Liveness of Transactional Memory

Victor Bushkov

Distributed Programming Laboratory

Part I

Defining transactional memory liveness

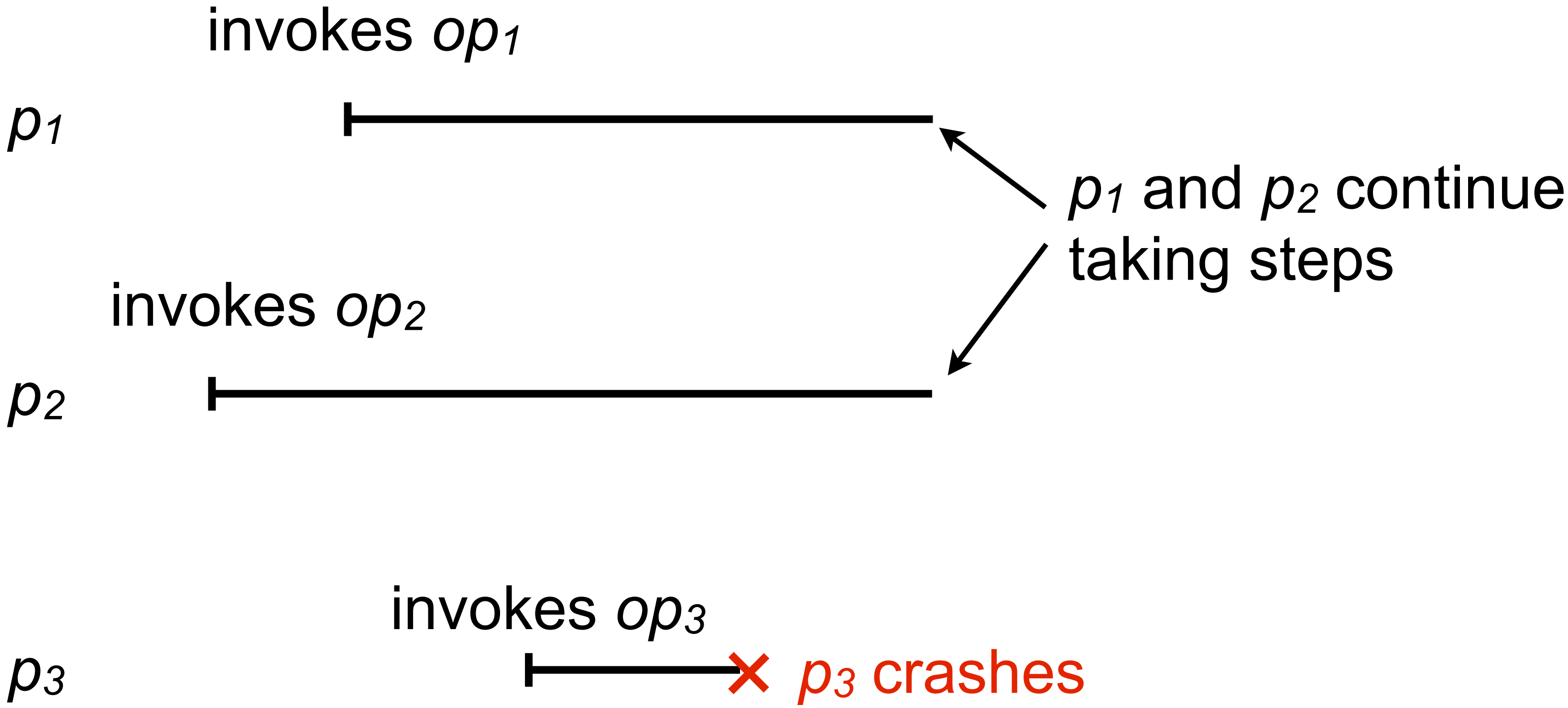
Properties covered so far

- wait-freedom
- lock-freedom
- obstruction-freedom

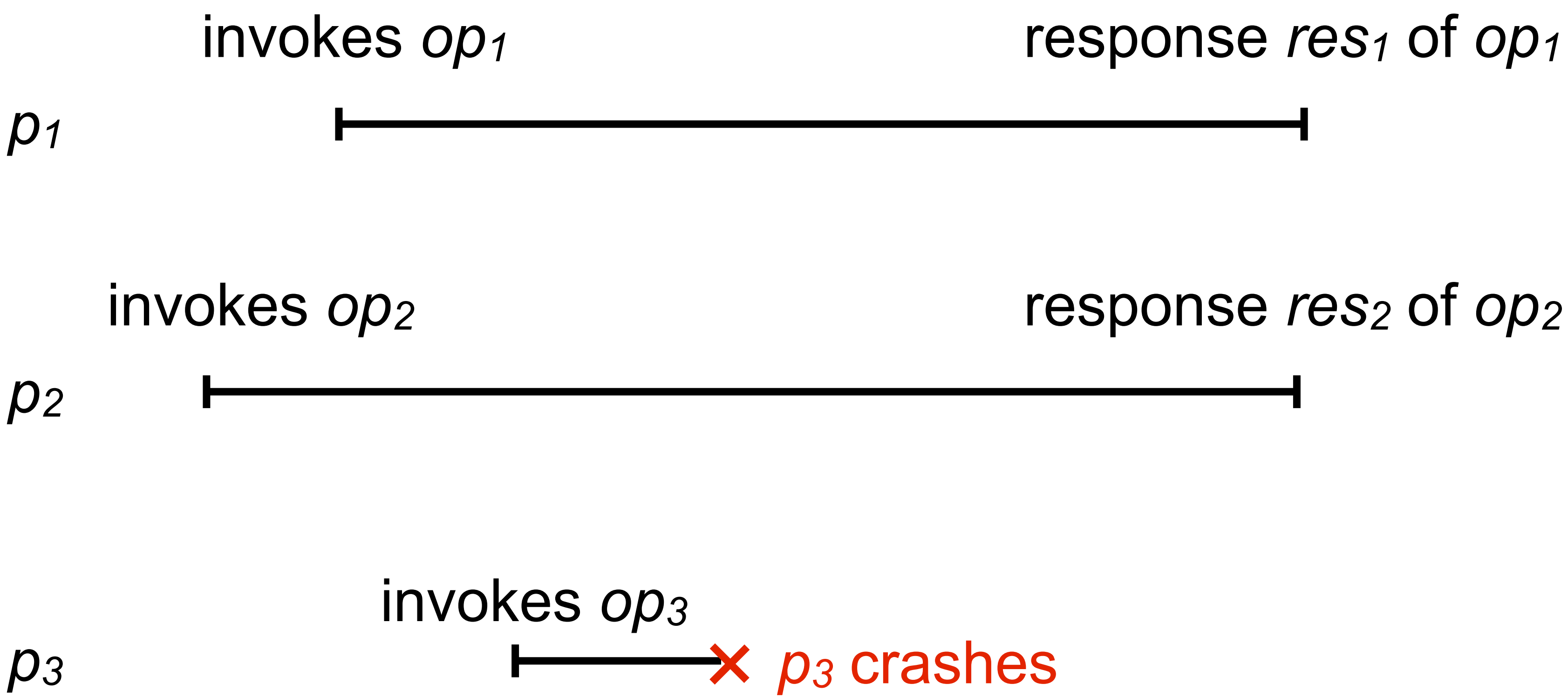
Wait-freedom

Every operation by *every* non-crashed process eventually returns a response

Wait-freedom: example



Wait-freedom: example

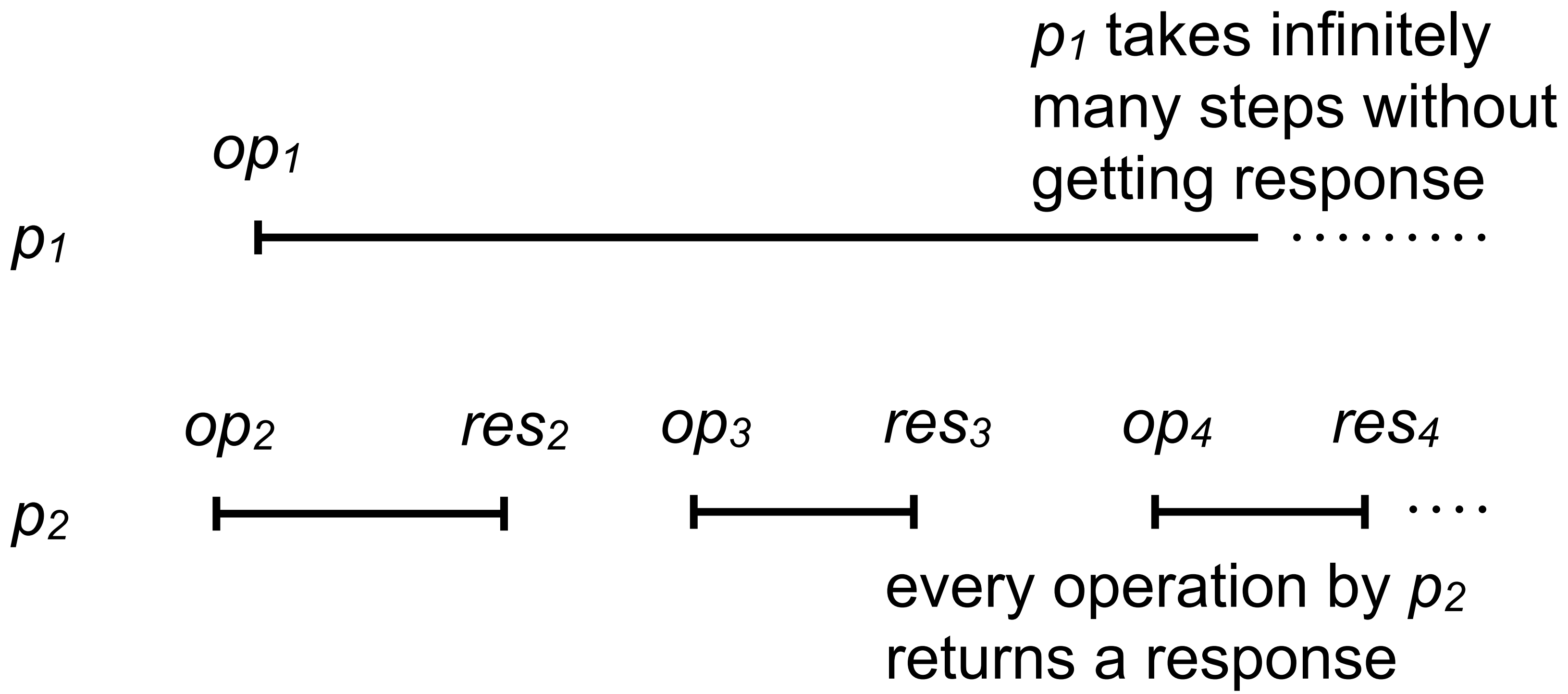


Lock-freedom

Every operation by *some* non-crashed process eventually returns a response

Lock-freedom: example

- execution is not wait-free
- but it is lock-free

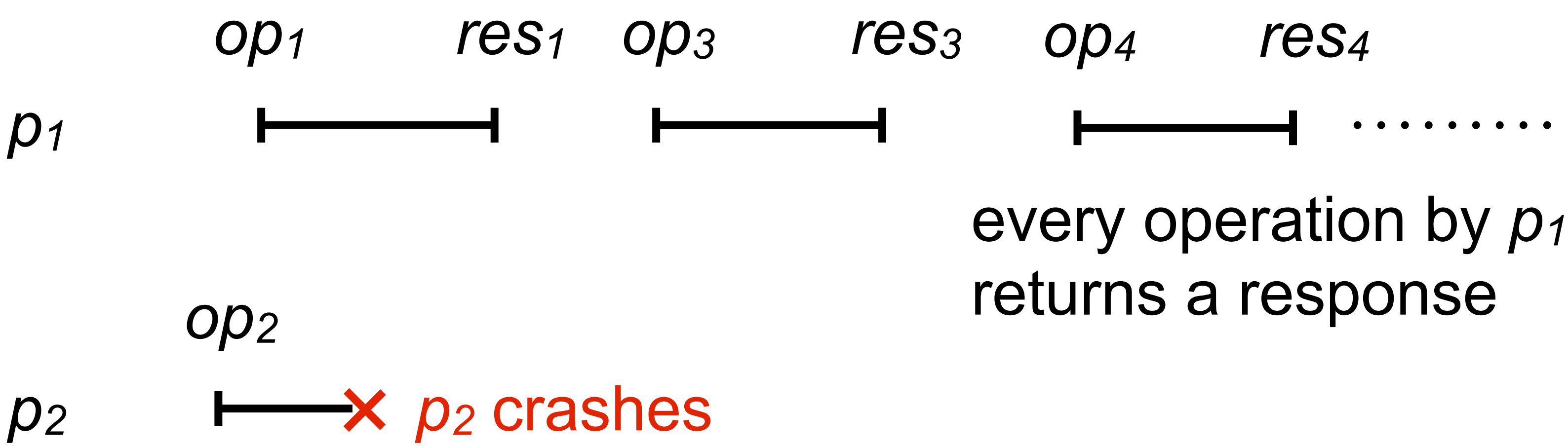


Obstruction-freedom

If a process *p* becomes the only process taking steps, then every operation by *p* eventually returns a response

