TDC1:
Consensus and BG Agreement

So far

Shared memory transformations

- safe bits -> multi-valued atomic registers

- Atomic registers -> atomic snapshots
- safe bit -> atomic bit (with constant overhead)

Today

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

System model

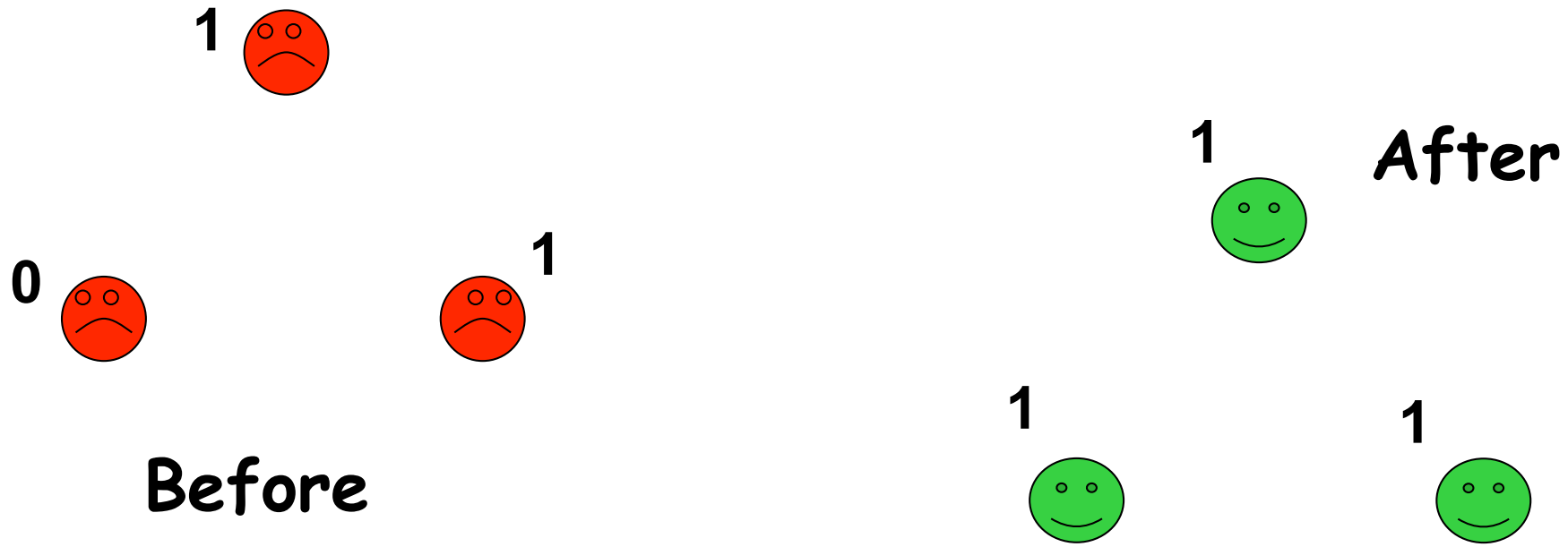
- N *asynchronous* processes $p_0,\ldots,p_{N-1}$ ($N \geq 2$) communicate via reading and writing in the shared memory

- 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

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



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)

Solving consensus

An algorithm A (a collection of automata $A_0,\ldots,A_{N-1}$) solves consensus in a given model M if every run of A in M satisfies Agreement, Validity, and Termination

A wait-free algorithm solves consensus assuming up to N-1 processes can crash

Optimistic consensus

Consider the case t=0, no process fails

Upon propose(v) by process p_i:

if i = 0 then D.write(v)

wait until D.read() ≠ T (default value not in V)

return D

(every process decides on p_0's input)

Impossibility of wait-free consensus [FLP85]

Theorem 1 No wait-free algorithm solves consensus for N=2

We give the proof for N=2, assuming that

p_0 proposes 0 and p_1 proposes 1

If no wait-free algorithm solves consensus for this case, Theorem 1 holds for any N≥2

Proof

By contradiction, suppose that an algorithm A solves wait-free consensus among p_0 (proposing 0) and p_1 (proposing 1)

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 a sequence of process ids:

0000,…

0101,…

1111,…

In every infinite run, every correct process decides!

