# Let it Recover:
## Multiparty Protocol-Induced Recovery

Rumyana Neykova, Nobuko Yoshida
Imperial College London

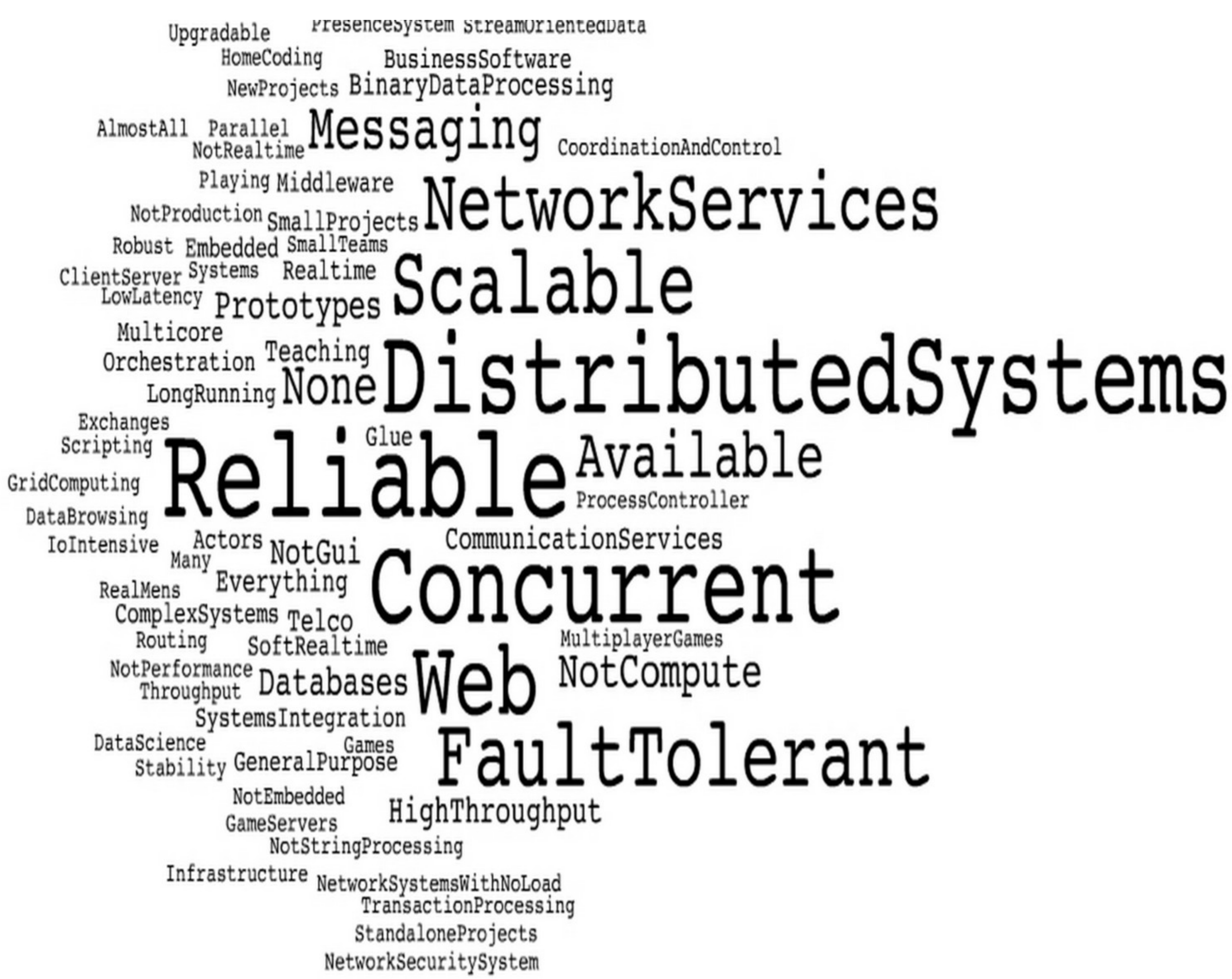
> “*Fail fast and recover quickly*”
>
> Erlang proverb

> “*Fail fast and recover quickly and safely* ”
>
> OPCT proverb (after this talk)

