Message Chains for Distributed System Verification

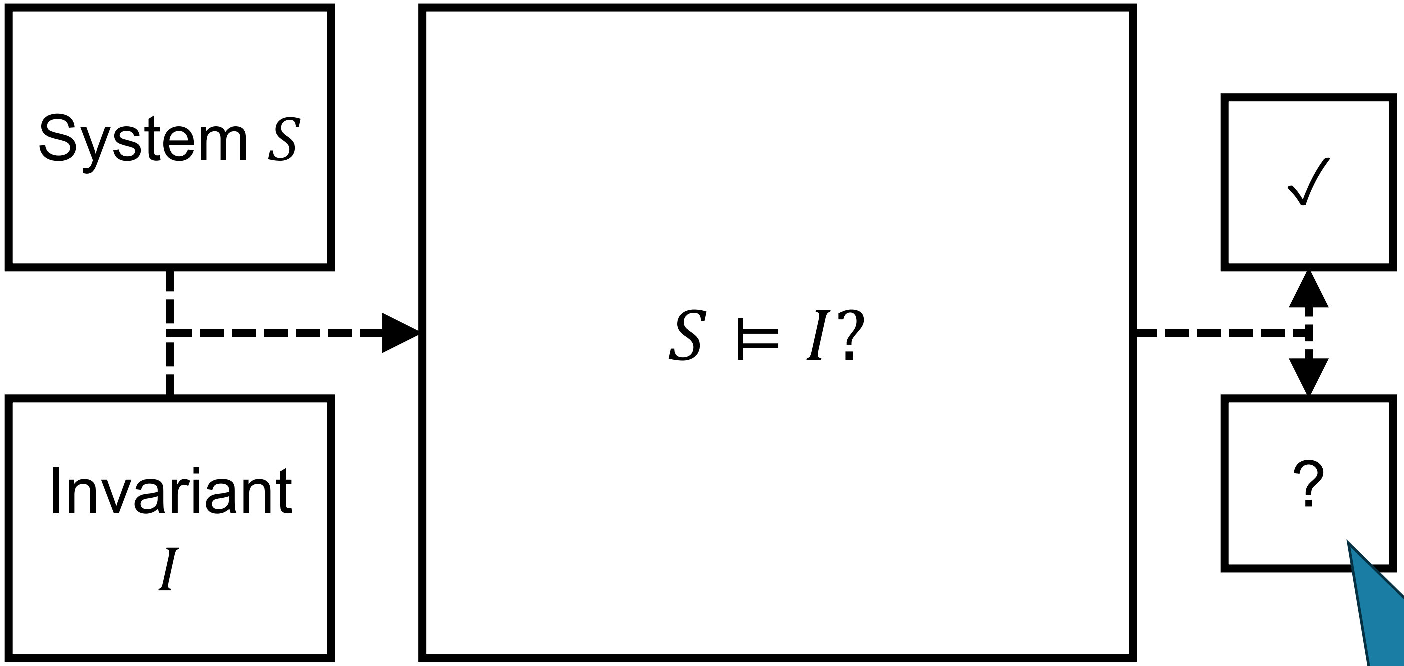
FEDERICO MORA, University of California, Berkeley
ANKUSH DESAI, Amazon Web Services
ELIZABETH POLGREEN, University of Edinburgh
SANJIT A. SESHIA, University of California, Berkeley

“Distributed bugs... are often severe... Not only are these outages widespread and expensive, they can be caused by bugs that were deployed to production months earlier... [For example] [amazon.com] went down because one remote server couldn't display any product information.”

- Jacob Gabrielson, Senior Principal Engineer at Amazon Web Services

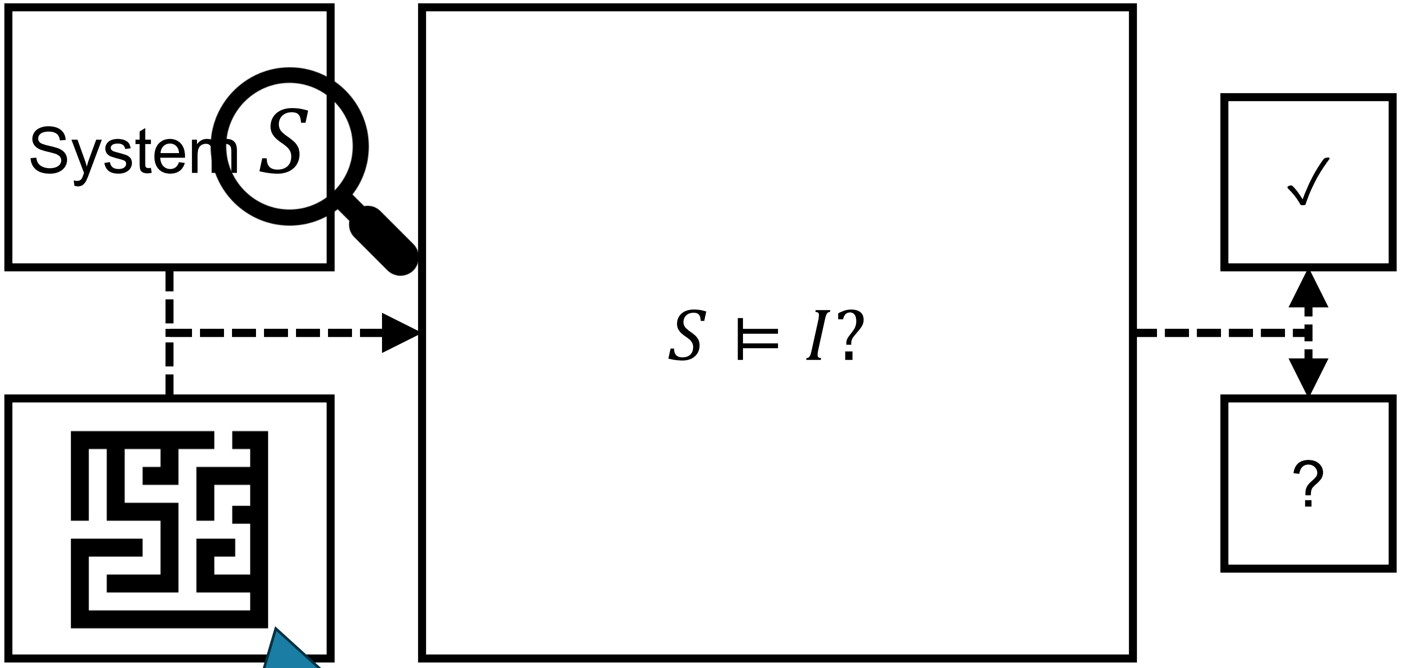
https://aws.amazon.com/builders-library/challenges-with-distributed-systems/

Formal Verification To The Rescue!



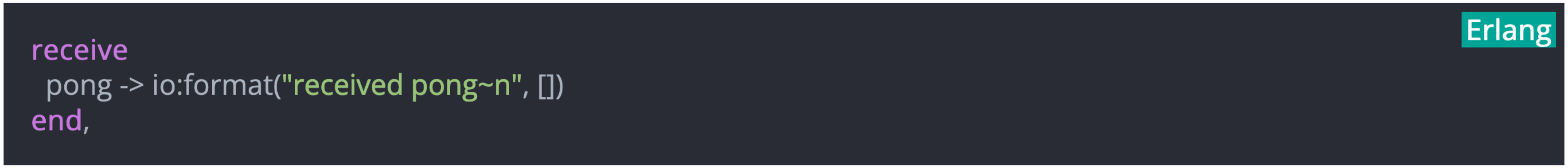
Usually we must iterate, making updates to S or I

More Detailed S; More Complicated I

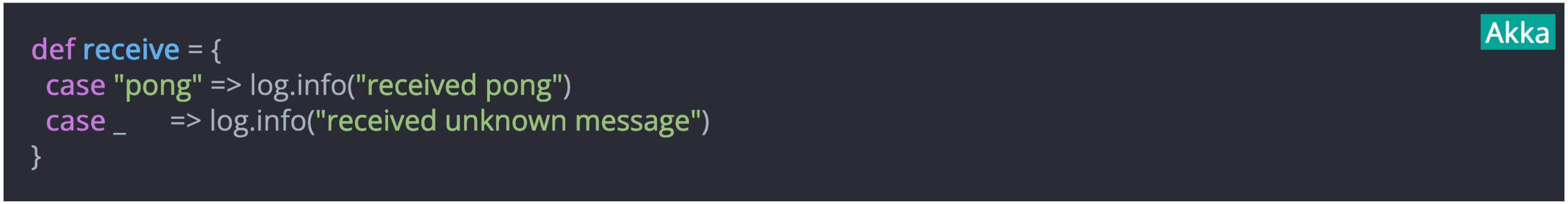


I becomes a maze!

