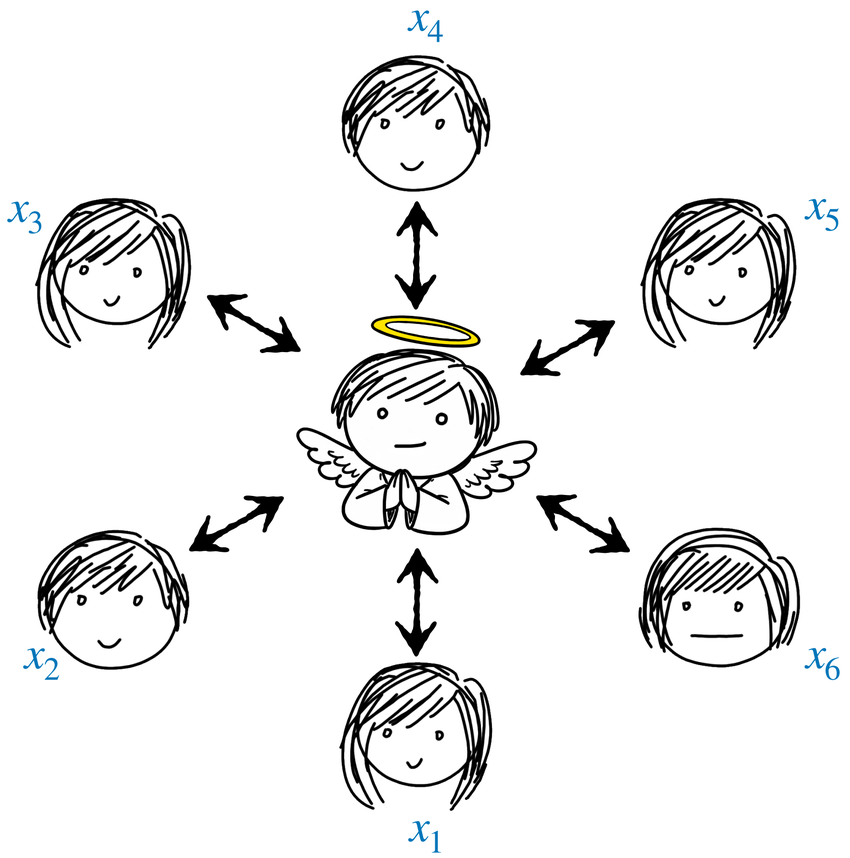
# CS4501 Cryptographic Protocols
# Lecture 19: OT is Symmetric
# Byzantine Agreement, Broadcast

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

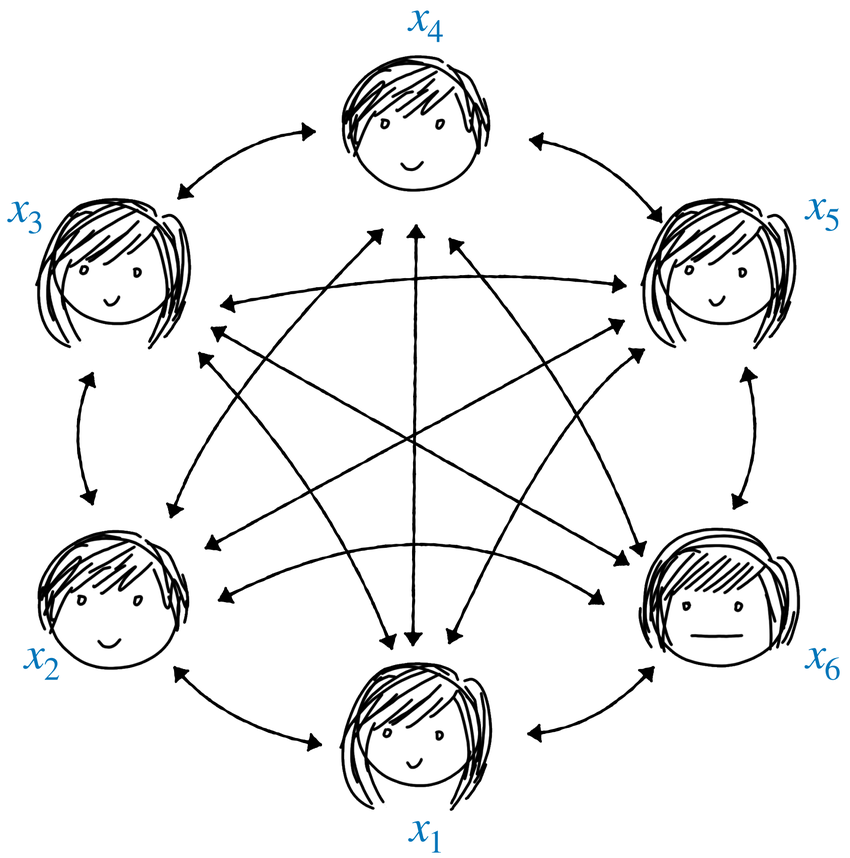
# Recap: The Ideal World

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.

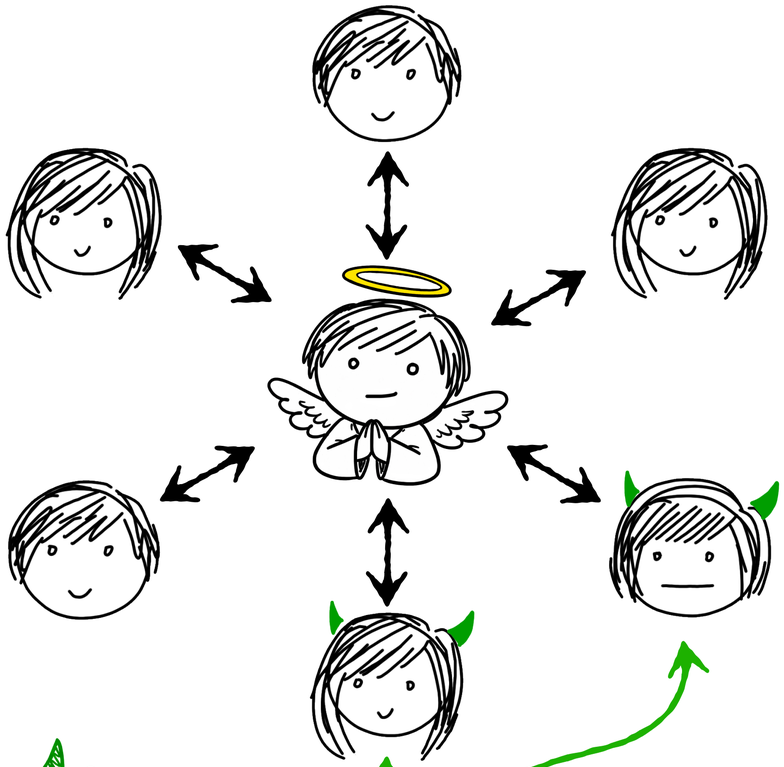


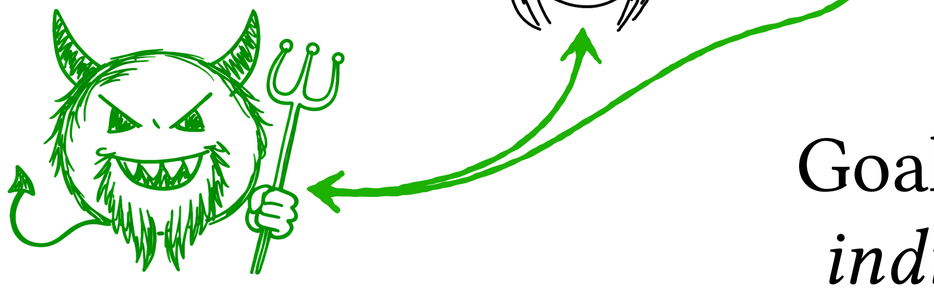
# Recap: The Real World

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



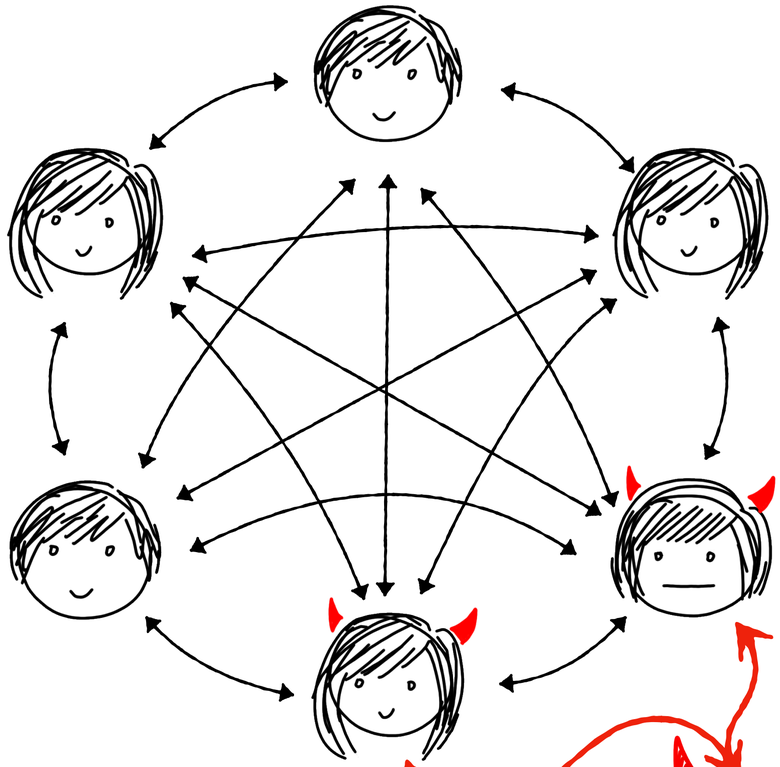
# For every real adversary, an ideal adversary





