Actor-based Concurrency in Newspeak

A Project Report by

Nikolay Botev

May 2012

Agenda

- Problem
- Solution
  - Theoretical Foundations
  - **Practical Implementation**
- Demo
- Questions

Problem

- "Highly concurrent systems are more power efficient
  - Dynamic power is proportional to **V^2fC**
  - Increasing frequency (f) also increases supply voltage (**V**): more than linear effect
  - Increasing cores increases capacitance (**C**) but has only a linear effect" (emphasis mine)

Source: Slide 6 of [13] (Katherine Yelick)

Problem

- “Hidden concurrency burns power
  - Speculation, dynamic dependence checking, etc.
  - **Push parallelism discovery to software** (compilers and application programmers) to save power” (emphasis mine)

Source: Slide 6 of [13]

Problem

- The Datacenter as a Computer
  - “Slower CPUs are more power efficient; typically, CPU power decreases by $O(k^2)$ when CPU frequency decreases by k.”
  - “[…] although hardware costs may diminish, software development costs may increase […],”
  - “[…] developers may have to spend a substantial amount of effort to optimize the code […]” (underline mine)

Solution – Theory

- The Actor Model
  - Carl Hewitt (1970s)
- A Theoretical Model of Computation
  - Alternative to: Turing Machine, von Neumann
- Unbounded Non-determinism

Solution – Theory

- The Actor Model
  - “An Actor is a computational entity that, in response to a message it receives, can concurrently:
    - send messages to other Actors;
    - create new Actors;
    - designate how to handle the next message it receives.” Source: [9]

Solution – Practice

Factorial in Newspeak

```
factorial: n <Integer> ^ <Integer> = (
    (n <= 1)
        ifTrue: [ ^1 ]
        ifFalse: [ ^(factorial: (n - 1)) * n ]
)
```

Solution – Practice

Factorial in Newspeak (asynchronous)

```
factorial: n <Integer> ^ <Promise[Integer]> = (
    (n <= 1)
        ifTrue: [ ^1 ]
        ifFalse: [ ^(factorial: (n - 1)) <-: * n ]
)
```

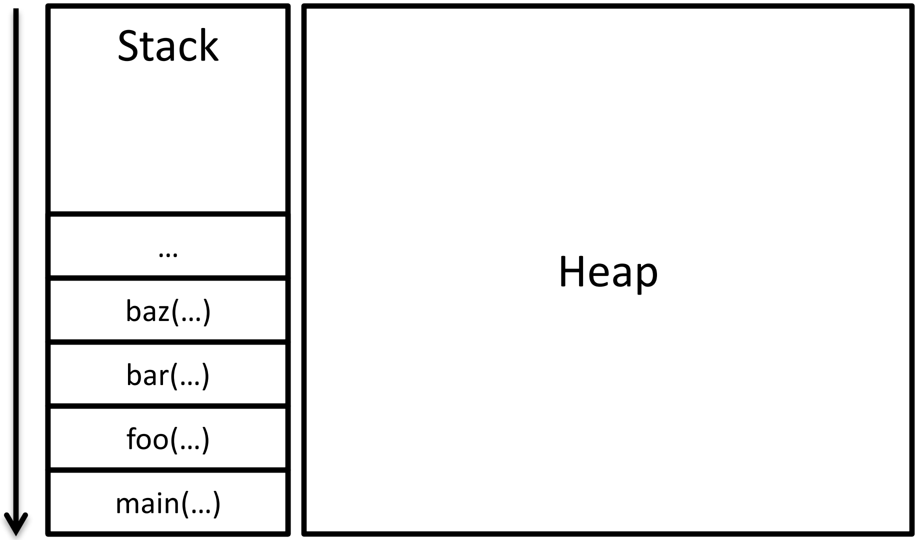
Solution – Practice

Using the factorial

```
| math f |
math:: actors createActor: Math mixin.
f:: math <-: factorial: 42.
f whenResolved: [
   Transcript show: 'Factorial of 42 is ', f.
].
```

