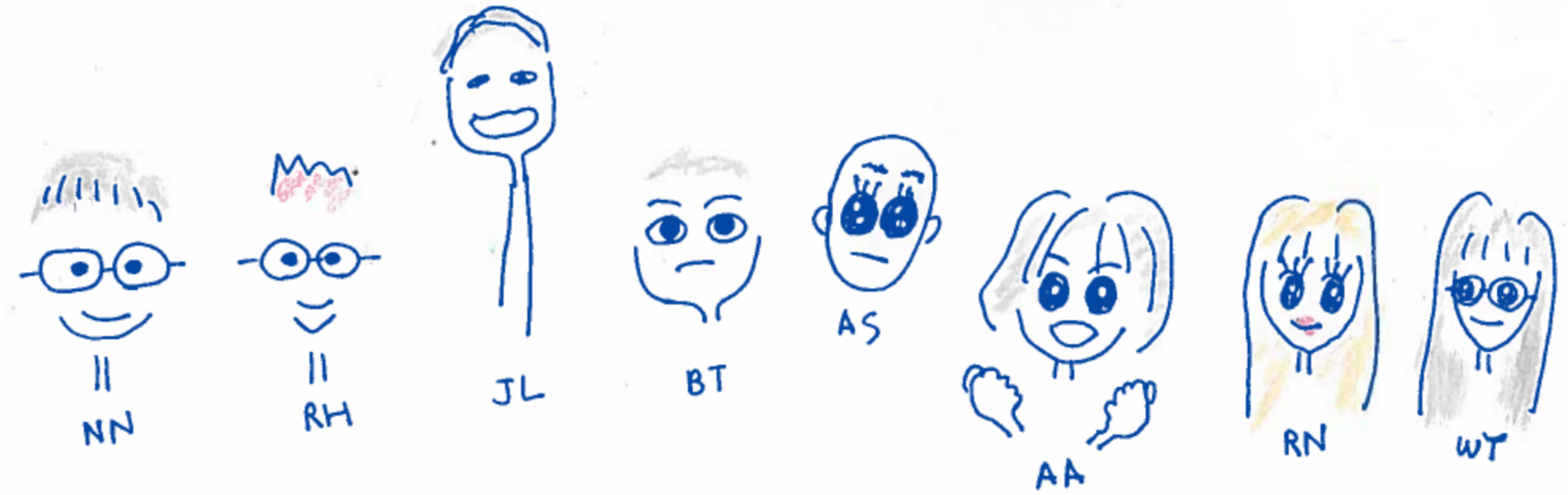
Let it Recover:
Multiparty Protocol-Induced Recovery

Rumyana Neykova, Nobuko Yoshida
Imperial College London

Session Type Mobility Group



www.mrg.doc.ic.ac.uk

Us ∈ Mobility Research Group

http://mrg.doc.ic.ac.uk/

Academic Staff

Nobuko Yoshida

Research Associate

Raymond Hu

Julien Lange

Nicholas Ng

Xinyu Niu

Alceste Scalas

Bernardo Toninho

PhD Student

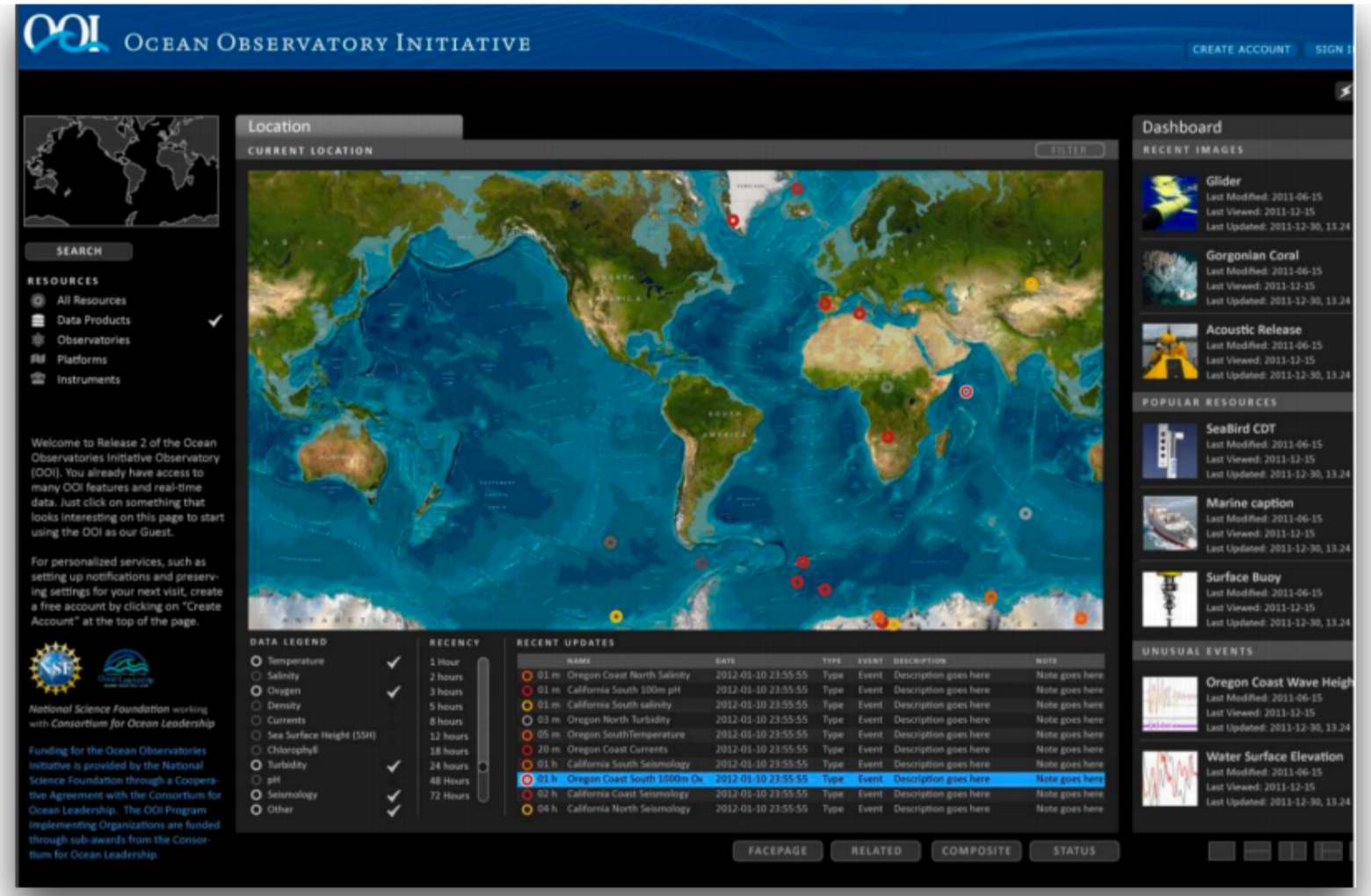
Assel Altayeva

Juliana Franco

Rumyana Neykova

Weizhen Yang

OOI Collaboration



- **TCS'16:** Monitoring Networks through Multiparty Session Types. Laura Bocchi , Tzu-Chun Chen , Romain Demangeon , Kohei Honda , Nobuko Yoshida
- **LMCS'16**: Multiparty Session Actors. Rumyana Neykova, Nobuko Yoshida
- **FMSD'15:** Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. Romain Demangeon , Kohei Honda , Raymond Hu , Rumyana Neykova , Nobuko Yoshida
- **TGC'13:** The Scribble Protocol Language. Nobuko Yoshida , Raymond Hu , Rumyana Neykova , Nicholas Ng

www.scribble.org

Home Getting Started Downloads Documentation Community

Scribble: Describing Multi Party Protocols

Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data. However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.

Describe

Scribble is a language for describing multiparty protocols from a global, or endpoint neutral, perspective.

Verify

Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols described using the language are sound, and do not suffer from deadlocks or livelocks.

Project

Endpoint projection is the term used for identifying the responsibility of a particular role (or endpoint) within a protocol.

Implement

Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b) using session type APIs to clearly describe the behaviour, and (c) statically verify the code against the projection.

Monitor

Use the endpoint projection for roles defined within a Scribble protocol, to monitor the activity of a particular endpoint, to ensure it correctly implements the expected behaviour.

Online tool : http://scribble.doc.ic.ac.uk/

```
1  module examples;
2
3  global protocol HelloWorld(role Me, role World) {
4    hello() from Me to World;
5    choice at World {
6      goodMorning1() from World to Me;
7    } or {
8      goodMorning1() from World to Me;
9    }
10 }
11
```

Load a sample | Check | Protocol: examples.HelloWorld | Role: Me | Project | Generate Graph

Interactions with Industries

Adam Bowen @adamnbowen · Sep 15
I didn't even know that session types existed an hour ago, but thanks to Nobuko Yoshida's great talk at **#pwlconf**, I want to learn more.