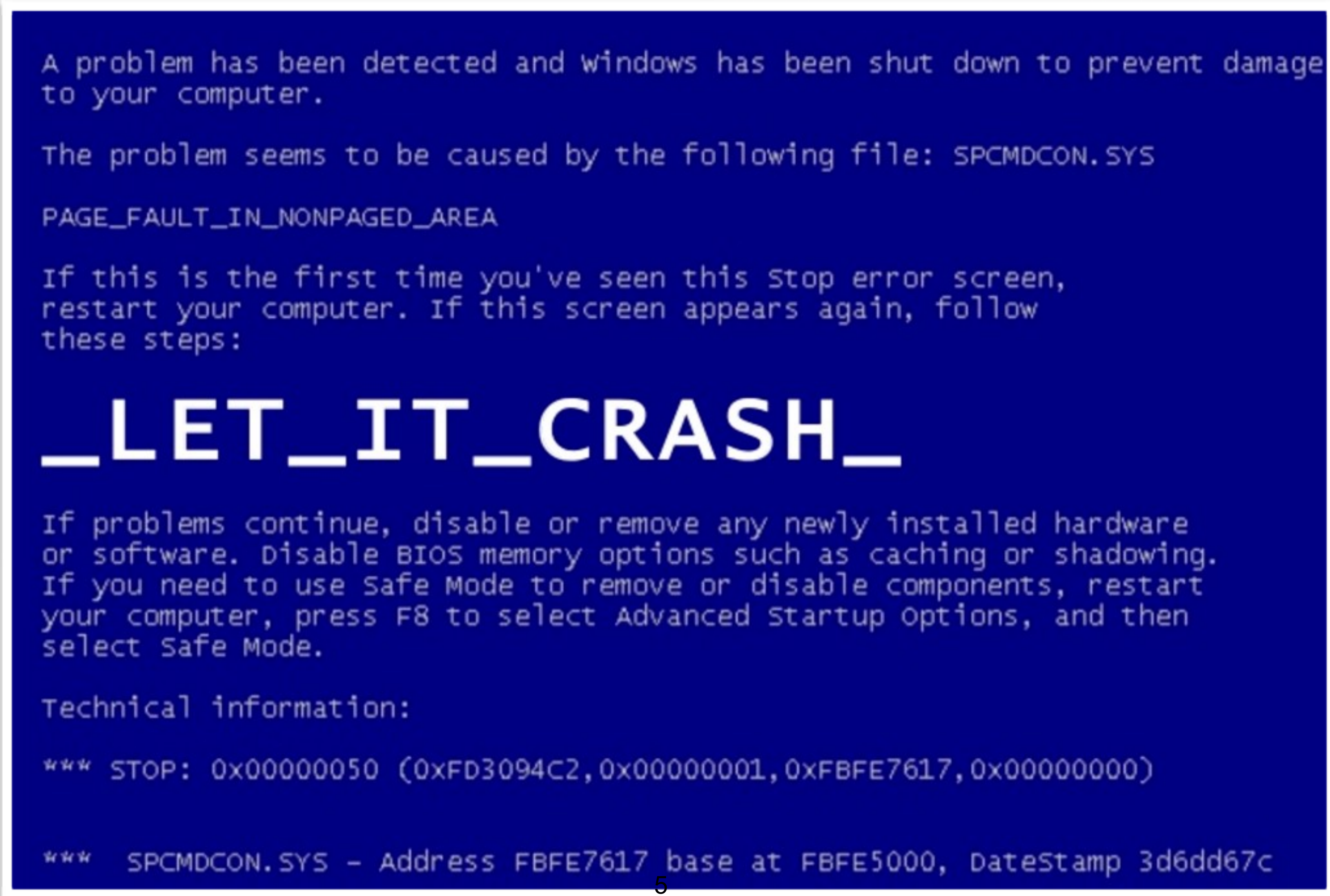
# *Part One*
# Background

# The Erlang programming language

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

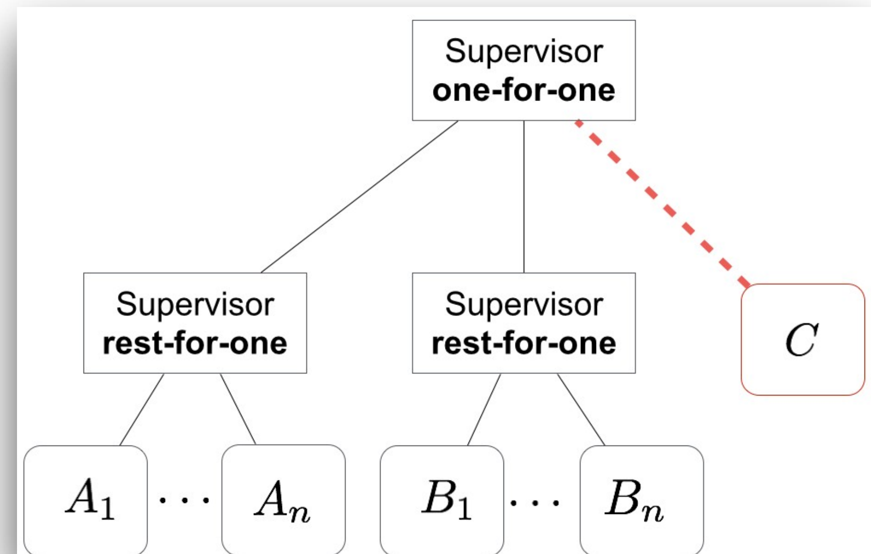


# Erlang's coding philosophy



# Let it crash: Erlang’s fault tolerance model

- Organise your processes in supervision trees



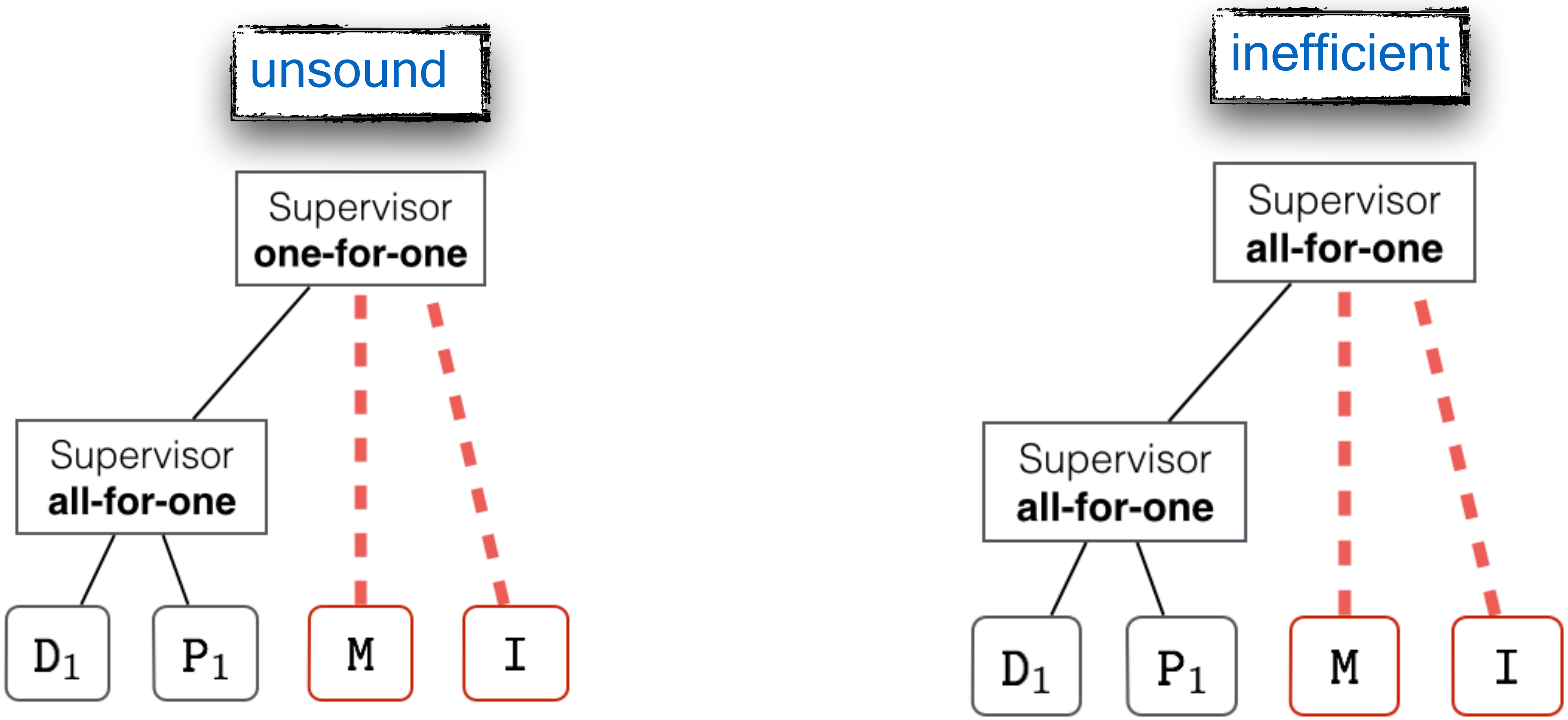

Supervision Strategies

- 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

# Supervision strategies: Drawbacks

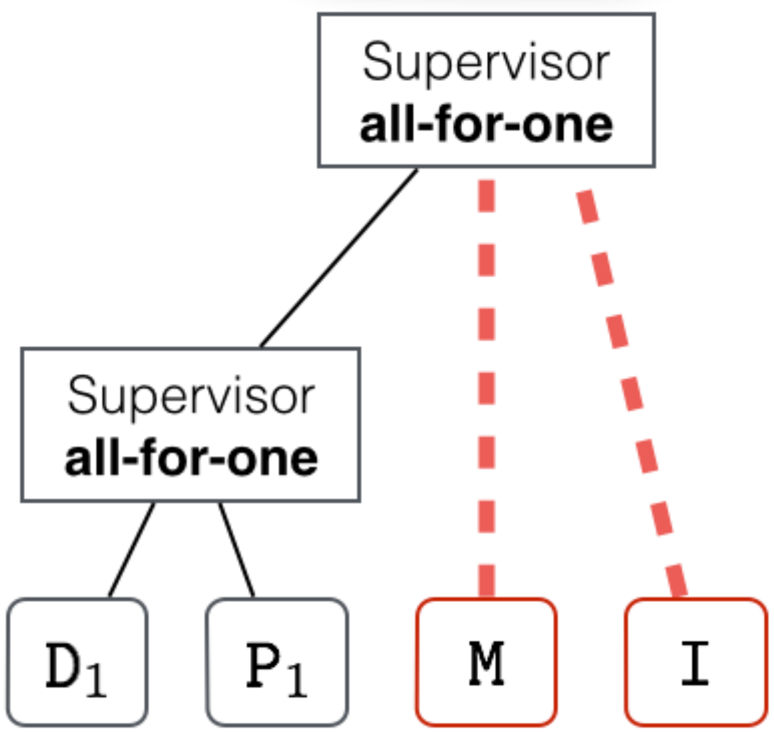
