
UM BREVE TOUR
PELO "NOVO" MUNDO DA
CONCORRÊNCIA
robotlolita

Apresentada por Quil (direto de Idol Hell)

http://robotlolita.me

@robotlolita em todo lugar

Concorrência & Paralelismo

Houve um tempo em que Purrgrammers precisavam compartilhar um computador para rodar programas.

Programas eram simples instruções sequenciais.

Só um programa podia executar por vez,

então o tempo de espera podia ser bem alto...


A GATARIA ESTÁ RECLAMANDO DE FICAR EM PÉ NA FILA POR MUITO TEMPO.
LIMITA O TEMPO DE USO.
CADEIRAS?
empurra
BASEADO NA COMIC DE @HEJIBITS

No final, pensaram em duas soluções...

Concorrência

Processos intercalados

Paralelismo

Processos independentes
executando ao mesmo tempo.

Concorrência

Componentes e programas diferentes compartilham recursos, portanto precisam ser ordenados corretamente.

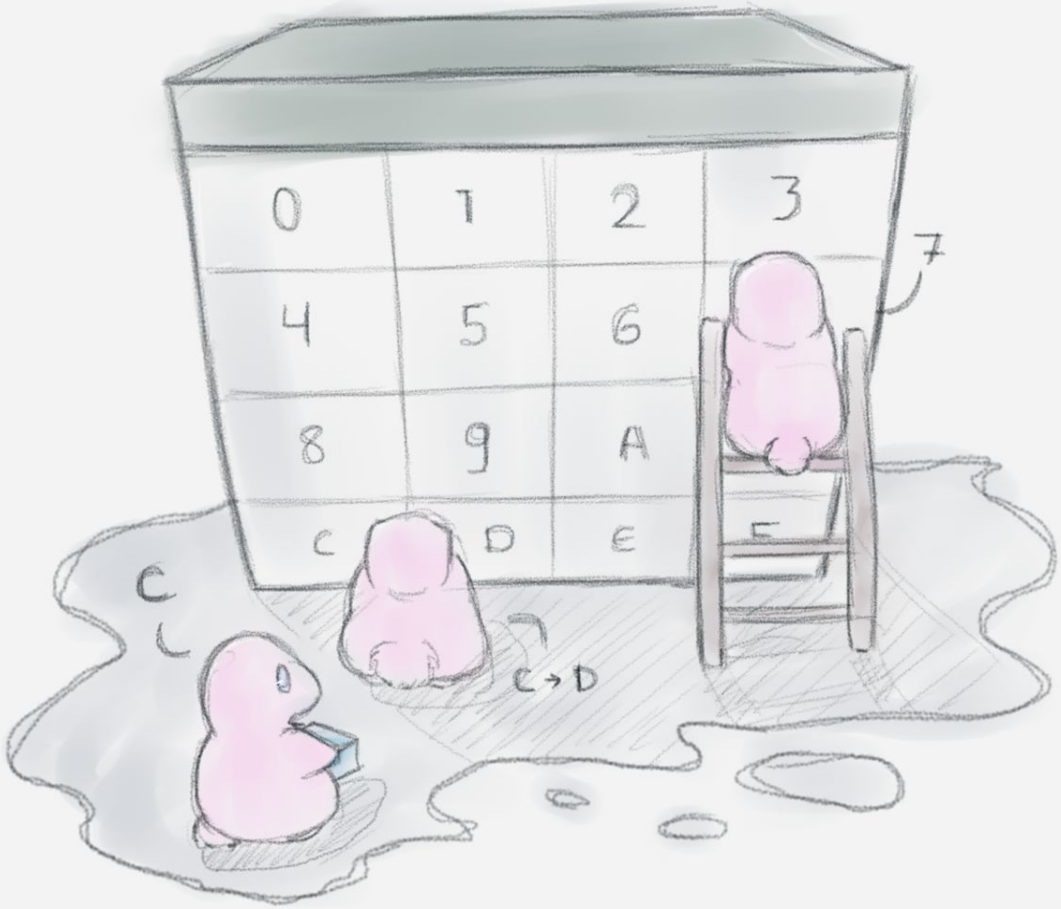
SEE ALSO:

Parallelism is not Concurrency, R. Harper

II

Mecanismos de Concorrência

Threads: vários componentes, uma memória



Processos: vários componentes, várias memórias

Agendamento de Tarefas

Preemptivo

Cooperativo

Modelos de Concorrência

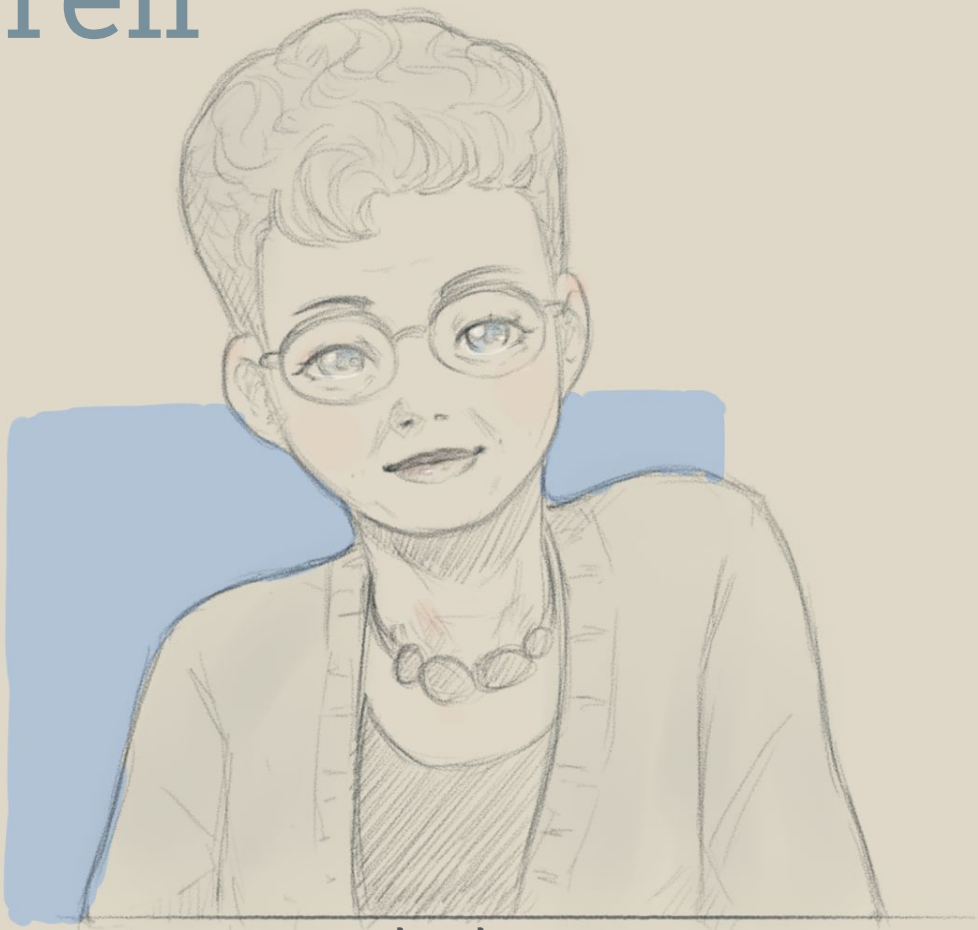
Actors

(ORIGEM)

- **1973,** Hewitt, Bishop, Steiger
- **Semântica operacional** (Greif, ‘75; Agha et al, ‘97)
- **Influências:** Lisp, Smalltalk, Packet Switching
- **Outros contribuidores:** Baker, Clinger, Agha
- **Acidentalmente:** Erlang (Armstrong, ‘03); E (Miller, ‘05)
- **Actors hoje:** Erlang, Elixir, Pony, Scala (Akka)

Irene Greif

(BY THE WAY)



Foi a primeira mulher a conseguir um PhD do MIT em CS, com sua tese sobre Actors.

Actors

(IDÉIA)

“E se tivéssemos milhares de CPUs, independentes e paralelizáveis?”

