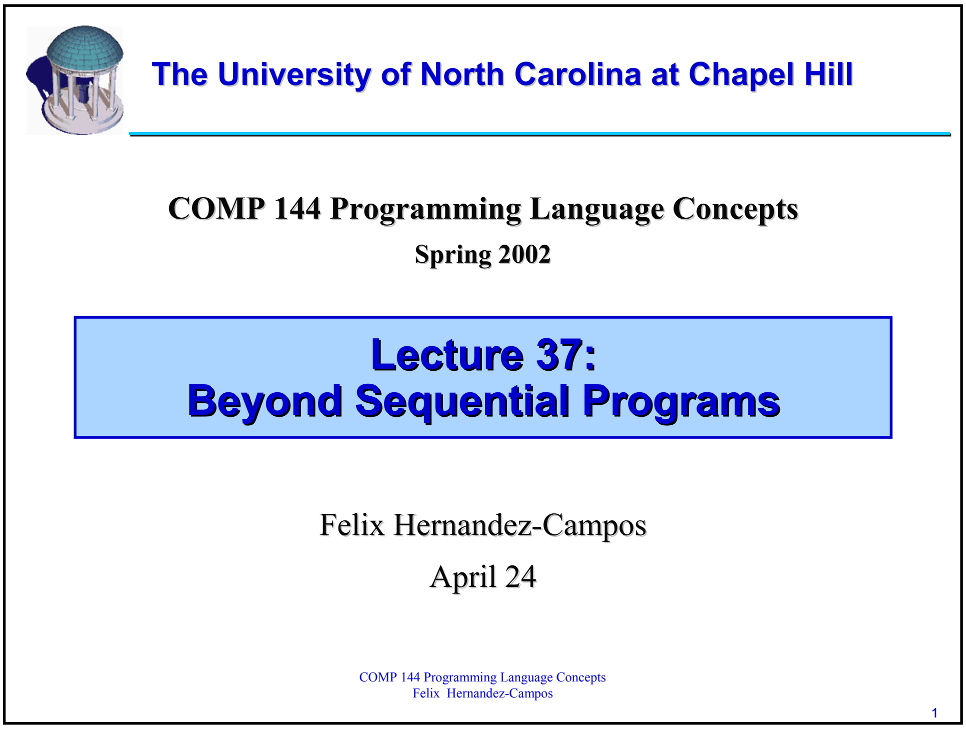


The University of North Carolina at Chapel Hill

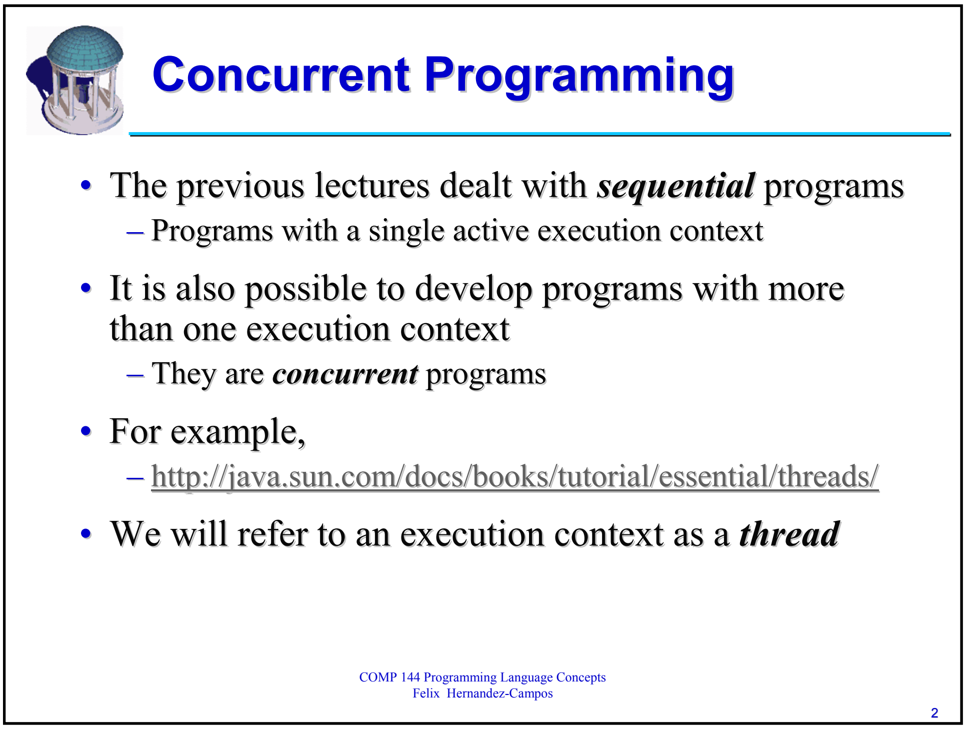
COMP 144 Programming Language Concepts

Spring 2002

Lecture 37: Beyond Sequential Programs

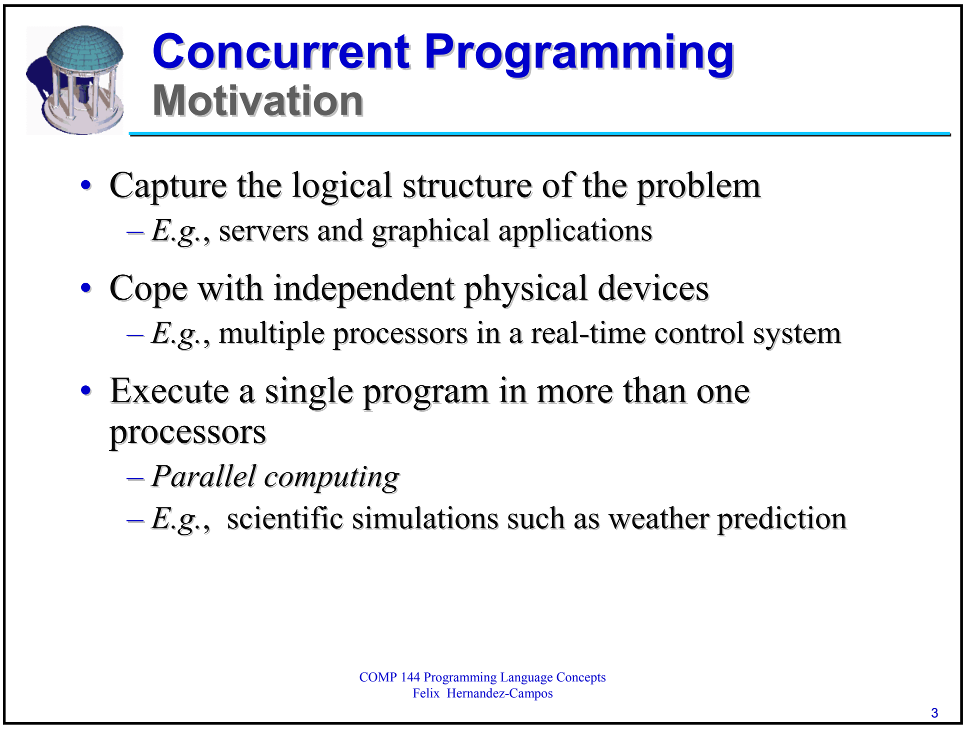
Felix Hernandez-Campos

April 24



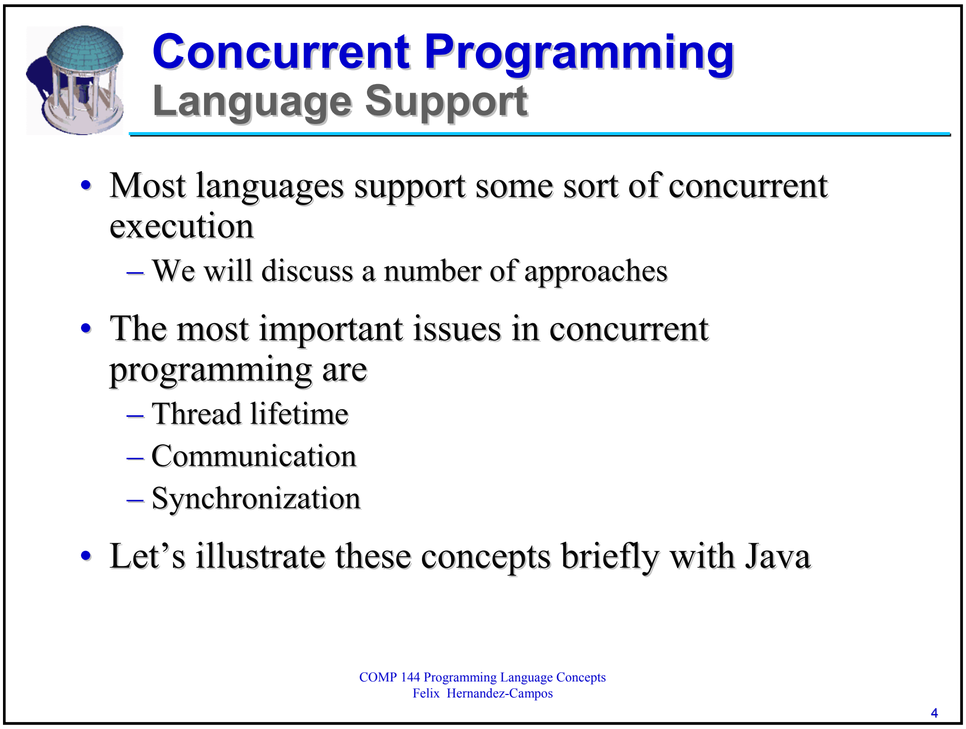
Concurrent Programming

- The previous lectures dealt with ***sequential*** programs
  - Programs with a single active execution context
- It is also possible to develop programs with more than one execution context
  - They are ***concurrent*** programs
- For example,
  - http://java.sun.com/docs/books/tutorial/essential/threads/
- We will refer to an execution context as a ***thread***



