COMP 322: Fundamentals of Parallel Programming

Lecture 25: Concurrent Objects, Linearizability of Concurrent Objects

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

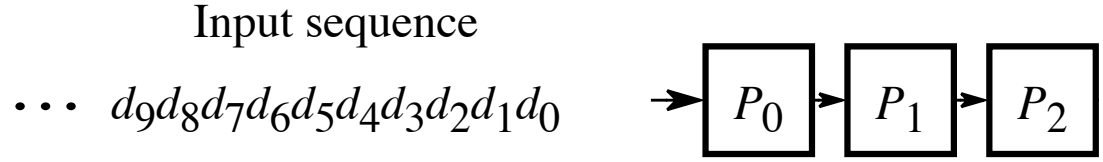
Contact email: vsarkar@rice.edu

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

Solution to Worksheet #24: Ideal Parallelism in Actor Pipeline

Consider a three-stage pipeline of actors set up so that P0.nextStage = P1, P1.nextStage = P2, and P2.nextStage = null. The process() method for each actor is shown below. Assume that 100 non-null messages are sent to actor P0 after all three actors are started, followed by a null message. What will the total WORK and CPL be for this execution? Recall that each actor has a sequential thread.

Solution: WORK = 300, CPL = 102

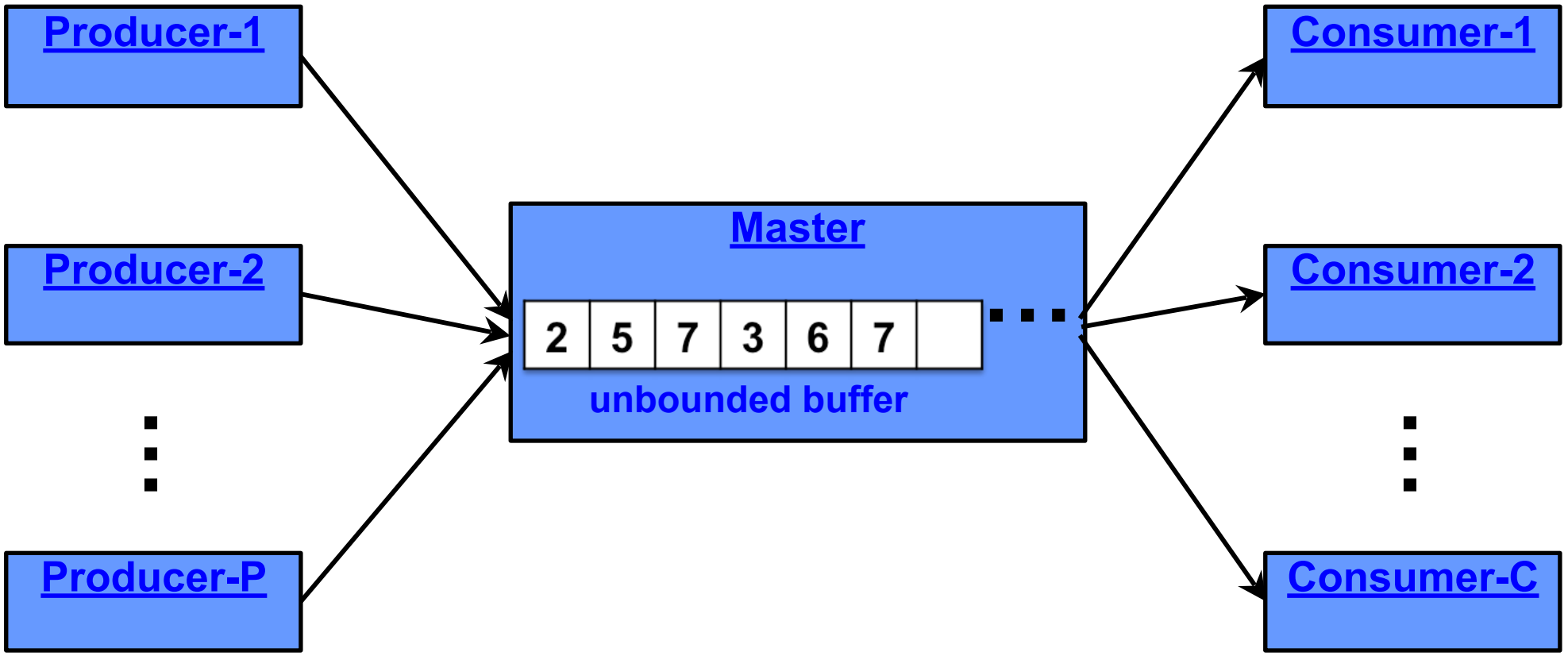



```
1.          protected void process(final Object msg) {
2.              if (msg == null) {
3.                  exit();
4.              } else {
5.                  doWork(1); // unit work
6.              }
7.              if (nextStage != null) {
8.                  nextStage.send(msg);
9.              }
10.         }
```