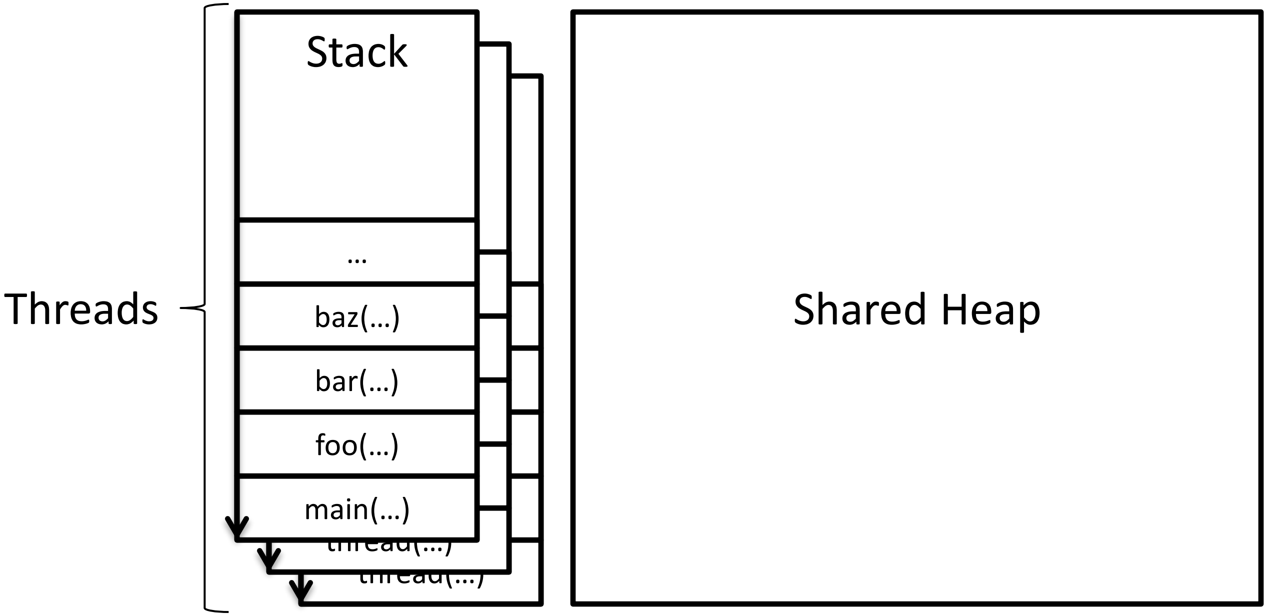
What is a Newspeak Actor?

Conventional Single-threaded Program



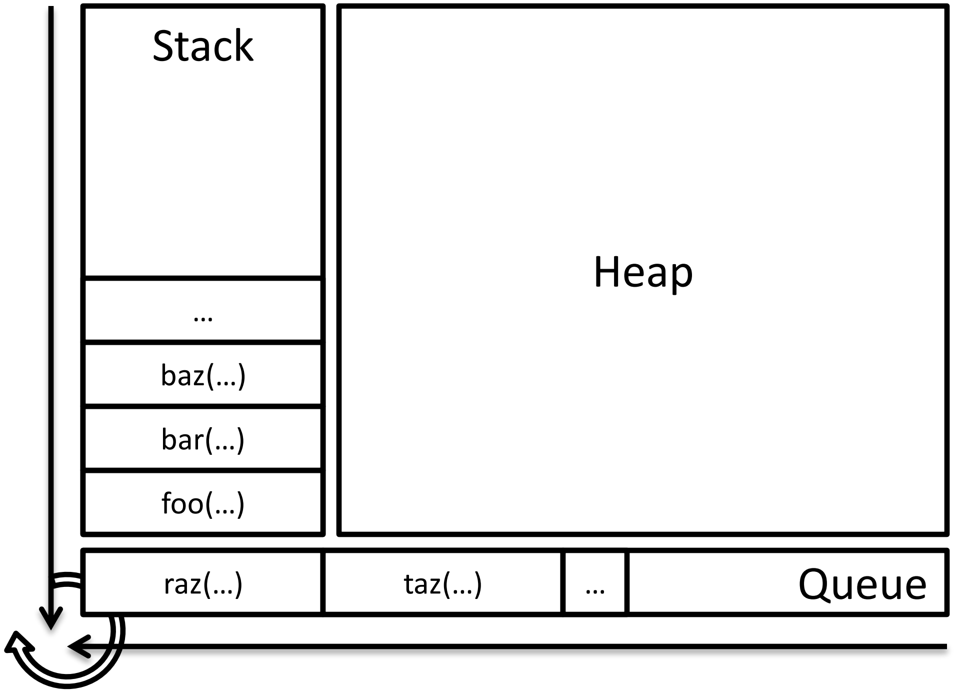
What is a Newspeak Actor?

