Algorand

Mădălina Bolboceanu

Bitdefender
theoretical research

RV/ILDS Blockchain Workshop
March 22, 2023

What is this talk about?

A protocol (Algorand*) based on *Byzantine Agreement*, which promises to solve the Blockchain Trilemma, and our proof-of-concept implementation of it in Python.

*** proposed by Chen and Micali, 2017**
www.algorand.com

Outline

1. Byzantine Agreement (BA)

a. Why BA?

b. What is BA?

c. How to build arbitrary value-BA from binary-BA

2. The BA protocol behind Algorand

a. A very intuitive BA protocol

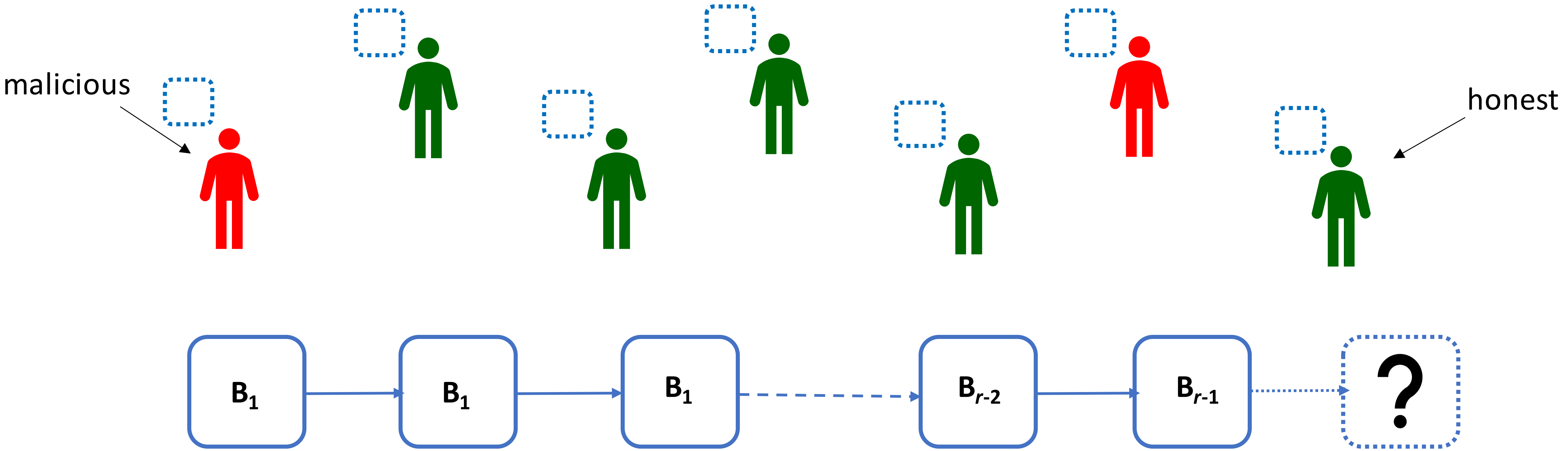
b. The protocol

3. Towards a practical protocol: Algorand

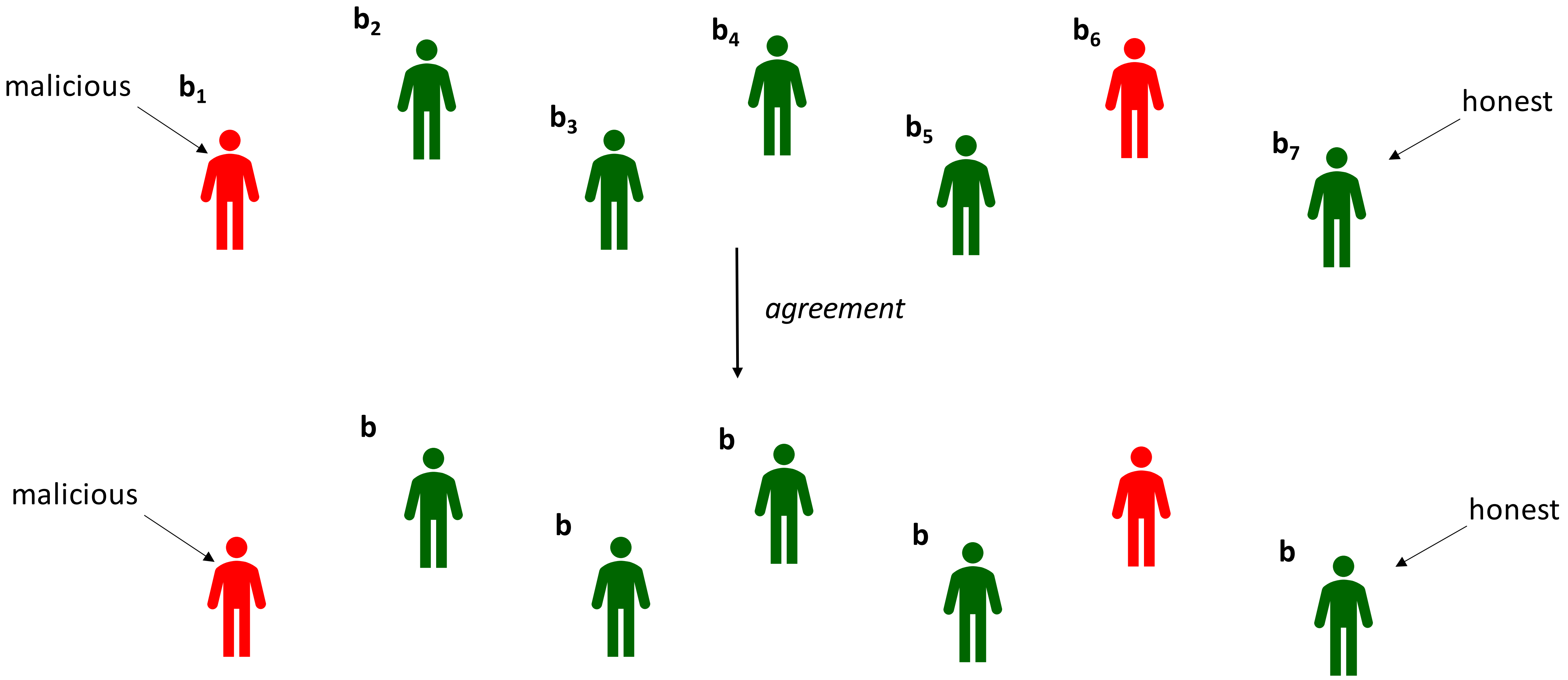
4. Results

Byzantine Agreement

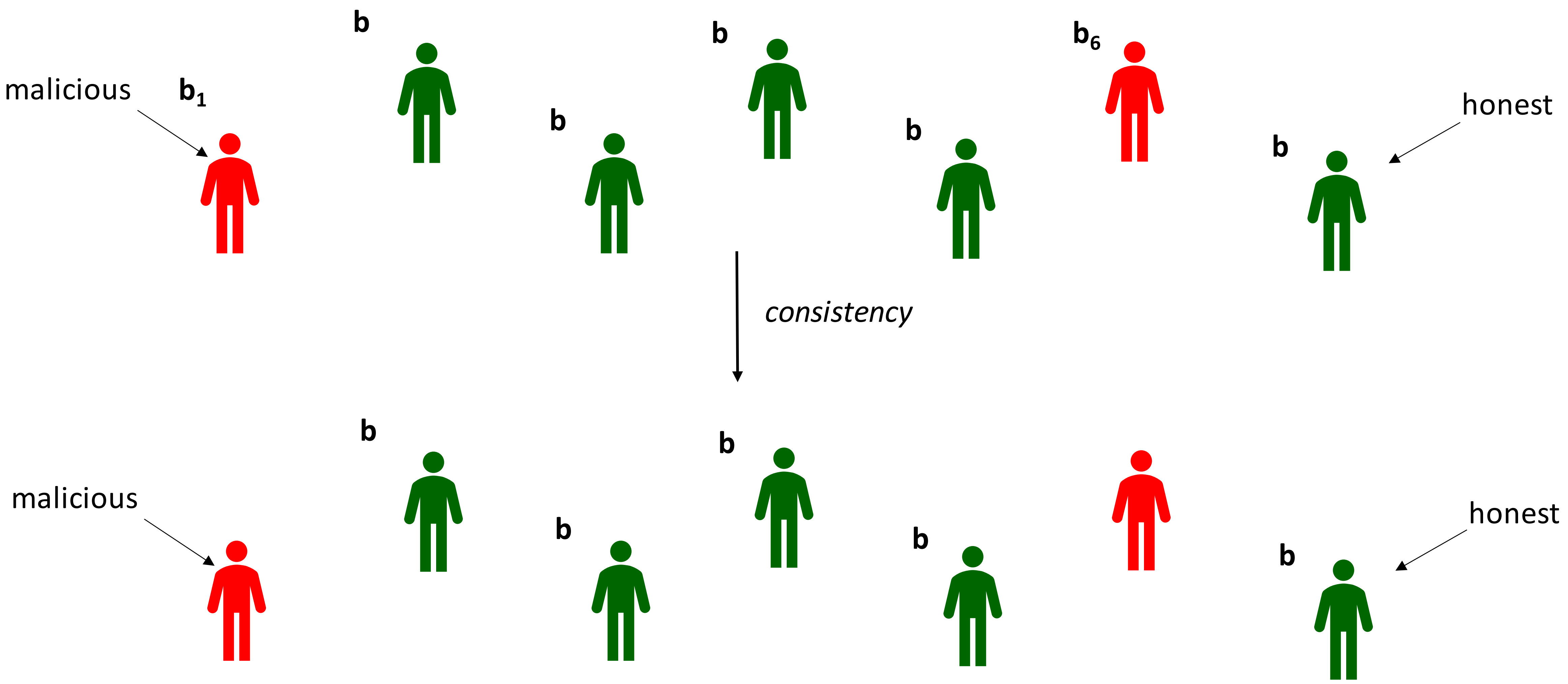
Why BA?



