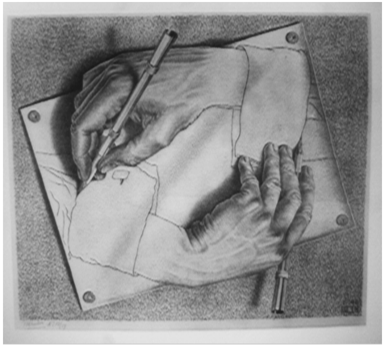
Principles of Concurrent and Distributed Programming

Emilio Tuosto

Academic Year 2025/2026

January 2026

Concurrency in Java



Processes vs. Threads

Multitasking

Many activities at once none of which "is aware" of the others (e.g., time slicing)

Processes

Running programs with their own execution environment containing basic run-time resources e.g. the processes' **address space**.

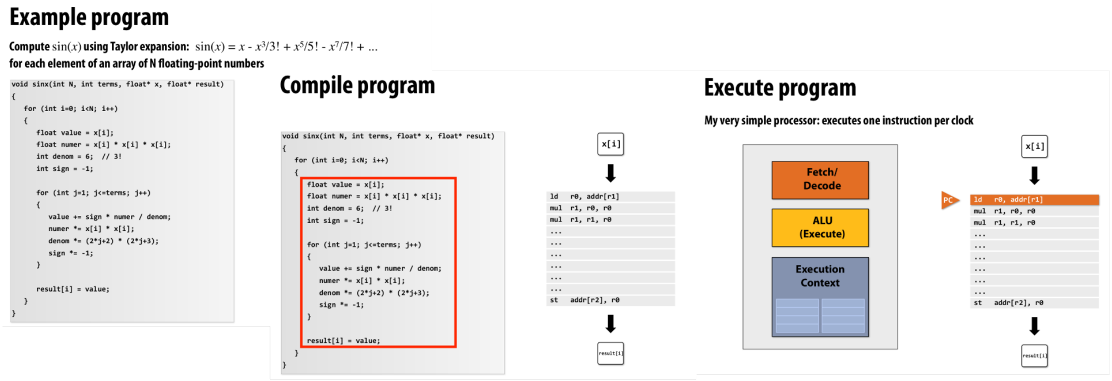
Threads

Sequential flows of control within a process (a process can consist of many **concurrent** threads)

> *Threads are also known as lightweight processes because creating a new thread requires fewer resources than creating a new process. Threads "lives" within a process and can share the process's resources (e.g., memory, files). In general multi-threaded applications have a "main" thread which can create new threads.*

Context switching

A (simplified) view of how processes interleave:



Example program

Compute $\sin(x)$ **using Taylor expansion:** $\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \ldots$
for each element of an array of N floating-point numbers

```
void sinx(int N, int terms, float* x, float* result)
{
   for (int i=0; i<N; i++)
   {
      float value = x[i];
      float numer = x[i] * x[i] * x[i];
      int denom = 6;  // 3!
      int sign = -1;

      for (int j=1; j<=terms; j++)
      {
         value += sign * numer / denom;
         numer *= x[i] * x[i];
         denom *= (2*j+2) * (2*j+3);
         sign *= -1;
      }

      result[i] = value;
   }
}
```

Compile program

```
ld   r0, addr[r1]
mul  r1, r0, r0
mul  r1, r1, r0
...
...
...
...
...
...
st   addr[r2], r0
```

Execute program

My very simple processor: executes one instruction per clock

Borrowed from `https://maxnilz.com/docs/006-arch/001-cpu-basics/`

Programming with threads

From [Eck02]

Concurrent programming is like stepping into an entirely new world and learning a new programming language, or at least a **new set of language concepts**. With the appearance of thread support in most microcomputer operating systems, extensions for threads have also been appearing in programming languages or libraries. In all cases, thread programming:

- Seems **mysterious** and requires a shift in the way you think about programming
- Looks similar to thread support in other languages, so **when you understand threads, you understand a common tongue**.

[...] threads are tricky.

Concurrency and Java OO

Some documentation

`https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html`

There is no general approach to concurrent programming.

Some rule of thumb

Java's "style" suggests

- to individuate active and passive objects
  - an active object is basically an object representing a thread
  - a passive object represents a resource that can be concurrently accessed by active objects
- Reason about how objects "interacts"
  - how does active objects' execution interleave?
  - how do active objects access shared resources?
- Acquire/release policy
  - in which order active objects acquire shared resources?
  - under which conditions shared resources can be invoked?
  - do active objects release all the acquired resources when they are not any longer needed?

Threads in Java

- Runnable: interface (method run() to be implemented)
- Threads: class (implements Runnable, run() is just empty)
  - Constructors
    - Thread()
    - Thread(Runnable target)
    - Thread(Runnable target, String name)
    - Thread(String name)
    - ... (see the Java thread API)
  - start, sleep, yield
  - interrupt

"When something has a Runnable interface, it simply means that it has a run() method, but there's nothing special about that –it doesn't produce any innate threading abilities, like those of a class inherited from Thread." [Eck02]

To create and run Java thread from a Runnable object:

- create the Runnable object
- use the special Thread constructors with runnable objects
- run the thread by invoking its start() method (which performs some initialisations and then calls the run() method)

Some examples

A simple scenario

Write a program that

- decides if staff is worth a promotion according to their state of service
- prints a report about the decision

Let's consider some solutions

- introducing some Java primitives for threads
- and showing how tricky concurrency can be

Some examples

A simple scenario

Write a program that

- decides if staff is worth a promotion according to their state of service
- prints a report about the decision

Let's consider some solutions

- introducing some Java primitives for threads
- and showing how tricky concurrency can be

Don't do this at home!

- PromotionConcurrent.java: a first attempt
- PromotionMoreConcurrent.java: an improved version
- IoC.java: pausing threads

Controlling threads

interrupt(): interrupts the thread on which it is invoked

yield(): Occasionally, a thread can decide to “give a hint to the thread scheduling mechanism” ([Eck02]) that it is keen to pass the control to another thread.
In Java this is done by invoking the yield() method from run.

join(): when invoked on a thread object, the invoking thread waits for the first thread to complete before proceeding (there is also a version with timeout). join() must be withing a try-catch statement because an interrupt() signal can abort the calling thread.

isAlive(): returns ‘true’ if the thread is running.

Mutual exclusion in Java

The mechanism that is offered by Java is method synchronisation

- Synchronised Methods can prevent thread interference and memory consistency errors
- Synchronisation based on (implicit) locks

The synchronized modifier can be used in method declarations or for determining critical sections.

- A method declared synchronized cannot be invoked while another synchronised method is executing
- (hence) If more than 2 threads try to invoke a synchronised method, only one of them actually access the object, while the other is blocked
- synchronized(obj){stm}: acquires the lock on obj, executes stm and releases the lock; stm is the critical section on the shared resource obj

Semaphores in java

```
public class Semaphore {
    private int counter = 0;
    private int threshold = 0;
    public Semaphore(int counter) { this.counter = counter; }

    public synchronized void P() {
            while(this.counter <= threshold) {
                    try {
                            this.wait();
                    } catch (InterruptedException e) {
                            e.printStackTrace();
                    }
            }
            this.counter--;
    }

    public synchronized void V() {
            this.counter++;
            if ( this.counter - 1 == threshold )
                    // this.notifyAll();
                    this.notify();
    }
}
```

Monitors in Java

It is important to remind that waiting threads must be notified before releasing the shared object

- public final void wait() throws InterruptedException
  - The thread is suspended and it is put on the object waiting list
- public final void wait(long timeout) throws InterruptedException
  - The thread is suspended until another wakes it up or until the time is elapsed
- public final void notify()
  - Choses and Wakes up a single thread among those waiting on the object monitor.
  - Which thread is chosen depends on the implementation of the JVM
  - This method should only be called by the "owner thread", namely the one which is
    - executing a synchronized statement that synchronizes on the object
    - executing the body of a synchronized statement that synchronizes on the object
  - Throws: IllegalMonitorStateException - if the current thread does not own the object
- public final void notifyAll()
  - Like notify, but awakes all the waiting threads

Remote Method Invocation in Java

https://docs.oracle.com/en/java/javase/24/docs/specs/rmi/intro.html

RPC vs RMI

Remote Method Invocation (RMI) is the Java correspondent of RPC.

- Instead of remote procedure calls, RMI implements remote method calls (i.e., calls of methods of remote objects)
- a key difference between RPC and Java RMI is that the latter allows Java objects to communicate, while the former provides, in general, a communication middleware for programs written in different languages
- RPC can be seen as a very primitive form of message oriented middleware and is data oriented. Java RMI, on the contrary, you can communicate objects, namely data & behaviour!

Remark

Observe that Java RMI allows objects running in a JVM to invoke methods of (Java) objects running in a different JVM

Distributed Objects: some terminology

- Distributed object : an object whose methods can be remotely invoked. A distributed object is provided, or exported by the object server .
- Remote method : a public method of a distributed object.
- Object registry : is the equivalent of the RPC name server. Namely, it is used by object servers to register their services and by object clients to look up for service references.
- Client/server proxy : is the equivalent of client/server stubs in RPC. Object clients call a remote method appear direct at the programmer level. However,
  - on the client host, the client proxy interacts with the software providing runtime support for the distributed object system
  - the runtime support transmits the actual call to the remote host (it also marshals the parameter to be transmitted)
  - similarly, on the object server side, the runtime support for the distributed object system handles the incoming messages (and their unmarshalling), and forwards the call to the server proxy

Java RMI: the first step

In Java:

- remote objects are those objects extending the java.rmi.Remote remote interface . Basically, interfaces plays the role of ID, hence the IDL of Java RMI is java.rmi.Remote
- the object server
  - implements a remote interface
  - generates stub and skeleton
  - register a distributed object implementing the interface
- An object client accesses the object by invoking the remote methods associated with the objects using syntax provided for remote method invocations

Remark

Within RMI, remote objects are treated differently from non-remote objects. For instance, what RMI actually passes when a remote object reference r_obj is sent to a remote object is a remote stub for r_obj. The stub acts as the local proxy for r_obj so that the caller is unaffected and calls r_obj via its stub.

Java RMI: the second step

Applications relying on distributed object must:

- Locate remote objects
  - by passing remote object references or
  - by using the object registry
- Communicate with remote objects
- Load class bytecodes for objects passed around: since RMI allows a caller to pass objects within calls to remote methods, RMI yields the necessary mechanisms for loading an object's code and for transmitting its data.

Remark

One of the central features of RMI is the possibility of dinamically downloading bytecodes of the class of an object when it is not defined in the caller's JVM. Basically, the types and the behavior of an object can be transmitted to possibly remote JVMs. RMI guarantees that the behavior of objects remains unchanged when they are sent to another JVM and allows new types to be introduced into a remote virtual machine, so that an application can be dynamically extended .

Creating Distributed Applications Using RMI

Using RMI to develop a distributed application requires you to follow these general steps:

1. Design and implement the distributed application components
2. Compile sources and generate stubs
3. Make classes network accessible: In this step you make everything–the class files associated with the remote interfaces, stubs, and other classes that need to be downloaded to clients–accessible via a Web server.
4. Start the application: Starting the application means to run:
    1. the RMI remote object registry
    2. the server
    3. the client

1. Design and implement the distributed application components

First, give an initial architecture for your application (this might require some revision at a later stage) and determine which components are local objects and remote objects .
This phase consists of:

- remote interfaces definition : this specifies the remote methods When designing remote interfaces you have to determine any local objects that will be used as parameters and return values for these methods
- remote objects implementation : generally, remote objects have to implement several remote interfaces (of course, the remote object class may implements other non-remote interfaces and define methods available only locally). Any local classes used in remote method invocations (as parameters or return values) must be implemented.
- clients implementation : clients invoking remote objects can be implemented at any time after the definition of remote interfaces or after deployment of remote objects.

2. Compile sources and generate stubs

This phase has two steps:

- use javac to compile the server classes (those implementing remote interfaces) and the client classes
- use rmic compiler in order to create stubs for remote objects.

Remark

The Java rmic compiler generates the stubs, namely, the programmer does not have to program client and server proxies and low lever programming detail.

Java remote interface

- In a remote interface each method signature must throw RemoteException Other than this, a remote interface has the same syntax as any other Java interface.
- RemoteException exception is raised if errors occur when processing remote method call. The exception is must be caught by the caller.
- RemoteException can be caused
  - by exceptions that may occur during communications (e.g., access or connection failures)
  - by problems in remote method invocations (e.g., errors resulting from object, stub, or skeleton not being found)

An example:

```
import java.rmi.*;

public interface ARemoteInterface extends Remote {
    String aRemoteMethod1( ... ) throws RemoteException;
    int aRemoteMethod2( ... ) throws RemoteException;
}
```

An example: the compute engine

The compute engine is a protocol to execute tasks on a remote engine. This protocol is based on interfaces supported by the compute engine and by the objects that are submitted to the compute engine.
The remotely accessible part is the compute engine itself, whose remote interface has a single method:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
/*
 * The 2 lines above can be replaced by
 * import java.rmi.*;
 */
public interface Compute extends Remote {
    public Object executeTask(Task t)
        throws RemoteException;
}
```

By extending java.rmi.Remote, the interface Compute allows its method to be called from any JVM. Any object implementing Compute becomes a remote object.
Notice that executeTask

- takes a Task
- can return any Object
- throws RemoteException

An example: the compute engine (2)

An interface for Task objects must be defined.

