Consensus

SLR206, P2

So far…

Shared-memory communication:

- safe bits => multi-valued atomic registers
- atomic registers => atomic snapshot

Today

Reaching agreement in shared memory:

- Consensus
  - ✓Impossibility of wait-free consensus
- 1-resilient consensus impossibility
- Universal construction

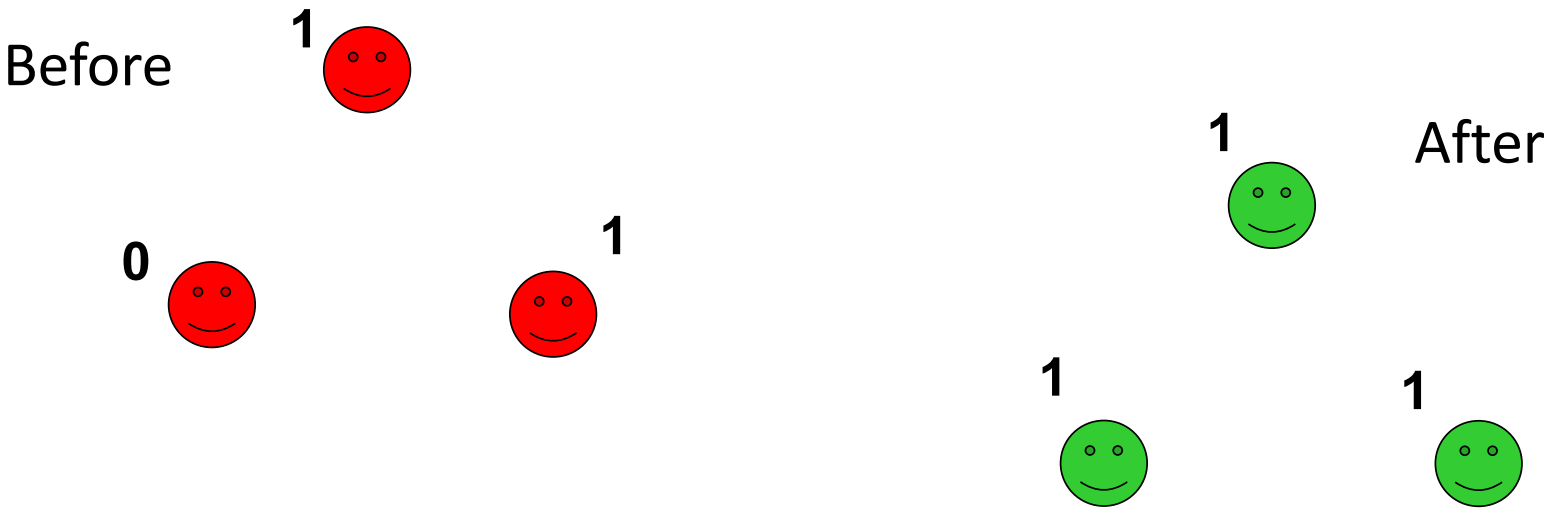
System model

- N *asynchronous* (no bounds on relative speeds) processes $p_0,\ldots,p_{N-1}$ ($N \geq 2$) communicate via atomic read-write registers

- Processes can fail by crashing
  - ✓A crashed process takes only finitely many steps (reads and writes)
  - ✓Up to t processes can crash: t-resilient system
  - ✓t=N-1: wait-free

Consensus

Processes *propose* values and must *agree* on a common decision value so that the decided value is a proposed value of some process



operational def

Consensus: definition

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

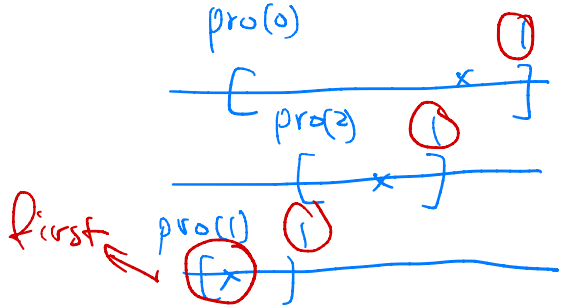
所有非失败的进程必须同意相同的值

- *Agreement:* No two processes decide on different values
- *Validity:* Every decided value is a proposed value 同意的值必须是提出的值之一
- *Termination:* No process takes infinitely many steps without deciding
  (Every correct process decides) 所有非失败进程最终必须同意一个值

$O=\{propose(v),\ v\in V\}$

(q_0, Q, O, R, σ) ⟵ sequential def

$q_0 = 1$ $Q = V\cup\{\perp\}$ Response: $R=V$



Th 0. An algorithm solve consensus iff it is linearizable wrt its seq. spec.

Optimistic (0-resilient) consensus

Consider the case t=0, no process fails → 节点之间无故障

Shared: 1WNR register D, initially T (default value not in V)

Upon propose(v) by process p_i:

```
if i = 0 then D.write(v)        // if p0 decide on v
wait until D.read() ≠ T         // wait until p0 decides
return D
```

(every process decides on p_0's input)

Impossibility of wait-free consensus [FLP85,LA87]

Wait-free is impossible

Theorem 1 No wait-free algorithm solves consensus

We give the proof for N=2, assuming that

p_0 proposes 0 and p_1 proposes 1

Implies the claim for all N≥2

Proof of Theorem 1

- We show that no 2-process wait-free solution exists for iterated read-write memory: $R_k[0]$, $R_k[1]$
- Code for p_i in round k: write to $R_k[i]$ and read $R_k[1-i]$:

```
k:= 0
repeat
        k := k+1;
        Rk[i].write(vi);
        vi := [vi, Rk[1-i].read()];
until not decided(vi)
```

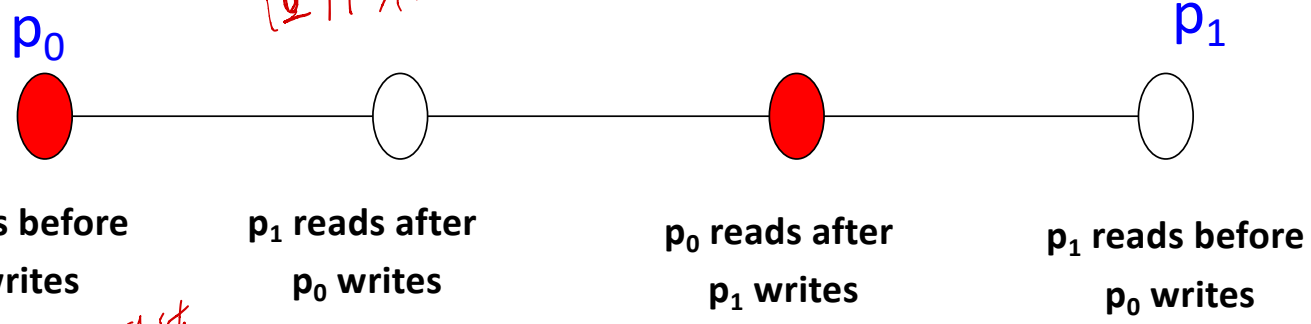
(until the current state does not map to a decision)

- The iterated memory is equivalent to non-iterated one for solving consensus

Proof of Theorem 1

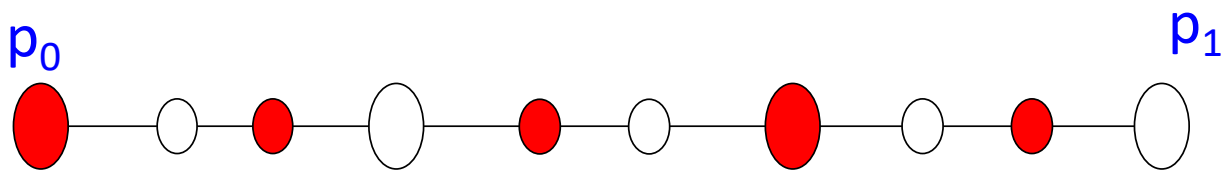
Initially each p_i only knows its input

One round of IIS:



