Synchronization

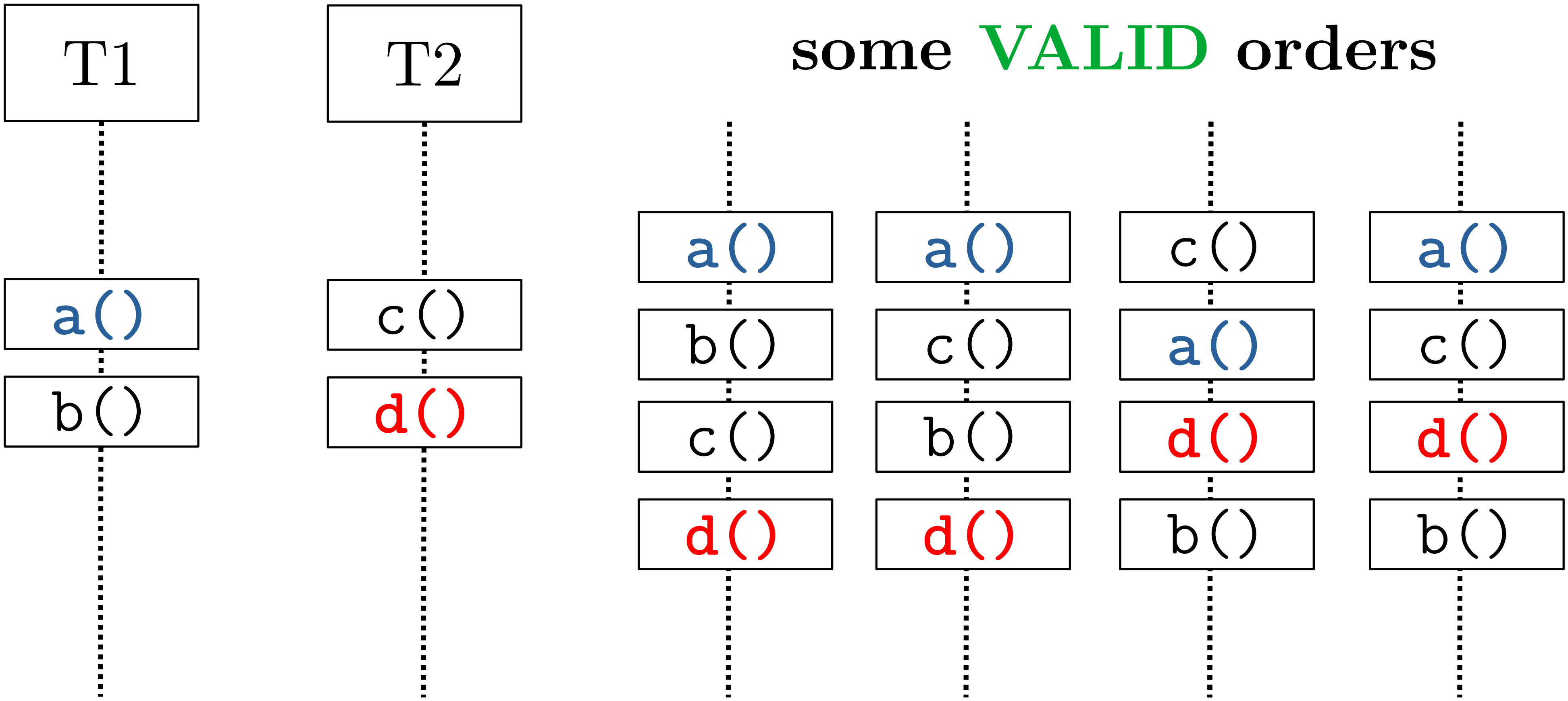
General Synchronization Task

- Operations or instructions of concurrent processes (or threads) can interleave.

- Depending on the situation, there are orders of operations/instructions that can be considered **VALID** and there are orders that can be considered **INVALID**.

- **Synchronization** is about implementing different mechanisms such that only the **VALID** orders of operation/instructions can occur regardless of how the threads/processes are scheduled by the operating system.

Order: Specified Order (i.e. `a() < d()`)



T1

`a()`

`b()`

T2

`c()`

`d()`

some VALID orders

`a()` `b()` `c()` `d()`

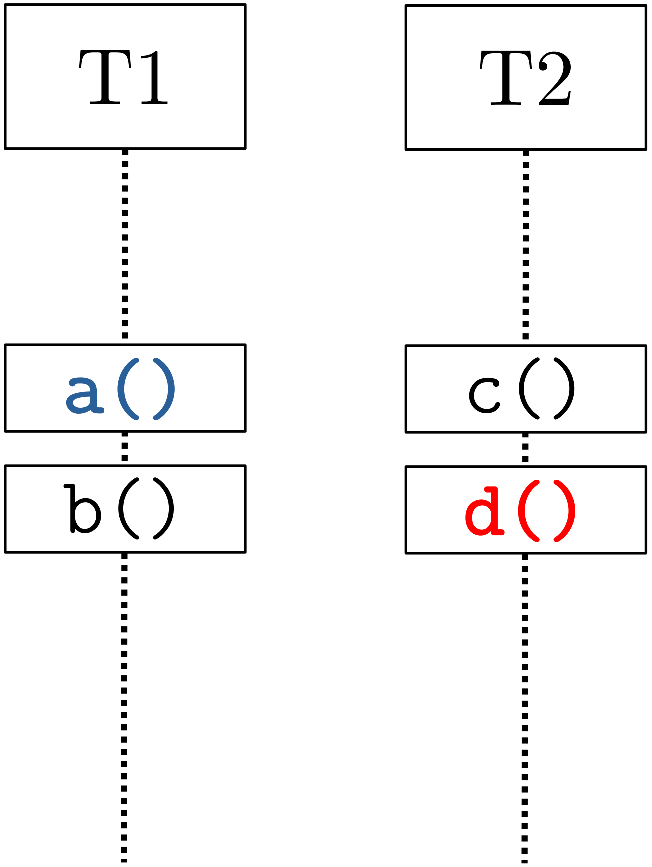
`a()` `c()` `b()` `d()`

`c()` `a()` `d()` `b()`

`a()` `c()` `d()` `b()`

Required: `a() < d()`

Order: Specified Order (i.e. `a() < d()`)



