The Art of Writing Reasonable Concurrent Code

Pre-Conference Workshop ACCU 2017

©2017

Felix Petriconi

2017-04-25

The [C++] language is too large for anyone to master

The [C++] language is too large for anyone to master So everyone lives within a subset

The [C++] language is too large for *anyone* to master So *everyone* lives within a subset

Sean Parent, C++Now, 2012

- School (UCSD Pascal, Turbo Pascal)
- ▶ Studied electrical engineering (Modula 2, Ada, C++)
- ▶ Student research assistant (1992-1996) (Turbo Pascal, C++, C)
- ► Freelance programmer 1996-2003 (Ericsson, Siemens-VDO, etc.)
 - Development of test software for embedded devices (Perl, C)
- ▶ Programmer and development manager 2003-today at MeVis Medical Solutions AG. Bremen. Germany
 - ▶ Development of medical devices in the area of mammography and radio therapy (C++, Ruby, Python)
- Programming activities:
 - ▶ Blog editor of ISO C++ website
 - ► Active member of C++ User Group Bremen
 - Contributor to Sean Parent's concurrency library
 - Member of ACCU conference committee
- Married with Nicole, having three children, living near Bremen, Germany
- ▶ Other interests: Classic film scores, composition

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

Vhy am I here?

Why are you here?

Why are we here?

Why am I here?

Reasonable

Concurrent Code

Felix Petriconi

Vhy am I here?

Why are you here'

I saw how we developed multi threaded code in the past.

I saw how easy it is to make mistakes.

I saw and still see how difficult it is to maintain this code.

- I saw how we developed multi threaded code in the past.
- I saw how easy it is to make mistakes.
- I saw and still see how difficult it is to maintain this code.
- I watched recordings from Sean Parent's talks about "Better Code".
- I was impressed.
- I wanted to learn more.

- I saw how we developed multi threaded code in the past.
- I saw how easy it is to make mistakes.
- I saw and still see how difficult it is to maintain this code.
- I watched recordings from Sean Parent's talks about "Better Code".
- I was impressed.
- I wanted to learn more.
- I'm collaborating in his open source project for a new library.
- I'm continuously learning there a lot.
- I care about sharing my knowledge, here at the ACCU conference.

Felix Petriconi

ny am I here?

lotivation

What is your motivation to be here?

Felix Petriconi

Why am I here?

vvny are you nere

Problems from my domain

- Loading of huge images blocks UI
- Storing of files blocks UI
- Re-coding of huge images takes very long
- ▶ DB accesses takes too long
- **.**.

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

Why am I here?

Why are you here?

Problems from my domain

Why do we have to talk about concurrency?

The free lunch is over!

The Art of Writing Reasonable

Felix Petriconi

Why am I here?

Why are you here?

Problems from my

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

Why am I here?

Why are you here:

Problems from my domain

The free lunch is over!

Herb Sutter, 2005¹

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

Why am I here?

Why are you here?

Problems from my domain

The free lunch is over!

Herb Sutter, 2005¹

¹The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software http://www.gotw.ca/publications/concurrency-ddj.htm

The free lunch is over

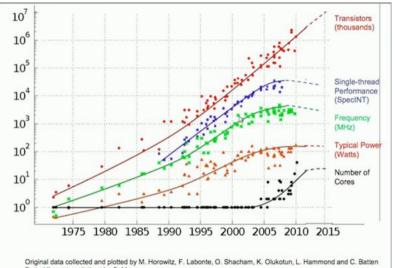




Why am I here?

Why are you

Problems from my



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten Dotted line extrapolations by C. Moore

Desktop Compute Power

8-core 3.5GHz (Sandy Bridge + AMD Radeon 6950)

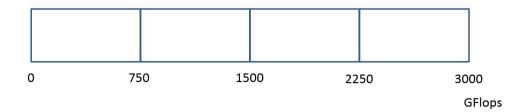
Reasonable
Concurrent Code

Felix Petriconi

/hy am I here?

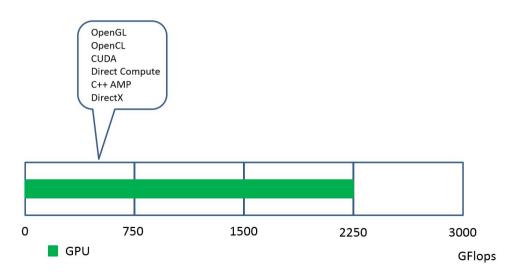
hy are you h

Problems from my domain



Desktop Compute Power

8-core 3.5GHz (Sandy Bridge + AMD Radeon 6950)



Reasonable
Concurrent Code

Felix Petriconi

Why am I here?

Why are you

Problems from my domain

Desktop Compute Power

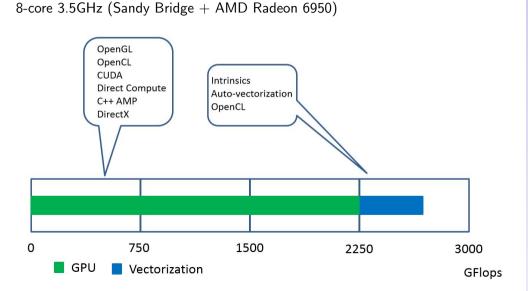
Reasonable
Concurrent Code

Felix Petriconi

Why am I here?

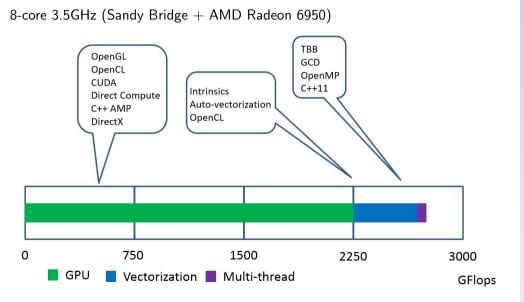
Motivation
Problems from my

domain



Why are you

Problems from my

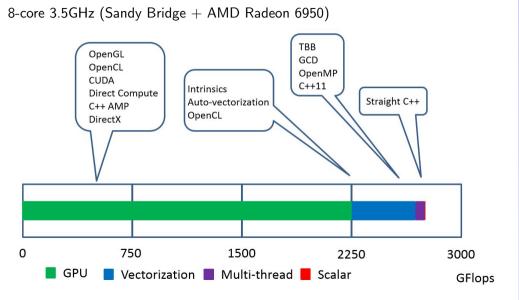


Felix Petriconi

Why am I here?

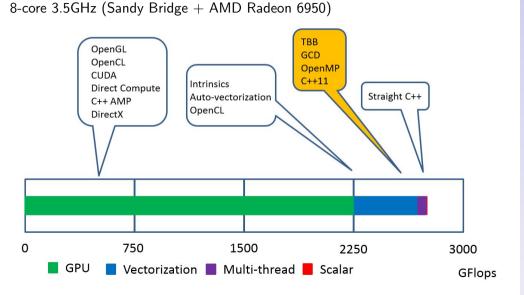
Why are you

Problems from my



Why are you

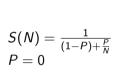
Problems from my domain

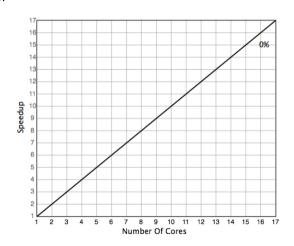


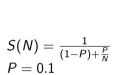
$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

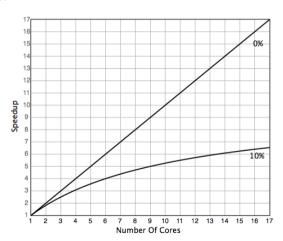
 P : Synchronization [0 - 1]

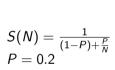
N: Number of Cores

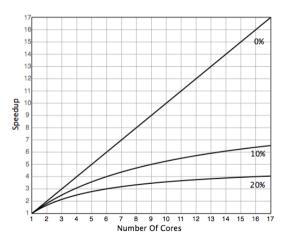


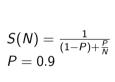


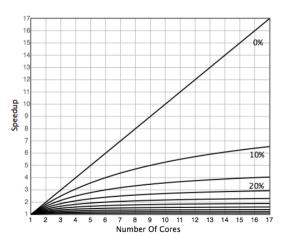












Outline Futures

Reasonable
Concurrent Code

Felix Petriconi

itures

```
Futures
   Why Futures?
   Introduction
   C++ Standard - Futures
       Exceptions
       Deficiencies
   Boost - Futures
       Deficiencies
       Future Continuation
       Future Join
   stlab - Futures
       Executors
       Error Recovery
       Join
       Splits
   Exercise 1
```

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

utures

Why Futures?

C++ Standard Futures Boost - Futures

ercise 1

Why using futures?

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

utures

Why Futures?

C++ Standard Futures Boost - Futures

> lab - Futur kercise 1

Why using futures? Aren't threads, mutex, atomics great?

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

utures

Why Futures?

Futures
Boost - Future

oost - Futures lab - Futures kercise 1

Why using futures?
Aren't threads, mutex, atomics great?
They are great tools "to shot yourself into the foot!"

Felix Petriconi

Why Futures?

- Why using futures? Aren't threads, mutex, atomics great? They are great tools "to shot yourself into the foot!" It is so easy
 - having race conditions
 - having dead locks
 - wasting CPU cycles through contention

The Art of Writin Reasonable Concurrent Code

Felix Petriconi

utures

Why Futures? Introduction

> Boost - Futur tlab - Future Exercise 1

Why using futures?
Aren't threads, mutex, atomics great?
They are great tools "to shot yourself into the foot!"
It is so easy

- having race conditions
- having dead locks
- wasting CPU cycles through contention

Do you program your application in assembly?

Why Futures? Introduction

Boost - Futures

Why using futures?
Aren't threads, mutex, atomics great?
They are great tools "to shot yourself into the foot!"
It is so easy

- having race conditions
- having dead locks
- wasting CPU cycles through contention

Do you program your application in assembly?

Only if it absolute time critical.

Then don't use tools from the level of assembly!

Felix Petriconi

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures

stlab - Futur Exercise 1



▶ Futures provide a mechanism to separate a function from its result

Why Futures?
Introduction
C++ Standard
Futures
Boost - Futures



- ▶ Futures provide a mechanism to separate a function from its result
- ▶ After the function is called the result appears "magically" in the future

Introduction

Futures Boost - Futur stlab - Future



- ▶ Futures provide a mechanism to separate a function from its result
- ▶ After the function is called the result appears "magically" in the future
- ▶ A future is a token to the result of a function

Future Introduction

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Why Futures? Introduction C++ Standard Futures Boost - Futures stlab - Futures



- ▶ Futures provide a mechanism to separate a function from its result
- ▶ After the function is called the result appears "magically" in the future
- ▶ A future is a token to the result of a function
- ▶ Added with C++11

Introduction C++ Standard Futures Boost - Futures



- ▶ Futures provide a mechanism to separate a function from its result
- ▶ After the function is called the result appears "magically" in the future
- ▶ A future is a token to the result of a function
- ► Added with C++11
- ► Futures, resp. promises where invented 1977/1978 by Daniel P. Friedman, David Wise, Henry Baker and Carl Hewitt

```
1 #include <future>
   #include <iostream>
   using namespace std;
   int main() {
     auto getTheAnswer = [] {
       this thread::sleep for(chrono::milliseconds(815)):
10
       return 42:
     };
12
13
     future < int > f2 = async(launch::async, getTheAnswer);
14
15
     // Do other stuff, getting the answer may take longer
     cout << f2.get() << '\n': // access the value</pre>
17 F
```

Futures
Why Futures?
Introduction
C++ Standard Futures
Exceptions
Deficiencies
Boost - Futures
stlab - Futures
Fyercies 1

```
1 #include <future>
   #include <iostream>
   using namespace std;
   int main() {
     auto getTheAnswer = [] {
       this thread::sleep for(chrono::milliseconds(815)):
10
       return 42:
     };
12
13
     future < int > f2 = async(launch::async, getTheAnswer);
14
     // Do other stuff, getting the answer may take longer
     cout << f2.get() << '\n': // access the value</pre>
17 F
```

Futures
Why Futures?
Introduction
C++ Standard Futures
Exceptions
Deficiencies
Boost - Futures
stlab - Futures

Exercise 1

```
1 #include <future>
  #include <iostream>
  using namespace std;
  int main() {
8
    auto getTheAnswer = [] {
9
      this thread::sleep for(chrono::milliseconds(815)):
     return 42:
11
    };
13
    future < int > f2 = async(launch::async, getTheAnswer);
14
15
    // Do other stuff, getting the answer may take longer
    17 }
```

Futures
Why Futures?
Introduction
C++ Standard Futures
Exceptions
Deficiencies
Boost - Futures
stlab - Futures
Fyercies 1

```
1 #include <future>
  #include <iostream>
  using namespace std;
  int main() {
    auto getTheAnswer = [] {
      this thread::sleep for(chrono::milliseconds(815)):
10
      return 42:
    };
12
    future < int > f2 = async(launch::async, getTheAnswer);
14
15
    // Do other stuff, getting the answer may take longer
    17 }
```

Futures?
Introduction
C++ Standard Futures
Exceptions
Deficiencies
Boost - Futures
stlab - Futures
Fyercies 1

```
1 #include <future>
  #include <iostream>
  using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this thread::sleep for(chrono::milliseconds(815)):
10
       return 42:
     };
12
13
     future < int > f2 = async(launch::async, getTheAnswer);
14
     // Do other stuff, getting the answer may take longer
     cout << f2.get() << '\n': // access the value
17 }
```

Futures
Why Futures?
Introduction
C++ Standard Futures
Exceptions
Deficiencies
Boost - Futures
stlab - Futures
Fyercies 1

C++ Standard -

Exercise 1

```
1 #include <future>
  #include <iostream>
  using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this thread::sleep for(chrono::milliseconds(815)):
       return 42:
     };
12
13
     future < int > f2 = async(launch::async, getTheAnswer);
14
15
     // Do other stuff, getting the answer may take longer
16
     cout << f2.get() << '\n': // access the value
17 3
```

Output

42

```
#include (future)
  #include <iostream>
   #include <exception>
  using namespace std:
  int main() {
     auto getTheAnswer = [] {
       throw runtime_error("Bad things happened: Vogons appeared!");
       return 42:
    };
12
     future < int > f2 = asvnc(launch::asvnc, getTheAnswer);
14
15
     // Do other stuff, getting the answer may take longer
16
     trv {
       cout << f2.get() << '\n': // try accessing the value</pre>
18
                                   // rethrows the stored exception
20
     catch (const runtime_error& ex) {
       cout << ex.what() << '\n':
23 }
```