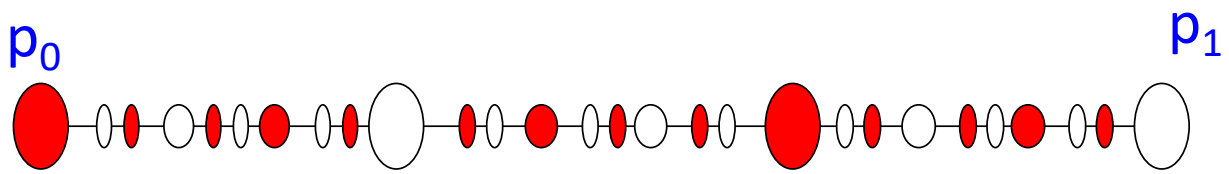
Proof sketch for Theorem 1

Two rounds:



Proof of Theorem 1

And so on…



每个进程只能独立运行 无法达成

Solo runs remain connected - no way to decide!

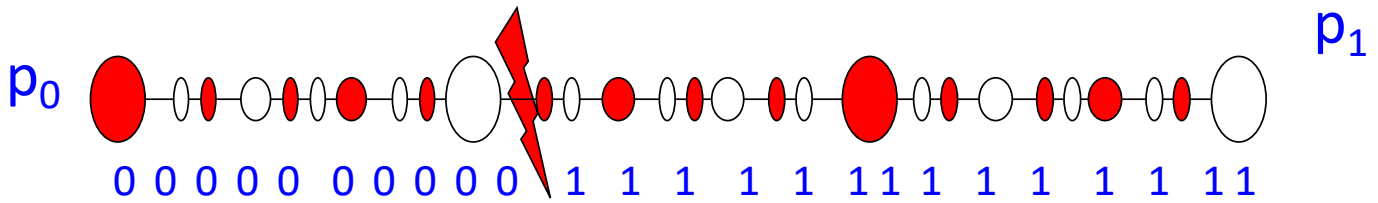
违反 agreement

Proof of Theorem 1

Suppose p_i (i=0,1) proposes i

- p_i must decide i in a solo run!

Suppose by round r every process decides



如果两个进程都能在独立运行时做出决定，那么就存在一个执行序列，其中一个进程看到另一个进程的决定并做出相反的决定。

There exists a run with conflicting decisions!

违反 agreement.

(n−1) − resilient ≅ wait free

1-resilient consensus? [FLP]

What if we have 1000000 processes and one of them can crash?

NO

We present a direct proof now
(an indirect proof by reduction to the wait-free impossibility also exists)

最多仅有一个进程失败

Impossibility of 1-resilient consensus [FLP85,LA87]

如果系统中的任何一个进程可能失败（例如因为崩溃），那么不能仅仅通过读写内存操作来保证所有非失败进程都能够达到一致的决定

Theorem 2 No 1-resilient (assuming that one process might fail) algorithm solves consensus in read-write

Proof

By contradiction, suppose that an algorithm A solves 1-resilient binary consensus among $p_0,\ldots p_{N-1}$

Proof

A run of A is a sequence of atomic *steps* (reads or writes) applied to the initial state

A run of A can be seen as an initial input configuration (one input per process) and a sequence of process ids $i_1,i_2,\ldots i_k,\ldots$ (all registers are atomic)

Every correct (taking sufficiently many steps) process decides!

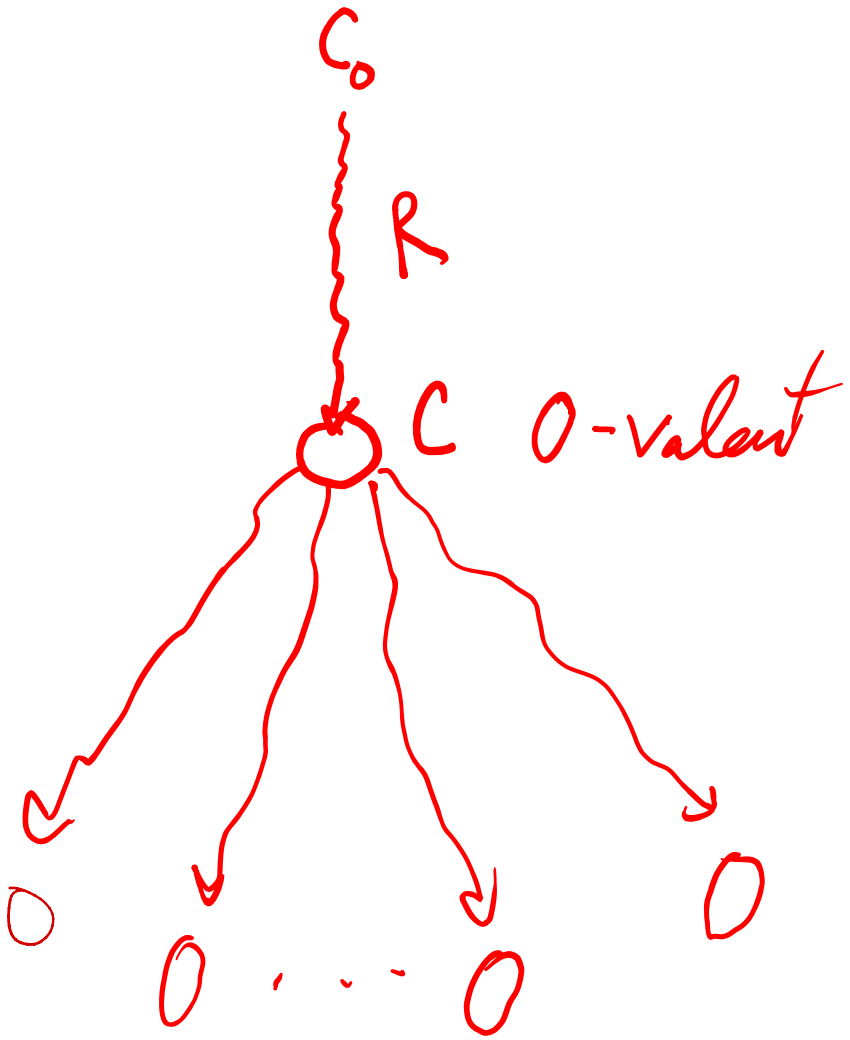
the possiblity of decision of n values.