Nobuko Yoshida
Imperial College, London

DoC researcher to speak at Golang UK conference

by Vicky Kapogianni
20 July 2016

DoC researcher to speak at industry-focused Golang UK conference on results of concurrency research

Click here to add content

.@nicholascwng rocking on @GolangUKconf about static deadlock detection in #golang #gouk16

Interactions with Industries

Distributed Systems vs. Compositionality

Dr. Roland Kuhn
@rolandkuhn — *CTO of Actyx*

actyx

Current State

- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)

Selected Publications 2016/2017

- **[FoSSaCS'17]** Julien Lange , NY : On the Undecidability of Asynchronous Session Subtyping.
- **[FASE'17]** Raymond Hu , NY : Explicit Connection Actions in Multiparty Session Types.
- **[CC'17]** Rumyana Neykova , NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[POPL'17]** Julien Lange , Nicholas Ng , Bernardo Toninho , NY: Fencing off Go: Liveness and Safety for Channel-based Programming.
- **[FPL'16]** Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk : EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- **[ECOOP'16]** Alceste Scala, NY: Lightweight Session Programming in Scala
- **[CC'16]** Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- **[FASE'16]** Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- **[TACAS'16]** Julien Lange, NY: Characteristic Formulae for Session Types.
- **[ESOP'16]** Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.
- **[POPL'16]** Dominic Orchard, NY: Effects as sessions, sessions as effects .

Selected Publications 2016/2017

- **[FoSSaCS'17] Julien Lange , NY : On the Undecidability of Asynchronous Session Subtyping.**
- **[FASE'17] Raymond Hu , NY : Explicit Connection Actions in Multiparty Session Types.**
- **[CC'17] Rumyana Neykova , NY: Let It Recover: Multiparty Protocol-Induced Recovery.**
- **[POPL'17] Julien Lange , Nicholas Ng , Bernardo Toninho , NY: Fencing off Go: Liveness and Safety for Channel-based Programming.**
- **[FPL'16] Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk: EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.**
- **[ECOOP'16] Alceste Scala, NY: Lightweight Session Programming in Scala**
- **[CC'16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.**
- **[FASE'16] Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.**
- **[TACAS'16] Julien Lange, NY: Characteristic Formulae for Session Types.**
- **[ESOP'16] Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.**
- **[POPL'16] Dominic Orchard, NY: Effects as Sessions, Sessions as Effects**.

Let's Start

Let it Recover:

Multiparty Protocol-Induced Recovery

"Fail fast and recover quickly"

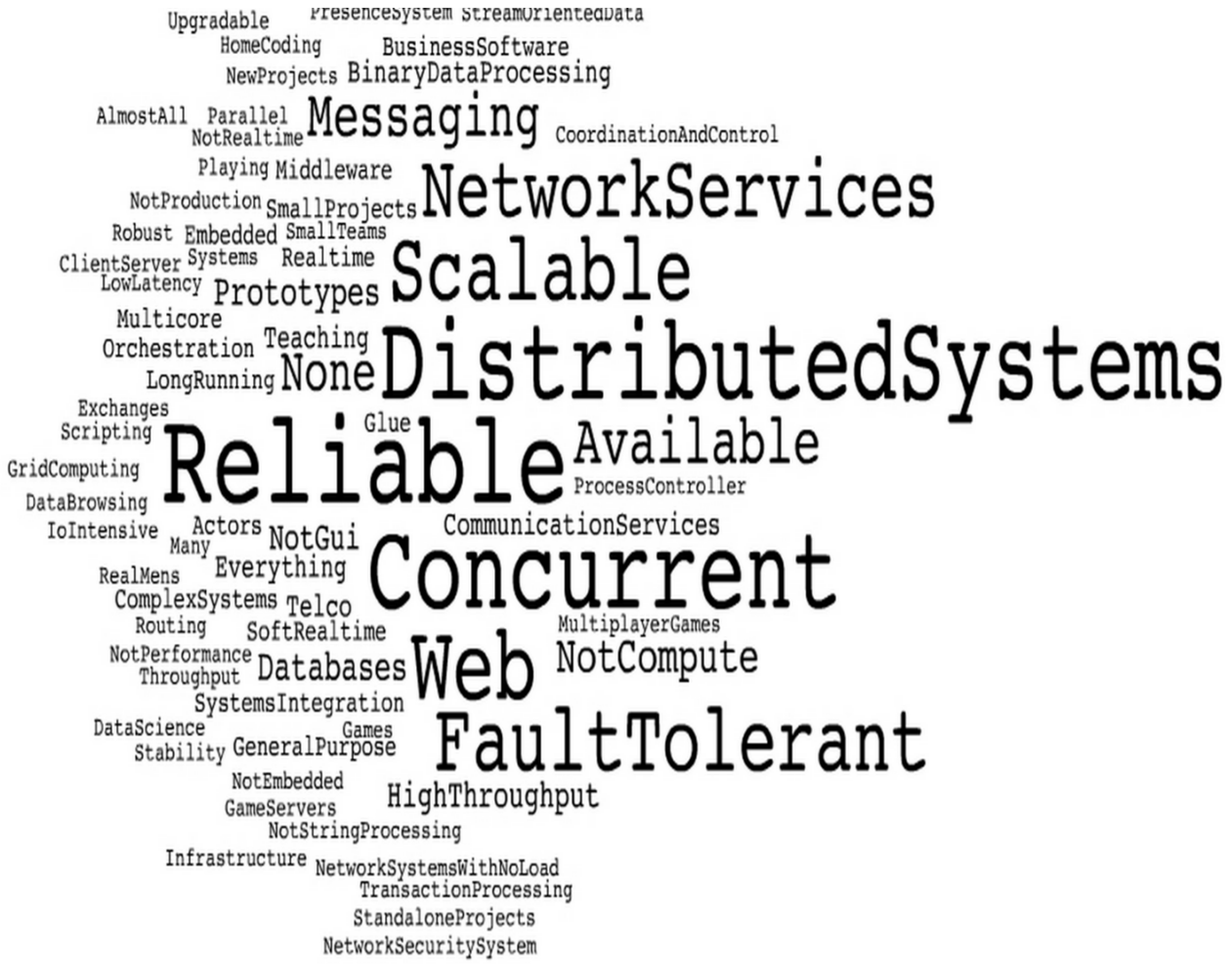
Erlang proverb

"Fail fast and recover quickly and safely"

CC proverb (after this talk)

The Erlang programming language

```
factorial(0) -> 1;
factorial(X) when X > 0 -> X * factorial(X-1).
```



Erlang's coding philosophy

A problem has been detected and windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

_LET_IT_CRASH_

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

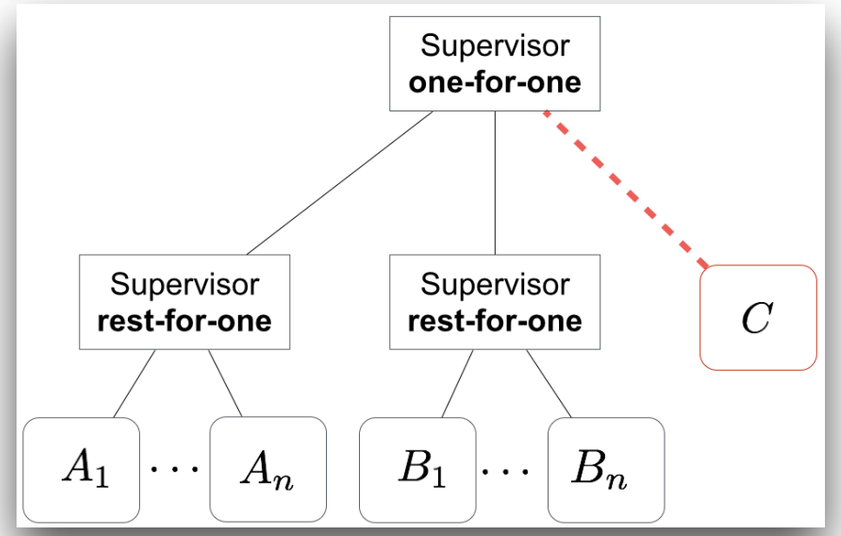
Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

