Shared Memory Consistency Models

Nitin Vaidya
UIUC

Algorithm 1

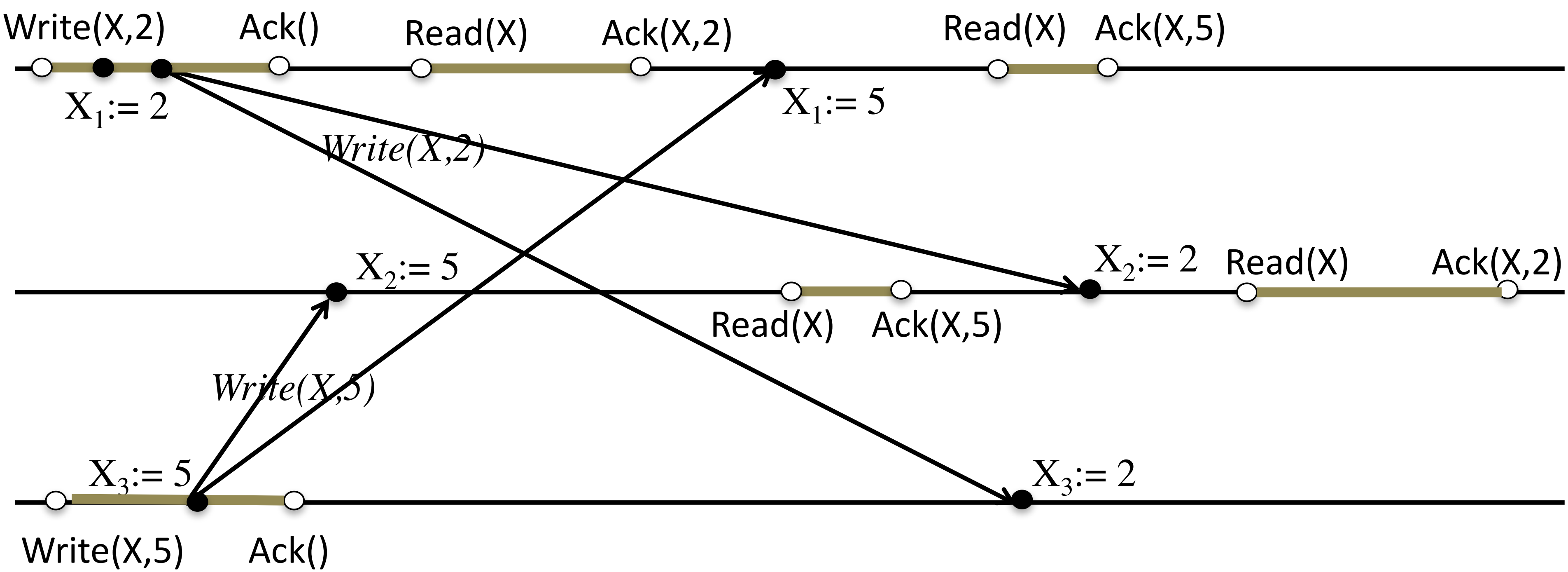


Figure 1: Algorithm 1

Algorithm 2

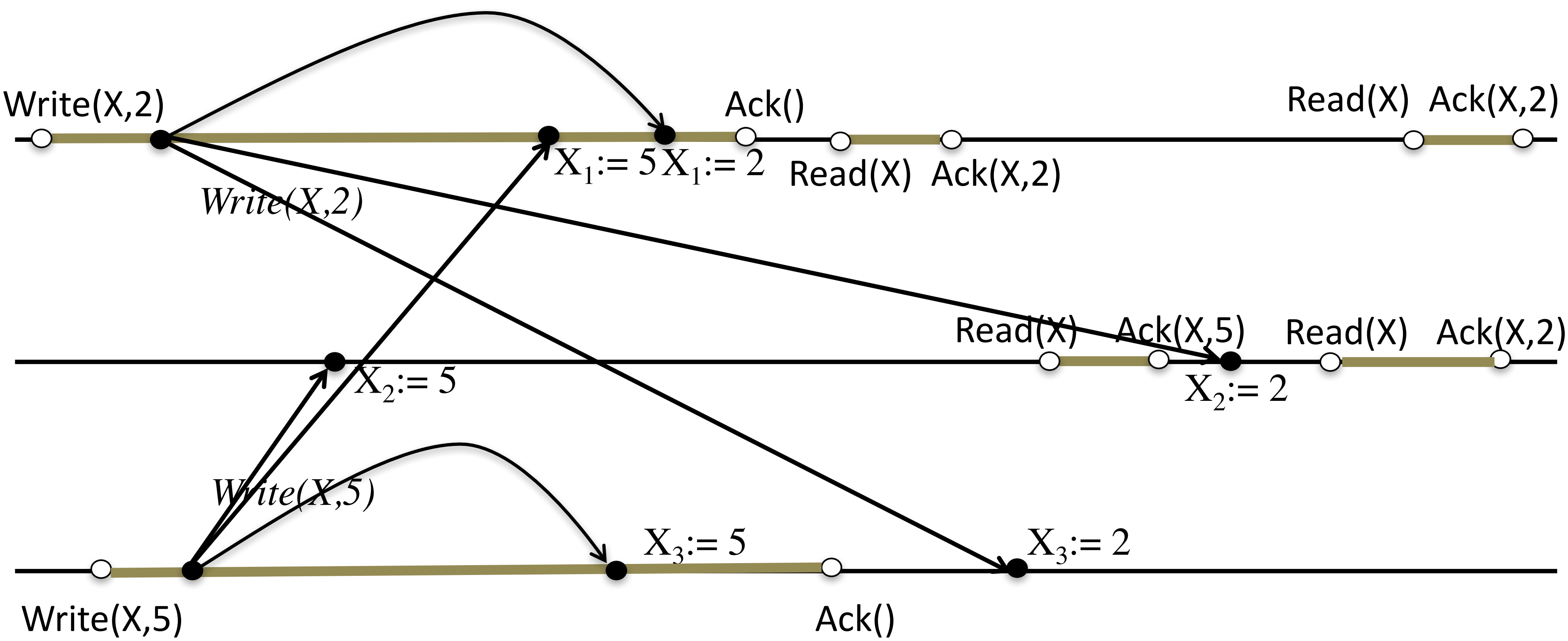


Figure 2: Algorithm 2

The figure shows the time at which the totally-ordered multicast messages are *delivered*

Algorithm 3

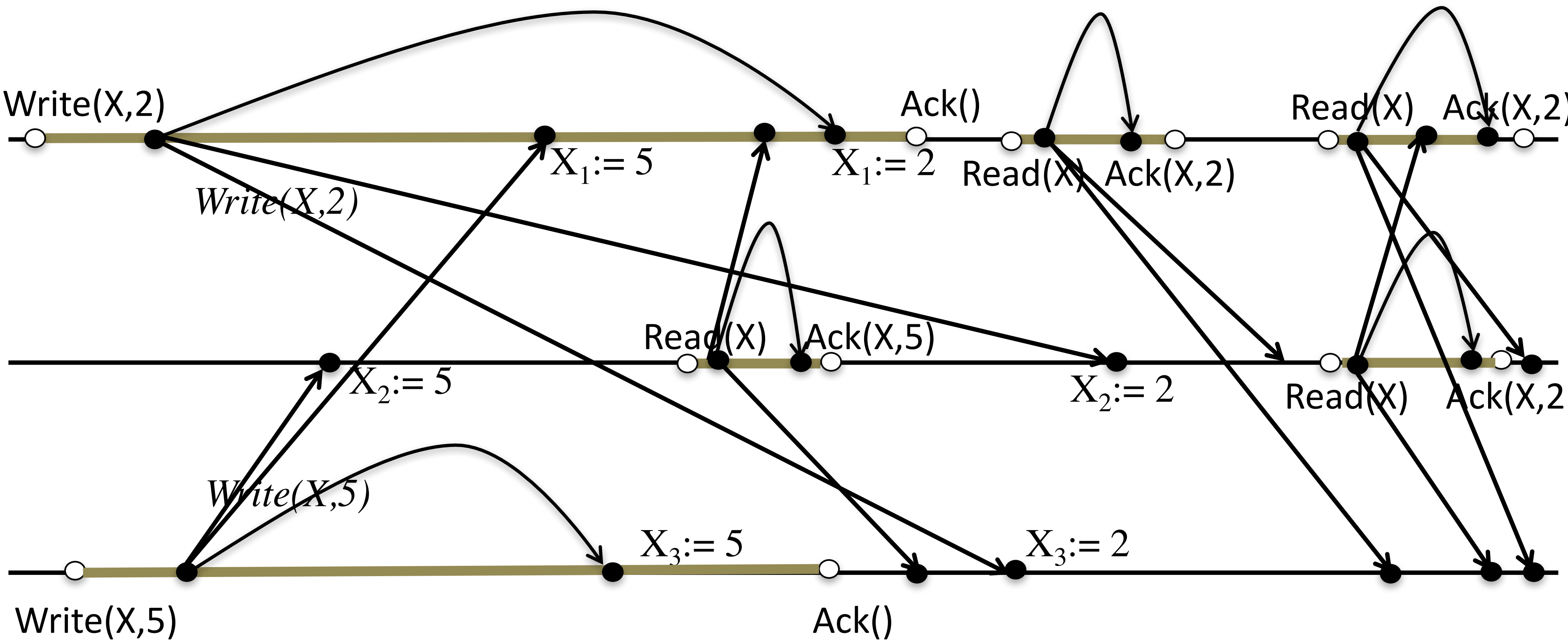


Figure 3: Algorithm 3

The figure shows the time at which the totally-ordered multicast messages are *delivered*

Now let us consider just the operation invocations and their response.

