COMP 322: Fundamentals of Parallel Programming

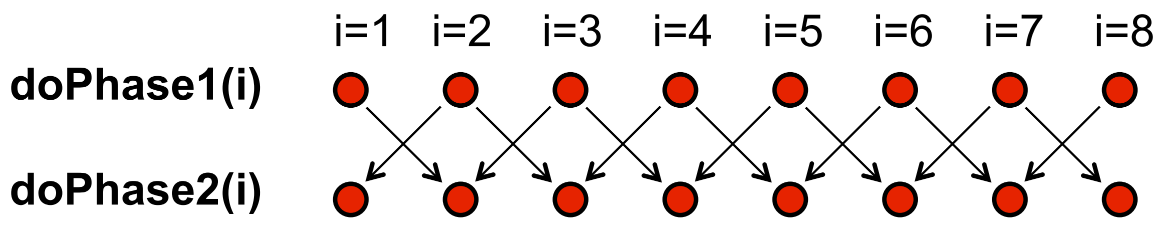
Lecture 17: Pipeline Parallelism, Signal Statement, Fuzzy Barriers

Vivek Sarkar, Eric Allen
Department of Computer Science, Rice University

Contact email: vsarkar@rice.edu

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

Solution to Worksheet #16: Left-Right Neighbor Synchronization using Phasers



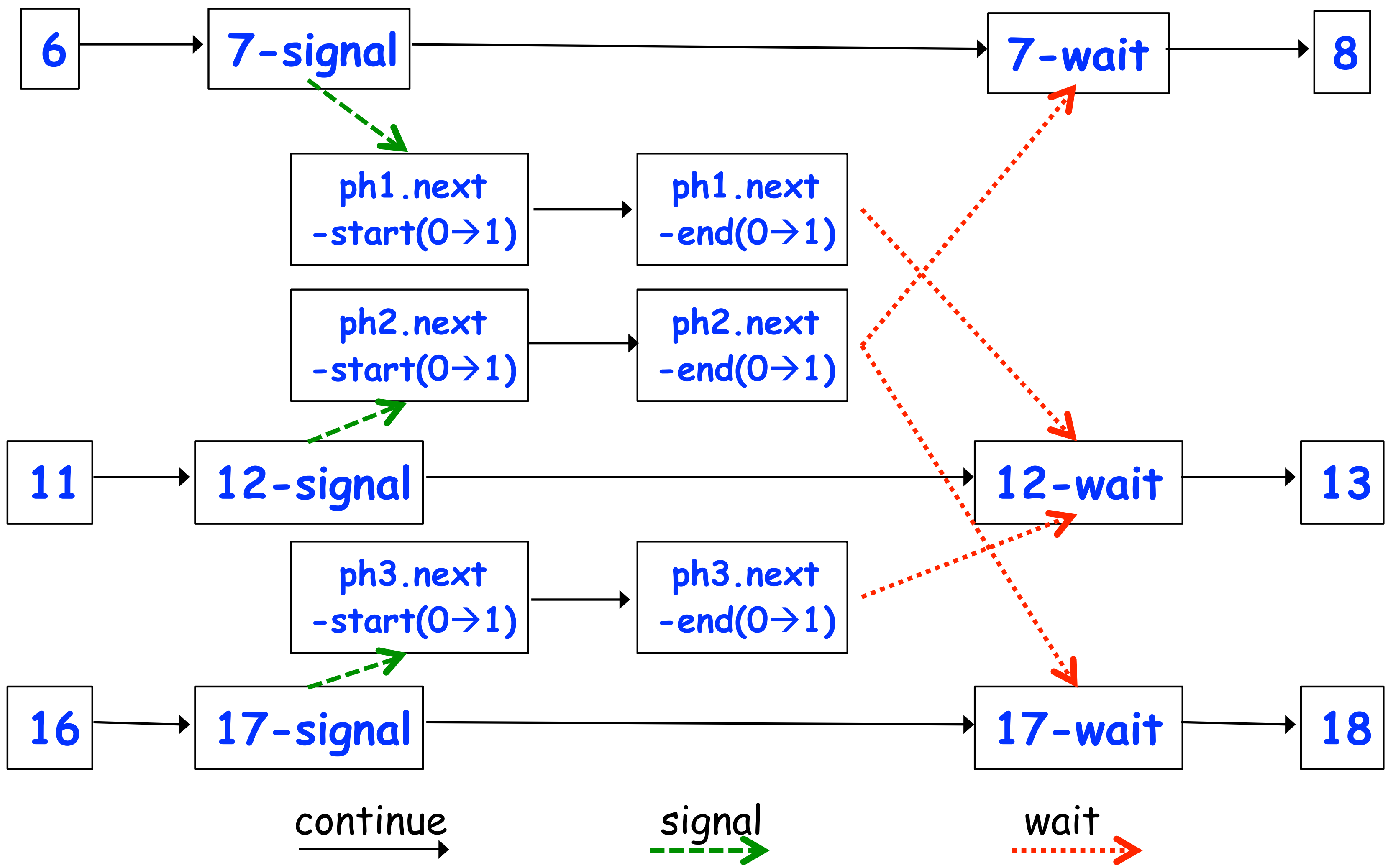
Complete the phased clause below to implement the left-right neighbor synchronization shown above.

