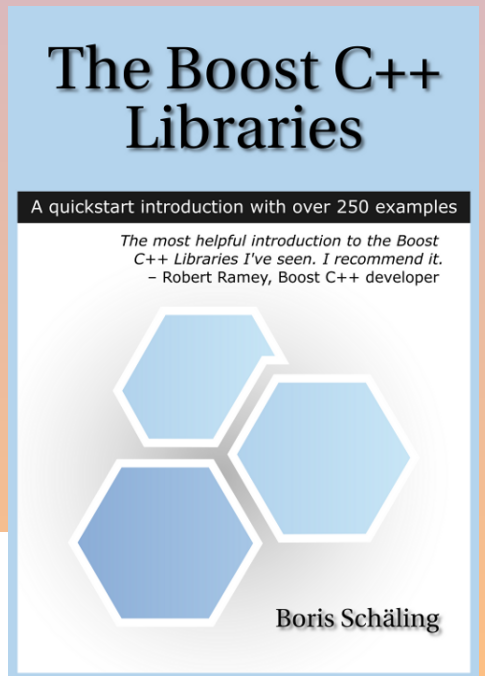
Boost, Part II

Akim Demaille akim@lrde.epita.fr

June, 1st 2015

(2016-11-16 09:53:54 +0100 121bea3)



Boost, Part II

Containers

Containers

Boost.Optional

Genericity in Haskell: Signature of Maybe

```
module Maybe(
    isJust, isNothing,
    fromJust, fromMaybe, listToMaybe, maybeToList,
    catMaybes, mapMaybe,
    -- ...and what the Prelude exports
    Maybe(Nothing, Just),
    maybe)
where

data Maybe a = Nothing | Just a

isJust, isNothing    :: Maybe a -> Bool
fromJust             :: Maybe a -> a
fromMaybe            :: a -> Maybe a -> a
listToMaybe          :: [a] -> Maybe a
maybeToList          :: Maybe a -> [a]
catMaybes            :: [Maybe a] -> [a]
mapMaybe             :: (a -> Maybe b) -> [a] -> [b]
```

Genericity in Haskell: Implementation of Maybe

```haskell
isJust                  :: Maybe a -> Bool
isJust (Just a)         =  True
isJust Nothing          =  False

isNothing               :: Maybe a -> Bool
isNothing               =  not . isJust

fromJust                :: Maybe a -> a
fromJust (Just a)       =  a
fromJust Nothing        =  error "Maybe.fromJust: Nothing"

fromMaybe               :: a -> Maybe a -> a
fromMaybe d Nothing     =  d
fromMaybe d (Just a)    =  a
```

Boost.Optional: Optional Return Values

```
char* get_async_input()
{
  if (queue.empty())
    return nullptr;
  else
    {
      static char res;
      res = queue.top();
      return &res;
    }
}

void receive_async_message()
{
  char* rcv;
  while ((rcv = get_async_input()) && !timeout())
    output(*rcv);
}
```

Boost.Optional: Optional Return Values

```
boost::optional<char> get_async_input()
{
  if (queue.empty())
    return {};
  else
    return {queue.top()};
}

void receive_async_message()
{
  boost::optional<char> rcv;
  // The safe Boolean conversion from 'rcv' is used here.
  while ((rcv = get_async_input()) && !timeout())
    output(*rcv);
}
```

Boost.Optional: Optional Local Variables

```
{
  boost::optional<std::string> name;
  if (database.open())
    name.reset(database.lookup(employer_name));
  else if (can_ask_user)
    name.reset(user.ask(employer_name));

  if (name)
    print(*name);
  else
    print("employer's name not found!");
}
```

Boost.Optional: Optional Data Members

```
class Figure
{
  boost::optional<rect> clipping_rect_;
public:
  Figure() { /* 'clipping_rect_' is uninitialized at this point. */ }

  void clip_in_rect(rect const& rect) { ...
    clipping_rect_.reset(rect); // initialized here.
  }

  void draw(canvas& cvs) {
    if (clipping_rect_)
      do_clipping(*clipping_rect_);
    cvs.drawXXX(..);
  }

  // May return nullptr.
  rect const* get_clipping_rect() { return get_pointer(clipping_rect_); }
};
```

Boost.Variant

Sum Types in Haskell: Maybe

```haskell
data Maybe a = Nothing | Just a

isJust                  :: Maybe a -> Bool
isJust (Just a)         =  True
isJust Nothing          =  False
```

C++ Unions

C++ 03 unions:

- may define member functions
- may define special member functions
- may not include objects (that have non trivial ctor/dtor)
  "POD", Plain Old Data

C++ Unions

C++ 03 unions may define (special) member functions.

```
union U
{
  U()
    : a(5)
  {}

  U(const U& r)
  { p = r.p; }

  U& operator=(const U& r)
  { p = r.p; return *this;}

  ~U()
  { delete [] p; }

  int a, b;
  char* p;
};
```

C++ Unions

C++ 11 unions may define (special) member functions.

```
union U
{
  U()
    : a(5)
  {}

  U(U&& r)
  { p = r.p; r.p = nullptr;}

  U&& operator=(U&& r)
  { p = r.p; r.p = nullptr; return *this;}

  ~U()
  { delete [] p; }

  int a, b;
  char* p;
};
```

C++ Unions

C++ 11 unions may include objects (with constructor and destructor), but then must define ctor/dtor.

```
union U
{
  U(){}
  ~U() {}
  int ival;
  std::string sval;
};
```

C++ Unions

C++ 11 unions may not have static members, or reference members.

```
union U
{
  int &r;          // wrong,
  int &&rr;        // wrong,
  static int si; // and wrong.
};
```

Static member functions are ok.

```
union U
{
  static int& instance();
};
```

C++ Unions

Unions may be anonymous.

```
int main()
{
  union
  {
    float f;
    unsigned int i;
  };
 f = 2;
 std::cout << std::hex << i << std::endl;
}
```

C++ Unions-like Classes

Unions may be anonymous.

```
class C
{
  union
  {
    int a;
    char* p;
  };
};
```

C++ 11 Unions May Leak

```
union U
{
  int ival;
  std::string sval;
  U() : sval() {}
  ~U() {}
  U& operator=(const std::string s) { sval = s; return *this; };
  U& operator=(int i)               { ival = i; return *this; };
};

int main()
{
  while (true)
    {
      U u;
      u = "Hello, World!";
      u = 42;
    }
}
```

Boost.Variant

```cpp
#include <boost/variant.hpp>
#include <iostream>

int main()
{
  boost::variant<int, std::string> u("hello world");
  std::cout << u << std::endl; // output: hello world.
}
```

Boost.Variant: Bad Style

```
using int_or_str = boost::variant<int, std::string>;
void times_two(int_or_str& arg)
{
  if (int* pi = boost::get<int>(&arg))
    *pi *= 2;
  else if (std::string* pstr = boost::get<std::string>(&arg))
    *pstr += *pstr;
}
```

There is also `int which() const`, and `std::type_info& type() const`.

Visitors

```
struct times_two_visitor : boost::static_visitor<>
{
  void operator()(int& i)           const { i *= 2; }
  void operator()(std::string& str) const { str += str; }
};

boost::apply_visitor(times_two_visitor(), v);
```

Boost.Variant

```
#include <boost/variant.hpp>
#include <iostream>

struct my_visitor : public boost::static_visitor<int>
{
  int operator()(int i)                    const { return i; }
  int operator()(const std::string& str) const { return str.size(); }
};

int main()
{
  boost::variant<int, std::string> u("hello world");
  std::cout << u << std::endl; // output: hello world.

  int result = boost::apply_visitor(my_visitor(), u);
  std::cout << result << std::endl; // output: 11.
}
```

Generic Visits

```
struct times_two_generic : public boost::static_visitor<>
{
  template <typename T>
  void operator()(T& arg) const
  {
    arg += arg;
  }
};

int main()
{
  std::vector<int_or_str> iss = {21, "knock"};
  times_two_generic visitor;
  std::for_each(begin(iss), end(iss), boost::apply_visitor(visitor));
  for (const auto& is: iss)
    std::cout << is << std::endl;
}
```

Binary Visitations

```
struct are_strict_equals : public boost::static_visitor<bool>
{
  template <typename T, typename U>
  bool operator()(const T&, const U&) const
  {
    return false;
  }

  template <typename T>
  bool operator()(const T& lhs, const T& rhs) const
  {
    return lhs == rhs;
  }
};
```

Boost.Any

Boost.Any: Creating

```cpp
#include <boost/any.hpp>

using many = std::list<boost::any>;
int main()
{
  boost::any a;
  many m;
  // Push empty.
  m.push_back(a);
  a = 42;
  m.push_back(a);
  m.push_back(51);
  const char *cp = "Hello, World!";
  m.push_back(cp);
  m.push_back(std::string{"Bonjour, Monde !"});
  count_all(m, std::cout);
}
```

Boost.Any: Querying

```
bool is_empty (const boost::any& a)
     { return a.empty(); }
bool is_int   (const boost::any& a)
     { return a.type() == typeid(int); }
bool is_string(const boost::any& a)
     { return boost::any_cast<std::string>(&a); }
bool is_char_ptr(const boost::any& a)
{
  try
  {
    boost::any_cast<const char *>(a);
    return true;
  }
  catch (const boost::bad_any_cast&)
  {
    return false;
  }
}
```

Boost.Any

```
void count_all(many& m, std::ostream& out)
{
#define COUNT(What)                                          \
  out << "#" #What " == "                                    \
      << std::count_if(begin(m), end(m), is_ ## What)        \
      << std::endl;

  COUNT(empty);
  COUNT(int);
  COUNT(char_ptr);
  COUNT(string);
#undef COUNT
}
```

```
#empty == 1
#int == 2
#char_ptr == 1
#string == 1
```

Boost.Variant vs. Boost.Any

Boost.Variant

- guarantees the type of its content is one of a finite, user-specified set of types
- provides compile-time checked visitation of its content
- enables generic visitation of its content
- offers an efficient, stack-based storage scheme

Boost.Any

- allows virtually any type for its content
- provides the no-throw guarantee for its `swap`
- makes little use of template metaprogramming techniques (good for error messages, and compilation).

Boost.Variant vs. Boost.Any

Boost.Variant

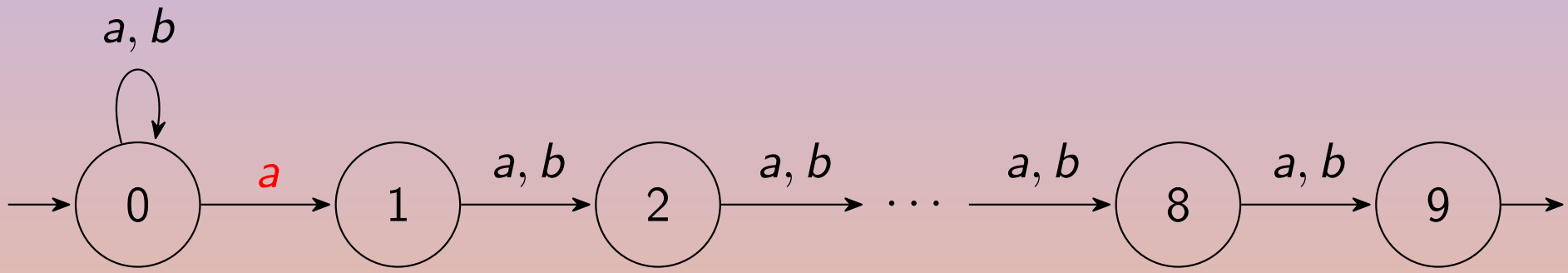
- guarantees the type of its content is one of a finite, user-specified set of types
- provides compile-time checked visitation of its content
- enables generic visitation of its content
- offers an efficient, stack-based storage scheme

Boost.Any

- allows virtually any type for its content
- provides the no-throw guarantee for its `swap`
- makes little use of template metaprogramming techniques (good for error messages, and compilation).

