CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

Early Stopping for Any Number of Corruptions

Julian Loss, CISPA Helmholtz Center for Information Security
Jesper Buus Nielsen, Aarhus University

Broadcast

Broadcast

- n parties: sender S with message m and $n - 1$ receivers

Broadcast

- n parties: sender S with message m and $n - 1$ receivers

Broadcast

- n parties: sender S with message m and $n - 1$ receivers
- Validity: all receivers output m if S is honest

m

m

m

m

Broadcast

- n parties: sender S with message m and $n - 1$ receivers
- Validity: all receivers output m if S is honest

Broadcast

- n parties: sender S with message m and $n - 1$ receivers
- Validity: all receivers output m if S is honest
- Consistency: all receivers output the same message

$\hat{m}$

$\hat{m}$

$\hat{m}$

Broadcast

- n parties: sender S with message m and $n-1$ receivers
- Validity: all receivers output m if S is honest
- Consistency: all receivers output the same message
- Termination: All parties terminate

$\hat{m}$

$\hat{m}$

$\hat{m}$

Dolev-Strong Protocol

Dolev-Strong Protocol

- Tolerates $t < n$ corruptions

Dolev-Strong Protocol

- Tolerates $t < n$ corruptions

Dolev-Strong Protocol

- Tolerates $t < n$ corruptions
- Runs in $t + 1$ rounds

Dolev-Strong Protocol

- Tolerates $t < n$ corruptions
- Runs in $t + 1$ rounds

Dolev-Strong Protocol

- Tolerates $t < n$ corruptions
- Runs in $t + 1$ rounds
- Can we terminate earlier when $f \ll t$ parties are corrupted?

Results and Comparison with Early-Stopping Literature

Results and Comparison with Early-Stopping Literature

- Dolev-Strong-Reischuk (JACM 1990): Any early stopping protocol runs in $\min\{f+2,\ t+1\}$ rounds in the **worst case**

Results and Comparison with Early-Stopping Literature

- Dolev-Strong-Reischuk (JACM 1990): Any early stopping protocol runs in $\min\{f+2,\ t+1\}$ rounds in the **worst case**
- Berman, Garay, Perry (WDAG 1992): $\min\{f+2,\ t+1\}$, exponential Communication

Results and Comparison with Early-Stopping Literature

- Dolev-Strong-Reischuk (JACM 1990): Any early stopping protocol runs in $\min\{f+2,\ t+1\}$ rounds in the **worst case**
- Berman, Garay, Perry (WDAG 1992): $\min\{f+2,\ t+1\}$, exponential Communication
- Abraham-Dolev (STOC 2015): $\min\{f+2,\ t+1\}$ rounds for $t < n/3$ and i.t. security

Results and Comparison with Early-Stopping Literature

- Dolev-Strong-Reischuk (JACM 1990): Any early stopping protocol runs in $\min\{f+2,\ t+1\}$ rounds in the **worst case**
- Berman, Garay, Perry (WDAG 1992): $\min\{f+2,\ t+1\}$, exponential Communication
- Abraham-Dolev (STOC 2015): $\min\{f+2,\ t+1\}$ rounds for $t < n/3$ and i.t. security
- Perry-Toueg (Manuscript 1984): $2 \cdot \min\{f+2,\ t+1\}$ rounds for $t < n/2$ and signatures

Results and Comparison with Early-Stopping Literature

- Dolev-Strong-Reischuk (JACM 1990): Any early stopping protocol runs in $\min\{f+2,\ t+1\}$ rounds in the **worst case**
- Berman, Garay, Perry (WDAG 1992): $\min\{f+2,\ t+1\}$, exponential Communication
- Abraham-Dolev (STOC 2015): $\min\{f+2,\ t+1\}$ rounds for $t < n/3$ and i.t. security
- Perry-Toueg (Manuscript 1984): $2 \cdot \min\{f+2,\ t+1\}$ rounds for $t < n/2$ and signatures
- **This work:** $O(\min\{f^2, t\})$ for $t < n$ and signatures

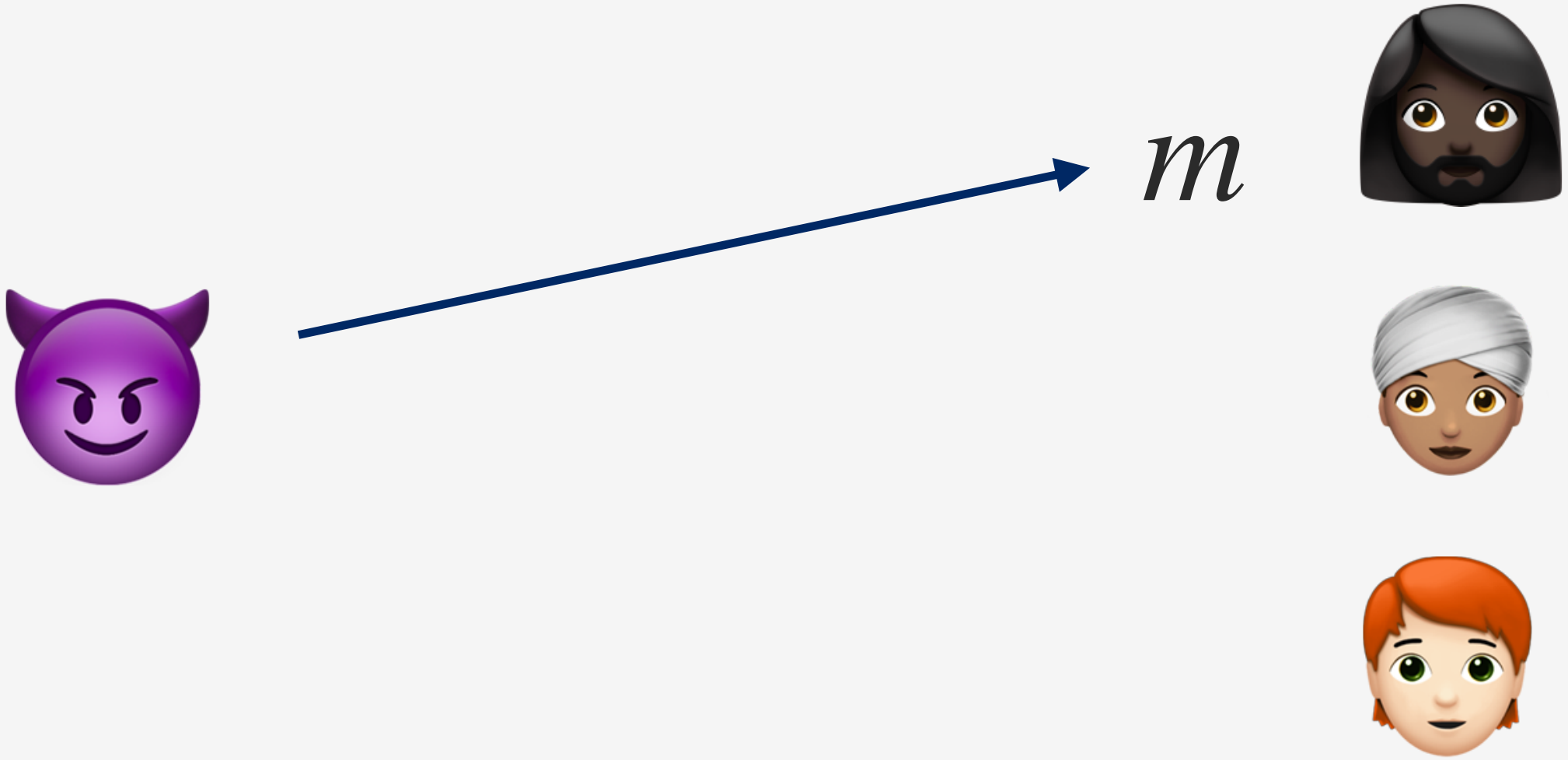
Detecting Equivocation is Easy

Detecting Equivocation is Easy

- **Equivocation:** Sender S sends different messages $m, \hat{m}$ to parties

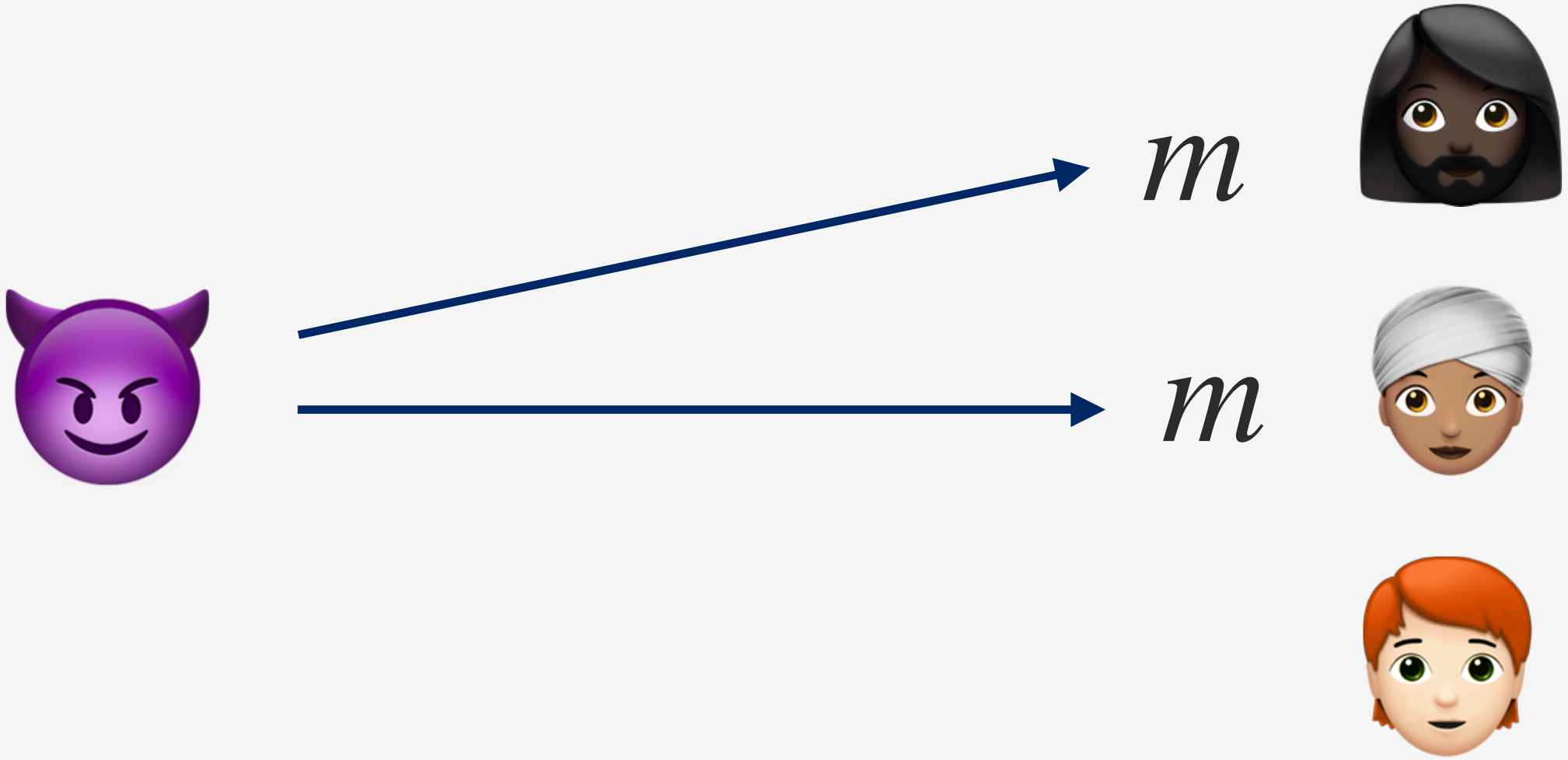
Detecting Equivocation is Easy

- **Equivocation:** Sender S sends different messages $m, \hat{m}$ to parties



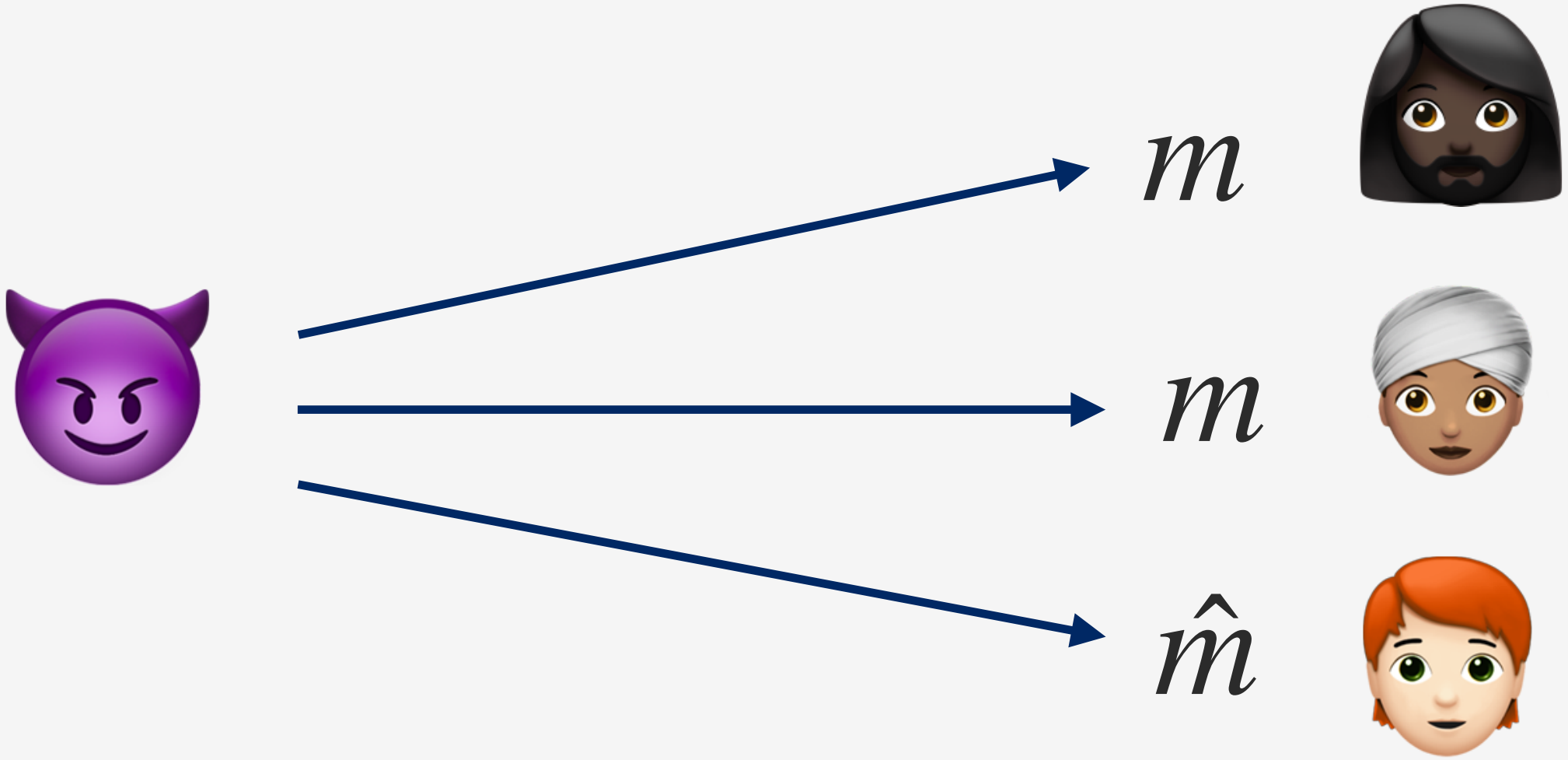
Detecting Equivocation is Easy

- **Equivocation:** Sender S sends different messages $m, \hat{m}$ to parties



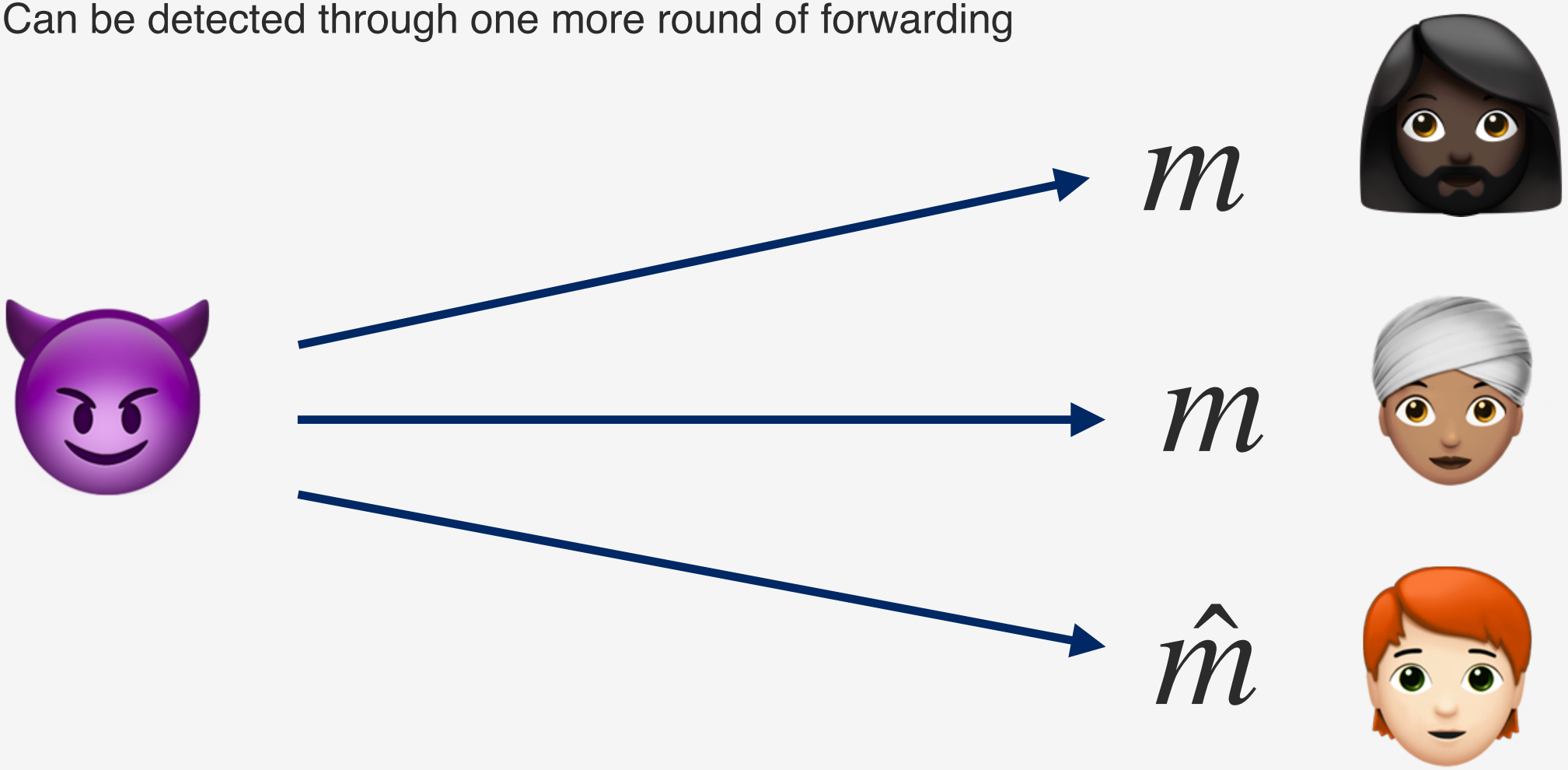
Detecting Equivocation is Easy

- **Equivocation:** Sender S sends different messages $m, \hat{m}$ to parties



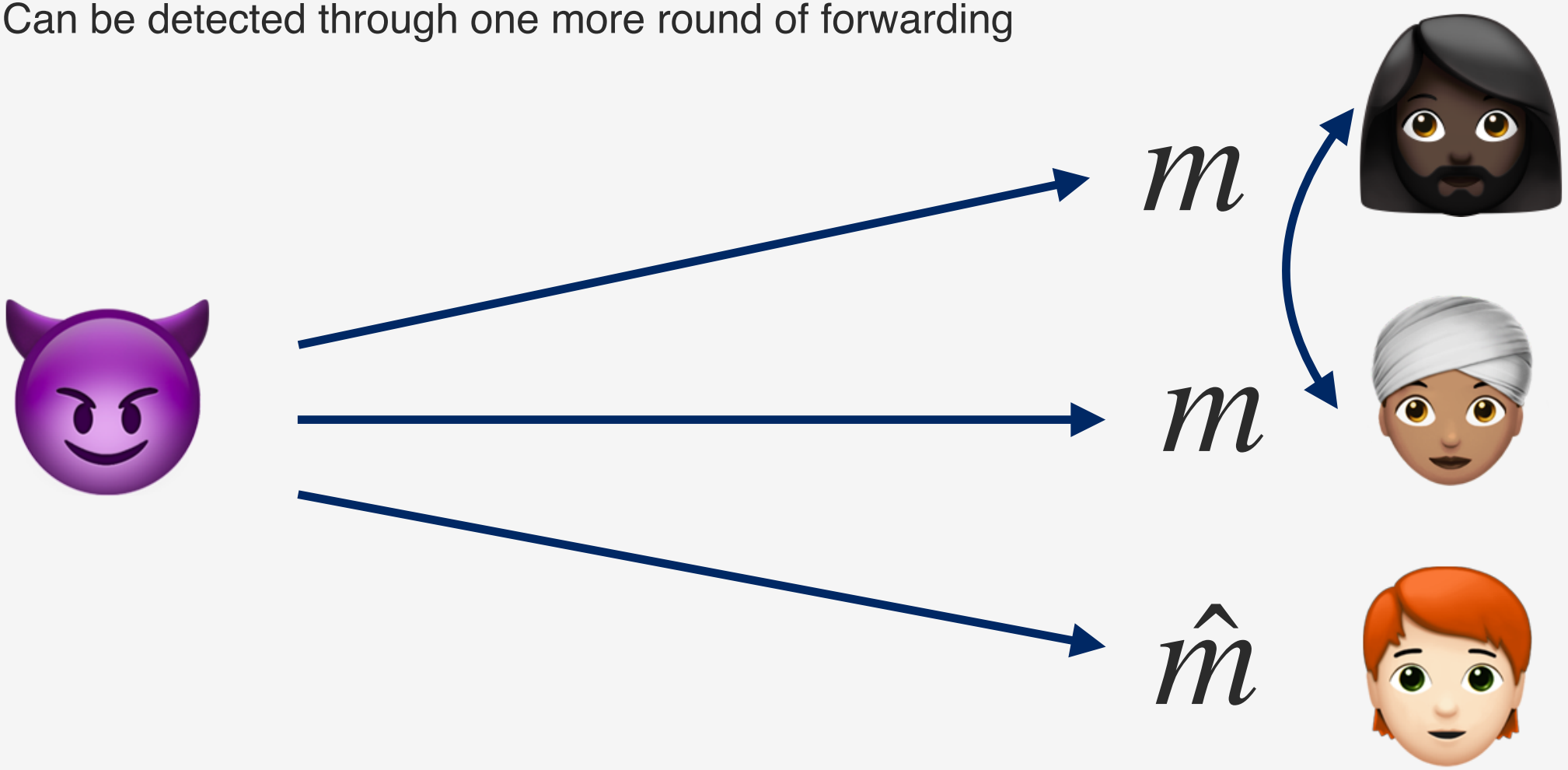
Detecting Equivocation is Easy

- **Equivocation:** Sender S sends different messages $m, \hat{m}$ to parties
- Can be detected through one more round of forwarding



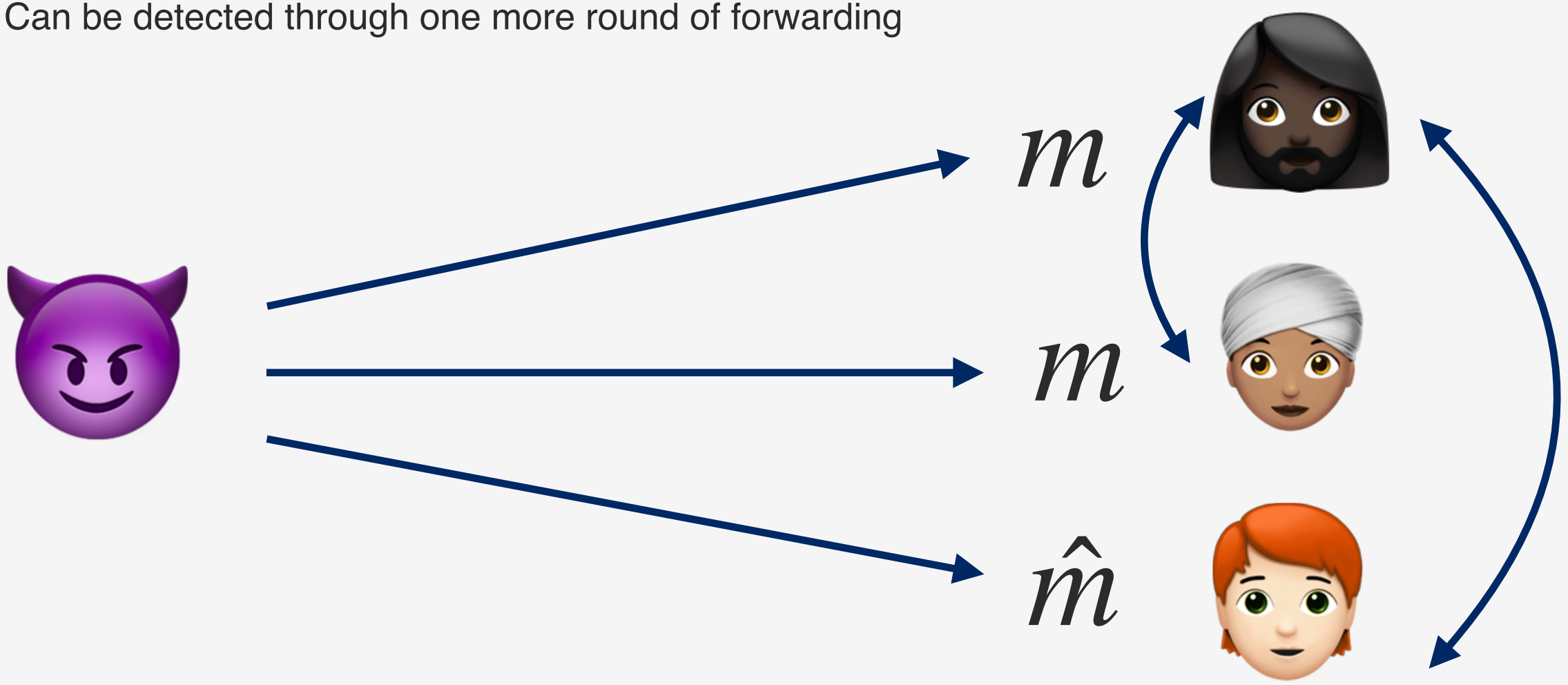
Detecting Equivocation is Easy

- **Equivocation:** Sender S sends different messages $m, \hat{m}$ to parties
- Can be detected through one more round of forwarding