```
import java.io.Serializable;

public interface Task extends Serializable {

    public Object execute();

}
```

Different kinds of tasks can be run by a Compute object provide that they implementat Task. It is possible to add further methods (or data) needed for the computation of the task.

Exercise

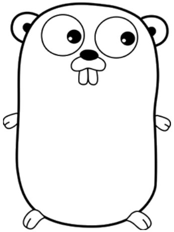
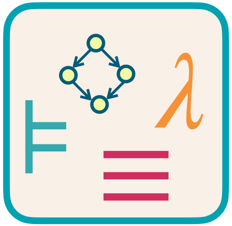
execute is not required to throw RemoteException. Why?

Remark

The Task interface extends the java.io.Serializable interface to let the RMI middleware serialise objects so that they can be transported from a JVM to another.
Implementing Serializable marks the class as being capable of conversion into a self-describing byte stream that can be used to reconstruct an exact copy of the serialized object when the object is read back from the stream.

This implies that local objects are passed by-value while remote objects are passed by-reference .

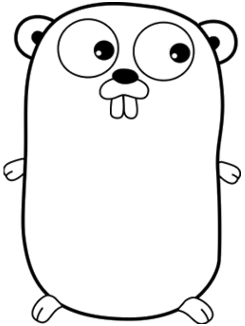
Channel-based concurrency

slides are courtesy of R. Bruni and F. Bonchi

Google Go

http://golang.org/



Go features

facilitate building reliable and efficient software

open source

compiled, garbage collected

functional and OO features

statically typed (light type system)

concurrent

Go principles

C, C++, Java:
too much typing (writing verbose code)
and too much typing (writing explicit types)
(and poor concurrency)

Python, JS:
no strict typing, no compiler issues
runtime errors that should be caught statically

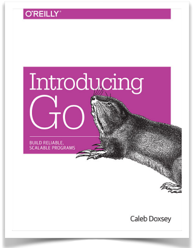
Google Go:
compiled, static types, type inference
(and nice concurrency primitives)

Go project

designed by Ken Thompson, Rob Pike, Robert Griesemer

2007: started experimentation at Google
nov 2009: first release (more than 250 contributors)
may 2012: version 1.0 (two yearly releases since 2013)
feb 2025: version 1.24.0

C. Doxsey, Introducing Go (2016). Ch: 1-4, 6-7, 10



Go concurrency

any function can be executed in a separate lightweight thread

```
go f(x)
```

goroutines run in the same address space

package `sync` provides basic synchronisation primitives

programmers are encouraged NOT TO USE THEM!

do not communicate by sharing memory
instead, share memory by communicating

use built-in high-level concurrency primitives:
channels and **message passing**
(inspired by process algebras)

Go channels

channels can be created and passed around

```
var ch = make(chan int)
```

creates a channel for transmitting integers

```
ch1 = ch
```

aliasing: `ch1` and `ch` now refers to the same channel

```
go f(ch)
go g(ch)
```

`f` and `g` share the channel `ch`

Directionality

channels are alway created bidirectional

```
var ch = make(chan int)
```

channel types can be annotated with directionality

```
var rec <-chan int
```

`rec` can only be used to receive integers

```
var snd chan<- int
```

`snd` can only be used to send integers

```
rec = ch
snd = ch
```

are valid assignments

```
rec = snd // invalid!
```

Go communication

to send a value (like *ch!2*) `ch <- 2`

to receive and store in `x` (like *ch?x*) `x = <- ch`

to receive and throw the value away `<- ch`

to close a channel (by the sender) `close(ch)`

to check if a channel has been closed (by the receiver)

```
x,ok = <- ch  // either value,true or 0,false
```

Go sync communication

by default the communication is **synchronous**

BOTH send and receive are BLOCKING!

asynchronous channels can be created
by allocating a buffer of fixed size

```
var ch = make(chan int, 100)
```

creates an **asynchronous channel** of size 100

receive on asynchronous channel is of course still blocking

send is blocking only if the buffer is full

no dedicated type for asynchronous channels:
buffering is a property of values not of types

Go communication

to choose between different options

```
select {
    case x = <- ch1: { … }
    case ch2 <- v: { … }
    // both send and receive actions
    default: { … }
}
```

the selection is made pseudo-randomly among enabled cases

if no case is enabled, the default option is applied

if no case is enabled, and no default option is given
the select blocks until (at least) one case is enabled

Example

non-blocking receive

```
select {
    case x = <- ch: { … }
    default: { … }
}
```

receives on `x` from `ch`, if data available

otherwise proceeds

Example

wait for first among many (senders)

```
select {
   case x = <- ch1: { … }
   case x = <- ch2: { … }
   case x = <- ch3: { … }
}
```

receives on `x` from any of `ch1`, `ch2`, `ch3`, if data available

otherwise waits

Example

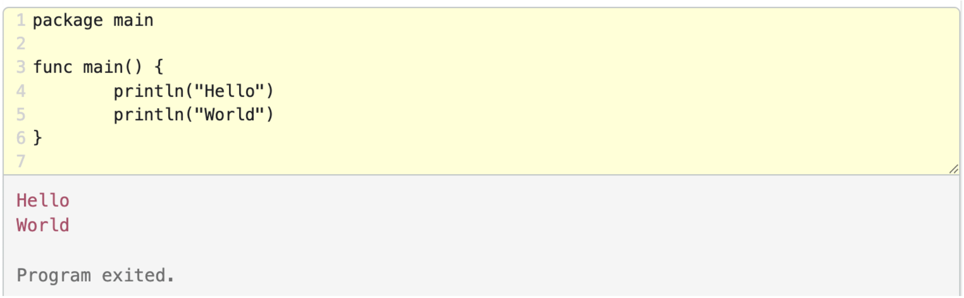
wait for first among many (receivers)

```
select {
   case ch1 <- v : { … }
   case ch2 <- v : { … }
   case ch3 <- v : { … }
}
```

sends `v` to any of `ch1`, `ch2`, `ch3`, if available to receive

otherwise waits

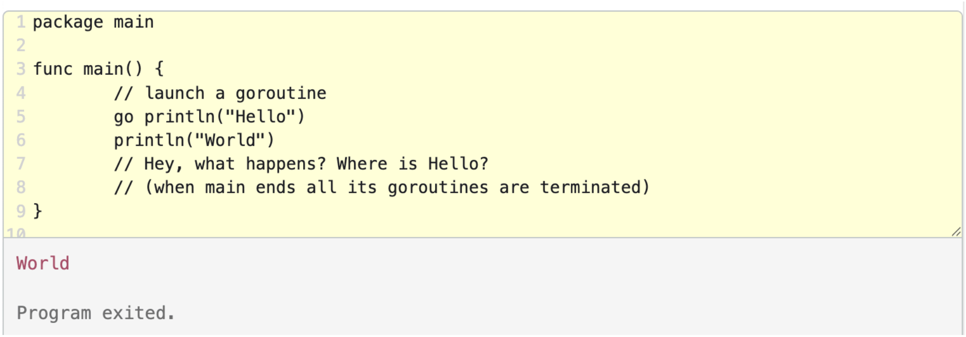
Hello concurrency



```go
package main

func main() {
	println("Hello")
	println("World")
}

```

```
Hello
World

Program exited.
```

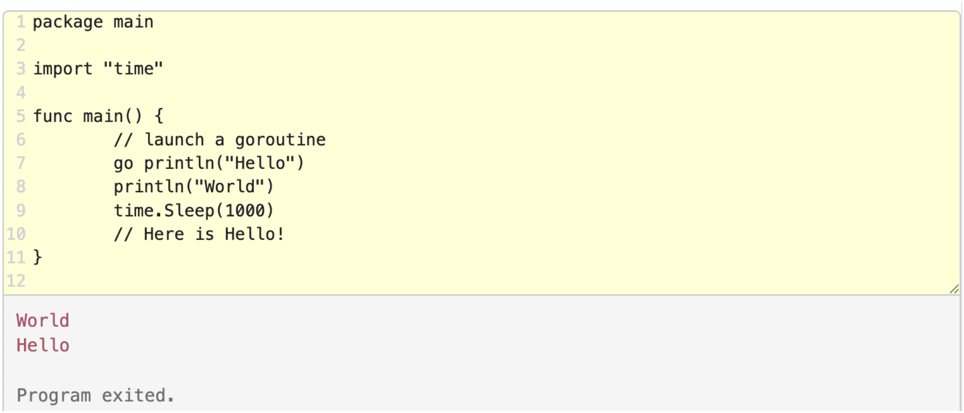
Hello concurrency



```go
package main

func main() {
	// launch a goroutine
	go println("Hello")
	println("World")
	// Hey, what happens? Where is Hello?
	// (when main ends all its goroutines are terminated)
}
```

```
World

Program exited.
```

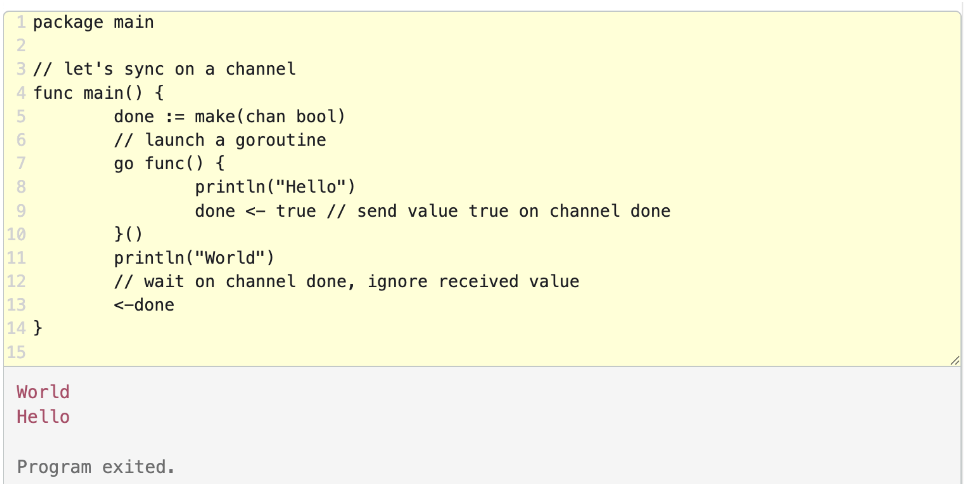
Hello concurrency



```go
package main

import "time"

func main() {
	// launch a goroutine
	go println("Hello")
	println("World")
	time.Sleep(1000)
	// Here is Hello!
}
```

```
World
Hello

Program exited.
```

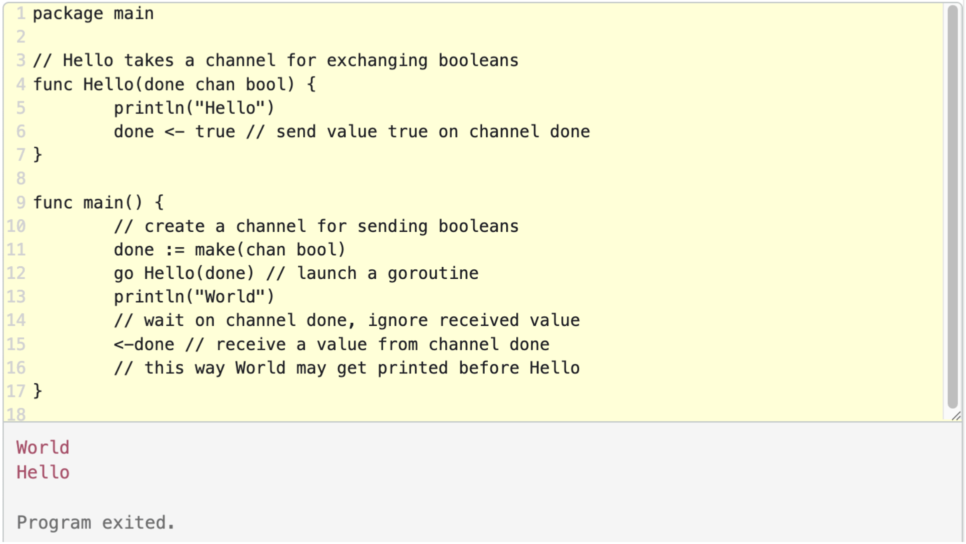
Hello concurrency



```
package main

// let's sync on a channel
func main() {
	done := make(chan bool)
	// launch a goroutine
	go func() {
		println("Hello")
		done <- true // send value true on channel done
	}()
	println("World")
	// wait on channel done, ignore received value
	<-done
}
```

```
World
Hello

Program exited.
```

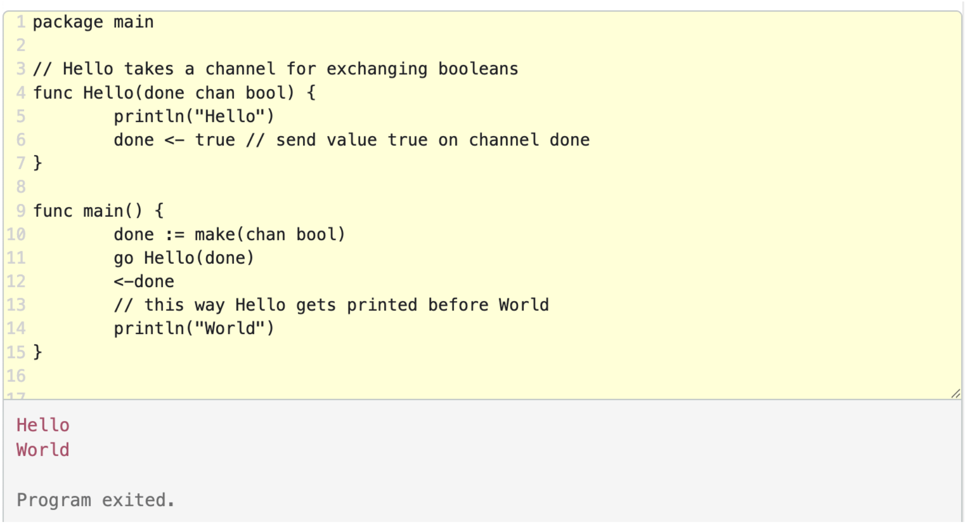
Hello concurrency



```go
package main

// Hello takes a channel for exchanging booleans
func Hello(done chan bool) {
	println("Hello")
	done <- true // send value true on channel done
}

func main() {
	// create a channel for sending booleans
	done := make(chan bool)
	go Hello(done) // launch a goroutine
	println("World")
	// wait on channel done, ignore received value
	<-done // receive a value from channel done
	// this way World may get printed before Hello
}

```

