Secure Computation with Sublinear Cost

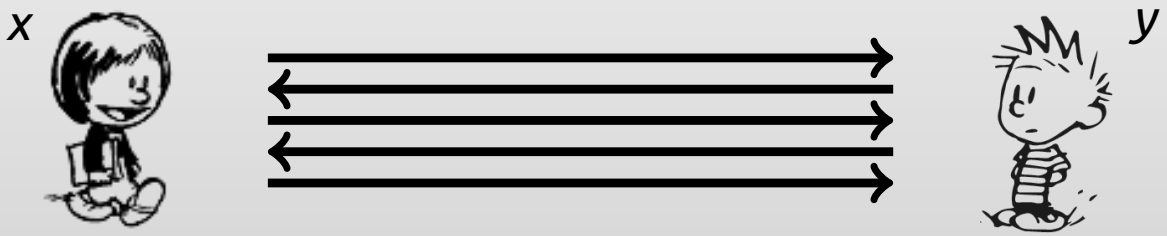
Mike Rosulek

Oregon State University OSU

Collaborators: Arash Afshar / Zhangxiang Hu / Payman Mohassel

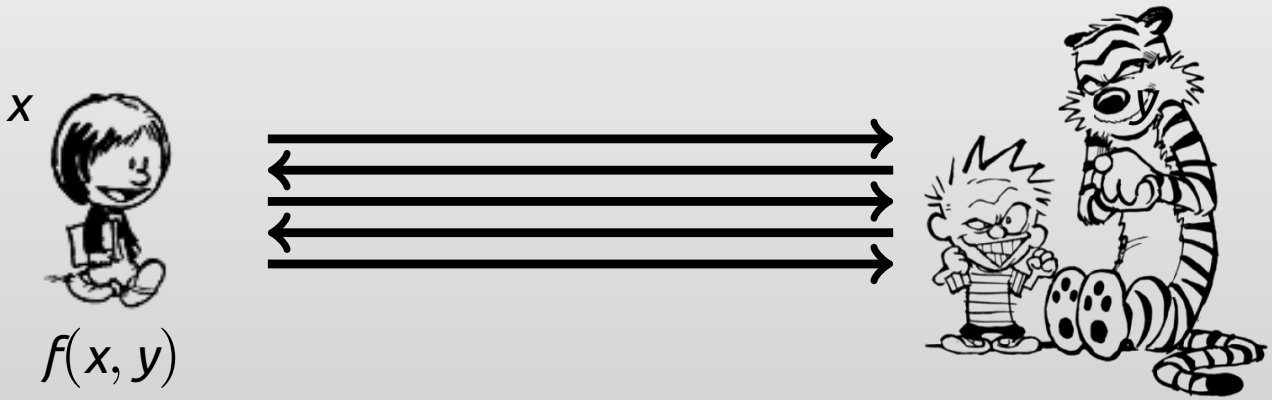
Secure 2-party computation

Secure 2-party computation

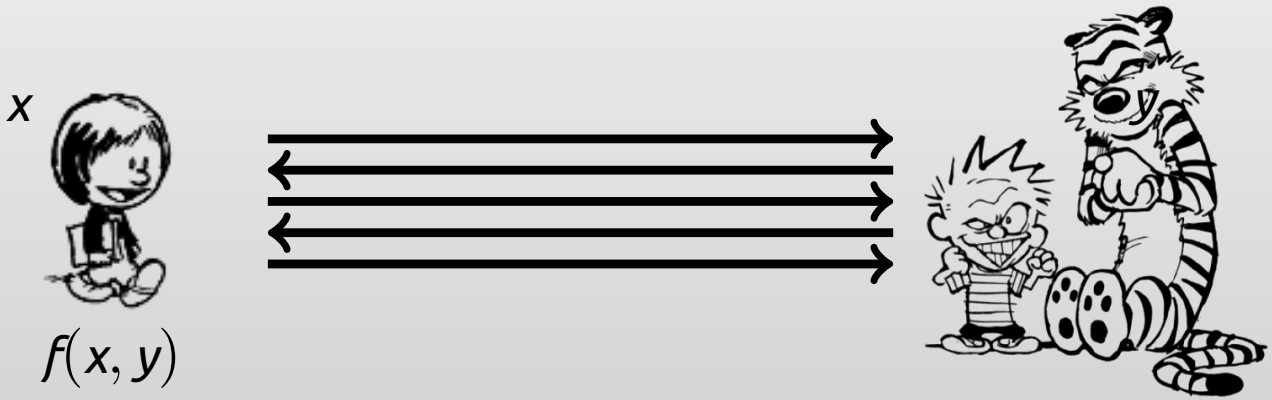


Secure 2-party computation

Secure 2-party computation



Secure 2-party computation

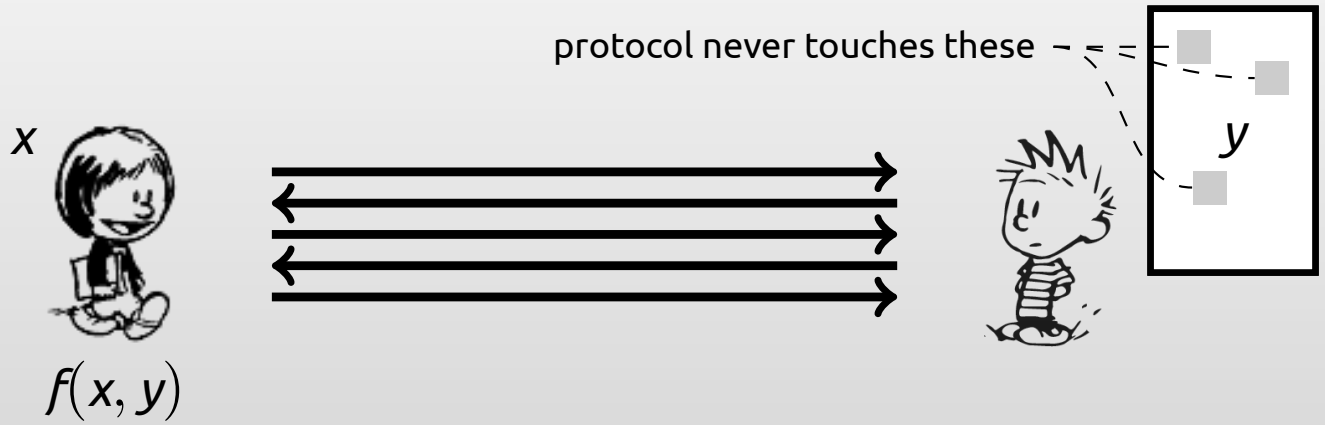


Examples:

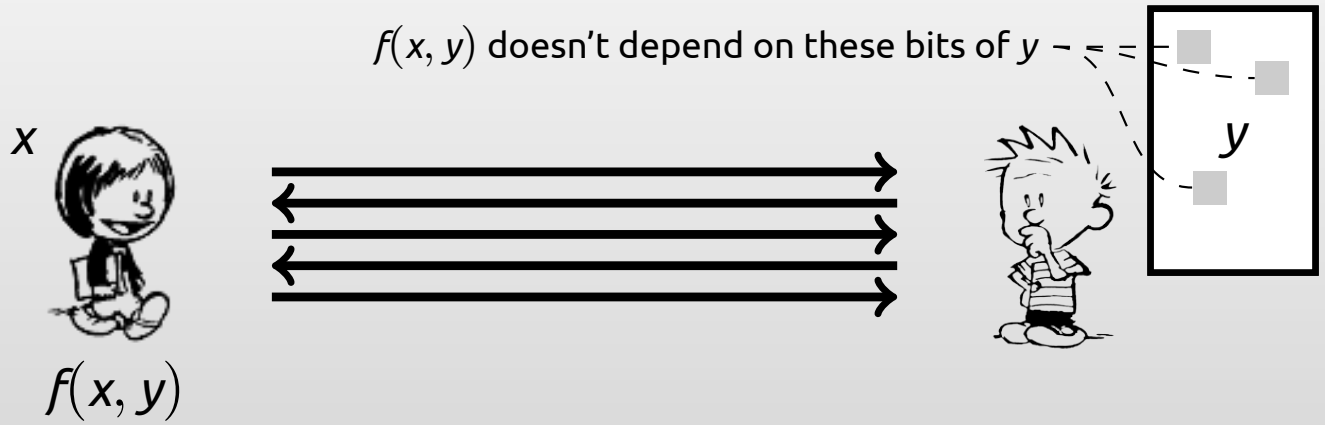
- Run proprietary classifier x on private data y
- Evaluate statistics on combined medical records x & y
- . . .

Fundamental Limits

Fundamental Limits



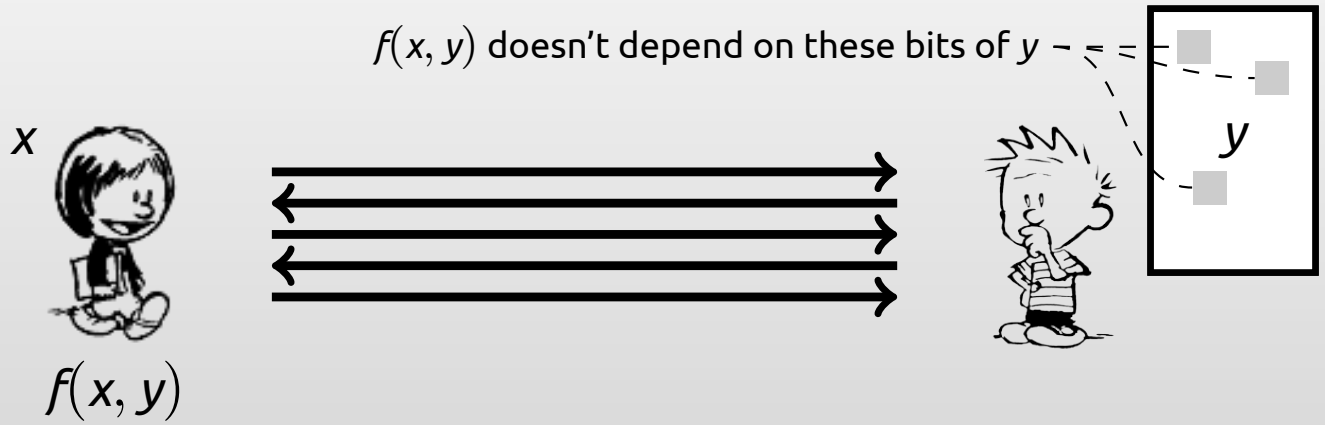
Fundamental Limits



Example:

- $y =$ genetic database
- $x =$ DNA markers
- $f(x, y) =$ diagnosis

Fundamental Limits



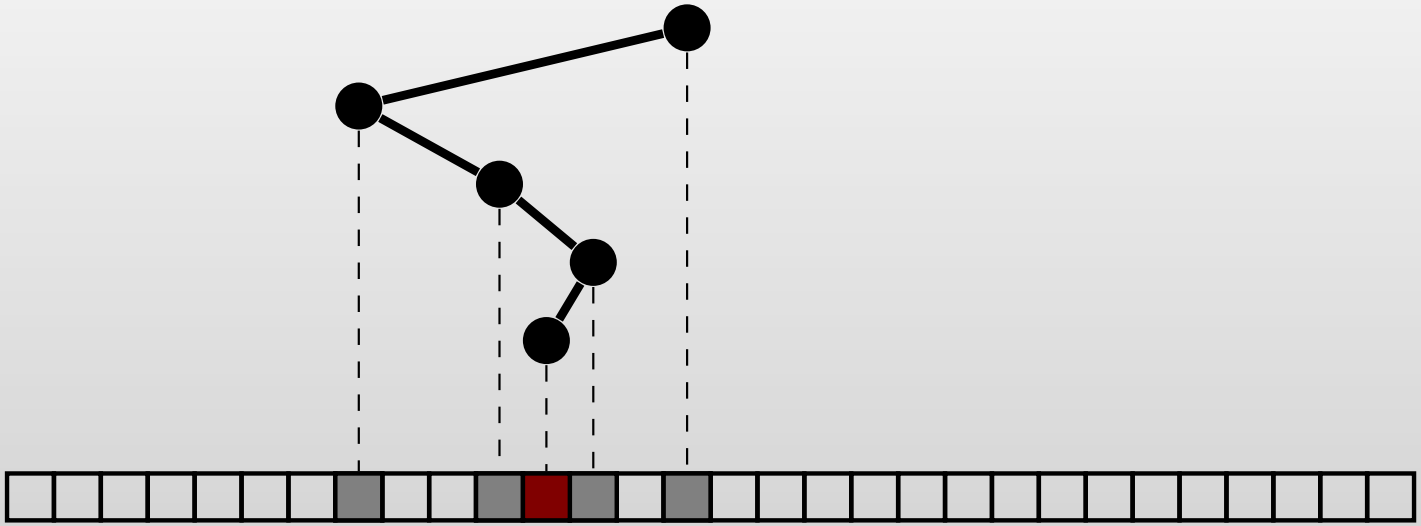
Example:

- $y =$ genetic database
- $x =$ DNA markers
- $f(x, y) =$ diagnosis

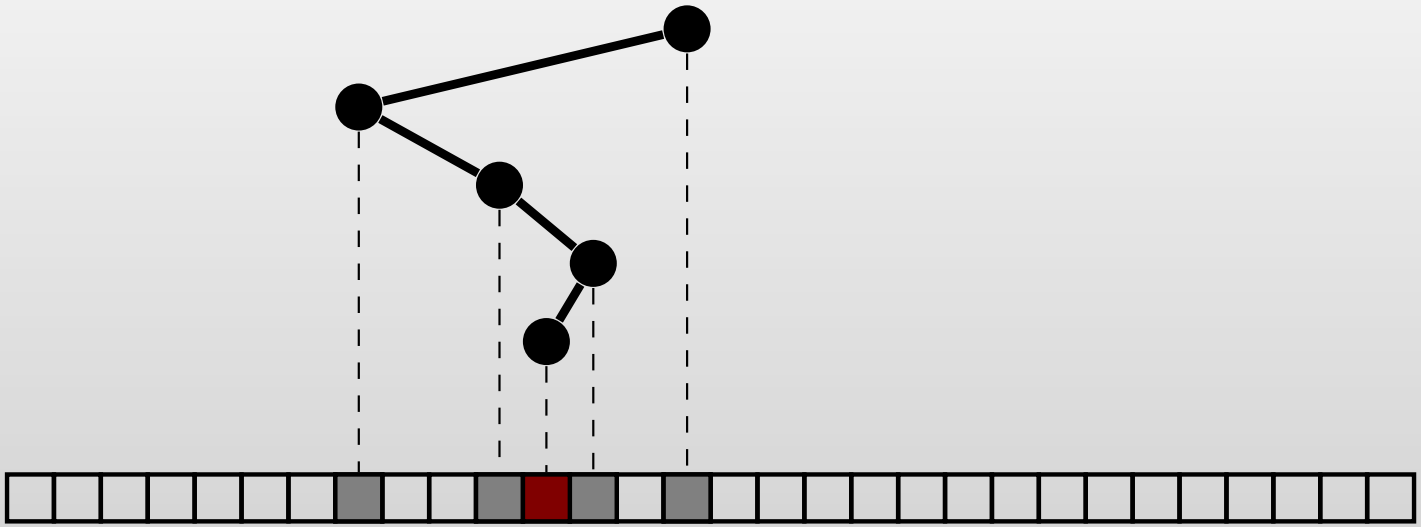
$\Rightarrow$ **in general**, security demands that all of the data is touched

Limits of Standard Techniques

Limits of Standard Techniques

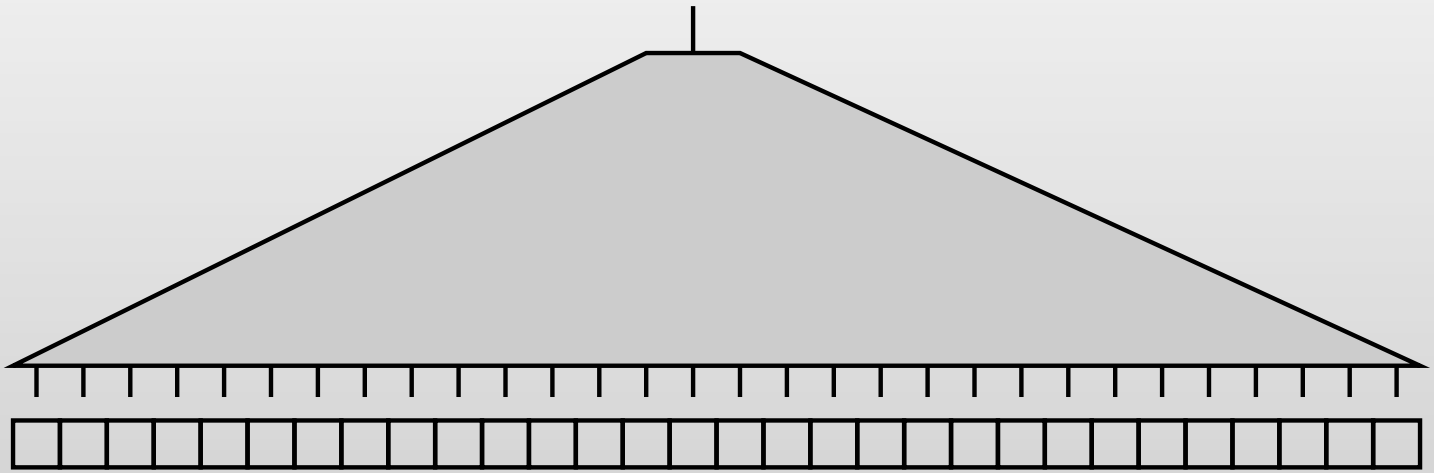


Limits of Standard Techniques



“to securely evaluate f,
first express f as a boolean circuit,
then …”

Limits of Standard Techniques



“to securely evaluate f,
first express f as a boolean circuit,
then …”

What We're Up Against

1. Security requires protocol cost at least **linear in size of inputs** (in general!)

What We're Up Against

1. Security requires protocol cost at least **linear in size of inputs** (in general!)
2. General-purpose 2PC scales with size of **circuit representation**, which is always at least linear in input size.

In this talk:

1. Instead of circuits, use a representation that can actually be sublinear in size.

In this talk:

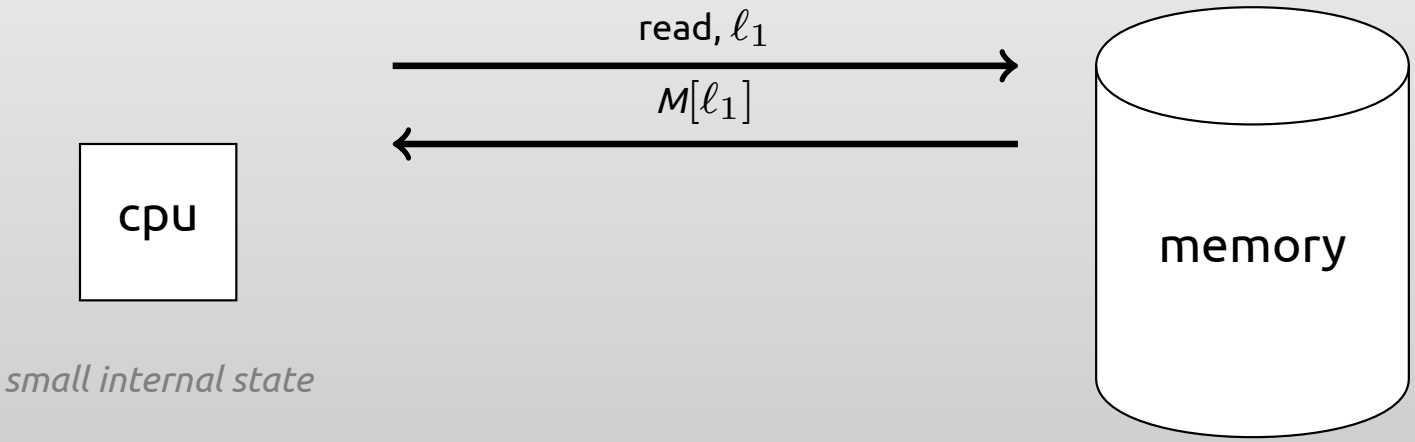
1. Instead of circuits, use a representation that can actually be sublinear in size.

2. Protocol must “touch every bit”, but amortize this cost across many executions.

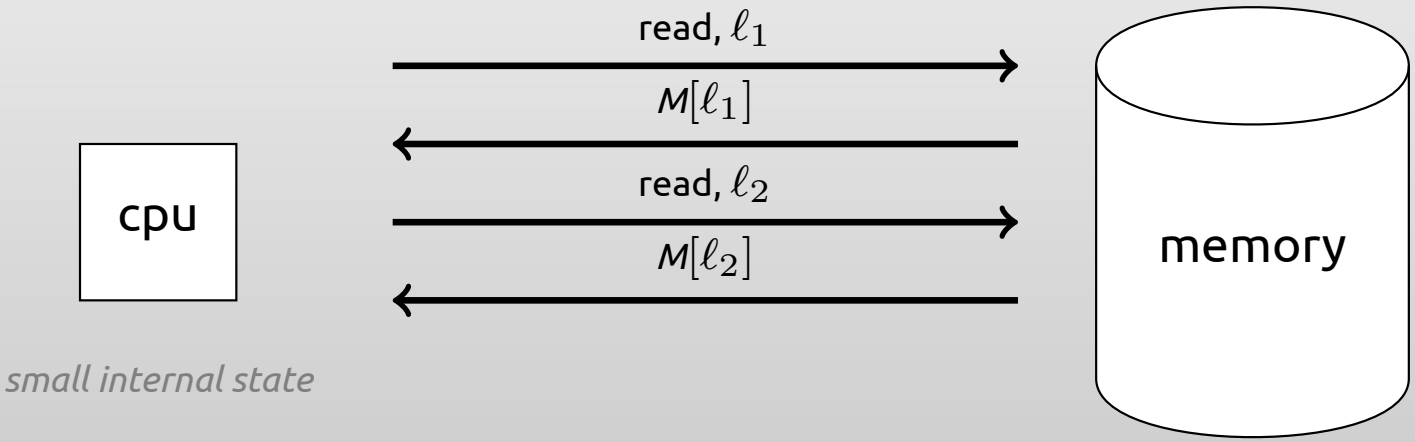
RAM programs

small internal state

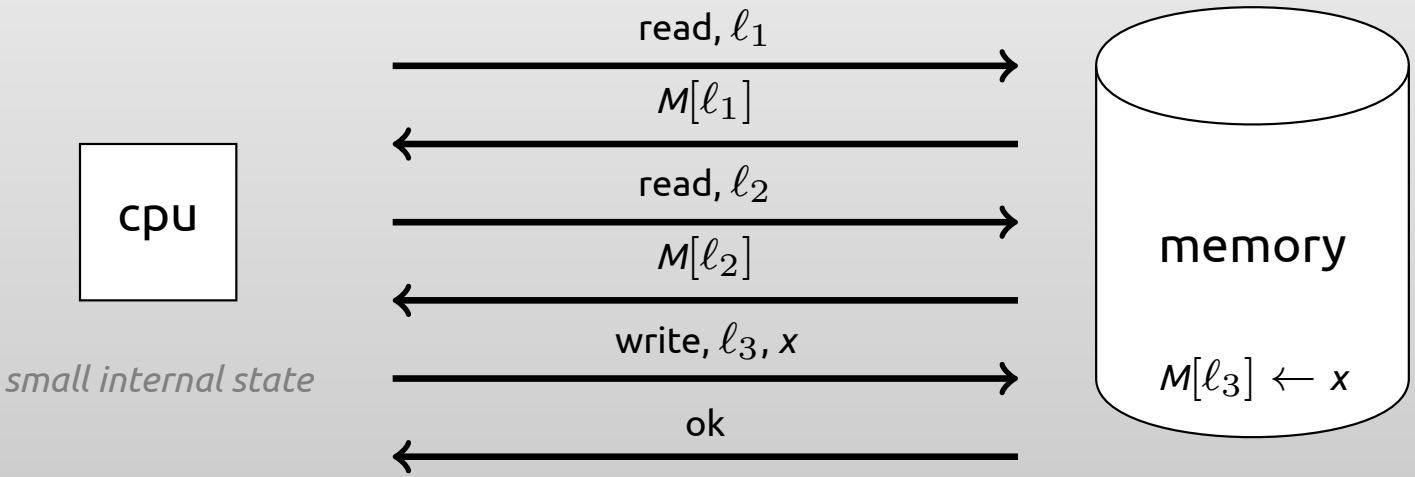
RAM programs



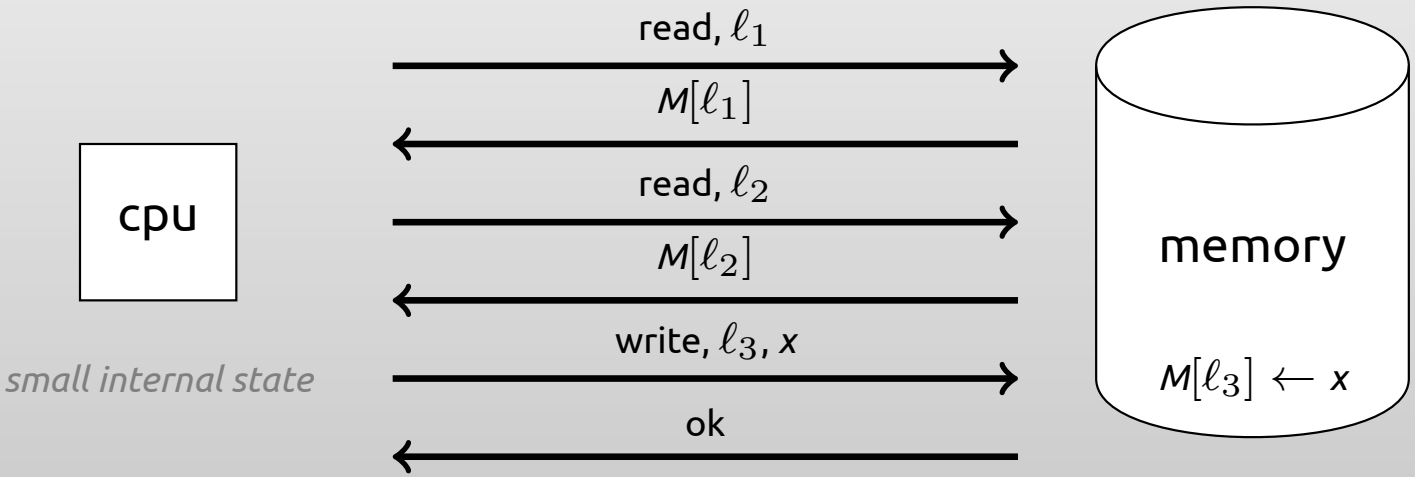
RAM programs



RAM programs

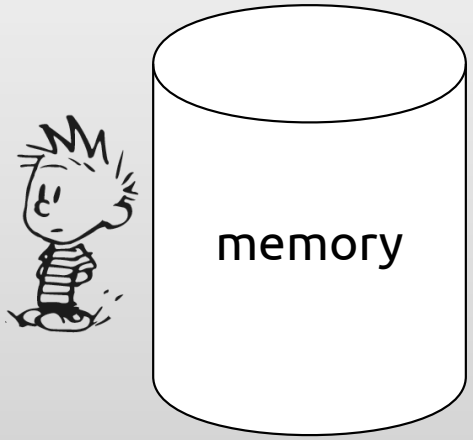


RAM programs



RAM program need not touch every bit of memory.