Concurrent Programming
Motivation

- Capture the logical structure of the problem
  - *E.g.*, servers and graphical applications
- Cope with independent physical devices
  - *E.g.*, multiple processors in a real-time control system
- Execute a single program in more than one processors
  - *Parallel computing*
  - *E.g.*, scientific simulations such as weather prediction

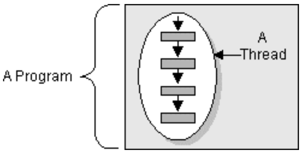


Concurrent Programming
Language Support

- Most languages support some sort of concurrent execution
  - We will discuss a number of approaches
- The most important issues in concurrent programming are
  - Thread lifetime
  - Communication
  - Synchronization
- Let’s illustrate these concepts briefly with Java

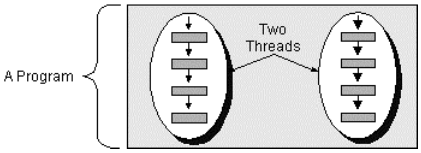
Concurrent Programming
Java Threads

- We will follow the Java Tutorial
  - http://java.sun.com/docs/books/tutorial/essential/threads/
- ***Threads*** are the units of execution context in Java

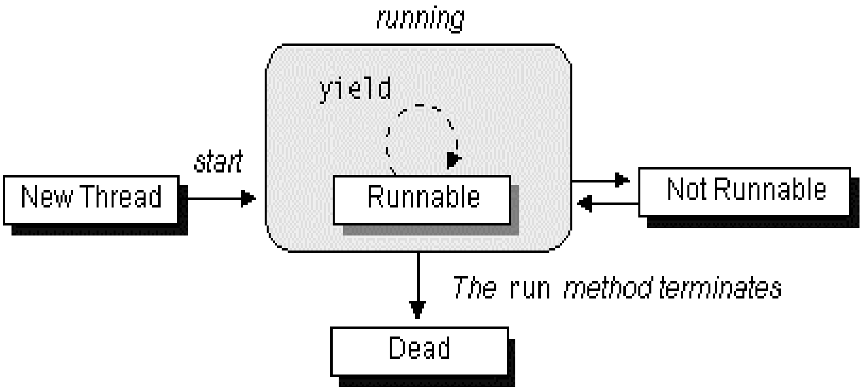


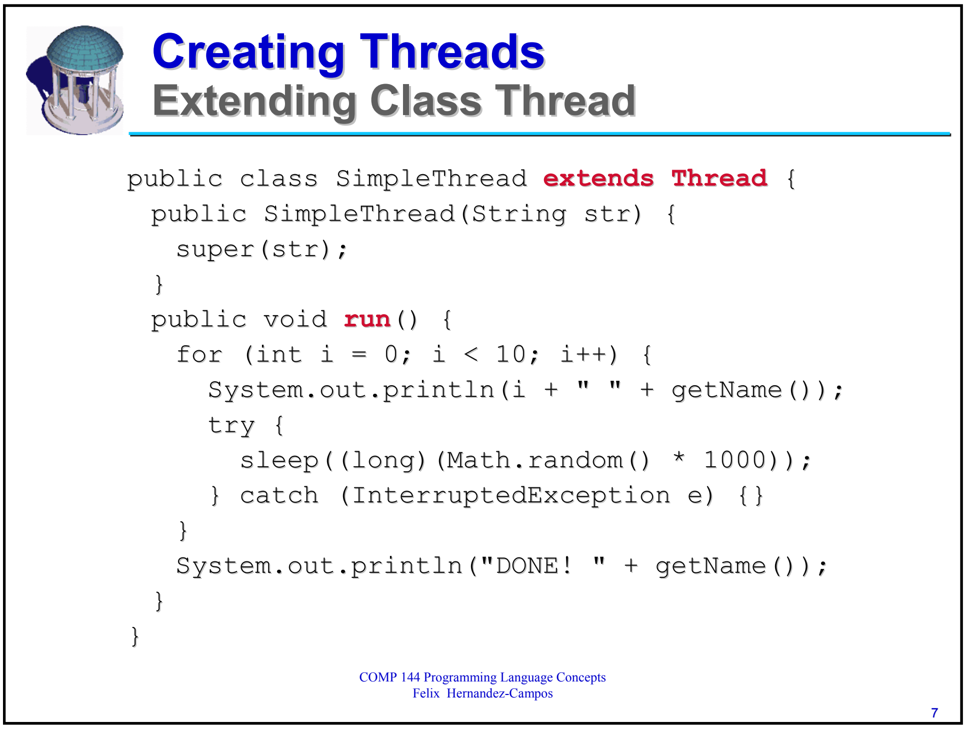
Sequential Program

Concurrent Program



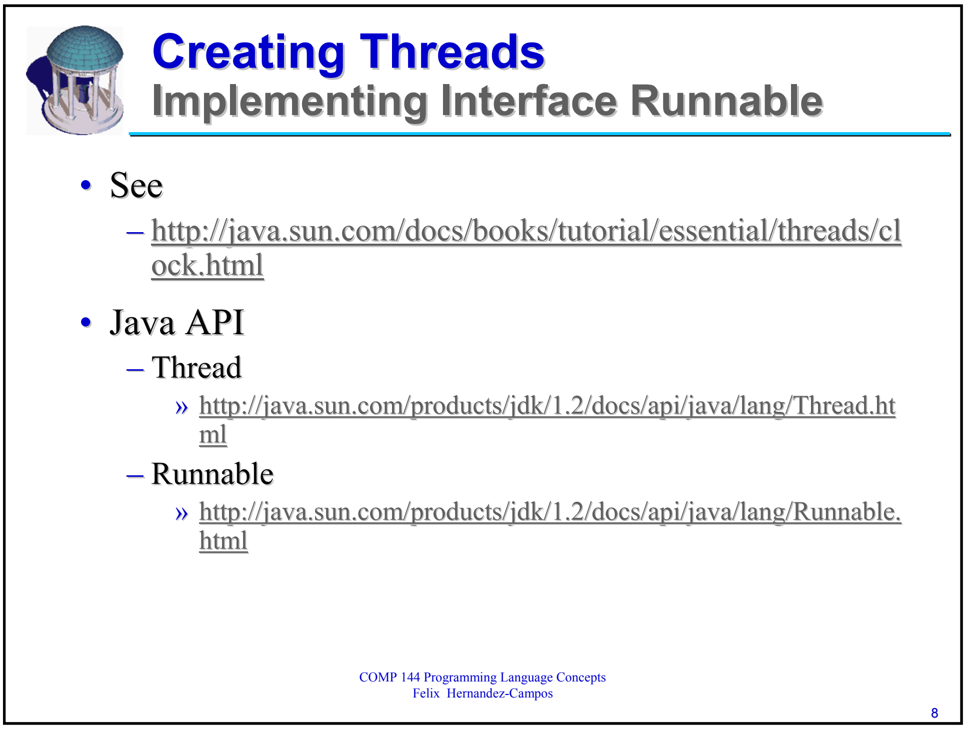
Thread Lifetime





Creating Threads
Extending Class Thread

```
public class SimpleThread extends Thread {
  public SimpleThread(String str) {
    super(str);
  }
  public void run() {
    for (int i = 0; i < 10; i++) {
      System.out.println(i + " " + getName());
      try {
        sleep((long)(Math.random() * 1000));
      } catch (InterruptedException e) {}
    }
    System.out.println("DONE! " + getName());
  }
}
```

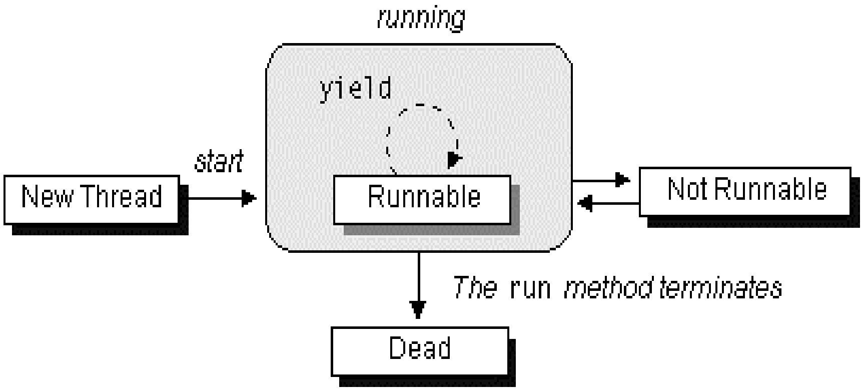


Creating Threads
Implementing Interface Runnable

- See
  - http://java.sun.com/docs/books/tutorial/essential/threads/clock.html
- Java API
  - Thread
    - http://java.sun.com/products/jdk/1.2/docs/api/java/lang/Thread.html
  - Runnable
    - http://java.sun.com/products/jdk/1.2/docs/api/java/lang/Runnable.html