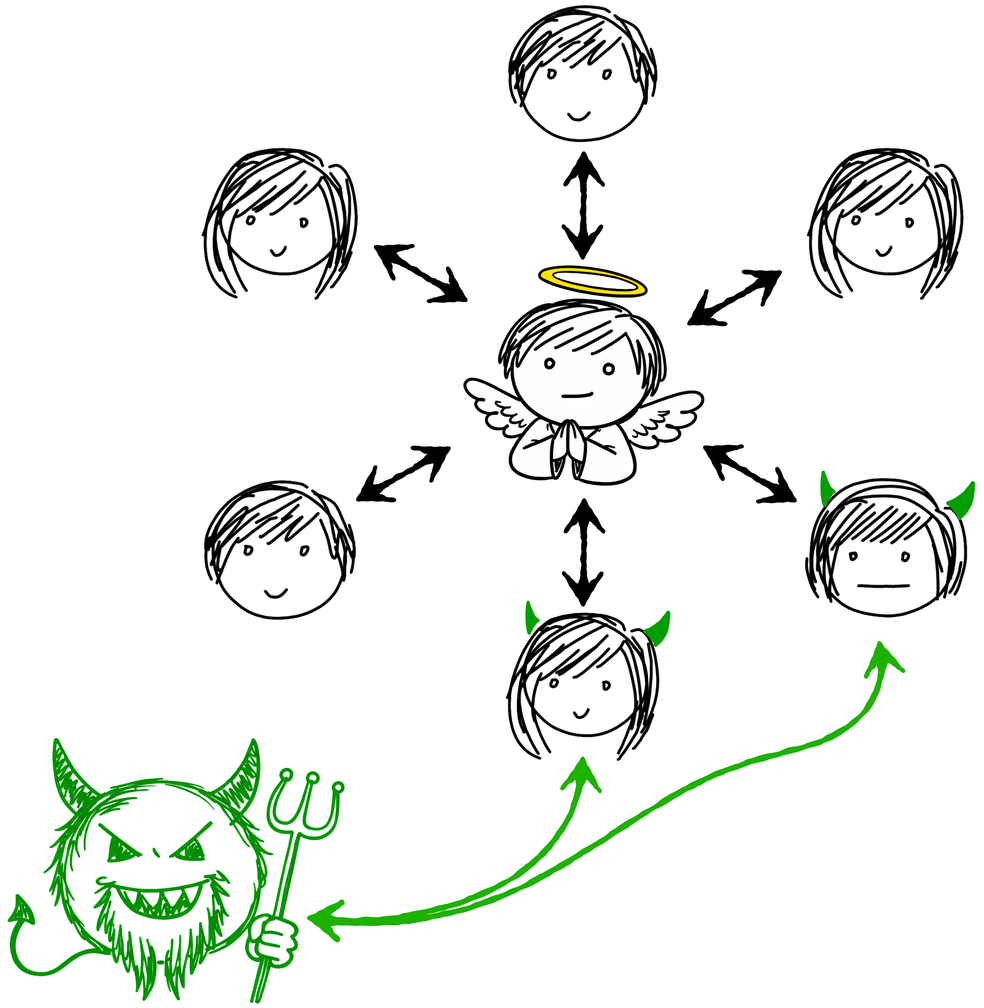
the Simulator $\mathcal{S}$

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

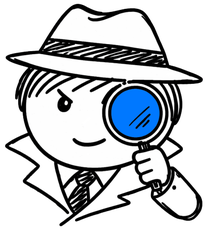


the Adversary $\mathcal{A}$

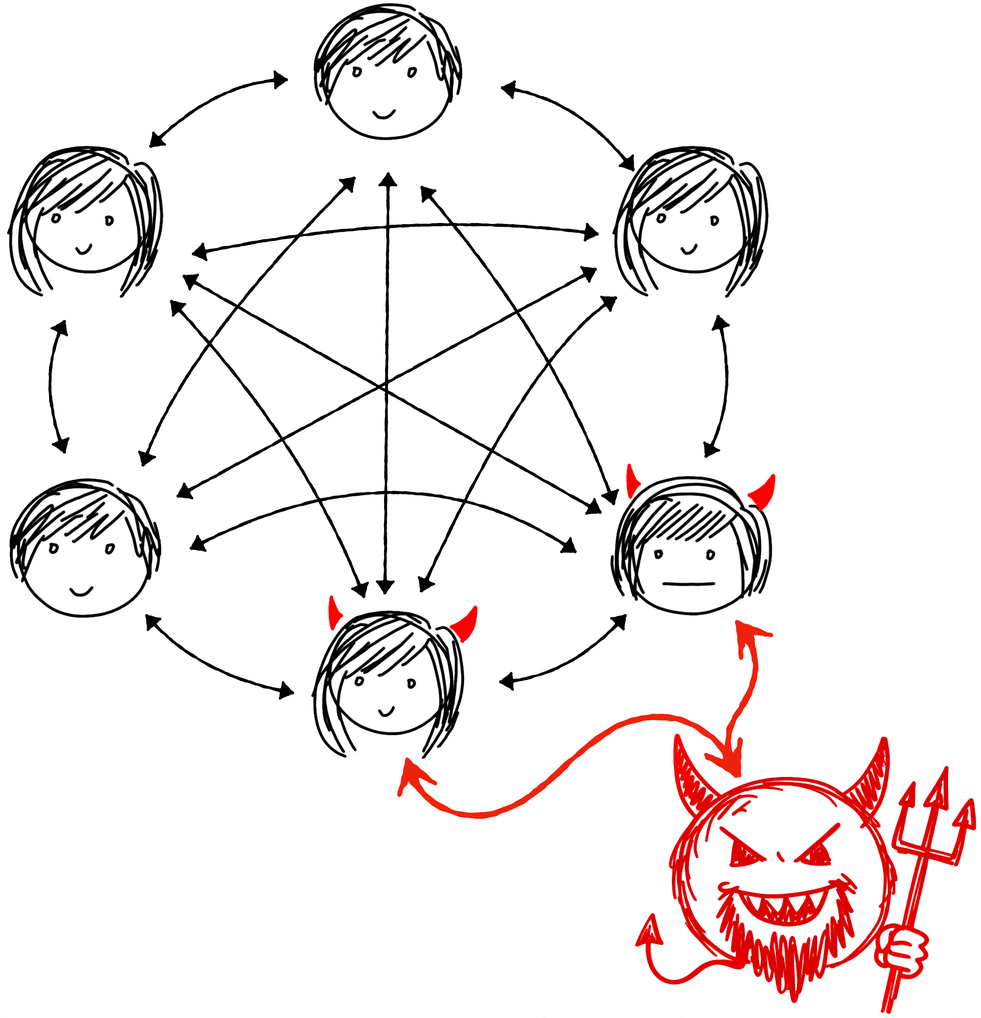
# Is $\mathcal{S}$ is Indistinguishable from $\mathcal{A}$? Who will Judge?



the Simulator $\mathcal{S}$



the Distinguisher $\mathcal{D}$

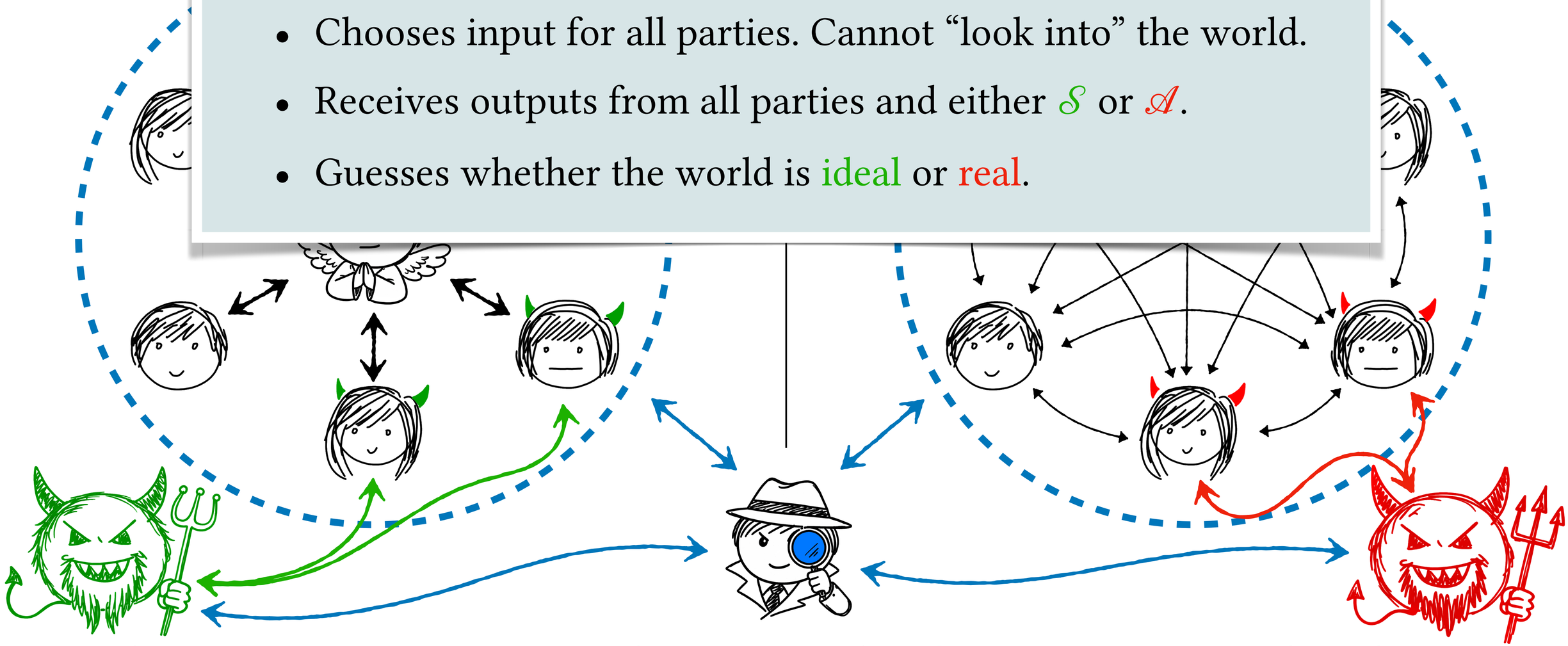


the Adversary $\mathcal{A}$

# Is $\mathcal{S}$ is dge?

the Distinguisher $\mathcal{D}$

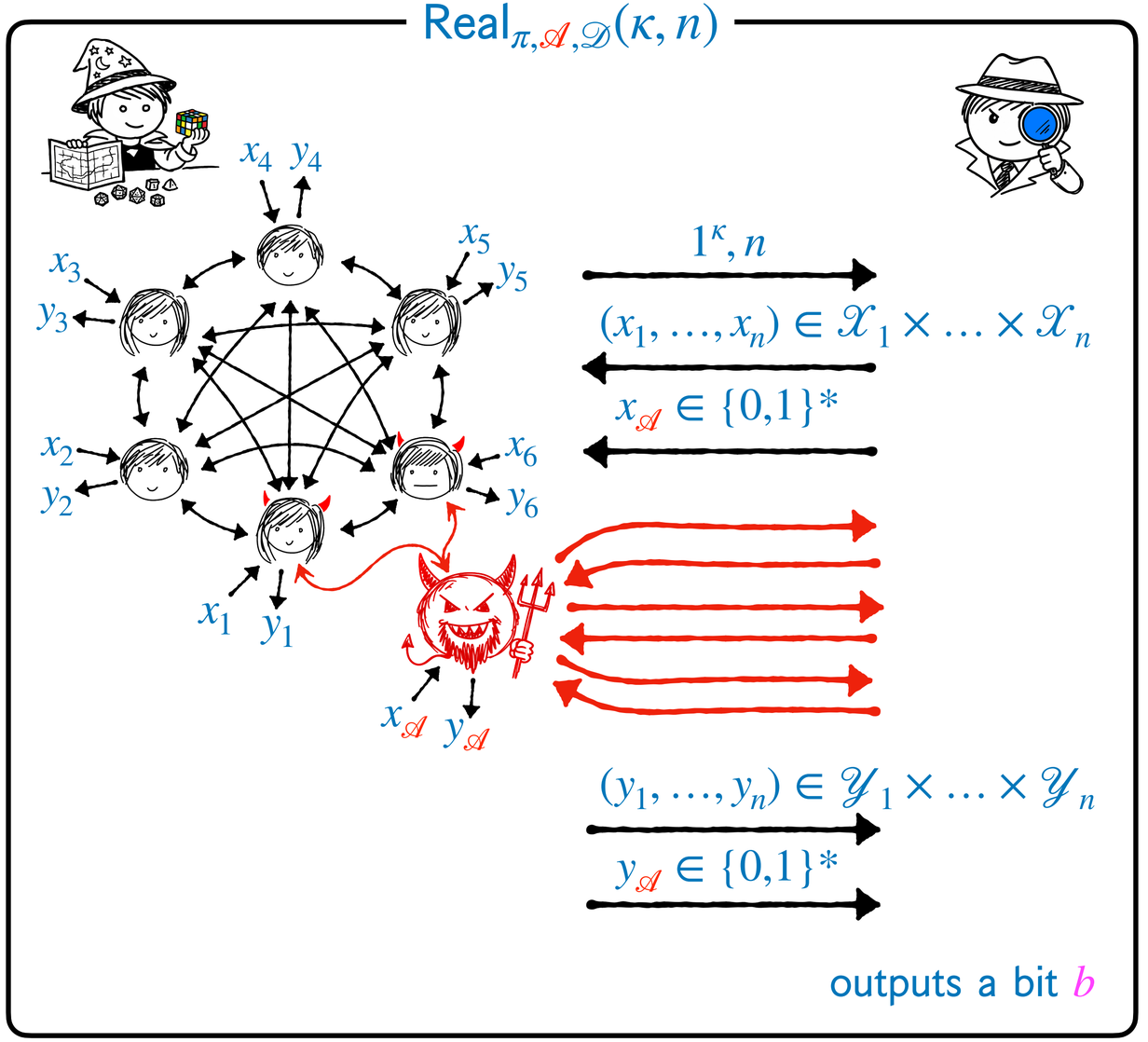
- Interacts with one of the worlds and attempts to determine *which* one by running an experiment.
- Chooses input for all parties. Cannot "look into" the world.
- Receives outputs from all parties and either $\mathcal{S}$ or $\mathcal{A}$.
- Guesses whether the world is ideal or real.



