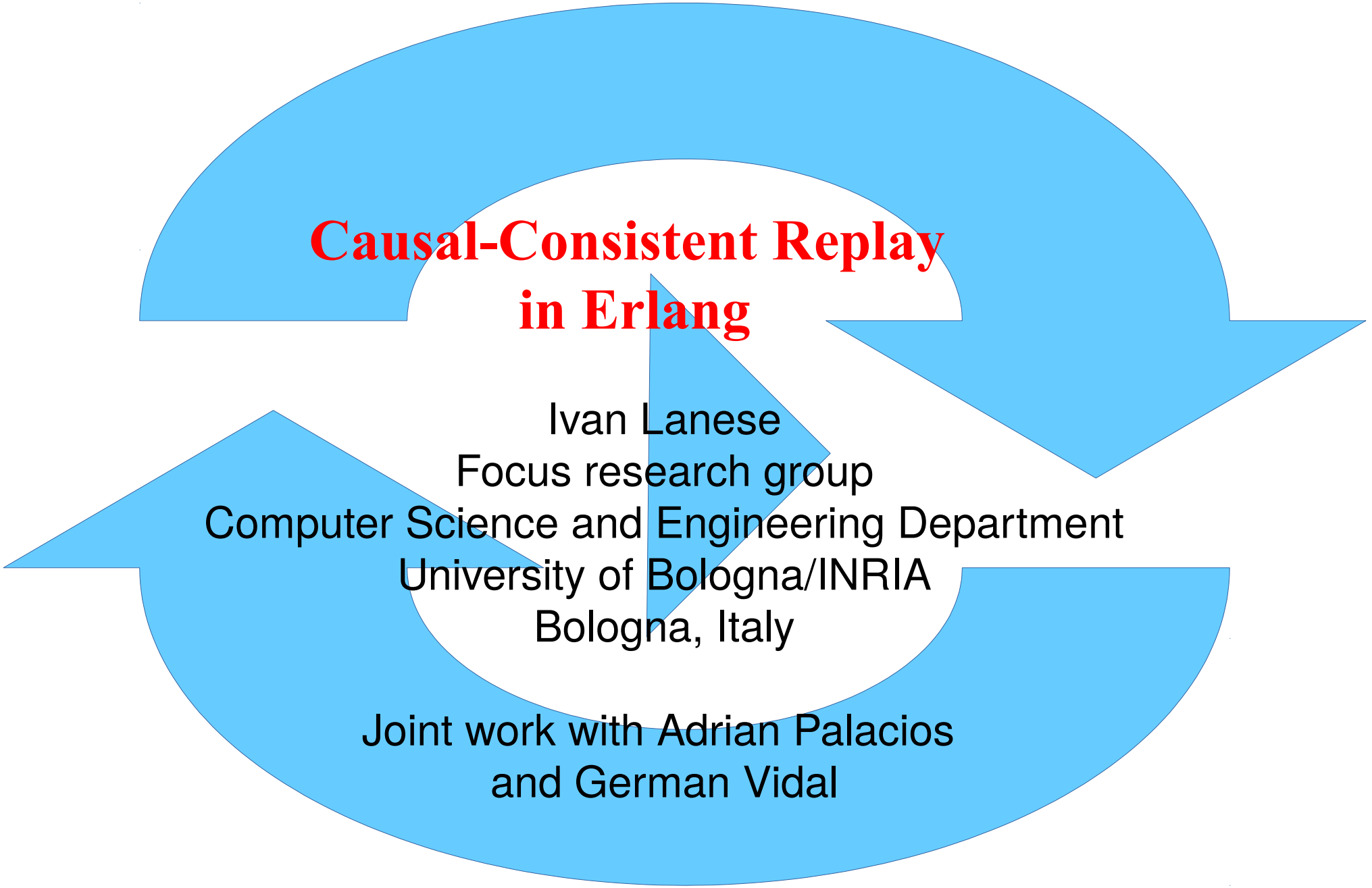


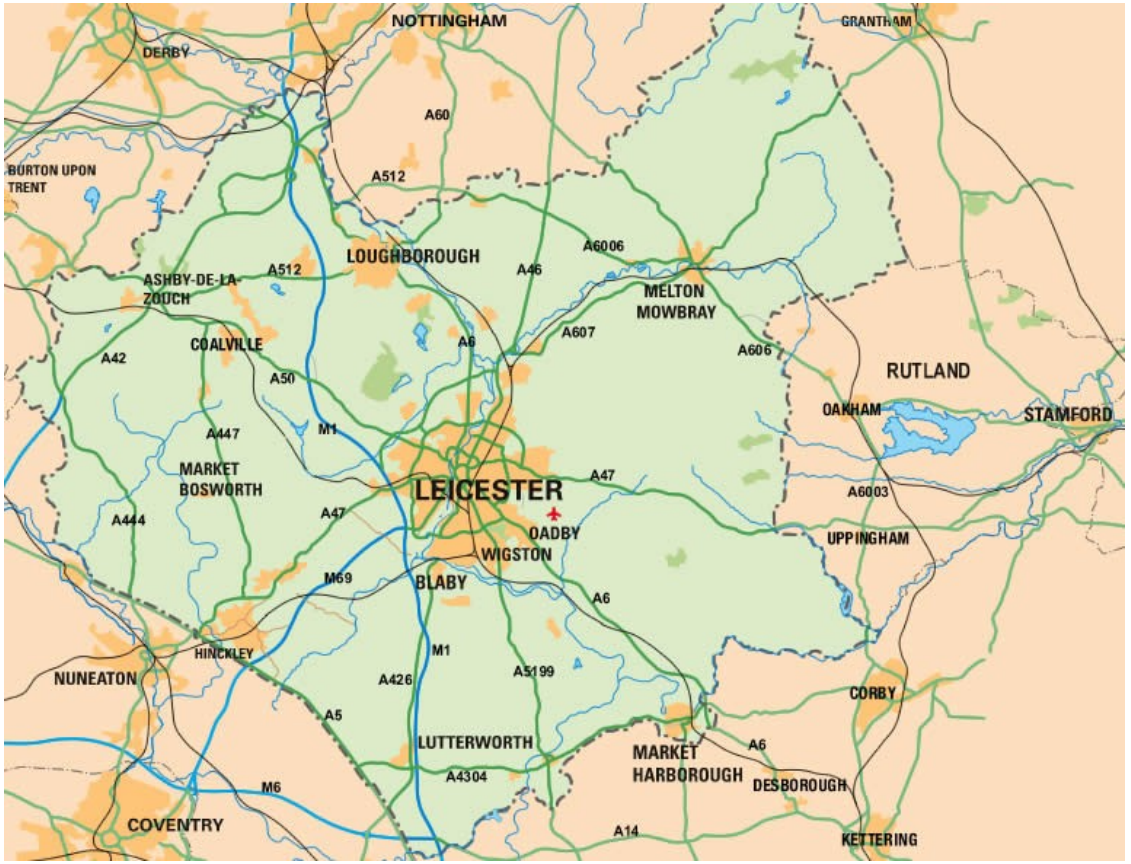
Causal-Consistent Replay in Erlang

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

Joint work with Adrian Palacios
and German Vidal

Roadmap

- Causal-consistent reversible debugging
- Causal-consistent replay
- Extension of CauDEr
- Future directions



Why debugging?

- Debugging of concurrent and distributed systems is one of the WG4 case studies
- Debugging of concurrent systems is particularly challenging
  - Misbehaviors may depend on the scheduling
  - Bugs may be in a different process than the one showing the misbehavior
- Reversibility can play a role
- We presented in past meetings an approach based on causal-consistent reversibility targeting Erlang
  - Prototyped as the CauDEr debugger

Causal-consistent reversibility

- Since [Danos & Krivine, CONCUR 2004] causal-consistent reversibility is the main notion of reversibility for concurrent systems
  - Any action can be undone, provided that its consequences (if any) are undone beforehand
  - Concurrent actions can be undone in any order, but causal-dependent actions are undone in reverse order

