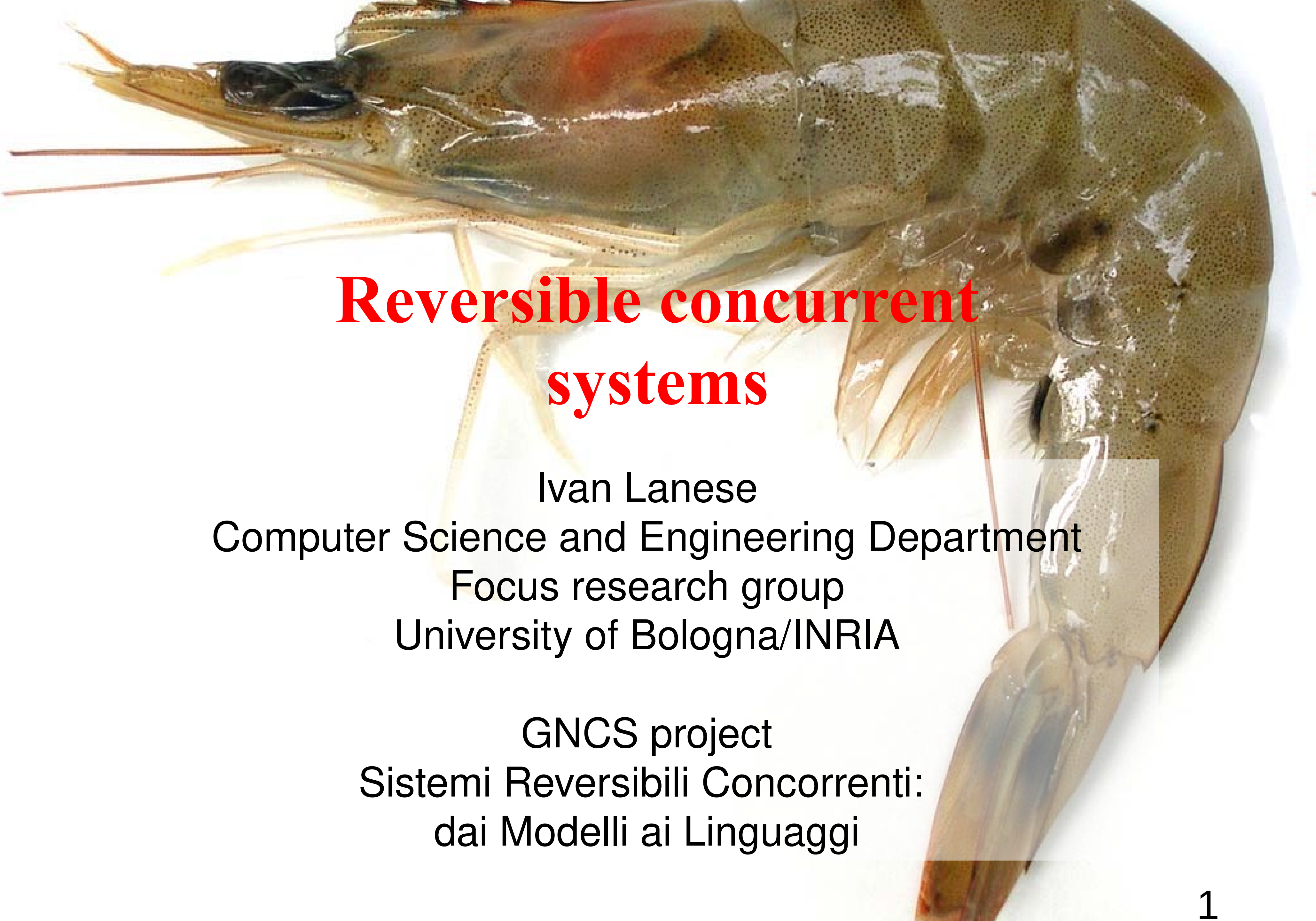


Reversible concurrent systems

Ivan Lanese
Computer Science and Engineering Department
Focus research group
University of Bologna/INRIA

GNCS project
Sistemi Reversibili Concorrenti:
dai Modelli ai Linguaggi

Roadmap

- Reversible computing
- Debugging Erlang programs
- Petri nets vs event structures
- Conclusion

Roadmap

- Reversible computing
- Debugging Erlang programs
- Petri nets vs event structures
- Conclusion

What is reversibility?