Futures

Introduction
C++ Standard Futures

Exceptions
Deficiencies
Boost - Futures
stlab - Futures
Exercise 1

```
#include (future)
  #include <iostream>
   #include <exception>
   using namespace std:
  int main() {
     auto getTheAnswer = [] {
       throw runtime_error("Bad things happened: Vogons appeared!");
       return 42:
11
    };
12
13
     future < int > f2 = asvnc(launch::asvnc, getTheAnswer);
14
15
     // Do other stuff, getting the answer may take longer
16
     trv {
       cout << f2.get() << '\n': // try accessing the value</pre>
18
                                   // rethrows the stored exception
20
     catch (const runtime_error& ex) {
       cout << ex.what() << '\n':
23 }
```

-utures Why Future

Introduction
C++ Standard Futures
Exceptions
Deficiencies

Boost - Future stlab - Future Exercise 1

```
Why Futures?
Introduction
C++ Standard -
Futures
Exceptions
Deficiencies
Boost - Futures
```

Exercise 1

```
#include (future)
  #include <iostream>
   #include <exception>
   using namespace std:
  int main() {
     auto getTheAnswer = [] {
       throw runtime_error("Bad things happened: Vogons appeared!");
       return 42:
    };
12
13
     future < int > f2 = async(launch::async, getTheAnswer);
14
15
     // Do other stuff, getting the answer may take longer
16
     trv {
       cout << f2.get() << '\n': // try accessing the value</pre>
18
                                   // rethrows the stored exception
20
     catch (const runtime_error& ex) {
       cout << ex.what() << '\n':
23 }
```

```
Why Futures
Introduction
C++ Stand
Futures
Exceptions
Deficiencies
Boost - Future
Stlab - Future
Exercise 1
```

```
#include (future)
  #include <iostream>
   #include <exception>
   using namespace std:
  int main() {
     auto getTheAnswer = [] {
       throw runtime_error("Bad things happened: Vogons appeared!");
       return 42:
     };
12
     future < int > f2 = asvnc(launch::asvnc, getTheAnswer);
14
15
     // Do other stuff, getting the answer may take longer
16
     trv {
17
       cout << f2.get() << '\n': // try accessing the value</pre>
18
                                   // rethrows the stored exception
20
     catch (const runtime_error& ex) {
21
       cout << ex.what() << '\n':
23 }
```

Futures Why Futures? Introduction C++ Standard Futures Exceptions

Futures

Exceptions

Deficiencies

Boost - Futures

stlab - Futures

Exercise 1

```
#include (future)
  #include <iostream>
   #include <exception>
   using namespace std:
   int main() {
     auto getTheAnswer = [] {
       throw runtime_error("Bad things happened: Vogons appeared!");
       return 42:
     };
12
13
     future < int > f2 = asvnc(launch::asvnc, getTheAnswer);
14
15
     // Do other stuff, getting the answer may take longer
     trv {
17
       cout << f2.get() << '\n': // try accessing the value</pre>
                                   // rethrows the stored exception
20
     catch (const runtime_error& ex) {
       cout << ex.what() << '\n':
23 }
```

Output

Bad things happened: Vogons appeared!

Exceptions

Exercise 1

```
#include (future)
  #include <iostream>
   #include <exception>
   using namespace std:
   int main() {
     auto getTheAnswer = [] {
       throw runtime_error("Bad things happened: Vogons appeared!");
       return 42:
     };
12
     future < int > f2 = asvnc(launch::asvnc, getTheAnswer);
14
15
     // Do other stuff, getting the answer may take longer
16
     trv {
17
       cout << f2.get() << '\n': // try accessing the value</pre>
18
                                   // rethrows the stored exception
20
     catch (const runtime_error& ex) {
21
       cout << ex.what() << '\n':
```

Output

Bad things happened: Vogons appeared!

Futures

Why Futures? Introduction C++ Standard -Futures Exceptions Deficiencies Boost - Futures stlab - Futures Exercise 1 ▶ No continuation – .then() X*

Felix Petriconi

itures

Why Futures? Introduction C++ Standard -Futures Exceptions Deficiencies Boost - Futures stlab - Futures Exercise 1

▶ No continuation – .then() X*

Felix Petriconi

Deficiencies

No join − .when_all() and .when_any() X*

Why Futures? Introduction C++ Standard -Futures Exceptions Deficiencies Boost - Futures stlab - Futures

- ► No continuation .then() X*
- No join − .when_all() and .when_any() X*
- ► No split continuation in different directions X

▶ No continuation – .then() X*

No join – .when_all() and .when_any() X*

▶ No split – continuation in different directions X

No cancellation (but can be modelled) X

* Comes with C++17(TS)

Deficiencies

- ▶ No continuation .then() X*
- No join .when_all() and .when_any() X*
- ▶ No split continuation in different directions X
- ▶ No cancellation (but can be modelled) X
- ► No progress monitoring (except ready) X

Deficiencies

▶ No continuation – .then() X* No join – .when_all() and .when_any() X* ▶ No split – continuation in different directions X

- ▶ No cancellation (but can be modelled) X
- ► No progress monitoring (except ready) X
- No custom executor X

* Comes with C++17(TS)

C++11/14 Future Deficiencies

- ▶ No continuation .then() X*
- No join .when_all() and .when_any() X*
- ▶ No split continuation in different directions X
- No cancellation (but can be modelled) X
- ► No progress monitoring (except ready) X
- No custom executor X
- ▶ Blocks on destruction (may even blocks until termination of used thread) X

* Comes with C++17(TS)

Deficiencies

- ▶ No continuation .then() X*
- No join .when_all() and .when_any() X*
- ▶ No split continuation in different directions X
- No cancellation (but can be modelled) X
- No progress monitoring (except ready) X
- No custom executor X
- ▶ Blocks on destruction (may even blocks until termination of used thread) 🗡
- .get() has two problems:

- ▶ No continuation .then() X*
- No join .when_all() and .when_any() X*
- ▶ No split continuation in different directions X
- No cancellation (but can be modelled) X
- No progress monitoring (except ready) X
- No custom executor X
- ▶ Blocks on destruction (may even blocks until termination of used thread) 🗡
- .get() has two problems:
 - 1. One thread resource is consumed which increases contention and possibly causing a deadlock X

* Comes with C++17(TS)

- ▶ No continuation .then() X*
- No join .when_all() and .when_any() X*
- ▶ No split continuation in different directions X
- No cancellation (but can be modelled) X
- No progress monitoring (except ready) X
- No custom executor X
- Blocks on destruction (may even blocks until termination of used thread) X
- .get() has two problems:
 - 1. One thread resource is consumed which increases contention and possibly causing a deadlock X
 - 2. Any subsequent non-dependent calculations on the task are also blocked X

* Comes with C++17(TS)

- ▶ No continuation .then() X*
- No join .when_all() and .when_any() X*
- ▶ No split continuation in different directions X
- No cancellation (but can be modelled) X
- No progress monitoring (except ready) X
- No custom executor X
- ▶ Blocks on destruction (may even blocks until termination of used thread) X
- .get() has two problems:
 - 1. One thread resource is consumed which increases contention and possibly causing a deadlock X
 - 2. Any subsequent non-dependent calculations on the task are also blocked X
- Don't behave as a regular type X
- * Comes with C++17(TS)

Futures

Why Futures? Introduction C++ Standard -Futures

Deficiencies

Future Continuation Future Join stlab - Futures Exercise 1

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Deficiencies

eficiencies uture Continua

stlab - Futures Exercise 1

► Continuation – .then() ✓

▶ Join – .when_all() and .when_any() ✓

Deficiencies

stlab - Futures

- ▶ Continuation .then() ✓
- ▶ Join .when_all() and .when_any() ✓
- ▶ No split continuation in different directions X
- ▶ No cancellation (but can be modelled) X
- ► No progress monitoring (except ready) X

Futures
Why Futures?
Introduction
C++ Standard Futures

Deficiencies

Future Join stlab - Futures Exercise 1

- ▶ Continuation .then() ✓
- ► Join .when_all() and .when_any() ✓
- No split continuation in different directions X
- ▶ No cancellation (but can be modelled) 🗡
- No progress monitoring (except ready) X
- Custom executor

Deficiencies

▶ Join – .when_all() and .when_any() ✓

▶ No split – continuation in different directions X

No cancellation (but can be modelled) X

- No progress monitoring (except ready) X
- Custom executor
- Blocks on destruction (may even blocks until termination of used thread) X
- .get() has two problems:

▶ Continuation – .then() ✓

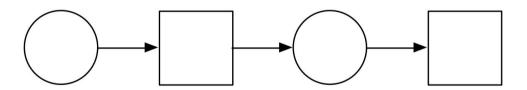
- 1. One thread resource is consumed which increases contention and possibly causing a deadlock X
- 2. Any subsequent non-dependent calculations on the task are also blocked X
- Don't behave as a regular type X

Futures

Exercise 1

Why Futures? Introduction C++ Standard -Futures Boost - Futures Deficiencies Future Continuation

Future Continuation
Future Join
stlab - Futures



```
#include <iostream>
#include <boost/thread/future.hpp>

using namespace std;

int main() {
 boost::future<int> answer = boost::async([]{ return 42; });

boost::future<void> done = answer.then(
 [](boost::future<int> a) { std::cout << a.get() << '\n';} );

// do something else
done.wait(); // waits until future done is fulfilled
}
</pre>
```

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Deficiencies
Future Continuation
Future Join

```
#include <iostream>
#include <boost/thread/future.hpp>

using namespace std;

int main() {
    boost::future<int> answer = boost::async([]{ return 42; });

boost::future<void> done = answer.then(
    [](boost::future<int> a) { std::cout << a.get() << '\n';} );

// do something else
done.wait(); // waits until future done is fulfilled
}</pre>
```

utures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Deficiencies
Future Continuation
Future Join

Future Join stlab - Futures

```
#include <iostream>
#include <boost/thread/future.hpp>

using namespace std;

int main() {
   boost::future<int> answer = boost::async([]{ return 42; });

boost::future<void> done = answer.then(
   [](boost::future<int> a) { std::cout << a.get() << '\n';} );

// do something else
done.wait(); // waits until future done is fulfilled
}</pre>
```

utures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Deficiencies
Future Continuation
Future Join

```
Future Continuation
```

stlab - Futures

```
#include <iostream>
  #include <boost/thread/future.hpp>
   using namespace std;
  int main() {
     boost::future < int > answer = boost::async([]{ return 42; });
8
     boost::future < void > done = answer.then(
       [](boost::future<int> a) { std::cout << a.get() << '\n';} );
11
12
     // do something else
     done.wait():
                  // waits until future done is fulfilled
14 }
```

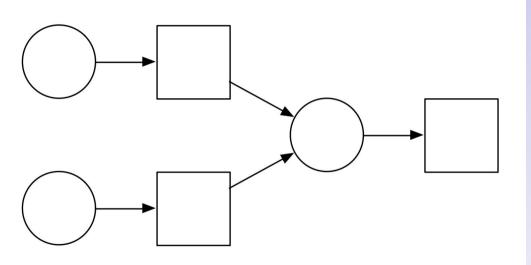
Output

42

utures

Why Futures? Introduction C++ Standard -Futures Boost - Futures Deficiencies Future Continuation

Future Join



C++17(TS) / Boost - Join

```
#include <iostream>
  #include <boost/thread/future.hpp>
  using namespace std;
  int main() {
     auto a = boost::async([]{ return 40; });
     auto b = boost::asvnc([]{ return 2: }):
9
10
     auto answer = boost::when_all(std::move(a), std::move(b)).then(
       [](auto f) {
         auto t = f.get();
        return get<0>(t).get() + get<1>(t).get();
14
      });
16
     // wait for the something else
     cout << answer.get() << '\n';
18 }
```

Felix Petriconi

utures

Why Futures? Introduction C++ Standard -Futures Boost - Futures Deficiencies Future Continuatio

Future Join

```
#include <iostream>
  #include <boost/thread/future.hpp>
  using namespace std;
  int main() {
     auto a = boost::async([]{ return 40; });
8
     auto b = boost::asvnc([]{ return 2: }):
10
     auto answer = boost::when_all(std::move(a), std::move(b)).then(
       [](auto f) {
         auto t = f.get();
        return get <0>(t).get() + get <1>(t).get();
14
      });
16
     // wait for the something else
     cout << answer.get() << '\n';
18 }
```

utures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Deficiencies
Future Continuatio

Future Join

```
#include <iostream>
  #include <boost/thread/future.hpp>
  using namespace std;
  int main() {
     auto a = boost::async([]{ return 40; });
     auto b = boost::asvnc([]{ return 2: }):
9
10
     auto answer = boost::when_all(std::move(a), std::move(b)).then(
       [](auto f) {
         auto t = f.get();
         return get <0>(t).get() + get <1>(t).get();
14
      });
16
     // wait for the something else
     cout << answer.get() << '\n';
18 }
```

utures

Why Futures?
ntroduction
C++ Standard Tutures
Boost - Futures
Deficiencies
Future Continuation

Future Join

```
#include <iostream>
  #include <boost/thread/future.hpp>
  using namespace std;
  int main() {
     auto a = boost::async([]{ return 40; });
     auto b = boost::asvnc([]{ return 2: }):
9
10
     auto answer = boost::when_all(std::move(a), std::move(b)).then(
       [](auto f) {
         auto t = f.get();
         return get <0>(t).get() + get <1>(t).get();
14
      });
16
     // wait for the something else
     cout << answer.get() << '\n';
18 }
```

utures

Why Futures? Introduction C++ Standard -Futures Boost - Futures Deficiencies Future Continuatio

Future Join

```
#include <iostream>
  #include <boost/thread/future.hpp>
  using namespace std;
  int main() {
     auto a = boost::async([]{ return 40; });
8
     auto b = boost::asvnc([]{ return 2: }):
10
     auto answer = boost::when_all(std::move(a), std::move(b)).then(
       [](auto f) {
         auto t = f.get();
13
        return get<0>(t).get() + get<1>(t).get();
14
      });
16
     // wait for the something else
     cout << answer.get() << '\n';
18 }
```

utures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Deficiencies
Future Continuation

Future Join

```
#include <iostream>
  #include <boost/thread/future.hpp>
   using namespace std;
  int main() {
     auto a = boost::async([]{ return 40; });
     auto b = boost::asvnc([]{ return 2: }):
9
10
     auto answer = boost::when_all(std::move(a), std::move(b)).then(
       [](auto f) {
         auto t = f.get();
         return get <0>(t).get() + get <1>(t).get();
14
       });
16
     // wait for the something else
     cout << answer.get() << '\n';</pre>
18 F
```

Output

42

andard -

ost - Futures eficiencies uture Continuation

Future Join

Exercise 1

```
Why Futures?
Introduction
C++ Standard -
Futures
Boost - Futures
Deficiencies
Future Continuation
Future Join
stlab - Futures
Exercise 1
```

```
#include <iostream>
  #include <boost/thread/future.hpp>
   using namespace std;
  int main() {
     auto a = boost::async([]{ return 40; });
     auto b = boost::async([]{ return 2; });
9
     auto answer = boost::when all(std::move(a), std::move(b)).then(
       [](auto f) {
         auto t = f.get():
         return get <0>(t).get() + get <1>(t).get();
14
       }):
16
     // wait for the something else
     cout << answer.get() << '\n';</pre>
18 }
```

