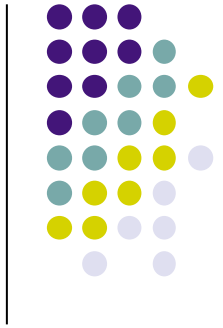
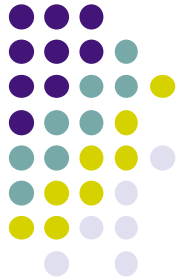
How to handle exceptional situations



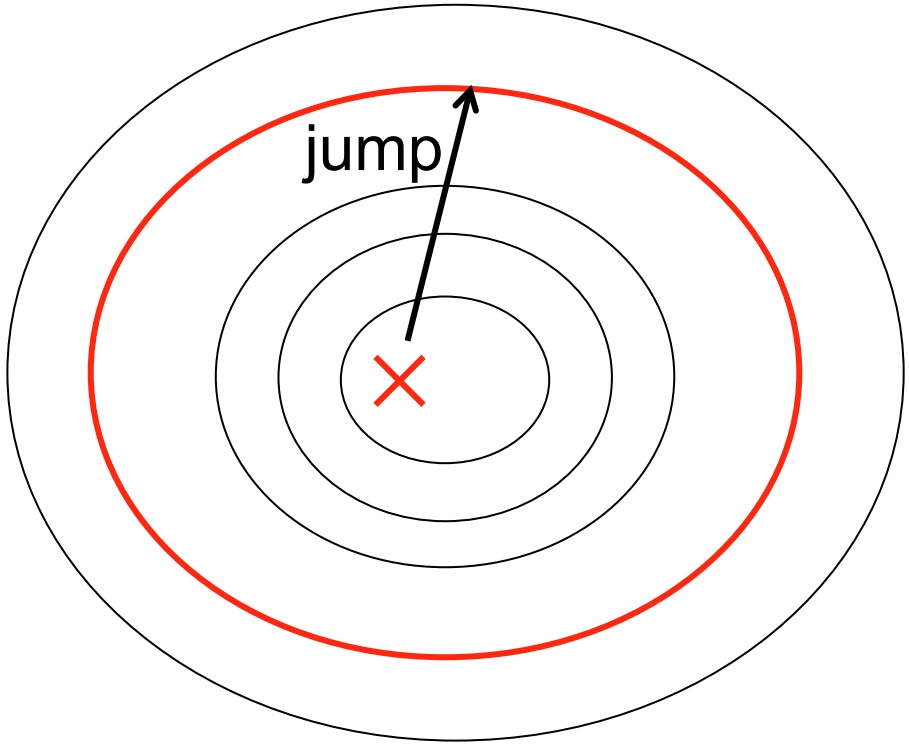
- How can we handle exceptional situations in a program?
  - Such as: division by 0, opening a nonexistent file, and so forth
  - Program errors but also errors from outside the program
  - Things that happen rarely but that must be taken care of
- We add a new programming concept called exceptions
  - We define exceptions and show how they are used
  - We give the semantics of exceptions in the abstract machine
- With exceptions, we can handle exceptional situations without cluttering up the program with rarely used error checking code

The containment principle



- When an error occurs, we would like to be able to recover from the error
- Furthermore, we would like the error to affect as little as possible of the program
- We propose the containment principle:
  - A program is a set of nested execution contexts
  - An error will occur inside an execution context
  - A recovery routine (exception handler) exists at the boundary of an execution context, to make sure the error does not propagate to higher execution contexts

Handling an exception



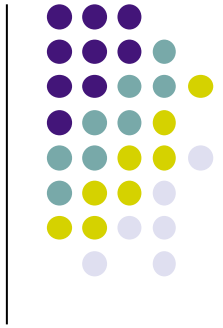
An error that raises an exception

An execution context

The execution context that catches the exception

- An executing program that encounters an error must jump to another part (the exception handler) and give it a reference (the exception) that describes the error