BA = *agreement* + consistency [PeaseShostakLamport80]



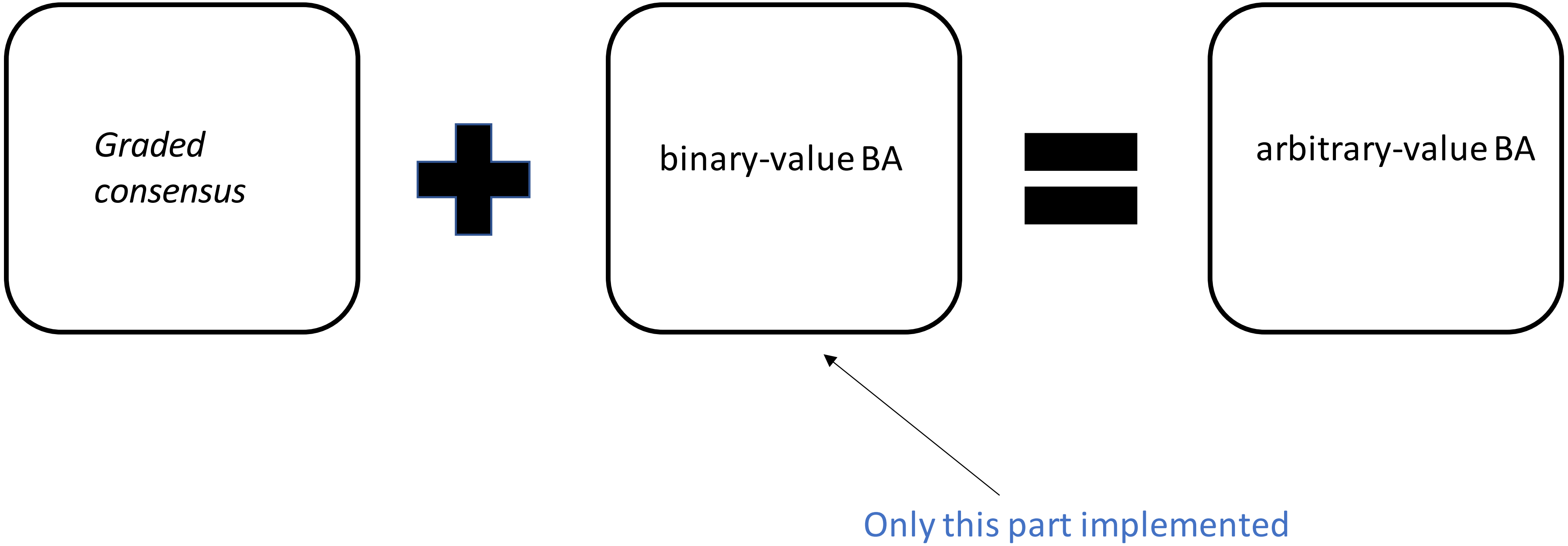
BA = agreement + *consistency* [PeaseShostakLamport80]



From binary-value BA to arbitrary-value BA

Many solutions: the trivial one, [TurpinCoan84], etc.

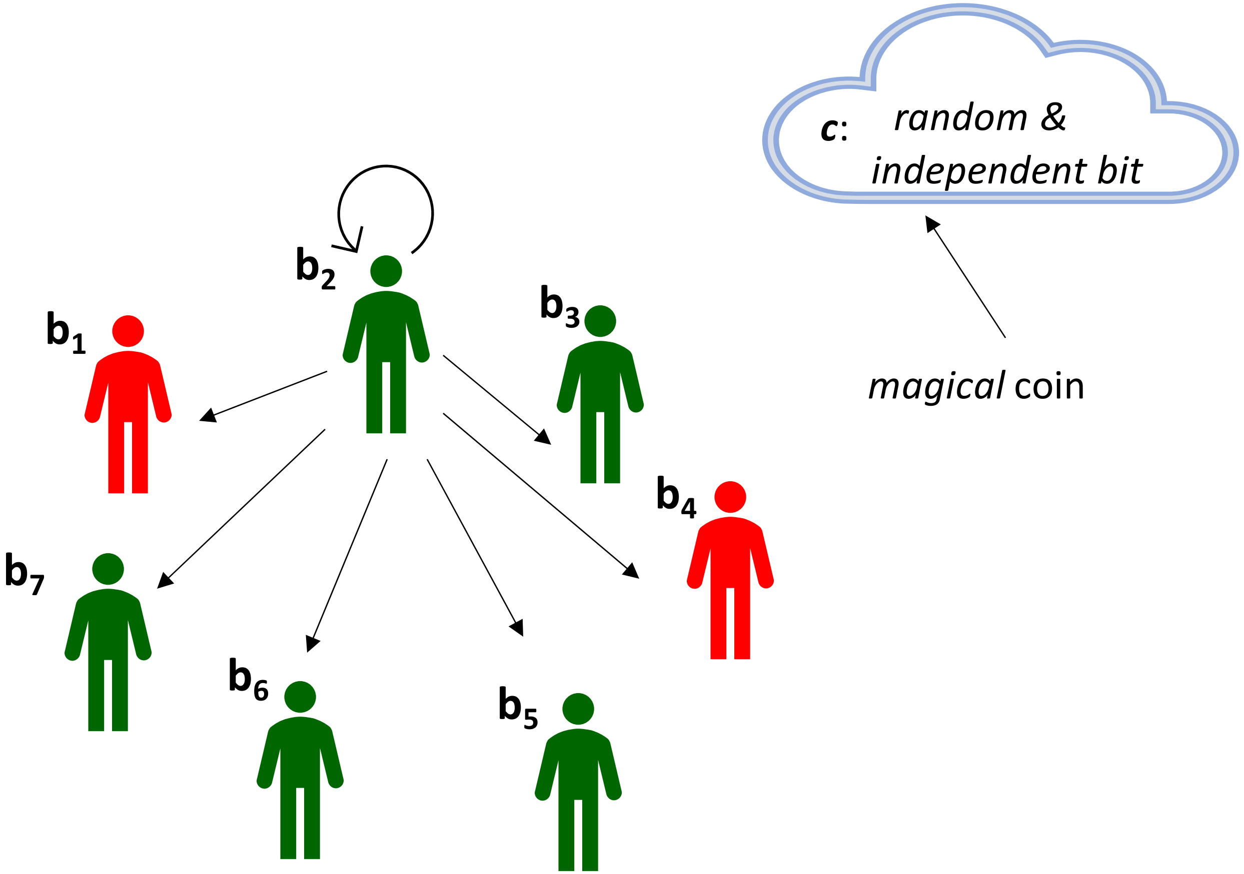
[Micali18] proposes a much cleaner solution:



The BA protocol behind Algorand

A very intuitive BA protocol [FeldmanMicali, 1997]

n=3t+1 players → 2t+1 honest
n=3t+1 players → t malicious



Every player i does:

1. Send $\mathbf{b}_i$ to all the players, including himself.

2. Update $\mathbf{b}_i$ as follows:

a) If $\#_i(0) \geq 2t+1$, then $\mathbf{b}_i = 0$
b) Else, if $\#_i(1) \geq 2t+1$, then $\mathbf{b}_i = 1$
c) Else, $\mathbf{b}_i = \boldsymbol{c}$.

✓ **Consistency**: if the honest players start with the same value, they will end up with that value.

✓ **Agreement**: if the honest players are not in agreement, they will be in agreement with probability ½.

A very intuitive BA protocol [FeldmanMicali, 1997]

n=3t+1 players → 2t+1 honest

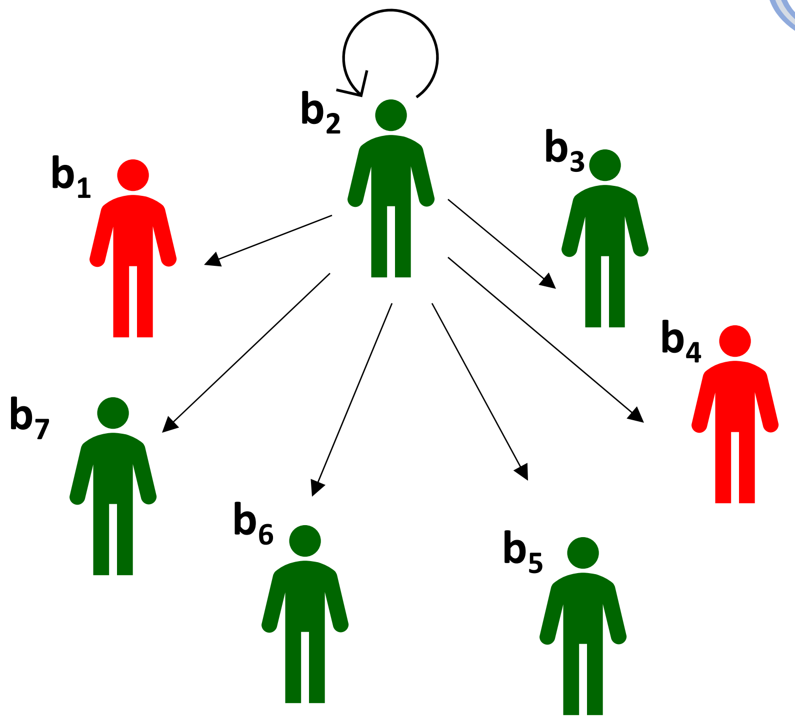
n=3t+1 players → t malicious

Every player i does:

1. Send $\mathbf{b}_i$ to all the players, including himself.

2. Update $\mathbf{b}_i$ as follows:

- If $\#_i(0) \geq 2t+1$, then $\mathbf{b}_i = 0$
- Else, if $\#_i(1) \geq 2t+1$, then $\mathbf{b}_i = 1$
- Else, $\mathbf{b}_i = \boldsymbol{c}$.



