from circuits to RAM programs in malicious-2PC

Abstract: Secure 2-party computation (2PC) is becoming practical in some domains. However, most approaches are limited by the fact that the desired functionality must be represented as a boolean circuit. In response, the random-access machine (RAM) model has recently been investigated as a promising alternative to circuits.

In this talk, I will discuss some pitfalls of basing malicious-secure 2PC on the RAM model rather than circuits. I will then describe two new protocols for malicious-secure 2PC of RAM programs, whose performance relative to the semi-honest model matches the state of the art for circuit-based 2PC techniques. For malicious security with statistical security parameter 2^{-s}, our protocol without preprocessing has overhead s compared to the semi-honest model; our protocol with preprocessing has overhead $\sim 2s/\log T$, where T is the running time of the RAM program.

▷ Arash Afshar @ UNIVERSITY OF CALGARY

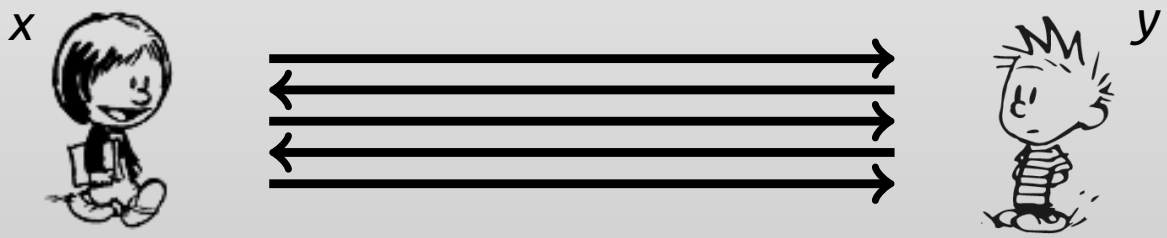
▷ Zhangxiang Hu @ Oregon State University OSU

▷ Payman Mohassel @ UNIVERSITY OF CALGARY

▶ Mike Rosulek @ Oregon State University OSU

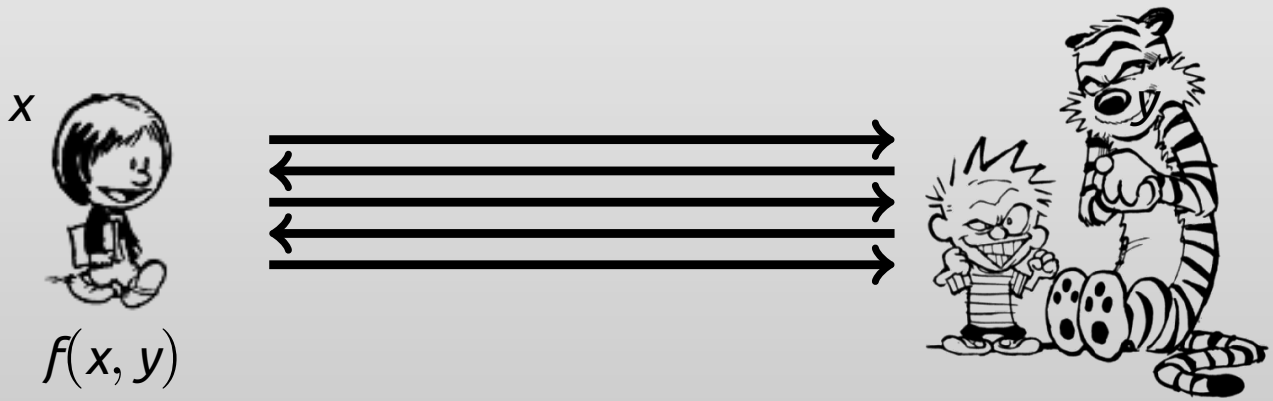
secure 2pc

secure 2pc



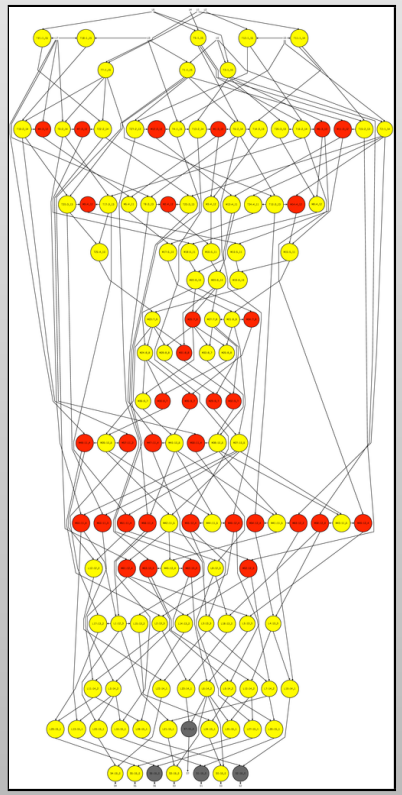
secure 2pc

secure 2pc



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

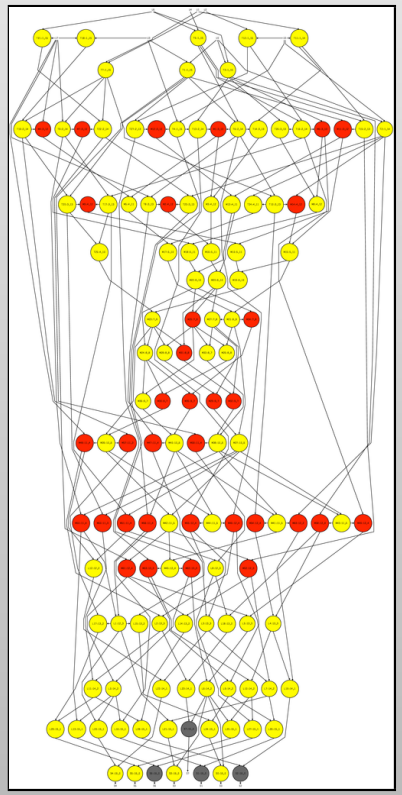
“to securely evaluate f,
first express f as a boolean circuit,
then …”



AES S-box (need 208 of these)

AES ✓

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

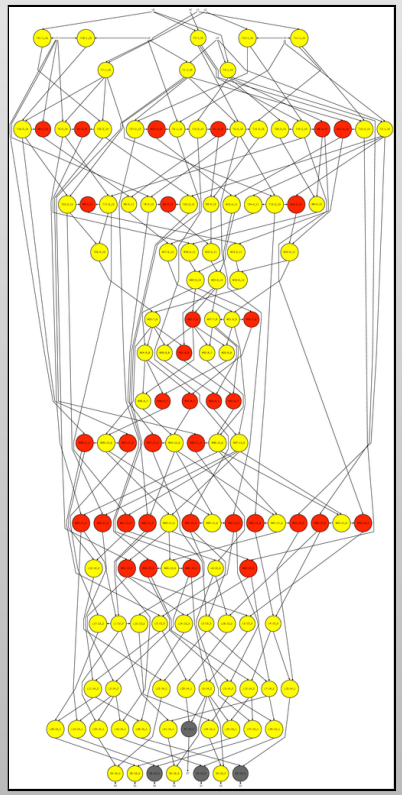


AES S-box (need 208 of these)

