



# 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).
```

A word cloud of various terms associated with the Erlang programming language. The words are arranged in a roughly circular shape, with some terms being larger and more prominent than others. The terms include: Messaging, NetworkServices, Scalable, DistributedSystems, Reliable, Concurrent, Web, FaultTolerant, HighThroughput, Infrastructure, NetworkSecuritySystem, StandaloneProjects, TransactionProcessing, NetworkSystemsWithNoLoad, GameServers, NotStringProcessing, NotEmbedded, Stability, GeneralPurpose, DataScience, SystemsIntegration, Throughput, NotPerformance, Routing, SoftRealtime, ComplexSystems, Telco, RealMens, Everything, NotGui, Actors, Many, IoIntensive, DataBrowsing, GridComputing, Scripting, Exchanges, LongRunning, None, Teaching, Multicore, Prototypes, LowLatency, ClientServer, Systems, Realtime, Robust, Embedded, SmallTeams, NotProduction, SmallProjects, Playing, Middleware, CoordinationAndControl, NotRealtime, AlmostAll, Parallel, NewProjects, BinaryDataProcessing, BusinessSoftware, HomeCoding, Upgradable, PresenceSystem, StreamOrientedData.

Upgradable PresenceSystem StreamOrientedData  
HomeCoding BusinessSoftware  
NewProjects BinaryDataProcessing  
AlmostAll Parallel Messaging CoordinationAndControl  
NotRealtime  
Playing Middleware  
NotProduction SmallProjects NetworkServices  
Robust Embedded SmallTeams Scalable  
