AARHUS
UNIVERSITY

CENTER FOR BASIC RESEARCH IN
PROGRAM VERIFICATION

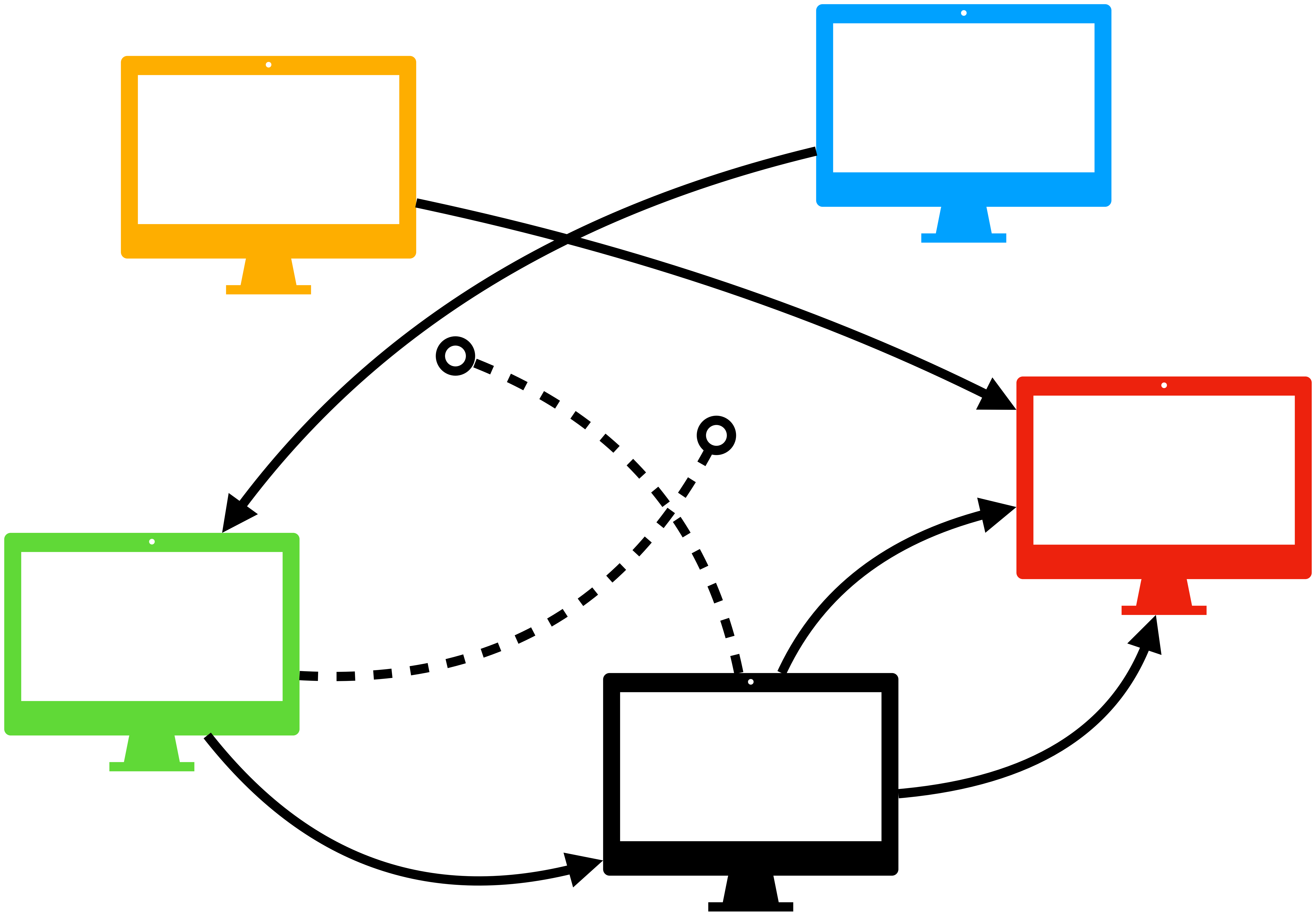
