COMP 322: Fundamentals of Parallel Programming

Lecture 28: Linearizability

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu

Rewrite the transferFunds() method below to use j.u.c. locks with calls to tryLock (see slide 4) instead of synchronized.

Your goal is to write a correct implementation that never deadlocks, unlike the buggy version below (which can deadlock).

Assume that each Account object already contains a reference to a ReentrantLock object dedicated to that object e.g., from.lock() returns the lock for the from object. Sketch your answer using pseudocode.

```
1.  public void transferFunds(Account from, Account to, int amount) {
2.    while (true) {
3.      // assume that trylock() does not throw an exception
4.      boolean fromFlag = from.lock.trylock();
5.      if (!fromFlag) continue;
6.      boolean toFlag = to.lock.trylock();
7.      if (!toFlag) { from.lock.unlock(); continue; }
8.      try { from.subtractFromBalance(amount);
9.            to.addToBalance(amount); break; }
10.     finally { from.lock.unlock(); to.lock.unlock(); }
11.   } // while
12. }
```
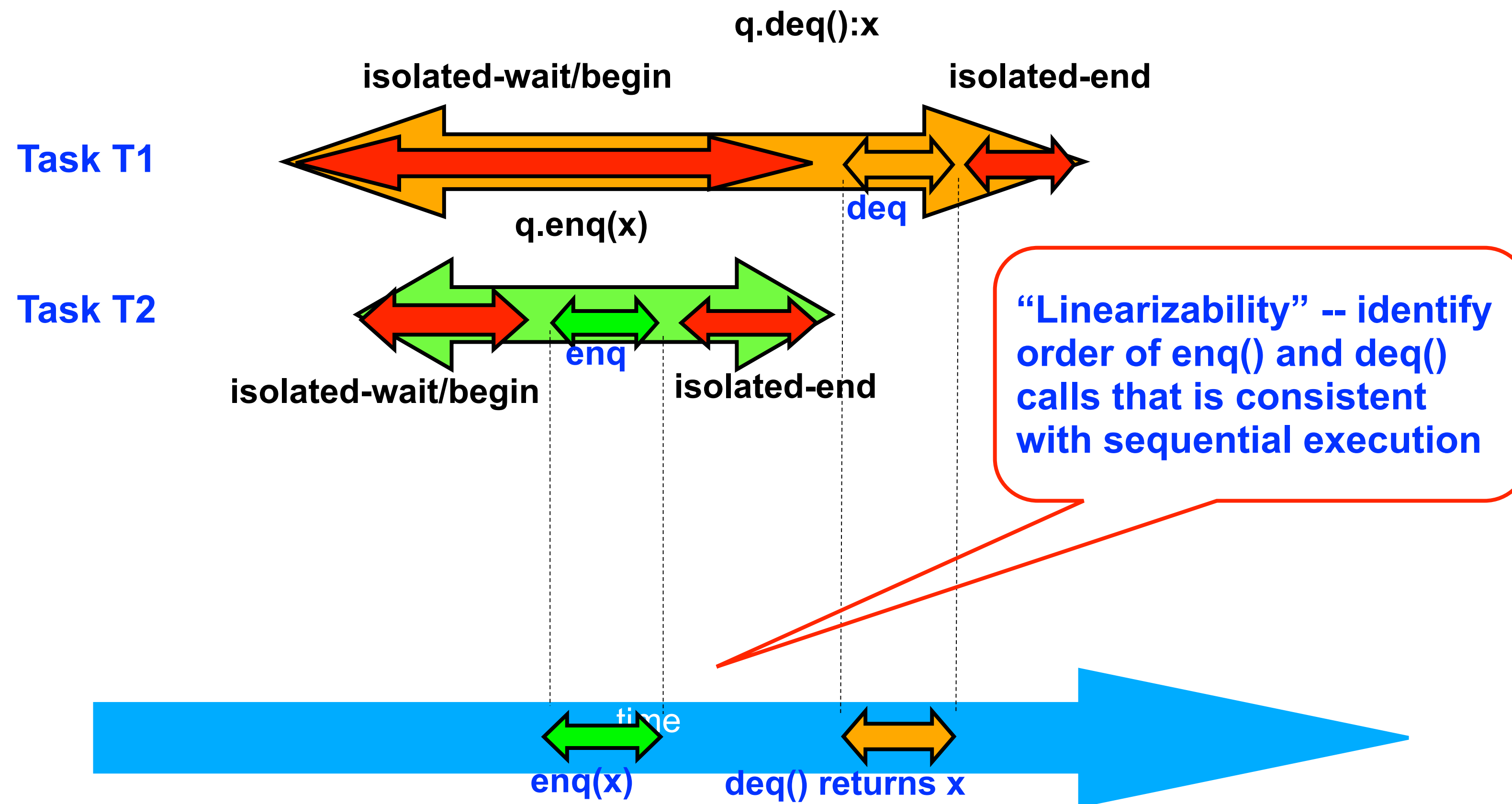
Linearizability: Correctness of Concurrent Objects

