COMP 322: Fundamentals of Parallel Programming

Lecture 17: Phasers (contd), Signal Statement, Fuzzy Barriers

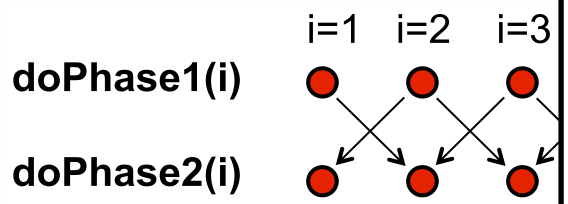
Vivek Sarkar
Department of Computer Science, Rice University
vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Worksheet #16: Left-Right Neighbor Synchronization using Phasers

Name: ____________________

Netid: ____________________



Complete the phased clause below to implement the left-right neighbor synchronization shown above.

```
1. finish (() -> {
2.   final HjPhaser[] ph =
         new HjPhaser[m+2]; // array of phaser objects
3.   forseq(0, m+1, (i) -> { ph[i] = newPhaser(SIG_WAIT) });
4.   forseq(1, m, (i) -> {
5.     asyncPhased(
          ph[i-1].inMode(ph[i-1].inMode(WAIT)),
          ph[i].inMode(ph[i].inMode(SIG)),
          ph[i+1].inMode(ph[i+1].inMode(WAIT)), () -> {
6.      doPhase1(i);
7.      next();
8.      doPhase2(i); }); // asyncPhased
9.   }); // forseq
10.}); // finish
```

Announcements

- Take-home midterm exam (Exam 1) will be given after lecture on Wednesday, February 26, 2014
  - Closed-book, closed computer, written exam that can be taken in any 2-hour duration during that period
  - Will need to be returned to Penny Anderson (Duncan Hall 3180) by 4pm on Friday, February 28, 2014
    - Exam can also be picked up from Penny Anderson starting 2pm on Feb 26th if you're unable to attend lecture.
  - No lecture on Friday, Feb 28th
- Homework 3 is due by by 11:59pm on Wednesday, March 12, 2014
  - Programming assignment is more challenging than in previosu homeworks --- start early!

Scope of Midterm Exam

- **Midterm exam will cover material from Lectures 1 - 17**
  - Lecture 18 (Feb 26th) will be a Midterm review
- **Excerpts from midterm exam instructions**
  - "closed-book, closed-notes, closed-computer"
  - "Record start time when you open the exam, and end time when you finish. The total duration must be at most 2 hours. "
  - "Since this is a written exam and not a programming assignment, syntactic errors in program text will not be penalized (e.g., missing semicolons, incorrect spelling of keywords, etc) so long as the meaning of your solution is unambiguous."
  - "If you believe there is any ambiguity or inconsistency in a question, you should state the ambiguity or inconsistency that you see, as well as any assumptions that you make to resolve it."