Boost.Flyweight

Vaucanson 2: De Bruijn 8



The Dot Language

```
digraph
{
  vcsn_context = "lal_char(ab)_b"
  rankdir = LR
  node [shape = circle]
  {
    node [style = invis, shape = none, label = "", width = 0, height = 0]
    I1
    F12
  }
  { 1 2 3 4 5 6 7 8 9 10 11 12 }
  I1 -> 1
  1 -> 1 [label = "a, b"]
  1 -> 2 [label = "a"]
  2 -> 3 [label = "a, b"]
  3 -> 4 [label = "a, b"]
  // ...
  11 -> 12 [label = "a, b"]
  12 -> F12
}
```

Vaucanson: Performances of the Parser

'`4096.dot`':

- 1 initial state
- 1 initial transitions
- 4096 inner states
- 12284 inner transitions
- 2048 final states
- 2048 final transitions
- 16 392 lines
- 86 033 words
- 420 787 characters

Vaucanson: Performances of the Parser

'`4096.dot`':

- 1 initial state
- 1 initial transitions
- 4096 inner states
- 12284 inner transitions
- 2048 final states
- 2048 final transitions
- 16 392 lines
- 86 033 words
- 420 787 characters

Vaucanson: Performances of the Parser

'`4096.dot`':

- 1 initial state
- 1 initial transitions
- 4096 inner states
- 12284 inner transitions
- 2048 final states
- 2048 final transitions
- 16 392 lines
- 86 033 words
- 420 787 characters

Vaucanson: Performances of the Parser

'`4096.dot`':

- 1 initial state
- 1 initial transitions
- 4096 inner states
- 12284 inner transitions
- 2048 final states
- 2048 final transitions
- 16 392 lines
- 86 033 words
- 420 787 characters

Vaucanson: Performances of the Parser

'4096.dot':

- 1 initial state
- 1 initial transitions
- 4096 inner states
- 12284 inner transitions
- 2048 final states
- 2048 final transitions
- 16 392 lines
- 86 033 words
- 420 787 characters

```
$ time vcsn-cat -f 4096.dot -Onull
vcsn-cat -f 4096.dot -Onull  9,07s user 0,03s system 99% cpu 9,109 total
```

Vaucanson: Moving to Boost.Flyweight

```
 struct semantic_type
 {
   // Identifiers (attributes and node names).
-  using string_t = std::string;
+  using string_t =
+    boost::flyweight<std::string, boost::flyweights::no_tracking>;
   string_t string;
   // A set of states.
-  using states_t = std::vector<std::string>;
+  using states_t = std::vector<string_t>;
   states_t states;
   // (Unlabeled) transitions.
-  using transitions_t = std::vector<std::pair<std::string, std::string>>;
+  using transitions_t = std::vector<std::pair<string_t, string_t>>;
   transitions_t transitions;
 };
```

Vaucanson: Performances of the Parser

Before:

```
$ time vcsn-cat -f 4096.dot -Onull
vcsn-cat -f 4096.dot -Onull  9,07s user 0,03s system 99% cpu 9,109 total
```

After:

```
$ time vcsn-cat -f 4096.dot -Onull
vcsn-cat -f 4096.dot -Onull  0,85s user 0,02s system 99% cpu 0,870 total
```

Boost.DynamicBitset

Boost.DynamicBitset

```
#include <iostream>
#include <string>

#include <boost/dynamic_bitset.hpp> // boost::dynamic_bitset<>
#include <boost/utility/binary.hpp> // BOOST_BINARY

template <typename T>
void output(std::string s, const T& t)
{
  s += ':';
  s.resize(24, ' ');
  std::cout << s << t << std::endl;
}
```

Boost.DynamicBitset

```
boost::dynamic_bitset<> mask(12, BOOST_BINARY(0101 0101 0101));

int main()
{
  output("mask", mask);
  boost::dynamic_bitset<> x;
  output("x.size()", x.size());
  std::cout << "Enter a bitset in binary: x = " << std::flush;
  if (std::cin >> x)
    {
      std::cout << std::endl;
      play(x);
    }
}
```

Boost.DynamicBitset

```
void play(const boost::dynamic_bitset<>& x)
{
  output("Input number", x);
  const std::size_t sz = x.size();
  output("x.size() is now", sz);

  try {
    unsigned long ul = x.to_ulong();
    output("As unsigned long", ul);
  } catch (const std::overflow_error &) {
    output("As unsigned long", "(overflow exception)");
  }

  mask.resize(sz);
  output("Mask (possibly resized)", mask);
  output("And with mask",           x & mask);
  output("Or with mask",            x | mask);
  output("Shifted left by 1",       x << 1);
  output("Shifted right by 1",      x >> 1);
}
```

Boost.DynamicBitset

Input:

```
101010
```

Output:

```
mask:                       010101010101
x.size():                   0
Enter a bitset in binary: x =
Input number:               101010
x.size() is now:            6
As unsigned long:           42
Mask (possibly resized):010101
And with mask:              000000
Or with mask:               111111
Shifted left by 1:          010100
Shifted right by 1:         010101
```

Determinize in Vaucanson

```
$ time vcsn-determinize -f de-bruijn-18.dot -Onull
real    0m13.049s
user    0m12.773s
sys     0m0.276s
$ time vcsn-determinize -f de-bruijn-20.dot -Onull
real    0m56.780s
user    0m55.775s
sys     0m0.988s
```

Vaucanson: Moving to dynamic_bitset<>

```
commit 995d223fb0794784ecda599d2bda41c5106f366a
Author: Victor Marie Santet <vsantet@lrde.epita.fr>
Date:   Mon Feb 25 17:40:27 2013 +0100

    determinize: use bitsets

    In order to increase performance, use bitsets to represent
    states sets to compute determinization algorithm.
    Before computing, stores all successors states.

    * vcsn/algos/determinize.hh: Specialize std::hash.
    Use boost::dynamic_bitset as states sets.
    Store successors sets before computing.
```

Determinize in Vaucanson

Before:

```
$ time vcsn-determinize -f de-bruijn-18.dot -Onull
real    0m13.049s
user    0m12.773s
sys     0m0.276s
$ time vcsn-determinize -f de-bruijn-20.dot -Onull
real    0m56.780s
user    0m55.775s
sys     0m0.988s
```

After:

```
$ time vcsn-determinize -f de-bruijn-18.dot -Onull
real     0m2.181s
user     0m2.048s
sys      0m0.132s
$ time vcsn-determinize -f de-bruijn-20.dot -Onull
real     0m8.653s
user     0m8.177s
sys      0m0.476s
```

Conversions

Conversions

Boost.Format

Boost.Format

```
boost::format fmter{"%2% %1%"};
fmter % 36; fmter % 77;

// You can take the string result:
string s  = fmter.str();

// possibly several times:
s = fmter.str();

// Streamable.
std::cout << fmter;

// You can also do all steps at once:
std::cout << boost::format("%2% %1%") % 36 % 77;

// using the str free function:
s = str(boost::format("%2% %1%") % 36 % 77);
```

Boost.Format: Performances

```
const auto runs = boost::irange(0, 1000 * 1000);

using boost::chrono::steady_clock;
{
  steady_clock::time_point start = steady_clock::now();
  {
    std::string s;
    for (auto i: runs)
      s = str(boost::format("%2% %1%") % i % i);
  }
  boost::chrono::duration<double> sec = steady_clock::now() - start;
  std::cout << boost::format("%|10|: %|4.2f|s\n")
               % "several" % sec.count();
}
```

Boost.Format: Performances

```
{
  steady_clock::time_point start = steady_clock::now();
  {
    std::string s;
    boost::format fmt{"%2% %1%"};
    for (auto i: runs)
      s = str(fmt % i % i);
  }
  boost::chrono::duration<double> sec = steady_clock::now() - start;
  std::cout << boost::format("%|10|: %|4.2f|s\n")
               % "once" % sec.count();
}
```

Boost.Format: Performances

```
{
  steady_clock::time_point start = steady_clock::now();
  {
    std::string s;
    for (auto i: runs)
      s = std::to_string(i) + " " + std::to_string(i);
  }
  boost::chrono::duration<double> sec = steady_clock::now() - start;
  std::cout << boost::format("%|10|: %|4.2f|s\n")
               % "none" % sec.count();
}
```

Boost.Format: Performances

```
several: 4.07s
   once: 1.61s
   none: 0.84s
```

```
several: 1.77s
   once: 0.87s
   none: 0.70s
```

Boost.Format: Performances

```
several: 4.07s
   once: 1.61s
   none: 0.84s
```

```
several: 1.77s
   once: 0.87s
   none: 0.70s
```

Boost.LexicalCast

atof

```
#include <cerrno>
#include <cstring>
#include <iostream>

int main(int argc, char const* argv[])
{
  for (int i = 0; i < argc; ++i)
    {
      errno = 0;
      float f = atof(argv[i]);
      int err = errno;
      std::cout << argv[i] << " => " << f;
      if (err)
        std::cout << " (" << strerror(err) << ")";
      std::cout << std::endl;
    }
}
```

atof

```
./4-boost/a.out => 0
1.2 => 1.2
2..2 => 2
e123f => 0
123f => 123
1e-999 => 0 (Result too large)
-123e-999 => -0 (Result too large)
```

atof

On Mac OS X, 'man atof':

```
SYNOPSIS
   #include <stdlib.h>

   double atof(const char *str);

   #include <xlocale.h>

   double atof_l(const char *str, locale_t loc);

IMPLEMENTATION NOTES
   The atof() and atof_l() functions have been deprecated by strtod()
   and strtod_l() and should not be used in new code.

ERRORS
   The function atof() need not affect the value of errno on an error.
```

So use strtof.

strtof

```
#include <cerrno>
#include <cstdlib>
#include <cstring>
#include <iostream>

int main(int argc, char const* argv[])
{
  for (int i = 0; i < argc; ++i)
    {
      errno = 0;
      char* end;
      float f = strtof(argv[i], &end);
      int err = *end ? EINVAL : errno;
      std::cout << argv[i] << " => " << f
                << " (" << strerror(err) << ")"
                << std::endl;
    }
}
```

strtof

```
./4-boost/a.out => 0 (Invalid argument)
1.2 => 1.2 (Undefined error: 0)
2..2 => 2 (Invalid argument)
e123f => 0 (Invalid argument)
123f => 123 (Invalid argument)
1e-999 => 0 (Result too large)
-123e-999 => -0 (Result too large)
```

- Beware that `strtof` does not exist in C++ 03!
- What about conversion to a short?

strtof

```
./4-boost/a.out => 0 (Invalid argument)
1.2 => 1.2 (Undefined error: 0)
2..2 => 2 (Invalid argument)
e123f => 0 (Invalid argument)
123f => 123 (Invalid argument)
1e-999 => 0 (Result too large)
-123e-999 => -0 (Result too large)
```

- Beware that `strtof` does not exist in C++ 03!
- What about conversion to a short?

The Proper Conversion To/From Strings in C++

```
template <typename Target, typename Source>
Target omnipotent_cast(const Source& input)
{
  std::stringstream ss;
  ss << input;
  if (ss.fail())
    throw std::logic_error("write fail");
  Target res;
  ss >> res;
  if (ss.fail())
    throw std::logic_error("read fail");
  return res;
}
```

Boost.LexicalCast

```
try
{
  std::cout << boost::lexical_cast<short>(arg) << std::endl;
}
catch (const boost::bad_lexical_cast& e)
{
  std::cerr << arg << ": not a short: " << e.what() << std::endl;
}
```

- *Much* faster than stream-based code
- Faster that C unsafe functions
- C++ 11 defines `to_string` (for '`unsigned? (int|long|long long)`' and '`float|double|long double`')
- Beware: behaves surprisingly on '`lexical_cast<int8_t>("127")`'
  - throws `boost::bad_lexical_cast`
  - because `int8_t` is `char` (or `signed char`)
  - in which case it reads a single byte ('1')

Boost.LexicalCast

```
try
{
  std::cout << boost::lexical_cast<short>(arg) << std::endl;
}
catch (const boost::bad_lexical_cast& e)
{
  std::cerr << arg << ": not a short: " << e.what() << std::endl;
}
```

- *Much* faster than stream-based code
- Faster that C unsafe functions
- C++ 11 defines `to_string` (for '`unsigned? (int|long|long long)`' and '`float|double|long double`')
- Beware: behaves surprisingly on '`lexical_cast<int8_t>("127")`'
  - throws `boost::bad_lexical_cast`
  - because `int8_t` is `char` (or `signed char`)
  - in which case it reads a single byte ('`1`')

Boost.LexicalCast

```
try
{
  std::cout << boost::lexical_cast<short>(arg) << std::endl;
}
catch (const boost::bad_lexical_cast& e)
{
  std::cerr << arg << ": not a short: " << e.what() << std::endl;
}
```

- *Much* faster than stream-based code
- Faster that C unsafe functions
- C++ 11 defines `to_string`
  (for '`unsigned? (int|long|long long)`' and '`float|double|long double`')
- Beware: behaves surprisingly on '`lexical_cast<int8_t>("127")`'
  - throws `boost::bad_lexical_cast`
  - because `int8_t` is `char` (or `signed char`)
  - in which case it reads a single byte ('`1`')

Boost.LexicalCast

```
try
{
  std::cout << boost::lexical_cast<short>(arg) << std::endl;
}
catch (const boost::bad_lexical_cast& e)
{
  std::cerr << arg << ": not a short: " << e.what() << std::endl;
}
```

- *Much* faster than stream-based code
- Faster that C unsafe functions
- C++ 11 defines `to_string`
  (for '`unsigned? (int|long|long long)`' and
  '`float|double|long double`')
- Beware: behaves surprisingly on '`lexical_cast<int8_t>("127")`'
  - throws `boost::bad_lexical_cast`
  - because `int8_t` is `char` (or `signed char`)
  - in which case it reads a single byte ('1')

Boost.LexicalCast

```
try
{
  std::cout << boost::lexical_cast<short>(arg) << std::endl;
}
catch (const boost::bad_lexical_cast& e)
{
  std::cerr << arg << ": not a short: " << e.what() << std::endl;
}
```

- *Much* faster than stream-based code
- Faster that C unsafe functions
- C++ 11 defines `to_string`
  (for '`unsigned? (int|long|long long)`' and '`float|double|long double`')
- Beware: behaves surprisingly on '`lexical_cast<int8_t>("127")`'
  - throws `boost::bad_lexical_cast`
  - because `int8_t` is `char` (or `signed char`)
  - in which case it reads a single byte ('1')

Boost.LexicalCast

```
try
{
  std::cout << boost::lexical_cast<short>(arg) << std::endl;
}
catch (const boost::bad_lexical_cast& e)
{
  std::cerr << arg << ": not a short: " << e.what() << std::endl;
}
```

- *Much* faster than stream-based code
- Faster that C unsafe functions
- C++ 11 defines `to_string`
  (for '`unsigned? (int|long|long long)`' and '`float|double|long double`')
- Beware: behaves surprisingly on '`lexical_cast<int8_t>("127")`'
  - throws `boost::bad_lexical_cast`
  - because `int8_t` is `char` (or `signed char`)
  - in which case it reads a single byte ('1')

Boost.LexicalCast

```
try
{
  std::cout << boost::lexical_cast<short>(arg) << std::endl;
}
catch (const boost::bad_lexical_cast& e)
{
  std::cerr << arg << ": not a short: " << e.what() << std::endl;
}
```

- *Much* faster than stream-based code
- Faster that C unsafe functions
- C++ 11 defines `to_string`
  (for '`unsigned? (int|long|long long)`' and '`float|double|long double`')
- Beware: behaves surprisingly on '`lexical_cast<int8_t>("127")`'
  - throws `boost::bad_lexical_cast`
  - because `int8_t` is `char` (or `signed char`)
  - in which case it reads a single byte ('1')

Boost.NumericConversion

Boost.NumericCast

```
using boost::numeric_cast;

try
  {
    int i = 42;
    short s = numeric_cast<short>(i); // Succeeds (is in range).
  }
catch (boost::numeric::negative_overflow& e)
  {
    std::cout << e.what() << std::endl;
  }
catch (boost::numeric::positive_overflow& e)
  {
    std::cout << e.what() << std::endl;
  }
```

Boost.NumericCast

```
float f = -42.1234;
try
  {
    // Throws a boost::numeric::negative_overflow.
    auto i = numeric_cast<unsigned int>(f);
  }
catch(boost::numeric::bad_numeric_cast& e)
  {
    std::cout << e.what() << std::endl;
  }

double d = f + numeric_cast<double>(123); // int -> double
```

Boost.NumericCast

```
try
  {
    auto ulmax = std::numeric_limits<unsigned long>::max();
    // Throws a boost::numeric::positive_overflow.
    auto c = numeric_cast<unsigned char>(ulmax);
  }
catch (boost::numeric::positive_overflow& e)
  {
    std::cout << e.what() << std::endl;
  }
```

- unsigned operations cannot cause overflow
- unsigned conversions are range checked by `numeric_cast`

Can C++ 11 Help?

