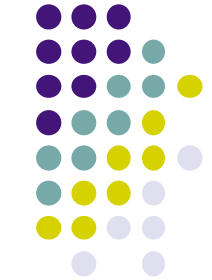
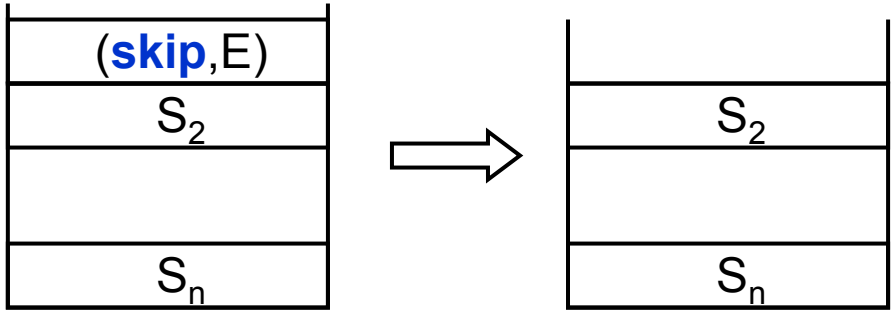
Semantic rules for kernel language instructions



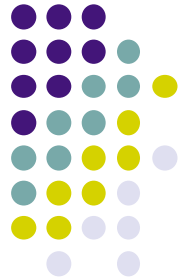
- For each instruction in the kernel language, we will define its rule in the abstract machine
- Each instruction takes one execution state as input and returns one execution state
  - Execution state = semantic stack + memory
- Let's look at three instructions in detail:
  - **skip**
  - $\langle s \rangle_1$ $\langle s \rangle_2$ (sequential composition)
  - **local** <x> **in** <s> **end**
- We will see the others in less detail. You can learn about them in the exercises and in the book.

skip

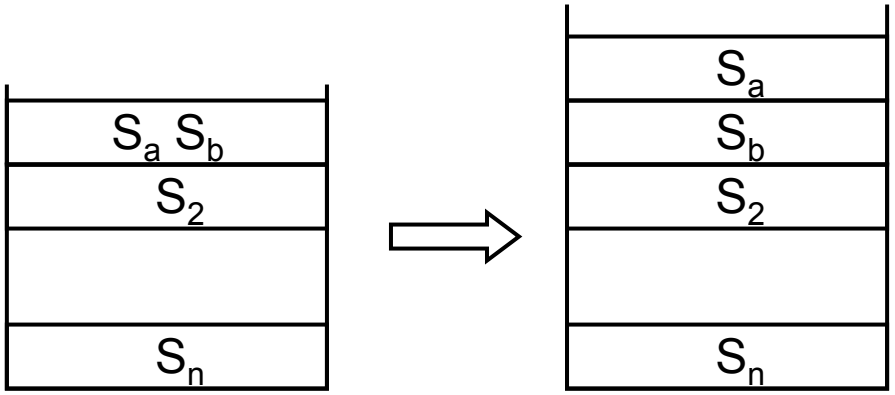
- The simplest instruction
- It does nothing at all!
- Input state: ([(**skip**,E), S_2, …, S_n], σ)
- Output state: ([S_2, …, S_n], σ)
- That's all



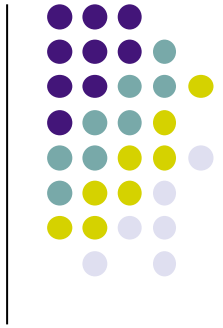
$\langle s \rangle_1 \langle s \rangle_2$ (sequential composition)



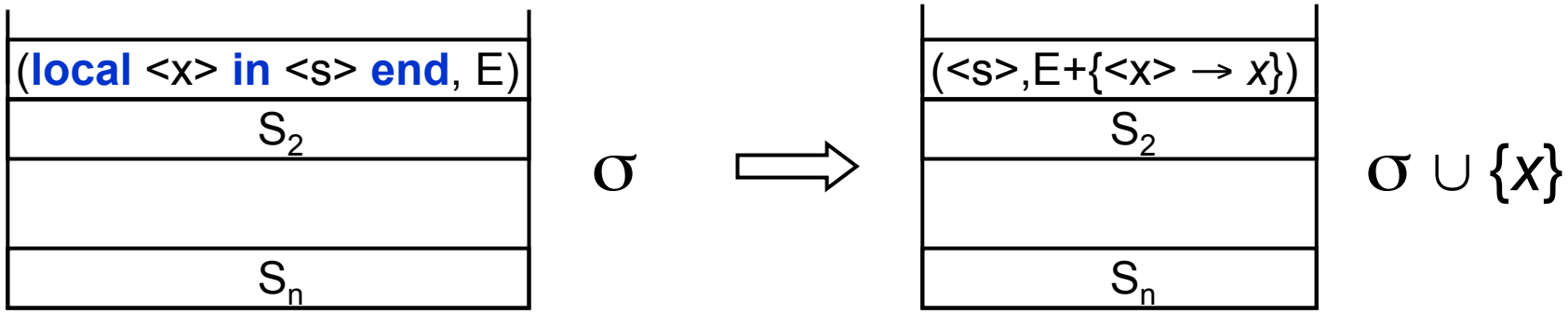
- Almost as simple as **skip**
- The instruction removes the top of the stack and adds two new elements
- Input state: $([(S_a\ S_b), S_2, \ldots, S_n], \sigma)$
- Output state: $([S_a, S_b, S_2, \ldots, S_n], \sigma)$



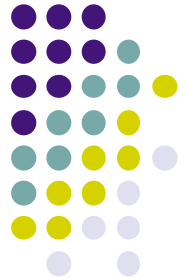
local <x> in <s> end



- Create a fresh new variable x in memory σ
- Add the link {X → *x*} to the environment E (using adjunction)



Some comments on the other instructions



- <x>=<v> (value creation + assignment)
  - Note: when <v> is a procedure, you have to create the contextual environment
- **if** <x> **then** $\langle s \rangle_1$ **else** $\langle s \rangle_2$ **end** (conditional)
  - Note: if <x> is unbound, the instruction will wait ("block") until <x> is bound to a value
  - The activation condition: "<x> is bound to a value"
- **case** <x> **of** <p> **then** $\langle s \rangle_1$ **else** $\langle s \rangle_2$ **end**
  - Note: **case** statements with more patterns are built by combining several kernel instructions
- {<x> $\langle y \rangle_1$ ... $\langle y \rangle_n$}
  - Note: since procedure definition and procedure call are the foundation of data abstraction, we will take a special look!