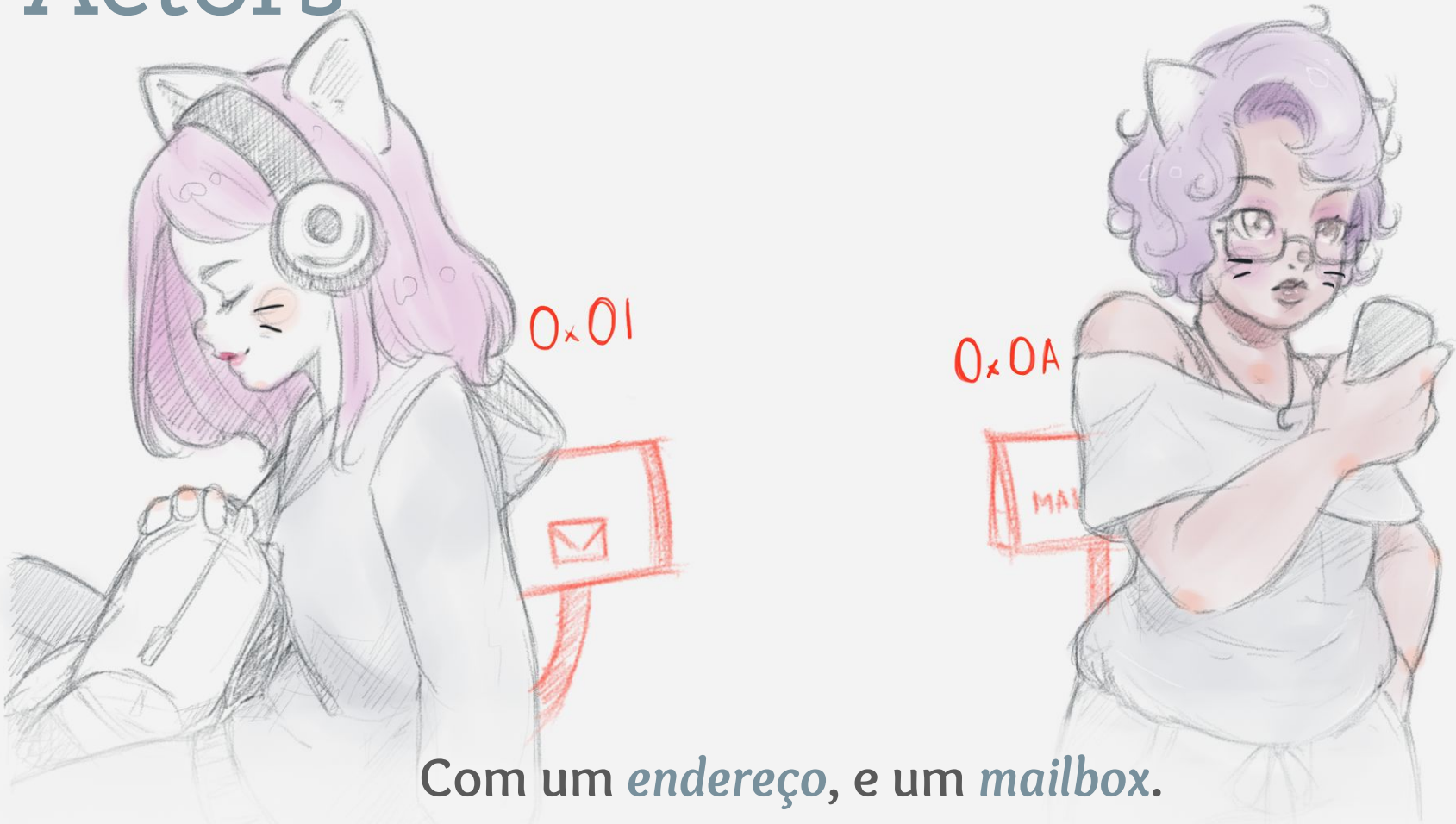
Actors

(modelo)

Cada *actor* é uma unidade isolada e completa de processamento.

Actors

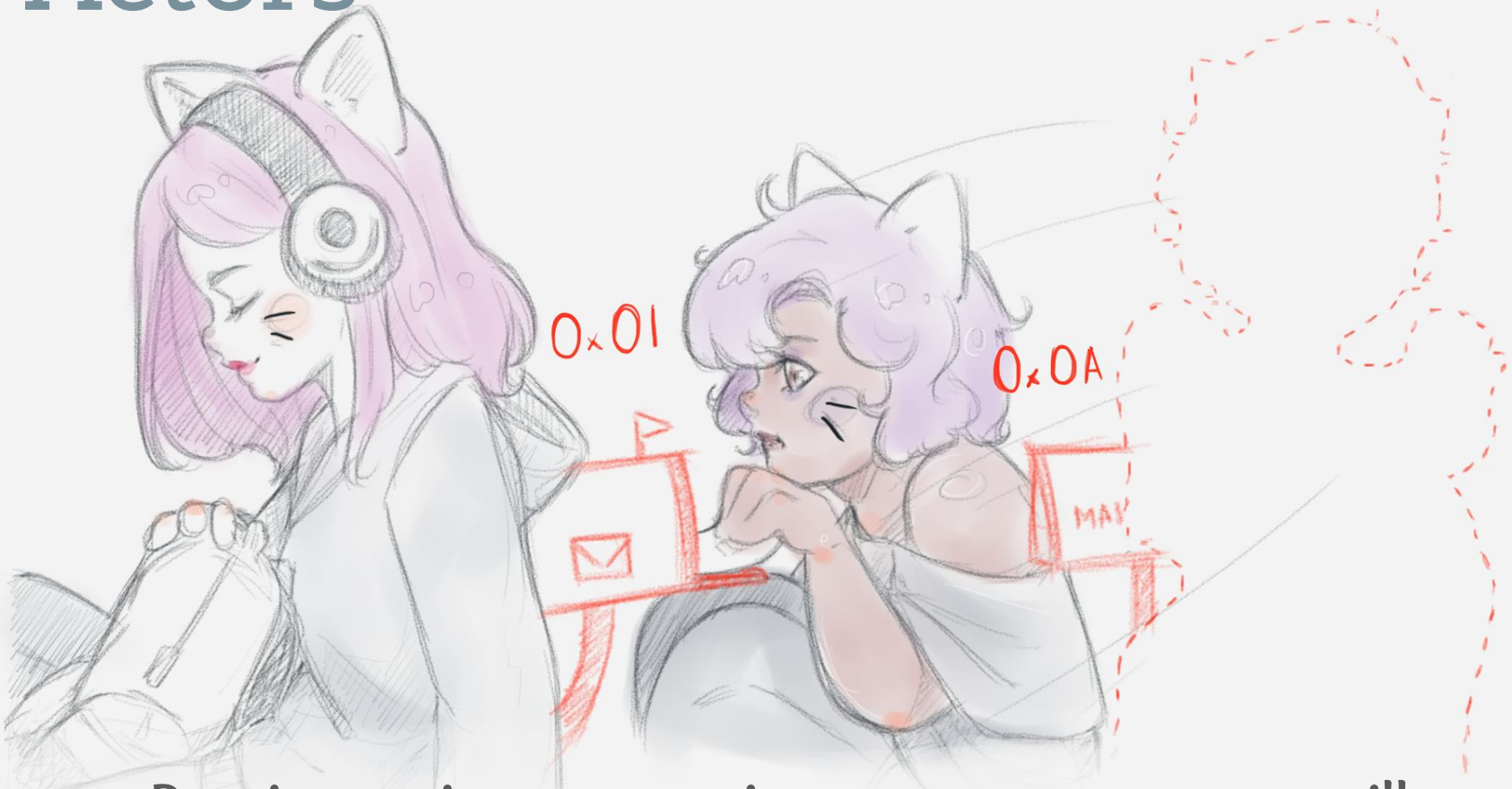
(MODELO)



Com um *endereço*, e um *mailbox*.

Actors

(MODELO)



Para interagir, *actors* enviam mensagens para o mailbox.

