CS4501 Cryptographic Protocols
Lecture 2: Adversaries and Simulation

https://jackdoerner.net/teaching/#2026/Spring/CS4501

Let’s Run a Protocol Together

I need two volunteers.

I am going to embarrass you.


♣♥ Matchmaking ♥♣
(how to go on a date with a cryptographer)
I promise the rest of the class won't be like this.

Does this protocol produce a correct result?

The function we're computing is AND

Yes = ♣♥
No = ♥♣

Yes = ♥♣
No = ♣♥

		Cards (Before Random Rotation)	Result
Yes	Yes	♣♥♥♥♣	
Yes	No	♣♥♥♣♥	
No	Yes	♥♣♥♥♣	
No	No	♥♣♥♣♥	

Does this protocol produce a correct result?

Yes = ♣♥
No = ♥♣

Yes = ♥♣
No = ♣♥

The function we're computing is AND

		Cards (Before Random Rotation)	Result
Yes	Yes	♣♥♥♥♣	Yes
Yes	No	♣♥♥♣♥	
No	Yes	♥♣♥♥♣	
No	No	♥♣♥♣♥	

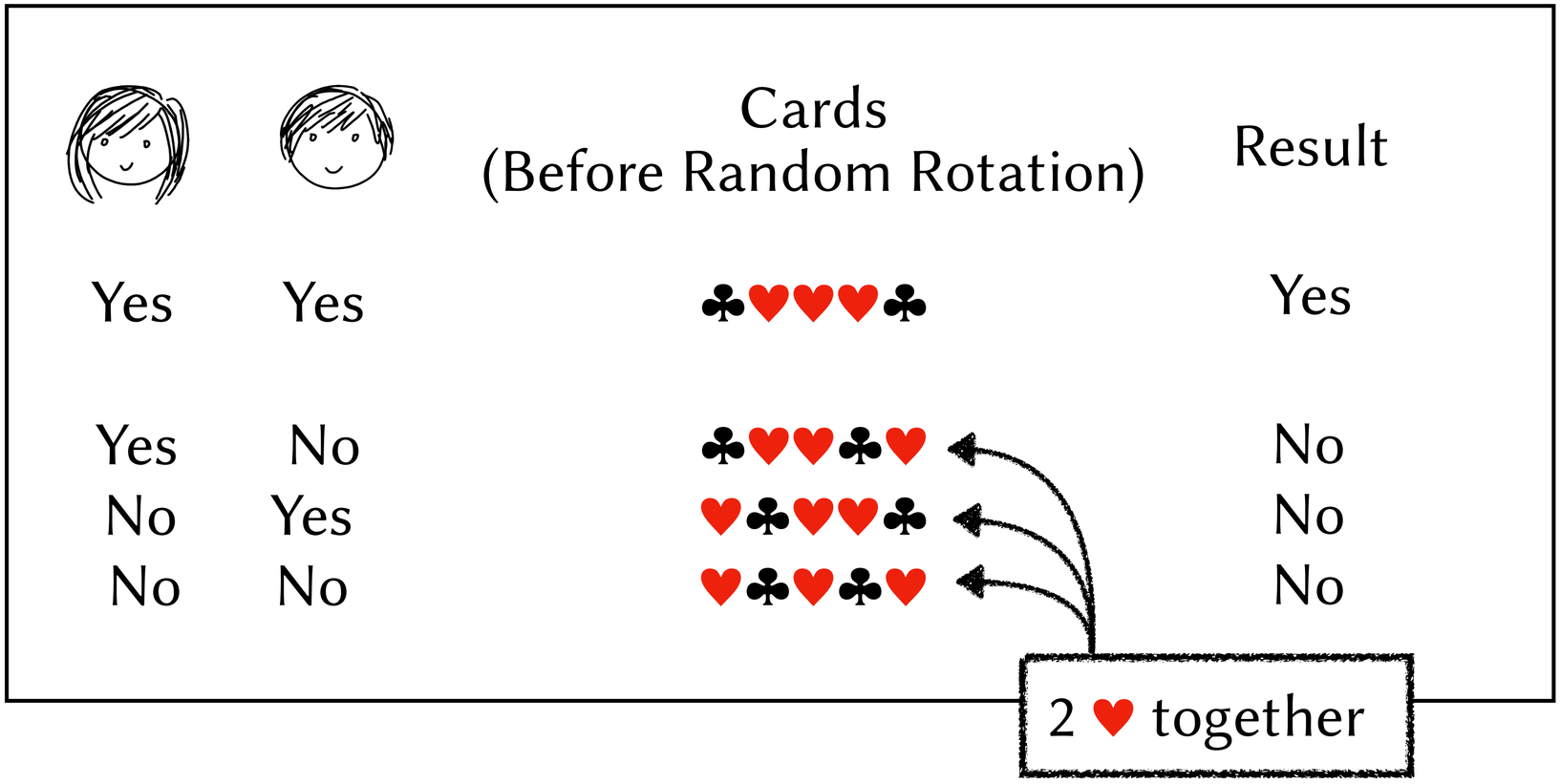
3 ♥ together

Does this protocol produce a correct result?

The function we're computing is AND

Yes = ♣♥
No = ♥♣

Yes = ♥♣
No = ♣♥



What can  and  learn from this?

Yes = ♣♥
No = ♥♣

Yes = ♥♣
No = ♣♥

		Cards (Before Random Rotation)	Result
Yes	Yes	♣♥♥♥♣	Yes
Yes	No	♣♥♥♣♥	No
No	Yes	♥♣♥♥♣	No
No	No	♥♣♥♣♥	No

Rotations of Each Other
Indistinguishable after random rotation!

What can  and  learn from this?

Yes = ♣♥
No = ♥♣

Yes = ♥♣
No = ♣♥

		Cards (Before Random Rotation)	Result
Yes	Yes	♣♥♥♥♣	Yes
Yes	No	♣♥♥♣♥	No
No	Yes	♥♣♥♥♣	No
No	No	♥♣♥♣♥	No

As promised, they learn only the result.
We proved that the protocol is *secure.*

What can  and  learn from this?

Question 1: what happens if  doesn't rotate the deck randomly?

Question 2: what happens if  reorders the cards instead of rotating.

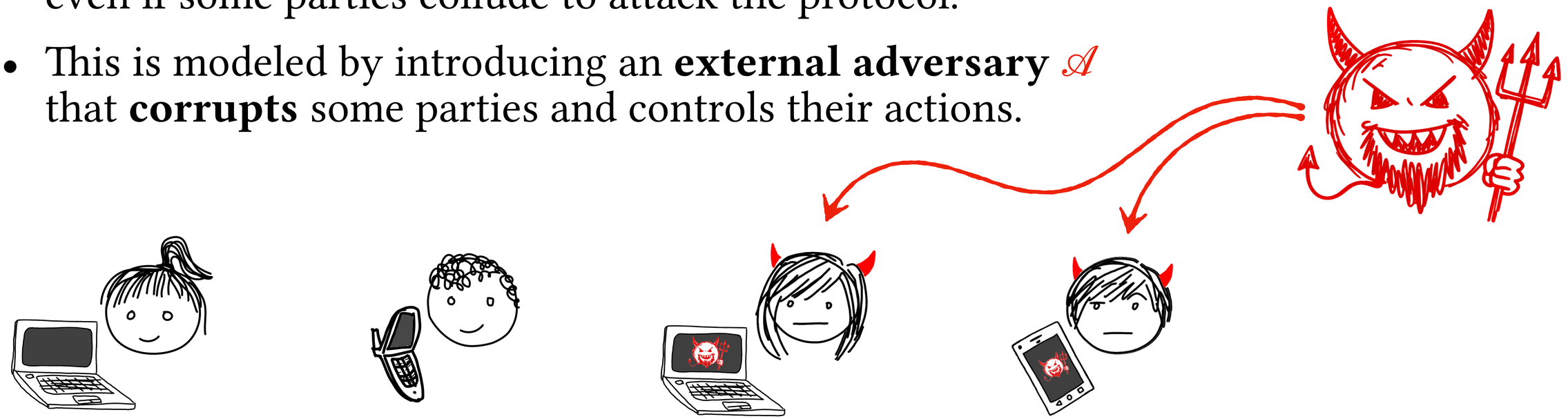
Question 3: what happens if  inputs YES regardless of her intention.

Why doesn't the last one violate the security of the protocol?

A More Nuanced View of Security

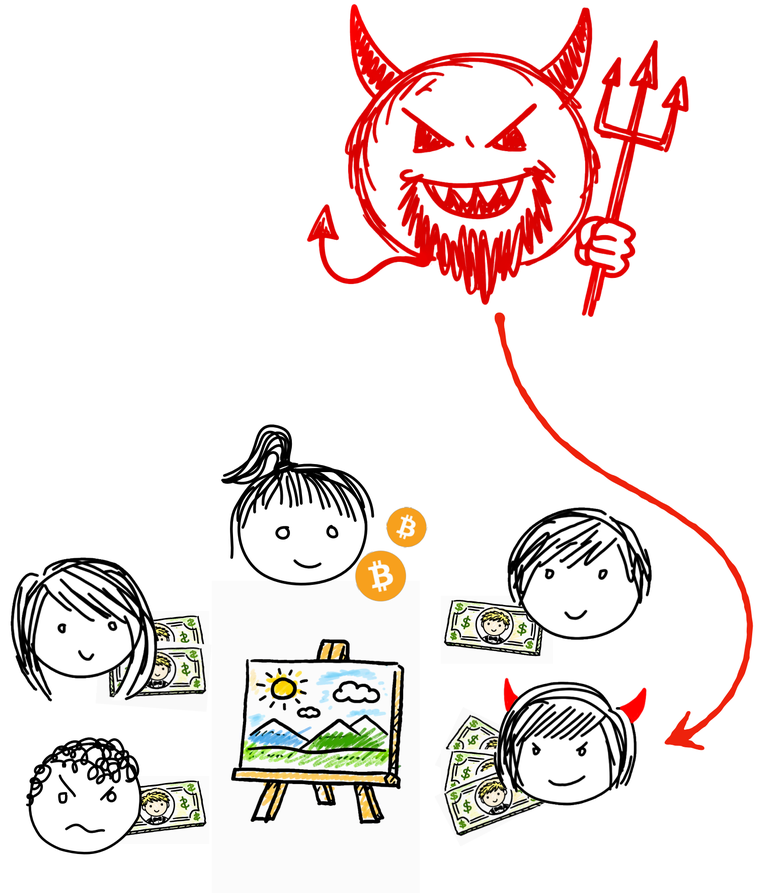
The Setting

- n parties, $P_1, \ldots, P_n$. You can just think of them as computers. Technically modeled as Interactive Turing Machines (ITMs).
- The parties wish to compute a *known* function $y = f(x_1, \ldots, x_n)$.
- Each P_i has private input x_i. For now they all want the same output y.
- The protocol performing the computation must achieve certain **security properties**, even if some parties collude to attack the protocol.
- This is modeled by introducing an **external adversary** $\mathcal{A}$ that **corrupts** some parties and controls their actions.