AES ✓

stable marriage ✗

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



AES S-box (need 208 of these)

AES ✓

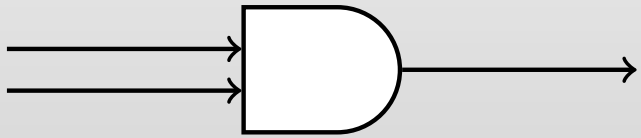
stable marriage ✗

binary search ✗

size of circuit = Ω(size of inputs)

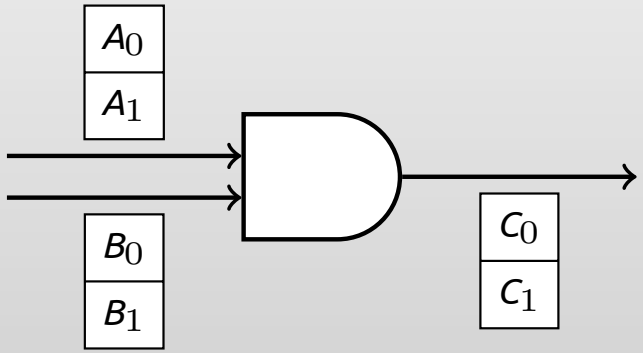
why we like (boolean) circuits

Garbled circuit technique [Yao86,BellareHoangRogaway12]



why we like (boolean) circuits

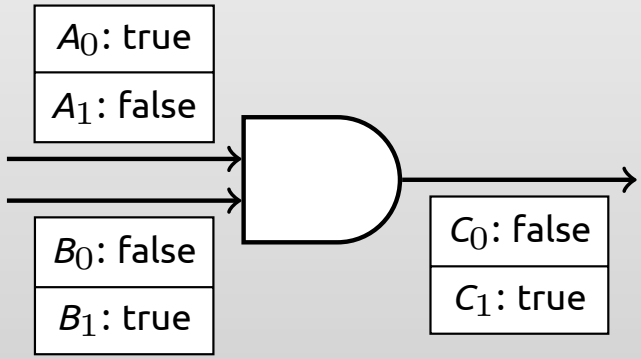
Garbled circuit technique [Yao86,BellareHoangRogaway12]



pick two random labels for each wire

why we like (boolean) circuits

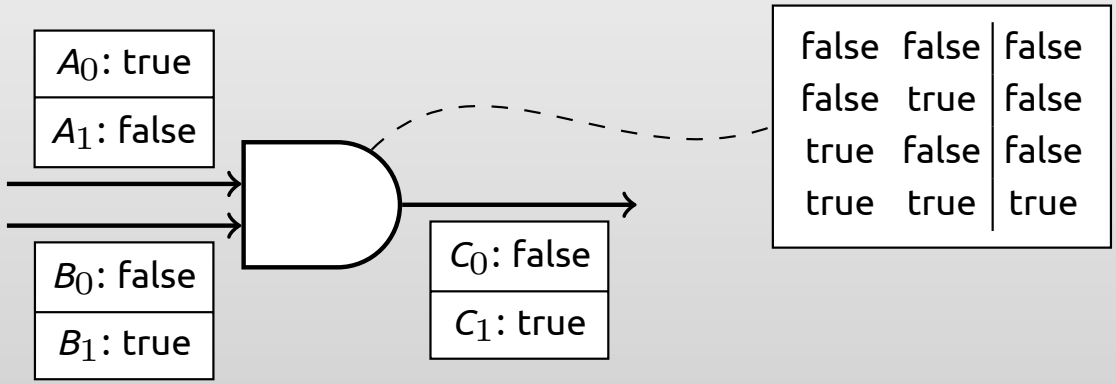
Garbled circuit technique [Yao86,BellareHoangRogaway12]



randomly associate labels with true/false

why we like (boolean) circuits

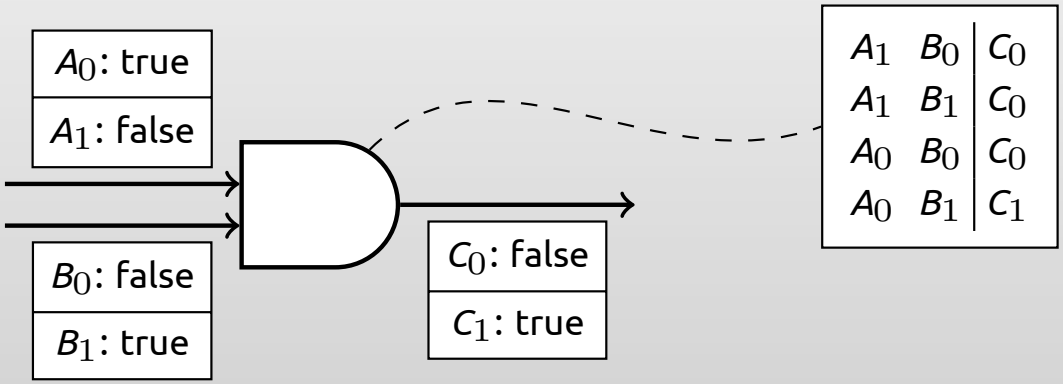
Garbled circuit technique [Yao86,BellareHoangRogaway12]



give “encrypted truth table” for each gate

why we like (boolean) circuits

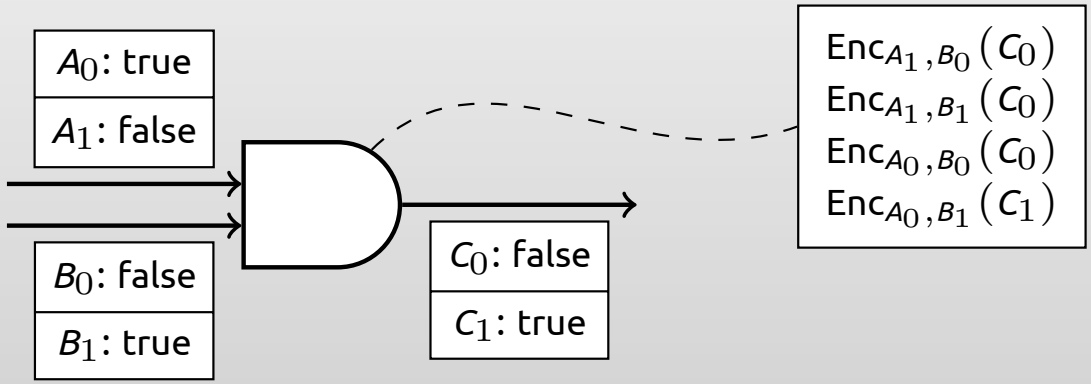
Garbled circuit technique [Yao86,BellareHoangRogaway12]