The try and raise instructions

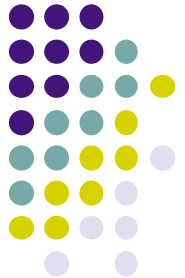


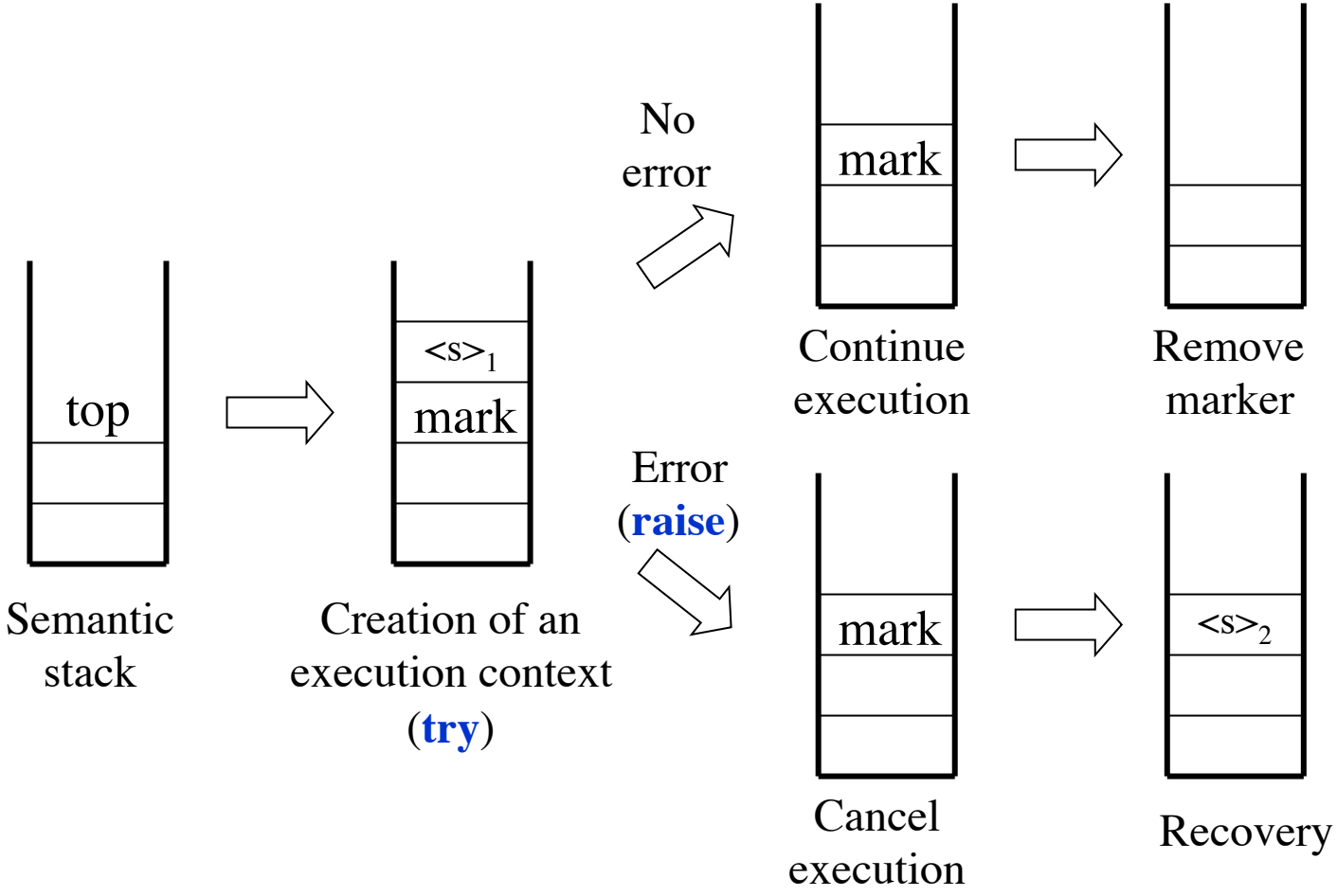
- We introduce two new instructions for handling exceptions:

```
try <s>1 catch <y> then <s>2 end  % Create an execution context
raise <x> end                     % Raise an exception
```

- With the following behavior:
  - **try** puts a "marker" on the stack and starts executing $\langle s \rangle_1$
  - If there is no error, $\langle s \rangle_1$ executes normally and removes the marker when it terminates
  - **raise** is executed when there is an error, which empties the stack up to the marker (the rest of $\langle s \rangle_1$ is therefore canceled)
    - Then $\langle s \rangle_2$ is executed
    - <y> refers to the same variable as <x>
    - The scope of <y> exactly covers $\langle s \rangle_2$

Semantics of exceptions





An execution context

- An execution context is the part of the semantic stack that starts with a marker and continues to the stack top:

```
try ... % Context 1
  try ... % Context 2
    try ... % Context 3
    catch <x> then <s>3 end
      ...
  catch <x> then <s>2 end
    ...
catch <x> then <s>1 end
```