Universal Composition

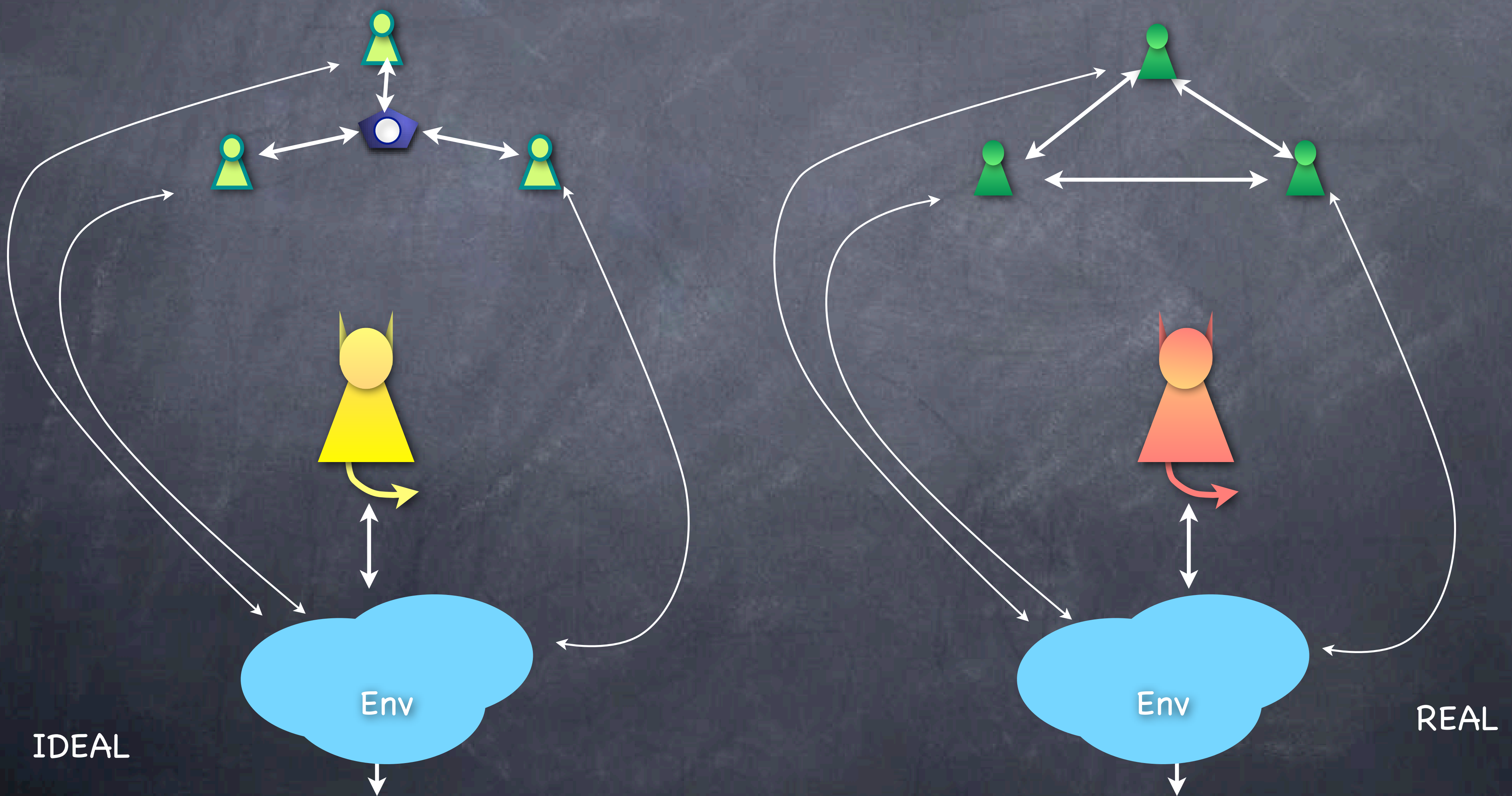
Universal Composition

Lecture 17

And the GMW-Paradigm for MPC Protocols

Security

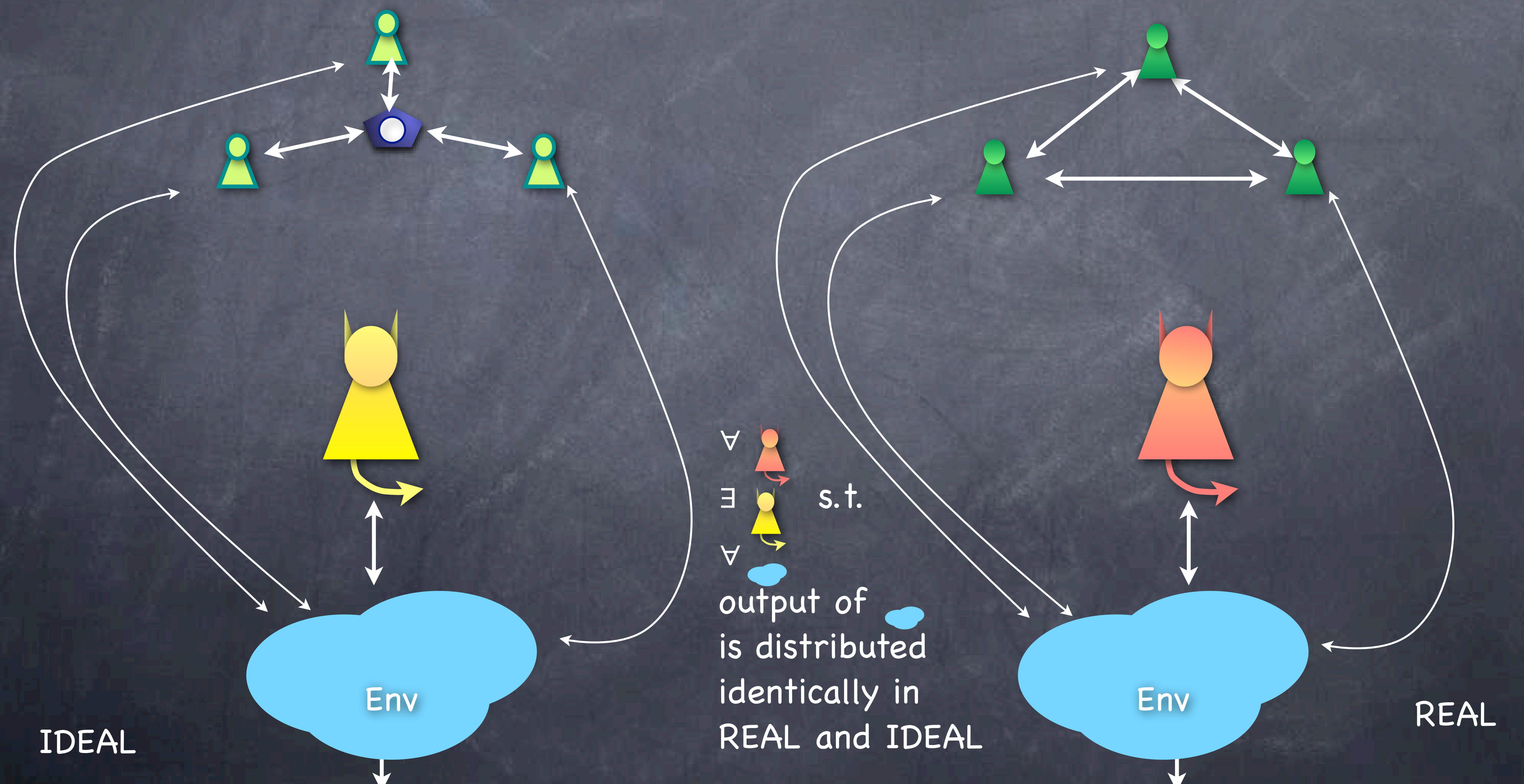
REAL (with protocol) is as secure as IDEAL (with functionality) if:



RECALL

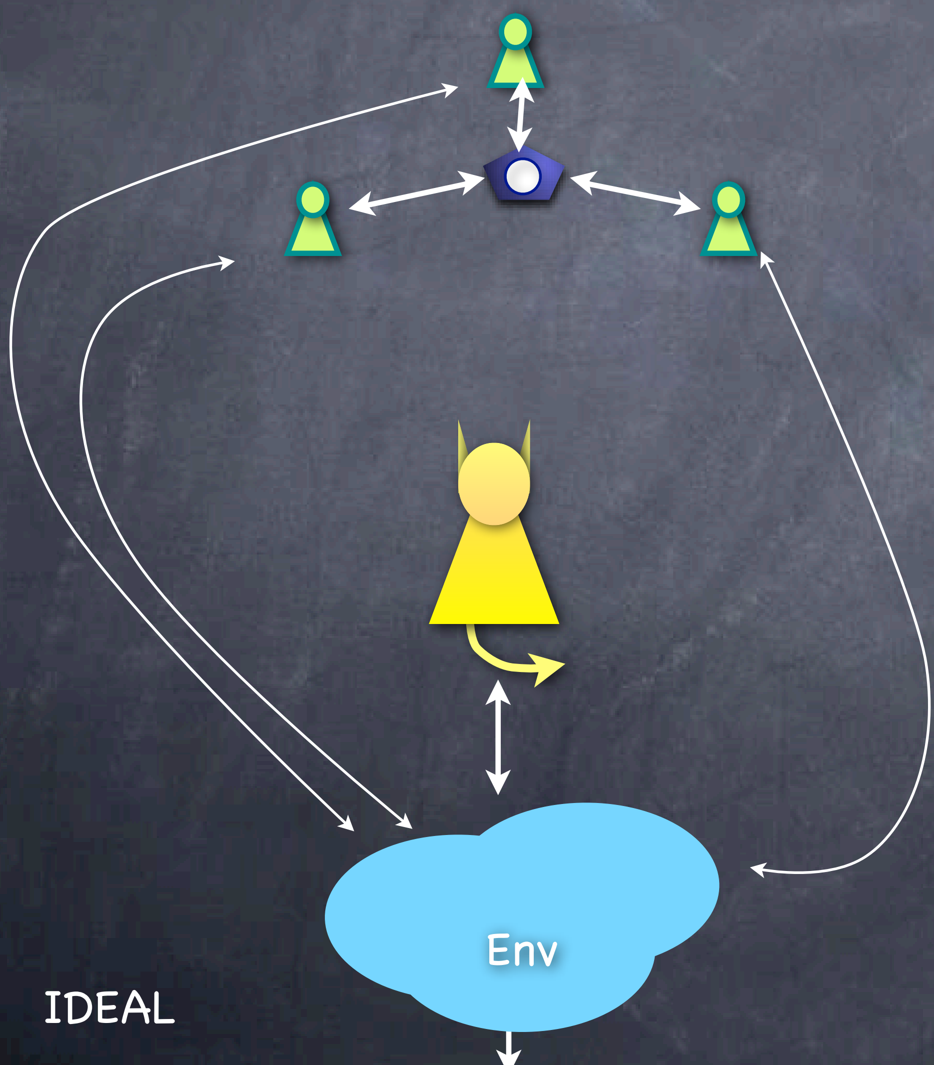
Security

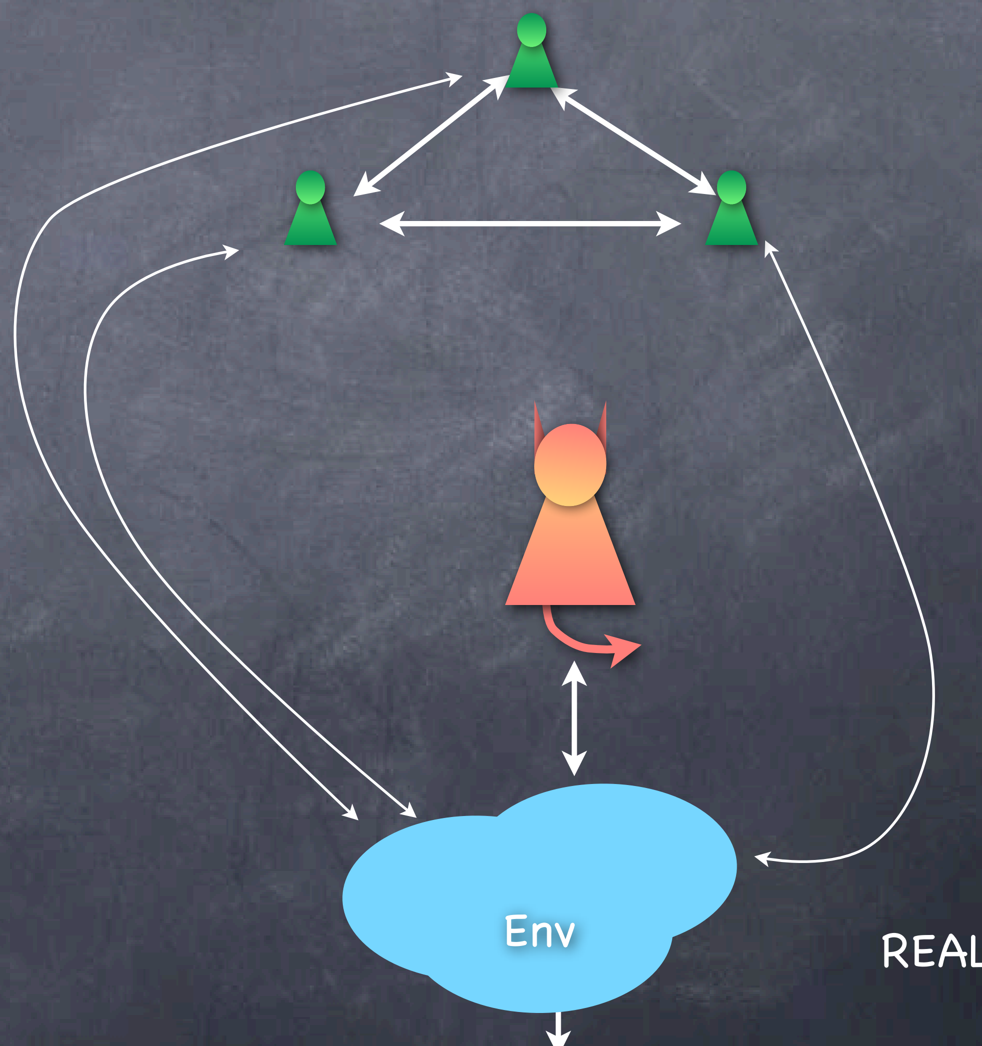
REAL (with protocol) is as secure as IDEAL (with functionality) if:



Security of Composed Systems

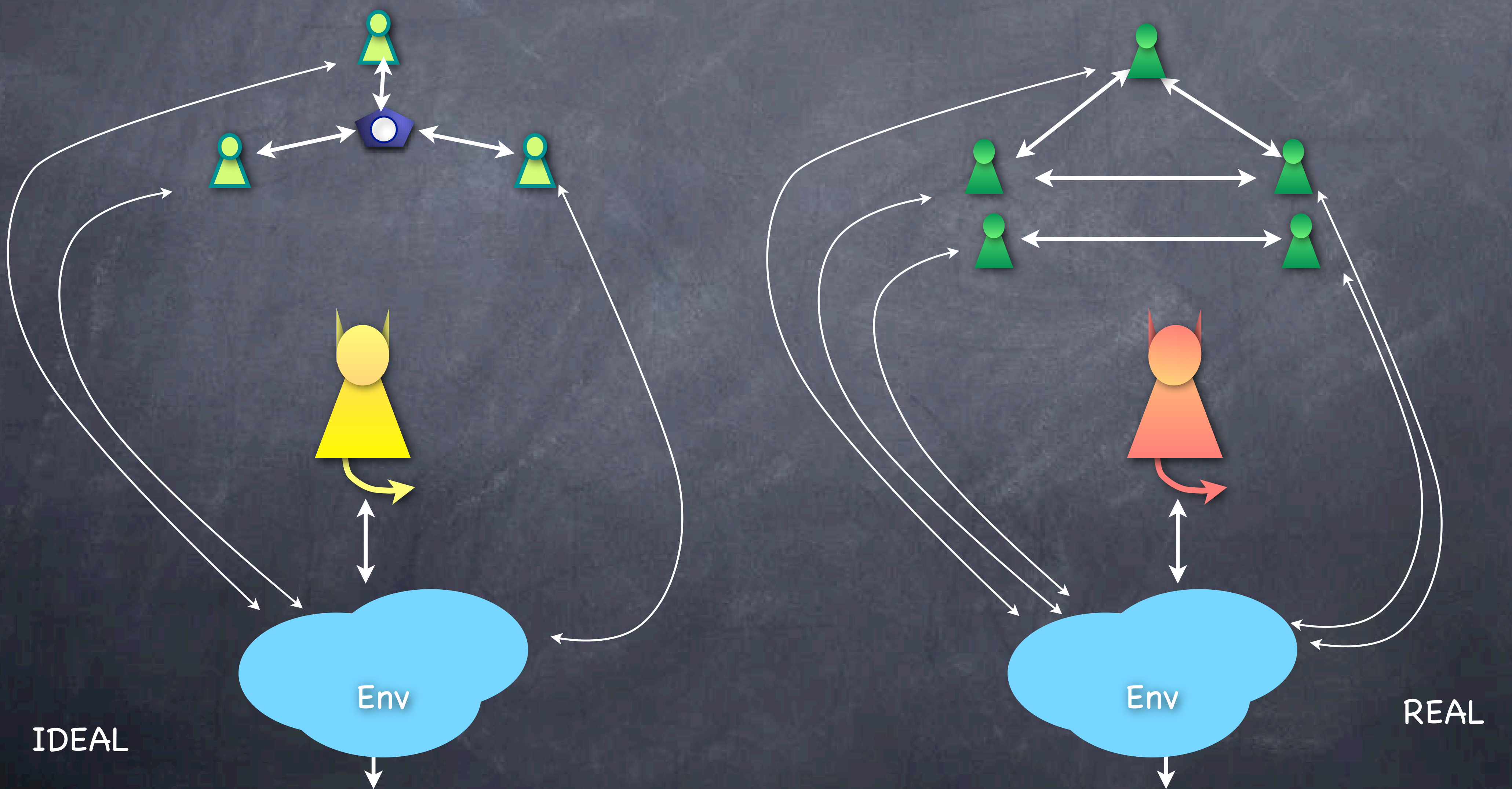
- Extend to allow a "composed system" with multiple functionalities





