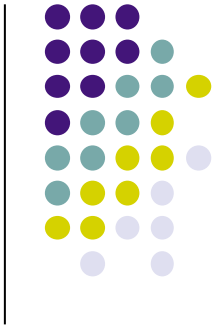
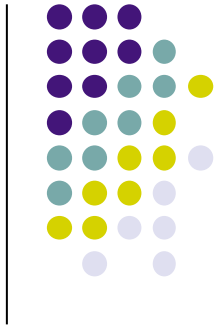
Semantics of cells (1)



- We have extended the kernel language with cells
  - Let us now extend the abstract machine to explain how cells execute
- There are now two stores in the abstract machine:
  - Single-assignment store (contains variables: immutable store)
  - Multiple-assignment store (contains cells: mutable store)
- A cell is a pair of two variables
  - The first variable is bound to the name of the cell (a constant)
  - The second variable is the cell's content
- Assigning a cell to a new content
  - The pair is changed: the second variable in the pair is replaced by another variable (the first variable stays the same)

Warning: The variables do *not* change! The single-assignment store is unchanged when a cell is assigned.

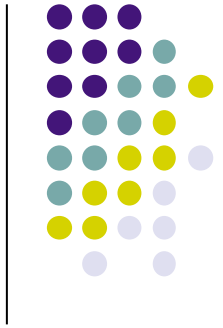
Semantics of cells (2)



- The full store $\sigma = \sigma_1 \cup \sigma_2$ has two parts:
  - Single-assignment store (contains variables)
    $\sigma_1 = \{t, u, v, x=\xi, y=\zeta, z=10, w=5\}$
  - Multiple-assignment store (contains pairs)
    $\sigma_2 = \{x{:}t, y{:}w\}$

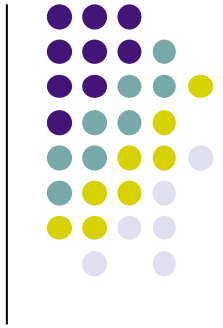
- In σ_2 there are two cells, *x* and *y*
  - The name of *x* is the constant ξ, the name of *y* is ζ
  - The operation X:=Z changes *x*:*t* into *x*:*z*
  - The operation @Y returns the variable *w*
    (assuming the environment $\{X \rightarrow x, Y \rightarrow y, Z \rightarrow z, W \rightarrow w\}$)

Imperative paradigm



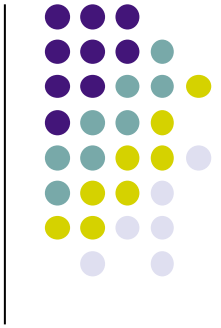
- By adding cells, we have left the functional paradigm and entered the imperative paradigm
  - Imperative paradigm = functional paradigm + cells
- The imperative paradigm allows programs to express and observe growth and change
  - This gives new ways of thinking that were not possible in the functional paradigm
- The imperative paradigm is the foundation of object-oriented programming (OOP)
  - OOP has new ways of structuring programs that are essential for building large systems

Kernel language of the imperative paradigm



- <s> ::= **skip**
  | <s>$_1$ <s>$_2$
  | **local** <x> **in** <s> **end**
  | <x>$_1$=<x>$_2$
  | <x>=<v>
  | **if** <x> **then** <s>$_1$ **else** <s>$_2$ **end**
  | {<x> <y>$_1$ … <y>$_n$}
  | **case** <x> **of** <p> **then** <s>$_1$ **else** <s>$_2$ **end**
  | {NewCell <y> <x>}
  | <x>:=<y>
  | <y>=@<x>
- <v> ::= <number> | <procedure> | <record>
- <number> ::= <int> | <float>
- <procedure> ::= **proc** {$ <x>$_1$ … <x>$_n$} <s> **end**
- <record>, <p> ::= <lit> | <lit>(<f>$_1$:<x>$_1$ … <f>$_n$:<x>$_n$)

Kernel language of the imperative paradigm



- <s> ::= **skip**
  | <s>$_1$ <s>$_2$
  | **local** <x> **in** <s> **end**
  | <x>$_1$=<x>$_2$
  | <x>=<v>
  | **if** <x> **then** <s>$_1$ **else** <s>$_2$ **end**
  | {<x> <y>$_1$ … <y>$_n$}
  | **case** <x> **of** <p> **then** <s>$_1$ **else** <s>$_2$ **end**
  | {NewCell <y> <x>}
  | {Exchange <x> <y> <z>}
- <v> ::= <number> | <procedure> | <record>
- <number> ::= <int> | <float>
- <procedure> ::= **proc** {$ <x>$_1$ … <x>$_n$} <s> **end**
- <record>, <p> ::= <lit> | <lit>(<f>$_1$:<x>$_1$ … <f>$_n$:<x>$_n$)

Second version
Both versions are equally expressive (since Exchange can be expressed with @ and := and vice versa), but the second version is more convenient for concurrent programming

<y>=@<x> and <x>:=<z>
(*atomically* : as one operation)