Scalable Performance for Scala Message-Passing Concurrency

Andrew Bate

Department of Computer Science
University of Oxford

UNIVERSITY OF OXFORD

cso.io

Motivation

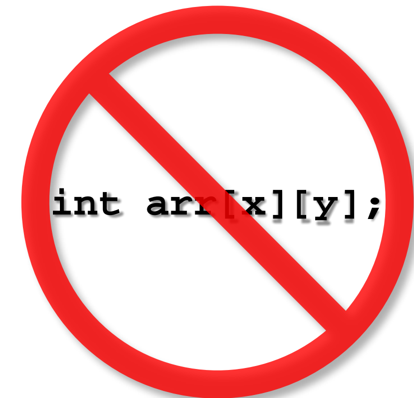
Multi-core commodity hardware

Non-uniform shared memory

Expose potential parallelism

Correctness and formal verification

Compatibility



1. Embedded DSL
2. Bytecode rewriting
3. Channels
4. Scheduler
5. Deadlock detection

EMBEDDED DOMAIN-SPECIFIC LANGUAGE

Why an Embedded DSL?

Ease of implementation

Leverage existing tools

Leverage known syntax

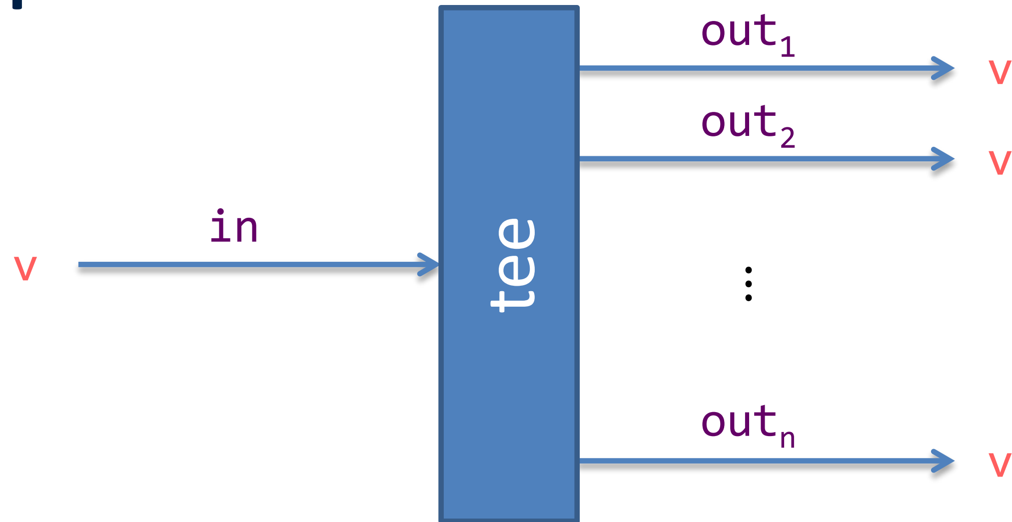
Higher-order functions
Rich type system

Lightweight syntax
Compile-time macros

Examples

```
def map[I, O](f: I => O)(in: ?[I], out: ![O]) =
  proc {
    repeat { out ! (f(in?)) }
    run (proc { in.closein } || proc { out.closeout })
  }
```