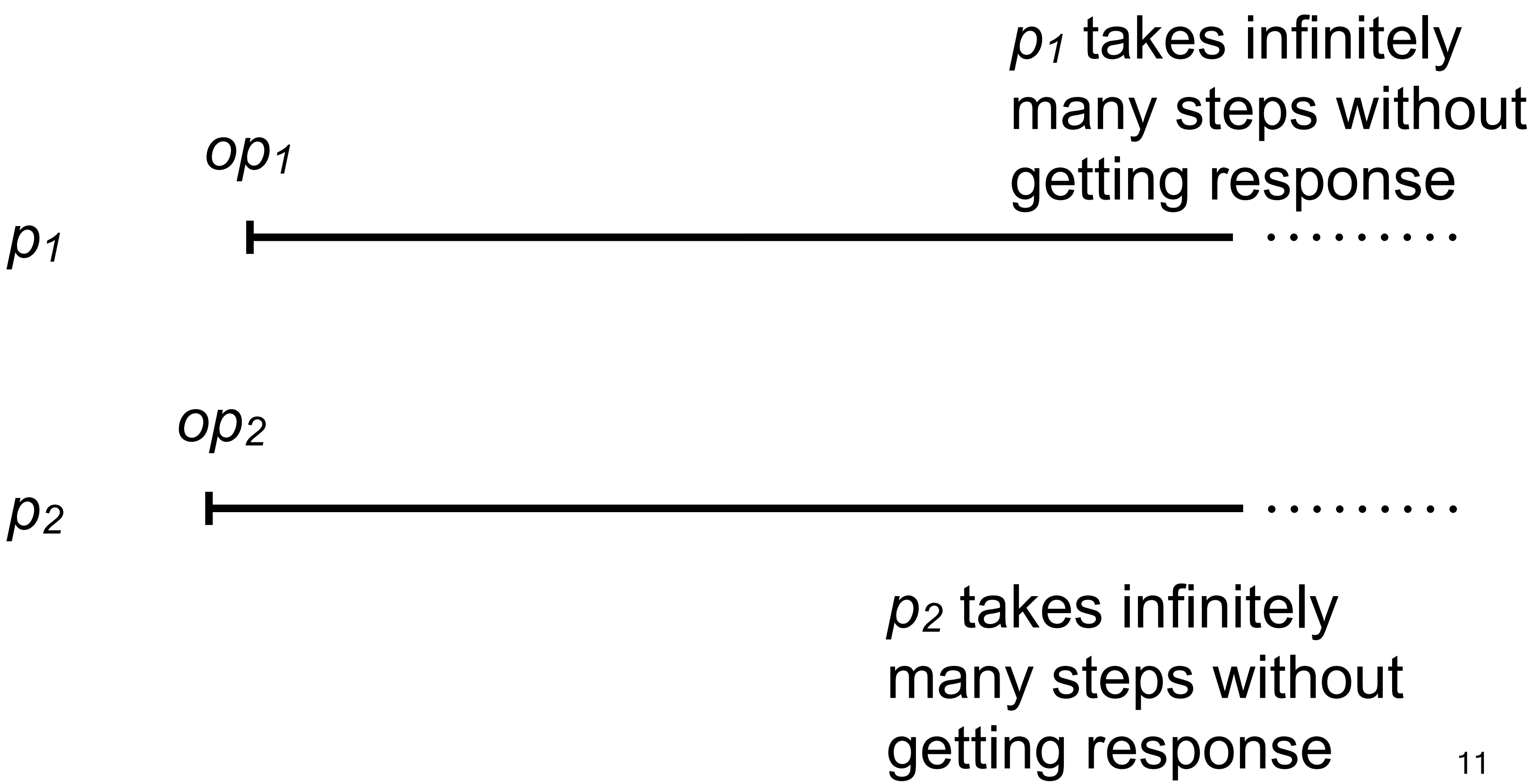
Obstruction-freedom: example

- execution is lock-free
- and it is obstruction-free



Obstruction-freedom: example

- execution is not lock-free
- but it is obstruction-free

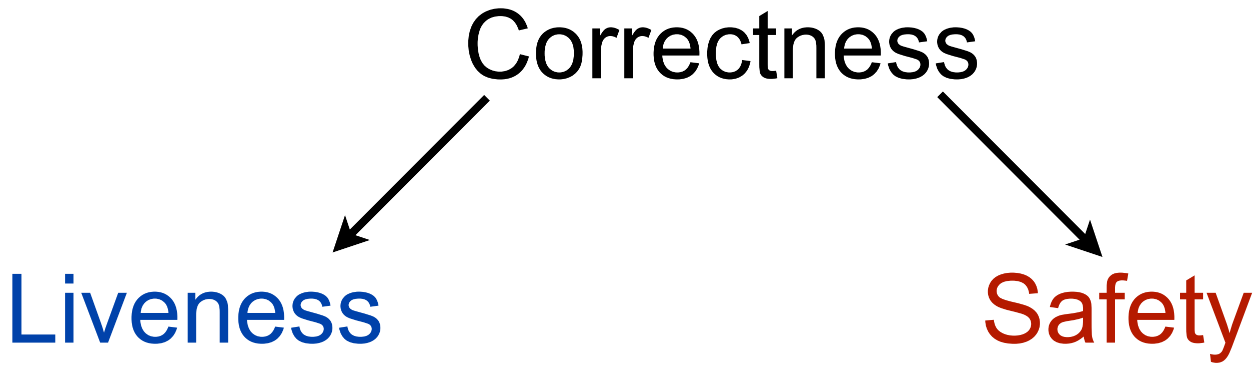


What is common between these three properties?

What is common between these three properties?

- state that some good event must *eventually* happen
- i.e. they are liveness properties

Liveness vs Safety



Liveness vs Safety

Correctness

Liveness

Safety

- wait-freedom (termination)
- lock-freedom
- obstruction-freedom

Liveness vs Safety

Correctness

Liveness

- wait-freedom (termination)
- lock-freedom
- obstruction-freedom

Safety

- validity and agreement
- regularity of registers
- atomicity (linearizability)
- opacity

Liveness vs Safety

Liveness: some good events should *eventually* happen

Safety: some bad events should *never* happen

Liveness vs Safety

Liveness: some good events should *eventually* happen

Safety: some bad events should *never* happen

- violated in finite execution

Liveness vs Safety

Liveness: some good events should *eventually* happen

- cannot be violated in a finite execution

Safety: some bad events should *never* happen

- violated in finite execution

Liveness of shared objects

- In shared objects good events are responses

Liveness of shared objects

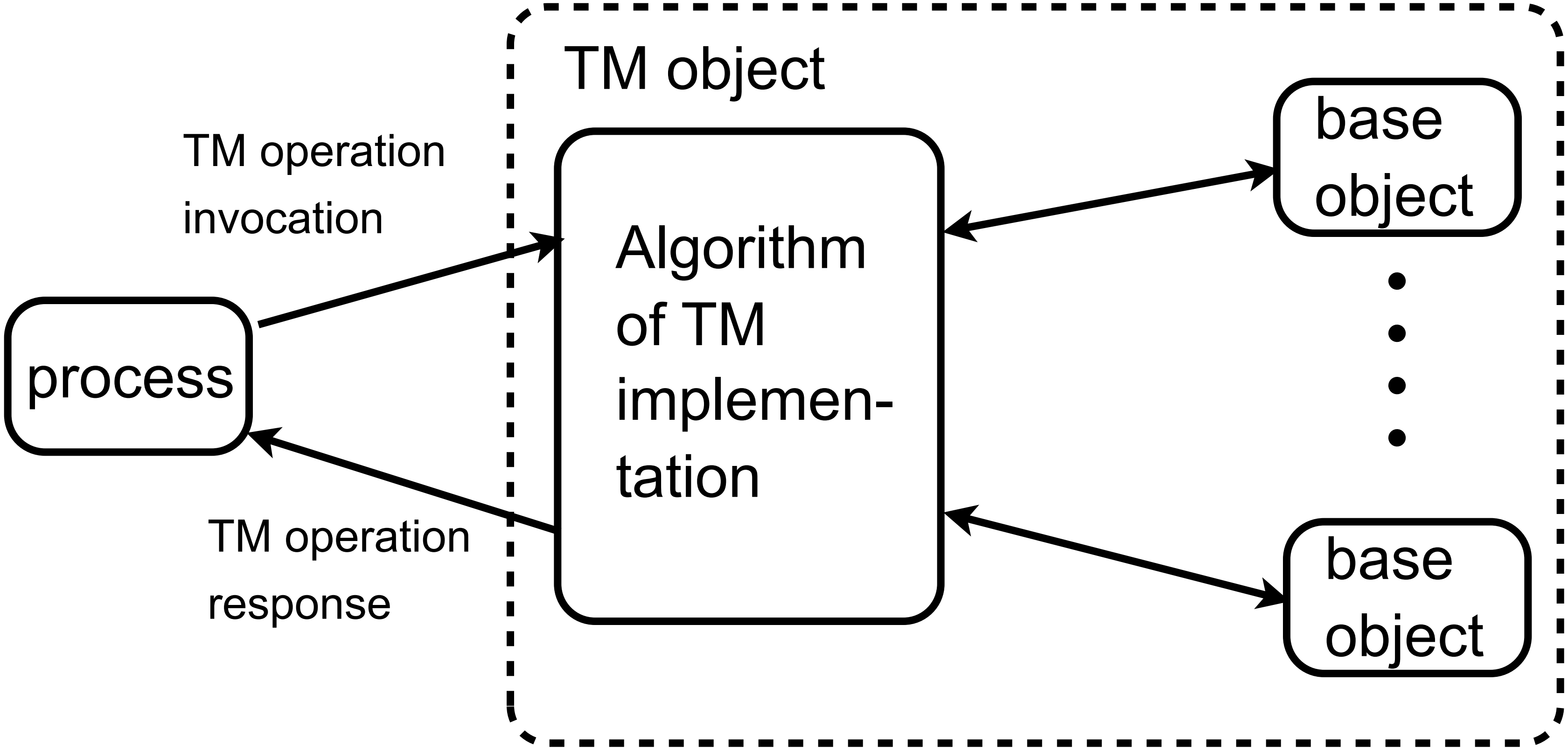
- In shared objects good events are responses
- In case of wait-freedom, lock-freedom, and obstruction-freedom *any* response is a good event i.e.:

Liveness of shared objects

- In shared objects good events are responses
- In case of wait-freedom, lock-freedom, and obstruction-freedom *any* response is a good event i.e.:

e.g. in case of wait-freedom we do not care if we get res_1 or some other response res'_1

Transactional memory (TM) as a shared objects



Transactional memory (TM) as a shared objects

examples of some TM operations

- *x*.*read*() - returns value of data item *x*
- *x*.*write*(*v*) - writes value *v* to data item *x*
- *commit*() - commits current transaction
- *begin_tr*() - starts a transaction

Transactional memory (TM) as a shared objects

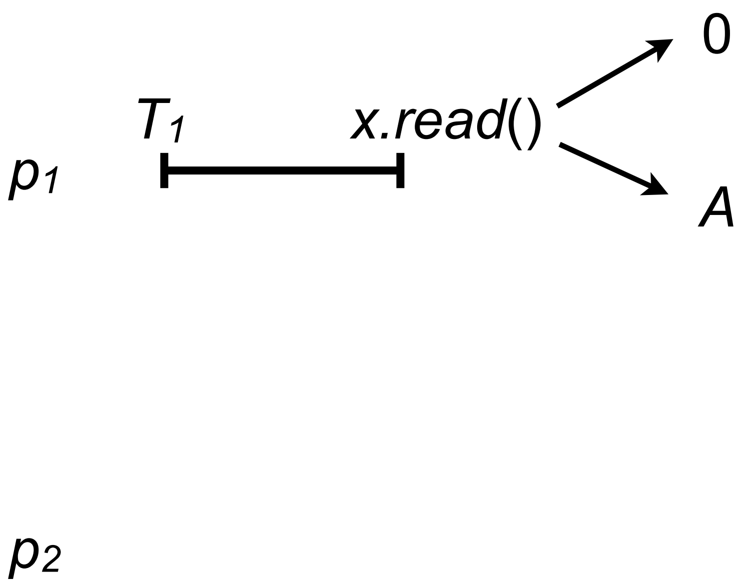
examples of some TM operations

- *x*.*read*() - returns value of data item *x*
- *x*.*write*(*v*) - writes value *v* to data item *x*
- *commit*() - commits current transaction
- *begin_tr*() - starts a transaction

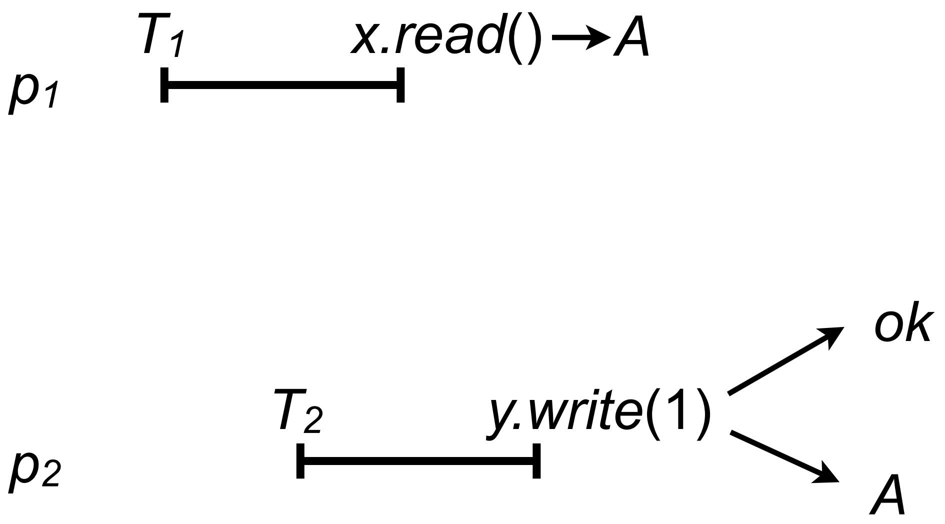
- every TM operation can return abort event *A* which aborts current transaction

Is wait-freedom enough in TM context?

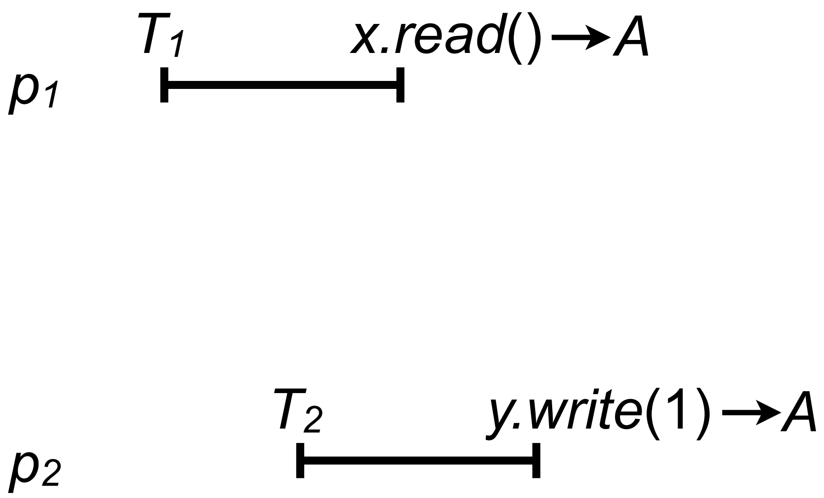
Is wait-freedom enough in TM context?



