C++ Coroutines

a negative overhead abstraction

Gor Nishanov • gorn@microsoft.com

1958

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLOWORLD.
000300*
000400 ENVIRONMENT DIVISION.
000500 CONFIGURATION SECTION.
000600 SOURCE-COMPUTER. RM-COBOL.
000700 OBJECT-COMPUTER. RM-COBOL.
000800
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400 DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500 DISPLAY "Hello world!" LINE 15 POSITION 10.
100600 STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800 EXIT.
```

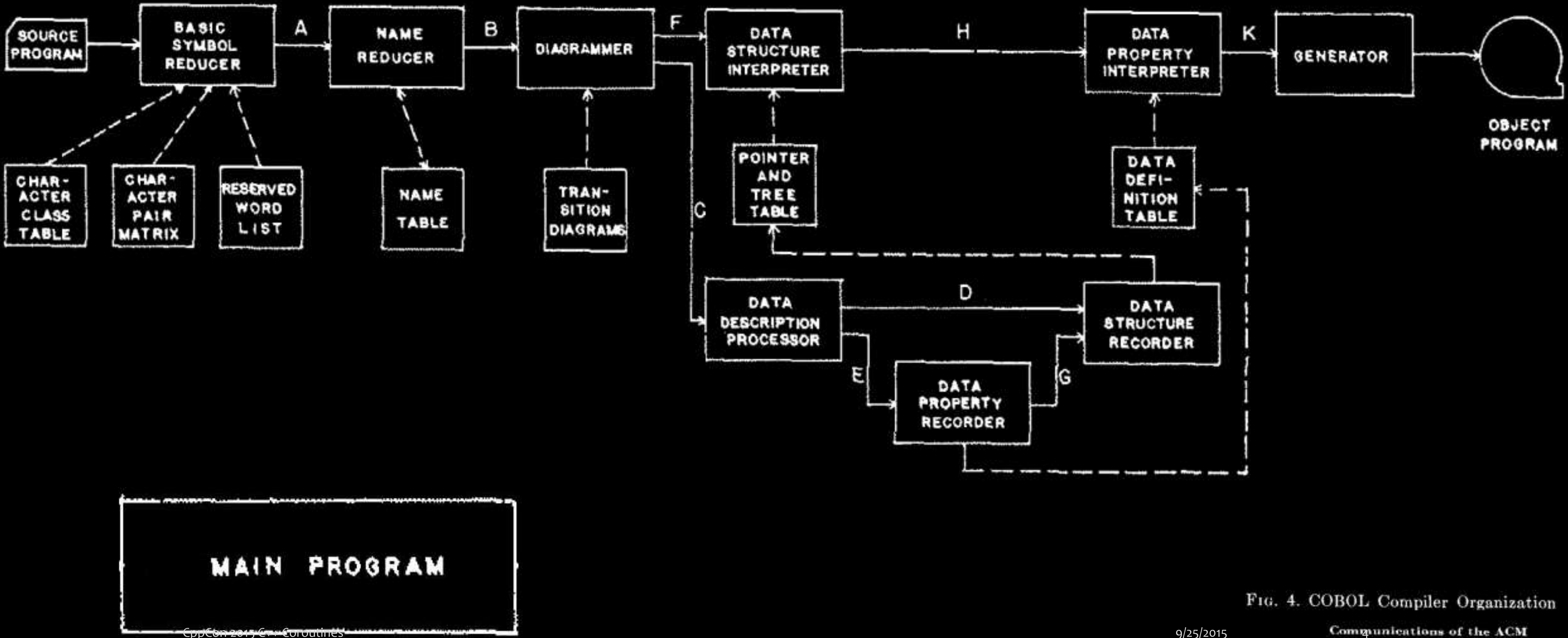
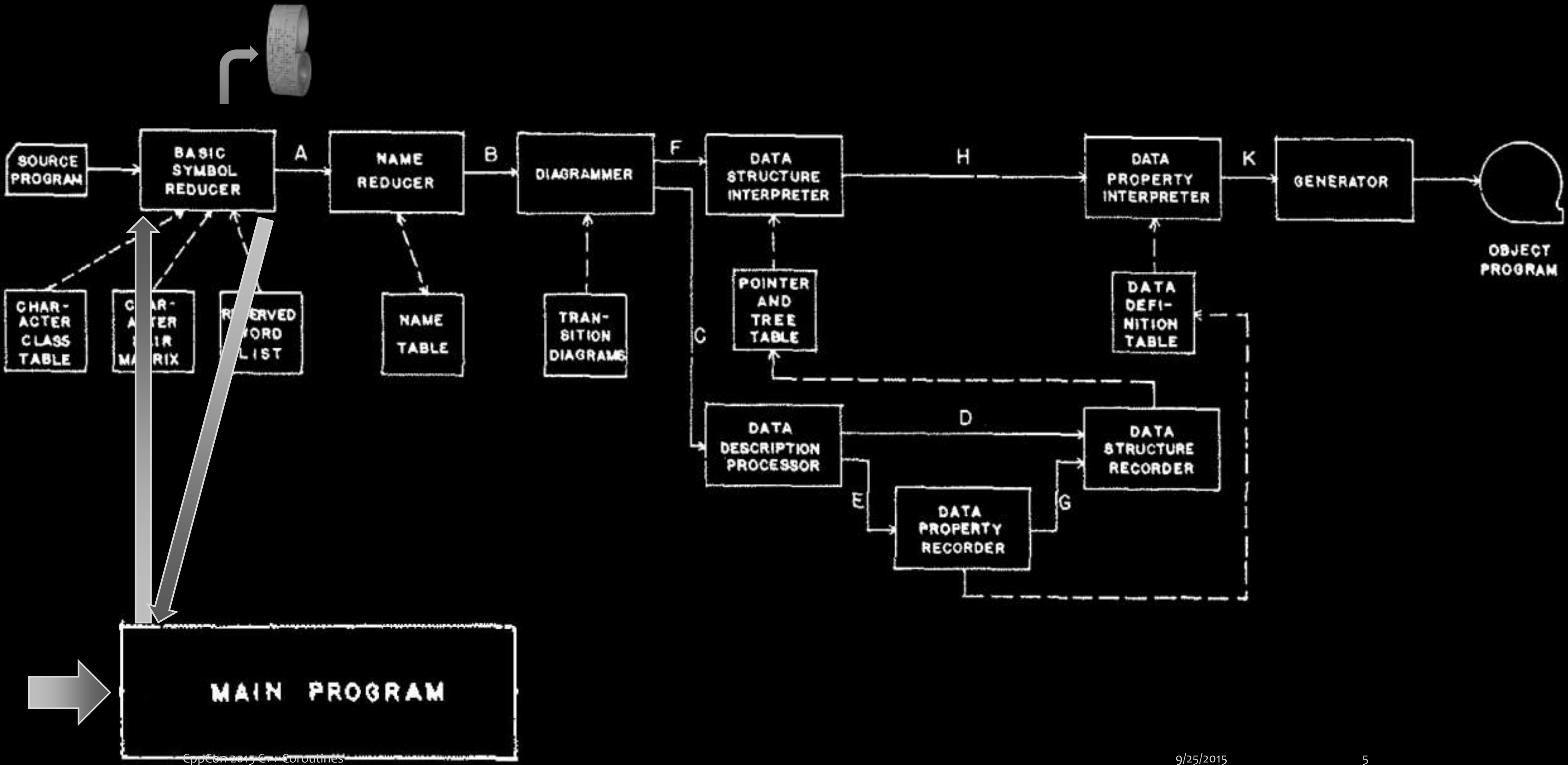
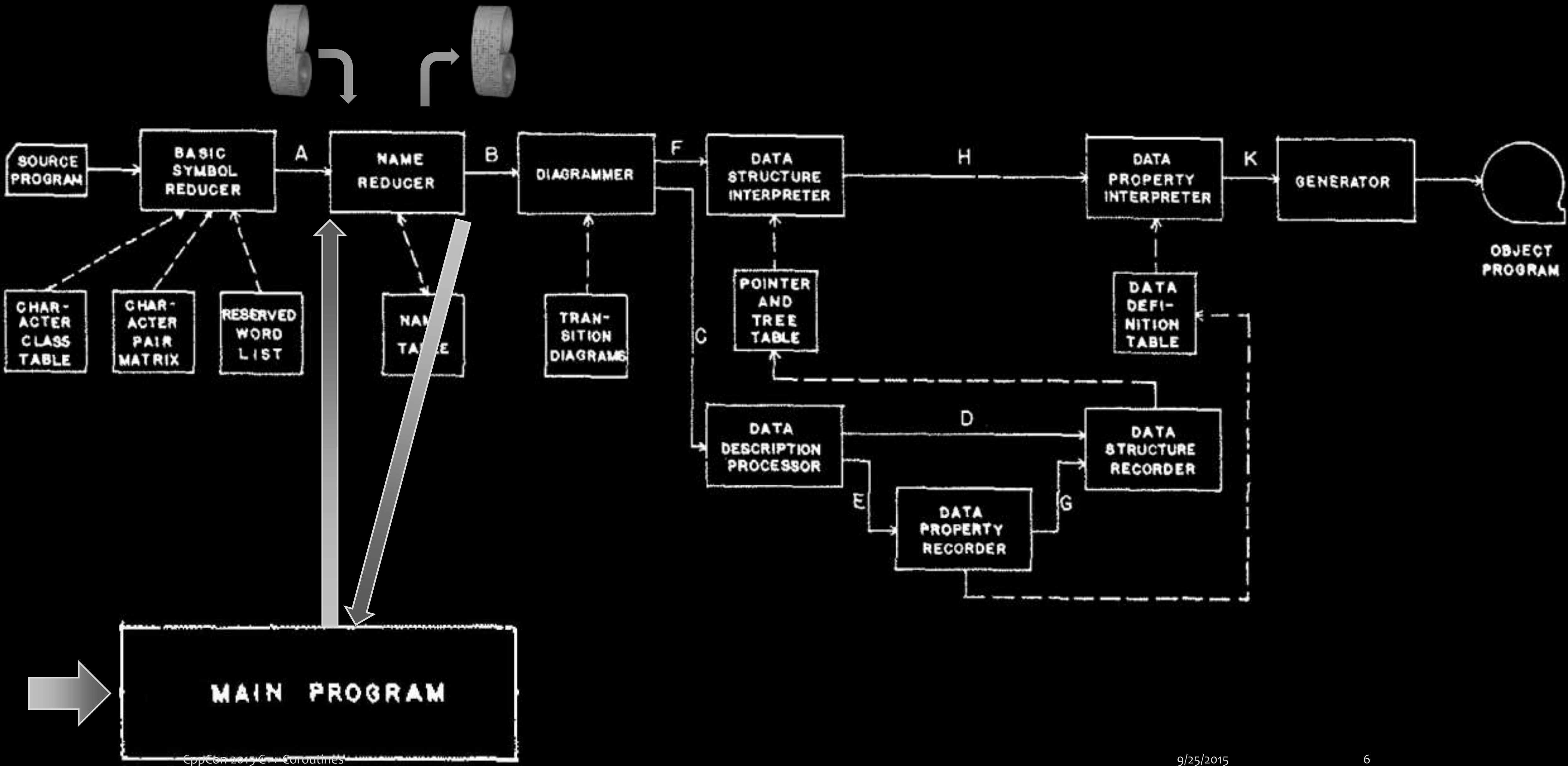
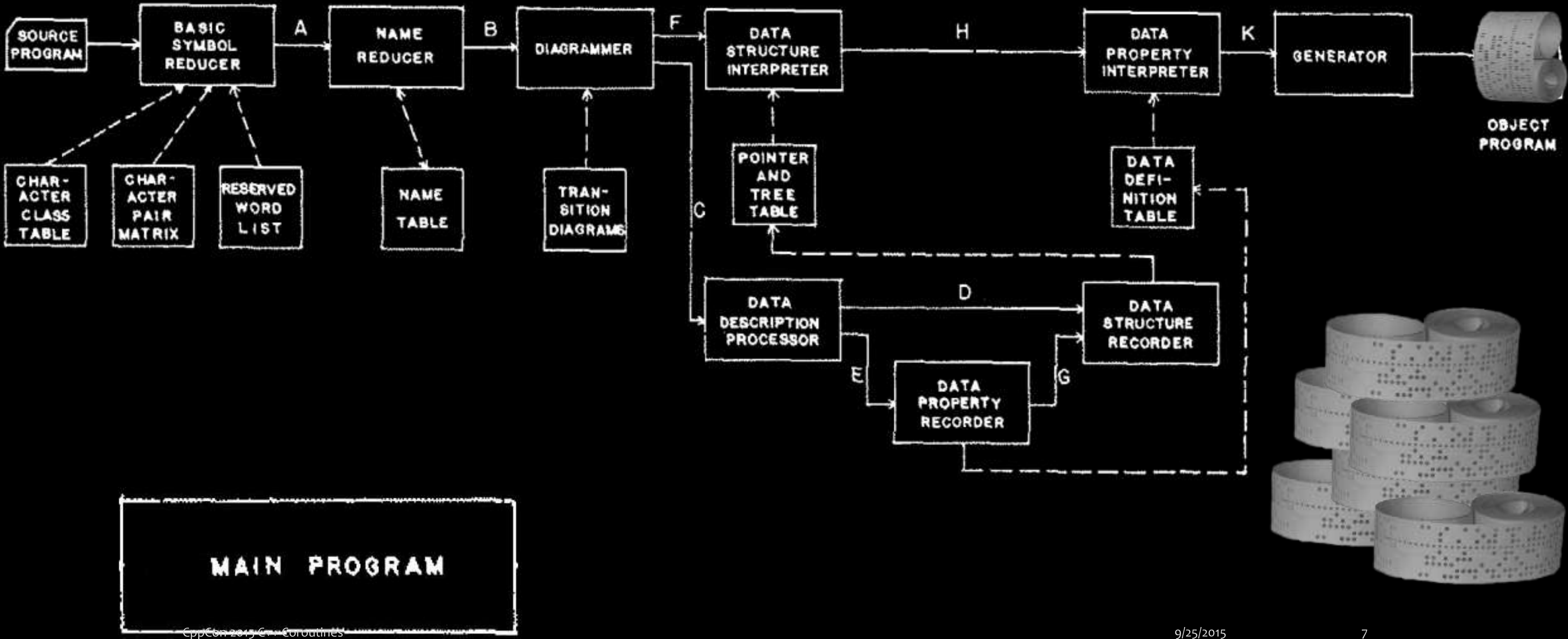
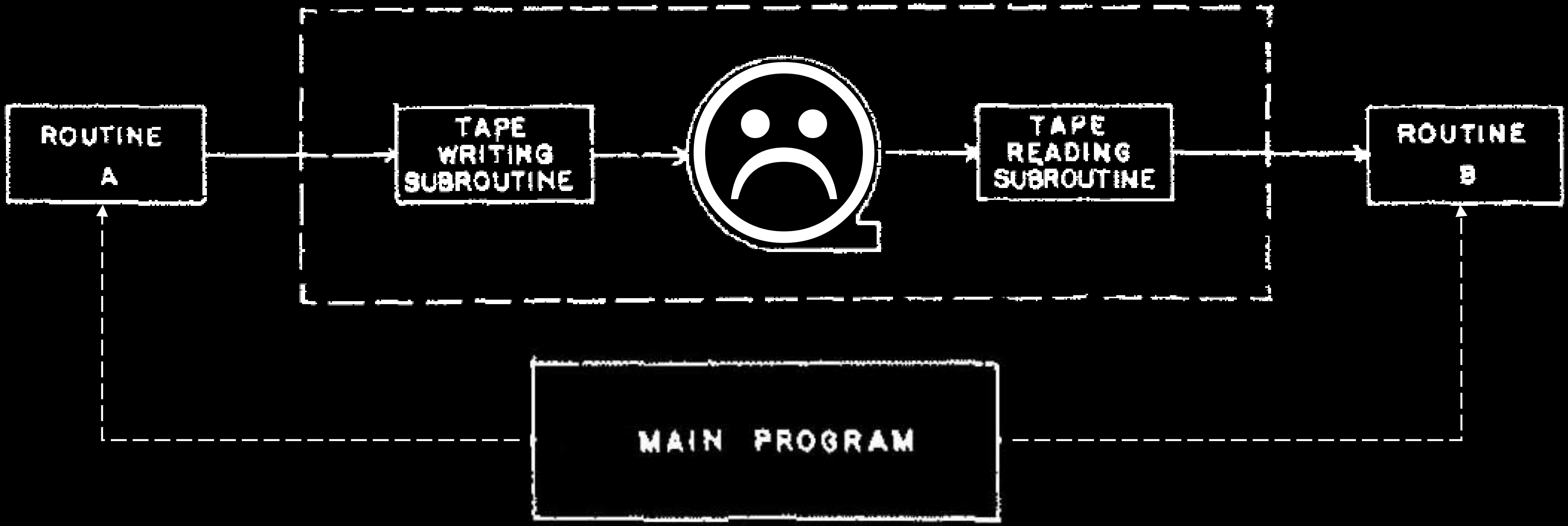


FIG. 4. COBOL Compiler Organization


SOURCE PROGRAM
BASIC SYMBOL REDUCER
A
NAME REDUCER
B
DIAGRAMMER
F
DATA STRUCTURE INTERPRETER
H
DATA PROPERTY INTERPRETER
K
GENERATOR
OBJECT PROGRAM
CHARACTER CLASS TABLE
