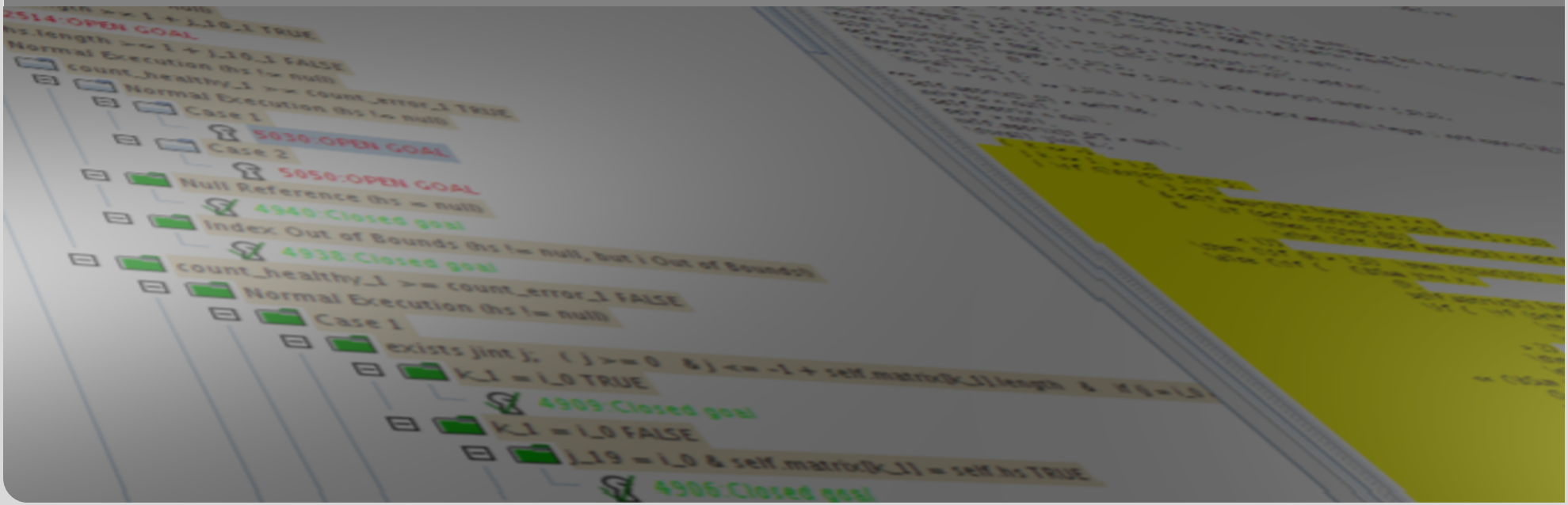
Byzantine Agreement Protocols

Formal Model in Event-B

Rodin User and Developer Workshop 2010

Roman Krenický and Mattias Ulbrich | September 21, 2010

LOGIC AND FORMAL METHODS

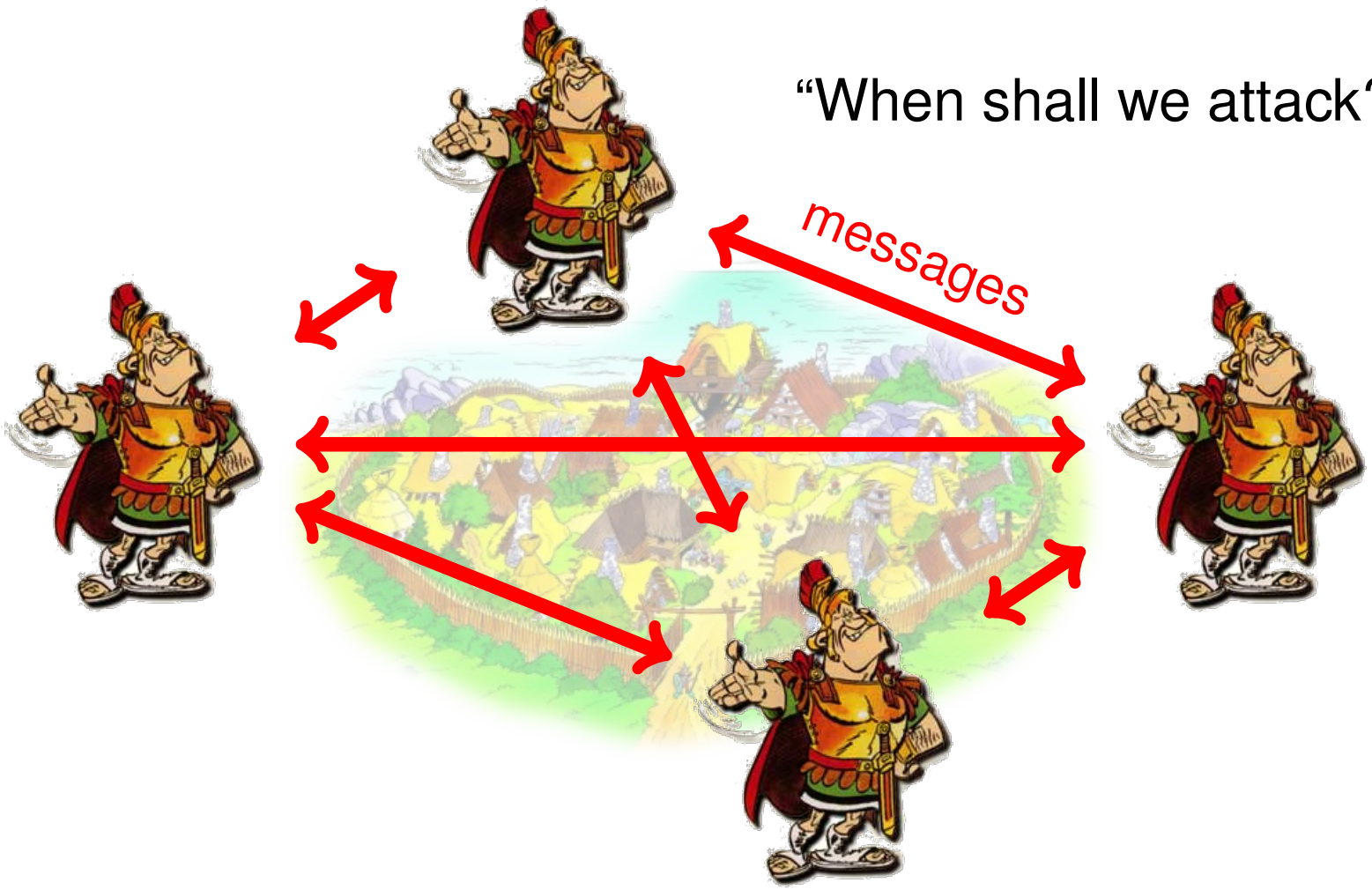


Outline

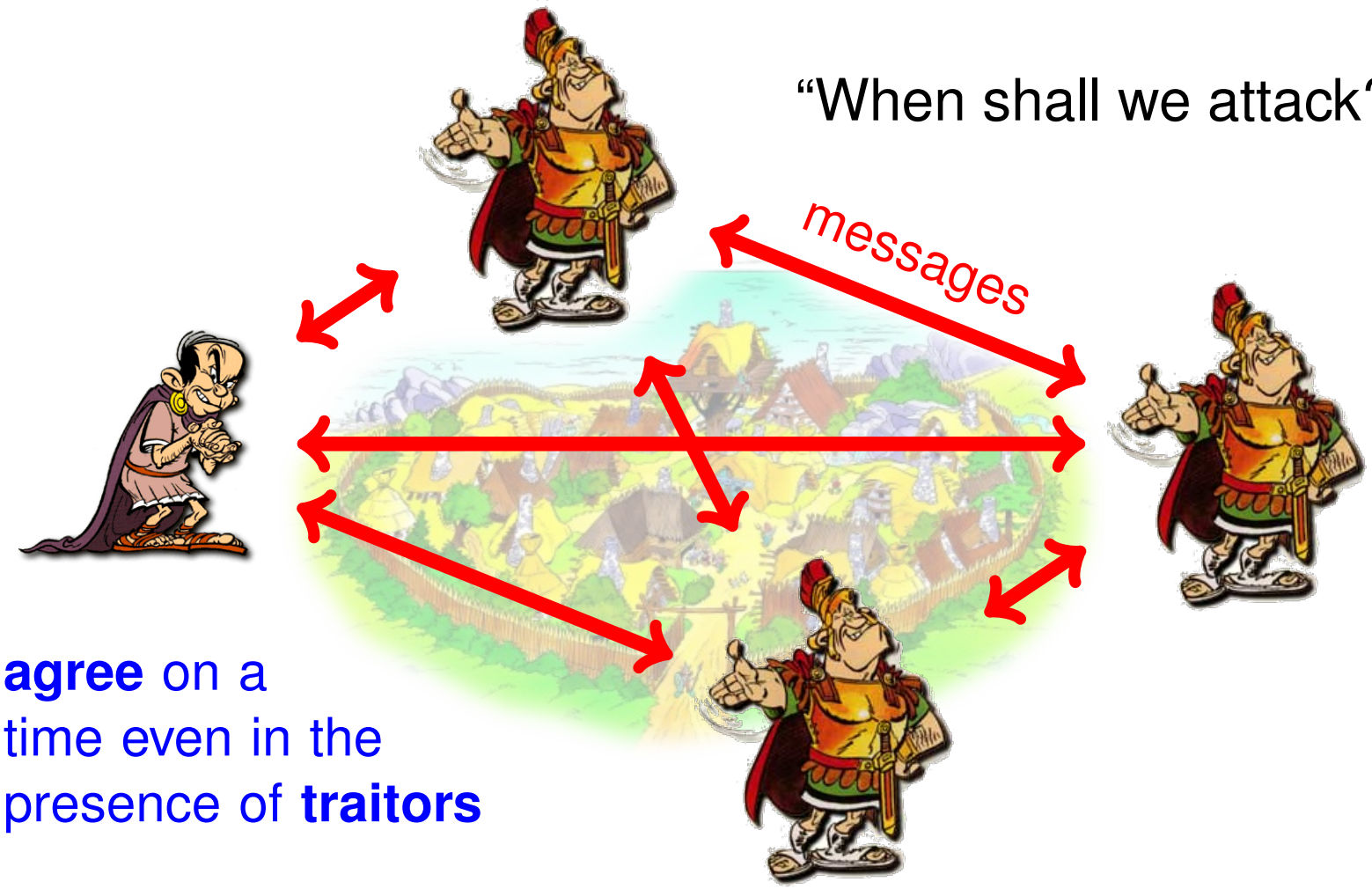
Byzantine Generals

Byzantine Generals