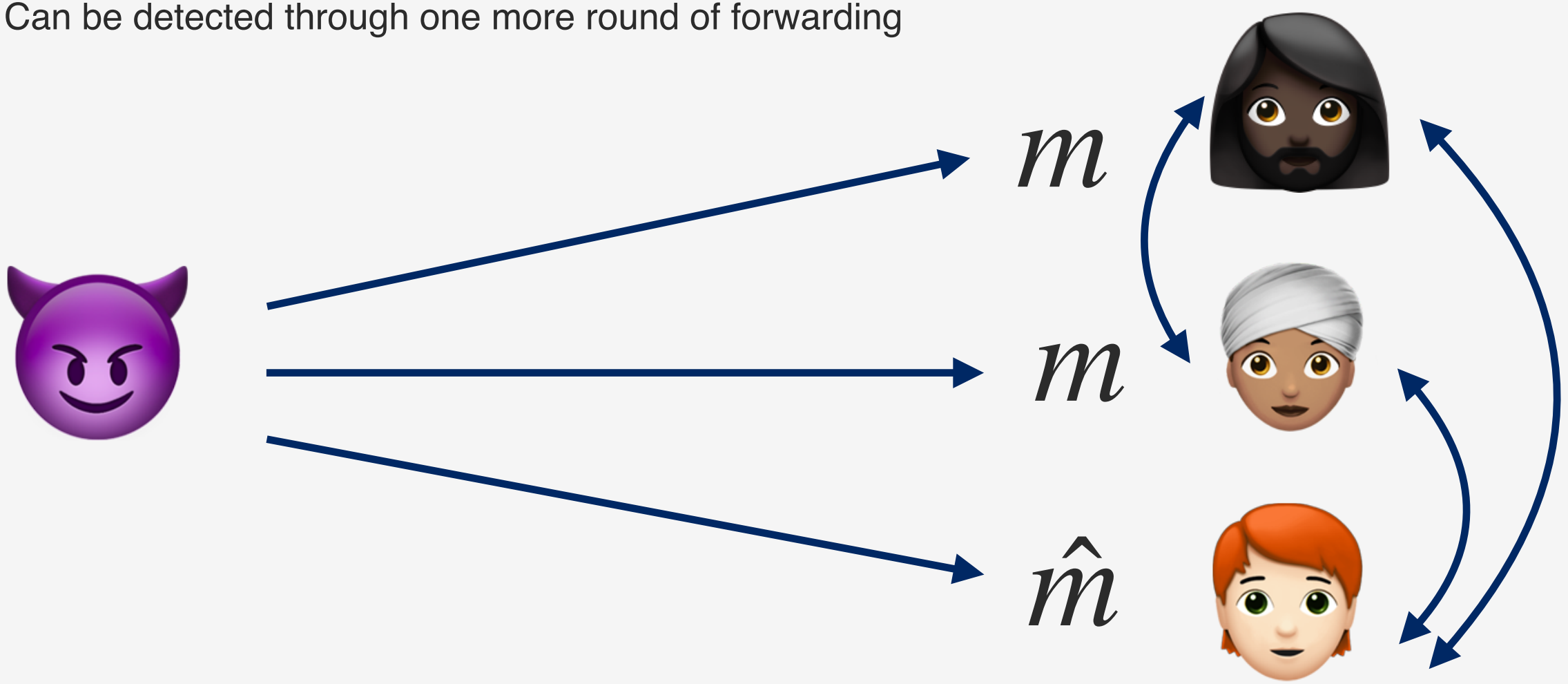
Detecting Equivocation is Easy

- **Equivocation:** Sender S sends different messages $m, \hat{m}$ to parties
- Can be detected through one more round of forwarding



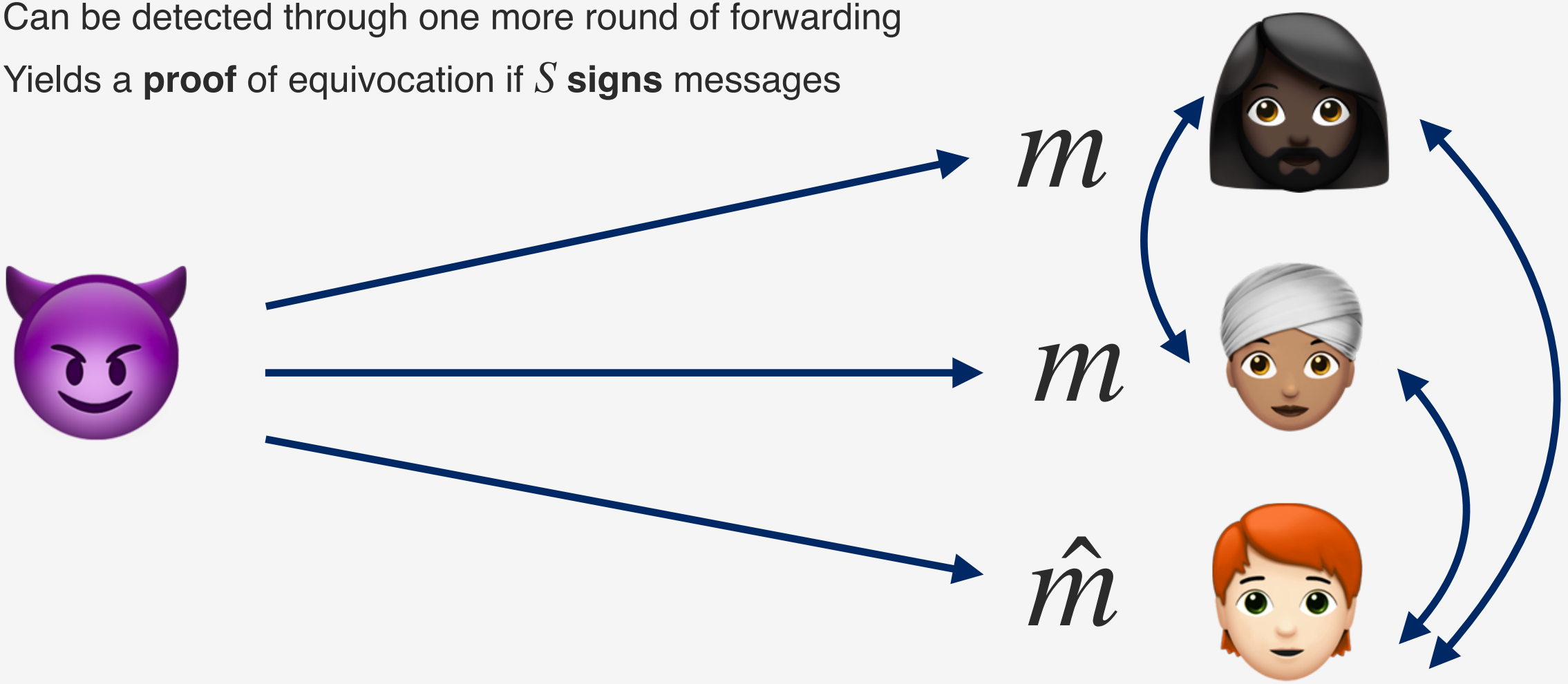
Detecting Equivocation is Easy

- **Equivocation:** Sender S sends different messages $m, \hat{m}$ to parties
- Can be detected through one more round of forwarding



Detecting Equivocation is Easy

- **Equivocation:** Sender S sends different messages $m, \hat{m}$ to parties
- Can be detected through one more round of forwarding
- Yields a **proof** of equivocation if S **signs** messages



Detecting Sender Crash is Hard

Detecting Sender Crash is Hard

- How to prove that sender sent no message?

Detecting Sender Crash is Hard

- How to prove that sender sent no message?
- Easy when $t < n/2$:

Detecting Sender Crash is Hard

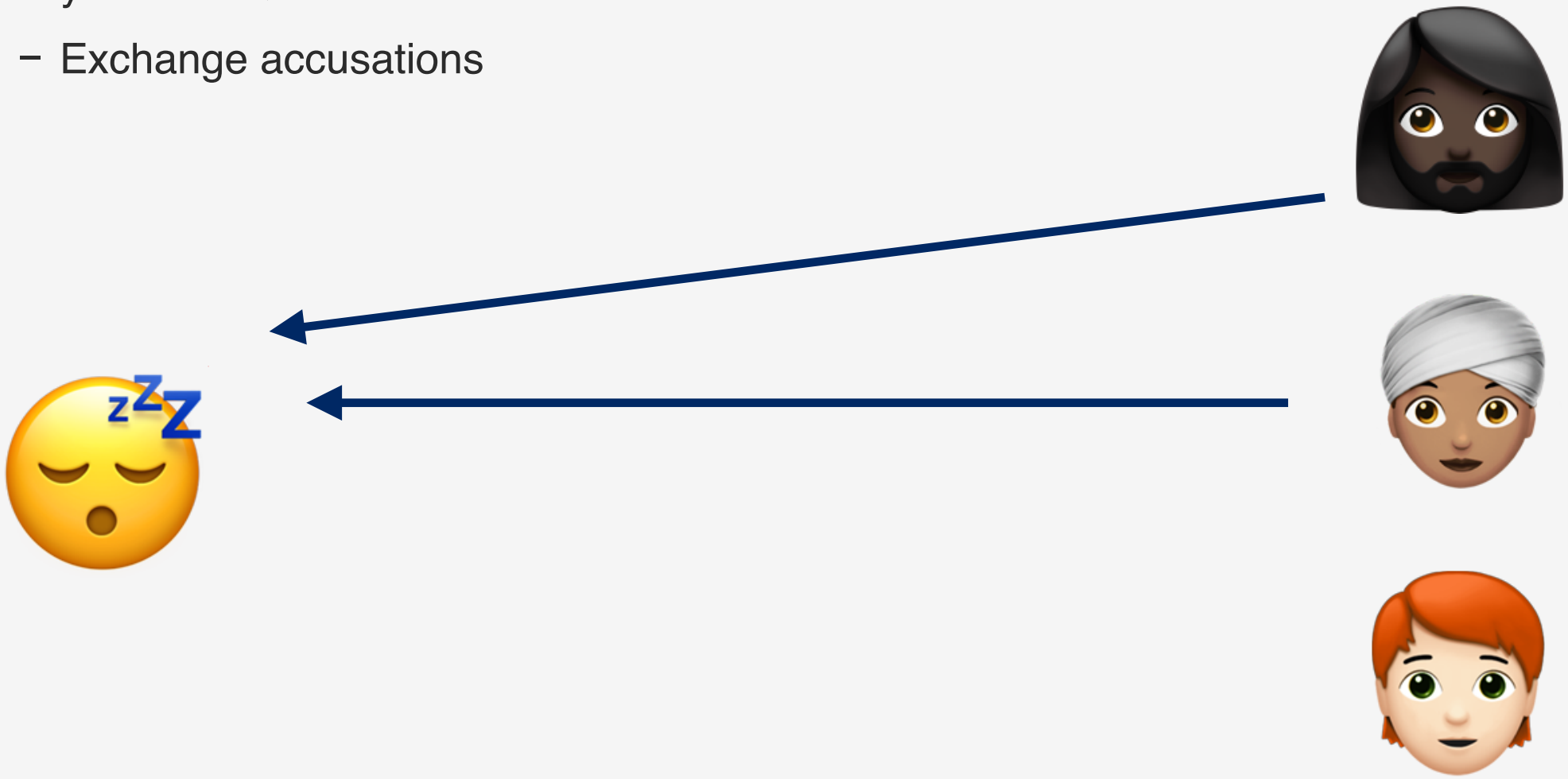
- How to prove that sender sent no message?
- Easy when $t < n/2$:
  - Exchange accusations

Detecting Sender Crash is Hard

- How to prove that sender sent no message?
- Easy when $t < n/2$:
  - Exchange accusations

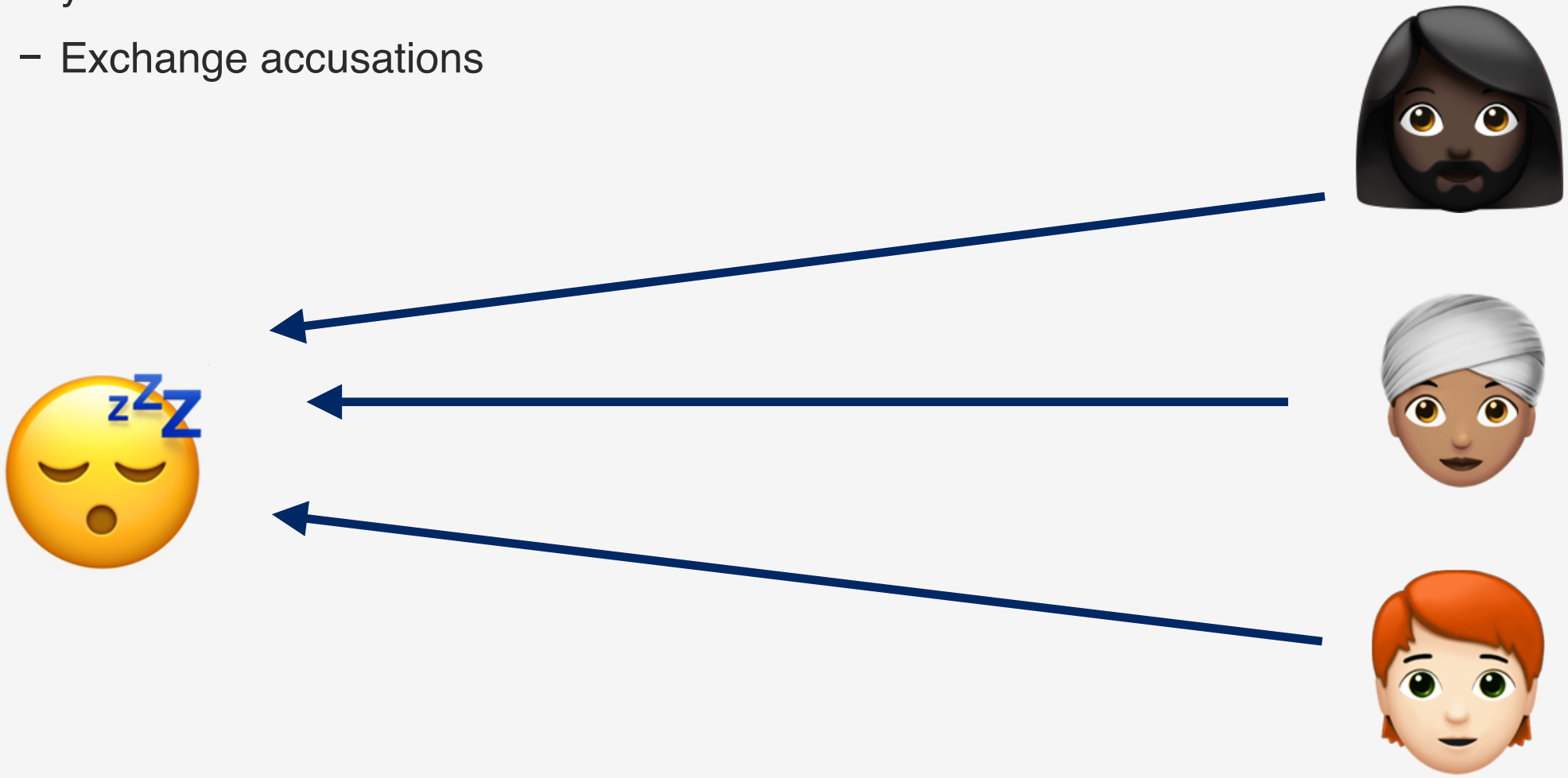
Detecting Sender Crash is Hard

- How to prove that sender sent no message?
- Easy when $t < n/2$:
  - Exchange accusations



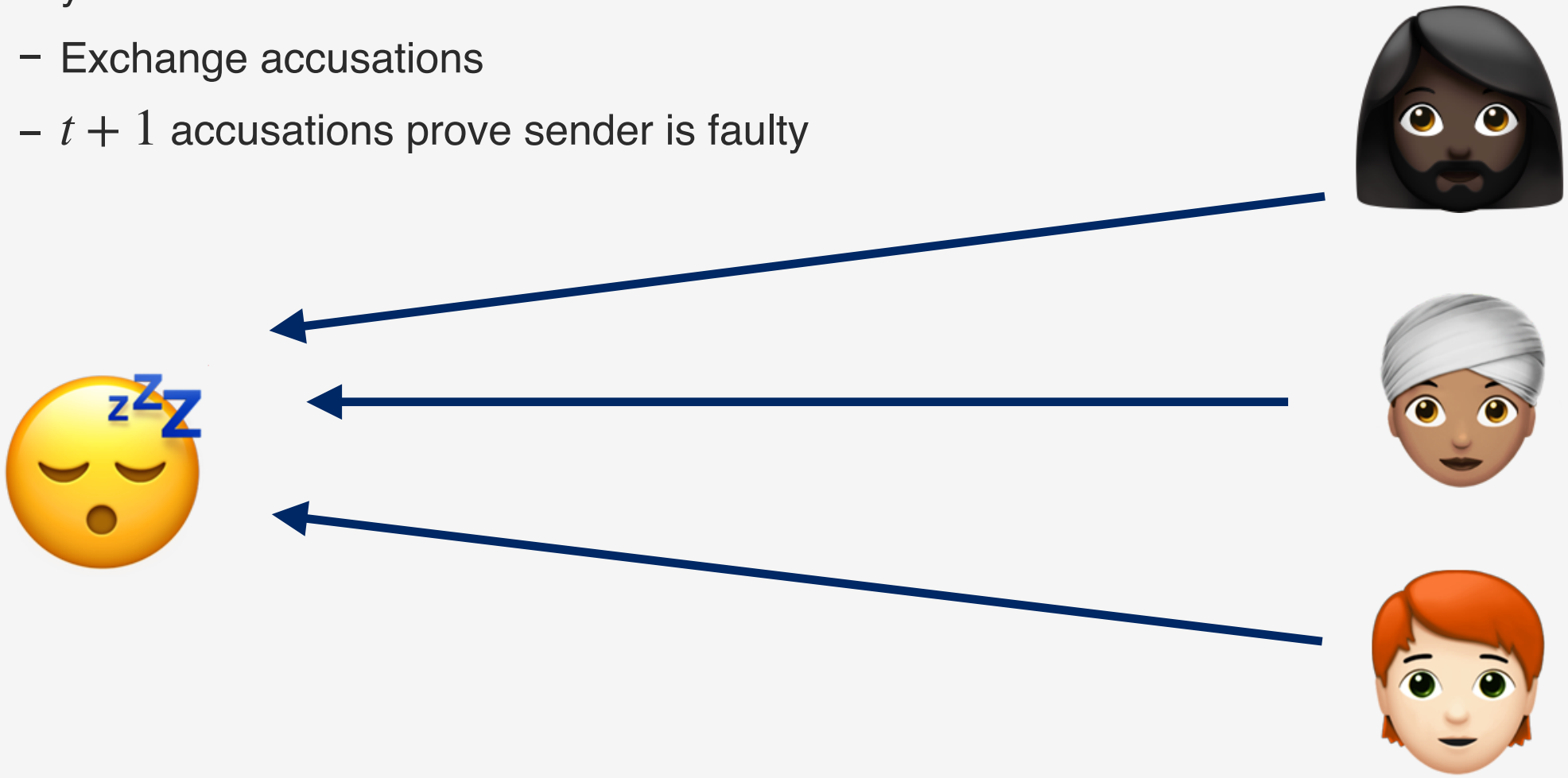
Detecting Sender Crash is Hard

- How to prove that sender sent no message?
- Easy when $t < n/2$:
  - Exchange accusations



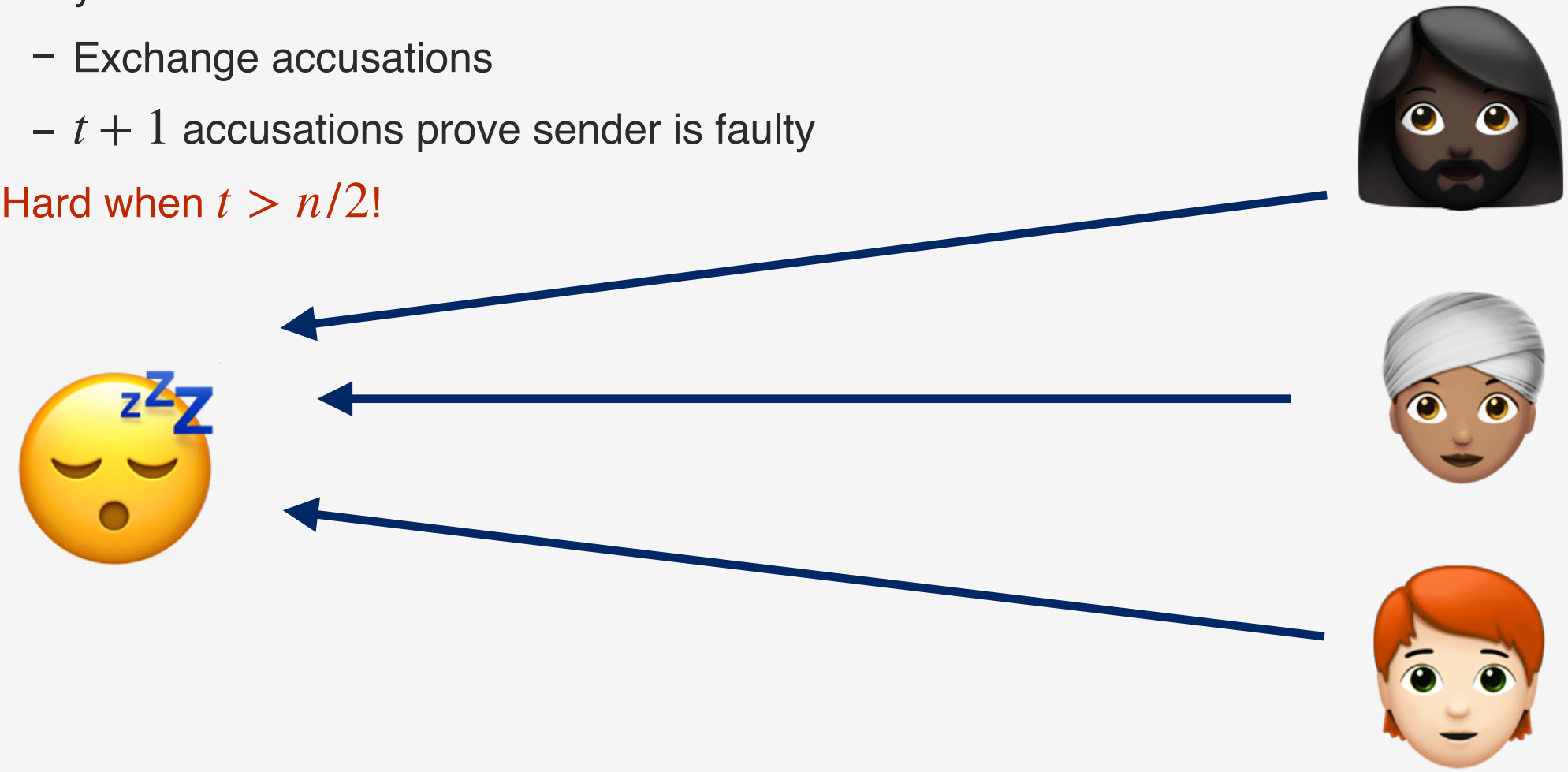
Detecting Sender Crash is Hard

- How to prove that sender sent no message?
- Easy when $t < n/2$:
  - Exchange accusations
  - $t + 1$ accusations prove sender is faulty



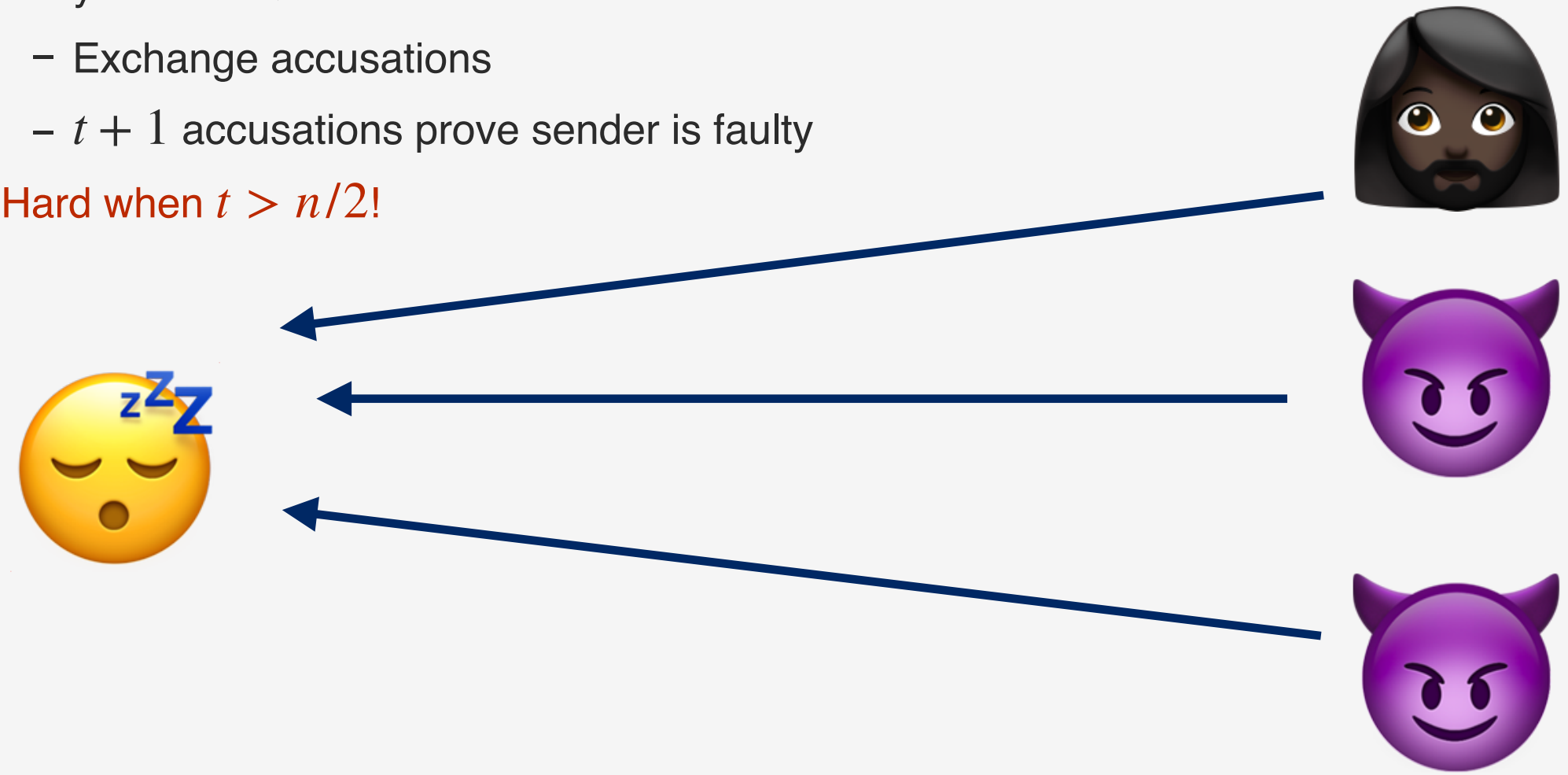
Detecting Sender Crash is Hard

- How to prove that sender sent no message?
- Easy when $t < n/2$:
  - Exchange accusations
  - $t + 1$ accusations prove sender is faulty
- Hard when $t > n/2$!