```
World
Hello

Program exited.
```

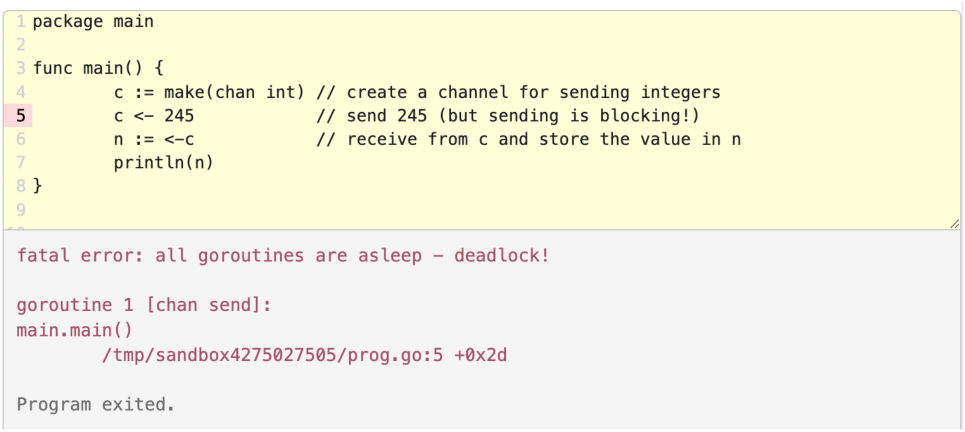
Hello concurrency



```go
package main

// Hello takes a channel for exchanging booleans
func Hello(done chan bool) {
	println("Hello")
	done <- true // send value true on channel done
}

func main() {
	done := make(chan bool)
	go Hello(done)
	<-done
	// this way Hello gets printed before World
	println("World")
}

```

```
Hello
World

Program exited.
```

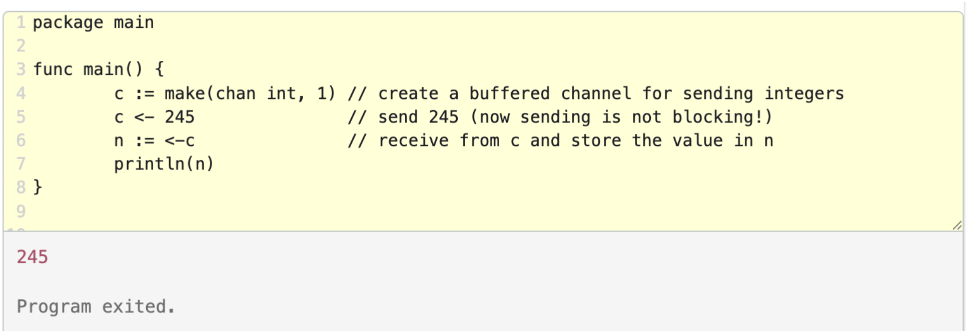
Hello deadlocks



```
1 package main
2 
3 func main() {
4         c := make(chan int) // create a channel for sending integers
5         c <- 245            // send 245 (but sending is blocking!)
6         n := <-c            // receive from c and store the value in n
7         println(n)
8 }
9 
```

```
fatal error: all goroutines are asleep - deadlock!

goroutine 1 [chan send]:
main.main()
        /tmp/sandbox4275027505/prog.go:5 +0x2d

Program exited.
```

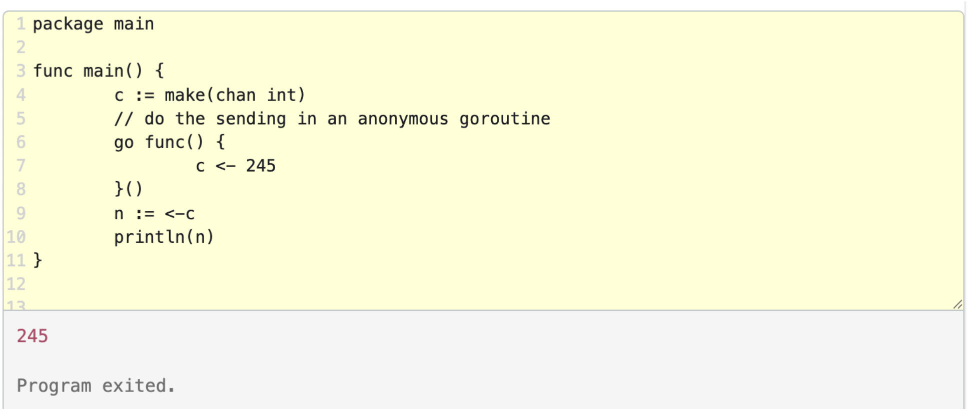
Buffering



```go
package main

func main() {
	c := make(chan int, 1) // create a buffered channel for sending integers
	c <- 245               // send 245 (now sending is not blocking!)
	n := <-c               // receive from c and store the value in n
	println(n)
}

```

```
245

Program exited.
```

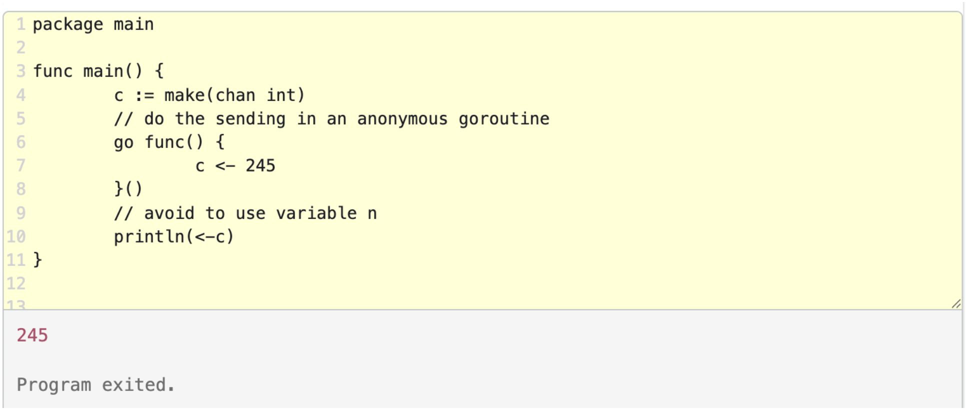
Communicating goroutines



```go
package main

func main() {
	c := make(chan int)
	// do the sending in an anonymous goroutine
	go func() {
		c <- 245
	}()
	n := <-c
	println(n)
}
```

```
245

Program exited.
```

Communicating goroutines



```go
package main

func main() {
	c := make(chan int)
	// do the sending in an anonymous goroutine
	go func() {
		c <- 245
	}()
	// avoid to use variable n
	println(<-c)
}

```

```
245

Program exited.
```

Name mobility

channels can be sent over channels (like in π-calculus)

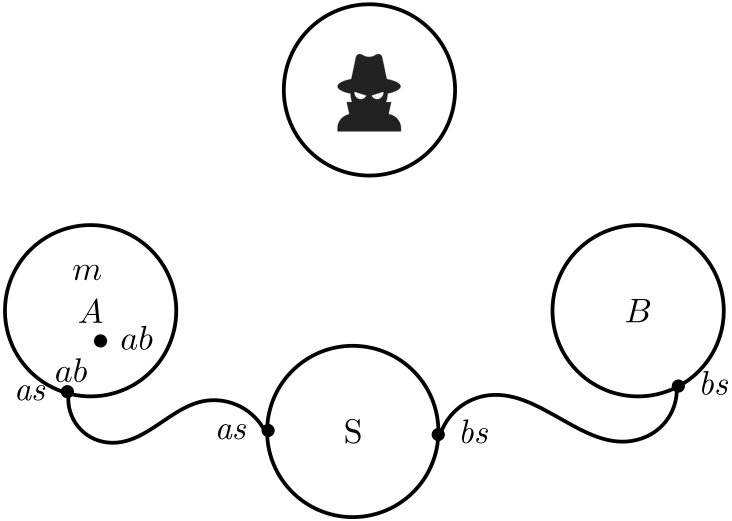
```
var mob = make(chan chan int)
```

a channel for communicating channels

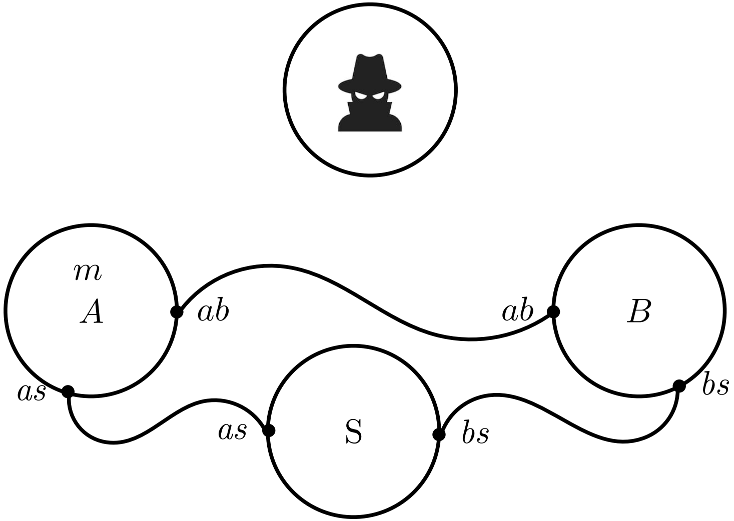
```
mob <- ch
```

send the channel `ch` over `mob`

Name mobility: secrecy



Name mobility: secrecy

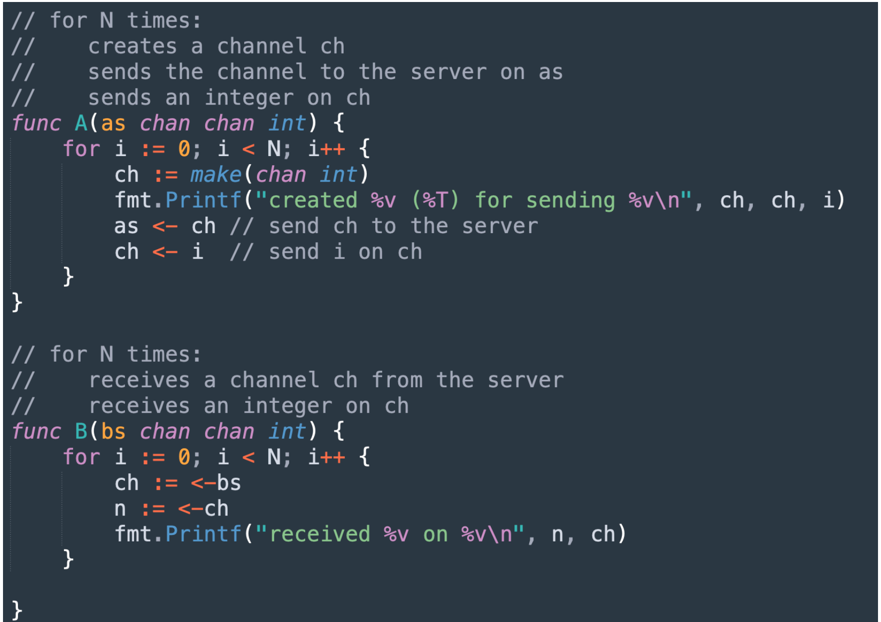


Name mobility

```go
func main() {
    as, bs := Serv() // launch server, get secure channels
    go A(as)         // launch A
    B(bs)            // run B
}
```

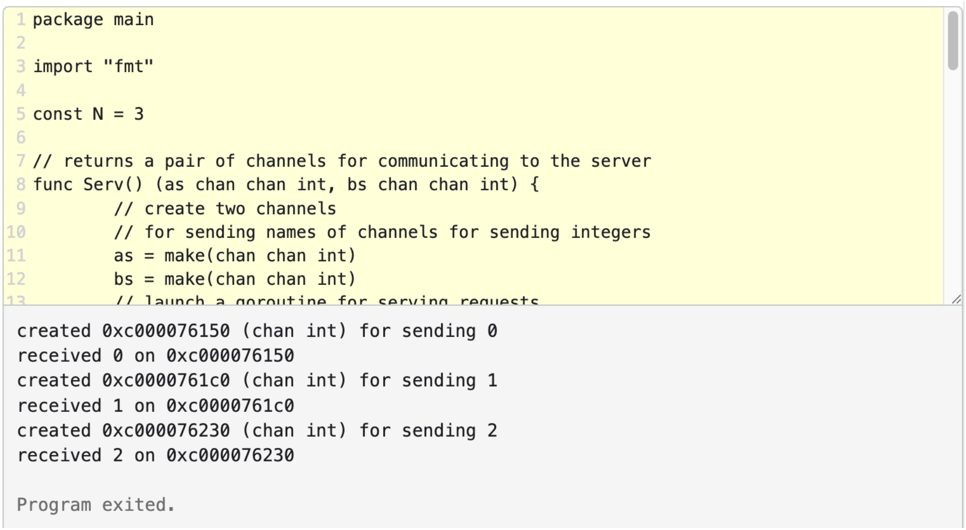
```go
// returns a pair of channels for communicating to the server
func Serv() (as chan chan int, bs chan chan int) {
    // create two channels
    // for sending names of channels for sending integers
    as = make(chan chan int)
    bs = make(chan chan int)
    // launch a goroutine for serving requests
    go func() {
        for {
            // forward messages from as to bs
            c := <-as
            bs <- c
        }
    }()
    return  // naked return
}
```

Name mobility



```go
// for N times:
//     creates a channel ch
//     sends the channel to the server on as
//     sends an integer on ch
func A(as chan chan int) {
    for i := 0; i < N; i++ {
        ch := make(chan int)
        fmt.Printf("created %v (%T) for sending %v\n", ch, ch, i)
        as <- ch // send ch to the server
        ch <- i  // send i on ch
    }
}

// for N times:
//     receives a channel ch from the server
//     receives an integer on ch
func B(bs chan chan int) {
    for i := 0; i < N; i++ {
        ch := <-bs
        n := <-ch
        fmt.Printf("received %v on %v\n", n, ch)
    }

