

COMPSCI 734 – Preview

Radu Nicolescu

Department of Computer Science
University of Auckland

8 Mar 2019

13 Mar 2019

- ① Kleisli fish
- ② Other exotic beasts in our Zoo: asyncs, actors, channels
- ③ Assignment side-bar

Function compositions and pipelines – high-order functions

- Consider

```
1 let f x = x + 1 // increment   f: int -> int
2 let g x = x + x // double     g: int -> int
3
4 let a = [|10; 20; 30|] // array
```

- Maps – high-order functions

```
1 let b = a |> Array.map f |> Array.map g
2
3 let b'' = Array.map g (Array.map f a)
4
5 let b' = a |> Array.map (f >> g) // map fusion
6
7 // b = b' = [|22; 42; 62|]
```

Map : High-order function

- Consider

```
1 let f x = x + 1           // increment f: int -> int  
2 let a = [|10; 20; 30|]   // array
```

- Map – high-order function

```
1 let b = a |> Array.map f  
2 // b = [|11; 21; 31|]
```

- Equivalent imperative code

```
1 let b' = Array.create a.Length 0  
2 for i = 0 to a.Length-1 do  
3     b'.[i] <- f a.[i]  
4 // b' = [|11; 21; 31|]
```

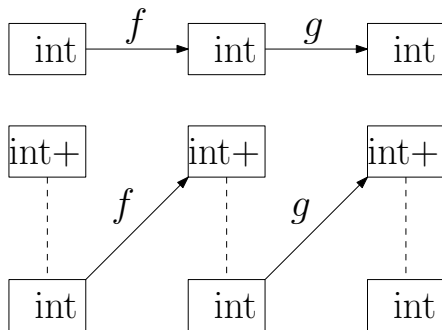
Function compositions – classical and Kleisli

- Software composability
- Classical composition \gg \ll \circ
- Limits of classical composition – e.g. what if function f may raise exceptions or doesn't exactly return an int?

```
1 let f x = 100 / x // x = 0 ?  
2  
3 let f x = [|x+1|] // singleton int array
```

- How do we compose this f with a g that takes just an int (and doesn't want to know about exceptions)?
- Kleisli composition (aka Kleisli “fish”) $\gg=>$ $\ll=<$

Classical vs Kleisli compositions



Kleisli compositions and pipelines

- Consider

```
1 let s x = [| x+1|]
2 let t x = [| x+x |]
3
4 let c = a |> Array.collect s |> Array.collect t
5
6 // c = [|22; 42; 62|]
```

- Kleisli fish definition

```
1 let (>=>) f g =
2   fun x -> x |> Array.collect f |> Array.collect g
```

- Compact expression

```
1 let c' = a |> (s >=> t)
2
3 // c' = [|22; 42; 62|]
```

Map vs Collect : High-order functions

- Consider

```
1 let s x = [|x+1|] // increment f: int -> int []  
2 let a = [|10; 20; 30|] // array
```

- Map – high-order function

```
1 let b = a |> Array.map s  
2 // b = [| [|11|]; [|21|]; [|31|] |]
```

- Collect – high-order function – aka flatMap

```
1 let b' = a |> Array.collect s  
2 // b' = [|11; 21; 31|]
```

- Equivalent imperative code – home exercise

Asyncns

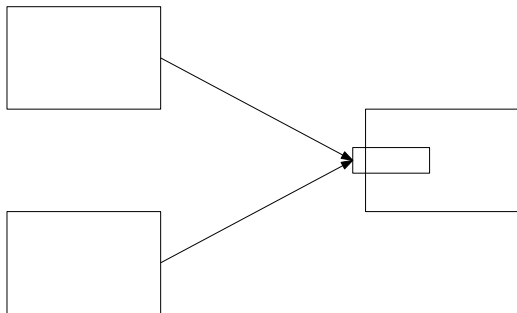
- Basic scenario

```
1 async {  
2     ... // thread#1  
3     let! r = page_download_async () // await  
4     ... // thread#1 or #2  
5 }
```

- For external time consuming tasks – only one thread is needed (JS)
- Also for parallel/concurrent processing (.NET)

