 0.1, -0.1, 0, 1.0;

patterns::training_task
<
   patterns::model::chain_hmm<observations::normal>,
   patterns::expectation_maximization_training
>
   em_training_task;

em_training_task.add_pattern_with_examples(2, examples);
```

Generation



Generation

```
patterns::generation::generator<ahmm_type>
    generator(ahmm);

boost::mt19937 engine;
engine.seed(static_cast<unsigned int>(std::time(0)));
boost::variate_generator
<
   boost::mt19937,
   boost::uniform_real<double>
>
   rng(engine, boost::uniform_real<double>());

double result = generator(rng).get();
```

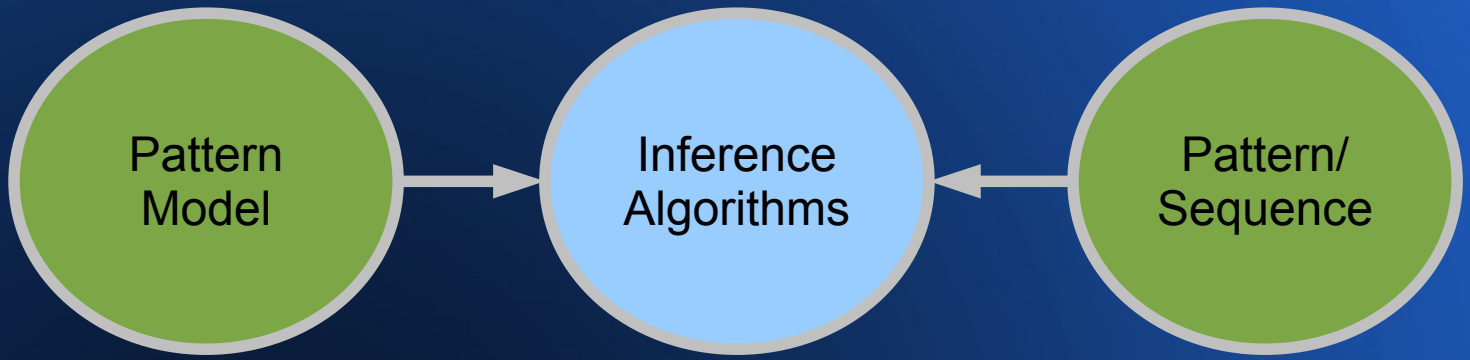
Synthesis (Task API)

```
patterns::synthesis_task
<
   patterns::model::chain_hmm<observations::normal>,
   patterns::expectation_maximization_training
>
   task;

std::vector<std::vector<double> > examples(2);
using namespace boost::assign;
examples.front() += 0, 1.1;
examples.back() += 0.1, 1.2;
task.add_pattern_with_examples(2, examples);

// ...
```

Synthesis (Task API)

```
std::vector<double> synthesized;

task.synthesize(synthesized);
// synthesized.size() == 2u
// e.g. synthesized == 0.03, 1.19

task.synthesize(synthesized);
// synthesized.size() == 2u
// e.g. synthesized == 0.18, 1.02

task.reset();
for(int i=0; i<2; i++)
   double value = task.generate();
```

Inference



Inference

```
typedef patterns::inference::best_match
<
   ahmm_type,
   ame::selectors::vector
>
   inference_type;

inference_type inference(ahmm);

double best_probability = p_match_sequence(inference,
    examples[0]);
double partial_probability = inference[2][1];
```

Classification

- Given a sequence of observations, classify it into one of the available patterns
  - The sequence is assumed to be “segmented”, i.e. containing an instance of one pattern

Classification (Task API)