}
```

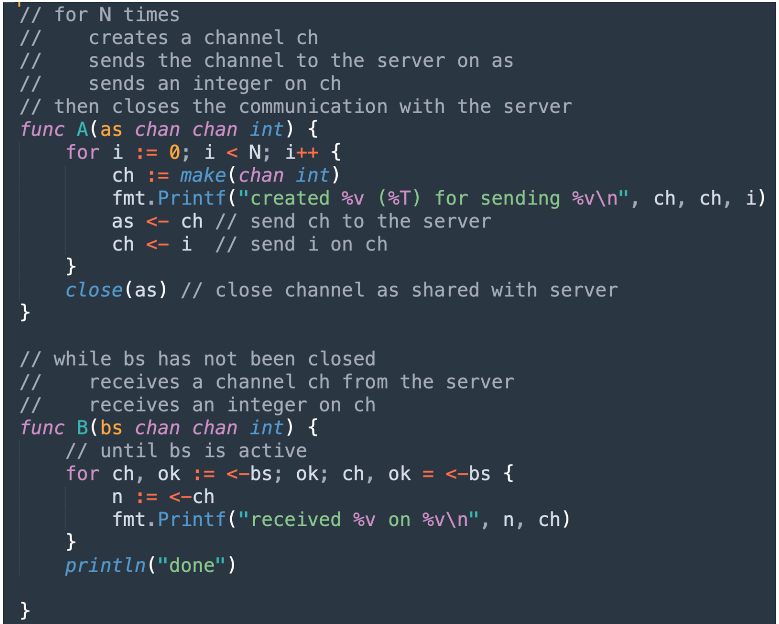
Name mobility



```go
package main

import "fmt"

const N = 3

// returns a pair of channels for communicating to the server
func Serv() (as chan chan int, bs chan chan int) {
        // create two channels
        // for sending names of channels for sending integers
        as = make(chan chan int)
        bs = make(chan chan int)
```

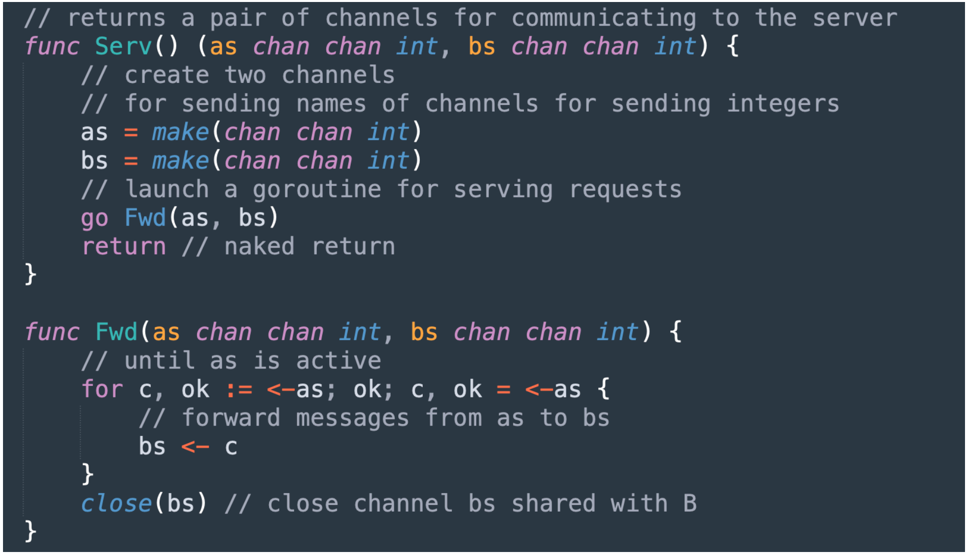
```
created 0xc000076150 (chan int) for sending 0
received 0 on 0xc000076150
created 0xc0000761c0 (chan int) for sending 1
received 1 on 0xc0000761c0
created 0xc000076230 (chan int) for sending 2
received 2 on 0xc000076230

Program exited.
```

Closing channels



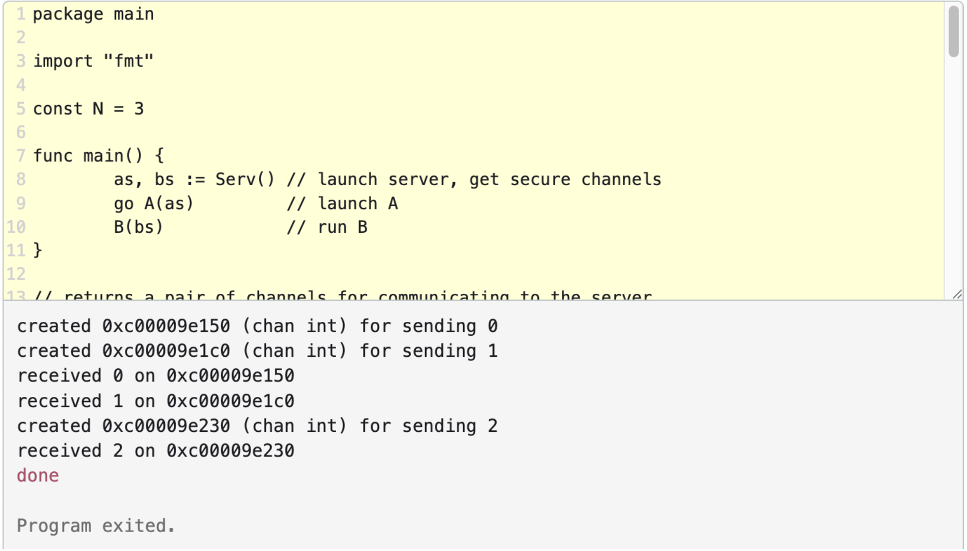
```go
// for N times
//    creates a channel ch
//    sends the channel to the server on as
//    sends an integer on ch
// then closes the communication with the server
func A(as chan chan int) {
    for i := 0; i < N; i++ {
        ch := make(chan int)
        fmt.Printf("created %v (%T) for sending %v\n", ch, ch, i)
        as <- ch // send ch to the server
        ch <- i  // send i on ch
    }
    close(as) // close channel as shared with server
}

// while bs has not been closed
//    receives a channel ch from the server
//    receives an integer on ch
func B(bs chan chan int) {
    // until bs is active
    for ch, ok := <-bs; ok; ch, ok = <-bs {
        n := <-ch
        fmt.Printf("received %v on %v\n", n, ch)
    }
    println("done")

}
```

Closing channels



```go
// returns a pair of channels for communicating to the server
func Serv() (as chan chan int, bs chan chan int) {
    // create two channels
    // for sending names of channels for sending integers
    as = make(chan chan int)
    bs = make(chan chan int)
    // launch a goroutine for serving requests
    go Fwd(as, bs)
    return // naked return
}

func Fwd(as chan chan int, bs chan chan int) {
    // until as is active
    for c, ok := <-as; ok; c, ok = <-as {
        // forward messages from as to bs
        bs <- c
    }
    close(bs) // close channel bs shared with B
}
```

Closing channels



```go
package main

import "fmt"

const N = 3

func main() {
        as, bs := Serv() // launch server, get secure channels
        go A(as)         // launch A
        B(bs)            // run B
}

// returns a pair of channels for communicating to the server
```

```
created 0xc00009e150 (chan int) for sending 0
created 0xc00009e1c0 (chan int) for sending 1
received 0 on 0xc00009e150
received 1 on 0xc00009e1c0
created 0xc00009e230 (chan int) for sending 2
received 2 on 0xc00009e230
done

Program exited.
```

Actor-based concurrency

wslides are courtesy of R. Bruni and F. Bonchi

Erlang: a concurrent programming language

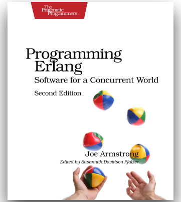
`http://www.erlang.org/`

Erlang: origins

named after Danish mathematician A. K. Erlang

1986: first experimentation at Ericsson, Sweden
1989: internal use only
1990: sold as a product
1998: open source

Joe Armstrong, “Programming Erlang”, ch.1-5, 11-12


The Pragmatic Programmers
Programming Erlang
Software for a Concurrent World
Second Edition
Joe Armstrong
Edited by Susannah Davidson Pfalzer

Features

declarative (functional, Prolog) programming

arbitrary size integers, tuples, lists, functions, higher-order

atoms everywhere

dynamically typed

open source

unfriendly syntax

variables are assigned only once

left-to-right evaluation, no pointers, no object-orientation

Features: concurrency

concurrent and distributed programming

asynchronous message passing
(no locks, no mutexes)

fault tolerance

hot swapping code

erlang processes are cheap

automatic memory allocation and garbage collection

can handle large telecom applications

Erl

Erlang: erl

erl is the Erlang VM emulator

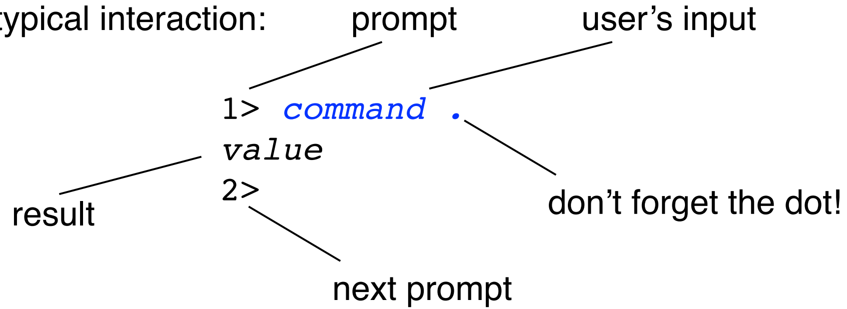
interactive shell or interpreter, executing read-eval-print loop

programmers enter expressions / declarations one at a time

they are compiled / executed

erl expressions

typical interaction:



```
1> command .
value
2>
```

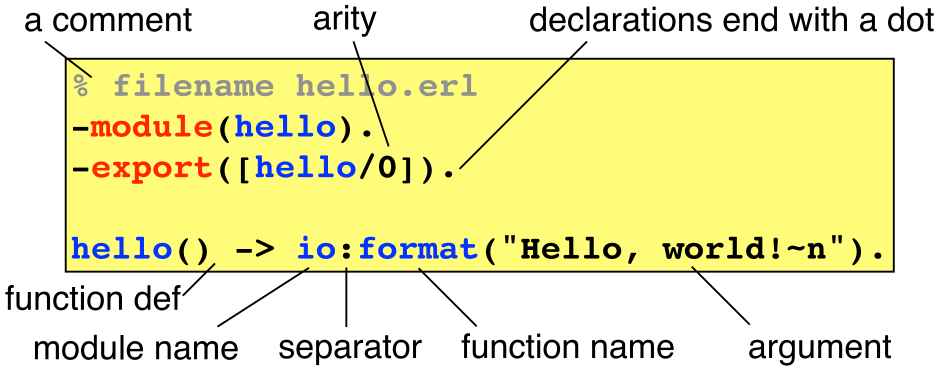
`halt().` to exit the emulator

Erlang modules

functions are organised in modules

one module for source file

filename is module name with suffix `.erl`



```erlang
% filename hello.erl
-module(hello).
-export([hello/0]).

hello() -> io:format("Hello, world!~n").
```

erl: module loading

compile and load the module

invoke the function

```
1> c(hello) .
{ok,hello}
2> hello:hello() .
Hello, world!
ok
3>
```

return value

next prompt

if you edit `hello.erl` and do `c(hello)` again
the new version of the module replaces the old one

Erlang basics

Function definition

separates function clauses with ;
last clause ends with .

variables start with upper-case letters X Head Tail
variables are local to function clauses

function definitions cannot be nested
non-exported functions are local to the module

pattern matching allowed

guards allowed (keyword `when`)

type-checking is done at runtime

Atoms, tuples, lists

numbers: arbitrary size integers, floating point values
(cannot start with .)

atoms: start with lower-case character
(can be single-quoted if needed, don't use camelCase)

```
true  ok  hello_world.  'this is an atom'
```

tuples: main data constructor
tagged tuples: the first element of the tuple is an atom
we can use pattern matching

```
{}  {movie,"Matrix"}   {movie,Title}
```

lists: can contain elements of any type
we can use pattern matching

```
[]  [1,2,ok]  [H|T]  [X,Y,Z]  [X,Y,Z| Tail]
```

Funs

funs: anonymous functions (lambda expressions)
can have several arguments and clauses

```
fun () -> 42 end

fun (X) -> X+1 end

fun (X,Y) -> {X, fun (Z) -> Z+Y end} end

fun (F,X) -> F(X) end
```

Type test & conversion

```
is_integer(X)
is_float(X)
is_number(X)
is_atom(X)
is_tuple(X)
is_list(X)
is_function(X)
is_pid(X)
…
```

```
atom_to_list(A)
list_to_atom(L)
tuple_to_list(T)
list_to_tuple(L)
…
```

Erlang concurrency

Processes

every Erlang code is executed by a process

processes are implemented by the VM (not by OS threads)

multitasking is preemptive (VM switching and scheduling)

processes need very little memory

switching between processes is very fast

the VM can handle a large number of processes

on multiprocessor/multicore machines, processes can be scheduled to run in parallel on separate CPUs/cores using multiple schedulers

different processes may be reading the same program code at the same time (no variable updates!)

Pids

each process has a process identifier

```
Pid = self()
```

new Erlang processes can be spawned to run functions

```
Pid = spawn(module,function,arguments)
Pid = spawn(fun () -> … end)
Pid = spawn(fun f/0)
Pid = spawn(fun m:f/0)
```

the spawn operation returns immediately
(the return value is the pid of the process)

children pids are available to parent process,
not vice versa (unless passed)

Communication

Messages can be sent to pids

```
Pid ! message
```

called bang

Processes can wait to receive (and select) some message

```
receive
    Pattern1 when Cond1 -> Exp1;
    Pattern2 when Cond2 -> Exp2;
    ...
    Patternk when Condk -> Expk
end
```

Communication

Messages can be sent to pids

```
Pid ! {1,2,3}
```

called bang

Processes can wait to receive (and select) some message

```
receive
    {X} when X>0 -> X;
    {X,Y} when Y>X -> X+Y;
    {X,Y,Z} when Y>X andalso Z>Y -> X+Z;
end
```

First matching clause for first message,
if none, first matching clause for second message,
if none, ...
if none it blocks (all messages are kept)

Communication

Messages can be sent to pids

```
Pid ! {1,2,3}
```

called bang

Processes can wait to receive (and select) some message

```
receive
    {X} when X>0 -> X;
    {X,Y} when Y>X -> X+Y;
    {X,Y,Z} when Y>X andalso Z>Y -> X+Z;
    _ -> 0
end
```

First matching clause for first message
(the last clause, called catch-all, will match anyway)

Communication

Messages can be sent to pids

```
Pid ! {1,2,3}
```

called bang

Processes can wait to receive (and select) some message