CHARACTER PAIR MATRIX
RESERVED WORD LIST
NAME TABLE
TRANSITION DIAGRAMS
C
POINTER AND TREE TABLE
DATA DEFINITION TABLE
DATA DESCRIPTION PROCESSOR
D
DATA STRUCTURE RECORDER
E
DATA PROPERTY RECORDER
G
MAIN PROGRAM


SOURCE PROGRAM
BASIC SYMBOL REDUCER
A
NAME REDUCER
B
DIAGRAMMER
F
DATA STRUCTURE INTERPRETER
H
DATA PROPERTY INTERPRETER
K
GENERATOR
OBJECT PROGRAM
CHAR-ACTER CLASS TABLE
CHAR-ACTER PAIR MATRIX
RESERVED WORD LIST
NAME TABLE
TRAN-SITION DIAGRAMS
C
POINTER AND TREE TABLE
DATA DEFI-NITION TABLE
DATA DESCRIPTION PROCESSOR
D
DATA STRUCTURE RECORDER
E
DATA PROPERTY RECORDER
G
MAIN PROGRAM


SOURCE PROGRAM
BASIC SYMBOL REDUCER
A
NAME REDUCER
B
DIAGRAMMER
F
DATA STRUCTURE INTERPRETER
H
DATA PROPERTY INTERPRETER
K
GENERATOR
OBJECT PROGRAM
CHAR-ACTER CLASS TABLE
CHAR-ACTER PAIR MATRIX
RESERVED WORD LIST
NAME TABLE
TRAN-SITION DIAGRAMS
C
POINTER AND TREE TABLE
DATA DEFI-NITION TABLE
DATA DESCRIPTION PROCESSOR
D