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

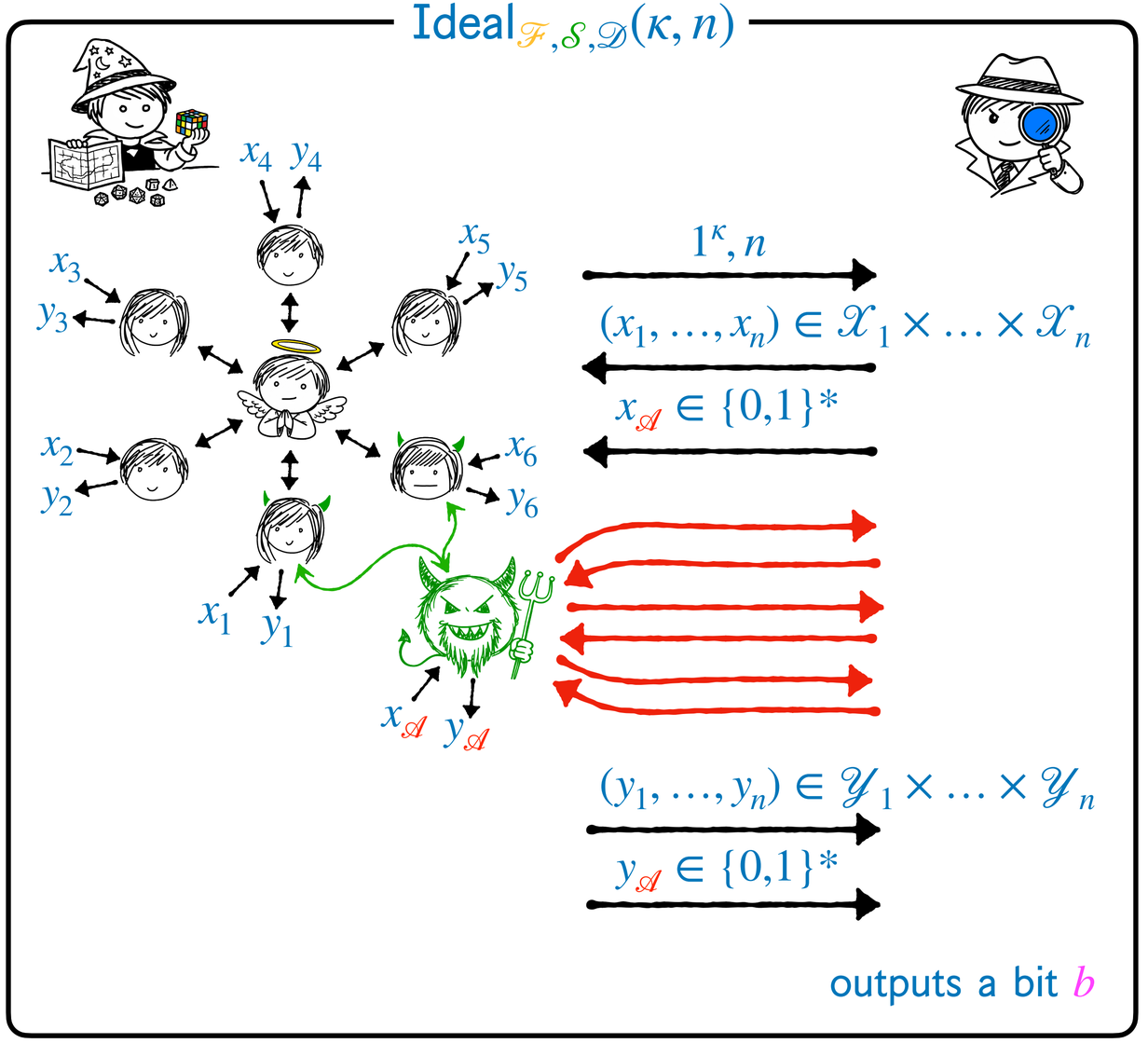
# Recap **Def 1**: The Real-World Experiment

1. The challenger communicates the parameters.
2. $\mathscr{D}$ supplies inputs for all parties and $\mathscr{A}$.
3. $\mathscr{A}$ corrupts some parties.
4. The protocol runs. During this time, $\mathscr{A}$ fully controls any corrupted parties.
5. $\mathscr{A}$ may send messages to $\mathscr{D}$, and receive responses.
6. The protocol ends when everyone halts with some output (including $\mathscr{A}$). These outputs are sent to $\mathscr{D}$.
7. $\mathscr{D}$ outputs a bit.



# Recap **Def 2**: The Ideal-World Experiment

1. The challenger communicates the parameters.
2. $\mathscr{D}$ supplies inputs for all parties and $\mathscr{A}$ (but it is received by $\mathscr{S}$).
3. $\mathscr{S}$ corrupts some parties.
4. The "dummy protocol" runs. During this time, $\mathscr{S}$ fully controls any corrupted parties (but they only interact with $\mathscr{F}$).
5. $\mathscr{S}$ may send messages to $\mathscr{D}$ (pretending to be $\mathscr{A}$).
6. The protocol ends when everyone halts with some output (including $\mathscr{S}$). These outputs are sent to $\mathscr{D}$.
7. $\mathscr{D}$ outputs a bit.



# Recap **Def 3**: Malicious Security

**Definition 3.** Let $n, t \in \mathbb{N}$ such that $t < n$, let $\mathcal{F}$ be a PPT ideal functionality that interacts with $n$ parties and (possibly) an adversary, and let $\pi$ be a PPT $n$-party protocol.

We say that $\pi$ *realizes* $\mathcal{F}$ in the presence of a *malicious* adversary that statically corrupts up to $t$ parties if for every PPT $\mathcal{A}$ that statically, maliciously corrupts up to $t$ parties, there exists some PPT simulator $\mathcal{S}$, such that for every PPT distinguisher $\mathcal{D}$, there exists a negligible function $\varepsilon$ such that for all $\kappa \in \mathbb{N}$

$$\left| \Pr[\mathsf{Real}_{\pi,\mathcal{A},\mathcal{D}}(\kappa, n) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{D}}(\kappa, n) = 1] \right| \leq \varepsilon(\kappa)$$

- For our previous definitions we had some intuition that "anything the adversary could achieve by corrupting the protocol it could also achieve using just the inputs and outputs of the corrupt party (via the Sim algorithm)"
- Now we have something richer! The adversary can *deviate arbitrarily* and the simulator *interacts* with the ideal functionality. Anything anything the adversary could achieve by corrupting the protocol it could also achieve in *the ideal interaction*.

# Our first maliciously-secure protocol!

- To give you a feeling for how a proof of malicious security fits together, we will prove that *OT is information-theoretically reversible*.
- Suppose Alice has two messages and Bob has a choice bit, and they want to perform OT. The *good* news is that they have access to an OT functionality. The *bad* news is that their OT functionality "goes the wrong way." It expects messages from Bob and a choice bit from Alice. We'll call this backwards OT functionality "TO."


- We will prove that OT and TO for one-bit messages are equivalent, even in the presence of a malicious adversary that corrupts one of the parties.
- Specifically, there exists a *one-message* protocol that *perfectly* realizes one-bit OT in the one-bit TO-hybrid model.

# Malicious OT in the TO-hybrid model

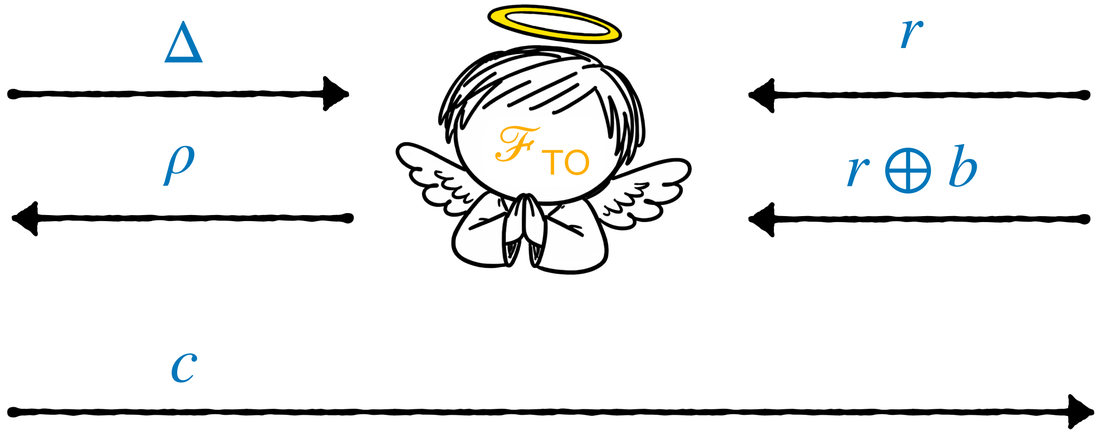
Let $\mathcal{M} = \{0,1\}$

receives inputs $m_0, m_1$

$\Delta := m_0 \oplus m_1$

receives input $b$

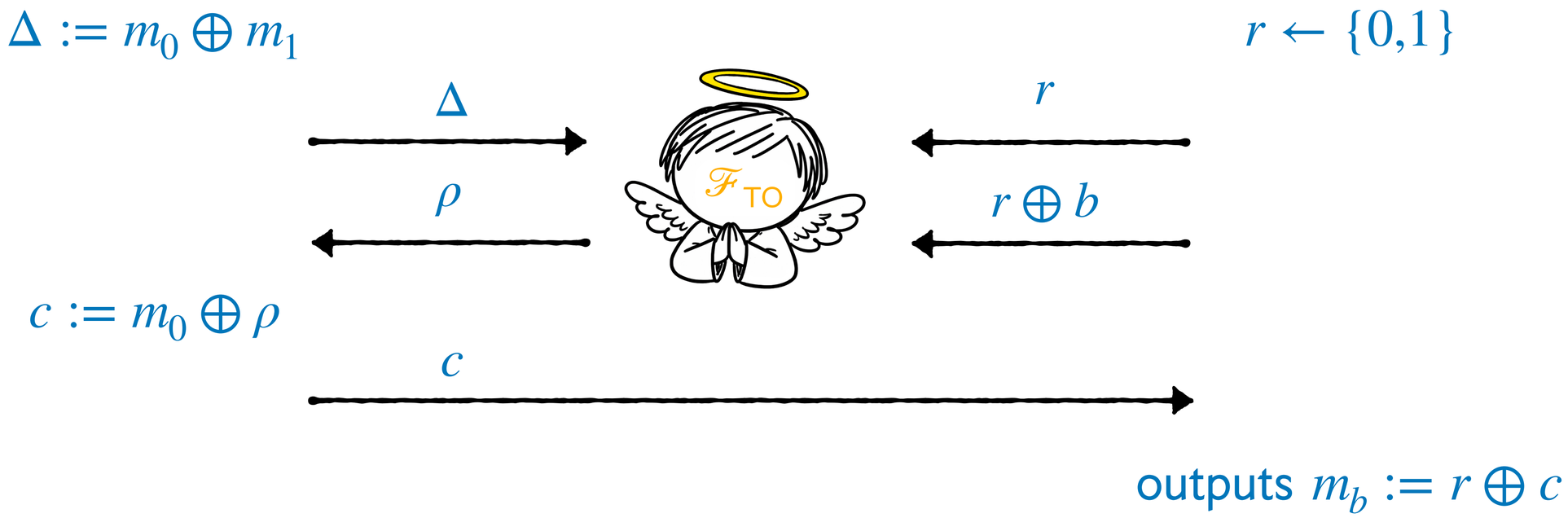
$r \leftarrow \{0,1\}$




$c := m_0 \oplus \rho$

outputs $m_b := r \oplus c$

# Correctness?




$$\rho = r \oplus (\Delta \wedge b)$$

$$\implies c = m_0 \oplus r \oplus (\Delta \wedge b)$$

$$= r \oplus m_0 \oplus (m_0 \wedge b) \oplus (m_1 \wedge b)$$

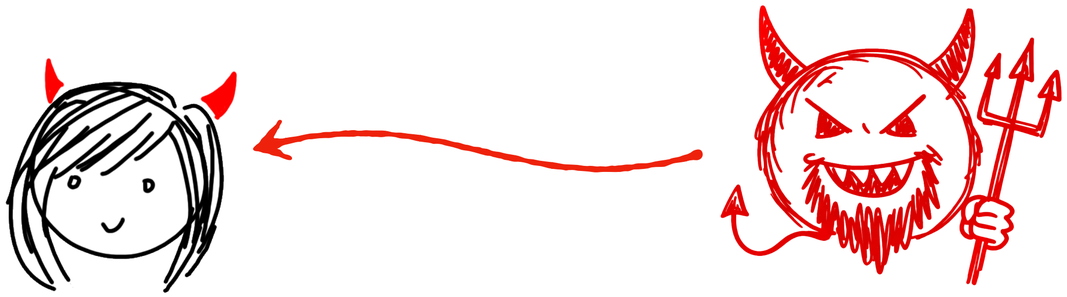
$$= r \oplus (m_0 \wedge (1 \oplus b)) \oplus (m_1 \wedge b)$$

$$\implies m_b = (m_0 \wedge (1 \oplus b)) \oplus (m_1 \wedge b)$$

... so the protocol is correct.