```
receive
    {X} when X>0 -> X;
    {X,Y} when Y>X -> X+Y;
    {X,Y,Z} when Y>X andalso Z>Y -> X+Z;
    after 0 -> 0
end
```

timeout

First matching clause for first message,
if none, first matching clause for second message,
if none, ...
if none it evaluates to 0 (all messages are kept)

Message passing

`receive … end`

`Pid`

`Pid ! message`

Message passing

messages are sent asynchronously
(the sender continues immediately)

any value can be sent as a message

each process has a message queue (mailbox)
no size limit, messages are kept until extracted

a message is received when it is extracted from the mailbox

messages are ordered from oldest to newest in the mailbox

the message that is extracted is not necessarily the oldest
(pattern matching can be used, if there is no match
the receiver suspends and keeps waiting)

Reply

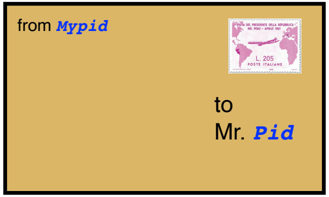
To reply a message, its sender must be known

its pid can be inserted in the message

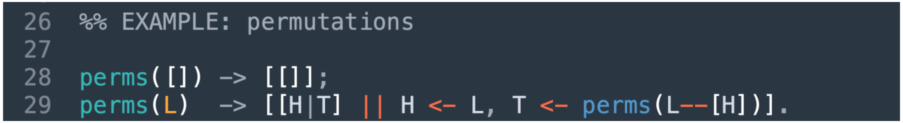
syntax for tuples

```
Pid ! { Mypid , message }
```

now the receiver `Pid` can reply to `Mypid`



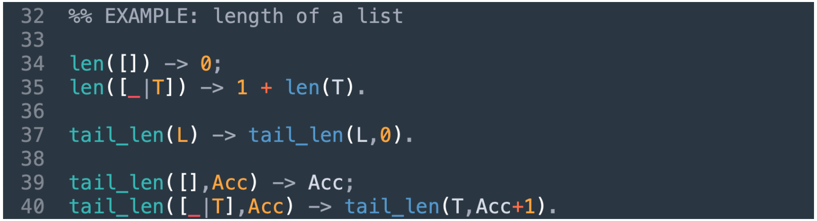
erl session



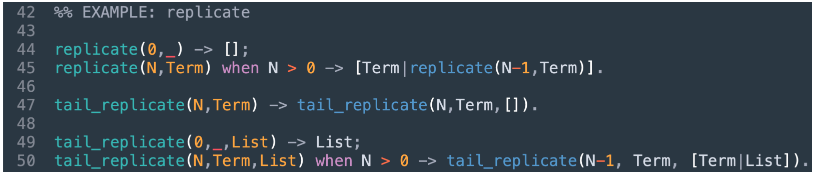
```erlang
%% EXAMPLE: permutations

perms([]) -> [[]];
perms(L)  -> [[H|T] || H <- L, T <- perms(L--[H])].
```

```
99> c(recursion).
recursion.erl:2:2: Warning: export_all flag enabled - all functions
will be exported
{ok,recursion}
100> recursion:perms("abc").
["abc","acb","bac","bca","cab","cba"]
101> recursion:perms("abcdef").
["abcdef","abcdfe","abcedf","abcefd","abcfde","abcfed",
 "abdcef","abdcfe","abdecf","abdefc","abdfce","abdfec",
 "abecdf","abecfd","abedcf","abedfc","abefcd","abefdc",
 "abfcde","abfced","abfdce","abfdec","abfecd","abfedc",
 "acbdef","acbdfe","acbedf","acbefd",
 [...]|...]
```

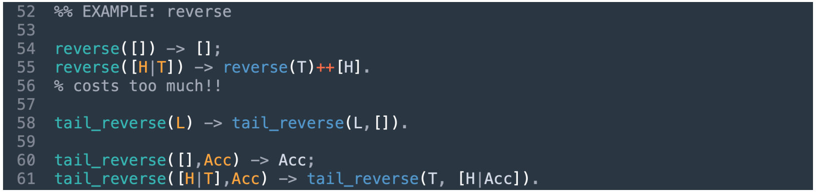
erl session



```erlang
%% EXAMPLE: length of a list

len([]) -> 0;
len([_|T]) -> 1 + len(T).

tail_len(L) -> tail_len(L,0).

tail_len([],Acc) -> Acc;
tail_len([_|T],Acc) -> tail_len(T,Acc+1).
```



```erlang
%% EXAMPLE: replicate

replicate(0,_) -> [];
replicate(N,Term) when N > 0 -> [Term|replicate(N-1,Term)].

tail_replicate(N,Term) -> tail_replicate(N,Term,[]).

tail_replicate(0,_,List) -> List;
tail_replicate(N,Term,List) when N > 0 -> tail_replicate(N-1, Term, [Term|List]).
```



```erlang
%% EXAMPLE: reverse

reverse([]) -> [];
reverse([H|T]) -> reverse(T)++[H].
% costs too much!!

tail_reverse(L) -> tail_reverse(L,[]).

tail_reverse([],Acc) -> Acc;
tail_reverse([H|T],Acc) -> tail_reverse(T, [H|Acc]).
```

[**Ex. 1**] Write a server in erlang to convert temperatures from Celsius degrees to Fahrenheit degrees and vice versa, using the formula $F = 1.8C + 32$. The server receives requests of the form $(Pid, \texttt{cs}, C)$ or $(Pid, \texttt{ft}, F)$ and replies to Pid by sending messages in analogous format. The server can be stopped by sending the message `stop`. All the other messages are ignored. Spawn a copy of the server, send it some temperatures to convert, check out the results and stop the server.

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive







    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} ->





    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();




    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();
     {Pid,ft,F} ->



    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();
     {Pid,ft,F} -> Pid ! {self(),cs,(F - 32) / 1.8},
                   convert();


    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();
     {Pid,ft,F} -> Pid ! {self(),cs,(F - 32) / 1.8},
                   convert();
     stop -> true;

    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();
     {Pid,ft,F} -> Pid ! {self(),cs,(F - 32) / 1.8},
                   convert();
     stop -> true;
     _ ->
    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();
     {Pid,ft,F} -> Pid ! {self(),cs,(F - 32) / 1.8},
                   convert();
     stop -> true;
     _ -> convert()
    end.
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2>
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> Conv = spawn(ex1,convert,[]).
<0.84.0>
3>
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> Conv = spawn(ex1,convert,[]).
<0.84.0>
3> Conv ! {self(),cs,23}.
{<0.77.0>,cs,23}
4>
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> Conv = spawn(ex1,convert,[]).
<0.84.0>
3> Conv ! {self(),cs,23}.
{<0.77.0>,cs,23}
4> receive
4>     {Conv,ft,F} -> io:format("23 celsius = ~p fahrenheit~n",[F])
4> end.
23 celsius = 73.4 fahrenheit
ok
5>
```

[**Ex. 2**] Write an erlang function `copy` that receives an integer n and if n is positive it prints n copies of n (one per line). Write an erlang function that receives a list of integers and spawn an instance of `copy` for each integer in the list.

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 ->
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) ->
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 ->
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
copy(_,_) -> true.
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
copy(_,_) -> true.

listCopy(L) ->
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
copy(_,_) -> true.

listCopy(L) -> [                              || N <- L ].
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
copy(_,_) -> true.

listCopy(L) -> [ spawn(ex2,copy,[N]) || N <- L ].
```

Ex. 2, copy

```
Eshell V10.2.1  (abort with ^G)
1> c(ex2).
{ok,ex2}
2>
```

Ex. 2, copy

```
Eshell V10.2.1  (abort with ^G)
1> c(ex2).
{ok,ex2}
2> ex2:listCopy(lists:seq(1,5)).
1
2
3
4
5
[<0.84.0>,<0.85.0>,<0.86.0>,<0.87.0>,<0.88.0>]
2
3
4
5
3
4
5
4
5
5
3>
```

On using the “right” primitives

Advanced primitives for concurrency

Join patterns are very high-level

Based on the join calculuc [FG96]

Integrated in some programming languages (Erlang, C#, etc.)

We'll see a combination of join patterns and actors introduced in [HHM+24a, HHM+24b]

- Novel specification of **fair join pattern matching** for actors
- Novel **stateful tree-based** matching algorithm with **proof of correctness**
- `JoinActors`: novel Scala 3 library for actors with fair join pattern matching

What are Join Patterns?

- Coordination mechanism for **concurrent** message passing programs
- Introduced in Join Calculus (Fournet et al., POPL 1996)

What are Join Patterns?

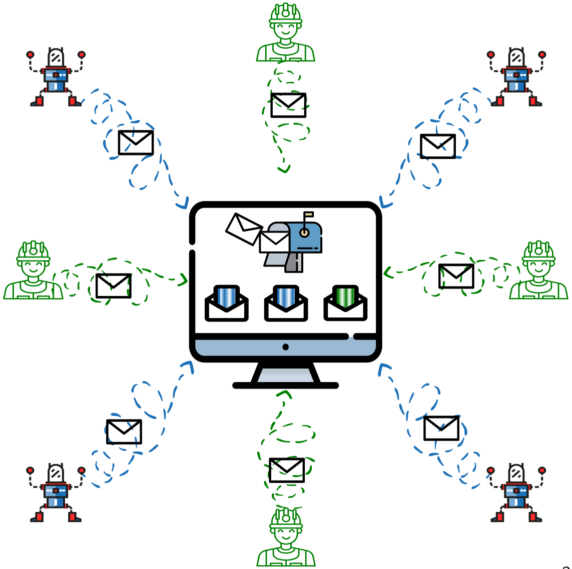
- Coordination mechanism for **concurrent** message passing programs
- Introduced in Join Calculus (Fournet et al., POPL 1996)
- Message passing programs may react to complex message sequences and conditions

What are Join Patterns?

- Coordination mechanism for **concurrent** message passing programs
- Introduced in Join Calculus (Fournet et al., POPL 1996)
- Message passing programs may react to complex message sequences and conditions
- Join patterns simplify specifying the **association of out-of-order messages**

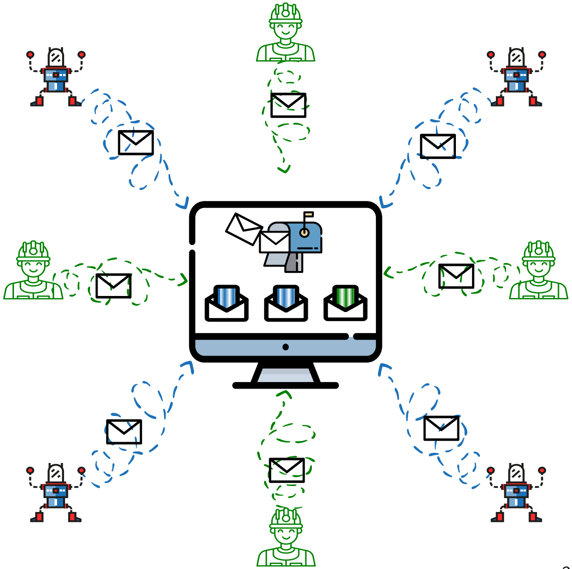
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers



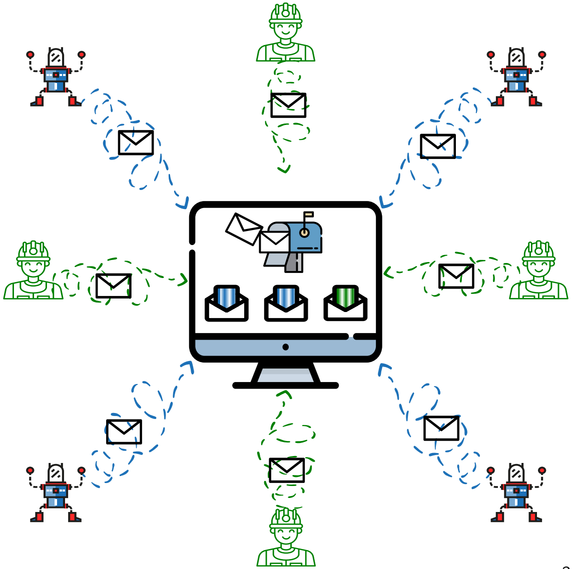
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**



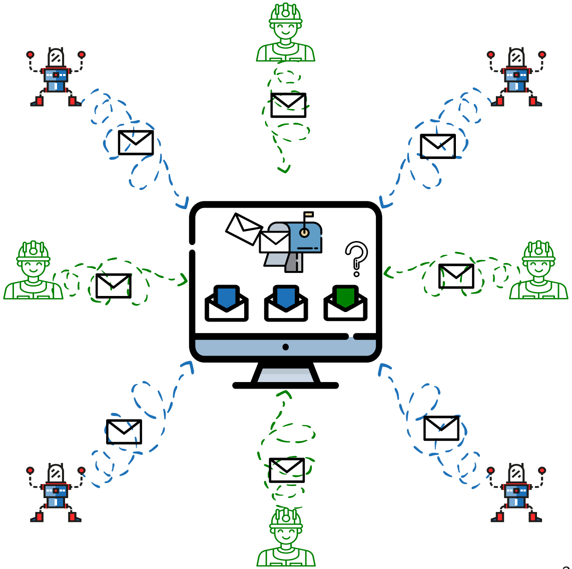
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**
- Monitor reacts to a combination of messages in the mailbox



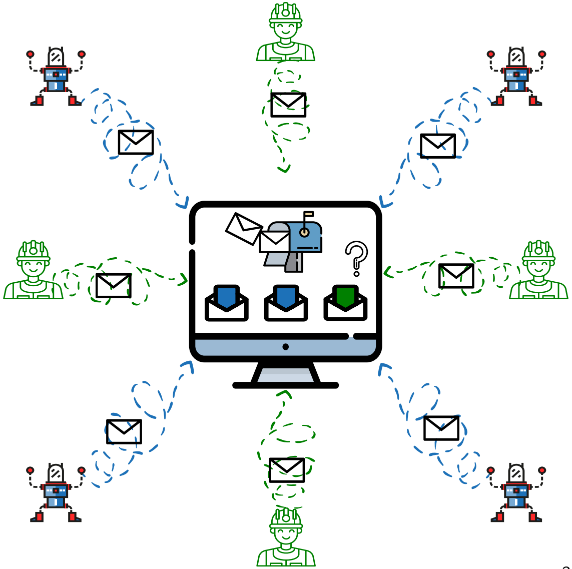
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**
- Monitor reacts to a combination of messages in the mailbox
- Traditionally, programmers write **custom code** for message association

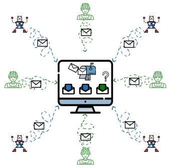


Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**
- Monitor reacts to a combination of messages in the mailbox
- Traditionally, programmers write **custom code** for message association (e.g., Akka/Pekko actors, Socket programming)



Factory Shop Monitor Using JoinActors

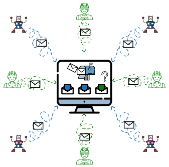


Using our `JoinActors` library we can **declaratively** specify **order-independent message associations**

