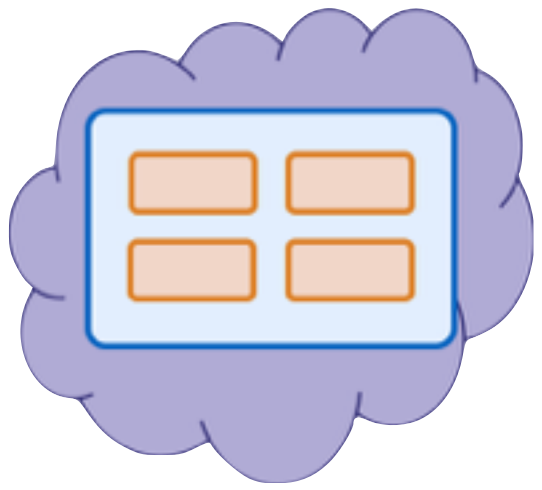
Programming Language Abstractions for Modularly Verified Distributed Systems



$$\vdash \{P\}\ c\ \{Q\}$$

James R. Wilcox   Zach Tatlock

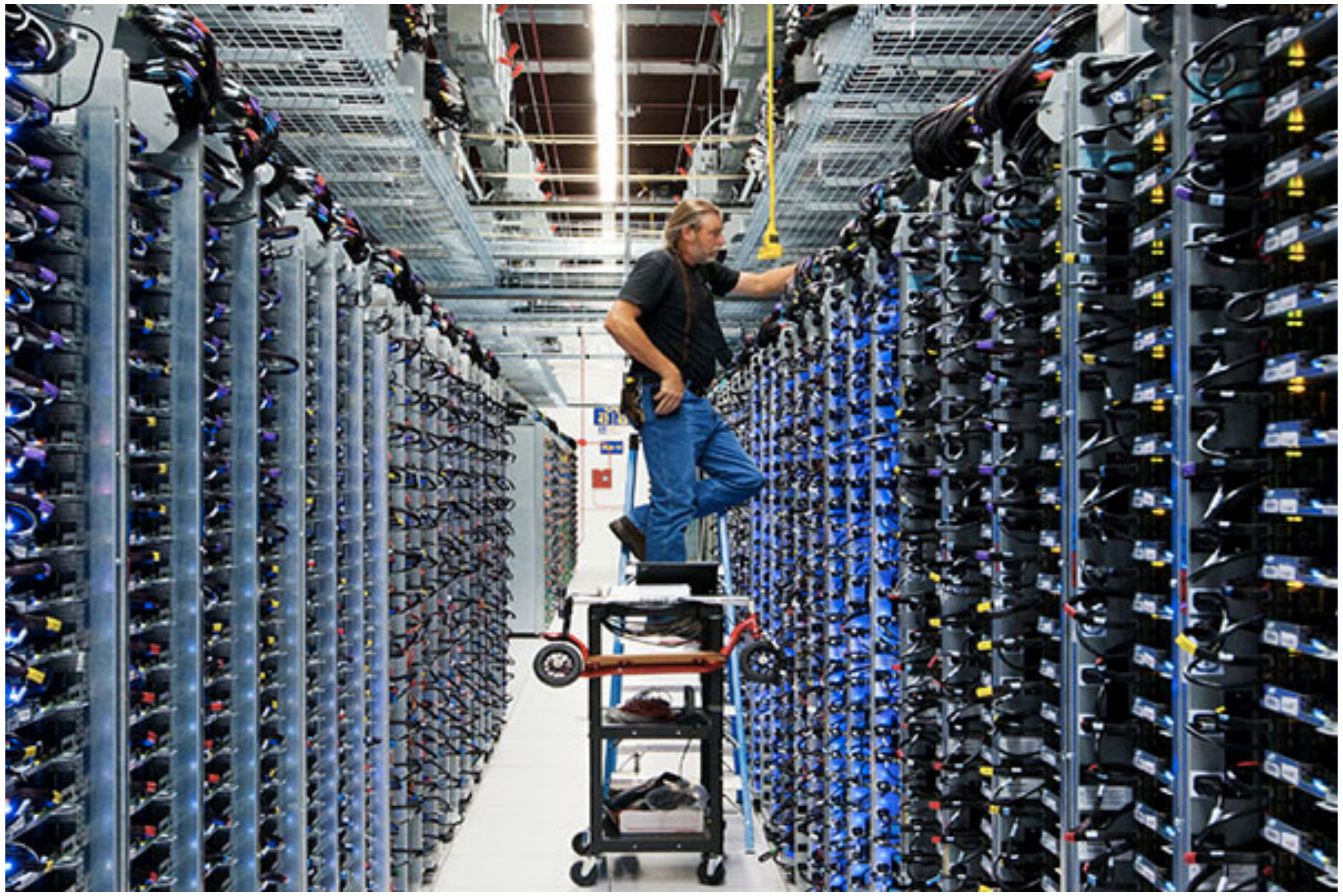
PLSE

PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

Ilya Sergey

UCL

Distributed Systems

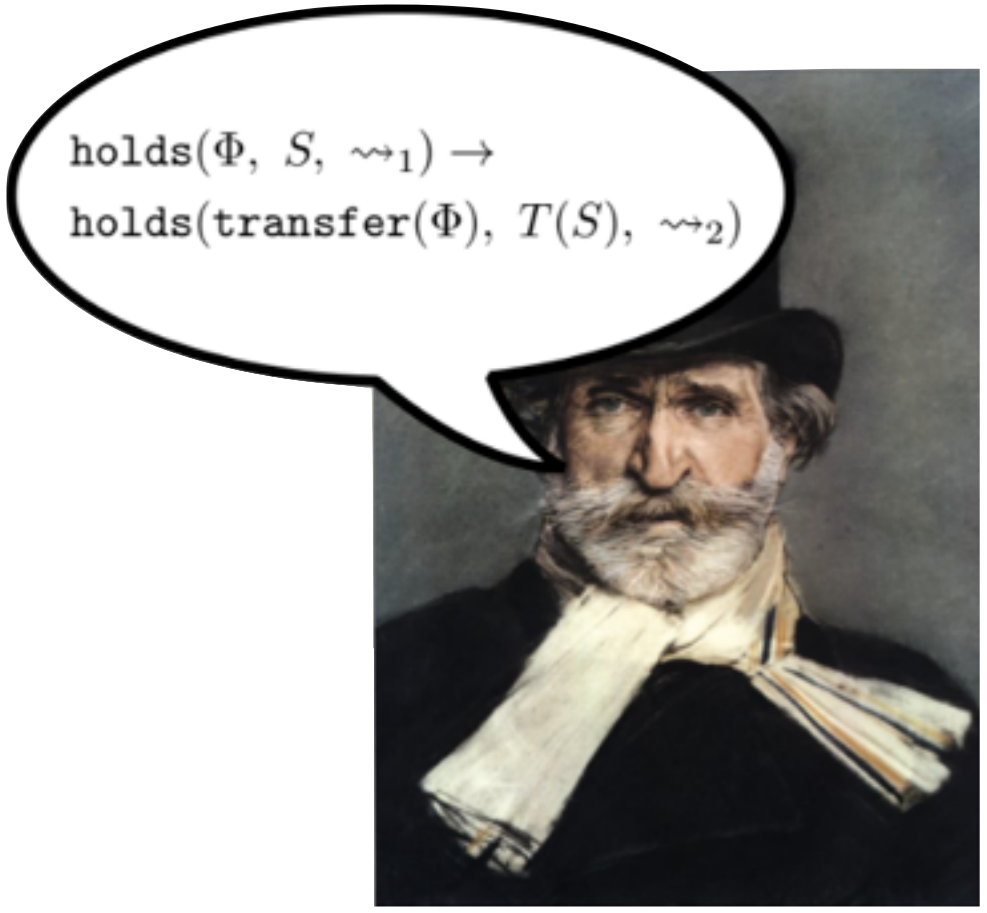


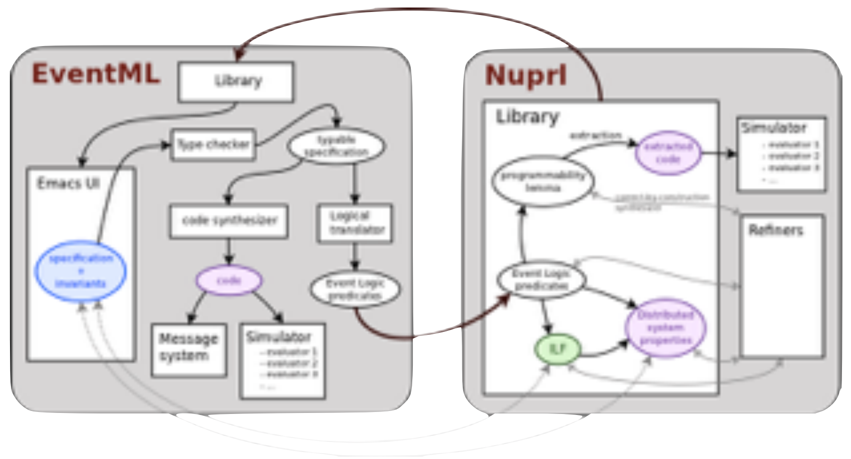
Distributed *Infrastructure*

Distributed *Applications*