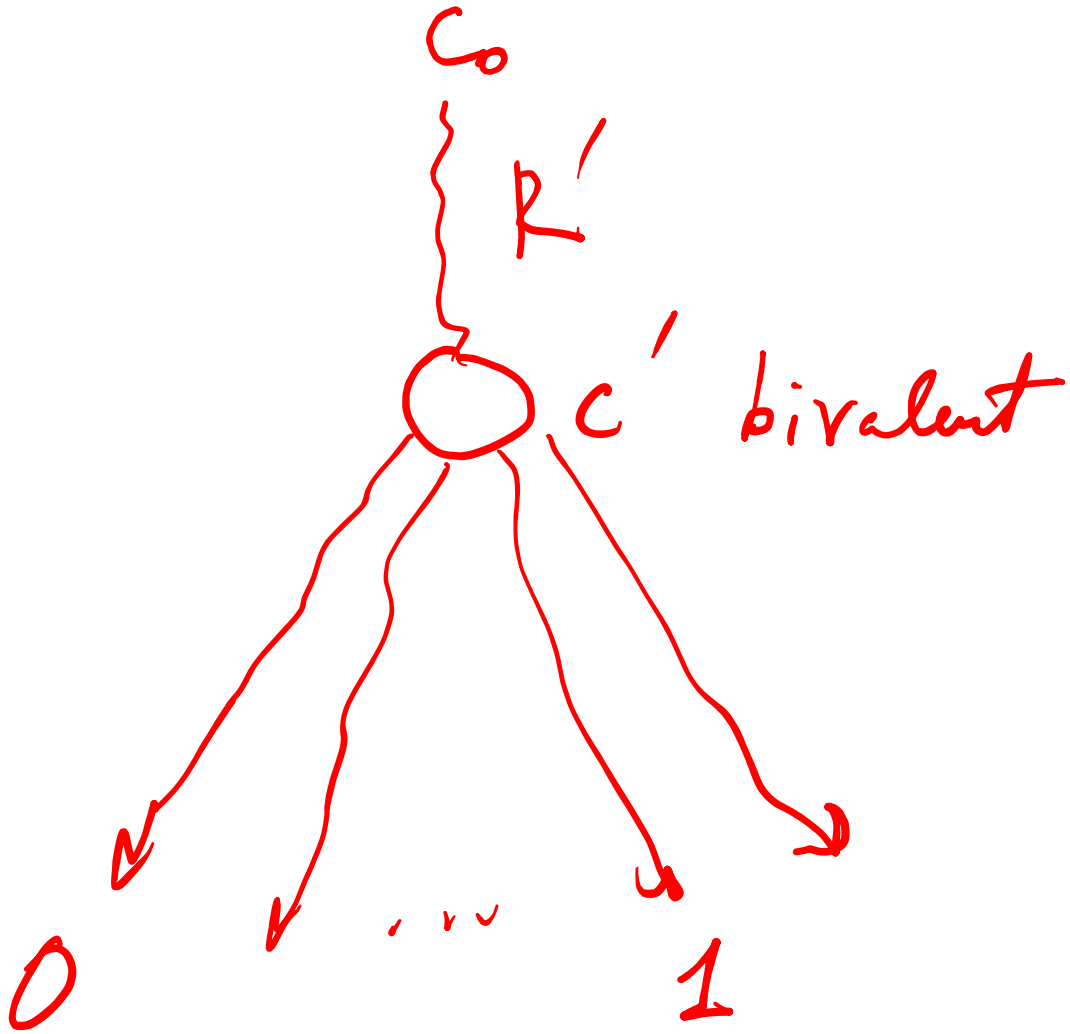
Proof: valence

Let R be a finite run (V=1) 1-valent : 决策集合可以是{0,1}. 但系统的所有决策是1

- We say that R is *v-valent* (for v in {0,1}) if (a) v is decided in some extension of R, and (b) 1-v is not decided in any extension of R

- We say that R is *bivalent* if for all v in {0,1}, there exists an extension of R in which v is decided.





Proof: valence claims

未做出决定

Claim 1 Every finite run is 0-valent, or 1-valent, or bivalent.
(by Termination)

Claim 2 Any run in which some process decides v is v-valent
(by Agreement)

Corollary 1: No process can decide in a bivalent run (by Agreement).

在一个bivalent运行中，系统仍然在两种可能的决定值之间“摇摆”——没有足够的信息来确保所有非故障进程能够做出一致的决定。如果在这种状态下某个进程做出了决定，那么这个决定只能是基于不完整的信息做出的，因此不能保证它会被所有其他进程接受。这与共识的定义相违背，即所有非故障进程都必须达成相同的决定值。

因此，如果一个运行是bivalent的，它必须继续执行下去直到它变成univalent，即要么变成0-valent或者1-valent，此时系统中的进程才能够安全地做出决策。这是FLP不可能性定理的核心部分，表明在一个异步系统中，如果存在故障可能性，那么系统可能会永远停留在bivalent状态，从而无法达成共识。

Bivalent input

Claim 3 There exists a bivalent input configuration (empty run)

Proof

Suppose not

Consider sequence of input configurations $C_0,\ldots,C_N$:

C_i: $p_0,..,p_{i-1}$ propose 1, and $p_i,..,p_{N-1}$ propose 0

- All C_i's are univalent
- C_0 is 0-valent (by Validity)
- C_N is 1-valent (by Validity)

Bivalent input

There exists i in {0,…N-1} such that C_i is 0-valent and C_{i+1} is 1-valent!

C_i and C_{i+1} differ only in the input value of p_i (it proposes 0 in C_i and 1 in C_{i+1})

Consider a run R starting from C_i in which p_i takes no steps (crashes initially): eventually all other processes decide 0

Consider R' that is like R except that it starts from C_{i+1}

- R and R' are indistinguishable!
- Thus, every process decides 0 in R' --- contradiction (C_{i+1} is 1-valent)

在您提供的材料中，**Claim 3** 是关于存在一个 bivalent 输入配置的声明。这个声明的证明基于反证法和连续性论证。这里提供了一个序列 $C_0, C_1, ..., C_N$ 的输入配置，其中 C_0 是所有进程都提出0的配置， C_N 是所有进程都提出1的配置。根据共识问题的有效性条件，C_0 必须是0-valent（因为如果决定了其他值将违反共识问题的有效性条件），而 C_N 必须是1-valent（同理）。

证明假设所有的配置都是 univalent，这意味着每个配置都将导致一个特定的决定值。但是，由于 C_0 和 C_N 的valence不同，必须存在一个点 i 在这个序列中，使得 C_i 是0-valent而 C_{i+1} 是1-valent。

考虑从 C_i 开始的运行 R，其中进程 p_i 崩溃并且不执行任何步骤。由于 C_i 是0-valent，所有其他进程最终决定0。现在考虑一个新的运行 R'，它从 C_{i+1} 开始，其他所有事情都与 R 相同（即进程 p_i 同样崩溃）。由于 C_i 和 C_{i+1} 只在 p_i 的输入值上有差异，且 p_i 在两个运行中都没有执行任何步骤，因此所有其他进程在 R 和 R' 中的观察结果应该是一样的，这意味着它们无法区分是从 C_i 还是 C_{i+1} 开始的运行。这就产生了一个矛盾，因为 C_{i+1} 是1-valent的，所以所有进程应该决定1。

这个矛盾表明，至少存在一个输入配置是 bivalent 的，即存在一个点 i，在 C_i 和 C_{i+1} 之间的配置是 bivalent 的，这意味着它既可能导致0也可能导致1的决定值，具体取决于进程的后续行为。这个概念在证明共识问题的不可能性结果中是核心的，因为它展示了在异步系统中无法保证所有运行都能达成一致的决策，尤其是在存在故障可能性时。

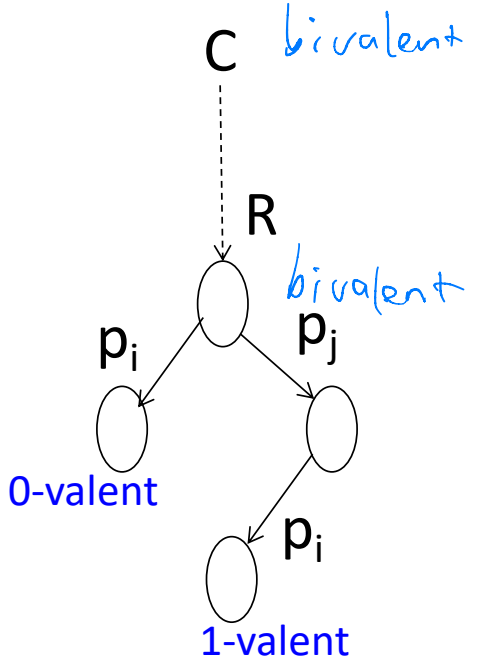
Critical run

Claim 4 There exists a finite run R and two processes p_i and p_j such that R.i is 0-valent and R.j.i is 1-valent (or vice versa)