Proof: valence

Let R be a finite run in Y

- We say that R is *v-valent* (for v in {0,1}) if v is decided in every infinite extension of R

- We say that R is *bivalent* if R is neither 0-valent nor 1-valent

Proof: valence claims

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.

Claim 3 The empty run (in Y) is bivalent.
(by Validity)

Proof (contd.): critical run

Claim 4 There exists a bivalent finite run R such that every extension of R is *univalent* (0-valent or 1-valent)

Proof of Claim 4: By contradiction, suppose every bivalent run has a bivalent extension

- Take the bivalent (by Claim 3) empty run
- We can extend it to a bivalent run =>
- Inductively, every bivalent run can be extended into a bivalent run =>
- We obtain an infinite run of A in which no process decides (by Corollary 1) =>
- Termination is violated!

Proof (contd.)

Consider a bivalent run R such that every one-step extension of R is 0-valent or 1-valent

W.L.O.G., assume that

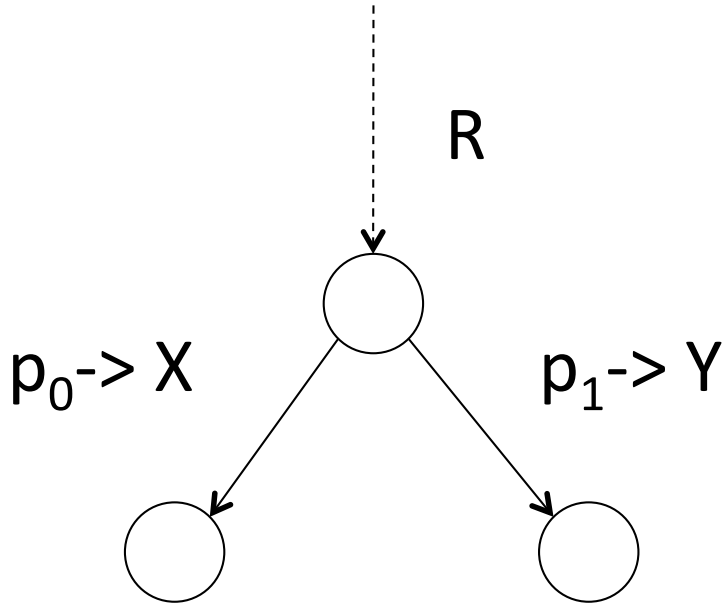