5508110: Why is this program erroneously rejected by three C++ compilers?

“I am having some difficulty compiling a C++ program that I've written. This program is very simple and, to the best of my knowledge, conforms to all the rules set forth in the C++ Standard. I've read over the entirety of ISO/IEC 14882:2003 twice to be sure.
The program is as follows:

```
#include <iostream>

int main(int argc, char** argv)
{
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

“Here is the output I received when trying to compile this program with Visual C++ 2010:

```
c:\dev>cl /nologo helloworld.png
cl : Command line warning D9024 : unrecognized source file type
     'helloworld.png', object file assumed
helloworld.png : fatal error LNK1107: invalid or corrupt file:
     cannot read at 0x5172
```

Dismayed, I tried g++ 4.5.2, but it was equally unhelpful:

```
c:\dev>g++ helloworld.png
helloworld.png: file not recognized: File format not recognized
collect2: ld returned 1 exit status
```

5508110: Why is this program erroneously rejected by three C++ compilers?

“I figured that Clang (version 3.0 trunk 127530) must work, since it is so highly praised for its standards conformance. Unfortunately, it didn't even give me one of its pretty, highlighted error messages:

```
c:\dev>clang++ helloworld.png
helloworld.png: file not recognized: File format not recognized
collect2: ld returned 1 exit status
clang++: error: linker (via gcc) command failed with exit code 1
   (use -v to see invocation)
```

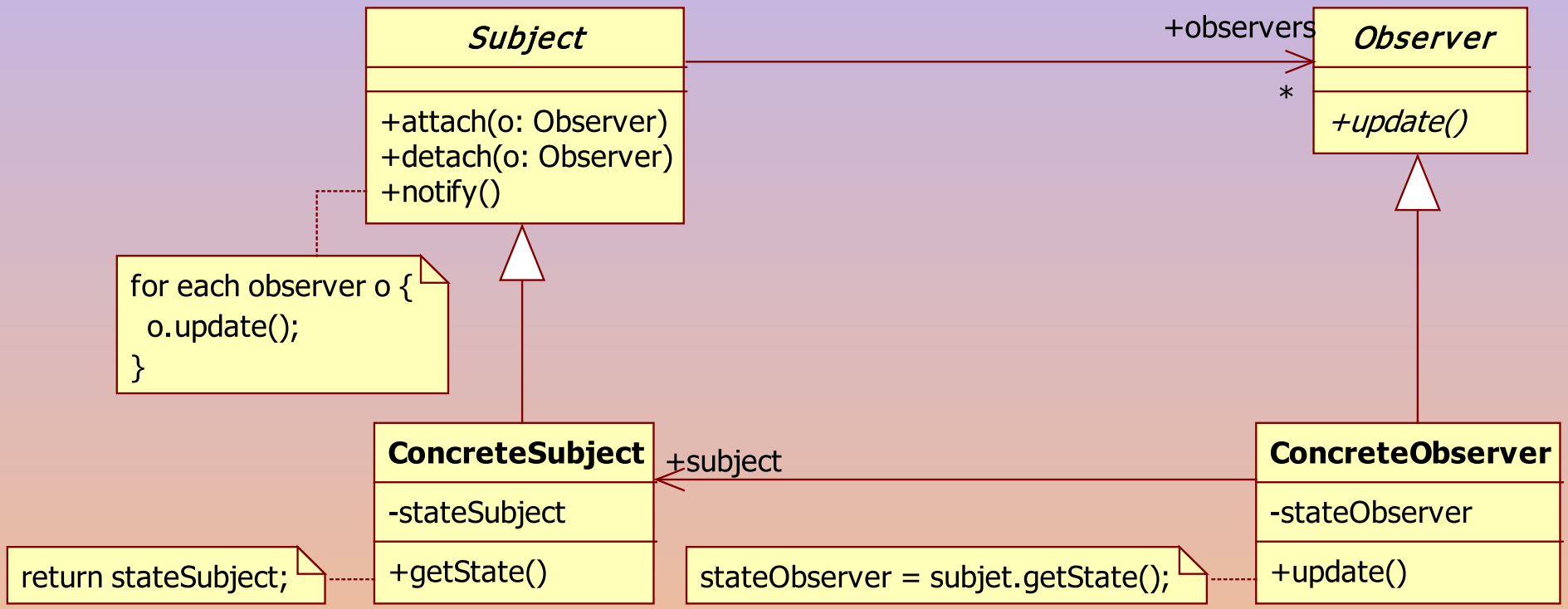
To be honest, I don't really know what any of these error message mean.

Control Flow

Control Flow

Boost.Signal (2)

Observer Design Pattern



Publish/Subscribe

- Loose coupling
- Scalability
- Topic based
- Filtering

Boost.Signals

```
#include <boost/signals2.hpp>
#include <iostream>

                void hello     ()       { std::cout << "Hello, "; }
struct World { void operator()() const { std::cout << "World!"; } };

int main()
{
  boost::signals2::signal<void ()> s;
  s.connect(hello);
  s.connect(World{});
  s.connect([]() { std::cout << std::endl; });
  s.connect(10, []() { std::cout << std::endl; });
  s.connect(0, World{});
  s.connect(0, hello, boost::signals2::at_front);
  s();
}
```

Boost.Signals

```
#include <boost/signals2.hpp>
#include <iostream>

                void hello     ()       { std::cout << "Hello, "; }
struct World { void operator()() const { std::cout << "World!"; } };

int main()
{
  boost::signals2::signal<void ()> s;
  s.connect(hello);
  s.connect(World{});
  s.connect([]() { std::cout << std::endl; });
  s.connect(10, []() { std::cout << std::endl; });
  s.connect(0, World{});
  s.connect(0, hello, boost::signals2::at_front);
  s();
}
```

```
Hello, World!
Hello, World!
```

Boost.Signals: Arguments

```
#include <boost/signals2.hpp>
#include <iostream>
#define ECHO(S) std::cout << #S " = " << S << std::endl;

int main()
{
  boost::signals2::signal<void (float, float)> sig;

  sig.connect([](float x, float y) { ECHO(x + y); });
  sig.connect([](float x, float y) { ECHO(x - y); });
  sig.connect([](float x, float y) { ECHO(x * y); });
  sig.connect([](float x, float y) { ECHO(x / y); });

  sig(5., 3.);
}
```

Boost.Signals: Arguments

```
#include <boost/signals2.hpp>
#include <iostream>
#define ECHO(S) std::cout << #S " = " << S << std::endl;

int main()
{
  boost::signals2::signal<void (float, float)> sig;

  sig.connect([](float x, float y) { ECHO(x + y); });
  sig.connect([](float x, float y) { ECHO(x - y); });
  sig.connect([](float x, float y) { ECHO(x * y); });
  sig.connect([](float x, float y) { ECHO(x / y); });

  sig(5., 3.);
}
```

```
x + y = 8
x - y = 2
x * y = 15
x / y = 1.66667
```

Boost.Signals: Return Value

```
#include <boost/signals2.hpp>
#include <iostream>

int main()
{
  boost::signals2::signal<float (float, float)> sig;

  sig.connect([](float x, float y) { return x + y; });
  sig.connect([](float x, float y) { return x - y; });
  sig.connect([](float x, float y) { return x * y; });
  sig.connect([](float x, float y) { return x / y; });

  std::cout << *sig(5., 3.) << std::endl;
}
```

Boost.Signals: Return Value

```
#include <boost/signals2.hpp>
#include <iostream>

int main()
{
  boost::signals2::signal<float (float, float)> sig;

  sig.connect([](float x, float y) { return x + y; });
  sig.connect([](float x, float y) { return x - y; });
  sig.connect([](float x, float y) { return x * y; });
  sig.connect([](float x, float y) { return x / y; });

  std::cout << *sig(5., 3.) << std::endl;
}
```

```
1.66667
```

Boost.Signals: Combiners

```
int main()
{
  boost::signals2::signal<float (float, float), minmax<float>> sig;

  sig.connect([](float x, float y) { return x + y; });
  sig.connect([](float x, float y) { return x - y; });
  sig.connect([](float x, float y) { return x * y; });
  sig.connect([](float x, float y) { return x / y; });

  std::cout << sig(5., 3.) << std::endl;
}
```

Boost.Signals: Combiners

```
int main()
{
  boost::signals2::signal<float (float, float), minmax<float>> sig;

  sig.connect([](float x, float y) { return x + y; });
  sig.connect([](float x, float y) { return x - y; });
  sig.connect([](float x, float y) { return x * y; });
  sig.connect([](float x, float y) { return x / y; });

  std::cout << sig(5., 3.) << std::endl;
}
```

```
(1.66667, 15)
```

Boost.Signals: Combiners

```
template <typename T>
struct minmax
{
  using result_type = std::tuple<T, T>;

  template <typename InputIterator>
  result_type operator()(InputIterator first, InputIterator last) const
  {
    if (first != last)
      {
        T min = *first++, max = min;
        for (/* empty */; first != last; ++first)
          if (*first < min) min = *first; else
          if (max < *first) max = *first;
        return std::make_tuple(min, max);
      }
    return {};
  }
};
```

Boost.Signals: Combiners: Print Tuples [6245735]

```
template <std::size_t> struct int_{}; // compile-time counter

template <typename Ch, typename Tr, typename Tuple, std::size_t I>
void print_tuple(std::basic_ostream<Ch,Tr>& o, Tuple const& t, int_<I>)
{
  print_tuple(o, t, int_<I-1>()); o << ", " << std::get<I>(t);
}

template <typename Ch, typename Tr, typename Tuple>
void print_tuple(std::basic_ostream<Ch,Tr>& o, Tuple const& t, int_<0>)
{
  o << std::get<0>(t);
}

template <typename Ch, typename Tr, typename... Args>
std::ostream&
operator<<(std::basic_ostream<Ch,Tr>& o, std::tuple<Args...> const& t)
{
  o << "("; print_tuple(o, t, int_<sizeof...(Args)-1>()); return o << ")";
}
```

Boost.Signals: Disconnection

```
boost::signals2::connection c = sig.connect(HelloWorld());
std::cout << "c is connected\n";
sig(); // Prints "Hello, World!"

c.disconnect(); // Disconnect the HelloWorld object
std::cout << "c is disconnected\n";
sig(); // Does nothing: there are no connected slots
```

Boost.Signals: Scoped Connections

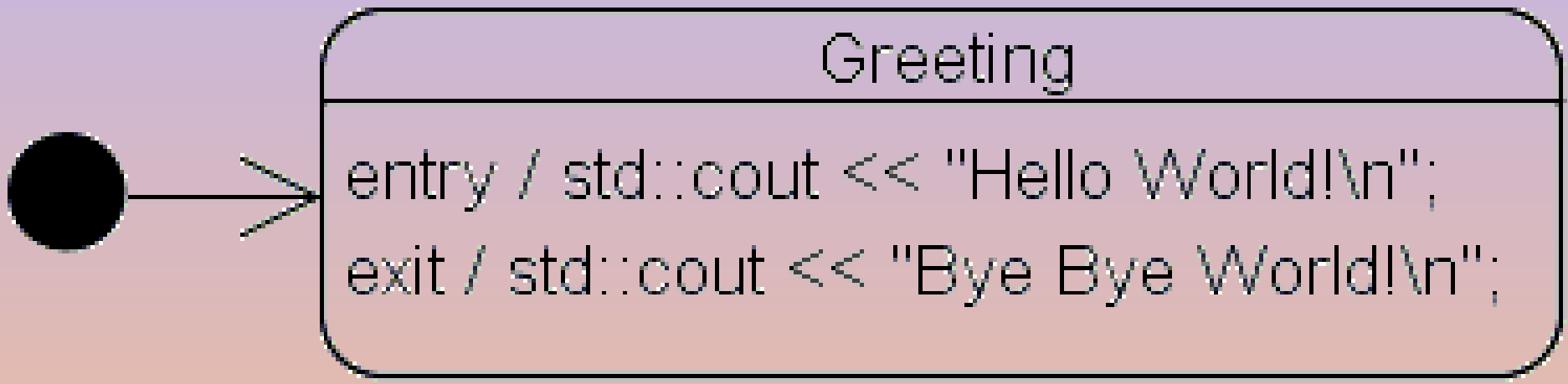
```
{
  boost::signals2::scoped_connection c(sig.connect(ShortLived{}));
  sig(); // will call ShortLived function object
} // scoped_connection goes out of scope and disconnects

sig(); // ShortLived function object no longer connected to sig
```

Boost.Signals: Blocking

```
boost::signals2::connection c = sig.connect(HelloWorld{});
std::cout << "c is not blocked.\n";
sig(); // Prints "Hello, World!"