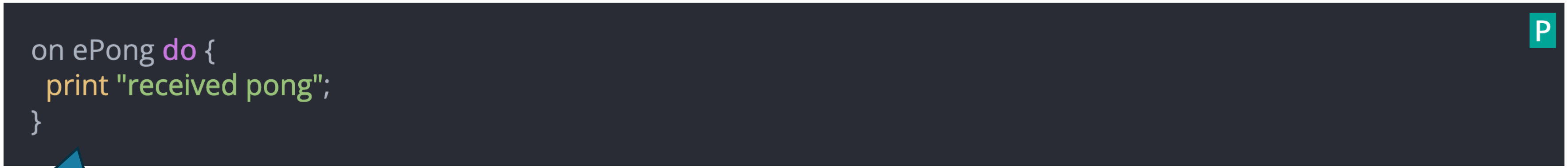
Goal: Approach Implementation Level System Detail



```
receive
  pong -> io:format("received pong~n", [])
end,
```



```
def receive = {
  case "pong" => log.info("received pong")
  case _      => log.info("received unknown message")
}
```



```
on ePong do {
  print "received pong";
}
```

These are called event handlers

Contributions Overview

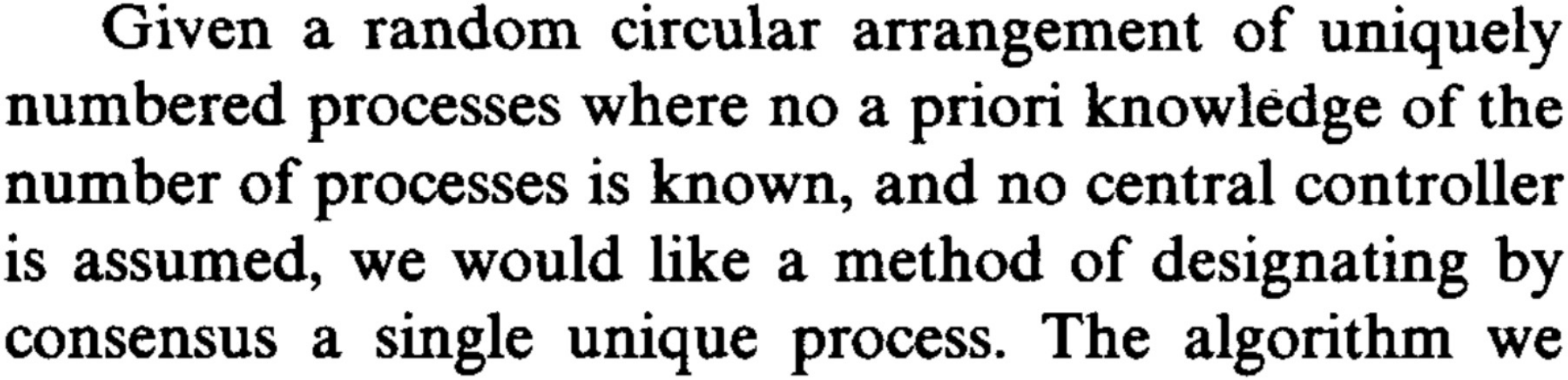
- We formally define
  - message chains and
  - message chain invariants.
- We show how to use them to
  - verify asynchronous, message-passing systems
  - while avoiding
    - reasoning about individual messages or buffers,
    - manually abstracting message passing, and
    - manually relating abstractions to messaging details.
- Present the UPVerifier

Running Example:

Chang And Roberts Algorithm For Decentralized Extrema-finding In Circular Configurations Of Processes

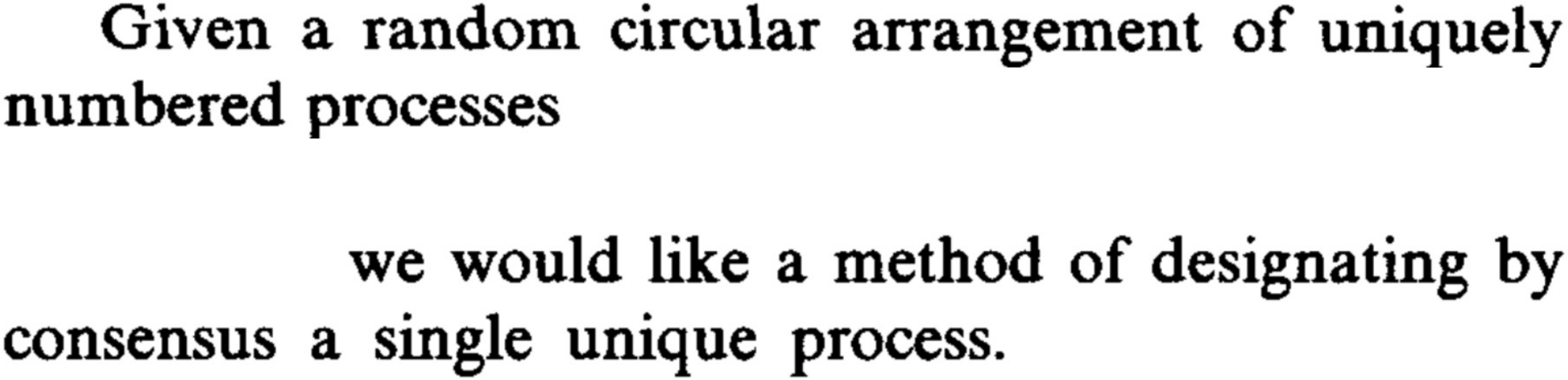
Chang and Roberts (1979)

Chang and Roberts Setting and Goal



Given a random circular arrangement of uniquely numbered processes where no a priori knowledge of the number of processes is known, and no central controller is assumed, we would like a method of designating by consensus a single unique process. The algorithm we

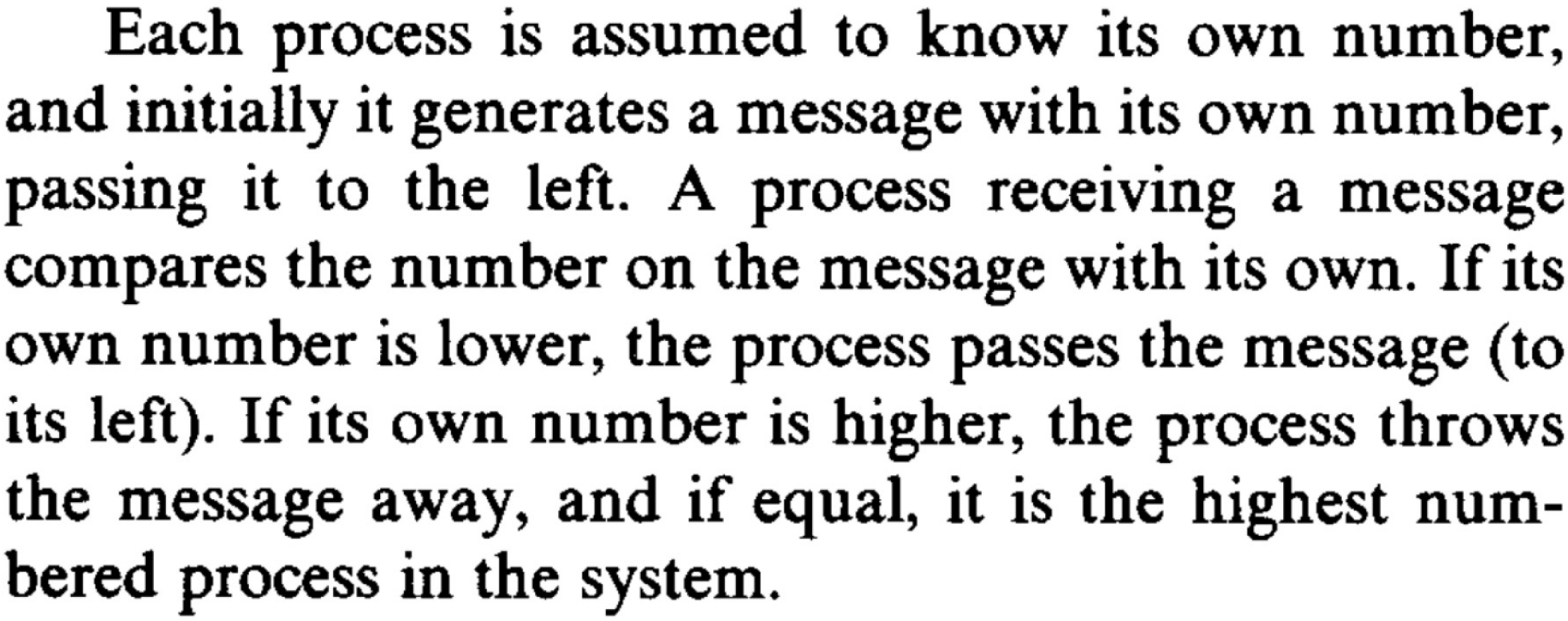
Chang and Roberts Setting and Goal



Given a random circular arrangement of uniquely numbered processes

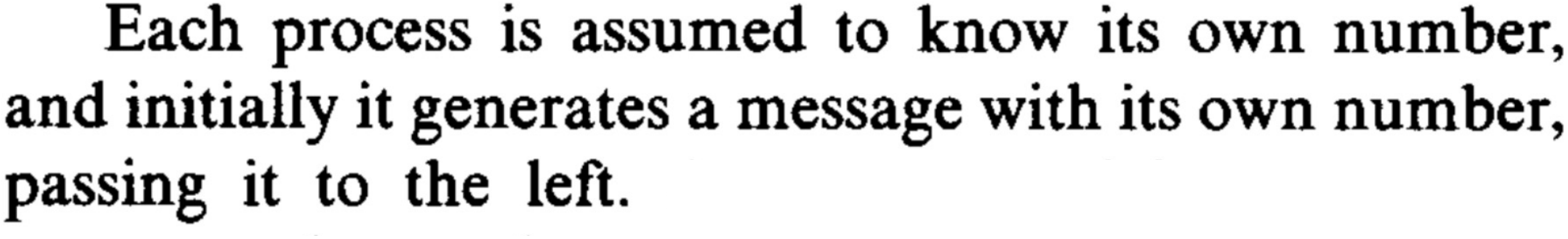
we would like a method of designating by consensus a single unique process.

