# Safe Sessions for Erlang

Adrián Palacios
(joint work with Ivan Lanese, Naoki Nishida and Germán Vidal)

Technical University of Valencia

PLDI 2017 Student Research Competition

June 21, 2017
Barcelona, Spain

## Introduction

**Breaking News!**

**Erlang** systems are fault-tolerant!

...thanks to the *Let It Crash* philosophy

## Introduction

**Breaking News!**

**Erlang** systems are fault-tolerant!

...thanks to the *Let It Crash* philosophy

## The Let It Crash philosophy

"Do not handle errors in your programs.
If a process is about to crash, let it crash and restart it immediately"

Made possible by a process supervision tree where:

- **Workers**: Do all the hard work
- **Supervisors**: Restart workers if they crash

## The Let It Crash philosophy

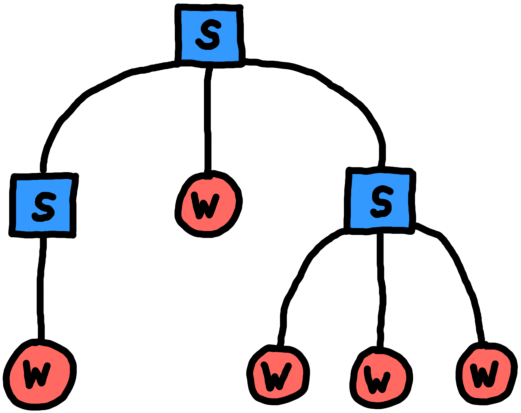
"Do not handle errors in your programs.
If a process is about to crash, let it crash and restart it immediately"

Made possible by a process supervision tree where:

- **Workers**: Do all the hard work
- **Supervisors**: Restart workers if they crash

## Supervision tree

- **Workers**: Do all the hard work
- **Supervisors**: Restart workers if they crash



## Our proposal

Supervisors do not make any verification when restarting workers
$\rightarrow$ **inconsistent system state**

**Safe Sessions**

We propose **safe sessions**, an automatic recovery strategy for Erlang, as a complement to the *Let It Crash* philosophy

In safe sessions, concurrent actions are registered, so that the system can return to a safe state in case of error.

Based on the reversible semantics for **Erlang** from [LOPSTR'16].

## Our proposal

Supervisors do not make any verification when restarting workers
$\rightarrow$ **inconsistent system state**

### Safe Sessions

We propose **safe sessions**, an automatic recovery strategy for Erlang, as a complement to the *Let It Crash* philosophy

In safe sessions, concurrent actions are registered, so that the system can return to a safe state in case of error.

Based on the reversible semantics for **Erlang** from [LOPSTR'16].

## Our proposal

Supervisors do not make any verification when restarting workers
$\rightarrow$ **inconsistent system state**

### Safe Sessions

We propose **safe sessions**, an automatic recovery strategy for Erlang, as a complement to the *Let It Crash* philosophy

In safe sessions, concurrent actions are registered, so that the system can return to a safe state in case of error.

Based on the reversible semantics for **Erlang** from [LOPSTR'16].

## The **Erlang** language

## Erlang's features

The **Erlang** language has:

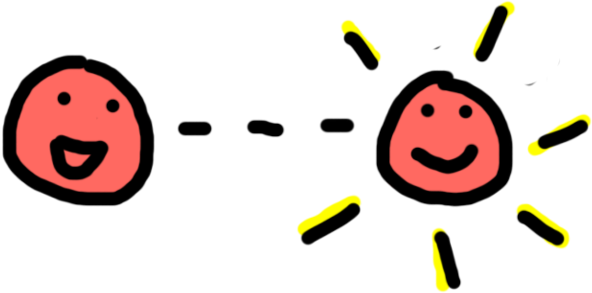
- functional and concurrent features
- concurrency based on the actor model

