Universal Composition

Lecture 14

Composition Issues

Composition Issues

- Multiple executions provide new opportunities for the hacker

Composition Issues

- Multiple executions provide new opportunities for the hacker

Play the GM's against each other

Will not lose against both!

Composition Issues

- Multiple executions provide new opportunities for the hacker
  - Person-in-the-middle attack

Play the GM's against each other

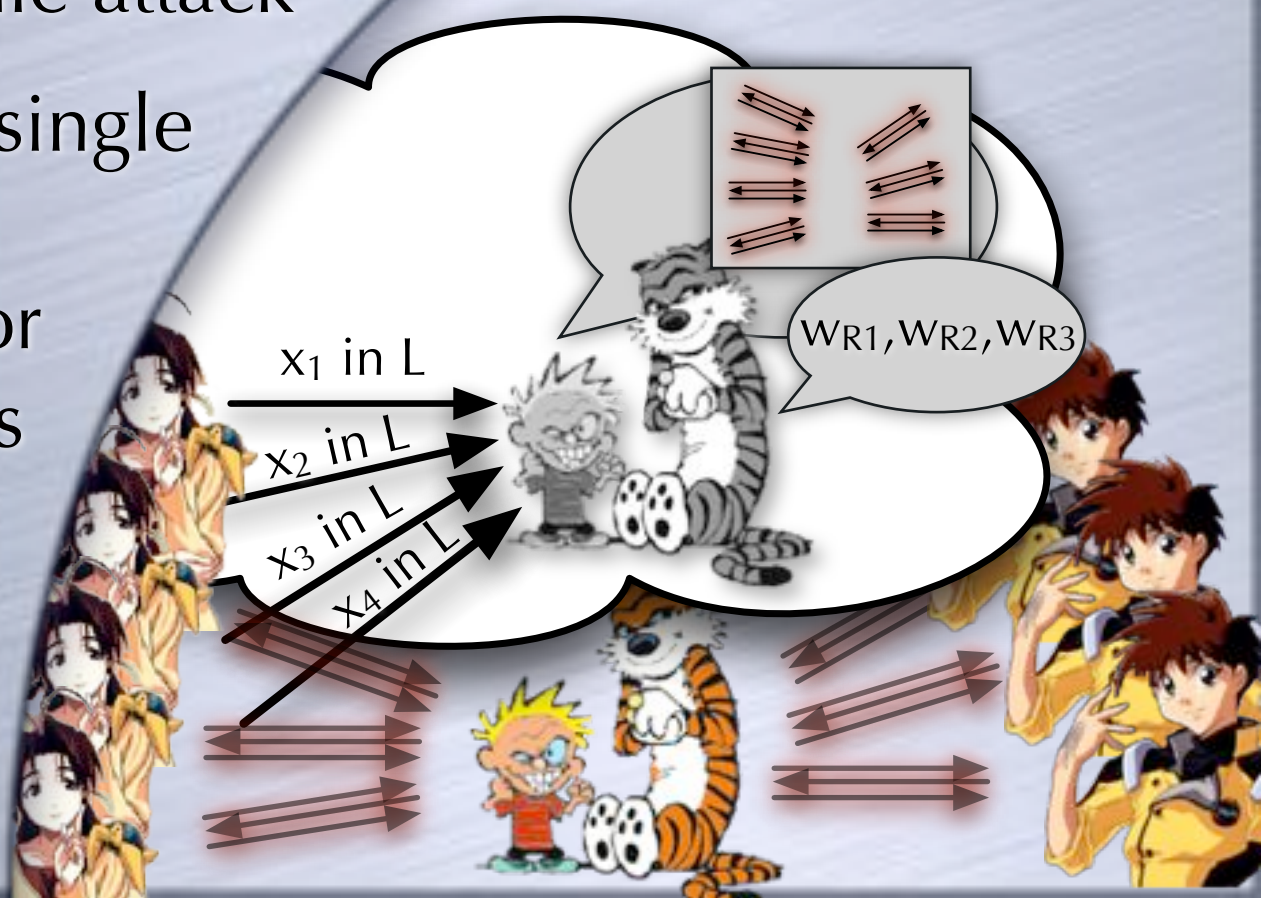
Will not lose against both!

Composition Issues

- Multiple executions provide new opportunities for the hacker
  - Person-in-the-middle attack
  - Simulatability of a single execution doesn't imply simulation for multiple executions

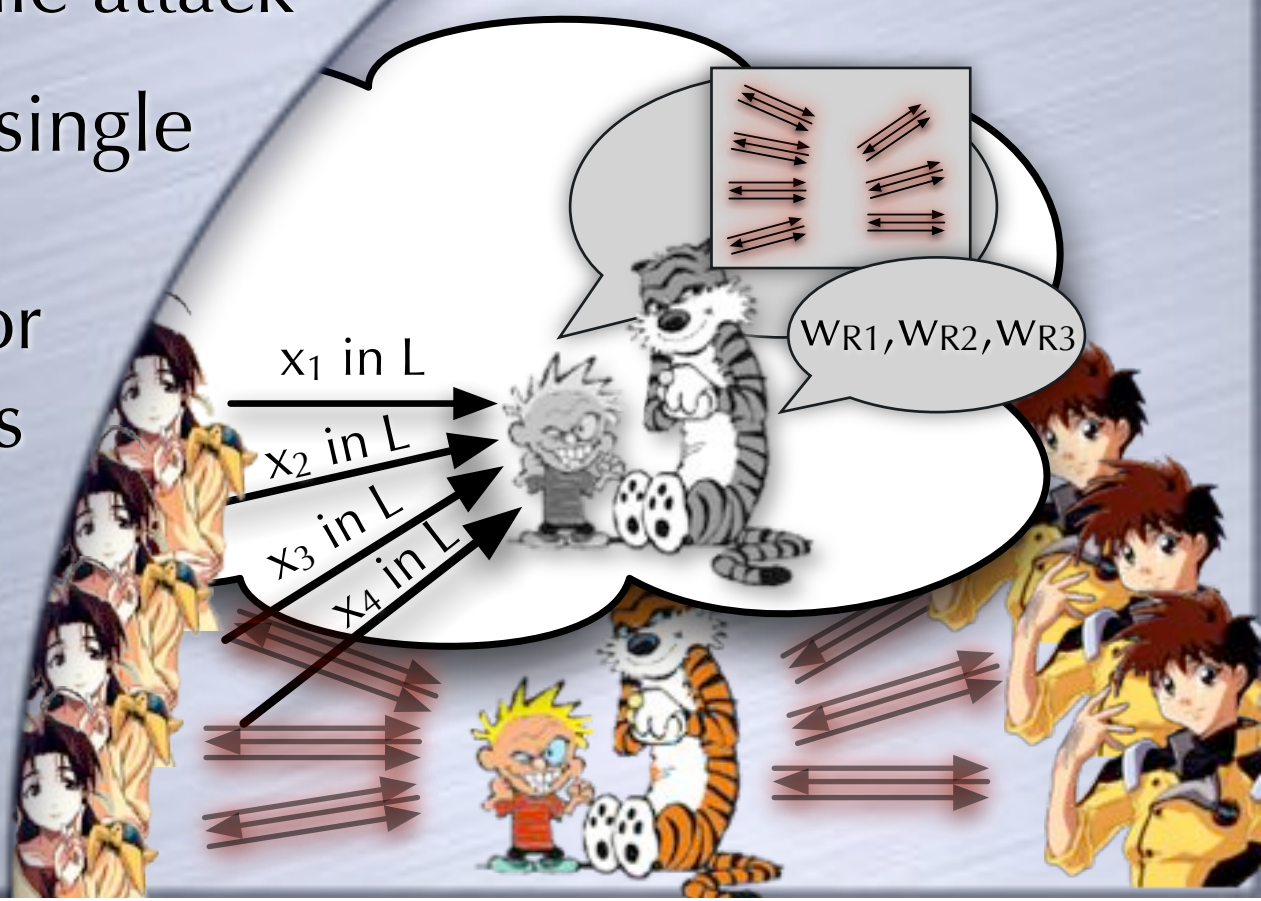
Composition Issues

- Multiple executions provide new opportunities for the hacker
  - Person-in-the-middle attack
  - Simulatability of a single execution doesn't imply simulation for multiple executions



Composition Issues

- Multiple executions provide new opportunities for the hacker
  - Person-in-the-middle attack
  - Simulatability of a single execution doesn't imply simulation for multiple executions
    - Or when run along with other protocols



Universal Composition

Universal Composition

- A security guarantee

Universal Composition

- A security guarantee
  - that can be given for a "composed system"

Universal Composition

- A security guarantee
  - that can be given for a "composed system"
  - such that security for each component separately implies security for the entire system

Universal Composition

- A security guarantee
  - that can be given for a "composed system"
  - such that security for each component separately implies security for the entire system
  - and is meaningful! (otherwise, "everything is secure" is composable)

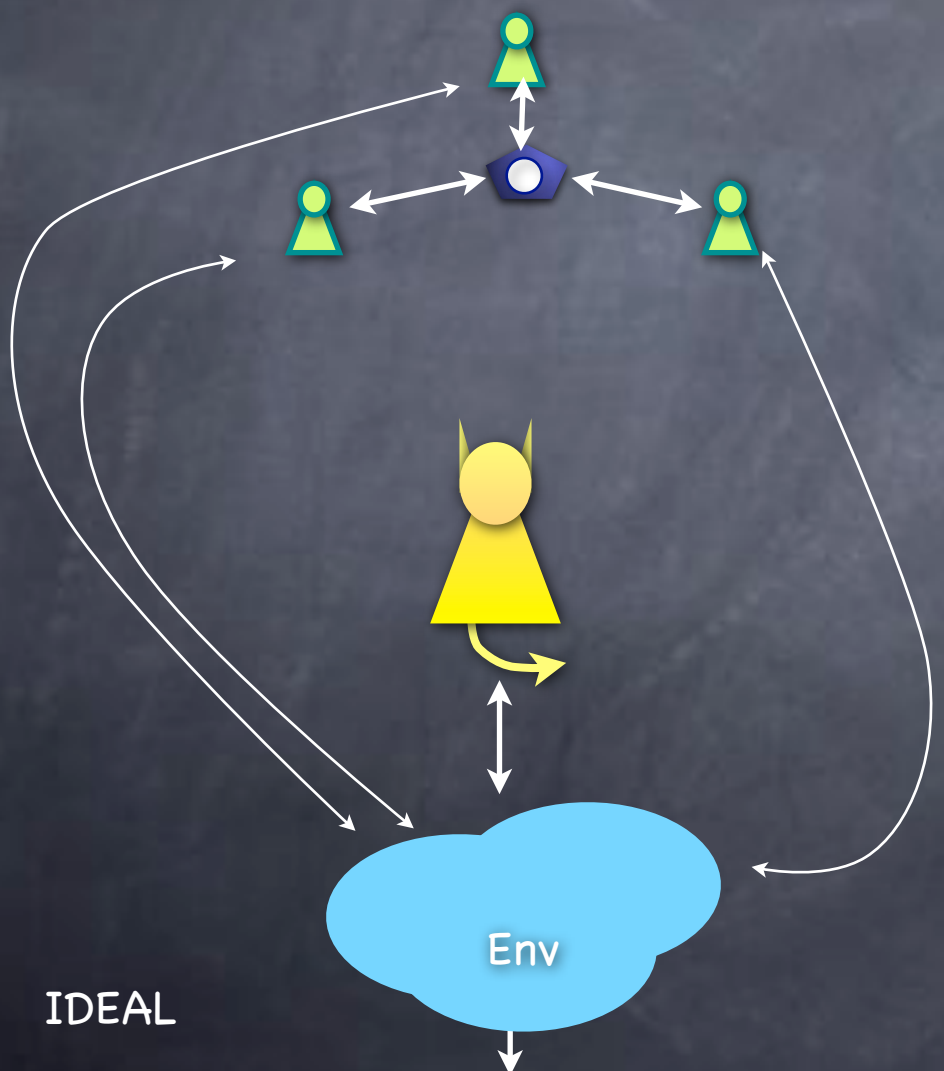
Universal Composition

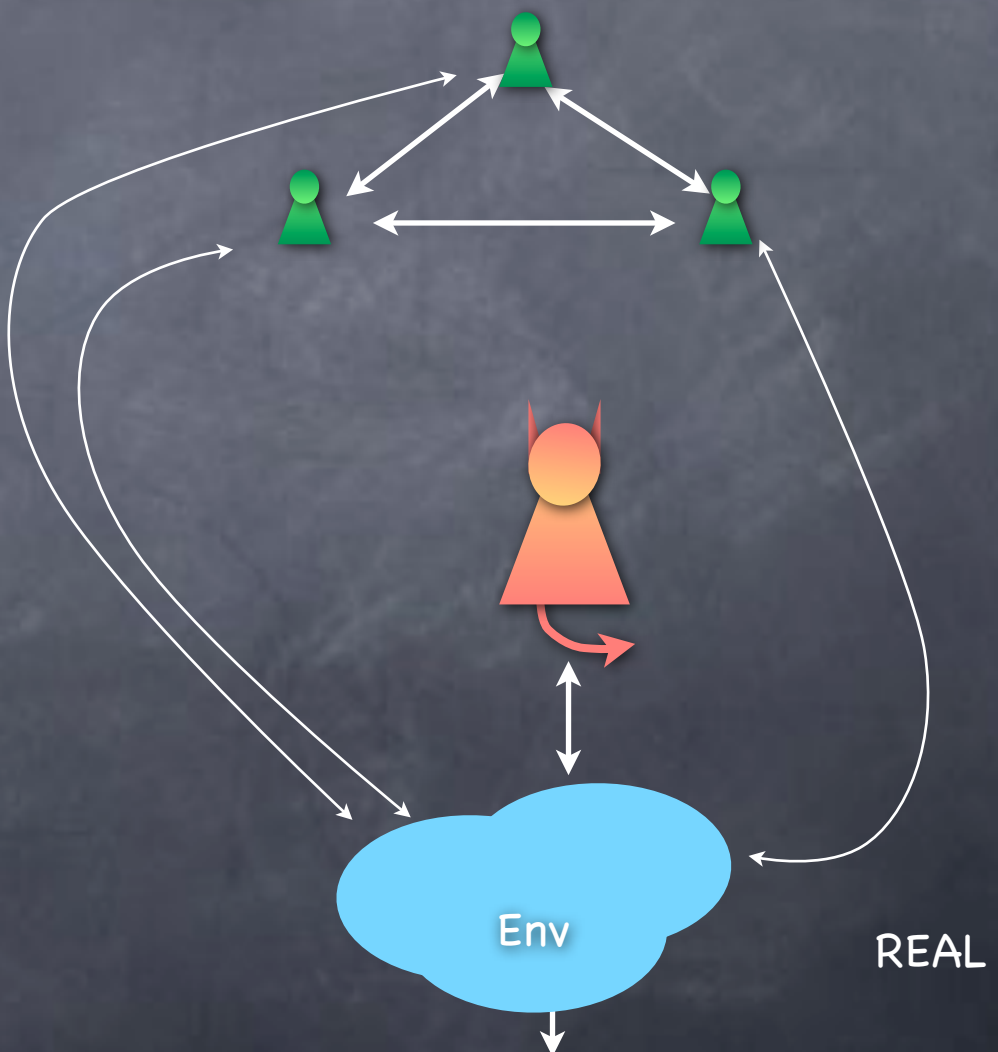
- A security guarantee
  - that can be given for a "composed system"
  - such that security for each component separately implies security for the entire system
  - and is meaningful! (otherwise, "everything is secure" is composable)
    - Will use SIM security

RECALL

Security

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

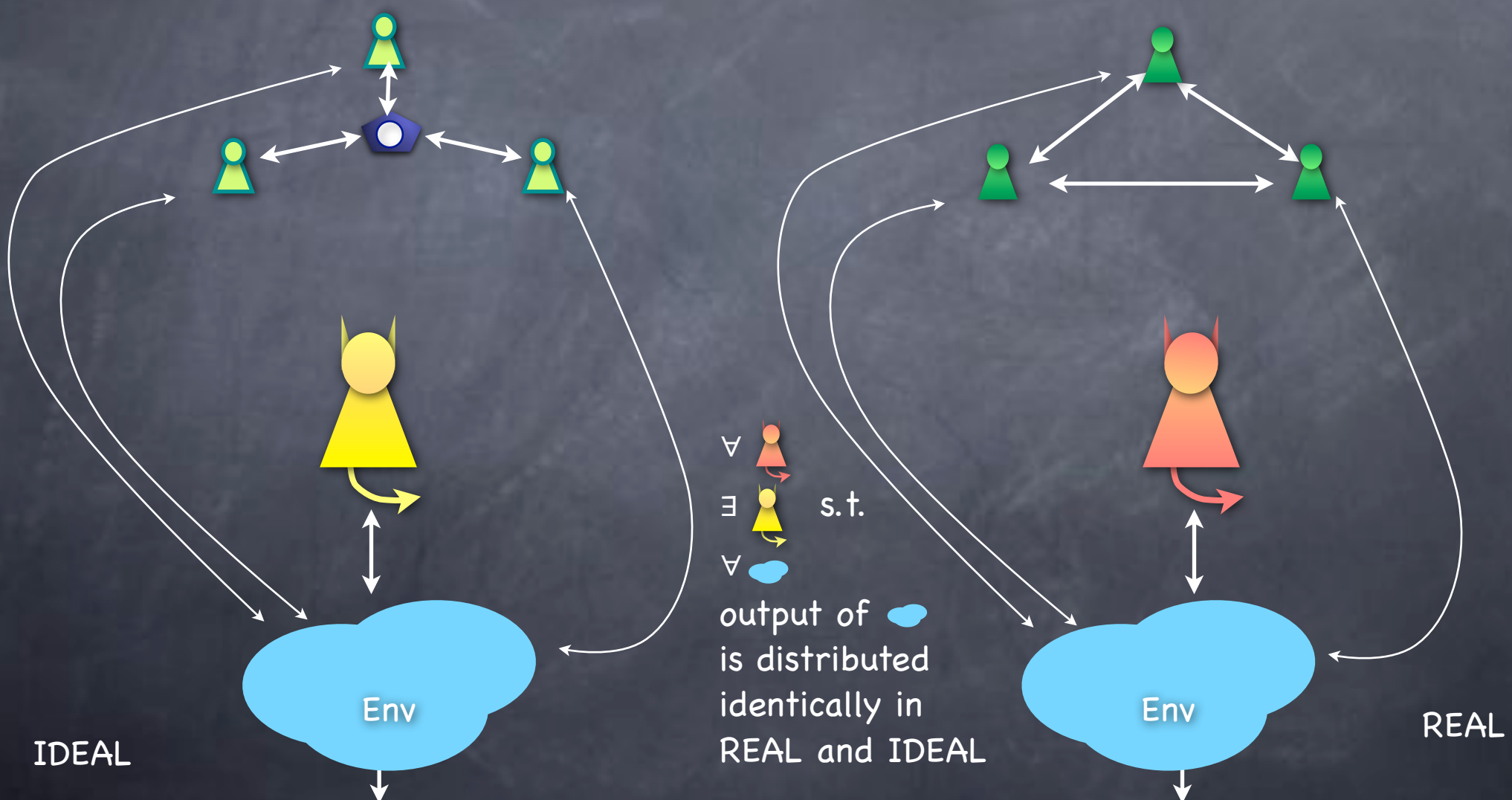




RECALL

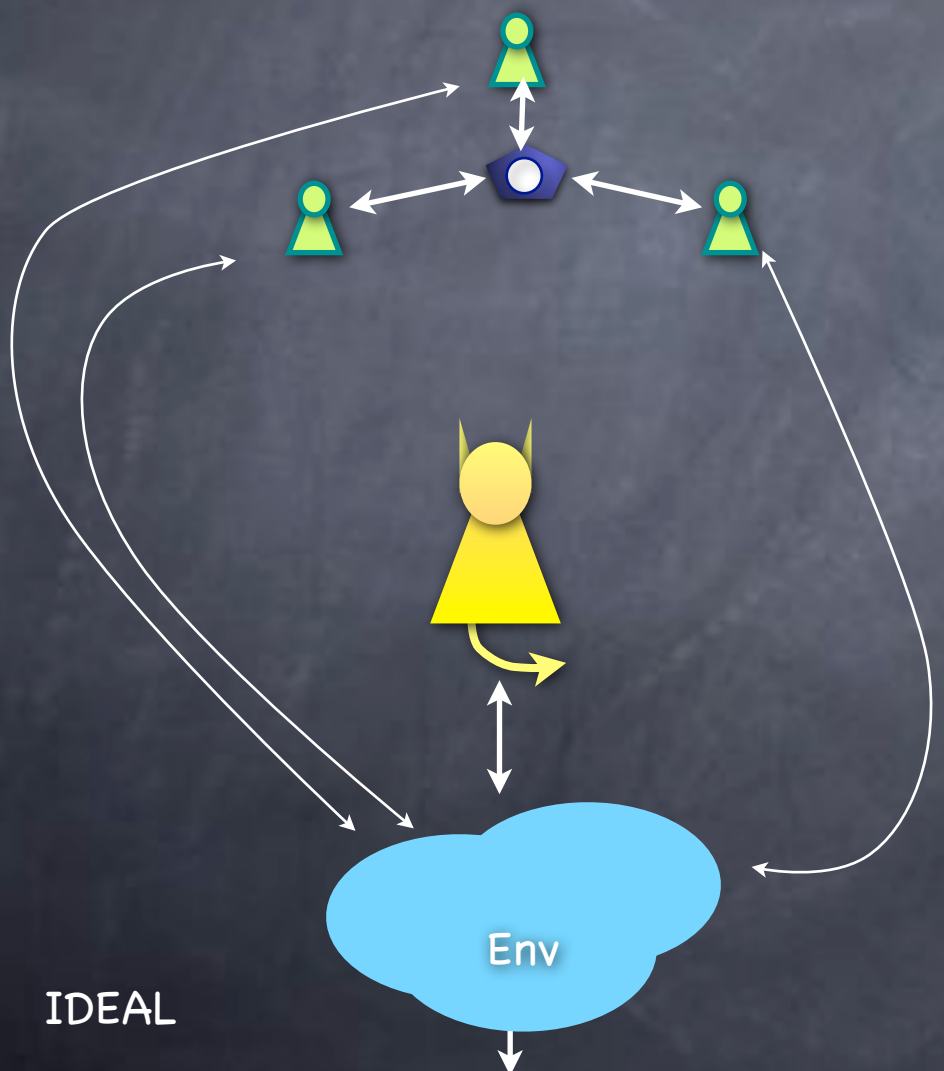
Security

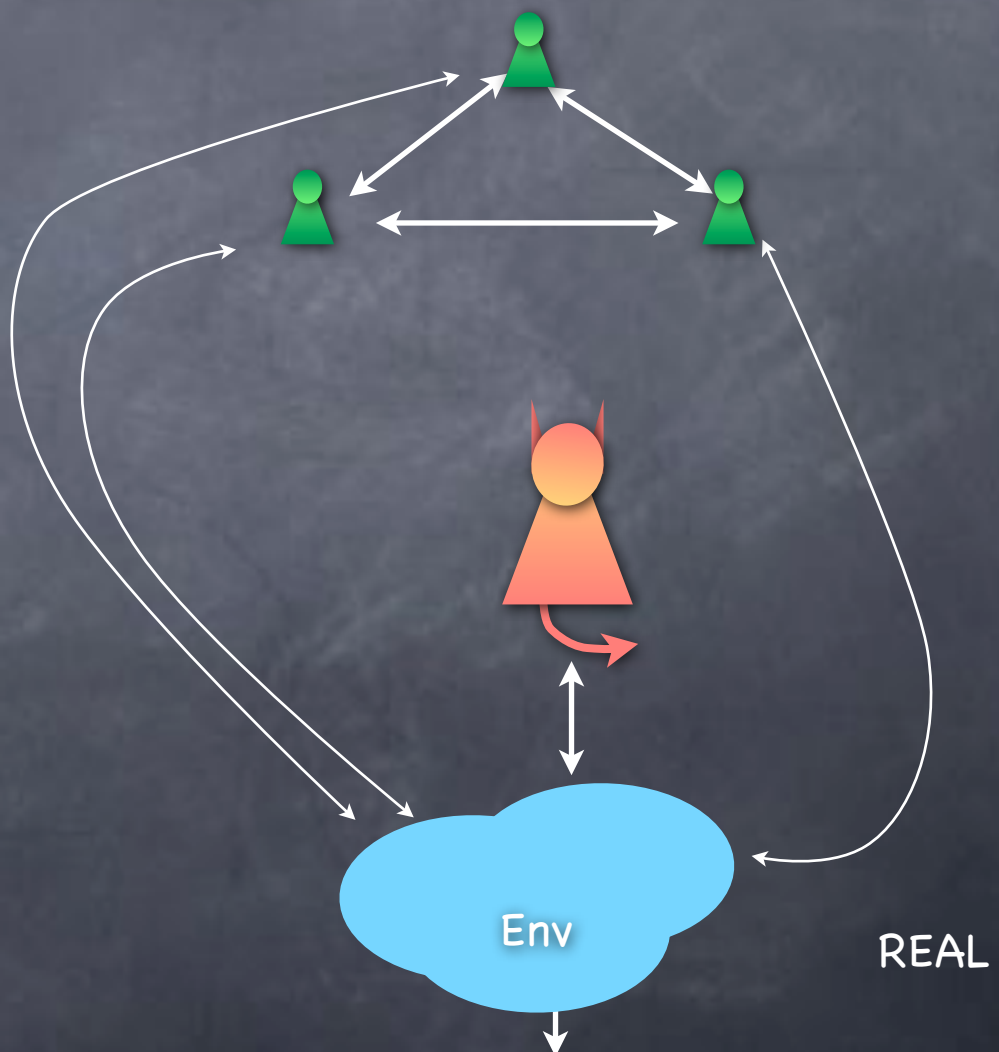
REAL (with protocol) is as secure as IDEAL (with functionality) if:



Security of Composed Systems

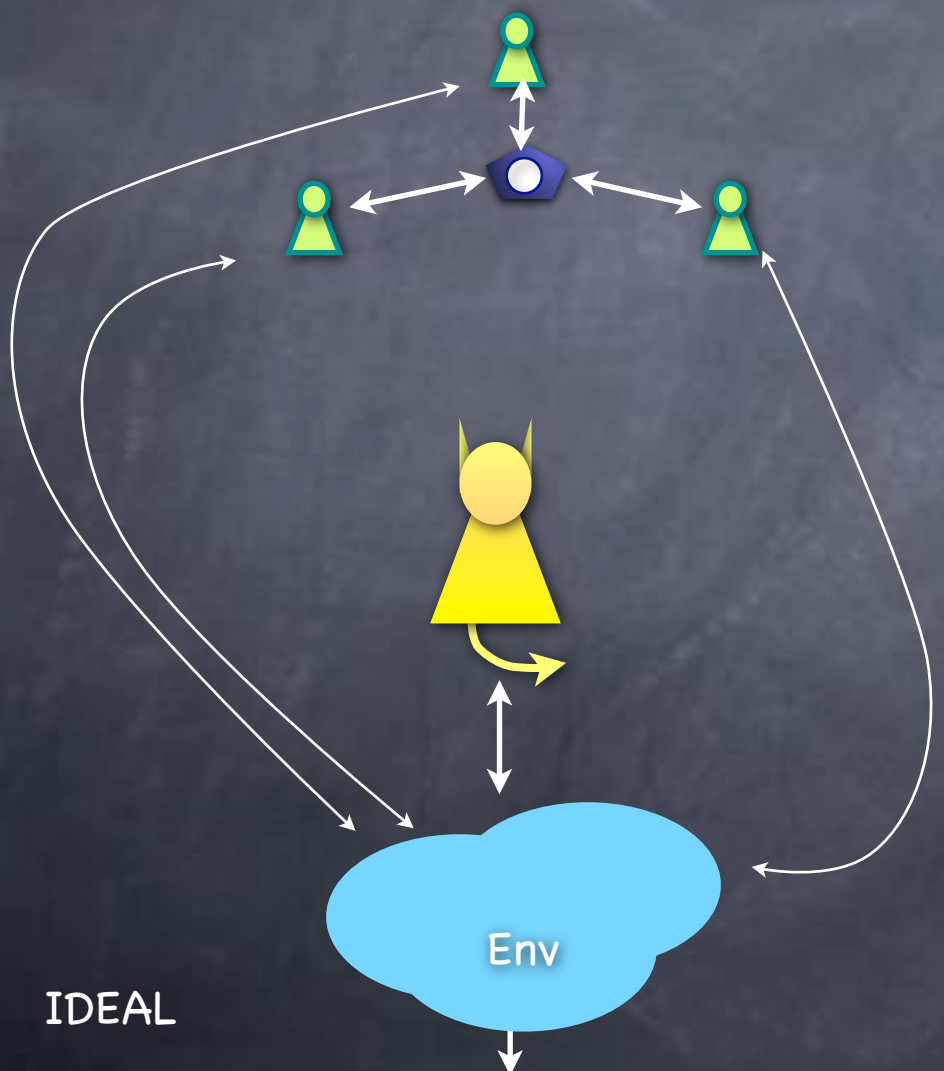
Extend to allow a "composed system" with multiple functionalities

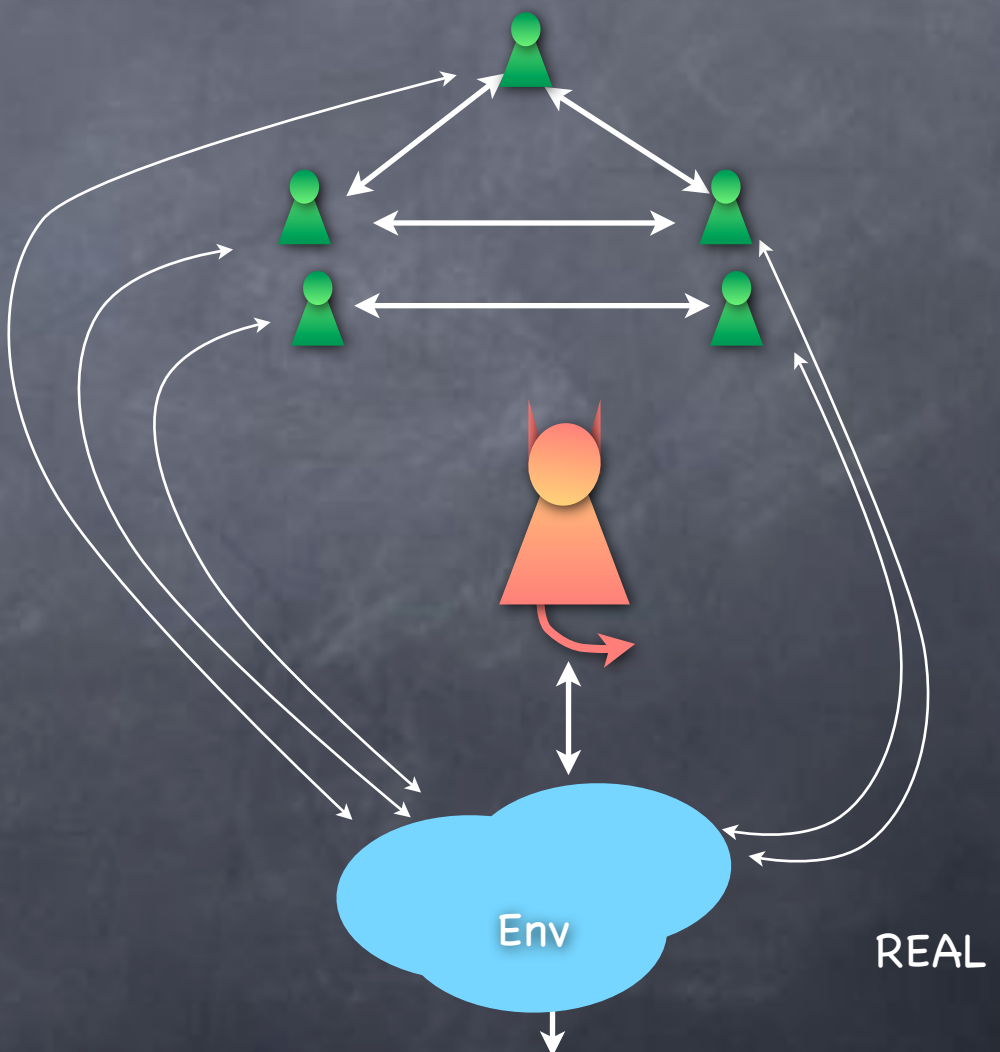




Security of Composed Systems

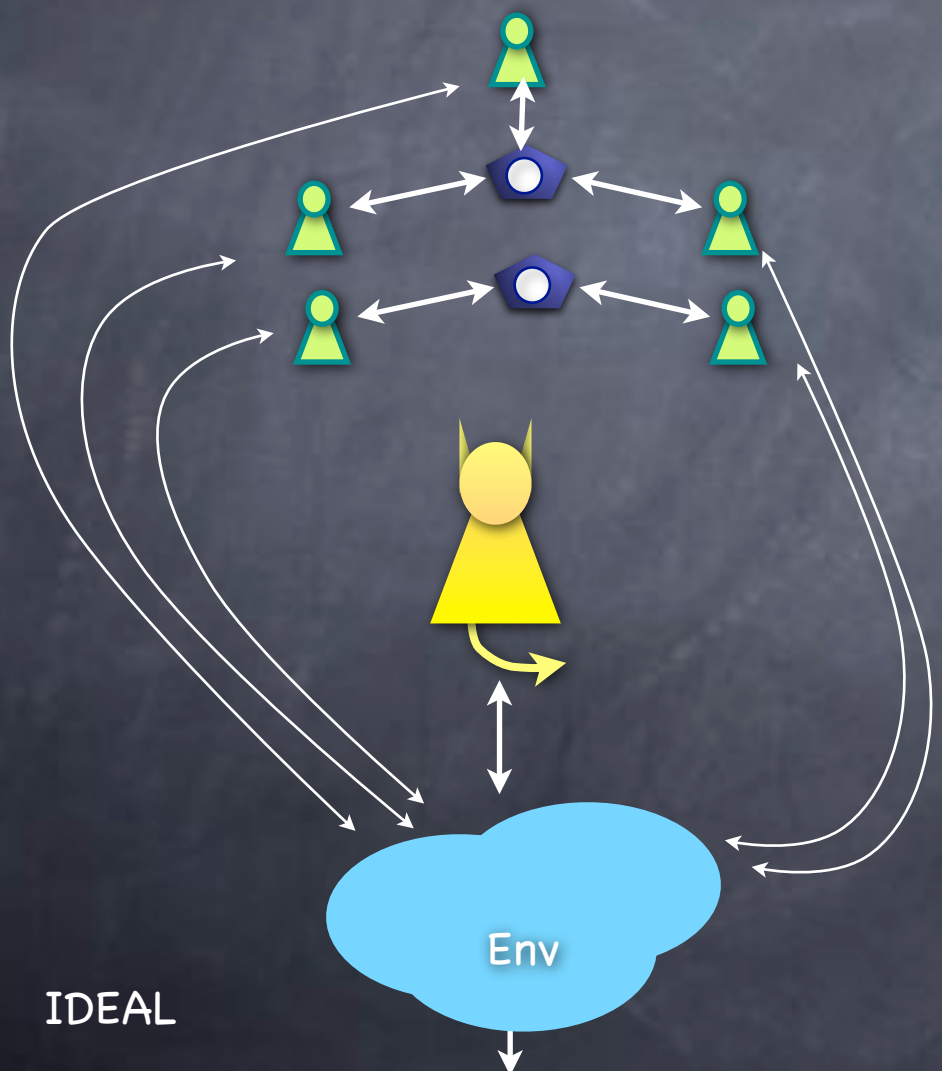
Extend to allow a "composed system" with multiple functionalities

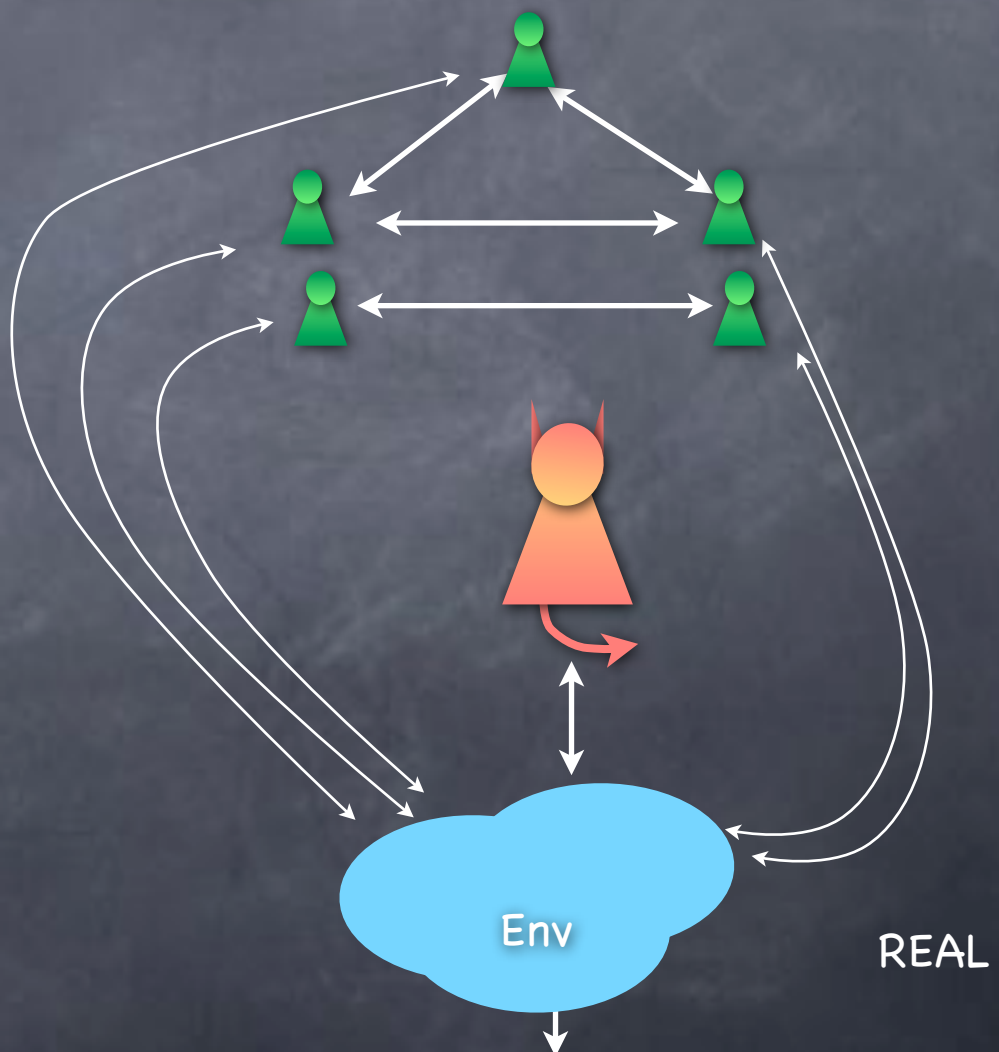




Security of Composed Systems

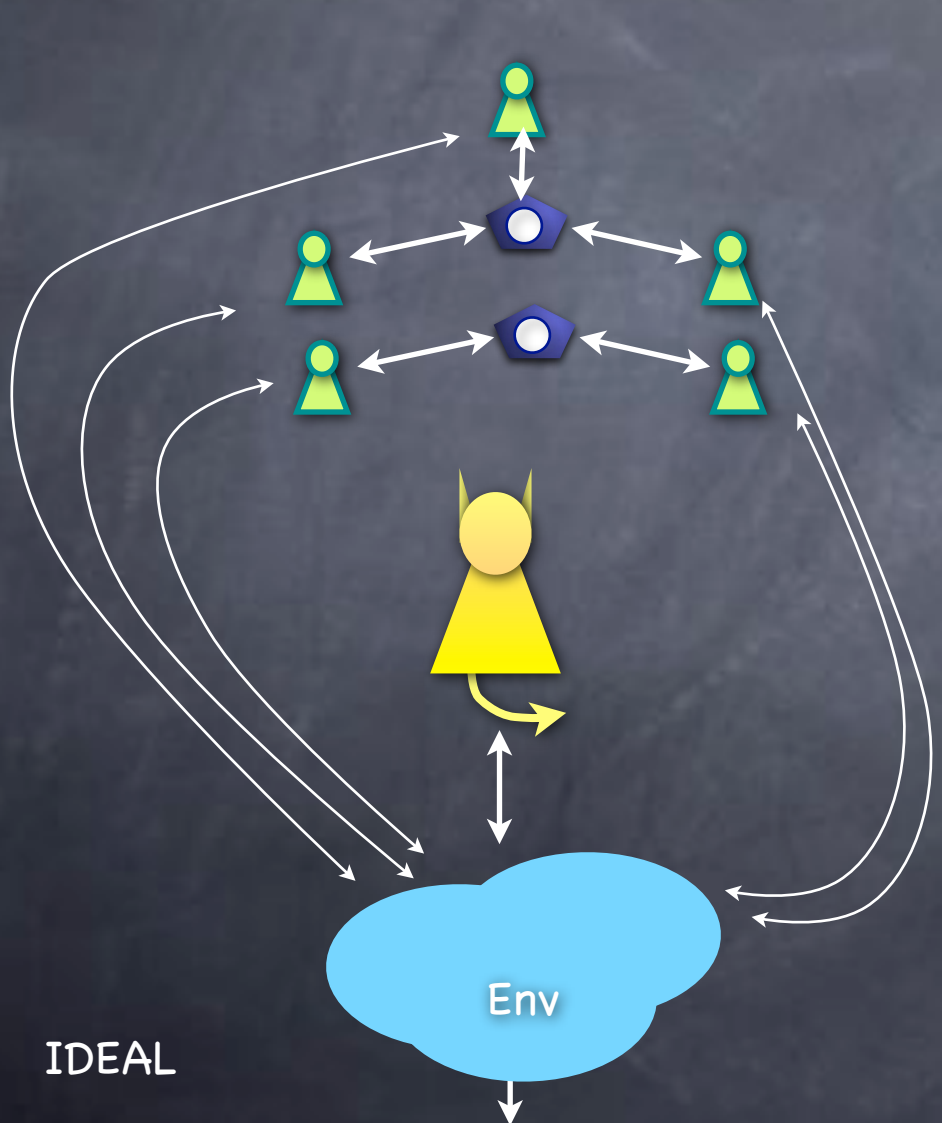
Extend to allow a "composed system" with multiple functionalities

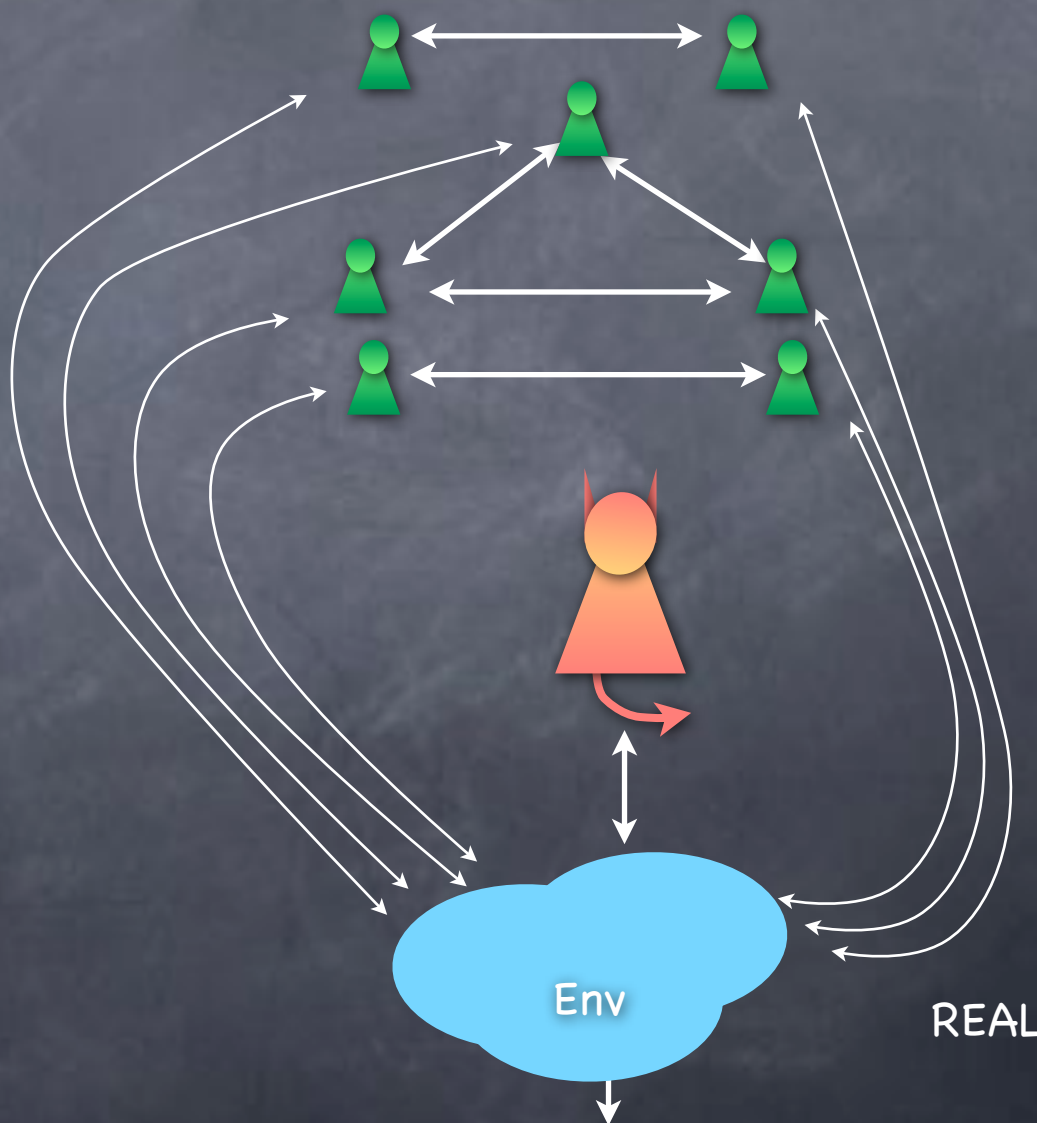




Security of Composed Systems

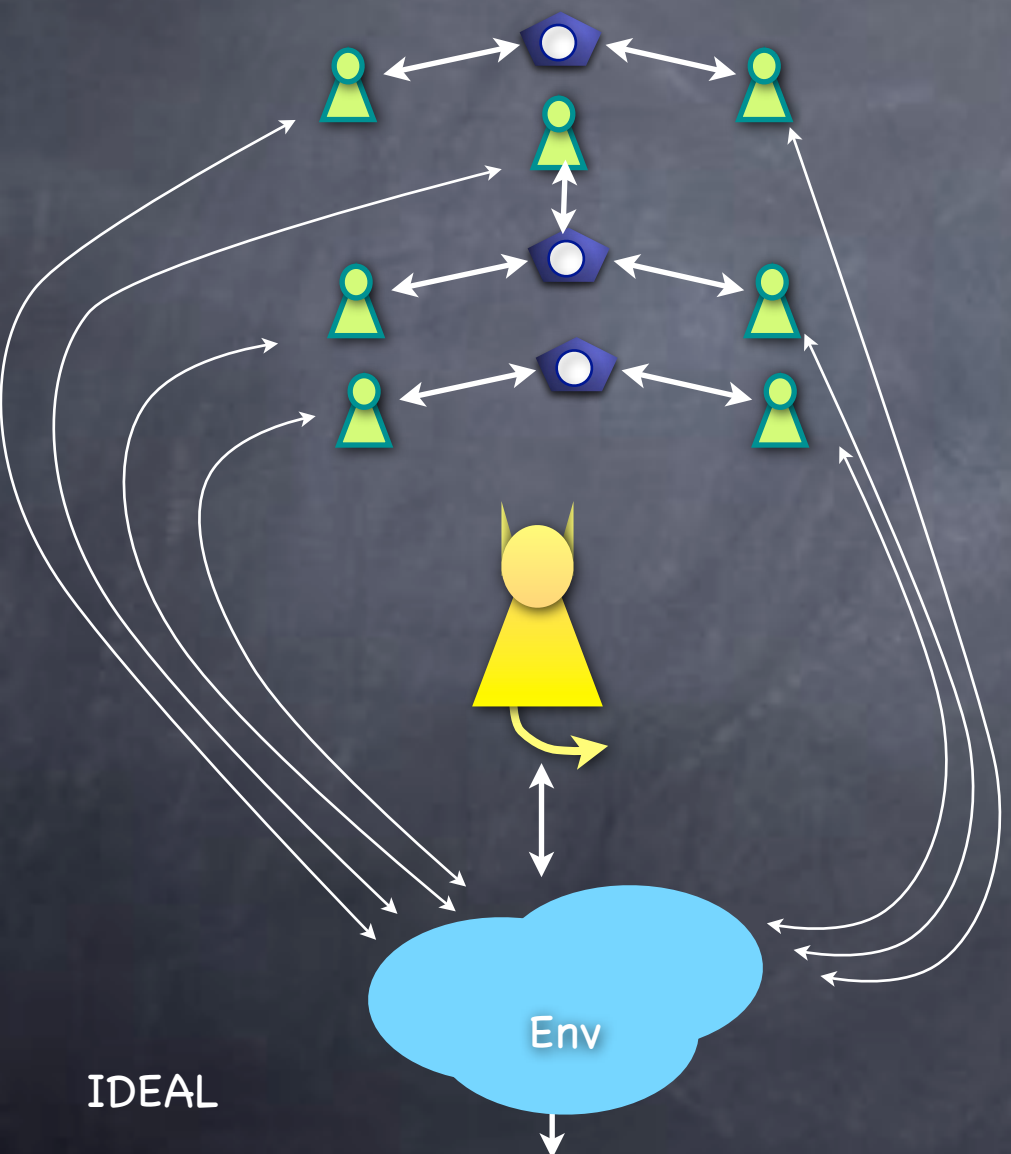
Extend to allow a "composed system" with multiple functionalities

