TDC1:
Solving consensus

So far

- Consensus in read-write shared memory
  - ✓Definition
  - ✓Impossibility of wait-free consensus
- 1-resilient simulation
  - ✓Consensus impossibility for the general case

Today

- Variants of consensus
  - ✓Safe agreement
  - ✓Commit-adopt
- Circumventing consensus impossibilities
  - ✓Using a failure detector
  - ✓Using “strong” objects (queues, CAS)

System model

- N *asynchronous* processes $p_0,\ldots,p_{N-1}$ ($N \geq 2$) communicate via reading and writing in the shared memory

- Processes can fail by crashing
  - ✓Up to t processes can crash: t-resilient system
  - ✓t=N-1: wait-free

- The processes communicate via atomic (NWNR) registers and atomic N-snapshots

Consensus: definition

A process *proposes* an *input* value in V (|V|≥2) and tries to *decide* on an *output* value in V

- *Agreement:* No two process decide on different values
- *Validity:* Every decided value is a proposed value
- *Termination:* No process takes infinitely many steps without deciding

  (Every correct process decides)

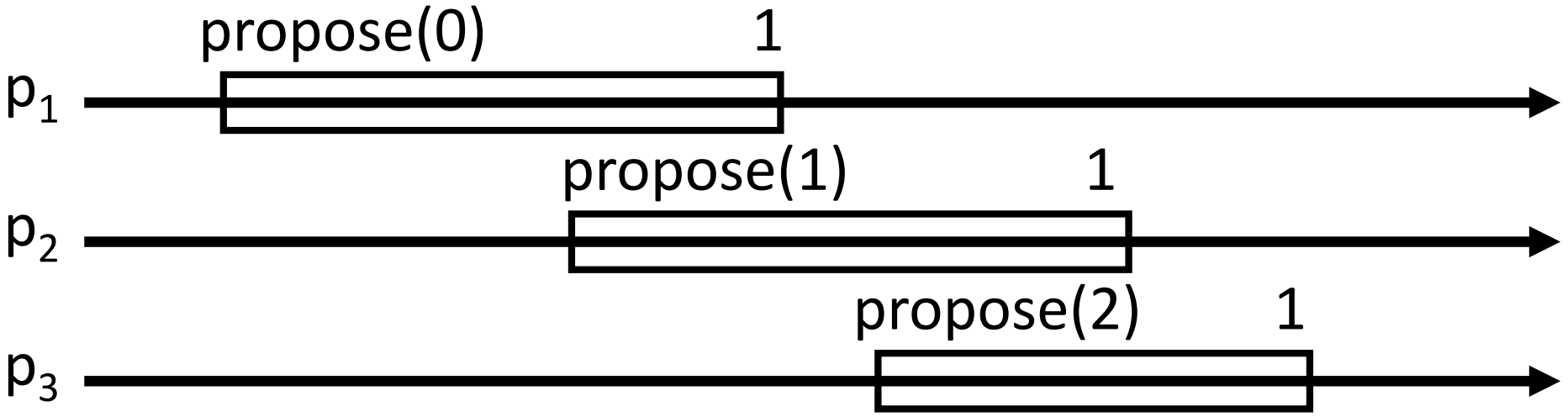
Consensus: sequential spec

A consensus object exports one operation propose(v), v in V, that returns a value in V

In a sequential execution, every invocation of propose() returns the argument of the first propose()

Safety: linearizability

Liveness: every propose() invoked by a correct process eventually returns



Solving consensus

For every model M

There is an algorithm A that in every run of A in M satisfies Agreement, Validity, and Termination

Iff

There is an linearizable implementation of a consensus object that guarantees that every invocation of propose() by a correct process eventually returns

Consensus impossibilities

- Wait-free consensus is impossible
  - ✓In particular, 2-process wait-free consensus
- 1-resilient consensus is impossible (by reduction)
  - ✓If not, 2-process wait-free consensus can be solved

Today

- Variants of consensus
  - ✓Safe agreement
  - ✓Commit-adopt
- Circumventing consensus impossibilities
  - ✓Using a failure detector
  - ✓Using “strong” objects (queues, CAS)