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

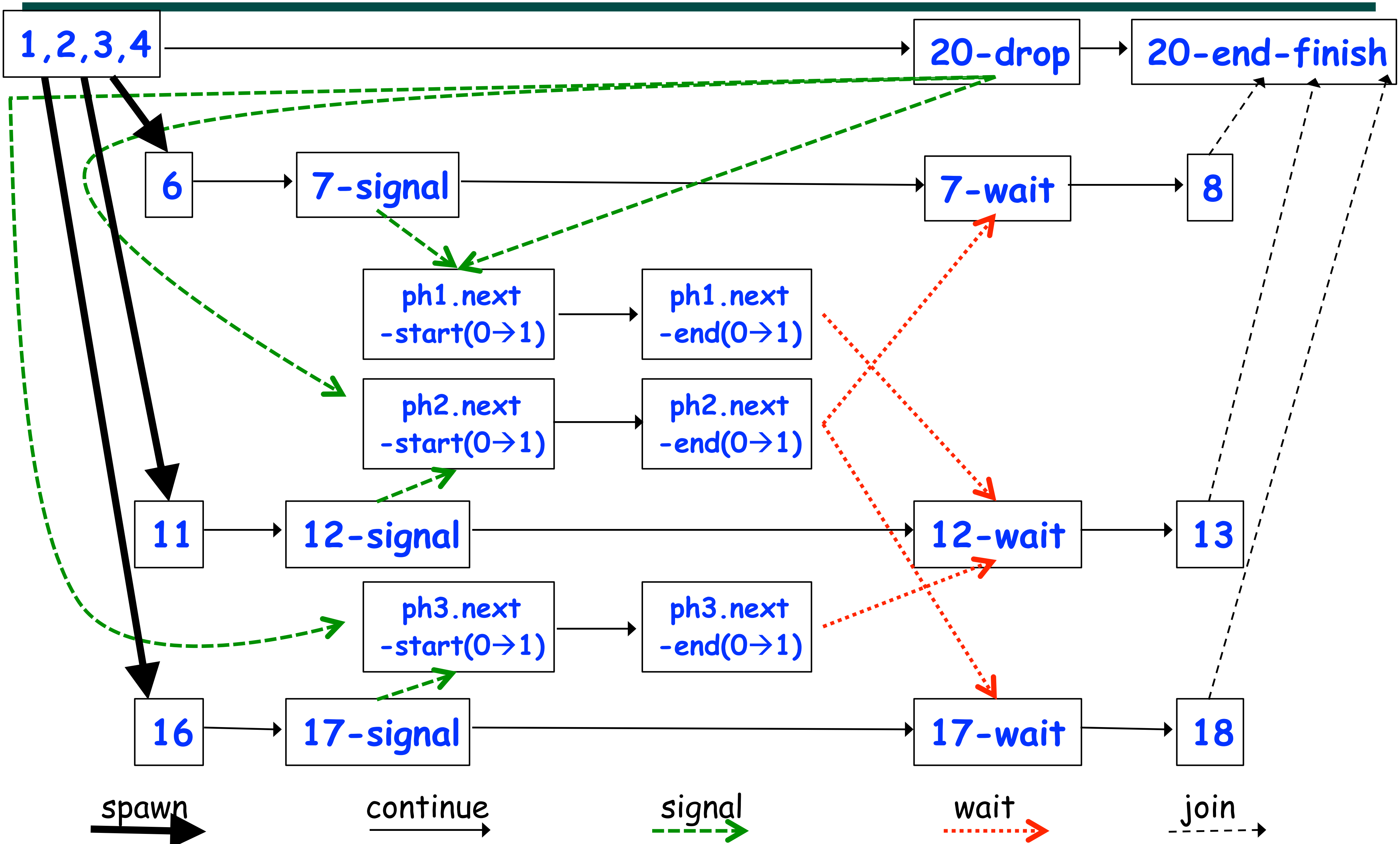
Left-Right Neighbor Synchronization (restricted to m=3)

```
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
(without async-finish nodes and edges)

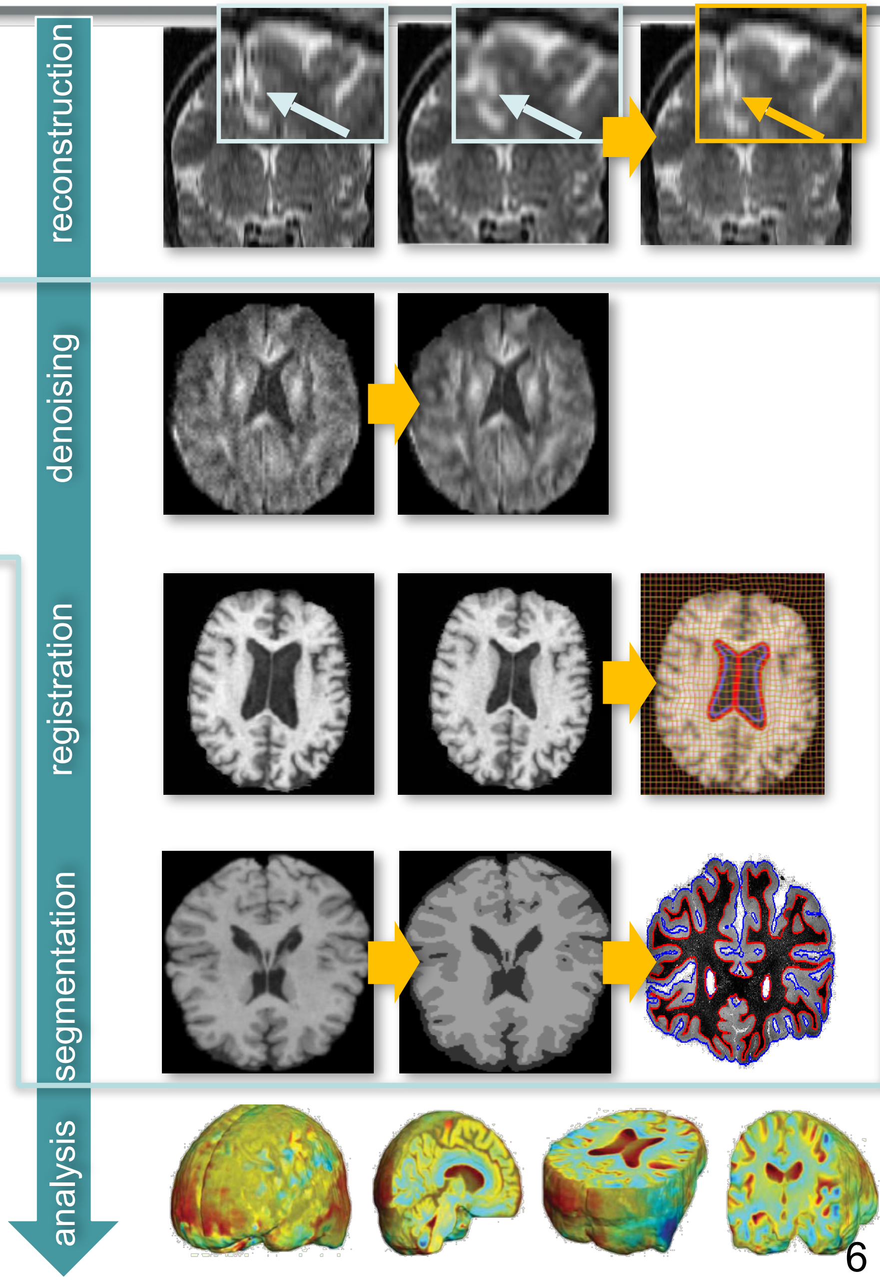


Computation Graph for m=3 example
(with async-finish nodes and edges)



Medical imaging pipeline

- New reconstruction methods
  - decrease radiation exposure (CT)
  - number of samples (MR)
- 3D/4D image analysis pipeline
  - Denoising
  - Registration
  - Segmentation
- Analysis
  - Real-time quantitative cancer assessment applications
- Potential:
  - order-of-magnitude performance improvement