Thread Lifetime

- http://java.sun.com/docs/books/tutorial/essential/threads/lifecycle.html



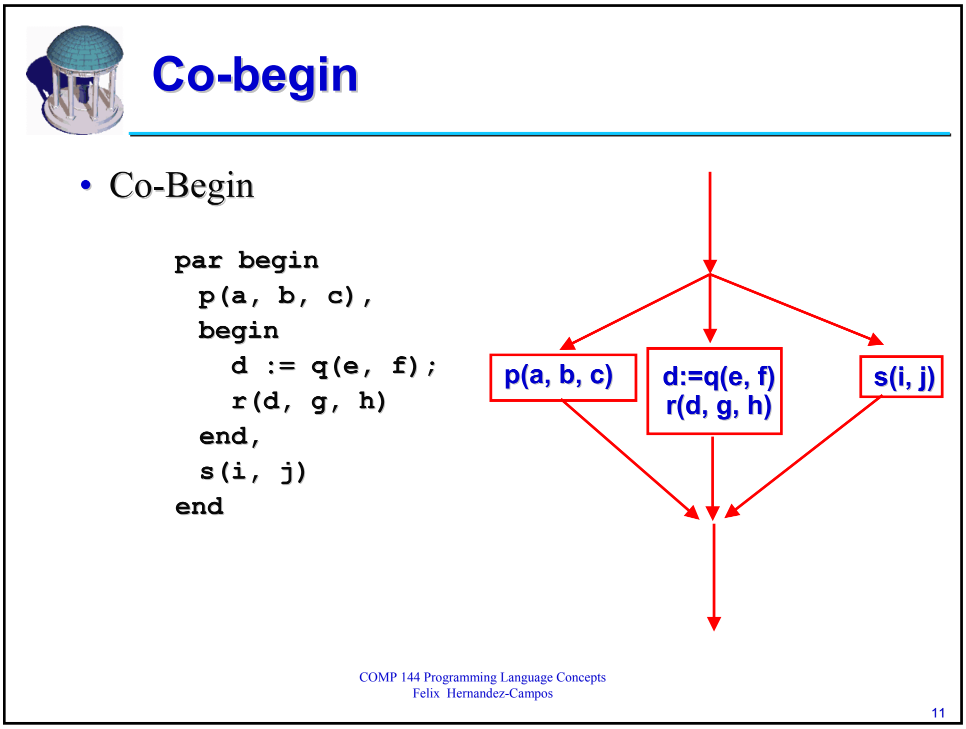
Co-begin

- Co-Begin (Algol-68, Occam, SR)
  - Sequential

```
begin
  a := 3,
  b := 4
end
```

  - Concurrent

```
par begin
  a := 3,
  b := 4
end
```



Co-begin

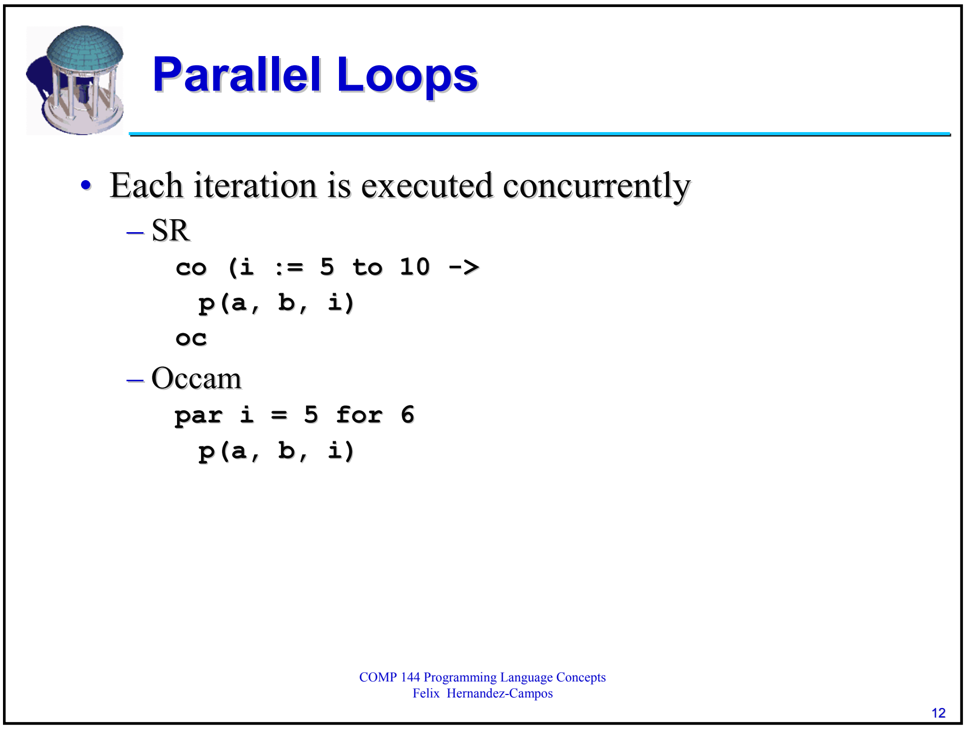
- Co-Begin

```
par begin
  p(a, b, c),
  begin
    d := q(e, f);
    r(d, g, h)
  end,
  s(i, j)
end
```

p(a, b, c)

d:=q(e, f)
r(d, g, h)

s(i, j)