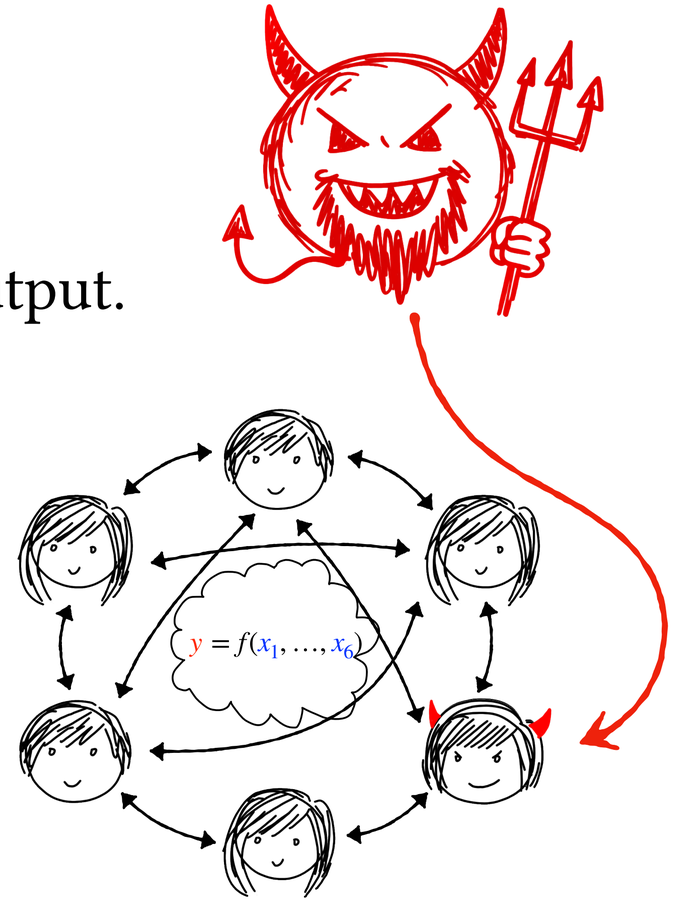
Security Properties (Auction Example)

- **Correctness:** $\mathcal{A}$ can't win using a bid that isn't the highest bid.
- **Privacy:** $\mathcal{A}$ learns the value of the highest bid, the identity of the highest bidder, and nothing else.
- **Independence of Inputs:** $\mathcal{A}$ cannot bid $1 more than the highest honest bid (except by chance)
- **Fairness:** $\mathcal{A}$ can't cause the protocol to abort if its bid isn't the highest (i.e. after learning the output).
- **Guaranteed Output Delivery:** $\mathcal{A}$ can't cause the protocol to abort at all (this is stronger than fairness and eliminates DoS attacks).



Security Properties (More Generally)

- **Correctness:** Corrupt parties can only influence the output by choosing their inputs.
- **Privacy:** Only the output is learned (nothing else).
- **Independence of Inputs:** Parties cannot choose their inputs as functions of other parties' inputs.
- **Fairness:** If one party learns the output, then all parties do.
- **Guaranteed Output Delivery:** All honest parties learn the output.



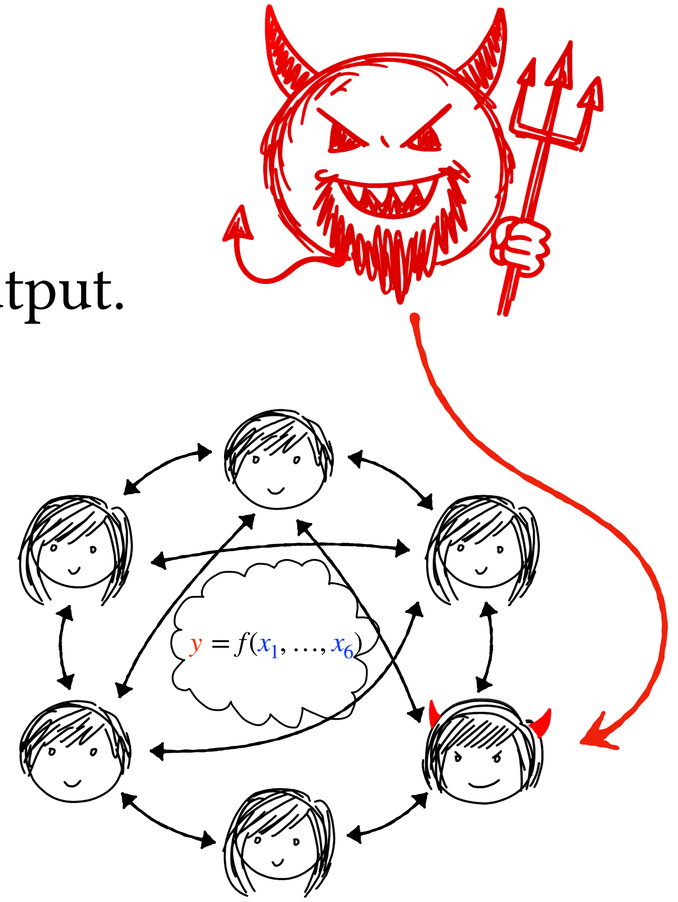
These aren't *formal* or *rigorous* definitions,
but we could make them so.

We might not want all of them all of the time.

And we could imagine many *more* security properties!

Security Properties (More Generally)

- **Correctness:** Corrupt parties can only influence the output by choosing their inputs.
- **Privacy:** Only the output is learned (nothing else).
- **Independence of Inputs:** Parties cannot choose their inputs as functions of other parties' inputs.
- **Fairness:** If one party learns the output, then all parties do.
- **Guaranteed Output Delivery:** All honest parties learn the output.



So far I have told you that *security* is related to $\mathcal{A}$ being able to emulate the protocol messages on it's own. It's intuitive how that's related to privacy, but can it possibly imply *guaranteed output* or even *correctness*?

We need to revisit our definition of security!

There's something else missing from the picture too…

Another Protocol Example

No embarrassment this time.

Example: n-Party Sum

- Let M be some positive integer fixed a priori.
- Each P_i has private input $x_i \leq M/n$.
- They wish to compute $y = \sum_{i \in [n]} x_i$.

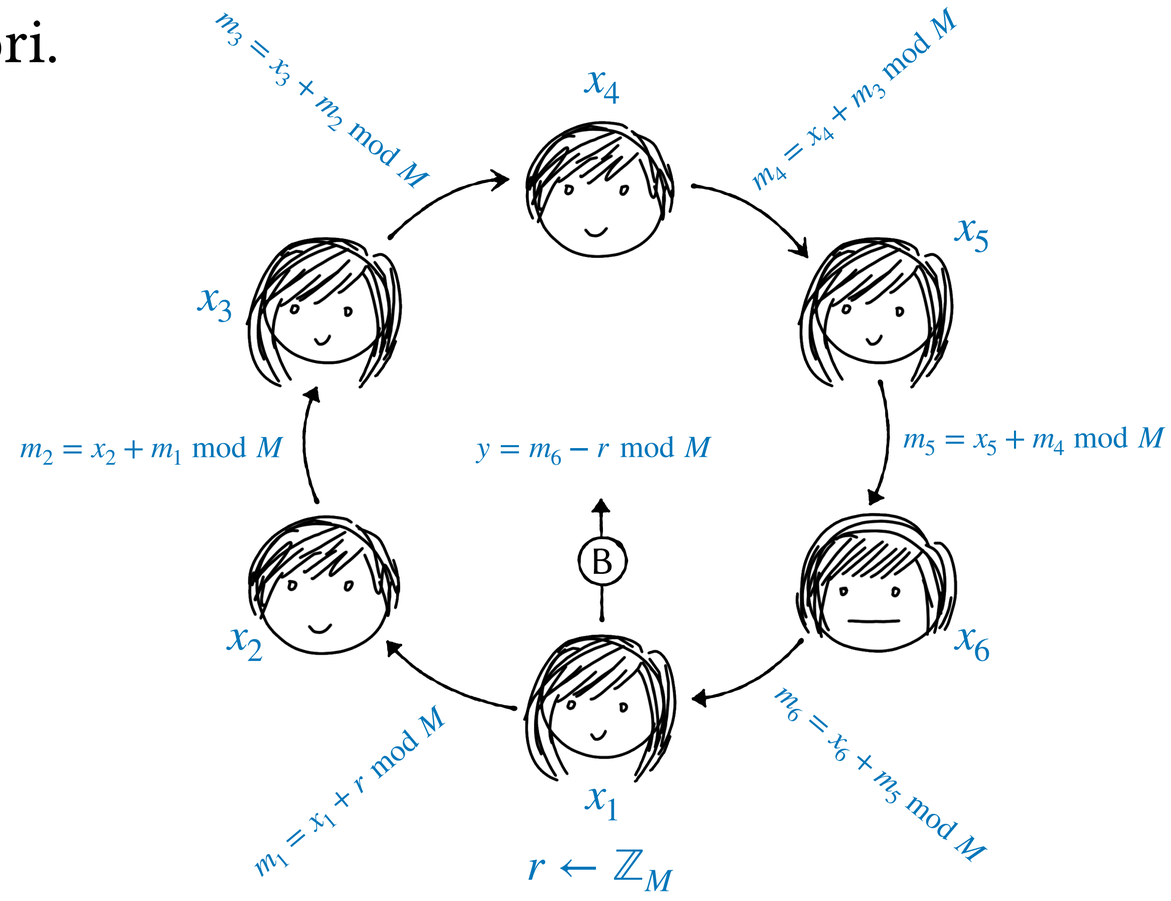
Notation

$[n] = \{1, \ldots, n\}$ $\quad$ $[n, m] = \{n, \ldots, m\}$

$\mathbb{Z}_M$ is the integers modulo M.
For our purposes, $\mathbb{Z}_M = [0, M-1]$.

$x \leftarrow D$ where D is a distribution means x is a random var distributed according to D

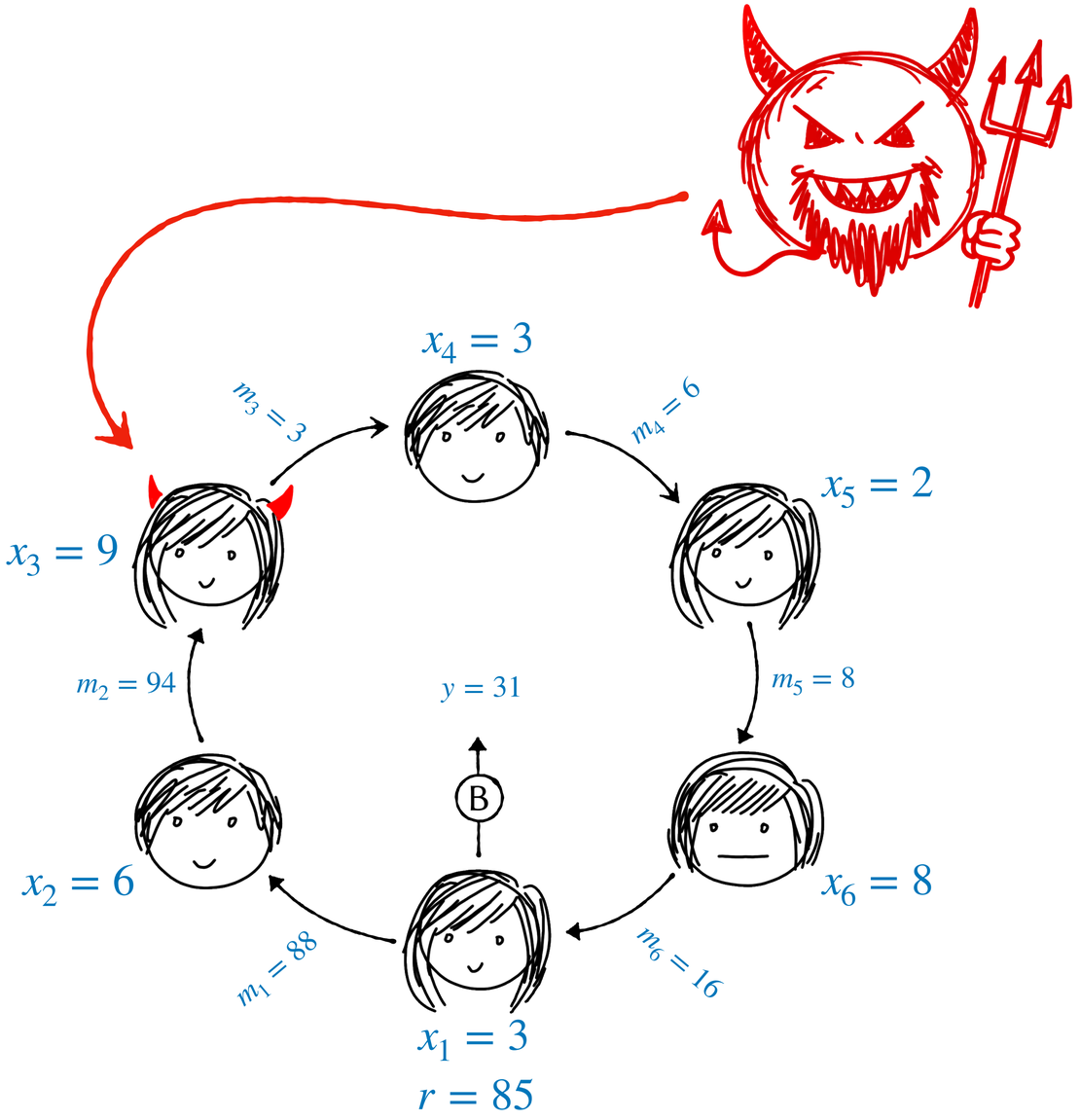
$x \leftarrow S$ where S is a set means x is sampled from the uniform distribution over S