# Simulation for Alice

Suppose $\mathcal{A}$ maliciously corrupts Alice. We must construct $\mathcal{S}_{\mathsf{A}}$ that produces an indistinguishable *experiment* (not just view) in the ideal world.

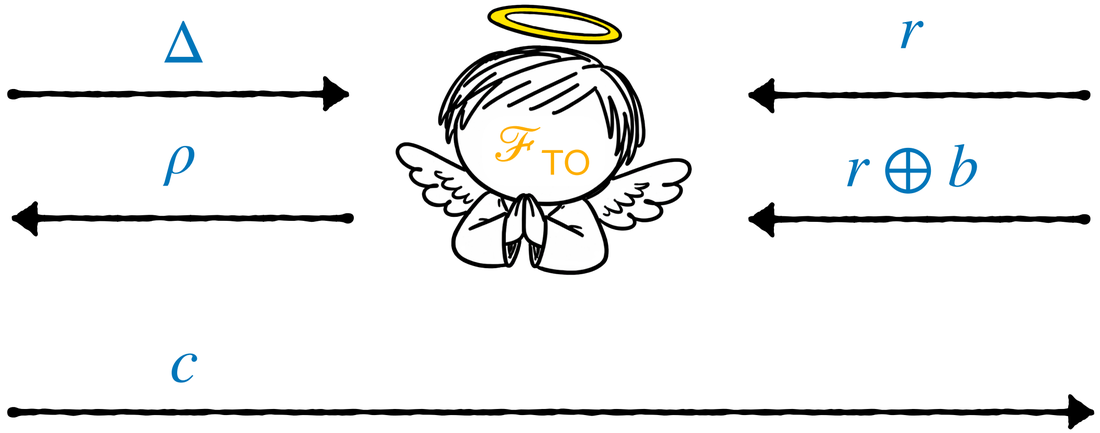


receives inputs $m_0, m_1$

$\Delta := m_0 \oplus m_1$

receives input $b$

$r \leftarrow \{0,1\}$



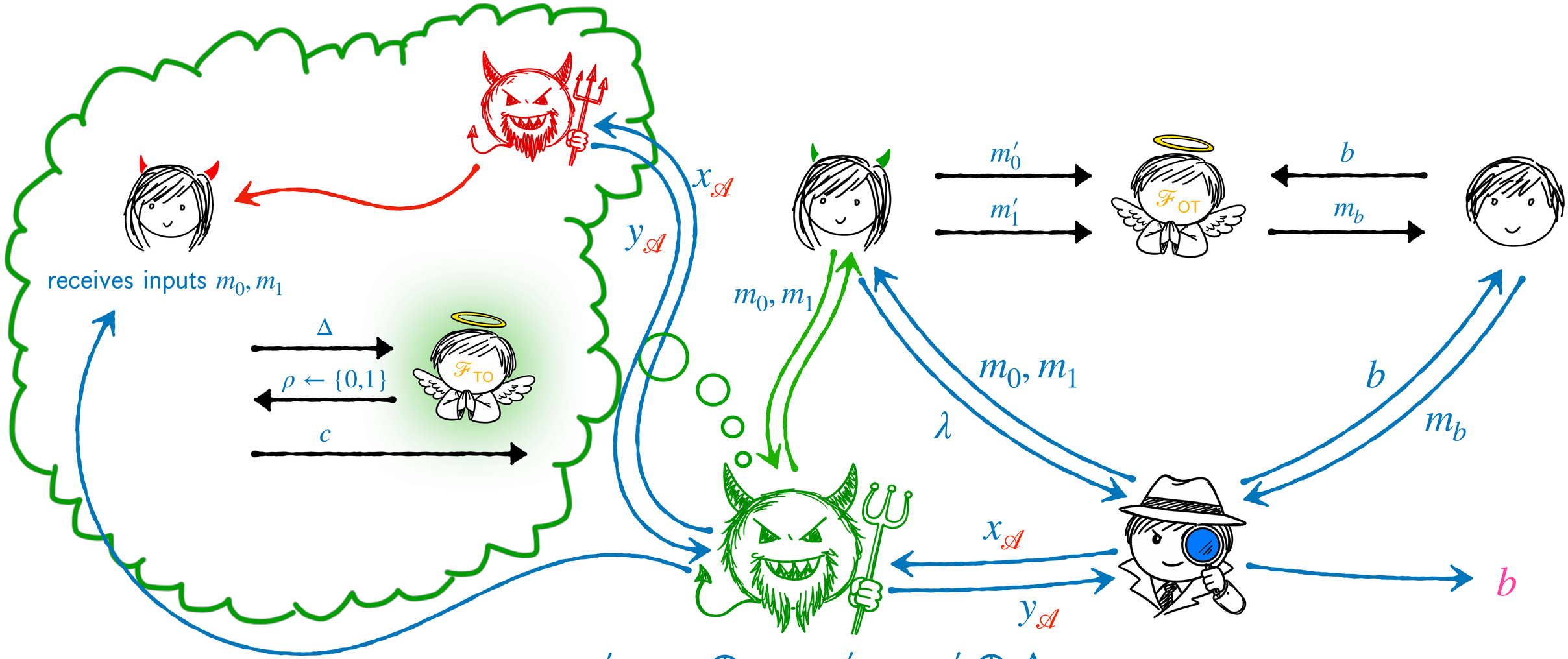

$c := m_0 \oplus \rho$

$c$

outputs $m_b := r \oplus c$

# Simulation for Alice

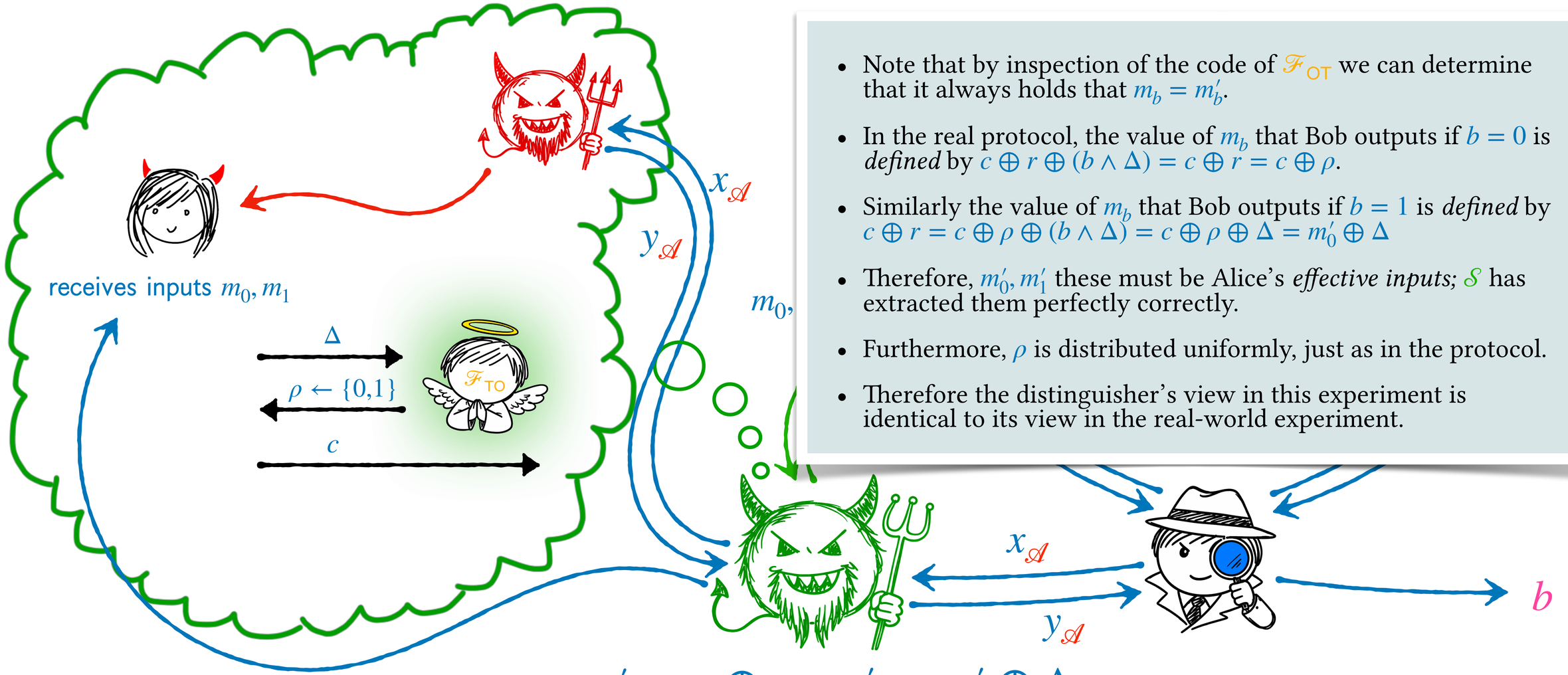
Suppose $\mathcal{A}$ maliciously corrupts Alice. We must construct $\mathcal{S}_{\mathsf{A}}$ that produces an indistinguishable *experiment* (not just view) in the ideal world.




$$m_0' := c \oplus \rho \quad m_1' := m_0' \oplus \Delta$$

# Simulation for Alice

Suppose $\mathcal{A}$ maliciously corrupts Alice. We must construct $\mathcal{S}_A$ that produces an indistinguishable *experiment* (not just view) in the ideal world.

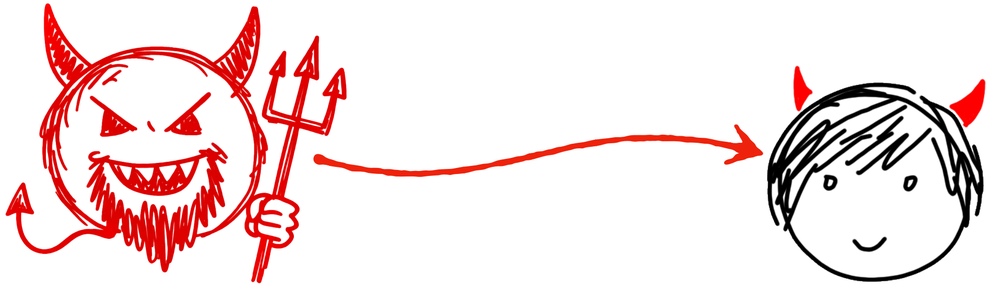



- Note that by inspection of the code of $\mathcal{F}_{\text{OT}}$ we can determine that it always holds that $m_b = m'_b$.
- In the real protocol, the value of $m_b$ that Bob outputs if $b = 0$ is *defined* by $c \oplus r \oplus (b \wedge \Delta) = c \oplus r = c \oplus \rho$.
- Similarly the value of $m_b$ that Bob outputs if $b = 1$ is *defined* by $c \oplus r = c \oplus \rho \oplus (b \wedge \Delta) = c \oplus \rho \oplus \Delta = m'_0 \oplus \Delta$
- Therefore, $m'_0, m'_1$ these must be Alice's *effective inputs;* $\mathcal{S}$ has extracted them perfectly correctly.
- Furthermore, $\rho$ is distributed uniformly, just as in the protocol.
- Therefore the distinguisher's view in this experiment is identical to its view in the real-world experiment.

$$m'_0 := c \oplus \rho \quad m'_1 := m'_0 \oplus \Delta$$

# Simulation for Bob

Suppose $\mathcal{A}$ maliciously corrupts Bob. We must construct $\mathcal{S}_B$ that produces an indistinguishable *experiment* (not just view) in the ideal world.