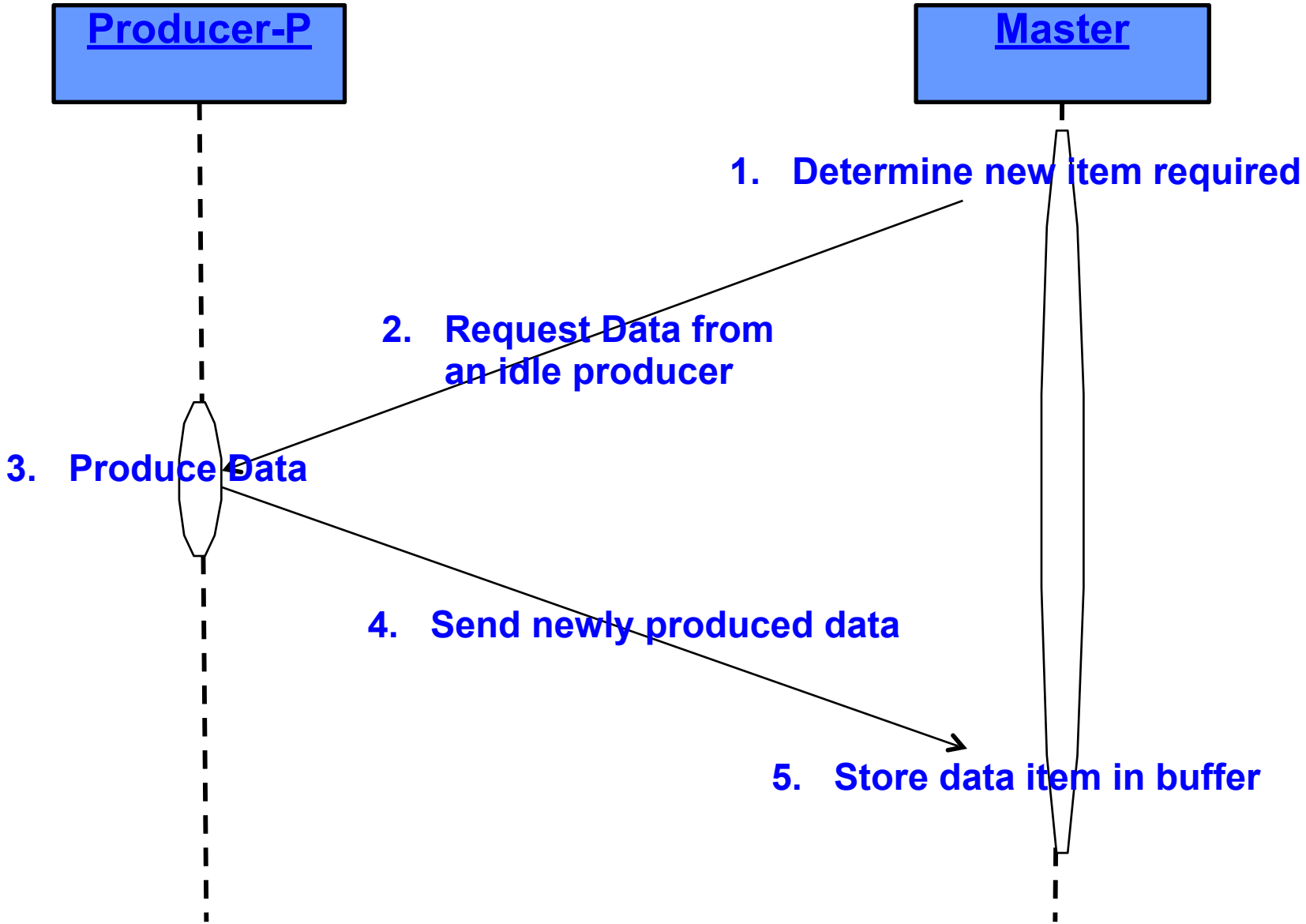

Concurrent Objects

- A concurrent object is an object that can correctly handle methods invoked in parallel by different tasks or threads
  - Also referred to as "thread-safe objects"
- For simplicity, it is usually assumed that the body of each method in a concurrent object is itself sequential
  - Assume that method does not create child async tasks
- Implementations of methods can be serialized (e.g., enclose each method in an actor or an object-based isolated statement) or can be concurrent (e.g., by using read-write modes in object-based isolation)
- A desirable goal is to develop implementations that are concurrent while being as close to the semantics of the serial version as possible
- Examples of concurrent objects: atomic variables, shared buffers, concurrent lists, concurrent hashmaps, …

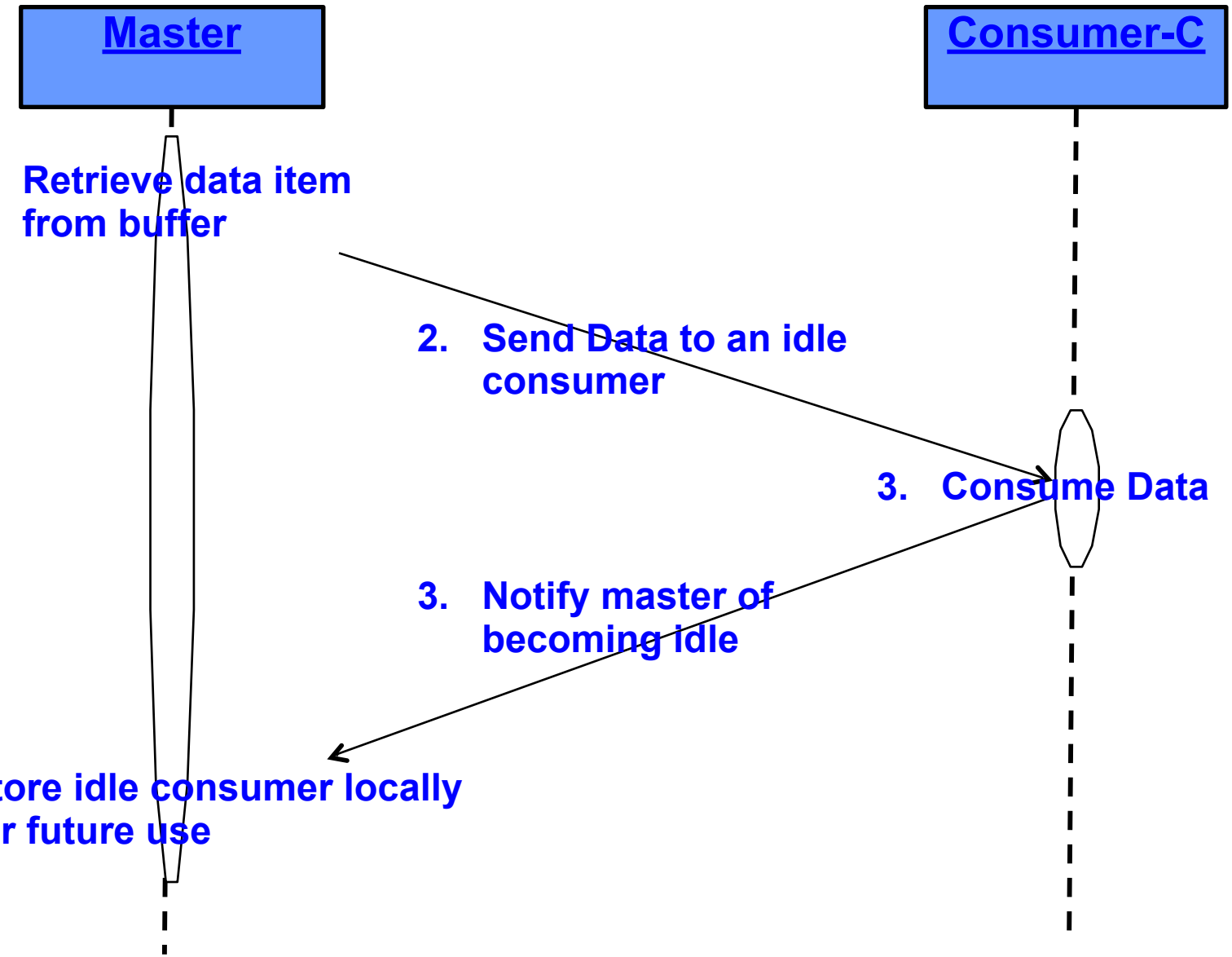
Example #1 of a Concurrent Object: Implementing an Unbounded Buffer using Actors



