Erlang - A monitoring-oriented programming language

Christian Colombo

joint work with
Adrian Francalanza, Rudolph Gatt, Kevin Falzon,
Andrew Gauci, Ruth Mizzi

University of Malta

Runtime Monitoring

Ch
ru

Execu
and

State space explosion

Checks at runtime

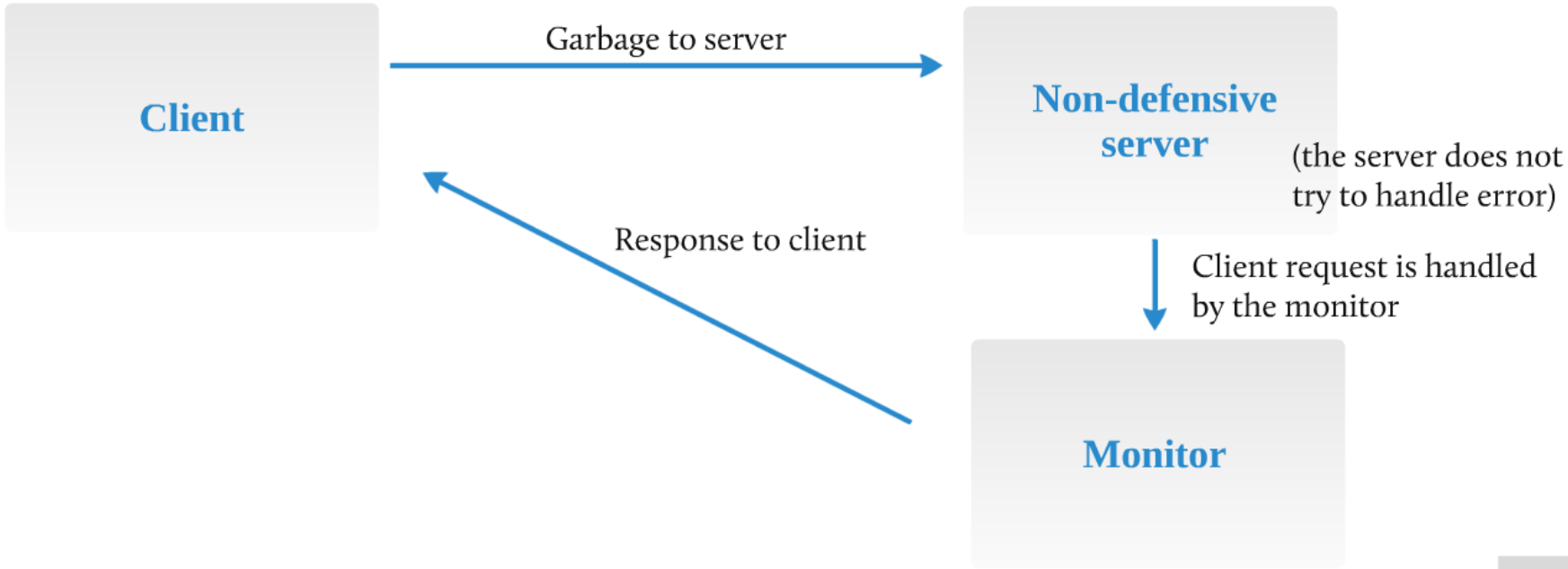
Runtime Monitoring

Links monitor to system

Executes system and monitor concurrently

Separates logic from exception handling

Non-defensive programming



State space explosion

Checks at runtime

Runtime Monitoring

Links monitor to system

Executes system and monitor concurrently

Separates logic from exception handling

r at

e

ncy is

Erlang

State space explosion

Checks at runtime

Runtime Monitoring

Links monitor to system

Executes system and monitor concurrently

Separates logic from exception handling

Concurrency worsens explosion problem

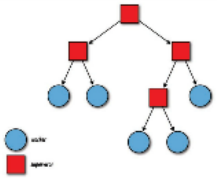
Type checking must occur at runtime

Erlang Systems

Monitor/Link

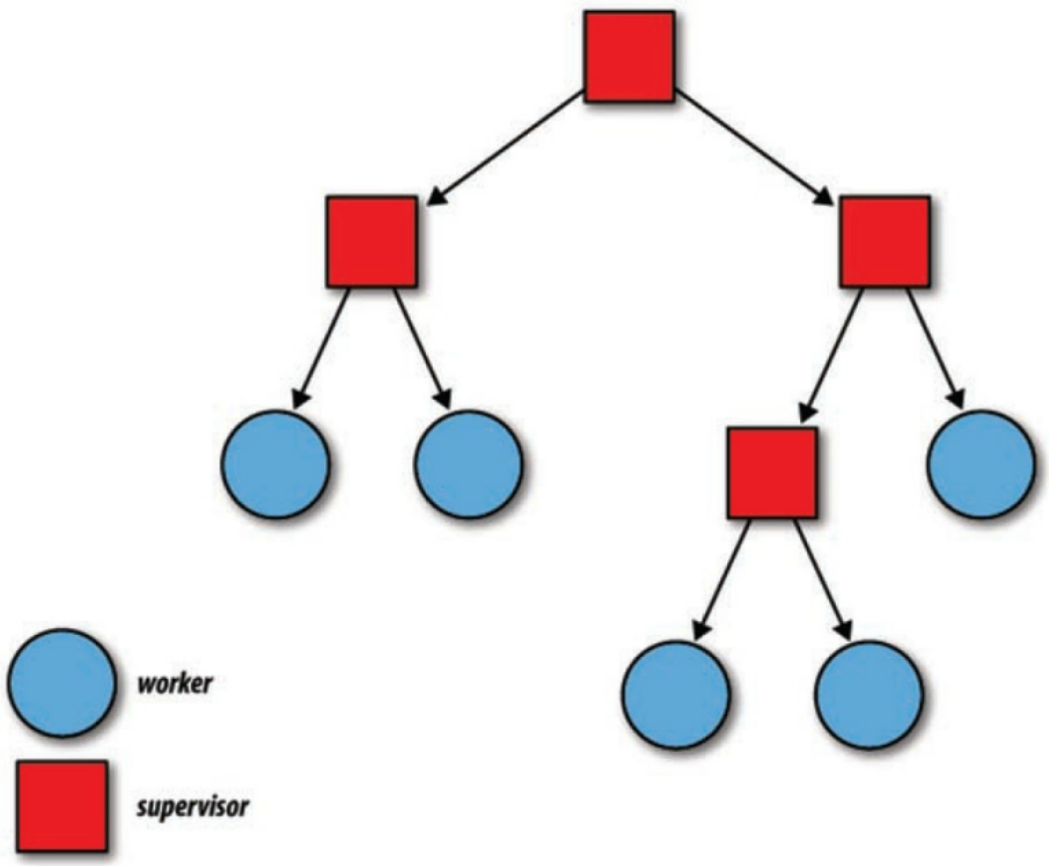
Concurrency is natural in Erlang

Supervision tree



The Elevator Example

Supervision tree



State space explosion

Checks at runtime

Runtime Monitoring

Links monitor to system

Executes system and monitor concurrently

Separates logic from exception handling

Concurrency worsens explosion problem

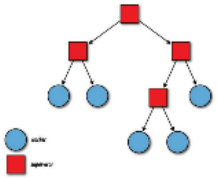
Type checking must occur at runtime

Erlang Systems

Monitor/Link

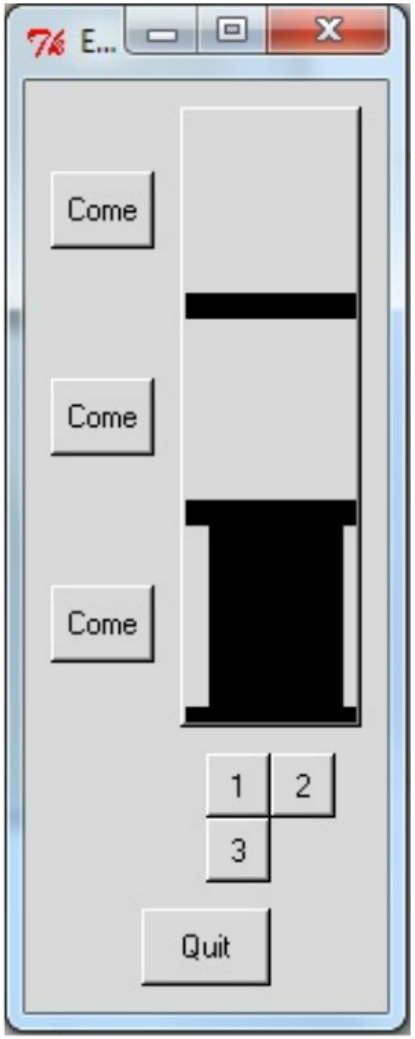
Concurrency is natural in Erlang

Supervision tree



The Elevator Example

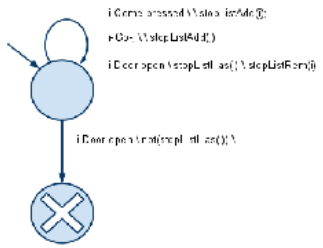
The Elevator Example

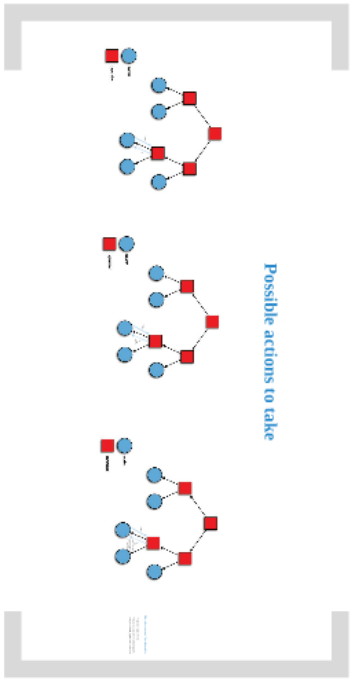


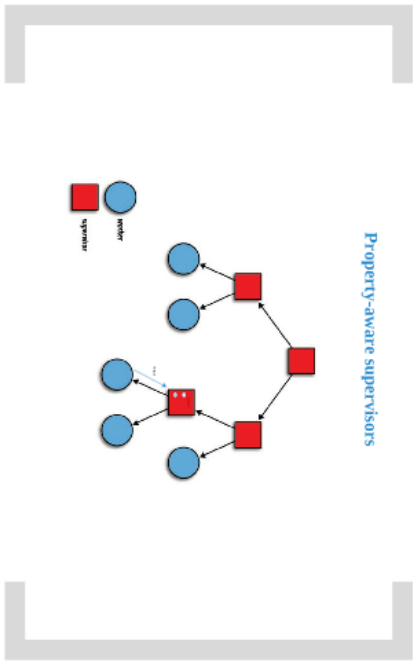
The door opens ONLY if requested!

