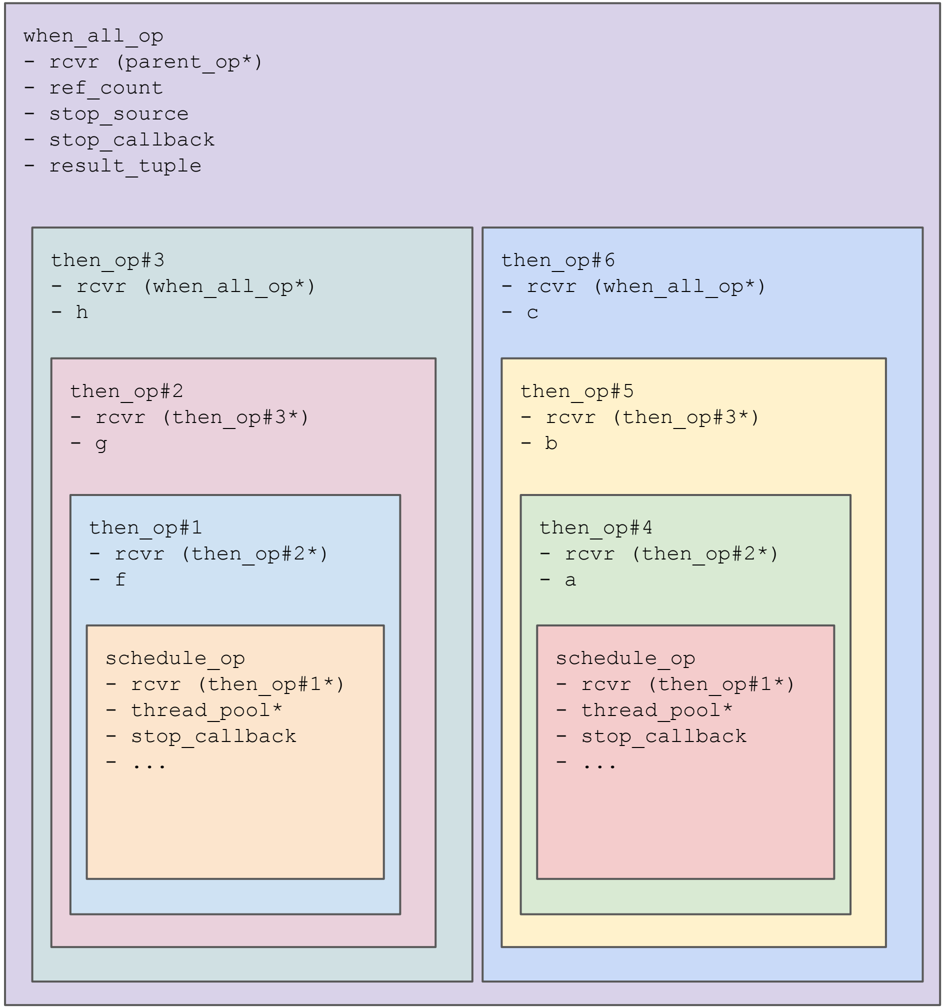
Reducing operation-state sizes for sub-object child operations

P3425R0

```
when_all(
  then(        // then_op#3
    then(      // then_op#2
      then(    // then_op#1
        schedule(thread_pool),
        f),
      g),
    h),
  then(        // then_op#6
    then(      // then_op#5
      then(    // then_op#4
        schedule(thread_pool),
        a),
      b),
    c))
```

```
when_all(
  then(        // then_op#3
    then(      // then_op#2
      then(    // then_op#1
        schedule(thread_pool),
        f),
      g),
    h),
  then(        // then_op#6
    then(      // then_op#5
      then(    // then_op#4
        schedule(thread_pool),
        a),
      b),
    c))
```



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

then_op#3
- rcvr (when_all_op*)
- h

then_op#2
- rcvr (then_op#3*)
- g

then_op#1
- rcvr (then_op#2*)
- f

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

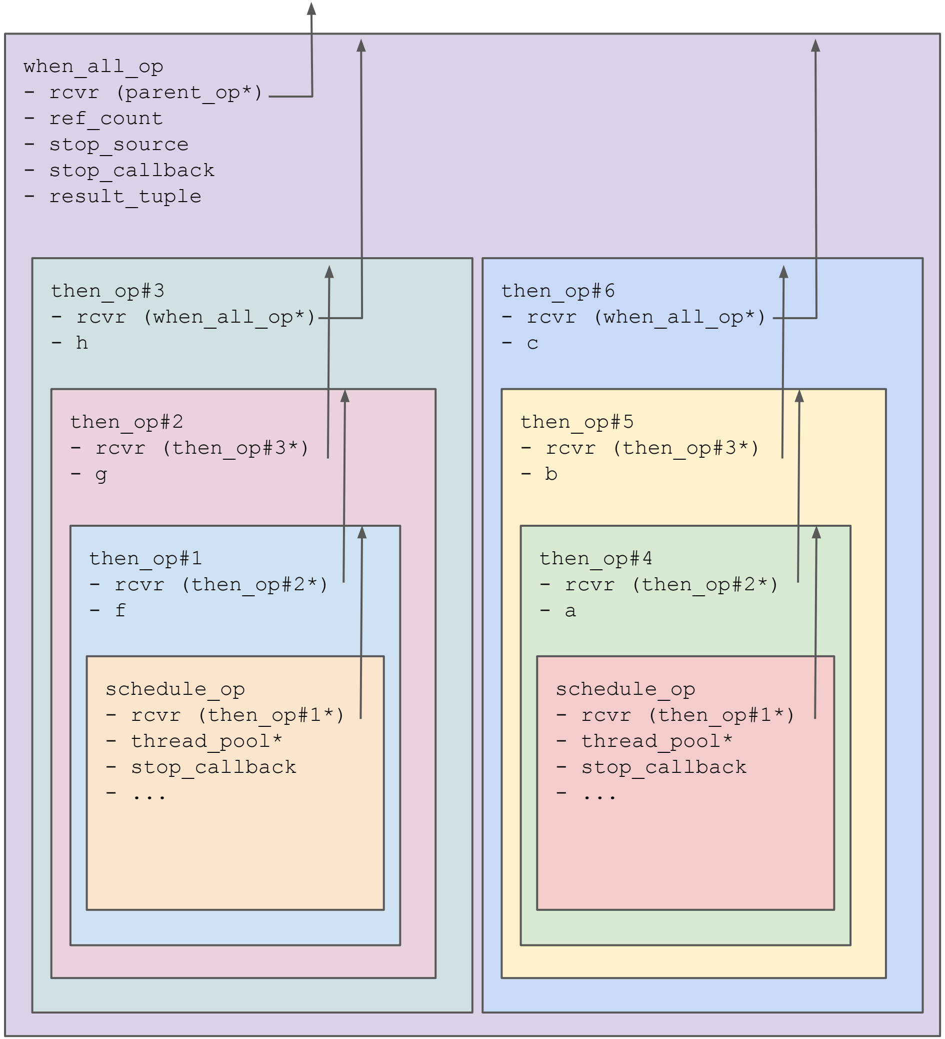
then_op#6
- rcvr (when_all_op*)
- c

then_op#5
- rcvr (then_op#3*)
- b

then_op#4
- rcvr (then_op#2*)
- a

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

```
when all(
  then(          // then_op#3
    then(        // then_op#2
      then(      // then op#1
        schedule(thread_pool),
        f),
      g),
    h),
  then(          // then_op#6
    then(        // then_op#5
      then(      // then op#4
        schedule(thread_pool),
        a),
      b),
    c))
```



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

then_op#3
- rcvr (when_all_op*)
- h

then_op#2
- rcvr (then_op#3*)
- g

then_op#1
- rcvr (then_op#2*)
- f

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

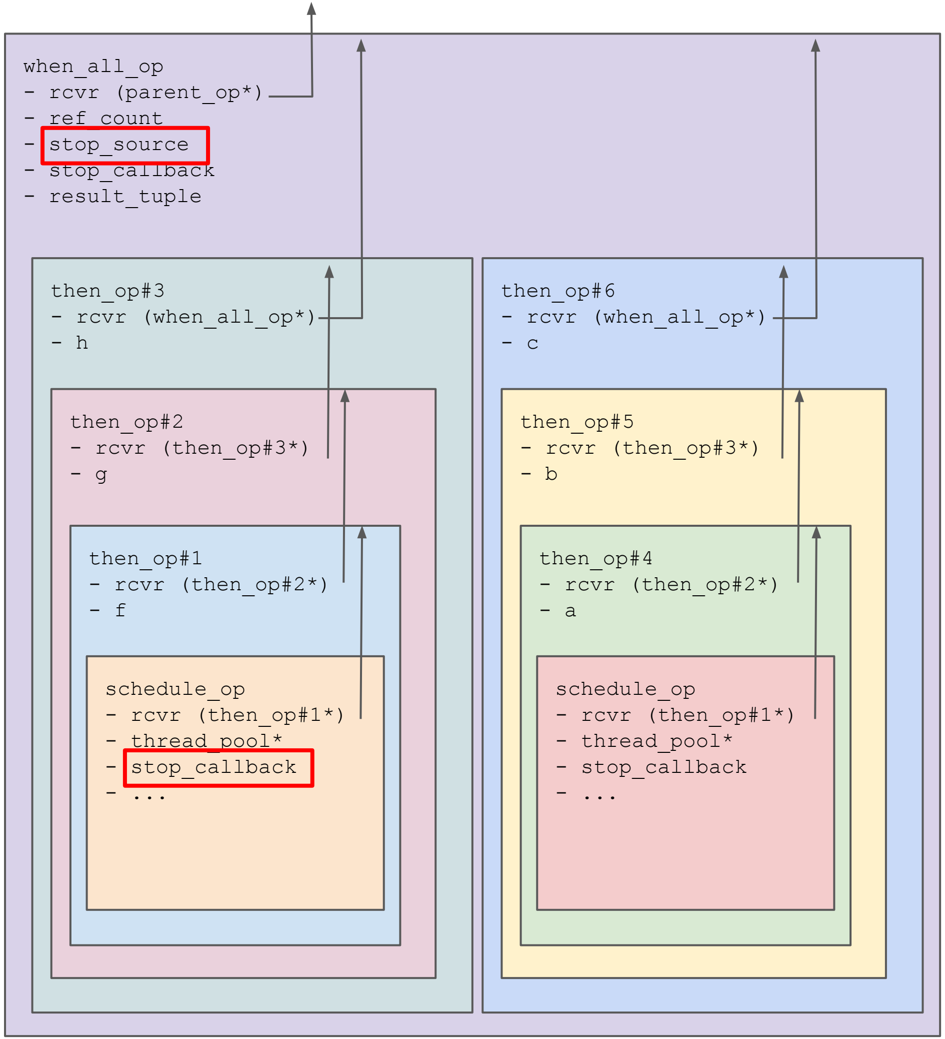
then_op#6
- rcvr (when_all_op*)
- c

then_op#5
- rcvr (then_op#3*)
- b

then_op#4
- rcvr (then_op#2*)
- a

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

```
void schedule_op::start() {
  auto st = get_stop_token(get_env(rcvr));
  stop_callback.emplace(st, on_stop{this});

  // ...
}
```



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

then_op#3
- rcvr (when_all_op*)
- h

then_op#2
- rcvr (then_op#3*)
- g

then_op#1
- rcvr (then_op#2*)
- f

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

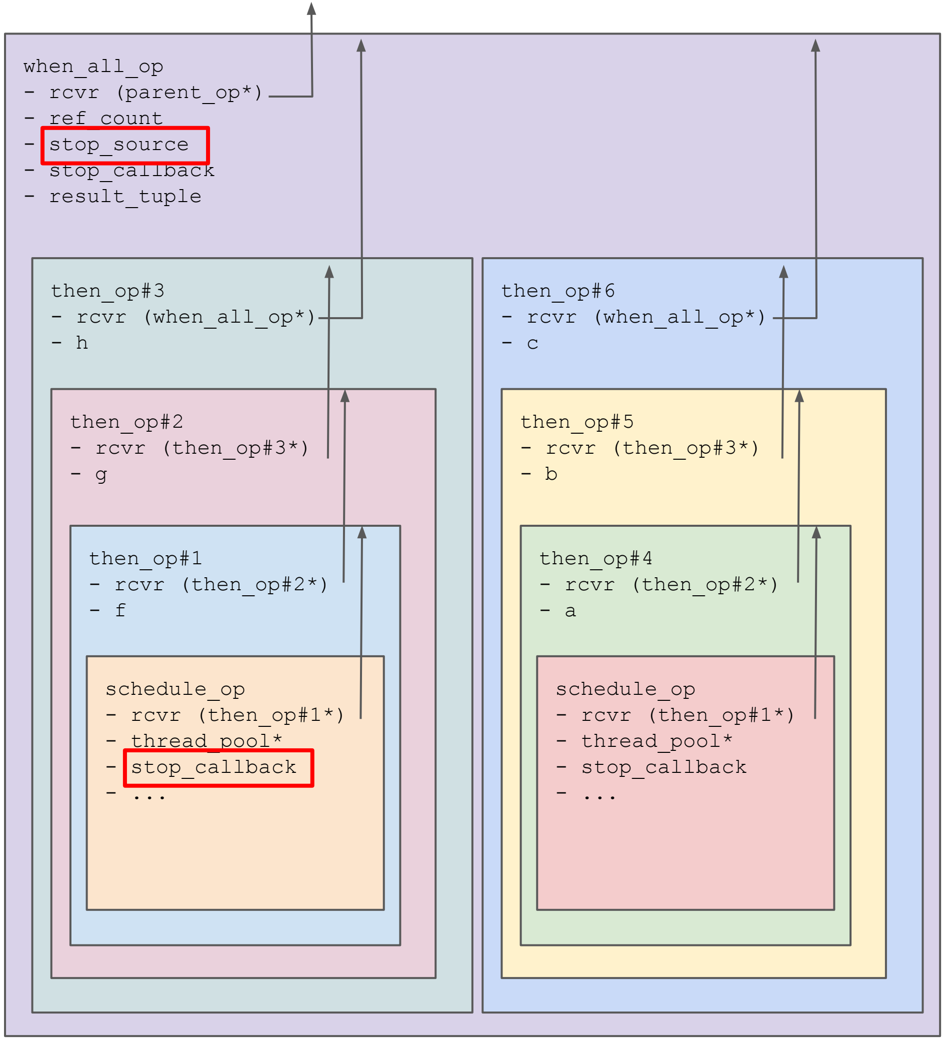
then_op#6
- rcvr (when_all_op*)
- c

then_op#5
- rcvr (then_op#3*)
- b

then_op#4
- rcvr (then_op#2*)
- a

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

```
void schedule_op::start() {
  auto st = get_stop_token(get_env(rcvr));
  stop_callback.emplace(st, on_stop{this});

  // ...
}
```