give “encrypted truth table” for each gate

why we like (boolean) circuits

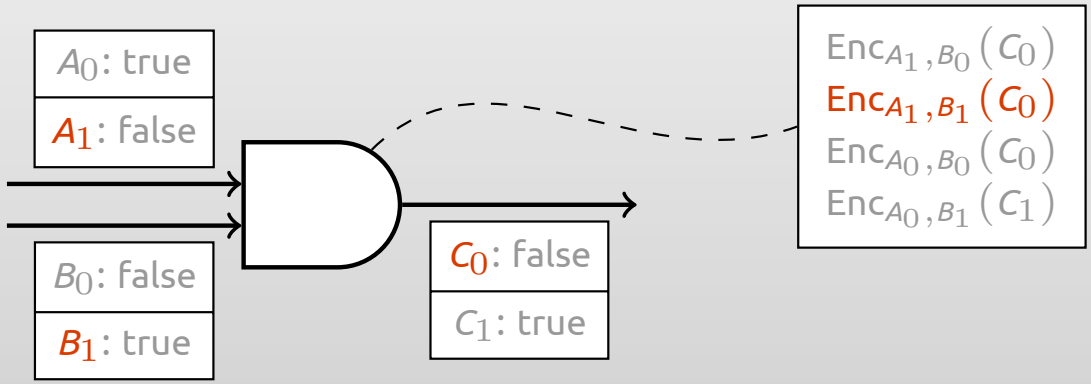
Garbled circuit technique [Yao86,BellareHoangRogaway12]



give “encrypted truth table” for each gate

why we like (boolean) circuits

Garbled circuit technique [Yao86,BellareHoangRogaway12]

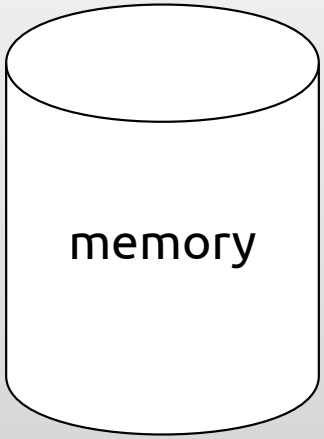


Informal security proof:

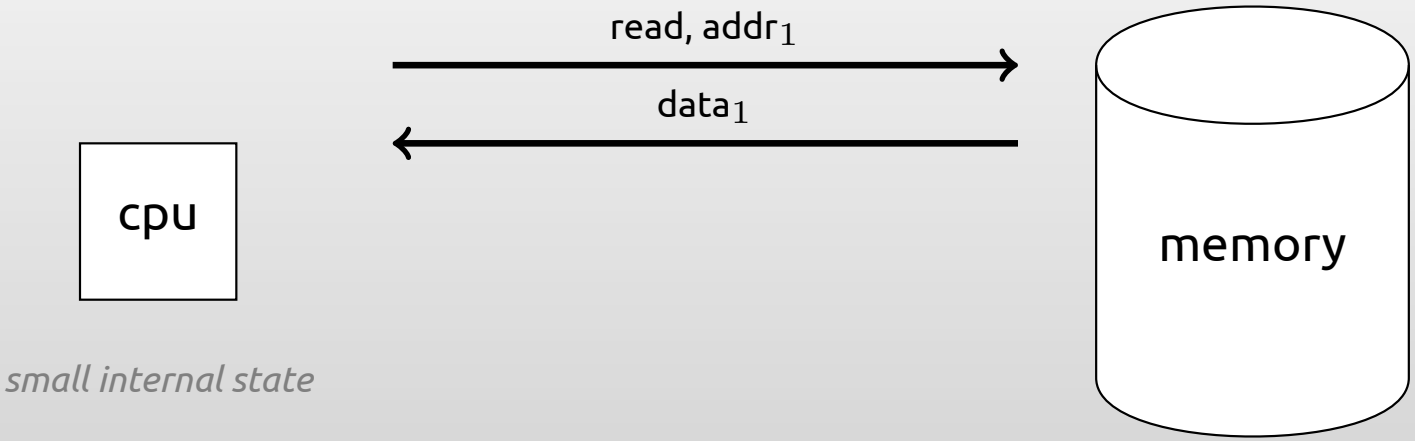
- Wire label leaks no information about logical value
- Receiver only learns one label for each wire (induction)

