Introduction to the Coroutines TS

Part I

Toby Allsopp
toby@mi6.gen.nz

Auckland C++ Meetup
9 August 2017

Overview

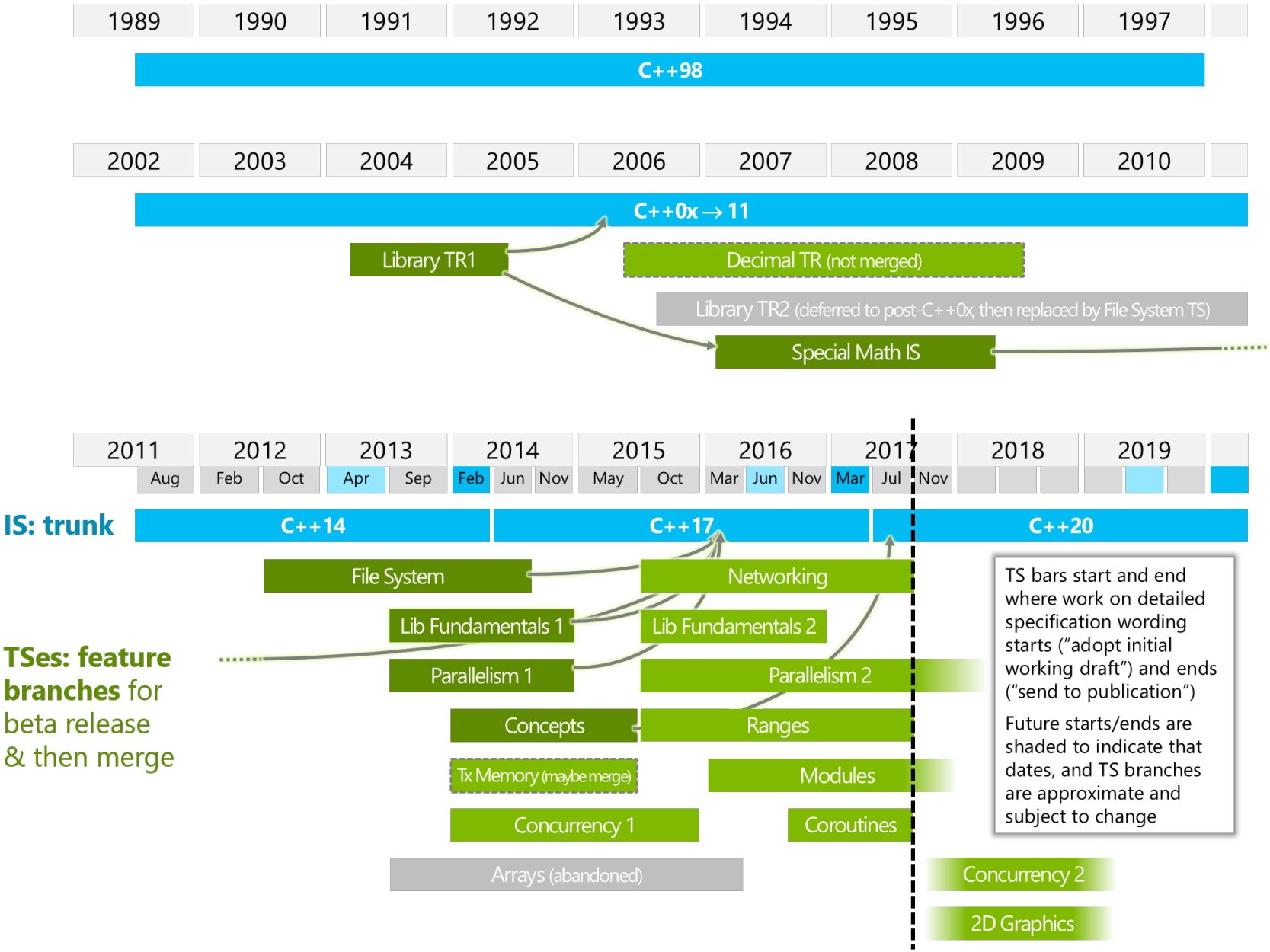
The Technical Specification

What’s a Coroutine?

What are they good for?

The Technical Specification

What's a Technical Specification?



