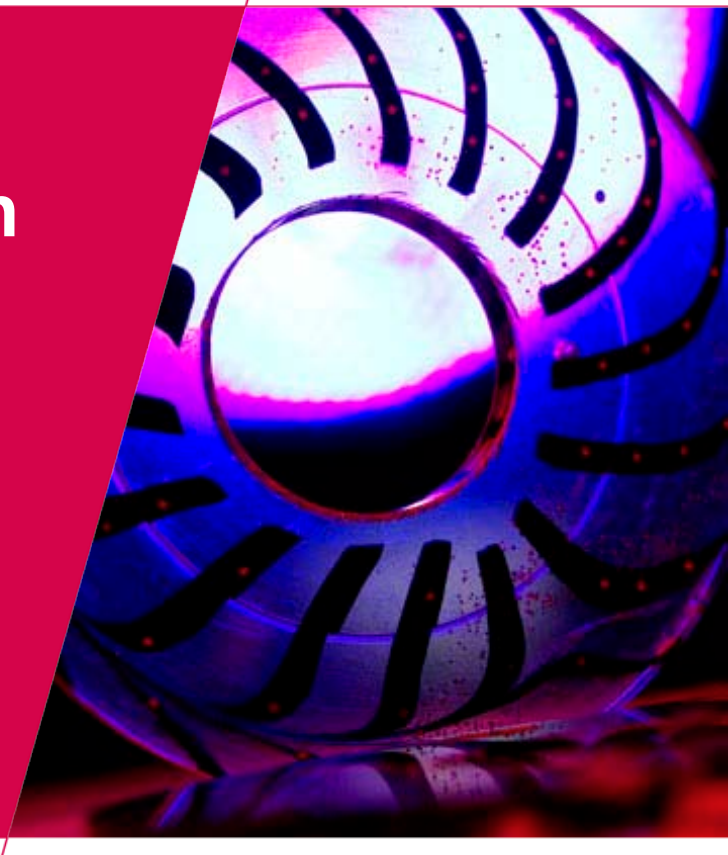
Translating synchronous to asynchronous communication and vice versa.

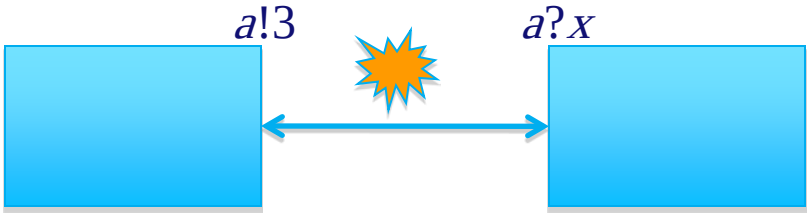
Jan Friso Groote



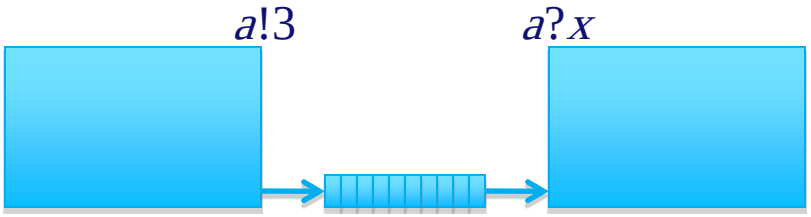
Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

About which notion of (a)synchrony are we talking?