ClientServer Systems Realtime  
LowLatency Prototypes  
Multicore  
Orchestration Teaching DistributedSystems  
LongRunning None  
Exchanges  
Scripting Reliable Available  
GridComputing  
DataBrowsing  
IoIntensive Actors NotGui CommunicationServices  
Many Everything Concurrent  
RealMens ComplexSystems Telco  
Routing SoftRealtime  
NotPerformance Throughput Databases Web NotCompute  
SystemsIntegration  
DataScience Games  
Stability GeneralPurpose FaultTolerant  
NotEmbedded HighThroughput  
GameServers  
NotStringProcessing  
Infrastructure NetworkSystemsWithNoLoad  
TransactionProcessing  
StandaloneProjects  
NetworkSecuritySystem

# 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:

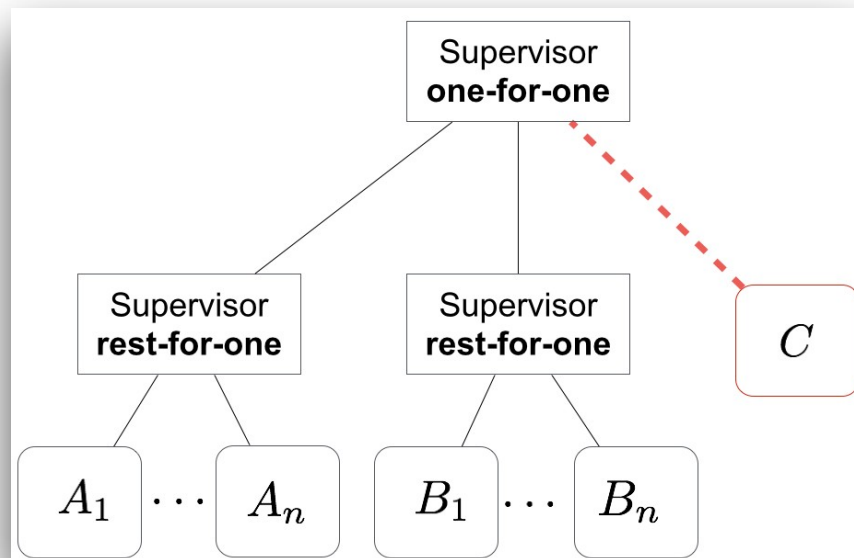
\*\*\* STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

\*\*\* 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

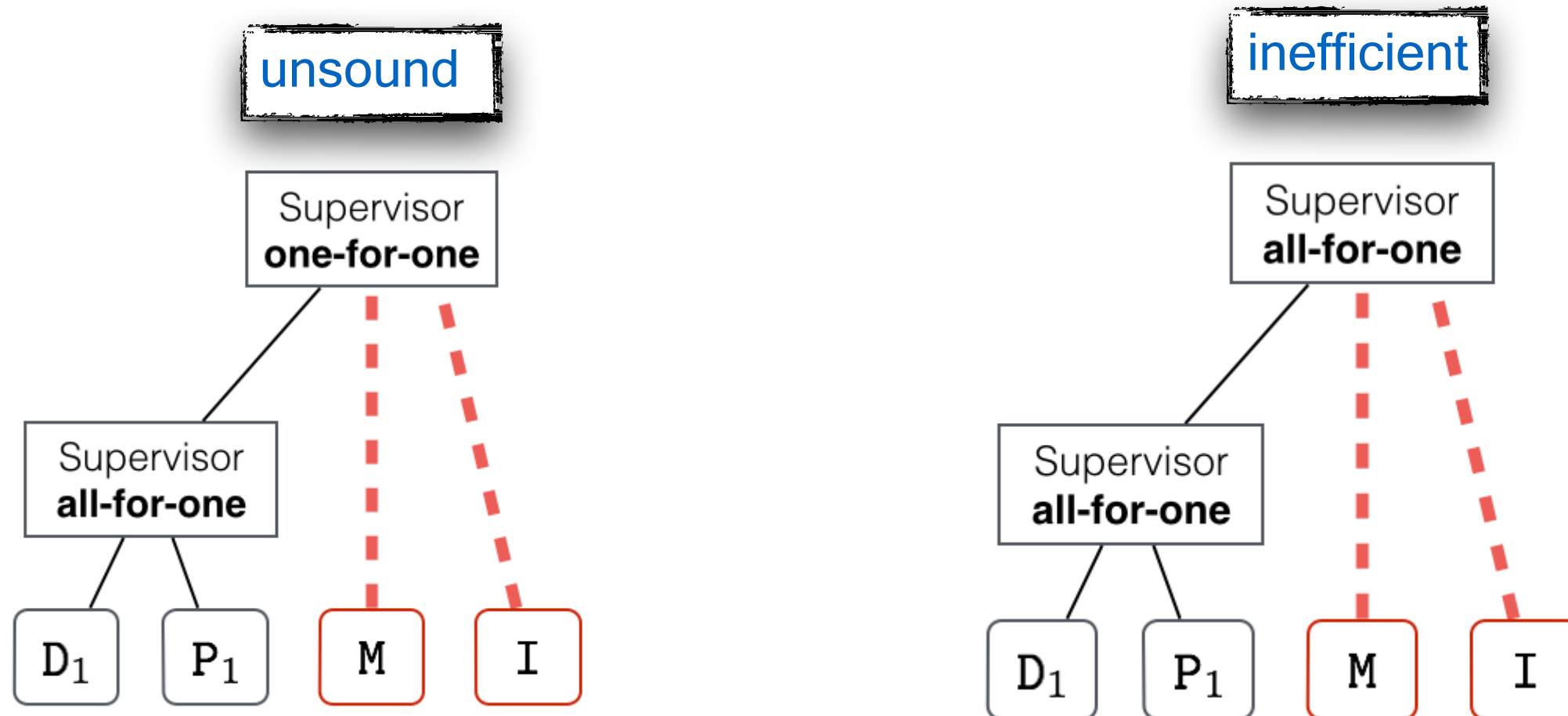
- one-for-one
- all-for-one
- rest-for-one

- 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



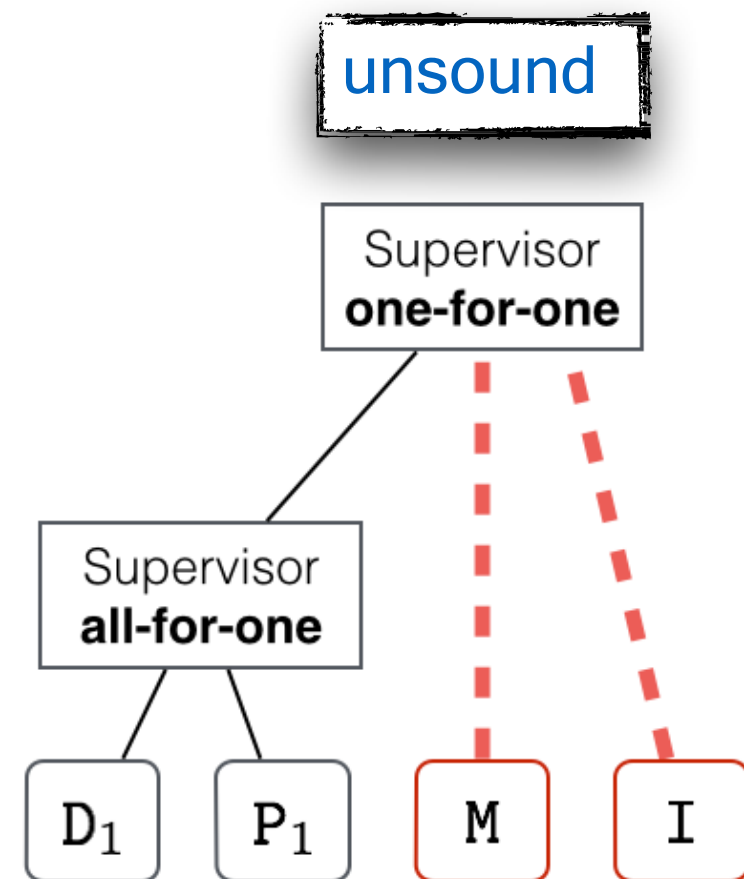
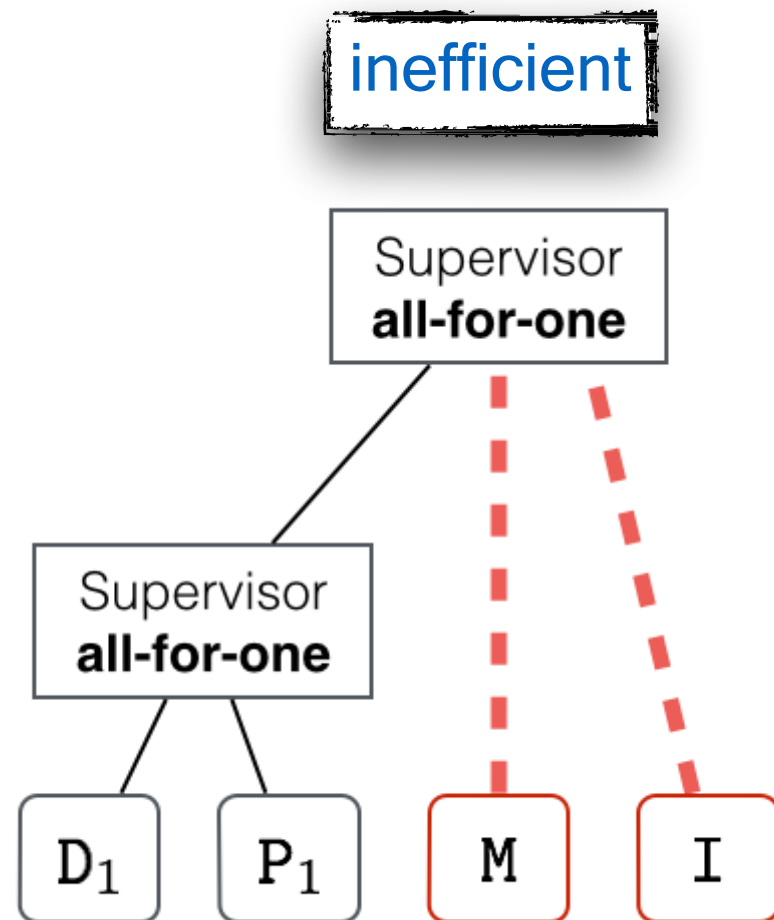
# Supervision strategies: Drawbacks

- Supervision strategies are: statically defined, error-prone



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

# How to generate *sound and efficient* supervision strategies?

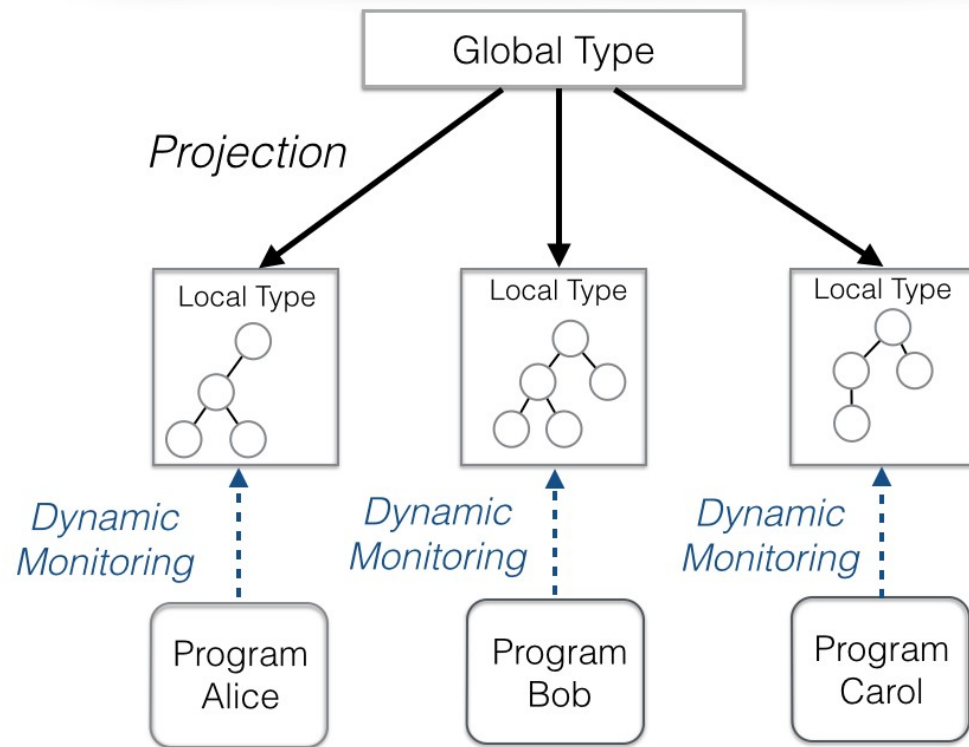


By using Session Types!