What is the type of f?

```
Introduction
C++ Standard -
Futures
Boost - Futures
Deficiencies
Future Continuation
Future Join
stlab - Futures
```

```
#include <iostream>
   #include <boost/thread/future.hpp>
   using namespace std;
  int main() {
     auto a = boost::async([]{ return 40; });
     auto b = boost::async([]{ return 2; });
9
     auto answer = boost::when all(std::move(a), std::move(b)).then(
       [](auto f) {
         auto t = f.get():
         return get <0>(t).get() + get <1>(t).get();
14
       }):
16
     // wait for the something else
     cout << answer.get() << '\n';</pre>
18 }
```

What is the type of f?
f is a future tuple of futures: future<tuple<future<int>>, future<int>>>

stlab - Futures

Felix Petriconi





stlab::future

Source: https://github.com/stlab/libraries Documentation: http://www.stlab.cc/libraries

utures

- ► Continuation .then() ✓
- ▶ Join .when_all() and .when_any() ✓

- ▶ Continuation .then() ✓
- ▶ Join .when_all() and .when_any() ✓
- ▶ Split continuation in different directions ✓

- ► Continuation .then() ✓
- ▶ Join .when_all() and .when_any() ✓
- ▶ Split continuation in different directions ✓
- ► Cancellation ✓

stlab - Futures

- ▶ Continuation .then() ✓
- ▶ Join .when_all() and .when_any() ✓
- ► Split continuation in different directions ✓
- Cancellation
- ▶ No progress monitoring (except ready), more planned X

- ▶ Continuation .then() ✓
- ▶ Join .when_all() and .when_any() ✓
- ► Split continuation in different directions ✓
- ► Cancellation ✓
- No progress monitoring (except ready), more planned X
- Custom executor

Cancellation

stlab - Futures

Custom executor

▶ Continuation – .then() ✓

▶ Join – .when_all() and .when_any() ✓

► Split – continuation in different directions ✓

No progress monitoring (except ready), more planned X

stlab - Futures

- ▶ Continuation .then() ✓
- ▶ Join .when_all() and .when_any() ✓
- ► Split continuation in different directions ✓
- Cancellation
- No progress monitoring (except ready), more planned X
- Custom executor
- Do not block on destruction
- ▶ Behave as a regular type ✓

stlab - Futures Executors

Join Splits

Exercise 1

.

► Continuation – .then() ✓

▶ Join – .when_all() and .when_any() ✓

▶ Split – continuation in different directions ✓

► Cancellation ✓

No progress monitoring (except ready), more planned X

▶ Custom executor ✓

Do not block on destruction

Behave as a regular type

Additional dependencies:

► C++14: boost (optional, variant)

► C++17: none

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
   using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this_thread::sleep_for(chrono::milliseconds(815)):
       return 42:
9
     };
     stlab::future<int> f =
       stlab::asvnc(
         stlab::default executor.// default executor
13
                                  // uses platfrom thread pool on Win/OSX
14
                                  // uses stlab thread pool on other OS
         getTheAnswer
16
17
18
     while (!f.get_try()) {      // does not block
       // Do other stuff, getting the answer may take longer :-)
20
21
     cout << f.get_try().value() << '\n'; // access the value</pre>
                               // throws exception .value() if not ready
24 3
```

Felix Petriconi

Futures

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
   using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this_thread::sleep_for(chrono::milliseconds(815));
       return 42:
9
     };
     stlab::future<int> f =
       stlab::asvnc(
         stlab::default executor.// default executor
13
                                  // uses platfrom thread pool on Win/OSX
14
                                  // uses stlab thread pool on other OS
         getTheAnswer
16
17
18
     while (!f.get_try()) {      // does not block
       // Do other stuff, getting the answer may take longer :-)
20
21
     cout << f.get_try().value() << '\n'; // access the value</pre>
                               // throws exception .value() if not ready
24 3
```

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
   using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this_thread::sleep_for(chrono::milliseconds(815)):
      return 42:
9
    };
10
     stlab::future<int> f =
       stlab::asvnc(
         stlab::default executor.// default executor
13
                                 // uses platfrom thread pool on Win/OSX
14
                                 // uses stlab thread pool on other OS
         getTheAnswer
16
17
18
     while (!f.get_try()) {      // does not block
19
       // Do other stuff, getting the answer may take longer :-)
20
```

// throws exception .value() if not ready

cout << f.get_try().value() << '\n'; // access the value</pre>

stlab::future

21

24 3

Futures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors Error Recovery Join Splits

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
   using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this_thread::sleep_for(chrono::milliseconds(815)):
       return 42:
9
     };
10
     stlab::future<int> f =
       stlab::asvnc(
         stlab::default executor.// default executor
13
                                  // uses platfrom thread pool on Win/OSX
                                  // uses stlab thread pool on other OS
14
         getTheAnswer
16
18
     while (!f.get_try()) {      // does not block
19
       // Do other stuff, getting the answer may take longer :-)
20
21
     cout << f.get_try().value() << '\n'; // access the value</pre>
                               // throws exception .value() if not ready
24 3
```

stlab::future

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
   using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this_thread::sleep_for(chrono::milliseconds(815)):
       return 42:
9
     };
10
     stlab::future<int> f =
       stlab::asvnc(
         stlab::default executor.// default executor
13
                                  // uses platfrom thread pool on Win/OSX
14
                                  // uses stlab thread pool on other OS
15
         getTheAnswer
16
17
18
     while (!f.get_try()) {      // does not block
19
       // Do other stuff, getting the answer may take longer :-)
20
21
     cout << f.get_try().value() << '\n'; // access the value</pre>
                               // throws exception .value() if not ready
24 3
```

Felix Petriconi

Futures

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
   using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this_thread::sleep_for(chrono::milliseconds(815)):
       return 42:
9
     };
     stlab::future<int> f =
       stlab::asvnc(
         stlab::default executor.// default executor
13
                                  // uses platfrom thread pool on Win/OSX
14
                                  // uses stlab thread pool on other OS
         getTheAnswer
16
17
18
     while (!f.get_try()) {      // does not block
19
       // Do other stuff, getting the answer may take longer :-)
20
21
     cout << f.get_try().value() << '\n'; // access the value</pre>
                               // throws exception .value() if not ready
24 3
```

Felix Petriconi

Futures

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
   using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this_thread::sleep_for(chrono::milliseconds(815)):
       return 42:
9
     };
     stlab::future<int> f =
       stlab::asvnc(
         stlab::default executor.// default executor
13
                                  // uses platfrom thread pool on Win/OSX
14
                                  // uses stlab thread pool on other OS
         getTheAnswer
16
17
     while (!f.get_try()) {      // does not block
       // Do other stuff, getting the answer may take longer :-)
20
21
     cout << f.get_try().value() << '\n'; // access the value</pre>
                               // throws exception .value() if not ready
24 3
```

Felix Petriconi

stlab - Futures

stlab - Futures

stlab::future

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
   using namespace std;
  int main() {
     auto getTheAnswer = [] {
       this_thread::sleep_for(chrono::milliseconds(815));
       return 42:
9
     };
     stlab::future<int> f =
       stlab::asvnc(
         stlab::default executor.// default executor
13
                                  // uses platfrom thread pool on Win/OSX
14
                                  // uses stlab thread pool on other OS
         getTheAnswer
16
17
18
     while (!f.get_try()) {      // does not block
19
       // Do other stuff, getting the answer may take longer :-)
20
21
     cout << f.get_try().value() << '\n'; // access the value</pre>
                              // throws exception .value() if not ready
24
```

Output

42

stlab::future - Exceptions

```
1 #include <stlab/future.hpp>
   #include <stlab/default executor.hpp>
   #include <iostream>
   #include <exception>
   int main() {
     auto getTheAnswer = [] {
       throw std::runtime_error("Bad thing happened: Vogons appeared!");
 9
       return 42;
10
     }:
     auto f = stlab::async(stlab::default_executor, getTheAnswer):
     trv {
14
       while (!f.get_try()) { // try accessing the value
15
                              // may rethrow a stored exception
16
         // Do other stuff, getting the answer may take longer
       std::cout << f.get_try().value() << '\n';
20
21
     catch (const std::runtime error& ex) {
       std::cout << ex.what() << '\n':
24 }
```

Felix Petriconi

Futures

```
1 #include <stlab/future.hpp>
   #include <stlab/default executor.hpp>
   #include <iostream>
   #include <exception>
   int main() {
     auto getTheAnswer = [] {
       throw std::runtime_error("Bad thing happened: Vogons appeared!");
 9
       return 42;
     7:
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer):
     trv {
14
       while (!f.get_try()) { // try accessing the value
15
                              // may rethrow a stored exception
16
         // Do other stuff, getting the answer may take longer
18
       std::cout << f.get_try().value() << '\n';
20
21
     catch (const std::runtime error& ex) {
       std::cout << ex.what() << '\n':
24 3
```

stlab - Futures

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
  #include <iostream>
   #include <exception>
   int main() {
     auto getTheAnswer = [] {
       throw std::runtime_error("Bad thing happened: Vogons appeared!");
 9
      return 42;
10
     }:
     auto f = stlab::async(stlab::default_executor, getTheAnswer);
     trv {
14
       while (!f.get_try()) { // try accessing the value
15
                              // may rethrow a stored exception
16
        // Do other stuff, getting the answer may take longer
18
       std::cout << f.get_try().value() << '\n';
20
21
     catch (const std::runtime error& ex) {
       std::cout << ex.what() << '\n':
24 3
```

utures

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
  #include <iostream>
   #include <exception>
   int main() {
     auto getTheAnswer = [] {
       throw std::runtime_error("Bad thing happened: Vogons appeared!");
 9
      return 42;
10
     }:
     auto f = stlab::async(stlab::default_executor, getTheAnswer):
12
     trv {
14
       while (!f.get_try()) { // try accessing the value
15
                              // may rethrow a stored exception
16
         // Do other stuff, getting the answer may take longer
17
       std::cout << f.get_try().value() << '\n';
20
21
     catch (const std::runtime error& ex) {
       std::cout << ex.what() << '\n':
```

stlab::future - Exceptions

24 3

stlab - Futures

stlab::future - Exceptions

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
   #include <exception>
   int main() {
     auto getTheAnswer = [] {
       throw std::runtime_error("Bad thing happened: Vogons appeared!");
 9
       return 42:
10
     }:
     auto f = stlab::async(stlab::default_executor, getTheAnswer);
12
     trv {
14
       while (!f.get_try()) { // try accessing the value
15
                              // may rethrow a stored exception
16
         // Do other stuff, getting the answer may take longer
17
18
       std::cout << f.get_try().value() << '\n';
20
21
     catch (const std::runtime error& ex) {
       std::cout << ex.what() << '\n':
24
```

Output

Bad things happened: Vogons appeared!

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; });
8
     stlab::future < void > done = answer.then(
10
       [](int a)
                                     // pass by value and not by future
         std::cout << a << '\n':
13
      });
14
15
     while (!done.get_try()) {
16
      // do something in the meantime
17
18 }
```

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; });
8
     stlab::future < void > done = answer.then(
10
       [](int a)
                                     // pass by value and not by future
         std::cout << a << '\n':
13
      });
14
15
     while (!done.get_try()) {
16
       // do something in the meantime
17
18 }
```

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors Error Recovery Join

```
Felix Petriconi
```

```
Why Futures?
Introduction
C++ Standard -
Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join
Splits
```

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; });
8
     stlab::future < void > done = answer.then(
       [](int a)
                                     // pass by value and not by future
         std::cout << a << '\n':
13
      });
14
15
     while (!done.get_try()) {
16
       // do something in the meantime
17
18 }
```

```
1 #include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; });
8
     stlab::future < void > done = answer.then(
       [](int a)
                                      // pass by value and not by future
         std::cout << a << '\n':
13
      });
14
15
     while (!done.get_try()) {
16
       // do something in the meantime
17
18 }
```

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors Error Recovery Join

stlab - Futures

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; } );
8
     stlab::future < void > done = answer.then(
       [](int a)
                                      // pass by value and not by future
         std::cout << a << '\n':
13
      });
14
15
     while (!done.get_try()) {
16
       // do something in the meantime
17
18 }
```

Output

42

```
Why Futures?
Untroduction
C+++ Standard -
Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join
Splits
Exercise 1
```

```
1 #include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; } );
8
     stlab::future < void > done = answer.then(
       [](int a)
                                      // pass by value and not by future
         std::cout << a << '\n':
13
      });
14
15
     while (!done.get_try()) {
16
       // do something in the meantime
17
18 }
```

Output

42

Executors

Felix Petriconi

Executors Exercise 1

Executors

Exercise 1

Executors are needed to customize where the task shall be executed

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors Error Recovery Join Splits

- Executors are needed to customize where the task shall be executed
- ► Executors can be general thread pools, serial queues, main queues, dedicated task groups, etc.

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
   #include <iostream>
   #include <QLineEdit>
   #include "OtScheduler.h"
  int main() {
     QLineEdit theAnswerEdit;
9
10
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; } );
13
     stlab::future<void> done = answer.then(
14
       QtScheduler().
                                             // different scheduler
       [&](int a) { the Answer Edit. set Value(a): }// here update in main thread
15
16
18
     while (!done.get_trv()) {
       // do something in the meantime
20
21 }
```

Tutures
Why Futures?
Introduction

Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors

Join

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
   #include <iostream>
   #include <QLineEdit>
   #include "OtScheduler.h"
  int main() {
     QLineEdit theAnswerEdit;
9
10
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; } );
13
     stlab::future<void> done = answer.then(
14
       QtScheduler().
                                             // different scheduler
       [&](int a) { the Answer Edit. set Value(a): }// here update in main thread
15
16
17
18
     while (!done.get_trv()) {
       // do something in the meantime
20
21 }
```

utures

Why Futures? Introduction C++ Standard Futures

Boost - Futures stlab - Futures Executors

rror Recovery .

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
   #include <iostream>
   #include <QLineEdit>
   #include "OtScheduler.h"
  int main() {
     QLineEdit theAnswerEdit;
9
10
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; } );
     stlab::future < void > done = answer.then(
14
       QtScheduler().
                                             // different scheduler
       [&](int a) { the Answer Edit. set Value(a): }// here update in main thread
15
16
18
     while (!done.get_trv()) {
       // do something in the meantime
20
21 }
```

Futures
Why Futures?

Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Fxecutors

Error Recovery

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
   #include <iostream>
   #include <QLineEdit>
   #include "OtScheduler.h"
  int main() {
     QLineEdit theAnswerEdit;
9
10
     auto answer =
       stlab::async(stlab::default_executor, []{ return 42; } );
13
     stlab::future<void> done = answer.then(
14
       QtScheduler().
                                             // different scheduler
       [&](int a) { the Answer Edit. set Value(a): }// here update in main thread
15
16
17
18
     while (!done.get_trv()) {
       // do something in the meantime
20
21 }
```

utures Why Futures

Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures

Executors
Error Recovery
Join

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery

Join Splits

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors

rror Recovery

Splits exercise 1

▶ In boost, executors derive from a common base class

Futures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors

Error Reco

Join Splits

xercise 1

▶ In boost, executors derive from a common base class

► In stlab the executors must provide template <typename F> void operator()(F f)

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Executors
Error Recovery
Join
Splits

- ▶ In boost, executors derive from a common base class
- ▶ In stlab the executors must provide template <typename F> void operator()(F f)
- ▶ Let's build exemplary a custom executor for the Qt GUI, that allows to perform updates in the Qt main event loop

```
#include <QApplication>
  #include <Event>
  class OtExecutor
5 {
     using result_type = void;
     class ExecutorEvent : public QEvent
10
     1:
   public:
     template <typename F>
     void operator()(F f) {
       auto event = new ExecutorEvent(std::move(f));
16
       QApplication::postEvent(event->receiver(), event);
18 }:
```

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors

Error Recovery Join

```
#include <QApplication>
  #include <Event>
   class OtExecutor
5 {
     using result_type = void:
     class ExecutorEvent : public QEvent
10
     1:
   public:
     template <typename F>
     void operator()(F f) {
       auto event = new ExecutorEvent(std::move(f));
16
       QApplication::postEvent(event->receiver(), event);
18 }:
```

utures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors

rror Recovery oin

```
#include <QApplication>
  #include <Event>
  class OtExecutor
5
     using result_type = void;
     class ExecutorEvent : public QEvent
10
     1:
   public:
     template <typename F>
     void operator()(F f) {
       auto event = new ExecutorEvent(std::move(f));
16
       QApplication::postEvent(event->receiver(), event);
18 }:
```

utures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors

rror Recovery

```
#include <QApplication>
  #include <Event>
  class OtExecutor
5 {
     using result_type = void;
     class ExecutorEvent : public QEvent
10
     1:
   public:
     template <typename F>
     void operator()(F f) {
14
       auto event = new ExecutorEvent(std::move(f));
16
       QApplication::postEvent(event->receiver(), event);
18 }:
```

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors

ror Recovery

```
#include <QApplication>
  #include <Event>
   class OtExecutor
5 {
     using result_type = void;
     class ExecutorEvent : public QEvent
10
     1:
   public:
     template <typename F>
     void operator()(F f) {
14
       auto event = new ExecutorEvent(std::move(f));
16
       QApplication::postEvent(event->receiver(), event);
18 }:
```

utures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors

rror Recovery

```
#include <QApplication>
  #include <Event>
   class OtExecutor
5 {
     using result_type = void;
     class ExecutorEvent : public QEvent
10
     1:
   public:
     template <typename F>
     void operator()(F f) {
       auto event = new ExecutorEvent(std::move(f));
16
       QApplication::postEvent(event->receiver(), event);
18 }:
```

utures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Frecutors

rror Recover

```
1 #include <QApplication>
   #include <Event>
   class OtExecutor
     using result_type = void;
     class EventReceiver:
 9
     class ExecutorEvent : public QEvent
       std::function < void() > f:
13
       std::unique_ptr < EventReceiver > _receiver;
14
     public:
16
       ExecutorEvent(std::function<void()> f)
         : QEvent (QEvent :: User)
18
         . f(std::move(f))
19
         . receiver(new EventReceiver()) {
20
         _receiver()->moveToThread(QApplication::instance()->thread());
       void execute() { _f(); }
24
       QObject *receiver() const { return _receiver.get(): }
26
     };
27
   public:
29 };
```

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery

```
1 #include <QApplication>
   #include <Event>
   class OtExecutor
     using result_type = void;
     class EventReceiver:
 9
     class ExecutorEvent : public QEvent
       std::function < void() > f:
13
       std::unique_ptr < EventReceiver > _receiver;
14
15
     public:
16
       ExecutorEvent(std::function < void() > f)
          : QEvent (QEvent :: User)
18
          . f(std::move(f))
19
          . receiver(new EventReceiver()) {
20
          _receiver()->moveToThread(QApplication::instance()->thread());
       void execute() { _f(); }
24
       QObject *receiver() const { return _receiver.get(); }
26
     };
27
   public:
29 };
```

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures

Executors
Error Recovery
Join
Splits

```
#include <QApplication>
   #include <Event>
   class OtExecutor
     using result_type = void;
     class EventReceiver:
 9
     class ExecutorEvent : public QEvent
       std::function < void() > f:
13
       std::unique_ptr < EventReceiver > _receiver;
14
     public:
16
       ExecutorEvent(std::function<void()> f)
         : QEvent (QEvent :: User)
18
         . f(std::move(f))
19
         . receiver(new EventReceiver()) {
20
         _receiver()->moveToThread(QApplication::instance()->thread());
       void execute() { _f(); }
24
       QObject *receiver() const { return _receiver.get(): }
26
     };
27
   public:
29 };
```

Futures
Why Futures?
Introduction
C++ Standard Futures

Boost - Futures
stlab - Futures
Executors
Error Recovery

on Splits xercise 1

```
#include <QApplication>
   #include <Event>
   class OtExecutor
     using result_type = void;
     class EventReceiver:
 9
     class ExecutorEvent : public QEvent
       std::function < void() > f:
       std::unique_ptr < EventReceiver > _receiver;
14
     public:
16
       ExecutorEvent(std::function<void()> f)
         : QEvent (QEvent :: User)
18
         . f(std::move(f))
19
         . receiver (new EventReceiver()) {
20
         _receiver()->moveToThread(QApplication::instance()->thread());
       void execute() { _f(); }
24
       QObject *receiver() const { return _receiver.get(): }
26
     };
27
   public:
29 };
```

Why Futures?
Introduction
C++ Standard Futures

Executors
Error Recovery
Join

28 public: 29 };

```
1 #include <QApplication>
   #include <Event>
   class OtExecutor
     using result_type = void;
     class EventReceiver:
9
     class ExecutorEvent : public QEvent
       std::function < void() > f:
13
       std::unique_ptr < EventReceiver > _receiver;
14
     public:
16
       ExecutorEvent(std::function<void()> f)
         : QEvent (QEvent :: User)
18
         . f(std::move(f))
19
         . receiver(new EventReceiver()) {
20
         receiver()->moveToThread(QApplication::instance()->thread());
       void execute() { _f(); }
24
       QObject *receiver() const { return _receiver.get(): }
26
     };
27
```

Felix Petriconi

Ny Futures?
Introduction
Introd

Executors
Error Recovery
Join
Splits

27

28 public: 29 };

```
1 #include <QApplication>
   #include <Event>
   class OtExecutor
     using result_type = void;
     class EventReceiver:
9
     class ExecutorEvent : public QEvent
       std::function < void() > f:
13
       std::unique_ptr < EventReceiver > _receiver;
14
     public:
16
       ExecutorEvent(std::function<void()> f)
         : QEvent (QEvent :: User)
18
         . f(std::move(f))
19
         . receiver(new EventReceiver()) {
20
         _receiver()->moveToThread(QApplication::instance()->thread());
       void execute() { _f(); }
24
       QObject *receiver() const { return _receiver.get(); }
26
     };
```

Felix Petriconi

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery

```
1 #include <QApplication>
   #include <Event>
   class OtExecutor
     using result_type = void;
     class EventReceiver:
 9
     class ExecutorEvent : public QEvent
       std::function < void() > f:
13
       std::unique_ptr < EventReceiver > _receiver;
14
     public:
16
       ExecutorEvent(std::function<void()> f)
         : QEvent (QEvent :: User)
18
         . f(std::move(f))
19
         . receiver(new EventReceiver()) {
20
         _receiver()->moveToThread(QApplication::instance()->thread());
       void execute() { _f(); }
24
25
       QObject *receiver() const { return _receiver.get(); }
26
     };
27
   public:
29 };
```

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures

Executors
Error Recovery
Join
Splits

QApplication::postEvent(event->receiver(). event):

29 30 }:

```
1 #include <QApplication>
   #include <Event>
   class OtExecutor
     class ExecutorEvent : public QEvent
       QObject *receiver() const { return _receiver.get(); }
9
10
11
     class EventReceiver : public QObject
13
     public:
14
       bool event (GEvent *event) override {
         auto mvEvent = dvnamic cast < Executor Event *> (event):
16
         if (mvEvent) {
           mvEvent -> execute():
18
           return true:
19
20
         return false:
   public:
     template <typename F>
26
     void operator()(F f) {
       auto event = new ExecutorEvent(std::move(f));
```

Felix Petriconi

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join

```
Executors
```

```
#include <QApplication>
   #include <Event>
   class OtExecutor
     class ExecutorEvent : public QEvent
       QObject *receiver() const { return _receiver.get(); }
9
     };
10
11
     class EventReceiver : public QObject
13
     public:
14
       bool event (GEvent *event) override {
         auto mvEvent = dvnamic cast < Executor Event *> (event):
         if (mvEvent) {
           mvEvent -> execute():
18
           return true:
19
20
         return false:
   public:
     template <typename F>
26
     void operator()(F f) {
       auto event = new ExecutorEvent(std::move(f));
       QApplication::postEvent(event->receiver(). event):
29
30 1:
```

QApplication::postEvent(event->receiver(). event):

29 30 }:

```
#include <QApplication>
   #include <Event>
   class OtExecutor
     class ExecutorEvent : public QEvent
       QObject *receiver() const { return _receiver.get(); }
9
     };
10
11
     class EventReceiver : public QObject
13
     public:
14
       bool event (GEvent *event) override {
         auto myEvent = dynamic cast < Executor Event *> (event):
16
         if (mvEvent) {
           mvEvent -> execute():
18
           return true:
19
20
         return false:
   public:
     template <typename F>
26
     void operator()(F f) {
       auto event = new ExecutorEvent(std::move(f));
```

Felix Petriconi

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery

void operator()(F f) {

auto event = new ExecutorEvent(std::move(f));

QApplication::postEvent(event->receiver(). event):

26

28

29 30 }:

```
#include <QApplication>
   #include <Event>
   class OtExecutor
     class ExecutorEvent : public QEvent
       QObject *receiver() const { return _receiver.get(); }
9
     };
10
11
     class EventReceiver : public QObject
13
     public:
14
       bool event (QEvent *event) override {
         auto mvEvent = dvnamic cast < Executor Event *> (event):
16
         if (mvEvent) {
           mvEvent -> execute():
18
           return true:
19
20
         return false:
   public:
     template <typename F>
```

Felix Petriconi

why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery

QApplication::postEvent(event->receiver(). event):

28

29 30 }:

```
#include <QApplication>
   #include <Event>
   class OtExecutor
     class ExecutorEvent : public QEvent
       QObject *receiver() const { return _receiver.get(); }
9
     };
10
11
     class EventReceiver : public QObject
13
     public:
14
       bool event (GEvent *event) override {
         auto mvEvent = dvnamic cast < Executor Event *> (event):
         if (mvEvent) {
           mvEvent -> execute():
18
           return true:
19
20
         return false:
   public:
     template <typename F>
26
     void operator()(F f) {
       auto event = new ExecutorEvent(std::move(f));
```

Felix Petriconi

utures
Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Executors
Error Recovery
Join

```
Executors
```

```
#include <QApplication>
   #include <Event>
   class OtExecutor
     class ExecutorEvent : public QEvent
       QObject *receiver() const { return _receiver.get(); }
9
     };
10
11
     class EventReceiver : public QObject
13
     public:
14
       bool event (GEvent *event) override {
         auto mvEvent = dvnamic cast < Executor Event *> (event):
16
         if (mvEvent) {
           mvEvent -> execute():
18
           return true:
19
20
         return false:
   public:
     template <typename F>
26
     void operator()(F f) {
       auto event = new ExecutorEvent(std::move(f));
28
       QApplication::postEvent(event->receiver(). event):
29
30 1:
```

21 3

```
1 int main() {
     auto getTheAnswer = [] {
       throw std::runtime_error("Bad thing happened: Vogons appeared");
       std::cout << "I have got the answer\n": return 42:
 5
     auto handleTheAnswer = [](int v) {
       if (v == 0) std::cout << "We have a problem!\n";</pre>
       else std::cout << "The answer is " << v << '\n':
9
     };
10
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer)
       .recover([](stlab::future<int> result) {
13
         if (result.error()) {
14
           std::cout << "Listen to Vogon poetry!\n";</pre>
           return 0:
16
         return result.get_trv().value();
18
     }).then(handleTheAnswer):
20
     while (!f.get_trv());
```

Felix Petriconi

Futures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors

Error Recovery

```
1 int main() {
     auto getTheAnswer = [] {
       throw std::runtime error("Bad thing happened: Vogons appeared"):
       std::cout << "I have got the answer\n"; return 42;
 5
     }:
     auto handleTheAnswer = [](int v) {
       if (v == 0) std::cout << "We have a problem!\n";</pre>
       else std::cout << "The answer is " << v << '\n':
9
     };
10
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer)
       .recover([](stlab::future<int> result) {
13
         if (result.error()) {
14
           std::cout << "Listen to Vogon poetry!\n";</pre>
           return 0:
16
         return result.get_trv().value();
18
     }).then(handleTheAnswer):
20
     while (!f.get_trv());
21 3
```

Futures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors

Error Recovery

21 3

```
1 int main() {
     auto getTheAnswer = [] {
       throw std::runtime error("Bad thing happened: Vogons appeared"):
       std::cout << "I have got the answer\n": return 42:
 5
     auto handleTheAnswer = [](int v) {
       if (v == 0) std::cout << "We have a problem!\n";</pre>
8
       else std::cout << "The answer is " << v << '\n':
9
    };
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer)
       .recover([](stlab::future<int> result) {
13
         if (result.error()) {
14
           std::cout << "Listen to Vogon poetry!\n";</pre>
           return 0:
16
         return result.get_trv().value();
18
     }).then(handleTheAnswer):
20
     while (!f.get_trv());
```

Felix Petriconi

utures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors

Error Recovery

21 3

```
1 int main() {
     auto getTheAnswer = [] {
       throw std::runtime error("Bad thing happened: Vogons appeared"):
       std::cout << "I have got the answer\n": return 42:
     auto handleTheAnswer = [](int v) {
       if (v == 0) std::cout << "We have a problem!\n";</pre>
       else std::cout << "The answer is " << v << '\n':
9
     };
10
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer)
       .recover([](stlab::future<int> result) {
13
         if (result.error()) {
14
           std::cout << "Listen to Vogon poetry!\n";</pre>
           return 0:
16
         return result.get_trv().value();
18
     }).then(handleTheAnswer):
20
     while (!f.get_trv());
```

Felix Petriconi

Futures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors

Error Recovery

