

# Separating Broadcast *from* Cheater Identification: The ECDSA Case

Yashvanth Kondi

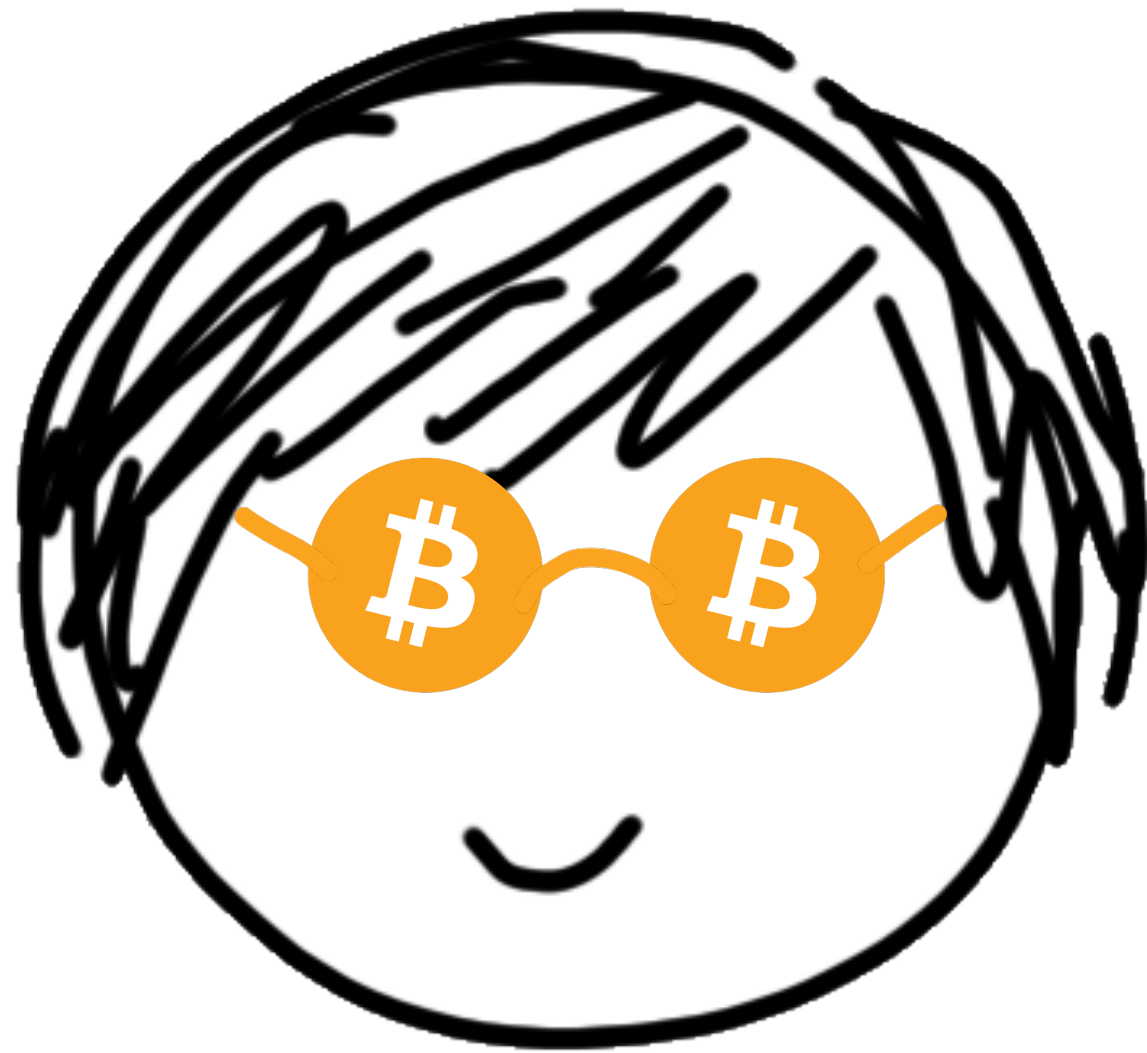
**S:LENCE**  
LABORATORIES

Divya Ravi

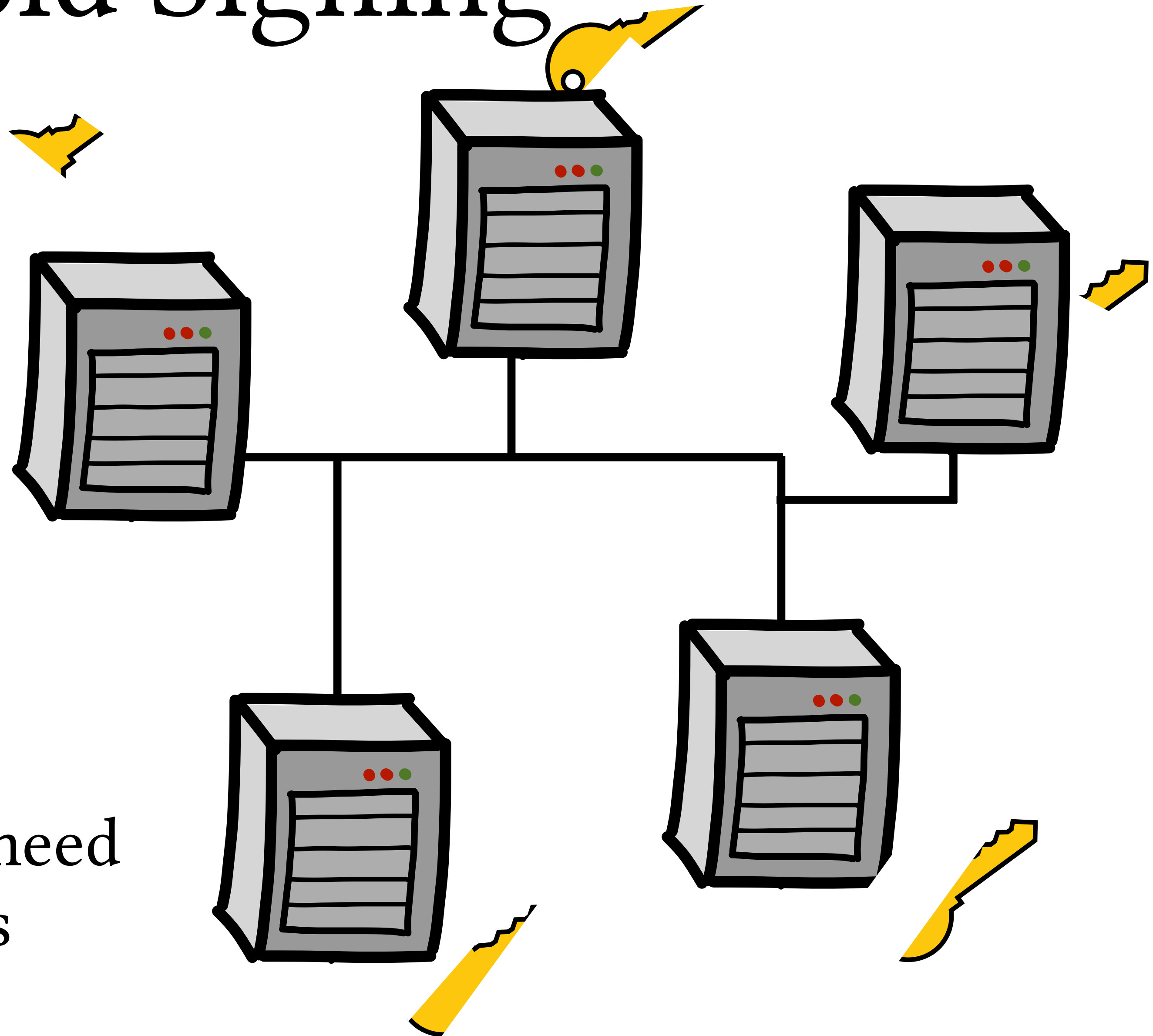


UNIVERSITY  
OF AMSTERDAM

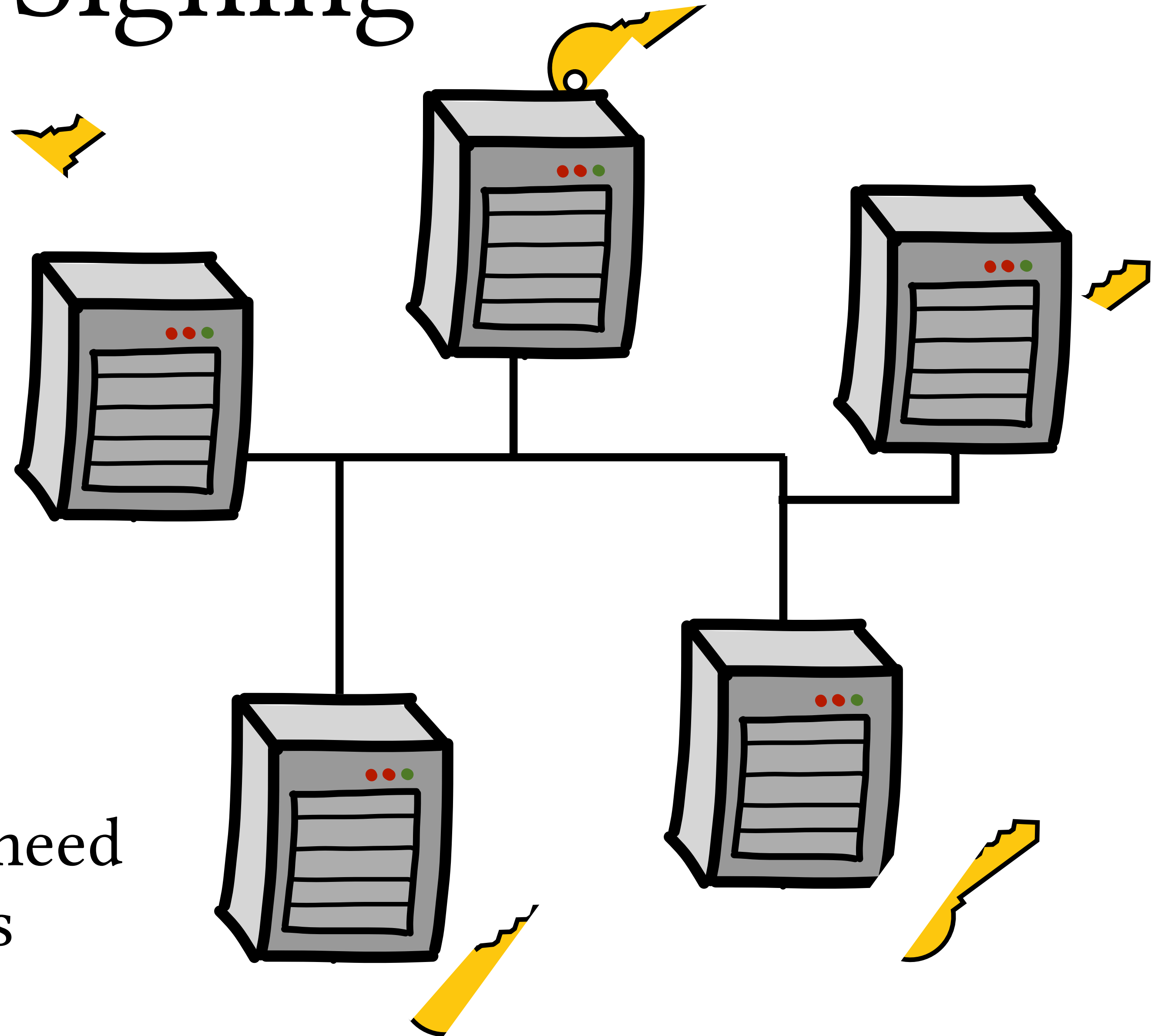
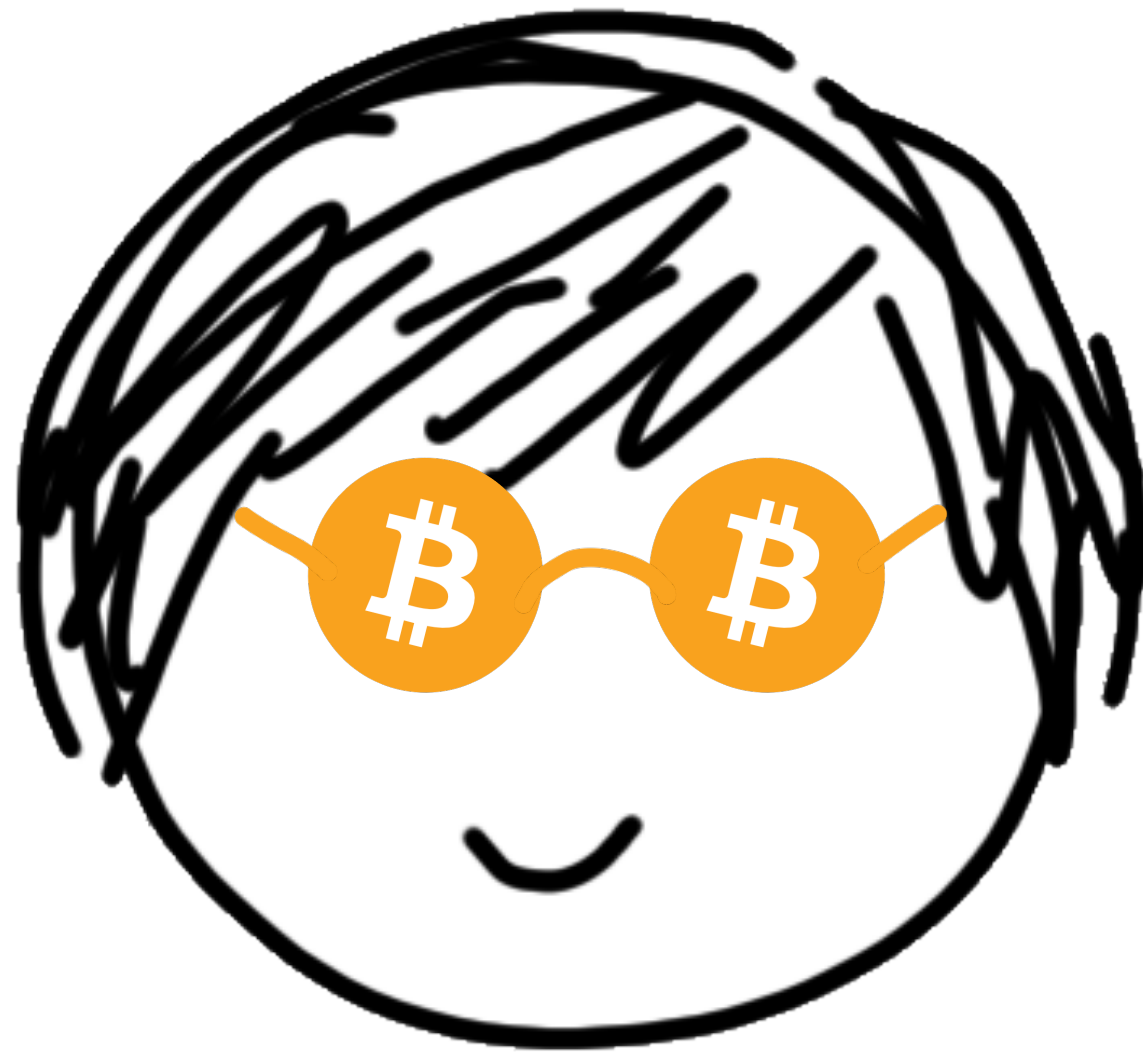
# Threshold Signing



**Distributed Risk:** Attacker will need to compromise multiple nodes

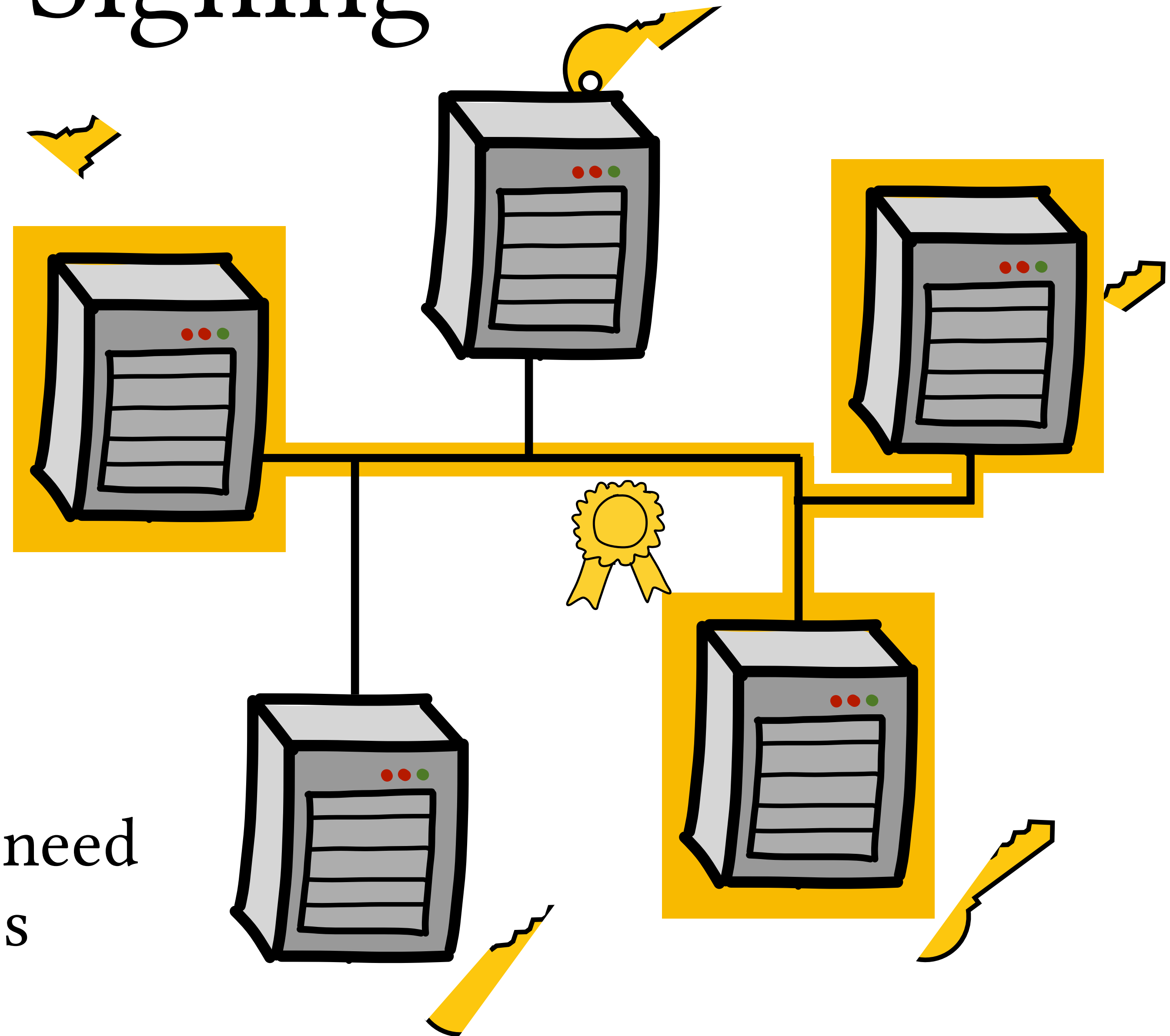
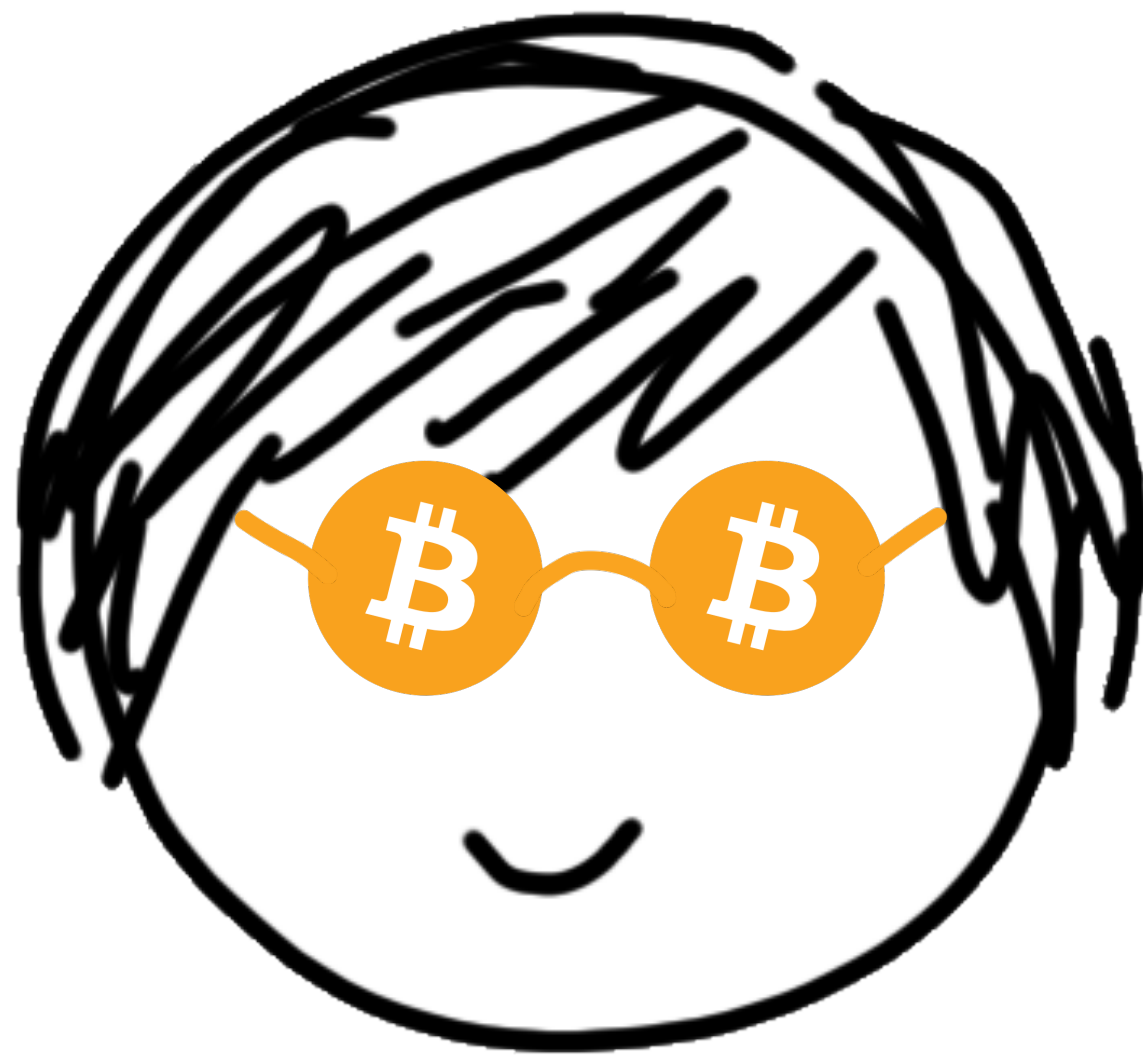


# $(3,n)$ Signing



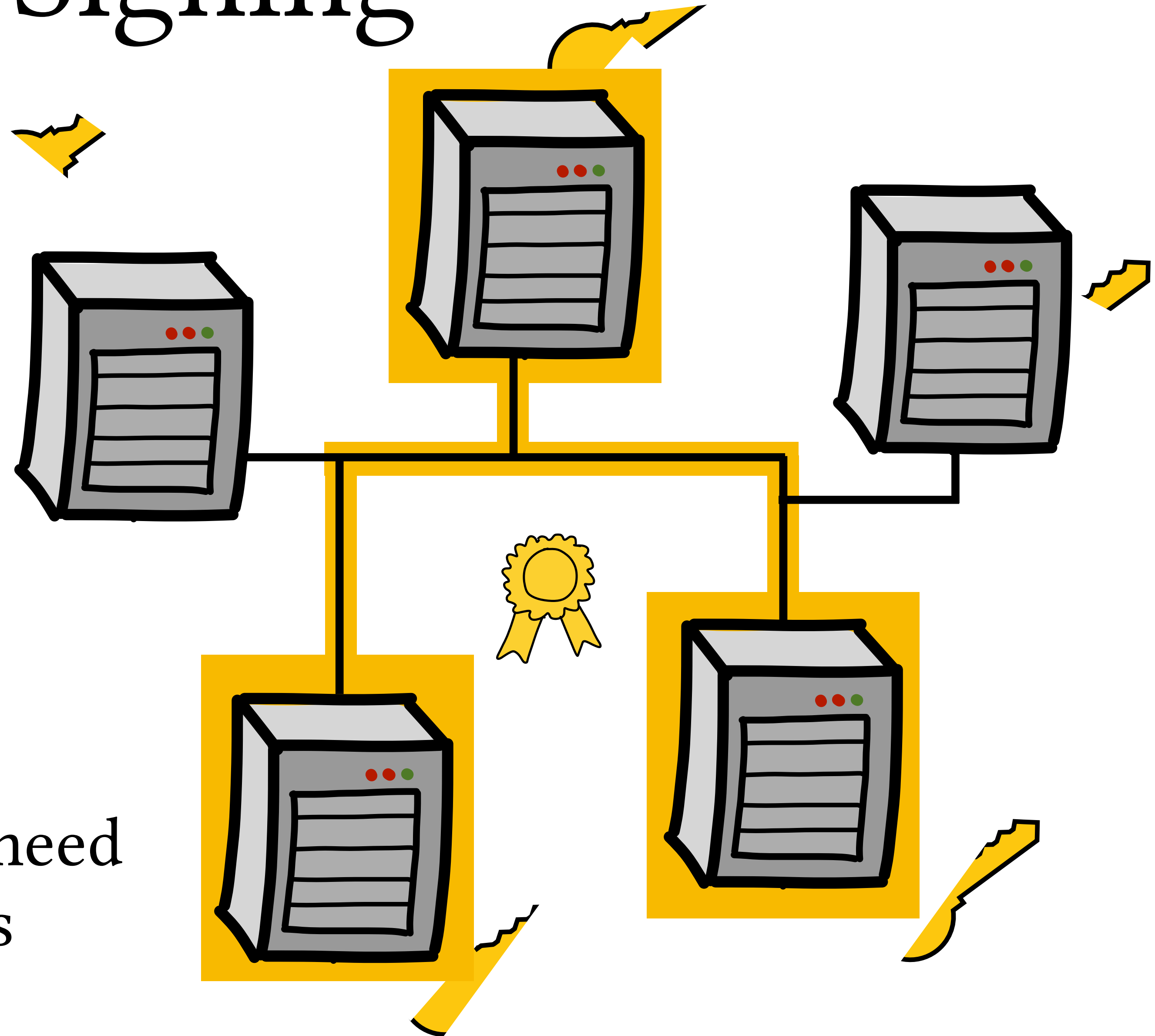
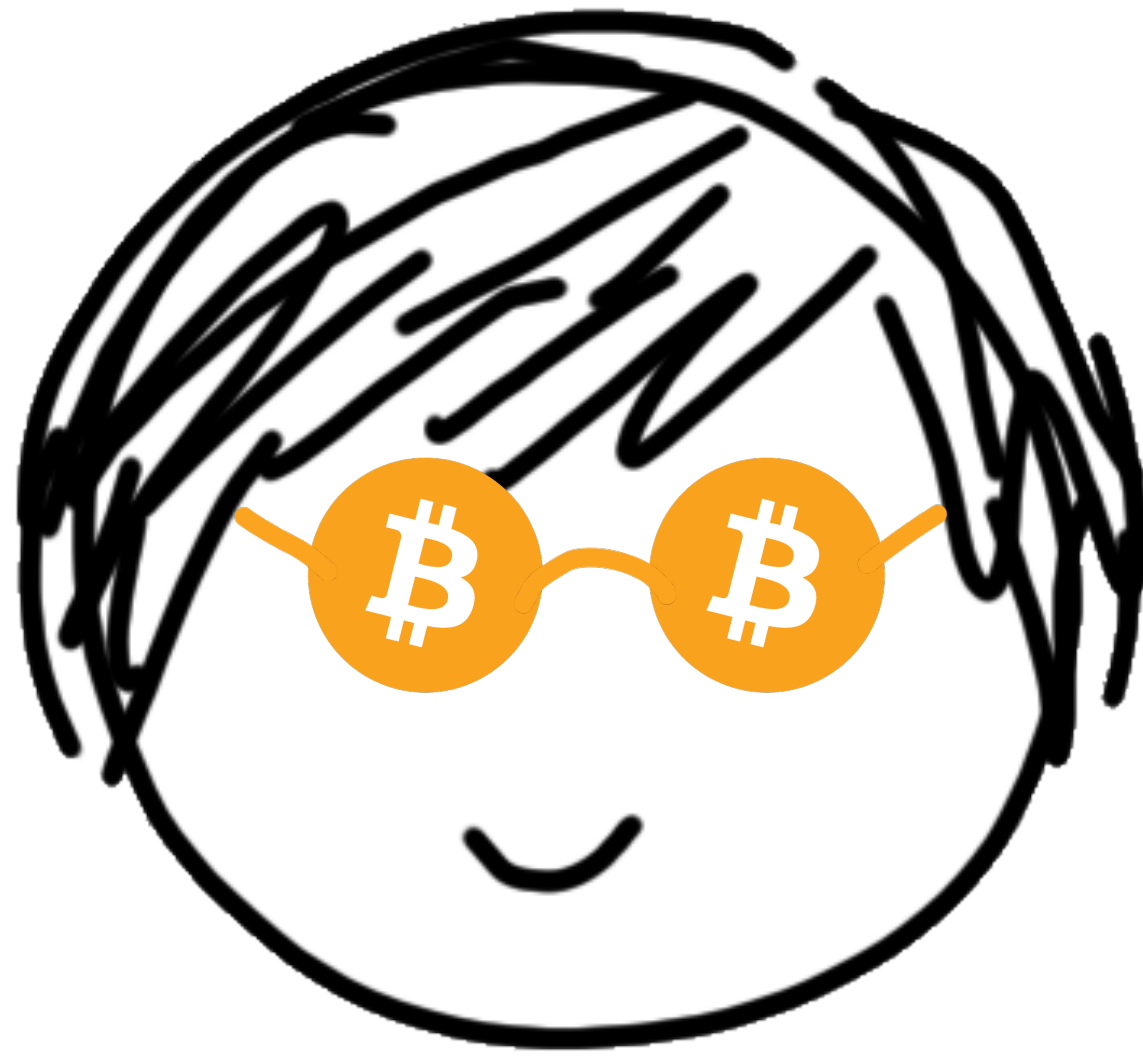
**Distributed Risk:** Attacker will need  
to compromise multiple nodes