These features make it appropiate for distributed applications

Ericsson WhatsApp Messenger (Facebook chat) Ejabberd

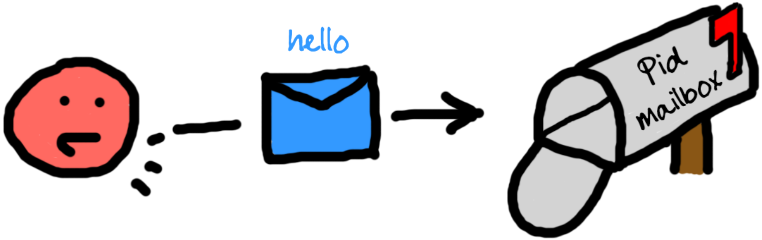
## Concurrent actions: Spawn

**Spawn:** Create a new process



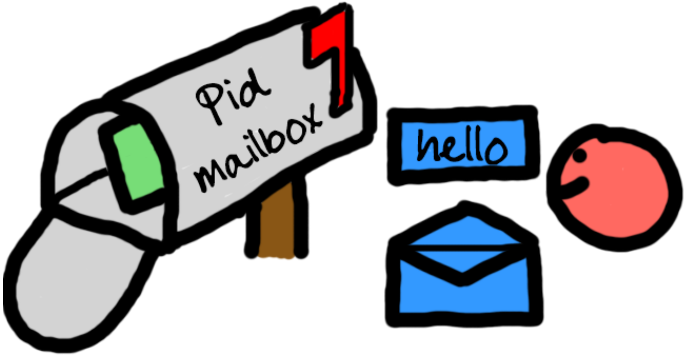
## Concurrent actions: Send

**Send:** Send a message to another process

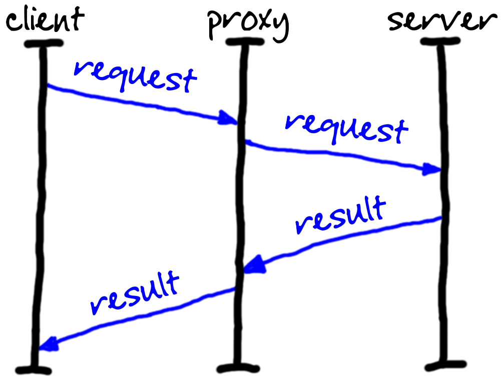


## Concurrent actions: Receive

**Receive:** Suspend execution until a message from the mailbox matches any of the receive clauses

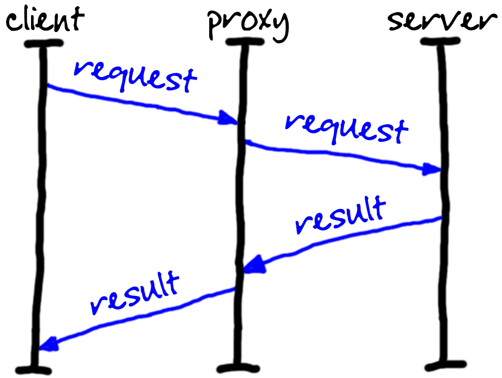


## Example



Many things can **go wrong** !

## Example



Many things can **go wrong** !

## Safe Sessions

## Safe Sessions

We add a new construct to **Erlang**

$\boxed{\text{safetry} \quad \textit{expr} \quad \text{end}}$

If *expr* goes wrong, we **restore** the process

## Safe Sessions

safetry *expr* end

If *expr* goes wrong, we **restore** the process

**Causal Consistency**

An action may be undone only if every action caused by that action has not been executed yet or has been undone

**Restoring** the state is not enough to ensure causal consistency, we must also undo the effects of its

- spawn actions
- send actions

We solve this by "propagating the safety" to dependent processes.

## Safe Sessions

safetry *expr* end

If *expr* goes wrong, we **restore** the process

### Causal Consistency

An action may be undone only if every action caused by that action has not been executed yet or has been undone

**Restoring** the state is not enough to ensure causal consistency, we must also undo the effects of its

- spawn actions
- send actions

We solve this by "propagating the safety" to dependent processes.