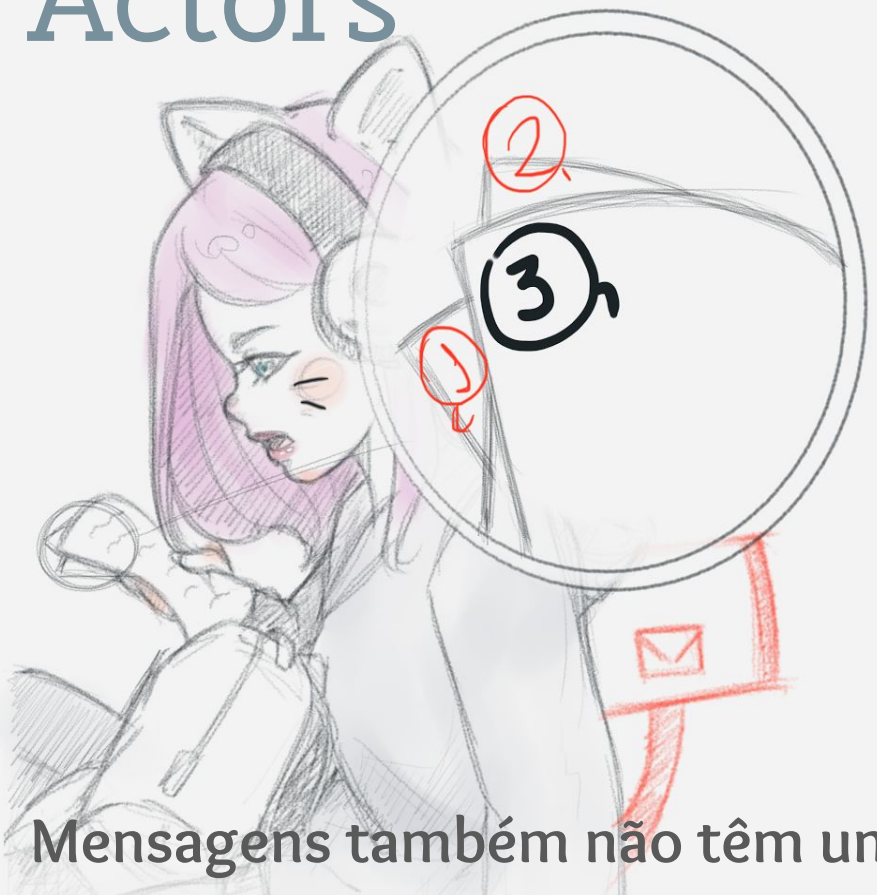
Actors

(MODELO)

Mensagens são *assíncronas*.

Actors

(MODELO)



Mensagens também não têm uma ordem de recebimento definida.
Isso permite otimizações e o torna mais tolerante à falhas.

Actors

(MODELO)

* Mailboxes are magic

Para responder a uma mensagem, *actors* enviam uma mensagem de volta para o remetente.

Actors

(conclusão)

- 👍 Sistemas distribuídos
- 👍 Capability security

CSP

(ORIGEM)

- 1977, Tony Hoare
- **Communicating Sequential Processes** (Hoare, ‘85)
- **Influências:** CCS (Milner, ‘80)
- **Outros contribuidores:** Brookes, Roscoe
- **CSP hoje:** Go, Clojure (core.async), Rust

CSP

(IDÉIA)

“Concorrência pode ser eficiente e correta com alguns componentes: Processos, Canais, Seqüências, Intercalações, e Escolhas.”

CSP

(MODELO)

Processos são entidades anônimas que podem executar computações.

CSP

(MODELO)

Eventualmente, *processos* precisam interagir entre si. Essa interação se dá através de *mensagens*.

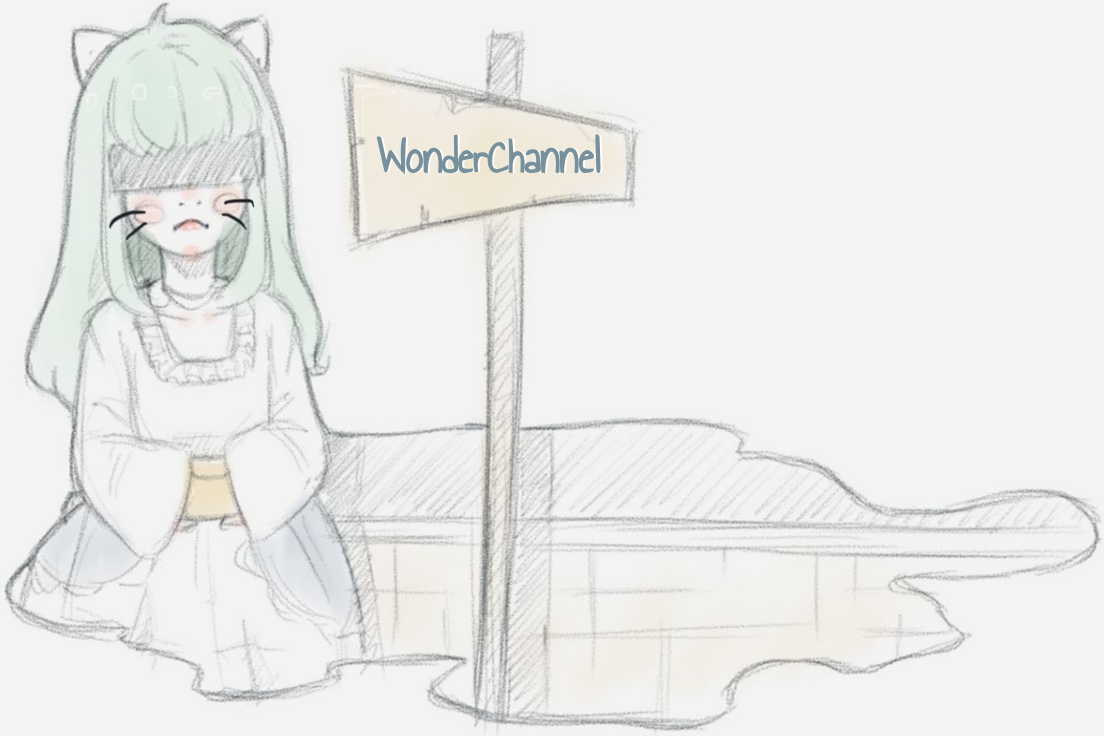
CSP

(MODELO)

Para enviar mensagens, processos precisam compartilhar um *canal*.