The possibility of executing a computation both in the standard, forward direction, and in the backward direction, going back to a past state

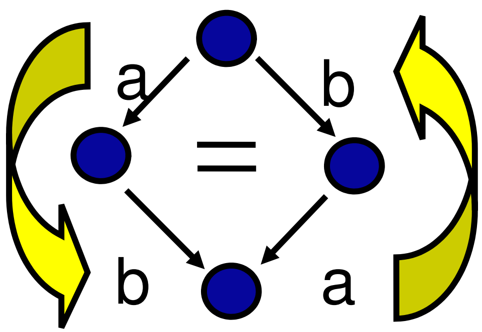
- In some areas systems are naturally reversible: biology, quantum computing, …
- In other areas making systems reversible can be useful
  - Undo option in text editors
  - Debugging, robotics, ...

Reversibility in concurrent systems

- Reversibility in a sequential setting: recursively undo the last action
- In concurrent systems execution of different actions may overlap in time
  - No uniquely defined last action
  - Actions form a partial order, not necessarily a total order
- Different approaches to reversibility exist
- We follow causal-consistent reversibility [Danos & Krivine, CONCUR 2004]

Causal-consistent reversibility

- Based on causality instead of time
- Causal dependencies must be respected
  - First reverse the consequences, then the causes
- Independent actions are reversed independently



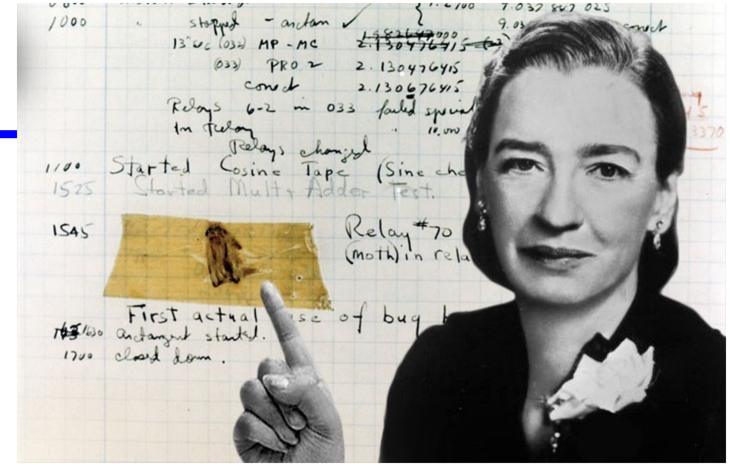
Reversibility in our project

- We considered many aspects of reversibility, both foundational and applicative
- In the talk I will focus on two contributions
  - How to apply causal-consistent reversibility to the debugging of Erlang programs?
    - Ongoing effort since many years
    - By myself, C. Sacerdoti Coen, G. Vidal, ...
  - How to extend the relation between event structures and Petri nets to the reversible setting?
    - By C. A. Mezzina, G. M. Pinna, E. Melgratti, I. Ulidowski

Roadmap

- Reversible computing
- Debugging Erlang programs
- Petri nets vs event structures
- Conclusion

Debugging



- Debugging amounts to find the wrong line of code (bug) causing a visible misbehavior
- The bug precedes and causes the misbehavior
  - Quite natural to use reversibility to go back from the misbehavior to the bug

Reversible debugging

- Sequential reversible debugging is well understood
  - Gdb (since 2009), Microsoft time-travel debugger, ...
- Concurrent reversible debugging not so developed
  - Most approaches just linearize the execution
  - Causal information is lost
- Can we use causal-consistent reversibility?