In the future we will omit the $\bmod M$ after every operation and just say "they work modulo M"

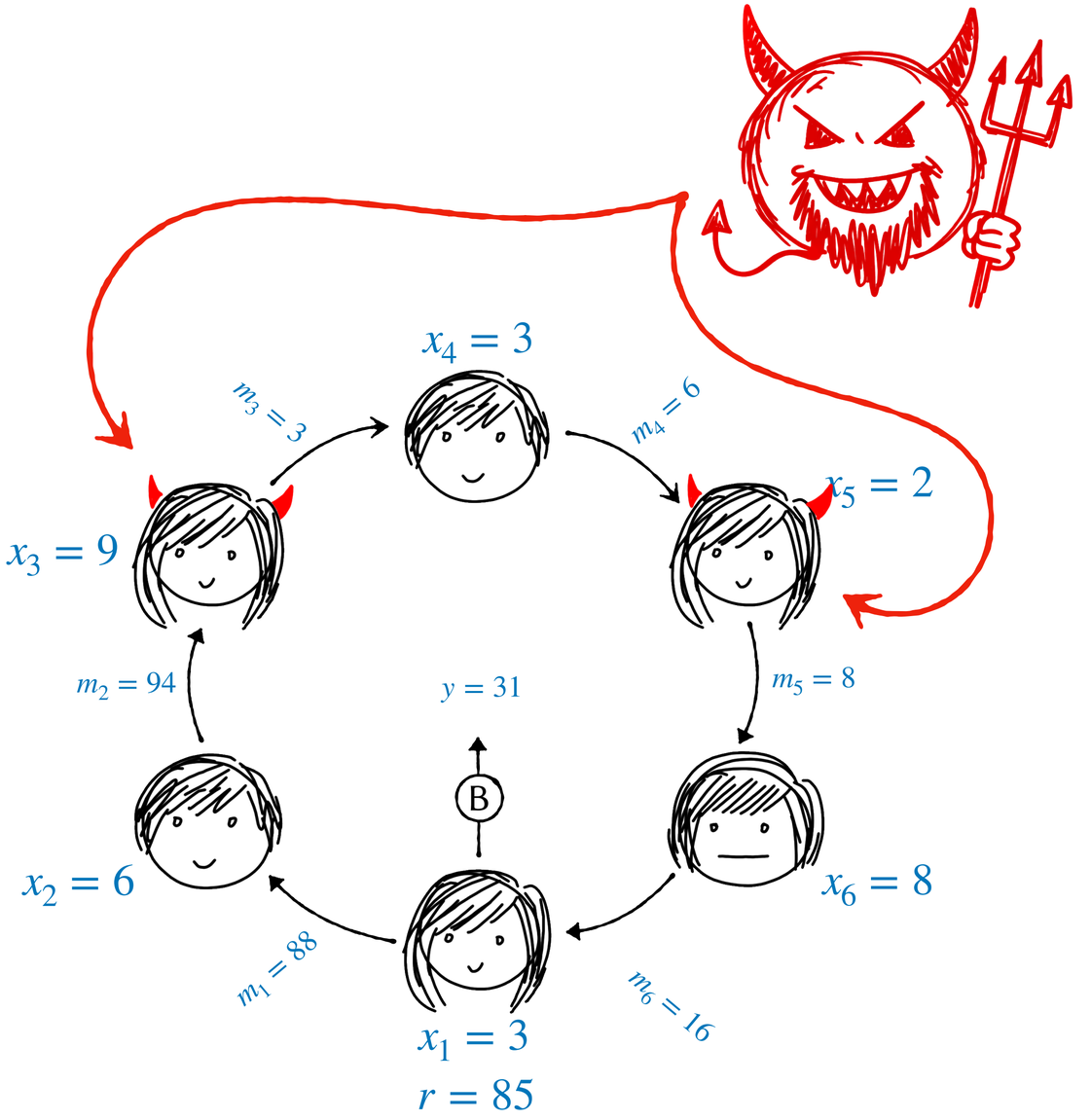
An Instance with 6 Parties

- Let $M = 100$, so $x_i \leq 16$
- Suppose one party is "honest but curious" (We call this a *semi-honest* adversary)
- $\mathscr{A}$ learns $m_2 = x_2 + x_1 + r$
- $\forall i \in \mathbb{Z}_M \;\; \Pr[m_2 = i] = 1/100$
- In other words, the distribution of m_2 is independent of x_1 and x_2.
- $\mathscr{A}$ could sample $m_2 \leftarrow \mathbb{Z}_M$ on its own.
- So the protocol is **private**.



An Instance with 6 Parties

- Let $M = 100$, so $x_i \leq 16$
- Suppose one party is "honest but curious" (We call this a *semi-honest* adversary)
- $\mathcal{A}$ learns $m_2 = x_2 + x_1 + r$
- $\forall i \in \mathbb{Z}_M \;\; \Pr[m_2 = i] = 1/100$
- In other words, the distribution of m_2 is independent of x_1 and x_2.
- $\mathcal{A}$ could sample $m_2 \leftarrow \mathbb{Z}_M$ on its own.
- So the protocol is **private**.
- But what if *two* parties are corrupted?



What can we take from this?

Security is not a property of the protocol *alone*.
It also depends upon the adversarial model.

How to Define Security

Option 1: Property Based

- Define a list of security requirements for the task to be accomplished (e.g. privacy, fairness, guaranteed output)
- Give a unique mathematical formalization for each property. Prove each property individually.
- For simple foundational primitives like *Encryption* or *Digital Signatures*, this is the kind of security definition we usually use. Usually only 1-2 properties we care about. (e.g. Signatures require the property of *Unforgeability*). Come to CS6222!
- For complex tasks, how do we know if we have covered all of our concerns? Proving many properties is also a huge chore.
- **Deeper Problem**: when we proved the properties of protocol A, we considered it on its own. Will it still have those properties when we use it to build protocol B?

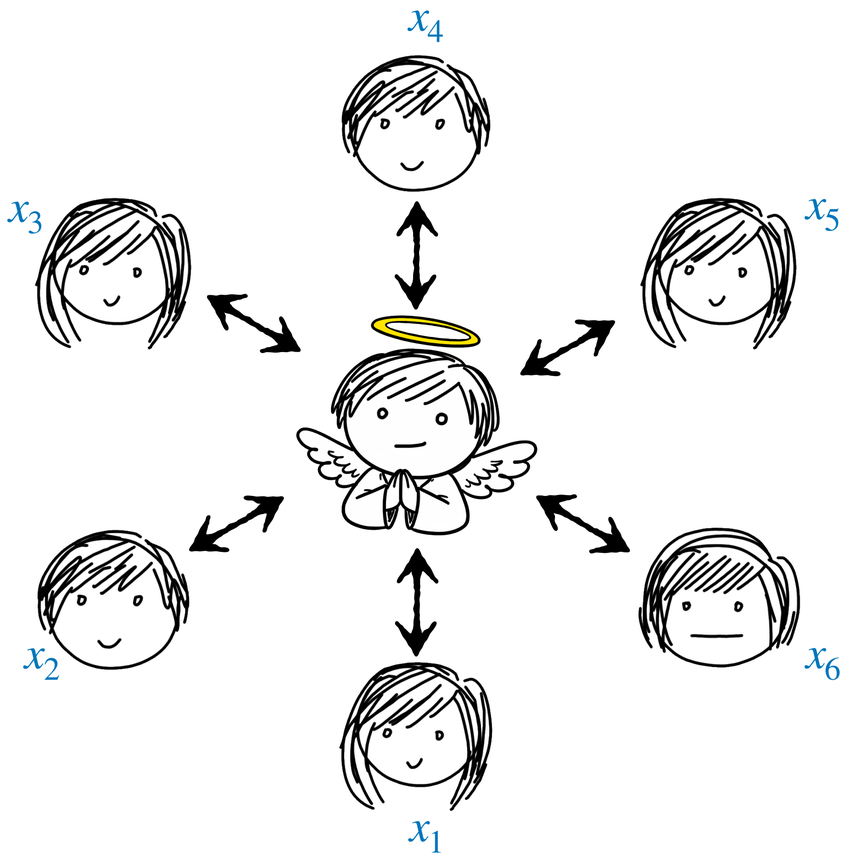
How to Define Security

Option 2: the simulation-based (a.k.a. real-ideal) paradigm

- Consider the real world in which the protocol is executed.
  Parties have inputs, communicate with each other, produce outputs.
- Consider an ideal world in which an additional trusted party exists.
  The trusted party helps the other parties carry out the computation.
  In this world security holds *by definition*, i.e. we define this world to be secure.
- We say that the real protocol is secure if whatever an adversary can achieve by attacking it can also be achieved by attacking the ideal computation involving the trusted party.
- Thus the behavior of the trusted party captures *all* properties of the protocol at once, not just the function it computes. We call this trusted party an ideal functionality. When a protocol is secure with respect to some functionality, we say that it realizes the functionality.