(R is called critical)

Proof of Claim 4: By construction, take the bivalent empty run C (by Claim 3 it exists)

We construct an ever-extending fair (giving each process enough steps) run which results in R



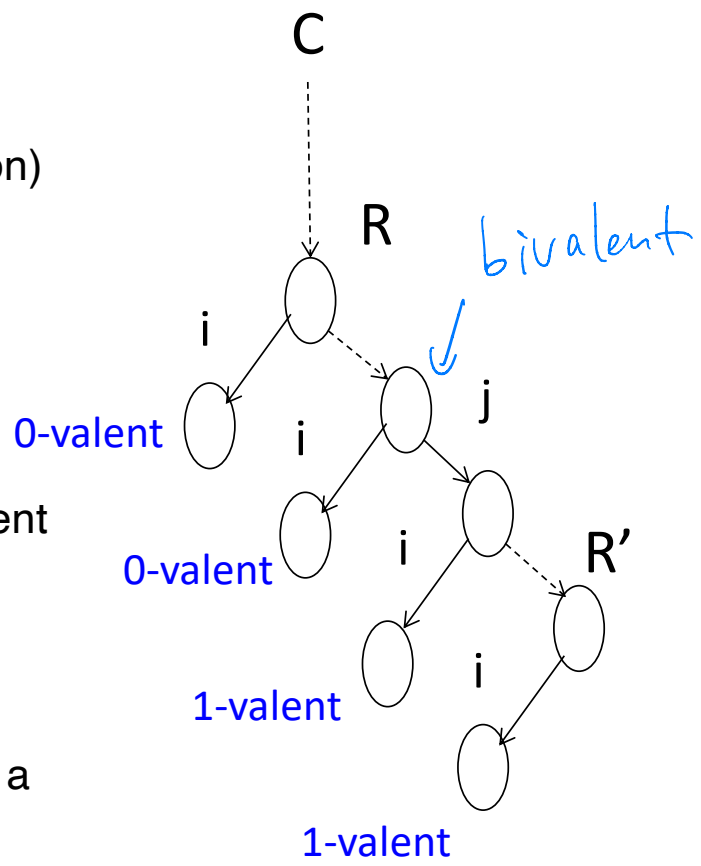
Proof of Claim 4: critical run

repeat forever

take the next process p_i (in round-robin fashion)

if for some R', an extension of R, R'.i is bivalent **then** R:=R'.i

else stop

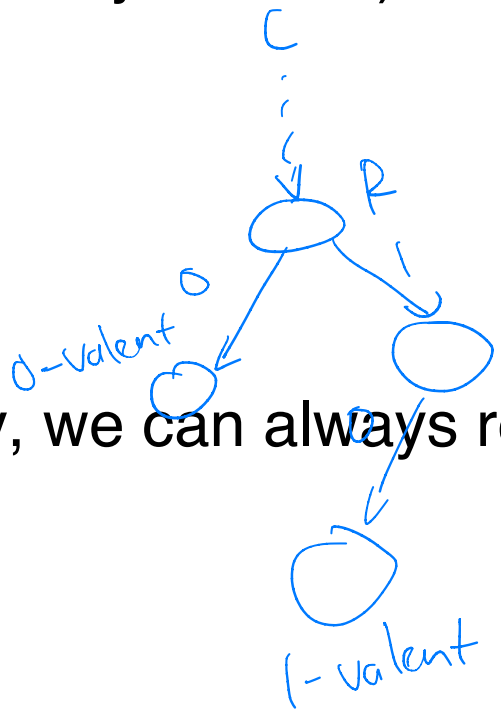


- If never stops – ever extending (infinite) bivalent runs in which every process is correct (takes infinitely many steps – contradiction with termination
- If stops – (suppose R.i is 0-valent) – consider a 1-valent extension
  - ✓ There is a critical configuration between R and R'

Proof (contd.)

Take a critical run R (exists by Claim 4) such that:

- R.0 is 0-valent
- R.1.0 is 1-valent



(without loss of generality, we can always rename processes or inputs appropriately ☺)

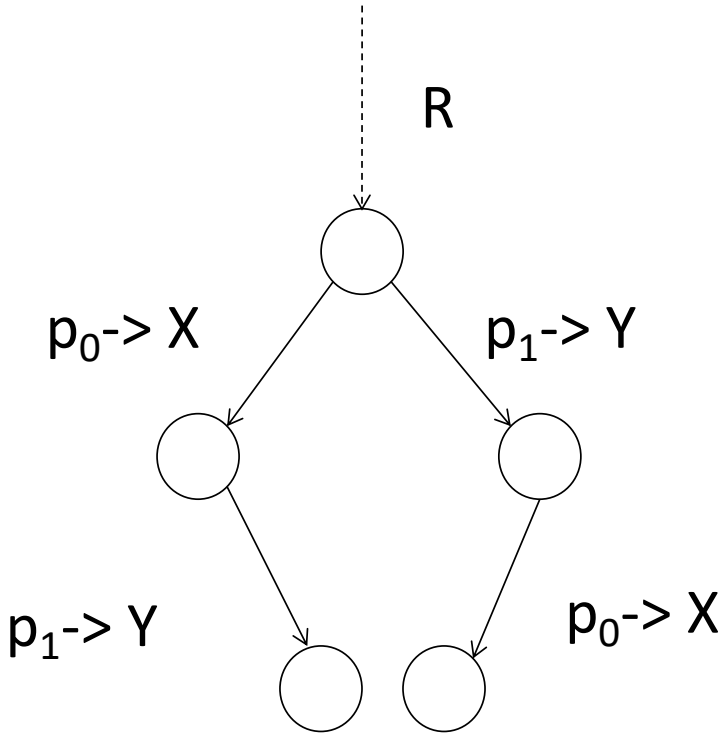
Proof (contd.): the next steps in R

Three cases, depending on the next steps of p_0 and p_1 in R

- p_0 and p_1 are about to access different objects in R
- p_1 reads X and p_0 reads X
- p_0 or p_1 writes in X