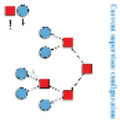
Trace of the elevator execution

Level	Action
1	Come pressed
1	Door opened
1	Go 2 pressed
1	Door closed
3	Come pressed
2	Door opened
2	Go 3 pressed
2	Door closed
3	Door opened
...	









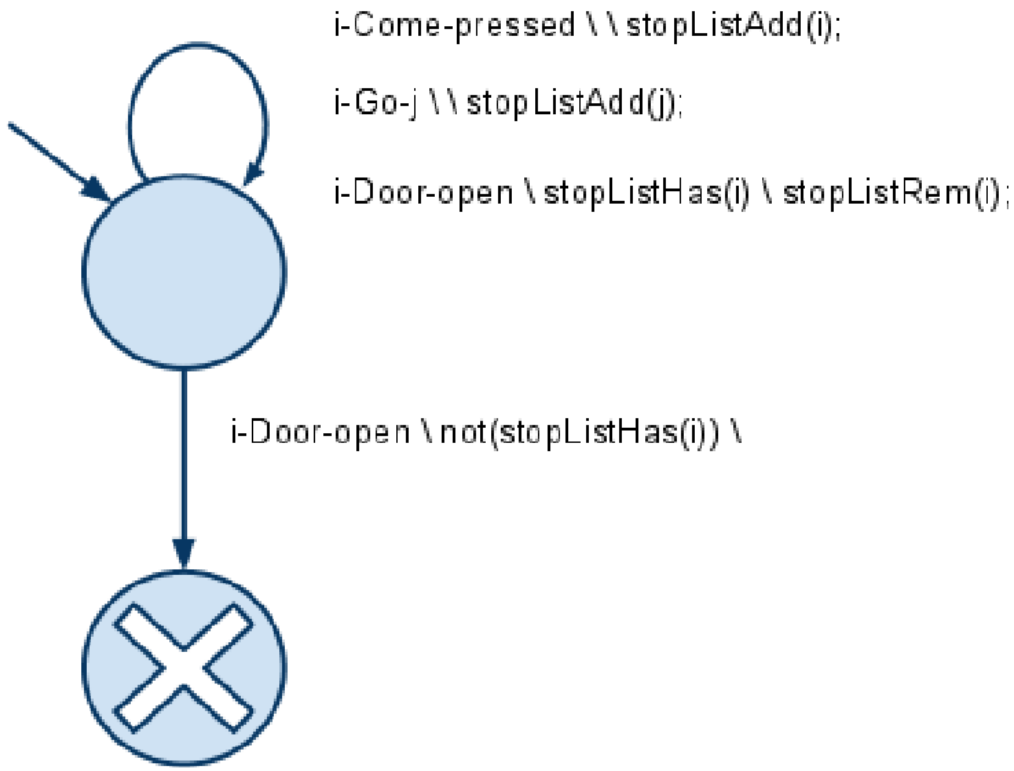
Trace of the elevator execution

Level	Action
1	Come pressed
1	Door opened
1	Go 2 pressed
1	Door closed
3	Come pressed
2	Door opened
2	Go 3 pressed
2	Door closed
3	Door opened
. . .	

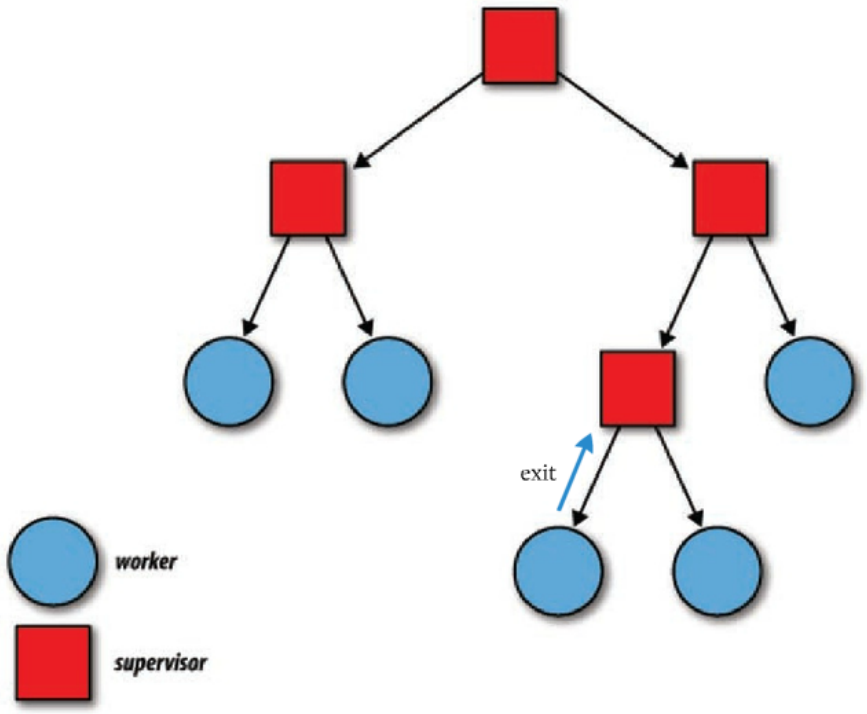
The door opens ONLY if requested!

Trace of the elevator execution

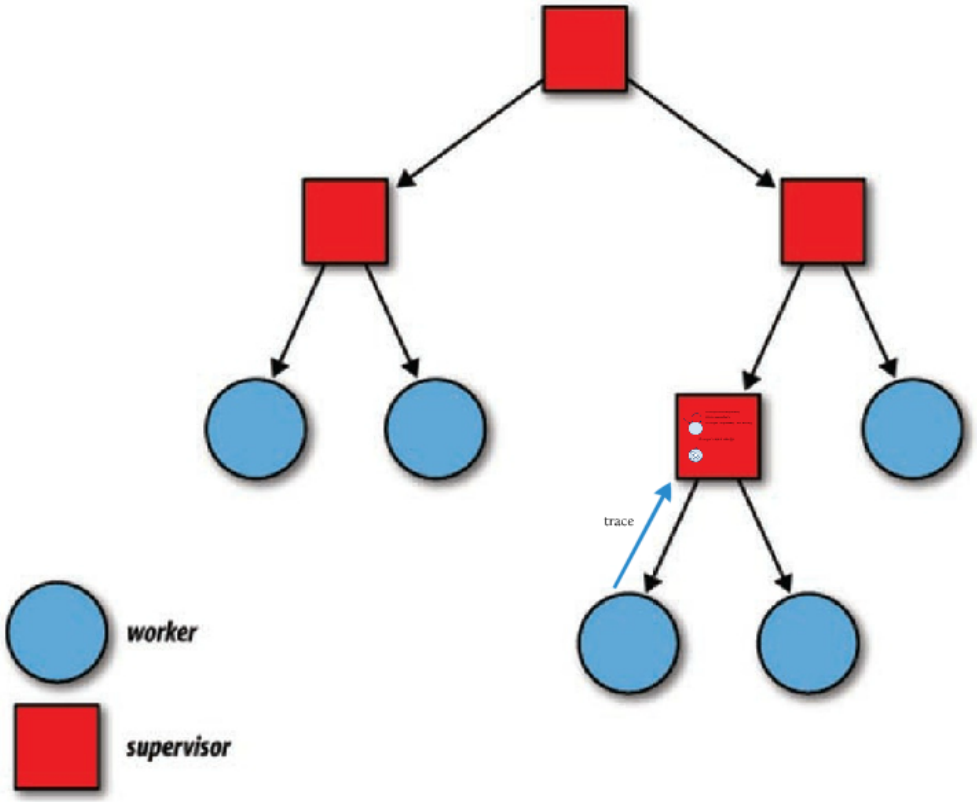
Level	Action
1	Come pressed
1	Door opened
1	Go 2 pressed
1	Door closed
3	Come pressed
2	Door opened
2	Go 3 pressed
2	Door closed
3	Door opened
...	