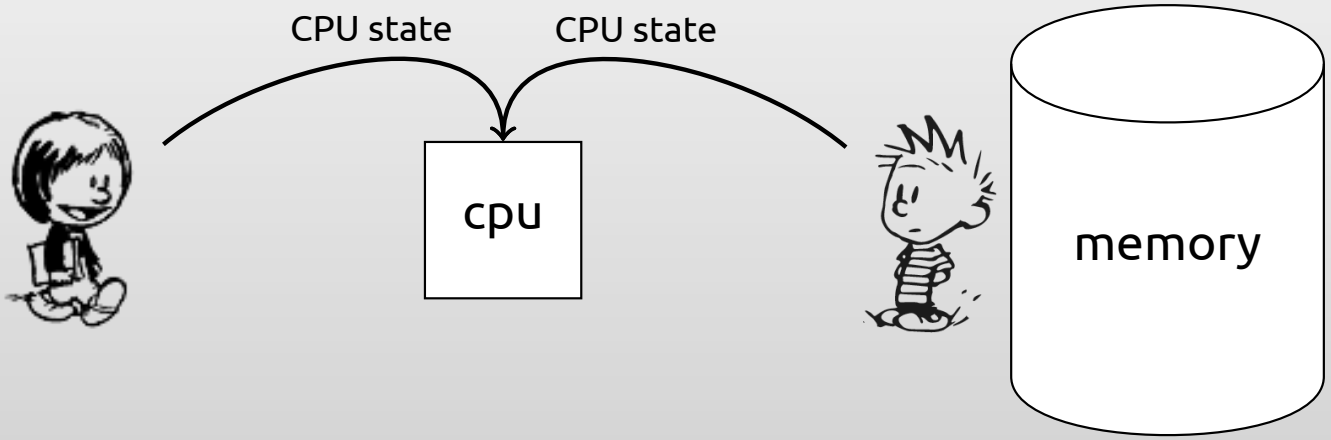
Idea: securely evaluate RAM



Basic outline:

- Imagine both parties' inputs stored in large memory

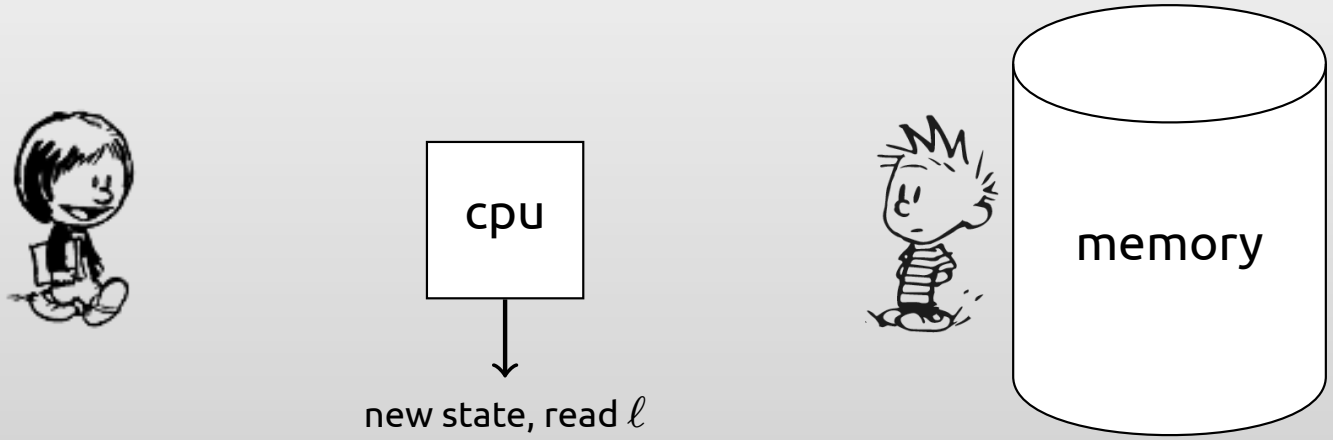
Idea: securely evaluate RAM



Basic outline:

- Imagine both parties' inputs stored in large memory
- Imagine they could evaluate CPU-next-instruction function

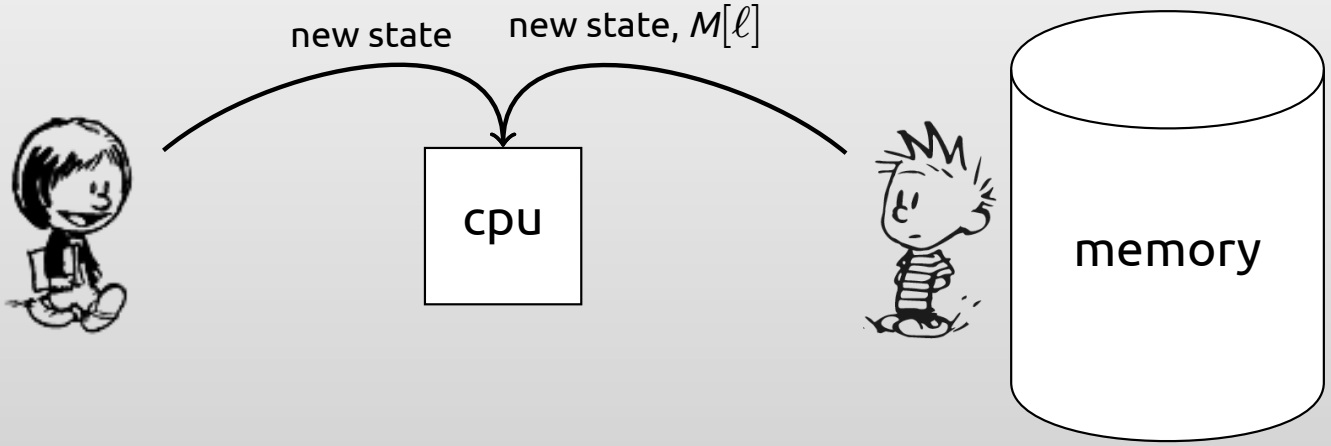
Idea: securely evaluate RAM



Basic outline:

- Imagine both parties' inputs stored in large memory
- Imagine they could evaluate CPU-next-instruction function

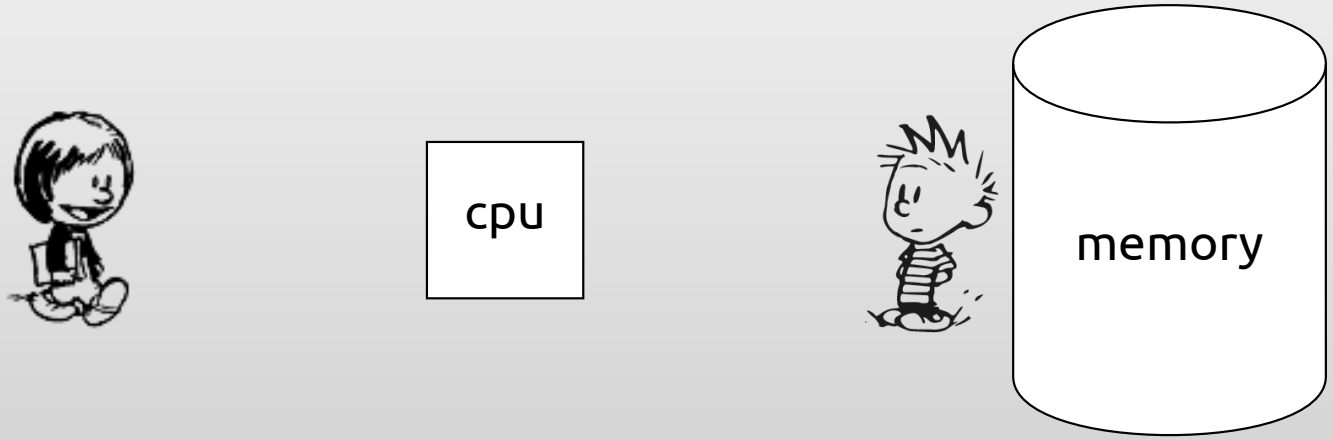
Idea: securely evaluate RAM



Basic outline:

- Imagine both parties' inputs stored in large memory
- Imagine they could evaluate CPU-next-instruction function

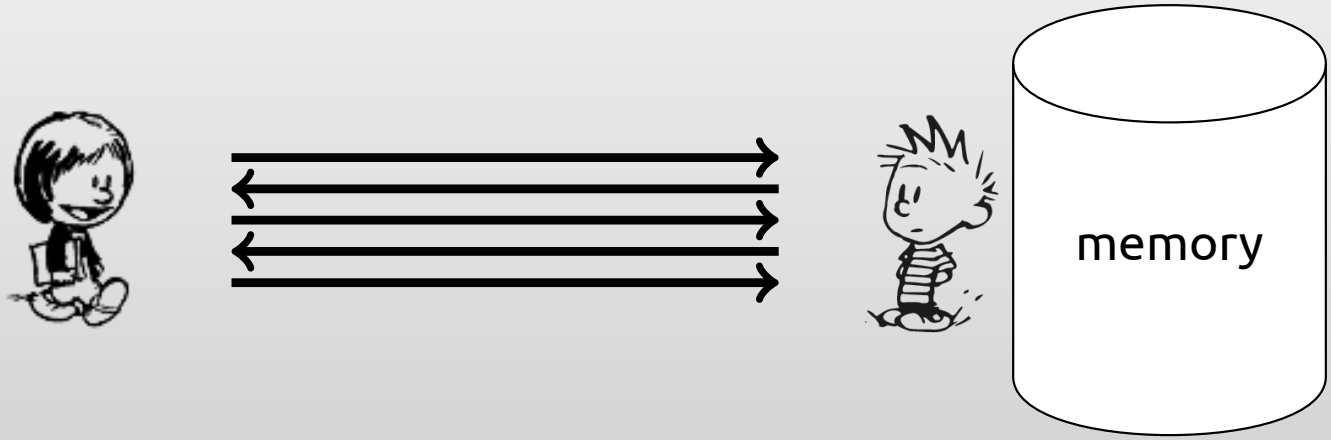
Idea: securely evaluate RAM



Basic outline:

- Imagine both parties' inputs stored in large memory
- Imagine they could evaluate CPU-next-instruction function
- Use (traditional) 2PC protocol to realize CPU-next-instruction

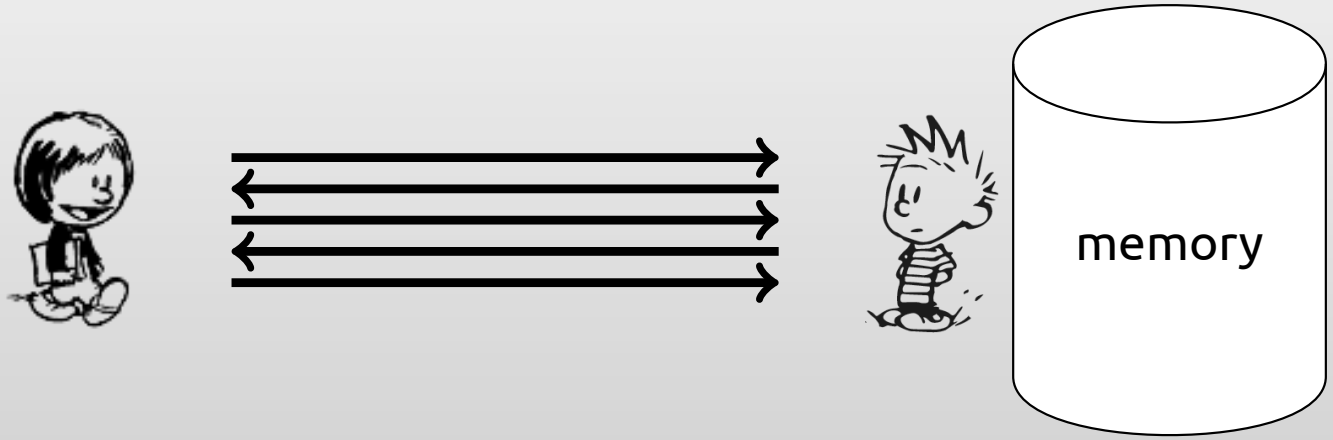
Idea: securely evaluate RAM



Basic outline:

- Imagine both parties' inputs stored in large memory
- Imagine they could evaluate CPU-next-instruction function
- Use (traditional) 2PC protocol to realize CPU-next-instruction

Idea: securely evaluate RAM

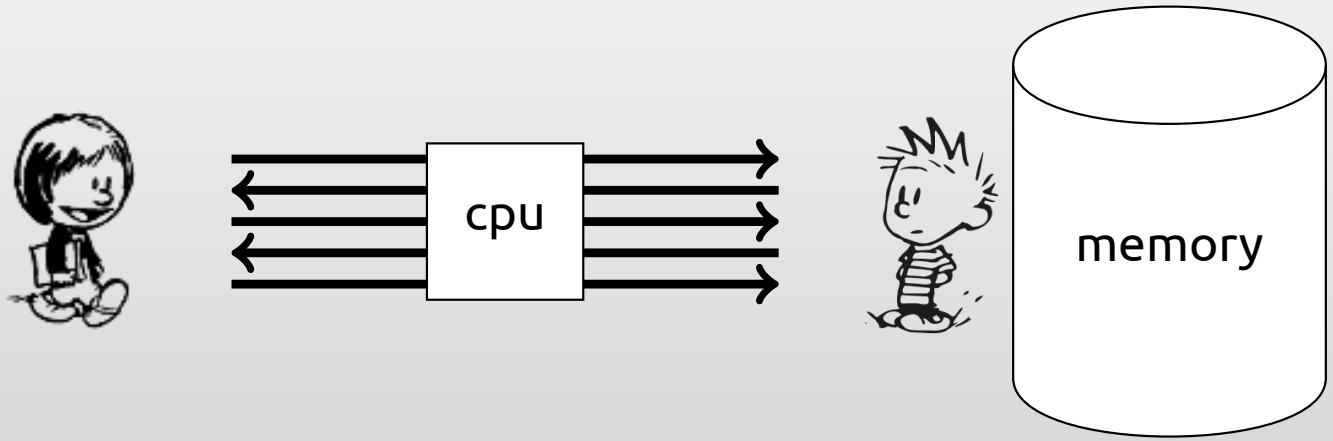


Basic outline:

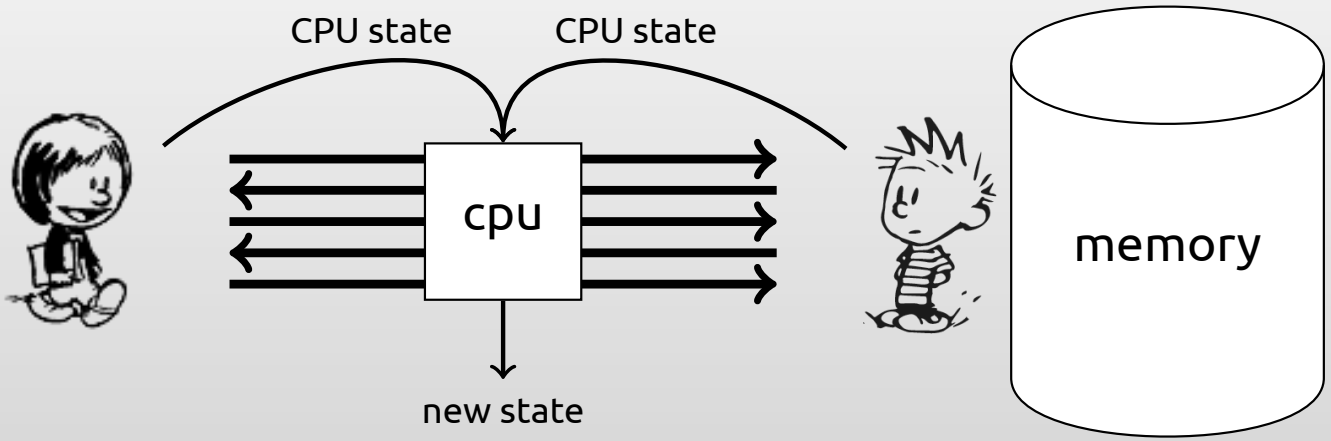
- Imagine both parties' inputs stored in large memory
- Imagine they could evaluate CPU-next-instruction function
- Use (traditional) 2PC protocol to realize CPU-next-instruction

Cost = (size of next-instruction function) $\times$ (number of instructions)

What can go wrong?

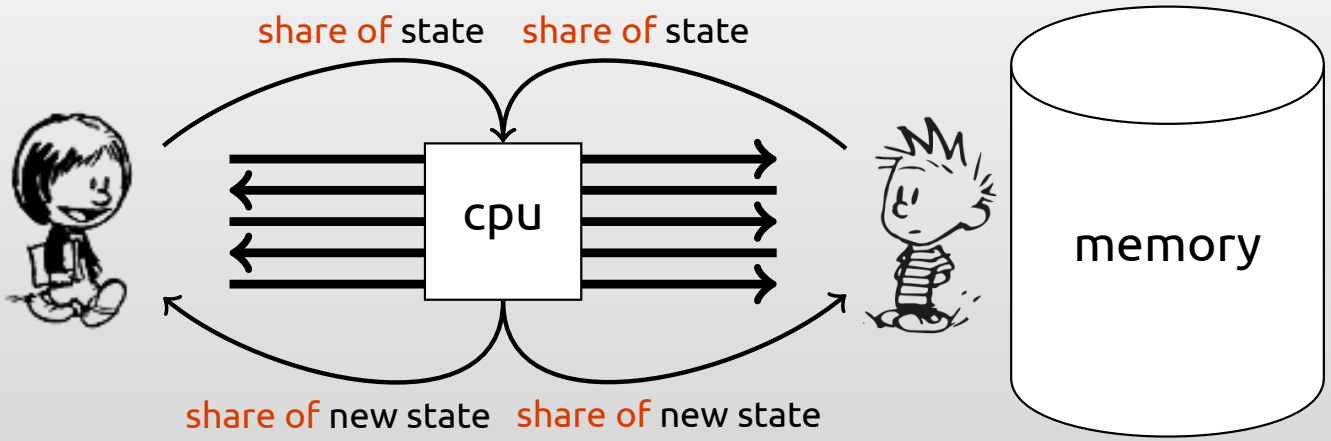


What can go wrong?



Internal state is public

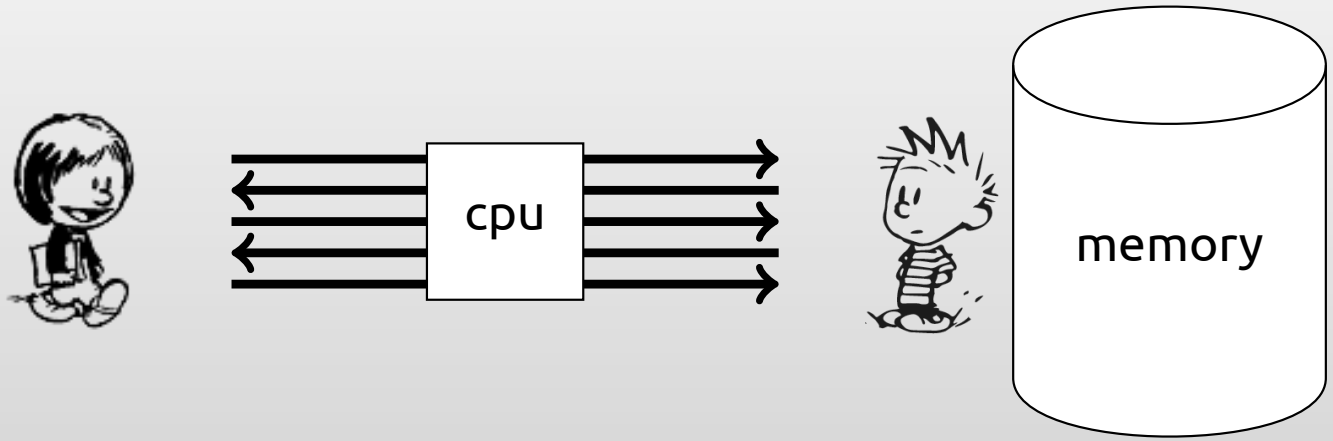
What can go wrong?



Internal state is public

⇒ Secret-share the state! ✓

What can go wrong?

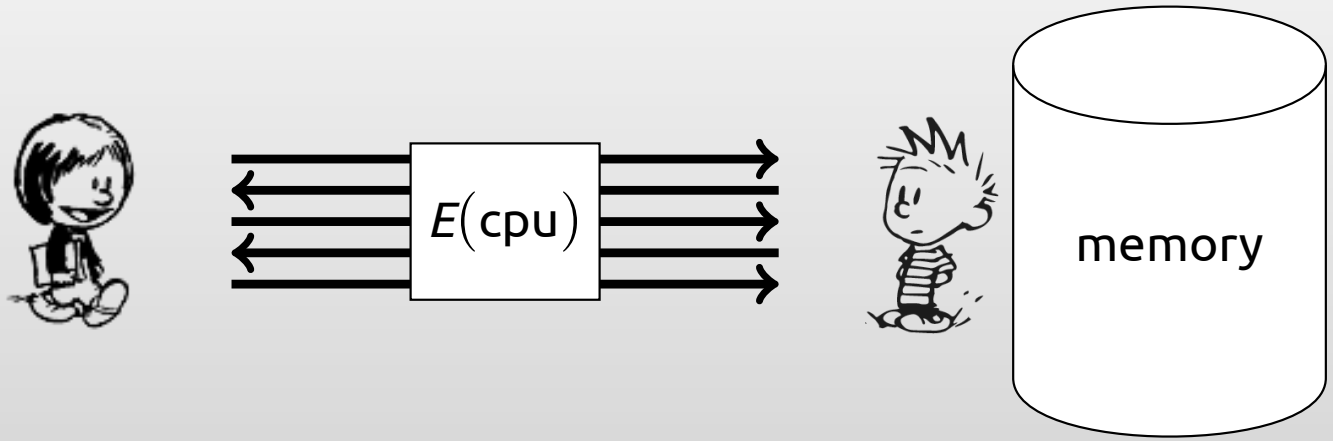


Internal state is public

⇒ Secret-share the state! ✓

Calvin sees all of the memory

What can go wrong?



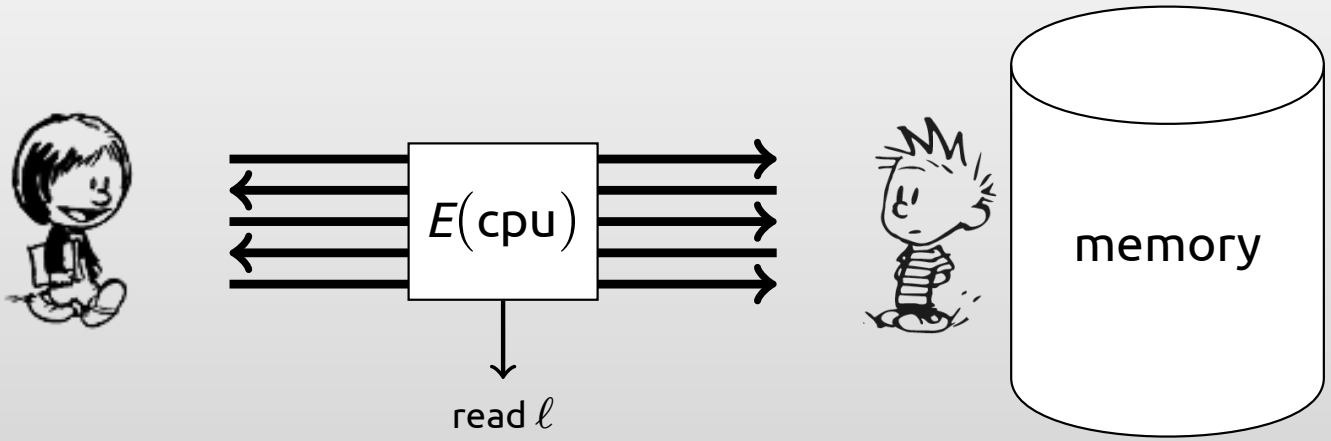
Internal state is public

⇒ Secret-share the state! ✓

Calvin sees all of the memory

⇒ Encrypt the memory, augment CPU-next-instruction with encryption/decryption. ✓

What can go wrong?