Unbounded Buffer Actor Interaction Diagram



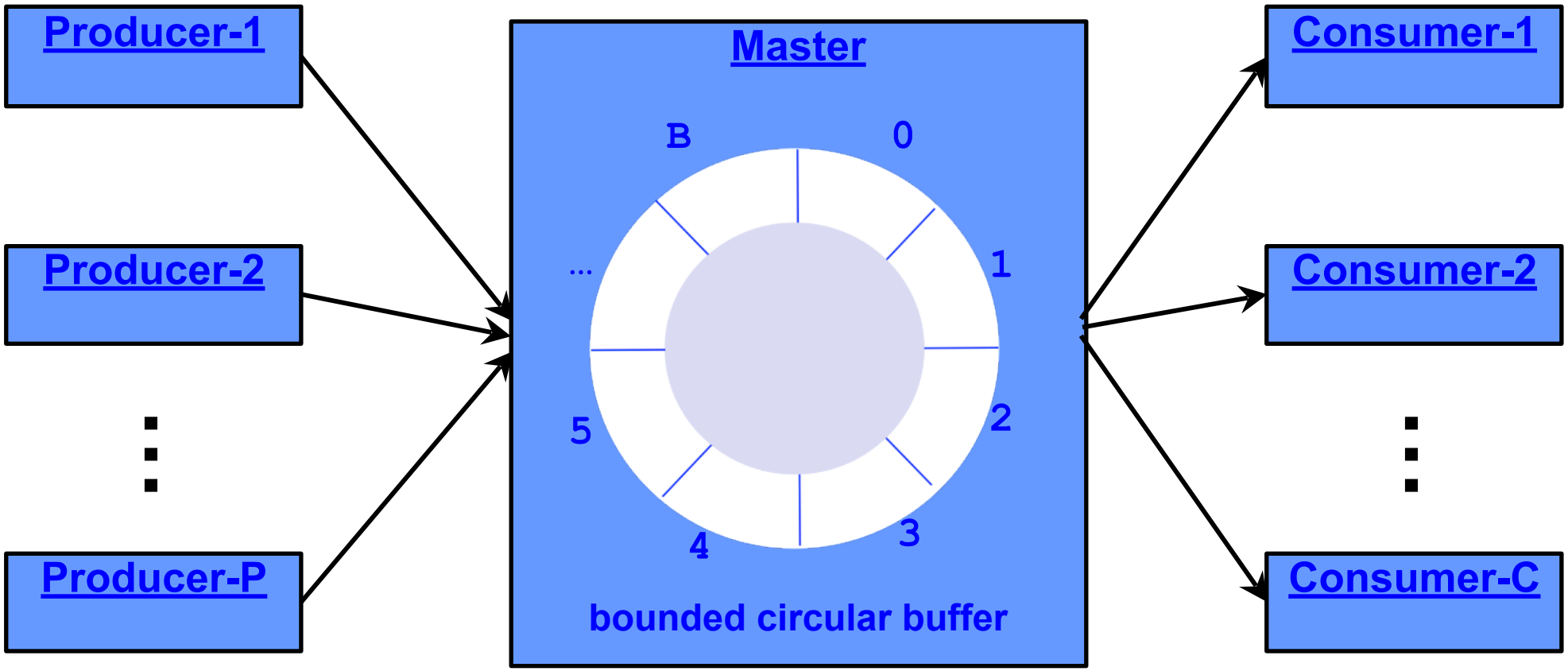
Unbounded Buffer Actor Interaction Diagram (contd)



Actor Responsibilities

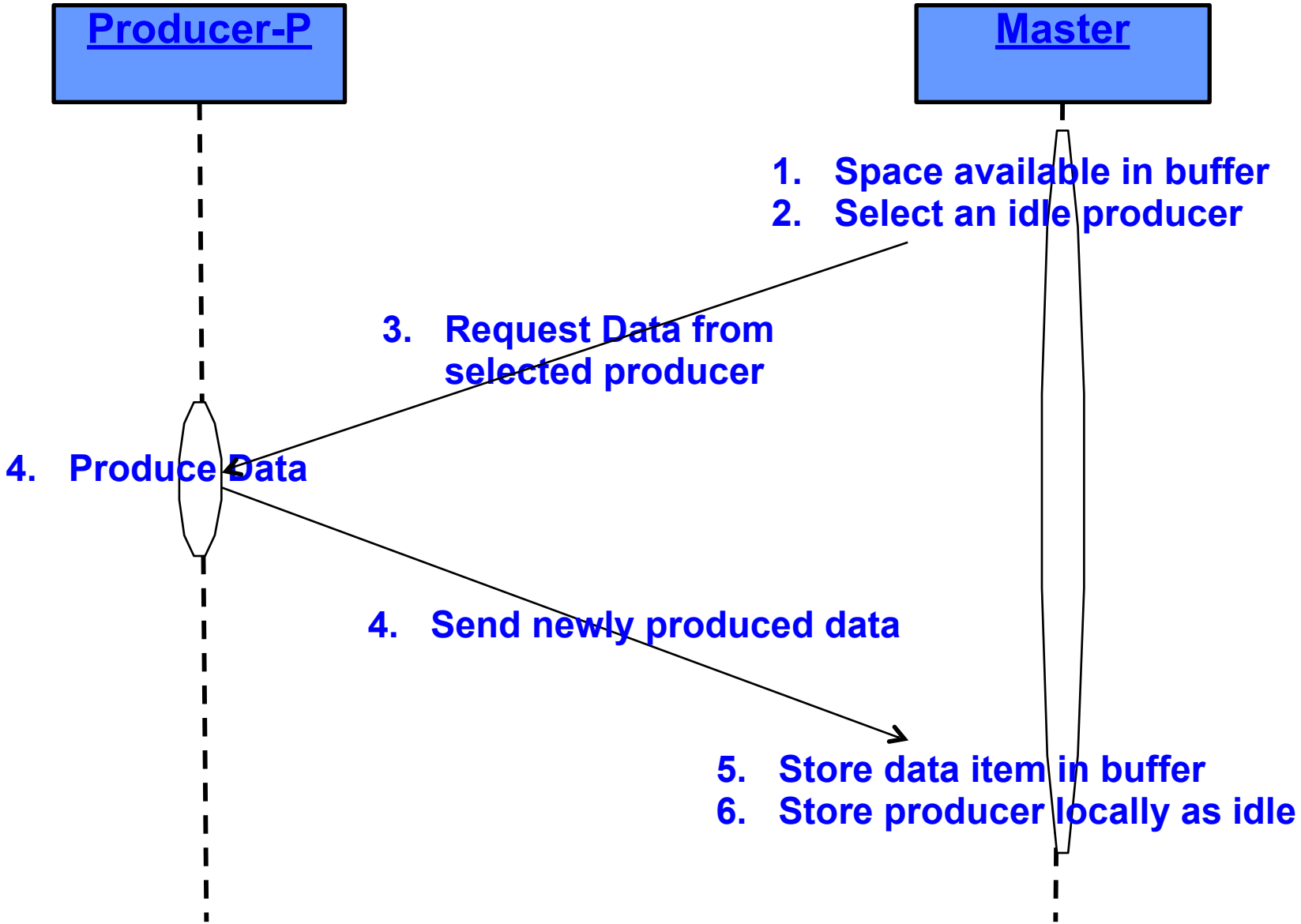
- **Master Actor**
  - —Receives Data Items from the producers
  - —Stores data items in its unbounded buffer
  - —Send data items to idle consumers
  - —Receives notifications when consumers are idle
- **Producer Actor**
  - —Receives requests to produce items
  - —Sends data items to the Master
- **Consumer Actor**
  - —Receives requests from Master to consume an item
  - —Sends notification to Master when it becomes idle