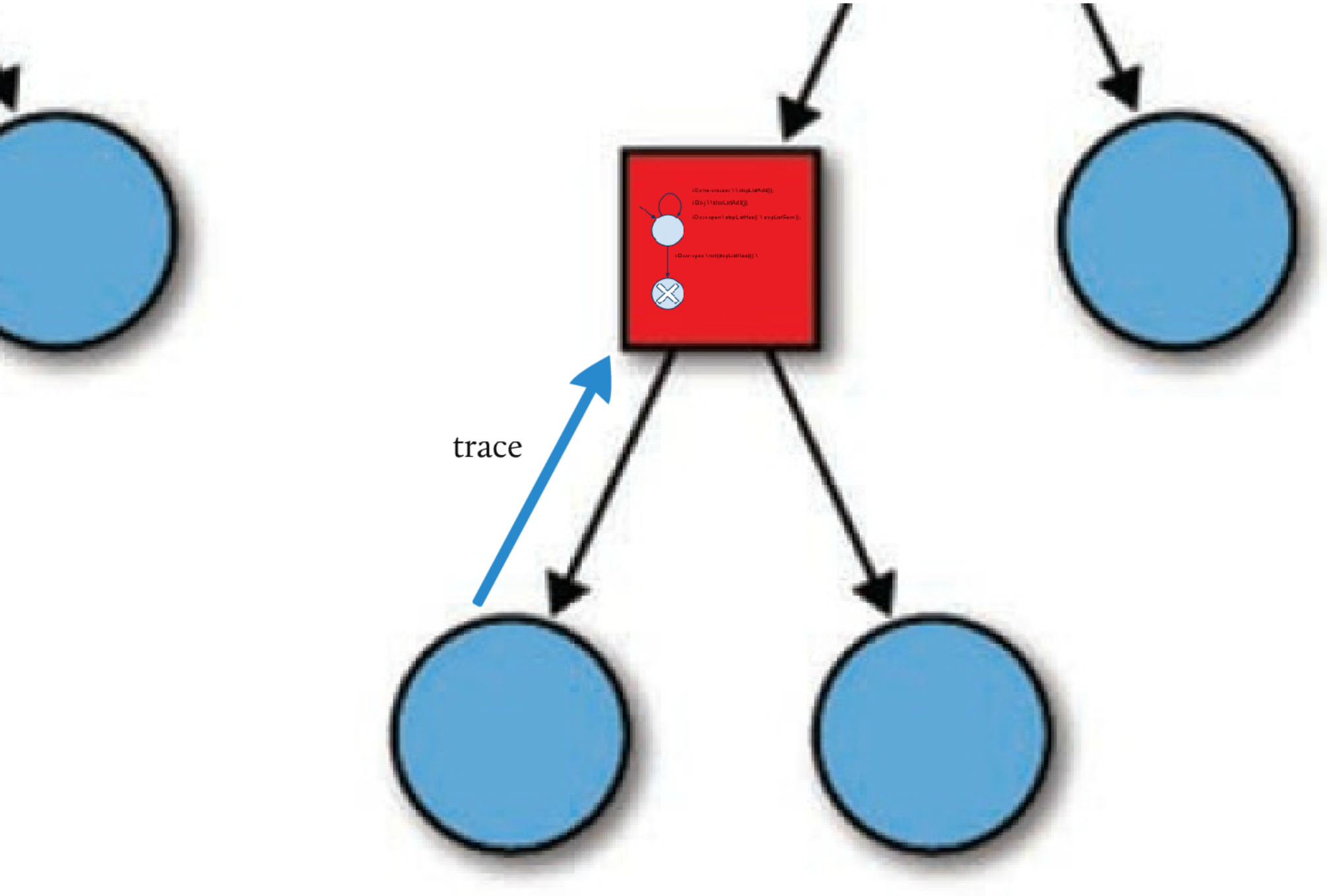


Current supervisor configuration

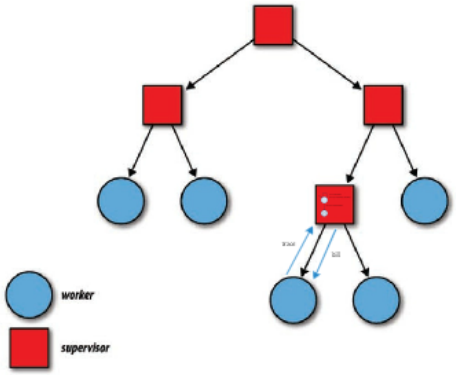


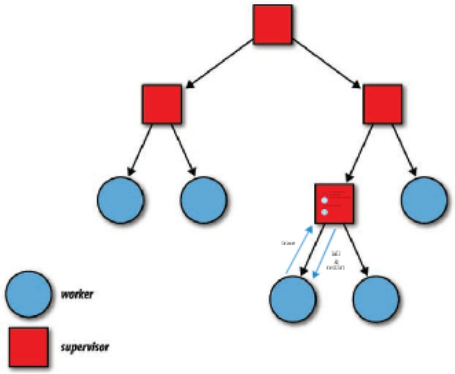
Property-aware supervisors

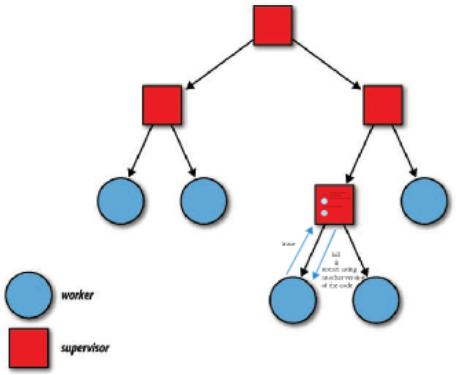



trace

Possible actions to take

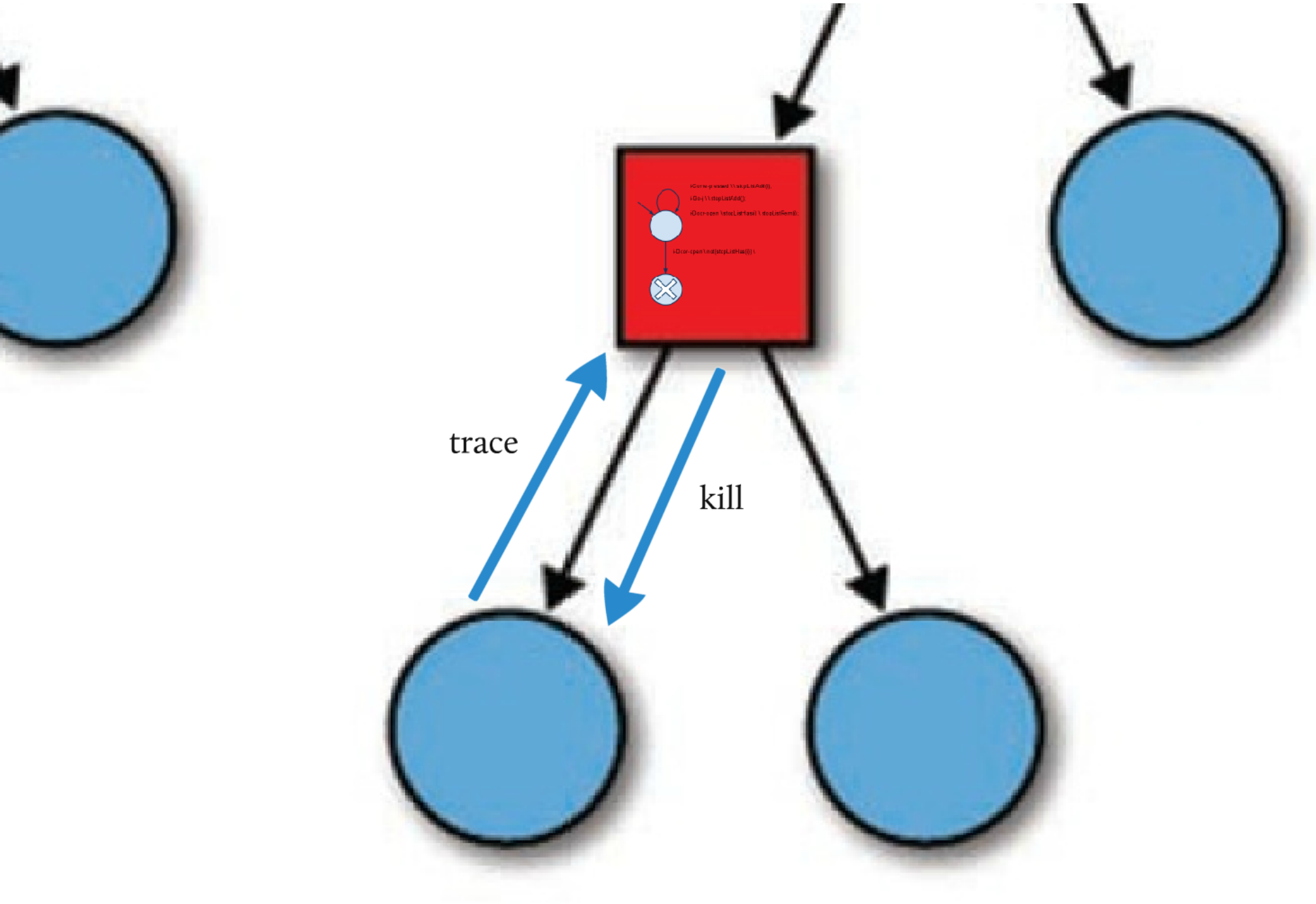


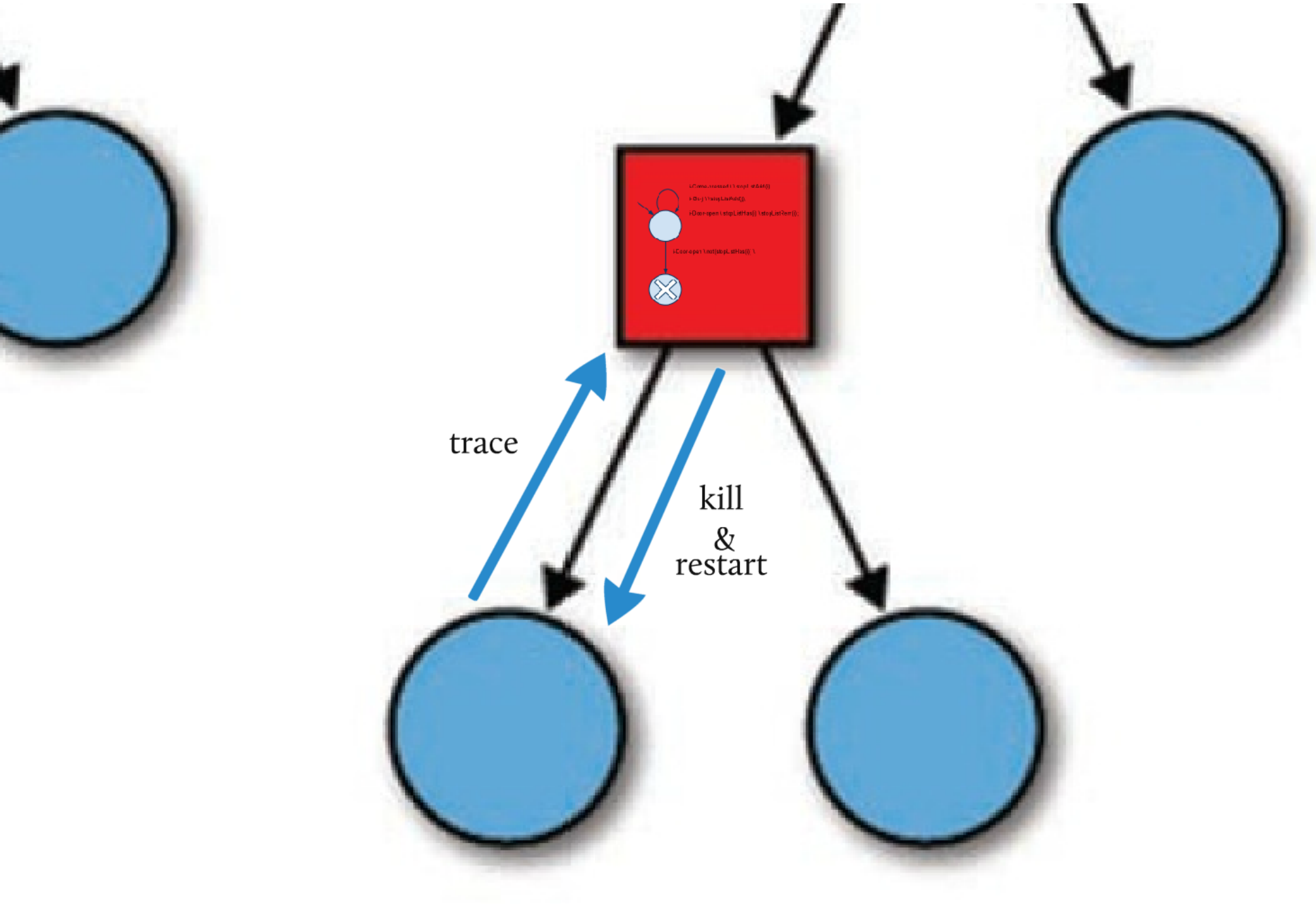


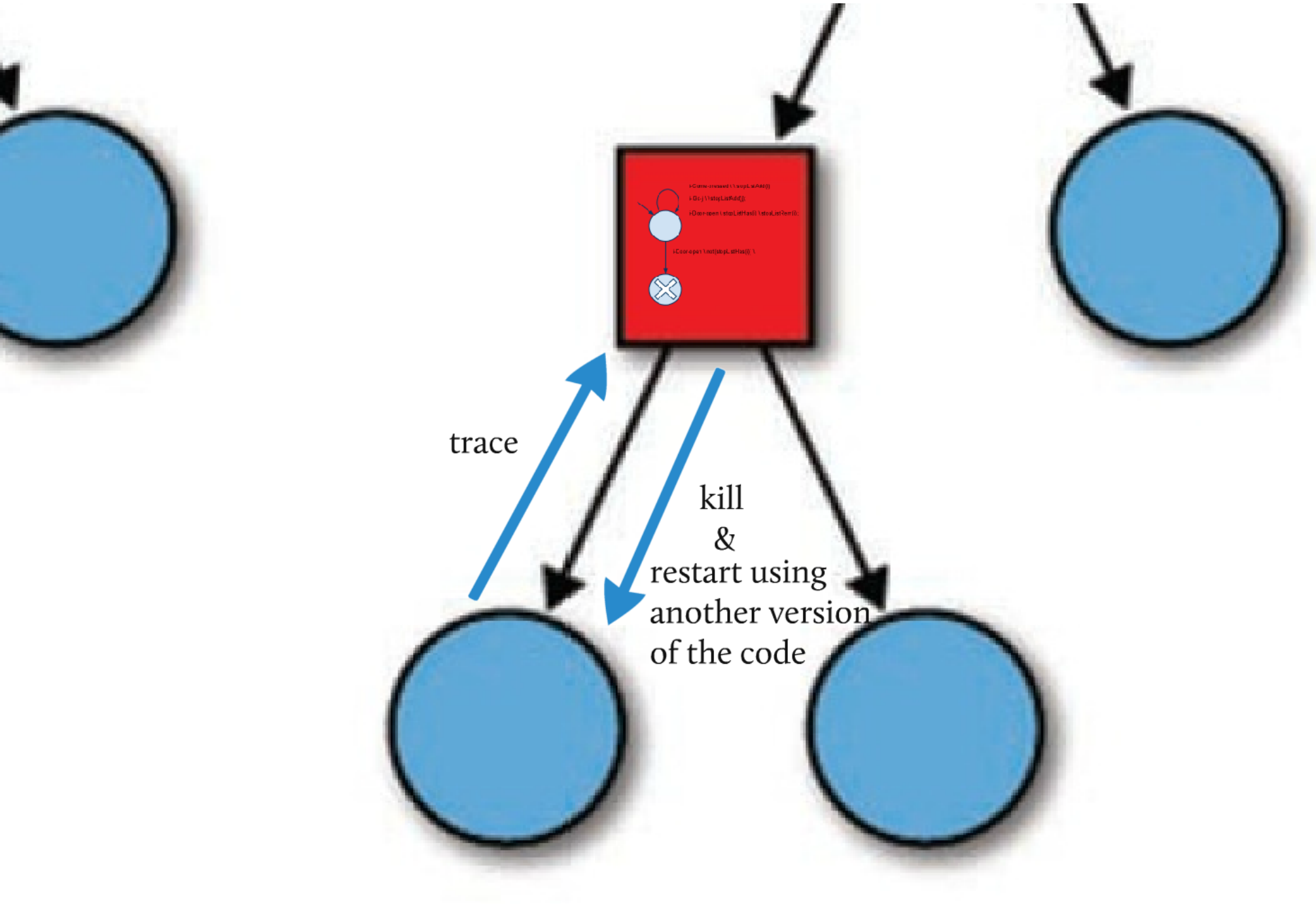


What other versions of the code may be

- "Safer code" version of the code
- The previous version of the code (when changed)
- The previous stable/working version of the system