Aneris

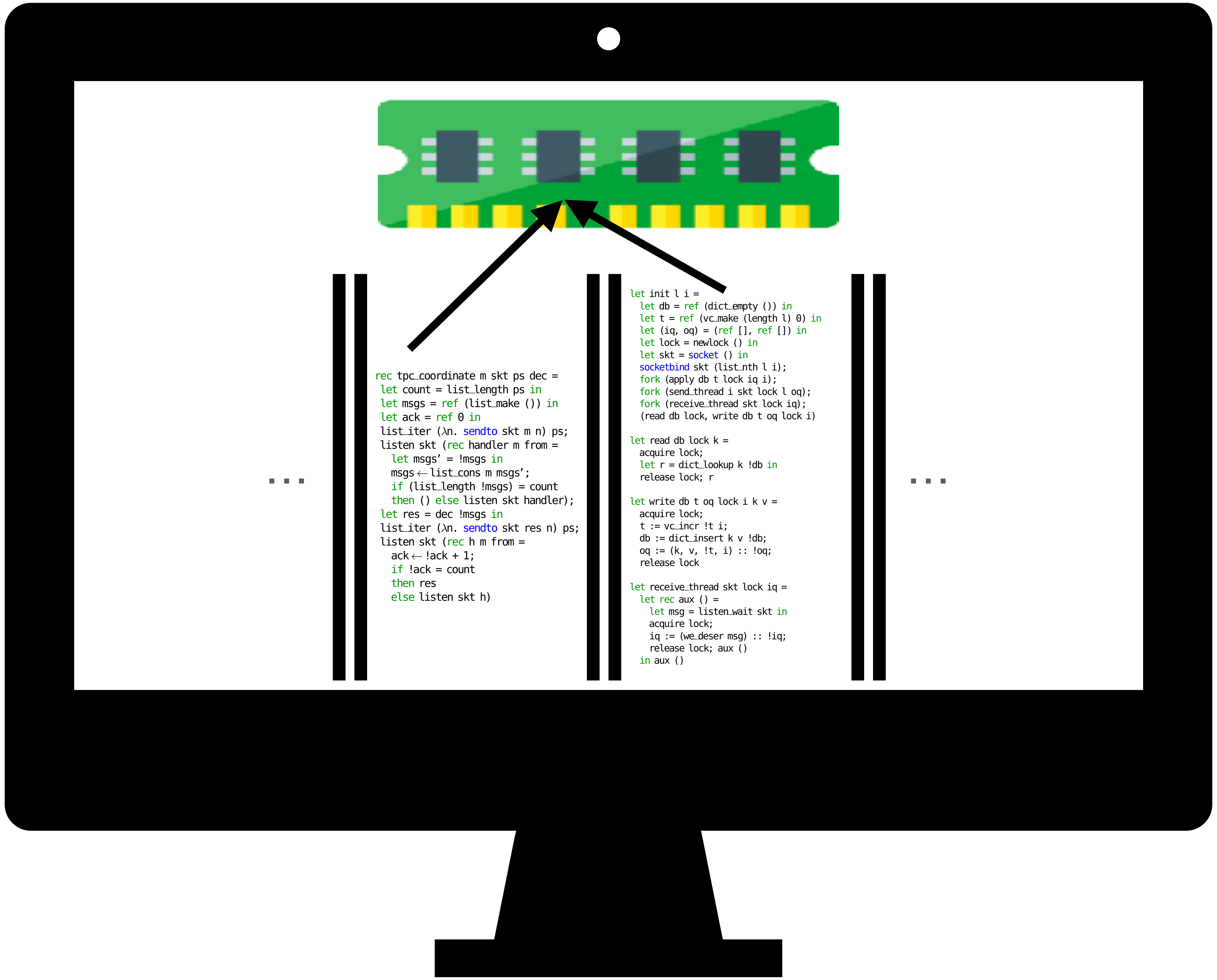
A Mechanised Logic for Modular Reasoning about Distributed Systems