Example #2 of a Concurrent Object: Implementing an Bounded Buffer using Actors

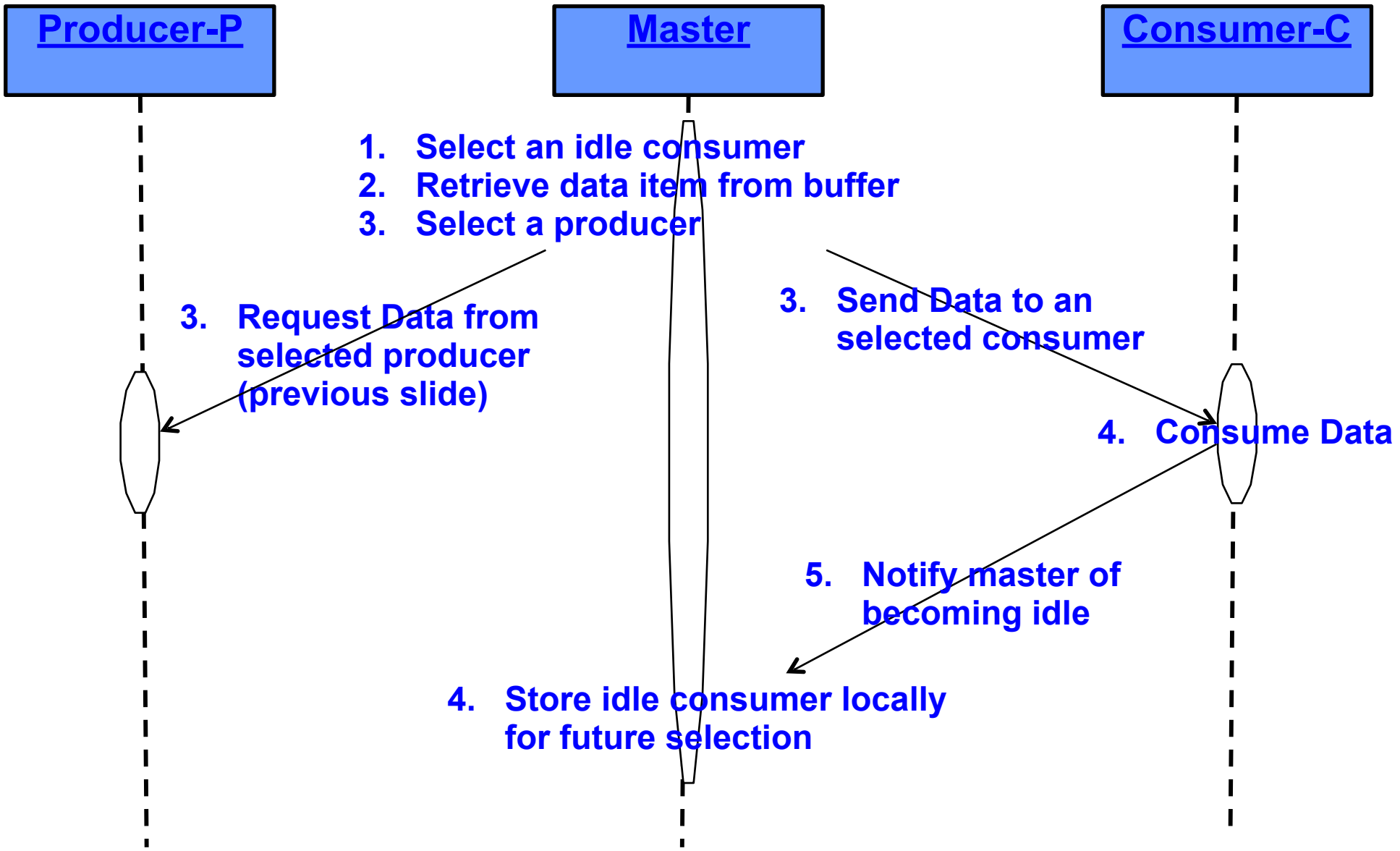


- Assume that B > P to allow for the case where producer messages may be in flight

Bounded Buffer Actor Interaction Diagram



Bounded Buffer Actor Interaction Diagram (contd)