Conventional **Multi-threaded** Program



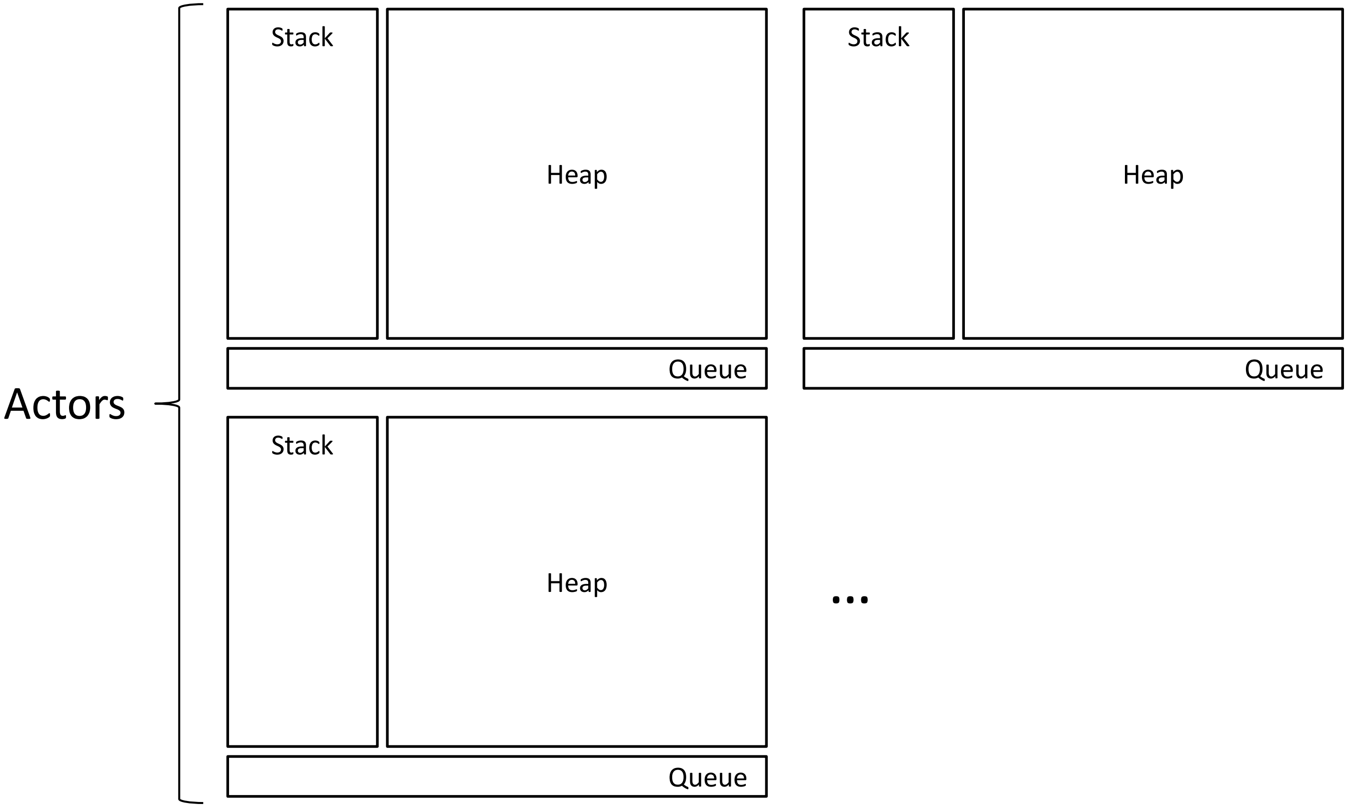
What is a Newspeak Actor?

Newspeak Actor



What is a Newspeak Actor?