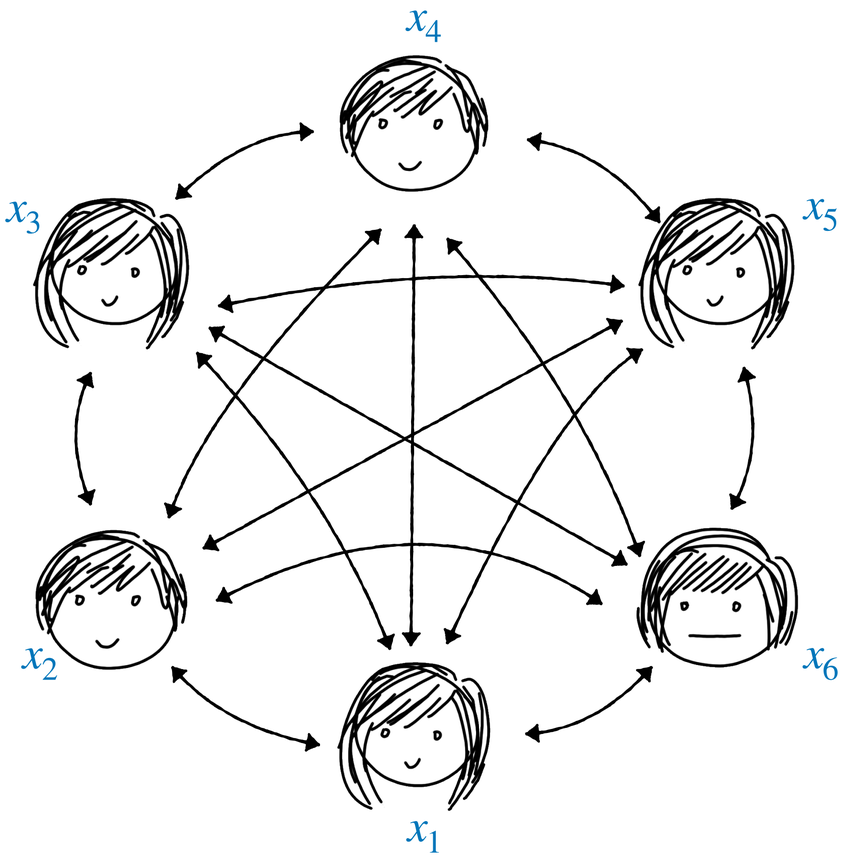
The Ideal World

1. Each P_i sends its input x_i to the ideal functionality ($\mathcal{F}$)
2. $\mathcal{F}$ computes $y = f(x_1, \ldots, x_n)$.
3. $\mathcal{F}$ sends y to every party, and they output it.

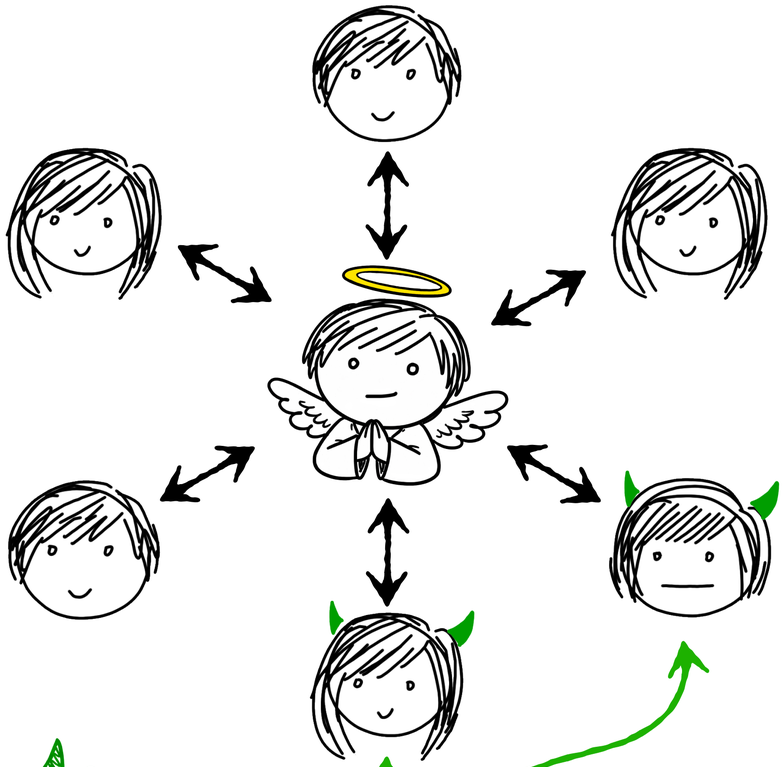


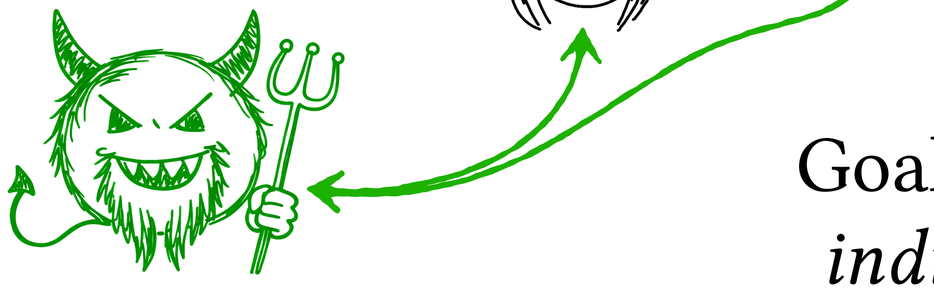
The Real World

1. The parties $(P_1, \ldots, P_n)$ run a protocol π on inputs $(x_1, \ldots, x_n)$
2. When π terminates, the parties output y.



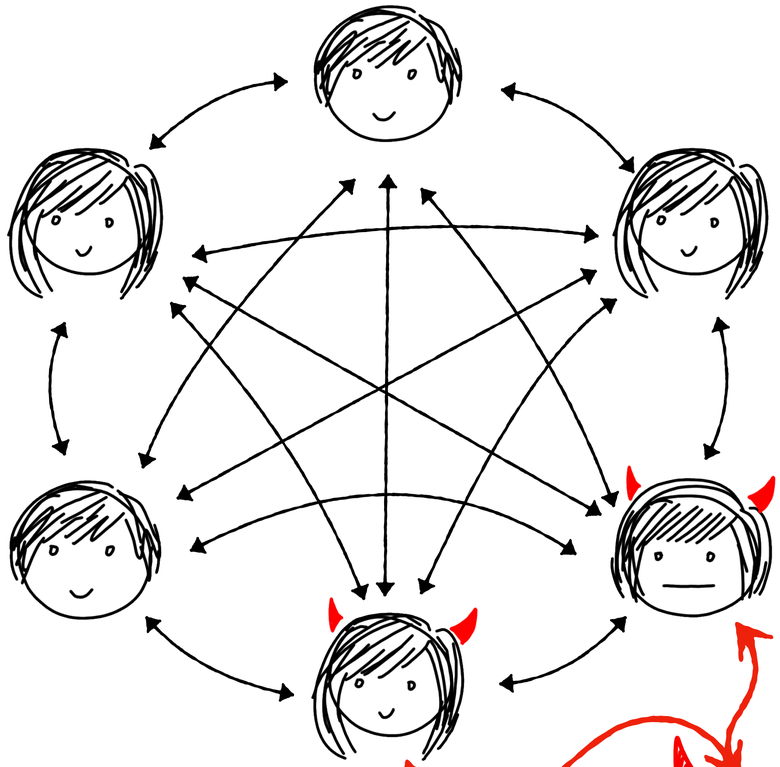
For every real adversary, an ideal adversary