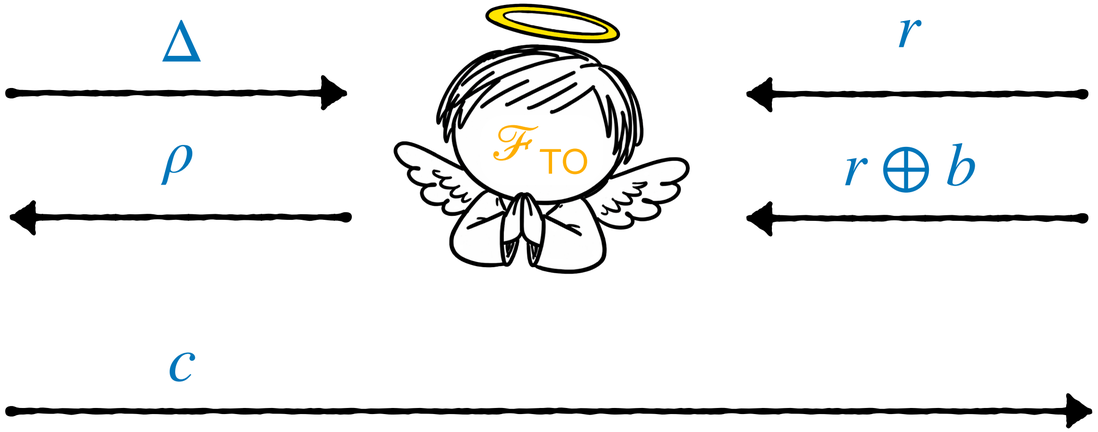


receives inputs $m_0, m_1$

$\Delta := m_0 \oplus m_1$

receives input $b$

$r \leftarrow \{0,1\}$

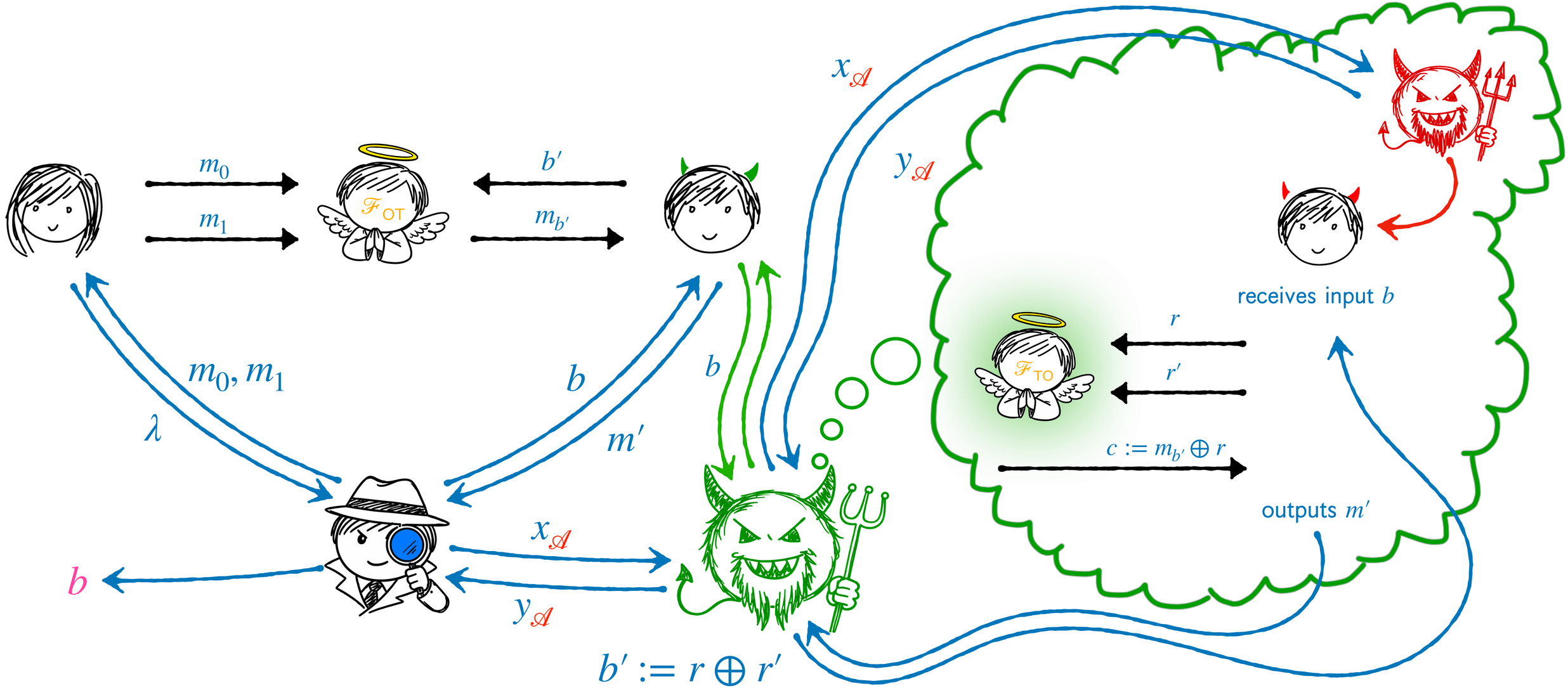



$c := m_0 \oplus \rho$

outputs $m_b := r \oplus c$

# Simulation for Bob

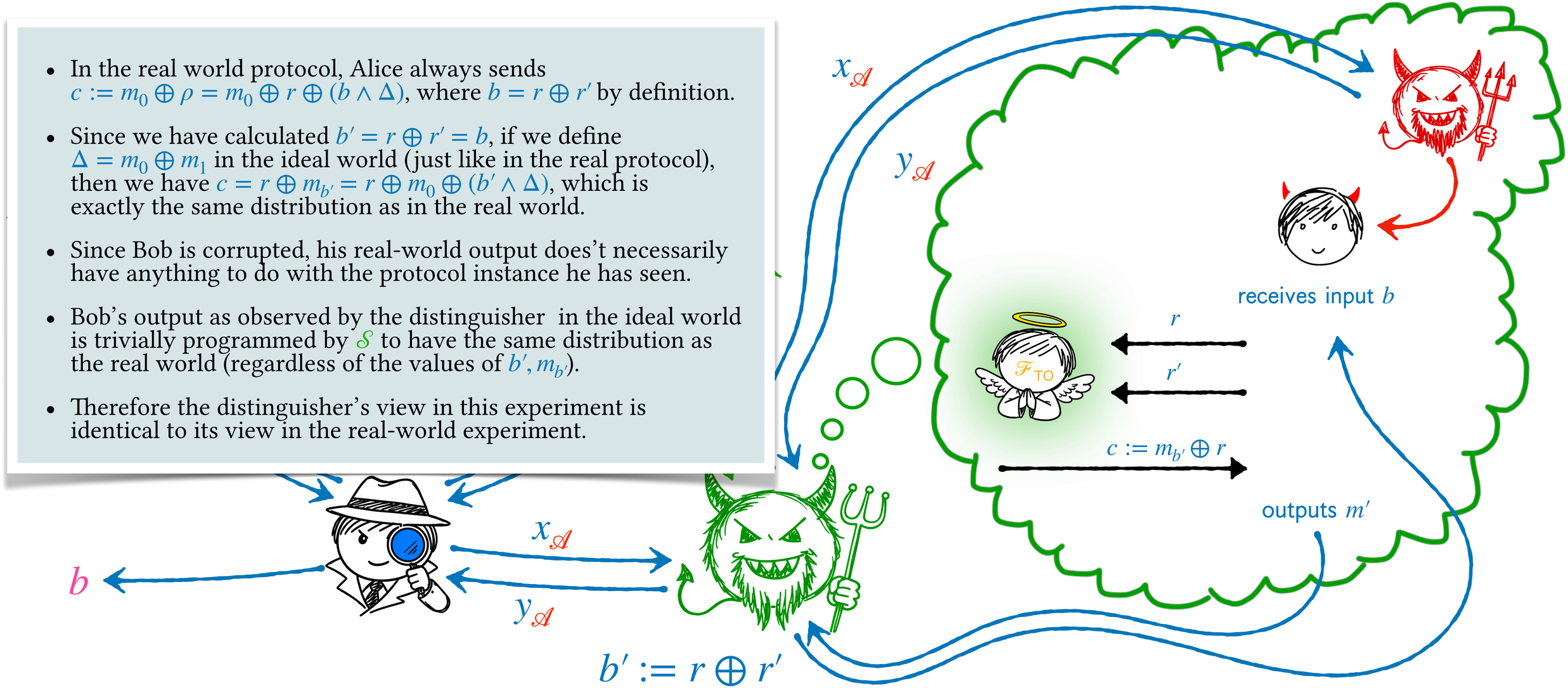
Suppose $\mathcal{A}$ maliciously corrupts Bob. We must construct $\mathcal{S}_B$ that produces an indistinguishable *experiment* (not just view) in the ideal world.



# Simulation for Bob

Suppose $\mathcal{A}$ maliciously corrupts Bob. We must construct $\mathcal{S}_B$ that produces an indistinguishable *experiment* (not just view) in the ideal world.

- In the real world protocol, Alice always sends $c := m_0 \oplus \rho = m_0 \oplus r \oplus (b \wedge \Delta)$, where $b = r \oplus r'$ by definition.
- Since we have calculated $b' = r \oplus r' = b$, if we define $\Delta = m_0 \oplus m_1$ in the ideal world (just like in the real protocol), then we have $c = r \oplus m_{b'} = r \oplus m_0 \oplus (b' \wedge \Delta)$, which is exactly the same distribution as in the real world.
- Since Bob is corrupted, his real-world output does't necessarily have anything to do with the protocol instance he has seen.
- Bob's output as observed by the distinguisher in the ideal world is trivially programmed by $\mathcal{S}$ to have the same distribution as the real world (regardless of the values of $b', m_{b'}$).
- Therefore the distinguisher's view in this experiment is identical to its view in the real-world experiment.



# Our first maliciously-secure protocol!

- We can combine $\mathcal{S}_A$ and $\mathcal{S}_B$ into a single distinguisher $\mathcal{S}$ that chooses its corruption in the ideal world dynamically based upon which party the emulated $\mathcal{A}$ corrupts.
- Notice that doing so gives us a PPT $\mathcal{S}$ that works for *any* PPT $\mathcal{A}$.
- Together with the two perfect simulation arguments we gave previously, this yields:

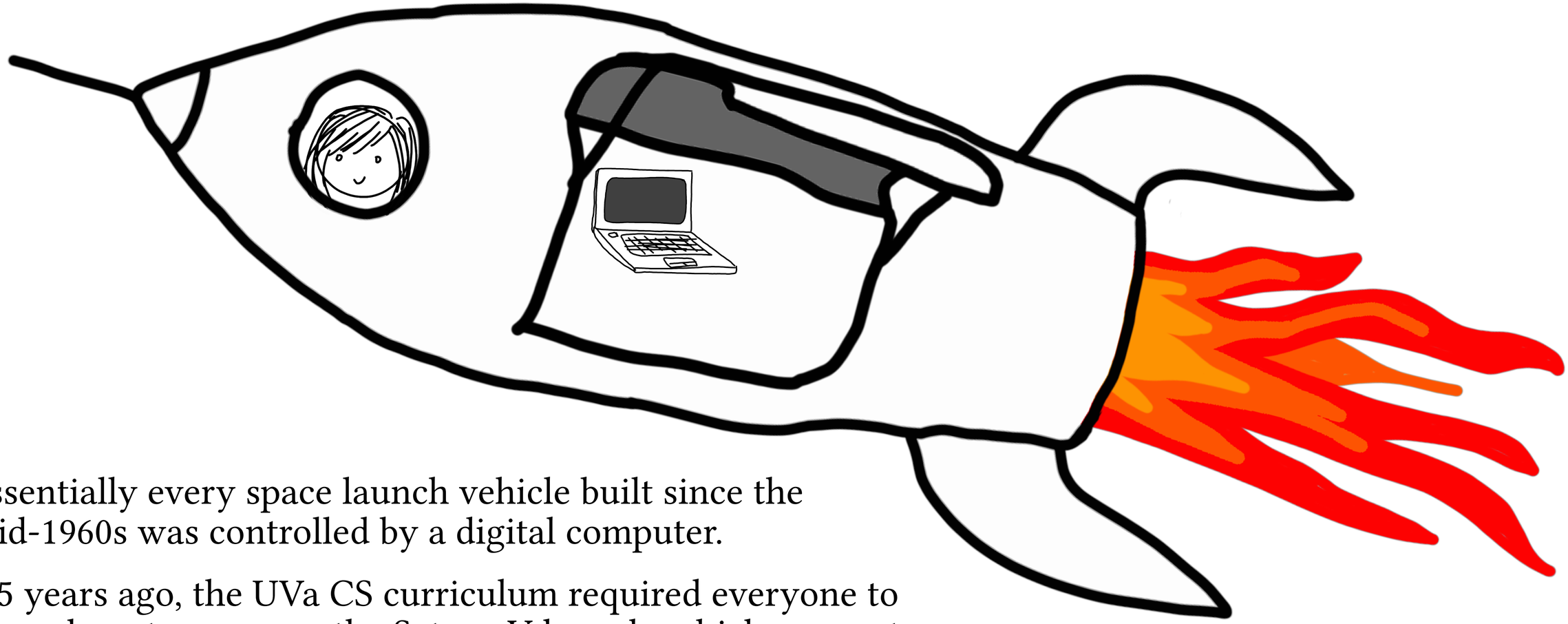
**Theorem 1:** there exists a one-message protocol in the $\mathcal{F}_{\mathsf{TO}}$-hyrid model that perfectly realizes $\mathcal{F}_{\mathsf{OT}}$ in the presence of a *malicious* adversary that statically corrupts one party.