  - power efficiency improvements
  - real-time clinical applications and simulations using patient imaging data

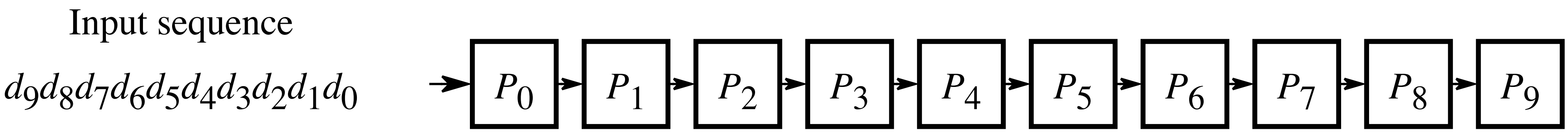


Slide credit: NSF Expeditions Center for Domain-Specific Computing (UCLA, Rice, OSU, UCSB)

Pipeline Parallelism: Another Example of Point-to-point Synchronization

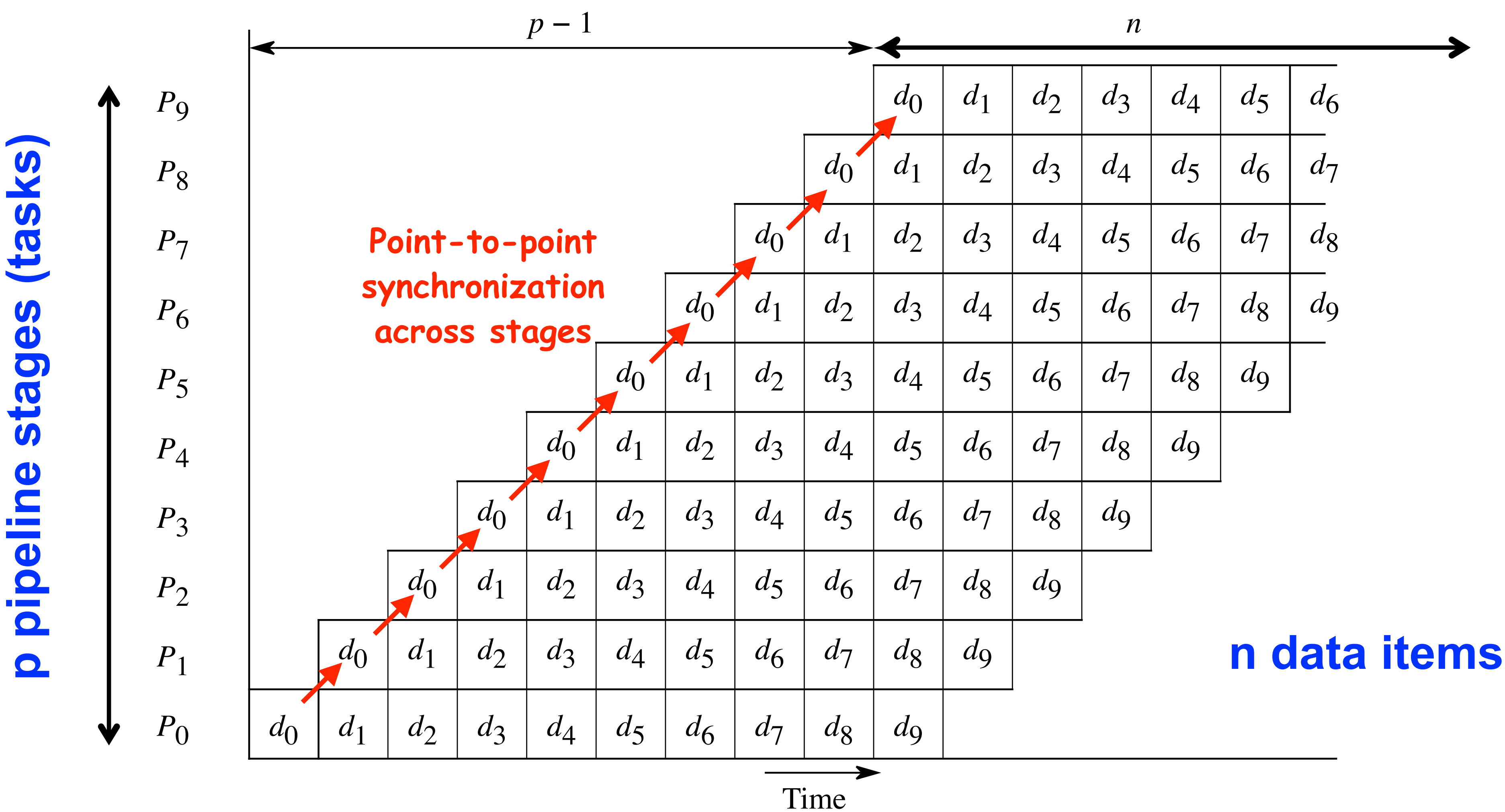
- Medical imaging pipeline with three stages
  1. Denoising stage generates a sequence of results, one per image.
  2. Registration stage's input is Denoising stage's output.
  3. Segmentation stage's input is Registration stage's output.
- Even though the processing is sequential for a single image, *pipeline parallelism* can be exploited via point-to-point synchronization between neighboring stages