{
  boost::signals2::shared_connection_block block(c); // block the slot
  std::cout << "c is blocked.\n";
  sig(); // No output: the slot is blocked
} // shared_connection_block going out of scope unblocks the slot
std::cout << "c is not blocked.\n";
sig(); // Prints "Hello, World!"
```

Boost.StateChart

Boost.StateChart: Hello World




```
Hello World!
Bye Bye World!
```

Boost.StateChart: Hello World

```
namespace sc = boost::statechart;

// Forward-declare the initial state.
struct Greeting;

// Boost.Statechart makes heavy use of the CRT pattern.
struct Machine : sc::state_machine< Machine, Greeting > {};

struct Greeting : sc::simple_state< Greeting, Machine > {
  // Being in a state = its object is alive.
   Greeting() { std::cout << "Hello World!\n"; }
  ~Greeting() { std::cout << "Bye Bye World!\n"; }
};

int main()
{
  Machine myMachine;
  myMachine.initiate();
  return 0;
}
```

Digression: Curiously Recurring Template Pattern

```
template <class Derived>
class Base
{
  // Methods within Base can access members of Derived.
};

class Derived : public Base<Derived>
{
  // ...
};
```

[WCuriously_recurring_template_pattern]

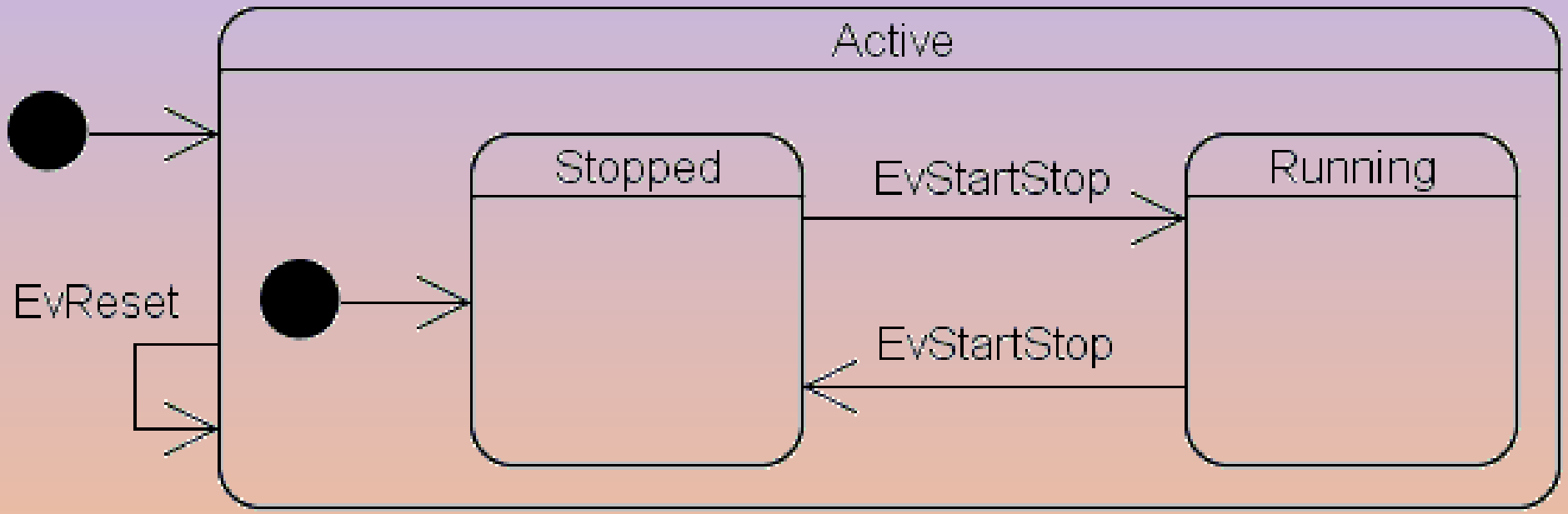
Digression: CRT For Static Polymorphism

```
template <class Derived>
struct Base
{
  void interface() {
    // ...
    static_cast<Derived*>(this)->implementation();
    // ...
  }
  static void static_func() {
    // ...
    Derived::static_sub_func();
    // ...
  }
};

struct Derived : Base<Derived>
{
  void implementation();
  static void static_sub_func();
};
```

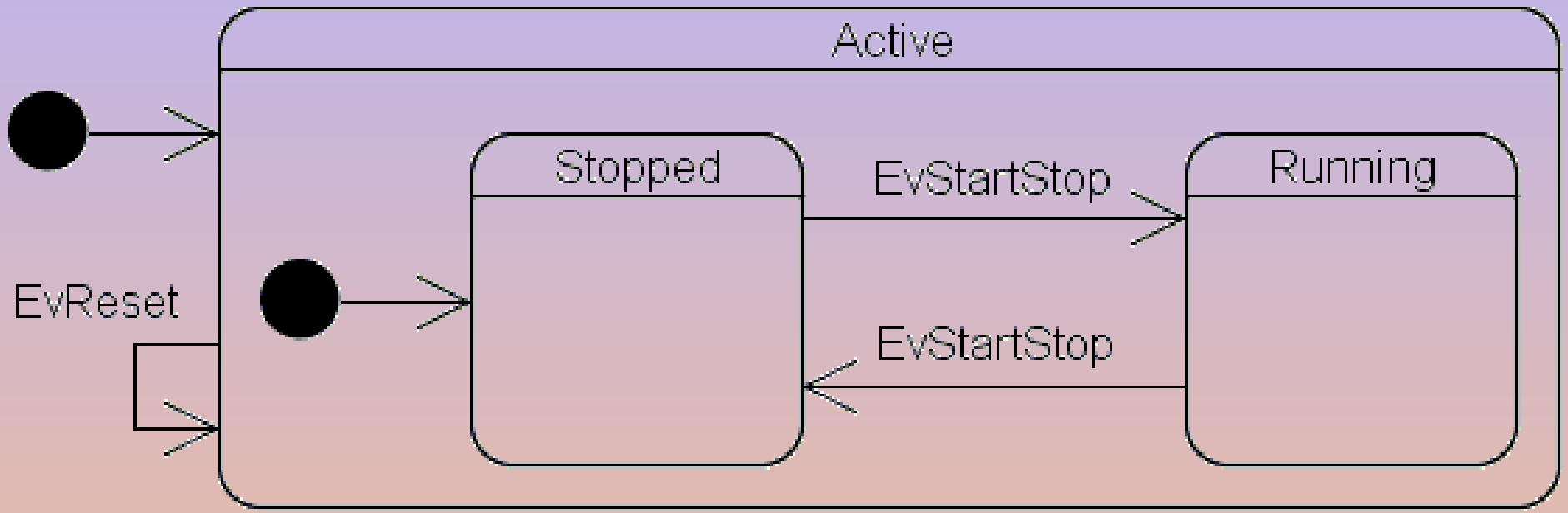
Digression: CRT For Polymorphic Copy Construction

```
class Shape
{
public:
  virtual ~Shape() {}
  virtual Shape *clone() const = 0;
};

template <typename Derived>
class Shape_CRTP : public Shape
{
public:
  virtual Shape *clone() const
  {
    return new Derived(static_cast<Derived const&>(*this));
  }
};

class Square: public Shape_CRTP<Square> {};
class Circle: public Shape_CRTP<Circle> {};
```

Boost.StateChart: Stop Watch

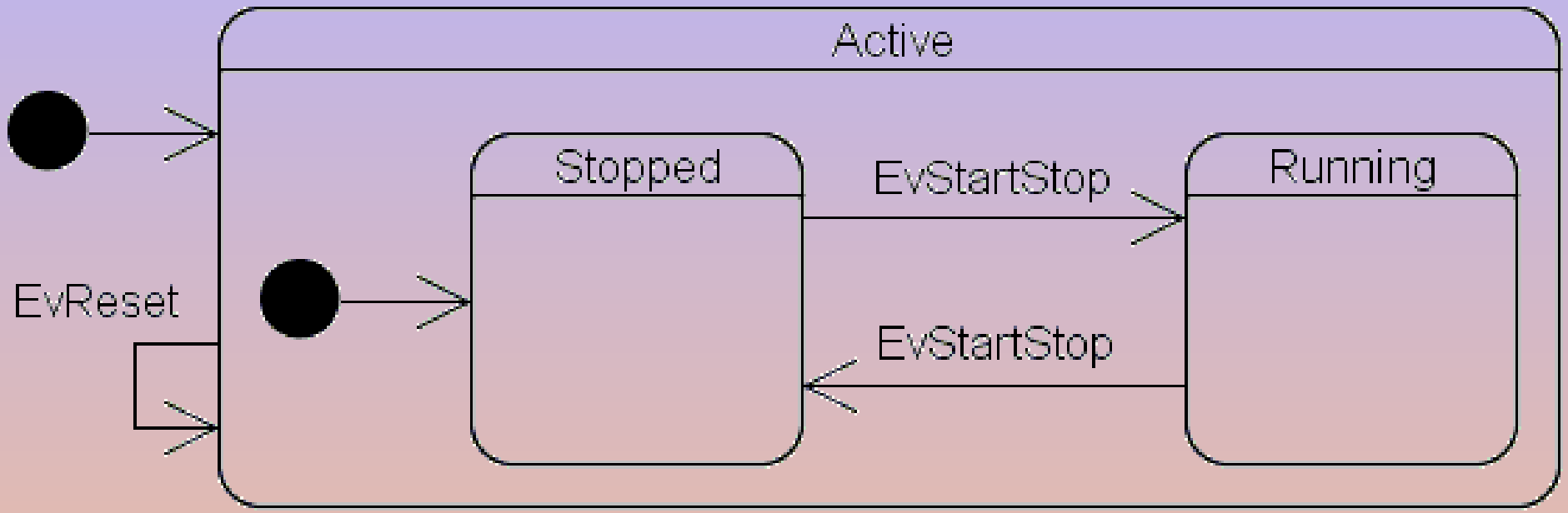


Boost.StateChart: Stop Watch: Events



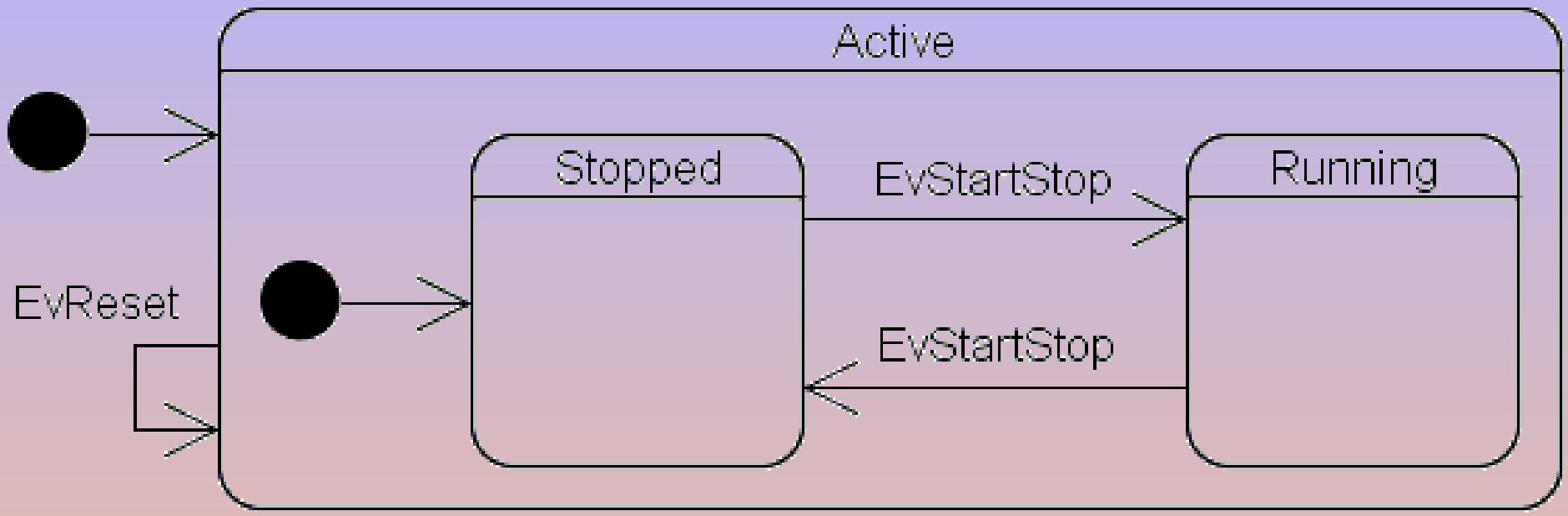

```
namespace sc = boost::statechart;

struct EvStartStop : sc::event< EvStartStop > {};
struct EvReset     : sc::event< EvReset >     {};
```

Boost.StateChart: Stop Watch: Machine



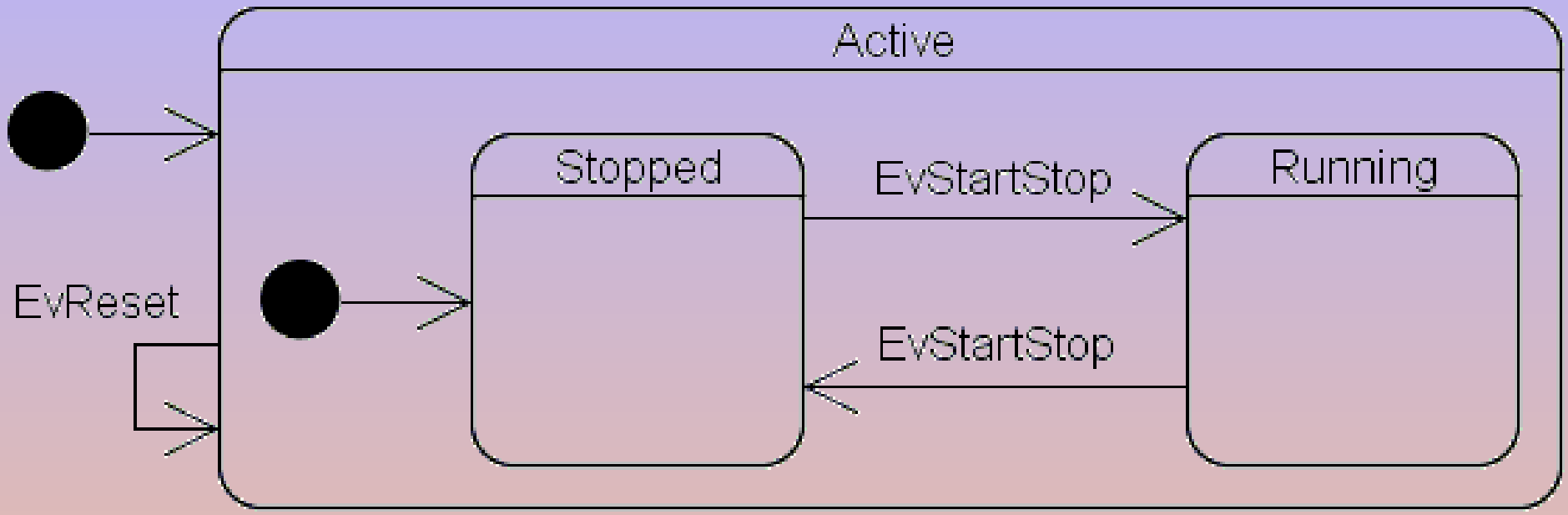