# Session Types Overview

## Dynamic Monitoring [RV'13, COORDINATION'14, FMDS'15]



- Global protocol (session type)

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

- 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

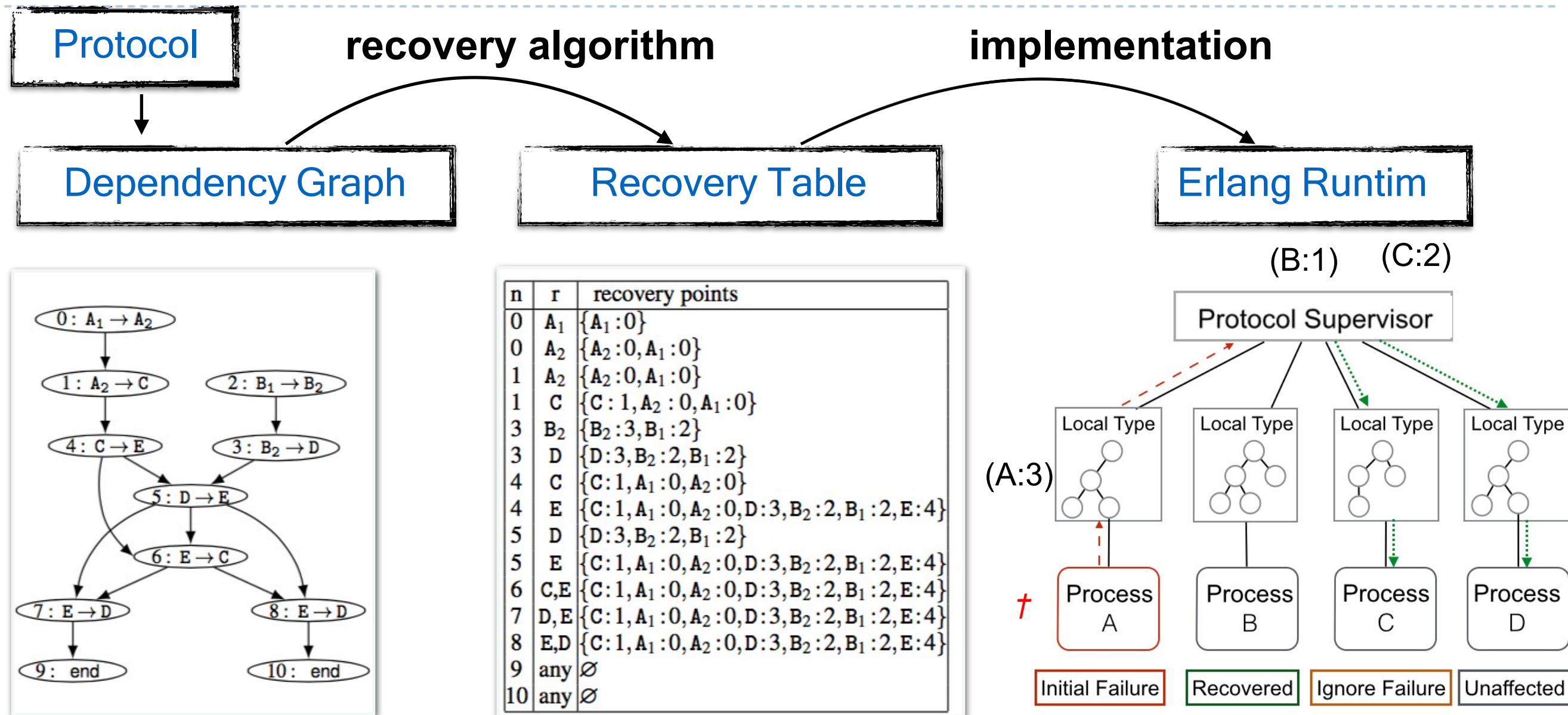
- Execution model of I/O actions by roles

- 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

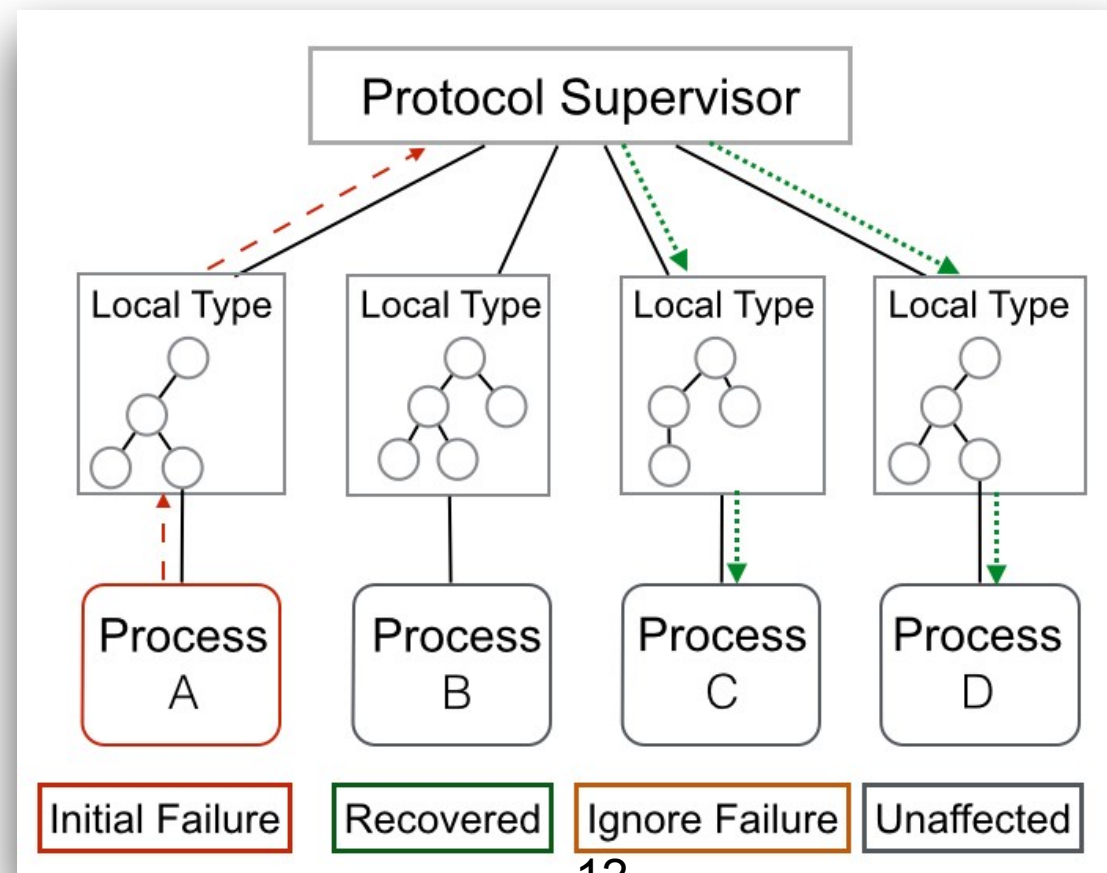


- A **recovered system** is free from deadlocks, orphan messages and reception error.