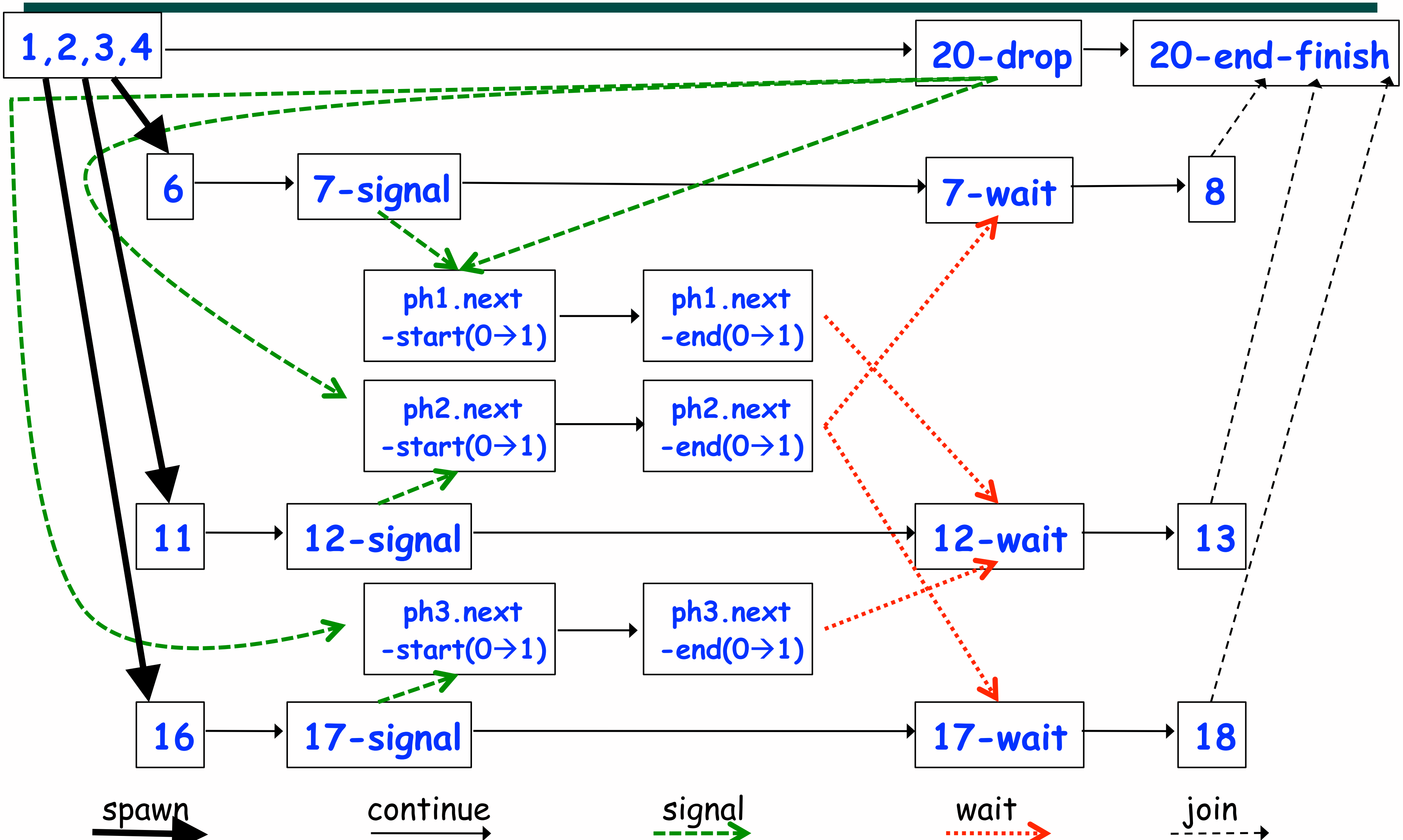
Summary of Phaser Construct (Recap)

- Phaser allocation
  - HjPhaser ph = newPhaser(mode);
    - Phaser ph is allocated with registration mode
    - Phaser lifetime is limited to scope of Immediately Enclosing Finish (IEF)
- Registration Modes
  - HjPhaserMode.SIG, HjPhaserMode.WAIT, HjPhaserMode.SIG_WAIT, HjPhaserMode.SIG_WAIT_SINGLE
    - NOTE: phaser WAIT is unrelated to Java wait/notify (which we will study later)
- Phaser registration
  - asyncPhased (ph_1.inMode(<$mode_1$>), ph_2.inMode(<$mode_2$>), … () -> <stmt>)
    - Spawned task is registered with ph_1 in $mode_1$, ph_2 in $mode_2$, …
    - Child task's capabilities must be subset of parent's
    - asyncPhased <stmt> propagates all of parent's phaser registrations to child
- Synchronization
  - next();
    - Advance each phaser that current task is registered on to its next phase
      All signals are performed, followed by all waits
    - Semantics depends on registration mode
    - Barrier is a special case of phaser, which is why *next* is used for both

