Session Types in C++:

A Programmer's Journey

MIODRAG MISHA DJUKIC

About me

Faculty of Technical Sciences, University of Novi Sad, Serbia

Teaching… a lot…

Background in compilers and embedded systems…

Motive?

Motive:
Can it be done in C++?

Session type

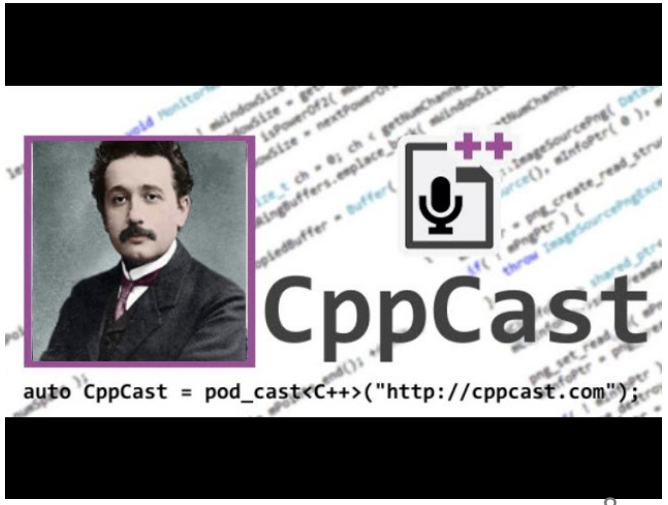
Session type ?

"A normal adult programmer never gives a thought about types. That is something which he has tough of and learned as a child. I, on the contrary, developed so slowly that I did not begin to wonder about types until I was an adult.“

– Albert Einstein

"A normal adult programmer never gives a thought about types. That is something which he has tough of and learned as a child. I, on the contrary, developed so slowly that I did not begin to wonder about types until I was an adult.“

– Albert Einstein, In an interview on Cppcast, Colorized



Definition of “type”?

Definition of “type”?

Step back: Definition of “set”?

Definition of “type”?

Step back: Definition of “set”?

Definition of “type”?

Step back: Definition of “set”?

Sets are primitive – no definition.

Definition of “type”?

Step back: Definition of “set”?

Sets are primitive – no definition.

But we have Type theory…

…a way to overcome Russell's paradox…

…“type” is primitive…

…

Maybe informally:
So, what is a type?

Maybe informally:
So, what is a type?

Depends on who you ask.

Maybe informally:
So, what is a type?

Depends on who you ask.

So called “dependent type”?

What are types used for in programming?

What are types used for in programming?

- Similar to money and its roles:
  - Medium of exchange
  - Store of value
  - Unit of account

- In many cases a single currency plays all three roles – but not always.

What are types used for in programming?

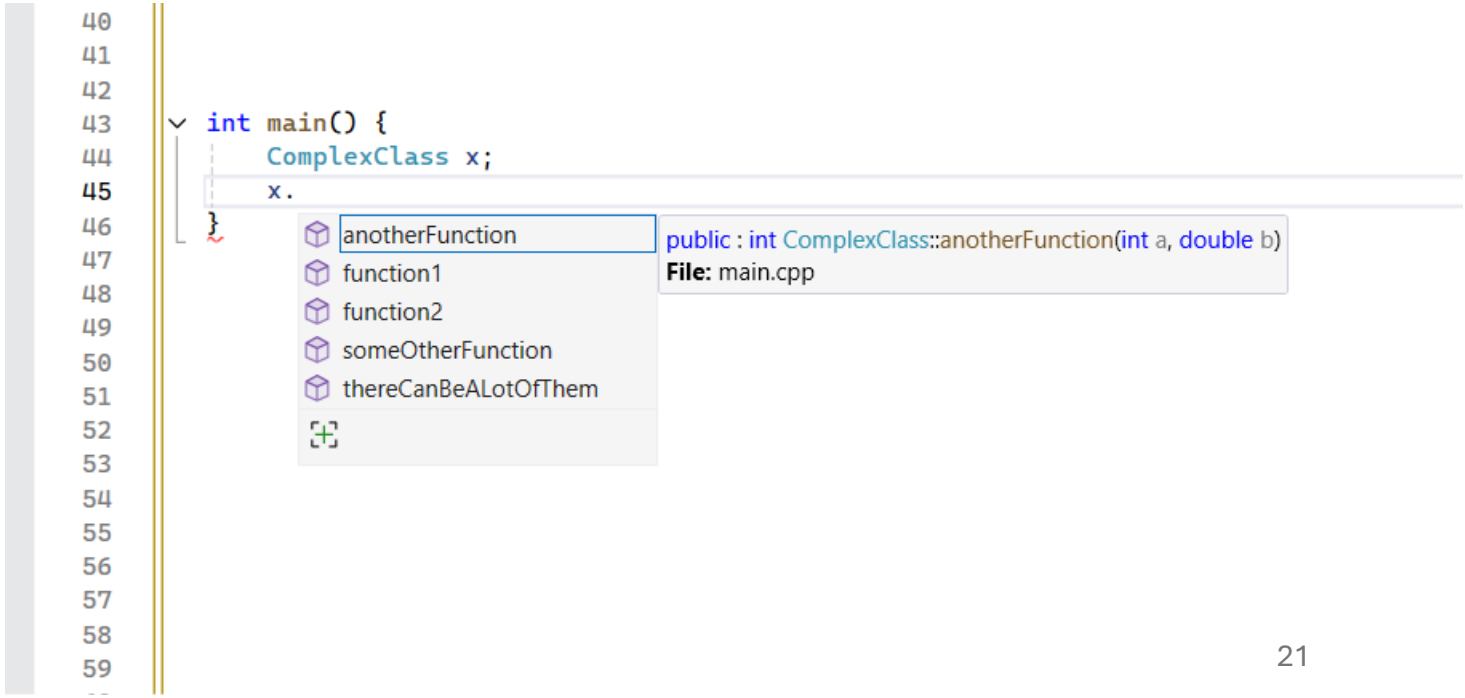
- Types are used for:
  - Abstraction
  - Documentation
  - Efficiency
  - Expressivity
  - Detecting errors
  - Safety

What are types used for in programming?

- Types are used for:
  - Abstraction
    - Knowing a type of something is enough to “work” with it. (We do not need to know all the details.)

What are types used for in programming?

- Types are used for:
  - Abstraction
  - Documentation
    - Type informs us how we should interact with something.



What are types used for in programming?

- Types are used for:
  - Abstraction
  - Documentation
  - Efficiency
    - Carefully chosen type can lead to more efficient code.

What are types used for in programming?

- Types are used for:
  - Abstraction
  - Documentation
  - Efficiency
  - Expressivity
    - Meaning is encoded in both operation and operands' type.

What are types used for in programming?

- Types are used for:
  - Abstraction
  - Documentation
  - Efficiency
  - Expressivity
  - Detecting errors
    - Doing something (by accident) that does not “fit” the type will be detected as error.

What are types used for in programming?

- Types are used for:
  - Abstraction
  - Documentation
  - Efficiency
  - Expressivity
  - Detecting errors
  - Safety

What are types used for in programming?

- Types are used for:
  - Abstraction
  - Documentation
  - Efficiency
  - Expressivity
  - Detecting errors
  - Safety
    - Varying explanations.
    - ~A guarantee that there are no certain kinds of errors.
    - ~Being unable to do a “wrong thing”.

What are types used for in programming?

- Types are used for:
  - Abstraction
  - Documentation
  - Efficiency
  - Expressivity
  - Detecting errors
  - Safety

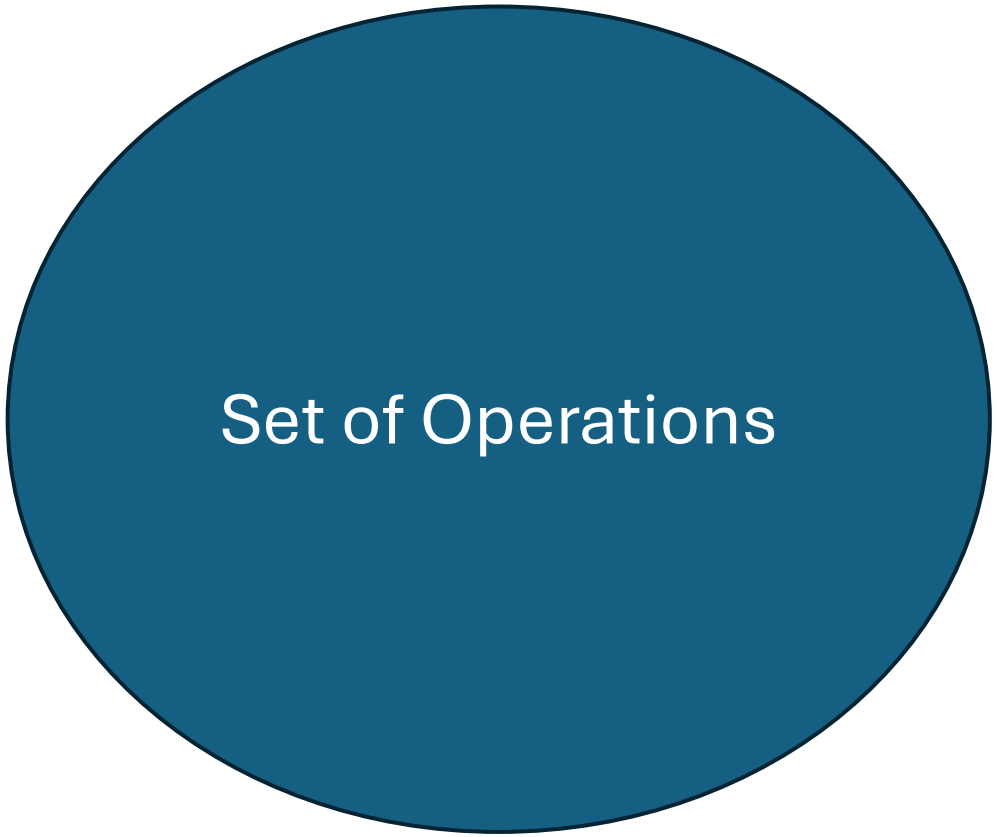
We do not use just types (from our language) to achieve all these things. Textual documentation, compiler (and other kinds) of optimizations, static analyzers, (contracts?)…

In C++, we can settle with this explanation:

"A type defines a set of possible values and a set of operations (for an object).“

– Bjarne Stroustrup, *Programming Principles and Practices Using C++*

But that is not what “type” in “Session type” means.