One of my goals in this class is to convince you that Cryptographic Protocols solve practical problems.

Way back at the beginning, you learned that they can help you get a date!

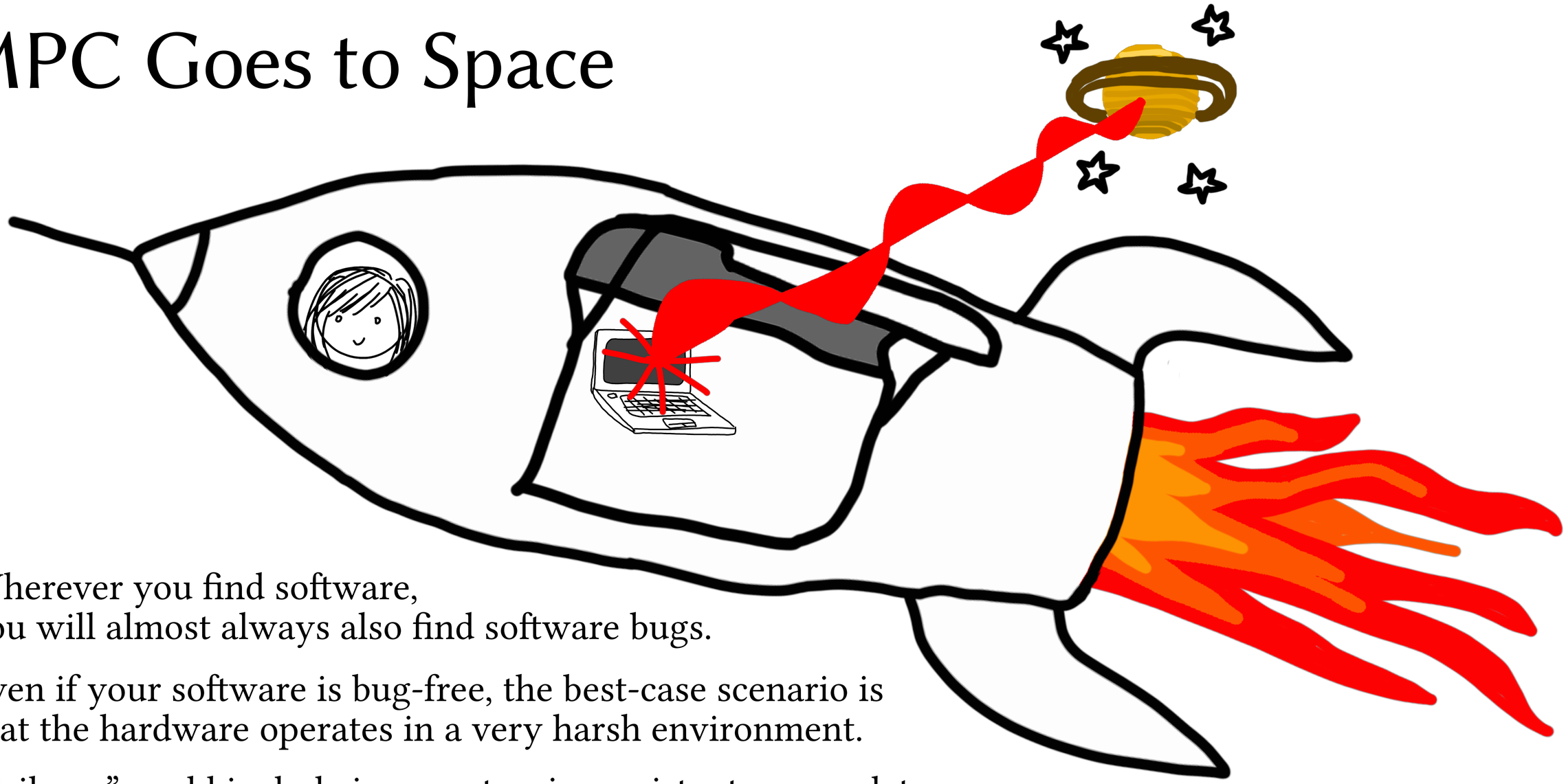
What else can we do with MPC?

# MPC Goes to Space



- Essentially every space launch vehicle built since the mid-1960s was controlled by a digital computer.
- (15 years ago, the UVa CS curriculum required everyone to learn how to program the Saturn V launch vehicle computer, because engineering school is where you learn practical skills)

# MPC Goes to Space



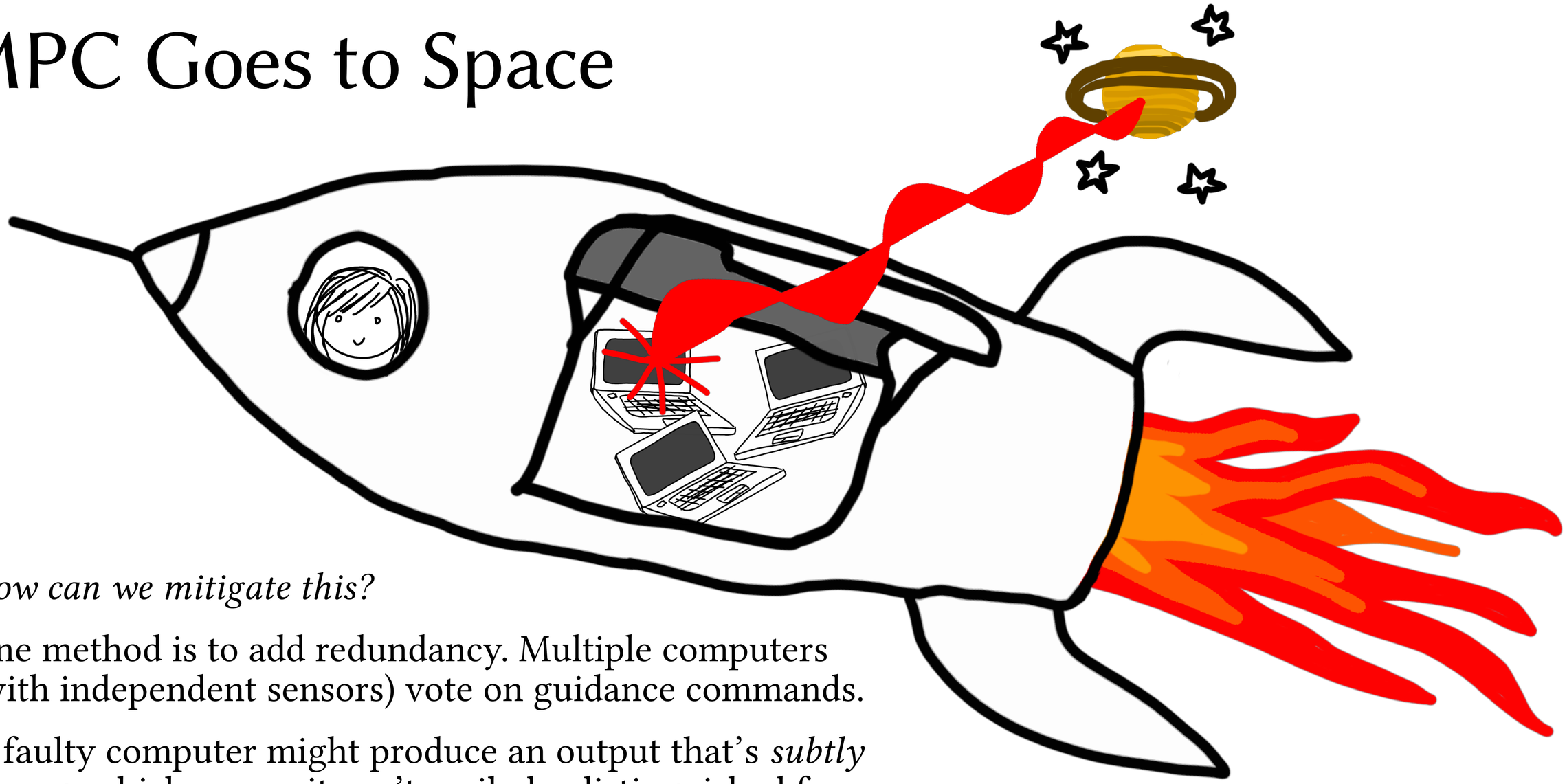
- Wherever you find software,
  you will almost always also find software bugs.
- Even if your software is bug-free, the best-case scenario is
  that the hardware operates in a very harsh environment.
- “Failures” could include incorrect or inconsistent sensor data,
  subtle errors due to cosmic rays causing bit flips in memory, or total crashes.

# MPC Goes to Space

**Pictured:** the Ariane V rocket encounters an integer overflow bug during its first launch.

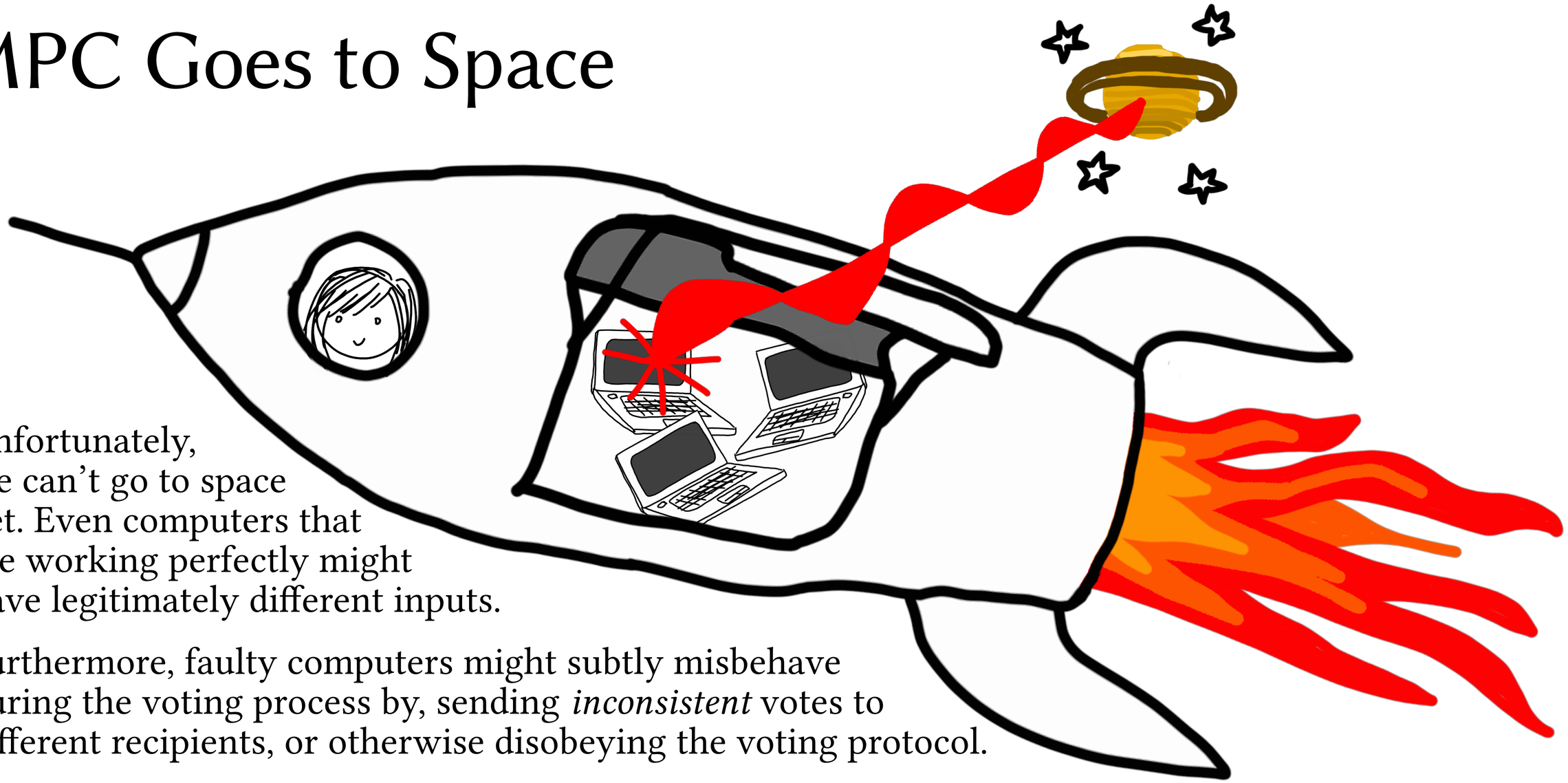
- Wherever you fi
  you will almost
- Even if your soft
  that the hardwa
- “Failures” could include incorrect or inconsistent sensor data, subtle errors due to cosmic rays causing bit flips in memory, or total crashes.