Detecting Sender Crash is Hard

- How to prove that sender sent no message?
- Easy when $t < n/2$:
  - Exchange accusations
  - $t + 1$ accusations prove sender is faulty
- Hard when $t > n/2$!



Main Tool: Polarisers

Main Tool: Polarisers

- Act as `certificates' when $t > n/2$

Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt

Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt

Corrupt

Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt

Alive

Corrupt

Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt
- **All** parties in Alive **accuse** parties in Corrupt

Alive

Corrupt

Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt
- **All** parties in Alive **accuse** parties in Corrupt

Alive

Corrupt

Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt
- **All** parties in Alive **accuse** parties in Corrupt

Alive

Corrupt

Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt
- **All** parties in Alive **accuse** parties in Corrupt

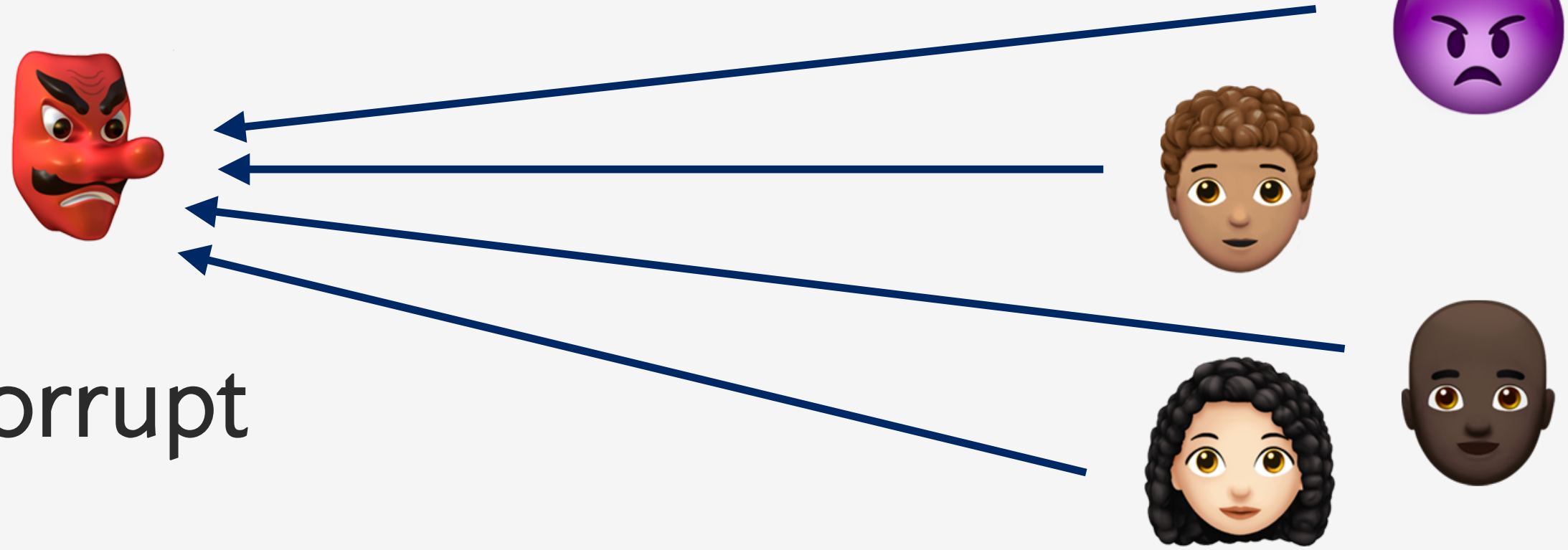
Alive

Corrupt

Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt
- **All** parties in Alive **accuse** parties in Corrupt

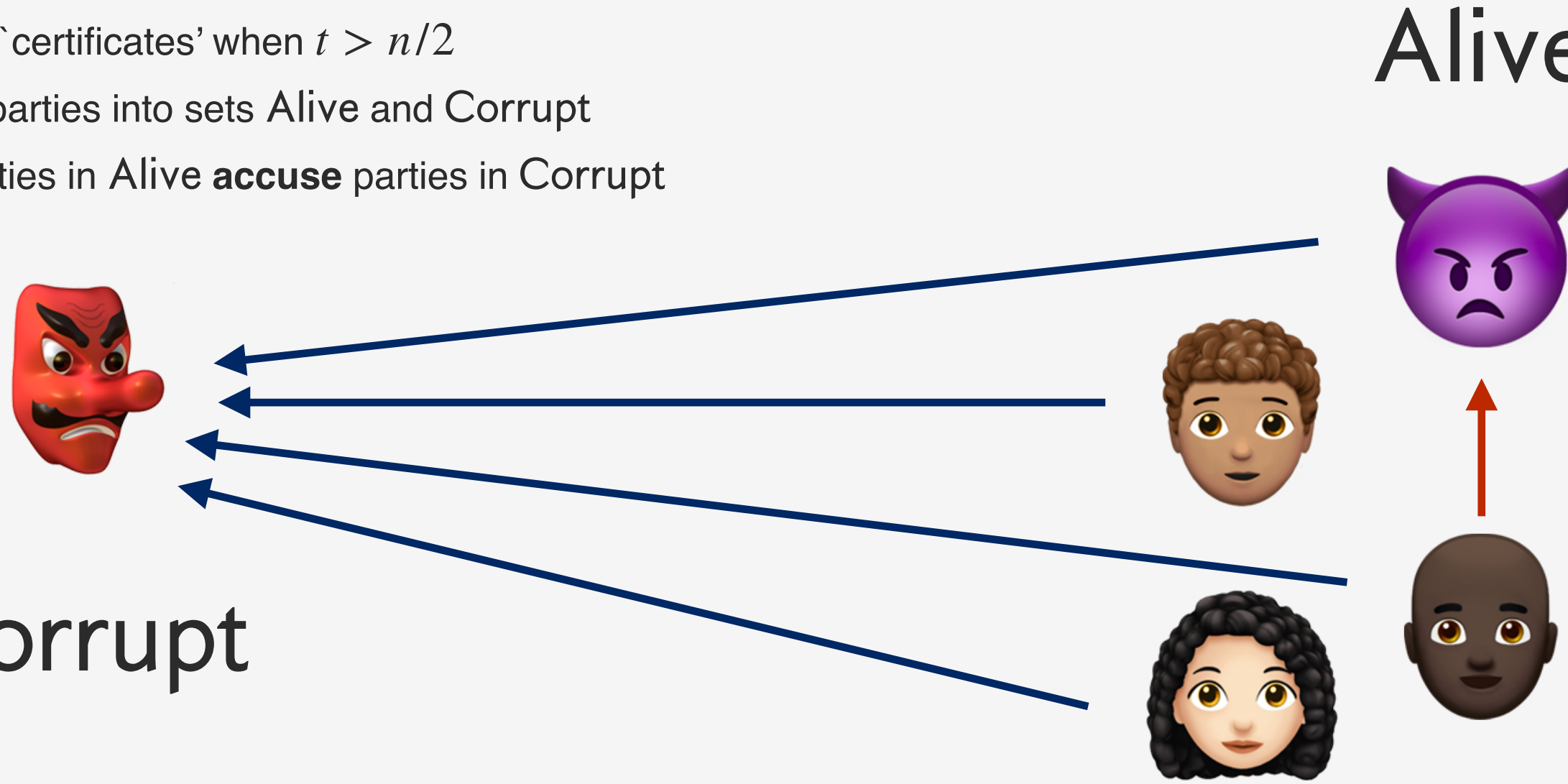
Alive



Corrupt

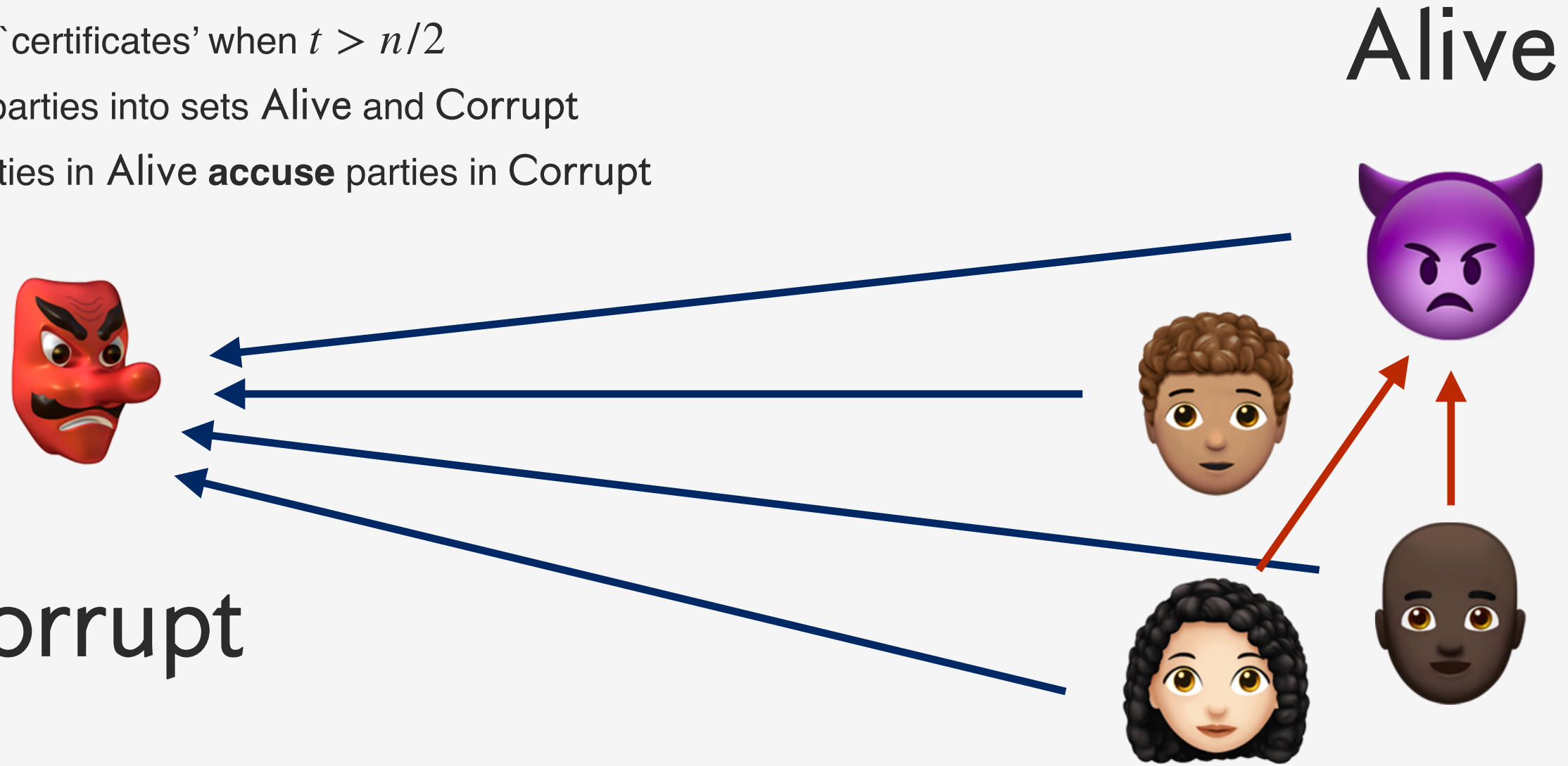
Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt
- **All** parties in Alive **accuse** parties in Corrupt



Main Tool: Polarisers

- Act as `certificates' when $t > n/2$
- Splits parties into sets Alive and Corrupt
- **All** parties in Alive **accuse** parties in Corrupt



Polariser Features

Polariser Features

- **Invariant:** Honest Parties do not accuse each other

Polariser Features

- **Invariant:** Honest Parties do not accuse each other
- **Key Observation:** Either honest parties are **all in** Alive or **all in** Corrupt

Polariser Features

- **Invariant:** Honest Parties do not accuse each other
- **Key Observation:** Either honest parties are **all in** Alive or **all in** Corrupt

Alive

Corrupt

Polariser Features

- **Invariant:** Honest Parties do not accuse each other
- **Key Observation:** Either honest parties are **all in** Alive or **all in** Corrupt

Alive

Corrupt

Polariser Features

- **Invariant:** Honest Parties do not accuse each other
- **Key Observation:** Either honest parties are **all in** Alive or **all in** Corrupt

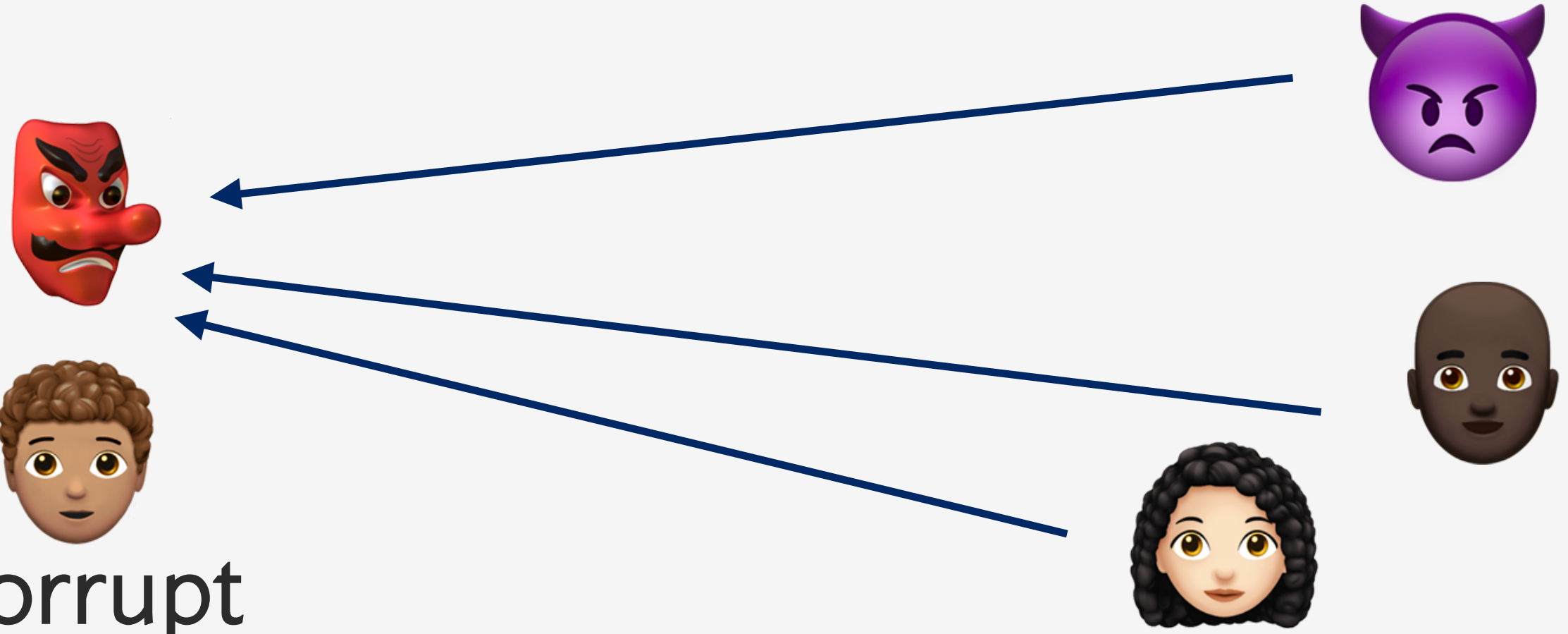
Alive

Corrupt

Polariser Features

- **Invariant:** Honest Parties do not accuse each other
- **Key Observation:** Either honest parties are **all in** Alive or **all in** Corrupt

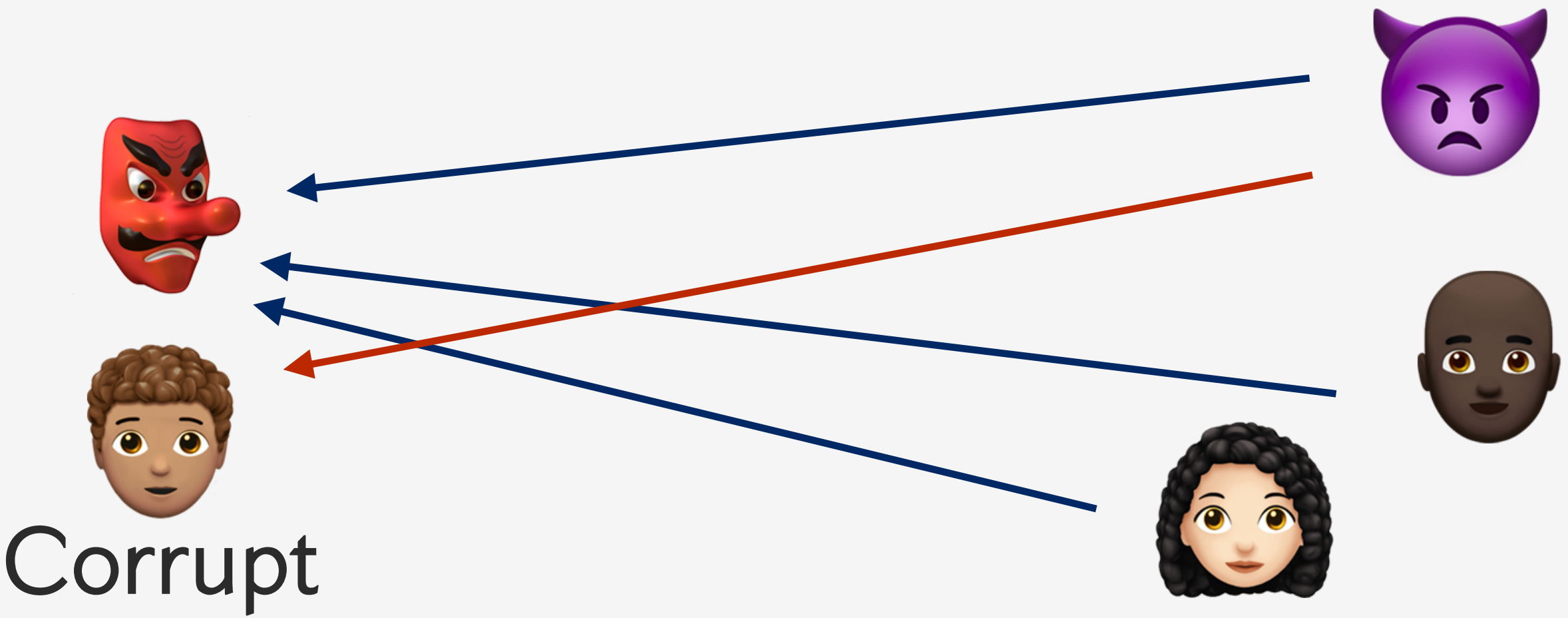
Alive



Corrupt

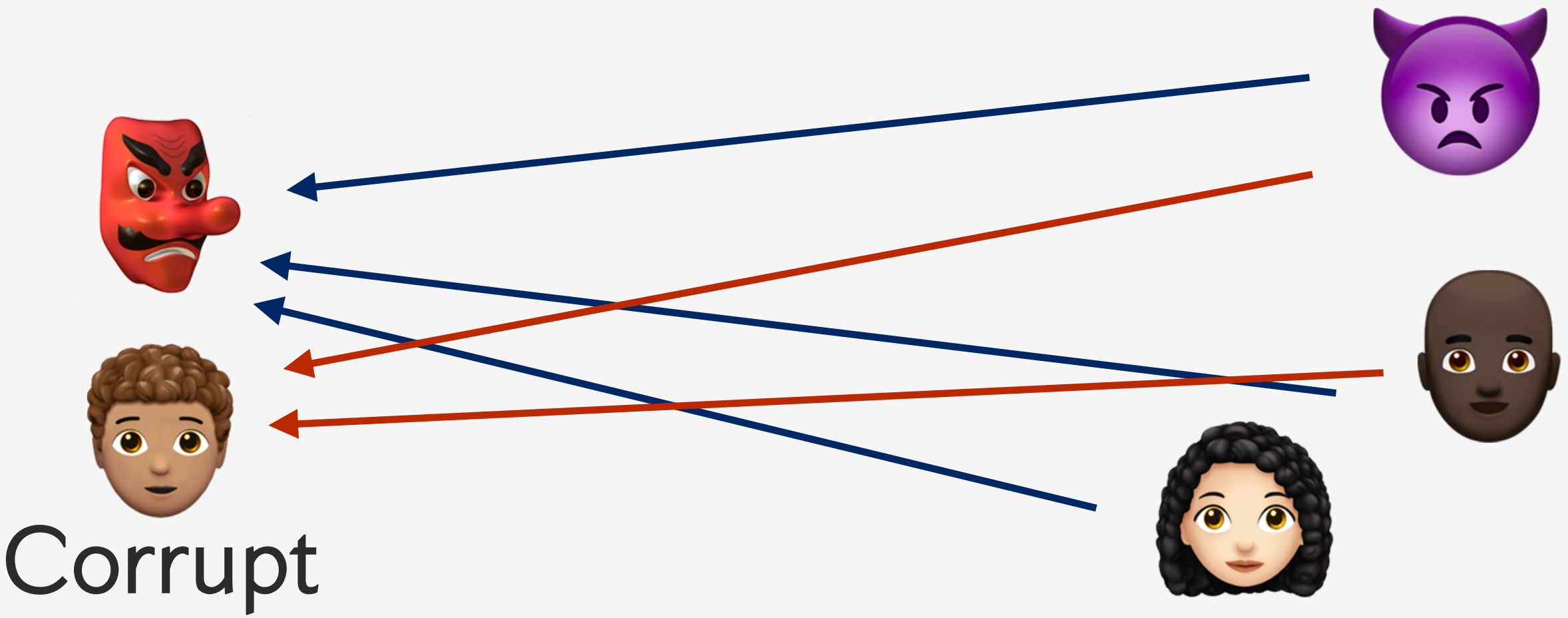
Polariser Features

- **Invariant:** Honest Parties do not accuse each other
- **Key Observation:** Either honest parties are **all in** Alive or **all in** Corrupt



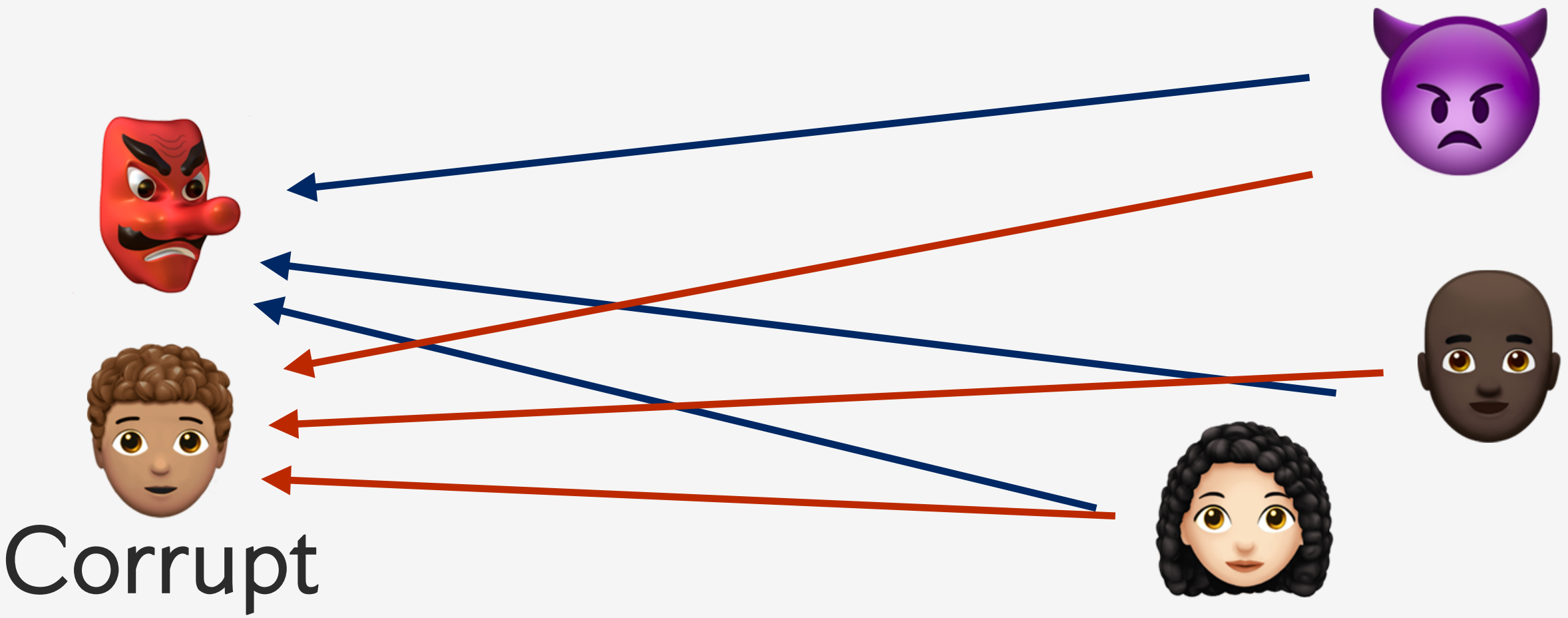
Polariser Features

- **Invariant:** Honest Parties do not accuse each other
- **Key Observation:** Either honest parties are **all in** Alive or **all in** Corrupt