Byzantine Generals

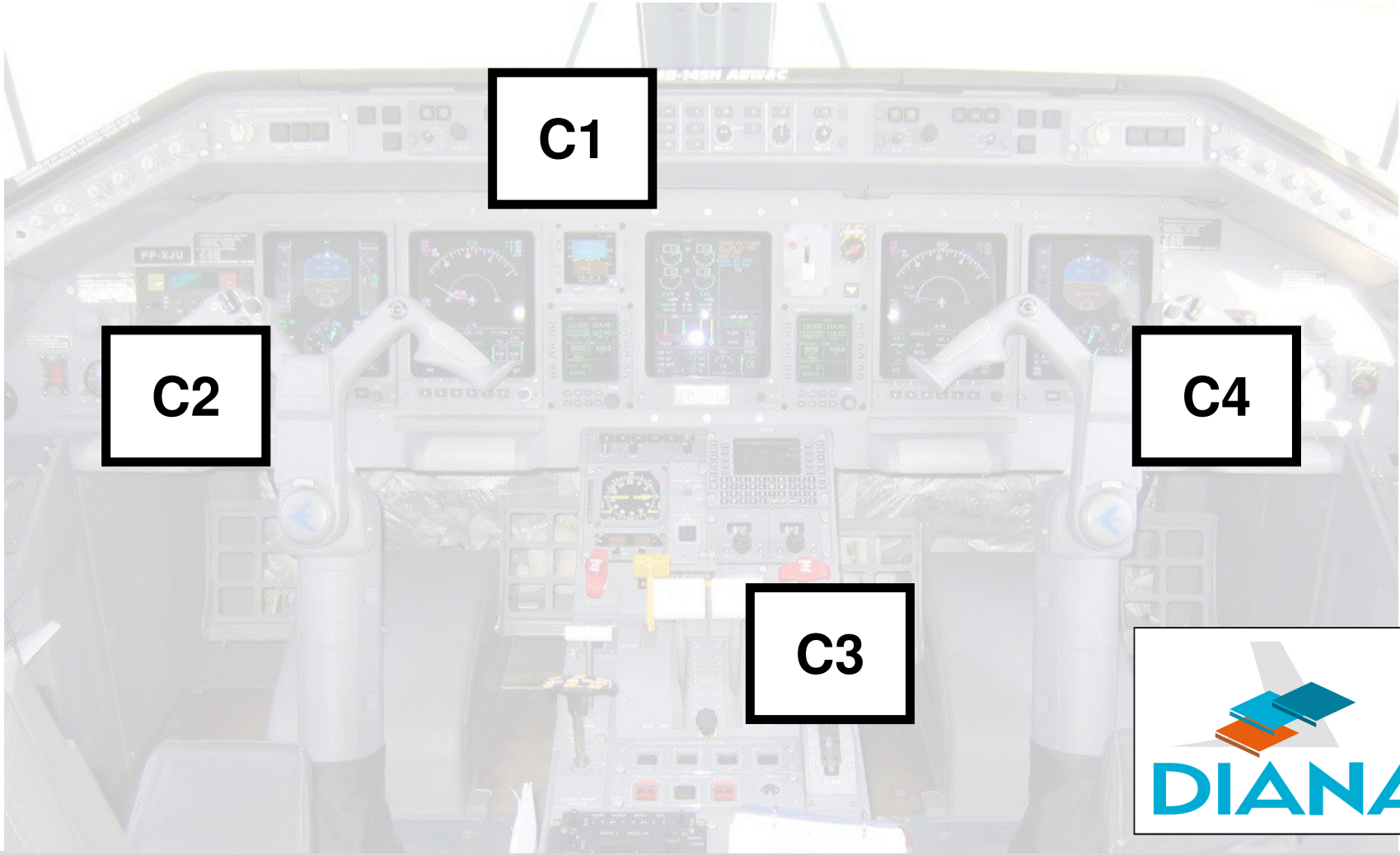


Byzantine Generals

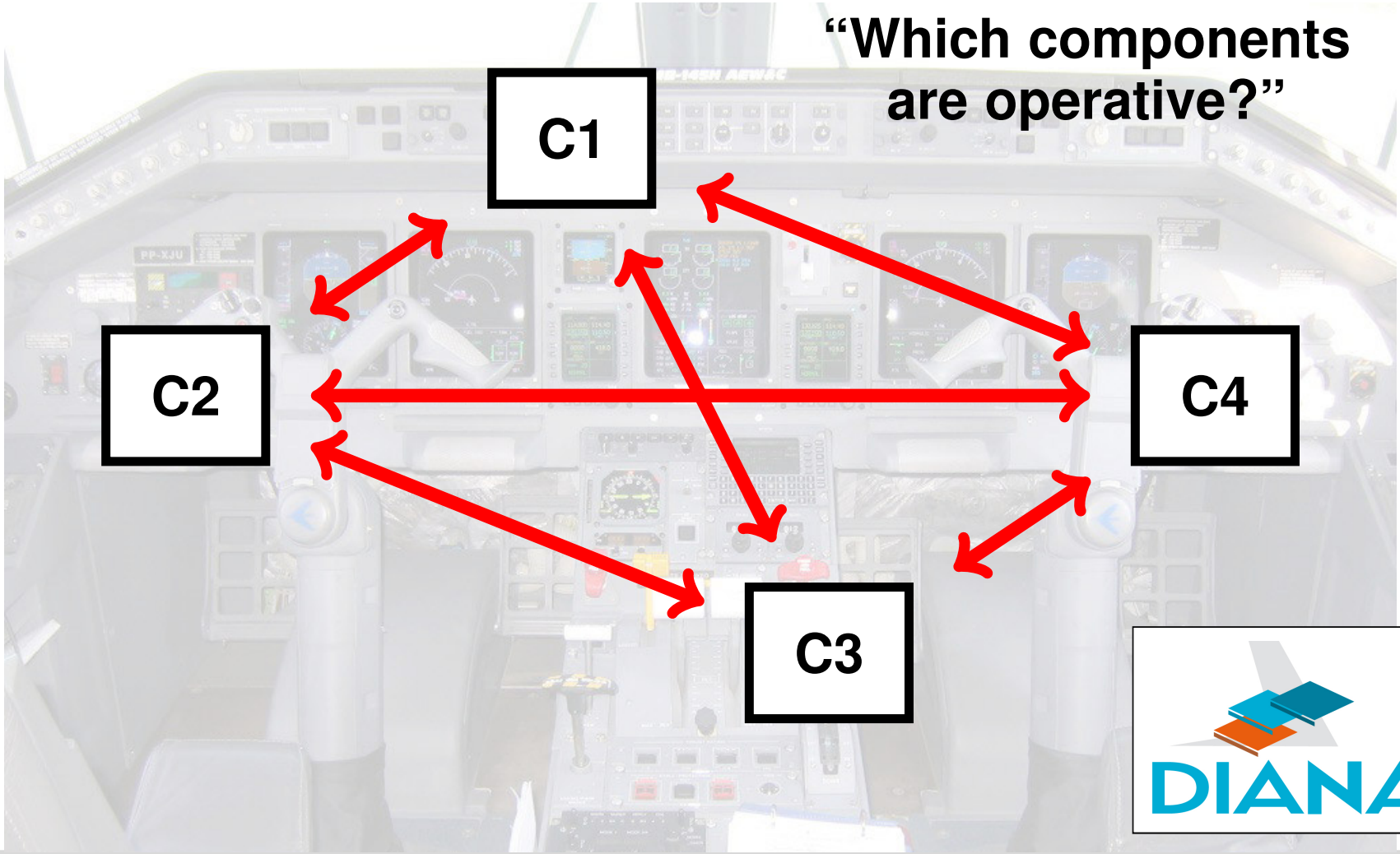


agree on a time even in the presence of **traitors**

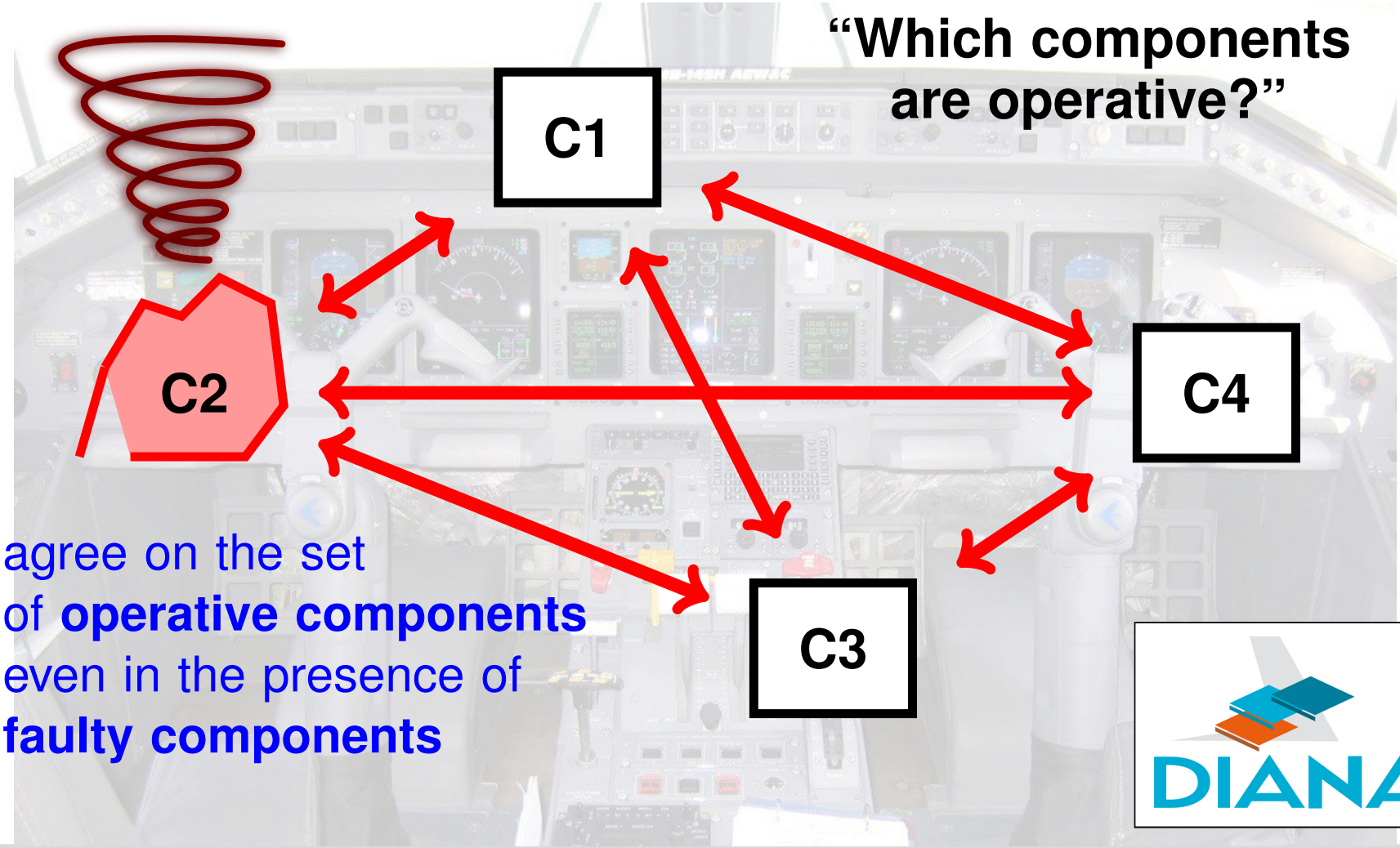
Application in Avionics



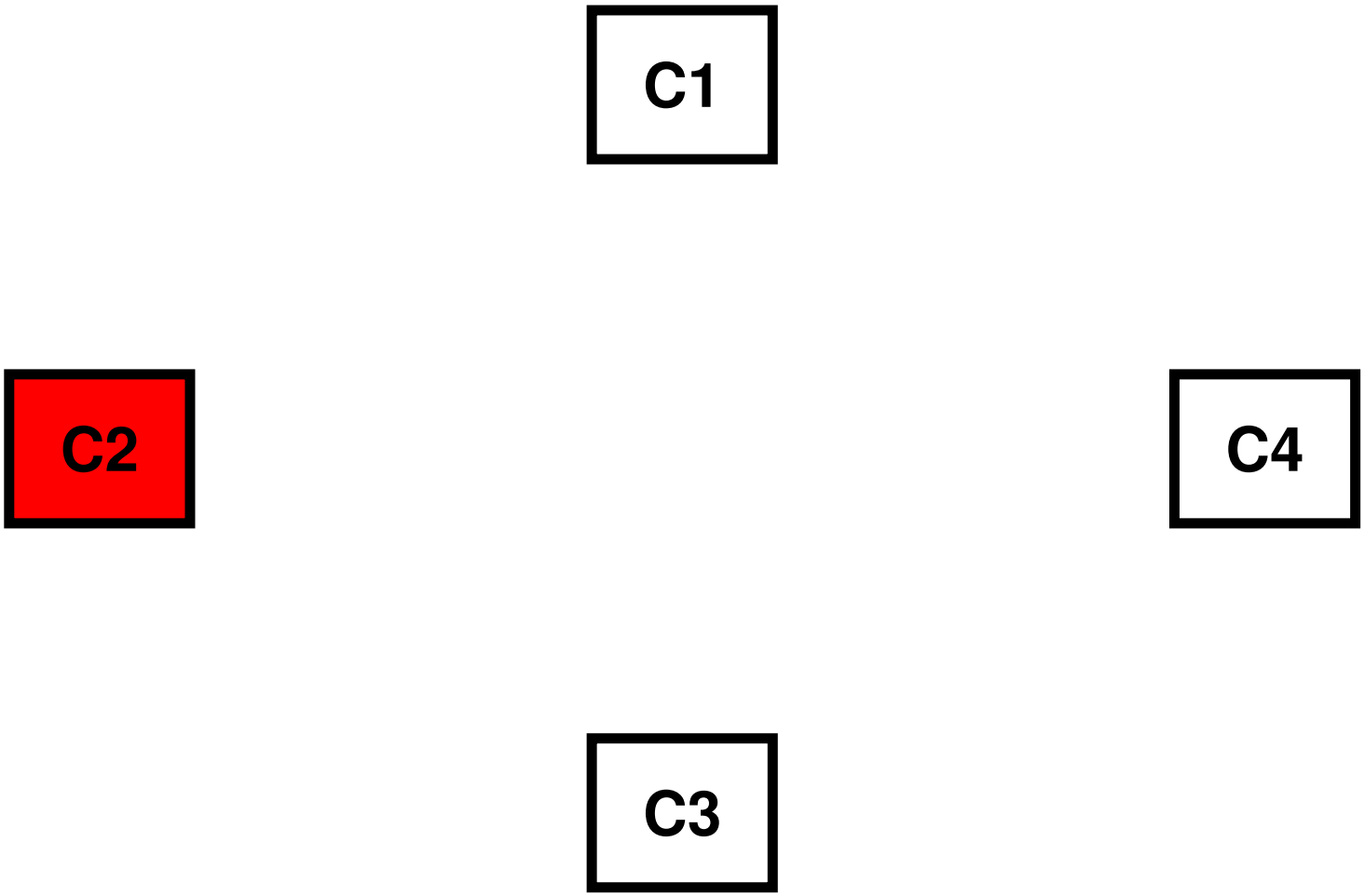
Application in Avionics



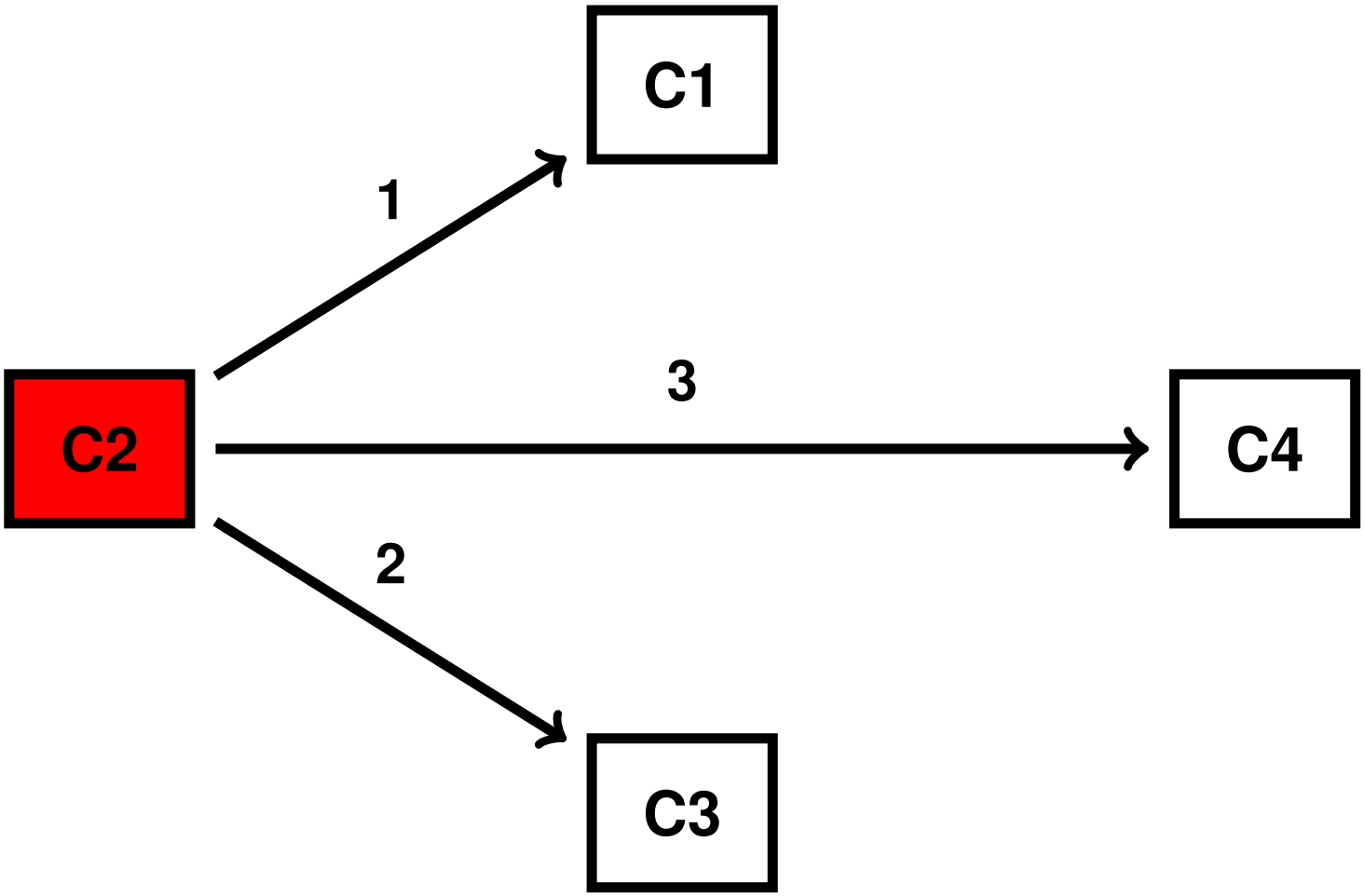