Causal-consistent debugging

- Introduced in [Giachino, Lanese & Mezzina, FASE 2014]
- Allows one to explore a concurrent computation back and forward
  - Any action can be undone provided its consequences have been undone beforehand
- Which action to undo can be selected by the user or by a scheduler
- But we can do better

Debugging and causality

- Standard debugging procedure:
  1. Observing an unexpected behavior
  2. Finding in the code the instruction that caused it
  3. Correcting the instruction
- Causal-consistent reversibility naturally tracks lot of causal information
- This information can be used to drive step 2 above
- Debugging strategy: follow causality links backward from the misbehavior to the bug
- We can use the roll operator to this end

Causal-consistent debugging: roll

- The roll operator allows one to undo a selected past action, including all and only its consequences
- Minimal set of undos needed to undo the selected action in a causal-consistent way
- Many interfaces for it:
  - N actions in a given process
  - Last assignment to a given variable
  - Send of a given message

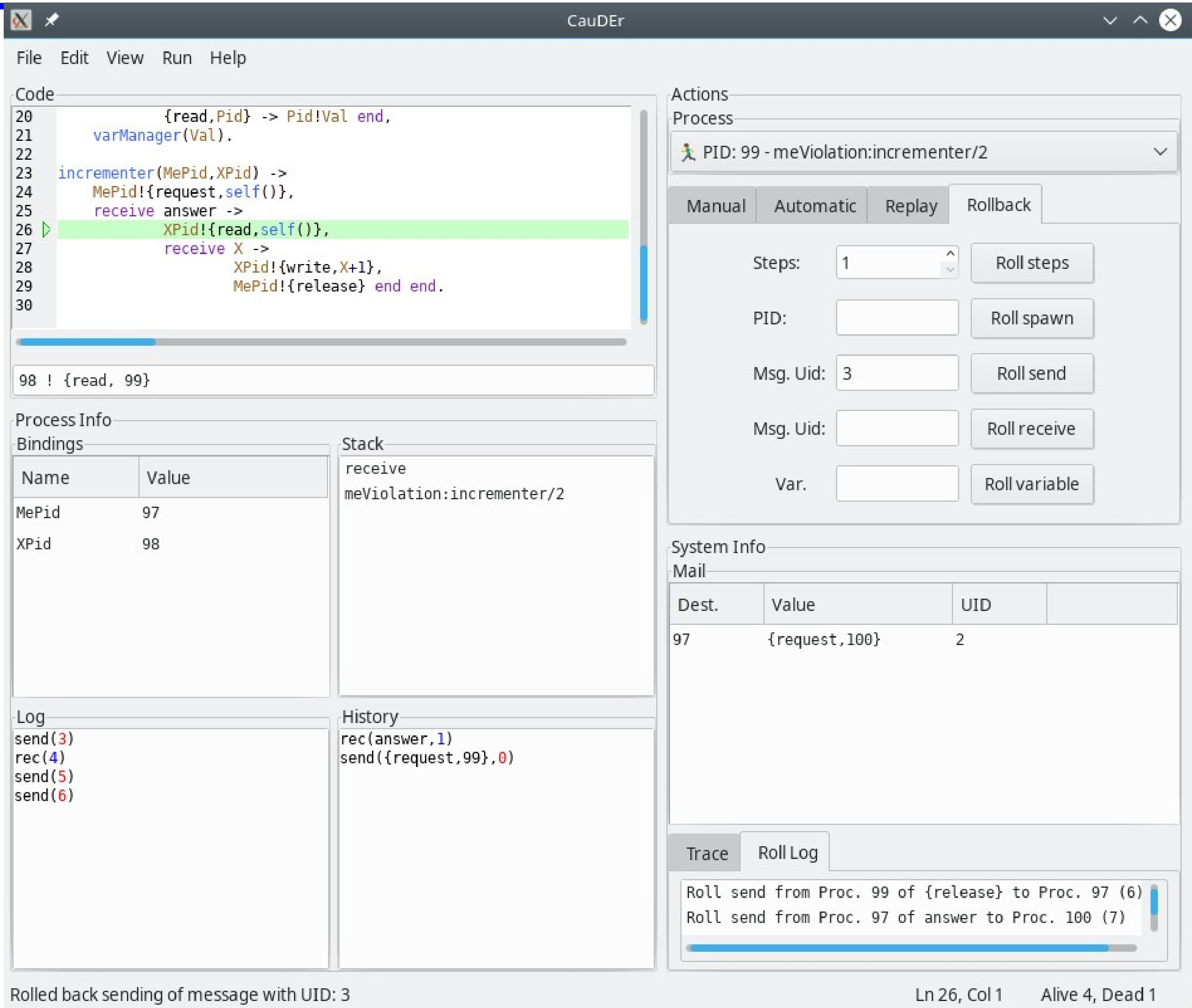
Causal-consistent roll at work

- The programmer executes the program and finds some unexpected behavior
- The roll allows him to find automatically the instruction that immediately caused the misbehavior
- Two possibilities:
  - The found instruction is wrong: bug found
  - The found instruction gets wrong data from previous instructions: iterate
- One can explore the tree of causes, navigating from one process to the other

CauDEr

- Causal-consistent Debugger for Erlang
- Applies the approach outlined above to a fragment of Erlang
  - Functional and concurrent language
  - Based on message passing
  - Used in mainstream applications such as some versions of Facebook chat
- https://github.com/mistupv/cauder-v2
- Support for further (non trivial) constructs in Erlang has been added during the project

CauDEr interface

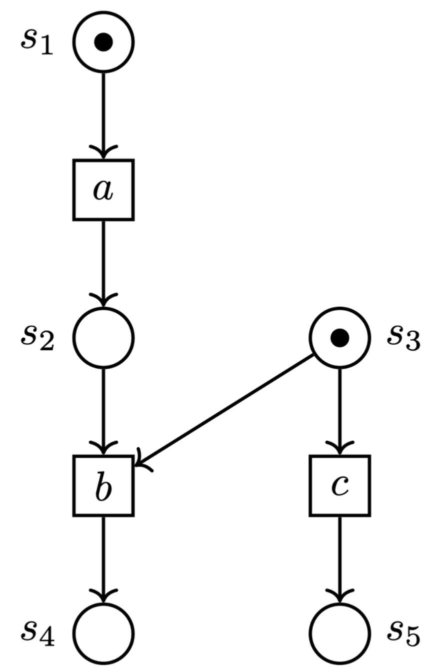


Roadmap

- Reversible computing
- Debugging Erlang programs
- Petri nets vs event structures
- Conclusion

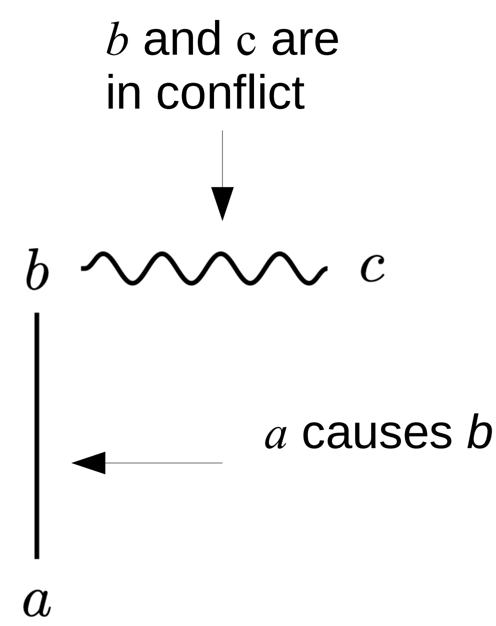
Petri nets

- Operational model for concurrency
- Based on tokens that enable transitions
- The one in figure is actually an occurrence net (ON)
  - No cycles and max 1 token per place
  - It is always clear where a token comes from
  - Plays well with reversibility
  - Can represent any net via unfolding



Event structures

- Denotational model for concurrency
- Based on events and relations among them
- These are actually prime event structures (PES)
  - Only causation and conflict relations
  - Variants have other relations
- Prime event structures correspond to occurrence nets
  - Classical result by Winskel