RAM programs

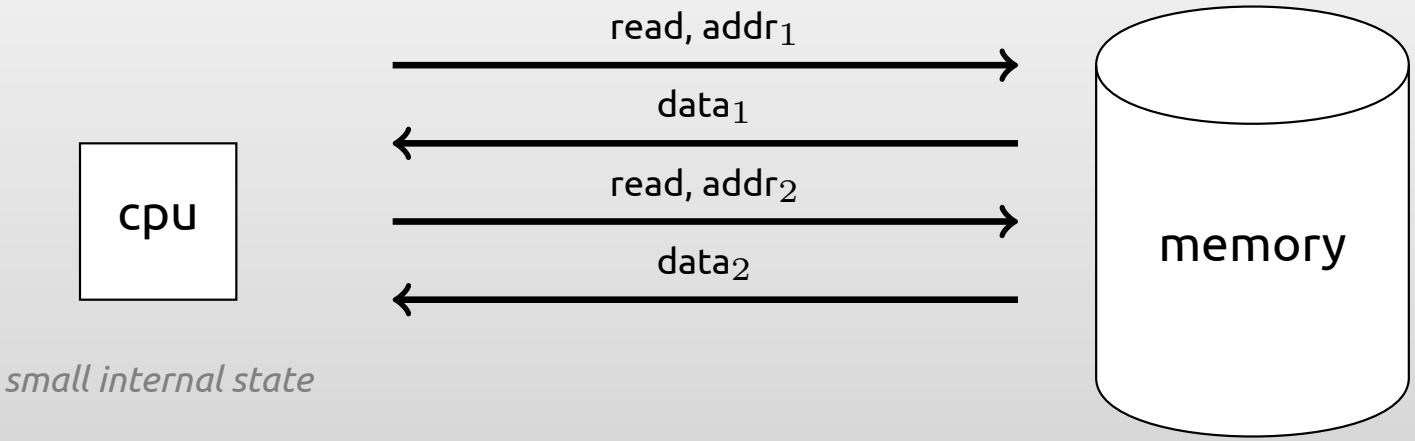
small internal state



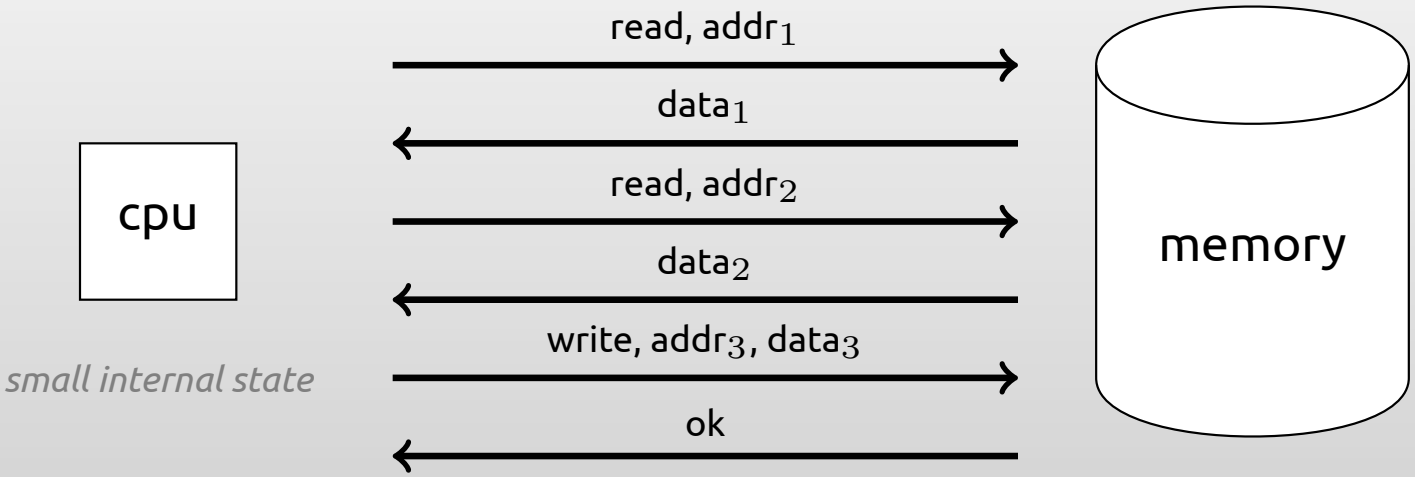
RAM programs



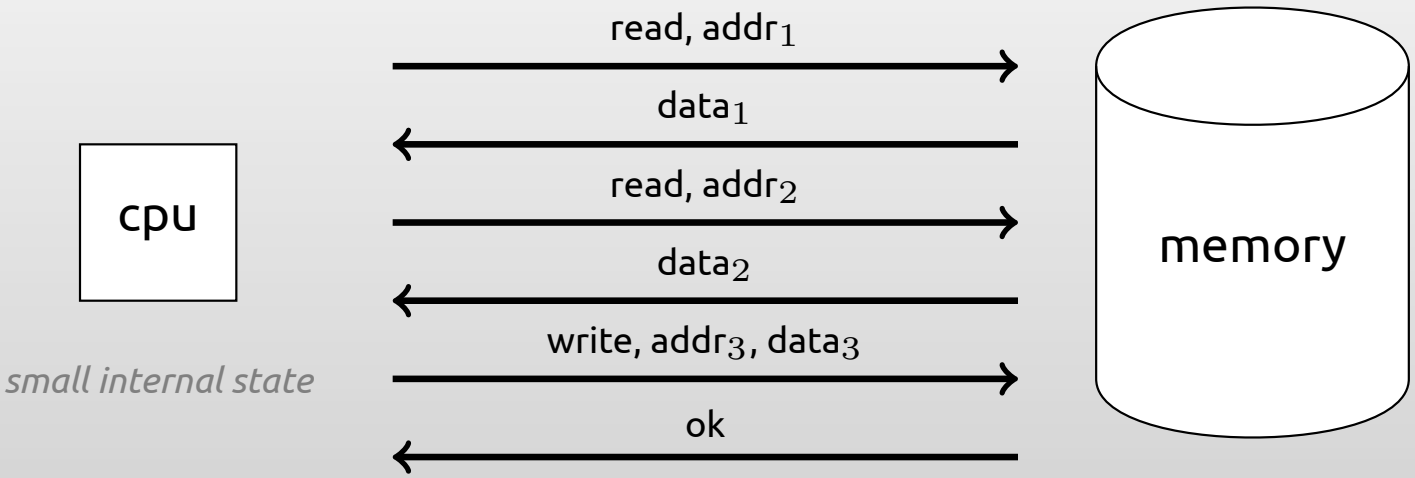
RAM programs



RAM programs



RAM programs



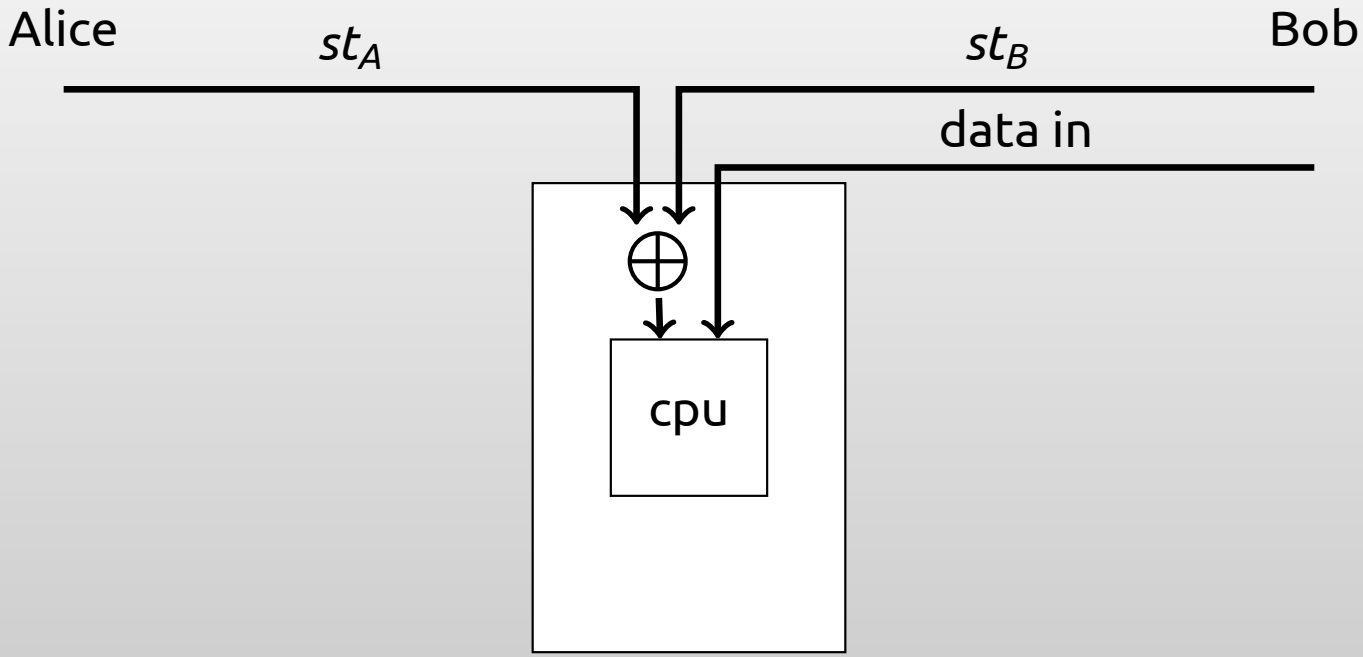
Oblivious RAM (ORAM) = memory access pattern leaks nothing about inputs/outputs/state [GoldreichOstrosvky96]

