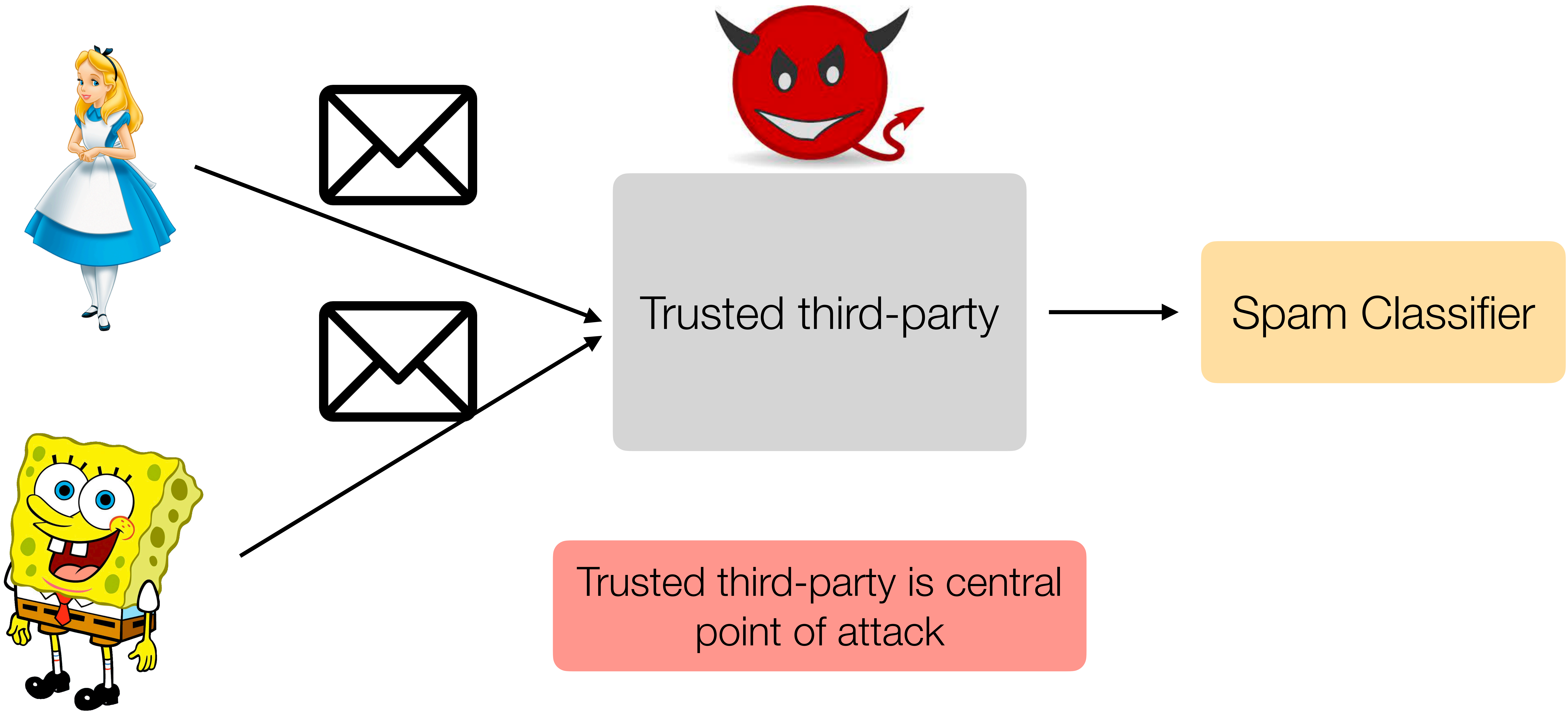
CS 350S: Privacy-Preserving Systems

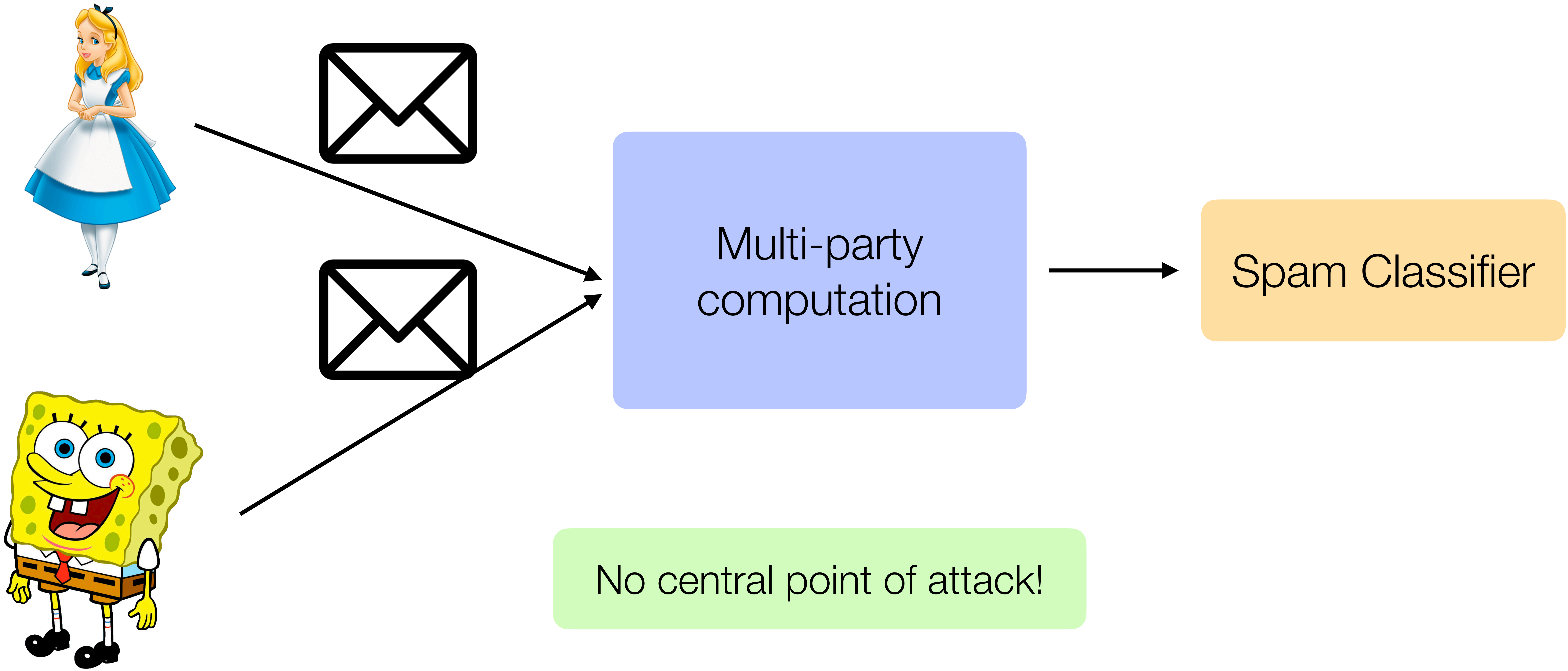
Multi-party computation II

Defining MPC



Defining MPC

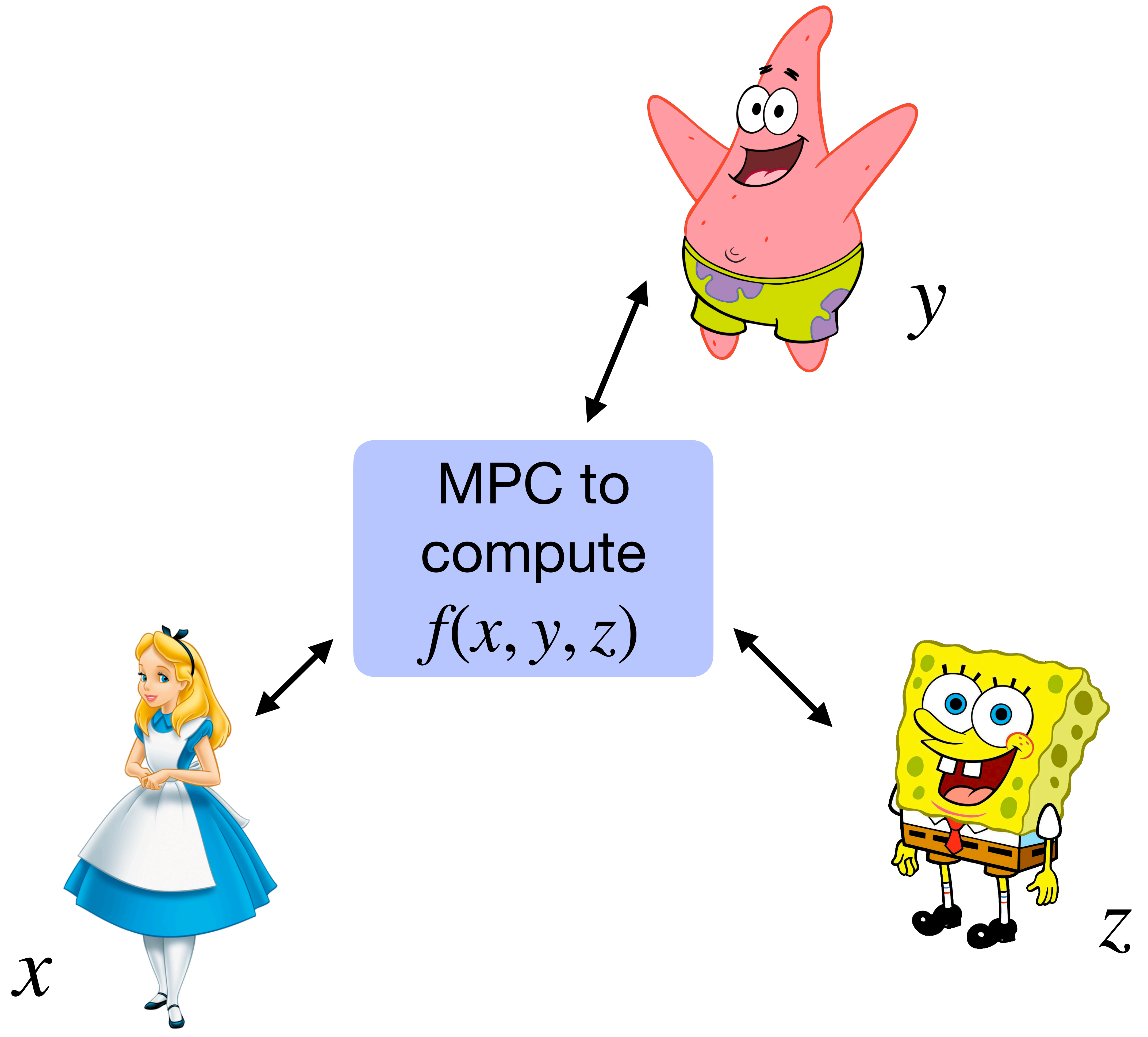
(Informal) Any computation that can be performed with a trusted third party can be securely computed *without* one!



Defining MPC

Parties with inputs x, y, z that want to jointly compute the function $f(x, y, z)$

- Assume encrypted, authenticated channels between parties
- Generalize to any number of parties



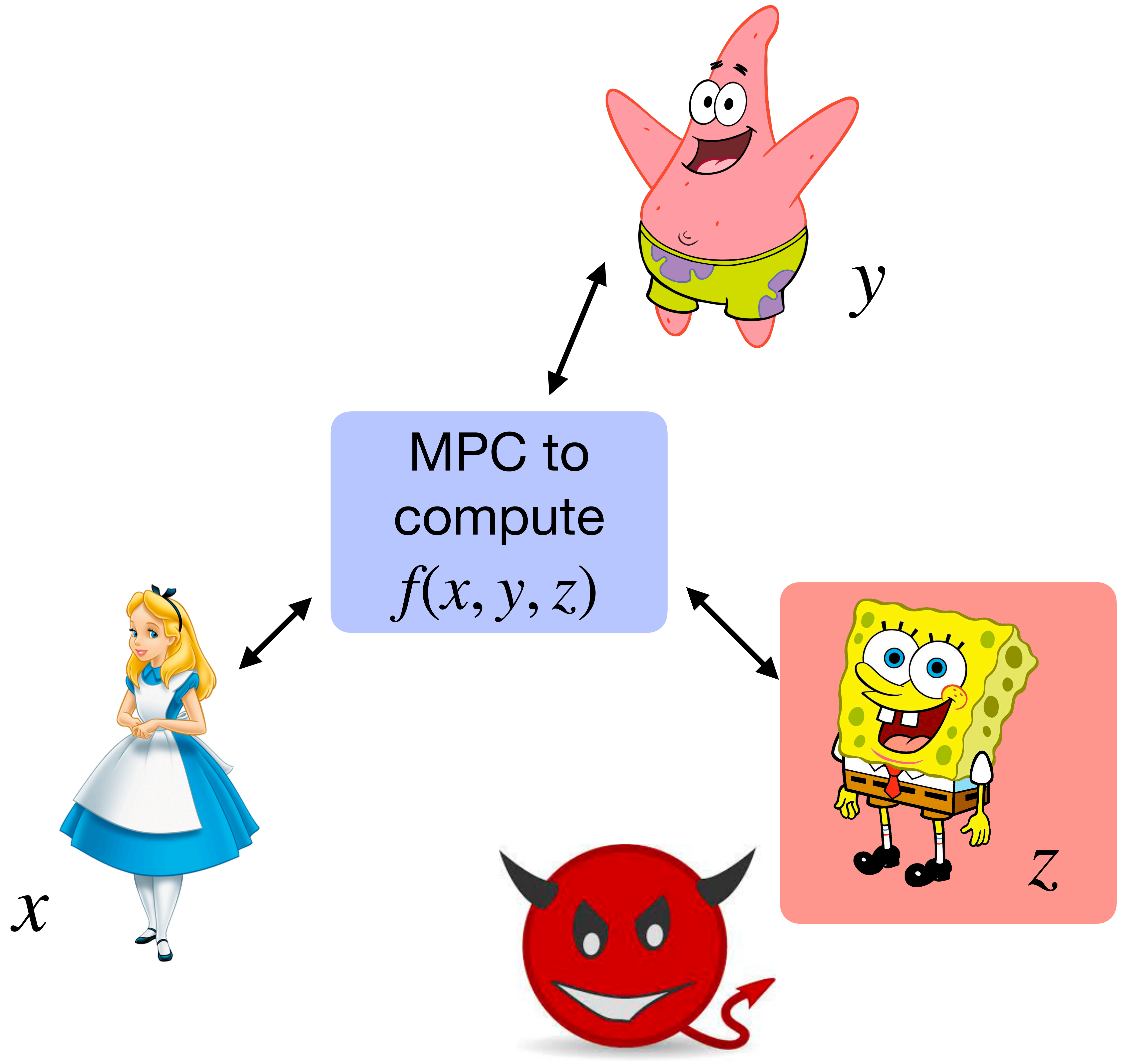
Defining MPC

Parties with inputs x, y, z that want to jointly compute the function $f(x, y, z)$

- Assume encrypted, authenticated channels between parties
- Generalize to any number of parties

Defends against attacker that compromises a subset of the parties

- Exact security properties depends on MPC protocol



Two adversary models in MPC

Semihonest: The corrupted parties *follow the protocol specification*. After the protocol completes, they look at the transcript and try to extract info about the honest parties' input.

Malicious: The corrupted parties may *arbitrarily deviate from the protocol specification* to learn extra info about the honest parties' inputs or trick them into producing the wrong output.