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

then_op#3
- rcvr (when_all_op*)
- h

then_op#2
- rcvr (then_op#3*)
- g

then_op#1
- rcvr (then_op#2*)
- f

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

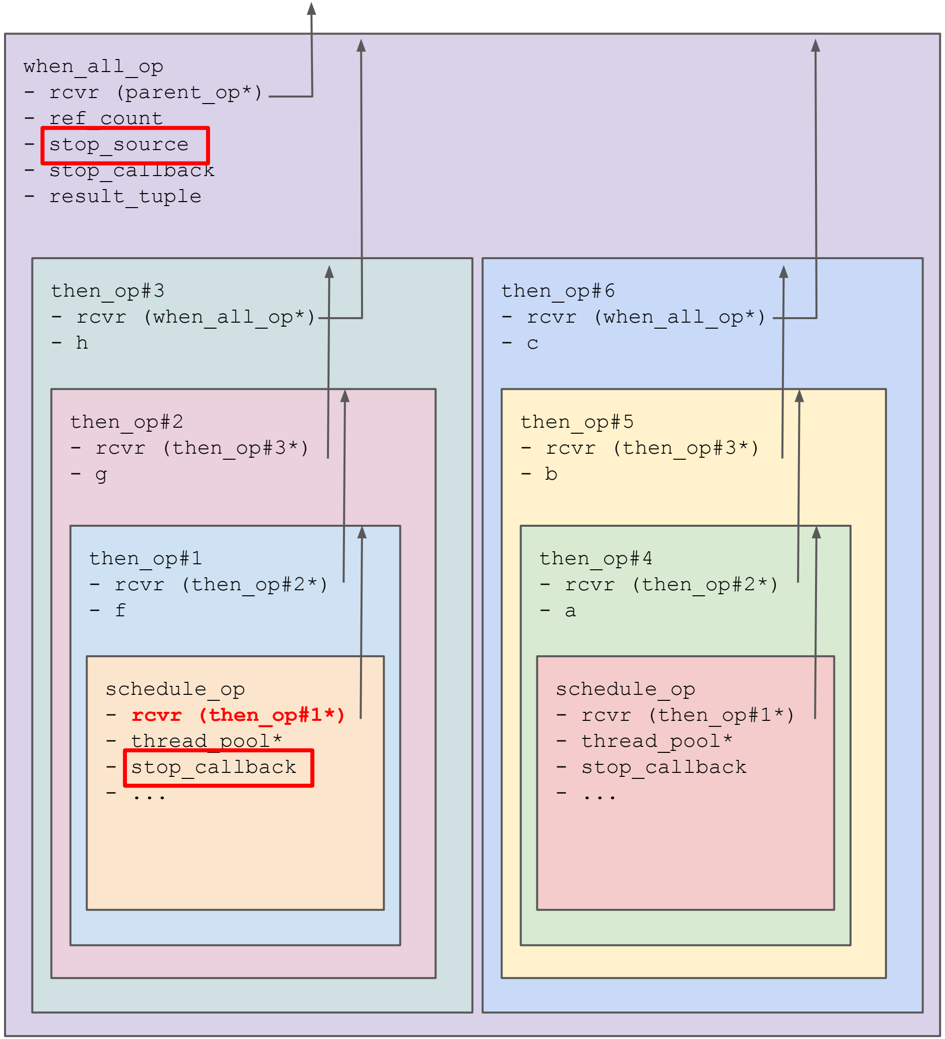
then_op#6
- rcvr (when_all_op*)
- c

then_op#5
- rcvr (then_op#3*)
- b

then_op#4
- rcvr (then_op#2*)
- a

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

```
void schedule_op::start() {
  auto st = get_stop_token(get_env(rcvr));
  stop_callback.emplace(st, on_stop{this});

  // ...
}

struct then_rcvr {
  then_op* op;

  auto get_env() const noexcept {
    return execution::get_env(op->rcvr);
  }
};
```



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

then_op#3
- rcvr (when_all_op*)
- h

then_op#2
- rcvr (then_op#3*)
- g

then_op#1
- rcvr (then_op#2*)
- f

schedule_op
- **rcvr (then_op#1*)**
- thread_pool*
- stop_callback
- ...

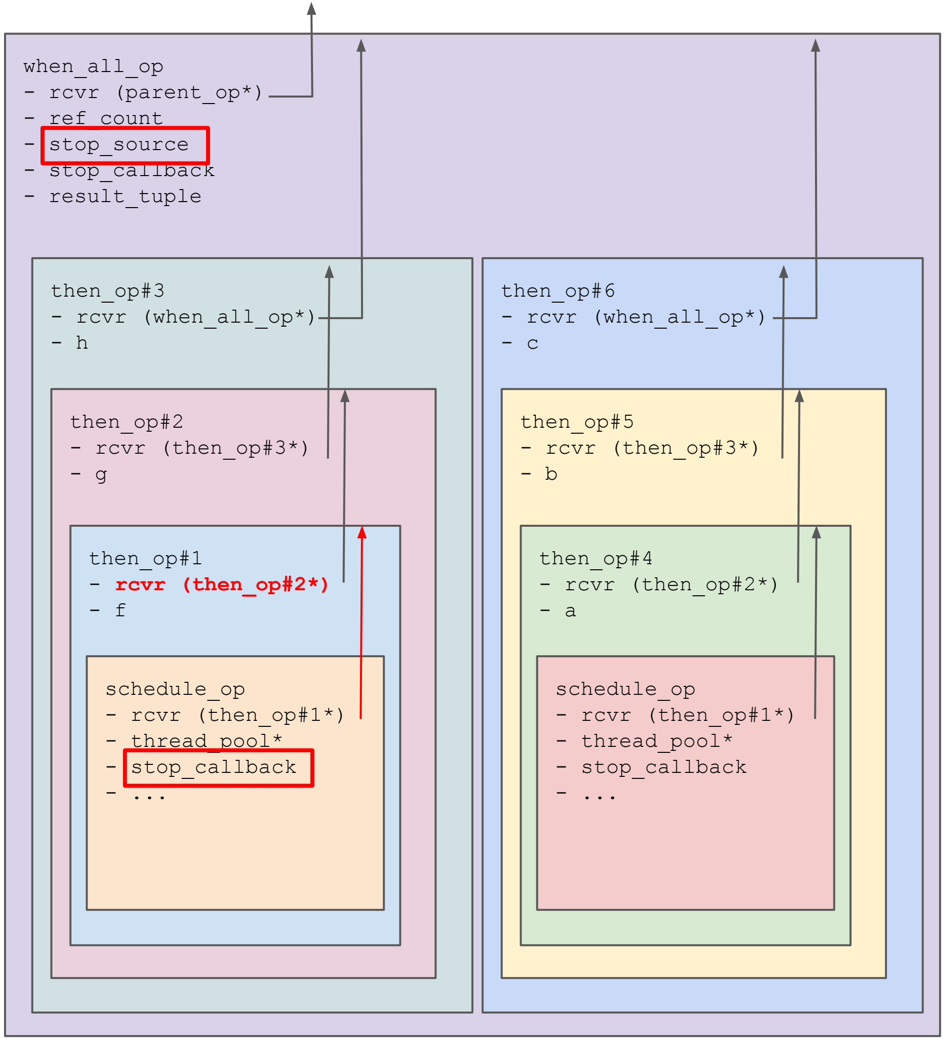
then_op#6
- rcvr (when_all_op*)
- c

then_op#5
- rcvr (then_op#3*)
- b

then_op#4
- rcvr (then_op#2*)
- a

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

```
void schedule_op::start() {
  auto st = get_stop_token(get_env(rcvr));
  stop_callback.emplace(st, on_stop{this});

  // ...
}

struct then_rcvr {
  then_op* op;

  auto get_env() const noexcept {
    return execution::get_env(op->rcvr);
  }
};
```



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

then_op#3
- rcvr (when_all_op*)
- h

then_op#2
- rcvr (then_op#3*)
- g

then_op#1
- **rcvr (then_op#2*)**
- f

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

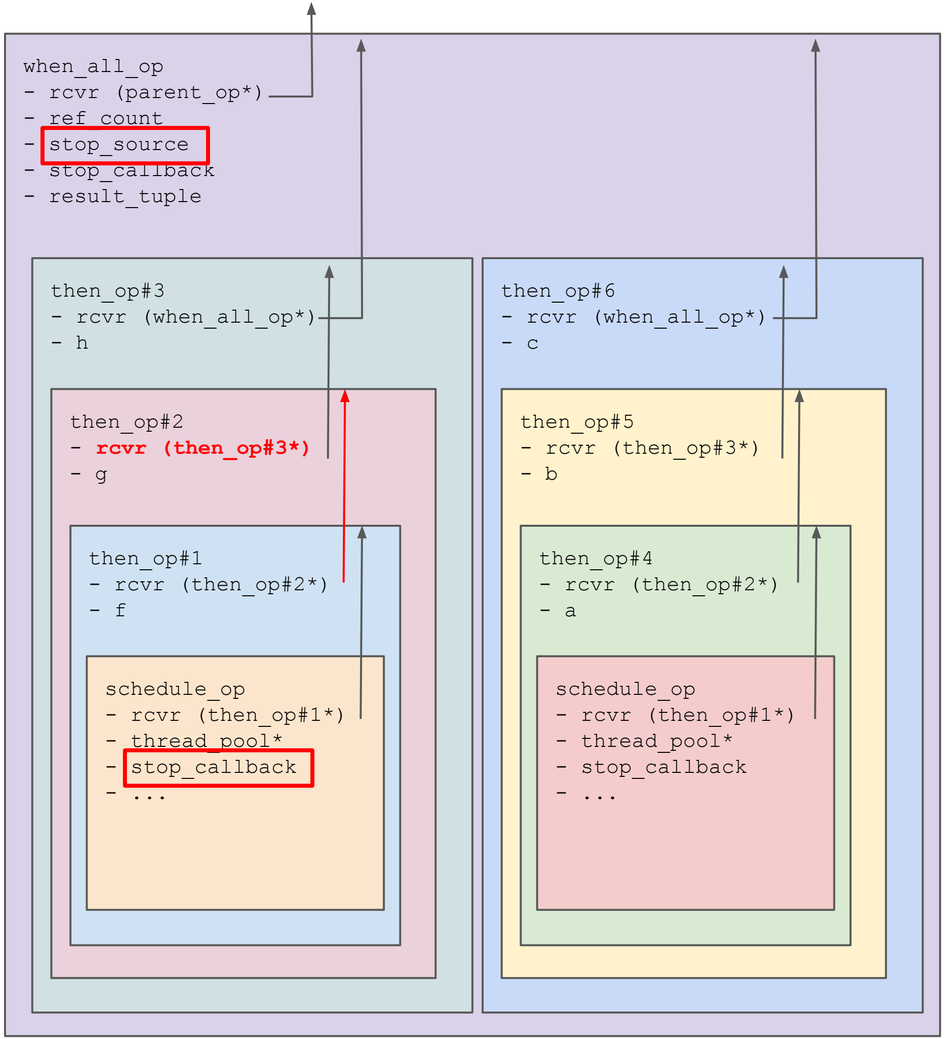
then_op#6
- rcvr (when_all_op*)
- c

then_op#5
- rcvr (then_op#3*)
- b

then_op#4
- rcvr (then_op#2*)
- a

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

```
void schedule_op::start() {
  auto st = get_stop_token(get_env(rcvr));
  stop_callback.emplace(st, on_stop{this});

  // ...
}

struct then_rcvr {
  then_op* op;

  auto get_env() const noexcept {
    return execution::get_env(op->rcvr);
  }
};
```



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

then_op#3
- rcvr (when_all_op*)
- h

then_op#2
- **rcvr (then_op#3*)**
- g

then_op#1
- rcvr (then_op#2*)
- f

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

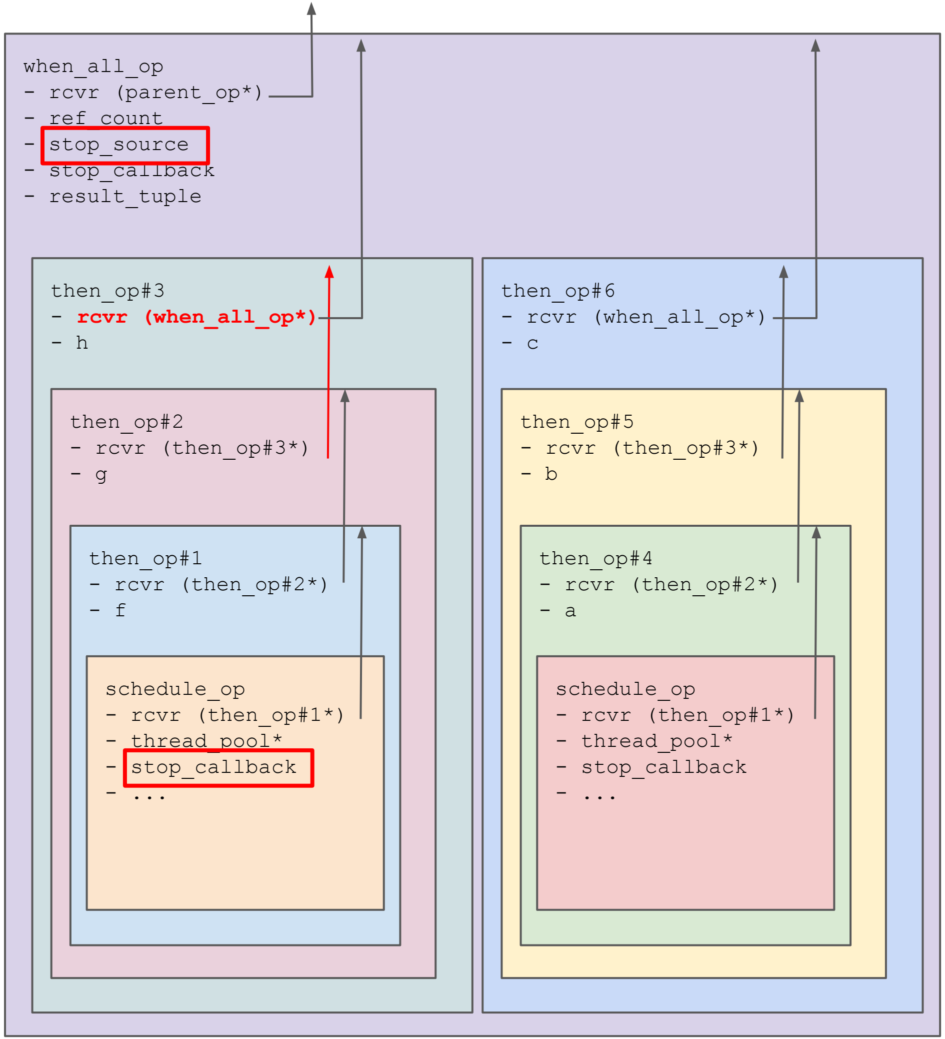
then_op#6
- rcvr (when_all_op*)
- c

then_op#5
- rcvr (then_op#3*)
- b

then_op#4
- rcvr (then_op#2*)
- a

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

```
void schedule_op::start() {
  auto st = get_stop_token(get_env(rcvr));
  stop_callback.emplace(st, on_stop{this});

  // ...
}

struct then_rcvr {
  then_op* op;

  auto get_env() const noexcept {
    return execution::get_env(op->rcvr);
  }
};

struct when_all_rcvr {
  when_all_op* op;

  when_all_env get_env() const noexcept {
    return when_all_env{op};
  }
};
```