Left-Right Neighbor Synchronization Example for m=3

```
finish(() -> { // Task-0
    final HjPhaser ph1 = newPhaser(SIG_WAIT);
    final HjPhaser ph2 = newPhaser(SIG_WAIT);
    final HjPhaser ph3 = newPhaser(SIG_WAIT);
    asyncPhased(ph1.inMode(SIG),ph2.inMode(WAIT),
      () -> { doPhase1(1);
        next(); // signals ph1, waits on ph2
        doPhase2(1);
    }); // Task T1
    asyncPhased(ph2.inMode(SIG),ph1.inMode(WAIT),ph3.inMode(WAIT),
      () -> { doPhase1(2);
         next(); // signals ph2, waits on ph3
         doPhase2(2);
     }); // Task T2
     asyncPhased(ph3.inMode(SIG),ph2.inMode(WAIT),
       () -> { doPhase1(3);
         next(); // signals ph3, waits on ph2
         doPhase2(3);
     }); // Task T3
}); // finish
```

Computation Graph for m=3 example



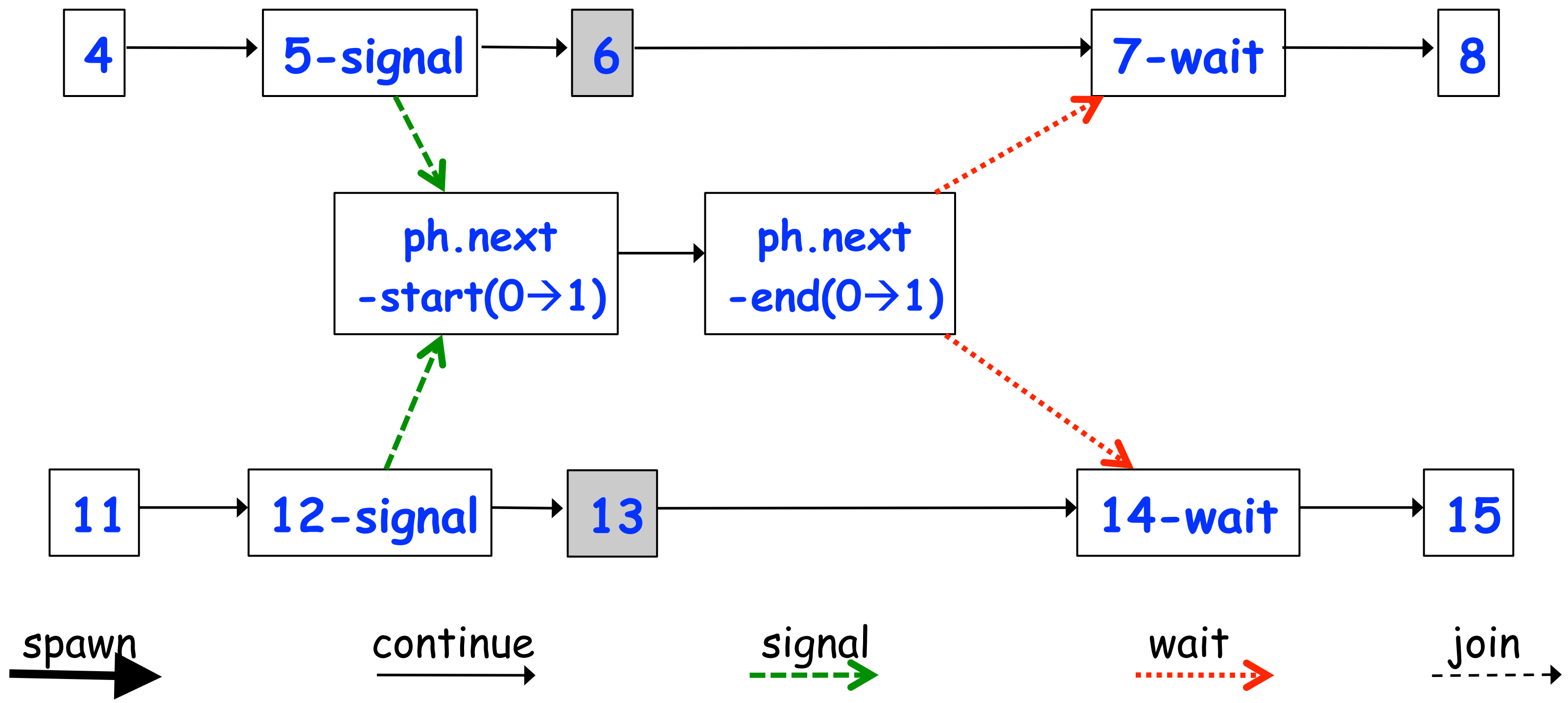
Signal statement

- When a task T performs a signal operation, it notifies all the phasers it is registered on that it has completed all the work expected by other tasks in the current phase ("shared" work).
  - Since signal is a non-blocking operation, an early execution of signal cannot create a deadlock.