- A *concurrent object* is an *object* that can correctly handle *methods* invoked *concurrently* by different tasks or threads

    —e.g., AtomicInteger, ConcurrentHashMap, ConcurrentLinkedQueue, …

- For the discussion of linearizability, we will assume that the body of each method in a concurrent object is itself sequential

    —Assume that methods do not create threads or async tasks

- 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.

- Without seeing the implementation of the FIFO queue, we can tell if an execution of calls to enq() and deq() is correct or not, in a sequential program

- *How can we tell if the execution is correct for a parallel program?*

Linearization: Identifying a sequential order of concurrent method calls

q.deq():x

isolated-wait/begin   isolated-end

Task T1

q.enq(x)   deq

Task T2

                    enq

isolated-wait/begin   isolated-end

"Linearizability" -- identify order of enq() and deq() calls that is consistent with sequential execution

time

enq(x)   deq() returns x

Informal Definition of Linearizability

- Assume that each method call takes effect "instantaneously" at some point in time between its invocation and return.

- An *execution (schedule) 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
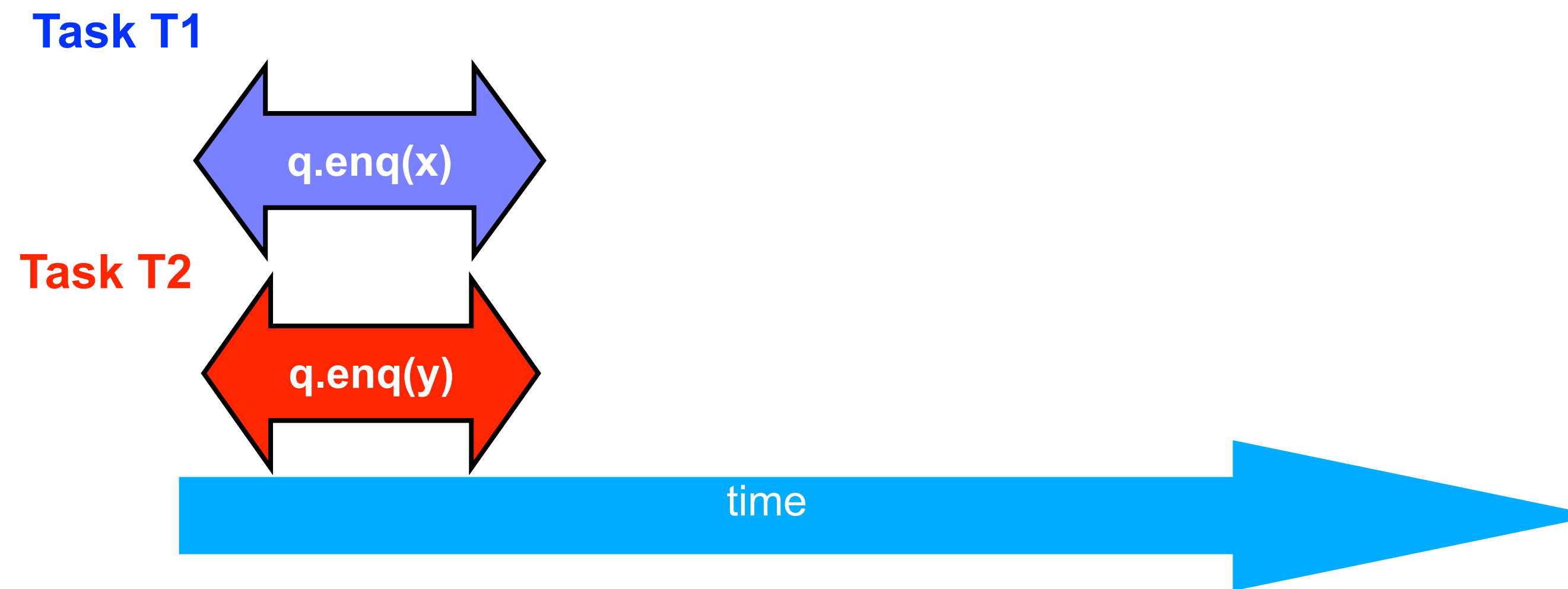
Example 1