CSP

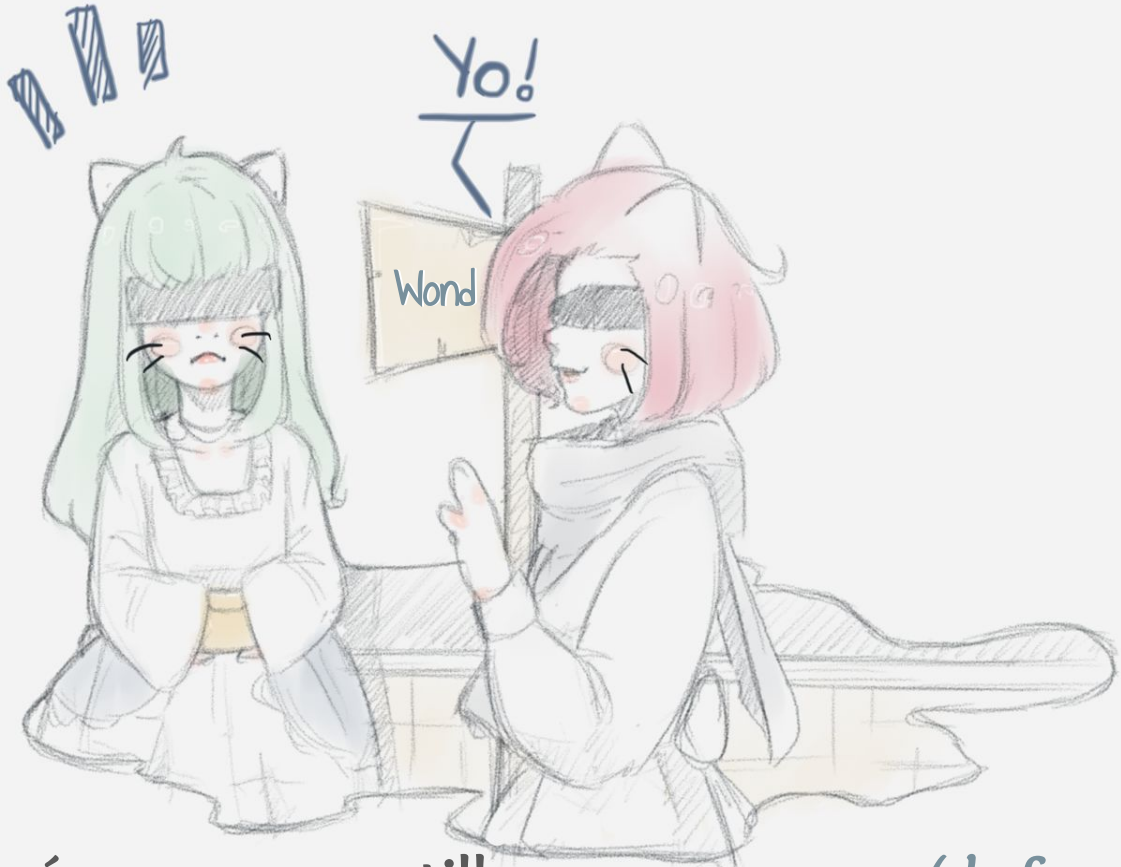
(MODELO)



Canais são pontos de encontro.

CSP

(MODELO)



Processos se reúnem, compartilham mensagens *(de forma síncrona)*, e então seguem com suas vidas.

CSP

(conclusão)