## Safe session (algorithm)

safetry *expr* end

When *p* enters the safetry block...

### Safe session

1. we take a snapshot of *p* before the evaluation of *expr*
2. if another process *q* is sent a message from *p*, or is spawned by *p*, we take a snapshot of *q* (safety propagation)
3. if the evaluation of *expr* fails
   - we restore the state from *p*
   - we restore the state of processes sent a message or spawned by *p* (safety propagation)
   - go to step 3
4. we discards all the snapshots

## Implementation

Safe sessions can be implemented using:

- Monitors:
  - Intercept incoming and outgoing messages
  - Send signals between themselves to propagate the safety
- Instrumentation:
  - Enable interaction of processes with their monitors

## Program Instrumentation

$$
\begin{array}{lcl}
[\![\text{safetry } \mathit{expr} \text{ end}]\!]_M & \to & M \;!\; \langle \text{start\_session} \rangle, \\
& & [\![\mathit{expr}]\!]_M, \\
& & M \;!\; \langle \text{end\_session} \rangle \\
[\![\text{spawn}(\ldots)]\!]_M & \to & M \;!\; \langle \text{spawn}(\ldots) \rangle, \\
& & \text{receive } \langle \text{spawn\_with}, P \rangle \to P \text{ end} \\
[\![\text{self}()]\!]_M & \to & M \\
[\![\mathit{Pid} \;!\; \mathit{expr}]\!]_M & \to & M \;!\; \langle \text{send}(\mathit{Pid}), [\![\mathit{expr}]\!]_M \rangle, \\
& & \text{receive } \langle \text{sent\_as}, E \rangle \to E \text{ end} \\
[\![\text{receive } \mathit{clauses} \text{ end}]\!]_M & \to & M \;!\; \langle \text{receive}, \mathit{clauses} \rangle, \\
& & \mathit{Arg} = \text{receive} \\
& & \quad \langle \text{rec\_msg}, \mathit{Msg} \rangle \to \mathit{Msg} \text{ end}, \\
& & \text{case } \mathit{Arg} \text{ of } [\![\mathit{clauses}]\!]_M \text{ end}
\end{array}
$$

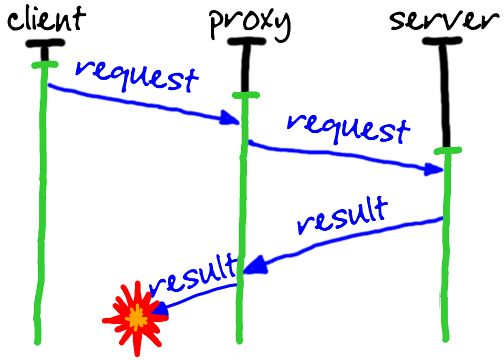
Concurrent actions are replaced by queries to the monitor.

## Program Instrumentation

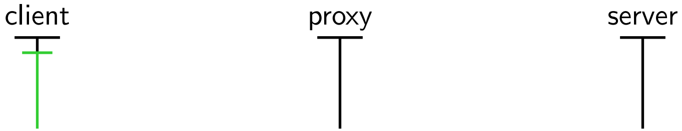
$$
\begin{array}{lcl}
[\![\text{safetry } \mathit{expr} \text{ end}]\!]_M & \rightarrow & M \; ! \; \langle \text{start\_session} \rangle, \\
 & & [\![\mathit{expr}]\!]_M, \\
 & & M \; ! \; \langle \text{end\_session} \rangle \\
[\![\text{spawn}(\ldots)]\!]_M & \rightarrow & M \; ! \; \langle \text{spawn}(\ldots) \rangle, \\
 & & \text{receive } \langle \text{spawn\_with}, P \rangle \rightarrow P \text{ end} \\
[\![\text{self}()]\!]_M & \rightarrow & M \\
[\![\mathit{Pid} \; ! \; \mathit{expr}]\!]_M & \rightarrow & M \; ! \; \langle \text{send}(\mathit{Pid}), [\![\mathit{expr}]\!]_M \rangle, \\
 & & \text{receive } \langle \text{sent\_as}, E \rangle \rightarrow E \text{ end} \\
[\![\text{receive } \mathit{clauses} \text{ end}]\!]_M & \rightarrow & M \; ! \; \langle \text{receive}, \mathit{clauses} \rangle, \\
 & & \mathit{Arg} = \text{receive} \\
 & & \quad \langle \text{rec\_msg}, \mathit{Msg} \rangle \rightarrow \mathit{Msg} \text{ end}, \\
 & & \text{case } \mathit{Arg} \text{ of } [\![\mathit{clauses}]\!]_M \text{ end}
\end{array}
$$