Application in Avionics



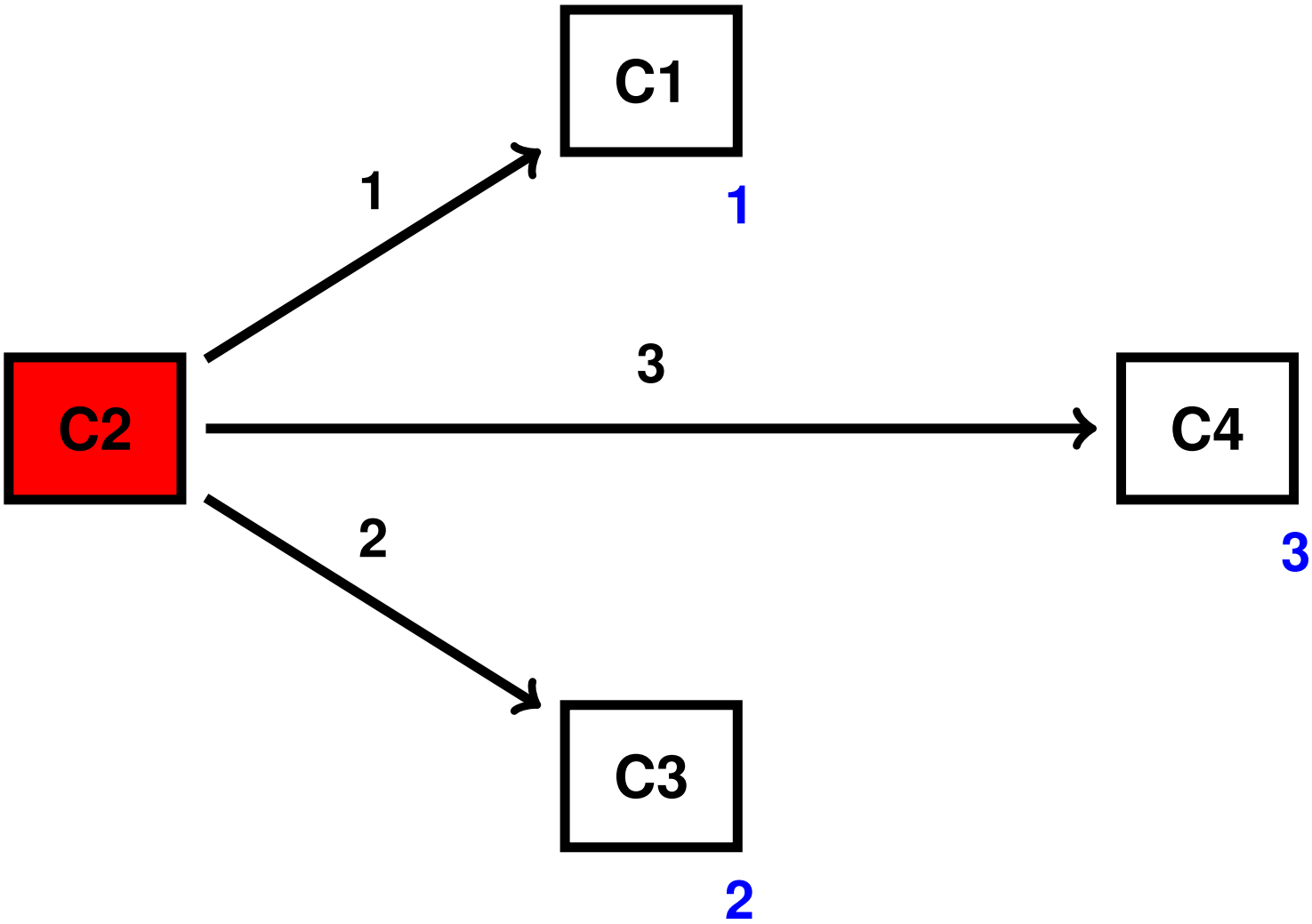
Explanation by Example



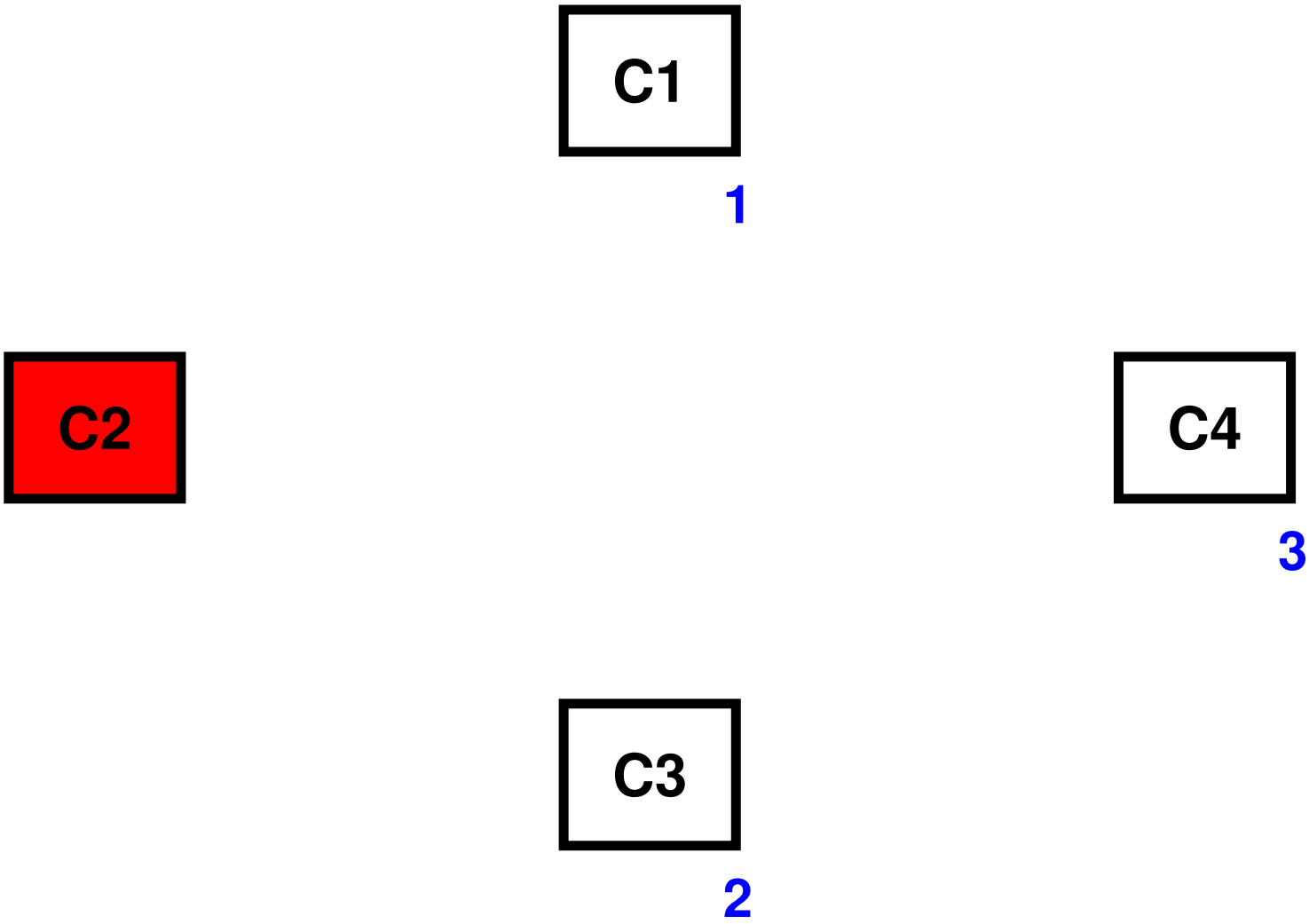
Explanation by Example



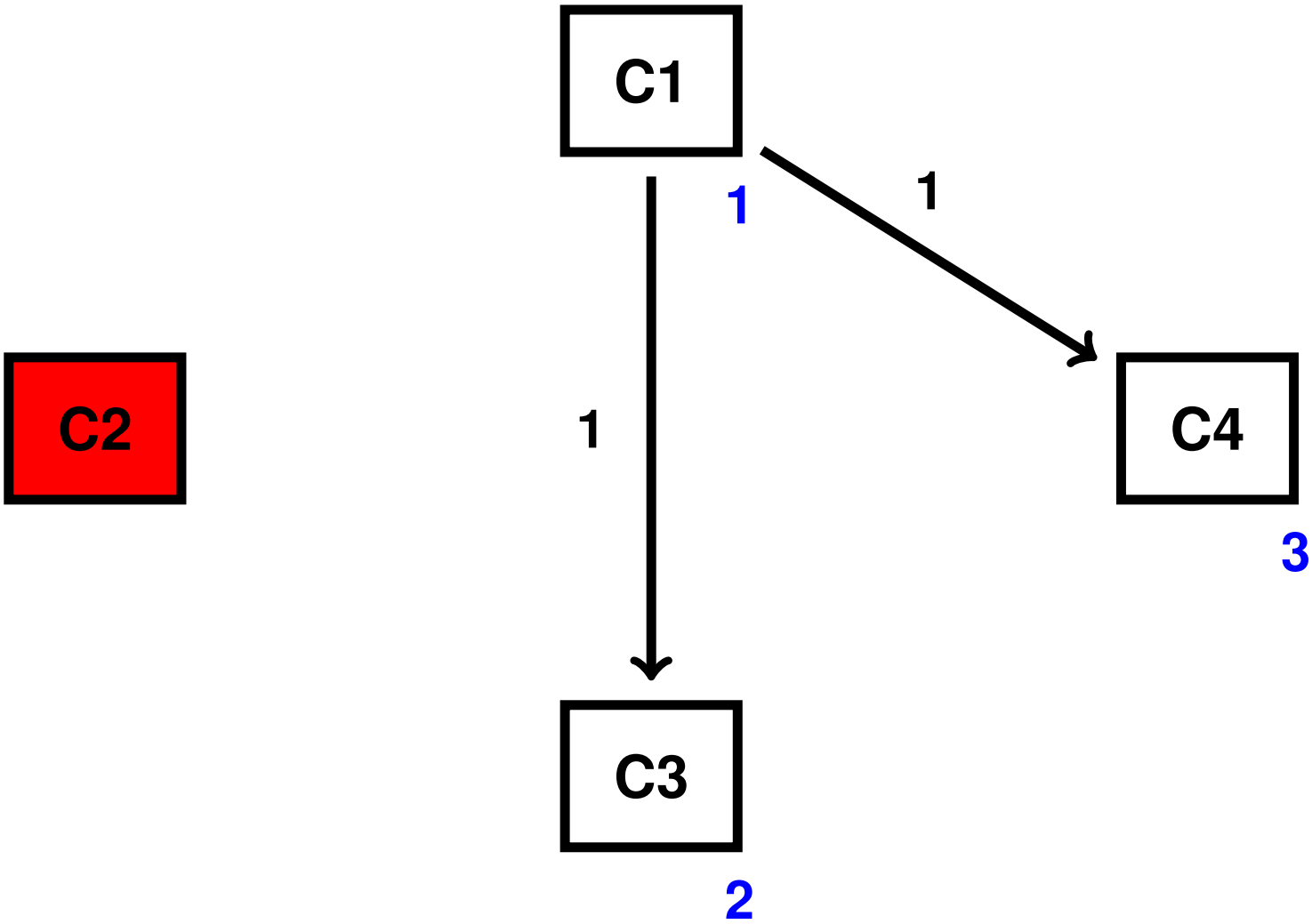
Explanation by Example



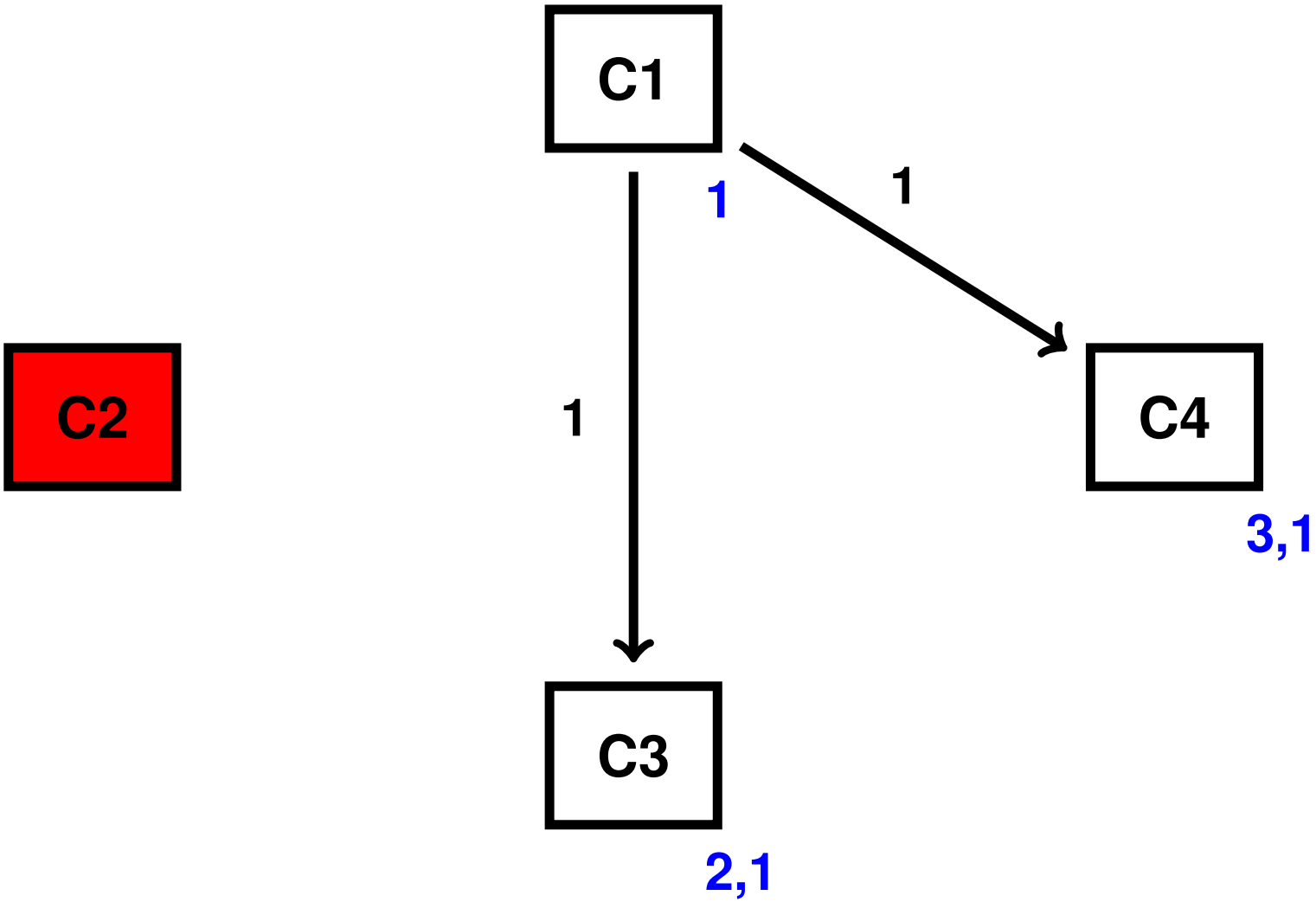
Explanation by Example



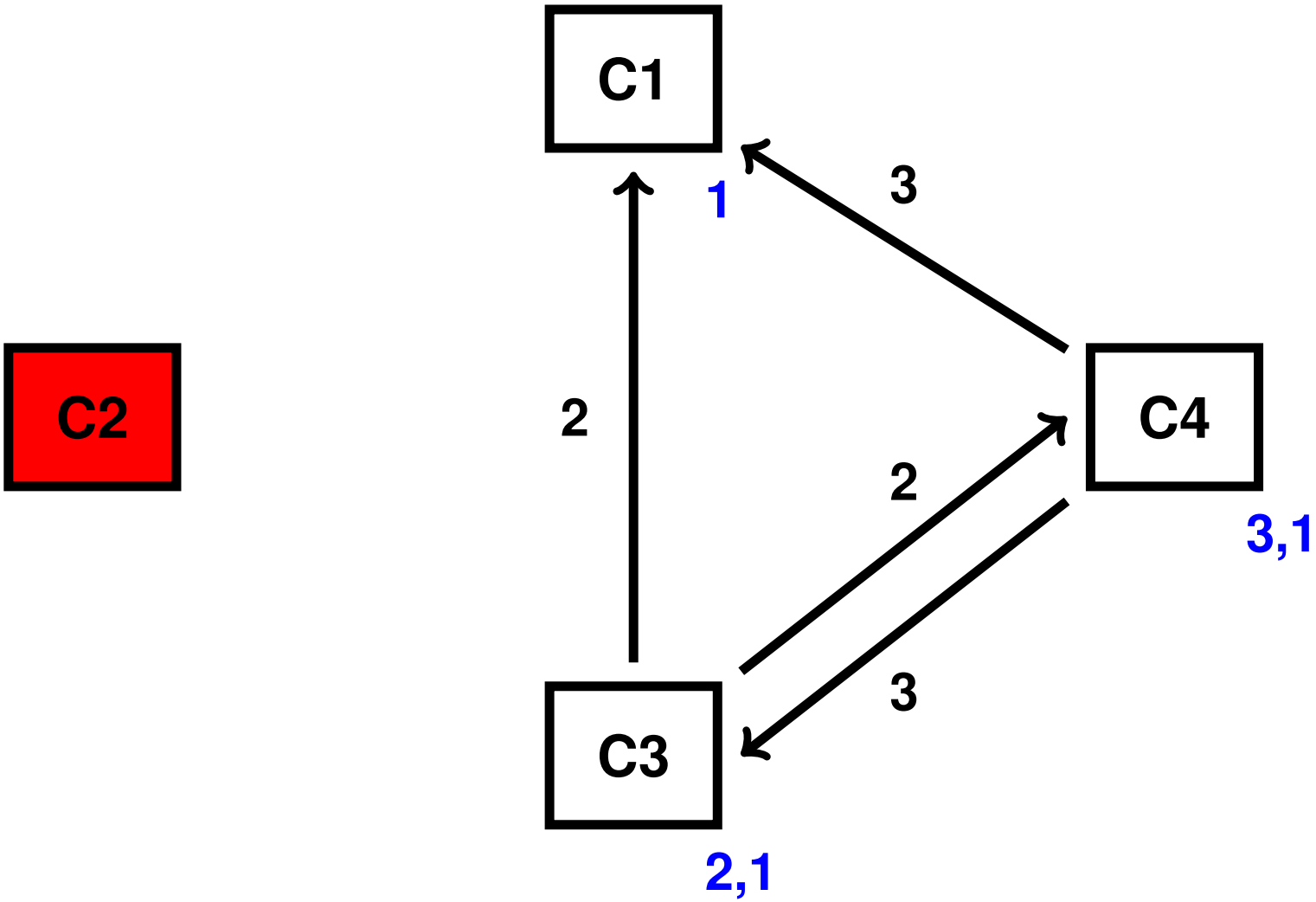