T1

`a()`

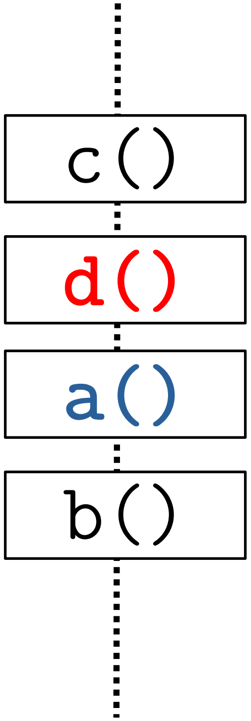
`b()`

T2

`c()`

`d()`

INVALID order



`c()`

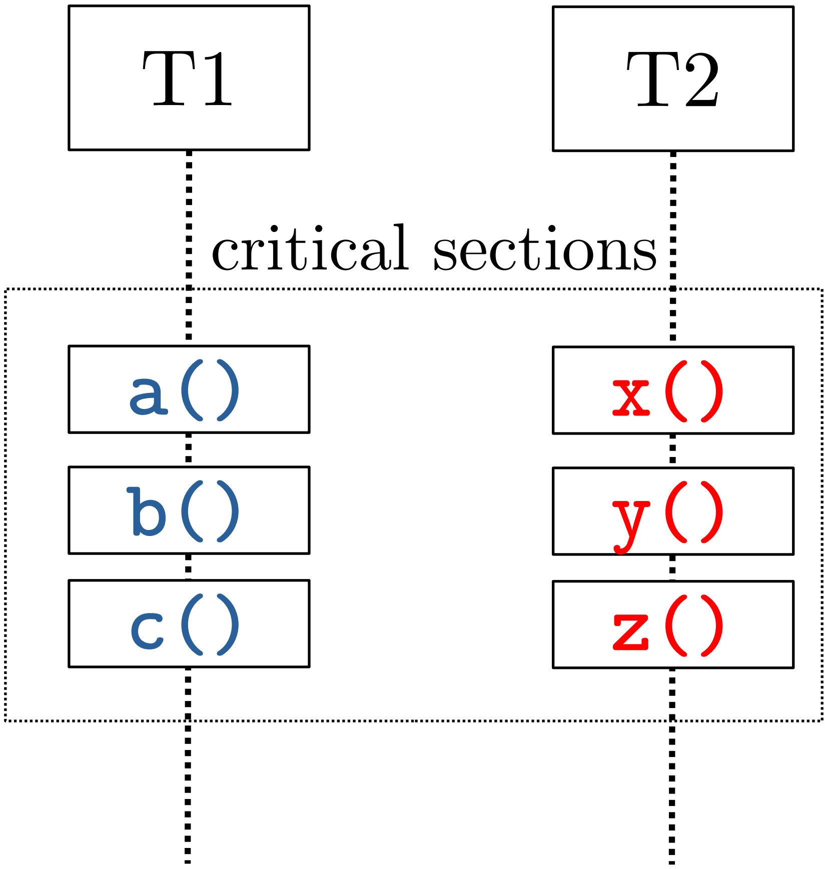
`d()`

`a()`

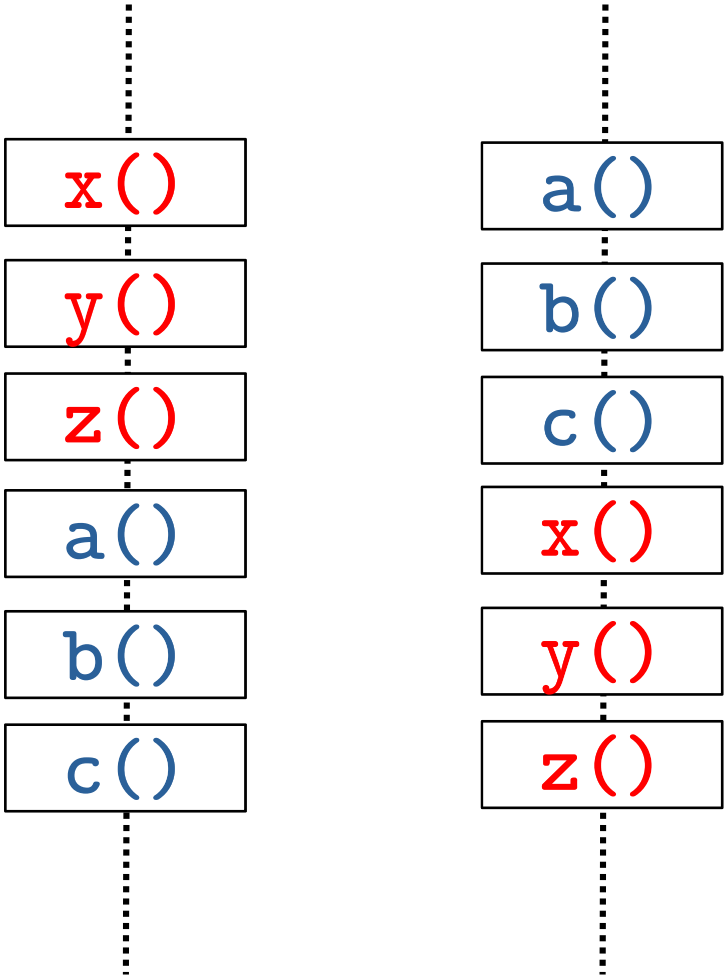
`b()`

Required: `a() < d()`

