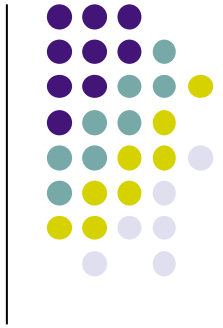
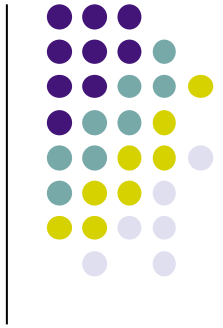
Thread semantics (1)

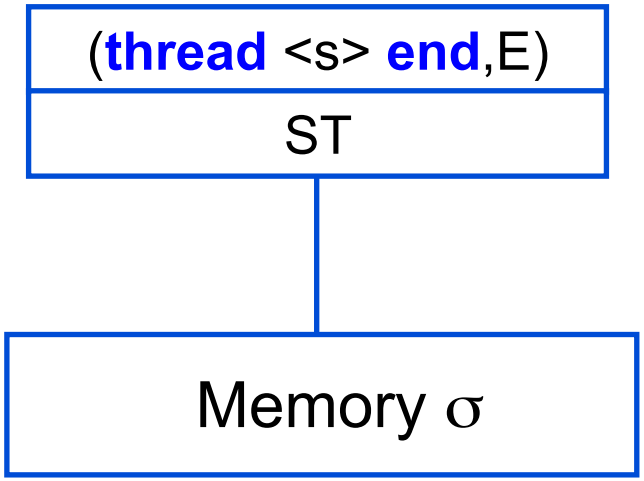


- We extend the abstract machine with threads
- Each thread has one semantic stack
  - The instruction **thread** <s> **end** creates a new stack
  - All stacks share the same memory
- There is one sequence of execution states, and threads take turns executing instructions
  - $(MST_1,\sigma_1) \rightarrow (MST_2,\sigma_2) \rightarrow (MST_3,\sigma_3) \rightarrow ...$
  - MST is a multiset of semantic stacks
  - This is called interleaving semantics

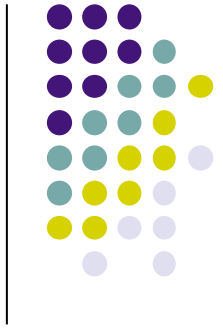
Thread semantics (2)



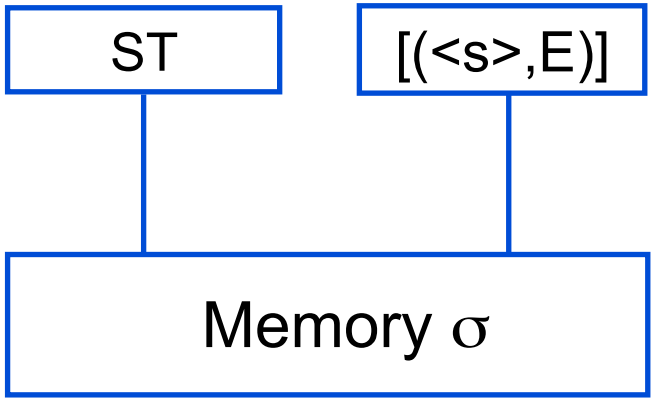
A semantic stack that is about to create a thread



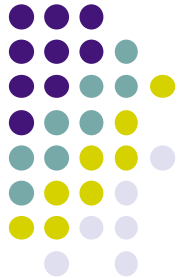
Thread semantics (3)



We now have two stacks!



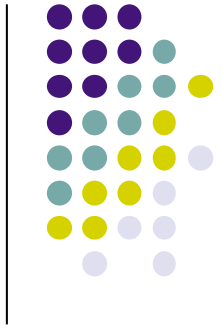
Why interleaving semantics?



- What happens when activities execute ”at the same time”?
- We can imagine that all threads execute in parallel, each with its own processor but all sharing the same memory
  - We have to be careful to understand what happens when threads operate simultaneously on the same memory word
  - If the threads share the same processor, then this problem is avoided (interleaving semantics)

- Interleaving semantics is much easier to reason about than truly concurrent semantics
  - Truly concurrent semantics also models the case where threads ”step on each others’ toes”, but usually this is not needed, since the hardware is careful to keep this from happening
  - For example, in a multicore processor the cache coherence protocol avoids simultaneous operations on one memory word

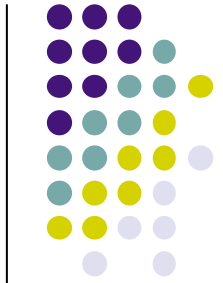
Order of execution states



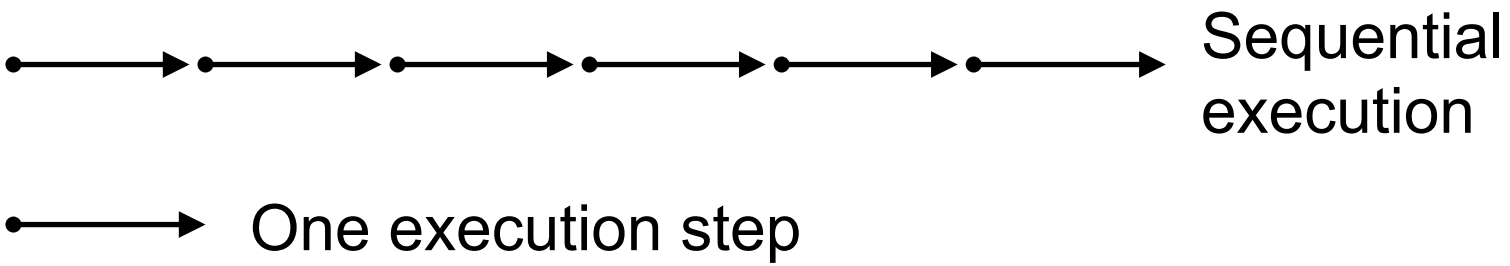
- In a sequential program, execution states are in a total order
  - Total order = when comparing any two execution states, one must happen before the other