- R.0 is 0-valent
- R.1 is 1-valent

Proof (contd.): cases and contradiction

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

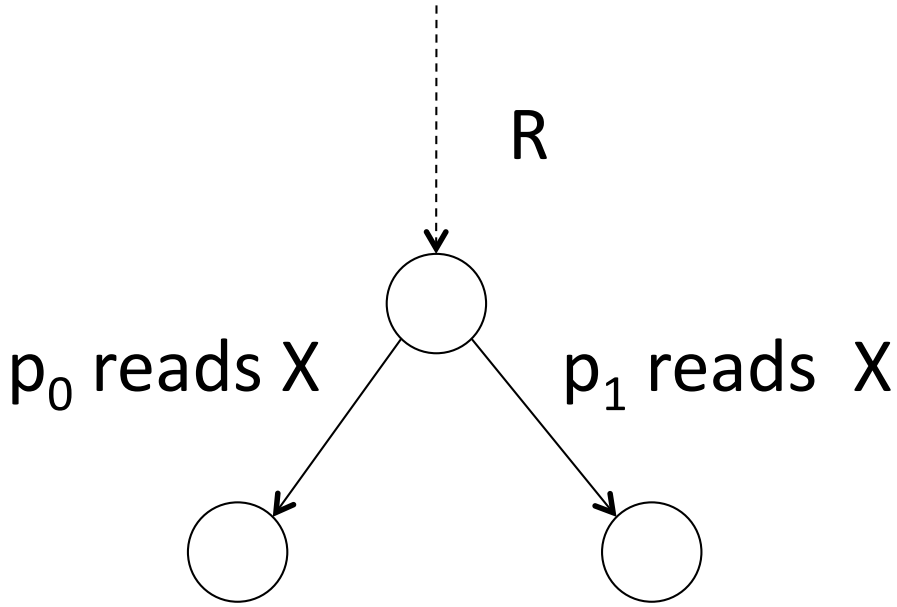
Proof (contd.): cases and contradiction

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



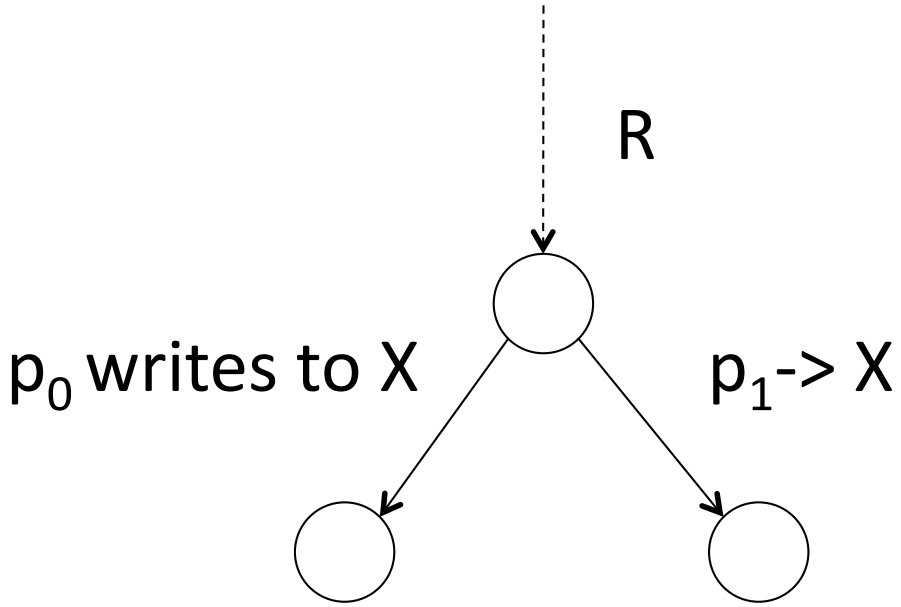
Proof (contd.): cases and contradiction

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



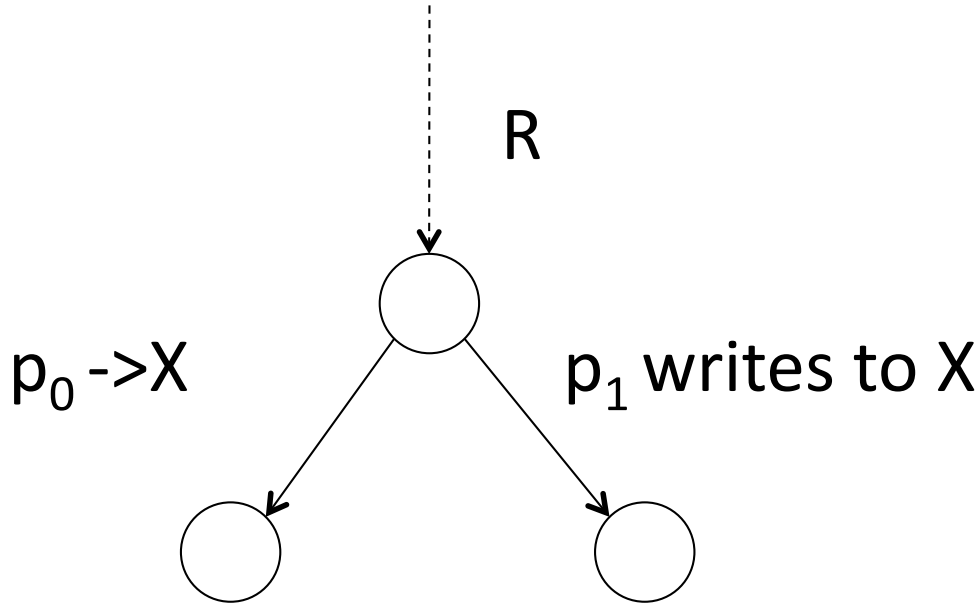
Proof (contd.): cases and contradiction

- p_0 is about to write to X
  - ✓Extensions of R.0 and R.1.0 are indistinguishable for p_0 (if p_1 crashes)



Proof (contd.): cases and contradiction

- p_0 is about to write to X
  - ✓Extensions of R.0.1 and R.1 are indistinguishable for p_1 (if p_0 crashes)



So…

- A bivalent R such that R.0 and R.1 are both univalent does not exist
- A contradiction with **Claim 4**

=> 2-process wait-free consensus is impossible

$\Rightarrow$ wait-free consensus is impossible

A topological approach [HS93]