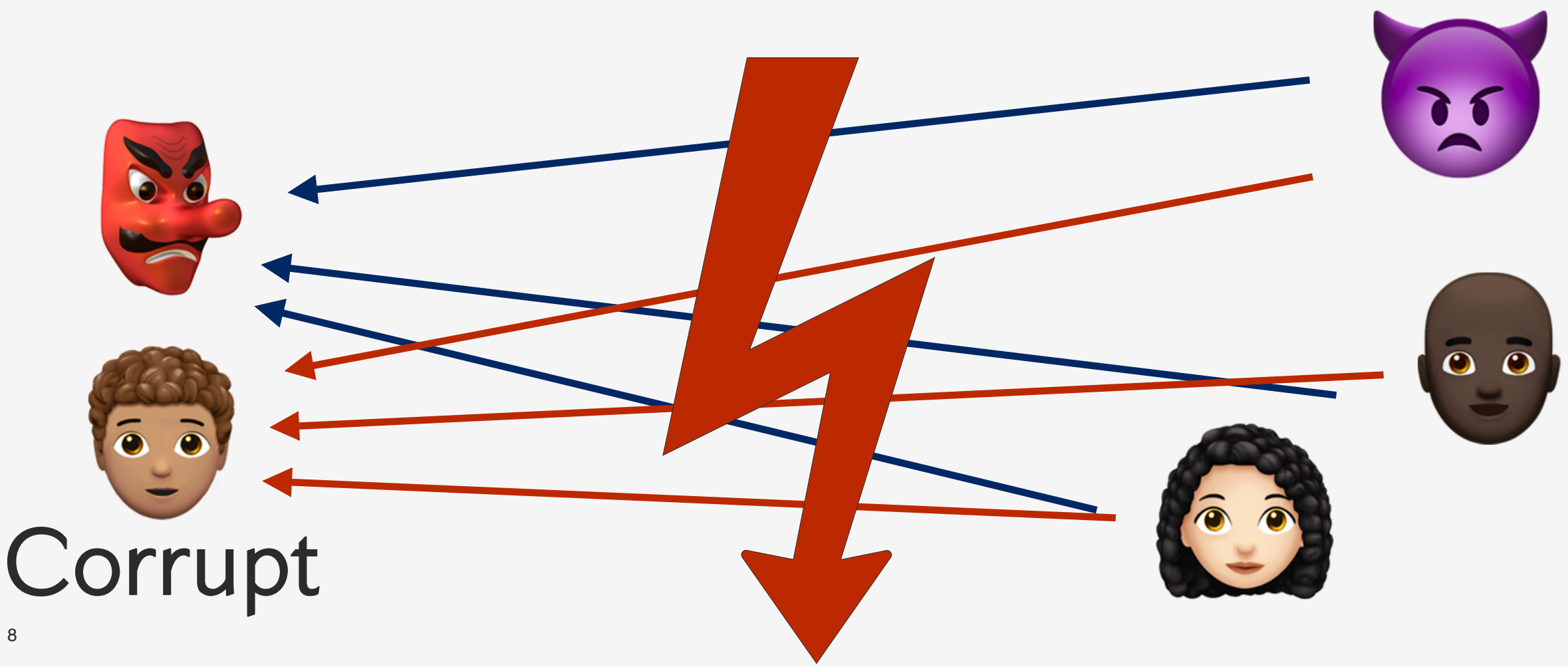
Polariser Features

- **Invariant:** Honest Parties do not accuse each other
- **Key Observation:** Either honest parties are **all in** Alive or **all in** Corrupt



Polariser Features

- **Invariant:** Honest Parties do not accuse each other
- **Key Observation:** Either honest parties are **all in** Alive or **all in** Corrupt



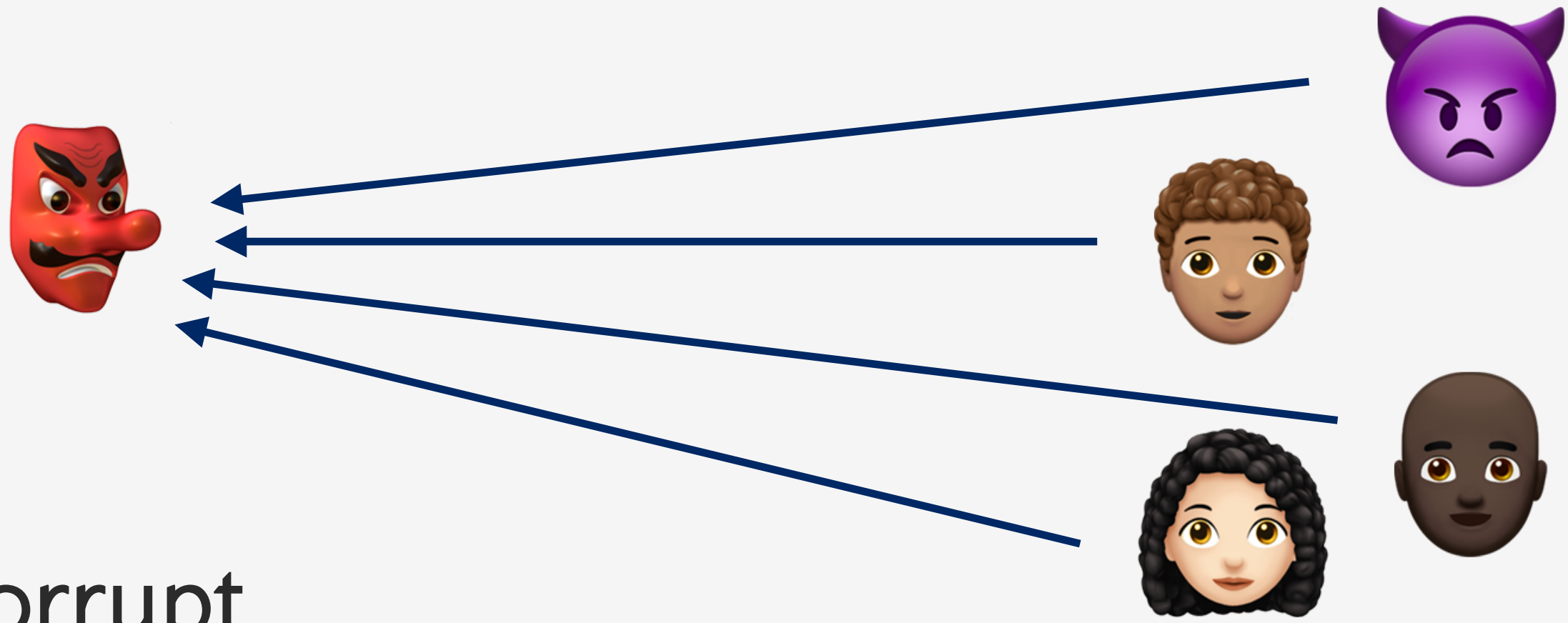
Updating Polarisers

Updating Polarisers

- Update polariser with new accusations

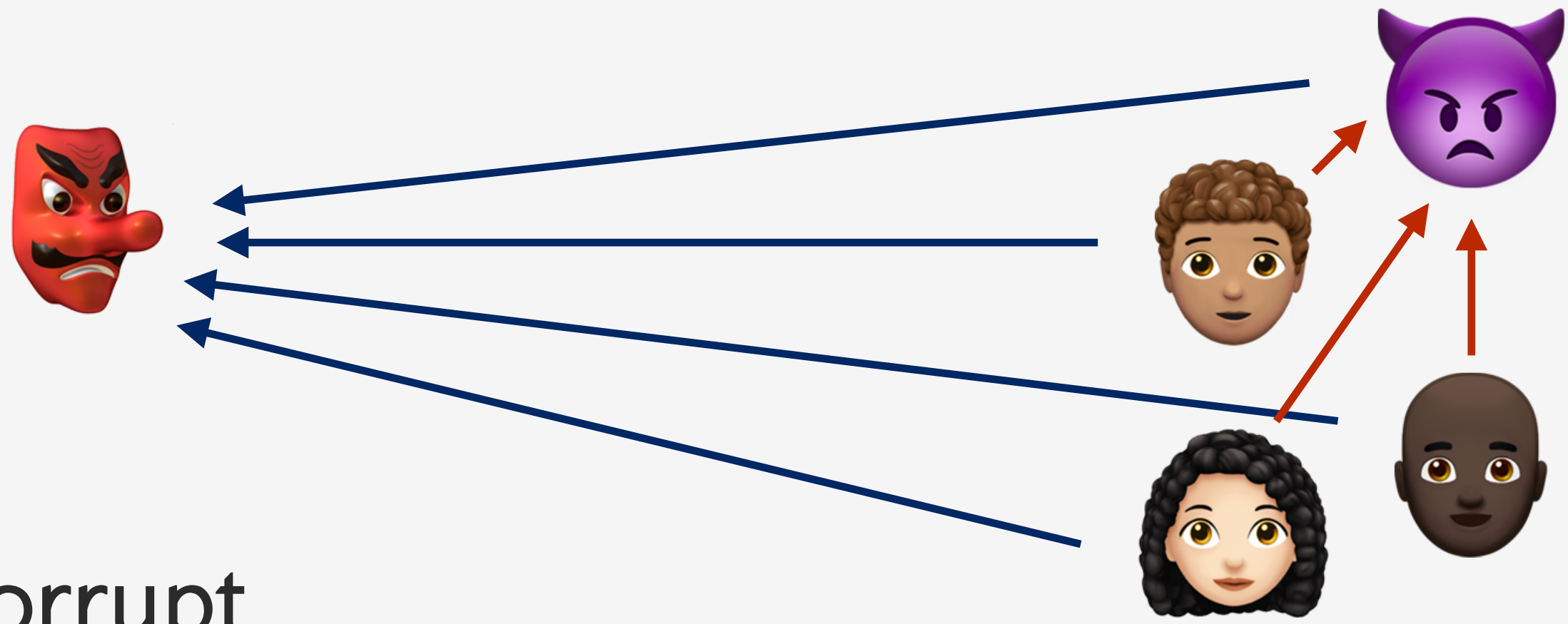
Updating Polarisers

- Update polariser with new accusations



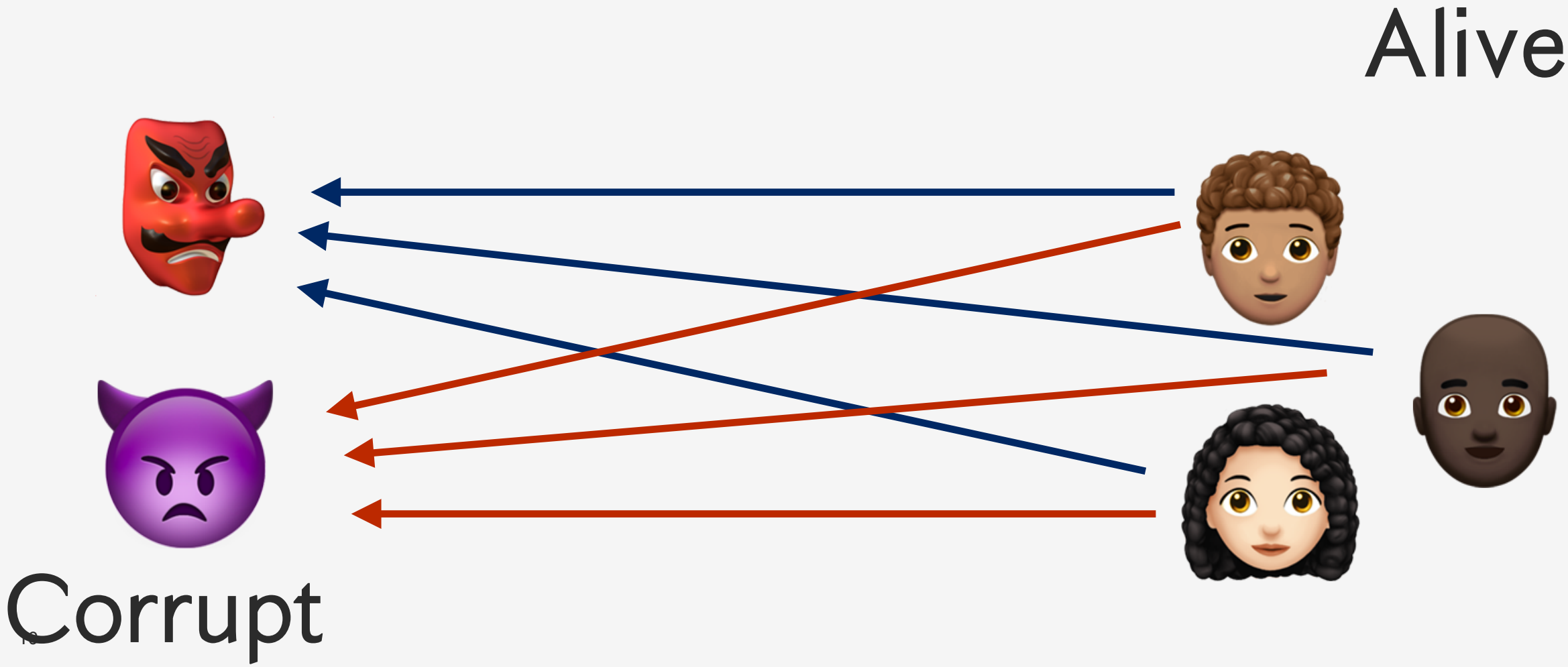
Updating Polarisers

- Update polariser with new accusations



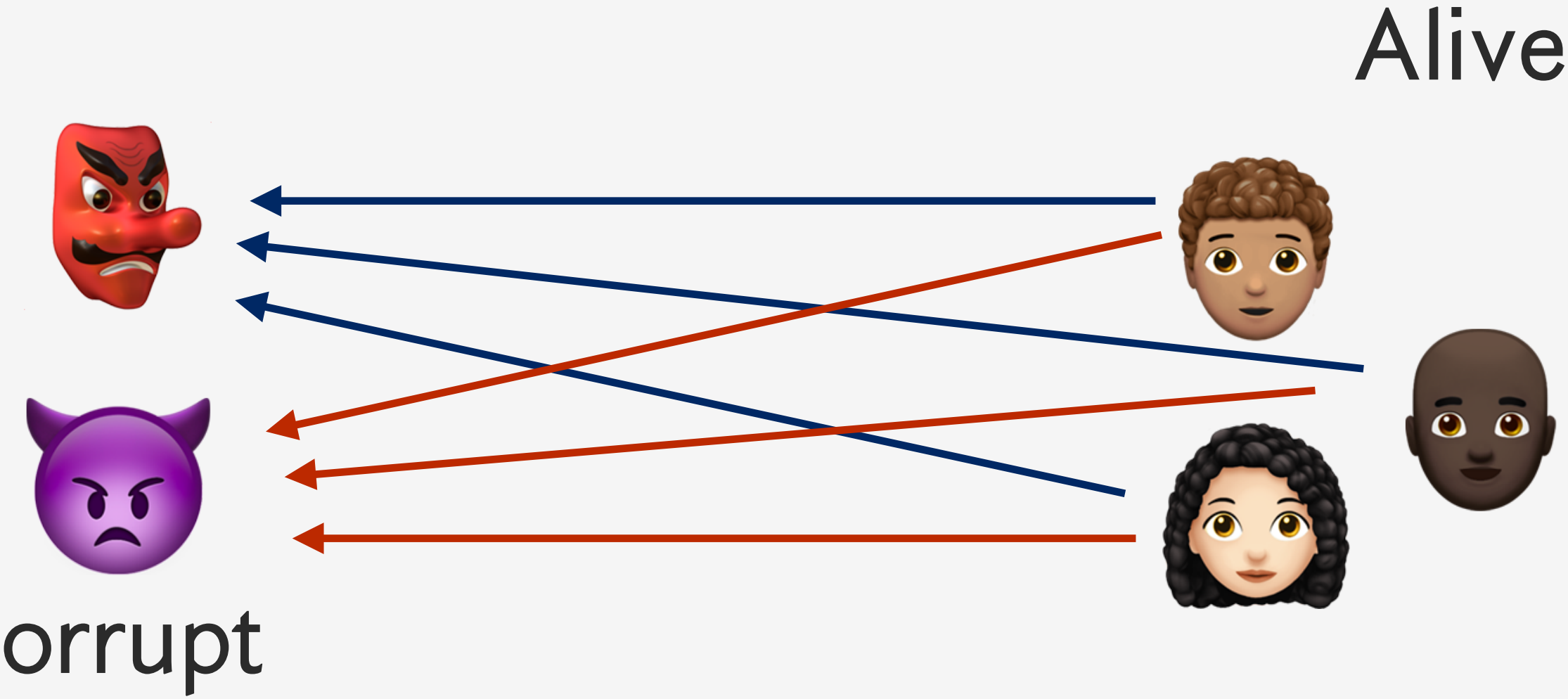
Updating Polarisers

- Update polariser with new accusations



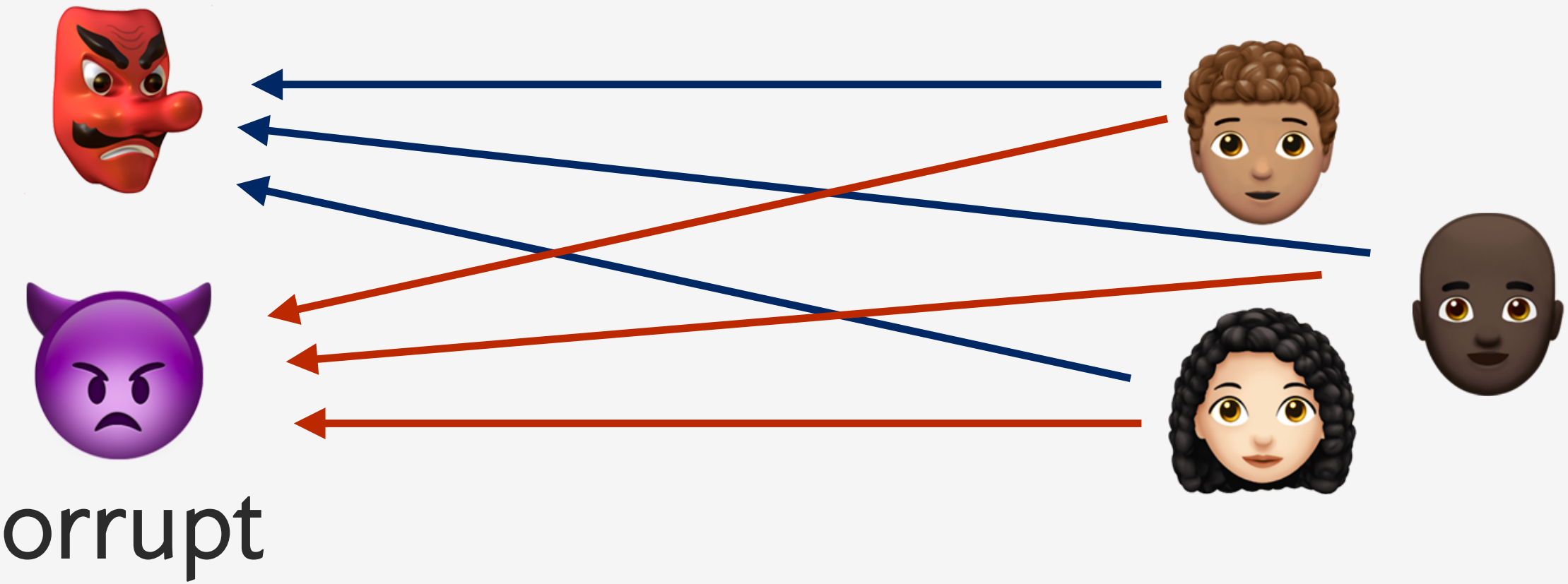
Updating Polarisers

- Update polariser with new accusations
- **Rule:** P only accepts polariser if $P \in$ Alive

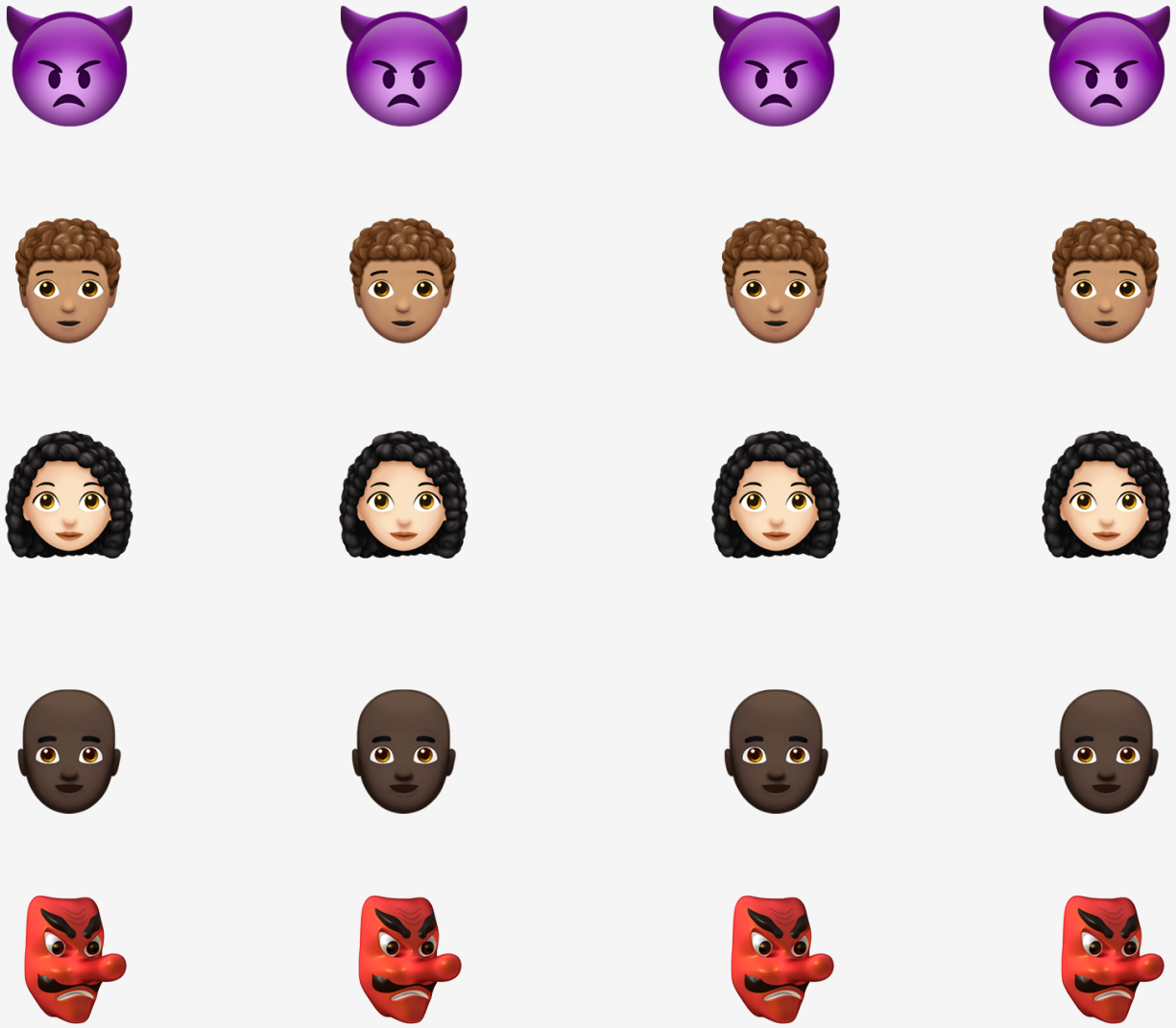


Updating Polarisers

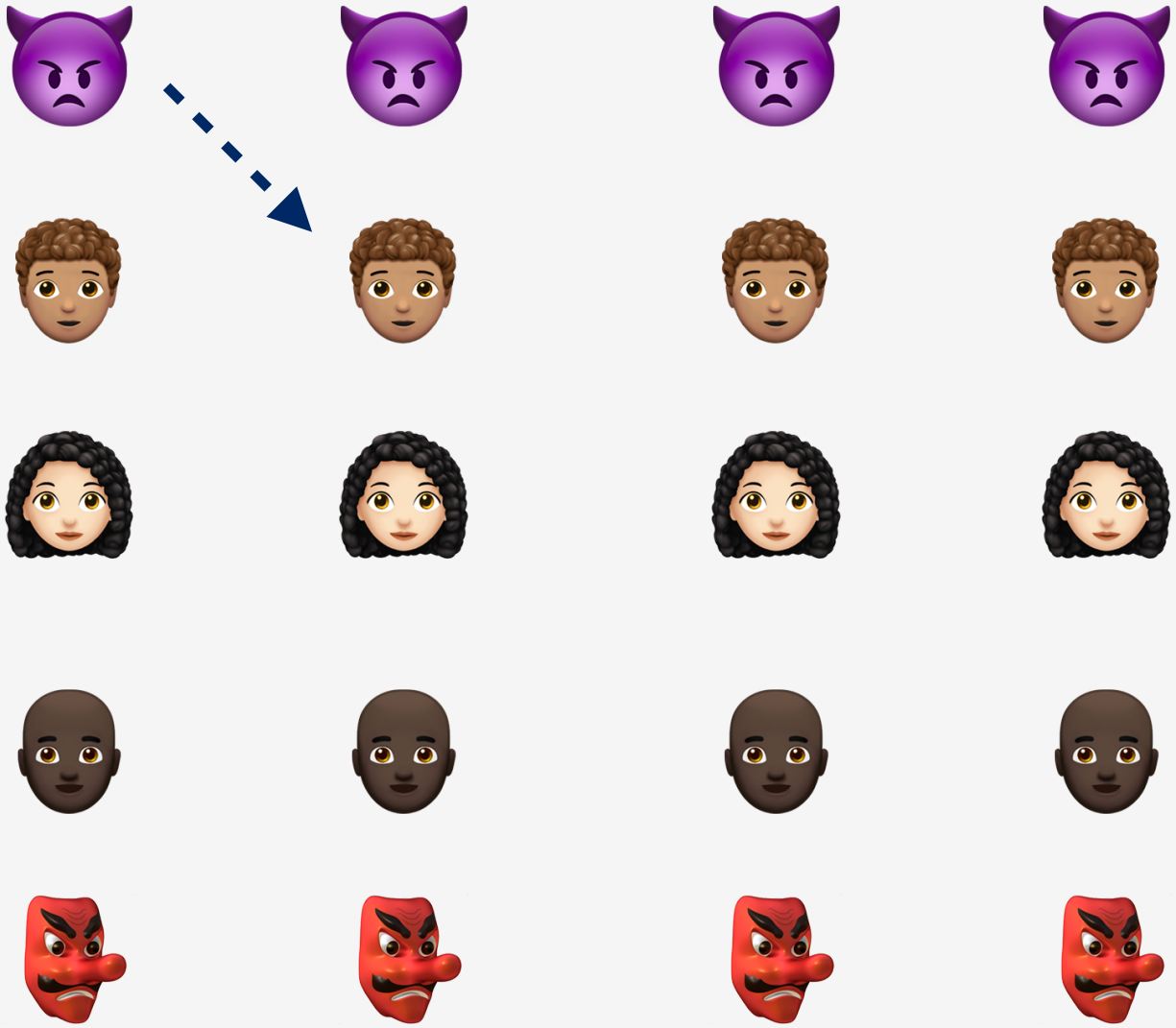
- Update polariser with new accusations
- **Rule:** P only accepts polariser if $P \in$ Alive
- $\Longrightarrow$ Polarisers can be **forwarded/updated** safely between honest parties!