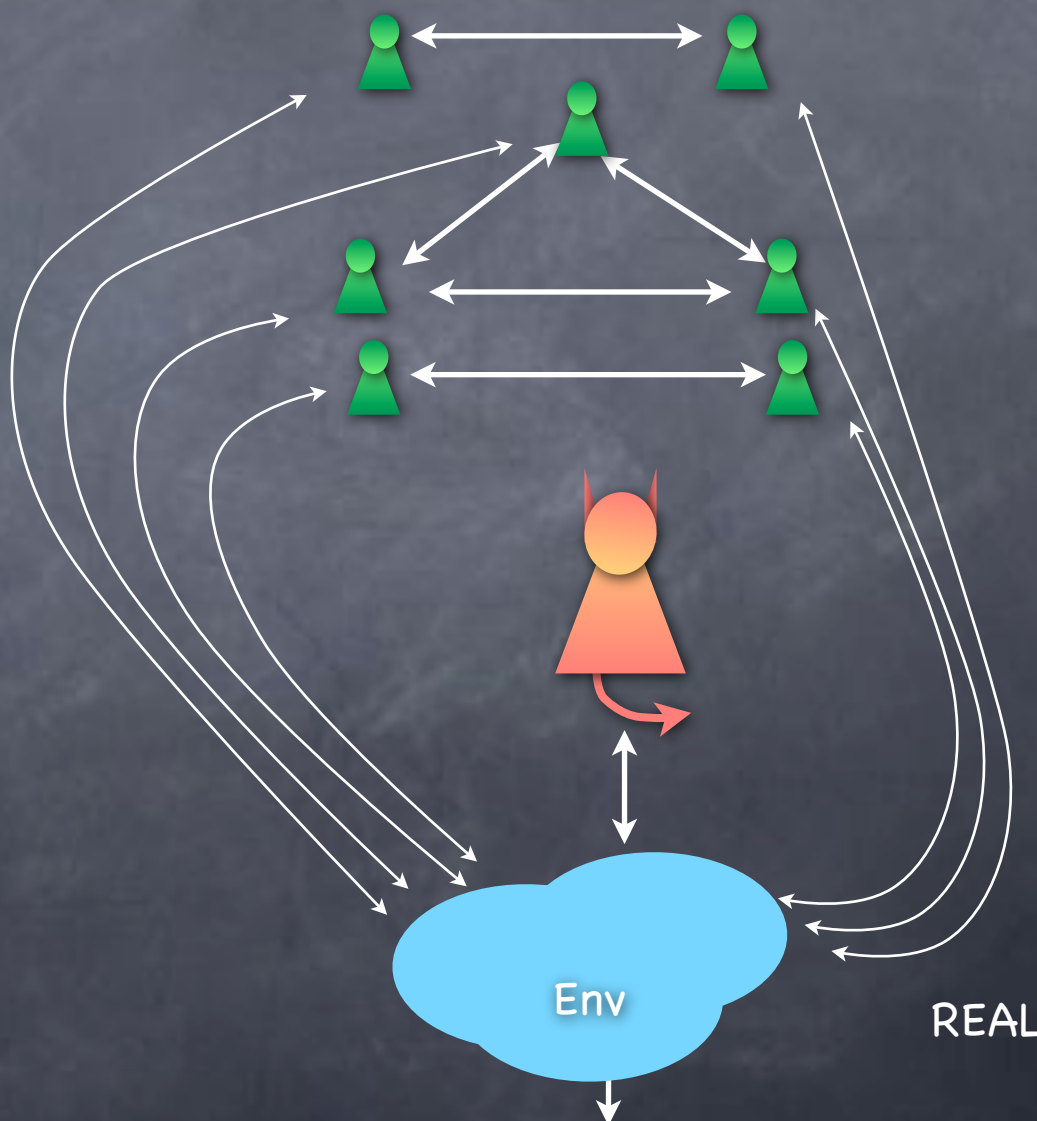




Security of Composed Systems

Extend to allow a "composed system" with multiple functionalities

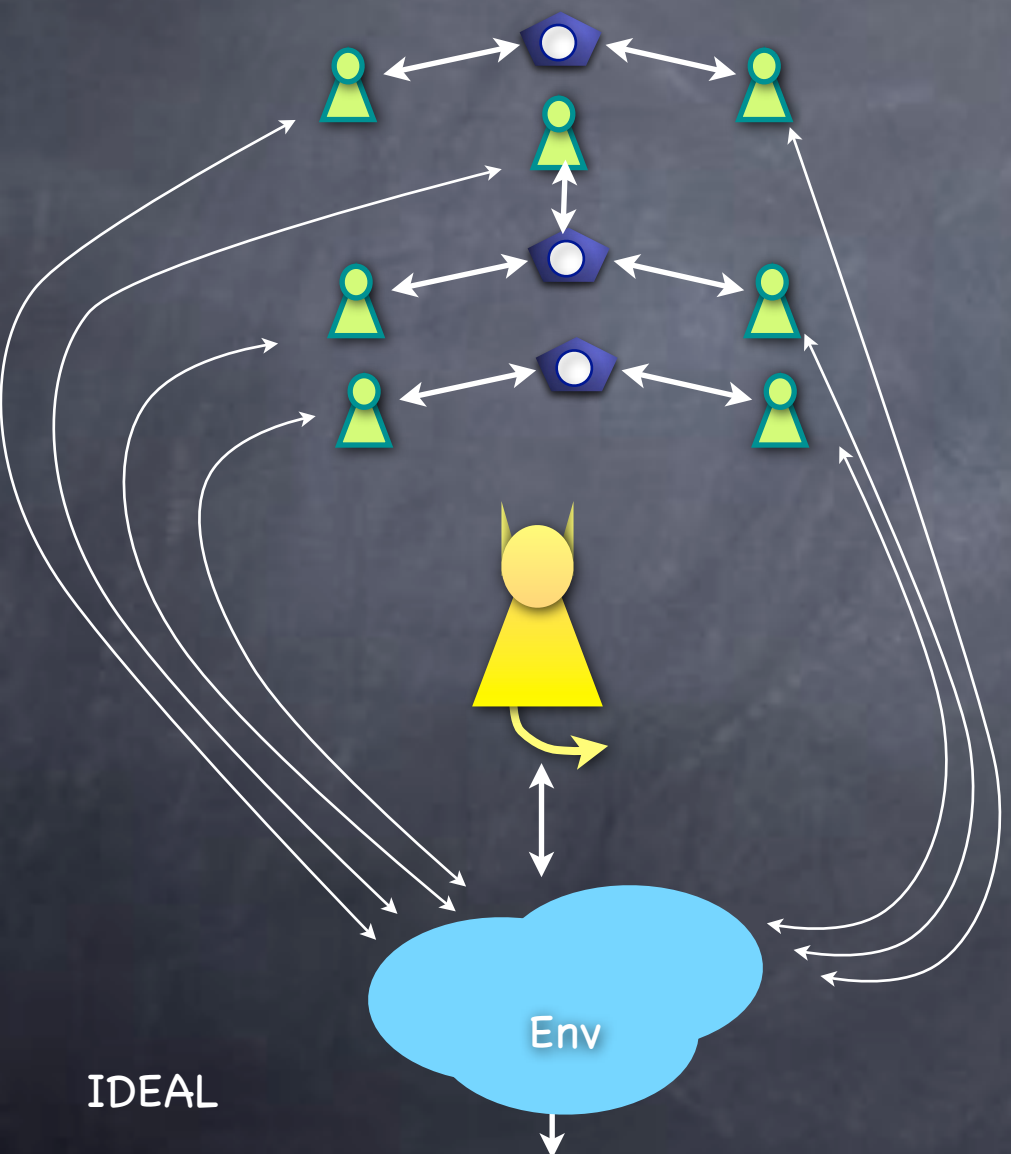


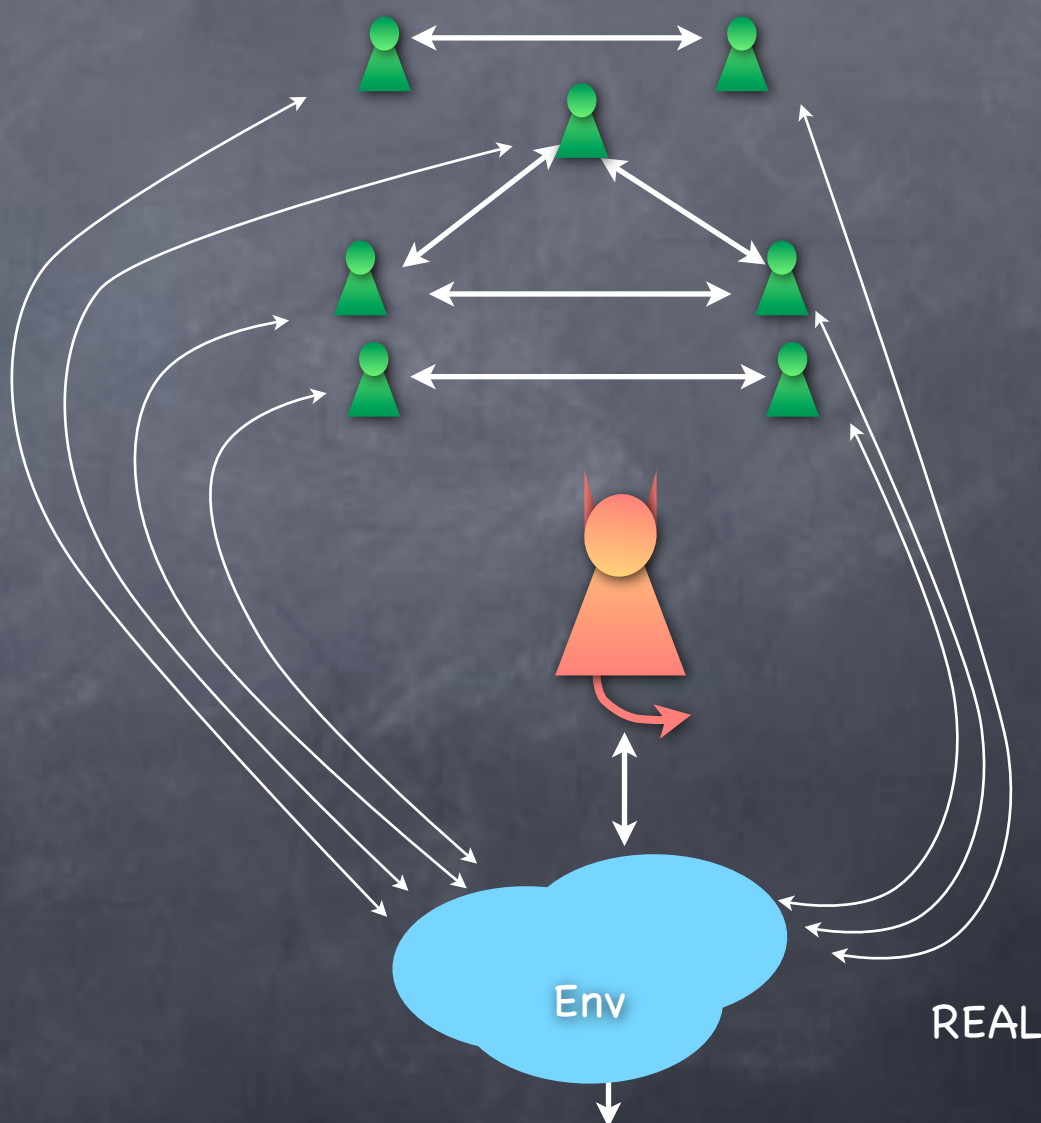


Security of Composed Systems

Extend to allow a "composed system" with multiple functionalities

REAL (with protocols) is as secure as IDEAL (with functionalities) if:

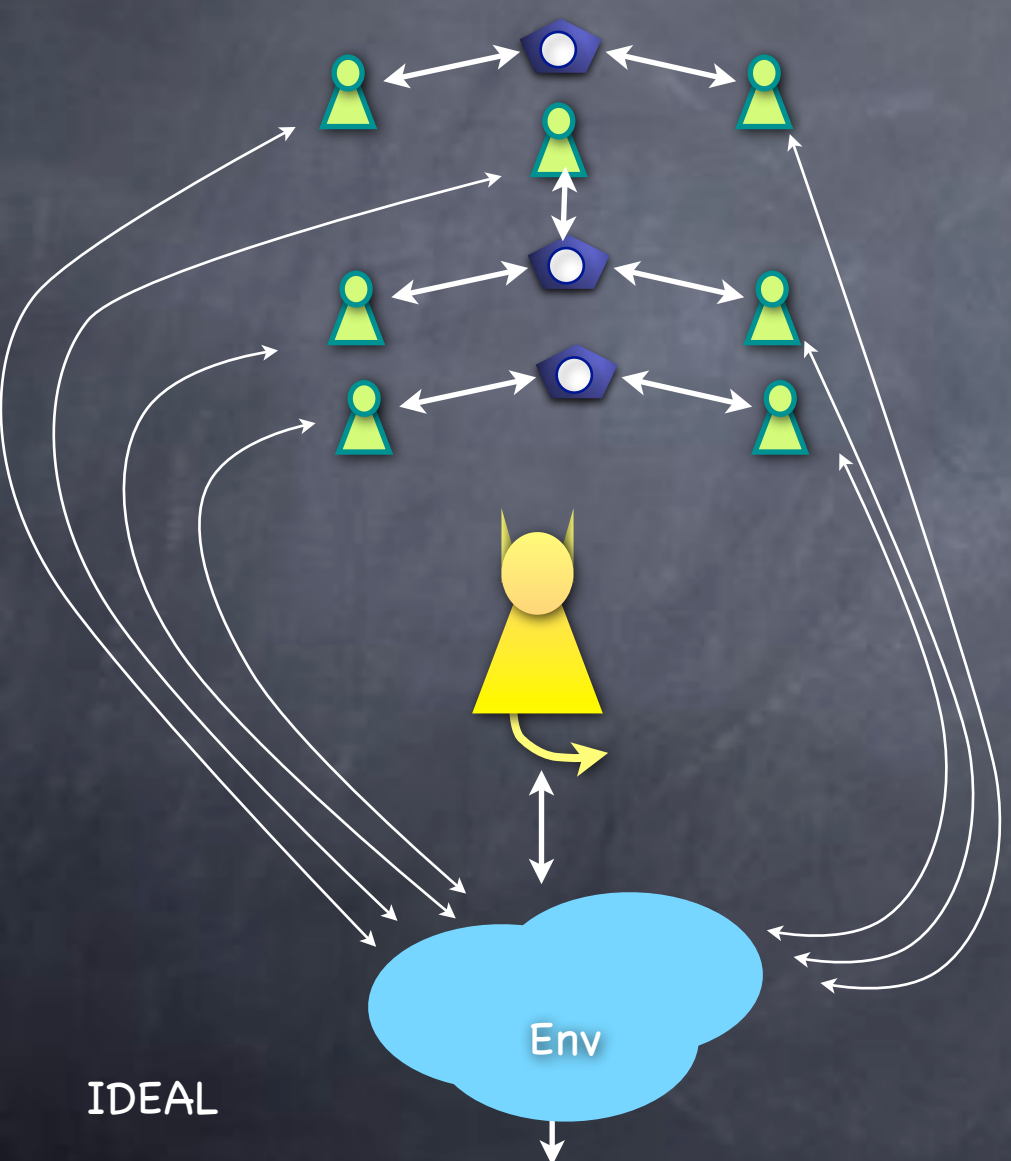


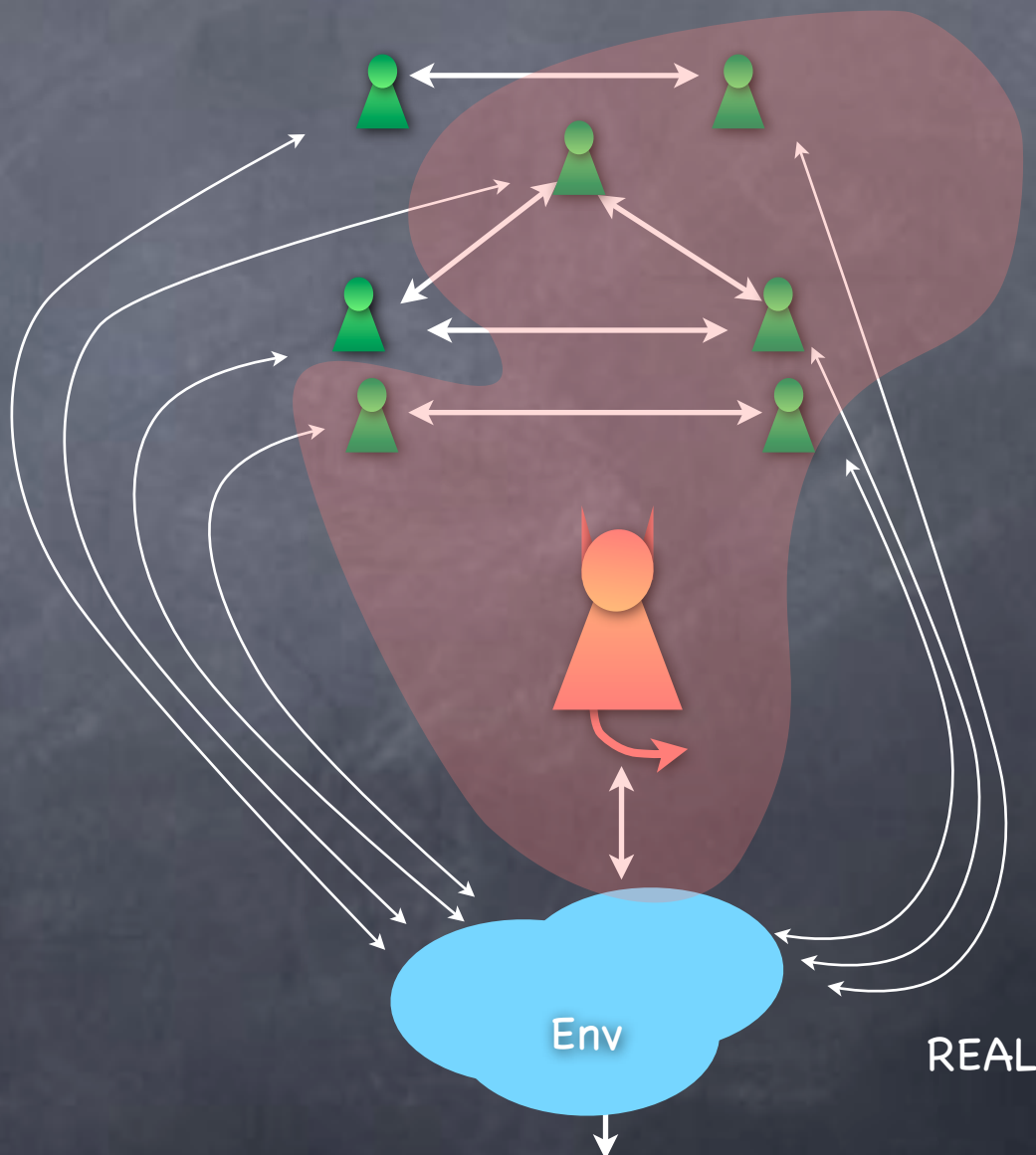


Security of Composed Systems

Extend to allow a "composed system" with multiple functionalities

REAL (with protocols) is as secure as IDEAL (with functionalities) if:

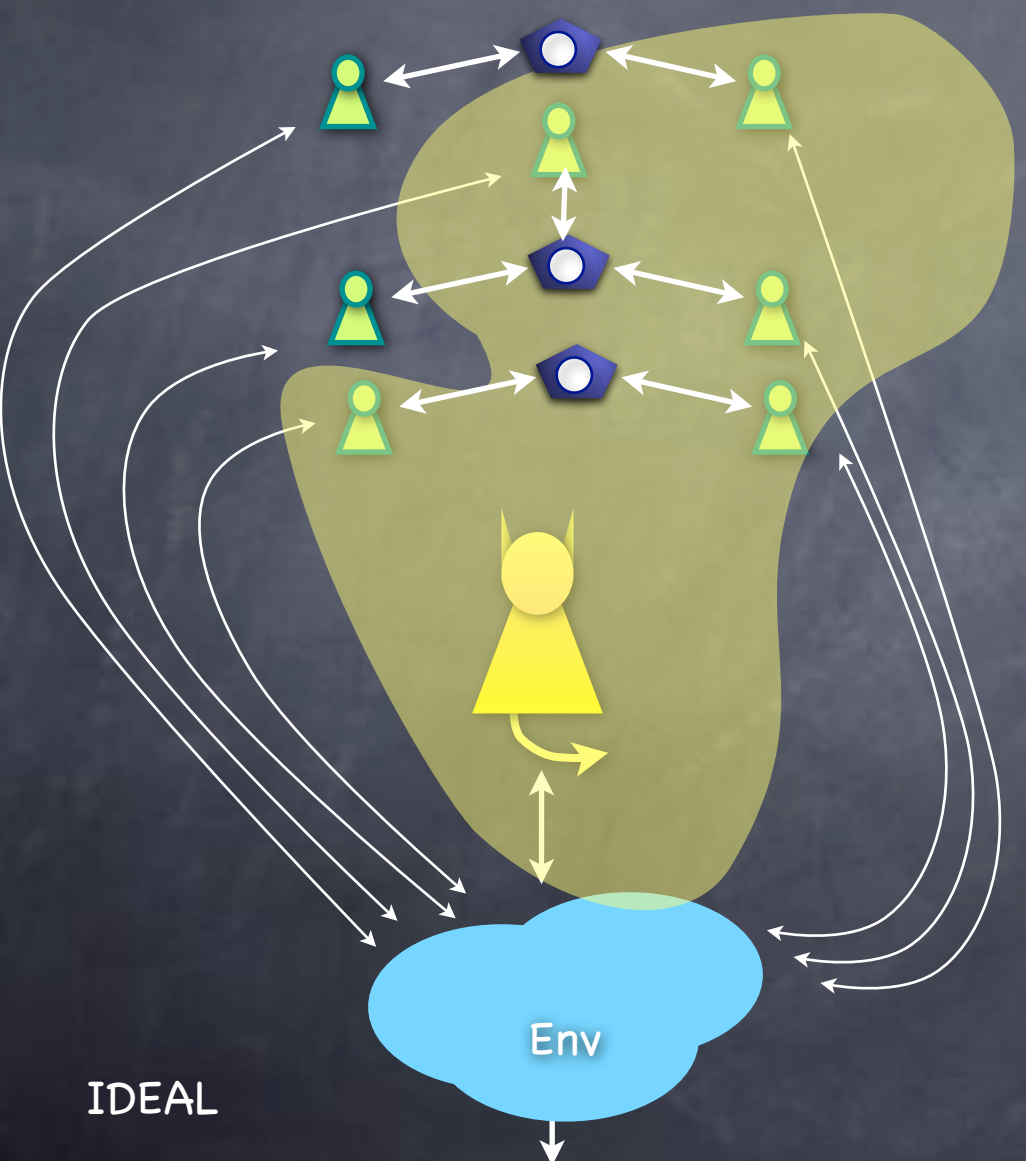


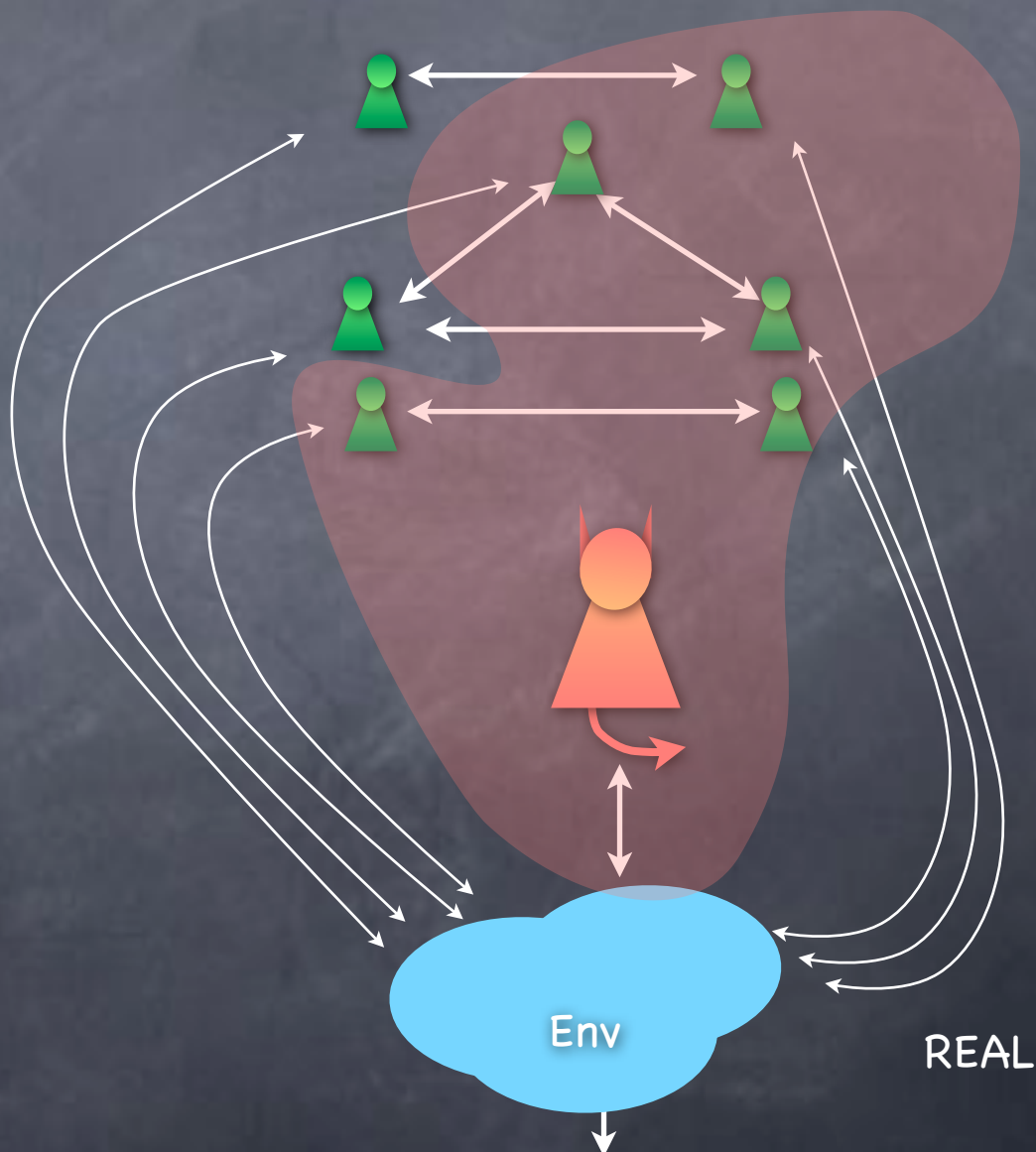


Security of Composed Systems

Extend to allow a "composed system" with multiple functionalities

REAL (with protocols) is as secure as IDEAL (with functionalities) if:

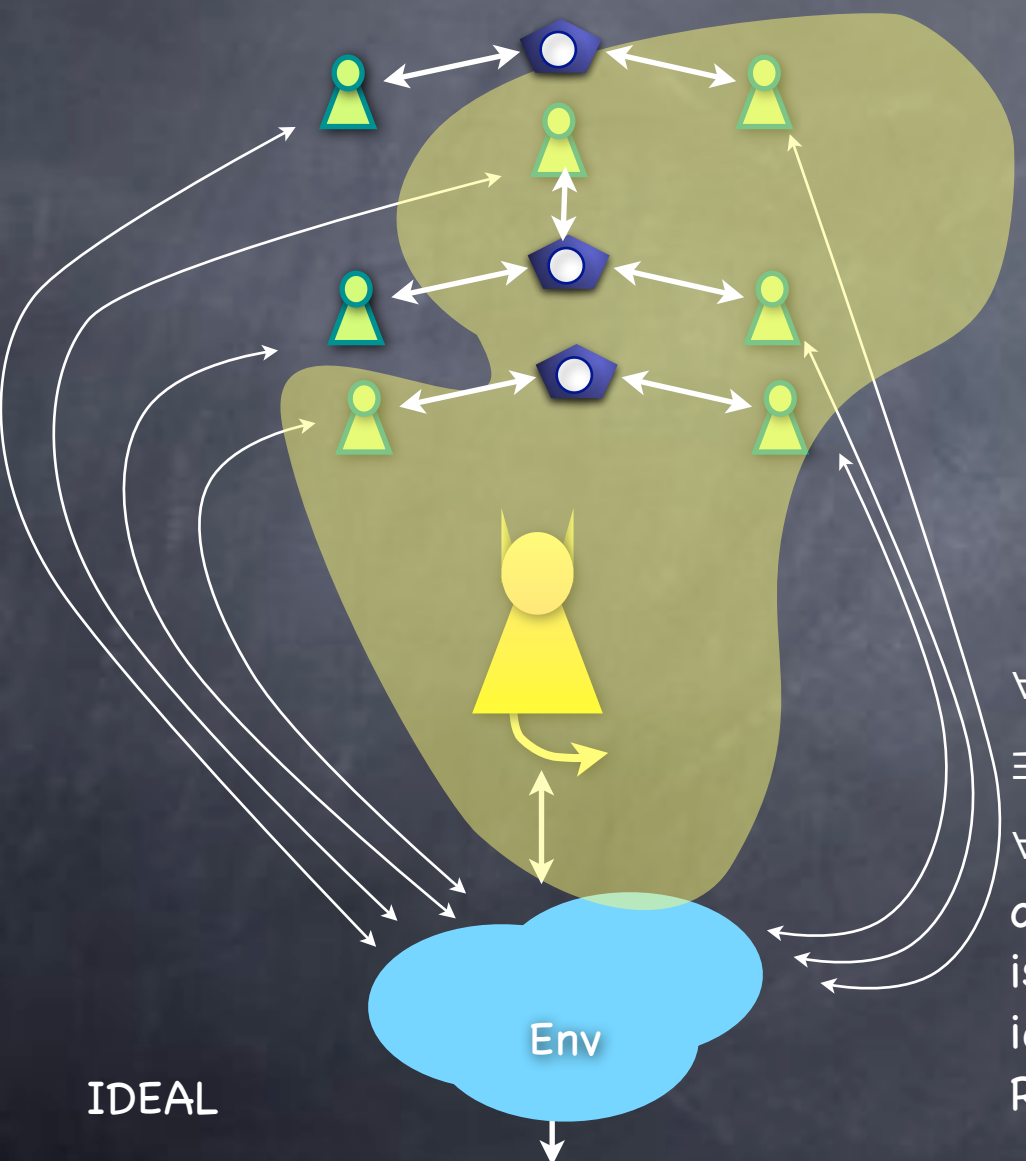




Security of Composed Systems

Extend to allow a "composed system" with multiple functionalities

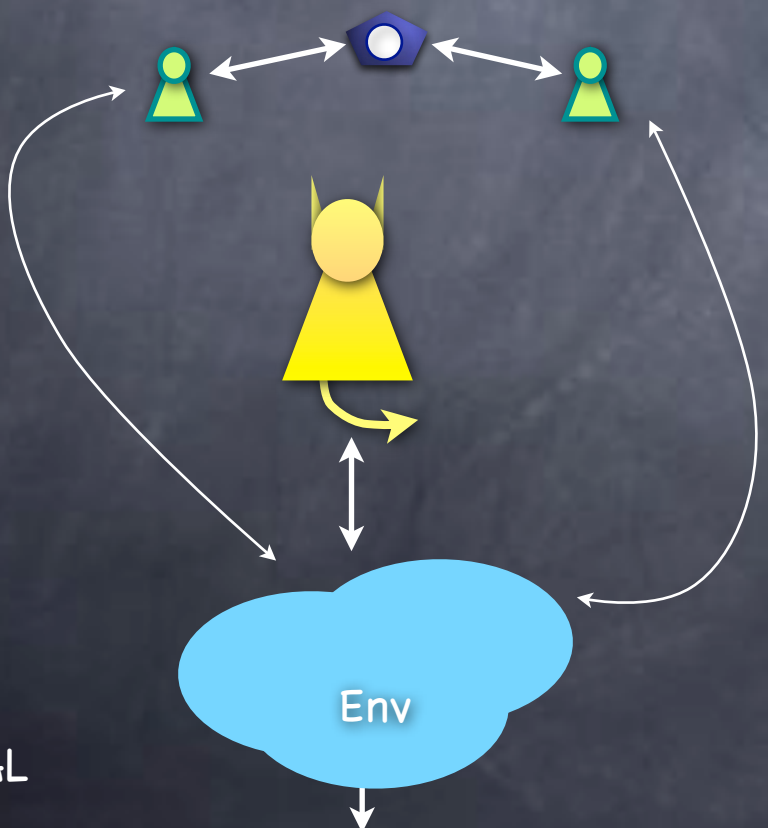
REAL (with protocols) is as secure as IDEAL (with functionalities) if:

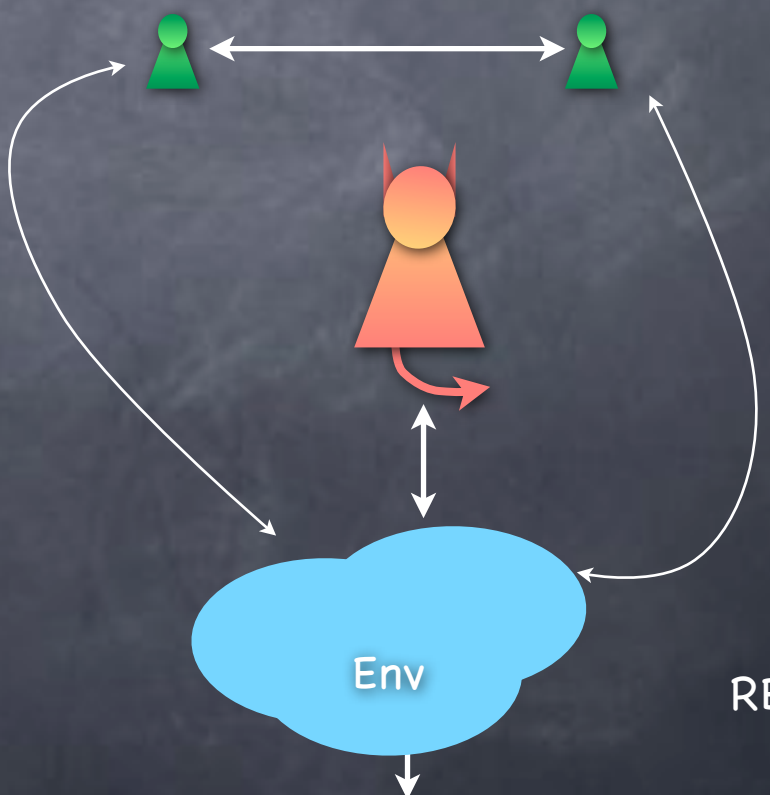


output of ☁ is distributed identically in REAL and IDEAL

Universal Composition - 1

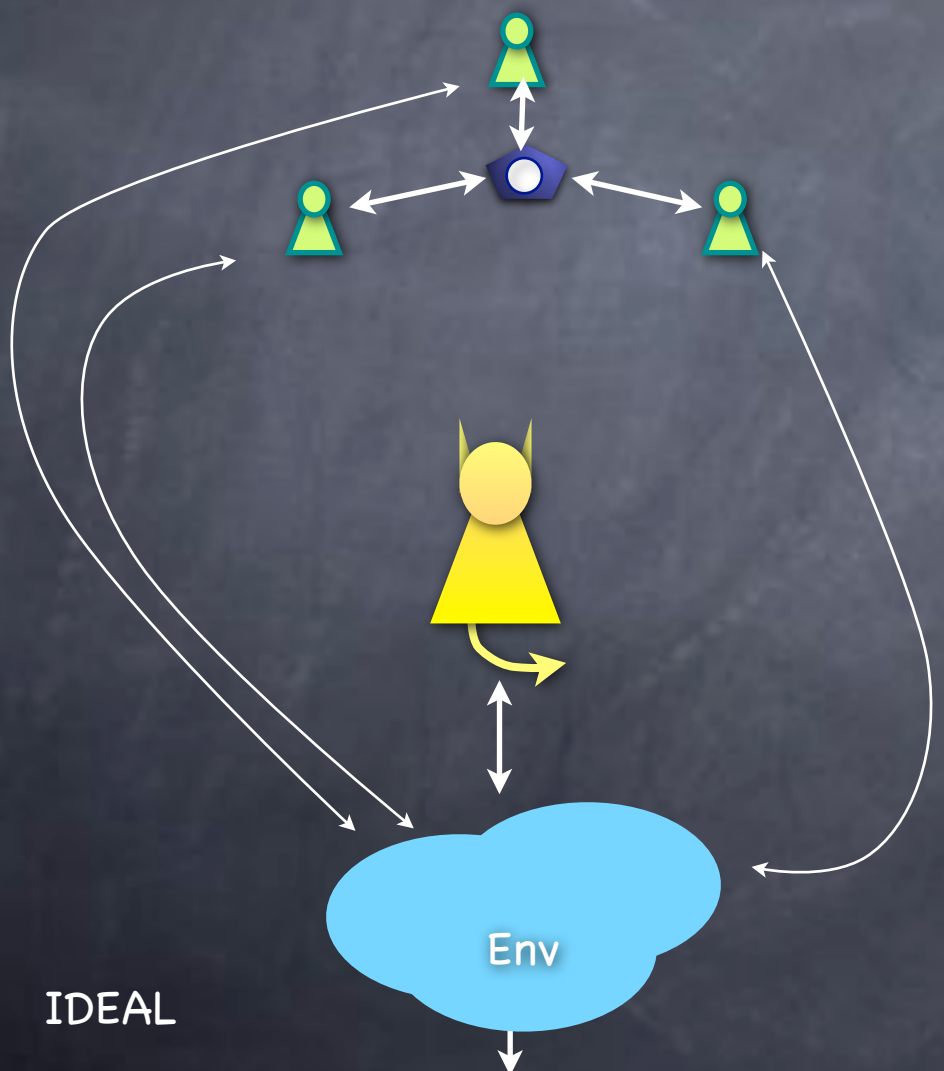
If each protocol secure (i.e., 
is as secure as  etc.)

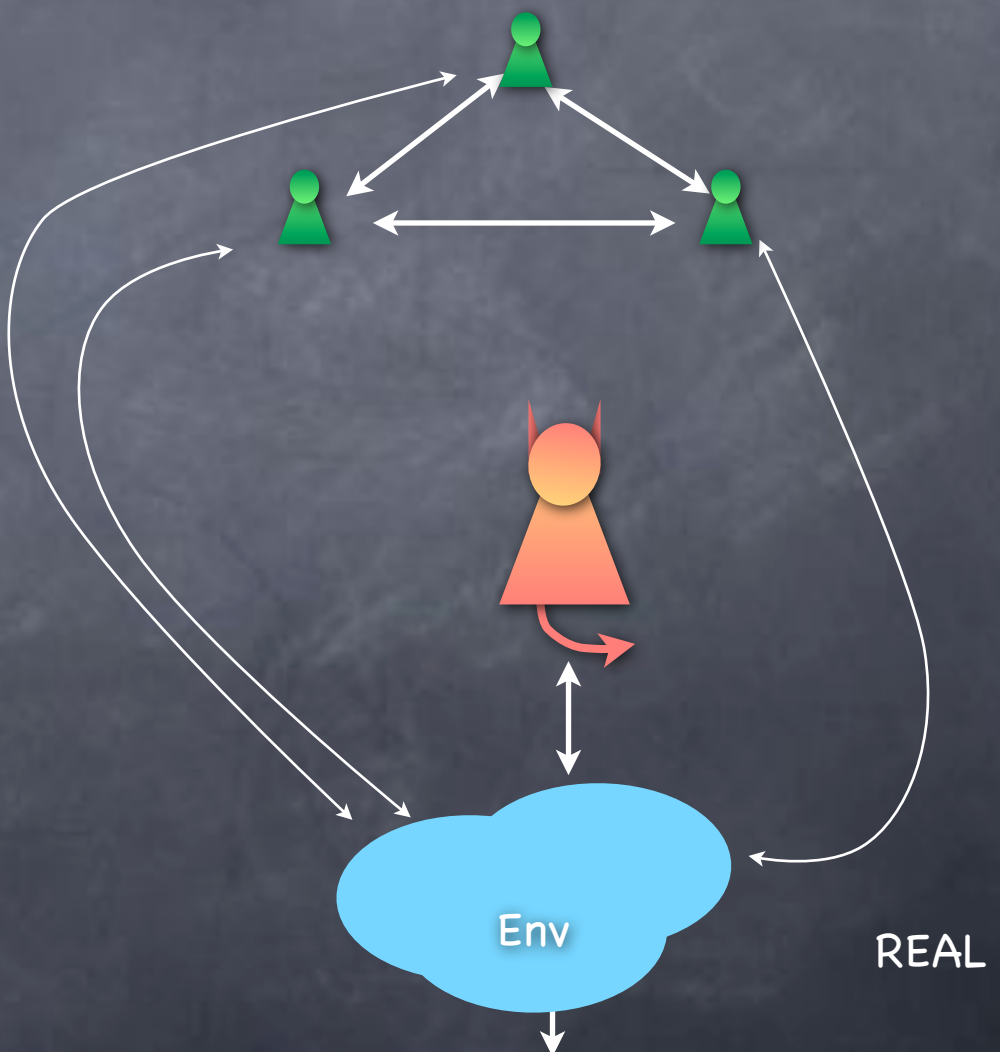




Universal Composition - 1

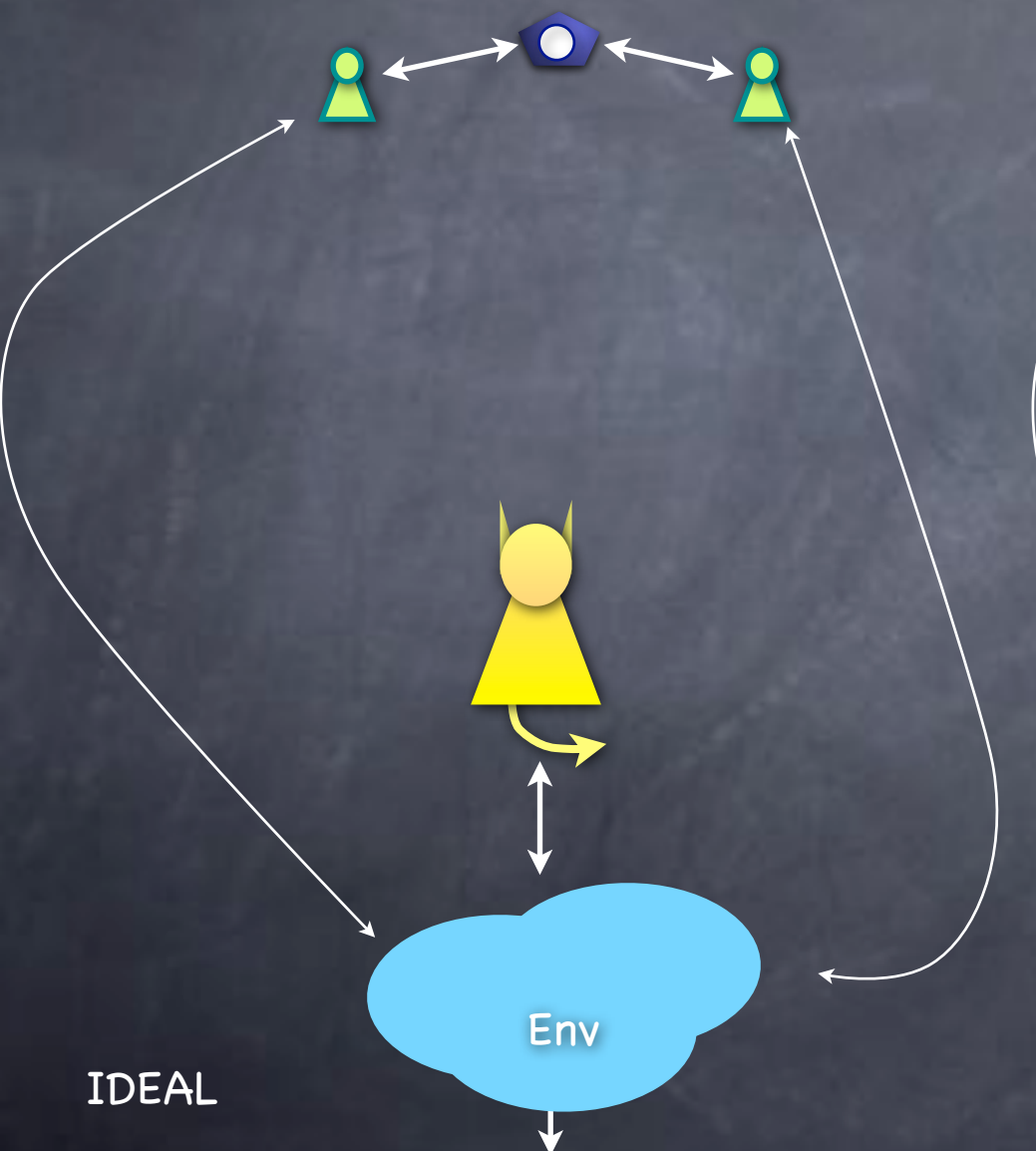
If each protocol secure (i.e., 
is as secure as  etc.)

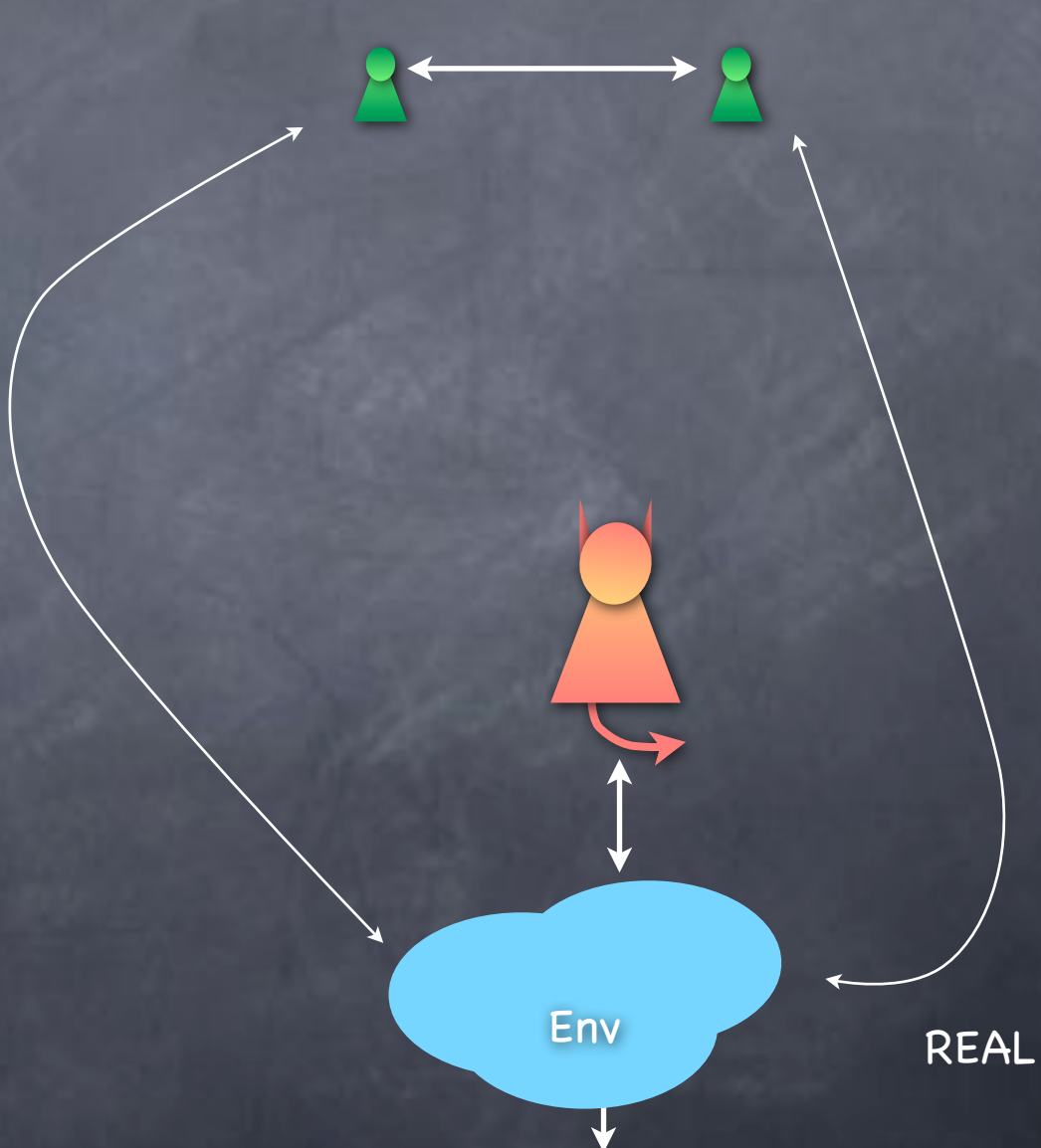




Universal Composition - 1

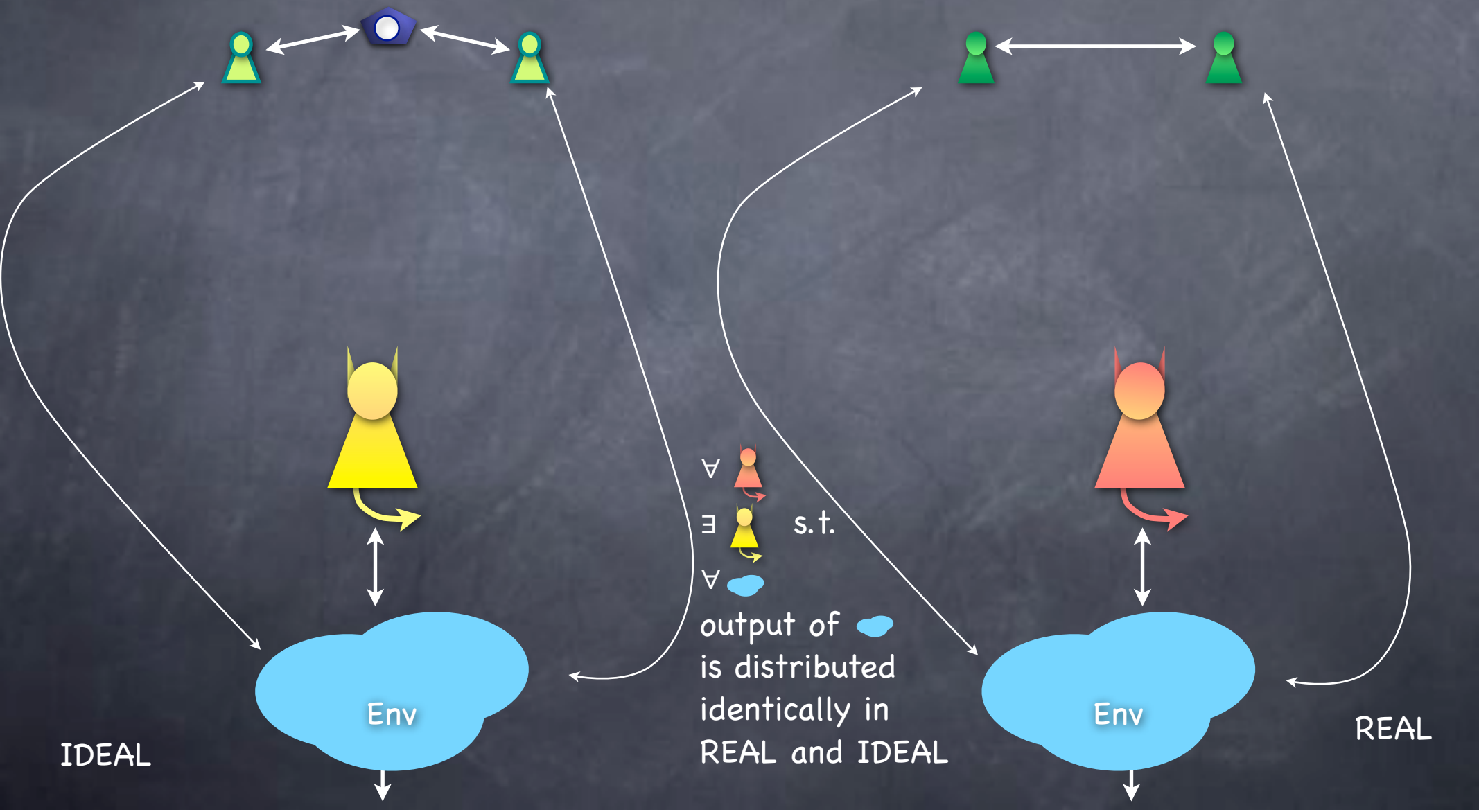
If each protocol secure (i.e., 
is as secure as  etc.)