# $(3,n)$ Signing



**Distributed Risk:** Attacker will need to compromise multiple nodes

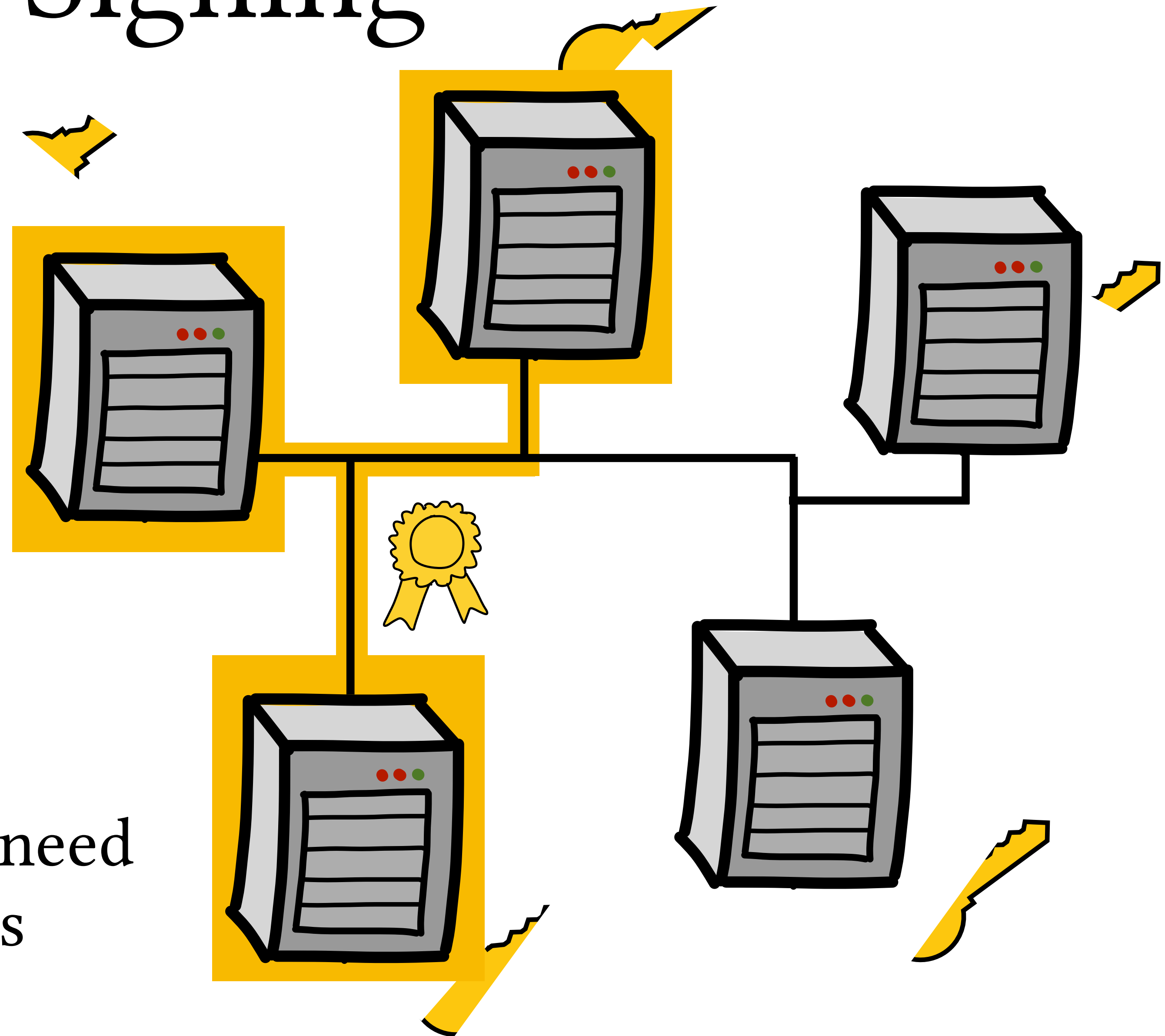
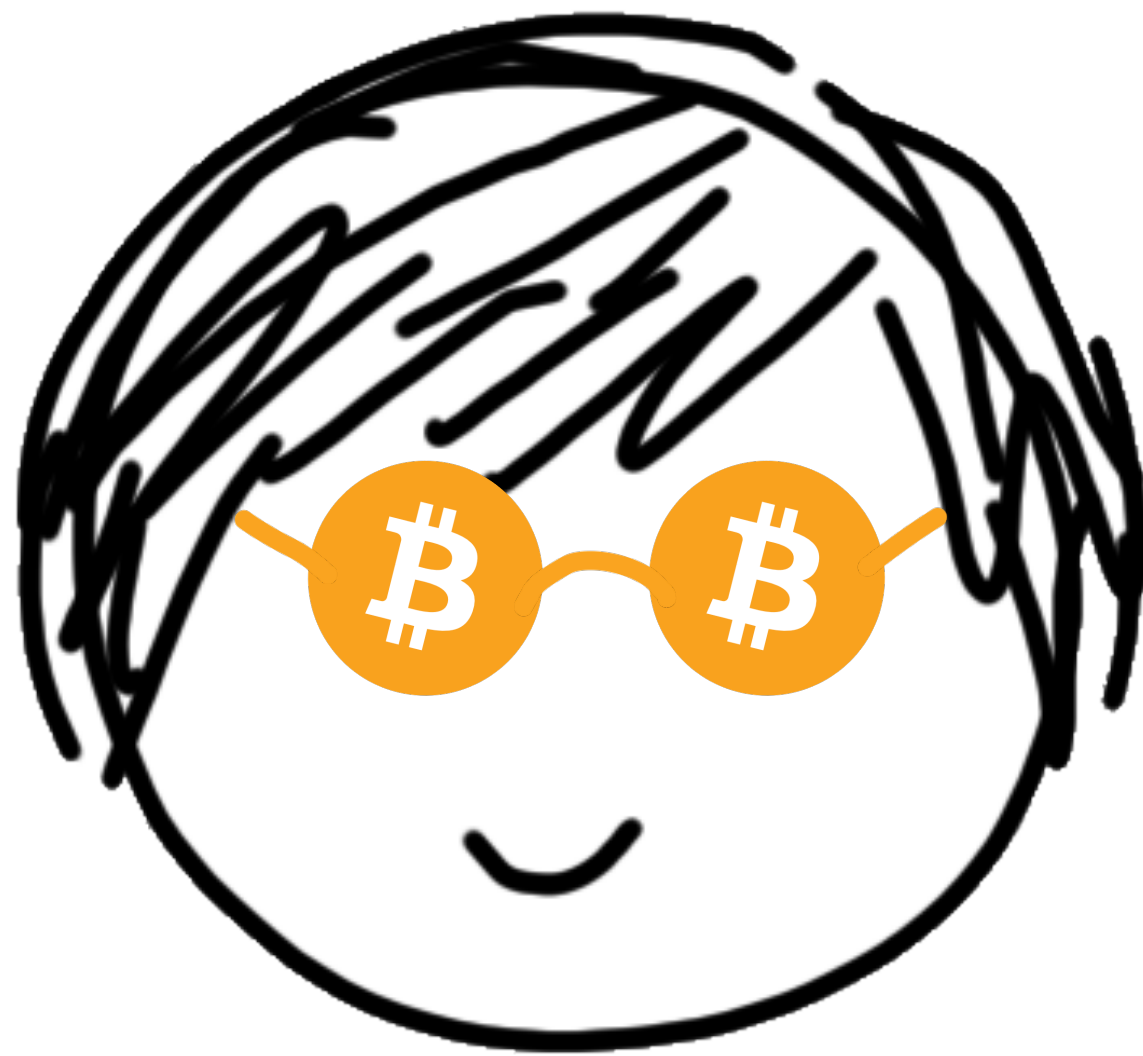
# $(3,n)$ Signing



**Distributed Risk:** Attacker will need to compromise multiple nodes

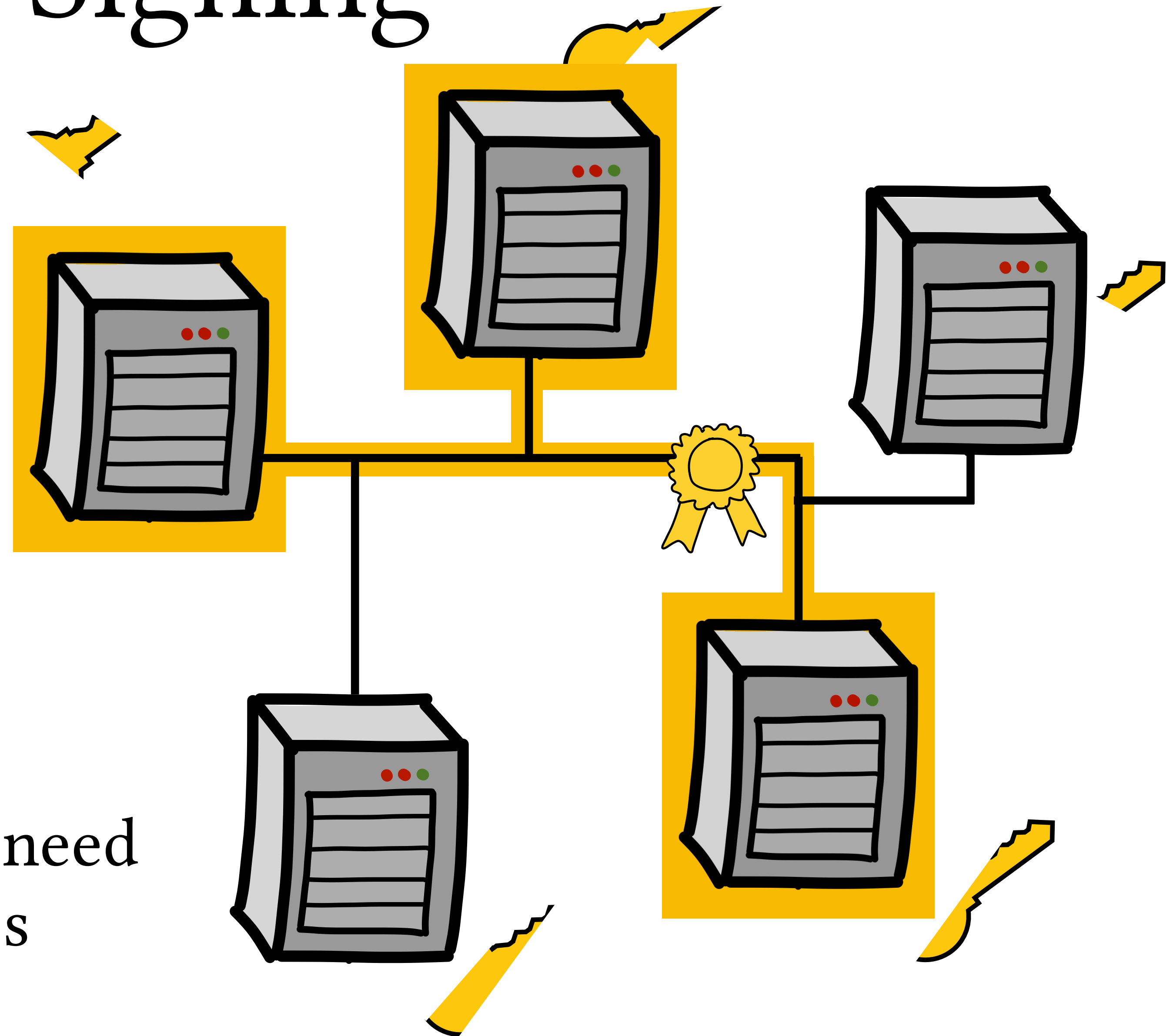
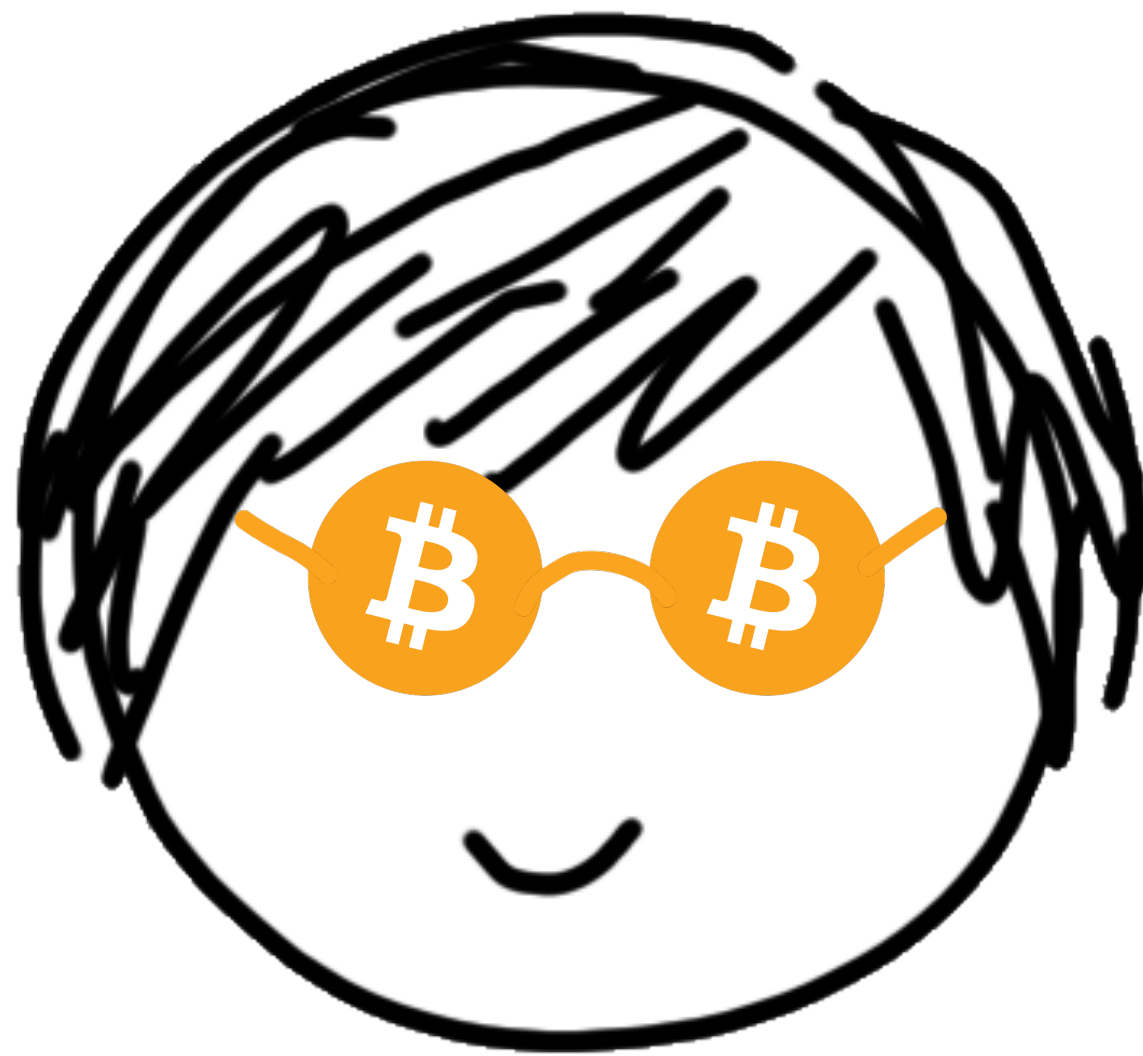


# $(3,n)$ Signing



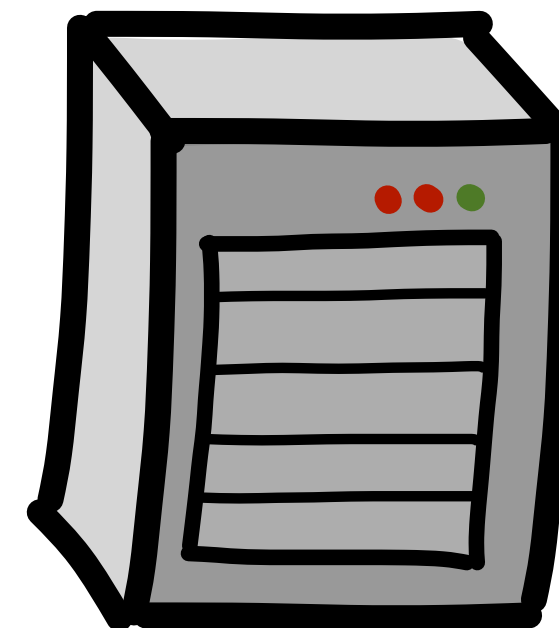
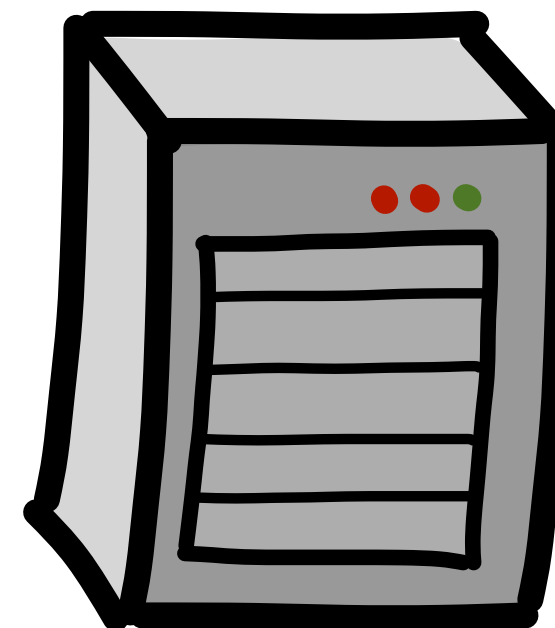
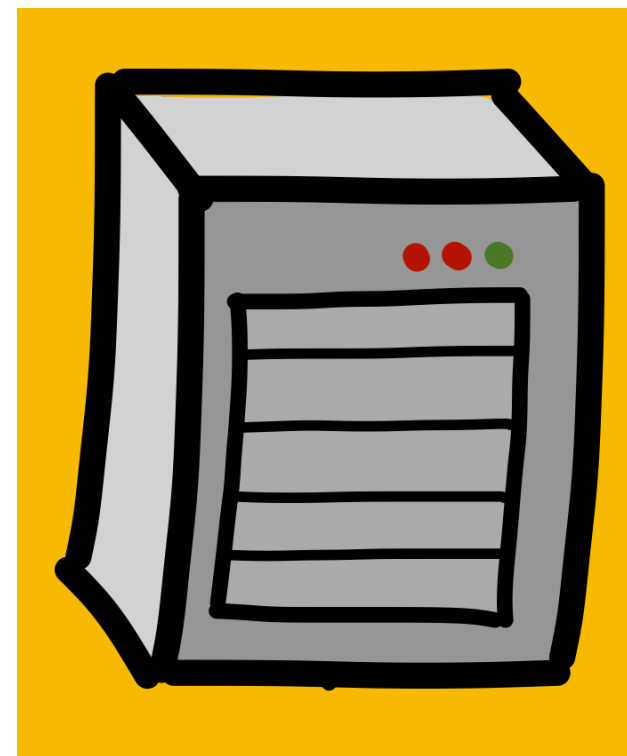
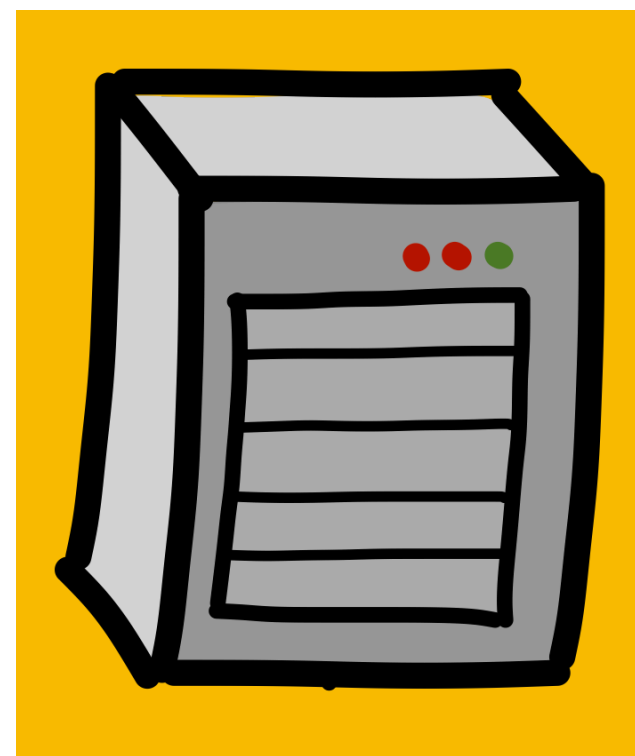
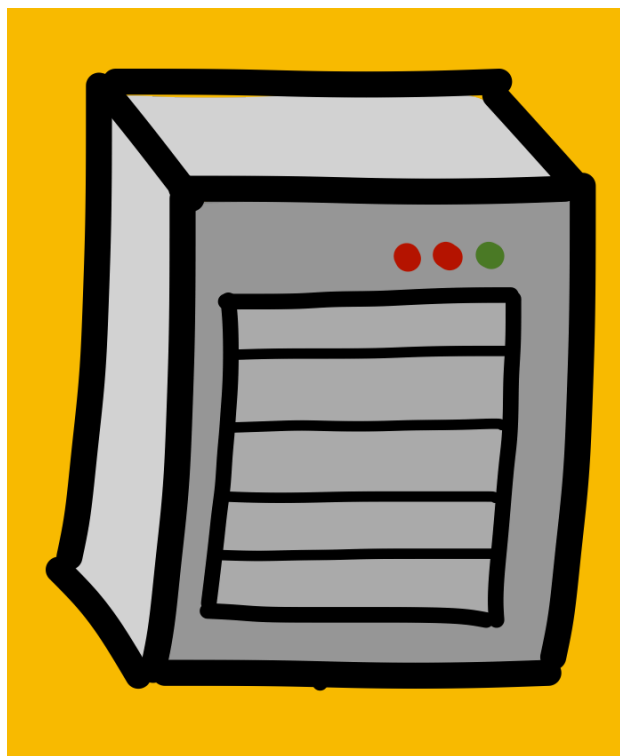
**Distributed Risk:** Attacker will need to compromise multiple nodes

# $(3,n)$ Signing



**Distributed Risk:** Attacker will need to compromise multiple nodes

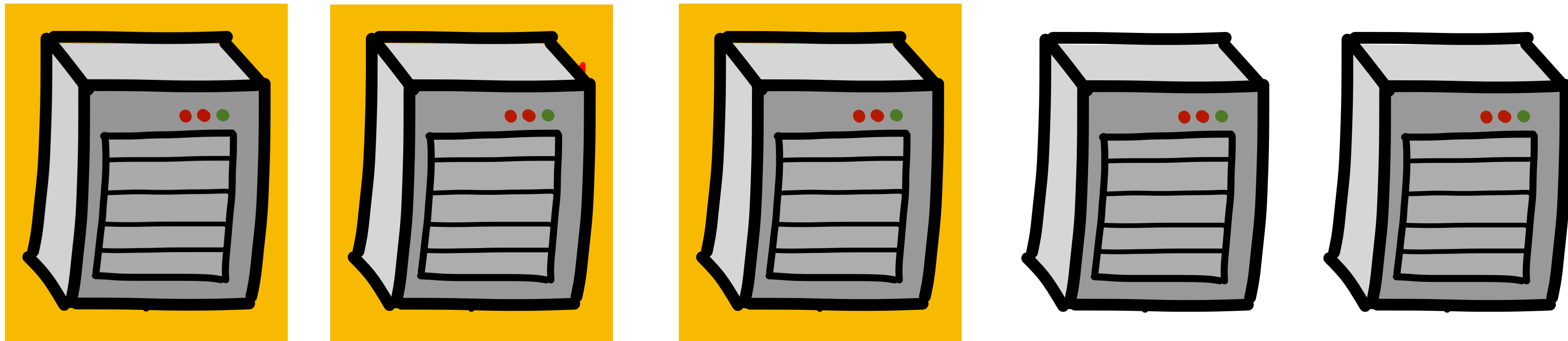
# $(3,n)$ Signing





# $(3, n)$ Signing

**Best Possible Security:** Protection against 2 corruptions



# $(3,n)$ Signing

**Best Possible Security:** Protection against 2 corruptions



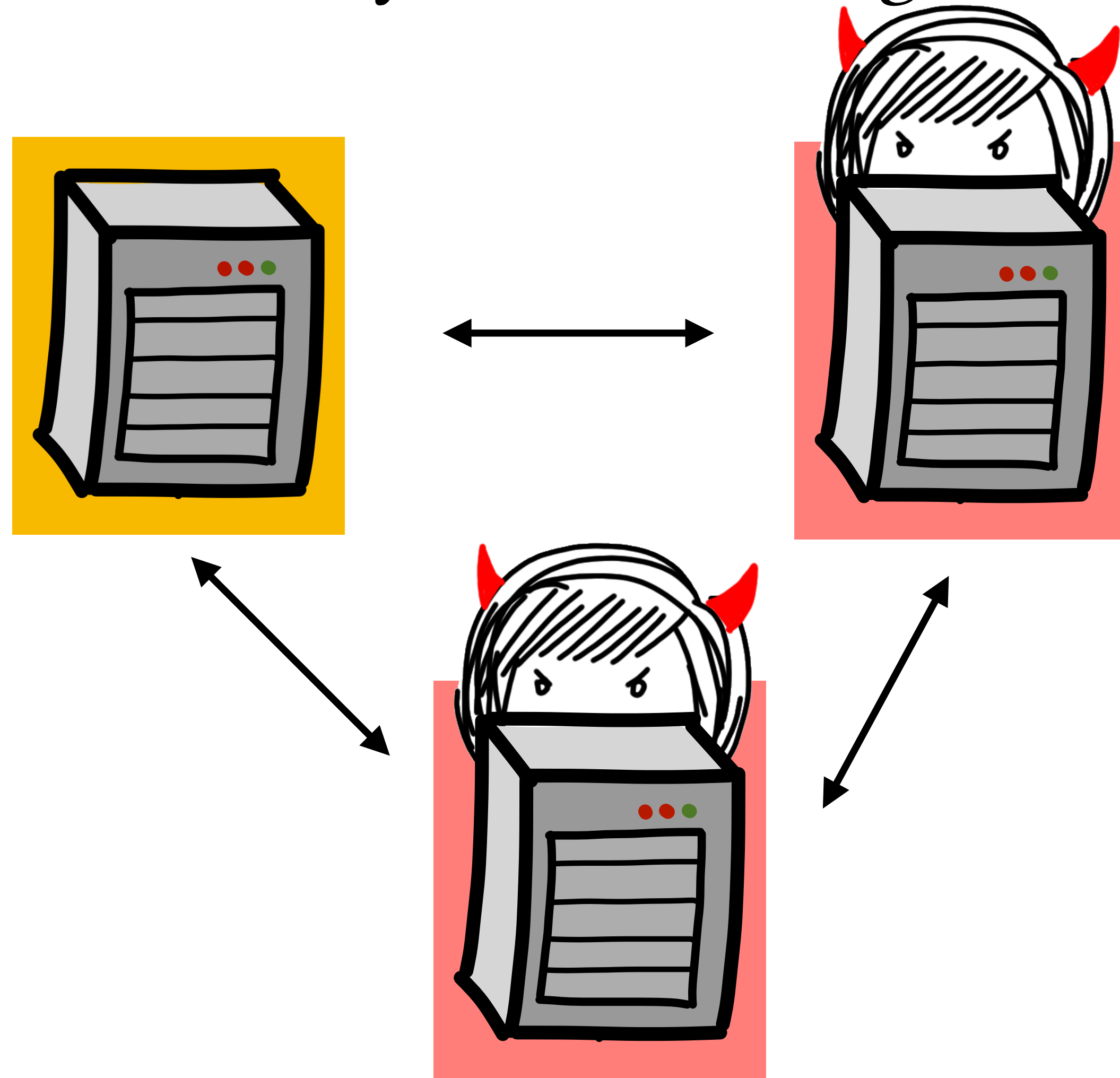
...out of five parties

“Global” honest majority

Necessary to retrieve  in case of a fault

# $(3,n)$ Signing

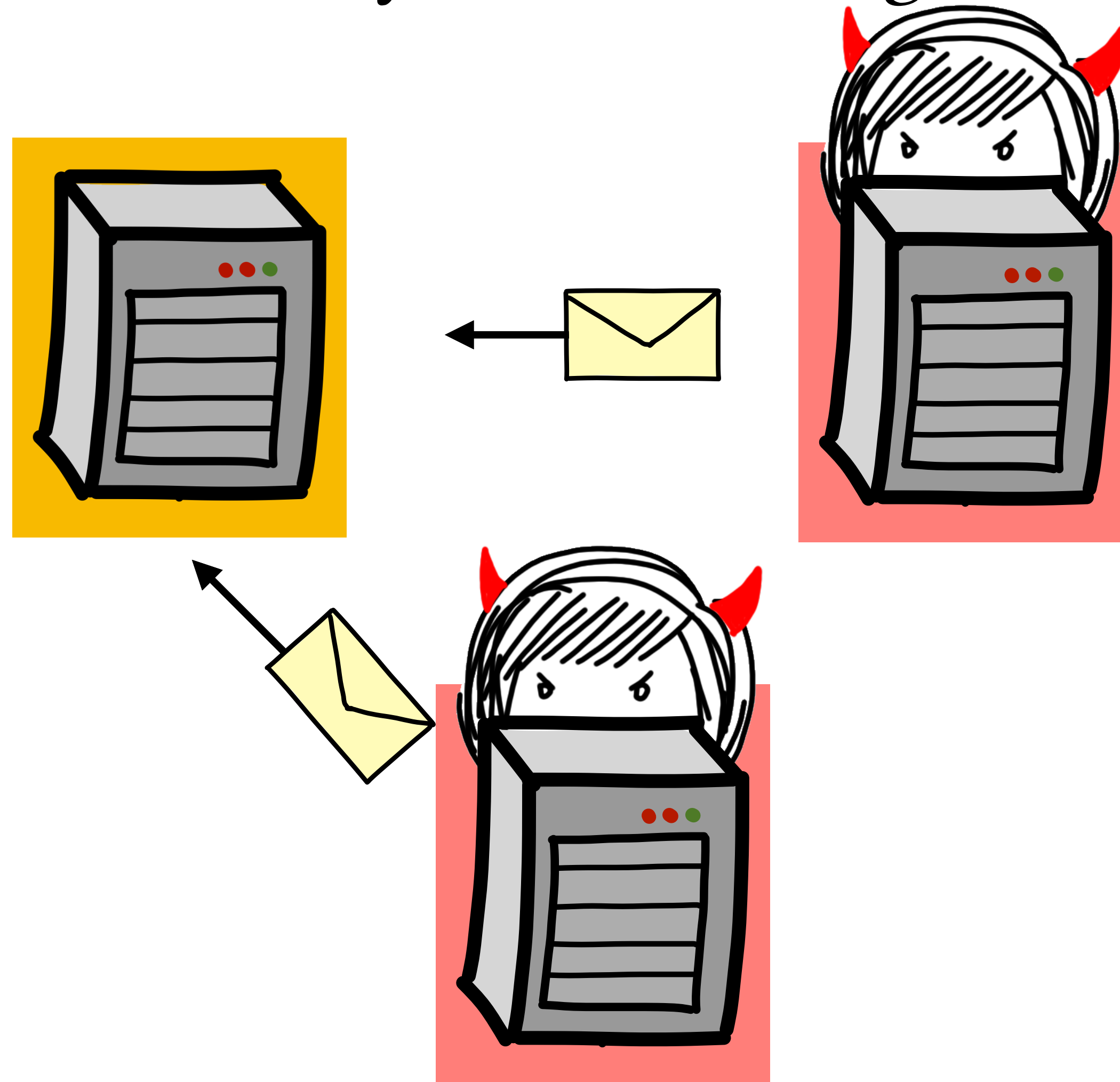
**Best Possible Security:** Protection against 2 corruptions