- Supervision strategies are: statically defined, error-prone



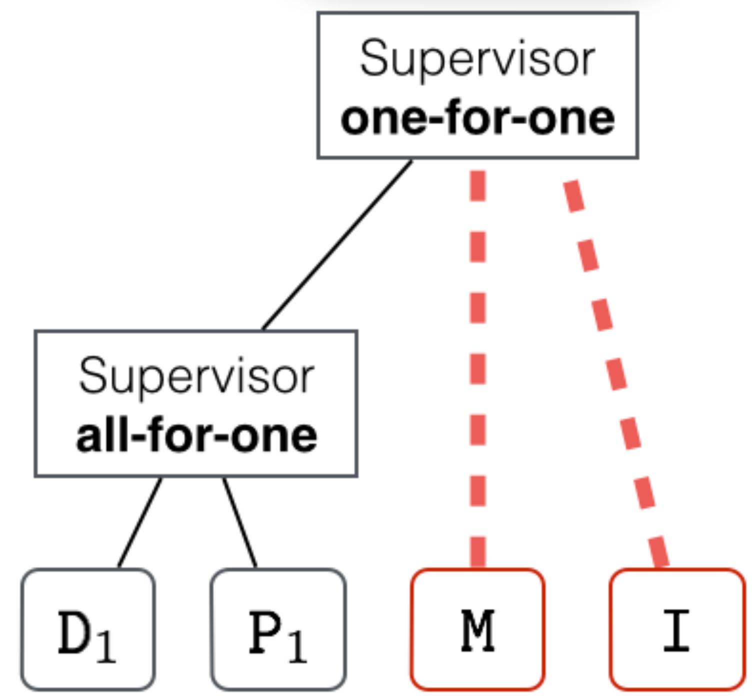

- A recovery may cause *deadlocks, orphan messages, reception errors*

How to generate *sound and efficient* supervision strategies?

inefficient

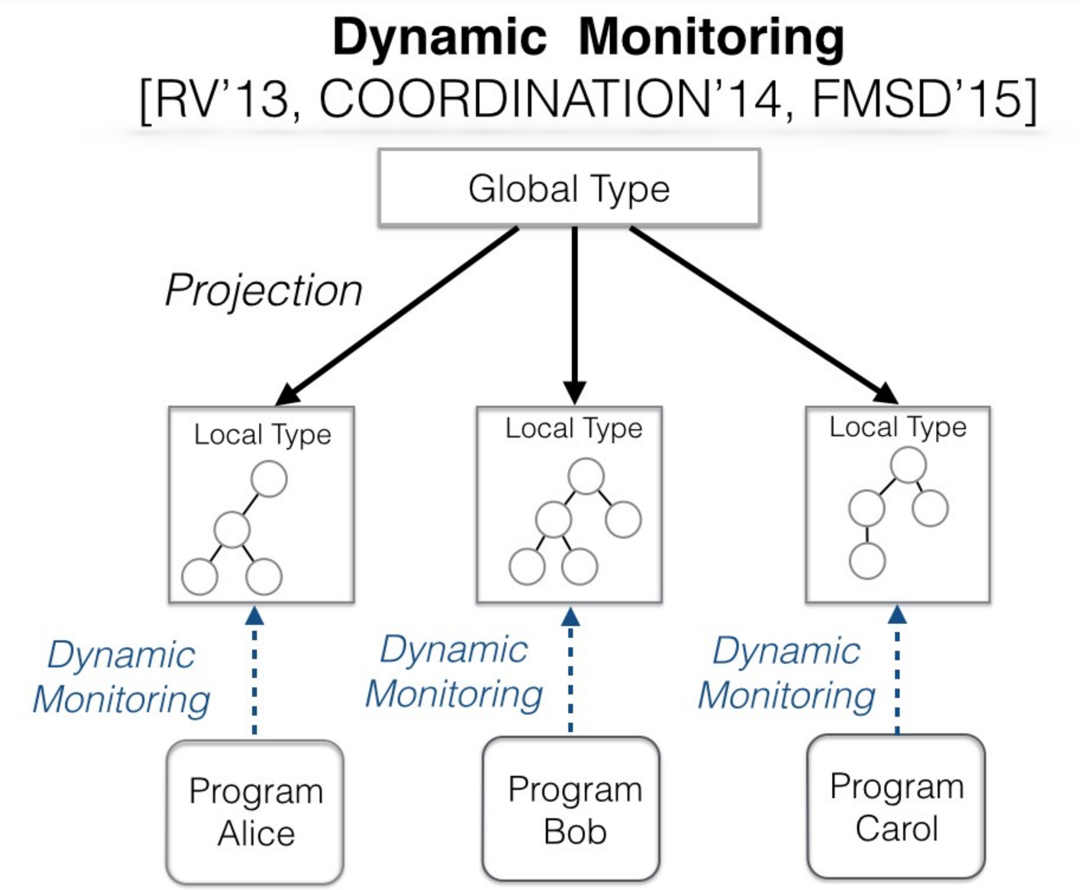



unsound




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

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

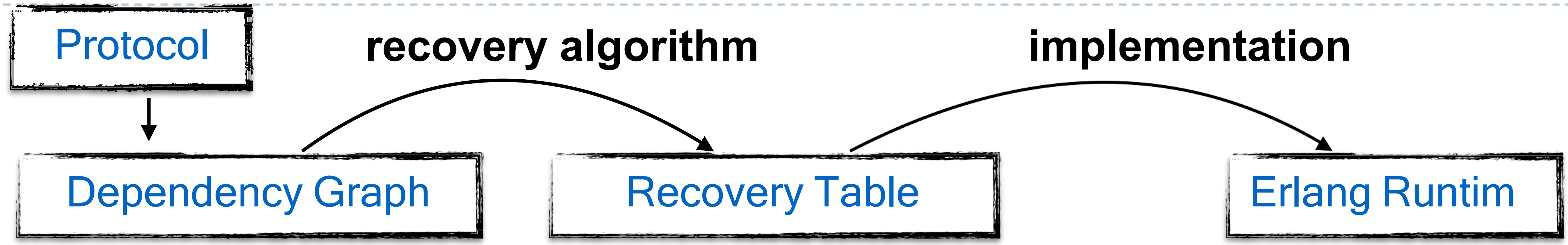
- Process language
  - Execution model of I/O actions by roles