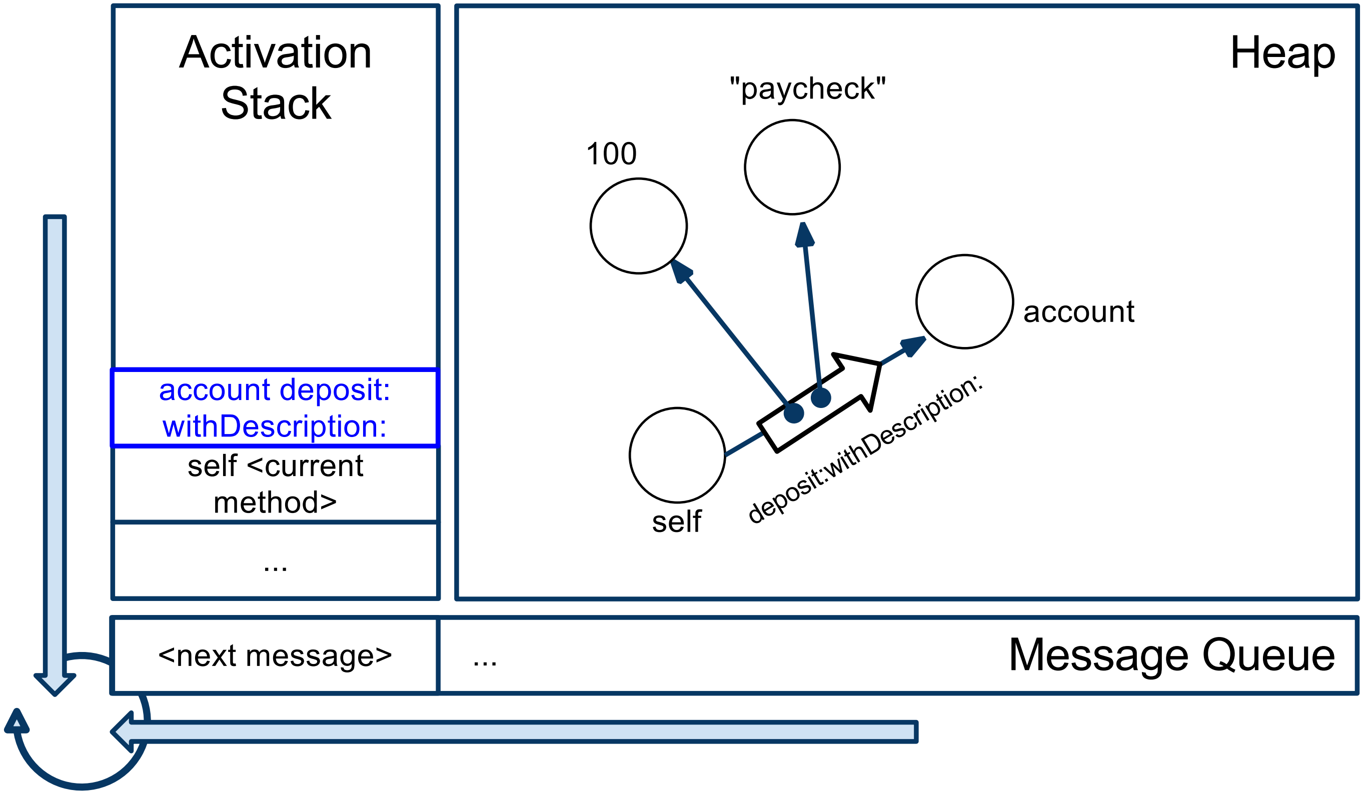
Newspeak Program



Actor Communication

Immediate-send

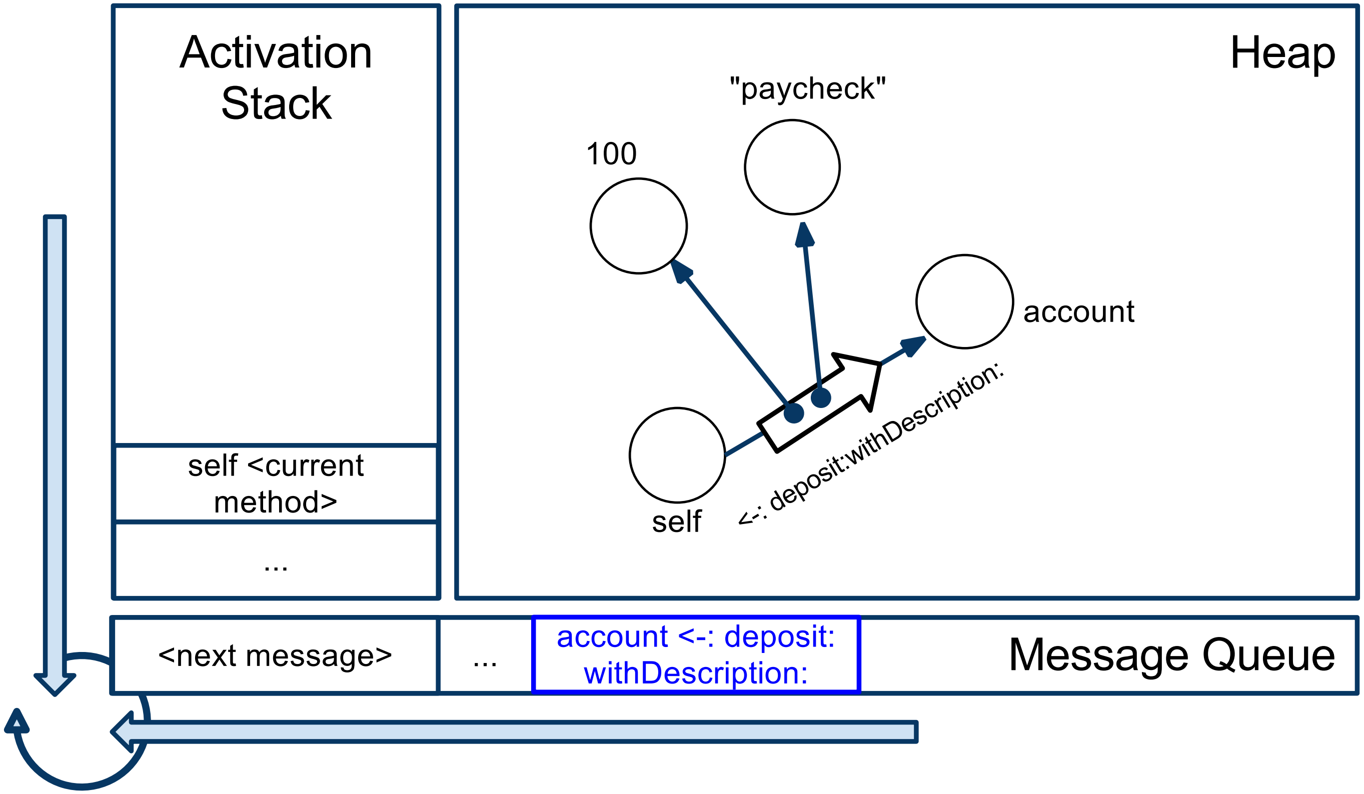
account deposit: 100 