trace
kill


trace
kill
&
restart


trace
kill
&
restart using
another version
of the code

What other versions of the code may be

- "Safe mode" version of the code
- The previous version of the code (before last patch)
- The previous (stable) implementation of the system

Summary of our idea

Existing Erlang linking mechanism is just a special case...

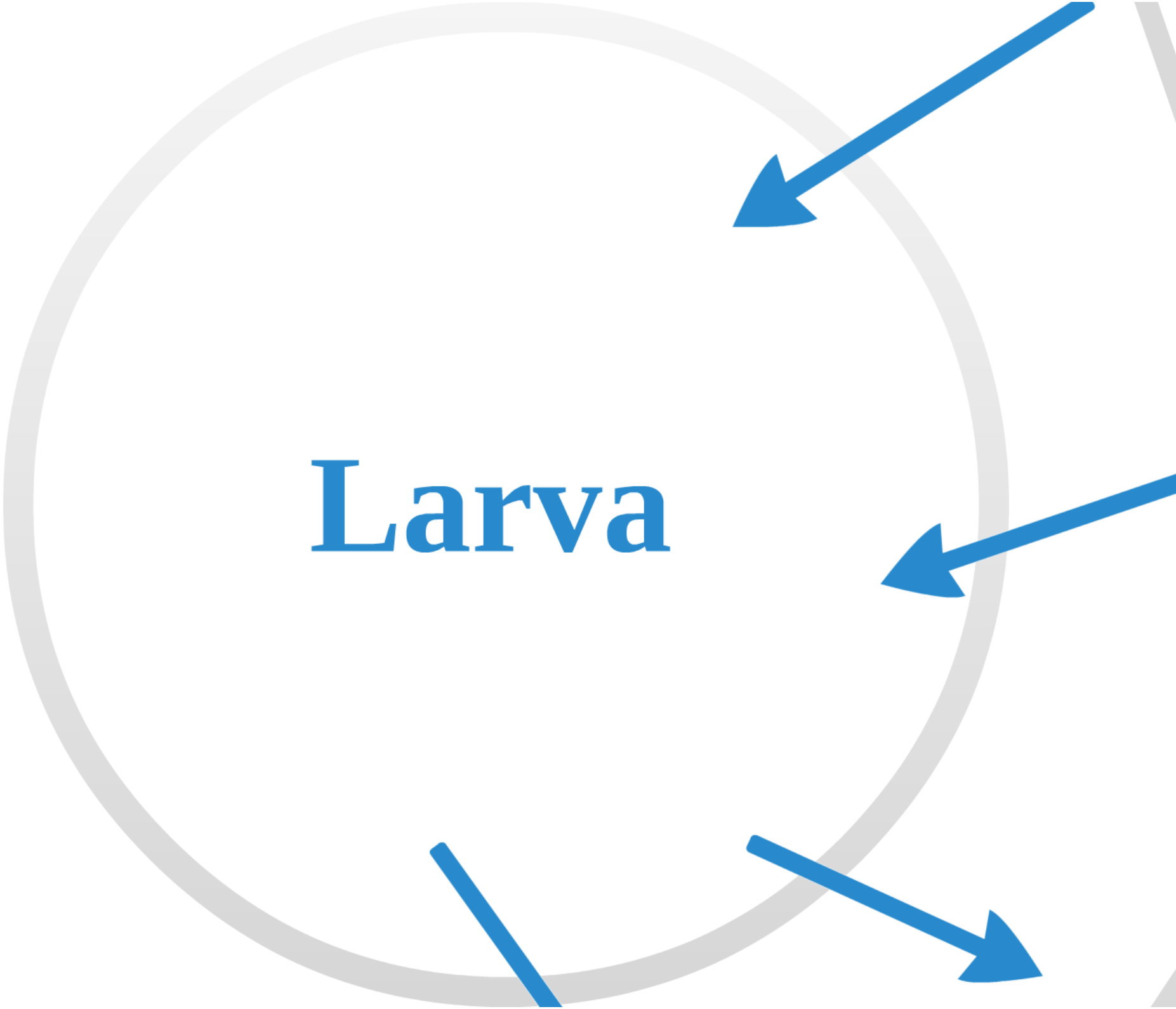
Erlang	Our Proposal
the supervisor is only notified of abnormal termination	the supervisor is notified of all relevant events, checking behaviour against properties