Explanation by Example



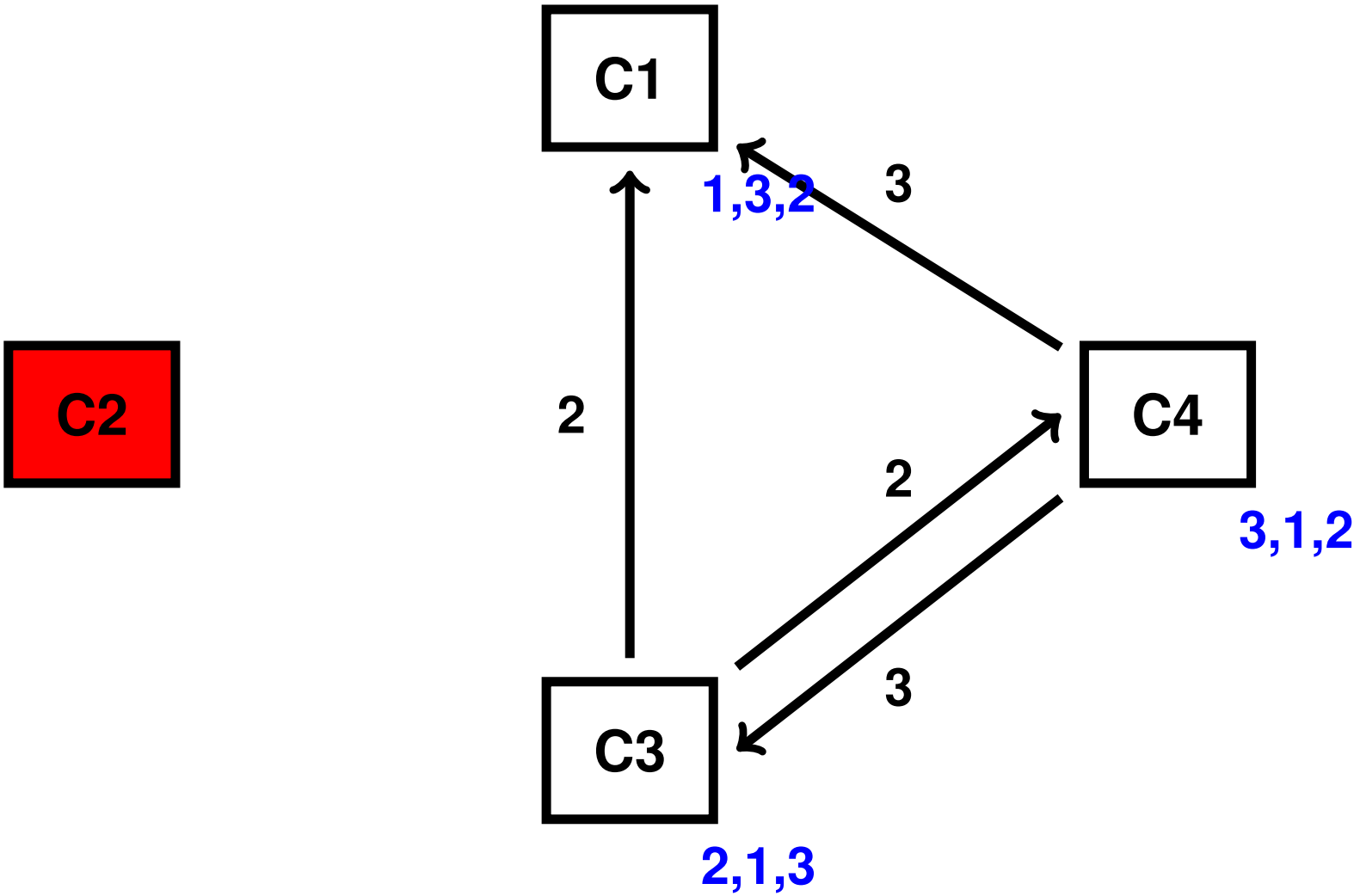
Explanation by Example



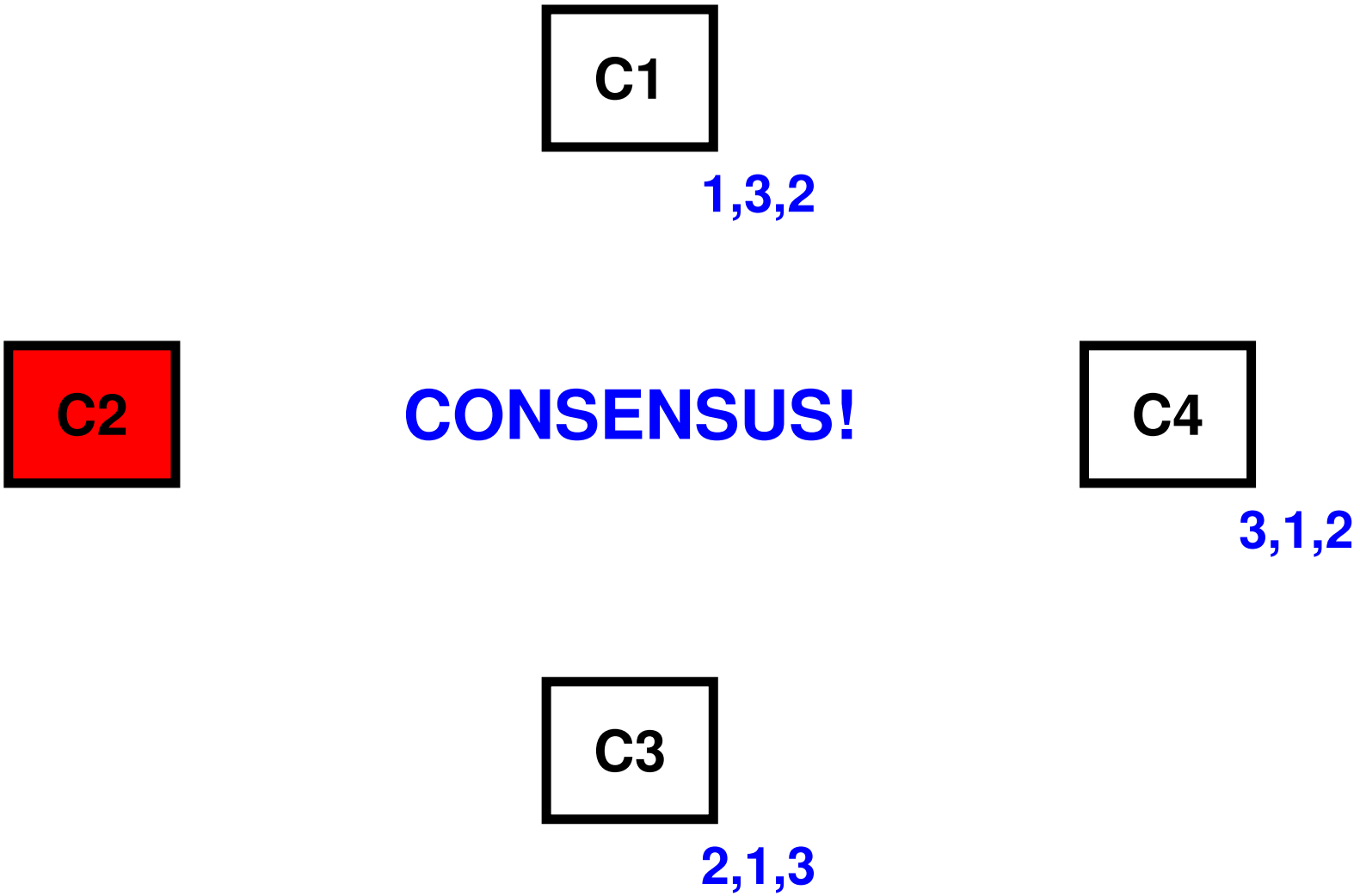
Explanation by Example



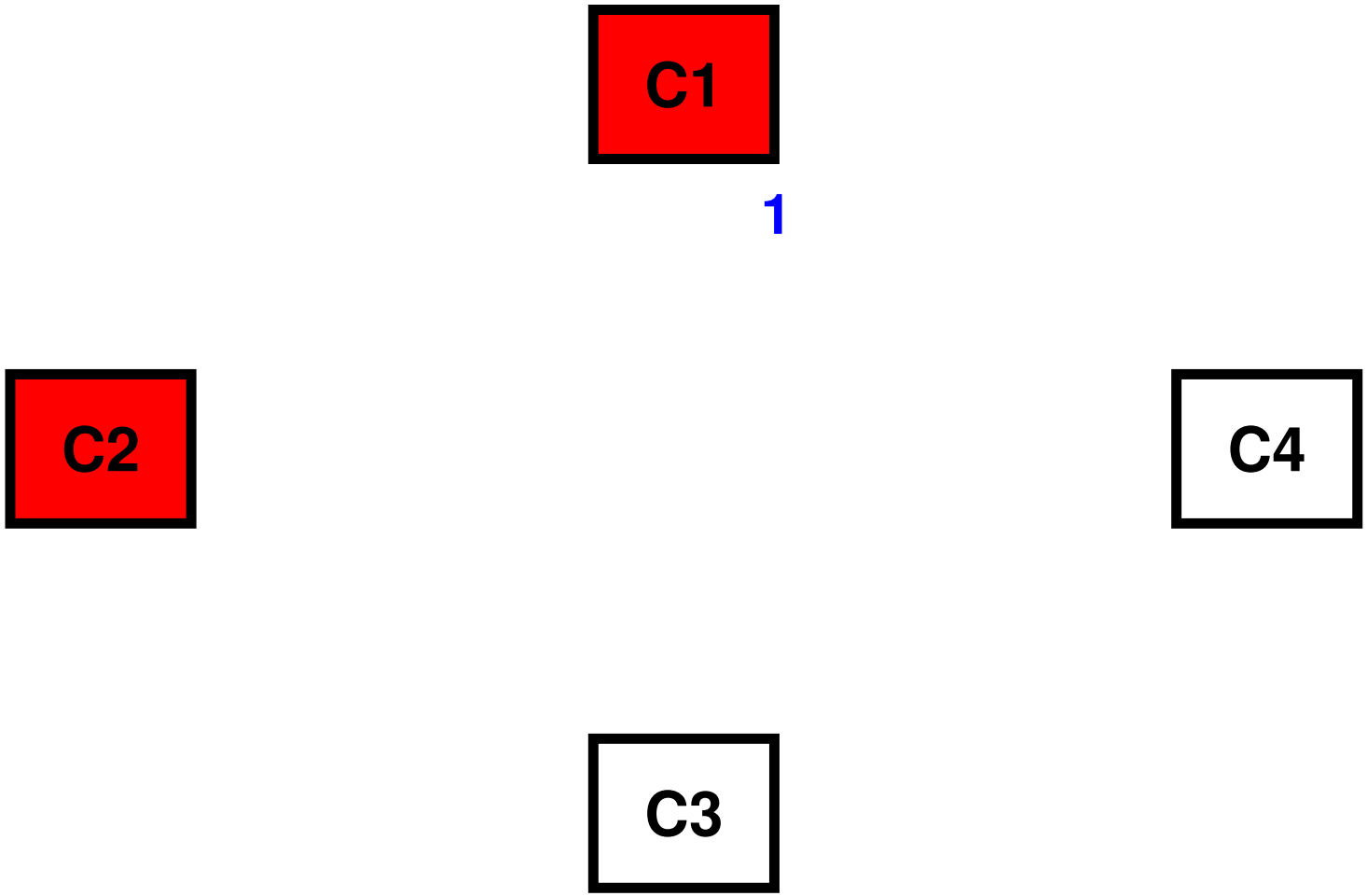
Explanation by Example



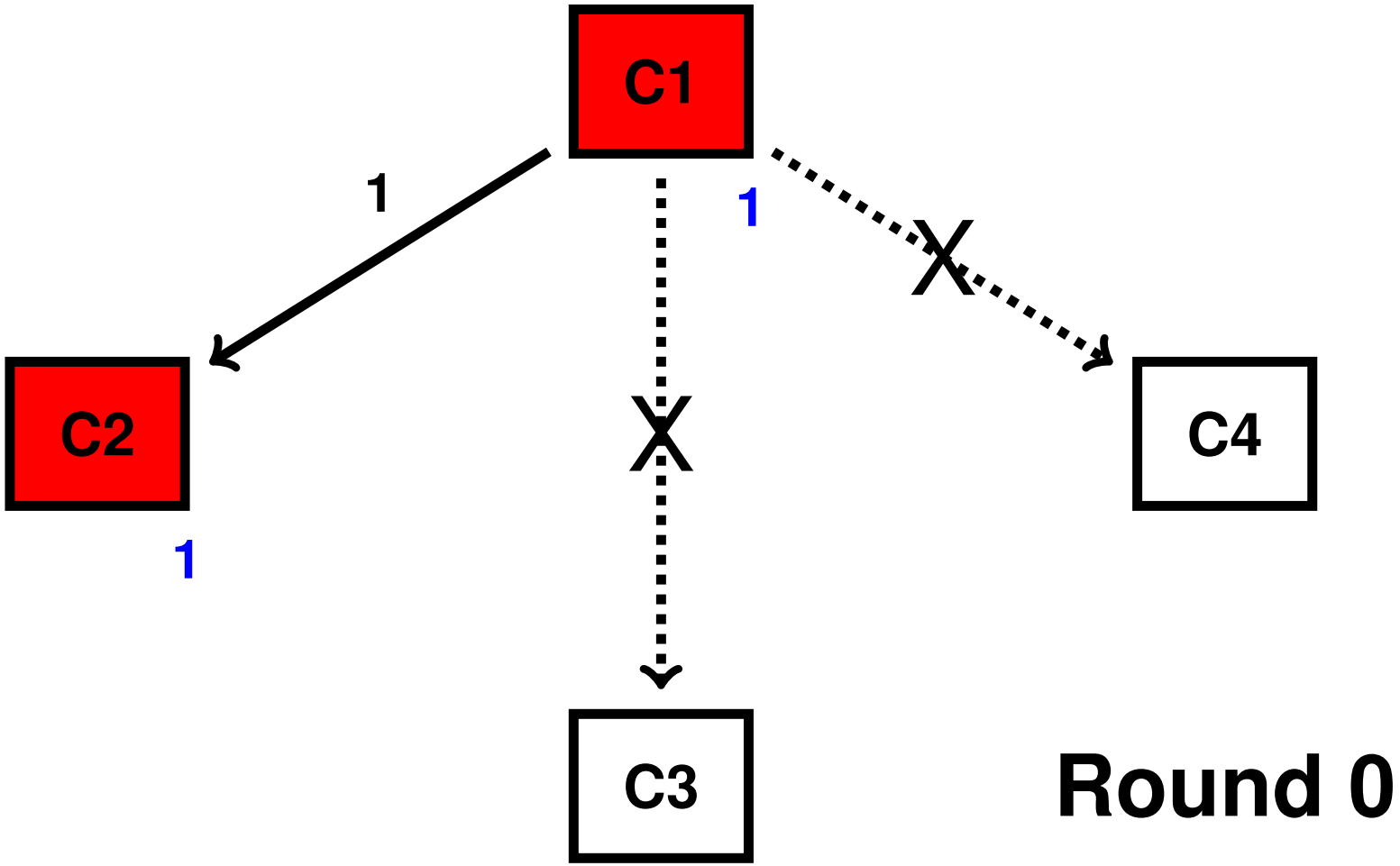
Explanation by Example



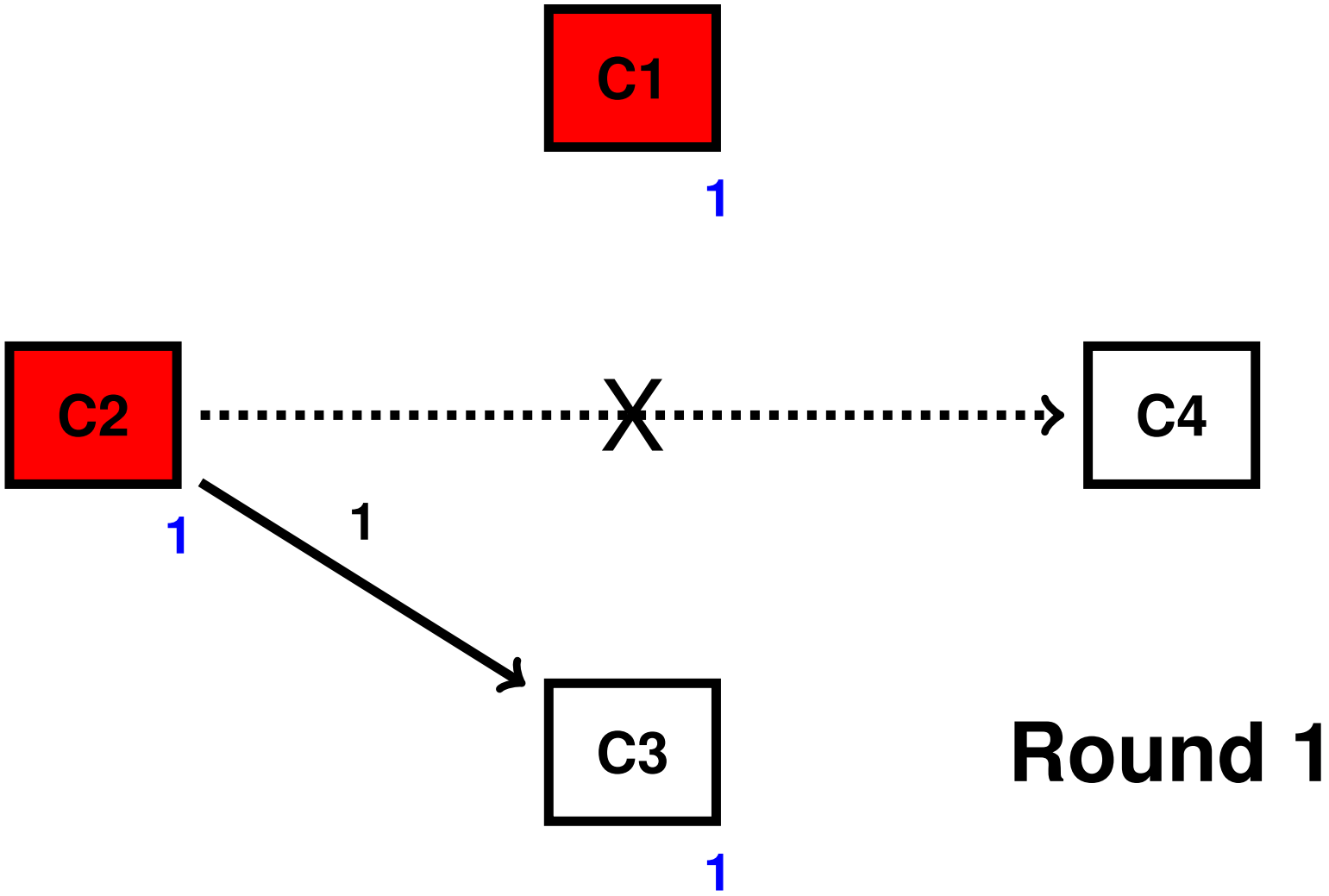