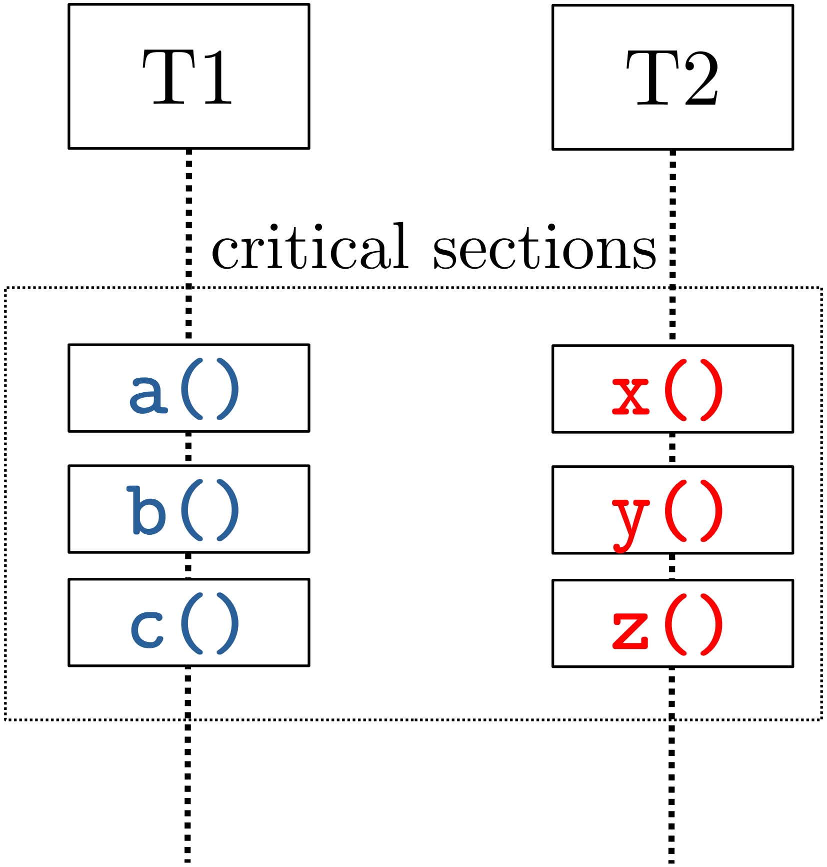
Order: Mutual Exclusion (of Critical Section)



VALID orders



Order: Mutual Exclusion (of Critical Section)


T1

T2

critical sections

a()

b()

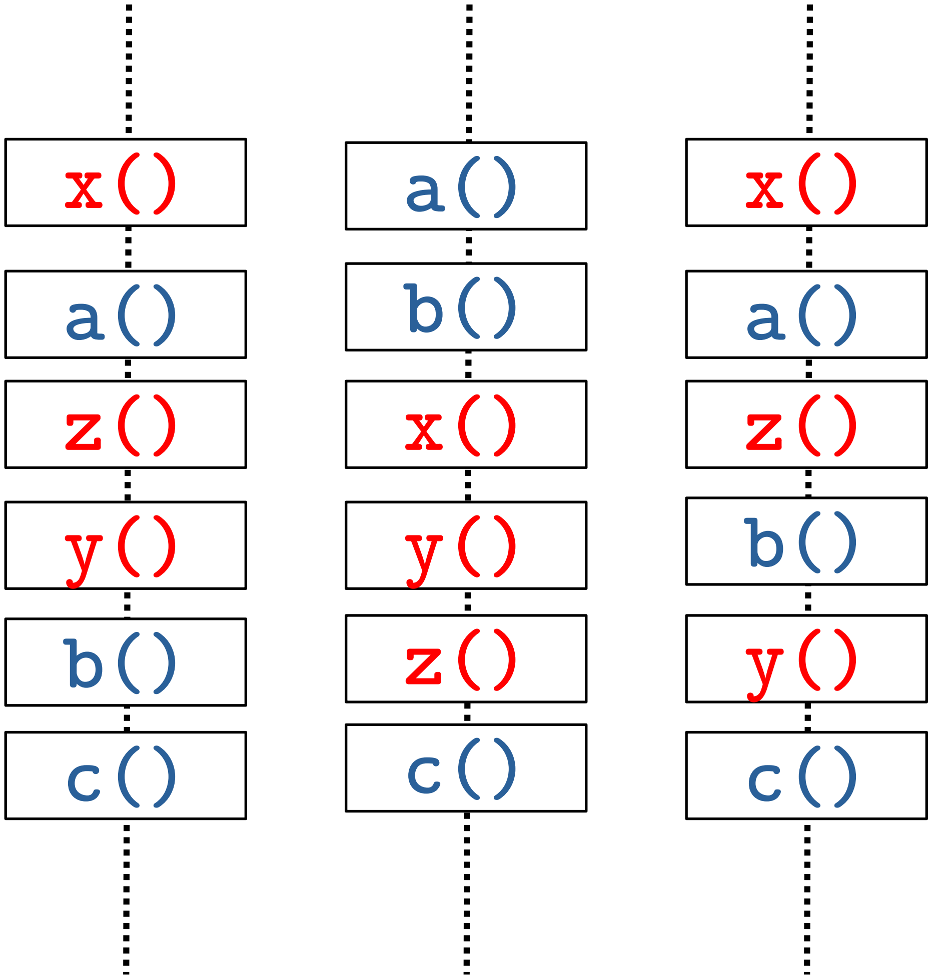
c()

x()

y()

z()