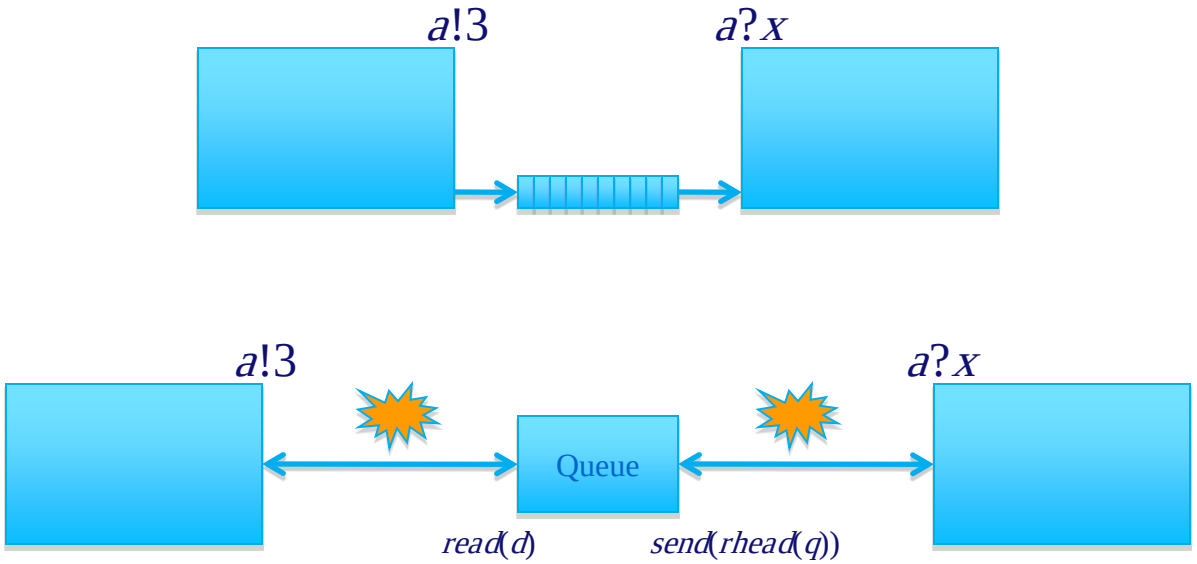


Synchronous communication: common in modelling



Asynchronous communication: common in implementation

Asynchronous → synchronous

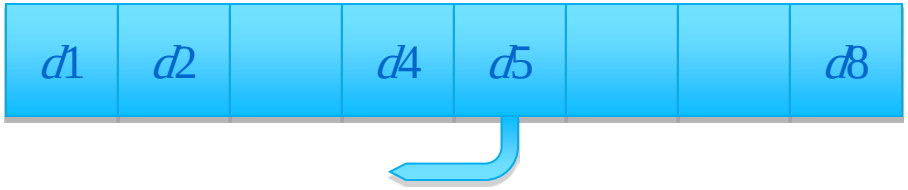


$$Queue(q{:}List(D)) = \sum_{d:D} read(d).Queue(d \triangleright q)$$
$$+ (\neg empty(q)) \rightarrow send(rhead(q)).Queue(rtail(q));$$