Chang and Roberts Algorithm



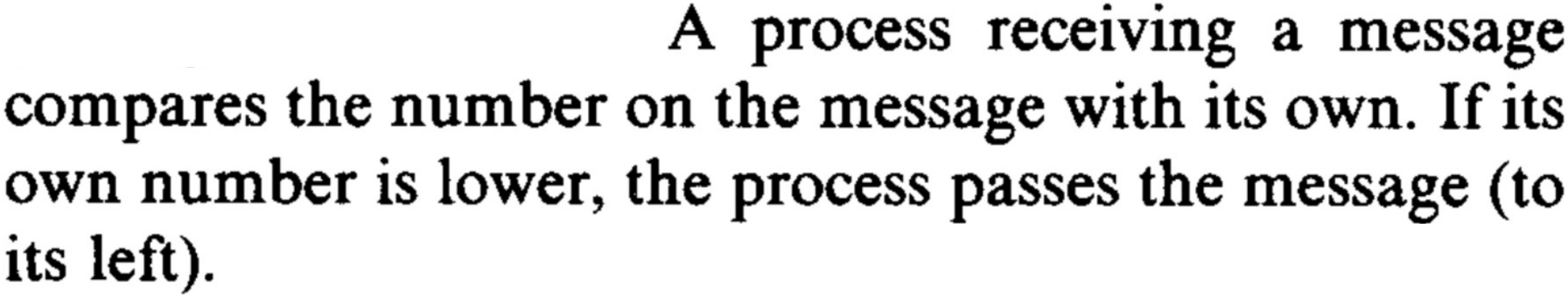
Each process is assumed to know its own number, and initially it generates a message with its own number, passing it to the left. A process receiving a message compares the number on the message with its own. If its own number is lower, the process passes the message (to its left). If its own number is higher, the process throws the message away, and if equal, it is the highest numbered process in the system.

Chang and Roberts Algorithm



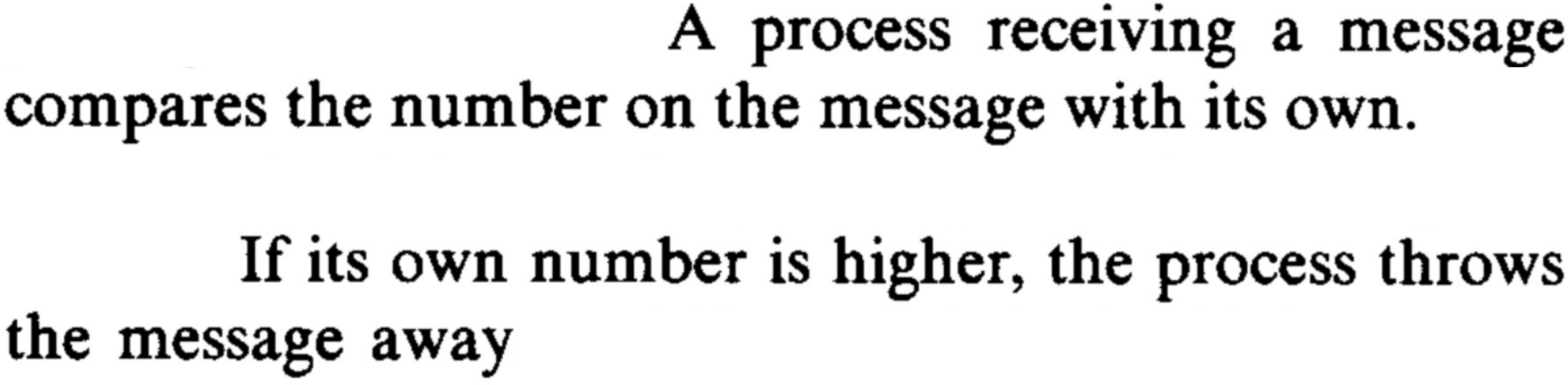
Each process is assumed to know its own number, and initially it generates a message with its own number, passing it to the left.

Chang and Roberts Algorithm



A process receiving a message compares the number on the message with its own. If its own number is lower, the process passes the message (to its left).

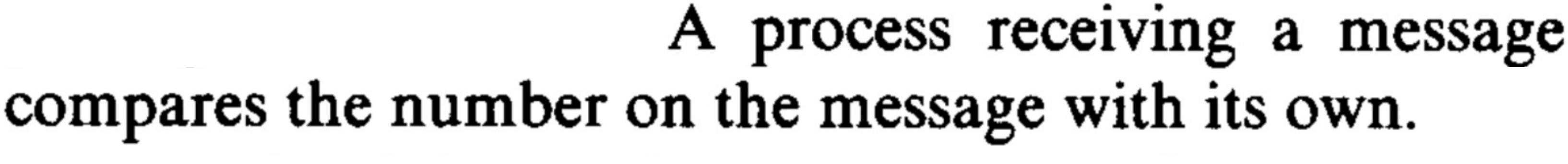
Chang and Roberts Algorithm



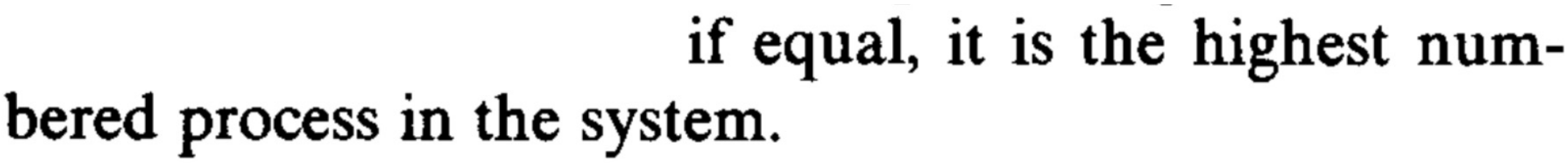
A process receiving a message compares the number on the message with its own.

If its own number is higher, the process throws the message away

Chang and Roberts Algorithm

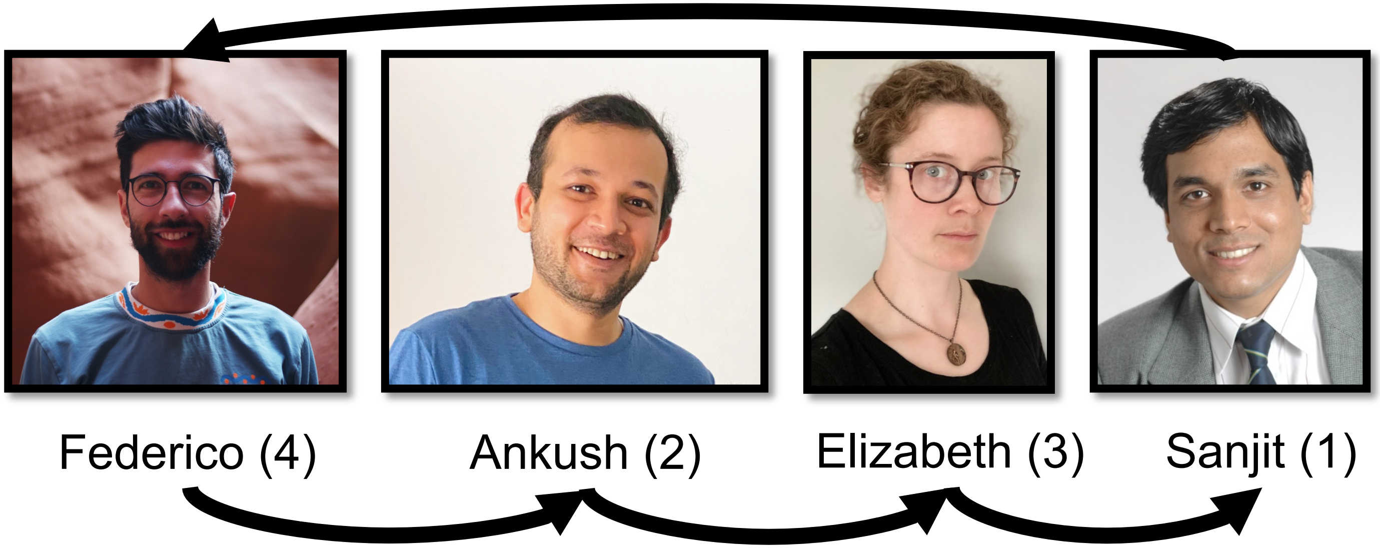


A process receiving a message compares the number on the message with its own.



if equal, it is the highest numbered process in the system.