Internal state is public

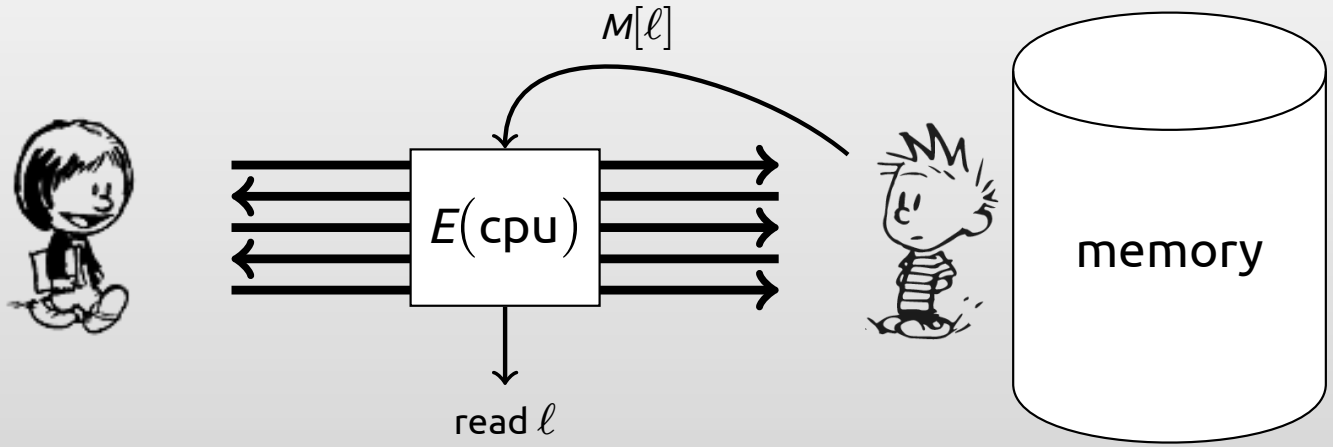
⇒ Secret-share the state! ✓

Calvin sees all of the memory

⇒ Encrypt the memory, augment CPU-next-instruction with encryption/decryption. ✓

Memory access pattern (read ℓ_1, write ℓ_2, . . .) public!

What can go wrong?



Internal state is public

⇒ Secret-share the state! ✓

Calvin sees all of the memory

⇒ Encrypt the memory, augment CPU-next-instruction with encryption/decryption. ✓

Memory access pattern (read ℓ_1, write ℓ_2, . . .) public!

??? Calvin must learn these so he knows what to do!

Oblivious RAM

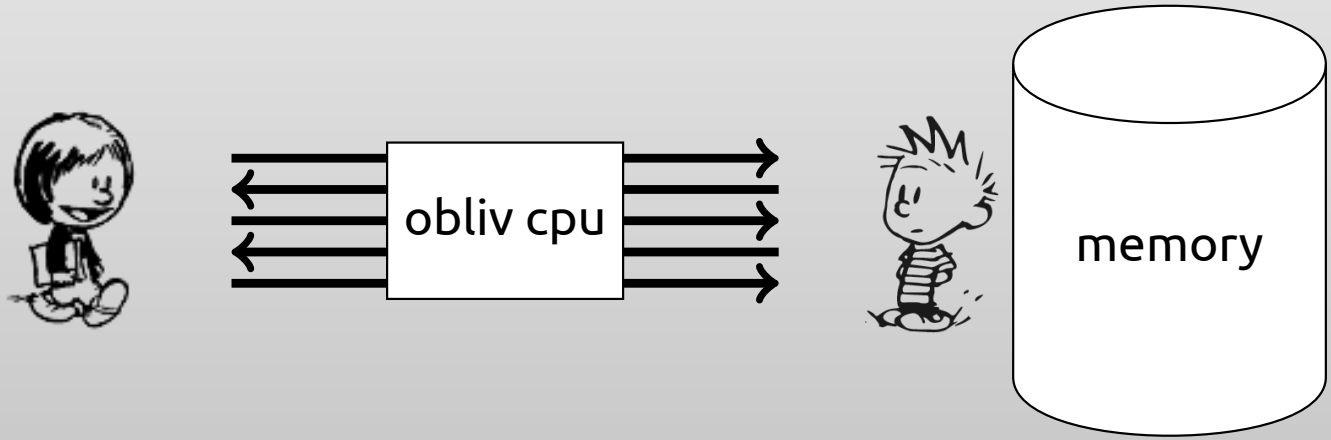
Oblivious RAM (ORAM) = memory access pattern leaks nothing about inputs/outputs/state [GoldreichOstrosvky96]

- Can convert any RAM program to ORAM, polylog overhead in runtime & memory [ShiChanStefanovLi11, ...]

Oblivious RAM

Oblivious RAM (ORAM) = memory access pattern leaks nothing about inputs/outputs/state [GoldreichOstrosvky96]

- Can convert any RAM program to ORAM, polylog overhead in runtime & memory [ShiChanStefanovLi11, ...]

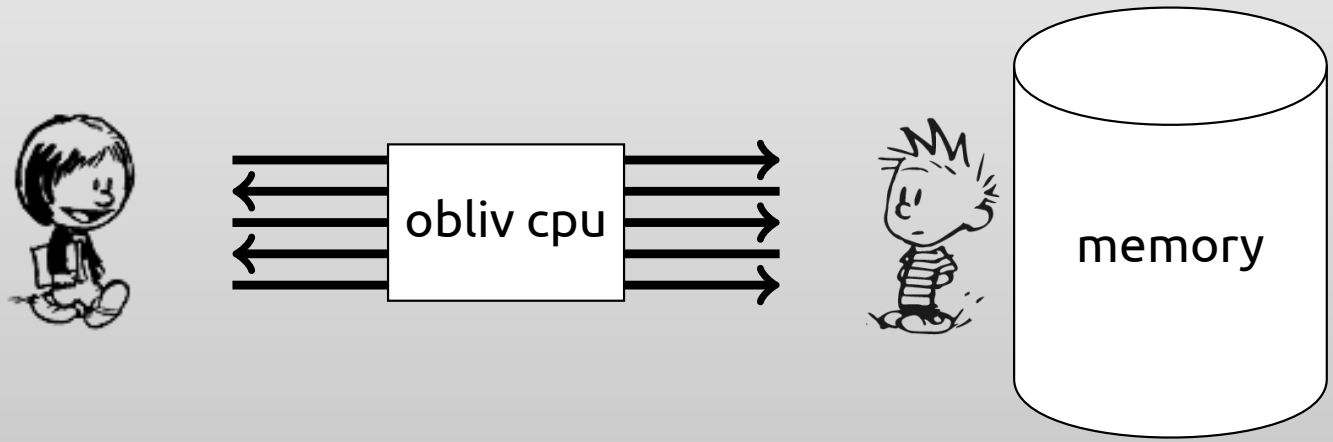


RAM-2PC paradigm [GKKKMRV12]

*"Use traditional 2PC to repeatedly evaluate next-instruction circuit of an **oblivious** RAM program."*

Wait, what?

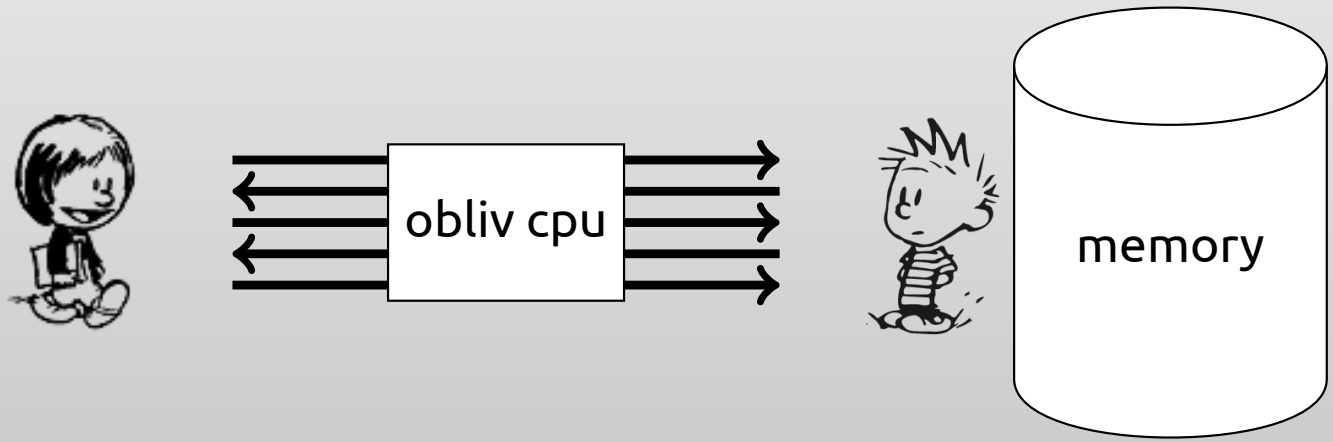
If original RAM program is sublinear, ORAM version is sublinear too!



Wait, what?

If original RAM program is sublinear, ORAM version is sublinear too!

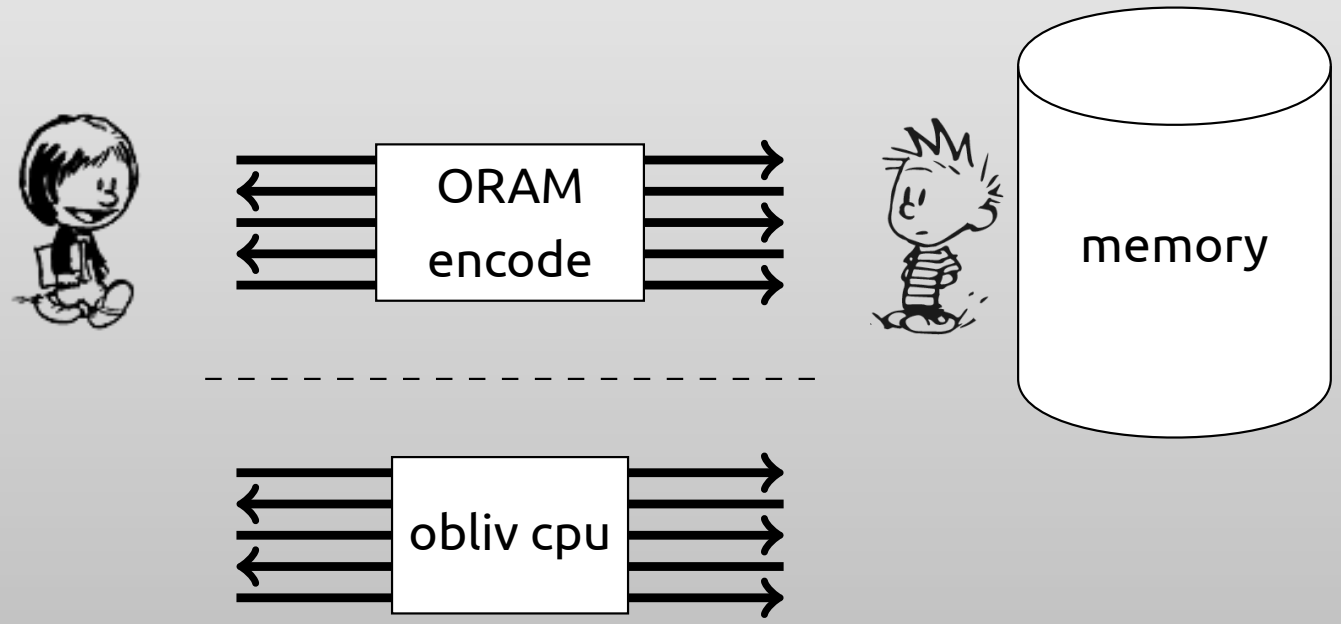
. . . *only after memory is initialized into proper data structure!*



Wait, what?

If original RAM program is sublinear, ORAM version is sublinear too!

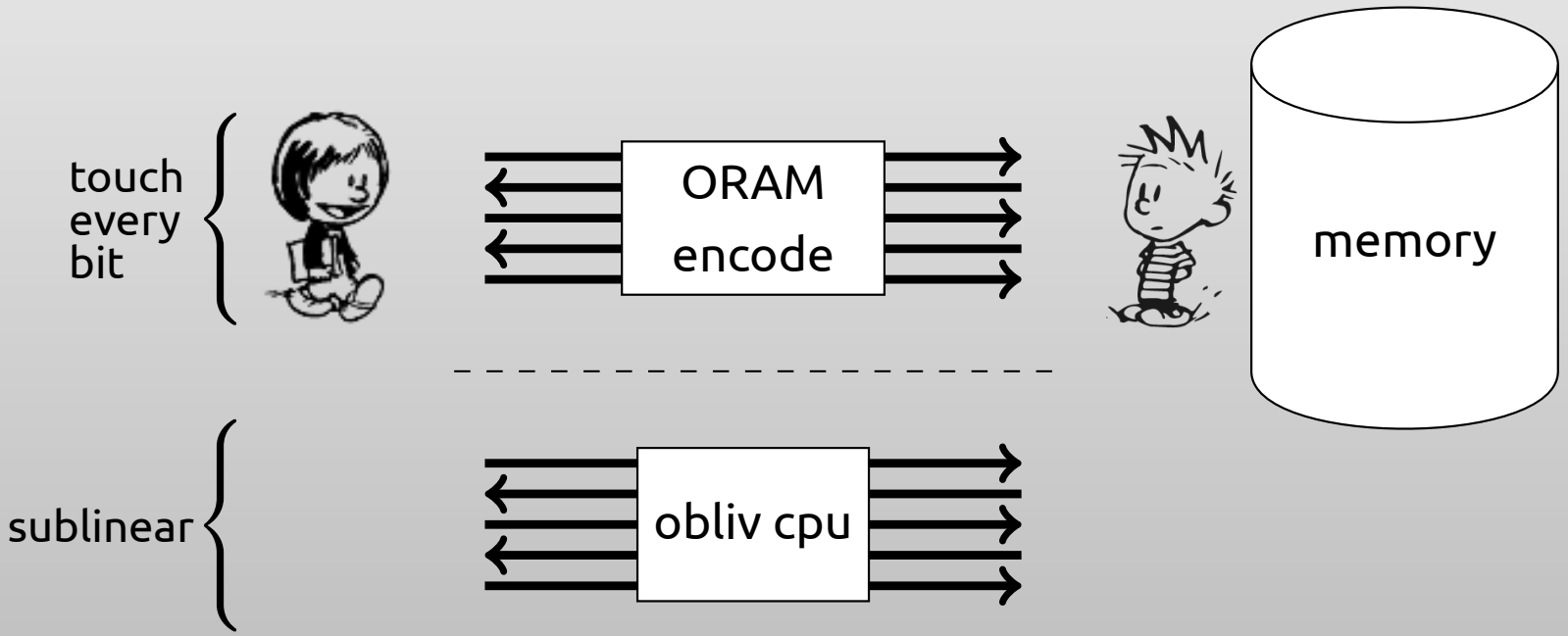
. . . *only after memory is initialized into proper data structure!*



Wait, what?

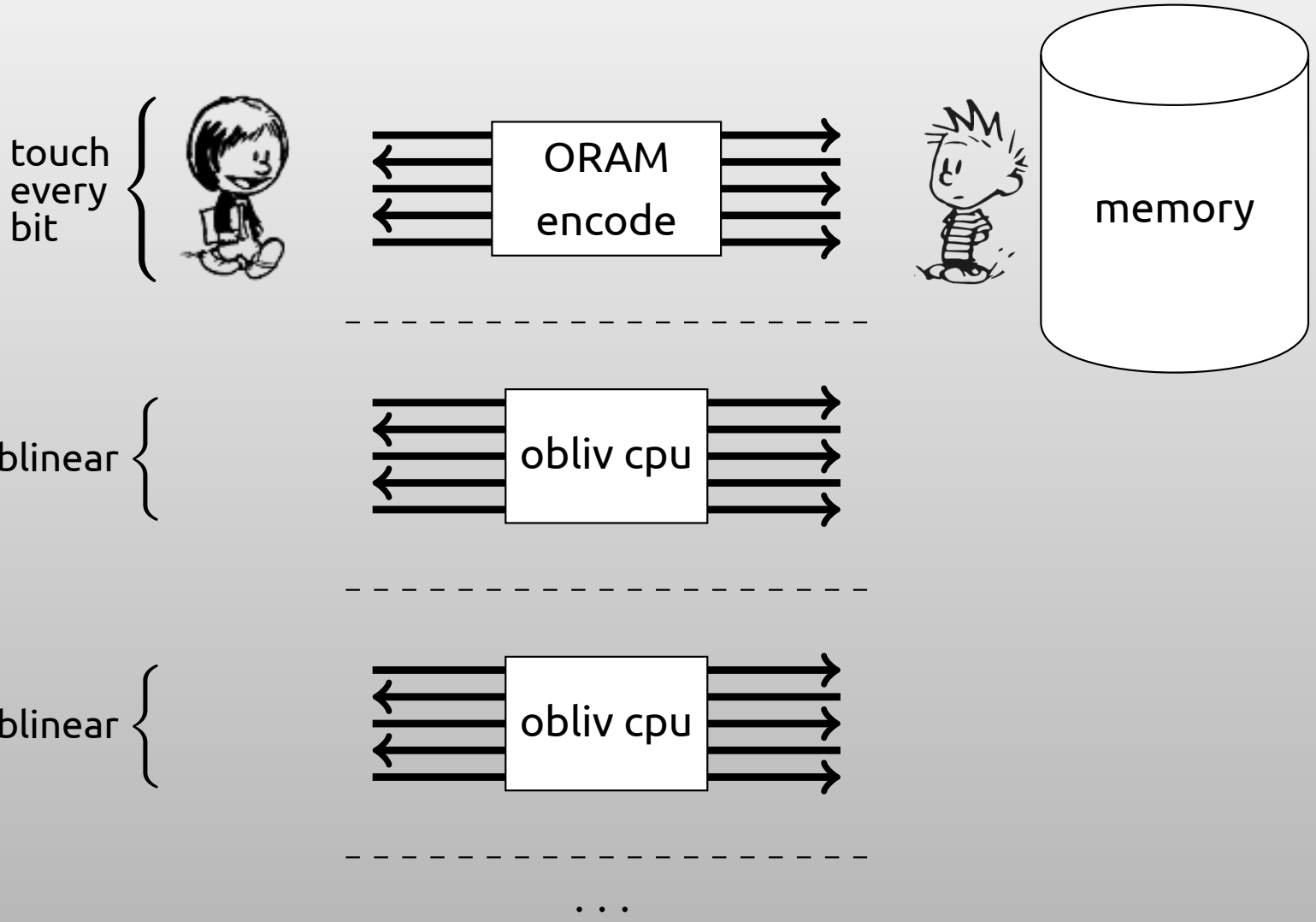
If original RAM program is sublinear, ORAM version is sublinear too!

. . . *only after memory is initialized into proper data structure!*



Amortizing

ORAM memory can be reused indefinitely



Summarizing

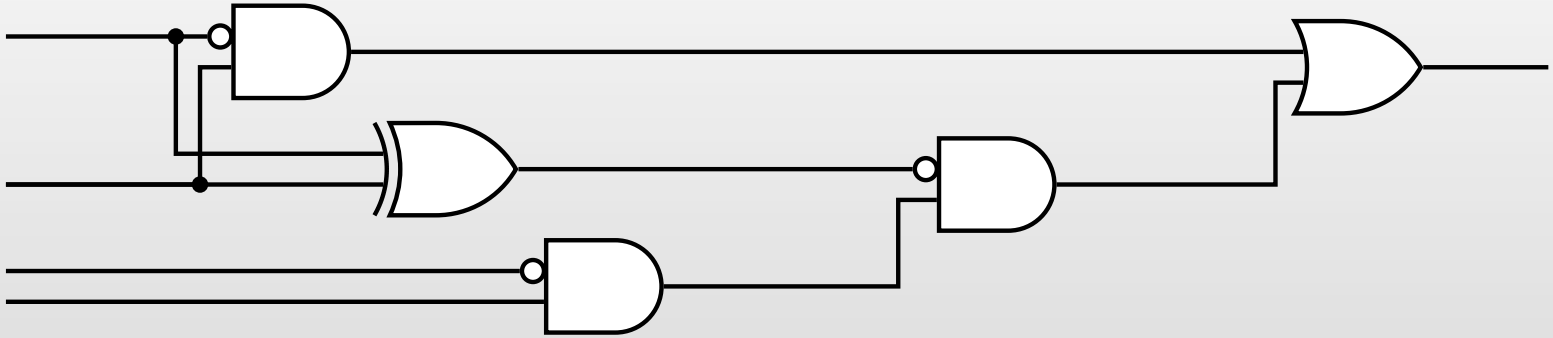
RAM-2PC paradigm [GKKKMRV12]

*"Use traditional 2PC to repeatedly evaluate next-instruction circuit of an **oblivious** RAM program."*

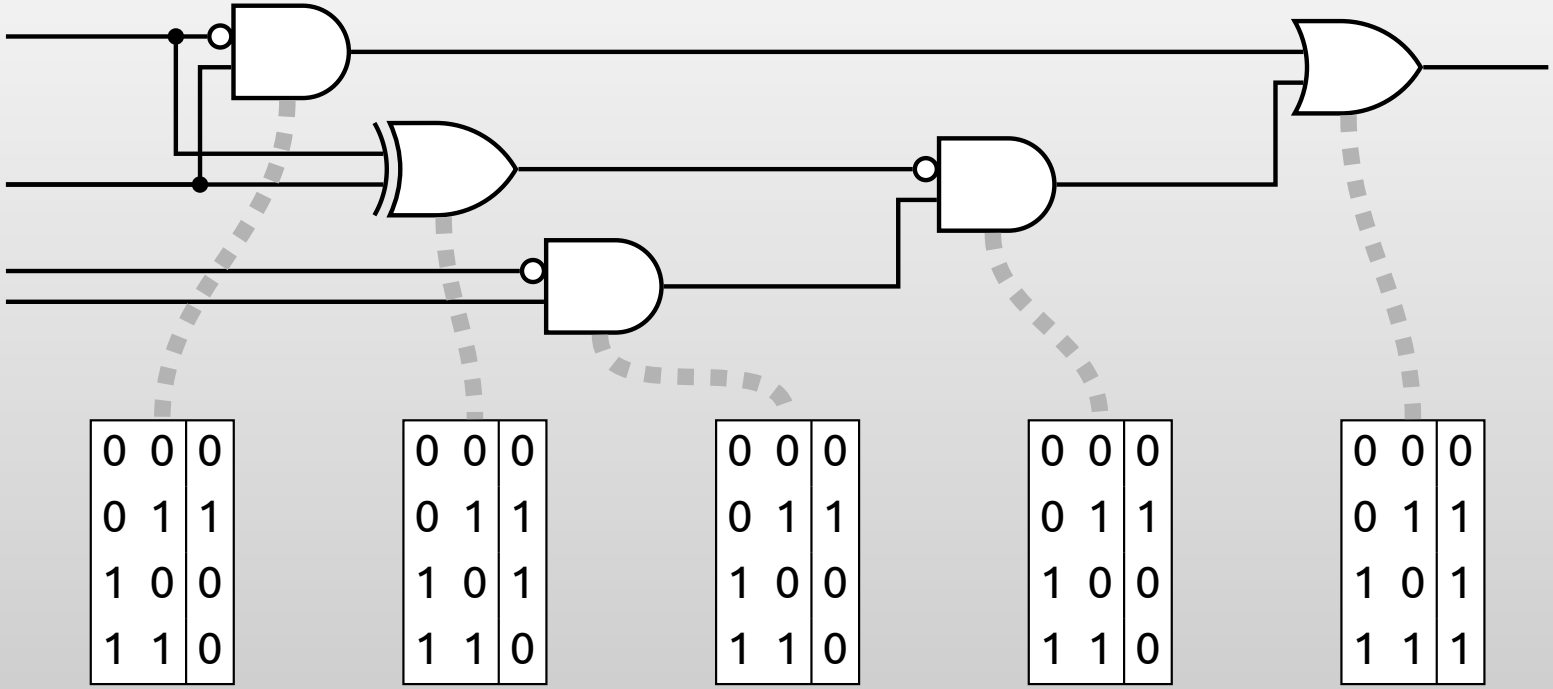
- Expensive $O(N)$ initialization phase
- Subsequent computations cost $\widetilde{O}(T)$, where $T =$ ORAM running time.