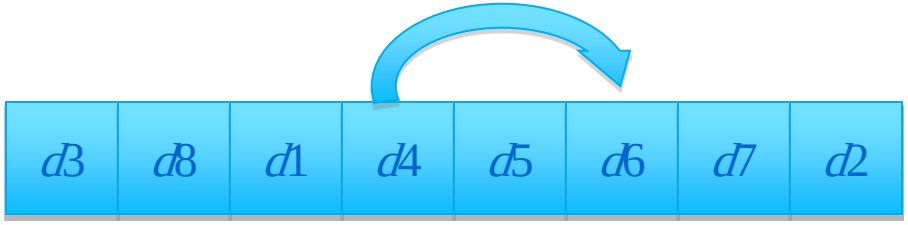
www.mcrl2.org

Asynchrony ≠ asynchrony

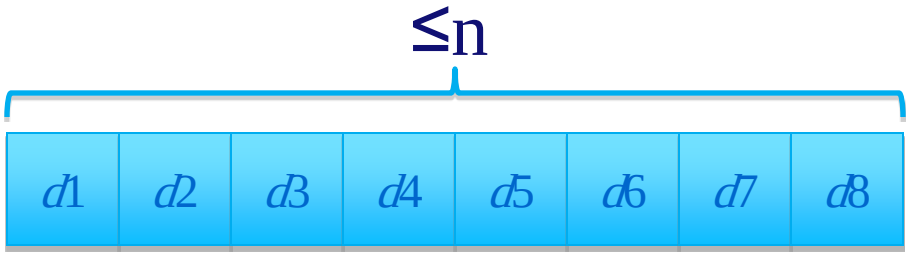
Lossy queue



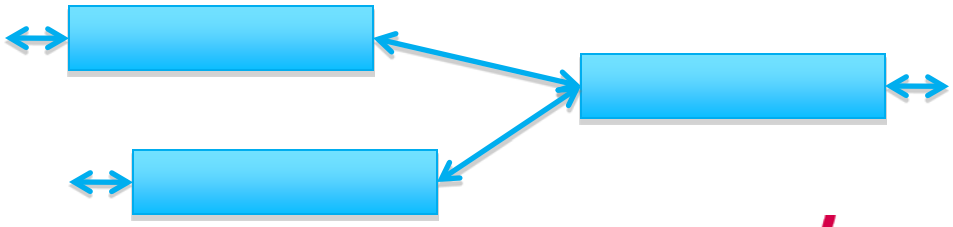
Priority queue



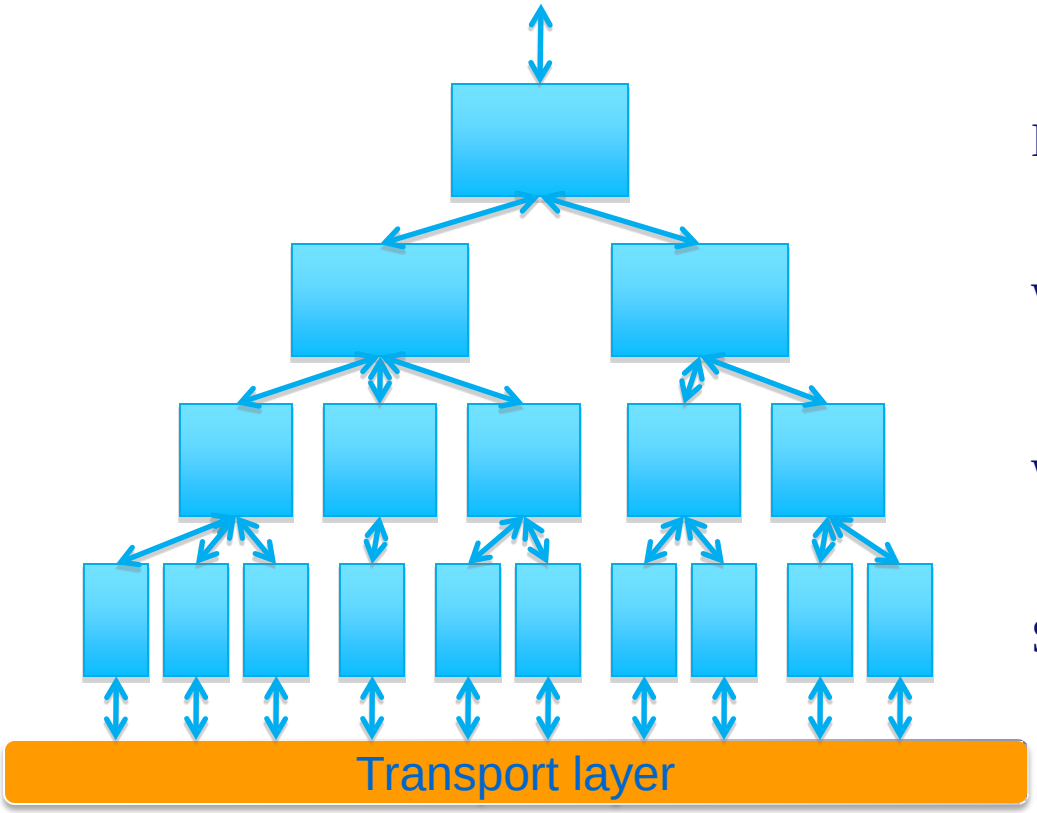
Bounded queue



Complex queue topologies



1986: Philips Computer Integrated Manufacturing architecture (Biemans, Sjoerdsma)

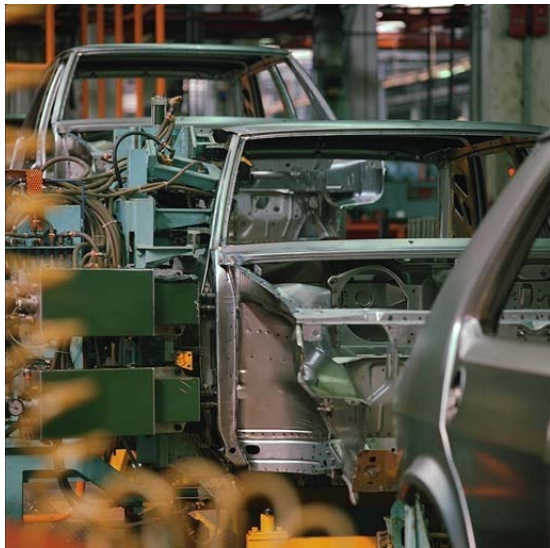


Management layer

Workcell layer

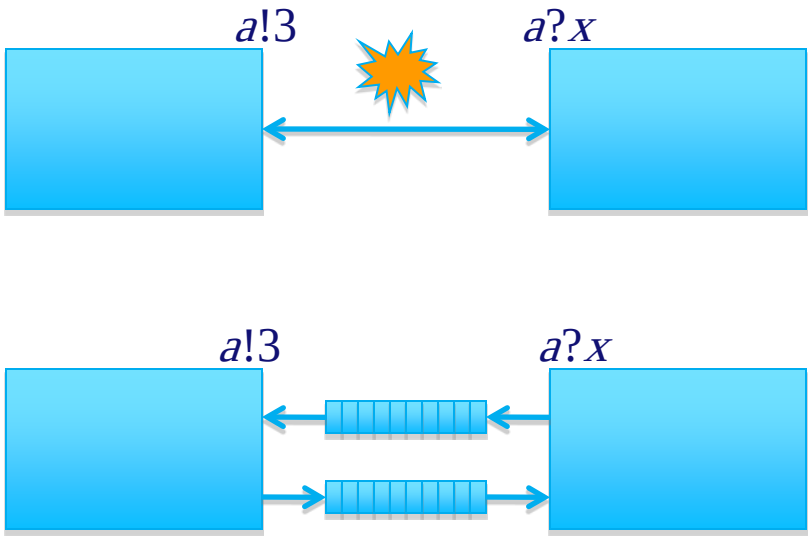
Workstation layer

Sensor/actuator layer



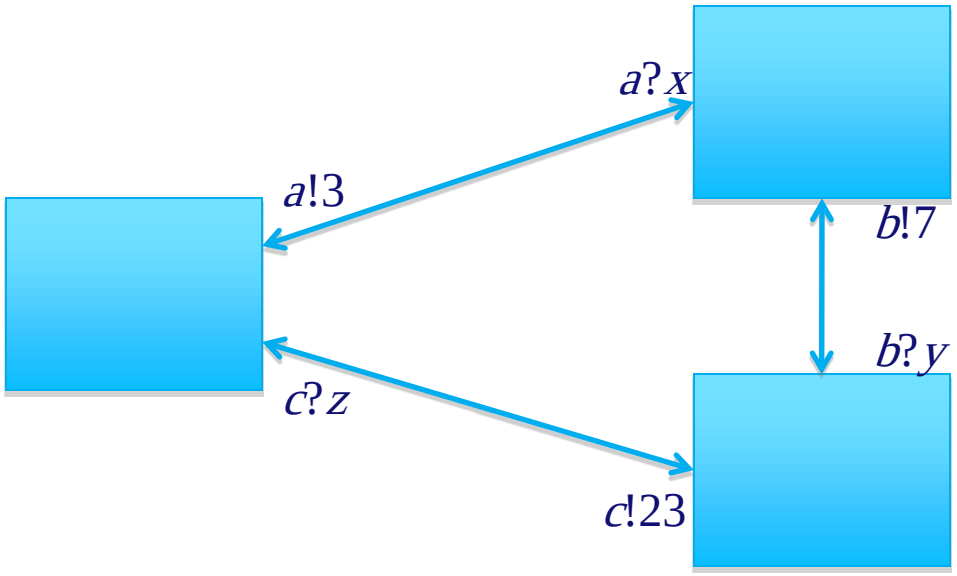
CIM architecture fully described in LOTOS
How to implement this synchronous language?

