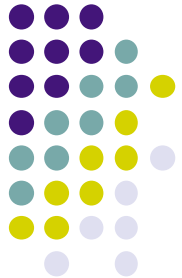
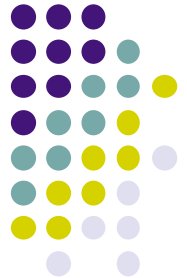
Deterministic concurrency



- Each thread in a deterministic dataflow program always executes the same instructions in the same order
  - This is true even though the threads can vary their relative speeds from one execution to the next
  - Speeds can vary because of input/output, hardware interrupts, cache misses, and other sources of timing changes

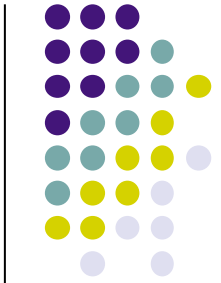
- A deterministic dataflow program always gives the same outputs for the same inputs, despite variations in thread speeds
  - We say the program has no observable nondeterminism (no race conditions)
  - This is a major advantage of the deterministic dataflow paradigm that is not shared by the two other paradigms

Nondeterminism and the scheduler



- Nondeterminism is the ability of the system to make decisions that are visible by a running program
  - The application programmer does not make the decisions
  - The decisions can vary from one execution to the next
- The scheduler is the part of the system that decides at each moment which thread to execute
  - This decision is called nondeterminism
- Nondeterminism is a property of any concurrent system
  - It must be, since the concurrent activities are independent
  - A crucial part of any concurrent program is how to manage its nondeterminism

Example of nondeterminism (1)

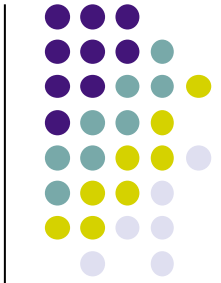


- What does the following program do?

```
declare X
thread X=1 end
thread X=2 end
```

- The execution order of the two threads is not fixed
  - X will be bound to 1 or 2, we don't know which
  - The other thread will have an error (raise an exception)
    - A variable cannot be assigned to two values
- This is an example of nondeterminism
  - A *choice* made by the system during execution
  - The system is free to choose one or the other

Example of nondeterminism (2)

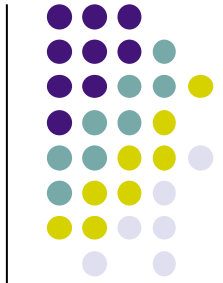


- What does the following program do?

```
declare X={NewCell 0}
thread X:=1 end
thread X:=2 end
```

- The execution order of the two threads is not fixed
  - Cell X will first be bound to one value, then to the other
  - When both threads terminate, X will contain 1 or 2, we don't know which
  - This time there is no error
- This is an example of nondeterminism
  - A *choice* made by the system during execution

Example of nondeterminism (3)

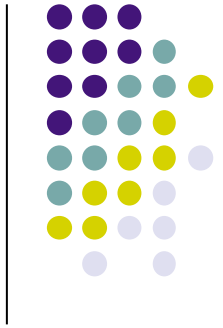


- What does the following program do?

```
declare X={NewCell 0}
thread X:=1 end
thread X:=1 end
```

- It makes a choice, just like the previous program
  - But in this case, the final results are the same
- This is still nondeterminism!
  - The important point is the choice: the running program still sees a difference in the threads' execution order
  - Maybe the results are the same by accident (depending on the computations done), but the choice remains

Managing nondeterminism



- Nondeterminism *must* always be managed
  - It should not affect program correctness
  - The most complicated case is when threads and cells are used in the same program (see previous example)
  - Unfortunately, this is exactly how many languages handle concurrency

- Deterministic dataflow has a major advantage
  - The result of a program is always the same (*except* if there is a programming error – if a thread raises an exception)
  - The nondeterminism of the scheduler does not affect the result