March 29, 2021 @ ESOP'20

Simon Oddershede Gregersen

joint work with Morten Krogh-Jespersen, Amin Timany, Marit Edna Ohlenbusch, and Lars Birkedal





```
rec tpc_coordinate m skt ps dec =
 let count = list_length ps in
 let msgs = ref (list_make ()) in
 let ack = ref 0 in
 list_iter (λn. sendto skt m n) ps;
 listen skt (rec handler m from =
   let msgs’ = !msgs in
   msgs ← list_cons m msgs’;
   if (list_length !msgs) = count
   then () else listen skt handler);
 let res = dec !msgs in
 list_iter (λn. sendto skt res n) ps;
 listen skt (rec h m from =
   ack ← !ack + 1;
   if !ack = count
   then res
   else listen skt h)
```

```
let init l i =
  let db = ref (dict_empty ()) in
  let t = ref (vc_make (length l) 0) in
  let (iq, oq) = (ref [], ref []) in
  let lock = newlock () in
  let skt = socket () in
  socketbind skt (list_nth l i);
  fork (apply db t lock iq i);
  fork (send_thread i skt lock l oq);
  fork (receive_thread skt lock iq);
  (read db lock, write db t oq lock i)

let read db lock k =
  acquire lock;
  let r = dict_lookup k !db in
  release lock; r

let write db t oq lock i k v =
  acquire lock;
  t := vc_incr !t i;
  db := dict_insert k v !db;
  oq := (k, v, !t, i) :: !oq;
  release lock

let receive_thread skt lock iq =
  let rec aux () =
    let msg = listen_wait skt in
    acquire lock;
    iq := (we_deser msg) :: !iq;
    release lock; aux ()
  in aux ()
```

This work

- An ML-like language, **AnerisLang**, with higher-order store, node-local concurrency and datagram-like network sockets
- A separation logic, **Aneris,** for modular reasoning about partial correctness properties of **implementations** of distributed systems while allowing
    - **Vertical** composition
    - **Horizontal** composition
- All theory and examples are mechanised on top of the Iris framework in the Coq proof assistant

In **separation logic**, propositions denote ownership of resources, *e.g.*, $\ell \mapsto v$. Specifications only talk about the footprint of the program.

$$\{P\}\, e\, \{v.Q\}$$

These specifications can be lifted through **framing** and **binding**.

$$\frac{\{P\}\, e\, \{v.Q\}}{\{P * R\}\, e\, \{v.Q * R\}} \qquad \frac{\{P\}\, e\, \{v.Q\} \qquad \forall v. \{Q\}\, K[v]\, \{w.R\}}{\{P\}\, K[e]\, \{w.R\}}$$

In **concurrent separation logic**, threads are considered one at a time and we need not reason about inter-leavings of threads explicitly.

$$\frac{\{P_1\}\, e_1\, \{v_1.Q_1\} \qquad \{P_2\}\, e_2\, \{v_2.Q_2\}}{\{P_1 * P_2\}\, e_1 \mid\mid e_2\, \{v_1, v_2.Q_1 * Q_2\}}$$

Through **resource invariants**, we can express protocols on shared state.

$$\boxed{\exists v.\, \ell \mapsto v * even(v)}$$

In Aneris, a **distributed separation logic**, we extend these reasoning principles to allow reasoning about nodes one at a time.

$$\frac{\{P_1 * \mathsf{FreePorts}(ip_1, A)\}\, e_1 \,\{\mathsf{True}\} \qquad \{P_2 * \mathsf{FreePorts}(ip_2, B)\}\, e_2 \,\{\mathsf{True}\}}{\{P_1 * P_2 * \mathsf{FreeIp}(ip_1) * \mathsf{FreeIp}(ip_2)\}\, e_1 \;{}_{ip_1}|||_{ip_2}\; e_2 \,\{\mathsf{True}\}}$$

If $A \cap B = \emptyset$ then

$$\mathsf{FreePorts}(ip, A) * \mathsf{FreePorts}(ip, B) \dashv\vdash \mathsf{FreePorts}(ip, A \cup B)$$

