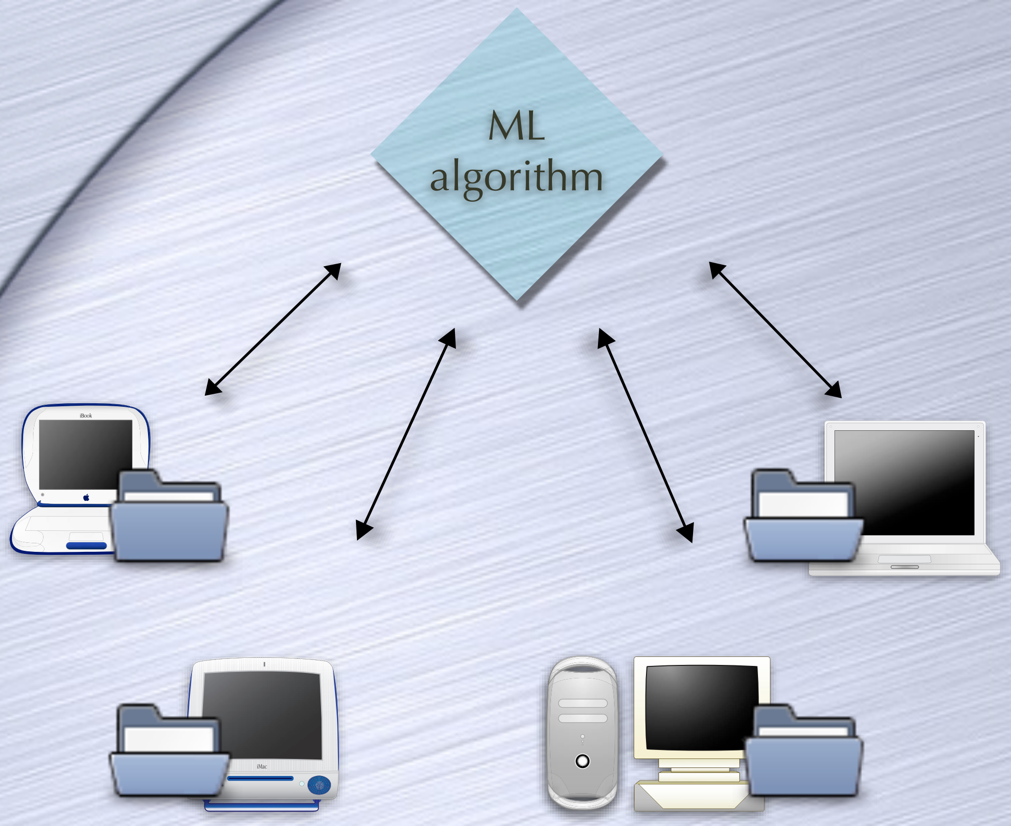
Secure Multi-Party Computation

A Quick Introduction

Manoj Prabhakaran :: IIT Bombay

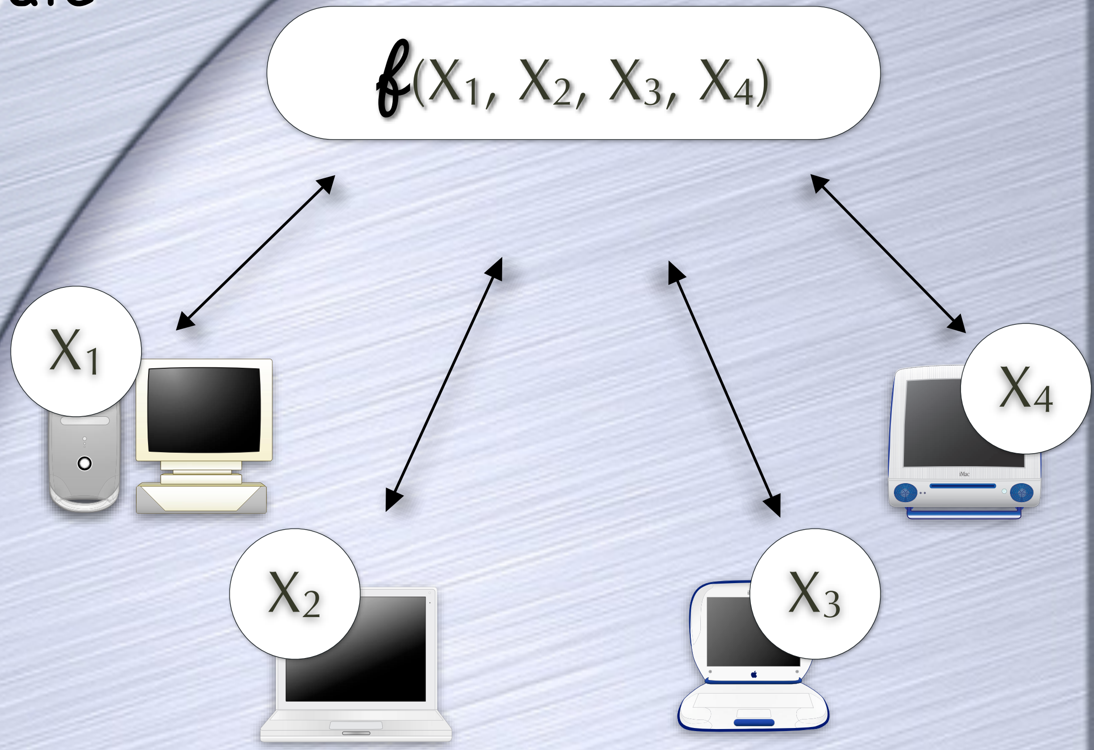
Using data without sharing?

- Hospitals which can't share their patient records with anyone
  - But want to learn from the combined data



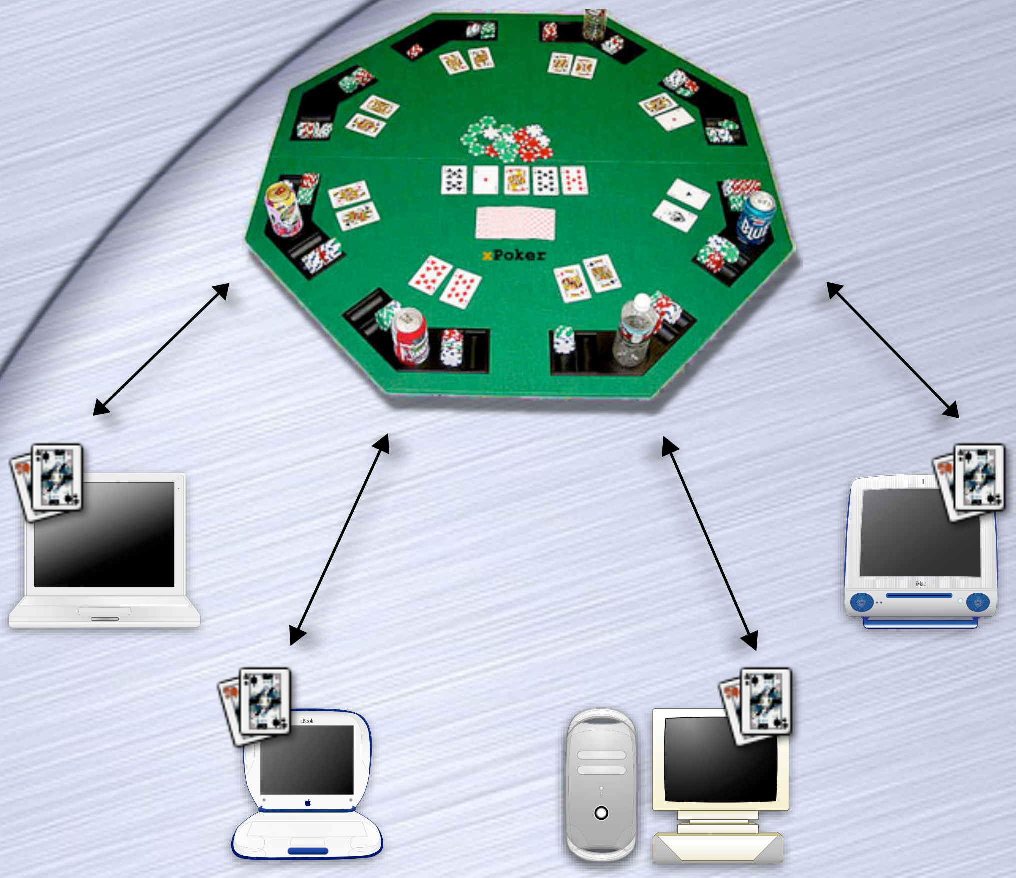
Secure Function Evaluation

- A general problem
- To compute a function of private inputs without revealing information about the inputs
  - Beyond what is revealed by the function



Poker With No Dealer?

- Need to ensure
  - Cards are shuffled and dealt correctly
  - Complete secrecy
  - No “cheating” by players, even if they collude
- No universally trusted dealer



The Ambitious Goal

- Without any trusted party, securely do
  - Distributed Machine Learning
  - E-commerc
  - Network G
  - E-voting
  - Secure fun
  - ....

Secure
Multi-Party Computation
(MPC)

Emulating Trusted Computation

- Encryption/Authentication allow us to emulate a trusted channel
- Secure MPC: to emulate a source of trusted computation
  - Trusted means it will not "leak" a party's information to others
  - And it will not cheat in the computation
- A tool for mutually distrusting parties to collaborate

Mental Poker

Adi Shamir, Ronald L. Rivest and Leonard M. Adleman

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ABSTRACT

Can two potentially dishonest players play a fair game of poker without using any cards—for example, over the phone? This paper provides the following answers:

1 No. (Rigorous mathematical proof supplied.)
2 Yes. (Correct and complete protocol given.)

This Tutorial

- What does it mean to be secure?
- How does one do MPC?
  - Warm up
  - Some classical protocols for computing “general” functions (will focus on passive corruption)
    - GMW
    - BGW
    - Yao’s Garbled Circuits
- Glimpses of various concepts

What does it mean to be Secure?

Terminology

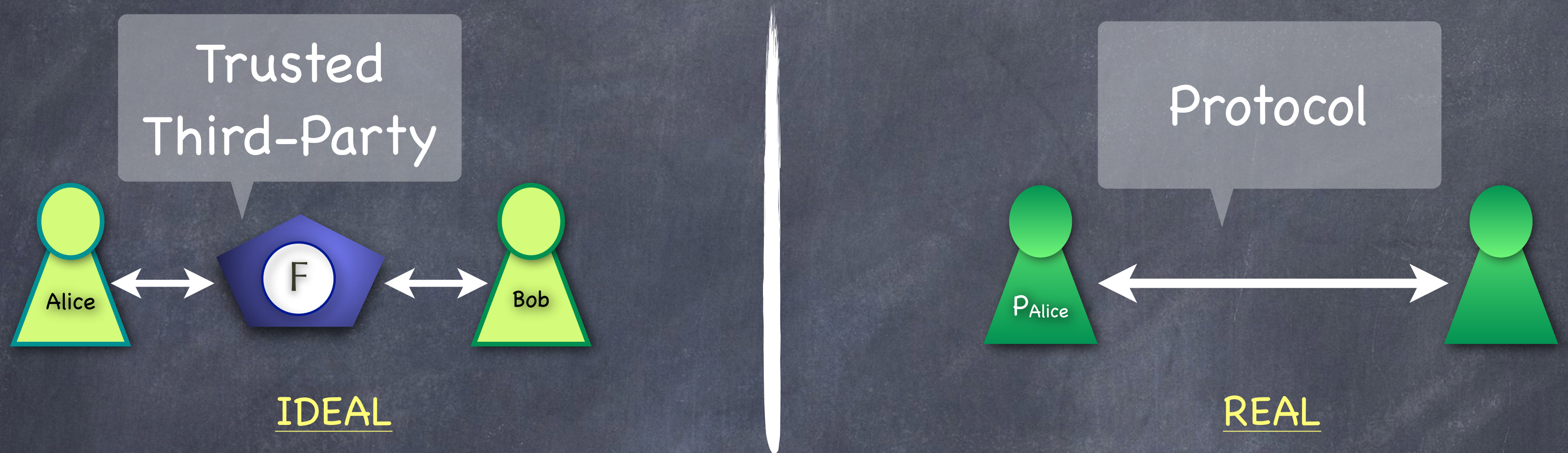
- Protocol: Instructions to the (honest) parties on what messages to send to whom based on input/local randomness and messages received so far.
  - The next-message function
- Functionality: What we are aiming to achieve
  - Specified as the program of a trusted party

Security Issues to Consider

- Protocol may leak a party's secrets
  - Clearly an issue
  - Even if we trust everyone not to cheat in our protocol (i.e., honest-but-curious)
    - Also, a liability for a party if extra information reaches it (e.g., in medical data mining)
- Protocol may give adversary illegitimate influence on the outcome
  - Say in poker, if adversary can influence hands dealt
  - In auction, if adversary can choose its bid to just beat the others'

Defining Security

- REAL/IDEAL paradigm



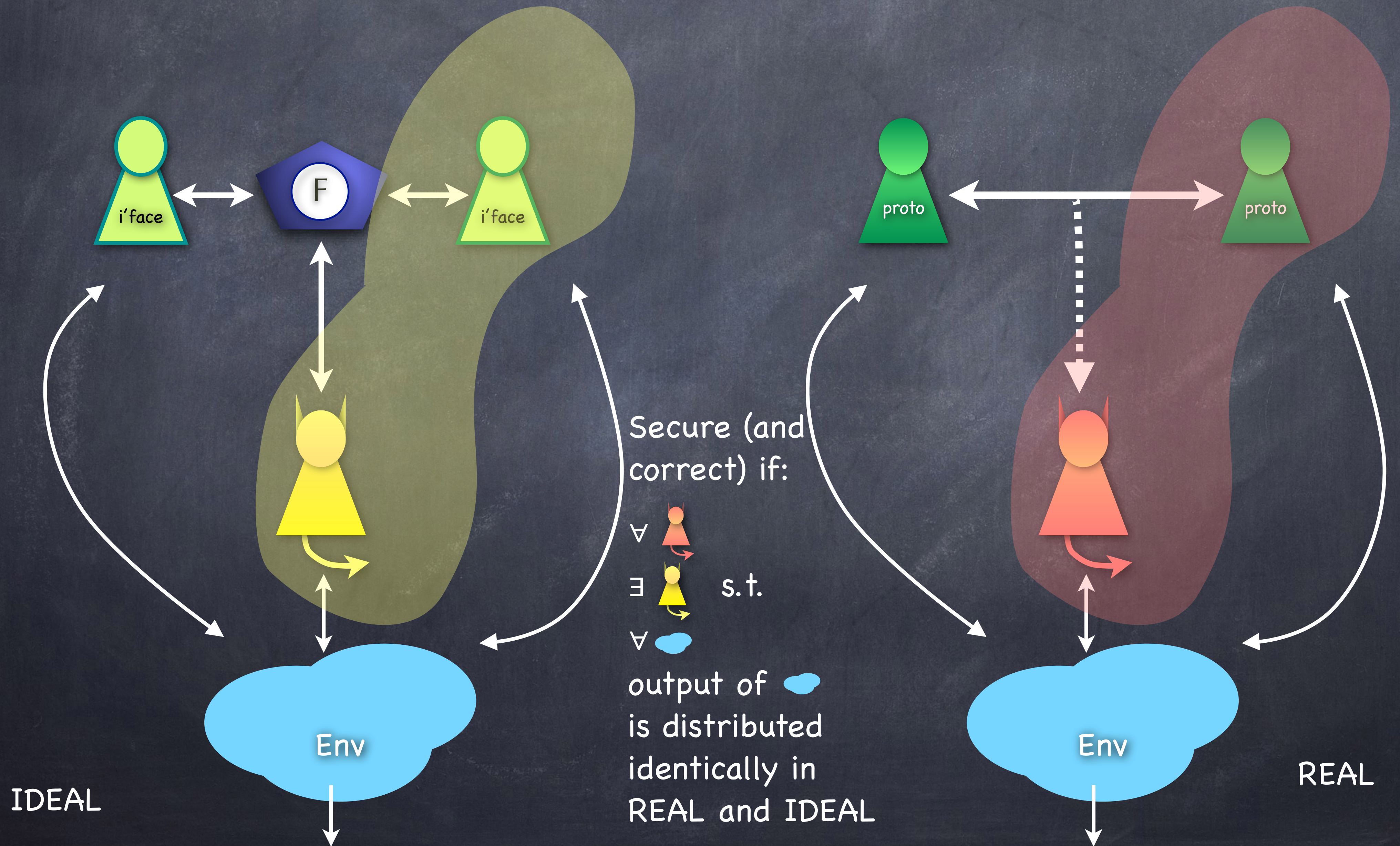
- Security guarantee: Whatever an adversary can do in the REAL world, an adversary could have done the same in the IDEAL world
  - Can't blame the protocol for anything undesirable

Adversary

- REAL-adversary can corrupt any set of players
  - IDEAL-adversary should corrupt the same set of players
- More sophisticated notion: adaptive adversary which corrupts players dynamically during/after the execution
  - We'll stick to static adversaries
- Passive vs. Active adversary: Passive adversary gets only read access to the internal state of the corrupted players. Active adversary overwrites their state and program.