Safe agreement

- Safety: agreement + validity
- Liveness: if every participant takes enough (three) shared-memory steps, then every correct process decides
  - ✓A process participates if it takes at least one step

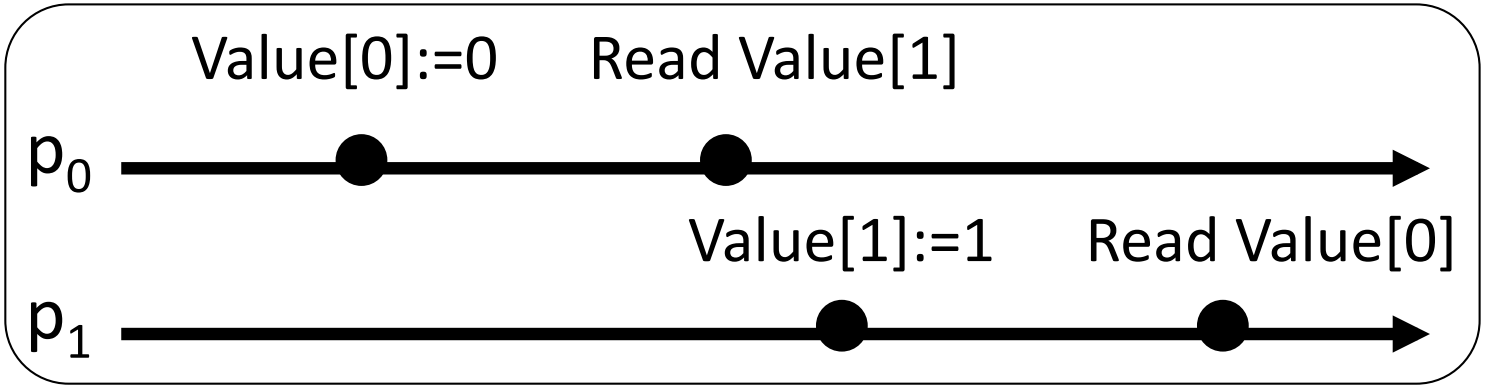
Two process safe agreement

Shared: Flag[0,1], initially T; Value[0,1], initially T

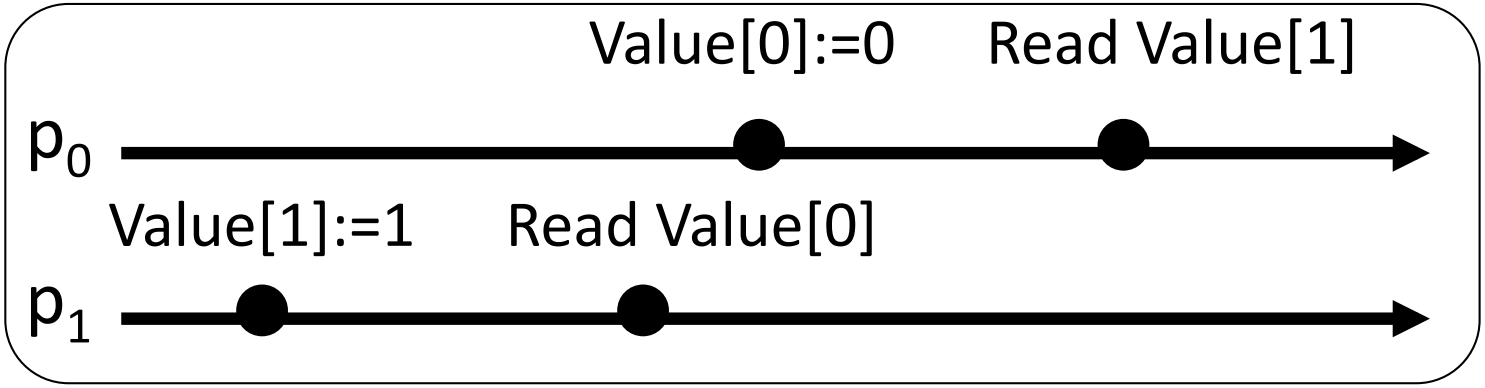
Upon propose(v) of process p_i:

```
Value[i] := v
x := Value[1-i]
if x = T then
        Flag[i] := 1            // p_i is the winner
        return(v)
Flag[i] := 0
wait until Flag[1-i] ≠ T
if Flag[1-i] = 1 then
        return x                // p_{1-i} is the winner
else
        return Value[0]         //break the symmetry
```