```
1 int main() {
     auto getTheAnswer = [] {
       throw std::runtime error("Bad thing happened: Vogons appeared"):
       std::cout << "I have got the answer\n": return 42:
     auto handleTheAnswer = [](int v) {
       if (v == 0) std::cout << "We have a problem!\n";</pre>
       else std::cout << "The answer is " << v << '\n':
9
     };
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer)
       .recover([](stlab::future<int> result) {
13
         if (result.error()) {
14
           std::cout << "Listen to Vogon poetry!\n";</pre>
           return 0:
16
         return result.get_trv().value();
18
     }).then(handleTheAnswer):
20
     while (!f.get_trv());
21 3
```

Futures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors

Error Recovery

```
1 int main() {
     auto getTheAnswer = [] {
       throw std::runtime error("Bad thing happened: Vogons appeared"):
       std::cout << "I have got the answer\n": return 42:
 5
     auto handleTheAnswer = [](int v) {
       if (v == 0) std::cout << "We have a problem!\n";</pre>
       else std::cout << "The answer is " << v << '\n':
9
     };
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer)
       .recover([](stlab::future<int> result) {
13
         if (result.error()) {
14
           std::cout << "Listen to Vogon poetry!\n";</pre>
15
           return 0:
16
         return result.get_trv().value();
18
     }).then(handleTheAnswer):
20
     while (!f.get_trv());
21 3
```

Futures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures

Error Recovery

```
1 int main() {
     auto getTheAnswer = [] {
       throw std::runtime error("Bad thing happened: Vogons appeared"):
       std::cout << "I have got the answer\n": return 42:
 5
     auto handleTheAnswer = [](int v) {
       if (v == 0) std::cout << "We have a problem!\n";</pre>
       else std::cout << "The answer is " << v << '\n':
9
     };
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer)
       .recover([](stlab::future<int> result) {
13
         if (result.error()) {
14
           std::cout << "Listen to Vogon poetry!\n";</pre>
           return 0:
16
         return result.get_trv().value();
18
     }).then(handleTheAnswer):
20
     while (!f.get_trv());
21 3
```

utures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures

Error Recovery

```
1 int main() {
     auto getTheAnswer = [] {
       throw std::runtime_error("Bad thing happened: Vogons appeared");
       std::cout << "I have got the answer\n": return 42:
 5
     auto handleTheAnswer = [](int v) {
       if (v == 0) std::cout << "We have a problem!\n";</pre>
       else std::cout << "The answer is " << v << '\n':
9
     };
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer)
       .recover([](stlab::future<int> result) {
13
         if (result.error()) {
14
           std::cout << "Listen to Vogon poetry!\n";</pre>
           return 0:
16
17
         return result.get_trv().value();
18
     }).then(handleTheAnswer):
20
     while (!f.get_trv());
21 3
```

utures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures

Error Recovery

Error Recovery

```
1 int main() {
     auto getTheAnswer = [] {
       throw std::runtime_error("Bad thing happened: Vogons appeared");
       std::cout << "I have got the answer\n": return 42:
     auto handleTheAnswer = [](int v) {
       if (v == 0) std::cout << "We have a problem!\n";</pre>
       else std::cout << "The answer is " << v << '\n':
9
     };
10
11
     auto f = stlab::async(stlab::default_executor, getTheAnswer)
       .recover([](stlab::future<int> result) {
13
         if (result.error()) {
14
           std::cout << "Listen to Vogon poetry!\n";</pre>
           return 0:
16
         return result.get_trv().value();
18
     }).then(handleTheAnswer):
20
     while (!f.get_trv());
21 3
```

Output

Listen to Vogon poetry! We have a problem!

```
#include <stlab/future.hpp>
   #include <stlab/default_executor.hpp>
   #include <iostream>
   using namespace stlab;
   int main() {
     auto a = async(default_executor,[]{ return 40; });
 9
     auto b = async(default_executor,[]{ return 2; });
10
11
     auto answer = when all(
       default_executor.
       [](int x, int y) { return x + y; },
14
       a. b):
15
16
     while (!answer.get trv()) {
17
     // wait for something else
18
     std::cout << answer.get_trv().value() << '\n':
20 }
```

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join
Splits

20 }

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  using namespace stlab;
  int main() {
     auto a = async(default_executor,[]{ return 40; });
9
     auto b = async(default_executor,[]{ return 2; });
10
11
     auto answer = when all(
       default_executor.
13
       [](int x, int y) { return x + y; },
14
       a. b):
15
16
     while (!answer.get trv()) {
17
     // wait for something else
18
     std::cout << answer.get_trv().value() << '\n':
```

Felix Petriconi

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join
Solits

```
#include <stlab/future.hpp>
   #include <stlab/default_executor.hpp>
   #include <iostream>
   using namespace stlab;
   int main() {
     auto a = async(default_executor,[]{ return 40; });
 9
     auto b = async(default_executor,[]{ return 2; });
10
11
     auto answer = when all(
       default_executor.
13
       [](int x, int y) { return x + y; },
14
       a. b):
15
16
     while (!answer.get trv()) {
17
     // wait for something else
18
     std::cout << answer.get_trv().value() << '\n':
20 }
```

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join
Splits

```
#include <stlab/future.hpp>
   #include <stlab/default_executor.hpp>
   #include <iostream>
   using namespace stlab;
   int main() {
     auto a = async(default_executor,[]{ return 40; });
 9
     auto b = async(default_executor,[]{ return 2; });
11
     auto answer = when all(
       default_executor.
13
       [](int x, int y) { return x + y; },
14
       a. b):
15
16
     while (!answer.get trv()) {
17
     // wait for something else
18
     std::cout << answer.get_trv().value() << '\n':
20 }
```

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Extlab - Futures
Executors
Error Recovery
Join
Solits

```
#include <stlab/future.hpp>
   #include <stlab/default_executor.hpp>
   #include <iostream>
   using namespace stlab;
   int main() {
     auto a = async(default_executor,[]{ return 40; });
 9
     auto b = async(default_executor,[]{ return 2; });
10
11
     auto answer = when all(
12
       default_executor.
13
       [](int x, int y) { return x + y; },
14
       a. b):
15
16
     while (!answer.get trv()) {
17
     // wait for something else
18
     std::cout << answer.get_trv().value() << '\n':
20 }
```

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
Executors
Error Recovery
Join

```
#include <stlab/future.hpp>
   #include <stlab/default_executor.hpp>
   #include <iostream>
   using namespace stlab;
   int main() {
     auto a = async(default_executor,[]{ return 40; });
 9
     auto b = async(default_executor,[]{ return 2; });
10
11
     auto answer = when all(
12
       default_executor.
       [](int x, int y) { return x + y; }.
14
       a. b):
15
16
     while (!answer.get trv()) {
17
     // wait for something else
18
     std::cout << answer.get_trv().value() << '\n':
20 }
```

utures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join

```
Why Futures?
Introduction
C++ Standard
Futures
Boost - Futures
stab - Futures
Executors
Error Recover
Join
Splits
Exercise 1
```

```
#include <stlab/future.hpp>
   #include <stlab/default_executor.hpp>
   #include <iostream>
   using namespace stlab;
   int main() {
     auto a = async(default_executor,[]{ return 40; });
 9
     auto b = async(default_executor,[]{ return 2; });
10
11
     auto answer = when all(
       default_executor.
       [](int x, int y) { return x + y; },
14
       a, b):
15
16
     while (!answer.get trv()) {
17
     // wait for something else
18
     std::cout << answer.get_trv().value() << '\n':
20 }
```

20 }

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  using namespace stlab;
  int main() {
     auto a = async(default_executor,[]{ return 40; });
9
     auto b = async(default_executor,[]{ return 2; });
10
11
     auto answer = when all(
       default_executor.
13
       [](int x, int y) { return x + y; },
14
       a. b):
15
16
     while (!answer.get_try()) {
17
     // wait for something else
18
19
     std::cout << answer.get_trv().value() << '\n';
```

Felix Petriconi

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join

loin

```
#include <stlab/future.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  using namespace stlab;
  int main() {
     auto a = async(default_executor,[]{ return 40; });
9
     auto b = async(default_executor,[]{ return 2; });
10
11
     auto answer = when all(
       default_executor.
       [](int x, int y) { return x + y; },
14
       a. b):
15
16
     while (!answer.get trv()) {
17
     // wait for something else
18
19
     std::cout << answer.get_trv().value() << '\n';
20 }
```

Output

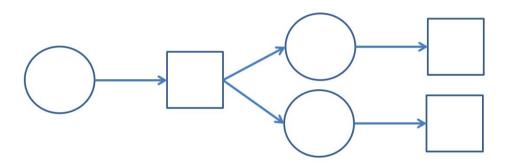
42

Futures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors Error Recovery

Splits

Exercise 1



```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
  using namespace stlab;
  int main() {
     auto answer = asvnc(default executor.[]{ return 42: }):
9
     auto dent = answer.then([](int a) {
       std::cout << "Tell the answer " << a << " Arthur Dent\n";
    }):
14
     auto marvin = answer.then([](int a) {
       std::cout << "May the answer " << a << " shear up Marvin\n":
16
     1):
     while (!dent.get_try() && !marvin.get_try()) {
     // wait for something else
20
21 }
```

stlab::future - Split

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
  using namespace stlab;
  int main() {
     auto answer = async(default_executor,[]{ return 42; });
9
     auto dent = answer.then([](int a) {
       std::cout << "Tell the answer " << a << " Arthur Dent\n";
    }):
14
     auto marvin = answer.then([](int a) {
       std::cout << "May the answer " << a << " shear up Marvin\n":
16
     1):
     while (!dent.get_try() && !marvin.get_try()) {
     // wait for something else
20
21 }
```

Futures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors Error Recovery Join Splits

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
  using namespace stlab;
  int main() {
     auto answer = async(default_executor,[]{ return 42; });
9
     auto dent = answer.then([](int a) {
       std::cout << "Tell the answer " << a << " Arthur Dent\n";
11
    }):
14
     auto marvin = answer.then([](int a) {
       std::cout << "May the answer " << a << " shear up Marvin\n";
16
     1):
     while (!dent.get_try() && !marvin.get_try()) {
     // wait for something else
20
21 }
```

Futures

Why Futures?
Introduction
C++ Standard Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join
Splits

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Executors Error Recovery Join Solits

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
  using namespace stlab;
  int main() {
     auto answer = async(default_executor,[]{ return 42; });
9
     auto dent = answer.then([](int a) {
       std::cout << "Tell the answer " << a << " Arthur Dent\n";
    }):
14
     auto marvin = answer.then([](int a) {
       std::cout << "May the answer " << a << " shear up Marvin\n";
16
     1):
17
     while (!dent.get_try() && !marvin.get_try()) {
     // wait for something else
20
21 }
```

stlab::future - Split

```
Why Futures?
Introduction
C++ Standard -
Futures
Boost - Futures
stlab - Futures
Executors
Exror Recovery
Join
Splits
Exercise 1
```

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
  using namespace stlab:
  int main() {
     auto answer = async(default_executor,[]{ return 42; });
     auto dent = answer.then([](int a) {
       std::cout << "Tell the answer " << a << " Arthur Dent\n":
    }):
     auto marvin = answer.then([](int a) {
14
       std::cout << "May the answer " << a << " shear up Marvin\n":
16
     1):
     while (!dent.get_try() && !marvin.get_try()) {
     // wait for something else
20
21
```

Output

Tell the answer May the answer 42 Arthur Dent

42 shear up Marvin

```
Why Futures?
Introduction
C++ Standard -
Futures
Boost - Futures
stlab - Futures
Executors
Error Recovery
Join
Splits
```

```
1 #include <stlab/future.hpp>
  #include <stlab/default executor.hpp>
   #include <iostream>
  using namespace stlab:
  int main() {
     auto answer = asvnc(default executor.[]{ return 42: }):
     auto dent = answer.then([](int a) {
       std::cout << "Tell the answer " << a << " Arthur Dent\n":
    }):
     auto marvin = answer.then([](int a) {
14
       std::cout << "May the answer " << a << " shear up Marvin\n":
16
    }):
     while (!dent.get_try() && !marvin.get_try()) {
     // wait for something else
20
21 3
```

Output

Tell the answer May the answer 42 Arthur Dent

42 shear up Marvin ⇒ Race condition by using std::cout

Futures

Why Futures? Introduction C++ Standard Futures Boost - Futures stlab - Futures Exercise 1

Demo Exercise 1

Why Futures? Introduction C++ Standard Futures Boost - Futures stlab - Futures Fxercise 1

Change the application in a way that

- using Start does not block the UI,
- ▶ it is possible to cancel the running operation,
- ▶ it is possible to restart it.

utures

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Exercise 1

Futures are a great concept to structure the code so that it runs with minimal contention.

Why Futures? Introduction C++ Standard -Futures Boost - Futures stlab - Futures Fxercise 1

Futures are a great concept to structure the code so that it runs with minimal contention.

After a single execution the graph cannot be used any more.

Outline Channels

Felix Petriconi

Channel Motivation

Channel - Stateless Process

Channel - Split

Channel - Join

Exercise 2

Channel Stateful Process Exercise 3

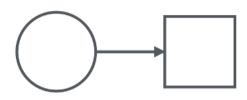


Felix Petriconi

Channel Motivation

Channel - Stateles Process

> Channel Stateful Process



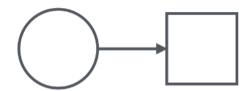
The Art of Writin Reasonable Concurrent Code

Felix Petriconi

Channel Motivation

Channel - Stateles Process

Channel Statef



► Each change triggers a notification to the sink values

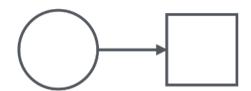
The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Channel Motivation

Process

Channel Statef Process



- ► Each change triggers a notification to the sink values
- ► Channels allow the creation of persistent execution graphs

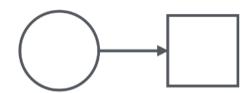
The Art of Writing Reasonable Concurrent Code

Felix Petriconi

Channel Motivation

Channel - Statele Process

Channel Statef



- ► Each change triggers a notification to the sink values
- Channels allow the creation of persistent execution graphs
- ▶ This is also known as reactive programming model

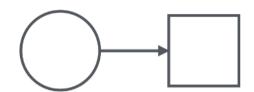
The Art of Writin Reasonable Concurrent Code

Felix Petriconi

Channel Motivation

Channel - Stateles
Process

Channel Statef



- ► Each change triggers a notification to the sink values
- Channels allow the creation of persistent execution graphs
- ▶ This is also known as reactive programming model
- ▶ First published by Tony Hoare 1978

Channel Motivation

Process
Channel - Split

Exercise 2
Channel Statefu

Channel State Process