Example Run 2



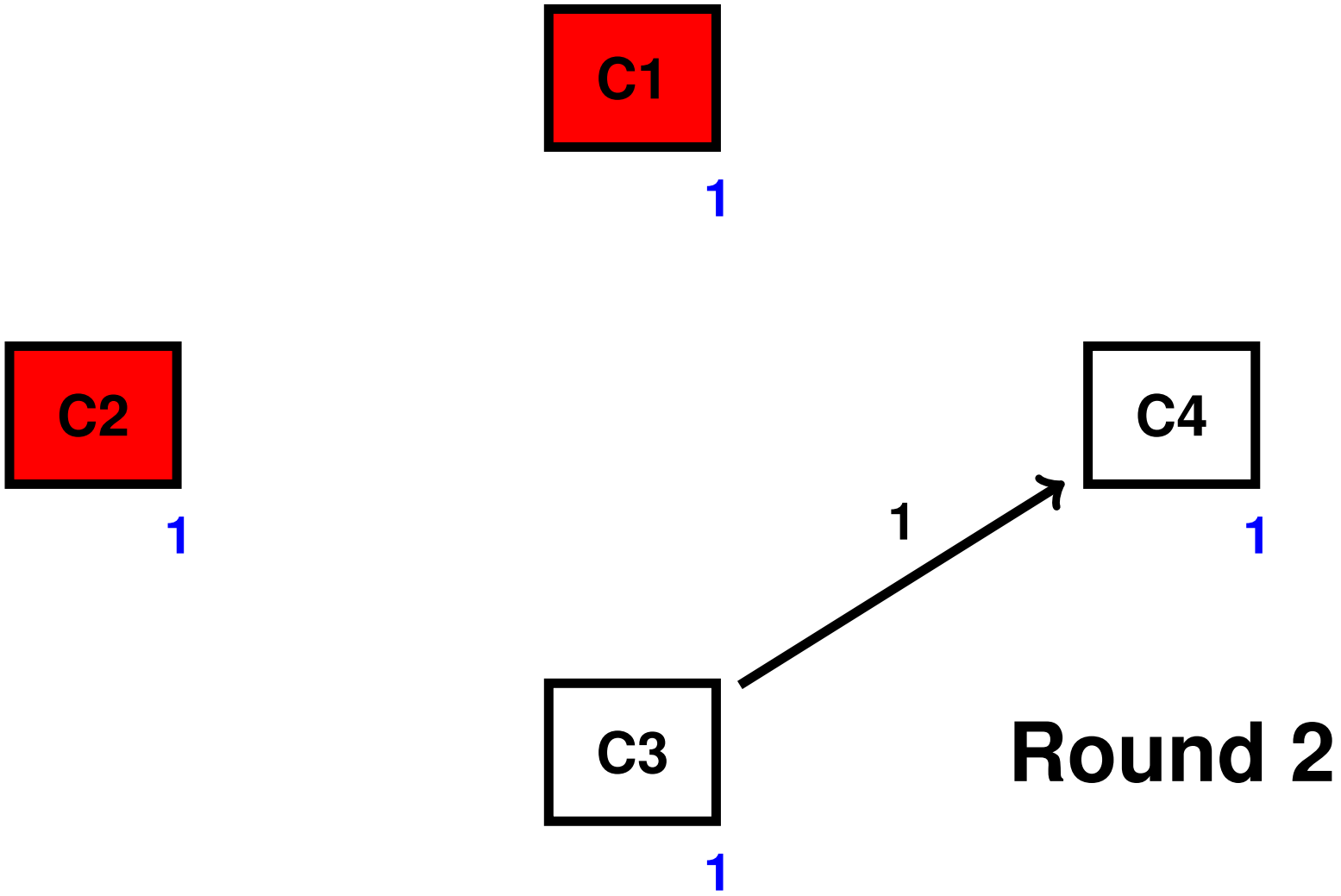
Example Run 2



Example Run 2



Example Run 2



Byzantine Agreement Algorithm

Verification Goals:

Validity If the transmitter *tt* is non-faulty, then all non-faulty receivers agree on the value sent by *tt*.

Agreement Any two non-faulty receivers agree on the value assigned to *tt*.