- [GKKKMRV12]: semi-honest security
- [AfsharHuMohasselR15]: malicious security
- [HuMohasselR15]: malicious security, one-sided privacy

Garbled circuit framework [Yao86]



Garbled circuit framework [Yao86]



0	0	0
0	1	1
1	0	0
1	1	0

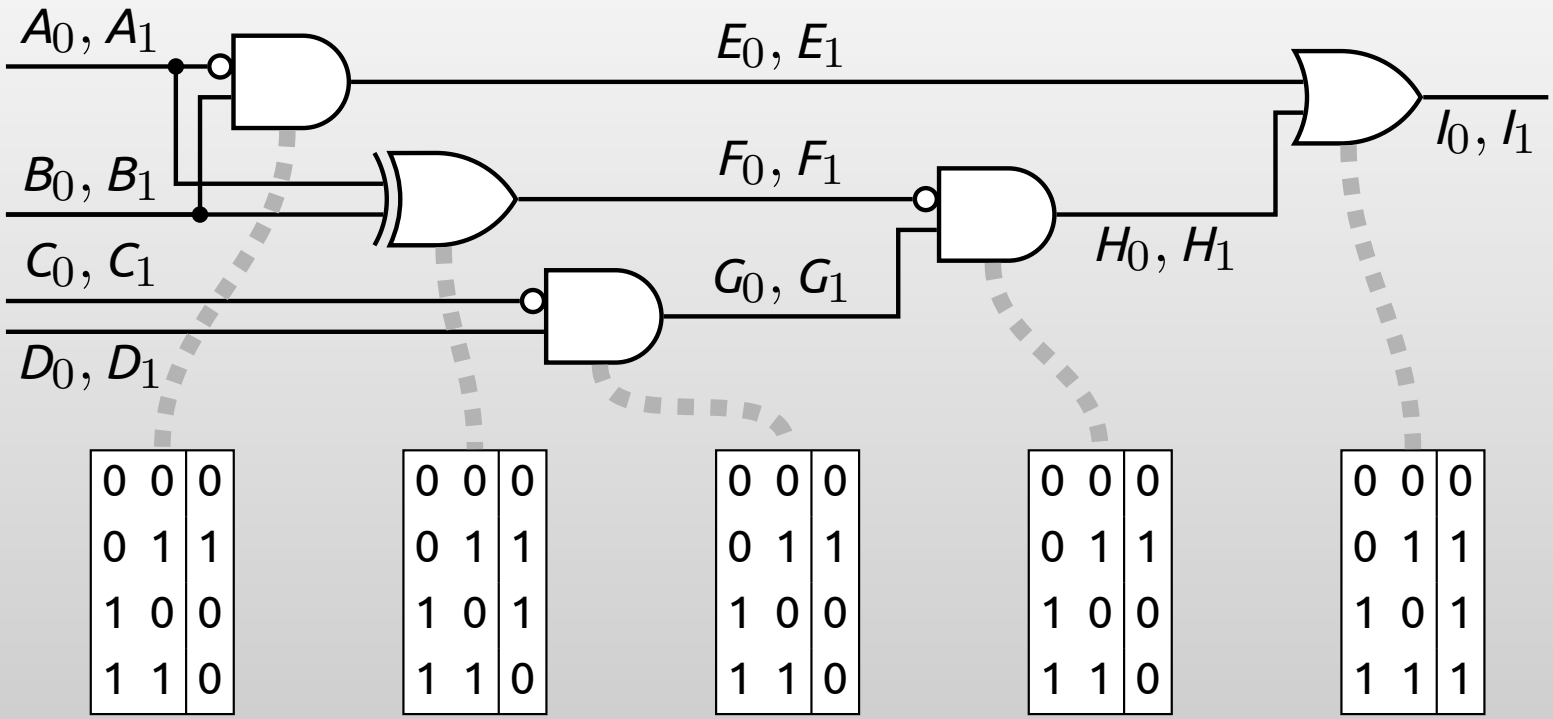
0	0	0
0	1	1
1	0	1
1	1	0

0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	1
1	1	1

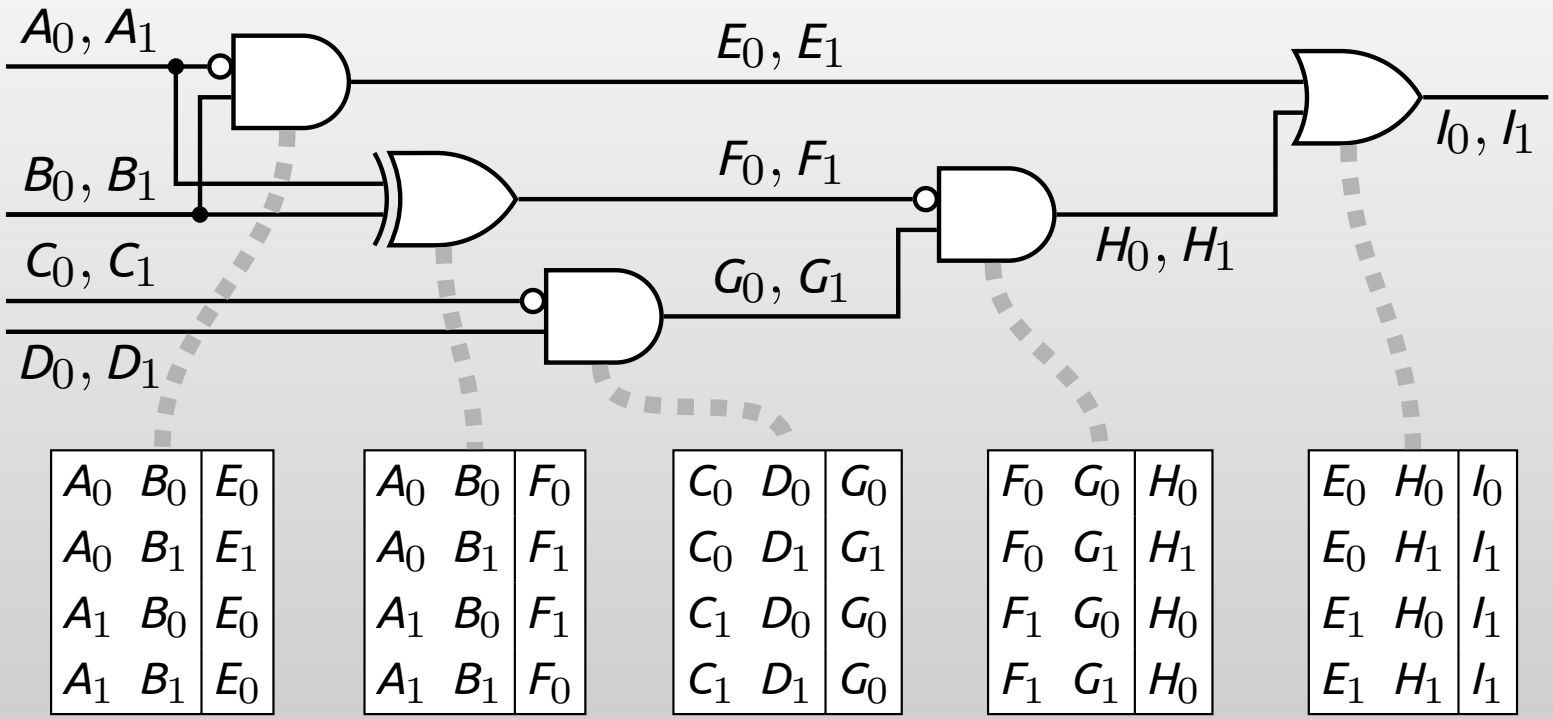
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire

Garbled circuit framework [Yao86]



A_0	B_0	E_0
A_0	B_1	E_1
A_1	B_0	E_0
A_1	B_1	E_0

A_0	B_0	F_0
A_0	B_1	F_1
A_1	B_0	F_1
A_1	B_1	F_0

C_0	D_0	G_0
C_0	D_1	G_1
C_1	D_0	G_0
C_1	D_1	G_0

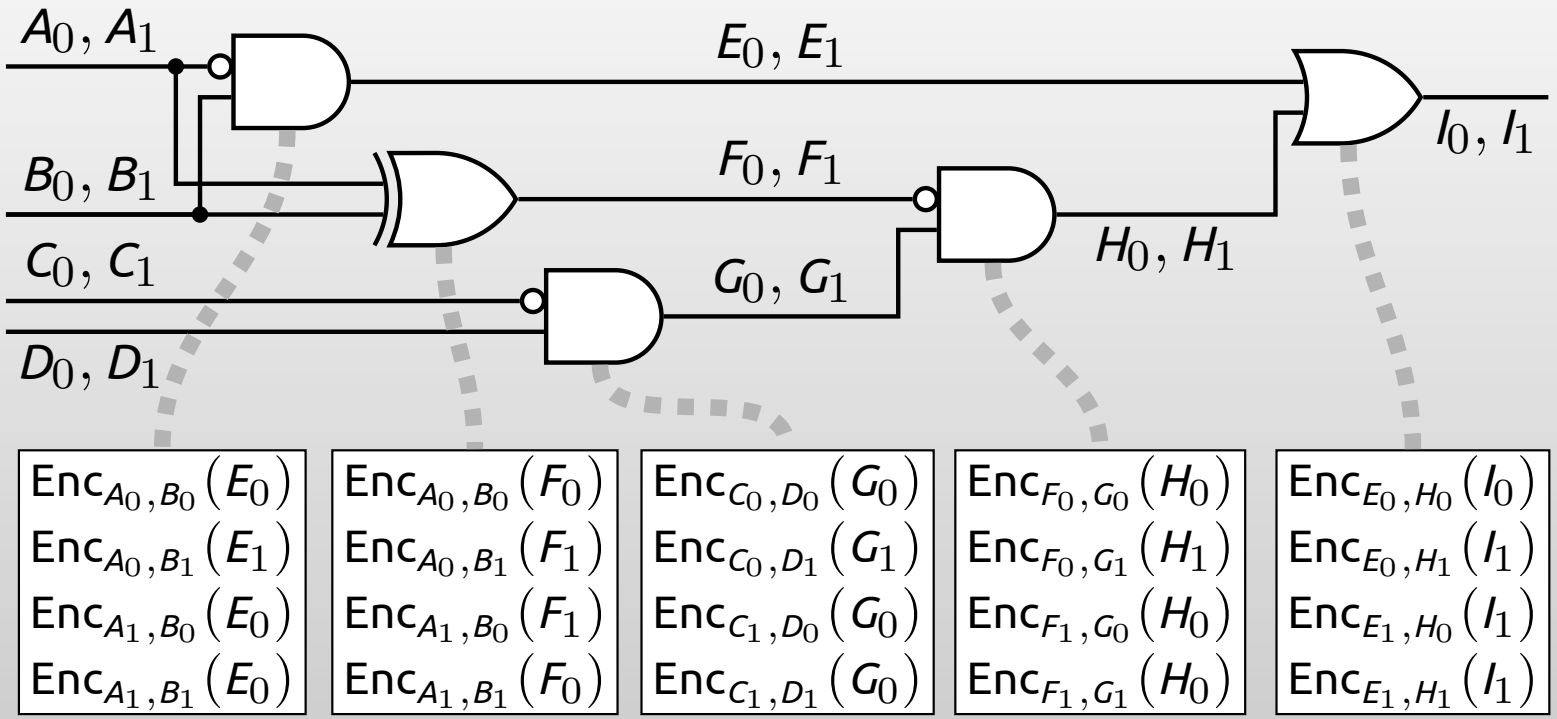
F_0	G_0	H_0
F_0	G_1	H_1
F_1	G_0	H_0
F_1	G_1	H_0

E_0	H_0	I_0
E_0	H_1	I_1
E_1	H_0	I_1
E_1	H_1	I_1

Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire

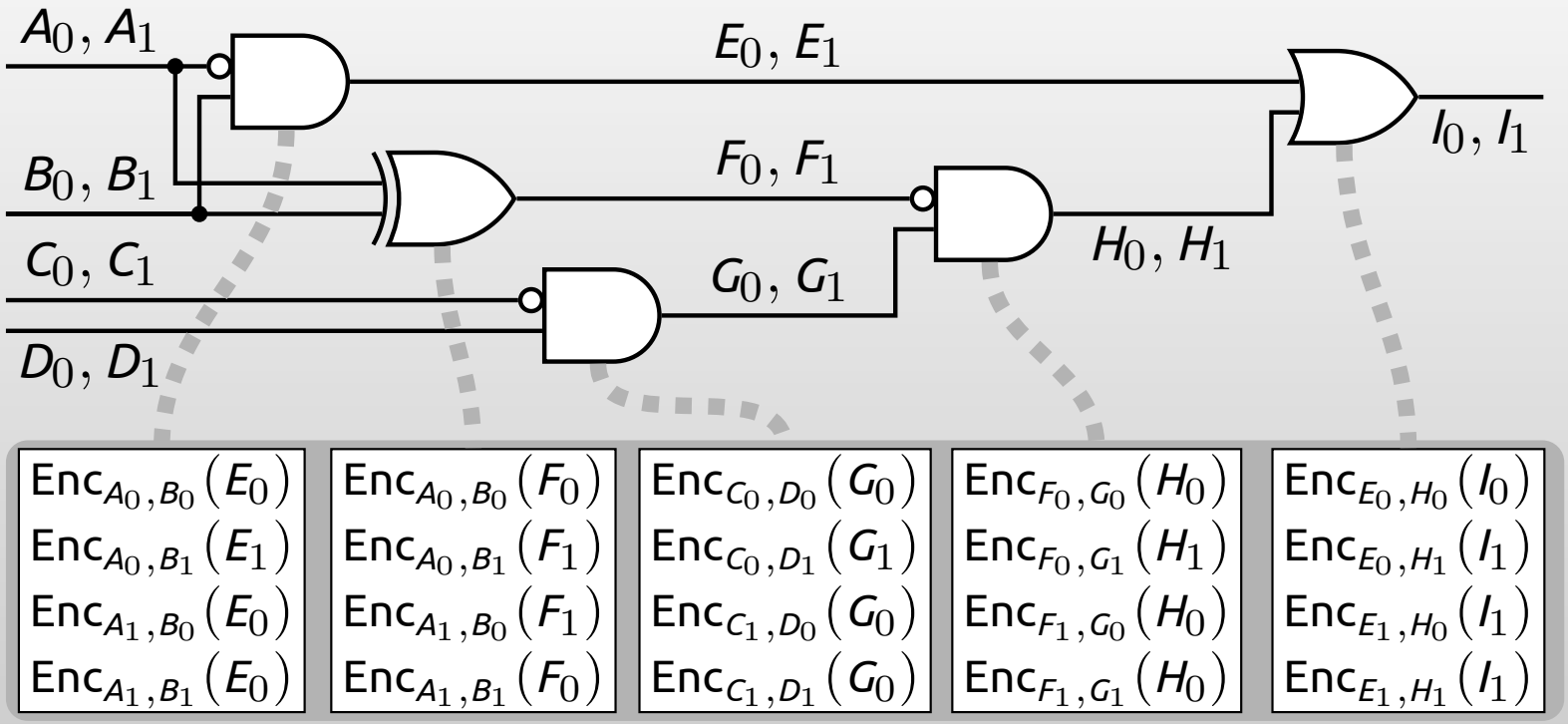
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- “Encrypt” truth table of each gate

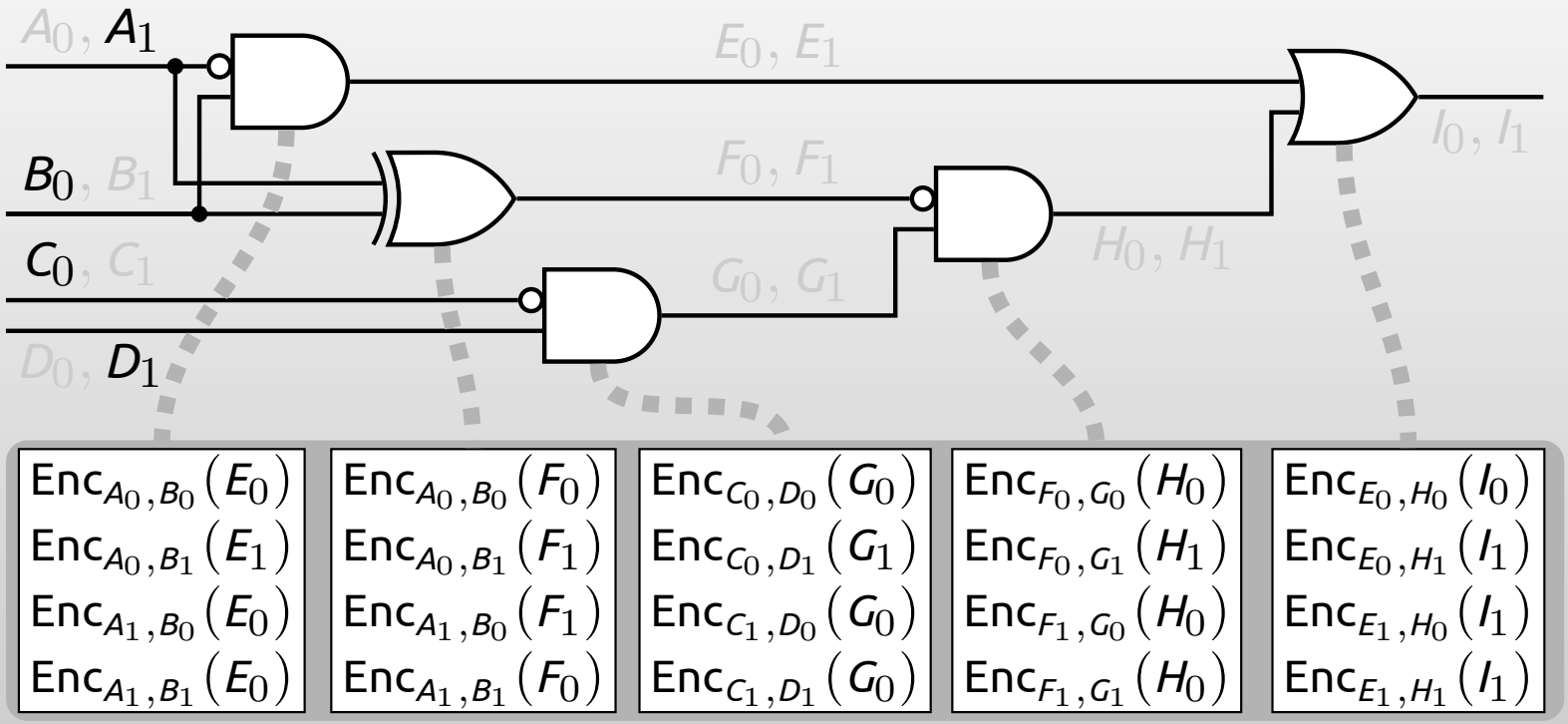
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** $\equiv$ all encrypted gates

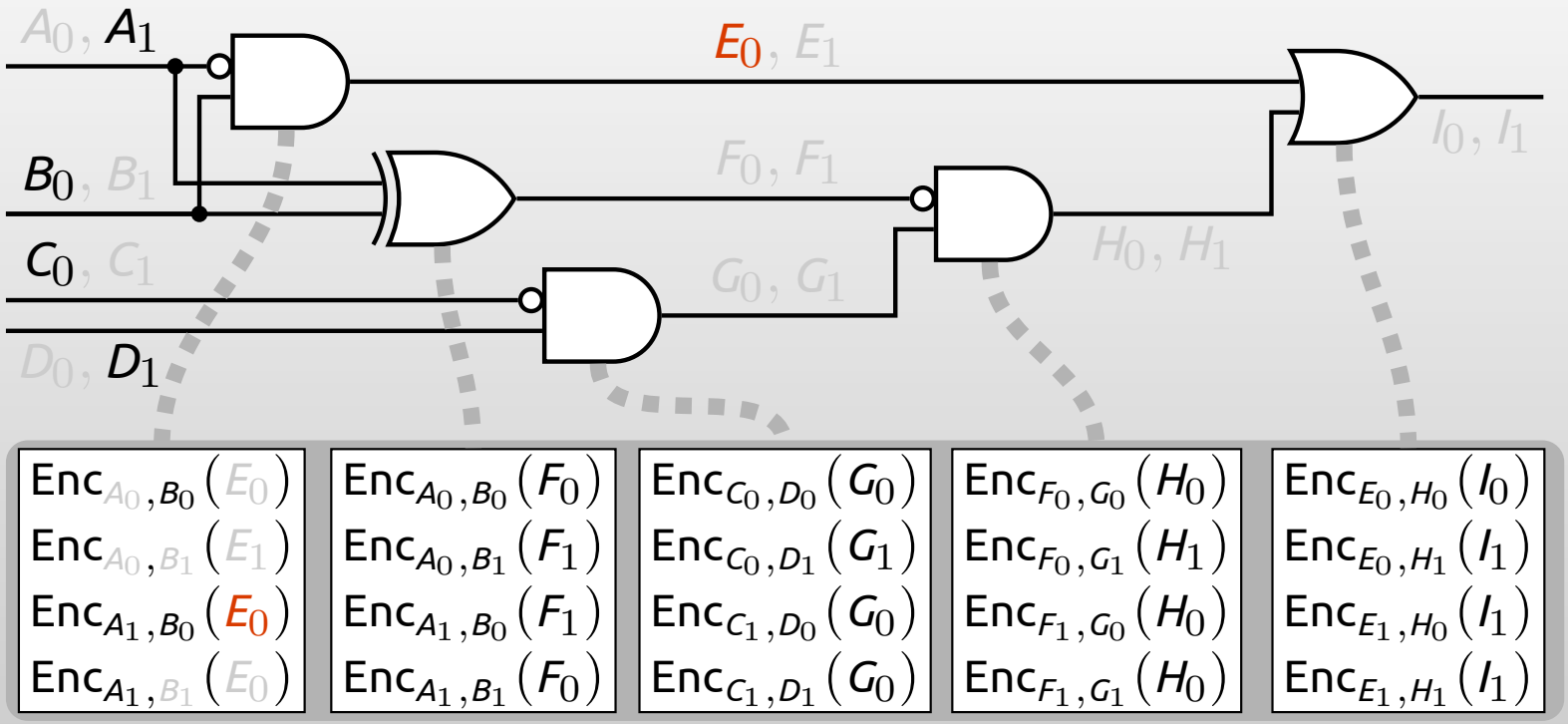
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** $\equiv$ all encrypted gates
- **Garbled encoding** $\equiv$ one label per wire

Garbled circuit framework [Yao86]



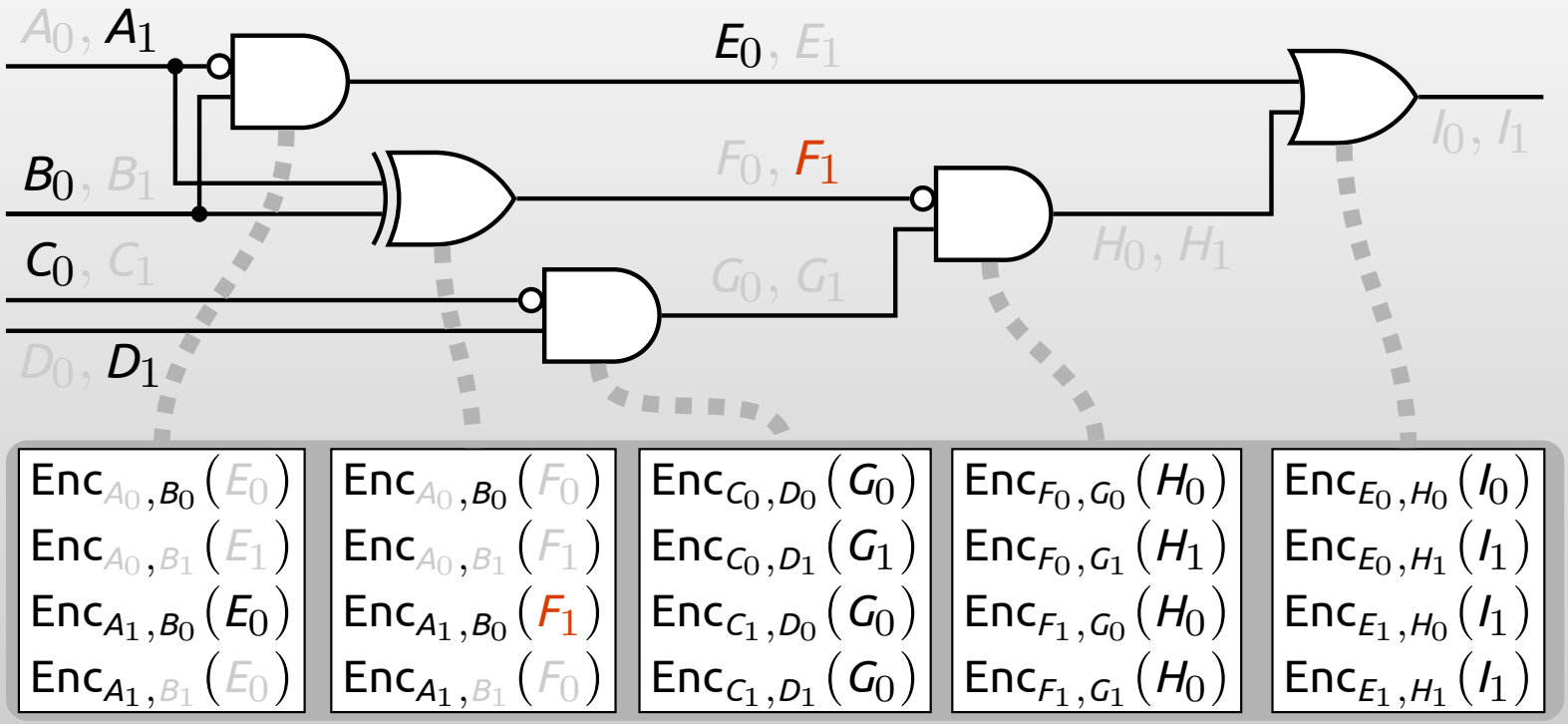
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** $\equiv$ all encrypted gates
- **Garbled encoding** $\equiv$ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable

Garbled circuit framework [Yao86]