Threshold signing via  
“Dishonest majority”  
MPC protocol

# $(3,n)$ Signing

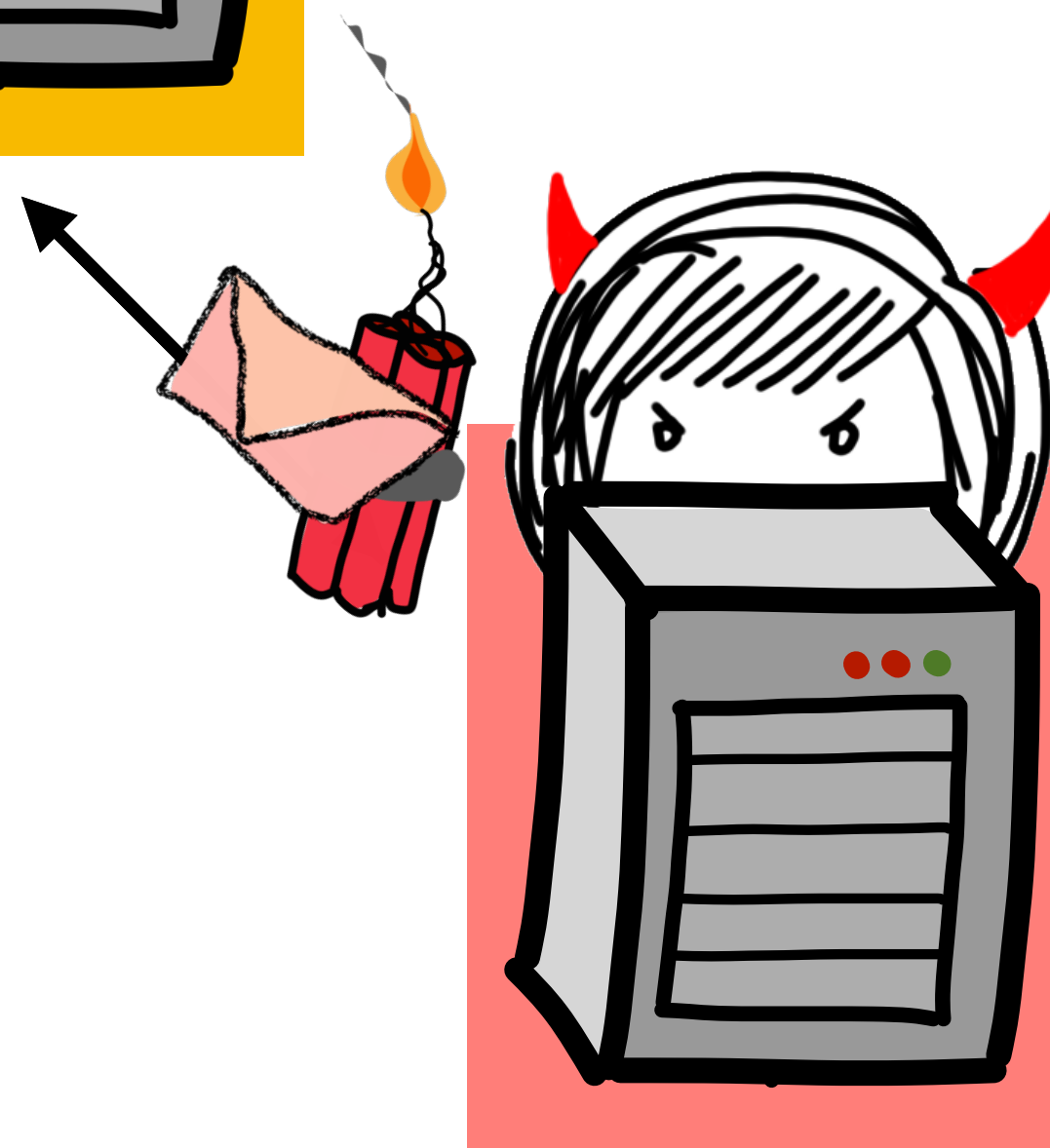
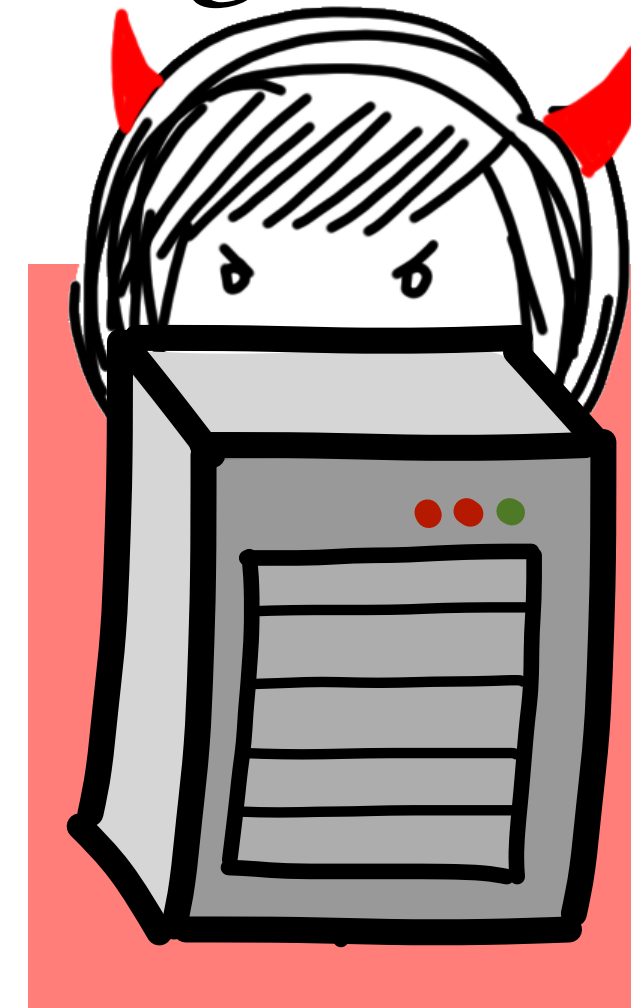
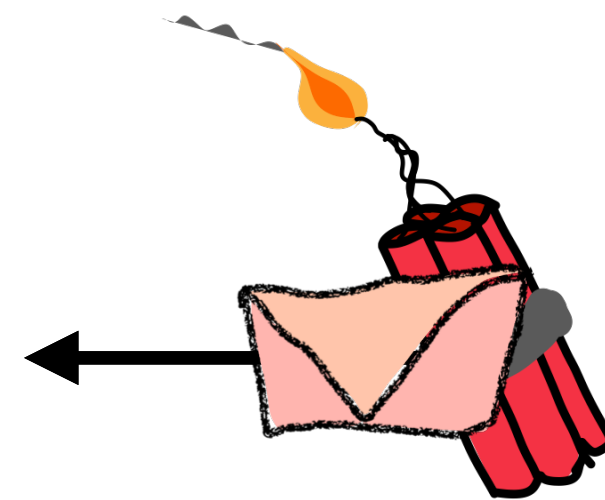
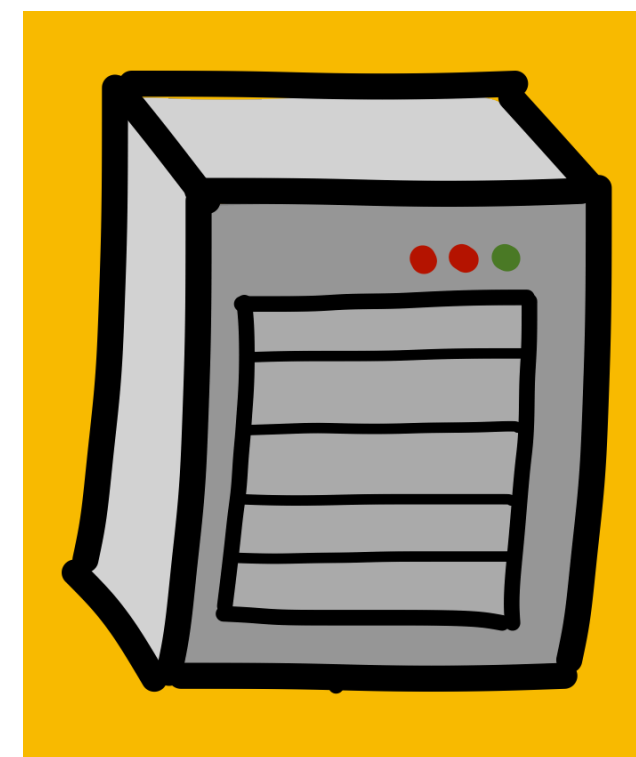
**Best Possible Security:** Protection against 2 corruptions



Threshold signing via  
“Dishonest majority”  
MPC protocol

# $(3,n)$ Signing

**Best Possible Security:** Protection against 2 corruptions



Threshold signing via  
“Dishonest majority”  
MPC protocol



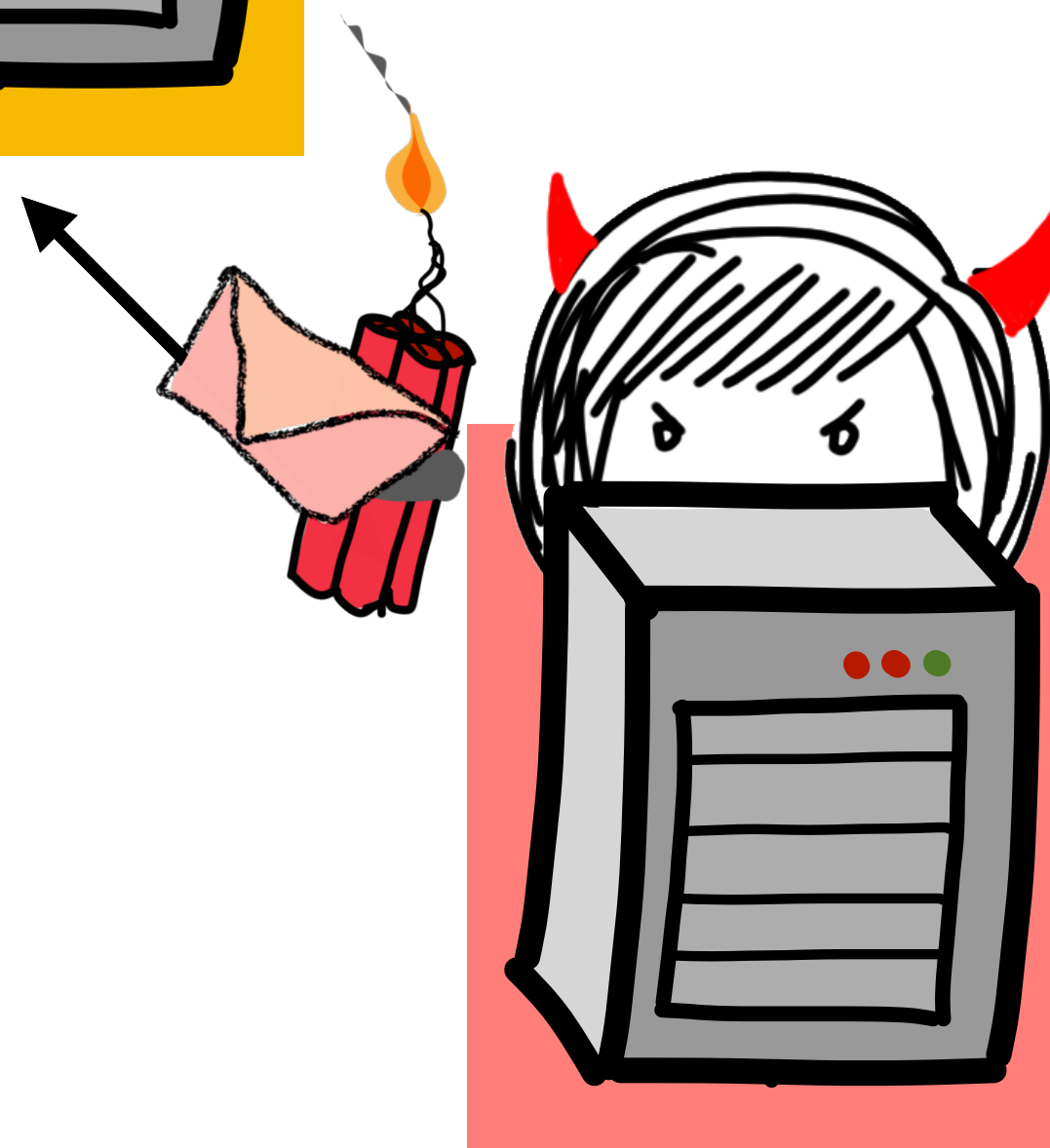
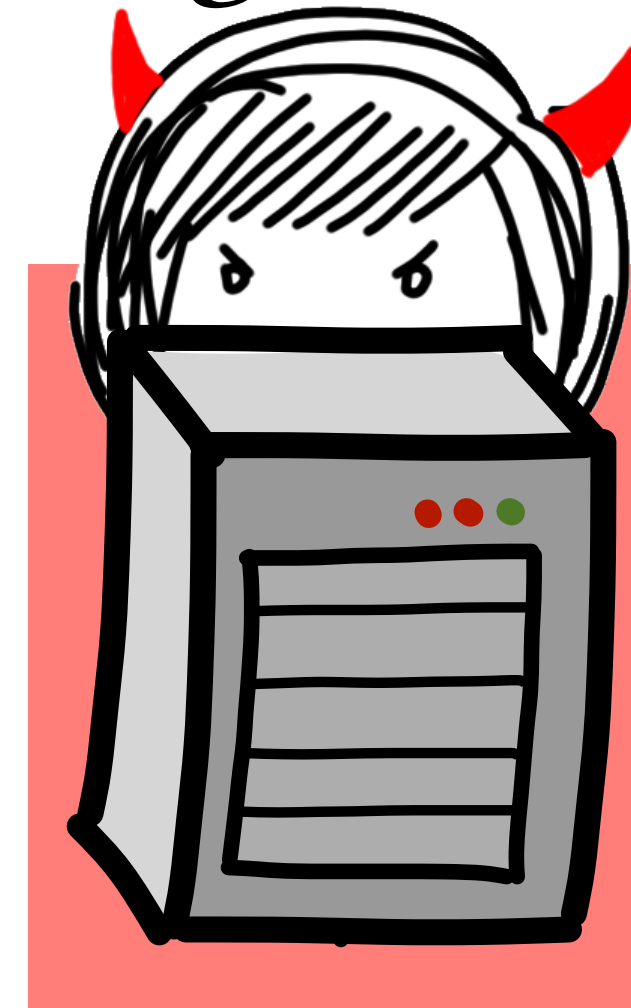
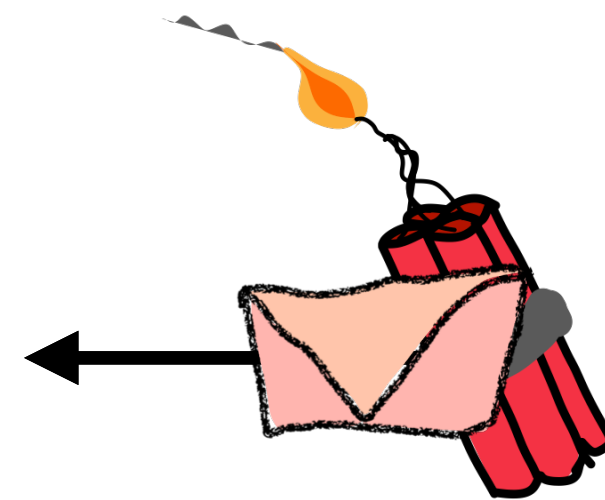
# $(3,n)$ Signing

**Best Possible Security:** Protection against 2 corruptions



still safe!

But, MPC fails  
→ no sig (DoS)  
“security w. abort”



Threshold signing via  
“Dishonest majority”  
MPC protocol

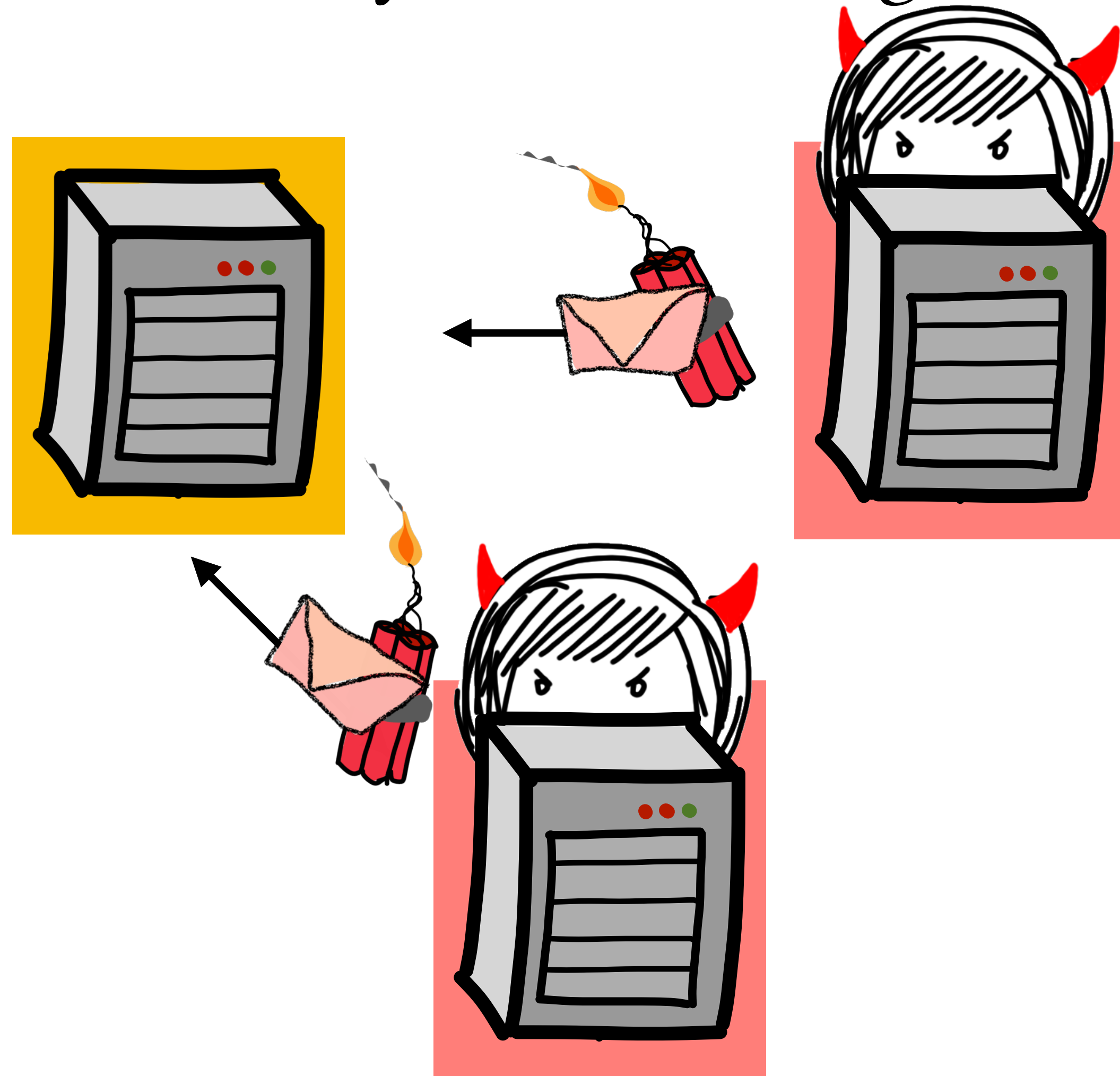
# $(3,n)$ Signing

**Best Possible Security:** Protection against 2 corruptions

 still safe!

But, MPC fails  
→ no sig (DoS)  
“security w. abort”

Folklore remedy:  
*Identifiable Abort*



Threshold signing via  
“Dishonest majority”  
MPC protocol

# $(3,n)$ Signing

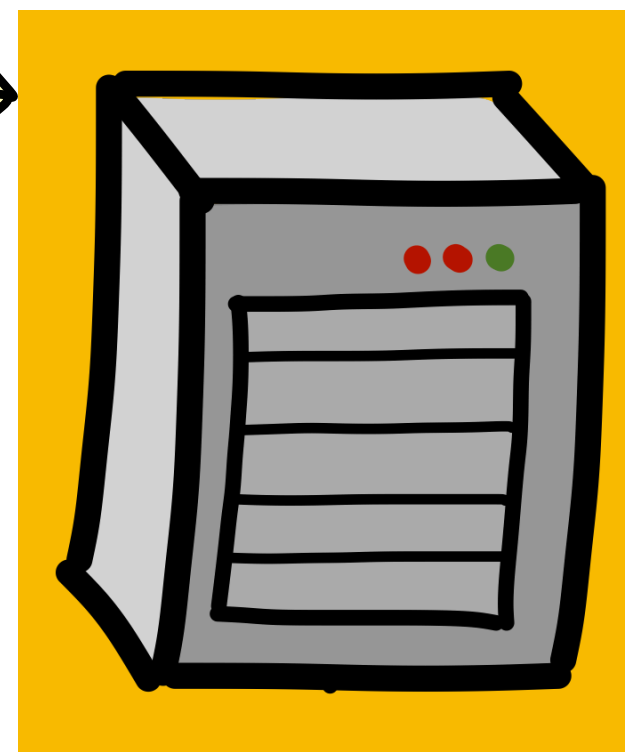
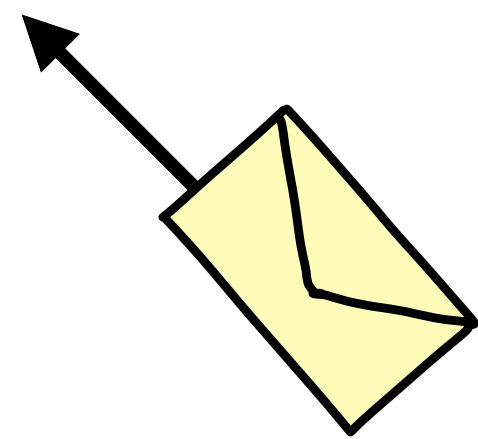
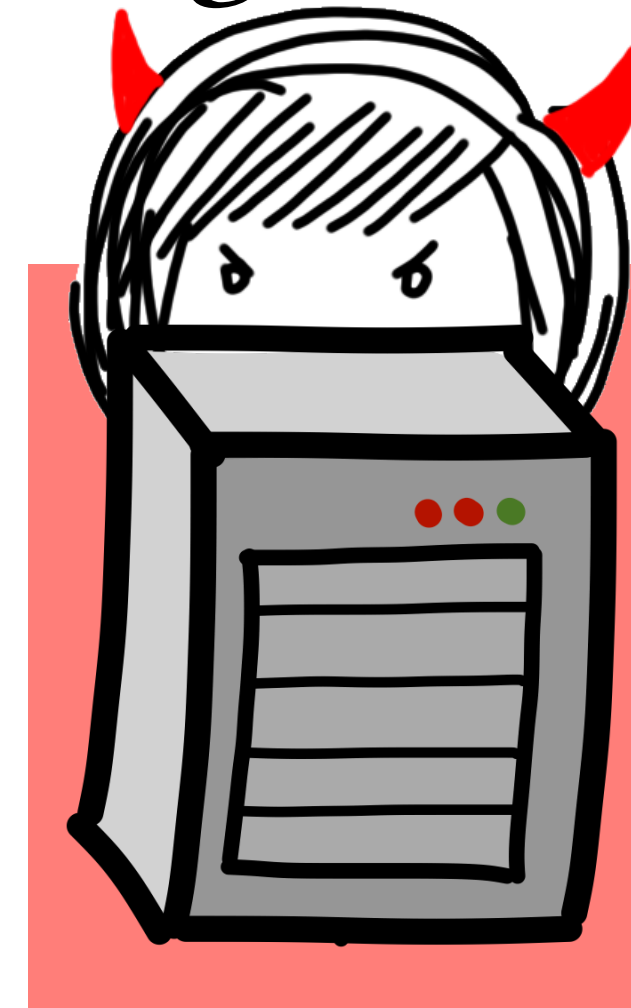
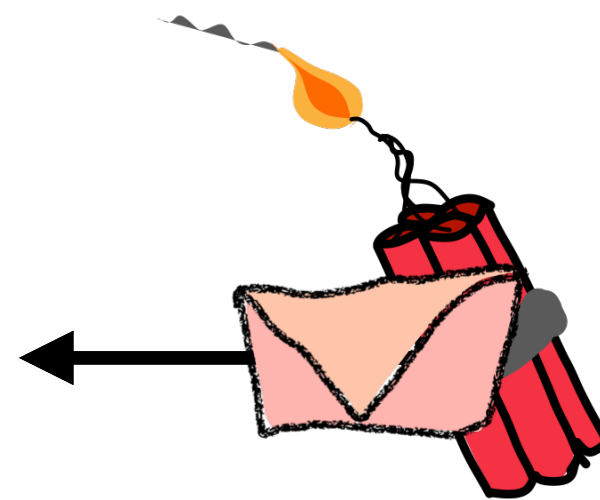
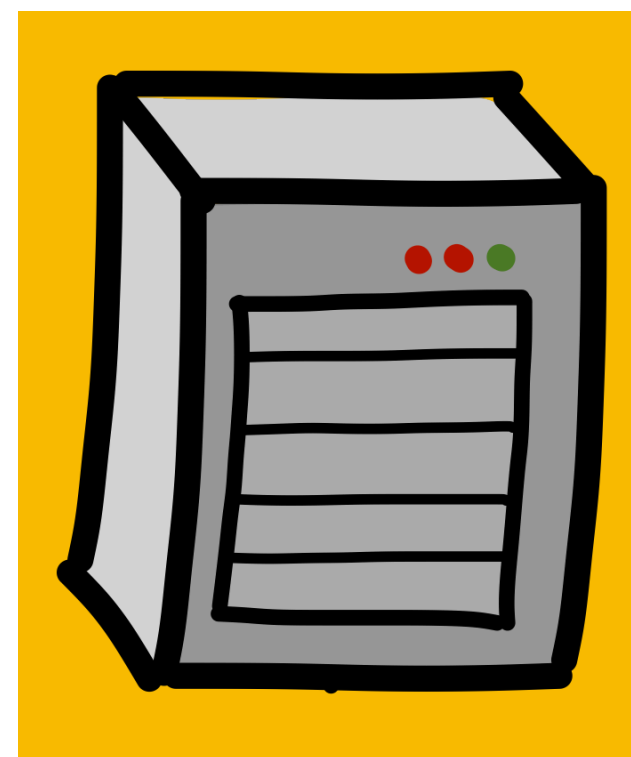
**Best Possible Security:** Protection against 2 corruptions



still safe!