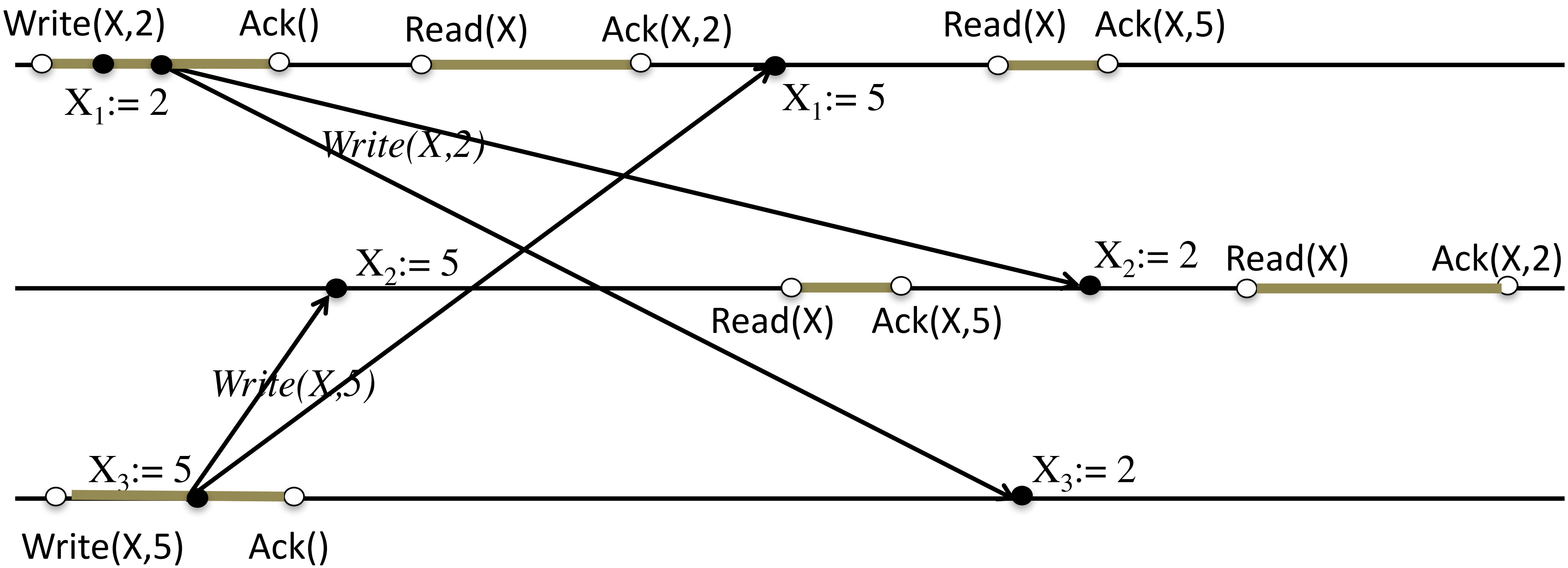


Figure 1: Algorithm 1

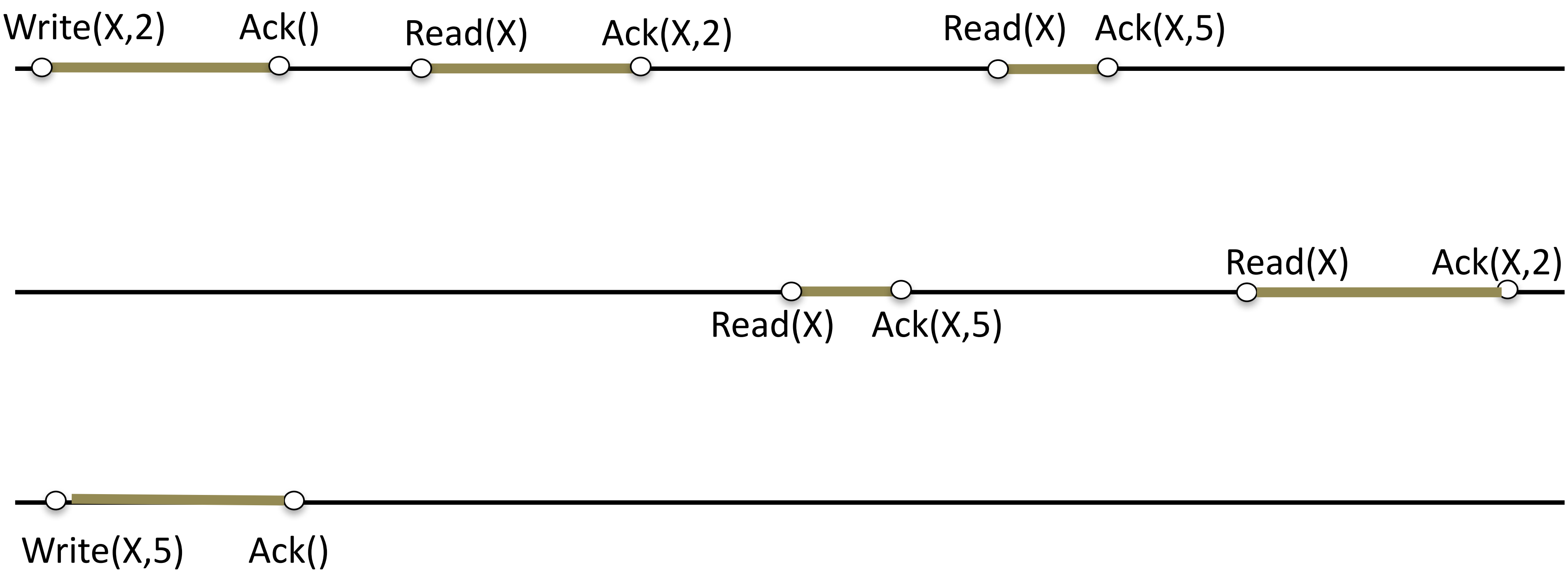


Figure 4: Redrawn Figure 1

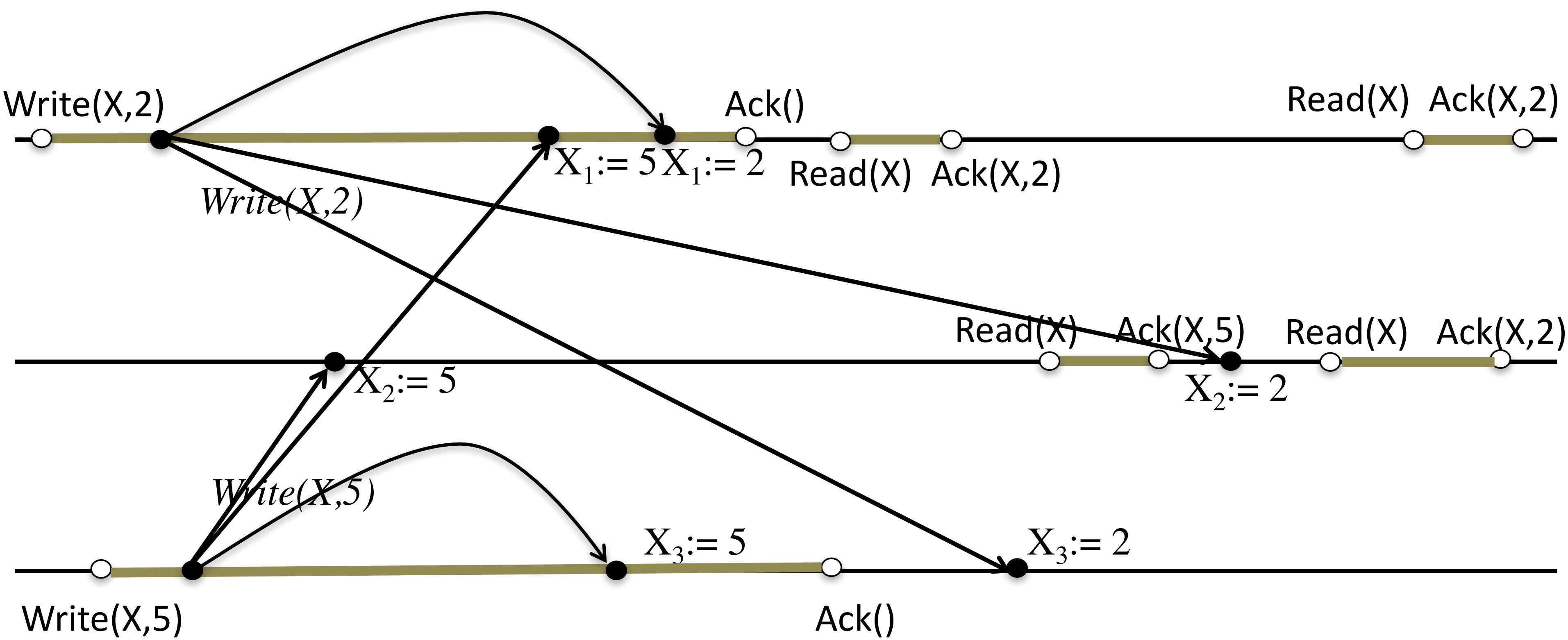


Figure 2: Algorithm 2

The figure shows the time at which the totally-ordered multicast messages are *delivered*

