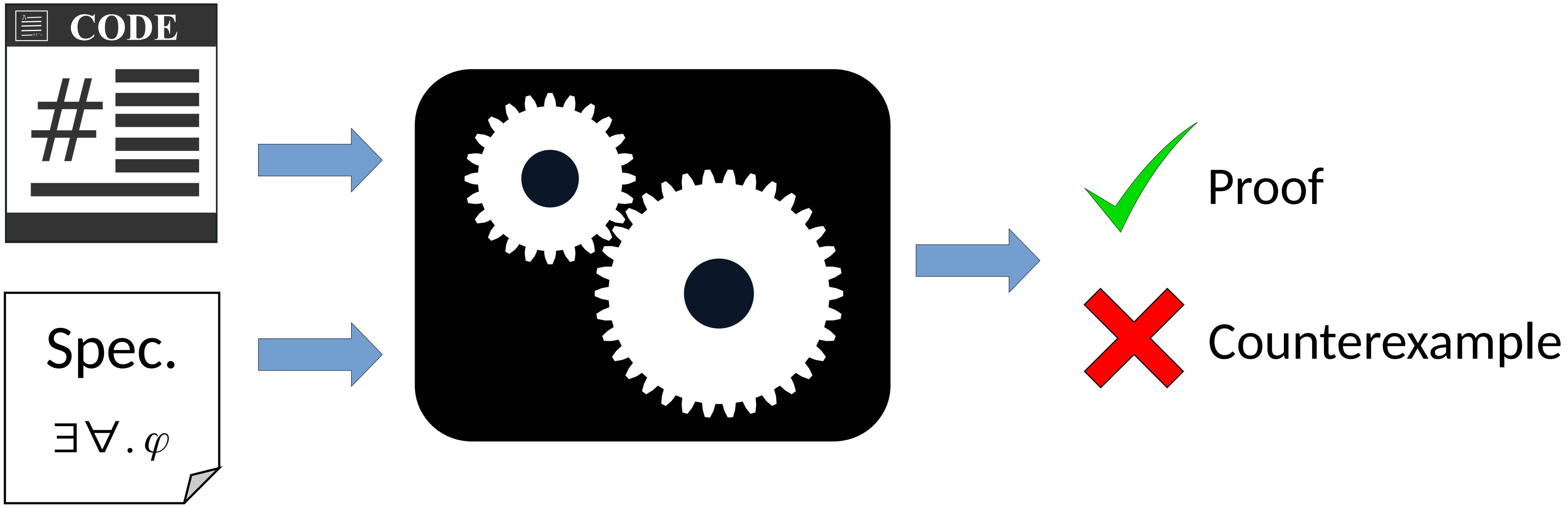
Verification of Message Passing Programs