Causal-consistent debugging

- Main operator: causal-consistent rollback
- Allows one to undo any past action, including all and only its consequences
- The choice of the action to undo depends on the misbehavior we want to investigate
- For instance:
  - Wrong value in variable X: **roll var** X executes a causal-consistent rollback of the last assignment to X
  - Unexpected message with identifier 5: **roll send** 5 executes a causal-consistent rollback of the sending of message 5

Erlang and Core Erlang

- We target the Erlang language
- Functional language
- Based on the actors concurrency model
  - Processes are actors that communicate asynchronously by message passing
  - Each process has its own local mailbox
  - No shared memory
- During compilation, Erlang is first translated to Core Erlang

CauDEr: Causal-Consistent Debugger for Erlang

- Only a prototype to test our ideas
- Supports a subset of Core Erlang
  - Sequential language + actor primitives
- Written in Erlang
- Available at https://github.com/mistupv/cauder
- Original description and underlying theory in [Lanese, Nishida, Palacios & Vidal, FLOPS 2018]

Shortcomings of CauDEr

- (Currently there are a lot of them, but here we concetrate here just on two of them)
- CauDEr allows the user to go back in the execution looking for the causes of a given misbehavior but…
- If the misbehavior occurs in an actual execution in production environment there is no way to reproduce it inside the debugger
- If during debugging one goes too much backward there is no way to go forward again with the guarantee to replay the same misbehaviors
- Causal-consistent replay solves both these problems

Causal-consistent rollback

- It allows one to undo any action, provided that its consequences (if any) are undone beforehand
- Concurrent actions can be undone in any order, but causal-dependent actions are undone in reverse order

Causal-consistent replay

- It allows one to redo any action, provided that its causes (if any) are redone beforehand
- Concurrent actions can be redone in any order, but causal-dependent actions are redone in original order

Causal-consistent replay

- It allows one to redo any action, provided that its causes (if any) are redone beforehand
- Concurrent actions can be redone in any order, but causal-dependent actions are redone in original order
- It is the dual of causal-consistent rollback
- It allows one to redo actions which are in the future w.r.t. the current state of the computation
- The choice of the future action to redo depends on the (mis)behavior we want to replay
- How do we know the relevant future actions?

Logging

- Future actions are taken from real executions
- We built a tracer that instruments an Erlang program and produces a log for each process
- We log only concurrency-related actions
- The log has the form

  {73,spawn,74}
  {73,send,5}
  {73,receive,7}
  …

  pid

  unique message identifier

Replay

- We extended CauDEr to take a log and allow the user to explore the logged execution
  - undo selected past actions (and their consequences)
  - redo selected future actions (and their causes)
- We always replay a computation causal equivalent to the original one
  - That is, equal up to swap of concurrent actions
- This is enough to replay the (mis)behaviors of the original computation
  - (For misbehaviors locally visible)

Extending CauDEr with replay

- Available at https://github.com/mistupv/cauder/tree/replay
- This is a branch of CauDEr repository
- Logger at https://github.com/Baha/tracer
- Description and underlying theory currently submitted

Demo time!

The example

- A simple TCP communication between 2 clients and 1 server
- The first client connects on closed port 57 and gets RST
- The second client connects on open port 50
  - Gets a SYN-ACK answer
  - Answers with ACK followed by data
- The server returns the first data received

Future directions

- Support Erlang instead of Core Erlang
  - Not technically difficult, but requires manpower
- Improve efficiency
  - In particular, the time overhead due to running instrumented code is currently too high
  - Particularly critical since instrumented code runs in production environment
- Support a larger subset of the language
  - Distribution, constructs for fault tolerance, ...

Concerning manpower (sorry for advertising)

- Project DCore on reversible debugging for actors (probably Java+Akka, but Erlang may also be a possibility) starting in October
- French ANR project, Bologna is part of it thanks to an agreement between Bologna and French INRIA
  - Working in Bologna with French salary
- Available positions:
  - post-doc (maximum 2 years)
  - PhD shared with INRIA Grenoble (Spades team)
- If interested please let me know

Finally

Thanks!

Questions?