DATA STRUCTURE RECORDER
E
DATA PROPERTY RECORDER
G
MAIN PROGRAM



Melvin Conway

Joel Erdwinn

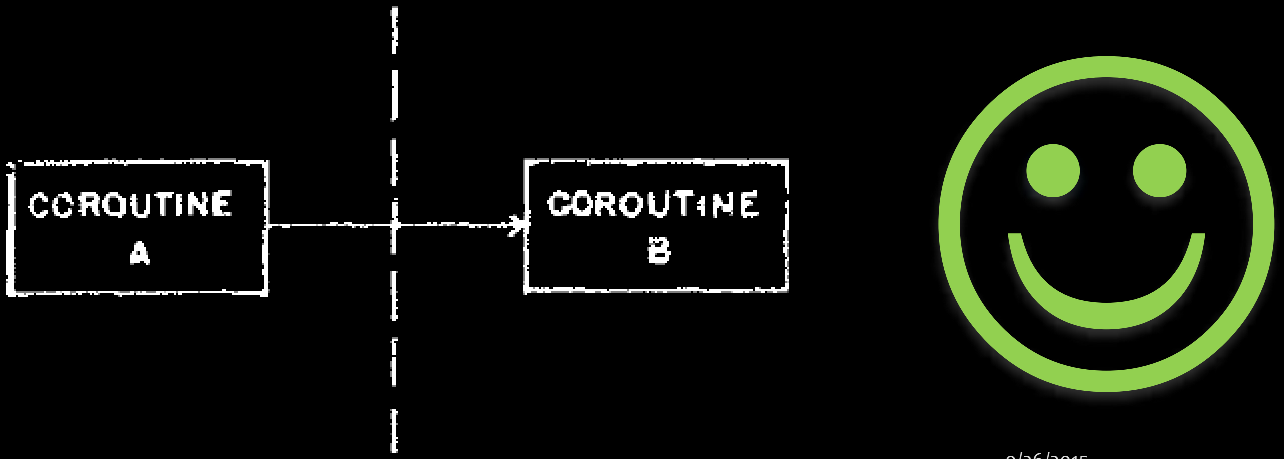
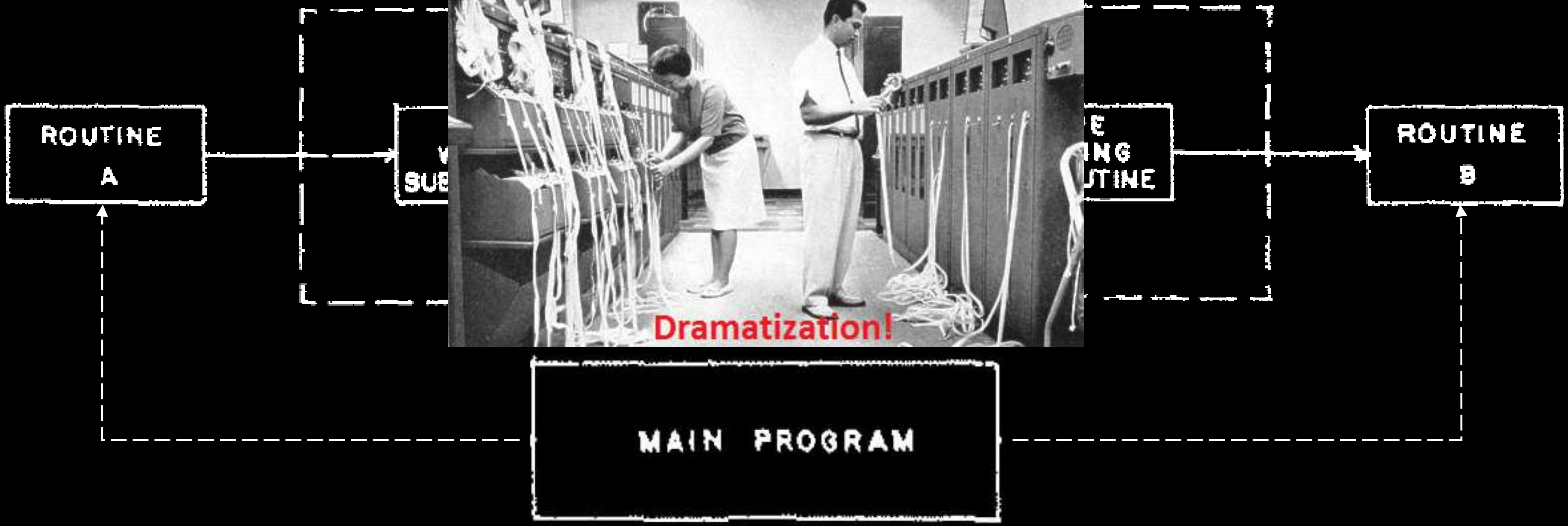
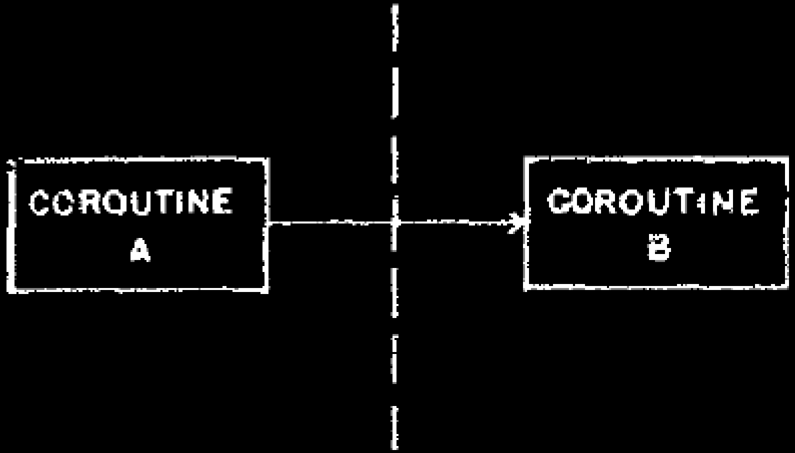


image credits: wikipedia commons, Communication of the ACM vol.6 No.7 July 1963

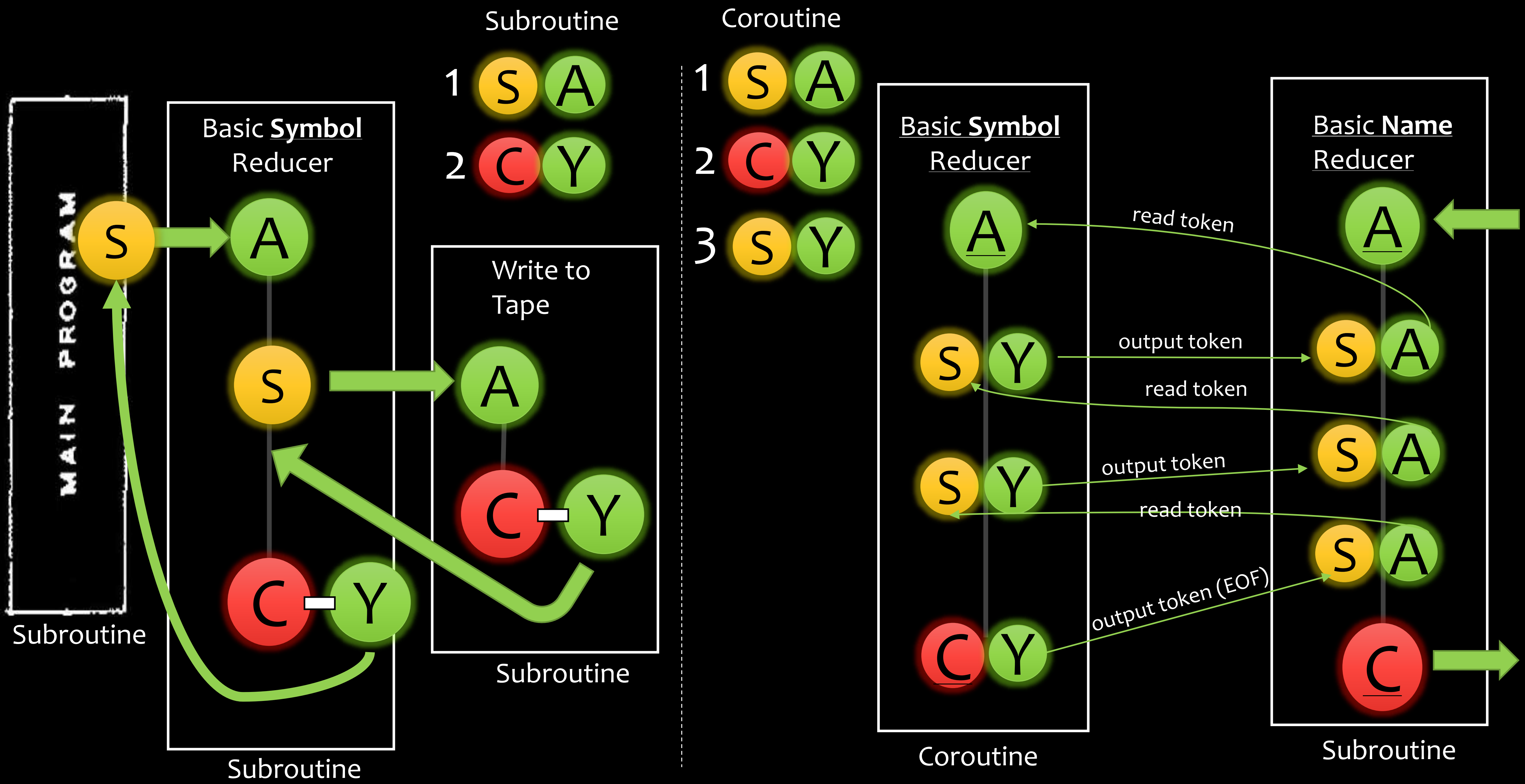




Melvin Conway

Joel Erdwinn

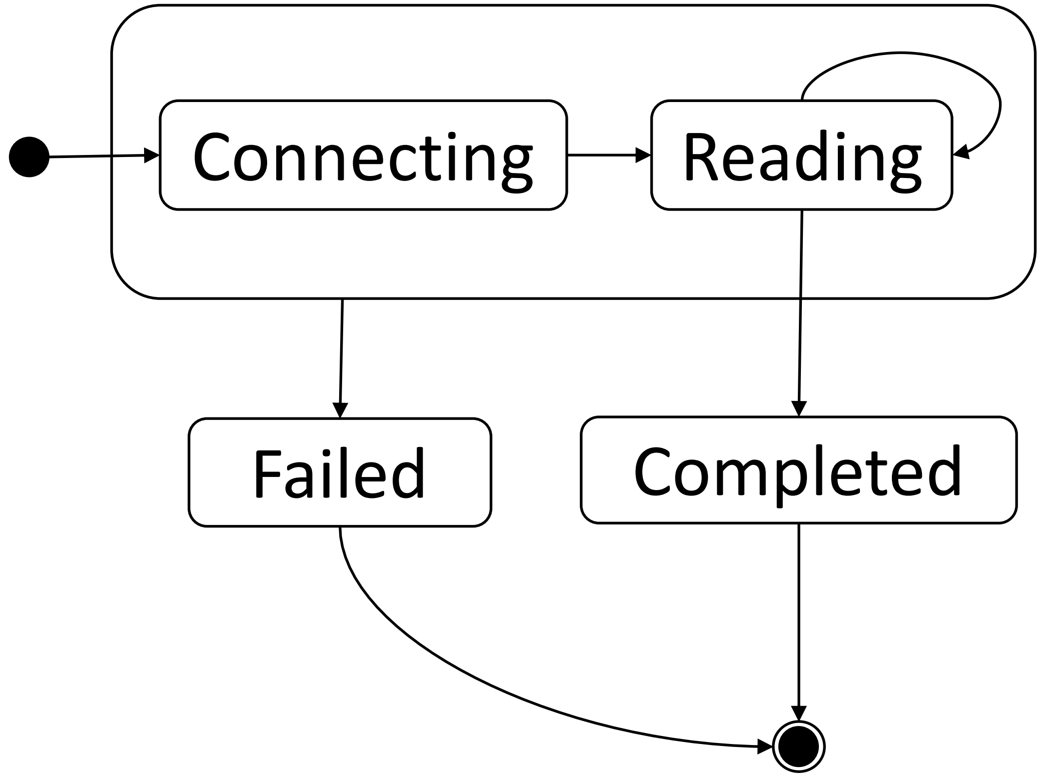
image credits: wikipedia commons, Communication of the ACM vol.6 No.7 July 1963


Subroutine
1 S A
2 C Y
Coroutine
1 S A
2 C Y
3 S Y
MAIN PROGRAM
S
Subroutine
Basic Symbol Reducer
A
S
C Y
Subroutine
Write to Tape
A
C Y
Subroutine
Basic Symbol Reducer
A
S Y
S Y
C Y
Coroutine
Basic Name Reducer
A
S A
S A
S A
C
Subroutine
read token
output token
read token
output token
read token
output token (EOF)

100 cards per minute!

2015

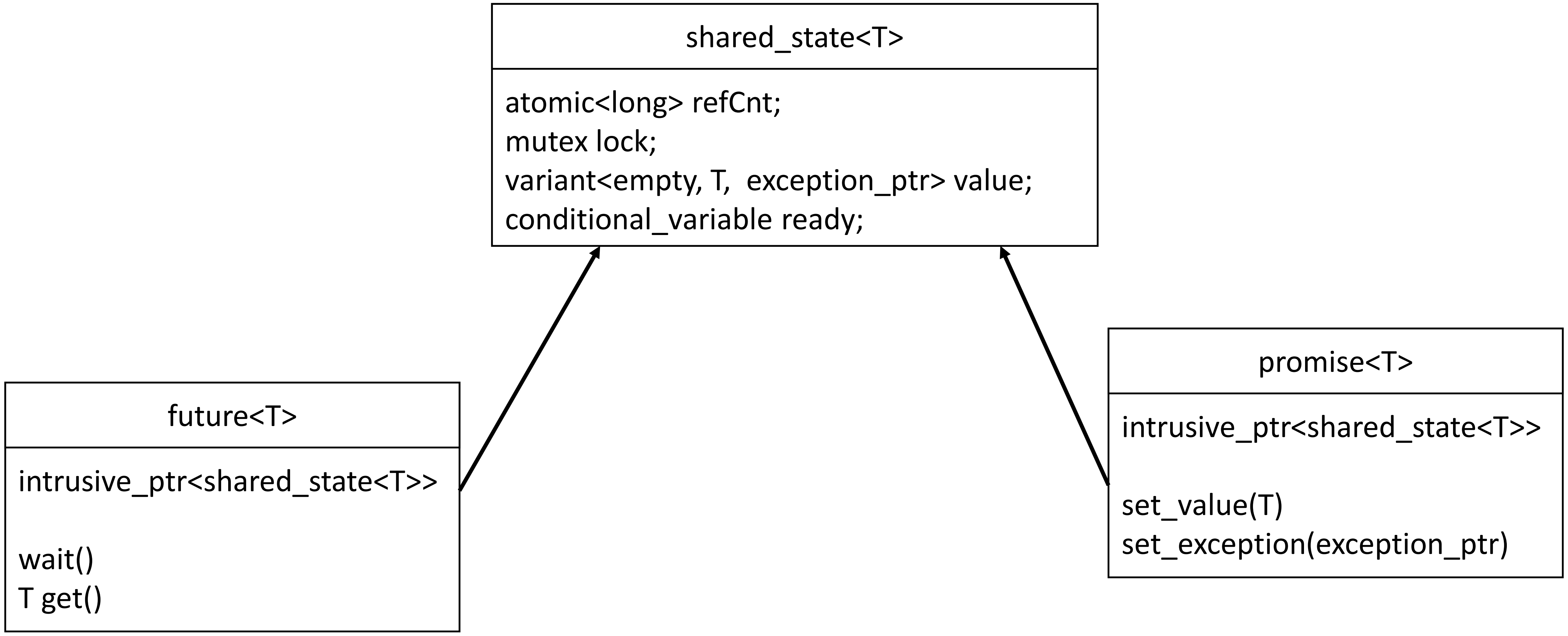
Async state machine



Trivial if synchronous