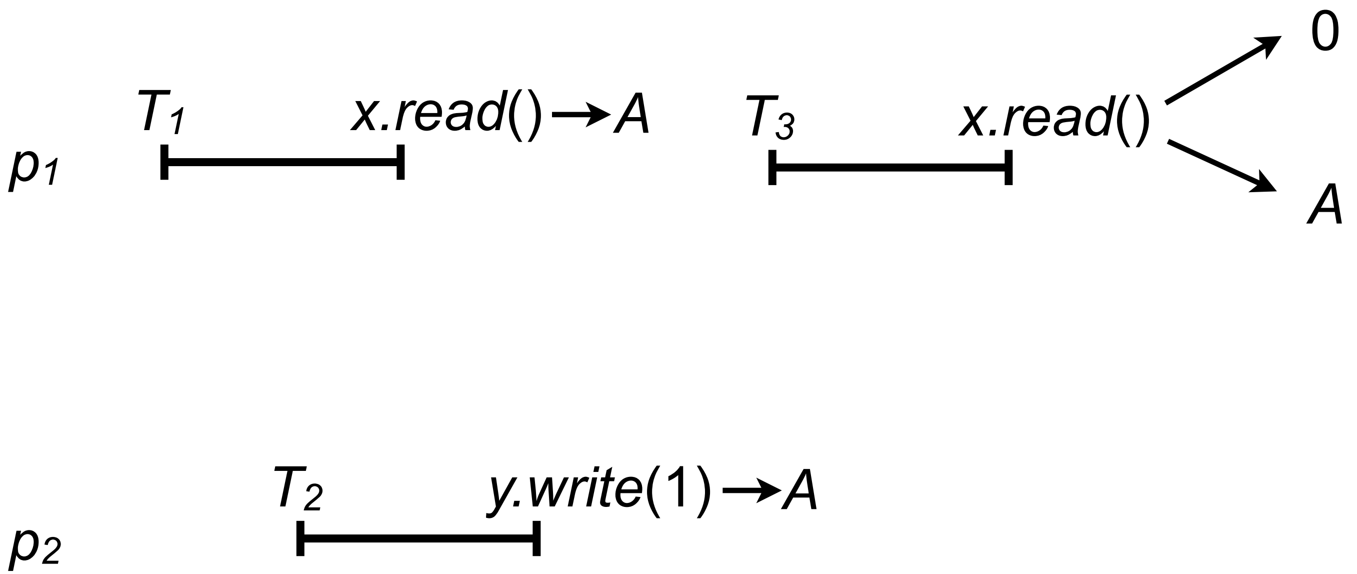
Is wait-freedom enough in TM context?



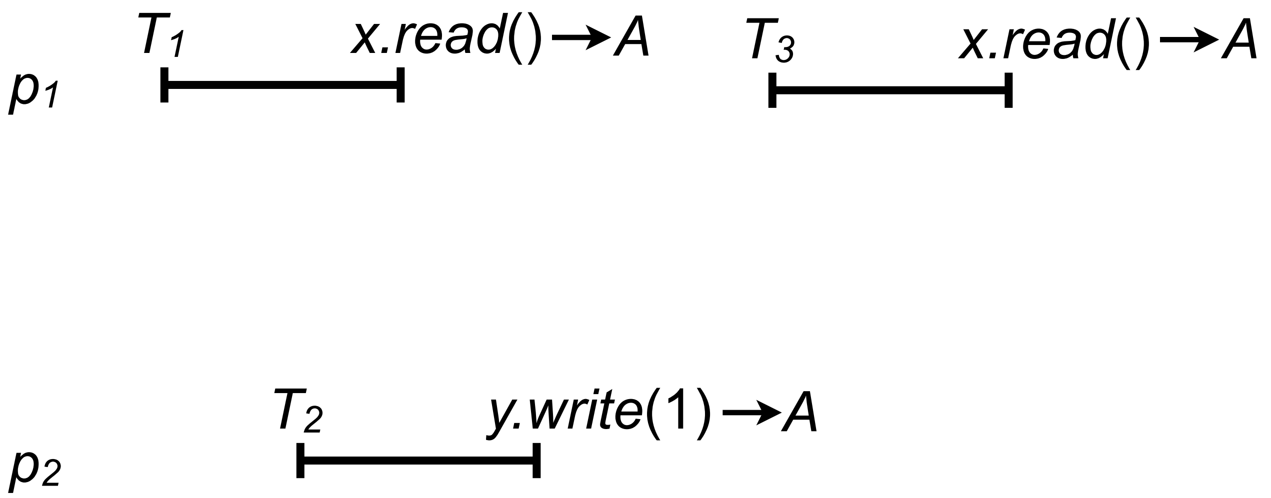
Is wait-freedom enough in TM context?



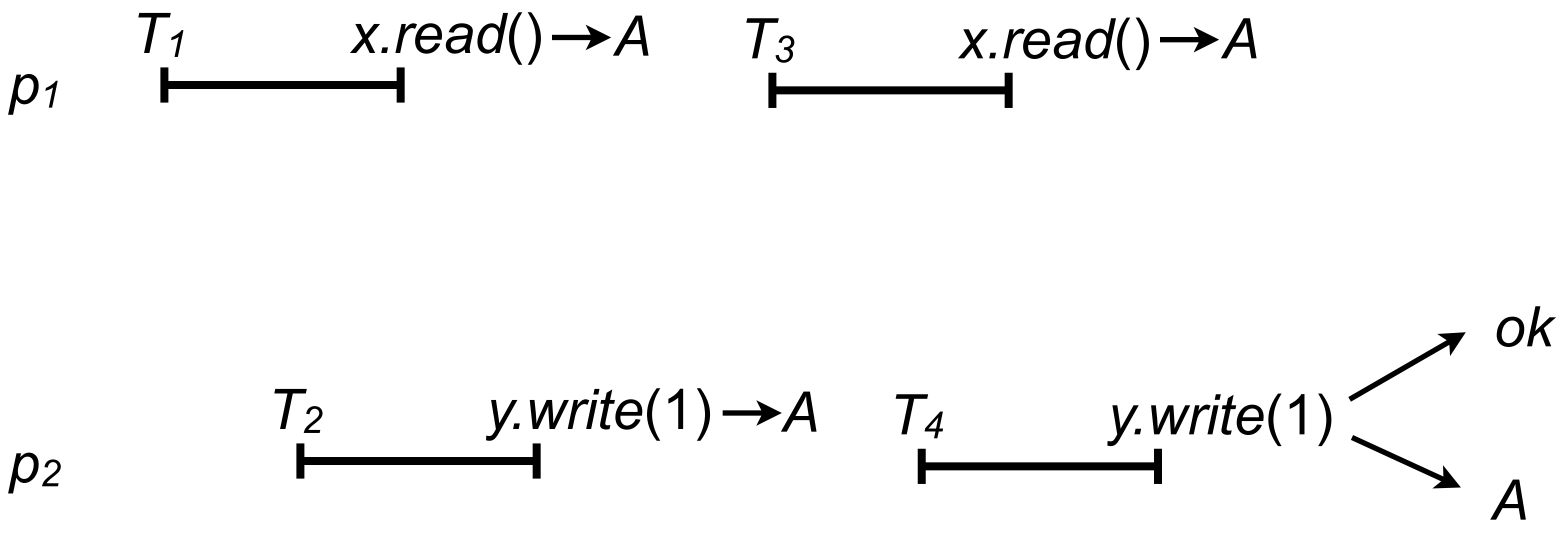
Is wait-freedom enough in TM context?



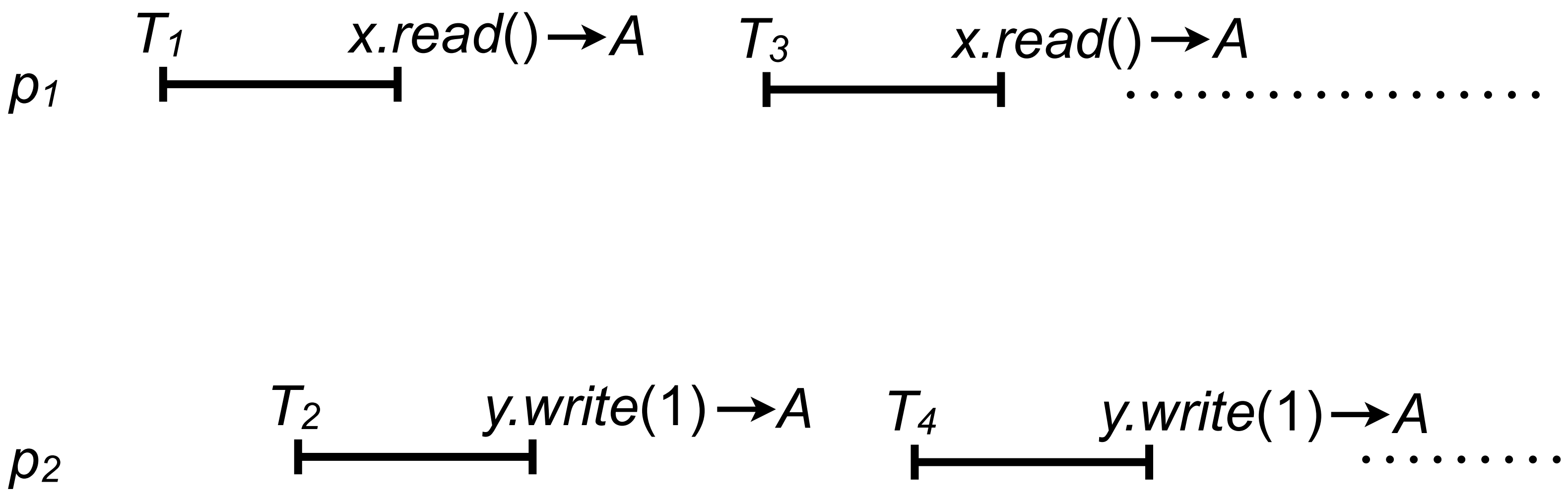
Is wait-freedom enough in TM context?



Is wait-freedom enough in TM context?



Is wait-freedom enough in TM context?



Meaningful progress

- wait-freedom is trivially ensured by aborting every TM operation

Meaningful progress

- wait-freedom is trivially ensured by aborting every TM operation
- operation termination is not enough

Meaningful progress

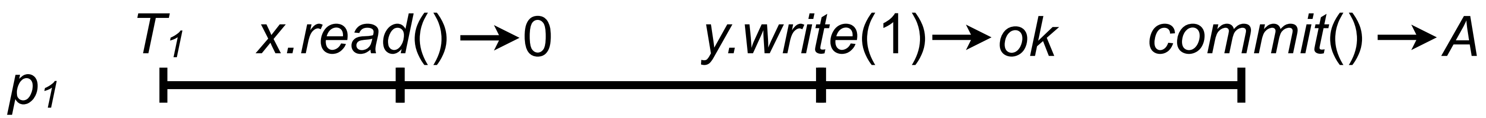
- wait-freedom is trivially ensured by aborting every TM operation
- operation termination is not enough
- operations need to receive meaningful responses

What about the following property?

- Every TM operation by every non-crashed process eventually returns a response which is not an abort event

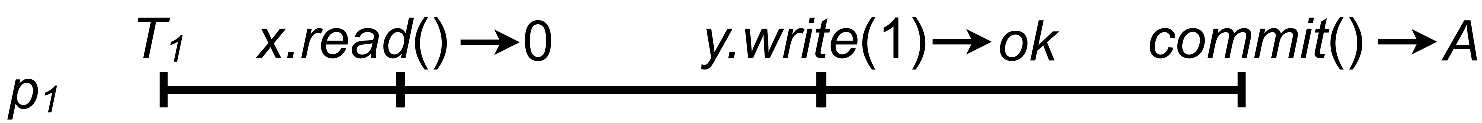
What about the following property?

- Every TM operation by every non-crashed process eventually returns a response which is not an abort event
- It can be violated in a finite execution → it is not liveness



What about the following property?

- Every TM operation by every non-crashed process eventually returns a response which is not an abort event
- It can be violated in a finite execution → it is not liveness
- TM loses its meaning without ability to abort (TM becomes equivalent to universal construction)



Meaningful progress

TM liveness property should

- allow every transaction to be aborted, and

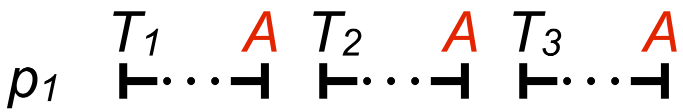
Meaningful progress

TM liveness property should

- allow every transaction to be aborted, and
- require processes to eventually commit some transaction (make progress)

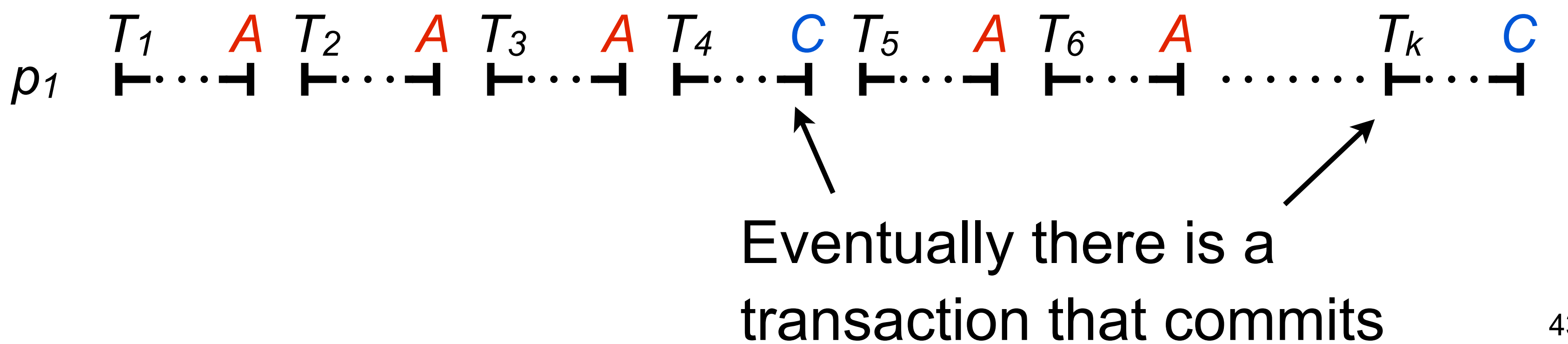
What does eventually committing some transactions mean?

- a process might have some of its transactions aborted



What does eventually committing some transactions mean?