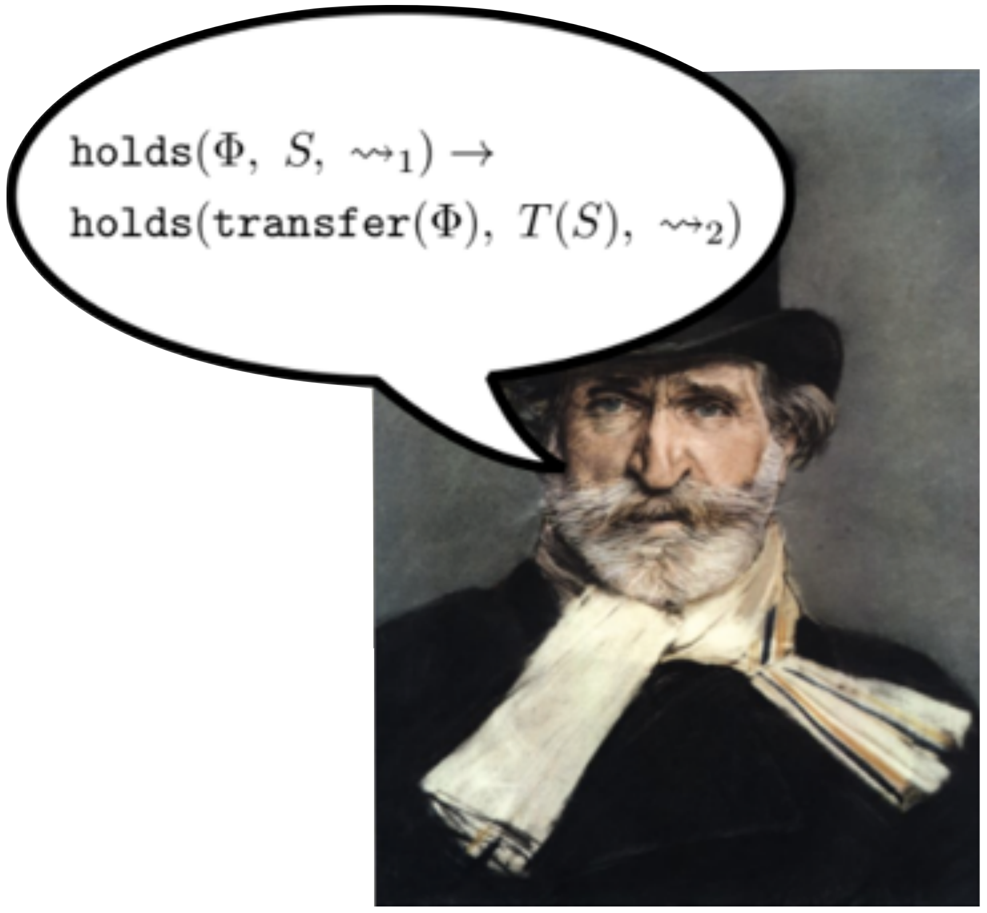
Verified Distributed Systems

Verified Distributed Systems


$\texttt{holds}(\Phi,\ S,\ \rightsquigarrow_1) \rightarrow$
$\texttt{holds}(\texttt{transfer}(\Phi),\ T(S),\ \rightsquigarrow_2)$

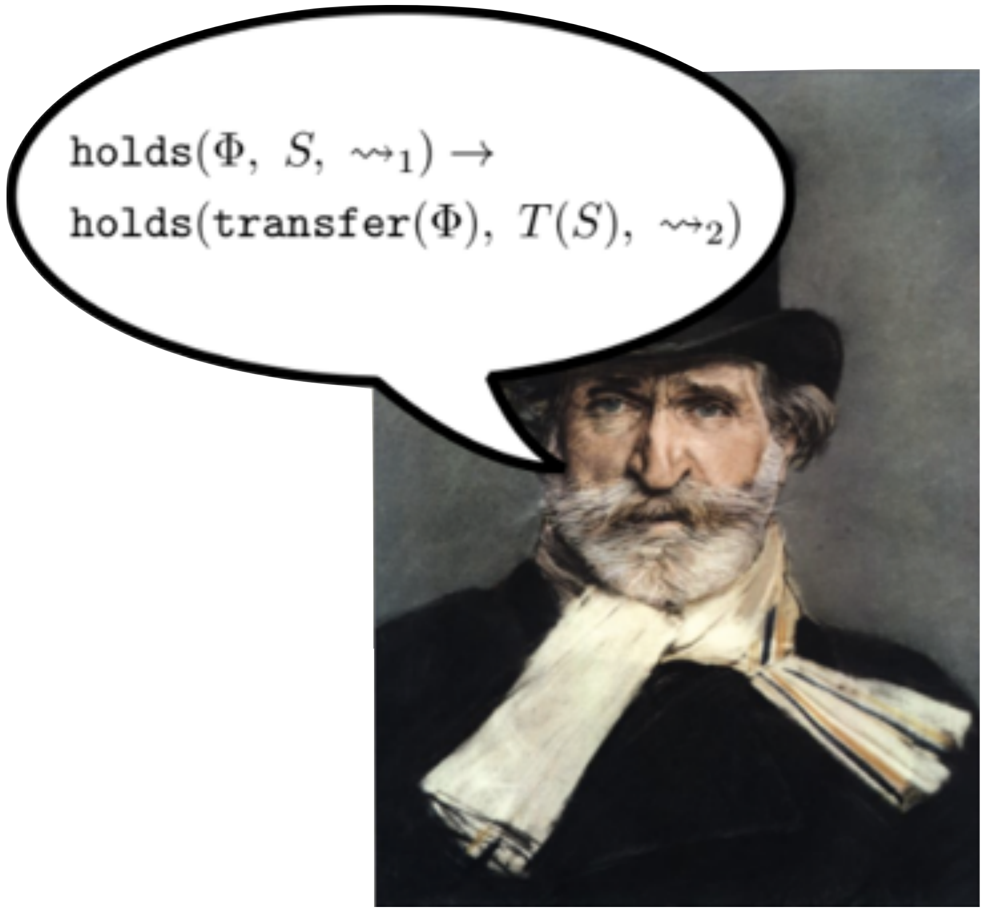


Verified Distributed *Infrastructure*

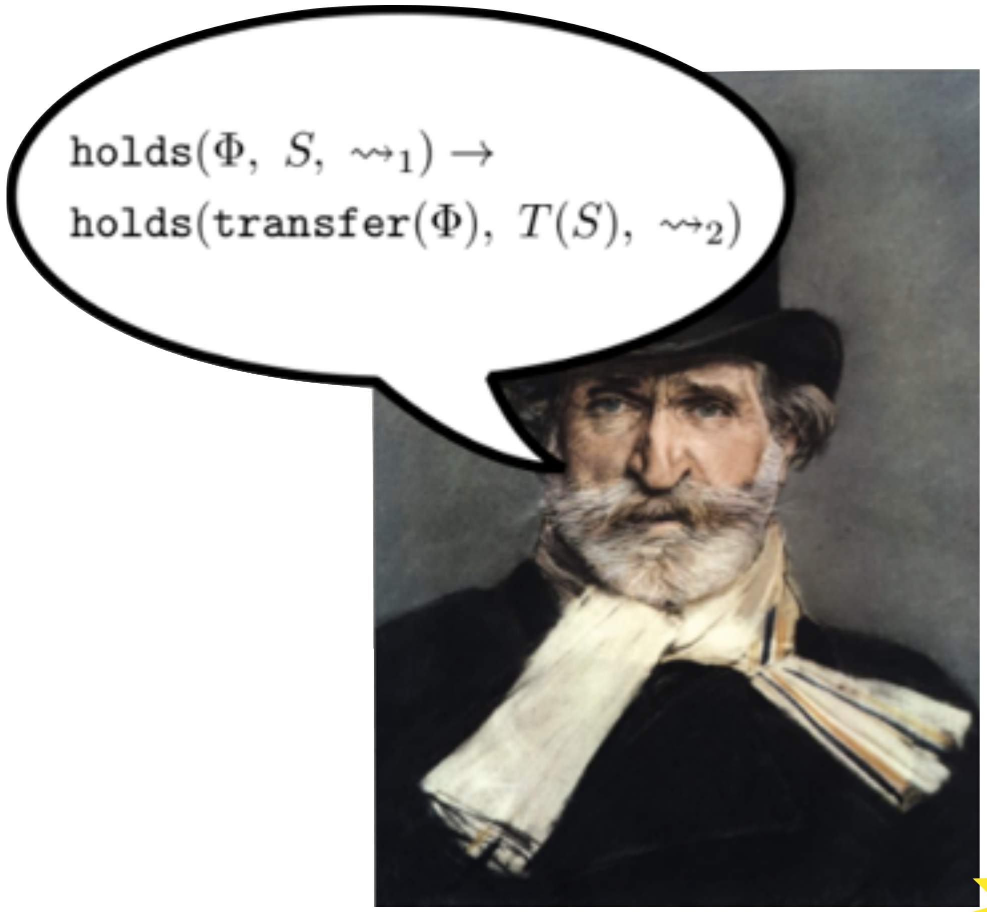

$\mathtt{holds}(\Phi,\ S,\ \leadsto_1) \to$
$\mathtt{holds}(\mathtt{transfer}(\Phi),\ T(S),\ \leadsto_2)$