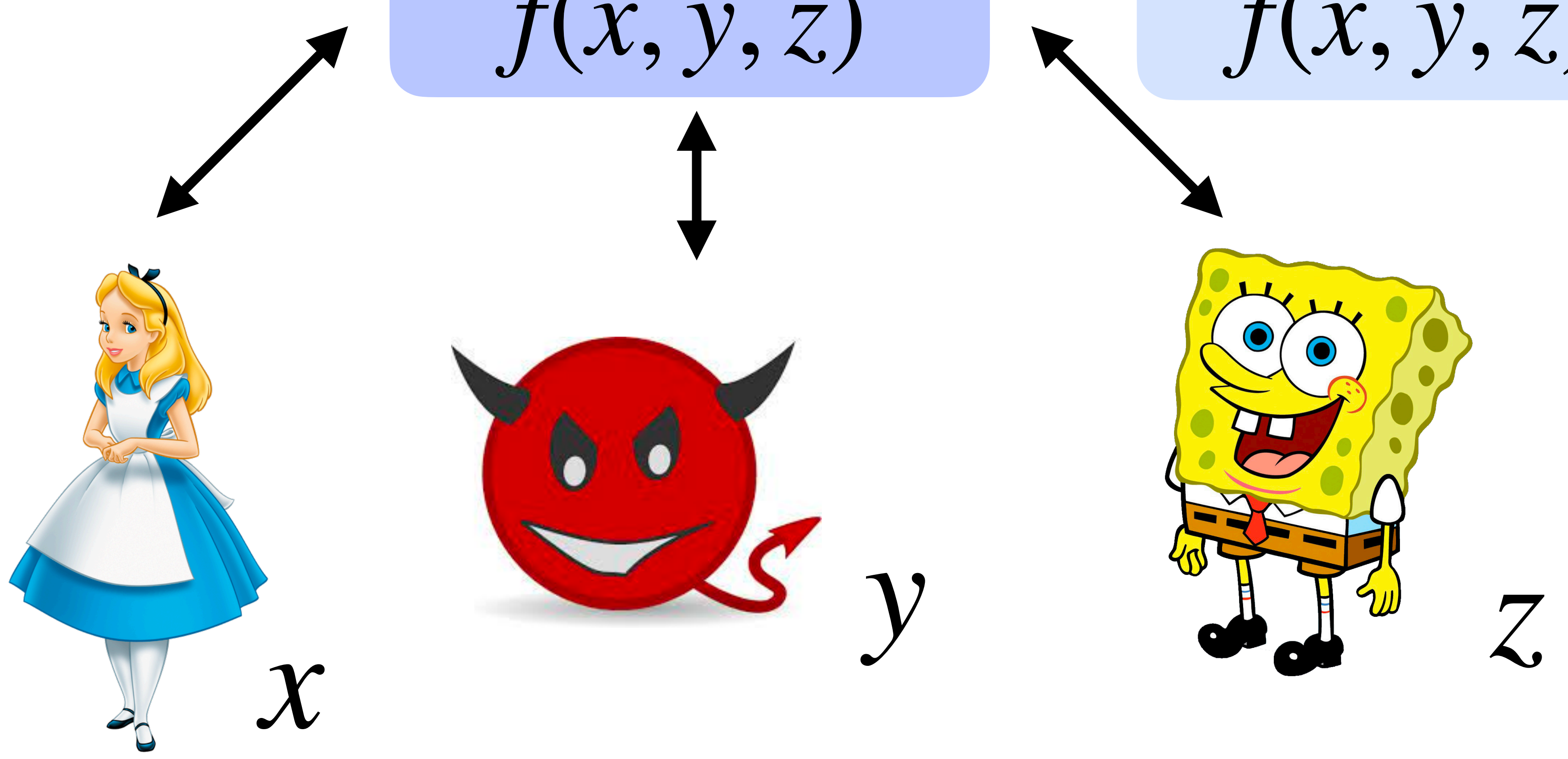
Defining semihonest security

Informally: Anything the adversary learns in an execution of the MPC, it could also have learned if all the parties were interacting with a trusted third party