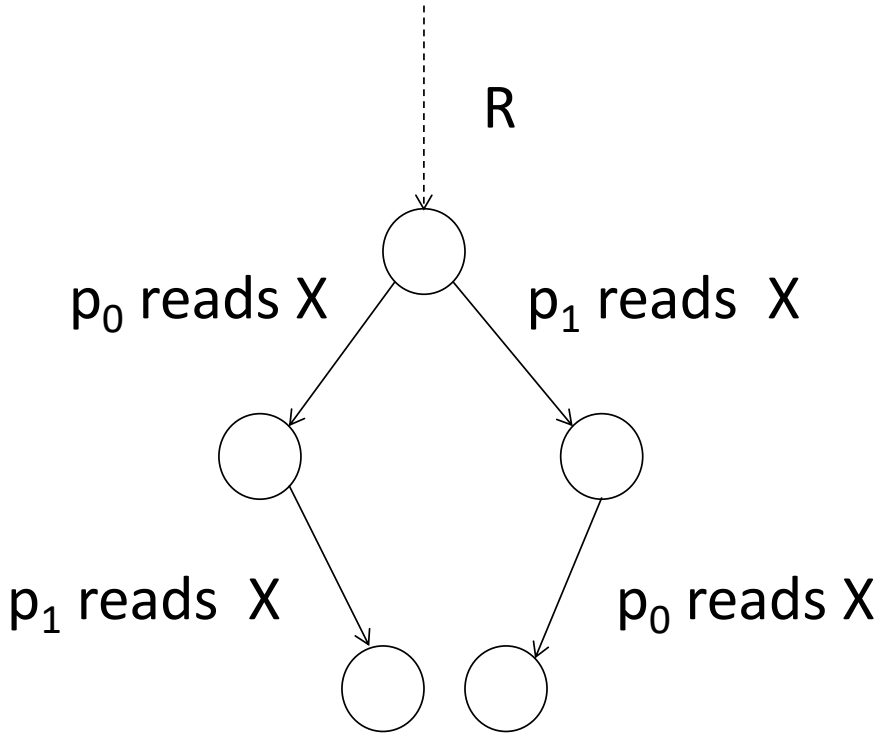
Proof (contd.): cases and contradiction

- p_0 and p_1 are about to access different objects in R
  - ✓R.0.1 and R.1.0 are indistinguishable



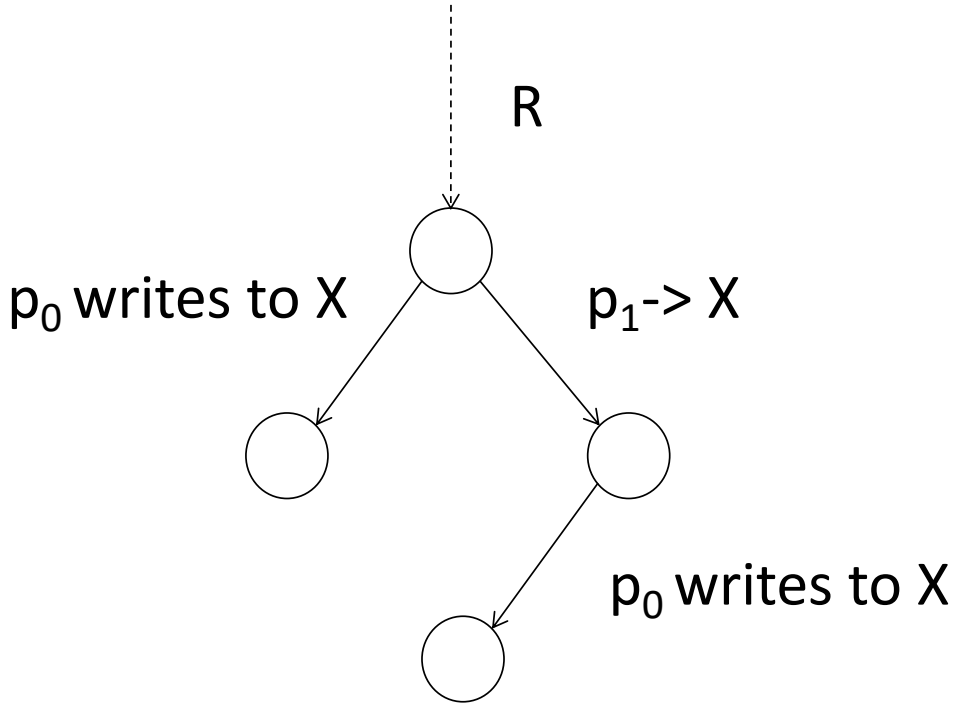
Proof (contd.): cases and contradiction

- p_0 and p_1 are about to read the same object X
  R.0.1 and R.1.0 are indistinguishable



Proof (contd.): cases and contradiction

- p_0 is about to write to X (the case when p_1 writes is symmetric)
  - ✓Extensions of R.0 and R.1.0 are indistinguishable for all except p_1 (assuming p_1 takes no more steps)



Thus

- No critical run exists
- A contradiction with **Claim 4**

$\Rightarrow$ 1-resilient consensus is impossible in read-write

Th 3. $\forall k \geq 2$, $(k+1)$ processes wait-free simulate n-process k-resilient system $(n \geq k)$.

BG-simulation

$\forall$ correct proc decides, if $\forall$ participant take enough steps

Safe-agreement | agreement, validity, safe-agreement termination

(cf. page 36)

在分布式系统中如何绕过共识问题的不可能性。由于FLP不可能性定理表明，在异步系统中，如果至少有一个进程可能失败，那么就不可能达成共识。

Circumventing the impossibility

- Strengthening the communication primitives (already discussed)
  - ✓Adding stronger objects: T&S, counters, queues, C&S, LL/SC
- Strengthening synchrony assumptions
  - ✓Adding failure detectors, e.g., Ω
- Relaxing liveness: safe agreement, obstruction-freedom, termination with probability 1

加强通信原语：在分布式系统中，基本的通信原语是读和写操作。通过增加更强大的同步原语，如测试和设置（T&S）、计数器、队列、比较和交换（C&S）、负载链接/存储条件（LL/SC），系统可以更有效地协调不同进程之间的操作。

加强同步假设：这意味着系统不再是完全异步的。引入故障检测器（如 Ω 故障检测器）可以让系统对进程的失败有更多的信息，从而可能允许系统在某些进程失败时继续达成共识。

放宽活性要求：这意味着我们不再要求系统总是能够达成决策，而是仅在某些条件下才能达成决策。例如，安全协议可能在没有进程失败时保证达成共识，或者在没有进程被无限阻塞的情况下保证达成共识（即obstruction-freedom）。此外，系统也可以设计成最终总能达成共识，但不保证具体何时能达成（即概率1的终止）。

Recall the 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

Recall 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)