- Can make any RAM program oblivious, polylog overhead in runtime & memory [ShiChanStefanovLi11, ...]
- Must still “touch” all of memory, in initialization phase
- Our results only need “metadata-obliviousness” (R vs W, address)

semi-honest RAM-2PC [GKKKMRV12]

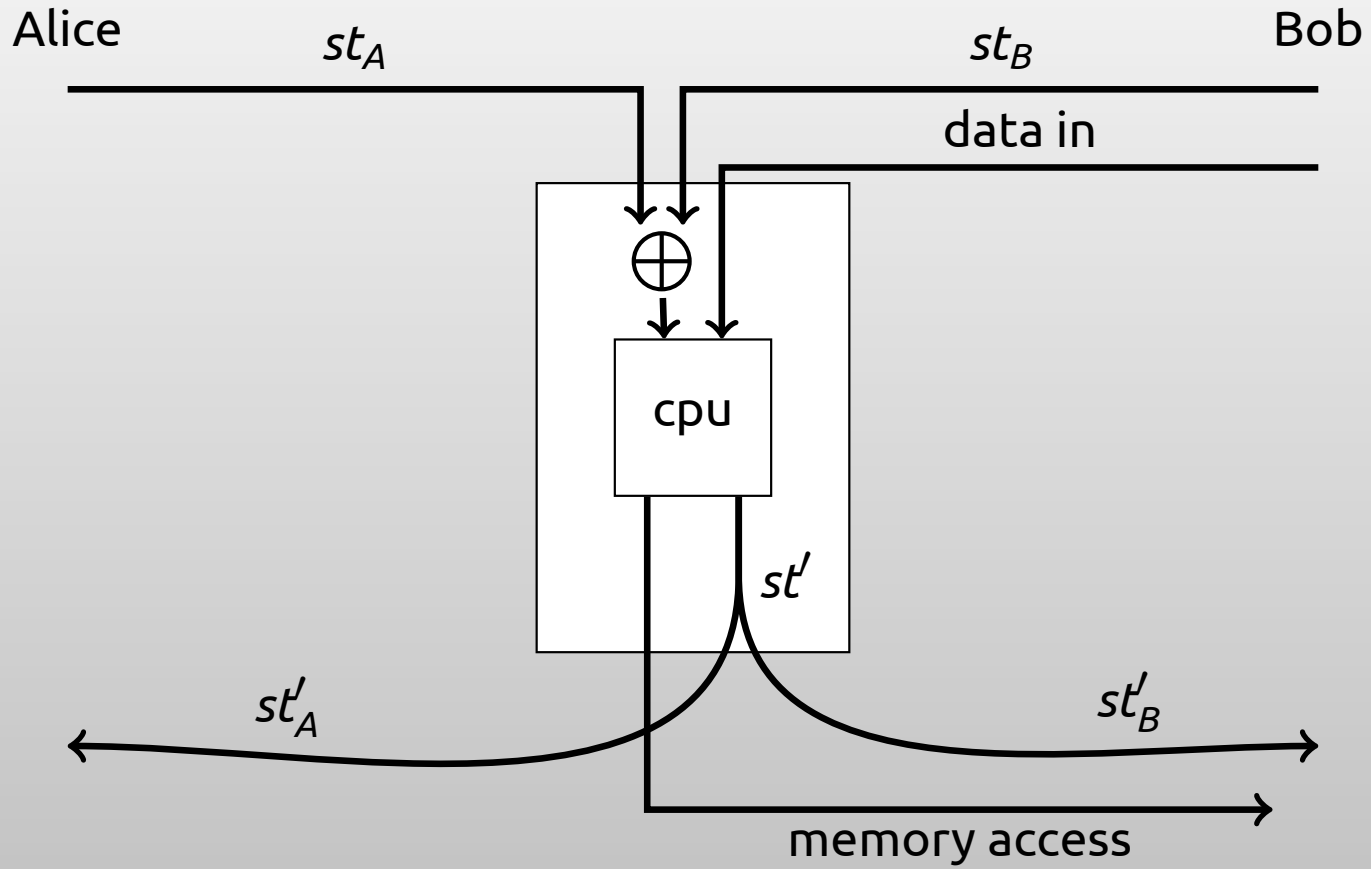
initialize memory; secret share initial CPU state

semi-honest RAM-2PC [GKKKMRV12]



secure 2pc of augmented CPU circuit

semi-honest RAM-2PC [GKKKMRV12]

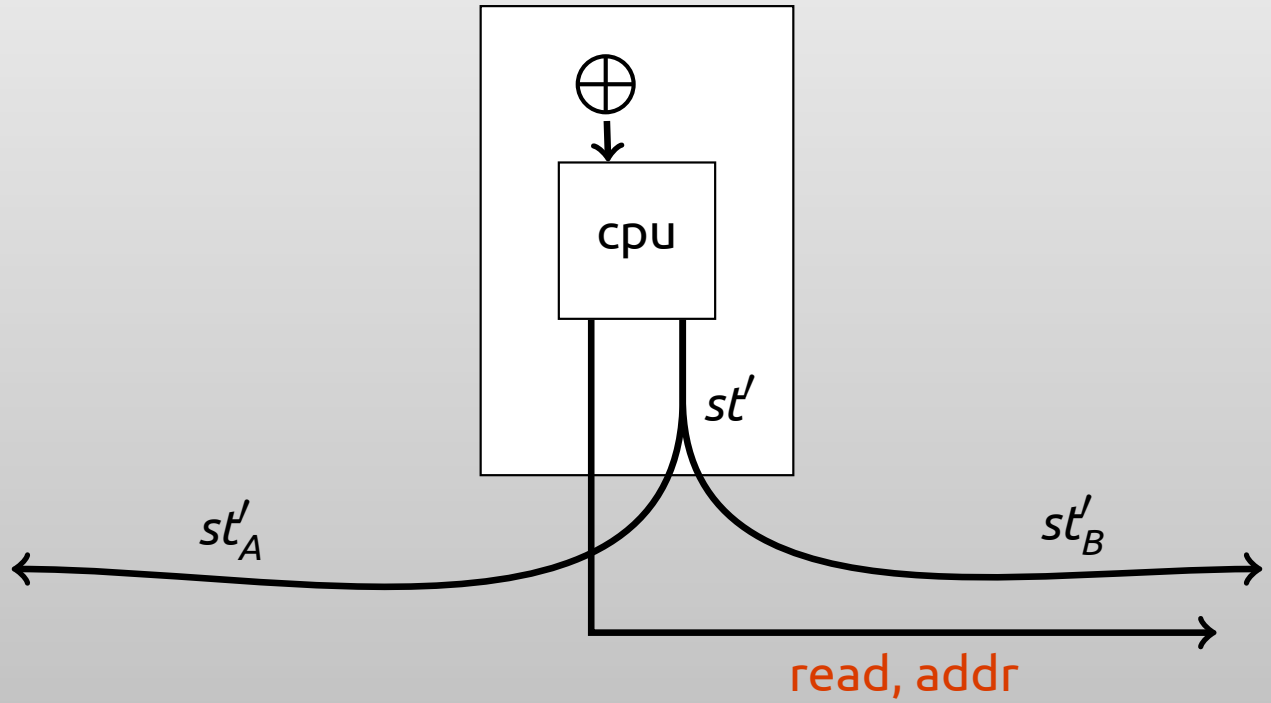


ORAM $\Rightarrow$ safe to let Bob handle all memory access

semi-honest RAM-2PC [GKKKMRV12]