Task T1

q.enq(x)

time

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

COMP 322, Spring 2020 (M.Joyner)

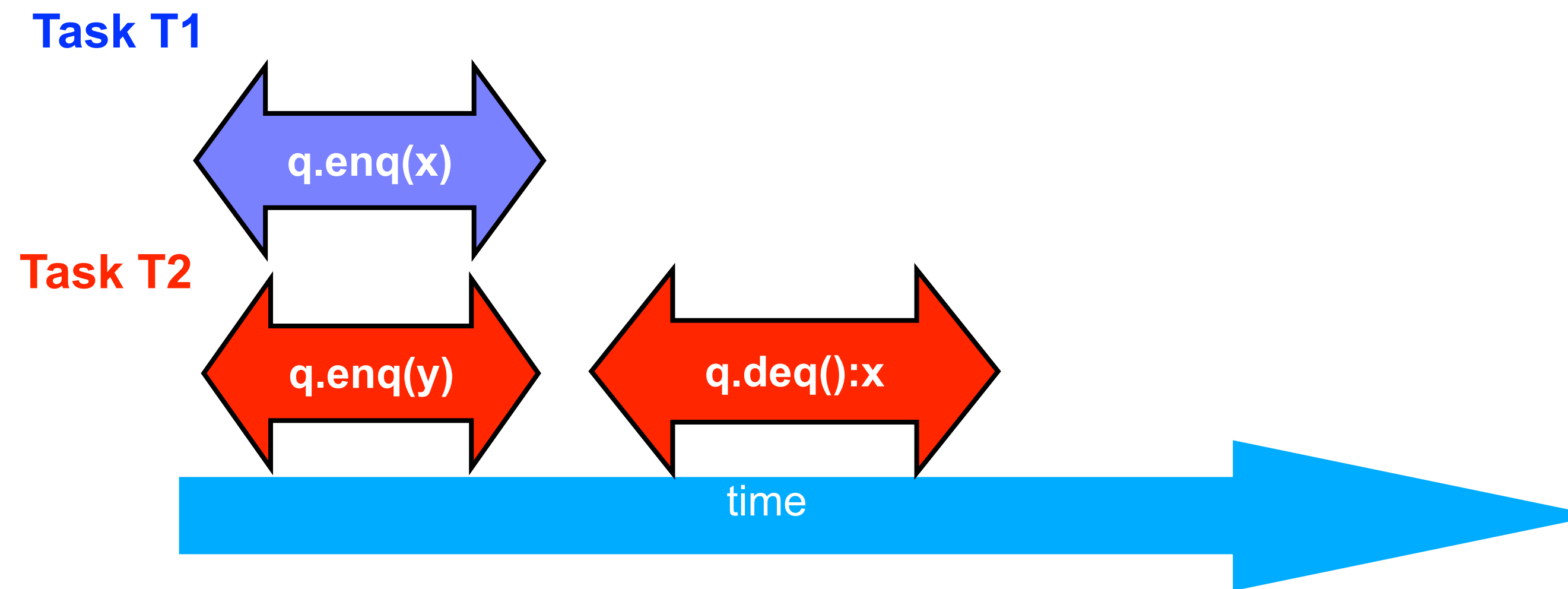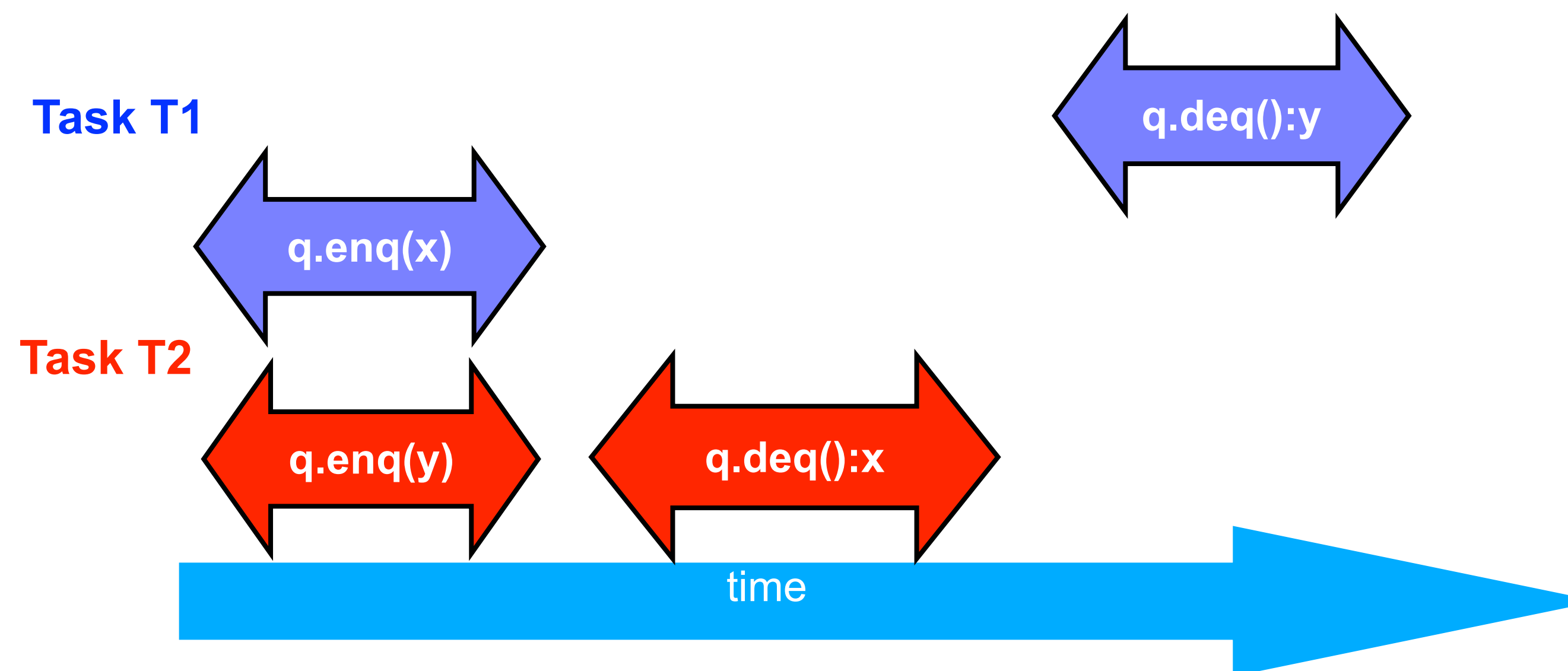# Example 1 cont.

Example 1 cont.

Task T1

q.enq(x)

Task T2

q.enq(y)

time

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

Example 1 cont.

Example 1 cont.

Task T1

q.deq():y

q.enq(x)

Task T2

q.enq(y)   q.deq():x

time

Example 1: is this execution linearizable?

Example 2: is this execution linearizable?