Rebuttal Election Example

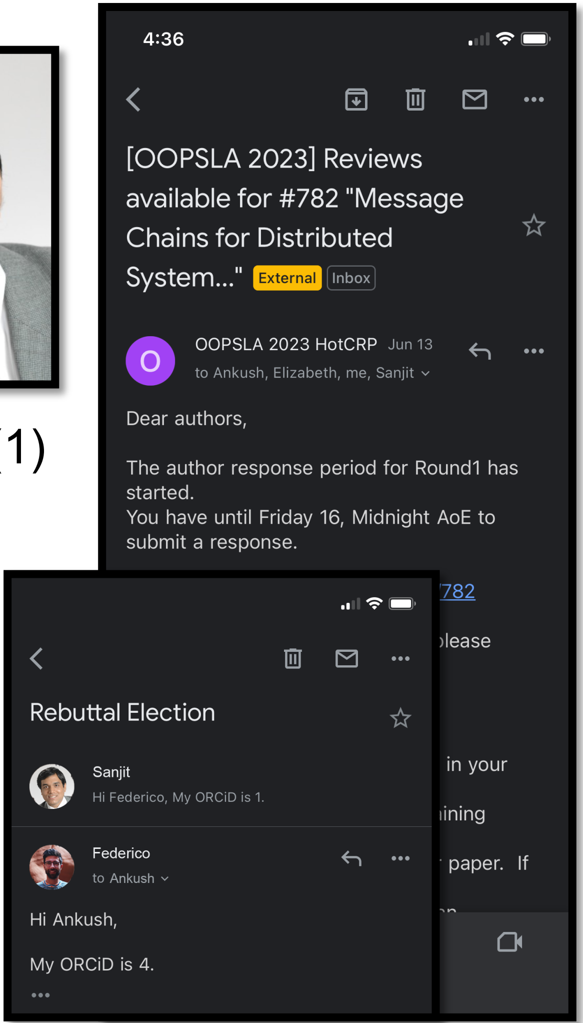


Federico (4)

Ankush (2)

Elizabeth (3)

Sanjit (1)



[OOPSLA 2023] Reviews available for #782 "Message Chains for Distributed System..." External Inbox

OOPSLA 2023 HotCRP Jun 13
to Ankush, Elizabeth, me, Sanjit

Dear authors,

The author response period for Round1 has started.
You have until Friday 16, Midnight AoE to submit a response.

782

please

in your

ining

paper. If

Rebuttal Election

Sanjit
Hi Federico, My ORCiD is 1.

Federico
to Ankush

Hi Ankush,

My ORCiD is 4.

Who Will Draft The Rebuttal?

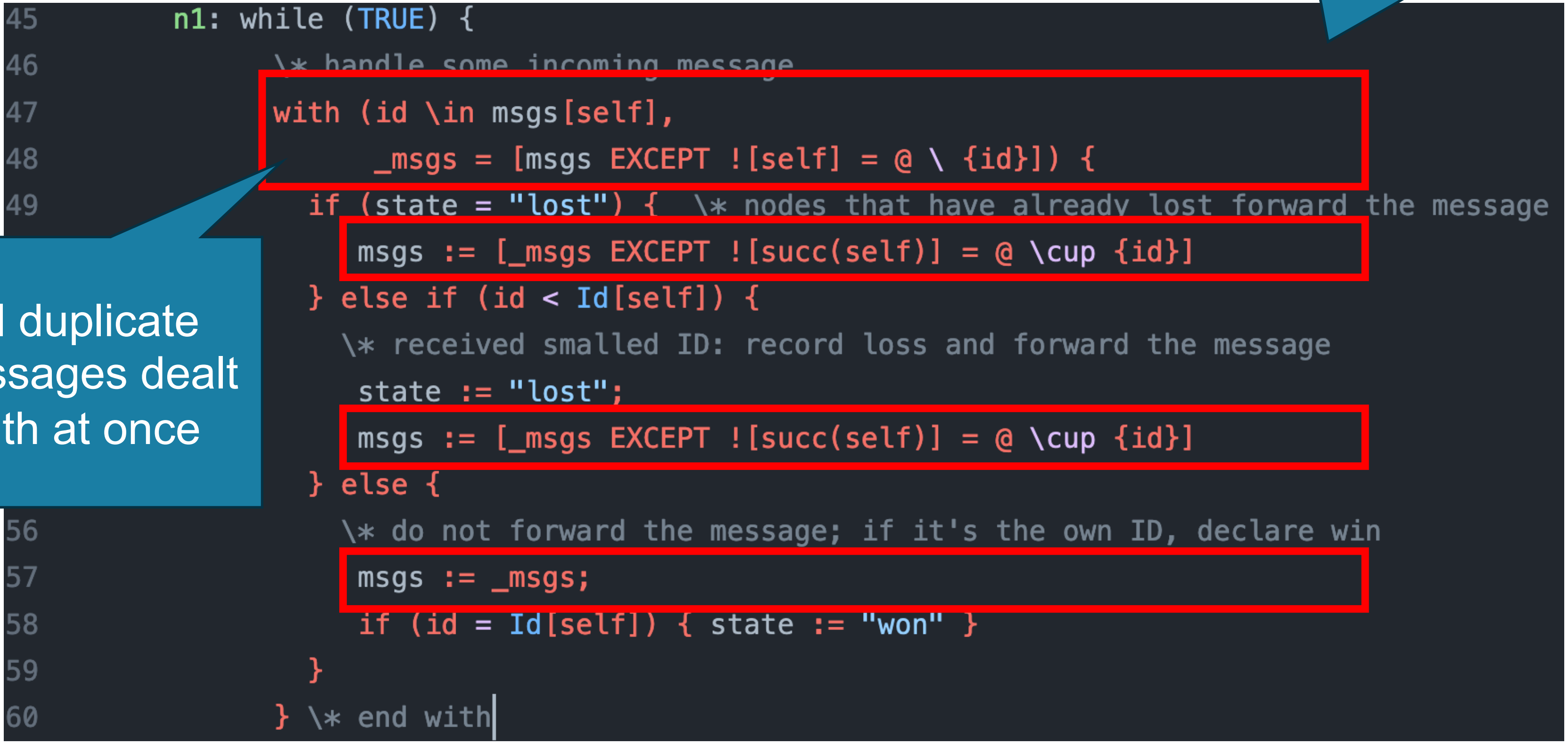
1. ORCiD gives unique ID
2. Author list gives circle arrangement
3. Email gives message passing

Example System Modelling:

Chang and Roberts Algorithm

Chang and Roberts in TLA+

“msgs” is a global shared memory variable that they manually maintain

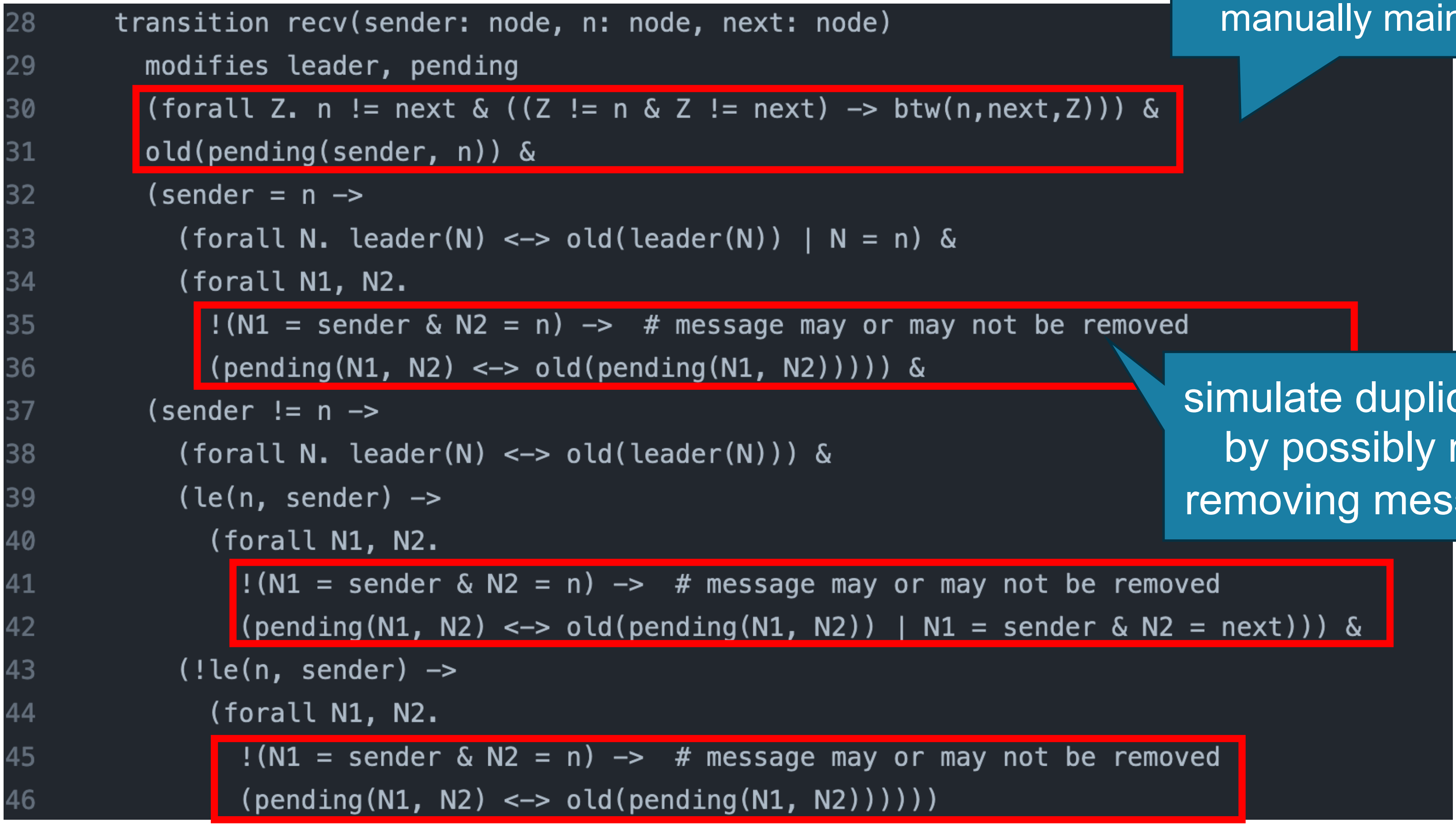


```
n1: while (TRUE) {
      \* handle some incoming message
      with (id \in msgs[self],
            _msgs = [msgs EXCEPT ![self] = @ \ {id}]) {
        if (state = "lost") {  \* nodes that have already lost forward the message
          msgs := [_msgs EXCEPT ![succ(self)] = @ \cup {id}]
        } else if (id < Id[self]) {
          \* received smalled ID: record loss and forward the message
          state := "lost";
          msgs := [_msgs EXCEPT ![succ(self)] = @ \cup {id}]
        } else {
          \* do not forward the message; if it's the own ID, declare win
          msgs := _msgs;
          if (id = Id[self]) { state := "won" }
        }
      } \* end with
```