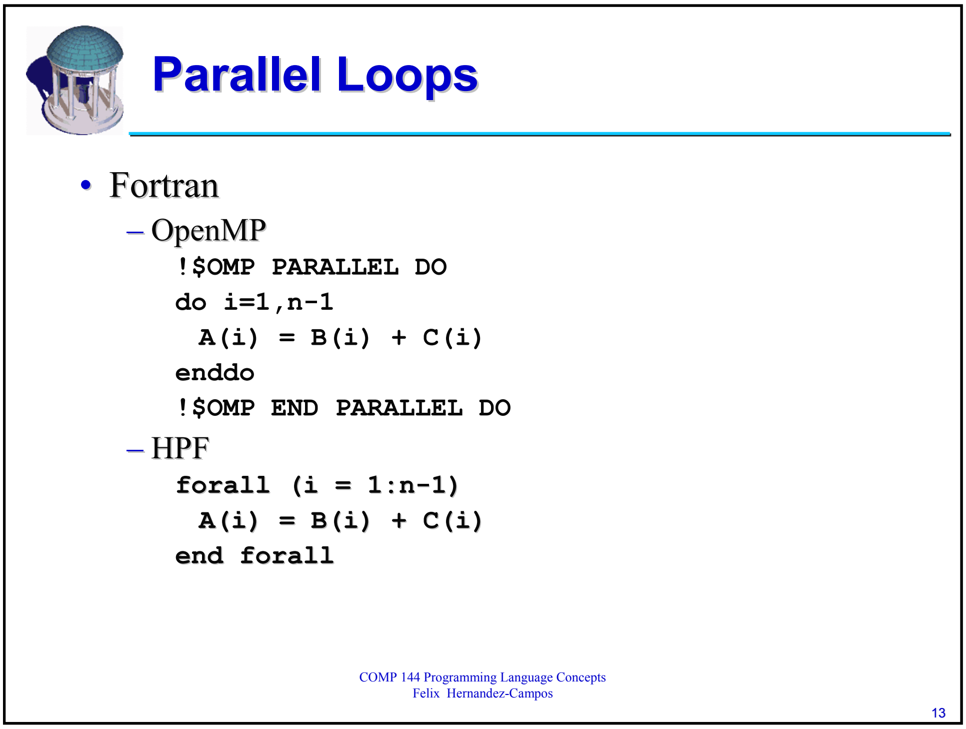
Parallel Loops

- Each iteration is executed concurrently
  - SR

```
co (i := 5 to 10 ->
  p(a, b, i)
oc
```

  - Occam

```
par i = 5 for 6
  p(a, b, i)
```



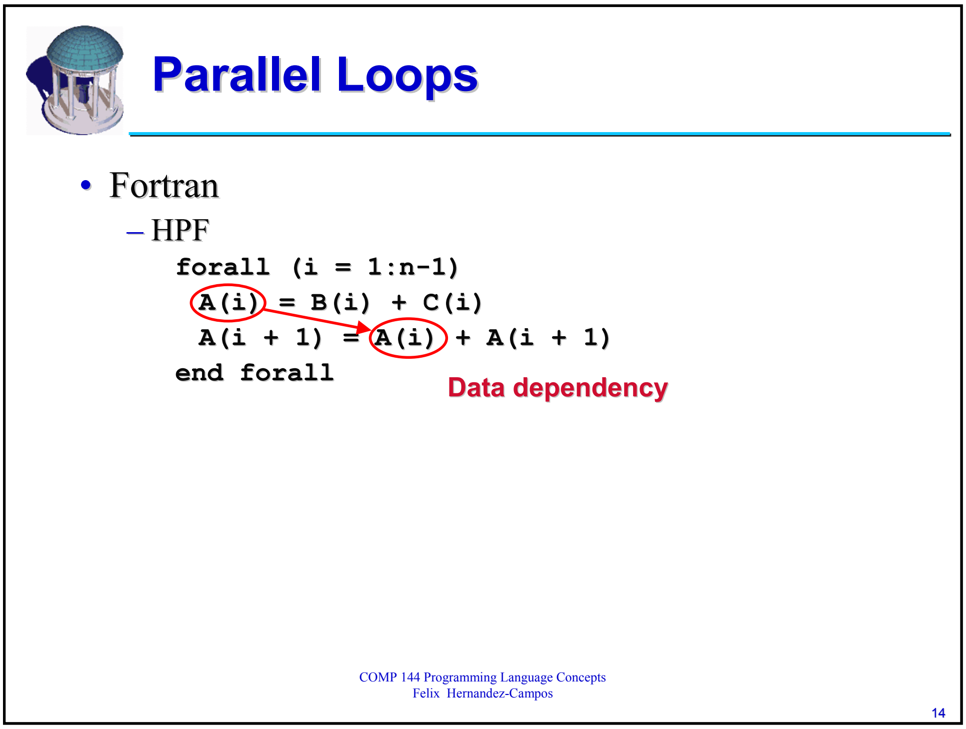
Parallel Loops

- Fortran
  - OpenMP

```
!$OMP PARALLEL DO
do i=1,n-1
  A(i) = B(i) + C(i)
enddo
!$OMP END PARALLEL DO
```