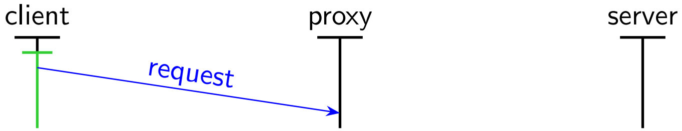
Concurrent actions are replaced by queries to the monitor.

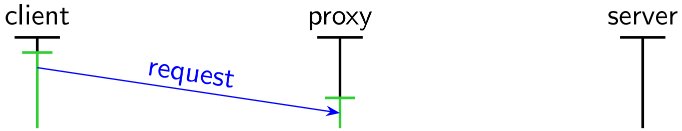
## Example with safe sessions

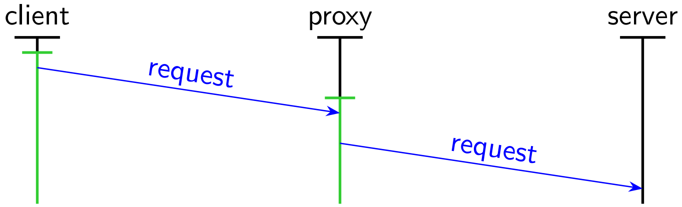


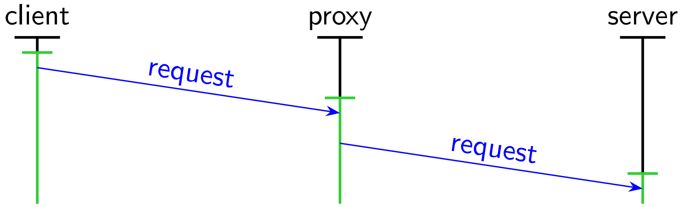
client
proxy
server

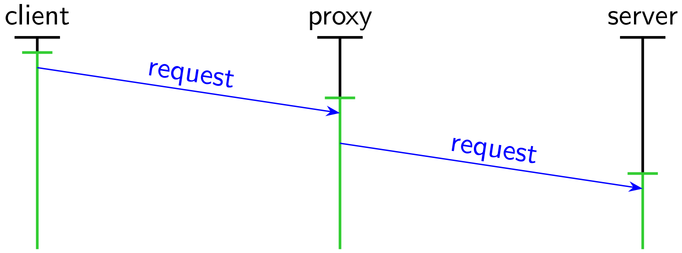

client
proxy
server

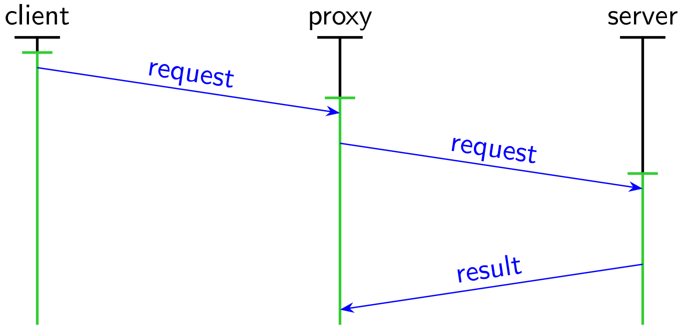

client
proxy
server
request

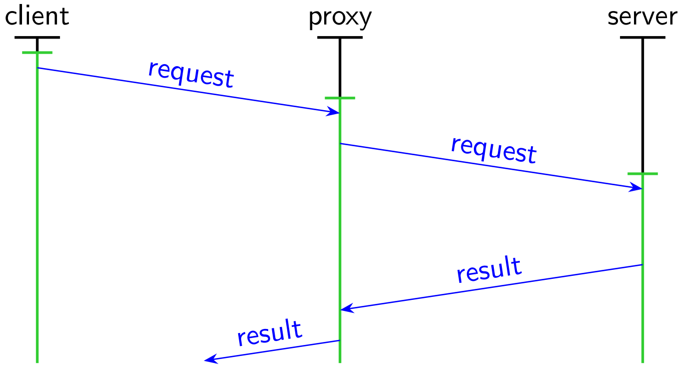

client
proxy
server
request

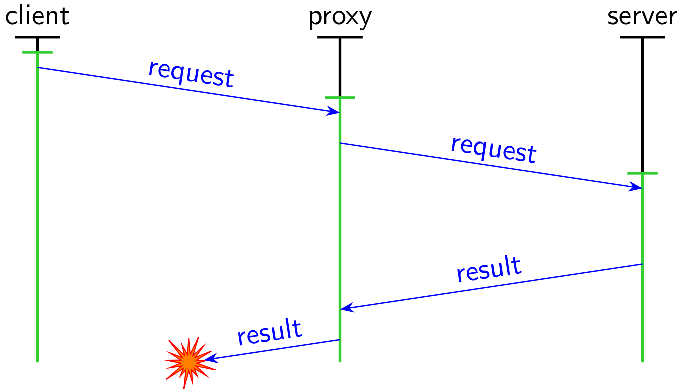

client
proxy
server
request
request