all duplicate messages dealt with at once

https://github.com/tlaplus/Examples/blob/master/specifications/chang_roberts/ChangRoberts.tla

Chang and Roberts in mypyvy



```
transition recv(sender: node, n: node, next: node)
 modifies leader, pending
 (forall Z. n != next & ((Z != n & Z != next) -> btw(n,next,Z))) &
 old(pending(sender, n)) &
 (sender = n ->
  (forall N. leader(N) <-> old(leader(N)) | N = n) &
  (forall N1, N2.
   !(N1 = sender & N2 = n) ->  # message may or may not be removed
   (pending(N1, N2) <-> old(pending(N1, N2))))) &
 (sender != n ->
  (forall N. leader(N) <-> old(leader(N))) &
  (le(n, sender) ->
   (forall N1, N2.
    !(N1 = sender & N2 = n) ->  # message may or may not be removed
    (pending(N1, N2) <-> old(pending(N1, N2)) | N1 = sender & N2 = next))) &
  (!le(n, sender) ->
   (forall N1, N2.
    !(N1 = sender & N2 = n) ->  # message may or may not be removed
    (pending(N1, N2) <-> old(pending(N1, N2))))))
```

“pending” is a global shared memory variable that they manually maintain

simulate duplicates by possibly not removing message

https://github.com/wilcoxjay/mypyvy/blob/pldi20-artifact/examples/pd/ring.pyv

Other Choices, Same Story

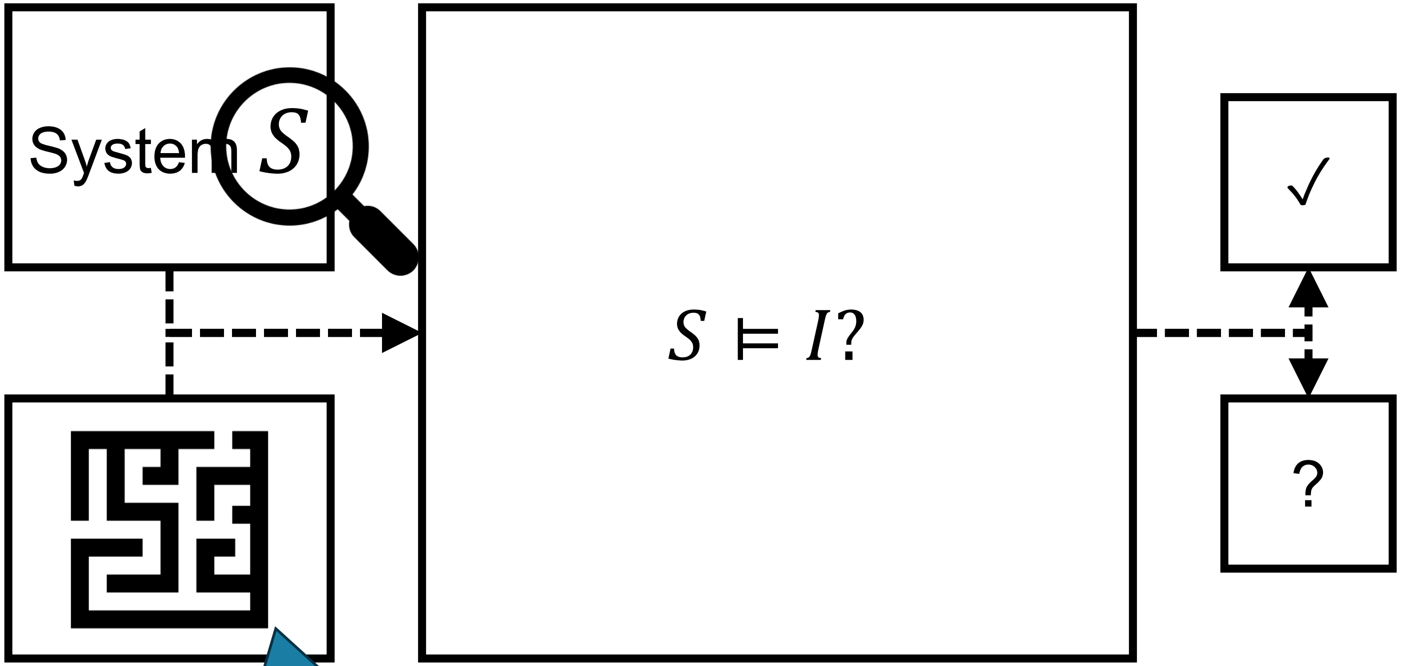
- Domain Specific Tools: TLA+, mypyvy, etc…
- Proof Assistants: Lean, Coq, etc…
- Software Verification Engines: Dafny, F*, etc…

- It is easier to write proofs about shared memory variables!

Chang and Roberts in the UPVerifier

```
on eNominate e do {
    let curr := e.payload.id in
    if curr = this then
        goto Won
    else if this <= curr then
        send left(this), eNominate(curr)
}
```

More Detailed S; More Complicated I



I becomes a maze!

Message Chains to the Rescue!

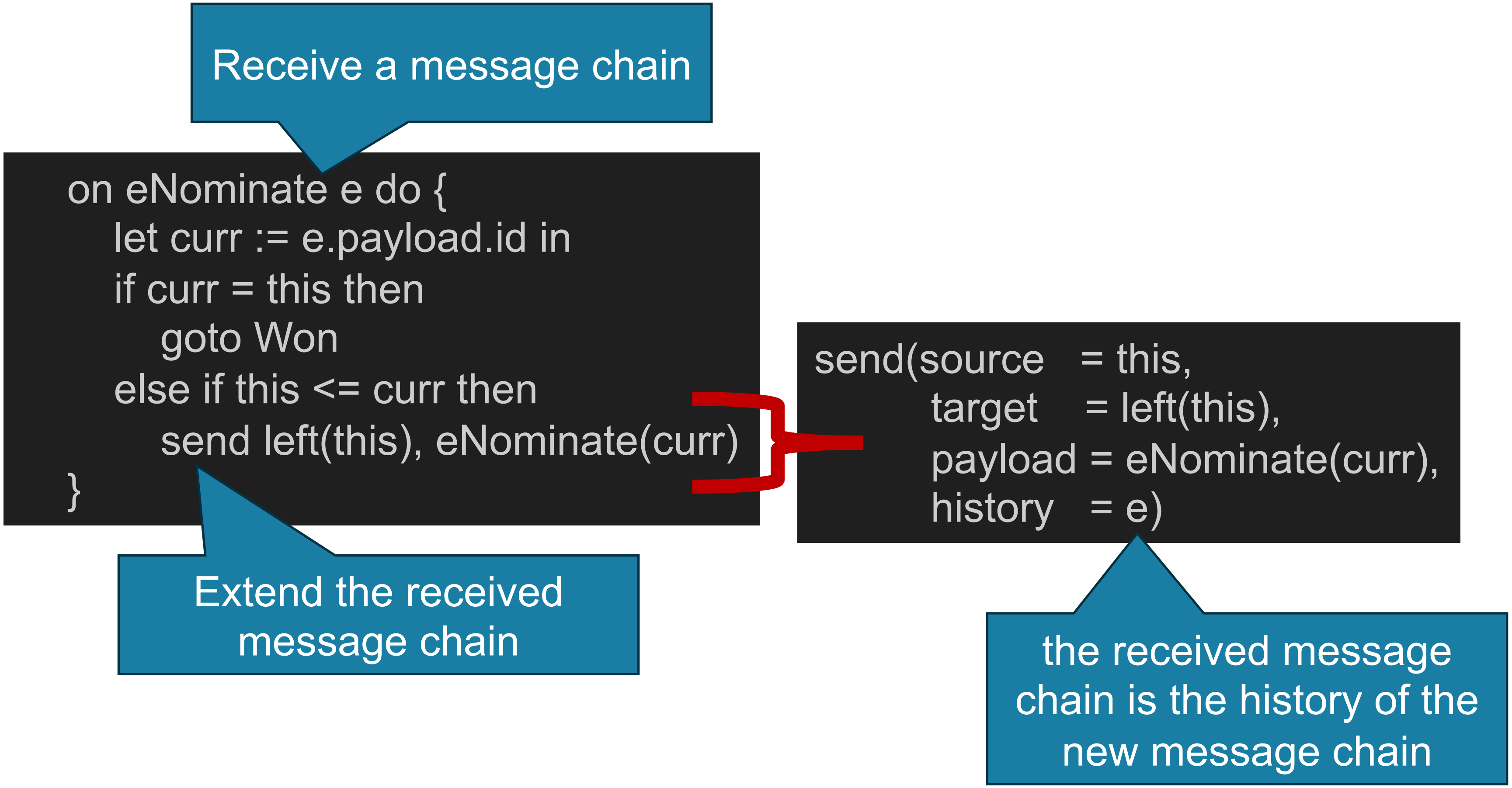
Message Chains Type,
Message Chain Semantics,
Message Chain Invariants,