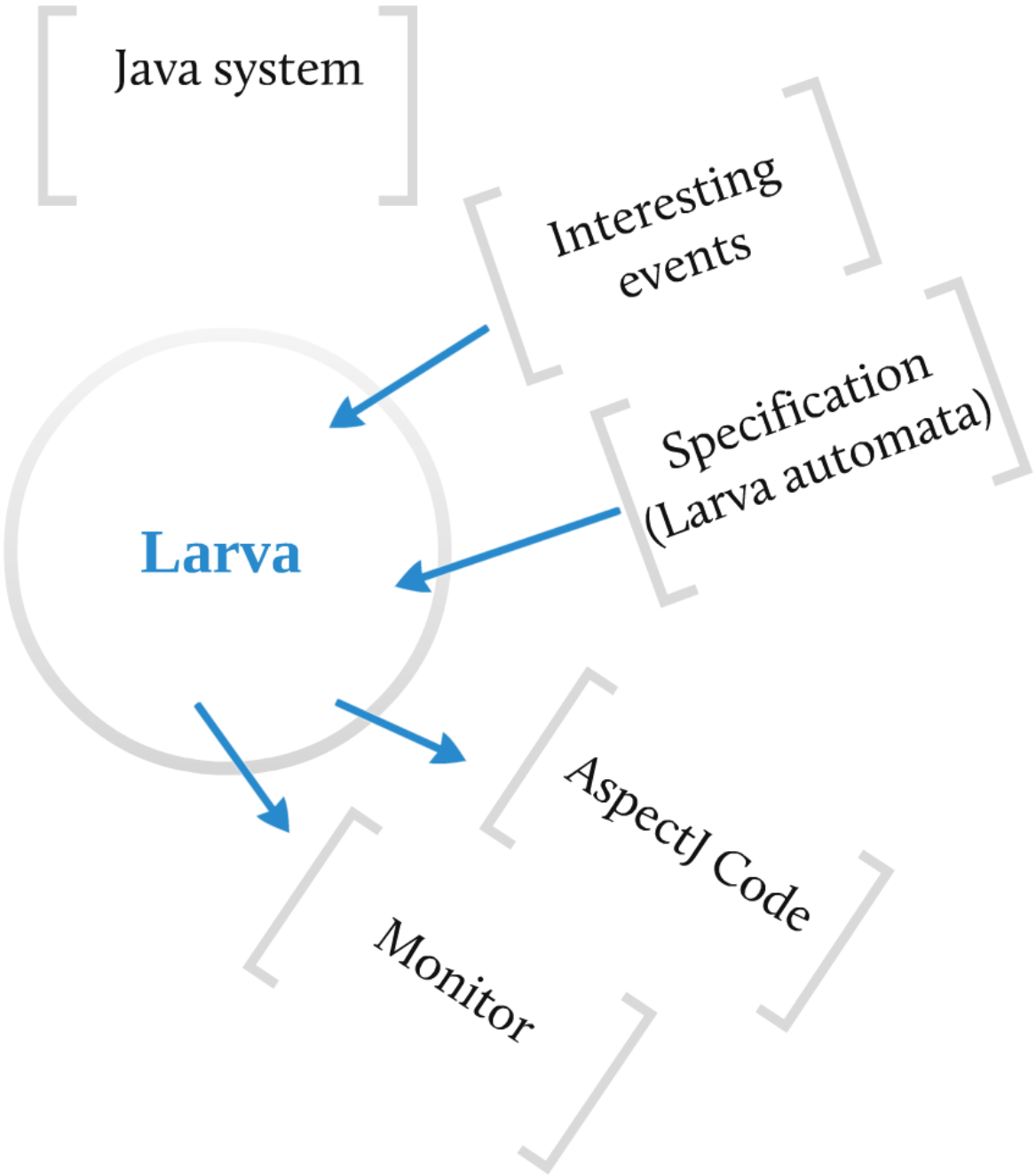
Not all errors lead to abnormal termination...
so force offending processes to fail-fast!

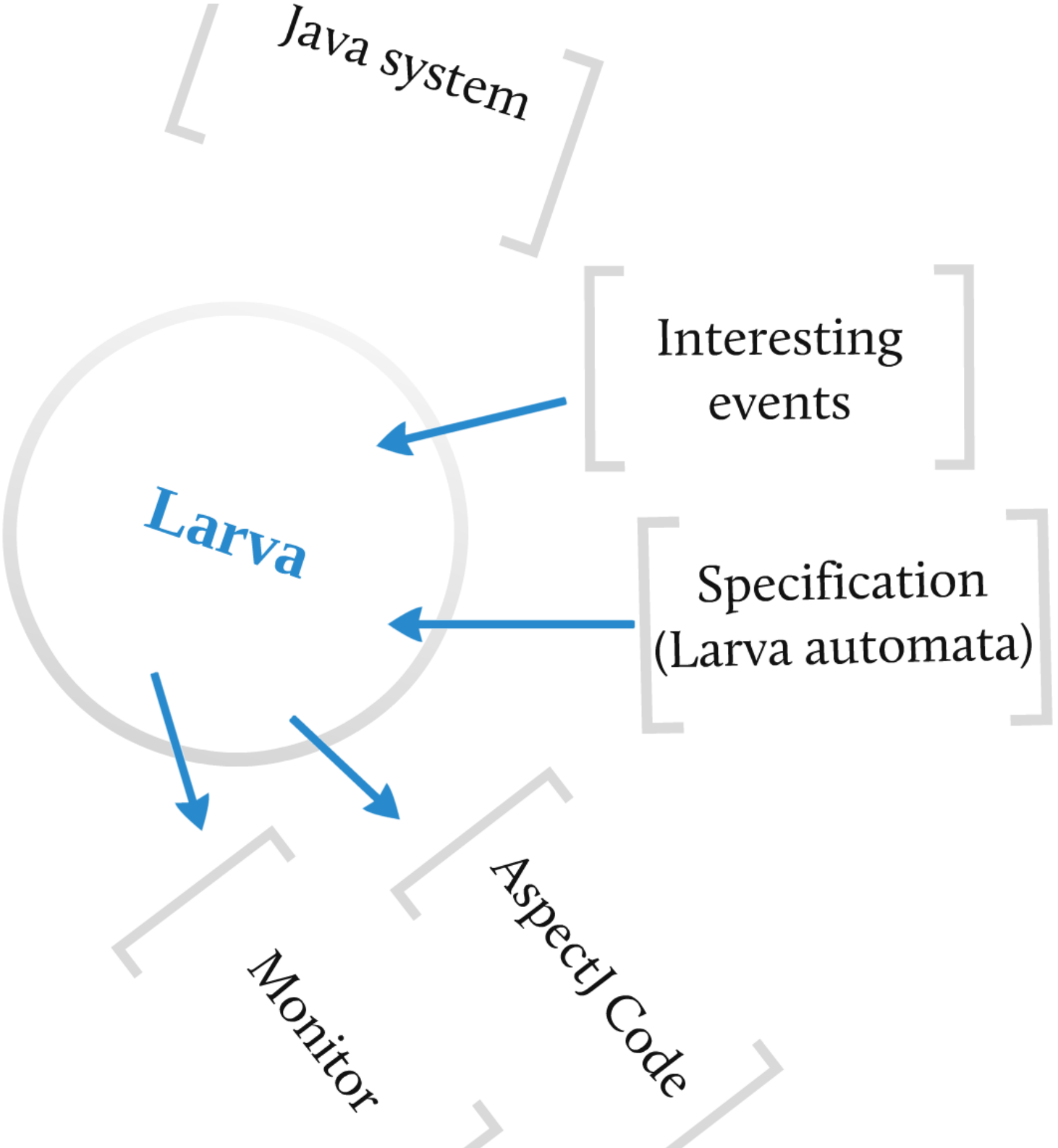
What tools are available for monitoring?