Verified Distributed *Infrastructure*

Verified Distributed *Applications*


$\mathtt{holds}(\Phi,\ S,\ \leadsto_1) \rightarrow$
$\mathtt{holds}(\mathtt{transfer}(\Phi),\ T(S),\ \leadsto_2)$

Verified Distributed *Applications*


$\mathtt{holds}(\Phi,\ S,\ \rightsquigarrow_1) \rightarrow$
$\mathtt{holds}(\mathtt{transfer}(\Phi),\ T(S),\ \rightsquigarrow_2)$

Verified Distributed *Applications*

Challenging to verify apps in terms of infra.

verify clients by starting over!

Indicates deeper problems with composition

one node's client is another's server!

Challenging to verify apps in terms of infra.

verify clients by starting over!

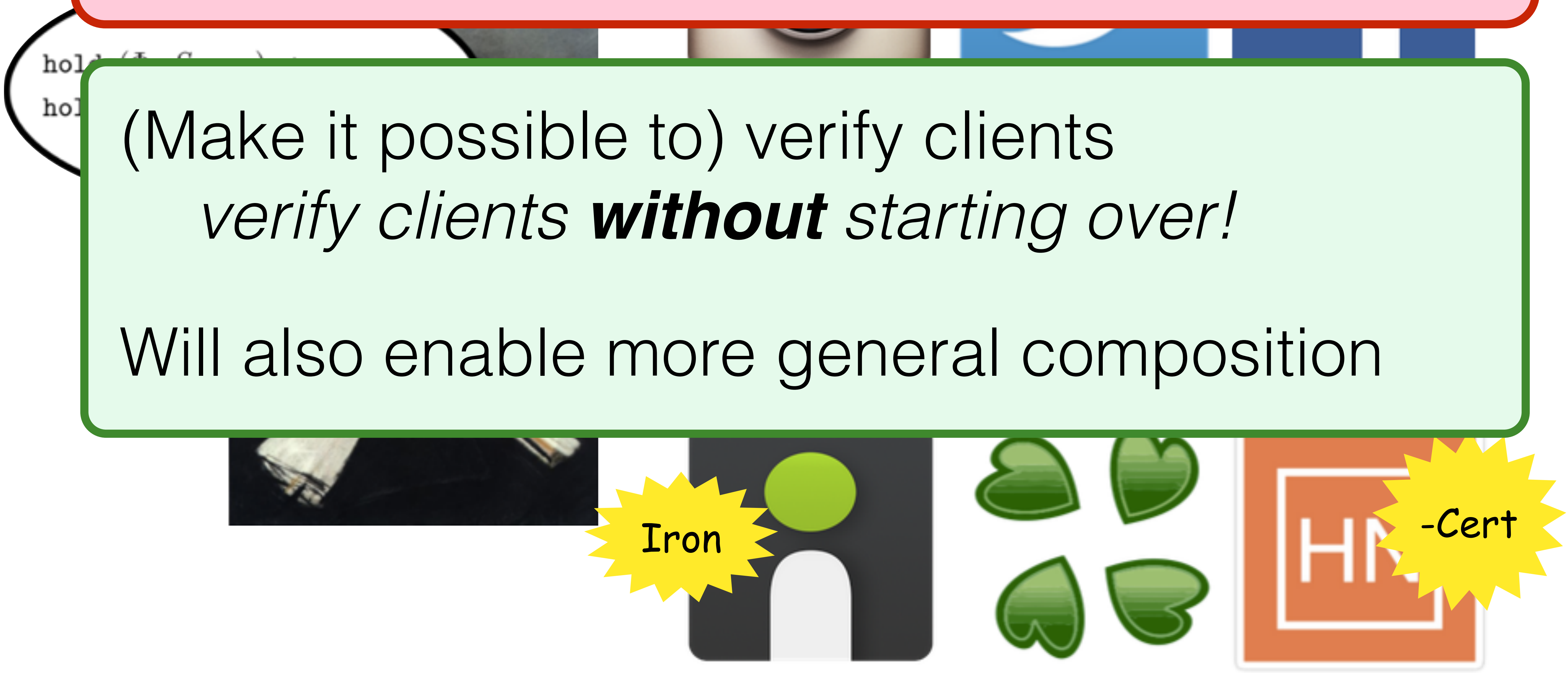
Indicates deeper problems with composition

one node's client is another's server!

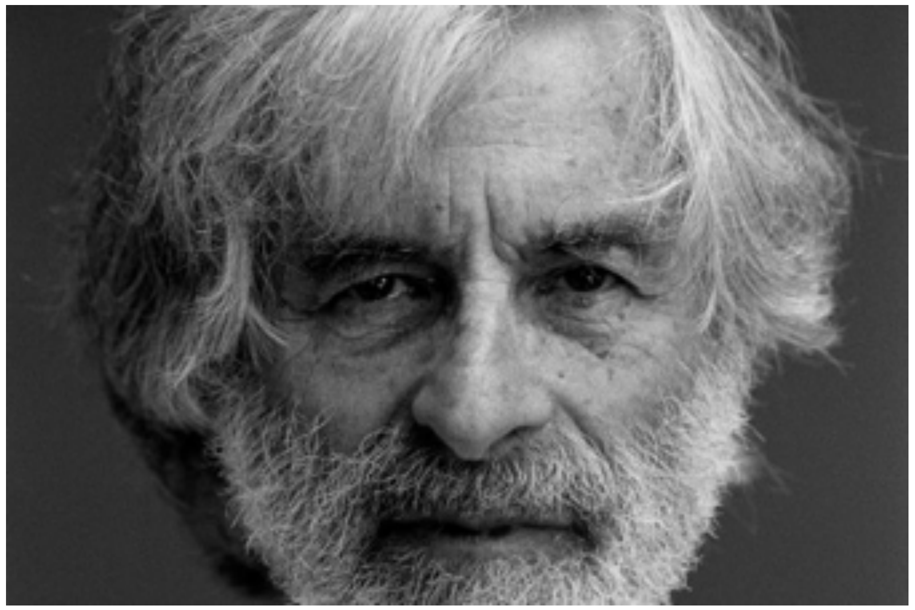
(Make it possible to) verify clients

*verify clients **without** starting over!*

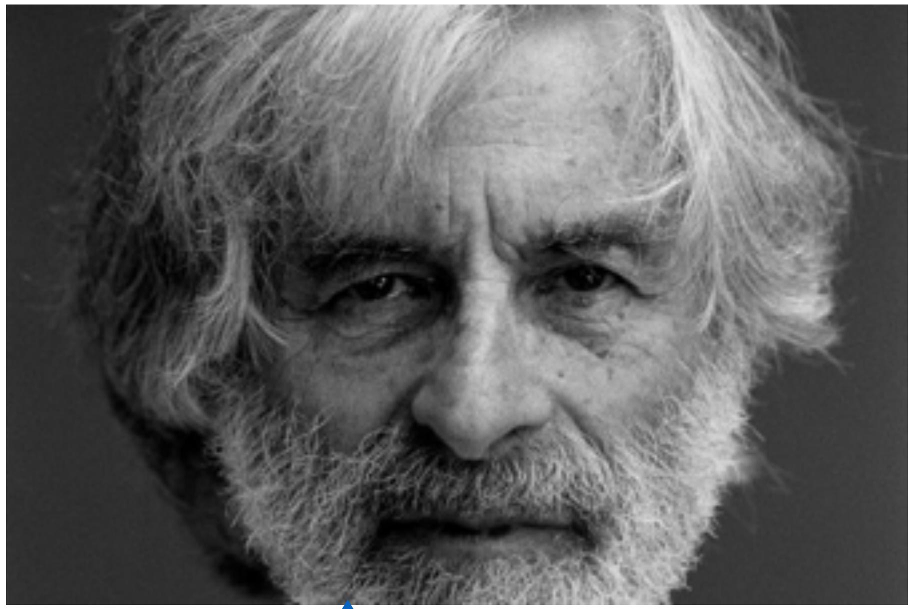
Will also enable more general composition