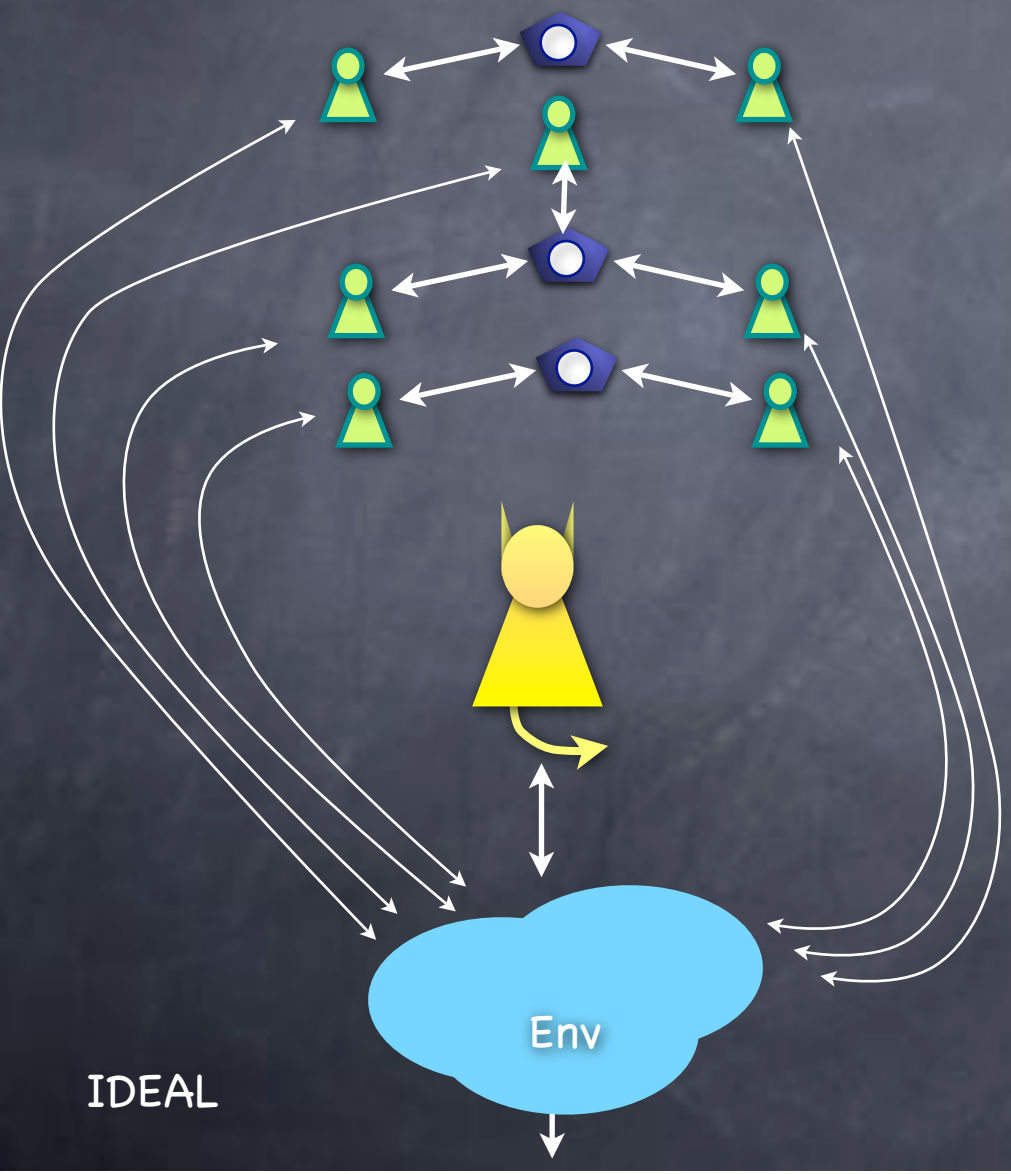
Universal Composition - 1

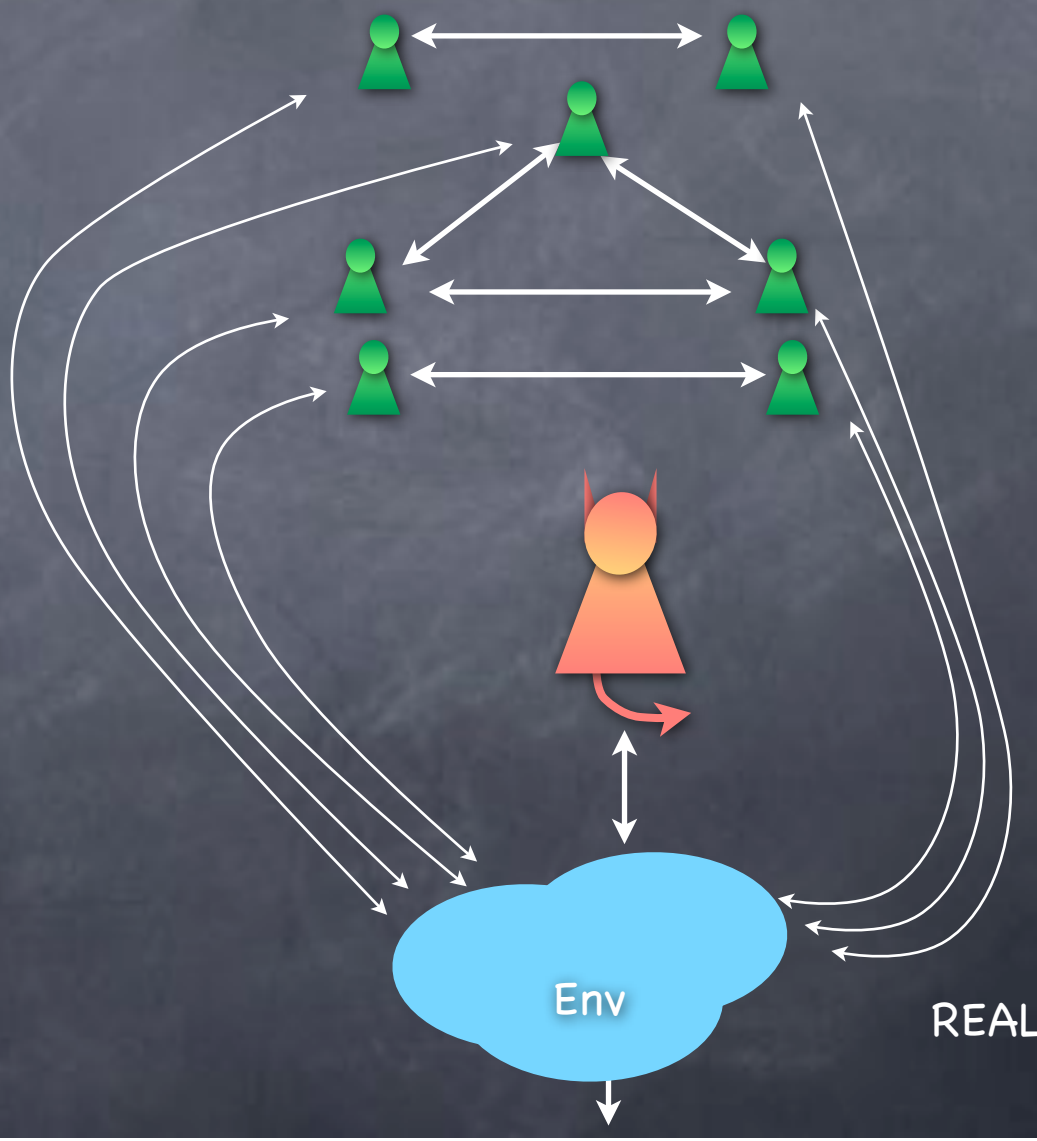
If each protocol secure (i.e., ↔)
is as secure as etc.)



Universal Composition - 1

then concurrent sessions are secure too

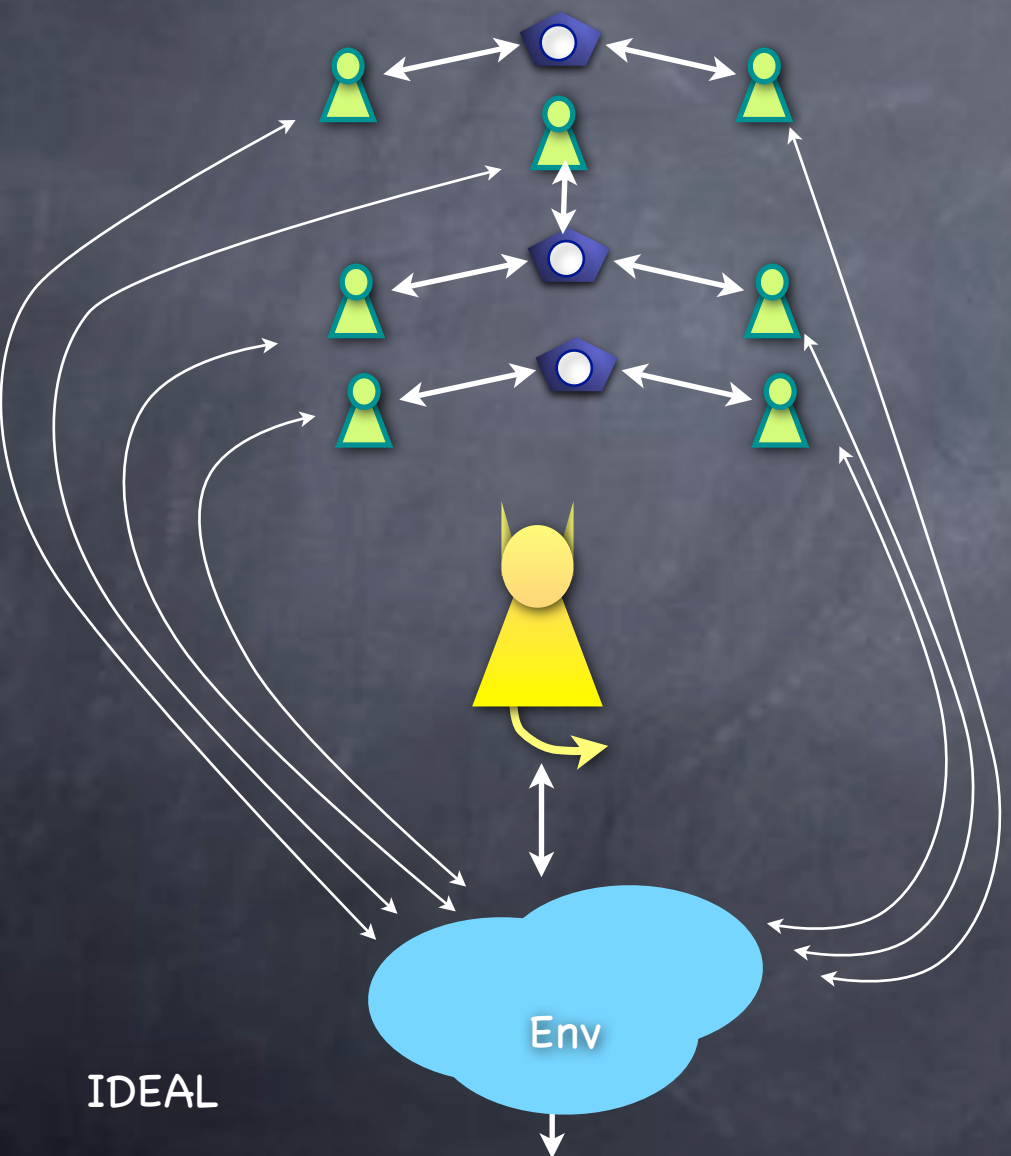


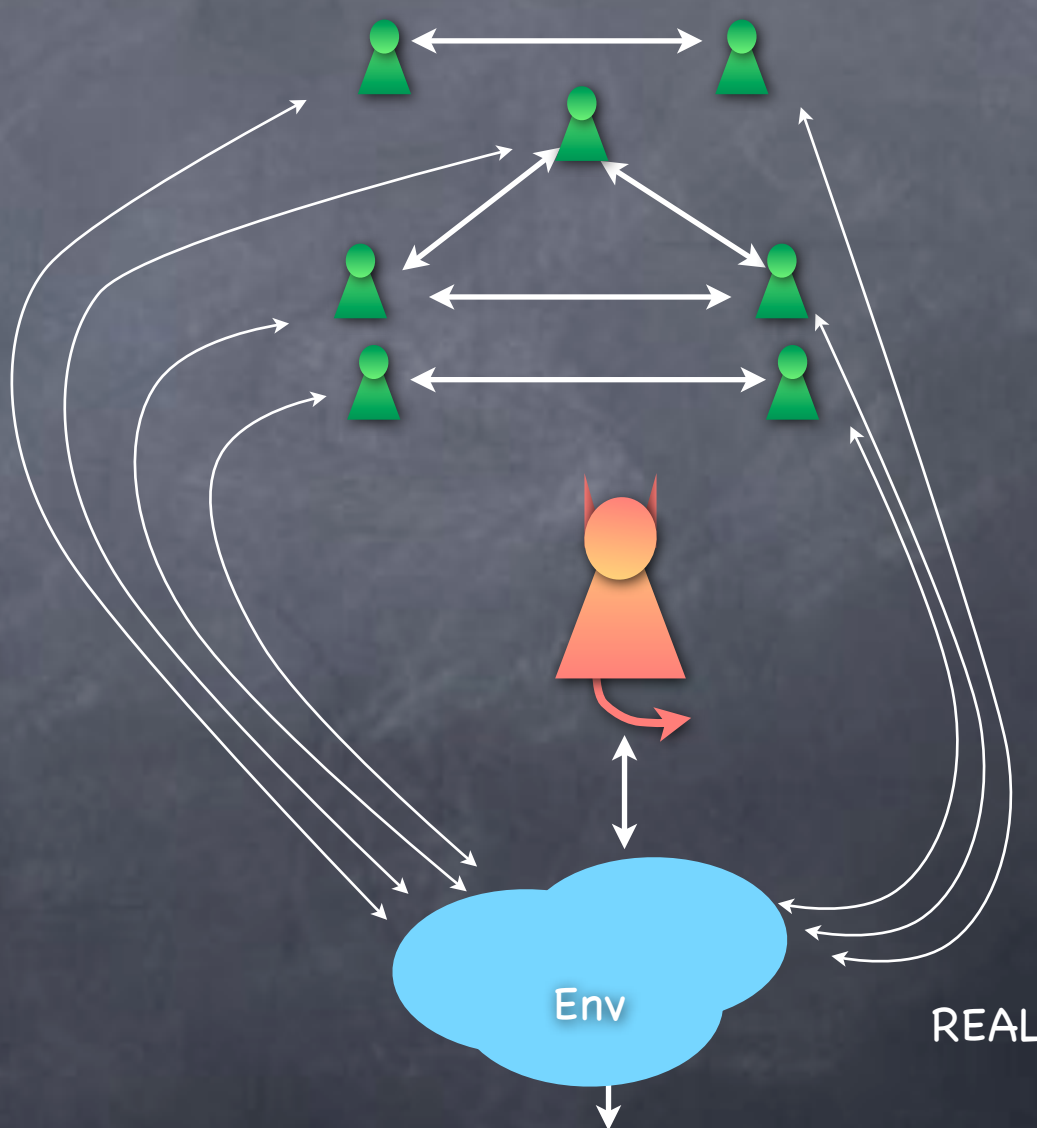


Universal Composition - 1

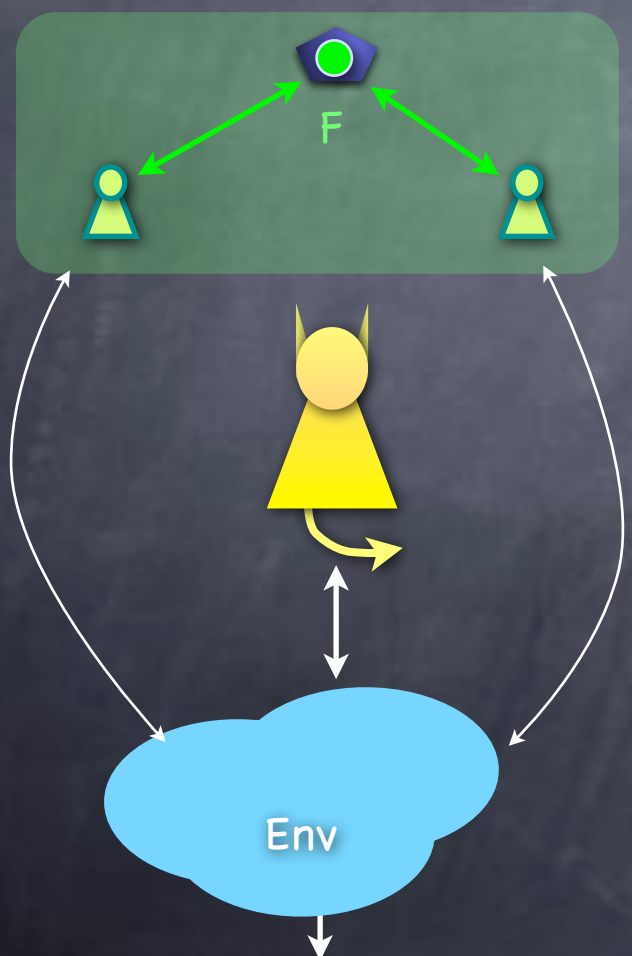
then concurrent sessions are secure too

i.e.,  is as secure as  etc.