withDescription: ‘paycheck’.



Actor Communication

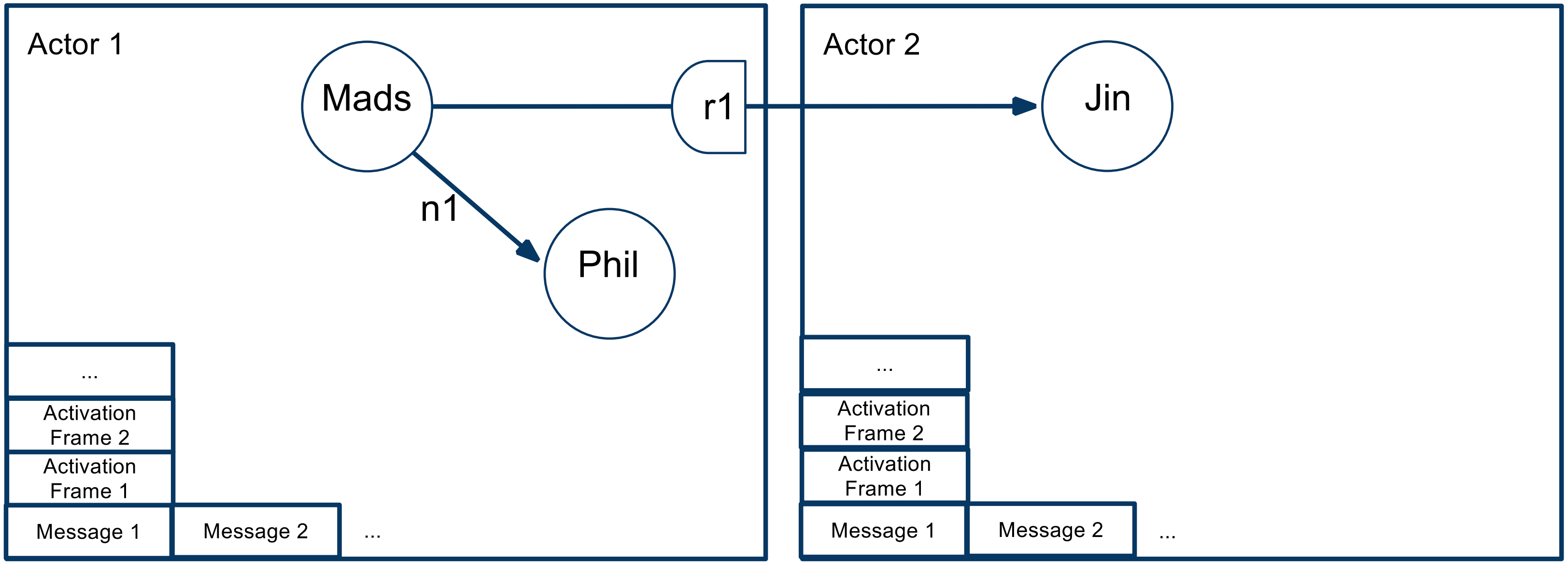
Eventual-send

account **<-:** deposit: 100 withDescription: ‘paycheck’.