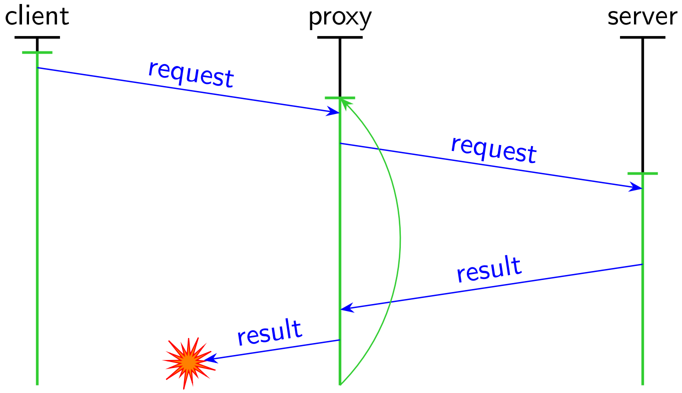

client
proxy
server
request
request

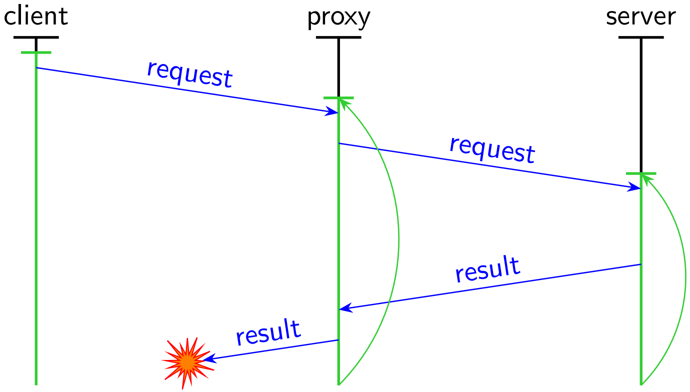

client
proxy
server
request
request

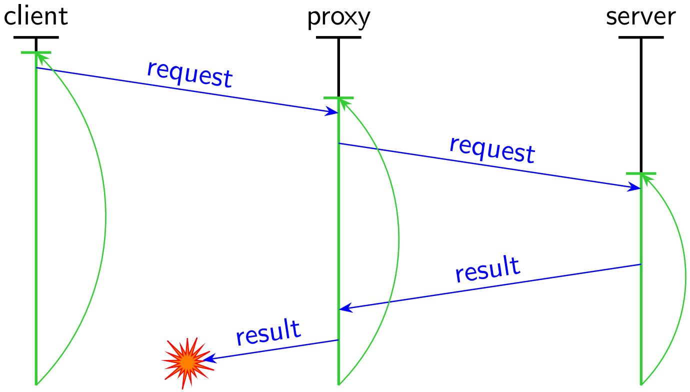

client
proxy
server
request
request
result

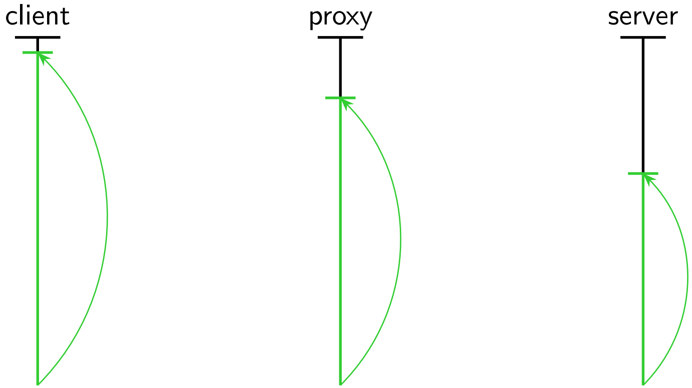

client
proxy
server
request
request
result
result

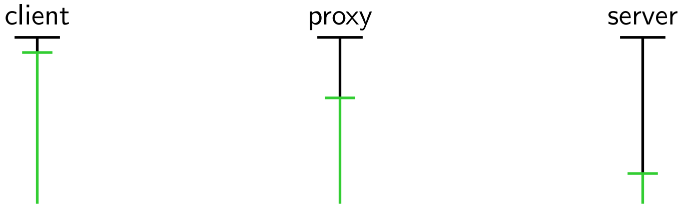

client
proxy
server
request
request
result
result

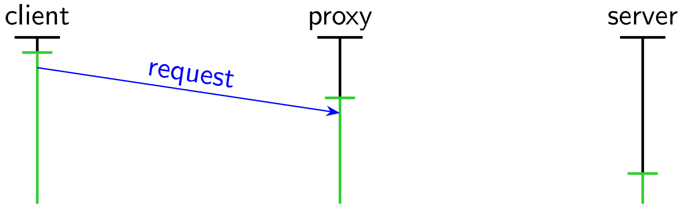

client
proxy
server
request
request
result
result

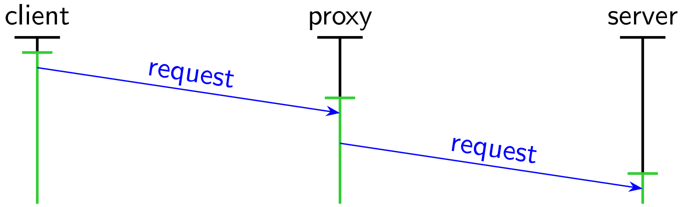