# MPC Goes to Space



- *How can we mitigate this?*
- One method is to add redundancy. Multiple computers (with independent sensors) vote on guidance commands.
- A faulty computer might produce an output that's *subtly* wrong, which means it can't easily be distinguished from a correct output. If we assume at most one computer fails, then we intuitively need at least three.

# MPC Goes to Space



- Unfortunately, we can't go to space yet. Even computers that are working perfectly might have legitimately different inputs.
- Furthermore, faulty computers might subtly misbehave during the voting process by, sending *inconsistent* votes to different recipients, or otherwise disobeying the voting protocol.
- No one computer can be trusted to count the votes. Nevertheless, all non-faulty computers must *agree* on a course of action. If all non-faulty computers arrive at the *same* course of action, based on their inputs, then that should be one they agree upon. *Can they?*

PROCEEDINGS OF THE IEEE, VOL. 66, NO. 10, OCTOBER 1978

1240

# SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control

WENSLEY, LESLIE LAMPORT, JACK GOLDBERG, SENIOR MEMBER, IEEE,
KARL N. LEVITT, P. M. MELLIAR-SMITH, ROBERT E. SHOSTAK,
CHARLES B. WEINSTOCK

from computer outputs. Com
reliability that is
frequently

# Reaching Agreement in the Presence of Faults

M. PEASE, R. SHOSTAK, AND L. LAMPORT

*SRI International, Menlo Park, California*

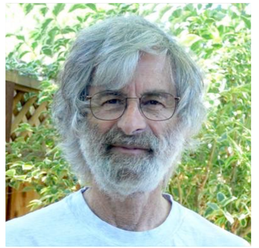
# The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound

the problem is solvable for any number of

"When it became clear that computers were going to be flying commercial aircraft, NASA began funding research to figure out how to make them reliable enough for the task. Part of that effort was the SIFT project at SRI. This project was perhaps most notable for producing the *Byzantine generals problem* and its solutions"
—Leslie Lamport



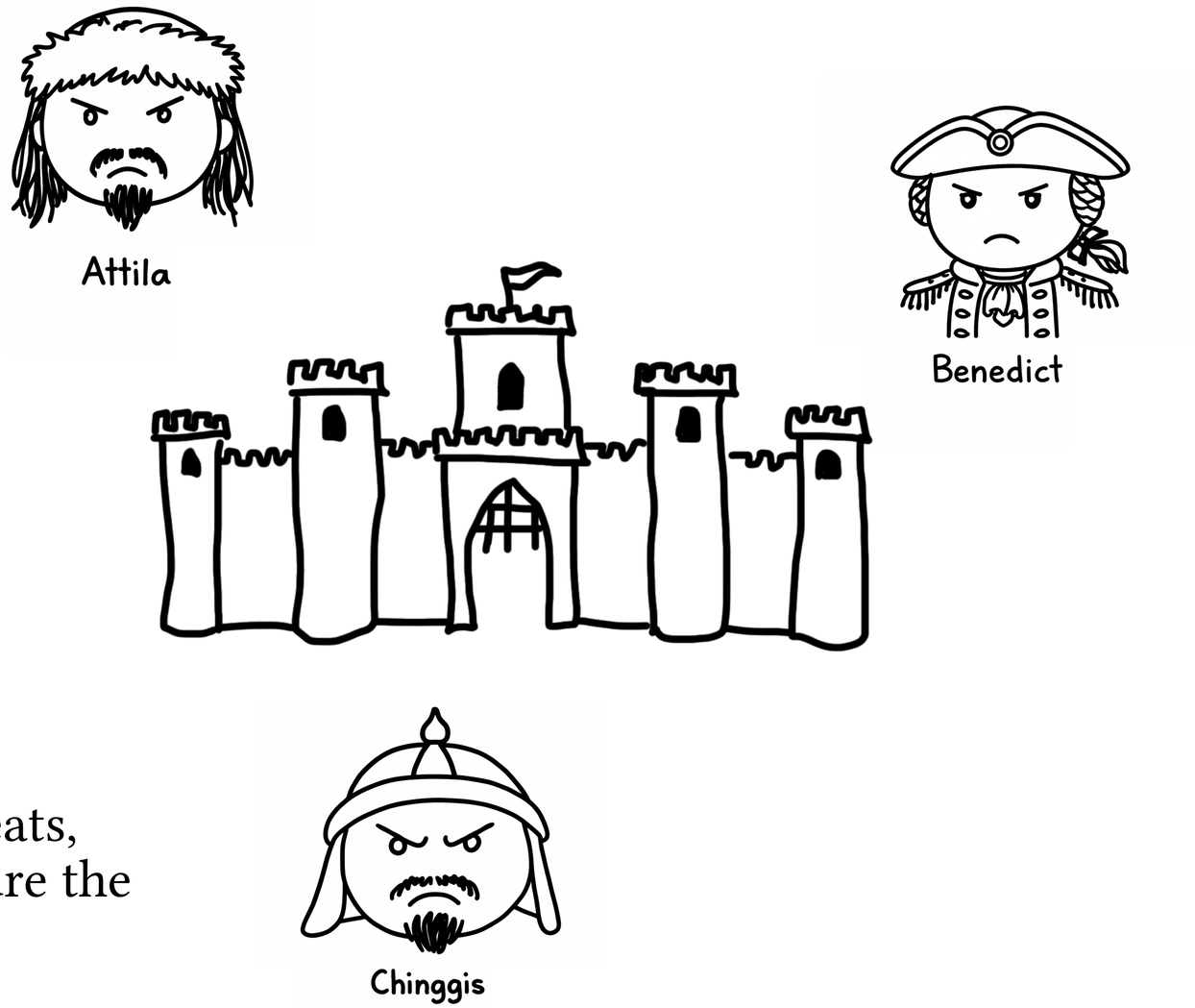
(I couldn't find a photo of Marshall Pease, so please accept this illustration from his short science fiction story "Generals Help Themselves")


But wait, what does fault-tolerant computation have to do with military strategy in the eastern roman empire?

Sidenote: SRI later built the Ariane V computer... oops!

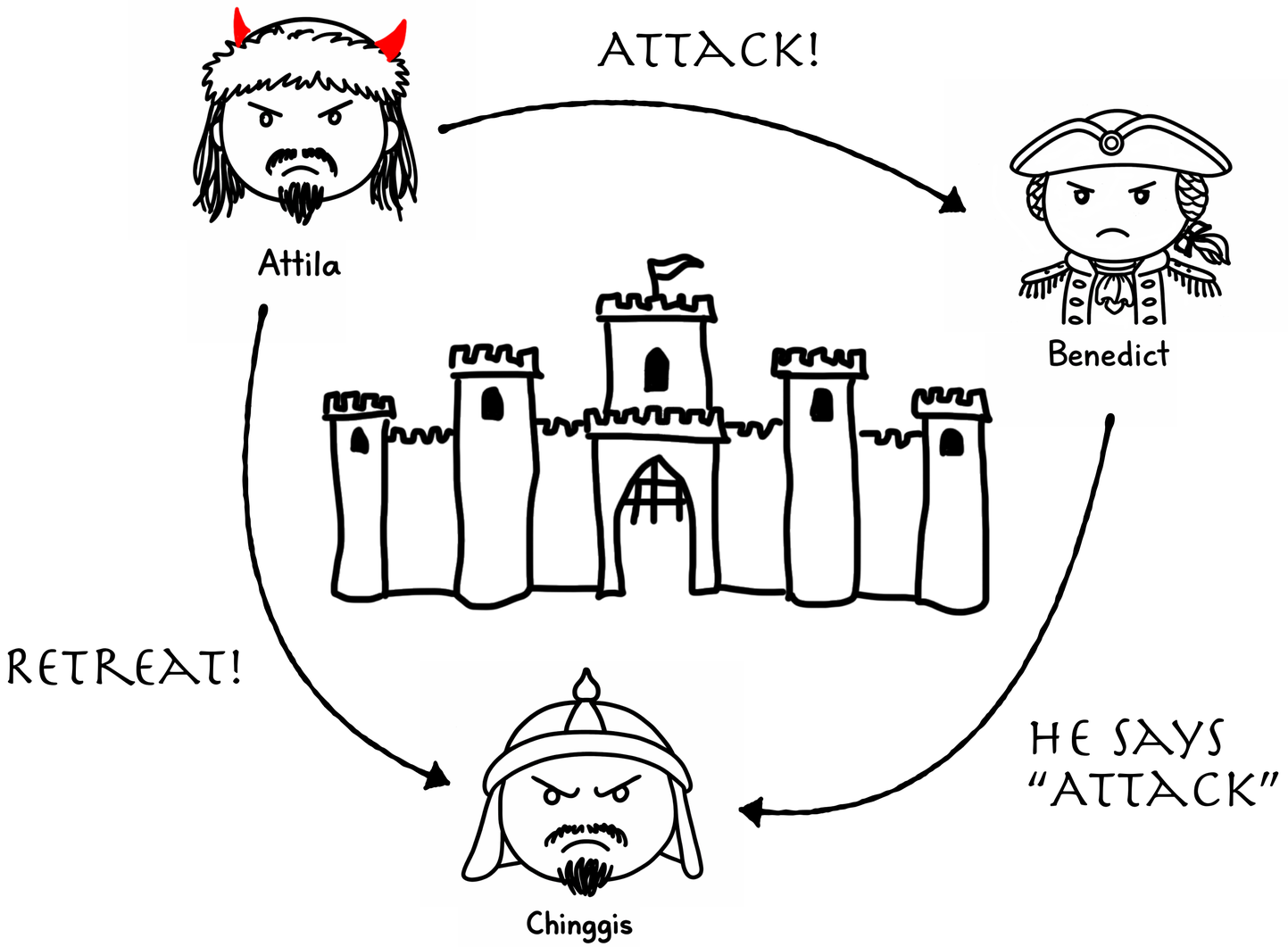
# MPC Lays Siege

- Three generals have encamped their armies atop three hills outside a city.
- They have agreed in advance that Attila will decide whether they should *attack* or *retreat*.
- The three must use (trusted) messengers in order to learn Attila's decision.
- One of the three might be a *traitor*, but none of them are sure who.
- If the two *loyal* generals both attack, they will conquer the city. If the two loyal generals both retreat, they will survive to fight another day.
- If one loyal general attacks, and the other retreats, then they will be defeated. How can they be sure the traitor won't cause this to happen?



