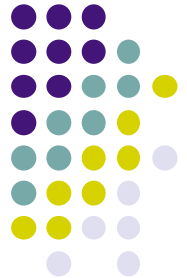
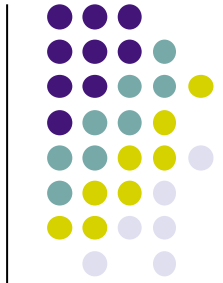
Procedure definition and procedure call



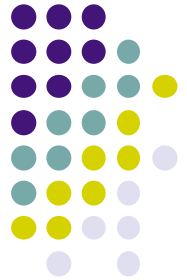
- Procedure definition and call are very important instructions, since they are the foundation of data abstraction
  - Higher-order programming
  - Layered program organization
  - Encapsulation
  - Object-oriented programming
  - Abstract data types
- This is why we will look at them separately

Procedure semantics



- Procedure definition
  - Create the contextual environment
  - Store the procedure value, which contains both procedure code and contextual environment
- Procedure call
  - Create a new environment by combining two parts:
    - The procedure's contextual environment
    - The formal arguments (identifiers in the procedure definition), which are made to reference the actual argument values
  - Execute the procedure body with this new environment
- We first give an example execution to show what the semantic rules have to do

Procedure example (1)



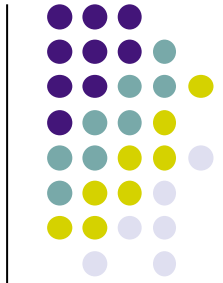
```
local Z in
   Z=1
   proc {P X Y} Y=X+Z end
end
```

- The free identifiers of the procedure (here, just Z) are the ones declared outside the procedure
- When executing P, the identifier Z must be known
- Z is part of the procedure's contextual environment, which must be part of the procedure's definition

Procedure example (2)

```
local P in
   local Z in
      Z=1
      proc {P X Y} Y=X+Z end  % CE_P = {Z→z}
   end
   local B A in
      A=10
      {P A B}                 % P's body Y=X+Z must do b=a+z
      {Browse B}              % Therefore: E_P = {Y→b, X→a, Z→z}
   end
end
```

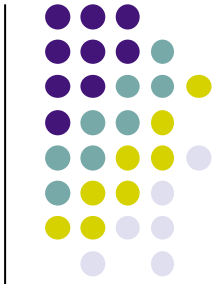
Semantic rule for procedure definition



- Semantic instruction:
  (<x>=**proc** {$ $\langle x \rangle_1 \ldots \langle x \rangle_n$} <s> **end**, E)
  - Formal arguments:
    $\langle x \rangle_1, \ldots, \langle x \rangle_n$
  - Free identifiers in <s>:
    $\langle z \rangle_1, \ldots, \langle z \rangle_k$
  - Contextual environment:
    $CE = E_{|\langle z \rangle 1, \ldots, \langle z \rangle k}$ (restriction of E)

- Create the following binding in memory:
  x=(**proc** {$ $\langle x \rangle_1 \ldots \langle x \rangle_n$} <s> **end**, CE)

Semantic rule for procedure call (1)

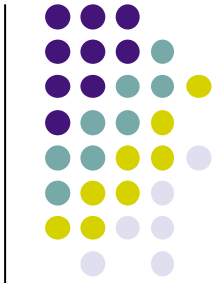


- Semantic instruction:

  $(\{\langle x\rangle\ \langle y\rangle_1 \ldots \langle y\rangle_n\}, E)$

- If the activation condition is false ($E(\langle x\rangle)$ unbound)
  - Suspension (wait, do not execute)
- If $E(\langle x\rangle)$ is not a procedure
  - Raise an error condition
- If $E(\langle x\rangle)$ is a procedure with the wrong number of arguments ($\neq n$)
  - Raise an error condition

Semantic rule for procedure call (2)



- Semantic instruction on stack:

  $(\{\langle x\rangle\ \langle y\rangle_1\ \ldots\ \langle y\rangle_n\}, E)$

  with procedure definition in memory:

  $E(\langle x\rangle) = (\textbf{proc}\ \{\$\ \langle z\rangle_1 \ldots \langle z\rangle_n\}\ \langle s\rangle\ \textbf{end}, CE)$

- Put the following instruction on the stack:

  $(\langle s\rangle, CE + \{\langle z\rangle_1 \rightarrow E(\langle y\rangle_1), \ldots, \langle z\rangle_n \rightarrow E(\langle y\rangle_n)\})$