  - HPF

```
forall (i = 1:n-1)
  A(i) = B(i) + C(i)
end forall
```

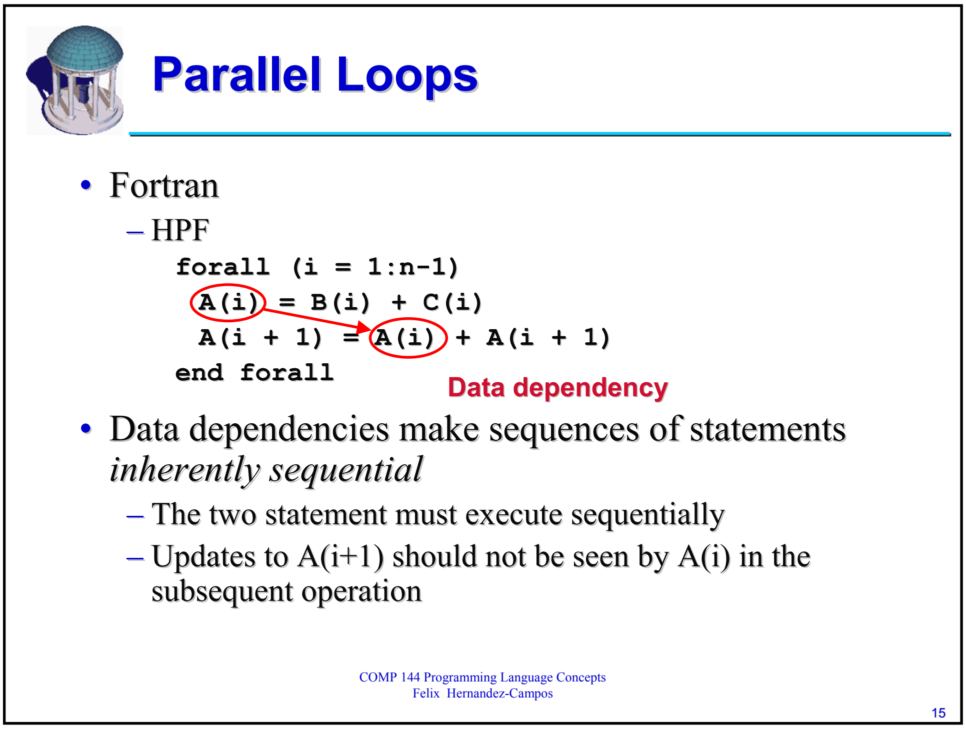


Parallel Loops

- Fortran
  - HPF

```
forall (i = 1:n-1)
  A(i) = B(i) + C(i)
  A(i + 1) = A(i) + A(i + 1)
end forall
```