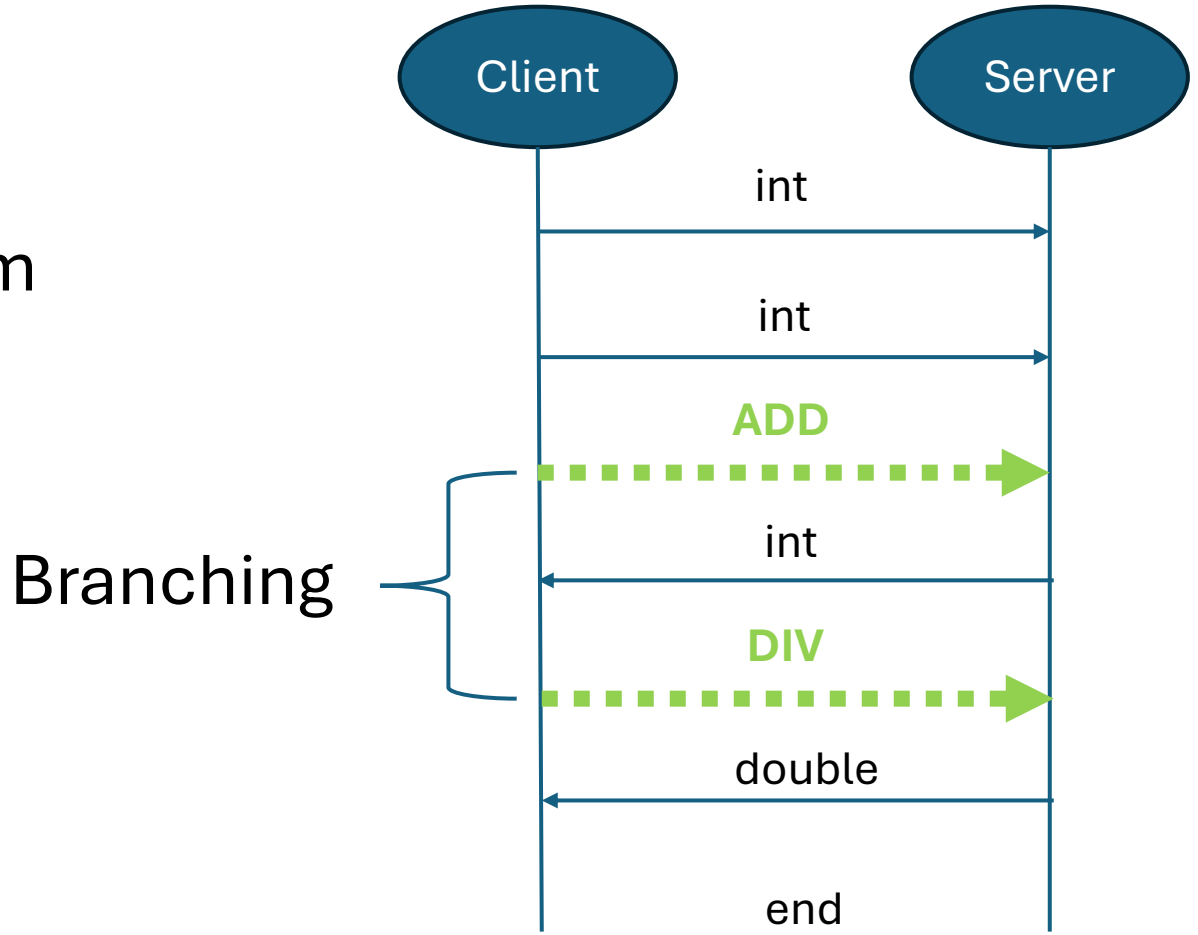

?
Session type

Session?

- Interaction of two or more entities.
- It has a beginning and (usually) an end.
- In between, a sequence of interactions is happening.

Session?

- Can we describe a valid sequence of interactions?
- Interaction diagram



Session?

- Can we describe a valid sequence of interactions?

- Or formula

?int; ?int; &(ADD: !int, DIV: !double); end

Session?

- Can we describe a valid sequence of interactions?

- Or formula

?int; ?int; &(ADD: !int, DIV: !double); end

dual formula !int; !int; ⊕(ADD: ?int, DIV: ?double); end

Session types

- Formal description of “message” (whatever that means exactly) sequences… so that it can be check for correctness.

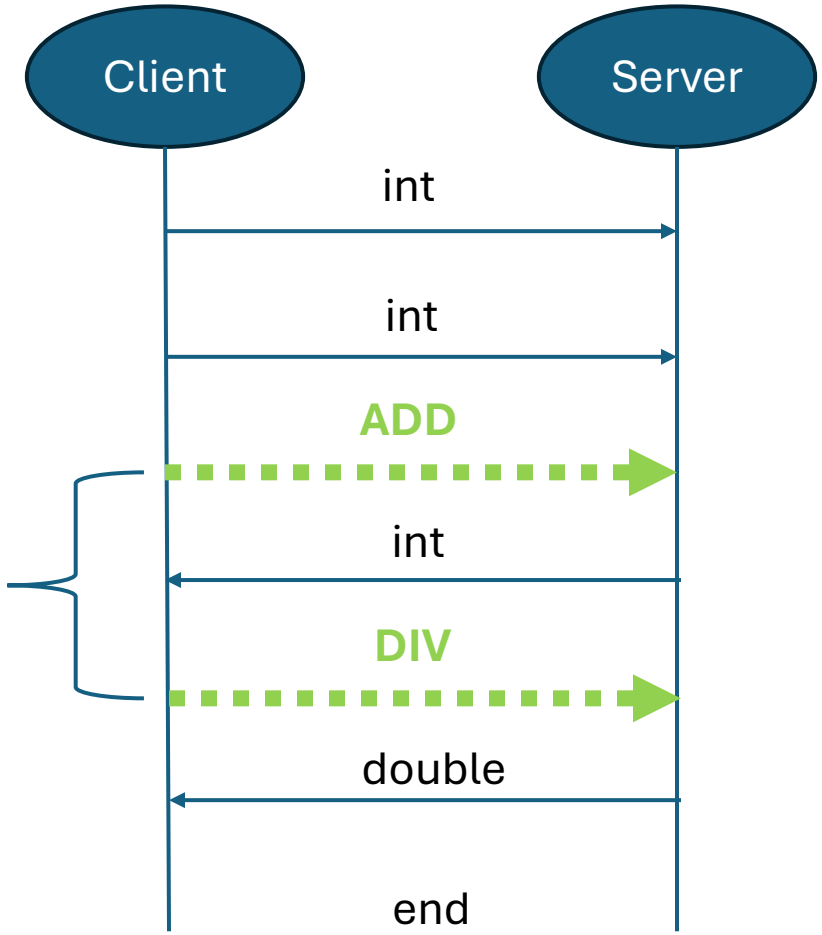
- Correctness can mean different things: valid message format and message order, absence of deadlocks, eventual termination…

Session types

- For just two processes: Binary Session Types
- Multiparty Session Types (MPST) for more than two… but we'll focus just on Binary today.

Session types

- They describe a behavior*:
  - First receive int (**no other operation is allowed!**)
  - Then, again receive int
  - …



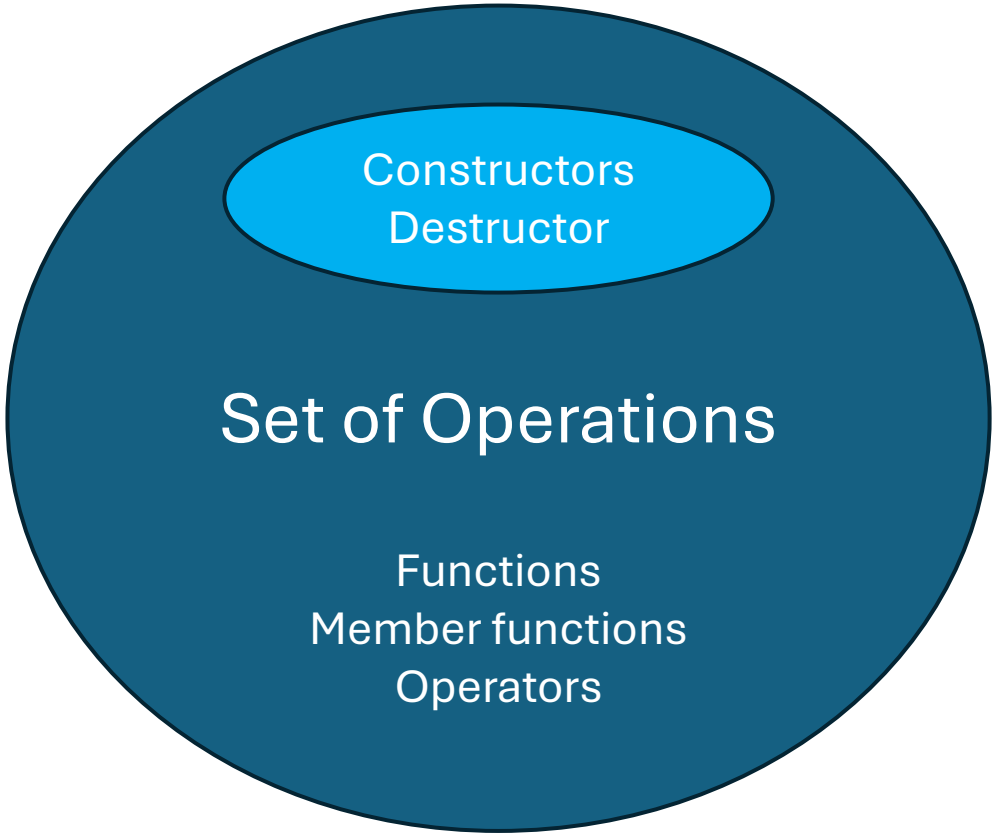
?int; ?int; &(ADD: !int, DIV: !double); end

*Name “Behavioral types” is also used. See “Behavioral types in programming languages”, D. Ancona et al., 2016.