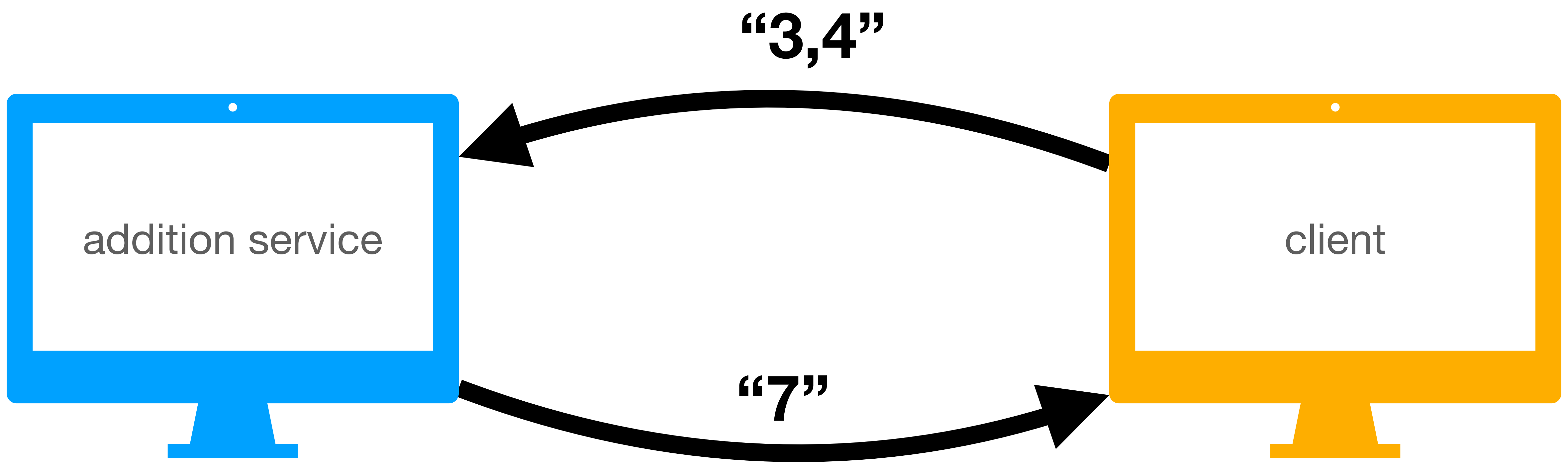
To express protocols on communication, we introduce **socket protocols**.

$$\Phi : \mathit{Message} \to \mathit{iProp}$$

With each socket address (pair of an ip an a port) we associate a protocol.

$$a \Mapsto \Phi$$

An addition service



```
rec server a =
  let skt = socket () in
  socketbind skt a;
  listen skt (rec handler msg from =
    let (n, m) = deserialize msg in
    let res = serialize (n + m) in
    sendto skt res from;
    listen skt handler)
```

```
rec client x y srv a =
  let skt = socket () in
  socketbind skt a;
  let m = serialize (x, y) in
  sendto skt m srv;
  let res = listenwait skt in
  deserialize res
```

Verifying an addition service

We primordially fix a socket protocol for the server.

$$
\begin{aligned}
\Phi_{add}(m) \triangleq \exists \Psi : \textit{Message} \rightarrow \textit{iProp}, (x, y : \mathbb{N}).\ & \\
& \mathsf{body}(m) = \mathit{serialize}(x, y) \ast \\
& \mathsf{from}(m) \Mapsto \Psi \ast \\
& \forall m'.\, \mathsf{body}(m') = \mathit{serialize}(x + y) \mathrel{-\!\!\ast} \Psi(m')
\end{aligned}
$$

Verifying an addition service

$\{(ip, p) \Mapsto \Phi_{add} * \mathsf{FreePorts}(ip, \{p\})\}$

$\quad \texttt{server}\ (ip, p)$

$\{False\}$

Verifying an addition service

$$\{(ip, p) \Mapsto \Phi_{add} * \mathsf{FreePorts}(ip, \{p\})\}$$

```
(rec server a =
  let skt = socket () in
  socketbind skt a;
  listen skt (rec handler msg from =
    let (n, m) = deserialize msg in
    let res = serialize (n + m) in
    sendto skt res from;
    listen skt handler)) (ip, p)
```

$$\{False\}$$

Verifying an addition service

```
(rec server a =
  let skt = socket () in
  socketbind skt a;
  listen skt (rec handler msg from =
    let (n, m) = deserialize msg in
    let res = serialize (n + m) in
    sendto skt res from;
    listen skt handler)) (ip, p)
```

$\{False\}$

$(ip, p) \Mapsto \Phi_{add}$

$\text{FreePorts}(ip, \{p\})$

Verifying an addition service

```
let skt = socket () in
socketbind skt (ip, p);
listen skt (rec handler msg from =
  let (n, m) = deserialize msg in
  let res = serialize (n + m) in
  sendto skt res from;
  listen skt handler)
```

$\{False\}$

$(ip, p) \Mapsto \Phi_{add}$

$\text{FreePorts}(ip, \{p\})$

Verifying an addition service

$\{h \hookrightarrow \texttt{None}\}$

```
let skt = h in
socketbind skt (ip, p);
listen skt (rec handler msg from =
  let (n, m) = deserialize msg in
  let res = serialize (n + m) in
  sendto skt res from;
  listen skt handler)
```

$\{False\}$

$(ip, p) \Mapsto \Phi_{add}$

$\text{FreePorts}(ip, \{p\})$

Verifying an addition service

```
let skt = h in
socketbind skt (ip, p);
listen skt (rec handler msg from =
  let (n, m) = deserialize msg in
  let res = serialize (n + m) in
  sendto skt res from;
  listen skt handler)
```