General structure of a One-Dimensional Pipeline



- **Assuming that the inputs d_0, d_1, . . . arrive sequentially, pipeline parallelism can be exploited by enabling task (stage) P_i to work on item d_{k-i} when task (stage) P_0 is working on item d_k.**

Timing Diagram for One-Dimensional Pipeline



- Horizontal axis shows progress of time from left to right, and vertical axis shows which data item is being processed by which pipeline stage at a given time.

Complexity Analysis of One-Dimensional Pipeline

- Assume
  - n = number of items in input sequence
  - p = number of pipeline stages
  - each stage takes 1 unit of time to process a single data item
- WORK = $n \times p$ is the total work for all data items
- CPL = $n + p - 1$ is the critical path length of the pipeline
- Ideal parallelism, PAR = WORK/CPL = $np/(n + p - 1)$
- Boundary cases
  - $p = 1 \rightarrow$ PAR $= n/(n + 1 - 1) = 1$
  - $n = 1 \rightarrow$ PAR $= p/(1 + p - 1) = 1$
  - $n = p \rightarrow$ PAR $= p/(2 - 1/p) \approx p/2$
  - $n \gg p \rightarrow$ PAR $\approx p$

Producer-Consumer pattern with phasers (used for implementing pipeline parallelism)

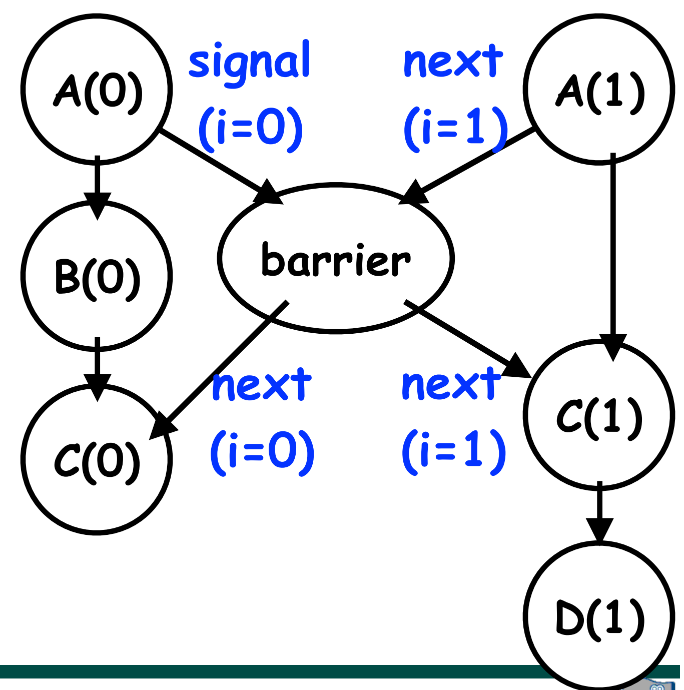
```
asyncPhased(ph.inMode(SIG), () -> {
    for (int i = 0; i < rounds; i++) {
        buffer.insert(…);
        // producer can go ahead as they are in SIG mode
        next();
    }
});