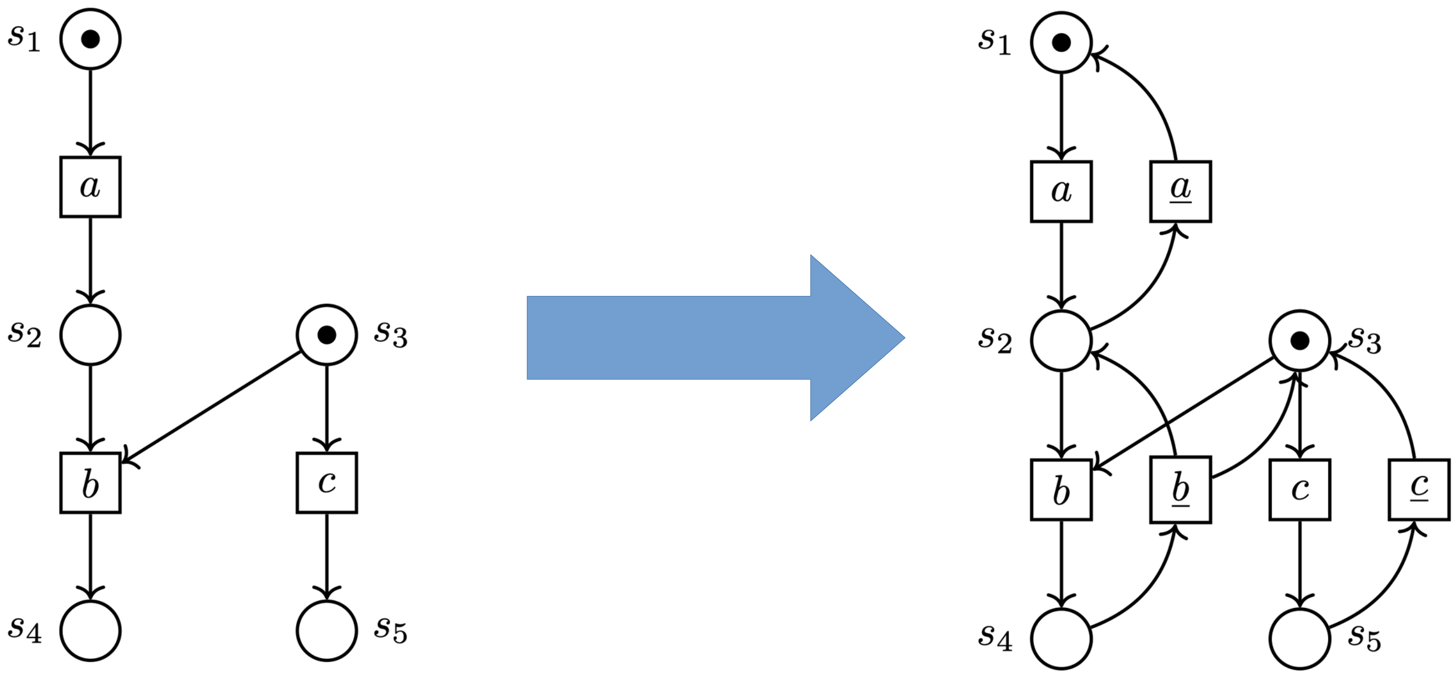


Reversible PES

- Extend PES with reversibility
  - Not only causal-consistent
- Not all the events need to be reversible
- Introduce:
  - Reverse causality $\prec$: a forward event is a cause of a backward one
  - Prevention $\triangleright$: a forward event forbids a backward one