Synchronous → asynchronous



Buckley & Silberschatz 1983
Ramesh 1987
Occam & transputer

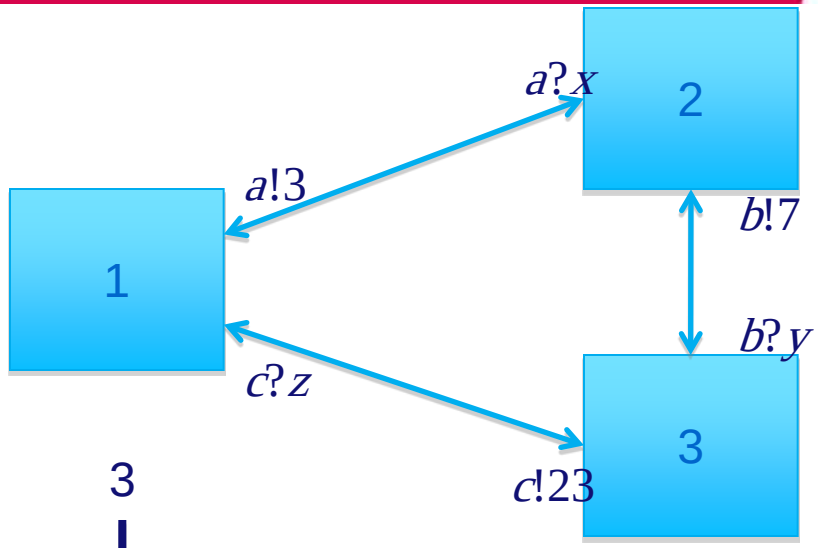
What is the problem? Conflicting interactions.

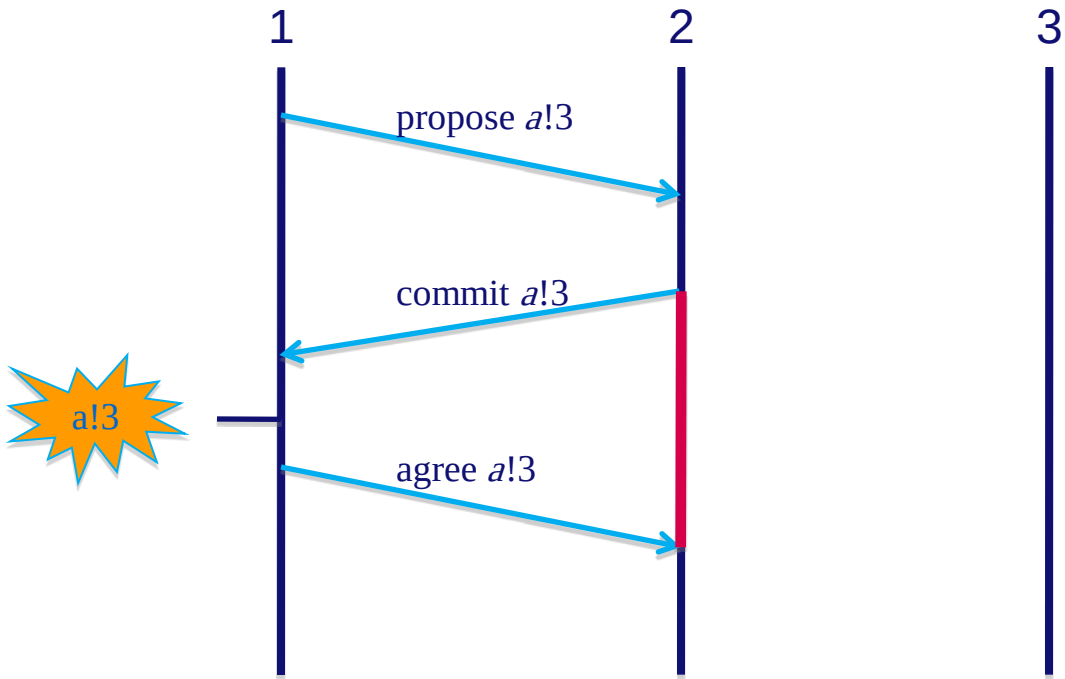


Which action will occur? $a!3$, $b!7$ or $c!23$

Solution.