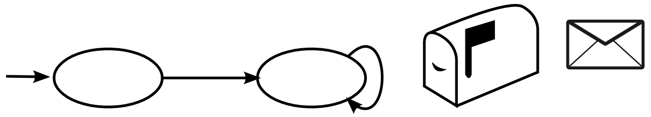
Damien Zufferey
MPI-SWS
27.05.2019

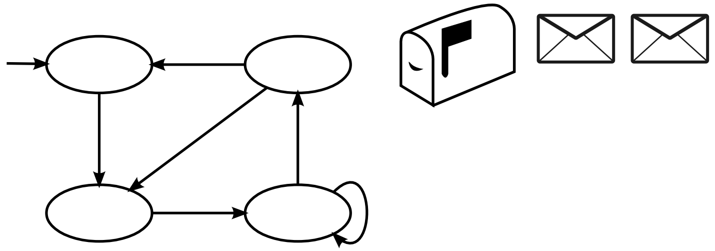
Automated Software Verification



P Language

Communicating State Machine



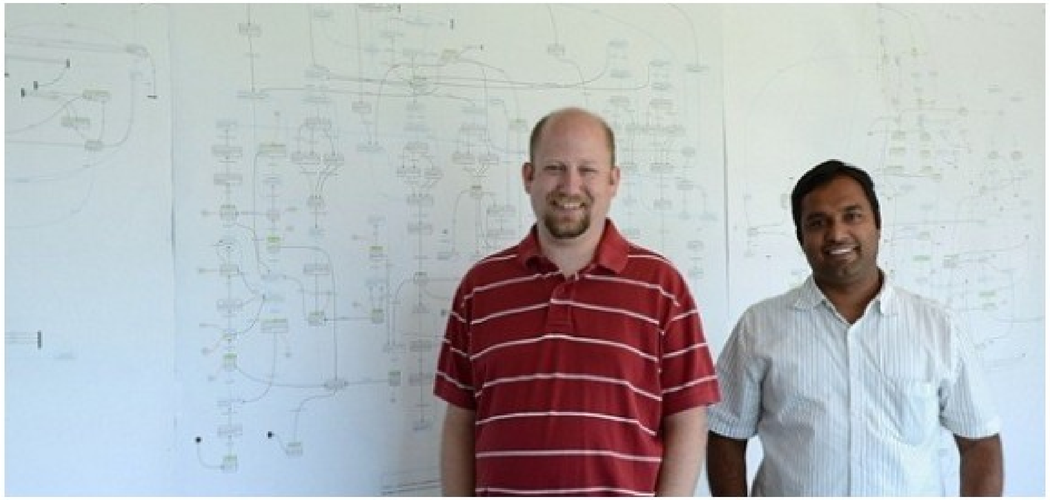


and an explicit state model checker

[PLDI 13] with Ankush Desai, Vivek Gupta, Ethan Jackson, Shaz Qadeer, and Sriram Rajamani.

Applied to Real Applications

With virtual device development underway, we started designing and prototyping. USB software is complex because it has to manage hubs and devices while still dealing with any errors. To create something with longevity we needed to visualize and document the flow. We designed three massive flow charts and a code generator to automatically convert a Visio diagram into software. Together with Microsoft Research, we refined a tool called Zing, which could validate every aspect of this software model.



Flow chart with its architects, Randy Aull and Vivek Gupta

Windows 8 USB driver
Helped find and fix >300 bugs during development...

What Kinds of Bugs?*

Many bugs manifest as an unexpected message is received and it is not handled properly.

So it is about keeping track of the state in a complex protocol across separate communicating programs.**

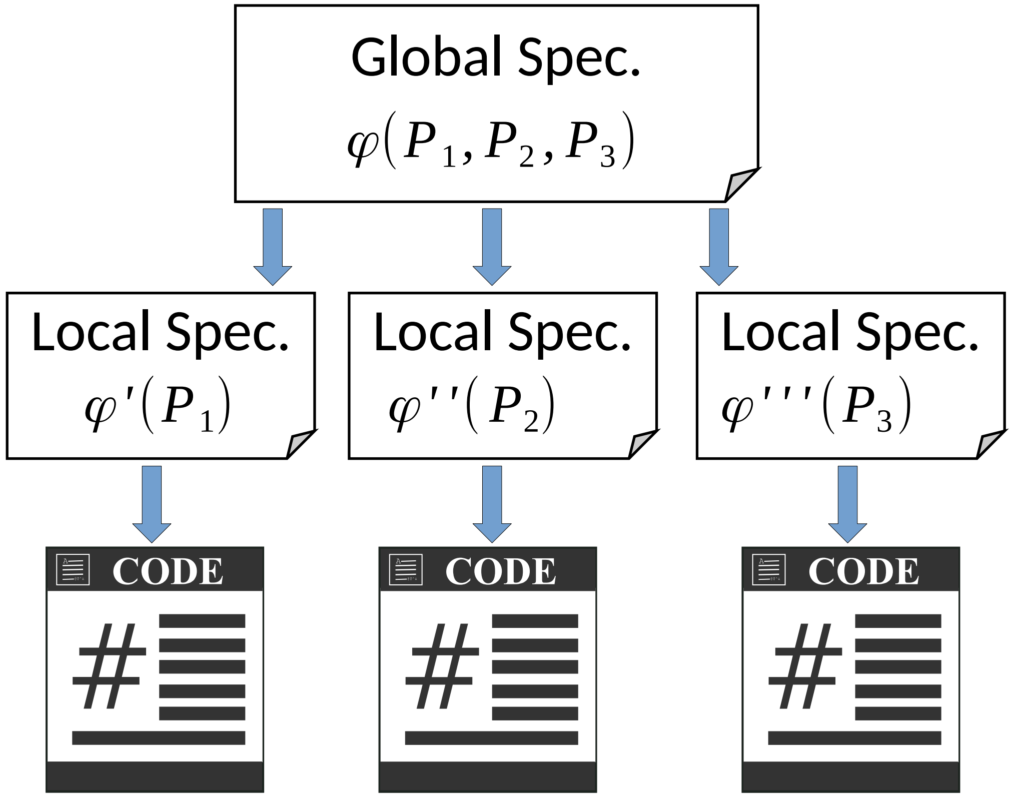
* I don’t have precise numbers, take this with a grain of salt.
** Incorrect/malicious behaviors complicate things...

Can We Do Better?