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

then_op#3
- **rcvr (when_all_op*)**
- h

then_op#2
- rcvr (then_op#3*)
- g

then_op#1
- rcvr (then_op#2*)
- f

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

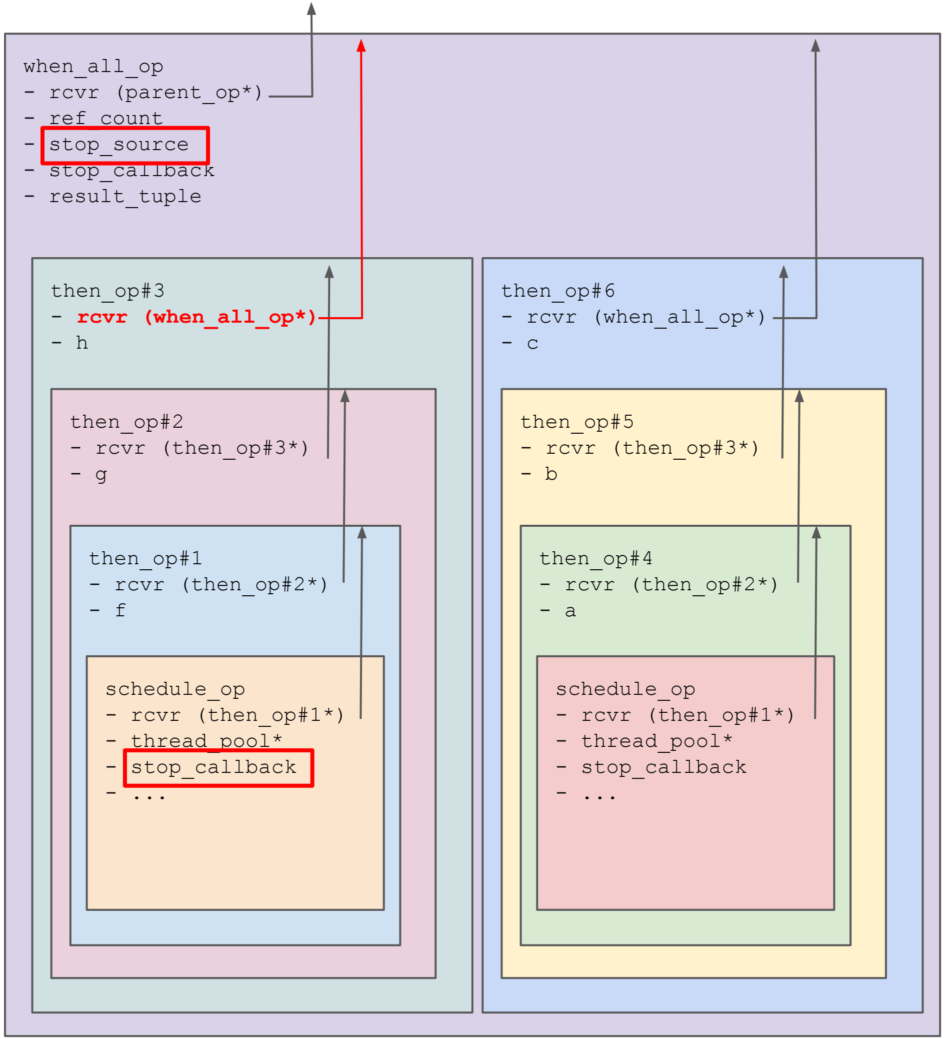
then_op#6
- rcvr (when_all_op*)
- c

then_op#5
- rcvr (then_op#3*)
- b

then_op#4
- rcvr (then_op#2*)
- a

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

```
void schedule_op::start() {
  auto st = get_stop_token(get_env(rcvr));
  stop_callback.emplace(st, on_stop{this});

  // ...
}

struct then_rcvr {
  then_op* op;

  auto get_env() const noexcept {
    return execution::get_env(op->rcvr);
  }
};

struct when_all_rcvr {
  when_all_op* op;

  when_all_env get_env() const noexcept {
    return when_all_env{op};
  }
};

struct when_all_env {
  when_all_op* op;

  auto query(get_stop_token_t) const noexcept
{
    return op->stop_source.get_token();
  }
```



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

then_op#3
- **rcvr (when_all_op*)**
- h

then_op#2
- rcvr (then_op#3*)
- g

then_op#1
- rcvr (then_op#2*)
- f

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

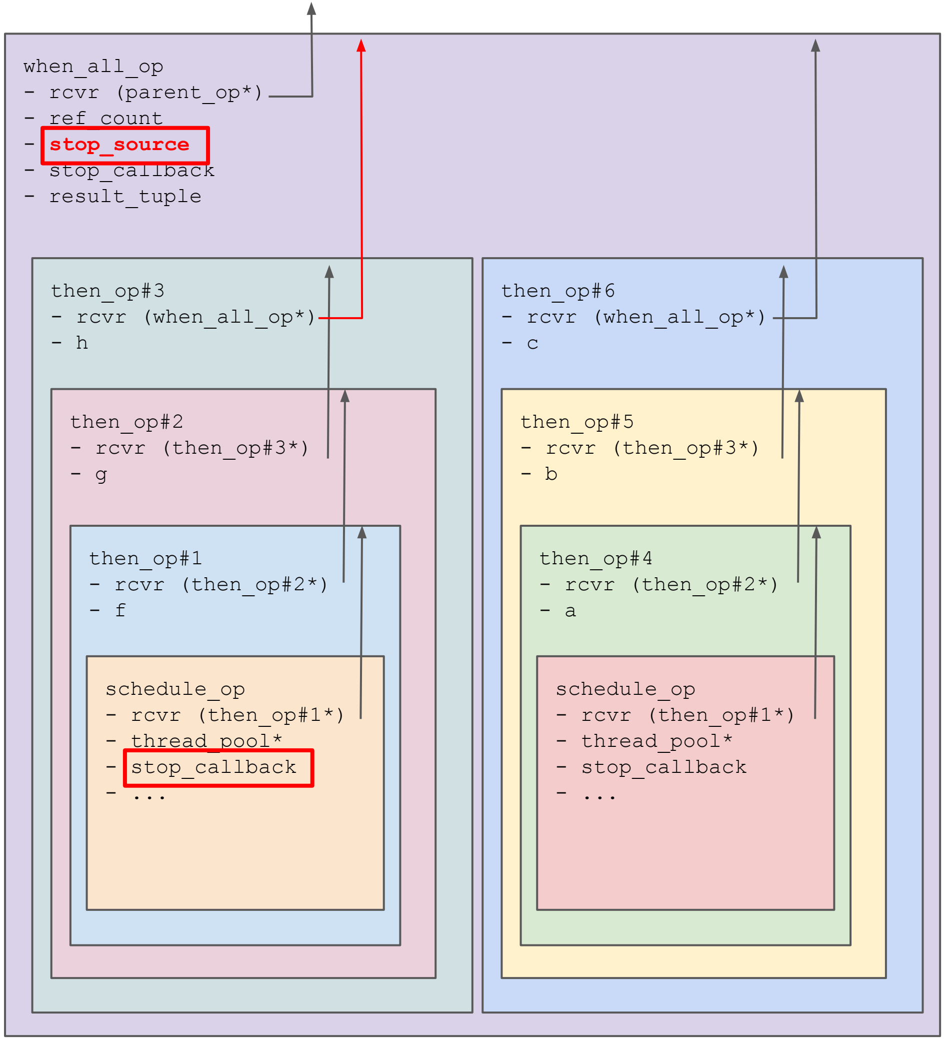
then_op#6
- rcvr (when_all_op*)
- c

then_op#5
- rcvr (then_op#3*)
- b

then_op#4
- rcvr (then_op#2*)
- a

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

```
void schedule_op::start() {
  auto st = get_stop_token(get_env(rcvr));
  stop_callback.emplace(st, on_stop{this});

  // ...
}

struct then_rcvr {
  then_op* op;

  auto get_env() const noexcept {
    return execution::get_env(op->rcvr);
  }
};

struct when_all_rcvr {
  when_all_op* op;

  when_all_env get_env() const noexcept {
    return when_all_env{op};
  }
};

struct when_all_env {
  when_all_op* op;

  auto query(get_stop_token_t) const noexcept
{
    return op->stop_source.get_token();
  }
```



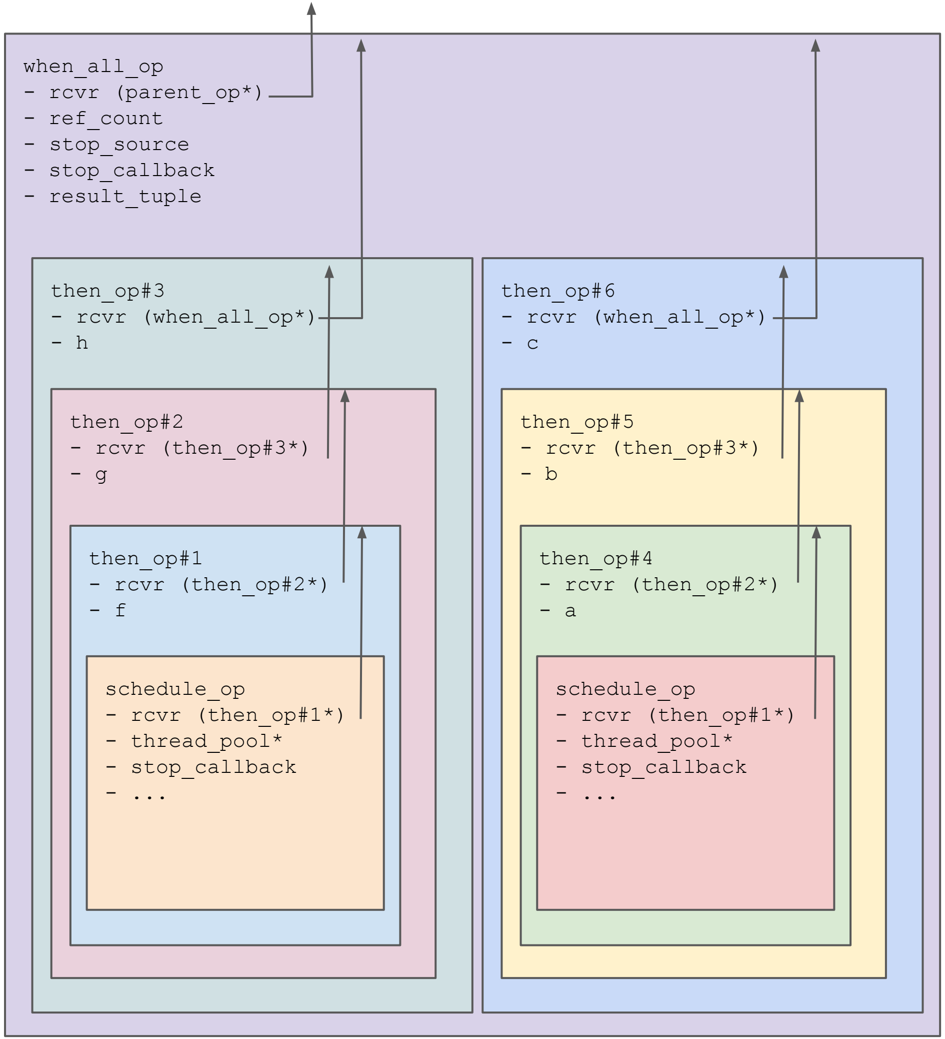
Overheads

- Each op stores receiver which has pointer to parent operation state.
    - This operation-state sub-tree is storing 9 pointers to parent operation-states.

      So 72 bytes + any extra padding due to alignment.

- Querying the environment requires “walking the stack”.
    - In this case, 4 successive pointer dereferences to obtain the stop-token.
    - Other queries satisfied by parent operations may have to walk longer chains depending on the query.

- On completion also need to dereference pointers to parent operation states.
    - e.g. to get address of ‘f’, ‘g’ and ‘h’ invocables, and address of result_tuple in when_all_op to store result.



```
when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple
```

```
then_op#3
- rcvr (when_all_op*)
- h
```

```
then_op#2
- rcvr (then_op#3*)
- g
```

```
then_op#1
- rcvr (then_op#2*)
- f
```

```
schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...
```

```
then_op#6
- rcvr (when_all_op*)
- c
```

```
then_op#5
- rcvr (then_op#3*)
- b
```

```
then_op#4
- rcvr (then_op#2*)
- a
```

```
schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...
```

Cost of Composition

Slower

```
when_all(
  then(
    then(
      then(
        schedule(thread_pool),
        f),
      g),
    h),
  then(
    then(
      then(
        schedule(thread_pool),
        a),
      b),
    c))
```

Faster

```
when_all(
  then(
    schedule(thread_pool),
    [f, g, h] {
      return h(g(f()));
    }),
  then(
    schedule(thread_pool),
    [a, b, c] {
      return c(b(a()));
    }))
```

Fewer levels => lower overhead

- smaller operation states
- less pointer chasing in queries/completion

Cost of Composition

- Cost of composing algorithms is non-zero overhead

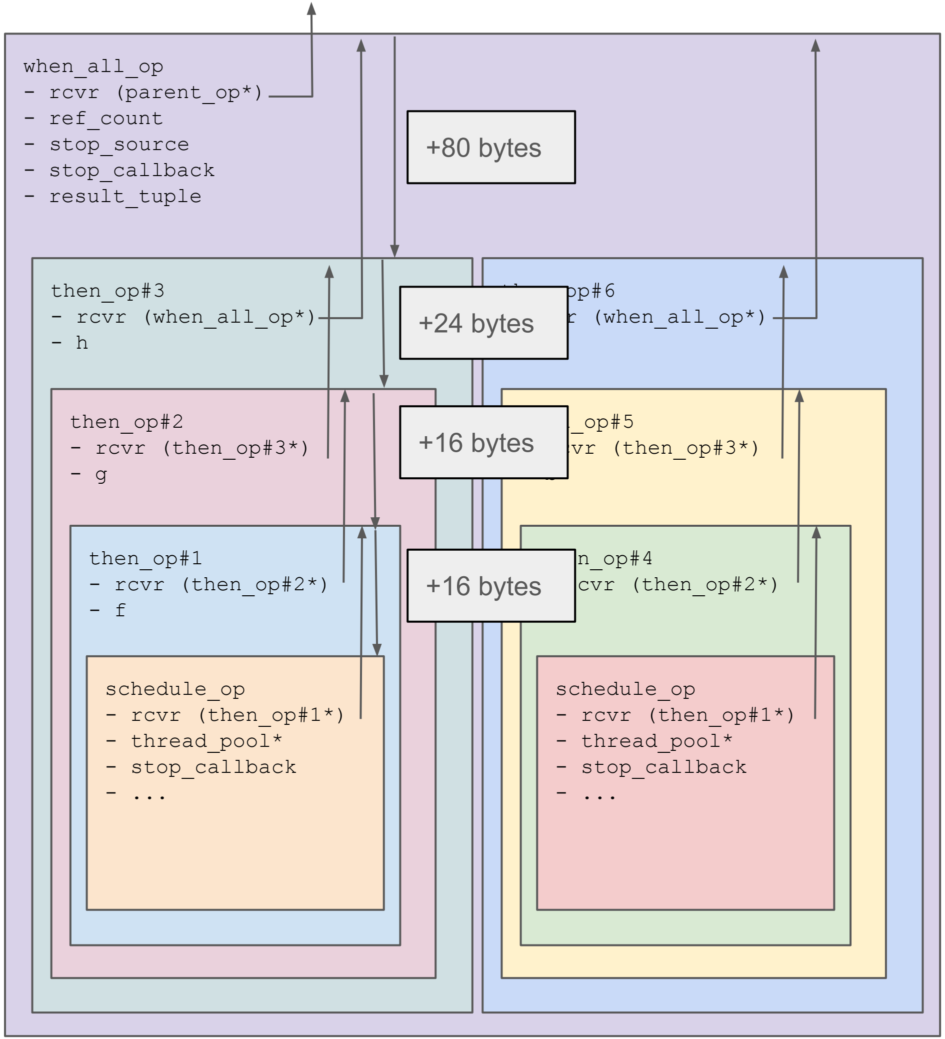
- Will encourage users to manually flatten expressions to avoid overheads
  - Not always possible.