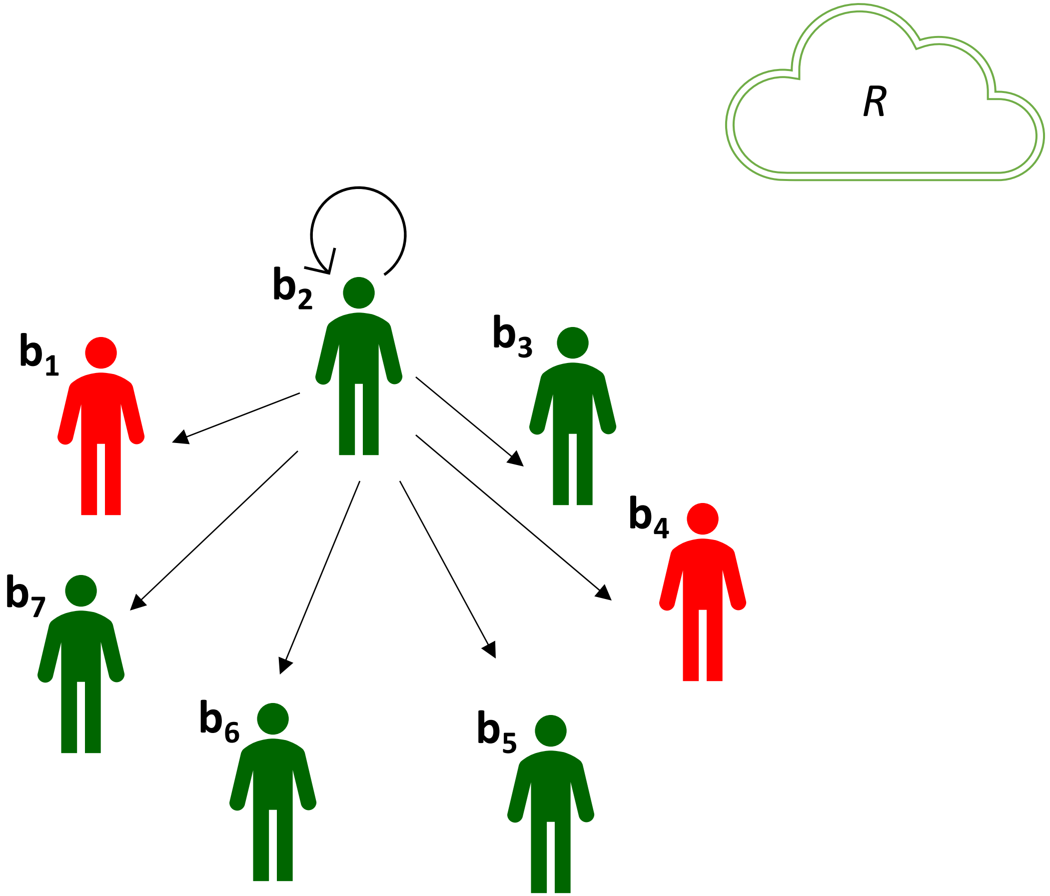
c: *random & independent bit*

Algorand: it suffices a *less magical coin*

Consistency: if the honest players start with the same value, they will end up with that value.

Agreement: if the honest players are not in agreement, they will be in agreement with probability ½.

Algorand's less magical coin



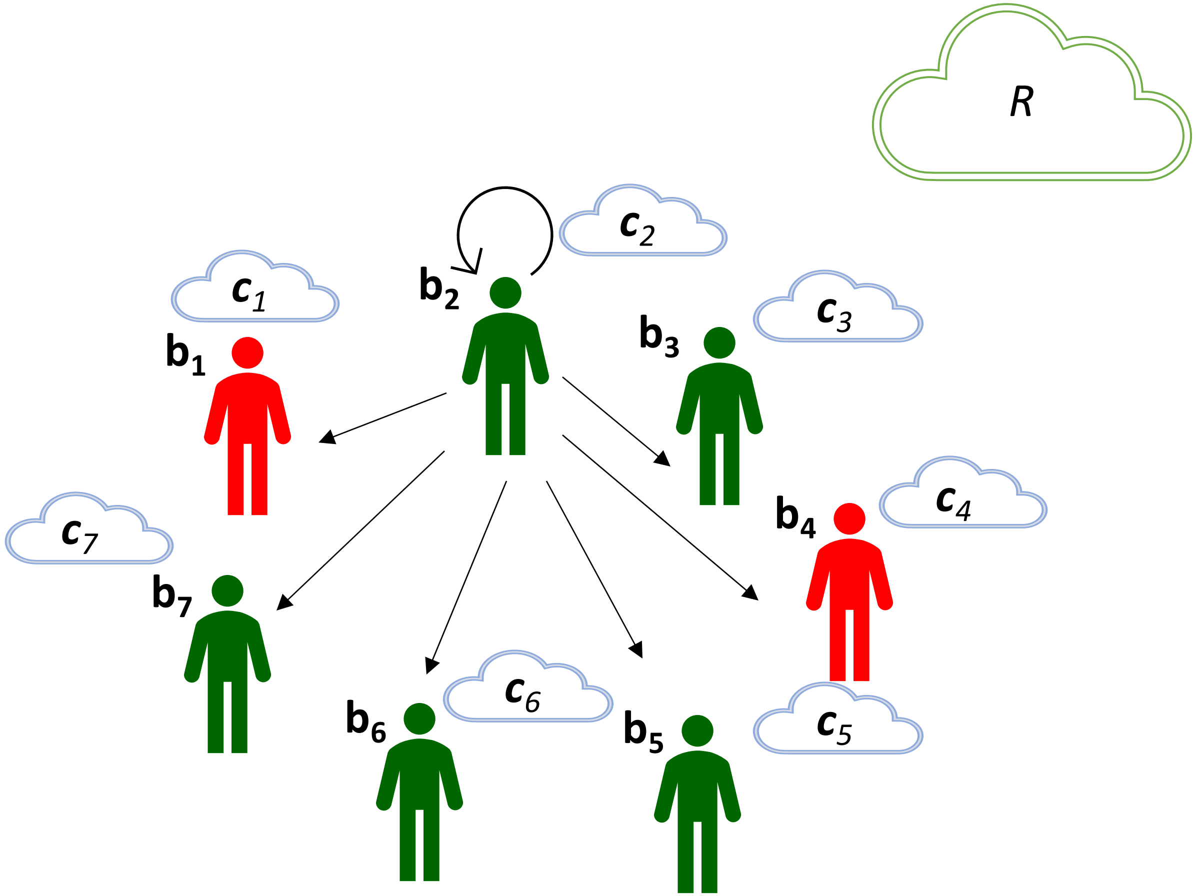
- R: common info
- Sig: digital signature scheme
- H: random oracle

Every player *i* does:

1. Send the value $v_i = \mathrm{Sig}_i(R)$
2. Compute the player *m* s.t. $H(v_m) \leq H(v_j)$ for all j
3. Set $c_i = \mathrm{lsb}\,(H(v_m))$

Algorand's less magical coin

- *R*: common info
- Sig: digital signature scheme
- H: random oracle



Every player *i* does:

1. Send the value $v_i = \mathrm{Sig}_i(R)$
2. Compute the player *m* s.t. $H(v_m) \leq H(v_j)$ for all *j*
3. Set $\boldsymbol{c}_i = \mathrm{lsb}\ (H(v_m))$

In the case of 2/3 honest majority, the $\boldsymbol{c}_i$'s are the same with probability 2/3.

➡ The honest players reach agreement with probability $\geq 1/3$.

But agreement probability is just 1/3, how to increase it?

… repeat the protocol with inputs(s) = outputs(s-1) for many steps s.

0 0 1 0 0 0 1 1 1 1…

1 1 1 0 0 0 1 1 1 1…

0 0 1 1 0 0 1 1 1 1…

0 1 0 0 1 1 1 1 1 1…

After k steps:

⟶ Pr[agreement] $\geq 1 - (2/3)^k$

Once they are in agreement, they will forever be in agreement (because of *Consistency*).

Even if they are already in agreement, they will continue to repeat the protocol and spend unnecessary steps because **they don't know** that they are in agreement.

How to fix this: the actual protocol [Micali2018]

n=3t+1 players → 2t+1 honest, t malicious

Every player i does:

step 1: Coin-Fixed-to-0 step

1.1 Send $\mathbf{b}_i$ to all the players, including himself.

1.2 Update $\mathbf{b}_i$:

- If $\#_i(0) \geq 2t+1$, then $\mathbf{b}_i = 0$, output 0, send 0* and halt
- Else, if $\#_i(1) \geq 2t+1$, then $\mathbf{b}_i = 1$
- Else, $b_i = 0$

step 2: Coin-Fixed-to-1 step

2.1 Send $\mathbf{b}_i$ to all the players, including himself.

2.2 Update $\mathbf{b}_i$:

- If $\#_i(1) \geq 2t+1$, then $\mathbf{b}_i = 1$, output 1, send 1* and halt
- Else, if $\#_i(0) \geq 2t+1$, then $\mathbf{b}_i = 0$
- Else, $\mathbf{b}_i = 1$

step 3: Coin-Genuinely-Flipped step

3.1 Send $\mathbf{b}_i$ and $\mathrm{Sig}_i(R, \textit{iteration})$ to all the players, including himself.

3.2 Compute its $\mathbf{c}_i$ and update $\mathbf{b}_i$:

- If $\#_i(0) \geq 2t+1$, then $\mathbf{b}_i = 0$
- Else, if $\#_i(1) \geq 2t+1$, then $\mathbf{b}_i = 1$
- Else, $\mathbf{b}_i = \mathbf{c}_i$, increase *iteration* by 1 and return to step 1

! It is adapted in Algorand with **minor technical changes**.

Key aspects of the protocol

A. If no halting and no agreement happen until Step 3, the honest players will be in agreement at the end of Step 3 with probability ≥1/3.

B. If, at some step, agreement holds on some bit **b**, then it continues to hold on **b**.

C. If, at some step, an honest player *i* halts, then agreement will hold at the end of the step.

step 1: Coin-Fixed-to-0 step

1.1 Send $\mathbf{b}_i$ to all the players, including himself.

1.2 Update $\mathbf{b}_i$:

- If $\#_i(0) \geq 2t+1$, then $\mathbf{b}_i = 0$, output 0, send 0* and halt
- Else, if $\#_i(1) \geq 2t+1$, then $\mathbf{b}_i = 1$
- Else, $b_i = 0$

step 2: Coin-Fixed-to-1 step

2.1 Send $\mathbf{b}_i$ to all the players, including himself.

2.2 Update $\mathbf{b}_i$:

- If $\#_i(1) \geq 2t+1$, then $\mathbf{b}_i = 1$, output 1, send 1* and halt
- Else, if $\#_i(0) \geq 2t+1$, then $\mathbf{b}_i = 0$
- Else, $\mathbf{b}_i = 1$

step 3: Coin-Genuinely-Flipped step

3.1 Send $\mathbf{b}_i$ and $\mathrm{Sig}_i(R, \mathit{iteration})$ to all the players, including himself.

3.2 Compute its $\mathbf{c}_i$ and update $\mathbf{b}_i$:

- If $\#_i(0) \geq 2t+1$, then $\mathbf{b}_i = 0$
- Else, if $\#_i(1) \geq 2t+1$, then $\mathbf{b}_i = 1$
- Else, $\mathbf{b}_i = \mathbf{c}_i$, increase *iteration* by 1 and return to step 1

✓ Agreement reached for many iterations.