Let it crash: Erlang's fault tolerance model

- Organise your processes in supervision trees



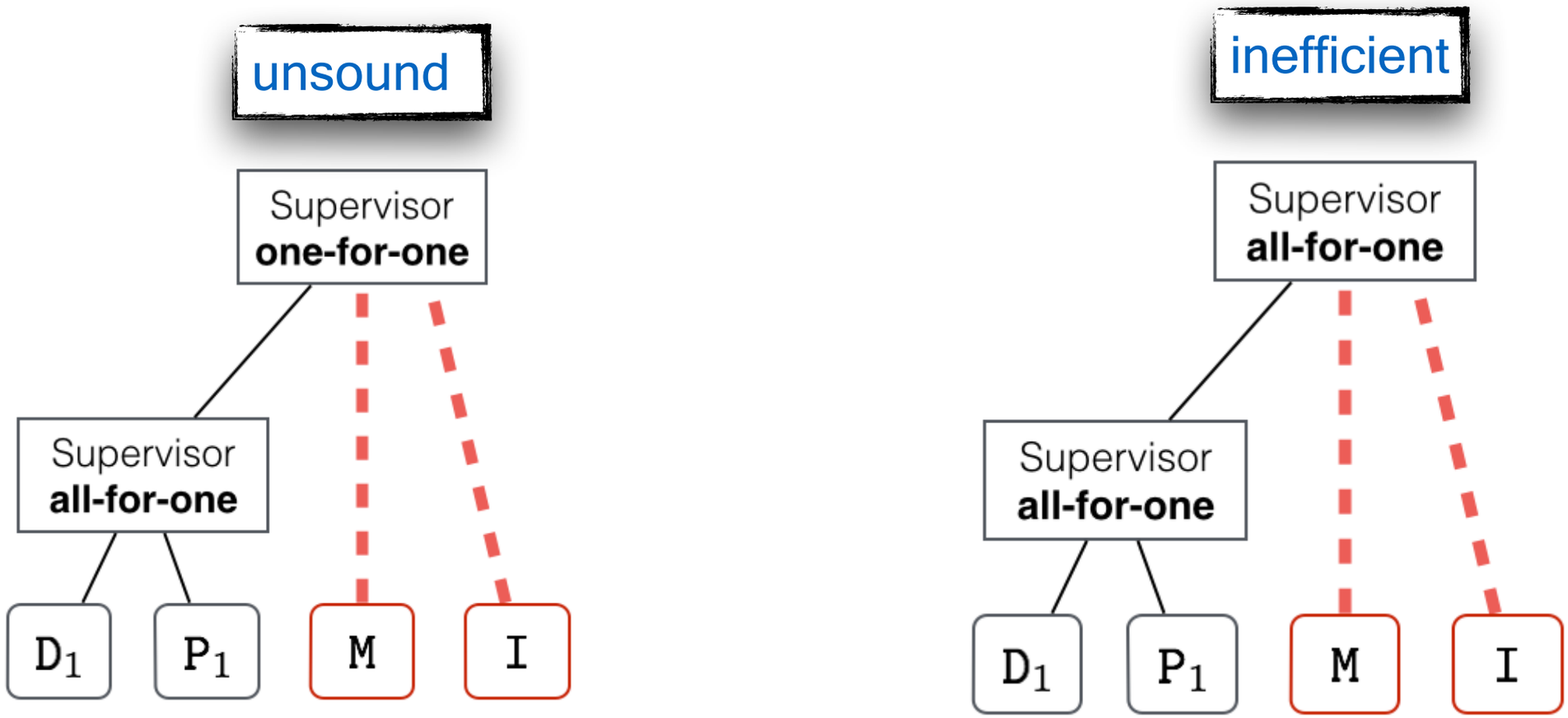
Supervision Strategies

- one-for-one
- all-for-one
- rest-for-one

- Do not program defensively, let the process crash
- In case of error, the process is automatically terminated
- Processes are linked. When a process crashes linked process are notified and (can be) restarted.
- Recently adopted by Scala & Akka, Google

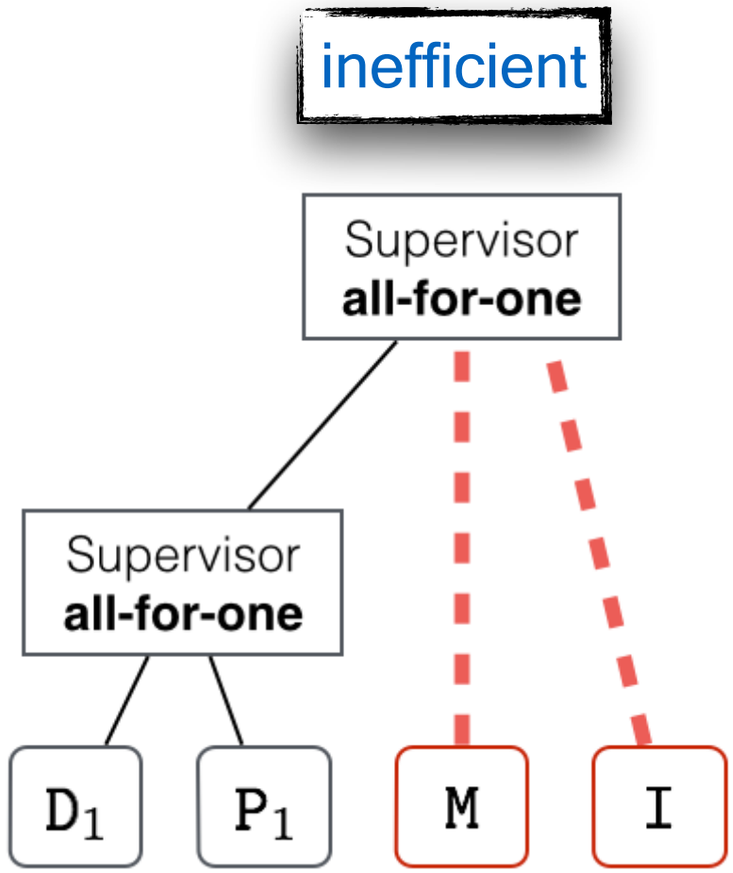
Supervision strategies: Drawbacks

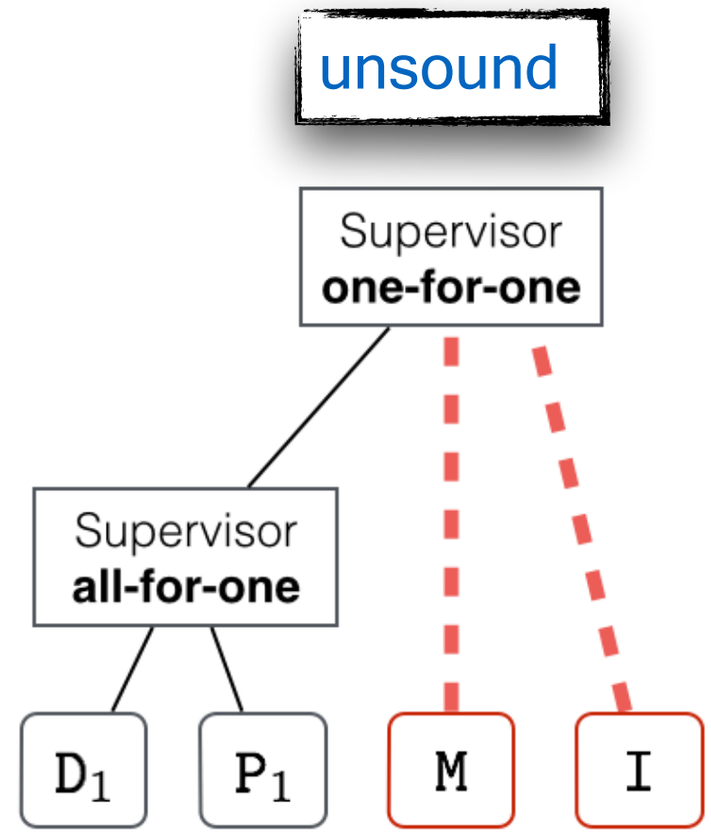
- Supervision strategies are: statically defined, error-prone