But, MPC fails  
→ no sig (DoS)  
“security w. abort”

Folklore remedy:  
*Identifiable Abort*



# $(3,n)$ Signing

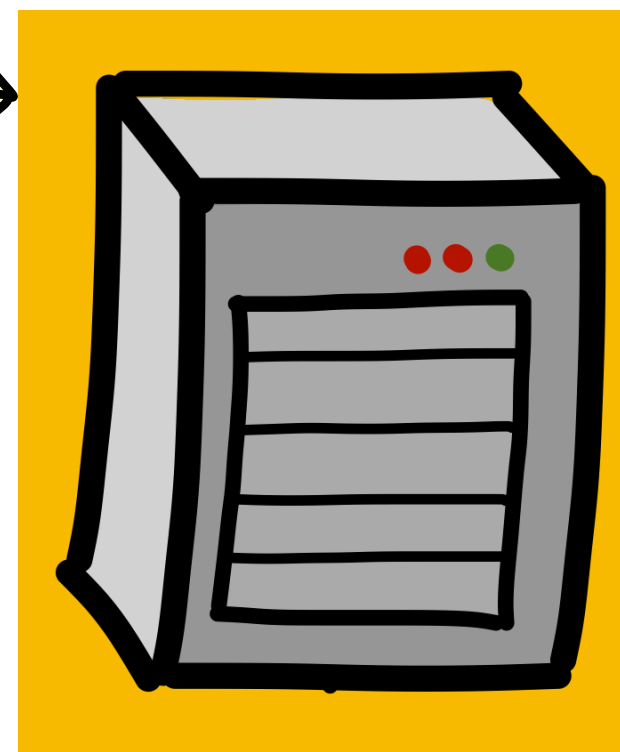
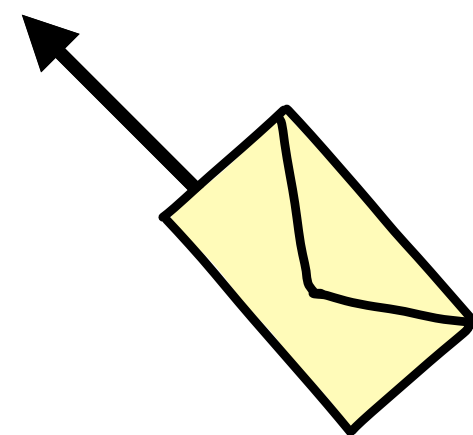
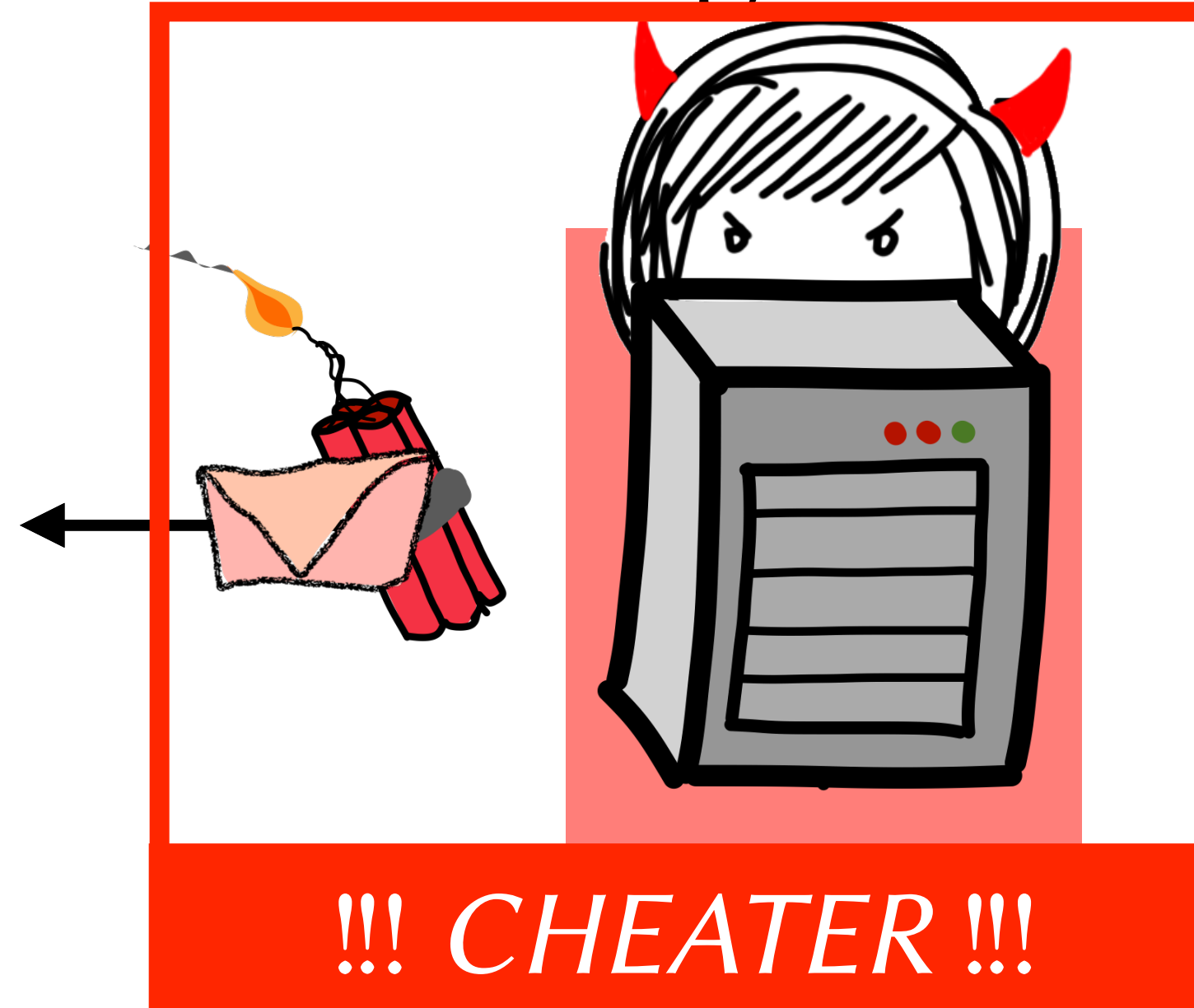
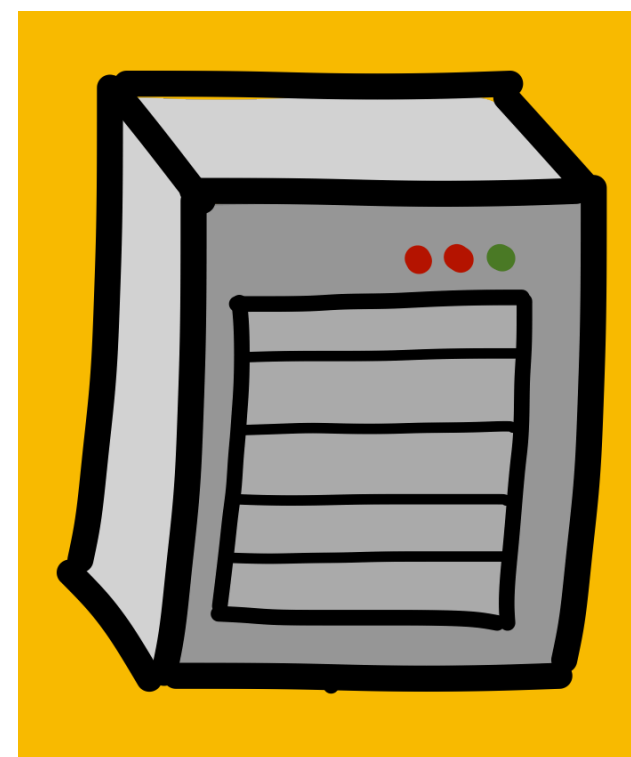
**Best Possible Security:** Protection against 2 corruptions



still safe!

But, MPC fails  
→ no sig (DoS)  
“security w. abort”

Folklore remedy:  
*Identifiable Abort*





# $(3,n)$ Signing

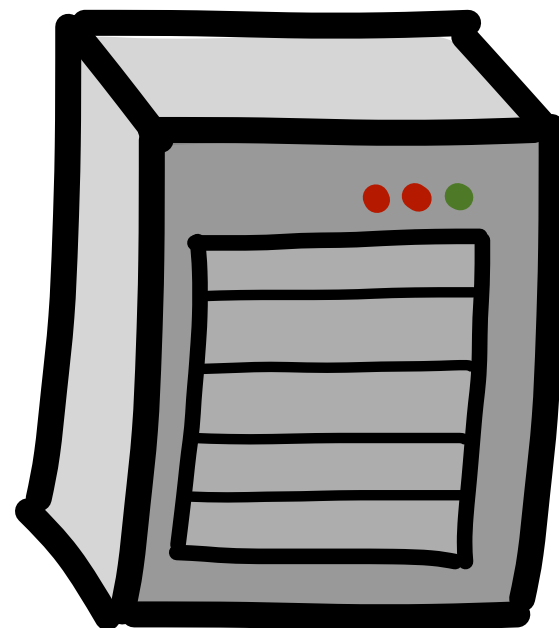
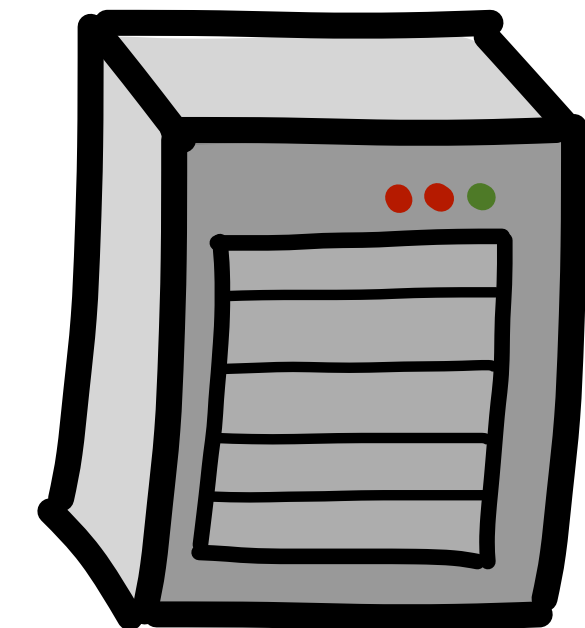
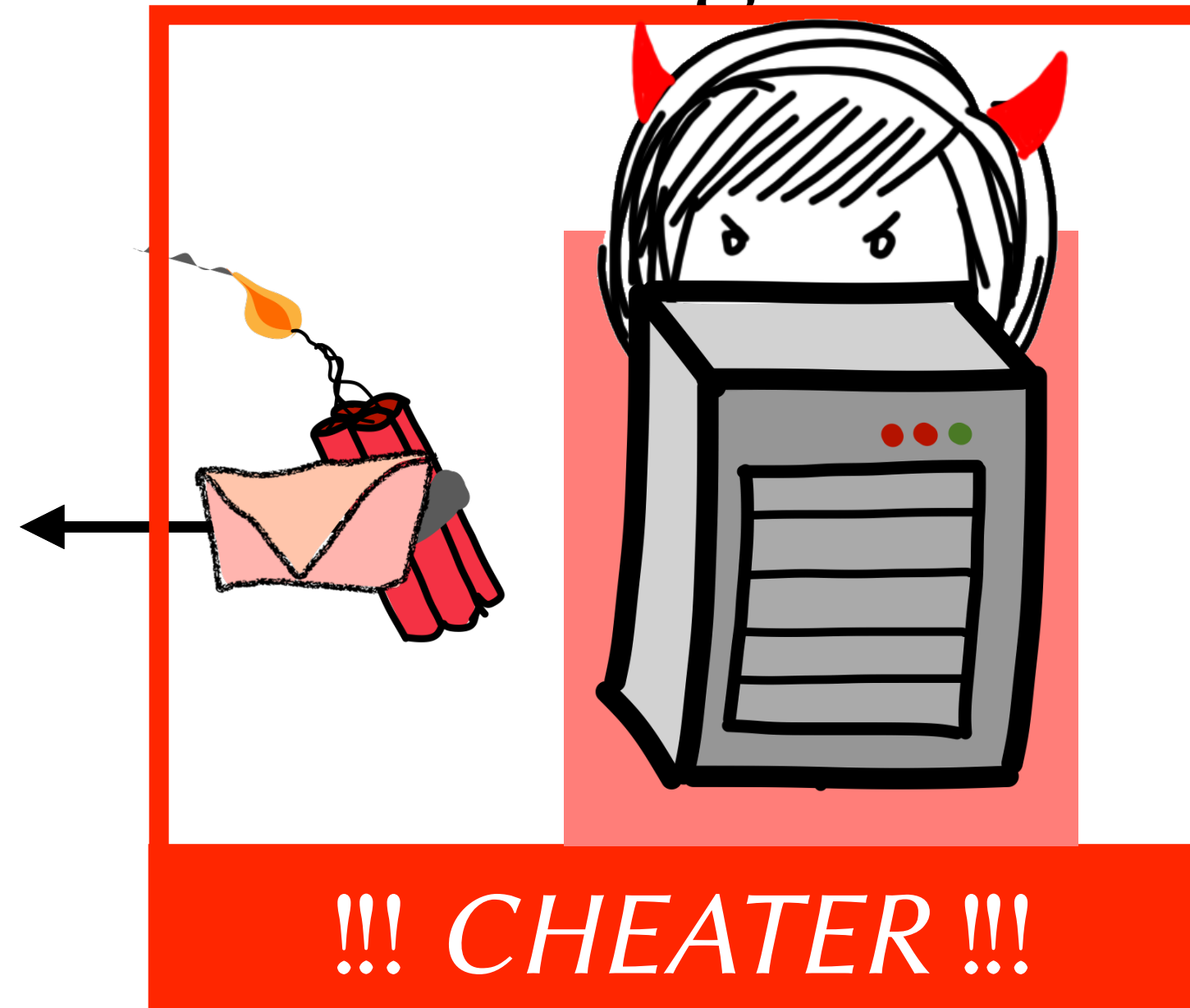
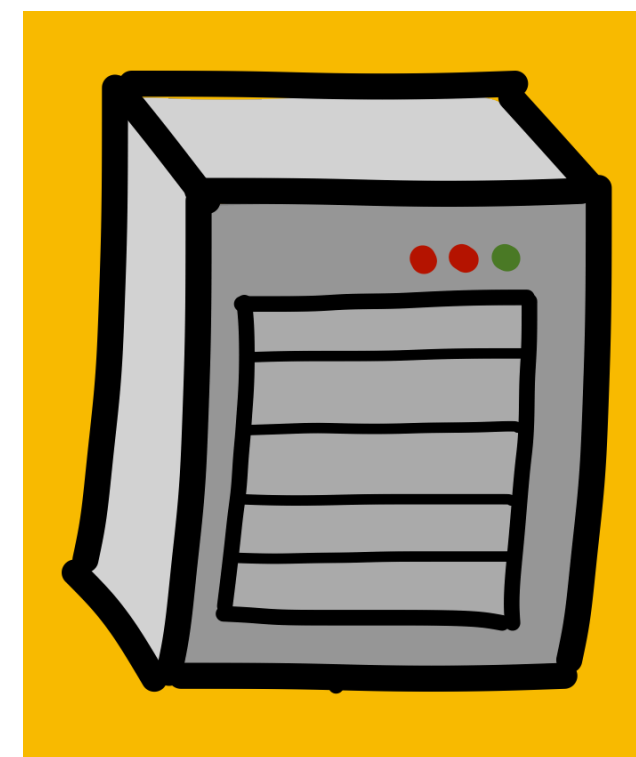
**Best Possible Security:** Protection against 2 corruptions



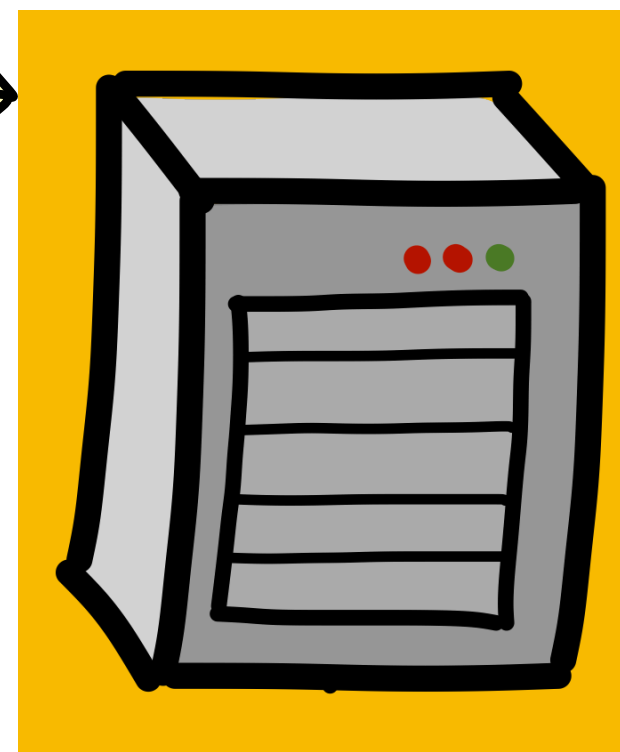
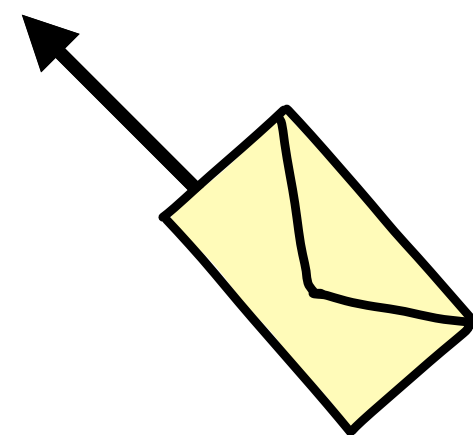
still safe!

But, MPC fails  
→ no sig (DoS)  
“security w. abort”

Folklore remedy:  
*Identifiable Abort*



“Global”  
honest majority





# $(3,n)$ Signing

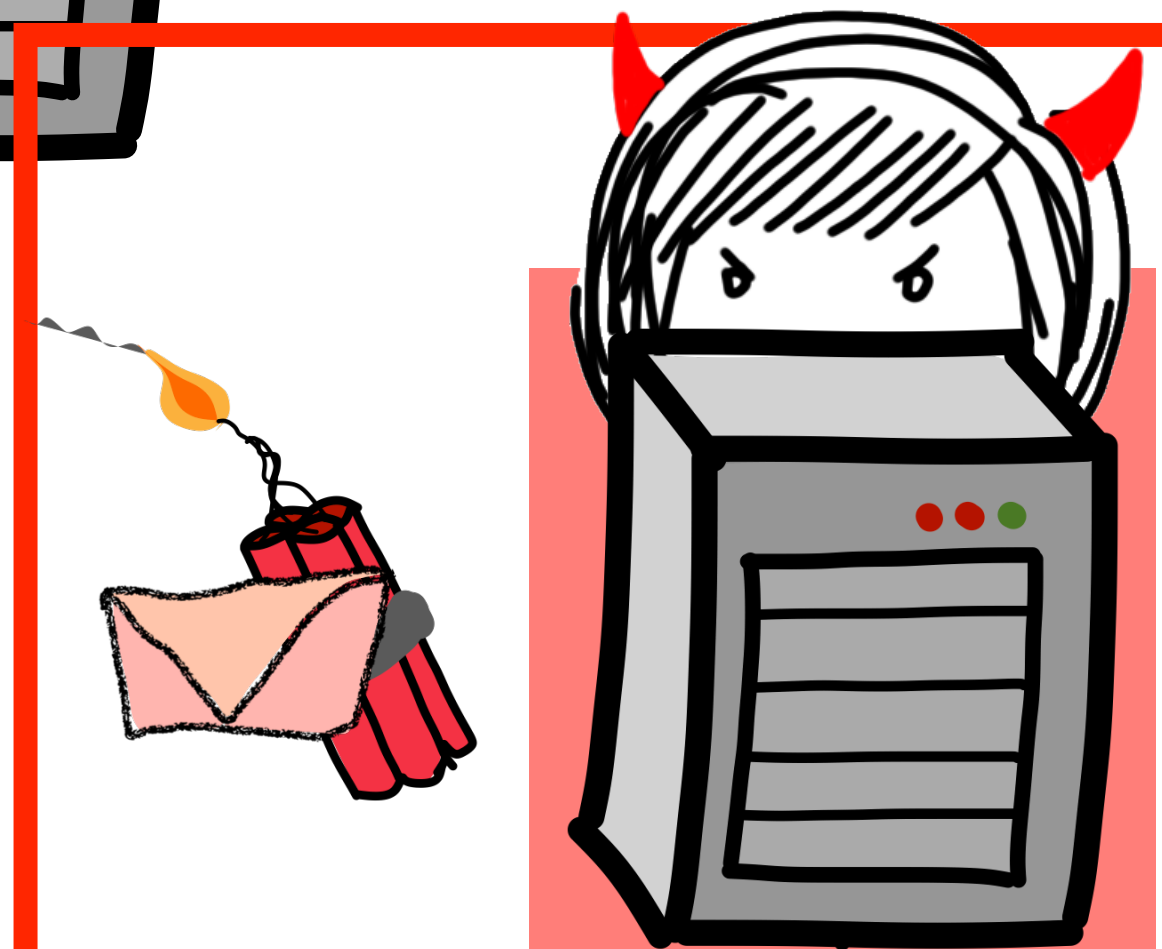
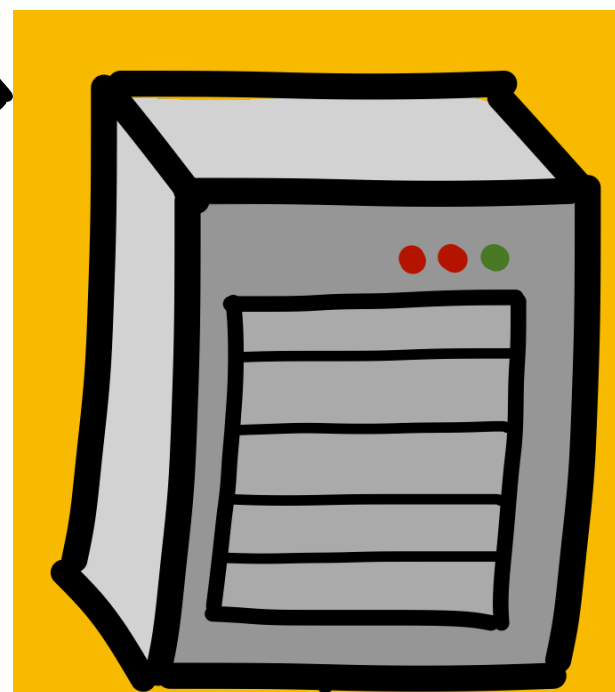
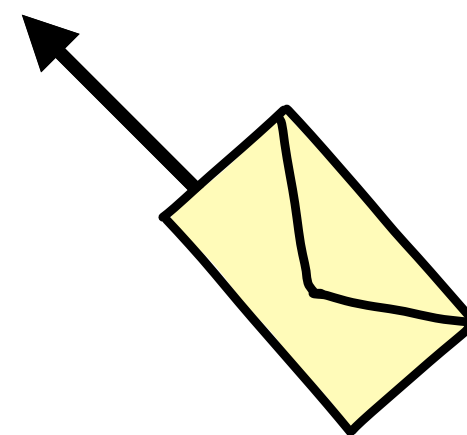
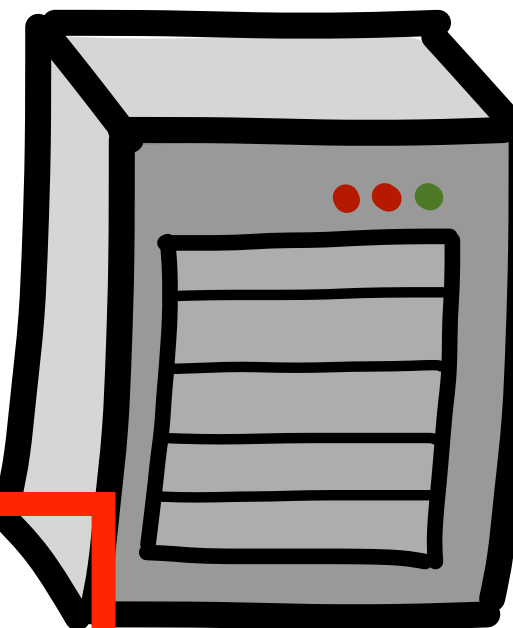
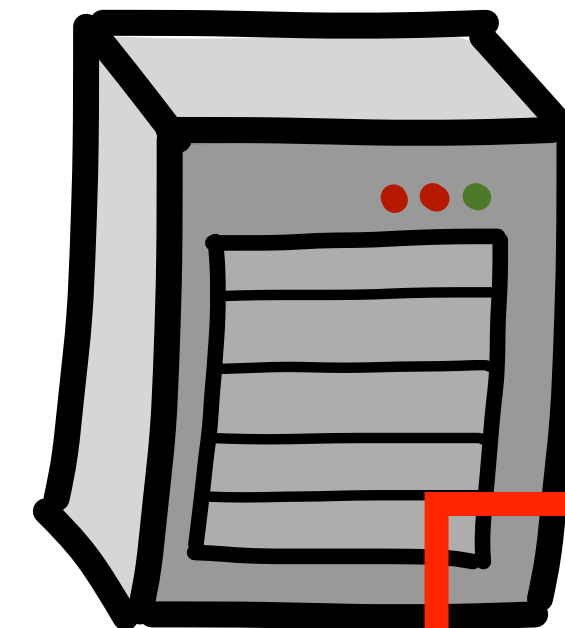
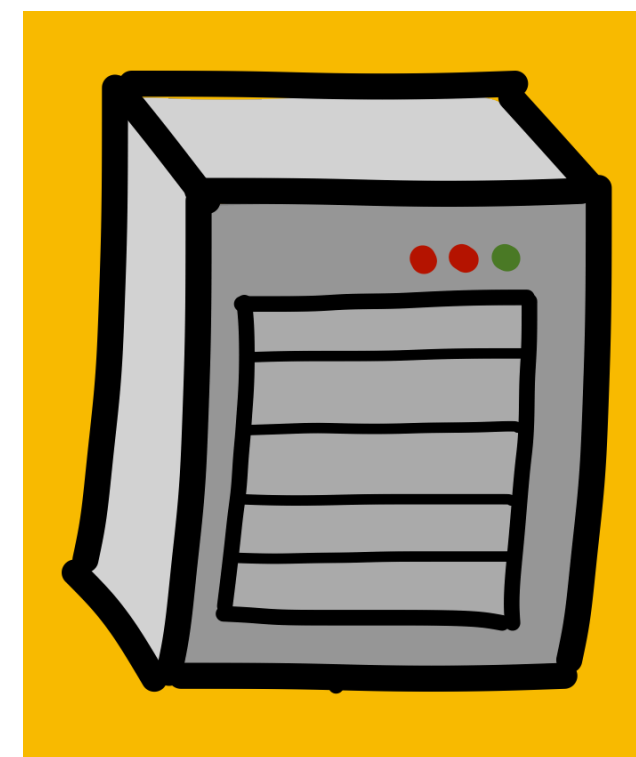
**Best Possible Security:** Protection against 2 corruptions



still safe!

But, MPC fails  
→ no sig (DoS)  
“security w. abort”

Folklore remedy:  
*Identifiable Abort*



!!! CHEATER !!!