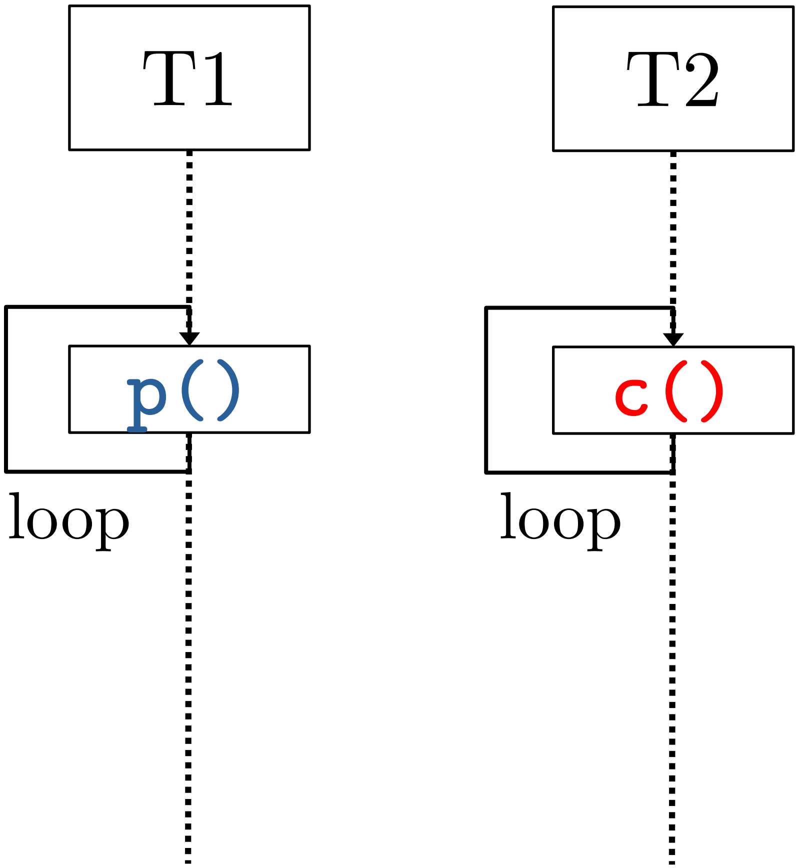
some **INVALID** orders


x()
a()
z()
y()
b()
c()

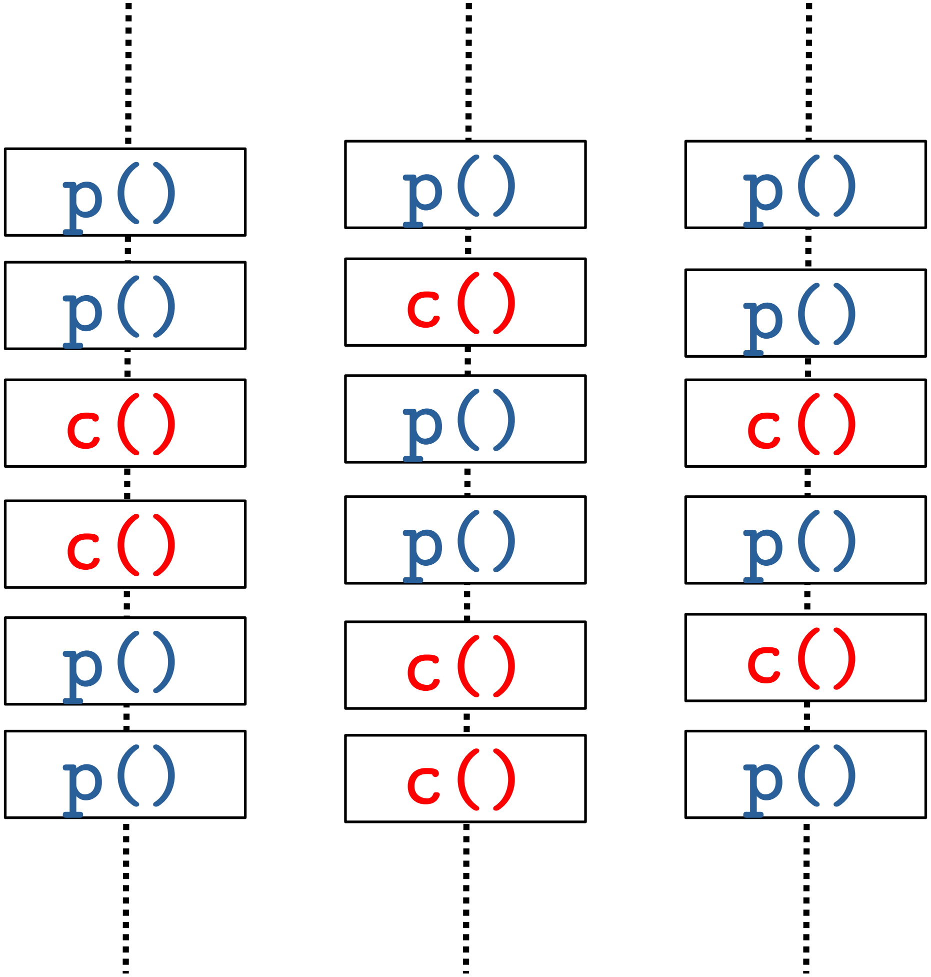
a()
b()
x()
y()
z()
c()

x()
a()
z()
b()
y()
c()

Order: Producer(p)-Consumer(c)