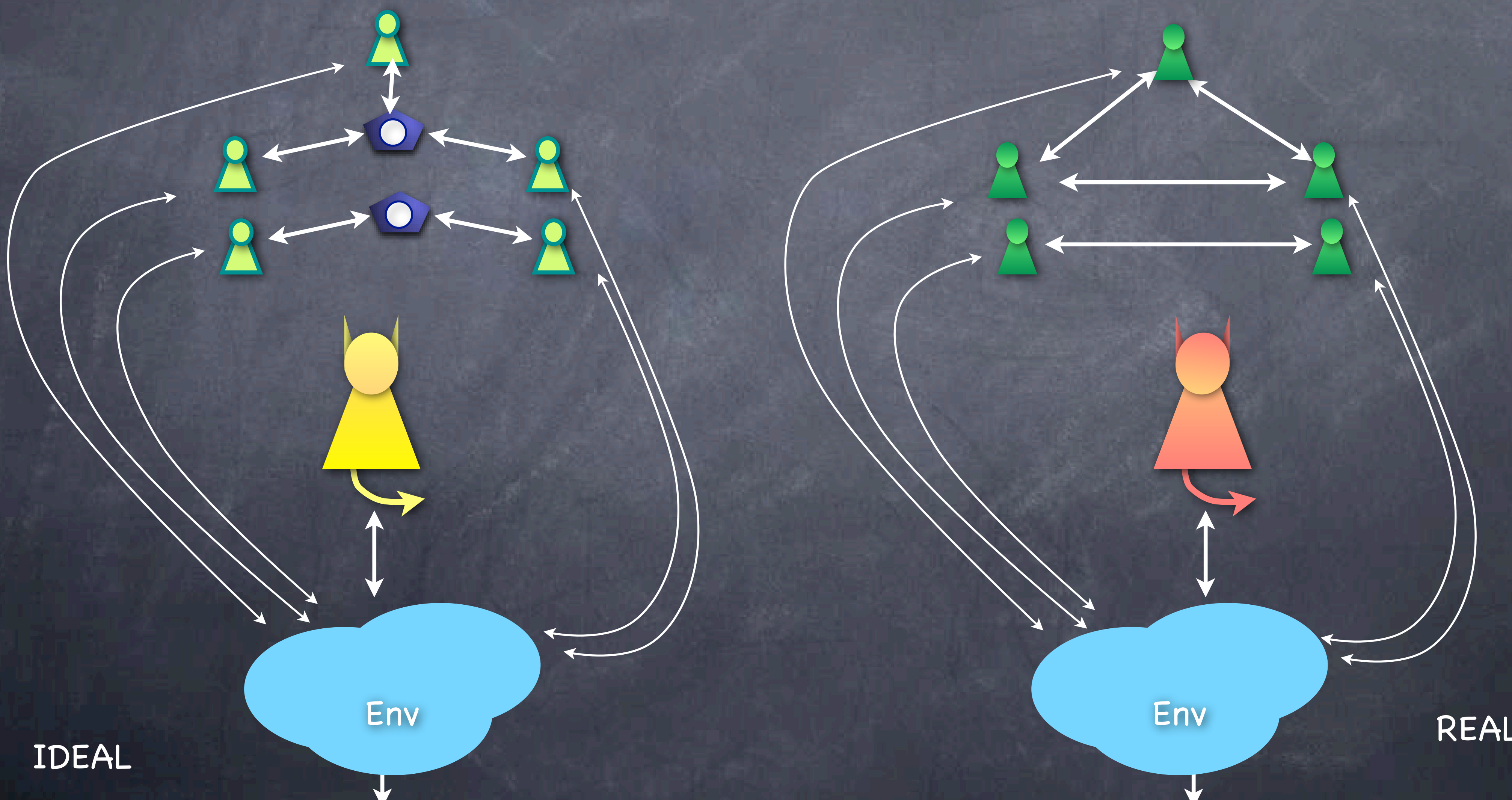
Security of Composed Systems

- Extend to allow a "composed system" with multiple functionalities



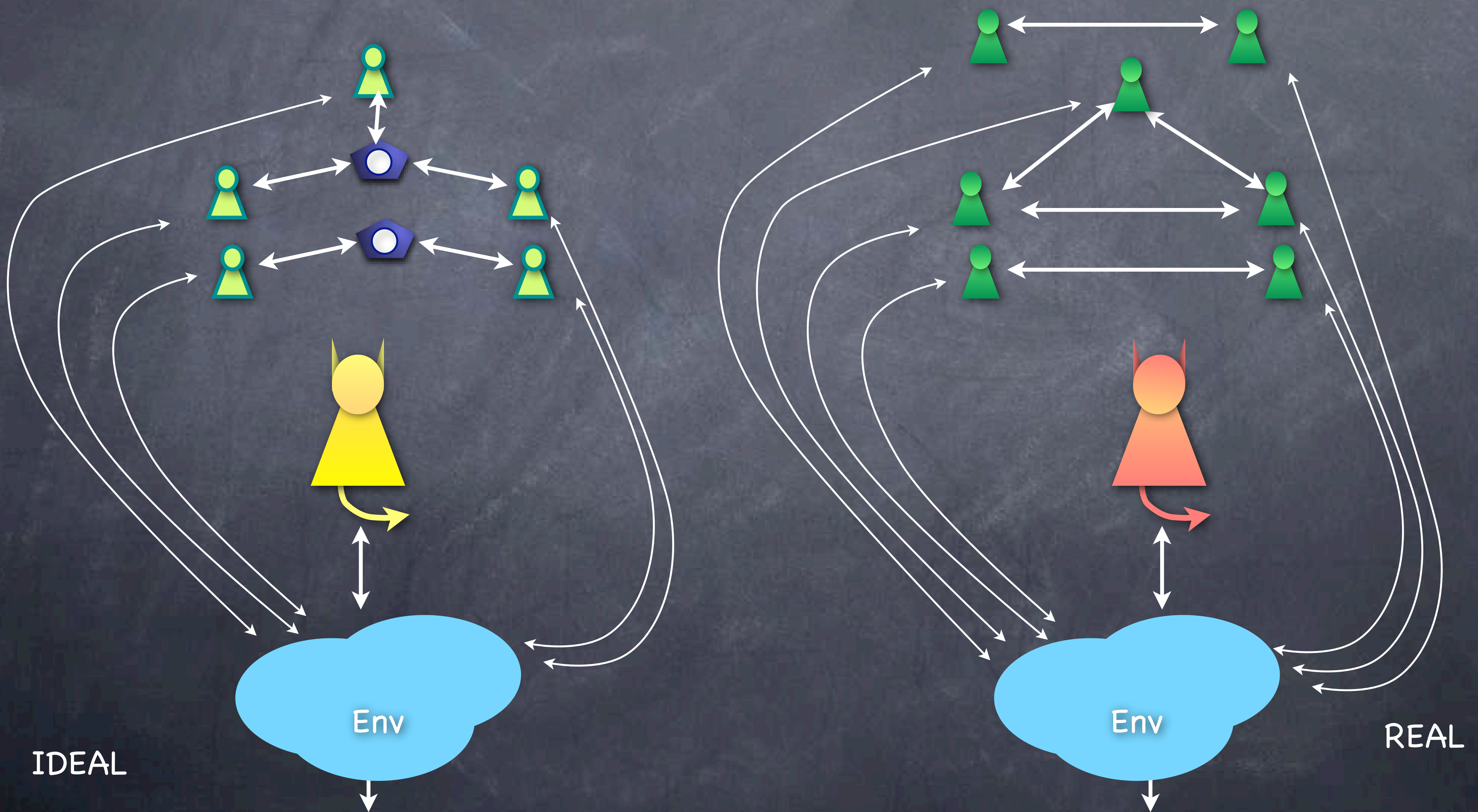
Security of Composed Systems

- Extend to allow a "composed system" with multiple functionalities



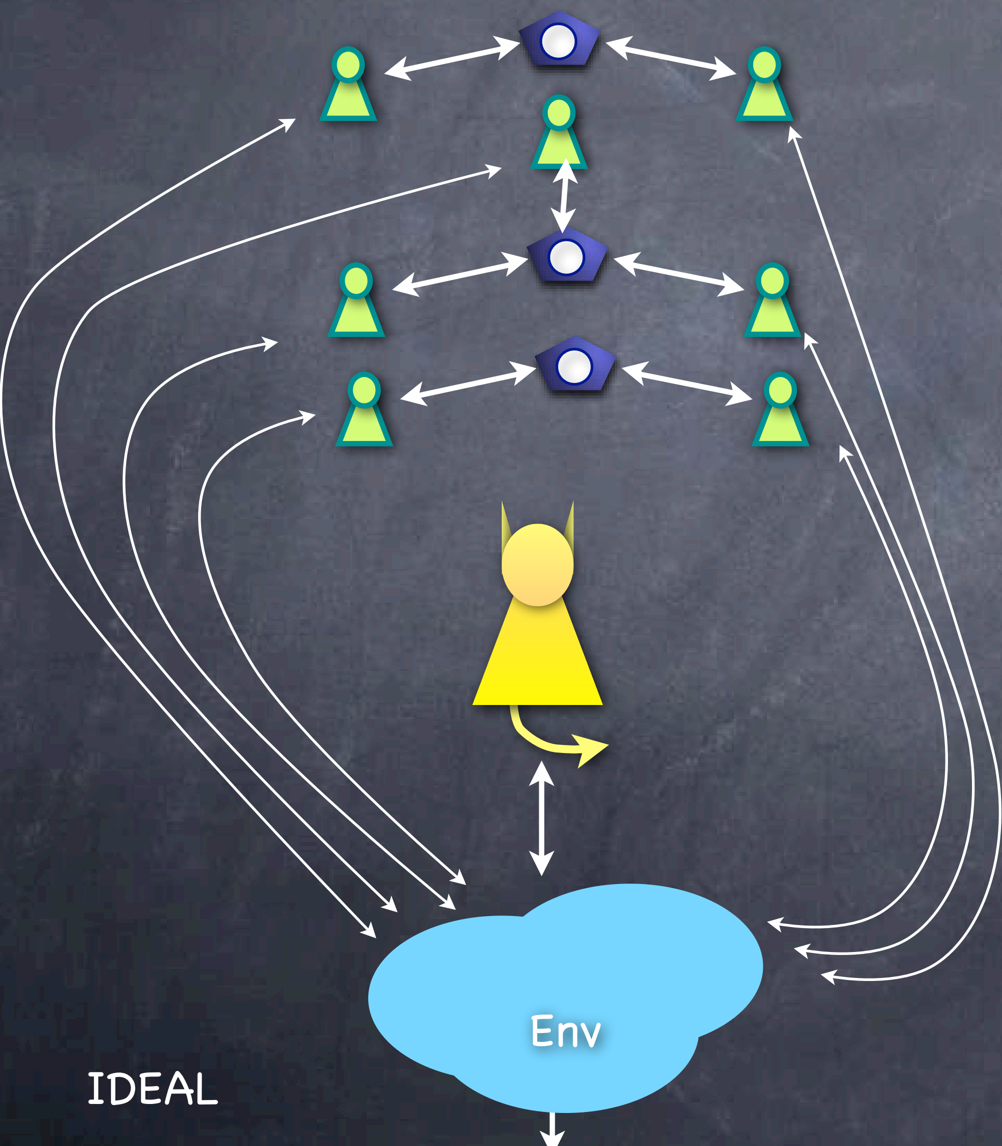
Security of Composed Systems

- Extend to allow a “composed system” with multiple functionalities

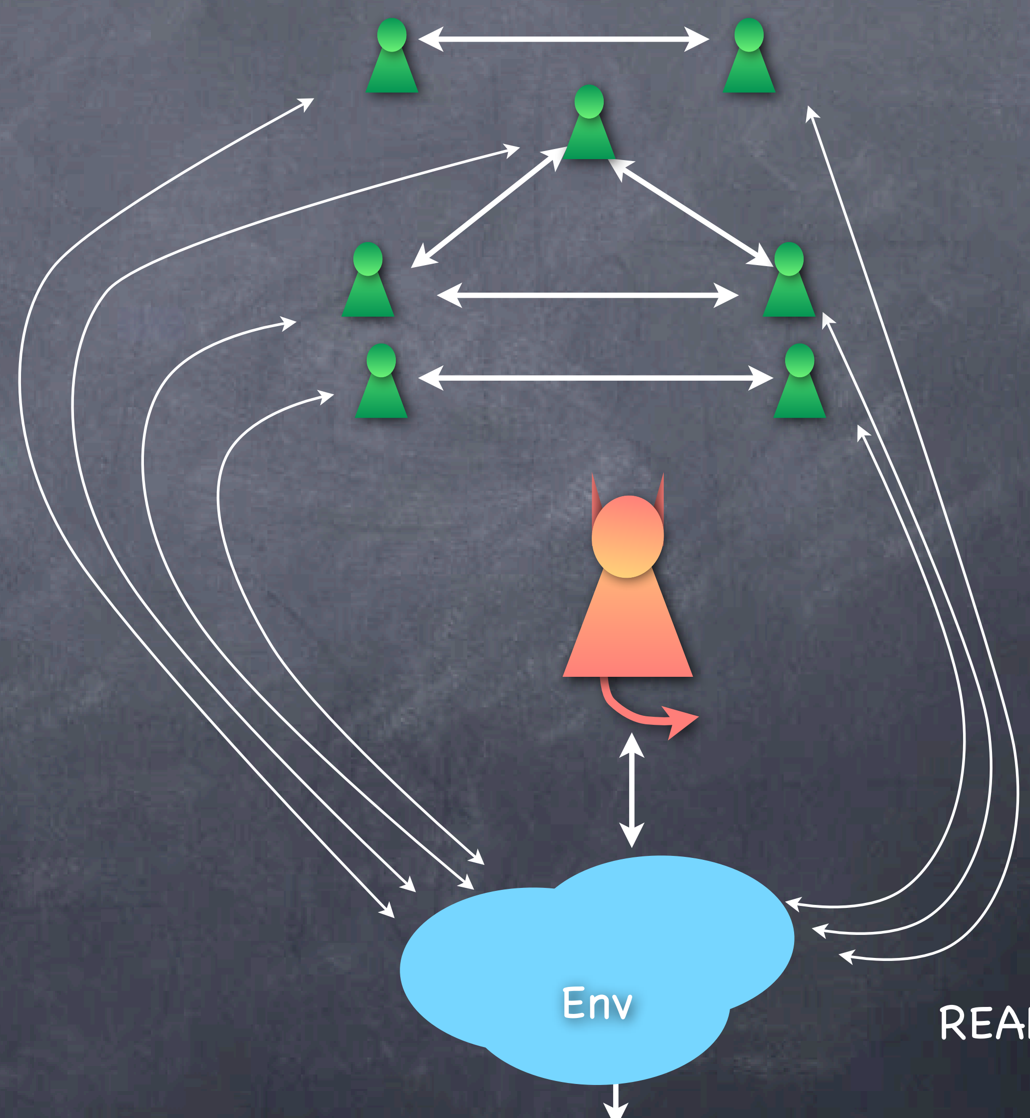


Security of Composed Systems

- Extend to allow a "composed system" with multiple functionalities



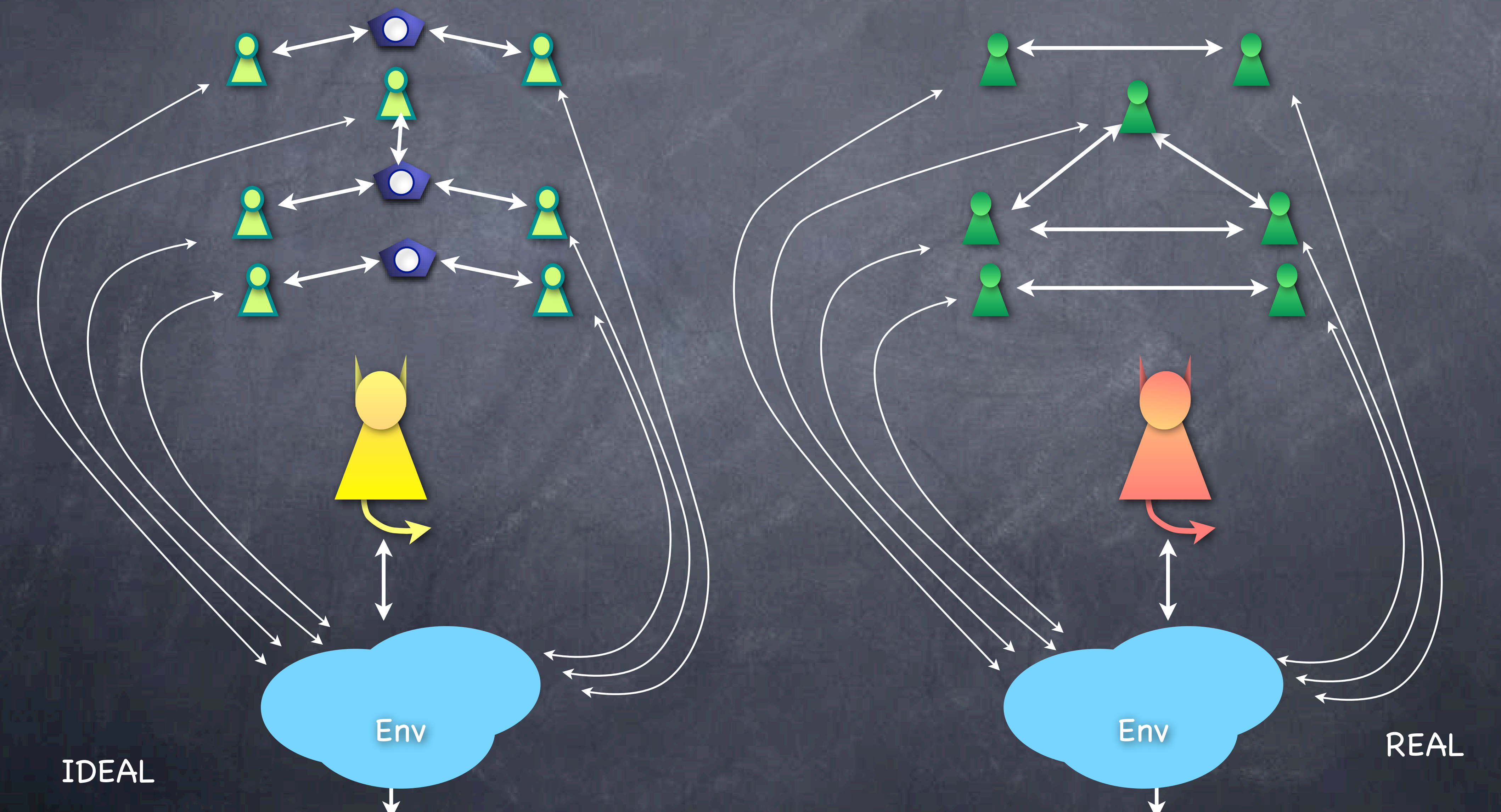
IDEAL



REAL

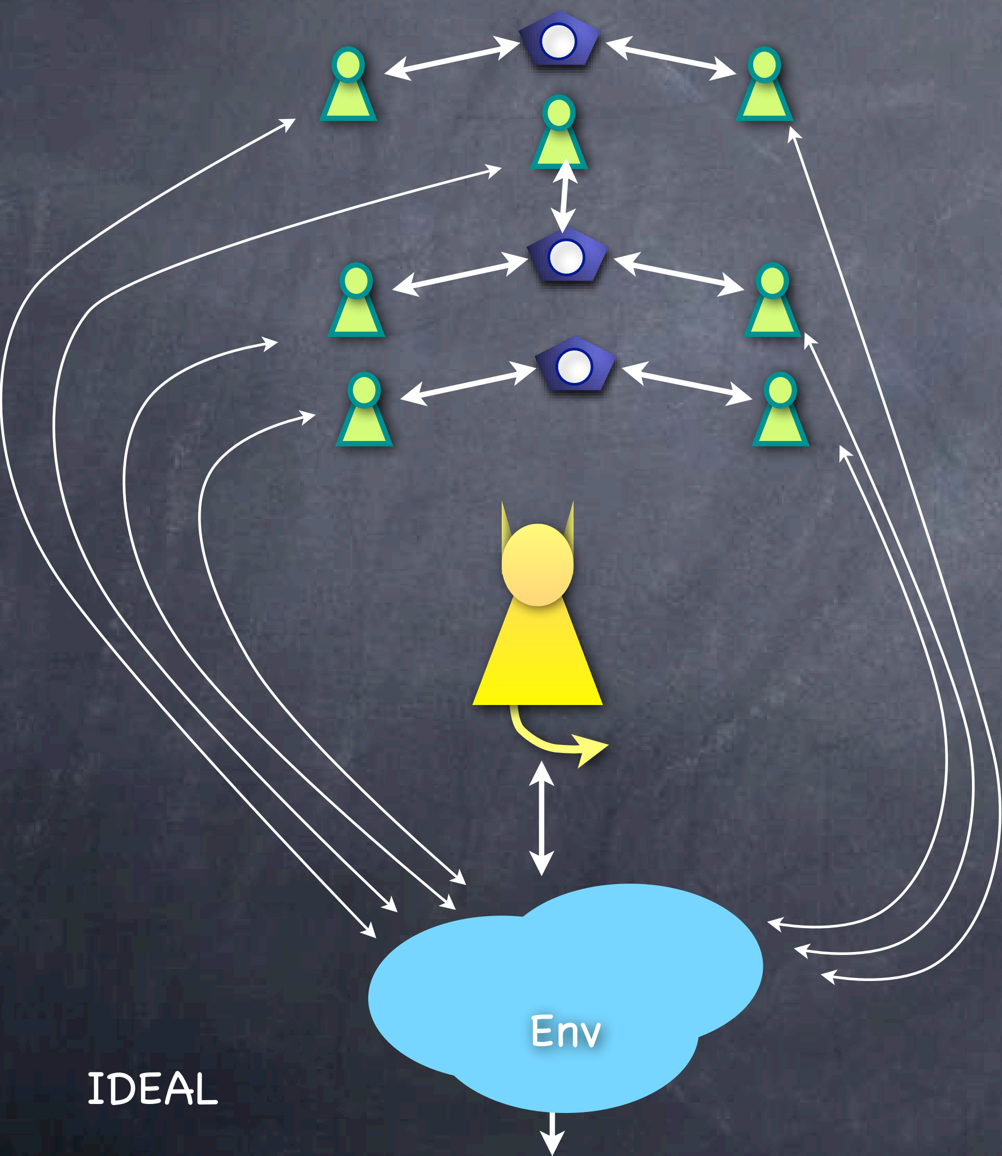
Security of Composed Systems

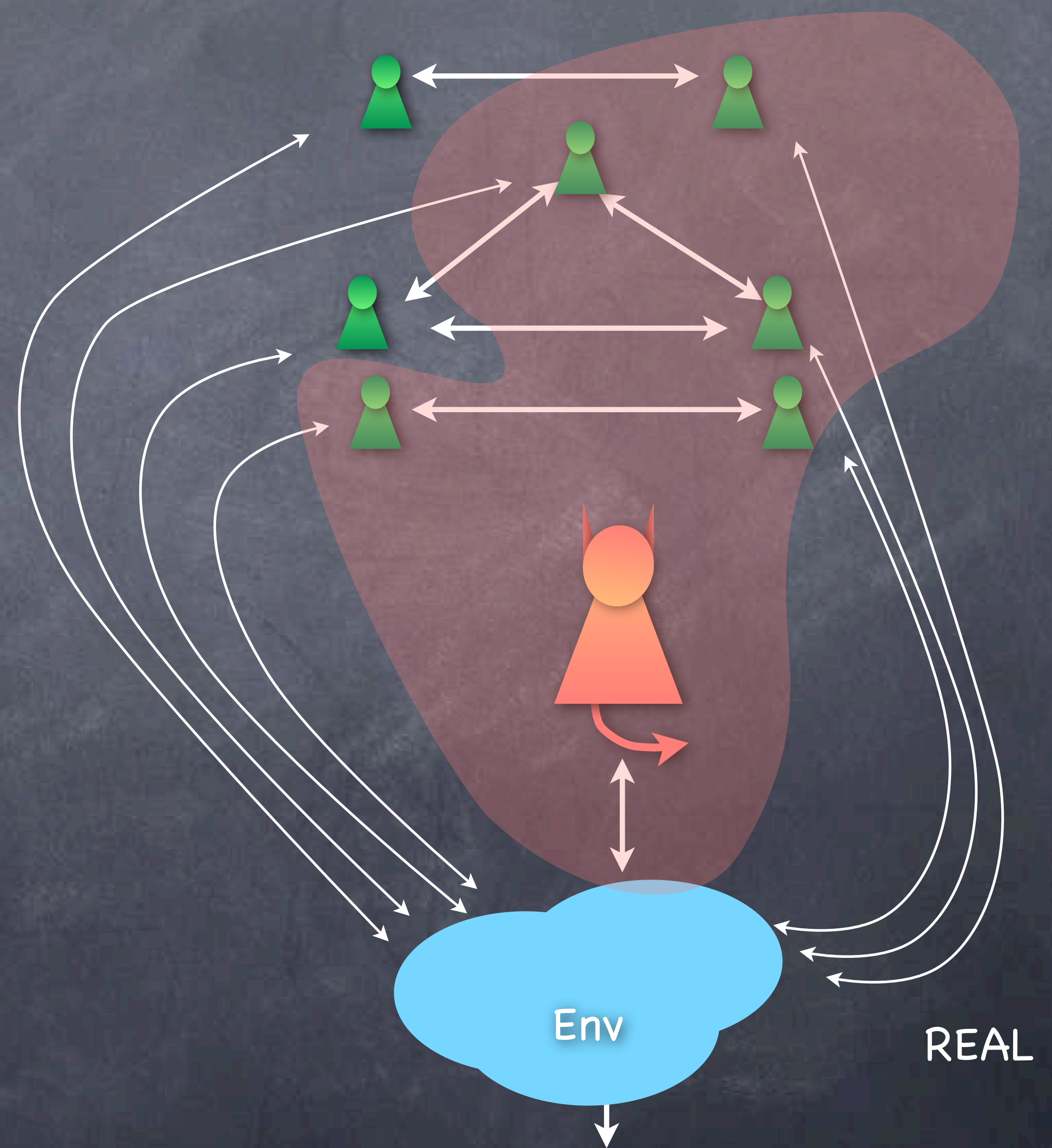
- Extend to allow a "composed system" with multiple functionalities