```
struct Active;
struct Running;
struct Stopped;

struct StopWatch : sc::state_machine< StopWatch, Active > {};
```

Boost.StateChart: Stop Watch: Active State



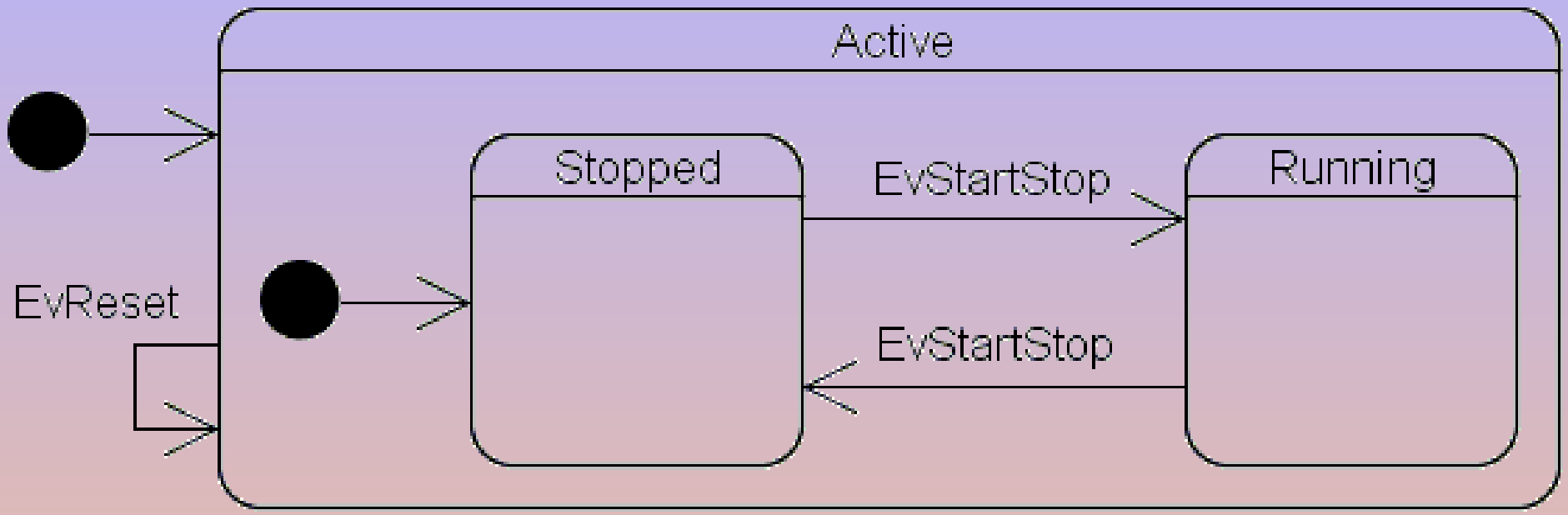

```
struct Active : sc::simple_state< Active, StopWatch, Stopped >
{
  using reactions = sc::transition< EvReset, Active >;

  double & elapsed()       { return elapsedTime_; }
  double   elapsed() const { return elapsedTime_; }

private:
  double elapsedTime_ = 0.0;
```

Boost.StateChart: Stop Watch: Stopped State




```
struct IElapsedTime { virtual double elapsed() const = 0; };

struct Stopped : IElapsedTime, sc::simple_state< Stopped, Active >
{
  using reactions = sc::transition< EvStartStop, Running >;
  virtual double elapsed() const override
    { return context<Active>().elapsed(); }
};
```

Boost.StateChart: Stop Watch: Running State



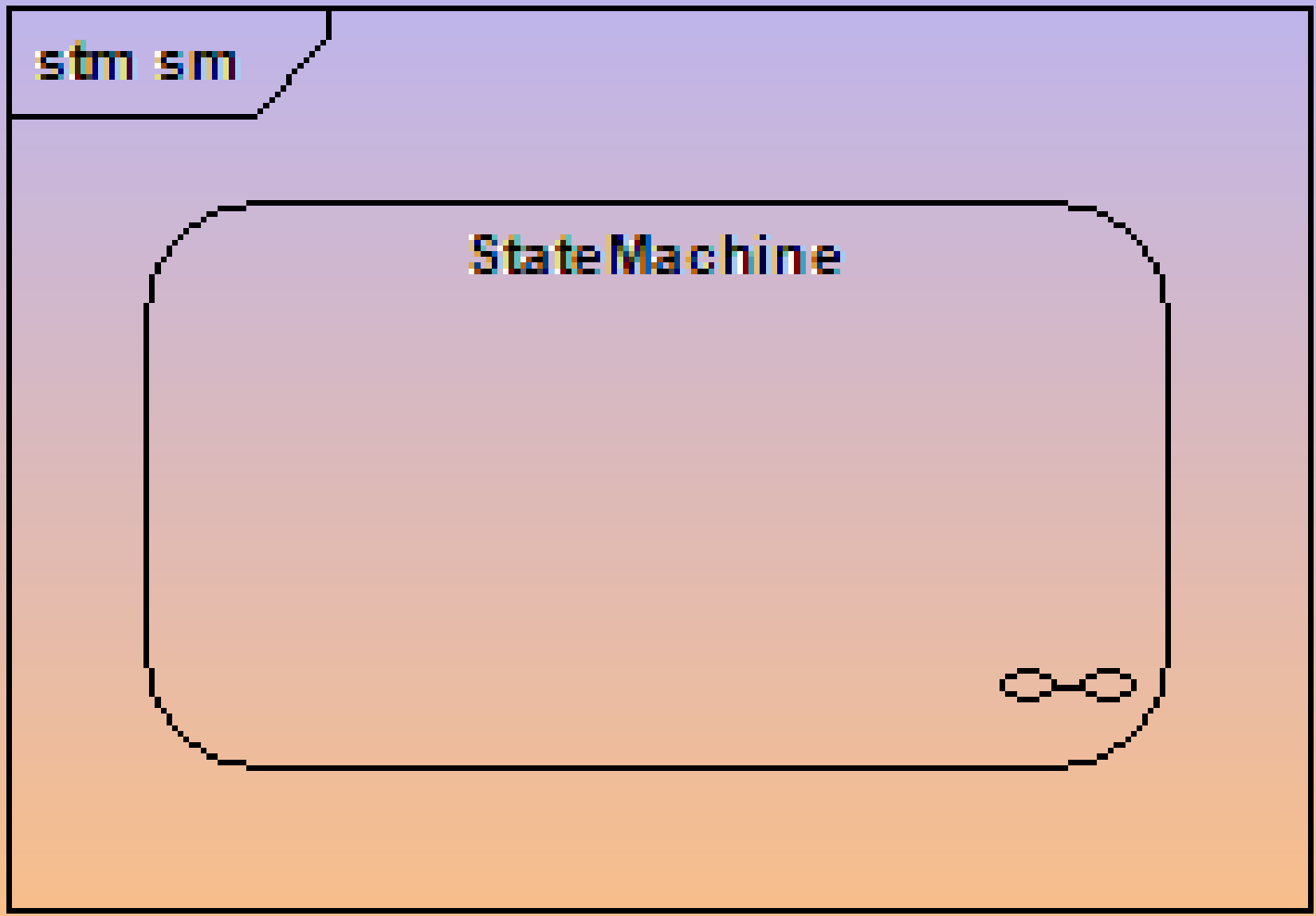

```
struct Running : IElapsedTime, sc::simple_state< Running, Active > {
  using reactions = sc::transition< EvStartStop, Stopped >;

  ~Running() { context<Active>().elapsed() = elapsed(); }

  virtual double elapsed() const override
  { return context<Active>().elapsed() + std::difftime(0, startTime_); }

private:
```

Boost.StateChart: Stop Watch: main

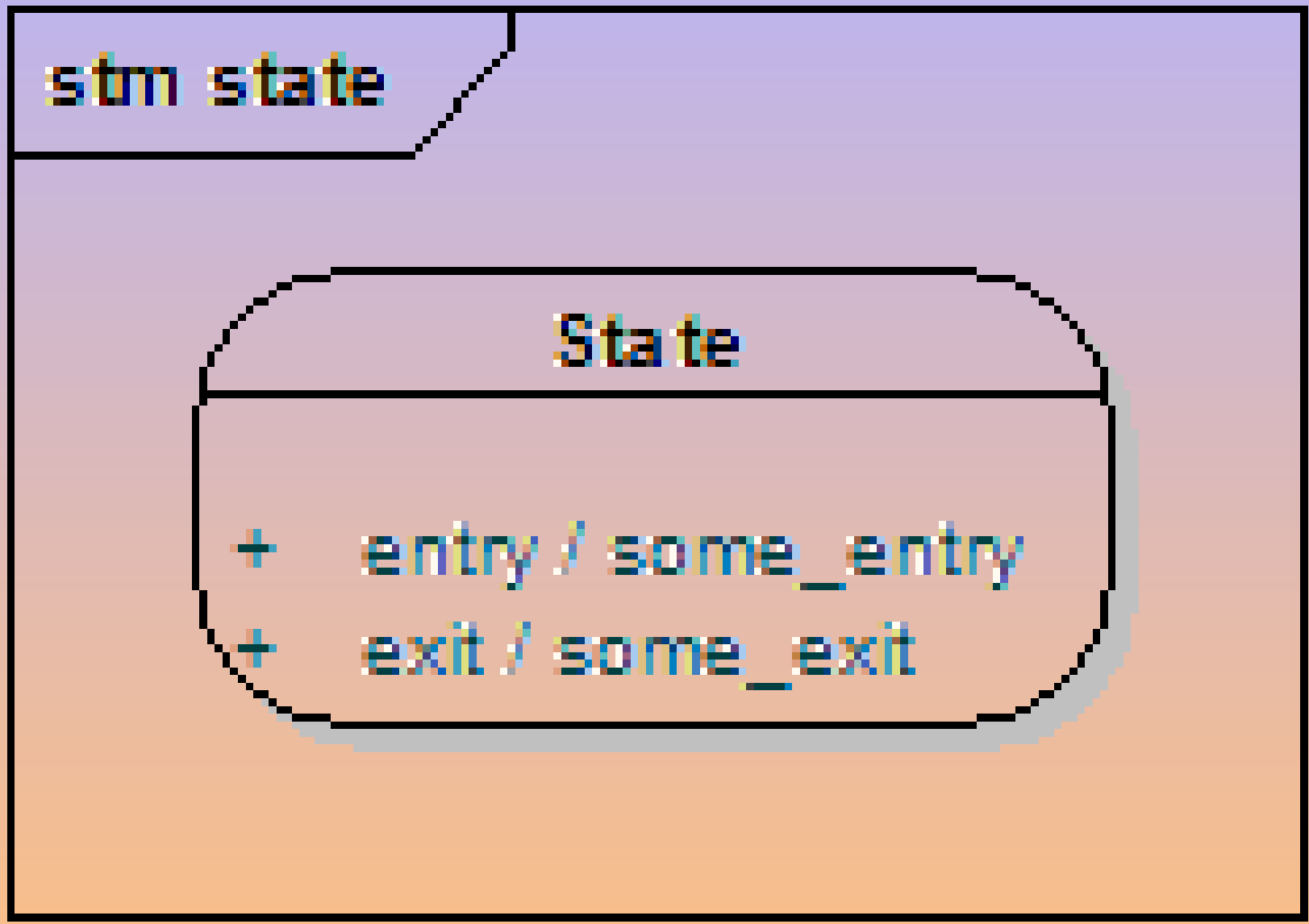
```
int main()
{
  StopWatch stopWatch;
  stopWatch.initiate();

  while (true)
    switch (([]() { char r; std::cin >> r; return r; })())
    {
    case 'r': stopWatch.process_event(EvReset{}); break;

    case 's': stopWatch.process_event(EvStartStop{}); break;

    case 'd':
      std::cout << "Elapsed time: "
        << stopWatch.state_cast<const IElapsedTime&>().elapsed() << "\n";
      break;