$\{False\}$

$(ip, p) \Rrightarrow \Phi_{add}$

$\mathsf{FreePorts}(ip, \{p\})$

$h \hookrightarrow \texttt{None}$

Verifying an addition service

```
socketbind h (ip, p);
listen h (rec handler msg from =
  let (n, m) = deserialize msg in
  let res = serialize (n + m) in
  sendto h res from;
  listen h handler)
```

$\{False\}$

$$(ip, p) \Mapsto \Phi_{add}$$

$$\text{FreePorts}(ip, \{p\})$$

$$h \hookrightarrow \texttt{None}$$

Verifying an addition service

$\{h \hookrightarrow \mathsf{Some}\ (ip, p)\}$

```
listen h (rec handler msg from =
  let (n, m) = deserialize msg in
  let res = serialize (n + m) in
  sendto h res from;
  listen h handler)
```

$\{False\}$

$(ip, p) \Mapsto \Phi_{add}$

Verifying an addition service

$$(ip, p) \Mapsto \Phi_{add}$$

$$h \hookrightarrow \mathsf{Some}\ (ip, p)$$

```
listen h (rec handler msg from =
  let (n, m) = deserialize msg in
  let res = serialize (n + m) in
  sendto h res from;
  listen h handler)
```

$$\{False\}$$

Verifying an addition service

$\{\Phi_{add}(m)\}$

```
let (n, m) = deserialize body(m) in
let res = serialize (n + m) in
sendto h res from(m)
```

$$(ip, p) \Mapsto \Phi_{add}$$

$$h \hookrightarrow \mathsf{Some}\ (ip, p)$$

Verifying an addition service

```
let (n, m) = deserialize $serialize(x, y)$ in
let res = serialize (n + m) in
sendto $h$ res from$(m)$
```

$$\Psi : \textit{Message} \to \textit{iProp}$$

$$x, y : \mathbb{N}$$

$$(ip, p) \Mapsto \Phi_{add}$$

$$h \hookrightarrow \mathsf{Some}\ (ip, p)$$

$$\mathsf{from}(m) \Mapsto \Psi$$

$$\forall m'.\, \mathsf{body}(m') = serialize(x + y)$$

$$-\!\!*\ \Psi(m')$$

Verifying an addition service

$\texttt{sendto}\ h\ \ serialize(x+y)\ \ \mathsf{from}(m)$

$\Psi : \textit{Message} \to \textit{iProp}$

$x, y : \mathbb{N}$

$(ip, p) \Mapsto \Phi_{add}$

$h \hookrightarrow \mathsf{Some}\ (ip, p)$

$\mathsf{from}(m) \Mapsto \Psi$

$\forall m'.\, \mathsf{body}(m') = serialize(x+y)$

$\quad -\!\!\ast\ \Psi(m')$

Verifying a client

$$\{\, srv \Mapsto \Phi_{add} * \mathsf{FreePorts}(ip, \{p\}) \,\}$$

$$\quad \texttt{client}\ x\ y\ srv\ (ip, p)$$

$$\{v. v = x + y\}$$

Verifying a client

```
let skt = socket () in
socketbind skt (ip, p);
let m = serialize (x, y) in
sendto skt m srv;
let res = listenwait skt in
deserialize res
```

$$\{v. v = x + y\}$$

$$srv \Mapsto \Phi_{add}$$

$$\text{FreePorts}(ip, \{p\})$$

Verifying a client

$$srv \Mapsto \Phi_{add}$$

$$h \hookrightarrow \mathsf{Some}\ (ip, p)$$

```
let m = serialize (x, y) in
sendto h m srv;
let res = listenwait h in
deserialize res
```

$$\{v.v = x + y\}$$

Verifying a client

$$\begin{aligned}\Phi_{add}(m) \triangleq \exists \Psi : \mathit{Message} \to \mathit{iProp}, (x, y : \mathbb{N}).\ & \mathsf{body}(m) = \mathit{serialize}(x, y) \ast \\ & \mathsf{from}(m) \Mapsto \Psi \ast \\ & \forall m'.\ \mathsf{body}(m') = \mathit{serialize}(x + y) \mathrel{-\!\!\ast} \Psi(m')\end{aligned}$$

```
sendto h serialize(x, y) srv;
let res = listenwait h in
deserialize res
```