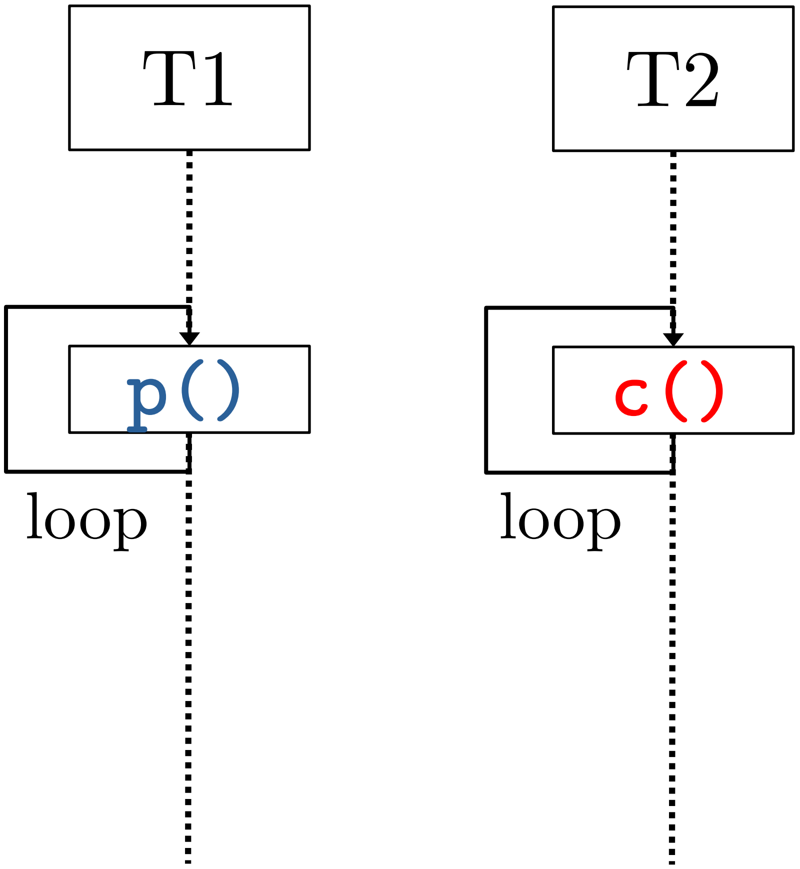


some VALID orders

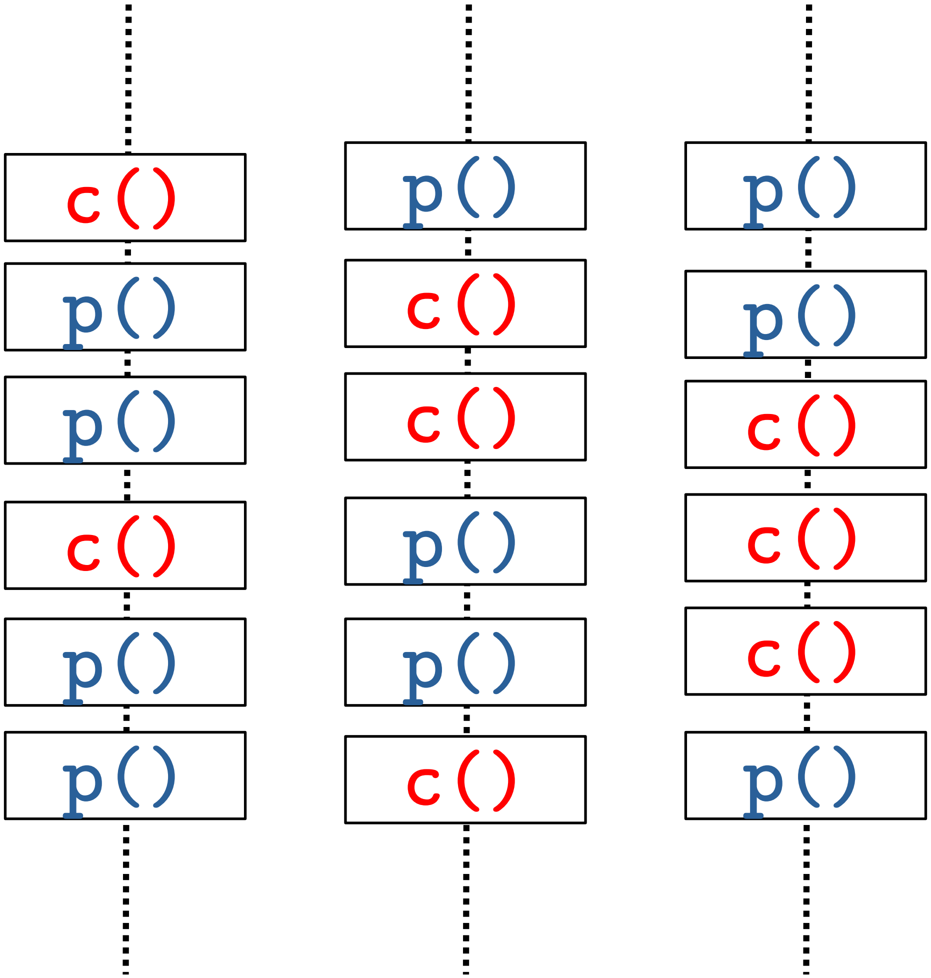


Order: before a `c()` occurs a corresponding `p()` should have occurred first

Order: Producer(p)-Consumer(c)



some **INVALID** orders



Order: before a `c()` occurs a corresponding `p()` should have occurred first

Synchronization Object: Semaphore

Semaphore `S` in an **integer** that signals permission or availability of resources.

`S = n > 0` means there are `n` permits or resources available.

`wait(S)` – a method called by a process/thread in order to request (to **wait** for) a permit or access to a resource. `Decrements S (i.e. S--;)`.

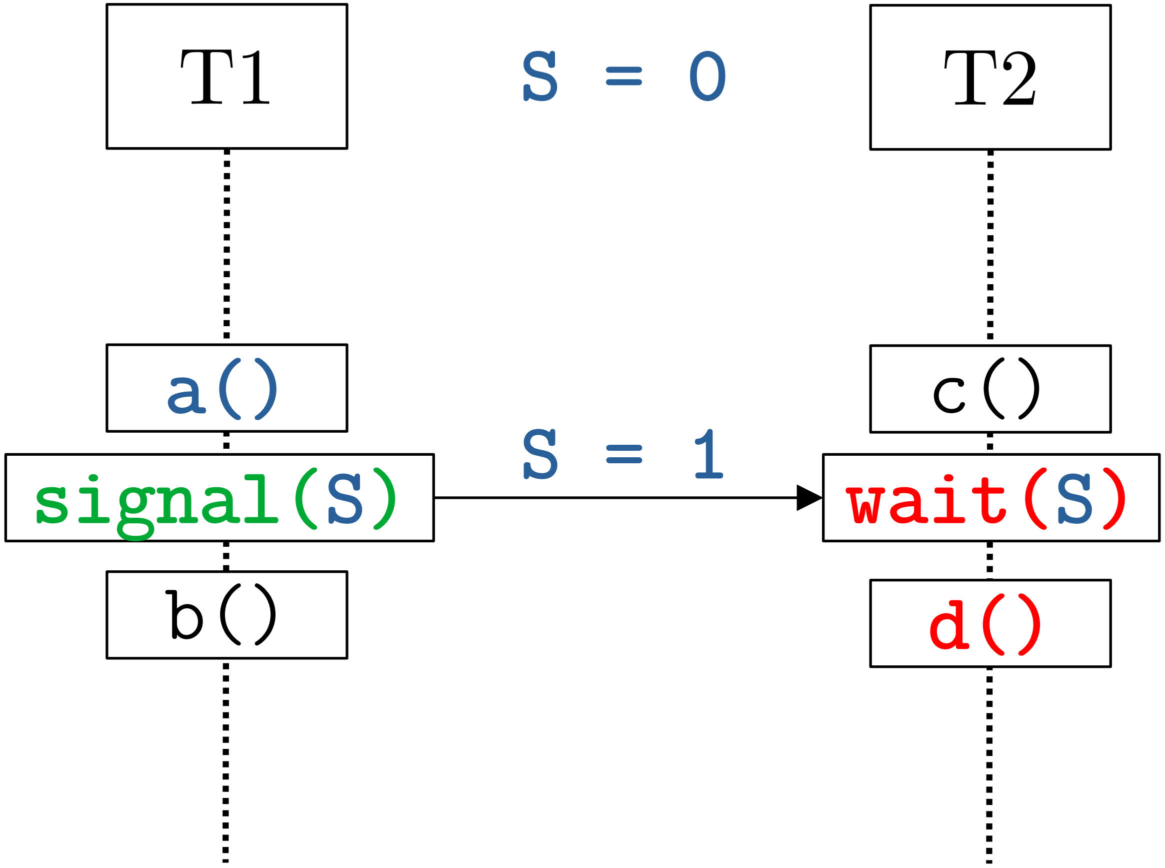
`signal(S)` – a method called by a process/thread in order to return a permit or a resource. `Increments S (i.e. S++;)`.