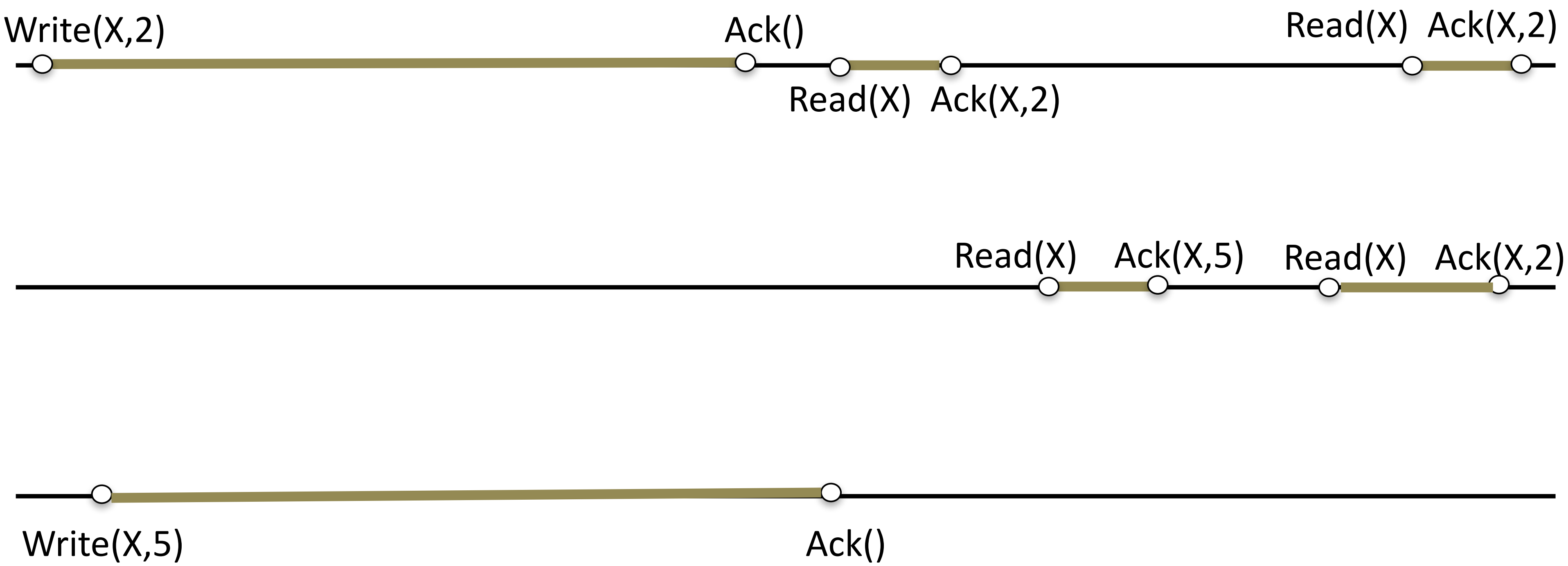


Figure 5: Redrawn Figure 2

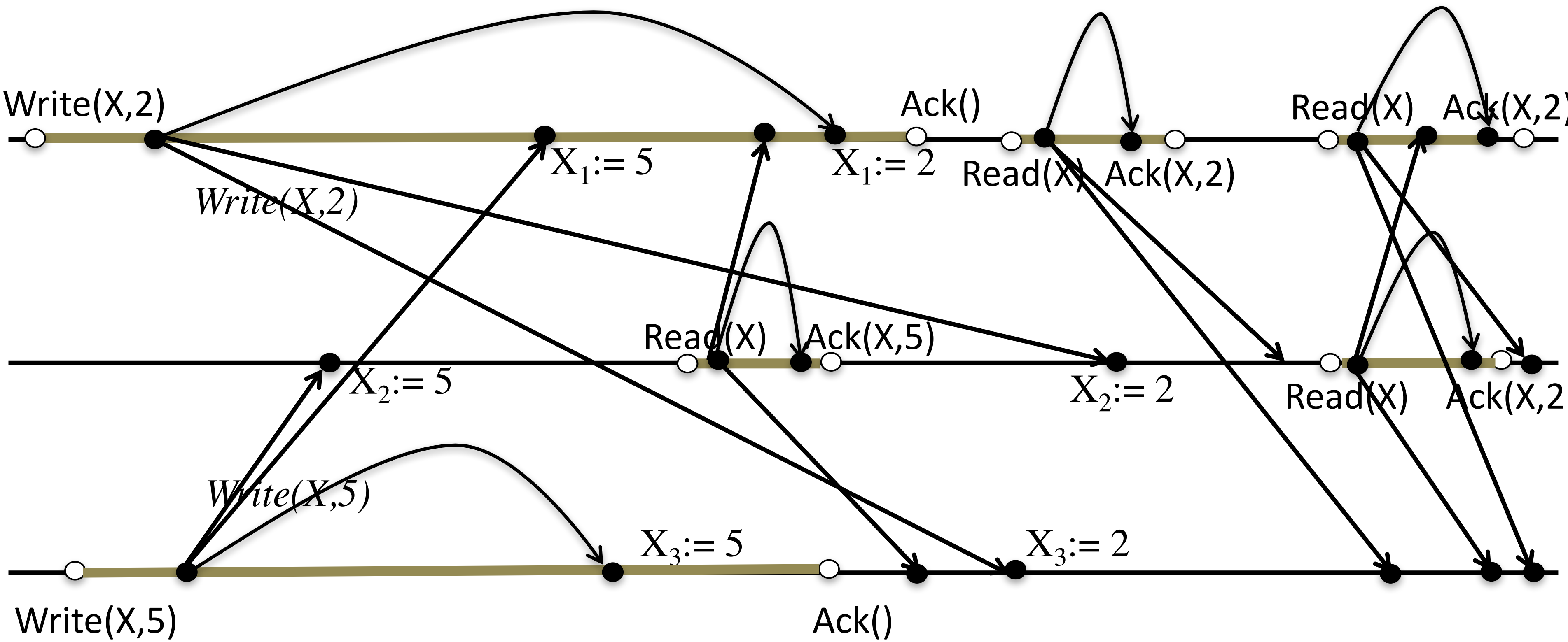


Figure 3: Algorithm 3

The figure shows the time at which the totally-ordered multicast messages are *delivered*

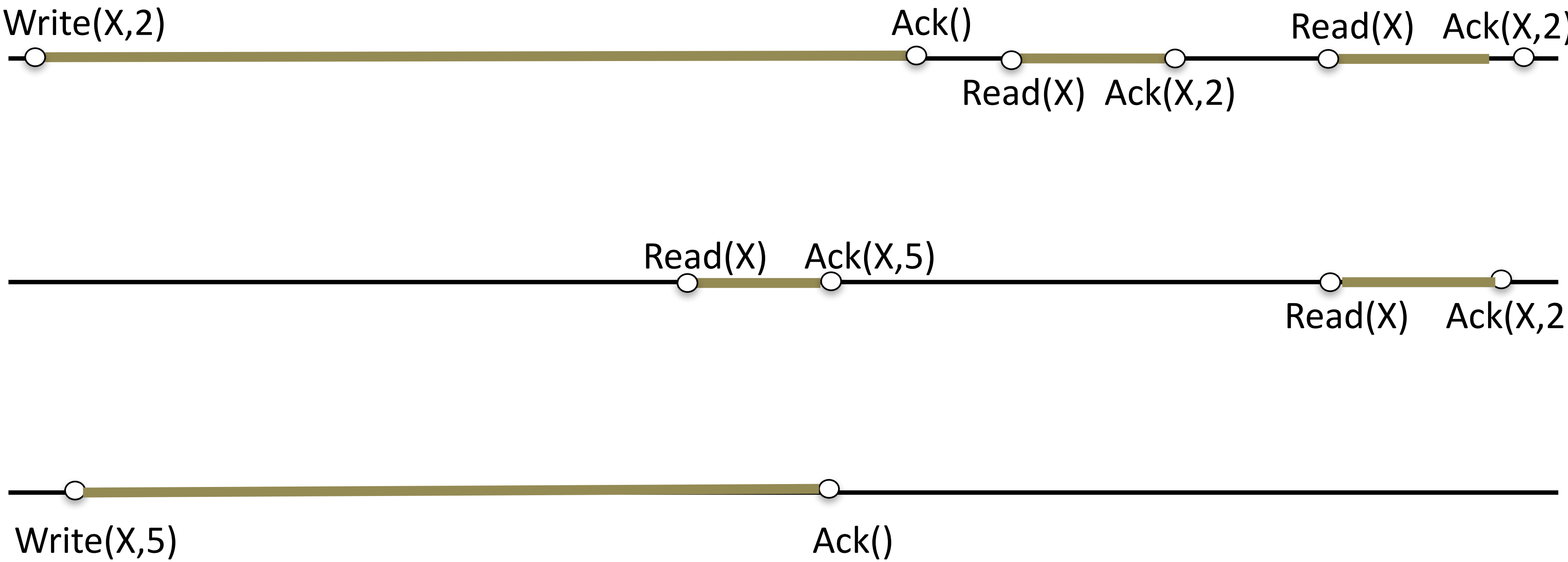
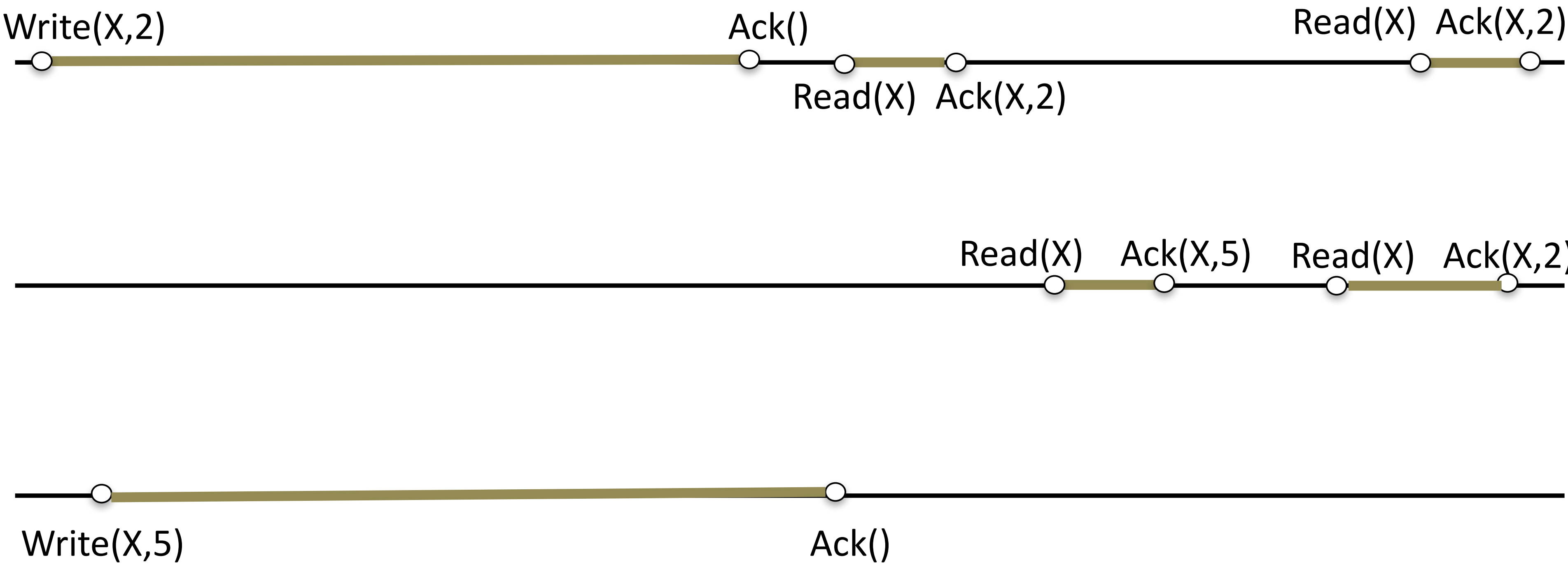