✓ Every player halts.

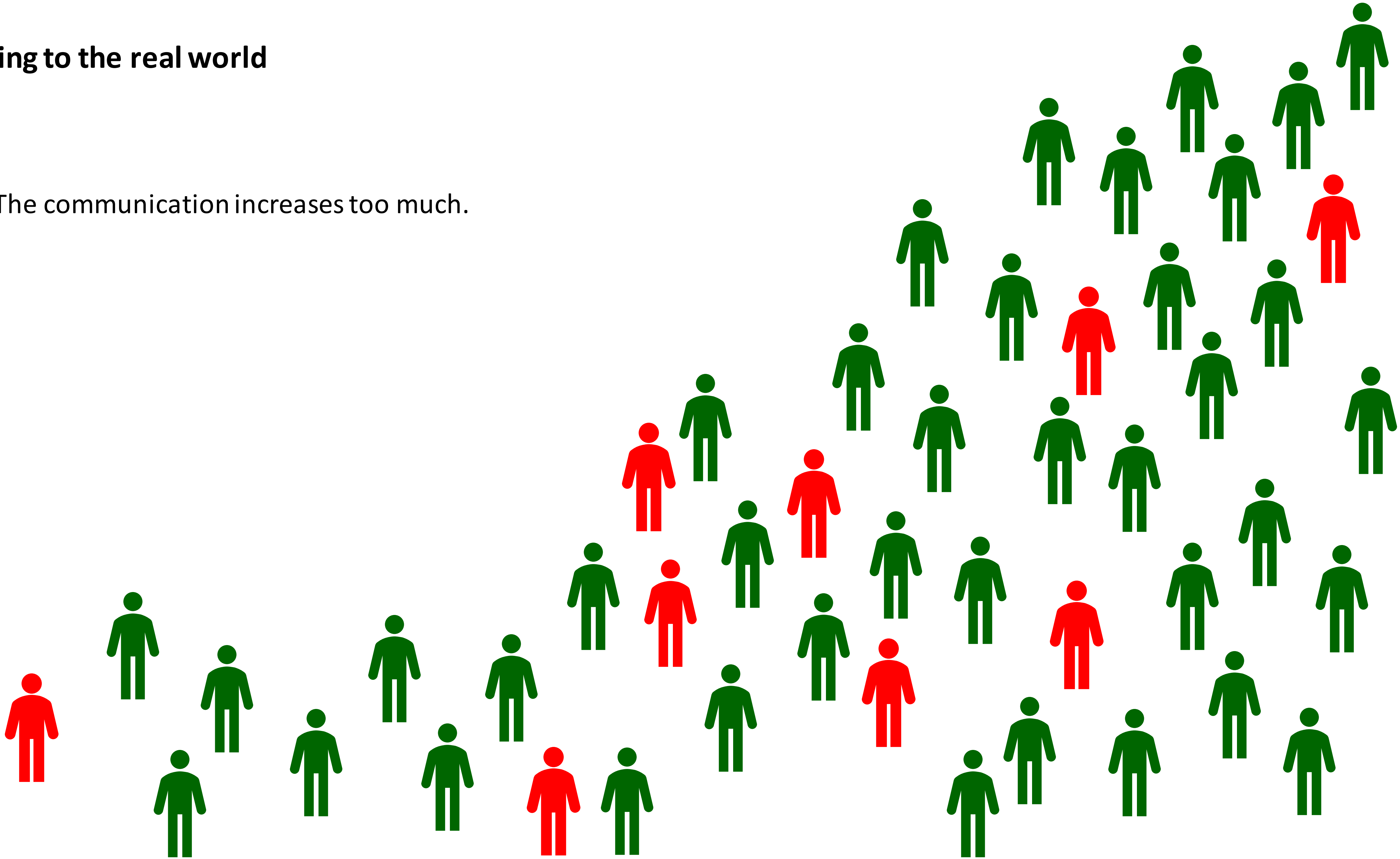
✓ Consistency

✓ Agreement

Towards a practical BA protocol: Algorand

Moving to the real world

The communication increases too much.

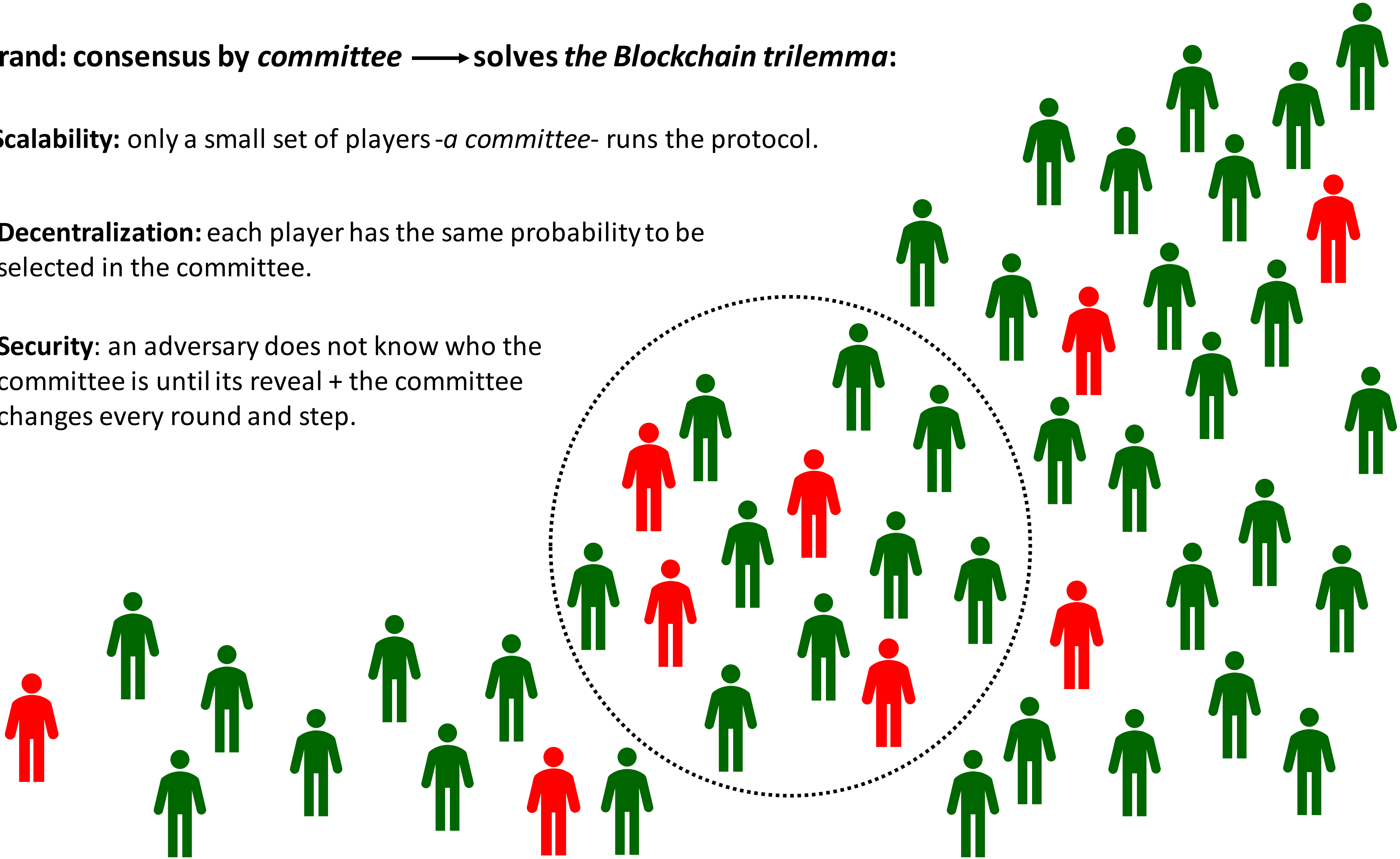


Algorand: consensus by *committee* ⟶ solves *the Blockchain trilemma*:

✓ **Scalability:** only a small set of players -*a committee*- runs the protocol.

✓ **Decentralization:** each player has the same probability to be selected in the committee.

✓ **Security**: an adversary does not know who the committee is until its reveal + the committee changes every round and step.



In the next slides...

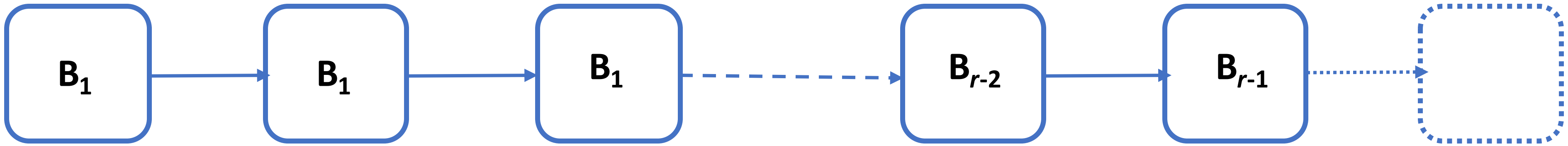
a. Who can propose a new block?

b. Who actually proposes the block?

c. Who can validate the proposed block?

Only this part implemented.

d. How many can validate the proposed block?