- A recovery may cause *deadlocks, orphan messages, reception errors*

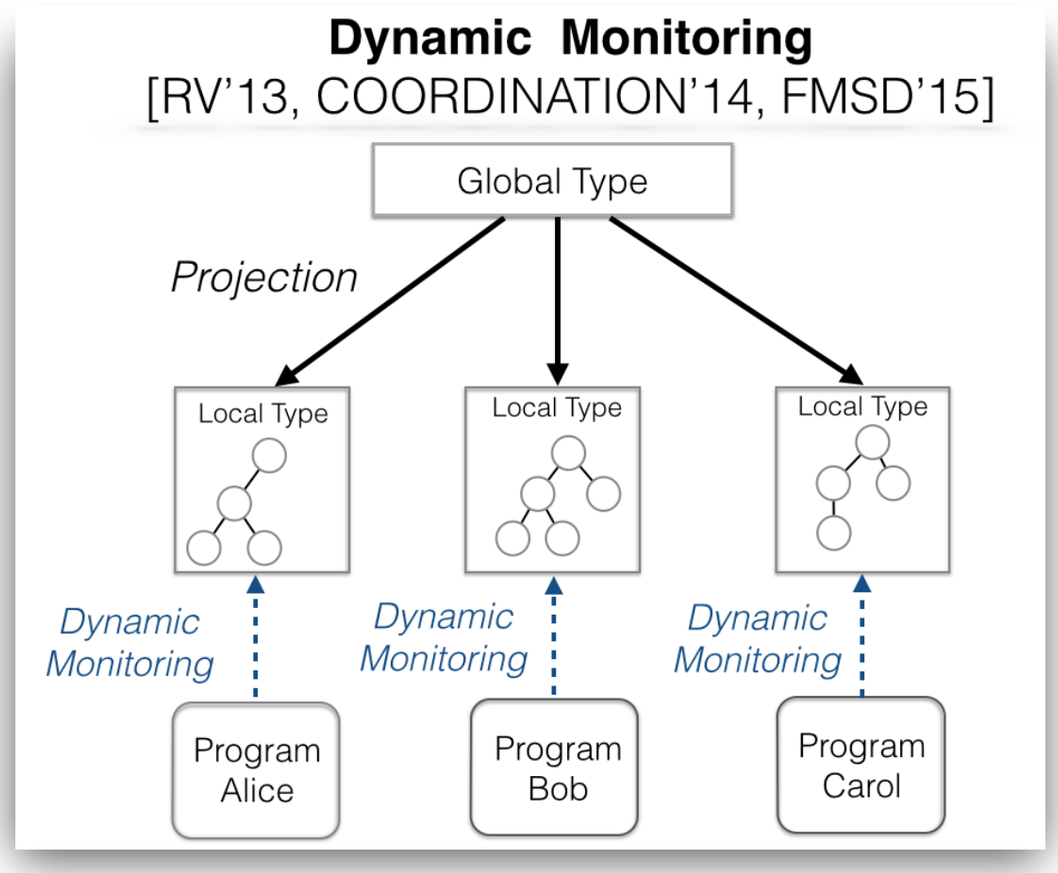
How to generate *sound and efficient* supervision strategies?





By using Session Types!

Session Types Overview



- Global protocol (session type)

$$G = A \to B : \langle U_1 \rangle . B \to C : \langle U_2 \rangle . C \to A : \langle U_3 \rangle$$

- Local protocol (session type)
  - Slice of global protocol relevant to one role
  - Mechanically derived from a global protocol

$$T_A = !\langle B, U_1 \rangle . ?\langle C, U_3 \rangle$$

- Process language
  - Execution model of I/O actions by roles

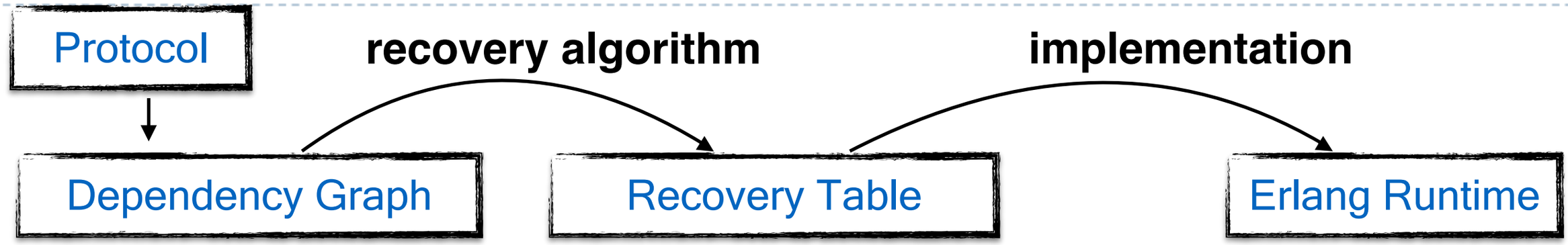
- A system of *well-behaved processes* is free from deadlocks, orphan messages and reception errors

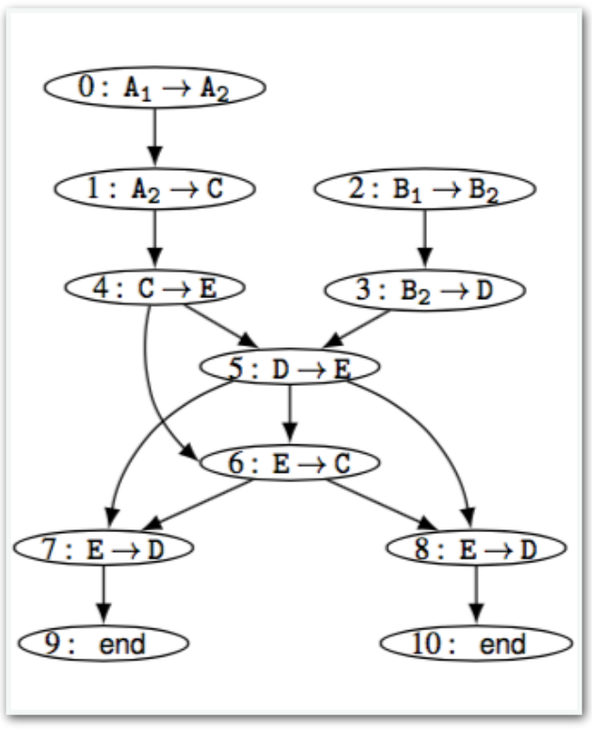
- The framework has been applied to Java, Python, MPI/C, Go…

Part Two

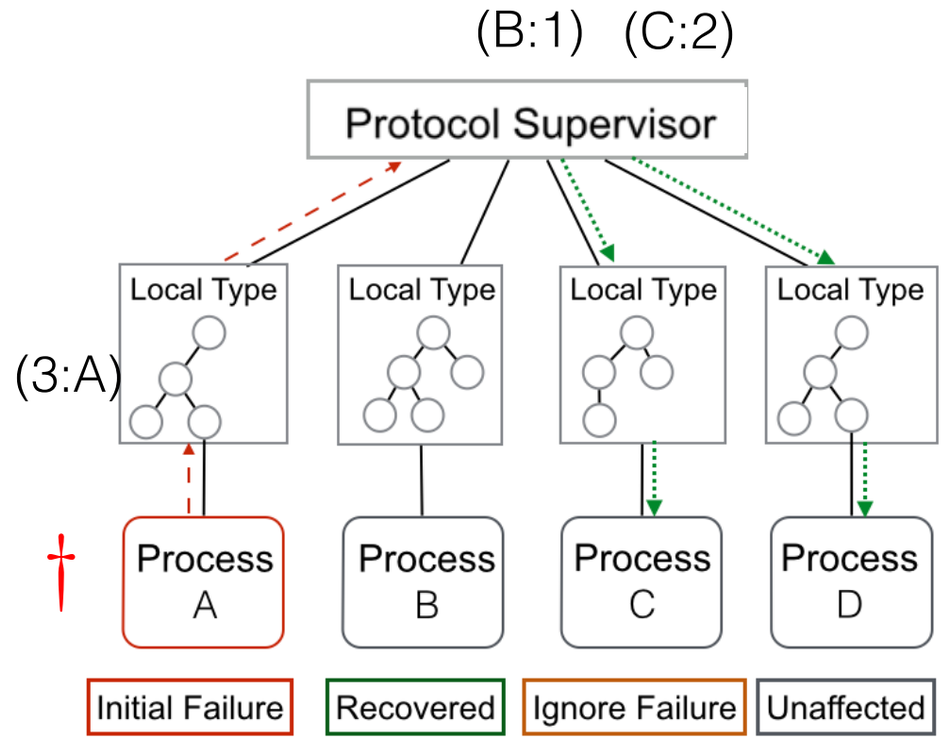
Let it Recover

Recovery (and talk) Workflow





n	r	recovery points
0	A_1	$\{A_1:0\}$
0	A_2	$\{A_2:0, A_1:0\}$
1	A_2	$\{A_2:0, A_1:0\}$
1	C	$\{C:1, A_2:0, A_1:0\}$
3	B_2	$\{B_2:3, B_1:2\}$
3	D	$\{D:3, B_2:2, B_1:2\}$
4	C	$\{C:1, A_1:0, A_2:0\}$
4	E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
5	D	$\{D:3, B_2:2, B_1:2\}$
5	E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
6	C,E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
7	D,E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
8	E,D	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
9	any	$\varnothing$
10	any	$\varnothing$