Message Chain Datatype

```
data [node_t, message_t] message_chain :=
  | empty
  | send {source: node_t,
              target: node_t,
              payload: message_t,
              history: message_chain[node_t, message_t]}
```

Verification Semantics of Send

CASE 1/2: Send inside event handler



Receive a message chain

```
on eNominate e do {
    let curr := e.payload.id in
    if curr = this then
        goto Won
    else if this <= curr then
        send left(this), eNominate(curr)
}
```

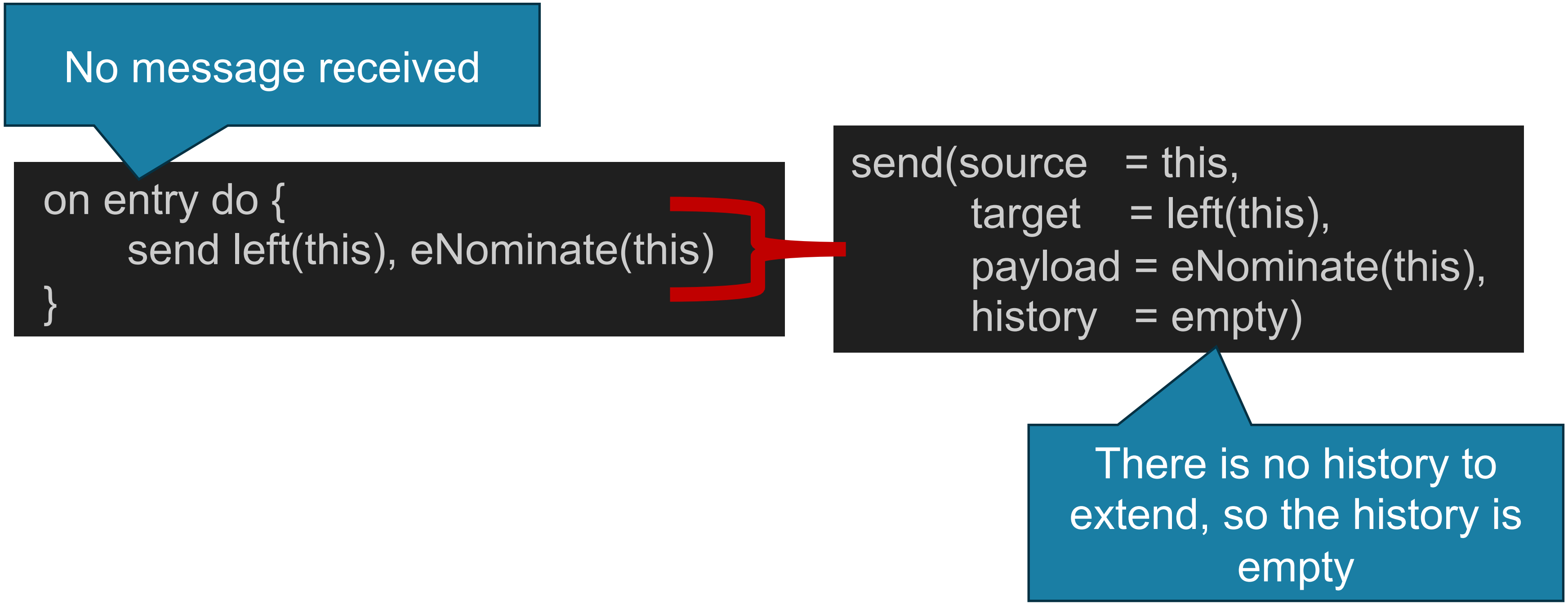
```
send(source   = this,
          target    = left(this),
          payload = eNominate(curr),
          history   = e)
```

Extend the received message chain

the received message chain is the history of the new message chain

Verification Semantics of Send

CASE 2/2: Send outside event handler



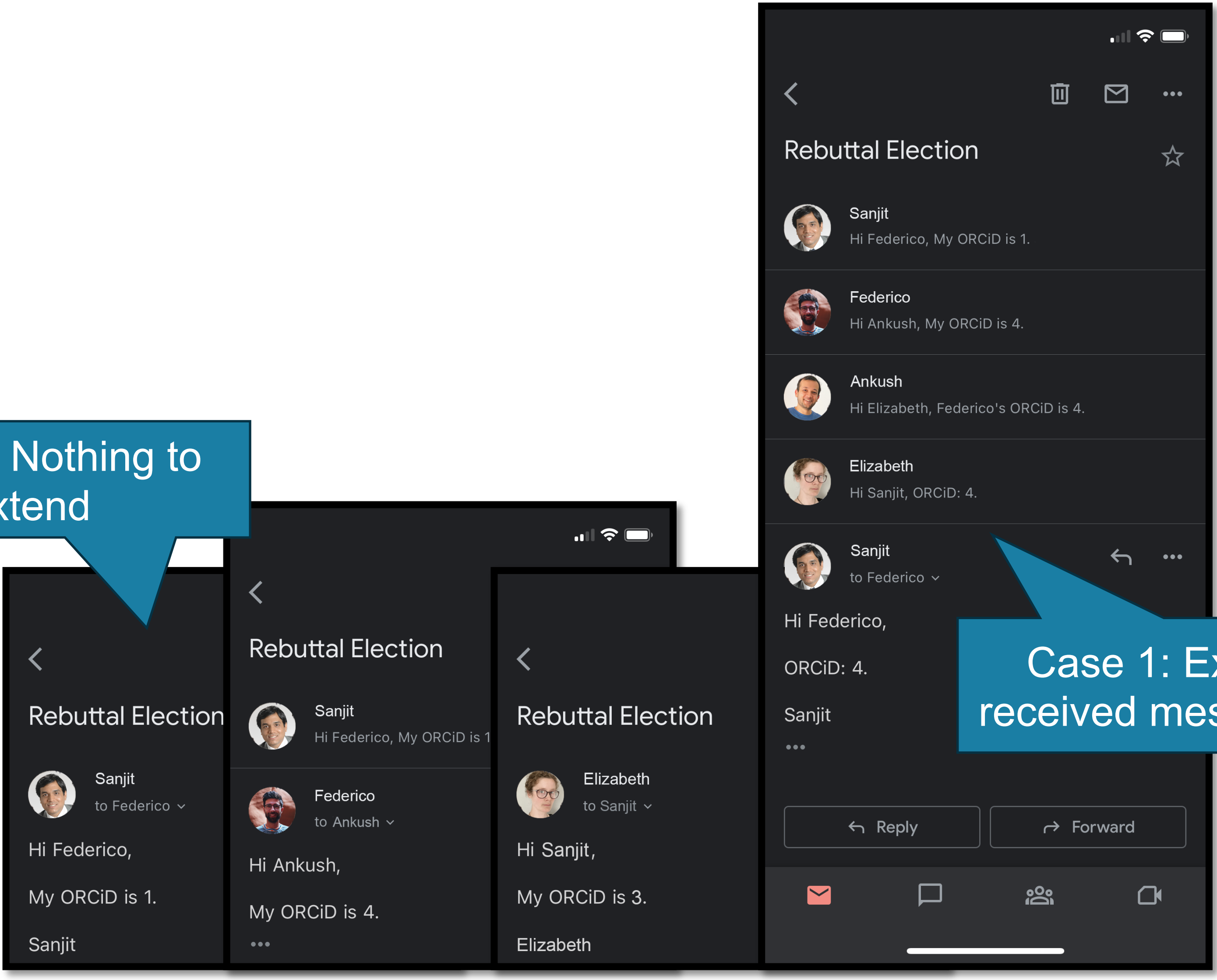
No message received

```
on entry do {
      send left(this), eNominate(this)
}
```

```
send(source   = this,
        target    = left(this),
        payload = eNominate(this),
        history   = empty)
```

There is no history to extend, so the history is empty

Example Message Chains



Case 2: Nothing to extend

Rebuttal Election

Sanjit
to Federico

Hi Federico,

My ORCiD is 1.

Sanjit

Rebuttal Election

Sanjit
Hi Federico, My ORCiD is 1

Federico
to Ankush

Hi Ankush,

My ORCiD is 4.