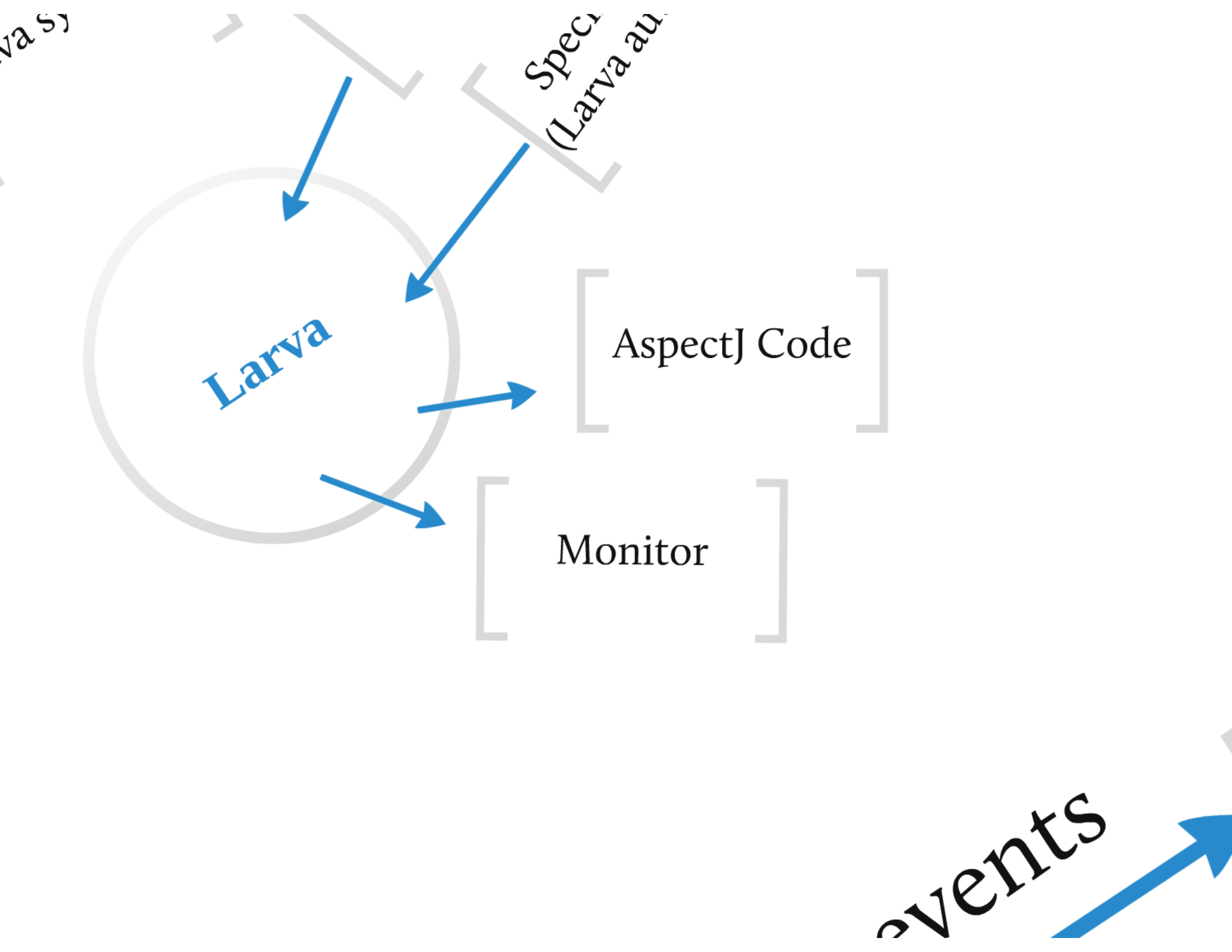
There is a tool for Java systems called Larva...

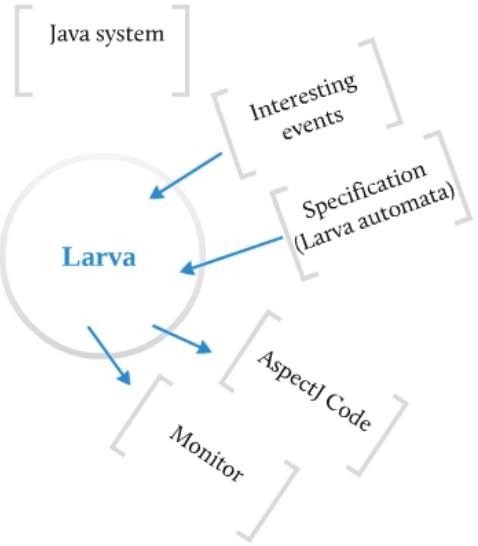
...can it be adapted for Erlang? ... ELarva!

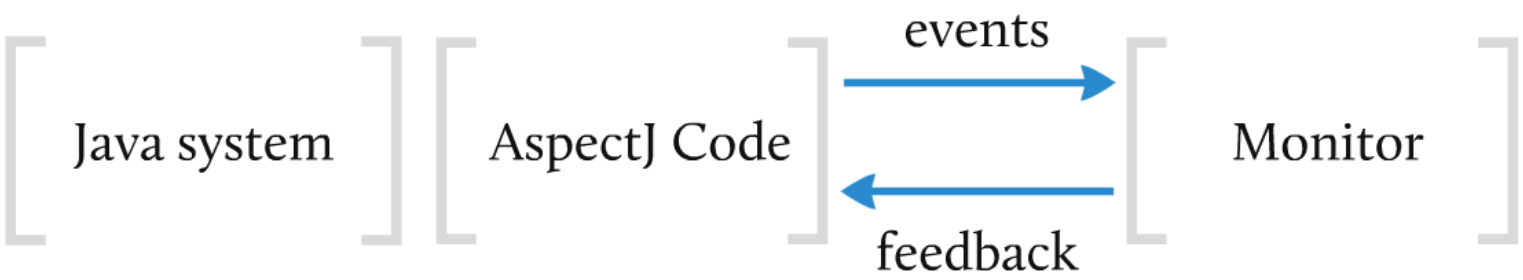

Larva

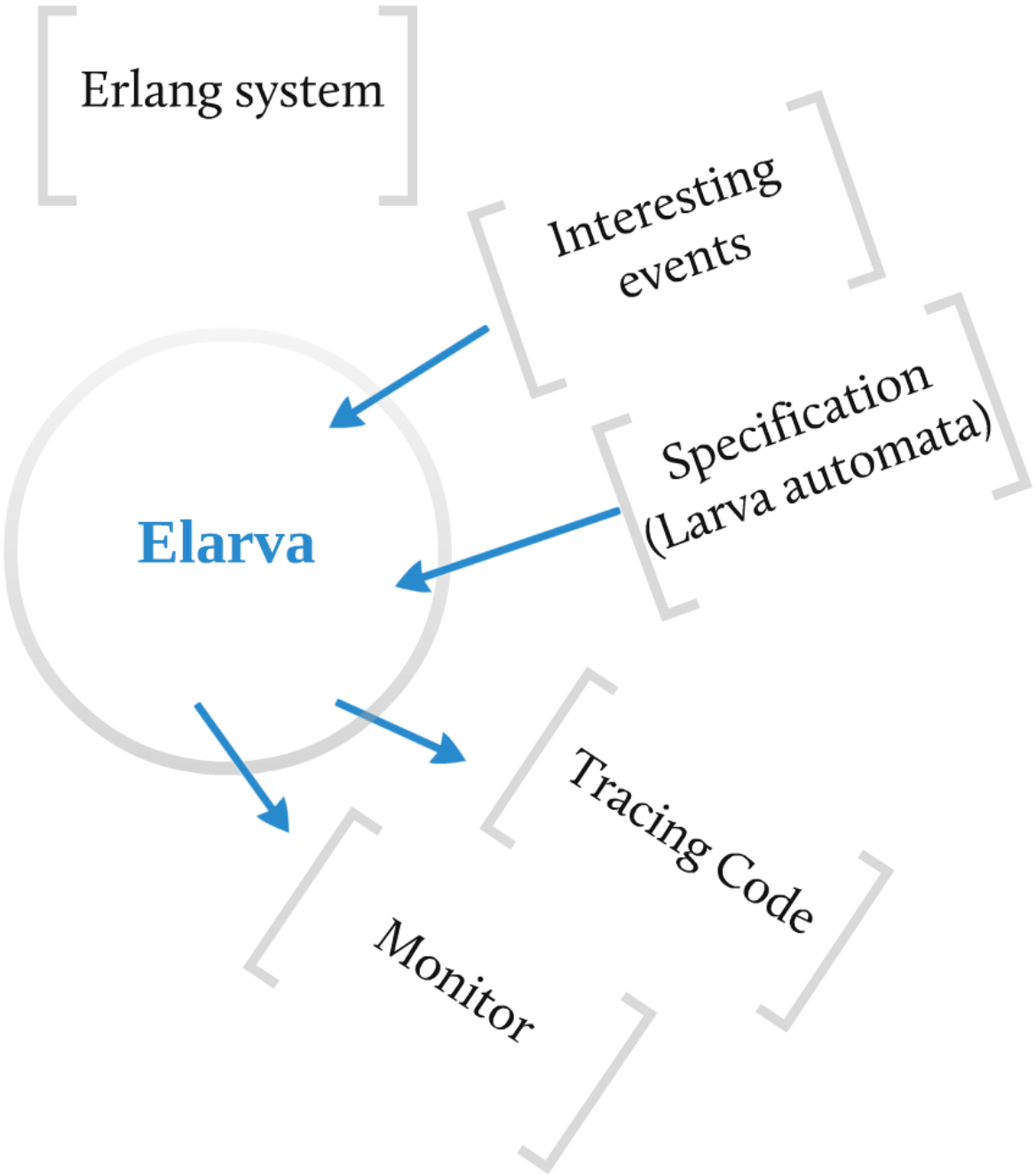

Java system
Interesting events
Specification (Larva automata)
Larva
AspectJ Code
Monitor