- REAL (with protocols) is as secure as IDEAL (with functionalities) if:



Security of Composed Systems

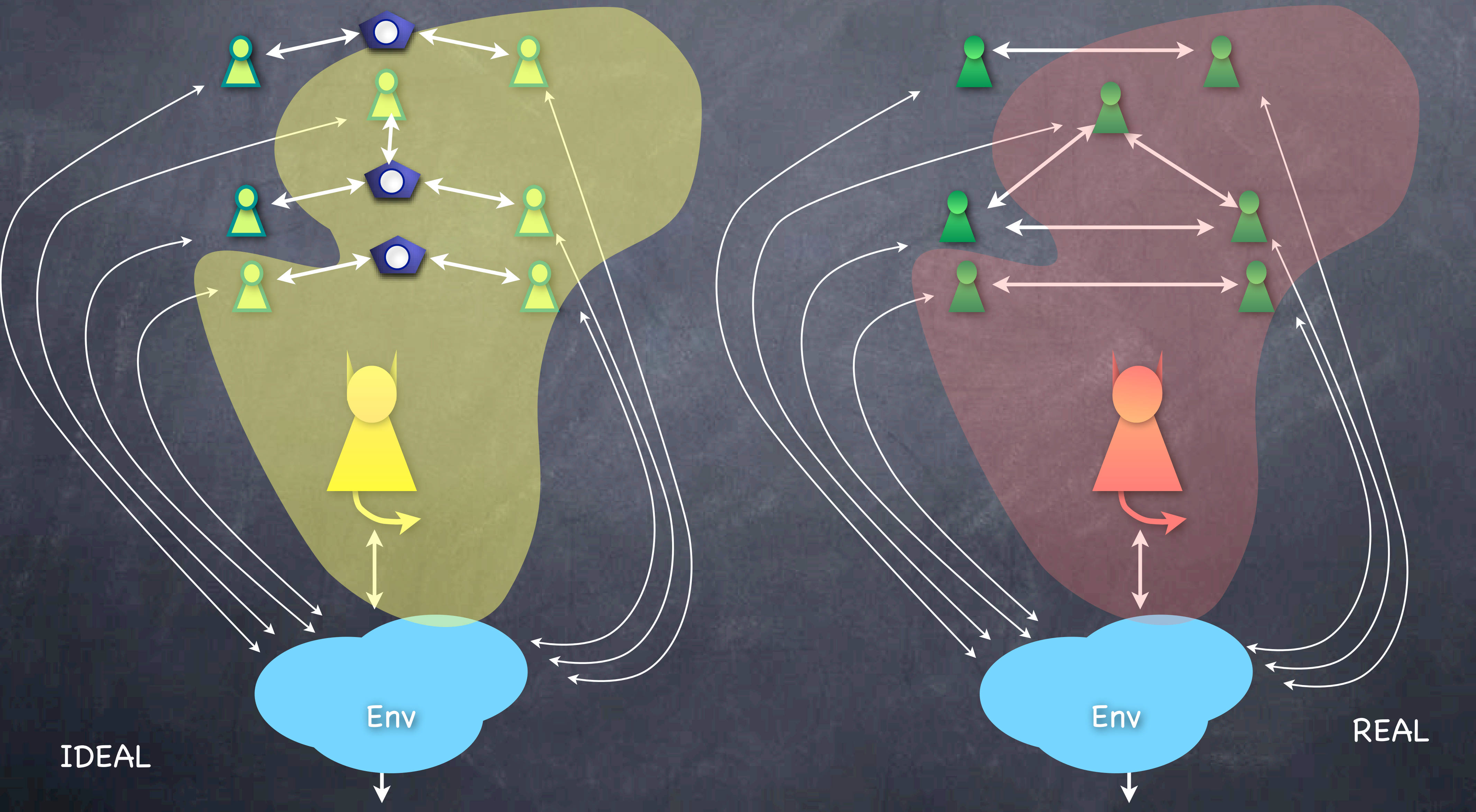
- Extend to allow a "composed system" with multiple functionalities
- REAL (with protocols) is as secure as IDEAL (with functionalities) if:





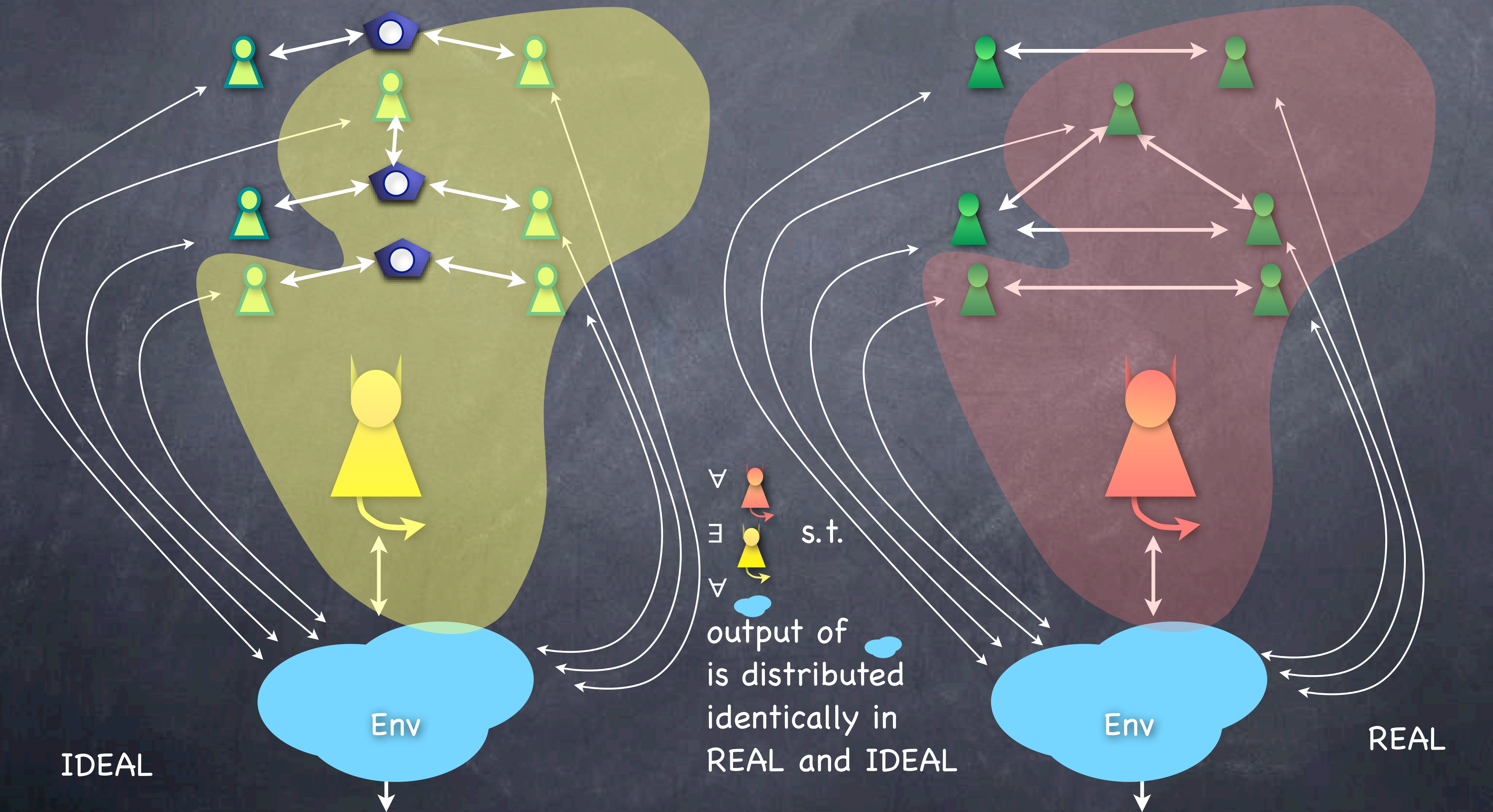
Security of Composed Systems

- Extend to allow a "composed system" with multiple functionalities
- REAL (with protocols) is as secure as IDEAL (with functionalities) if:

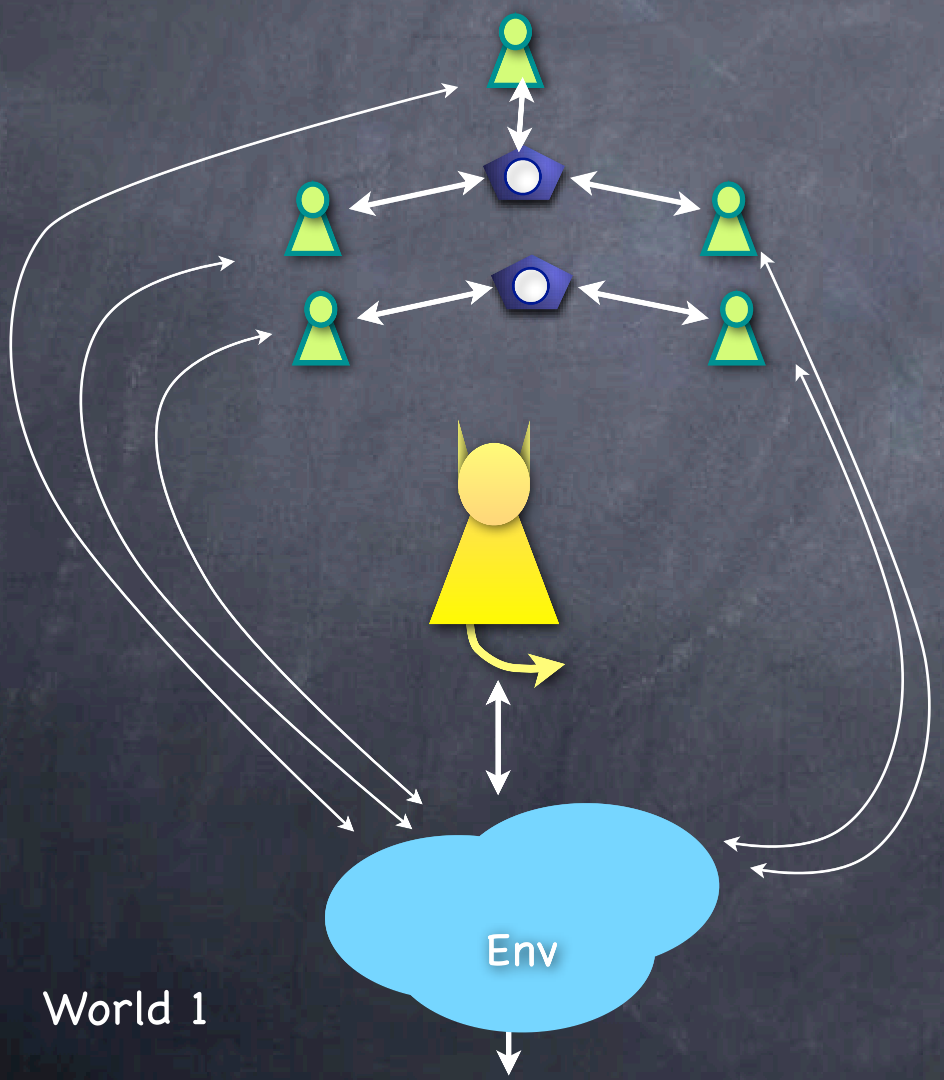


Security of Composed Systems

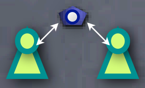
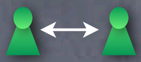
- Extend to allow a "composed system" with multiple functionalities
- REAL (with protocols) is as secure as IDEAL (with functionalities) if:

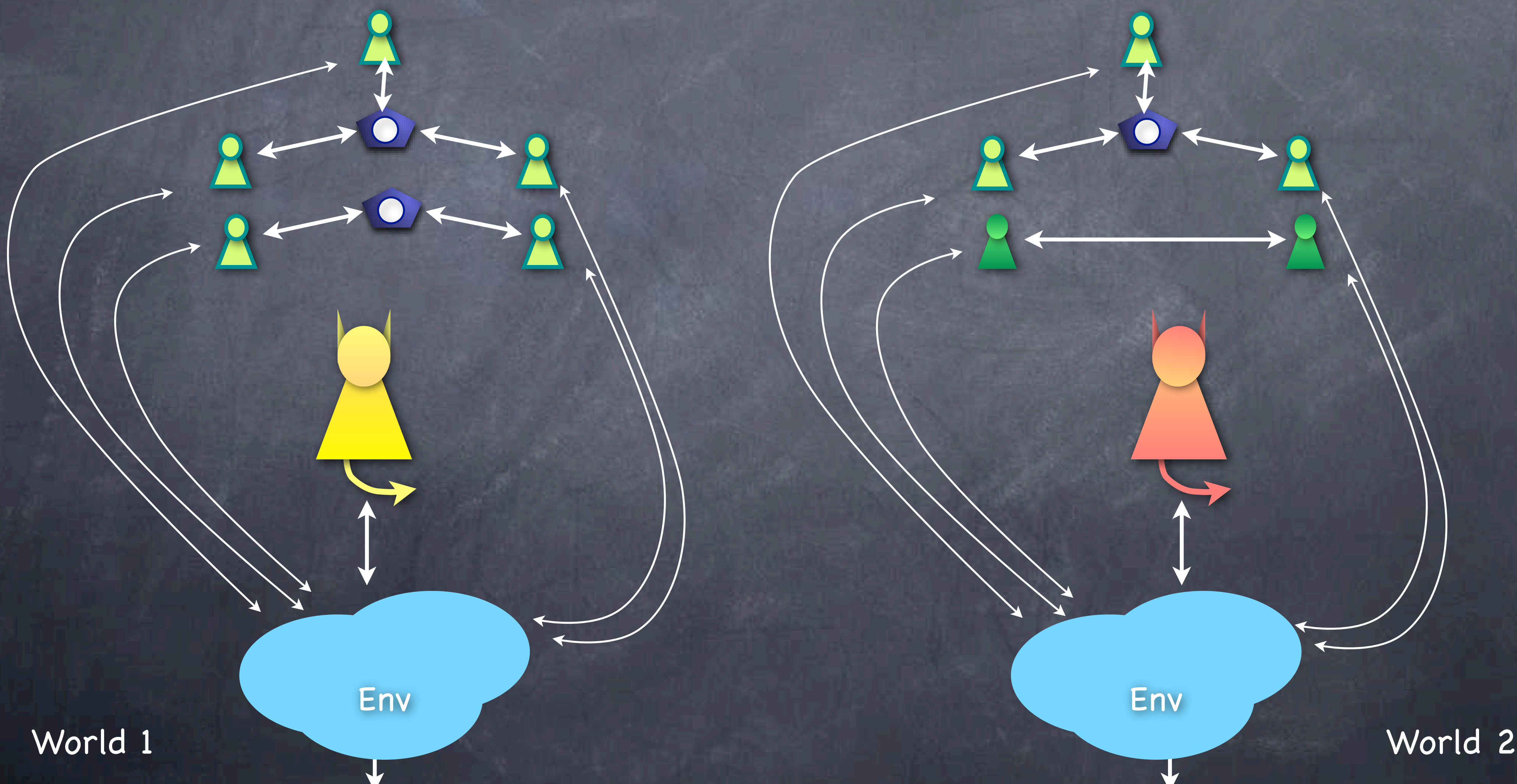


Universal Composition



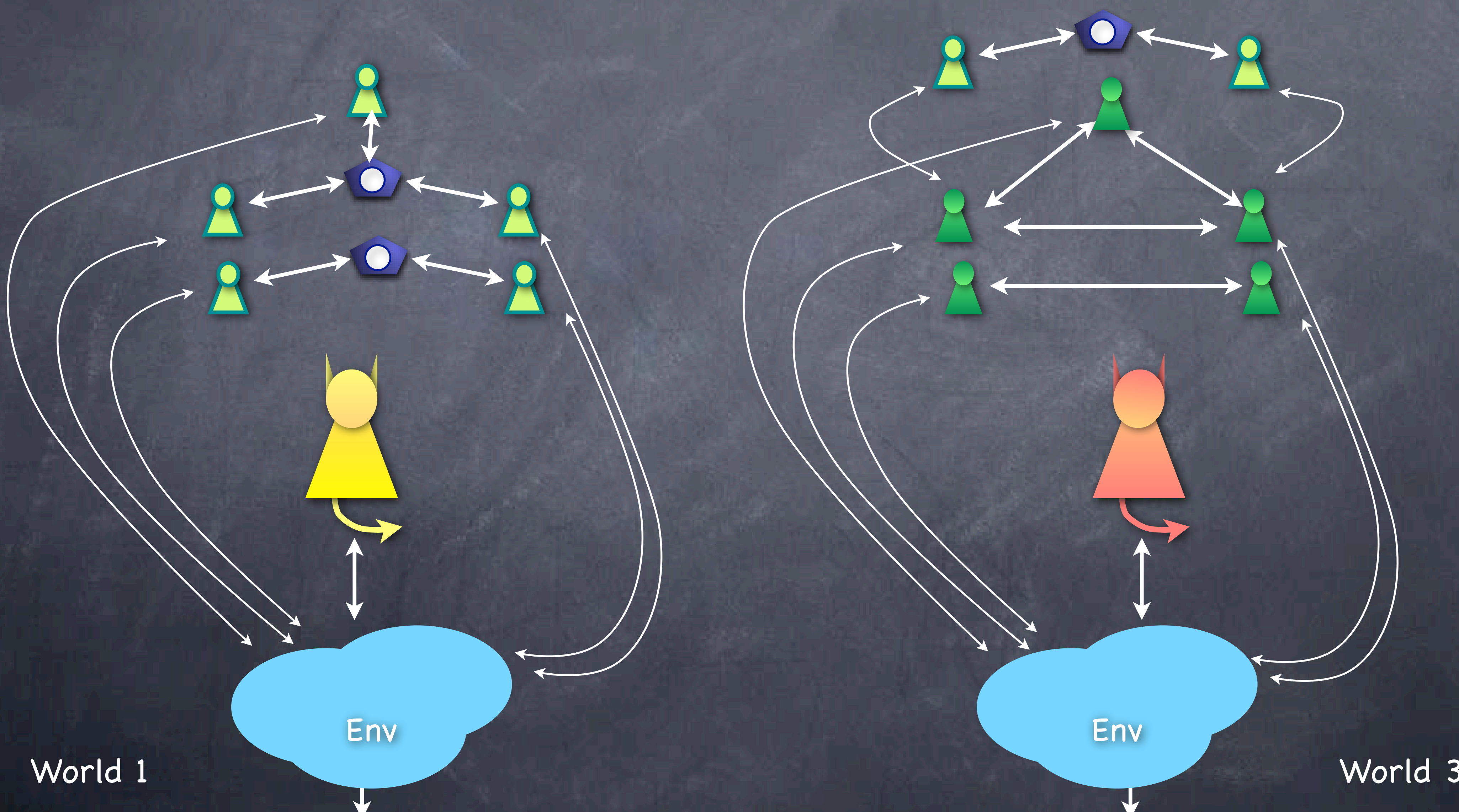
Universal Composition

Replace protocol  with  which is as SIM-secure, etc.

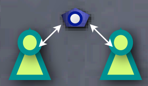


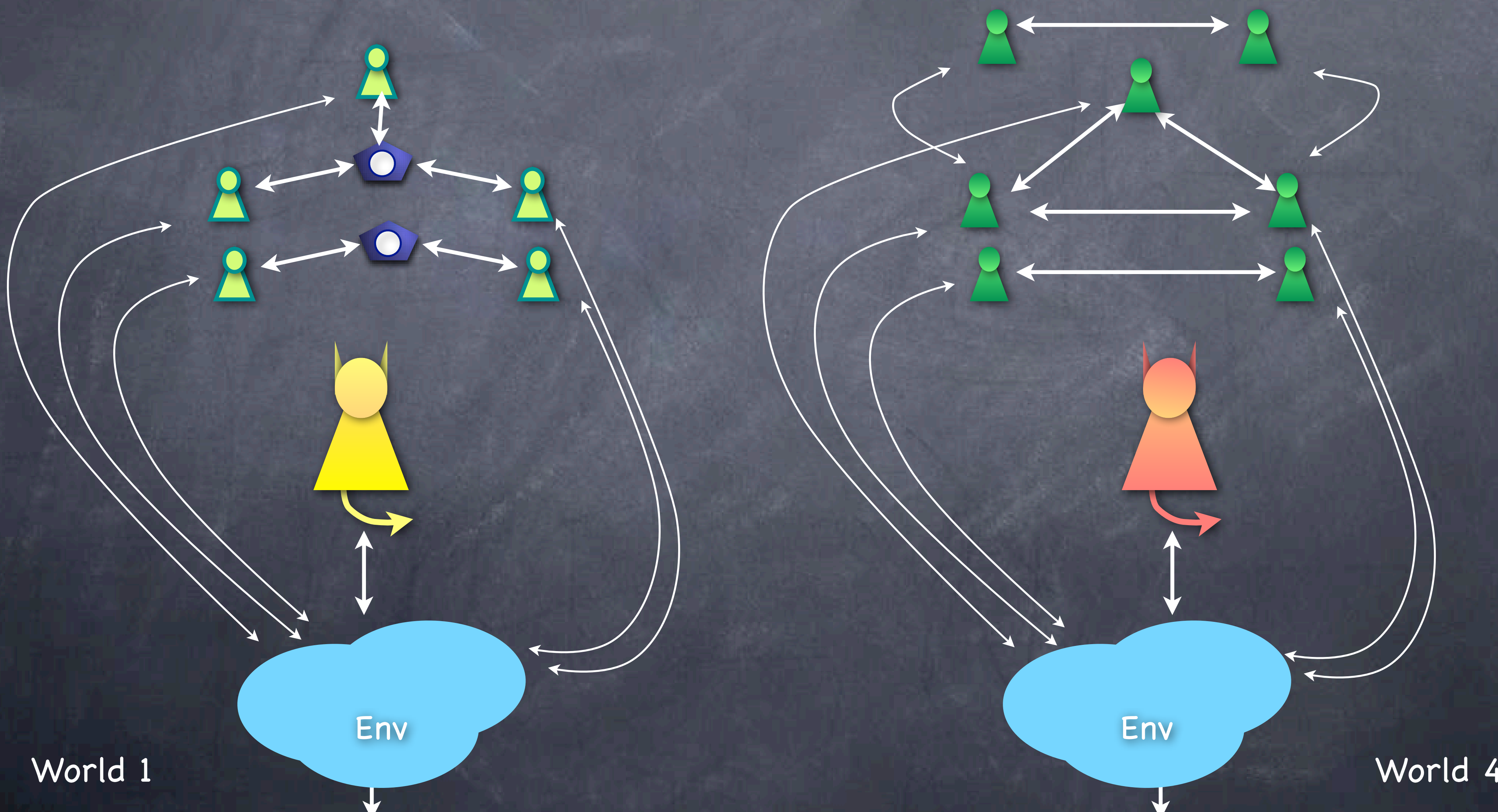
Universal Composition

Replace protocol with which is as SIM-secure, etc.

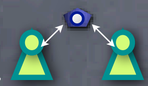


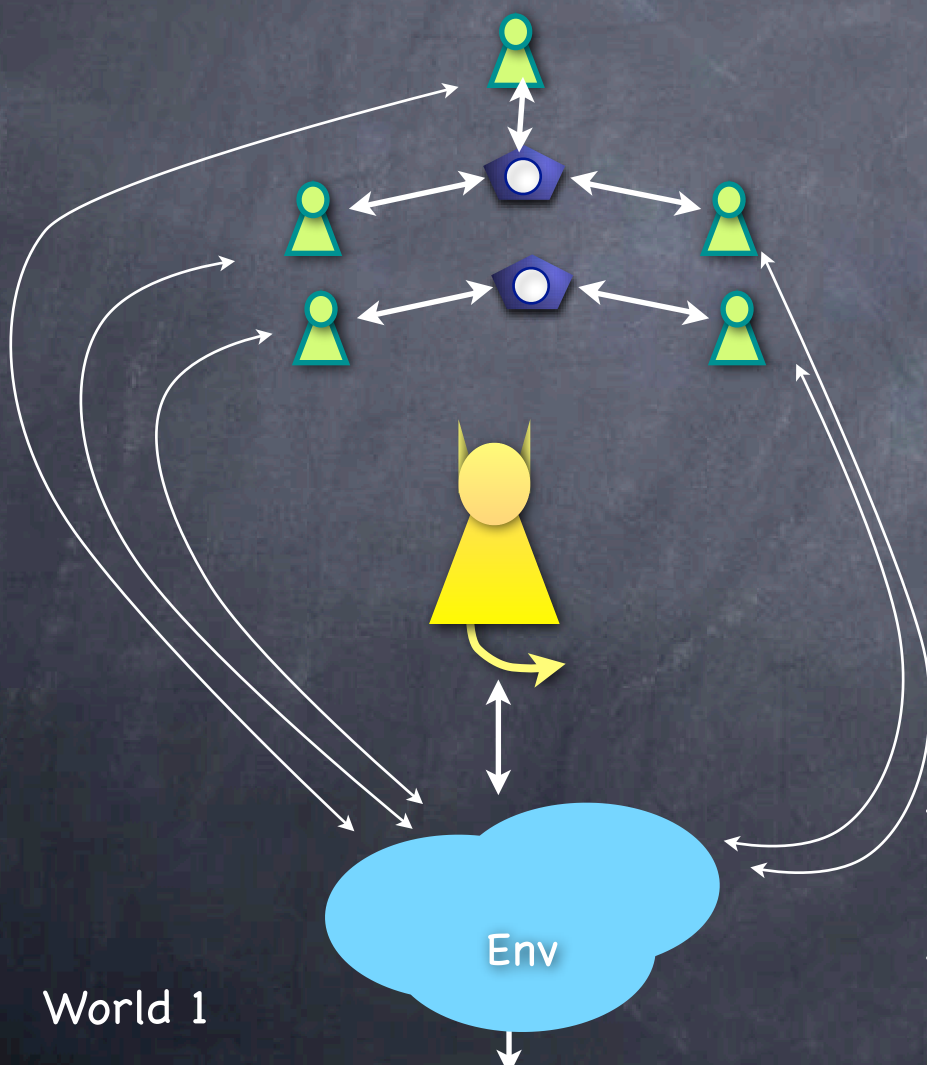
Universal Composition

Replace protocol  with  which is as SIM-secure, etc.

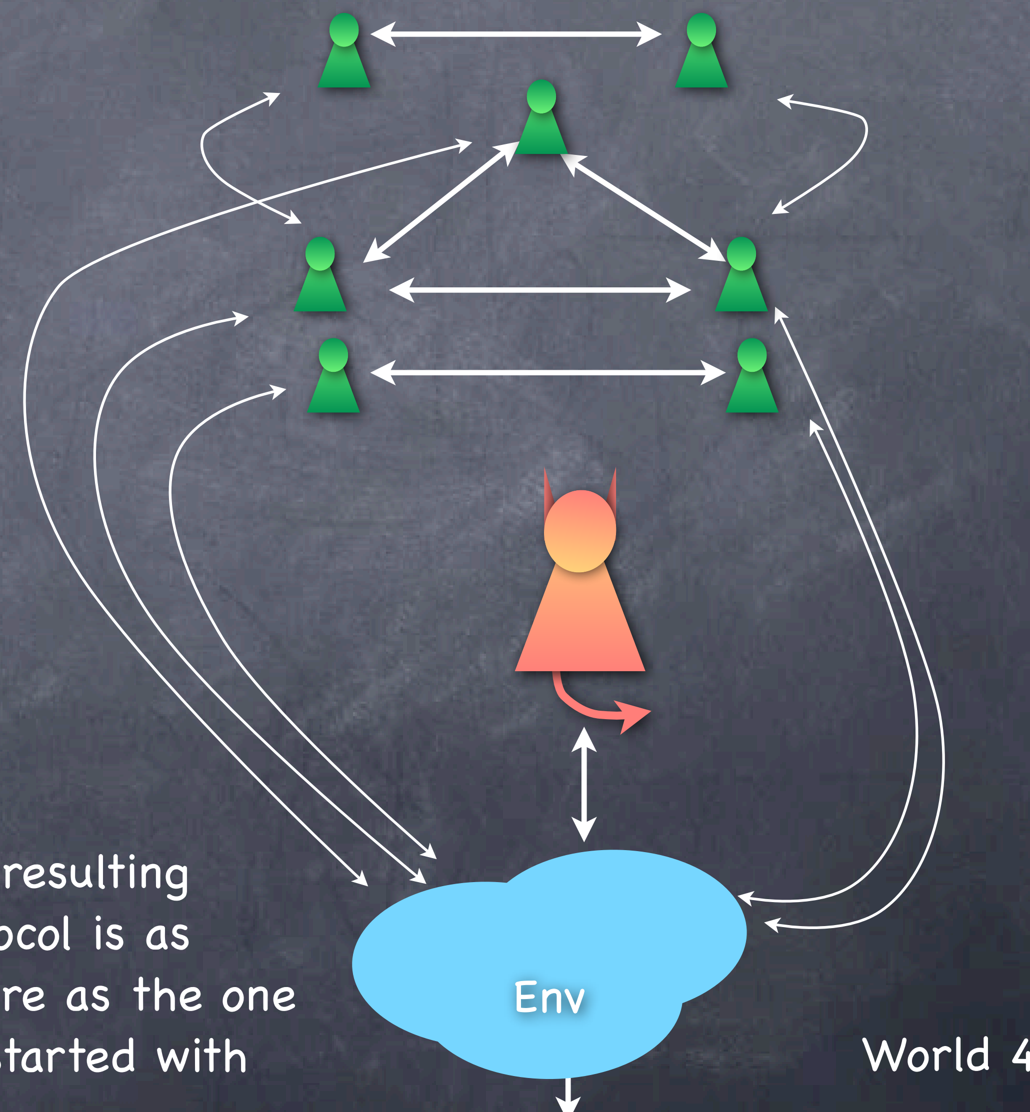


Universal Composition

Replace protocol  with  which is as SIM-secure, etc.