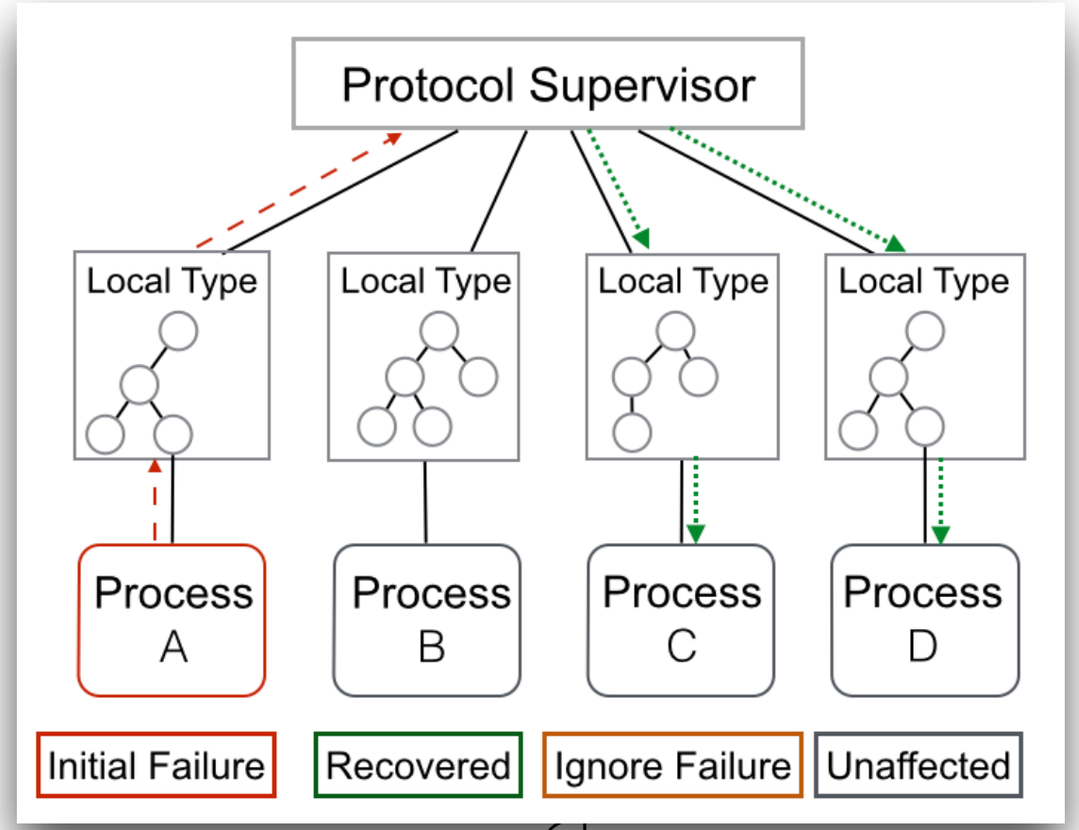


- A **recovered system** is free from deadlocks, orphan messages and reception error.
- Outperforms one of the built-in recovery strategies in Erlang

This talk: Safe Recovery for Session Protocols

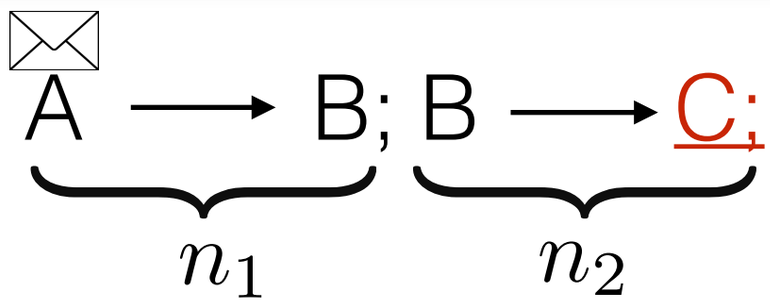
Approach

- ***Recovery algorithm*** to analyse a global protocol as to calculate the dependencies of a failed process.
- Local supervisors ***monitor*** the state of the process in the protocol
- Protocol supervisors use the algorithms ***at runtime*** to decide which process to recover



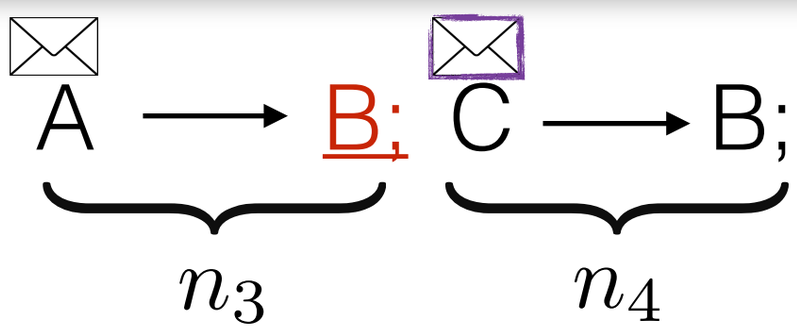
Causalities

$\prec_{io}$ -input-output dependencies (assert the order between a reception of a message and a send action) should recover

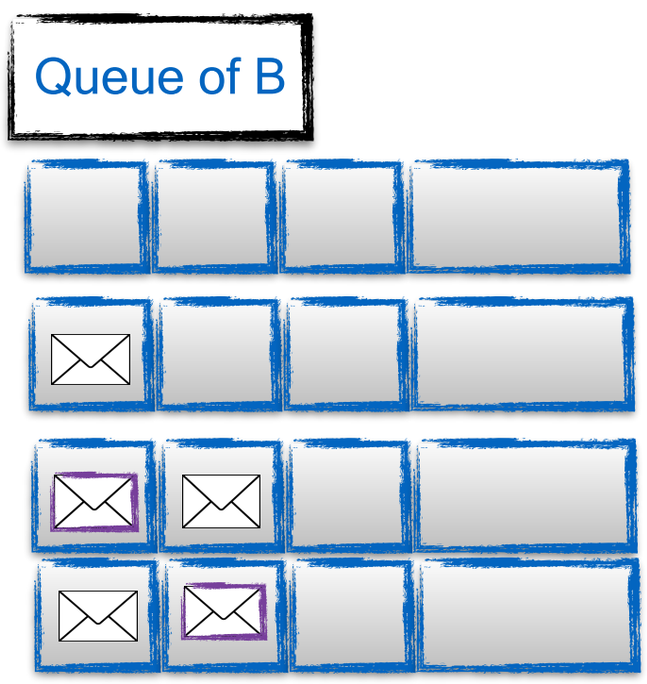


$n_1 \prec_{io} n_2$

$\lhd$ -precedence dependencies (represent the order between two nodes which have a common participant) should recover

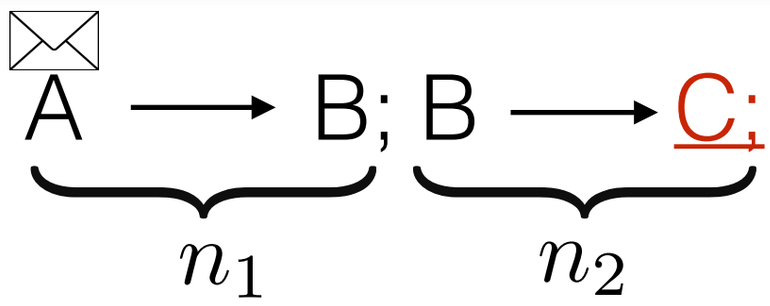


$n_3 \lhd n_4$ $n_3 \not\prec_{io} n_4$



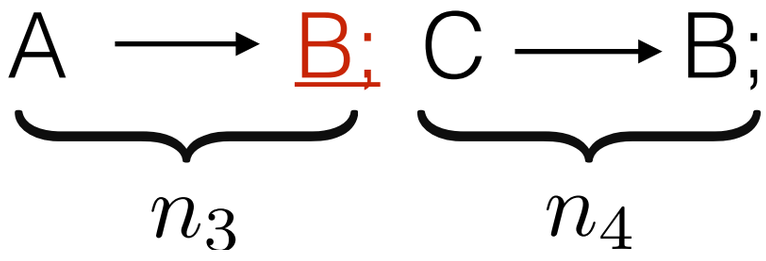
Causalities

$\prec_{io}$ -input-output dependencies (assert the order between a reception of a message and a send action) should recover



$n_1 \prec_{io} n_2$

$\lhd$ -precedence dependencies (represent the order between two nodes which have a common participant) should recover



$n_3 \lhd n_4$ $n_3 \not\prec_{io} n_4$

$\prec_{\dagger}$ -guarded dependencies (represent dependencies of the failed node) should not recover