- Will encourage users to write their own “fused” operation senders
  - More complicated, more maintenance.

An Observation

- Each child operation-state object is a sub-object of the parent operation-state object.

- Child sub object contains a pointer to parent object.
    - This will always be a value that is a constant offset from the address of the child object.



when_all_op
- rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

+80 bytes

then_op#3
- rcvr (when_all_op*)
- h

+24 bytes

op#6
r (when_all_op*)

then_op#2
- rcvr (then_op#3*)
- g

+16 bytes

_op#5
vr (then_op#3*)

then_op#1
- rcvr (then_op#2*)
- f

+16 bytes

n_op#4
cvr (then_op#2*)

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

schedule_op
- rcvr (then_op#1*)
- thread_pool*
- stop_callback
- ...

An Observation

- Each child operation-state object is a sub-object of the parent operation-state object.

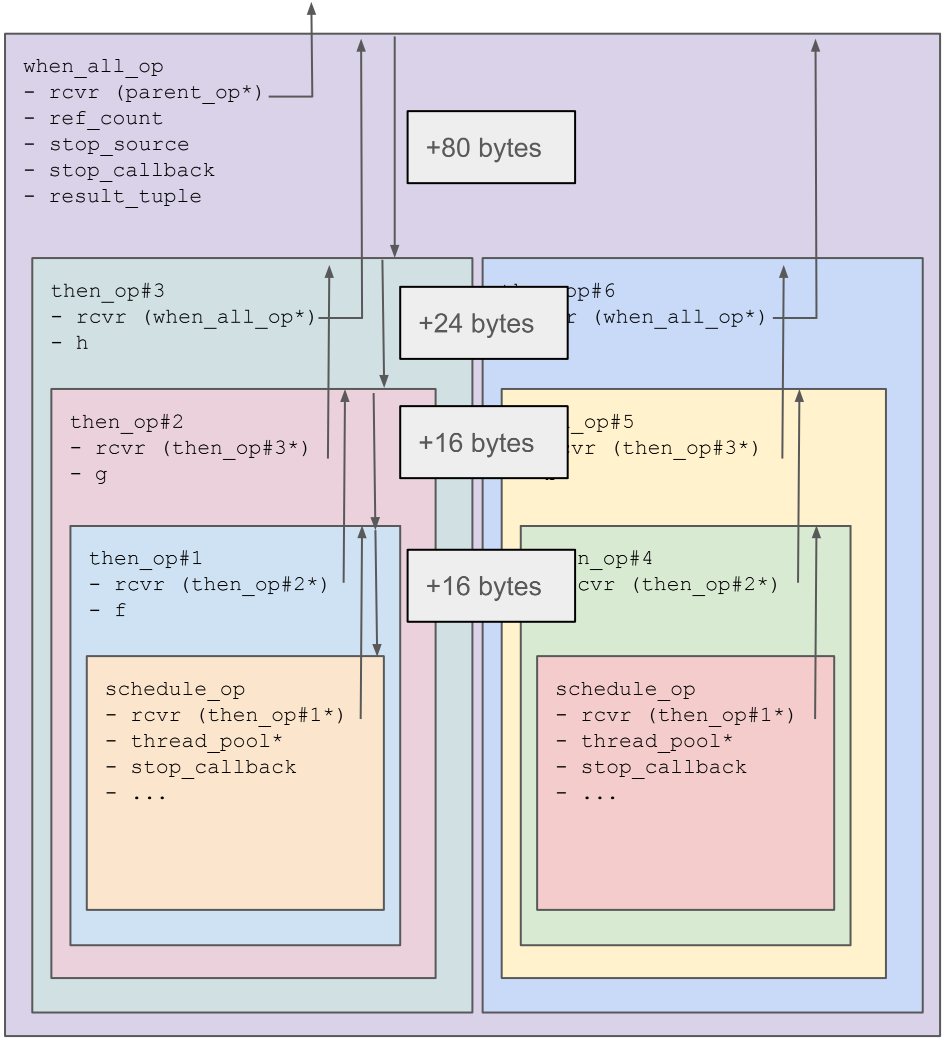
- Child sub object contains a pointer to parent object.
    - This will always be a value that is a constant offset from the address of the child object.

- **What if we could just compute the address of the parent object from the address of the child object and then recreate the receiver on-demand?**
    - Avoid needing to store the receiver.

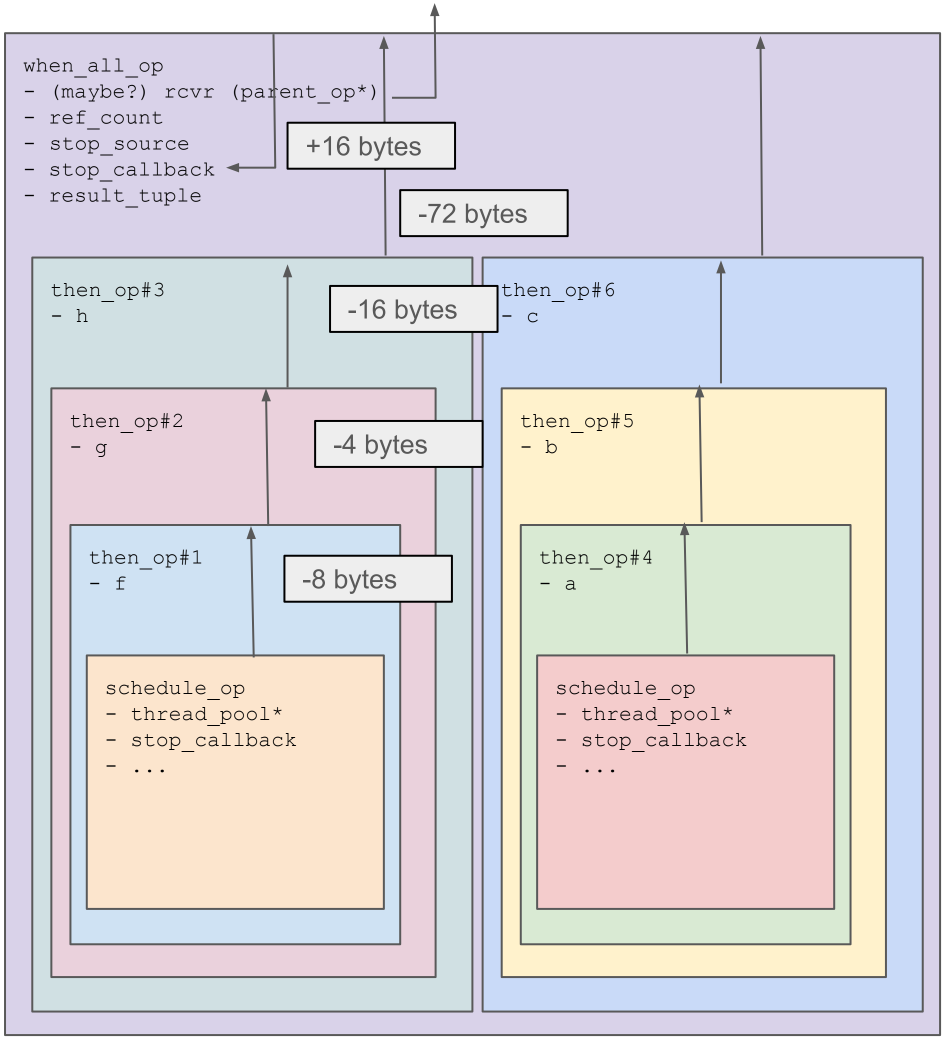
      Saves a pointer of storage for every sub-object op-state.

    - Compiler can constant-fold all of the offsets to compute the address of a parent object many levels up the stack.

      Eliminates pointer chasing for sub-object op-states.



```
void schedule_op::start() noexcept {
  // Evaluate:
  //  auto st = std::get_stop_token(std::get_env(
  //              this->get_receiver()));
  //
  // Lowers to equivalent to:
  auto* _op1 = reinterpret_cast<then_op_1*>(
       reinterpret_cast<unsigned char*>(this) - 8);
  auto* _op2 = reinterpret_cast<then_op_2*>(
       reinterpret_cast<unsigned char*>(_op1) - 4);
  auto* _op3 = reinterpret_cast<then_op_3*>(
       reinterpret_cast<unsigned char*>(_op2) - 16);
  auto* _op4 = reinterpret_cast<when_all_op*>(
       reinterpret_cast<unsigned char*>(_op3) - 72);
  auto st = _op4->stop_source.get_token();
  // ...
}
```



when_all_op
- (maybe?) rcvr (parent_op*)
- ref_count
- stop_source
- stop_callback
- result_tuple

+16 bytes

-72 bytes

then_op#3
- h

-16 bytes

then_op#6
- c

then_op#2
- g

-4 bytes

then_op#5
- b

then_op#1
- f

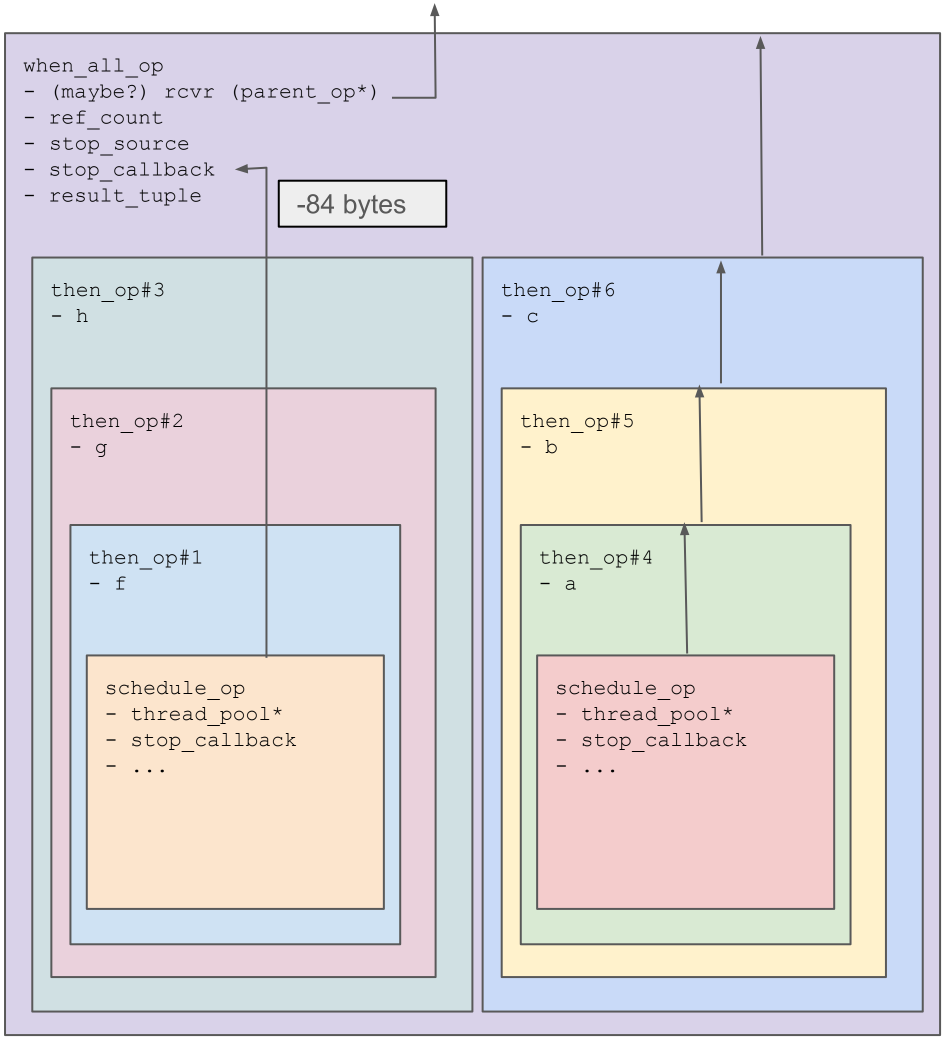
-8 bytes

then_op#4
- a

schedule_op
- thread_pool*
- stop_callback
- ...

schedule_op
- thread_pool*
- stop_callback
- ...

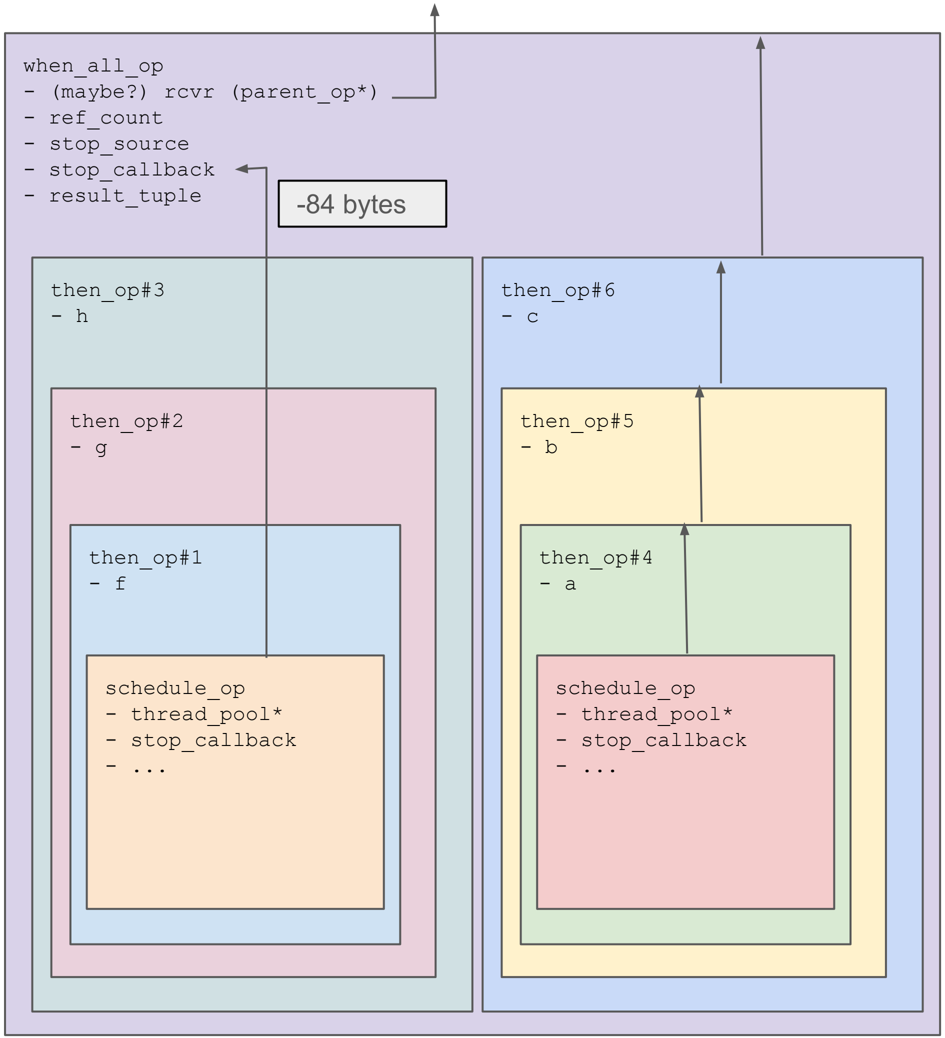
```
void schedule_op::start() noexcept {
  // Evaluate:
  //  auto st = std::get_stop_token(std::get_env(
  //              this->get_receiver()));
  //
  // Lowers to equivalent to:
  auto* ss = reinterpret_cast<inplace_stop_token*>(
       reinterpret_cast<unsigned char*>(this) - 84);
  auto st = ss->get_token();
  // ...
}
```