Figure 6: Redrawn Figure 3

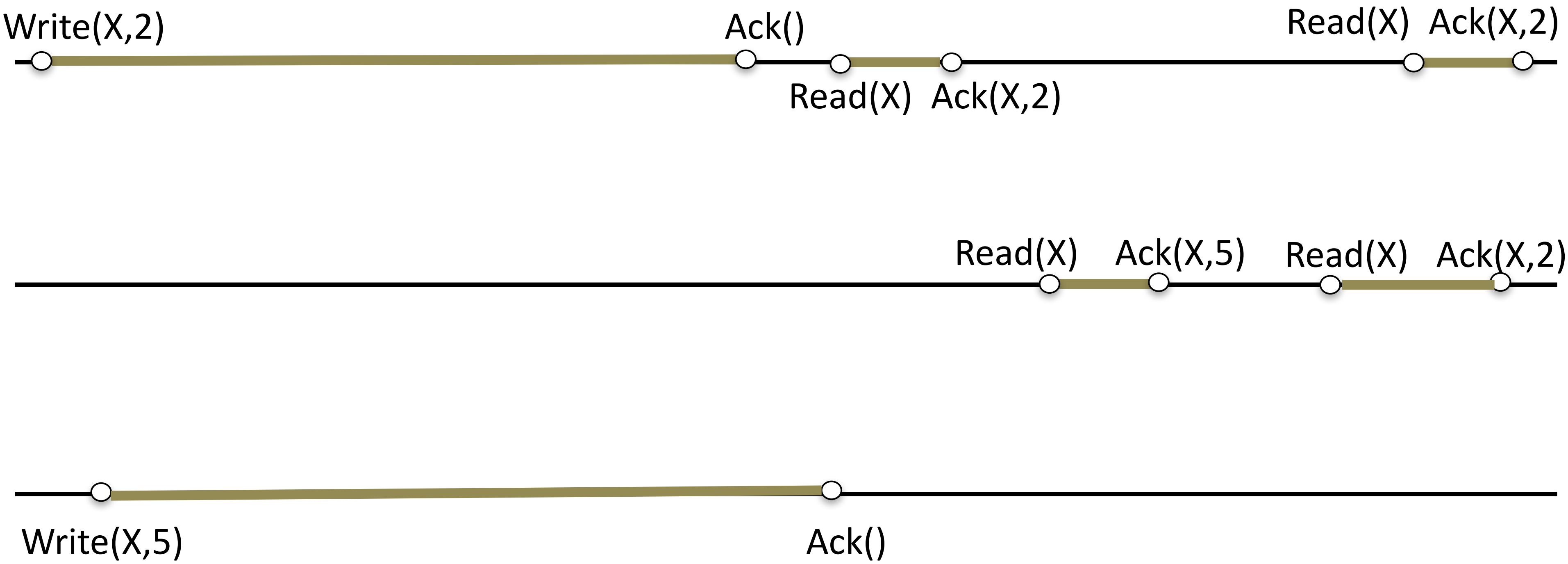
Permutations

Figure 5:



$$Write_1(X,2),\ Write_3(X,5),\ Read_1(X,2),\ Read_2(X,5),\ Read_2(X,2),\ Read_1(X,2)$$

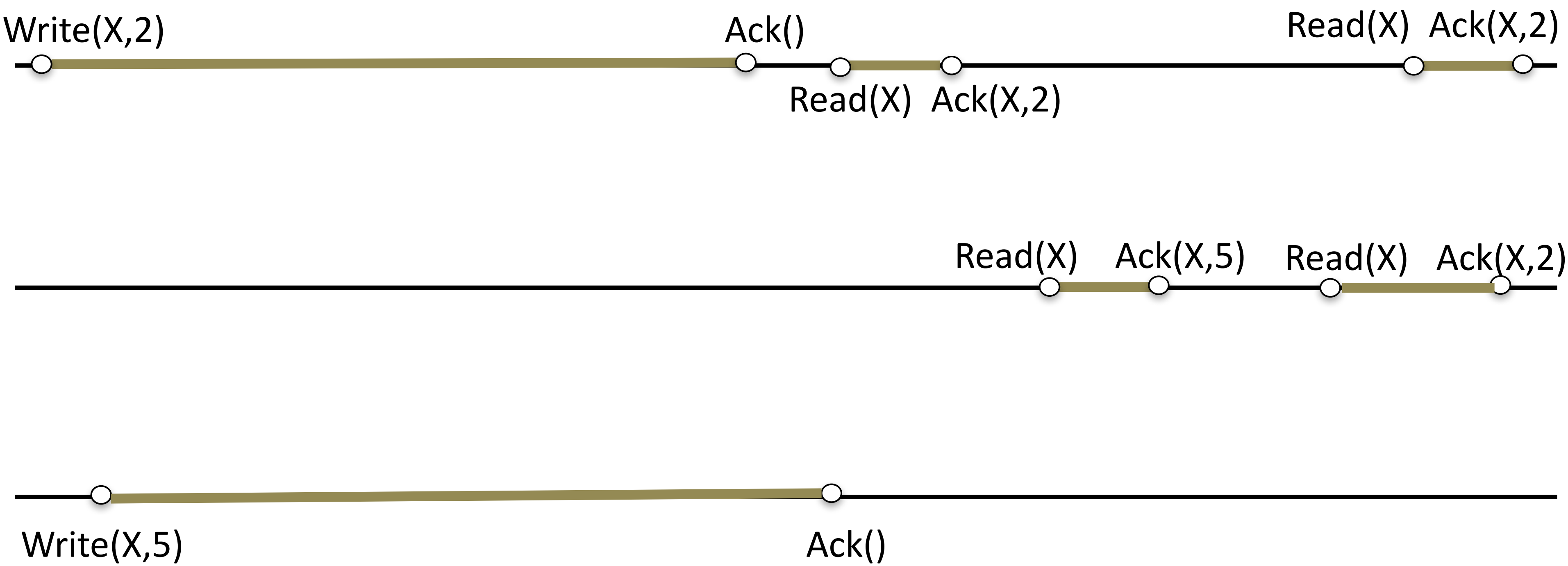
Figure 5:



$$Write_1(X,2),\ Write_3(X,5),\ Read_1(X,2),\ Read_2(X,5),\ Read_2(X,2),\ Read_1(X,2)$$

Permutation per-process order preserving

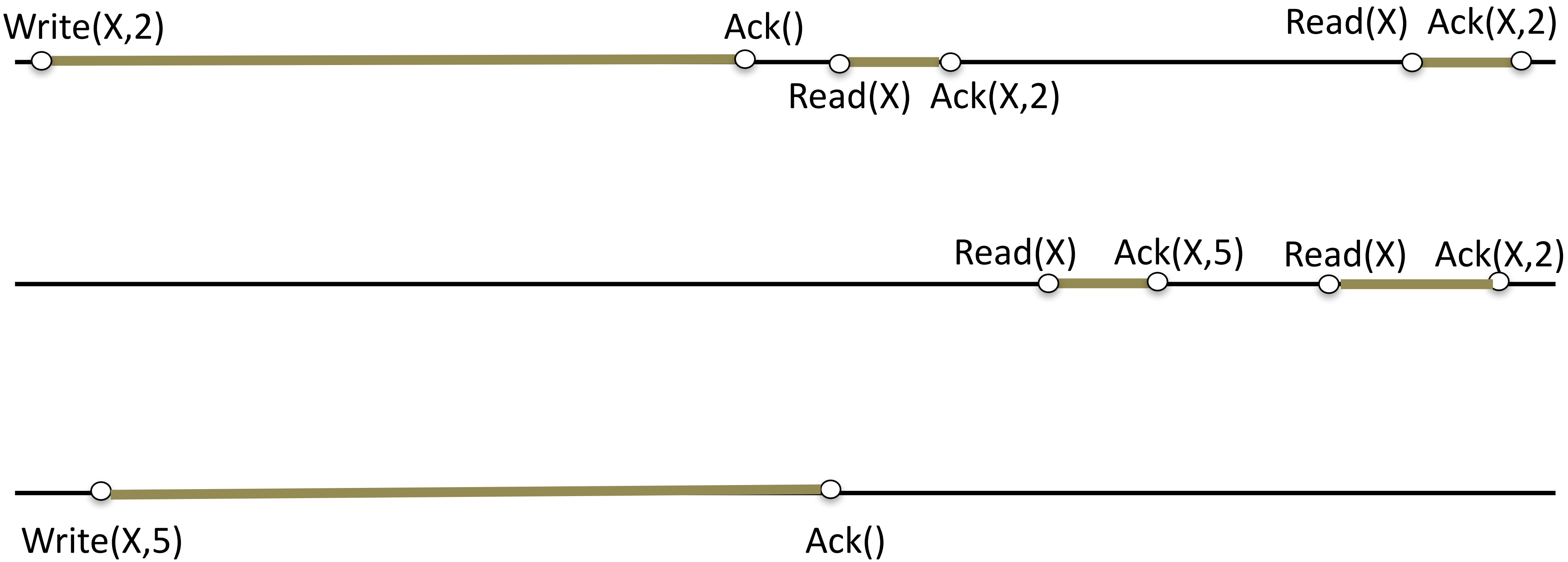
Figure 5:



$$Write_1(X,2),\ Write_3(X,5),\ Read_1(X,2),\ Read_2(X,5),\ Read_2(X,2),\ Read_1(X,2)$$

Permutation NOT valid

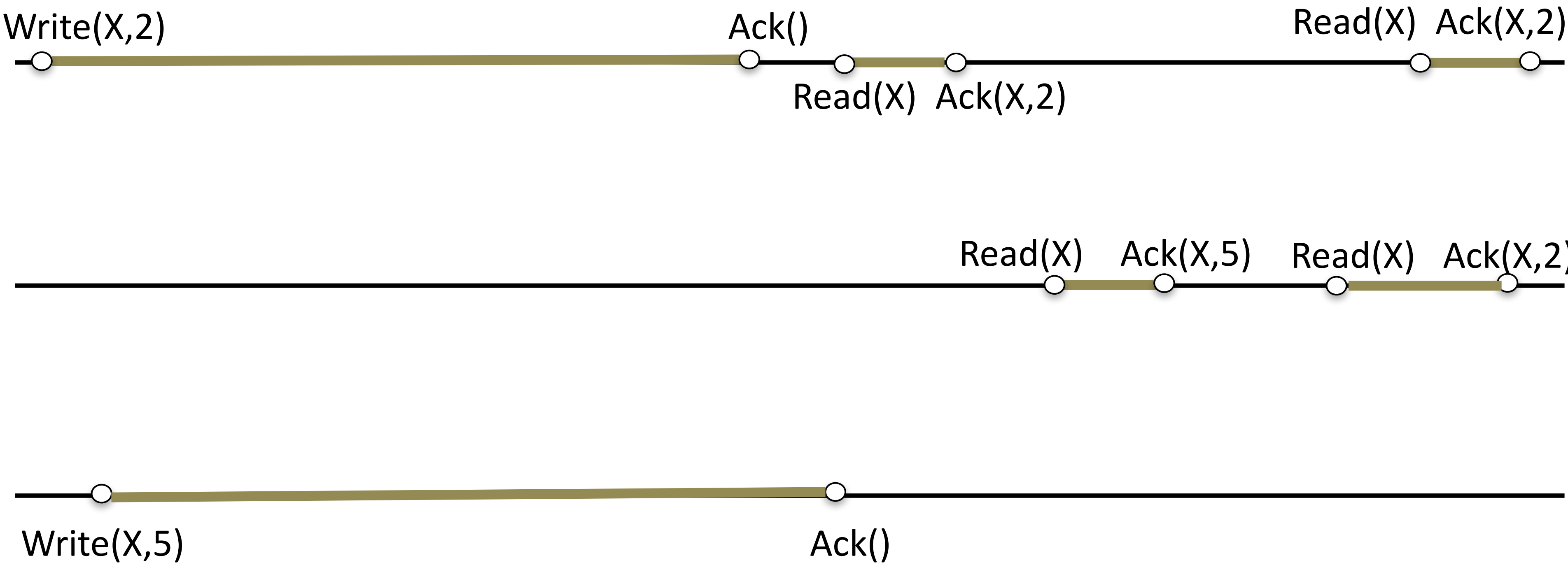
Figure 5:



$$Write_3(X,5),\ Read_2(X,5),\ Write_1(X,2),\ Read_2(X,2),\ Read_1(X,2),\ Read_1(X,2)$$