- 👍 Concorrência e paralelismo locais
- 👍 Backpressure

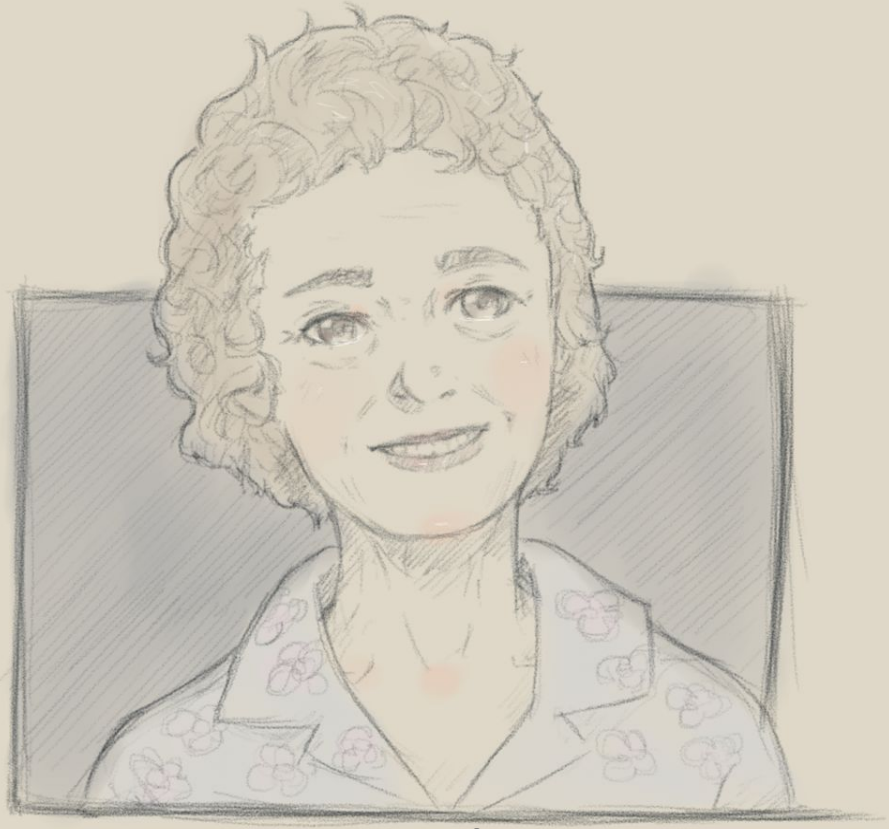
Futures

(origem)

- 1976~1977, Baker & Hewitt; Hibbard; Friedman & Wise
- **Call-Streams/Pipelining** (Liskov, ‘88)
- **async/await** (Fischer et al, ‘07)
- **Promises hoje:** *Basicamente todas as linguagens. Srsly!*

Barbara Liskov

(BY THE WAY)



Uma das pioneiras da computação, fez contribuições significativas em PL e concorrência.

Futures

(IDÉIA)

“Ao invés de esperar um valor ser computado, uma representação do valor eventual pode ser utilizada, permitindo que o programa continue executando até que o valor seja, de fato, necessário.”

Futures

(modelo)

Algumas computações podem demorar muito. Ao invés de esperá-las, um *future* é retornado imediatamente.

Futures

(MODELO)

O programa continua executando normalmente, como se já tivesse o valor que ainda vai ser computado.

Futures

(MODELO)

Quando o programa depende realmente do valor para continuar, ele pára e espera o valor ser computado.

Futures

(conclusão)

- 👍 Pipelining

Streams

(origem)

- **..., 1997,** Hudak e Conal; **~2010,** Microsoft, **2012,** Czaplicki
- **AKA:** Signals (FRP), Observables (Rx)
- **Influências:** Data-flow e programação síncrona
- **Streams hoje:** *Elm; JS, JVM, .NET, Dart, Swift, ... (Rx), Furipota*

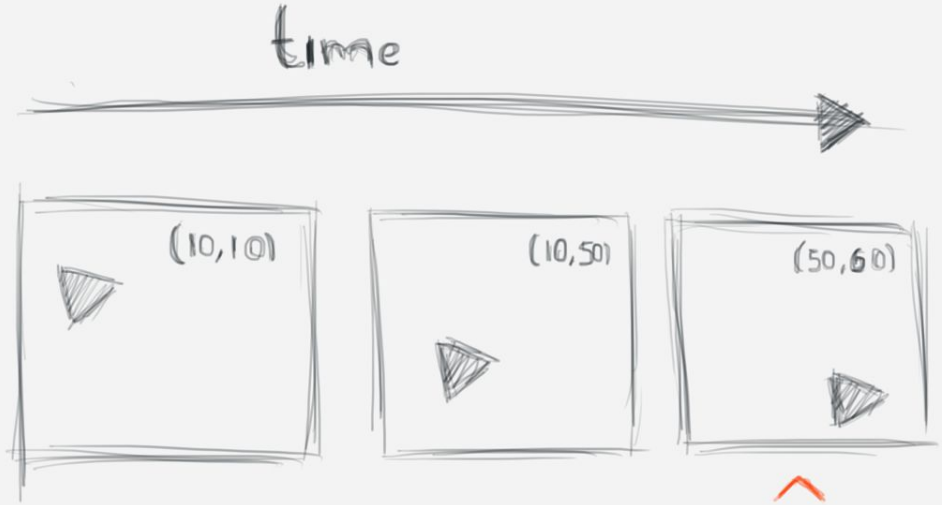
Streams

(IDÉIA)

“E se computações pudessem ser definidas em cima de valores que mudam com o tempo?”