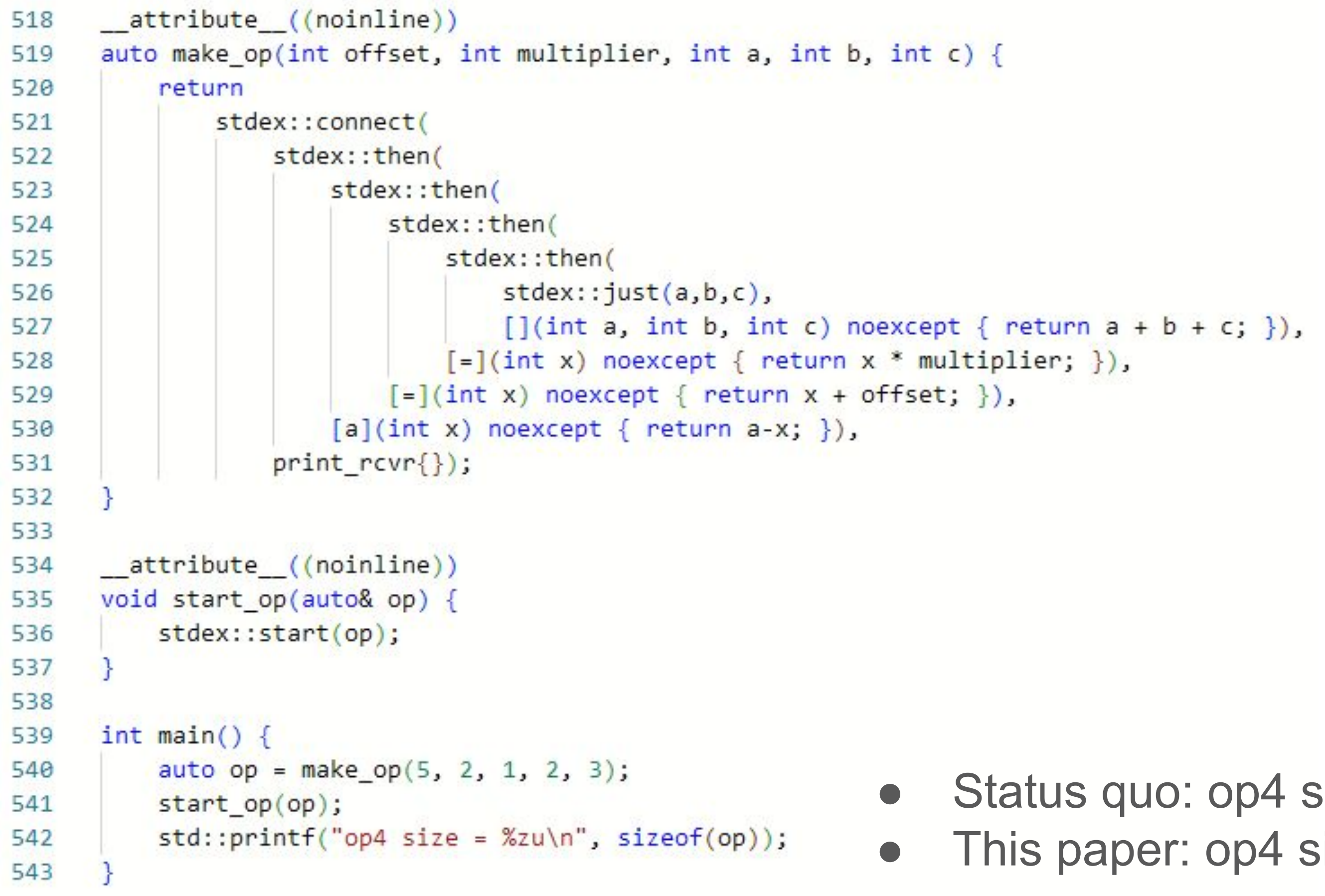
```
void schedule_op::start() noexcept {
  // Evaluate:
  //   auto st = std::get_stop_token(std::get_env(
  //                 this->get_receiver()));
  //
  // Lowers to equivalent to:
  auto* ss = reinterpret_cast<inplace_stop_token*>(
      reinterpret_cast<unsigned char*>(this) - 84);
  auto st = ss->get_token();
  // ...
}
```

- Eliminates storage of 8x pointers
  - Reduced operation-state size by *at least* 64-bytes

- Eliminates pointer-chasing when looking up stop-token/completing
  - Now a constant offset from leaf operation-state.



Example - https://godbolt.org/z/TjsbhW7T4

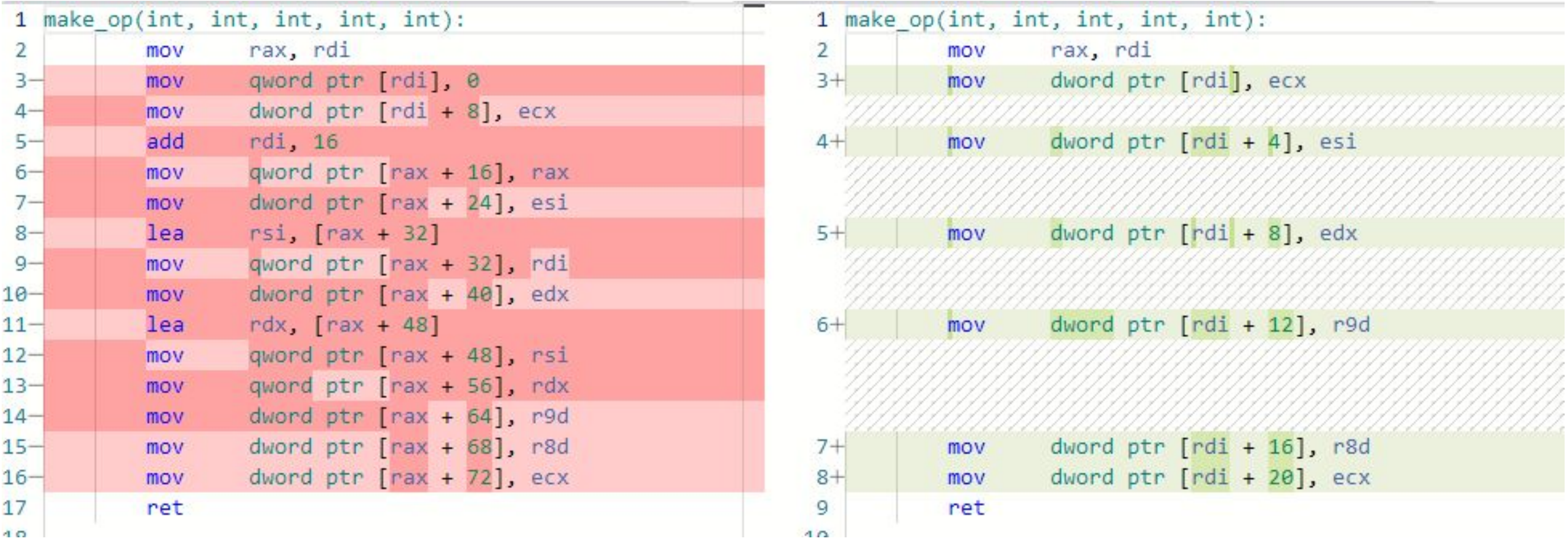


```
__attribute__((noinline))
auto make_op(int offset, int multiplier, int a, int b, int c) {
    return
        stdex::connect(
            stdex::then(
                stdex::then(
                    stdex::then(
                        stdex::then(
                            stdex::just(a,b,c),
                            [](int a, int b, int c) noexcept { return a + b + c; }),
                        [=](int x) noexcept { return x * multiplier; }),
                    [=](int x) noexcept { return x + offset; }),
                [a](int x) noexcept { return a-x; }),
            print_rcvr{});
}

__attribute__((noinline))
void start_op(auto& op) {
    stdex::start(op);
}

int main() {
    auto op = make_op(5, 2, 1, 2, 3);
    start_op(op);
    std::printf("op4 size = %zu\n", sizeof(op));
}
```

- Status quo: op4 size = 80
- This paper: op4 size = 24

Example - https://godbolt.org/z/TjsbhW7T4

Lots of extra instructions in `connect()` needed to initialize pointer-to-parent members

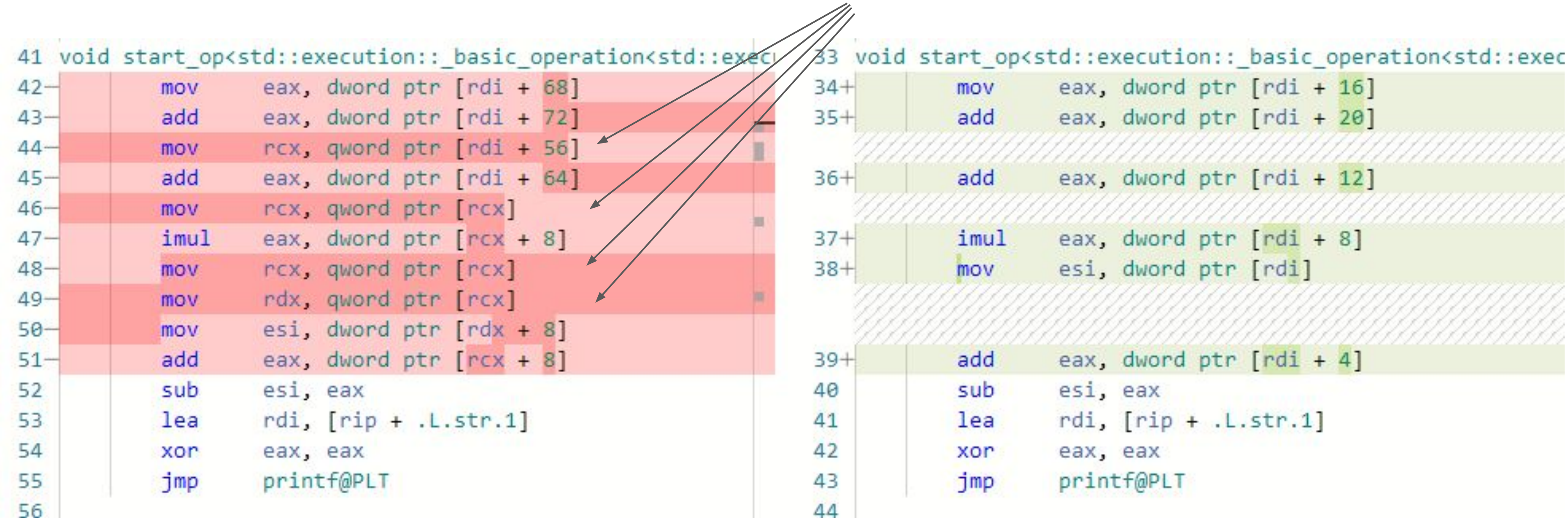


```
make_op(int, int, int, int, int):
        mov     rax, rdi
        mov     qword ptr [rdi], 0
        mov     dword ptr [rdi + 8], ecx
        add     rdi, 16
        mov     qword ptr [rax + 16], rax
        mov     dword ptr [rax + 24], esi
        lea     rsi, [rax + 32]
        mov     qword ptr [rax + 32], rdi
        mov     dword ptr [rax + 40], edx
        lea     rdx, [rax + 48]
        mov     qword ptr [rax + 48], rsi
        mov     qword ptr [rax + 56], rdx
        mov     dword ptr [rax + 64], r9d
        mov     dword ptr [rax + 68], r8d
        mov     dword ptr [rax + 72], ecx
        ret
```

```
make_op(int, int, int, int, int):
        mov     rax, rdi
        mov     dword ptr [rdi], ecx
        mov     dword ptr [rdi + 4], esi
        mov     dword ptr [rdi + 8], edx
        mov     dword ptr [rdi + 12], r9d
        mov     dword ptr [rdi + 16], r8d
        mov     dword ptr [rdi + 20], ecx
        ret
```

Example - https://godbolt.org/z/TjsbhW7T4

Extra pointer dereferences



```
void start_op<std::execution::_basic_operation<std::execu
        mov     eax, dword ptr [rdi + 68]
        add     eax, dword ptr [rdi + 72]
        mov     rcx, qword ptr [rdi + 56]
        add     eax, dword ptr [rdi + 64]
        mov     rcx, qword ptr [rcx]
        imul    eax, dword ptr [rcx + 8]
        mov     rcx, qword ptr [rcx]
        mov     rdx, qword ptr [rcx]
        mov     esi, dword ptr [rdx + 8]
        add     eax, dword ptr [rcx + 8]
        sub     esi, eax
        lea     rdi, [rip + .L.str.1]
        xor     eax, eax
        jmp     printf@PLT
```

```
void start_op<std::execution::_basic_operation<std::exec
        mov     eax, dword ptr [rdi + 16]
        add     eax, dword ptr [rdi + 20]

        add     eax, dword ptr [rdi + 12]

        imul    eax, dword ptr [rdi + 8]
        mov     esi, dword ptr [rdi]

        add     eax, dword ptr [rdi + 4]
        sub     esi, eax
        lea     rdi, [rip + .L.str.1]
        xor     eax, eax
        jmp     printf@PLT
```

Proposal

- New opt-in protocol for “inlinable receiver”
  - Allows parent/child operations to negotiate to apply the optimisation when both support it

- Applying the protocol to sender-algorithms proposed by P2300R10
  - Require implementations to define internal receivers that opt-int to the “inlinable receiver” interface
  - Would be a potential ABI break to implement this later (changes layout of operation-states)

Inlinable Receivers

```
// <execution>
namespace std::execution
{
  template<class T, class ChildOp>
  concept inlinable_receiver =
    receiver<T> &&
    requires(ChildOp* op) {
      { T::make_receiver_for(op} } noexcept -> std::same_as<T>;
    };
}
```