"A type defines a set of possible values and a set of operations (for an object).“

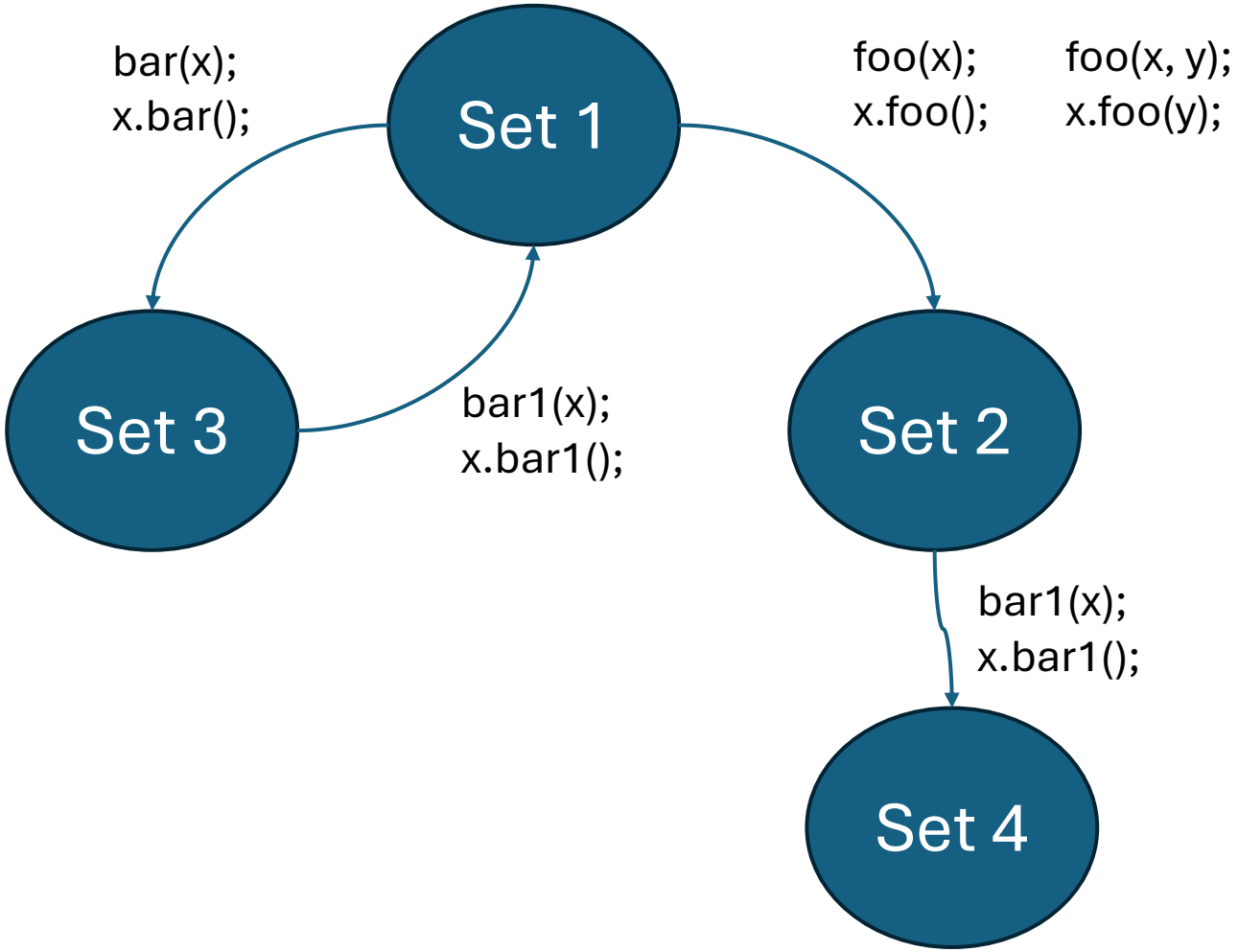
- Bjarne Stroustrup, *Programming Principles and Practices Using C++*

Again, this is what we have in C++:

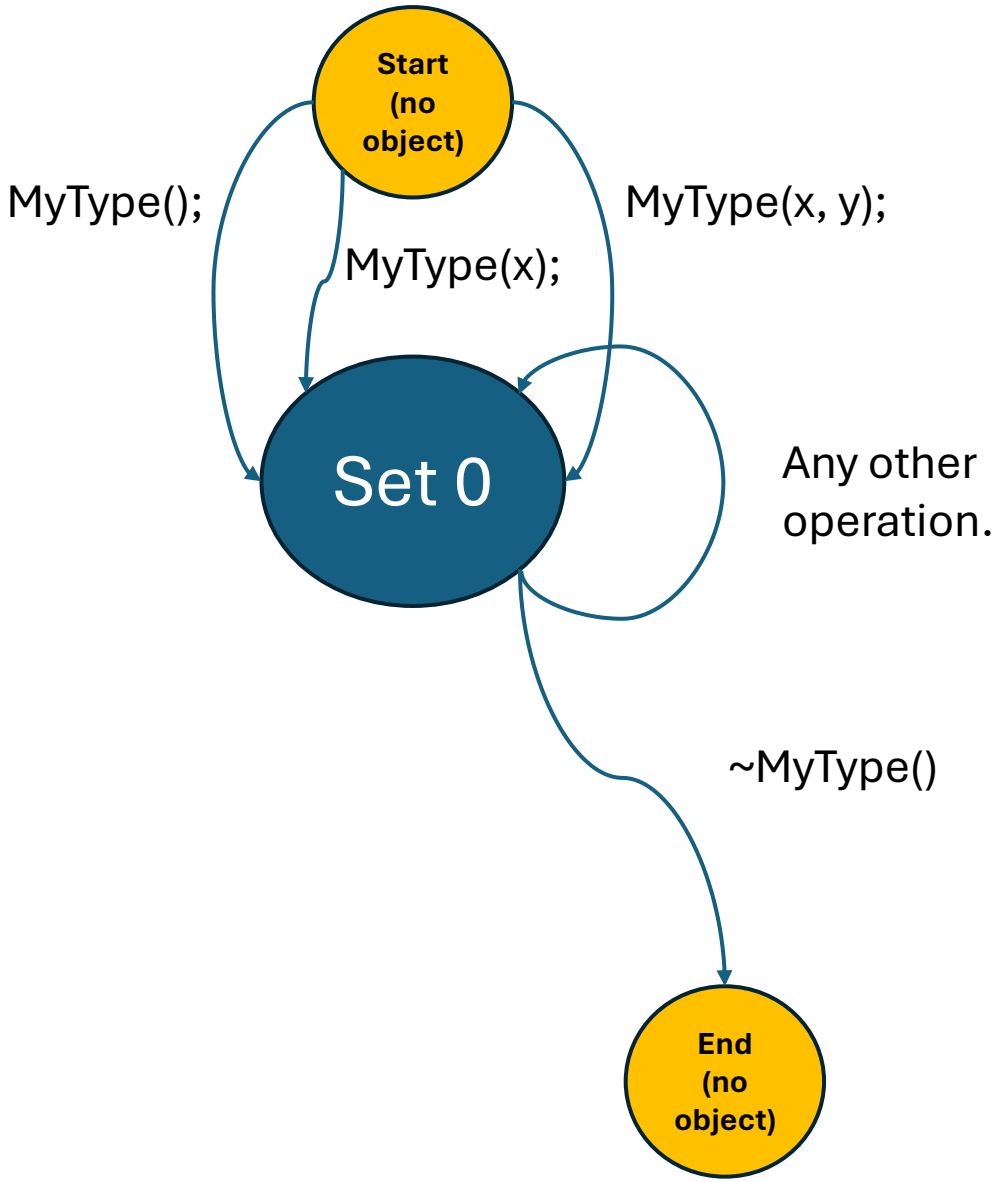


So… like “Bjarne’s types” are changing after every operation…

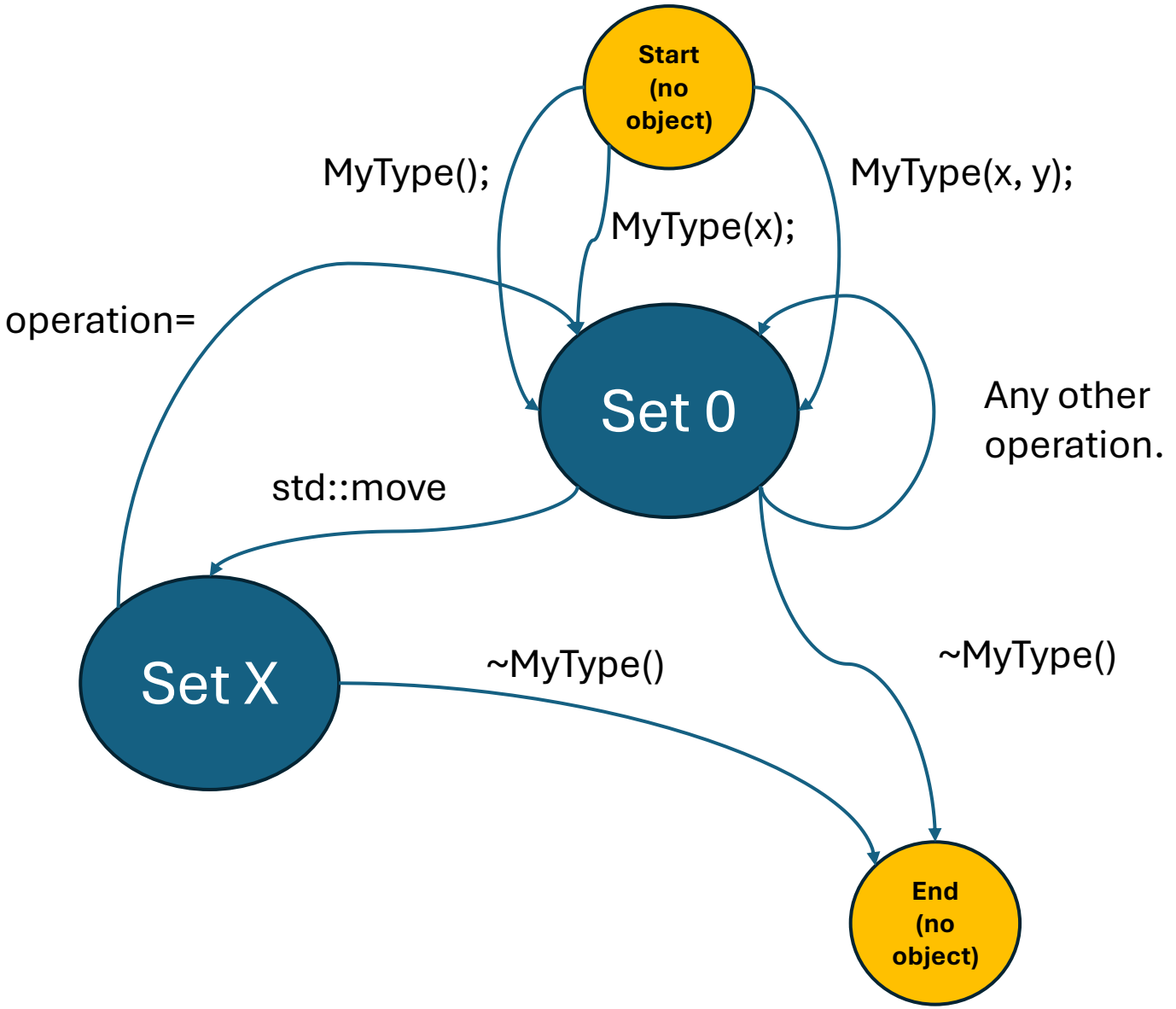
...like this:


bar(x);
x.bar();
Set 1
foo(x);
x.foo();
foo(x, y);
x.foo(y);
Set 3
bar1(x);
x.bar1();
Set 2
bar1(x);
x.bar1();
Set 4