Composition: A way to make proofs harder



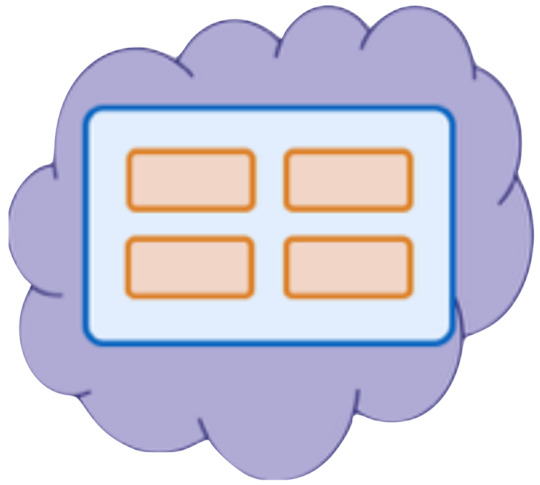
Composition: A way to make proofs harder



When distracting language issues are removed and the underlying mathematics is revealed, compositional reasoning is seen to be of little use.

Approach

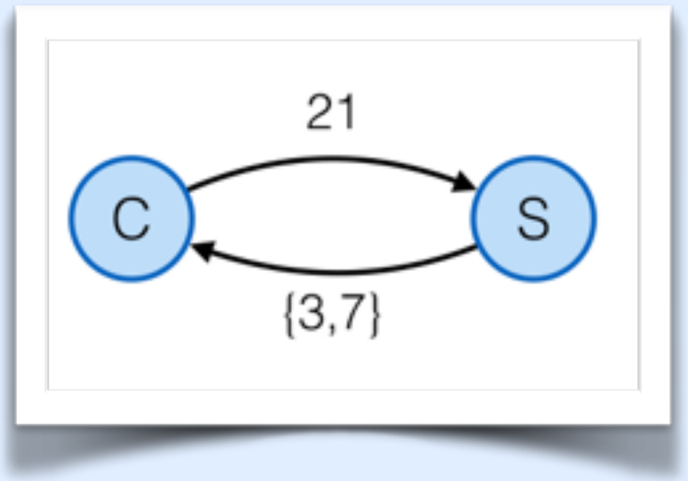
Distributed Hoare Type Theory



$$\vdash \{P\}\ c\ \{Q\}$$

Distributed Interactions

Servers and Clients



Combining Services

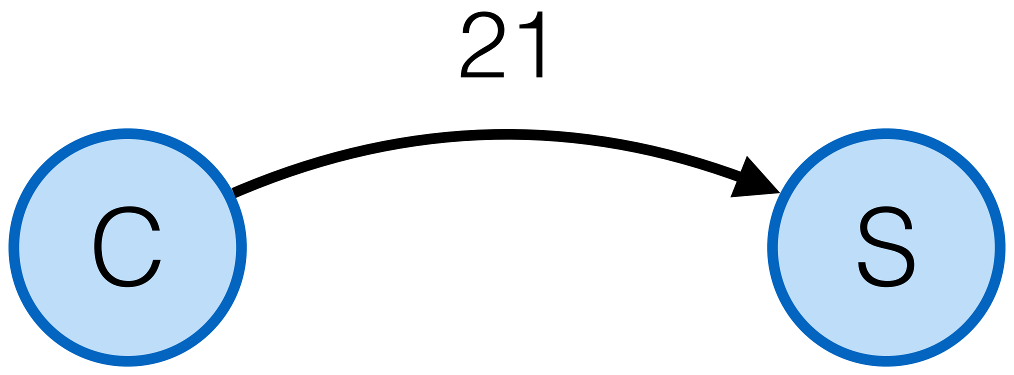
∘

Optimizations

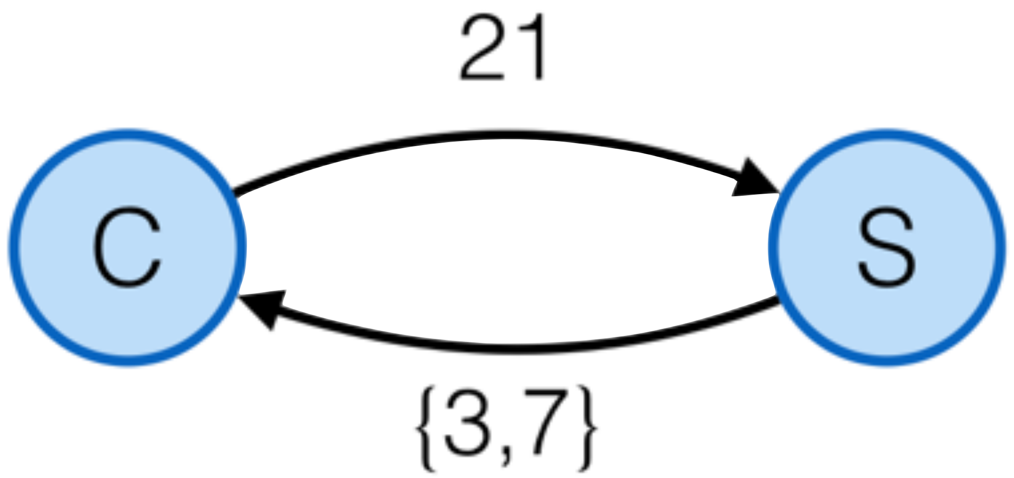
```
gcc -O3
```

Horizons

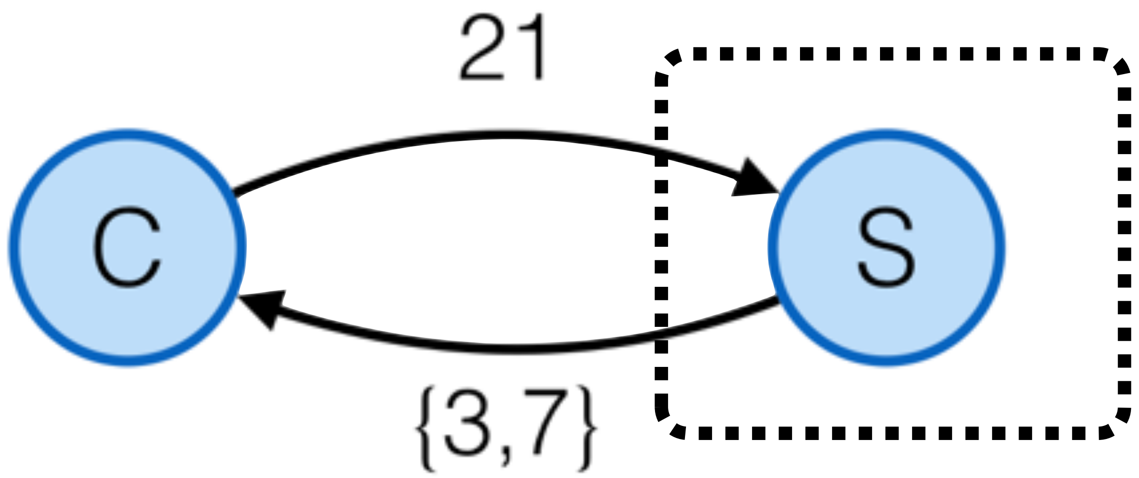
Cloud Compute



Cloud Compute



Cloud Compute



Cloud Compute: Server

```
while True:
  (from, n) <- recv
  send (n, factors(n)) to from
```

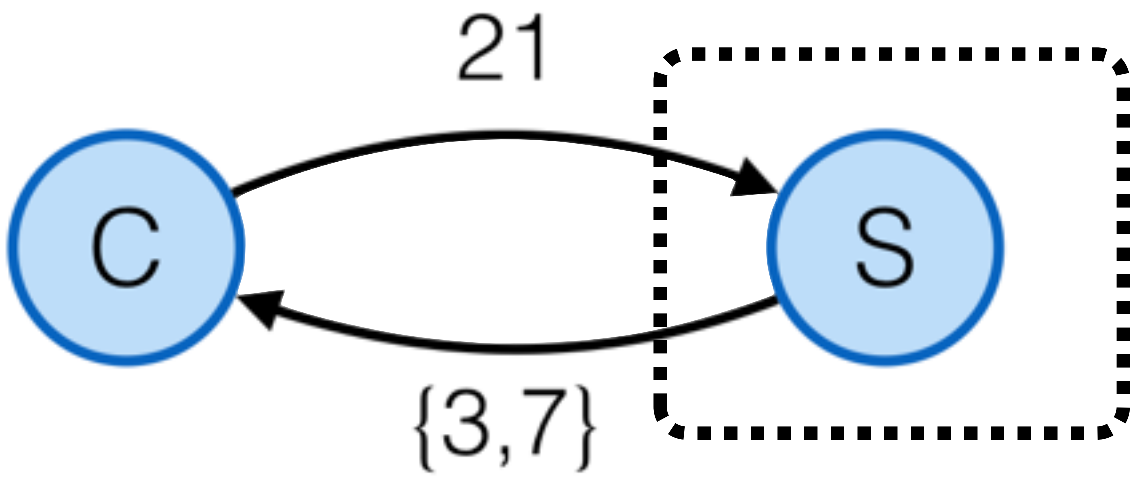
Cloud Compute: Server

```
while True:
  (from, n) <- recv
  send (n, factors(n)) to from
```

