

CS 311O

Promises

Prof. Clarkson
Fall 2019

Today's music: *Call Me Maybe* by Carly Rae Jepsen

❄️❄️❄️ Today's questions will not count for attendance points ❄️❄️❄️

CLICKER QUESTION 1

Review

Previously in 3110: Advanced functional programming

- Streams
- Laziness

Today:

- **Promises**: a data structure and (functional) programming paradigm for concurrency

Concurrency

- Networks have multiple computers
- Computers have multiple processors
- Processors have multiple cores

...all working semi-independently

...all sharing resources

sequential: non-overlapping in duration

concurrent: overlapping in duration

- **parallel:** happening at the same time
- **interleaved:** rapidly switching between

Concurrency

At any given time, my laptop is...

- Streaming music
- Running a web server
- Syncing with web services
- ~~Scanning for viruses~~
- Running OCaml

The OS plays a big role in making it look like those all happen simultaneously

Concurrency

Applications might also want concurrency:

- **Web server** that handles many clients at once
- **Scientific calculations** that exploit parallel architecture to get speedup
- **GUIs** that want to respond to users while doing computation (e.g., rendering) in the background

CLICKER QUESTION 2

Programming models for concurrency

Threads: procedures executed concurrently

- CS 2110: **java.lang.Thread**
- Others:
 - OCaml **Thread**
 - pthreads
 - OpenMP

Programming models for concurrency

Promises: values computed concurrently

- CS 3110: OCaml **Lwt**
- Others:
 - **async/await** in JavaScript and .NET
 - **java.util.concurrent.Future**
 - OCaml **Async**

(and many other models)

PROMISES

Promises

Computation that promises to produce a value sometime in the future

Aka:

- future
- delayed
- deferred

Lwt : OCaml library for promises

Promises



A **promise** – ' **a** **Lwt . t** – is like a box:

- It starts out empty
- At some point in the future, it could be filled with a value of type ' **a**
- Once it's filled, the box's contents can never be changed ("write once")