Data dependency



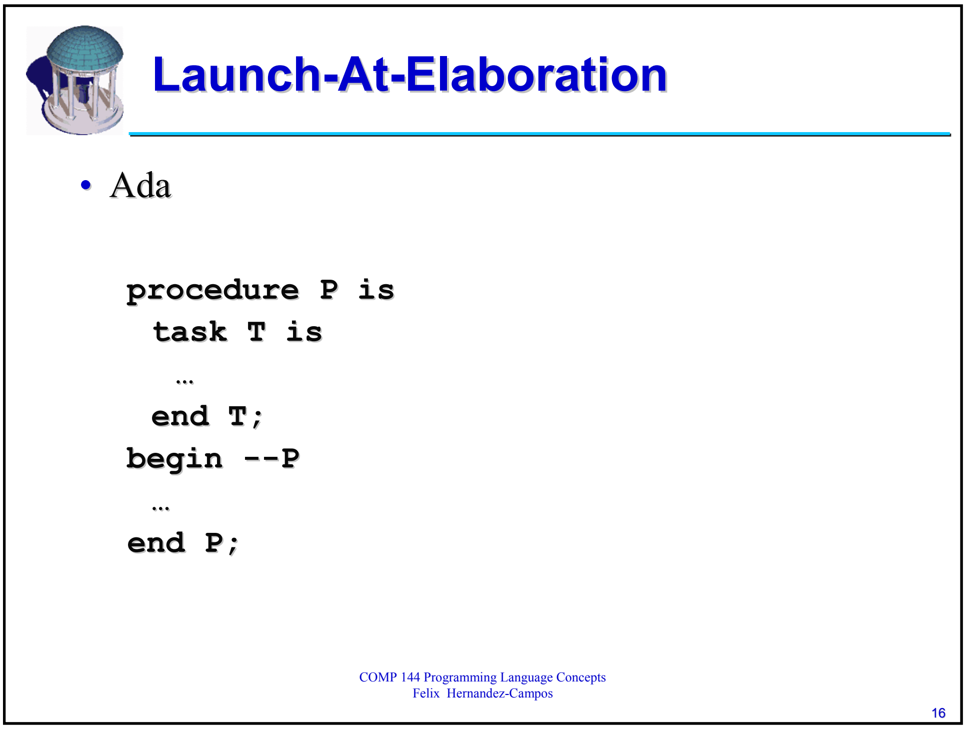
Parallel Loops

- Fortran
  - HPF

```
forall (i = 1:n-1)
 A(i) = B(i) + C(i)
 A(i + 1) = A(i) + A(i + 1)
end forall
```

Data dependency

- Data dependencies make sequences of statements *inherently sequential*
  - The two statement must execute sequentially
  - Updates to A(i+1) should not be seen by A(i) in the subsequent operation



Launch-At-Elaboration

- Ada

```
procedure P is
  task T is
    …
  end T;
begin --P
  …
end P;
```

Fork/Join

- In previous mechanisms, the creation of threads is implicit, while it is explicit in the fork-join models
  - The fork join model is more general
  - *E.g.*, Ada

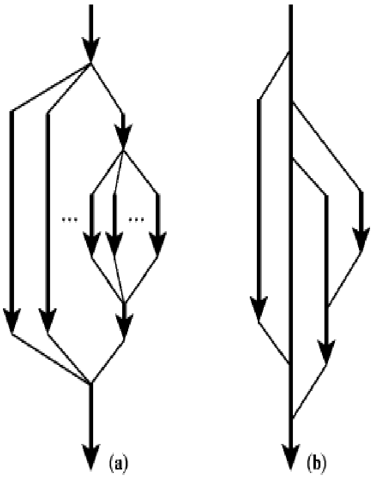
```
task type T is
  …
begin
  …
end T;
pt : access T := new T;
```

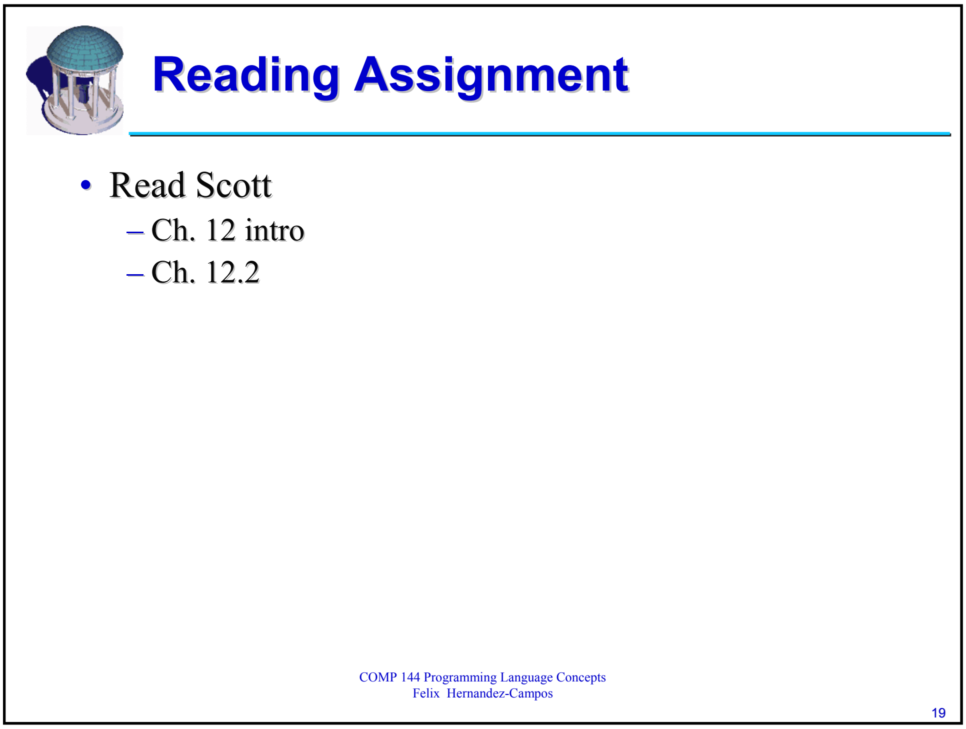
Forks **a new thread**

- Java threads follow the fork/join model

Fork/Join

- (a) Threads are always properly nested with co-begin statements
- (b) The fork/join mechanism is more general
  - In Java,
    - » Fork by creating a Thread and executing `start()`
    - » Join by executing `join()` on the thread





Reading Assignment

- Read Scott
  - Ch. 12 intro
  - Ch. 12.2