The Coroutines Technical Specification

- *Programming Languages — C++ Extensions for Coroutines*
- PDTS Draft: [N4663] (published 2017-03-25)
- TS Draft: [N4680] (not yet published)
- Championed by Gor Nishanov (MSFT)
- Voted for publication at the July ISO C++ committee meeting
  - All national body comments have been addressed
  - No guarantee it will be included in C++20

The Coroutines Technical Specification

- *Programming Languages — C++ Extensions for Coroutines*
- PDTS Draft: [N4663] (published 2017-03-25)
- TS Draft: [N4680] (not yet published)
- Championed by Gor Nishanov (MSFT)
- Voted for publication at the July ISO C++ committee meeting
  - All national body comments have been addressed
  - No guarantee it will be included in C++20
- Implemented in MSVC and Clang
  - Visual Studio 2015 (minor differences wrt. TS)
  - Clang trunk (will be version 5.0 I think)

What’s a Coroutine?

What are these things?

- Lots of subtly different things in many different languages are called “coroutines”
- It’s important not to get confused by experience with other kinds of coroutines

What are these things?

- Lots of subtly different things in many different languages are called “coroutines”
- It’s important not to get confused by experience with other kinds of coroutines
- Generalization of functions — coroutines *are* functions
- Can be suspended and resumed

Prior Art

- Boost.Coroutine
- Boost.Coroutine2

Other Proposals

- call/cc

What are they good for?

Canonical coroutine applications

- Generators
- Futures
- Async streams

Example: Printing Primes

```
1  struct prime_tester {
2    vector<int> primes;
3
4    bool test(int n) {
5      if (none_of(primes.begin(), primes.end(),
6              [n](int p) { return n%p == 0; })) {
7        primes.push_back(n);
8        return true;
9      }
10     return false;
11   }
12 }
```

Example: Printing Primes

```
1  template <typename ShouldStop, typename Print>
2  void print_primes(ShouldStop should_stop,
3                    Print print) {
4    prime_tester tester;
5    for (int n = 2; ; ++n) {
6      if (should_stop()) break; // <-- UGLY
7      if (tester.test(n))
8        print(n);
9    }
10 }
```

Primes generator

```
1
2  generator<int> primes() {
3
4    prime_tester tester;
5    for (int n = 2; ; ++n)
6
7      if (tester.test(n))
8        co_yield n;
9
10 }
```

Primes generator

- To use the generator, just iterate over it.

```
1 for (int n : primes()) {
2   cout << n << "\n";
3   if (n >= 100) break;
4 }
```

- This is nice and clean: `primes` is only concerned with generating all the primes and all the logic about what to do with them is external

Primes generator

```
1 generator<int> primes() {
2   prime_tester tester;
3   for (int n = 2; ; ++n)
4     if (tester.test(n))
5       co_yield n;
6   }
7 }
```

- **co_yield** is the first of three new keywords
- `generator<T>` is a class template with `begin()` and `end()` members
- `generator` is not provided by the TS but you can make it yourself

Generator composability

```
generator<int> ints(int from) {
  for (int i = from; ; ++i)
    co_yield i;
}

template <typename Range, typename Pred>
generator<int> filter(Range range, Pred pred) {
  for (auto&& x : range)
    if (pred(x))
      co_yield x;
}

generator<int> primes() {
  return filter(ints(2), is_prime);
}
```

Example: Reading from a Socket

Synchronously:

```
1        string  connect_and_read(string host,
2                                 int port) {
3   string result;
4   auto socket =           connect(host, port);
5   std::array<char, 1024> buffer;
6   while (int nread =
7                        socket.read(buffer))
8     result.append(buffer, nread);
9      return result;
10 }
```

Example: Reading from a Socket

Asynchronously:

```
1  future<string> connect_and_read(string host,
2                                  int port) {
3    string result;
4    auto socket = co_await connect(host, port);
5    std::array<char, 1024> buffer;
6    while (int nread =
7              co_await socket.read(buffer))
8      result.append(buffer, nread);
9    co_return result;
10 }
```

- Two more new keywords: **`co_await`** and **`co_return`**
- `future` could be `std::future` or any other class that has the coroutine customization points implemented

Example: Fetching Primes Asynchronously

```
1  future<bool> is_prime(int n);
2
3  stream<int> primes() {
4    for (int n : ints(2))
5      if (co_await is_prime(n))
6        co_yield n;
7  }
8
9  bool stop;
10 for co_await (int n : primes()) {
11   cout << n << "\n";
12   cin >> stop;
13   if (stop) break;
14 }
```