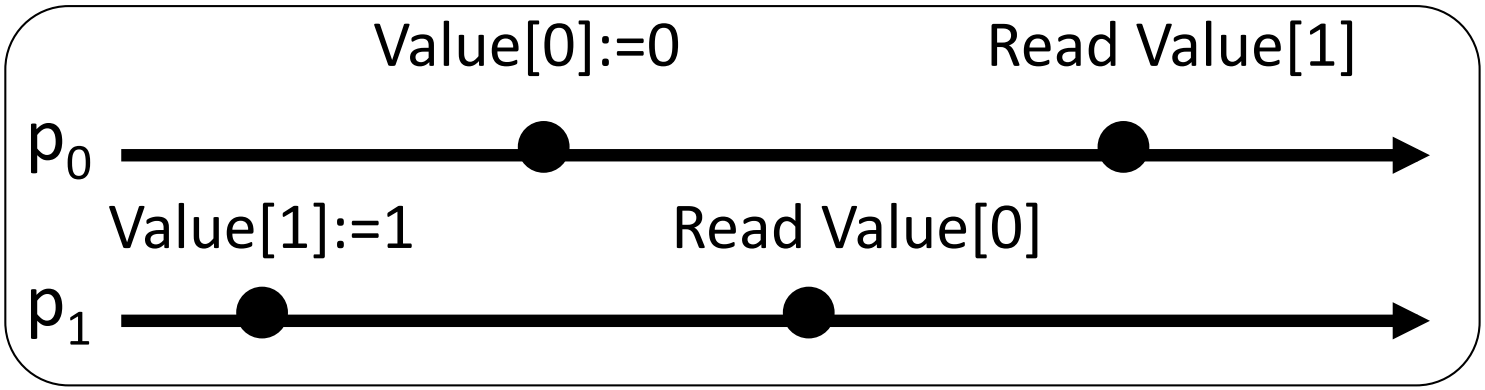
Correctness: at most one winner



p_0 is the only winner



p_1 is the only winner



No winners

Safe agreement for N processes

- Similar idea, but use atomic snapshots
- Atomic snapshot object A exports operations $update_i(.)$, i=0,…N-1, and scan()
  - ✓$update_i(.)$ accepts a value and returns ok
  - ✓scan() returns a vector S[0,…,N-1] so that each S[i] is the argument of the latest $update_i(.)$
    the initial value T if no such $update_i()$

- If at most one $update_i()$ for each i, then all scans are related by containment: for all scans S and S', we have
  - ✓for all j: S[j]=T or S[j]=S'[j], or
  - ✓for all j: S'[j]=T or S[j]=S'[j]

Safe agreement for N processes

Shared: atomic snapshot objects

A[0,…,N-1], B[0,…,N-1], all initially T

Upon propose(v) by process p_i:

A.update$_i$(v)
U:=scan(A)
B.update$_i$(U)
repeat
V:=scan(B)
until for all j in U such that U[j] ≠T, V[j]≠T
decide on the smallest input in the smallest V[j]≠T

Safe agreement: correctness

Liveness: immediate

Safety (intuition):

Consider p_t that wrote the smallest snapshot S to B[t]

- ✓ for all B[j]≠ T, p_t is in B[j]
- ✓ every p_i waits until p_t writes
- ✓ every p_i decides on the smallest input in S

- HW: give a complete proof

Today

- Variants of consensus
  - ✓Safe agreement
  - ✓Commit-adopt
- Circumventing consensus impossibilities
  - ✓Using a failure detector
  - ✓Using “strong” objects (queues, CAS)

Commit-adopt

A process p_i *proposes* an *input* value in V (|V|≥2) and *decides* on a tuple (c,v) where c is a boolean and v is in V

- ✓We say p_i *adopts* v
- ✓If c=true, we say p_i *commits* on v

Commit-adopt: properties

- *Validity:* Every adopted value is an input value of some process
- *Termination:* Every correct process decides
- *CA-Agreement:*
  - ✓If a process commits on a value v, then no process can adopt a value v'≠v
  - ✓If all inputs are the same, then no process decides on (false,*)

    (every process that decides commits on a value)

Commit-adopt: proof

Validity and Termination: immediate

CA-Agreement:

Claim 1 B[0,…,N-1] never contains (true,v) and (true,v') where v≠v'

Suppose not: p_i wrote (true,v) in B[i] and pj wrote (true,v') in B[j], v≠v'

Previously, p_i wrote v in A[i] and pj wrote v' in A[j] (let pi be the first to write)

But p_j should have seen A[i] ≠v' - a contradiction!