Examples



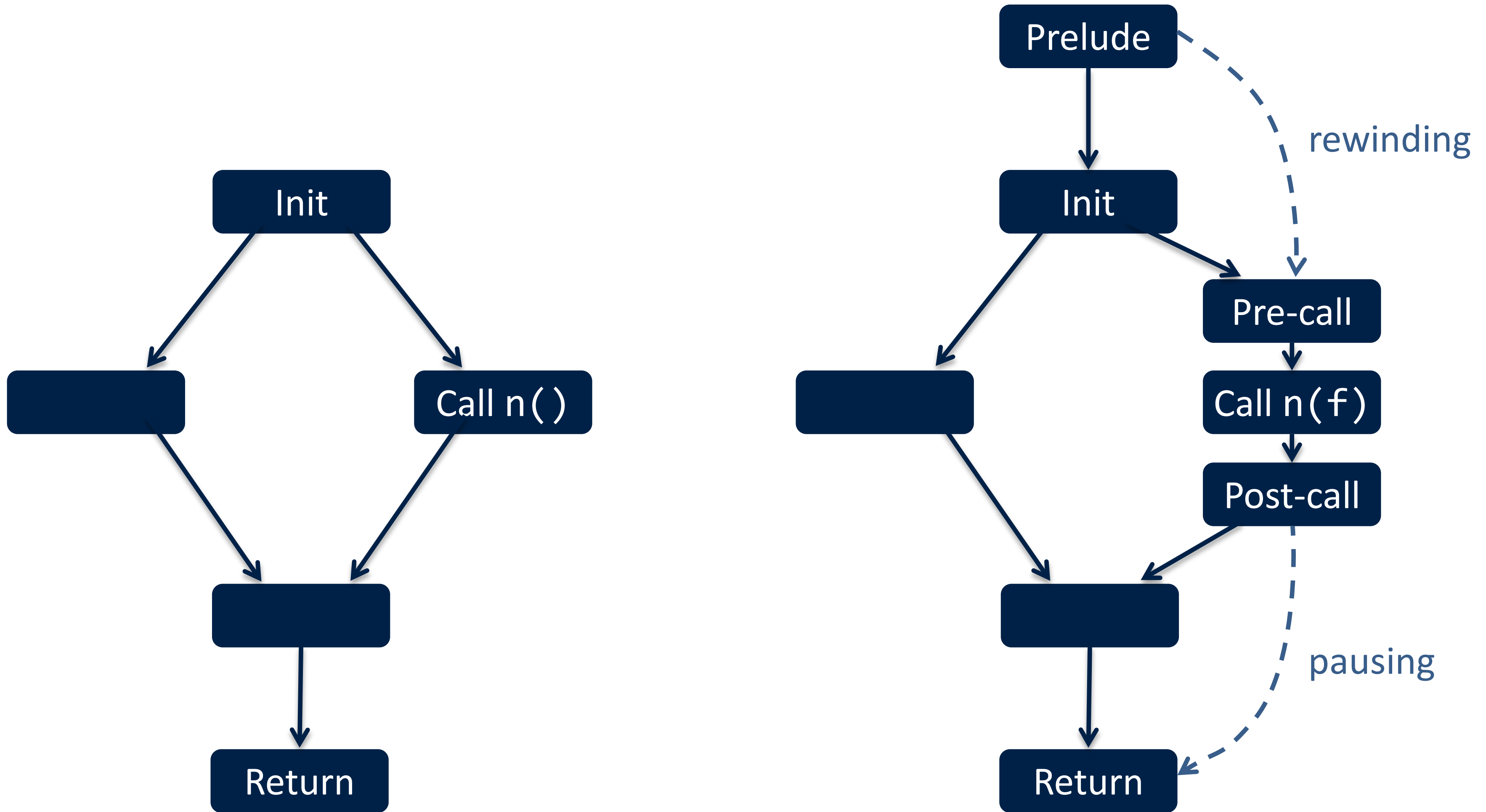

```
def tee[@specialized T](in: ?[T], outs: Seq[![T]]) =
  proc {
    var v = null
    val outputs = (|| (out <- outs) proc { out ! v })
    repeat { v = in?; run outputs }
    run (proc { in.closein } || (|| (out <- outs) proc { out.closeout }))
  }
```