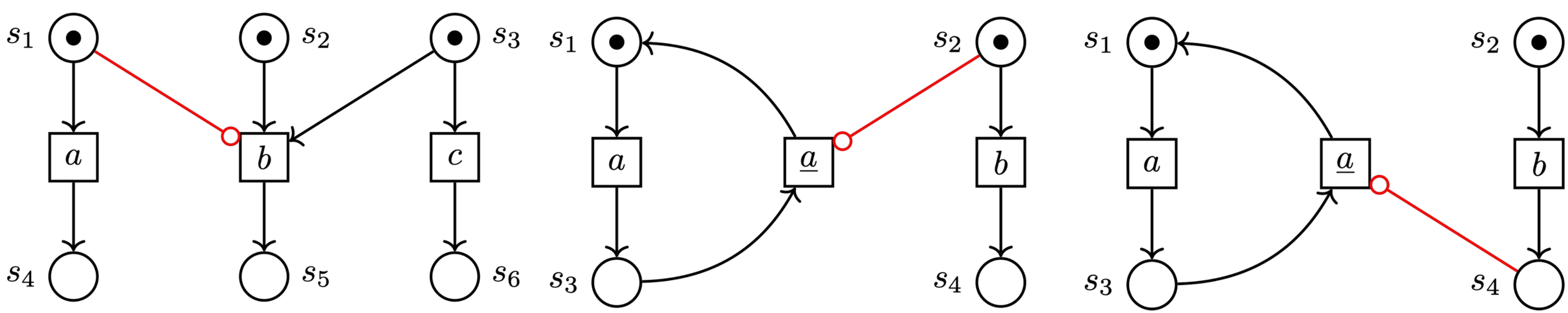
From ON to reversible ON

- Adding reverse transitions in ON gives rise to causal-consistent reversibility



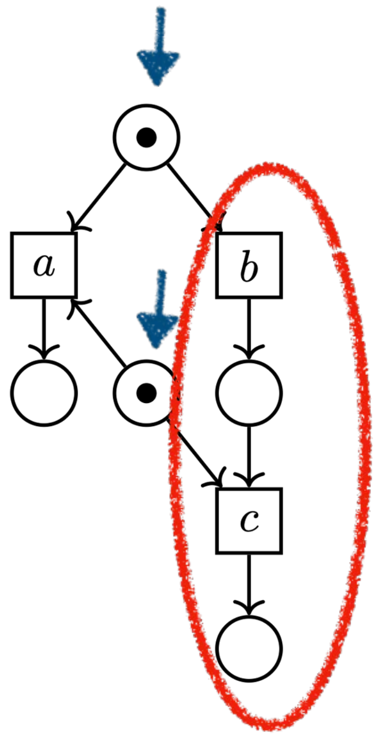
Modeling rPES with Petri nets

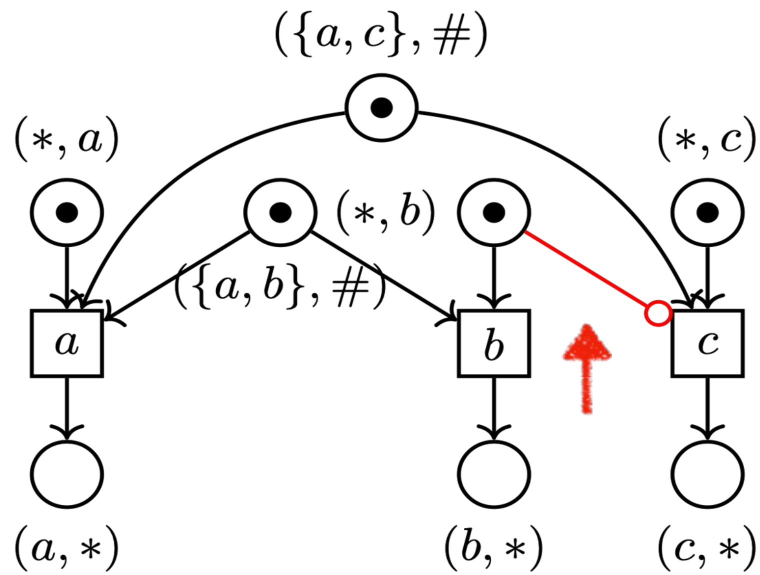
- We need to introduce inhibitor arcs
- Inhibitor arcs can model causality as well
- We can reduce to "flat" nets
- We call them Causal Nets (CN)
- Reversible Causal Nets (rCN) extend causal nets with reverse actions and additional inhibitor arcs



$a < b \quad b\#c$ $\qquad$ $b \prec \underline{a}$ $\qquad$ $b \triangleright \underline{a}$