Commit-adopt : protocol

Shared objects:

N atomic registers A[0,…,N-1], initially T

N atomic registers B[0,…,N-1], initially T

Upon propose(v) by process p_i:

v_i := v

A[i] := v_i

V := read A[0,…,N-1]

if all non-T values in V are v then

B[i] := (true,vi)

else

B[i] := (false,vi)

V := read B[0,…,N-1]

if all non-T values in V are (true,*) then

return (true,v_i)

else if V contains (true,v) then

v_i := v

return (false,v_i)

Commit-adopt: proof

Validity and Termination: immediate

CA-Agreement:

Claim 1 B[0,…,N-1] never contains (true,v) and (true,v') where v≠v'

Suppose not: p_i wrote (true,v) in B[i] and p_j wrote (true,v') in B[j], v≠v'

Previously, p_i wrote v in A[i] and p_j wrote v' in A[j] (let p_i be the first to write)

But p_j should have seen A[i] ≠v' - a contradiction!

Commit-adopt: proof (contd.)

Claim 2 If p_i returns (true,v) then no process pj returns (c,v') where v≠v'

Suppose not: let p_j return (c,v') where v≠v'.

By Claim 1, p_j has previously written some (false,v'') in B[j]

Since p_j hasn't adopted v, it hasn't found (true,v) in B[1,…,N]

But then p_i should have read (false,v'') in B[j] – a contradiction!

Commit-adopt: proof (contd.)

Claim 3 If all inputs are the same then no process returns (false,*)

Immediate: both “if” conditions are true, i.e., the non-T values in A and B are the same