```
int tcp_reader(int total)
{
    char buf[4 * 1024];
    auto conn = Tcp::Connect("127.0.0.1", 1337);
    for (;;)
    {
        auto bytesRead = conn.Read(buf, sizeof(buf));
        total -= bytesRead;
        if (total <= 0 || bytesRead == 0) return total;
    }
}
```

std::future<T> and std::promise<T>


shared_state<T>
atomic<long> refCnt;
mutex lock;
variant<empty, T, exception_ptr> value;
conditional_variable ready;
future<T>
intrusive_ptr<shared_state<T>>
wait()
T get()
promise<T>
intrusive_ptr<shared_state<T>>
set_value(T)
set_exception(exception_ptr)

N4399 Working Draft, Technical Specification for C++ Extensions for Concurrency

.then

```
future<int> tcp_reader(int64_t total) {
    struct State {
        char buf[4 * 1024];
        int64_t total;
        Tcp::Connection conn;
        explicit State(int64_t total) : total(total) {}
    };
    auto state = make_shared<State>(total);
    return Tcp::Connect("127.0.0.1", 1337).then(
        [state](future<Tcp::Connection> conn) {
            state->conn = std::move(conn.get());
            return do_while([state]()->future<bool> {
                if (state->total <= 0) return make_ready_future(false);
                return state->conn.read(state->buf, sizeof(state->buf)).then(
                    [state](future<int> nBytesFut) {
                        auto nBytes = nBytesFut.get()
                        if (nBytes == 0) return make_ready_future(false);
                        state->total -= nBytes;
                        return make_ready_future(true);
                });
            });
    });
}
```

```
future<void> do_while(function<future<bool>()> body) {
    return body().then([=](future<bool> notDone) {
        return notDone.get() ? do_while(body) : make_ready_future(); });
}
```

Forgot something

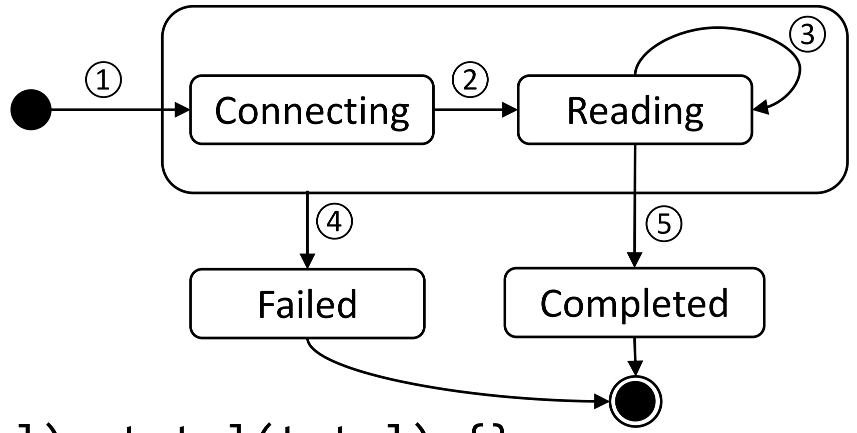
```
int tcp_reader(int total)
{
    char buf[4 * 1024];
    auto conn = Tcp::Connect("127.0.0.1", 1337);
    for (;;)
    {
        auto bytesRead = conn.Read(buf, sizeof(buf));
        total -= bytesRead;
        if (total <= 0 || bytesRead == 0) return total;
    }
}
```

```
future<int> tcp_reader(int64_t total) {
    struct State {
        char buf[4 * 1024];
        int64_t total;
        Tcp::Connection conn;
        explicit State(int64_t total) : total(total) {}
    };
    auto state = make_shared<State>(total);
    return Tcp::Connect("127.0.0.1", 1337).then(
        [state](future<Tcp::Connection> conn) {
            state->conn = std::move(conn.get());
            return do_while([state]()->future<bool> {
                if (state->total <= 0) return make_ready_future(false);
                return state->conn.read(state->buf, sizeof(state->buf)).then(
                    [state](future<int> nBytesFut) {
                        auto nBytes = nBytesFut.get()
                        if (nBytes == 0) return make_ready_future(false);
                        state->total -= nBytes;
                        return make_ready_future(true);
                }); // read
            }); // do_while
     }); // Tcp::Connect
}
```

```
future<int> tcp_reader(int64_t total) {
    struct State {
        char buf[4 * 1024];
        int64_t total;
        Tcp::Connection conn;
        explicit State(int64_t total) : total(total) {}
    };
    auto state = make_shared<State>(total);
    return Tcp::Connect("127.0.0.1", 1337).then(
        [state](future<Tcp::Connection> conn) {
             state->conn = std::move(conn.get());
             return do_while([state]()->future<bool> {
                 if (state->total <= 0) return make_ready_future(false);
                 return state->conn.read(state->buf, sizeof(state->buf)).then(
                     [state](future<int> nBytesFut) {
                         auto nBytes = nBytesFut.get()
                         if (nBytes == 0) return make_ready_future(false);
                         state->total -= nBytes;
                         return make_ready_future(true);
                 }); // read
             }); // do_while
     }).then([state](            ){return make_ready_future(state->total)});
}
```

Hand-crafted async state machine (1/3)