Byzantine Agreement Algorithm

Round 0: Transmitter sends signed message to all receivers.

Byzantine Agreement Algorithm

Round 0: Transmitter sends signed message to all receivers.

Round *n*: If a component receives a message, it proceeds as follows:

1. Verify the signature(s) of the message (discard on error)

Byzantine Agreement Algorithm

Round 0: Transmitter sends signed message to all receivers.

Round *n*: If a component receives a message, it proceeds as follows:

1. Verify the signature(s) of the message (discard on error)
2. Discard the message if the value has been observed earlier.

Byzantine Agreement Algorithm

Round 0: Transmitter sends signed message to all receivers.

Round *n*: If a component receives a message, it proceeds as follows:

1. Verify the signature(s) of the message (discard on error)
2. Discard the message if the value has been observed earlier.
3. Add signature to the message and pass it on to all nodes that have not yet seen the message.

Byzantine Agreement Algorithm

Round 0: Transmitter sends signed message to all receivers.

Round *n*: If a component receives a message, it proceeds as follows:

1. Verify the signature(s) of the message (discard on error)
2. Discard the message if the value has been observed earlier.
3. Add signature to the message and pass it on to all nodes that have not yet seen the message.

GOAL: Prove that this algorithm has the “validity” and “agreement” properties.

Verification

Quote

We know of no area in computer science or mathematics in which informal reasoning is more likely to lead to errors than in the study of this type of algorithm.

Taken from: *The Byzantine Generals Problem*

Leslie Lamport, Robert Shostak, and Marshall Pease
ACM Transactions on Programming Languages and Systems
Volume 4, pp. 383–401,1982.

Formal Verification

So far ...

- (Explicit) Model Checking for $|nodes| \leq 4$
- PVS (i.e., HOL) formalisation and proofs of Oral Messages (recursive) [Lincoln and Rushby, '93]

Formal Verification

So far ...

- (Explicit) Model Checking for $|nodes| \leq 4$
- PVS (i.e., HOL) formalisation and proofs of Oral Messages (recursive) [Lincoln and Rushby, '93]

Now and here: **Event-B and RODIN**

Rigorous **O**pen **D**evelopment Env**i**ron**n**ment for Complex Systems

Model

- system := a set of MODULES, either **faulty** or **non-faulty**.
- modules send and receive **messages** containing VALUES
- one dedicated module **transmitter**
- **round based** (transmitter acts in round 0)
- there is a "good" value V_0 intended, observed, ... value
- non-faulty transmitter $\Longrightarrow$ send V_0 to everyone else
- round > 0 $\Longrightarrow$ transmitter silent, other modules **relay**
- non-faulty $\Longrightarrow$ relays every message to modules that have not seen this message yet.
- faulty $\Longrightarrow$ may **drop** messages, but **NOT** forge.

Model

- system := a set of MODULES, either **faulty** or **non-faulty**.
- modules send and receive **messages** containing VALUES
- one dedicated module **transmitter**
- **round based** (transmitter acts in round 0)
- there is a "good" value V_0 intended, observed, ... value
- non-faulty transmitter $\Longrightarrow$ send V_0 to everyone else
- round > 0 $\Longrightarrow$ transmitter silent, other modules **relay**
- non-faulty $\Longrightarrow$ relays every message to modules that have not seen this message yet.
- faulty $\Longrightarrow$ may **drop** messages, but **NOT** forge.

Model

- system := a set of MODULES, either **faulty** or **non-faulty**.
- modules send and receive **messages** containing VALUES
- one dedicated module **transmitter**
- **round based** (transmitter acts in round 0)
- there is a "good" value V_0 intended, observed, ... value
- non-faulty transmitter $\Longrightarrow$ send V_0 to everyone else
- round > 0 $\Longrightarrow$ transmitter silent, other modules **relay**
- non-faulty $\Longrightarrow$ relays every message to modules that have not seen this message yet.
- faulty $\Longrightarrow$ may **drop** messages, but **NOT** forge.

Model

- system := a set of MODULES, either **faulty** or **non-faulty**.
- modules send and receive **messages** containing VALUES
- one dedicated module **transmitter**
- **round based** (transmitter acts in round 0)
- there is a "good" value V_0 intended, observed, ... value
- non-faulty transmitter $\Longrightarrow$ send V_0 to everyone else
- round > 0 $\Longrightarrow$ transmitter silent, other modules **relay**
- non-faulty $\Longrightarrow$ relays every message to modules that have not seen this message yet.
- faulty $\Longrightarrow$ may **drop** messages, but **NOT** forge.

Model

- system := a set of MODULES, either **faulty** or **non-faulty**.
- modules send and receive **messages** containing VALUES
- one dedicated module **transmitter**
- **round based** (transmitter acts in round 0)
- there is a "good" value V_0 intended, observed, ... value
- non-faulty transmitter $\Longrightarrow$ send V_0 to everyone else
- round > 0 $\Longrightarrow$ transmitter silent, other modules **relay**
- non-faulty $\Longrightarrow$ relays every message to modules that have not seen this message yet.
- faulty $\Longrightarrow$ may **drop** messages, but **NOT** forge.

Model

- system := a set of MODULES, either **faulty** or **non-faulty**.
- modules send and receive **messages** containing VALUES
- one dedicated module **transmitter**
- **round based** (transmitter acts in round 0)
- there is a "good" value V_0 intended, observed, ... value
- non-faulty transmitter $\Longrightarrow$ send V_0 to everyone else
- round > 0 $\Longrightarrow$ transmitter silent, other modules **relay**
- non-faulty $\Longrightarrow$ relays every message to modules that have not seen this message yet.
- faulty $\Longrightarrow$ may **drop** messages, but **NOT** forge.

Model

- system := a set of MODULES, either **faulty** or **non-faulty**.
- modules send and receive **messages** containing VALUES
- one dedicated module **transmitter**
- **round based** (transmitter acts in round 0)
- there is a "good" value V_0 intended, observed, ... value
- non-faulty transmitter $\Longrightarrow$ send V_0 to everyone else
- round $> 0 \Longrightarrow$ transmitter silent, other modules **relay**
- non-faulty $\Longrightarrow$ relays every message to modules that have not seen this message yet.
- faulty $\Longrightarrow$ may **drop** messages, but **NOT** forge.

Model

- system := a set of MODULES, either **faulty** or **non-faulty**.
- modules send and receive **messages** containing VALUES
- one dedicated module **transmitter**
- **round based** (transmitter acts in round 0)
- there is a "good" value V_0 intended, observed, ... value
- non-faulty transmitter $\Longrightarrow$ send V_0 to everyone else
- round > 0 $\Longrightarrow$ transmitter silent, other modules **relay**
- non-faulty $\Longrightarrow$ relays every message to modules that have not seen this message yet.
- faulty $\Longrightarrow$ may **drop** messages, but **NOT** forge.

Model

- system := a set of MODULES, either **faulty** or **non-faulty**.
- modules send and receive **messages** containing VALUES
- one dedicated module **transmitter**
- **round based** (transmitter acts in round 0)
- there is a "good" value V_0 intended, observed, ... value
- non-faulty transmitter $\Longrightarrow$ send V_0 to everyone else
- round > 0 $\Longrightarrow$ transmitter silent, other modules **relay**
- non-faulty $\Longrightarrow$ relays every message to modules that have not seen this message yet.
- faulty $\Longrightarrow$ may **drop** messages, but **NOT** forge.

Context for Byzantine Agreement

CONTEXT CONTEXT

SETS

CONSTANTS

AXIOMS

END

Context for Byzantine Agreement

```
CONTEXT  Context
    SETS
            Module
            Value
CONSTANTS

  AXIOMS



     END
```

Context for Byzantine Agreement

CONTEXT CONTEXT

SETS

MODULE

VALUE

CONSTANTS

faulty, *transmitter*, V_0

AXIOMS

END

Context for Byzantine Agreement

CONTEXT Context

SETS

Module

Value

CONSTANTS

$faulty$, $transmitter$, V_0

AXIOMS

axm1 : $faulty \subseteq \text{Module}$

axm2 : $transmitter \in \text{Module}$

axm3 : $V_0 \in \text{Value}$

axm4 : $\text{finite}(faulty)$

END

First machine

MACHINE MESSAGES
SEES CONTEXT

VARIABLES

INVARIANTS

...

First machine

MACHINE MESSAGES
SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS

...

First machine

MACHINE MESSAGES
SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS

ty_mess : $messages \subseteq \text{MODULE} \times \text{MODULE} \times \text{VALUE}$

...

messages messages being sent in the *current* round

First machine

MACHINE MESSAGES
SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS

ty_mess : $messages \subseteq \text{MODULE} \times \text{MODULE} \times \text{VALUE}$

ty_round : $round \in \mathbb{N}$

...

messages messages being sent in the *current* round

round the number of the current round

First machine

MACHINE MESSAGES
SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS

ty_mess : $messages \subseteq \text{MODULE} \times \text{MODULE} \times \text{VALUE}$

ty_round : $round \in \mathbb{N}$

ty_collected : $collected \in \text{MODULE} \rightarrow \mathbb{P}(\text{VALUE})$

...

messages messages being sent in the *current* round

round the number of the current round

collected values observed in previous rounds

First machine (2)

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

First machine (2)

messages messages being sent in the *current* round

round the number of the current round

collected values observed in previous rounds

MACHINE MESSAGES SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS ...

EVENTS

Initialisation ...

EVENT ROUND $\widehat{=}$

begin

end

END

First machine (2)

messages messages being sent in the *current* round

round the number of the current round

collected values observed in previous rounds

MACHINE MESSAGES SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS ...

EVENTS

Initialisation ...

EVENT ROUND $\widehat{=}$

begin

act1: $round := round + 1$

end

END

First machine (2)

messages messages being sent in the *current* round

round the number of the current round

collected values observed in previous rounds

MACHINE MESSAGES SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS ...

EVENTS

Initialisation ...

EVENT ROUND $\widehat{=}$

begin

act1: $round := round + 1$

act2: $messages :\in \mathbb{P}(\text{MODULE} \times \text{MODULE} \times \text{VALUE})$

end

END

First machine (2)

messages messages being sent in the *current* round

round the number of the current round

collected values observed in previous rounds

MACHINE MESSAGES SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS ...

EVENTS

Initialisation ...

EVENT ROUND $\widehat{=}$

begin

act1: $round := round + 1$

act2: $messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

end

END

First machine (2)

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

MACHINE MESSAGES SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS ...

EVENTS

Initialisation ...

EVENT ROUND $\widehat{=}$

begin

act1: $round := round + 1$

act2: $messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

act3: $collected := \lambda m \cdot collected(m) \cup$

end

END

First machine (2)

messages messages being sent in the *current* round

round the number of the current round

collected values observed in previous rounds

MACHINE MESSAGES SEES CONTEXT

VARIABLES *messages*, *round*, *collected*

INVARIANTS ...

EVENTS

Initialisation ...

EVENT ROUND $\widehat{=}$

begin

act1: $round := round + 1$

act2: $messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

act3: $collected := \lambda m \cdot collected(m) \cup \{v \mid (s, m, v) \in messages\}$

end

END

First refinement: signed messages

All messages are signed in a trustworthy manner:

No forgery possible $\Longrightarrow$ Consider only **relayed** messages.

First refinement: signed messages

All messages are signed in a trustworthy manner:

No forgery possible $\Longrightarrow$ Consider only **relayed** messages.

round k:

First refinement: signed messages

All messages are signed in a trustworthy manner:

No forgery possible $\Longrightarrow$ Consider only **relayed** messages.

round k: $s \xrightarrow{v} r$

round $k+1$: $r \xrightarrow{v} n$

Signed messages (2)

MACHINE SIGNEDMESSAGES REFINES MESSAGES

VARIABLES *messages*, *round*, *collected*

INVARIANTS

EVENTS

END

Signed messages (2)

MACHINE SIGNEDMESSAGES REFINES MESSAGES

VARIABLES *messages*, *round*, *collected*

INVARIANTS

EVENTS

EVENT ROUND refines ROUND $\widehat{=}$
begin
act1, act3 as above

end
END

Signed messages (2)

MACHINE SIGNEDMESSAGES REFINES MESSAGES

VARIABLES *messages*, *round*, *collected*

INVARIANTS

EVENTS

EVENT ROUND refines ROUND $\widehat{=}$

begin

act1, act3 as above

was: $messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

end

END

Signed messages (2)

MACHINE SIGNEDMESSAGES REFINES MESSAGES

VARIABLES *messages*, *round*, *collected*

INVARIANTS

EVENTS

EVENT ROUND refines ROUND $\widehat{=}$

begin

act1, act3 as above

act2: $messages :\in \mathbb{P}\left(\{(r, n, v) \mid (s, r, v) \in messages\}\right)$

was: $messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

end

END

Signed messages (2)

MACHINE SIGNEDMESSAGES REFINES MESSAGES

VARIABLES *messages*, *round*, *collected*

INVARIANTS

val1: $\forall s, r, v \cdot (s, r, v) \in messages \Rightarrow v \in collected(transmitter)$

EVENTS

EVENT ROUND refines ROUND $\widehat{=}$

begin

act1, act3 as above

act2: $messages :\in \mathbb{P}\left(\{(r, n, v) \mid (s, r, v) \in messages\}\right)$

was: $messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

end

END

Signed messages (2)