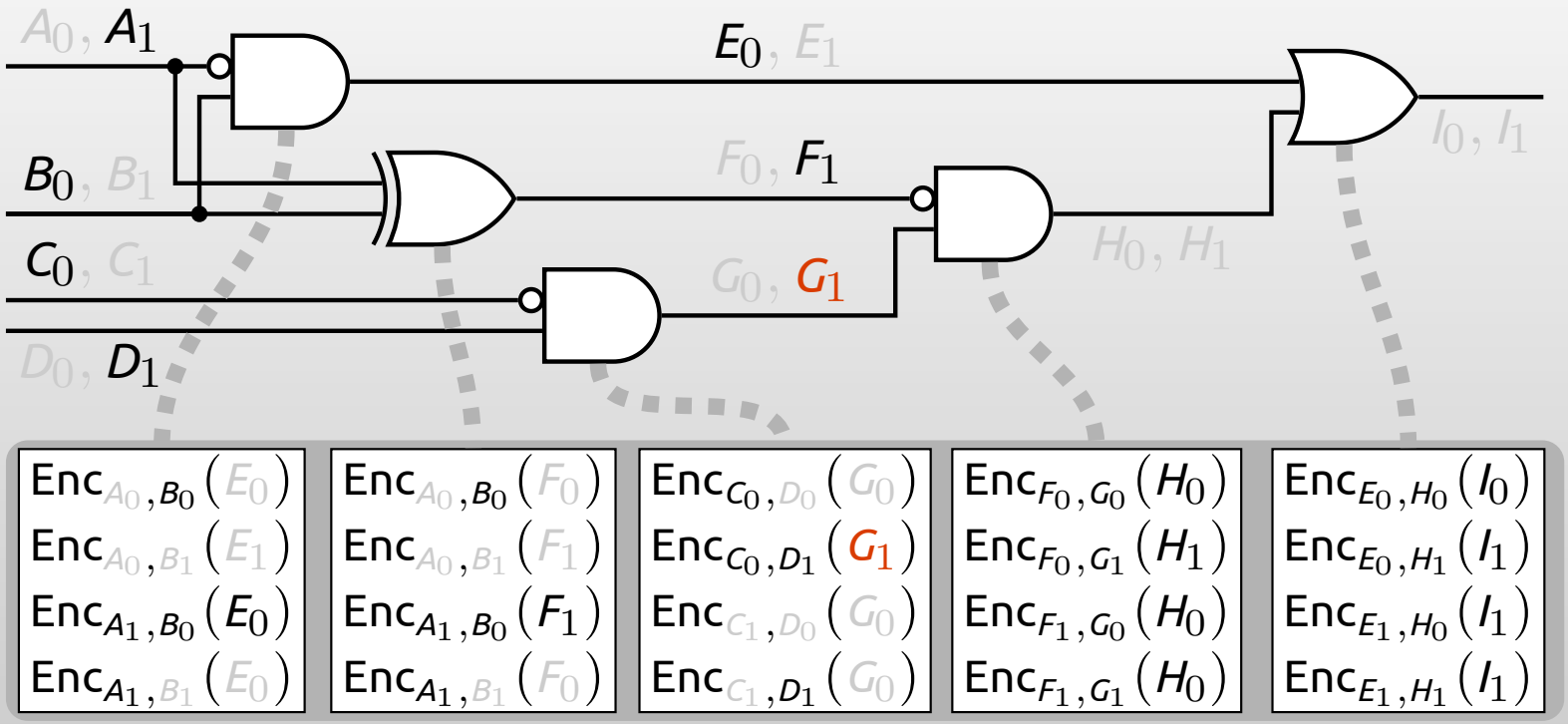
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** $\equiv$ all encrypted gates
- **Garbled encoding** $\equiv$ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuit framework [Yao86]



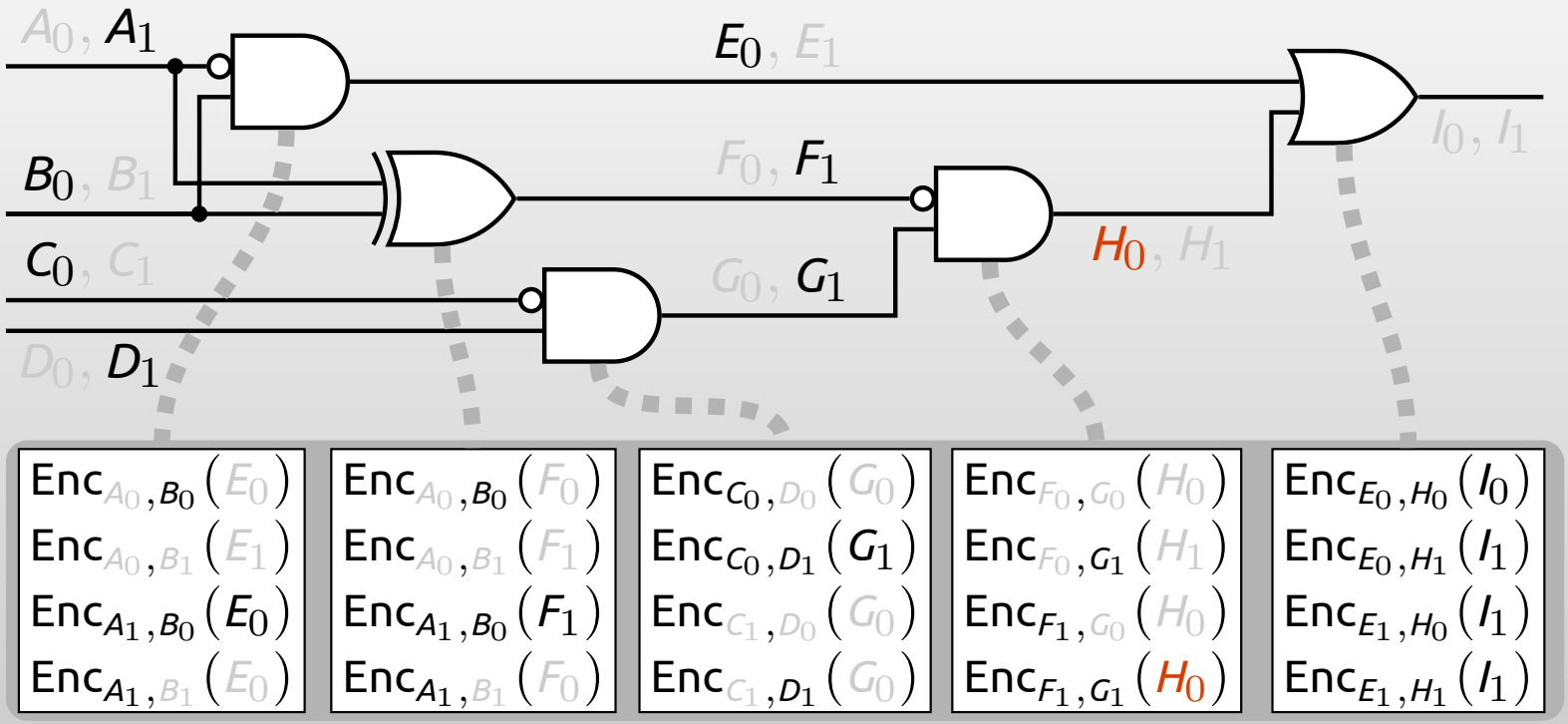
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** $\equiv$ all encrypted gates
- **Garbled encoding** $\equiv$ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuit framework [Yao86]



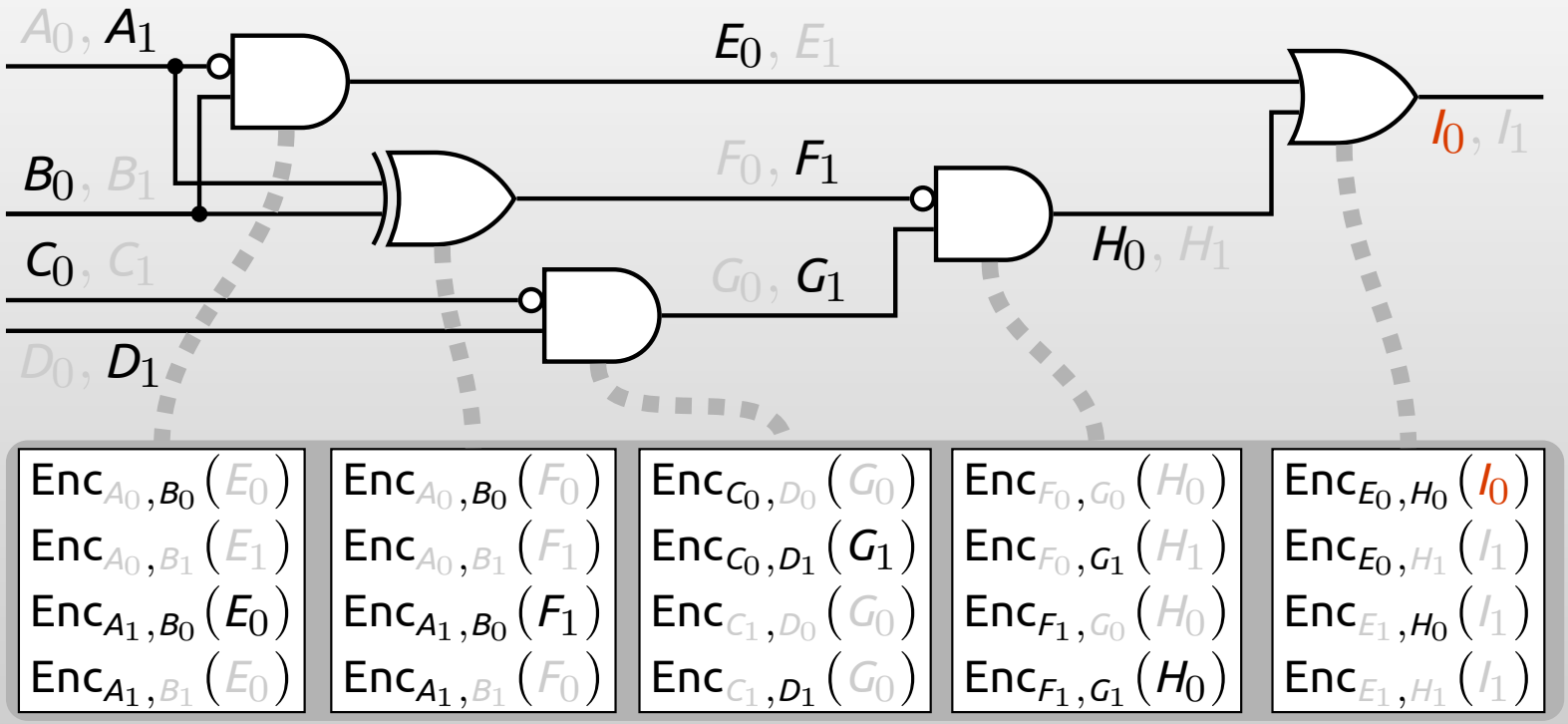
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** $\equiv$ all encrypted gates
- **Garbled encoding** $\equiv$ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuit framework [Yao86]



Garbling a circuit:

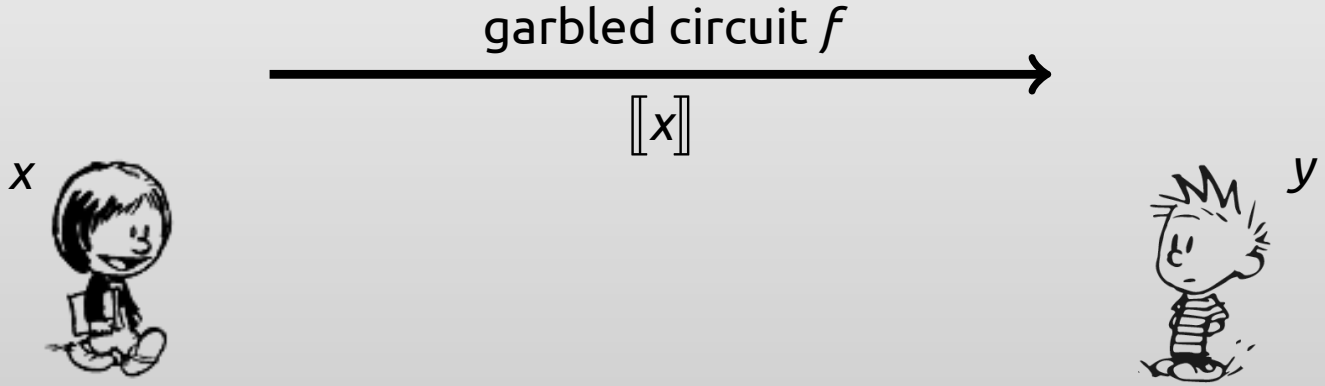
- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** $\equiv$ all encrypted gates
- **Garbled encoding** $\equiv$ one label per wire

Garbled evaluation:

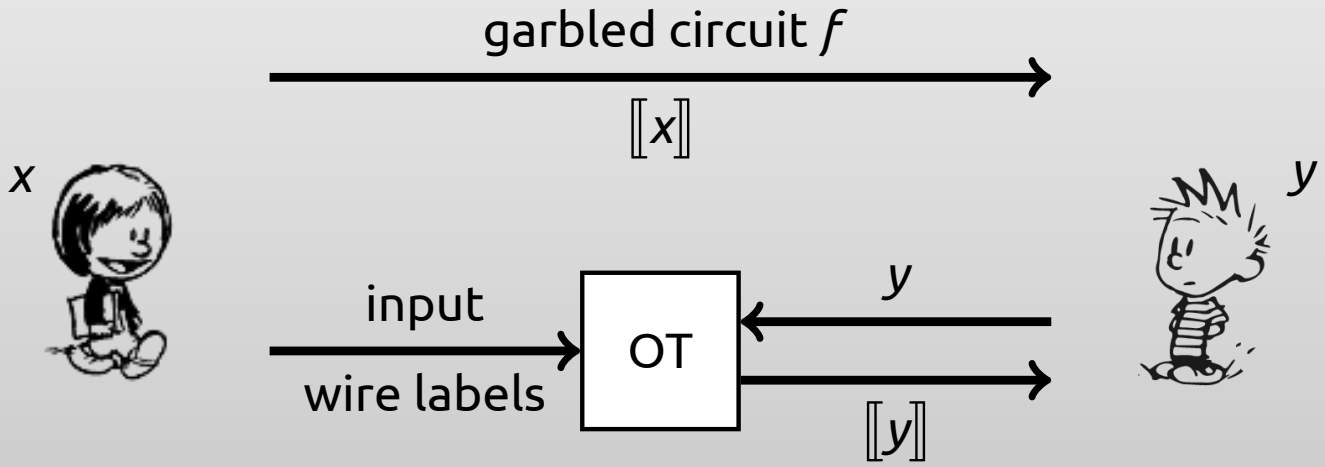
- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuits for 2PC

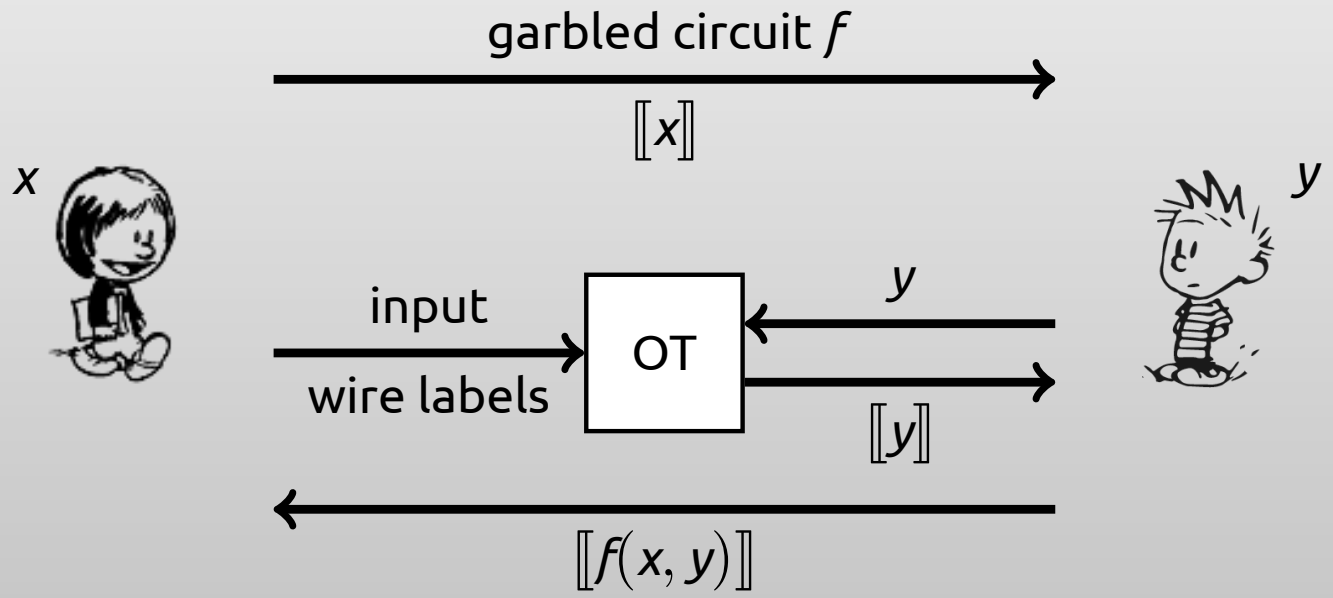
Garbled circuits for 2PC



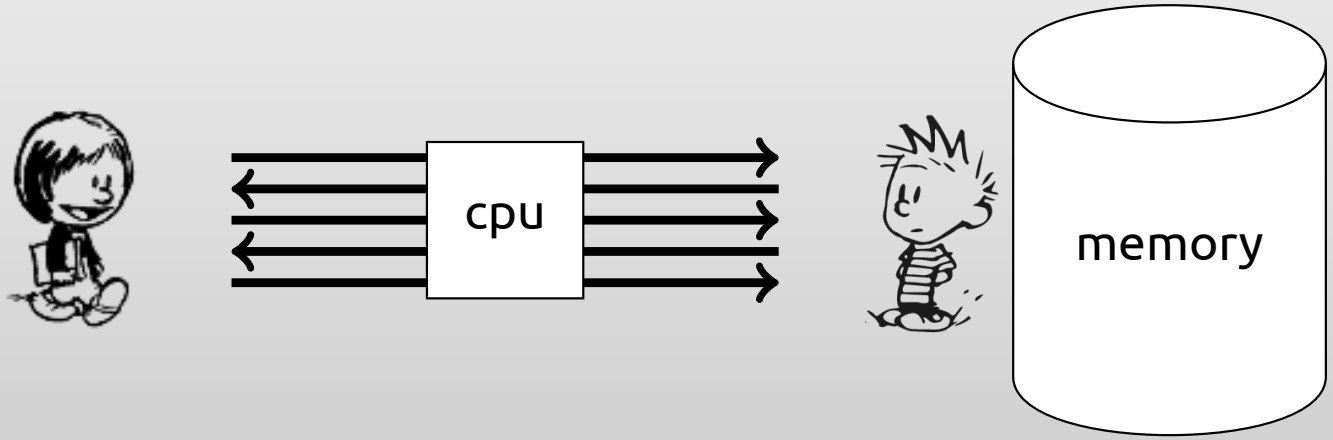
Garbled circuits for 2PC



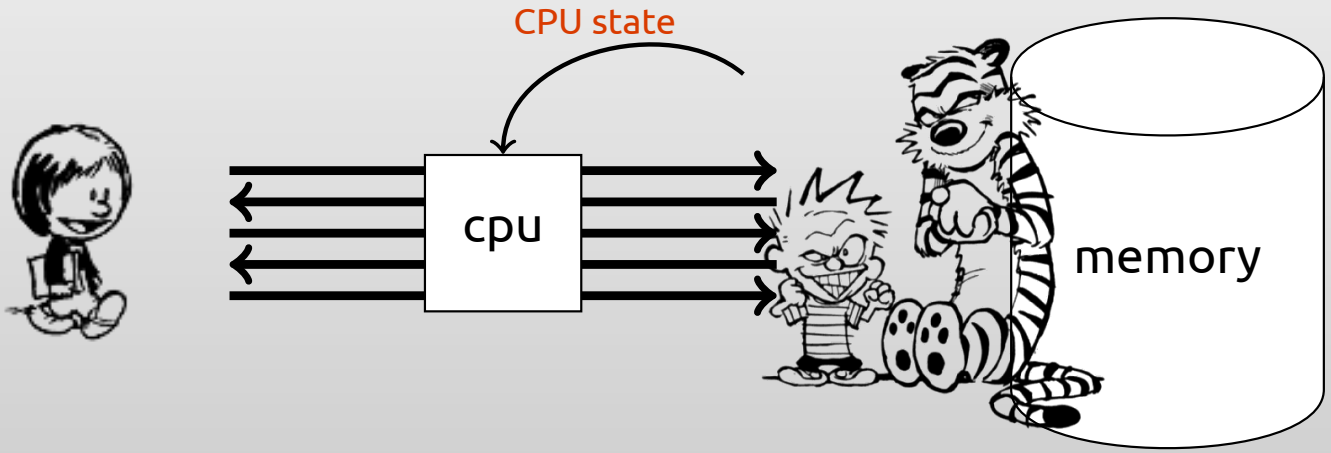
Garbled circuits for 2PC



What can go wrong? (II)



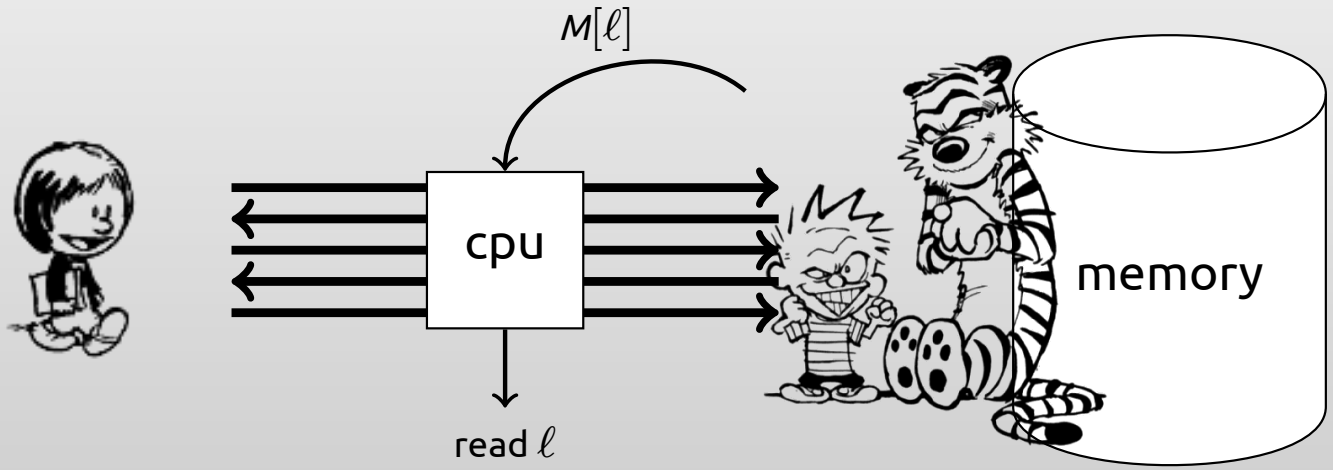
What can go wrong? (II)



Corrupt party can mess up computation by:

- Providing wrong (share of) CPU state

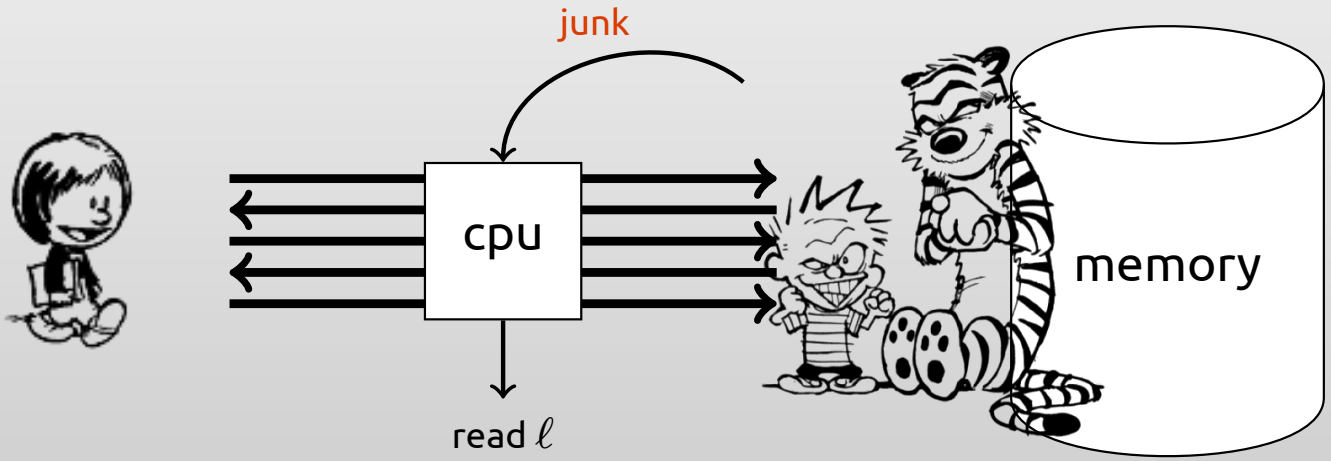
What can go wrong? (II)



Corrupt party can mess up computation by:

- Providing wrong (share of) CPU state

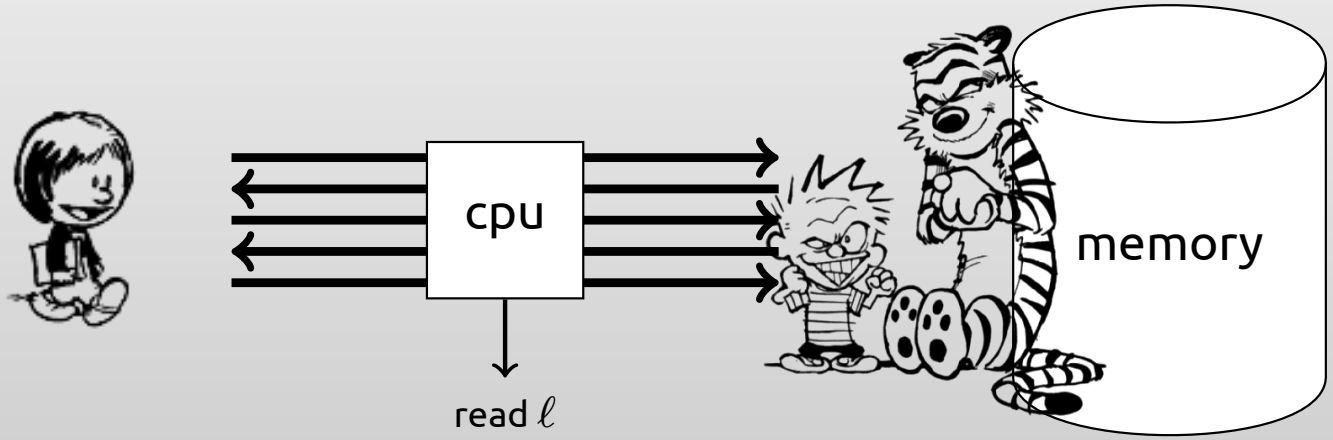
What can go wrong? (II)



Corrupt party can mess up computation by:

- Providing wrong (share of) CPU state
- Providing wrong memory contents

What can go wrong? (II)

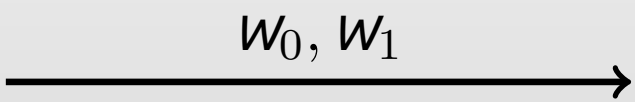


Corrupt party can mess up computation by:

- Providing wrong (share of) CPU state
- Providing wrong memory contents

Our approach [AfsharHuMohasselR15]

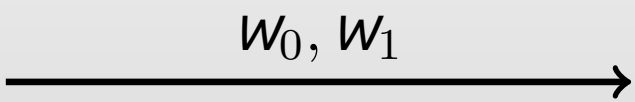
Idea: represent state/memory [re]using **garbled encodings!**



- **Privacy**: Given W_b, can't guess b
- **Authenticity:** Given W_b, can't guess W_{1-b}

Our approach [AfsharHuMohasselR15]

Idea: represent state/memory [re]using **garbled encodings!**

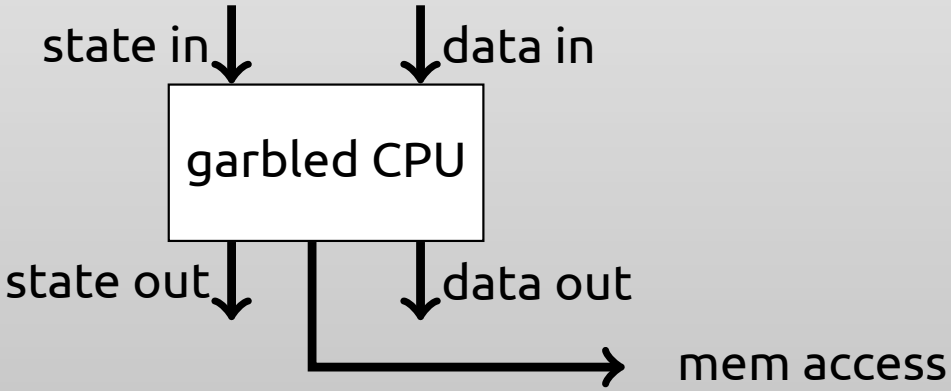


- **Privacy**: Given W_b, can't guess b
- **Authenticity:** Given W_b, can't guess W_{1-b}

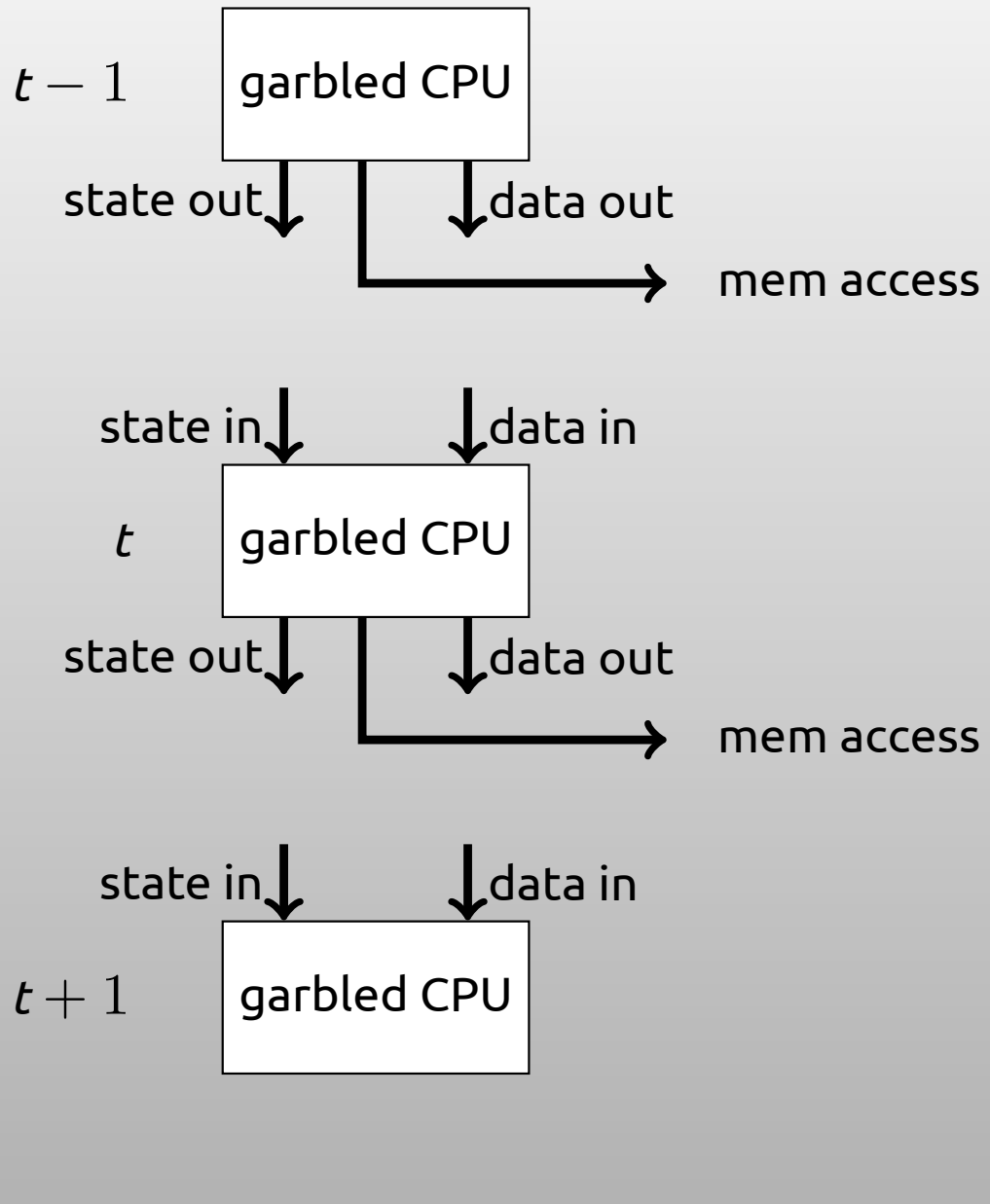
Benefits:

- CPU next-instruction circuit doesn't need to encrypt/decrypt (garbled encoding already hides the information)
- CPU next-instruction circuit doesn' need to secret-share CPU state

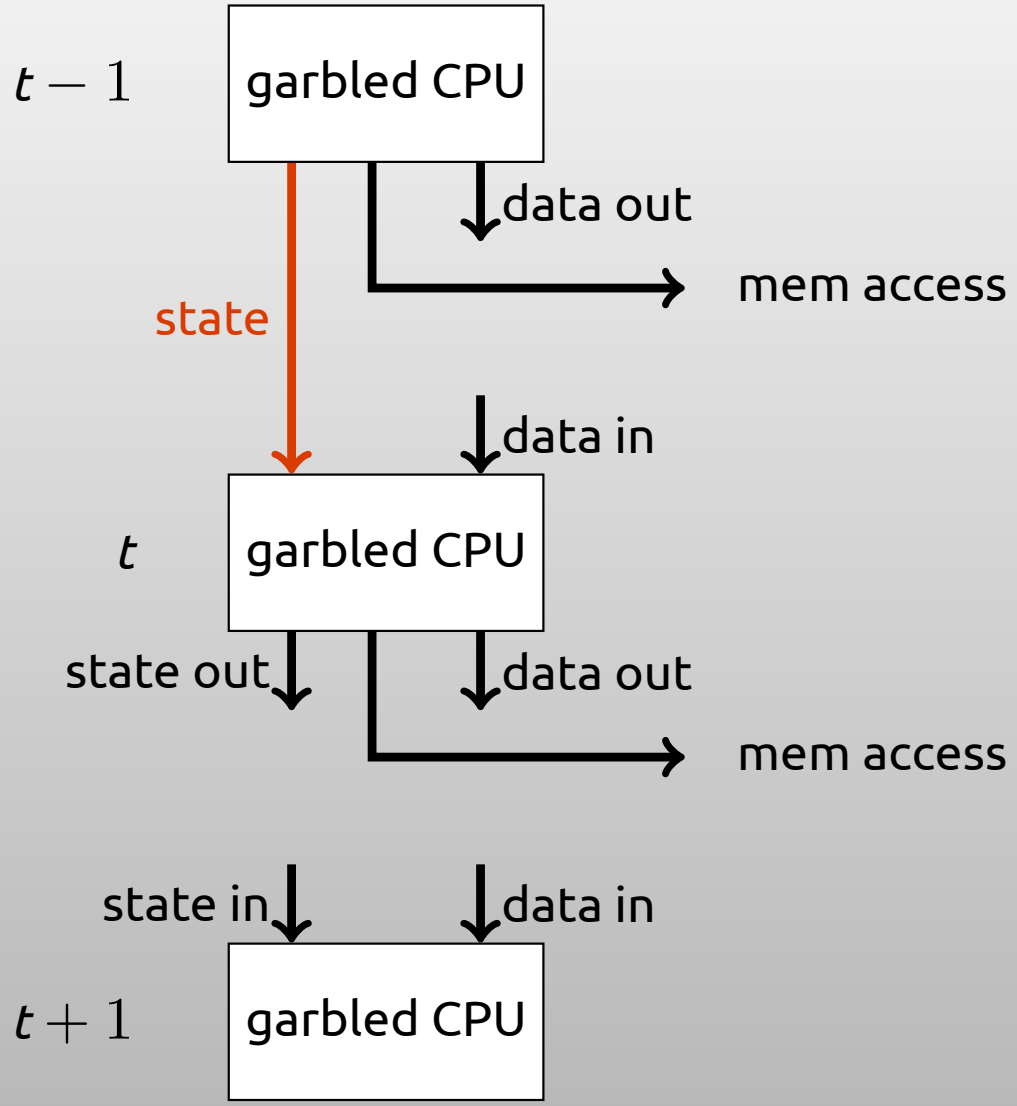
Reusing garbled encodings



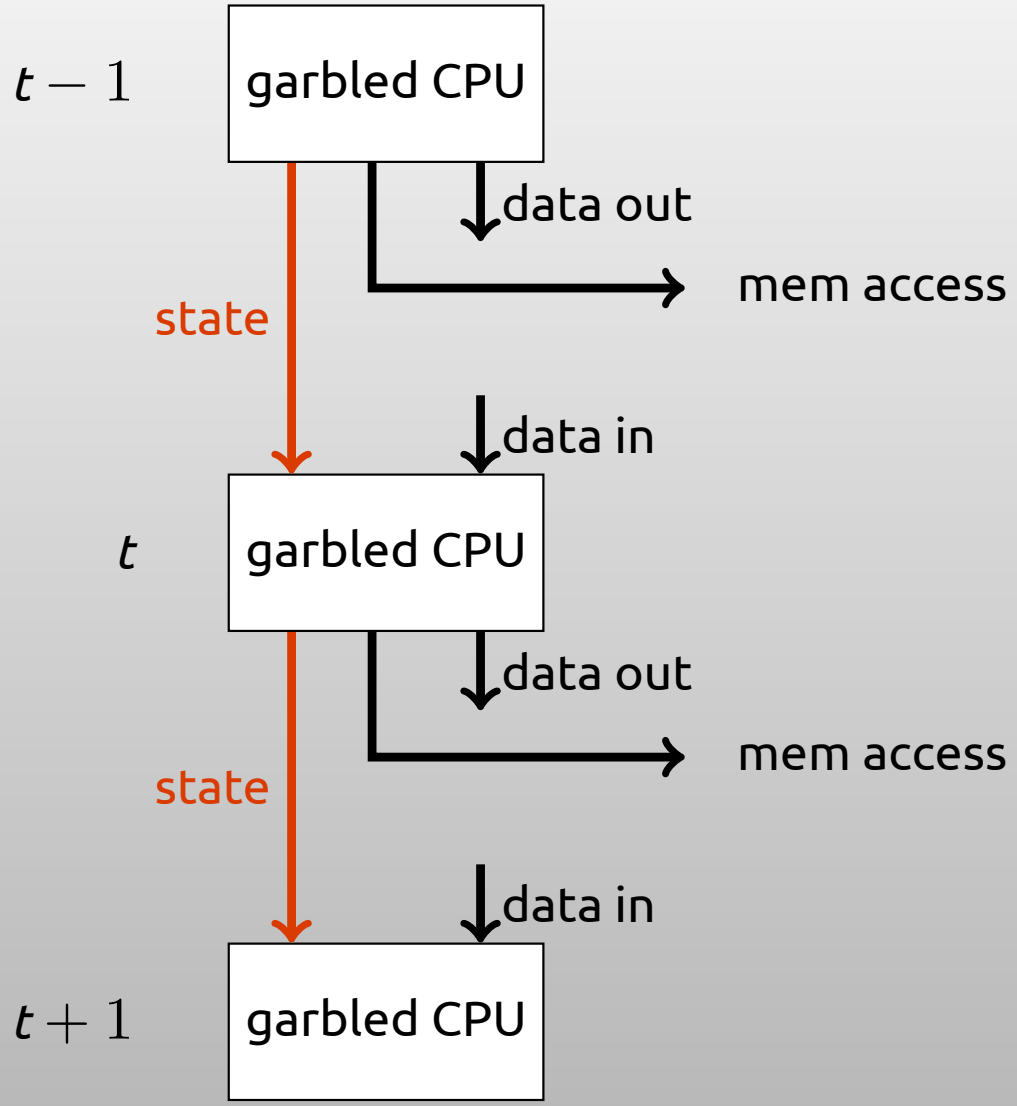
Reusing garbled encodings



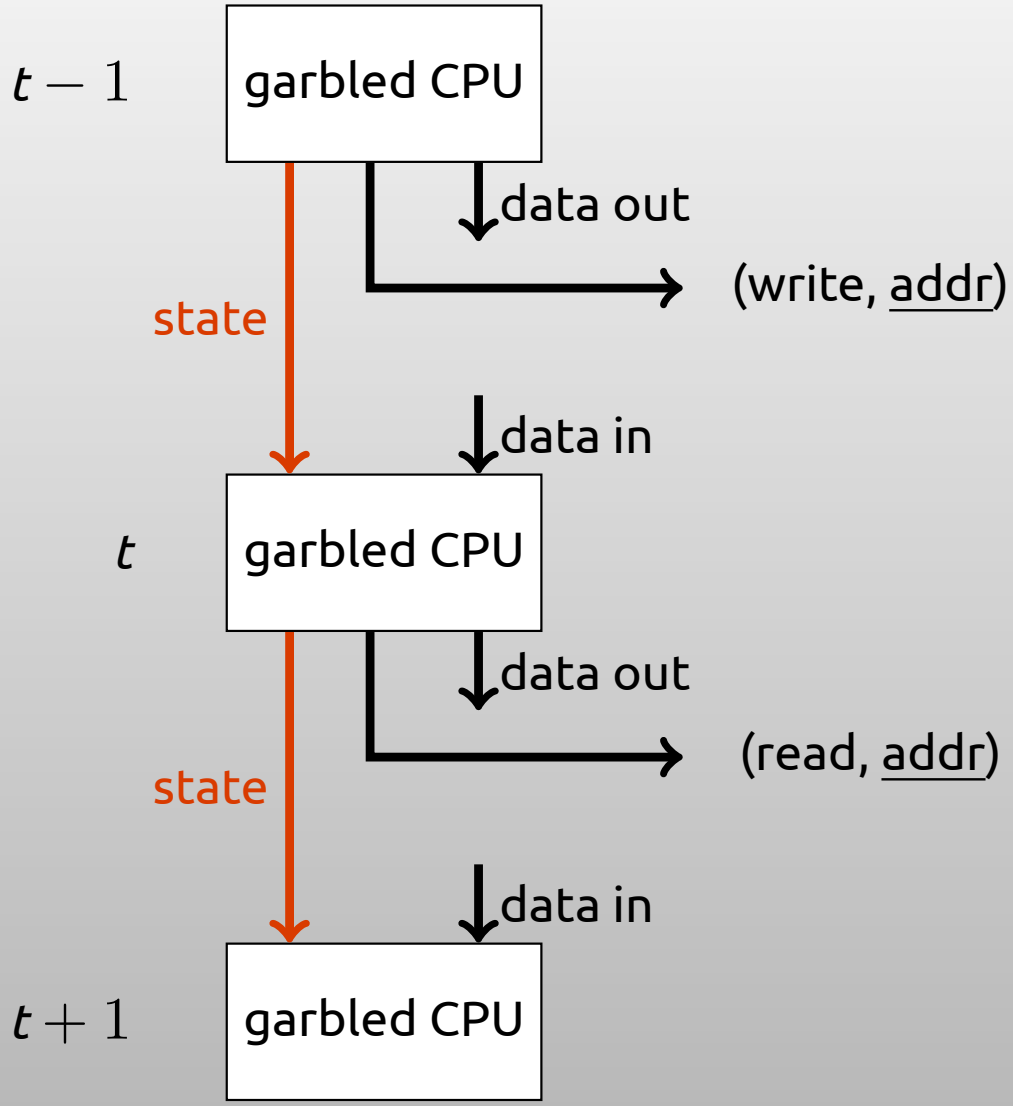
Reusing garbled encodings



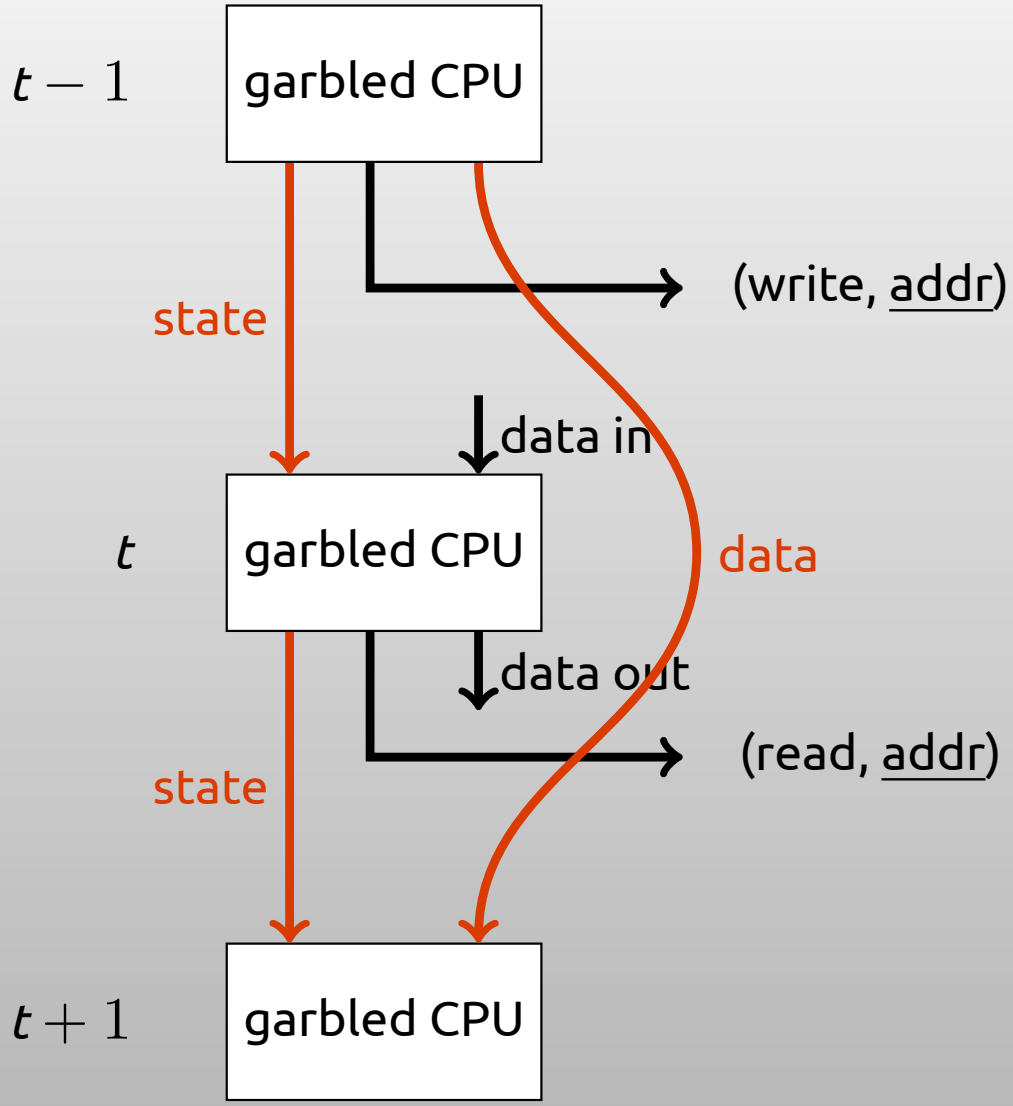
Reusing garbled encodings



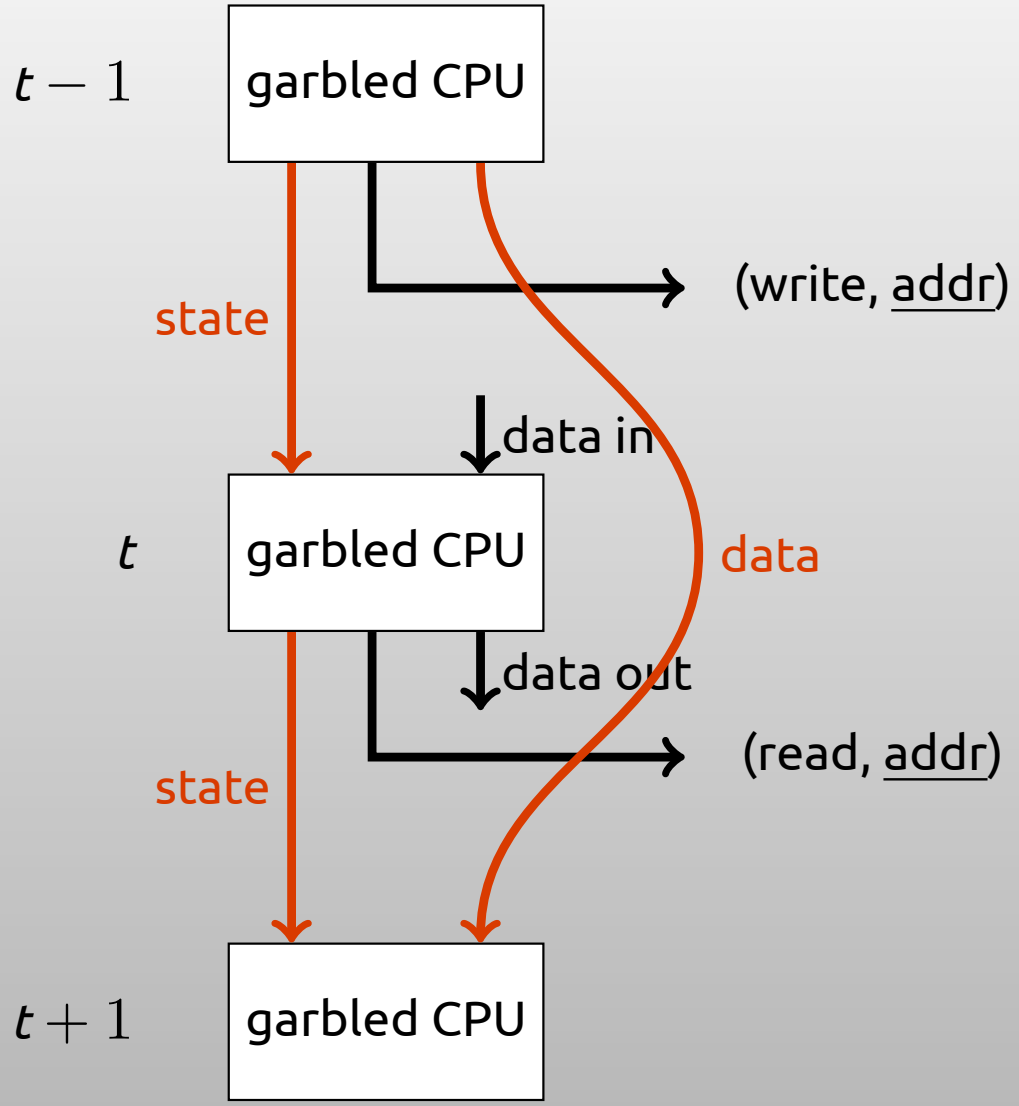
Reusing garbled encodings



Reusing garbled encodings



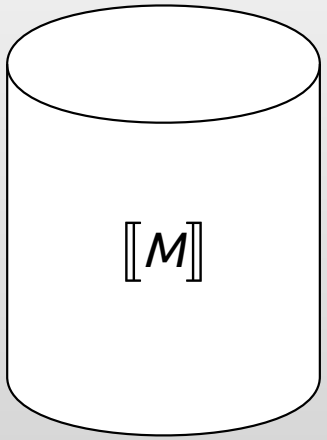
Reusing garbled encodings



Must know ORAM access pattern to choose appropriate garbled encoding for next circuit.