- Outperforms one of the built-in recovery strategies in Erlang

# This talk: Safe Recovery for Session Protocols

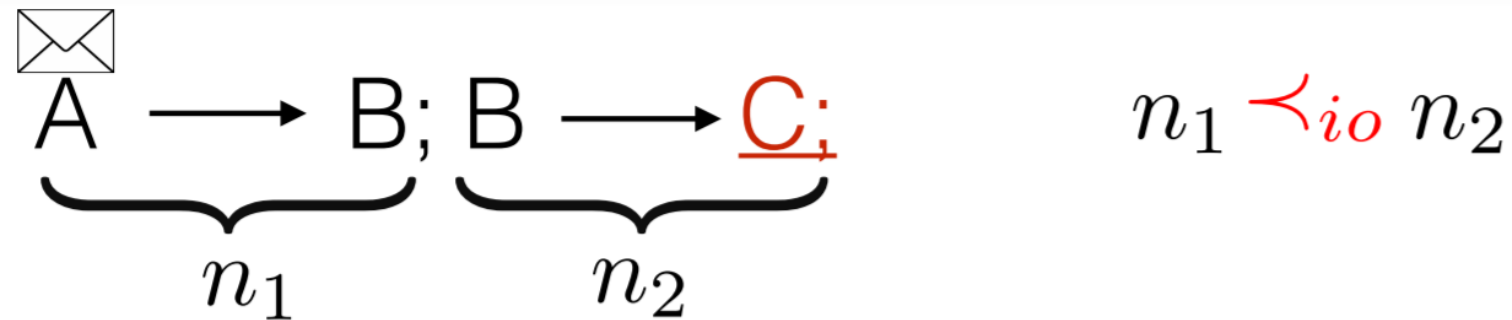
## Approach

- **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

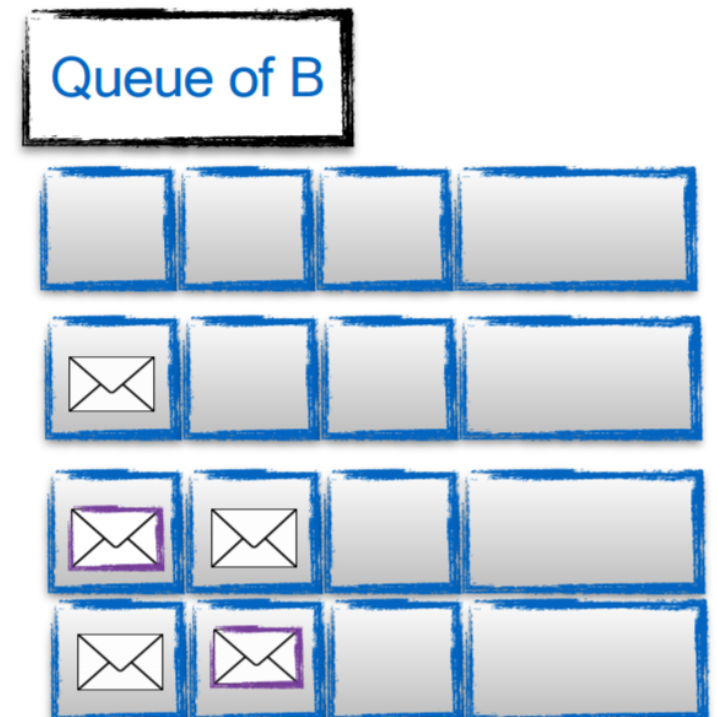
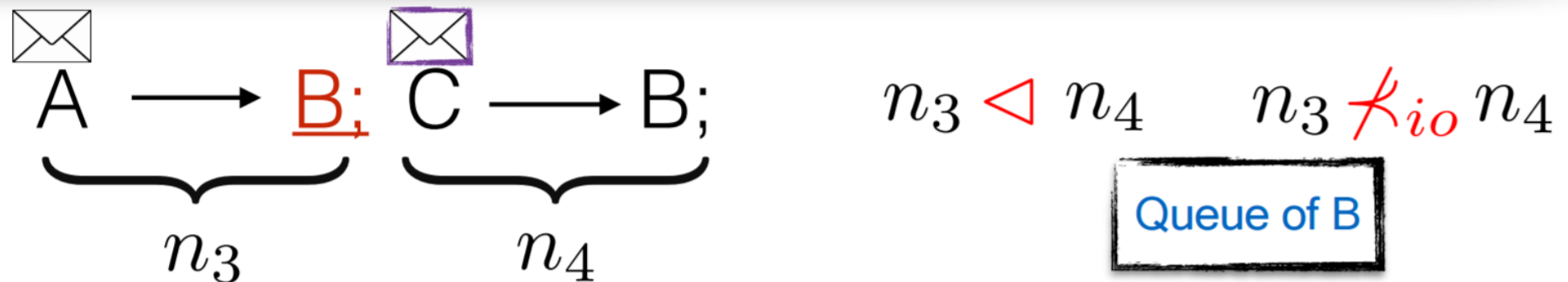


# Causalities

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



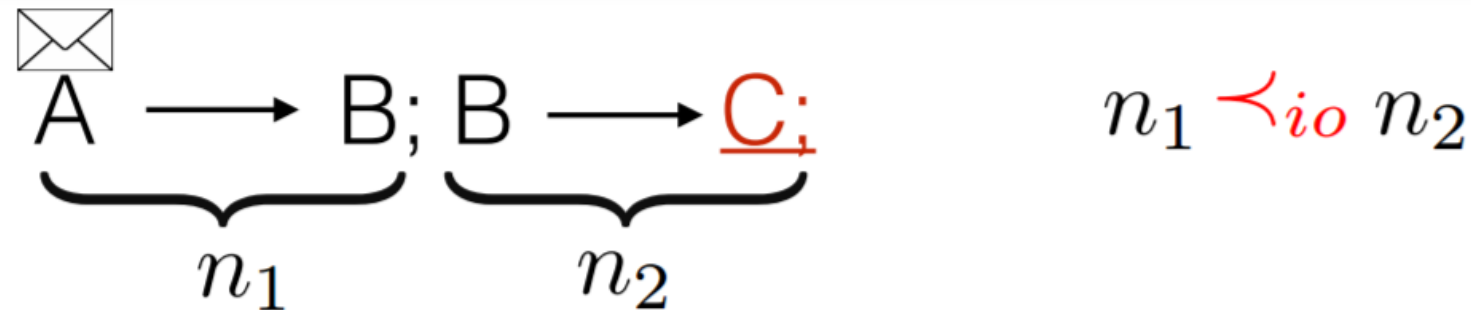
$\triangleleft$  -precedence dependencies (represent the order between two nodes which have a common participant) should recover



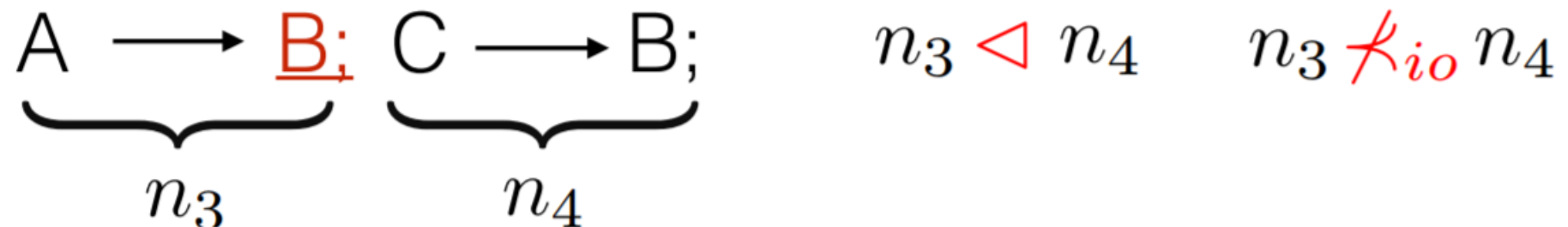


# Causalities

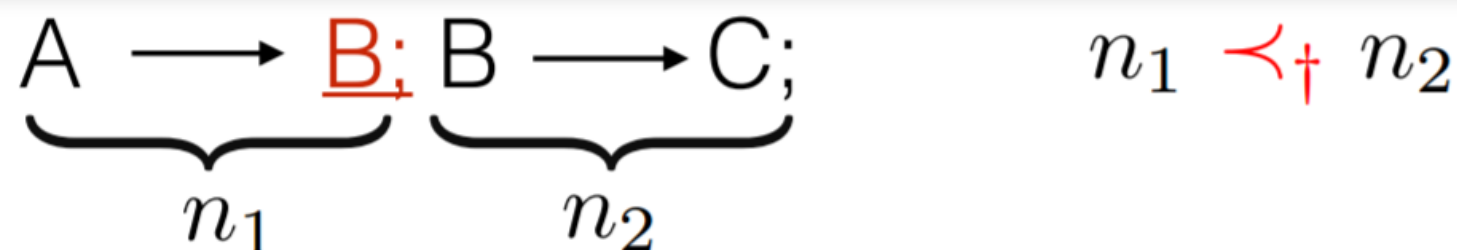
$\prec_{io}$  -input-output dependencies (assert the order between a reception of a message and a send action) should recover



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



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



*Part Three*

Recovery Algorithm

# Recovery Algorithm

## **Algorithm** *Calculating affected nodes*

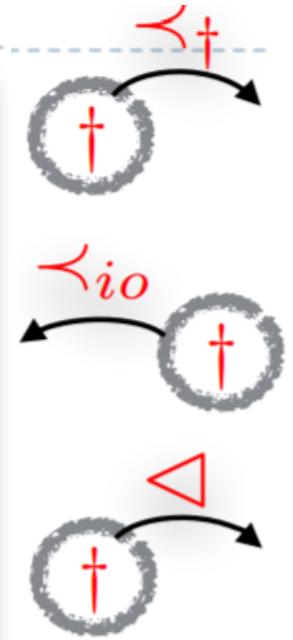
**Input:**  $n_i$  (a failed node),  $p$  (a failed role)

**Output:**  $\mathcal{N}$  (a set of affected nodes)

1.  $\mathcal{N} = \mathcal{N}^{\rightarrow} = \{n \mid n_i \triangleleft n \wedge n = r \rightarrow p\} \cup \{n_i\}$