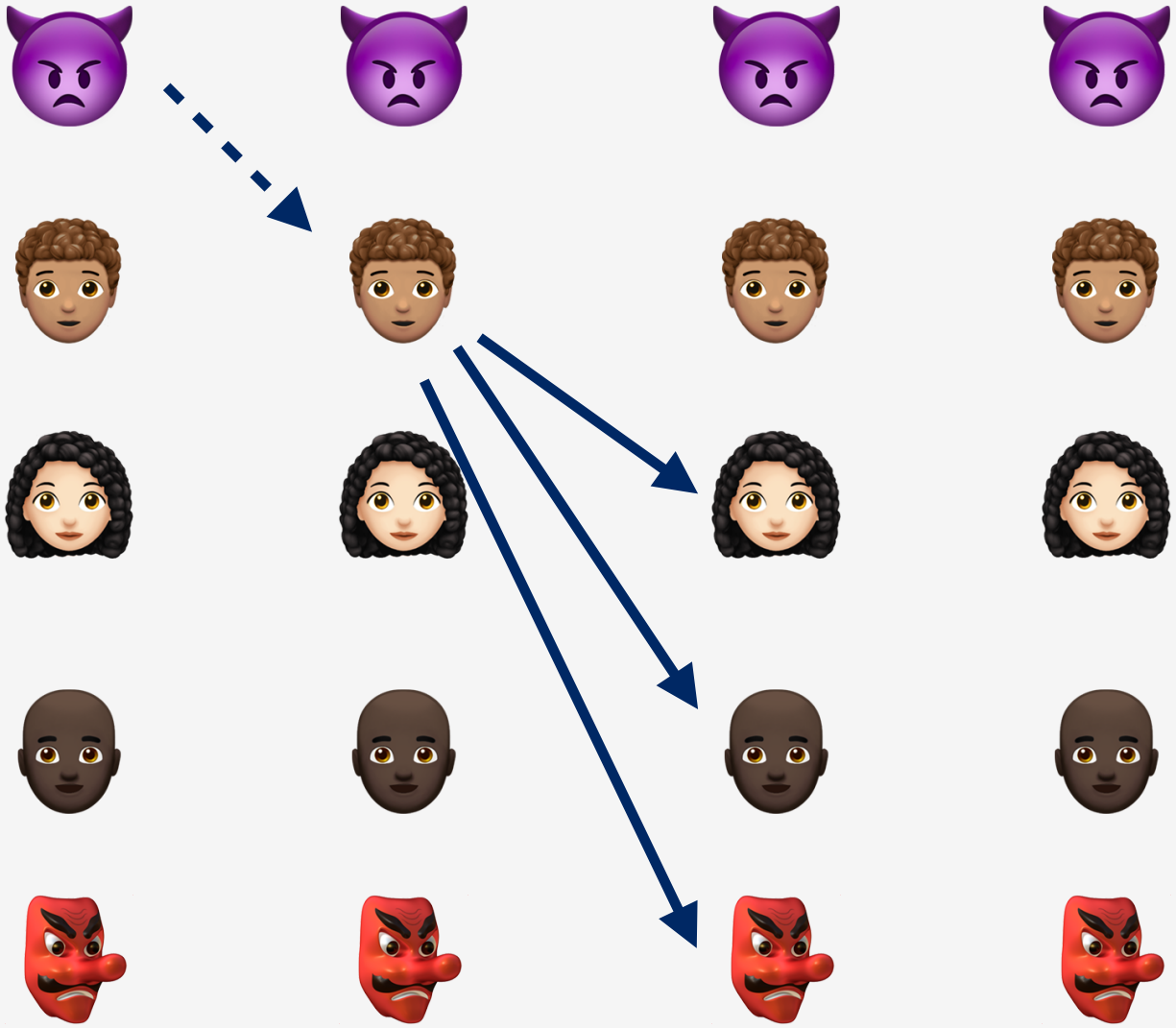
Constructing a Polariser in $f + 2$ Rounds, Example 1



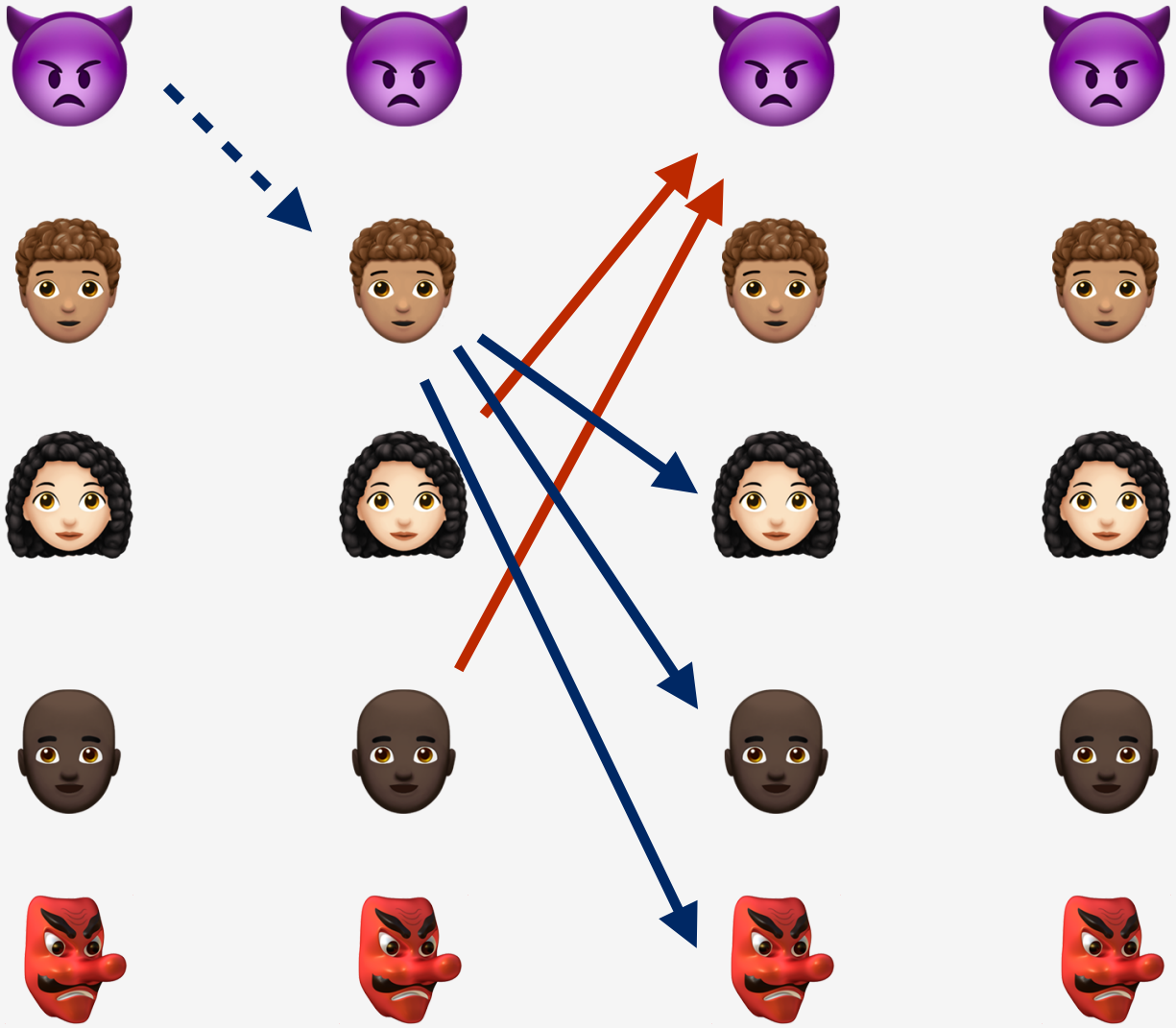
Constructing a Polariser in $f + 2$ Rounds, Example 1



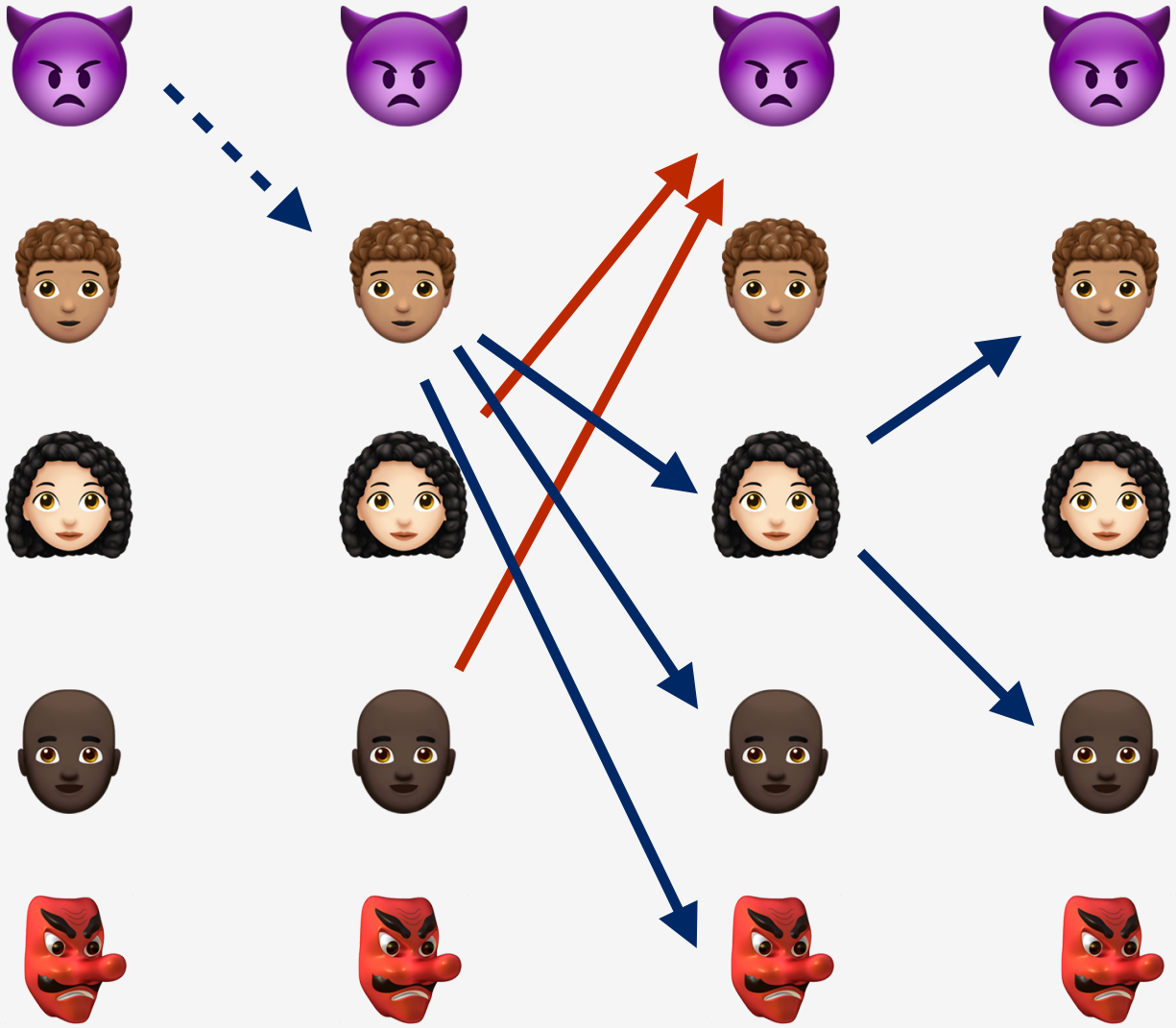
Constructing a Polariser in $f + 2$ Rounds, Example 1



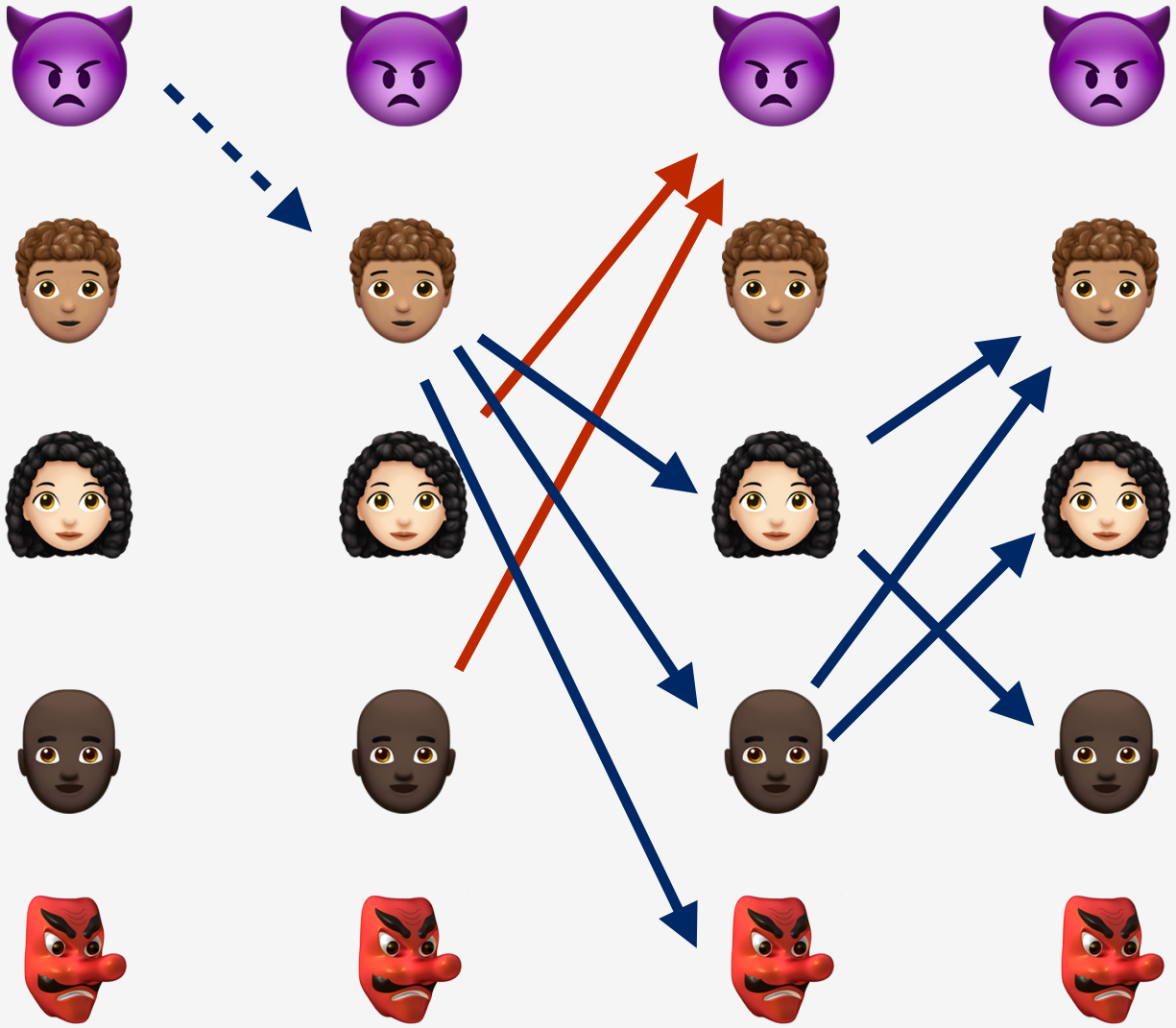
Constructing a Polariser in $f + 2$ Rounds, Example 1



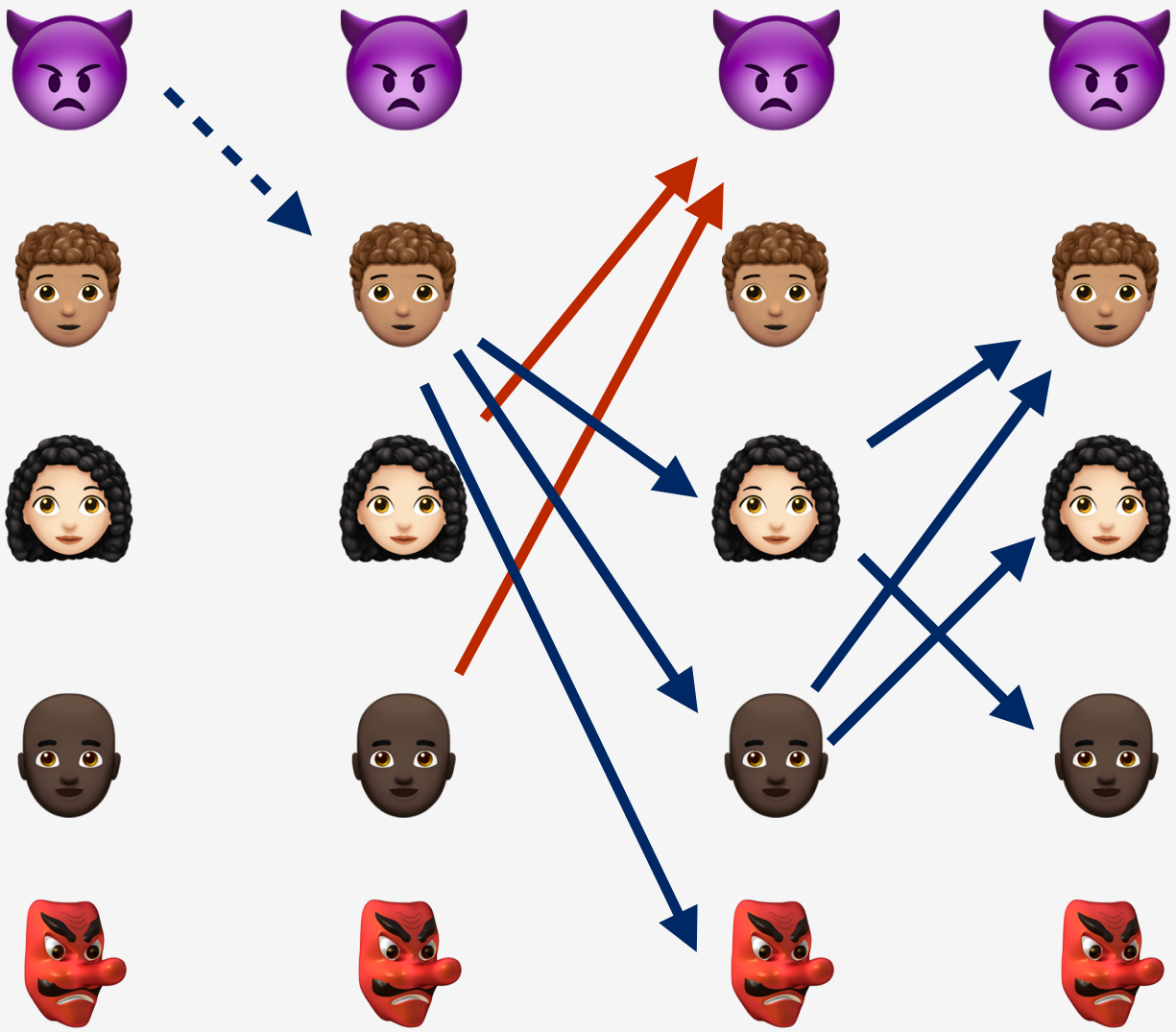
Constructing a Polariser in $f + 2$ Rounds, Example 1



Constructing a Polariser in $f + 2$ Rounds, Example 1

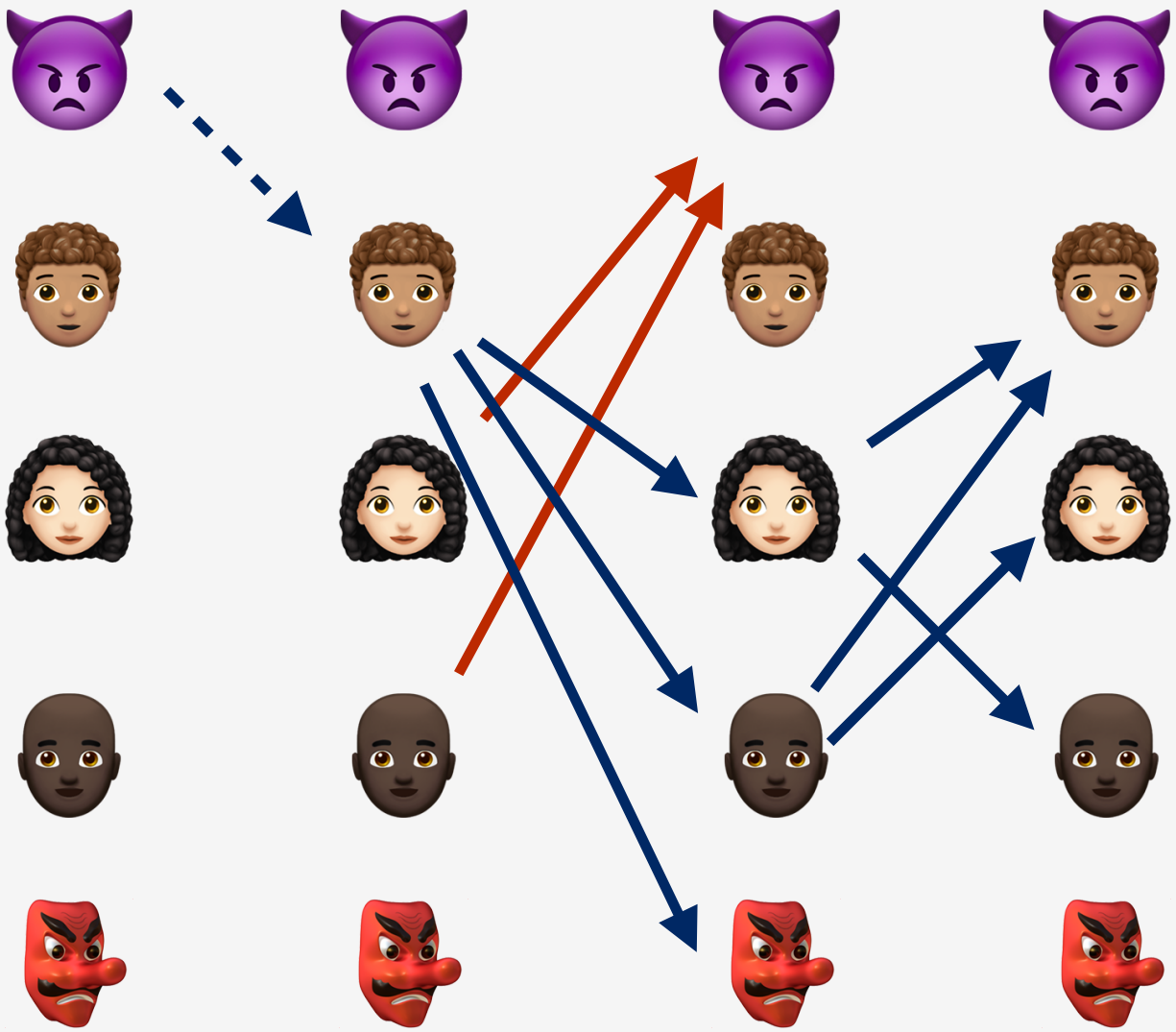


Constructing a Polariser in $f + 2$ Rounds, Example 1



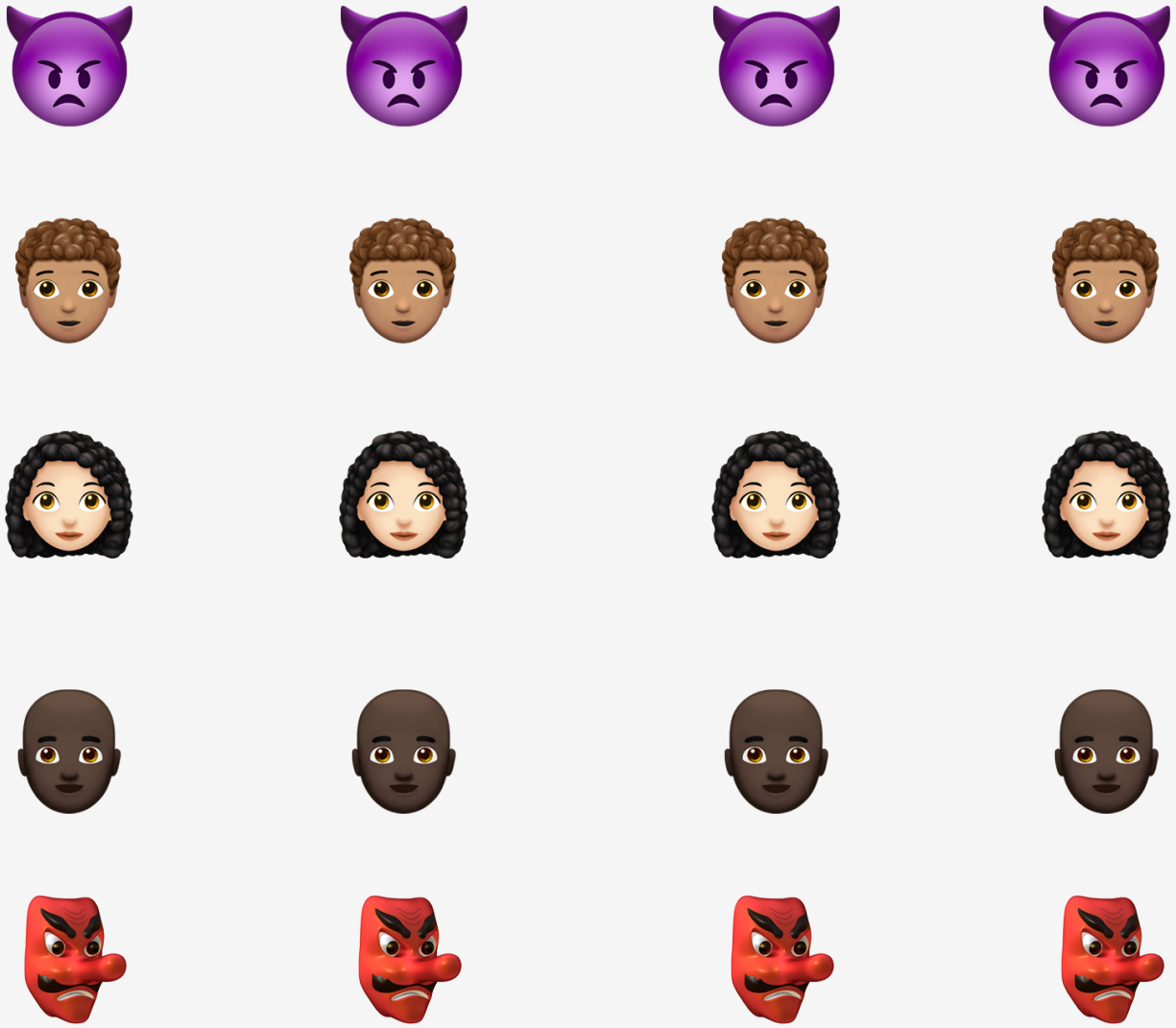
- All parties output P_1's message by Round 4