$$\{v. v = x + y\}$$

$$srv \Mapsto \Phi_{add}$$

$$h \hookrightarrow \mathsf{Some}\ (ip, p)$$

Verifying a client

$$\begin{aligned}\Phi_{add}(m) \triangleq\; & \exists \Psi : \textit{Message} \to \textit{iProp}, (x, y : \mathbb{N}).\\ & \mathsf{body}(m) = \textit{serialize}(x, y) * \\ & \mathsf{from}(m) \Mapsto \Psi * \\ & \forall m'.\, \mathsf{body}(m') = \textit{serialize}(x + y) \mathrel{-\!\!*} \Psi(m')\end{aligned}$$

```
sendto h serialize(x, y) srv;
let res = listenwait h in
deserialize res
```

$$\{v. v = x + y\}$$

$$(ip, p) \Mapsto \Phi_{client}$$

$$srv \Mapsto \Phi_{add}$$

$$h \hookrightarrow \mathsf{Some}\ (ip, p)$$

$$\Phi_{client}(m) \triangleq \mathsf{body}(m) = \textit{serialize}(x + y)$$

Verifying a client

```
let res = listenwait h in
deserialize res
```

$$\{v.v = x + y\}$$

$$(ip, p) \Mapsto \Phi_{client}$$

$$srv \Mapsto \Phi_{add}$$

$$h \hookrightarrow \mathsf{Some}\ (ip, p)$$

$$\Phi_{client}(m) \triangleq \mathsf{body}(m) = serialize(x + y)$$

Verifying a client

$$\{\Phi_{client}(m)\}$$

```
let res = body(m) in
deserialize res
```

$$\{v.v = x + y\}$$

$$(ip, p) \Mapsto \Phi_{client}$$

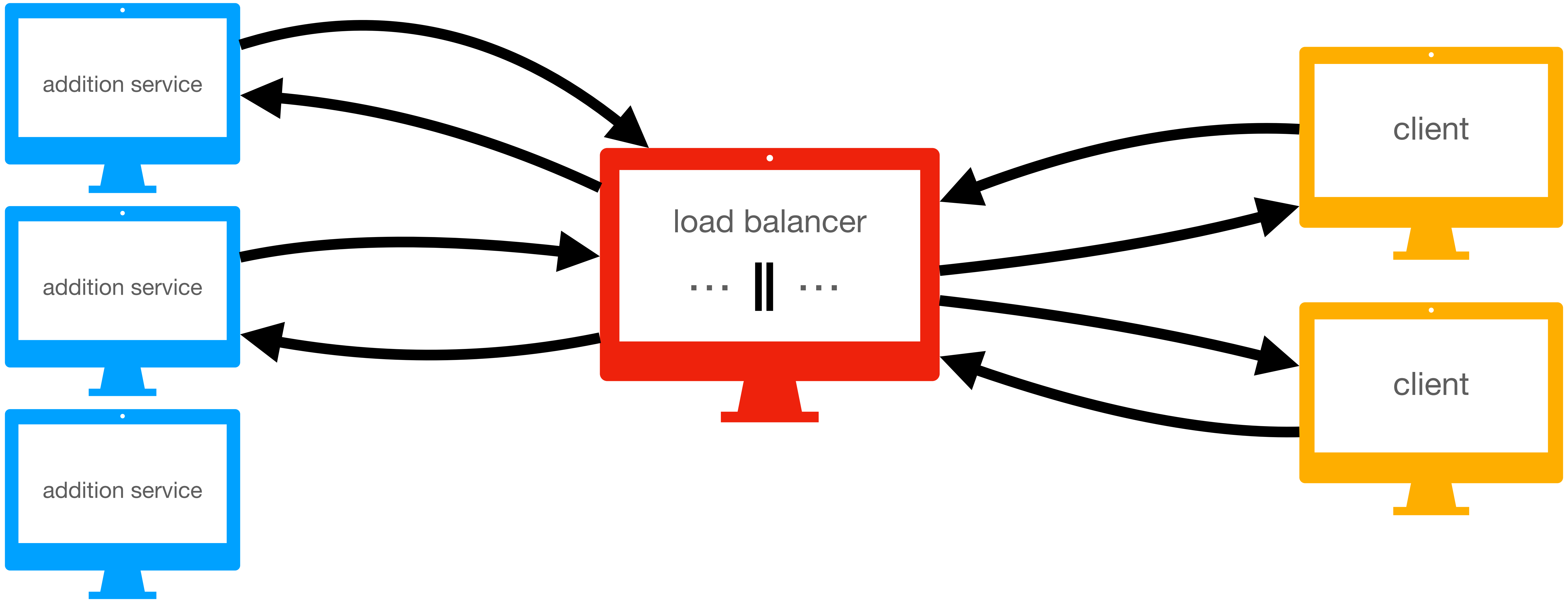
$$srv \Mapsto \Phi_{add}$$

$$h \hookrightarrow \mathsf{Some}\ (ip, p)$$

$$\Phi_{client}(m) \triangleq \mathsf{body}(m) = serialize(x + y)$$

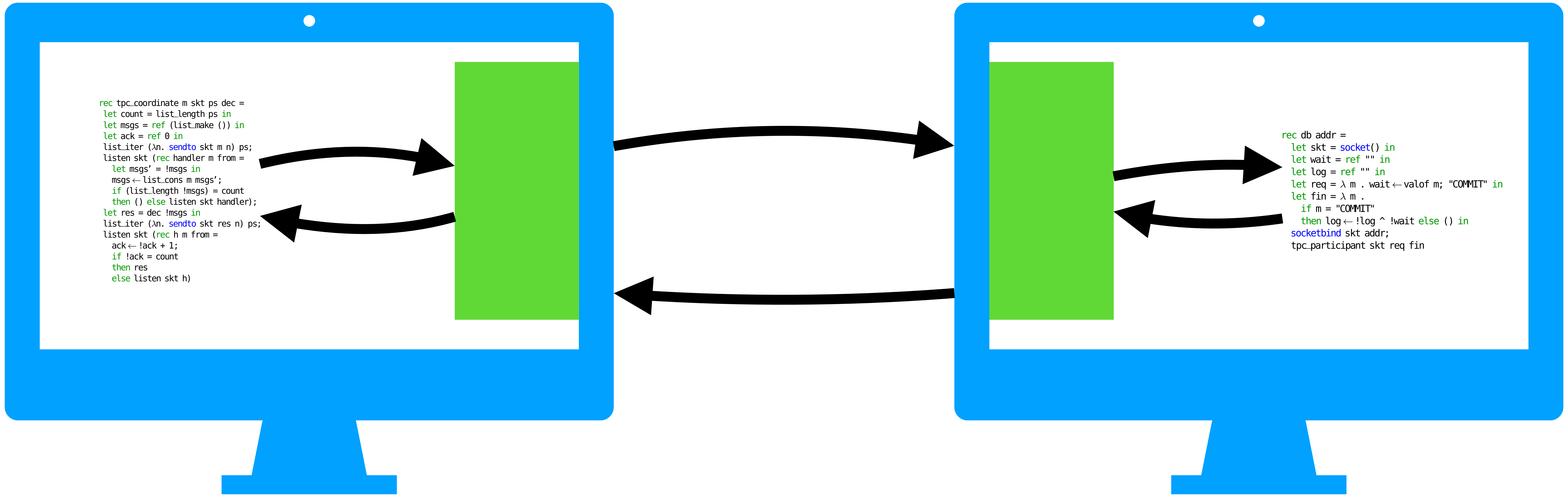
Horizontal composition

- The addition service showcases **horizontal compositional reasoning**