2.  $\mathcal{S} = \{n \mid ((n_i \triangleleft n' \wedge n' = p \rightarrow r) \vee n' = n_i) \wedge n' \ll_{IO} n\} \setminus \{n_i\}$
3. **repeat**
4.      $\mathcal{N}^{\leftarrow} = \{n \mid n \ll_{IO} n' \vee (n \triangleleft n' \wedge n \in \mathcal{S}) \wedge n' \in \mathcal{N}^{\rightarrow}\}$
5.      $\mathcal{N}^{\rightarrow} = \{n \mid n' \triangleleft n \wedge n' \in \mathcal{N}^{\leftarrow}\} \setminus (\mathcal{N} \cup \mathcal{S})$
6.      $\mathcal{N} = \mathcal{N} \cup \mathcal{N}^{\leftarrow}$       $\mathcal{S} = \mathcal{S} \setminus \mathcal{N}^{\leftarrow}$
7. **until**  $\mathcal{N}^{\leftarrow} = \mathcal{N}^{\rightarrow} = \emptyset$
8. **return**  $\mathcal{N}$

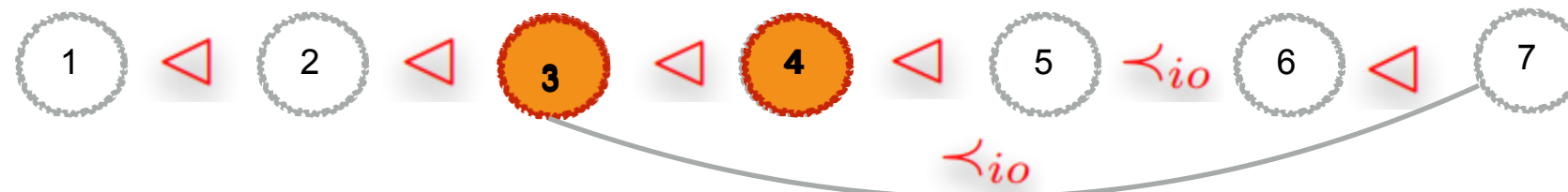
# Recovery Algorithm

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



- Step 1: Initialise the  $\prec_+$  dependencies of the failed node
- Step 2: Backward traversal of  $\prec_{io}$  dependencies
- Step 3: Forward Traversal of  $\triangleleft$  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

$\prec_{io}$

$\triangleleft$

Final Condition

$\prec_+$ : 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	...