Universally Composable [Canetti'01]

Defining Security



Secure (and correct) if:

∀

∃ s.t.

∀

output of is distributed identically in REAL and IDEAL

(Some) Security Models

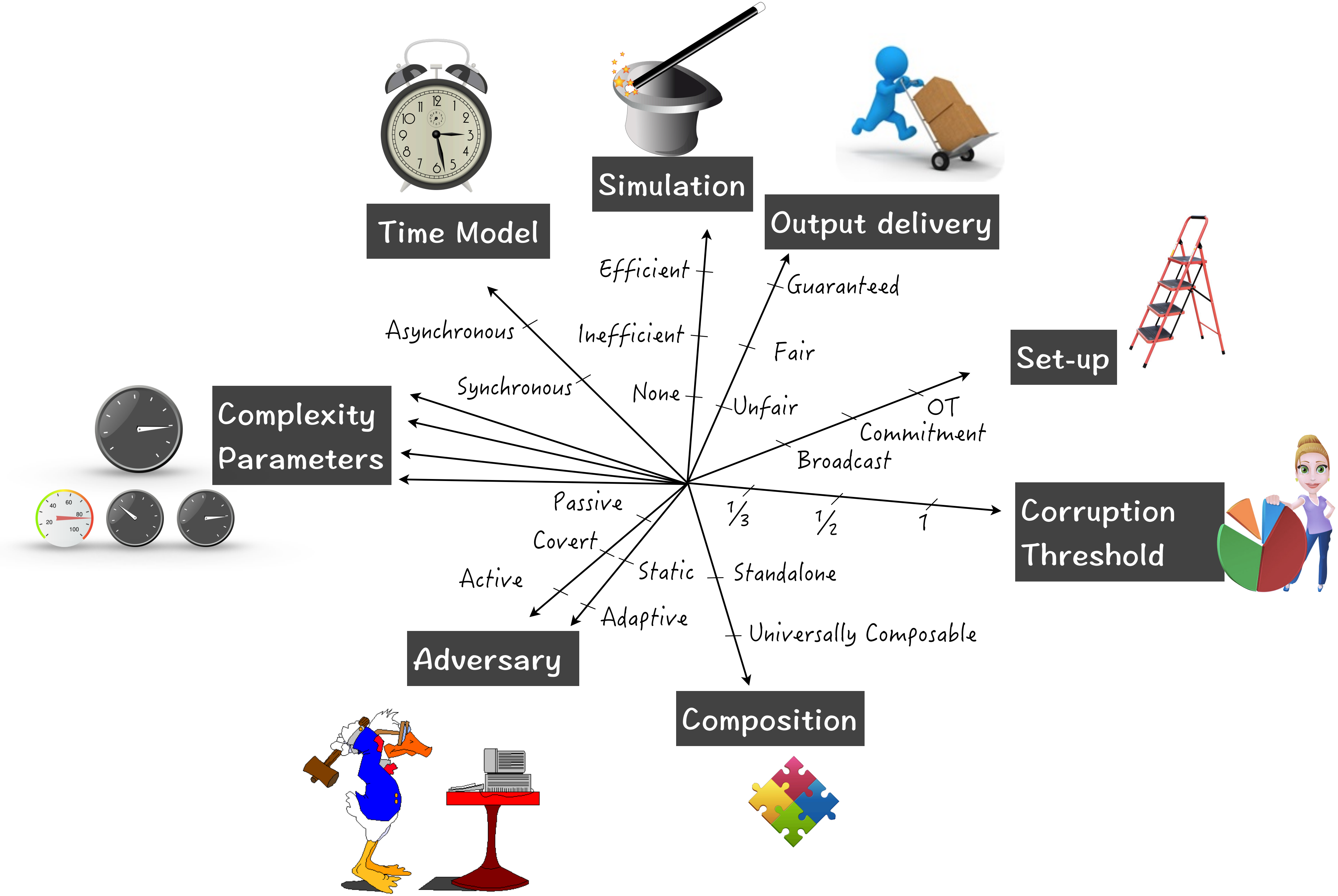
- UC security: Standard simulation-based security model
- Passive (a.k.a honest-but-curious) adversary: where corrupt parties stick to the protocol (but we don't want to trust them with information)
- Honest-majority security: adversary can corrupt only a strict minority of parties. (Not useful when only two parties involved)
- Standalone security: environment is not "live": interacts with the adversary before and after (but not during) the protocol
- Functionality-specific non-simulation-based definitions: usually leave out subtle attacks (e.g. malleability related attacks)
- Protocols using a trusted party for some basic functionality (a.k.a. setup)
- Angel-UC (UC + a helpful oracle for adversary in the ideal world)

Is MPC Possible?

- Can we securely realize every functionality?
- No & Yes!

Univ. Composable / Angel-UC / Standalone / Passive	All subsets corruptible	Honest Majority
Computationally Unbounded (No Setup)	No	Yes
Computationally Unbounded with Setup	Yes	Yes
Computationally Bounded (PPT) (No Setup)	No / Yes / Yes / Yes	Yes

MPC Dimensions



Doing MPC

Warm Up

A simple example

- An auction, with Alice and Bob bidding
- Rules:
  - A bid is an integer in the range [0,100]
  - Alice can bid only even integers and Bob odd integers
  - Person with the higher bid wins
- Goal: find out the winning bid (winner & amount) without revealing anything more about the losing bid (beyond what is revealed by the winning bid)

A simple example

- Secure protocol:
  - Count down from 100
  - At each even round Alice announces whether her bid equals the current count; at each odd round Bob does the same
  - Stop if a party says yes
- Dutch flower auction
- Perfectly secure against active adversary as well
  - Standalone. Not UC.

A second example

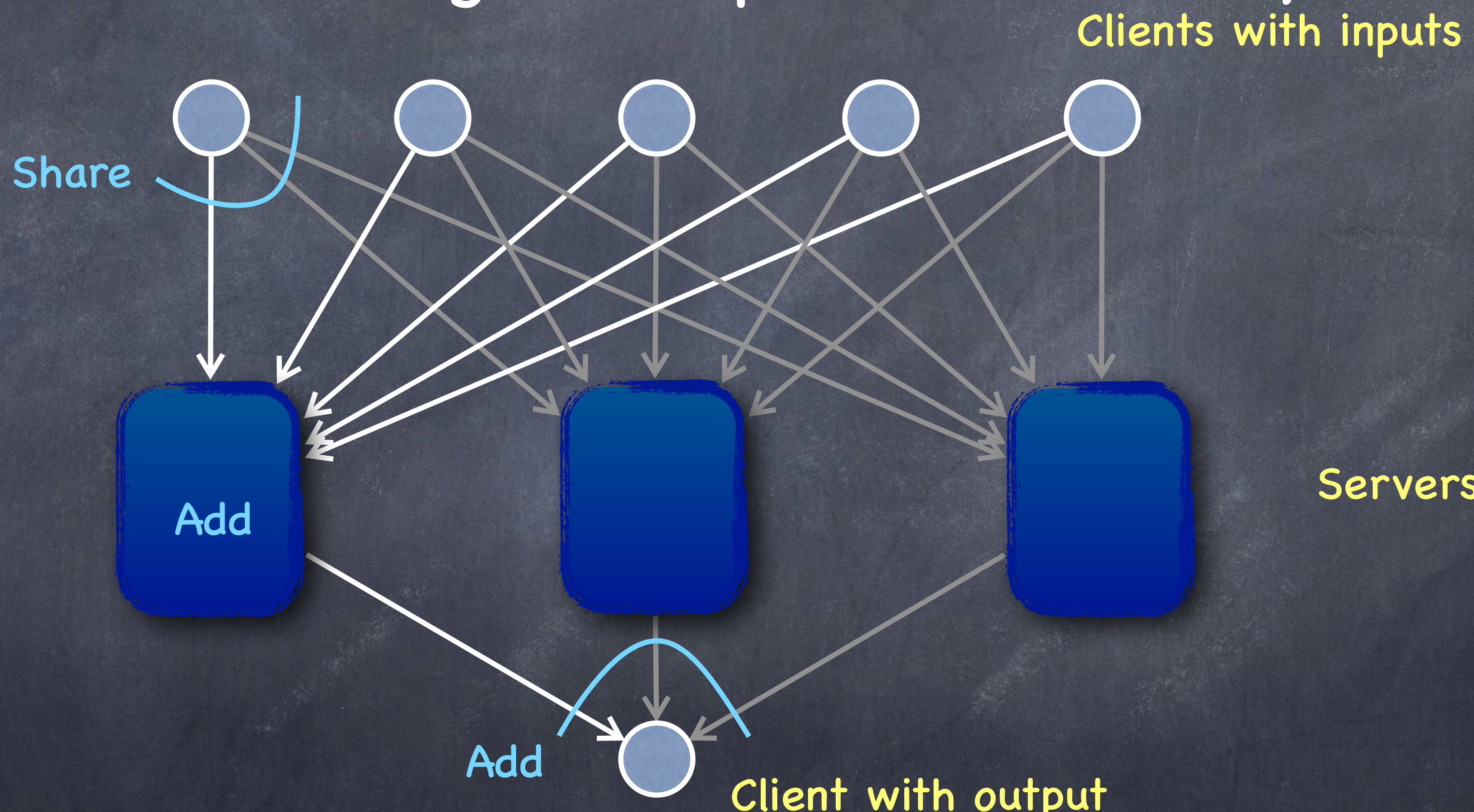
- n parties would like to sum up their inputs (integers in a certain range)
- Each party should learn only the final output and their own input (and anything that can be inferred from those two)
- Protocol idea: use additive secret sharing

Additive Secret Sharing

- Fix any "secret" s (all elements from a finite group)
- Let a, b be uniformly random conditioned on $s = a + b$.
  - e.g., pick a uniformly at random, set $b = s - a$
- Each of a, b by itself carries no information about s
- Generalises to multiple shares: $a_1,\ldots,a_n$ uniformly random conditioned on $s = a_1 + \ldots + a_n$
  - Any subset of up to n-1 shares has no information about s

A second example

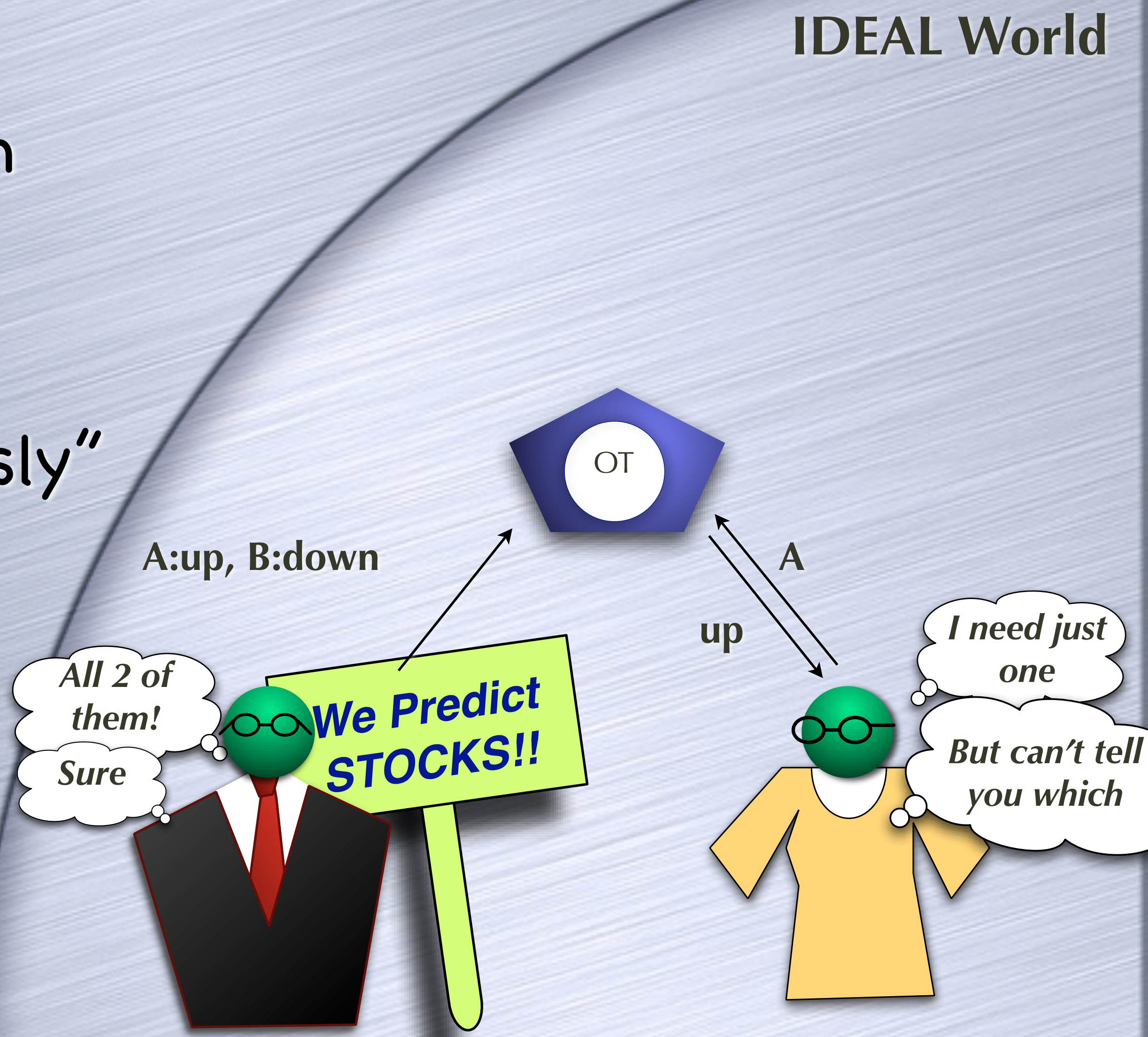
- Summation, secure against a passive adversary

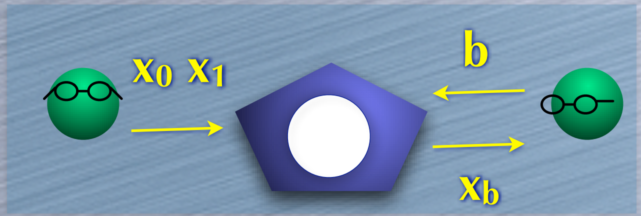


- No colluding set of servers/clients will learn more than the inputs/output of the clients in the collusion, provided that at least one server stays out of the collusion