(Contrast with naively converting ORAM to circuit)

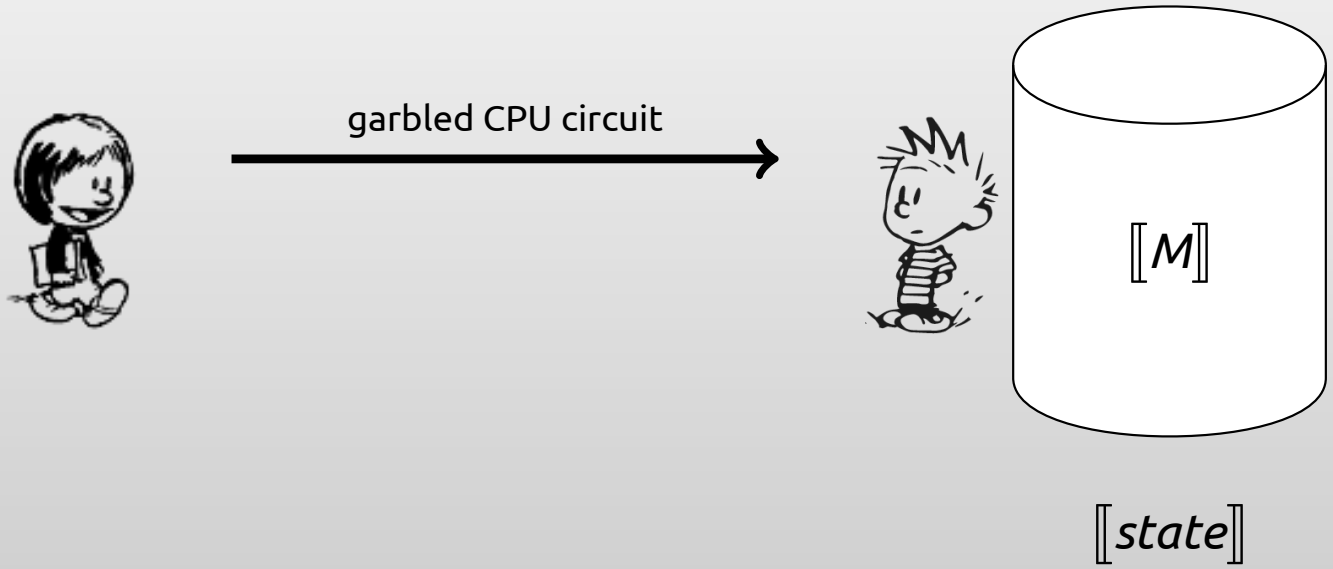
Our approach [AfsharHuMohasselR15]



⟦state⟧

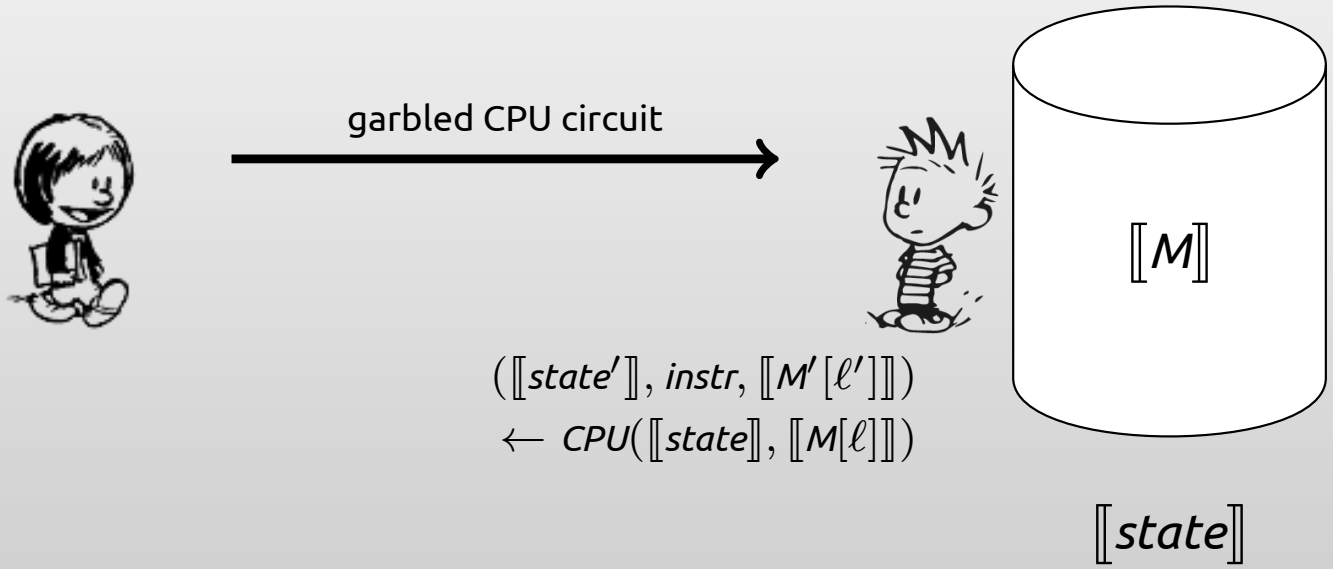
- Memory and state encoded with garbled encoding.

Our approach [AfsharHuMohasselR15]



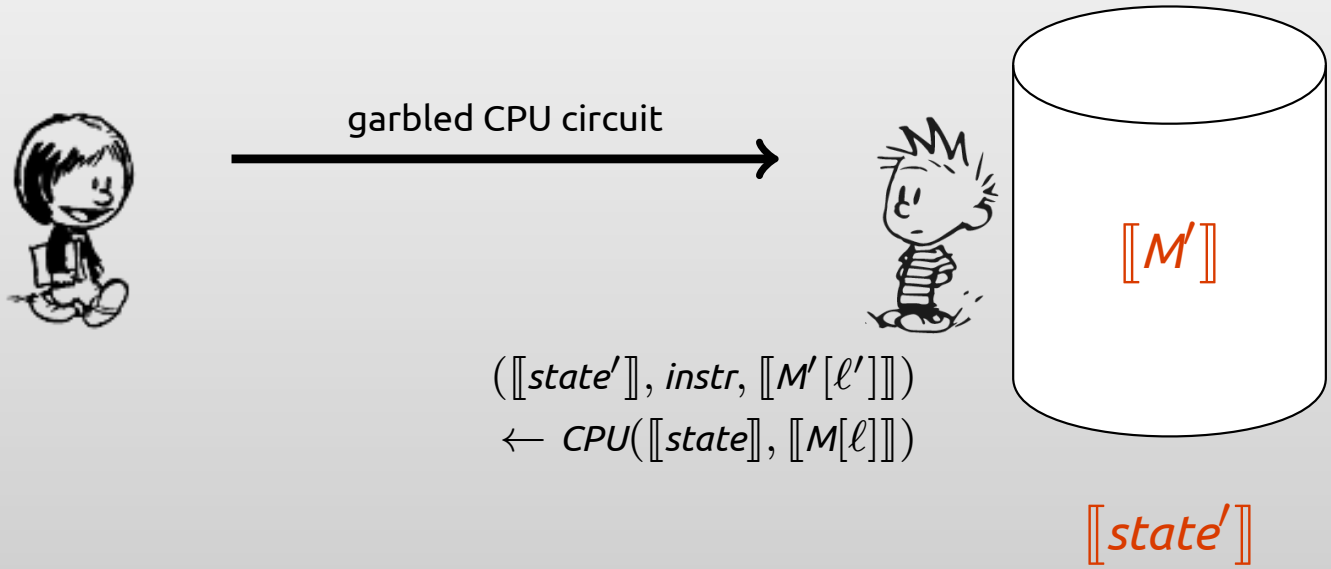
- Memory and state encoded with garbled encoding.
- Susie garbles circuit with input encoding matching previous output encoding

Our approach [AfsharHuMohasselR15]



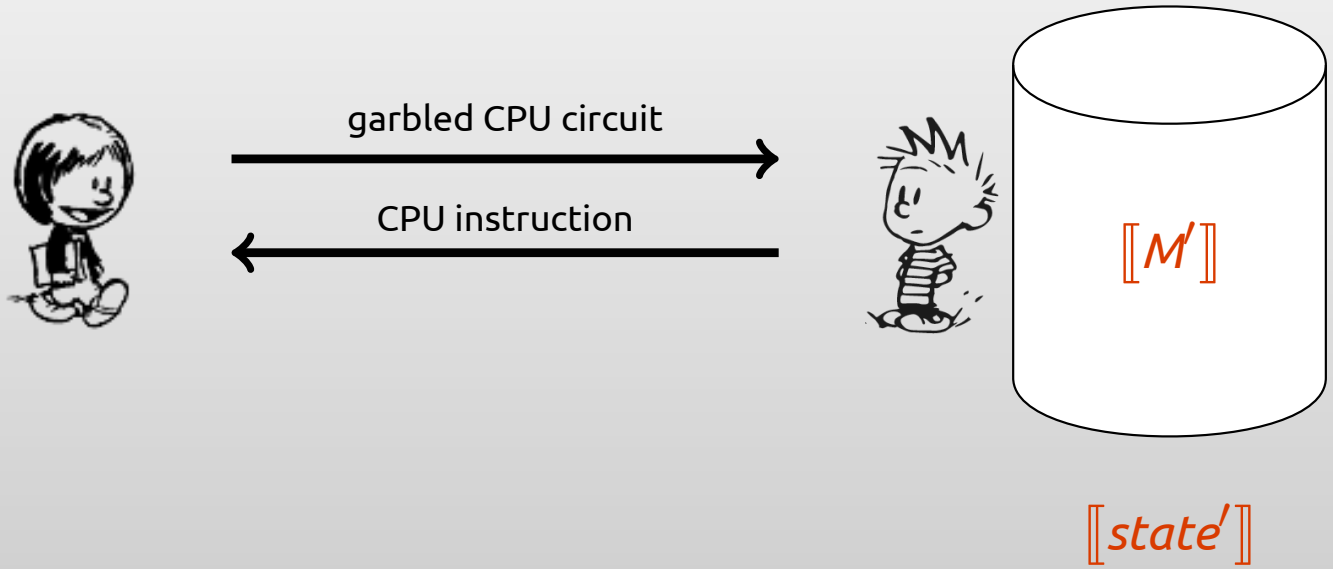
- Memory and state encoded with garbled encoding.
- Susie garbles circuit with input encoding matching previous output encoding
- Only valid input Calvin can provide is previous circuit's output.

Our approach [AfsharHuMohasselR15]



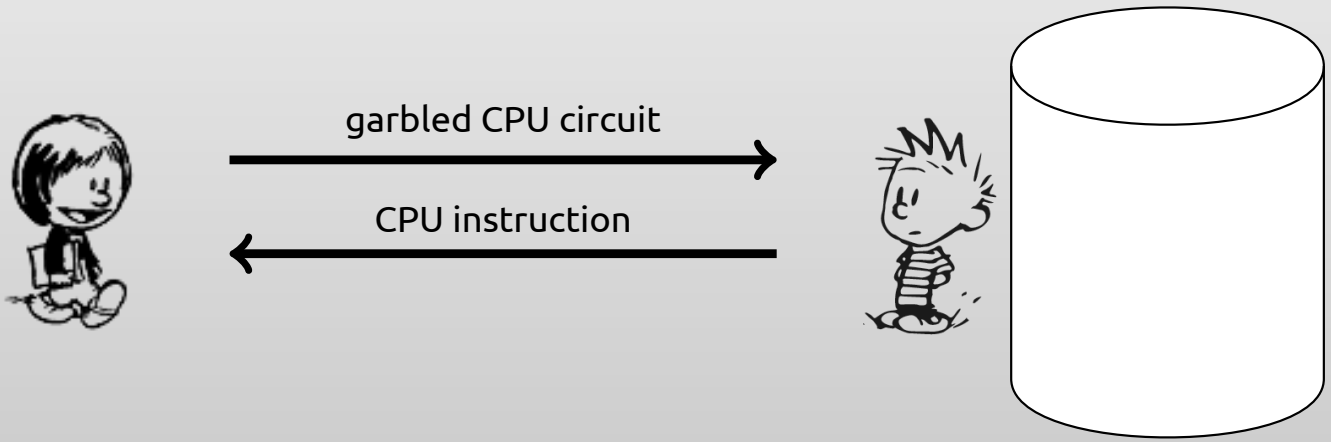
- Memory and state encoded with garbled encoding.
- Susie garbles circuit with input encoding matching previous output encoding
- Only valid input Calvin can provide is previous circuit's output.

Our approach [AfsharHuMohasselR15]



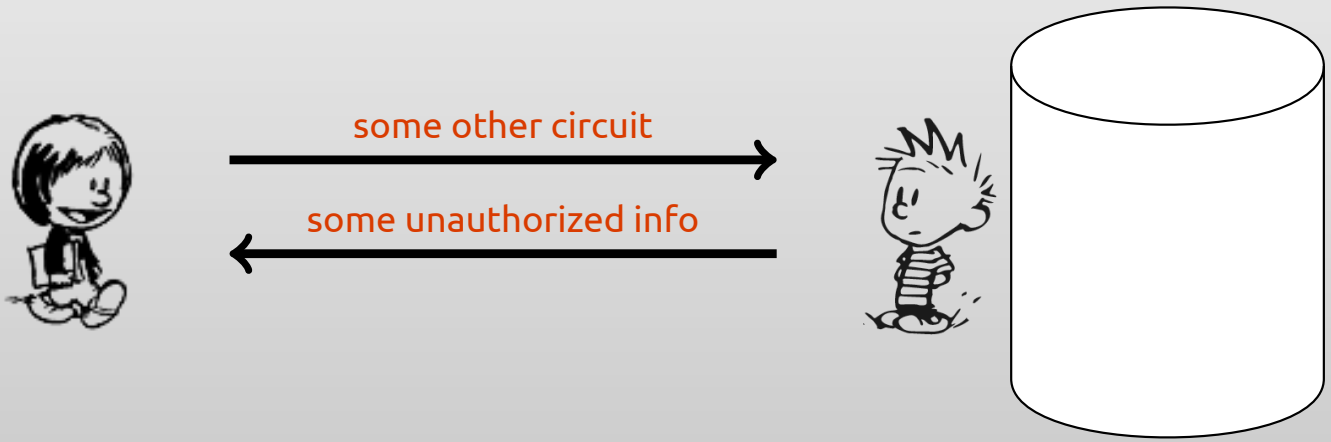
- Memory and state encoded with garbled encoding.
- Susie garbles circuit with input encoding matching previous output encoding
- Only valid input Calvin can provide is previous circuit's output.

Malicious garbler



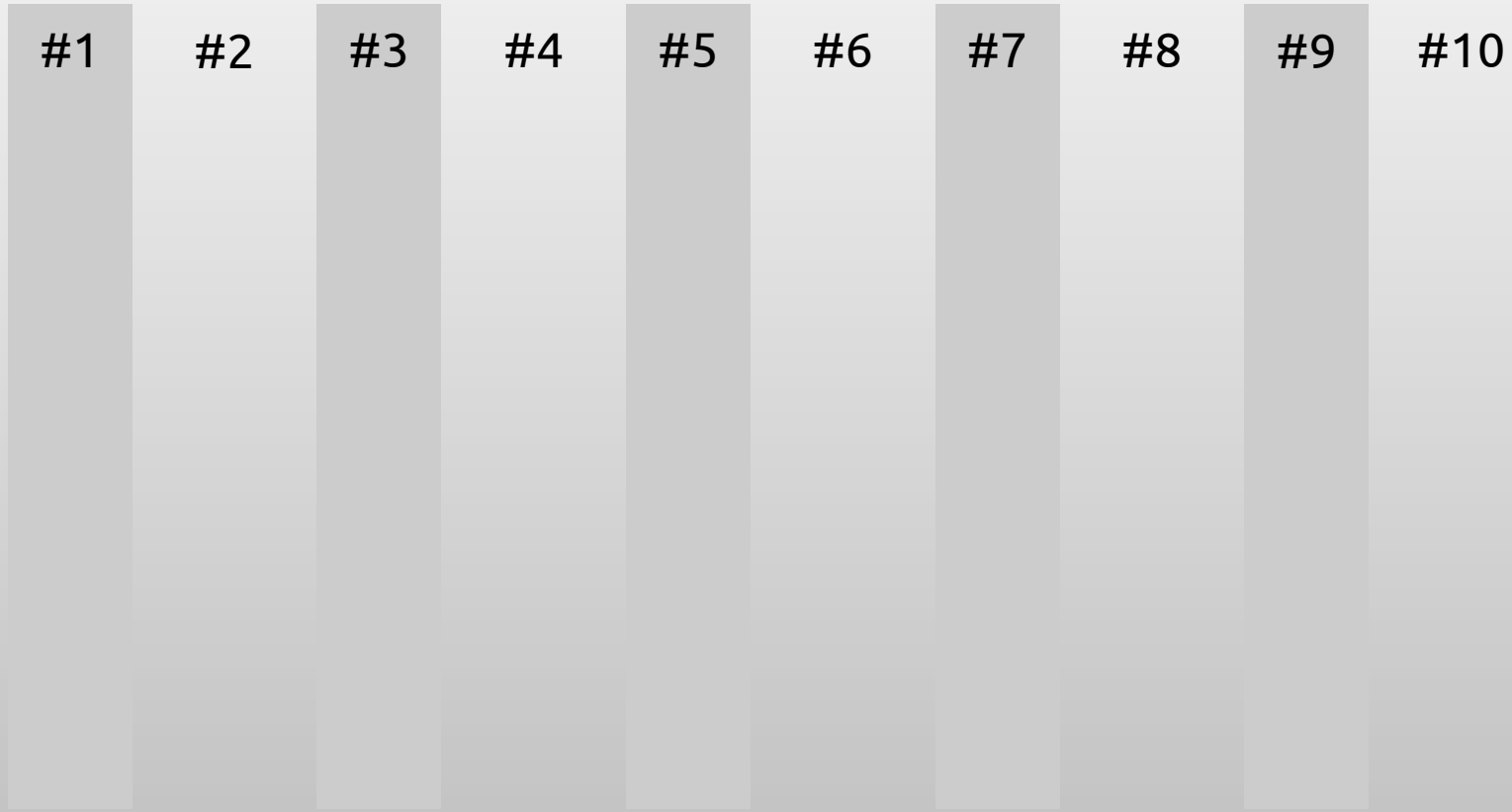
Main challenge: malicious garbler generates invalid garbled circuits.

Malicious garbler



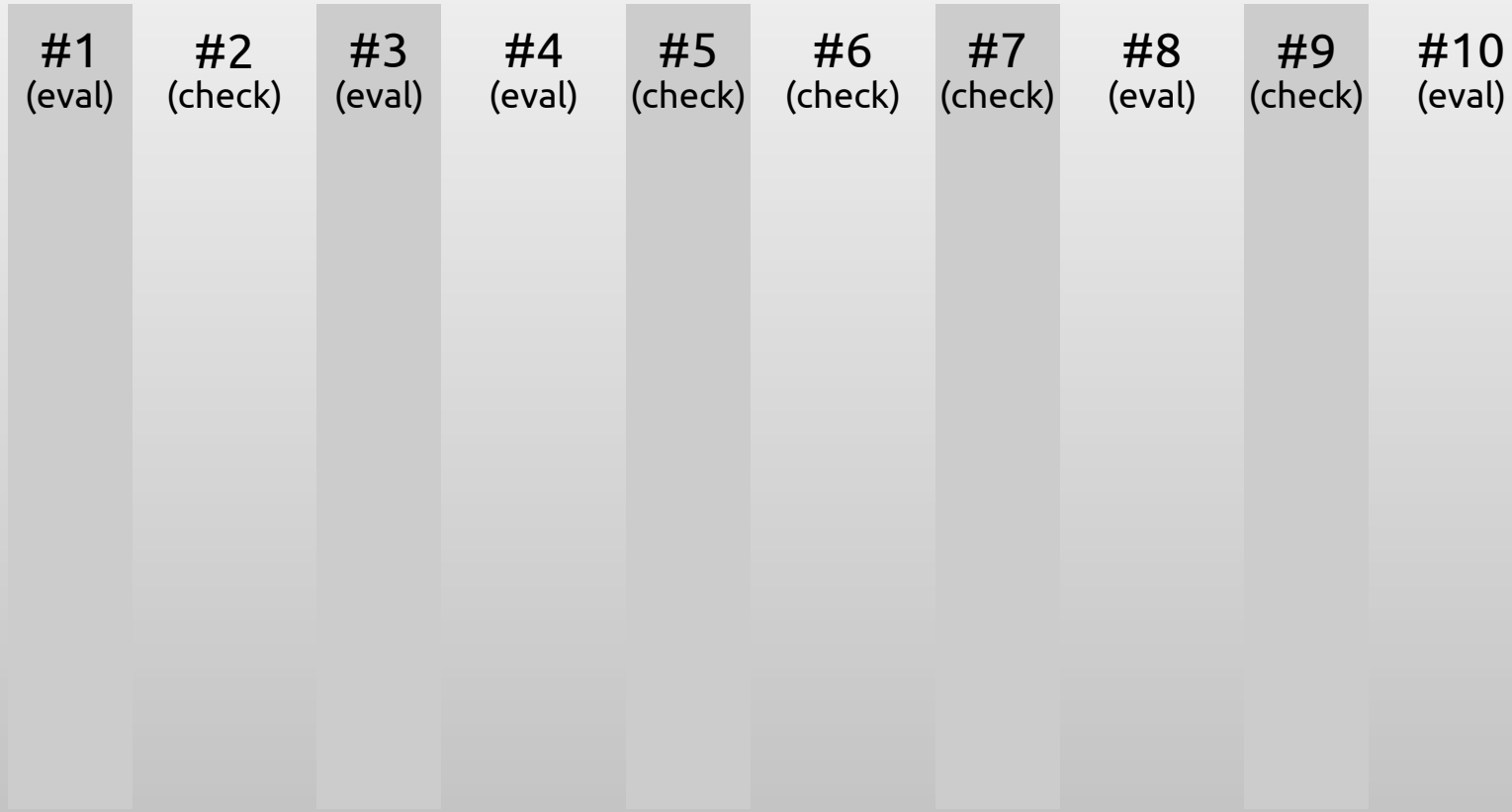
Main challenge: malicious garbler generates invalid garbled circuits.

cut and choose



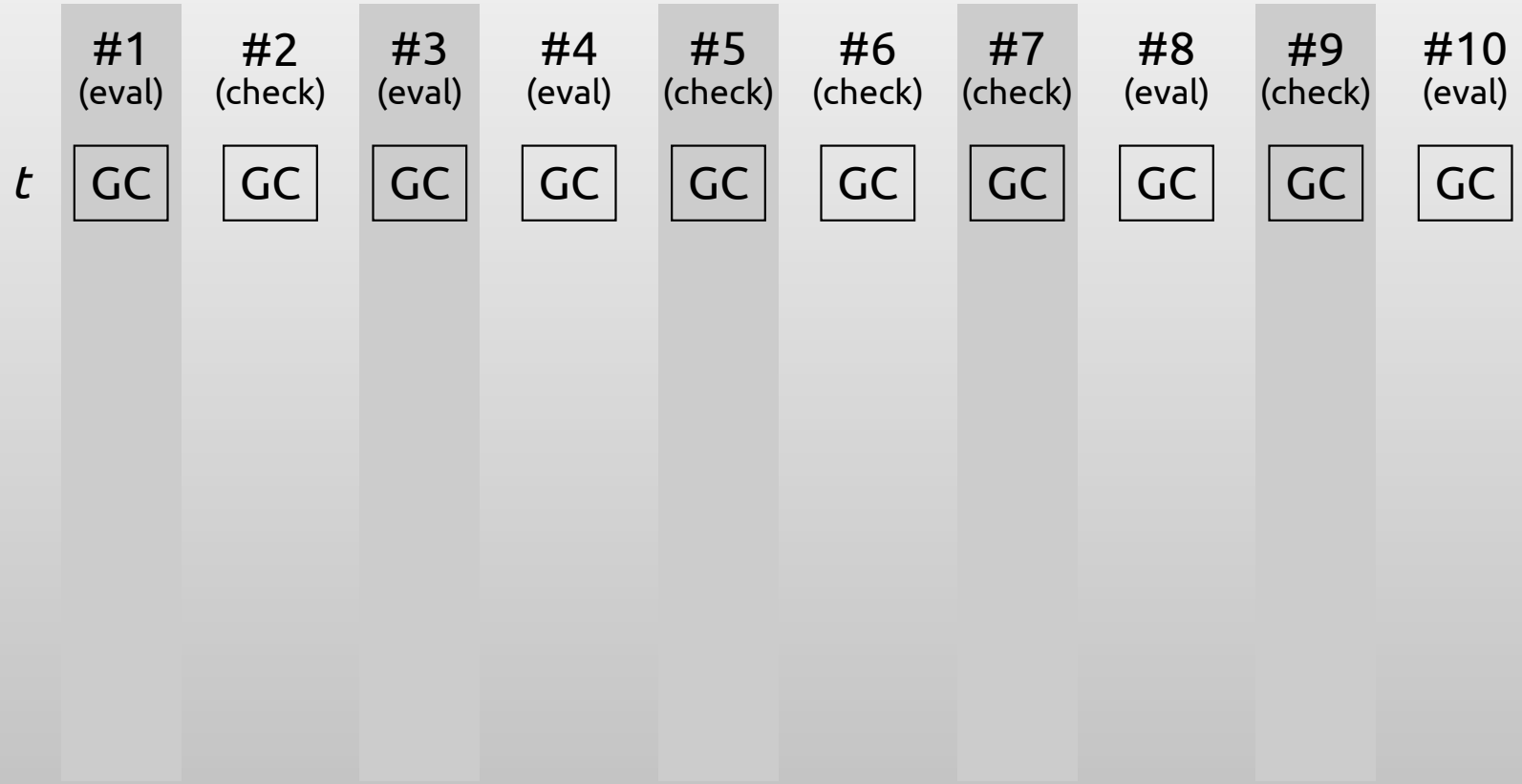
establish many threads of computation

cut and choose



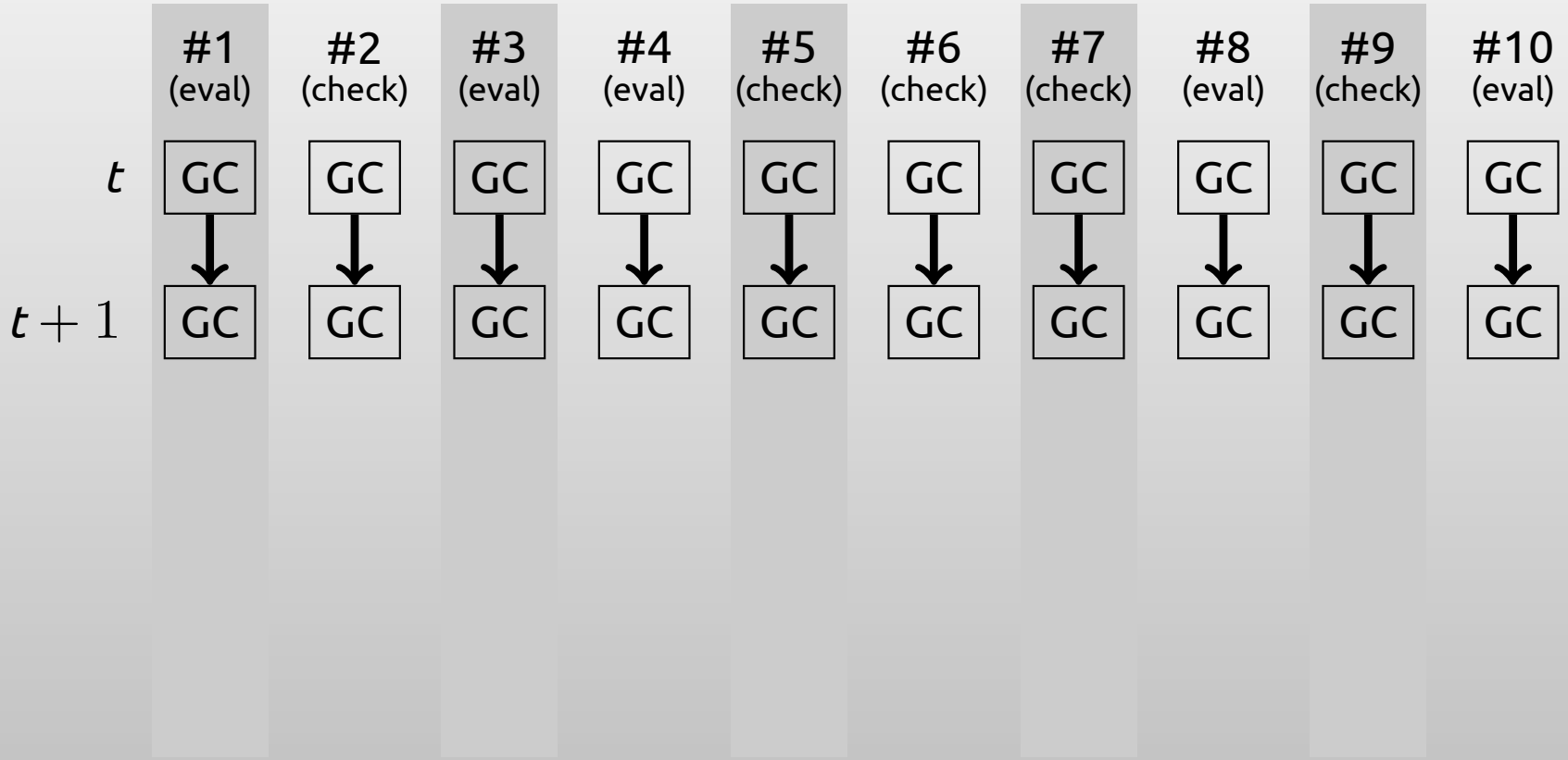
receiver secretly sets each thread to “check” or “eval”