1. Order all processes
2. Propose interaction to higher parties
3. Commit to proposals from lower parties
4. (Dis)agree to commits as soon as possible

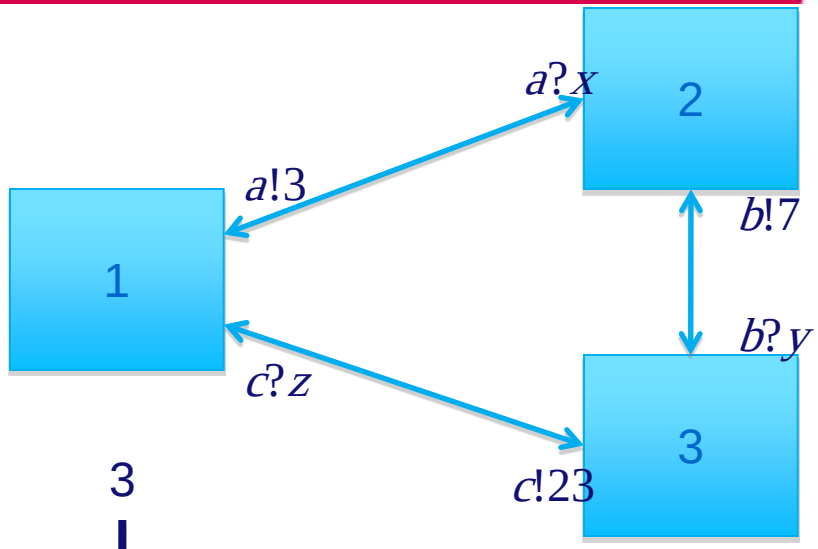


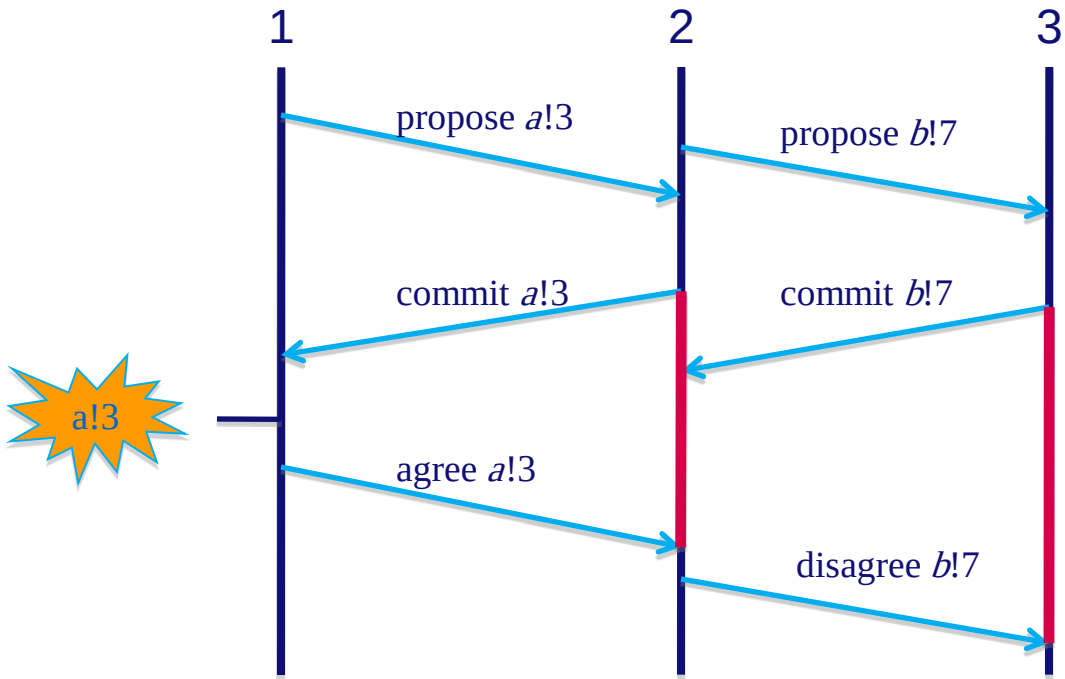


Buckley and Silberschatz, 1983
Ramesh, 1987

Solution.

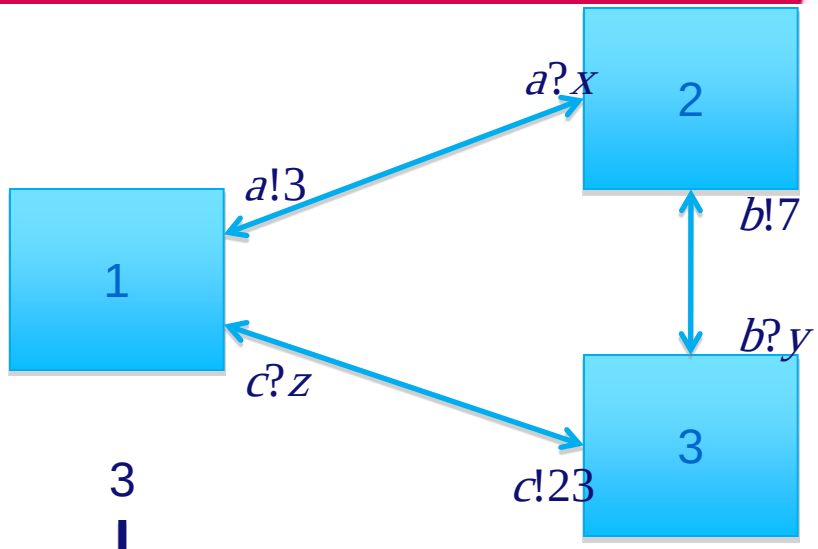
1. Order all processes
2. Propose interaction to higher parties
3. Commit to proposals from lower parties
4. (Dis)agree to commits as soon as possible