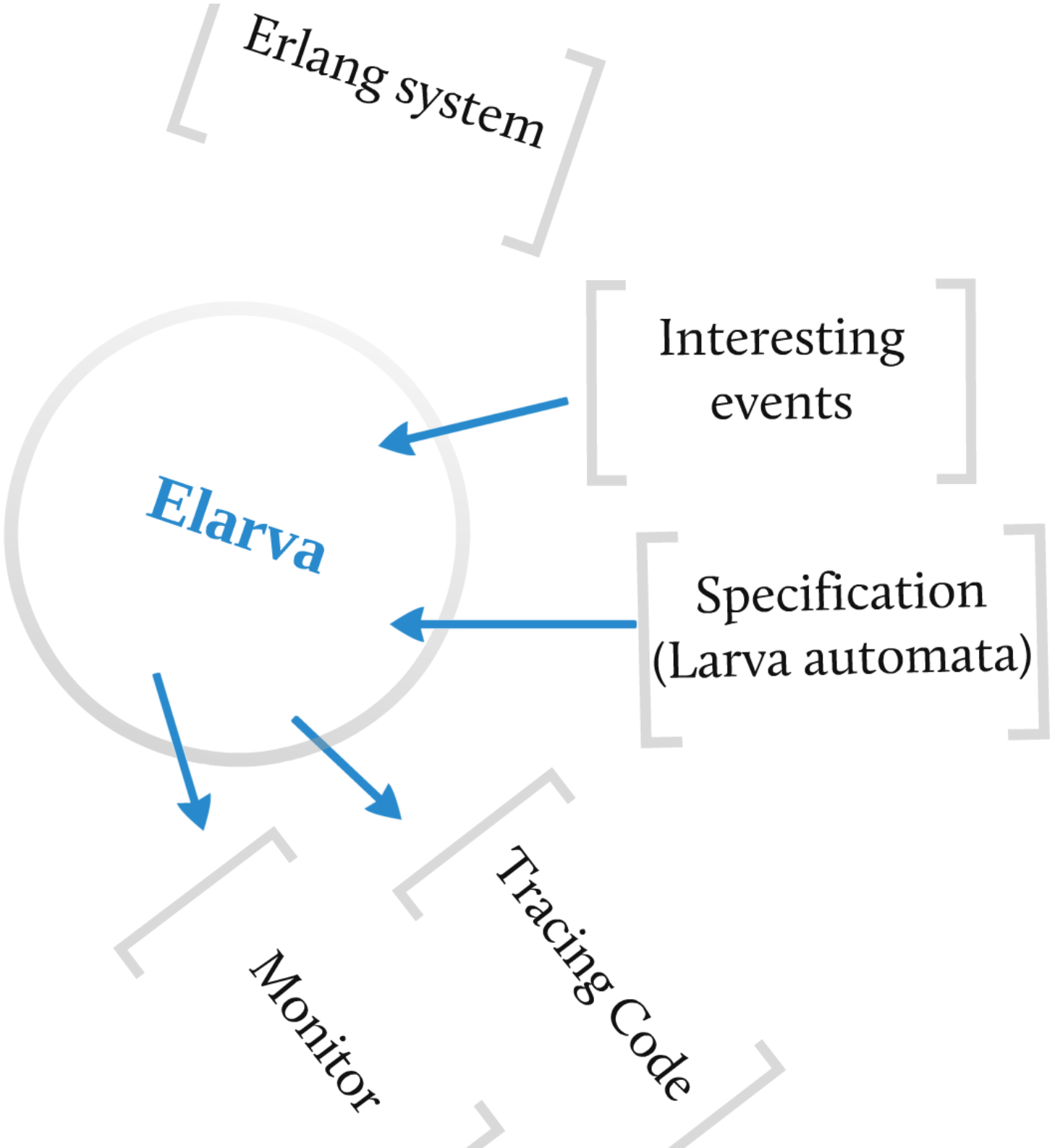

Java system
Interesting events
Larva
Specification (Larva automata)
Monitor
AspectJ Code

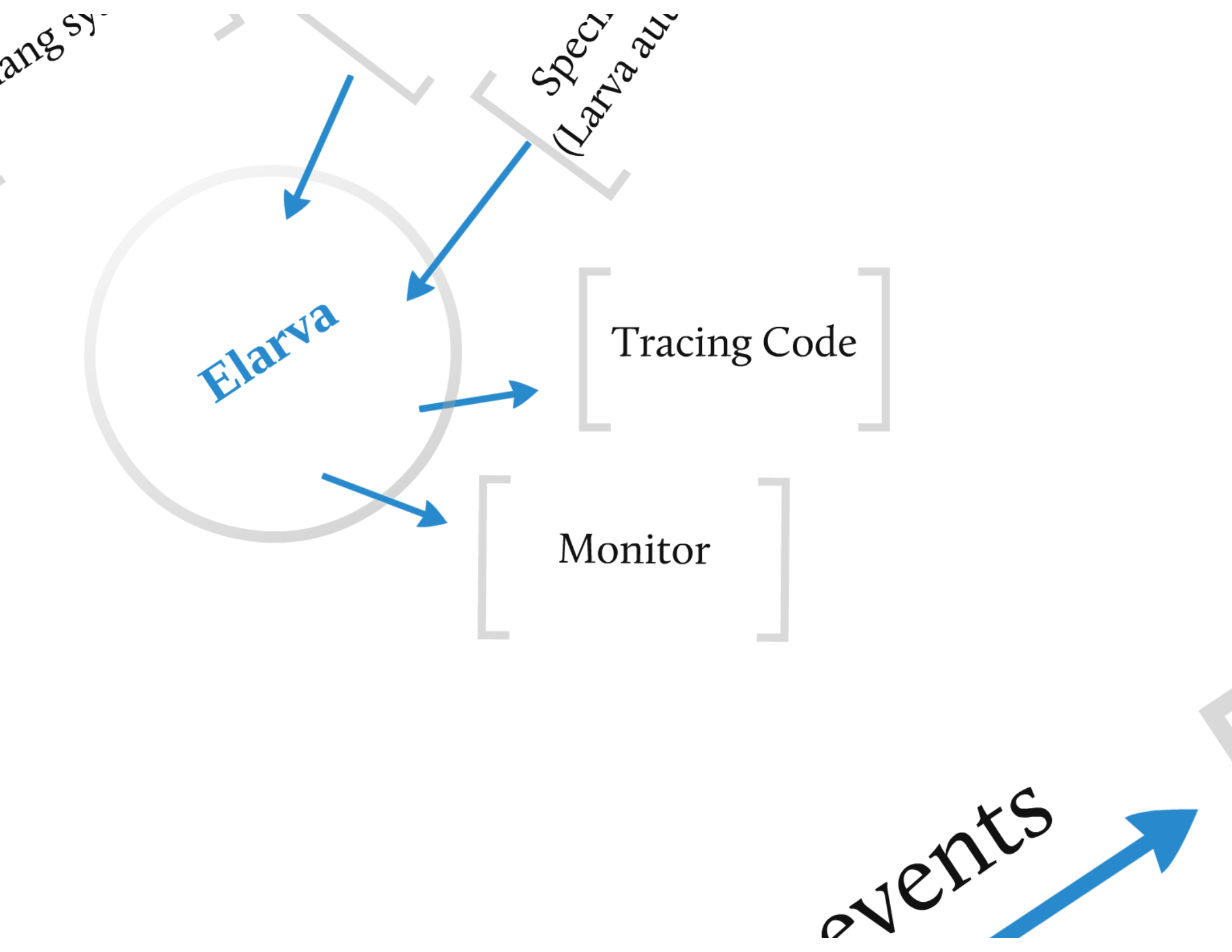

Larva
AspectJ Code
Monitor
events

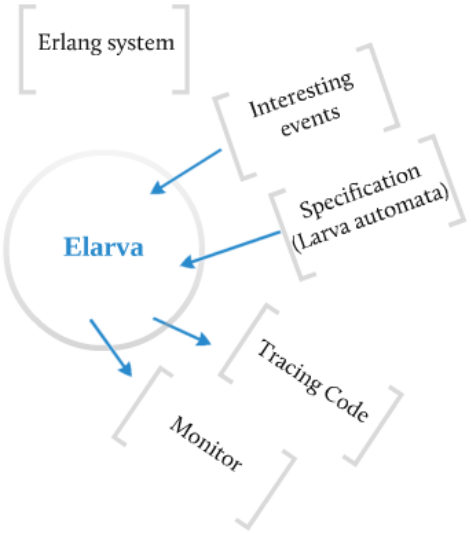

Java system
Interesting events
Specification (Larva automata)
Larva
AspectJ Code
Monitor

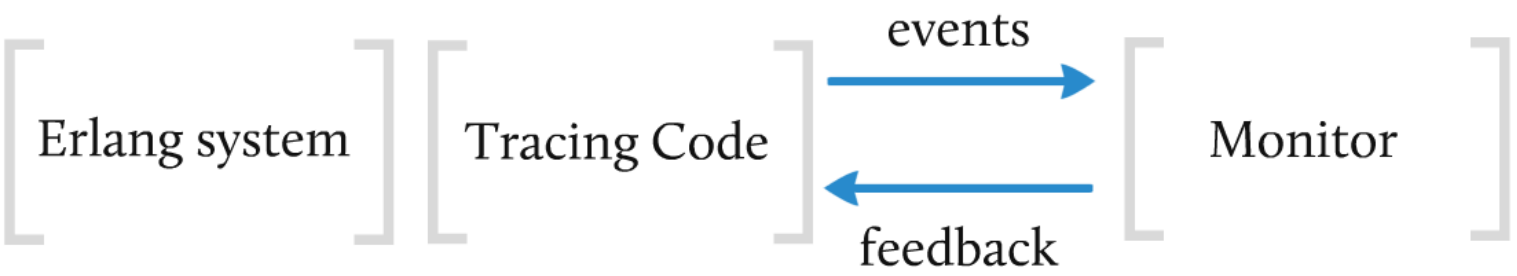

Java system
AspectJ Code
events
feedback
Monitor


Erlang system
Interesting events
Specification (Larva automata)
Elarva
Tracing Code
Monitor


Erlang system
Interesting events
Elarva
Specification (Larva automata)
Monitor
Tracing Code


Elarva
Tracing Code
Monitor
events


Erlang system
Interesting events
Specification (Larva automata)
Elarva
Tracing Code
Monitor


Erlang system
Tracing Code
events
feedback
Monitor

"I have to write formal properties!"

True!... but you would probably already have them!

Don't throw away your tests!