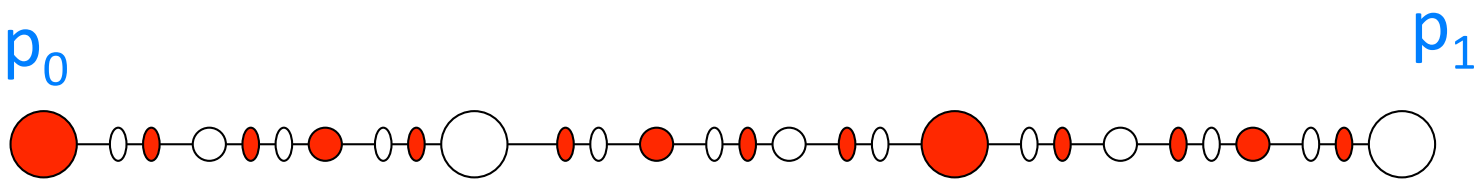
Iterated memory: use “fresh“ memory in each round

while not done

write(view) in $Memory_K$

view := snapshot($Memory_K$)

k := k+1



p_0 reads before p_1 writes

p_1 reads after p_0 writes

p_0 reads after p_1 writes

p_1 reads before p_0 writes

Solo runs remain connected

(remains to show that the iterated model is not weaker)

1-resilient consensus

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

NO

If we could, then we would solve 2-process wait-free consensus by simulation

Today

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

What if no participant fails?

- Suppose that every process either takes no steps (does not participate) or takes enough steps (three is enough)
- Can we solve consensus among p_0 and p_1?

Yes we can!

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
```

Safe agreement: correctness

Liveness:

- ✓Suppose each participant takes 3 shared-memory steps: every wait terminates

(If a participant "dies" between the writes – block)

Safety:

- At most one process wins!
- If nobody wins – decide on the value of p_0

HW: What about any N≥2?

Blocked vs. resolved

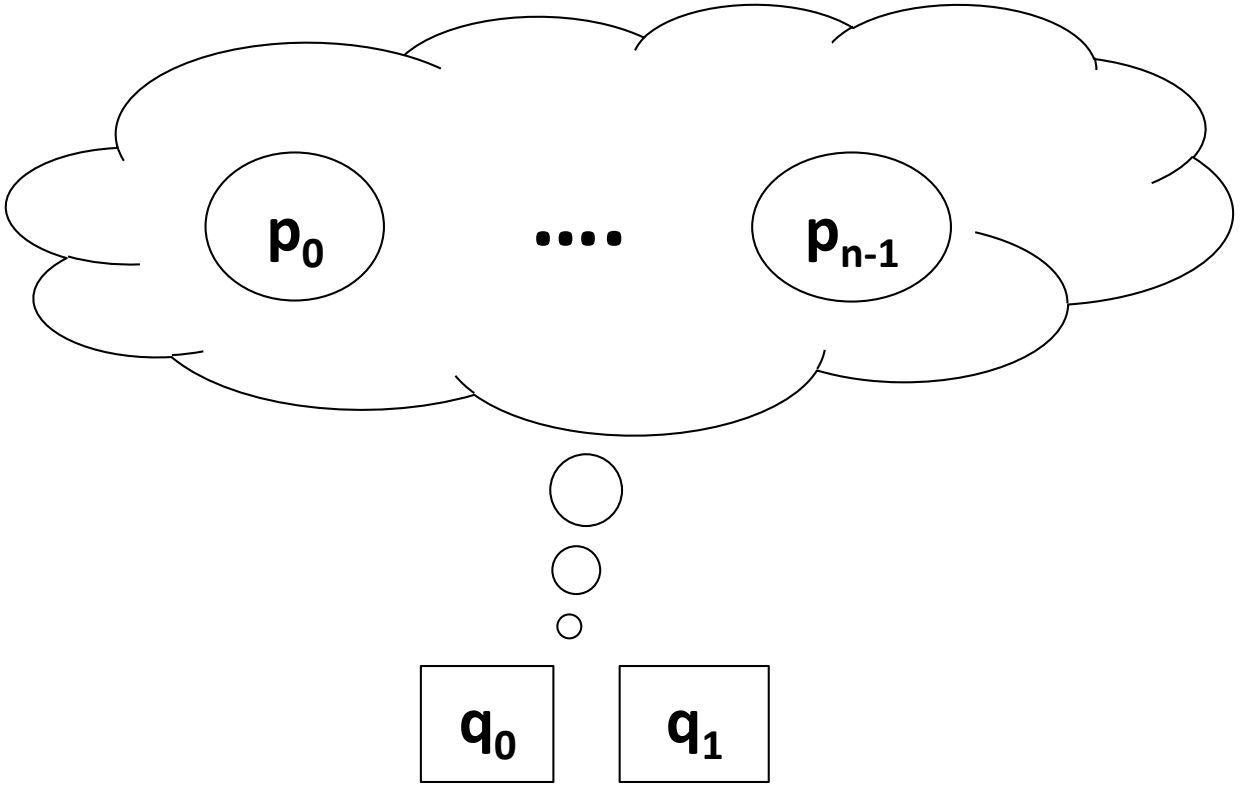
Shared: Flag[0,1], initially T; Value[0,1], initially T

Upon propose(v) of process p_i:

```
Value[i] := v
x := Value[1-i]
if x = T then
        Flag[i] := 1
        return(v)              // Resolved with v
Flag[i] := 0
wait until Flag[1-i] ≠ T       // Blocked
if Flag[1-i] = 1 then
        return x               // Resolved with x
else
        return Value[0]        //Resolved with Value[0]
```

BG simulation [BG93]

- 2 *simulators* q_0,q_1
- n simulated processes p_0,..,p_{n-1} (n≥2)



BG simulation: the idea

WLOG, each simulated p_j runs the full-information protocol

repeat forever:

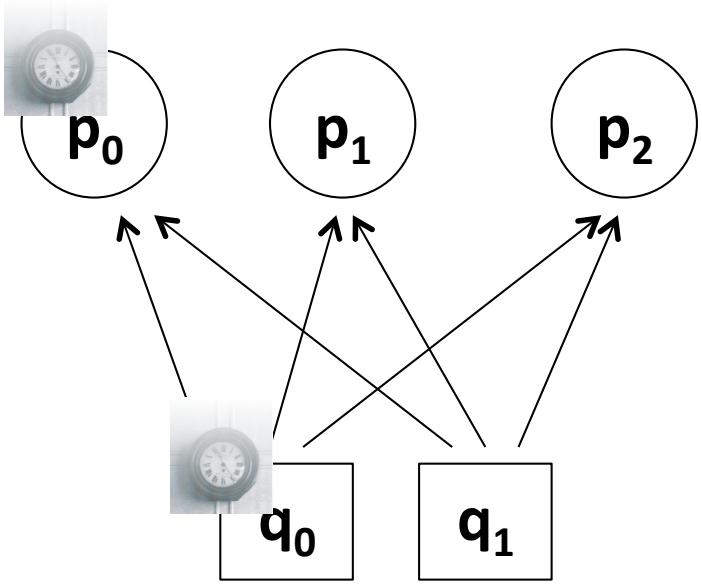
$update_j(view_j)$

$view_j$:= snapshot()

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

Simulation order

- One Safe Agreement per each simulated snapshot
- When the currently simulated step of p_j is resolved or blocked – 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
   if resolved(j,last[j]) then        // return true for last[j]=0
         last[j] := last[j]+1; k := last[j]
         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] := 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 of FIP 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

Liveness

Claim 2 At most one process fails in R

- A faulty simulator may block at most one process
- At most 1 simulators fails
- At most 1 simulated process fails – a 1-resilient run!

1-resilient consensus is impossible

Theorem 2 No 1-resilient algorithm solves consensus for N≥2

Suppose not: there is an algorithm A that solves consensus among $p_0,\ldots,p_{n-1}$ where all but one are correct

Then we solve wait-free consensus among q_0 and q_1 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

Correctness

- Safety: all decisions come from a run of a consensus algorithm A
  - ✓no two simulators decide differently (by Agreement)
  - ✓Every proposed value is an input of some simulator – every decided value is a proposed value (by Validity)

- Liveness: if a simulator is correct – the run of A is one-resilient, eventually every correct simulated process decides (one is enough)

- E. Borowsky, E. Gafni: Generalized FLP impossibility result for t-resilient asynchronous computations. STOC 1993

- Next class (May 10): discussion of Problem Set 1 (same place, same time)

- Slides and literature: ISIS