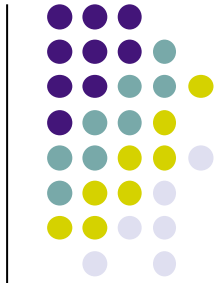
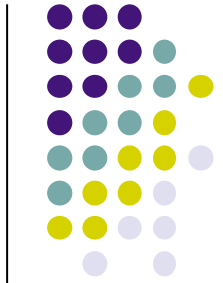
Deterministic dataflow techniques and semantics



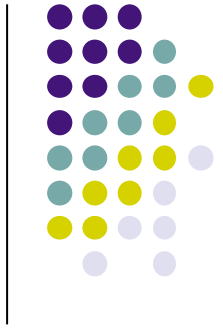
- Concurrency transparency
  - Adding threads to make a program more incremental, without changing the result
- A **for** loop abstraction that collects results
  - Using cells to build concurrency abstractions
- Multi-agent programming
  - Sieve of Eratosthenes: dynamically building a pipeline of concurrent agents
  - Digital logic simulation: using higher-order programming together with deterministic dataflow
- Thread semantics
  - Extending the abstract machine with multiple semantic stacks

Concurrency transparency



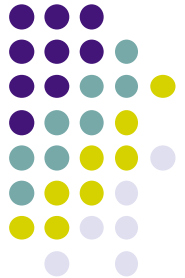
- We saw that multi-agent programs are deterministic
  - Their nondeterminism is not observable
  - The agent Trans with input 1|2|3|_ always outputs 1|4|9|_
- In these programs, concurrency does not change the result but only the order in which computations are done (that is, when the result is calculated)
  - It is possible to add threads at will to a program without changing the result (we call this concurrency transparency)
  - The only effect of added threads is to make the program more incremental (to remove roadblocks)
- Concurrency transparency is only true of declarative paradigms
  - It is no longer true when using cells and threads together (Java!)

Example of transparency (1)



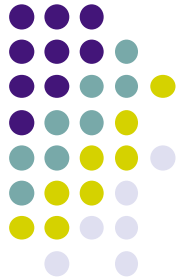
```
fun {Map Xs F}
   case Xs
   of nil  then nil
   [] X|Xr then
      {F X} | {Map Xr F}
   end
end
```

Example of transparency (2)

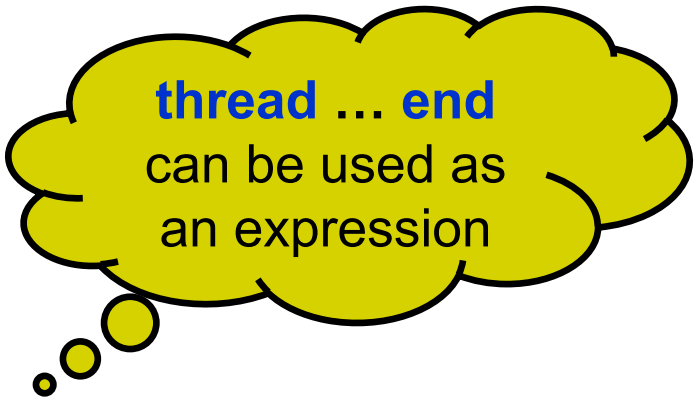


```
fun {CMap Xs F}
   case Xs
   of nil  then nil
   [] X|Xr then
      thread {F X} end | {CMap Xr F}
   end
end
```

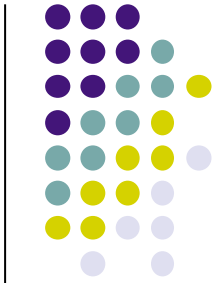
Example of transparency (3)



```
fun {CMap Xs F}
   case Xs
   of nil  then nil
   [] X|Xr then
      thread {F X} end | {CMap Xr F}
   end
end
```



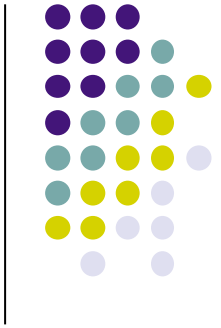
Example of transparency (4)



```
fun {CMap Xs F}
   case Xs
   of nil  then nil
   [] X|Xr then
      thread {F X} end | {CMap Xr F}
   end
end
```

- What happens when we execute:
  ```
  declare F
  {Browse {CMap [1 2 3 4] F}}
  ```

Example of transparency (5)

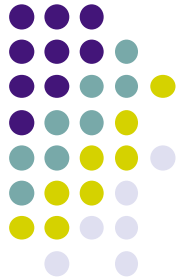


```
fun {CMap Xs F}
   case Xs
   of nil  then nil
   [] X|Xr then
      thread {F X} end | {CMap Xr F}
   end
end
```

```
declare F
{Browse {CMap [1 2 3 4] F}}
```

- The Browser displays [_ _ _ _]
  - CMap calculates a list with unbound variables
  - The new threads wait until F is bound
- What would happen if {F X} was not in its own thread?
  - Nothing would be displayed! The CMap call would block.