- a process might have some of its transactions aborted
- but for any point in time of the execution eventually there is a transaction that commits



Can we require eventual commitment of *any* process?

```
begin_tr()
    while(value = i) do {
        value := x.read( );
        x.write(value + 1);
        i := i+1;
    }
commit()
```

Initially:

value, *i* = -1

x = 0

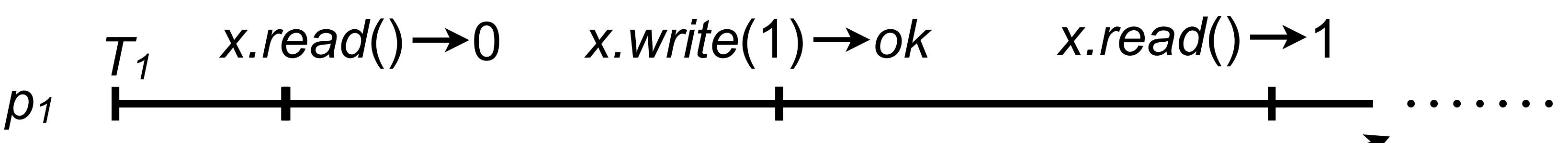
Can we require eventual commitment of *any* process?

```
begin_tr()
   while(value = i) do {
      value := x.read( );
      x.write(value + 1);
      i := i+1;
   }
commit()
```

Initially:

value, *i* = -1

x = 0



p_1 repeatedly reads and writes *x* without ever invoking a commit request

Correct processes

We cannot require progress of processes which are not correct in a given infinite execution α:

- processes which crash in α, or

Correct processes

We cannot require progress of processes which are not correct in a given infinite execution α:

- processes which crash in α, or
- processes which execute a transaction which is not aborted and does not invoke a commit request in α

Correct processes

We cannot require progress of processes which are not correct in a given infinite execution α:

- processes which crash in α, or
- processes which execute a transaction which is not aborted and does not invoke a commit request in α

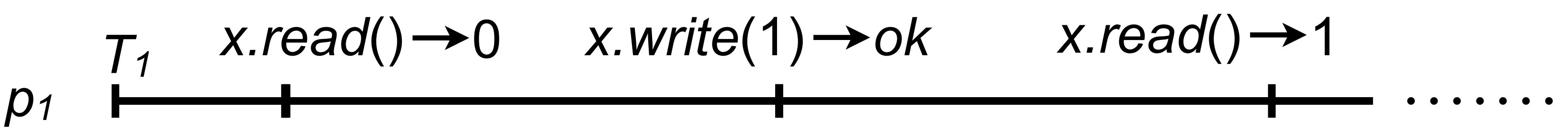
Correct processes

```
begin_tr()
    while(value = i) do {
        value := x.read( );
        x.write(value + 1);
        i := i+1;
    }
commit()
```

Initially:

value, *i* = -1

x = 0



p_1 is not correct in the given execution

Correct processes

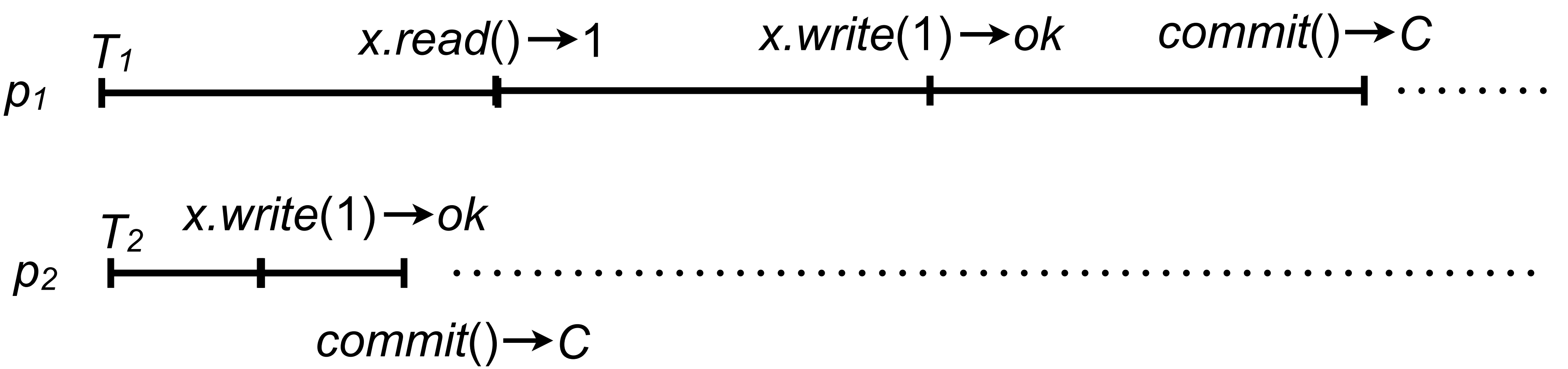
```
begin_tr()
    while(value = i) do {
        value := x.read( );
        x.write(value + 1);
        i := i+1;
    }
commit()
```

Initially:

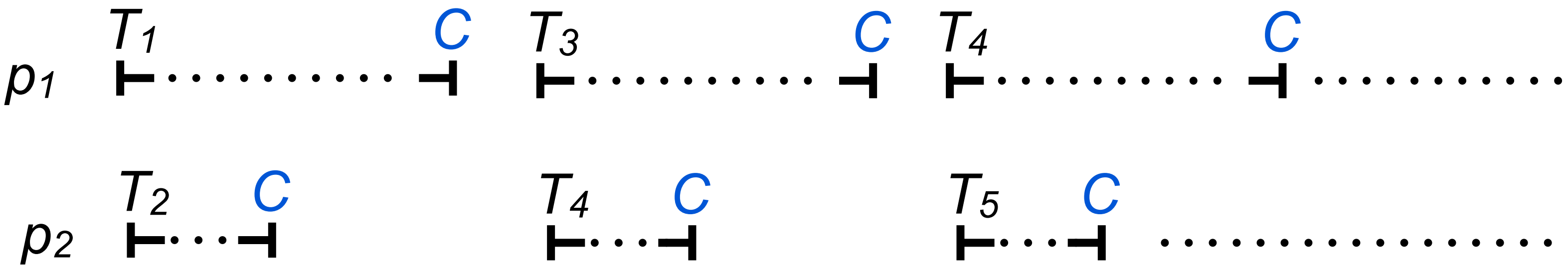
value, i = -1

x = 0

p_1 is correct in the given execution

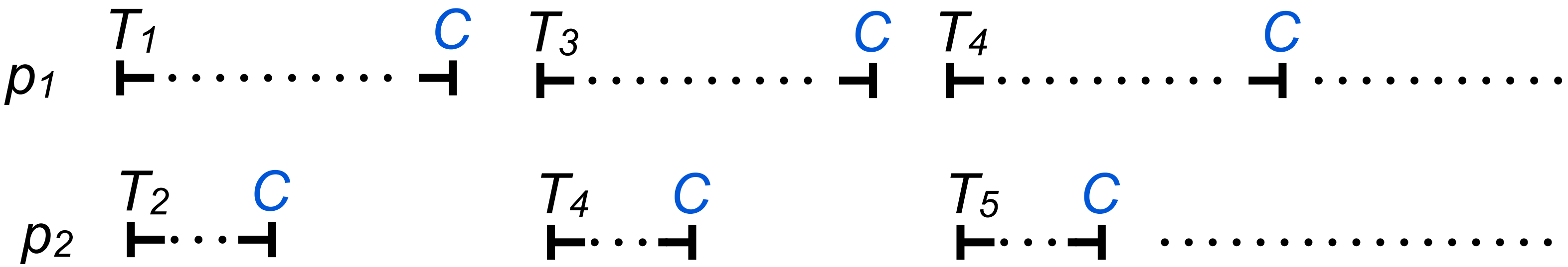


Correct processes



- p_1 is correct in the given execution

Correct processes



- p_1 is correct in the given execution
- the notion of a correct process depends on an execution

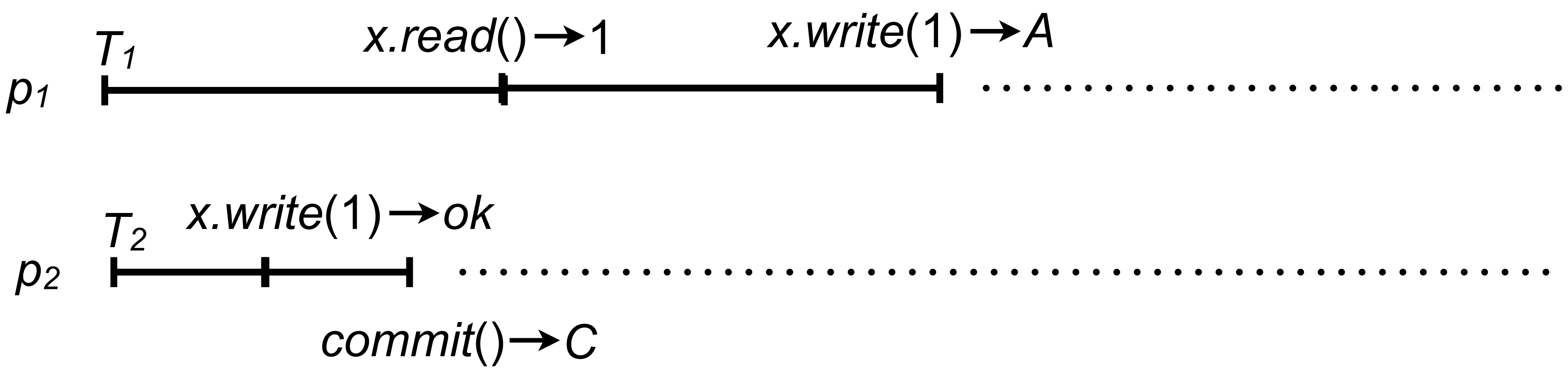
Correct processes

```
begin_tr()
   while(value = i) do {
      value := x.read( );
      x.write(value + 1);
      i := i+1;
   }
commit()
```

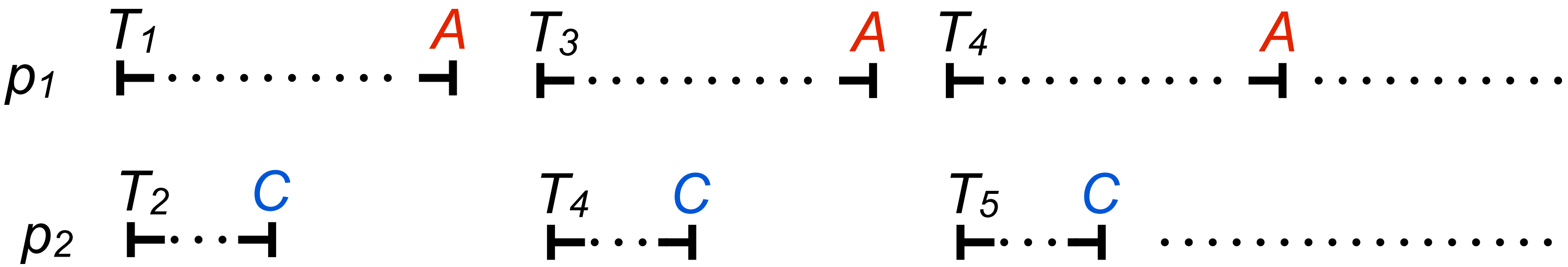
Initially:

value, *i* = -1

x = 0

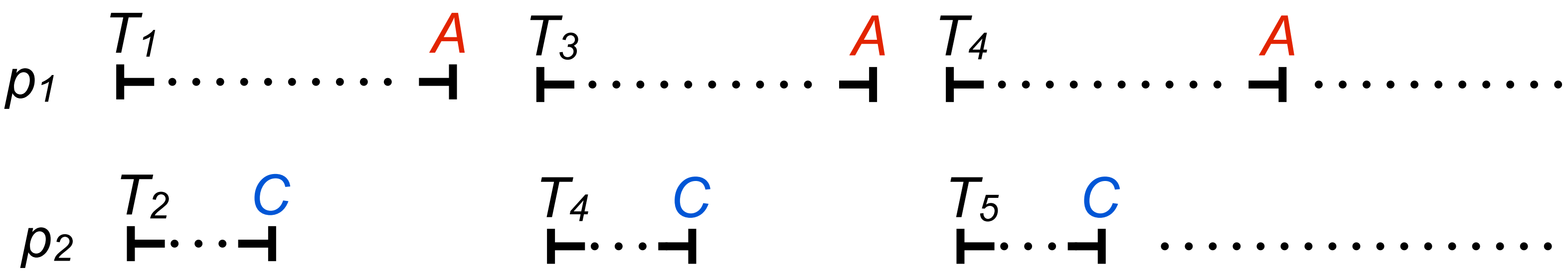


Correct processes



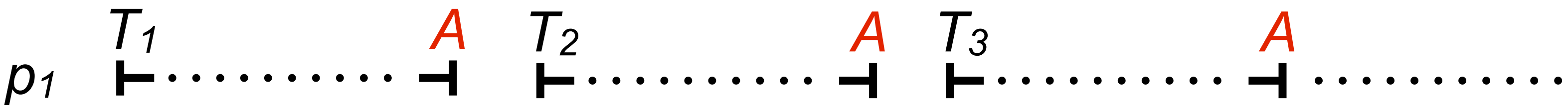
- p_1 is correct in the given execution

Correct processes



- p_1 is correct in the given execution
- a process which is never given possibility to invoke a commit request is still considered correct

Correct processes



- p_1 is correct in the given execution
- a process which is never given possibility to invoke a commit request is still considered correct

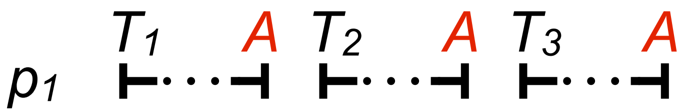
Making progress (in TM context)

A correct process *p* makes progress in an infinite execution α if infinitely many transaction of *p* commit in α

Making progress (in TM context)

A correct process p makes progress in an infinite execution α if infinitely many transaction of p commit in α

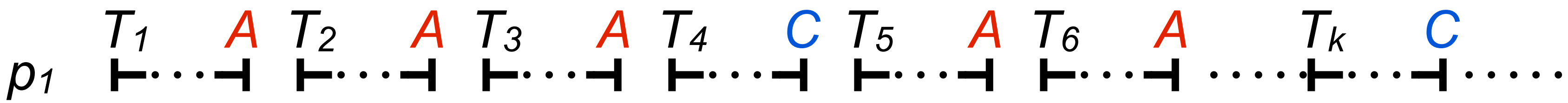
- a process might have some of its transactions aborted