Actor Communication

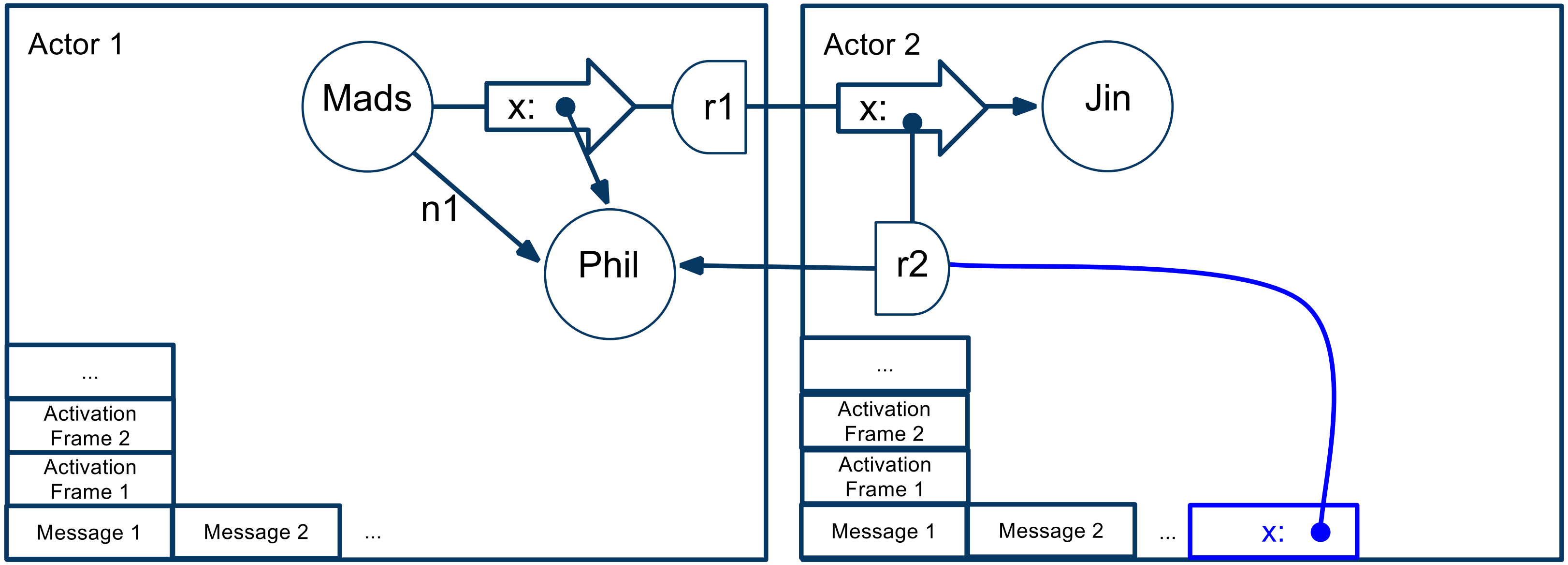
Far References



Actor Communication

Far References

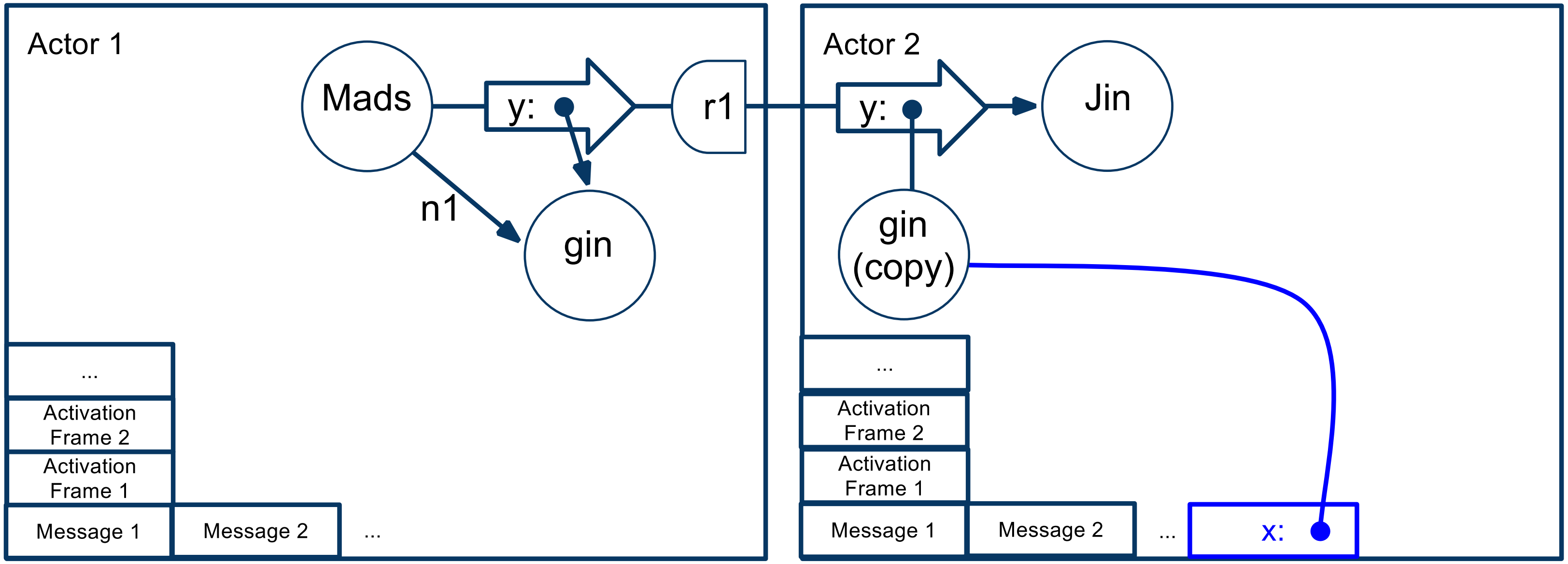
Mads says, r1 <-: x: n1.