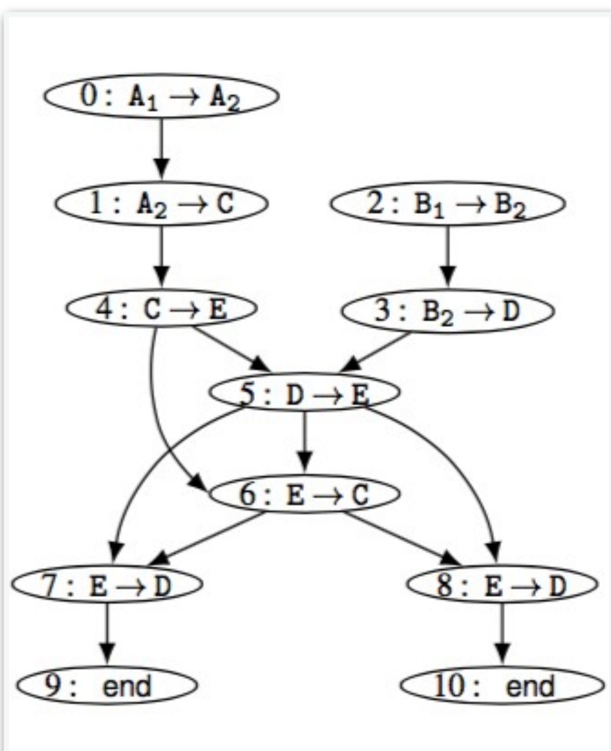
- A system of *well-behaved processes* is free from deadlocks, orphan messages and reception errors