1. Embedded DSL
2. Bytecode rewriting
3. Channels
4. Scheduler
5. Deadlock detection

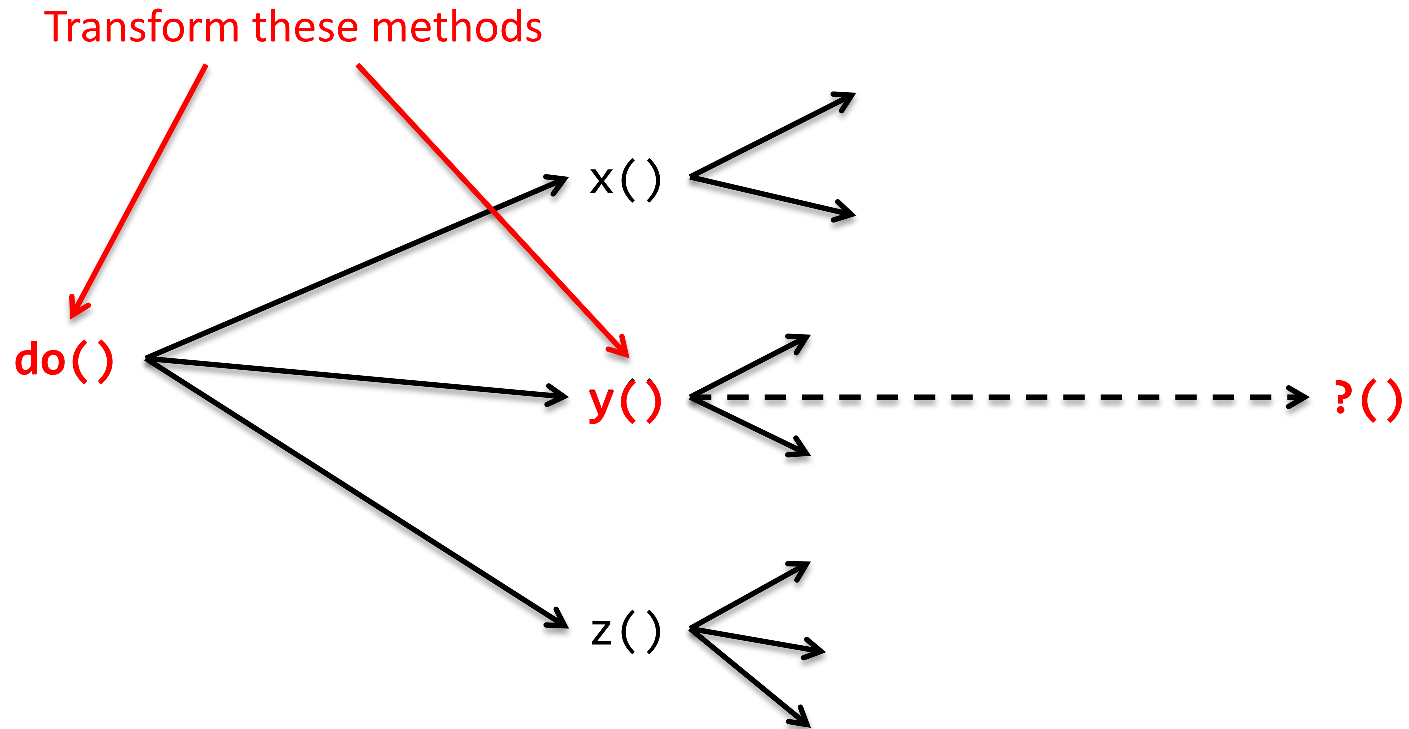
SCALABLE PERFORMANCE

through bytecode rewriting

CPS Transformation



Analysing the call graph



Engineering

Live variable analysis

Lazy load and store

Constant inlining

Functional Expressions

```
for (i <- 0 until n; j <- i until n) println(i)
```

```
intWrapper(0).until(n).foreach(
  i: Int => intWrapper(i).until(n).foreach(j: Int => println(i))
)
```

```
var i = 0
while (i < n) {
  var j = i
  while (j < n) { println(i); j += 1 }
  i += 1
}
```