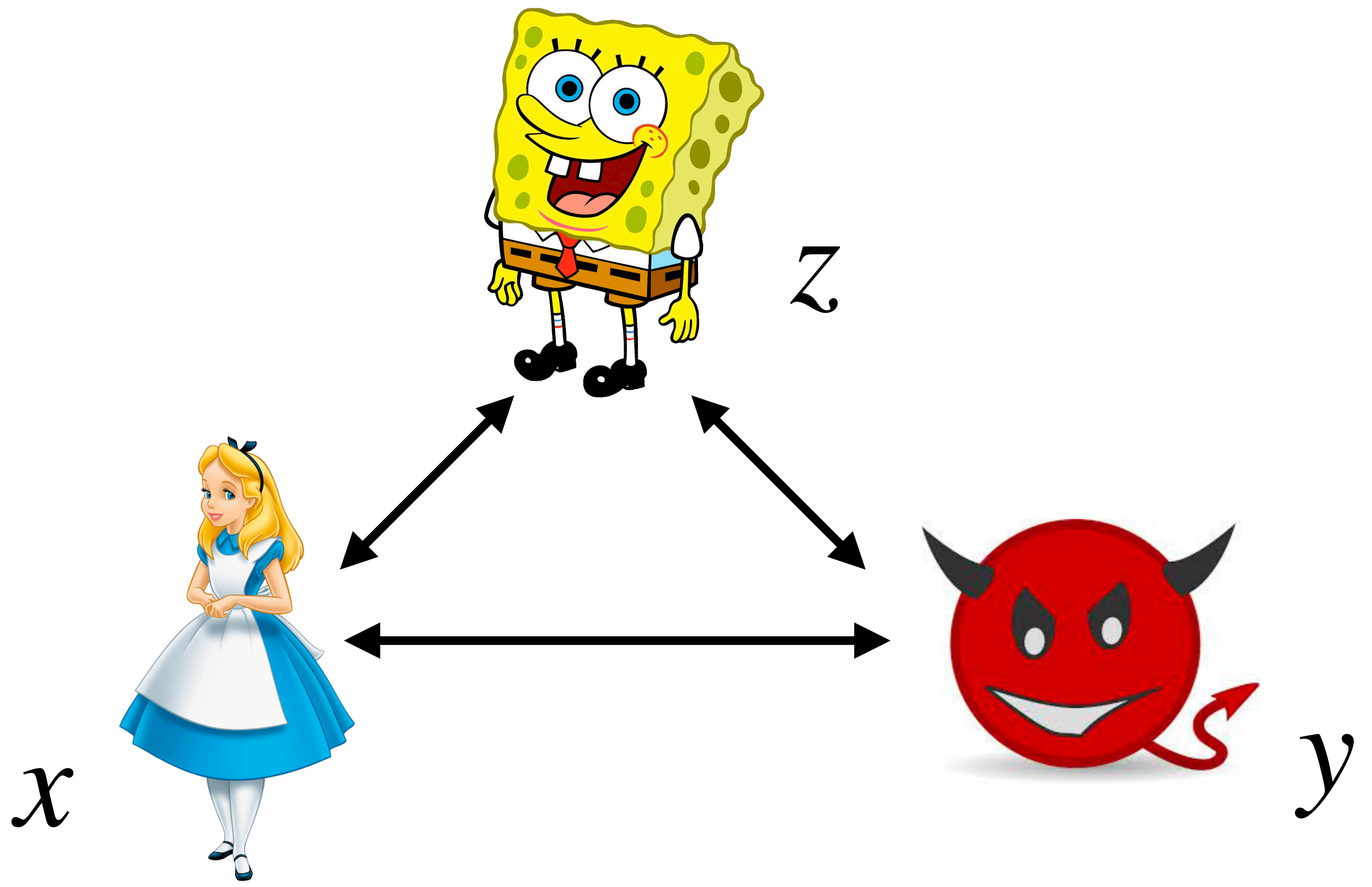
Ideal world

Trusted third-party for $f(x, y, z)$

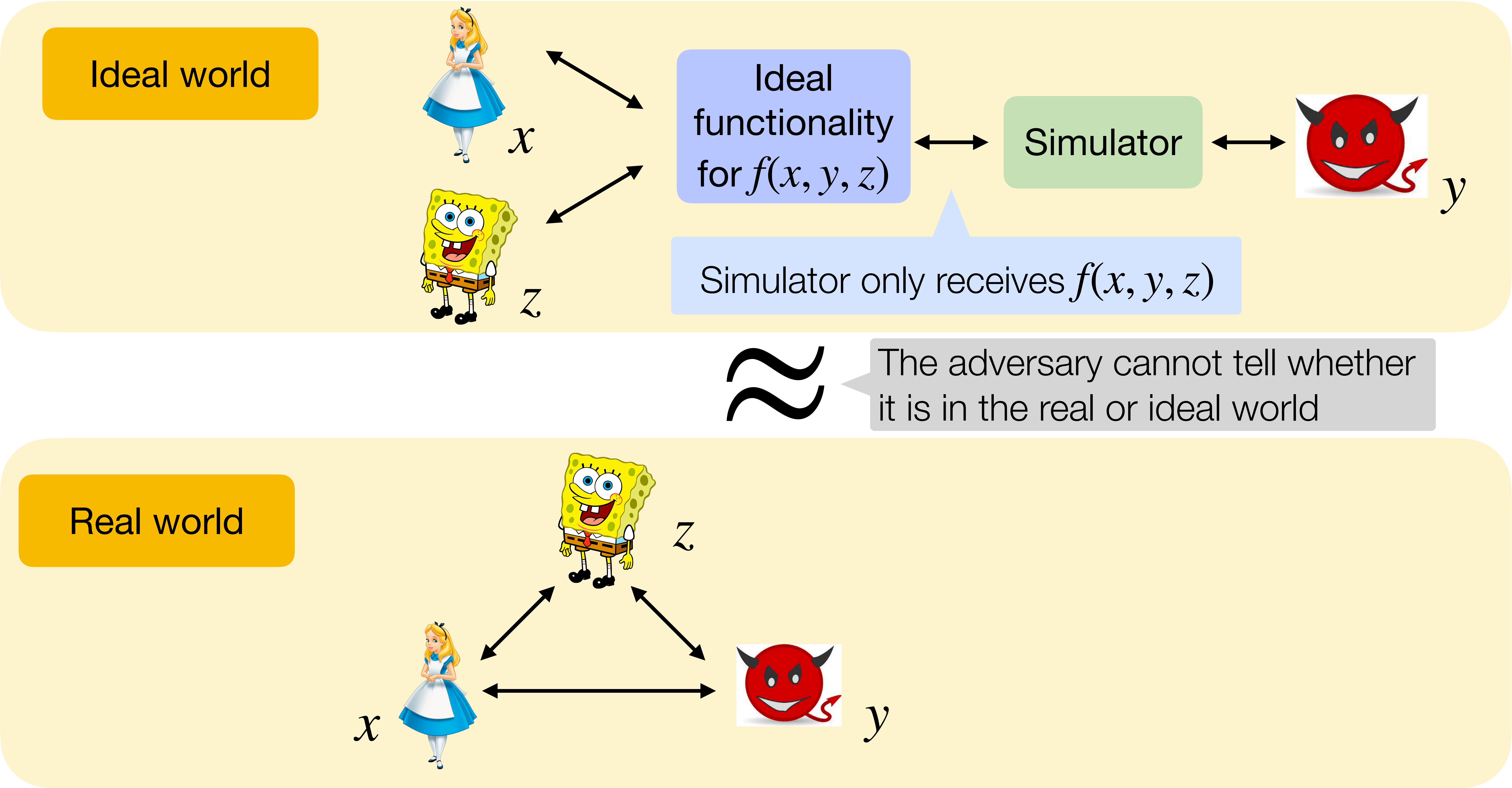
In ideal world, attacker learns
- Input of corrupted party y
- Output of computation $f(x, y, z)$



Real world



Defining semihonest security



Outline

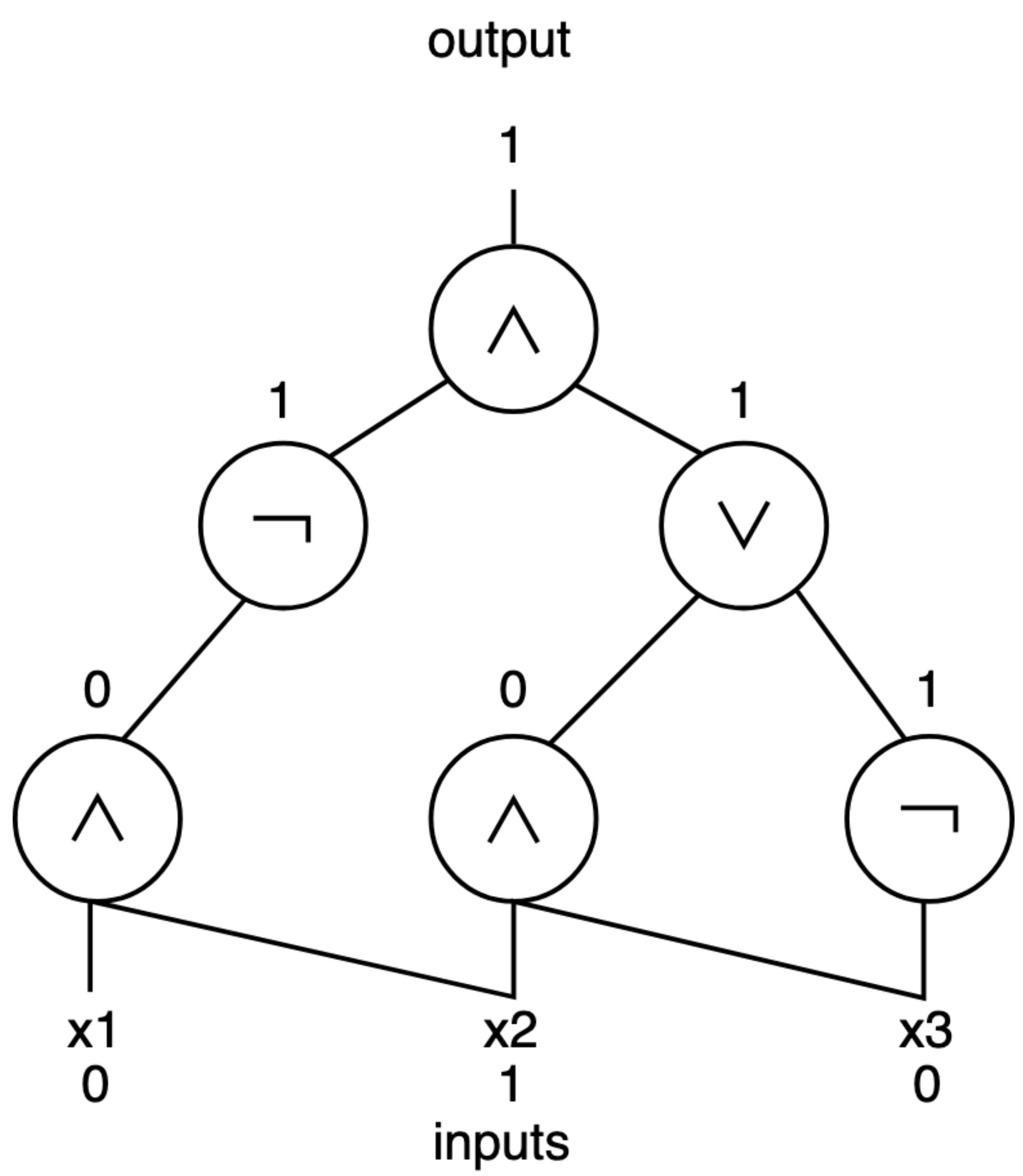
1. **Garbled circuits**
2. MAGE
3. Applications of MPC
4. Student presentation

Yao's garbled circuits [Yao82]

Multi-party computation for boolean circuits

Starting point: evaluating 1 AND gate

Next: generalize to a function f



Starting point: Garbled AND gate [Yao82]

Want: privately compute an AND gate for both parties' inputs (α, β)

2 parties: Garbler and Evaluator with inputs (b_0, b_1)

α	β	$\alpha \wedge \beta$
0	0	0
0	1	0
1	0	0
1	1	1

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$

α	β	$\alpha \wedge \beta$
0	0	0
0	1	0
1	0	0
1	1	1

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table

α	β	$\alpha \wedge \beta$
0	0	0
0	1	0
1	0	0
1	1	1

α	β	$\alpha \wedge \beta$
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, 0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, 0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, 0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, 1))$