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

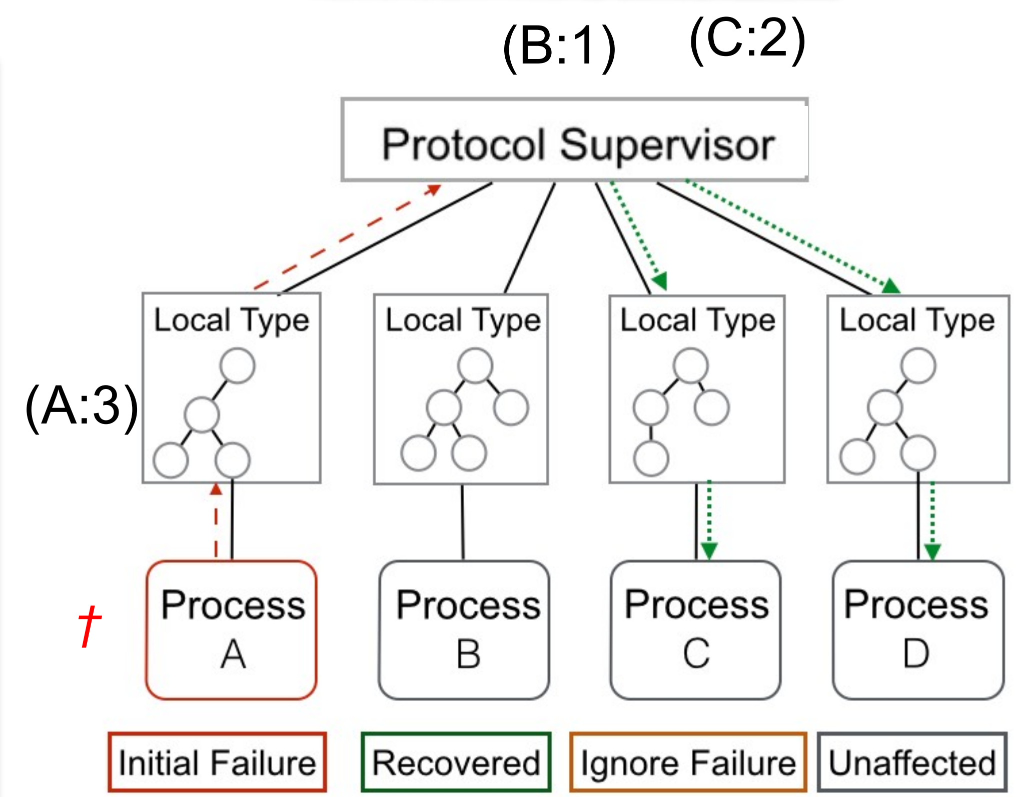
# *Part Two*
# Let It Recover

# Recovery 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 | $\varnothing$ |

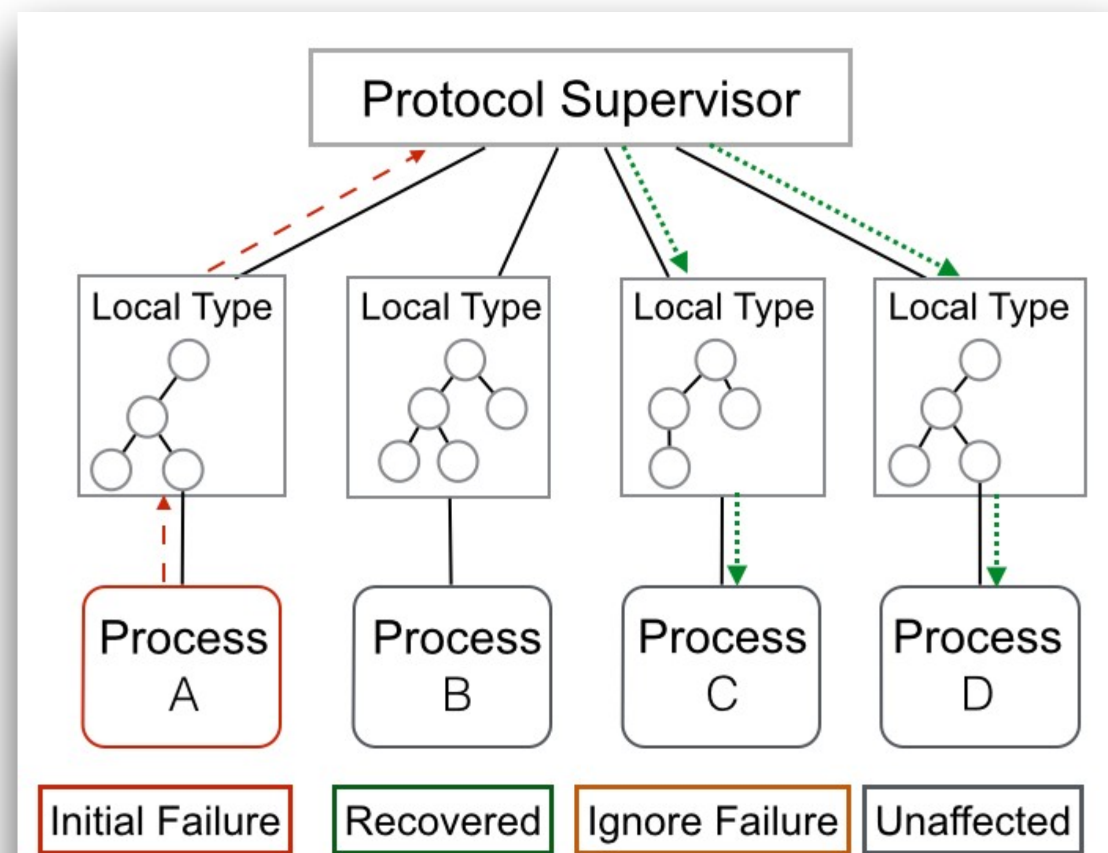



- A **recovered system** is free from deadlocks, orphan messages and reception error.
- Outperforms one of the built-in recovery strategies in Erlang

# 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



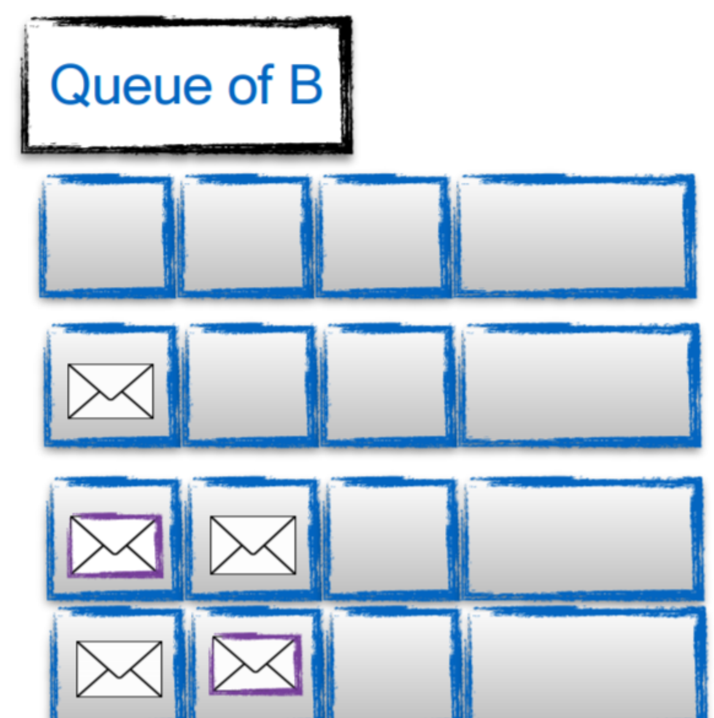
# Causalities

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

$$\underbrace{A \longrightarrow B;}_{n_1}\ \underbrace{B \longrightarrow C;}_{n_2} \qquad n_1 \prec_{io} n_2$$

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

$$\underbrace{A \longrightarrow B;}_{n_3}\ \underbrace{C \longrightarrow B;}_{n_4} \qquad n_3 \lhd n_4 \qquad n_3 \not\prec_{io} n_4$$



# Causalities

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

$$\underbrace{A \longrightarrow B;}_{n_1}\ \underbrace{B \longrightarrow \underline{C;}}_{n_2} \qquad n_1 \prec_{io} n_2$$

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

$$\underbrace{A \longrightarrow \underline{B;}}_{n_3}\ \underbrace{C \longrightarrow B;}_{n_4} \qquad n_3 \lhd n_4 \qquad n_3 \not\prec_{io} n_4$$

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

$$\underbrace{A \longrightarrow \underline{B;}}_{n_1}\ \underbrace{B \longrightarrow C;}_{n_2} \qquad n_1 \prec_{\dagger} n_2$$

# *Part Three*

# Recovery Algorithm

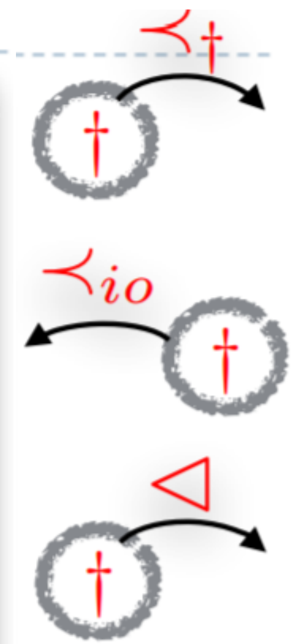
# Recovery Algorithm

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

$\mathscr{N} = \mathscr{N}^{\rightarrow} = \{\mathsf{n} \mid \mathsf{n}_i \lhd \mathsf{n} \wedge \mathsf{n} = \mathtt{r} \rightarrow \mathsf{p}\} \cup \{\mathsf{n}_i\}$
$\mathscr{S} = \{\mathsf{n} \mid ((\mathsf{n}_i \lhd \mathsf{n}' \wedge \mathsf{n}' = \mathsf{p} \rightarrow \mathtt{r}) \vee \mathsf{n}' = \mathsf{n}_i) \wedge \mathsf{n}' \twoheadleftarrow_{\mathtt{IO}} \mathsf{n}\} \setminus \{\mathsf{n}_i\}$
**repeat**
    $\mathscr{N}^{\leftarrow} = \{\mathsf{n} \mid \mathsf{n} \twoheadleftarrow_{\mathtt{IO}} \mathsf{n}' \vee (\mathsf{n} \lhd \mathsf{n}' \wedge \mathsf{n} \in \mathscr{S}) \wedge \mathsf{n}' \in \mathscr{N}^{\rightarrow}\}$
    $\mathscr{N}^{\rightarrow} = \{\mathsf{n} \mid \mathsf{n}' \lhd \mathsf{n} \wedge \mathsf{n}' \in \mathscr{N}^{\leftarrow}\} \setminus (\mathscr{N} \cup \mathscr{S})$
    $\mathscr{N} = \mathscr{N} \cup \mathscr{N}^{\leftarrow} \quad \mathscr{S} = \mathscr{S} \setminus \mathscr{N}^{\leftarrow}$
**until** $\mathscr{N}^{\leftarrow} = \mathscr{N}^{\rightarrow} = \varnothing$
**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 $\longrightarrow$ E; 2:C $\longrightarrow$ ~~E~~;

3:B $\longrightarrow$ A; 4:C $\longrightarrow$ A; 5:A $\longrightarrow$ ~~D~~;

6:D $\longrightarrow$ E; 7:E $\longrightarrow$ B;


| 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:3, B:3, C:4 |
| A | A:3, B:3, C:5 |
| 3, B | C:2, E:2 |
| 4, C | C:1, B:1, … |
| 4, A | … |
| … | |