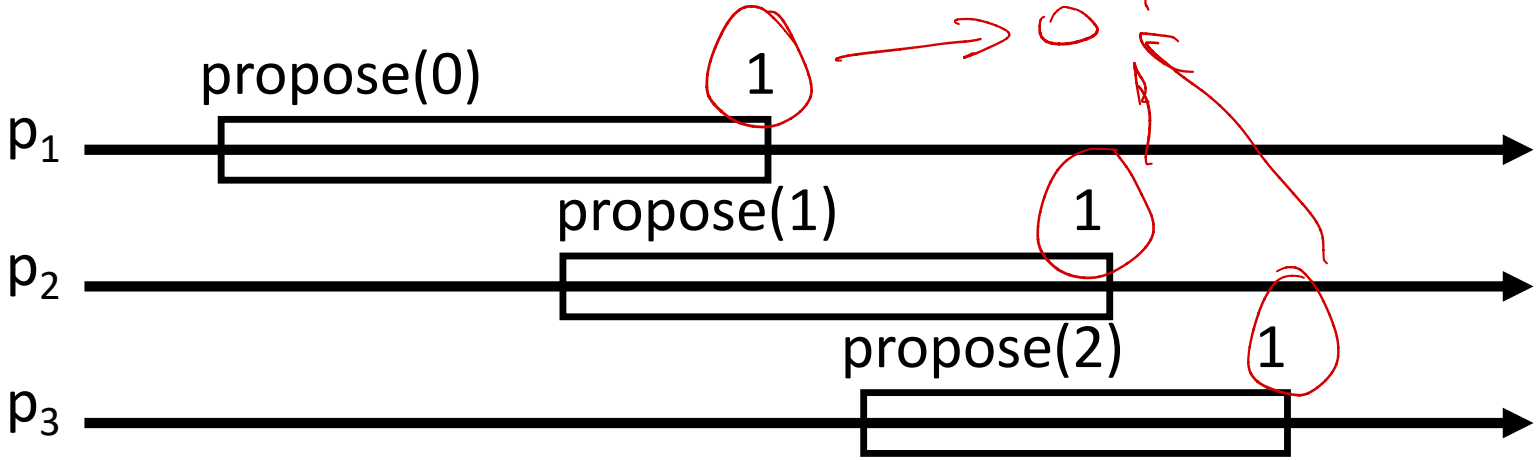
Consensus: sequential spec

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

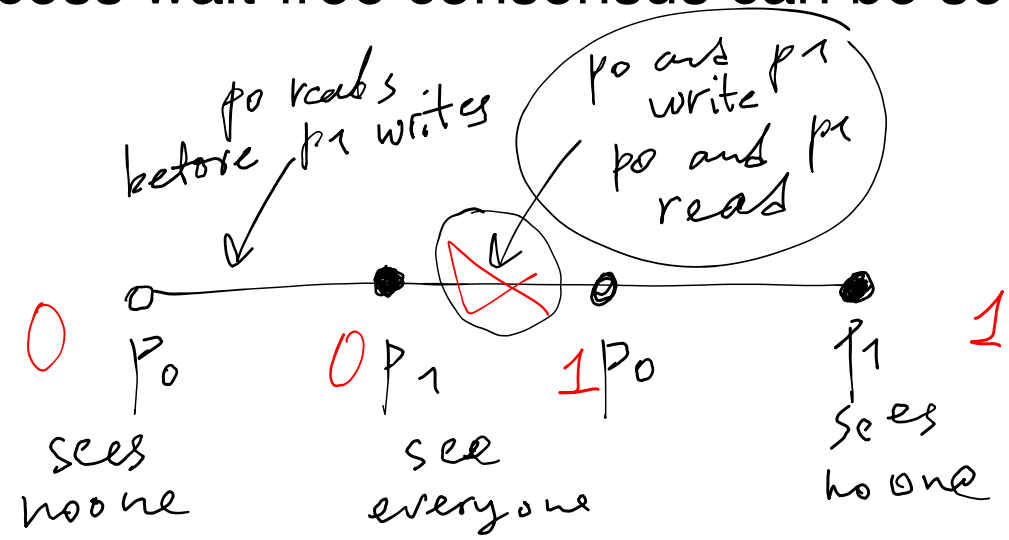
There is an algorithm A that in every run of A in M satisfies Agreement, Validity, and Termination

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
  - ✓If not, 2-process wait-free consensus can be solved



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

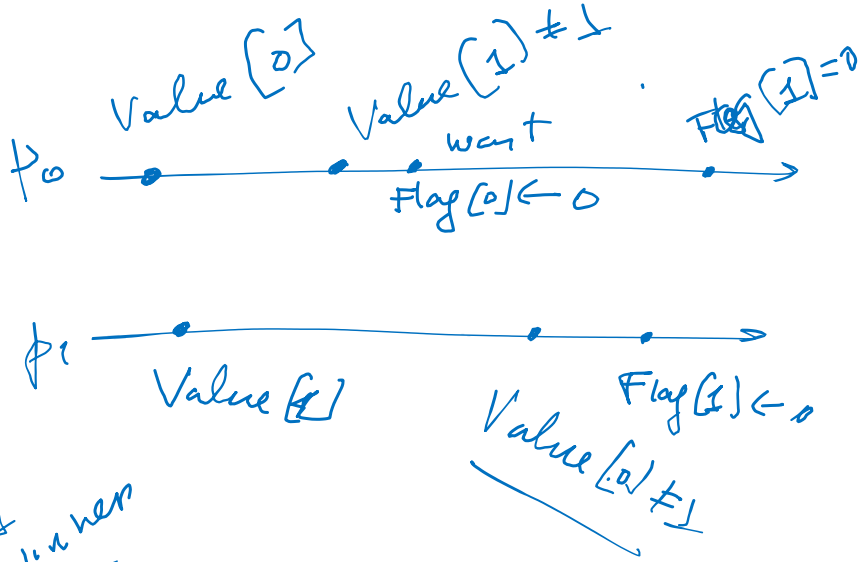
Two process safe agreement

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

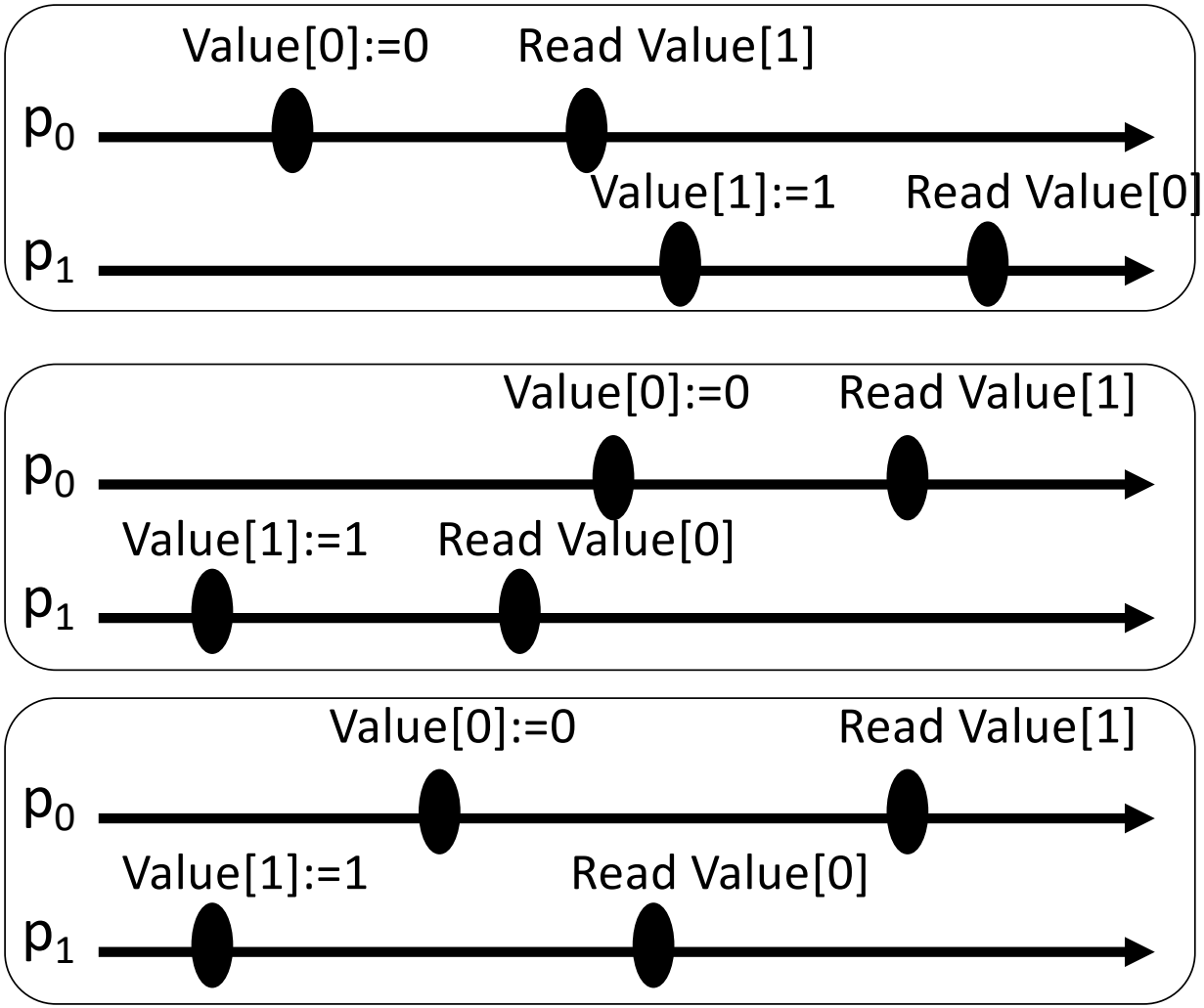
Upon propose(v) of process p_i:

```
Value[i] := v
x := Value[1-i]
if x = T then
        Flag[i] := 1
        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
```