Constructing a Polariser in $f + 2$ Rounds, Example 1

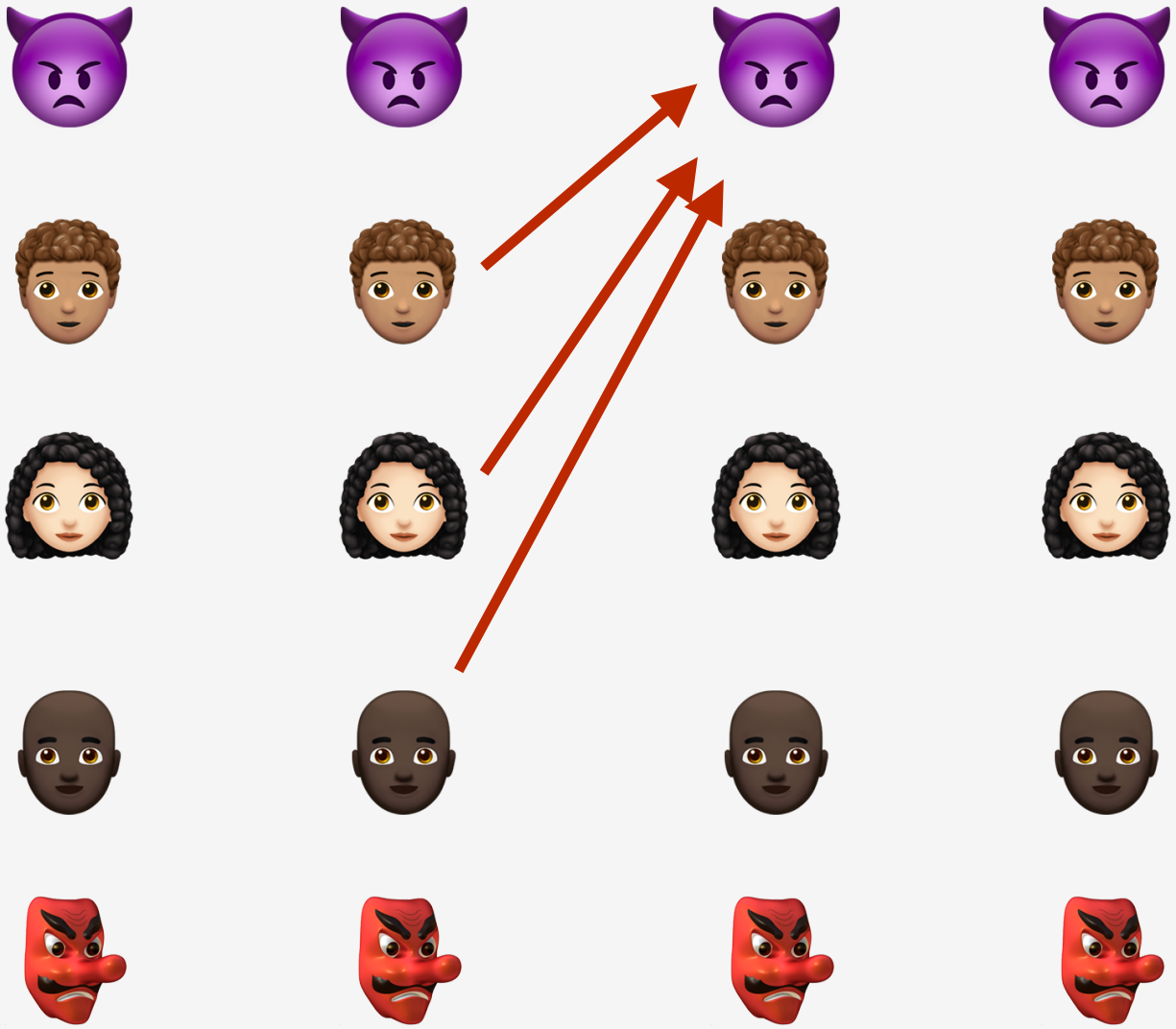


- All parties output P_1's message by Round 4
- No malicious party identified

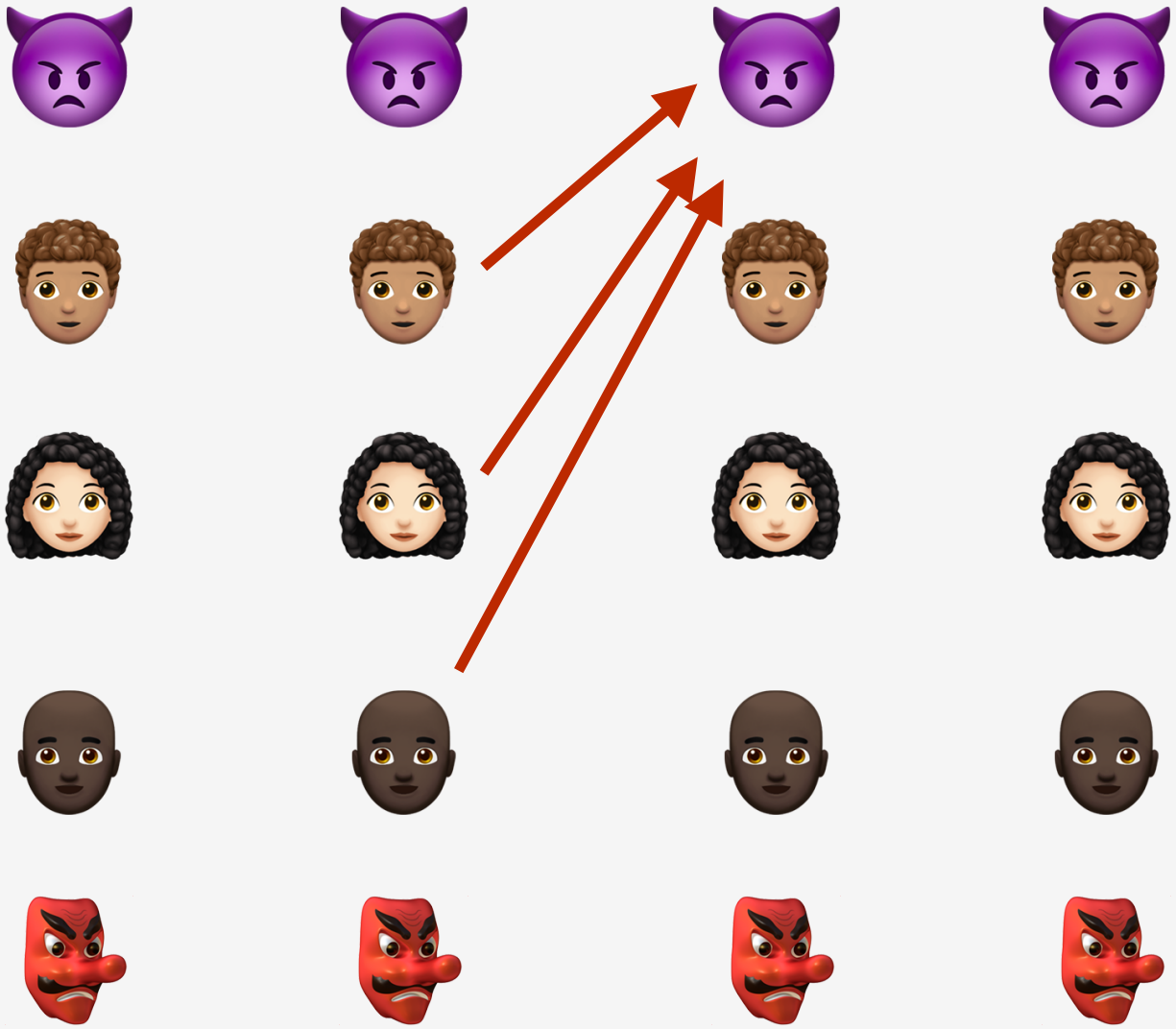
Constructing a Polariser in $f + 2$ Rounds, Example 2



Constructing a Polariser in $f + 2$ Rounds, Example 2

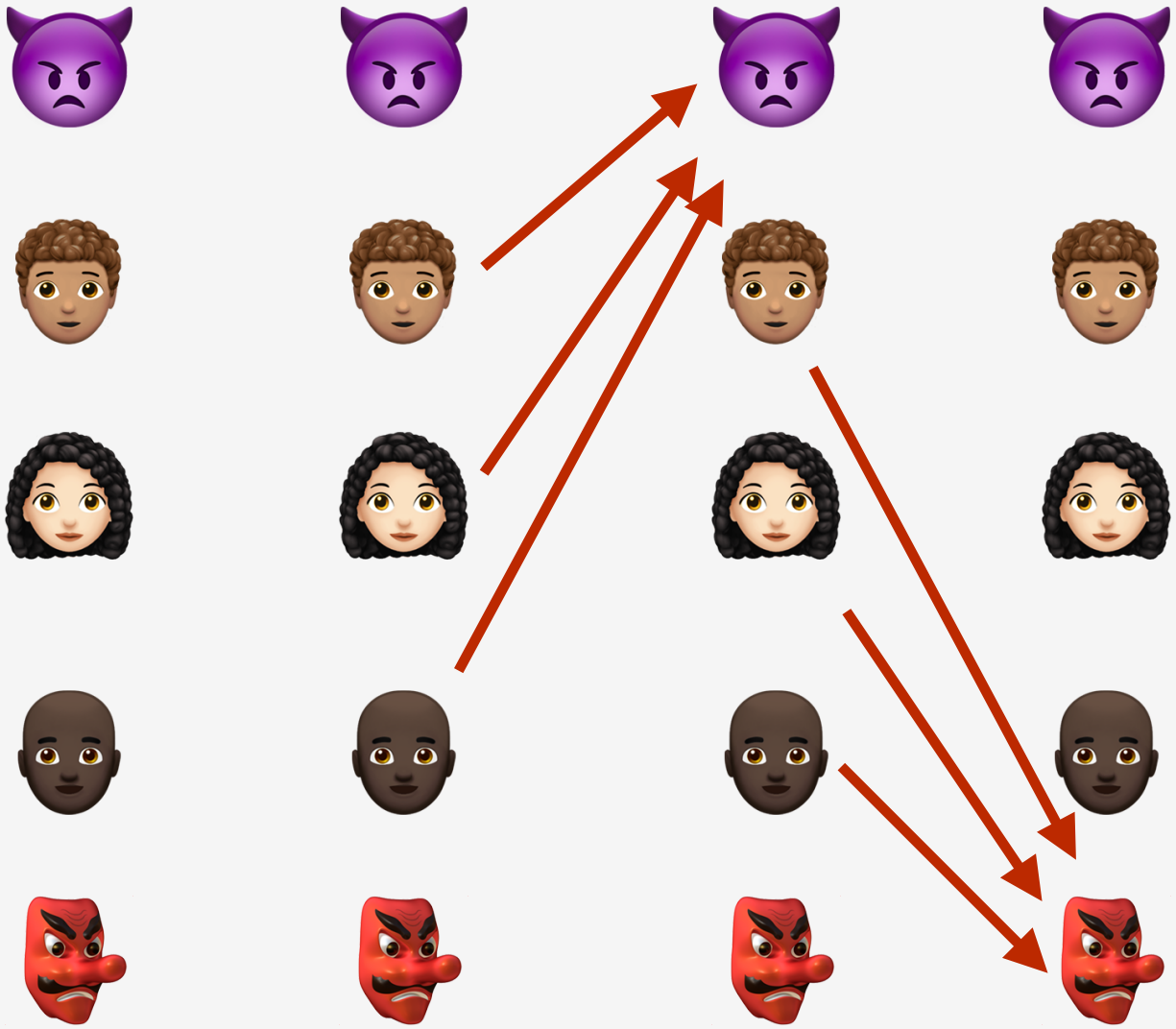


Constructing a Polariser in $f + 2$ Rounds, Example 2



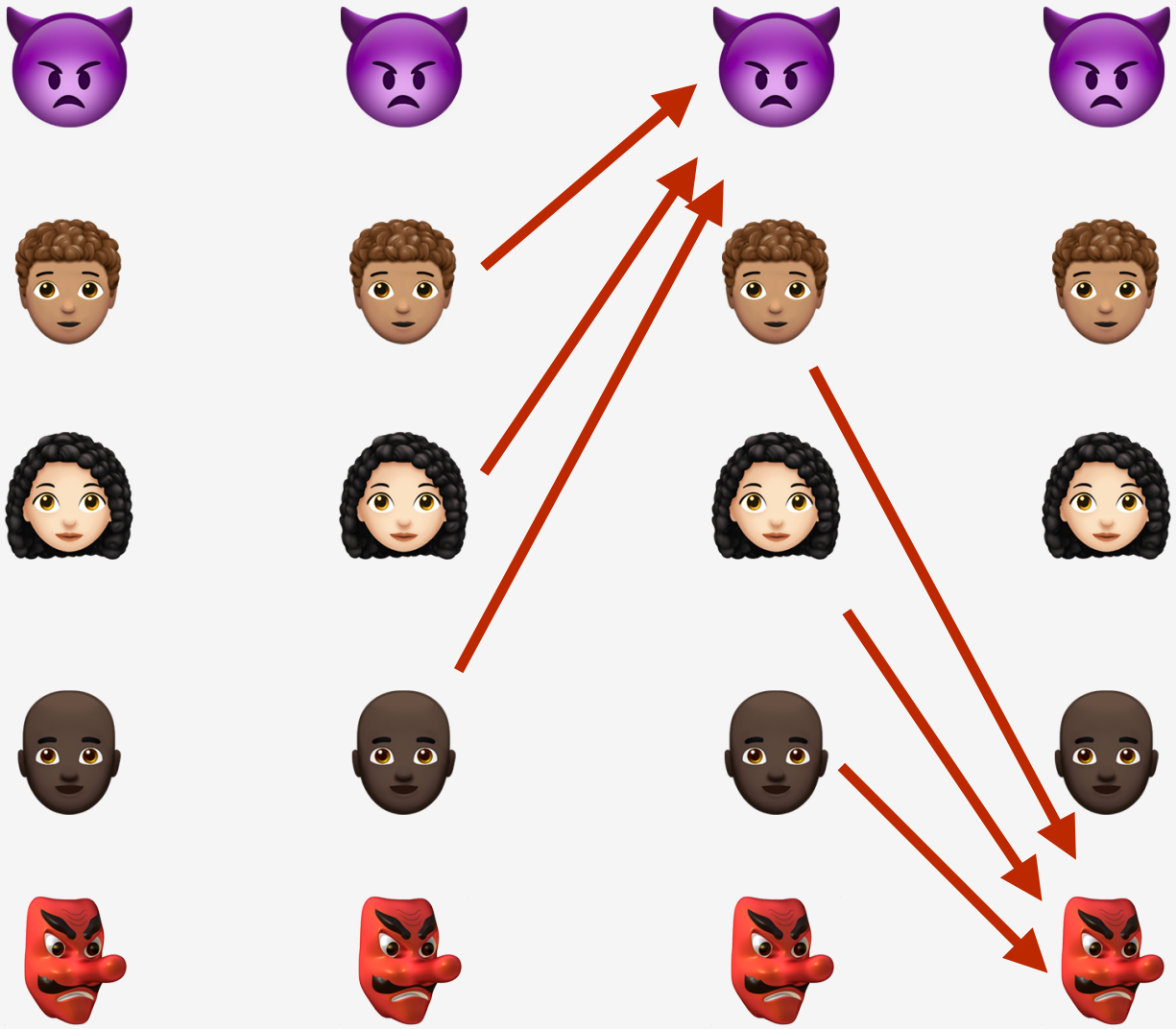
- All honest parties Accuse P_1 in Round 2

Constructing a Polariser in $f + 2$ Rounds, Example 2



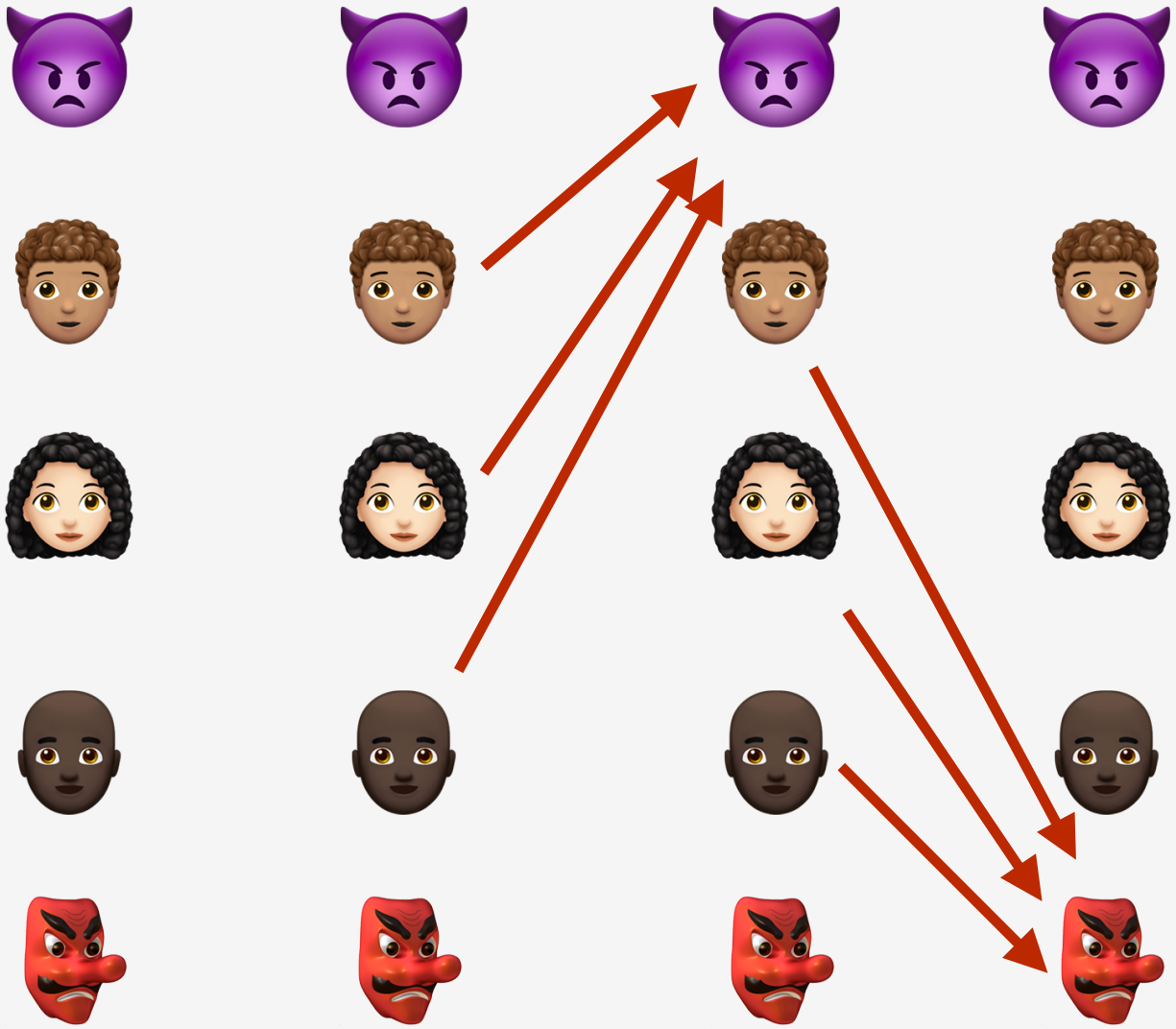
- All honest parties Accuse P_1 in Round 2

Constructing a Polariser in $f + 2$ Rounds, Example 2



- All honest parties Accuse P_1 in Round 2
- All honest parties Accuse P_5 in Round 3

Constructing a Polariser in $f + 2$ Rounds, Example 2



- All honest parties Accuse P_1 in Round 2
- All honest parties Accuse P_5 in Round 3
- P_1 and P_5 are added to Corrupt in Round 4

Tool: Justified Outputs and Inputs

Tool: Justified Outputs and Inputs

- Output m is **justified** if it comes with a forwardable proof π

Tool: Justified Outputs and Inputs

- Output m is **justified** if it comes with a forwardable proof π
- Polarisers justify empty $\perp$ outputs

Tool: Justified Outputs and Inputs

- Output m is **justified** if it comes with a forwardable proof π
- Polarisers justify empty $\perp$ outputs
- **Idea:** Recursively combine justified outputs from subprotocols to justify **next input**

Tool: Justified Outputs and Inputs

- Output m is **justified** if it comes with a forwardable proof π
- Polarisers justify empty $\perp$ outputs
- **Idea:** Recursively combine justified outputs from subprotocols to justify **next input**
- **Caveat:** May lead to exponential blowup!