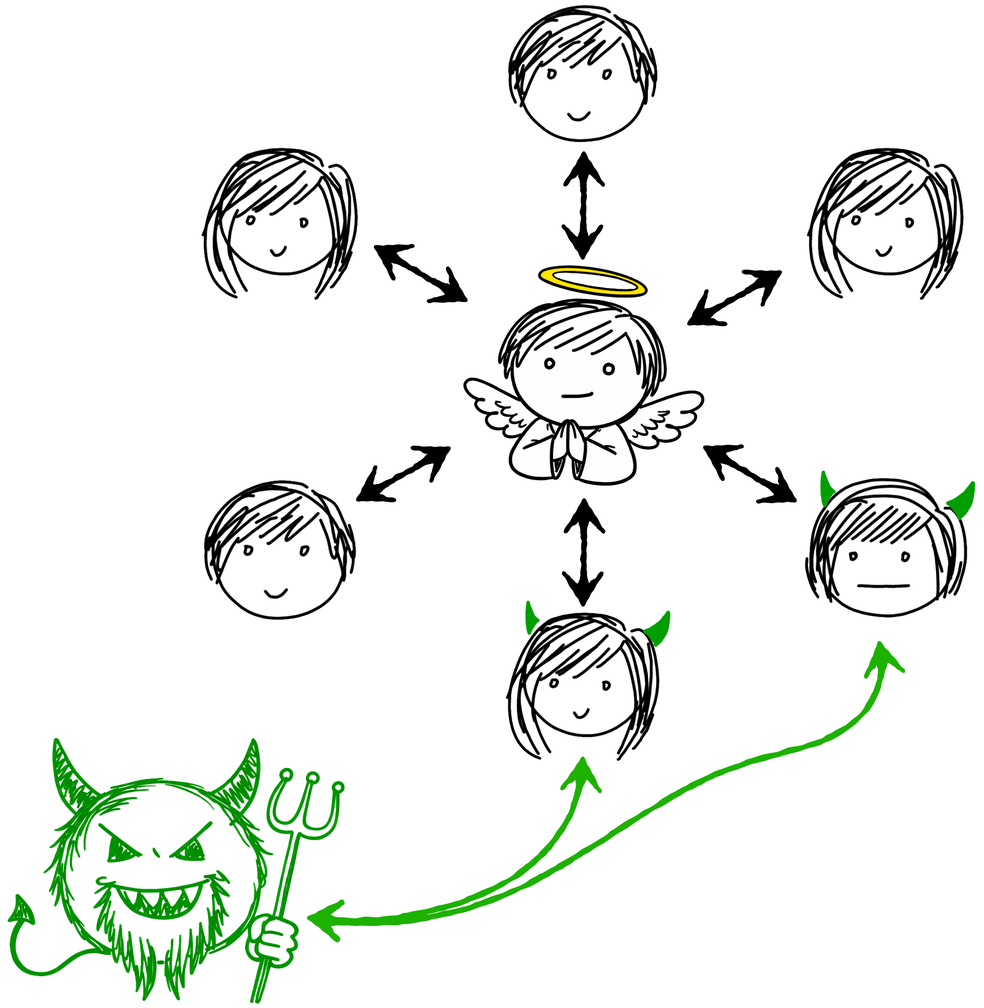
the Simulator $\mathcal{S}$

Goal of $\mathcal{S}$: produce output *indistinguishable* from $\mathcal{A}$

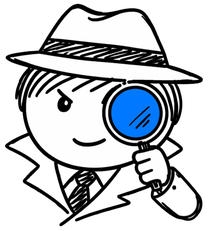


the Adversary $\mathcal{A}$

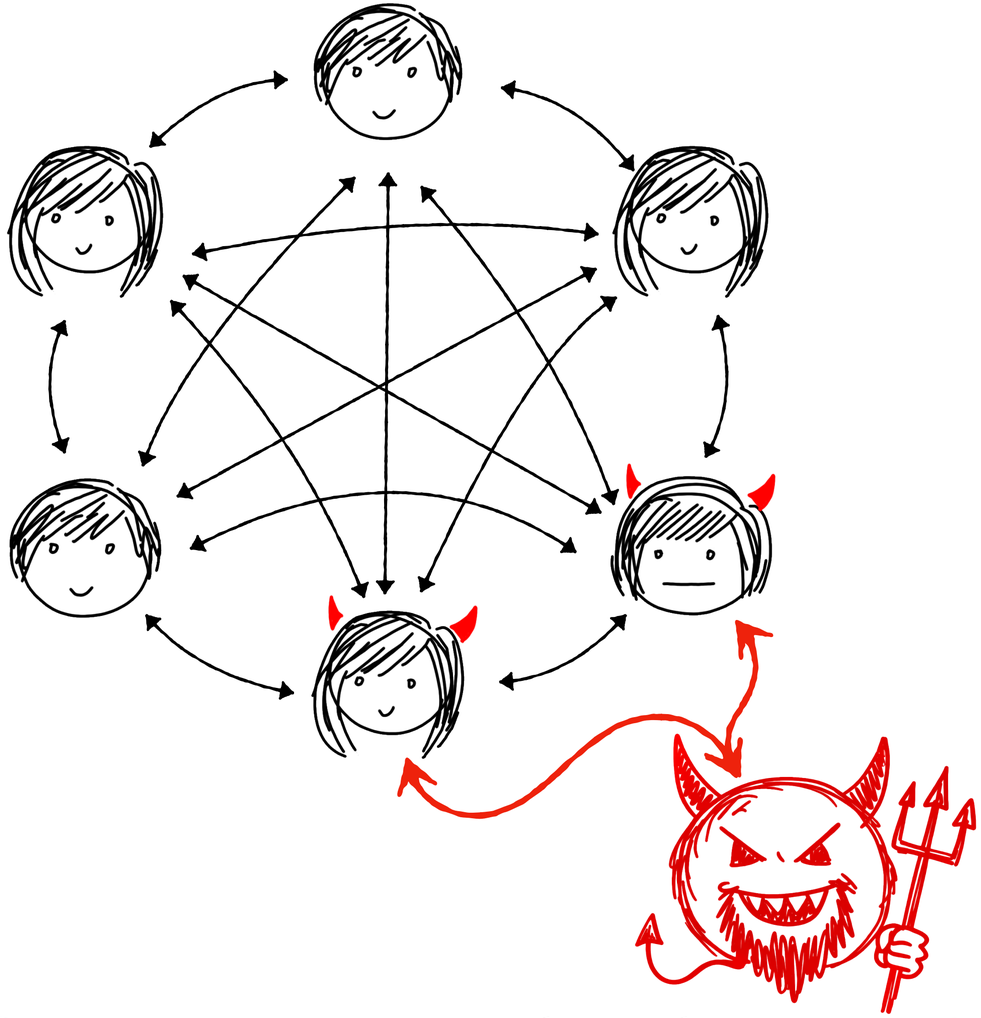
Is $\mathcal{S}$ is Indistinguishable from $\mathcal{A}$? Who will Judge?



the Simulator $\mathcal{S}$



the Distinguisher $\mathcal{D}$

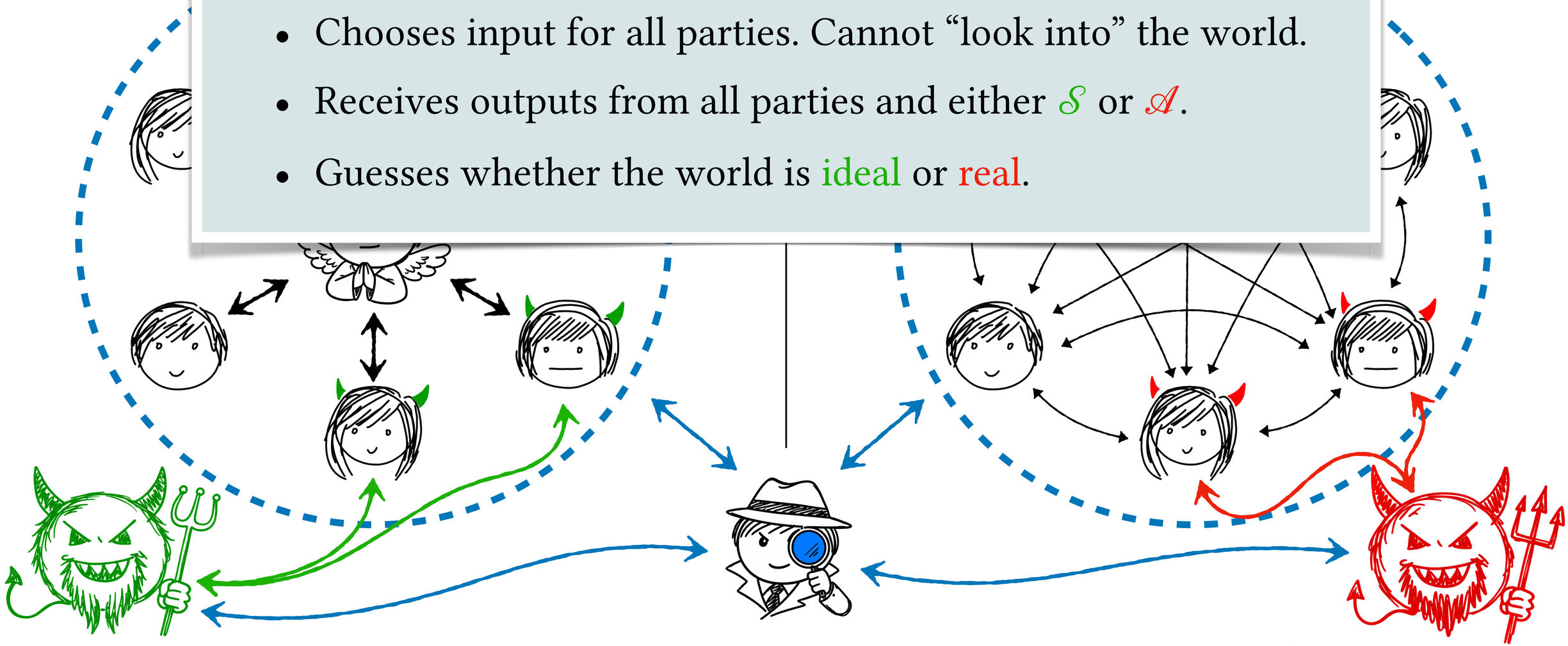


the Adversary $\mathcal{A}$

Is $\mathcal{S}$ is dge?

the Distinguisher $\mathcal{D}$

- Interacts with one of the worlds and attempts to determine *which* one by running an experiment.
- Chooses input for all parties. Cannot "look into" the world.
- Receives outputs from all parties and either $\mathcal{S}$ or $\mathcal{A}$.
- Guesses whether the world is ideal or real.



the Simulator $\mathcal{S}$ | the Distinguisher $\mathcal{D}$ | the Adversary $\mathcal{A}$

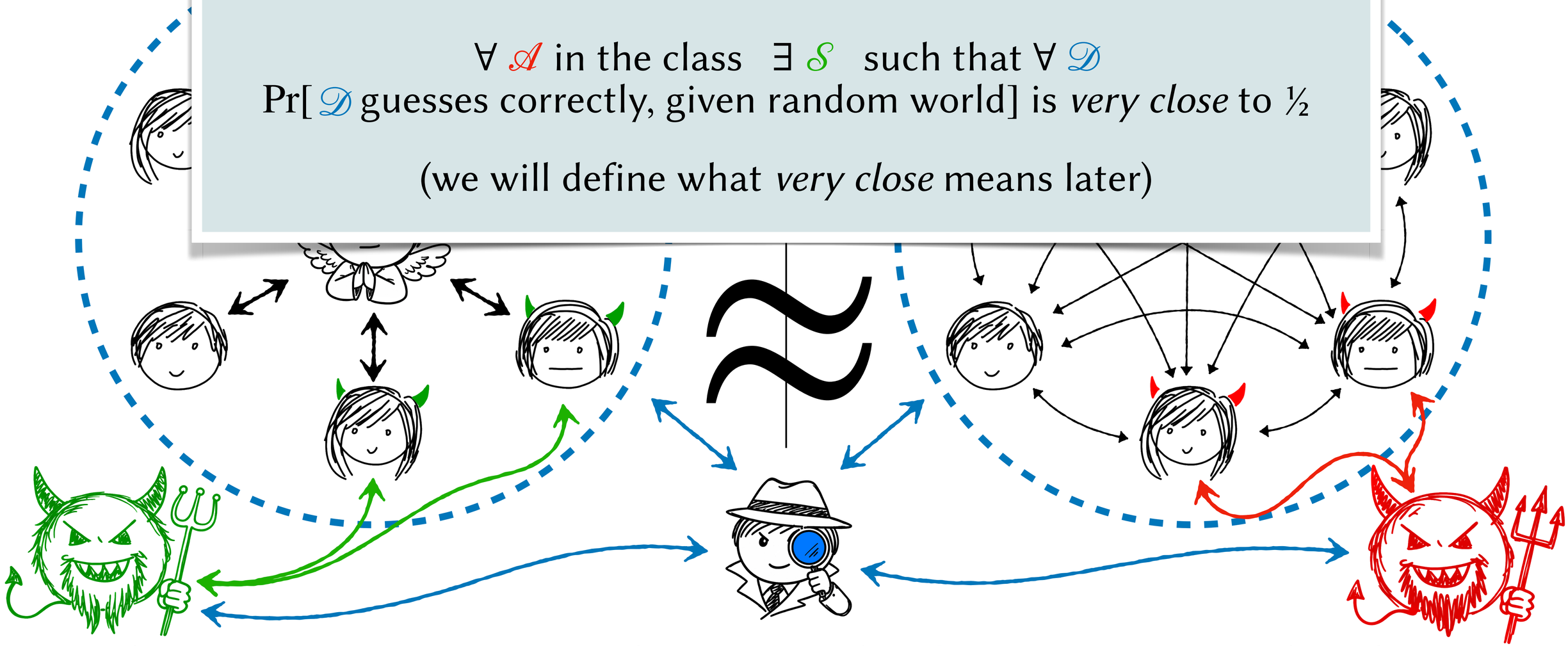
Is $\mathcal{S}$ is dge?

Finally, How to Define Simulation-Based Security!

The protocol π *realizes* the ideal functionality $\mathcal{F}$ in the presence of some class of adversaries if:

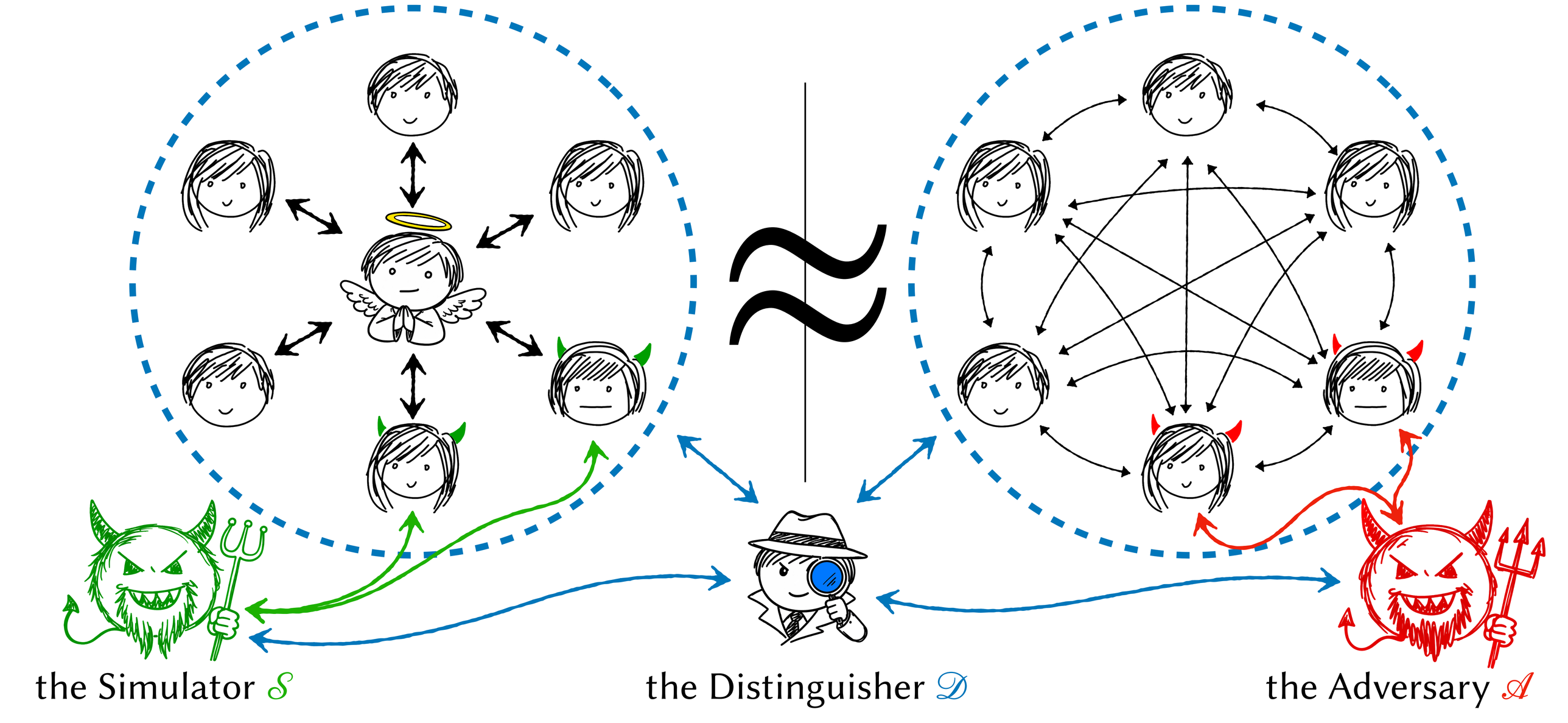
$\forall\ \mathcal{A}$ in the class $\exists\ \mathcal{S}$ such that $\forall\ \mathcal{D}$
Pr[$\mathcal{D}$ guesses correctly, given random world] is *very close* to ½

(we will define what *very close* means later)



the Simulator $\mathcal{S}$ the Distinguisher $\mathcal{D}$ the Adversary $\mathcal{A}$

The Simulation-Based Security Paradigm

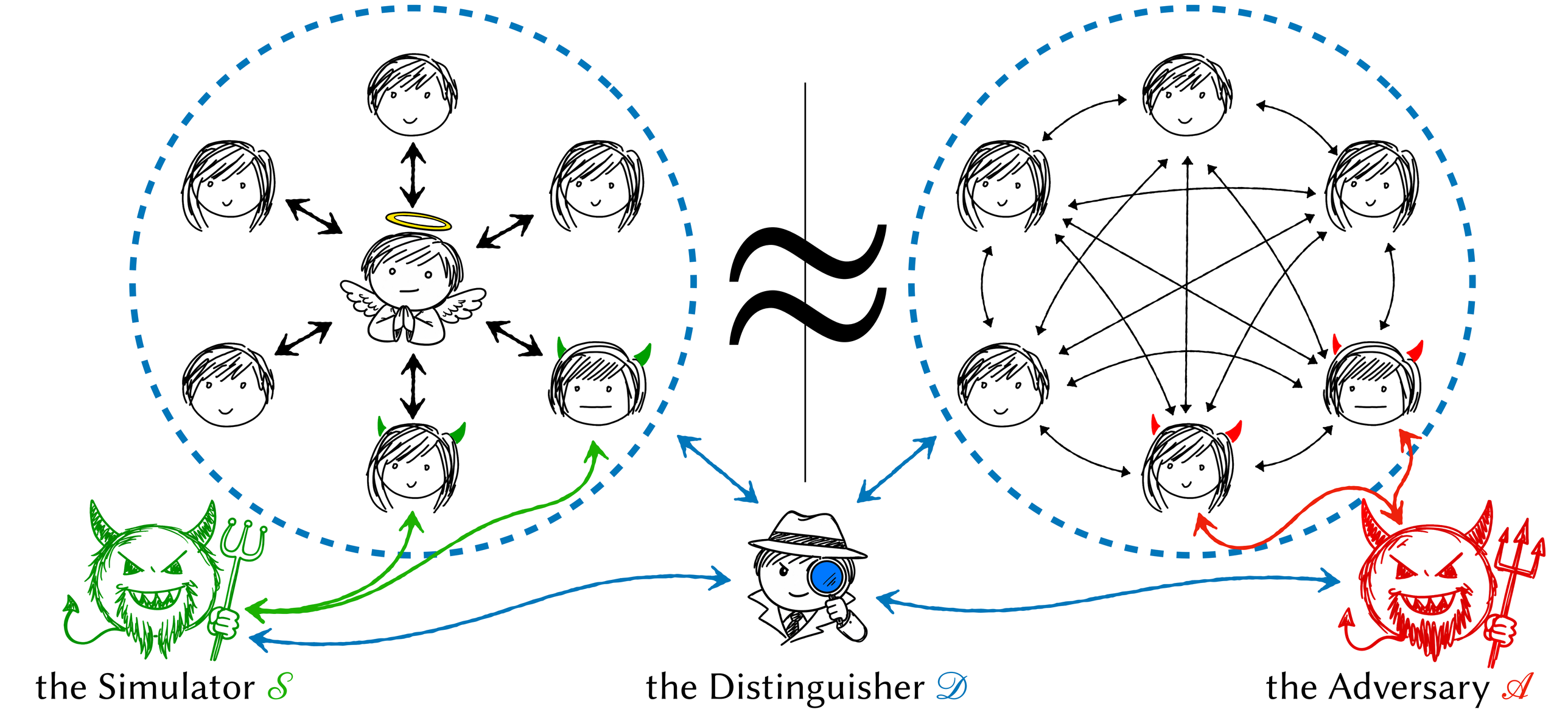


the Simulator $\mathcal{S}$

the Distinguisher $\mathcal{D}$

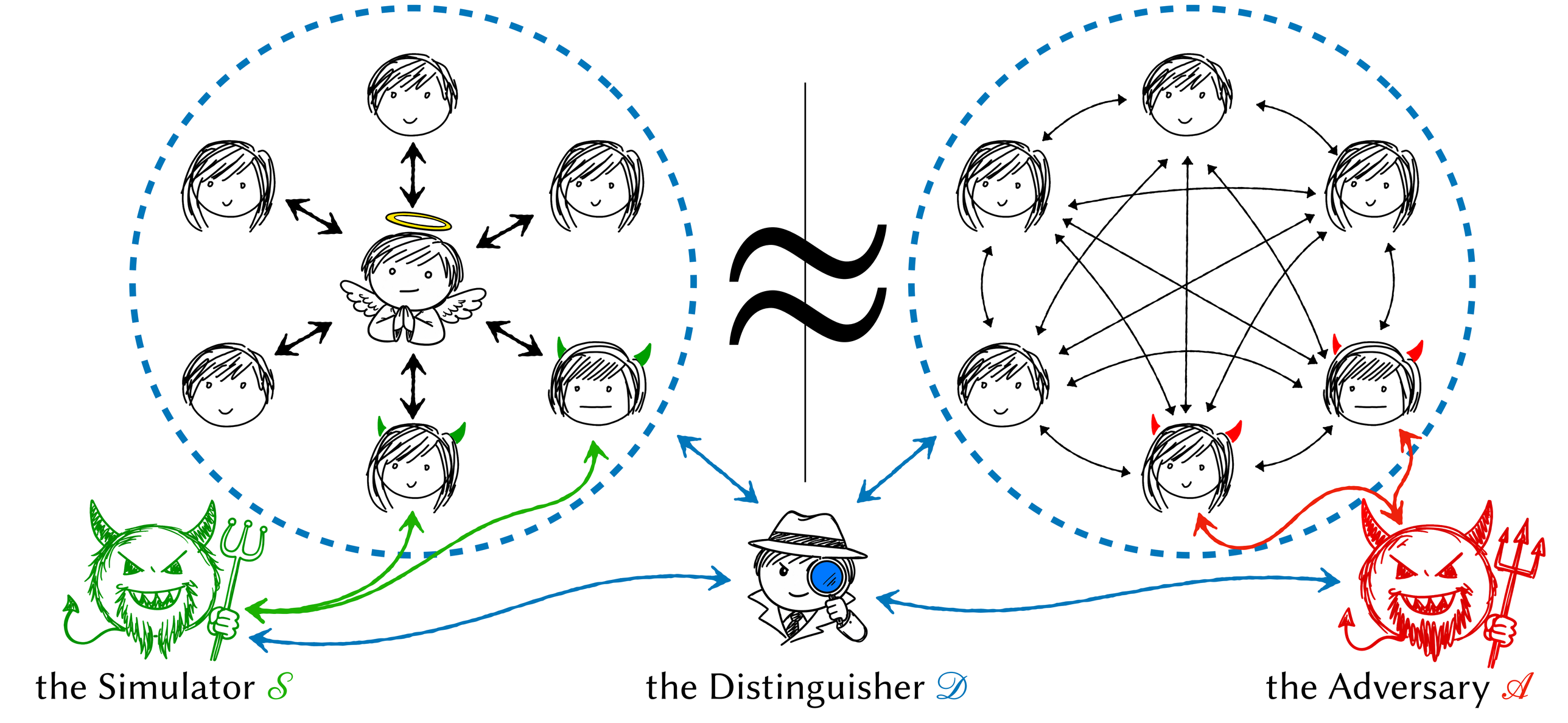
the Adversary $\mathcal{A}$

Sanity Check: Correctness?



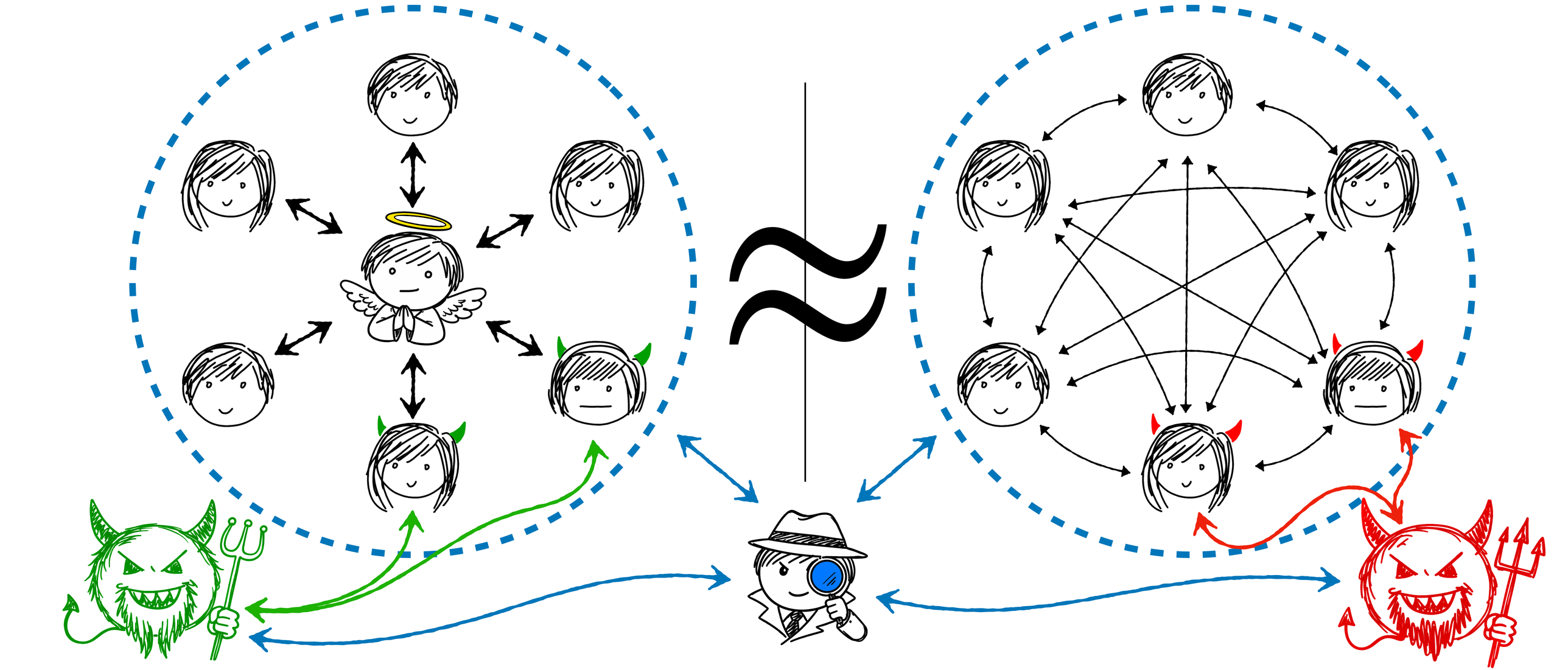
the Simulator $\mathcal{S}$ the Distinguisher $\mathcal{D}$ the Adversary $\mathcal{A}$

Sanity Check: Correctness?