# $(3,n)$ Signing

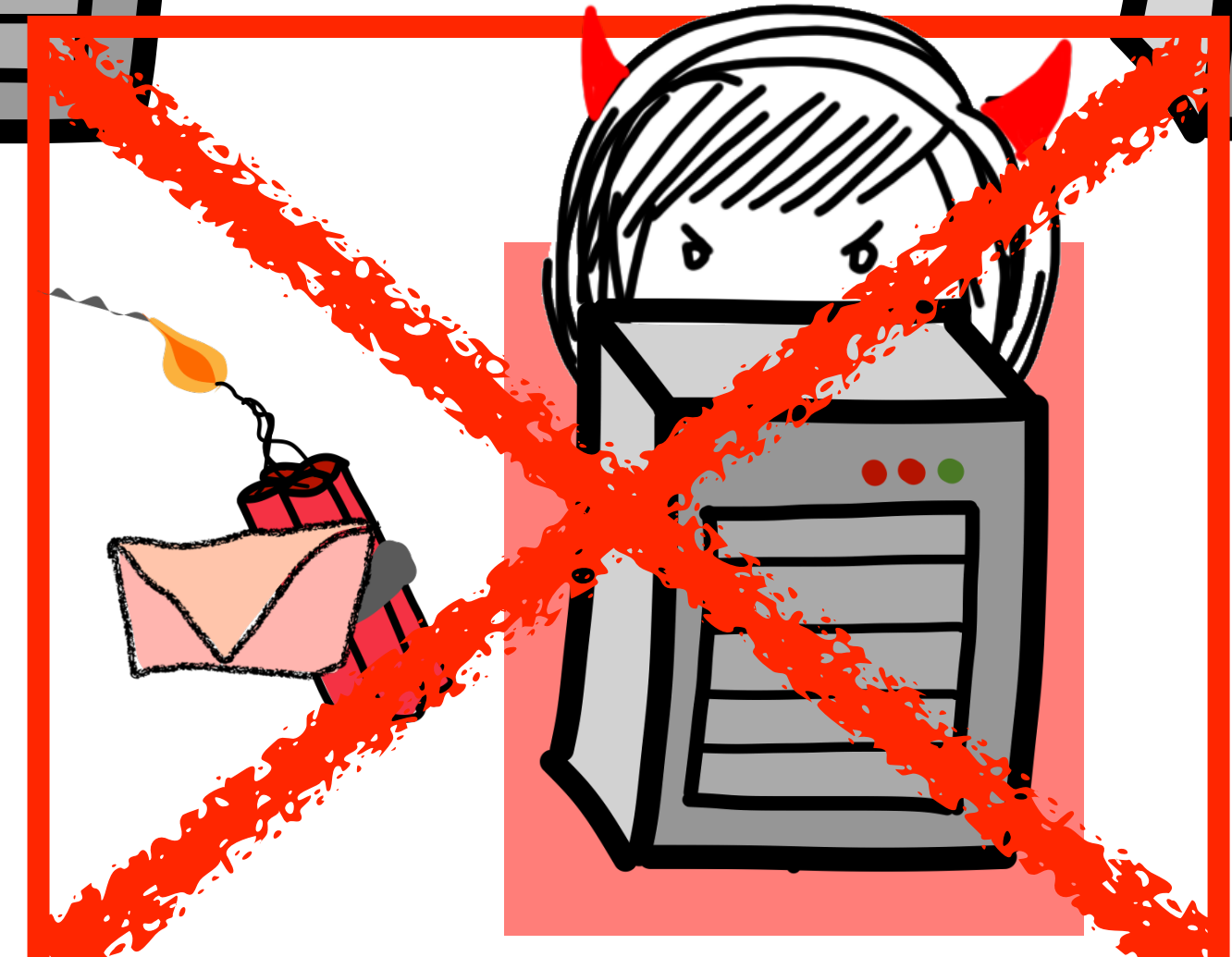
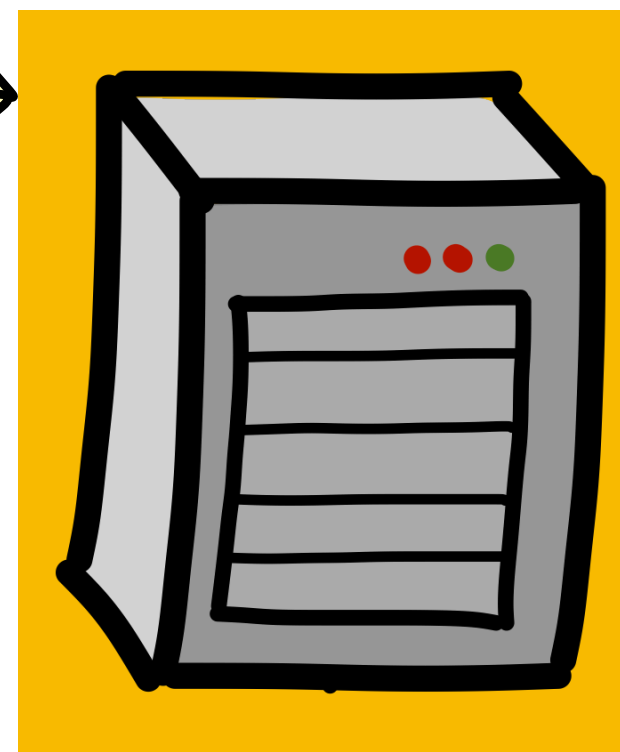
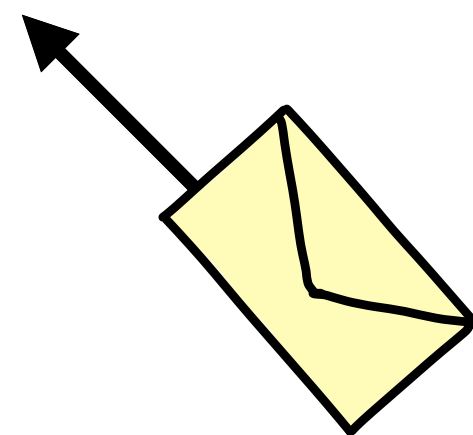
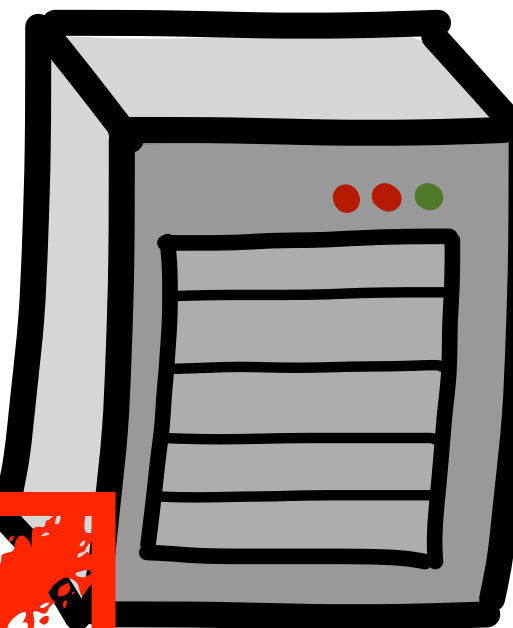
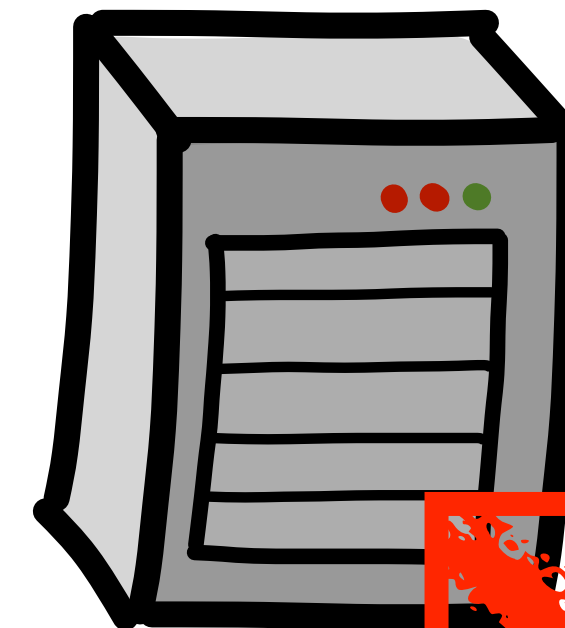
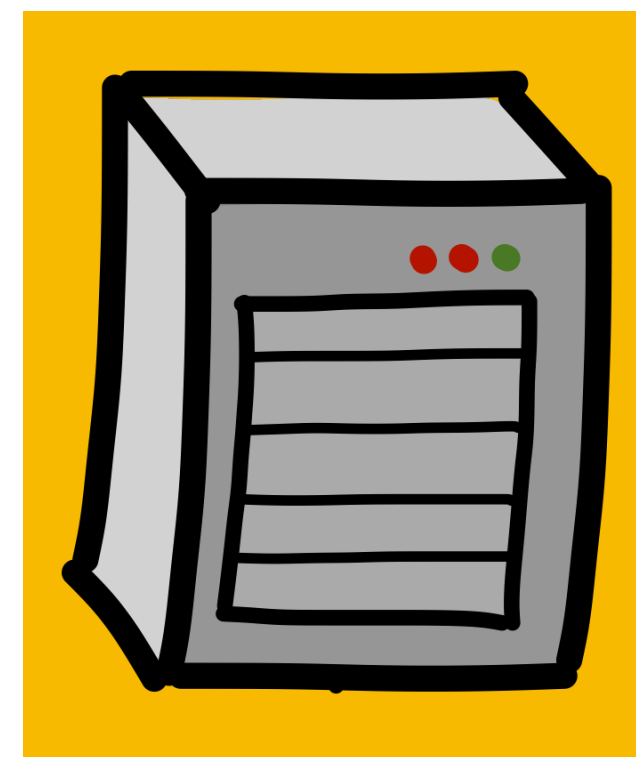
**Best Possible Security:** Protection against 2 corruptions



still safe!

But, MPC fails  
→ no sig (DoS)  
“security w. abort”

Folklore remedy:  
*Identifiable Abort*



!!! CHEATER !!!

# $(3,n)$ Signing

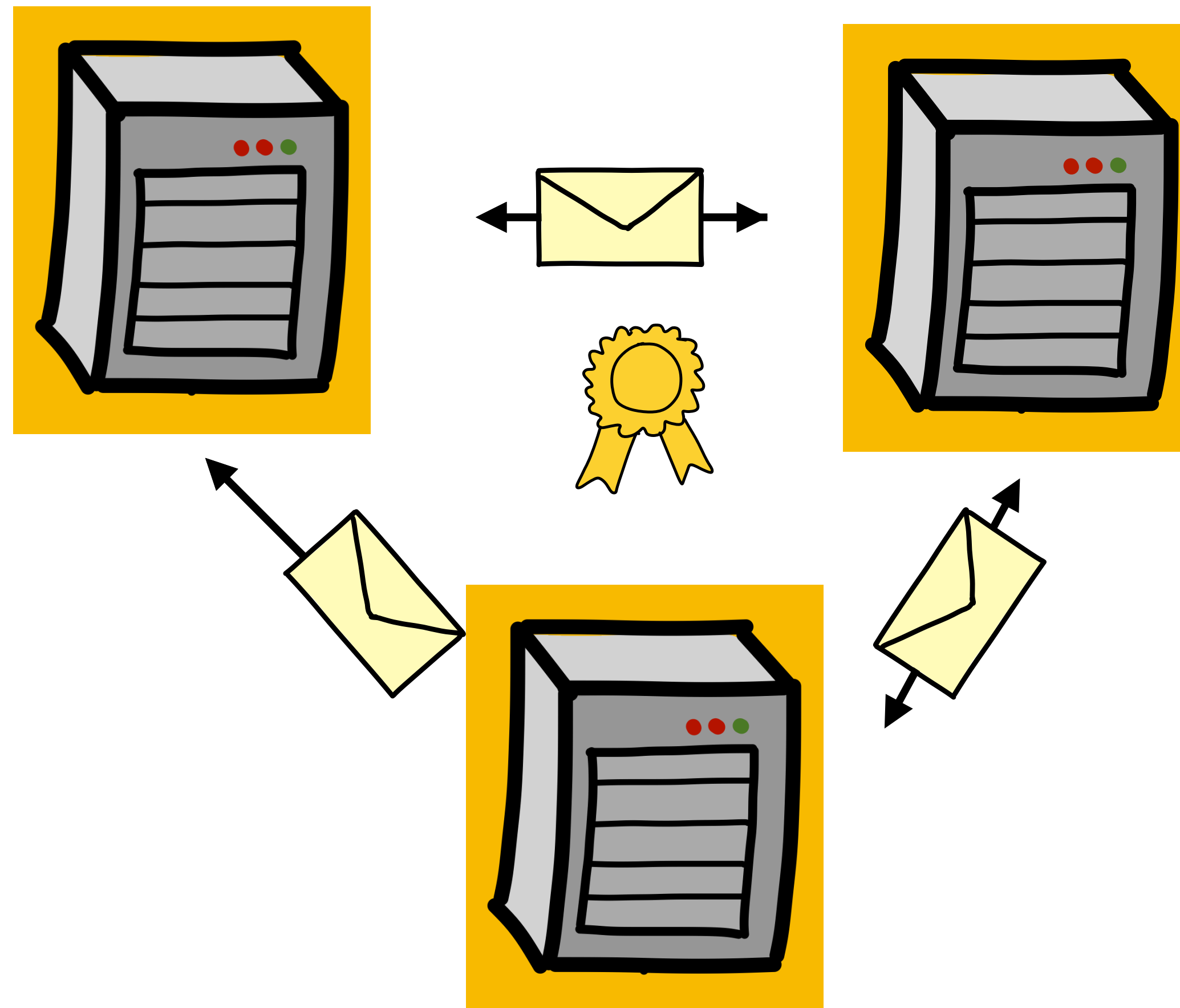
**Best Possible Security:** Protection against 2 corruptions



still safe!

But, MPC fails  
→ no sig (DoS)  
“security w. abort”

Folklore remedy:  
*Identifiable Abort*

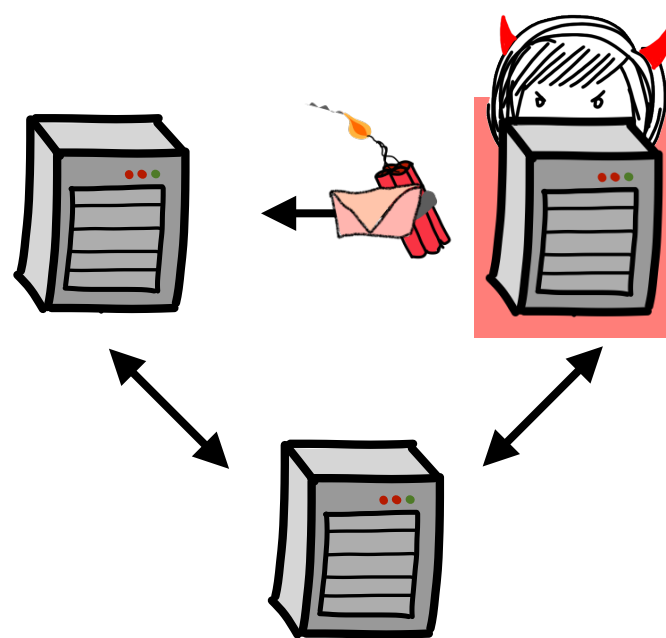


# Identification Mechanisms

- Cheater *could* be found through out of band methods.
- We want **certifiable** protocol mechanism to identify who crashed the protocol  
⇒ each party either gets output, or **identity of cheating party + cert. of cheat**

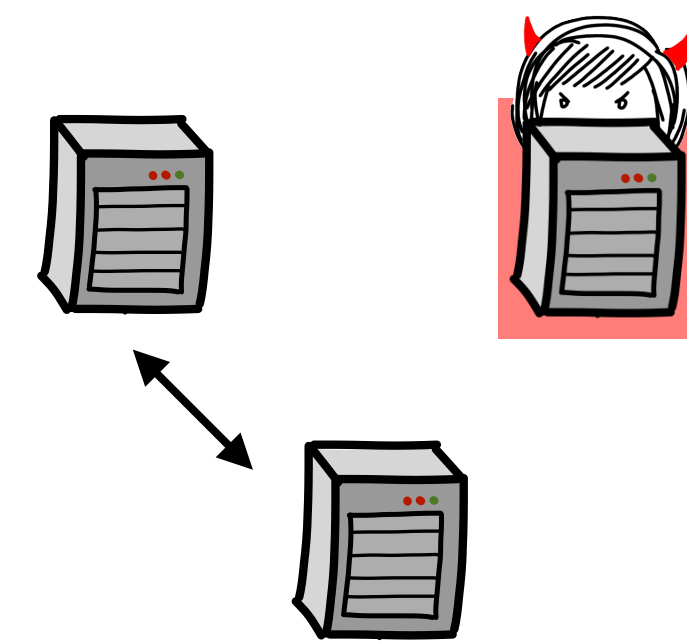
Note: no consensus on identity

- Two ways to crash protocol:




1. Malformed protocol message

⋮



2. No message at all

# Anatomy of MPC-ECDSA w. IA



Baseline security-with-abort protocol



# Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee  
wellformedness of every sent message

Baseline security-with-abort protocol

# Anatomy of MPC-ECDSA w. IA

Mitigate via  
ZK proofs,  
opening input

Mechanism to guarantee  
wellformedness of every sent message

[Canetti Gennaro Goldfeder  
Makriyannis Peled 20],  
[Cohen Doerner  
**K** shelat 24]

Baseline security-with-abort protocol

# Anatomy of MPC-ECDSA w. IA

Mitigate via  
ZK proofs,  
opening input

Mechanism to guarantee  
wellformedness of every sent message

[Canetti Gennaro Goldfeder  
Makriyannis Peled 20],  
[Cohen Doerner  
**K** shelat 24]

Baseline security-with-abort protocol

Mechanism to guarantee  
each party sends *some* message every round

# Anatomy of MPC-ECDSA w. IA

Mitigate via  
ZK proofs,  
opening input

Mechanism to guarantee  
wellformedness of every sent message

[Canetti Gennaro Goldfeder  
Makriyannis Peled 20],  
[Cohen Doerner  
**K** shelat 24]

Baseline security-with-abort protocol

Send all  
messages over  
broadcast

Mechanism to guarantee  
each party sends *some* message every round

# Anatomy of MPC-ECDSA w. IA

Mitigate via  
ZK proofs,  
opening input

Mechanism to guarantee  
wellformedness of every sent message

[Canetti Gennaro Goldfeder  
Makriyannis Peled 20],  
[Cohen Doerner  
**K** shelat 24]

Baseline security-with-abort protocol

Send all  
messages over  
broadcast

Mechanism to guarantee  
each party sends *some* message every round

Can of worms



# “Broadcast”?

- Engineering Anecdota:
  - “Do I really need to implement broadcast?”
    - “*yes*”
    - “Is it just for some theoretical proof nonsense?”
      - “*no, it’s to catch parties that don’t send messages for example*”
    - “That seems unnecessary, I can just scan the network logs”
- In some settings: coordinator routes all messages
  - ⇒ implicit single point of failure
- Other settings: use external broadcast channel like a blockchain
  - ⇒ expensive, slow, introduces external dependencies

# Broadcast Protocols

- [Cohen Lindell 14] MPC-IA implies broadcast: compute  $\mathcal{F}_{\text{PKI}}$  with IA
  - PKI already available (+synchrony), broadcast is *feasible* [Dolev Strong 83] ...but **round complexity is an issue**:  $O(t)$  deterministic, or expected  $O(1)$  randomized with large constants [Katz Koo 06][Abraham Devadas Dolev Nayak Ren 19]
  - This is straightforward in the security with abort setting, via simple echo broadcast [Goldwasser Lindell 02]
- Can we construct a simple instantiation of BC as suitable for IA?  
**Goal**: an ECDSA-IA protocol that is easy to deploy over p2p channels

# BC-IA Properties

- **Consistency**: All honest parties that output a valid (dealer signed) message will be in agreement
- If the sender is corrupt, an honest party alternatively obtains a certificate:
  - (An attempt to) violate consistency, yields a **certificate of cheating**  $\Omega$
  - If the sender sends nothing, yields a **certificate of non-responsiveness**  $\omega$
- $\Omega$  vs.  $\omega$ : Definite misbehaviour vs. potential network fault—different penalties
- **Defamation-freeness**: Honest party can't be framed with  $\Omega$  or  $\omega$

# Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee  
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee  
each party sends *some* message every round

# Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee  
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee  
each party sends *some* message every round

This work: define “Broadcast-IA”

# Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee  
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee  
each party sends *some* message every round

This work: define “Broadcast-IA”

- Impossible w. dishonest majority
- 2-round honest-majority protocol



# Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee  
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee  
each party sends *some* message every round

Simple honest-majority ECDSA

This work: define “Broadcast-IA”

- Impossible w. dishonest majority
- 2-round honest-majority protocol

# Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee  
wellformedness of every sent message

Light ZK proofs in  $\mathbb{G}$   
+ verifiable complaints

Baseline security-with-abort protocol

Simple honest-majority ECDSA

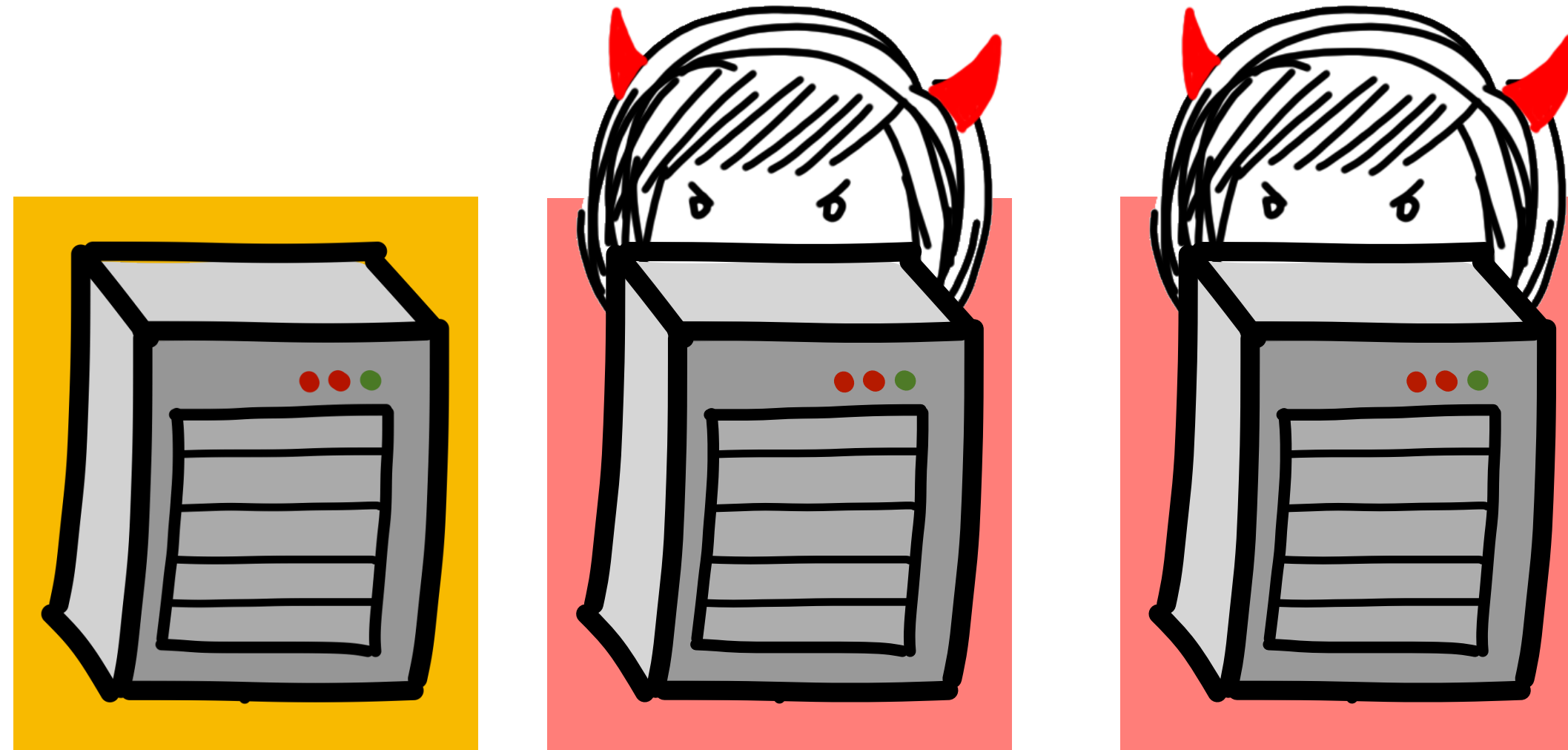
Mechanism to guarantee  
each party sends *some* message every round

This work: define “Broadcast-IA”

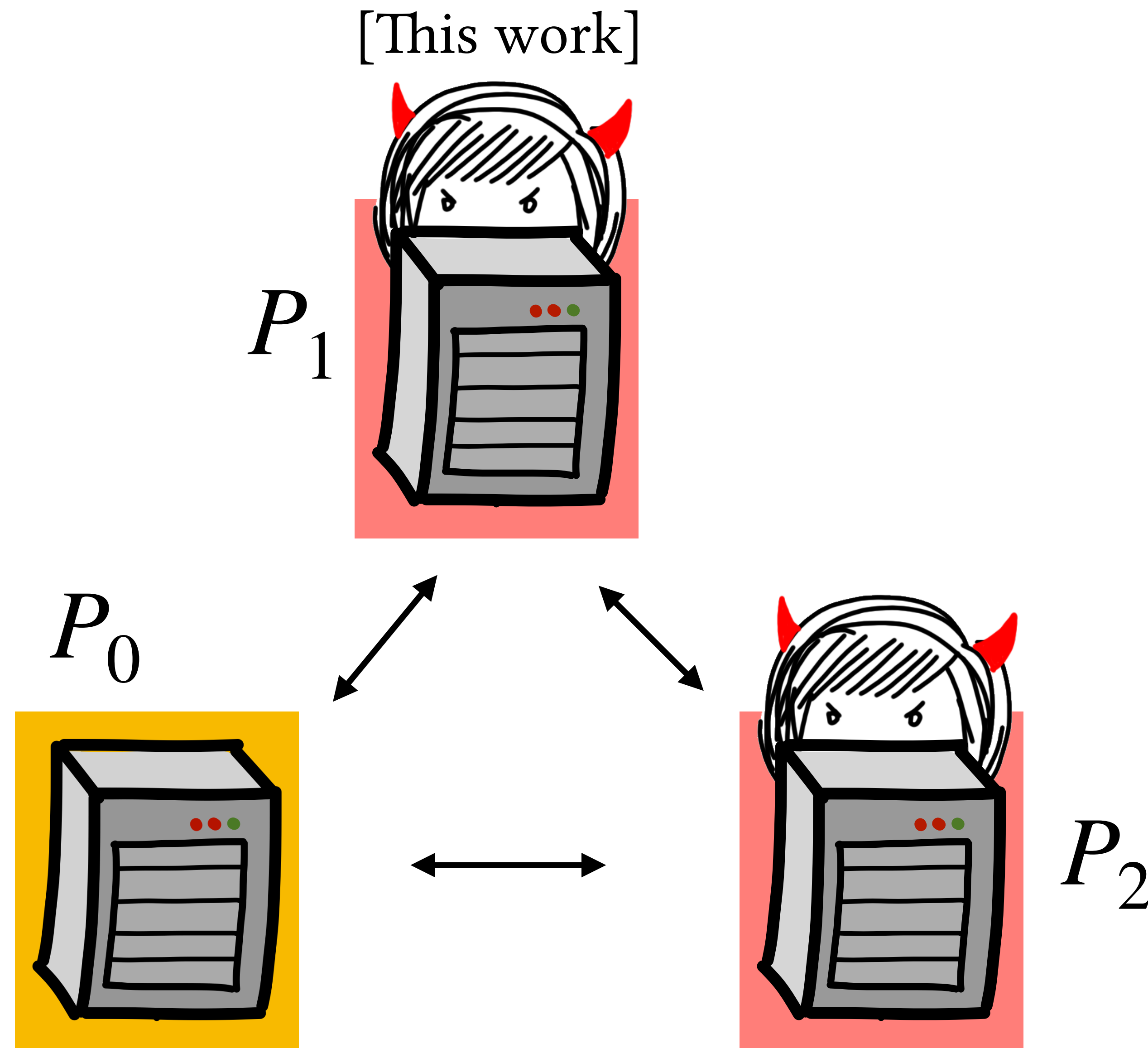
- Impossible w. dishonest majority
- 2-round honest-majority protocol

# Broadcast-IA is Impossible with Dishonest Majority

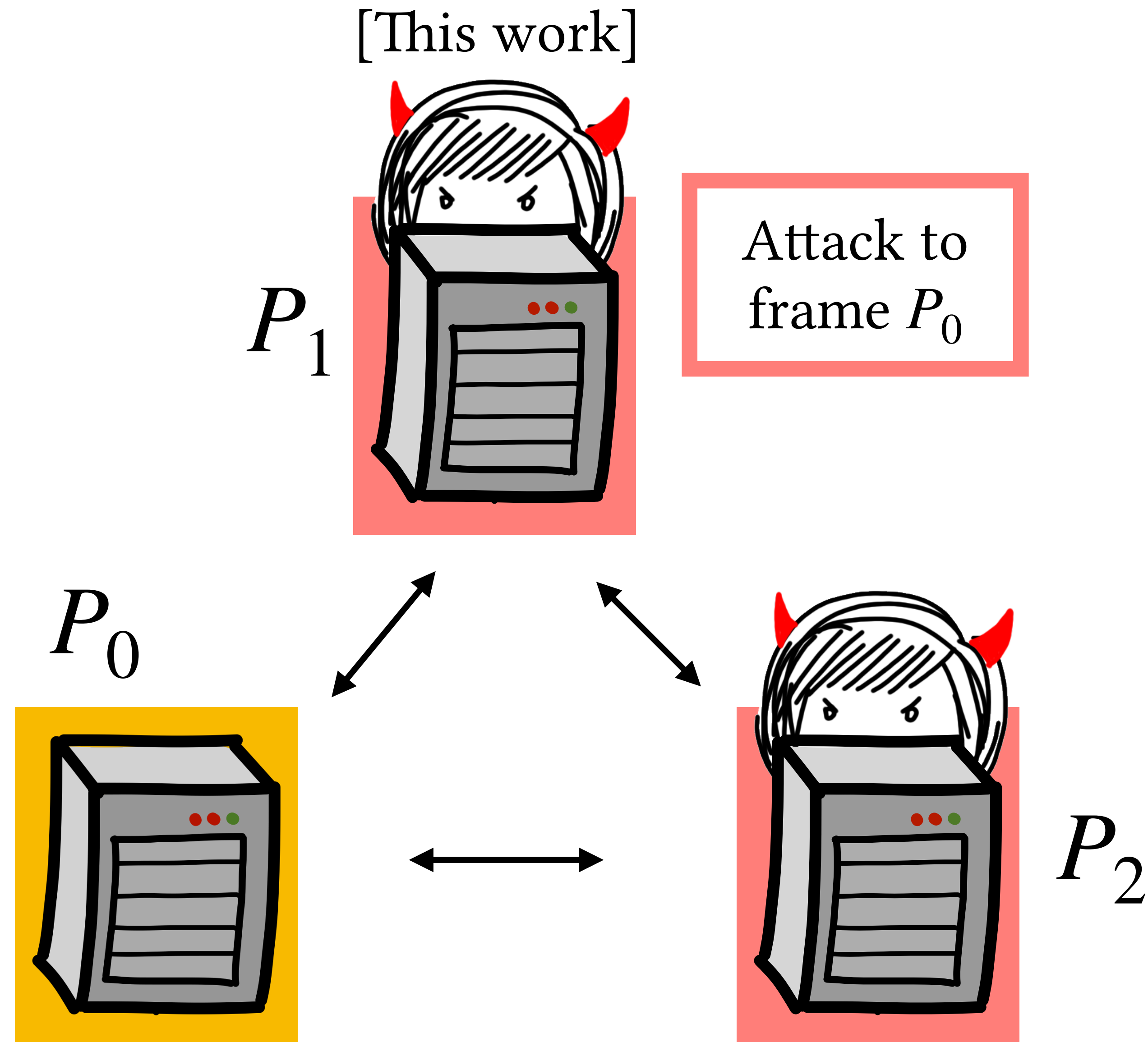
[This work]