```
class tcp_reader
{
    char buf[64 * 1024];
    Tcp::Connection conn;
    promise<int> done;
    int total;

    explicit tcp_reader(int total): total(total) {}

  ② void OnConnect(error_code ec, Tcp::Connection newCon);
  ③ void OnRead(error_code ec, int bytesRead);
  ④ void OnError(error_code ec);
  ⑤ void OnComplete();

public:
  ① static future<int> start(int total);
};
```



```
int main() {
   cout << tcp_reader::start(1000 * 1000 * 1000).get(); }
```

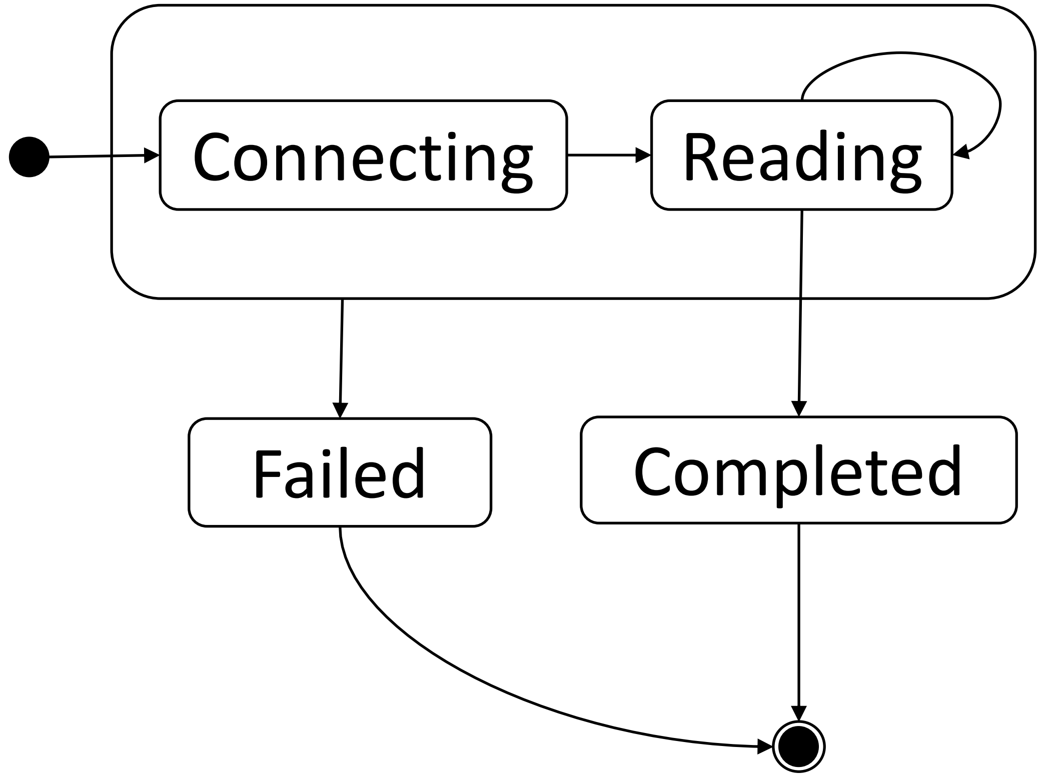
Hand-crafted async state machine (2/3)

```
future<int> tcp_reader::start(int total) {
   auto p = make_unique<tcp_reader>(total);
   auto result = p->done.get_future();
   Tcp::Connect("127.0.0.1", 1337,
      [raw = p.get()](auto ec, auto newConn) {
            raw->OnConnect(ec, std::move(newConn));
      });
   p.release();
   return result;
}

void tcp_reader::OnConnect(error_code ec,
                           Tcp::Connection newCon)
{
   if (ec) return OnError(ec);
   conn = std::move(newCon);
   conn.Read(buf, sizeof(buf),
      [this](error_code ec, int bytesRead)
         { OnRead(ec, bytesRead); });
}
```

Hand-crafted async state machine (3/3)

```cpp
void tcp_reader::OnRead(error_code ec, int bytesRead) {
   if (ec) return OnError(ec);
   total -= bytesRead;
   if (total <= 0 || bytesRead == 0) return OnComplete();
   conn.Read(buf, sizeof(buf),
      [this](error_code ec, int bytesRead) {
         OnRead(ec, bytesRead); });
}

void OnError(error_code ec) {
   auto cleanMe = unique_ptr<tcp_reader>(this);
   done.set_exception(make_exception_ptr(system_error(ec)));
}

void OnComplete() {
   auto cleanMe = unique_ptr<tcp_reader>(this);
   done.set_value(total);
}
```

Async state machine



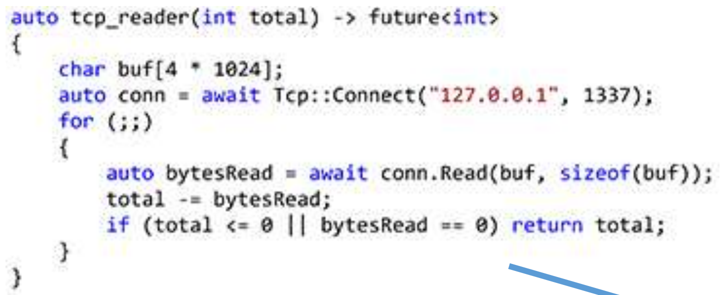
Trivial

```
auto tcp_reader(int total) -> int
{
    char buf[4 * 1024];
    auto conn =       Tcp::Connect("127.0.0.1", 1337);
    for (;;)
    {
        auto bytesRead =       conn.Read(buf, sizeof(buf));
        total -= bytesRead;
        if (total <= 0 || bytesRead == 0) return total;
    }
}
```

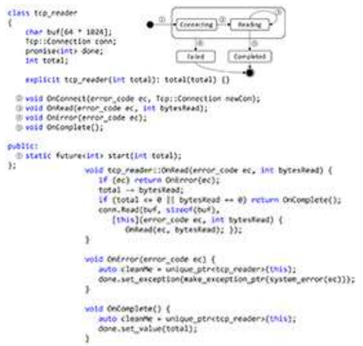
Trivial

```
auto tcp_reader(int total) -> future<int>
{
    char buf[4 * 1024];
    auto conn = await Tcp::Connect("127.0.0.1", 1337);
    for (;;)
    {
        auto bytesRead = await conn.Read(buf, sizeof(buf));
        total -= bytesRead;
        if (total <= 0 || bytesRead == 0) return total;
    }
}
```

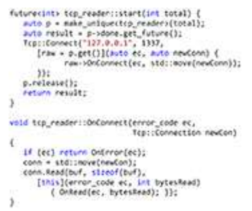
What about perf?



```
auto tcp_reader(int total) -> future<int>
{
    char buf[4 * 1024];
    auto conn = await Tcp::Connect("127.0.0.1", 1337);
    for (;;)
    {
        auto bytesRead = await conn.Read(buf, sizeof(buf));
        total -= bytesRead;
        if (total <= 0 || bytesRead == 0) return total;
    }
}
```



```
class tcp_reader
{
    char buf[64 * 1024];
    Tcp::Connection conn;
    promise<int> done;
    int total;

    explicit tcp_reader(int total): total(total) {}

    void OnConnect(error_code ec, Tcp::Connection newCon);
    void OnRead(error_code ec, int bytesRead);
    void OnError(error_code ec);
    void OnComplete();

public:
    static future<int> start(int total);
};
            void tcp_reader::OnRead(error_code ec, int bytesRead) {
               if (ec) return OnError(ec);
               total -= bytesRead;
               if (total <= 0 || bytesRead == 0) return OnComplete();
               conn.Read(buf, sizeof(buf),
                  [this](error_code ec, int bytesRead) {
                     OnRead(ec, bytesRead); });
            }

            void OnError(error_code ec) {
               auto cleanMe = unique_ptr<tcp_reader>(this);
               done.set_exception(make_exception_ptr(system_error(ec)));
            }

            void OnComplete() {
               auto cleanMe = unique_ptr<tcp_reader>(this);
               done.set_value(total);
            }
```



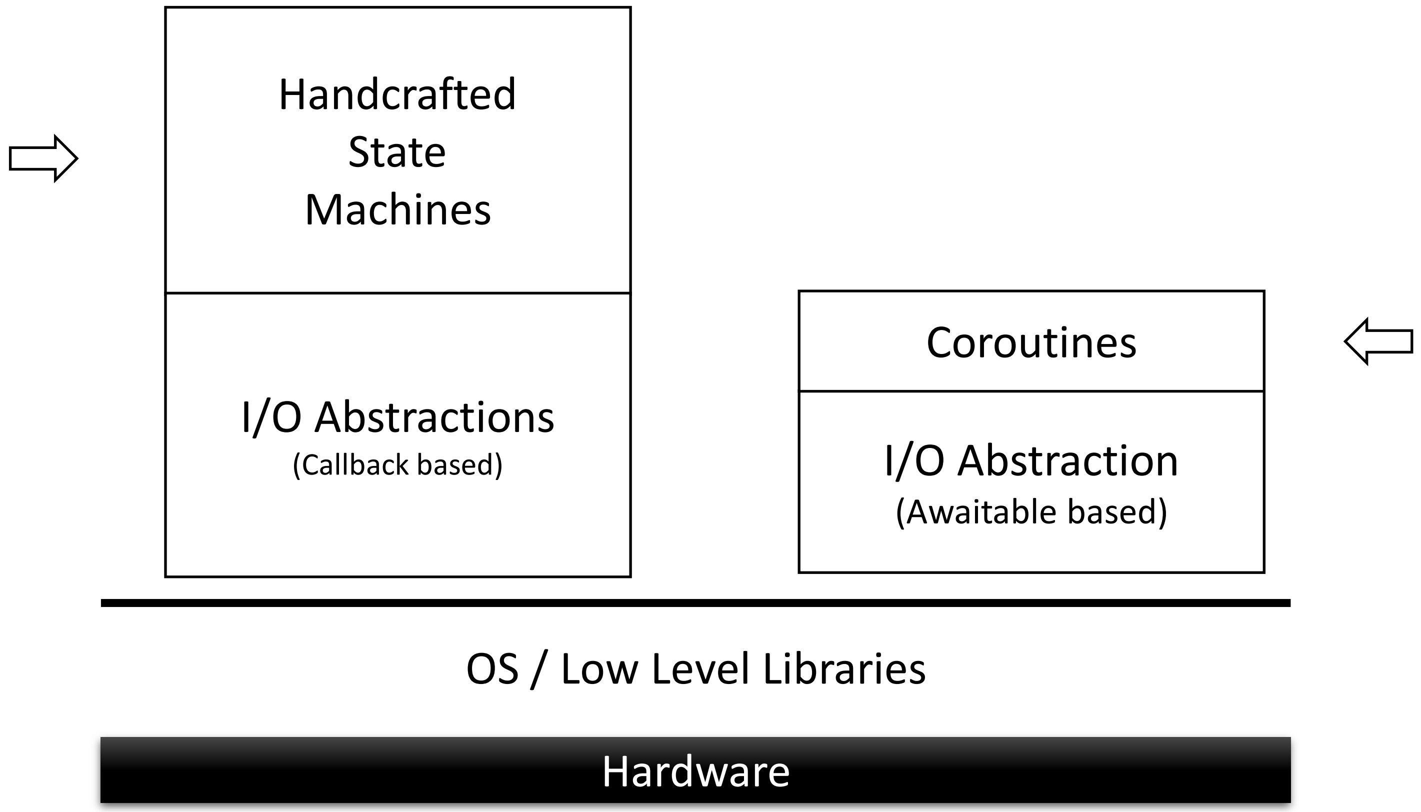
```
future<int> tcp_reader::start(int total) {
   auto p = make_unique<tcp_reader>(total);
   auto result = p->done.get_future();
   Tcp::Connect("127.0.0.1", 1337,
      [raw = p.get()](auto ec, auto newConn) {
            raw->OnConnect(ec, std::move(newConn));
      });
   p.release();
   return result;
}

void tcp_reader::OnConnect(error_code ec,
                           Tcp::Connection newCon)
{
   if (ec) return OnError(ec);
   conn = std::move(newCon);
   conn.Read(buf, sizeof(buf),
      [this](error_code ec, int bytesRead)
         { OnRead(ec, bytesRead); });
}
```