$n_1 \prec_{\dagger} n_2$

Part Three

Recovery Algorithm

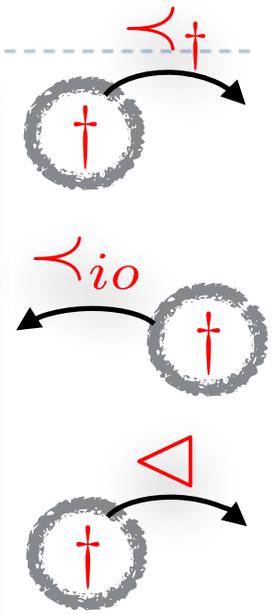
Recovery Algorithm

Algorithm *Calculating affected nodes*
Input: n_i (a failed node), $\mathtt{p}$ (a failed role)
Output: $\mathscr{N}$ (a set of affected nodes)

1. $\mathscr{N} = \mathscr{N}^{\rightarrow} = \{\mathsf{n} \mid \mathsf{n}_i \lhd \mathsf{n} \wedge \mathsf{n} = \mathtt{r} \rightarrow \mathtt{p}\} \cup \{\mathsf{n}_i\}$
2. $\mathscr{S} = \{\mathsf{n} \mid ((\mathsf{n}_i \lhd \mathsf{n}' \wedge \mathsf{n}' = \mathtt{p} \rightarrow \mathtt{r}) \vee \mathsf{n}' = \mathsf{n}_i) \wedge \mathsf{n}' \ll_{\mathtt{IO}} \mathsf{n}\} \setminus \{\mathsf{n}_i\}$
3. **repeat**
4. $\quad \mathscr{N}^{\leftarrow} = \{\mathsf{n} \mid \mathsf{n} \ll_{\mathtt{IO}} \mathsf{n}' \vee (\mathsf{n} \lhd \mathsf{n}' \wedge \mathsf{n} \in \mathscr{S}) \wedge \mathsf{n}' \in \mathscr{N}^{\rightarrow}\}$
5. $\quad \mathscr{N}^{\rightarrow} = \{\mathsf{n} \mid \mathsf{n}' \lhd \mathsf{n} \wedge \mathsf{n}' \in \mathscr{N}^{\leftarrow}\} \setminus (\mathscr{N} \cup \mathscr{S})$
6. $\quad \mathscr{N} = \mathscr{N} \cup \mathscr{N}^{\leftarrow} \quad \mathscr{S} = \mathscr{S} \setminus \mathscr{N}^{\leftarrow}$
7. **until** $\mathscr{N}^{\leftarrow} = \mathscr{N}^{\rightarrow} = \varnothing$
8. **return** $\mathscr{N}$

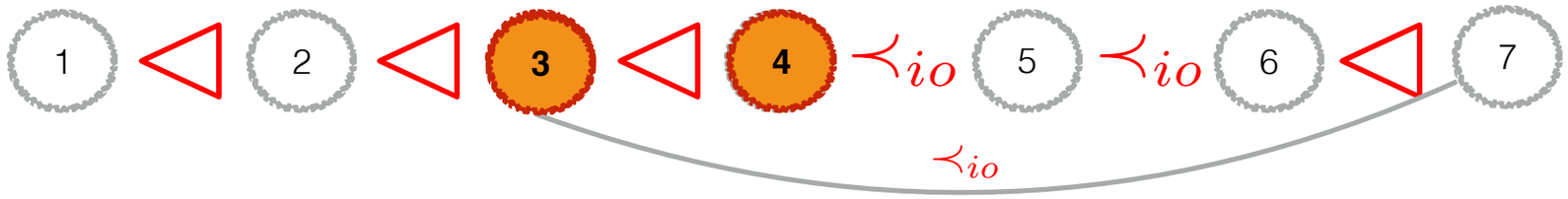
Recovery Algorithm

- Step 1: Initialise the $\prec_\dagger$ dependencies of the failed node
- Step 2: Backward traversal of $\prec_{io}$ dependencies
- Step 3: Forward Traversal of $\lhd$ dependencies
- Step 4: Repeat 2-3 until no new dependencies are added



- Step 1: Initialise the $\prec_\dagger$ dependencies of the failed node
- Step 2: Backward traversal of $\prec_{io}$ dependencies
- Step 3: Forward Traversal of $\lhd$ dependencies
- Step 4: Repeat 2-3 until no new dependencies are added

1:B ⟶ E; 2:C ⟶ E;
3:B ⟶ A; 4:C ⟶ A; 5:A ⟶ D;
6:D ⟶ E; 7:B ⟶ E;



Initialise	$\prec_{io}$	$\lhd$	Final condition	
$\prec_\dagger$:5, 6, 7	3	3, 4	3, 4	not done
	4	4	3, 4	done

Recovery points

- recovery point: take the top node from the set of recovery nodes

1:B ⟶ C; 2:C ⟶ E;
3:B ⟶ A; 4:C ⟶ A;

- Global Recovery Table

Failure	Recovery points
…	…
3, A	A:3, B:3, C:4
3, B	A:3, B:3, C:5
4, C	C:2, E:2
4, A	C:1, B:1, …
…	…

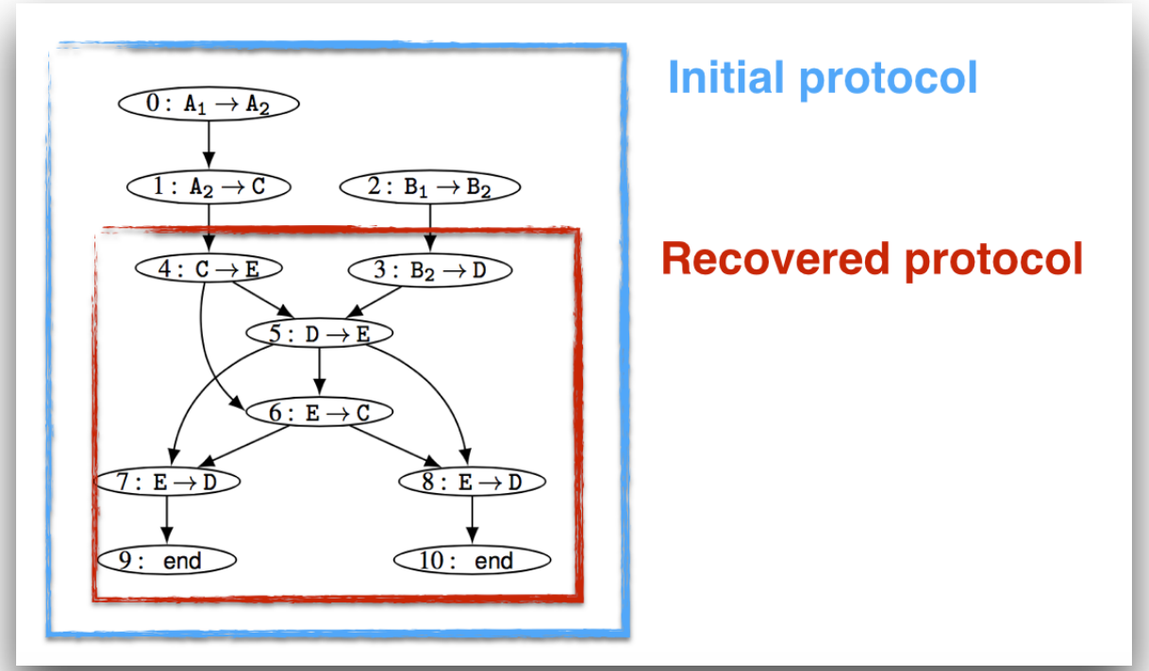
Main Results: Transparency and Safety (informally)

Theorem: Transparency

The recovered protocol is a reduction of the initial protocol.
The configuration of the system after a failure is reachable from the initial configuration.