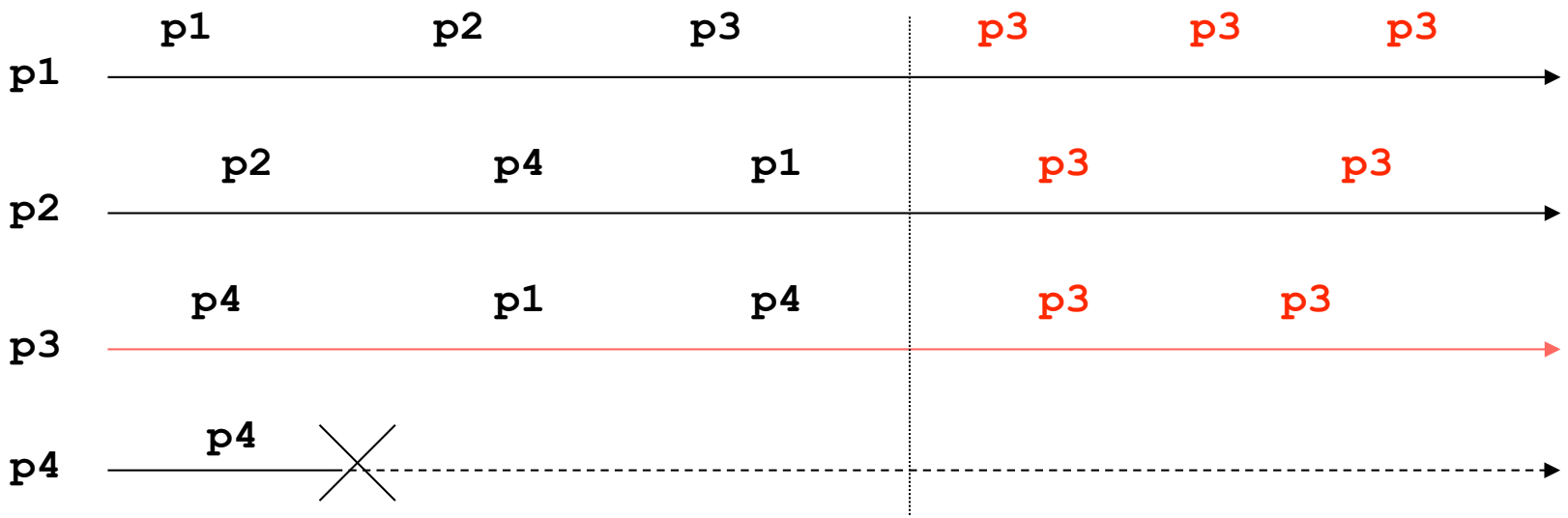
Today

- Variants of consensus
  - ✓Safe agreement
  - ✓Commit-adopt
- Circumventing consensus impossibilities
  - ✓Using a leader oracle Ω
  - ✓Using “strong” objects (queues, CAS)

Leader election Ω

At every process and each time Ω outputs a process identifier

Eventually, the same correct process is output at every correct process



Consensus with Ω and Commit-adopt

Shared:

D[1,…,∞], atomic registers, initially T

$CA_1, CA_2, \ldots$ a series of commit-adopt instances

Upon propose(v) by process p_i:

$v_i := v$
r := 0
repeat forever
    r++
    $(c, v_i) := CA_r(v_i)$ // r-th instance of commit-adopt
    if c=true then
        D[r]:=v_i // let the others learn your value
        return v_i
    repeat
        if Ω outputs p_i then
            D[r]:= vi // advertise your value if leader
    until D[r]=v' where v'≠T //wait until the leader writes its value
    v_i := v' //adopt the leader's value

Commit-adopt: correctness

- Validity: immediate from validity of CA
- Agreement: by CA-agreement, if a process decides, everybody adopts it
- Termination: some correct process never decides, but then no process ever decides
  - ✓Consider a round r in which every process elects the same leader
  - ✓The same value is adopted by every process

Many details skipped: HW for the complete proof

Test&Set atomic objects

Exports one operation test&set() that returns a value in {0,1}

The first atomic operation on a T&S object returns 1, all other operations return 0

2-process consensus with T&S

Shared objects:

T&S TS

Atomic registers R[0] and R[1]

Upon propose(v) by process p_i (i=0,1):

```
R[i] := v
if TS.test&set()=1 then
        return R[i]
else
        return R[1-i]
```

3-process consensus with T&S?

Assume A solves consensus among three-processes using registers and T&S objects

Consider the *critical bivalent* run R of A: every one-step extension of R is univalent (HW: show that it exists)

W.L.O.G., assume that

- $R.p_0$ is 0-valent
- $R.p_1$ is 1-valent

We establish a case where some process cannot distinguish a 0-valent state from a 1-valent one

Three or more with T&S

- If p_0 and p_1 access different objects in R, or p_0 and p_1 access the same *register* in R, then we come back to the read-write case
- Suppose p_0 and p_1 access the same T&S object
  - ✓ p_2 cannot distinguish $R.p_0$ and $R.p_1$ in a solo extension
    => p_2 can never decide

=> T&S and registers cannot solve 3-process consensus (in a wait-free manner)

FIFO Queues

Exports two operations enqueue() and dequeue()

- enqueue(v) adds v to the end of the queue
- dequeue() returns the first element in the queue

  (LIFO queue returns the last element)

2-process consensus with queues

Shared:

Queue Q, initialized (winner,loser)

Atomic registers R[0] and R[1]

Upon propose(v) by process p_i (i=0,1):

```
R[i] := v
if Q.dequeue()=winner then
        return R[i]
else
        return R[1-i]
```

So far…

- 2-process consensus cannot be solved using registers
- N-process consensus can be solved using registers and Ω
- 2-process consensus can be solved using registers and T&S or queues (but not 3-process consensus)

Why consensus is interesting?

Because it is universal!

(can implement any object)

Homework: due May 31

- Prove the N-process safe agreement algorithm
- Prove the Ω-based consensus algorithm
- Three process consensus with queues and registers?
  - ✓Similar to the impossibility of 2-process consensus with registers

- No class next week, no office hours this week