From ON to CN

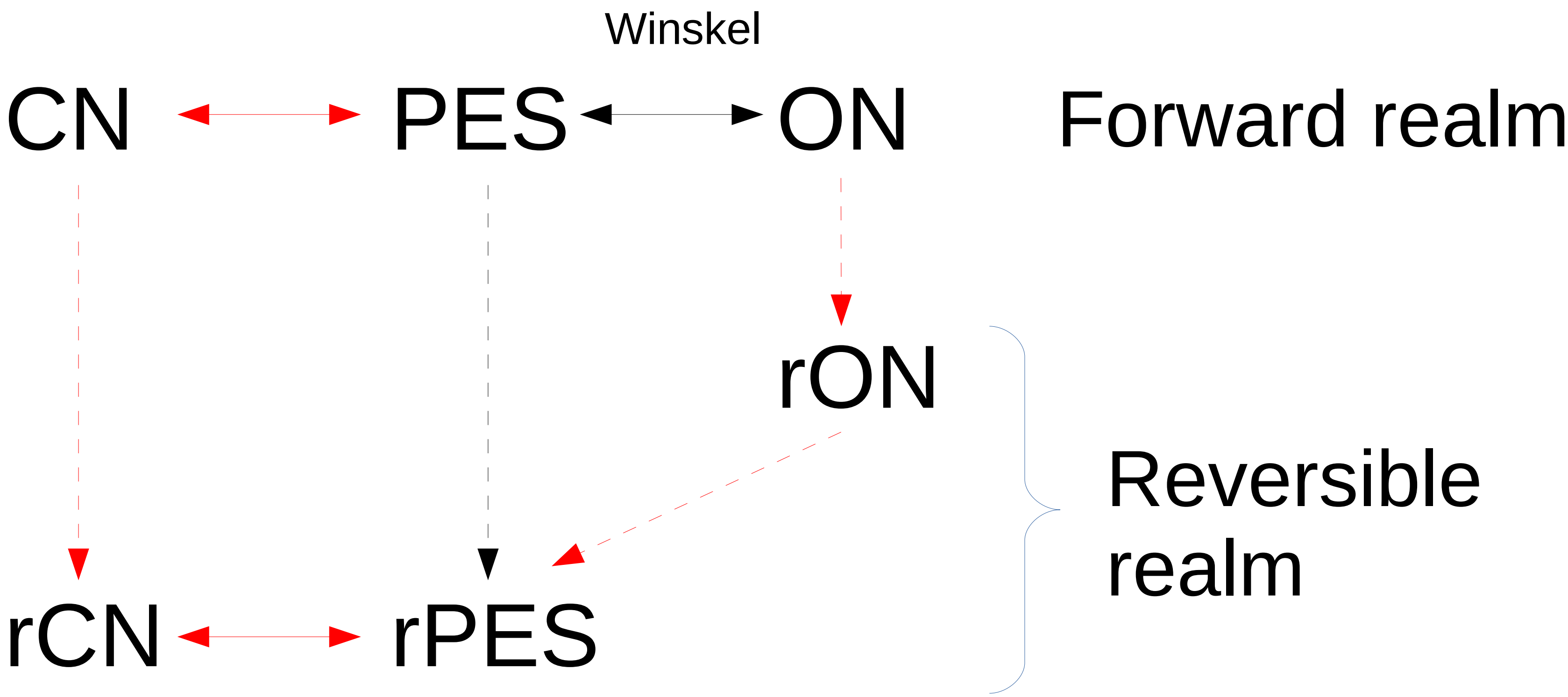




a # b **a < c**

a # c

Graphical summary of results



Roadmap

- Reversible computing
- Debugging Erlang programs
- Petri nets vs event structures
- Conclusion

Other contributions

- Understanding the interplay between reversibility and time
- Developing techniques for axiomatic reasoning on reversible processes
- Understanding reversibility in Markov chains

Conclusion

- Reversibility is a niche area with many possible applications and open questions
- Studied by both computer scientists and mathematicians (and not only)
- If curious, Reversible Computation conference will be held in Urbino, on July 5-6
  - Online participation will also be allowed

Thanks!

Questions?