Streams

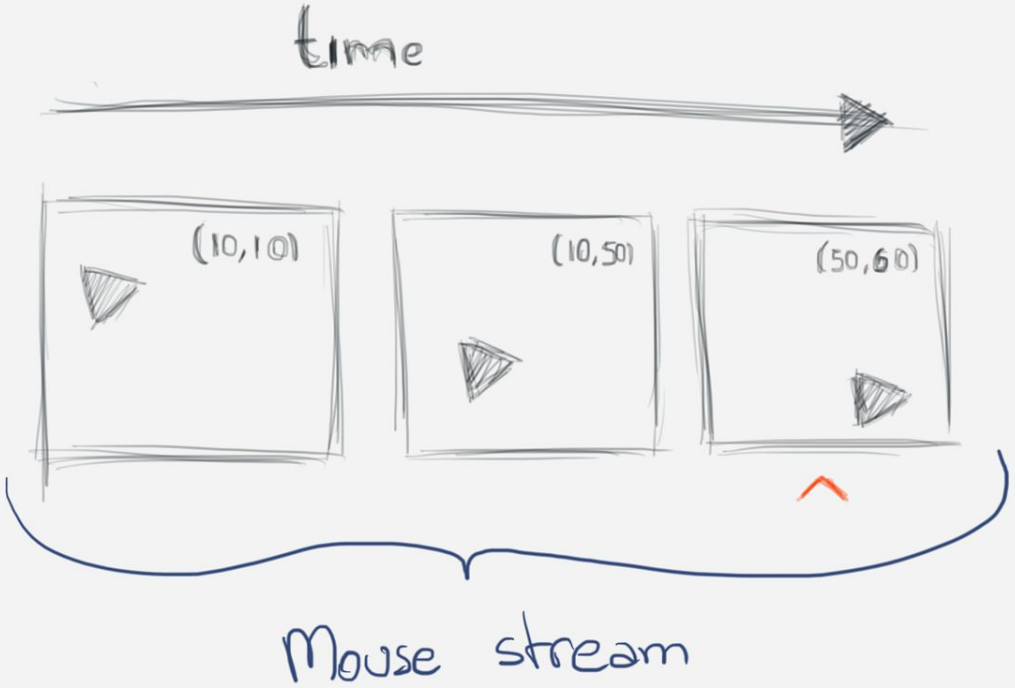
(MODELO)



Valores como a posição do mouse mudam ao longo do tempo, mas a maioria das linguagens permite usar apenas o valor mais recente.

Streams

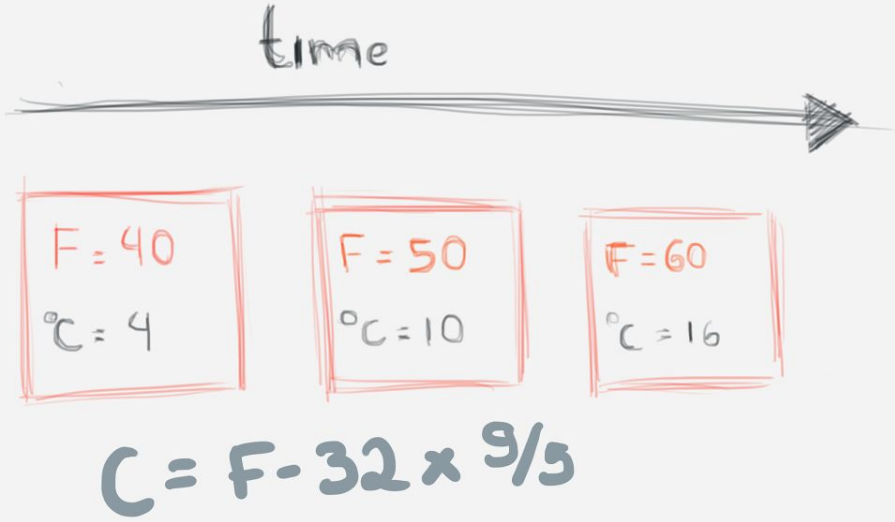
(MODELO)



Streams representam a evolução de um valor ao longo do tempo.

Streams

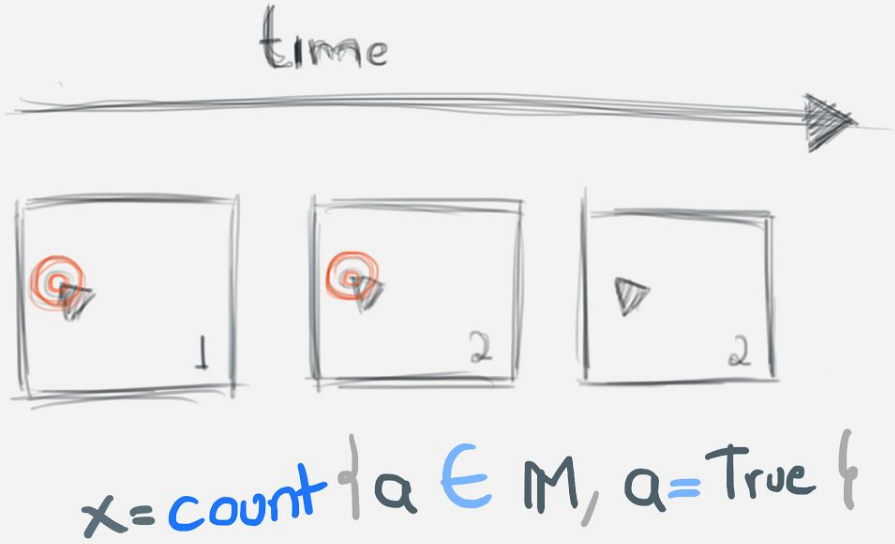
(MODELO)



Isso permite descrever conceitos como “°C = (F - 32) × 9/5”