```
patterns::classification_task
<
   patterns::model::chain_hmm<observations::normal>,
   patterns::expectation_maximization_training,
   patterns::forward_inference
>
   task;
```

Classification (Task API)

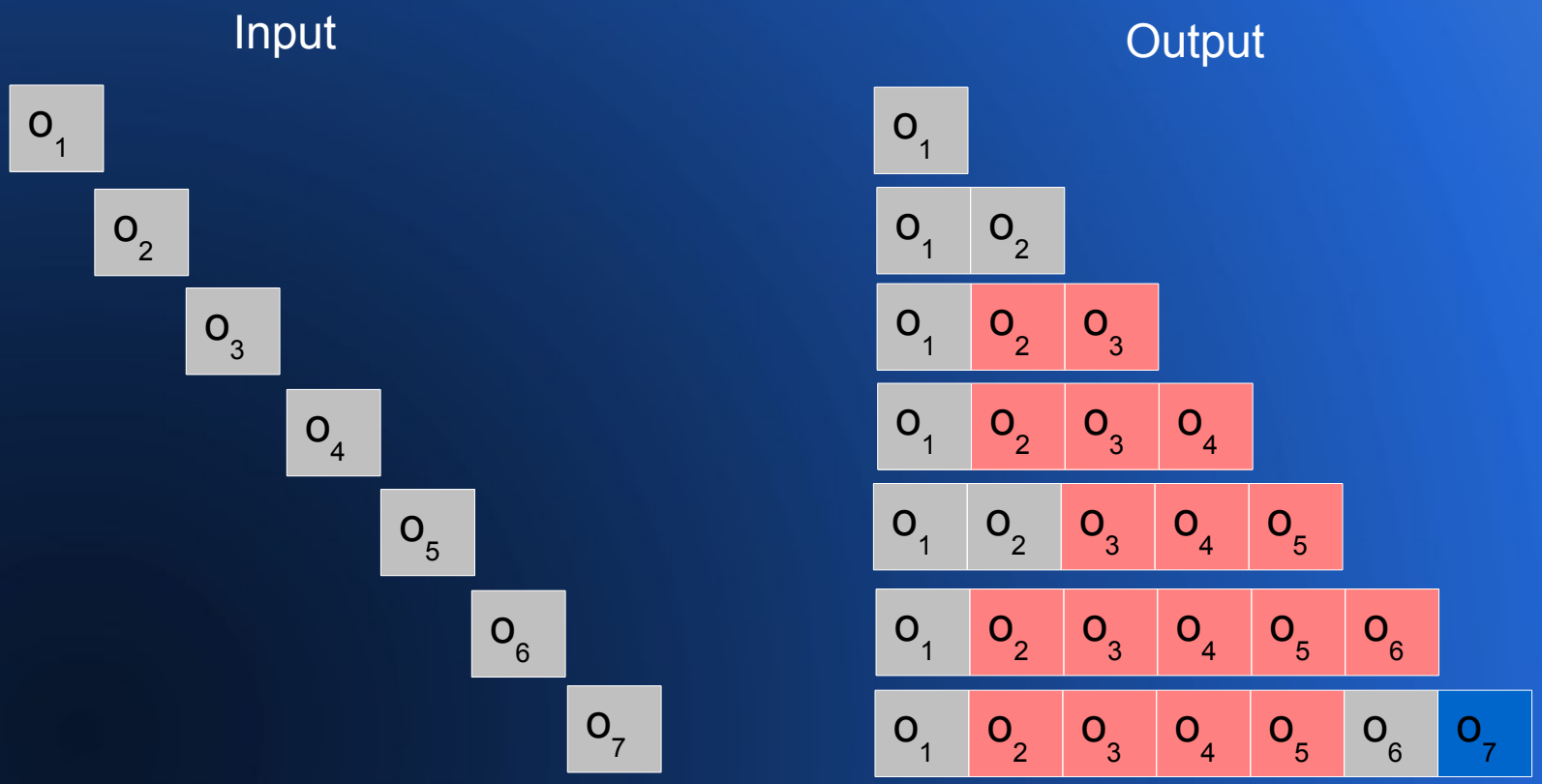
```
std::vector<std::vector<double> > examples(2);
using namespace boost::assign;
examples[0] += 0, 0.1, -0.1, 0.2, 0.2, 0.2, 1.1;
examples[1] += 0.1, 1.2;
task.add_pattern_with_examples(2, examples);

examples.clear();
examples.resize(2);
examples[0] += -0.4, 1.1;
examples[1] += -0.3, 1.2;
task.add_pattern_with_examples(2, examples);
```

Classification (Task API)

```
std::vector<double> pattern;
pattern += -0.2, 1.1;

// task.classify(pattern) should return 1
int classification_result = task.classify(pattern);
```

On-Line Recognition

- Data comes real-time and is not segmented



On-Line Recognition (Task API)