    default: return 0;
    }
}
```

Boost.MetaStateMachine (MSM)

Boost.MSM: Machine

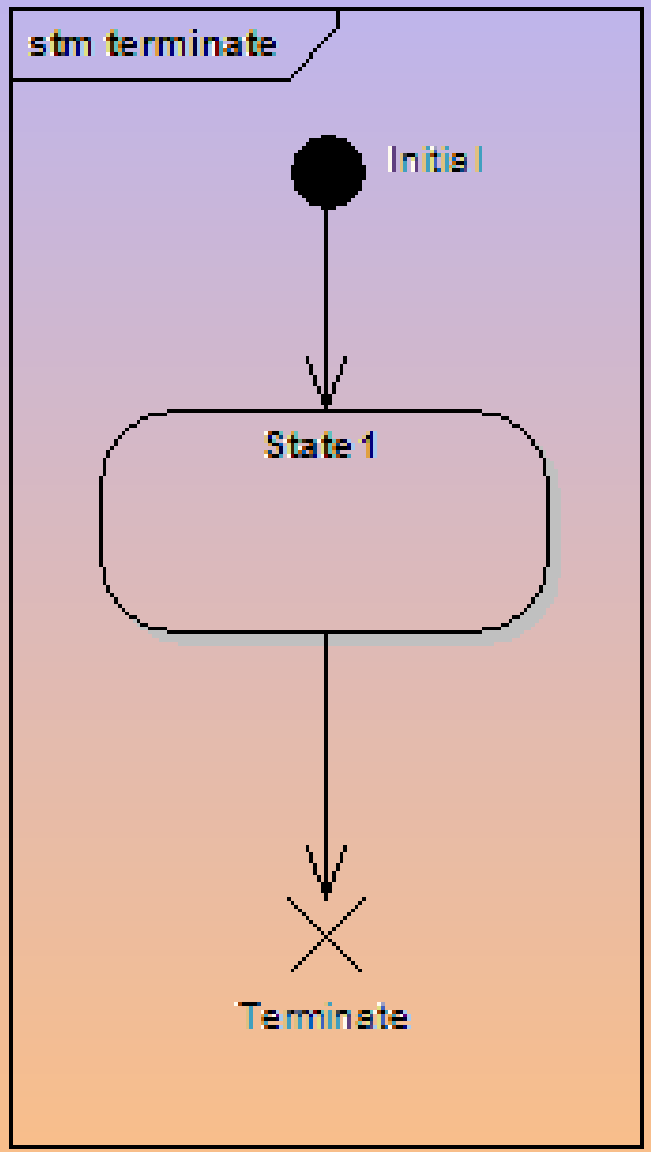


Boost.MSM: State

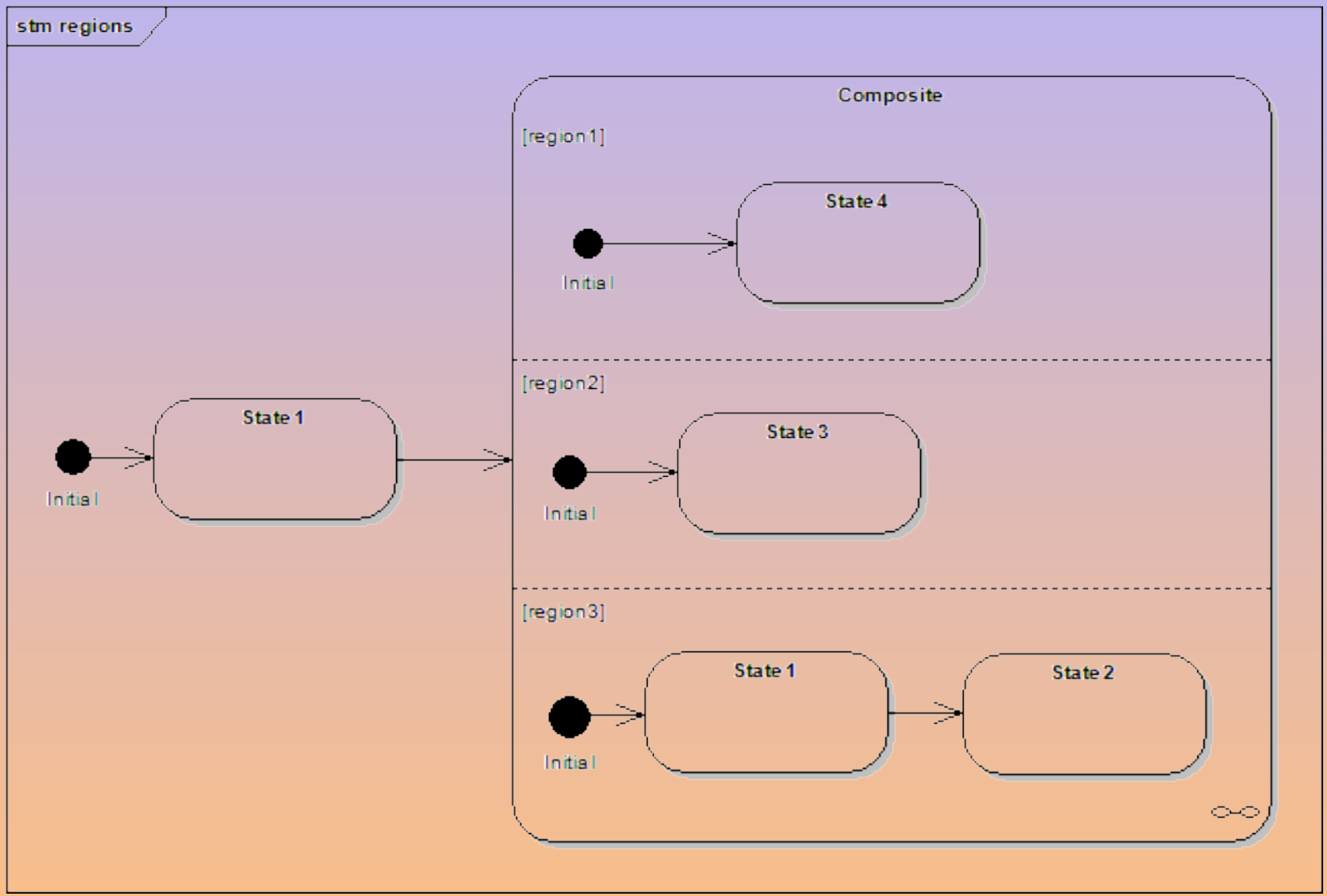


Boost.MSM: Transition

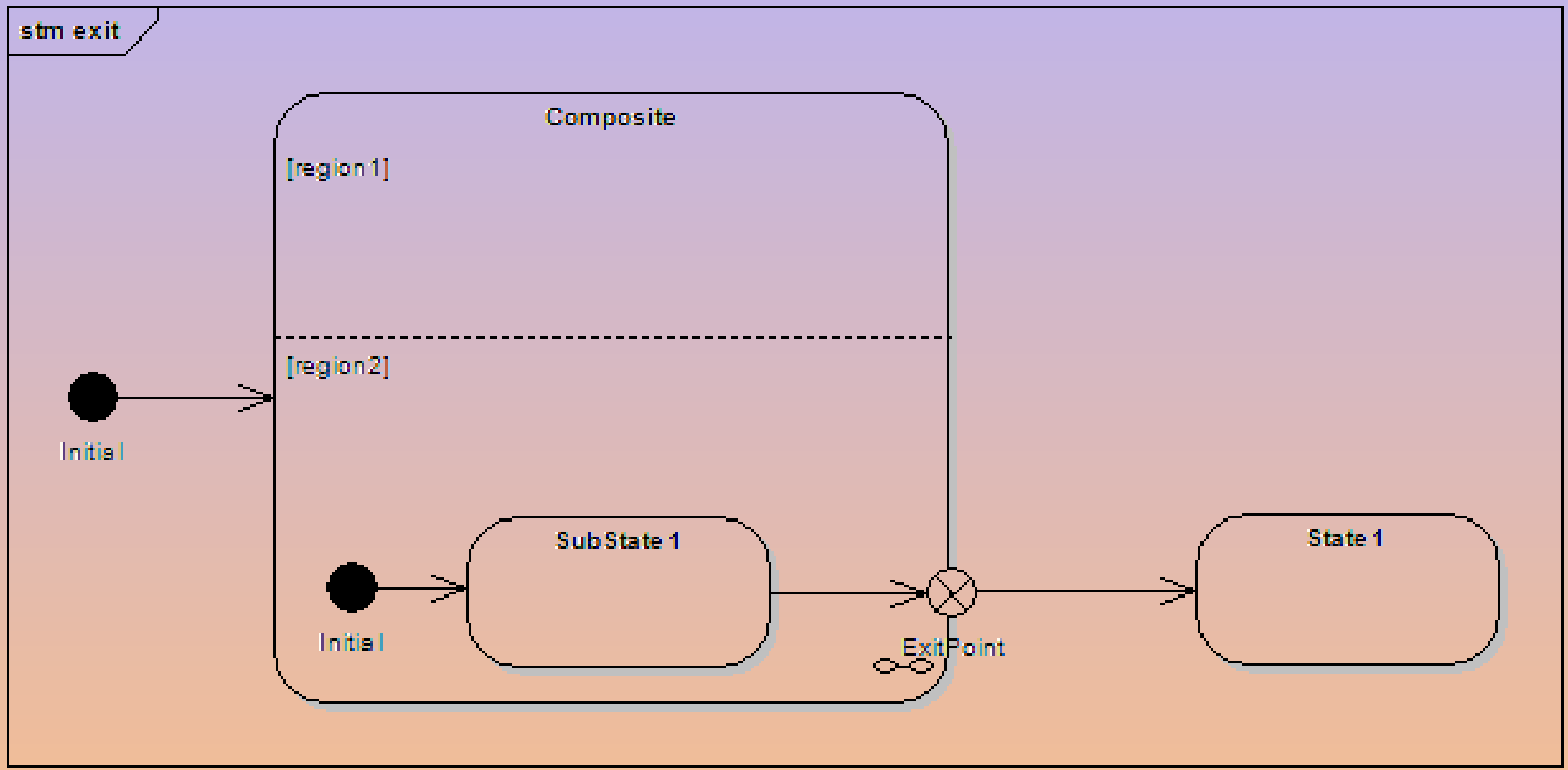
Boost.MSM: Initial/Final State



Boost.MSM: Sub-Machines

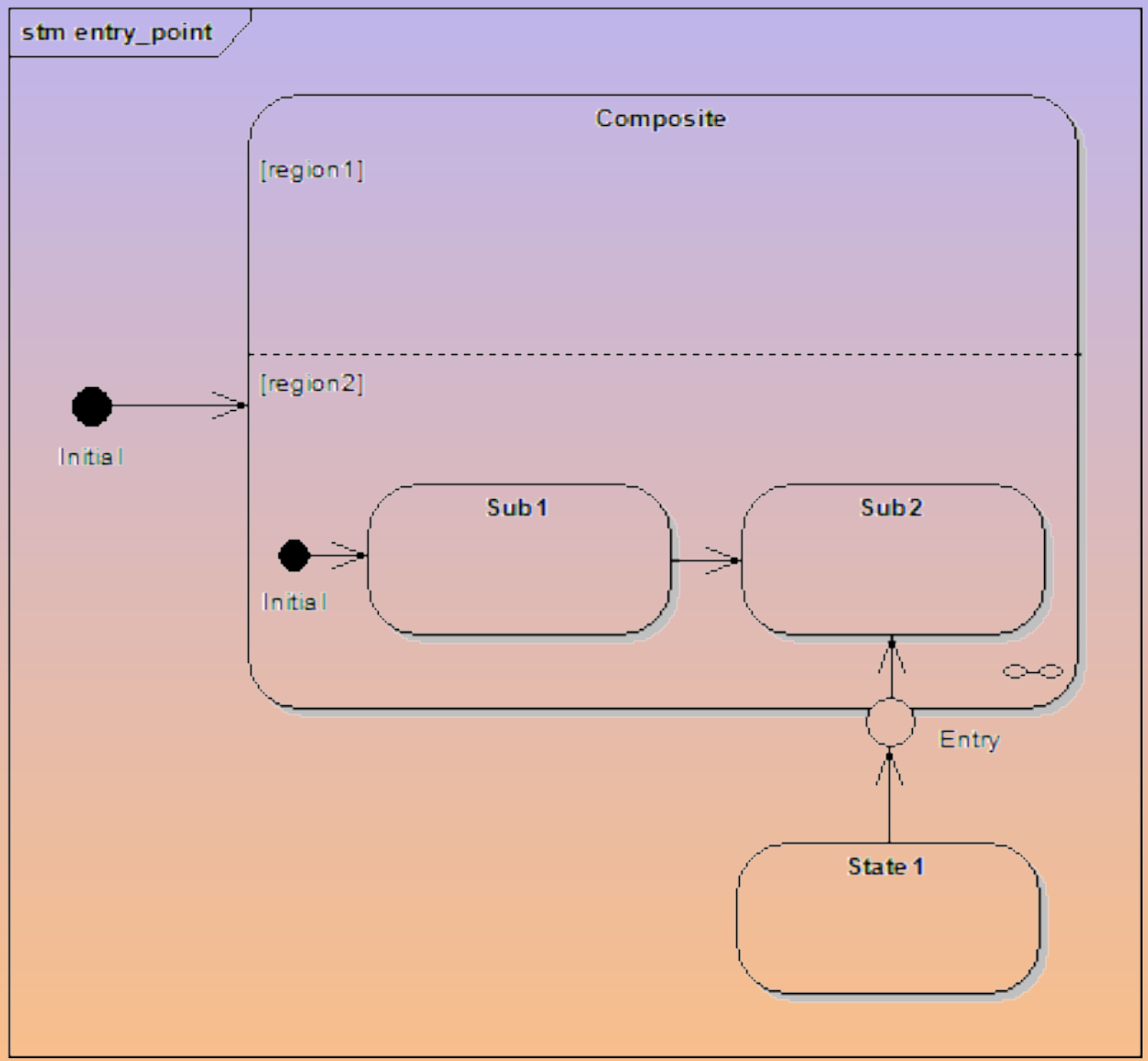


Boost.MSM: Sub-Machines Exit

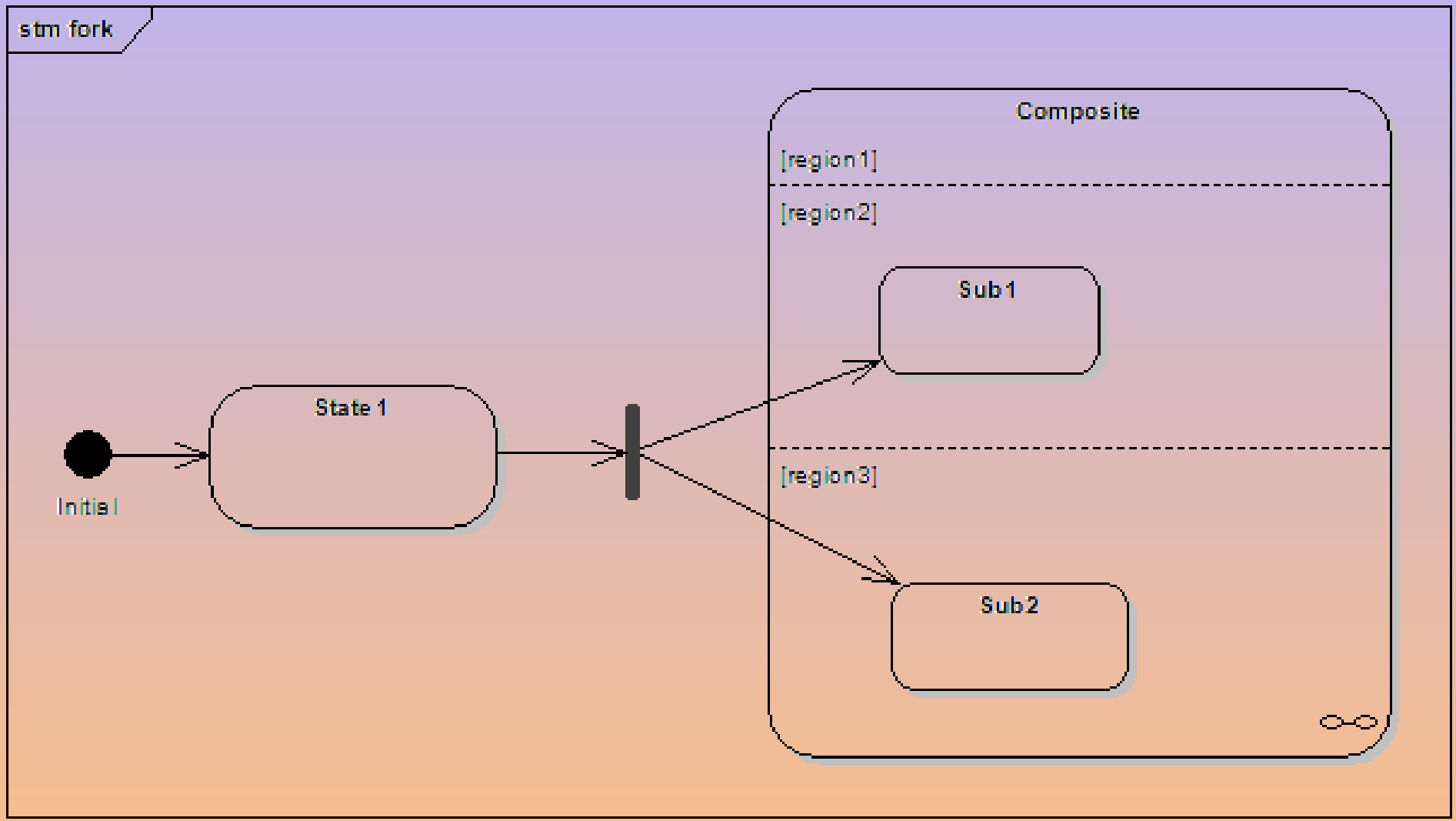


Forces termination of all the contained machines.

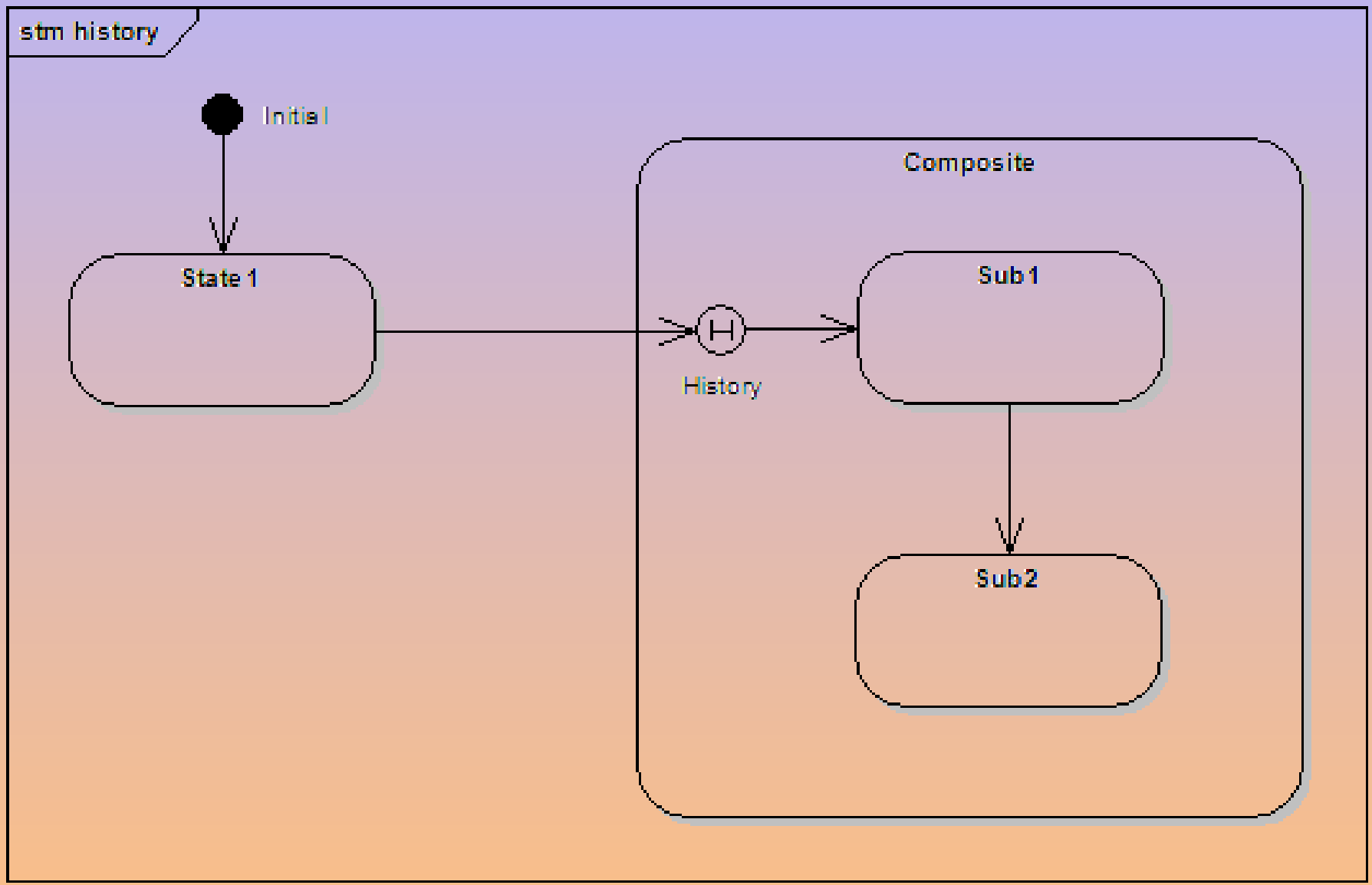
Boost.MSM: Sub-Machines Entry-point



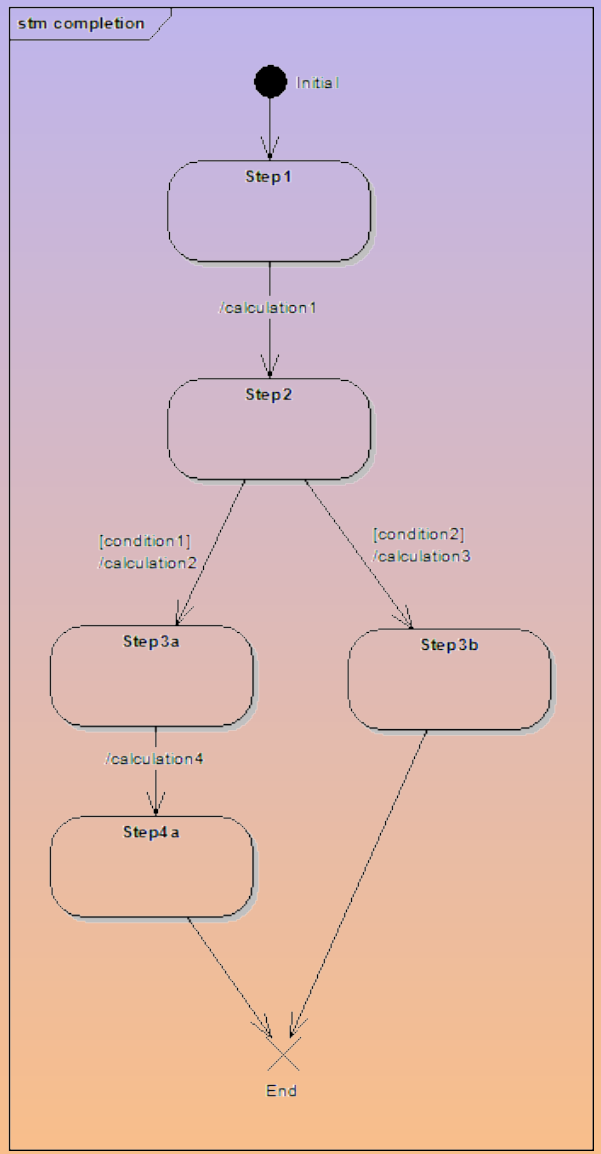
Boost.MSM: Sub-Machines Fork



Boost.MSM: Sub-Machines History



Boost.MSM: Spontaneous Transitions



Boost.MSM: CD Player: Basic Front-End