MACHINE SIGNEDMESSAGES REFINES MESSAGES

VARIABLES *messages*, *round*, *collected*

INVARIANTS

val1: $\forall s, r, v \cdot (s, r, v) \in messages \Rightarrow v \in collected(transmitter)$

val2: $\forall n \cdot collected(n) \subseteq collected(transmitter)$

EVENTS

EVENT ROUND refines ROUND $\widehat{=}$

begin

act1, act3 as above

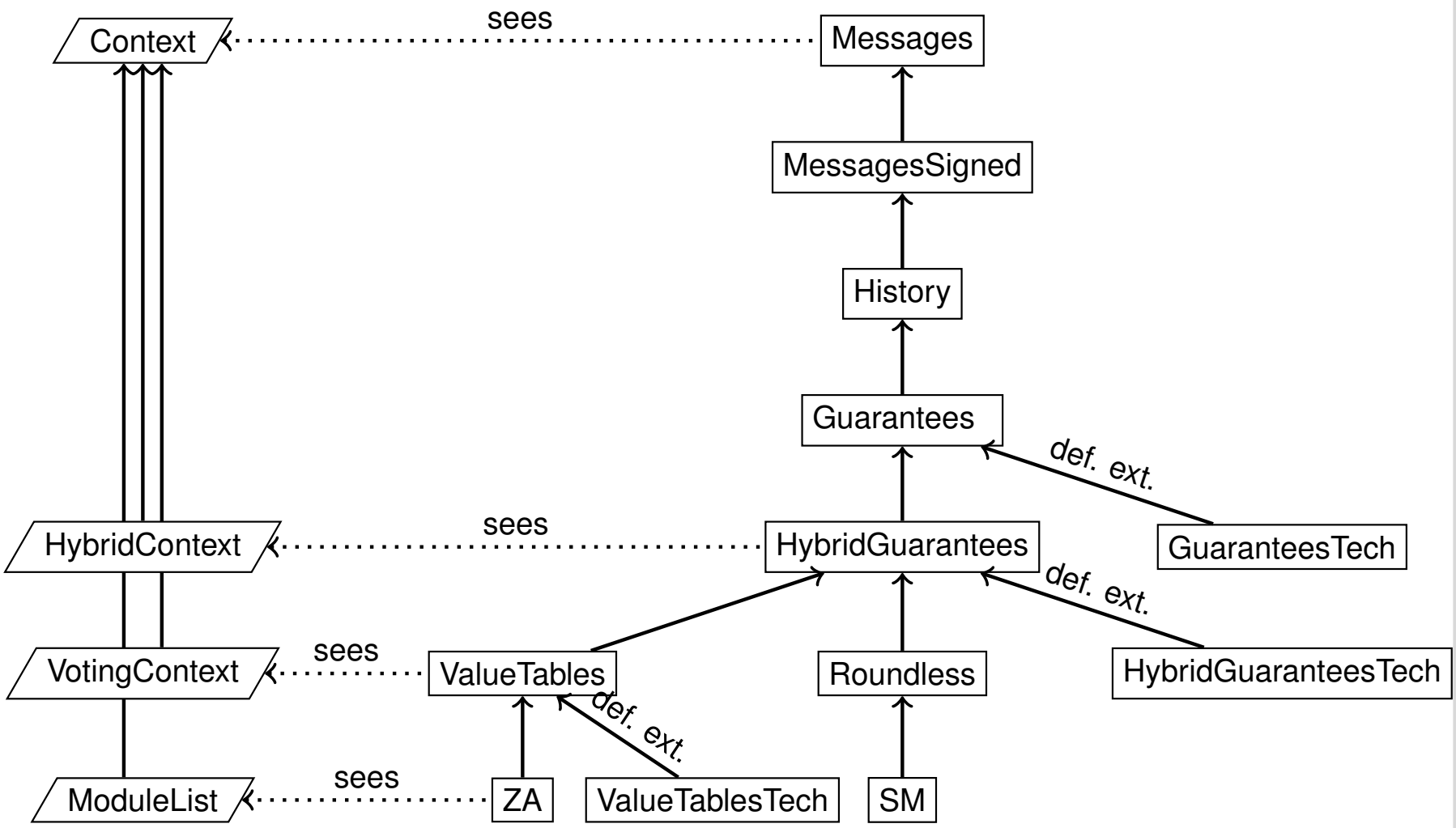
act2: $messages :\in \mathbb{P}\left(\{(r, n, v) \mid (s, r, v) \in messages\}\right)$