Theorem: Safety

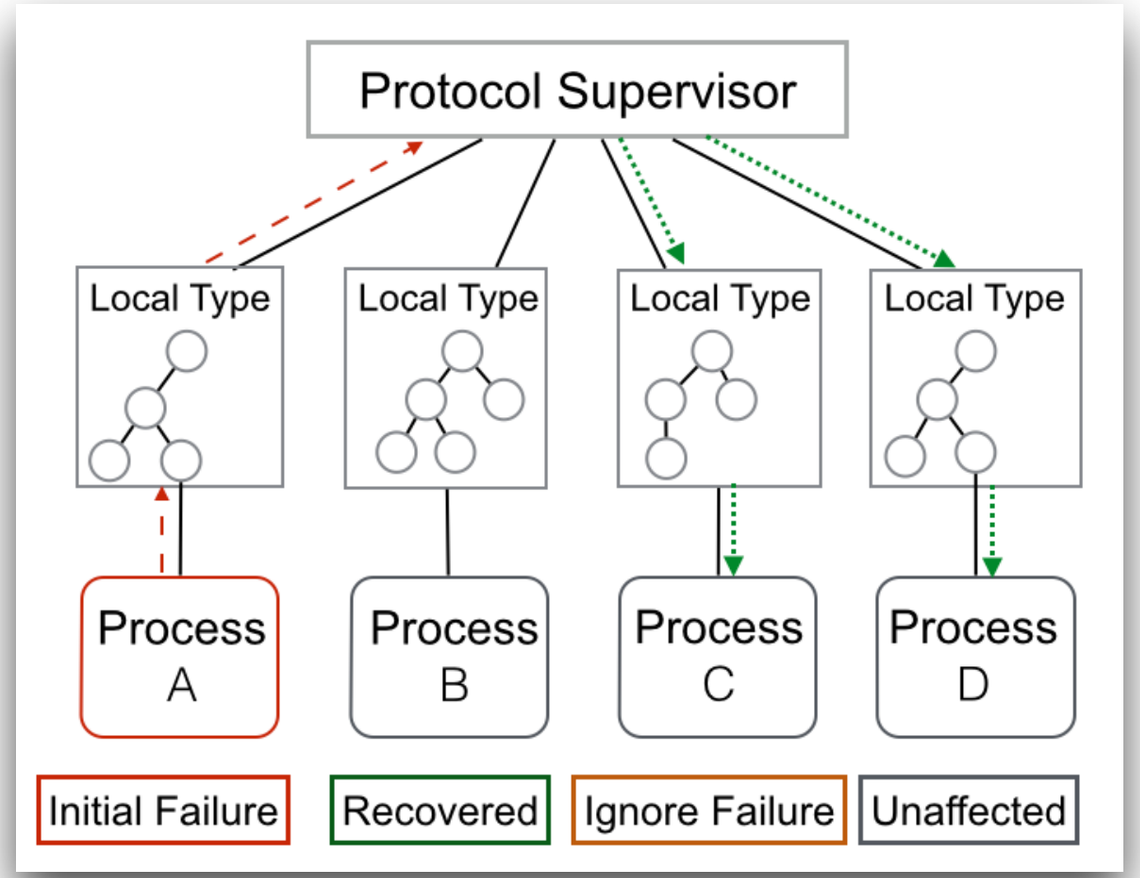
Any reachable configuration which is an initial configuration of well-formed global protocol is free from deadlock, an orphan massage and a reception error.



Part Four

Recovery Implementation

Enabling Protocol Recovery in



protocol supervisor

(recover processes)

local supervisors

(monitor the process behaviour)

gen_server

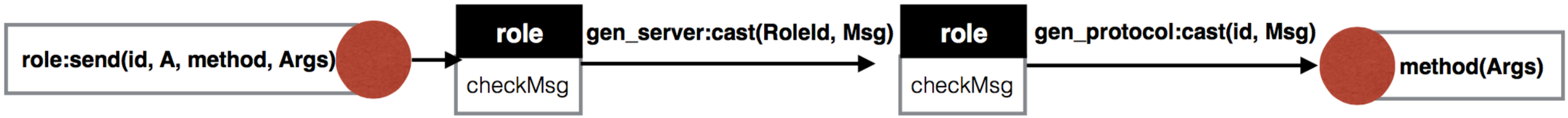
(used to implement processes)

gen_server

stores recovery tables

protocol specification

Enabling Protocol Recovery in Erlang: Example




```
global protocol Trading (
  role A, role B,
  role C, role D){

  quote(int) from A to C;

  quote(int) from B to D;

  quote(int) from C to E;
  quote(int) from D to E;
  choice at E {
    accept() from E to C;
    accept() from E to D;
  or {
    reject() from E to C;
    reject() from E to D;
  }
}
```

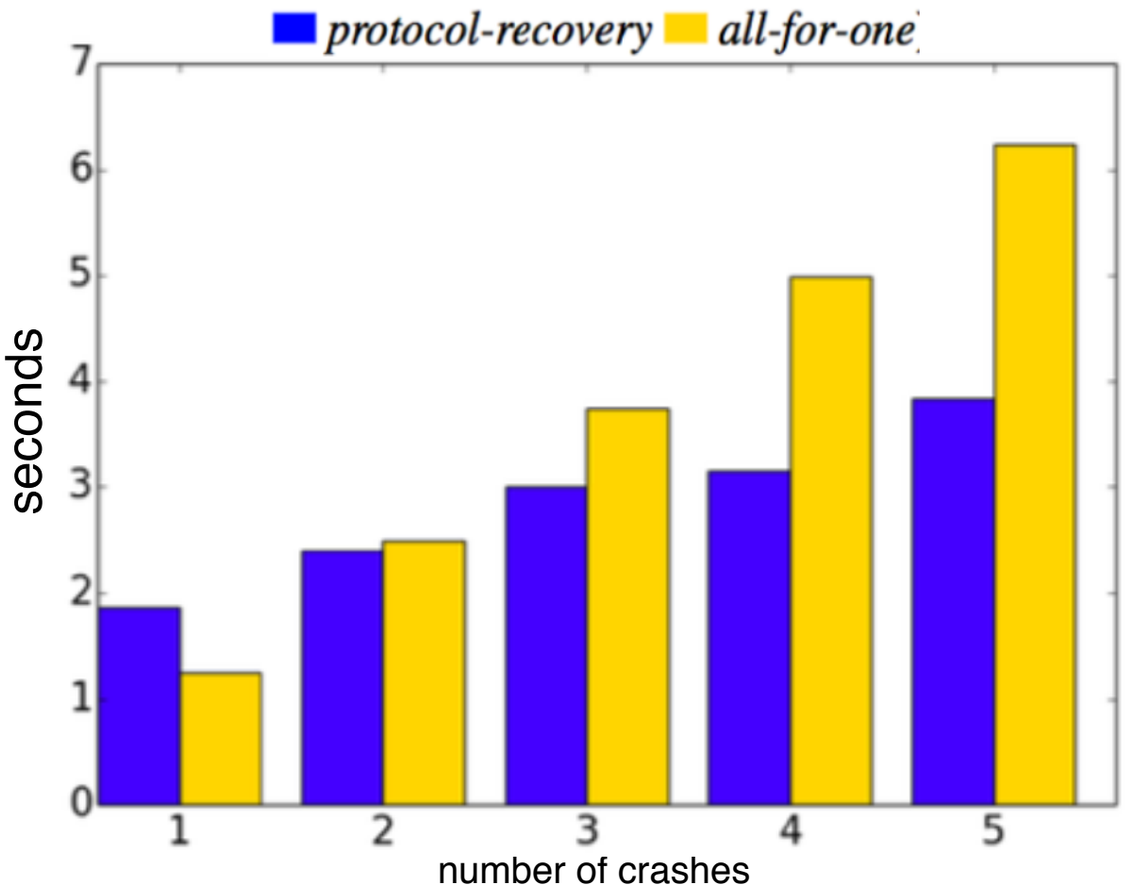
```
% Handlers for C and D
 quote({msg,Val},State) →
  role:send(State#state.role, ?E, quote, Val).
```