...	

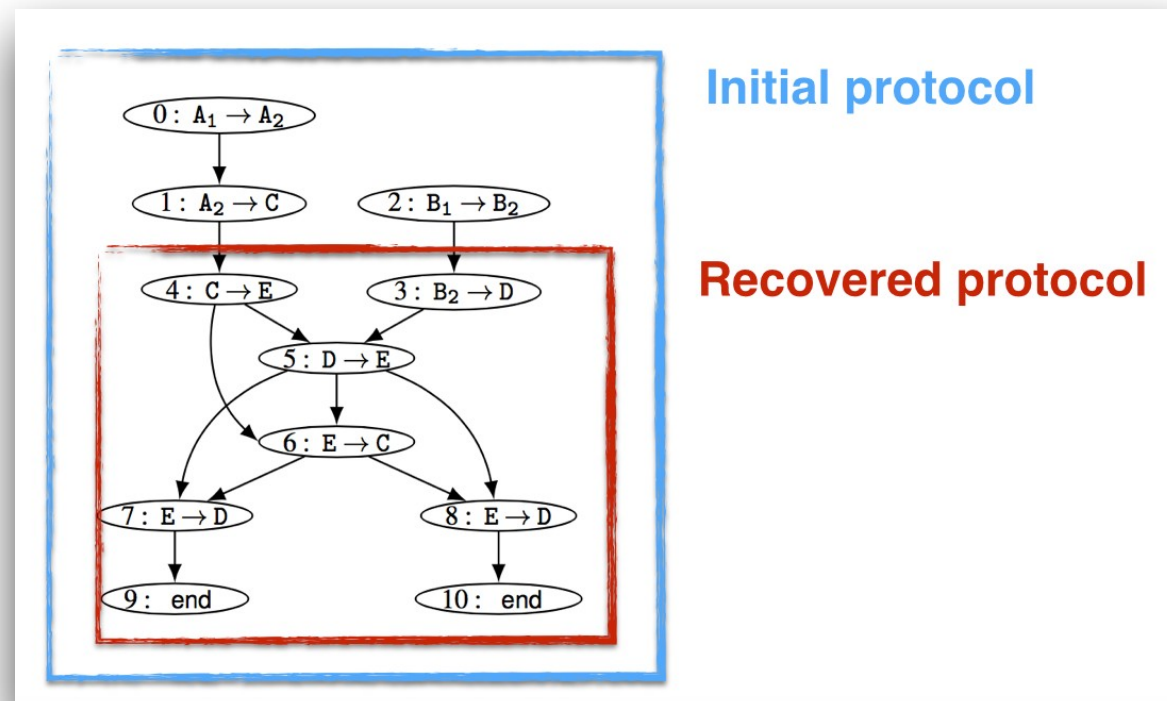
# Main Results: Transparency and Safety (informally)

## 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

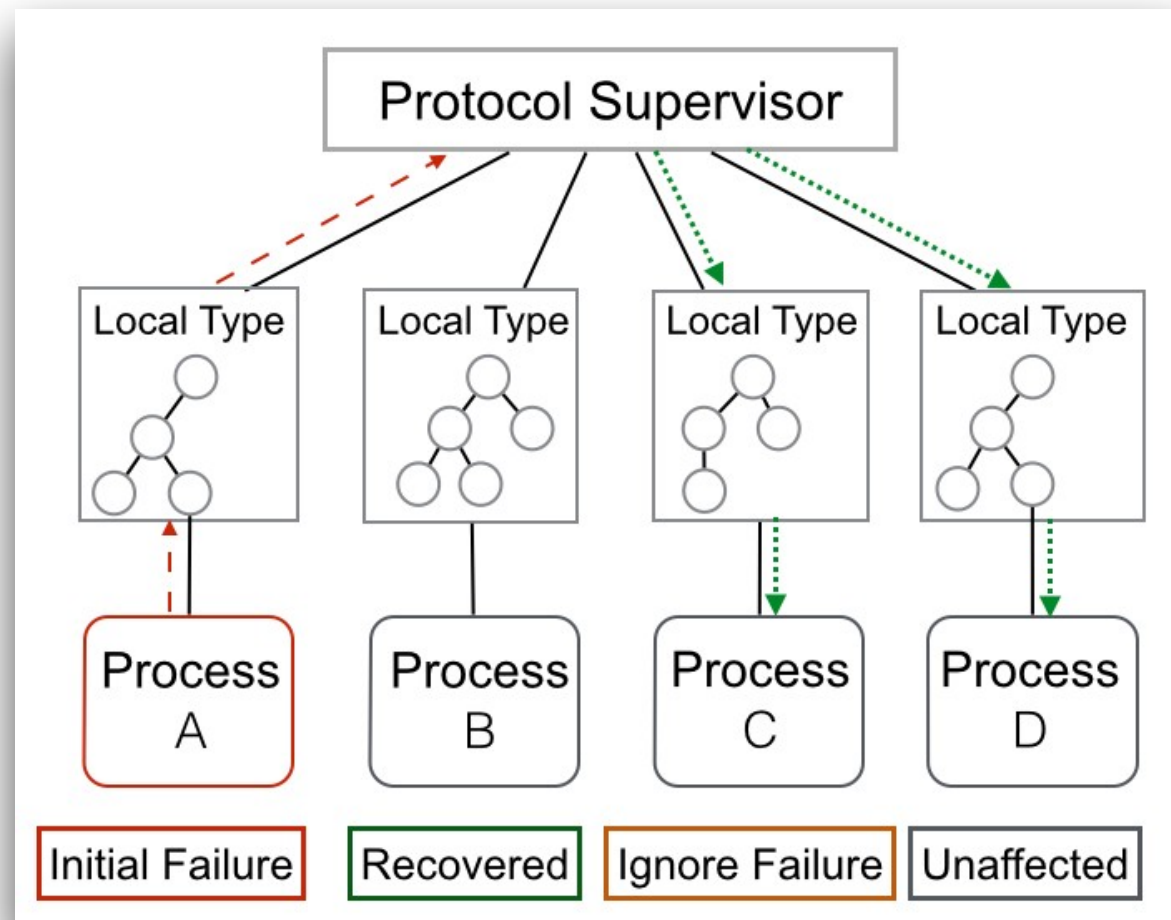
Any reachable configuration which is an initial configuration of well-formed global protocol is free from deadlock, an orphan message and a reception error.