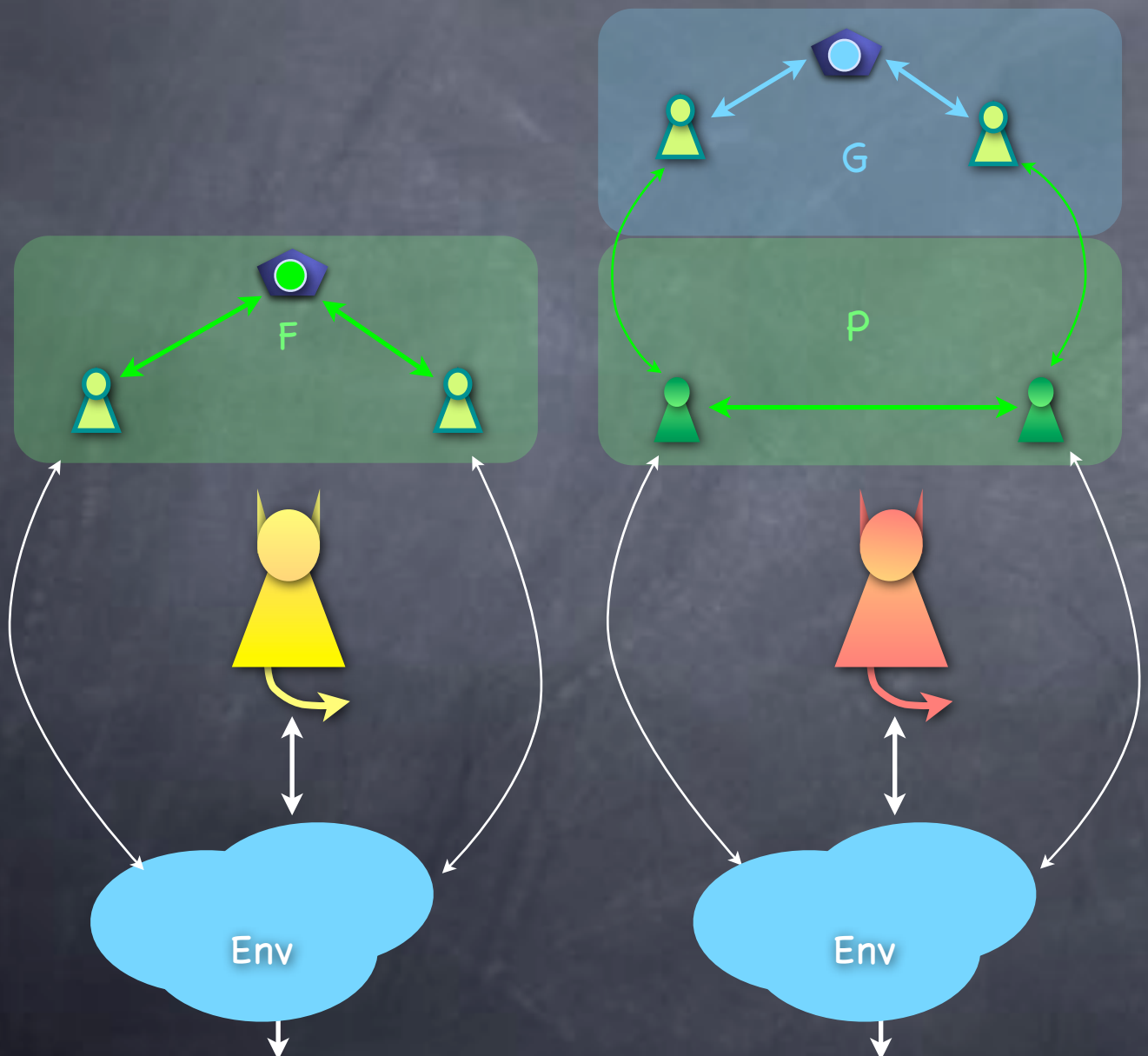




Universal Composition - 2

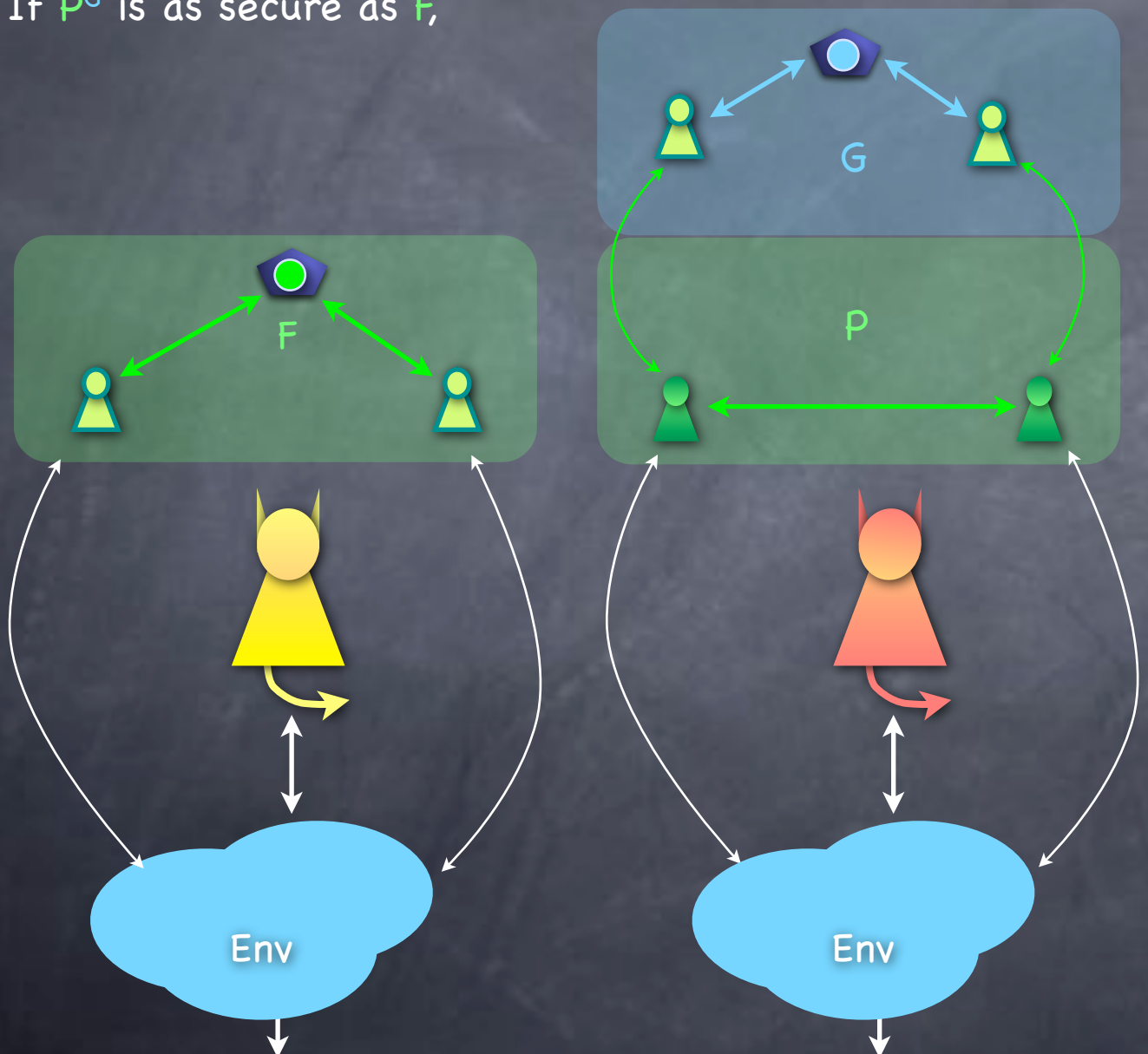


Universal Composition - 2



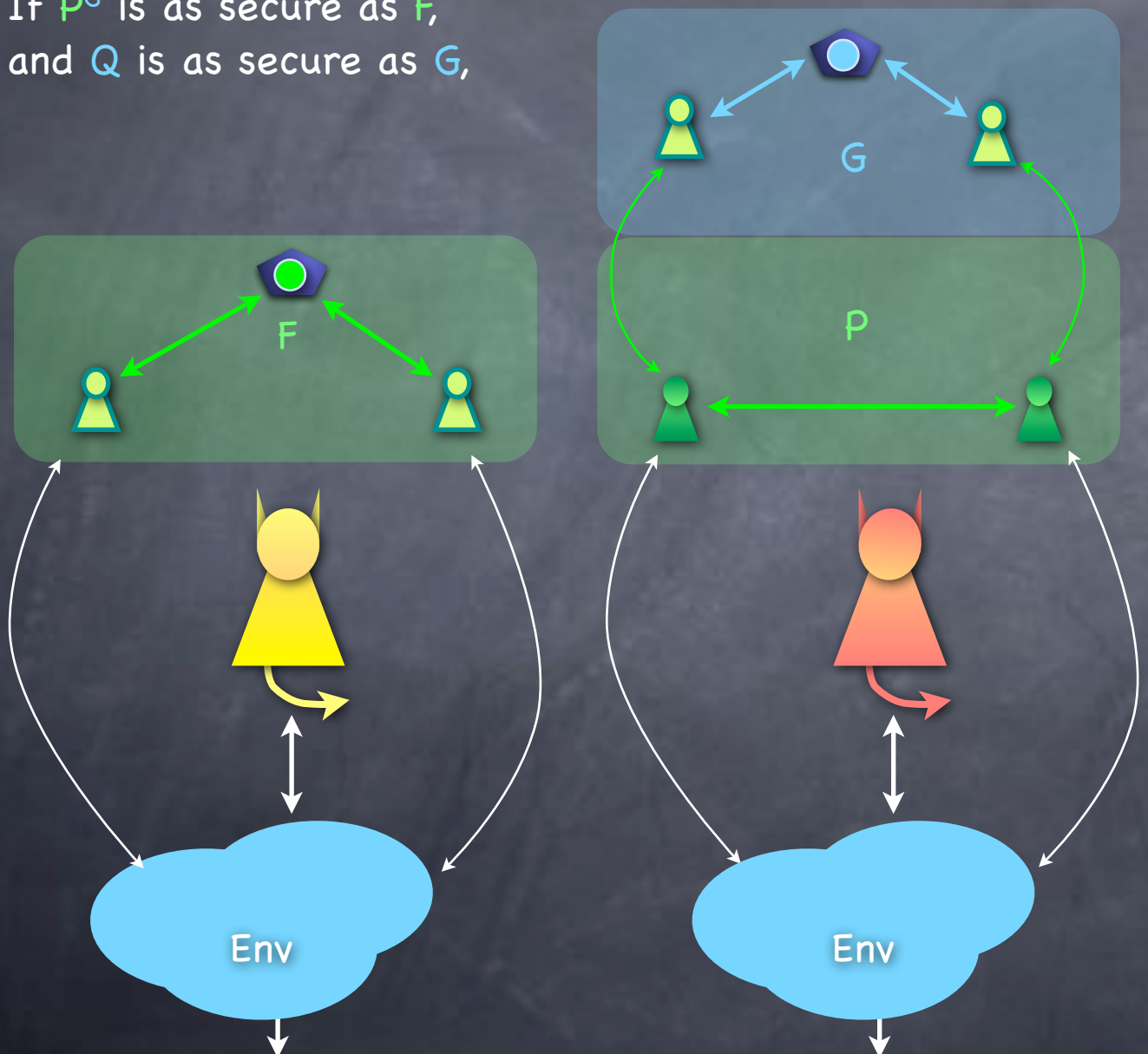
Universal Composition - 2

If P^G is as secure as F,



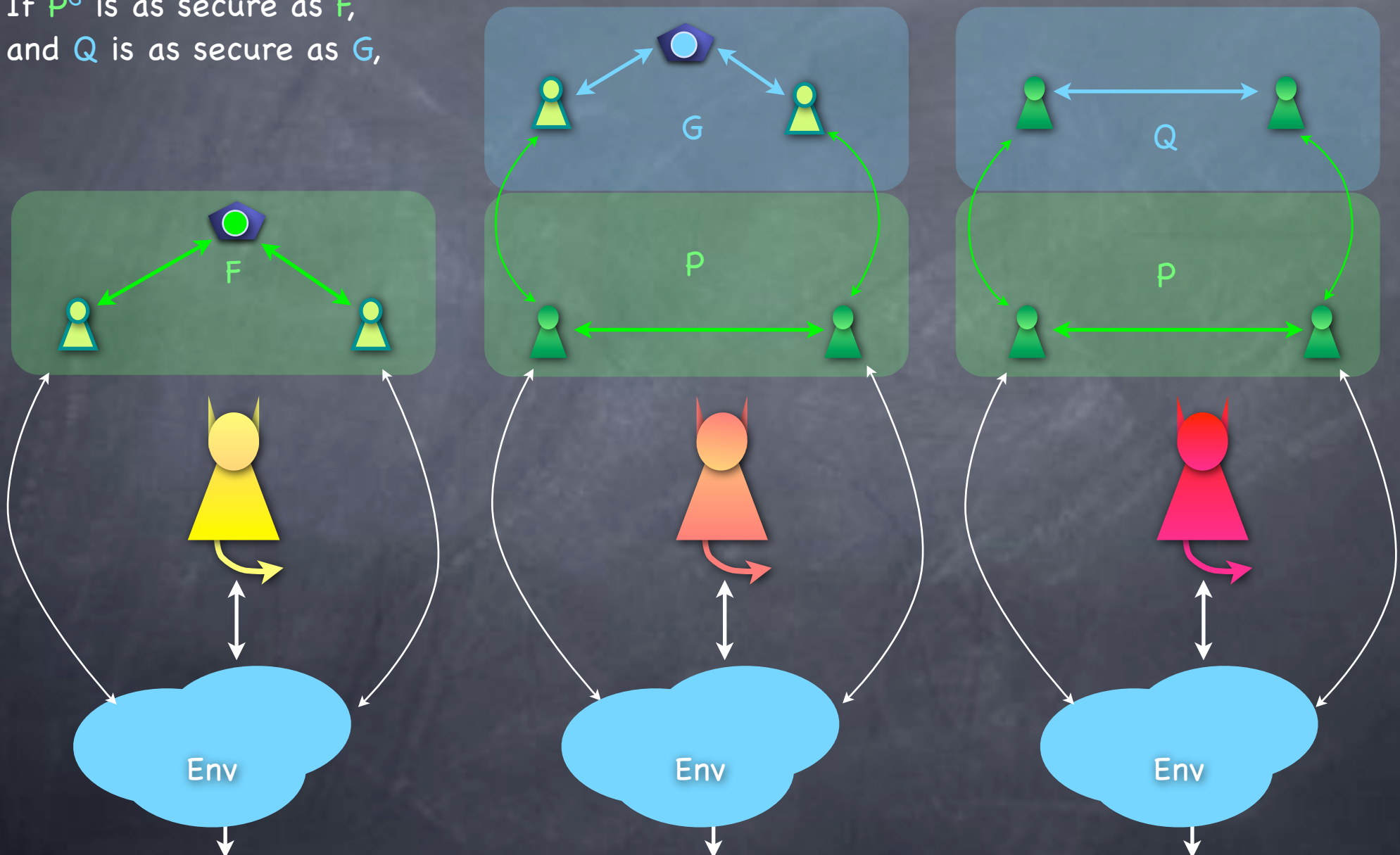
Universal Composition - 2

If P^G is as secure as F,
and Q is as secure as G,



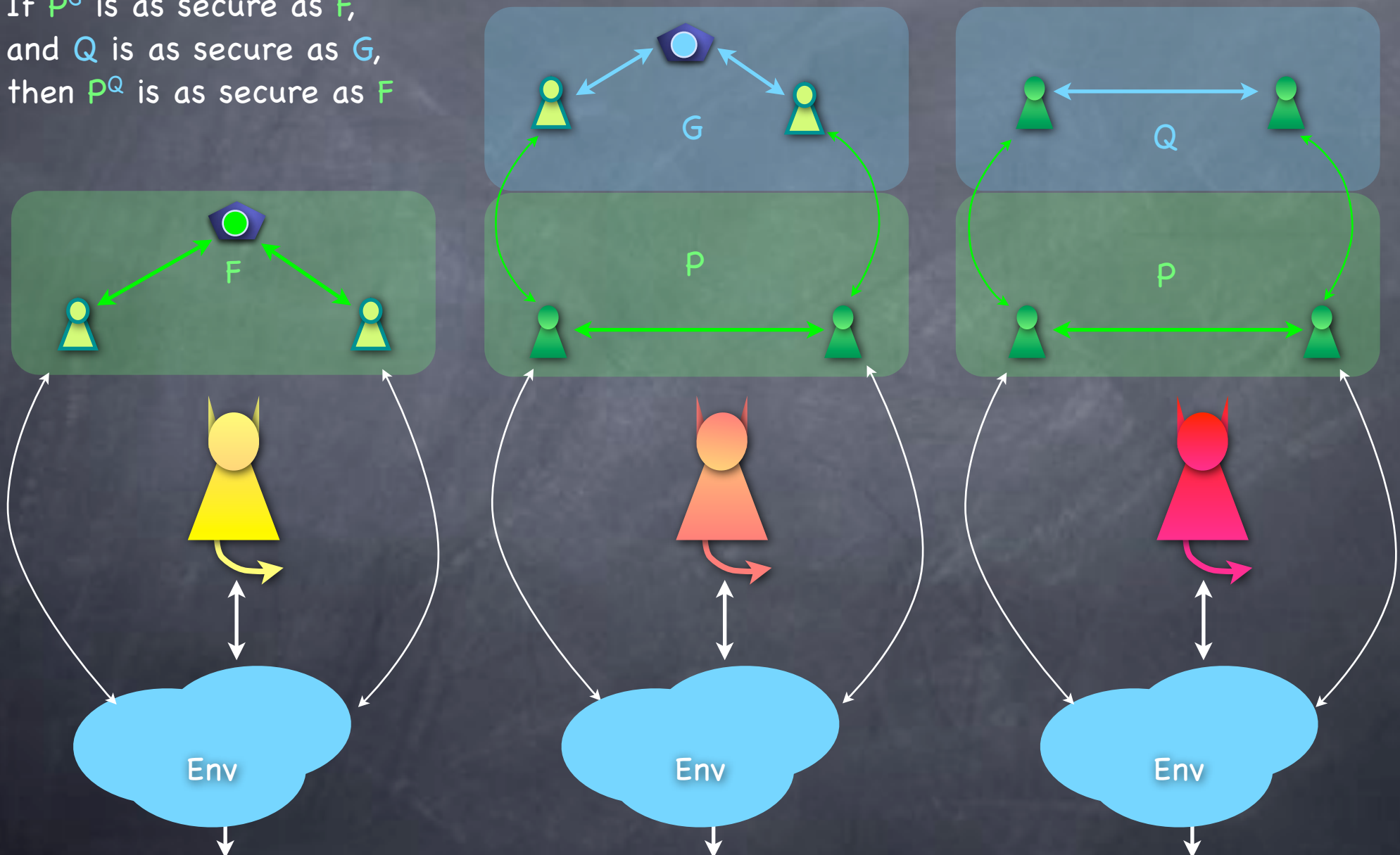
Universal Composition - 2

If P^G is as secure as F,
and Q is as secure as G,



Universal Composition - 2

If P^G is as secure as F,
and Q is as secure as G,
then P^Q is as secure as F



Universal Composition

Universal Composition

- More generally:

Universal Composition

- More generally:
  - Start from world A (think "IDEAL")

Universal Composition

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

Universal Composition

- More generally:
  - Start from world A (think "IDEAL")
    - Repeat (for any poly number of times):
      - For some X,Y such that Y is as secure as X, substitute an X-session by a Y-session

Universal Composition

- More generally:
  - Start from world A (think "IDEAL")
    - Repeat (for any poly number of times):
      - For some X,Y such that Y is as secure as X, substitute an X-session by a Y-session
    - Say we obtain world B (think "REAL")

Universal Composition

- More generally:
  - Start from world A (think "IDEAL")
    - Repeat (for any poly number of times):
      - For some X,Y such that Y is as secure as X, substitute an X-session by a Y-session
    - Say we obtain world B (think "REAL")
    - UC Theorem: Then world B is as secure as world A

Universal Composition

- More generally:
  - Start from world A (think "IDEAL")
    - Repeat (for any poly number of times):
      - For some X,Y such that Y is as secure as X, substitute an X-session by a Y-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

UC and SIM-security

UC and SIM-security

- Key to universal composition is allowing an arbitrary environment in the SIM-security definition

UC and SIM-security

- Key to universal composition is allowing an arbitrary environment in the SIM-security definition
  - Even when considering only one component, other components could be present in the environment

UC and SIM-security

- Key to universal composition is allowing an arbitrary environment in the SIM-security definition
  - Even when considering only one component, other components could be present in the environment
- Considering an arbitrary environment is anyway necessary for the security guarantee to be useful

UC and SIM-security

- Key to universal composition is allowing an arbitrary environment in the SIM-security definition
  - Even when considering only one component, other components could be present in the environment
- Considering an arbitrary environment is anyway necessary for the security guarantee to be useful
  - But by itself may not imply universal composition: e.g. with PPT REAL world, unbounded IDEAL (simulator or functionality)

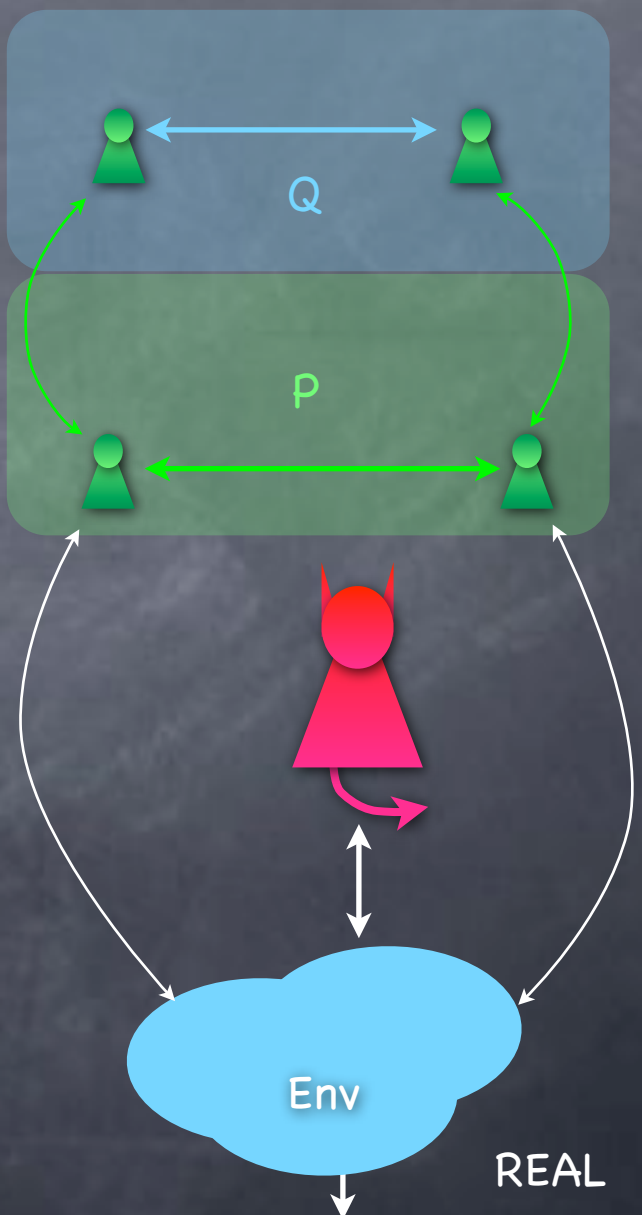
UC and SIM-security

- Key to universal composition is allowing an arbitrary environment in the SIM-security definition
  - Even when considering only one component, other components could be present in the environment
- Considering an arbitrary environment is anyway necessary for the security guarantee to be useful
  - But by itself may not imply universal composition: e.g. with PPT REAL world, unbounded IDEAL (simulator or functionality)
  - Also, UC by itself does not imply a meaningful security (nor require an environment)

UC and SIM-security

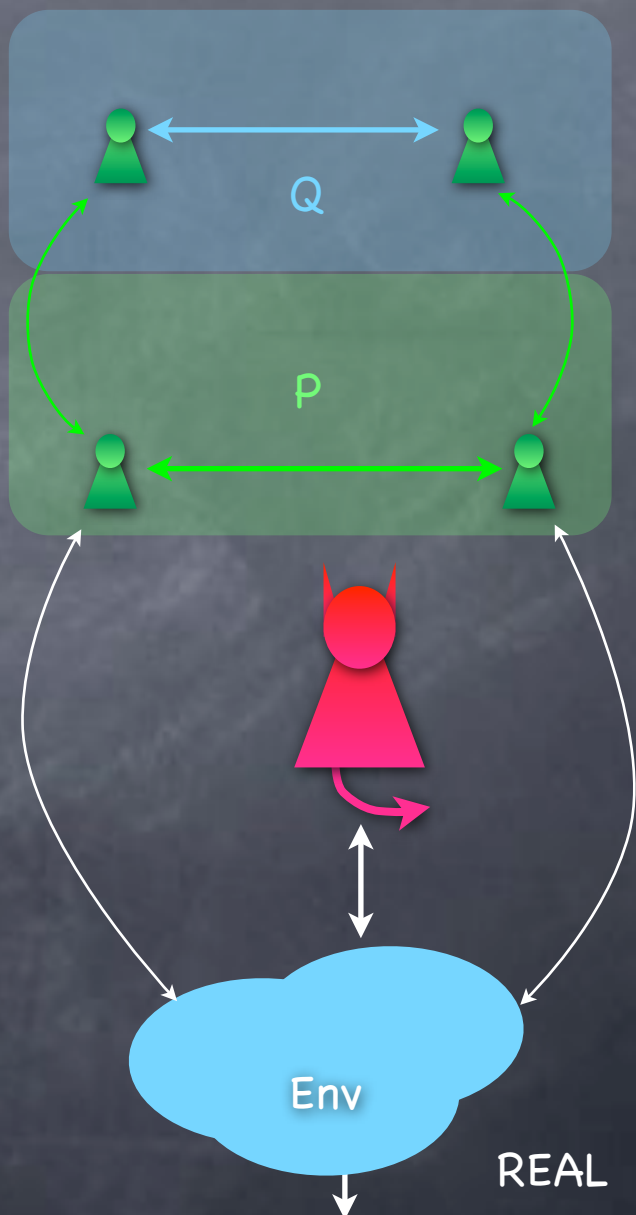
- Key to universal composition is allowing an arbitrary environment in the SIM-security definition
  - Even when considering only one component, other components could be present in the environment
- Considering an arbitrary environment is anyway necessary for the security guarantee to be useful
  - But by itself may not imply universal composition: e.g. with PPT REAL world, unbounded IDEAL (simulator or functionality)
  - Also, UC by itself does not imply a meaningful security (nor require an environment)
    - e.g. Define security of composed system as security of each individual component; Or, define everything secure.

Proving the UC theorem



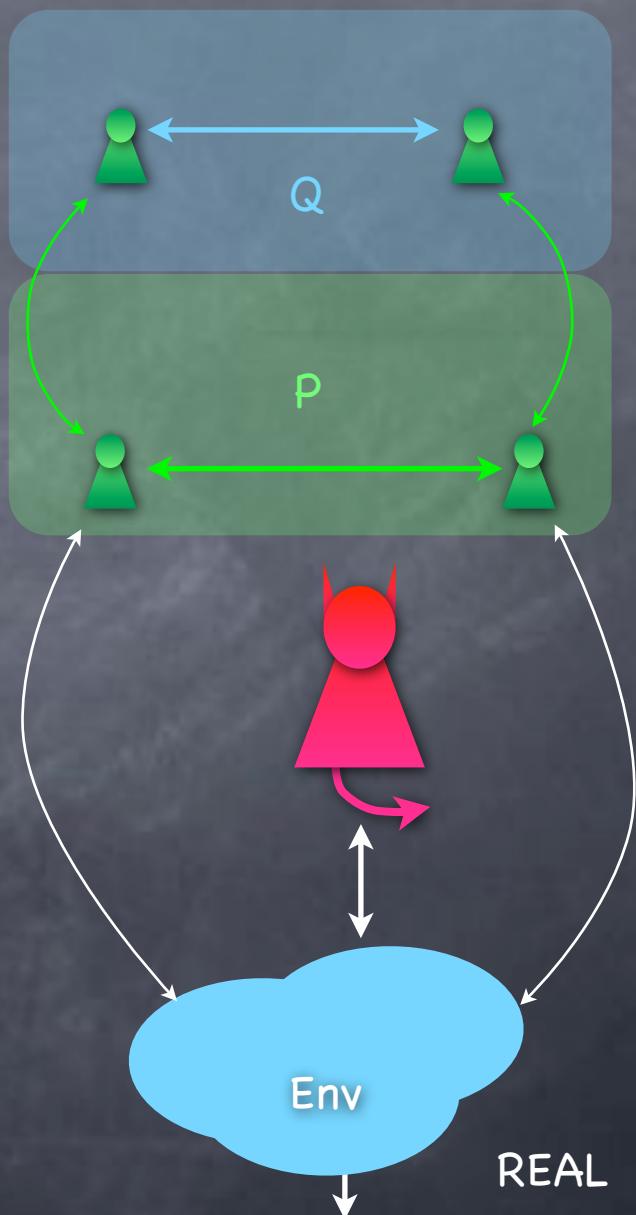
Proving the UC theorem

- Start from REAL (with an arbitrary adversary and environment)



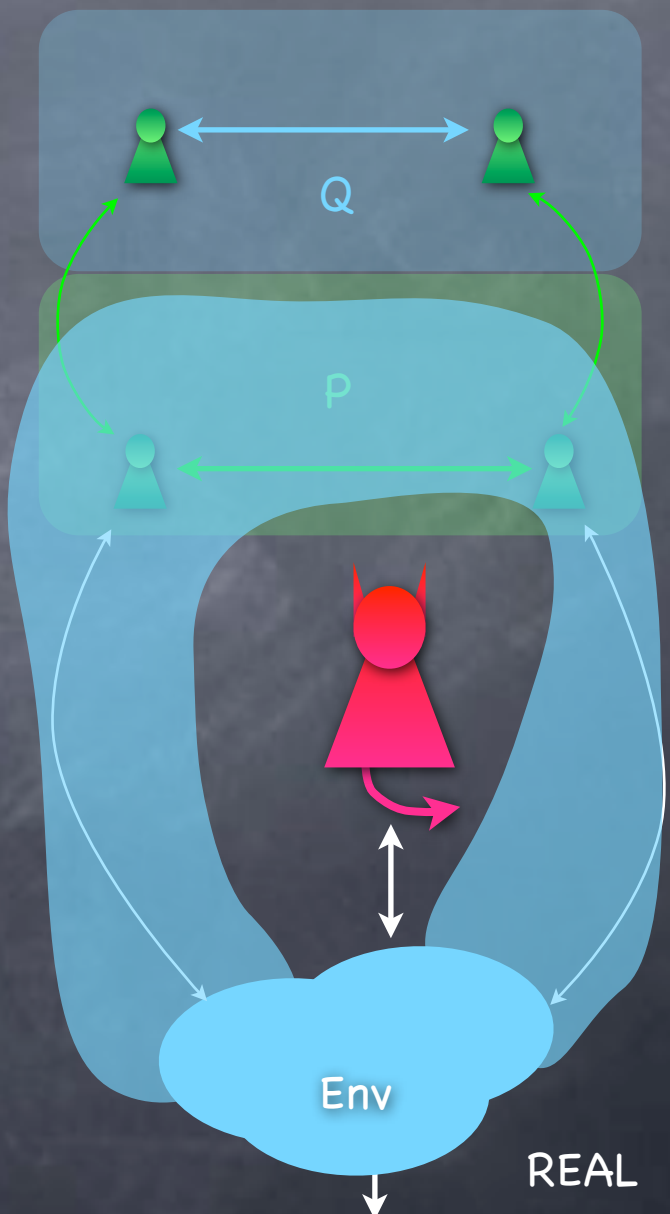
Proving the UC theorem

- Start from REAL (with an arbitrary adversary and environment)
- Consider new environment s.t. only Q (and adversary) is outside it



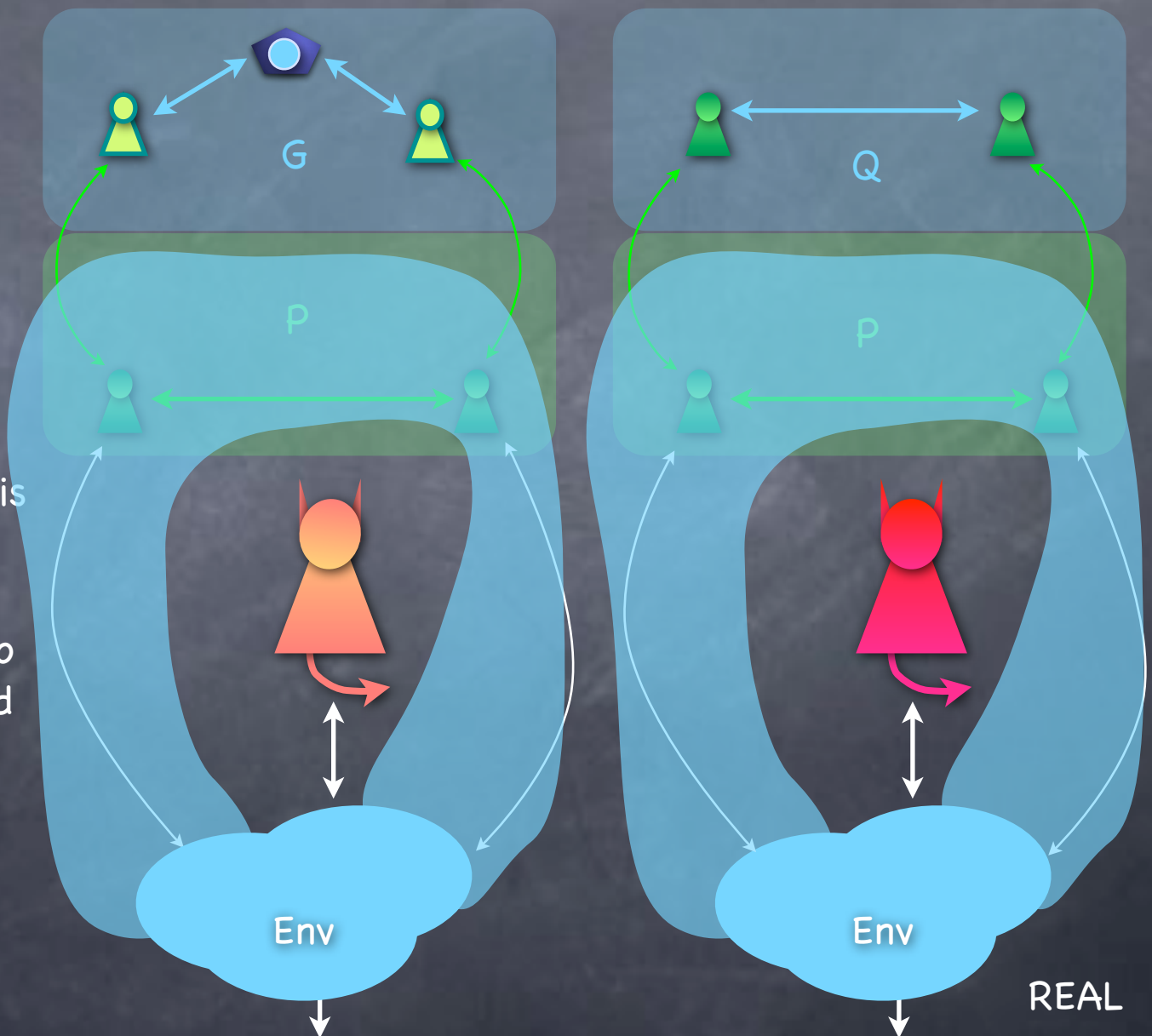
Proving the UC theorem

- Start from REAL (with an arbitrary adversary and environment)
- Consider new environment s.t. only Q (and adversary) is outside it