Making progress (in TM context)

A correct process *p* makes progress in an infinite execution α if infinitely many transaction of *p* commit in α

- a process might have some of its transactions aborted
- but for any point in time of the execution eventually there is a transaction that does not abort (and consequently commits)

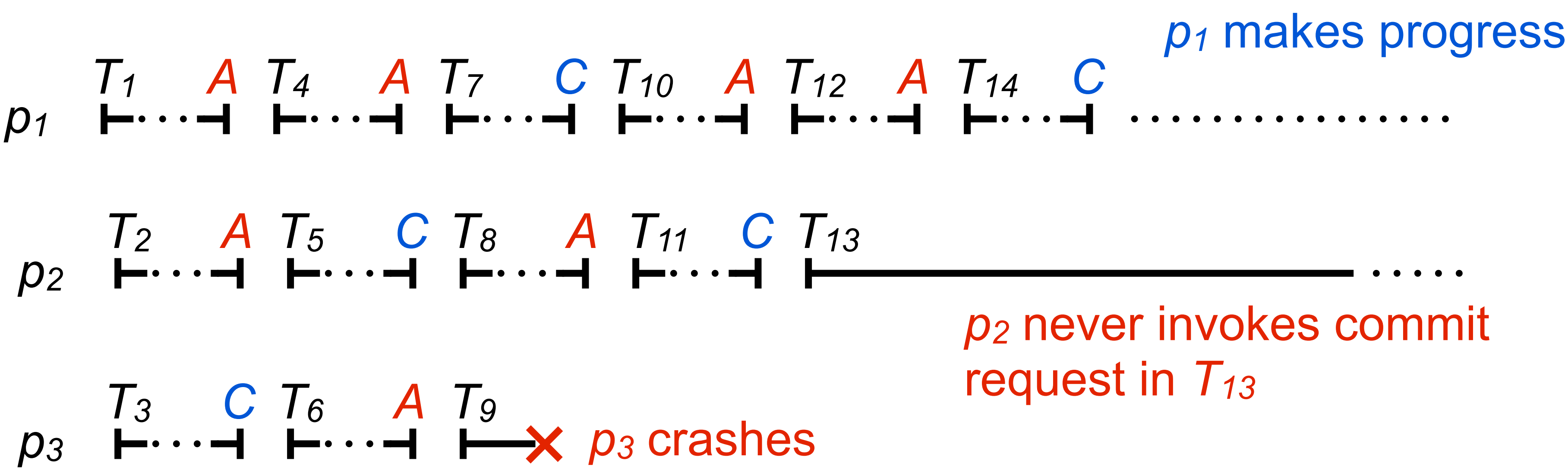


TM liveness

An infinite execution α is TM-wait-free if *every* correct process makes progress in α

TM liveness

An infinite execution α is TM-wait-free if *every* correct process makes progress in α

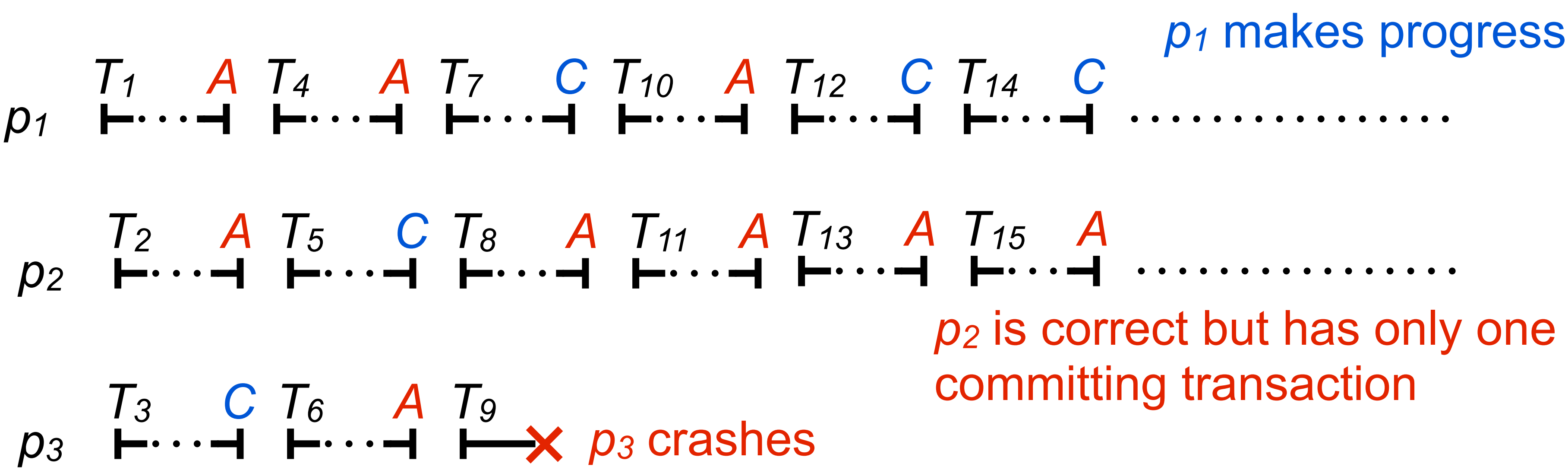


TM liveness

An infinite execution α is TM-lock-free if *some* correct process makes progress in α

TM liveness

An infinite execution α is TM-lock-free if *some* correct process makes progress in α

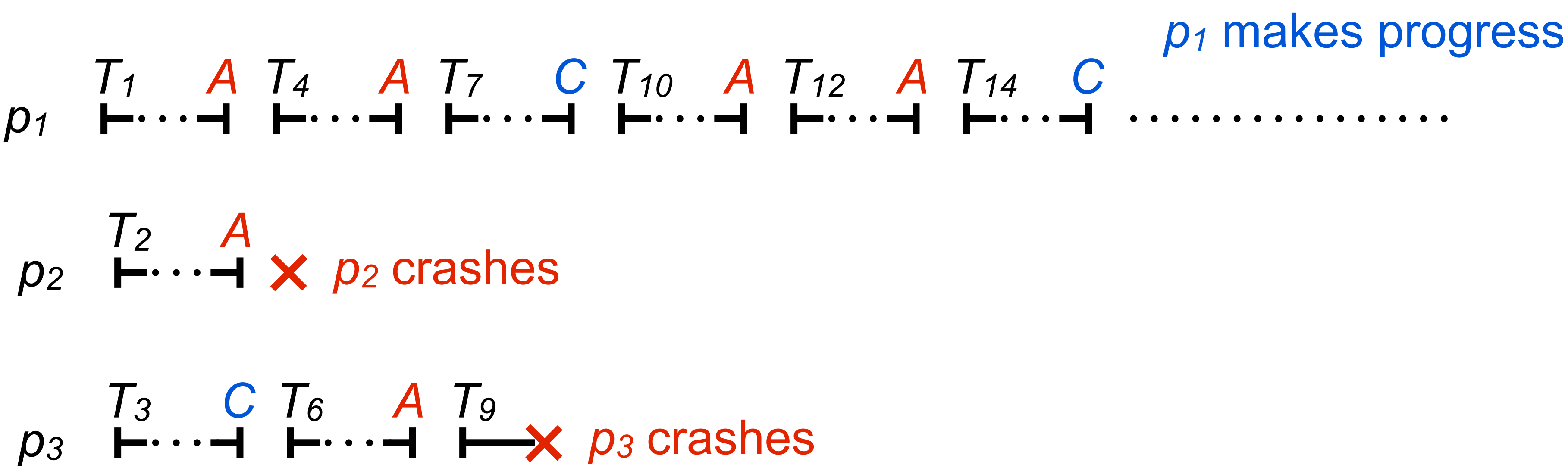


TM liveness

An infinite execution α is TM-obstruction-free if for every correct process *p* in α the following holds: if eventually *p* becomes the only process taking steps, then *p* makes progress in α

TM liveness

An infinite execution α is TM-obstruction-free if for every correct process *p* in α the following holds: if eventually *p* becomes the only process taking steps, then *p* makes progress in α



Liveness: take home

When arguing about liveness of a shared object implementation, things to keep in mind:

Liveness: take home

When arguing about liveness of a shared object implementation, things to keep in mind:

- depending on the context liveness properties might be defined different ways

Liveness: take home

When arguing about liveness of a shared object implementation, things to keep in mind:

- depending on the context liveness properties might be defined different ways

- specification might include several different kinds of liveness properties (e.g. TM-obstruction-freedom for transactions + wait-freedom for individual TM operations)

Liveness: take home

When arguing about liveness of a shared object implementation, things to keep in mind:

- depending on the context liveness properties might be defined different ways
- specification might include several different kinds of liveness properties (e.g. TM-obstruction-freedom for transactions + wait-freedom for individual TM operations)
- be accurate when specifying which processes should make progress

Part II

The impossibility of TM-wait-freedom

Wait-freedom

- Wait-freedom forms the basis of consensus number hierarchy

Wait-freedom

- Wait-freedom forms the basis of consensus number hierarchy
- In most cases we need to use powerful base objects (like consensus, CAS) to implement wait-freedom

Wait-freedom

- Wait-freedom forms the basis of consensus number hierarchy
- In most cases we need to use powerful base objects (like consensus, CAS) to implement wait-freedom
- Not the case for TM-wait-freedom:
  - it cannot be implemented together with opacity irrespectively of the power of base objects being used

Impossibility

Theorem

- There is no TM implementation that:
  - ensures TM-wait-freedom and

Impossibility

Theorem

- There is no TM implementation that:
  - ensures TM-wait-freedom and
  - opacity

Impossibility

Theorem

- There is no TM implementation that:
  - ensures TM-wait-freedom and
  - opacity
  - in an asynchronous system

Proof

To prove the result

- We use processes and a scheduler as an *adversary*

Proof

To prove the result

- We use processes and a scheduler as an *adversary*

- The *adversary* forces any TM implementation to produce an execution that violates TM-wait-freedom

Proof: processes

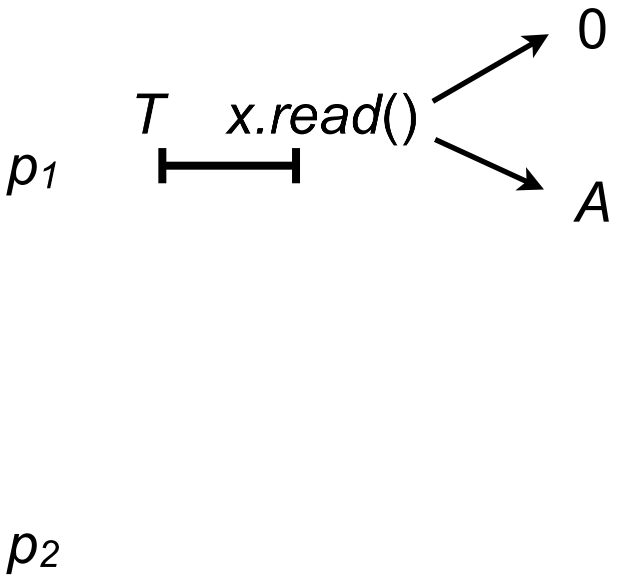
- consider a system of two processes p_1 and p_2

Proof: processes

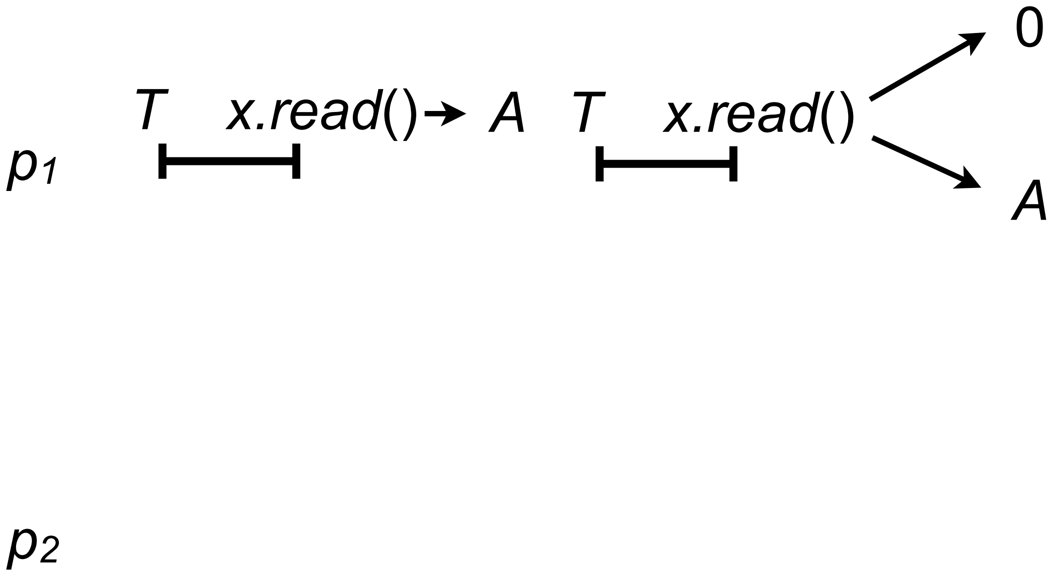
- consider a system of two processes p_1 and p_2
- processes keep executing infinitely many transactions with the following code

```
begin_tr()

    value := x.read( );
    x.write(value + 1);

commit()
```