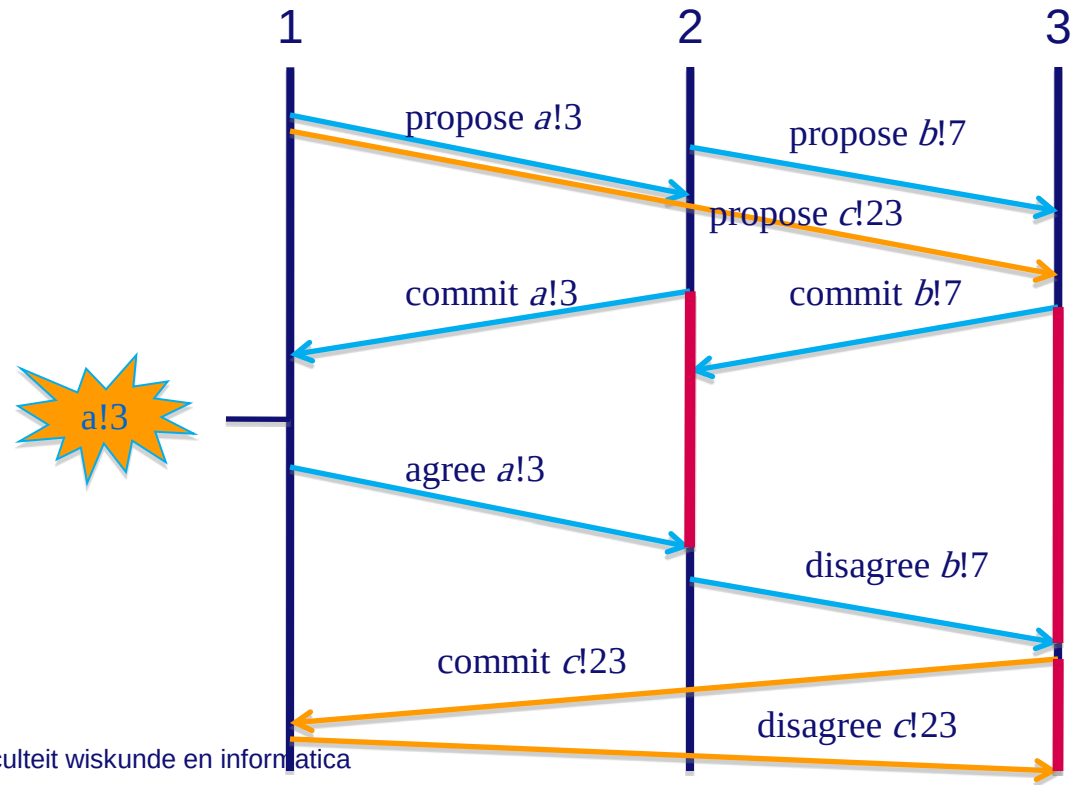




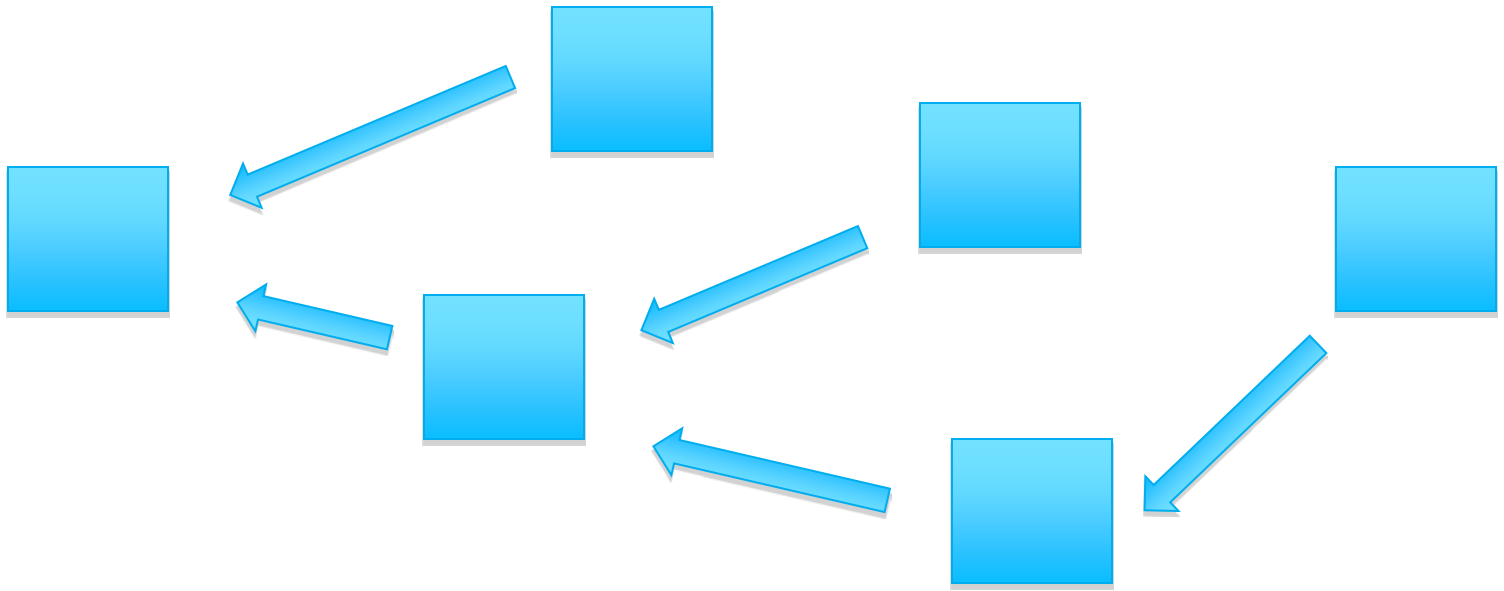
Solution.

