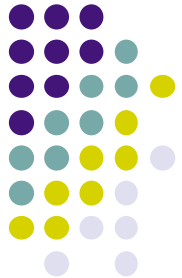
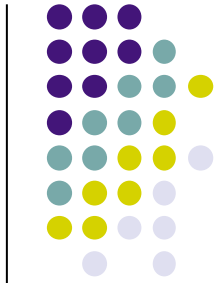
Threads



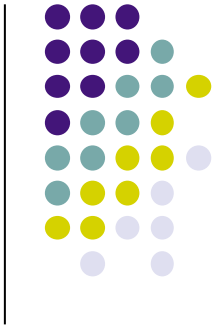
- We add a language concept to support concurrent activities
  - In a program, an activity is a sequence of executing instructions
  - We add this concept to the language and call it a thread
- Each thread is sequential
- Each thread is independent of the others
  - There is no order defined between different threads
  - The system executes all threads using interleaving semantics: it is as if only one thread executes at a time, with execution stepping from one thread to another
  - The system guarantees that each thread receives a fair share of the computational capacity of the processor
- Two threads can communicate if they share a variable
  - For example, the variable corresponding to identifier X in the example we just saw

Thread creation



- Creating a thread in Oz is simple
- Any instruction can be executed in a new thread:
  thread <s> **end**
- For example:
  declare X
  thread {Browse X+1} **end**
  thread X=1 **end**
- What does this small program do?
  - Several executions are possible, but they all eventually arrive at the same result: 2 is displayed!

A small program (1)

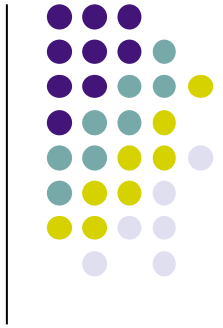


- A small program with several threads:

```
declare X0 X1 X2 X3 in
thread X1=1+X0 end
thread X3=X1+X2 end
{Browse [X0 X1 X2 X3]}
```

- The Browser displays [X0 X1 X2 X3]
  - The variables are all unbound
  - The Browser also uses dataflow:
    when a variable is bound, the display is updated

A small program (2)

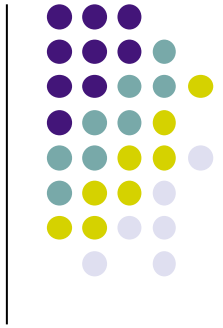


- A small program with several threads:

```
declare X0 X1 X2 X3 in
thread X1=1+X0 end
thread X3=X1+X2 end
{Browse [X0 X1 X2 X3]}
```

- Two threads will wait:
  - X1=1+X0 waits (since X0 is unbound)
  - X3=X1+X2 waits (since X1 and X2 are unbound)

A small program (3)

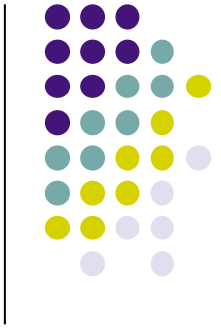


- A small program with several threads:

```
declare X0 X1 X2 X3 in
thread X1=1+X0 end
thread X3=X1+X2 end
{Browse [X0 X1 X2 X3]}
```

- Let's bind one variable
  - Bind X0=4

A small program (4)

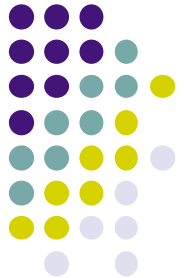


- A small program with several threads:

```
declare X0 X1 X2 X3 in
thread X1=1+X0 end
thread X3=X1+X2 end
{Browse [X0 X1 X2 X3]}
```

- Let's bind one variable
  - Bind X0=4
    - The first thread executes and binds X1=5
    - The Browser displays [4 5 X2 X3]

A small program (5)

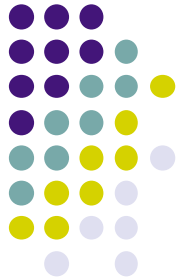


- A small program with several threads:

```
declare X0 X1 X2 X3 in
thread X1=1+X0 end  % terminated
thread X3=X1+X2 end
{Browse [X0 X1 X2 X3]}
```

- The second thread is still waiting
  - Because X2 is still unbound

A small program (6)

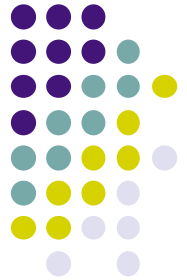


- A small program with several threads:

```
declare X0 X1 X2 X3 in
thread X1=1+X0 end  % terminated
thread X3=X1+X2 end
{Browse [X0 X1 X2 X3]}
```

- Let's do another binding
  - Bind X2=7
    - The second thread executes and binds X3=12
    - The Browser displays [4 5 7 12]

The Browser is a dataflow program



- The Browser executes with its own threads
- For each unbound variable that is displayed, there is a thread in the Browser that waits until the variable is bound
  - When the variable is bound, the display is updated
- This does not work with cells
  - The Browser targets the dataflow paradigm
  - The Browser does not look at the content of cells, since they do not execute with dataflow