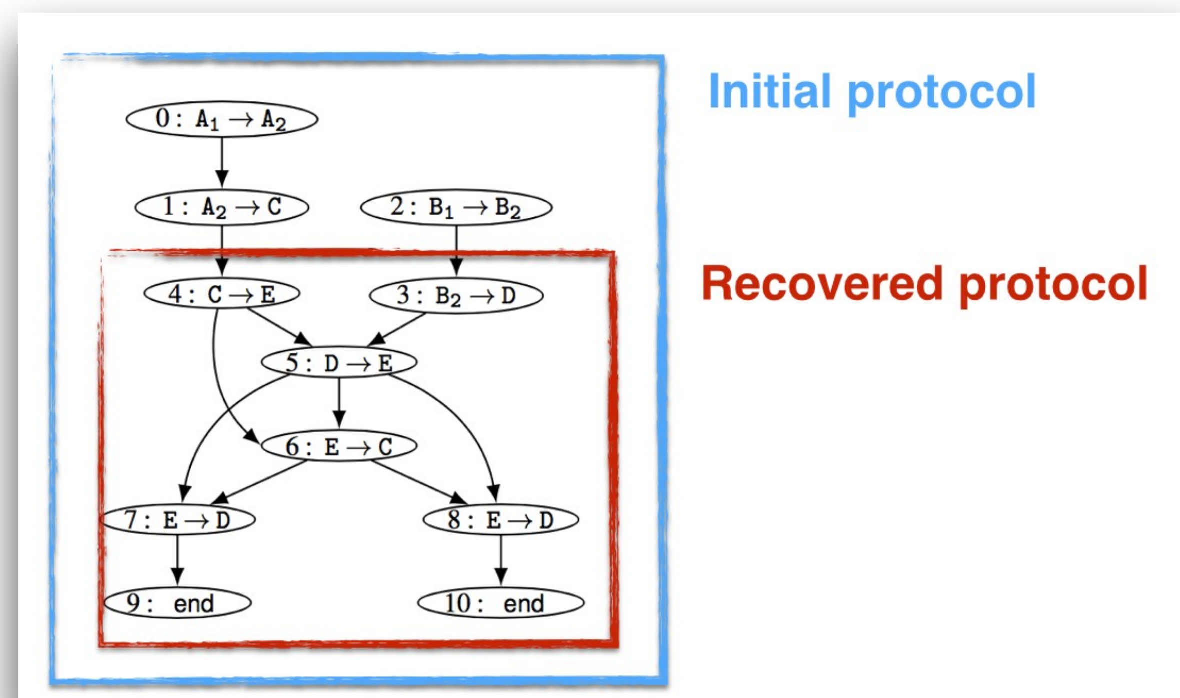
# Main Results: Transparency and Safety (informally)

**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

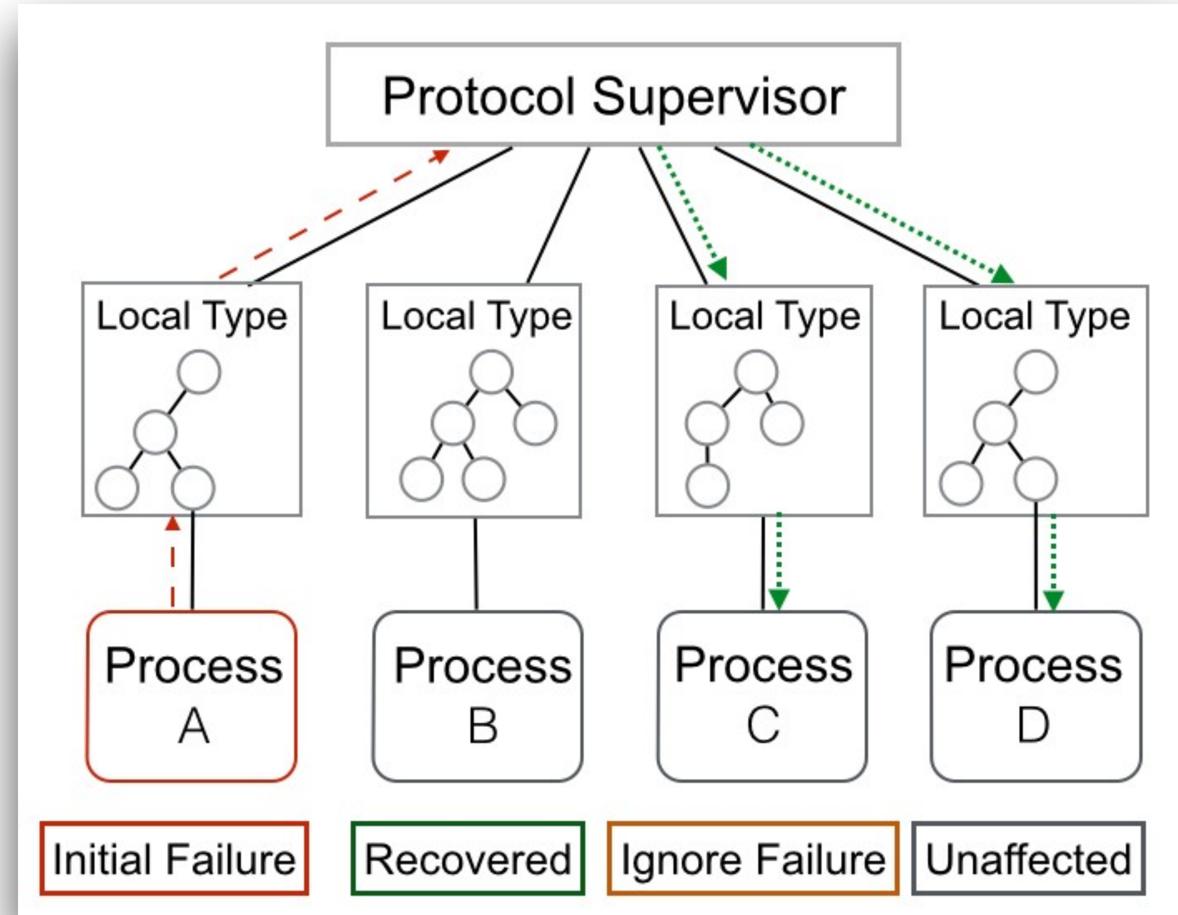
Any reachable configuration which is an initial configuration of well-formed global protocol is free from deadlock, an orphan massage and a reception error.