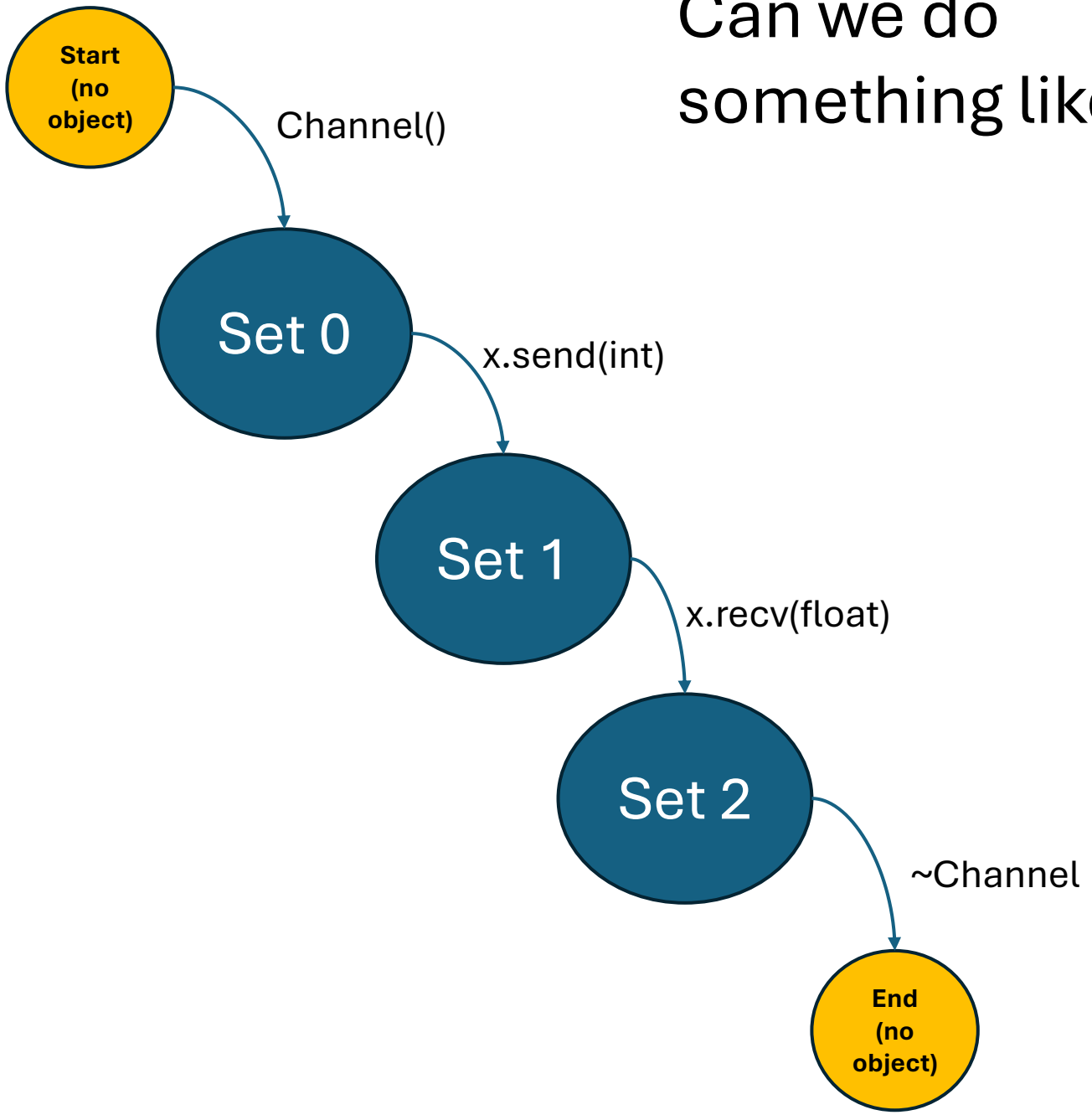
What we have currently:



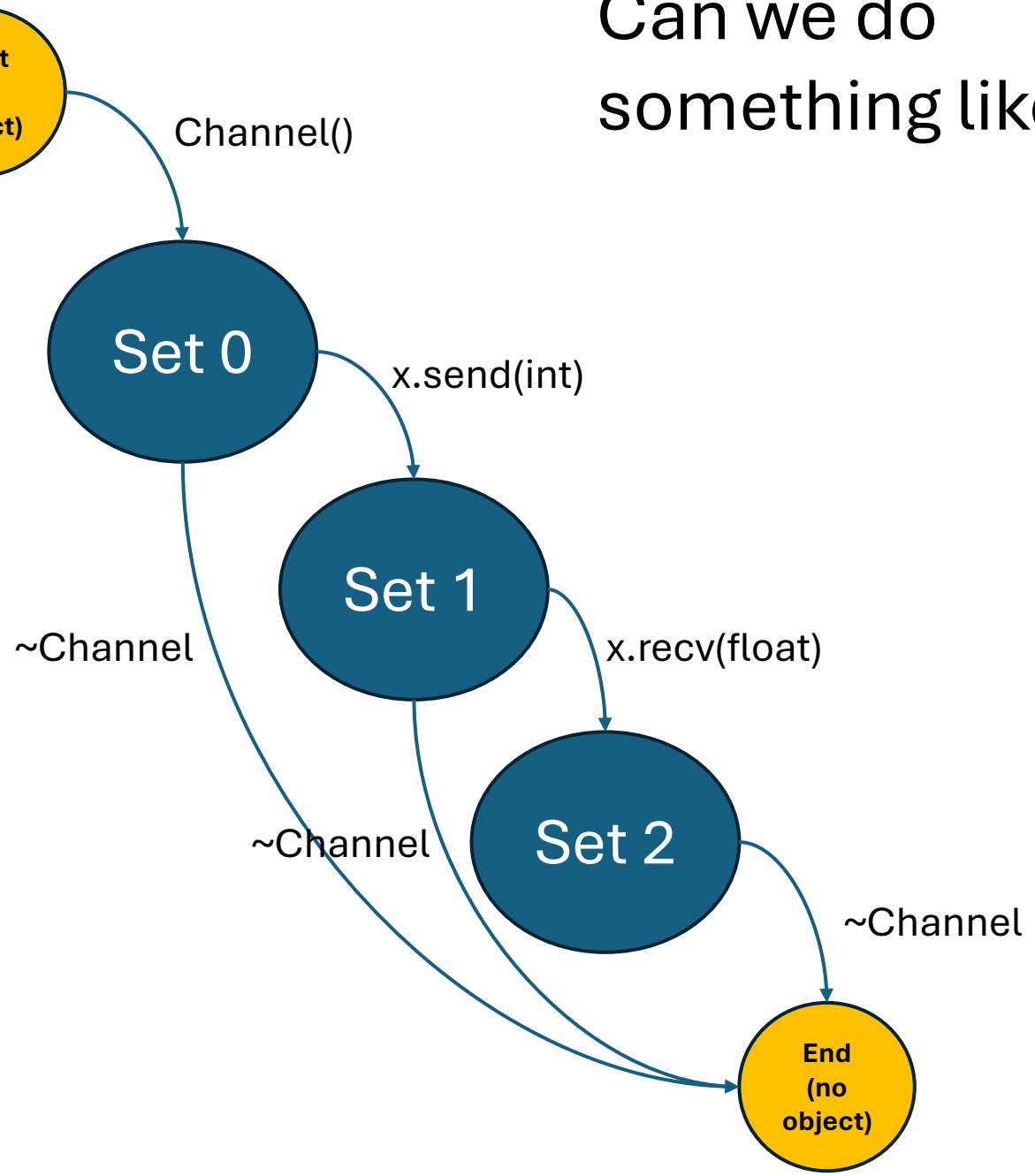
What we have currently:



Can we do something like this?



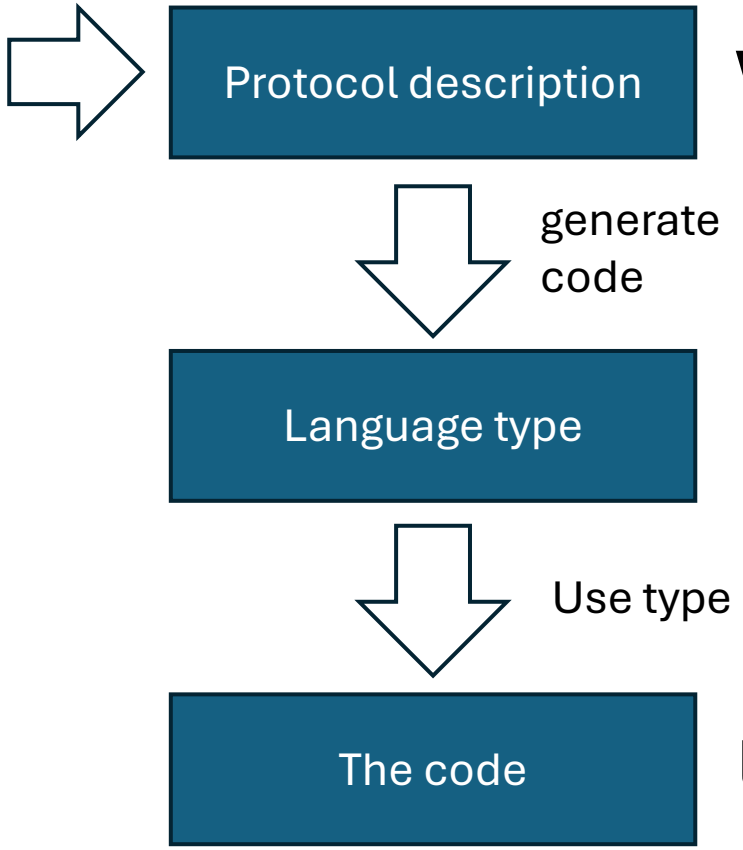
Can we do something like this?



How can we practically use Session types?