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table
3. The garbler permutes $c_{00}, c_{01,}, c_{10}, c_{11}$ and sends them to the evaluator

α	β	$\alpha \wedge \beta$
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, 0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, 0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, 0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, 1))$

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table
3. The garbler permutes $c_{00}, c_{01,}, c_{10}, c_{11}$ and sends them to the evaluator
4. The garbler sends the key for its own input bit $k_L^{b_0}$ to the evaluator

α	β	$\alpha \wedge \beta$
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, 0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, 0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, 0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, 1))$

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table
3. The garbler permutes $c_{00}, c_{01,}, c_{10}, c_{11}$ and sends them to the evaluator
4. The garbler sends the key for its own input bit $k_L^{b_0}$ to the evaluator
5. The evaluator retrieves $k_R^{b_1}$ (and no other info) from the garbler using oblivious transfer

α	β	$\alpha \wedge \beta$
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, 0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, 0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, 0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, 1))$

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table
3. The garbler permutes $c_{00}, c_{01,}, c_{10}, c_{11}$ and sends them to the evaluator
4. The garbler sends the key for its own input bit $k_L^{b_0}$ to the evaluator
5. The evaluator retrieves $k_R^{b_1}$ (and no other info) from the garbler using oblivious transfer
6. The evaluator tries to decrypt each of $c_{00}, c_{01}, c_{10}, c_{11}$. Only 1 will decrypt to 0/1.

α	β	$\alpha \wedge \beta$
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, 0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, 0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, 0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, 1))$

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table
3. The garbler permutes $c_{00}, c_{01,}, c_{10}, c_{11}$ and sends them to the evaluator
4. The garbler sends the key for its own input bit $k_L^{b_0}$ to the evaluator
5. The evaluator retrieves $k_R^{b_1}$ (and no other info) from the garbler using oblivious transfer
6. The evaluator tries to decrypt each of $c_{00}, c_{01}, c_{10}, c_{11}$. Only 1 will decrypt to 0/1.
7. The evaluator sends the decrypted bit back to the garbler.

α	β	$\alpha \wedge \beta$
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, 0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, 0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, 0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, 1))$

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table
3. The garbler permutes $c_{00}, c_{01,}, c_{10}, c_{11}$ and sends them to the evaluator
4. The garbler sends the key for its own input bit $k_L^{b_0}$ to the evaluator
5. The evaluator retrieves $k_R^{b_1}$ (and no other info) from the garbler using oblivious transfer
6. The evaluator tries to decrypt each of $c_{00}, c_{01}, c_{10}, c_{11}$. Only 1 will decrypt to 0/1.
7. The evaluator sends the decrypted bit back to the garbler.

Why does the *garbler* not learn more than the output?

k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, 0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, 0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, 0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, 1))$

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table
3. The garbler permutes $c_{00}, c_{01,}, c_{10}, c_{11}$ and sends them to the evaluator
4. The garbler sends the key for its own input bit $k_L^{b_0}$ to the evaluator
5. The evaluator retrieves $k_R^{b_1}$ (and no other info) from the garbler using oblivious transfer
6. The evaluator tries to decrypt each of $c_{00}, c_{01}, c_{10}, c_{11}$. Only 1 will decrypt to 0/1.
7. The evaluator sends the decrypted bit back to the garbler.

Why does the *garbler* not learn more than the output?

Garbler only gets the oblivious transfer messages and the output from the evaluator

k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, 1))$

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table
3. The garbler permutes $c_{00}, c_{01,}, c_{10}, c_{11}$ and sends them to the evaluator
4. The garbler sends the key for its own input bit $k_L^{b_0}$ to the evaluator
5. The evaluator retrieves $k_R^{b_1}$ (and no other info) from the garbler using oblivious transfer
6. The evaluator tries to decrypt each of $c_{00}, c_{01}, c_{10}, c_{11}$. Only 1 will decrypt to 0/1.
7. The evaluator sends the decrypted bit back to the garbler.

Why does the *evaluator* not learn more than the output?

k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, 0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, 0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, 0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, 1))$

Starting point: Garbled AND gate

1. For each of (α, β), the garbler samples a pair of keys $(k_L^0, k_L^1), (k_R^0, k_R^1)$
2. The garbler generates ciphertexts for the truth table
3. The garbler permutes $c_{00}, c_{01,}, c_{10}, c_{11}$ and sends them to the evaluator
4. The garbler sends the key for its own input bit $k_L^{b_0}$ to the evaluator
5. The evaluator retrieves $k_R^{b_1}$ (and no other info) from the garbler using oblivious transfer
6. The evaluator tries to decrypt each of $c_{00}, c_{01}, c_{10}, c_{11}$. Only 1 will decrypt to 0/1.
7. The evaluator sends the decrypted bit back to the garbler.

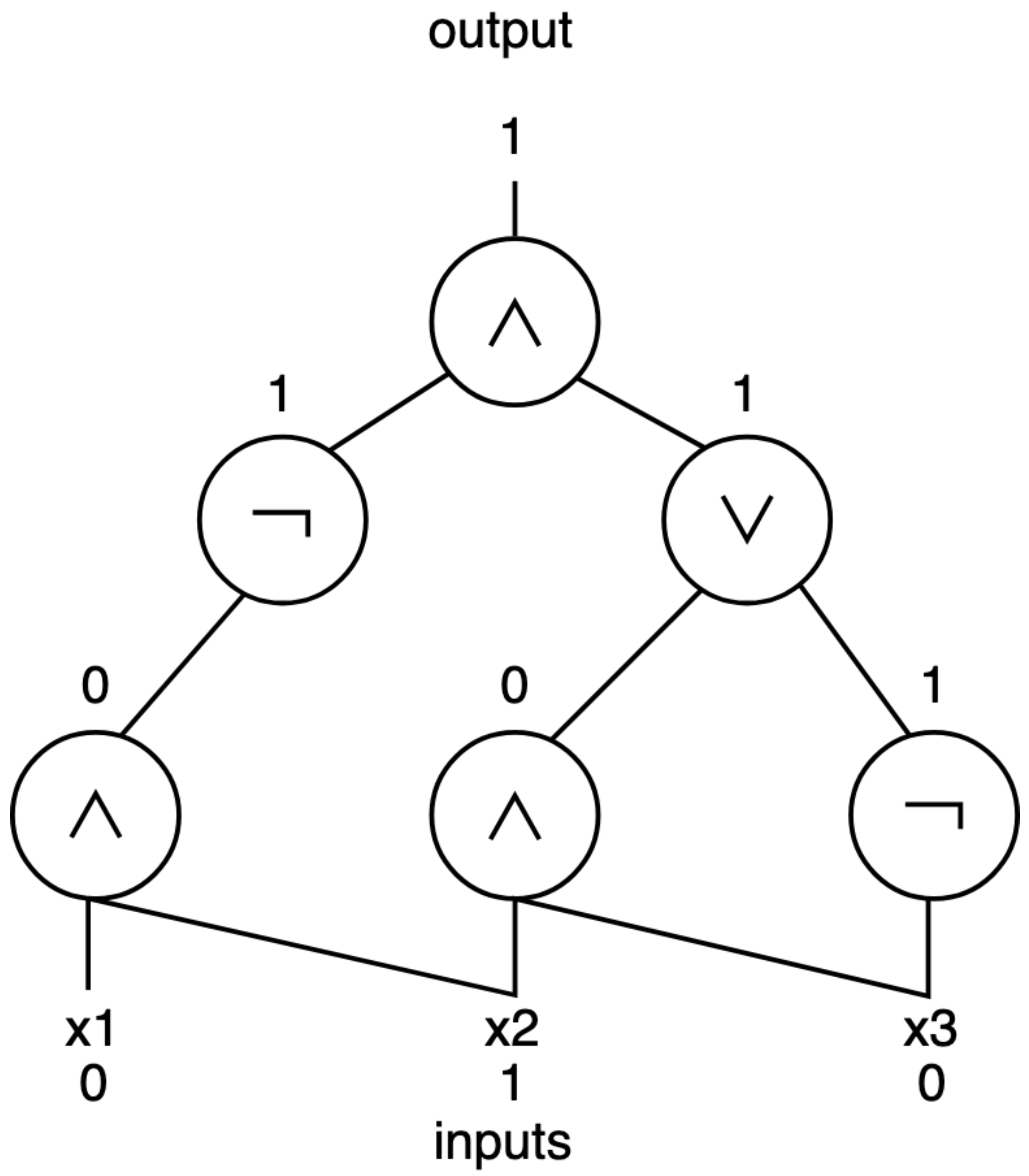
Why does the *evaluator* not learn more than the output?