// p_i is the winner



Correctness: at most one winner



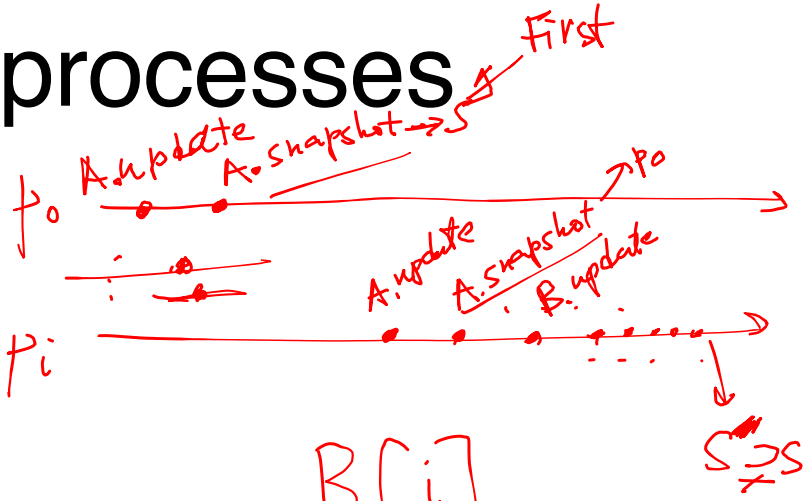
p_0 is the only winner

p_1 is the only winner

No winners

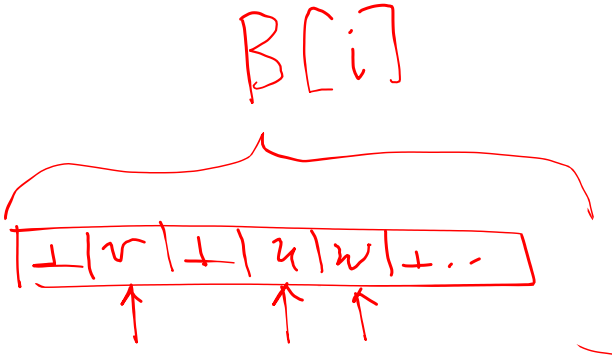
Safe agreement for N processes

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:=A.snapshot()
B.update$_i$(U)
repeat
V:=B.snapshot()
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



The outcome of the first snapshot on A

终止条件是指对于在 U 中不是 T 的所有位置 j，在 V 中的相应位置也不是 T。一旦这个条件被满足，每个进程将决定在 V 中非 T 值的最小输入。

这个终止条件确保了以下几点：

所有的进程在决定之前至少参与了一轮更新（因为它们都在 A 中留下了自己的值）。

每个进程至少完成了一次快照操作，这意味着它们至少看到了其他进程的一些提案值。

当决定被做出时，它基于的是一组已知的提案值的快照，这个快照是由进程们在 A 中执行的更新操作形成的。

如何实现安全协议：

该算法能够实现安全协议，因为它满足了以下两个关键的共识属性：

- **协议（Agreement）**：由于所有进程都决定了 `V` 中非 `T` 值的最小输入，所以它们都会做出相同的决策。
- **有效性（Validity）**：由于进程只会决定实际被提议的值，所以决策是有效的。

此外，算法的设计确保了即使在有进程失败的情况下，非故障进程仍然能够达成一致。原子快照对象的使用保证了在进程读取和写入值时的一致性，即使这些操作是并发执行的。因此，该算法在满足安全共识的基本要求的同时，也克服了异步系统中共识的不可能性。

Safe agreement: correctness

Liveness: immediate

Safety (intuition):

Consider p_t that took the smallest snapshot in A

- ✓ 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

Quiz 7.1: safe agreement

- Complete the proof of correctness of safe agreement
- Show that any 2-process safe agreement algorithm and registers and be used to solve mutual exclusion, and vice versa. Is it true for N>2 processes?
- Would the algorithm be correct if N atomic registers are used instead of an atomic snapshot object?

Circumventing consensus impossibility: commit-adopt

A variant of consensus with weaker safety (relaxed agreement)

Can be used for solving consensus with an oracle

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 是一种在分布式系统中用于达成共识的算法，它相对于标准共识算法有一个更弱的安全保证。这个算法中，每个进程提议一个值，并最终决定一个二元组（c, v），其中 c 是一个布尔值，v 是提议值的集合中的一个值。在这个上下文中：

如果 c 是 false，那么进程被认为是采纳（adopt）了值 v。
如果 c 是 true，那么进程被认为是提交（commit）了值 v。

Commit-adopt 算法的关键是**允许进程在没有足够信息来提交一个值时先采纳一个值**。如果一个进程采纳了一个值，它仍然可以在将来基于更多信息改变它的决策。但是，如果一个进程提交了一个值，它就必须坚持这个决策，并且不能更改。

Commit-adopt: properties

- *Validity:* Every adopted value is an input value of some process
- *Termination:* Every correct process decides (commits or adopts)
- *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*)

如果所有进程的输入一致，那么它们最终的决定也会一致，并且是提交（true）的状态。

Standard-Agreement: No two process decide on different values

这里的差别在于，CA–Agreement 允许存在一个“采纳但不提交”的中间状态。这意味着进程可以暂时持有一个值，而不是最终决定它。

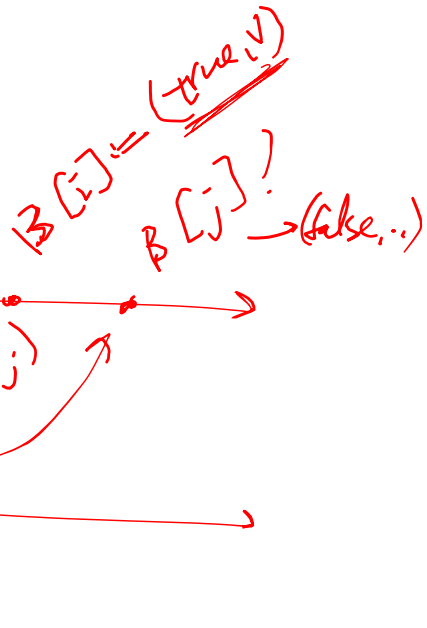
Commit-adopt : protocol

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
U := read A[0,…,N-1]
if all non-T values in U are v then
    B[i] := (true,v_i)
else
    B[i] := (false,v_i)
U := read B[0,…,N-1]
if all non-T values in U are (true,*) then
    return (true,v_i)
else if U contains (true,v') then
    v_i := v'
return (false,v_i)



Register A：用于每个进程初次提议它的值。这是进程表达其初始意向的地方，而其他进程可以观察到这一点。

Register B：用于进程在观察到其他所有进程的提议后，提交或采纳值。如果一个进程在 B 中看到所有非 T 的值都是相同的，并且这些值都是提交（true），它就知道可以安全地提交该值。如果它看到一个提交的值（true, v'），则它将采纳这个值。

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: 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

Quiz 7.2: commit-adopt

- Would the CA algorithm is correct if regular registers were used?
- Give an obstruction-free consensus algorithm using CA
  - ✓Obstruction-freedom: every process that runs solo from some point on eventually decides

Next

- BG simulation
- Combining registers with stronger objects
  - ✓Consensus and test-and-set (T&S)
  - ✓Consensus and queues
- Universality of consensus
  - ✓Consensus can be used to implement any object

BG agreement

SLR, period 2

So far…

In read-write

- Wait-free set agreement
- 1-resilient 1-set agreement (consensus) is impossible

What about k-resilient k-set agreement?