Three approaches

1)



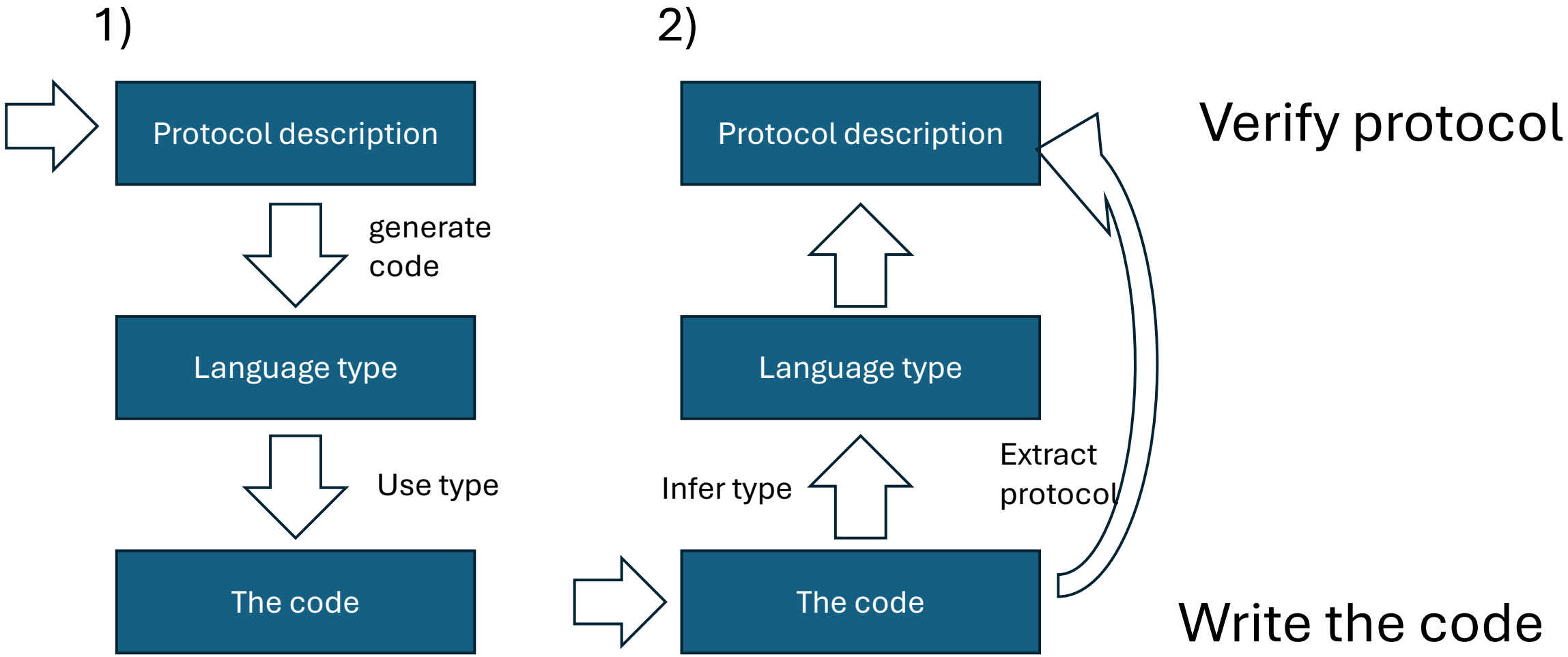
Write and verify protocol

Use according to type

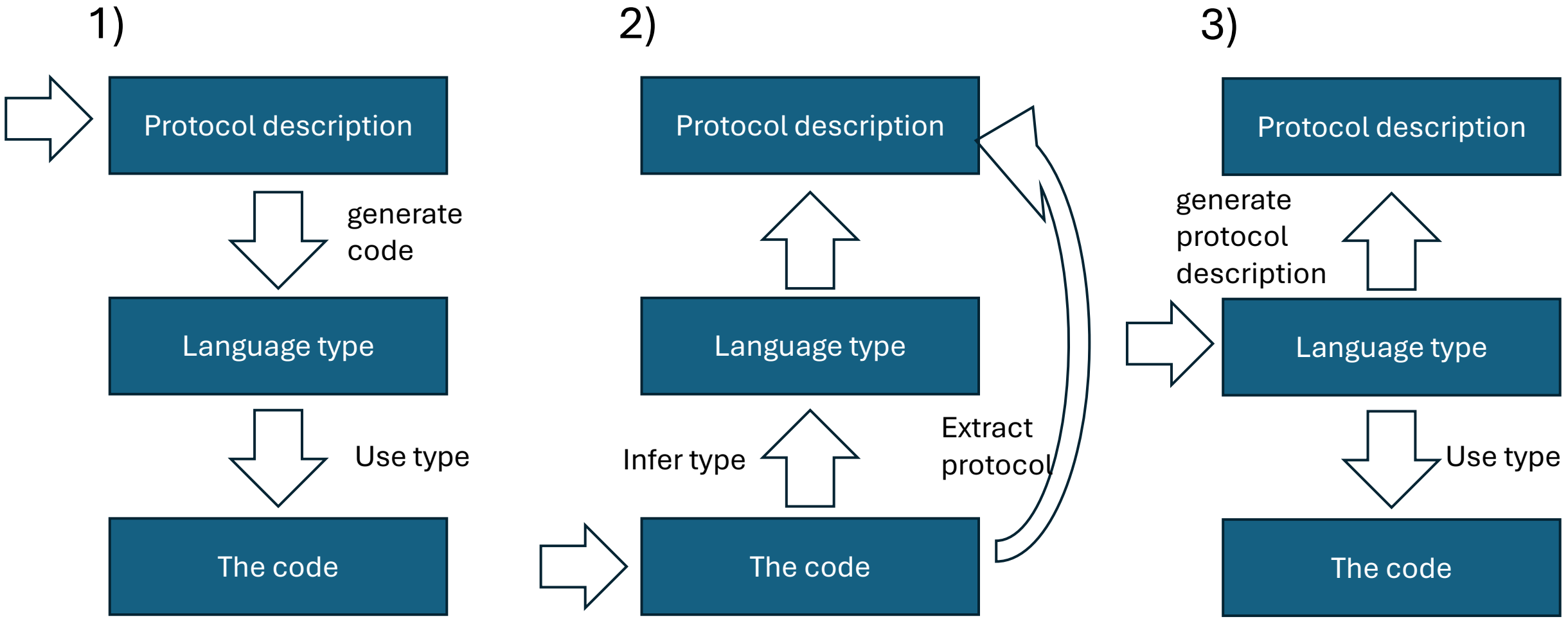
“Scribble” – for writing and verifying protocol

```
module scribble.example.Purchasing;

type <xsd> "QuoteRequest" from "Purchasing.xsd" as QuoteRequest;
...

global protocol BuyGoods (role Buyer, role Seller) {
      quote(QuoteRequest) from Buyer to Seller;

      choice at Seller {
            quote(Quote) from Seller to Buyer;
            buy(Order) from Buyer to Seller;
            buy(OrderAck) from Seller to Buyer;
      } or {
            quote(OutOfStock) from Seller to Buyer;
      }
}
```

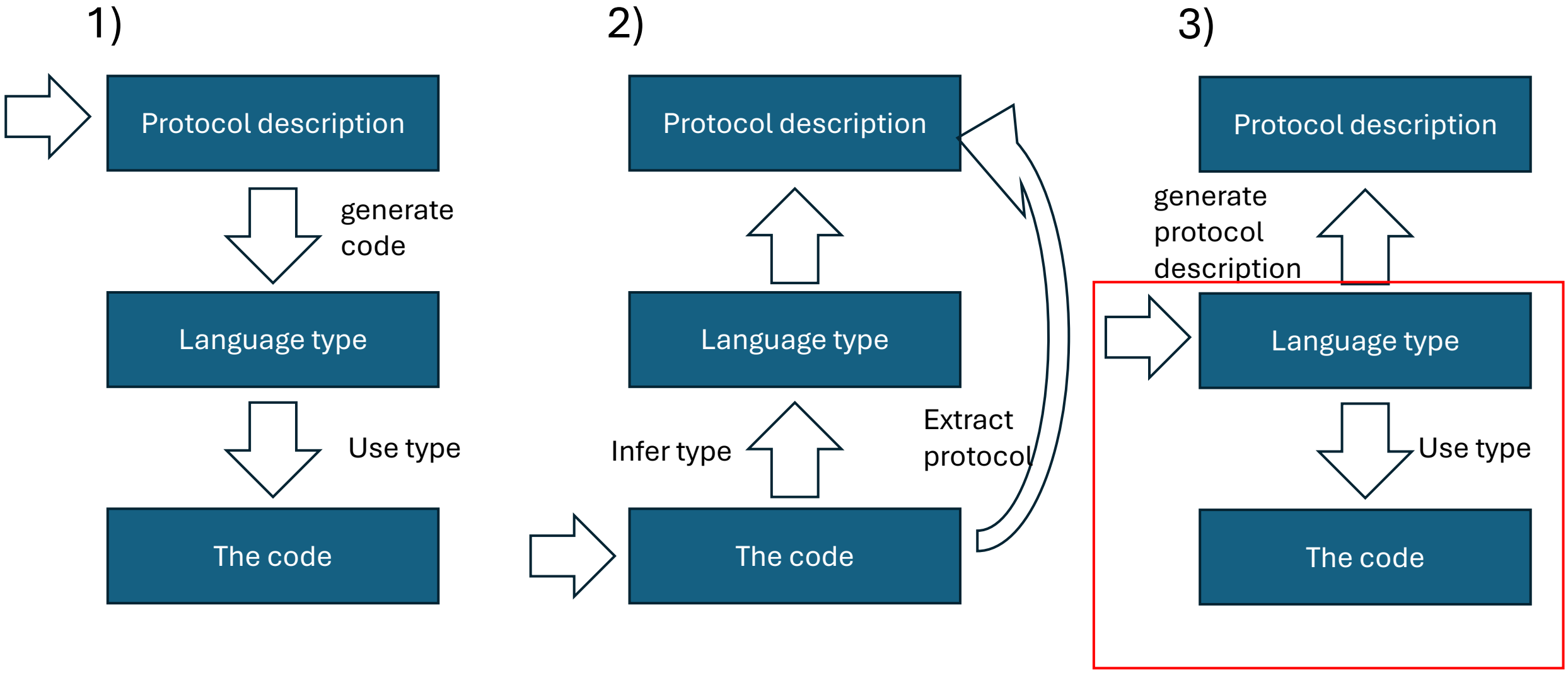
Three approaches



Three approaches



Three approaches



File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search ProjectX Sign in

Debug x64 Local Windows Debugger GitHub Copilot

Solution Explorer

Search Solution Explorer (Ct

Solution 'ProjectX' (1 of 1 proj

ProjectX

References

External Dependencies

Header Files

Resource Files

Source Files

main.cpp

main.cpp

ProjectX (Global Scope)

1

100 % No issues found Ln: 1 Ch: 1 TABS CRLF

Error List

Entire Solution 0 Errors 0 Warnings 0 Messages Build + IntelliSense Search Error List