More Features

Tail call optimisations

Shared memory

SBT plugin support

1. Embedded DSL
2. Bytecode rewriting
3. Channels
4. Scheduler
5. Deadlock detection

CHANNELS

More Features

Generalised alt

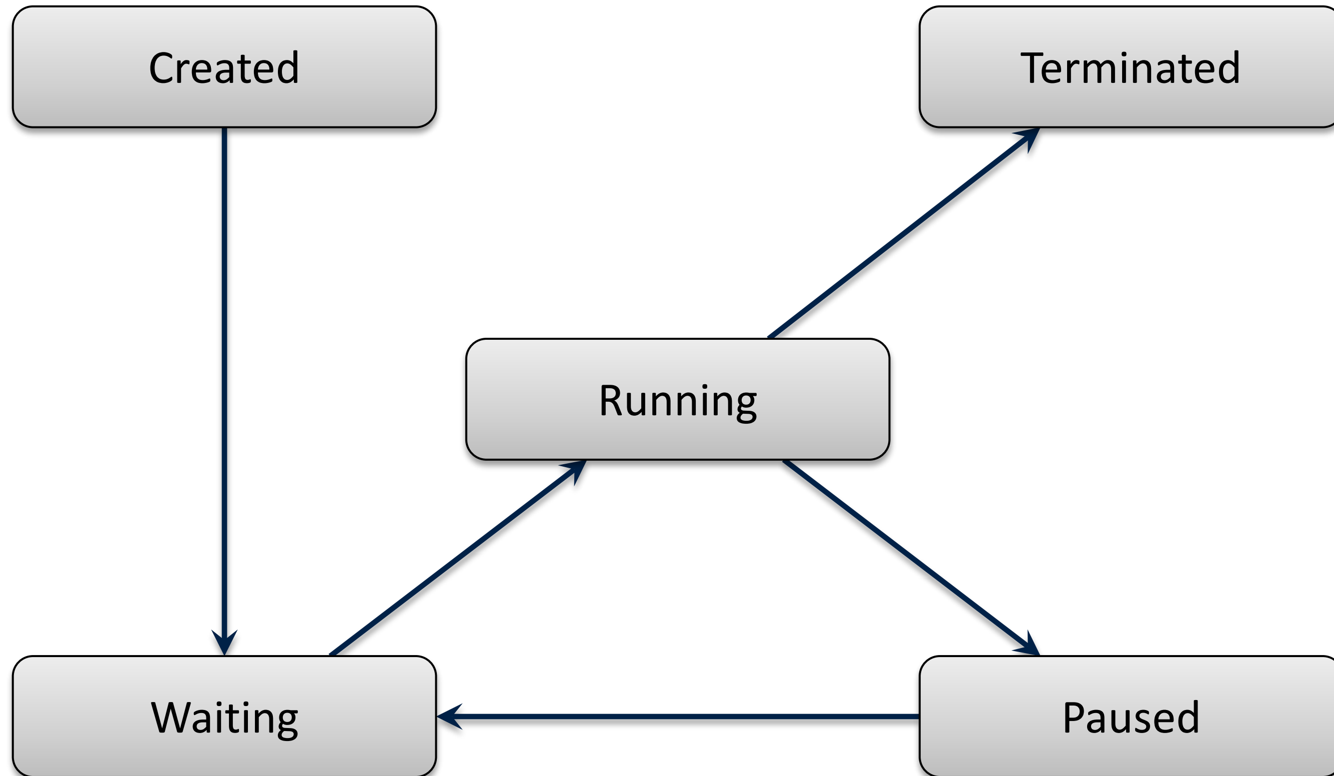
Specialization for primitives

Optimised extended rendezvous

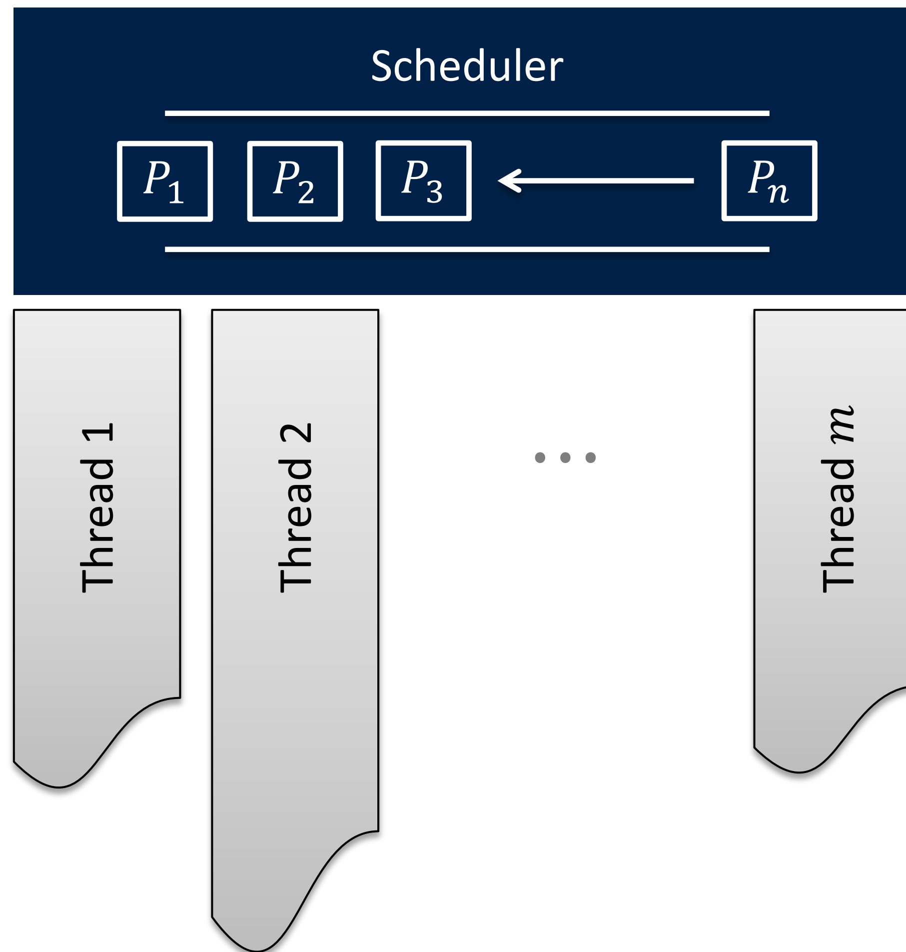
1. Embedded DSL
2. Bytecode rewriting
3. Channels
4. Scheduler
5. Deadlock detection