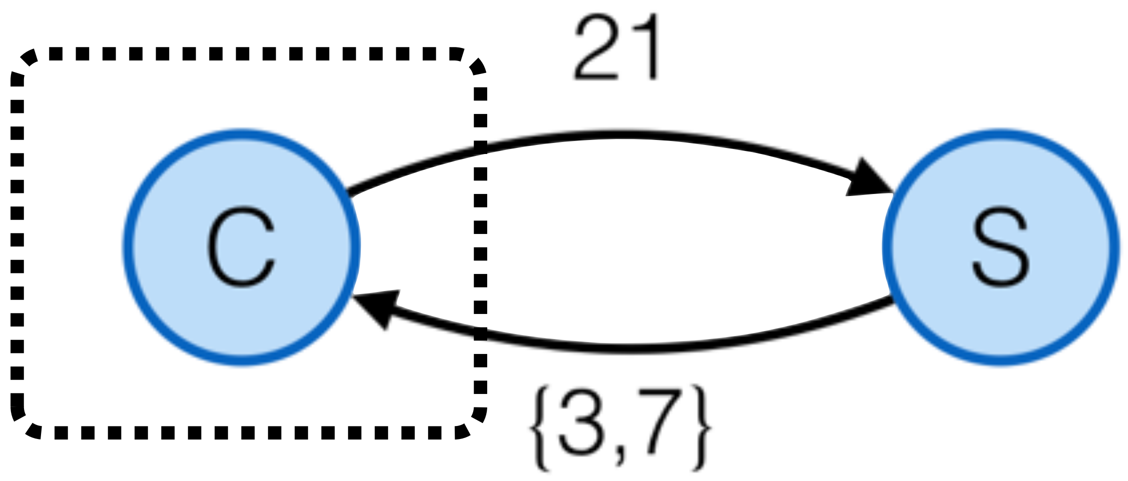
Traditional specification:
  messages from server have correct factors

Proved by finding an invariant of the system

Cloud Compute: Server



Cloud Compute: Client



Cloud Compute: Client

```
send 21 to server
(_, ans) <- recv
assert ans == {3, 7}
```

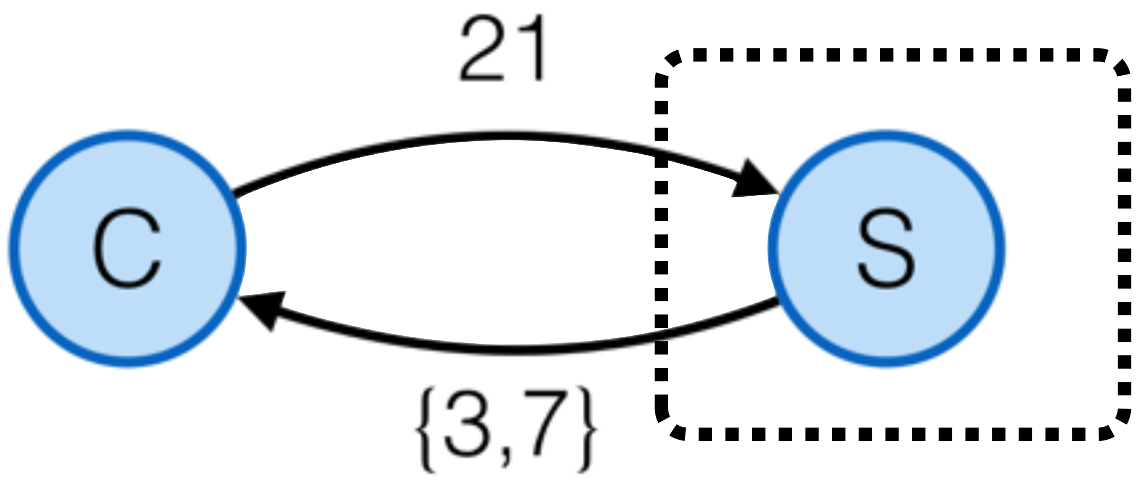
Cloud Compute: Client

```
send 21 to server
(_, ans) <- recv
assert ans == {3, 7}
```