```
int main() {
    printf("Hello, world\n");
}
```

	Coroutines	Hand-Crafted	Hello
MB/s	495 (1.3x)	380	0
Binary size (Kbytes)	25 (0.85x)	30	9

Visual C++ 2015 RTM. Measured on Lenovo W540 laptop. Transmitting & Receiving 1GB over loopback IP addr

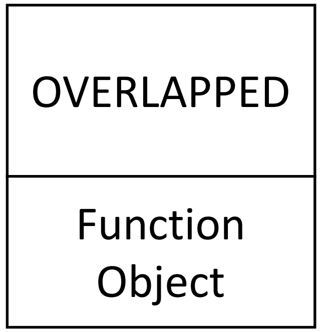
Coroutines are closer to the metal



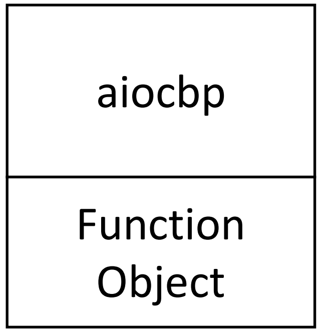
How to map high level call to OS API?

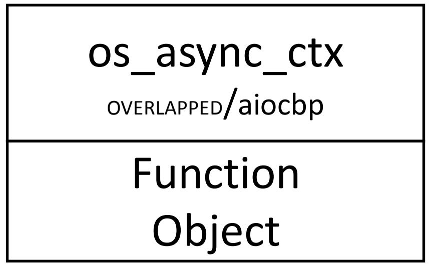
```
conn.Read(buf, sizeof(buf),
      [this](error_code ec, int bytesRead)
         { OnRead(ec, bytesRead); });

template <class Cb>
void Read(void* buf, size_t bytes, Cb && cb);
```

Windows: WSARecv(fd, ..., OVERLAPPED*)



Posix aio: aio_read(fd, ..., aiocbp*)





```
struct OverlappedBase : os_async_context {
    virtual void Invoke(std::error_code, int bytes) = 0;
    virtual ~OverlappedBase() {}

    static void io_complete_callback(CompletionPacket& p) {
        auto me = unique_ptr<OverlappedBase>(static_cast<OverlappedBase*>(p.overlapped));
        me->Invoke(p.error, p.byteTransferred);
    }
};
```

After open associate a socket handle with a threadpool and a callback

```
   ThreadPool::AssociateHandle(sock.native_handle(), &OverlappedBase::io_complete_callback);

template <typename Fn> struct CompletionWithCount : OverlappedBase, private Fn
{
    CompletionWithCount(Fn fn) : Fn(std::move(fn)) {}

    void Invoke(std::error_code ec, int count) override { Fn::operator()(ec, count); }
};

template <typename Fn> unique_ptr<OverlappedBase> make_handler_with_count(Fn && fn) {
   return std::make_unique<CompletionWithCount<std::decay_t<Fn>>(std::forward<Fn>(fn));
}
```

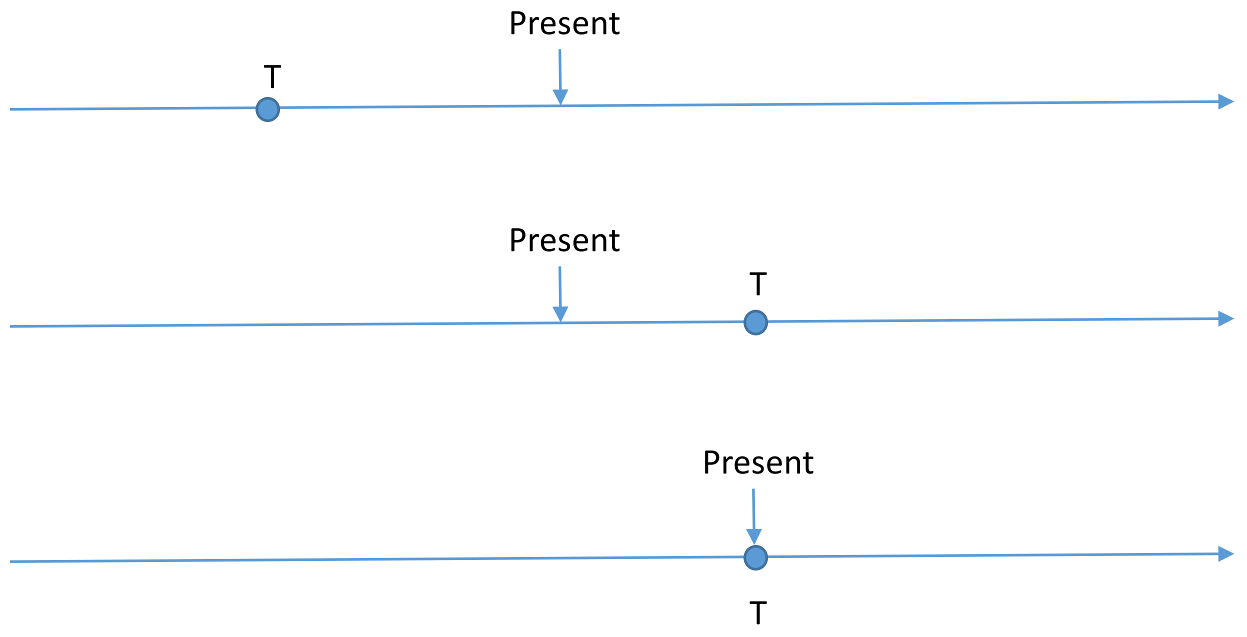
```
conn.Read(buf, sizeof(buf),
      [this](error_code ec, int bytesRead)
         { OnRead(ec, bytesRead); });
```

```
template <typename F>
void Read(void* buf, int len, F && cb) {
    return Read(buf, len, make_handler_with_count(std::forward<F>(cb)));
}

void Read(void* buf, int len, std::unique_ptr<detail::OverlappedBase> o)
{
    auto error = sock.Receive(buf, len, o.get());
    if (error) {
        if (error.value() != kIoPending) {
            o->Invoke(error, 0);
            return;
        }
    }
    o.release();
}
```

```
await conn.Read(buf, sizeof(buf));
```

?

Awaitable – Concept of the Future<T>



.await_ready()
`F<T> → bool`

.await_suspend(cb)
`F<T> x Fn → void`

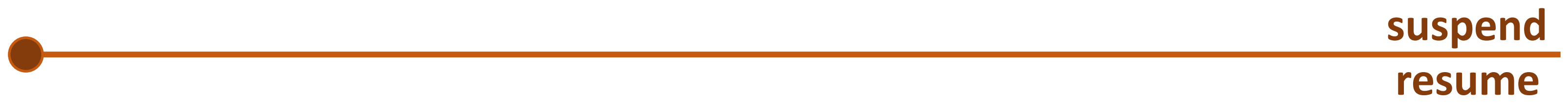
.await_resume()
`F<T> → T`

```
await expr-of-awaitable-type
```

await <expr>

Expands into an expression equivalent of

```
{
  auto && tmp = <expr>;
  if (!await_ready(tmp)) {
     await_suspend(tmp, <coroutine-handle>);
```



```
  }
  return  await_resume(tmp);
}
```

REMEMBER THIS?

Overlapped Base from before

```
struct OverlappedBase : os_async_context
{
    virtual void Invoke(std::error_code, int bytes) = 0;
    virtual ~OverlappedBase() {}


    static void io_complete_callback(CompletionPacket& p) {
        auto me = static_cast<OverlappedBase*>(p.overlapped);
        auto cleanMe = unique_ptr<OverlappedBase>(me);

        me->Invoke(p.error, p.byteTransferred);
    }
};
```

Overlapped Base for awaitable

```
struct AwaiterBase : os_async_context
{
    coroutine_handle<> resume;
    std::error_code err;
    int bytes;

    static void io_complete_callback(CompletionPacket& p) {
        auto me = static_cast<AwaiterBase*>(p.overlapped);
        me->err = p.error;
        me->bytes = p.byteTransferred;
        me->resume();
    }
};
```