- Ciphertexts: private via semantic security
- $k_L^{b_0}$: randomly generated key, evaluator does not learn value of b_0
- Oblivious transfer messages hide retrieved value

Yao's garbled circuits

Now: extend the warm-up to a general 2PC protocol for arbitrary boolean circuits

Say that evaluator has $x \in \{0,1\}^n$, garbler has $y \in \{0,1\}^n$, want to compute $f(x, y)$



Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")

α	β	$\alpha \wedge \beta$
0	0	0
0	1	0
1	0	0
1	1	1

α	β	$\alpha \wedge \beta$
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, k_{\text{out}}^0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, k_{\text{out}}^0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, k_{\text{out}}^0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, k_{\text{out}}^1))$

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)

α	β	$\alpha \wedge \beta$
0	0	0
0	1	0
1	0	0
1	1	1

α	β	$\alpha \wedge \beta$
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, k_{\text{out}}^0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, k_{\text{out}}^0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, k_{\text{out}}^0))$
k_L^1	k_R^1	$c_{10} = E(k_L^1, E(k_R^1, k_{\text{out}}^1))$

Yao’s garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect “correct” decryption”)
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)
4. The garbler sends the ciphertexts for each gate (randomly permuted) to the evaluator

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)
4. The garbler sends the ciphertexts for each gate (randomly permuted) to the evaluator
5. The garbler sends the keys for its input wires $k_1^{x_1}, \ldots, k_n^{x_n}$

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)
4. The garbler sends the ciphertexts for each gate (randomly permuted) to the evaluator
5. The garbler sends the keys for its input wires $k_1^{x_1}, \ldots, k_n^{x_n}$
6. The evaluator fetches $k_n^{y_1}, \ldots, k_{2n}^{y_n}$ using oblivious transfer

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)
4. The garbler sends the ciphertexts for each gate (randomly permuted) to the evaluator
5. The garbler sends the keys for its input wires $k_1^{x_1}, \ldots, k_n^{x_n}$
6. The evaluator fetches $k_n^{y_1}, \ldots, k_{2n}^{y_n}$ using oblivious transfer
7. The evaluator has the key for each of the $2n$ input wires and evaluates the circuit with these keys to get $f(x, y)$

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)
4. The garbler sends the ciphertexts for each gate (randomly permuted) to the evaluator
5. The garbler sends the keys for its input wires $k_1^{x_1}, \ldots, k_n^{x_n}$
6. The evaluator fetches $k_n^{y_1}, \ldots, k_{2n}^{y_n}$ using oblivious transfer
7. The evaluator has the key for each of the $2n$ input wires and evaluates the circuit with these keys to get $f(x, y)$
8. The evaluator sends the output $f(x, y)$ to the garbler

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)
4. The garbler sends the ciphertexts for each gate (randomly permuted) to the evaluator
5. The garbler sends the keys for its input wires $k_1^{x_1}, \ldots, k_n^{x_n}$
6. The evaluator fetches $k_n^{y_1}, \ldots, k_{2n}^{y_n}$ using oblivious transfer
7. The evaluator has the key for each of the $2n$ input wires and evaluates the circuit with these keys to get $f(x, y)$
8. The evaluator sends the output $f(x, y)$ to the garbler

Why is the protocol correct, i.e., output $f(x, y)$?

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)
4. The garbler sends the ciphertexts for each gate (randomly pe
5. The garbler sends the keys for its input wires $k_1^{x_1}, \ldots, k_n^{x_n}$
6. The evaluator fetches $k_n^{y_1}, \ldots, k_{2n}^{y_n}$ using oblivious transfer
7. The evaluator has the key for each of the $2n$ input wires and keys to get $f(x, y)$
8. The evaluator sends the output $f(x, y)$ to the garbler

Why is the protocol correct, i.e., output $f(x, y)$?

- At each gate, evaluator gets the key for the correct output of the gate.
- Given the correct input keys, the evaluate gets correct shares of 0/1 at the output gates
- Oblivious transfer ensures that evaluator gets the right keys

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)
4. The garbler sends the ciphertexts for each gate (randomly permuted) to the evaluator
5. The garbler sends the keys for its input wires $k_1^{x_1}, \ldots, k_n^{x_n}$
6. The evaluator fetches $k_n^{y_1}, \ldots, k_{2n}^{y_n}$ using oblivious transfer
7. The evaluator has the key for each of the $2n$ input wires and evaluates the circuit with these keys to get $f(x, y)$
8. The evaluator sends the output $f(x, y)$ to the garbler

Why is the protocol private?

Yao's garbled circuits

1. For each input and internal wire of the circuit, the garbler assigns a pair of keys (k_w^0, k_w^1)
2. For each gate of the circuit, the garbler generates 4 ciphertexts that encrypt the key associated with the output wire for the truth table of the gate (can detect "correct" decryption")
3. For each gate connected to an output wire, the garbler encrypts 0/1 according to the truth table (as before)
4. The garbler sends the ciphertexts for each gate (randomly pe
5. The garbler sends the keys for its input wires $k_1^{x_1}, \ldots, k_n^{x_n}$
6. The evaluator fetches $k_n^{y_1}, \ldots, k_{2n}^{y_n}$ using oblivious transfer
7. The evaluator has the key for each of the $2n$ input wires and e keys to get $f(x, y)$
8. The evaluator sends the output $f(x, y)$ to the garbler

Why is the protocol private?

- The garbler only sees oblivious transfer messages and the output
- The evaluator only sees the permuted ciphertexts, oblivious transfer messages, and the output

Outline

1. Garbled circuits
2. **MAGE**
3. Applications of MPC
4. Student presentation

MAGE motivation

Garbled circuit: Each bit corresponds to 4 ciphertexts

- Blowup in space

Homomorphic encryption: Ciphertexts require more storage than plaintext values

Size of memory limits the types of computations that are practical to run

MAGE key idea

[Kumar, Culler, Popa]

Observation: memory access patterns of MPC programs are deterministic

- Data values are not visible, and so memory accesses are data-independent (obliviousness)

Key idea: Obliviousness makes it possible to compute the set of memory accesses in advance