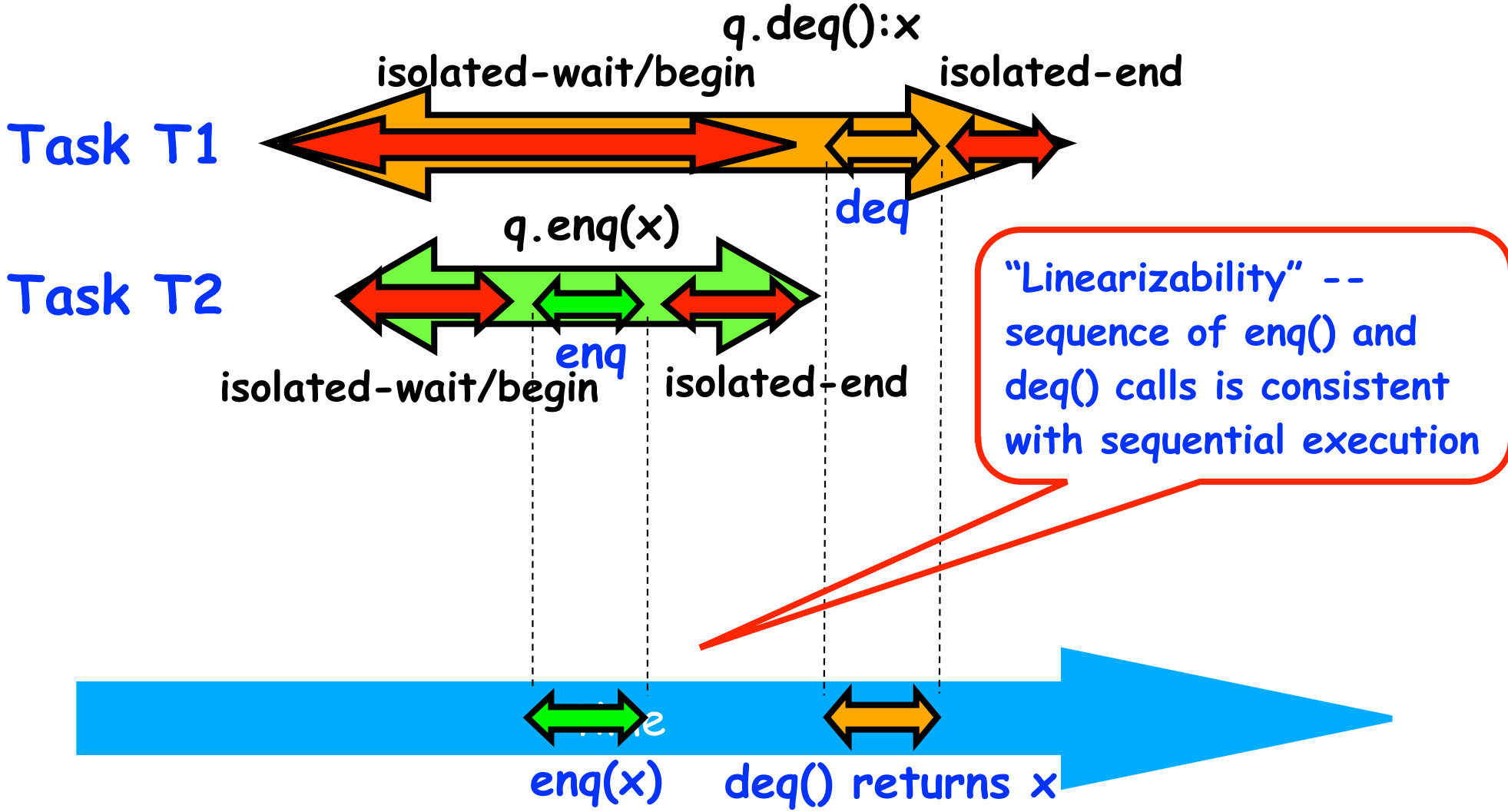
Actor Responsibilities

- **Master Actor**
  - —Sends requests to an *idle* producer when there is space in buffer
  - —Receives Data Items from the producers
  - —Stores data items in its bounded buffer
  - —Send data items to *idle* consumers, thus making space in buffer
  - —Receives notifications when consumers are idle
- **Producer Actor**
  - —Receives requests from Master to produce items
  - —Sends data items to the Master indirectly notifying it is now idle
- **Consumer Actor**
  - —Receives requests from Master to consume an item
  - —Sends notification to Master when it becomes idle

Correctness of a Concurrent Object

- Consider a simple FIFO (First In, First Out) queue as a canonical example of a concurrent object
  - Method q.enq(o) inserts object o at the tail of the queue
    - Assume that there is unbounded space available for all enq() operations to succeed
  - Method q.deq() removes and returns the item at the head of the queue.
    - Throws EmptyException if the queue is empty.
- What does it mean for a concurrent object like a FIFO queue to be correct?
  - What is a concurrent FIFO queue?
  - FIFO implies a strict temporal order
  - Concurrent implies an ambiguous temporal order

Describing the concurrent via the sequential



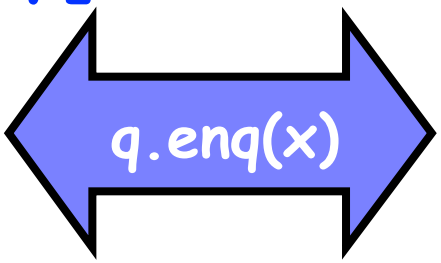
Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt

Informal definition of Linearizability

- Assume that each method call takes effect "instantaneously" at some specific point in time between its invocation and return.
- An *execution is linearizable* if we can choose *one set of* instantaneous points that is consistent with a sequential execution in which methods are executed at those points
  - It's okay if some other set of instantaneous points is not linearizable
- A *concurrent object is linearizable* if all its executions are linearizable
  - Linearizability is a "black box" test based on the object's behavior, not its internals