Oblivious Transfer

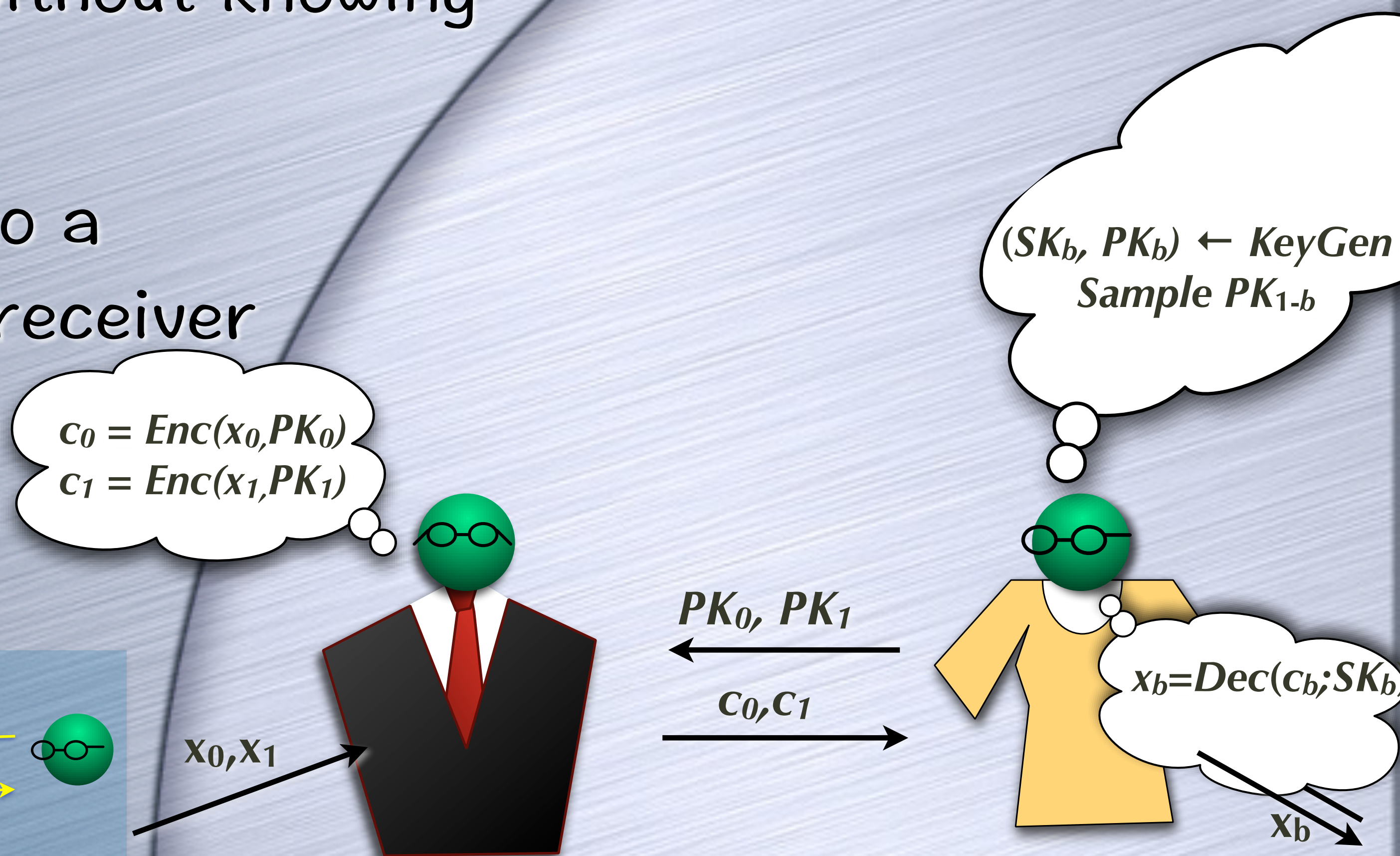
- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"





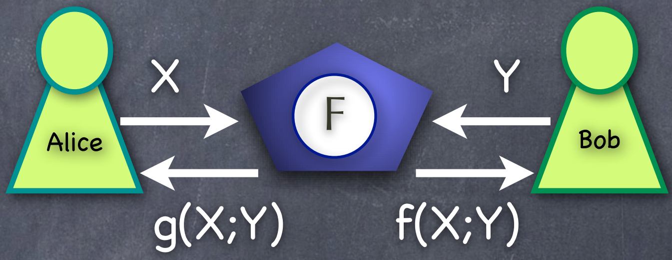
An OT Protocol (passive corruption)

- Using a (special) encryption
  - PKE in which one can sample a public-key without knowing secret-key
- c_{1-b} inscrutable to a passive corrupt receiver
- Sender learns nothing about b



2-Party SFE

- Secure Function Evaluation (SFE) IDEAL:
  - Trusted party takes (X;Y). Outputs g(X;Y) to Alice, f(X;Y) to Bob



    - Randomized Functions: g(X;Y;r) and f(X;Y;r) s.t. neither party knows r (beyond what is revealed by output)
- OT is an instance of a (deterministic) 2-party SFE
  - $g(x_0,x_1;b)$ = none; $f(x_0,x_1;b) = x_b$
  - Single-Output SFE: only one party gets any output

2-Party SFE

- Can reduce any SFE (even randomized) to a single-output deterministic SFE
  - $f'(X, M, r_1; Y, r_2) = (\ g(X; Y; r_1 \oplus r_2) \oplus M,\ f(X; Y; r_1 \oplus r_2)\ )$. Compute $f'(X, M, r_1; Y, r_2)$ with random M, r_1, r_2
  - Bob sends $g(X, Y; r_1 \oplus r_2) \oplus M$ to Alice
  - Passive secure
  - Generalizes to active security and more than 2 parties
- **Can reduce any single-output deterministic SFE to OT!**

“Completeness” of OT

- **Can reduce any single-output deterministic SFE to OT!**
- For passive security
  - Proof of concept for 2 parties: An inefficient reduction
  - “Basic GMW”: Information-theoretic reduction to OT
  - Yao’s garbled circuit for 2 parties (later today)
- In fact, OT is complete even for active security

"Completeness" of OT: Proof of Concept

- Single-output 2-party function f
- Alice (who knows x, but not y) prepares a table for $f(x,\cdot)$ with $N = 2^{|y|}$ entries (one for each y)
- Bob uses y to decide which entry in the table to pick up using 1-out-of-N OT (without learning the other entries)
- Bob learns only f(x,y) (in addition to y). Alice learns nothing beyond x.
- Problem: N is exponentially large in |y|

1-out-of-N OT

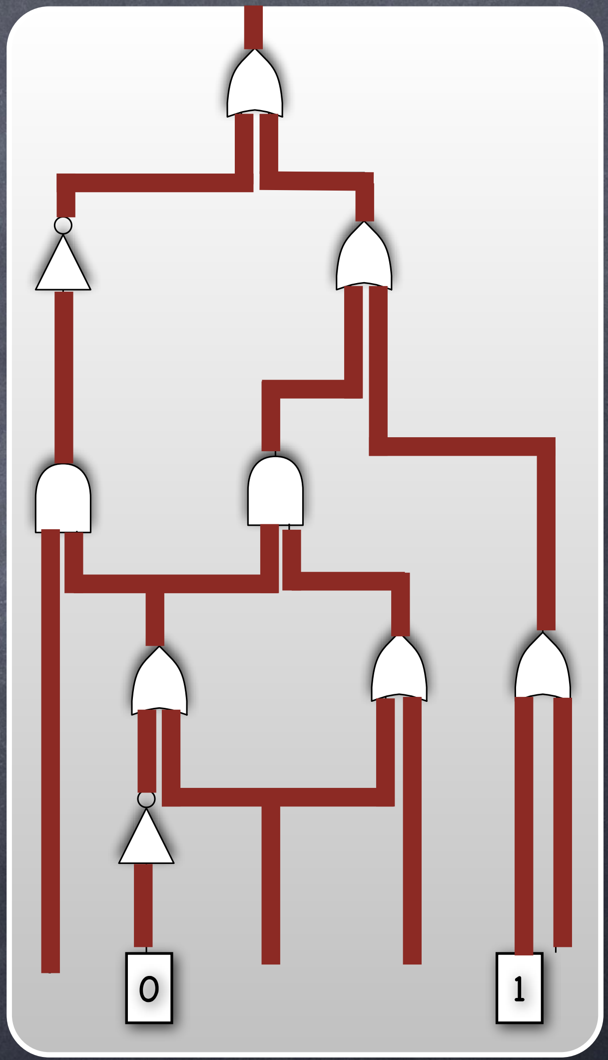
- $f((x_1,\ldots,x_N); i) = (\perp; x_i)$
- For passive security: simply run N copies of 1-out-of-2 OT, with inputs for j^{th} instance being $(0,x_j; b_j)$ where $b_j = 1$ iff $j=i$
- Aside: active security easily achievable too using a randomized protocol using N-1 copies of 1-out-of-2 OT

Doing MPC

Basic GMW

Functions as Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of
- Arithmetic circuits: Wire values from a field and the gates are addition/multiplication



Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)
- e.g.: X > Y for two bit inputs $X=x_1x_0$, $Y=y_1y_0$: $(x_1 \wedge \neg y_1) \vee (\neg(x_1 \oplus y_1) \wedge (x_0 \wedge \neg y_0))$
- Can directly convert a truth-table into a circuit, but circuit size exponential in input size
- Can convert any ("efficient") program into a ("small") circuit
- Interesting problems already given as succinct programs/circuits

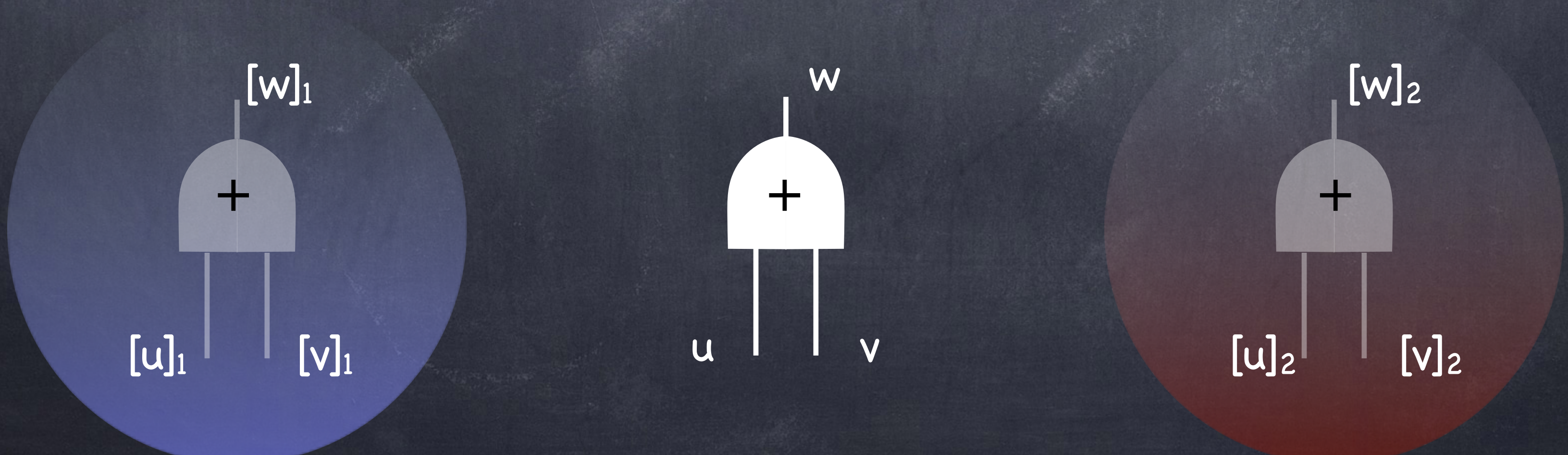
	00	01	10	11
00	0	0	0	0
01	1	0	0	0
10	1	1	0	0
11	1	1	1	0

Basic GMW

- Adapted from the famous Goldreich-Micali-Wigderson (1987) protocol (due to Goldreich-Vainish, Haber-Micali,...)
- Efficient passive secure MPC based on OT, without any other computational assumptions
  - Extends to arithmetic circuits, using "OLE" instead of OT
- Idea: Computing on **additively secret-shared values**
  - Will write $[s]_i$ to denote shares of s, so that $s = [s]_1 + \cdots + [s]_m$ for m-way sharing
    - Start with m=2

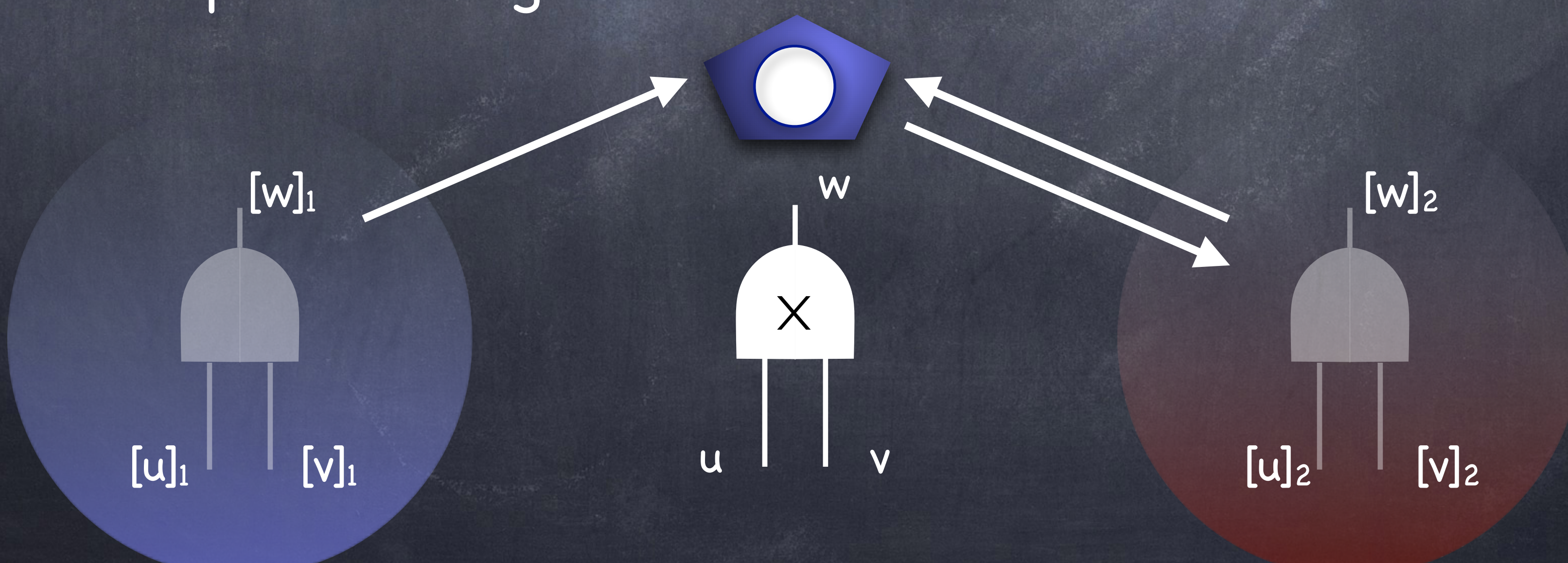
Computing on Shares

- Let gates be + & × over any field
  - XOR & AND for Boolean circuits (field GF(2))
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.
- $w = u + v$: Each one locally computes $[w]_i = [u]_i + [v]_i$



Computing on Shares

- What about $w = u \times v$?
  - $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive (proof-of-concept) protocol
    - Note: Bob's input is $([u]_2,[v]_2)$. Over the binary field, this requires a single 1-out-of-4 OT.



GMW: many parties

- m-way sharing: $s = [s]_1 + \ldots + [s]_m$

Allows security against arbitrary number of corruptions

- Addition, local as before
- Multiplication: For $w = u \times v$
$[w]_1 + .. + [w]_m = ([u]_1 + .. + [u]_m) \times ([v]_1 + .. + [v]_m)$
  - Party i computes $[u]_i[v]_i$
  - For every pair (i,j), $i \neq j$, Party i picks random a_{ij} and lets Party j securely compute b_{ij} s.t. $a_{ij} + b_{ij} = [u]_i[v]_j$ using the naive protocol (a single 1-out-of-2 OT)
  - Party i sets $[w]_i = [u]_i[v]_i + \Sigma_j (a_{ij} + b_{ji})$

Levels of Security

	Unlimited Corruption	Honest Majority		
Passive	"GMW" protocol (given OT)			
Active	Unfair		Fair	Full
	?			