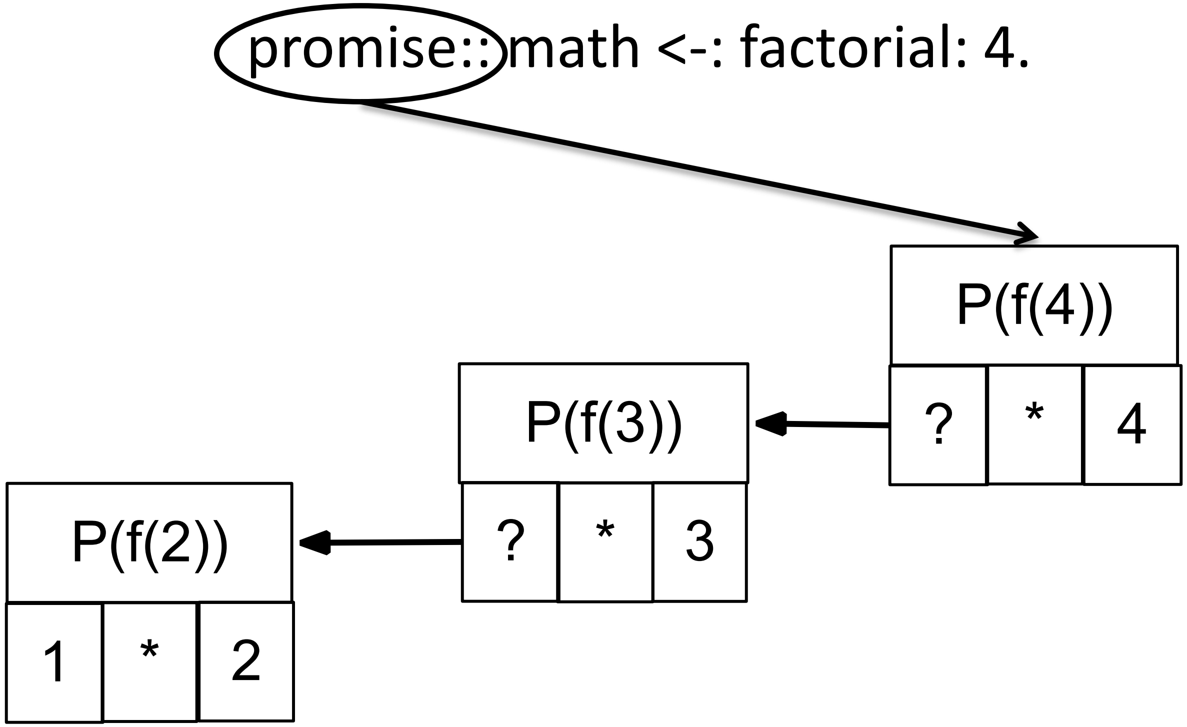
Actor Communication

Far References – Pass-by-Value

Mads says, r1 <-: x: n1.



Promise Pipelining



promise:: math <-: factorial: 4.

P(f(4))

? * 4

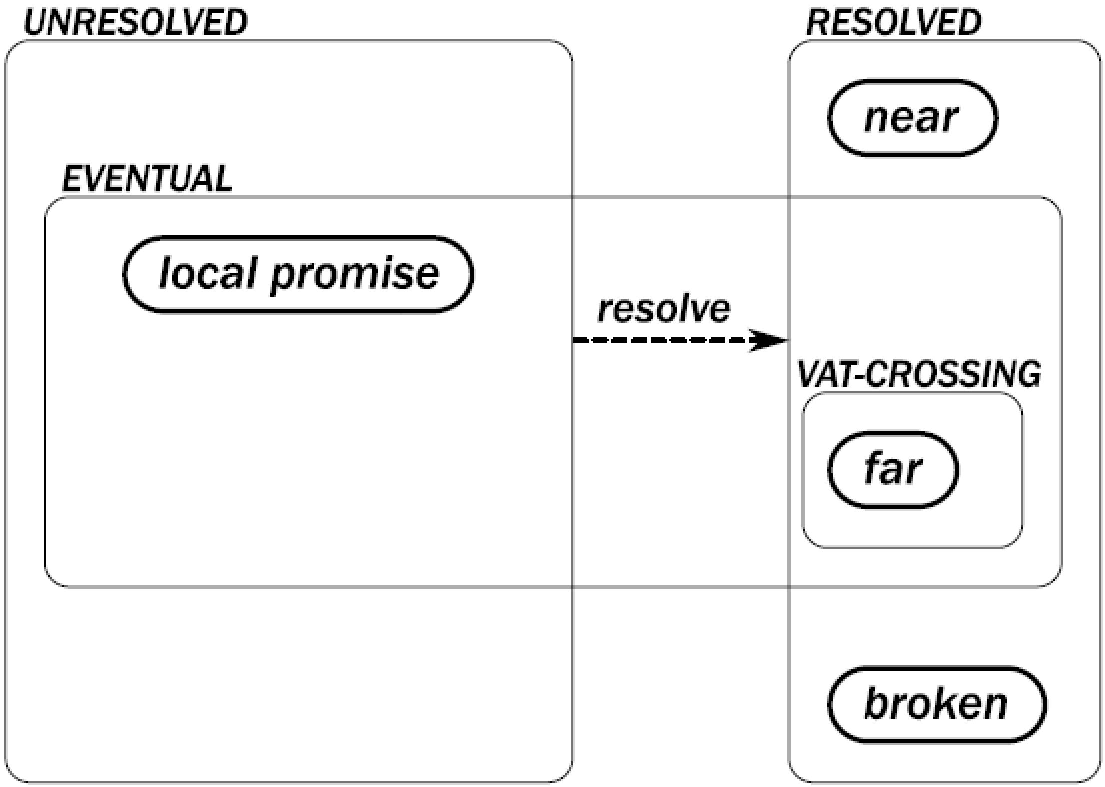
P(f(3))

? * 3

P(f(2))

1 * 2

Reference States



Source: [13] (modified without permission)

Summary

- Create an actor via **createActor:**
- Send a message to an actor via **<-:**
  - Composable thanks to Promise pipelining
- Await a Promise via whenResolved:catch:
  - Bridges concurrent and sequential computation

Key Qualities of Actors

- Simple,
- Lightweight,
- Automatic.

No.

Need.

To pool.

No pools, please!

The alternative is liberating!

Distinctions from E

- Actor creation based on mixins
- Asynchronous control structures
- Actor mirrors
- Reference states

Conclusion

Newspeak 4 Actors:

A small first step towards

“iAdaptive Concurrency” [9]

Demo

Questions

Thank You!