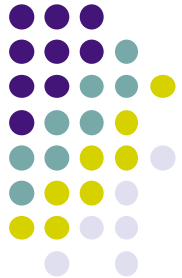
Example of transparency (6)



```
fun {CMap Xs F}
   case Xs
   of nil  then nil
   [] X|Xr then
      thread {F X} end | {CMap Xr F}
   end
end
```

- What happens when we add:
  F = **fun** {$ X} X+1 **end**

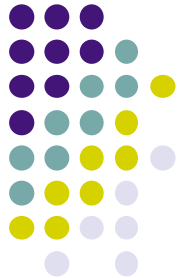
Example of transparency (7)



```
fun {CMap Xs F}
   case Xs
   of nil  then nil
   [] X|Xr then
      thread {F X} end | {CMap Xr F}
   end
end
```

- The Browser displays [2 3 4 5]
- With or without the thread creation, the final result is always [2 3 4 5]

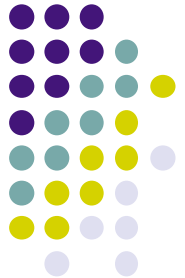
“Concurrency for dummies”



- Threads can be added at will to a functional program without changing the result
- Therefore it is very easy to take a functional program and make it concurrent
- It suffices to insert **thread** … **end** in those places that need concurrency

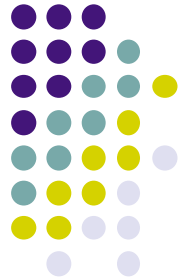
- Warning: concurrency for dummies does not work in a program with explicit state (= with cells)!
  - For example, it does not work in Java
  - En Java, concurrency is handled with the concept of a monitor, which coordinates how multiple threads access an object. This is *much more complicated* than deterministic dataflow.

Why does it work? (1)



```
fun {Fib X}
    if X==0 then 0
    elseif X==1 then 1
    else
        thread {Fib X-1} end + {Fib X-2}
    end
end
```

Why does it work? (2)

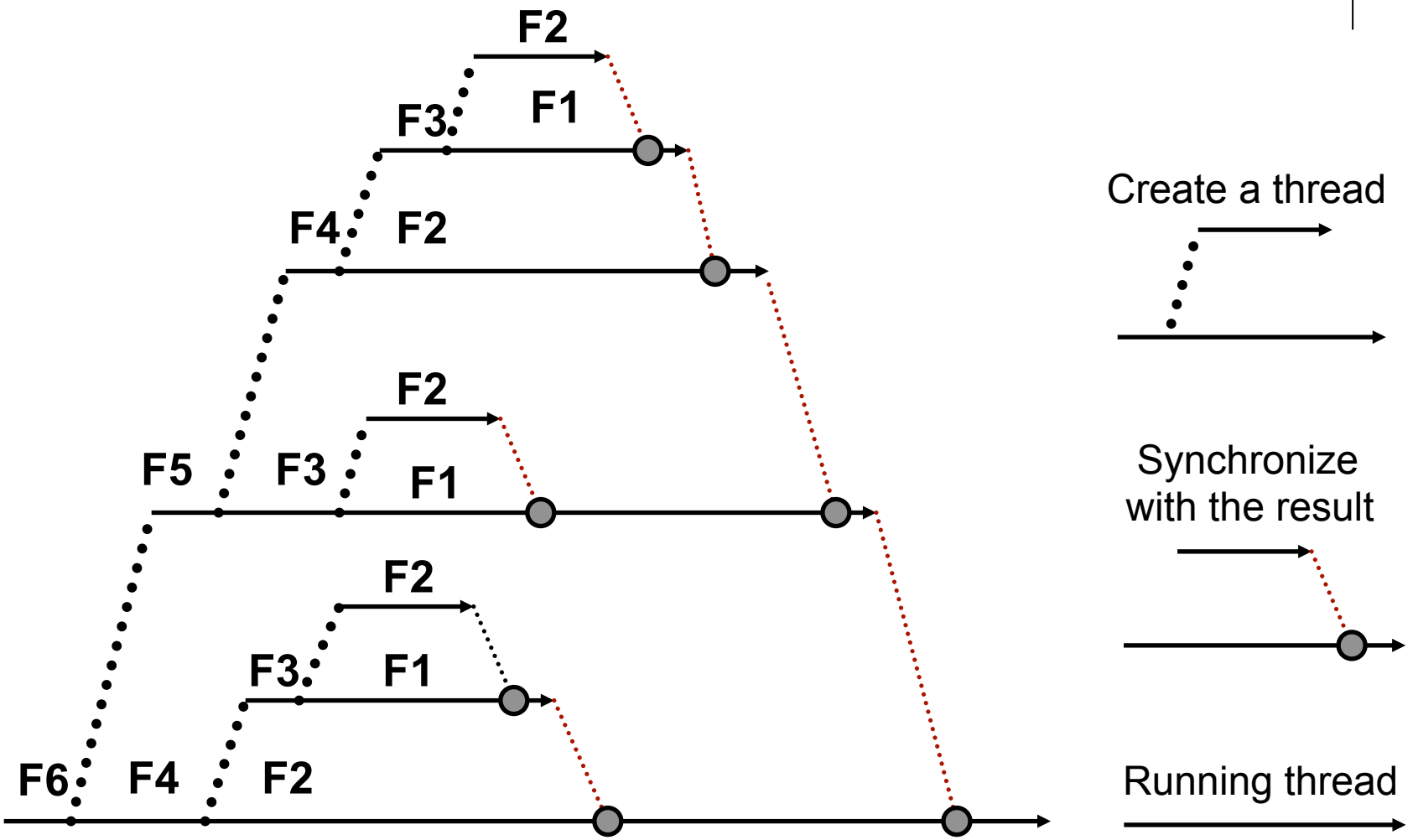


```
fun {Fib X}
   if X==0 then 0 elseif X==1 then 1
   else F1 F2 in
      F1 = thread {Fib X-1} end
      F2 = {Fib X-2}
      F1 + F2
   end
end
```