Levels of Security

- Unfair
  - Adversary can cause honest parties to abort (not receive output) but get its own output
  - In fact, it can decide which honest parties abort after seeing its own output
- Fair
  - Adversary can cause everyone to abort, but then it will not see its own output
- Full
  - Guaranteed Output Delivery
- Fairness and Guaranteed Output Delivery possible in general only with honest majority

GMW against active corruption

- Original GMW approach: Use Zero Knowledge proofs to force the parties to run the protocol honestly
  - Needs (passive-secure) OT to be implemented using a protocol
- Kilian/IPS: Direct information-theoretic reduction to OT
- Alternate construction: information-theoretic reduction to OT, starting from passive-secure GMW

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Computing shares a_{12}, b_{12} s.t. $a_{12} + b_{12} = [u]_1 \cdot [v]_2$:
  - Alice picks a_{12} and sends $(-a_{12}, [u]_1 - a_{12})$ to OT.
    Bob sends $[v]_2$ to OT.
  - What if Alice sends arbitrary (x,y) to OT? Effectively, setting $a_{12} = -x$, $[u]_1' = y-x$.
  - And what Bob sends to OT is $[v]_2'$
- i.e., arbitrary behaviour of Alice & Bob while sharing $[u]_1 \cdot [v]_2$ correspond to them locally changing their shares $[u]_1$ and $[v]_2$

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Arbitrary behaviour of Alice while sharing $[u]_1 \cdot [v]_2$ and $[u]_2 \cdot [v]_1$ corresponds to her locally changing her shares of u and v
  - Alice changing her share from $[u]_1$ to $[u]_1'$ is effectively changing u to $u+\Delta_u$, where $\Delta_u = [u]_1' - [u]_1$ depends only on her own view
- Over all effect: a corrupt party can arbitrarily add Δ_u and Δ_v to wires u and v before multiplication
- Also, can add deltas to all input and output wires

Active-Secure Variant of Basic GMW

- Any active attack on Basic GMW protocol corresponds to an additive attack on the wires of the circuit
- Idea: "Compile" the circuit such that any additive attack amounts to error (w.h.p.), resulting in random output
- Additive Manipulation Detecting (AMD) circuits
  - Extension of AMD codes
    - e.g. encode x as a vector (x, r, xr) where r is random from a large field. Additive attacks (without knowing r) detected unless $(x+\delta_1)(r+\delta_2) = (xr+\delta_3)$: i.e., $\delta_1 \cdot r + x \cdot \delta_2 + \delta_1 \cdot \delta_2 = \delta_3$. Unlikely unless $\delta_1 = 0$.

Levels of Security

	Unlimited Corruption	Honest Majority	
Passive	"GMW" protocol (given OT)	"BGW" protocol (no setup/computational hardness)	
Active	Unfair	Fair	Full
	via AMD circuits		

Doing MPC

Basic BGW

BGW

- Protocol by Ben-Or, Goldwasser, Wigderson. We will first look at the simpler setting of passive corruption.
  - Passive secure MPC for arithmetic circuits (over large enough fields) assuming honest majority, but without any computational assumptions or setup
- Idea: Computing on **secret-shared values**
  - **Shamir secret-sharing:** threshold, linear secret-sharing, also allowing multiplication

Threshold Secret-Sharing

- (n,t)-secret-sharing
  - Divide a message m into n shares $s_1,\ldots,s_n$, such that
    - any t shares are enough to reconstruct the secret
    - up to t-1 shares have no information about the secret
- Additive secret-sharing is (n,n) secret-sharing

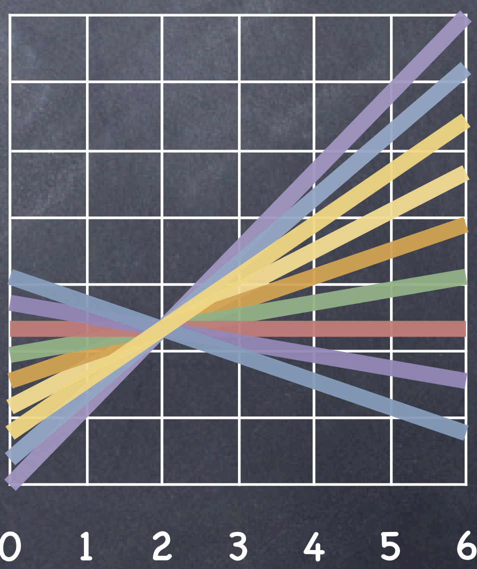
e.g., $(s_1,\ldots,s_{t-1})$ has the same distribution for every m in the message space

Threshold Secret-Sharing

- First, (n,2) secret-sharing
- Message-space = share-space = F, a field (e.g. integers mod a prime)
  - Share(m): pick random r. Let $s_i = r \cdot a_i + m$ (for $i=1,\ldots,n < |F|$)
  - Reconstruct(s_i, s_j): $r = (s_i - s_j)/(a_i - a_j)$; $m = s_i - r \cdot a_i$
  - Each s_i by itself is uniformly distributed, irrespective of m [Why?]
  - "Geometric" interpretation
    - Sharing picks a random "line" $y = f(x)$, such that $f(0) = m$. Shares $s_i = f(a_i)$.
    - s_i is independent of m: exactly one line passing through (a_i,s_i) and $(0,m')$ for any secret m'
    - But can reconstruct the line from two points!

a_i are n distinct, non-zero field elements

Since a_i^{-1} exists, exactly one solution for $r \cdot a_i + m = d$, for every value of d



Threshold Secret-Sharing

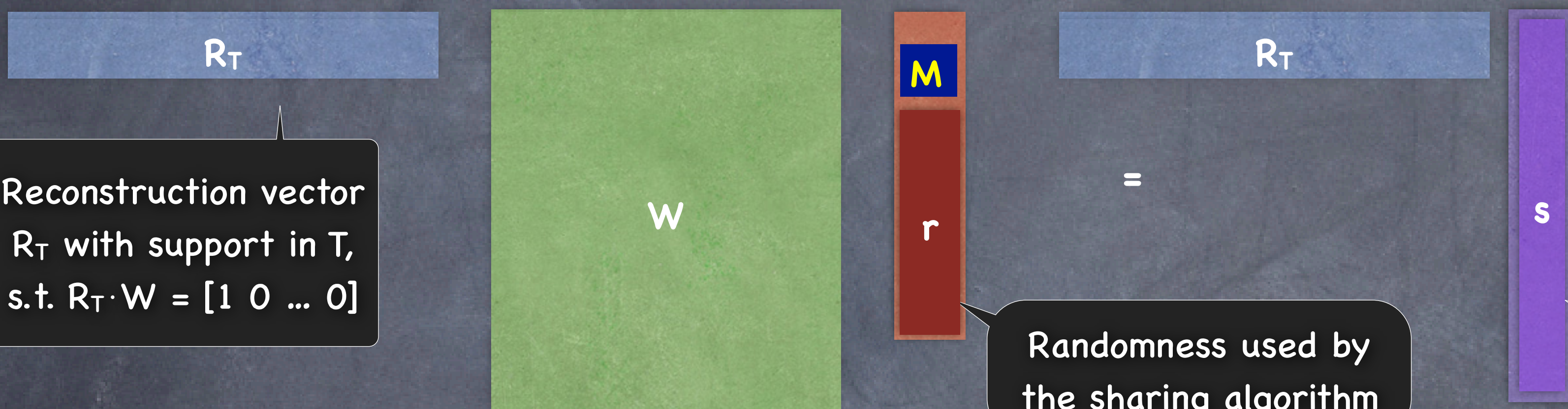
Shamir Secret-Sharing

- (n,t) secret-sharing in a field F
- Generalizing the geometric/algebraic view: instead of lines, use polynomials
  - Share(m): Pick a random degree t-1 polynomial f(X), such that f(0) = m. Shares are $s_i = f(a_i)$.
    - Random polynomial with f(0) = m: $c_0 + c_1X + c_2X^2 + \ldots + c_{t-1}X^{t-1}$ by picking $c_0 = m$ and $c_1,\ldots,c_{t-1}$ at random.
  - Reconstruct($s_1,\ldots,s_t$): Lagrange interpolation to find $m = c_0$
    - Need t points to reconstruct the polynomial. Given t-1 points, out of $|F|^{t-1}$ polynomials passing through (0,m') (for any m') there is exactly one that passes through the t-1 points
- Is a "Linear Secret-Sharing Scheme"

Linear Secret-Sharing

- Another look at additive secret-sharing

Working with a commutative group here. Multiplication by ±1 and 0 well-defined in a group. But more broadly, we shall consider a **field**.



Reconstruction vector R_T with support in T, s.t. $R_T \cdot W = [1\ 0\ \ldots\ 0]$

Randomness used by the sharing algorithm

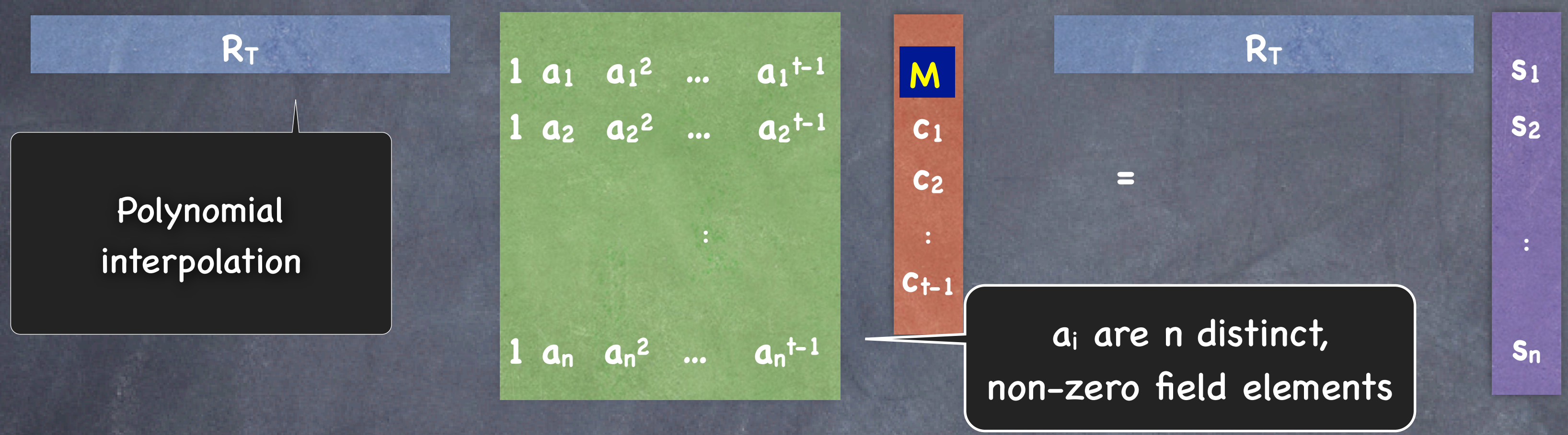
Each coordinate is a separate share

More generally, a share can have multiple coordinates

- Linear Secret-Sharing over a field: message and shares are field elements
- Reconstruction by a set $T \subseteq [n]$: <u>solve</u> the message from given shares
  - i.e., solve $W_T \begin{bmatrix} M \\ r \end{bmatrix} = s_T$ for M

Linearity of Shamir Secret-Sharing

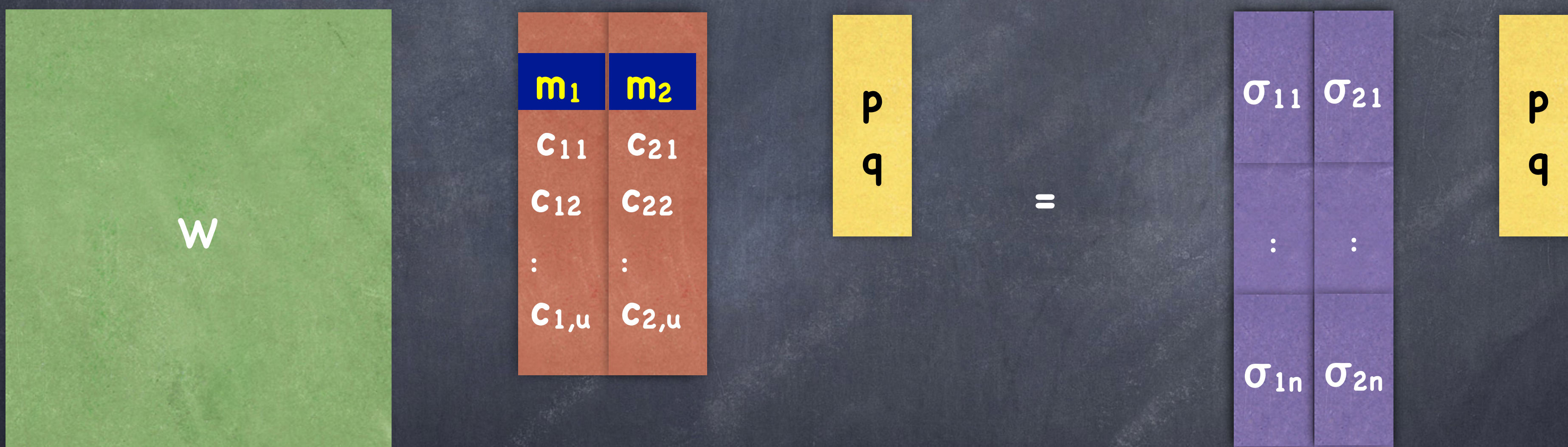
- Shamir's scheme is a linear secret-sharing scheme



- Which sets $T \subseteq [n]$ can reconstruct? i.e., T s.t. W_T spans [1 0 ... 0]?
  - W_T spans [1 0 ... 0] iff $|T| \geq t$
    - For $|T|=t$, W_T is a Vandermonde matrix, and is a basis for $\mathbb{F}^t$
    - For $|T| < t$, can add a row [1 0 ... 0] and (optionally) more rows of the form $[1 \; a \; a^2 \dots a^t]$ to get a Vandermonde matrix. So [1 0 ... 0] is independent of the rows of W_T
- Secrecy: guaranteed for any linear secret-sharing scheme

Linear Secret-Sharing: Computing on Shares

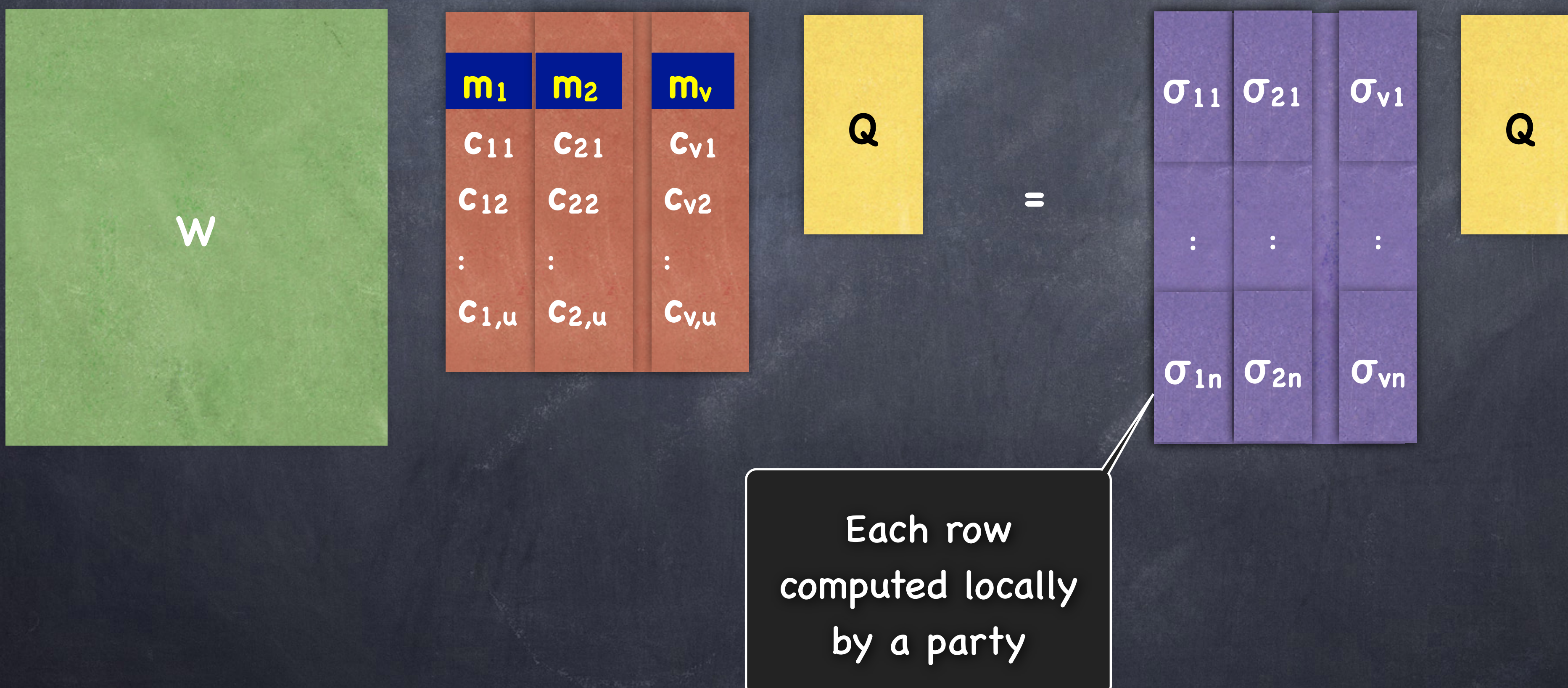
- Suppose two secrets m_1 and m_2 shared using the same secret-sharing scheme