asyncPhased(ph.inMode(WAIT), () -> {
    for (int i = 0; i < rounds; i++) {
        next();
        buffer.remove(…);
    }
});
```

Signal statement & Fuzzy barriers

- 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 ("shared" work) in the current phase.
- 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 overlapped with the phase transition (referred to as a "split-phase barrier" or "fuzzy barrier")

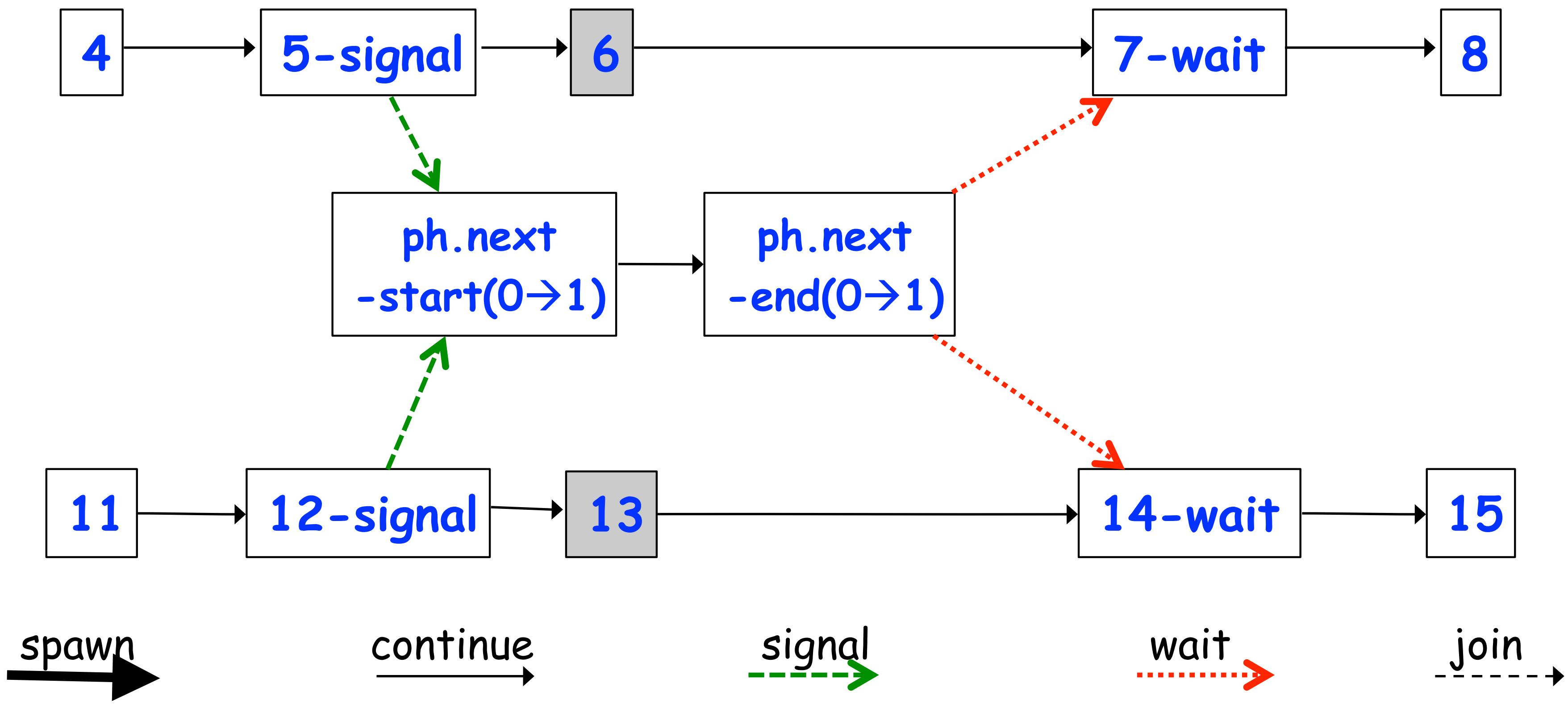
```
forall (point[i] : [0:1]) {
  A(i); // Phase 0
  if (i==0) { signal; B(i); }
  next; // Barrier
  C(i); // Phase 1
  if (i==1) { D(i); }
}
```



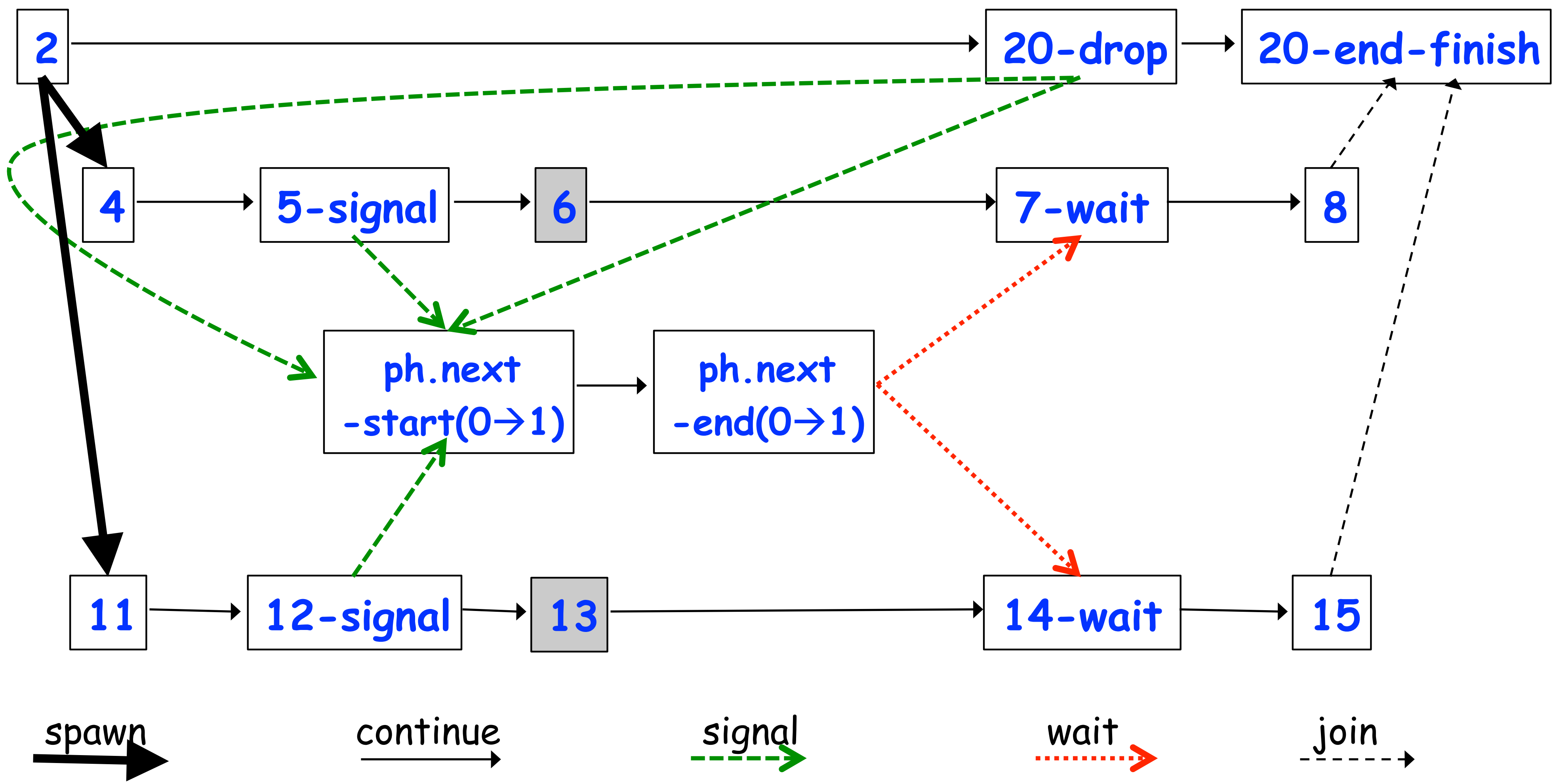
Another Example of a Split-Phase Barrier using the Signal Statement

```
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-finish nodes and edges)



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



Announcements

- Take-home midterm exam (Exam 1) will be given after lecture on Wednesday, February 25, 2015
  - Closed-book, closed computer, written exam that can be taken in any 2-hour duration during that period
  - Will need to be returned to Bel Martinez (Duncan Hall 3122) by 4pm on Friday, February 27, 2015
    - Exam can also be picked up from Bel Martinez starting 2pm on Feb 25th if you're unable to attend lecture.
  - No lecture on Friday, Feb 27th
- Homework 3 is due by by 5:00pm on Friday, March 13, 2015
  - Programming assignment is more challenging than in previous homeworks --- start early!

Scope of Midterm Exam

- Midterm exam will cover material from Lectures 1 - 18
  - —Lecture 19 (Feb 25th) 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."