Permutation valid (and per-process order-preserving)

Figure 5:

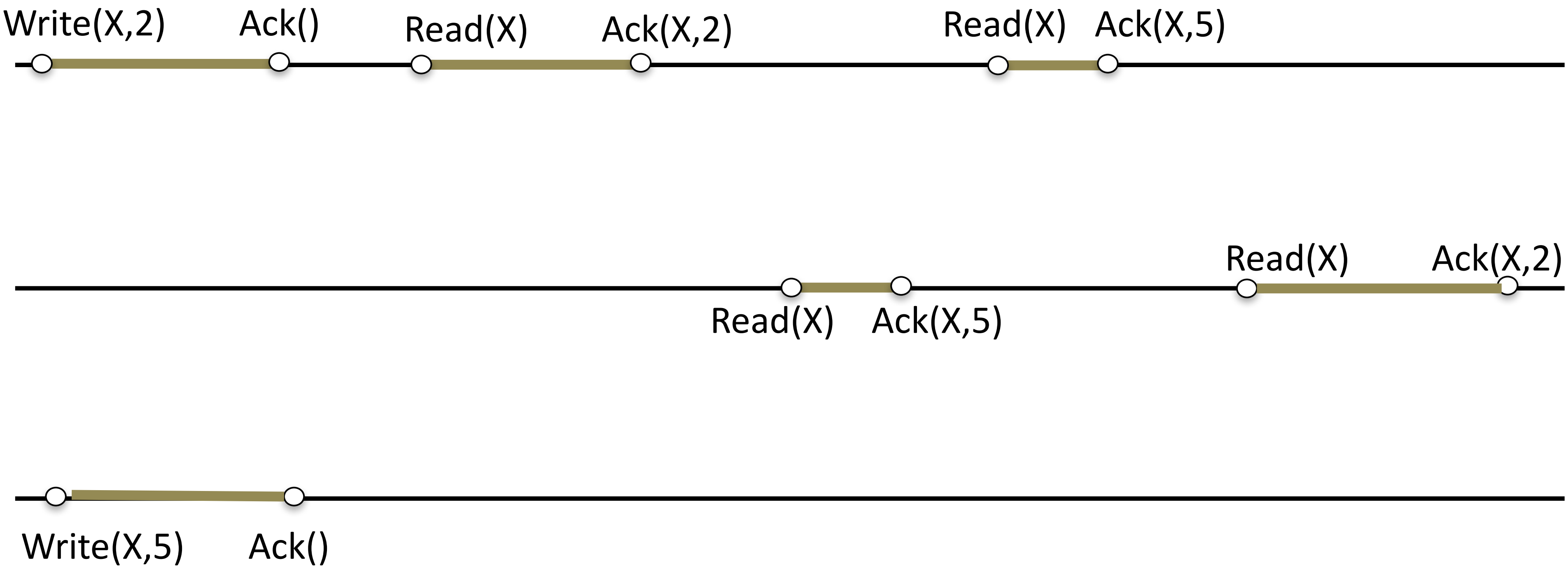


$$Write_3(X,5),\ Read_2(X,5),\ Write_1(X,2),\ Read_2(X,2),\ Read_1(X,2),\ Read_1(X,2)$$

$$Write_3(X,5),\ Read_2(X,5),\ Write_1(X,2),\ Read_1(X,2),\ Read_2(X,2),\ Read_1(X,2)$$

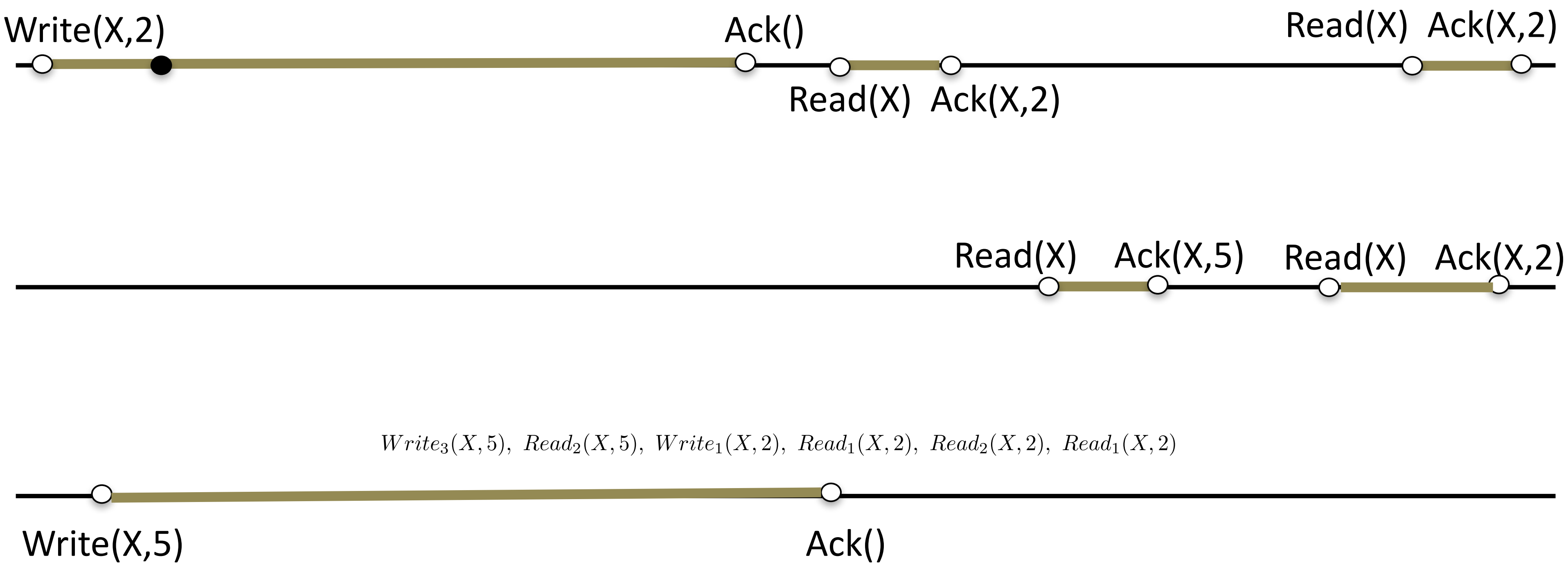
Such permutations not necessarily unique

Figure 4



Is there a valid and per-process order-preserving permutation?

Figure 5:

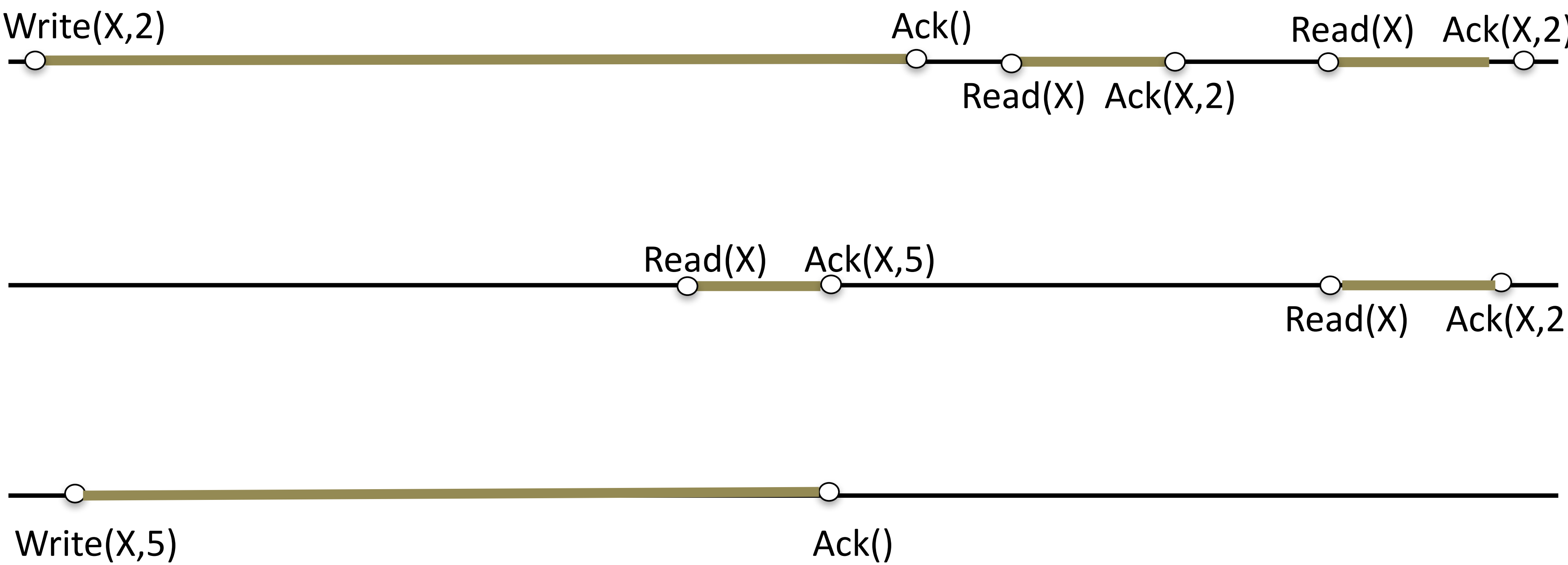


$$Write_3(X,5),\ Read_2(X,5),\ Write_1(X,2),\ Read_2(X,2),\ Read_1(X,2),\ Read_1(X,2)$$

Permutation valid (and per-process order-preserving)

But not real-time order-preserving

Figure 6



$$Write_3(X,5),\ Read_2(X,5),\ Write_1(X,2),\ Read_1(X,2),\ Read_2(X,2),\ Read_1(X,2)$$

Valid, per-process order preserving, real-time order-preserving

Consistency Model

Linearizability

An execution is linearizable if there exists a permutation that is

valid,

per-process order-preserving, and

real-time order-preserving

Linearizability

Intuitively …

Each operation in a linearizable execution appears to “take effect” instantaneously at some time between its invocation and its response