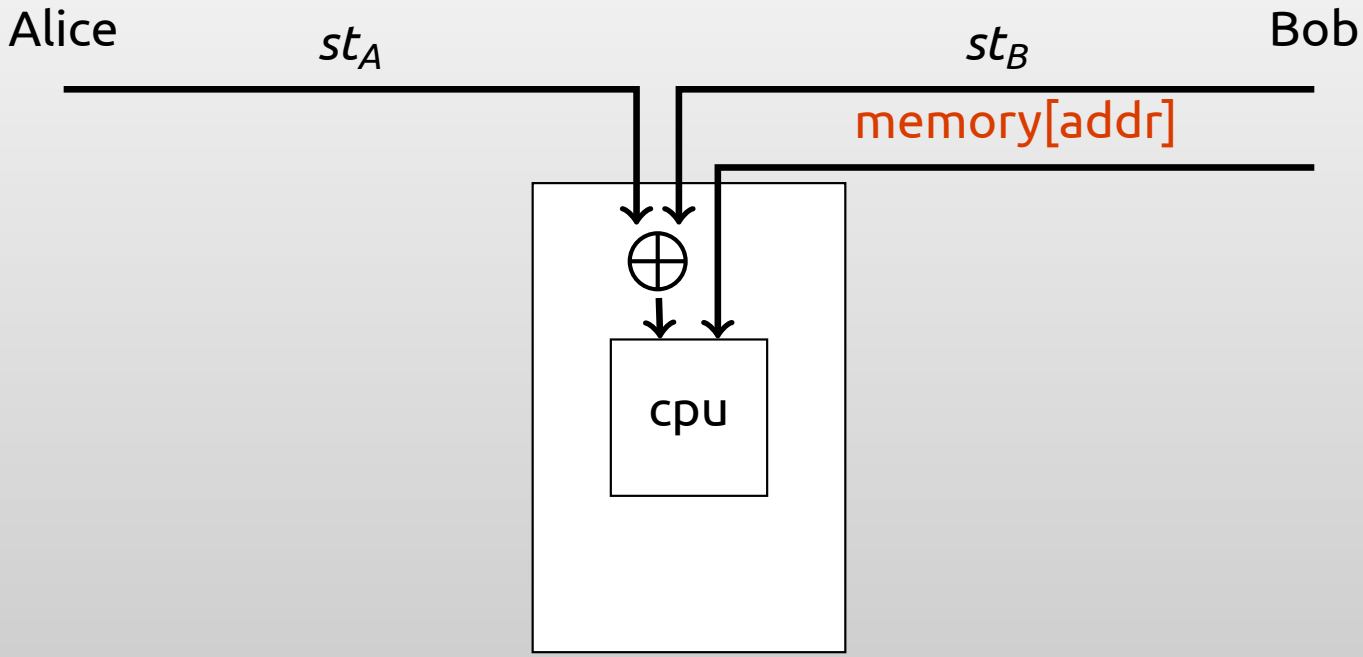
Alice

Bob



example: CPU wants to read

semi-honest RAM-2PC [GKKKMRV12]

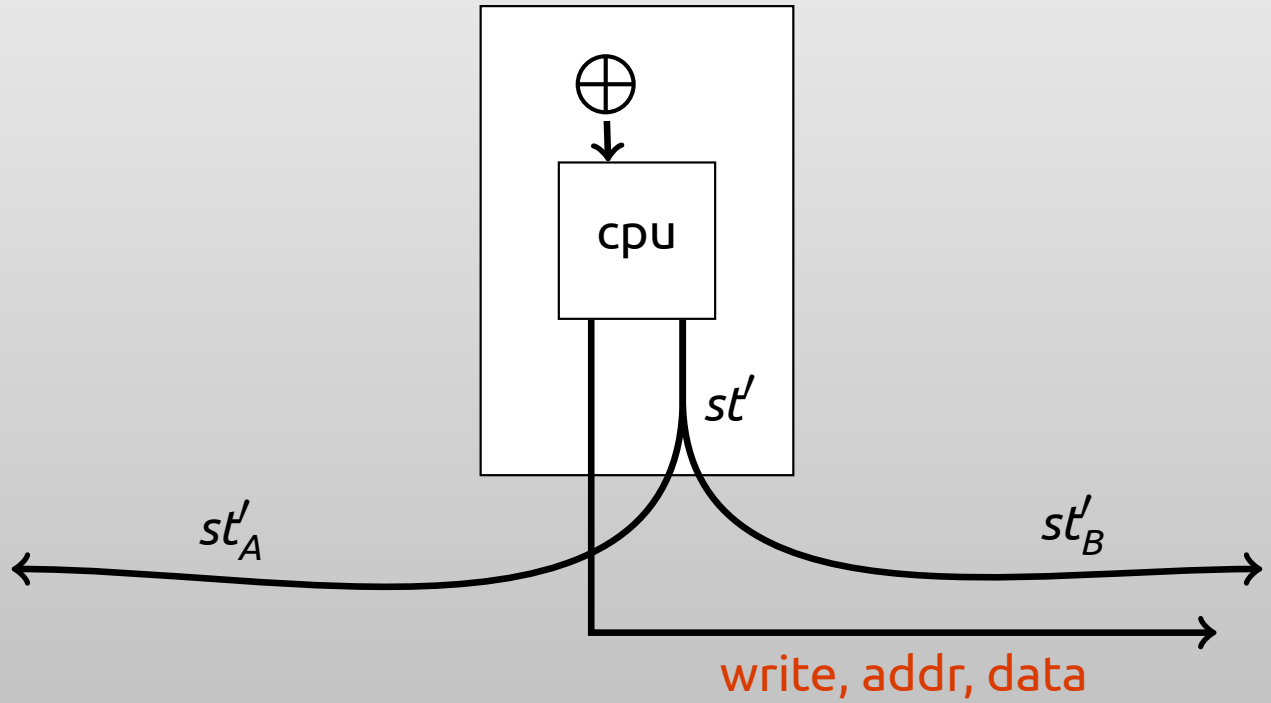


example: CPU wants to read

semi-honest RAM-2PC [GKKKMRV12]