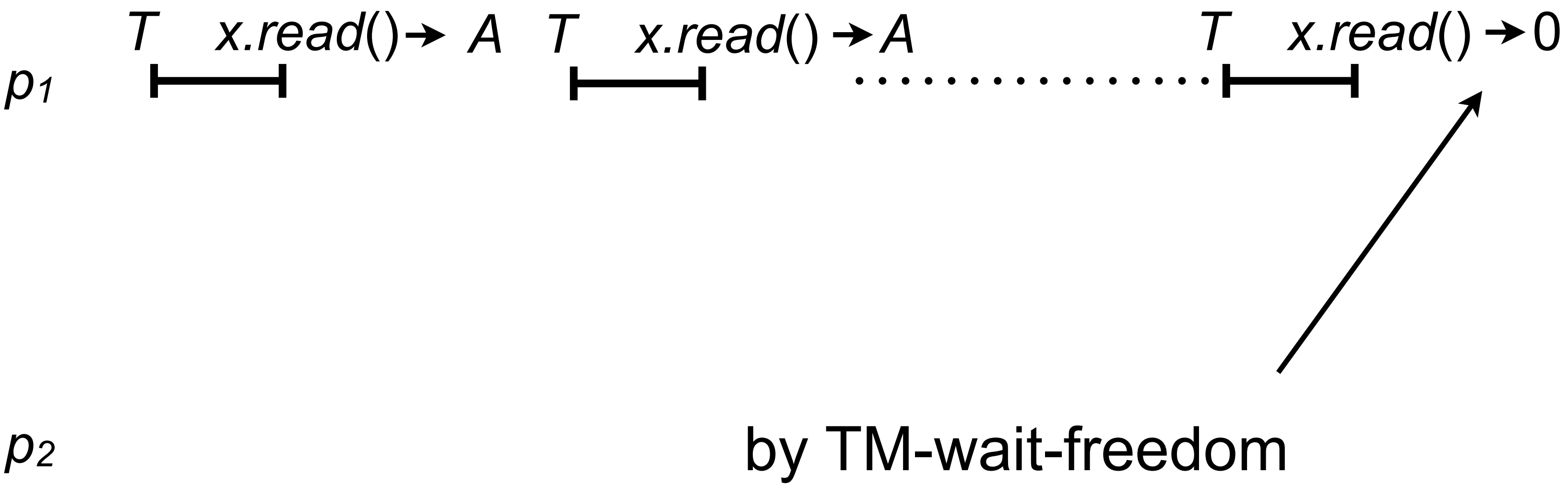
Proof: execution



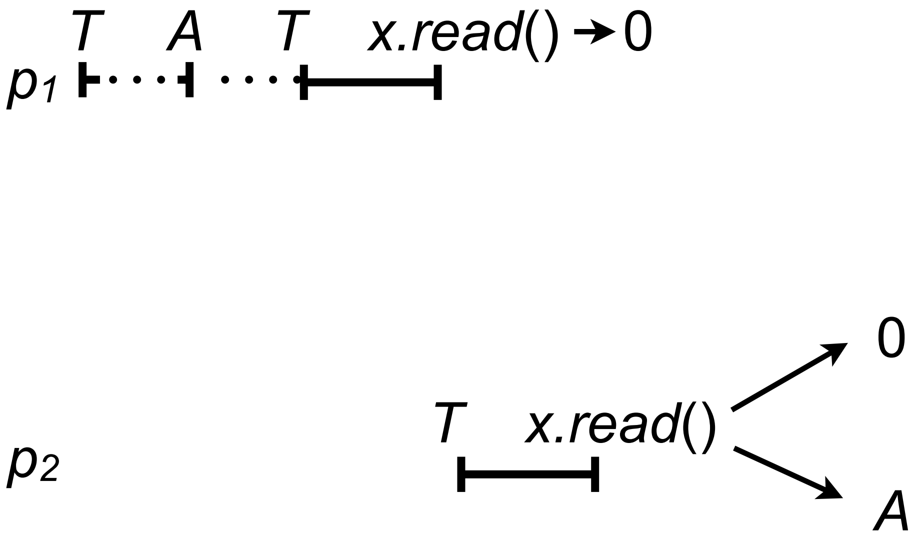
Proof: execution



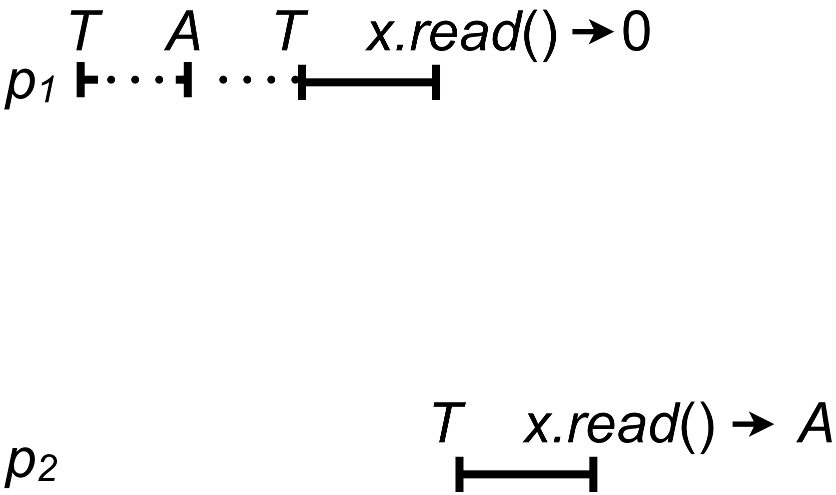
Proof: execution



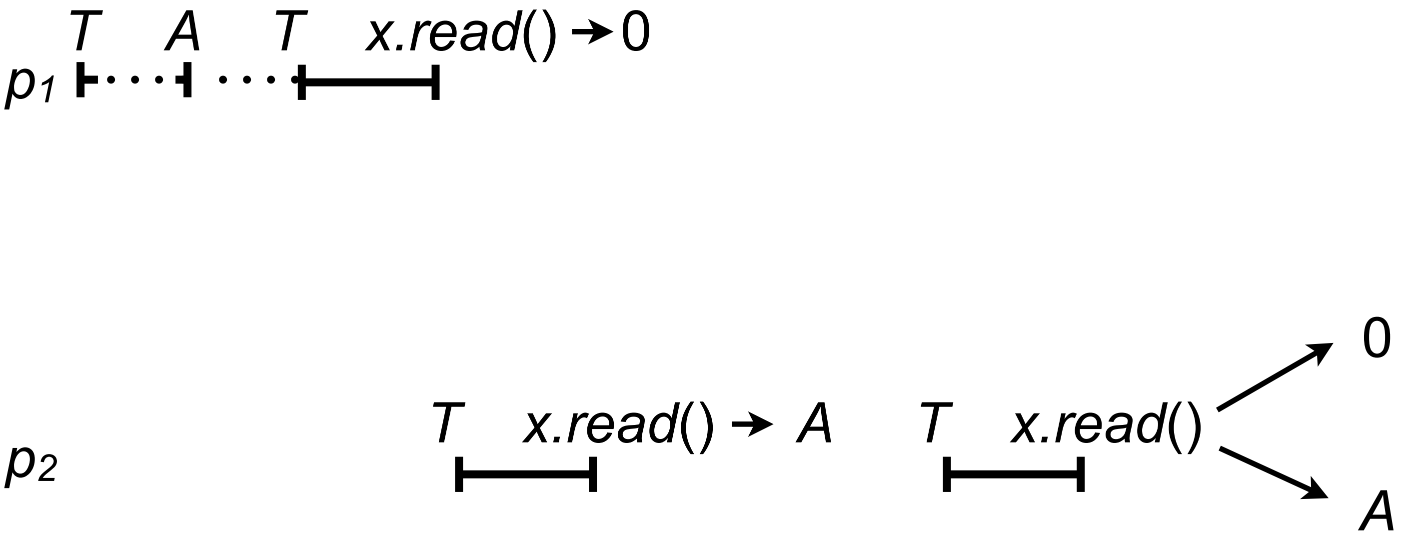
Proof: execution



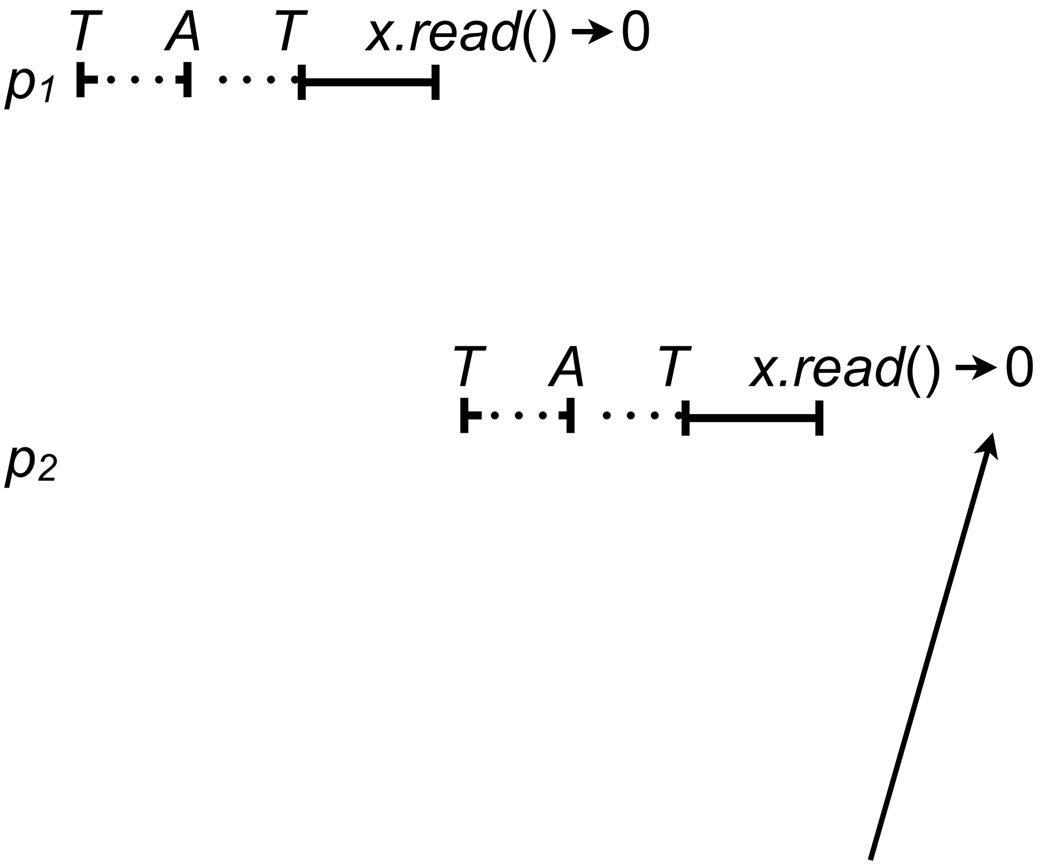
Proof: execution



Proof: execution

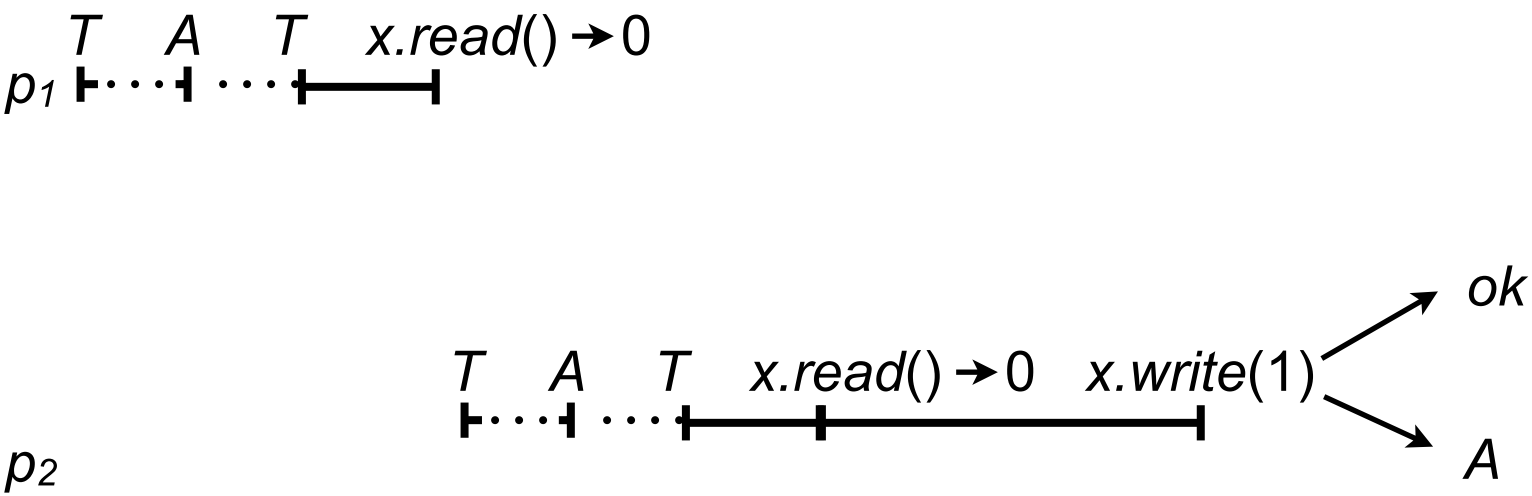