The resulting protocol is as secure as the one we started with



Universal Composition

Universal Composition

- Start from world A (think “IDEAL”)

Universal Composition

- Start from world A (think "IDEAL")
  - Repeat (for any poly number of times):

Universal Composition

- Start from world A (think "IDEAL")
  - Repeat (for any poly number of times):
    - For some 2 "protocols" (that possibly make use of ideal functionalities) I and R such that R is as secure as I, substitute an I-session by an R-session

Universal Composition

- Start from world A (think "IDEAL")
  - Repeat (for any poly number of times):
    - For some 2 "protocols" (that possibly make use of ideal functionalities) I and R such that R is as secure as I, substitute an I-session by an R-session
  - Say we obtain world B (think "REAL")

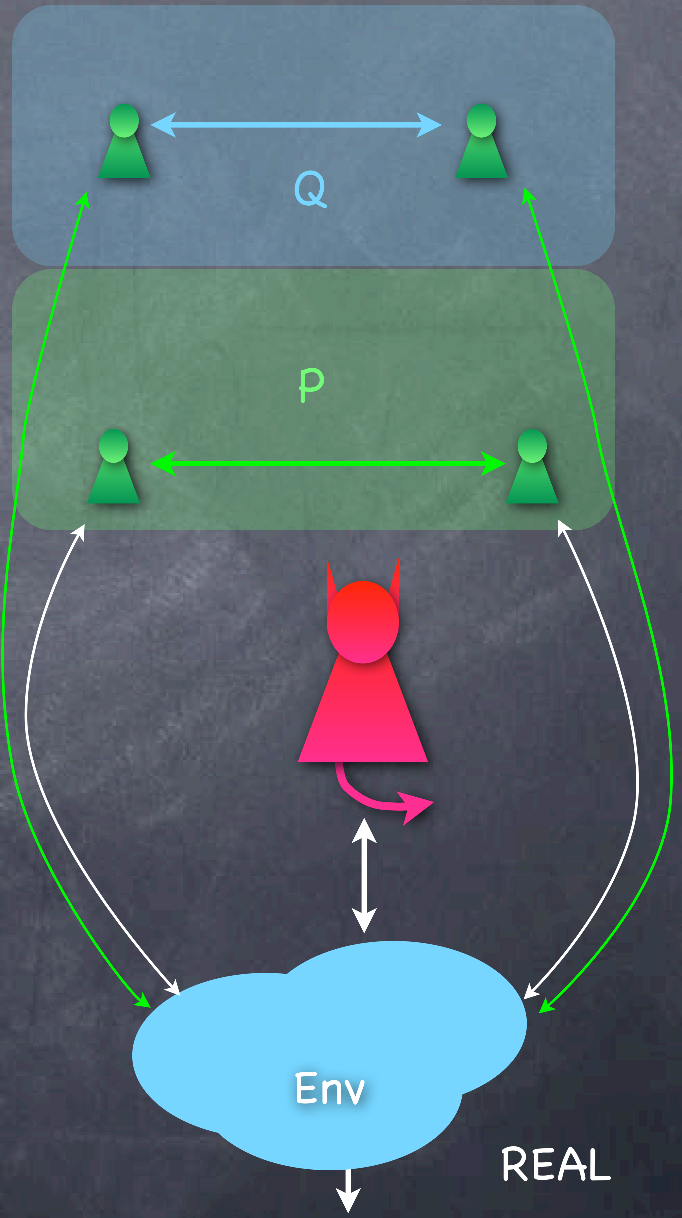
Universal Composition

- Start from world A (think "IDEAL")
  - Repeat (for any poly number of times):
    - For some 2 "protocols" (that possibly make use of ideal functionalities) I and R such that R is as secure as I, substitute an I-session by an R-session
  - Say we obtain world B (think "REAL")
  - UC Theorem: Then world B is as secure as world A

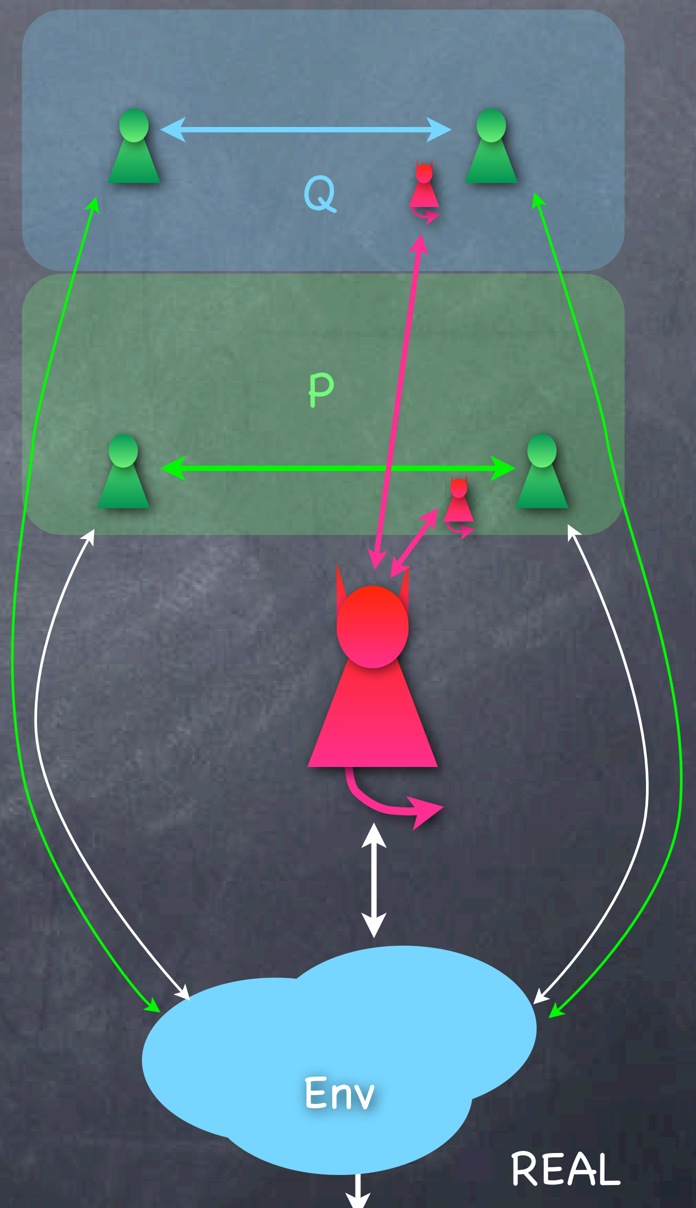
Universal Composition

- Start from world A (think "IDEAL")
  - Repeat (for any poly number of times):
    - For some 2 "protocols" (that possibly make use of ideal functionalities) I and R such that R is as secure as I, substitute an I-session by an R-session
  - Say we obtain world B (think "REAL")
  - UC Theorem: Then world B is as secure as world A
- Gives a modular implementation of the IDEAL world

Proving the UC theorem

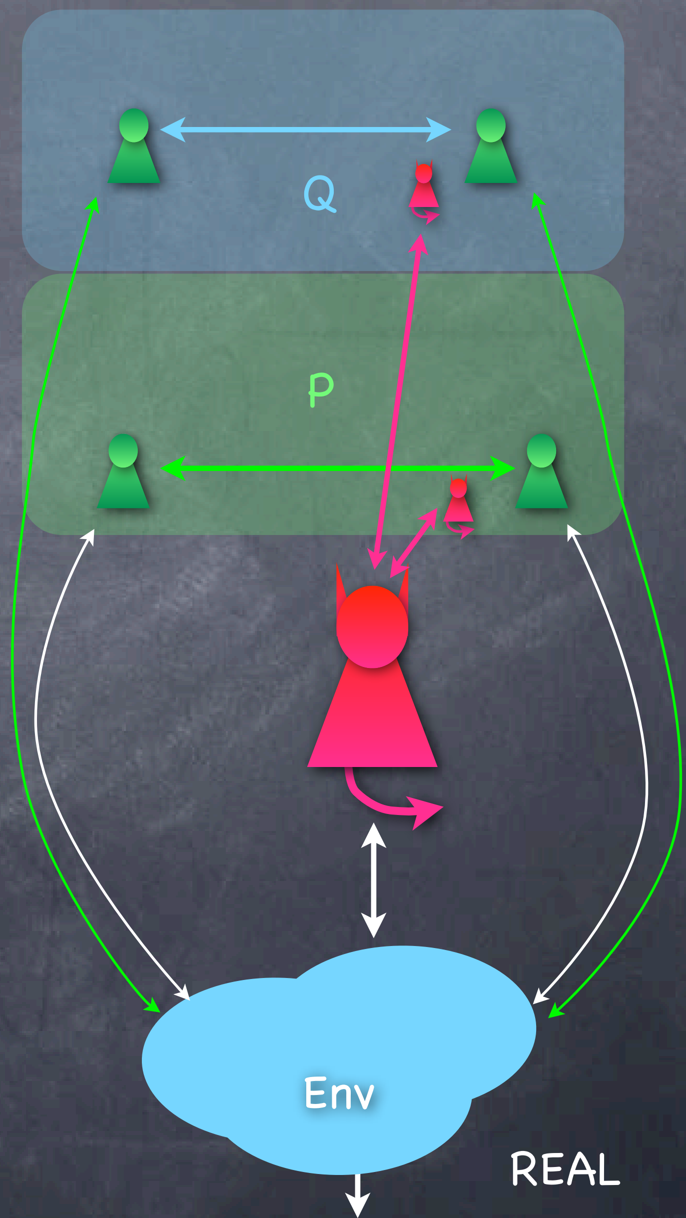


Proving the UC theorem



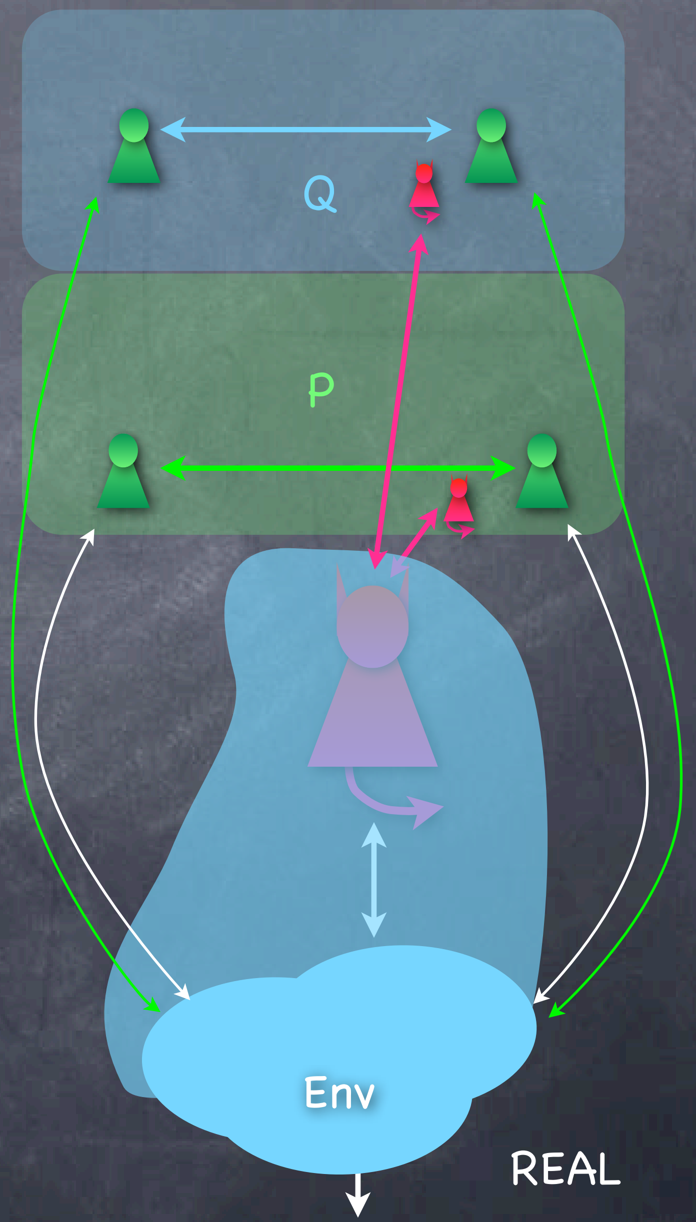
Proving the UC theorem

- Consider environment which runs the adversary internally, and depends on "dummy adversaries" to interface with the protocols



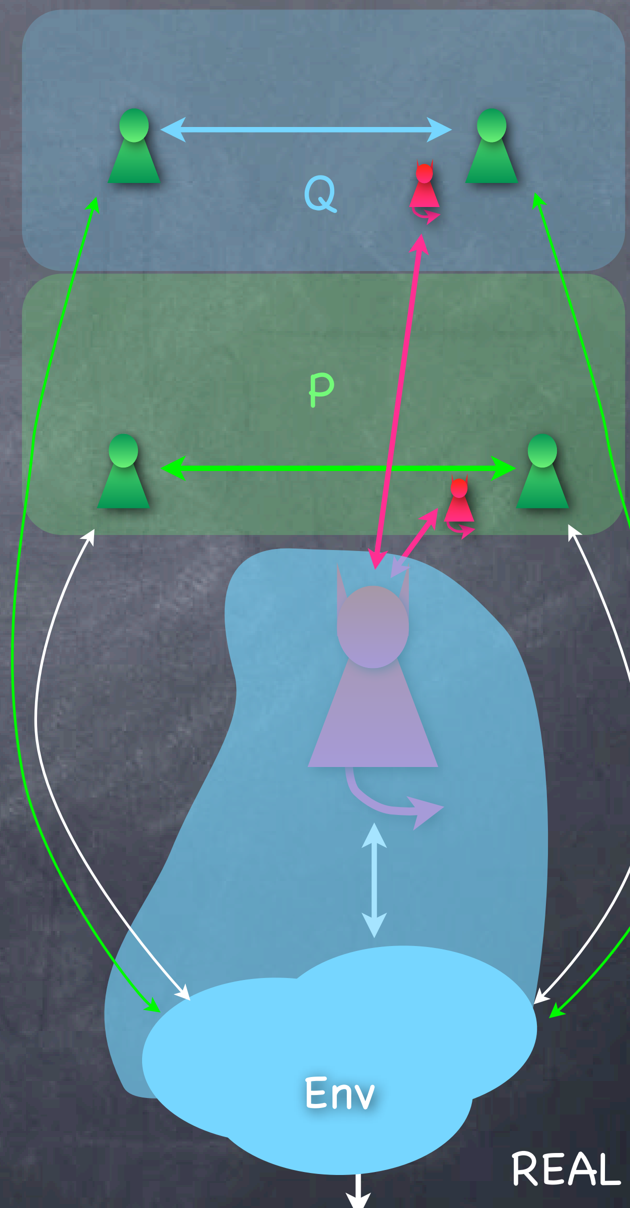
Proving the UC theorem

- Consider environment which runs the adversary internally, and depends on "dummy adversaries" to interface with the protocols



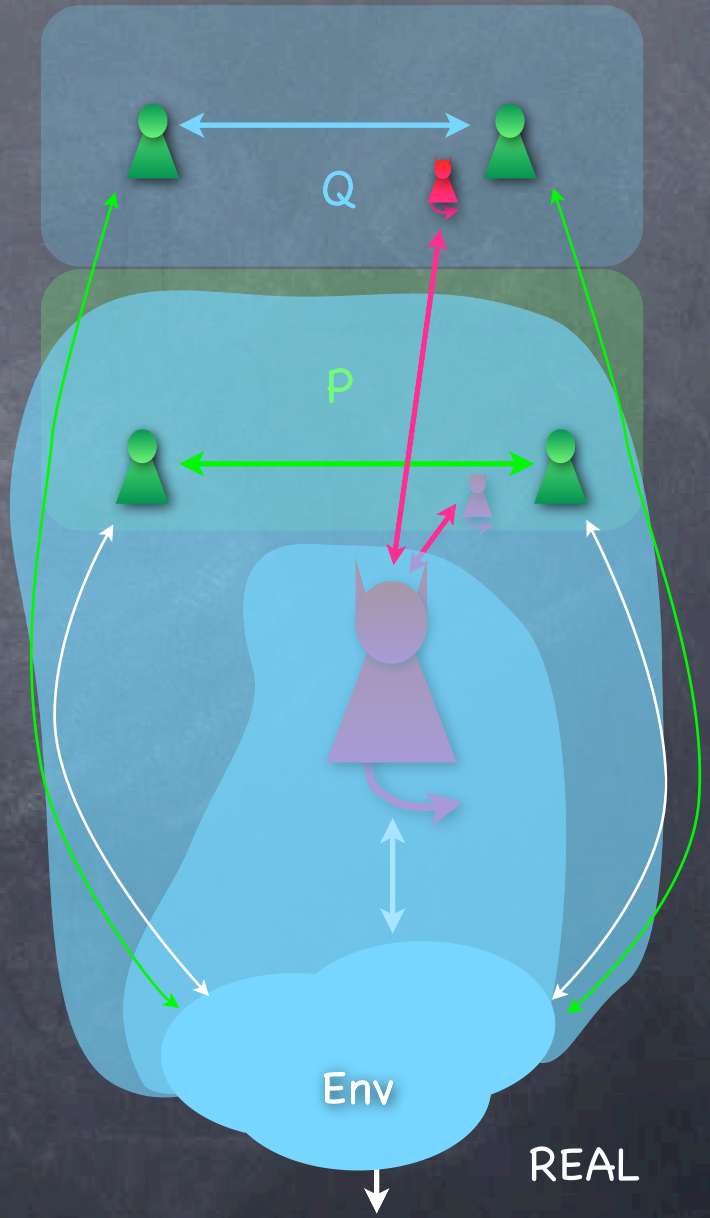
Proving the UC theorem

- Consider environment which runs the adversary internally, and depends on "dummy adversaries" to interface with the protocols
- Now consider new environment s.t. only Q (and its adversary) is outside it