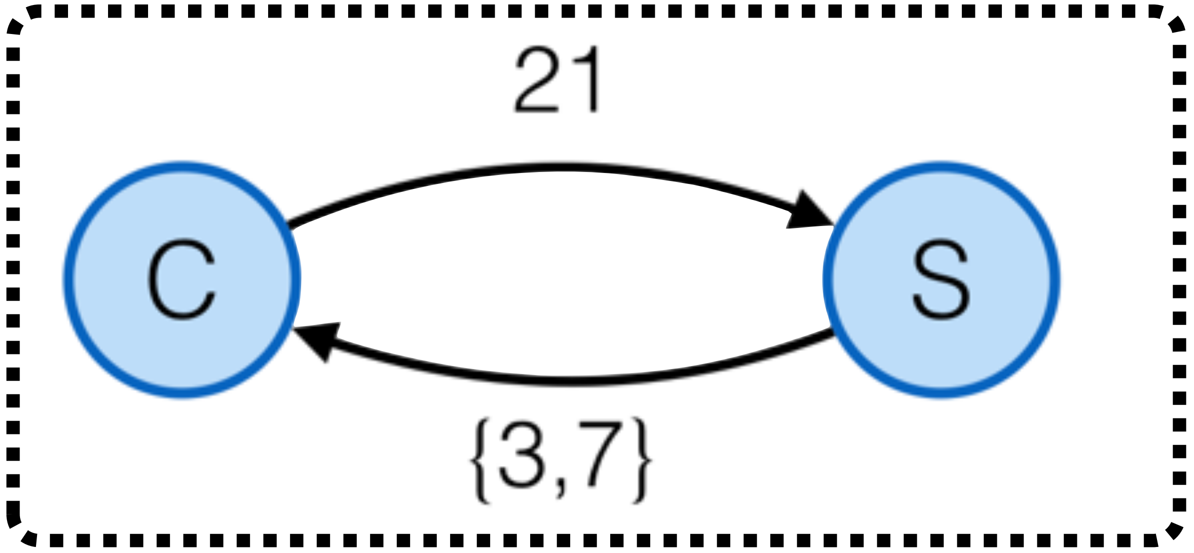
Expand system to include clients

Need to reason about client-server interaction
    introduce protocol

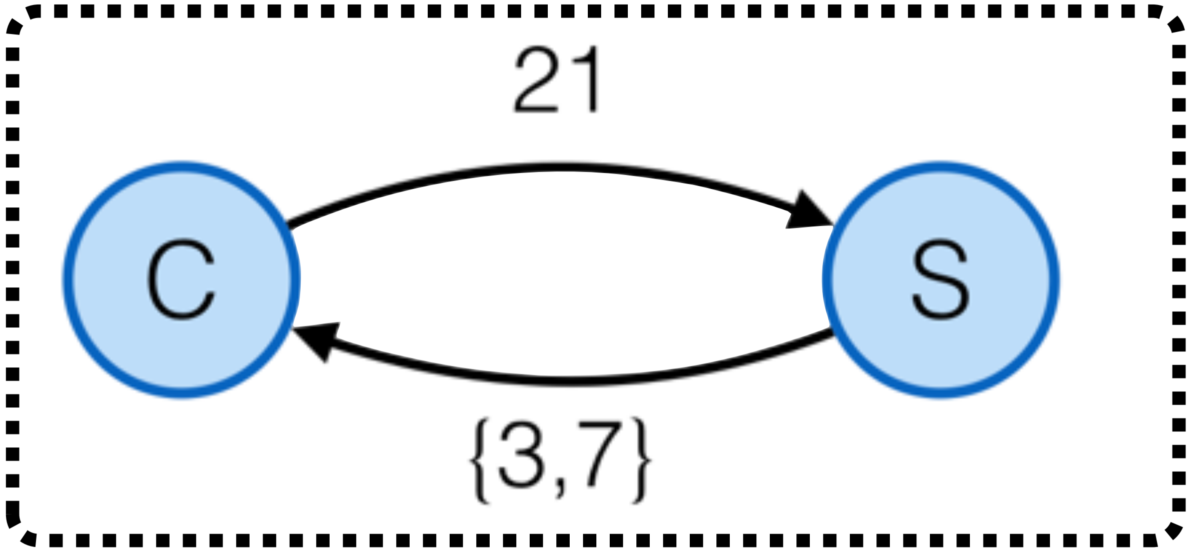
Protocols



Protocols

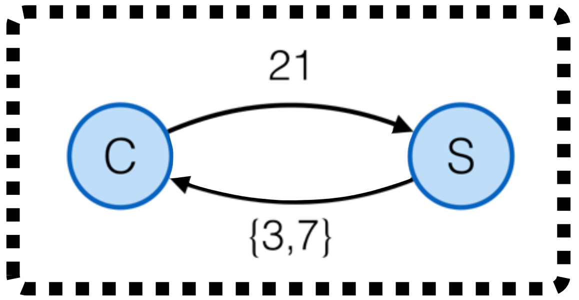


Protocols

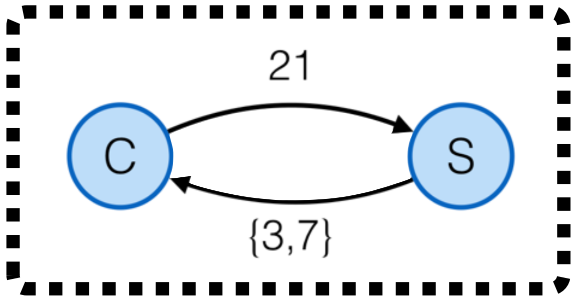


Protocols make it possible to verify clients!

Protocols



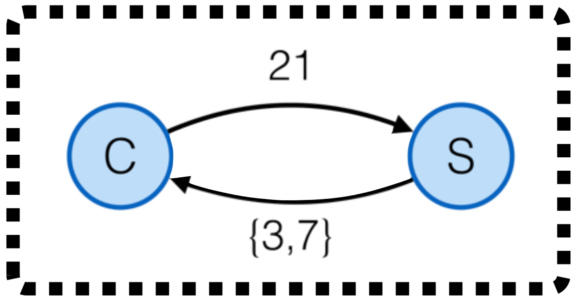
Protocols



State:

abstract state of each node

Protocols



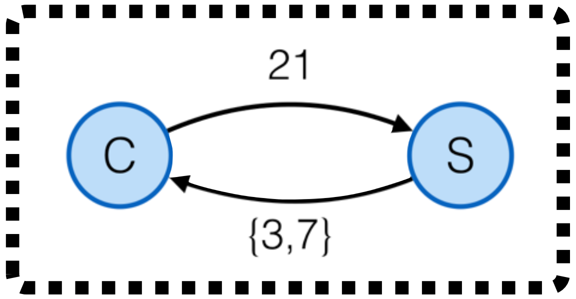
State:

abstract state of each node

Transitions:

allowed sends and receives

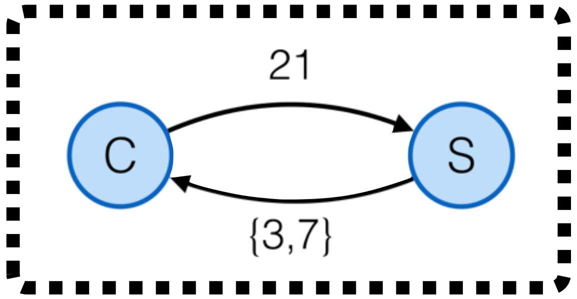
Cloud Compute Protocol



State:

Transitions:

Cloud Compute Protocol

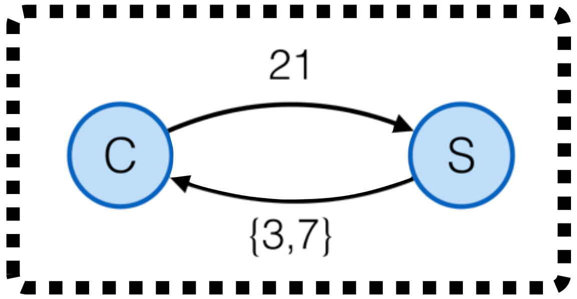


State:

```
permissions: Set<Msg>
```

Transitions:

Cloud Compute Protocol



State:

```
permissions: Set<Msg>
```

Transitions:

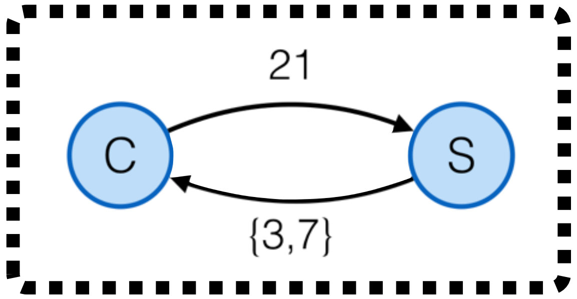
Send Req

Recv Req

Send Resp

Recv Resp

Cloud Compute: Protocol



Recv Request `n`

Effect: add `(from, n)` to `perm`

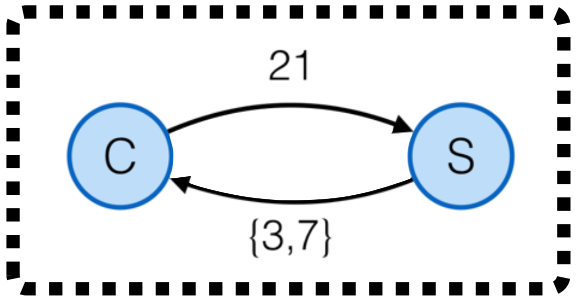
Send `Req`

Recv `Req`

Send `Resp`

Recv `Resp`

Cloud Compute: Protocol



Send Response `(n,l)`

Requires:

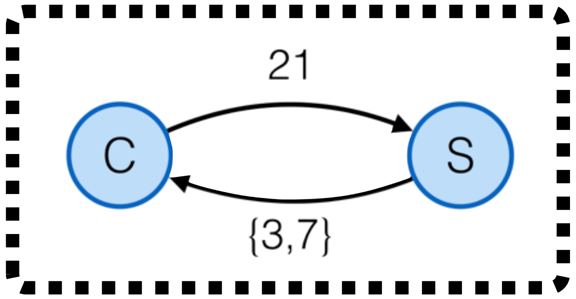
`l == factors(n)`

`(n,to)` in `perm`

Effect:

removes `(n,to)` from `perm`

Cloud Compute: Protocol



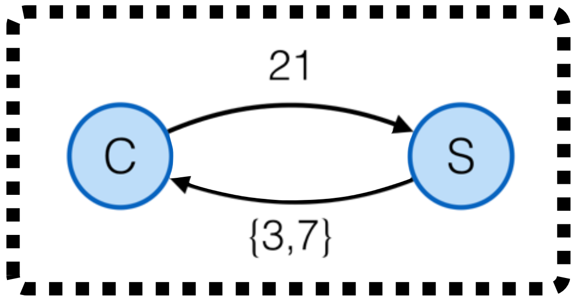
Recv Response `l`

Ensures:

`l == factors(n)`

`(n,to)` in `perm`

Cloud Compute: Protocol


$$\begin{tikzcd} C \arrow[r, bend left, "21"] & S \arrow[l, bend left, "\{3,7\}"] \end{tikzcd}$$

State:

```
permissions: Set<Msg>
```

Transitions:

Send Req

Recv Req

Send Resp

Recv Resp

Cloud Compute: Protocol

$$\begin{tikzcd} C \arrow[r, bend left, "21"] & S \arrow[l, bend left, "{\{3,7\}}"] \end{tikzcd}$$

Protocols make it possible to verify clients!