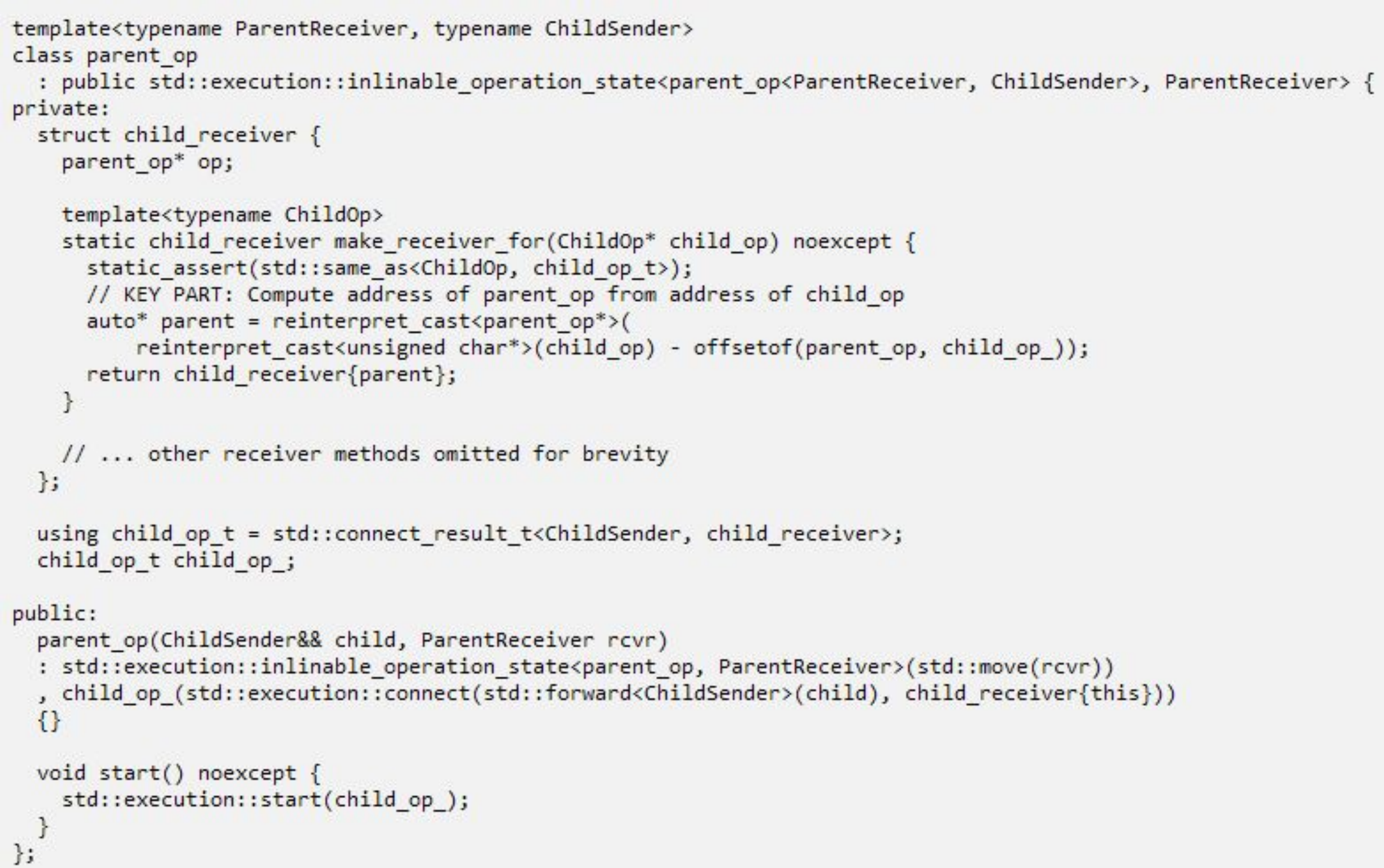
inlinable_operation_state CRTP Base Helper

```cpp
namespace std::execution
{
  // Default: Stores receiver for types that don’t implement protocol
  template<class DerivedOp, receiver Rcvr>
  struct inlinable_operation_state {
    explicit inlinable_operation_state(Rcvr&& r) noexcept(is_nothrow_move_constructible_v<Rcvr>)
      : rcvr(std::move(r) {}

    Rcvr& get_receiver() noexcept { return rcvr; }
  private:
    Rcvr rcvr;
  };

  // Specialisation: Constructs receiver on demand when receiver implements protocol
  template<class DerivedOp, receiver Rcvr>
    requires inlinable_receiver<Rcvr, DerivedOp>
  struct inlinable_operation_state<DerivedOp, Rcvr> {
    explicit inlinable_operation_state(Rcvr&&) noexcept {}
    Rcvr get_receiver() noexcept {
      return Rcvr::make_receiver_for(static_cast<DerivedOp*>( this));
    }
  };
}
```

Usage

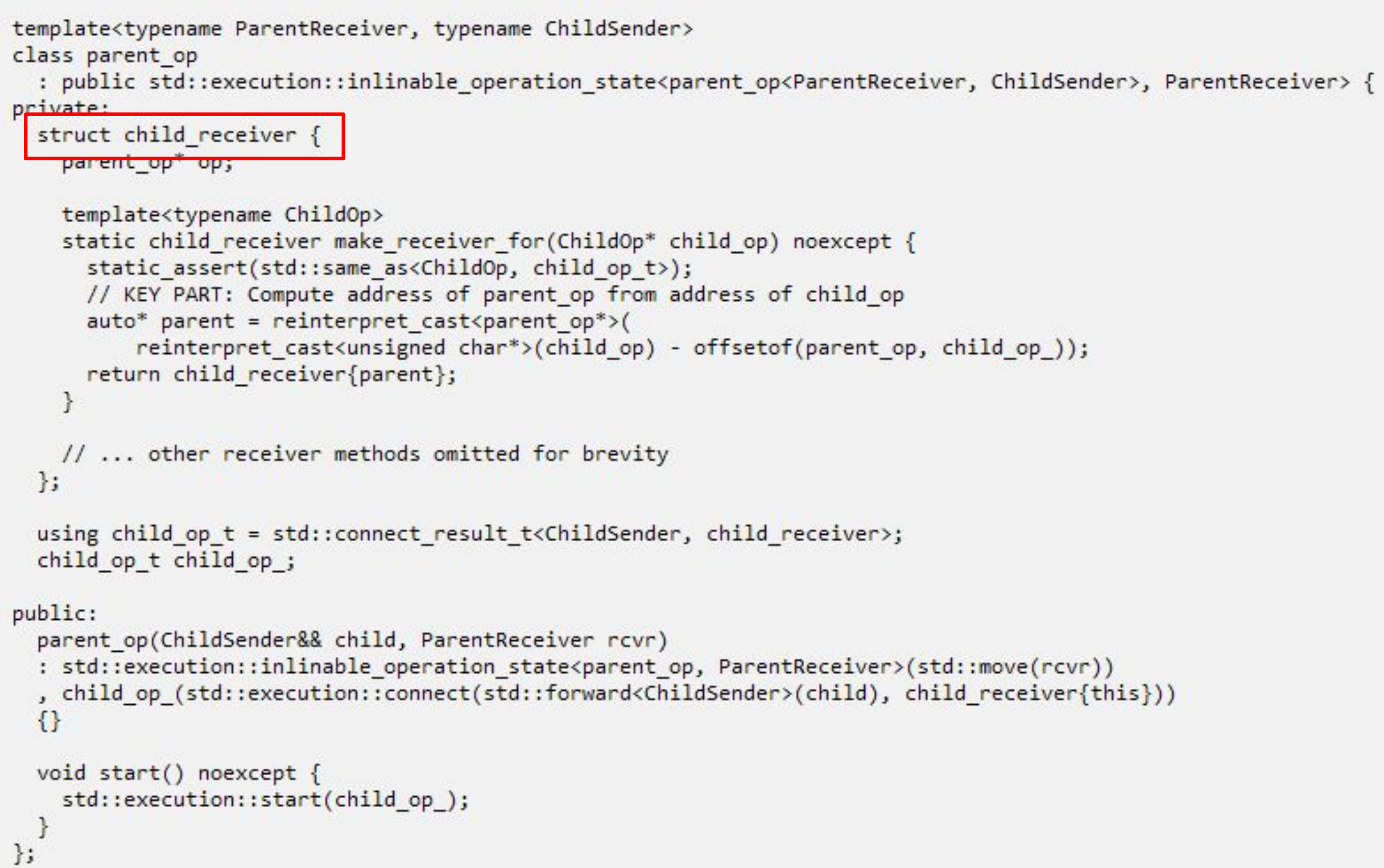
```
using std::execution::inlinable_operation_state;

template<class Rcvr>
struct my_operation_state
   : inlinable_operation_state<my_operation_state<Rcvr>, Rcvr> {
  my_operation_State(Rcvr r, int other_arg)
    : inlinable_operation_state<my_operation_state, Rcvr>(std::move(r))
    , other_state(other_arg)
  {}

  void start() & noexcept {
    decltype(auto) rcvr = this->get_receiver();
    // use rcvr…
  }

private:
  int other_state;
};
```

Implementing the `inlinable_receiver` protocol



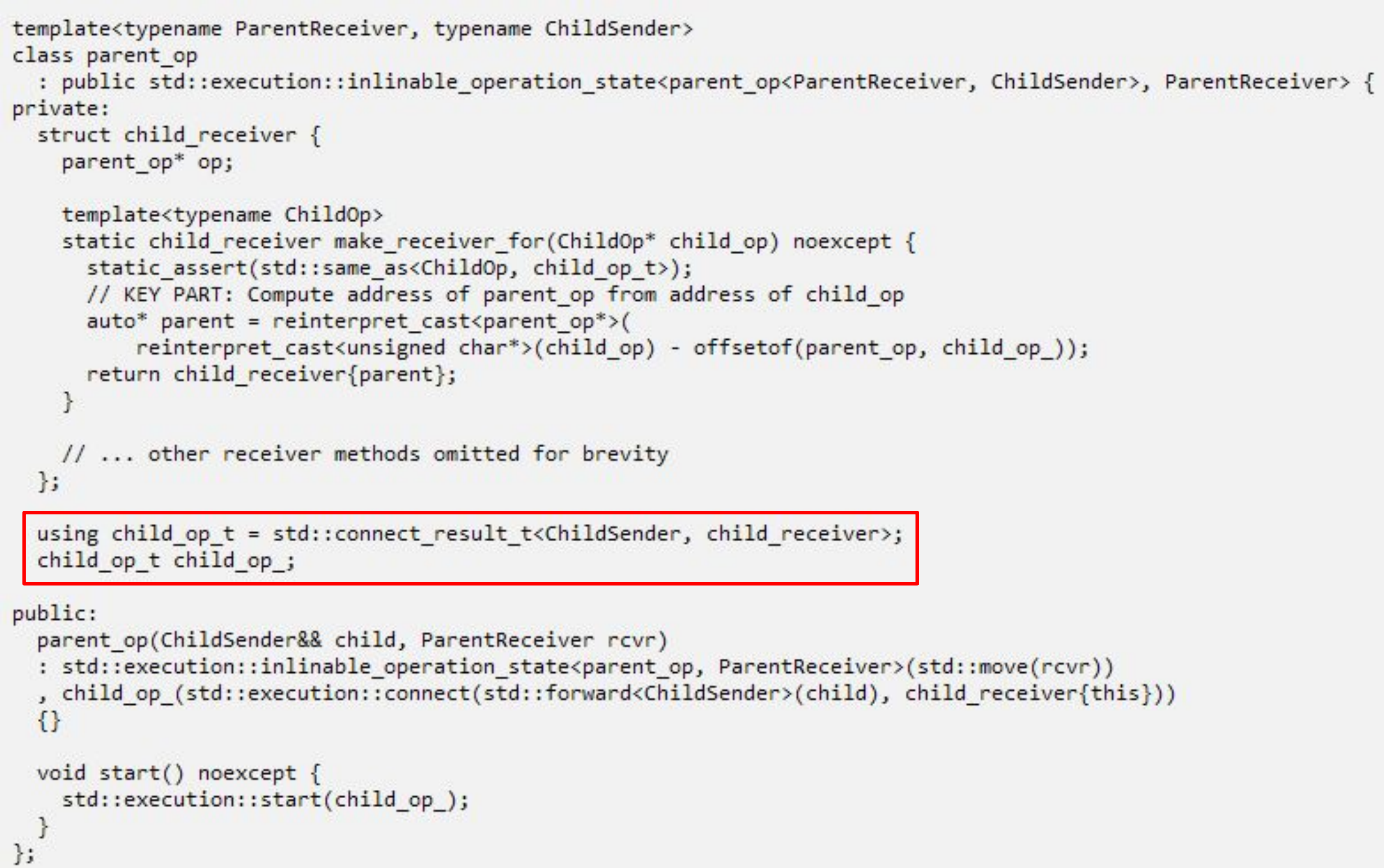
```cpp
template<typename ParentReceiver, typename ChildSender>
class parent_op
  : public std::execution::inlinable_operation_state<parent_op<ParentReceiver, ChildSender>, ParentReceiver> {
private:
  struct child_receiver {
    parent_op* op;

    template<typename ChildOp>
    static child_receiver make_receiver_for(ChildOp* child_op) noexcept {
      static_assert(std::same_as<ChildOp, child_op_t>);
      // KEY PART: Compute address of parent_op from address of child_op
      auto* parent = reinterpret_cast<parent_op*>(
          reinterpret_cast<unsigned char*>(child_op) - offsetof(parent_op, child_op_));
      return child_receiver{parent};
    }

    // ... other receiver methods omitted for brevity
  };

  using child_op_t = std::connect_result_t<ChildSender, child_receiver>;
  child_op_t child_op_;

public:
  parent_op(ChildSender&& child, ParentReceiver rcvr)
  : std::execution::inlinable_operation_state<parent_op, ParentReceiver>(std::move(rcvr))
  , child_op_(std::execution::connect(std::forward<ChildSender>(child), child_receiver{this}))
  {}

  void start() noexcept {
    std::execution::start(child_op_);
  }
};
```

Implementing the `inlinable_receiver` protocol



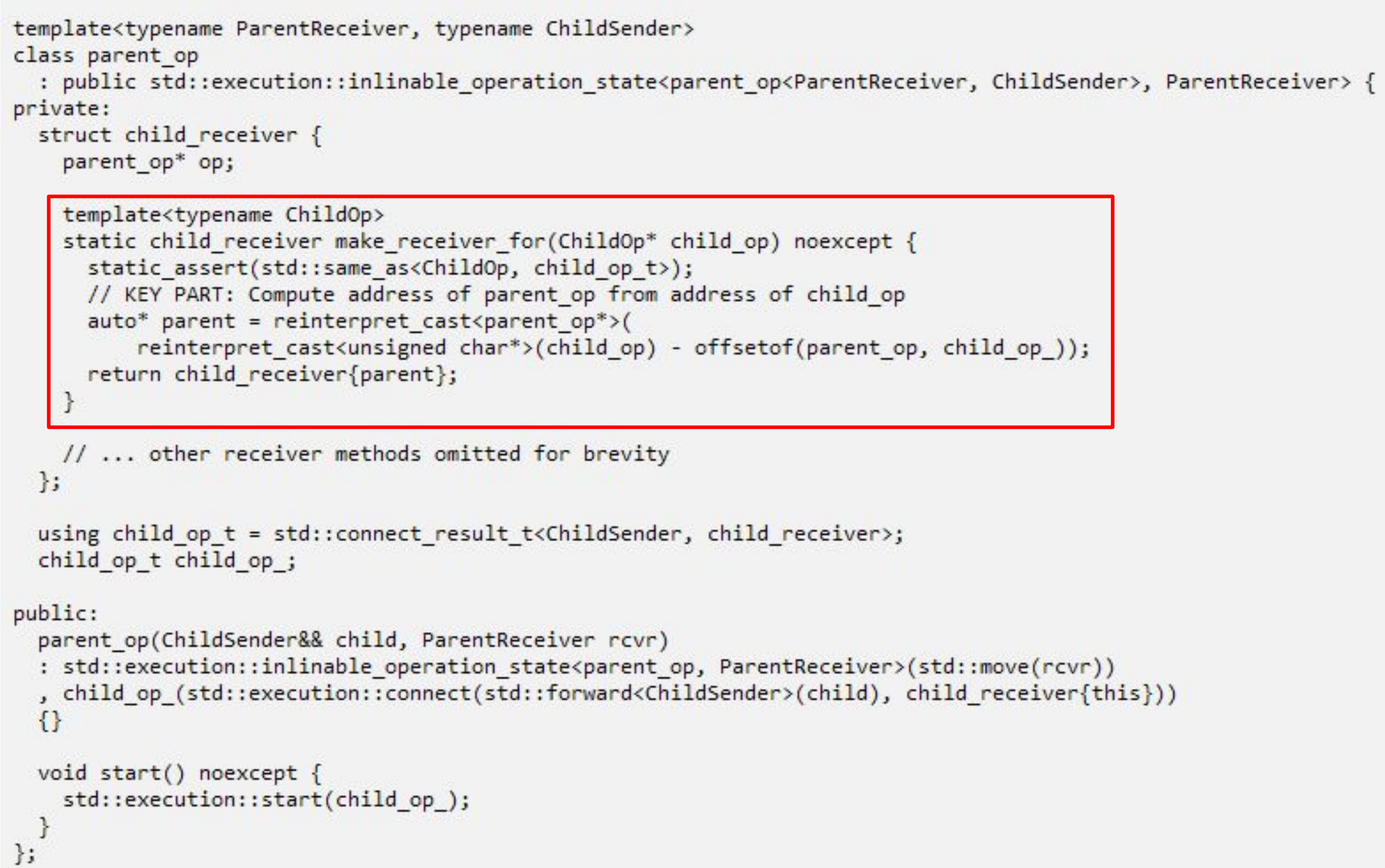
```
template<typename ParentReceiver, typename ChildSender>
class parent_op
  : public std::execution::inlinable_operation_state<parent_op<ParentReceiver, ChildSender>, ParentReceiver> {
private:
  struct child_receiver {
    parent_op* op;

    template<typename ChildOp>
    static child_receiver make_receiver_for(ChildOp* child_op) noexcept {
      static_assert(std::same_as<ChildOp, child_op_t>);
      // KEY PART: Compute address of parent_op from address of child_op
      auto* parent = reinterpret_cast<parent_op*>(
          reinterpret_cast<unsigned char*>(child_op) - offsetof(parent_op, child_op_));
      return child_receiver{parent};
    }

    // ... other receiver methods omitted for brevity
  };

  using child_op_t = std::connect_result_t<ChildSender, child_receiver>;
  child_op_t child_op_;

public:
  parent_op(ChildSender&& child, ParentReceiver rcvr)
  : std::execution::inlinable_operation_state<parent_op, ParentReceiver>(std::move(rcvr))
  , child_op_(std::execution::connect(std::forward<ChildSender>(child), child_receiver{this}))
  {}

  void start() noexcept {
    std::execution::start(child_op_);
  }
};
```