Synchronization Object: Semaphore

```
wait(S)
{
  while(S <= 0)
  {
    // busy wait
  }
  S--;
}
```

```
signal(S)
{
  S++;
}
```

Semaphore: Specified Order (i.e. `a() < d()`)



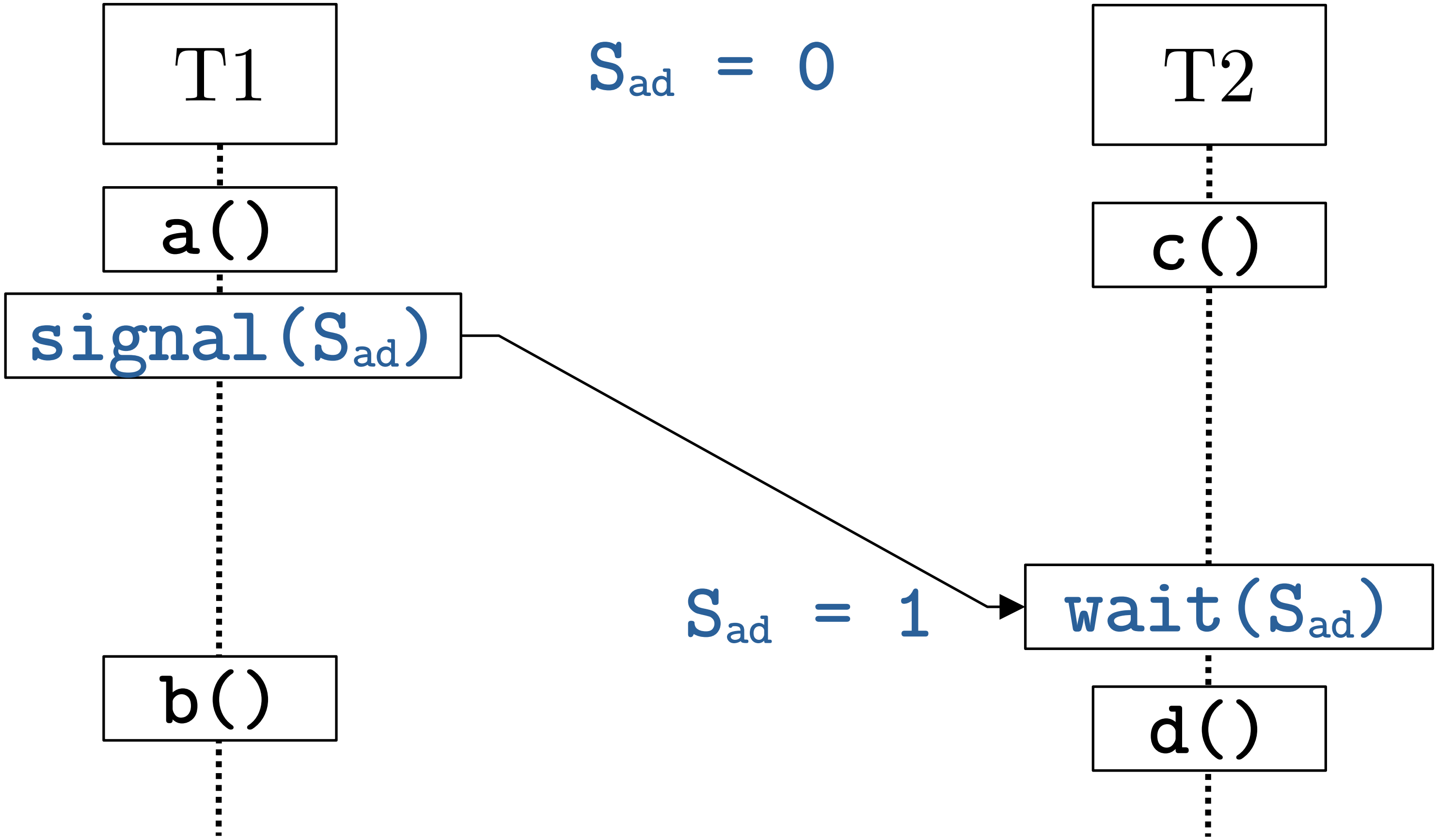
Required: `a() < d()`

Initial condition: `S = 0` means *0 permit* is available.

T2 can only exit `wait(S)` and call `d()` if there is at least ***1 permit***.

The ***1 permit*** is only available when **T1** calls `signal(S)` after `a()` executes.