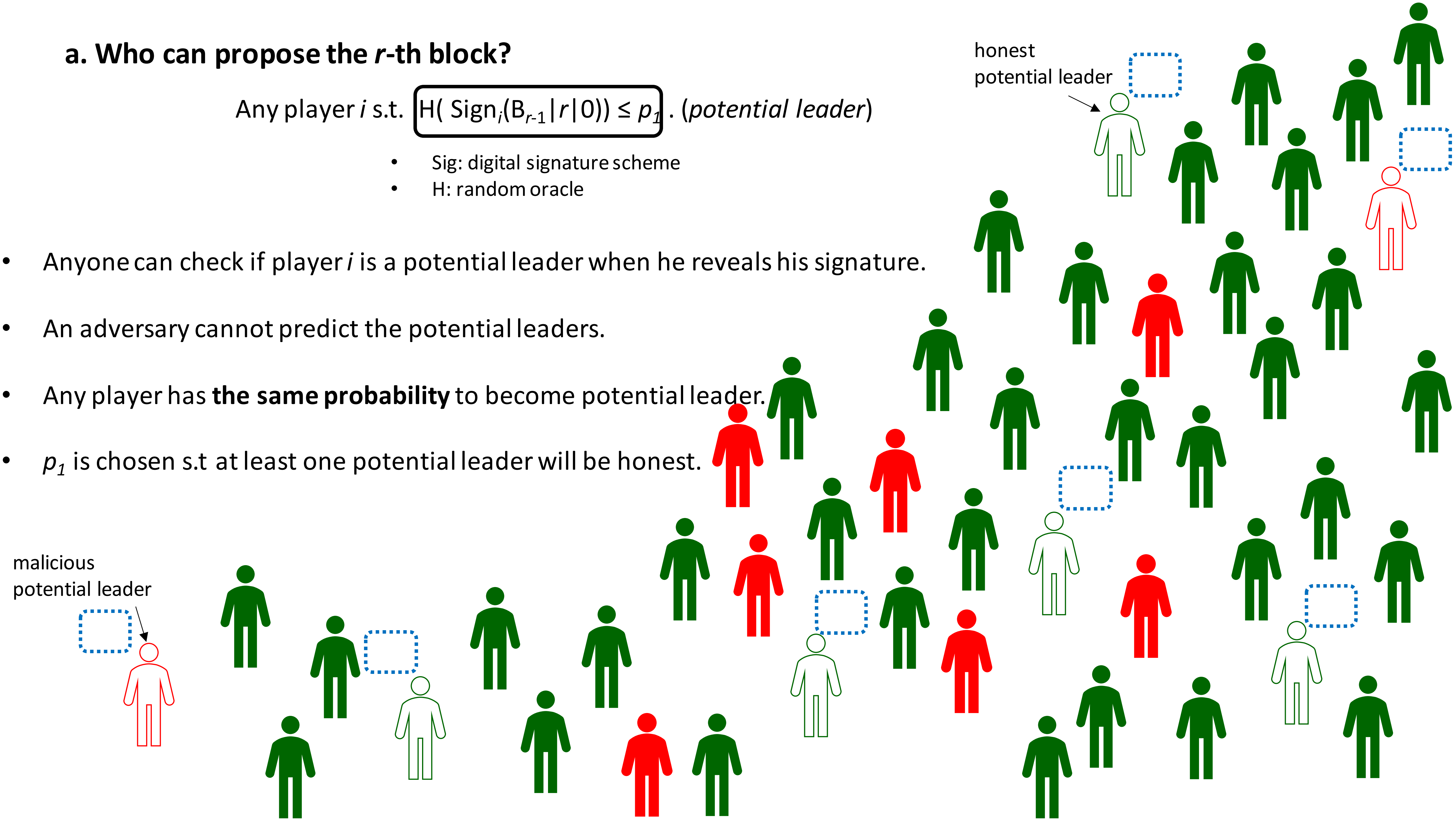


a. Who can propose the *r*-th block?

Any player i s.t. $\boxed{H(\,Sign_i(B_{r-1}|r|0)) \le p_1}$. (*potential leader*)

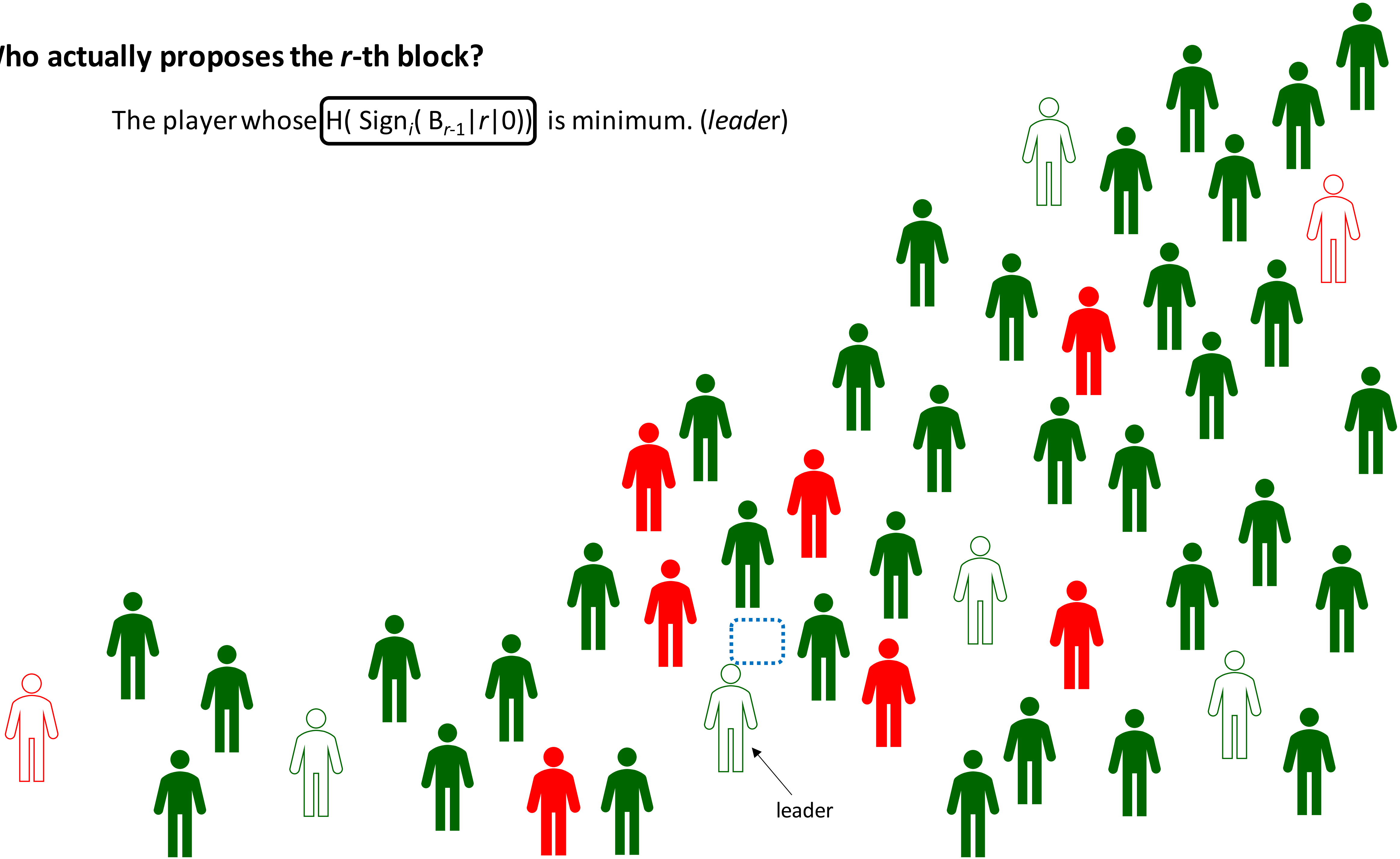
- Sig: digital signature scheme
- H: random oracle

- Anyone can check if player *i* is a potential leader when he reveals his signature.
- An adversary cannot predict the potential leaders.
- Any player has **the same probability** to become potential leader.
- p_1 is chosen s.t at least one potential leader will be honest.



b. Who actually proposes the *r*-th block?

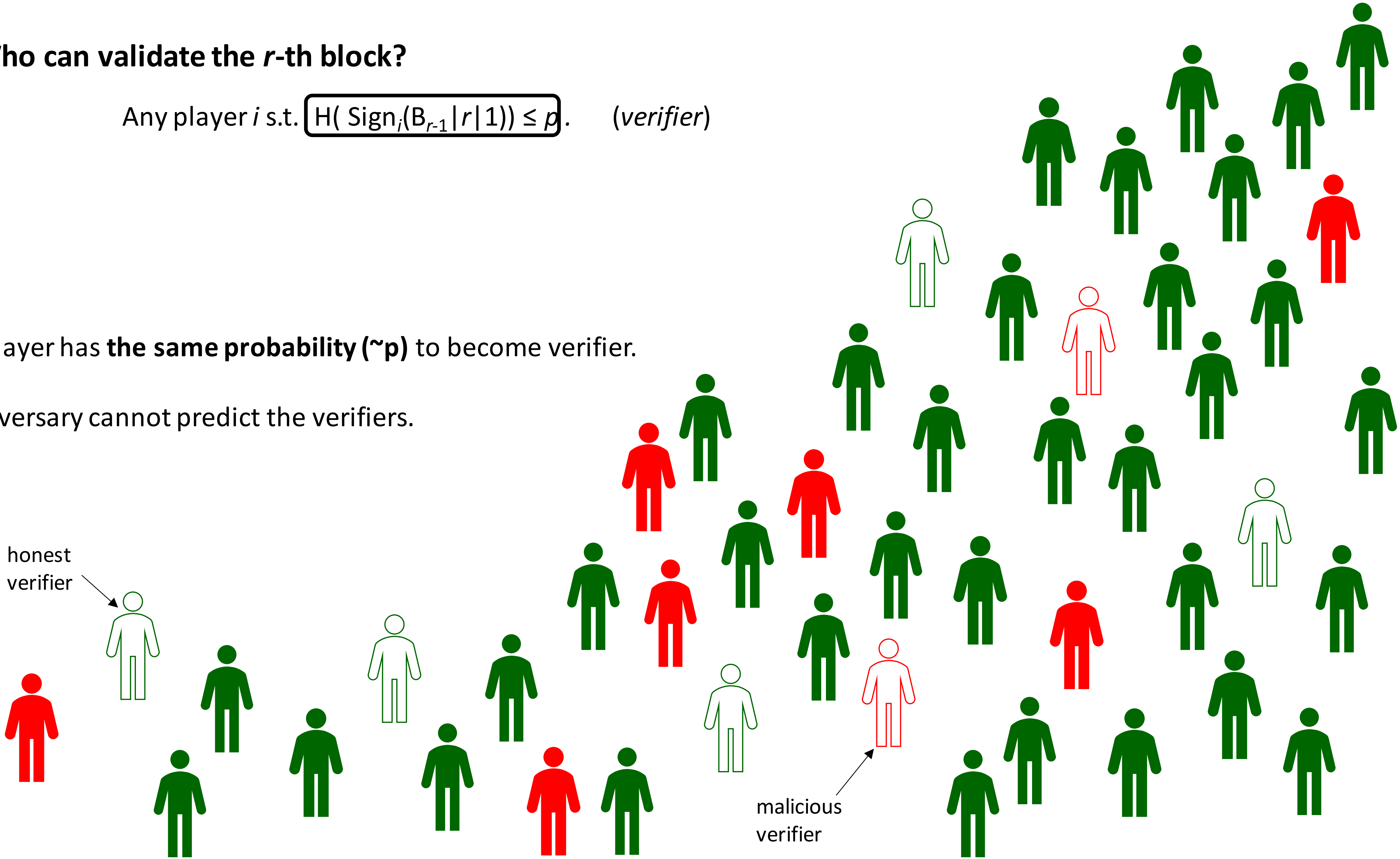
The player whose $\boxed{H(\, Sign_i(\, B_{r-1} | r | 0))}$ is minimum. (*leader*)



c. Who can validate the *r*-th block?