- Can prefetch data ahead of time

Traditional memory management vs. MAGE

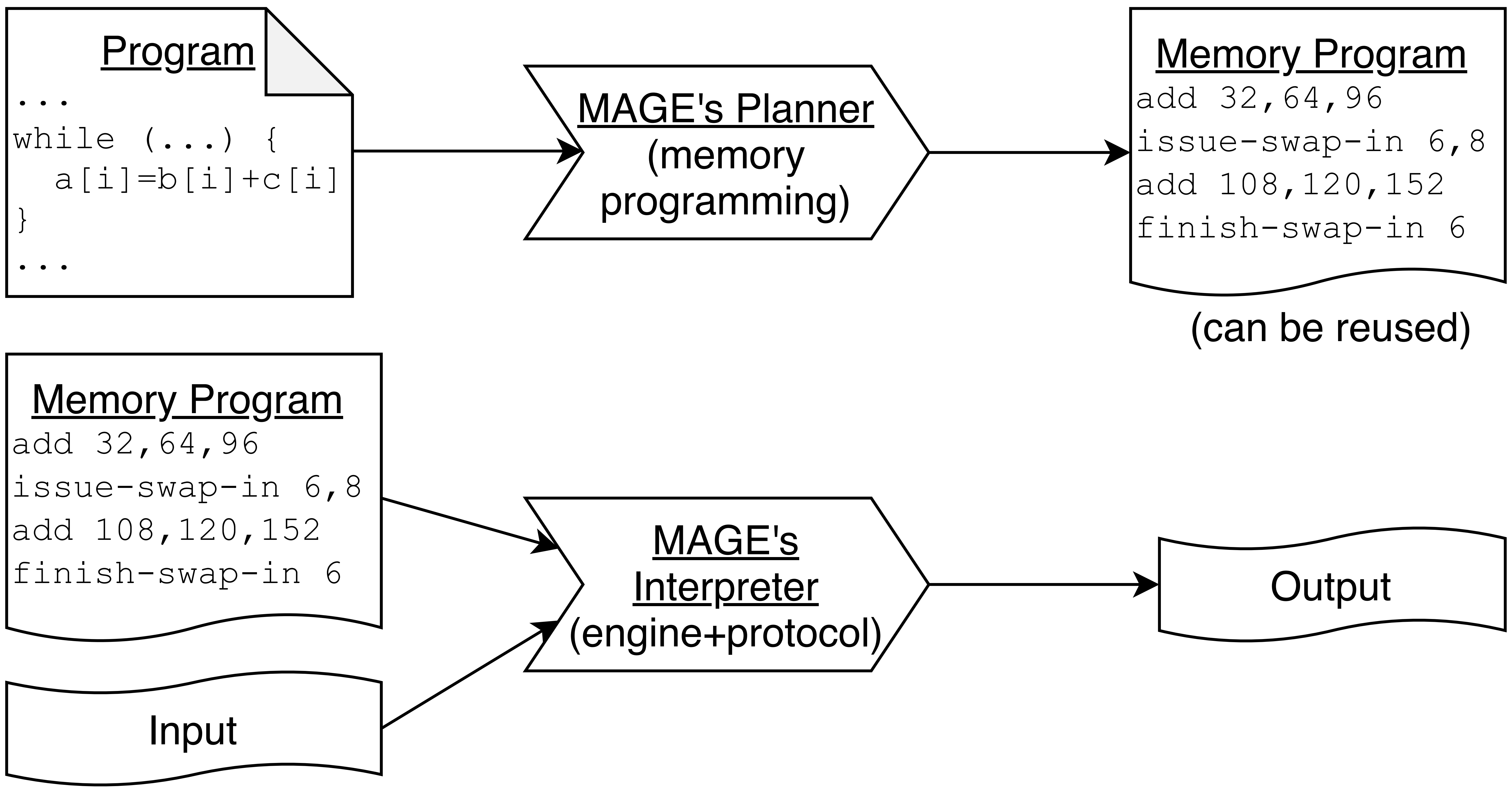
Virtual memory

- Don’t know which memory address will be accessed next
- Page in memory from disk as needed

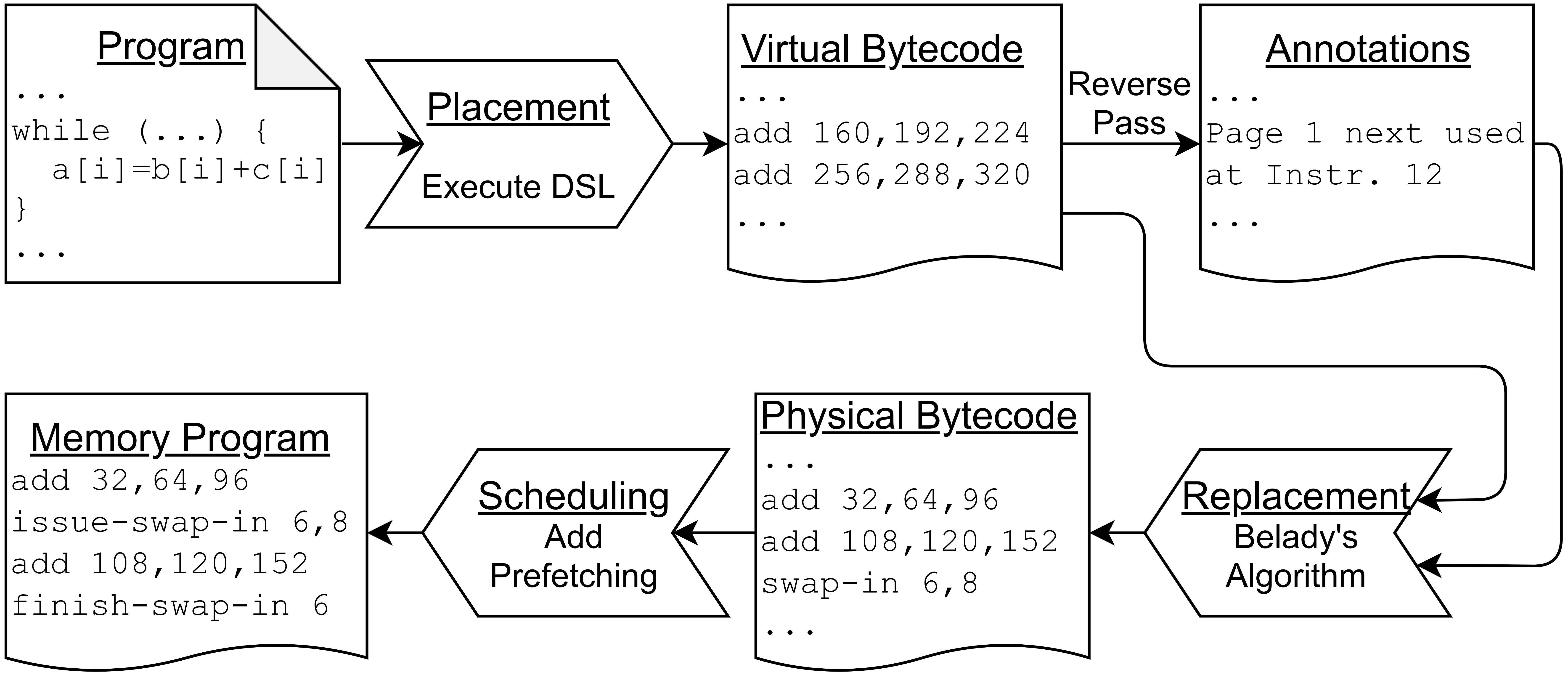
MAGE

- Know exactly which memory address will be accessed next
- Prefetch memory before it is needed
- Dramatically reduces overheads of virtual memory