## *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



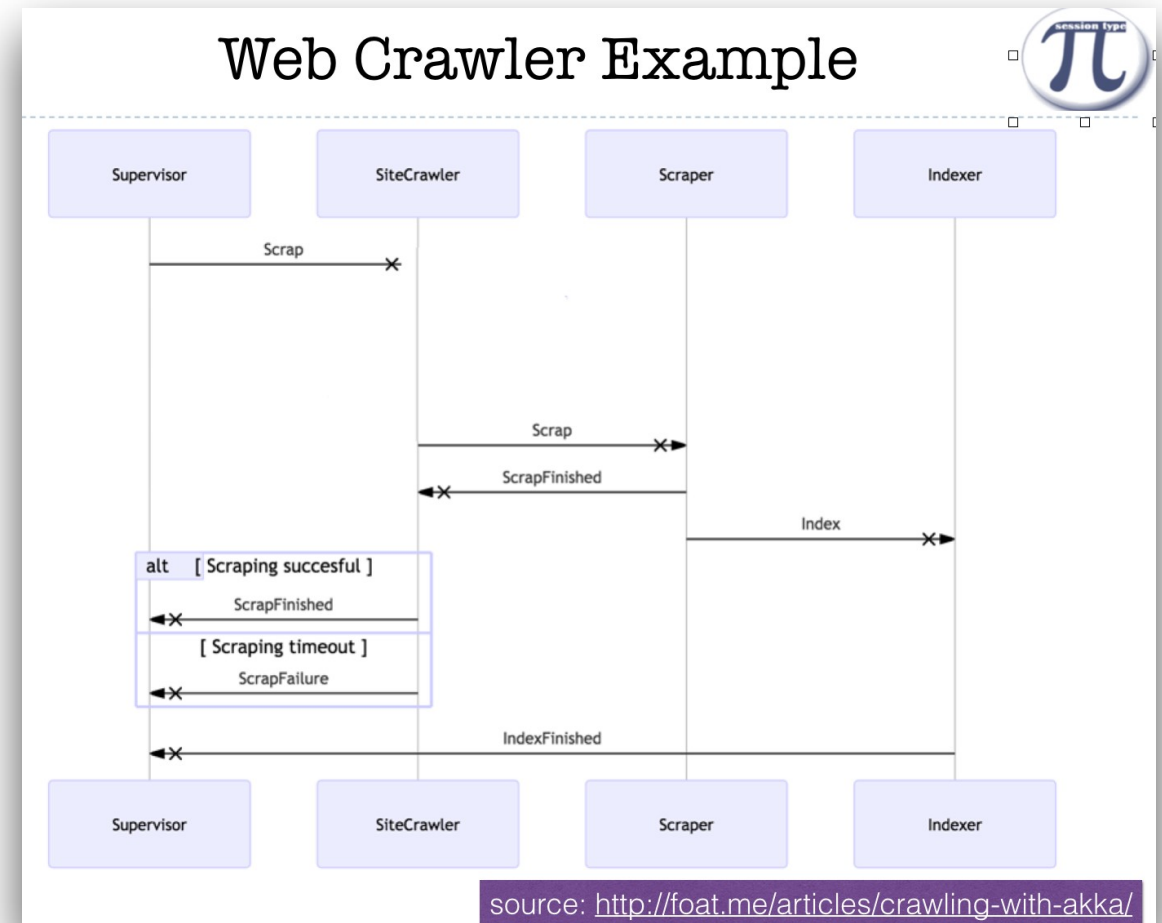
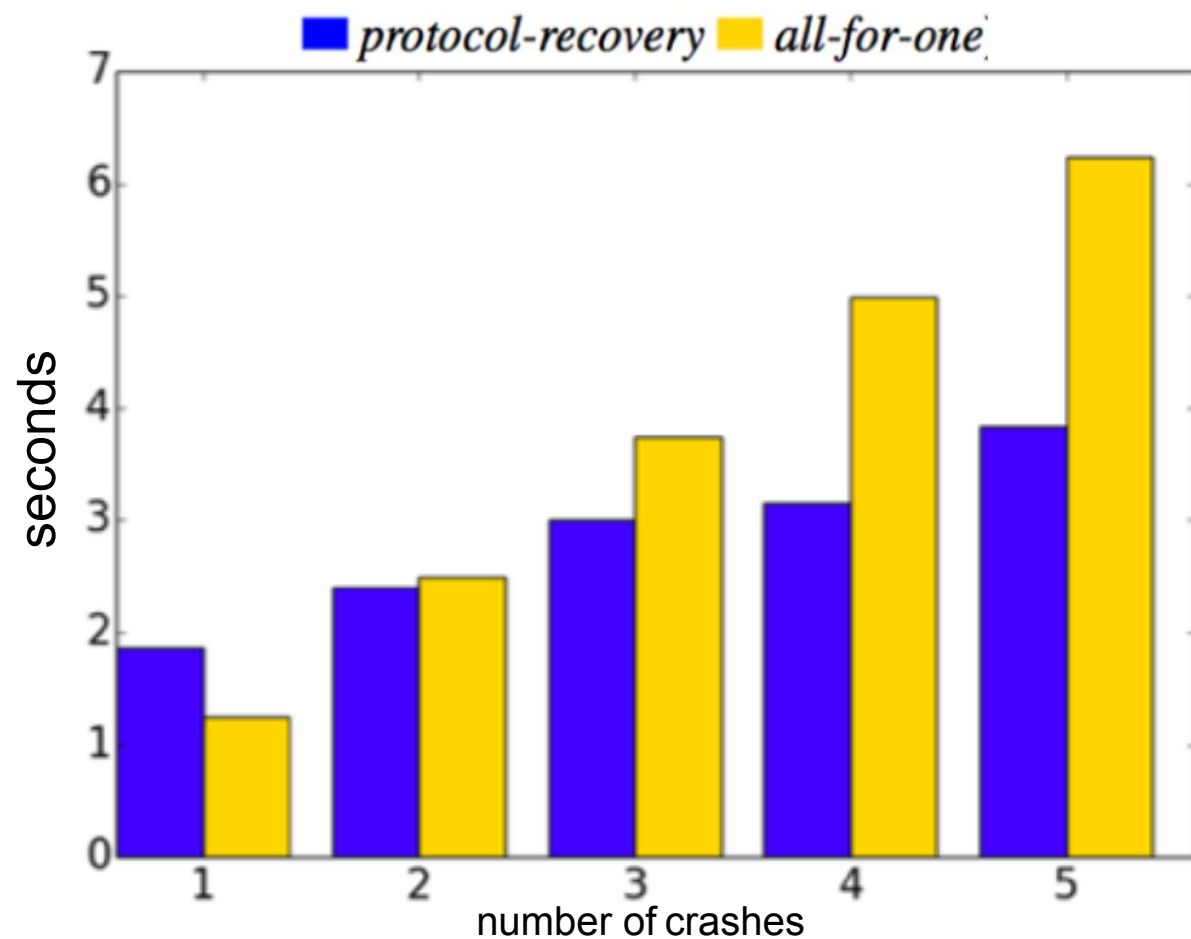
```
global protocol Trading (  
  role A, role B,  
  role C, role D){  
  
  quote(int) from A to C;  
  
  quote(int) from B to D;  
  
  quote(int) from C to E;  
  quote(int) from D to E;  
  choice at E {  
    accept() from E to C;  
    accept() from E to D;  
  or {  
    reject() from E to C;  
    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}.