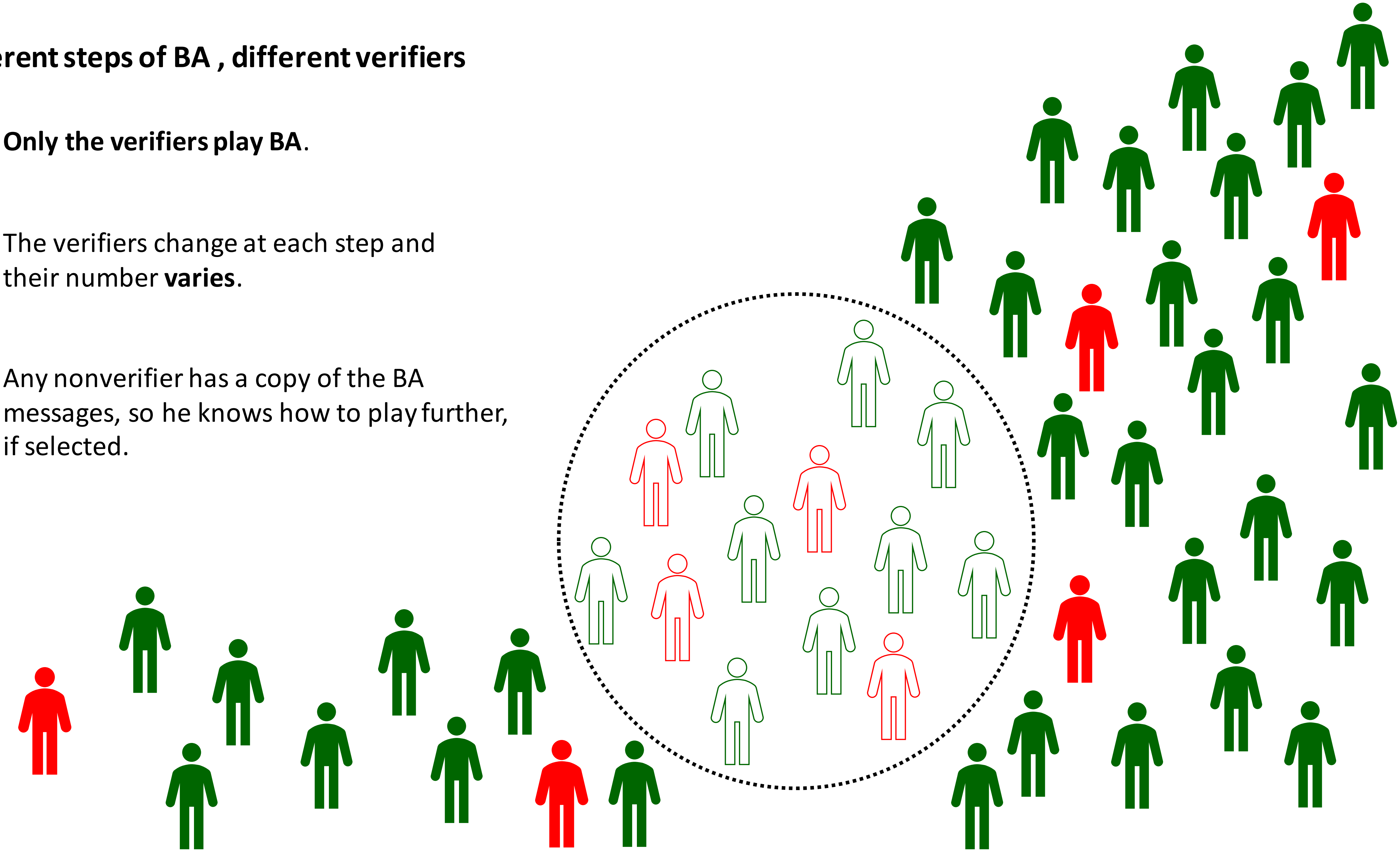
Any player i s.t. $\boxed{H(\ Sign_i(B_{r-1}|r|1)) \leq p}$. (*verifier*)

- Any player has **the same probability (~p)** to become verifier.
- An adversary cannot predict the verifiers.



Different steps of BA , different verifiers

- **Only the verifiers play BA**.
- The verifiers change at each step and their number **varies**.
- Any nonverifier has a copy of the BA messages, so he knows how to play further, if selected.



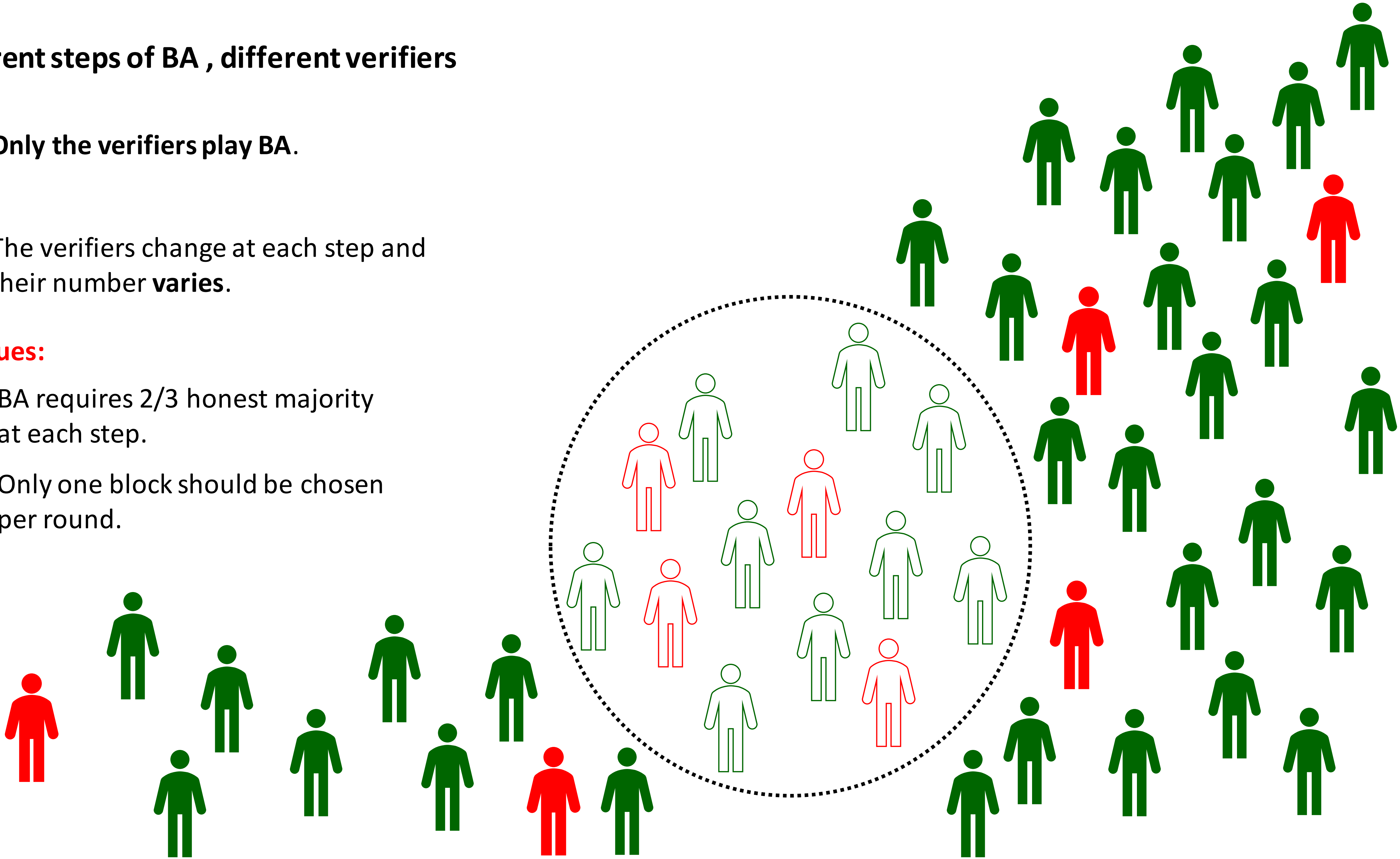
Different steps of BA , different verifiers

- **Only the verifiers play BA**.
- The verifiers change at each step and their number **varies**.

Issues:

a) BA requires 2/3 honest majority at each step.

b) Only one block should be chosen per round.



d. How many can validate the *r*-th proposed block?