Implementing the `inlinable_receiver` protocol



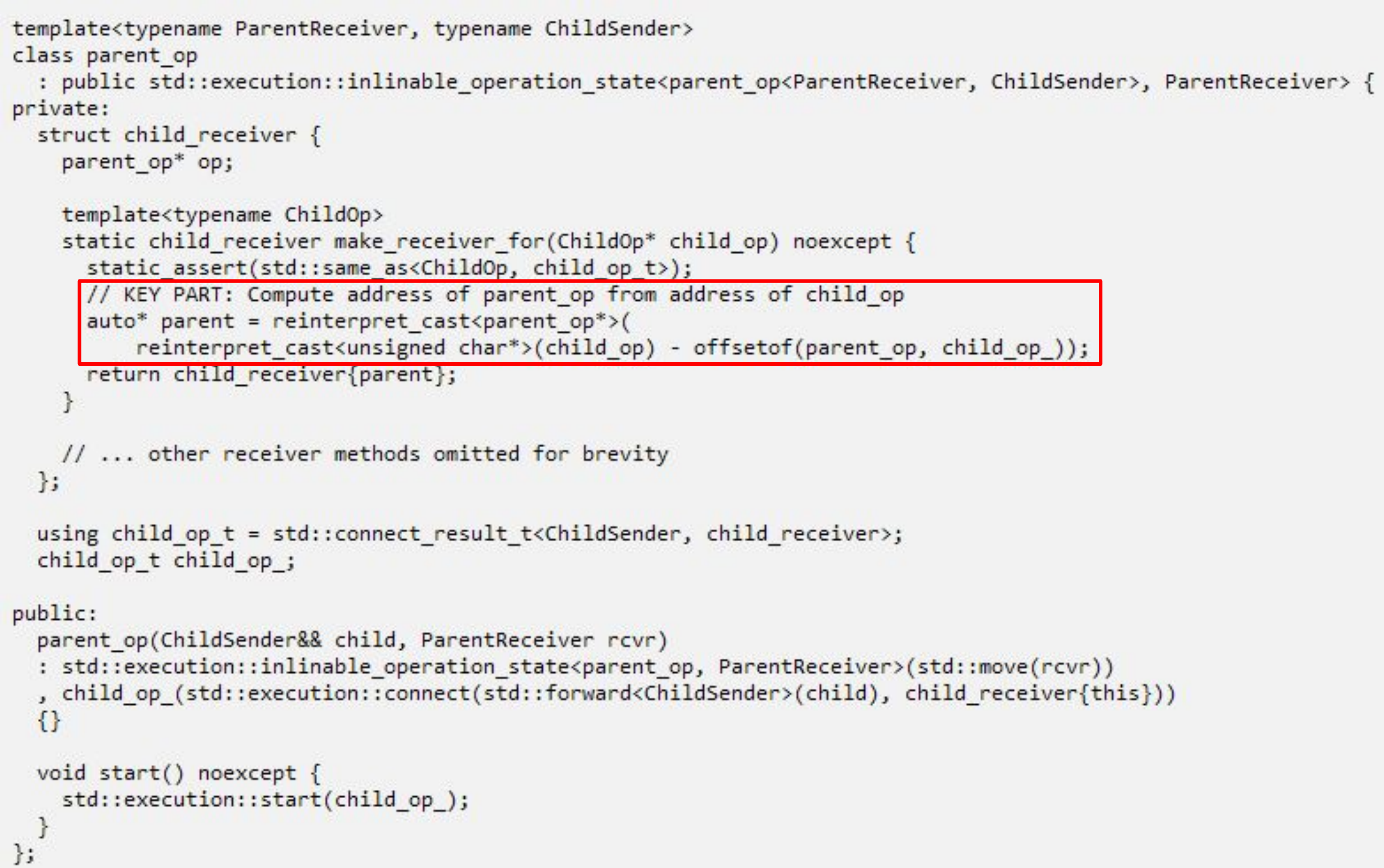
```
template<typename ParentReceiver, typename ChildSender>
class parent_op
  : public std::execution::inlinable_operation_state<parent_op<ParentReceiver, ChildSender>, ParentReceiver> {
private:
  struct child_receiver {
    parent_op* op;

    template<typename ChildOp>
    static child_receiver make_receiver_for(ChildOp* child_op) noexcept {
      static_assert(std::same_as<ChildOp, child_op_t>);
      // KEY PART: Compute address of parent_op from address of child_op
      auto* parent = reinterpret_cast<parent_op*>(
          reinterpret_cast<unsigned char*>(child_op) - offsetof(parent_op, child_op_));
      return child_receiver{parent};
    }

    // ... other receiver methods omitted for brevity
  };

  using child_op_t = std::connect_result_t<ChildSender, child_receiver>;
  child_op_t child_op_;

public:
  parent_op(ChildSender&& child, ParentReceiver rcvr)
  : std::execution::inlinable_operation_state<parent_op, ParentReceiver>(std::move(rcvr))
  , child_op_(std::execution::connect(std::forward<ChildSender>(child), child_receiver{this}))
  {}

  void start() noexcept {
    std::execution::start(child_op_);
  }
};
```

Implementing the `inlinable_receiver` protocol



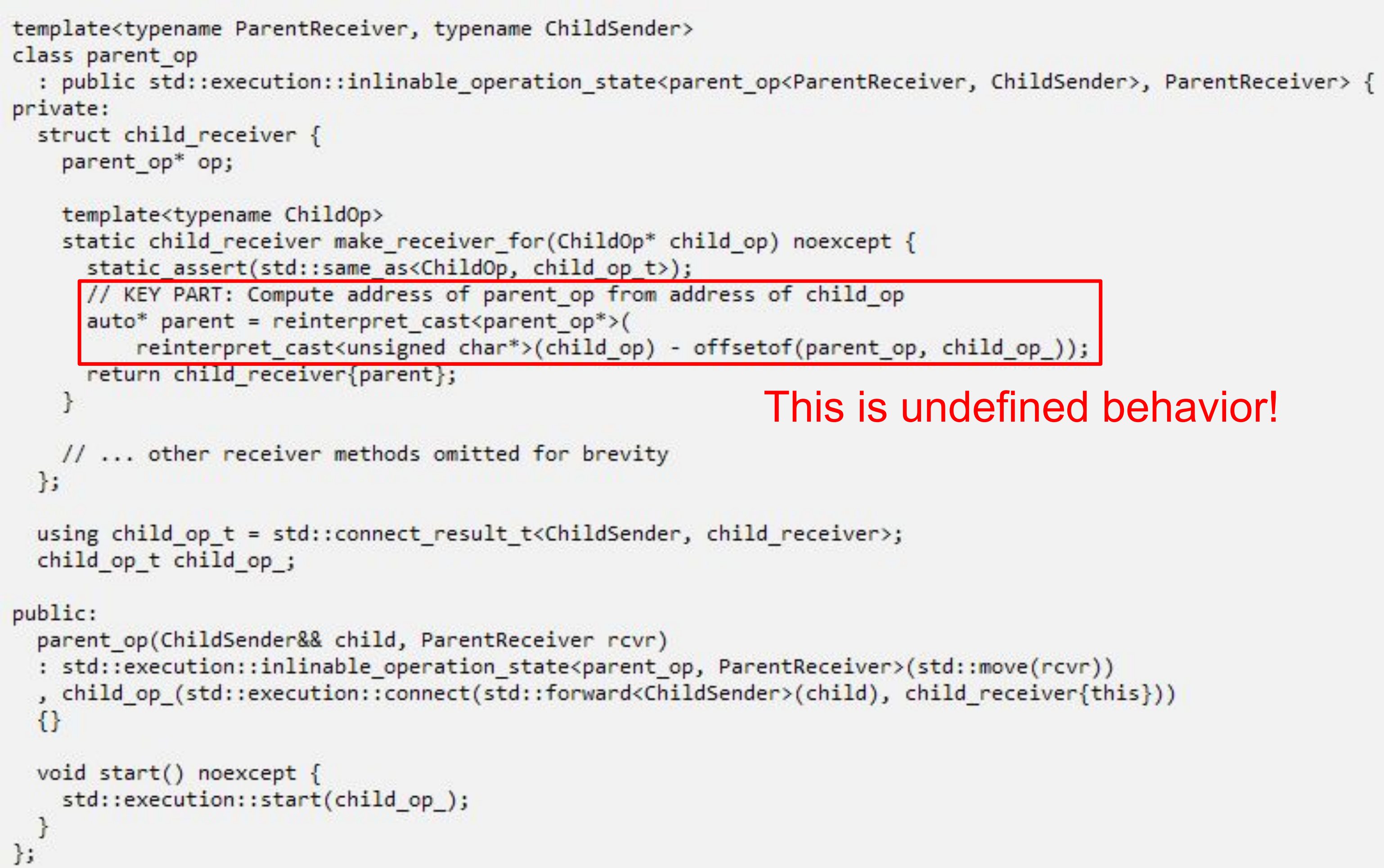
```
template<typename ParentReceiver, typename ChildSender>
class parent_op
  : public std::execution::inlinable_operation_state<parent_op<ParentReceiver, ChildSender>, ParentReceiver> {
private:
  struct child_receiver {
    parent_op* op;

    template<typename ChildOp>
    static child_receiver make_receiver_for(ChildOp* child_op) noexcept {
      static_assert(std::same_as<ChildOp, child_op_t>);
      // KEY PART: Compute address of parent_op from address of child_op
      auto* parent = reinterpret_cast<parent_op*>(
          reinterpret_cast<unsigned char*>(child_op) - offsetof(parent_op, child_op_));
      return child_receiver{parent};
    }

    // ... other receiver methods omitted for brevity
  };

  using child_op_t = std::connect_result_t<ChildSender, child_receiver>;
  child_op_t child_op_;

public:
  parent_op(ChildSender&& child, ParentReceiver rcvr)
  : std::execution::inlinable_operation_state<parent_op, ParentReceiver>(std::move(rcvr))
  , child_op_(std::execution::connect(std::forward<ChildSender>(child), child_receiver{this}))
  {}

  void start() noexcept {
    std::execution::start(child_op_);
  }
};
```

Implementing the `inlinable_receiver` protocol



```cpp
template<typename ParentReceiver, typename ChildSender>
class parent_op
  : public std::execution::inlinable_operation_state<parent_op<ParentReceiver, ChildSender>, ParentReceiver> {
private:
  struct child_receiver {
    parent_op* op;

    template<typename ChildOp>
    static child_receiver make_receiver_for(ChildOp* child_op) noexcept {
      static_assert(std::same_as<ChildOp, child_op_t>);
      // KEY PART: Compute address of parent_op from address of child_op
      auto* parent = reinterpret_cast<parent_op*>(
          reinterpret_cast<unsigned char*>(child_op) - offsetof(parent_op, child_op_));
      return child_receiver{parent};
    }

    // ... other receiver methods omitted for brevity
  };

  using child_op_t = std::connect_result_t<ChildSender, child_receiver>;
  child_op_t child_op_;

public:
  parent_op(ChildSender&& child, ParentReceiver rcvr)
  : std::execution::inlinable_operation_state<parent_op, ParentReceiver>(std::move(rcvr))
  , child_op_(std::execution::connect(std::forward<ChildSender>(child), child_receiver{this}))
  {}

  void start() noexcept {
    std::execution::start(child_op_);
  }
};
```

Implementing the `inlinable_receiver` protocol



```
template<typename ParentReceiver, typename ChildSender>
class parent_op
  : public std::execution::inlinable_operation_state<parent_op<ParentReceiver, ChildSender>, ParentReceiver> {
private:
  struct child_receiver {
    parent_op* op;

    template<typename ChildOp>
    static child_receiver make_receiver_for(ChildOp* child_op) noexcept {
      static_assert(std::same_as<ChildOp, child_op_t>);
      // KEY PART: Compute address of parent_op from address of child_op
      auto* parent = reinterpret_cast<parent_op*>(
          reinterpret_cast<unsigned char*>(child_op) - offsetof(parent_op, child_op_));
      return child_receiver{parent};
    }

    // ... other receiver methods omitted for brevity
  };

  using child_op_t = std::connect_result_t<ChildSender, child_receiver>;
  child_op_t child_op_;

public:
  parent_op(ChildSender&& child, ParentReceiver rcvr)
  : std::execution::inlinable_operation_state<parent_op, ParentReceiver>(std::move(rcvr))
  , child_op_(std::execution::connect(std::forward<ChildSender>(child), child_receiver{this}))
  {}

  void start() noexcept {
    std::execution::start(child_op_);
  }
};
```

This is undefined behavior!

Getting address of parent object from sub-object

- If sub-object is an unambiguous base-class of parent-object
  - In this case can use `static_cast` to down-cast sub-object address to parent-object address
  - See [expr.static.cast] p11

- If sub-object and parent-objects are “pointer-interconvertible”
  - In this case can use `reinterpret_cast` to cast from pointer to sub-object to pointer to parent-object.
  - See [basic.compound] p5

“pointer interconvertible”

Two objects are “pointer interconvertible” only if:

- the parent-object is a union and the sub-object is a non-static data-member of that union; or
- the parent-object is a “standard layout” class object and the sub-object is the first non-static data-member of the parent-object or any base-class sub-object of the parent-object; or
    - See [class.prop] p3 for definition of “standard layout”
- there exists an intermediate sub-object, C, such that the parent-object is pointer-interconvertible with C and C is pointer-interconvertible with the sub-object (i.e. the relationship is transitive)

What does this mean?