*Part Four*

# Recovery Implementation

# Enabling Protocol Recovery in ERLANG




**protocol supervisor**

(recover processes)

**local supervisors**

(monitor the process behaviour)

**gen_server**

(used to implement processes)


**gen_server**

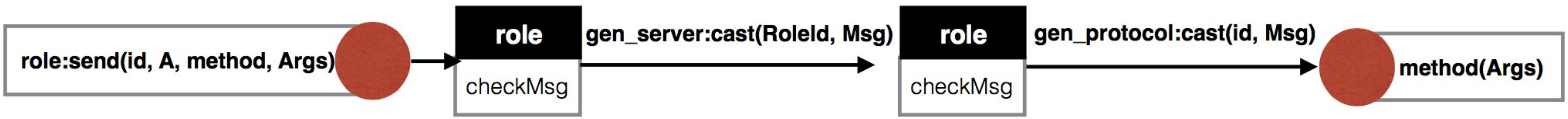
mnesia

**stores recovery tables**

Scribble

**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;
  }
}
```

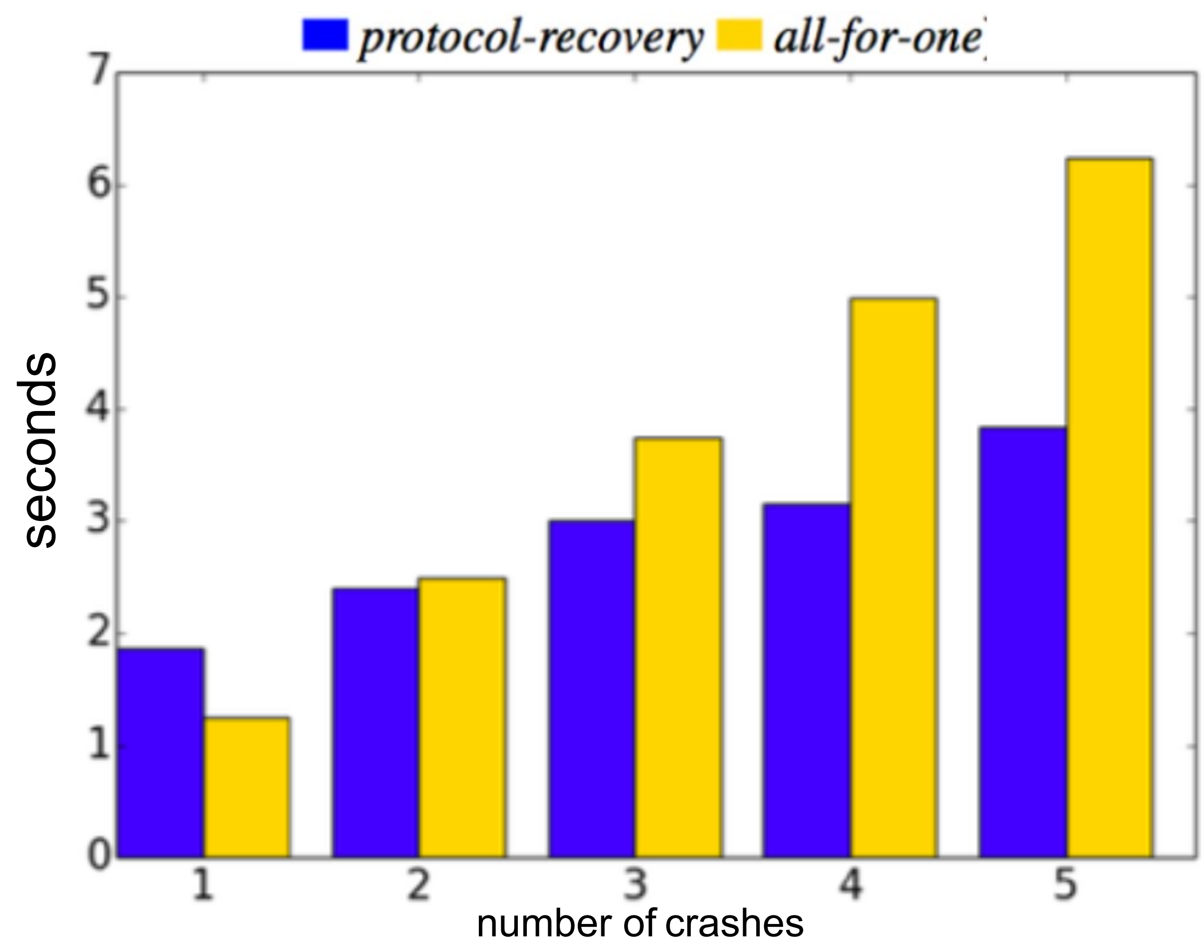
```
% Handlers for C and D
quote({msg,Val},State) →
 role:send(State#state.role, ?E, quote, Val).
```

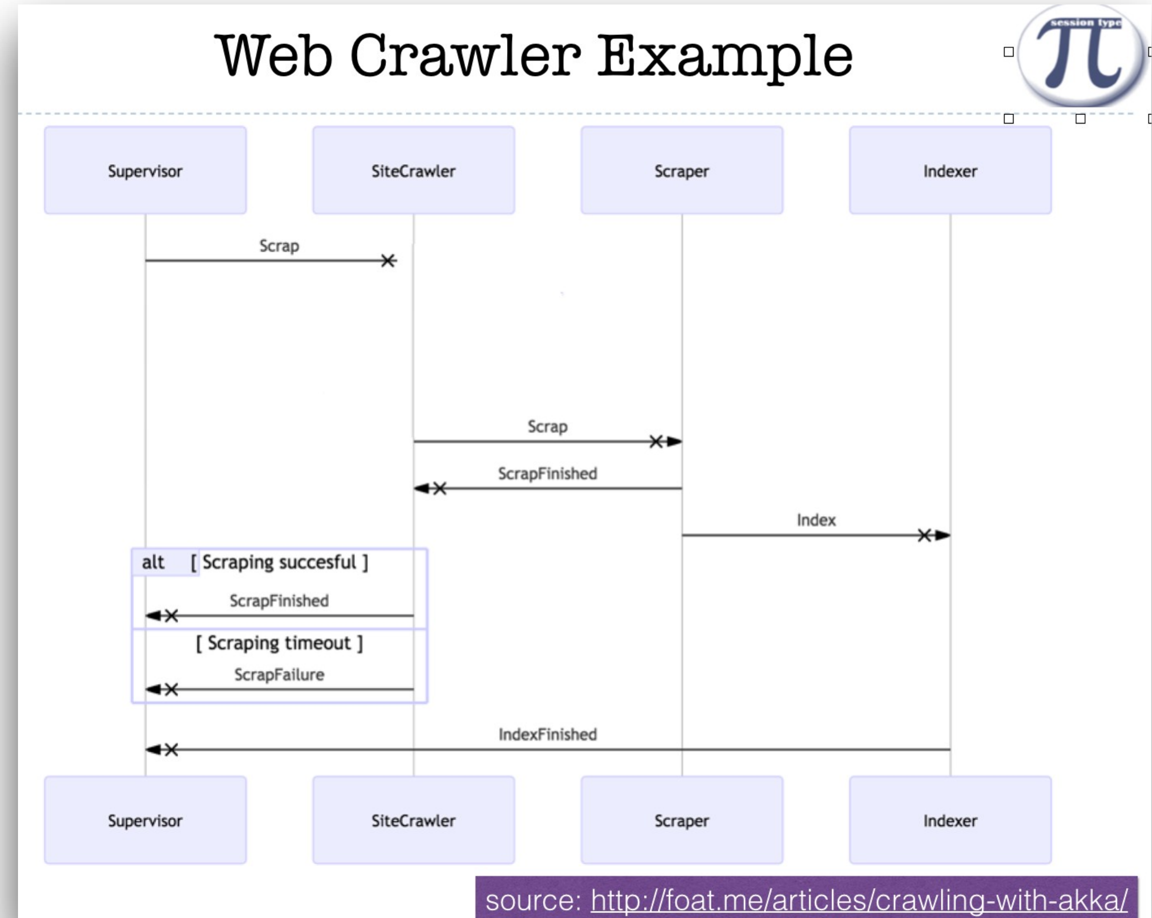
```
% Handlers for E
quote({msg, Val},State) when State.prev==undef →
  {noreply,State#state{prev=Val}};