Code Description Project File Line Details

Solution Explorer Git Changes

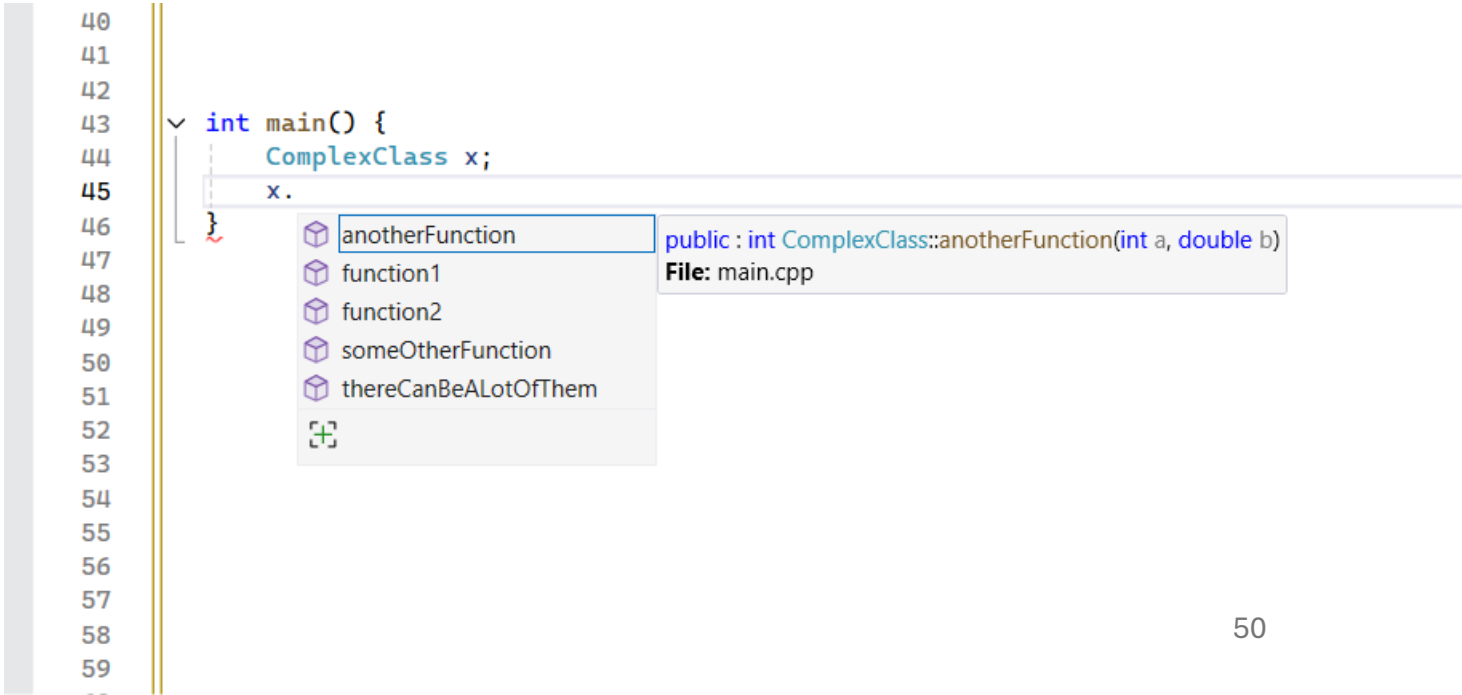
Error List Output

Ready Add to Source Control Select Repository

Again, we have different approaches

- Functional style -> always new objects
  - Monads, continuation passing…

- But this is so cool:



Communication?

- Abstraction: channels… or actors

Two main cases in practice

- concurrent queues

- networking sockets

Communication?

- Abstraction: channels… or actors

Two main cases in practice

- concurrent queues

Bounded/Unbounded, Sync/Async, MPMC, MPSC, SPMC, SPSC, Lockfree, Blocking…

- networking sockets

Communication?

- Abstraction: channels… or actors

Two main cases in practice

- concurrent queues

Bounded/Unbounded, Sync/Async, MPMC, MPSC, SPMC, SPSC, Lockfree, Blocking…

- networking sockets

Communication?

- Abstraction: channels… or actors

Two main cases in practice

- concurrent queues

- networking sockets

io_context, resolver, acceptor, end_point,
asio::buffer, async_read/write, serialization…

Communication?

- Abstraction: channels… or actors

Two main cases in practice

- concurrent queues

- networking sockets

io_context, resolver, acceptor, end_point,
asio::buffer, async_read/write, serialization…

Communication?

- Abstraction: channels… or actors

Two main cases in practice

- concurrent queues

Ultra simple queue
Send/Receive

- networking sockets

Communication?

- Abstraction: channels… or actors

- Seriously: channels or actors?

- 1) One channel full duplex
- 2) Two channels, one for each “actor” (endpoint)

Our simplified communication

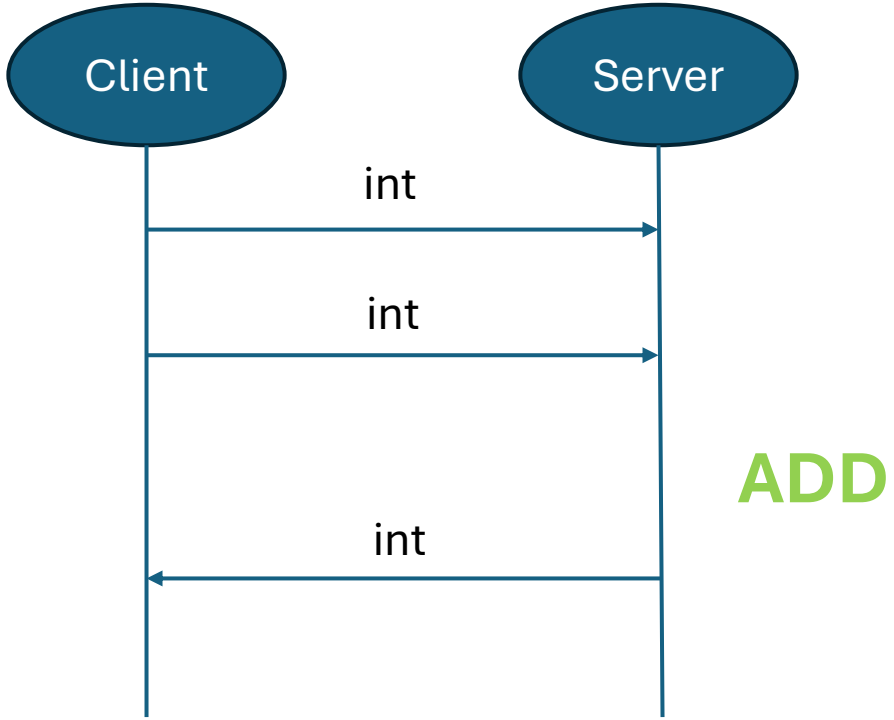
```
struct Comm {
   Comm(SQ& forSending, SQ& forReceiving);
   ...

   template<typename T>
   void send(T x) { ... }

   template<typename T>
   T recv() { ... }

};

Comm(Q1, Q2); // One end-point
Comm(Q2, Q1); // The other end-point
```

Ultra simple queue Send/Receive

Two channels/queues

```
void serverFunc(Comm chan) {
      auto v1 = chan.recv<int>();
      auto v2 = chan.recv<int>();
      chan.send(v1 + v2);
      chan.close();
}
```

?int; ?int; !int; end

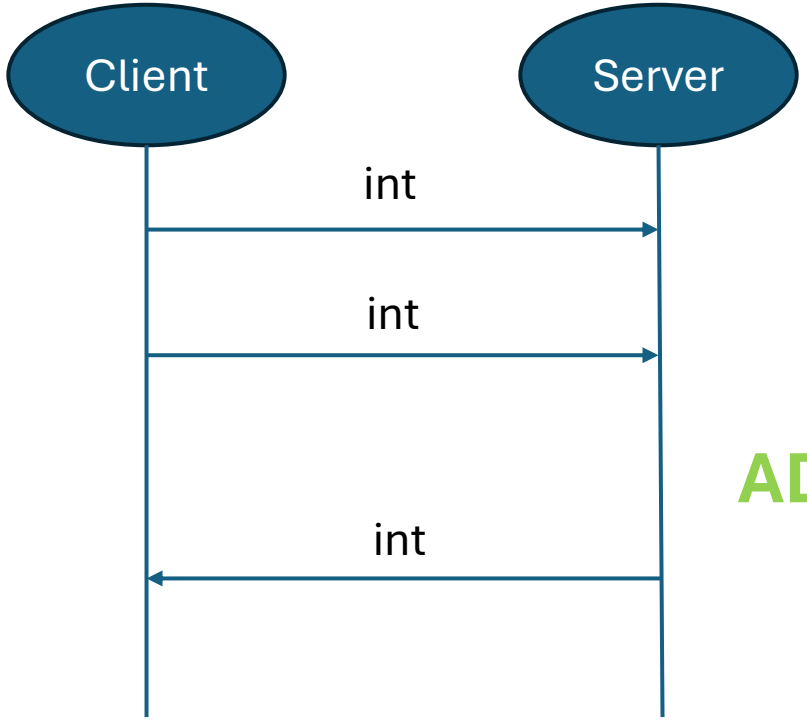



```
void clientFunc(Comm chan) {
    cout << "First num: ";
    int x;
    cin >> x;
    chan.send(x);
    cout << "Second num: ";
    cin >> x;
    chan.send(x);
    auto r = chan.recv<int>();
    cout << "Result: " << r << endl;
    chan.close();
}
```

```
void serverFunc(Comm chan) {
      auto v1 = chan.recv<int>();
      auto v2 = chan.recv<int>();
      chan.send(v1 + v2);
      chan.close();
}
```

?int; ?int; !int; end

All the things that can go wrong here?