```
struct transition_table : mpl::vector<
//      Start    Event         Target    Action                 Guard
//    +-------+-----------+--------+--------------------+----------------------+
a_row<Stopped , play       , Playing, &CDP::start_playback                      >,
a_row<Stopped , open_close, Open   , &CDP::open_drawer                          >,
 _row<Stopped , stop       , Stopped                                            >,
//    +-------+-----------+--------+--------------------+----------------------+
a_row<Open    , open_close, Empty  , &CDP::close_drawer                         >,
//    +-------+-----------+--------+--------------------+----------------------+
a_row<Empty   , open_close, Open   , &CDP::open_drawer                          >,
  row<Empty   ,cd_detected, Stopped, &CDP::store_cd_info, &CDP::good_disk_format>,
  row<Empty   ,cd_detected, Playing, &CDP::store_cd_info, &CDP::auto_start      >,
//    +-------+-----------+--------+--------------------+----------------------+
a_row<Playing , stop       , Stopped, &CDP::stop_playback                       >,
a_row<Playing , pause      , Paused , &CDP::pause_playback                      >,
a_row<Playing , open_close, Open   , &CDP::stop_and_open                        >,
//    +-------+-----------+--------+--------------------+----------------------+
a_row<Paused  , end_pause  , Playing, &CDP::resume_playback                     >,
a_row<Paused  , stop       , Stopped, &CDP::stop_playback                       >,
a_row<Paused  , open_close, Open   , &CDP::stop_and_open                        >
//    +-------+-----------+--------+--------------------+----------------------+
> {};
```

Boost.MSM: CD Player: Functor Front-End

```
struct transition_table : mpl::vector<
  //  Start     Event         Target     Action           Guard
  // +-------+------------+--------+----------------+------------------+
  Row<Stopped, play        , Playing, start_playback , none             >,
  Row<Stopped, open_close  , Open   , open_drawer    , none             >,
  Row<Stopped, stop        , Stopped, none           , none             >,
  // +-------+------------+--------+----------------+------------------+
  Row<Open   , open_close  , Empty  , close_drawer   , none             >,
  // +-------+------------+--------+----------------+------------------+
  Row<Empty  , open_close  , Open   , open_drawer    , none             >,
  Row<Empty  , cd_detected, Stopped, store_cd_info   , good_disk_format >,
  Row<Empty  , cd_detected, Playing, store_cd_info   , auto_start       >,
  // +-------+------------+--------+----------------+------------------+
  Row<Playing, stop        , Stopped, stop_playback  , none             >,
  Row<Playing, pause       , Paused , pause_playback , none             >,
  Row<Playing, open_close  , Open   , stop_and_open  , none             >,
  // +-------+------------+--------+----------------+------------------+
  Row<Paused , end_pause   , Playing, resume_playback, none             >,
  Row<Paused , stop        , Stopped, stop_playback  , none             >,
  Row<Paused , open_close  , Open   , stop_and_open  , none             >
  // +-------+------------+-m------+----------------+------------------+
> {};
```

Boost.MSM: CD Player: Functor Front-End

```
struct store_cd_info
{
  template <class Fsm,class Evt,class SourceState,class TargetState>
  void operator()(Evt const&, Fsm& fsm, SourceState&,TargetState& )
  {
    std::cout << "player::store_cd_info" << std::endl;
    fsm.process_event(play());
  }
};
```

Boost.MSM: eUML Front-End

A transition can be defined using eUML as:

```
source + event [guard] / action == target
```

or as

```
target == source + event [guard] / action
```

Boost.MSM: CD Player: eUML Front-End

```
BOOST_MSM_EUML_TRANSITION_TABLE((
Stopped + play [some_guard] / (some_action , start_playback)  == Playing,
Stopped + open_close/ open_drawer                             == Open    ,
Stopped + stop                                                == Stopped,
Open    + open_close / close_drawer                           == Empty   ,
Empty   + open_close / open_drawer                            == Open    ,
Empty   + cd_detected [good_disk_format] / store_cd_info      == Stopped
),transition_table)
```

MSM vs. StateChart

MSM

- much faster
- no RTTI, does not require anything virtual
- has a more complete UML2 support
- offers better front-ends
- tough on compilation

StateChart

- simpler
- spread the layout over multiple translation units (states, transitions)

Misc

Misc

Boost.IOState

Boost.IOState

```
#include <ostream>
#include <ios>

void hex(std::ostream &os, char byte)
{
  os << std::hex << static_cast<unsigned>(byte);
}
```

Boost.IOState

```
#include <boost/io/ios_state.hpp>
#include <ios>
#include <iostream>
#include <ostream>

void hex(std::ostream &os, char byte)
{
  boost::io::ios_flags_saver ifs(os);
  os << std::hex << static_cast<unsigned>(byte);
}
```

Boost.IOState

```
int  main()
{
  {
    boost::io::ios_all_saver ias(std::cout);
    hex(std::cout, 'A');
  }

  //...