- Later, when T performs a next operation, the next degenerates to a wait since a signal has already been performed in the current phase.
- The execution of "local work" between signal and next is performed during phase transition
  - Referred to as a "split-phase barrier" or "fuzzy barrier"

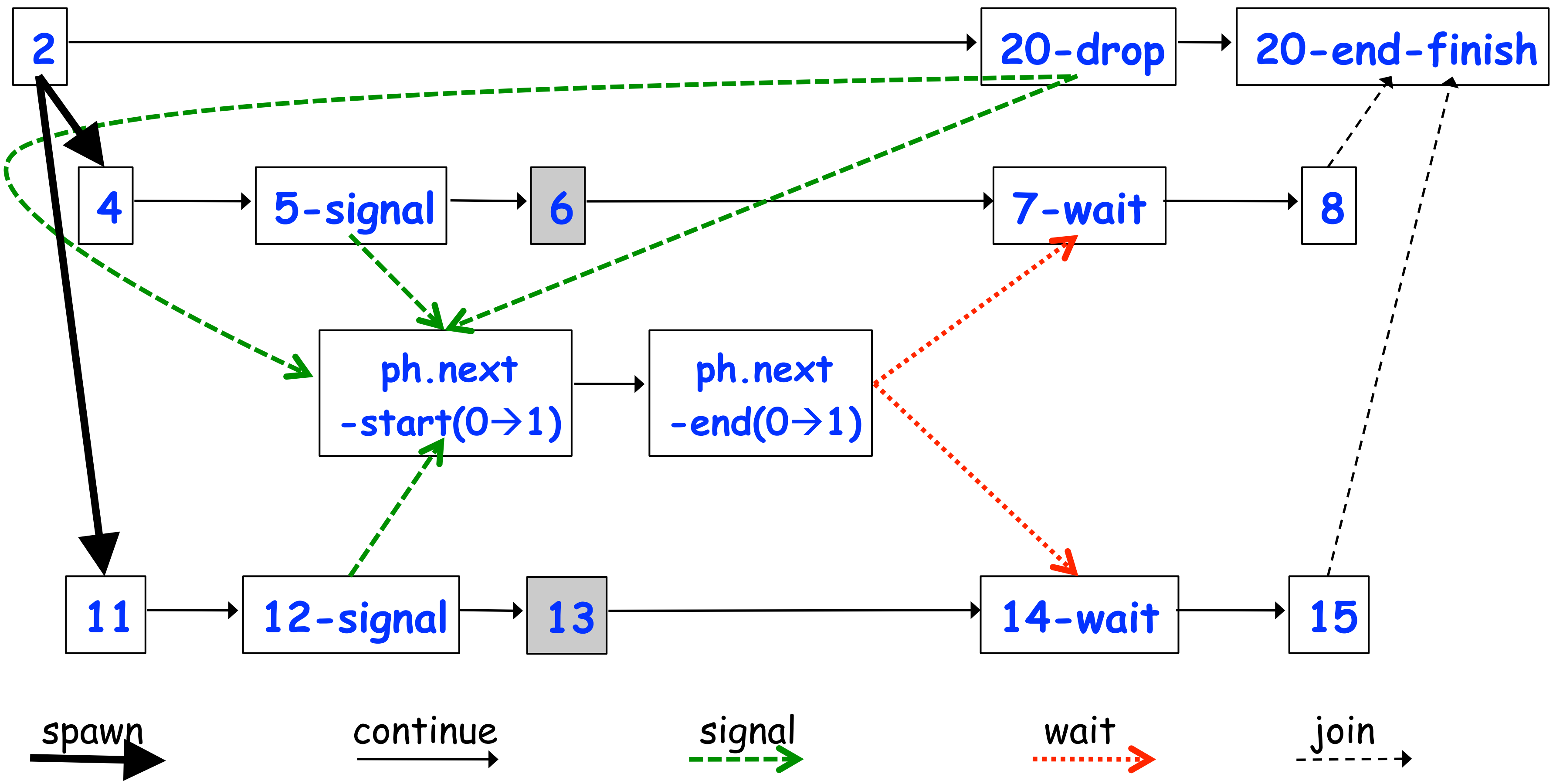
Example of Split-Phase Barrier using Signal Statement

```
finish(() -> {
  final HjPhaser ph = newPhaser(SIG_WAIT);
  asyncPhased(ph.inMode(SIG_WAIT), () -> { // Task T1
    a = ... ;   // Shared work in phase 0
    signal();     // Signal completion of a's computation
    b = ... ;   // Local work in phase 0
    next();       // Barrier -- wait for T2 to compute x
    b = f(b,x); // Use x computed by T2 in phase 0
  });
  asyncPhased(ph.inMode(SIG_WAIT), () -> { // Task T2
    x = ... ;   // Shared work in phase 0
    signal();     // Signal completion of x's computation
    y = ... ;   // Local work in phase 0
    next();      // Barrier -- wait for T1 to compute a
    y = f(y,a); // Use a computed by T1 in phase 0
  });
}); // finish
```