Proof: execution

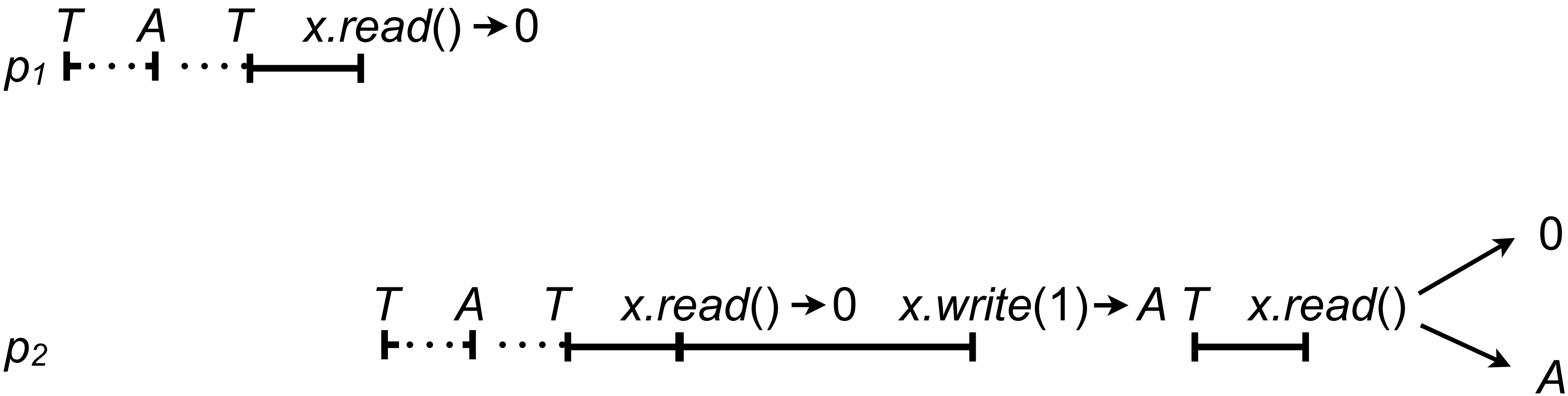


by TM-wait-freedom

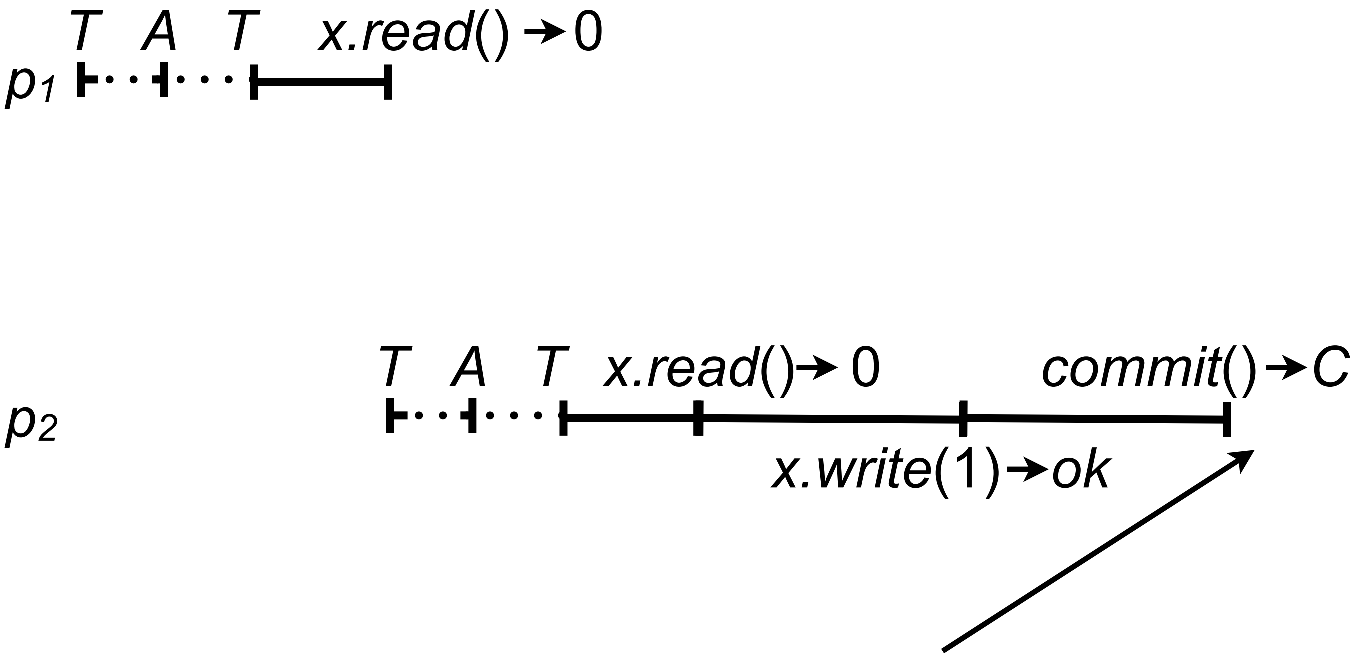
Proof: execution



Proof: execution

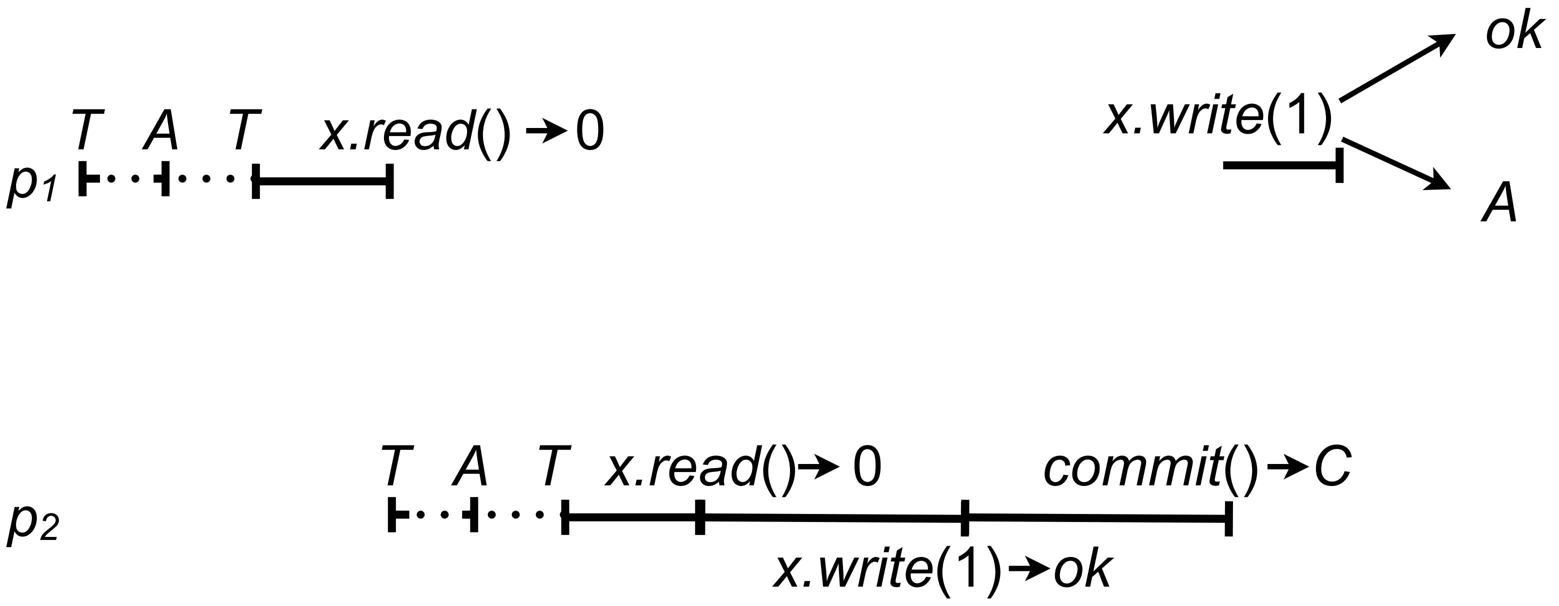


Proof: execution

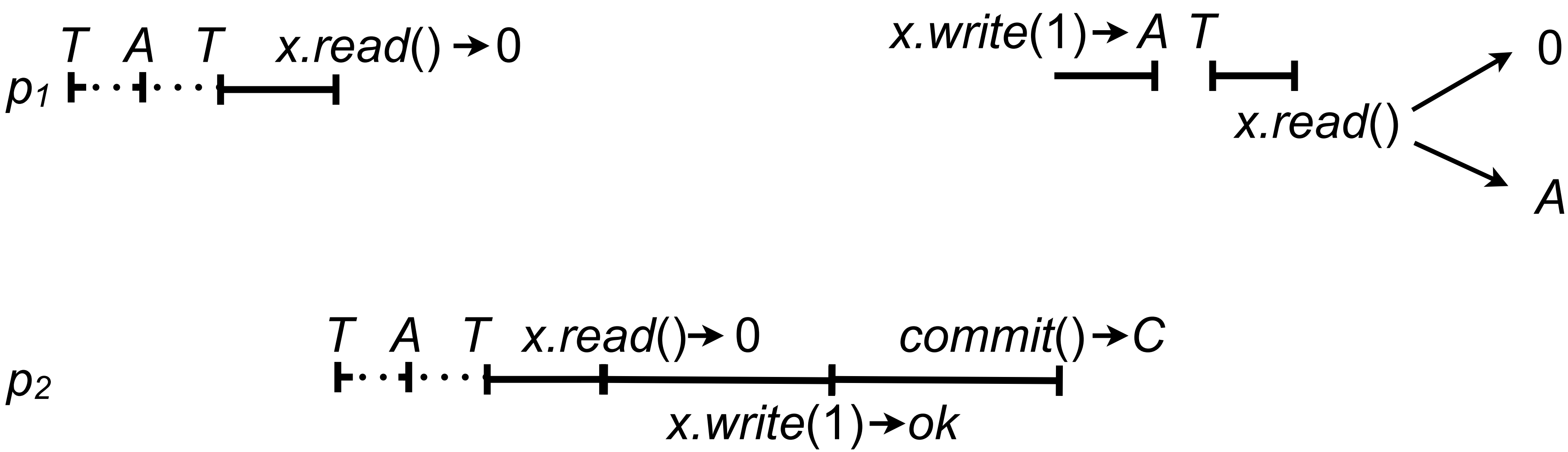


p_2 repeats executing the transaction until eventually the transaction is committed (by TM-wait-freedom)