```scala
def monitor() = Actor[...] {
  receive { (...) => {
    case (Fault(id1, _), Fix(id2, _)) if id1 == id2 => ...

    case (Fault(_, ts1), Fault(id2, ts2), Fix(id3, _))
                if id2 == id3 && ts2 - ts1 > TEN_MIN => ...
  }}
}
```

- Uses Scala 3 macros

Join Patterns More Formally



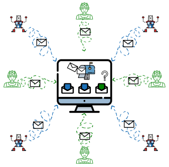
Let $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Refer to the paper for more details

Join Patterns Matching



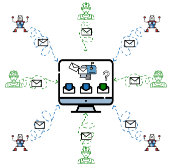
The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

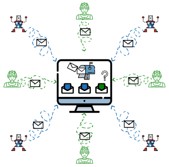
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$\mathcal{M}$ =

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

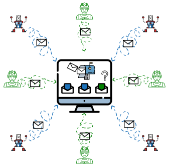
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

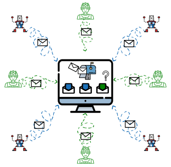
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

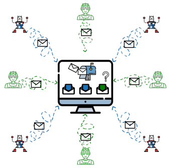
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

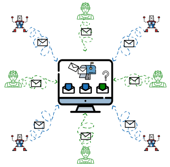
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathrm{D} = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

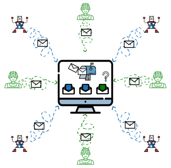
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1\,(1, 10{:}35) \cdot \texttt{Fault}_2\,(2, 10{:}40) \cdot \texttt{Fault}_3\,(3, 10{:}55) \cdot \texttt{Fix}_4\,(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$.

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

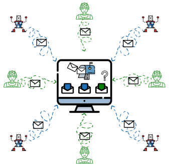
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1\,(1, 10{:}35) \cdot \texttt{Fault}_2\,(2, 10{:}40) \cdot \texttt{Fault}_3\,(3, 10{:}55) \cdot \texttt{Fix}_4\,(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

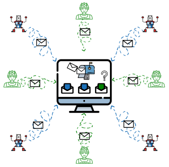
$$\Pi_2 = \texttt{Fault}(\_, t_1) \land \texttt{Fault}(id_2, t_2) \land \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**
  - $\Pi_1 : \langle\{\texttt{Fault}_3, \texttt{Fix}_4\}\rangle$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

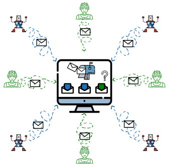
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**
    - $\Pi_1 : \langle\{\texttt{Fault}_3, \texttt{Fix}_4\}\rangle$
    - $\Pi_2 : \langle\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\}\rangle$

Join Patterns Matching



The join definition for the factory shop floor monitor is $D = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

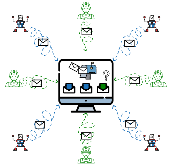
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1\,(1, 10{:}35) \cdot \texttt{Fault}_2\,(2, 10{:}40) \cdot \texttt{Fault}_3\,(3, 10{:}55) \cdot \texttt{Fix}_4\,(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**
  - $\Pi_1 : \langle\{\texttt{Fault}_3, \texttt{Fix}_4\}\rangle$
  - $\Pi_2 : \langle\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\}\rangle$
- In existing literature, the selection is either
  - **Non-deterministic** choice. This is usually undesirable
  - Pick **longest-matching sequence**

Our Proposal: “Fair Match”



Recall that we have the following $\mathsf{D} = \Pi_1 + \Pi_2$ where:

$$\Pi_1 = \mathtt{Fault}(id_1, \_) \land \mathtt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

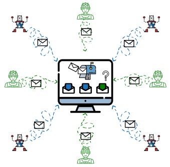
$$\Pi_2 = \mathtt{Fault}(\_, t_1) \land \mathtt{Fault}(id_2, t_2) \land \mathtt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

And the following final mailbox configuration:

$$\mathcal{M} = \mathtt{Fault}_1(1, 10{:}35) \cdot \mathtt{Fault}_2(2, 10{:}40) \cdot \mathtt{Fault}_3(3, 10{:}55) \cdot \mathtt{Fix}_4(3, 11{:}00)$$

- A “fair” match is the one that consumes the **oldest** messages in $\mathcal{M}$
- No message that can be matched is left in the mailbox **indefinitely**

Our Proposal: “Fair Match”



Recall that we have the following $D = \Pi_1 + \Pi_2$ where:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

And the following final mailbox configuration:

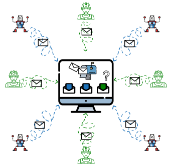
$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- A “fair” match is the one that consumes the **oldest** messages in $\mathcal{M}$
- No message that can be matched is left in the mailbox **indefinitely**
- Now we can pick the **fairest** match from $\mathcal{M}$:

$$\Pi_1 : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

$$\Pi_2 : \langle \{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

Our Proposal: "Fair Match"



Recall that we have the following $\mathsf{D} = \Pi_1 + \Pi_2$ where:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

And the following final mailbox configuration:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- A "fair" match is the one that consumes the **oldest** messages in $\mathcal{M}$
- No message that can be matched is left in the mailbox **indefinitely**
- Now we can pick the **fairest** match from $\mathcal{M}$:

$$\Pi_1 : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

$$\Pi_2 : \langle \{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

$$\mathsf{D} : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

“Fair” Match Formalisation

We have formalised this notion of “fair” join pattern matching declaratively using inference rules:

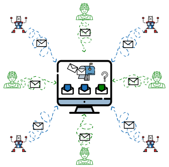
$$\frac{\forall i \in \{1,\ldots,n\} : \mu_i\sigma = m_i \qquad \gamma\sigma}{m_1 \cdot \ldots \cdot m_n \ \vDash_\sigma \ \mu_1 \wedge \ldots \wedge \mu_n \text{ if } \gamma} \ \textbf{Match Messages Against Pattern}$$

$$\frac{\mathcal{M}[\mathcal{I}] \vDash_\sigma \Pi \text{ for some } \sigma}{\mathcal{M} \vDash_{\mathcal{I}} \Pi} \ \textbf{Pick Messages From } \mathcal{M}$$

$$\frac{\mathcal{M} \vDash_{\mathcal{I}} \Pi \qquad \forall \mathcal{I}'. \, (\mathcal{M} \vDash_{\mathcal{I}'} \Pi \implies \mathcal{I} \leqslant_{\text{lex}} \mathcal{I}')}{\mathcal{M} \vDash \Pi \rightsquigarrow \mathcal{I}} \ \textbf{Select Fairest Match}$$

- Translate inference rules into a “fair” message matching **brute-force algorithm**
- Current implementations use matching without fairness e.g. (Haller et al. COORDINATION 2008, Plociniczak and Eisenbach COORDINATION 2010, Avila et al. 2020)
- Refer to the paper for more details

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts

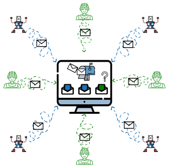
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} =$$

Brute-force Algorithm for “Fair” Message Matching



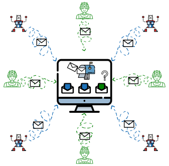
Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_)\cdot$$

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(\mathit{id_1}, \_) \land \texttt{Fix}(\mathit{id_2}, \_) \text{ if } \mathit{id_1} = \mathit{id_2}$$

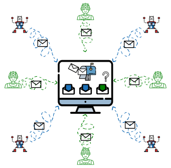
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(\mathit{3}, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(\mathit{3}, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

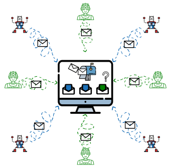
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_)\rangle$ – Not enough messages ×

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

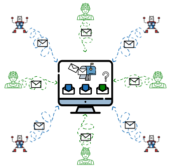
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_)\rangle$ – Not enough messages ×

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

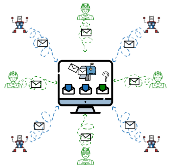
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_)\rangle$ – Not enough messages ×
$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_)\rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(\mathit{id_1}, \_) \land \texttt{Fix}(\mathit{id_2}, \_) \text{ if } \mathit{id_1} = \mathit{id_2}$$

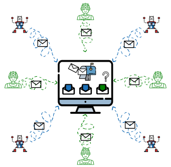
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(\mathit{3}, \_) \cdot \texttt{Fault}_2\,(\mathit{1}, \_) \cdot \texttt{Fault}_3\,(\mathit{2}, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(\mathit{3}, \_) \rangle$ – Not enough messages ✗
$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(\mathit{3}, \_) \cdot \texttt{Fault}_2\,(\mathit{1}, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

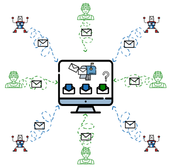
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_)\rangle$ – Not enough messages ×
$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_)\rangle$
$\mathcal{M}[1 \cdot 2 \cdot 3]$:

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

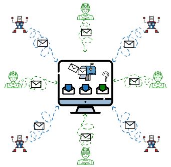
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ×
$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$
$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts

We have that Π_1:

$$\Pi_1 = \mathtt{Fault}(id_1, \_) \wedge \mathtt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \mathtt{Fix}_1\,(3, \_) \cdot \mathtt{Fault}_2\,(1, \_) \cdot \mathtt{Fault}_3\,(2, \_) \cdot$$

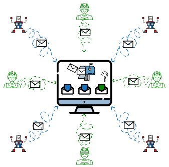
- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \mathtt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ✗

$\mathcal{M}[1 \cdot 2]$: $\langle \mathtt{Fix}_1\,(3, \_) \cdot \mathtt{Fault}_2\,(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \mathtt{Fix}_1\,(3, \_) \cdot \mathtt{Fault}_2\,(1, \_) \rangle$, $\langle \mathtt{Fix}_1\,(3, \_) \cdot \mathtt{Fault}_3\,(2, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts

We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot \texttt{Fault}_3(2, \_) \cdot \texttt{Fault}_4(3, \_)$$

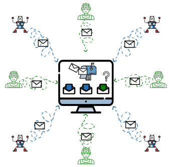
- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ✗

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$, $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_3(2, \_) \rangle$

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts

We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot \texttt{Fault}_4\,(3, \_)$$

- Find a match for Π_1 from $\mathcal{M}$

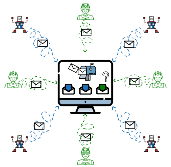
$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ✗

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3 \cdot 4]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_) \rangle$,

Brute-force Algorithm for “Fair” Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \cdot \texttt{Fault}_3\,(2, \_) \cdot \texttt{Fault}_4\,(3, \_)$$

- Find a match for Π_1 from $\mathcal{M}$

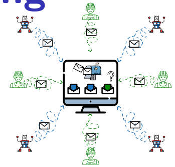
$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_) \rangle$ – Not enough messages ✗

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3 \cdot 4]$: $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_2\,(1, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_3\,(2, \_) \rangle$, $\langle \texttt{Fix}_1\,(3, \_) \cdot \texttt{Fault}_4\,(3, \_) \rangle$

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

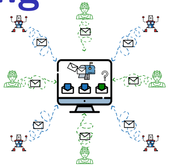
and the following mailbox:

$$\mathcal{M} =$$

$\emptyset$

Check if $id_1 = id_2$

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

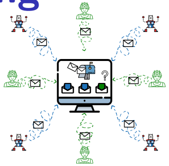
and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot$$

$\emptyset$

Check if $id_1 = id_2$

Stateful Tree-based Algorithm for "Fair" Message Matching



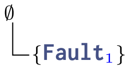
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

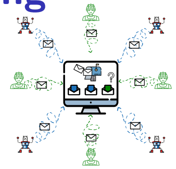
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot$$



Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

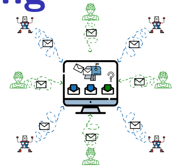
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot$$

$$\begin{array}{l} \emptyset \\ \quad\llcorner\{\texttt{Fault}_1\} \end{array}$$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

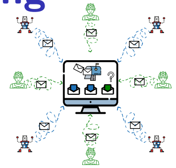
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot$$

$\emptyset$
- $\{\texttt{Fault}_1\}$
- $\{\texttt{Fault}_2\}$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for “Fair” Message Matching



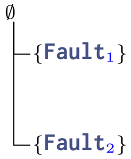
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

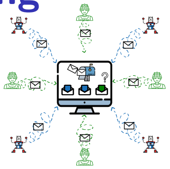
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot$$



Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for “Fair” Message Matching



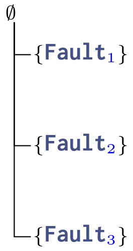
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

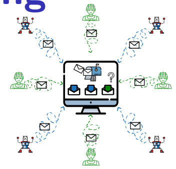
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot$$



Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for "Fair" Message Matching



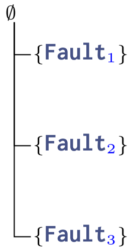
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

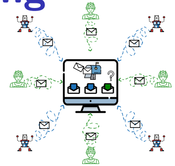
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$



Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for “Fair” Message Matching



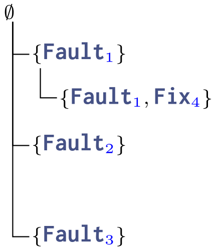
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

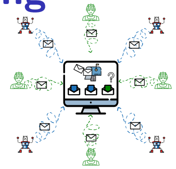
and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$



Check if $id_1 = id_2$

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

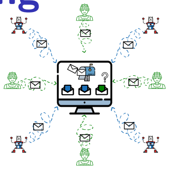
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