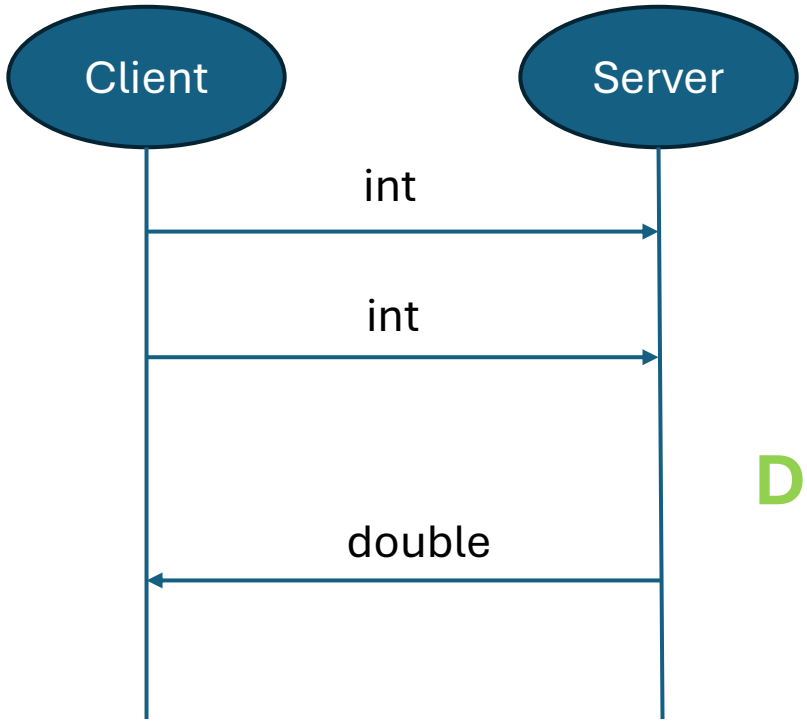


ADD

```
void clientFunc(Comm chan) {
    cout << "First num: ";
    int x;
    cin >> x;
    chan.send(x);
    cout << "Second num: ";
    cin >> x;
    chan.send(x);
    auto r = chan.recv<int>();
    cout << "Result: " << r << endl;
    chan.close();
}
```

```
void serverFunc(Comm chan) {
      auto v1 = chan.recv<int>();
      auto v2 = chan.recv<int>();
      chan.send((double)v1 / v2);
      chan.close();
}
```

?int; ?int; !double; end

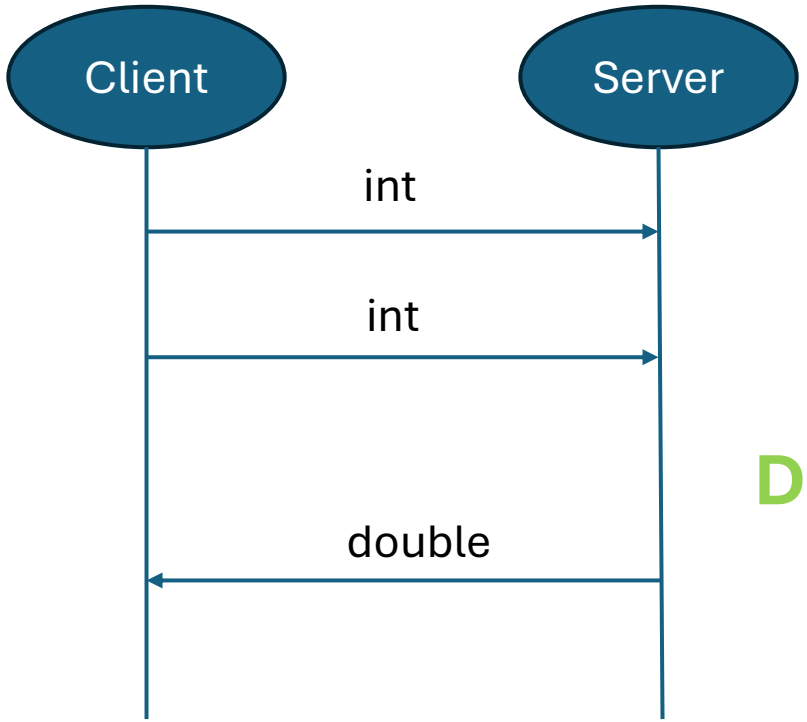


DIV

```
void clientFunc(Comm chan) {
    cout << "First num: ";
    int x;
    cin >> x;
    chan.send(x);
    cout << "Second num: ";
    cin >> x;
    chan.send(x);
    auto r = chan.recv<int>();
    cout << "Result: " << r << endl;
    chan.close();
}
```

```
void serverFunc(Comm chan) {
      auto v1 = chan.recv<int>();
      auto v2 = chan.recv<int>();
      chan.send((double)v1 / v2);
      chan.close();
}
```

?int; ?int; !double; end



DIV

```
void clientFunc(Comm chan) {
    cout << "First num: ";
    int x;
    cin >> x;
    chan.send(x);
    cout << "Second num: ";
    cin >> x;
    chan.send(x);
    auto r = chan.recv<int>();
    cout << "Result: " << r << endl;
    chan.close();
}
```

```
struct Quit {};

template<typename A, typename T>
struct Send {};

template<typename A, typename T>
struct Recv {};

template<typename L, typename R>
struct Offer {};

template<typename L, typename R>
struct Choose {};
```

```
using Server = Recv<int, Recv<int, Send<int, Quit>>>;
```

```
struct Quit {};

template<typename A, typename T>
struct Send {};

template<typename A, typename T>
struct Recv {};

template<typename L, typename R>
struct Offer {};

template<typename L, typename R>
struct Choose {};
```

```
using Server = Recv<int, Recv<int, Send<int, Quit>>>;
using Client = Send<int, Send<int, Recv<int, Quit>>>;
```

```
struct Quit { using Dual = Quit; };

template<typename A, typename T>
struct Send { using Dual = Recv<A, typename T::Dual>; };

template<typename A, typename T>
struct Recv { using Dual = Send<A, typename T::Dual>; };

template<typename L, typename R>
struct Offer { using Dual = Choose<typename L::Dual, typename R::Dual>; };

template<typename L, typename R>
struct Choose { using Dual = Offer<typename L::Dual, typename R::Dual>; };
```

```
using Server = Recv<int, Recv<int, Send<int, Quit>>>;
using Client = Server::Dual;
```

Possible implementations:

```
Comm c; Protocol p;

int i;
auto p1 = send(c, p, i);
auto [p2, f] = recv(c, p1);
close(c, p2);
```

```
Comm c; Protocol p;

int i;
auto p1 = c.send(p, i);
auto [p2, f] = c.recv(p1);
c.close(p2);
```

```
Comm c; Protocol p;

int i;
auto p1 = p.send(c, i);
auto [p2, f] = p1.recv(c);
p2.close(c);
```

```
Comm c; Protocol p;

int i;
auto p1 = send(c, p, i);
auto [p2, f] = recv(c, p1);
close(c, p2);
```

```
Comm c; Protocol p;

int i;
auto p1 = c.send(p, i);
auto [p2, f] = c.recv(p1);
c.close(p2);
```

```
Comm c; Protocol p;

int i;
auto p1 = p.send(c, i);
auto [p2, f] = p1.recv(c);
p2.close(c);
```

```
Channel c;

int i;
auto c1 = c.send(i);
auto [c2, f] = c1.recv();
c2.close();
```

```
using Server = Recv<int, Recv<int, Send<int, Quit>>>;
using Client = Server::Dual;
```

```
void serverFunc(Comm chan) {
        auto v1 = chan.recv<int>();
        auto v2 = chan.recv<int>();
        chan.send(v1 + v2);
        chan.close();
}
```

```
using Server = Recv<int, Recv<int, Send<int, Quit>>>;
using Client = Server::Dual;
```

```
void serverFunc(Comm chan) {
        auto v1 = chan.recv<int>();
        auto v2 = chan.recv<int>();
        chan.send(v1 + v2);
        chan.close();
}
```

```
void serverFunc(Channel<Server> chan) {
    auto[c1, v1] = chan.recv();
    c1.
}
```

m_comm

recv — public : inline auto Chan<F

```
using Server = Recv<int, Recv<int, Send<int, Quit>>>;
using Client = Server::Dual;
```

```
void serverFunc(Comm chan) {
        auto v1 = chan.recv<int>();
        auto v2 = chan.recv<int>();
        chan.send(v1 + v2);
        chan.close();
}
```

```
void serverFunc(Channel<Server> chan) {
    auto[c1, v1] = chan.recv();
    auto[c2, v2] = c1.recv();
    c2.
}
```

m_comm

send

public : inline auto Chan<

```
using Server = Recv<int, Recv<int, Send<int, Quit>>>;
using Client = Server::Dual;
```

```
void serverFunc(Comm chan) {
        auto v1 = chan.recv<int>();
        auto v2 = chan.recv<int>();
        chan.send(v1 + v2);
        chan.close();
}
```

```
void serverFunc(Channel<Server> chan) {
    auto[c1, v1] = chan.recv();
    auto[c2, v2] = c1.recv();
    c2.send(v1 + v2).
}
```

close public
m_comm File: se

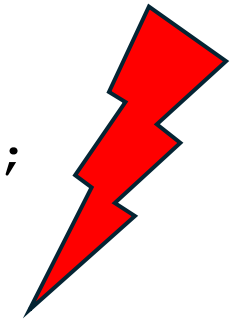
```
 using Server = Recv<int, Recv<int, Send<int, Quit>>>;
 using Client = Server::Dual;

void serverFunc(Comm chan) {
      auto v1 = chan.recv<int>();
      auto v2 = chan.recv<int>();
      chan.send(v1 + v2);
      chan.close();
}
```

```
void serverFunc(Channel<Server> chan) {
    auto[c1, v1] = chan.recv();
    auto[c2, v2] = c1.recv();
    c2.send(v1 + v2).close();
}
```