```
typedef patterns::semantic_recognition_task<
        observations::normal>
    recognition_task_type;

recognition_task_type recognition_task
(
    task,
    0.01,
    ame::selectors::circular_buffer(20)
);
```

On-Line Recognition (Task API)

```
std::vector<double> sequence;
sequence += -0.35,-0.35,1.15,10.0,10.0,0.1,1.2,0.95,1.2;

std::vector<double>::iterator it=sequence.begin();

std::vector<patterns::semantic_event> result;

result = recognition_task.match(*it++);
// result.size() == 1 : one event occurred
//   (beginning of pattern 2)
// result[0].type == patterns::semantic_event::detected
// result[0].pattern == 2
// result[0].beginning == 0
// result[0].end == -1 (no end detected yet)
```

On-Line Recognition (Task API)

```
// -0.35,-0.35,1.15,10.0,10.0,0.1,1.2,0.95,1.2;

result = recognition_task.match(*it++);
// result.size() == 1 : revising start of pattern 2
// result[0].type == patterns::semantic_event::revised
// result[0].pattern == 2
// result[0].beginning == 1
// result[0].end == -1 (no end detected yet)
```

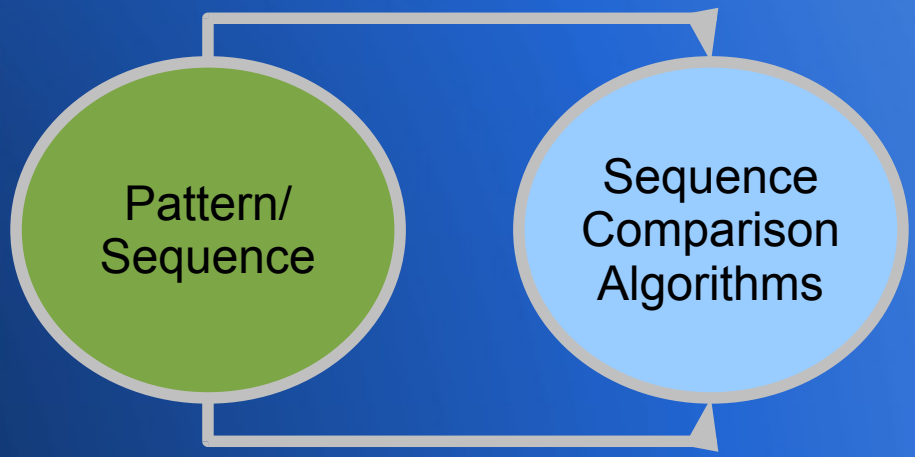
On-Line Recognition (Task API)

```
// -0.35,-0.35,1.15,10.0,10.0,0.1,1.2,0.95,1.2;

result = recognition_task.match(*it++);
// result.size() == 0 : no events
```

On-Line Recognition (Task API)

```
// -0.35,-0.35,1.15,10.0,10.0,0.1,1.2,0.95,1.2;

result = recognition_task.match(*it++);
// result.size() == 1 : detecting end of pattern 2
// result[0].type == patterns::semantic_event::detected
// result[0].pattern == 2
// result[0].beginning == 1
// result[0].end == 2
```

Sequence Comparison



Sequence Comparison

```
struct function
{
    double operator()(char a, char b) const
    {
        return (a == b) ? 1 : -1;
    }
};

using namespace ame::patterns::dp;

std::vector<element<char> > alignment =
    compute_edit_distance(
        std::string("bananarama"),
        std::string("bannana"),
        function()
    );

BOOST_FOREACH(const element<char> &el, alignment)
    std::cout << (el.source ? *el.source : ' ') << ", "
        << (el.target ? *el.target : ' ')
        << std::endl;
```

```
a,
m,
a, a
r, n
a, a
n, n
a,
n, n
a, a
b, b
```

Additional Thoughts

- ofxPatterns
- Dataset concepts / generic experiment algorithms
- Generic library for a generic problem

More Information

- Google “AME Patterns”
- http://ame4.hc.asu.edu/amelia/patterns/
  - (get code from SVN)
- http://ame4.hc.asu.edu/amelia/ofxpatterns/
  - (packaged releases)
- http://ame4.hc.asu.edu/amelia/
- stjepan.rajko@gmail.com