SCHEDULER

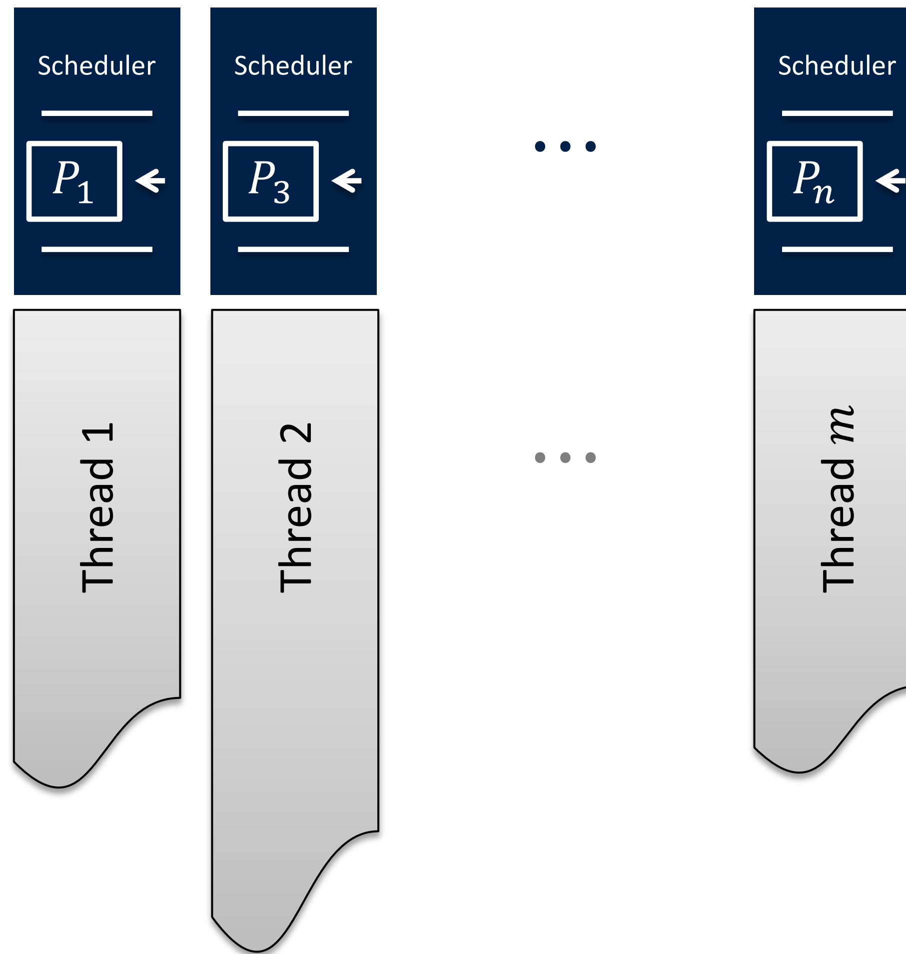
Scheduler States



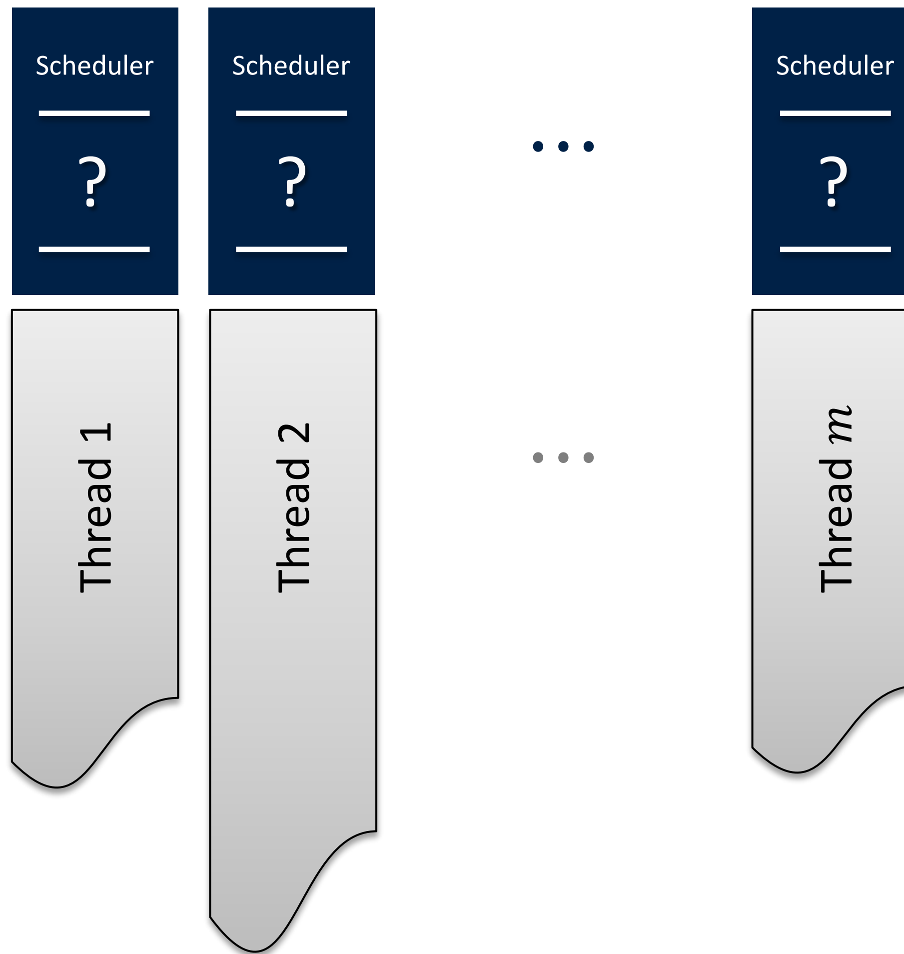
Scheduling: Central FIFO