cut and choose



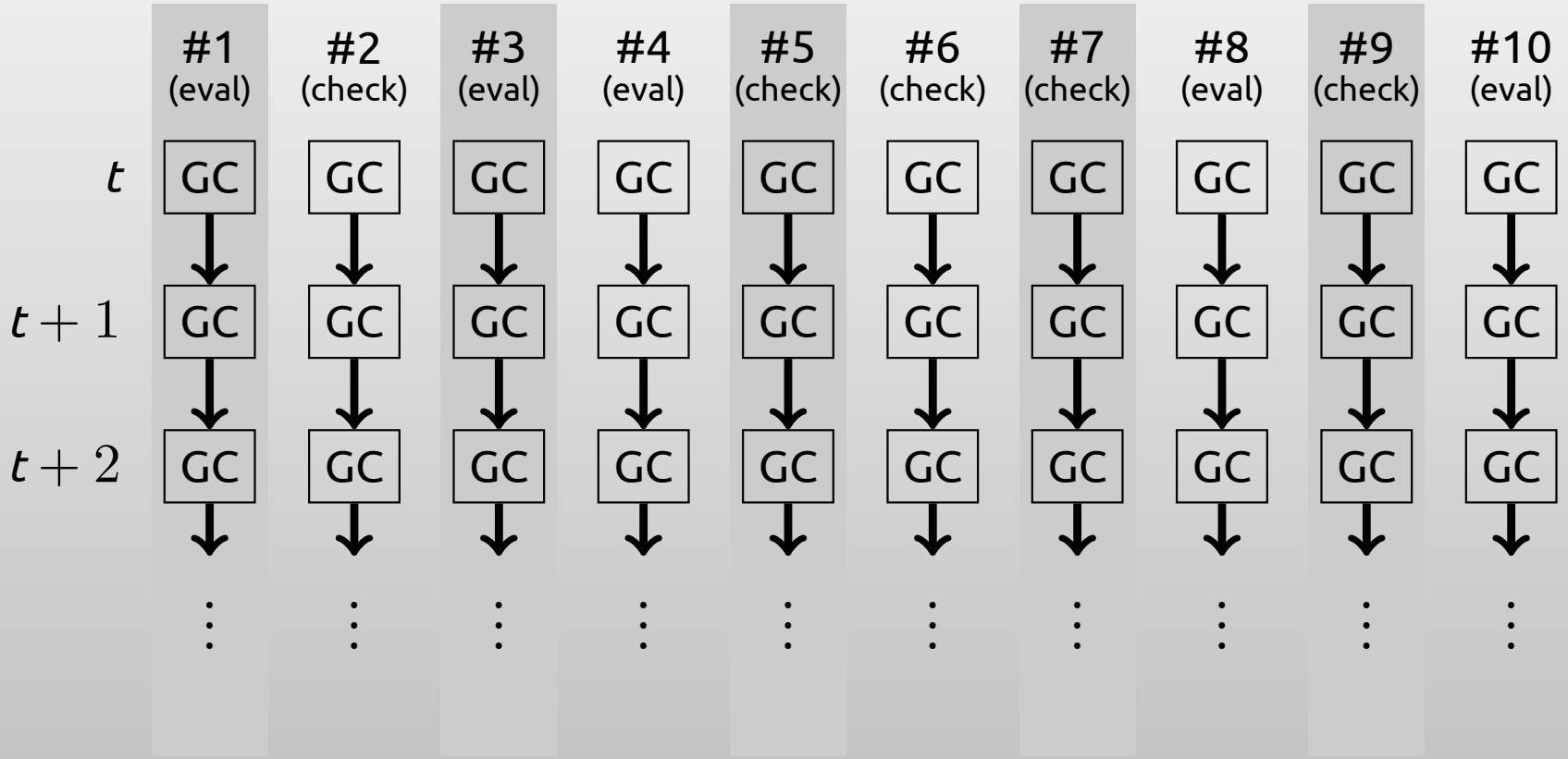
sender generates garbled circuits, reusing wire labels within each thread

cut and choose



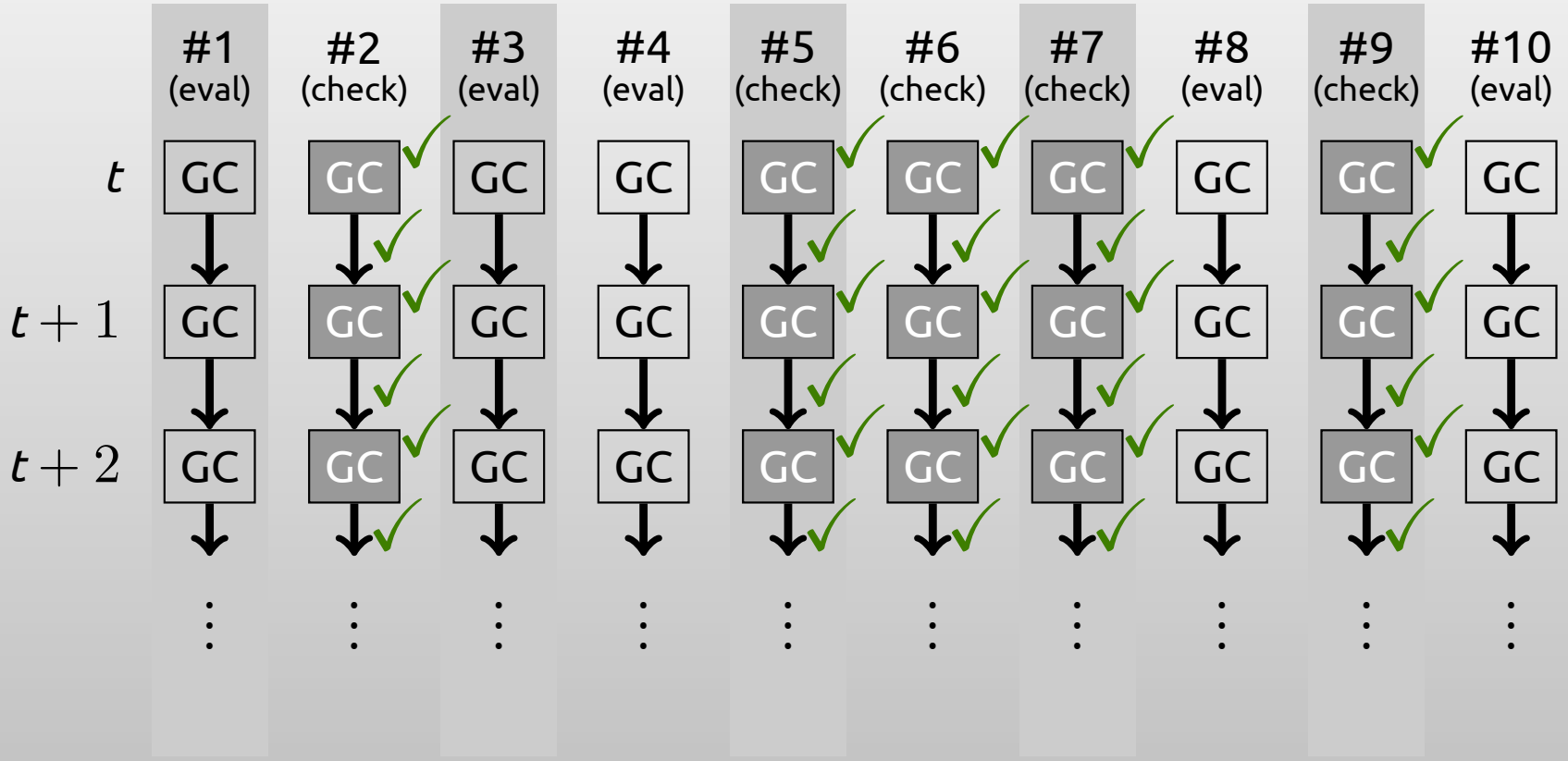
sender generates garbled circuits, reusing wire labels within each thread

cut and choose



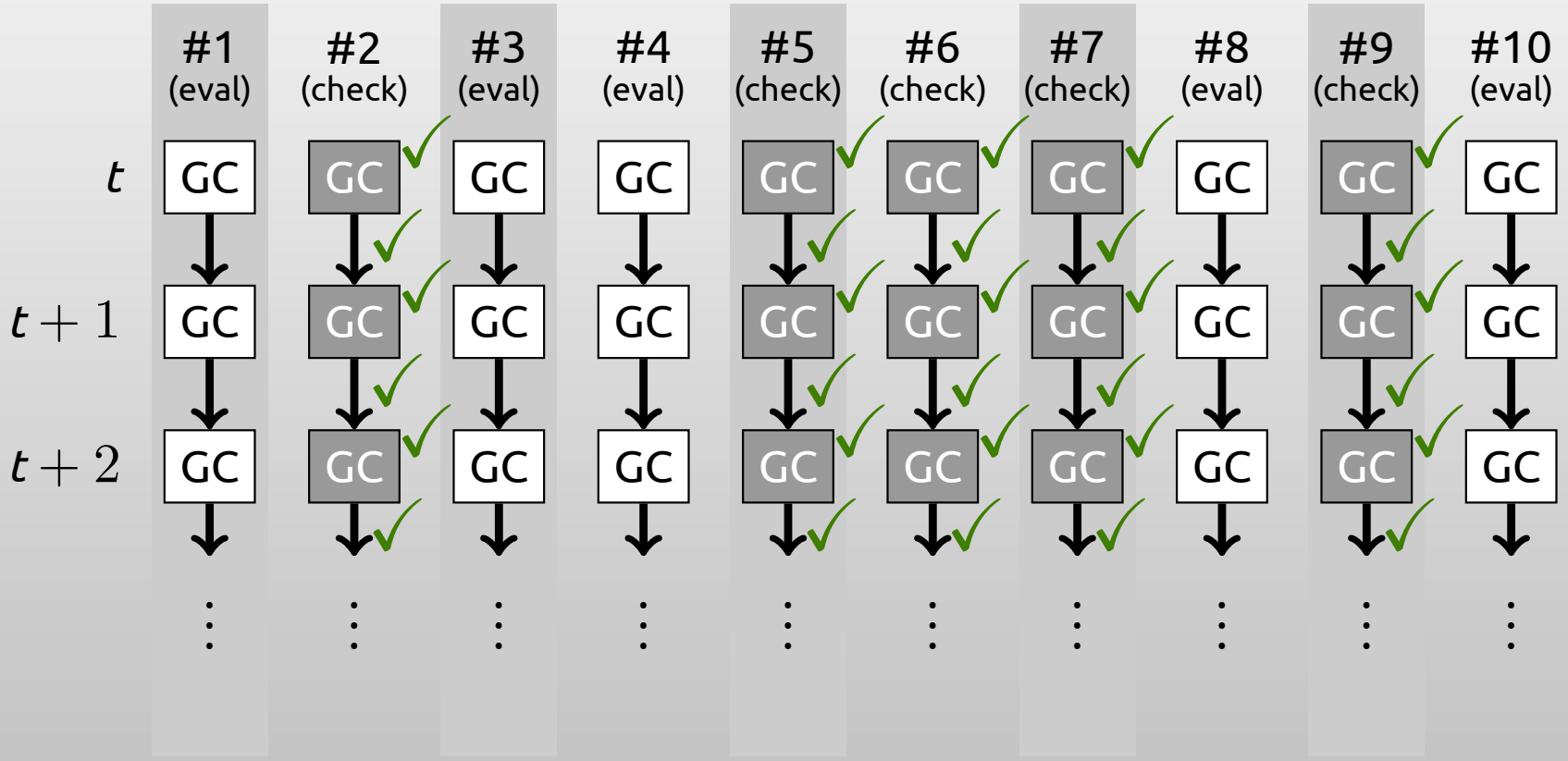
sender generates garbled circuits, reusing wire labels within each thread

cut and choose



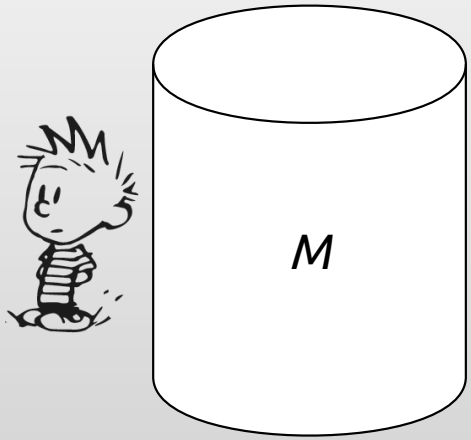
check-threads: receiver gets both labels per wire $\Rightarrow$ check correct behavior

cut and choose



eval-threads: receiver gets one garbled encoding $\Rightarrow$ learns only prescribed outpu

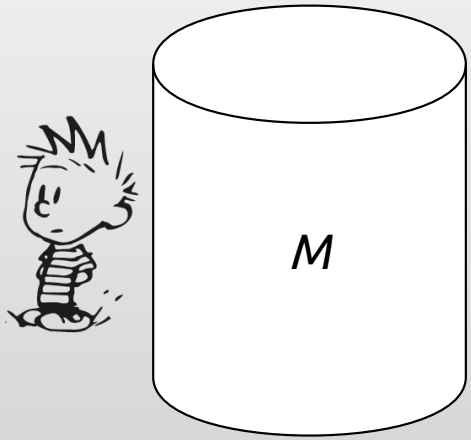
One-sided secrets



Setting:

- *M* is Calvin's secret input; expensive ORAM initialization commits him to *M*
- Repeatedly run *public* ORAM program on *M*
- Example: *M* = user database; check for membership

One-sided secrets



Setting:

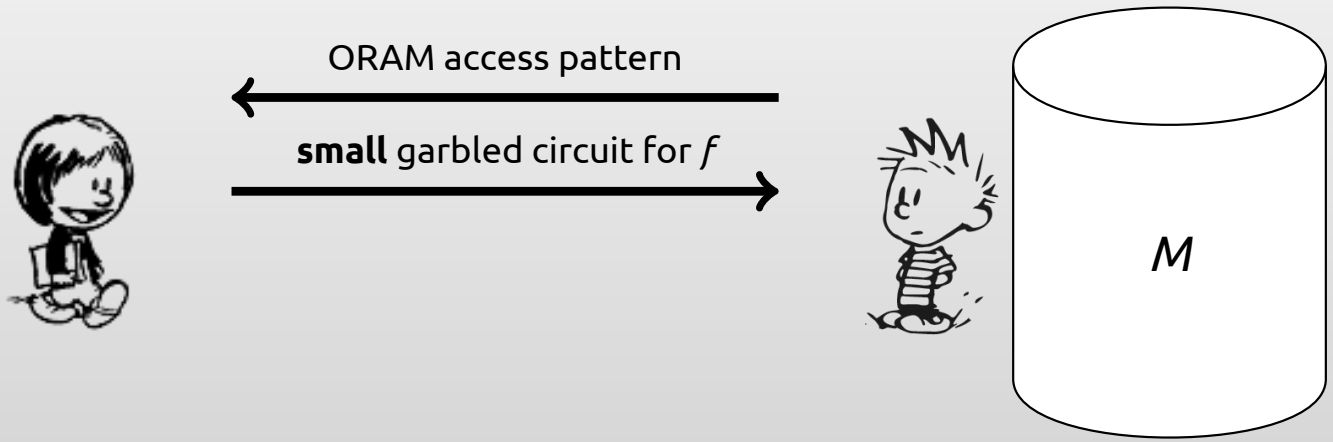
- *M* is Calvin's secret input; expensive ORAM initialization commits him to *M*
- Repeatedly run *public* ORAM program on *M*
- Example: *M* = user database; check for membership

In this case we can avoid cut & choose, avoid high interaction!

Avoiding cut and choose [HuMohasselR15]

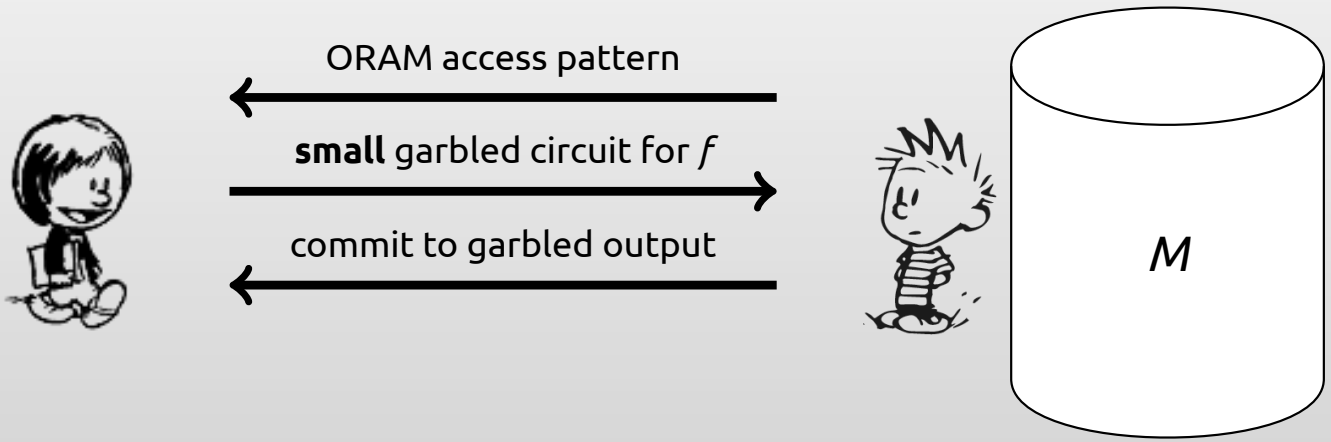
- Calvin knows all inputs, can run ORAM in his head

Avoiding cut and choose [HuMohasselR15]



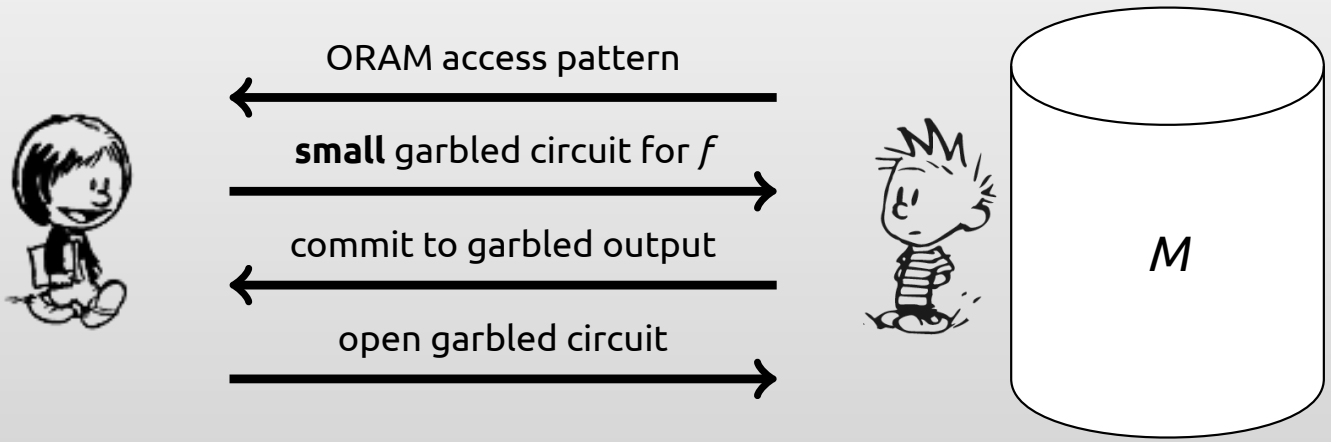
- Calvin knows all inputs, can run ORAM in his head
- Knowing ORAM access pattern, can convert to **small** circuit

Avoiding cut and choose [HuMohasselR15]



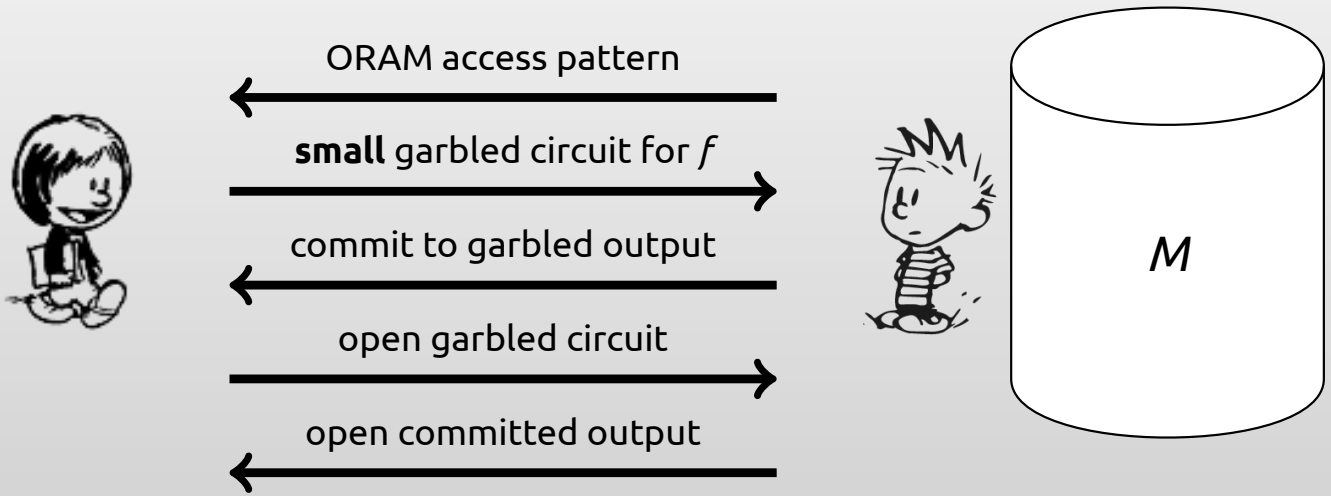
- Calvin knows all inputs, can run ORAM in his head
- Knowing ORAM access pattern, can convert to **small** circuit
- Calvin can evaluate garbled circuit

Avoiding cut and choose [HuMohasselR15]



- Calvin knows all inputs, can run ORAM in his head
- Knowing ORAM access pattern, can convert to **small** circuit
- Calvin can evaluate garbled circuit
- Susie can open garbled circuit (no secrets to hide!)

Avoiding cut and choose [HuMohasselR15]



- Calvin knows all inputs, can run ORAM in his head
- Knowing ORAM access pattern, can convert to **small** circuit
- Calvin can evaluate garbled circuit
- Susie can open garbled circuit (no secrets to hide!)
- Calvin opens committed output knowing GC was correctly generated

Conclusion

RAM-based 2PC can provide sublinear cost in **amortized sense**, using practical 2PC techniques

- [GKKKRV12] = general paradigm, semi-honest security
- [AHMR15] = malicious security
- [HMR15] = malicious security with one-sided secrets; no cut-and-choose, constant rounds

Challenges:

- Expensive pre-processing (ORAM initialization): communication & computation
- Applying pre-processing to multiple users?
- For which computations must we “touch every bit?”

Conclusion

RAM-based 2PC can provide sublinear cost in **amortized sense**, using practical 2PC techniques

- [GKKKRV12] = general paradigm, semi-honest security
- [AHMR15] = malicious security
- [HMR15] = malicious security with one-sided secrets; no cut-and-choose, constant rounds

Challenges:

- Expensive pre-processing (ORAM initialization): communication & computation
- Applying pre-processing to multiple users?
- For which computations must we “touch every bit?”

thanks!