Rebuttal Election

Elizabeth
to Sanjit

Hi Sanjit,

My ORCiD is 3.

Elizabeth

Rebuttal Election

Sanjit
Hi Federico, My ORCiD is 1.

Federico
Hi Ankush, My ORCiD is 4.

Ankush
Hi Elizabeth, Federico's ORCiD is 4.

Elizabeth
Hi Sanjit, ORCiD: 4.

Sanjit
to Federico

Hi Federico,

ORCiD: 4.

Sanjit

Case 1: Extend the received message chain

Message Chain Invariants

s is the system, alive is true iff e is in a node’s message buffer

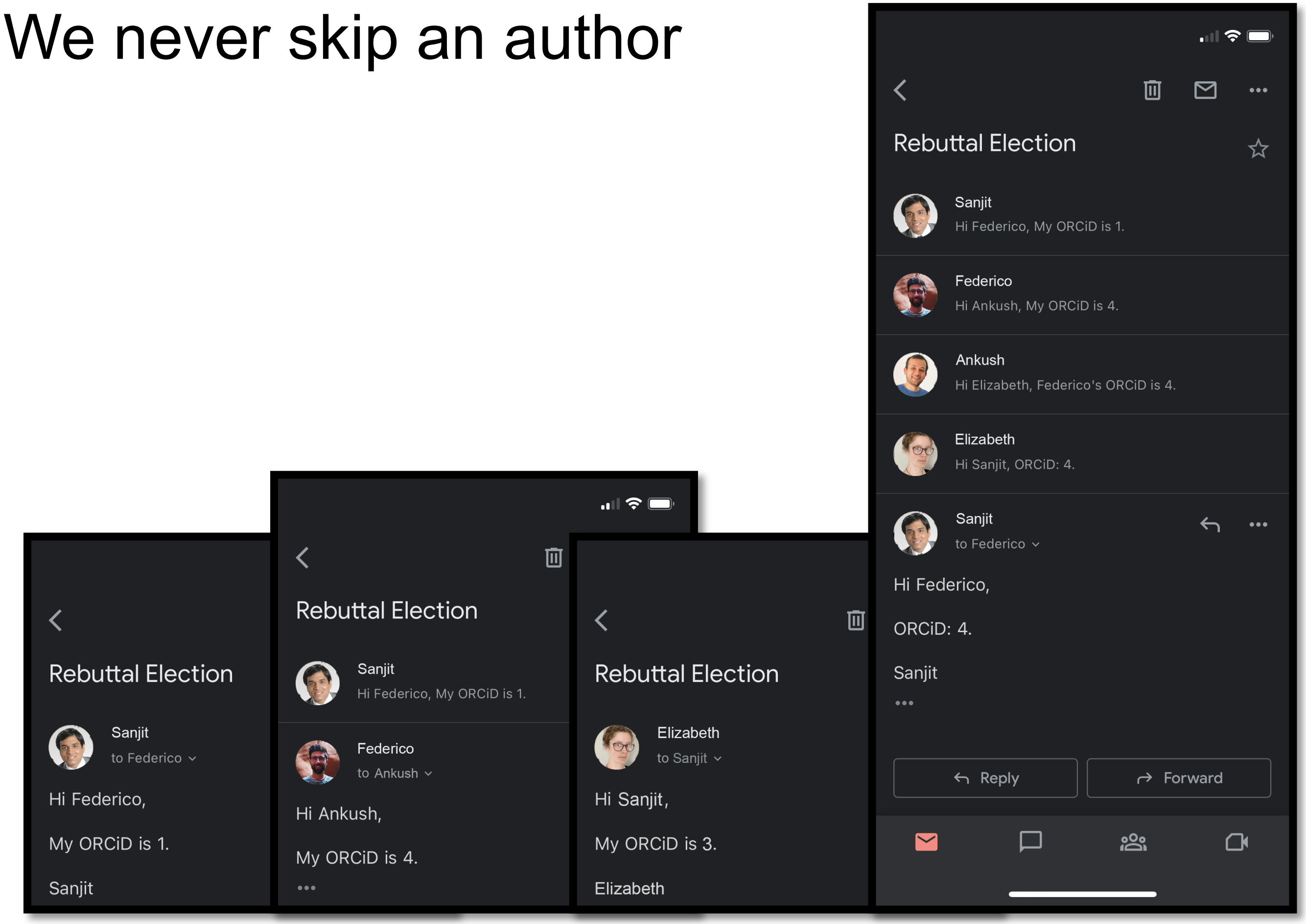
P is a predicate

```
forall (e: message_chain[node_t, message_t]) alive(s, e) ==> P(e)
```

Everything is fixed except for P: P defines the invariant.

Example Message Chain Invariants

1. Most recent email holds largest ORCiD
2. We never skip an author



Rebuttal Election

Sanjit
to Federico

Hi Federico,

My ORCiD is 1.

Sanjit

Rebuttal Election

Sanjit
Hi Federico, My ORCiD is 1.

Federico
to Ankush

Hi Ankush,

My ORCiD is 4.

Rebuttal Election

Elizabeth
to Sanjit

Hi Sanjit,

My ORCiD is 3.

Elizabeth

Rebuttal Election

Sanjit
Hi Federico, My ORCiD is 1.

Federico
Hi Ankush, My ORCiD is 4.

Ankush
Hi Elizabeth, Federico's ORCiD is 4.

Elizabeth
Hi Sanjit, ORCiD: 4.

Sanjit
to Federico

Hi Federico,

ORCiD: 4.

Sanjit

Example Verification With Message Chain Invariants:

Chang and Roberts Algorithm

Chang and Roberts Full Verification

1. There is at most one leader:

```
forall (l: node, n: node)   (leader(s, l) and leader(s, n)) ==> l = n
```

2. The leader has the largest ID:

```
forall (l: node, n: node)   leader(s, l) ==> n <= l
```

Two auxiliary message chain invariants: P(e) :=

1. The largest ID is always front:

```
forall (n: node)   participated(e, n) ==> n <= e.payload.id
```

2. If a node has not participated, then the node is
    - the next target, or
    - the next target comes before that node in the ring:

```
forall (n: node)   not participated(e, n) ==> between(e.payload.id, e.target, n) or e.target = n
```

Helper function that is true iff n is a source in e

Example Verification Analysis

- What does our proof guarantee?
  - At most one node will be elected no matter how many nodes are in the system.

- Why is our proof extra cool?
  - We never talk about individual messages or buffers,
  - invariants are (relatively) intuitive, and
  - abstraction is
    - created automatically and
    - automatically related to the implementation level detail

Specification Mining

Specification Mining Definition

- "... a machine learning approach to discovering formal specifications" that code likely obeys.
  - Input: observed program executions
  - Output: suggested invariant
- In our setting, we are interested in mining message chain invariants.
- Note: not auxiliary invariant inference, which requires a target specification.

Ernst (2000); Ammons, Bodík, and Larus (2002)

Specification Mining Example

```
[Empty] -> True,
[Send(N1, N2, 1, Empty)] -> True,
[Send(N2, N3, 2, Empty)] -> True,
[Send(N1, N2, 4, Send(N4, N1, 4, Empty))] -> True,
[Send(N4, N1, 3, Send(N3, N4, 2, Empty))] -> True,
[Send(N4, N1, 4, Send(N3, N4, 3, Send(N2, N3, 2, Empty)))] -> True,
[Send(N4, N1, 4, Send(N3, N4, 3, Send(N2, N3, 2, Send(N1, N2, 1,Empty))))] -> True,
[Send(N3, N4, 4, Send(N2, N3, 4, Send(N1, N2, 4, Send(N4, N1, 4,Empty))))] -> True,

[Send(N2, N1, 0, Send(N1, N2, 1, Empty))] -> False,
[Send(N1, N4, 3, Send(N3, N4, 4, Empty))] -> False,
[Send(N4, N3, 4, Send(N4, N1, 5, Send(N4, N2, 2, Empty)))] -> False,
```

```
P(e) := forall (n: node)   participated(e, n) ==> n <= e.payload.id
```