Transitions:

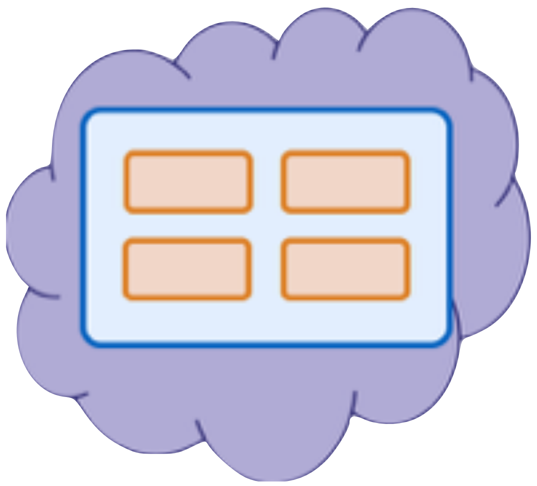
Send Req

Recv Req

Send Resp

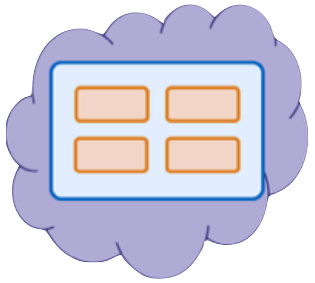
Recv Resp

From Protocols to Types

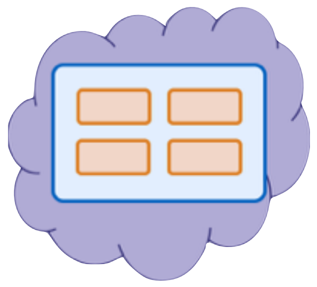


$\vdash \{P\}\ c\ \{Q\}$

From Protocols to Types

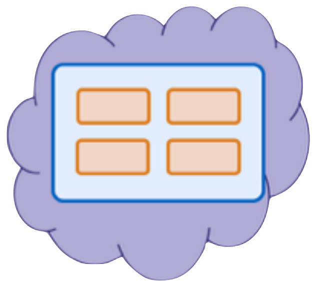

$\vdash \{ \quad \}$ `send m to h` $\{ \quad \}$

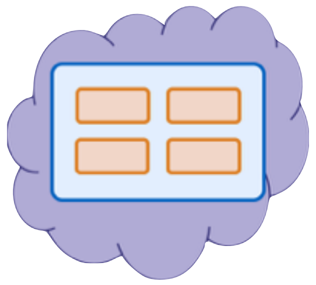
From Protocols to Types



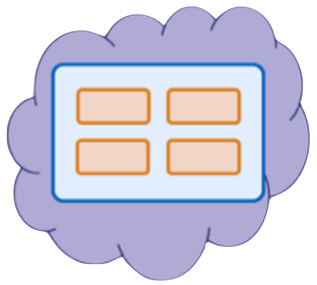
$$\frac{}{\vdash \{\quad\}\ \texttt{\textbf{send}}_{\boxed{t}}\ \texttt{m}\ \texttt{\textbf{to}}\ \texttt{h}\ \{\qquad\qquad\}}$$

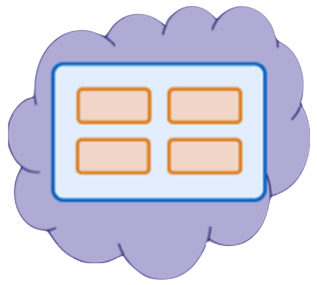
From Protocols to Types

$\boxed{t} \in$ 

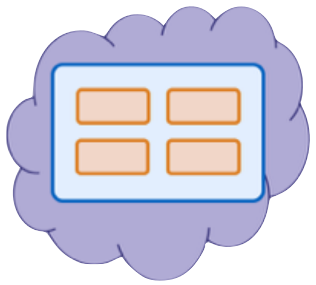
 $\vdash \{ \quad \}\ \texttt{\textbf{send}}_{\boxed{t}}\ \texttt{m}\ \texttt{\textbf{to}}\ \texttt{h}\ \{ \qquad \}$

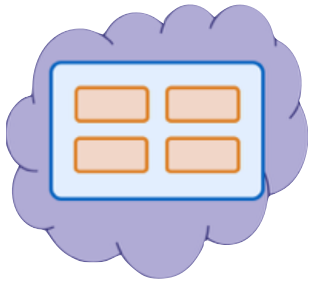
From Protocols to Types

$\boxed{t} \in$  $\qquad P \Rightarrow Pre_{\boxed{t}}$

 $\vdash \{P\}$ **send**$_{\boxed{t}}$ `m` **to** `h` $\{ \qquad\qquad\qquad \}$

From Protocols to Types

$\boxed{t} \in$  $\qquad P \Rightarrow Pre_{\boxed{t}}$

 $\vdash \{P\}$ $\texttt{\textbf{send}}_{\boxed{t}}$ `m` **`to`** `h` $\{sent_{\boxed{t}}(\texttt{m},\texttt{h})\}$

Cloud Compute: Client

```
send 21 to server
(_, ans) <- recv
assert ans == {3, 7}
```

Cloud Compute: Client

```
send 21 to server
(_, ans) <- recv
assert ans == {3, 7}
```

recv ensures correct factors

Cloud Compute: More Clients

```
send 21 to server₁
send 35 to server₂
(_, ans₁) <- recv
(_, ans₂) <- recv
assert ans₁ ∪ ans₂ == {3, 5, 7}
```

Cloud Compute: More Clients

```
send 21 to server_1
send 35 to server_2
(_, ans_1) <- recv
(_, ans_2) <- recv
assert ans_1 ∪ ans_2 == {3, 5, 7}
```

Same protocol enables verification

Cloud Compute: More Clients

```
send 21 to server₁
send 35 to server₂
(_, ans₁) <- recv
(_, ans₂) <- recv
assert ans₁ ∪ ans₂ == {3, 5, 7}
```

Combining Services

Same protocol enables verification

Cloud Compute: Server

```
while True:
  (from, n) <- recv
  send (n, factors(n)) to from
```

Cloud Compute: Server

```
while True:
  (from, n) <- recv
  send (n, factors(n)) to from
```

Precondition on **send** requires correct factors

Cloud Compute: More Servers

Optimizations

`gcc -O3`

```
cache = {}
while True:
   (from, n) <- recv
   ans = if n ∈ cache then cache[n]
         else factors(n)
   cache[n] = ans
   send (n, ans) to from
```

Cloud Compute: More Servers

```
cache = {}
while True:
   (from, n) <- recv
   ans = if n ∈ cache then cache[n]
         else factors(n)
   cache[n] = ans
   send (n, ans) to from
```

Optimizations

`gcc -O3`

Still follows protocol!

Cloud Compute: More Servers

```
while True:
  (from, n) <- recv
  send n to backend
  (_, ans) <- recv
  send (n, ans) to from
```

Cloud Compute: More Servers

```
while True:
    (from, n) <- recv
    send n to backend
    (_, ans) <- recv
    send (n, ans) to from
```

Still follows protocol!

Cloud Compute: More Servers

```
while True:
    (from, n) <- recv
    send n to backend
    (_, ans) <- recv
    send (n, ans) to from
```

Still follows protocol!

Any combination of transitions follows protocol

Well-typed programs don't go wrong!

One node's client is another's server!