1. Order all processes
2. Propose interaction to higher parties
3. Commit to proposals from lower parties
4. (Dis)agree to commits as soon as possible





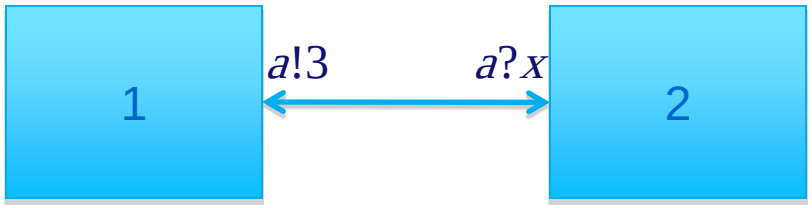
Commit dependencies are non cyclic.

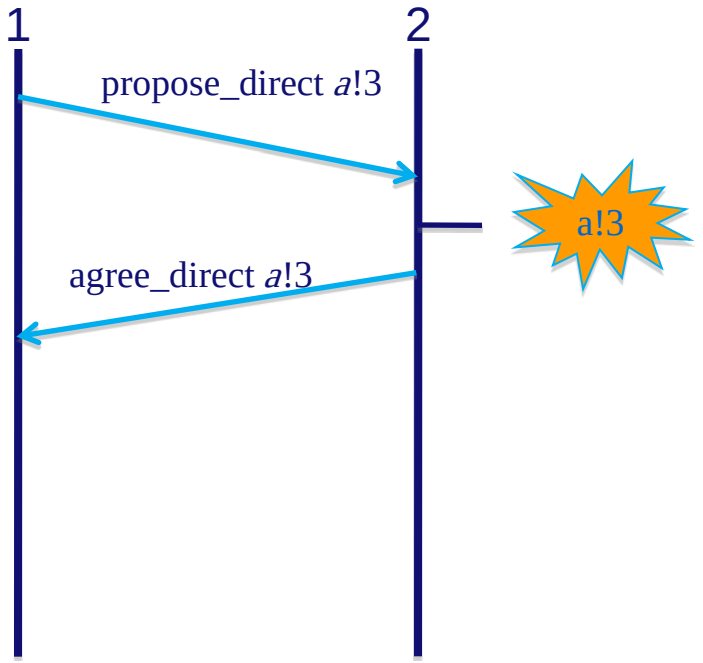


Waiting commits to be (dis)agreed.

Complexity: three messages per proposed interaction.

Optimisations are often possible: 2 messages.



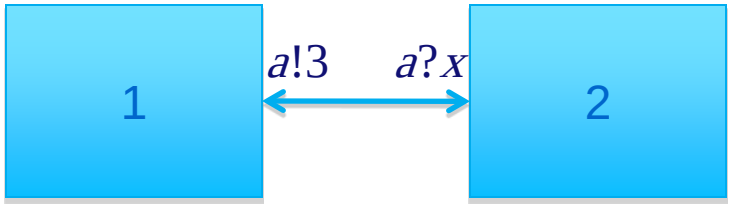


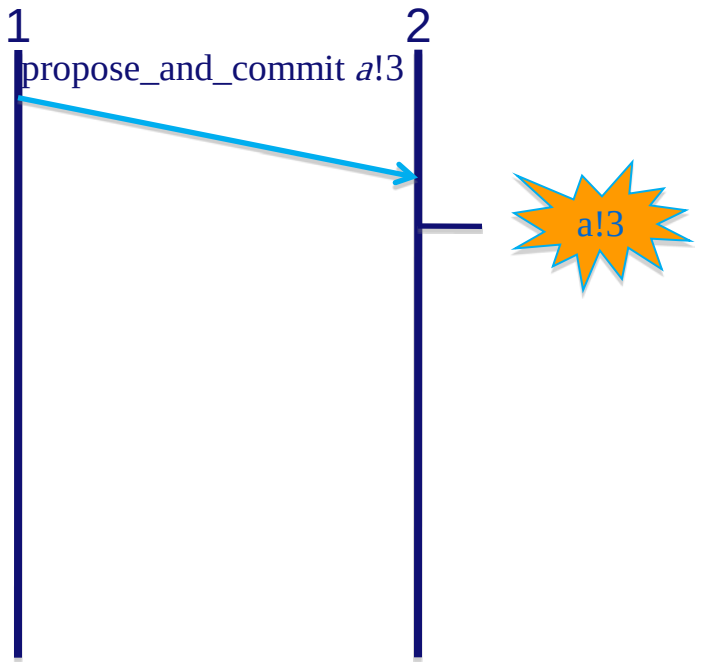
a!3 does not occur in a choice.