Task T1
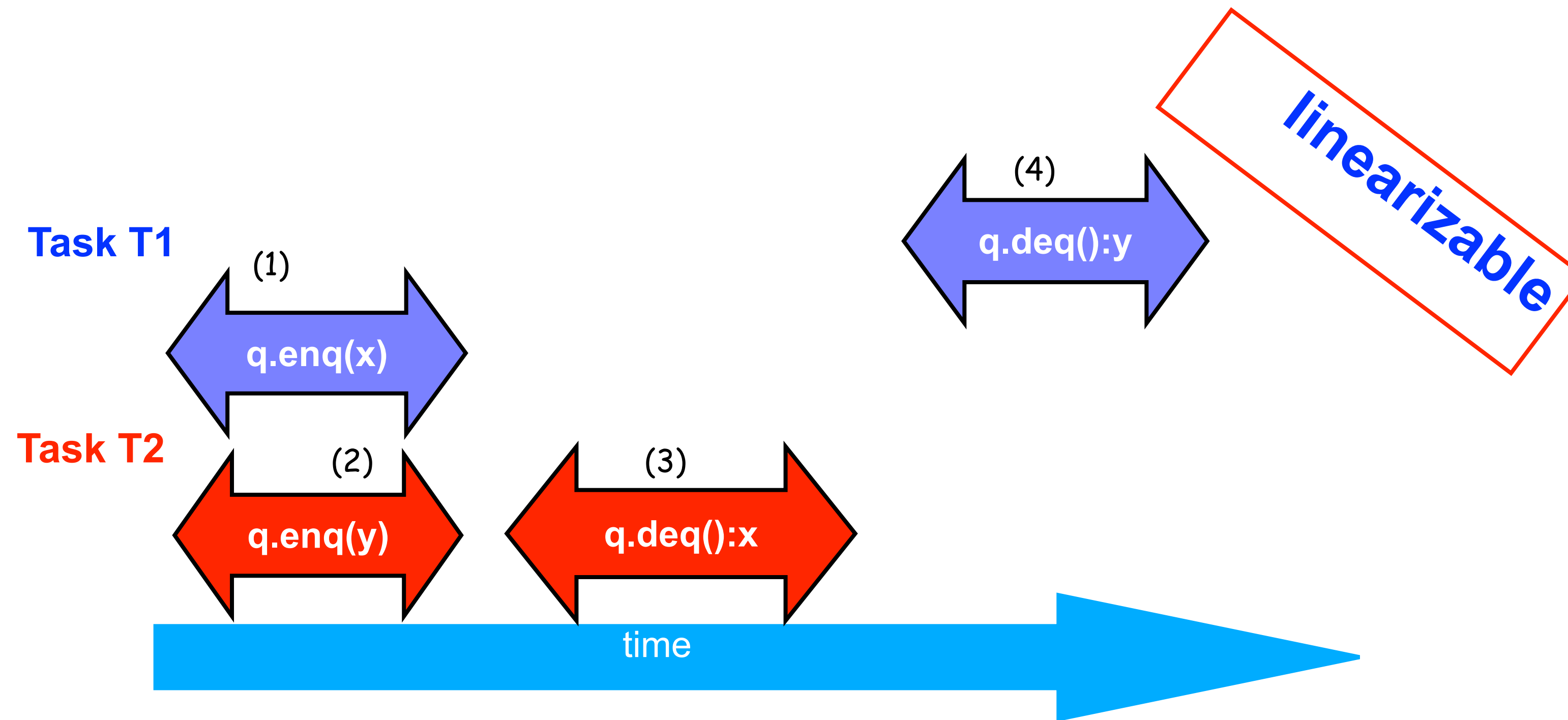
q.enq(x)

q.deq():y

Task T2

q.enq(y)