# Broadcast-IA is Impossible with Dishonest Majority

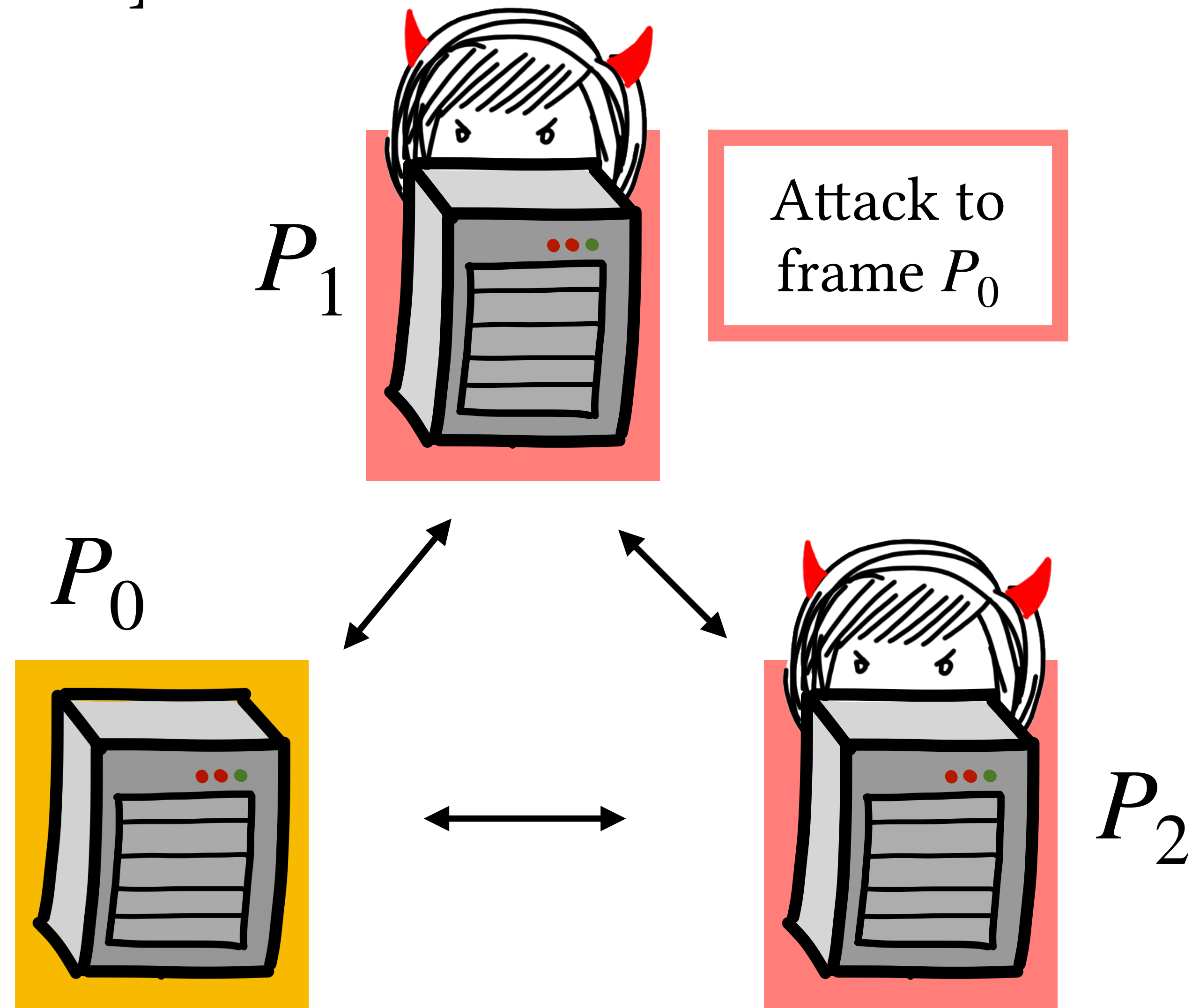


# Broadcast-IA is Impossible with Dishonest Majority



# Broadcast-IA is Impossible with Dishonest Majority

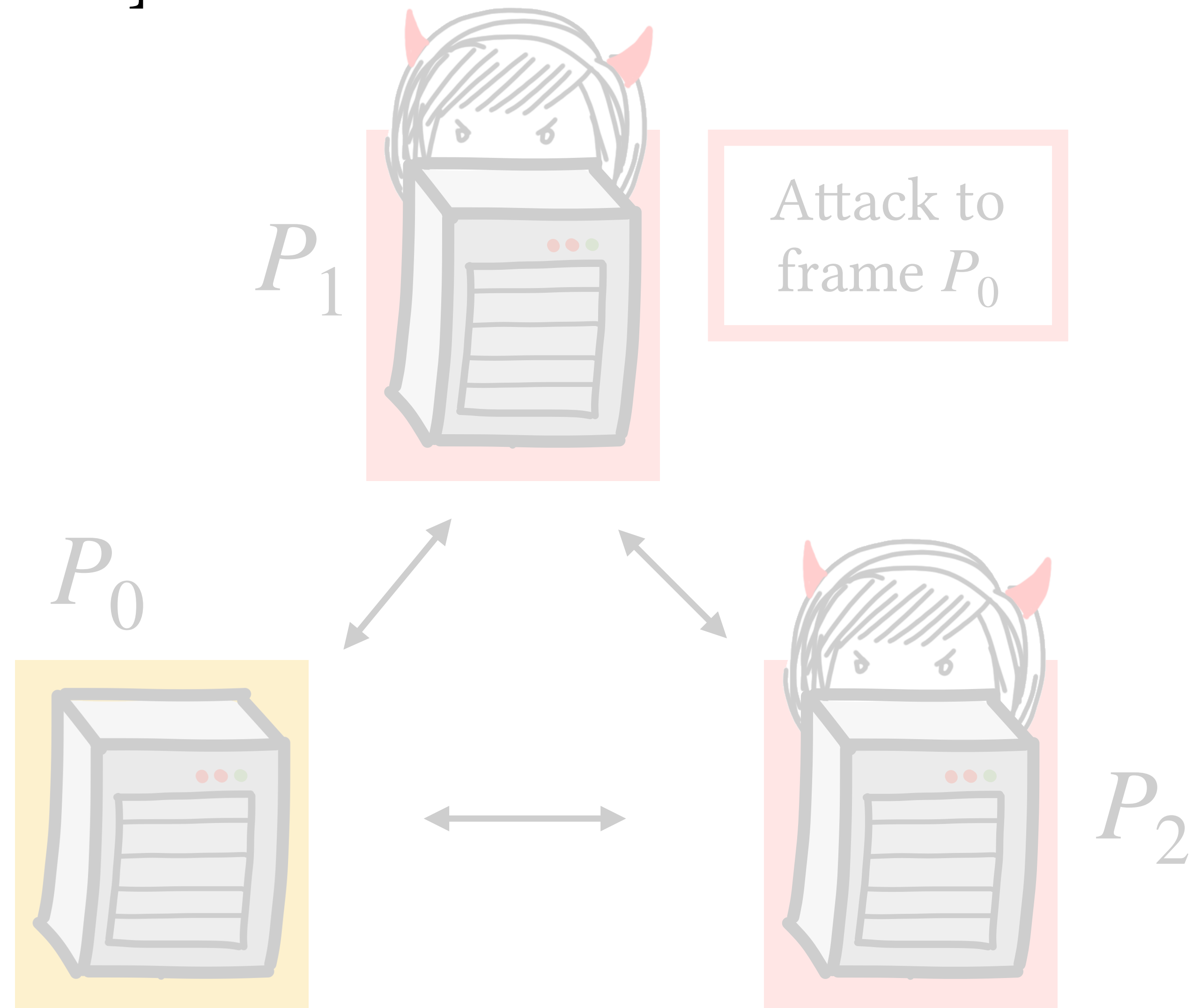
[This work]





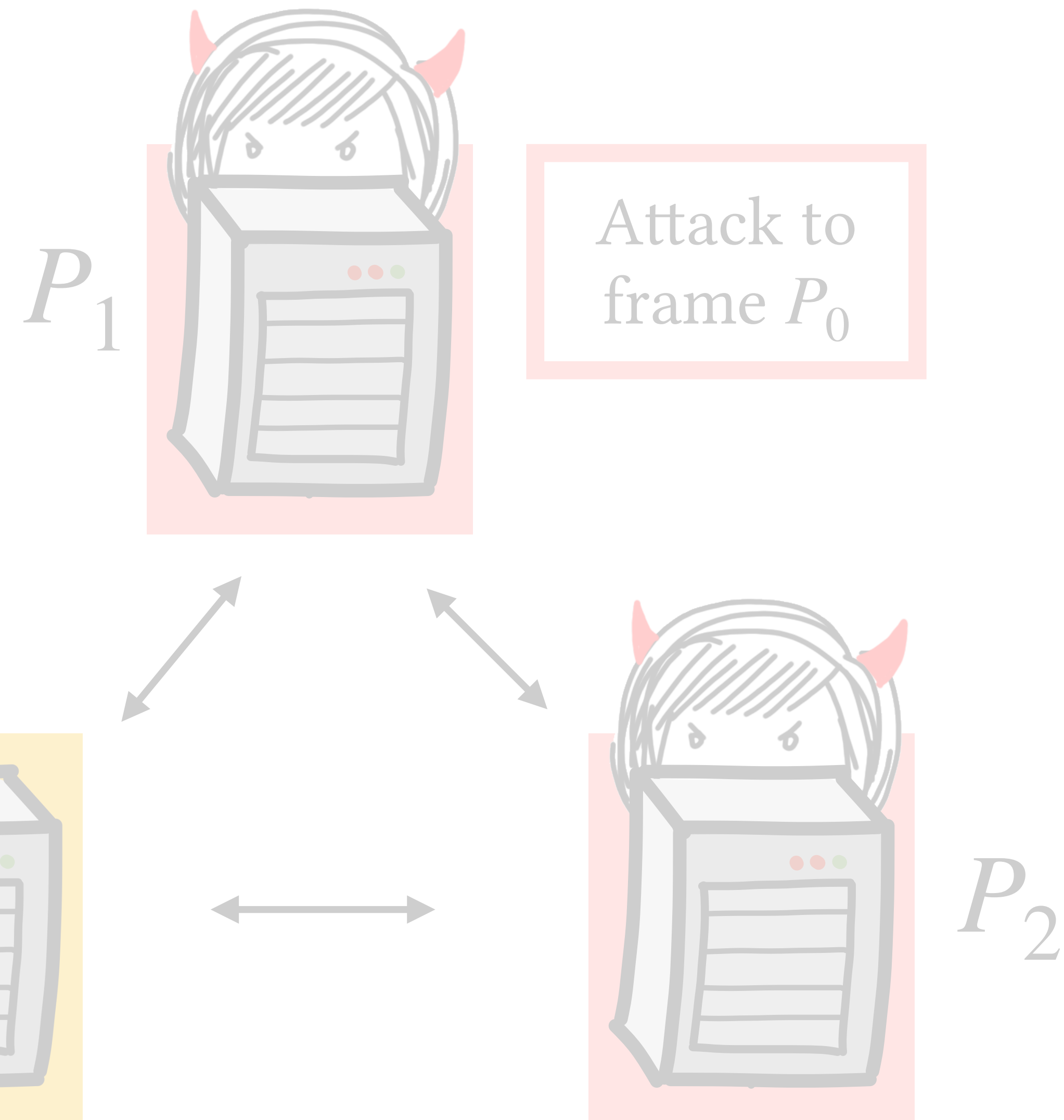
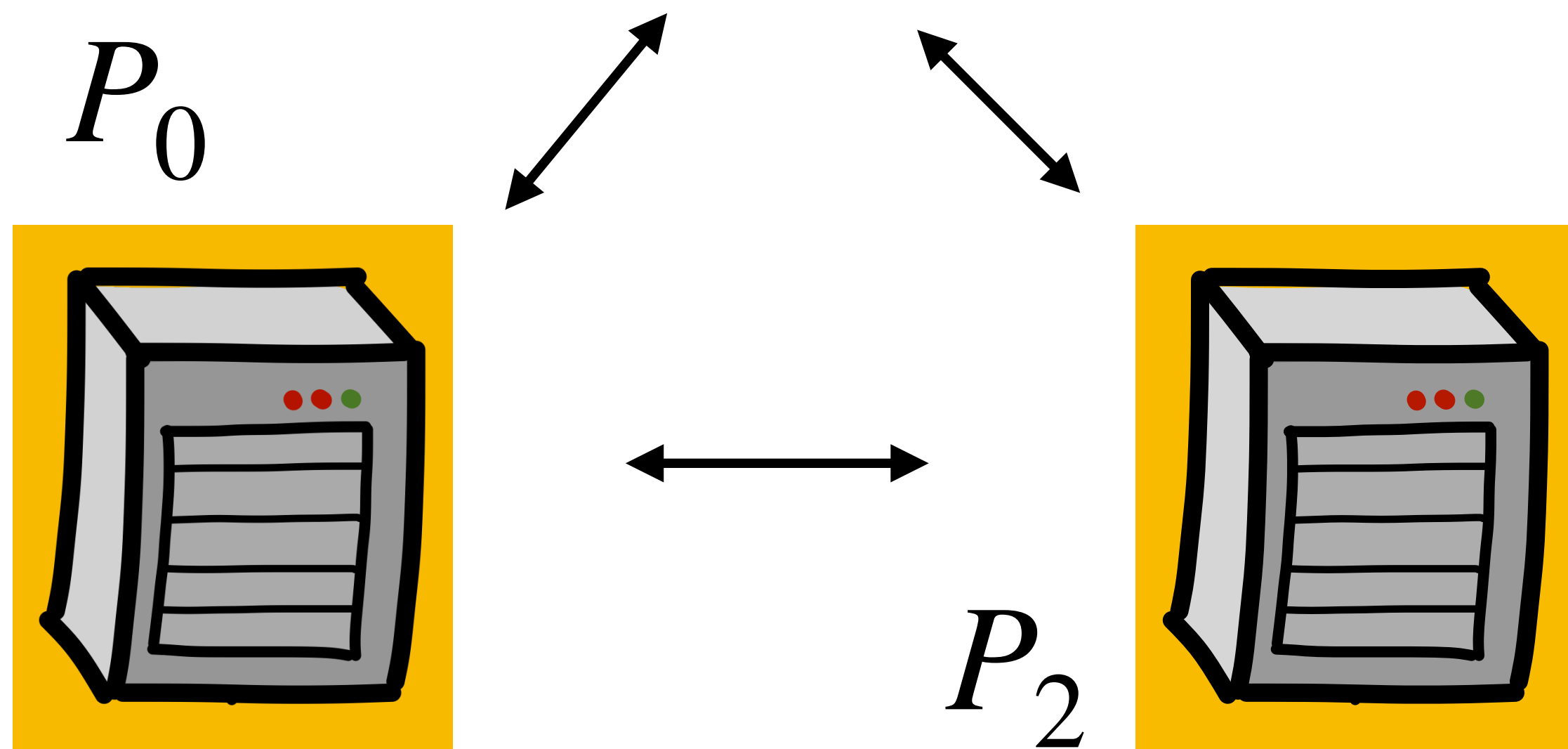
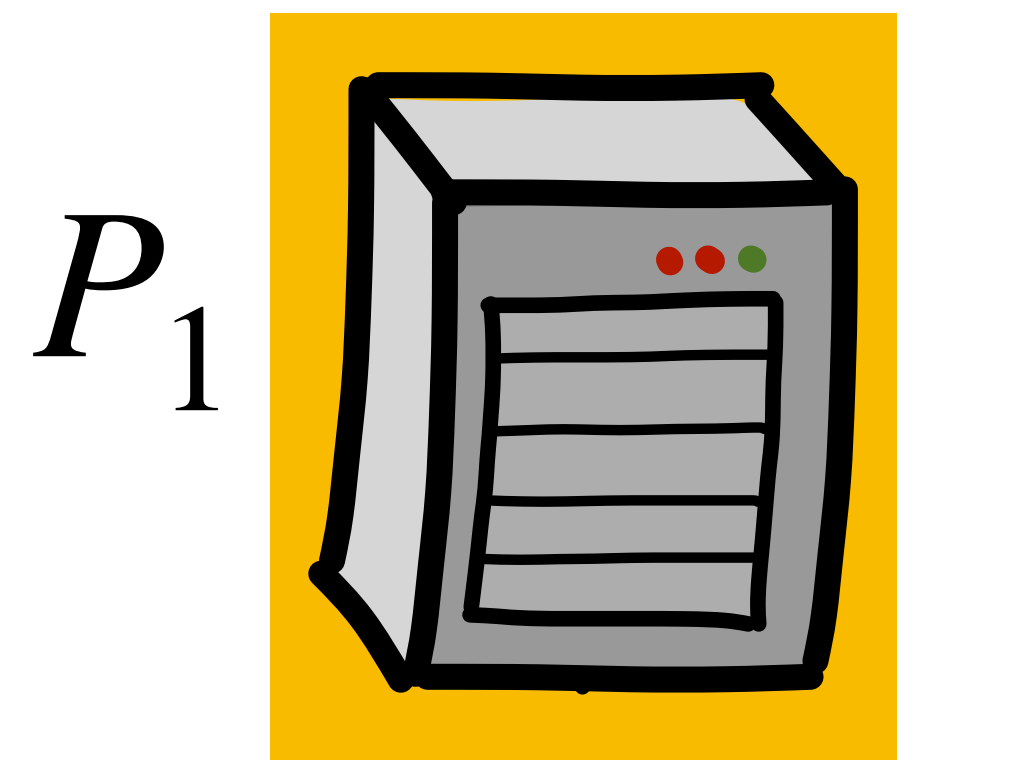
# Broadcast-IA is Impossible with Dishonest Majority

[This work]



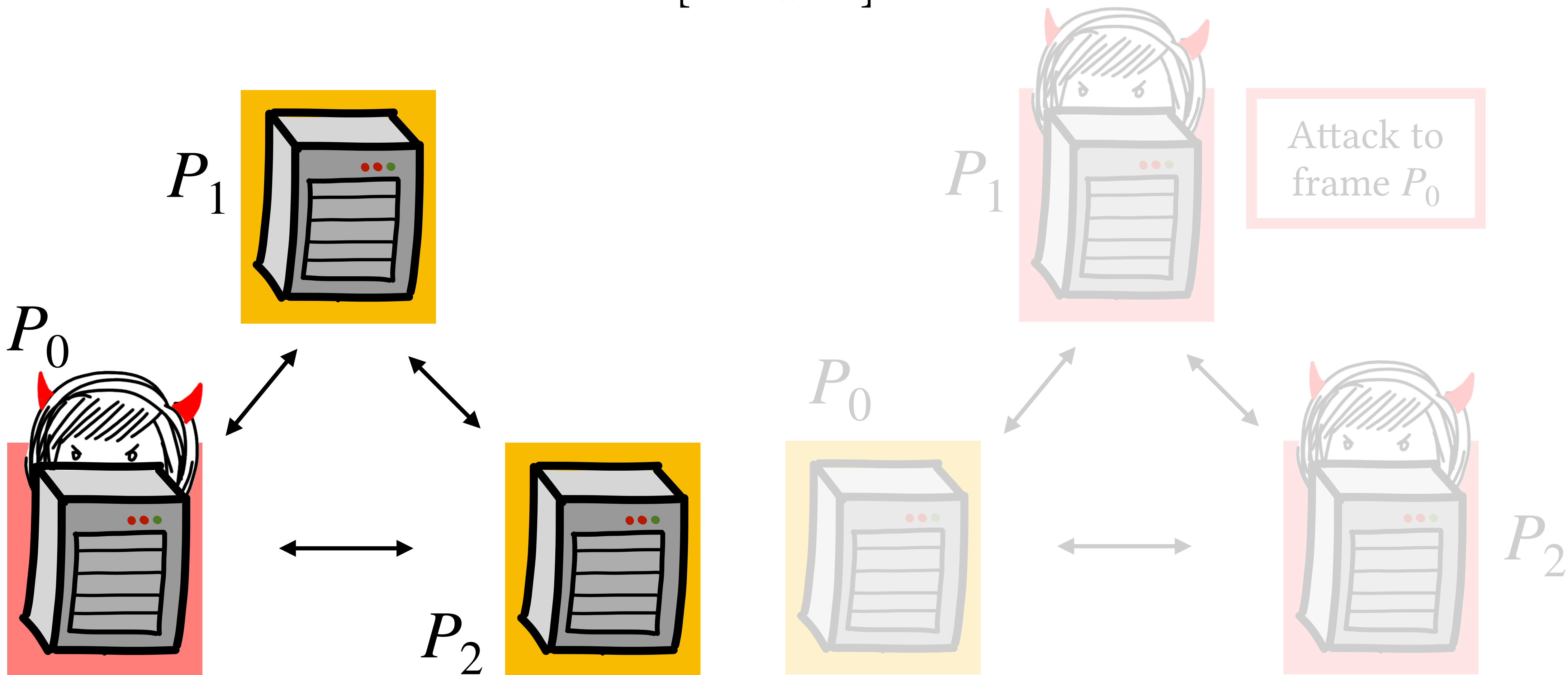
# Broadcast-IA is Impossible with Dishonest Majority

[This work]



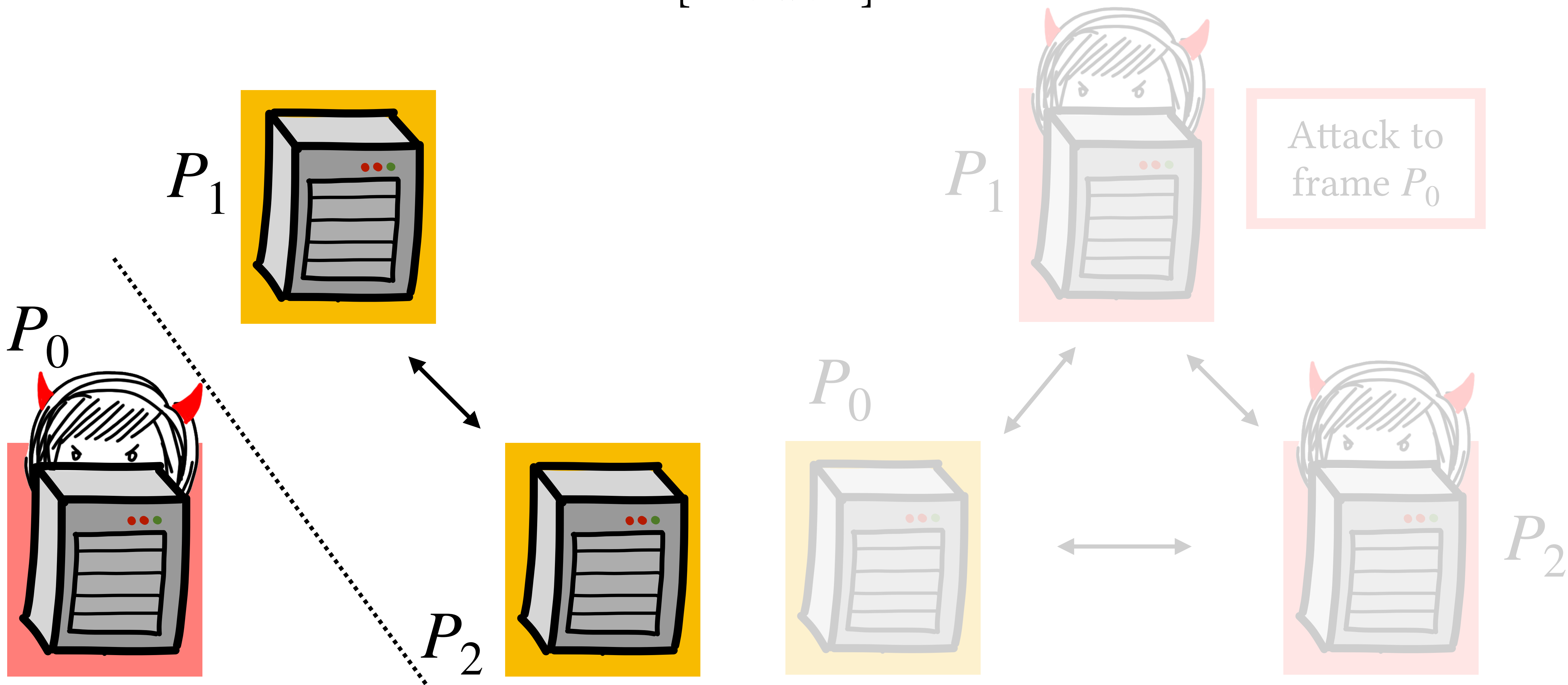
# Broadcast-IA is Impossible with Dishonest Majority

[This work]



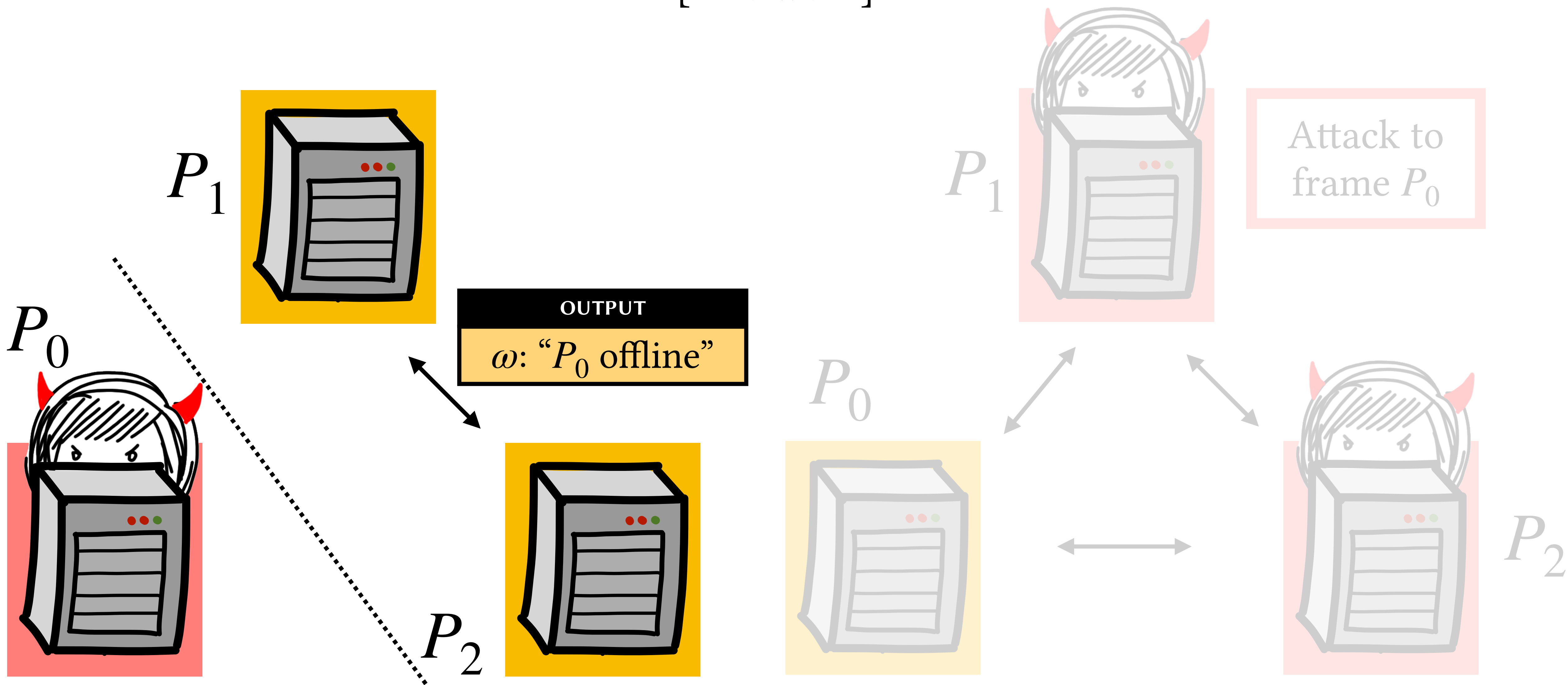
# Broadcast-IA is Impossible with Dishonest Majority

[This work]



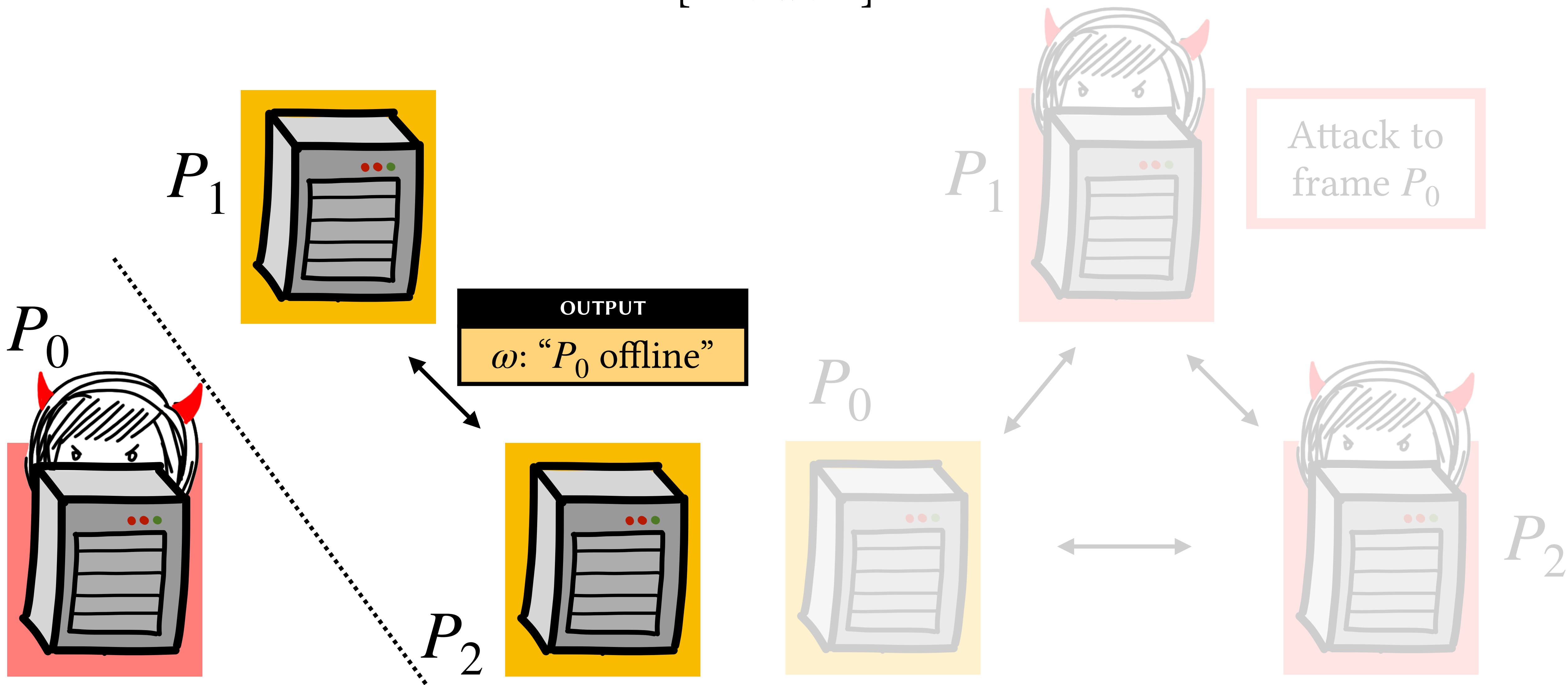
# Broadcast-IA is Impossible with Dishonest Majority