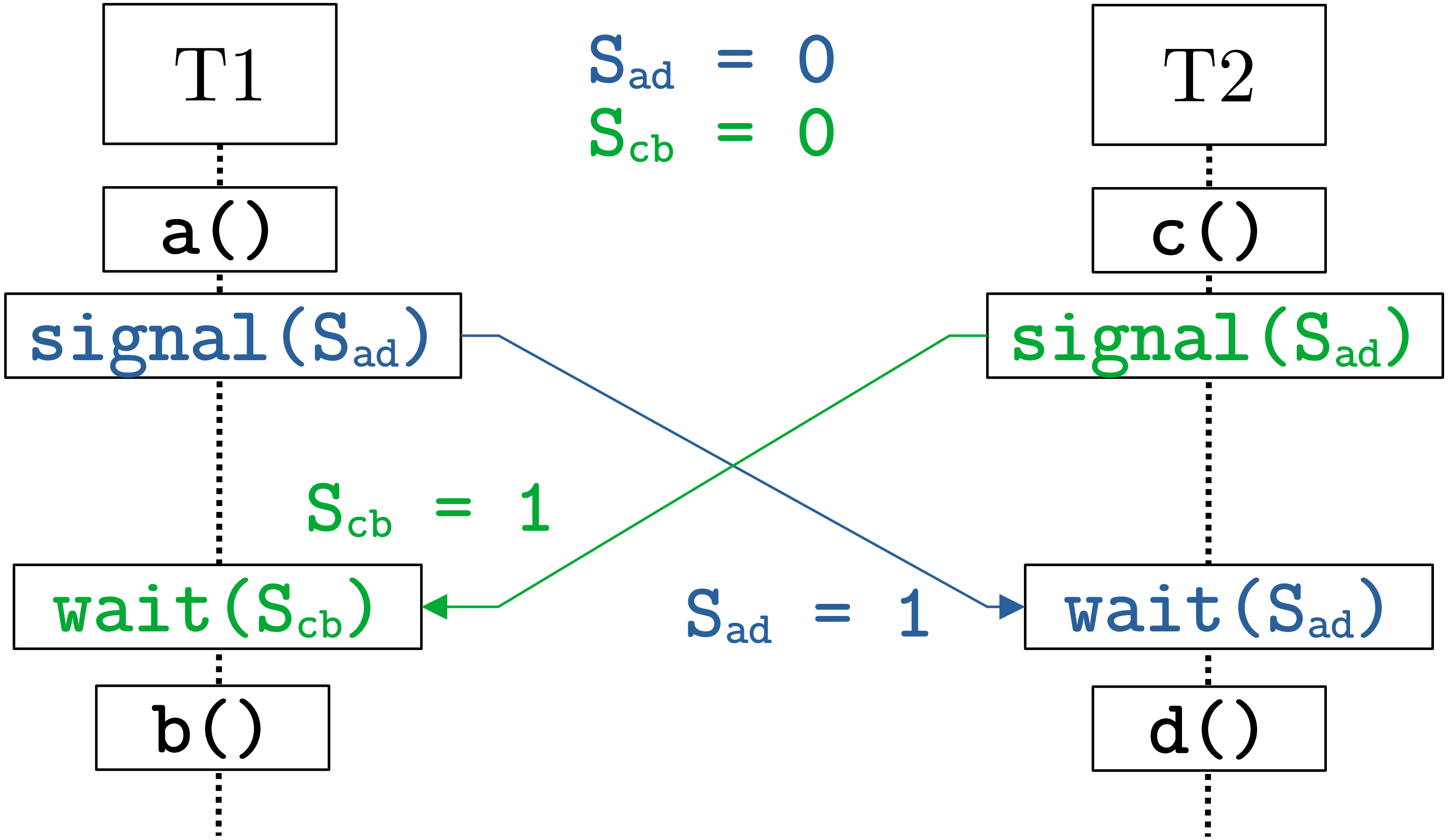
Semaphore: Specified Order (i.e. a()<d() and c()<b())



Required: a() < d()

Required: c() < b()

Semaphore: Specified Order (i.e. a()<d() and c()<b())



T1

a()

signal(S_{ad})

S_{cb} = 1

wait(S_{cb})

b()

S_{ad} = 0

S_{cb} = 0

S_{ad} = 1

T2

c()

signal(S_{ad})