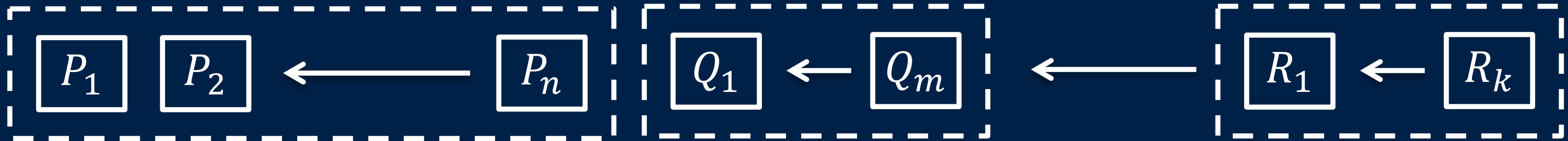
Scheduling: FIFO per thread



Scheduling: Batches per thread



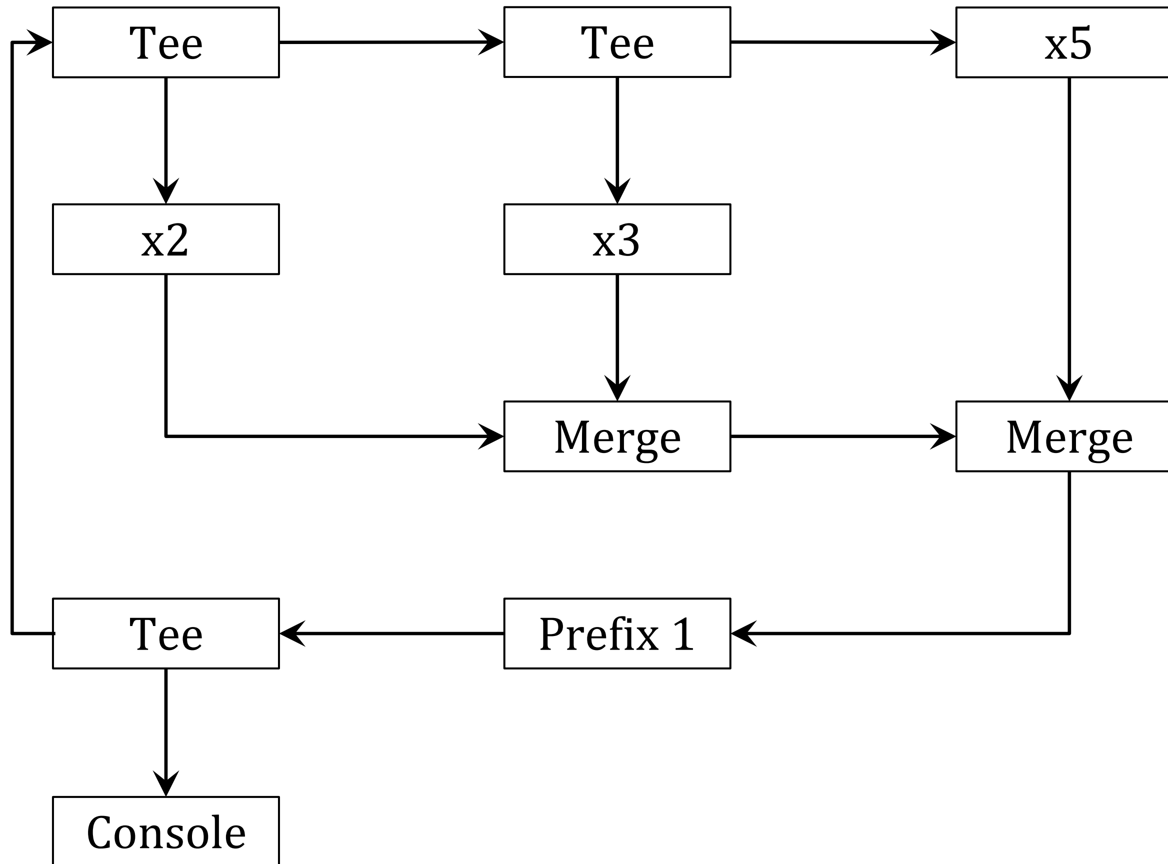