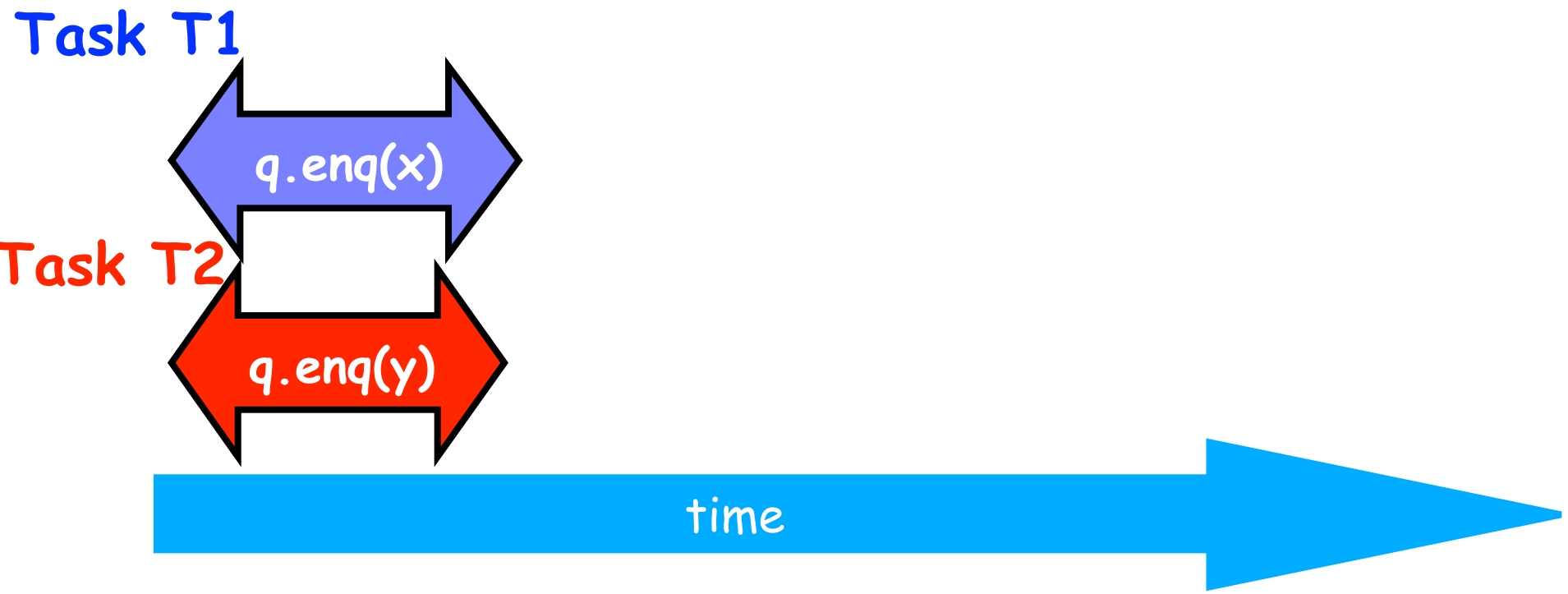
Example 1

Task T1



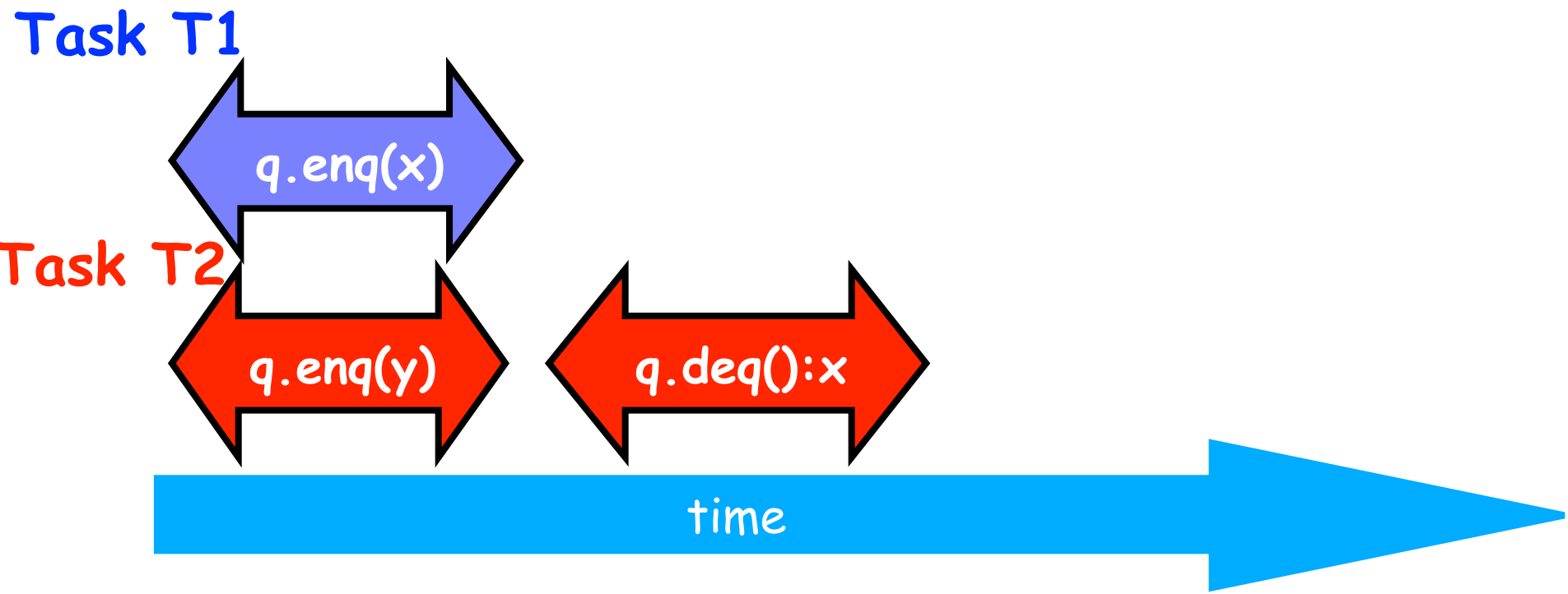
Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt

Example 1 (contd)