Alice

Bob

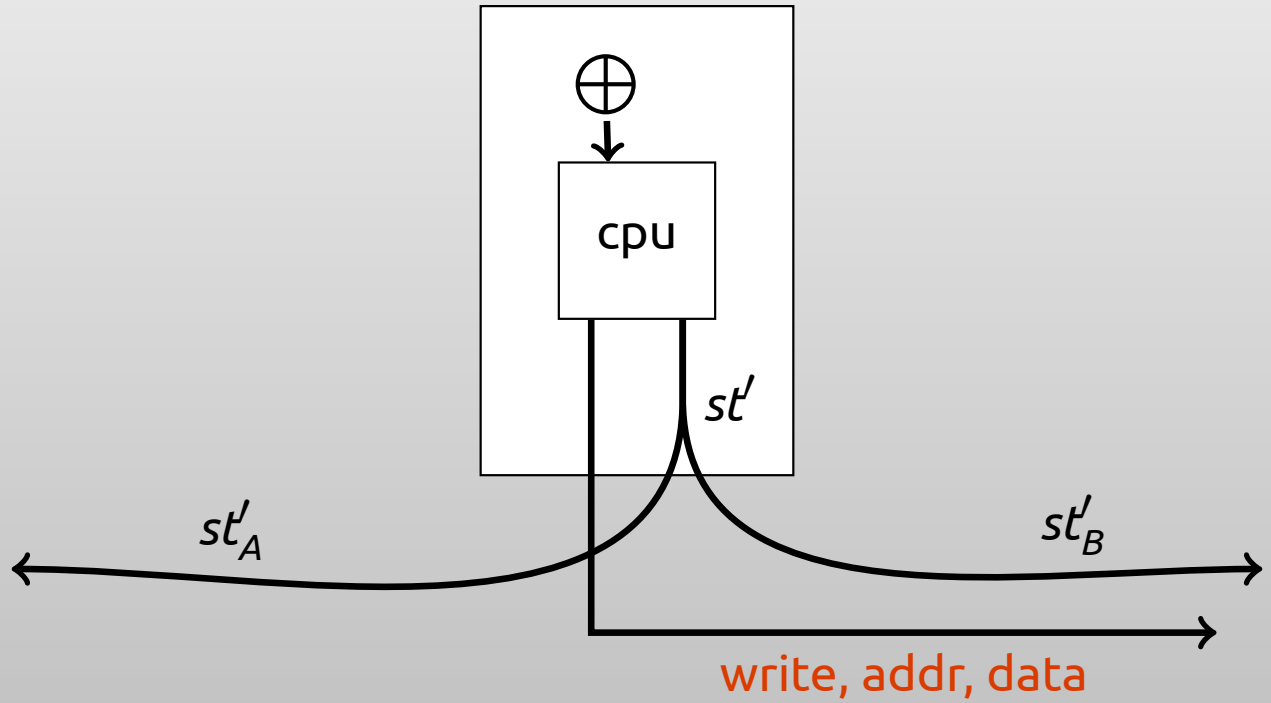


example: CPU wants to write

semi-honest RAM-2PC [GKKKMRV12]