Horizons

Sophisticated protocol composition

e.g. computation uses separate database

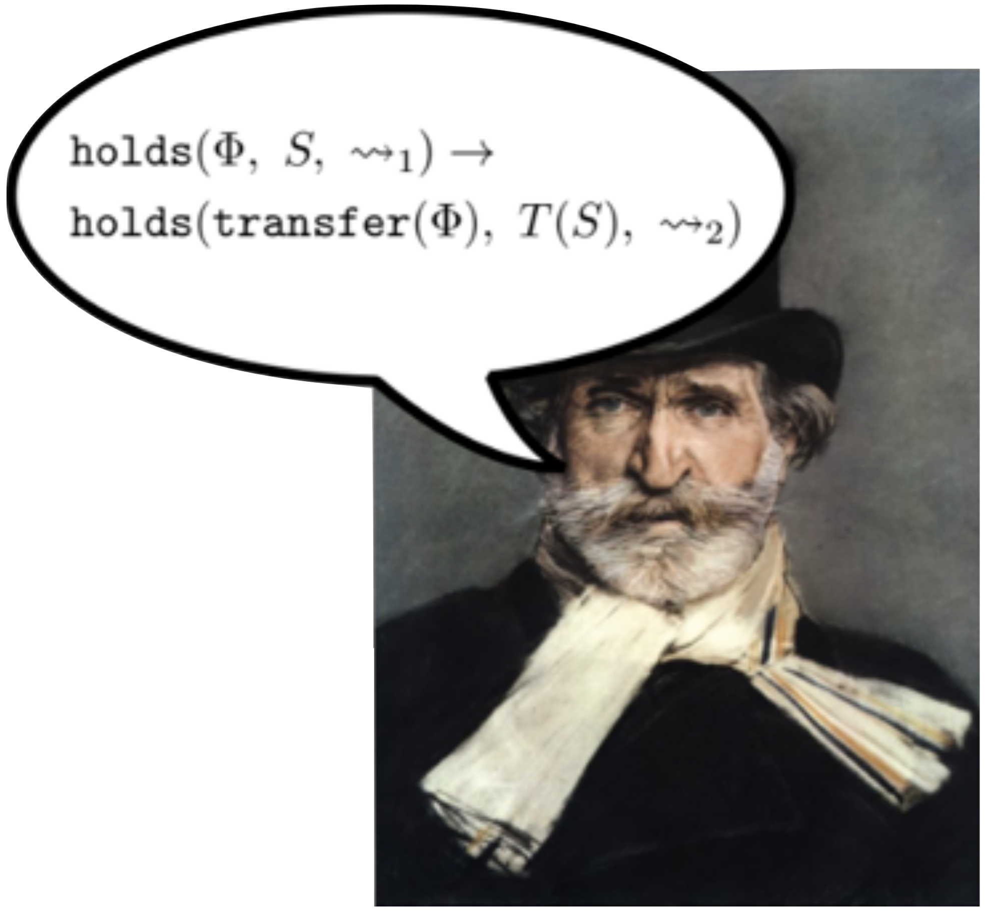
Adding other effects

e.g. mutable heap, threads, failure…

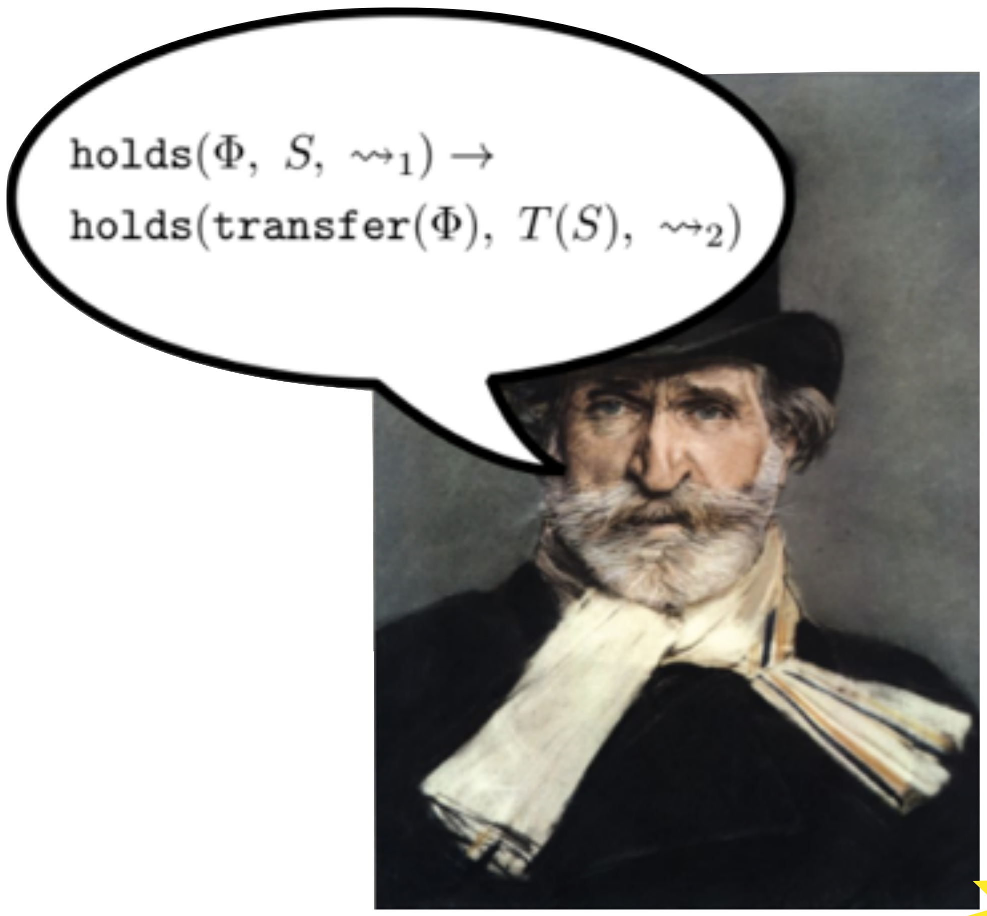
Fault tolerance

what do Verdi's VSTs look like here?

Verified Distributed *Applications*


$\mathtt{holds}(\Phi,\ S,\ \rightsquigarrow_1) \rightarrow$
$\mathtt{holds}(\mathtt{transfer}(\Phi),\ T(S),\ \rightsquigarrow_2)$

Verified Distributed *Applications*


$\texttt{holds}(\Phi,\ S,\ \rightsquigarrow_1) \rightarrow$
$\texttt{holds}(\texttt{transfer}(\Phi),\ T(S),\ \rightsquigarrow_2)$

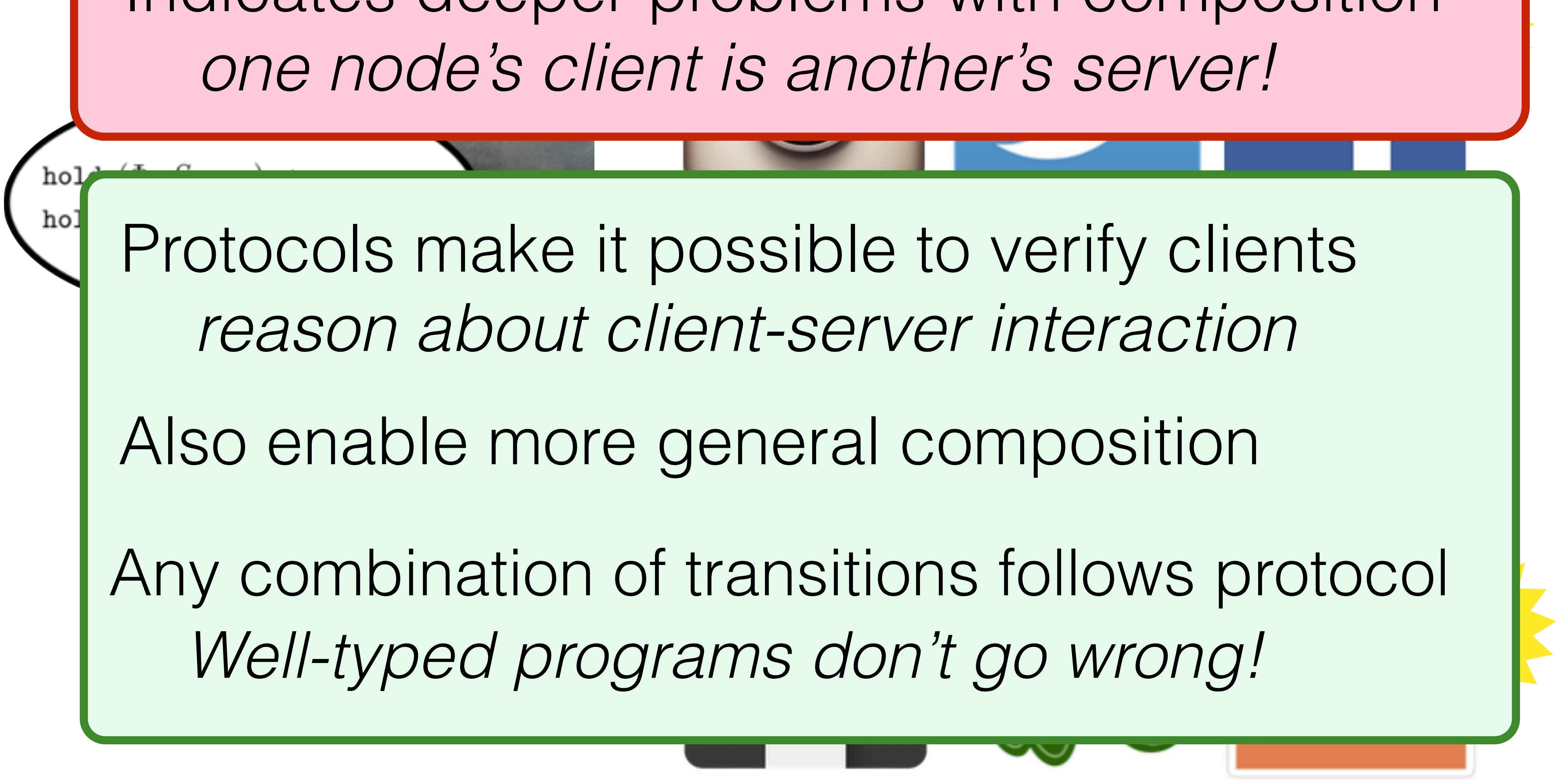
Verified Distributed *Applications*

Challenging to verify apps in terms of infra.

verify clients by starting over!

Indicates deeper problems with composition

one node's client is another's server!



Challenging to verify apps in terms of infra.

verify clients by starting over!

Indicates deeper problems with composition

one node's client is another's server!

Protocols make it possible to verify clients

reason about client-server interaction

Also enable more general composition

Any combination of transitions follows protocol

Well-typed programs don't go wrong!

Protocols make it possible to verify clients

reason about client-server interaction

Also enable more general composition

Any combination of transitions follows protocol

Well-typed programs don't go wrong!

Protocols make it possible to verify clients

reason about client-server interaction

Also enable more general composition

Any combination of transitions follows protocol

Well-typed programs don't go wrong!

Composition is hard

but important for infrastructure

Achieve with types

syntactic theory of composition