- Then for any $p,q \in F$, shares of $p \cdot m_1 + q \cdot m_2$ can be computed <u>locally</u> by each party i as $\sigma_i = p \cdot \sigma_{1i} + q \cdot \sigma_{2i}$

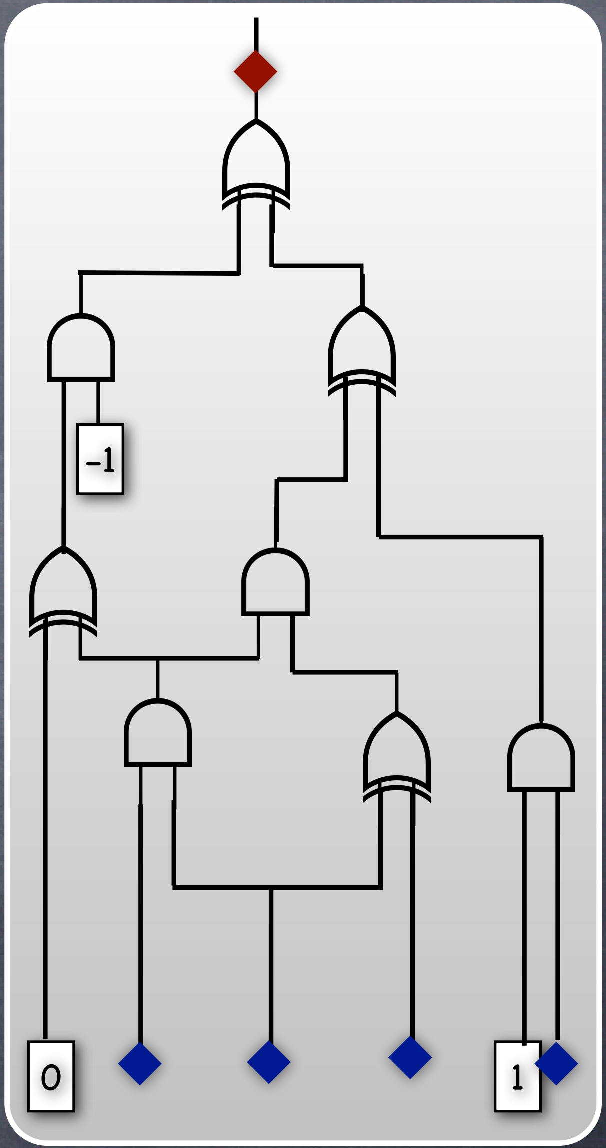
Linear Secret-Sharing: Computing on Shares

- More generally, can compute shares of any linear transformation



BGW

- Wire values will be kept linearly secret-shared among all servers
- Each input value is secret-shared among the servers by the input client "owning" the input gate
- Linear operations computed by each server on its shares, locally (no communication)
  - Shares of x, y → Shares of ax+by
- Multiplication will involve communication
  - Coming up
- Output gate evaluation: servers send their shares to the output client owning the gate



Passive-Secure BGW

- Question: How to go from shares(x), shares(y) to shares(x·y) securely?
- Idea 1: Use multiplicative structure of Shamir secret-sharing
  - For polynomials, multiplication commutes with evaluation: $(f\cdot g)(x) = f(x)\cdot g(x)$
  - In particular, to get a polynomial h with $h(0)= f(0)\cdot g(0)$, simply define $h = f\cdot g$. Shares $h(x)$ can be computed as $f(x)\cdot g(x)$
  - But note: h has a higher degree!
    - Problem 1: If original degree $\geq N/2$, can't reconstruct the product even if all servers reveal their new shares
      - Solution: Use degree $d < N/2$ (limits to $d < N/2$ corruption)
    - Problem 2: Can't continue protocol after one multiplication

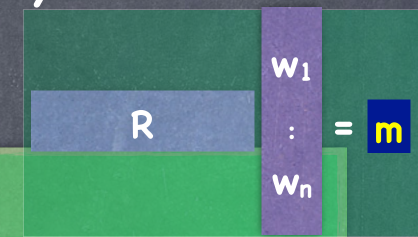
Passive-Secure BGW

- Problem: If x, y shared using a degree d polynomial, x·y is shared using a degree 2d polynomial
- Solution: Bring it back to the original secret-sharing scheme!
  - Share switching (coming up)
- Note: All N servers together should be able to linearly reconstruct the degree-2d sharing
  - Start with N > 2d
  - Can tolerate only up to d (< N/2) corrupt servers (and any number of corrupt clients)

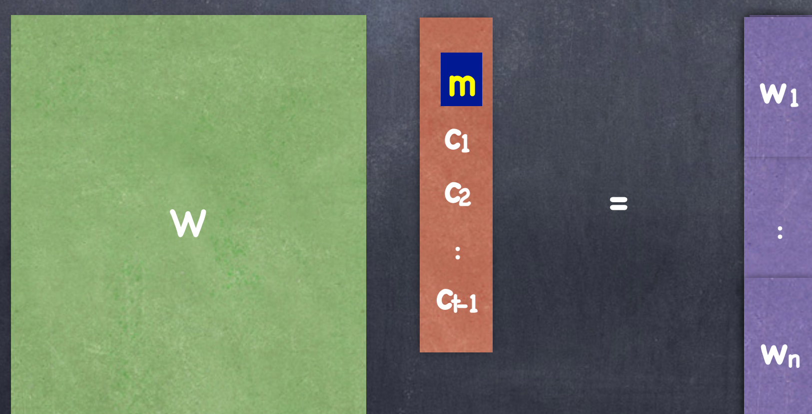
≤ (N-1)/2

Switching Schemes

- Can move from any linear secret-sharing scheme W to any other linear secret-sharing scheme Z "securely"

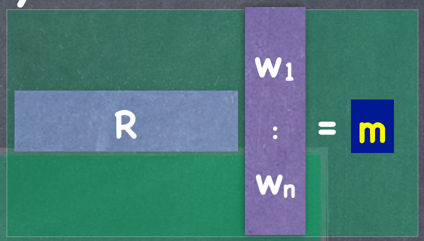


- Given shares $(w_1, \ldots, w_n) \leftarrow$ W.Share(m)
- Share each w_i using scheme Z: $(\sigma_{i1},\ldots,\sigma_{in}) \leftarrow$ Z.Share(w_i)
- Locally each party j reconstructs using scheme W: $z_j \leftarrow$ W.Recon $(\sigma_{1j},\ldots,\sigma_{nj})$

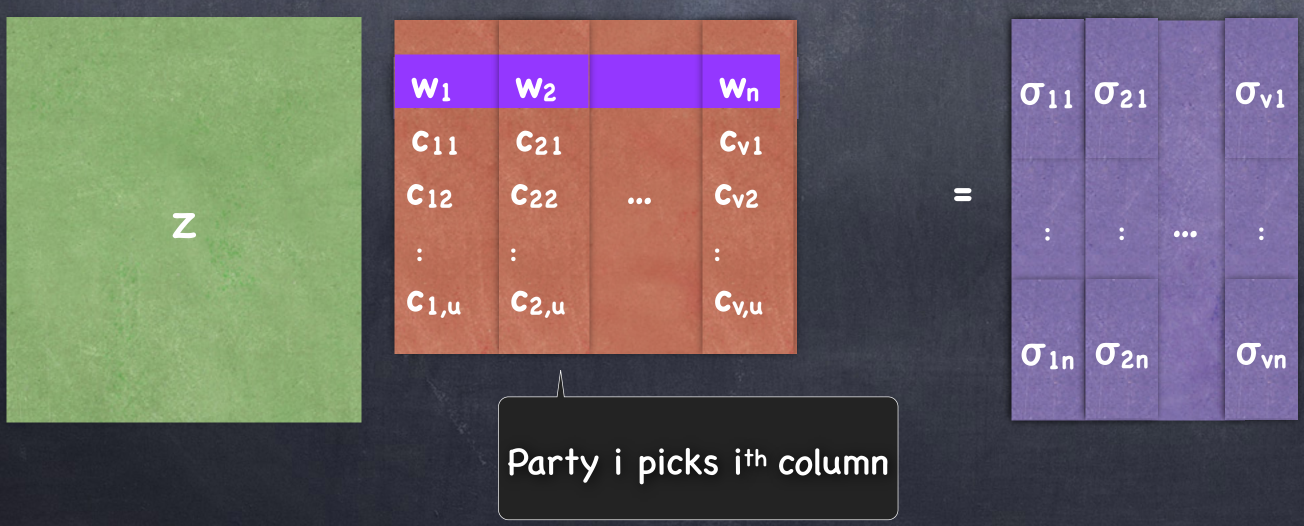


Switching Schemes

- Can move from any linear secret-sharing scheme W to any other linear secret-sharing scheme Z "securely"

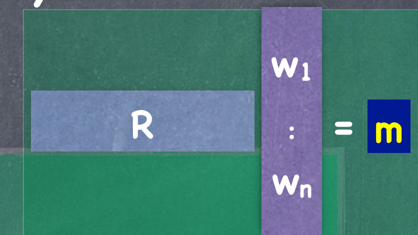


- Given shares $(w_1, \ldots, w_n) \leftarrow$ W.Share(m)
- Share each w_i using scheme Z: $(\sigma_{i1},\ldots,\sigma_{in}) \leftarrow$ Z.Share(w_i)
- Locally each party j reconstructs using scheme W: $z_j \leftarrow$ W.Recon $(\sigma_{1j},\ldots,\sigma_{nj})$

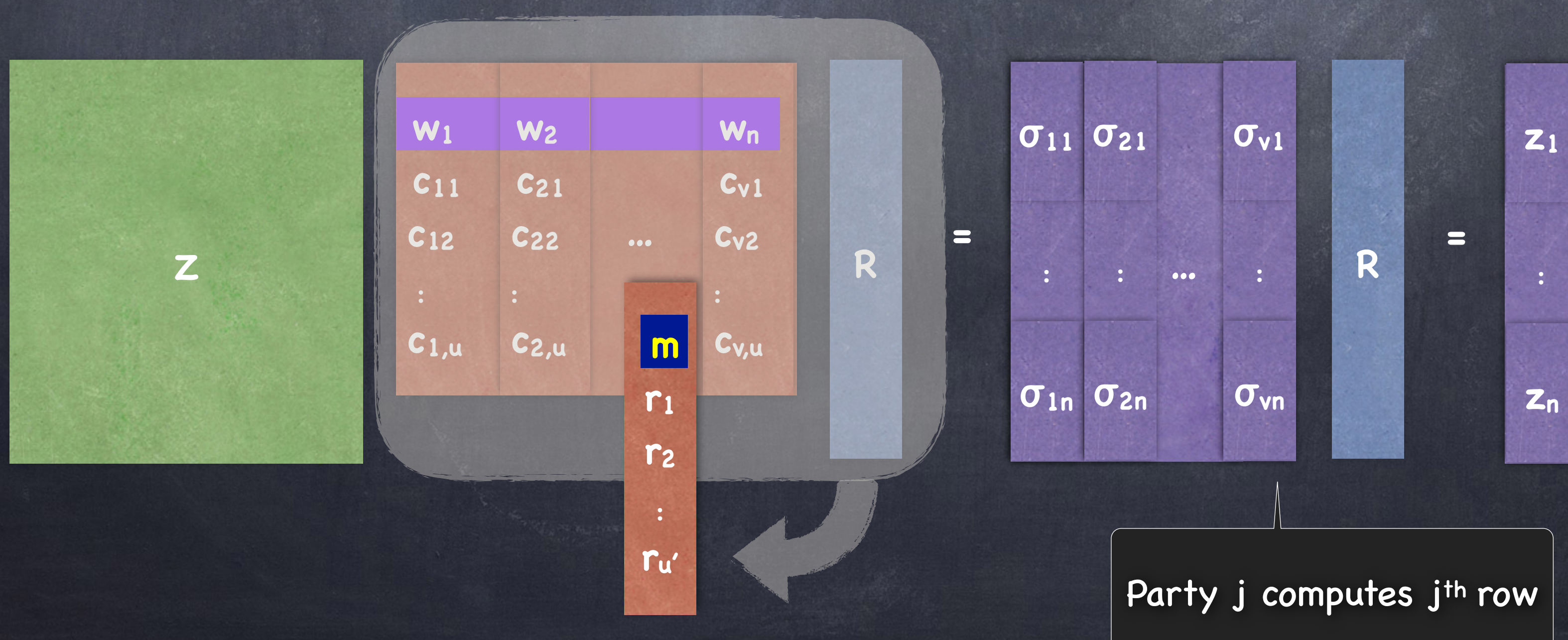


Switching Schemes

- Can move from any linear secret-sharing scheme W to any other linear secret-sharing scheme Z "securely"

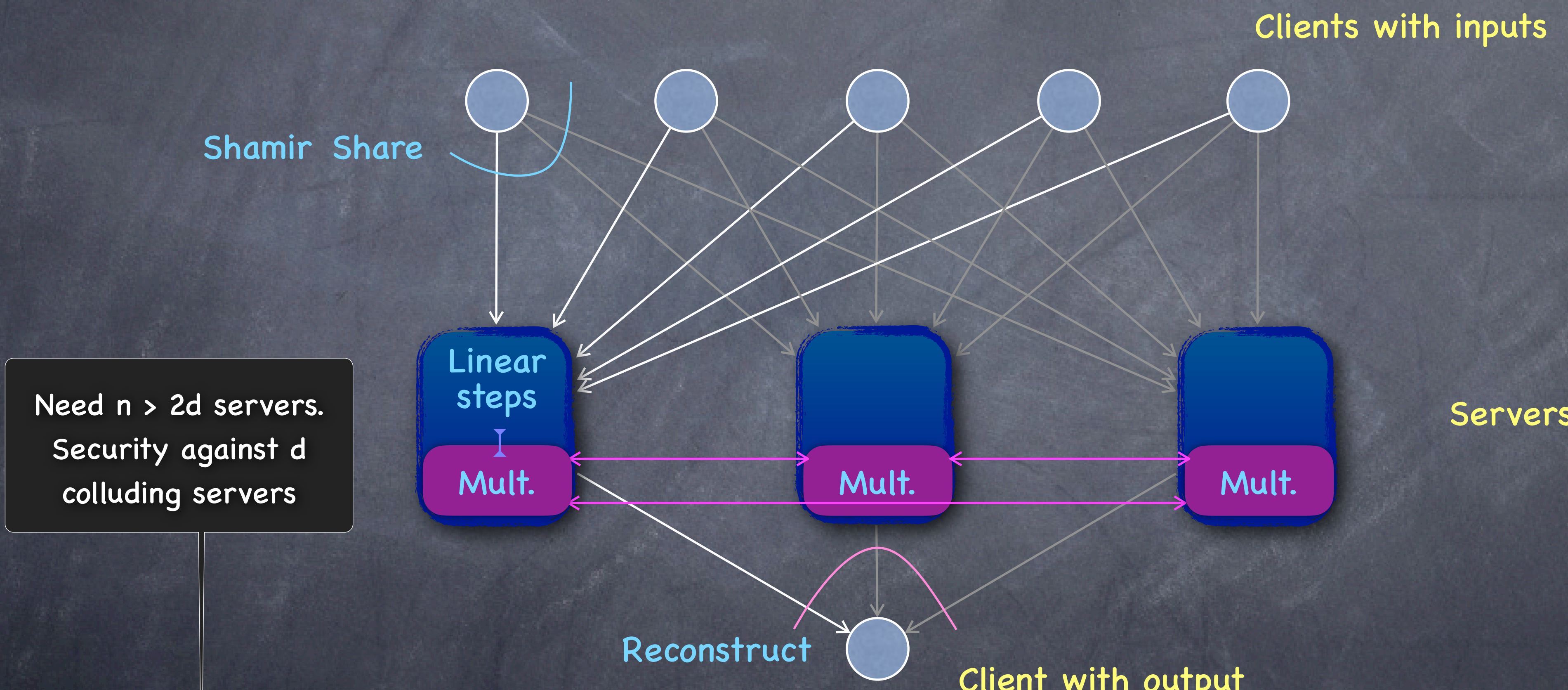


- Given shares $(w_1, \ldots, w_n) \leftarrow$ W.Share(m)
- Share each w_i using scheme Z: $(\sigma_{i1},\ldots,\sigma_{in}) \leftarrow$ Z.Share(w_i)
- Locally each party j reconstructs using scheme W:
  $z_j \leftarrow$ W.Recon $(\sigma_{1j},\ldots,\sigma_{nj})$