Streams

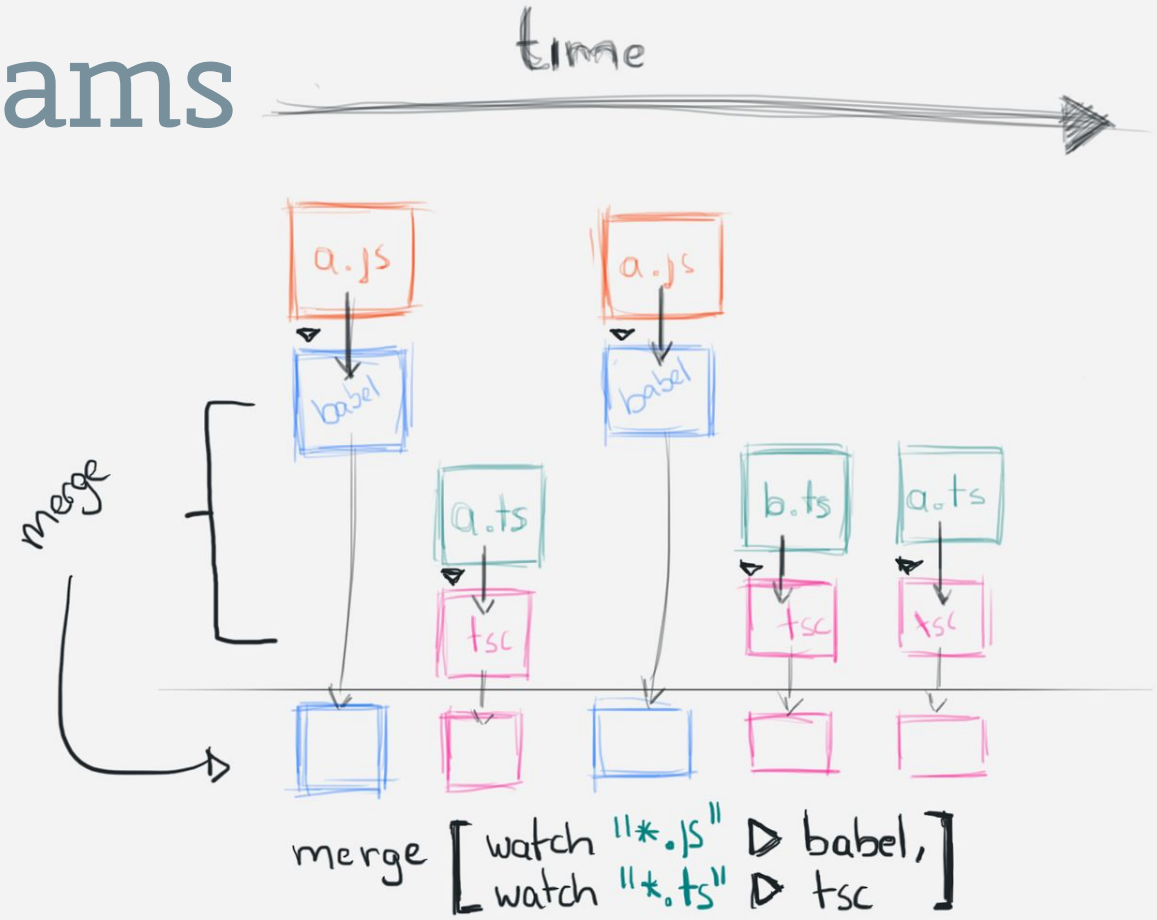
(MODELO)



Ou conceitos como “o número de cliques do mouse até agora”

Streams

(MODELO)



Streams podem também ser combinados.

Streams

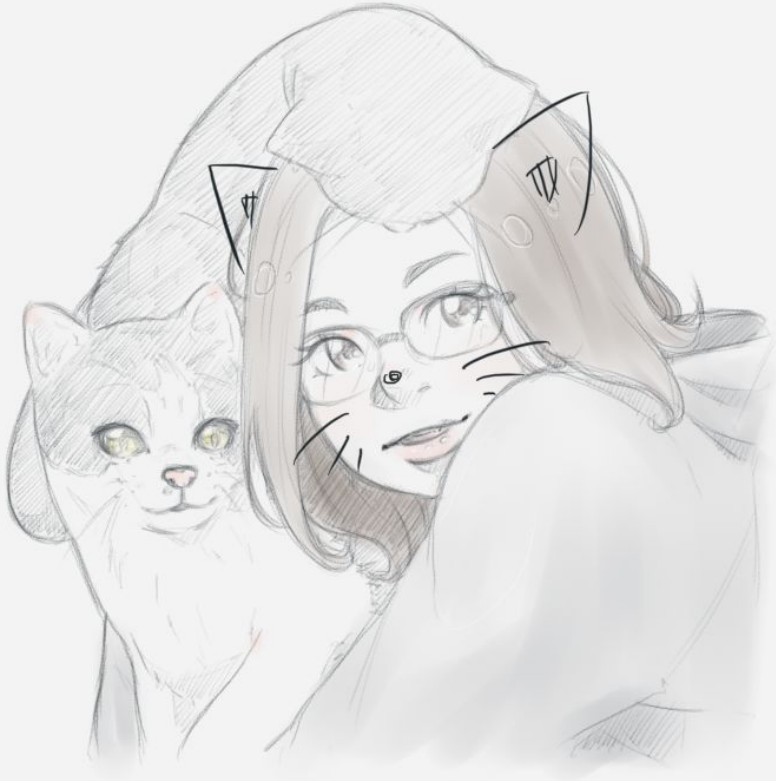
(conclusão)

- 👍 **Aplicações reativas** *(e.g.: GUIs, CI, CD, --watch ...)*

Exciting stuff

Spores

Heather Miller, 2015



LVars

Lindsey Kuper, 2013

Materiais recomendados

History of Actors
(Tony Garnock-Jones, 2016)

Channels, Concurrency, and Cores: A new CML implementation
(Andy Wingo, 2017)

Core.Async - CSP using Channels, in Clojure
(Rich Hickey, 2014)

How Do Promises Work?
(Quil, 2015)

What does it mean to be Reactive?
(Erik Meijer, 2014)

Esses slides estão disponíveis em concurrency.robotlolita.me

The end.