was: $messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

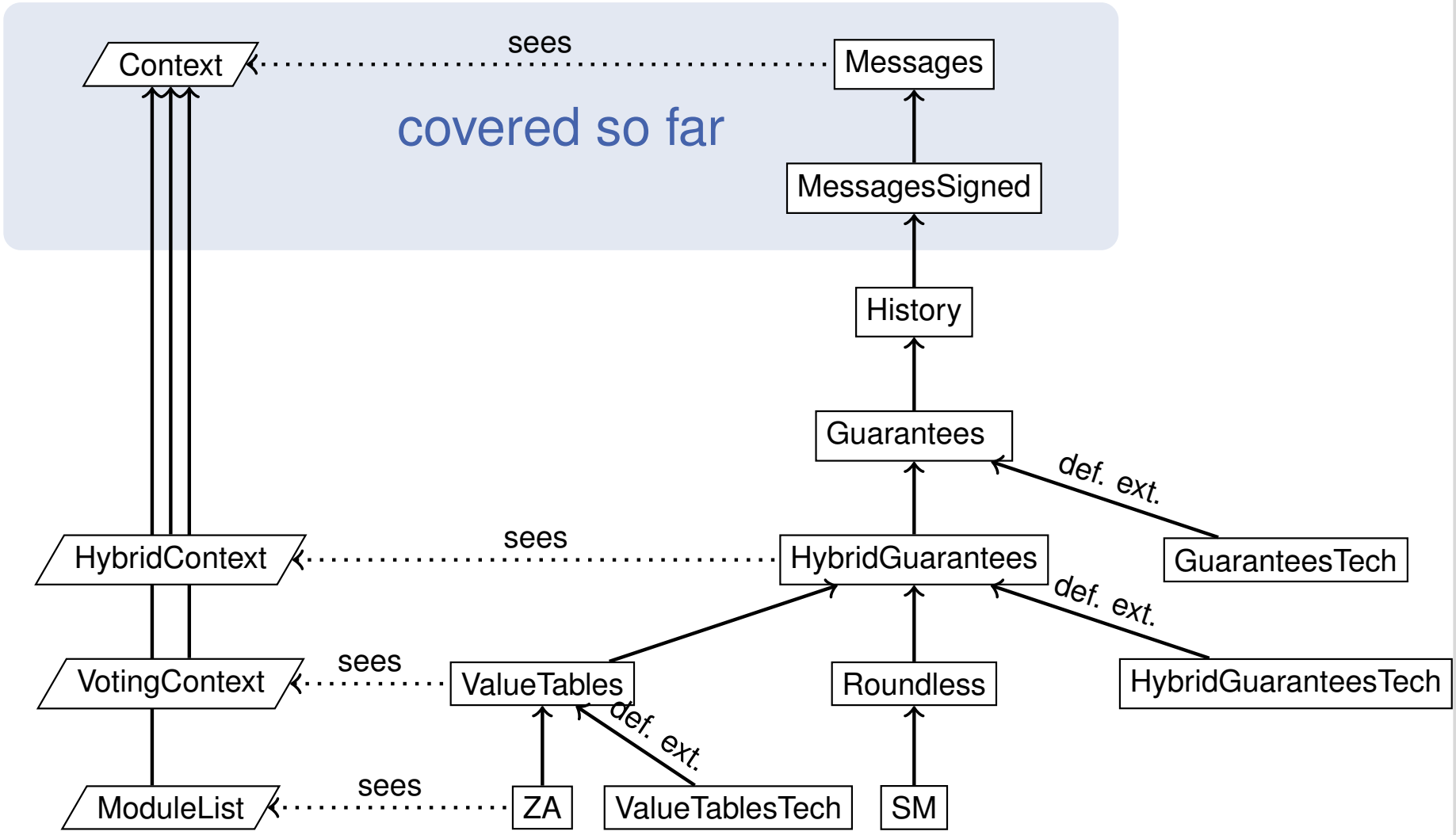
end

END

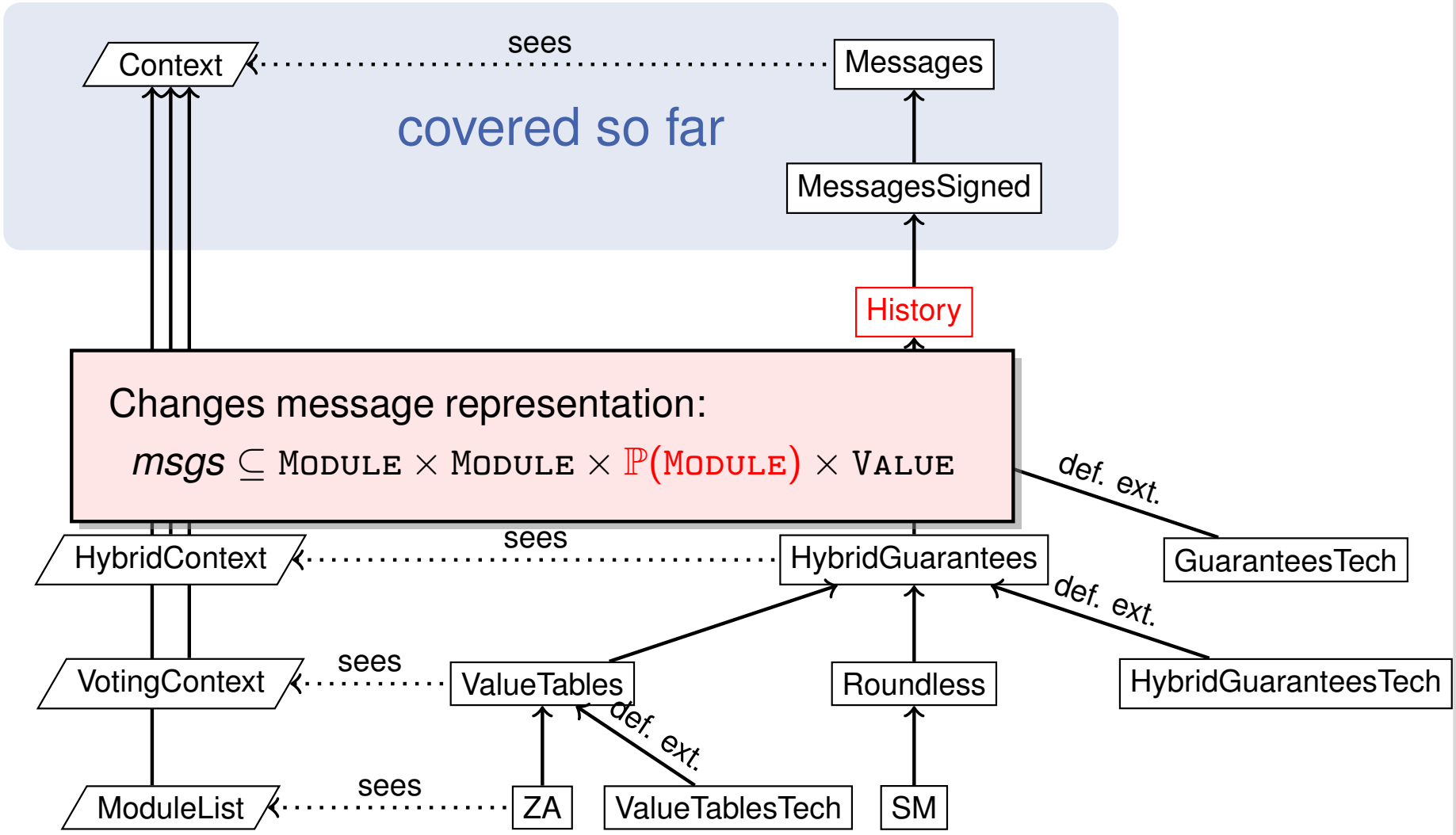
Refinement Tower



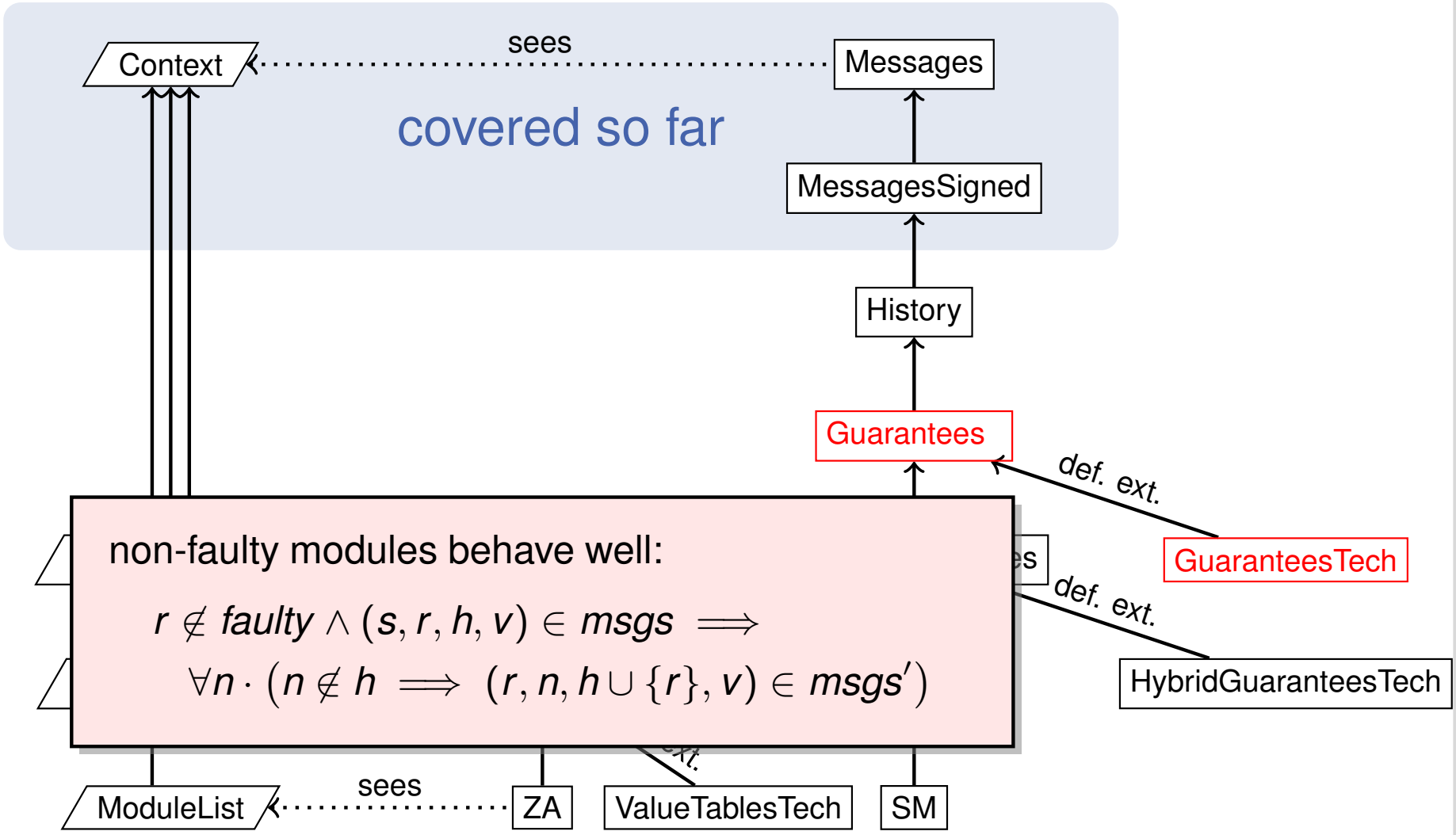
Refinement Tower



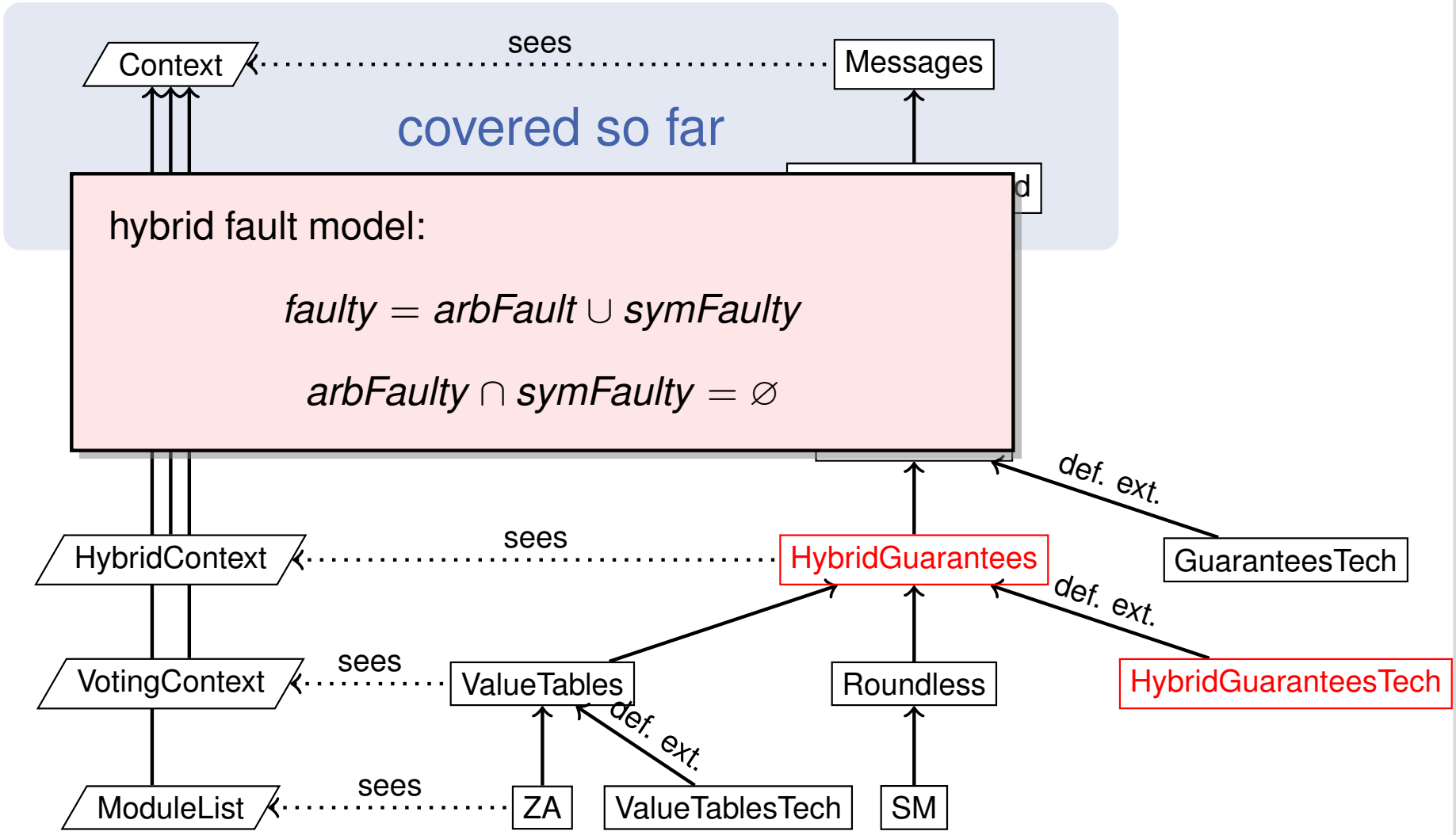
Refinement Tower



Refinement Tower



Refinement Tower



Refinement Tower

new event structure:

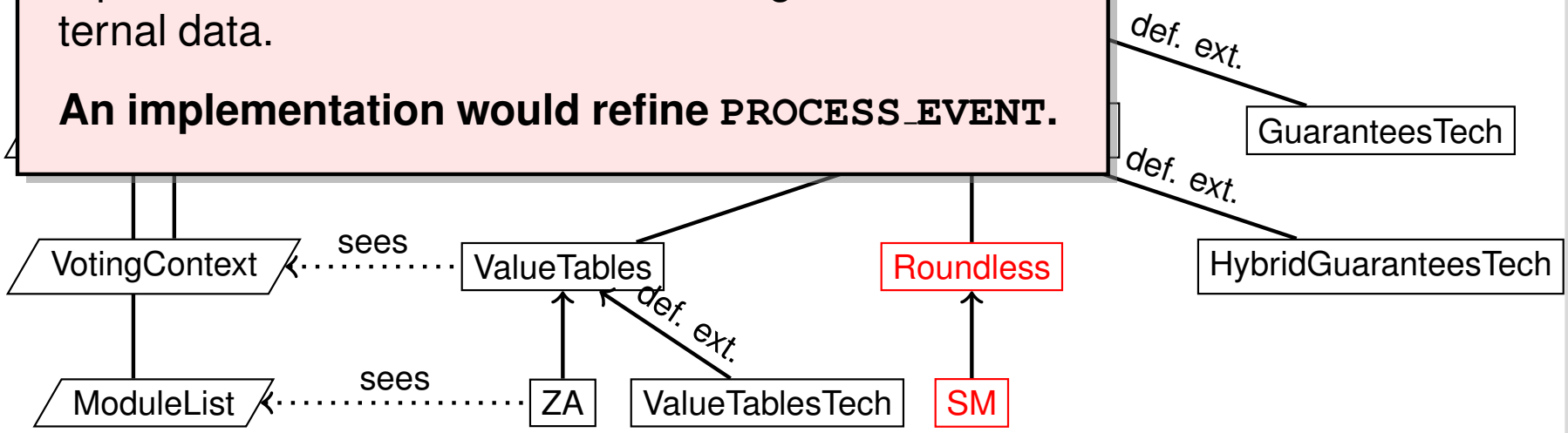
`PROCESS_EVENT refines SKIP`

modifies internal data structures (invisible to abstract machine) and

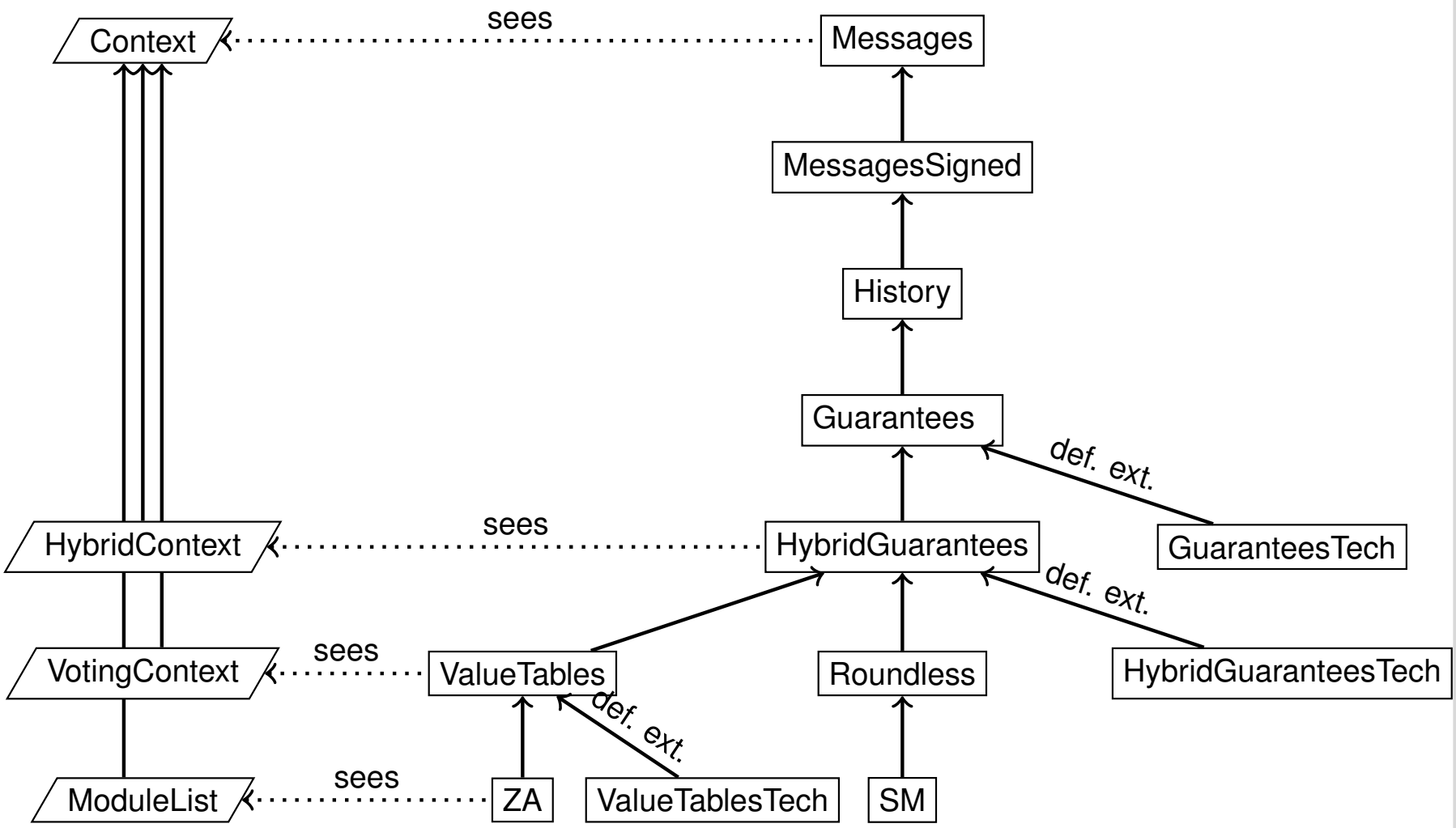
`ROUND_SWITCH refines ROUND`

reproduces the effect of a round change from the internal data.

An implementation would refine `PROCESS_EVENT`.



Refinement Tower



Agreement!

In Guarantees:

$$round \geq card(faulty) + 1 \implies \\ (\forall n, m \cdot n \notin faulty \land m \notin faulty \Rightarrow \\ collected(n) = collected(m))$$

Agreement!

In GUARANTEES:

$$round \geq card(faulty) + 1 \implies \\ \left(\forall n, m \cdot n \notin faulty \land m \notin faulty \Rightarrow \right. \\ \left. collected(n) = collected(m)\right)$$

In HYBRIDGUARANTEES:

$$round \geq card(arbFaulty) + 1 \implies \\ \left(\forall n, m \cdot n \notin faulty \land m \notin faulty \Rightarrow \right. \\ \left. collected(n) = collected(m)\right)$$

Procedure

- Rather complex proof obligations

Procedure

- Rather complex proof obligations

$\Rightarrow$ little automation

Procedure

- Rather complex proof obligations

$\Rightarrow$ little automation

- Perform manual steps to identify lemmata

Procedure

- Rather complex proof obligations

$\Rightarrow$ little automation

- Perform manual steps to identify lemmata

$\Rightarrow$ introduce as theorem invariants

Procedure

- Rather complex proof obligations

$\Rightarrow$ little automation

- Perform manual steps to identify lemmata

$\Rightarrow$ introduce as theorem invariants

- needed two-state invariants

  (e.g., messages of last round have been relayed)

Procedure

- Rather complex proof obligations

⇒ little automation

- Perform manual steps to identify lemmata

⇒ introduce as theorem invariants

- needed two-state invariants
  (e.g., messages of last round have been relayed)

⇒ concept of *definitorial extensions*:
technical refinements with extra variables,
conduct proofs there.

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23%

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23%

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23% 

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23% 

- first order set theory with relations suitable

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23% 

- first order set theory with relations suitable 

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23% 

- first order set theory with relations suitable 
- no ADT support (extension mechanism on its way)

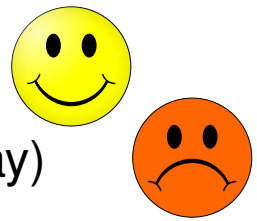
Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23%

- first order set theory with relations suitable
- no ADT support (extension mechanism on its way)



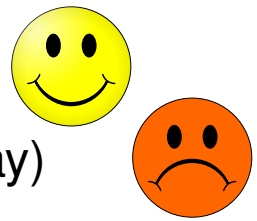
Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23%

- first order set theory with relations suitable
- no ADT support (extension mechanism on its way)
- refinement



Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23% 

- first order set theory with relations suitable 
- no ADT support (extension mechanism on its way) 
- refinement 

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23% 

- first order set theory with relations suitable 
- no ADT support (extension mechanism on its way) 
- refinement 
- good tool support (only minor bugs, →next slide)

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23% 

- first order set theory with relations suitable 
- no ADT support (extension mechanism on its way) 
- refinement 
- good tool support (only minor bugs, →next slide) 

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23% 

- first order set theory with relations suitable 
- no ADT support (extension mechanism on its way) 
- refinement 
- good tool support (only minor bugs, →next slide) 
- no sequential decomposition (unlike classical B)

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

Experiences

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 pm
Proofs:	322 proof obligations
Automation:	74/322, 23% 

- first order set theory with relations suitable  
- no ADT support (extension mechanism on its way)
- refinement 
- good tool support (only minor bugs, →next slide)  
- no sequential decomposition (unlike classical B)

Further reading: [Krenický, Ulbrich: Technical Report 2010-07]

RODIN issues

- MISSING INTERACTIVE RULES

RODIN issues

- MISSING INTERACTIVE RULES
- POs seem suited for modern SMT solvers with quantifiers (such Z3)
  (from experiences in source code verification)

RODIN issues

- MISSING INTERACTIVE RULES
- POs seem suited for modern SMT solvers with quantifiers (such Z3)
  (from experiences in source code verification)
- RODIN 1.1 seemed more stable than 1.2 (e.g., “swallowing of formulae”)