CSP

PROGRAMMATION ORIENTÉ AUTISTES

Il était une fois

http://www.infoq.com/presentations/transducer-clojure

(à la minute 42 il dit *channel*)

Il était une autre fois

https://github.com/fxg42/async-comparison

CALLBACK HELL YEAH

```
find: (cb) ->
 Mongo.connect(CONNECTION_STRING, (err, conn) ->
  cb err
  else
    coll = conn.getCollection('person')
    coll.find().toArray (err, data) ->
      if err then cb err
      else cb null, data
```

ASYNC AUTO *AKA* MAKE

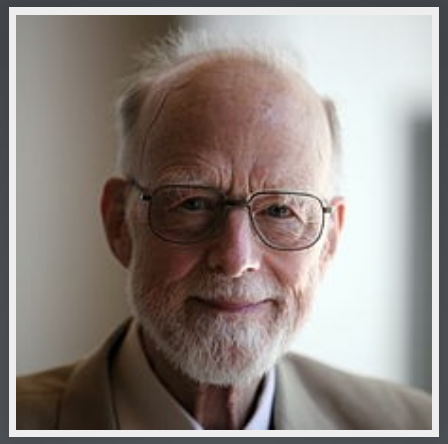
```
getConnection  = (cb)-> ... cb null, connection
getCollection = (cb, {connection})->... cb null, collection
doFind = (cb, {collection})->... cb null, peoples
find: (cb) ->
  async.auto
    connection: getConnection
    collection: ["connection",getCollection]
    find: ["collection", doFind]
  , (err, {find}) -> cb err, find
```

CSP

```
getConnection  = -> ... put chan, connection

find: (cb) ->
  try
    go ->
      connection = yield getConnection()
      collection = connection.getCollection("person")
      collection.find().toArray(cb)
  catch e
    cb e
```

CSP BEGINS



C.A.R. HOARE

- Classe
  - Quicksort
  - Go To Statement Considered Harmful
- Pas classe
  - Logique de Hoare (vérification formelle)
  - null
- Channel Sequencial Processing (1978)

LA BASE

- chan
- put
- take
- go block

Put & take sont bloquants

Rappel générateurs ES6

```
var x = function*(){
  yield 'Bonjour'
  yield 'le'
  yield 'monde'
}
var y = x()
y.next() //> Object {value: 'Bonjour, done: false}
y.next() //> Object {value: 'le', done: false}
y.next() //> Object {value: 'monde', done: false}
y.next() //> Object {value: undefined, done: true}
```

```
 fn ->
  yield 'Bonjour'
```

Exemple

```
csp = require 'js-csp'
player = (name, table) ->
  csp.go ->
    while true
      ball = yield csp.take table
      ball.hits++
      console.log name, ball.hits
      yield csp.timeout(100)
      yield csp.put table, ball

table = csp.chan()

player "ping", table
player "pong", table

csp.putAsync table, hits: 0
```

Le ping pong c'est bien beau
mais moi je fais des vrais systèmes

- pipe (unix |)

```
-> - ->
```

- split

```
                    |->
->(split(predicat))
                    |->
```

- merge

```
->|
  (merge) ->
->|
```

- pipline

```
-> (doSomething()) ->
```

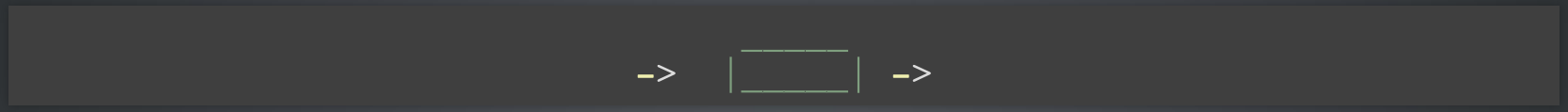
- mult

```
    | ->
->  | ->
    | ->
```

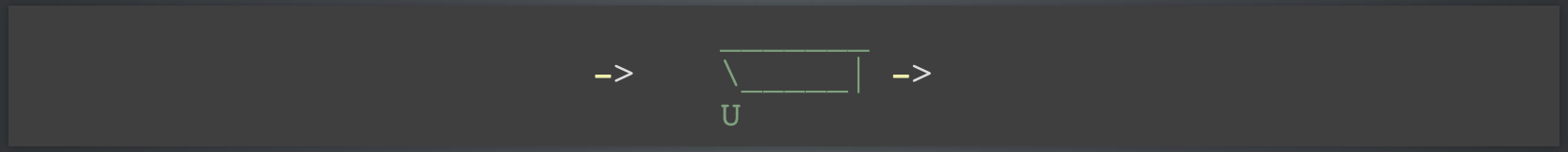
- Pub/Sub
- Mix

♥(CHANNEL) = BUFFER

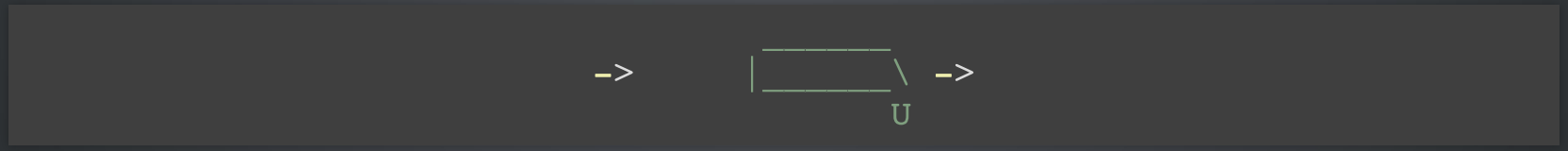
Fixe



Dropping



Sliding



CSP != ACTEUR

	CSP	*Acteur*
On Parle à	Channel	Processus
Isolation	Tous pour un	Chacun pour sois
Nb processus	∞ ++	∞ ++
Réseau	Un jour ...	les doigts dans le nez

IMPLÉMENTATION

- Ada
- Haskell
- Go (Compilateur)
- Clojure core.async (Lib macro)
- cspjs (Marco sweetjs)
- js-csp/node-csp (Genérateur)

JS-CSP: la réponse à tout mes problèmes ?

js-csp n'est pas:

- La réponse à tout tes problèmes
- La seule option (Async.js n'est pas mort, même si ça a 5 ans)
- (encore) **Super** facilement interfacable avec:
  - streams
  - promesses
  - callback
  - event
- décidé sur quoi faire des erreurs
- Multi-tread
- fait pour le réseau

js-csp permet:

- donner un style synchrone a du code asynchrone
- Simplifier la coordonination de tâches
- de raisoner sur des petits processus simple

RÉFÉRENCE

- David Nolen: http://swannodette.github.io/
- Rick Hickey: http://www.infoq.com/presentations/core-async-clojure
- James Long: http://jlongster.com/Taming-the-Asynchronous-Beast-with-CSP-in-JavaScript
- Rob Pike: https://www.youtube.com/watch?v=f6kdp27TYZs