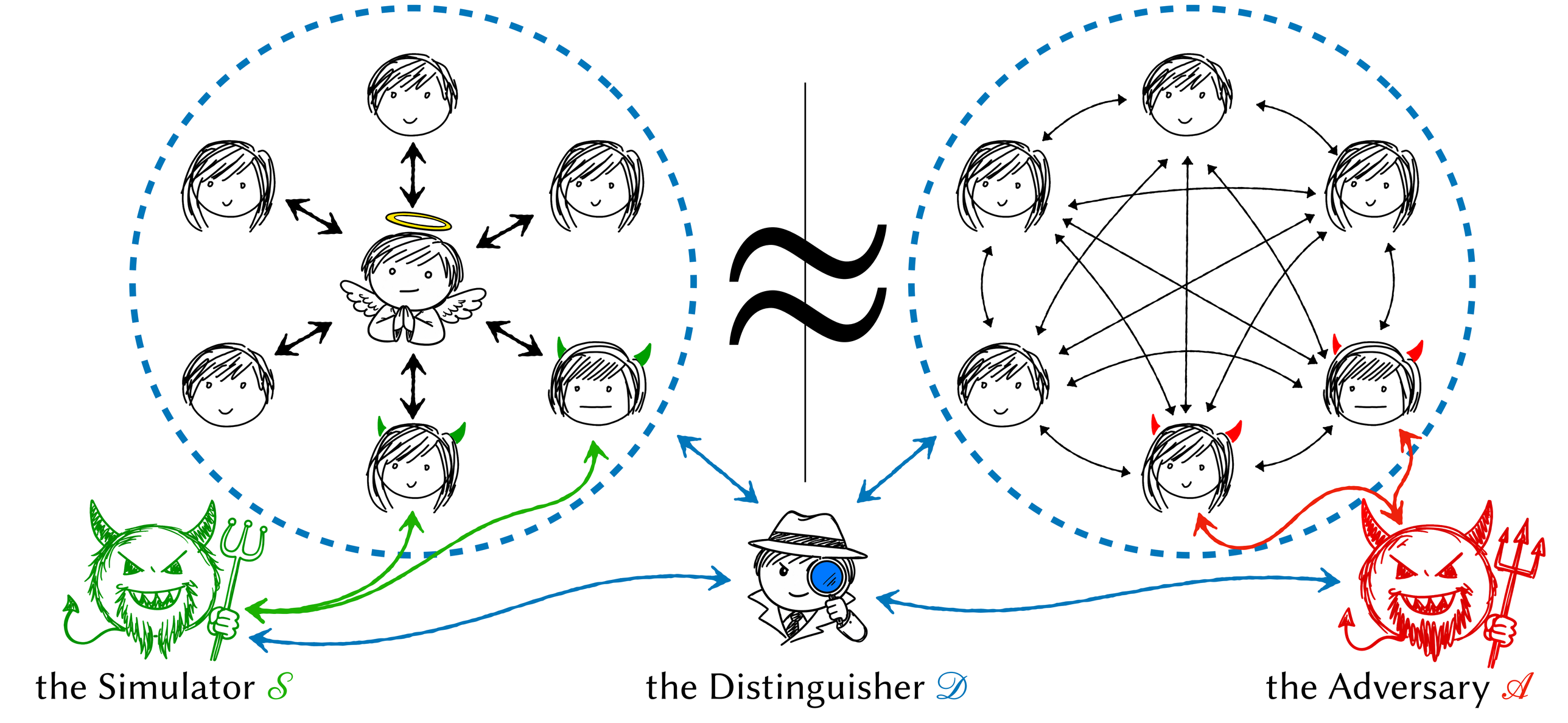
the Simulator $\mathcal{S}$ the Distinguisher $\mathcal{D}$ the Adversary $\mathcal{A}$

Sanity Check: Privacy?



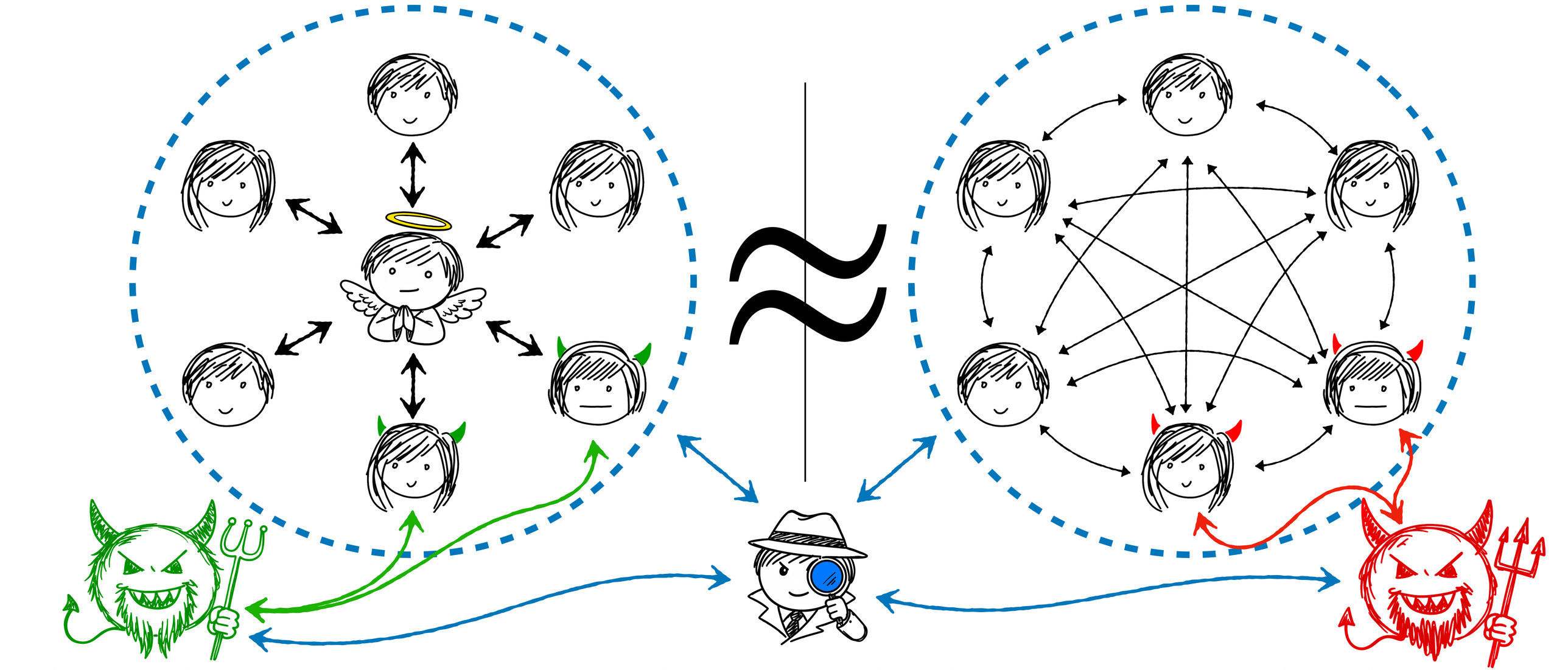
the Simulator $\mathcal{S}$ the Distinguisher $\mathcal{D}$ the Adversary $\mathcal{A}$

Sanity Check: Input Independence?



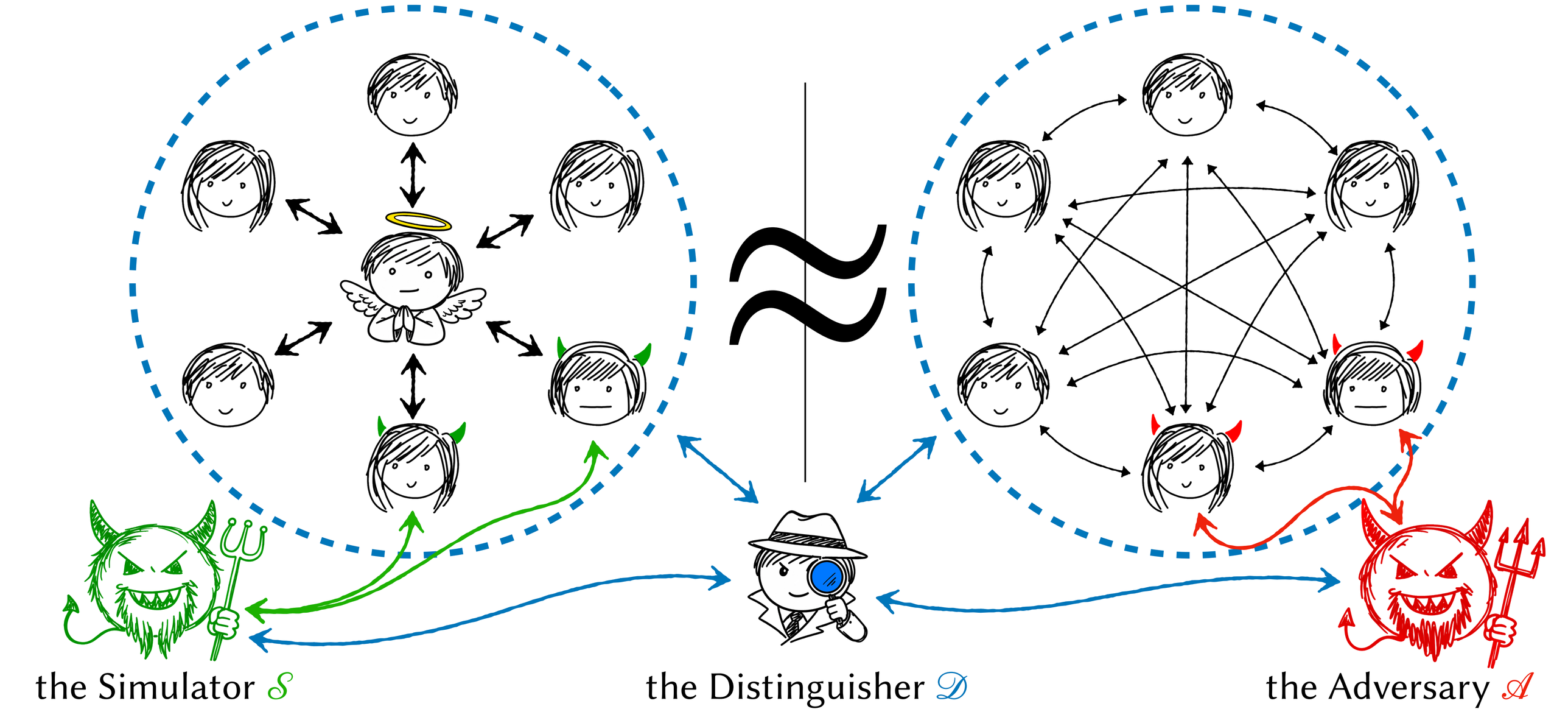
the Simulator $\mathcal{S}$ the Distinguisher $\mathcal{D}$ the Adversary $\mathcal{A}$