Building Block: Graded Broadcast with Justifiable Output

m

Building Block: Graded Broadcast with Justifiable Output

- Graded Validity: all receivers output $(m,2,\pi)$ if sender is honest

Building Block: Graded Broadcast with Justifiable Output

- Graded Validity: all receivers output $(m,2,\pi)$ if sender is honest

m

$(m,2,\pi)$

$(m,2,\pi)$

$(m,2,\pi)$

Building Block: Graded Broadcast with Justifiable Output

- Graded Validity: all receivers output $(m,2,\pi)$ if sender is honest
- Graded Consistency: receivers with $g > 0$ output the same message

m

$(m,2,\pi)$

$(m,2,\pi)$

$(m,2,\pi)$

Building Block: Graded Broadcast with Justifiable Output

- Graded Validity: all receivers output $(m,2,\pi)$ if sender is honest
- Graded Consistency: receivers with $g > 0$ output the same message

$(m,1,\pi)$

$(\perp,0,\pi)$

$(m,1,\pi)$

Building Block: Graded Broadcast with Justifiable Output

- Graded Validity: all receivers output $(m,2,\pi)$ if sender is honest
- Graded Consistency: receivers with $g > 0$ output the same message
- Termination: All parties terminate in $O(f)$ rounds

$(m,1,\pi)$

$(\perp,0,\pi)$

$(m,1,\pi)$

Phase King in Dishonest Majority

Phase King in Dishonest Majority

- Run GBC in **phases** with a rotating **king**

Phase King in Dishonest Majority

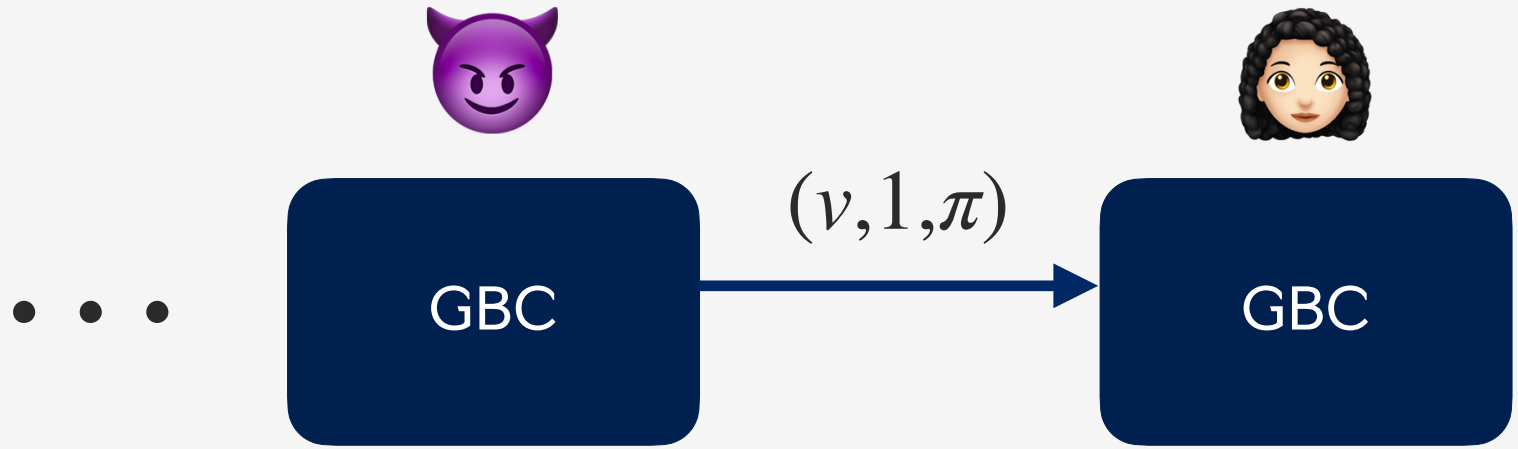
- Run GBC in **phases** with a rotating **king**

Phase King in Dishonest Majority

- Run GBC in **phases** with a rotating **king**
- Current King sends justified output from previous phase via GBC

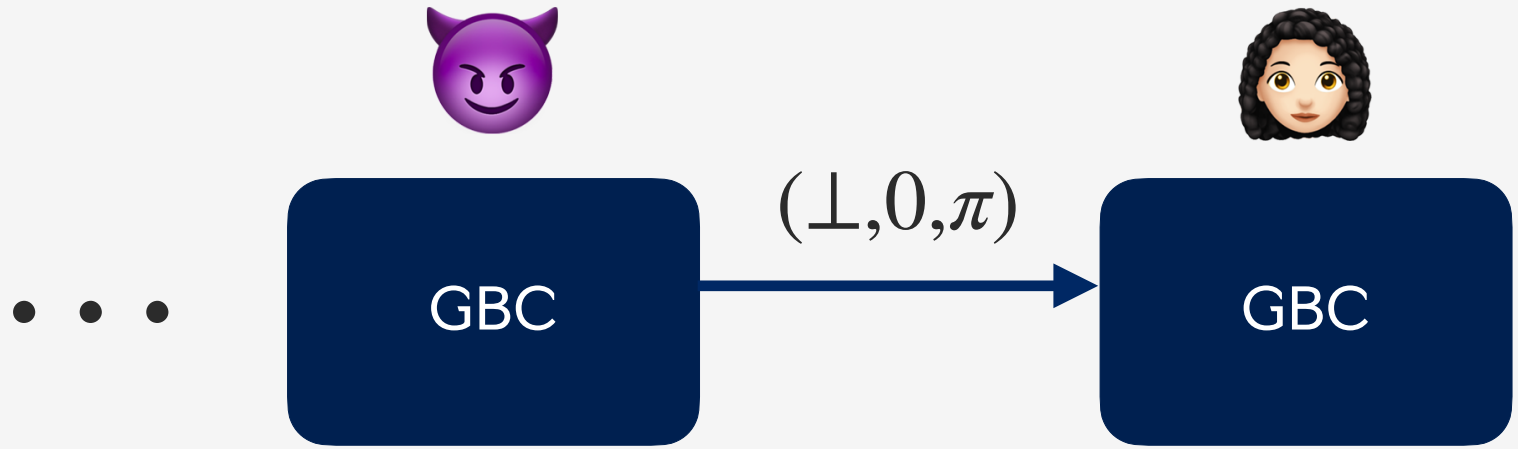
Phase King in Dishonest Majority

- Run GBC in **phases** with a rotating **king**
- Current King sends justified output from previous phase via GBC



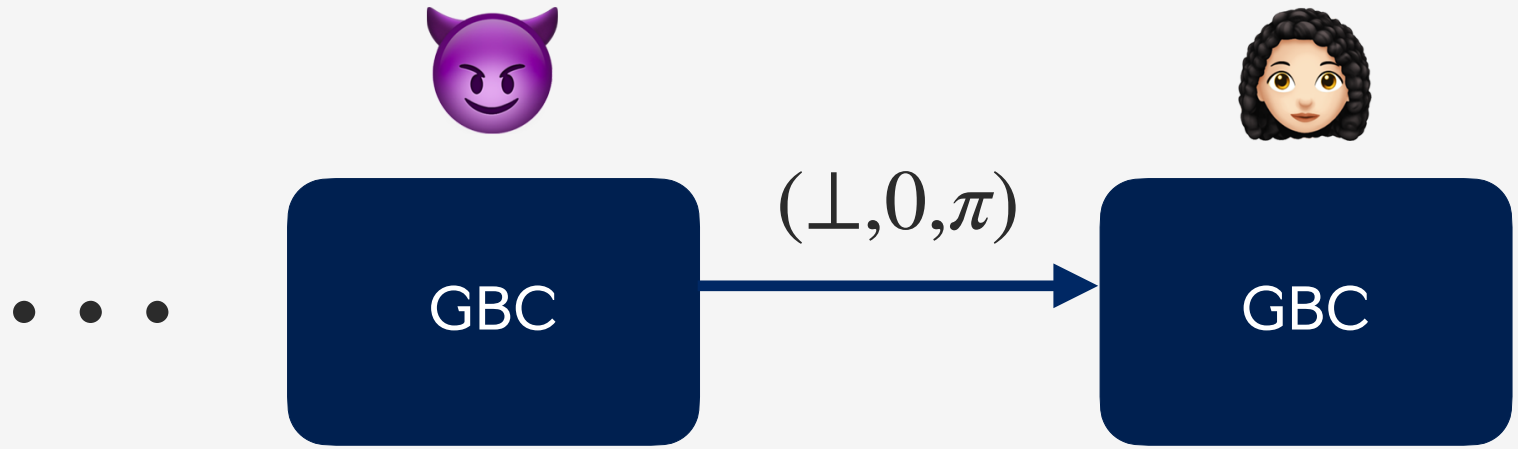
Phase King in Dishonest Majority

- Run GBC in **phases** with a rotating **king**
- Current King sends justified output from previous phase via GBC



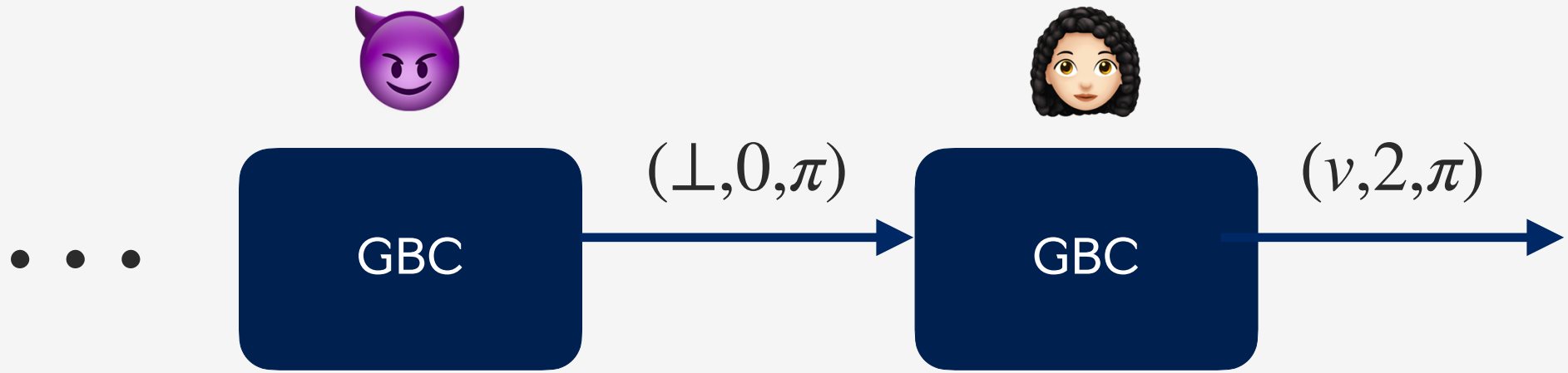
Phase King in Dishonest Majority

- Run GBC in **phases** with a rotating **king**
- Current King sends justified output from previous phase via GBC
- First honest king produces **forwardable** proof of termination



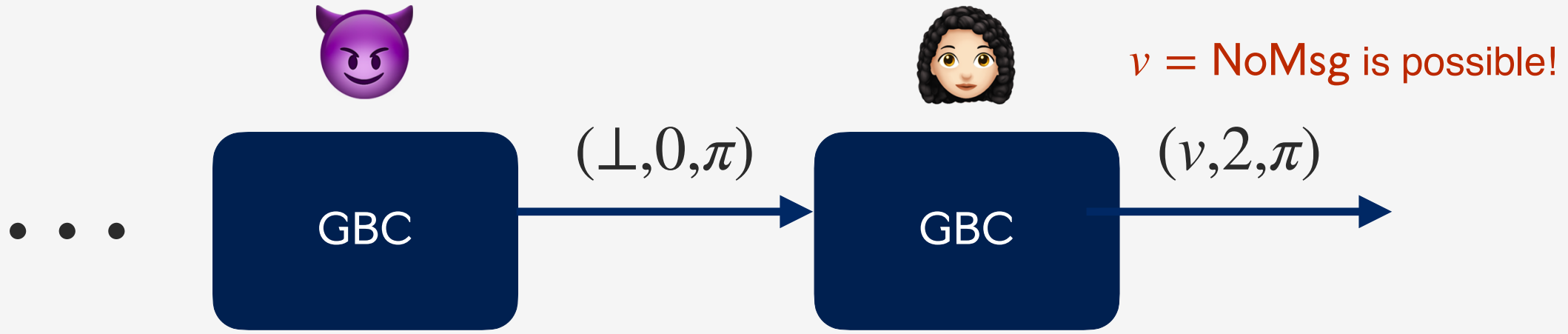
Phase King in Dishonest Majority

- Run GBC in **phases** with a rotating **king**
- Current King sends justified output from previous phase via GBC
- First honest king produces **forwardable** proof of termination

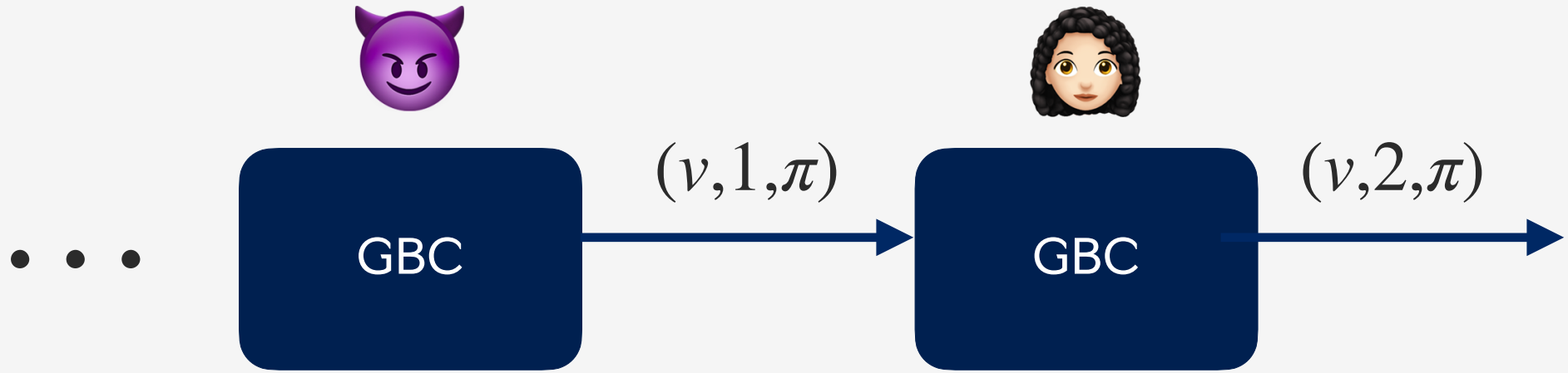


Phase King in Dishonest Majority

- Run GBC in **phases** with a rotating **king**
- Current King sends justified output from previous phase via GBC
- First honest king produces **forwardable** proof of termination

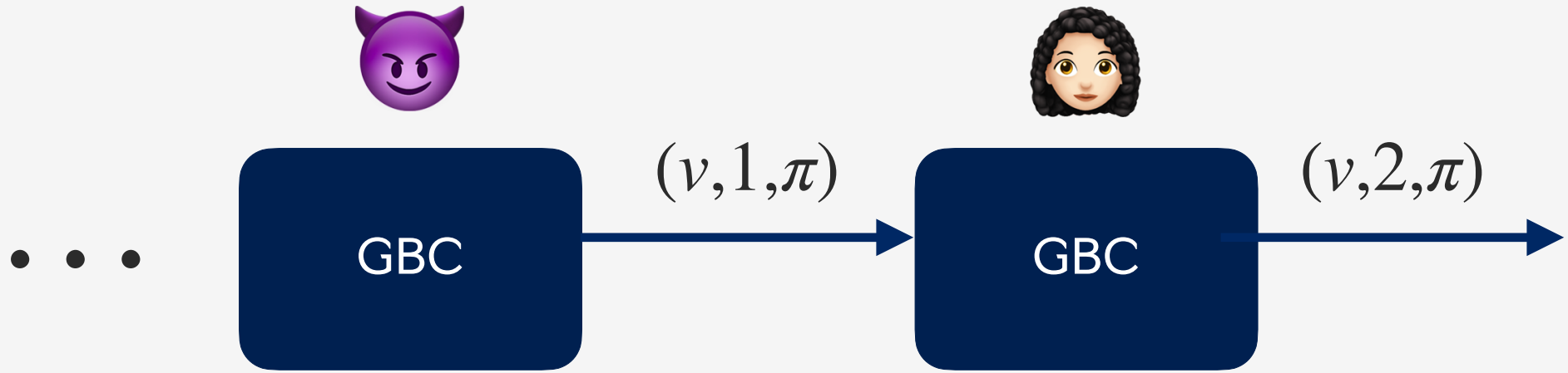


Phase King in Dishonest Majority



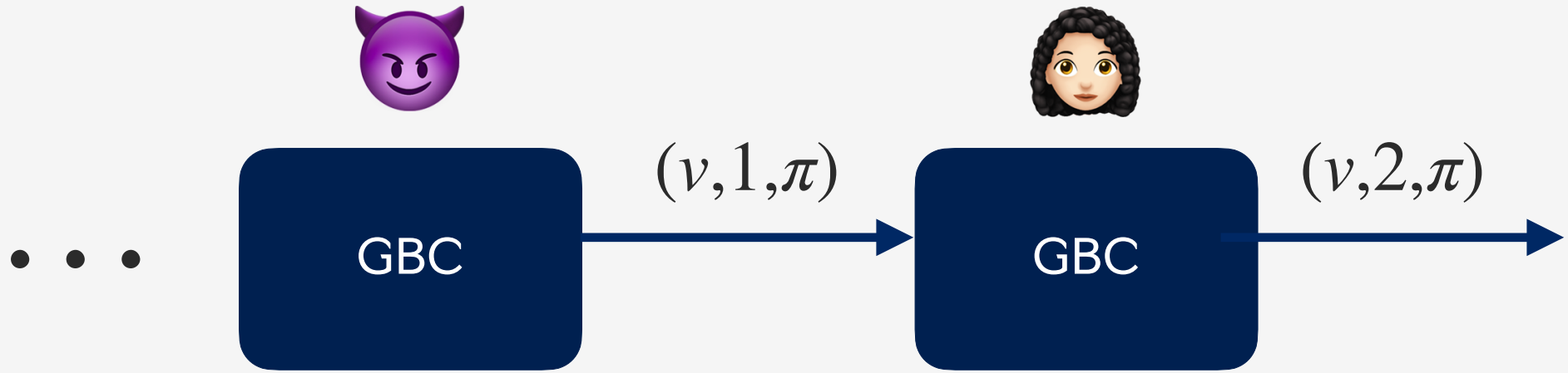
Phase King in Dishonest Majority

- Inputs/Outputs are justified Recursively



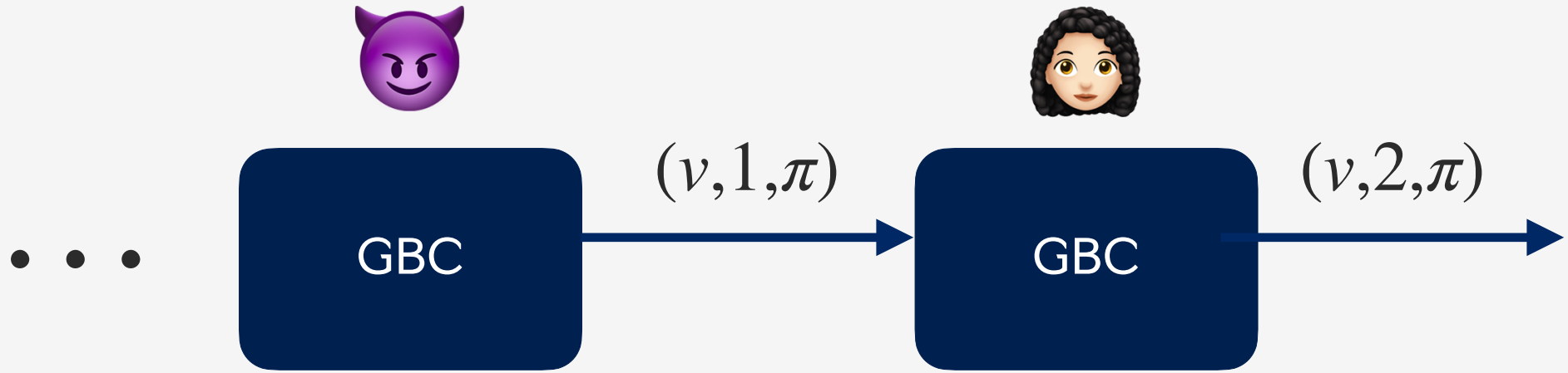
Phase King in Dishonest Majority

- Inputs/Outputs are justified Recursively
- **Observation:** Only one value can **ever** be justified with grade $g > 0$



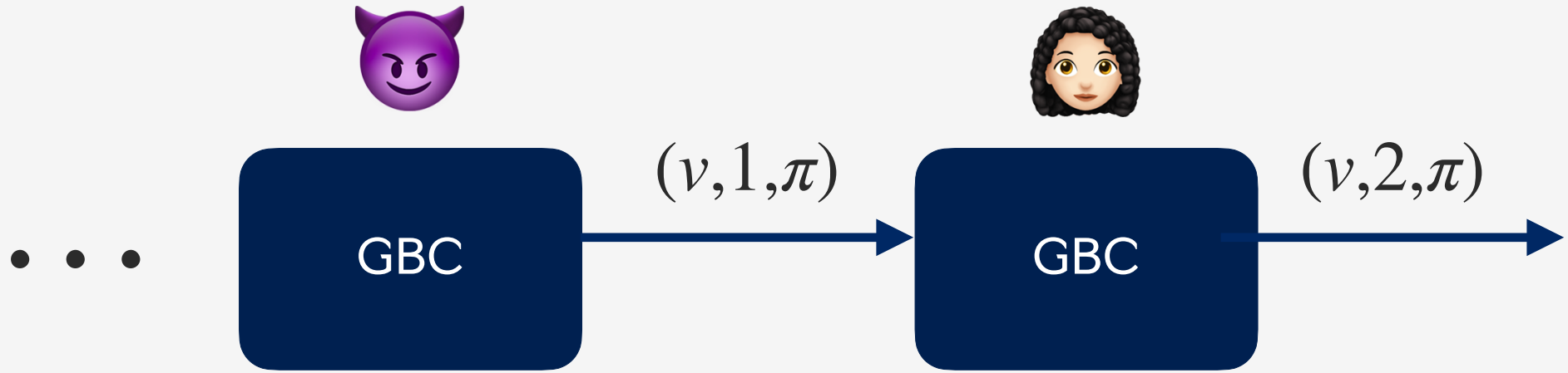
Phase King in Dishonest Majority

- Inputs/Outputs are justified Recursively
- **Observation:** Only one value can **ever** be justified with grade $g > 0$
  - $\Longrightarrow$ Complexity is kept polynomial (send same messages only once)



Phase King in Dishonest Majority

- Inputs/Outputs are justified Recursively
- **Observation:** Only one value can **ever** be justified with grade $g > 0$
  - $\Longrightarrow$ Complexity is kept polynomial (send same messages only once)
  - $\Longrightarrow$ Conflicting proofs of termination do not exist



Remaining Issue

Remaining Issue

- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$

Remaining Issue

- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!

Remaining Issue

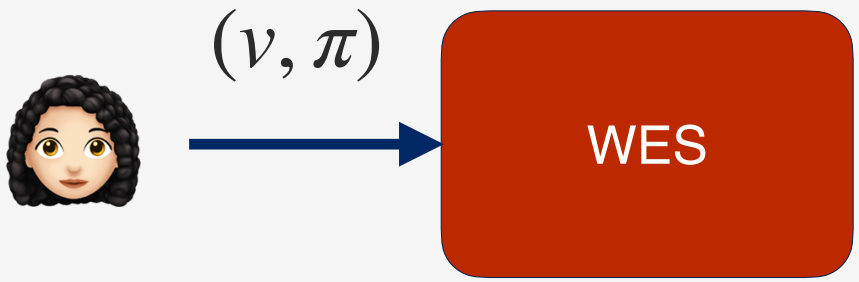
- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!
- Our final building block is **Weak Early Stopping (WES)** broadcast:

Remaining Issue

- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!
- Our final building block is **Weak Early Stopping (WES)** broadcast:
  - Terminates in $O(f)$ rounds for **honest sender**

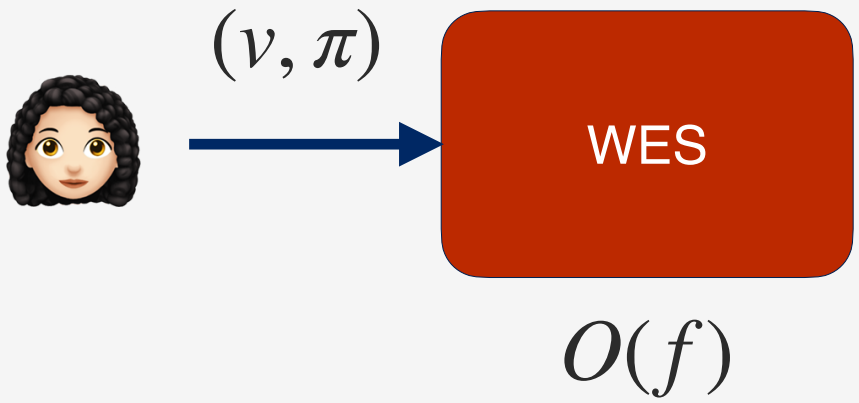
Remaining Issue

- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!
- Our final building block is **Weak Early Stopping (WES)** broadcast:
  - Terminates in $O(f)$ rounds for **honest sender**



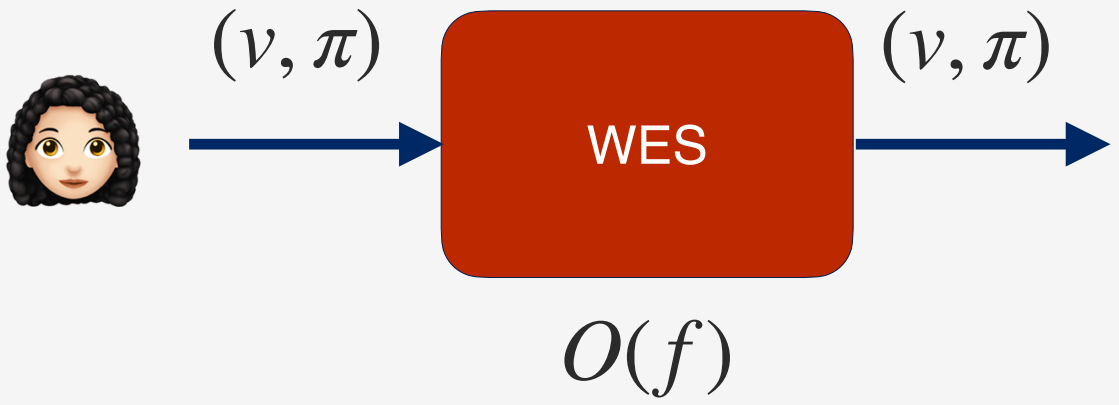
Remaining Issue

- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!
- Our final building block is **Weak Early Stopping (WES)** broadcast:
  - Terminates in $O(f)$ rounds for **honest sender**



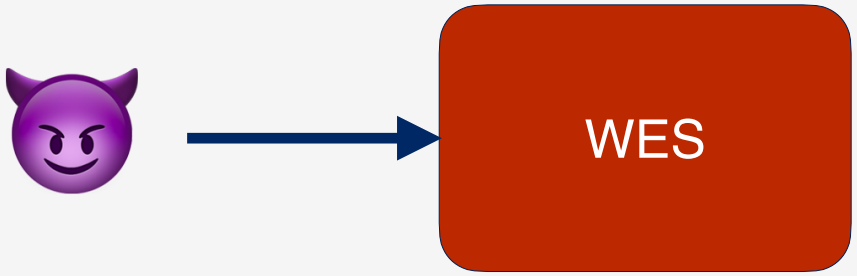
Remaining Issue

- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!
- Our final building block is **Weak Early Stopping (WES)** broadcast:
  - Terminates in $O(f)$ rounds for **honest sender**



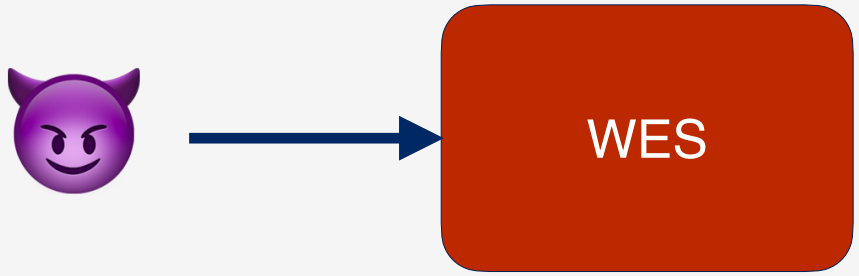
Remaining Issue

- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!
- Our final building block is **Weak Early Stopping (WES)** broadcast:
  - Terminates in $O(f)$ rounds for **honest sender**