```
sizeof(void*)
no dtor
```

```
mov rcx, [rcx]
jmp [rcx]
```

```
await conn.Read(buf, sizeof(buf));
```

?

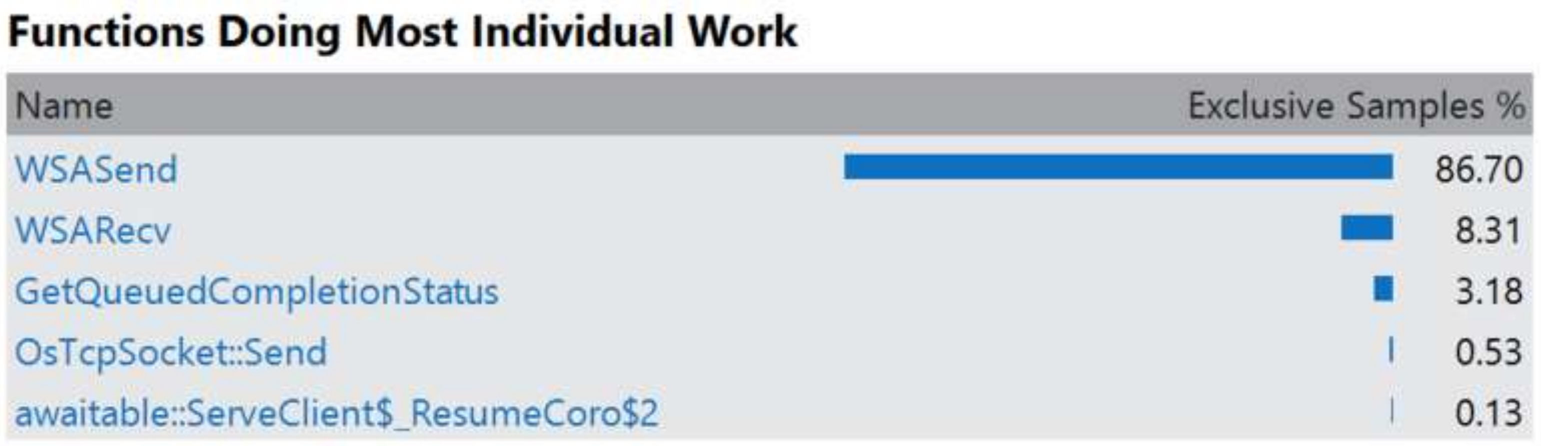
```
auto Connection::Read(void* buf, int len) {
    struct awaiter: AwaiterBase {
        Connection* me;
        void* buf;
        awaiter(Connection* me, void* buf, int len) : me(me), buf(buf) { bytes = len; }

        bool await_ready() { return false; }

        void await_suspend(coroutine_handle<> h) {
            this->resume = h;
            auto error = me->sock.Receive(buf, bytes, this);
            if (error.value() != kIoPending)
                throw system_error(err);
        }

        int await_resume() {
            if (this->err) throw system_error(err);
            return bytes;
        }
    };
    return awaiter{ this, buf, len };
}
```

```
struct AwaiterBase : os_async_context {
    coroutine_handle<> resume;
    std::error_code err;
    int bytes;

    static void io_complete_callback(CompletionPacket& p){
        auto me = static_cast<AwaiterBase*>(p.overlapped);
        me->err = p.error;
        me->bytes = p.byteTransferred;
        me->resume();
    }
};
```

Trivial

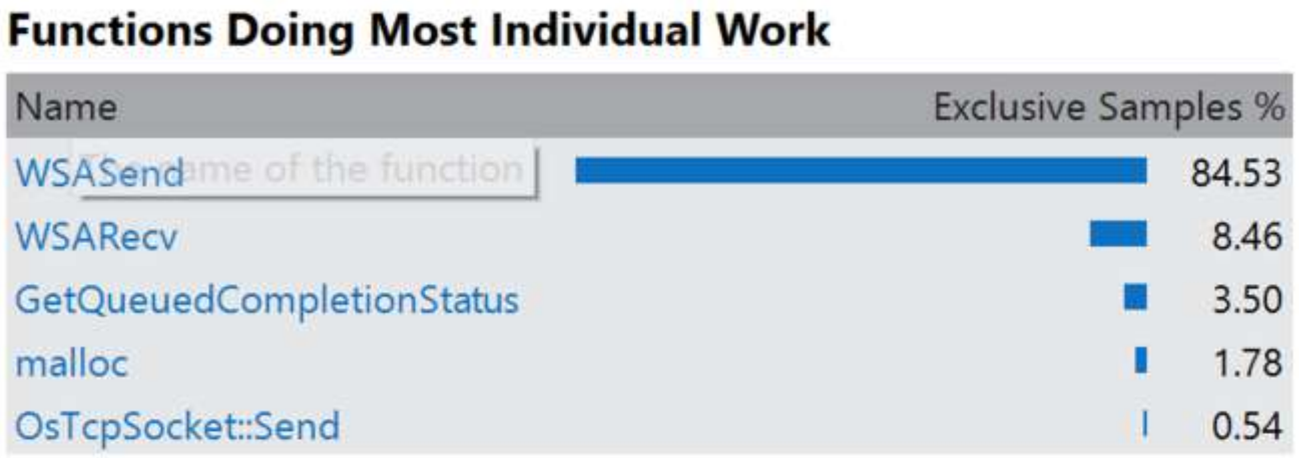
```
auto tcp_reader(int total) -> future<int>
{
    char buf[4 * 1024];
    auto conn = await Tcp::Connect("127.0.0.1", 1337);
    for (;;)
    {
        auto bytesRead = await conn.Read(buf, sizeof(buf));
        total -= bytesRead;
        if (total <= 0 || bytesRead == 0) return total;
    }
}
```

Can we make it better?



Functions Doing Most Individual Work

Name	Exclusive Samples %
WSASend	86.70
WSARecv	8.31
GetQueuedCompletionStatus	3.18
OsTcpSocket::Send	0.53
awaitable::ServeClient$_ResumeCoro$2	0.13



Functions Doing Most Individual Work

Name	Exclusive Samples %
WSASend	84.53
WSARecv	8.46
GetQueuedCompletionStatus	3.50
malloc	1.78
OsTcpSocket::Send	0.54

50% I/O completes synchronously
50% I/O with I/O pending error

```
SetFileCompletionNotificationModes(h,
        FILE_SKIP_COMPLETION_PORT_ON_SUCCESS);
```

```
SetFileCompletionNotificationModes(h,
          FILE_SKIP_COMPLETION_PORT_ON_SUCCESS);
```

Take advantage of synchronous completions

```
void Read(void* buf, int len, std::unique_ptr<detail::OverlappedBase> o)
{
    auto error = sock.Receive(buf, len, o.get());
    if (error) {
        if (error.value() != kIoPending) {
            o->Invoke(error, 0);
            return;
        }
    }
    o.release();
}
```

```
SetFileCompletionNotificationModes(h,
          FILE_SKIP_COMPLETION_PORT_ON_SUCCESS);
```

Take advantage of synchronous completions

```
void Read(void* buf, int len, std::unique_ptr<detail::OverlappedBase> o)
{
    auto error = sock.Receive(buf, len, o.get());

        if (error.value() != kIoPending) {
            o->Invoke(error, len);
            return;
        }

    o.release();
}
```

```
SetFileCompletionNotificationModes(h,
          FILE_SKIP_COMPLETION_PORT_ON_SUCCESS);
```

Take advantage of synchronous completions

```
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >::Invoke(std::error_code ec, int count) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >::Invoke(std::error_code ec, int count) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >::Invoke(std::error_code ec, int count) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >::Invoke(std::error_code ec, int count) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >::Invoke(std::error_code ec, int count) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >::Invoke(std::error_code ec, int count) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >::Invoke(std::error_code ec, i t count) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2 450c1> >::Invoke(std:: rror_code ec   t count) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c     :Invoke(std    r_code     t count) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >                           Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >                             31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c245                                 e 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >                             31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450                            ) Line 31
SuperLean.exe!improved::tcp_reader::OnRead(std::error_code ec, int bytesRead) Line 254
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >::Invo           nt       Line 31
SuperLean.exe!improved::detail::CompletionWithSizeT<<lambda_ee38b7a750c7f550b4ee1dd60c2450c1> >::In  e(std:   _code   int count  ine 31
SuperLean.exe!improved::detail::io_complete_callback(CompletionPacket & p) Line 22
SuperLean.exe!CompletionQueue::ThreadProc(void * lpParameter) Line 112       C++
```

Stack Overflow

Need to implement it on the use side

```
void tcp_reader::OnRead(std::error_code ec, int bytesRead) {

       if (ec) return OnError(ec);
       total -= (int)bytesRead;
       if (total <= 0 || bytesRead == 0) return OnComplete();
       bytesRead = sizeof(buf);

       conn.Read(buf, bytesRead,
        [this](std::error_code ec, int bytesRead) {
                                 OnRead(ec, bytesRead); }) ;
}
```

Now handling synchronous completion

```
void tcp_reader::OnRead(std::error_code ec, int bytesRead) {
    do {
       if (ec) return OnError(ec);
       total -= (int)bytesRead;
       if (total <= 0 || bytesRead == 0) return OnComplete();
       bytesRead = sizeof(buf);
    } while (
       conn.Read(buf, bytesRead,
        [this](std::error_code ec, int bytesRead) {
                                                  OnRead(ec, bytesRead); }));
}
```

Let’s measure the improvement (handwritten)

	MB/s		Executable size	
	Handcrafted	Coroutine	Handcrafted	Coroutine
Original	380	495	30	25
Synchr Completion. Opt	485		30	

```
auto Connection::Read(void* buf, int len) {
    struct awaiter: AwaiterBase {
        Connection* me;
        void* buf;
        awaiter(Connection* me, void* buf, int len) : me(me), buf(buf) { bytes = len; }

        bool await_ready() { return false; }

        void await_suspend(coroutine_handle<> h) {
            this->resume = h;
            auto error = me->sock.Receive(buf, bytes, this);
            if (error.value() == kIoPending) return;
            if (error) throw system_error(err);
            return;
        }

        int await_resume() {
           if (this->err) throw system_error(err);
           return bytes;
        }
    };
    return awaiter{ this, buf, len };
}
```

```
SetFileCompletionNotificationModes(h,
        FILE_SKIP_COMPLETION_PORT_ON_SUCCESS);
```

```
struct AwaiterBase : os_async_context {
    coroutine_handle<> resume;
    std::error_code err;
    int bytes;

    static void io_complete_callback(CompletionPacket& p){
        auto me = static_cast<AwaiterBase*>(p.overlapped);
        me->err = p.error;
        me->bytes = p.byteTransferred;
        me->resume();
    }
};
```

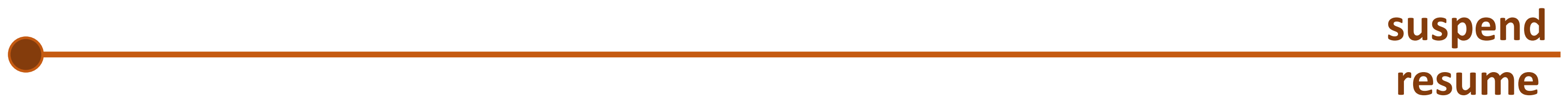
```cpp
auto Connection::Read(void* buf, int len) {
    struct awaiter: AwaiterBase {
        Connection* me;
        void* buf;
        awaiter(Connection* me, void* buf, int len) : me(me), buf(buf) { bytes = len; }

        bool await_ready() { return false; }

        bool await_suspend(coroutine_handle<> h) {
            this->resume = h;
            auto error = me->sock.Receive(buf, bytes, this);
            if (error.value() == kIoPending) return true;
            if (error) throw system_error(err);
            return false;
        }

        int await_resume() {
           if (this->err) throw system_error(err);
           return bytes;
        }
    };
    return awaiter{ this, buf, len };
}
```