Given

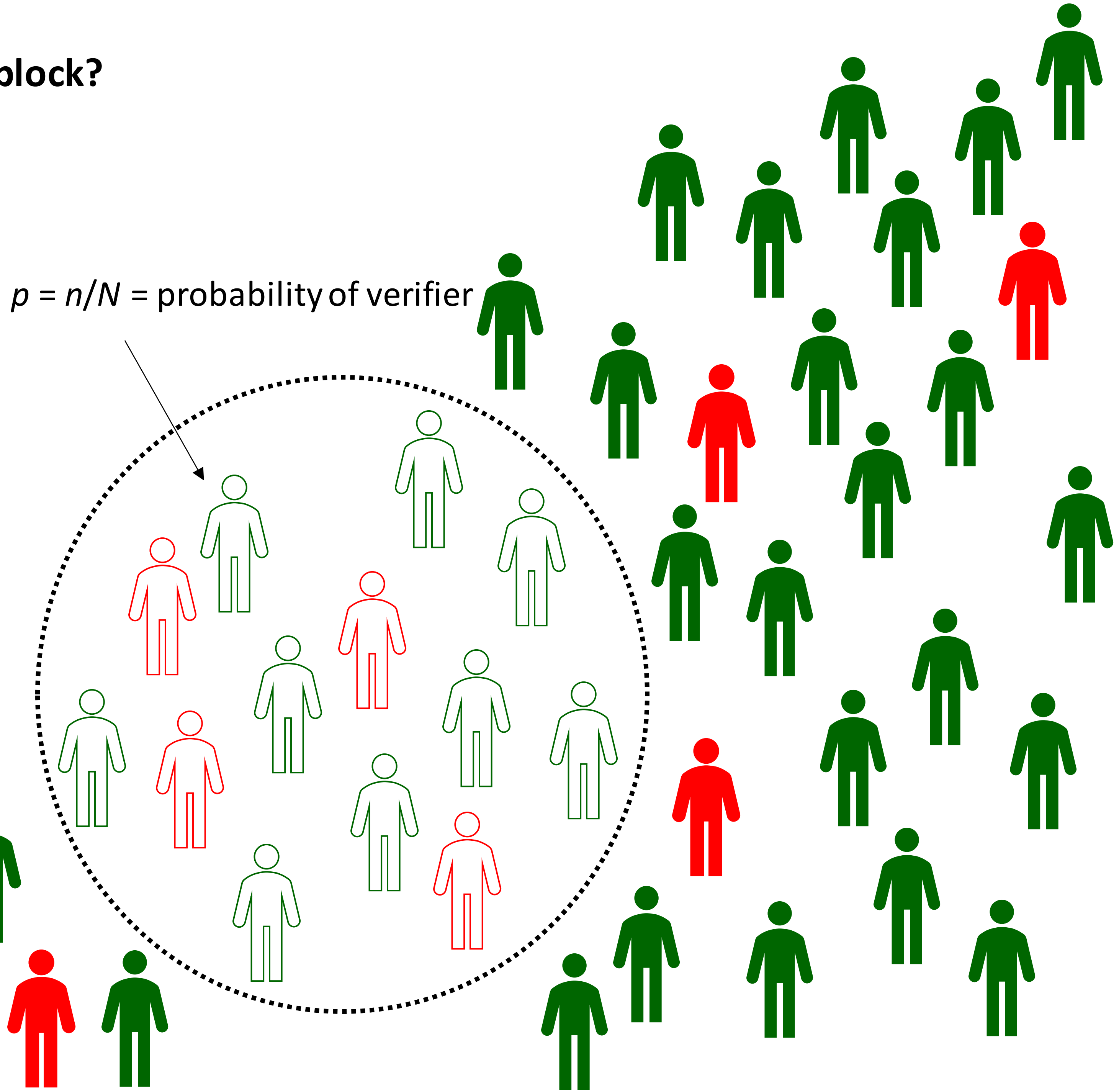
- N = # players
- h = ratio of honest players, in [2/3,1]
- F = failure probability, small value in (0,1)

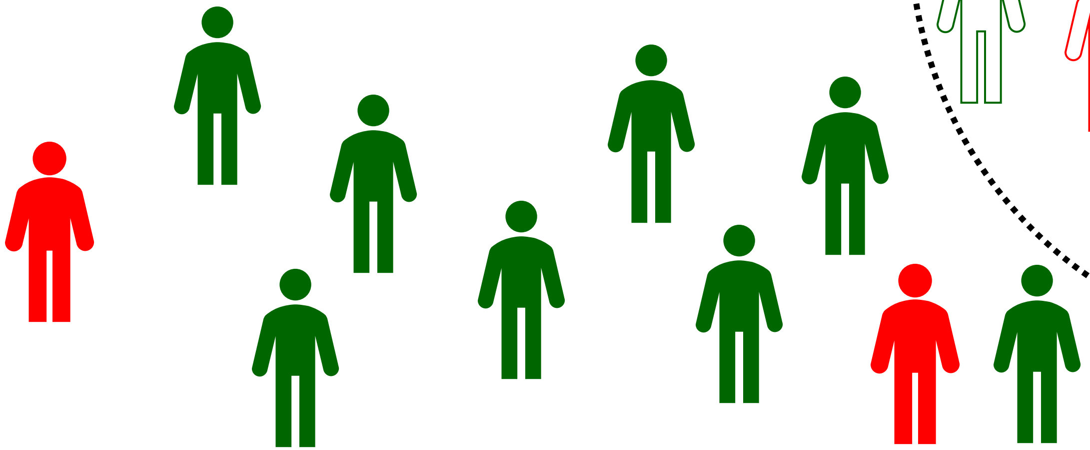
Find

- n = expected # verifiers
  s.t. with probability at least 1- F,

a) BA requires 2/3 honest majority at each step.

b) Only one block should be chosen per round.





Results

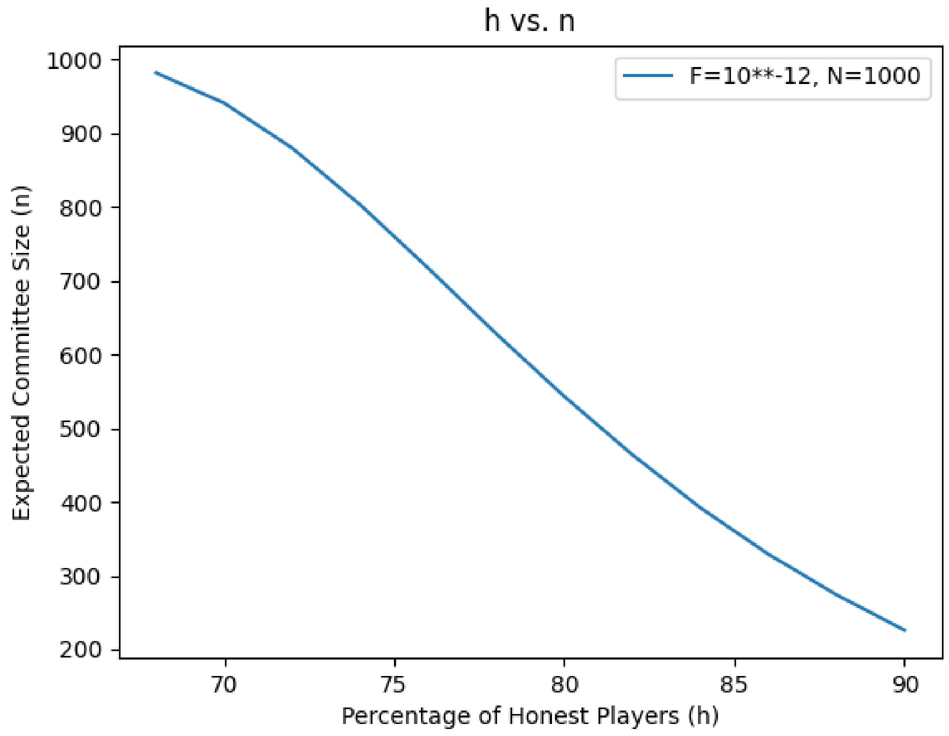
of our PoC Python implementation based on ideas from Algorand's whitepaper.

Sets of parameters

#players *(N)*	Ratio of honest players *(h)*	Fail probability *(F)*	Expected #verifiers *(n)*	Prob. of verifier *p= n/N*
1000	0.8	10^{-12}	543	0.543
1000	0.8	10^{-9}	474	0.474
1500	0.8	10^{-12}	681	0.454
1500	0.8	10^{-9}	574	0.382
2000	0.8	10^{-12}	779	0.389
2000	0.8	10^{-9}	643	0.321

$F \searrow \Rightarrow n \nearrow$

h vs *n* for *N* = 1000 and *F* = 10^{-12}



Ratio of honest players (*h*)	Expected # verifiers (*n*)
0.68	982
0.7	941
0.72	880
0.74	803
0.76	717
0.78	628
0.8	543
0.82	464
0.84	392
0.86	329
0.88	274
0.9	226

$h \nearrow 1 \Rightarrow n \searrow$

Results

rounds = 10
steps = 9

h = 0.8 (ratio of honest players)
F = 10^{-12} (fail probability)

#players *(N)*	Expected #verifiers *(n)*	Time per round (avg)	Comm. per round (avg)
100	82	7.76sec	0.05MB
150	119	23sec	0.17MB
200	155	58sec	0.38MB
250	190	161sec	0.64MB
500	337	2096sec	3.19MB