Passive-Secure BGW: Summary

- A function f given as a program with linear steps and multiplications: arithmetic circuit (over a finite field)



- Locally multiplying degree d shares of M_1 and M_2 gives a degree 2d share of $M_1 \cdot M_2$. Then switch back to a fresh degree d sharing (involves communicating degree d shares of degree 2d shares)

Passive-Secure BGW: Security

- First consider the protocol till just before output reconstruction
- We want that the adversary learns nothing about the honest parties' inputs
    - The only messages received are from <u>fresh</u> degree d secret sharings (even in the multiplication step), even though the messages being shared are not uniform
    - To the adversary, this appears as uniform random shares

Passive-Secure BGW: Security

- First consider the protocol till just before output reconstruction
  - Adversary learns nothing about the honest parties' inputs
- Now consider the output reconstruction step as well
- Observation: Enough to show security against an adversary who actually corrupts the maximum allowed number of servers, d
  - Consider the messages received by the adversary for each output wire it owns
  - Fully determined by the d shares it already has and the output value (which it is allowed to learn)
  - So entire view determined by own inputs, the random values from the computation phase, and own outputs

Levels of Security

	Unlimited Corruption	Honest Majority		
Passive	"GMW" protocol (given OT)	"BGW" protocol (no setup/computational hardness)		
Active	Unfair		Fair	Full
	via AMD circuits			

Fair Honest-Majority MPC

- Tool: Error-Correcting Secret-Sharing (ECSS)
  - a.k.a. robust secret-sharing
  - Allows reconstruction as long as a majority of the shares submitted are correct
    - e.g., Mutually authenticating shares (using statistical MACs). To reconstruct, look for a clique of size n/2 of mutually consistent shares.

Fair Honest-Majority MPC

- Share inputs using ECSS
- Run unfair protocol to obtain ECSS shares of output
- If no abort, each honest party broadcasts OK
- If all say OK, then send ECSS shares for reconstruction
- Adversary can cause abort for all parties, but without knowing its own outputs. Cannot change output by corrupting < n/2 parties.
  - Note: requires broadcast to be fully secure (guaranteed output delivery). Possible to implement when < n/3 corrupt parties

Full Security

- The main difficulty, compared to active-secure MPC, is in identifying who cheated
- Not possible to exactly identify one cheating party
  - e.g., [P_1 sends garbage to P_2 over a private link] $\equiv$ [P_2 discards what P_1 sent, replacing it with garbage]
- Can hope to identify a set of 2 parties, at least one of which is corrupt
- $id_{1/2}$-abort-security: Either all honest parties get output, or they agree on a set of parties, at least half of which are corrupt

Full Security

- Assume we have $id_{1/2}$-abort-secure protocol for general functions
  - Requires additional techniques, involving consistency checks and complaining if the checks fail (omitted)
- ECSS share inputs
- Run a $id_{1/2}$-abort-secure protocol to obtain ECSS shares of outputs
  - If abort/error, eliminate the identified set (who reshare their inputs among active players). Repeat.
- If no abort, send shares for reconstruction
- Note: honest majority maintained among active parties

Levels of Security

	Unlimited Corruption	Honest Majority		
Passive	"GMW" protocol (given OT)	"BGW" protocol (no setup/computational hardness)		
Active	Unfair		Fair	Full
	via AMD circuits		via error-correcting secret-sharing	via identification & elimination

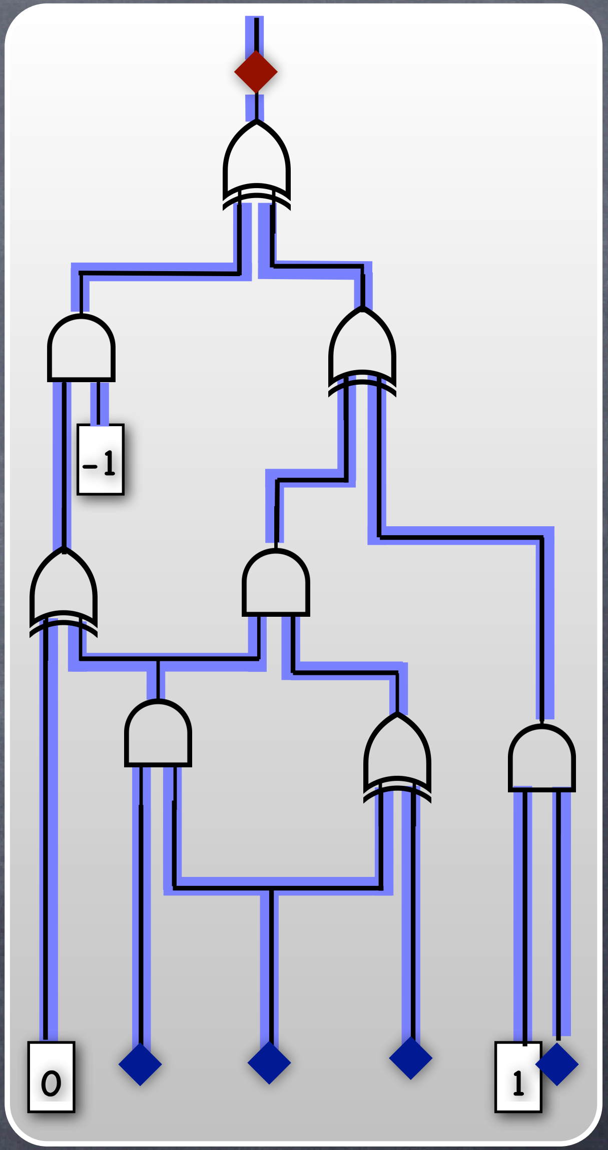
Doing MPC

Yao's Garbled Circuit

Recall

Functions as Circuits

- Directed acyclic graph
  - Nodes: multiplication and addition gates, constant gates, inputs, output(s)
  - Edges: wires carrying values from F
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of



2-Party MPC for General Circuits

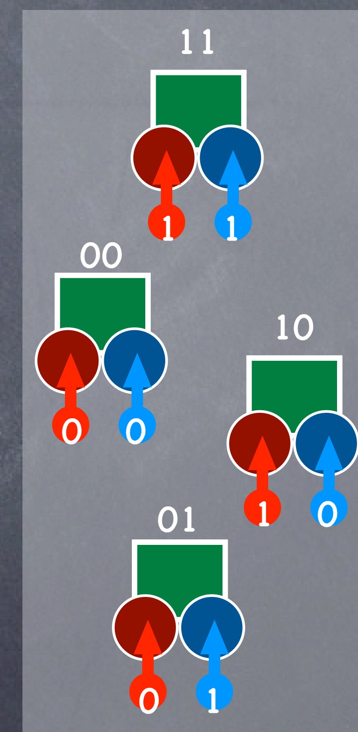
	0	1
0	0	1
1	1	1

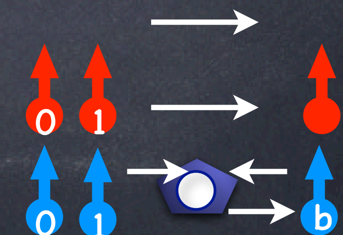
- "General": evaluate any arbitrary (boolean) circuit
  - One-sided output: both parties give inputs, only one party gets outputs
  - Either party maybe corrupted passively
- Consider evaluating OR (single gate circuit)
  - Alice holds x=a, Bob has y=b; Bob should get OR(x,y)

A Physical Protocol

- Alice prepares 4 boxes B_{xy} corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$
- Inside $B_{xy=ab}$ she places the bit OR(a,b) and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box)
- She un-labels the four boxes and sends them in random order to Bob. Also sends the key $K_{x=a}$ (labeled only as K_x).
  - So far Bob gets no information
- Bob "obliviously picks up" $K_{y=b}$, and tries the two keys K_x,K_y on the four boxes. For one box both locks open and he gets the output.

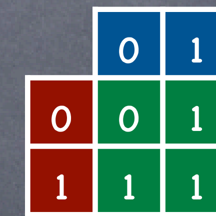
	0	1
0	0	1
1	1	1

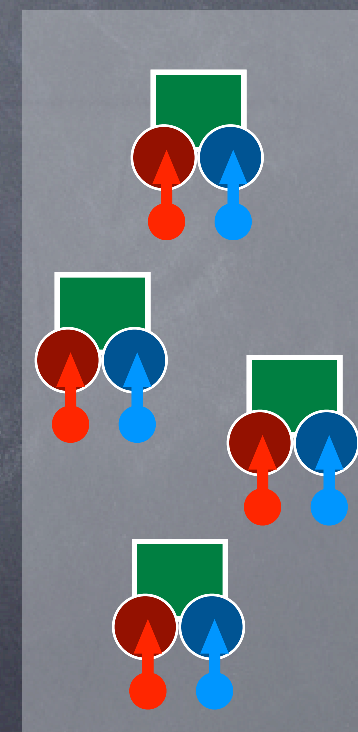


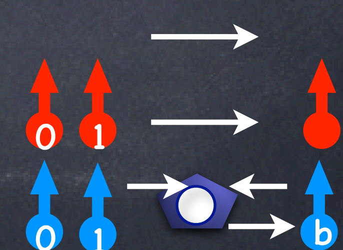


A Physical Protocol

- Secure?
- For curious Alice: only influence from Bob is when he picks up his key $K_{y=b}$
  - But this is done "obliviously", so she learns nothing
- For curious Bob: What he sees is predictable (i.e., can be simulated), given the final outcome
  - What Bob sees: His key opens K_y in two boxes, Alice's opens K_x in two boxes; only one random box fully opens. It has the outcome.
  - Note when y=1, cases x=0 and x=1 appear same

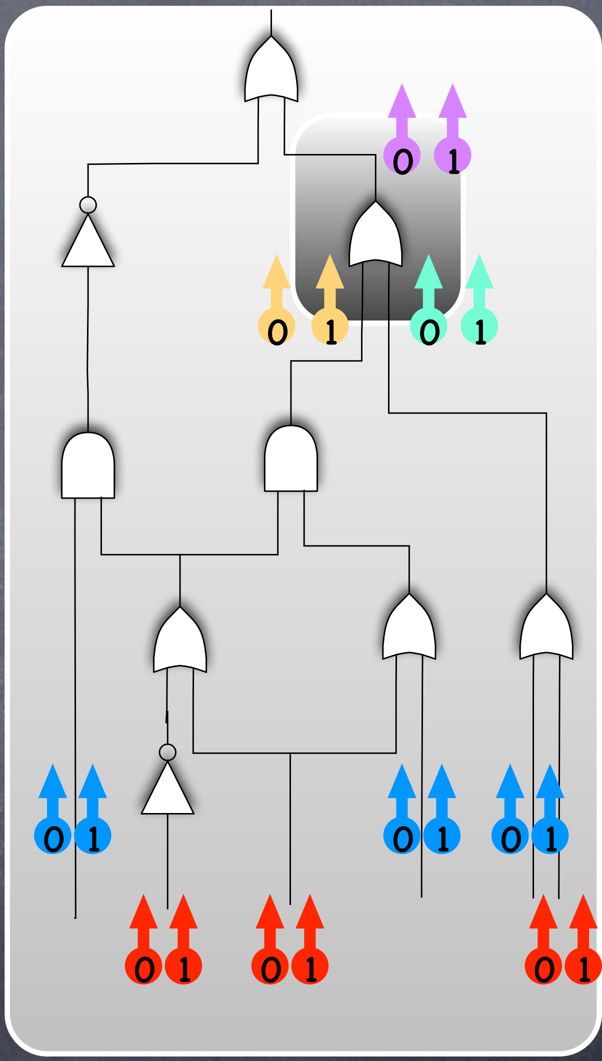






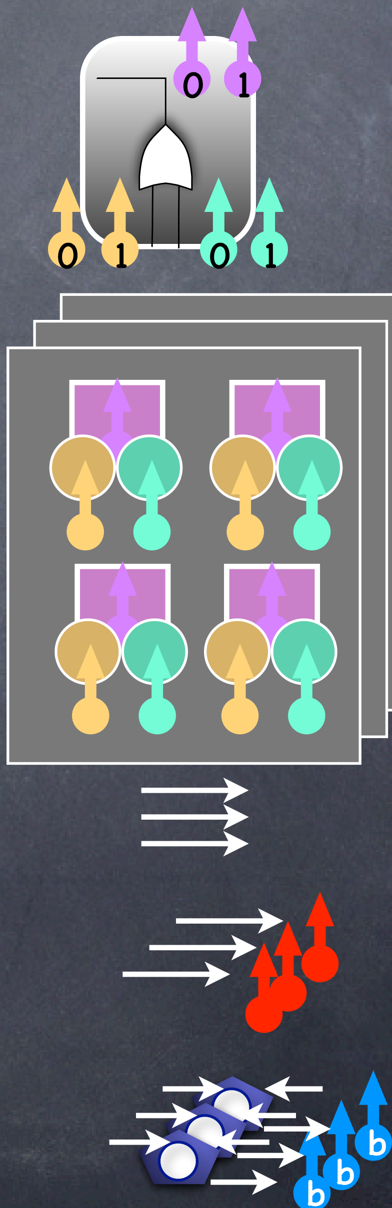
Larger Circuits

- Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate
  - For each wire w in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$



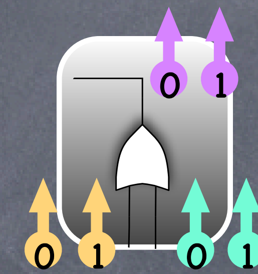
Larger Circuits

- Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate
  - For each wire w in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$
  - For each gate G with input wires (u,v) and output wire w, prepare 4 boxes B_{uv} and place $K_{w=G(a,b)}$ inside box $B_{uv=ab}$. Lock $B_{uv=ab}$ with keys $K_{u=a}$ and $K_{v=b}$
- Give to Bob: Boxes for each gate, one key for each of Alice's input wires
  - Obliviously: one key for each of Bob's input wires
- Boxes for output gates have values instead of keys

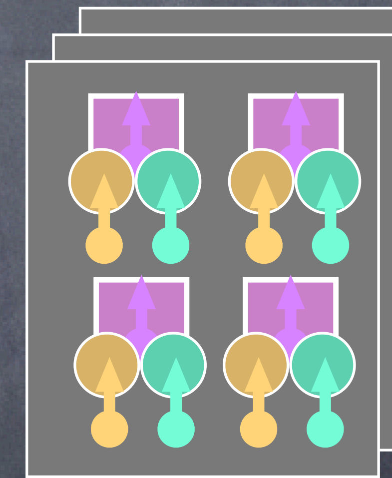


Larger Circuits

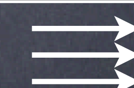
- Evaluation: Bob gets one key for each input wire of a gate, opens one box for the gate, gets one key for the output wire, and proceeds

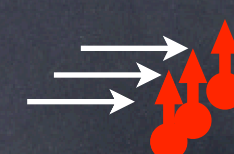


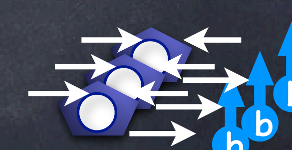
  - Gets output from a box for the output gate



- Security similar to before
  - Curious Alice sees nothing
  - Bob can simulate his view given final output: Bob could prepare boxes and keys (stuffing unopenable boxes arbitrarily); for an output gate, place the output bit in the box that opens





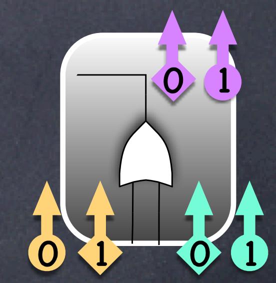


Garbled Circuit

- That was too physical!
- Yao's Garbled circuit: boxes/keys replaced by Symmetric Key Encryption (specifically, using a Pseudorandom Function or PRF)
  - $Enc_K(m) = PRF_K(index) \oplus m$, where index is a wire index (distinct for different wires fanning-out of the same gate)
  - Double lock: $Enc_{Kx}(Enc_{Ky}(m))$
  - PRF in practice: a block-cipher, like AES
- Uses Oblivious Transfer for strings: For passive security, can just repeat bit-OT several times to transfer longer keys

Garbled Circuit

- One issue when using encryption instead of locks
  - Given four doubly locked boxes (in random order) and two keys, we simply tried opening all locks until one box fully opened
  - With encryption, cannot quite tell if a box opened or not! Outcome of decryption looks random in either case.
  - Simple solution: encode the keys so that wrong decryption does not result in outputs that look like valid encoding of keys
  - Better solution: For each wire, the 0 & 1 keys have distinct "shape" labels, assigned at random. Each locked box marked with the shape of the two keys needed to unlock it.