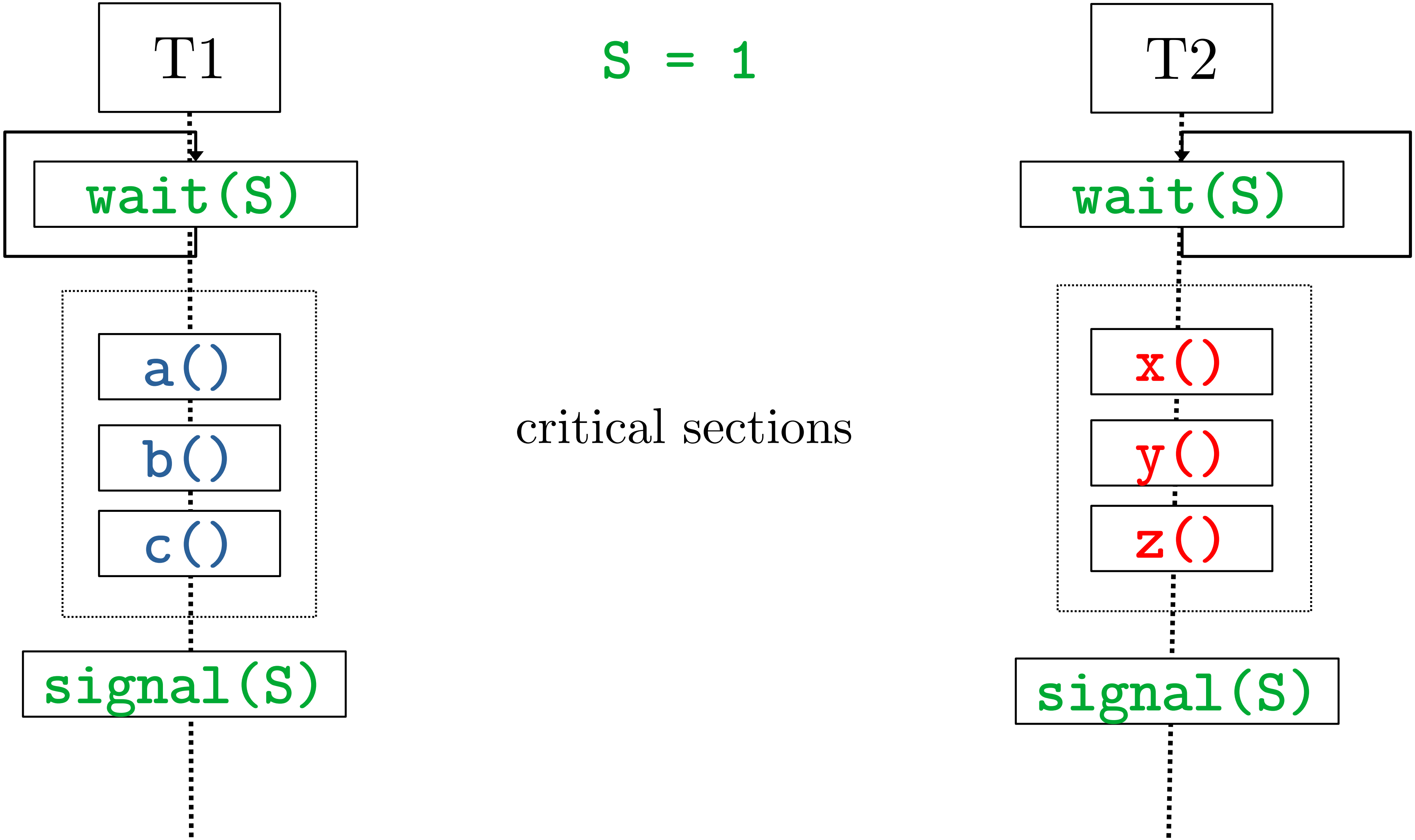
wait(S_{ad})

d()

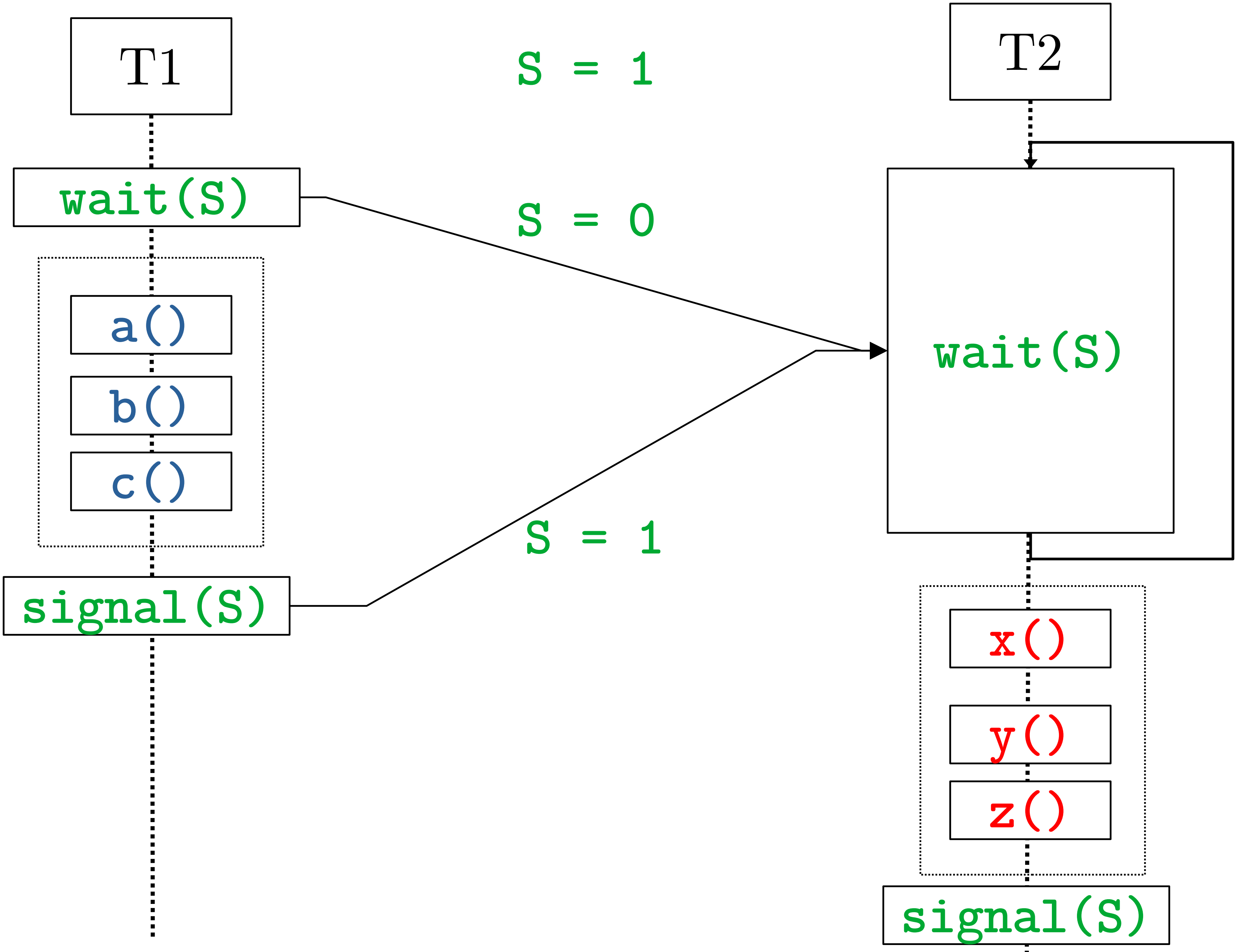
Required: a() < d()

Required: c() < b()

Semaphore: Mutual Exclusion



Semaphore: Mutual Exclusion



T1
S = 1
T2
wait(S)
S = 0
wait(S)
a()
b()
c()
S = 1
signal(S)
x()
y()
z()
signal(S)