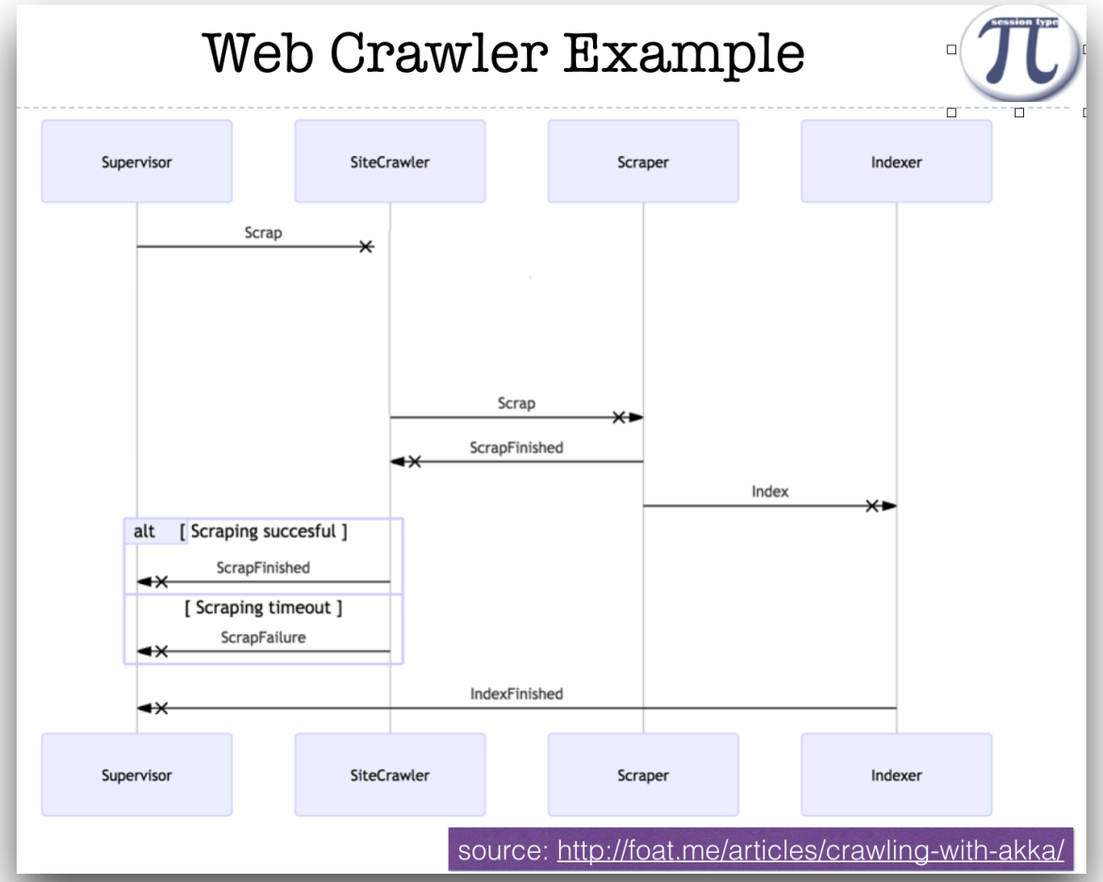
```
% Handlers for E
quote({msg, Val},State) when State.prev==undef →
  {noreply,State#state{prev=Val}};;

quote({msg, Val},State) when State#state.prev>Val →
  role:send(State#state.role, ?C, reject, empty),
  role:send(State#state.role, ?D, accept, empty),
  {noreply,State};

 quote({msg, Val},State) when State#state.prev<Val →
  role:send(State#state.role, ?C, accept, empty),
  role:send(State#state.role, ?D, reject, empty),
  {noreply,State}.
```

Scribble

ERLANG

Evaluation: Web Crawler Example

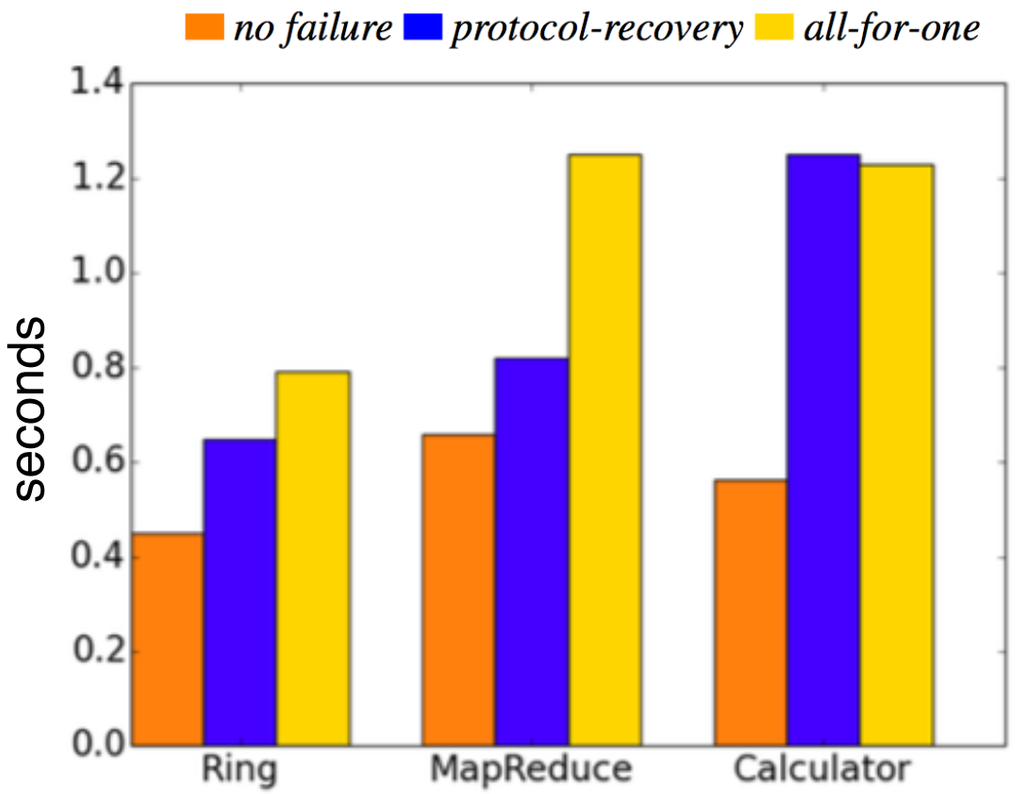




- A process is chosen at random at the start
- Improvement when several failures occur
- By mistake initially we implemented all-for-one that introduced a deadlock

source: http://foat.me/articles/crawling-with-akka/

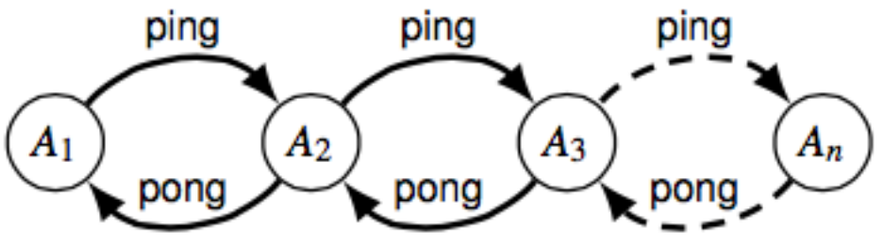
Evaluation: Concurrency Patterns



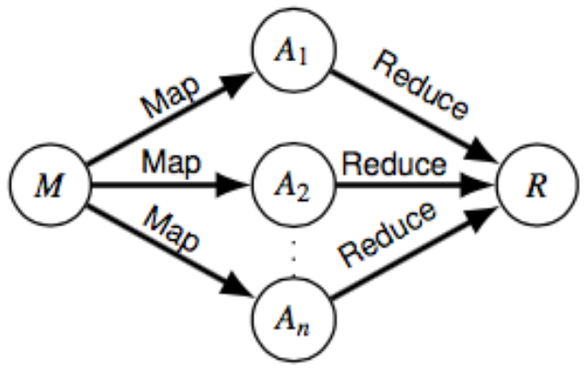
Example	#roles	#states	GRT (sec)	affected roles
MapReduce [21]	n+1	n+2	0.11	W[1] ... W[n]
Ring [21]	n	2*n	0.16	W[1] ... W[n]
Calculator [18]	n+1	4*n	0.75	A[1]

- 52% improvement when
  - intense local computation
  - disconnected interactions
- Up to 7% overhead when all roles are restarted

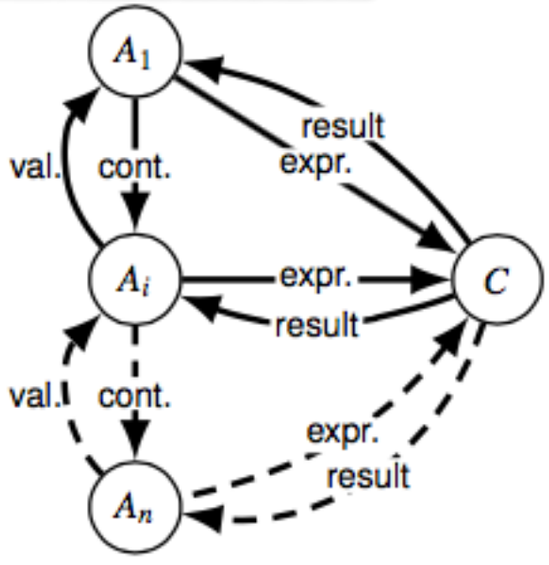
Ring



Map Reduce



Calculator



Future work & Resources

Framework summary

- Ensure processes are safe and conform to a protocol (even in cases of failures)
- Create supervision trees and link processes dynamically based on a protocol structure

Future work

- Support for stateful processes
- Integration with checkpoints
- Replications and recovery actions

Additional Resources

- Scribble webpage: scribble.doc.ic.ac.uk
- Project source: https://gitlab.doc.ic.ac.uk/rn710/codeINspire
- MRG webpage: **http://mrg.doc.ic.ac.uk/**

Q & A