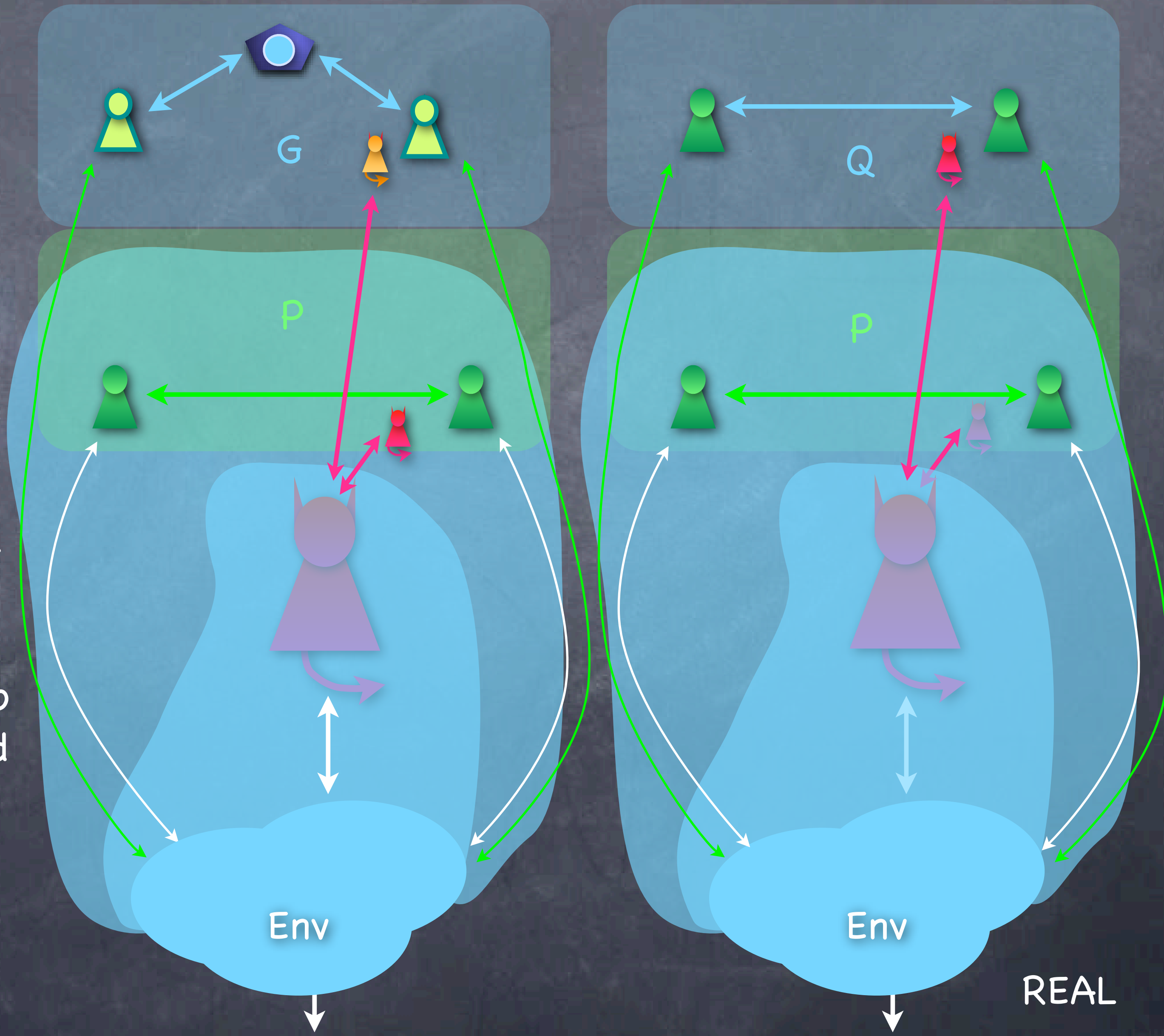
Proving the UC theorem

- Consider environment which runs the adversary internally, and depends on "dummy adversaries" to interface with the protocols
- Now consider new environment s.t. only Q (and its adversary) is outside it



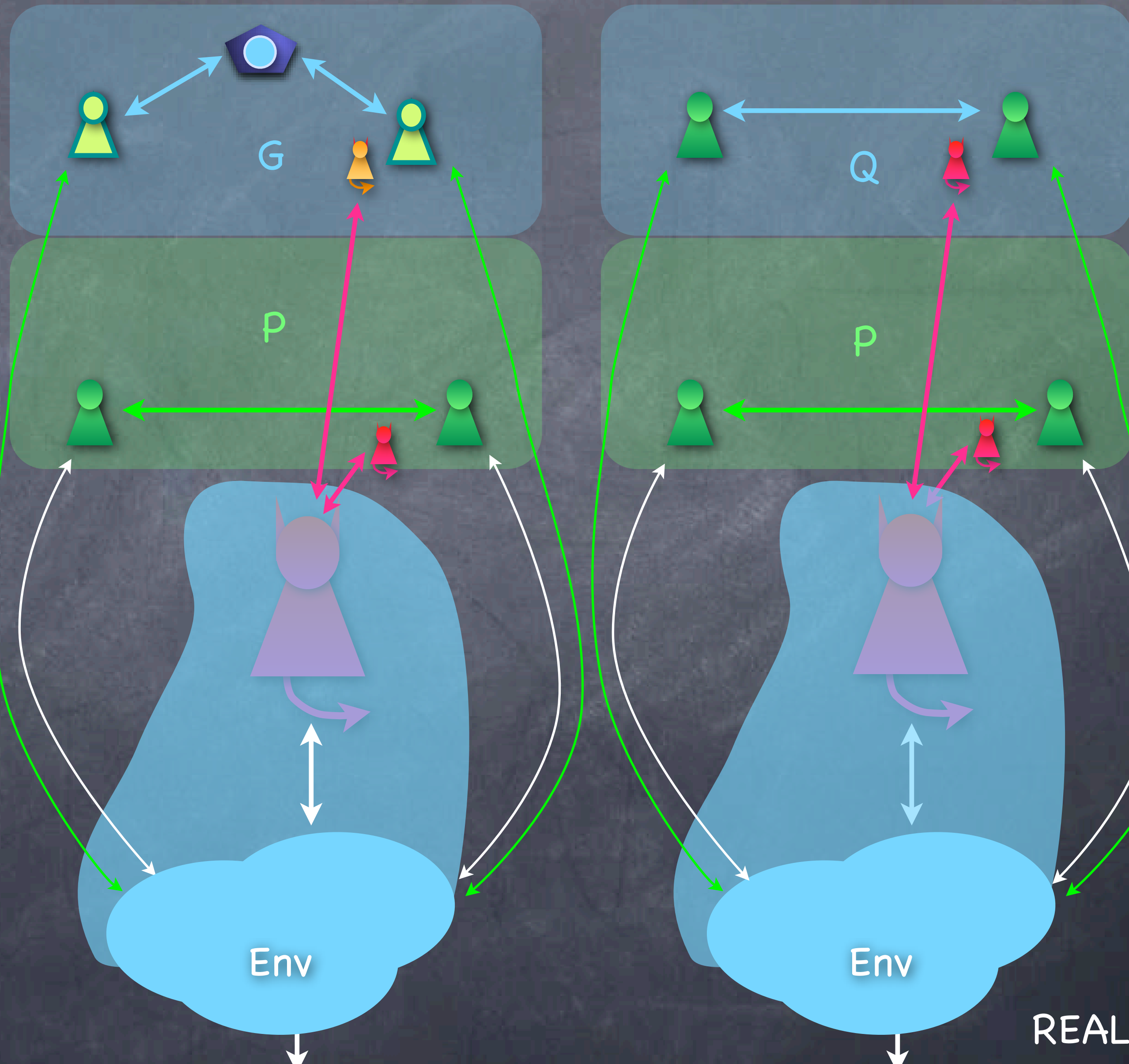
Proving the UC theorem

- Consider environment which runs the adversary internally, and depends on "dummy adversaries" to interface with the protocols
- Now consider new environment s.t. only Q (and its adversary) is outside it
- Use "Q is as secure as G" to get a new world with G and a new adversary

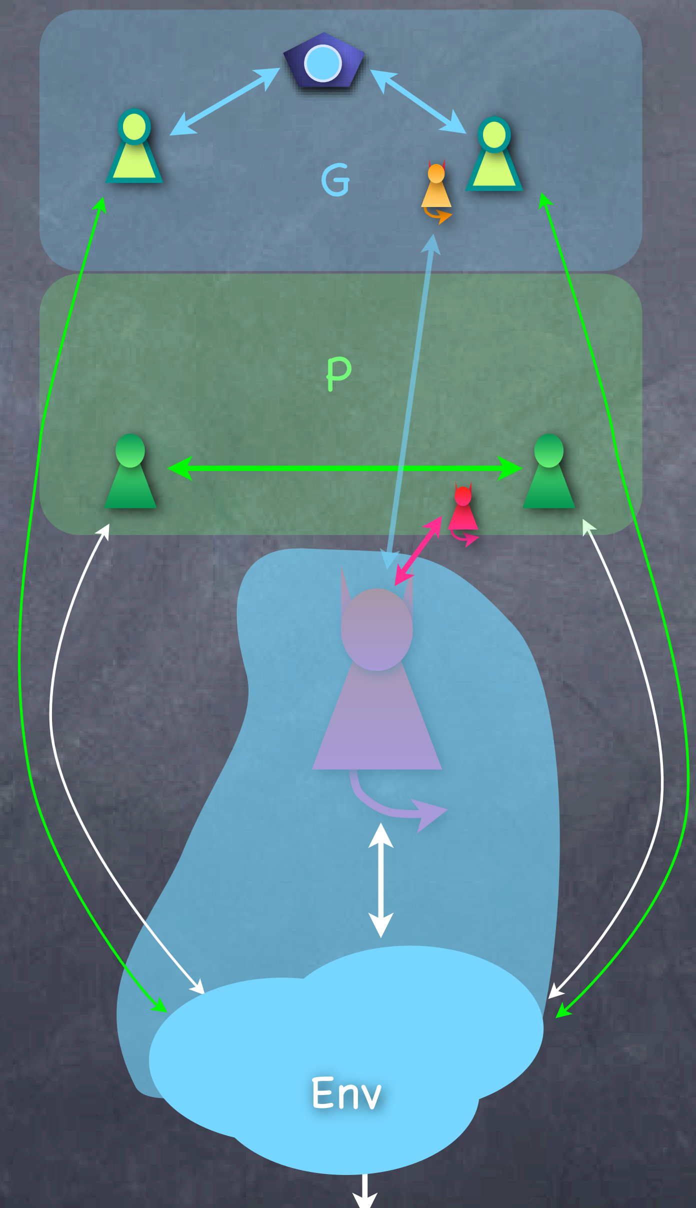


Proving the UC theorem

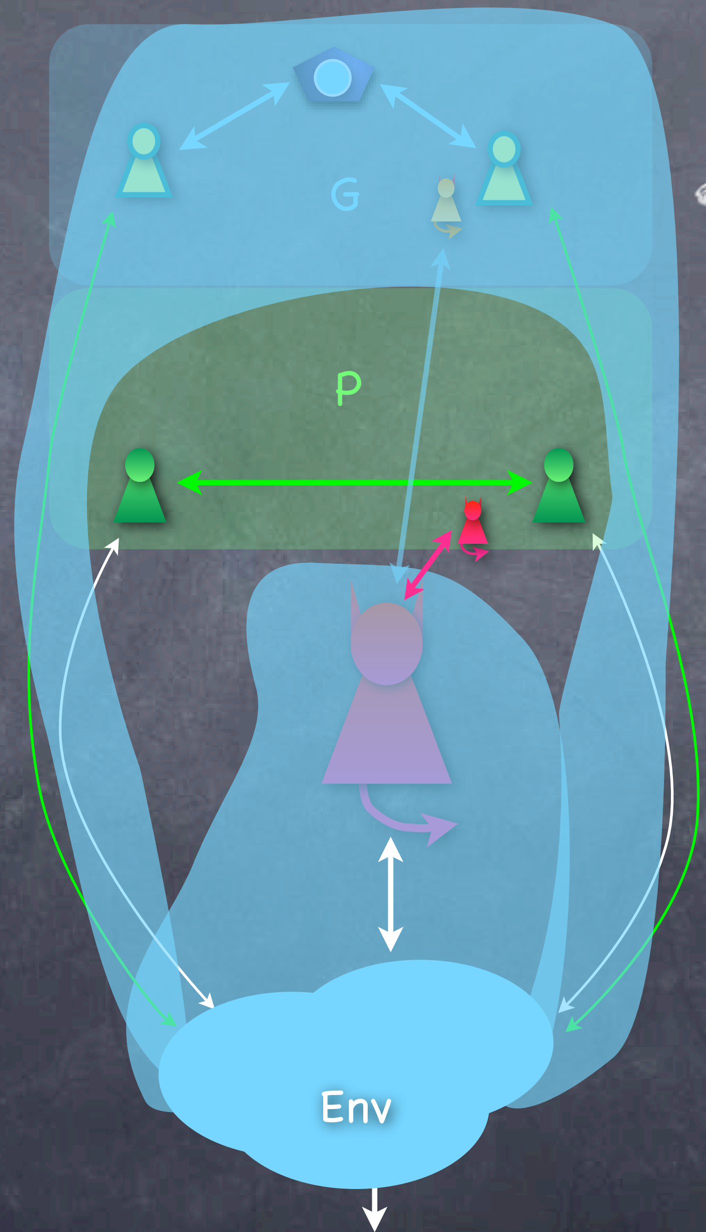
- Consider environment which runs the adversary internally, and depends on "dummy adversaries" to interface with the protocols
- Now consider new environment s.t. only Q (and its adversary) is outside it
- Use "Q is as secure as G" to get a new world with G and a new adversary



Proving the UC theorem

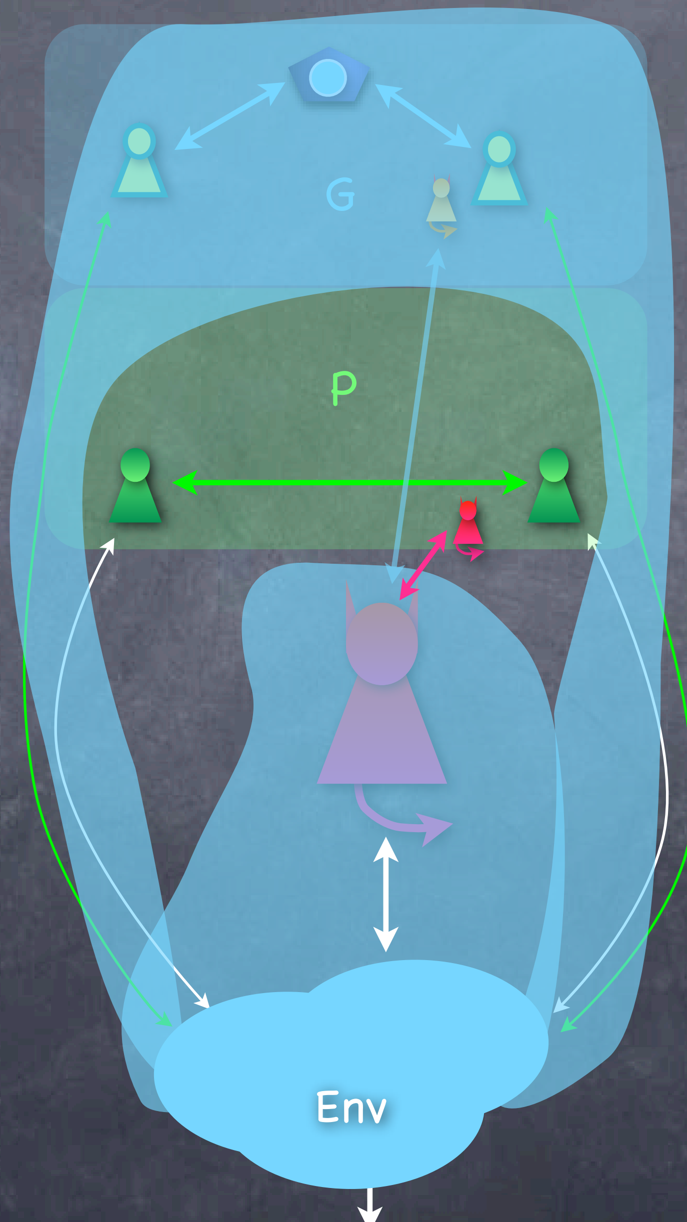


Proving the UC theorem



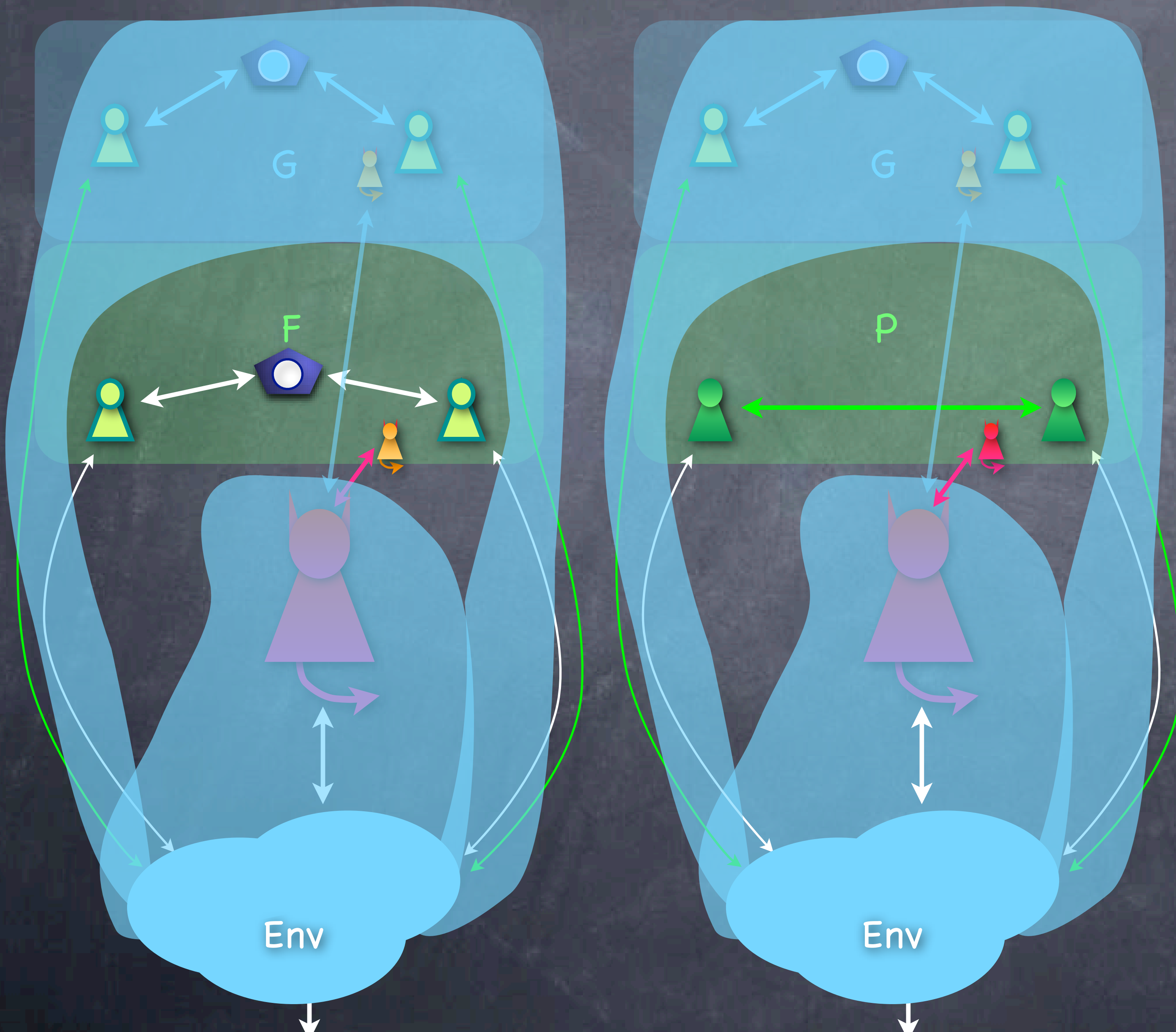
- Now consider new environment s.t. only P (and adversary) is outside it