```cpp
struct AwaiterBase : os_async_context {
    coroutine_handle<> resume;
    std::error_code err;
    int bytes;

    static void io_complete_callback(CompletionPacket& p){
        auto me = static_cast<AwaiterBase*>(p.overlapped);
        me->err = p.error;
        me->bytes = p.byteTransferred;
        me->resume();
    }
};
```

await <expr>

Expands into an expression equivalent of

```
{
  auto && tmp = <expr>;
  if (!await_ready(tmp)) {
     await_suspend(tmp, <coroutine-handle>);
```



```
  }
  return  await_resume(tmp);
}
```

await <expr>

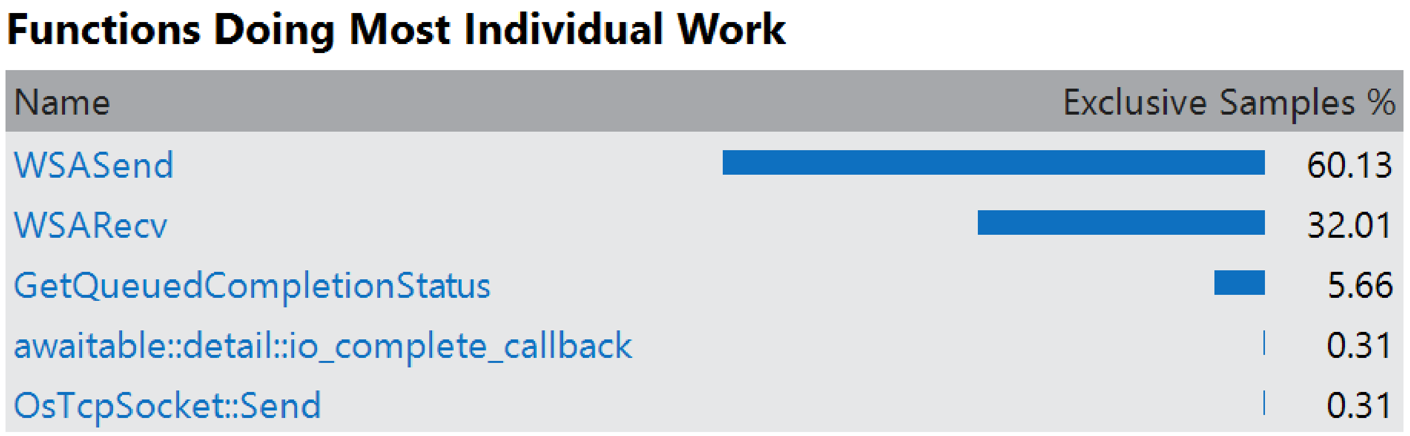
Expands into an expression equivalent of

```
{
  auto && tmp = <expr>;
  if (!await_ready(tmp) &&
     await_suspend(tmp, <coroutine-handle>) {
     ●──────────────────────────────────────────── suspend
                                                   resume
  }
  return  await_resume(tmp);
}
```

Let’s measure the improvement (coroutine)

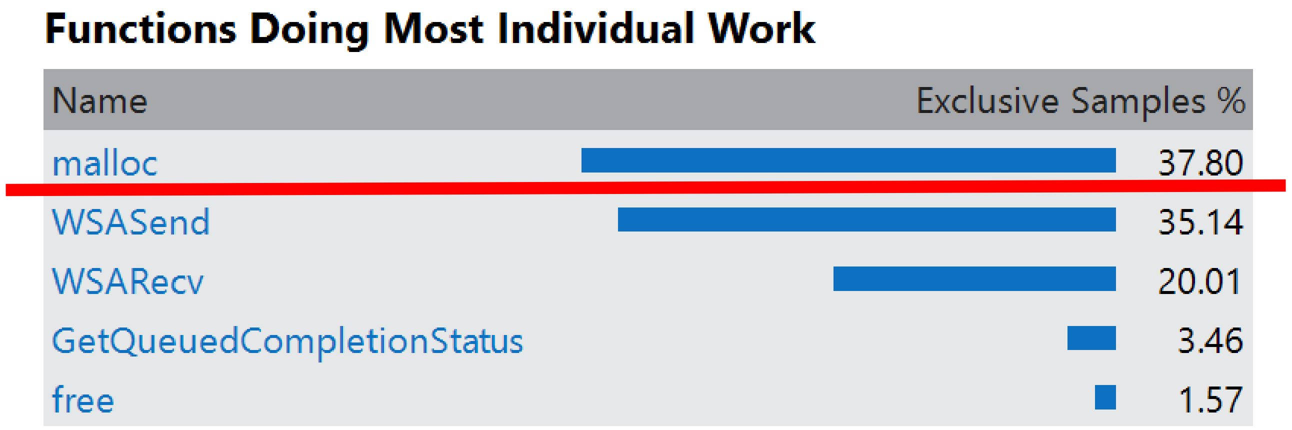
	MB/s		Executable size	
	Handcrafted	Coroutine	Handcrafted	Coroutine
Original	380	495	30	25
Synchr Completion. Opt	485	**1028**	30	25

Can we make it better?



Functions Doing Most Individual Work

Name	Exclusive Samples %
WSASend	60.13
WSARecv	32.01
GetQueuedCompletionStatus	5.66
awaitable::detail::io_complete_callback	0.31
OsTcpSocket::Send	0.31



Functions Doing Most Individual Work

Name	Exclusive Samples %
malloc	37.80
WSASend	35.14
WSARecv	20.01
GetQueuedCompletionStatus	3.46
free	1.57

Getting rid of the allocations

```
class tcp_reader {
   std::unique_ptr<detail::OverlappedBase> wo;
   …

   tcp_reader(int64_t total) : total(total) {
      wo = detail::make_handler_with_count(
         [this](auto ec, int nBytes) {OnRead(ec, nBytes); });
      …
   }

   void OnRead(std::error_code ec, int bytesRead) {
      if (ec) return OnError(ec);
      do {
         total -= (int)bytesRead;
         if (total <= 0 || bytesRead == 0) return OnComplete();
         bytesRead = sizeof(buf);
      } while (conn.Read(buf, bytesRead, wo.get()));
   }
```

Let’s measure the improvement (handcrafted)

	MB/s		Executable size	
	Handcrafted	Coroutine	Handcrafted	Coroutine
Original	380	495	30	25
Synchr Completion. Opt	485	1028	30	25
Prealloc handler	690	1028	28	25

Coroutines are popular!

DART 1.9

```
Future<int> getPage(t) async {
  var c = new http.Client();
  try {
    var r = await c.get('http://url/search?q=$t');
    print(r);
    return r.length(
  } finally {
    await c.close();
  }
}
```

Python: PEP 0492

```
async def abinary(n):
    if n <= 0:
        return 1
    l = await abinary(n - 1)
    r = await abinary(n - 1)
        + 1 + r
```

C#

```
async Task<string> WaitAsynchronouslyAsync()
{
    await  Task.Delay(10000);
    return "Finished";
}
```

CK (programming language)

```
  nc function gen1(): Awaitable<int> {
  $x =  await Batcher::fetch(1);
  $y =  await Batcher::fetch(2);
  return $x + $y;
}
```

C++17

```
future<string> WaitAsynchronouslyAsync()
{
    await  sleep_for(10ms);
    return "Finished“s;
}
```

Generalized Function

Coroutine

Designer

```
namespace std {
  template <typename T, typename… anything>
  struct coroutine_traits<boost::unique_future<T>, anything…> {
    struct promise_type {
      boost::promise<T> promise;
      auto get_return_object() { return promise.get_future(); }

      template <class U> void return_value(U && value) {
         promise.set_value(std::forward<U>(value));
      }

      void set_exception(std::exception_ptr e) {
        promise.set_exception(std::move(e));
      }
      bool initial_suspend() { return false; }
      bool final_suspend() { return false; }
    };
  };
}
```

User

```
auto tcp_reader(int total) -> future<int>
{
    char buf[4 * 1024];
    auto conn = await Tcp::Connect("127.0.0.1", 1337);
    for (;;)
    {
        auto bytesRead = await conn.Read(buf, sizeof(buf));
        total -= bytesRead;
        if (total <= 0 || bytesRead == 0) return total;
    }
}
```

Compiler does not care

POF

Monadic*

await - suspend

Task

await

Generator

yield

Async Generator

await + yield

image credits: Три богатыря и змей горыныч

Design Principles

- **Scalable** (to **b**illions of concurrent coroutines)
- **Efficient** (resume and suspend operations comparable in cost to a function call overhead)
- Seamless interaction with existing facilities **with no overhead**
- **Open ended** coroutine machinery allowing library designers to develop coroutine libraries exposing various high-level semantics, such as generators, goroutines, tasks and more.
- **Usable** in environments where **exceptions** are forbidden or **not available**

Coroutines – a negative overhead abstraction

- Proposal is working through C++ standardization committee (C++17?)
- **<u>Experimental</u>** implementation in VS 2015 RTM
- Clang implementation is in progress
- more details:
  - CppCon 2014 presentation on coroutines http://github.com/cppcon
  - http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2015/n4499.pdf
  - Pre-Kona mailing (P0054, P0055, P0056)

Thank you!

Kavya Kotacherry, Daveed Vandevoorde, Richard Smith, Jens Maurer, Lewis Baker, Kirk Shoop, Hartmut Kaiser, Kenny Kerr, Artur Laksberg, Jim Radigan, Chandler Carruth, Gabriel Dos Reis, Deon Brewis, Jonathan Caves, James McNellis, Stephan T. Lavavej, Herb Sutter, Pablo Halpern, Robert Schumacher, Viktor Tong, Geoffrey Romer, Michael Wong, Niklas Gustafsson, Nick Maliwacki, Vladimir Petter, Shahms King, Slava Kuznetsov, Tongari J, Lawrence Crowl, Valentin Isac
and many more who contributed

Questions?