```
1 #include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
    stlab::sender <int > send: // sending part of the channel
     stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) = // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       [](int x){ std::cout << x << '\n'; }; // stateless process
     auto printer_process =
                                   // attaching process to the receiving
14
      receiver | printer;
15
                                   // part
16
    receiver.set readv():
                                   // no more processes will be attached
17
                                   // process starts to work
     send(1): send(2): send(3):
                                  // start sending into the channel
20
     int end: std::cin >> end: // simply wait to end application
21 3
```

```
Channel
Motivation
```

Process
Channel - Split

Exercise 2

Channel Statefu

Channel Statefu Process

```
#include <stlab/channel.hpp>
  #include <stlab/default executor.hpp>
  #include <iostream>
  int main() {
    stlab::sender <int > send: // sending part of the channel
     stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) = // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       fl(int x)f std::cout << x << '\n'; }; // stateless process</pre>
     auto printer_process =
                                    // attaching process to the receiving
14
      receiver | printer;
15
                                    // part
16
    receiver.set readv():
                                    // no more processes will be attached
17
                                   // process starts to work
     send(1): send(2): send(3):
                                  // start sending into the channel
20
     int end: std::cin >> end: // simply wait to end application
21 3
```

```
Channel
Motivation
```

Process Channel - Split

> Exercise 2 hannel Statefu

Channel Statefi Process

```
1 #include <stlab/channel.hpp>
  #include <stlab/default executor.hpp>
  #include <iostream>
  int main() {
     stlab::sender < int > send: // sending part of the channel
     stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) =
                                // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       fl(int x){ std::cout << x << '\n'; }; // stateless process</pre>
     auto printer_process =
                                    // attaching process to the receiving
14
      receiver | printer;
15
                                    // part
16
    receiver.set readv():
                                    // no more processes will be attached
17
                                    // process starts to work
     send(1): send(2): send(3):
                                  // start sending into the channel
20
     int end: std::cin >> end:
                                 // simply wait to end application
21 3
```

Channel Motivation

Process
Channel - Split

Exercise 2

Channel Statefu

Channel State Process

```
1 #include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
    stlab::sender <int > send: // sending part of the channel
     stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) =
                                // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       [](int x){ std::cout << x << '\n'; }; // stateless process
13
     auto printer_process =
                                    // attaching process to the receiving
14
      receiver | printer;
15
                                    // part
16
    receiver.set readv():
                                    // no more processes will be attached
17
                                   // process starts to work
     send(1): send(2): send(3):
                                  // start sending into the channel
20
     int end: std::cin >> end:
                                 // simply wait to end application
21 3
```

```
Channel
Motivation
```

Process
Channel - Split

Exercise 2
hannel Statefu

Process

```
1 #include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
     stlab::sender <int > send; // sending part of the channel
    stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) =
                                // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       fl(int x){ std::cout << x << '\n'; }; // stateless process</pre>
     auto printer_process =
                                    // attaching process to the receiving
14
      receiver | printer;
15
                                    // part
16
    receiver.set readv():
                                    // no more processes will be attached
17
                                    // process starts to work
     send(1): send(2): send(3):
                                  // start sending into the channel
20
     int end: std::cin >> end: // simply wait to end application
21 3
```

```
Felix Petriconi
```

```
Channel
Motivation
```

Process
Channel - Split

Channel - Join
Exercise 2
Channel Statefu

Channel State Process

```
1 #include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
    stlab::sender < int > send: // sending part of the channel
     stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) = // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       [](int x){ std::cout << x << '\n'; }; // stateless process
     auto printer_process =
                                   // attaching process to the receiving
14
      receiver | printer;
                                   // part
16
    receiver.set readv():
                                   // no more processes will be attached
17
                                   // process starts to work
     send(1): send(2): send(3):
                                  // start sending into the channel
20
     int end: std::cin >> end: // simply wait to end application
21 3
```

```
Felix Petriconi
```

```
Channel
Motivation
```

Process
Channel - Split

channel - Join exercise 2

Process

```
1 #include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
    stlab::sender < int > send: // sending part of the channel
     stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) = // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       [](int x){ std::cout << x << '\n'; }; // stateless process
11
13
     auto printer_process =
                                    // attaching process to the receiving
14
      receiver | printer;
15
                                    // part
16
    receiver.set readv():
                                    // no more processes will be attached
17
                                   // process starts to work
     send(1): send(2): send(3):
                                  // start sending into the channel
20
     int end: std::cin >> end: // simply wait to end application
21 3
```

Channel Motivation

Process Channel - Split

Exercise 2

Channel Stat Process

```
1 #include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
    stlab::sender < int > send: // sending part of the channel
    stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) = // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       [](int x){ std::cout << x << '\n'; }; // stateless process
     auto printer_process =
14
      receiver | printer;
                                    // attaching process to the receiving
                                    // part
16
    receiver.set readv():
                                    // no more processes will be attached
17
                                   // process starts to work
     send(1): send(2): send(3):
                                  // start sending into the channel
20
     int end: std::cin >> end:
                                 // simply wait to end application
21 3
```

Channel Motivation

Process
Channel - Split

Channel - Join Exercise 2 Channel Stateft

Channel Statefu Process

```
1 #include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
  int main() {
     stlab::sender < int > send: // sending part of the channel
     stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) = // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       [](int x){ std::cout << x << '\n'; }; // stateless process
13
     auto printer_process =
14
      receiver | printer;
                                    // attaching process to the receiving
15
                                    // part
16
    receiver.set readv():
                                    // no more processes will be attached
17
                                    // process starts to work
     send(1): send(2): send(3):
                                   // start sending into the channel
20
     int end: std::cin >> end:
                                  // simply wait to end application
21 3
```

Channel Motivation

Process
Channel - Split

Exercise 2
Channel Statefu

Channel Stat Process

```
1 #include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
   #include <iostream>
   int main() {
     stlab::sender < int > send: // sending part of the channel
     stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) = // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       fl(int x){ std::cout << x << '\n'; }; // stateless process</pre>
13
     auto printer_process =
                                    // attaching process to the receiving
14
      receiver | printer;
15
                                    // part
16
     receiver.set readv():
                                    // no more processes will be attached
17
                                    // process starts to work
     send(1): send(2): send(3):
                                    // start sending into the channel
20
     int end: std::cin >> end:
                                  // simply wait to end application
21 3
```

Channel - Join Exercise 2

Channel Statef

```
1 #include <stlab/channel.hpp>
  #include <stlab/default executor.hpp>
  #include <iostream>
  int main() {
     stlab::sender < int > send: // sending part of the channel
     stlab::receiver < int > receiver; // receiving part of the channel
     std::tie(send, receiver) = // combining both to a channel
       stlab::channel<int>(stlab::default executor):
9
     auto printer =
       [](int x){ std::cout << x << '\n'; }; // stateless process
13
     auto printer_process =
                                    // attaching process to the receiving
14
      receiver | printer;
15
                                    // part
16
    receiver.set readv():
                                    // no more processes will be attached
17
                                    // process starts to work
     send(1): send(2): send(3):
                                   // start sending into the channel
20
     int end: std::cin >> end:
                                   // simply wait to end application
21 3
```

#include <stlab/default executor.hpp>

1 #include <stlab/channel.hpp>

#include <iostream>

Felix Petriconi

```
int main() {
 stlab::sender < int > send: // sending part of the channel
  stlab::receiver < int > receiver; // receiving part of the channel
  std::tie(send, receiver) =
                            // combining both to a channel
    stlab::channel<int>(stlab::default executor):
  auto printer =
    fl(int x){ std::cout << x << '\n'; }; // stateless process</pre>
  auto printer_process =
                                // attaching process to the receiving
   receiver | printer;
                                // part
 receiver.set readv():
                                // no more processes will be attached
                                // process starts to work
  send(1); send(2); send(3);
                               // start sending into the channel
  int end: std::cin >> end: // simply wait to end application
    Output
```

3

9

12

14

16

20

21 3

Channel - Split



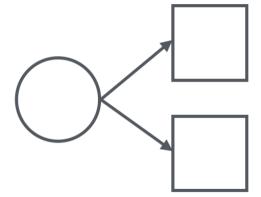
Felix Petriconi



Channel - Stateles

Channel - Split Channel - Join Exercise 2

Channel Statefu



Channel - Split Process

```
using namespace stlab;
  int main() {
     sender <int> send:
     receiver < int > receiver:
     std::tie(send. receiver) = channel<int>(default executor);
     auto printerA = [](int x){ printf("Process A %d\n", x); };
8
     auto printerB = [](int x){ printf("Process B %d\n", x); };
10
     auto printer_processA = receiver | printerA;
11
     auto printer_processB = receiver | printerB;
12
13
     receiver.set_readv():
                                  // no more processes will be attached
14
                                    // process may start to work
     send(1): send(2): send(3):
     int end: std::cin >> end:
17 }
```

Felix Petriconi

Channel Motivation

Channel - Stateles

Channel - Split Channel - Join

Channel Stateful Process

```
Felix Petriconi
```

```
Channel
Motivation
```

Channel - Stateles Process

Channel - Split Channel - Join Exercise 2

Channel Statefu

Channel Sta Process

```
using namespace stlab;
  int main() {
     sender < int > send:
     receiver <int> receiver:
     std::tie(send. receiver) = channel<int>(default executor);
     auto printerA = [](int x){ printf("Process A %d\n", x); };
8
     auto printerB = [](int x){ printf("Process B %d\n", x); };
10
     auto printer_processA = receiver | printerA;
11
     auto printer_processB = receiver | printerB;
12
13
     receiver.set_readv();
                                   // no more processes will be attached
14
                                    // process may start to work
     send(1): send(2): send(3):
     int end: std::cin >> end:
17 }
```

```
Felix Petriconi
```

```
hannal
```

Motivation

Channel - Split

Exercise 2
Channel State

Process

```
using namespace stlab;
  int main() {
     sender < int > send:
     receiver < int > receiver:
     std::tie(send. receiver) = channel<int>(default executor);
     auto printerA = [](int x){ printf("Process A %d\n", x); };
8
     auto printerB = [](int x){ printf("Process B %d\n", x); };
10
     auto printer_processA = receiver | printerA;
11
     auto printer_processB = receiver | printerB;
12
13
     receiver.set_readv():
                                  // no more processes will be attached
14
                                    // process may start to work
     send(1): send(2): send(3):
     int end: std::cin >> end:
17 }
```

```
Felix Petriconi
```

Channel - Split

```
using namespace stlab;
  int main() {
     sender <int> send:
     receiver < int > receiver:
     std::tie(send. receiver) = channel<int>(default executor);
     auto printerA = [](int x){ printf("Process A %d\n", x); };
8
     auto printerB = [](int x){ printf("Process B %d\n", x); };
9
10
     auto printer_processA = receiver | printerA;
11
     auto printer_processB = receiver | printerB;
12
13
     receiver.set_readv();
                                   // no more processes will be attached
14
                                    // process may start to work
     send(1): send(2): send(3):
     int end: std::cin >> end:
17 }
```

Channel - Split Process

```
using namespace stlab;
  int main() {
     sender < int > send:
     receiver < int > receiver:
     std::tie(send. receiver) = channel<int>(default executor);
     auto printerA = [](int x){ printf("Process A %d\n", x); };
8
     auto printerB = [](int x){ printf("Process B %d\n", x); };
10
     auto printer_processA = receiver | printerA;
11
     auto printer_processB = receiver | printerB;
12
13
     receiver.set_readv():
                                  // no more processes will be attached
14
                                    // process may start to work
     send(1): send(2): send(3):
     int end: std::cin >> end:
17 }
```

Felix Petriconi

Channel Motivation

Channel - Stateles

Channel - Split Channel - Join

Exercise 2
Channel Statefu
Process

```
Felix Petriconi
```

```
Channel
Motivation
```

Process Channel - Split

Channel - Join Exercise 2

Process

```
using namespace stlab;
  int main() {
     sender < int > send:
     receiver < int > receiver:
     std::tie(send. receiver) = channel<int>(default executor);
     auto printerA = [](int x){ printf("Process A %d\n", x); };
8
     auto printerB = [](int x){ printf("Process B %d\n", x); };
10
     auto printer_processA = receiver | printerA;
11
     auto printer_processB = receiver | printerB;
12
13
     receiver.set_readv();
                                    // no more processes will be attached
14
                                     // process may start to work
     send(1): send(2): send(3):
     int end: std::cin >> end:
17 }
```

```
Felix Petriconi
```

Channel - Split

```
using namespace stlab;
  int main() {
     sender < int > send:
     receiver < int > receiver:
     std::tie(send. receiver) = channel<int>(default executor);
     auto printerA = [](int x){ printf("Process A %d\n", x); };
8
     auto printerB = [](int x){ printf("Process B %d\n", x); };
10
     auto printer_processA = receiver | printerA;
11
     auto printer_processB = receiver | printerB;
12
13
     receiver.set_readv();
                                     // no more processes will be attached
14
                                     // process may start to work
     send(1): send(2): send(3):
     int end: std::cin >> end:
17 }
```

```
Felix Petriconi
```

```
Channel
Motivation
```

Channel - Split

Exercise 2

Channel S Process

```
using namespace stlab;
  int main() {
     sender < int > send:
     receiver < int > receiver:
     std::tie(send. receiver) = channel<int>(default executor);
     auto printerA = [](int x){ printf("Process A %d\n", x); };
8
     auto printerB = [](int x){ printf("Process B %d\n", x); };
10
     auto printer_processA = receiver | printerA;
11
     auto printer_processB = receiver | printerB;
12
13
     receiver.set_readv();
                                  // no more processes will be attached
14
                                    // process may start to work
     send(1): send(2): send(3):
     int end: std::cin >> end:
17 }
```

Channel - Split Process

send(1): send(2): send(3): int end: std::cin >> end:

auto printer_processA = receiver | printerA; auto printer_processB = receiver | printerB;

using namespace stlab;

int main() { sender < int > send: receiver <int> receiver:

8

10

12 13

14

17 3

Process A 1 Process B 1 Process A 2

Process B 2 Process B 3

Process A 3

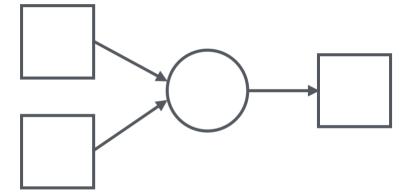
std::tie(send. receiver) = channel<int>(default executor): auto printerA = [](int x){ printf("Process A %d\n", x); }; auto printerB = [](int x){ printf("Process B %d\n", x); }; receiver.set_readv(): // no more processes will be attached // process may start to work

Channel Motivation

> hannel - Statele rocess

Channel - Split

Channel Statefu



```
Channel
Motivation
```

```
Channel - Split
```

Exercise 2

Channel Stat

```
1 using namespace stlab:
  int main() {
     sender < int > sendA . sendB:
     receiver < int > receiver A . receiver B :
     std::tie(sendA, receiverA) = channel<int>(default_executor);
     std::tie(sendB, receiverB) = channel<int>(default_executor);
8
     auto printer = [](int x, int y){ printf("Process %d %d\n", x, y); };
10
11
     auto printProcess = join(default_executor, printer,
12
       receiverA . receiverB):
13
14
     receiverA.set_ready();
15
     receiverB.set readv():
17
     sendA(1): sendA(2): sendB(3): sendA(4): sendB(5): sendB(6):
     int end: std::cin >> end:
20 }
```