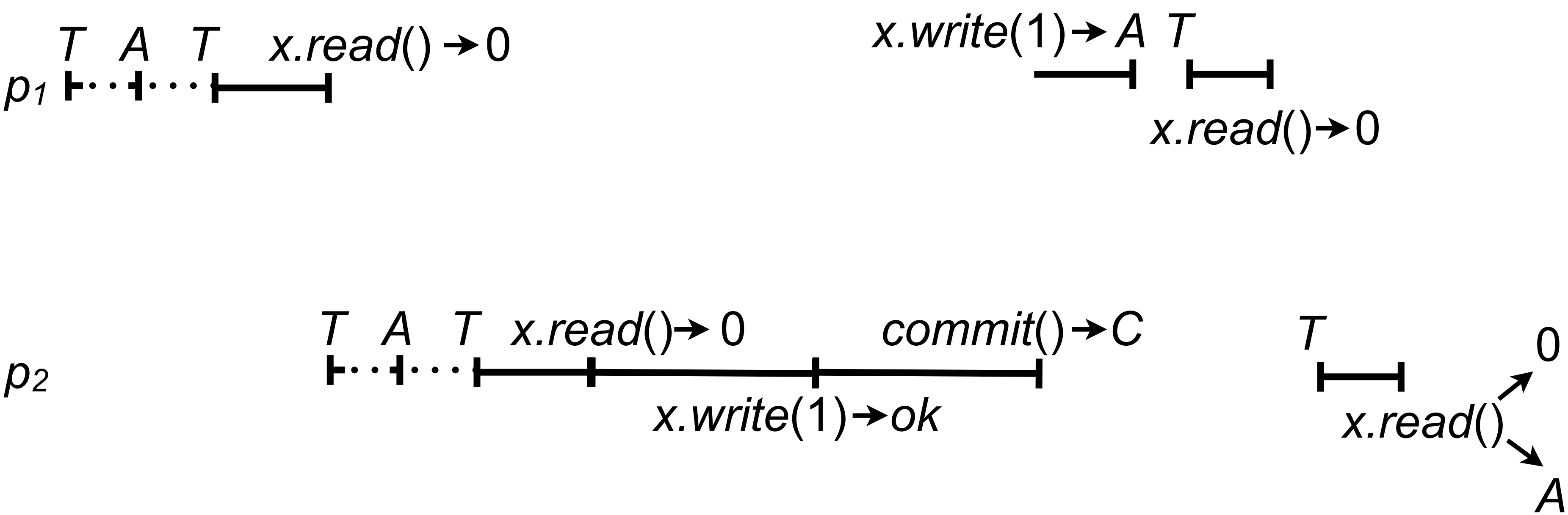
Proof: execution



Proof: execution

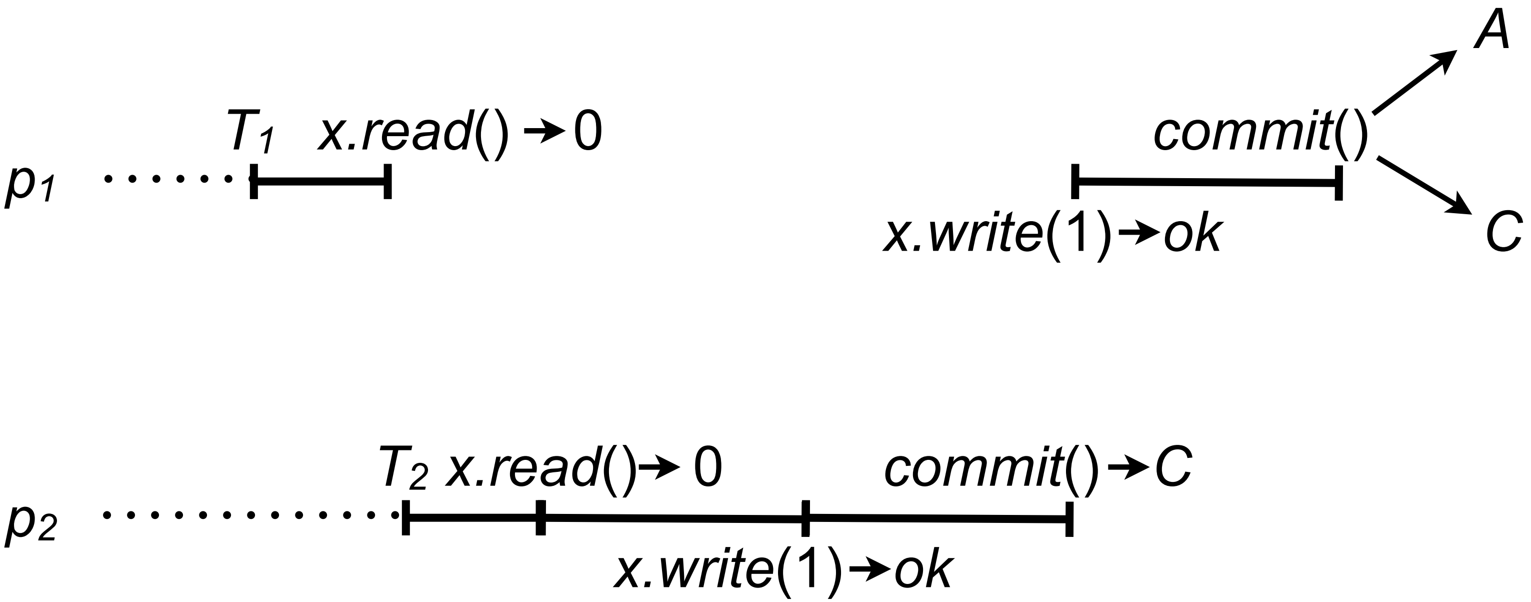


Proof: execution

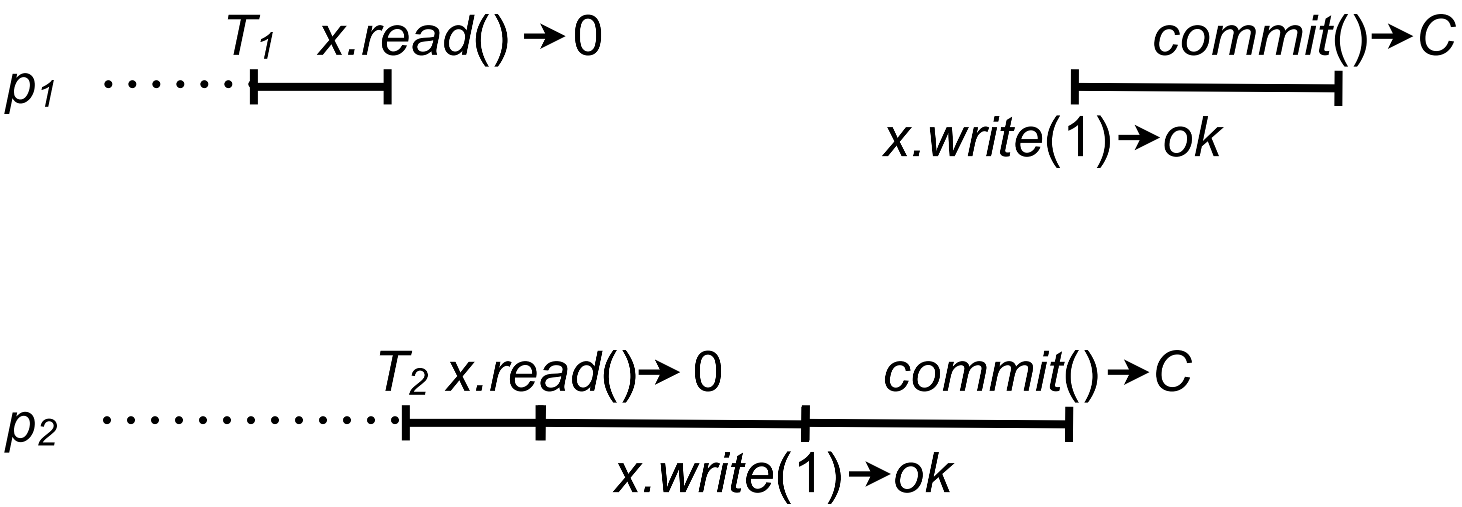


if the write by p_1 aborts we repeat the whole execution again until the write by p_1 is not aborted (by TM-wait-freedom)

Proof: execution

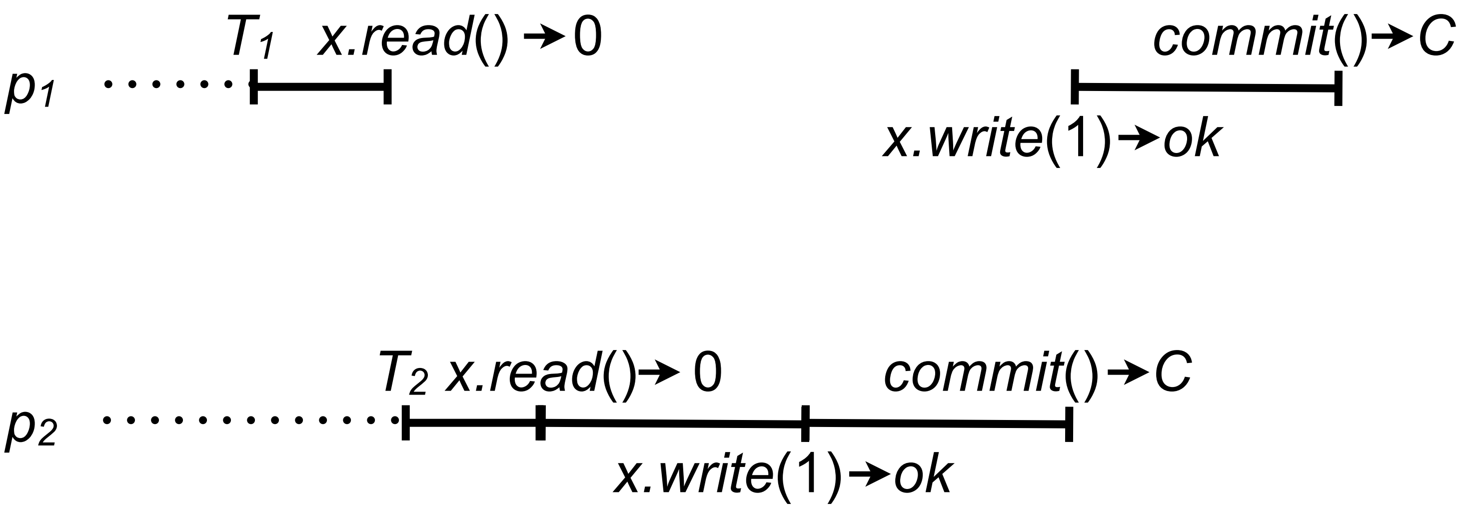


Proof: execution



what happens if T_1 is allowed to commit?

Proof: execution

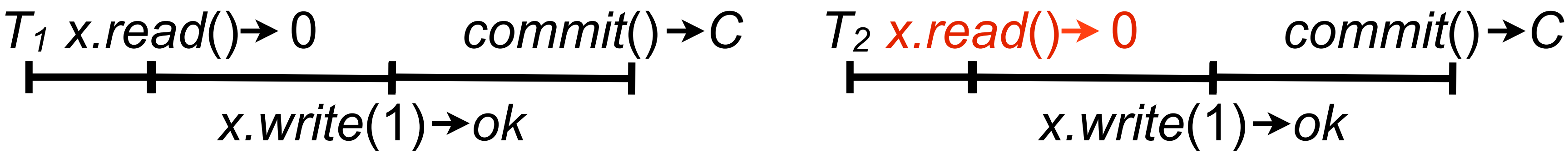


what happens if T_1 is allowed to commit?

- opacity is violated

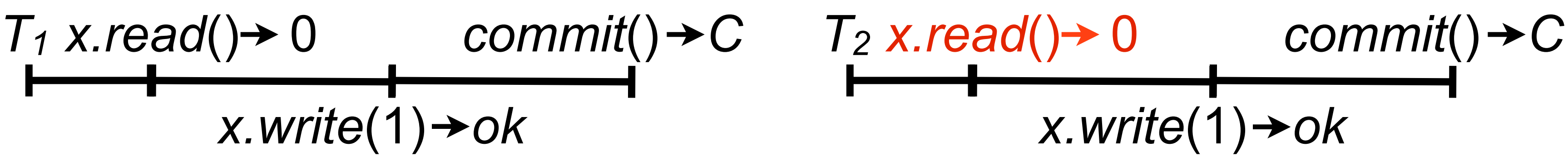
Proof: violating opacity

T_1 is serialized before T_2

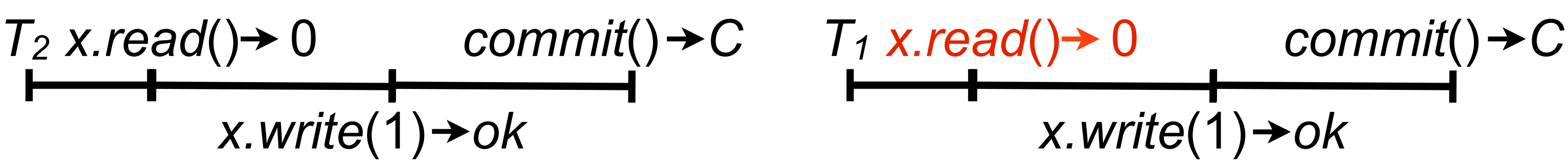


Proof: violating opacity

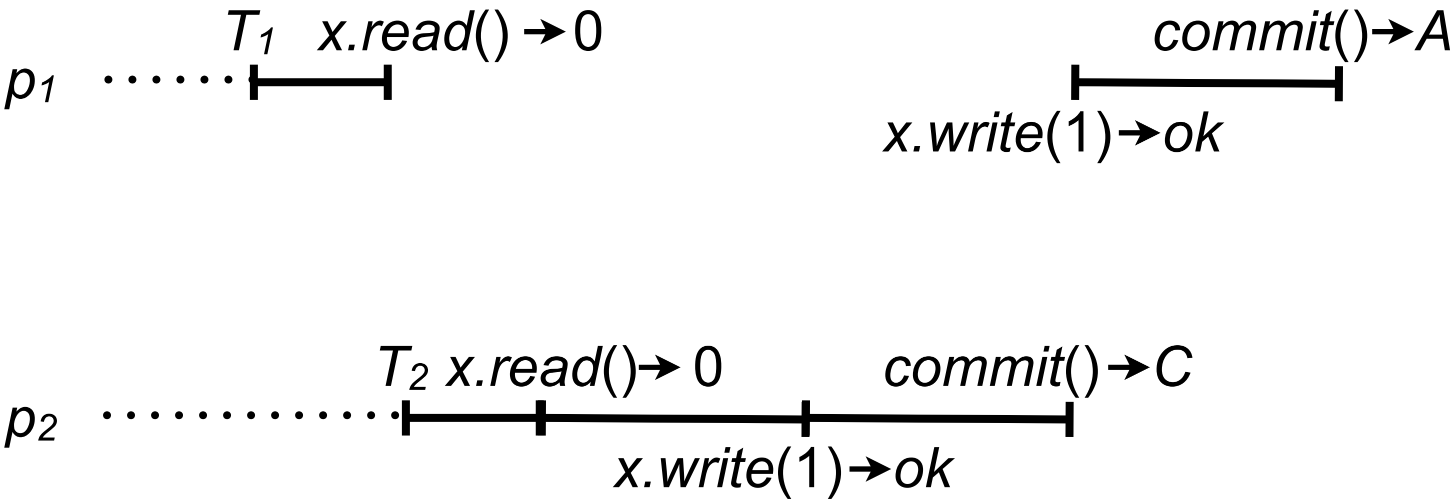
T_1 is serialized before T_2



T_2 is serialized before T_1

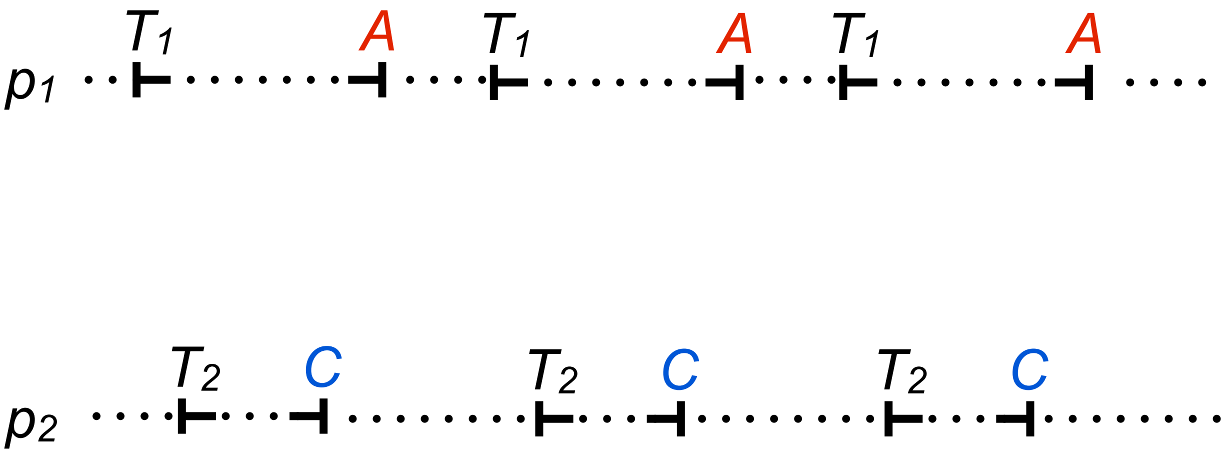


Proof: execution



after aborting T_1 we repeat the execution infinitely often

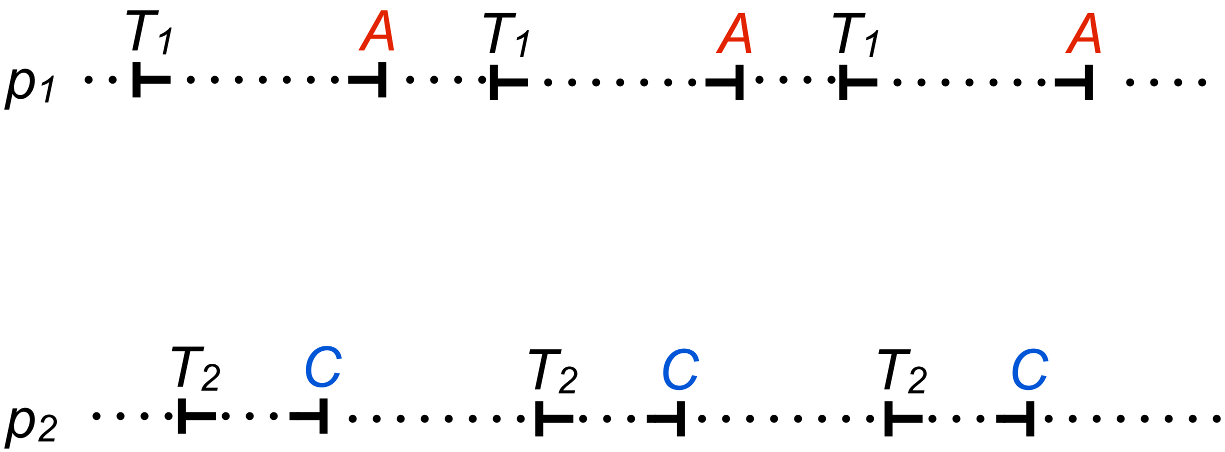
Proof: execution



We get an infinite execution in which:

- p_1 is correct

Proof: execution



We get an infinite execution in which:

- p_1 is correct
- p_1 does not make progress

Circumventing impossibility

To implement TM-wait-freedom

- consider a safety property weaker than opacity

Circumventing impossibility

To implement TM-wait-freedom

- consider a safety property weaker than opacity

- consider a weaker model
  - partially synchronous system in which some process crashes are detectable and no transaction can loop forever without invoking a commit request

Circumventing impossibility

To implement TM-wait-freedom

- consider a safety property weaker than opacity

- consider a weaker model
  - partially synchronous system in which some process crashes are detectable and no transaction can loop forever without invoking a commit request
  - model in which a transaction can be executed by several processes (helping mechanism)

Resources

Overview paper on the liveness of TM:

https://lpd.epfl.ch/site/_media/education/tm_liveness_paper.pdf