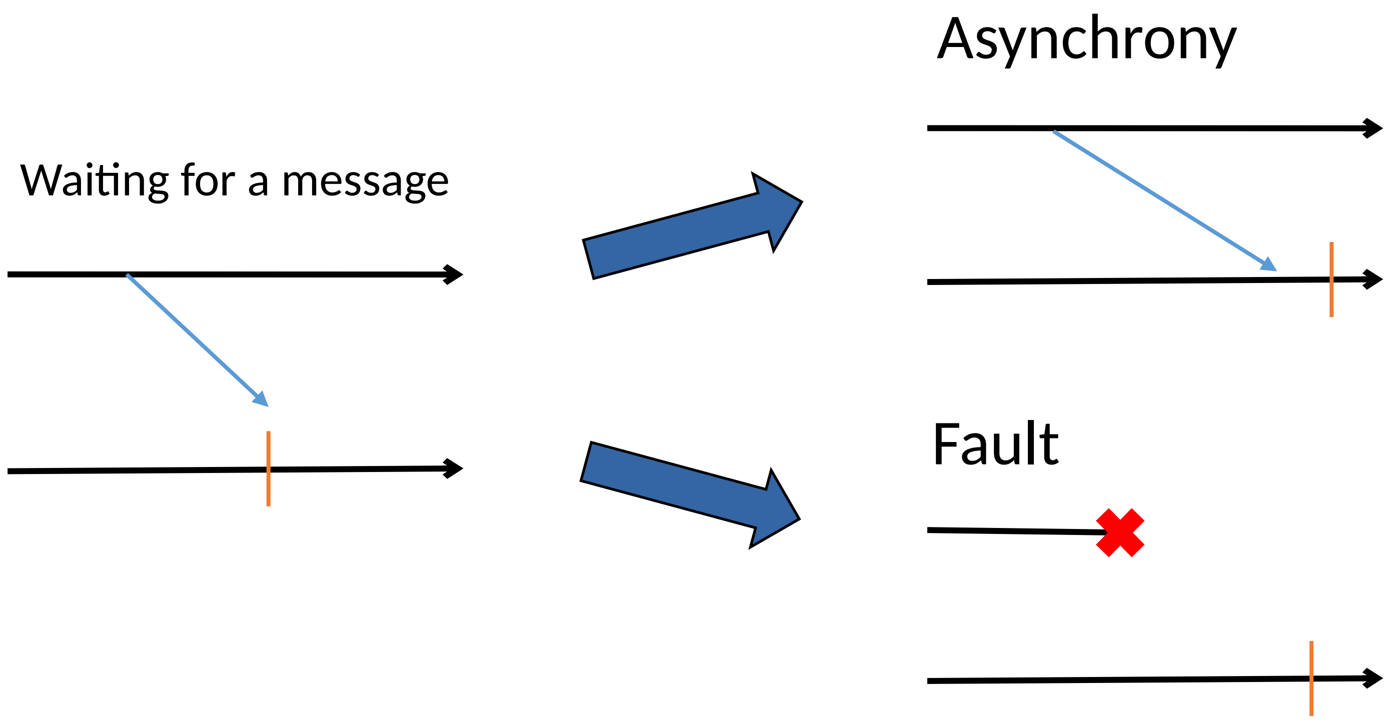
- Model-checking is at best PSpace.
- Can we structure the communication such that it is easier to catch these bugs or find a model were these bugs don’t happen?

Multiparty Session Types

- Workflow
  - Starts with a global spec (communication protocol)
  - Project to local specs
  - Check the code against local spec
- Limitation: strong model
  - Perfect channels
  - No crash
  - Etc.



When Processes Fail?



Wait longer can help but how to avoid blocking?

Asynchronous models are not good at dealing with faults.

Getting it wrong means dangling messages.

Control-flow Inversion

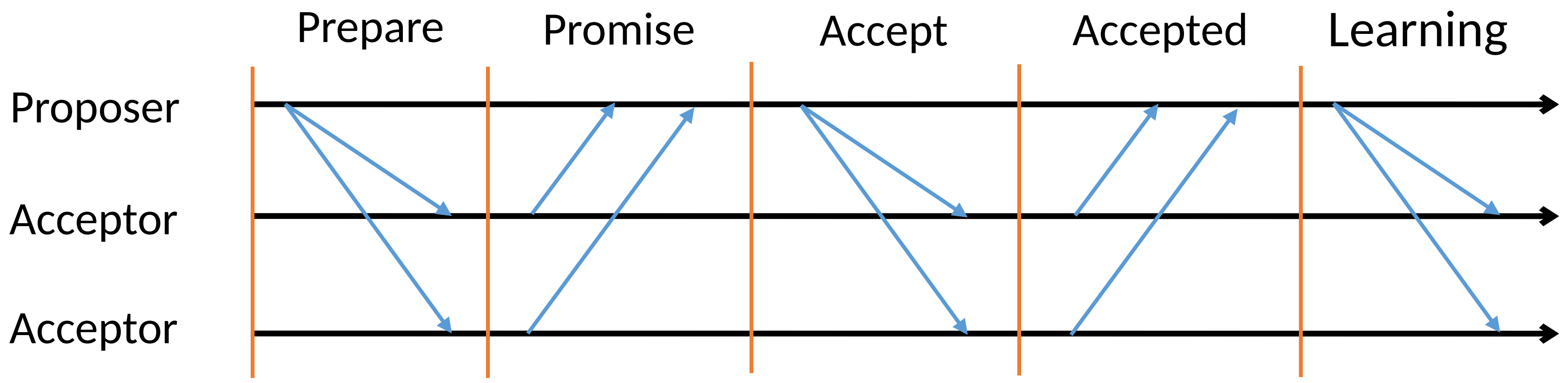
Protocol structure replaced by dispatch:

Protocol:
(1)Msg A
(2)Msg B

```
var state = 1

while (true) {
    on receive {
        case Msg A =>
            if (state == 1) …              normal case
            else if (state == 2) …         message reordered
        case Msg B =>
            if (state == 1) …              message dropped
            else if (state == 2) …         normal case
    }
}
```

Communication-closed Rounds
[Elrad & Francez 82]



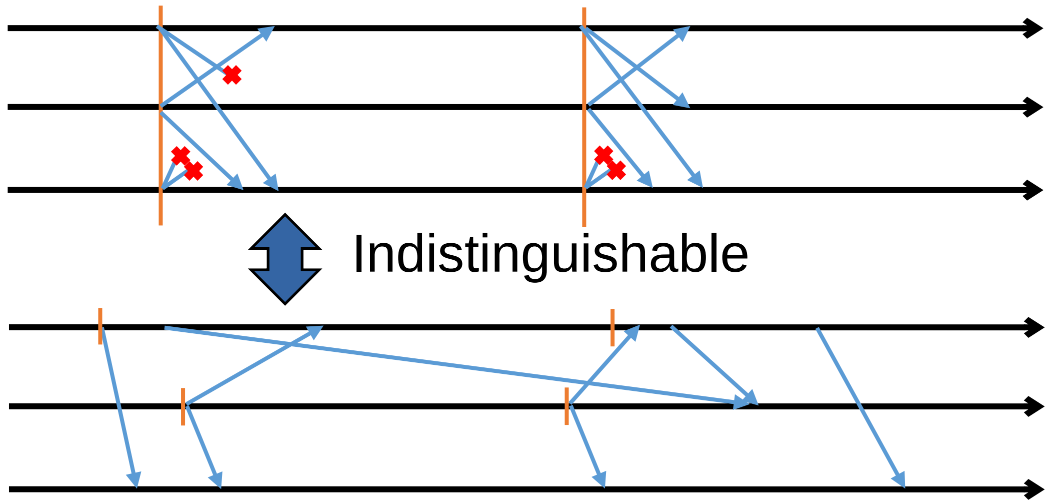
Paxos algorithm organized in communication-closed rounds.

No message cross the boundaries between rounds.

How Does It Model a Real System?

Model faults/asynchrony as an adversarial environment [Gafni 98]
Project all the "faults" on the messages

Lockstep semantics:

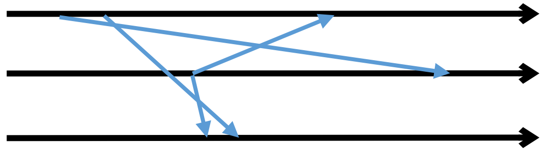


Asynchronous execution:

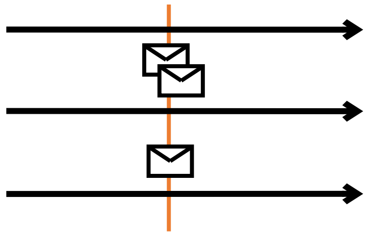
Local views are preserved.

Benefits for the Verification

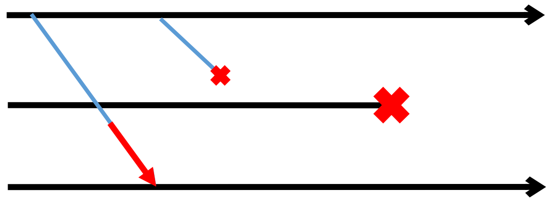
- Asynchrony (interleaving, delays)



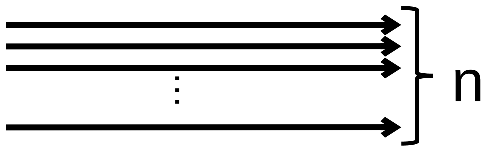
- Channels



- Faults



- Parametric systems



[POPL 16, ongoing] with Cezara Drăgoi, Thomas A. Henzinger, Josef Widder.

CC Rounds Limits

- Does not compose well:
  - Only sequential composition between rounds
  - No branching: successor of a round is unique and statically known
- Performance impact

Desiderata for a New Model

- Flexibility and compositionality of MSP
  - Branching, parallel, delegations, etc.
- Robustness of CC rounds
  - Dealing with faulty/malicious processes
- “Easy” to verify