Scheduling: Batches per thread



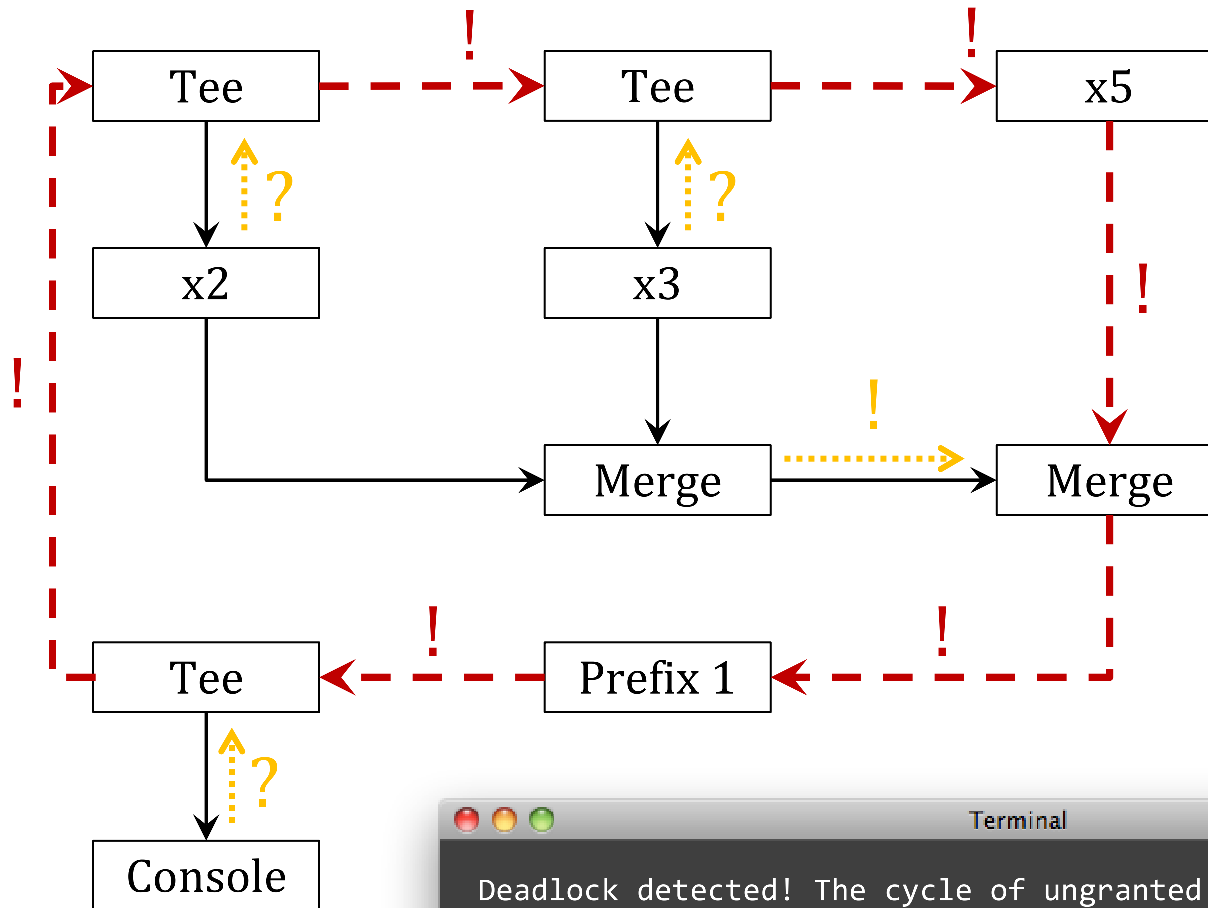
1. Embedded DSL
2. Bytecode rewriting
3. Channels
4. Scheduler
5. **Deadlock detection**