QuickCheck

state machines

QuickCheck automata

Larva

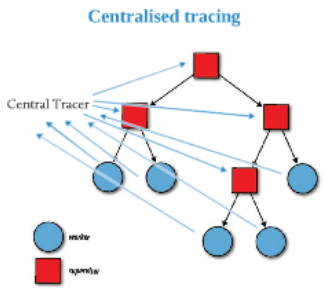
Larva automata

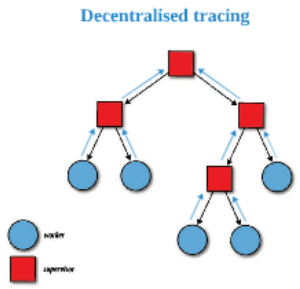
WORK IN PROGRESS

Centralisation

Tracing is centralised

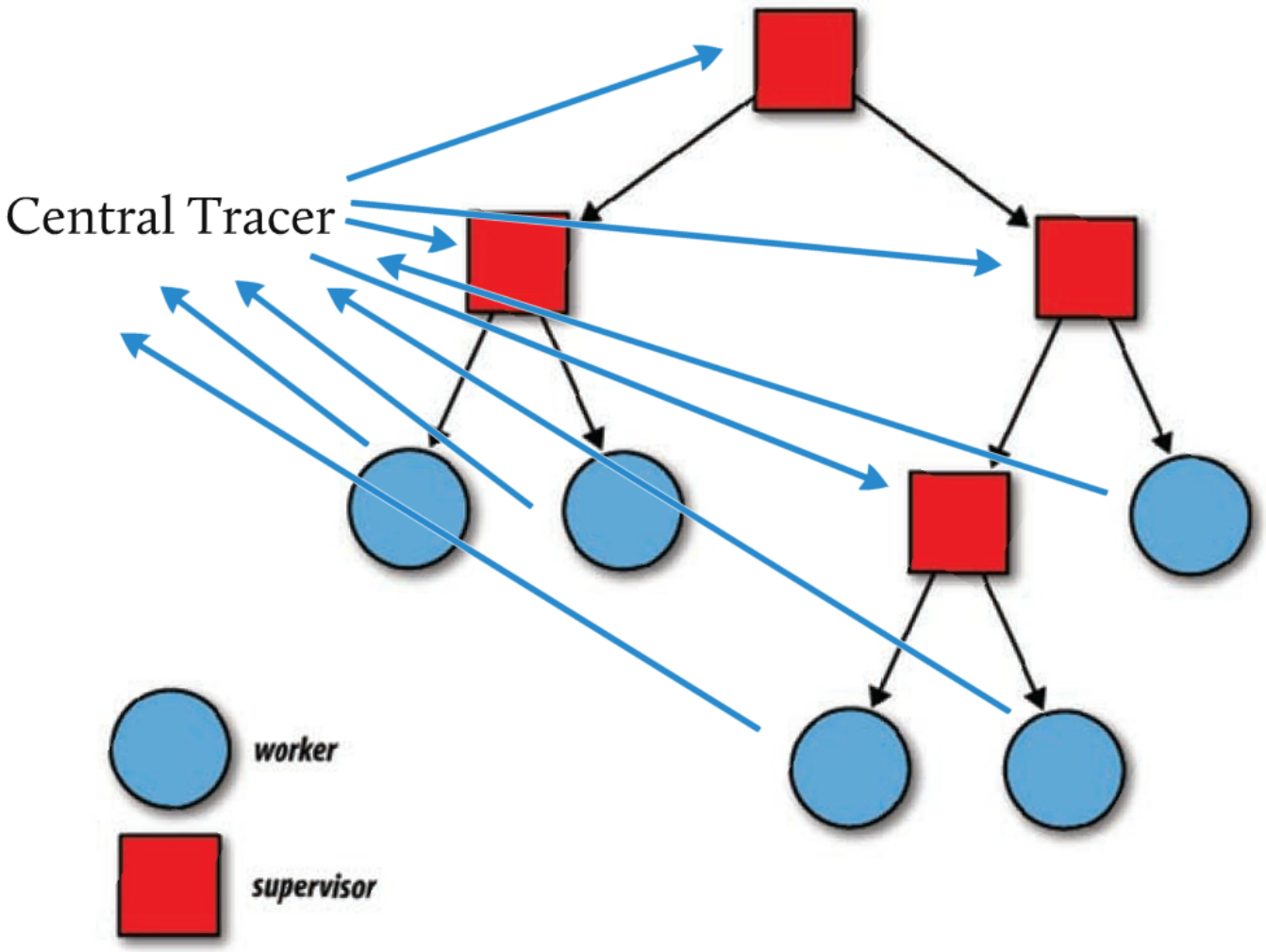
... we have a bottleneck of incoming events



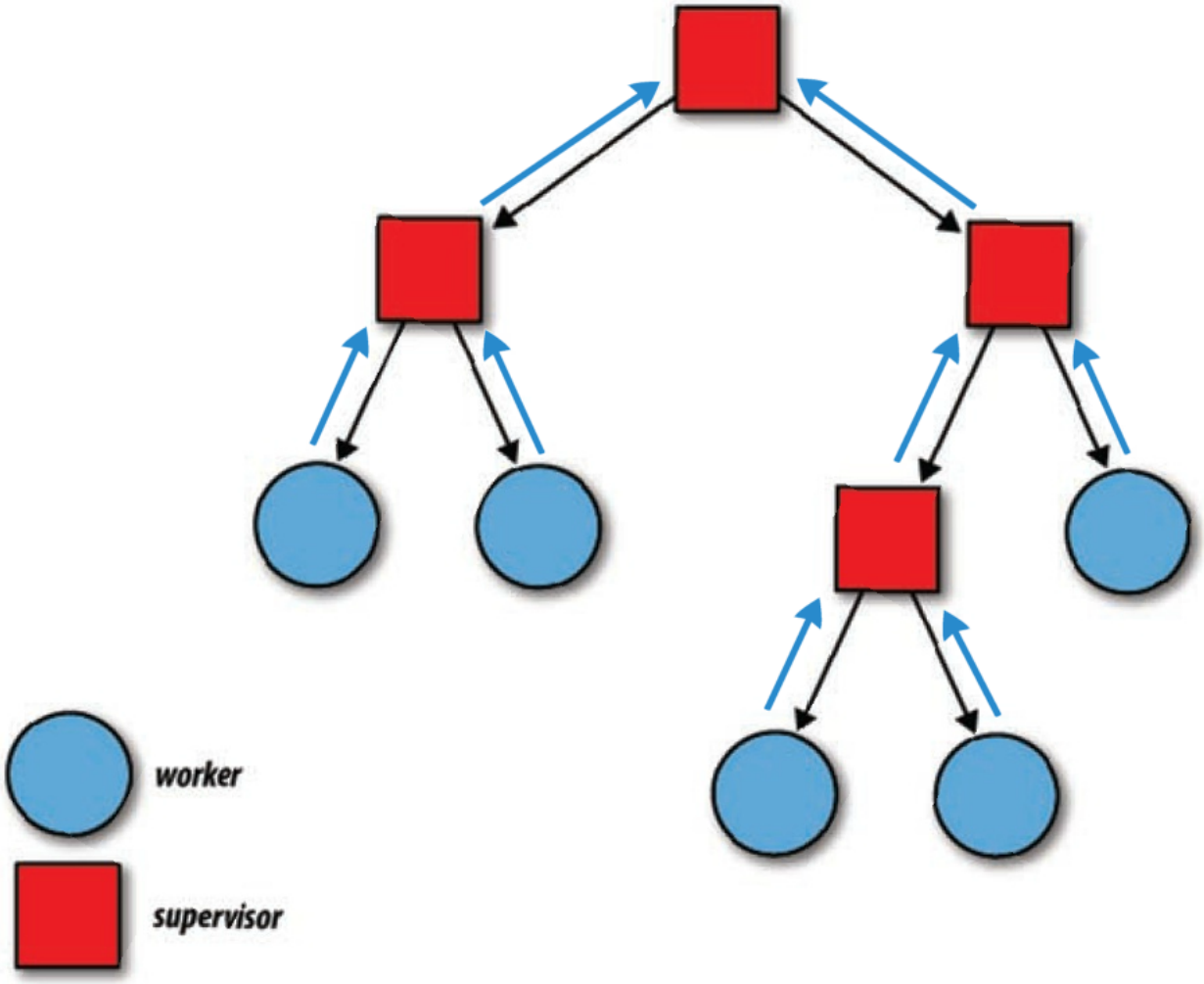


We wish to have decentralised tracing...

Centralised tracing

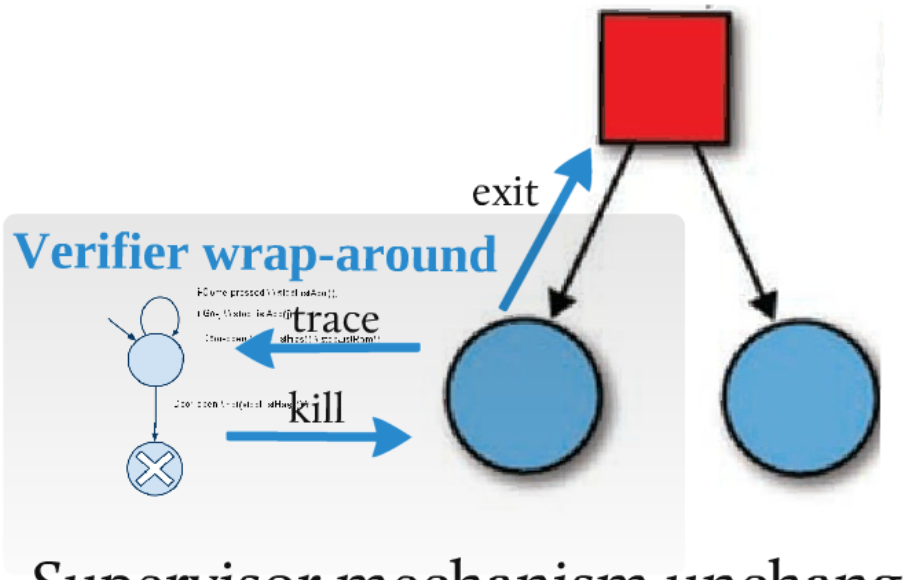


Decentralised tracing



A Cleaner Approach

Wrapping the worker inside a verifier



Supervisor mechanism unchanged!

Feedback

Questions, comments, suggestions...

...Thanks!