[This work]



# Broadcast-IA is Impossible with Dishonest Majority

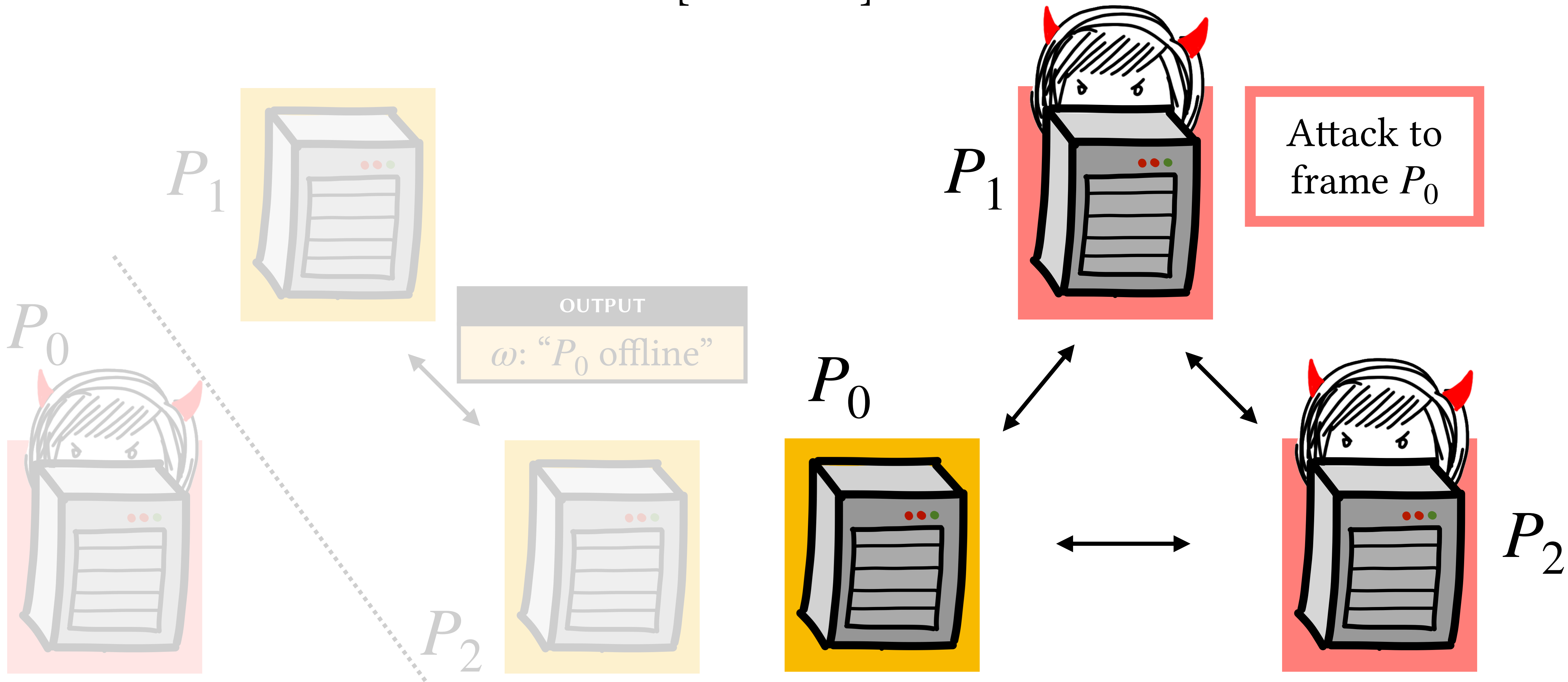
[This work]





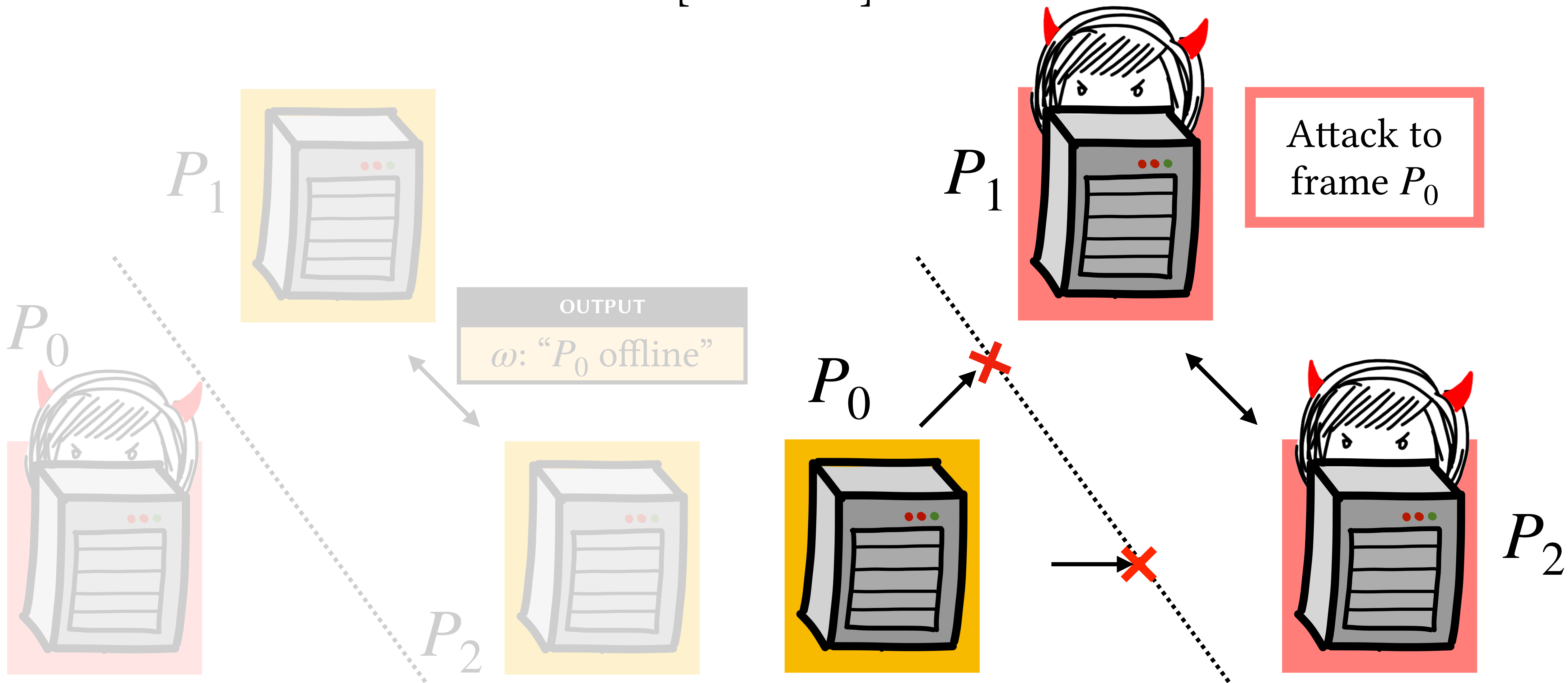
# Broadcast-IA is Impossible with Dishonest Majority

[This work]



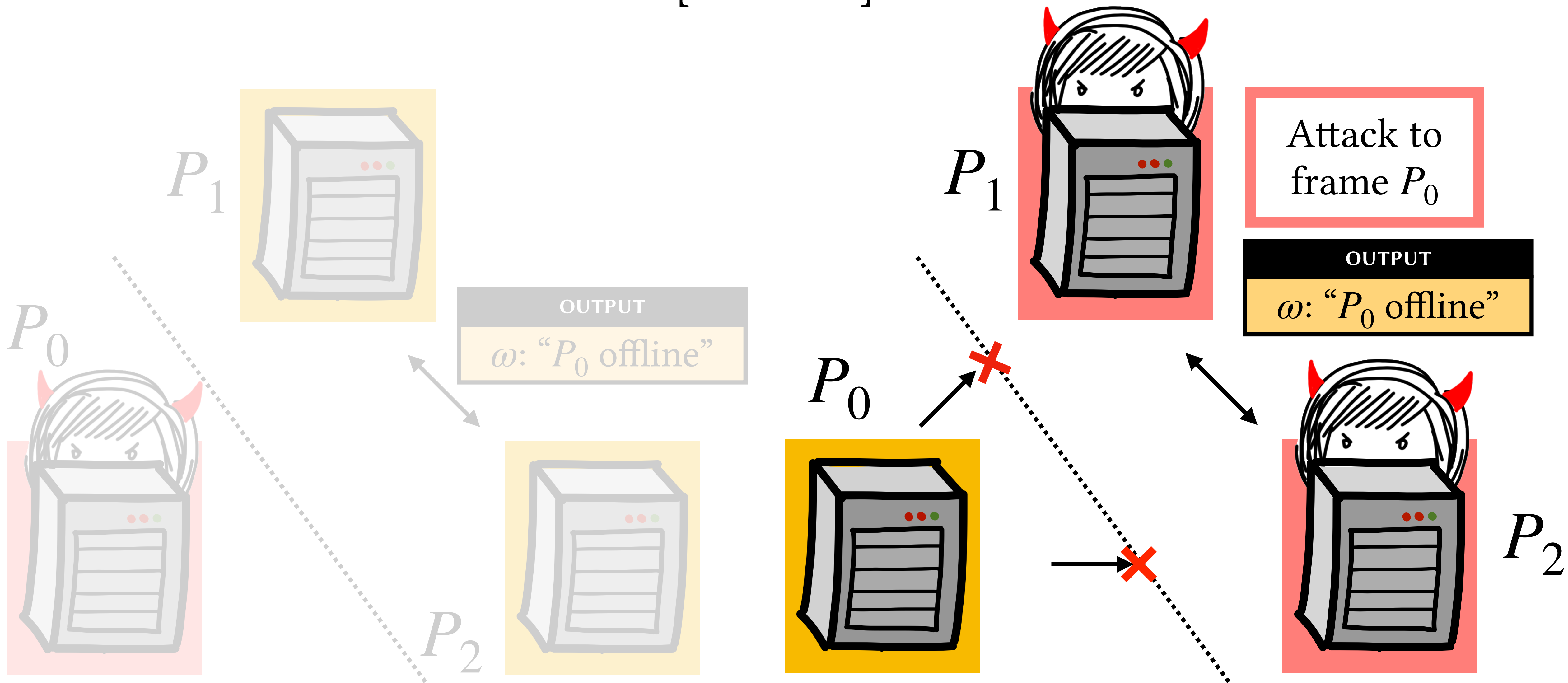
# Broadcast-IA is Impossible with Dishonest Majority

[This work]



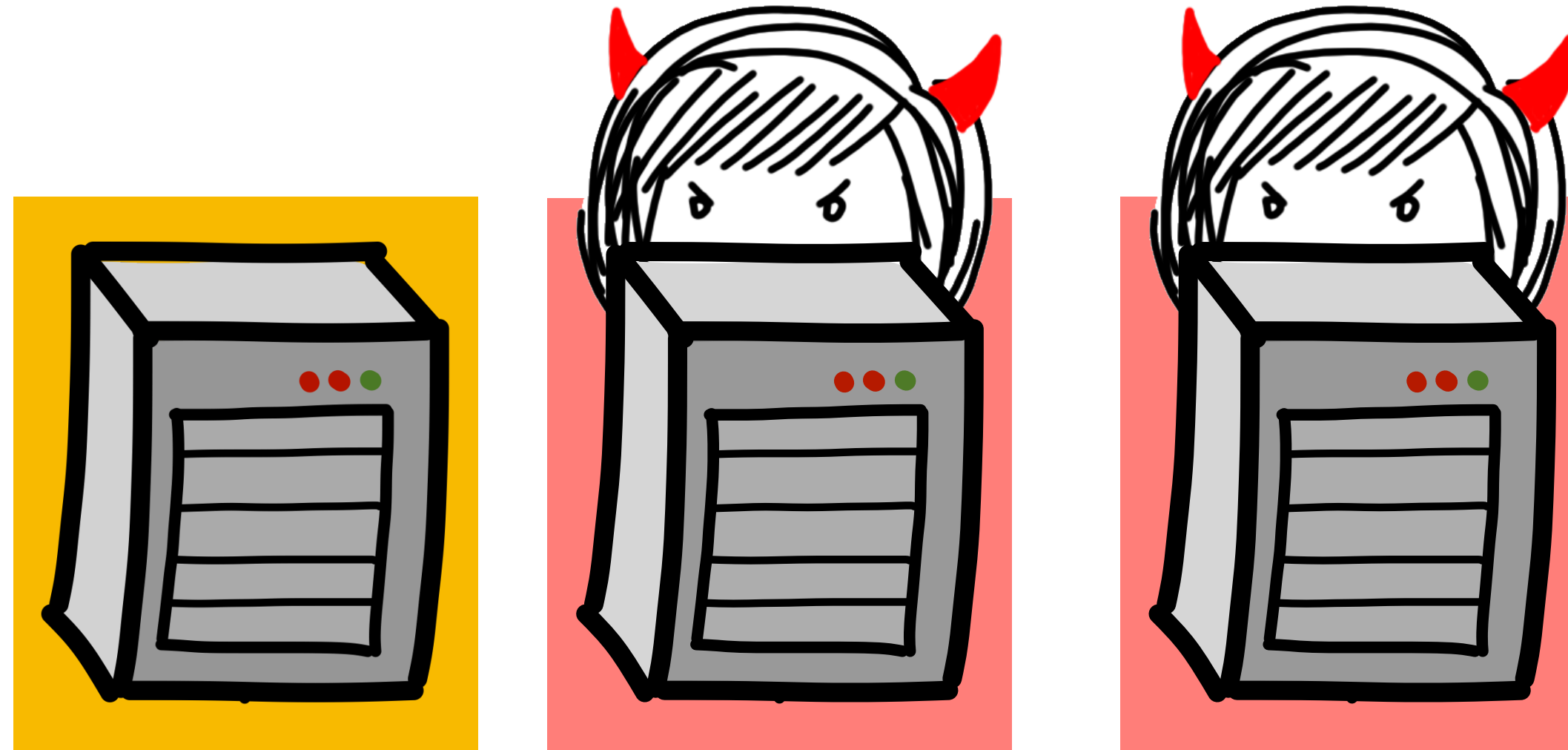
# Broadcast-IA is Impossible with Dishonest Majority

[This work]



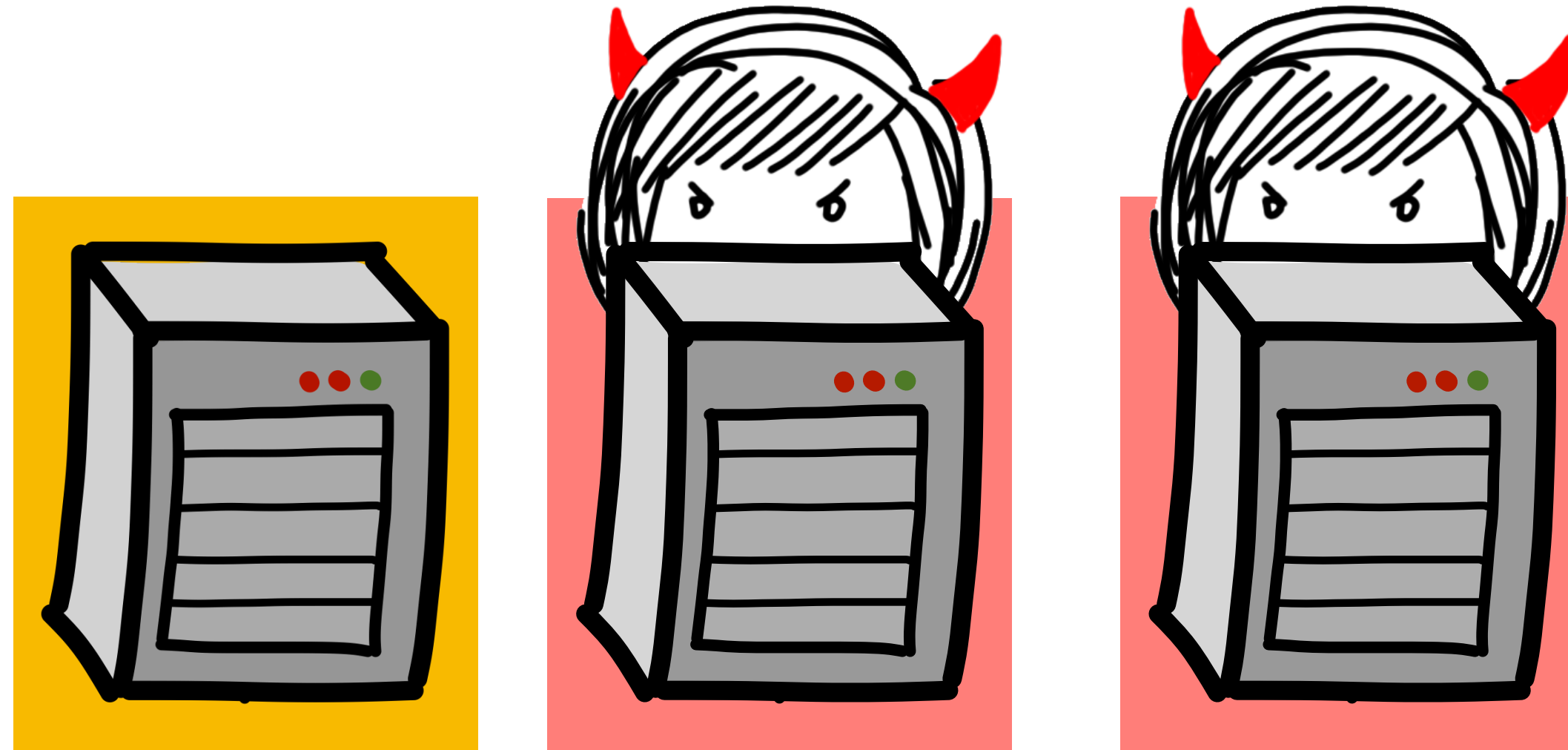
# Broadcast-IA is Impossible with Dishonest Majority

[This work]



# Broadcast-IA is Impossible with Dishonest Majority

[This work]





# Broadcast-IA with Honest Majority

[This work]



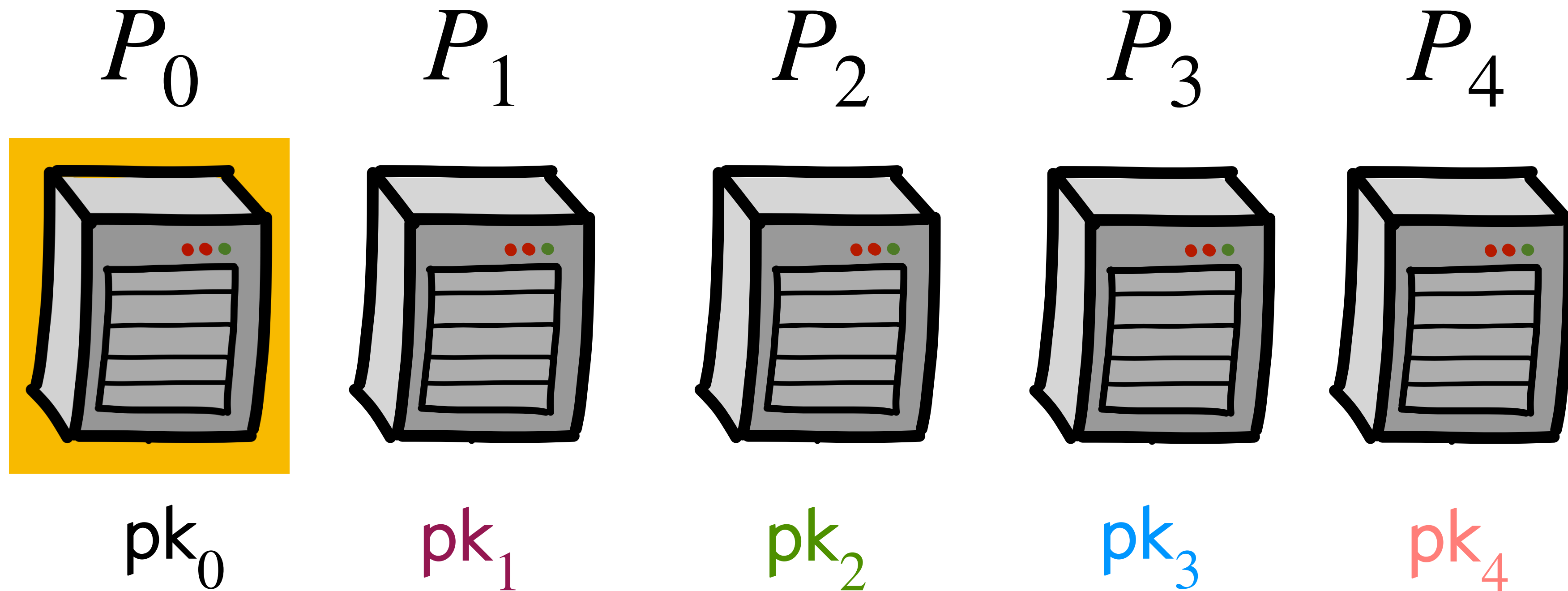
Recall: Global honest majority

Use it proactively



# Broadcast-IA with Honest Majority

[This work]



$P_0$  wishes to broadcast  $m$

# Broadcast-IA with Honest Majority

[This work]

Round 1

⋮

Round 2

⋮

Output

# Broadcast-IA with Honest Majority

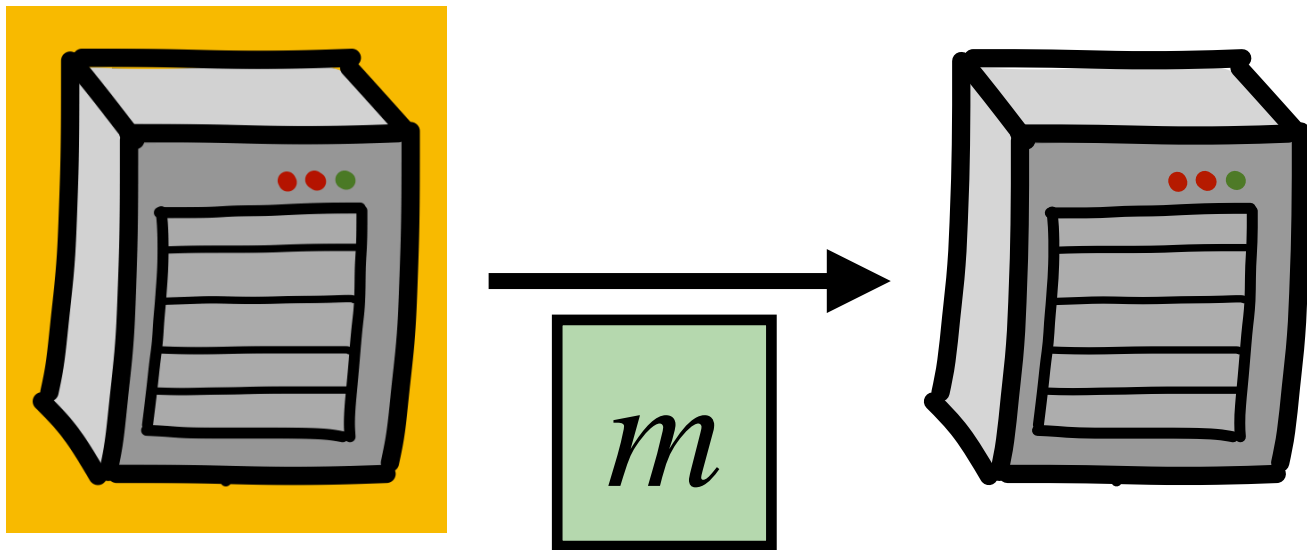
[This work]

Round 1

Sign  $m$ ,  
Send to all

$P_0$

$P_i$



Round 2

...

Output

# Broadcast-IA with Honest Majority

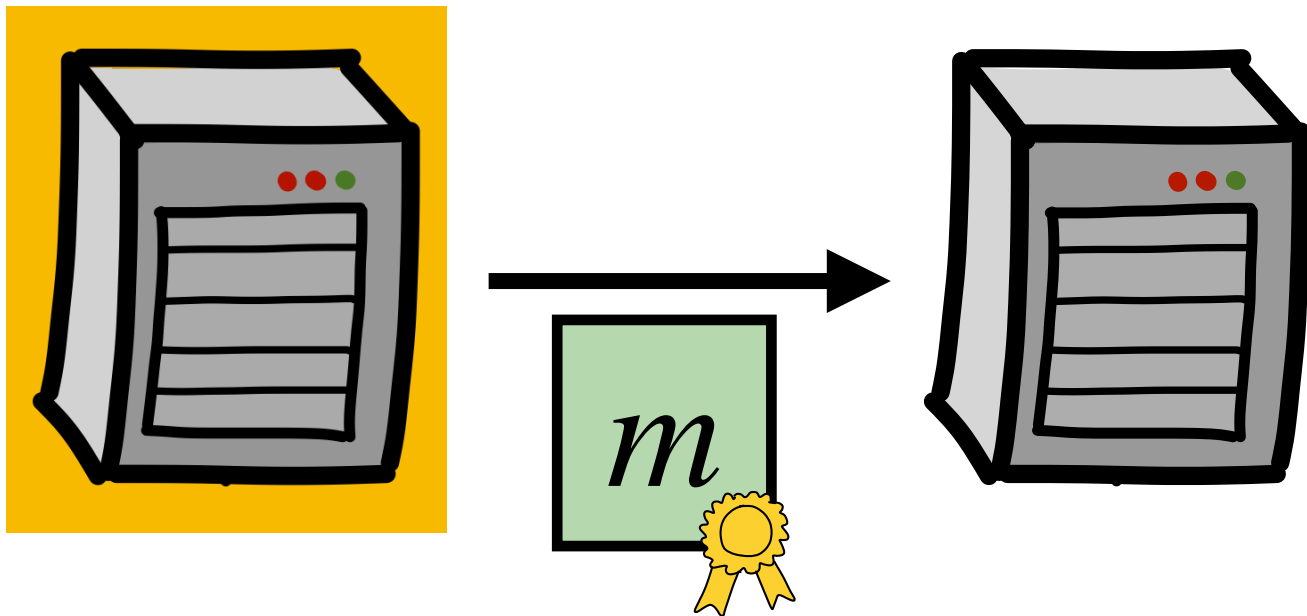
[This work]