This point of time is called its *linearization point*

Linearization Points

If we can find linearization points such that the permutation of the operations as per the real-time order of the linearization points is valid

then the execution is linearizable

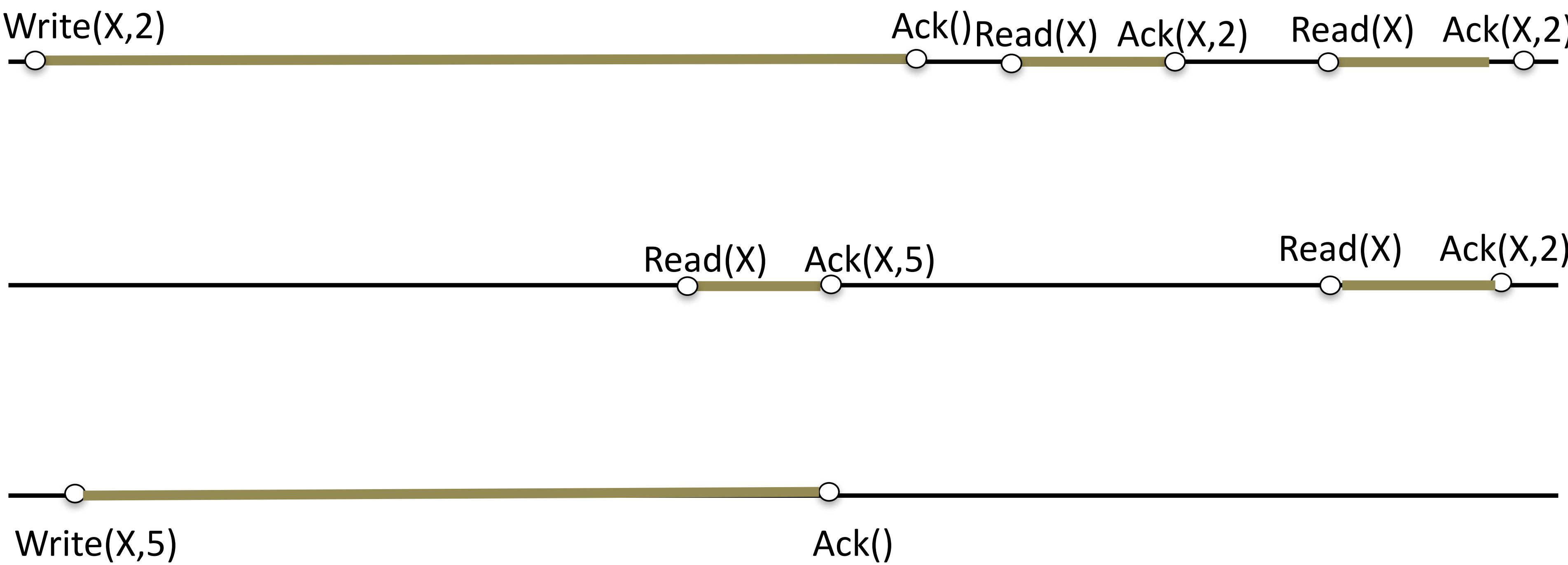


Figure 6 … can we find suitable linearization points ?

$$Write_3(X,5),\ Read_2(X,5),\ Write_1(X,2),\ Read_1(X,2),\ Read_2(X,2),\ Read_1(X,2)$$

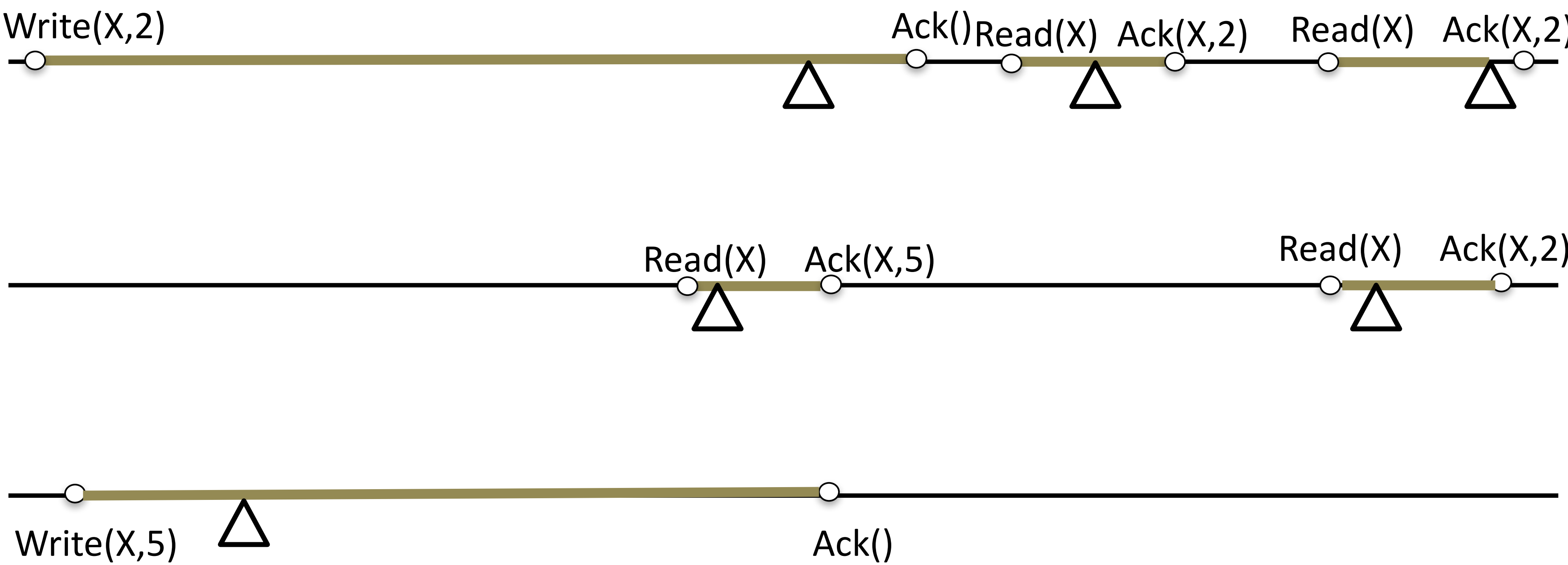


Figure 7: Execution of Figure 6 with linearization points marked by triangles

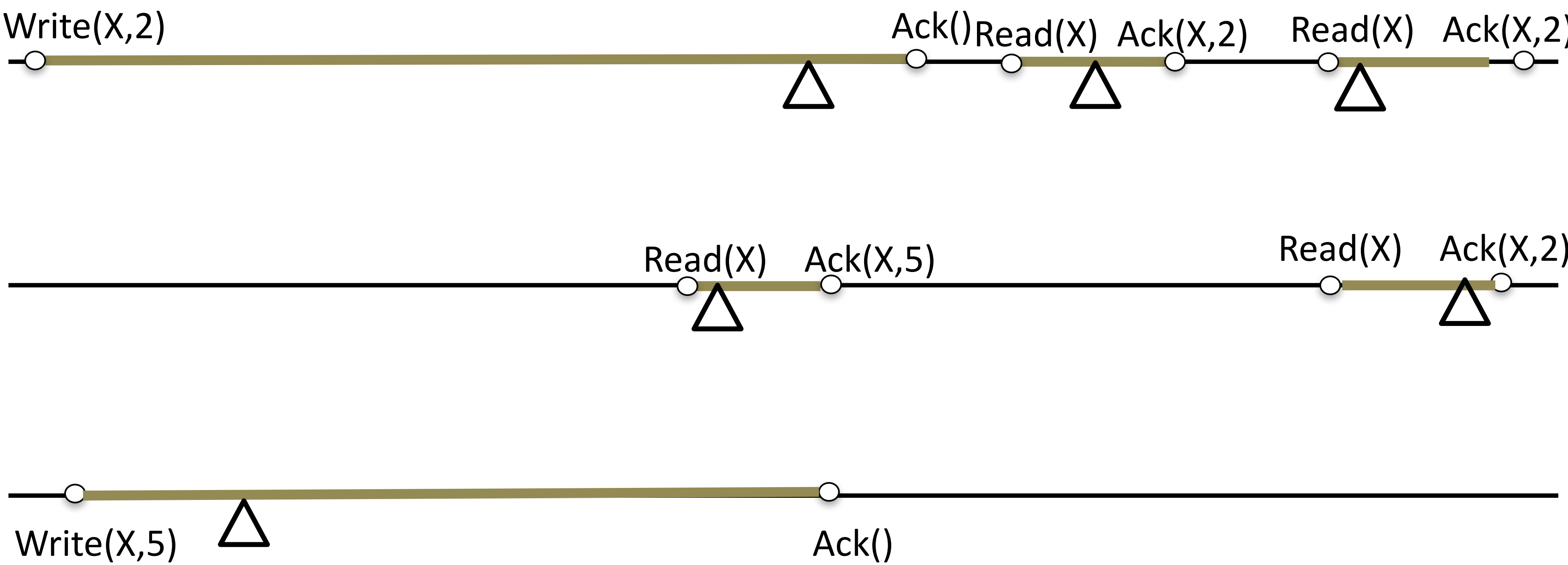


Figure 8: Alternate linearization points (compare with Figure 7)

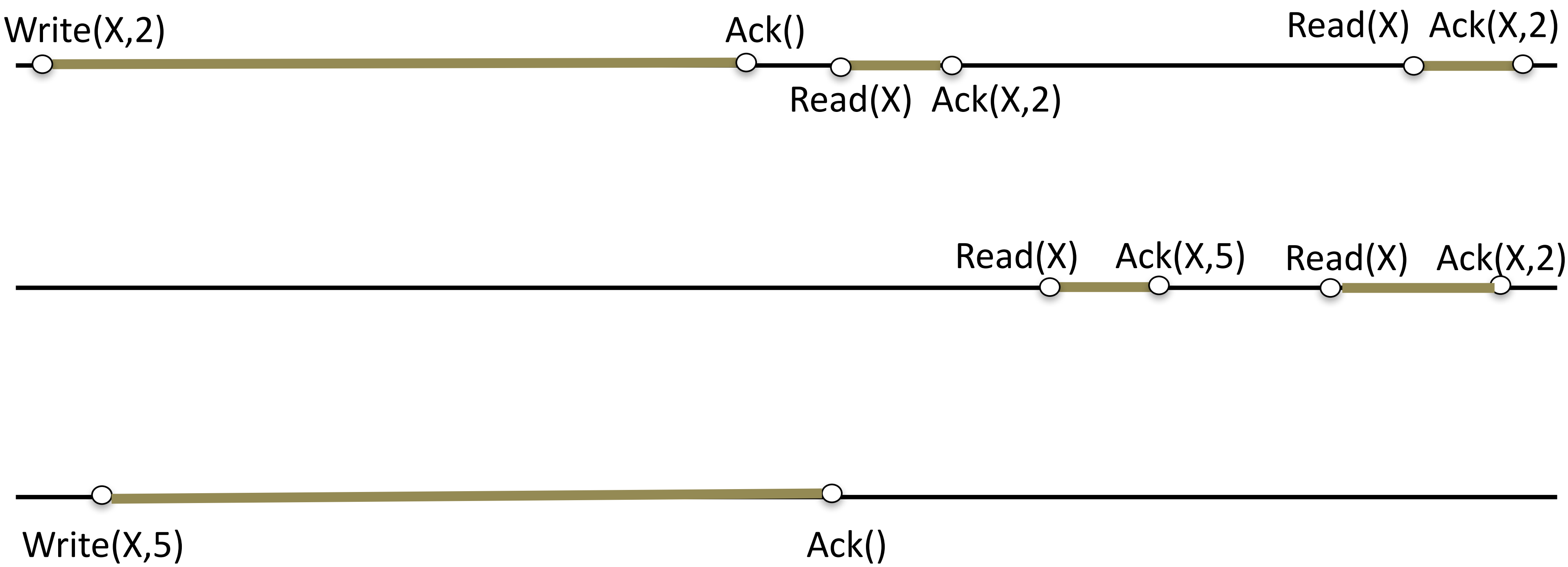


Figure 5 ... can we find suitable linearization points ?