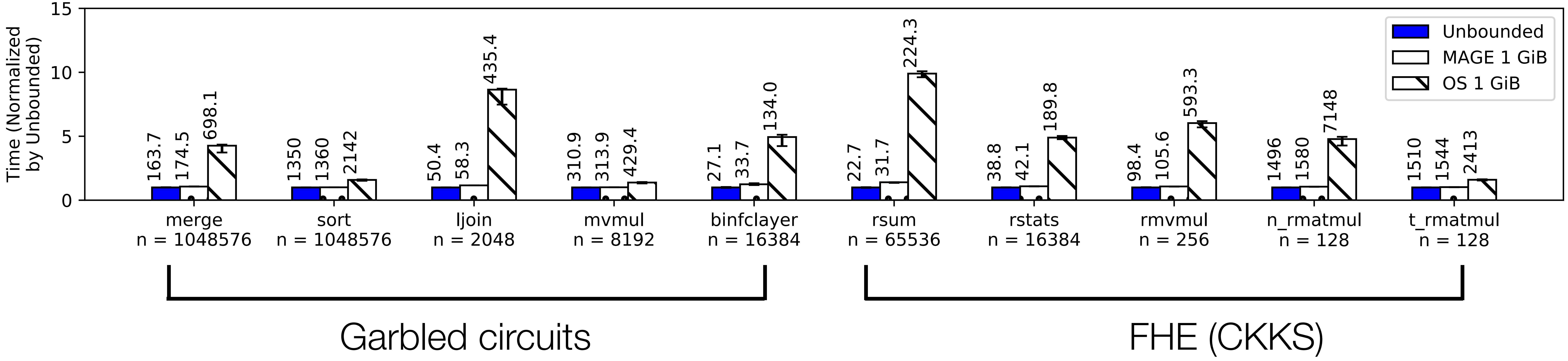
MAGE design steps



MAGE planner



MAGE evaluation



Outline

1. Garbled circuits
2. MAGE
3. **Applications of MPC**
4. Student presentation

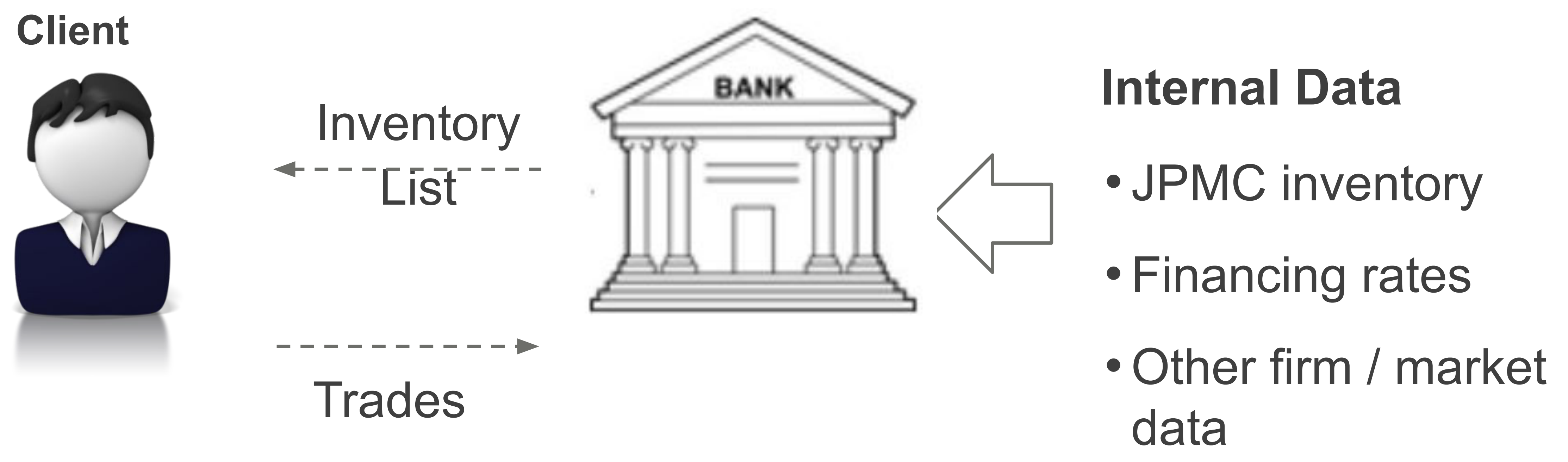
Privacy-preserving inventory matching at J.P. Morgan

[Polychroniadou, Asharov, Diamond, et al.]

JP Morgan publishes a daily list of inventory at a discount to clients

- Based on aggregated information on previous transactions from clients

This inventory list makes it possible to offer good rates, but can leak some information about trading strategy

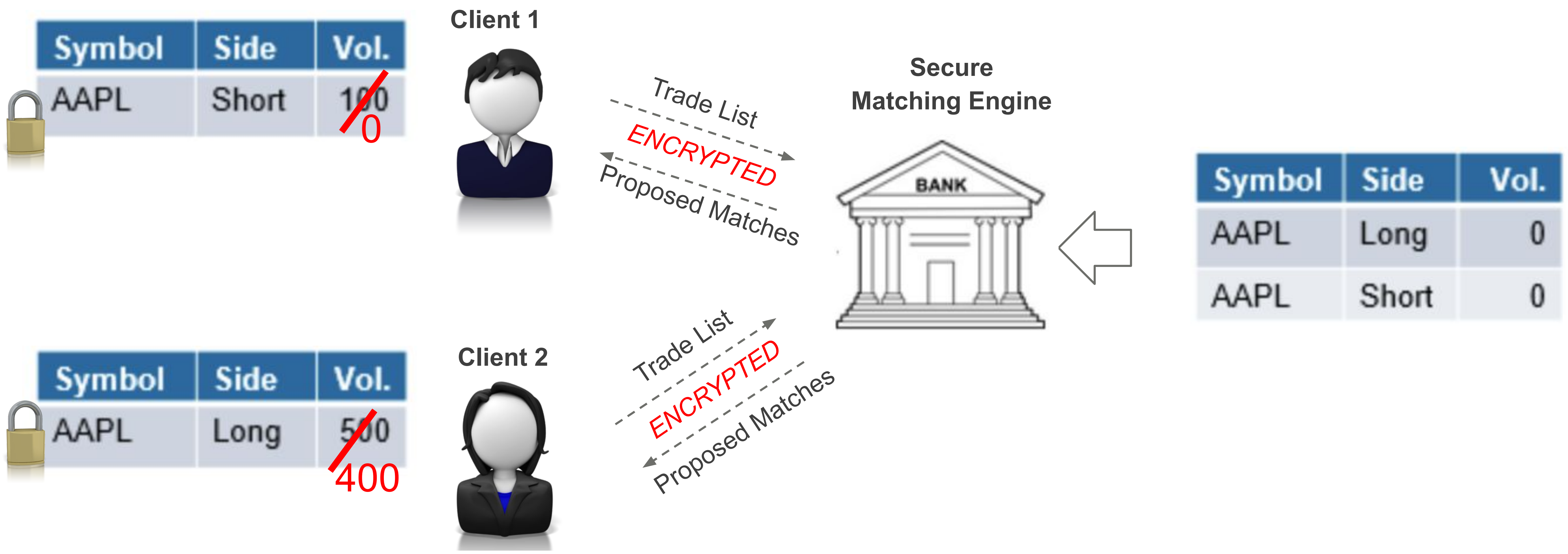


Privacy-preserving inventory matching at J.P. Morgan

[Polychroniadou, Asharov, Diamond, et al.]

Approach: Inventory matching between clients in MPC

- Clients send encrypted trade list
- Server provides full inventory list
- Match trades against other clients in MPC
- As of 2023, running in production



Cryptocurrency wallets

Fireblocks, Fordefi, Safeheron, …

Cryptocurrency wallets manage secret keys for signing user transactions

Keeping secret keys in a single location creates a single point of security failure (potentially worth millions of dollars!)

Approach: split secret key across different entities and use MPC to sign a transaction

- Secret key is never materialized in one location
- Often combined with secure hardware to offer an additional layer of protection

Genomic analysis

[Jagadeesh, Wu, Birgmeier, Boneh, Bejerano]

For medical research: Want to compare patient's genome with as many other genomes as possible

For privacy: Hide genome, as it reveals sensitive information

Approach: use MPC to find and reveal causative genetic variants while protecting remaining variants

- Helped diagnose real patients and discover previously unknown gene-disease associations
- Participants learned nothing about each other except shared disease-causing gene

Outline

1. Garbled circuits
2. MAGE
3. Applications of MPC
4. **Student presentation**

References

Jagadeesh, Karthik A., David J. Wu, Johannes A. Birgmeier, Dan Boneh, and Gill Bejerano. "Deriving genomic diagnoses without revealing patient genomes." *Science* 357, no. 6352 (2017): 692-695.

Kumar, Sam, David E. Culler, and Raluca Ada Popa. "{MAGE}: Nearly Zero-Cost Virtual Memory for Secure Computation." In 15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21), pp. 367-385. 2021.

Polychroniadou, Antigoni, Gilad Asharov, Benjamin Diamond, Tucker Balch, Hans Buehler, Richard Hua, Suwen Gu, Greg Gimler, and Manuela Veloso. "Prime match: A {Privacy-Preserving} inventory matching system." In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 6417-6434. 2023.

Yakoubov, Sophia. "A gentle introduction to yao's garbled circuits." preprint on webpage at https://web. mit. edu/sonka89/www/papers/2017ygc. pdf (2017).

Yao, Andrew C. "Protocols for secure computations." In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pp. 160-164. IEEE, 1982.

https://crypto.stanford.edu/cs355/18sp/lec6.pdf