Actors

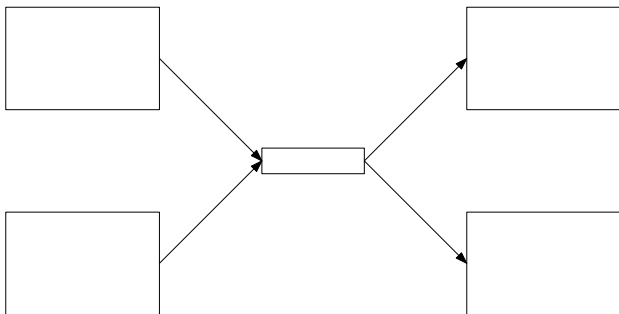
- Basic scenario



- Actors are named and each actor has its own potentially unbounded message queue, typically called mailbox
- Many-to-one messaging, but replies are often possible (ack)
- Simulating several queues often possible

Channels

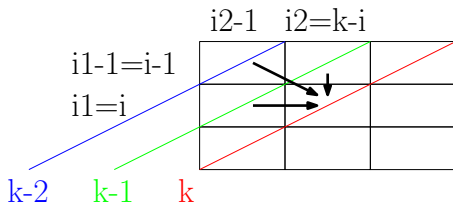
- Basic scenario



- Channels are named and typically have zero-size buffers (synchronous rendezvous)
- Many-to-many messaging, other configurations often possible, e.g non-zero buffer size
- Theoretically equivalence: actors vs channels

Assignment side-bar – rolling diag wave NW \rightarrow SW

- Basic computation



- Using one array, a:

```

1 a.[i1, i2] <- if s1.[i1-1] = s2.[i2-1]
2               then a.[i1-1, i2-1] + 1
3               else max a.[i1, i2-1] a.[i1-1, i2]

```

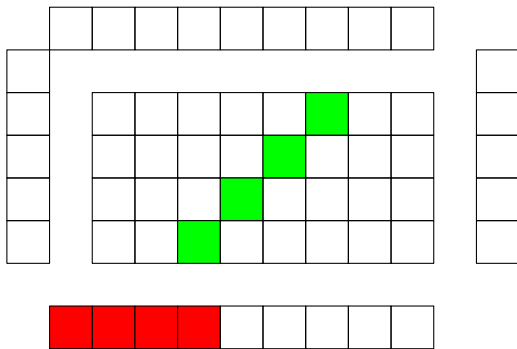
- Recycling three vectors, d0 (blue), d1 (green), d2 (red):

```

1 d2.[i] <- if s.[i-1] = s2.[k-i]
2           then d0.[i-1] + 1
3           else max d1.[i-1] d1.[i]

```

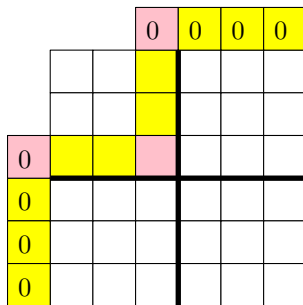
Assignment side-bar – rolling diag wave and sentinels



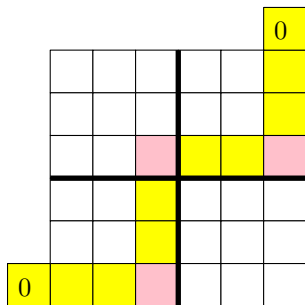
Assignment side-bar – sentinels and bands

0	0	0	0	0	0	0
0						
0						
0						
0						
0						
0						
0						

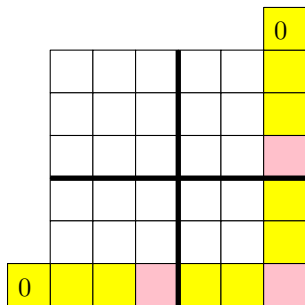
Assignment side-bar – sentinels and bands