Linearizability

Intuitively …

Each operation in a linearizable execution appears to "take effect" instantaneously at some time between its invocation and its response

… this preserves per-process
and real-time order both

This point of time is called its *linearization point*

Sequential Consistency

An execution is sequentially consistent if there exists a permutation that is

valid, and

per-process order-preserving

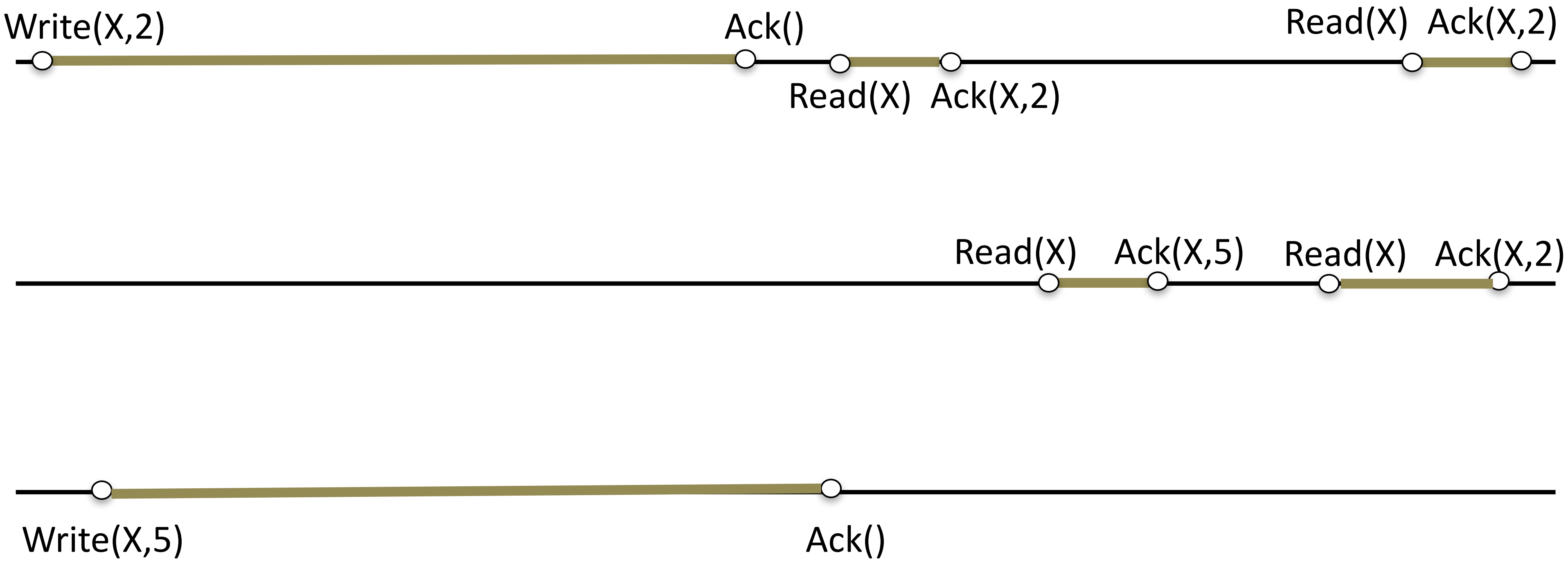
Sequential Consistency

An execution is sequentially consistent if there exists a permutation that is

valid, and

per-process order-preserving

An execution that is linearizable is also sequentially consistent



$$Write_3(X,5),\ Read_2(X,5),\ Write_1(X,2),\ Read_2(X,2),\ Read_1(X,2),\ Read_1(X,2)$$

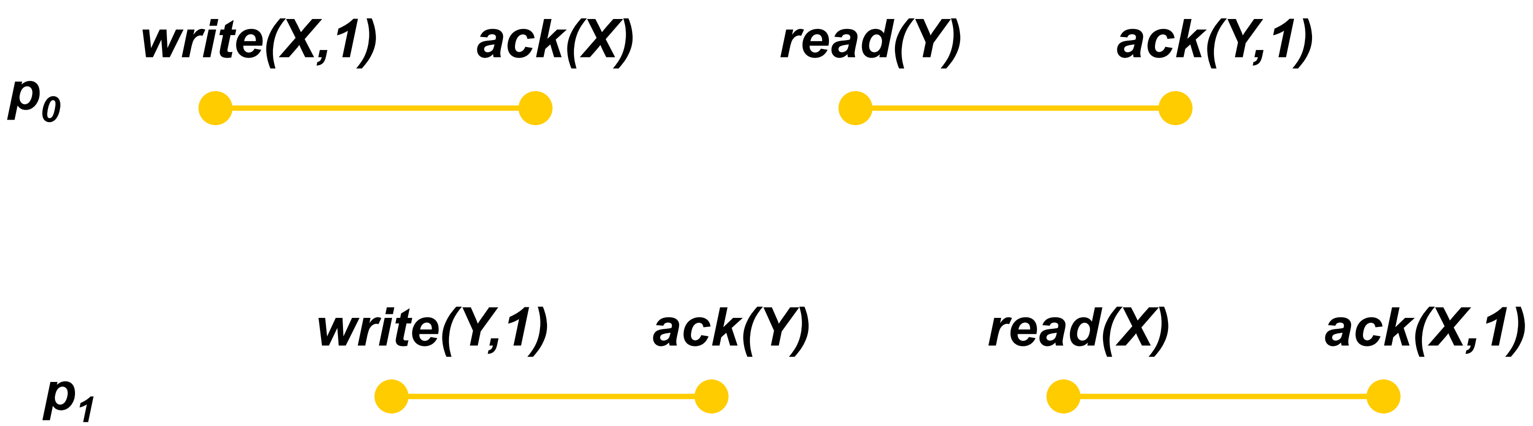
Figure 5 … not linearizable,
but satisfies sequential consistency

Sequential Consistency

Sequential Consistency

Example 1

Suppose there are two shared variables, *X* and *Y*, both initially 0

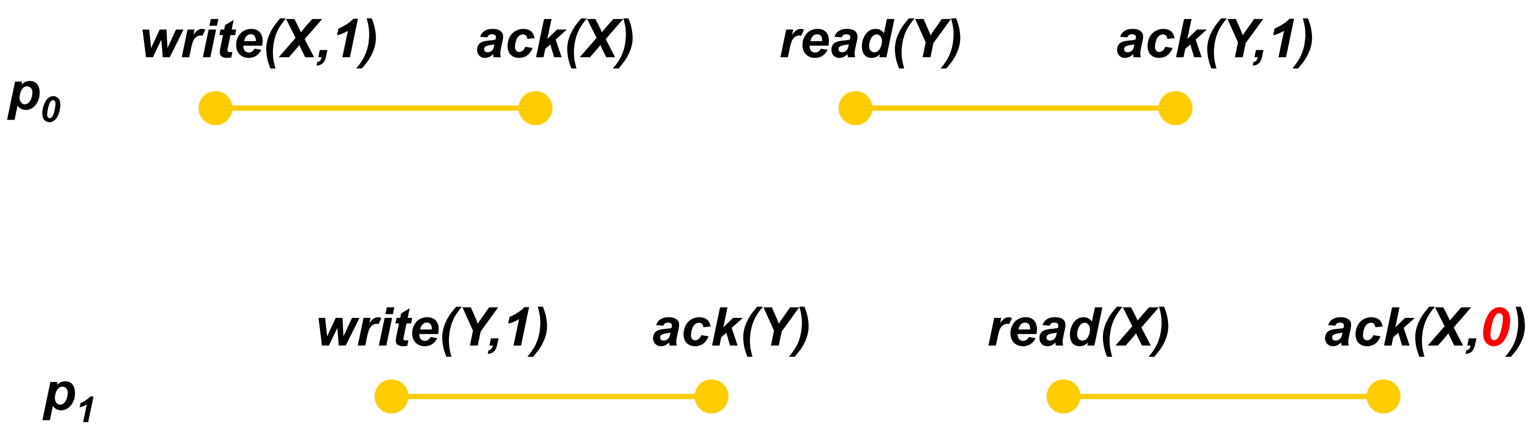


linearizability?
sequential consistency?

Example from Prof. Welch's slides

Example 2

Suppose there are two shared variables, *X* and *Y*, both initially 0



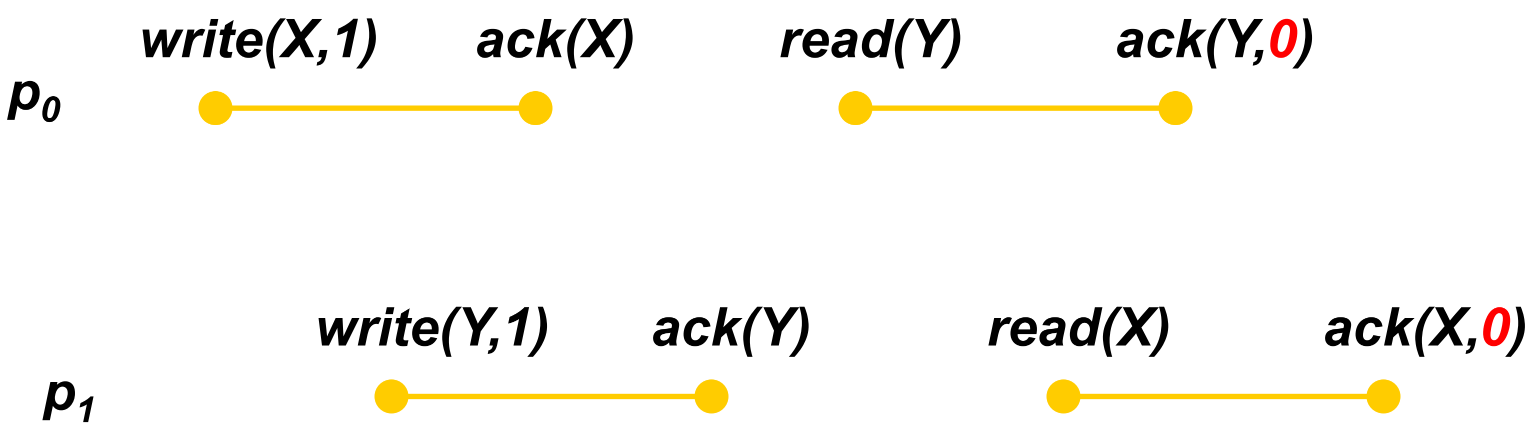
linearizability?

sequential consistency?