- In the paper we also introduce a load balancer, but use the existing specifications for the server and client to verify the system



Vertical composition

- In the paper we showcase **vertical compositional reasoning** by implementing and verifying the two-phase commit protocol
- We use the two-phase commit implementation and specification as a **library** to implement and verify a replicated logging system



```
rec tpc_coordinate m skt ps dec =
 let count = list_length ps in
 let msgs = ref (list_make ()) in
 let ack = ref 0 in
 list_iter (λn. sendto skt m n) ps;
 listen skt (rec handler m from =
   let msgs' = !msgs in
   msgs ← list_cons m msgs';
   if (list_length !msgs) = count
   then () else listen skt handler);
 let res = dec !msgs in
 list_iter (λn. sendto skt res n) ps;
 listen skt (rec h m from =
   ack ← !ack + 1;
   if !ack = count
   then res
   else listen skt h)
```

```
rec db addr =
  let skt = socket() in
  let wait = ref "" in
  let log = ref "" in
  let req = λ m . wait ← valof m; "COMMIT" in
  let fin = λ m .
    if m = "COMMIT"
    then log ← !log ^ !wait else () in
  socketbind skt addr;
  tpc_participant skt req fin
```

Distributed Causal Memory @ POPL'21

Modular Specification and Verification in Higher-Order Distributed Separation Logic

An implementation, modular specification, and verification of a causally-consistent distributed database and its clients in Aneris.