Proving the UC theorem



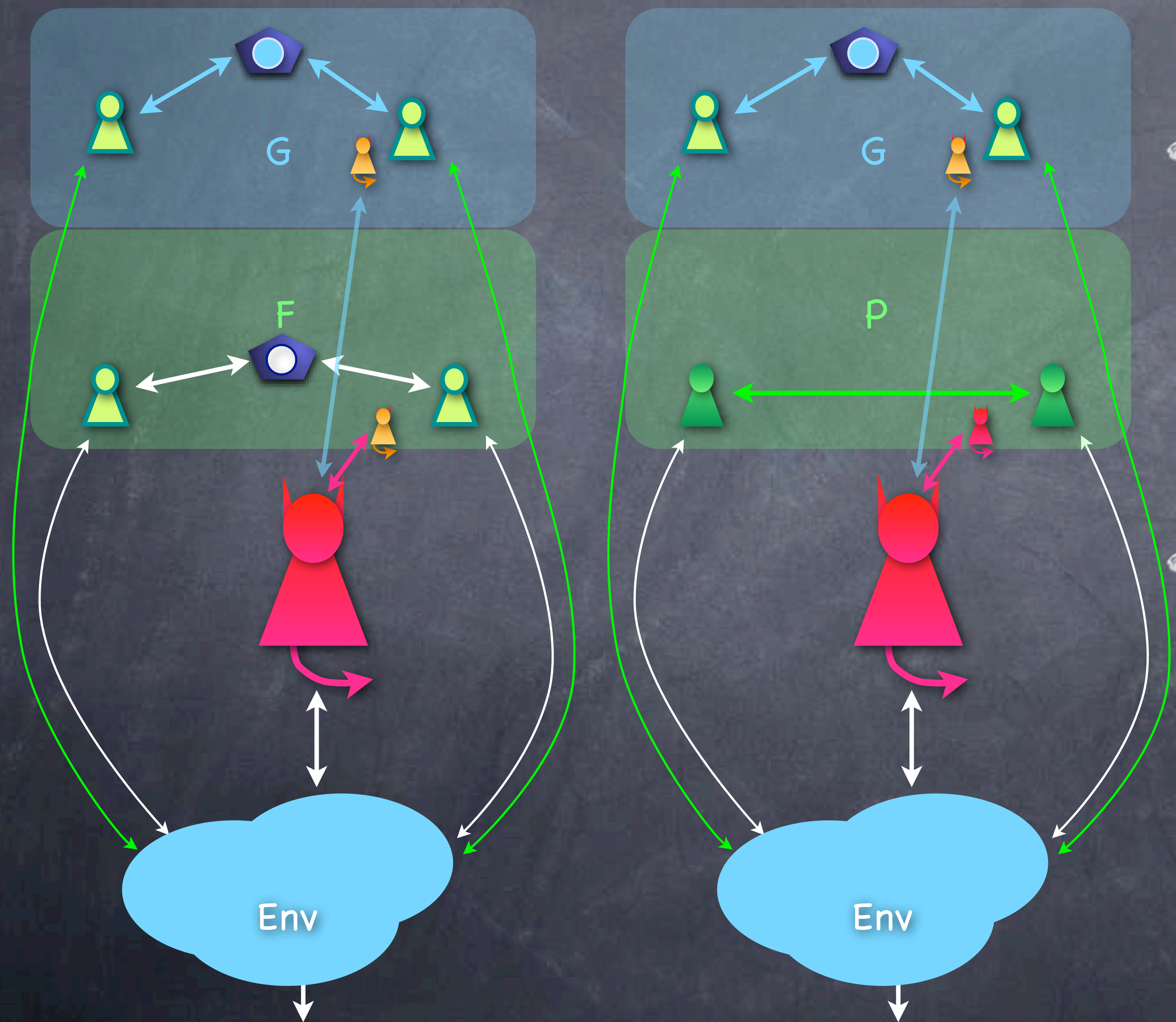
- Now consider new environment s.t. only P (and adversary) is outside it
  - Note: G and simulator for Q/G are inside the new environment

Proving the UC theorem



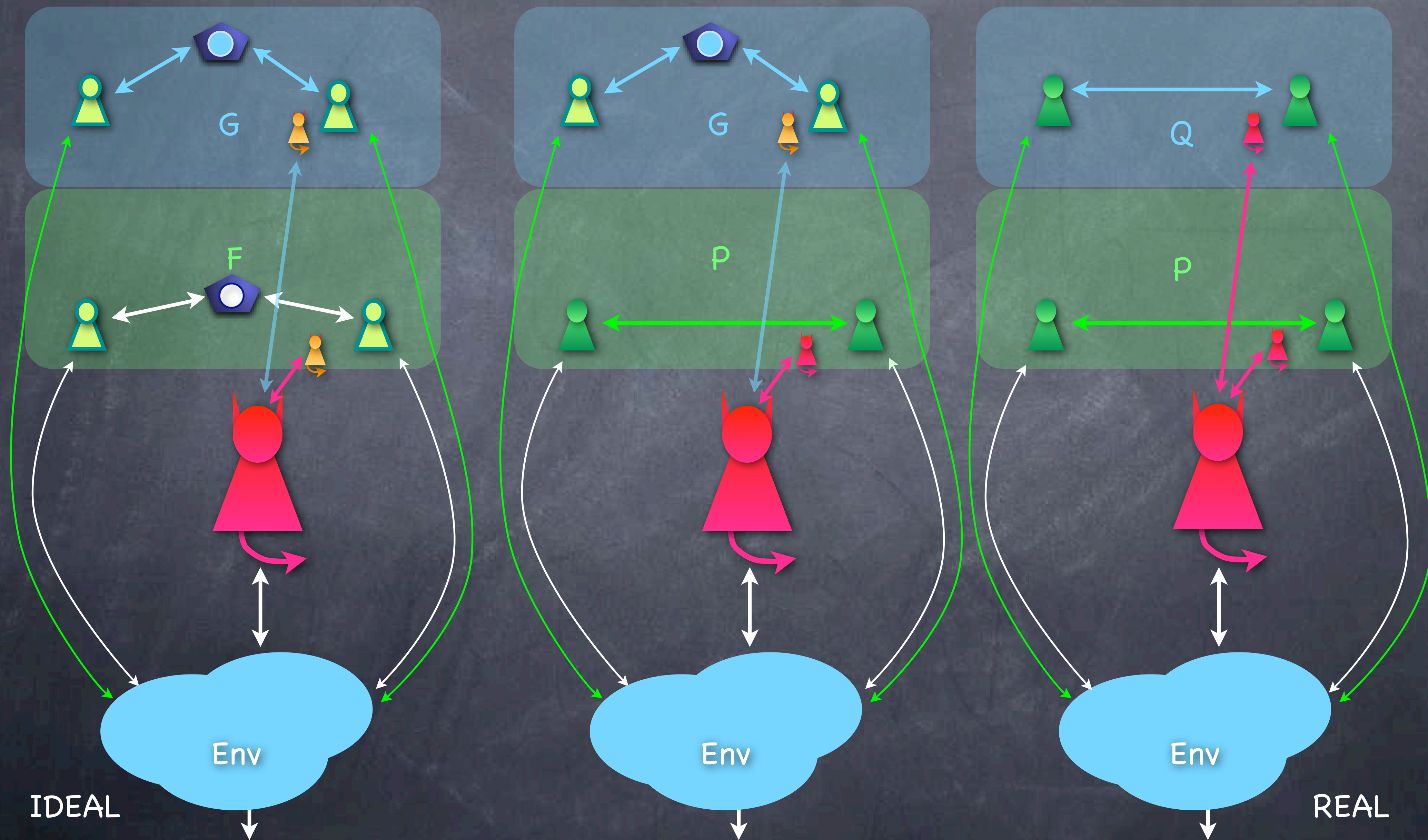
- Now consider new environment s.t. only P (and adversary) is outside it
  - Note: G and simulator for Q/G are inside the new environment
- Use "P is as secure as F" to get a new world with F and a new adversary

Proving the UC theorem

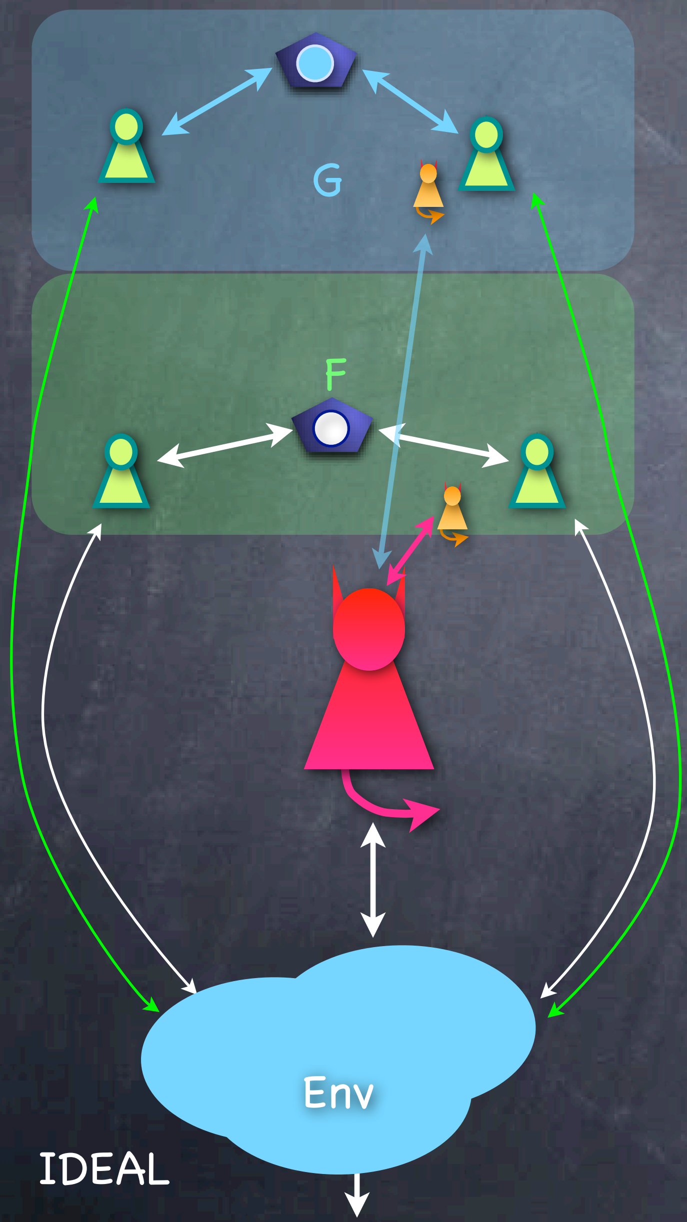


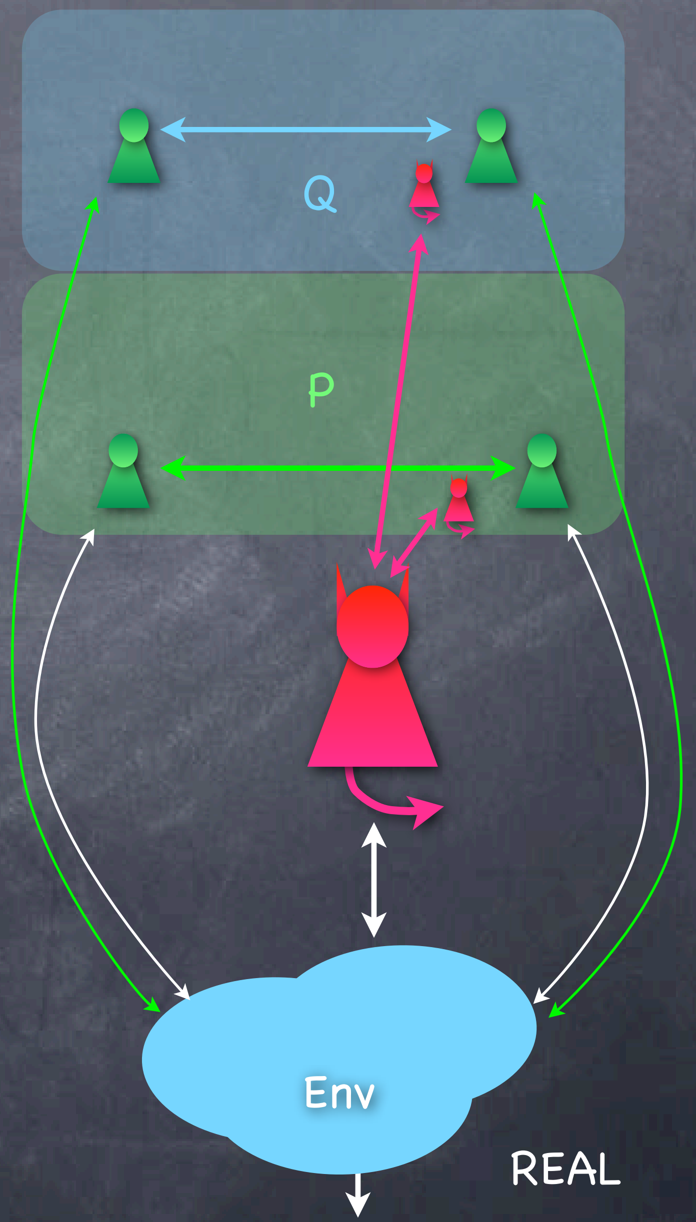
- Now consider new environment s.t. only P (and adversary) is outside it
  - Note: G and simulator for Q/G are inside the new environment
- Use "P is as secure as F" to get a new world with F and a new adversary

Proving the UC theorem

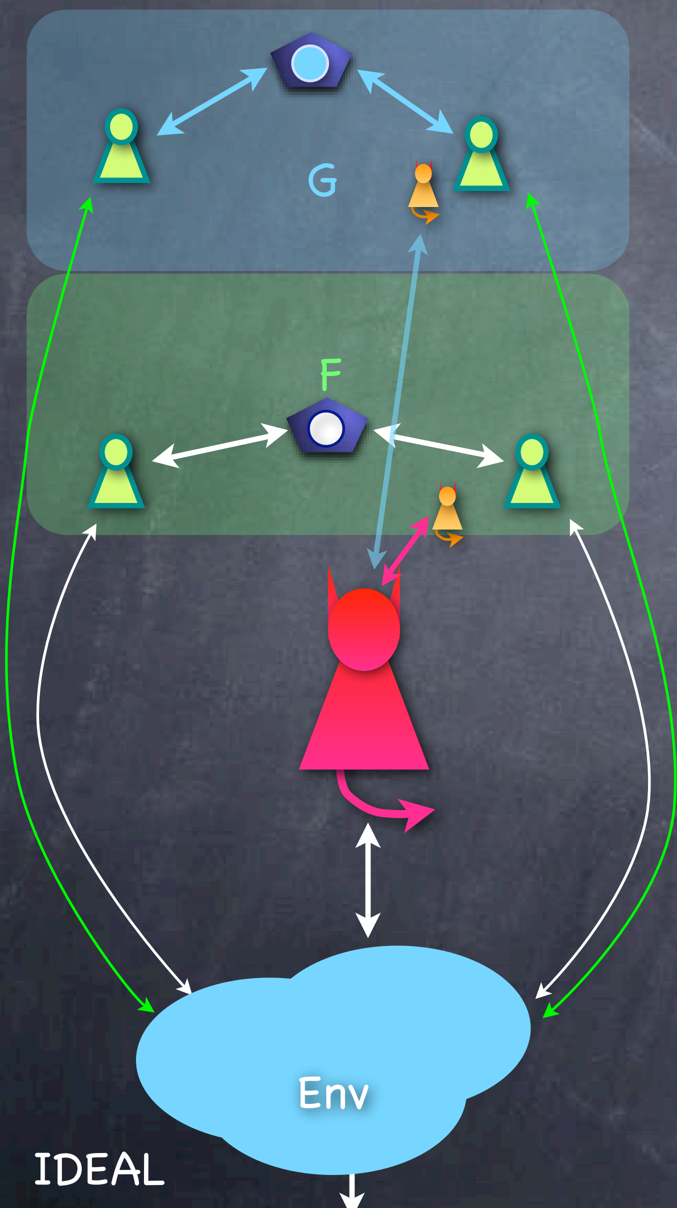


Proving the UC theorem

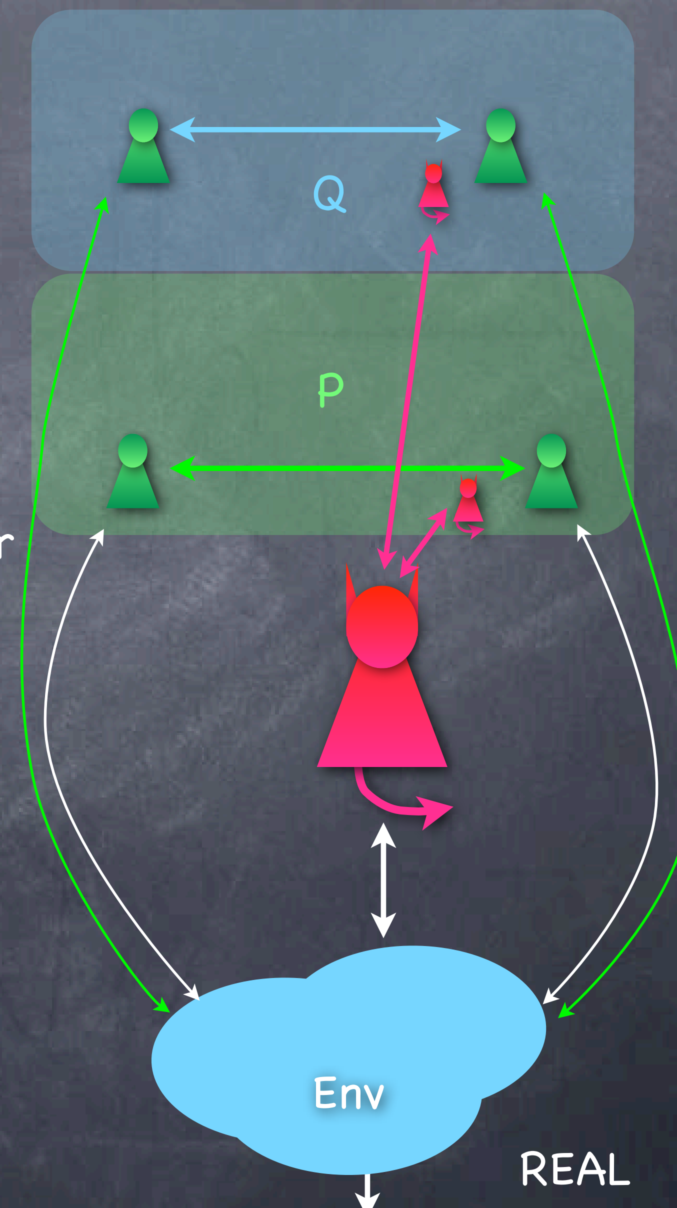




Proving the UC theorem



- Hence REAL ≈ IDEAL
- Main idea: Environment can model other sessions (real or ideal)



Secure MPC?