Round 1

Sign  $m$ ,  
Send to all

$P_0$

$P_i$

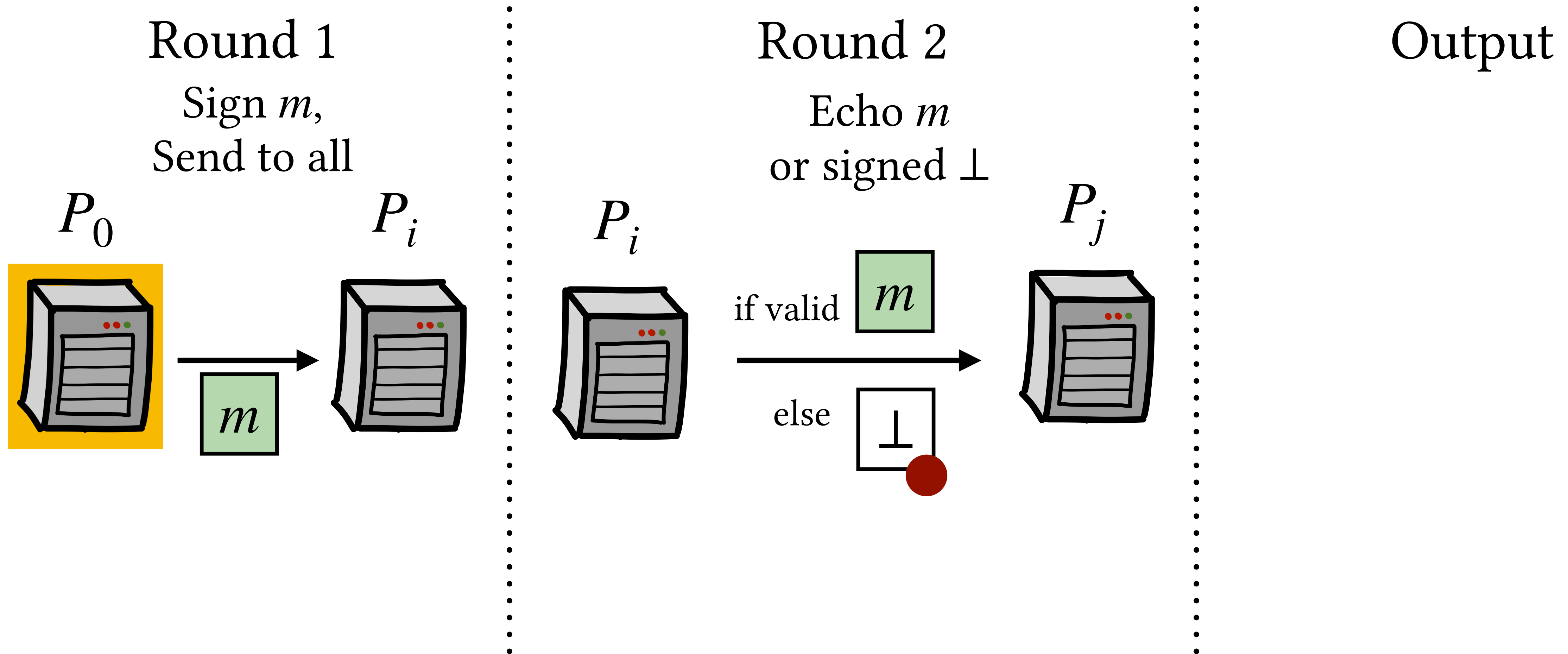


Round 2

Output

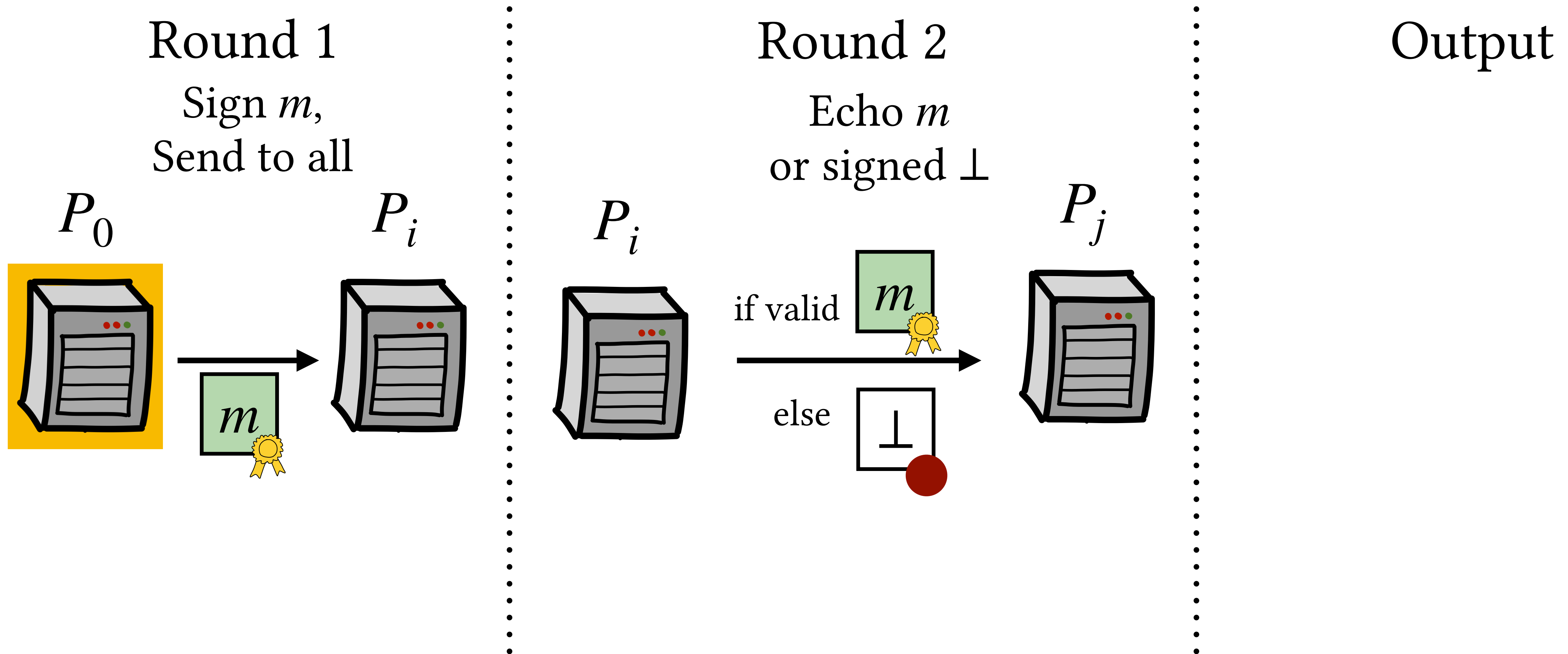
# Broadcast-IA with Honest Majority

[This work]



# Broadcast-IA with Honest Majority

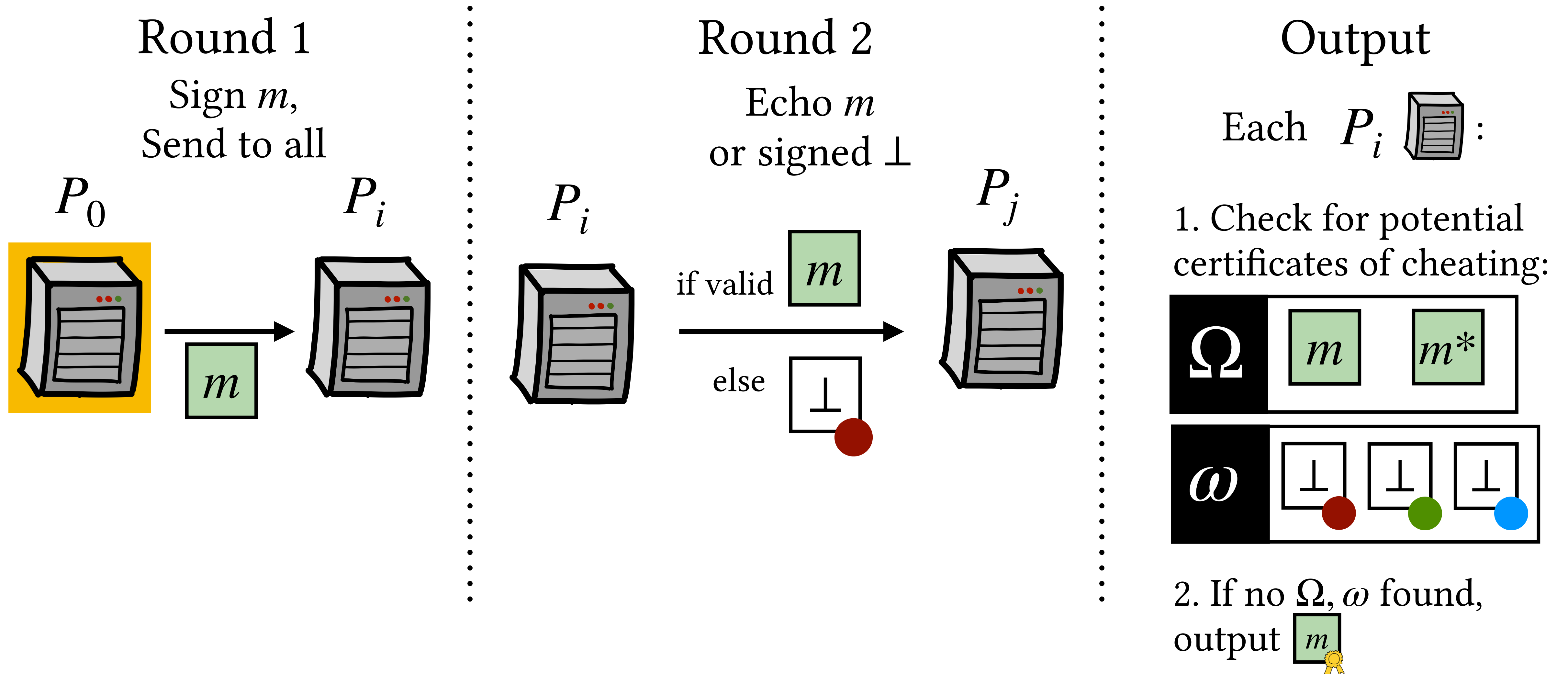
[This work]





# Broadcast-IA with Honest Majority

[This work]

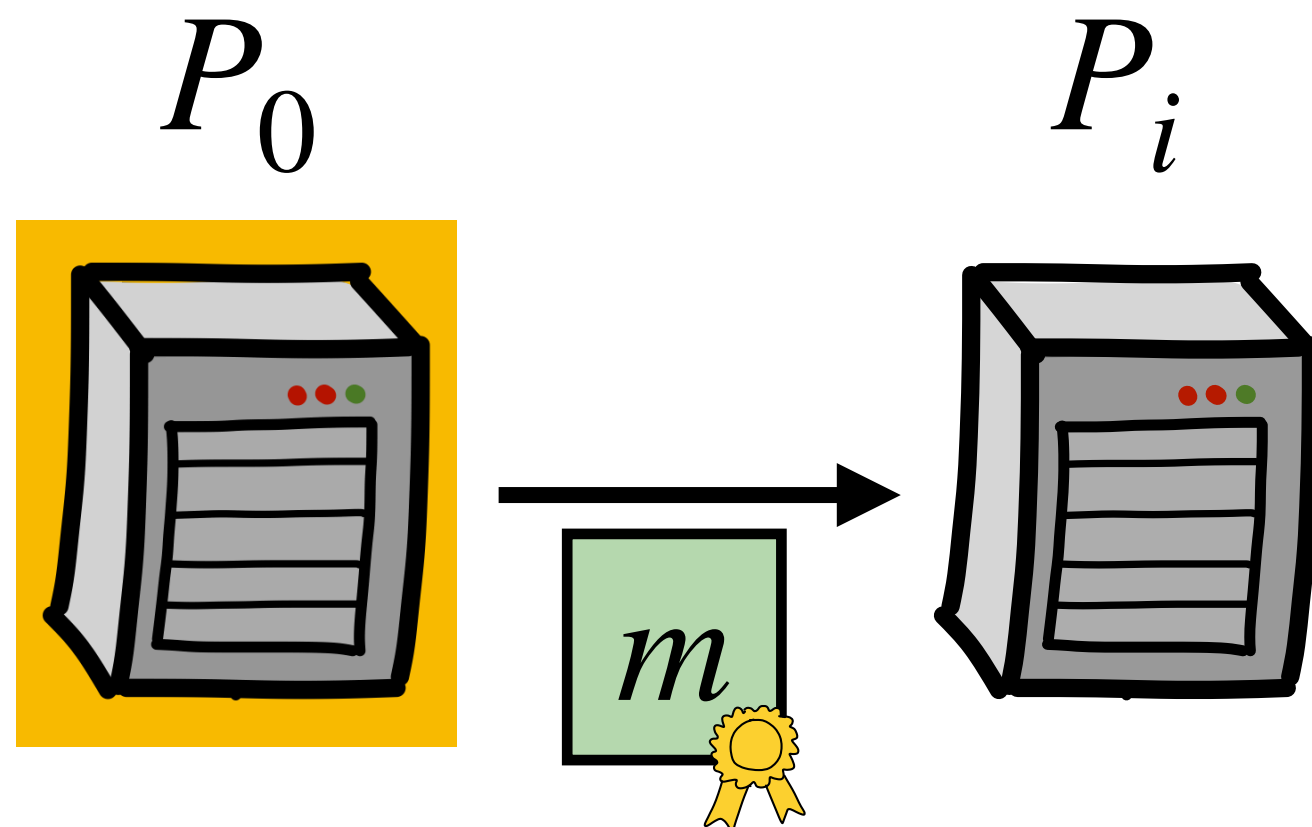


# Broadcast-IA with Honest Majority

[This work]

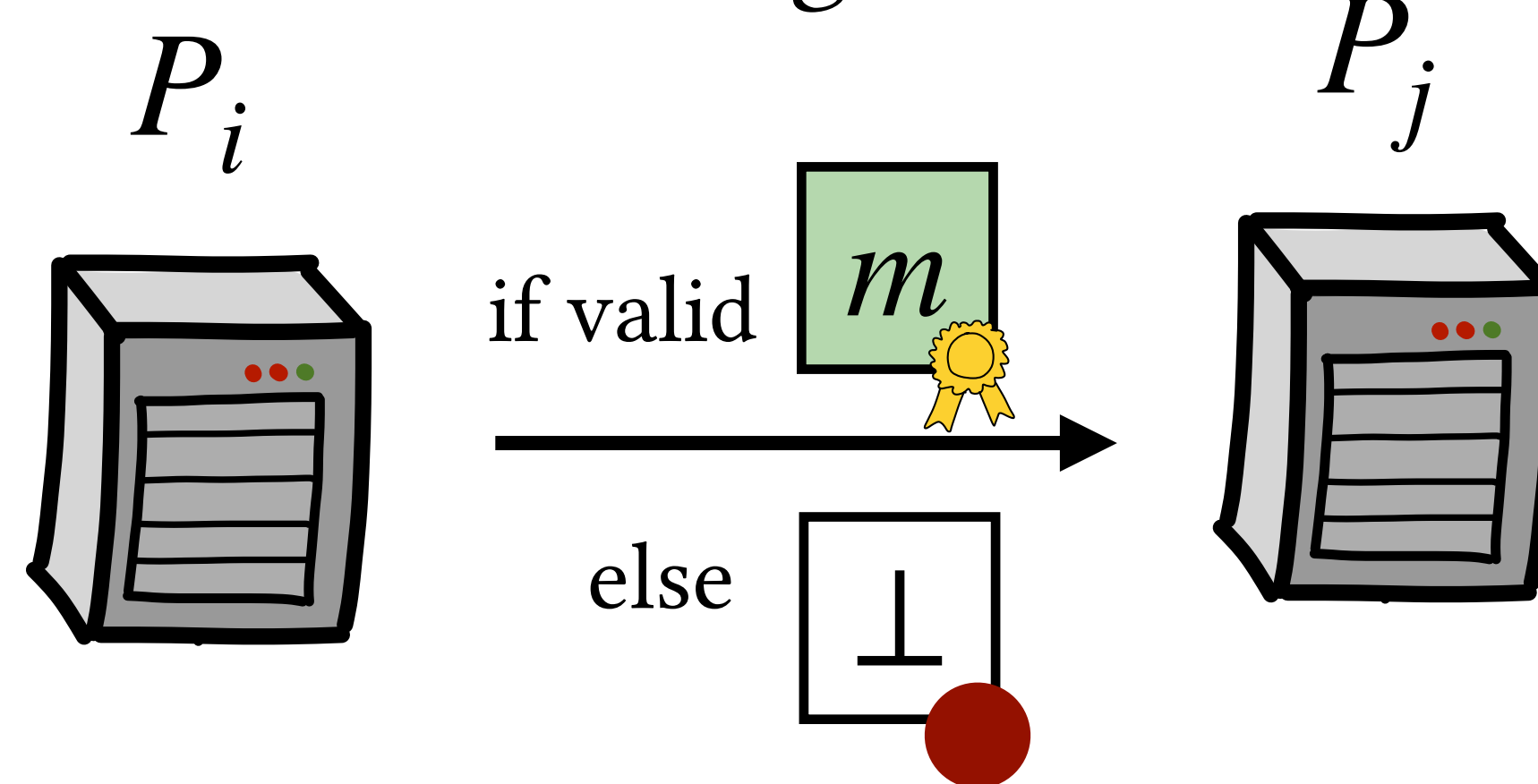
Round 1

Sign  $m$ ,  
Send to all



Round 2

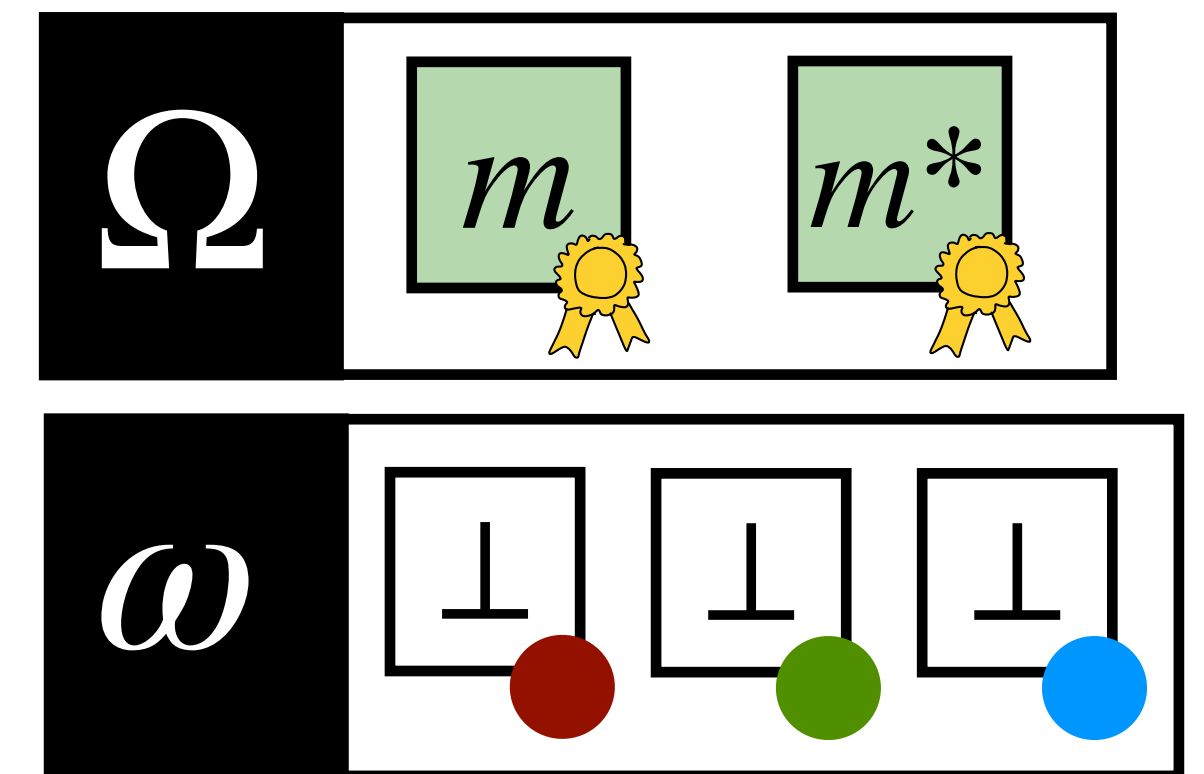
Echo  $m$   
or signed  $\perp$



Output

Each  $P_i$  (server icon):

1. Check for potential certificates of cheating:



2. If no  $\Omega$ ,  $\omega$  found,  
output  $m$  (green box with yellow ribbon)

# Broadcast-IA: Analysis

- **Honest  $P_0$ :** Complete, defamation-free
  - No  $\Omega$ : Will not sending conflicting  $m, m^*$
  - No  $\omega$ : At most  $t$  corrupt parties will echo  $\perp \Rightarrow$  not enough sigs
- **Corrupt  $P_0$ :** Consistent
  - If any honest parties receive  $m, m^* \Rightarrow$  yields  $\Omega$
  - If  $m$  withheld from *all* honest parties  $\Rightarrow$  yields  $\omega$
  - Send  $m$  to any honest party  $\Rightarrow m$  committed as output
- Notes on output  $m$ :
  1. Accompanied by  $\text{sig}(m)$  from  $P_0$ : proves  $P_0$  sent  $m$  to  $P_i$
  2.  $P_i$  producing  $\text{sig}(m)$  DOES NOT prove that some  $P_j$  also output  $m$

# Signing from ECDSA Tuples

(Recall from Jack's talk)

[Abram Nof Orlandi Scholl Shlomovits 22]

$$[sk] \quad [k] \quad [\phi] \quad [\phi \cdot k] \quad [\phi \cdot sk]$$

Round 1

Round 2

Establish  $R = [k] \cdot G$

Round 3

Reveal  $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot sk]$   
and  $\beta = [\phi \cdot k]$

Output  $(s = \alpha/\beta, R)$

# Sampling ECDSA Tuples

Round 1

Round 2

Establish  $R = [k] \cdot G$

$[sk]$   $[k]$   $[\phi]$   $[\phi \cdot k]$   $[\phi \cdot sk]$

# Sampling ECDSA Tuples

Random string:  $G_1, G_2 \in \mathbb{G}$  unknown DLog

BC-IA 1

Pedersen VSS: public deg- $t$  poly  $C \in \mathbb{G}[X]$   
Each  $P_i$  (should) hold  $f(i), h(i) \in \mathbb{Z}_q$  s.t.

$$f(i)G_1 + h(i)G_2 = C(i)$$

BC-IA 2

if  $P_i$  didn't get output, b'casts proof of cheat

DKG: Prise apart  $f$  and  $h$ : use  $f$ , discard  $h$   
 $P_i$  b'casts  $F(i) = f(i)G_1, H(i) = h(i)G_2$  and PoK

$[sk]$     $[k]$     $[\phi]$     $[\phi \cdot k]$     $[\phi \cdot sk]$



# Sampling ECDSA Tuples

Random string:  $G_1, G_2 \in \mathbb{G}$  unknown DLog

BC-IA 1

Pedersen VSS: public deg- $t$  poly  $C \in \mathbb{G}[X]$   
Each  $P_i$  (should) hold  $f(i), h(i) \in \mathbb{Z}_q$  s.t.

$$f(i)G_1 + h(i)G_2 = C(i)$$

BC-IA 2

if  $P_i$  didn't get output, b'casts proof of cheat

DKG: Prise apart  $f$  and  $h$ : use  $f$ , discard  $h$   
 $P_i$  b'casts  $F(i) = f(i)G_1, H(i) = h(i)G_2$  and PoK

$[sk]$   $[k]$

DKG

$[\phi]$

VSS

$[\phi \cdot k]$   $[\phi \cdot sk]$

Local mult + rerandomize

BC-IA 2

if  $P_i$  didn't get output, b'casts proof of cheat

DKG: Prise apart  $f$  and  $h$ : use  $f$ , discard  $h$   
 $P_i$  b'casts  $F(i) = f(i)G_1$ ,  $H(i) = h(i)G_2$  and PoK

$[sk]$   $[k]$

DKG

$[\phi]$

VSS

$[\phi \cdot k]$   $[\phi \cdot sk]$

Local mult + rerandomize

BC-IA 3

Reveal  $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot sk]$   
and  $\beta = [\phi \cdot k]$

Output ( $s = \alpha/\beta, R$ )

BC-IA 2

if  $P_i$  didn't get output, b'casts proof of cheat

DKG: Prise apart  $f$  and  $h$ : use  $f$ , discard  $h$   
 $P_i$  b'casts  $F(i) = f(i)G_1$ ,  $H(i) = h(i)G_2$  and PoK

$[sk]$   $[k]$

DKG

$[\phi]$

VSS

$[\phi \cdot k]$   $[\phi \cdot sk]$

Local mult + rerandomize

$P_i$ 's publicly  
committed share

BC-IA 3

Reveal  $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot sk]$   
and  $\beta = [\phi \cdot k]$

Output ( $s = \alpha/\beta, R$ )

BC-IA 2

if  $P_i$  didn't get output, b'casts proof of cheat

DKG: Prise apart  $f$  and  $h$ : use  $f$ , discard  $h$   
 $P_i$  b'casts  $F(i) = f(i)G_1$ ,  $H(i) = h(i)G_2$  and PoK

$[sk]$   $[k]$

DKG

$pk_i, R_i$

$[\phi]$

VSS

$[\phi \cdot k]$   $[\phi \cdot sk]$

Local mult + rerandomize

$P_i$ 's publicly  
committed share

BC-IA 3

Reveal  $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot sk]$   
and  $\beta = [\phi \cdot k]$

Output ( $s = \alpha/\beta, R$ )

BC-IA 2

if  $P_i$  didn't get output, b'casts proof of cheat

DKG: Prise apart  $f$  and  $h$ : use  $f$ , discard  $h$   
 $P_i$  b'casts  $F(i) = f(i)G_1$ ,  $H(i) = h(i)G_2$  and PoK

$[sk]$   $[k]$

DKG

$pk_i, R_i$

$[\phi]$

VSS

$Com(\phi_i)$

$[\phi \cdot k]$   $[\phi \cdot sk]$

Local mult + rerandomize

$P_i$ 's publicly  
committed share

BC-IA 3

Reveal  $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot sk]$   
and  $\beta = [\phi \cdot k]$

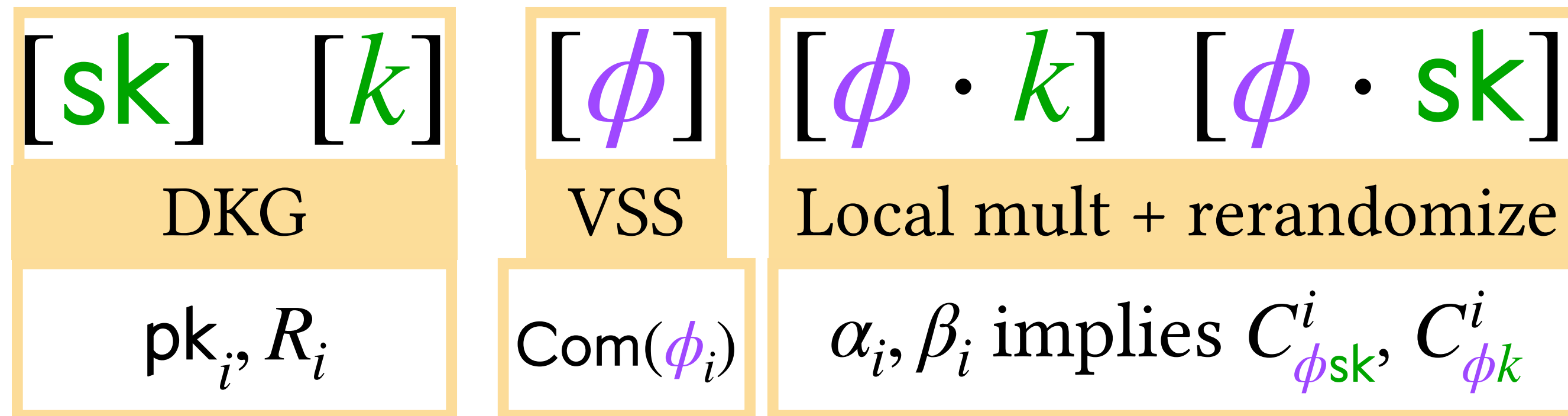
Output ( $s = \alpha/\beta, R$ )

## BC-IA 2

if  $P_i$  didn't get output, b'casts proof of cheat

DKG: Prise apart  $f$  and  $h$ : use  $f$ , discard  $h$   
 $P_i$  b'casts  $F(i) = f(i)G_1$ ,  $H(i) = h(i)G_2$  and PoK

$P_i$ 's publicly  
committed share



## BC-IA 3

Reveal  $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot sk]$   
 and  $\beta = [\phi \cdot k]$

Output ( $s = \alpha/\beta, R$ )

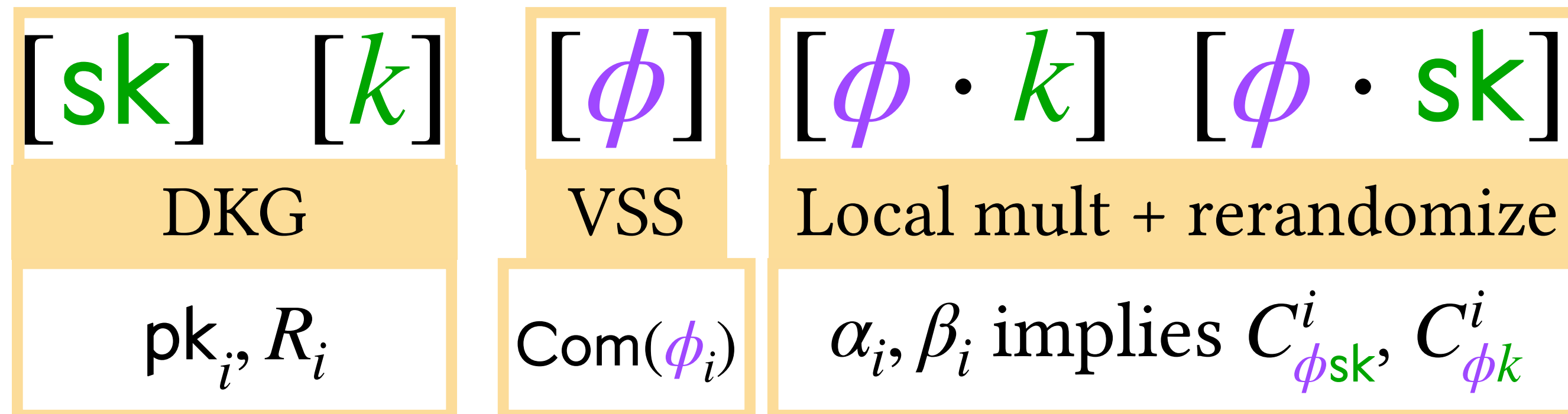


## BC-IA 2

if  $P_i$  didn't get output, b'casts proof of cheat

DKG: Prise apart  $f$  and  $h$ : use  $f$ , discard  $h$   
 $P_i$  b'casts  $F(i) = f(i)G_1$ ,  $H(i) = h(i)G_2$  and PoK

$P_i$ 's publicly  
committed share



## BC-IA 3

Reveal  $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot sk]$   
 and  $\beta = [\phi \cdot k]$

+ NIZK proving  
 $pk_i, R_i, Com(\phi_i),$   
 $C_{\phi k}^i, C_{\phi sk}^i$

Output ( $s = \alpha/\beta, R$ )

# Efficiency

- Envisioned mode of operation:
  - Run [DKLs23] (sec w. abort) by default
  - Fall back to this protocol if too many aborts observed
- Worst case execution path most relevant to measuring efficiency
  - $(t, n) = (10, 21)$  : ~500ms compute time on standard hardware
  - Relative to dishonest majority
  - noticeably slower than (s.w.a.) OT-based ECDSA [DKLs23]
  - order of magnitude faster than Paillier-based ECDSA-IA [CGGMP20]
- Actual worst-case performance depends on network conditions
  - Up to  $6 \times$  Network Timeout

# In Conclusion

- Cheater identification requires some form of broadcast
  - Broadcast protocols are expensive
  - Tempting to resort to heuristics, external channels
- We define Broadcast-IA to certify cheaters: silent parties and protocol deviations
  - Prove *impossible* w. dishonest majority
  - 2-round  $t < n/2$  construction over p2p channels (synchrony + PKI)
- We build VSS-IA  $\rightarrow$  DKG-IA  $\rightarrow$  ECDSA-IA with  $t < n/2$ 
  - Leverage global honest majority

## Thanks!

eprint coming soon, (pre)preprint on [ykondi.net](https://ykondi.net)

Thanks **Eysa Lee** for