Sanity Check: Fairness?



the Simulator $\mathcal{S}$ the Distinguisher $\mathcal{D}$ the Adversary $\mathcal{A}$

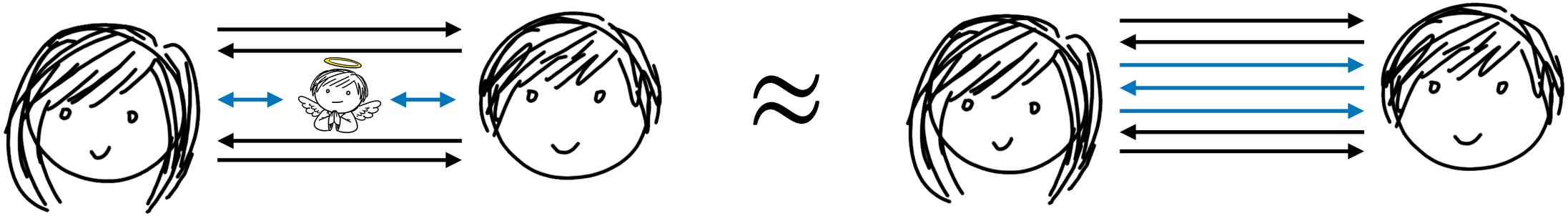
Sanity Check: Guaranteed Output Delivery?



the Simulator $\mathcal{S}$ the Distinguisher $\mathcal{D}$ the Adversary $\mathcal{A}$

This is complicated. Why do we do it?

- This way of defining security is *extremely* general. We can capture any computational task and security behavior using the code of the functionality.
- While the *model* is complicated, the *guarantee* is simple to understand: Just imagine a trusted party performing the task that you desire.
- The functionality gives a complete picture of the security that the protocol achieves. Nothing is accidentally missed.
- Supports *composition*: use functionalities as parties in larger protocols. Reason about security in a modular and reusable fashion!



Specifying the Details

We've met the players, but in order to understand what we're achieving, we must know more about them.

- **Functionality:** what do we want to compute, and with what IO behavior? A functionality can also capture *vulnerabilities* by directly taking inputs from or leaking information to the simulator $\mathcal{S}$.
- **Adversarial Model:** what kinds of behavior and interaction with the system do we want to protect against? E.g. who can be corrupted?
- **Security Type:** how strong should our protection be? How much computing power does $\mathcal{D}$ have? How much better than random is the guesswork of $\mathcal{D}$ allowed to be?
- **Network Model:** Who is connected to who in the real world? Are those connections private? Do the parties have a shared clock? Is there a way to *broadcast* reliably?

Facets of Adversarial Models

Behavior:

- *Semi-honest* (a.k.a. *Passive*): corrupted parties follow the protocol honestly, but share their internal state with $\mathcal{A}$, who tries to learn more than is allowed.
- *Malicious* (a.k.a. *Active*): corrupted parties can deviate from the protocol instructions in arbitrary ways. $\mathcal{A}$ coordinates their actions.

Adversarial Power:

- *Unbounded* (i.e. "all-powerful"): $\mathcal{A}$ has unlimited computing power.
  Can break any cryptographic assumption. We can still achieve security using information theory!
- *Computationally Bounded*: $\mathcal{A}$ runs in Probabilistic Polynomial Time (PPT).

Behavior:

- *Semi-honest* (a.k.a. *Passive*): corrupted parties follow the protocol honestly, but share their internal state with $\mathcal{A}$, who tries to learn more than is allowed.
- *Malicious* (a.k.a. *Active*): corrupted parties can deviate from the protocol instructions in arbitrary ways. $\mathcal{A}$ coordinates their actions.

Adversarial Power:

- *Unbounded* (i.e. "all-powerful"): $\mathcal{A}$ has unlimited computing power. Can break any cryptographic assumption. We can still achieve security using information theory!
- *Computationally Bounded*: $\mathcal{A}$ runs in Probabilistic Polynomial Time (PPT).

Corruption Strategy:

- *Static*: corruptions are determined at the beginning of the experiment. Honest parties always stay honest.
- *Adaptive*: $\mathcal{A}$ can dynamically corrupt parties during the protocol (security is very hard to achieve in this setting).

Security Types

Perfect:

- $\mathscr{D}$ and $\mathscr{A}$ have unbounded computational power.
- The real and ideal experiments must be identically distributed from the perspective of $\mathscr{D}$.
- $\mathscr{D}$ must be able to do no better than a random guess.

Statistical:

- $\mathscr{D}$ and $\mathscr{A}$ have unbounded computational power.
- The real and ideal experiments must be *statistically indistinguishable*.
  (Their *statistical distance* must be *negligible* relative to the *security parameter*)

Perfect:

- $\mathscr{D}$ and $\mathscr{A}$ have unbounded computational power.
- The real and ideal experiments must be identically distributed from the perspective of $\mathscr{D}$.
- $\mathscr{D}$ must be able to do no better than a random guess.

Statistical:

- $\mathscr{D}$ and $\mathscr{A}$ have unbounded computational power.
- The real and ideal experiments must be *statistically indistinguishable*. (Their *statistical distance* must be *negligible* relative to the *security parameter*)

Computational:

- $\mathscr{D}$ and $\mathscr{A}$ are *efficient*. They run in PPT.
- We can make *cryptographic* assumptions. That is, we can assume certain computational problems can't be solved by $\mathscr{D}$ or $\mathscr{A}$.
- The real and ideal experiments must be *computationally indistinguishable*. This means that the outputs of $\mathscr{D}$ are *statistically close* when it interacts with the real and ideal experiments, even though the experiments themselves might have *statistically far* distributions.

We need to formalize all of this mathematically!

But first, let's look at what we will achieve in this class.

The "Fundamental Theorem" of Multiparty Computation

Every computable function f can be securely computed
(even if $n - 1$ parties are corrupted)

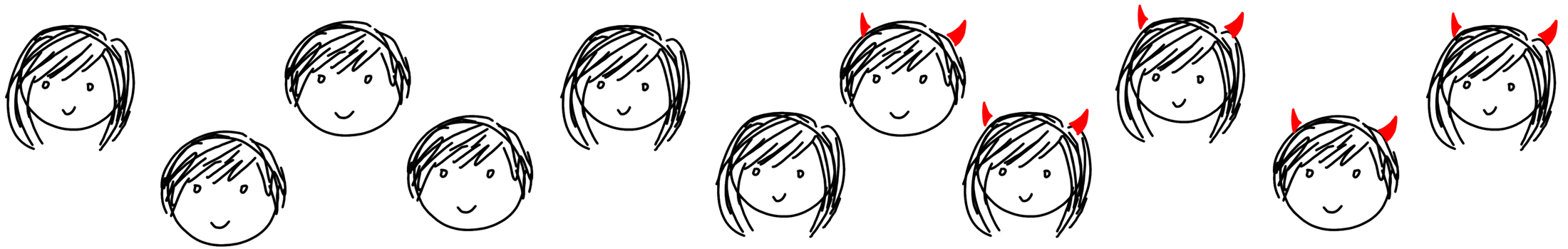
The "Fundamental Bound" of Multiparty Computation

If a majority of parties are maliciously corrupted, there exist functionalities $\mathcal{F}$ that cannot be realized regardless of what assumptions are made about the adversary's power.

Assuming an Honest Majority

Let n be the number of parties and let t be the number of corruptions

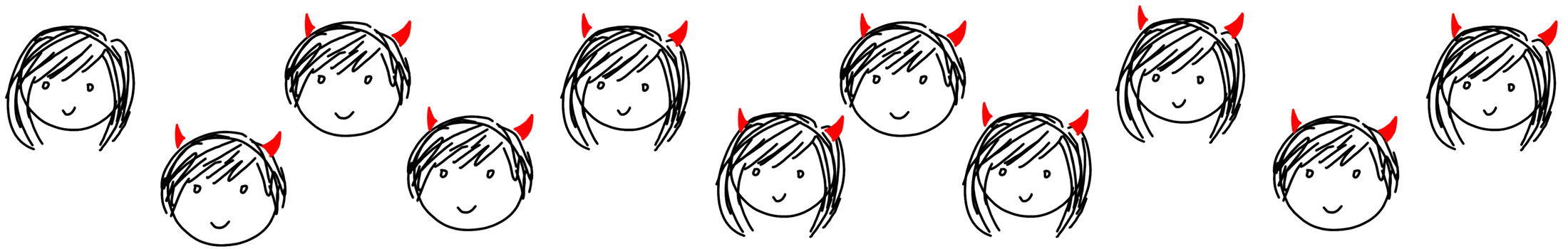
Behavior	Corruption Bound	Network Assumptions	Crypto Assumptions	Security
Semi-Honest	$t < n/2$	private, authenticated point-to-point channels	—	Perfect
Malicious	$t < n/3$	private, authenticated point-to-point channels	—	Perfect
Malicious	$t < n/2$	private, authenticated p2p channels + broadcast	—	Statistical



Assuming a *Dis*honest Majority

Let n be the number of parties and let t be the number of corruptions

Behavior	Corruption Bound	Network Assumptions	Crypto Assumptions	Security
Semi-Honest	$t < n$	authenticated point-to-point channels	Oblivious Transfer	Computational
Malicious	$t < n$	authenticated p2p channels + broadcast	Oblivious Transfer	Computational (*no fairness or GOD*)



Syllabus (tentative):

A taxonomy of adversaries; a variety of techniques

(now the taxonomy should be clearer than it was before)

	Part 1: *Information-theoretic* techniques. Adversaries with unbounded power	**Part 2:** *Cryptographic* techniques. Adversaries with bounded power
Semi-honest Adversaries: *follow* the rules of the protocol	Secret Sharing BGW protocol for an honest majority	Oblivious Transfer GMW protocol for a dishonest majority Yao's protocol for two parties Fully Homomorphic Encryption
Malicious Adversaries: *break* the rules of the protocol	Verifiable Secret Sharing BGW protocol for honest supermajority	Coin Tossing Zero-Knowledge Proofs GMW Compiler Byzantine Agreement + Broadcast

Overarching Questions:

How do we characterize unknown adversaries? How do we formalize intuitive security notions?
What kinds computation can we perform securely in each setting?

CS4501 Cryptographic Protocols
Lecture 2: Adversaries and Simulation

https://jackdoerner.net/teaching/#2026/Spring/CS4501