client
proxy
server
request
request
result
result

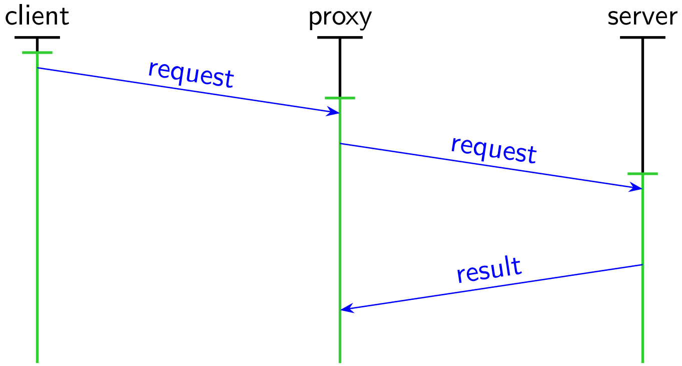

client
proxy
server
request
request
result
result

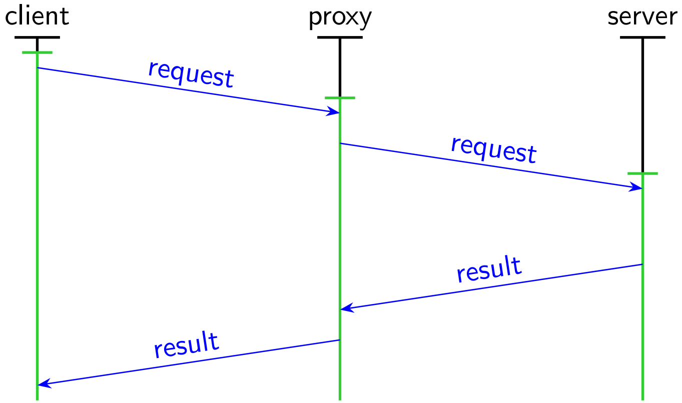

client
proxy
server

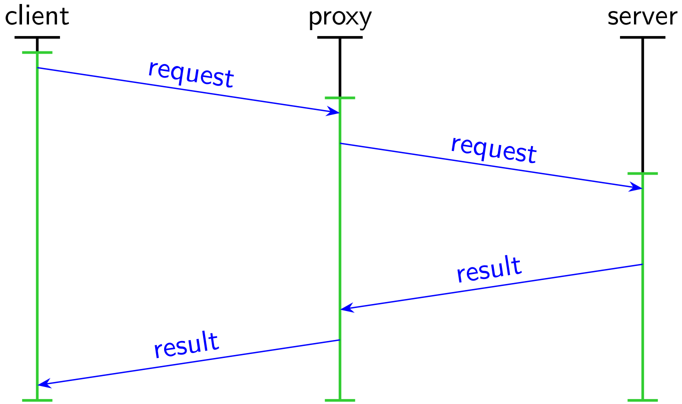

client
proxy
server

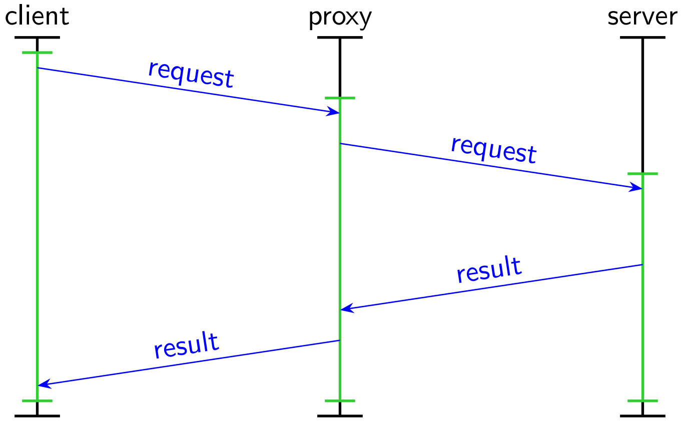

client
proxy
server
request


client
proxy
server
request
request


client
proxy
server
request
request
result


client
proxy
server
request
request
result
result


client
proxy
server
request
request
result
result


client
proxy
server
request
request
result
result

## Conclusions

## Conclusions

Some related work:

- **Field and Varela [POPL'05]:**
  - checkpoint-based approach with some similarities
  - they aim at defining a new language, rather than extending one
- **Neykova and Yoshida [CC'17]**
  - interprocedural recovery strategy based on session types
  - not so fine-grained, we could define an intraprocedural strategy

In the future, we will:

- refine our design of safe sessions
- develop an implementation
- compare our implementation against other approaches

Thanks for your attention!