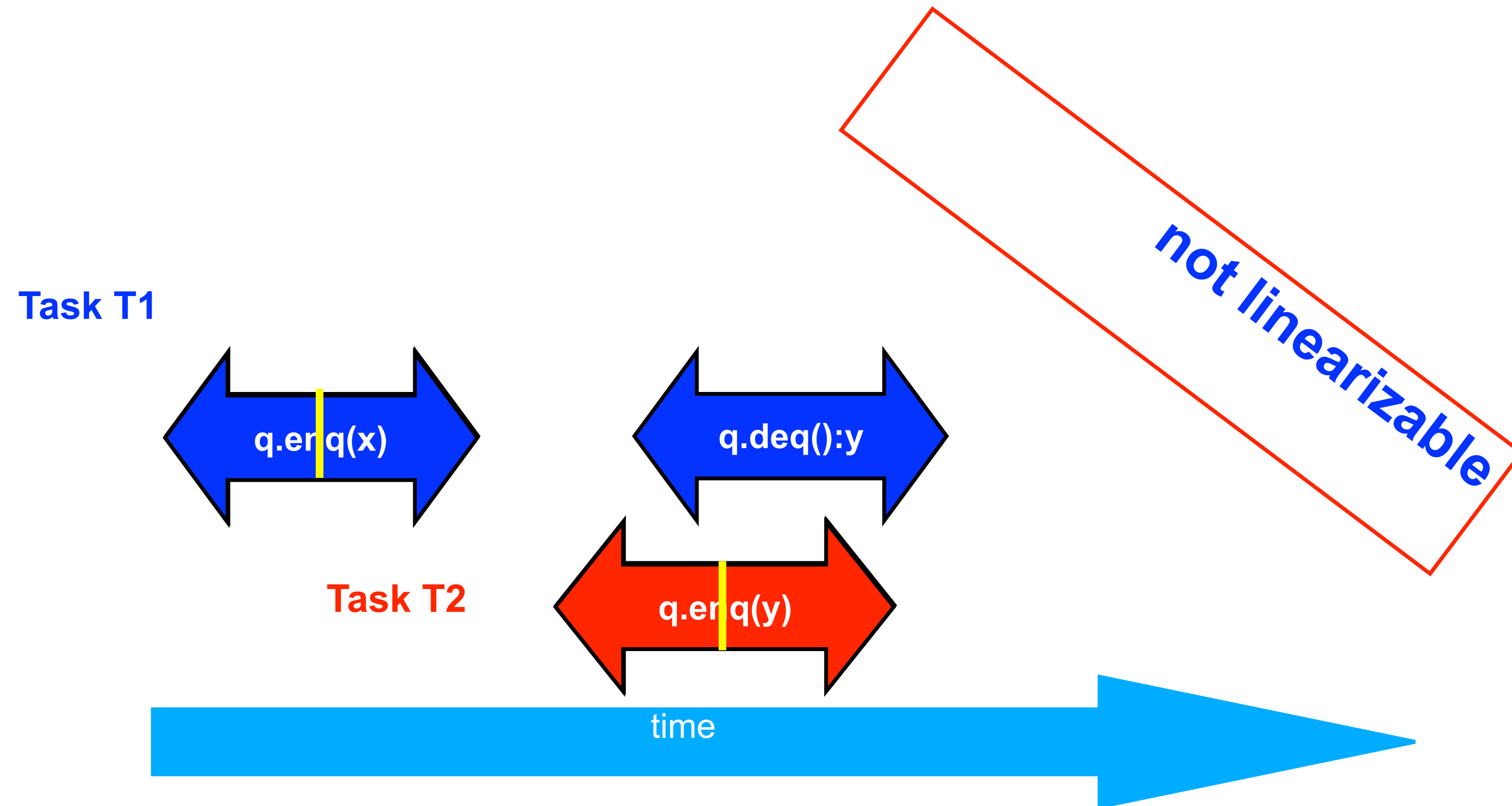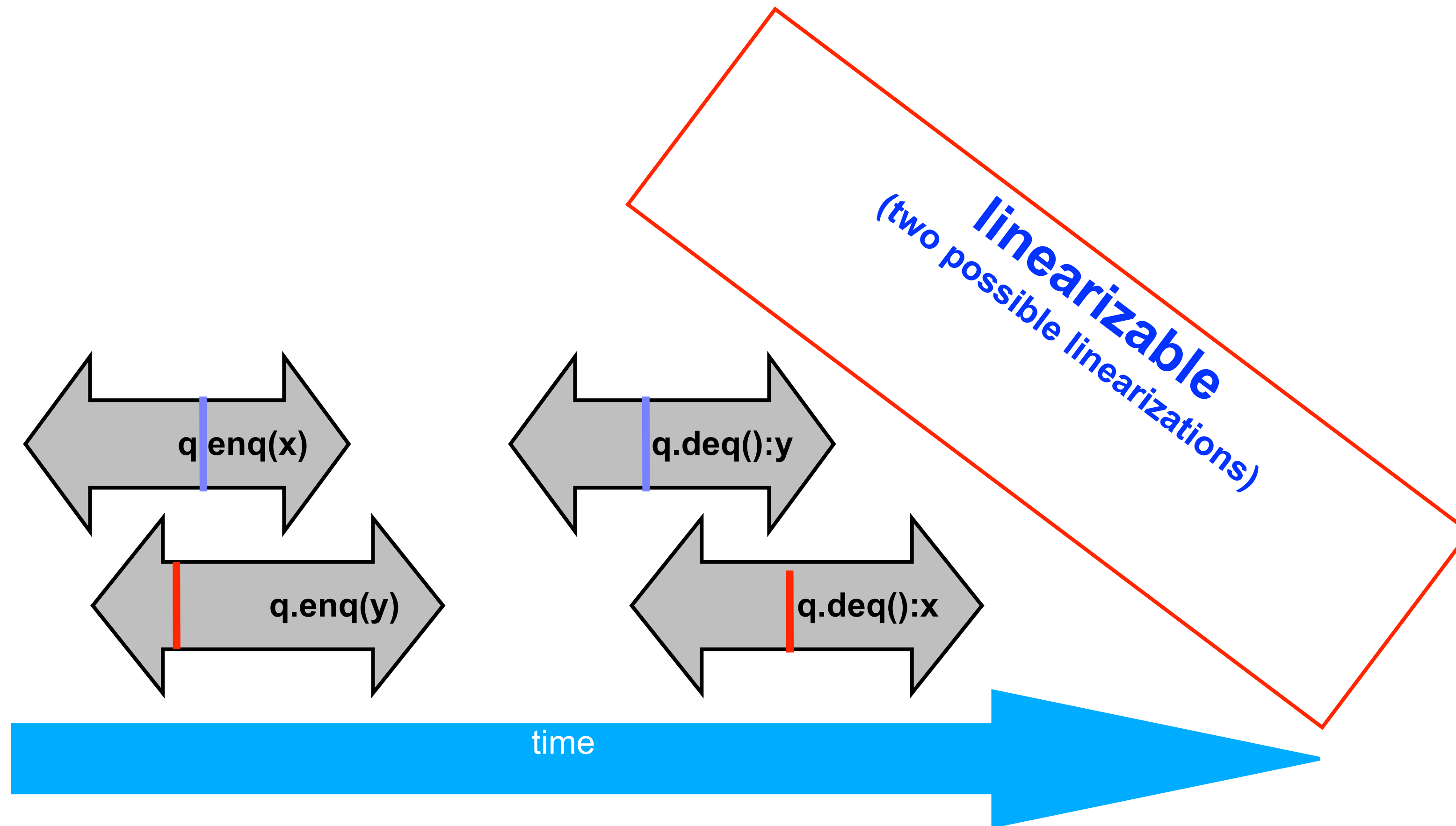not linearizable

time

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?

q.enq(x)

q.deq():y

q.enq(y)

q.deq():x

linearizable
(two possible linearizations)

time

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()

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

Announcements & Reminders

- The entire written + programming (Checkpoint #3) is due by Friday, April 3rd at 11:59pm

- Quiz for Unit 6 is due Monday, April 6th at 11:59pm

- Hw # 4 will be available today, due Wednesday, April 22nd by 11:59pm
  —Checkpoint 1 due Monday, April 13th by 11:59pm