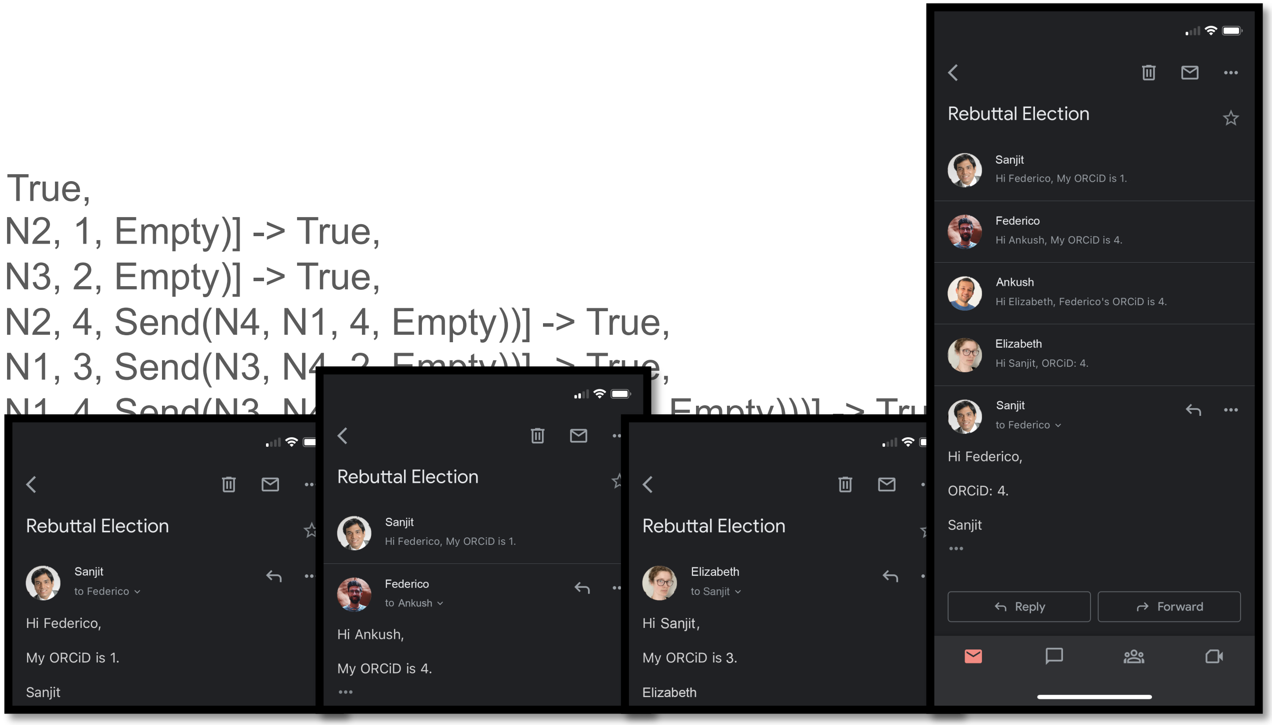
Actually, a recursive function that is equivalent to this

Lee and Cho (2023)

Specification Mining Example

```
[Empty] -> True,
[Send(N1, N2, 1, Empty)] -> True,
[Send(N2, N3, 2, Empty)] -> True,
[Send(N1, N2, 4, Send(N4, N1, 4, Empty))] -> True,
[Send(N4, N1, 3, Send(N3, N4, 2, Empty))] -> True,
[Send(N4, N1, 4, Send(N3, N4                 Empty)))] -> Tru
[Send(N4,
[Send(N3,

[Send(N2,
[Send(N1,
[Send(N4,
```



Rebuttal Election

Sanjit
to Federico

Hi Federico,

My ORCiD is 1.

Sanjit

Rebuttal Election

Sanjit
Hi Federico, My ORCiD is 1.

Federico
to Ankush

Hi Ankush,

My ORCiD is 4.

Rebuttal Election

Elizabeth
to Sanjit

Hi Sanjit,

My ORCiD is 3.

Elizabeth

Rebuttal Election

Sanjit
Hi Federico, My ORCiD is 1.

Federico
Hi Ankush, My ORCiD is 4.

Ankush
Hi Elizabeth, Federico's ORCiD is 4.

Elizabeth
Hi Sanjit, ORCiD: 4.

Sanjit
to Federico

Hi Federico,

ORCiD: 4.

Sanjit

P(e) := forall (n: node)   participated(e, n) ==> n <= e.payload.id

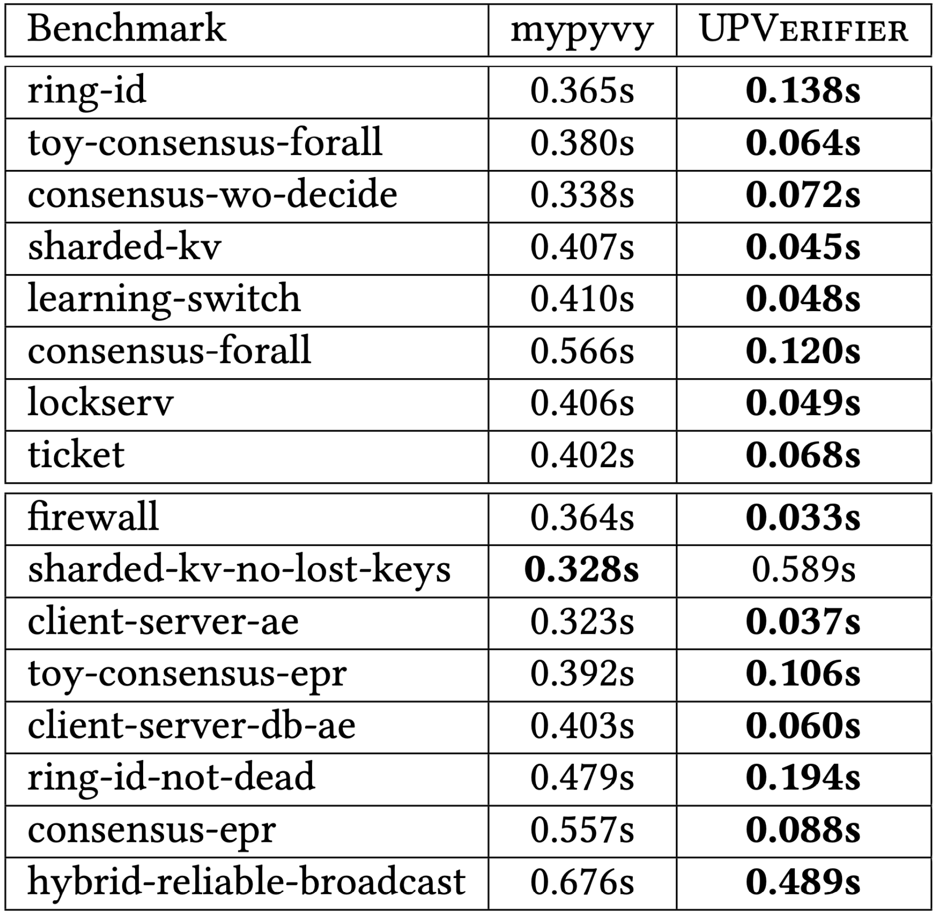
P(e) := “Most recent email holds largest ORCiD”

Empirical Evaluation

Research Questions

- RQ1: Expressive power comparison to related work
  - Show we can translate proofs to the UPVerifier
- RQ2: Performance comparison to related work
- RQ3: Existing proofs with and without message chains
  - More case studies, like Chang and Roberts verification
- RQ4: Specification mining evaluation
  - More case studies, like Chang and Roberts spec. mining
- RQ5: Industrial distributed systems

RQ2: Performance Comparison



Benchmark	mypyvy	UPVerifier
ring-id	0.365s	**0.138s**
toy-consensus-forall	0.380s	**0.064s**
consensus-wo-decide	0.338s	**0.072s**
sharded-kv	0.407s	**0.045s**
learning-switch	0.410s	**0.048s**
consensus-forall	0.566s	**0.120s**
lockserv	0.406s	**0.049s**
ticket	0.402s	**0.068s**
firewall	0.364s	**0.033s**
sharded-kv-no-lost-keys	**0.328s**	0.589s
client-server-ae	0.323s	**0.037s**
toy-consensus-epr	0.392s	**0.106s**
client-server-db-ae	0.403s	**0.060s**
ring-id-not-dead	0.479s	**0.194s**
consensus-epr	0.557s	**0.088s**
hybrid-reliable-broadcast	0.676s	**0.489s**

Koenig et el. (2020)

RQ5: Industrial Distributed Systems

- GlobalClock
  - 150 lines of code.
  - ~1.5 seconds to verify.
  - Message chain invariant used to distinguish phases for local clocks.

- Two-Phase Commit Multi-Version Concurrency Control
  - 250 lines of code.
  - ~12 minutes to verify.
  - Interesting message chain invariants because of “broadcasting” and “flooding,” which are nonlinear.

Summary and Thank you!

- We formally define
  - message chains and
  - message chain invariants.
- We show how to use them to
  - verify asynchronous, message-passing systems
  - while avoiding
    - reasoning about individual messages or buffers,
    - manually abstracting message passing, and
    - manually relating abstractions to messaging details.
- Present the UPVerifier

Works Cited In Presentation

Ernest Chang and Rosemary Roberts. 1979. An improved algorithm for decentralized extrema-finding in circular configurations of processes. Commun. ACM 22, 5 (May 1979), 281–283. https://doi.org/10.1145/359104.359108

M. D. Ernst, J. Cockrell, W. G. Griswold and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," in *IEEE Transactions on Software Engineering*, vol. 27, no. 2, pp. 99-123, Feb. 2001, doi: 10.1109/32.908957.

Glenn Ammons, Rastislav Bodík, and James R. Larus. 2002. Mining specifications. In Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '02). Association for Computing Machinery, New York, NY, USA, 4–16. https://doi.org/10.1145/503272.503275

Woosuk Lee and Hangyeol Cho. 2023. Inductive Synthesis of Structurally Recursive Functional Programs from Non-recursive Expressions. Proc. ACM Program. Lang. 7, POPL, Article 70 (January 2023), 31 pages.

Jason R. Koenig, Oded Padon, Neil Immerman, and Alex Aiken. 2020. First-order quantified separators. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020). Association for Computing Machinery, New York, NY, USA, 703–717. https://doi.org/10.1145/3385412.3386018