DEADLOCK DETECTION

Example



Example



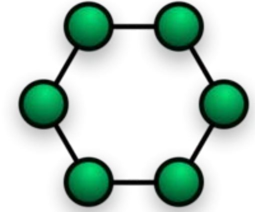
Terminal

```
Deadlock detected! The cycle of ungranted requests is:
Prefix1 -!-> Tee1        Tee3   -!-> x5
Tee1    -!-> Tee2        x5     -!-> Merge2
Tee2    -!-> Tee3        Merge2 -!-> Prefix1
```

1. Embedded DSL
2. Bytecode rewriting
3. Channels
4. Scheduler
5. Deadlock detection

PERFORMANCE EVALUATION

Ring topology



Java primitives

CSO$_2$ FIFO Scheduler

CSO$_2$ Batch Scheduler

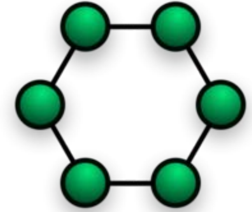
Time to pass a message 300 times around an n process ring (ms)

100000
10000
1000
100

1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 11000 12000 13000 14000 15000 16000 17000 18000 19000 20000

Number n of processes spawned

Ring topology



Java primitives

CSO_2 FIFO Scheduler

CSO_2 Batch Scheduler

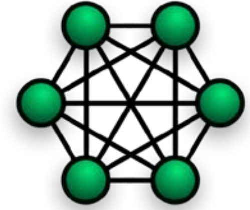
Time to pass a message 300 times around an n process ring (ms)

300000
250000
200000
150000
100000
50000
0

1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 11000 12000 13000 14000 15000 16000 17000 18000 19000 20000

Number n of processes spawned

Fully connected topology



Time to pass n^2 messages (ms)

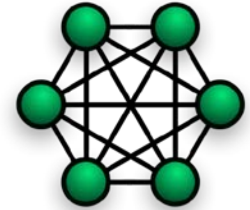
1000000
100000
10000
1000
100
10

1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 11000 12000 13000 14000 15000 16000 17000 18000 19000 20000

Number n of processes / actors spawned

Erlang
Scala Actors
JCSP
Java Primitives
Occam
CSO_2 FIFO Scheduler
CSO_2 Batch Scheduler
Go

Fully connected topology



Time to pass n^2 messages (ms)

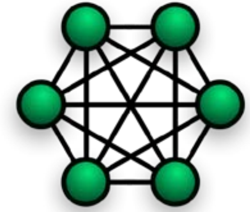
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0

- Erlang
- Scala Actors
- JCSP
- Java Primitives
- Occam
- CSO_2 FIFO Scheduler
- CSO_2 Batch Scheduler
- Go

1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 11000 12000 13000 14000 15000 16000 17000 18000 19000 20000

Number n of processes / actors spawned

Fully connected topology



Time to pass n^2 messages (ms)

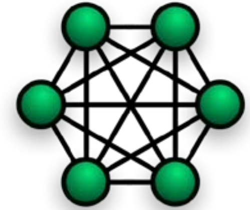
60000
50000
40000
30000
20000
10000
0

JCSP
Occam
CSO_2 FIFO Scheduler
CSO_2 Batch Scheduler
Go

1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 11000 12000 13000 14000 15000 16000 17000 18000 19000 20000

Number n of processes / actors spawned

Fully connected topology



CSO$_2$ FIFO Scheduler

Go

CSO$_2$ Batch Scheduler

Time to pass n^2 messages (ms)

16000
14000
12000
10000
8000
6000
4000
2000
0

1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 11000 12000 13000 14000 15000 16000 17000 18000 19000 20000

Number n of processes / actors spawned

Summary

- High performance library for building massively concurrent systems on the JVM
- Deadlock detection
- Outperforms Java primitives, JCSP, Scala Actors, Occam, and very close to Go