```
Felix Petriconi
```

```
Motivation
```

Channel - Split Channel - Join

Exercise 2

Channel Sta Process

```
1 using namespace stlab:
  int main() {
     sender < int > sendA . sendB:
     receiver < int > receiverA . receiverB:
     std::tie(sendA, receiverA) = channel<int>(default_executor);
     std::tie(sendB, receiverB) = channel<int>(default_executor);
8
     auto printer = [](int x, int y){ printf("Process %d %d\n", x, y); };
10
11
     auto printProcess = join(default_executor, printer,
12
       receiverA. receiverB):
13
14
     receiverA.set_ready();
15
     receiverB.set readv():
17
     sendA(1): sendA(2): sendB(3): sendA(4): sendB(5): sendB(6):
     int end: std::cin >> end:
20 }
```

```
Channel
Motivation
```

```
Process
Channel - Split
Channel - Join
```

Exercise 2

```
Channel State
Process
```

```
1 using namespace stlab:
  int main() {
     sender < int > sendA . sendB:
     receiver < int > receiver A . receiver B :
     std::tie(sendA, receiverA) = channel<int>(default_executor);
     std::tie(sendB, receiverB) = channel<int>(default_executor);
8
     auto printer = [](int x, int y){ printf("Process %d %d\n", x, y); };
10
11
     auto printProcess = join(default_executor, printer,
12
       receiverA . receiverB):
13
14
     receiverA.set_ready();
15
     receiverB.set readv():
17
     sendA(1): sendA(2): sendB(3): sendA(4): sendB(5): sendB(6):
     int end: std::cin >> end:
20 }
```

```
Channel
Motivation
```

Process
Channel - Split

Channel - Join Exercise 2

Channel State

```
1 using namespace stlab:
  int main() {
     sender < int > sendA . sendB:
     receiver < int > receiver A . receiver B :
     std::tie(sendA, receiverA) = channel<int>(default_executor);
     std::tie(sendB, receiverB) = channel<int>(default_executor);
8
     auto printer = [](int x, int y){ printf("Process %d %d\n", x, y); };
10
11
     auto printProcess = join(default_executor, printer,
12
       receiverA. receiverB):
14
     receiverA.set_ready();
15
     receiverB.set readv():
17
     sendA(1): sendA(2): sendB(3): sendA(4): sendB(5): sendB(6):
     int end: std::cin >> end:
20 }
```

```
Channel
Motivation
```

Channel - Split

Exercise 2

Channel Statefu Process

```
1 using namespace stlab:
  int main() {
     sender < int > sendA . sendB:
     receiver < int > receiver A . receiver B :
     std::tie(sendA, receiverA) = channel<int>(default_executor);
     std::tie(sendB, receiverB) = channel<int>(default_executor);
8
     auto printer = [](int x, int y){ printf("Process %d %d\n", x, y); };
10
11
     auto printProcess = join(default_executor, printer,
12
       receiverA. receiverB):
14
     receiverA.set_ready();
15
     receiverB.set readv():
16
17
     sendA(1): sendA(2): sendB(3): sendA(4): sendB(5): sendB(6):
     int end: std::cin >> end:
20 }
```

```
Channel
Motivation
```

Process
Channel - Split

Channel - Join Exercise 2

Channel Statef Process

```
using namespace stlab:
  int main() {
     sender < int > sendA . sendB:
     receiver < int > receiver A . receiver B :
     std::tie(sendA, receiverA) = channel<int>(default_executor);
     std::tie(sendB, receiverB) = channel<int>(default_executor);
8
     auto printer = [](int x, int y){ printf("Process %d %d\n", x, y); };
10
11
     auto printProcess = join(default_executor, printer,
12
       receiverA . receiverB):
13
14
     receiverA.set_ready();
15
     receiverB.set readv():
17
     sendA(1): sendA(2): sendB(3): sendA(4): sendB(5): sendB(6):
     int end: std::cin >> end:
20 }
```

using namespace stlab:

sender < int > sendA . sendB:

int main() {

8

10 11

12

13 14

15

16 17

20 }

Felix Petriconi

```
Channel
Motivation
```

```
Channel - Statele
Process
```

Channel - Join Exercise 2

Channel Stat

```
receiver<int> receiverA, receiverB;
std::tie(sendA, receiverB) = channel<int>(default_executor);
std::tie(sendB, receiverB) = channel<int>(default_executor);
auto printer = [](int x, int y){ printf("Process %d %d\n", x, y); };
auto printProcess = join(default_executor, printer,
    receiverA, receiverB);
receiverA.set_ready();
receiverB.set_ready();
sendA(1); sendA(2); sendB(3); sendA(4); sendB(5); sendB(6);
int end; std::cin >> end;
```

Output

```
Process 1 3
Process 2 5
Process 4 6
```

ocess

Channel - Join

Exercise 2

Channel Statefu Process

Beside join() there are:

- ▶ zip()The process takes the passed values in a round-robin manner, starting with the result from the first receiver.
- merge()The process takes the values in an arbitrary order.

Exercise 2

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Channel Motivation

nannel - State

Channel - Spli Channel - Joir

Exercise 2

Channel Stateful Process

Demo Exercise 2

Create a process chain with

- the inputs
 - ▶ one int input
 - ▶ one std::string input
 - ▶ one double input
- ▶ all inputs are joined to a process that concatenates all the results into a string and
- the result is split into
 - one process that prints the result into console.
 - one process that stores the result into a file
- show with two value triplets, that the implementation works
- don't use any synchronization primitive

Conclusion

Felix Petriconi

Exercise 2

▶ Stateless processes (from the point of view of the channel) have a 1:1 relationship from input to output

Channel Stateful Process - Motivation

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Channel Motivation

Channel -Process

Channel State Process

Exercise

- Some problems need a processor with state
- ▶ Some problems have an n : m relationship from input to output
- ▶ The picture becomes more complicated with states:
 - ▶ When to proceed?
 - ▶ How to handle situations when less than expected values come downstream?

Channel Motivation

Channel Stateful

Exercise 3

```
#include <stlab/channel.hpp>
   using process_state_scheduled =
     std::pair cess_state, std::chrono::system_clock::time_point>;
   struct process_signature
       void await(T... val);
 9
       U yield();
12
       process_state_scheduled state() const;
13
14
       void close():
                                            // optional
16
       void set_error(std::exception_ptr): // optional
17 };
```

```
#include <stlab/channel.hpp>
   using process state scheduled =
     std::pair < process_state, std::chrono::system_clock::time_point >;
   struct process_signature
       void await(T... val);
       U vield();
       process state scheduled state() const:
14
       void close():
                                            // optional
16
       void set_error(std::exception_ptr); // optional
17 };
```

The await method is called on the process whenever a new value was received from upstream. The type T stands here for any semi regular or move-only type. The number of arguments depends on the number of attached upstream sender. Potential state changes from awaitable to yieldable should happen while this method is invoked

The yield method is called on the process whenever the process_state_scheduled.first is process_state::yield or a timeout was provided with the recent call to state() and that has elapsed.

Stateful Process Signature - state

```
#include <stlab/channel.hpp>
   using process_state_scheduled =
     std::pair < process_state, std::chrono::system_clock::time_point >;
   struct process_signature
       void await(T... val):
       U vield();
       process state scheduled state() const:
13
14
       void close():
                                            // optional
16
       void set_error(std::exception_ptr); // optional
17 };
```

This method must return the current state of the process. Typical return values are await forever and vield immediate. By explicit using the second part of the return type, one can set a possible timeout. Subsequent calls without an intermittent await(), close(), or yield() must return the same values. Otherwise the result is undefined

```
#include <stlab/channel.hpp>
   using process_state_scheduled =
     std::pairrocess_state, std::chrono::system_clock::time_point>;
   struct process_signature
       void await (T... val):
10
       U vield():
       process state scheduled state() const:
13
14
       void close():
                                            // optional
16
       void set error(std::exception ptr): // optional
17 }:
```

The optional close() method is called on the process whenever the process state is await_forever and the incoming queue went dry. As well it is called when an exception is thrown while calling await() or yield() and no set_error() is available.

Felix Petriconi

Channel Motivation

Process

Exercise 3

```
#include <stlab/channel.hpp>
   using process state scheduled =
     std::pairrcess_state, std::chrono::system_clock::time_point>;
   struct process_signature
       void await (T. . . val) ·
10
       U vield();
11
       process_state_scheduled state() const;
13
14
       void close():
                                            // optional
16
       void set_error(std::exception_ptr); // optional
17 F:
```

The method set_error() is optional. It is called if either on calling await() or yield() an exception was thrown. The pointer of the caught exception is passed. In case that the process does not provide this method, close() is called instead of.

```
#include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
   using namespace stlab:
  struct adder
9
  int main() {
     sender < int > send:
     receiver <int> receiver:
     std::tie(send. receiver) = channel<int>(default executor):
14
15
     auto calculator = receiver | adder{} |
16
       [](int x) { std::cout << x << '\n': }:
17
18
     receiver.set_readv():
19
20
     while (true) {
       int x:
       std::cin >> x:
       send(x):
24
25 }
```

Channel Motivation

Process

Exercise 3

#include <stlab/channel.hpp> #include <stlab/default_executor.hpp>

#include <iostream> using namespace stlab:

> sender < int > send: receiver <int> receiver:

receiver.set_readv():

while (true) { int x: std::cin >> x: send(x):

auto calculator = receiver | adder{} |

[](int x) { std::cout << x << '\n': }:

struct adder

10 int main() {

9

14 15

16

17 18

19 20

24 25 3

Felix Petriconi

```
std::tie(send. receiver) = channel<int>(default executor):
```

Channel - Stateful Process Example

```
#include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
   using namespace stlab:
  struct adder
9
10 int main() {
     sender < int > send:
     receiver <int> receiver:
13
     std::tie(send. receiver) = channel<int>(default executor):
14
15
     auto calculator = receiver | adder{} |
16
       [](int x) { std::cout << x << '\n': }:
17
18
     receiver.set_readv():
19
20
     while (true) {
       int x:
       std::cin >> x:
       send(x):
24
25 }
```

Felix Petriconi

Channel Motivation

Channel - State

Channel Stateful Process

```
Channel
Motivation
```

Process
Channel Stateful

Exercise 3

```
#include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
   using namespace stlab:
  struct adder
9
  int main() {
     sender < int > send:
     receiver <int> receiver:
     std::tie(send, receiver) = channel<int>(default_executor);
14
15
     auto calculator = receiver | adder{} |
16
       [](int x) { std::cout << x << '\n': }:
17
18
     receiver.set_readv():
19
20
     while (true) {
       int x:
       std::cin >> x:
       send(x):
24
25 3
```

```
Channel
Motivation
```

Process

Channel Stateful

Exercise 3

```
1 #include <stlab/channel.hpp>
  #include <stlab/default_executor.hpp>
  #include <iostream>
   using namespace stlab:
  struct adder
9
  int main() {
     sender < int > send:
     receiver <int> receiver:
     std::tie(send. receiver) = channel<int>(default executor):
14
15
     auto calculator = receiver | adder{} |
16
       [](int x) { std::cout << x << '\n': }:
17
18
     receiver.set_readv():
19
20
     while (true) {
21
       int x:
       std::cin >> x:
       send(x):
24
```

nannei lotivation

hannel Stateful

```
struct adder
     int sum = 0:
     process state scheduled state = await forever:
     void await(int x) {
       sum += x:
       if (x == 0) {
         _state = yield_immediate;
10
11
13
     int vield() {
       int result = _sum;
       sum = 0:
16
       _state = await_forever:
       return result:
18
20
     auto state() const { return _state: }
21
   int main() {
     auto calculator = receiver | adder{} |
       [](int x) { std::cout << x << '\n'; };</pre>
     while (true) {
26
       int x:
       std::cin >> x:
29
       send(x):
30
31 }
```

Felix Petriconi

Channel Motivation

> hannel Stateful rocess

```
int sum = 0:
     process state scheduled state = await forever:
     void await(int x) {
       sum += x:
       if (x == 0) {
         _state = yield_immediate;
10
13
     int vield() {
       int result = _sum;
       sum = 0:
16
       _state = await_forever:
       return result:
18
20
     auto state() const { return _state: }
21
  int main() {
     auto calculator = receiver | adder{} |
       [](int x) { std::cout << x << '\n'; };</pre>
     while (true) {
26
       int x:
       std::cin >> x:
29
       send(x):
30
31 }
```

Felix Petriconi

otivation

nannel Stateful ocess

```
int sum = 0:
     process state scheduled state = await forever:
     void await(int x) {
       sum += x:
      if (x == 0) {
         _state = yield_immediate;
10
13
     int vield() {
      int result = _sum;
       sum = 0:
16
       _state = await_forever:
       return result:
18
20
     auto state() const { return _state: }
21
  int main() {
     auto calculator = receiver | adder{} |
       [](int x) { std::cout << x << '\n': }:
     while (true) {
26
      int x:
       std::cin >> x:
29
       send(x):
30
31 }
```

9

10

16

18 20

21

26

29

30

```
int sum = 0:
     process state scheduled state = await forever:
     void await(int x) {
      sum += x:
      if (x == 0) {
        _state = yield_immediate;
     int vield() {
      int result = _sum;
       sum = 0:
       _state = await_forever:
       return result:
     auto state() const { return _state: }
  int main() {
     auto calculator = receiver | adder{} |
       [](int x) { std::cout << x << '\n': }:
     while (true) {
      int x:
      std::cin >> x:
       send(x):
31 }
```

```
int sum = 0:
     process state scheduled state = await forever:
     void await(int x) {
       sum += x:
      if (x == 0) {
         _state = yield_immediate;
10
11
     int vield() {
14
       int result = _sum;
15
       sum = 0:
16
       _state = await_forever:
       return result:
18
19
20
     auto state() const { return _state: }
21
   int main() {
     auto calculator = receiver | adder{} |
       [](int x) { std::cout << x << '\n': }:
26
     while (true) {
       int x:
       std::cin >> x:
29
       send(x):
30
31 }
```