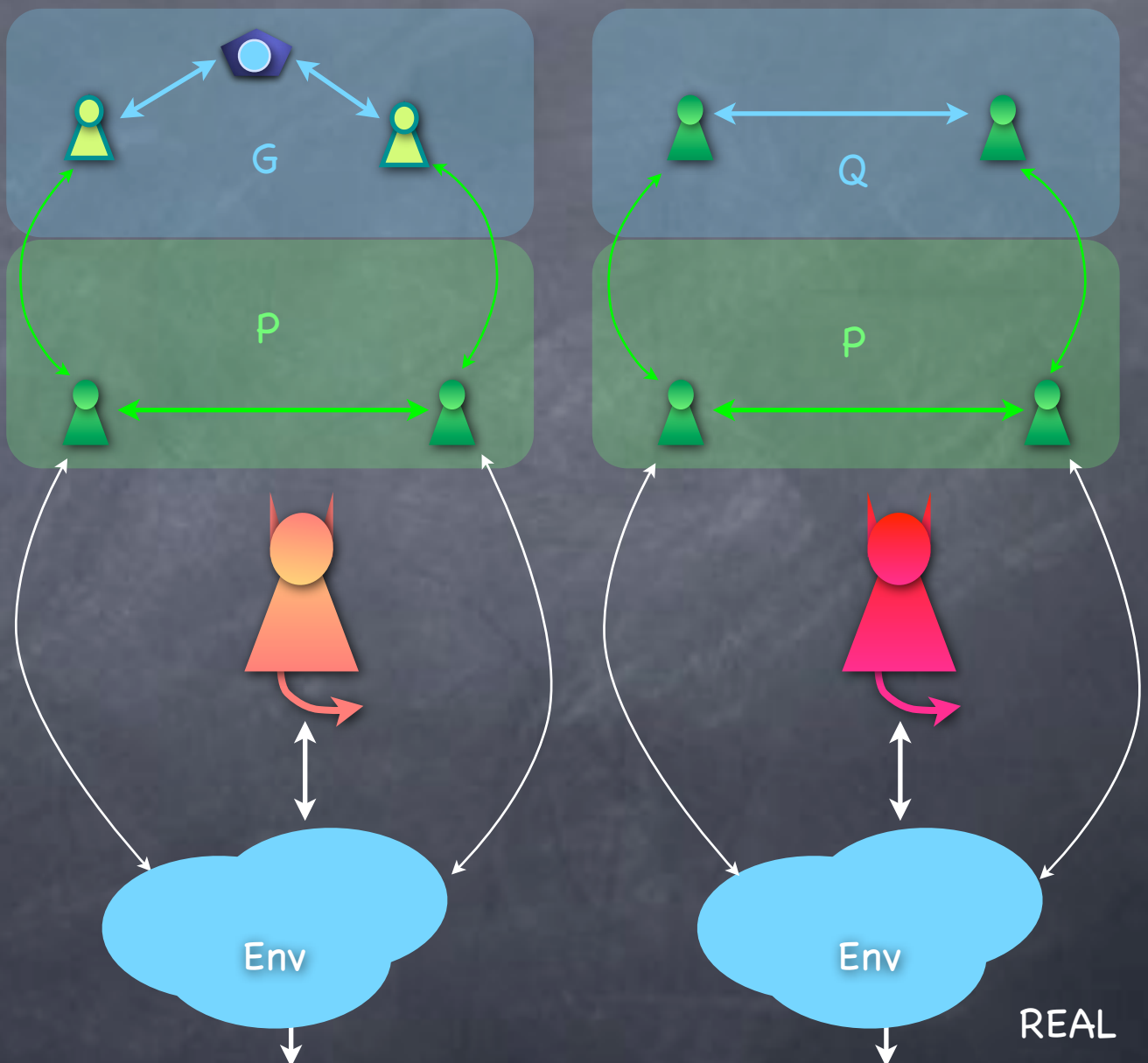
Proving the UC theorem

- Start from REAL (with an arbitrary adversary and environment)
- Consider new environment s.t. only Q (and adversary) is outside it
- Use "Q is as secure as G" to get a new world with G and a new adversary

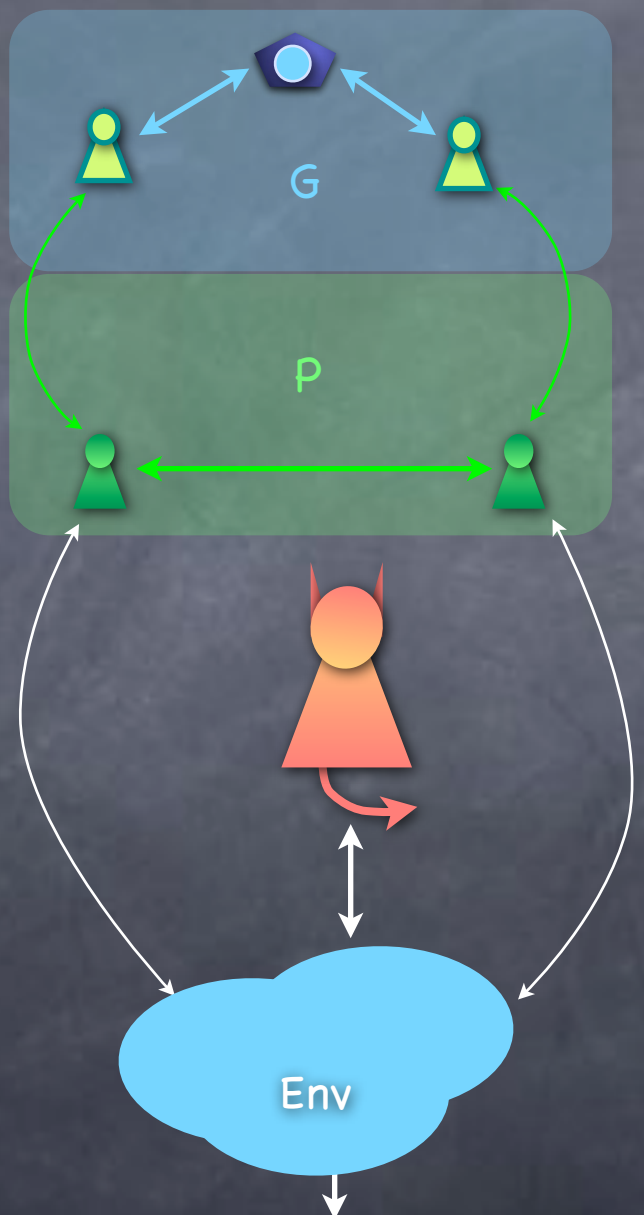


Proving the UC theorem

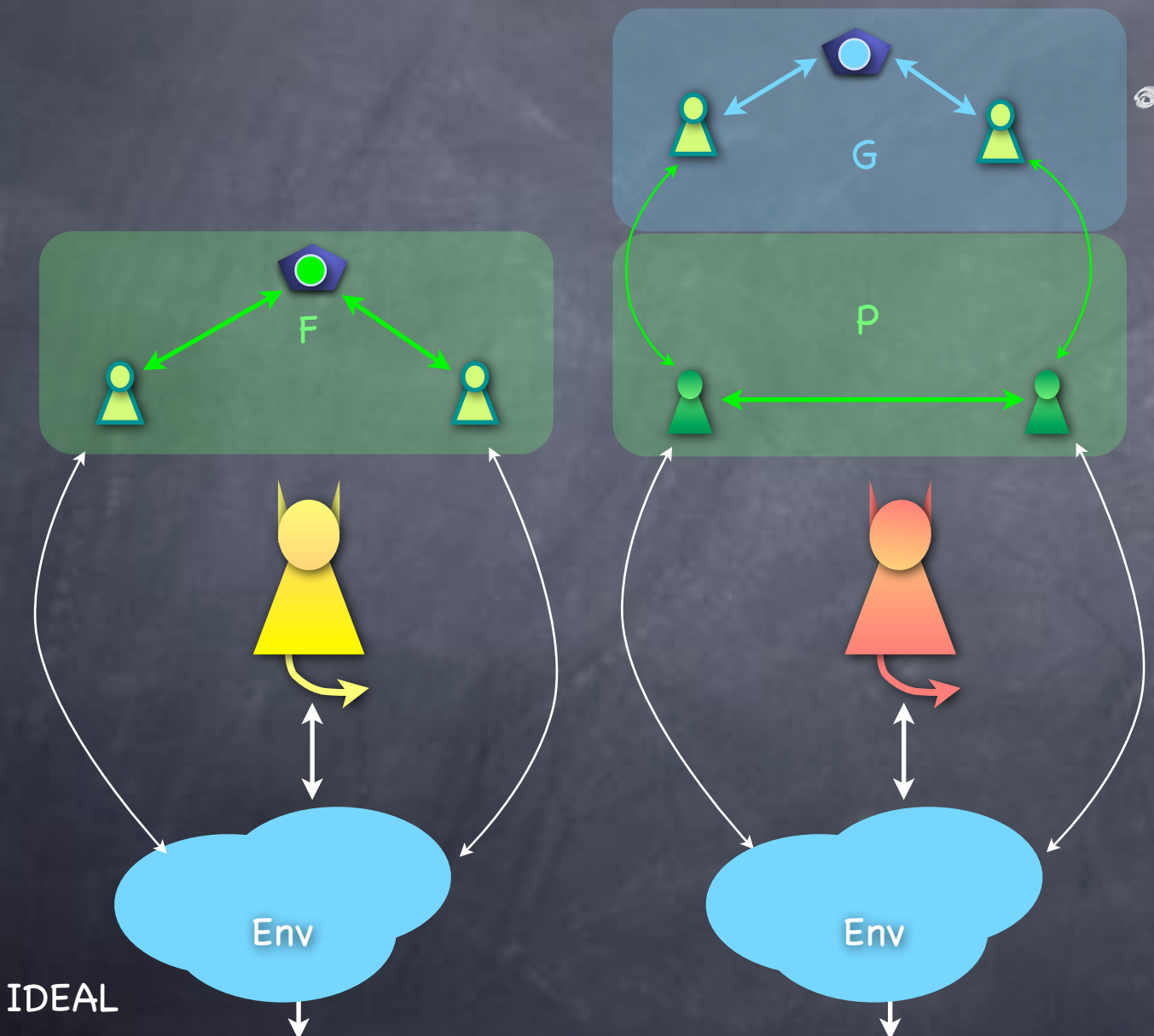
- Start from REAL (with an arbitrary adversary and environment)
- Consider new environment s.t. only Q (and adversary) is outside it
- Use "Q is as secure as G" to get a new world with G and a new adversary



Proving the UC theorem

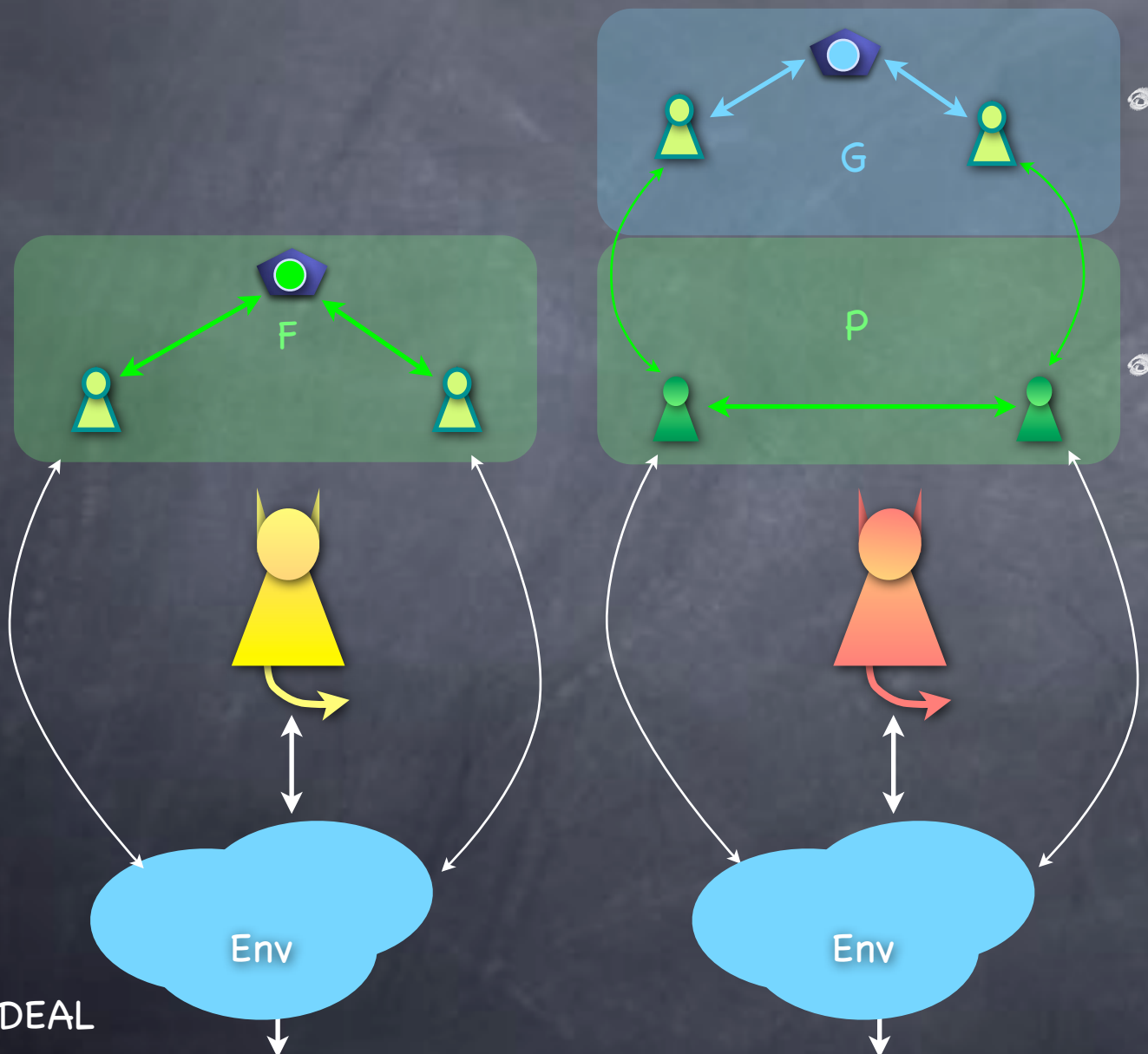


Proving the UC theorem



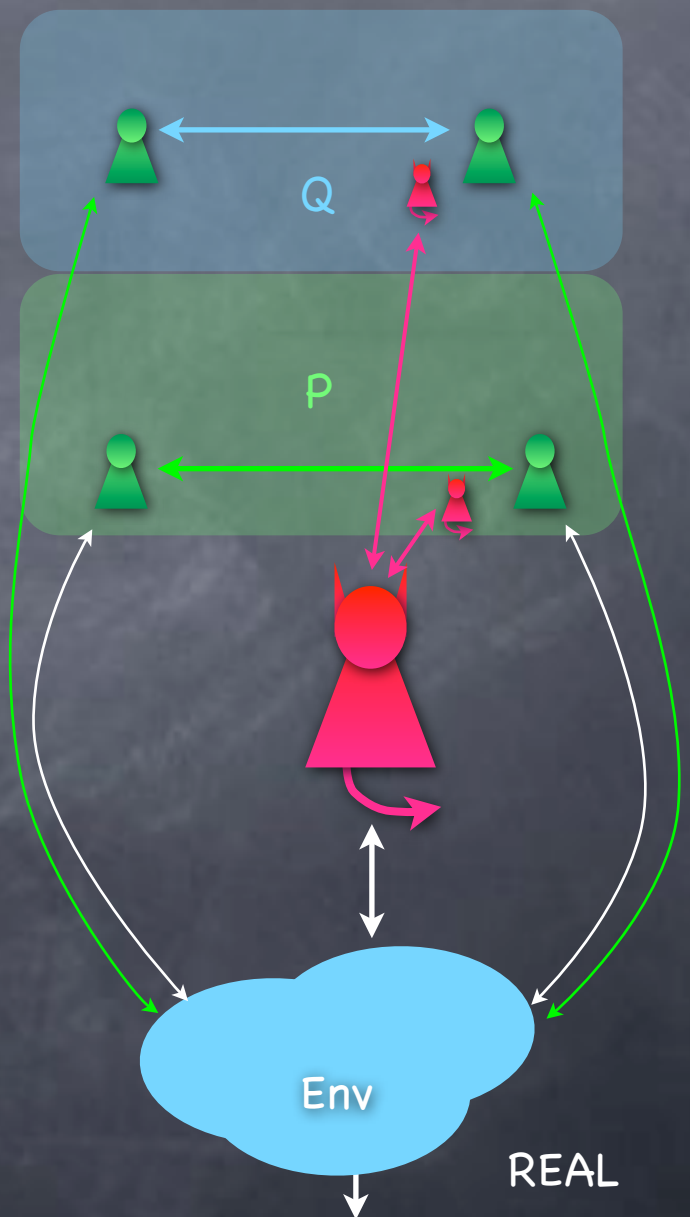
- Now use "P^G is as secure as F" guarantee to get the IDEAL world (with a new adversary)

Proving the UC theorem



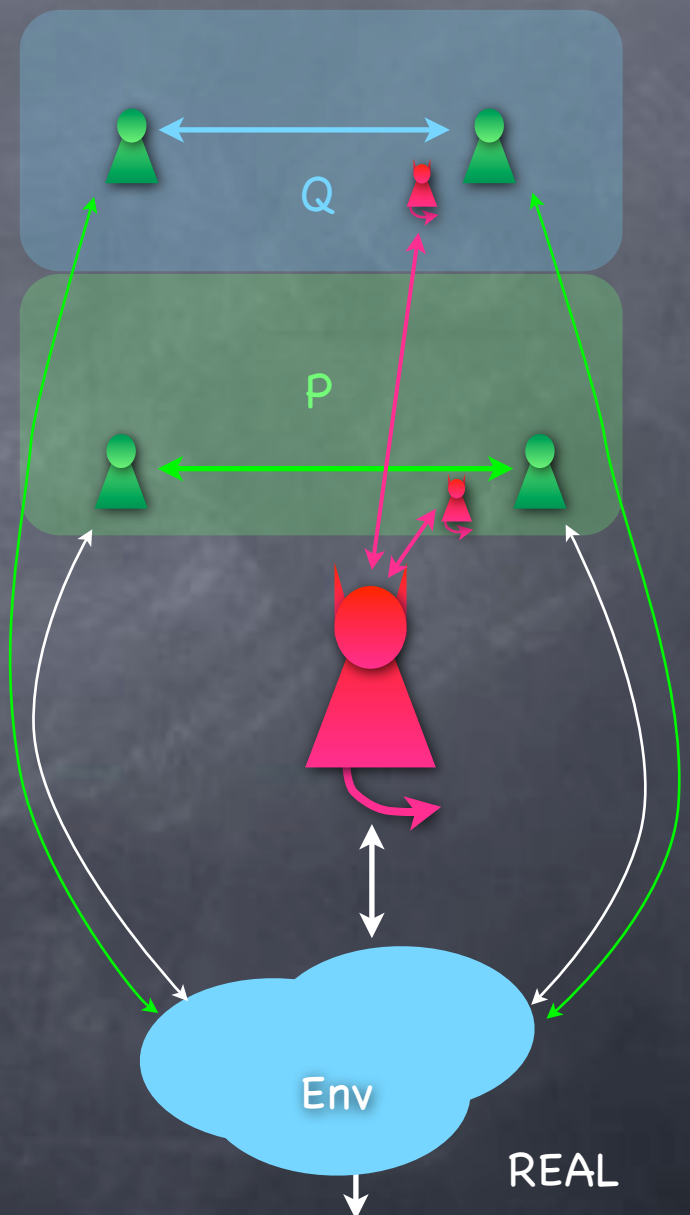
- Now use "P^G is as secure as F" guarantee to get the IDEAL world (with a new adversary)
- When concurrent sessions (instead of a single subroutine) need to be slightly more careful

Proving the UC theorem



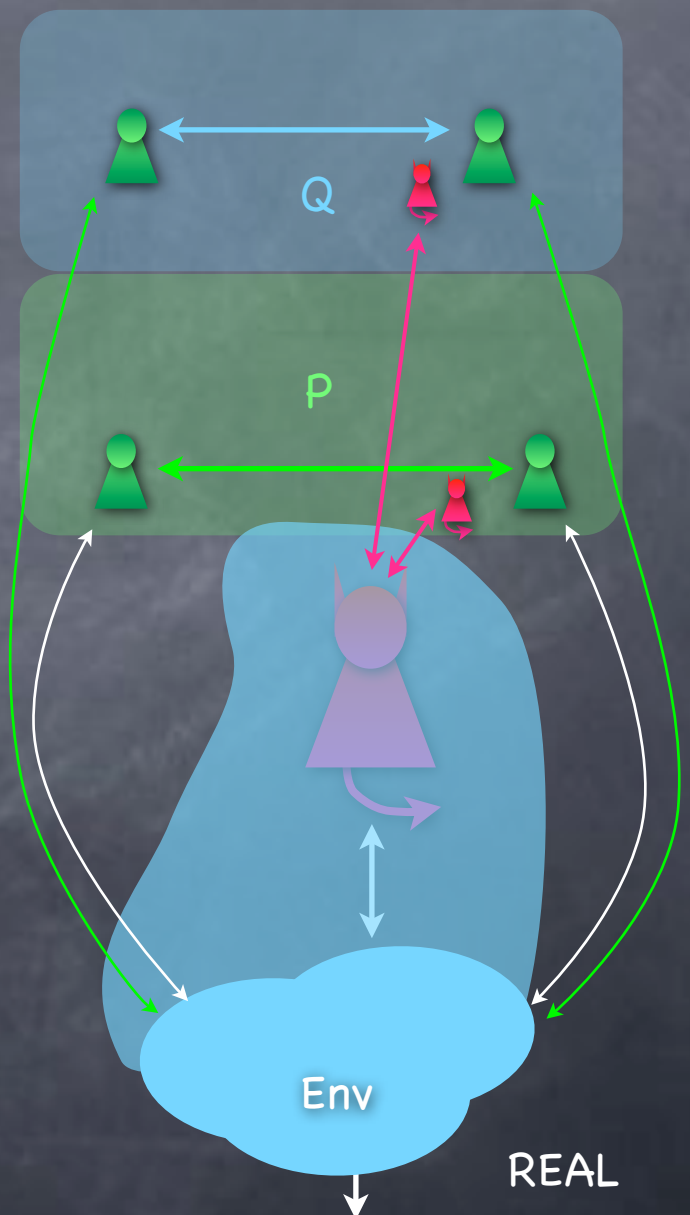
Proving the UC theorem

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



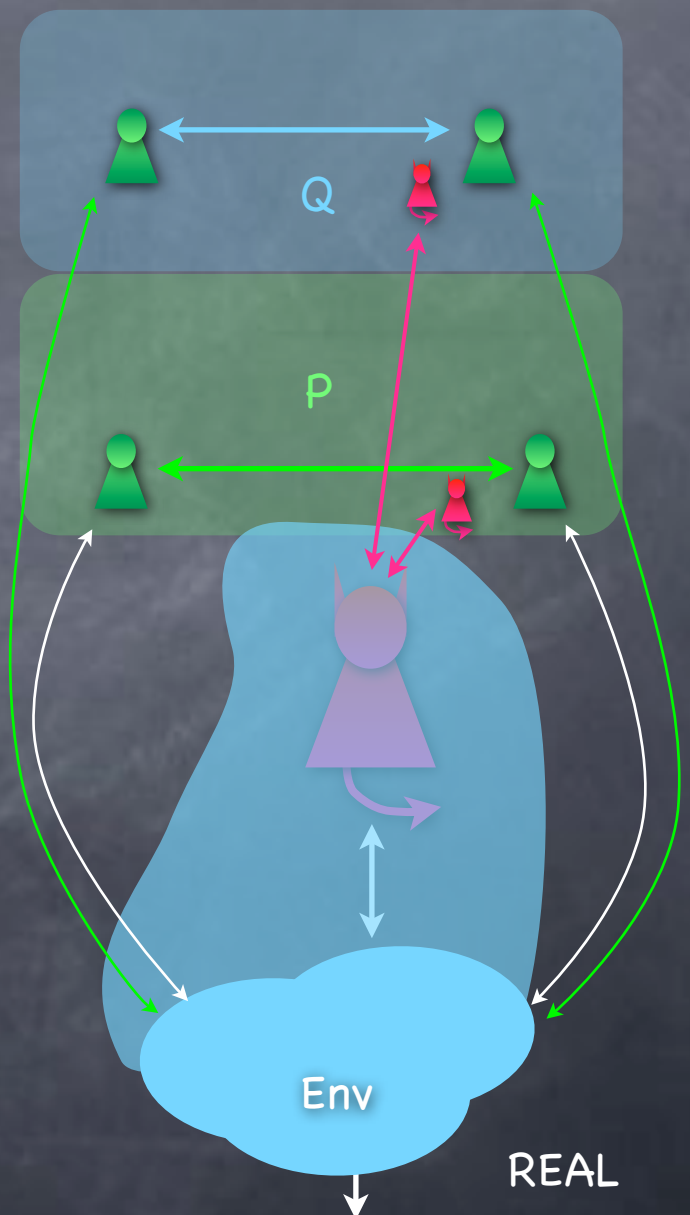
Proving the UC theorem

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



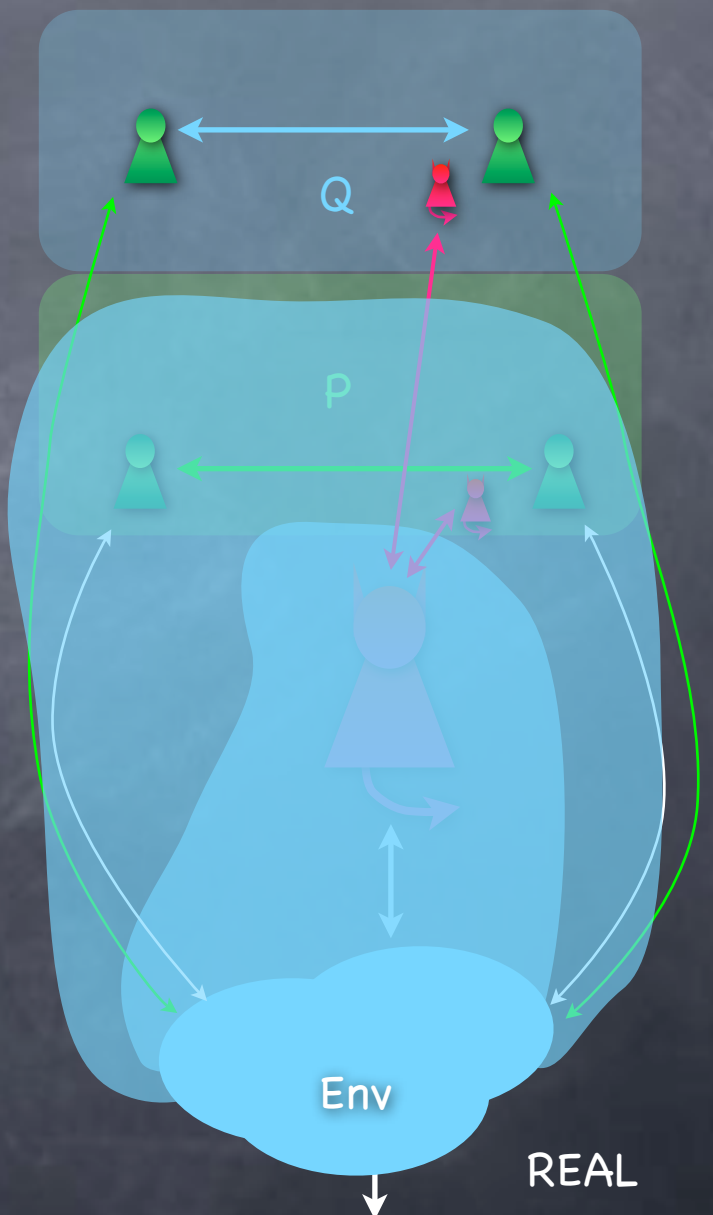
Proving the UC theorem

- 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 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 adversary) is outside it