Defining MPC

A simple example

- Recall the Dutch flower auction protocol
  - Count down from 100
  - At each even round Alice announces whether her bid equals the current count; at each odd round Bob does the same
  - Stop if a party says yes
- Perfectly secure against active adversary as well
  - But is that ideal enough?

Attack on Dutch Flower Auction

- Alice and Bob are taking part in two auctions
- Alice's goal: ensure that Bob wins at least one auction with some bid z, and the winning bid in the other auction $\in \{z, z-1\}$
- Easy in the protocol: run the two protocols lockstep. Wait till Bob says yes in one. Done if Bob says yes in the other simultaneously. Else Alice will say yes in the next round.
- Why is this an attack?
  - Impossible for Alice to ensure this in IDEAL!

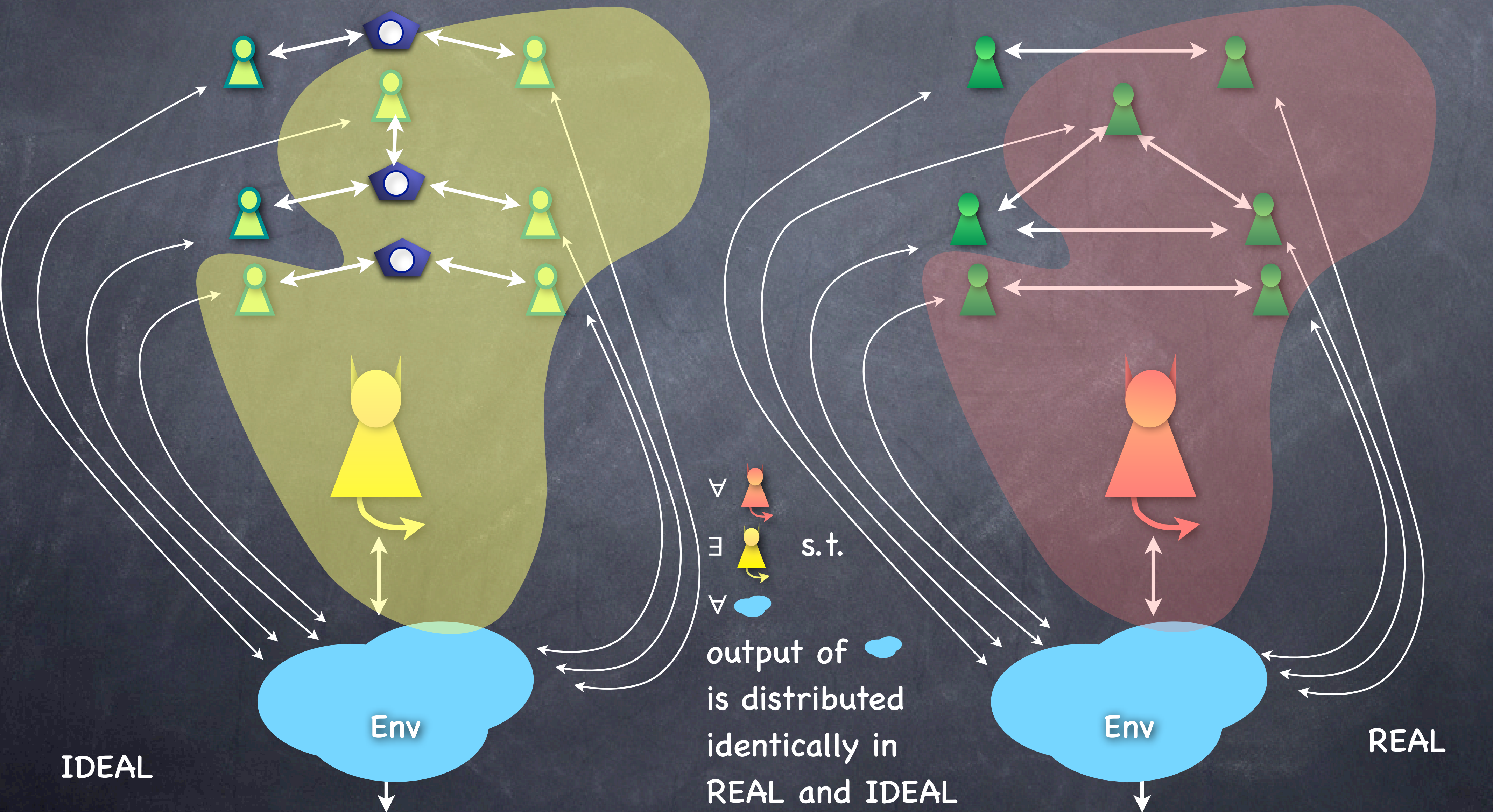
Attack on Dutch Flower Auction

- Alice's goal: ensure that Bob wins at least one auction with some bid z, and the winning bid in the other auction $\in \{z, z-1\}$
- Impossible to ensure this in IDEAL!
- Alice can get a result in one session, before running the other. But what should she submit as her input x in the first one?
  - Trouble if $x \neq 0$, because she could win (i.e., $z-1=x$) and Bob's input in the other session may be $\neq x+1$
  - Trouble if $x=0$, because Bob could win with input 1 (i.e., $z=1$) and in the other session his input > 1

Composition Issues

- Standalone security definition does not ensure security when composed
- Different modes of composition
  - Sequential composition: protocols executed one after the other. Adversary communicates with the environment between executions.
  - Concurrent composition: multiple sessions (typically of the same protocol) are active at the same time, and the adversary can coordinate its actions across the sessions

Concurrent Executions

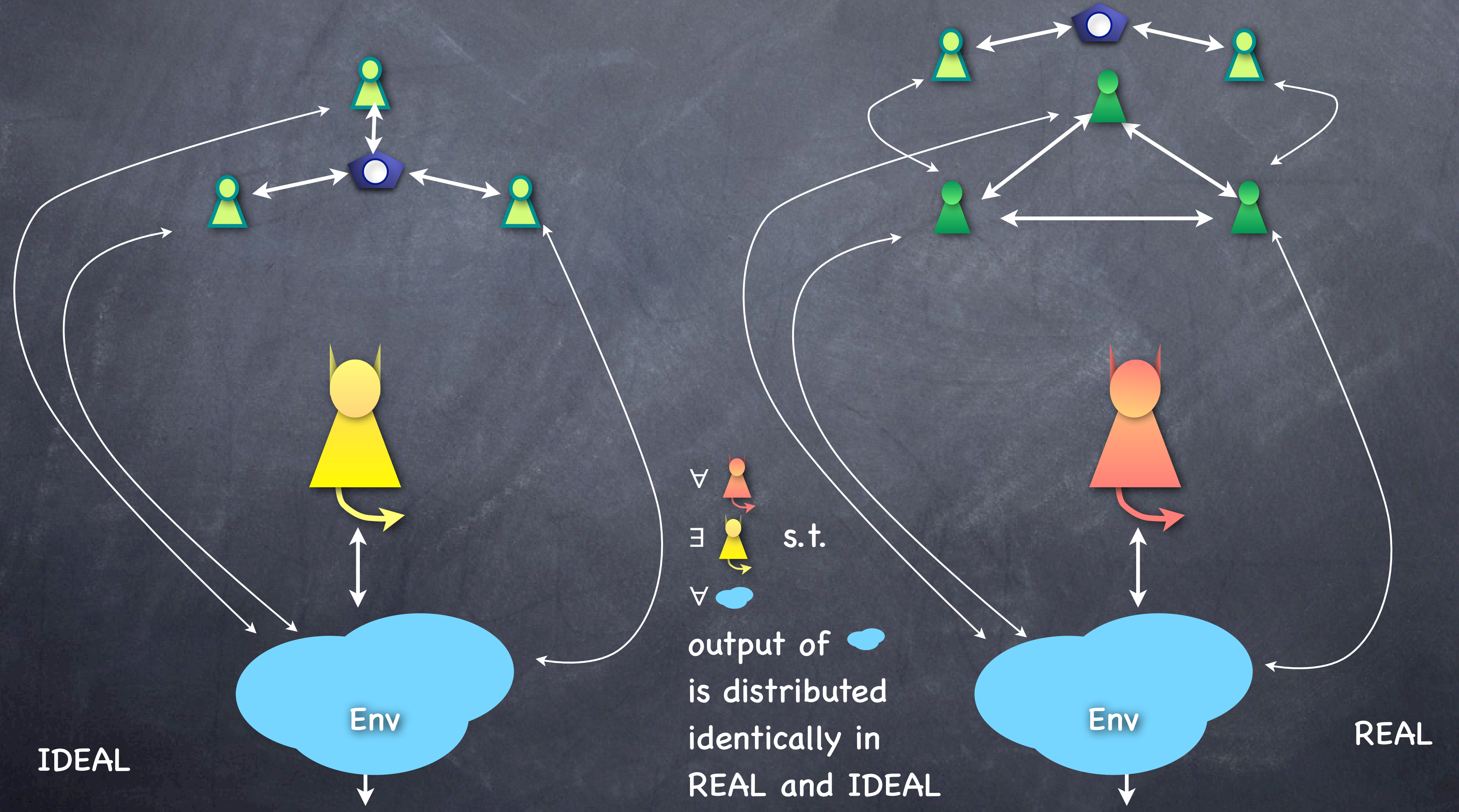


Composition Issues

- Standalone security definition does not ensure security when composed
- Different modes of composition
  - Sequential composition: protocols executed one after the other. Adversary communicates with the environment between executions.
  - Concurrent composition: multiple sessions (typically of the same protocol) are active at the same time, and the adversary can coordinate its actions across the sessions
    - Also, subroutine calls

Subroutines

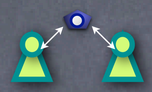
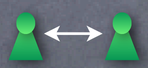
A "REAL" protocol in which parties access (another) IDEAL protocol

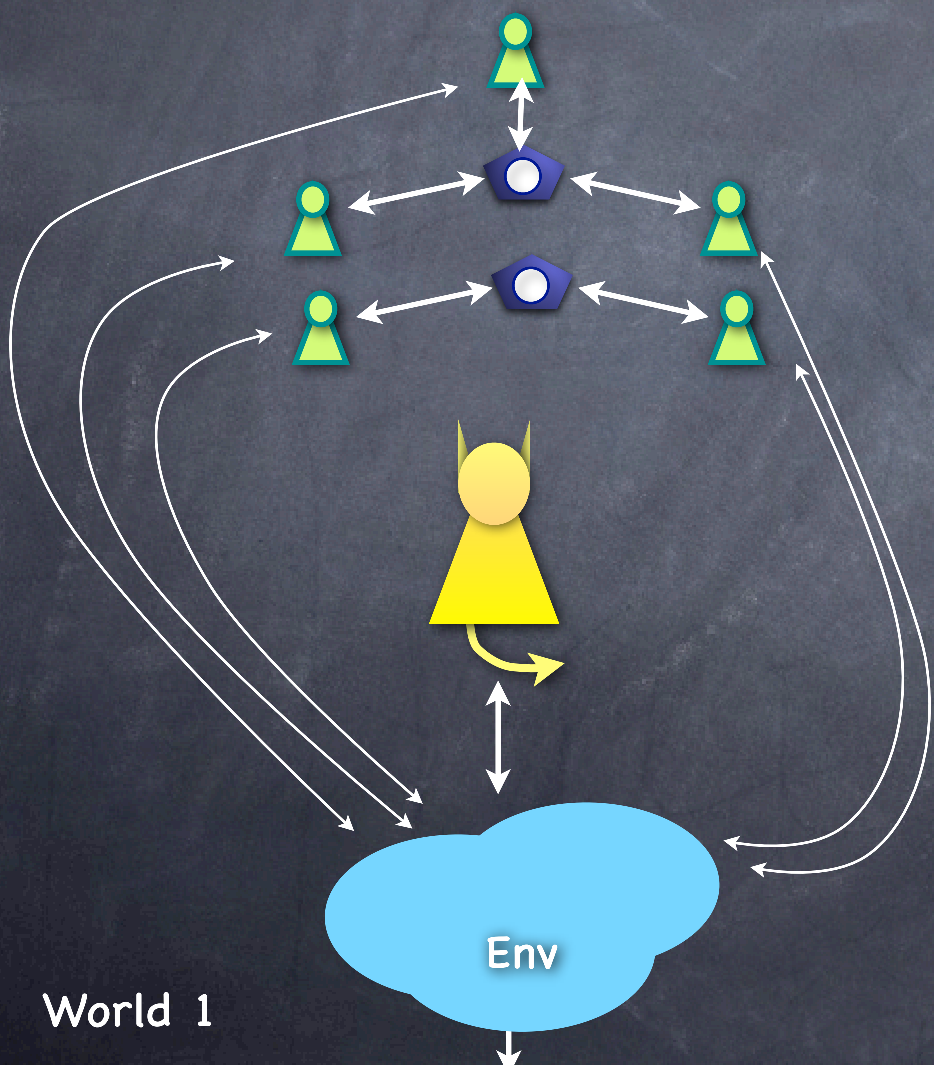


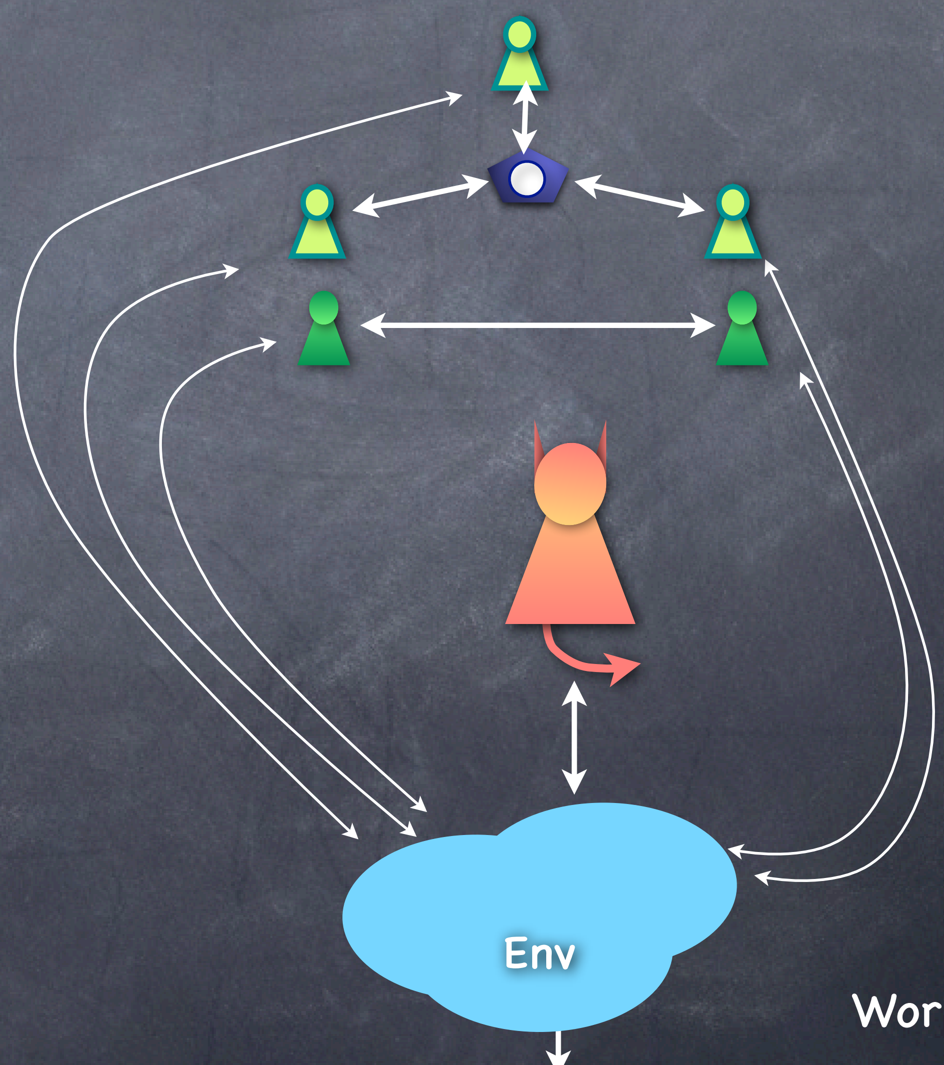
Composition Issues

- Standalone security definition doesn't ensure security when composed
- Different modes of composition
  - Sequential composition: protocols executed one after the other. Adversary communicates with the environment between executions. (OK by standalone security definition.)
  - Concurrent composition: multiple sessions (typically of the same protocol) are active at the same time, and the adversary can coordinate its actions across the sessions
    - Also, subroutine calls
  - Universal composition: Executed in an arbitrary environment which may include other protocol sessions (possibly calling this session as a subroutine). Live communication between environment and adversary.