Theorem: Implementation by 2 messages preserves testing equivalence (LOTOS, 1988)

This is the communication scheme in OCCAM (Transputer)

Optimisations are often possible: 1 message.





a!3 does not occur in a choice, and has no subsequent activity.

Typical example: p || *a*!3.δ

Conclusion.

Asynchronous communication

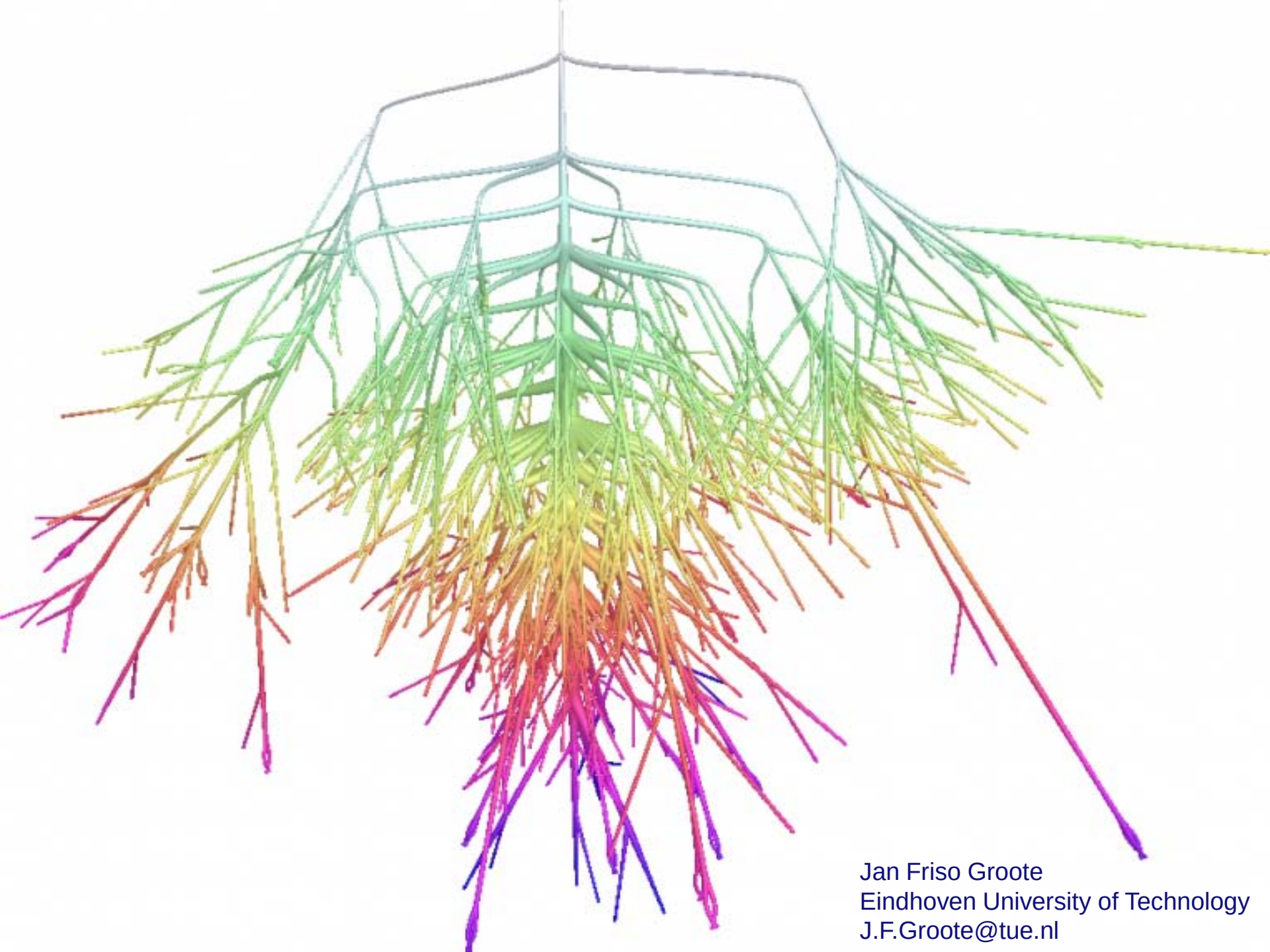
$\approx$

Synchronous communication

Challenge

Challenge: Implement synchronous models efficiently in an asynchrous, distributed way

obtain_resource1.Use_resource1
+
obtain_resource2.Use_resource2
+
receive_cancel_message.Do_something_else
+
…… (further alternatives)



Jan Friso Groote
Eindhoven University of Technology
J.F.Groote@tue.nl