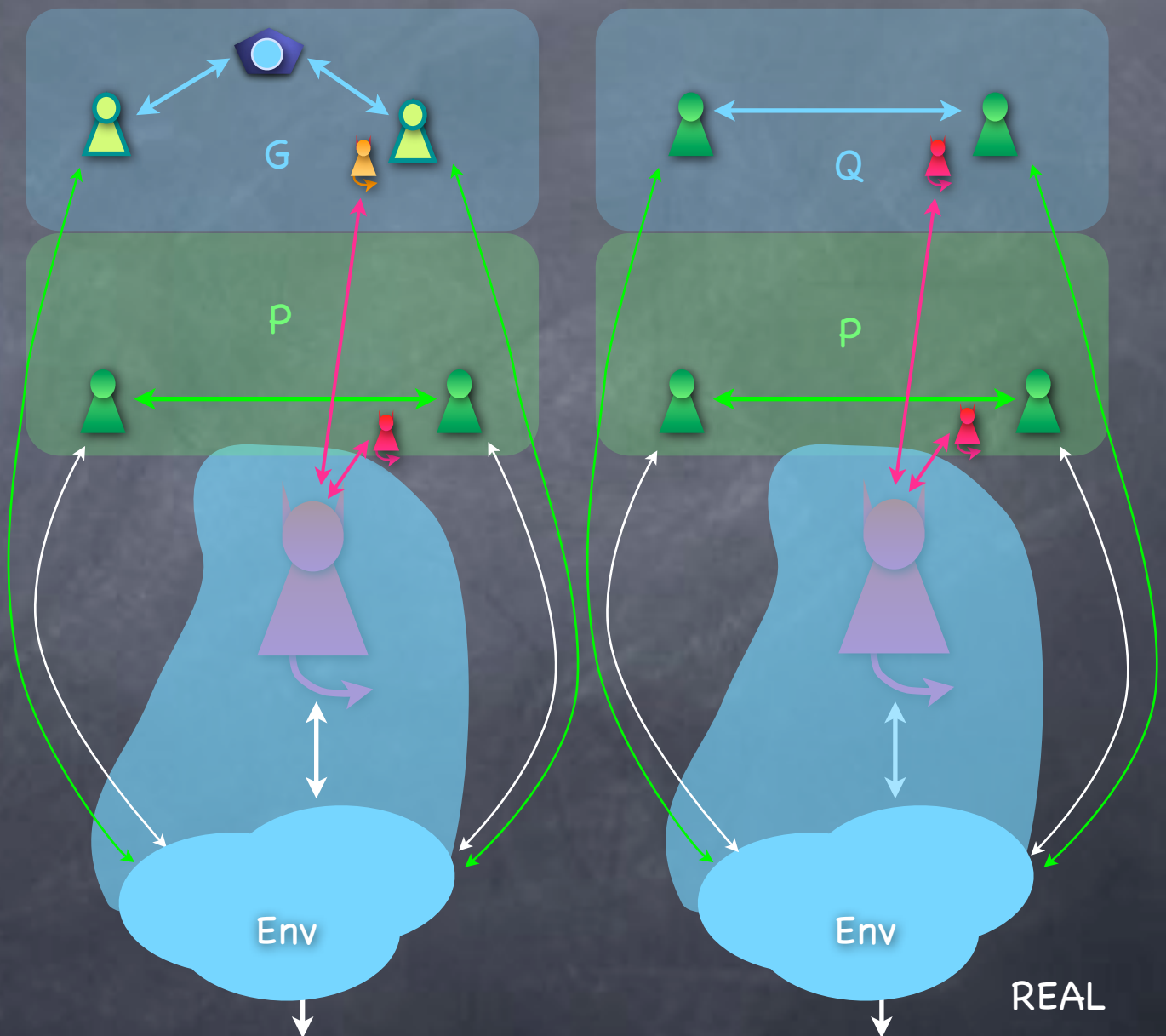


Proving the UC theorem

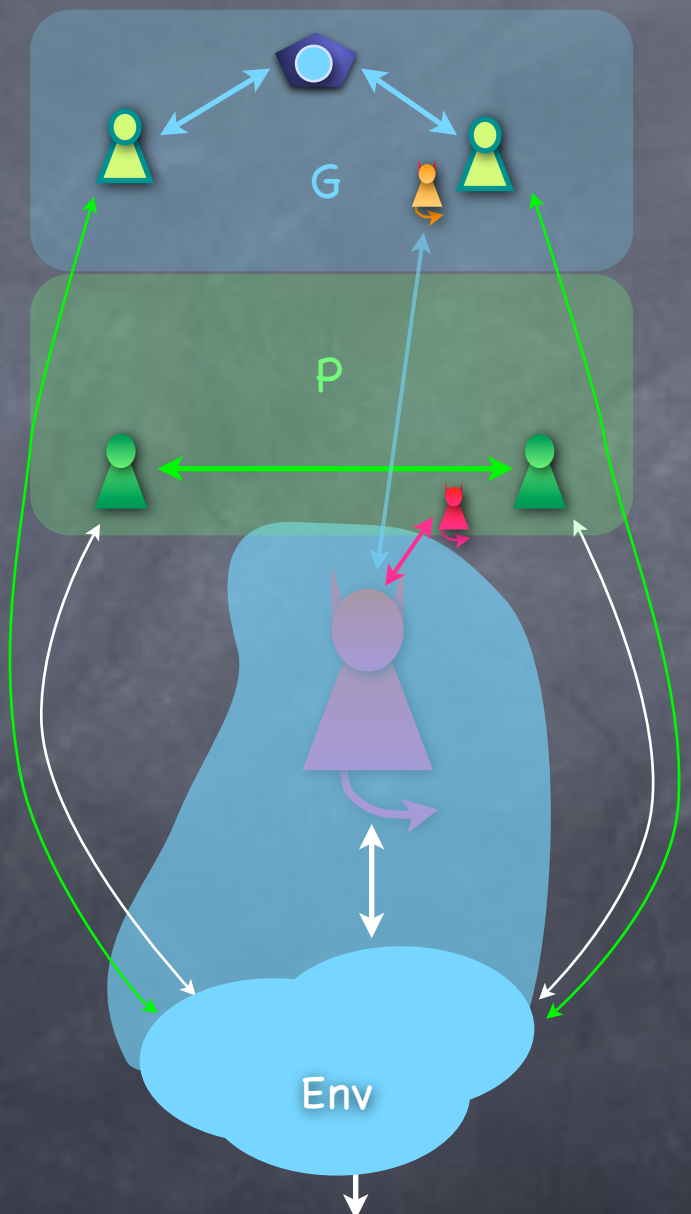
- Consider environment which runs the adversary internally, and depends on "dummy adversaries" to interface with the protocols
- Now consider new environment s.t. only Q (and adversary) is outside it
- Use "Q is as secure as G" to get a new world with G and a new adversary

Proving the UC theorem

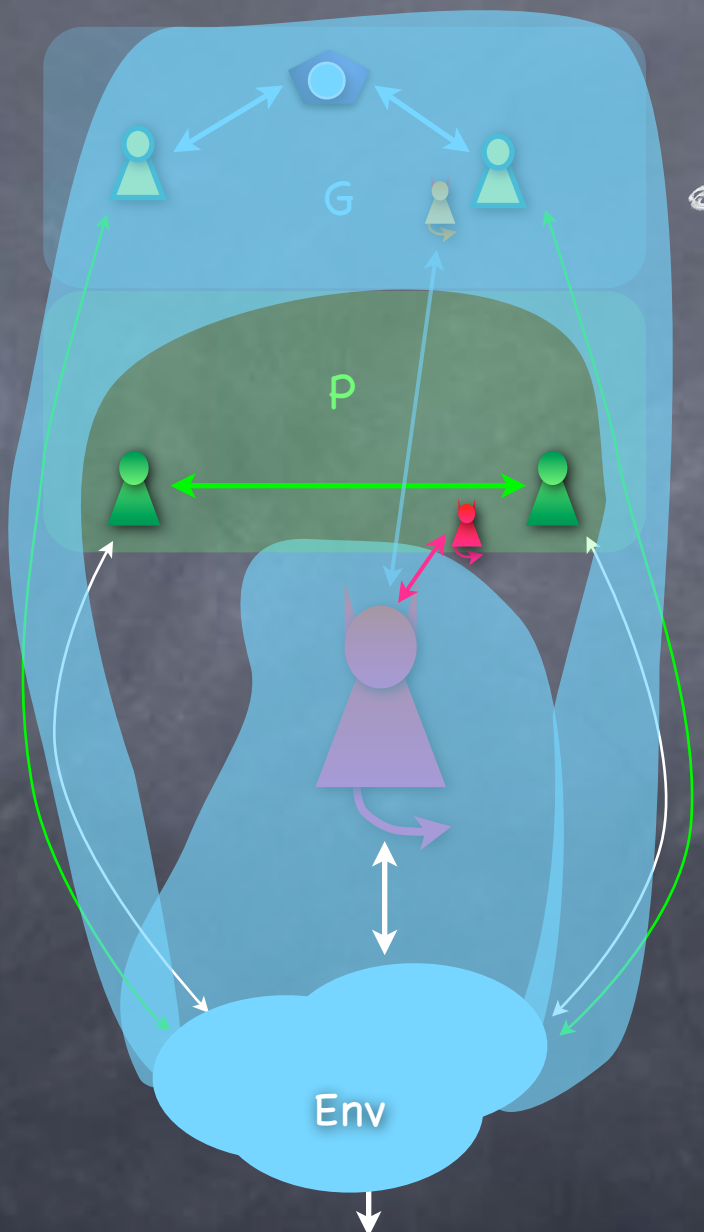
- Consider environment which runs the adversary internally, and depends on "dummy adversaries" to interface with the protocols
- Now consider new environment s.t. only Q (and adversary) is outside it
- Use "Q is as secure as G" to get a new world with G and a new adversary



Proving the UC theorem

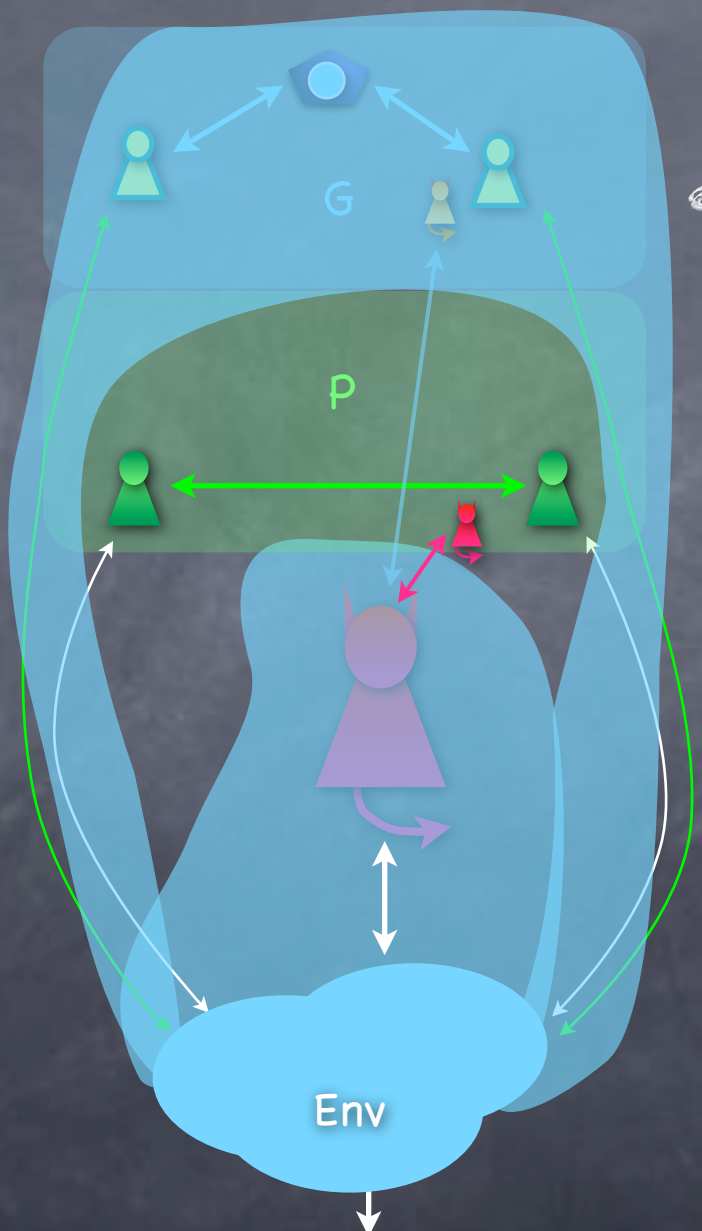


Proving the UC theorem



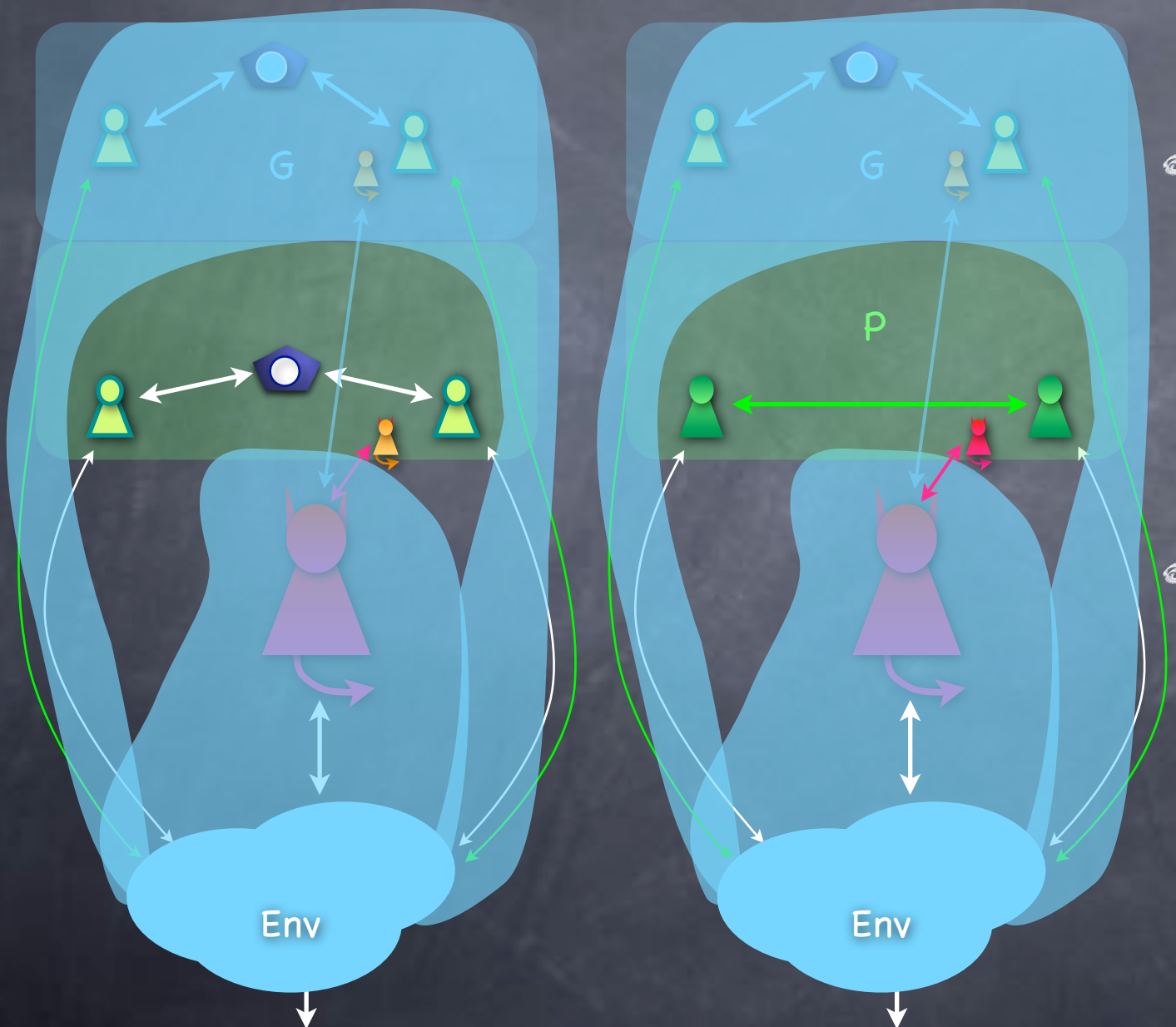
- Now consider new environment s.t. only P (and adversary) is outside it

Proving the UC theorem



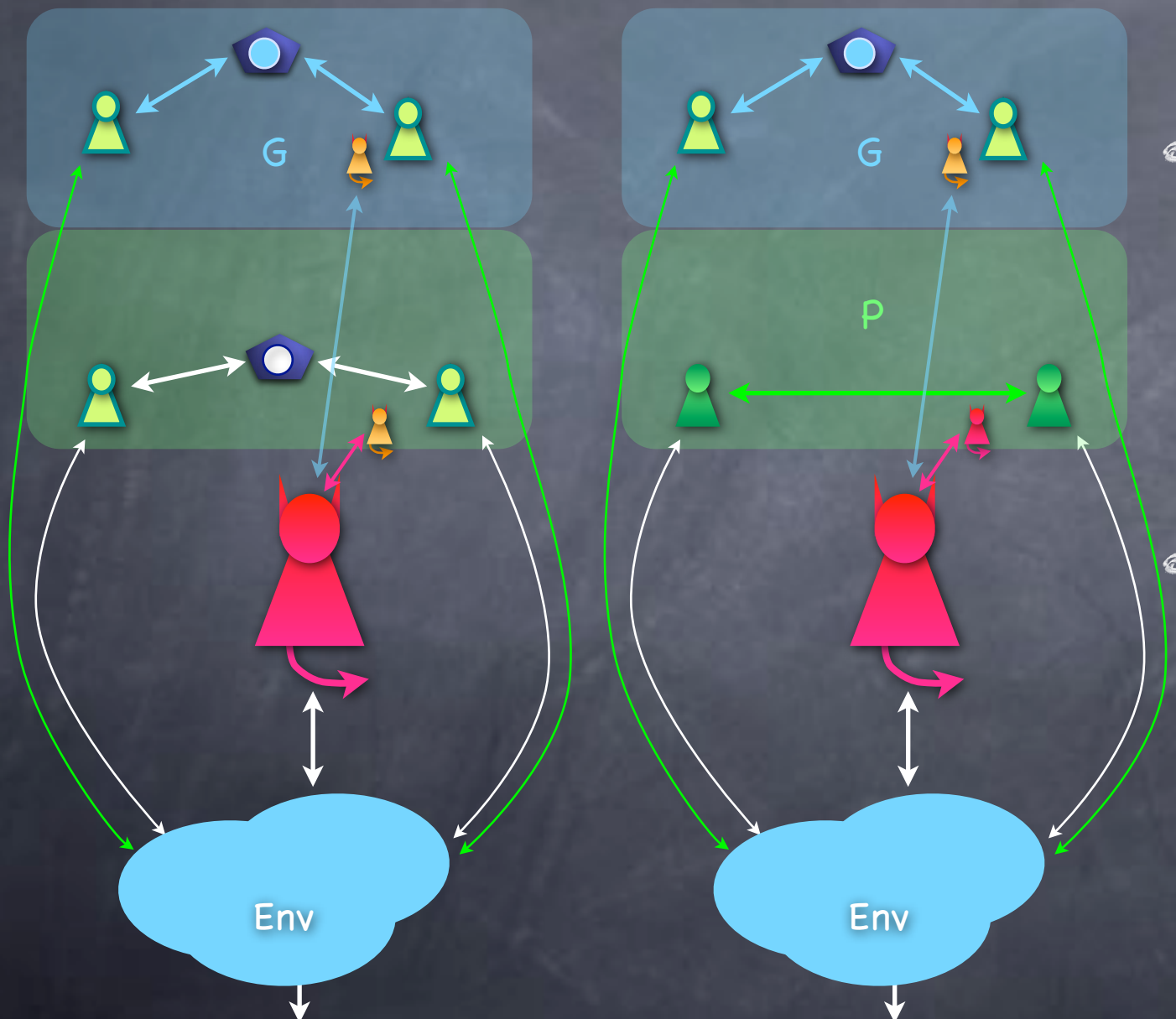
- Now consider new environment s.t. only P (and adversary) is outside it
  - Note: G and simulator for Q/G are inside the new environment

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

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 (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 (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 (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 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 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 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 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 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

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
- Initially all parties receive an additive share of the value on each of the input wires (sent by the party owning that wire). e.g., $x^{(i)}$ share for party i for wire x; $x^{(1)} + x^{(2)} + \ldots + x^{(m)} = x$

Shared Evaluation (GMW)

- (Against passive corruption)
- Circuit for evaluating the function: AND (.) and XOR (+) gates
- Initially all parties receive an additive share of the value on each of the input wires (sent by the party owning that wire). e.g., $x^{(i)}$ share for party i for wire x; $x^{(1)} + x^{(2)} + \ldots + x^{(m)} = x$
- 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
- Initially all parties receive an additive share of the value on each of the input wires (sent by the party owning that wire). e.g., $x^{(i)}$ share for party i for wire x; $x^{(1)} + x^{(2)} + \ldots + x^{(m)} = x$
- XOR evaluations done locally: if z=x+y e.g. $z^{(i)}=x^{(i)}+y^{(i)}$
- For AND: need $z^{(1)} + z^{(2)} + \ldots + z^{(m)} = [x^{(1)}+\ldots+x^{(m)}] [y^{(1)}+\ldots+y^{(m)}]$ (and $z^{(i)}$ random otherwise)

Shared Evaluation (GMW)

- (Against passive corruption)
- Circuit for evaluating the function: AND (.) and XOR (+) gates
- Initially all parties receive an additive share of the value on each of the input wires (sent by the party owning that wire). e.g., $x^{(i)}$ share for party i for wire x; $x^{(1)} + x^{(2)} + \ldots + x^{(m)} = x$
- XOR evaluations done locally: if z=x+y e.g. $z^{(i)}=x^{(i)}+y^{(i)}$
- For AND: need $z^{(1)} + z^{(2)} + \ldots + z^{(m)} = [x^{(1)}+\ldots+x^{(m)}] [y^{(1)}+\ldots+y^{(m)}]$ (and $z^{(i)}$ random otherwise)
  - For every pair of parties i,j, need to re-share $x^{(i)}y^{(j)}+x^{(j)}y^{(i)}$

Shared Evaluation (GMW)

- (Against passive corruption)
- Circuit for evaluating the function: AND (.) and XOR (+) gates
- Initially all parties receive an additive share of the value on each of the input wires (sent by the party owning that wire). e.g., $x^{(i)}$ share for party i for wire x; $x^{(1)} + x^{(2)} + \ldots + x^{(m)} = x$
- XOR evaluations done locally: if z=x+y e.g. $z^{(i)}=x^{(i)}+y^{(i)}$
- For AND: need $z^{(1)} + z^{(2)} + \ldots + z^{(m)} = [x^{(1)}+\ldots+x^{(m)}]\ [y^{(1)}+\ldots+y^{(m)}]$ (and $z^{(i)}$ random otherwise)
  - For every pair of parties i,j, need to re-share $x^{(i)}y^{(j)}+x^{(j)}y^{(i)}$
  - Done using OT (Party i prepares 4 values indexed by $x^{(j)}y^{(j)}$)

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

- Uses a "commit-and-prove" (CaP) functionality

Security against active corruption

- Uses a "commit-and-prove" (CaP) functionality
  - Party i can send x; all parties get "committed"

Security against active corruption

- Uses a "commit-and-prove" (CaP) functionality
  - Party i can send x; all parties get "committed"
  - Later send statement R s.t. R(x) holds; all parties get R

Security against active corruption

- Uses a "commit-and-prove" (CaP) functionality
  - Party i can send x; all parties get "committed"
  - Later send statement R s.t. R(x) holds; all parties get R
    - e.g. $R_y(x)$: x=y (opening the commitment)

Security against active corruption

- Uses a "commit-and-prove" (CaP) functionality
  - Party i can send x; all parties get "committed"
  - Later send statement R s.t. R(x) holds; all parties get R
    - e.g. $R_y(x)$: x=y (opening the commitment)
- Can use for "coin-tossing into the well"

Security against active corruption

- Uses a "commit-and-prove" (CaP) functionality
  - Party i can send x; all parties get "committed"
  - Later send statement R s.t. R(x) holds; all parties get R
    - e.g. $R_y(x)$: x=y (opening the commitment)
- Can use for "coin-tossing into the well"
  - Only party i gets the random string, but can later prove statements involving that string to the others

Security against active corruption

- Uses a "commit-and-prove" (CaP) functionality
  - Party i can send x; all parties get "committed"
  - Later send statement R s.t. R(x) holds; all parties get R
    - e.g. $R_y(x)$: x=y (opening the commitment)
- Can use for "coin-tossing into the well"
  - Only party i gets the random string, but can later prove statements involving that string to the others
  - 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

- Uses a "commit-and-prove" (CaP) functionality
  - Party i can send x; all parties get "committed"
  - Later send statement R s.t. R(x) holds; all parties get R
    - e.g. $R_y(x)$: x=y (opening the commitment)
- Can use for "coin-tossing into the well"
  - Only party i gets the random string, but can later prove statements involving that string to the others
  - 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, but also commit to input x_i along with r's

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, but also commit to input x_i along with r's
  - 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, but also commit to input x_i along with r's
  - 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
    - A statement about the messages so far (publicly known) and randomness and input (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
  - Impossible in the plain model

Today

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

Today

- Universal Composition
  - SIM security definition gives universal composition
- SIM-secure MPC
  - Impossible in the plain model
  - Possible with various modified SIM-security definitions (still UC)
  - e.g.: GMW-style (first for passive corruption, and use CaP to transform it for active corruption); uses CaP