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

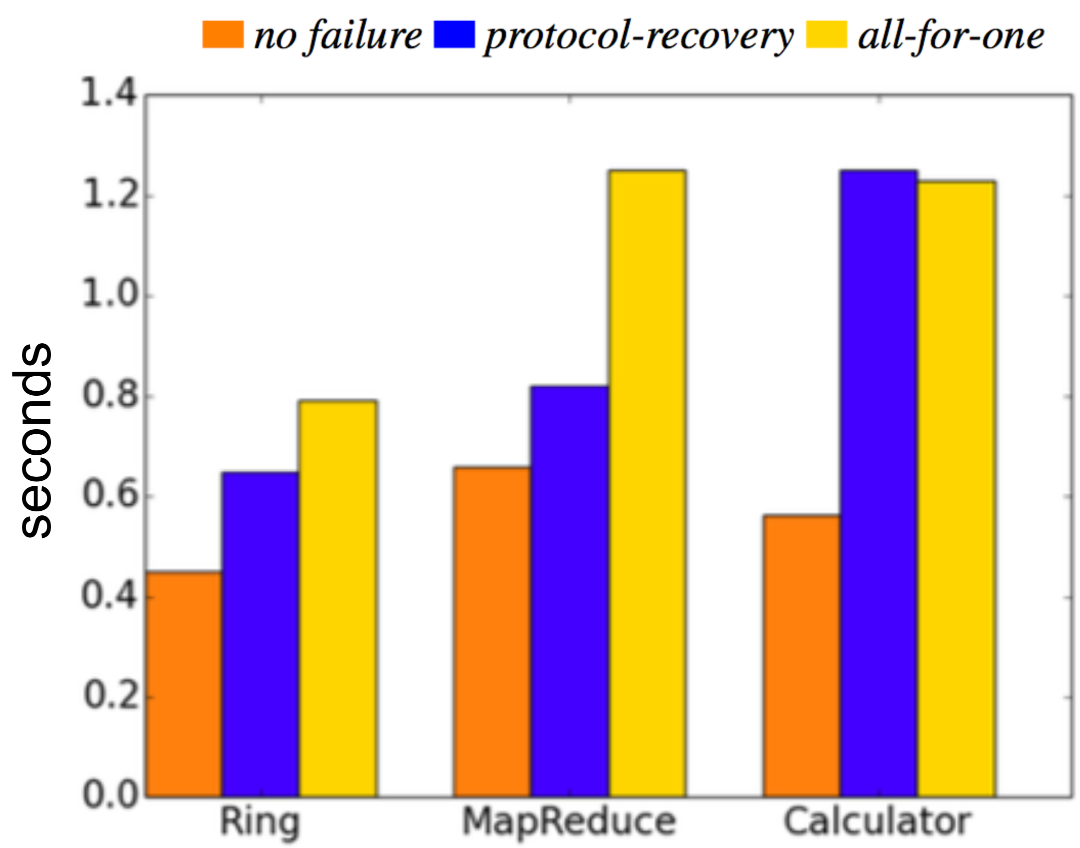



- A process is chosen at random at the start
- Improvement when several failures occur
- By mistake initially we implemented all-for-one that introduced a deadlock

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

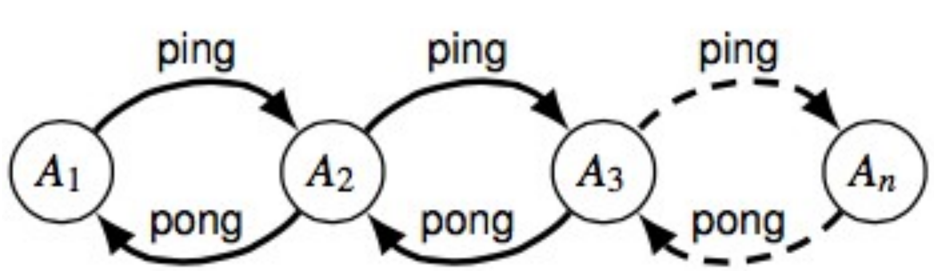
# Evaluation: Concurrency Patterns



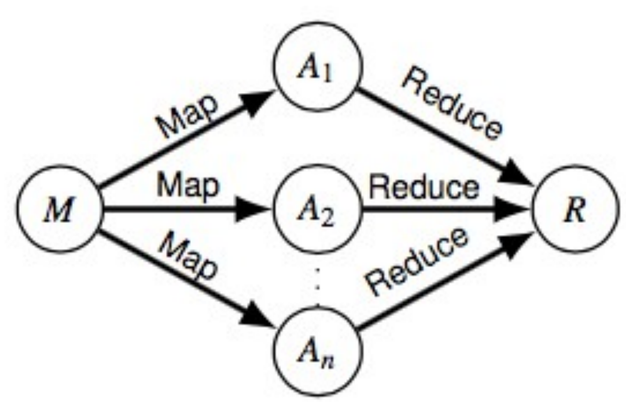

| Example | #roles | #states | GRT (sec) | affected roles |
|---|---|---|---|---|
| MapReduce [21] | `n+1` | `n+2` | 0.11 | `W[1] ... W[n]` |
| Ring [21] | `n` | `2*n` | 0.16 | `W[1] ... W[n]` |
| Calculator [18] | `n+1` | `4*n` | 0.75 | `A[1]` |

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

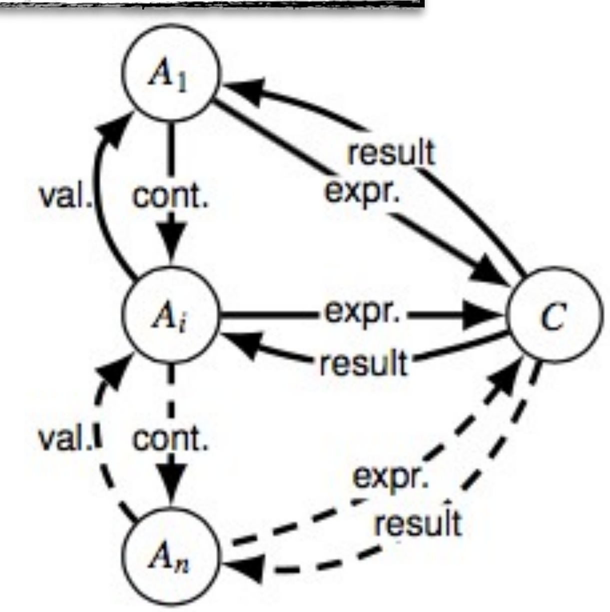
## Ring




## 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

# 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/