  {
    boost::io::ios_all_saver ias(std::cerr);
    hex(std::cerr, 'b');
    ias.restore();
    hex(std::cerr, 'C');
 }
}
```

BCP

Boost 1.58.0: 131 libraries

Accumulators, Algorithm, Align, Any, Array, Asio, Assert, Assign, Atomic, Bimap, Bind, Call Traits, Chrono, Circular Buffer, Compatibility, Compressed Pair, Concept Check, Config, Container, Context, Conversion, Core, Coroutine, CRC, Date Time, Dynamic Bitset, Enable If, Endian, Exception, Filesystem, Flyweight, Foreach, Format, Function, Function Types, Functional, Functional/Factory, Functional/Forward, Functional/Hash, Functional/Overloaded Function, Fusion, Geometry, GIL, Graph, Heap, ICL, Identity Type, In Place Factory, Typed In Place Factory, Integer, Interprocess, Interval, Intrusive, IO State Savers, Iostreams, Iterator, Lambda, Lexical Cast, Local Function, Locale, Lockfree, Log, Math, Math Common Factor, Math Octonion, Math Quaternion, Math/Special Functions, Math/Statistical Distributions, Member Function, Meta State Machine, Min-Max, Move, MPI, MPL, Multi-Array, Multi-Index, Multiprecision, Numeric Conversion, Odeint, Operators, Optional, Parameter, Phoenix, Pointer Container, Polygon, Pool, Predef, Preprocessor, Program Options, Property Map, Property Tree, Proto, Python, Random, Range, Ratio, Rational, Ref, Regex, Result Of, Scope Exit, Serialization, Signals2, Smart Ptr, Sort, Spirit, Statechart, Static Assert, String Algo, Swap, System, Test, Thread, ThrowException, Timer, Tokenizer, Tribool, TTI, Tuple, Type Erasure, Type Index, Type Traits, Typeof, uBLAS, Units, Unordered, Utility, Uuid, Value Initialized, Variant, Wave, Xpressive

Boost 1.58.0: 131 libraries

Accumulators, Algorithm, Align, Any, Array, Asio, Assert, Assign, Atomic, Bimap, Bind, Call Traits, Chrono, Circular Buffer, Compatibility, Compressed Pair, Concept Check, Config, Container, Context, Conversion, Core, Coroutine, CRC, Date Time, Dynamic Bitset, Enable If, Endian, Exception, Filesystem, Flyweight, Foreach, Format, Function, Function Types, Functional, Functional/Factory, Functional/Forward, Functional/Hash, Functional/Overloaded Function, Fusion, Geometry, GIL, Graph, Heap, ICL, Identity Type, In Place Factory, Typed In Place Factory, Integer, Interprocess, Interval, Intrusive, IO State Savers, Iostreams, Iterator, Lambda, Lexical Cast, Local Function, Locale, Lockfree, Log, Math, Math Common Factor, Math Octonion, Math Quaternion, Math/Special Functions, Math/Statistical Distributions, Member Function, Meta State Machine, Min-Max, Move, MPI, MPL, Multi-Array, Multi-Index, Multiprecision, Numeric Conversion, Odeint, Operators, Optional, Parameter, Phoenix, Pointer Container, Polygon, Pool, Predef, Preprocessor, Program Options, Property Map, Property Tree, Proto, Python, Random, Range, Ratio, Rational, Ref, Regex, Result Of, Scope Exit, Serialization, Signals2, Smart Ptr, Sort, Spirit, Statechart, Static Assert, String Algo, Swap, System, Test, Thread, ThrowException, Timer, Tokenizer, Tribool, TTI, Tuple, Type Erasure, Type Index, Type Traits, Typeof, uBLAS, Units, Unordered, Utility, Uuid, Value Initialized, Variant, Wave, Xpressive

Using Boost

- Use Benoît Sigoure's 'boost.m4' Autoconf's macro
- Rely on the distribution
- Install it by hand
- BoostPro is discontinued
- Try bcp

bcp

```
$ mkdir foo
$ bcp boost/regex.hpp foo
```

bcp

```
$ mkdir foo
$ bcp boost/regex.hpp foo
```

```
$ bcp regex foo
```

bcp

```
$ mkdir foo
$ bcp boost/regex.hpp foo
```

```
$ bcp regex foo
```

```
$ bcp scoped_ptr regex --namespace=myboost myboost
```

bcp

```
$ mkdir foo
$ bcp boost/regex.hpp foo
```

```
$ bcp regex foo
```

```
$ bcp scoped_ptr regex --namespace=myboost myboost
```

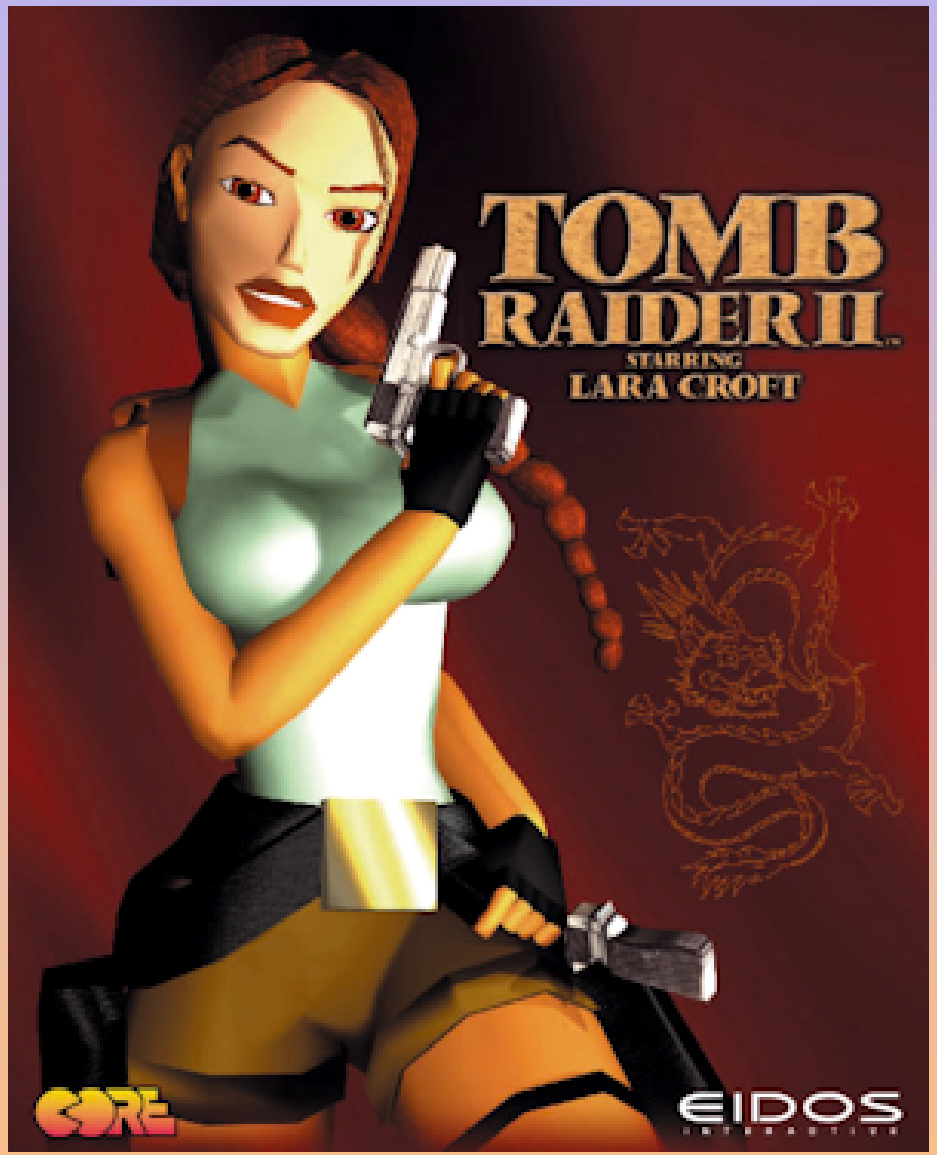
```
$ git ls-files -z | xargs -0 \
    bcp --report --namespace=myboost --namespace-alias --scan {} myboost
```

Other Libraries

Other Libraries

TR2

TR2



TR2: Request for Proposals (2005)

Special interest in Unicode, XML/HTML, Networking and usability for novice programmers.

- Threads
- The Asio C++ library (networking).
- Signals/Slots
- Filesystem Library (Boost.Filesystem)
- Boost.Any
- Lexical conversion library
- New string algorithms
- More complete taxonomy of algebraic properties for numeric libraries
- Heterogeneous comparison lookup for associative containers

Google Tools

Google Tools

- googletest
- glog
- gflags
- sparsehash

Folly

Folly: Facebook Open-source LibrarY

Open-source C++ library developed and used at Facebook.

`https://github.com/facebook/folly`

`https://groups.google.com/d/forum/facebook-folly`

Folly: dynamic (Tailored for JSON)

```
dynamic twelve = 12;
dynamic str = "string";
dynamic nul = nullptr;
dynamic boolean = false;

// Arrays can be initialized with brackets.
dynamic array = { "array ", "of ", 4, " elements" };
assert(array.size() == 4);
dynamic emptyArray = {};
assert(array.empty());

// Maps from dynamics to dynamics are called objects.  The
// dynamic::object constant is how you make an empty map from dynamics
// to dynamics.
dynamic map = dynamic::object;
map["something"] = 12;
map["another_something"] = map["something"] * 2;

// Dynamic objects may be intialized this way.
dynamic map2 = dynamic::object("something", 12)("another_something", 24);
```

Folly: Strings

- 100% compatible with `std::string`.
- Thread-safe reference counted copy-on-write for strings large strings.
- Uses malloc.
- Jemalloc-friendly.
- `find`: 30x speed for successful searches and 1.5x for failed ones (Boyer-Moore)
- Offers conversions to and from `std::string`.

Folly: Vectors

“Simply replacing `std::vector` with `folly::fbvector` will improve the performance of your C++ code using vectors with common coding patterns.

The improvements are:

- *always non-negative,*
- *almost always measurable,*
- *frequently significant,*
- *sometimes dramatic,*
- *and occasionally spectacular.*

Folly: Format

```
std::cout << format("The answers are {} and {}", 23, 42);
// => "The answers are 23 and 42"

std::cout << format("The answers are {1}, {0}, and {1} again", 23, 42);
// => "The answers are 42, 23, and 42 again"

std::vector<int> v {23, 42};
std::map<std::string, std::string> m1{ {"what", "answer"} };
std::cout << format("The only {1[what]} is {0[1]}", v, m1);
// => "The only answer is 42"

std::map<std::string, std::string> m2{{"what","answer"},{"value","42"}};
std::cout << vformat("The only {what} is {value}", m2);
// => "The only answer is 42"

std::tuple<int, std::string, int> t {42, "hello", 23};
std::cout << vformat("{0} {2} {1}", t);
// => "42 23 hello"
```

Folly: Format

```
// "X<10": fill with 'X', left-align ('<'), width 10
std::cout << format("{:X<10} {}", "hello", "world");
// => "helloXXXXX world"

// Format supports printf-style format specifiers
std::cout << format("{0:05d} decimal = {0:04x} hex", 42);
// => "00042 decimal = 002a hex"
```


TERRA INCOGNITA
programmers eat each other here
Dreadful Random Memory Corruption Dragon of Doom
REWARD
Obelisk of Constexpr
C++11 border
λ
Notorious Gates of Anonymity
Nest of the Cute long long
auto
C++98 border
final
private
Data stronghold
protected
public
friend
override
Diamond mine
Virtual inheritance mountains
Ancient structure with class circle
Multiple inheritance forest
Waterfall of destruction
Little Lost Goto
Inheritance forest
Sutter's Mill
Blame Stroustrup tower
Type casting ghosts dance on the meadow
THE HOLY STANDARD TERRITORY
ISO/IEC 14882:2003
ISO/IEC 14882:1998
ISO/IEC 14882:2011
Pure virtual river
Bugs live here there and everywhere
Bitwise canyon
Memory managment desert
Castle of a Dynamic Memory
The Epic Heap Memory Allocation Siege
New
Delete
Heap
Quicksands. Beware!
not enough memory
access violation
memory leaks
malloc Oasis
free Oasis
realloc
#include fort
Exceptional swamps
try
throw
catch
unexpected
terminate
Curiously recurring template Ouroboros
Templates maze
Alexandrescu castle
Loki
std::thread
std::rope
std::unordered_set
std::power
STL
std::unordered_map
The Great New Standard War
std::pair
std::map
std::multimap
ios lake
Ruins of std::auto_ptr
istream
iostream
ostream
std::list
std::vector
boost::graph
boost::gregorian
boost::bimap
BOOST
boost::asio
boost::shared_ptr
boost::tuple
boost::regex
Concepts Shipwreck
WTL
external libraries seas
Qt
C++/Tk
ACE
The C++ Lands
Its amazing creatures and weird beasts
2012 edition
explorer
Alena (http://alenacpp.blogspot.com)
cartographer
Jim (http://jimblog.me)

Boost, Part II

Questions?

1. Containers
2. Conversions
3. Control Flow
4. Misc
5. Other Libraries