No: otherwise k+1 processes can (wait-free) solve it

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

Safe agreement: blocked vs. resolved

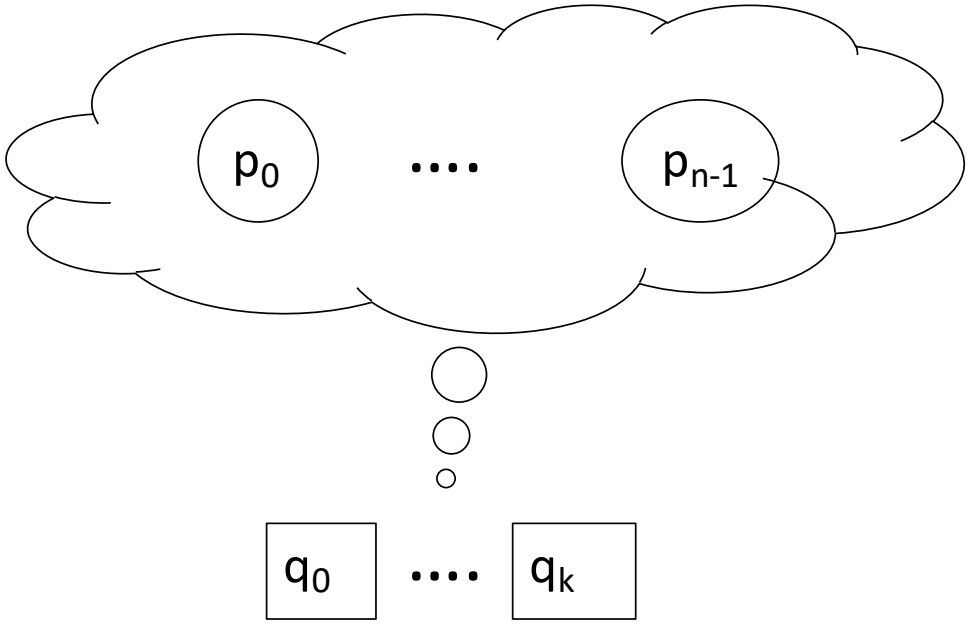
Shared: atomic snapshot objects
A[0,…,N-1], B[0,…,N-1], all initially T

Upon propose(v) by process p_i:

A.update$_i$(v)
U:=A.snapshot()
B.update$_i$(U)
repeat
V:=B.snapshot() //Blocked
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 //Resolved

BG simulation [Borowsky-Gafni, 1993]

- k+1 *simulators* $q_0,\ldots,q_k$
- n simulated processes $p_0,..,p_{n-1}$ (n>k)



BG simulation: the idea

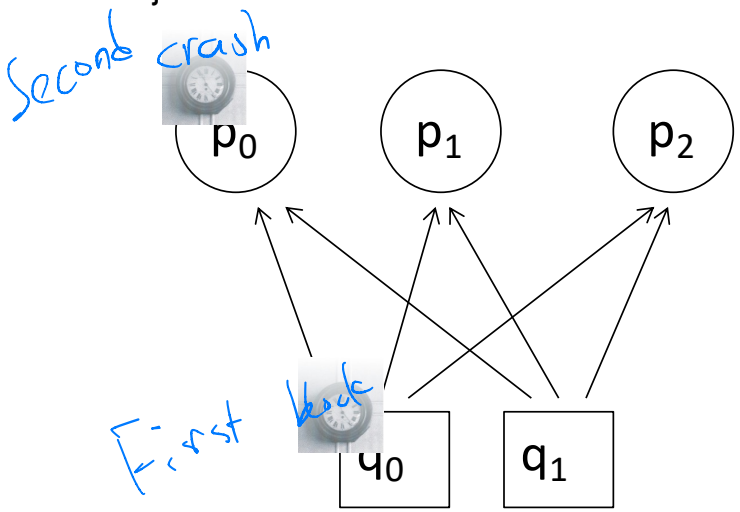
WLOG, each simulated p_j runs the full-information protocol

repeat forever:
  $update_j(view_j)$
  $view_j$:= snapshot()

(If 1-resilient consensus is solvable with atomic snapshots, it is solvable with 1WNR registers)

Simulation order

- Run the BG agreements in round-robin: every simulator tries to simulate every process
- Every simulated step is agreed upon (using safe agreement)
- When done with the “write-phase” of safe agreement for a step of p_j – proceed to $p_{j+1 \bmod n}$



$p_0p_1p_2p_0p_1p_2p_1p_2p_1p_2,\ldots$

Simulation: code for each q_i

```
Initially last[j]=0, 0≤j≤n-1        // the last step of p_j
                                    // simulated by q_i
j:=0
repeat forever                      (last step.
   if resolved(j,last[j]) then      // return true for last[j]=0
        last[j]++; k := last[j]     // start simulating the next step
   v := getState(j,k-1)             // get the view of p_j
   if p_j decides x in v then return x  // terminate
   run SafeAgreement_{j,k}(v) until resolved or blocked
   if resolved(j,k) with x then publish(j,k,x)
   j := (j +1) mod n
```

Simulation: code for each q_i

```
Initially state[j]=initial, 0≤j≤n-1   // the latest state of p_j
                                      // observed by q_i

publish(j,k,x)          // announce state of p_j
   state[j] := (k,x)        // state of p_j after step k (seen by q_i)
   S.update_i(state)

getState(j,k)
   snap := S.snapshot()   // get the "most recent" view of all
   for each s=1,…,n do
        view[s] := the most recent view of p_s in snap
   if k=0 then view[j]:=input_i  // use q_i's input for p_j
   return view
```

Safety

Claim 1 Each simulated view are agreed upon => no view k of p_j can be seen differently by different simulators

Claim 2 There exists a run R on $p_0,\ldots,p_{n-1}$, such that the simulated run looks like R to each p_j

- p_j goes through the views of R

All proposed views come from snapshots: all snapshots are totally-ordered and include the "latest" updates

- Safe:
  - each simulated view is agreed upon
- Live?
  - No! What if an instance of BG-agreement blocks?

Liveness

Claim 3 In an infinite simulation, at most k processes fail in the resulting R

- A faulty simulator may block at most one process
- At most k simulators fail
- At most k simulated processes fail – a k-resilient run!

k-resilient k-set agreement is impossible

Theorem No k-resilient algorithm solves k-set agreement for $n \geq k+1$ processes, assuming that wait-free (k+1)-process k-set agreement is impossible

Proof Suppose not: there is an algorithm A that solves k-set agreement among $p_0,\ldots,p_{n-1}$ where at most k can fail

Then $q_0,\ldots, q_k$ can solve k-set agreement by simulating A

- Each q_i proposes its input for each simulated p_j
- Each q_i decides on the first value returned by A

k-set agreement (k-set consensus)

Inputs from V (|V|>=k+1), outputs in V

- Validity: every output is an input of some process
- k-Agreement: at most k distinct outputs
- Termination: every correct process outputs

k=1 is consensus

Correctness

- Safety: all decisions come from a run of a k-set agreement algorithm A
  - At most k different values decided
  - Every proposed value is an input of some simulator – every decided value is a proposed value
- Liveness: if a simulator is correct – the simulated run of A is k-resilient, and eventually every correct simulated process decides (one is enough)