- A parent operation-state object with multiple child operation-states is not going to be able to use the “standard layout” first member case.
- Operation state types are not always going to be “standard layout” anyway.

Therefore:

- Child operation-state objects need to be base-classes, or a first member of a standard-layout base class.
  - Then we can cast from the address of that first member to the address of the base-class using `reinterpret_cast`.
  - Then we can down-cast from the base-class to the derived class using `static_cast`.

Implementation Helper (not proposed)

```
template<class ParentOp, class Tag, class Env, class ChildSender>
class child_operation_state {
  struct receiver {
    template<typename ChildOp>
    static receiver make_receiver_for(ChildOp* child) noexcept { /* … */ }

    // get_env/set_value/set_error/set_stopped - forward to calls to ParentOp with Tag{}.

    ParentOp* parent;
  };

  using child_op_t = connect_result_t<ChildSender, receiver>;

protected:
  child_operation_state(ChildSender&& sndr) {
    ::new (&storage) child_op_t(execution::connect(std::forward<ChildSender>(sndr),
                                                   receiver{static_cast<ParentOp*>(this)}));
  }

  ~child_operation_state() {
    reinterpret_cast<child_op_t*>(&storage)->~child_op_t();
  }

private:
  alignas(child_op_t) unsigned char storage[sizeof(child_op_t)];
};
```

Implementation Helper (not proposed)

```
template<class ParentOp, class Tag, class Env, class ChildSender>
class child_operation_state {
  struct receiver {
    template<typename ChildOp>
    static receiver make_receiver_for(ChildOp* child) noexcept { /* … */ }

    // get_env/set_value/set_error/set_stopped - forward to calls to ParentOp with Tag{}.

    ParentOp* parent;
  };

  using child_op_t = connect_result_t<ChildSender, receiver>;

protected:
  child_operation_state(ChildSender&& sndr) {
    ::new (&storage) child_op_t(execution::connect(std::forward<ChildSender>(sndr),
                                                   receiver{static_cast<ParentOp*>(this)}));
  }

  ~child_operation_state() {
    reinterpret_cast<child_op_t*>(&storage)->~child_op_t();
  }

private:
  alignas(child_op_t) unsigned char storage[sizeof(child_op_t)];
};
```

Implementation Helper (not proposed)

```
template<class ParentOp, class Tag, class Env, class ChildSender>
class child_operation_state {
  struct receiver {
    template<typename ChildOp>
    static receiver make_receiver_for(ChildOp* child) noexcept { /* … */ }

    // get_env/set_value/set_error/set_stopped - forward to calls to ParentOp with Tag{}.

    ParentOp* parent;
  };

  using child_op_t = connect_result_t<ChildSender, receiver>;

protected:
  child_operation_state(ChildSender&& sndr) {
    ::new (&storage) child_op_t(execution::connect(std::forward<ChildSender>(sndr),
                                                   receiver{static_cast<ParentOp*>(this)}));
  }

  ~child_operation_state() {
    reinterpret_cast<child_op_t*>(&storage)->~child_op_t();
  }

private:
  alignas(child_op_t) unsigned char storage[sizeof(child_op_t)];
};
```

Implementation Helper (not proposed)

```
template<class ParentOp, class Tag, class Env, class ChildSender>
class child_operation_state {
  struct receiver {
    template<typename ChildOp>
    static receiver make_receiver_for(ChildOp* child) noexcept { /* … */ }

    // get_env/set_value/set_error/set_stopped - forward to calls to ParentOp with Tag{}.

    ParentOp* parent;
  };

  using child_op_t = connect_result_t<ChildSender, receiver>;

protected:
  child_operation_state(ChildSender&& sndr) {
    ::new (&storage) child_op_t(execution::connect(std::forward<ChildSender>(sndr),
                                                   receiver{static_cast<ParentOp*>(this)}));
  }

  ~child_operation_state() {
    reinterpret_cast<child_op_t*>(&storage)->~child_op_t();
  }

private:
  alignas(child_op_t) unsigned char storage[sizeof(child_op_t)];
};
```

Implementation Helper (not proposed)

```
template<class ParentOp, class Tag, class Env, class ChildSender>
class child_operation_state {
  struct receiver {
    template<typename ChildOp>
    static receiver make_receiver_for(ChildOp* child) noexcept { /* … */ }

    // get_env/set_value/set_error/set_stopped - forward to calls to ParentOp with Tag{}.

    ParentOp* parent;
  };

  using child_op_t = connect_result_t<ChildSender, receiver>;

protected:
  child_operation_state(ChildSender&& sndr) {
    ::new (&storage) child_op_t(execution::connect(std::forward<ChildSender>(sndr),
                                                   receiver{static_cast<ParentOp*>(this)}));
  }

  ~child_operation_state() {
    reinterpret_cast<child_op_t*>(&storage)->~child_op_t();
  }

private:
  alignas(child_op_t) unsigned char storage[sizeof(child_op_t)];
};
```

Implementation Helper (not proposed)

```
template<class ParentOp, class Tag, class Env, class ChildSender>
class child_operation_state {
  struct receiver {
    template<typename ChildOp>
    static receiver make_receiver_for(ChildOp* child) noexcept { /* … */ }

    // get_env/set_value/set_error/set_stopped - forward to calls to ParentOp with Tag{}.

    ParentOp* parent;
  };

  using child_op_t = connect_result_t<ChildSender, receiver>;

protected:
  child_operation_state(ChildSender&& sndr) {
    ::new (&storage) child_op_t(execution::connect(std::forward<ChildSender>(sndr),
                                                   receiver{static_cast<ParentOp*>(this)}));
  }

  ~child_operation_state() {
    reinterpret_cast<child_op_t*>(&storage)->~child_op_t();
  }

private:
  alignas(child_op_t) unsigned char storage[sizeof(child_op_t)];
};
```

Implementation Helper (not proposed)

```
template<class ParentOp, class Tag, class Env, class ChildSender>
class child_operation_state {
  struct receiver {
    template<typename ChildOp>
    static receiver make_receiver_for(ChildOp* child) noexcept { /* … */ }

    // get_env/set_value/set_error/set_stopped - forward to calls to ParentOp with Tag{}.

    ParentOp* parent;
  };

  using child_op_t = connect_result_t<ChildSender, receiver>;

protected:
  child_operation_state(ChildSender&& sndr) {
    ::new (&storage) child_op_t(execution::connect(std::forward<ChildSender>(sndr),
                                                   receiver{static_cast<ParentOp*>(this)}));
  }

  ~child_operation_state() {
    reinterpret_cast<child_op_t*>(&storage)->~child_op_t();
  }

private:
  alignas(child_op_t) unsigned char storage[sizeof(child_op_t)];
};
```

Implementation Helper - make_receiver_for()

```
template<typename ChildOp>
static receiver make_receiver_for(ChildOp* child)noexcept {
  // Cast from address of object to address of storage backing that object.
  auto* storage = reinterpret_cast<storage_t*>(child);

  // Cast from address of ‘storage’ member to ‘child_operation_state’ parent object
  // First member of standard layout class is “pointer interconvertible” with parent object
  auto* self = reinterpret_cast<child_operation_state*>(storage);

  // Cast from child_operation_state to derived class inheriting from this class.
  auto* parent = static_cast<ParentOp*>(self);

  // Initialise receiver with pointer to parent.
  return receiver{parent};
}
```

Usage

```
template<class ParentRcvr,  class ChildSndr>
struct my_parent_op
  : inlinable_operation_state<my_parent_op<ParentRcvr, ChildSndr>, ParentRcvr>
  , child_operation_state<my_parent_op<ParentRcvr, ChildSndr>,
                          first_child_tag,
                          env_of_t<ParentRcvr>,
                          ChildSndr> {
  using child_t = child_operation_state<my_parent_op, first_child_tag, env_of_t<ParentRcvr>, ChildSndr>;

  my_parent_op(ParentRcvr rcvr, ChildSndr&& child,  int arg)
    : inlinable_operation_state<my_parent_op, ParentRcvr>(std::move(rcvr))
    , child_t(std::forward<ChildSndr>(child))
    , other_state(arg)
  {}

  void start() {
    child_t::start_child();
  }

  void complete(first_child_tag, set_value_t,  auto&&... datums)  noexcept {  /*...*/  }

  auto get_env(first_child_tag)  noexcept {
    return execution::get_env(this->get_receiver());
  }

private:
  int other_state;
};
```

Usage

```
template<class ParentRcvr, class ChildSndr>
struct my_parent_op
  : inlinable_operation_state<my_parent_op<ParentRcvr, ChildSndr>, ParentRcvr>
  , child_operation_state<my_parent_op<ParentRcvr, ChildSndr>,
                          first_child_tag,
                          env_of_t<ParentRcvr>,
                          ChildSndr> {
  using child_t = child_operation_state<my_parent_op, first_child_tag, env_of_t<ParentRcvr>, ChildSndr>;

  my_parent_op(ParentRcvr rcvr, ChildSndr&& child,  int arg)
    : inlinable_operation_state<my_parent_op, ParentRcvr>(std::move(rcvr))
    , child_t(std::forward<ChildSndr>(child))
    , other_state(arg)
  {}

  void start() {
    child_t::start_child();
  }

  void complete(first_child_tag, set_value_t,  auto&&... datums) noexcept { /*...*/ }

  auto get_env(first_child_tag)  noexcept {
    return execution::get_env(this->get_receiver());
  }

private:
  int other_state;
};
```

Usage

```
template<class ParentRcvr, class ChildSndr>
struct my_parent_op
  : inlinable_operation_state<my_parent_op<ParentRcvr, ChildSndr>, ParentRcvr>
  , child_operation_state<my_parent_op<ParentRcvr, ChildSndr>,
                          first_child_tag,
                          env_of_t<ParentRcvr>,
                          ChildSndr> {
  using child_t = child_operation_state<my_parent_op, first_child_tag, env_of_t<ParentRcvr>, ChildSndr>;

  my_parent_op(ParentRcvr rcvr, ChildSndr&& child, int arg)
    : inlinable_operation_state<my_parent_op, ParentRcvr>(std::move(rcvr))
    , child_t(std::forward<ChildSndr>(child))
    , other_state(arg)
  {}

  void start() {
    child_t::start_child();
  }

  void complete(first_child_tag, set_value_t, auto&&... datums) noexcept { /*...*/ }

  auto get_env(first_child_tag) noexcept {
    return execution::get_env(this->get_receiver());
  }

private:
  int other_state;
};
```

Usage

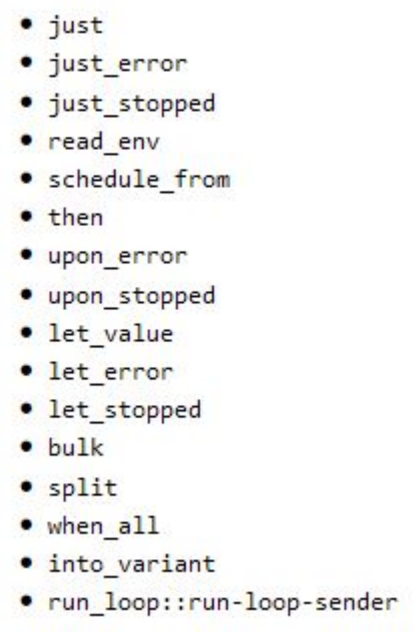
```
template<class ParentRcvr, class ChildSndr>
struct my_parent_op
  : inlinable_operation_state<my_parent_op<ParentRcvr, ChildSndr>, ParentRcvr>
  , child_operation_state<my_parent_op<ParentRcvr, ChildSndr>,
                          first_child_tag,
                          env_of_t<ParentRcvr>,
                          ChildSndr> {
  using child_t = child_operation_state<my_parent_op, first_child_tag, env_of_t<ParentRcvr>, ChildSndr>;

  my_parent_op(ParentRcvr rcvr, ChildSndr&& child, int arg)
    : inlinable_operation_state<my_parent_op, ParentRcvr>(std::move(rcvr))
    , child_t(std::forward<ChildSndr>(child))
    , other_state(arg)
  {}

  void start() {
    child_t::start_child();
  }

  void complete(first_child_tag, set_value_t, auto&&... datums) noexcept { /*...*/ }

  auto get_env(first_child_tag) noexcept {
    return execution::get_env(this->get_receiver());
  }

private:
  int other_state;
};
```

Usage

```
template<class ParentRcvr, class ChildSndr>
struct my_parent_op
  : inlinable_operation_state<my_parent_op<ParentRcvr, ChildSndr>, ParentRcvr>
  , child_operation_state<my_parent_op<ParentRcvr, ChildSndr>,
                          first_child_tag,
                          env_of_t<ParentRcvr>,
                          ChildSndr> {
  using child_t = child_operation_state<my_parent_op, first_child_tag, env_of_t<ParentRcvr>, ChildSndr>;

  my_parent_op(ParentRcvr rcvr, ChildSndr&& child,  int arg)
    : inlinable_operation_state<my_parent_op, ParentRcvr>(std::move(rcvr))
    , child_t(std::forward<ChildSndr>(child))
    , other_state(arg)
  {}

  void start() {
    child_t::start_child();
  }

  void complete(first_child_tag, set_value_t,  auto&&... datums) noexcept { /*...*/ }

  auto get_env(first_child_tag) noexcept {
    return execution::get_env(this->get_receiver());
  }

private:
  int other_state;
};
```

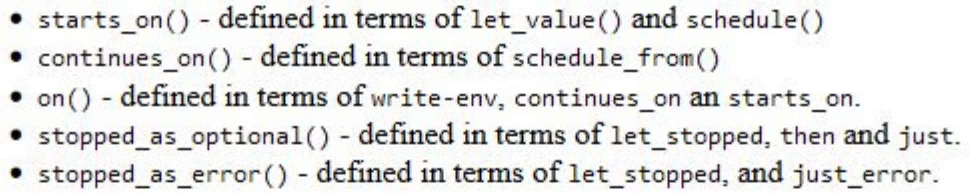
Applying to existing algorithms

Need Updating



- just
- just_error
- just_stopped
- read_env
- schedule_from
- then
- upon_error
- upon_stopped
- let_value
- let_error
- let_stopped
- bulk
- split
- when_all
- into_variant
- run_loop::run-loop-sender

Don’t need updating



- starts_on() - defined in terms of let_value() and schedule()
- continues_on() - defined in terms of schedule_from()
- on() - defined in terms of write-env, continues_on an starts_on.
- stopped_as_optional() - defined in terms of let_stopped, then and just.
- stopped_as_error() - defined in terms of let_stopped, and just_error.

All defined in terms of exposition-only ‘basic-operation’.

This needs refactoring to allow child ops to be base-classes.

Can’t we do this later?

- May be unable to retrospectively apply this protocol to existing sender algorithms in future standard
  - Changes will affect layout of operation-state types -> could be an ABI break

- Algorithms that don’t implement this protocol will inhibit optimisations because they will fall back to storing receivers / walking chain of pointers
  - Ideally we want the most common standard algorithms to support this

- Defining the protocol up-front allows other sender implementers to also opt-in to the protocol.

Implementation Experience

- Implemented in private experimental code-base
  - In process of open-sourcing

- Not yet ported to stdexec/beman projects