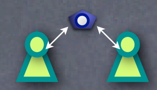
Universal Composition

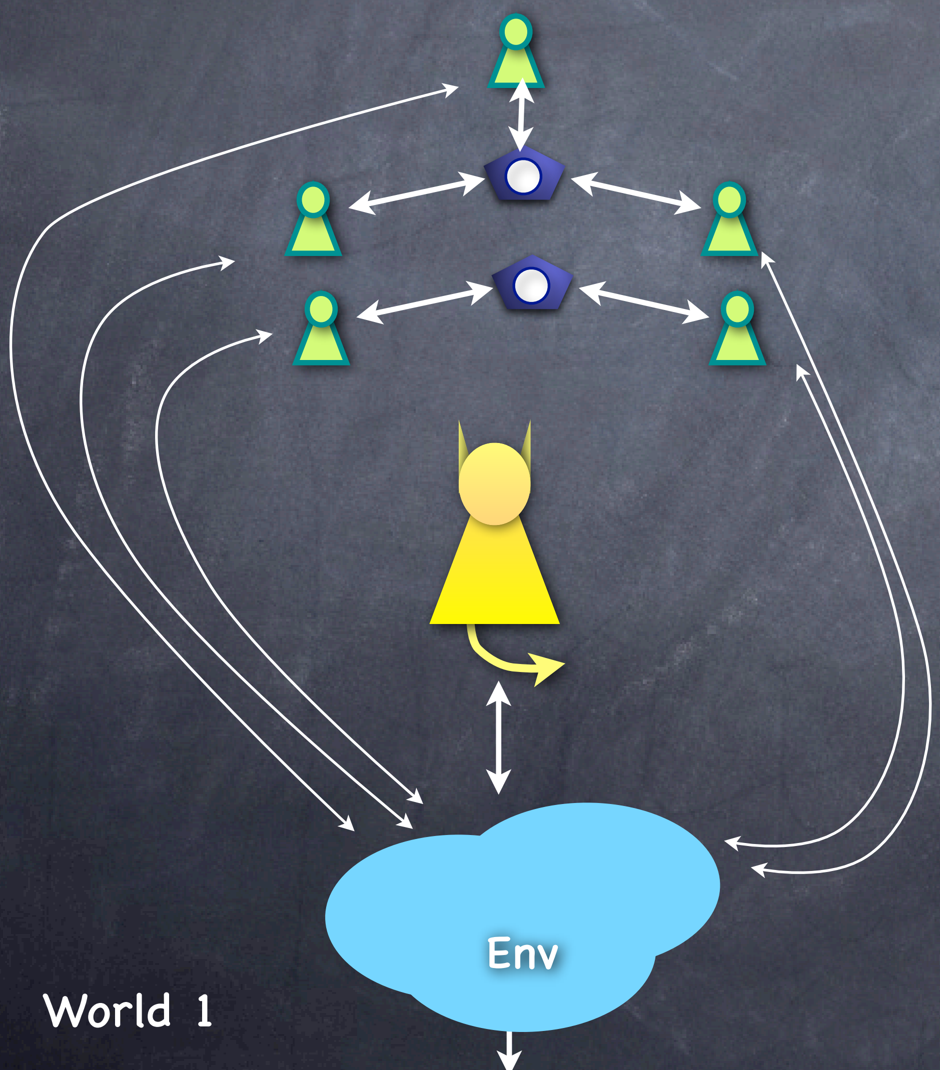
Replace protocol  with  which is as secure, etc.

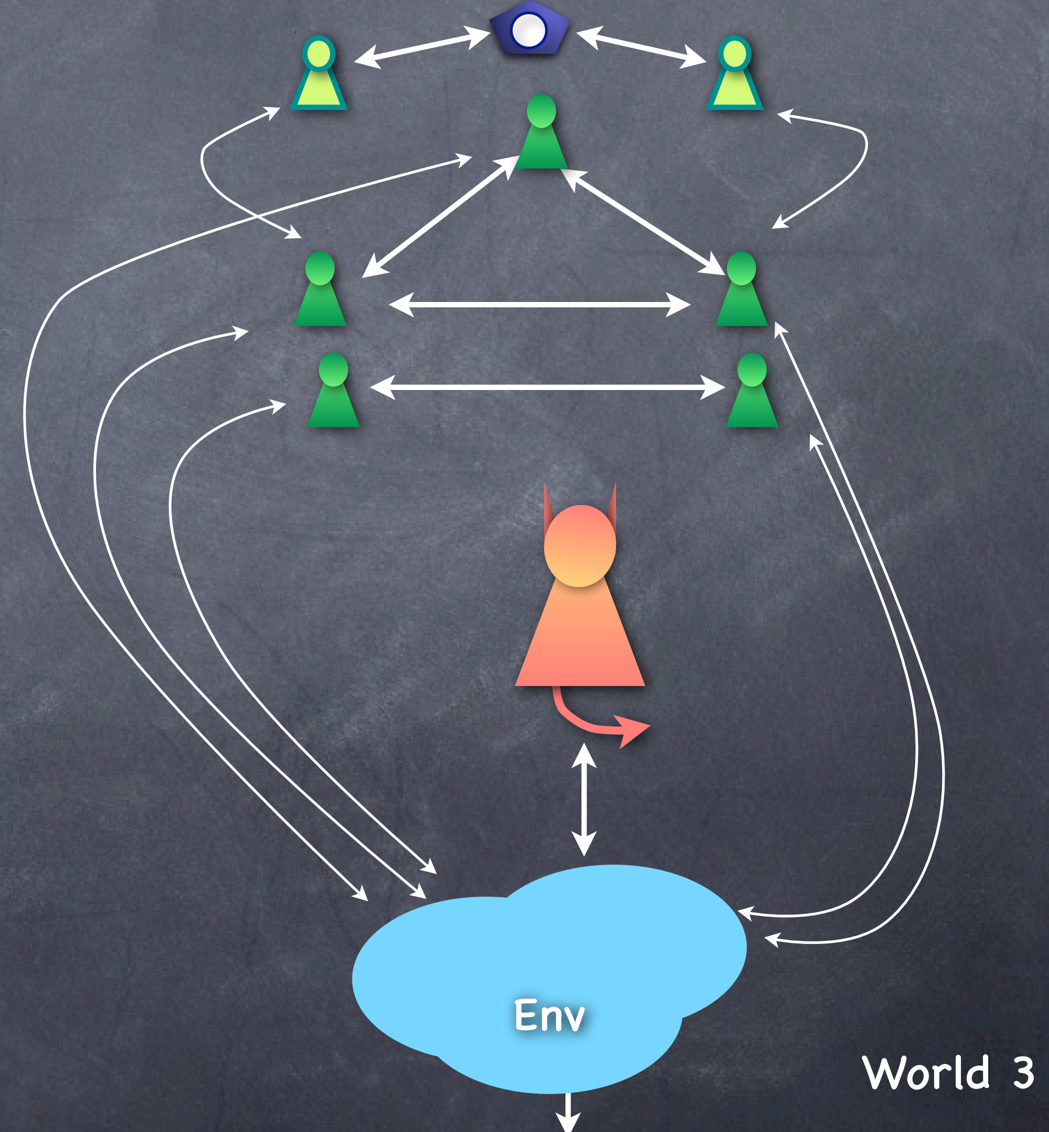




Universal Composition

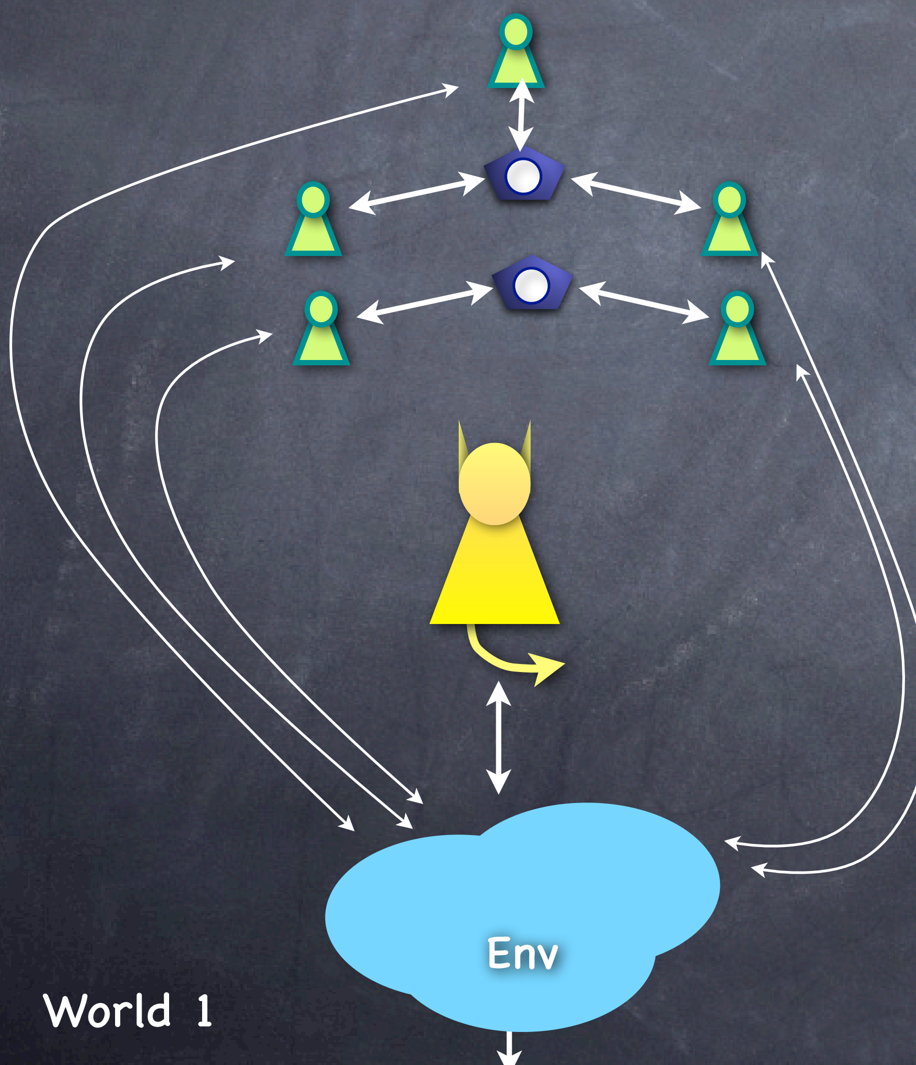
Replace protocol  with  which is as secure, etc.

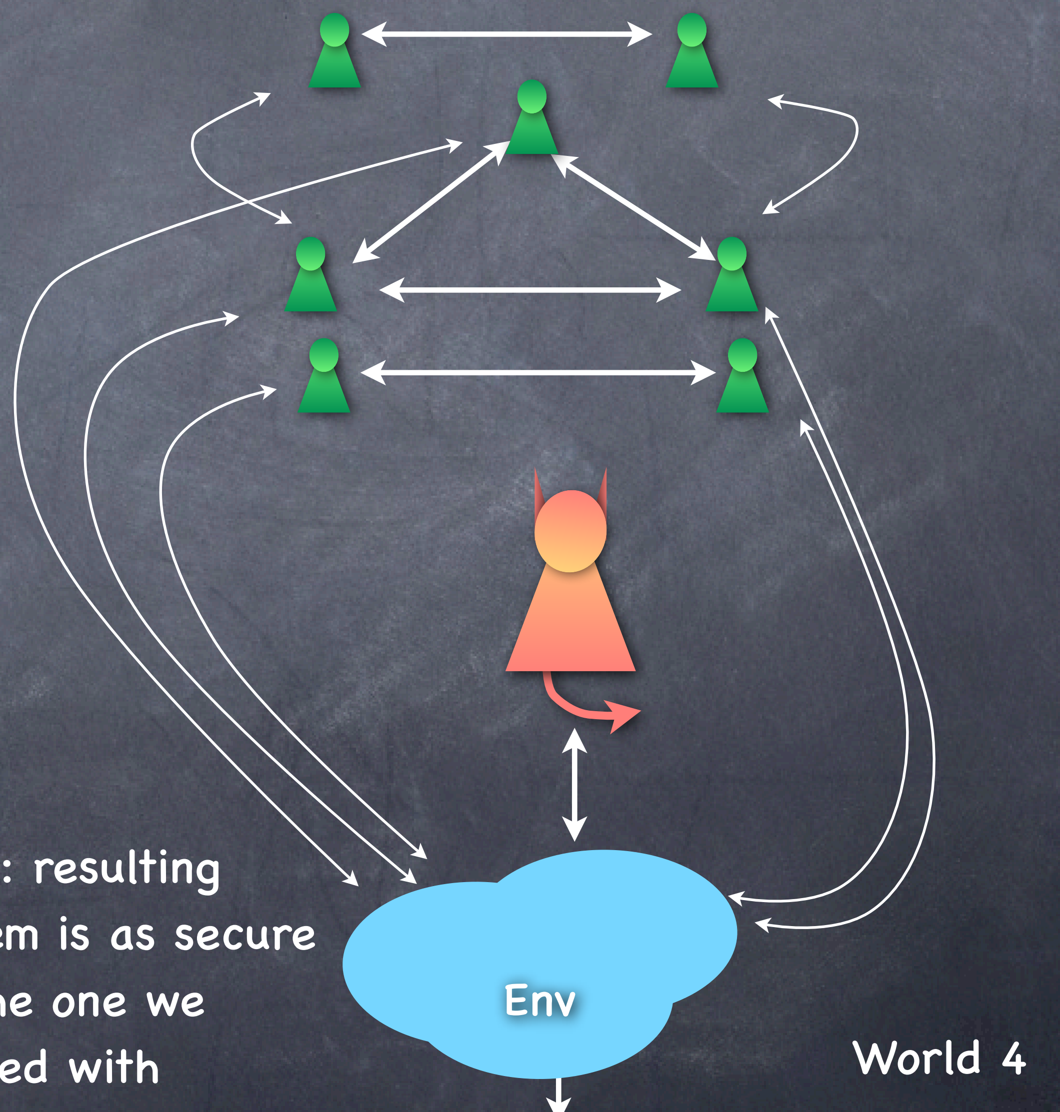




Universal Composition

Replace protocol [icon] with [icon] which is as secure, etc.





Hope: resulting system is as secure as the one we started with

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