Example from Prof. Welch's slides

Example 3

Suppose there are two shared variables, *X* and *Y*, both initially 0



linearizability?

sequential consistency?

Example from Prof. Welch's slides

Implementation

- Algorithm 2 achieves sequential consistency
  - That is, all executions that result when using algorithm 2 satisfy sequential consistency

- Algorithm 3 achieves linearizability

Happened-Before for Shared Memory

Program Order

- Operations O_1 and O_2 at the same process p

$O_1 < O_2$: if O_1 completes at p sometime before O_2 is invoked

Reads-From

- Write operation W
- Read operation R
- May be at same or different processes

R reads from W   W --> R

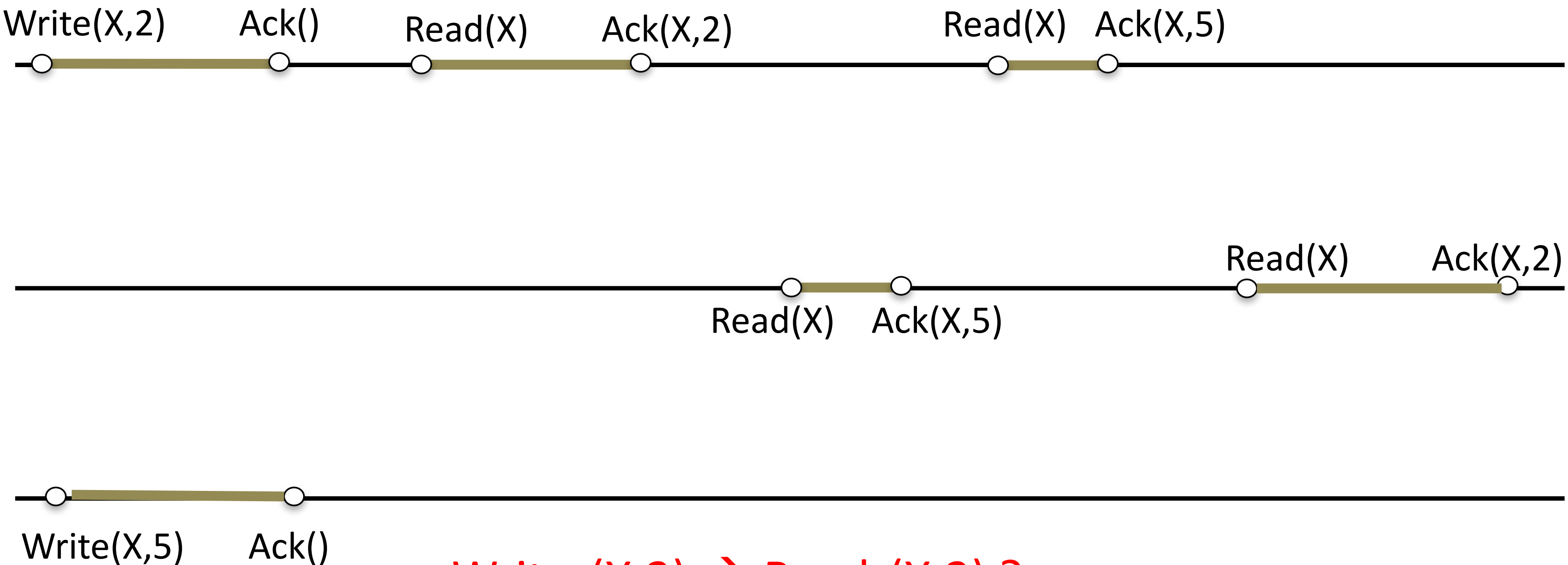
if R returns value written by W

(some ambiguity if same value written by multiple writes … assume unique values written)

Happened-Before

- (Program order) If $O_1 < O_2$ then $O_1 \rightarrow O_2$
- (Reads-from) If O_1 --> O_2 then $O_1 \rightarrow O_2$
- (Transitivity) If $O_1 \rightarrow O_2$ and $O_2 \rightarrow O_3$ the $O_1 \rightarrow O_3$

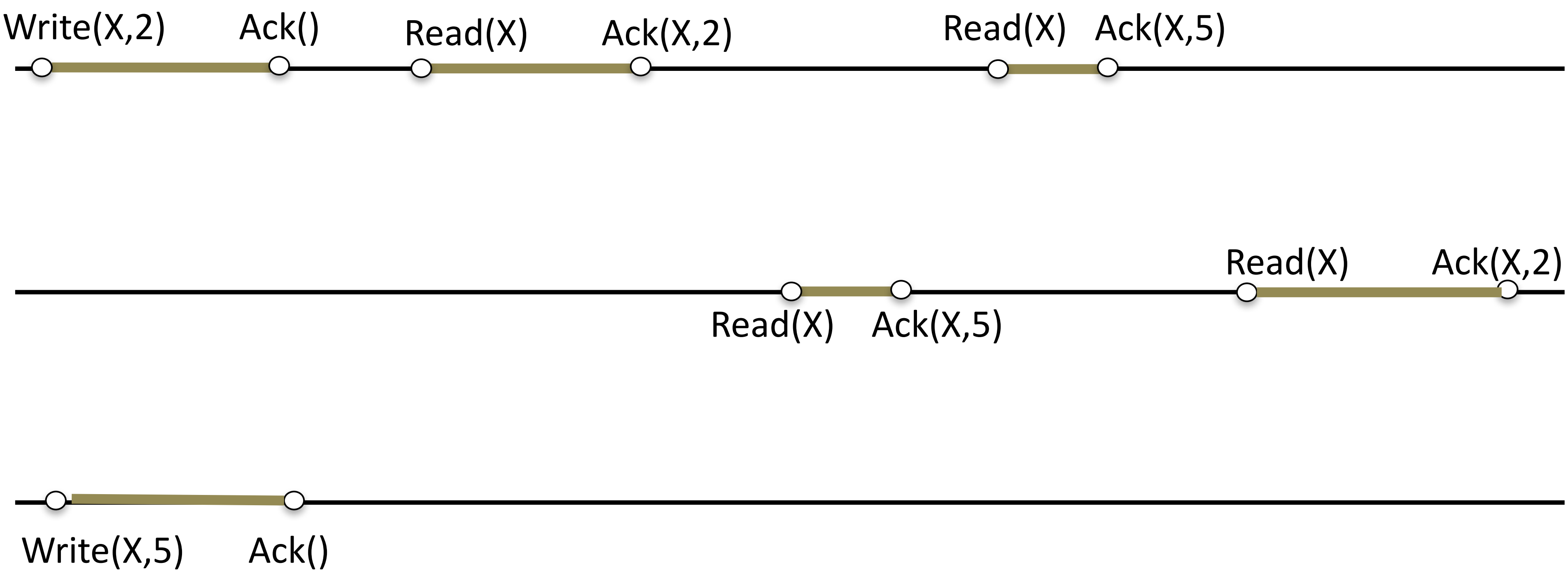
Figure 4



$\text{Write}_1(X,2) \rightarrow \text{Read}_1(X,2)$?

$\text{Write}_1(X,2) \rightarrow \text{Read}_1(X,5)$?

Figure 4

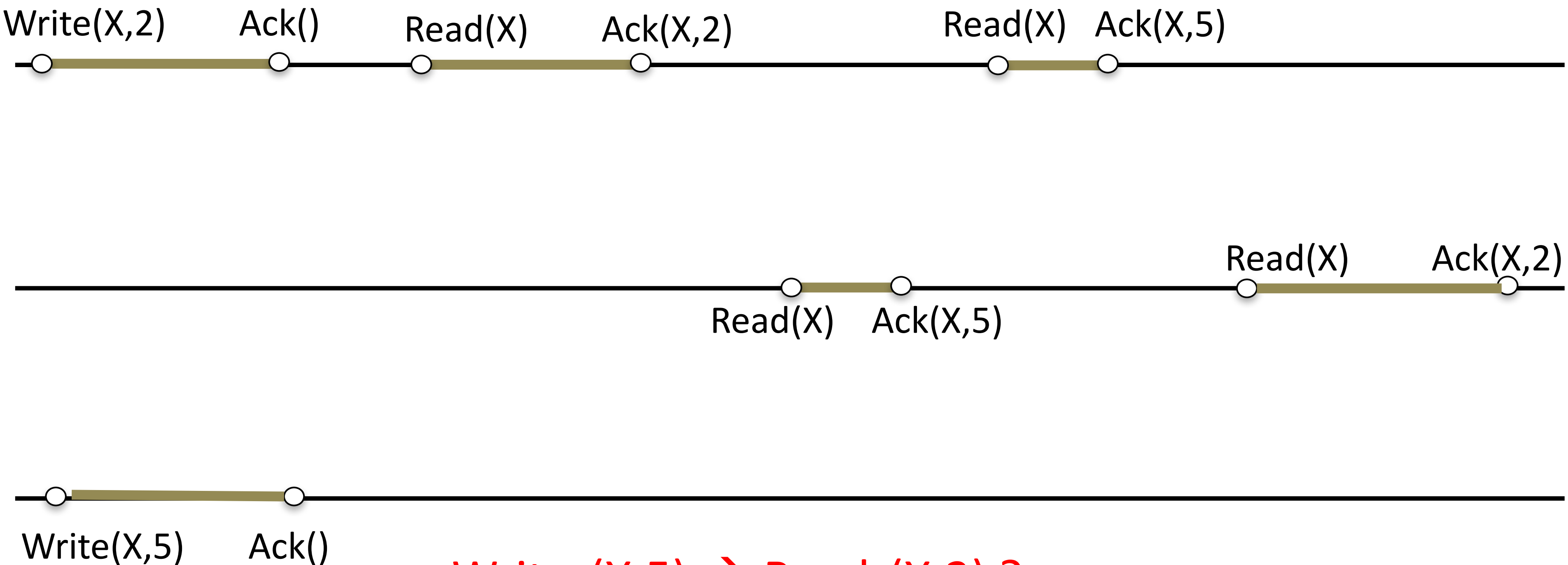


$Read_2(X,2) \rightarrow Write_1(X,2)$?

$Write_1(X,2) \rightarrow Read_2(X,2)$?

$Write_3(X,5) \rightarrow Read_2(X,5)$?

Figure 4



Write$_3$(X,5) $\rightarrow$ Read$_1$(X,2) ?

Read$_2$(X,2) $\rightarrow$ Read$_1$(X,5) ?

Write$_3$(X,5) $\rightarrow$ Read$_2$(X,2) ?