- In a concurrent program, execution states of the same thread are in a total order
  - The execution states of the complete program (with multiple threads) are in a partial order
  - Partial order = when comparing any two execution states, there might be no order between them (either may happen first)
- In a concurrent program, many executions are compatible with the partial order
  - In the actual execution, the scheduler chooses one

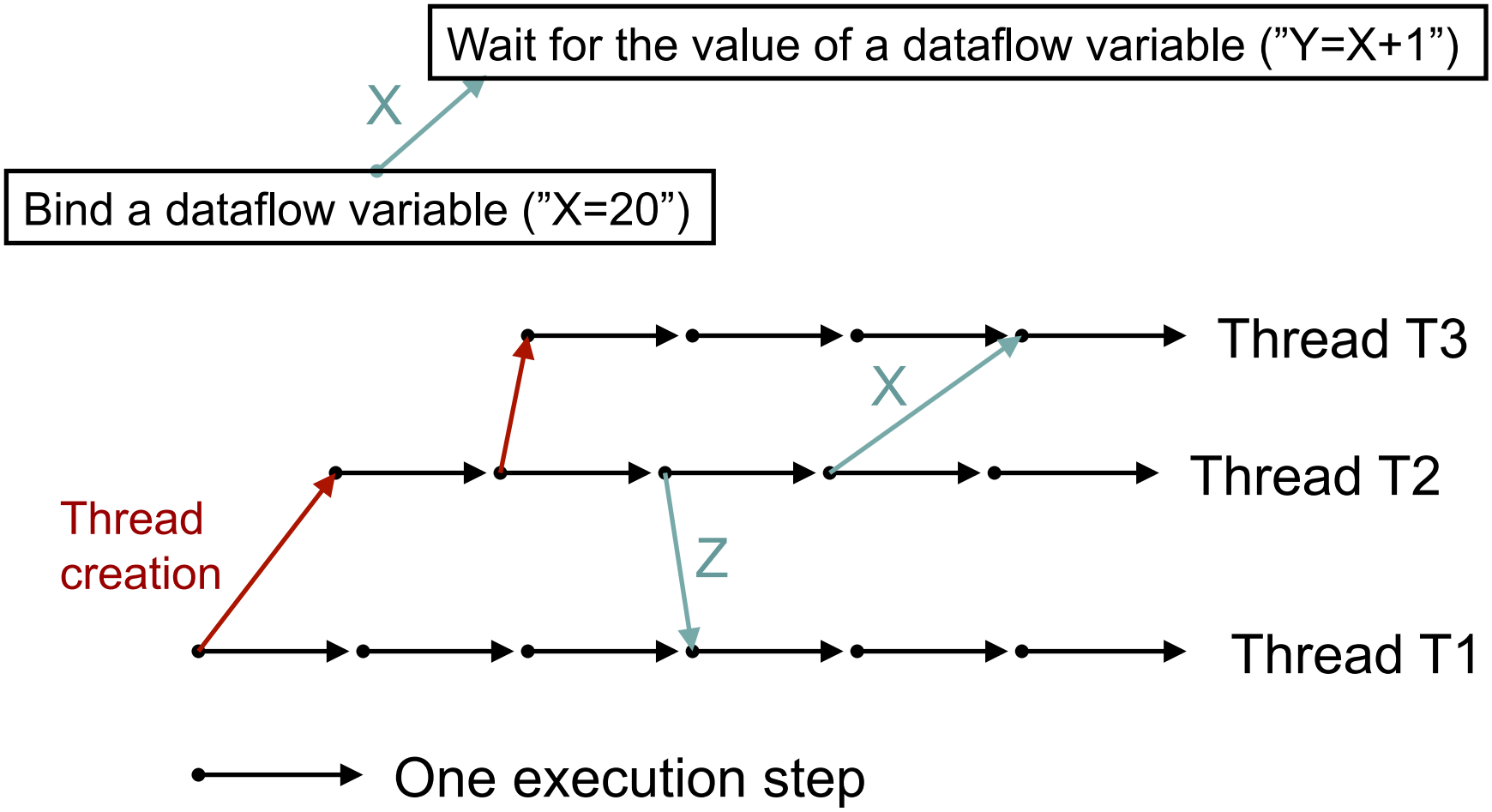
Total order in a sequential program



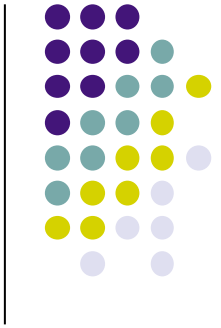
- In a sequential program, execution states are in a total order
- A sequential program has one thread
- Earlier paradigms always had this situation



Partial order in a concurrent program



Partial order in a concurrent program



- In a concurrent program, many executions are compatible with the partial order
- The scheduler chooses one of them during the actual execution (nondeterminism)