Dataflow dependency

It works because variables can only be bound to one value

Execution of {Fib 6}

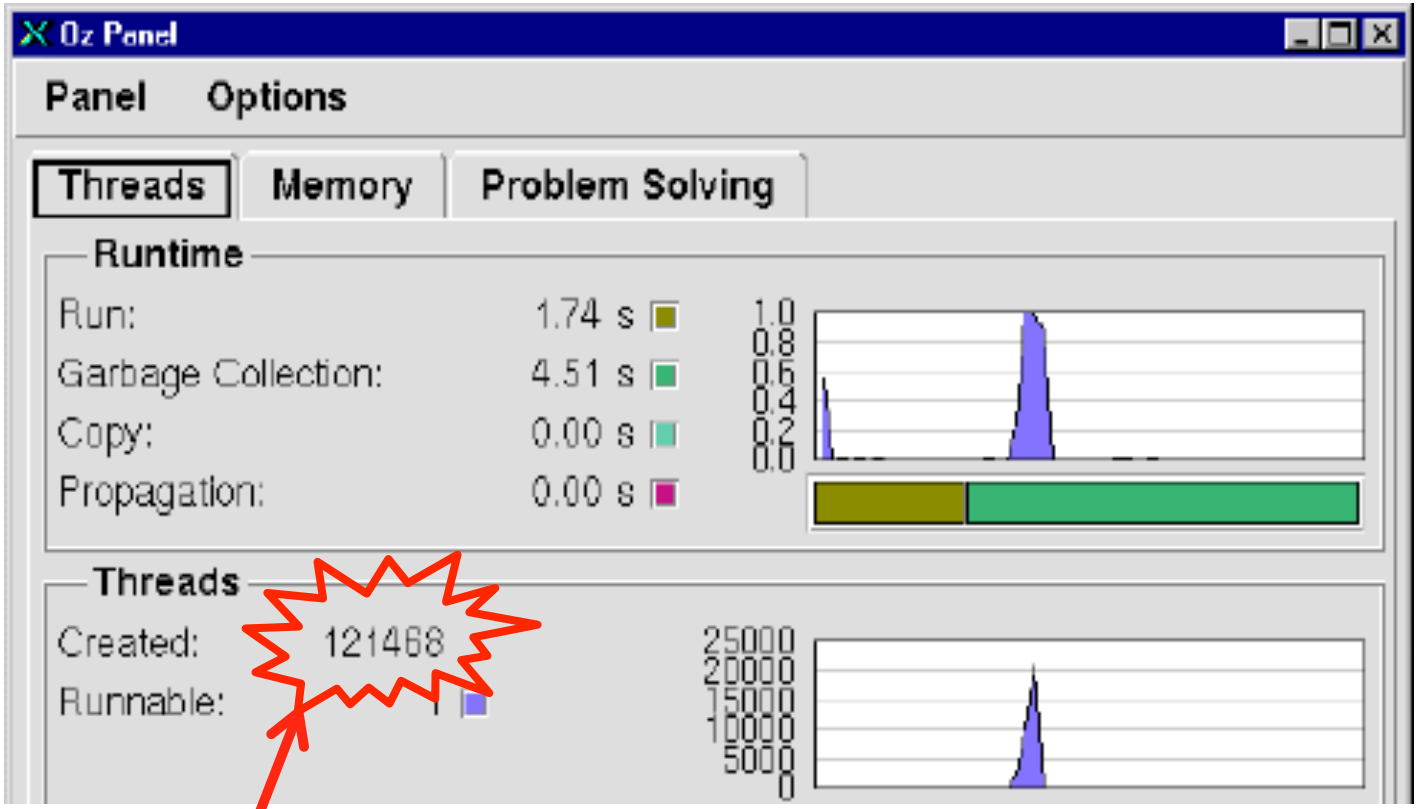


Create a thread

Synchronize with the result

Running thread

Observing the execution of Fib



Total number of threads created since system startup

Oz Compiler Panel (in Oz menu)