Léon Gondelman, Simon Oddershede Gregersen, Abel Nieto, Amin Timany, and Lars Birkedal. 2021. Distributed Causal Memory: Modular Specification and Verification in Higher-Order Distributed Separation Logic. *Proc. ACM Program. Lang.* 5, POPL, Article 42 (January 2021), 29 pages. https://doi.org/10.1145/3434323

Distributed Causal Memory: Modular Specification and Verification in Higher-Order Distributed Separation Logic

42

LÉON GONDELMAN, Aarhus University, Denmark
SIMON ODDERSHEDE GREGERSEN, Aarhus University, Denmark
ABEL NIETO, Aarhus University, Denmark
AMIN TIMANY, Aarhus University, Denmark
LARS BIRKEDAL, Aarhus University, Denmark

We present the first specification and verification of an implementation of a causally-consistent distributed database that supports modular verification of full functional correctness properties of clients and servers. We specify and reason about the causally-consistent distributed database in Aneris, a higher-order distributed separation logic for an ML-like programming language with network primitives for programming distributed systems. We demonstrate that our specifications are useful, by proving the correctness of small, but tricky, synthetic examples involving causal dependency and by verifying a session manager library implemented on top of the distributed database. We use Aneris's facilities for modular specification and verification to obtain a highly modular development, where each component is verified in isolation, relying only on the specifications (not the implementations) of other components. We have used the Coq formalization of the Aneris logic to formalize all the results presented in the paper in the Coq proof assistant.

CCS Concepts: • **Theory of computation → Program verification; Distributed algorithms; Separation logic**.

Additional Key Words and Phrases: Distributed systems, causal consistency, separation logic, higher-order logic, concurrency, formal verification

ACM Reference Format:
Léon Gondelman, Simon Oddershede Gregersen, Abel Nieto, Amin Timany, and Lars Birkedal. 2021. Distributed Causal Memory: Modular Specification and Verification in Higher-Order Distributed Separation Logic. *Proc. ACM Program. Lang.* 5, POPL, Article 42 (January 2021), 29 pages. https://doi.org/10.1145/3434323

1 INTRODUCTION

The ubiquitous distributed systems of the present day internet often require highly available and scalable distributed data storage solutions. The CAP theorem [Gilbert and Lynch 2002] states that a distributed database cannot at the same time provide *consistency*, *availability* and *partition (failure) tolerance*. Hence, many such systems choose to sacrifice aspects of data consistency for the sake of availability and fault tolerance [Bailis et al. 2013; Chang et al. 2008; Lloyd et al. 2011; Tyulenev et al. 2019]. In these systems different replicas of the database may, at the same point in time, observe different, inconsistent data. Among different notions of weaker consistency guarantees, a popular one is *causal consistency*. With causal consistency different replicas can observe different data, yet, it is guaranteed that data are observed in a causally related order: if a node *n* observes an operation

Authors' addresses: Léon Gondelman, Aarhus University, Denmark, gondelman@cs.au.dk; Simon Oddershede Gregersen, Aarhus University, Denmark, gregersen@cs.au.dk; Abel Nieto, Aarhus University, Denmark, abeln@cs.au.dk; Amin Timany, Aarhus University, Denmark, timany@cs.au.dk; Lars Birkedal, Aarhus University, Denmark, birkedal@cs.au.dk.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2021 Copyright held by the owner/author(s).
2475-1421/2021/1-ART42
https://doi.org/10.1145/3434323

Proc. ACM Program. Lang., Vol. 5, No. POPL, Article 42. Publication date: January 2021.

Conclusion

- A programming language and a separation logic, **Aneris,** for modular reasoning about partial correctness properties of **implementations** of distributed systems
- Scalable development and verification of distributed systems through **vertical and horizontal compositional reasoning**
- Extensive case studies showcasing our reasoning principles
- All theory and examples are mechanised on top of the Iris separation logic framework in the Coq proof assistant

Thank you for watching

Contact	**gregersen@cs.au.dk**
Paper	https://cs.au.dk/~gregersen/papers/2020-esop-aneris-final.pdf
Coq development	https://github.com/logsem/aneris