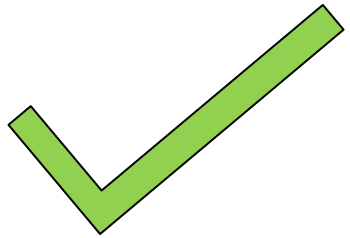
```
using Server = Recv<int, Recv<int, Send<int, Quit>>>;
using Client = Server::Dual;

void serverFunc(Comm chan) {
      auto v1 = chan.recv<int>();
      auto v2 = chan.recv<int>();
      chan.send(v1 + v2);
      chan.close();
}

void clientFunc(Comm chan) {
   cout << "First num: ";
   int x;
   cin >> x;
   chan.send(x);
   cout << "Second num: ";
   cin >> x;
   chan.send(x);
   auto r = chan.recv<int>();
   cout << "Result: " << r << endl;
   chan.close();
}
```

```
void serverFunc(Channel<Server> chan) {
   auto[c1, v1] = chan.recv();
   auto[c2, v2] = c1.recv();
   c2.send(v1 + v2).close();
}

void clientFunc(Channel<Client> chan) {
   cout << "First num: ";
   int x;
   cin >> x;
   auto c1 = chan.send(x);
   cout << "Second num: ";
   cin >> x;
   auto [c2, r] = c1.send(x).recv();
   cout << "Result: " << r << endl;
   c2.close();
}
```

```
using Server = Recv<int, Recv<int, Send<double, Quit>>>;
using Client = Server::Dual;

void serverFunc(Comm chan) {
    auto v1 = chan.recv<int>();
    auto v2 = chan.recv<int>();
    chan.send((double)v1 / v2);
    chan.close();
}

void clientFunc(Comm chan) {
    cout << "First num: ";
    int x;
    cin >> x;
    chan.send(x);
    cout << "Second num: ";
    cin >> x;
    chan.send(x);
    auto r = chan.recv<int>();
    cout << "Result: " << r << endl;
    chan.close();
}
```



```
void serverFunc(Channel<Server> chan) {
    auto[c1, v1] = chan.recv();
    auto[c2, v2] = c1.recv();
    c2.send((double)v1 / v2).close();
}

void clientFunc(Channel<Client> chan) {
    cout << "First num: ";
    int x;
    cin >> x;
    auto c1 = chan.send(x);
    cout << "Second num: ";
    cin >> x;
    auto [c2, r] = c1.send(x).recv();
    cout << "Result: " << r << endl;
    c2.close();
}
```



```
using Addition = Recv<int, Recv<int, Send<int,    Quit>>>;
using Division = Recv<int, Recv<int, Send<double, Quit>>>;

using Server = Offer<Addition, Division>;
using Client = Server::Dual;


void serverFunc(Channel<Server> chan) {
    auto offer = chan.offer();
    if (offer.index() == 0) {
        auto[c1, v1] = take<0>(chan).recv();
        auto[c2, v2] = c1.recv();
        c2.send(v1 + v2).close();
    }
    else {
        auto[c1, v1] = take<1>(chan).recv();
        auto[c2, v2] = c1.recv();
        c2.send((double)v1 / v2).close();
    }
}
```

```
using Addition = Recv<int, Recv<int, Send<int,    Quit>>>;
using Division = Recv<int, Recv<int, Send<double, Quit>>>;

using Server = Offer<Addition, Division>;
using Client = Server::Dual;

void clientFunc(Channel<Client> chan) {
    cout << "Choose service (0-add, 1-div): ";
    int service;
    cin >> service;
    if (service == 0) {
        auto c1 = chan.sel0();
        cout << "First num: ";
        int x;
        cin >> x;
        auto c2 = c1.send(x);
        ...
    }
    else {
        auto c1 = chan.sel1();
        ...
```

```
struct Quit { using Dual = Quit; };

template<typename A, typename T>
struct Send { using Dual = Recv<A, typename T::Dual>; };

template<typename A, typename T>
struct Recv { using Dual = Send<A, typename T::Dual>; };

template<typename L, typename R>
struct Offer { using Dual = Choose<typename L::Dual, typename R::Dual>; };

template<typename L, typename R>
struct Choose { using Dual = Offer<typename L::Dual, typename R::Dual>; };

template<typename T>
struct Loop { using Dual = Loop<typename T::Dual>; };

struct LoopRepeat { using Dual = LoopRepeat; };
```

```
using Addition = Recv<int, Recv<int, Send<int,    Offer<LoopRepeat, Quit>>>>;
using Division = Recv<int, Recv<int, Send<double, Offer<LoopRepeat, Quit>>>>;

using Server = Loop<Offer<Addition, Division>>;
using Client = Server::Dual;
```

```
template<typename A, typename T, typename CTX, typename CommType>
struct Chan<Send<A, T>, CTX, CommType> {
...
   auto send(A x) -> Chan<T, CTX, CommType> {
      m_comm.send(x);
      return { Chan(std::move(*this)).m_comm };
   }
private:
   CommType m_comm;
};

template<typename A, typename T, typename CTX, typename CommType>
struct Chan<Recv<A, T>, CTX, CommType> {
...
   auto recv() -> std::tuple<Chan<T, CTX, CommType>, A> {
      A x;
      m_comm.recv(x);
      return { Chan<T, CTX, CommType>(Chan(std::move(*this)).m_comm), x };
   }
};
```

```
template<typename L, typename R, typename CTX, typename CommType>
struct Chan<Offer<L, R>, CTX, CommType> {
      auto offer() -> Branch<Chan<L, CTX, CommType>, Chan<R, CTX, CommType>>
...
```

```
template<typename L, typename R, typename CTX, typename CommType>
struct Chan<Choose<L, R>, CTX, CommType> {
      auto sel0() -> Chan<L, CTX, CommType>;
      auto sel1() -> Chan<R, CTX, CommType>;
...
```

```
struct Chan<Loop<T>, CTX, CommType> {
      Chan<T, std::tuple<T, CTX>, CommType> enter();


template<typename T, typename CTX, typename CommType>
struct Chan<LoopRepeat, std::tuple<T, CTX>, CommType> {
      Chan<T, std::tuple<T, CTX>, CommType> loop();
```

```
using Service = Recv<int, Recv<int, Send<int, Offer<LoopRepeat, Quit>>>>;

using Server = Loop<Service>;
using Client = Server::Dual;
```

```
void serverFunc(Channel<Server> chan) {
    auto c1 = chan.enter();
    while (true) {
        auto [c2, v1] = c1.recv();
        auto [c3, v2] = c2.recv();
        auto c4 = c3.send(v1 + v2);
        auto o1 = c4.offer();
        if (o1.index() == 0) {
            c1 = take<0>(o1).loop();
        }
        else {
            take<1>(o1).close();
            break;
        }
    }
}
```

```
using Service = Recv<int, Recv<int, Send<int, Offer<LoopRepeat, Quit>>>>;

using Server = Loop<Service>;
using Client = Server::Dual;
```

```
void clientFunc(Channel<Client> chan) {
    auto c1 = chan.enter();
    while (true) {
        std::cout << "First num: ";
        int x;
        std::cin >> x;
        auto c2 = c1.send(x);
        c1.send(99);
        std::cout << "Second num: ";
        std::cin >> x;
        auto c3 = c2.send(x);
...
```

```
using Service = Recv<int, Recv<int, Send<int, Offer<LoopRepeat, Quit>>>>;

using Server = Loop<Service>;
using Client = Server::Dual;
```

```
void clientFunc(Channel<Client> chan) {
    auto c1 = chan.enter();
    while (true) {
        std::cout << "First num: ";
        int x;
        std::cin >> x;
        auto c2 = c1.send(x);
        c1.send(99);
        std::cout << "Second num: ";
        std::cin >> x;
        auto c3 = c2.send(x);
...
```

<source>:540:9: warning: Method called on moved-from object 'c1' [clang-analyzer-cplusplus.Move]
540 | c1.send(99);

Overhead?

- Code size, Clang 19:

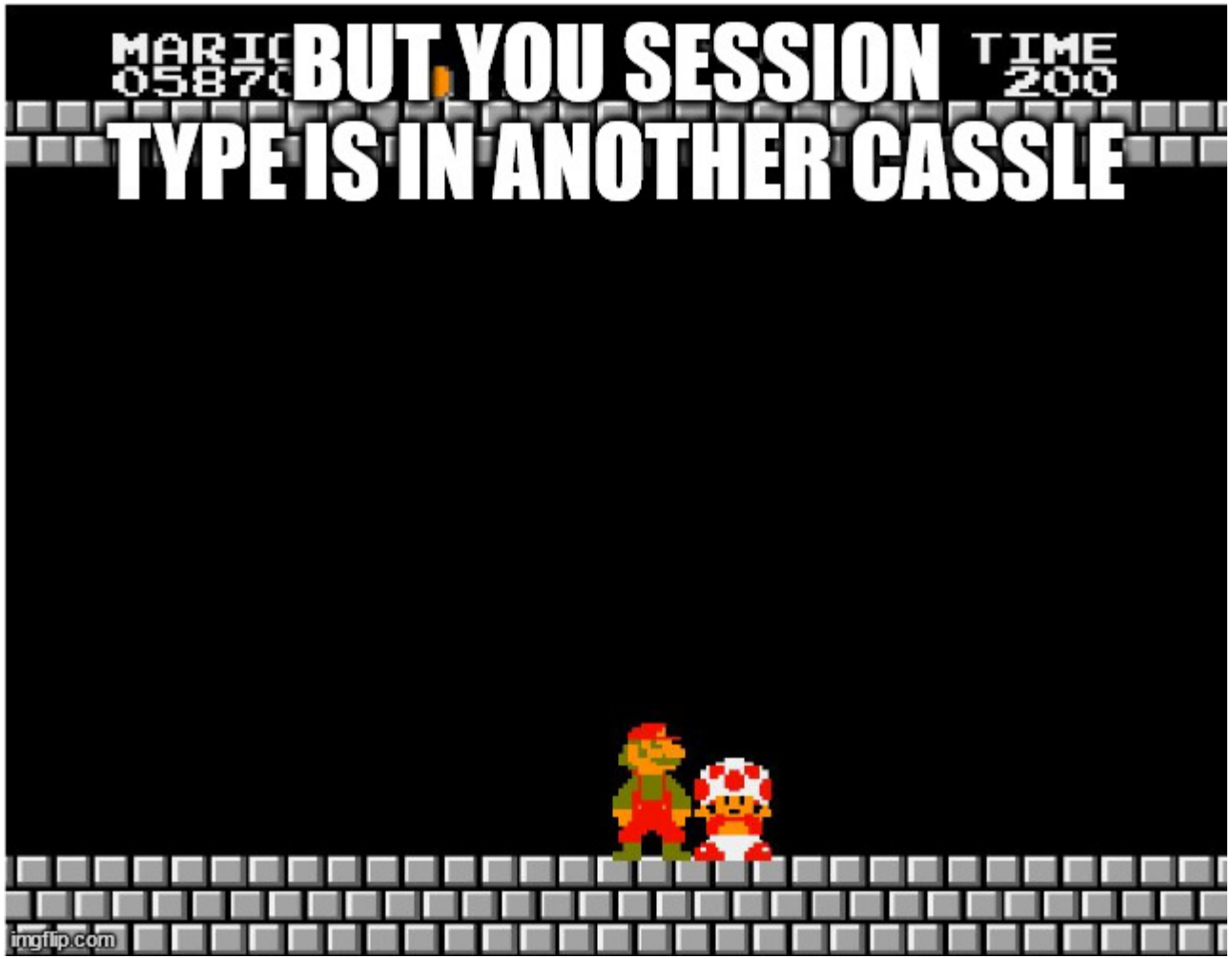
	No_ST	With_ST	
• Total:	743	vs 789	– **6.2%**
• Client func:	192	vs 205	– **6.8%**
• Server func:	57	vs 86	- **50%**

Further thing to do:

- Improve implementation:
  - Nicer session type expression
  - Drop connection handling
  - …
- Generation of Scribble code and verification
- Multiparty Session Types?

Let us check:

- Types are used for:
  - Abstraction
  - Documentation
  - Efficiency
  - Expressivity
  - Detecting errors
  - Safety


BUT YOU SESSION
TYPE IS IN ANOTHER CASSLE
TIME
imgflip.com

Thank you!