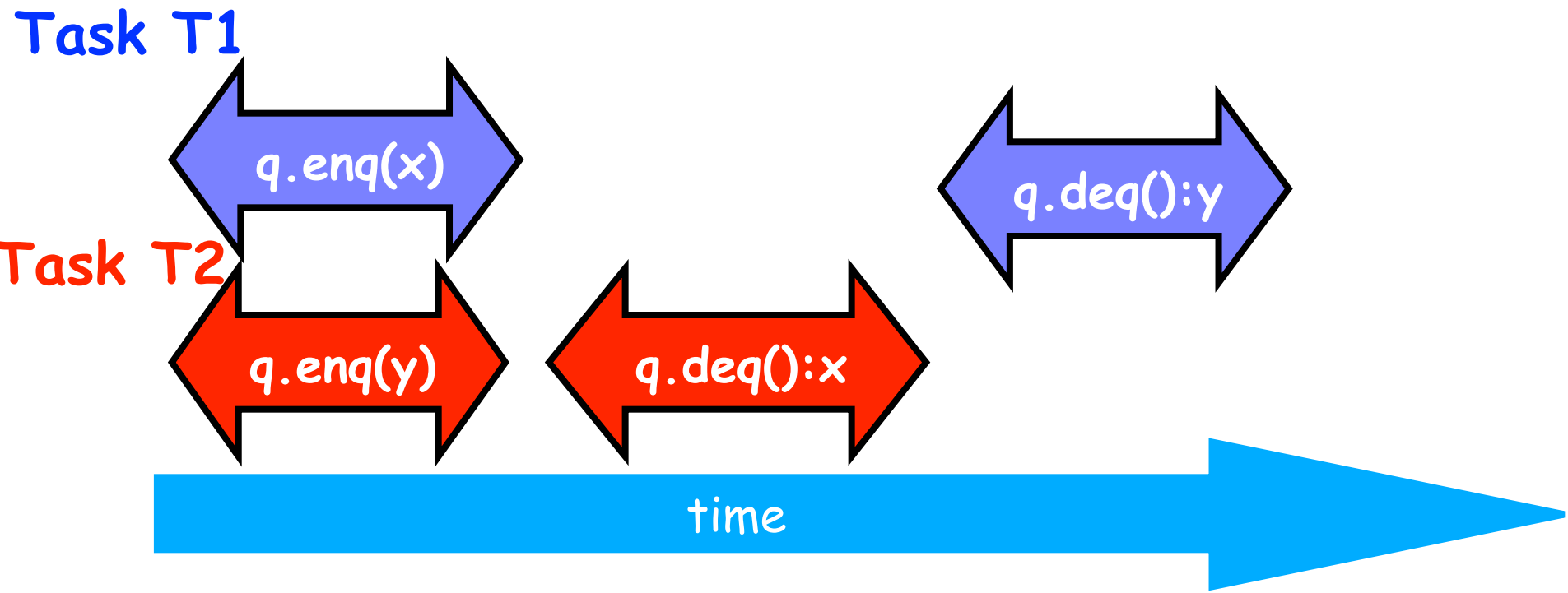
Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt

Example 1 (contd)



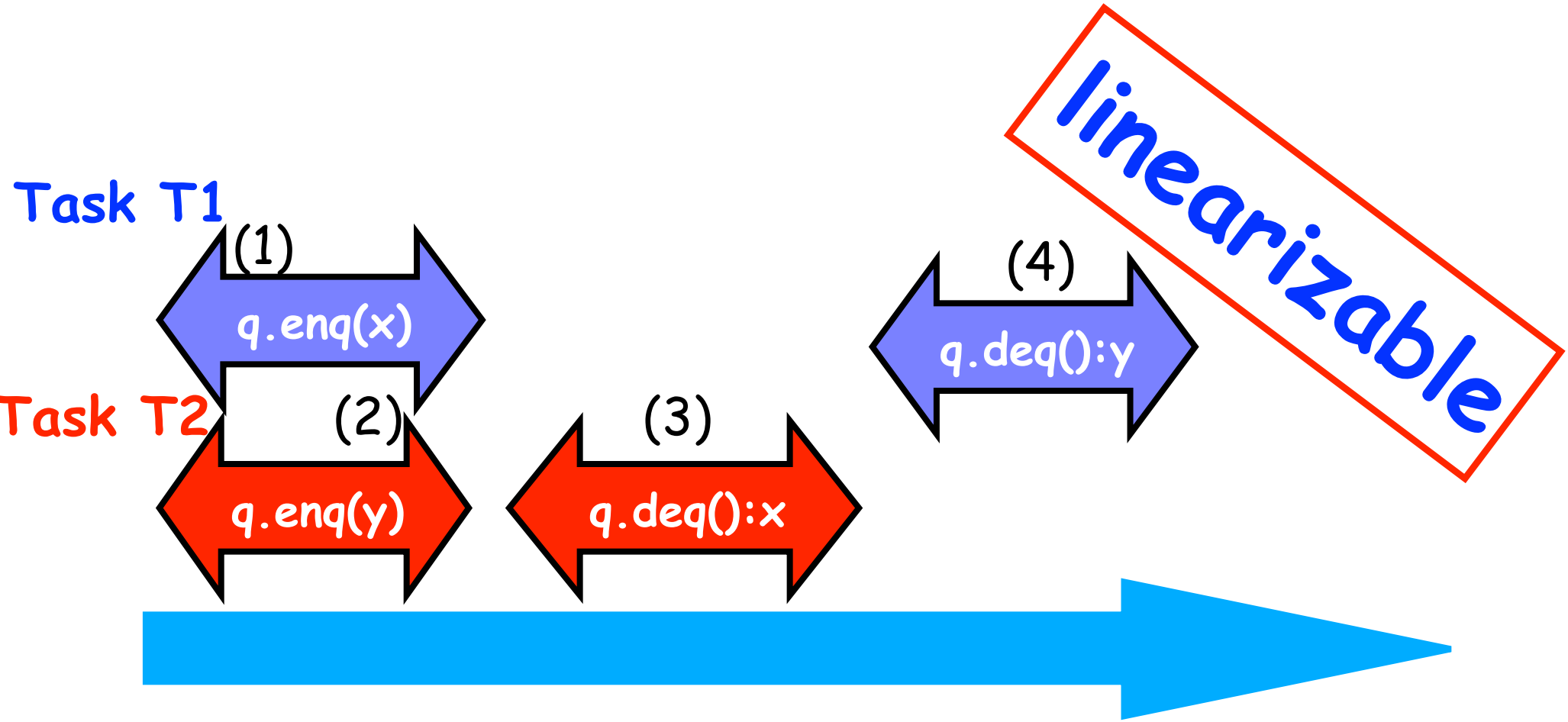
Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt

Example 1 (contd)