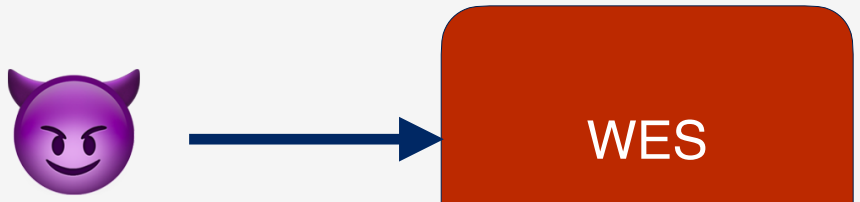
Remaining Issue

- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!
- Our final building block is **Weak Early Stopping (WES)** broadcast:
  - Terminates in $O(f)$ rounds for **honest sender**
  - Terminates in $O(t)$ for dishonest sender



Remaining Issue

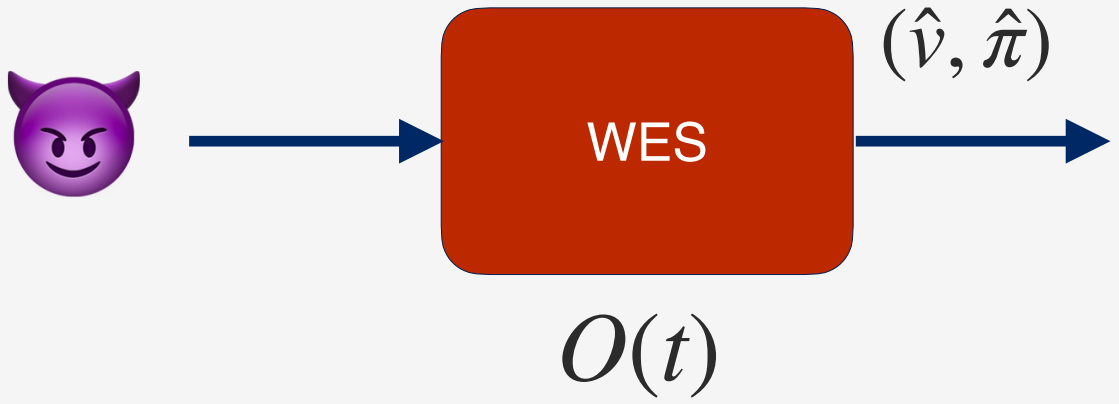
- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!
- Our final building block is **Weak Early Stopping (WES)** broadcast:
  - Terminates in $O(f)$ rounds for **honest sender**
  - Terminates in $O(t)$ for dishonest sender



$O(t)$

Remaining Issue

- Construction **always** runs $f + 1$ phase king iterations of length $O(f)$
- Might lead to $O(f^2) > O(t)$ rounds!
- Our final building block is **Weak Early Stopping (WES)** broadcast:
  - Terminates in $O(f)$ rounds for **honest sender**
  - Terminates in $O(t)$ for dishonest sender



Final Protocol

Final Protocol

- Run DC for $O(t)$ rounds

Final Protocol

- Run DC for $O(t)$ rounds

Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output

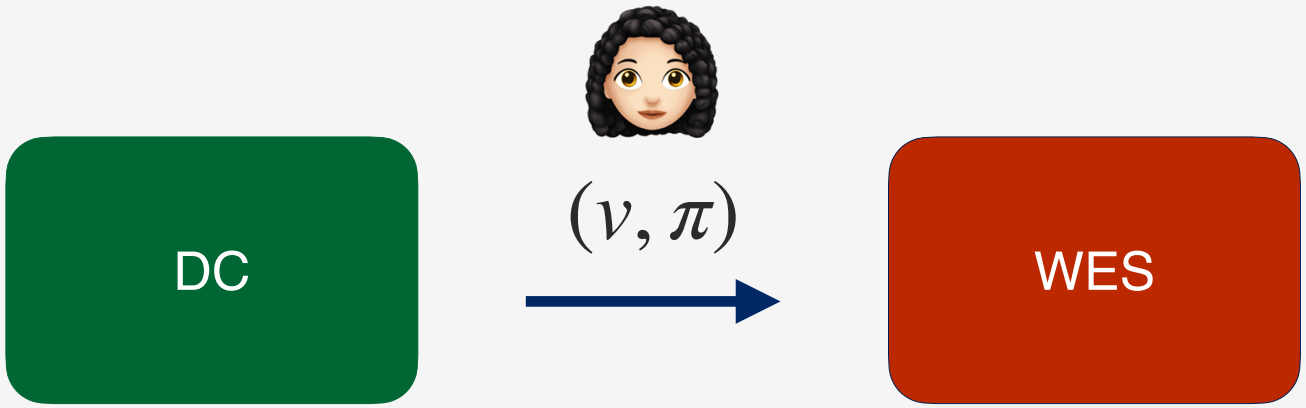
DC

Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output

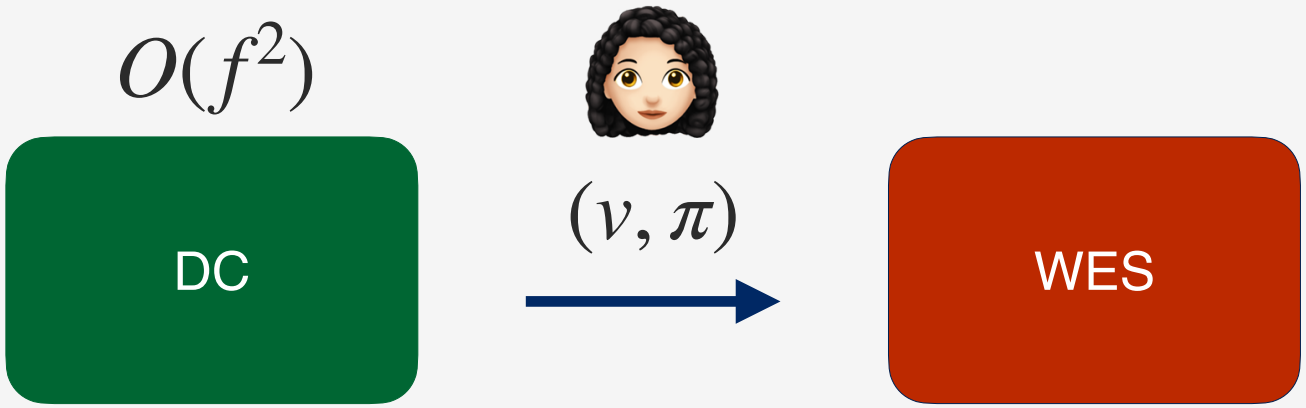
Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output



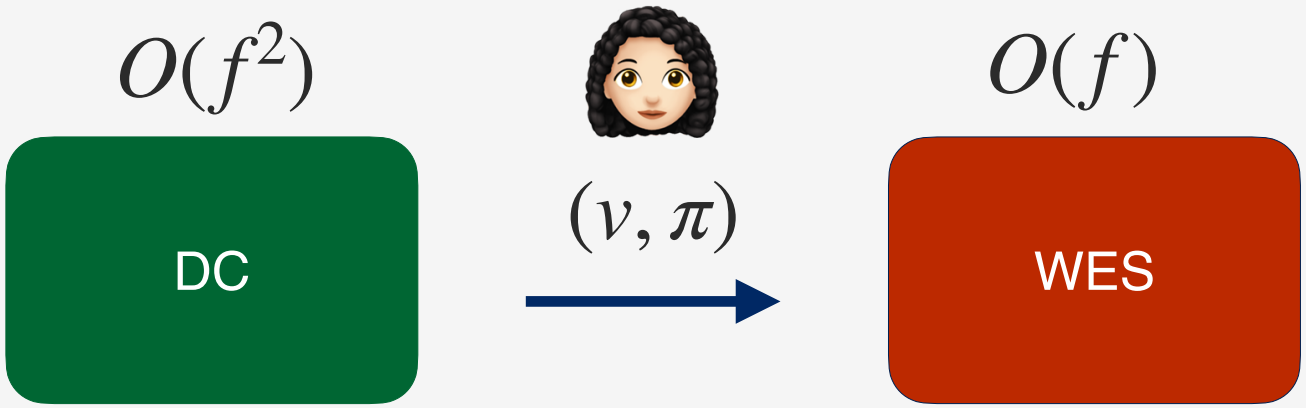
Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output



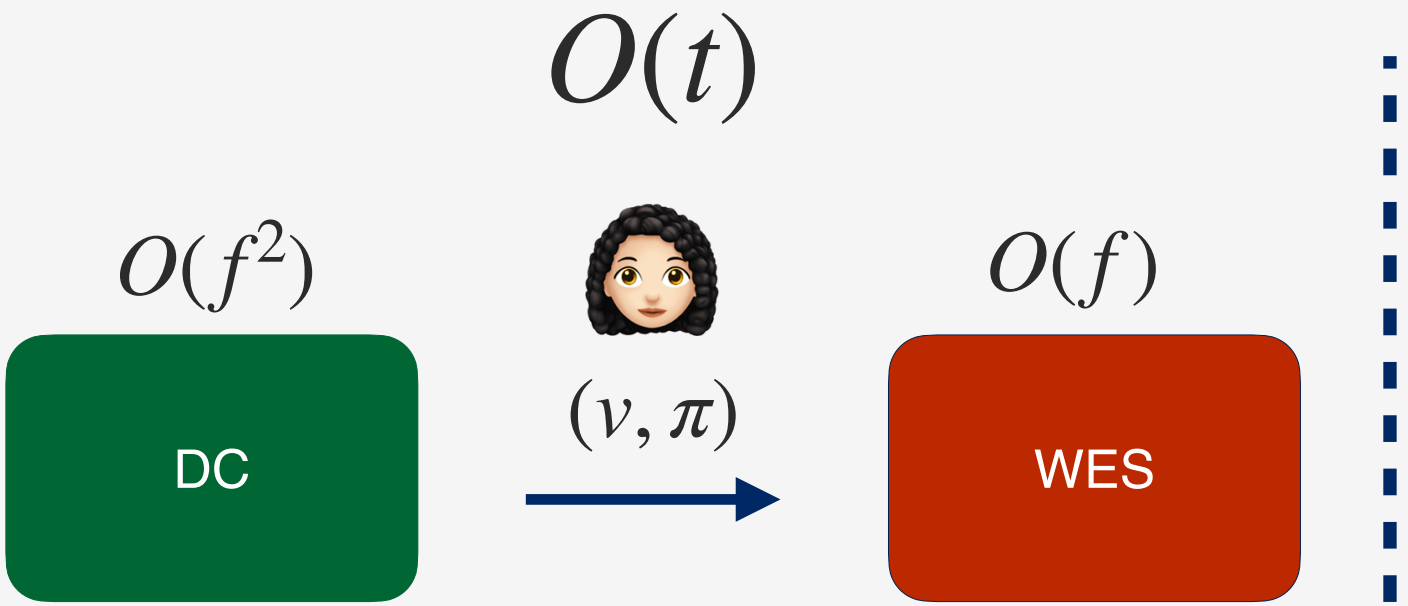
Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output



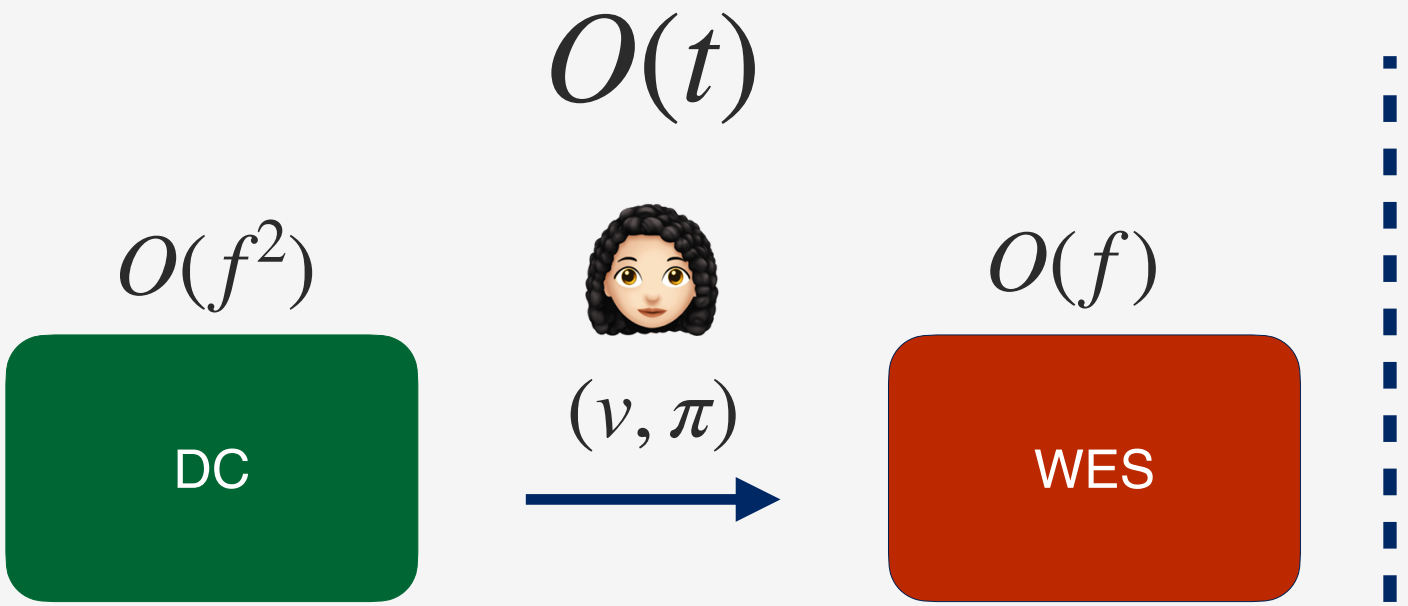
Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output



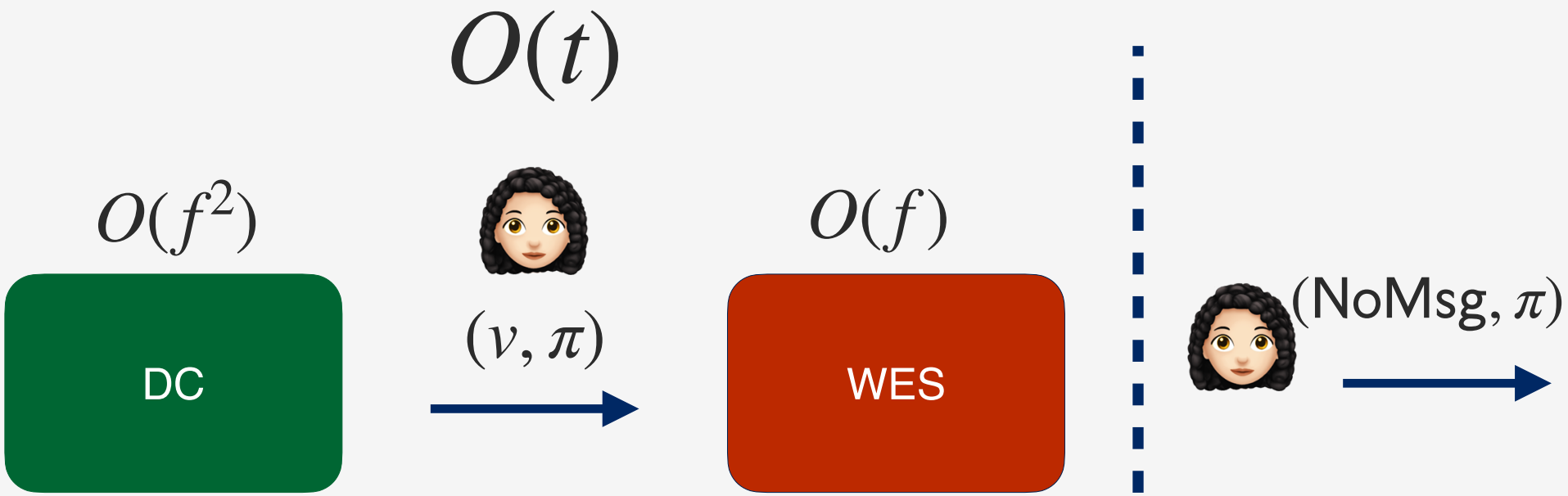
Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output
- If there is no early output, broadcast justified default output



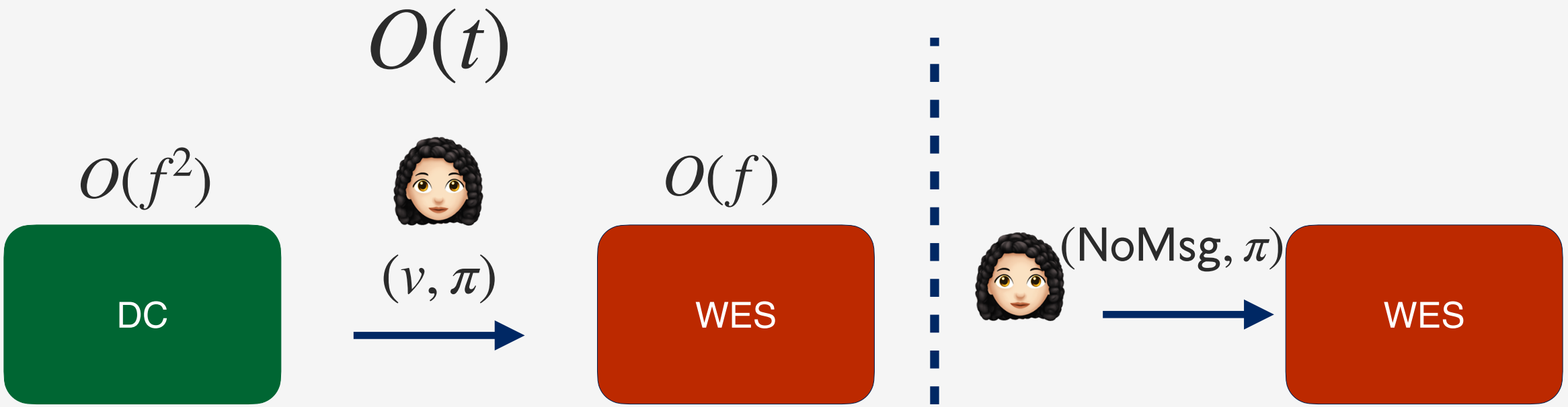
Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output
- If there is no early output, broadcast justified default output



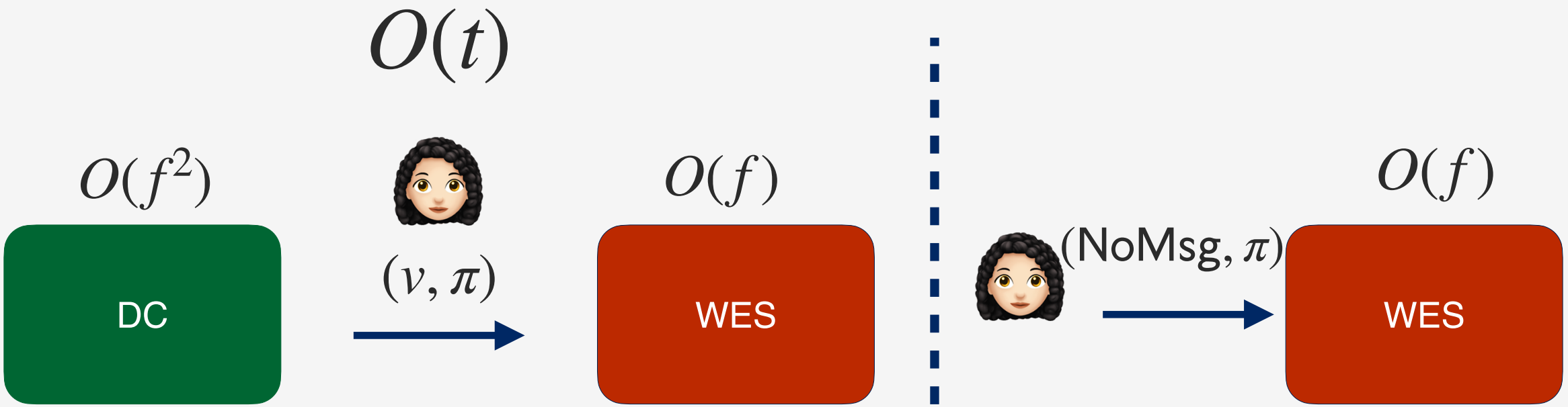
Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output
- If there is no early output, broadcast justified default output



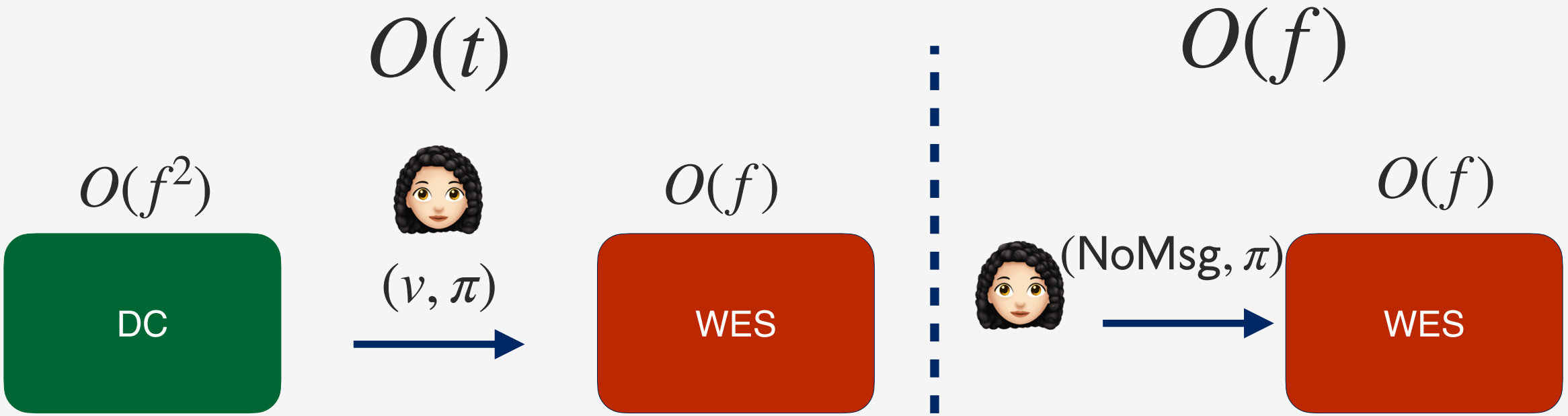
Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output
- If there is no early output, broadcast justified default output



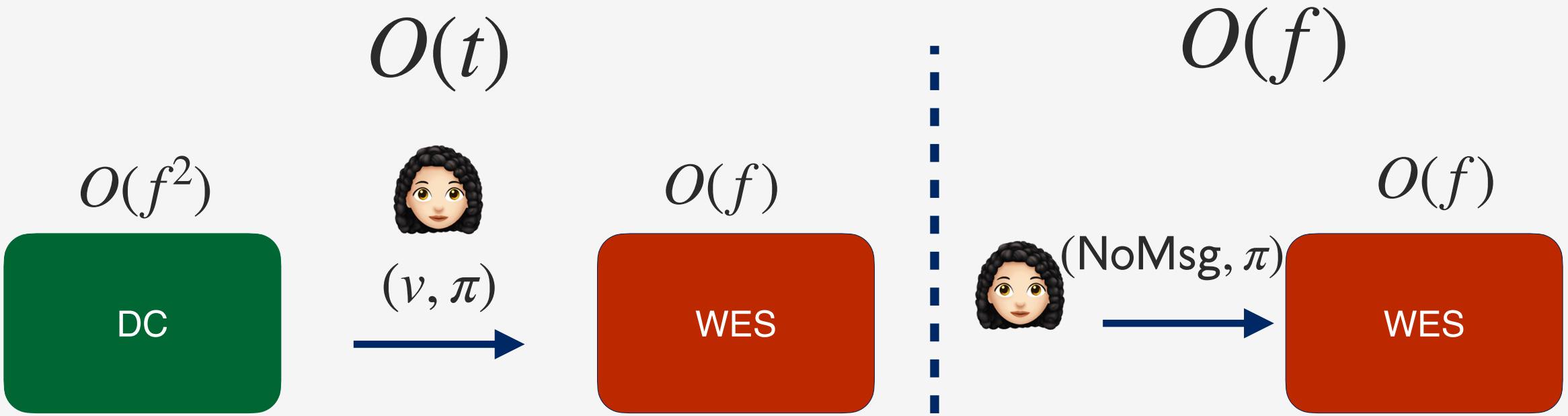
Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output
- If there is no early output, broadcast justified default output



Final Protocol

- Run DC for $O(t)$ rounds
- Use WES to broadcast an early justified output
- If there is no early output, broadcast justified default output
- Total running time is $O(\min\{f^2, t\})$ rounds



Open Questions

Open Questions

- Can we get an $o(f^2)$ round early stopping protocol?

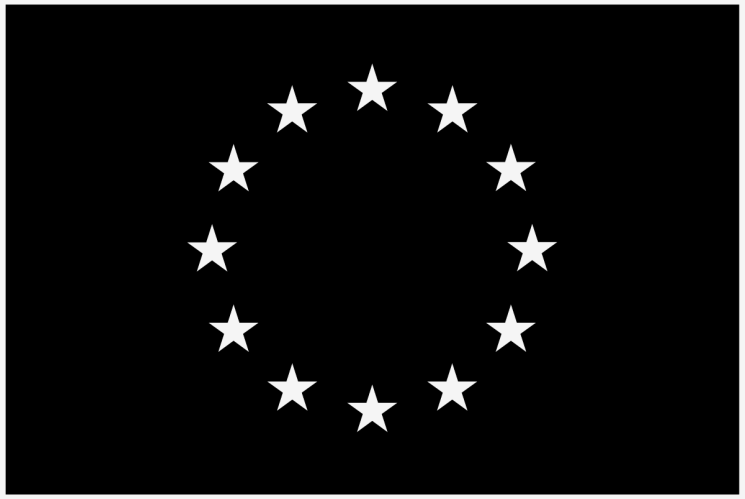
Open Questions

- Can we get an $o(f^2)$ round early stopping protocol?
- Can heavy cryptography (obfuscation, FHE, TLPs) help?

Open Questions

- Can we get an $o(f^2)$ round early stopping protocol?
- Can heavy cryptography (obfuscation, FHE, TLPs) help?
- Can we use exponential information gathering?

Funding Information



Funded by the European Union

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

Thank you!