```

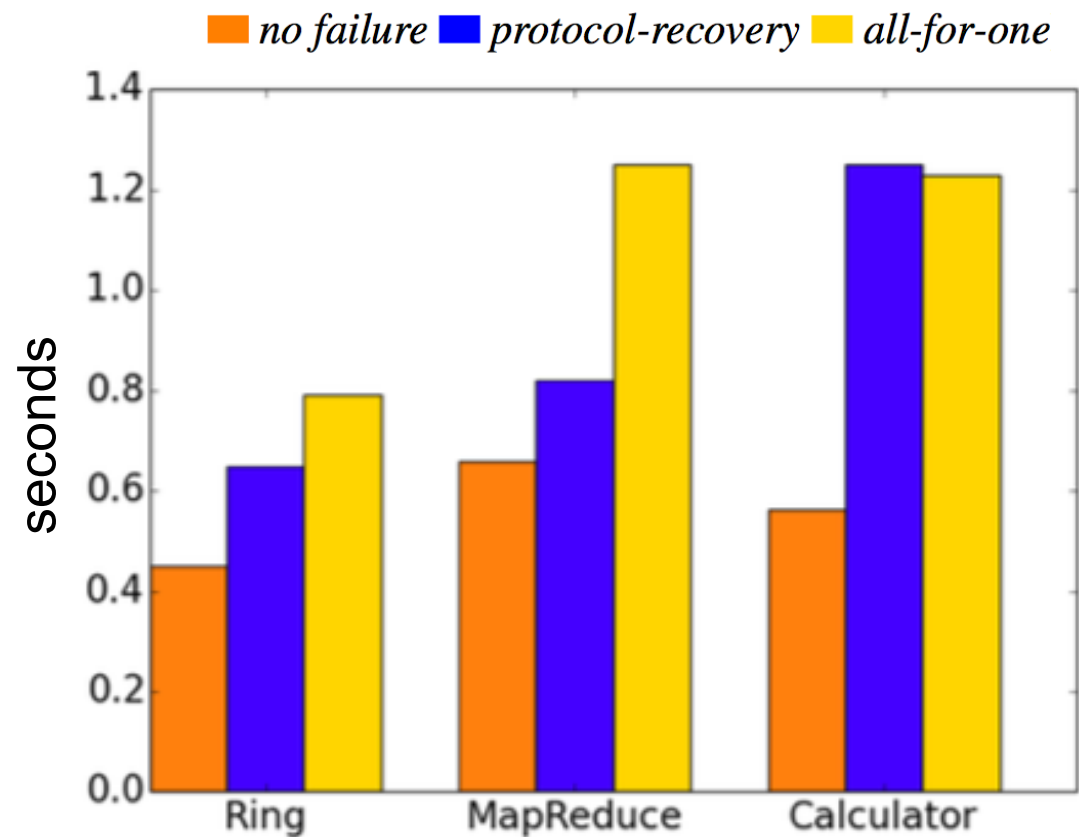


# Evaluation: Web Crawler Example



- 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

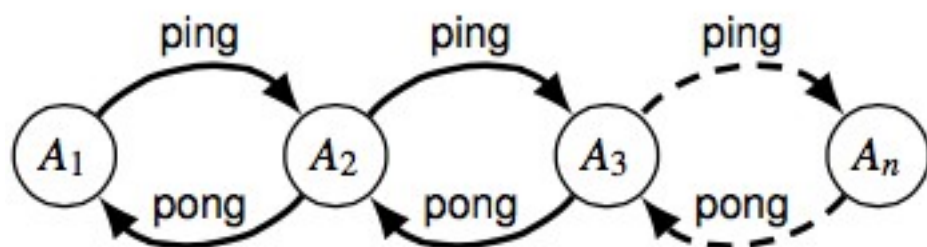
# Evaluation: Concurrency Patterns



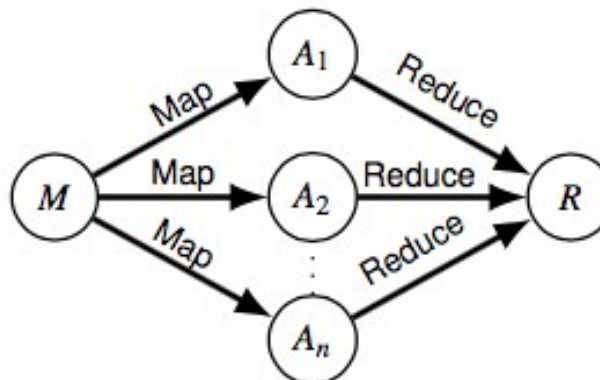
Example	#roles	#states	GRT (sec)	affected roles
MapReduce [21]	$n+1$	$n+2$	0.11	$W[1] \dots W[n]$
Ring [21]	$n$	$2*n$	0.16	$W[1] \dots W[n]$
Calculator [18]	$n+1$	$4*n$	0.75	$A[1]$

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

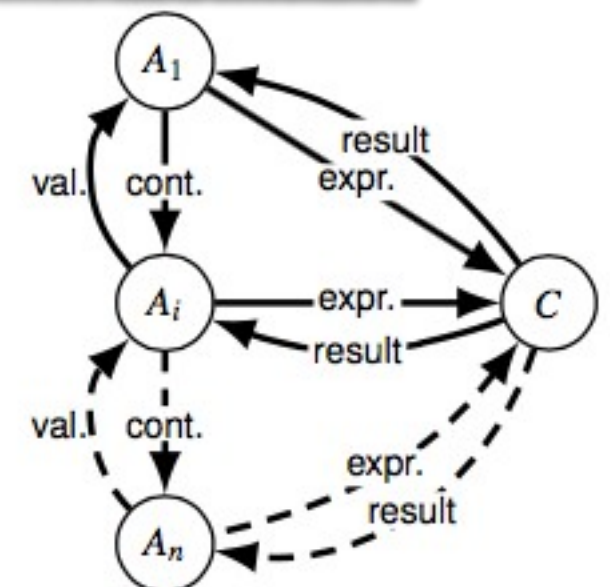
Ring



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](http://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](http://scribble.doc.ic.ac.uk)
- Project source: <https://gitlab.doc.ic.ac.uk/rn710/codeINspire>
- MRG webpage: <http://mrg.doc.ic.ac.uk/>