```
∅
├─{Fault₁}
│  └─{Fault₁, Fix₄}
├─{Fault₂}
│
└─{Fault₃}
```

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$

Stateful Tree-based Algorithm for “Fair” Message Matching



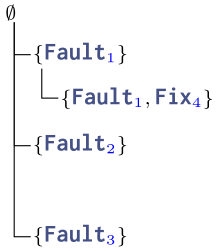
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

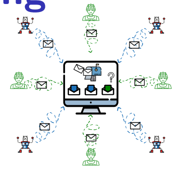
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ✗

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

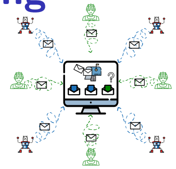
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_2\}$
  - $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ $\times$

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

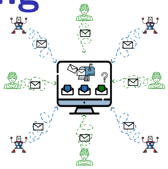
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_2\}$
    - $\{\texttt{Fault}_2, \texttt{Fix}_4\}$
  - $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×

Stateful Tree-based Algorithm for “Fair” Message Matching



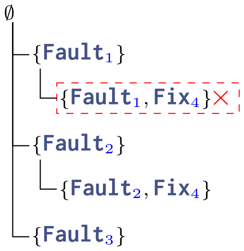
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

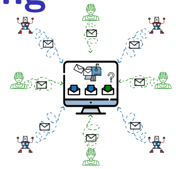


$\emptyset$
$\{\texttt{Fault}_1\}$
$\{\texttt{Fault}_1, \texttt{Fix}_4\} \times$
$\{\texttt{Fault}_2\}$
$\{\texttt{Fault}_2, \texttt{Fix}_4\}$
$\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ $\times$
- **Attempt 2**: $2 \neq 3$

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

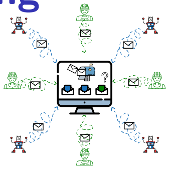
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\{\texttt{Fault}_1, \texttt{Fix}_4\}$ ×
  - $\{\texttt{Fault}_2\}$
    - $\{\texttt{Fault}_2, \texttt{Fix}_4\}$
  - $\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×

Stateful Tree-based Algorithm for “Fair” Message Matching



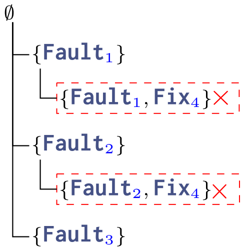
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

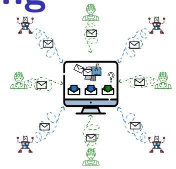
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$


$\emptyset$
$\{\texttt{Fault}_1\}$
$\{\texttt{Fault}_1, \texttt{Fix}_4\}$ ×
$\{\texttt{Fault}_2\}$
$\{\texttt{Fault}_2, \texttt{Fix}_4\}$ ×
$\{\texttt{Fault}_3\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

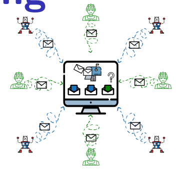
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\{\texttt{Fault}_1, \texttt{Fix}_4\}$ ×
  - $\{\texttt{Fault}_2\}$
    - $\{\texttt{Fault}_2, \texttt{Fix}_4\}$ ×
  - $\{\texttt{Fault}_3\}$
    - $\{\texttt{Fault}_3, \texttt{Fix}_4\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×

Stateful Tree-based Algorithm for “Fair” Message Matching



Use state to track **partial matches** and avoid redundant matching attempts
Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

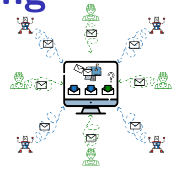
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\boxed{\{\texttt{Fault}_1, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_2\}$
    - $\boxed{\{\texttt{Fault}_2, \texttt{Fix}_4\} \times}$
  - $\{\texttt{Fault}_3\}$
    - $\{\texttt{Fault}_3, \texttt{Fix}_4\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$

Stateful Tree-based Algorithm for “Fair” Message Matching



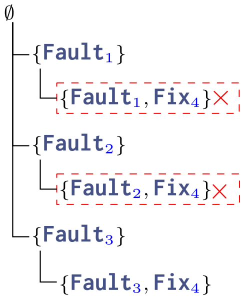
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \land \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

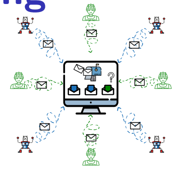
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$


$\emptyset$
$\{\texttt{Fault}_1\}$
$\{\texttt{Fault}_1, \texttt{Fix}_4\}$ ×
$\{\texttt{Fault}_2\}$
$\{\texttt{Fault}_2, \texttt{Fix}_4\}$ ×
$\{\texttt{Fault}_3\}$
$\{\texttt{Fault}_3, \texttt{Fix}_4\}$

Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$ ✓

Stateful Tree-based Algorithm for “Fair” Message Matching



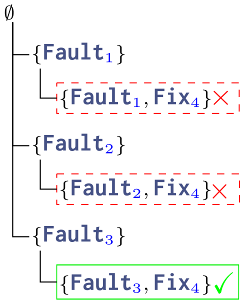
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

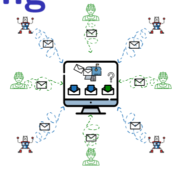
$$\mathcal{M} = \texttt{Fault}_1\,(1, \_) \cdot \texttt{Fault}_2\,(2, \_) \cdot \texttt{Fault}_3\,(3, \_) \cdot \texttt{Fix}_4\,(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ✗
- **Attempt 2**: $2 \neq 3$ ✗
- **Attempt 3**: $3 = 3$ ✓

Stateful Tree-based Algorithm for “Fair” Message Matching



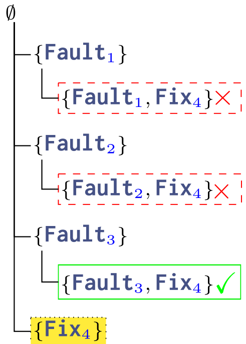
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \mathtt{Fault}(id_1, \_) \wedge \mathtt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \mathtt{Fault}_1\,(1, \_) \cdot \mathtt{Fault}_2\,(2, \_) \cdot \mathtt{Fault}_3\,(3, \_) \cdot \mathtt{Fix}_4\,(3, \_)$$

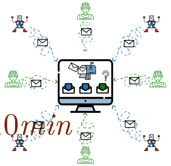


Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$ ✓

We don’t record a partial match $\mathtt{Fix}_4$ because we matched earlier

Tree Construction (continued)



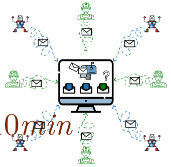
We now consider the second join pattern Π_2:

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \ \&\& \ t_2 - t_1 > 10min$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

Tree Construction (continued)



We now consider the second join pattern Π_2:

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- $\emptyset$
  - $\{\texttt{Fault}_1\}$
    - $\{\texttt{Fault}_1, \texttt{Fault}_2\}$
      - $\{\texttt{Fault}_1, \texttt{Fault}_2, \texttt{Fix}_4\}$ ✗
    - $\{\texttt{Fault}_1, \texttt{Fault}_3\}$
      - $\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}$ ✓
  - $\{\texttt{Fault}_2\}$
    - $\{\texttt{Fault}_2, \texttt{Fault}_3\}$
      - $\{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\}$
  - $\{\texttt{Fault}_3\}$
    - $\{\texttt{Fault}_3, \texttt{Fix}_4\}$
  - $\{\texttt{Fix}_4\}$

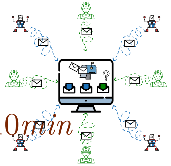
Check if $id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$:

- **Attempt 1**:

  $1 = 3 \;\&\&\; 10:40 - 10:35 > 10min$ ✗
- **Attempt 2**:

  $3 = 3 \;\&\&\; 10:55 - 10:35 > 10min$ ✓

Tree Construction (continued)

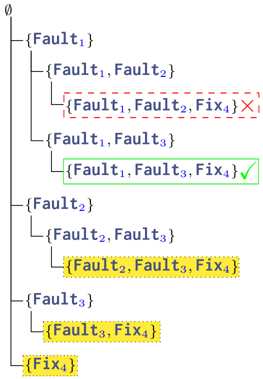


We now consider the second join pattern Π_2:

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$


$\emptyset$
$\{\texttt{Fault}_1\}$
$\{\texttt{Fault}_1, \texttt{Fault}_2\}$
$\{\texttt{Fault}_1, \texttt{Fault}_2, \texttt{Fix}_4\}$ ✗
$\{\texttt{Fault}_1, \texttt{Fault}_3\}$
$\{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}$ ✓
$\{\texttt{Fault}_2\}$
$\{\texttt{Fault}_2, \texttt{Fault}_3\}$
$\{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\}$
$\{\texttt{Fault}_3\}$
$\{\texttt{Fault}_3, \texttt{Fix}_4\}$
$\{\texttt{Fix}_4\}$

Check if $id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$:

- **Attempt 1**:

  $1 = 3 \;\&\&\; 10:40 - 10:35 > 10min$ ✗

- **Attempt 2**:

  $3 = 3 \;\&\&\; 10:55 - 10:35 > 10min$ ✓

We avoid computing (partial) matches

Performance Evaluation

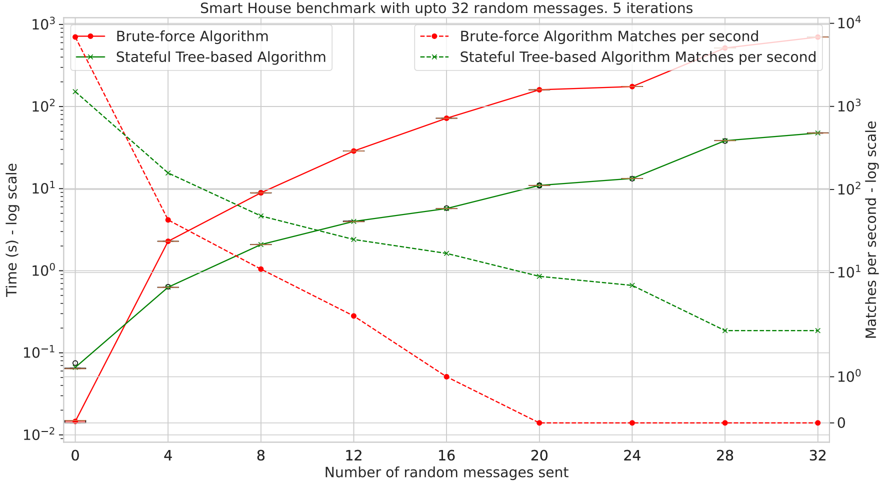


Figure: Smart House benchmark based on (Rodriguez-Avila et al. 2021)

Contributions & Future Work

Contributions:

- Novel specification of **fair and deterministic join pattern matching**
- Novel **stateful tree-based matching algorithm** to avoid redundant recomputations
- **Proof of correctness** of the stateful fair matching algorithm
- `JoinActors`: novel Scala 3 library with brute-force & stateful matching
- Established a **benchmark suite** to evaluate join pattern matching performance

Contributions & Future Work

Contributions:

- Novel specification of **fair and deterministic join pattern matching**
- Novel **stateful tree-based matching algorithm** to avoid redundant recomputations
- **Proof of correctness** of the stateful fair matching algorithm
- `JoinActors`: novel Scala 3 library with brute-force & stateful matching
- Established a **benchmark suite** to evaluate join pattern matching performance

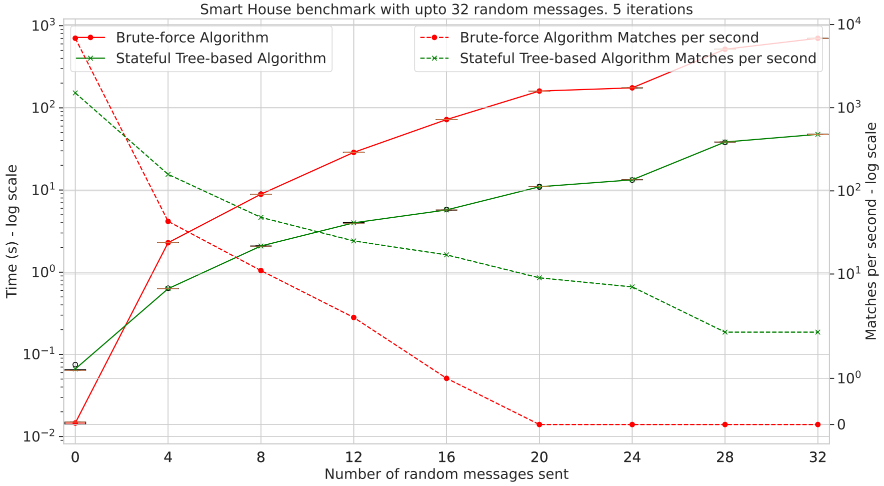
Future Work:

- Expand benchmark suite with more examples from the literature
- Refine and optimise the Scala 3 implementation of join patterns
- Alternative matching policies
- Verify join pattern unreachablity

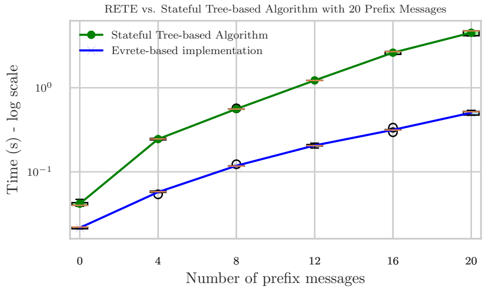
Smart House Example (Rodriguez-Avila et al. 2021) I

```
case (Motion(_, mStatus, mRoom, t0),
      AmbientLight(_, value, alRoom, t1),
      Light(_, lStatus, lRoom, t2)) if bathroomOccupied(...) => ...

case (Motion(_, mStatus0, mRoom0, t0),
      Contact(_, cStatus, cRoom, t1),
      Motion(_, mStatus1, mRoom1, t2)) if occupiedHome(...) => ...

case (Motion(_, mStatus0, mRoom0, t0),
      Contact(_, cStatus, cRoom, t1),
      Motion(_, mStatus1, mRoom1, t2)) if emptyHome(...) => ...
```

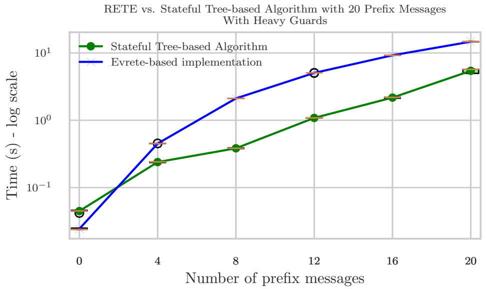
Smart House Example (Rodriguez-Avila et al. 2021) II



JoinActors vs. Evrete Benchmark



JoinActors vs. Evrete (lower is better)



JoinActors vs. Evrete (lower is better)

- Evrete is a mature and highly optimised RETE-based Java rule engine library
- JoinActors is our proof-of-concept Scala 3 actor library

Join Patterns Implementation in Scala 3

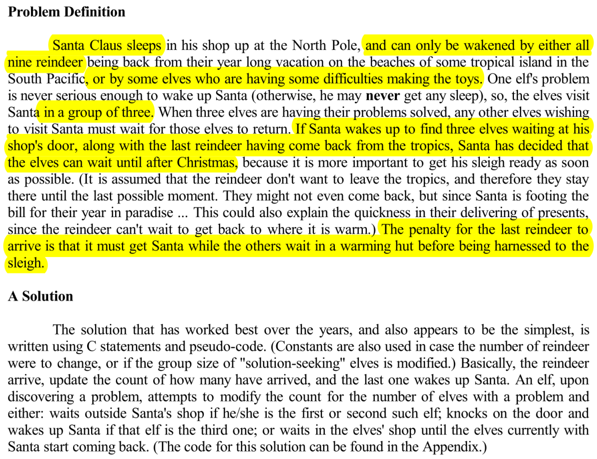
```scala
inline def receive[M, T](
    inline f: ActorRef[M] => PartialFunction[Any, Result[T]]
): MatchingAlgorithm => Matcher[M, Result[T]]
```

Macro Expansion & Code Transformation

The body of `receive`:

```scala
...
expr.asTerm match
    case Inlined(_, _, Block(_, Block(stmts, _))) =>
      stmts.head match
        case DefDef(_, List(TermParamClause(params)), _, Some(Block(_,
          ↪ Block(body, _)))) =>
          body.head match
            case DefDef(_, _, _, Some(Match(_, cases))) =>
              cases.flatMap { generateJoinPattern[M, T](_) }
...
```

A problem in concurrency [Tro94]



Problem Definition

Santa Claus sleeps in his shop up at the North Pole, and can only be wakened by either all nine reindeer being back from their year long vacation on the beaches of some tropical island in the South Pacific, or by some elves who are having some difficulties making the toys. One elf's problem is never serious enough to wake up Santa (otherwise, he may **never** get any sleep), so, the elves visit Santa in a group of three. When three elves are having their problems solved, any other elves wishing to visit Santa must wait for those elves to return. If Santa wakes up to find three elves waiting at his shop's door, along with the last reindeer having come back from the tropics, Santa has decided that the elves can wait until after Christmas, because it is more important to get his sleigh ready as soon as possible. (It is assumed that the reindeer don't want to leave the tropics, and therefore they stay there until the last possible moment. They might not even come back, but since Santa is footing the bill for their year in paradise ... This could also explain the quickness in their delivering of presents, since the reindeer can't wait to get back to where it is warm.) The penalty for the last reindeer to arrive is that it must get Santa while the others wait in a warming hut before being harnessed to the sleigh.

A Solution

The solution that has worked best over the years, and also appears to be the simplest, is written using C statements and pseudo-code. (Constants are also used in case the number of reindeer were to change, or if the group size of "solution-seeking" elves is modified.) Basically, the reindeer arrive, update the count of how many have arrived, and the last one wakes up Santa. An elf, upon discovering a problem, attempts to modify the count for the number of elves with a problem and either: waits outside Santa's shop if he/she is the first or second such elf; knocks on the door and wakes up Santa if that elf is the third one; or waits in the elves' shop until the elves currently with Santa start coming back. (The code for this solution can be found in the Appendix.)

```
receive
    {reindeer, Pid1} and {reindeer, Pid2} and {reindeer, Pid3}
      and {reindeer, Pid4} and {reindeer, Pid5} and {reindeer, Pid6}
      and {reindeer, Pid7} and {reindeer, Pid8} and {reindeer, Pid9} ->
        io:format("Ho, ho, ho! Let’s deliver presents!~n"),
        [Pid1, Pid2, Pid3, Pid4, Pid5, Pid6, Pid7, Pid8, Pid9];
    {elf, Pid1} and {elf, Pid2} and {elf, Pid3} ->
        io:format("Ho, ho, ho! Let’s discuss R&D possibilities!~n"),
        [Pid1, Pid2, Pid3]
end
```

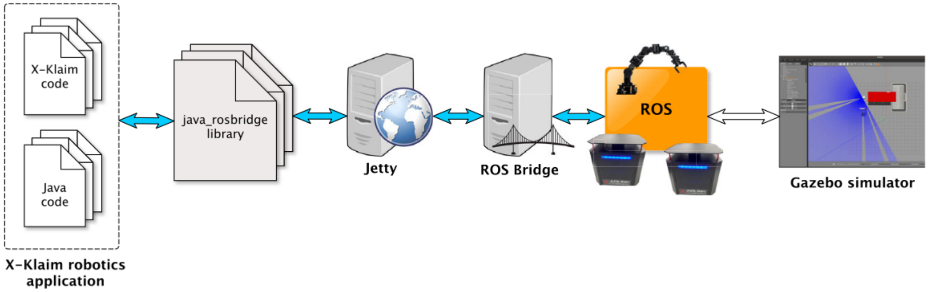
The solution with semaphores takes about 2 pages of C code [Tro94]!

Applying Concurrency with generative communication [CG89]

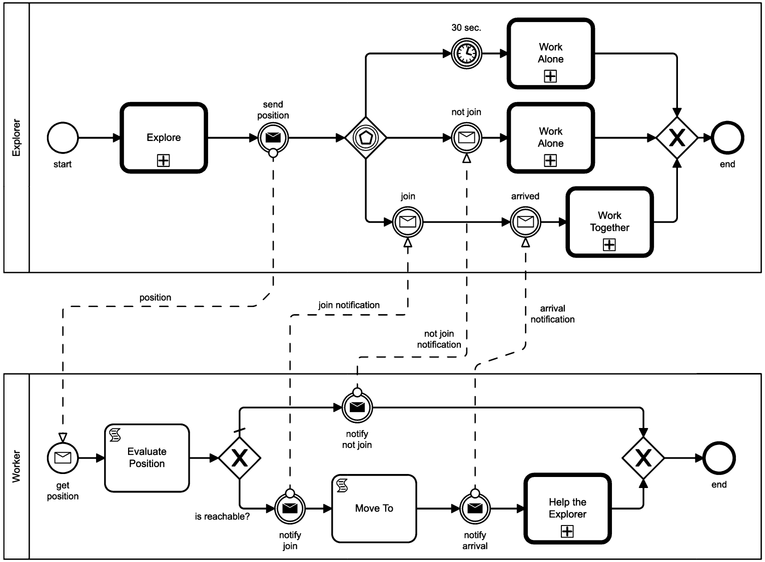
A model-driven approach for multi-robots missions [BTBS26]

Multi-robot application are complex: robots' interactions are "low-level"

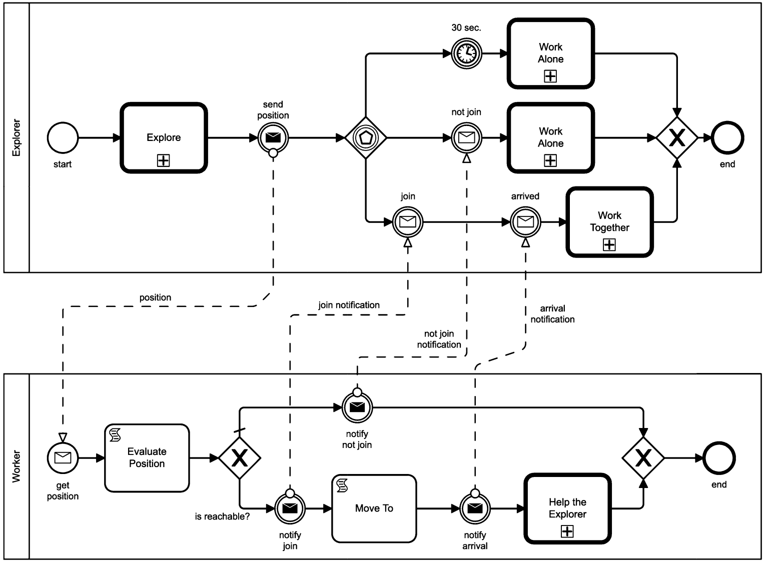
Model-driven development based on BPMN and X-KLAIM lowers barries



Business Process Modelling Notation [BTBS26]

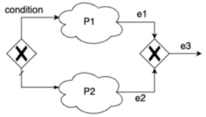


Business Process Modelling Notation



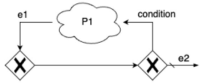
From BPMN to X-Klaim [BTBS26]

XOR



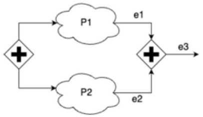
```
if(condition){
  translate(P1)
  in(e1)@self }
else{
  translate(P2)
  in(e2)@self }
out(e3)@self
```

Loop



```
while(condition){
  translate(P1)
  in(e1)@self }
out(e2)@self
```

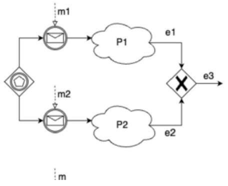
AND



```
eval(new ProcP1())@self
eval(new ProcP2())@self
in(e1)@self
in(e2)@self
out(e3)@self
```

```
// Processes to be
// added to the node
proc ProcP1(){
  translate(P1)
}

proc ProcP2(){
  translate(P2)
}
```

Event-Based (between messages)



```
var eventOccured = false
while(!eventOccured){
  if(in(m1,vars1)@self within pollTimeout){
    eventOccured = true
    translate(P1)
    in(e1)@self }
  else if(in(m2,vars2)@self within pollTimeout){
    eventOccured = true
    translate(P2)
    in(e2)@self } }
out(e3)@self
```

Klaim

Network-aware programming and generative communication:
X-KLAIM: eXtended Kernel Language for Agents Interaction and Mobility

Network

```
net MRS {
node Drone { eval(new DroneBehavior(Tractor)) @ self }
node Tractor { eval(new TractorBehavior()) @ self }
}
```

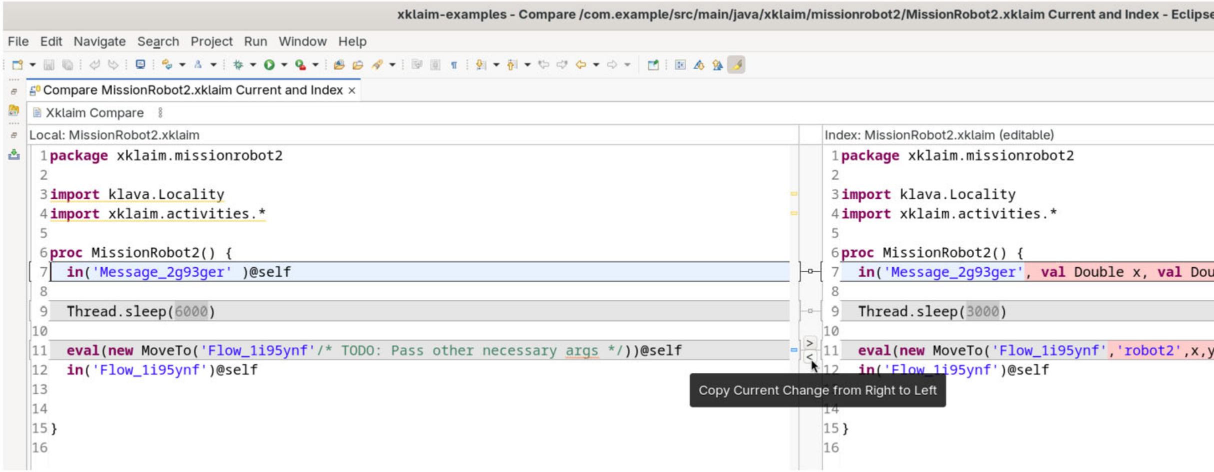
where

Some processes

```
proc DroneBehavior(Locality Tractor) {
eval(new WeedHandler(Tractor)) @ self
eval(new TakeOff("e1"))tractor @ self
in("e1") @ self
eval(new Explore("e2")) @ self
in("e2") @ self
eval(new Land("e3")) @ self
in("e3") @ self
}
```

```
proc TractorBehavior() {
in(WEED_POSITION, var Double x, var Double y) @ self
eval(new MoveTo("e4", x, y)) @ self
in("e4") @ self
eval(new CutGrass("e5")) @ self
in("e5") @ self
eval(new ReturnToBase("e6")) @ self
in("e6") @ self
```

Programming support



xklaim-examples - Compare /com.example/src/main/java/xklaim/missionrobot2/MissionRobot2.xklaim Current and Index - Eclipse

Compare MissionRobot2.xklaim Current and Index

Local: MissionRobot2.xklaim

```
package xklaim.missionrobot2

import klava.Locality
import xklaim.activities.*

proc MissionRobot2() {
  in('Message_2g93ger' )@self

  Thread.sleep(6000)

  eval(new MoveTo('Flow_1i95ynf'/* TODO: Pass other necessary args */))@self
  in('Flow_1i95ynf')@self



}

```

Index: MissionRobot2.xklaim (editable)

```
package xklaim.missionrobot2

import klava.Locality
import xklaim.activities.*

proc MissionRobot2() {
  in('Message_2g93ger', val Double x, val Dou

  Thread.sleep(3000)

  eval(new MoveTo('Flow_1i95ynf','robot2',x,y
  in('Flow_1i95ynf')@self



}

```

[BTBS26] Khalid Bourr, Francesco Tiezzi, Lorenzo Bettini, and Stefano Seriani. Translating bpmn models into x-klaim programs for developing multi-robot missions. *International Journal on Software Tools for Technology Transfer*, pages 1433–2787, January 2026.

[CG89] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[Eck02] Bruce Eckel. *Thinking in Java, 4rd Edition*. Prentice-Hall, December 2002. Chapter 13. The beta version of the 3rd edition is available at `http://www.javaclue.org/pub/java/ebooks/tij/tij-3rd-ed.pdf`.

[FG96] Cedric Fournet and George Gonthier. The reflexive CHAM and the join-calculus. In *Conference Record of POPL '96: The* 23rd *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 372–385, St. Petersburg Beach, Florida, January 1996.

[HHM+24a] Philipp Haller, Ayman Hussein, Hernán C. Melgratti, Alceste Scalas, and Emilio Tuosto. Fair join pattern matching for actors.

In Jonathan Aldrich and Guido Salvaneschi, editors, *38th European Conference on Object-Oriented Programming, ECOOP 2024, Vienna, Austria, September 16-20, 2024*, volume 313 of *LIPIcs*, pages 17:1–17:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[HHM$^+$24b] Philipp Haller, Ayman Hussein, Hernán C. Melgratti, Alceste Scalas, and Emilio Tuosto. Fair join pattern matching for actors (artifact). *Dagstuhl Artifacts Ser.*, 10(2):8:1–8:3, 2024.

[Tro94] John A. Trono. A new exercise in concurrency. *SIGCSE Bull.*, 26(3):8–10, September 1994.