```
int sum = 0:
     process state scheduled state = await forever:
     void await(int x) {
       _sum += x;
       if (x == 0) {
 9
         _state = yield_immediate;
10
11
13
     int vield() {
       int result = _sum;
       sum = 0:
16
       _state = await_forever:
       return result:
18
20
     auto state() const { return _state: }
21
   int main() {
     auto calculator = receiver | adder{} |
       [](int x) { std::cout << x << '\n': }:
26
     while (true) {
       int x:
       std::cin >> x:
29
       send(x):
30
31 }
```

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Channel Motivation

Process

rocess

Exercise 3

Demo Exercise 3



Felix Petriconi

Channel Motivation

Channel - Stateles
Process

Channel Stateful Process

Exercise 3

The Art of Writing
Reasonable
Concurrent Code

 $^{^2 \}hbox{By Ventriloquist - Own work, CC BY-SA 3.0, $https://en.wikipedia.org/w/index.php?curid=32753387}$



A process which inputs cards of eighty characters and outputs their text, tightly packed into lines of 125 characters each.

- ▶ Write one process unpack that collect 80 chars in a bunch and yields them one after the other
- ▶ Write one process pack that packs 125 chars and yields them.
- Concatenate unpack pack as a process chain.
- ▶ In a next step write one process filter that drops all newlines from the stream
- ► Concatenate now unpack filter pack as a process chain.

Reasonable Concurrent Code

Felix Petriconi

Channel Motivation

Process

Frocess

Exercise 3

By Ventriloquist - Own work, CC BY-SA 3.0, https://en.wikipedia.org/w/index.php?curid=32753387

Process Analysis

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Process Analysis Example Use Cases Exercise 4

- Are there performance or usability problems?
- Identify the overall critical part
- Disassemble this part into individual processes
- ► Chain the processes with futures or channels

Use case example I

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Example Use Cases
Exercise 4

Use case example I

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Process Analysis
Example Use Cases
Exercise 4

Problem within our mammography application:

lacktriangle Medical device shall open every case in $< 1~{
m s}$

- lacktriangle Medical device shall open every case in $< 1~{
 m s}$
- Loading of patient data and first images takes about 0.6 s

- ▶ Medical device shall open every case in < 1 s
- ▶ Loading of patient data and first images takes about 0.6 s
- ► Reading of additional data structures (CAD³reports) may take more than 0.4 s

³Computer Aided Detection

- lacktriangle Medical device shall open every case in $< 1~{
 m s}$
- ▶ Loading of patient data and first images takes about 0.6 s
- ► Reading of additional data structures (CAD³reports) may take more than 0.4 s
- Direct access to any CAD report might be required

³Computer Aided Detection

- ▶ Medical device shall open every case in < 1 s
- ▶ Loading of patient data and first images takes about 0.6 s
- ► Reading of additional data structures (CAD³reports) may take more than 0.4 s
- Direct access to any CAD report might be required
- ▶ If the user skips this case and advances to the next one, outstanding load operations should be cancelled or at least be ignored

³Computer Aided Detection

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

Process Analysis
Example Use Cases
Exercise 4

Demo Exercise 4

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

Process Analysis
Example Use Cases
Exercise 4

Improve the application that the UI is always responsible

- On Reset the reports are newly read
- ▶ If one presses 1 or 2 while the reset is running, the reports shall be displayed as soon as they become available.

High level concurrency sessions at ACCU 2017:

- ▶ Thinking Outside the Synchronisation Quadrant by Kevlin Henney (Wed.)
- Coroutines in Python by Robert Smallshire (Thur.)
- Coroutines in C++ by Dominic Robinson (Fr.)
- Concurrency / Coroutines by Anthony Williams (Sat.)

Why do we have to synchronize?

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

ynchronization

nchronization th Mutex

nchronization thout Mutex

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Synchronization

ynchronization ith Mutex

Synchronization without Mutex

Why do we have to synchronize? Because we have to ensure sequential consistency.

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

Synchronization

ynchronization ith Mutex

Synchronization without Mutex

Why do we have to synchronize? Because we have to ensure sequential consistency. What synchronization mechanism do you know?

Reasonable

Concurrent Code

Felix Petriconi

Synchronization

ynchronization ith Mutex

Synchronization without Mutex

Why do we have to synchronize? Because we have to ensure sequential consistency. What synchronization mechanism do you know?

▶ Synchronization primitives (mutex, atomic, memory fence, ...)

Reasonable

Concurrent Code

Felix Petriconi

Synchronization

ynchronization ith Mutex

without Mutex

Why do we have to synchronize?

Because we have to ensure sequential consistency.

What synchronization mechanism do you know?

- ▶ Synchronization primitives (mutex, atomic, memory fence, ...)
- Guaranteed sequential access

ynchronizatioi vithout Mutex

```
1 template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
               guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
       );
13
14
     V operator[](const K& key) {
       unique_lock < mutex > lock(_guard);
15
16
       return data[kev]:
18 };
   int main() {
     registry < string, int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?", 42); };
     auto f1 = async(launch::async, work);
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer? " << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

```
template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
               guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
       );
13
14
     V operator[](const K& key) {
       unique_lock < mutex > lock(_guard);
15
16
       return data[kev]:
17
18 };
   int main() {
     registry < string, int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?", 42); };
     auto f1 = async(launch::async, work);
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer? " << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

Synchronization

Synchronization with Mutex

ynchronization ithout Mutex

```
1 template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
               guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
       );
13
14
     V operator[](const K& key) {
       unique_lock < mutex > lock(_guard);
15
16
       return data[kev]:
17
18 };
   int main() {
     registry < string, int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?", 42); };
     auto f1 = async(launch::async, work);
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer? " << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

Synchronization

Synchronization with Mutex

Synchronization vithout Mutex

```
Felix Petriconi

Synchronization
Synchronization
with Mutex

Synchronization
```

```
1 template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
               guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
11
       );
13
14
     V operator[](const K& key) {
       unique_lock < mutex > lock(_guard);
15
16
       return data[kev]:
18 };
   int main() {
     registry < string, int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?", 42); };
     auto f1 = async(launch::async, work);
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer? " << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

ynchronizatior vithout Mutex

```
1 template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
               guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
       );
13
14
     V operator[](const K& key) {
       unique_lock < mutex > lock(_guard);
16
       return data[kev]:
18 };
   int main() {
     registry < string, int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?", 42); };
     auto f1 = async(launch::async, work);
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer? " << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

ynchronizatioi vithout Mutex

```
1 template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
               guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
       );
13
14
     V operator[](const K& key) {
       unique_lock < mutex > lock(_guard);
15
16
       return data[kev]:
18 };
   int main() {
     registry < string , int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?". 42): }:
     auto f1 = async(launch::async, work);
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer? " << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

Synchronization vithout Mutex

```
1 template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
                guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
       );
13
14
     V operator[](const K& key) {
       unique_lock < mutex > lock(_guard);
15
16
       return data[kev]:
18 };
   int main() {
     registry < string, int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?", 42); };
     auto f1 = async(launch::async, work);
24
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer? " << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

Felix Petriconi

vnchronization

Synchronization with Mutex

ynchronization ithout Mutex

```
1 template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
               guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
       );
13
14
     V operator[](const K& key) {
15
       unique_lock < mutex > lock(_guard);
16
       return data[kev]:
17
18 };
   int main() {
     registry < string, int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?", 42); };
     auto f1 = async(launch::async, work);
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer?" << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

Where are the problems in the code?

```
1 template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
               guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
       );
13
14
     V operator[](const K& key) {
15
       unique_lock < mutex > lock(_guard);
16
       return data[kev]:
17
18 };
   int main() {
     registry < string, int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?", 42); };
     auto f1 = async(launch::async, work);
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer?" << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

Where are the problems in the code?

```
1 template <typename K, typename V>
   class registry
     map < K. V > data:
     mutex
               guard:
   public:
     void insert(const K& key, const V& value) {
       unique_lock < mutex > lock(_guard);
 9
       _data.insert(
         make_pair("What is the answer?", 42)
       );
14
     V operator[](const K& key) {
15
       unique_lock < mutex > lock(_guard);
16
       return data[kev]:
17
18 };
   int main() {
     registry < string, int > my_registry;
     auto work = [&] { my_registry.insert("What is the answer?", 42); };
     auto f1 = async(launch::async, work);
     auto f2 = asvnc(launch::asvnc, work);
     f1.get(); f2.get();
26
     cout << "What is the answer?" << my_registry["What is the answer?"] << '\n';</pre>
27 1
```

Where are the problems in the code?

chronization

nchronization

nchronization thout Mutex

Reasonable
Concurrent Code

Felix Petriconi

ynchronization

Synchronization with Mutex

ynchronizatior vithout Mutex

Mutex - What would be a better name for it?

Reasonable Concurrent Code

Felix Petriconi

chronization

nchronization th Mutex

chronization nout Mutex

Mutex - What would be a better name for it?

Bottleneck!⁴

⁴Kevlin Henney, NDC London 2017

Reasonable
Concurrent Code

Felix Petriconi

ynchronization

nchronization th Mutex

Synchronization vithout Mutex

How can the code be transformed into something without a mutex in the client code?

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

vnchronization

nchronizatio th Mutex

Synchronization without Mutex

How can the code be transformed into something without a mutex in the client code?

What is needed to perform that transformation? Which tools do we have in our tool box?

Felix Petriconi

Synchronization

synchronizati with Mutex

Synchronization without Mutex

```
template <tvpename K. tvpename V>
   class registry
     std::shared_ptr<map<K, V>> _data;
     serial_queue
                                  _queue;
   public:
     void insert(K kev. V val) {
 8
       _queue.async([_d = _data,
                      _kev = std::move(kev),
10
                     val = std::move(val)] {
           d->emplace(std::move(key), std::move(val));
11
12
       }):
13
14
     future < V > operator[](K kev) {
16
       return _queue.async([_d = _data,
                             kev = std::move(kev)] {
18
         return _d->at(kev):
19
       }):
20
21 }:
```

```
template <tvpename K. tvpename V>
   class registry
     std::shared_ptr<map<K, V>> _data;
     serial_queue
                                  _queue;
   public:
     void insert(K kev. V val) {
 8
       _queue.async([_d = _data,
                      _kev = std::move(kev),
10
                     val = std::move(val)] {
           d->emplace(std::move(key), std::move(val));
11
12
       }):
13
14
     future < V > operator[](K kev) {
16
       return _queue.async([_d = _data,
                             kev = std::move(kev)] {
18
         return _d->at(kev):
19
       }):
20
21 }:
```

```
template <typename K. typename V>
   class registry
     std::shared_ptr<map<K, V>> _data;
     serial_queue
                                  _queue;
   public:
     void insert(K kev. V val) {
 8
       _queue.async([_d = _data,
                      _kev = std::move(kev),
10
                     val = std::move(val)] {
           d->emplace(std::move(key), std::move(val));
11
12
       }):
13
14
     future < V > operator[](K kev) {
16
       return _queue.async([_d = _data,
                             kev = std::move(kev)] {
18
         return _d->at(kev):
19
       }):
20
21 }:
```

Felix Petriconi

Synchronization

oynchronizati with Mutex

Synchronizatioi without Mutex

```
template <typename K, typename V>
   class registry
     std::shared_ptr<map<K, V>> _data;
     serial_queue
                                  _queue;
   public:
     void insert(K kev. V val) {
 8
       _queue.async([_d = _data,
                      _kev = std::move(kev),
10
                     val = std::move(val)] {
           d->emplace(std::move(key), std::move(val));
11
12
       }):
13
14
     future < V > operator[](K kev) {
16
       return _queue.async([_d = _data,
                             kev = std::move(kev)] {
18
         return _d->at(kev):
19
       }):
20
21 }:
```

Synchronization without Mutex

```
template <typename K, typename V>
   class registry
     std::shared_ptr<map<K, V>> _data;
     serial_queue
                                  _queue;
   public:
     void insert(K kev. V val) {
 8
       _queue.async([_d = _data,
                      _kev = std::move(kev),
10
                     val = std::move(val)] {
11
           d->emplace(std::move(key), std::move(val));
12
       });
13
14
15
     future < V > operator[](K kev) {
16
       return _queue.async([_d = _data,
                             kev = std::move(kev)] {
18
         return _d->at(kev):
19
       }):
20
21 }:
```

Felix Petriconi

Synchronization

Synchronizatic vith Mutex

Synchronization without Mutex

```
template <typename K. typename V>
   class registry
     std::shared_ptr<map<K, V>> _data;
     serial_queue
                                  _queue;
   public:
     void insert(K kev. V val) {
8
       _queue.async([_d = _data,
                      _kev = std::move(kev),
10
                     val = std::move(val)] {
           d->emplace(std::move(key), std::move(val));
11
12
       }):
13
14
15
     future < V > operator [] (K kev) {
       return _queue.async([_d = _data,
17
                             kev = std::move(kev)] {
18
         return _d->at(kev):
19
       }):
20
21
```

Synchronization Epilogue

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

ynchronization

Synchronization with Mutex

Synchronizatioi without Mutex

So we try to avoid mutexes wherever it is possible.

Synchronization Epilogue

The Art of Writing
Reasonable
Concurrent Code

Felix Petriconi

ynchronization

Synchronization with Mutex

Synchronization without Mutex

All computer wait at the same speed

Image Preparation Pipeline

- ▶ A medial device shall display multi-frame image data sets
- ► Each incoming data set is JPEG 2000 compressed
- ► The slices must be decompressed and then compressed in FELICS⁵ format for fast decompression and display
- Reading and writing to disk takes a reasonable amount of time

⁵Special compression algorithm for 16bit grayscale images

- Concurrency library https://github.com/stlab/libraries
- Documentation http://www.stlab.cc/libraries
- ► Communicating Sequential Processes by C. A. R. Hoare http://usingcsp.com/cspbook.pdf
- ▶ The Theory and Practice of Concurrency by A.W. Roscoehttp: //www.cs.ox.ac.uk/people/bill.roscoe/publications/68b.pdf

Acknowledgement

Contact

Software Principles and Algorithms

- Elements of Programming by Alexander Stepanov, Paul McJones, Addison Wesley
- ► From Mathematics to Generic Programming by Alexander Stepanov, Daniel Rose, Addison Wesley

Concurrency and Parallelism

- ► HPX http://stellar-group.org/libraries/hpx/
- ► C++CSP https://www.cs.kent.ac.uk/projects/ofa/c++csp
- ► CAF_C++ Actor Framework http://actor-framework.org/
- ► C++ Concurrency In Action by Anthony Williams, Manning (2nd edition coming soon)

- ► Goals for better code by Sean Parent: http://sean-parent.stlab.cc/papers-and-presentations
- ► Goals for better code by Sean Parent: Concurrency: https://youtu.be/au0xX4h8SCI?t=16354
- ► Thinking Outside the Synchronization Quadrant by Kevlin Henney: https://vimeo.com/205806162

Acknowledgement Contact

- ▶ My family, who gave me the freedom to develop over months the library, prepare this tutorial and let me travel to the ACCU.
- Sean Parent, who taught me over time lots about concurrency and abstraction. He gave me the permission to use whatever I needed from his presentations for my own.
- My company MeVis Medical Solutions AG, that released me from work during the ACCU.

Contact

The Art of Writing Reasonable Concurrent Code

Felix Petriconi

ference

cknowledgem

ontact

► Mail: felix@petriconi.net

▶ Web: https://petriconi.net

► Twitter: @FelixPetriconi