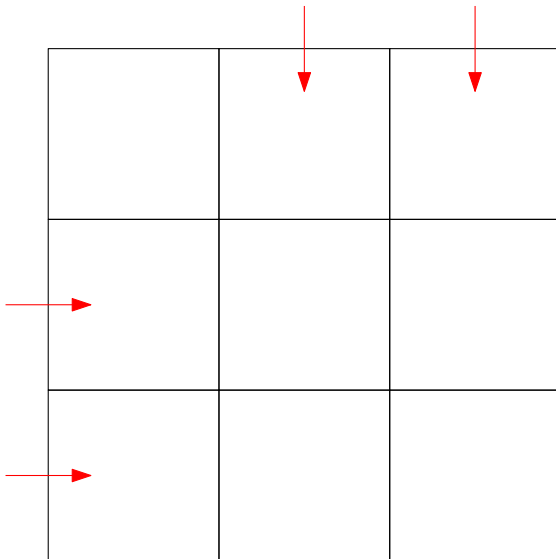
Assignment side-bar – sentinels and bands



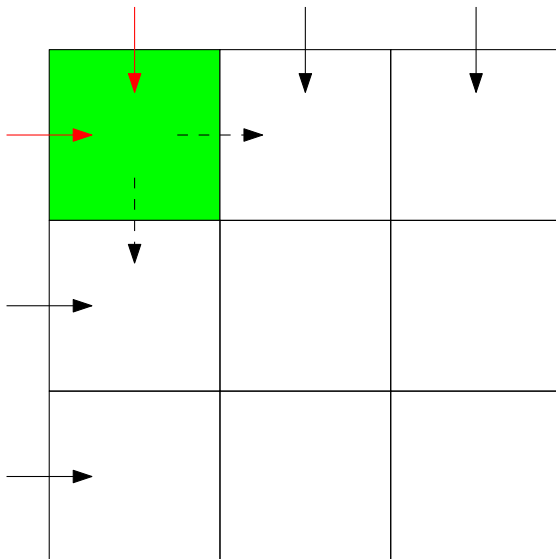
Assignment side-bar – sentinels and bands



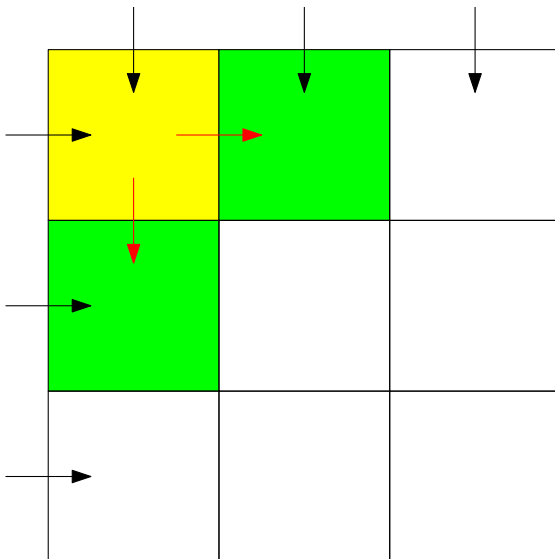
Assignment side-bar – actors



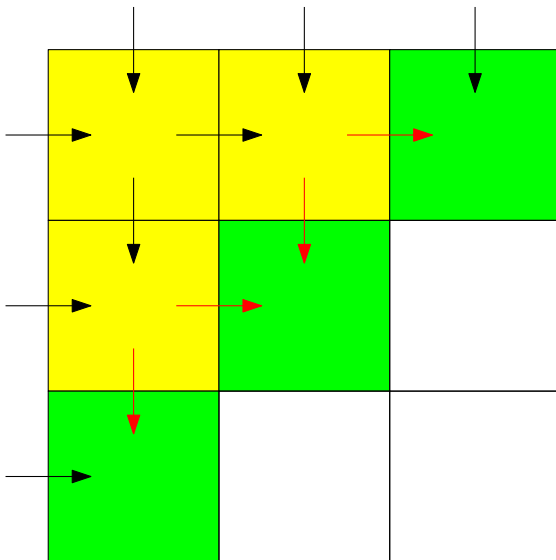
Assignment side-bar – actors



Assignment side-bar – actors



Assignment side-bar – actors



Assignment side-bar – challenges

- Diagonal wave by recycling just two 1D vectors (not three)?
- Just one 1D vector of actors (not a 2D array)?