Secure MPC?

- SIM-security is a strong security definition, and also enjoys the UC property

Secure MPC?

- SIM-security is a strong security definition, and also enjoys the UC property
- But impossible to have "non-trivial" SIM-secure MPC!

Secure MPC?

- SIM-security is a strong security definition, and also enjoys the UC property
- But impossible to have "non-trivial" SIM-secure MPC!
- Possible when:

Secure MPC?

- SIM-security is a strong security definition, and also enjoys the UC property
- But impossible to have "non-trivial" SIM-secure MPC!
- Possible when:
  - Passive corruption

Secure MPC?

- SIM-security is a strong security definition, and also enjoys the UC property
- But impossible to have "non-trivial" SIM-secure MPC!
- Possible when:
  - Passive corruption
  - Honest majority

Secure MPC?

- SIM-security is a strong security definition, and also enjoys the UC property
- But impossible to have "non-trivial" SIM-secure MPC!
- Possible when:
  - Passive corruption
  - Honest majority
  - Given trusted setups

Secure MPC?

- SIM-security is a strong security definition, and also enjoys the UC property
- But impossible to have "non-trivial" SIM-secure MPC!
- Possible when:
  - Passive corruption
  - Honest majority
  - Given trusted setups
  - Using alternate security definition (e.g., "Angel-aided simulation": still meaningful and UC)

Secure MPC?

- SIM-security is a strong security definition, and also enjoys the UC property
- But impossible to have "non-trivial" SIM-secure MPC!
- Possible when:
  - Passive corruption
  - Honest majority
  - Given trusted setups
  - Using alternate security definition (e.g., "Angel-aided simulation": still meaningful and UC)

Secure MPC?

- SIM-security is a strong security definition, and also enjoys the UC property
- But impossible to have "non-trivial" SIM-secure MPC!
- Possible when:
  - Passive corruption
  - Honest majority
  - Given trusted setups
  - Using alternate security definition (e.g., "Angel-aided simulation": still meaningful and UC)

UC-Secure MPC

UC-Secure MPC

- Secure against passive corruption

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"
    - Use OT (realizable, for passive corruption, using TOWP)

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"
    - Use OT (realizable, for passive corruption, using TOWP)
- Turn it into a protocol secure against active corruption

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"
    - Use OT (realizable, for passive corruption, using TOWP)
- Turn it into a protocol secure against active corruption
  - Using a trusted "commit-and-prove" (CaP) functionality

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"
    - Use OT (realizable, for passive corruption, using TOWP)
- Turn it into a protocol secure against active corruption
  - Using a trusted "commit-and-prove" (CaP) functionality
  - CaP not realizable (for active corruption) in the plain model

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"
    - Use OT (realizable, for passive corruption, using TOWP)
- Turn it into a protocol secure against active corruption
  - Using a trusted "commit-and-prove" (CaP) functionality
  - CaP not realizable (for active corruption) in the plain model
    - Realizable using some other "simpler" trusted setups

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"
    - Use OT (realizable, for passive corruption, using TOWP)
- Turn it into a protocol secure against active corruption
  - Using a trusted "commit-and-prove" (CaP) functionality
  - CaP not realizable (for active corruption) in the plain model
    - Realizable using some other "simpler" trusted setups
      - e.g.: trusted party just provides random strings

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"
    - Use OT (realizable, for passive corruption, using TOWP)
- Turn it into a protocol secure against active corruption
  - Using a trusted "commit-and-prove" (CaP) functionality
  - CaP not realizable (for active corruption) in the plain model
    - Realizable using some other "simpler" trusted setups
      - e.g.: trusted party just provides random strings

Shared Evaluation (GMW)

Shared Evaluation (GMW)

- (Against passive corruption)

Shared Evaluation (GMW)

- (Against passive corruption)
- Circuit for evaluating the function: AND (.) and XOR (+) gates

Shared Evaluation (GMW)

- (Against passive corruption)
- Circuit for evaluating the function: AND (.) and XOR (+) gates
- Plan: "Compute" the value on each wire of the circuit, bottom-up

Shared Evaluation (GMW)

- (Against passive corruption)
- Circuit for evaluating the function: AND (.) and XOR (+) gates
- Plan: "Compute" the value on each wire of the circuit, bottom-up
  - After computing a wire value x, for each $i = 1$ to m, party i has a random share $x^{(i)}$ such that $x^{(1)} + x^{(2)} + \ldots + x^{(m)} = x$

Shared Evaluation (GMW)

- (Against passive corruption)
- Circuit for evaluating the function: AND (.) and XOR (+) gates
- Plan: "Compute" the value on each wire of the circuit, bottom-up
  - After computing a wire value x, for each i = 1 to m, party i has a random share $x^{(i)}$ such that $x^{(1)} + x^{(2)} +\ldots+ x^{(m)} = x$
- Initialization: for each input wire, its shares are generated by the party owning that wire, and sent to every party

Shared Evaluation (GMW)

- (Against passive corruption)
- Circuit for evaluating the function: AND (.) and XOR (+) gates
- Plan: "Compute" the value on each wire of the circuit, bottom-up
  - After computing a wire value x, for each i = 1 to m, party i has a random share $x^{(i)}$ such that $x^{(1)} + x^{(2)} + \ldots + x^{(m)} = x$
- Initialization: for each input wire, its shares are generated by the party owning that wire, and sent to every party
- XOR evaluations done locally: if $z=x+y$ e.g. $z^{(i)}=x^{(i)}+y^{(i)}$

Shared Evaluation (GMW)

- (Against passive corruption)
- Circuit for evaluating the function: AND (.) and XOR (+) gates
- Plan: "Compute" the value on each wire of the circuit, bottom-up
  - After computing a wire value x, for each i = 1 to m, party i has a random share $x^{(i)}$ such that $x^{(1)} + x^{(2)} +...+ x^{(m)} = x$
- Initialization: for each input wire, its shares are generated by the party owning that wire, and sent to every party
- XOR evaluations done locally: if $z=x+y$ e.g. $z^{(i)}=x^{(i)}+y^{(i)}$
- For AND: need $z^{(1)} + z^{(2)} +...+ z^{(m)} = [x^{(1)}+...+x^{(m)}]\ [y^{(1)}+...+y^{(m)}]$ (and $z^{(i)}$ random otherwise). Will use OT.

Shared Evaluation (GMW)

- For AND: need $z^{(1)} + z^{(2)} +\ldots+ z^{(m)} = [x^{(1)}+\ldots+x^{(m)}]\ [y^{(1)}+\ldots+y^{(m)}]$

Shared Evaluation (GMW)

- For AND: need $z^{(1)} + z^{(2)} +\ldots+ z^{(m)} = [x^{(1)}+\ldots+x^{(m)}]\ [y^{(1)}+\ldots+y^{(m)}]$
- e.g.: m=2. Then, need $z^{(1)} + z^{(2)} = [x^{(1)}+x^{(2)}]\ [y^{(1)}+y^{(2)}]$

Shared Evaluation (GMW)

- For AND: need $z^{(1)} + z^{(2)} +\ldots+ z^{(m)} = [x^{(1)}+\ldots+x^{(m)}]\ [y^{(1)}+\ldots+y^{(m)}]$
- e.g.: m=2. Then, need $z^{(1)} + z^{(2)} = [x^{(1)}+x^{(2)}]\ [y^{(1)}+y^{(2)}]$
  - Party 1 sets $z^{(1)}$ to be random, and sets $(w_{00},w_{01},w_{10},w_{11})$ such that if $(x^{(2)},y^{(2)}) = (a,b)$, then $z^{(1)}+w_{ab} = [x^{(1)}+x^{(2)}]\ [y^{(1)}+y^{(2)}]$

Shared Evaluation (GMW)

- For AND: need $z^{(1)} + z^{(2)} +\ldots+ z^{(m)} = [x^{(1)}+\ldots+x^{(m)}]\,[y^{(1)}+\ldots+y^{(m)}]$
- e.g.: m=2. Then, need $z^{(1)} + z^{(2)} = [x^{(1)}+x^{(2)}]\,[y^{(1)}+y^{(2)}]$
  - Party 1 sets $z^{(1)}$ to be random, and sets $(w_{00},w_{01},w_{10},w_{11})$ such that if $(x^{(2)},y^{(2)}) = (a,b)$, then $z^{(1)}+w_{ab} = [x^{(1)}+x^{(2)}]\,[y^{(1)}+y^{(2)}]$
    - Party 2 picks up w_{ab} for $(a,b)= (x^{(2)},y^{(2)})$ using "1-out-of-4" OT

Shared Evaluation (GMW)

- For AND: need $z^{(1)} + z^{(2)} + ... + z^{(m)} = [x^{(1)}+...+x^{(m)}]\,[y^{(1)}+...+y^{(m)}]$
- e.g.: m=2. Then, need $z^{(1)} + z^{(2)} = [x^{(1)}+x^{(2)}]\,[y^{(1)}+y^{(2)}]$
  - Party 1 sets $z^{(1)}$ to be random, and sets $(w_{00},w_{01},w_{10},w_{11})$ such that if $(x^{(2)},y^{(2)}) = (a,b)$, then $z^{(1)}+w_{ab} = [x^{(1)}+x^{(2)}]\,[y^{(1)}+y^{(2)}]$
    - Party 2 picks up w_{ab} for $(a,b)= (x^{(2)},y^{(2)})$ using "1-out-of-4" OT

Can be implemented using 1-out-of-2 OT [Exercise]

Shared Evaluation (GMW)

- For AND: need $z^{(1)} + z^{(2)} + \ldots + z^{(m)} = [x^{(1)}+\ldots+x^{(m)}]\ [y^{(1)}+\ldots+y^{(m)}]$
- e.g.: m=2. Then, need $z^{(1)} + z^{(2)} = [x^{(1)}+x^{(2)}]\ [y^{(1)}+y^{(2)}]$
  - Party 1 sets $z^{(1)}$ to be random, and sets $(w_{00},w_{01},w_{10},w_{11})$ such that if $(x^{(2)},y^{(2)}) = (a,b)$, then $z^{(1)}+w_{ab} = [x^{(1)}+x^{(2)}]\ [y^{(1)}+y^{(2)}]$
    - Party 2 picks up w_{ab} for $(a,b)= (x^{(2)},y^{(2)})$ using "1-out-of-4" OT
    - Can do it slightly more efficiently using 2 1-out-of-2 OTs

Can be implemented using 1-out-of-2 OT [Exercise]

Shared Evaluation (GMW)

- For AND: need $z^{(1)} + z^{(2)} + \ldots + z^{(m)} = [x^{(1)}+\ldots+x^{(m)}]\,[y^{(1)}+\ldots+y^{(m)}]$
- e.g.: m=2. Then, need $z^{(1)} + z^{(2)} = [x^{(1)}+x^{(2)}]\,[y^{(1)}+y^{(2)}]$

Can be implemented using 1-out-of-2 OT [Exercise]

  - Party 1 sets $z^{(1)}$ to be random, and sets $(w_{00},w_{01},w_{10},w_{11})$ such that if $(x^{(2)},y^{(2)}) = (a,b)$, then $z^{(1)}+w_{ab} = [x^{(1)}+x^{(2)}]\,[y^{(1)}+y^{(2)}]$
    - Party 2 picks up w_{ab} for $(a,b)= (x^{(2)},y^{(2)})$ using "1-out-of-4" OT
    - Can do it slightly more efficiently using 2 1-out-of-2 OTs
- Finally, the parties are holding shares of values of output wires

Shared Evaluation (GMW)

- For AND: need $z^{(1)} + z^{(2)} + \ldots + z^{(m)} = [x^{(1)}+\ldots+x^{(m)}]\ [y^{(1)}+\ldots+y^{(m)}]$
- e.g.: m=2. Then, need $z^{(1)} + z^{(2)} = [x^{(1)}+x^{(2)}]\ [y^{(1)}+y^{(2)}]$

Can be implemented using 1-out-of-2 OT [Exercise]

  - Party 1 sets $z^{(1)}$ to be random, and sets $(w_{00},w_{01},w_{10},w_{11})$ such that if $(x^{(2)},y^{(2)}) = (a,b)$, then $z^{(1)}+w_{ab} = [x^{(1)}+x^{(2)}]\ [y^{(1)}+y^{(2)}]$
    - Party 2 picks up w_{ab} for $(a,b)= (x^{(2)},y^{(2)})$ using "1-out-of-4" OT
    - Can do it slightly more efficiently using 2 1-out-of-2 OTs
- Finally, the parties are holding shares of values of output wires
- Reconstruct the output: all parties send their shares of the output wire for party i to that party. Party i adds up all the shares of that output wire.

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"
    - Use OT (realizable, for passive corruption, using TOWP)
- Turn it into secure against active corruption
  - Using a trusted "commit-and-prove" (CaP) functionality
  - CaP not realizable (for active corruption) in the plain model
    - Realizable using some other "simpler" trusted setups
      - e.g.: trusted party just provides random strings

UC-Secure MPC

- Secure against passive corruption
  - 2-Party: Yao's Garbled circuit
  - General Multi-party: "Shared Evaluation"
    - Use OT (realizable, for passive corruption, using TOWP)
- Turn it into secure against active corruption
  - Using a trusted "commit-and-prove" (CaP) functionality
  - CaP not realizable (for active corruption) in the plain model
    - Realizable using some other "simpler" trusted setups
      - e.g.: trusted party just provides random strings

Security against active corruption

Security against active corruption

- Commit-and-Prove (CaP) functionality

Security against active corruption

- Commit-and-Prove (CaP) functionality
  - Party i can send x; all parties get the message "committed"

Security against active corruption

- Commit-and-Prove (CaP) functionality
  - Party i can send x; all parties get the message "committed"
  - Later send a statement R s.t. R(x) holds; all parties get R

Security against active corruption

- Commit-and-Prove (CaP) functionality
  - Party i can send x; all parties get the message "committed"
  - Later send a statement R s.t. R(x) holds; all parties get R
    - e.g. $R_{f,y}(x)$: $f(x)=y$ ($f\equiv id$ corresponds to opening)

Security against active corruption

- Commit-and-Prove (CaP) functionality
  - Party i can send x; all parties get the message "committed"
  - Later send a statement R s.t. R(x) holds; all parties get R
    - e.g. $R_{f,y}(x)$: f(x)=y (f≡id corresponds to opening)
- Can use for "coin-tossing into the well"

Security against active corruption

- Commit-and-Prove (CaP) functionality
  - Party i can send x; all parties get the message "committed"
  - Later send a statement R s.t. R(x) holds; all parties get R
    - e.g. $R_{f,y}(x)$: f(x)=y (f≡id corresponds to opening)
- Can use for "coin-tossing into the well"
  - Only party i gets a random string, but can later prove statements involving that string to the others

Security against active corruption

- Commit-and-Prove (CaP) functionality
  - Party i can send x; all parties get the message "committed"
  - Later send a statement R s.t. R(x) holds; all parties get R
    - e.g. $R_{f,y}(x)$: f(x)=y ($f \equiv id$ corresponds to opening)
- Can use for "coin-tossing into the well"
  - Only party i gets a random string, but can later prove statements involving that string to the others
  - Implementation: All parties commit strings r_j (using CaP); then all except party i open (publicly); let their xor be s; define $r = s + r_i$

Security against active corruption

- Commit-and-Prove (CaP) functionality
  - Party i can send x; all parties get the message "committed"
  - Later send a statement R s.t. R(x) holds; all parties get R
    - e.g. $R_{f,y}(x)$: f(x)=y (f≡id corresponds to opening)
- Can use for "coin-tossing into the well"
  - Only party i gets a random string, but can later prove statements involving that string to the others
  - Implementation: All parties commit strings r_j (using CaP); then all except party i open (publicly); let their xor be s; define $r = s+r_i$
    - Party i can later prove R(r) using $R_s(r_i) := R(r_i \oplus s)$

Security against active corruption

Security against active corruption

- Given protocol P with security against passive corruption, new protocol P* with security against active corruption:

Security against active corruption

- Given protocol P with security against passive corruption, new protocol P* with security against active corruption:
  - Input commitment and random-tape generation phase: coin-tossing into the well using CaP; commit to inputs x_i also

Security against active corruption

- Given protocol P with security against passive corruption, new protocol P* with security against active corruption:
  - Input commitment and random-tape generation phase: coin-tossing into the well using CaP; commit to inputs x_i also
  - Execution phase: Run protocol P using random-tape generated in the first phase. Followup each protocol message with a proof (using CaP) that the message was produced by the protocol

Security against active corruption

- Given protocol P with security against passive corruption, new protocol P* with security against active corruption:
  - Input commitment and random-tape generation phase: coin-tossing into the well using CaP; commit to inputs x_i also
  - Execution phase: Run protocol P using random-tape generated in the first phase. Followup each protocol message with a proof (using CaP) that the message was produced by the protocol
    - This is a statement about the messages so far (publicly known) and randomness and input (both committed using CaP)

Today

Today

- Universal Composition

Today

- Universal Composition
  - SIM security definition gives universal composition

Today

- Universal Composition
  - SIM security definition gives universal composition
- SIM-secure MPC

Today

- Universal Composition
  - SIM security definition gives universal composition
- SIM-secure MPC
  - Possible with various modified SIM-security definitions (still UC)

Today

- Universal Composition
  - SIM security definition gives universal composition
- SIM-secure MPC
  - Possible with various modified SIM-security definitions (still UC)
    - Impossible in the plain model

Today

- Universal Composition
  - SIM security definition gives universal composition
- SIM-secure MPC
  - Possible with various modified SIM-security definitions (still UC)
    - Impossible in the plain model
  - GMW paradigm: first build a protocol that is secure against passive corruption (using OT), and use ZK proofs (or CaP) to transform it to be secure against active corruption

Today

- Universal Composition
  - SIM security definition gives universal composition
- SIM-secure MPC
  - Possible with various modified SIM-security definitions (still UC)
    - Impossible in the plain model
  - GMW paradigm: first build a protocol that is secure against passive corruption (using OT), and use ZK proofs (or CaP) to transform it to be secure against active corruption
    - Very general (but not very efficient)