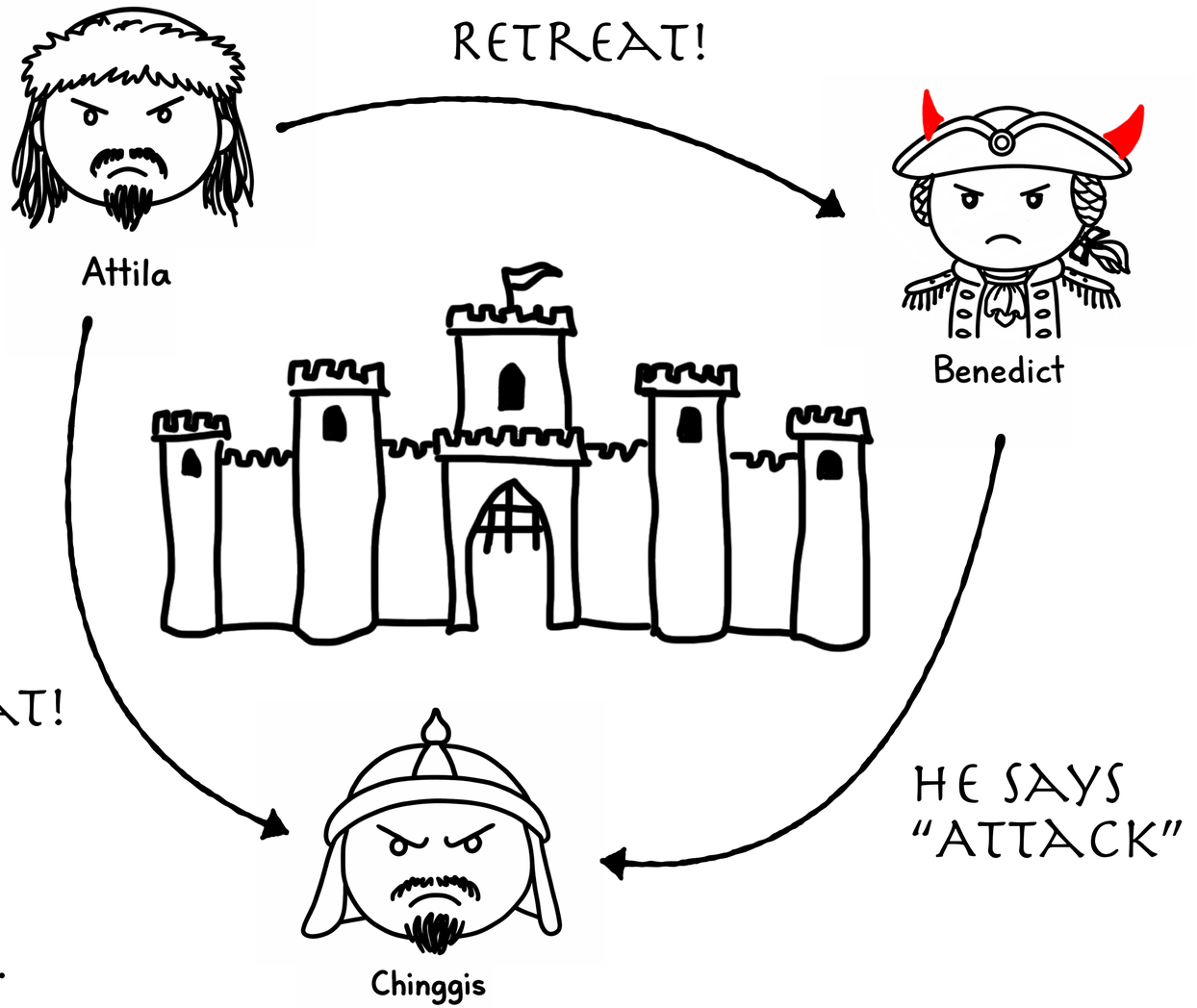
# MPC Lays Siege

- In particular, if Attila is a traitor, he might instruct the others to do different things.
- They might be able to detect this by comparing the messages they receive from him.



# MPC Lays Siege

- In particular, if Attila is a traitor, he might instruct the others to do different things.
- They might be able to detect this by comparing the messages they receive from him.
- But if Benedict is the traitor in this simple protocol, he might try to frame Attila in order to confuse Chinggis.
- Is there a more advanced protocol that allows them to overcome this problem?
- This scenario is distinguished from the first one because *only* Attila has an input, and they all want to agree on what *his* input is.



Now we have two different problems that seem very similar.

Sometimes they are equivalent (via reduction) and sometimes not.

Both will help us to understand what can be achieved against malicious adversaries.

# Byzantine Agreement

**Definition 4.** Let $n, t \in \mathbb{N}$ such that $t < n$, and let $P_1, \ldots, P_n$ be a group of parties that are connected by synchronous, authenticated channels. Let $\pi$ be a protocol in which each $P_i$ has an input bit $x_i \in \{0,1\}$ and an output bit $y_i \in \{0,1\}$. We say that $\pi$ is a $t$-secure *Byzantine Agreement* protocol if the following conditions hold with all-but-negligible probability when $\mathcal{A}$ maliciously corrupts up to $t$ parties:

1. **Consistency:** All honest parties output the same bit $y$.

2. **Validity:** If all honest parties start with the same input $x$, then $y = x$.

Think back to the beginning of the class, when we introduced the idea of *simulation-based* security definitions, we said it was also possible to define the security of a protocol by enumerating specific properties. This is what we have done above!

Notice that we specified malicious corruptions. *What about honest corruptions?*

# Byzantine Agreement

**Definition 4.** Let $n, t \in \mathbb{N}$ such that $t < n$, and let $P_1, \ldots, P_n$ be a group of parties that are connected by synchronous, authenticated channels. Let $\pi$ be a protocol in which each $P_i$ has an input bit $x_i \in \{0,1\}$ and an output bit $y_i \in \{0,1\}$. We say that $\pi$ is a $t$-secure *Byzantine Agreement* protocol if the following conditions hold with all-but-negligible probability when $\mathcal{A}$ maliciously corrupts up to $t$ parties:

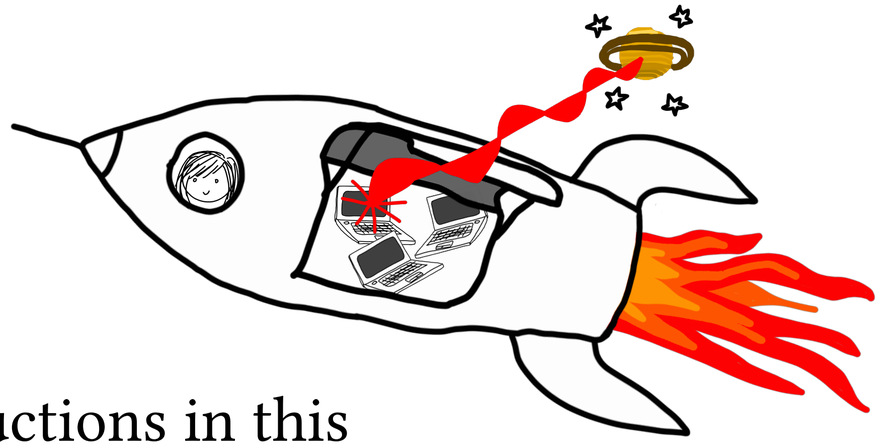
1. **Consistency:** All honest parties output the same bit $y$.

2. **Validity:** If all honest parties start with the same input $x$, then $y = x$.

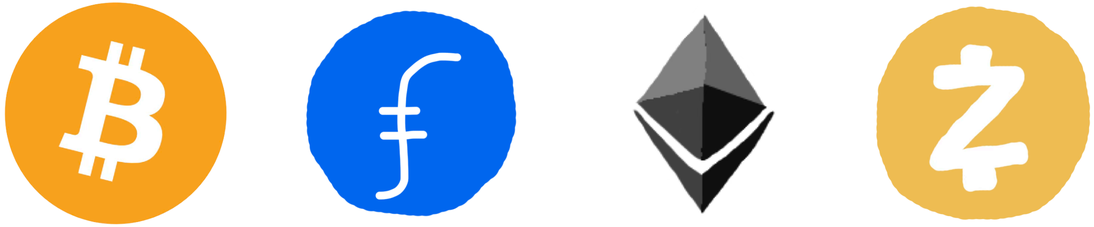
This definition is *weaker* than a simulation-based definition, so when we prove it is impossible to achieve, we have proven a *stronger* bound.

In particular, ruling this out rules out any protocol that realizes any functionality that has the above properties. *Can you think of such a functionality?*

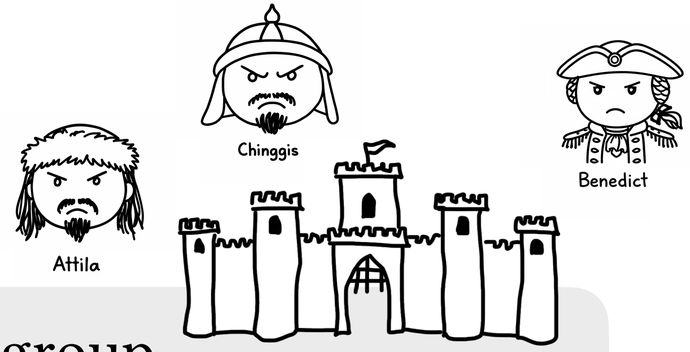
# Byzantine Agreement



- We won't use BA protocols directly in any of our constructions in this class, but we will use the concept of BA to rule out that malicious security is possible for certain functionalities in certain parameter regimes.
- This doesn't mean BA isn't directly useful though!
- Byzantine Agreement is also known as *consensus*. This is the most basic security property that Blockchain protocols must achieve. Many modern advancements in the consensus literature are motivated by (or derive techniques from) blockchains.



# Byzantine Generals (Broadcast)




**Definition 5.** Let $n, t \in \mathbb{N}$ such that $t < n$, and let $P_1, \ldots, P_n$ be a group of parties that are connected by synchronous, authenticated channels. Let $\pi$ be a protocol in which $P_1$ has an input $x \in \{0,1\}$ and every $P_i$ produces an output $y_i \in \{0,1\}$. We say that $\pi$ is a $t$-secure *Broadcast* protocol if the following conditions hold with all-but-negligible probability when $\mathcal{A}$ maliciously corrupts up to $t$ parties:

1. **Consistency:** All honest parties output the same bit $y$.

2. **Validity:** If $P_1$ is honest, then $y = x$.

As before, this is a *property-based definition*. A broadcast *functionality* simply takes an input from one party and outputs it to all others. In other words, a simulation-secure broadcast protocol is one that securely computes $f(m, \lambda, \ldots, \lambda) = (m, m, \ldots, m)$.

*Do you agree that the broadcast functionality satisfies the properties?*

# Roadmap for the Next Two-ish Lectures

**Theorem 2:** In the setting where $t < n/2$, there exists a $t$-secure BA protocol if an only if there exists a $t$-secure broadcast protocol.

**Theorem 3:** When $t \geq n/3$, there does not exist a $t$-secure BA protocol in the plain model, even assuming authenticated channels.

**Theorem 4:** When $t < n/3$, there exists a perfectly $t$-secure BA protocol in the plain model, assuming authenticated channels.

**Theorem 5.** Assuming the existence of one-way functions and a *public-key infrastructure*, there exists a (computationally) $t$-secure broadcast protocol in the setting where $t < n$.

**First, a question:** *why can't a $t$-secure BA protocol exist when $t \geq n/2$?*

# BA $\Longrightarrow$ Broadcast

To prove Theorem 2, we will consider each direction of the implication individually.

**Lemma 1:** Let $t < n$. If there exists $t$-secure BA, then there exists $t$-secure broadcast.

**Proof:** Consider the following putative broadcast protocol:

1. $P_1$ sends $x$ to all parties. Let $x_i$ denote the alleged value of $x$ actually received by $P_i$, and let $x_i = 0$ if no message was received by $P_i$.

2. The parties $P_1, \ldots, P_n$ run BA on inputs $(x_1, \ldots, x_n)$.

**Consistency of Broadcast:** follows trivially from consistency of the BA protocol!

**Validity of Broadcast:** if $P_1$ is honest, then every $P_i$ receives $x_i = x$; given this fact, validity of broadcast follows from validity of the BA protocol. ∎

# Broadcast $\Longrightarrow$ BA

**Lemma 2:** Let $t < n/2$. If there exists $t$-secure broadcast, then there exists $t$-secure BA

**Proof:** Consider the following putative BA protocol:

1. Every $P_i$ broadcasts $x_i$ to all parties. Let $\hat{x}_i$ denote the alleged value of $x_i$ actually received by $P_j$ for all $j \in [n] \backslash \{i\}$, and let $\hat{x}_i = 0$ if $P_i$ fails to broadcast at all.

2. The parties $P_1, \ldots, P_n$ output the *majority* function on $(\hat{x}_1, \ldots \hat{x}_n)$. If there is no majority, then they output 0.

**Consistency of BA:** By the consistency of the broadcast protocol, all honest parties agree on $(\hat{x}_1, \ldots \hat{x}_n)$, and the majority function is deterministic.

**Validity of Broadcast:** Suppose there is some $x$ such that every honest $P_i$ has $x_i = x$. Since $t < n/2$, the majority of $(\hat{x}_1, \ldots \hat{x}_n)$ must be $x$. ∎

# CS4501 Cryptographic Protocols
# Lecture 19: OT is Symmetric
# Byzantine Agreement, Broadcast

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