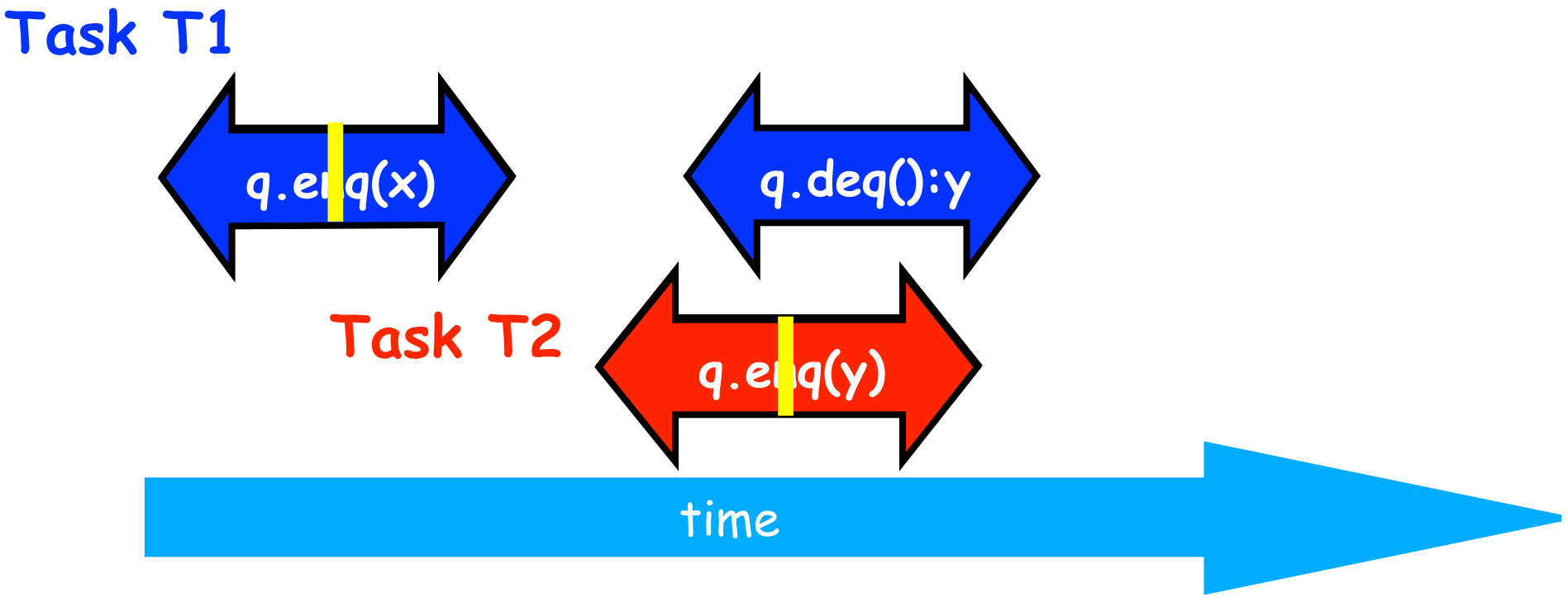
Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt

Example 1 (contd)



Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt

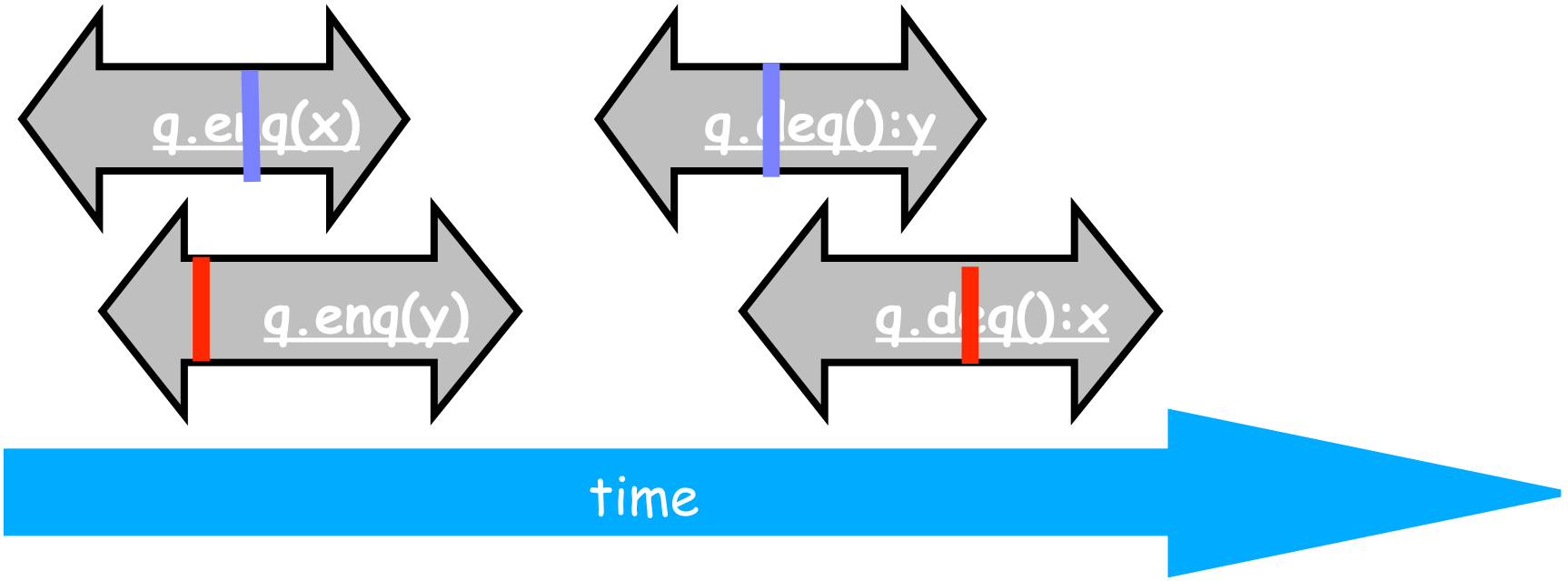
Example 2: is this execution linearizable?



Source: http://www.elsevierdirect.com/companions/9780123705914/Lecture%20Slides/03~Chapter_03.ppt

Example 3

Is this execution linearizable? How many possible linearizations does it have?



Example 4: execution of an isolated implementation of FIFO queue q

Is this a linearizable execution?

Time	Task A	Task B
0	Invoke `q.enq(x)`	
1	Work on `q.enq(x)`	
2	Work on `q.enq(x)`	
3	Return from `q.enq(x)`	
4		Invoke `q.enq(y)`
5		Work on `q.enq(y)`
6		Work on `q.enq(y)`
7		Return from `q.enq(y)`
8		Invoke `q.deq()`
9		Return `x` from `q.deq()`

Example 5: execution of a concurrent implementation of a FIFO queue q

Is this a linearizable execution?

Time	Task A	Task B
0	Invoke `q.enq(x)`	
1	Work on `q.enq(x)`	Invoke `q.enq(y)`
2	Work on `q.enq(x)`	Return from `q.enq(y)`
3	Return from `q.enq(x)`	
4		Invoke `q.deq()`
5		Return `x` from `q.deq()`

Linearizability of Concurrent Objects (Summary)

Concurrent object

- A concurrent object is an object that can correctly handle methods invoked in parallel by different tasks or threads
  - Examples: concurrent queue, AtomicInteger

Linearizability

- Assume that each method call takes effect "instantaneously" at some distinct point in time between its invocation and return.
- An execution is linearizable if we can choose instantaneous points that are consistent with a sequential execution in which methods are executed at those points
- An object is linearizable if all its possible executions are linearizable