Our crypto team at Bitdefender

Mădălina Bolboceanu

Radu Țițiu

Miruna Roșca

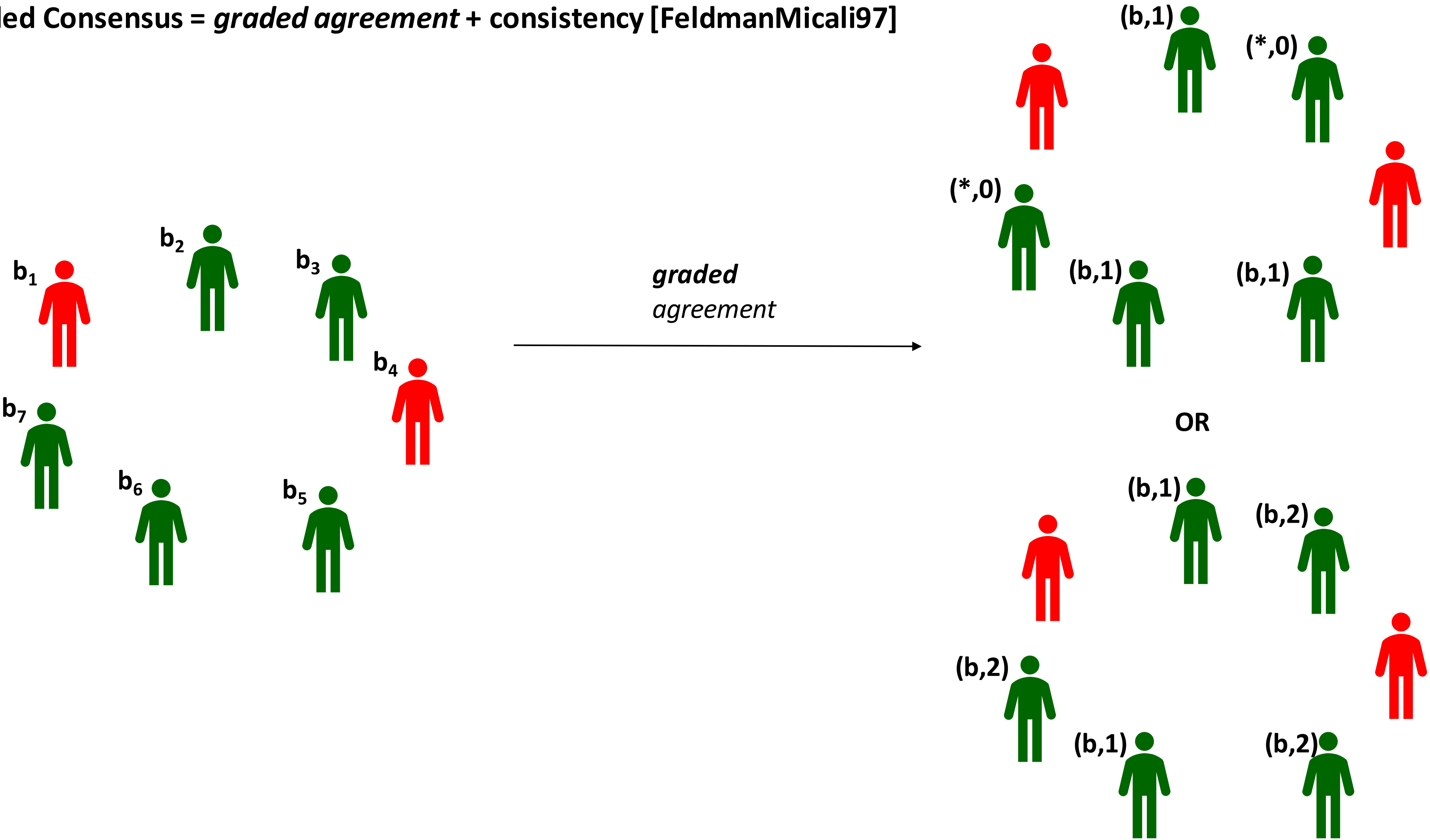
Andrei Pantea

Dacian Stroia

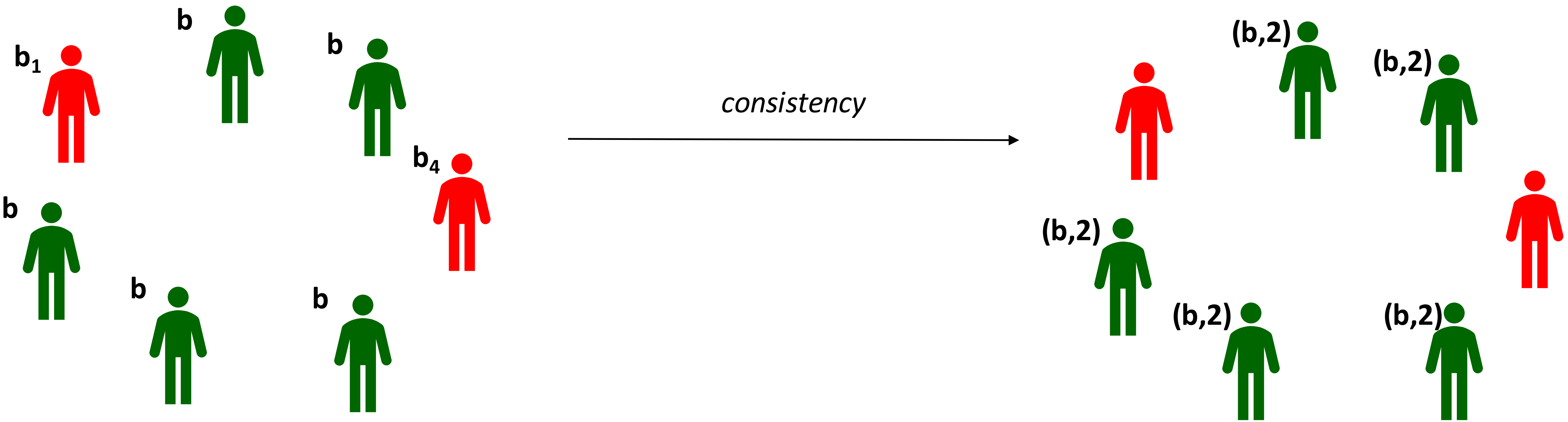
Thank you.

Appendix

Graded Consensus = *graded agreement* + consistency [FeldmanMicali97]



Graded Consensus = graded agreement + *consistency* [FeldmanMicali97]



[Micali2018] in Algorand

n = expected # of verifiers

B_r = the r-th block

steps = multiple of 3

- step 0: verifiers send BA inputs
- step $s \geq 1$: every verifier i does:

CERT = the set of $2n/3+1$ identical messages used in obtaining B_r.

step 1

1.1 Check if he can get B_r from messages of previous steps.

1.2 If not, update $\mathbf{b}_i$:

- If $\#_i(0) \geq 2n/3+1$, then $\mathbf{b}_i = 0$, output 0, gets B_r and send CERT
- Else, if $\#_i(\mathbf{b}) \geq 2/3*$(msg received)+1, then $\mathbf{b}_i = \mathbf{b}$
- Else, $\mathbf{b}_i = 0$

1.3 Send $\mathbf{b}_i$ to all the players, including himself.

step 2

2.1 Check if he can get B_r from messages of previous steps.

2.2 If not, update $\mathbf{b}_i$:

- If $\#_i(1) \geq 2n/3+1$, then $\mathbf{b}_i = 1$, output 1, gets B_r and send CERT
- Else, if $\#_i(\mathbf{b}) \geq 2/3*$(msg received)+1, then $\mathbf{b}_i = \mathbf{b}$
- Else, $\mathbf{b}_i = 1$

2.3 Send $\mathbf{b}_i$ to all the players, including himself.

step 3

3.1 Check if he can get B_r from messages of previous steps.

3.2 If not, update $\mathbf{b}_i$:

- If $\#_i(\mathbf{b}) \geq 2/3*$(msg received)+1, then $\mathbf{b}_i = \mathbf{b}$
- Else, $\mathbf{b}_i = \mathrm{lsb}(\min_j H(\mathrm{Sig}_j(B_{r-1}|r|s))$, return to step 1

3.3 Send $\mathbf{b}_i$ to all the players, including himself.

[Micali2018] in Algorand

n = expected # of verifiers　B$_r$ = the r-th block

steps = multiple of 3

- step 0: verifiers send BA inputs
- step $s \geq 1$: every verifier i does:

CERT = the set of $2n/3+1$ identical messages used in obtaining B$_r$.

step 1

1.1 Check if he can get B$_r$ from messages of previous steps.

1.2 If not, update $\mathbf{b}_i$:

- If $\#_i(0) \geq 2n/3+1$, then $\mathbf{b}_i = 0$, output 0, gets B$_r$ and send CERT
- Else, if $\#_i(\mathbf{b}) \geq 2/3*$(msg received)+1, then $\mathbf{b}_i = \mathbf{b}$
- Else, $\mathbf{b}_i = 0$

1.3 Send $\mathbf{b}_i$ to all the players, including himself.

step 2

2.1 Check if he can get B$_r$ from messages of previous steps.

2.2 If not, update $\mathbf{b}_i$:

- If $\#_i(1) \geq 2n/3+1$, then $\mathbf{b}_i = 1$, output 1, gets B$_r$ and send CERT
- Else, if $\#_i(\mathbf{b}) \geq 2/3*$(msg received)+1, then $\mathbf{b}_i = \mathbf{b}$
- Else, $\mathbf{b}_i = 1$

2.3 Send $\mathbf{b}_i$ to all the players, including himself.

step 3

3.1 Check if he can get B$_r$ from messages of previous steps.

3.2 If not, update $\mathbf{b}_i$:

- If $\#_i(\mathbf{b}) \geq 2/3*$(msg received)+1, then $\mathbf{b}_i = \mathbf{b}$
- Else, $\mathbf{b}_i = \text{lsb}(\min_j H(\text{Sig}_j(B_{r-1}|r|s))$, return to step 1

3.3 Send $\mathbf{b}_i$ to all the players, including himself.

- Last step (Step 2-like): Every verifier i checks if he can get B$_r$ from messages of previous steps. If not, i outputs 1, gets B$_r$ and sends CERT = {1}.

[Micali2018] in Algorand

n = expected # of verifiers B_r = the *r*-th block

steps = multiple of 3

- step 0: verifiers send BA inputs
- step *s*≥1: every verifier *i* does:

CERT = the set of $2n/3+1$ identical messages used in obtaining B_r.

step 1

1.1 Check if he can get B_r from messages of previous steps.

1.2 If not, update $\mathbf{b}_i$:

- If $\#_i(0) \geq 2n/3+1$, then $\mathbf{b}_i = 0$, output 0, gets B_r and send CERT
- Else, if $\#_i(\mathbf{b}) \geq 2/3*(\text{msg received})+1$, then $\mathbf{b}_i = \mathbf{b}$
- Else, $\mathbf{b}_i = 0$

1.3 Send $\mathbf{b}_i$ to all the players, including himself.

step 2

2.1 Check if he can get B_r from messages of previous steps.

2.2 If not, update $\mathbf{b}_i$:

- If $\#_i(1) \geq 2n/3+1$, then $\mathbf{b}_i = 1$, output 1, gets B_r and send CERT
- Else, if $\#_i(\mathbf{b}) \geq 2/3*(\text{msg received})+1$, then $\mathbf{b}_i = \mathbf{b}$
- Else, $\mathbf{b}_i = 1$

2.3 Send $\mathbf{b}_i$ to all the players, including himself.

step 3

3.1 Check if he can get B_r from messages of previous steps.

3.2 If not, update $\mathbf{b}_i$:

- If $\#_i(\mathbf{b}) \geq 2/3*(\text{msg received})+1$, then $\mathbf{b}_i = \mathbf{b}$
- Else, $\mathbf{b}_i = \text{lsb}(\min_j H(Sig_j(B_{r-1}|r|s))$, return to step 1

3.3 Send $\mathbf{b}_i$ to all the players, including himself.

- Last step (Step 2-like): Every verifier *i* checks if he can get B_r from messages of previous steps. If not, *i* outputs 1, gets B_r and sends CERT = {1}.
- Nonverifiers can check if they can get B_r, too. If not, they count $2n/3+1$ bits of 1 from Last step and get B_r.