Computation Graph for Split-Phase Barrier Example (without async and finish nodes and edges)



Full Computation Graph for Split-Phase Barrier Example (Figure 52)



Data Races and Determinism extended to Phasers

- **A parallel program is said to be *functionally deterministic* if it always computes the same answer when given the same input**
- **A parallel program is said to be *structurally deterministic* if it always produces the same computation graph when given the same input**
- **Race-Free Determinism**
  - **If a parallel program is written using the constructs learned so far in Module 1 (finish, async, futures, accumulators, data-driven tasks, barriers, phasers) and is known to be race-free, *then it must be both functionally deterministic and structurally deterministic***

Worksheet #17: Critical Path Length for Computation with Signal Statement

Name: ______________________ **Netid:** ______________________

Compute the WORK and CPL values for the program shown below. How would they be different if the signal() statement was removed? (Hint: draw a computation graph as in slide 10)

```
finish(() -> {
  final HjPhaser ph = newPhaser(SIG_WAIT);
  asyncPhased(ph.inMode(SIG_WAIT), () -> { // Task T1
    A(0); doWork(1);   // Shared work in phase 0
    signal();
    B(0); doWork(100); // Local work in phase 0
    next(); // Wait for T2 to complete shared work in phase 0
    C(0); doWork(1);
  });
  asyncPhased(ph.inMode(SIG_WAIT), () -> { // Task T2
    A(1); doWork(1);   // Shared work in phase 0
    next(); // Wait for T1 to complete shared work in phase 0
    C(1); doWork(1);
    D(1); doWork(100); // Local work in phase 0
  });
}); // finish
```