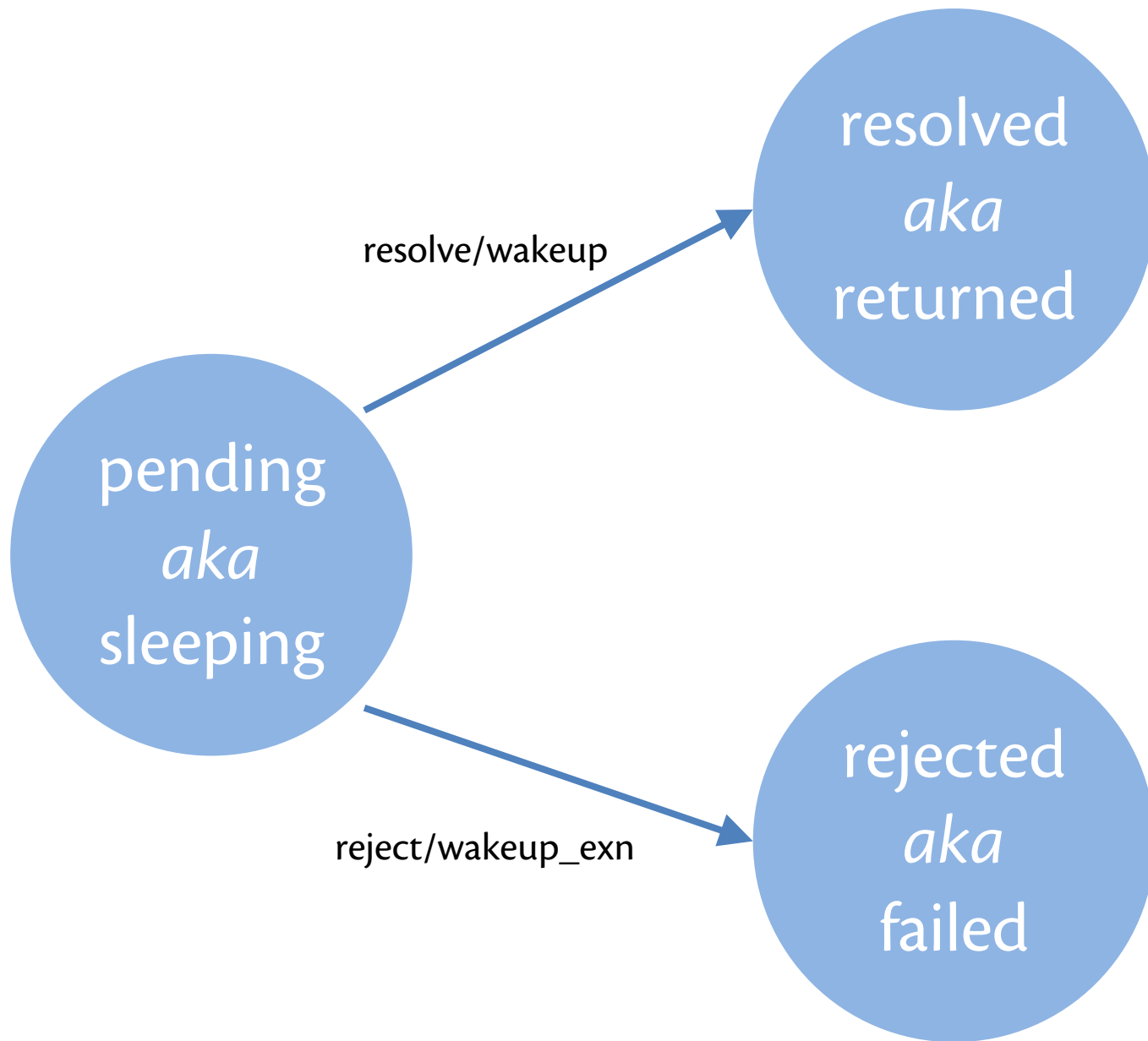
Resolver



A **resolver** – 'a **Lwt.u** – is what fills the box

Terminology:

- promise is **pending** aka sleeping: box is empty
- promise is **resolved** aka returned: box is full
- promise is **rejected** aka failed: box contains exn



Promise signature

```
(** A signature for Lwt-style promises,  
    with better names *)  
module type Promise = sig  
  
  type 'a promise  
  type 'a resolver  
  
  type 'a state =  
    Pending | Resolved of 'a | Rejected of exn  
  
  (** [state p] is the state of the promise *)  
  val state : 'a promise -> 'a state
```

Promise signature

```
(** [resolve r x] resolves the promise  
    [p] associated with [r] with value [x].  
    Requires: [p] is pending. *)
```

```
val resolve : 'a resolver -> 'a -> unit
```

```
(** [reject r x] rejects the promise [p]  
    associated with [r] with exception [x].  
    Requires: [p] is pending. *)
```

```
val reject : 'a resolver -> exn -> unit
```


Promise signature

```
(** [make ()] is a new promise and  
    resolver. The promise is pending. *)
```

```
val make : unit -> 'a promise * 'a resolver
```

```
(** [return x] is a new promise that is  
    already resolved with value [x]. *)
```

```
val return : 'a -> 'a promise
```

```
end
```

Digression on Cornell history

- `ivars` = promises+resolvers
- Used for parallel computing in language called Id
 - [Arvind, Nikhil, and Pingali 1986]
 - Keshav Pingali, Cornell CS prof 1986-2006
- Implemented in *Concurrent ML* by John Reppy (Cornell PhD 1992)



Lwt

Typical use of library is to do asynchronous I/O

- Launch an I/O operation as a promise
- OS helps to resolve promise

Source of parallelism: OS, not OCaml

call me maybe?

CALLBACKS

Managing Promises

What if program has many promises "in flight"?

- Web server handling many client
- Spreadsheet updating many cells
- Game updating many enemies

Need a way to manage dependencies of computations upon promises...

bind **promise** **callback**

bind :

'a Lwt.t

-> ('a -> 'b Lwt.t)

-> 'b Lwt.t

promise >>= callback

(>>=) :

'a Lwt.t

-> ('a -> 'b Lwt.t)

-> 'b Lwt.t

Implementing bind

- Store a list of callbacks with each promise
- After promise is resolved, Lwt runs callbacks
- If promise never resolved (or fails), no callback



Callback execution

- **Single-threaded:** only one callback running at a time
- **Cooperative:** callbacks run to completion
- **Nondeterministic:** unspecified which runs first

Upcoming events

- MS3 due in 1 week

This is resolved.

THIS IS 3110