Alice

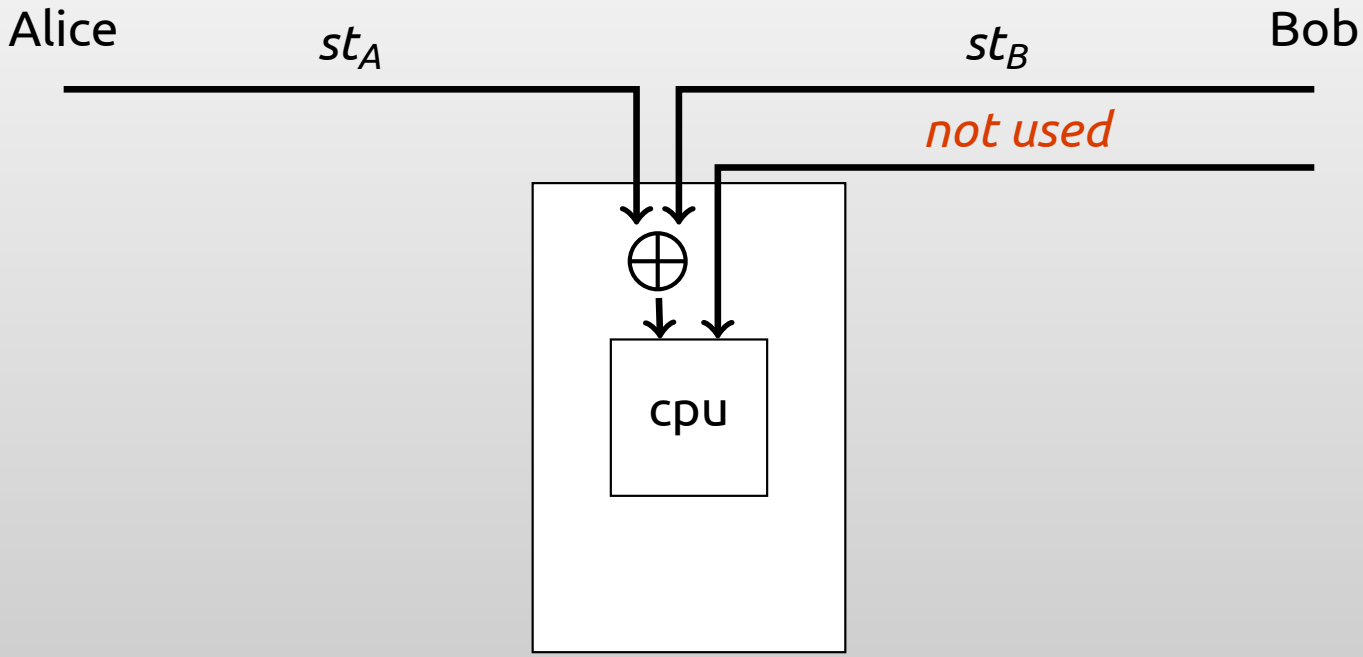
Bob



memory[addr] := data

example: CPU wants to write

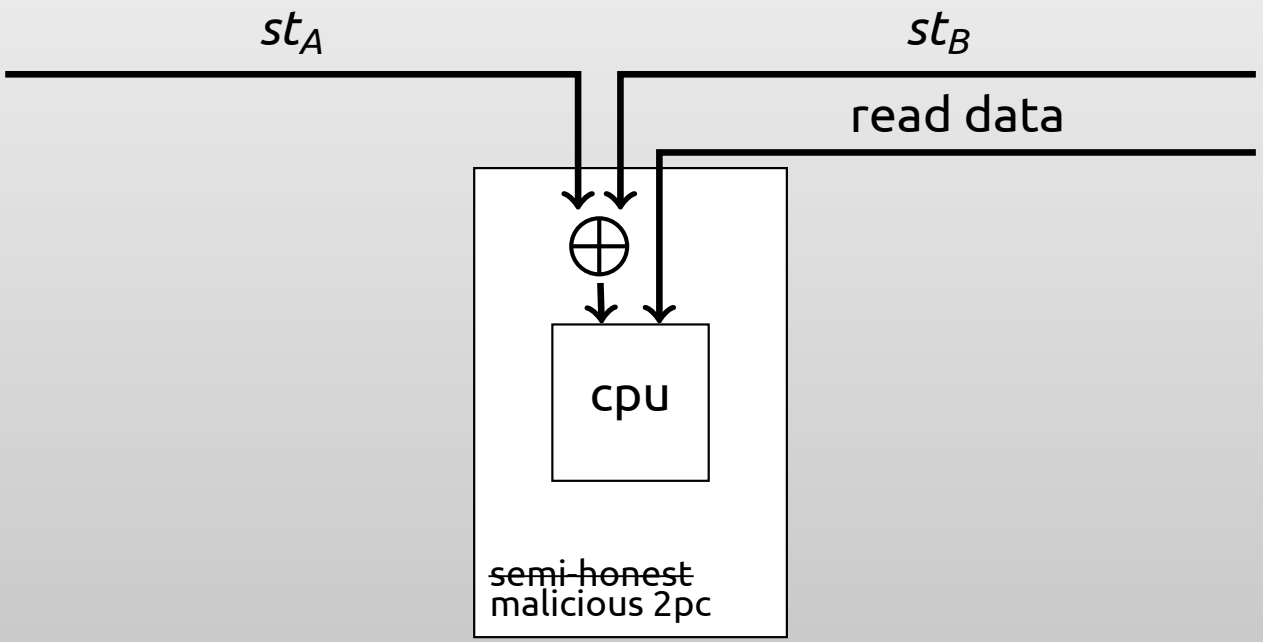
semi-honest RAM-2PC [GKKKMRV12]



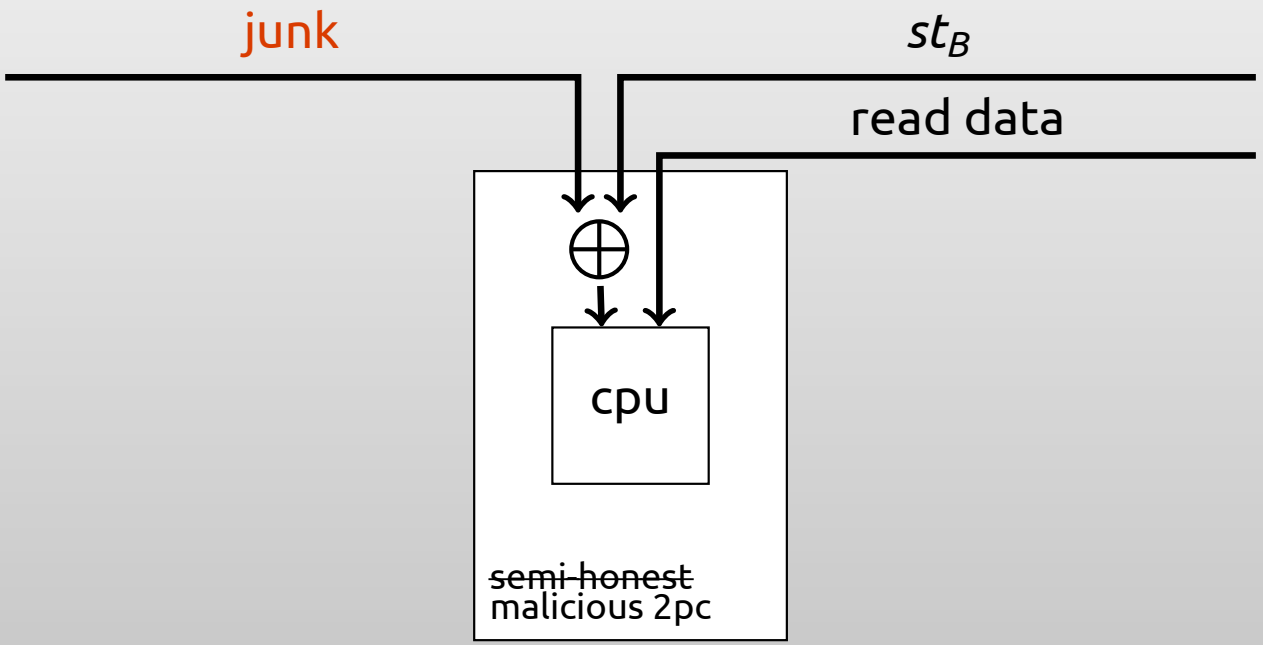
example: CPU wants to write

malicious-security pitfalls

malicious-security pitfalls

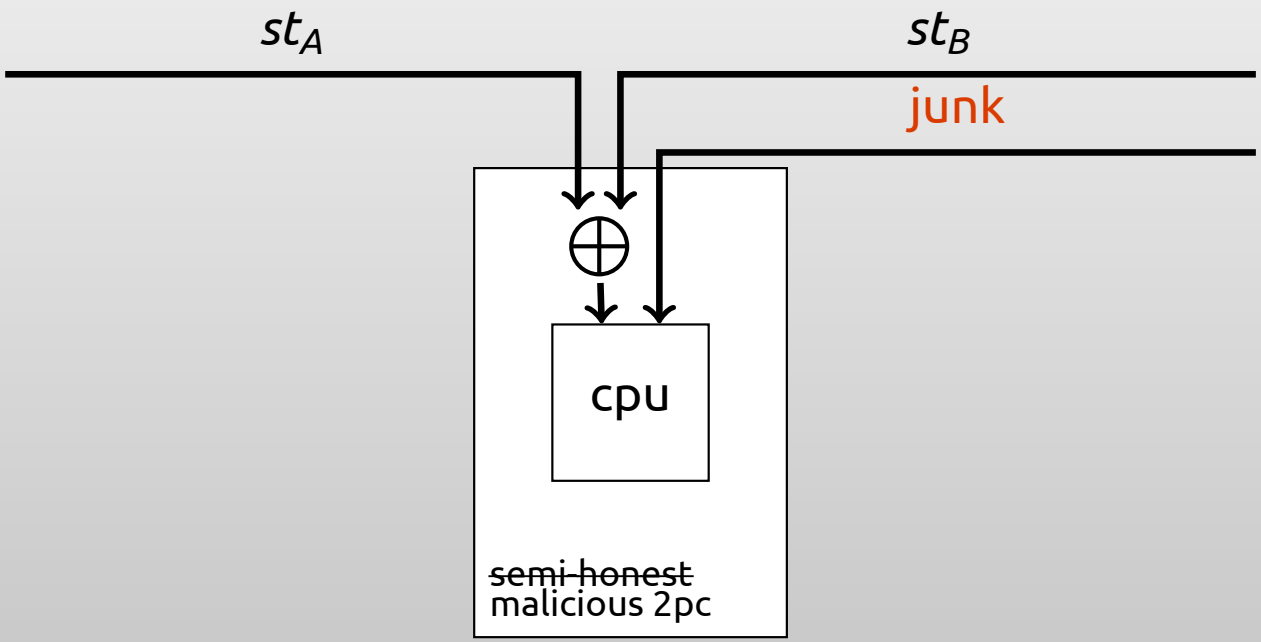


malicious-security pitfalls



Integrity of state

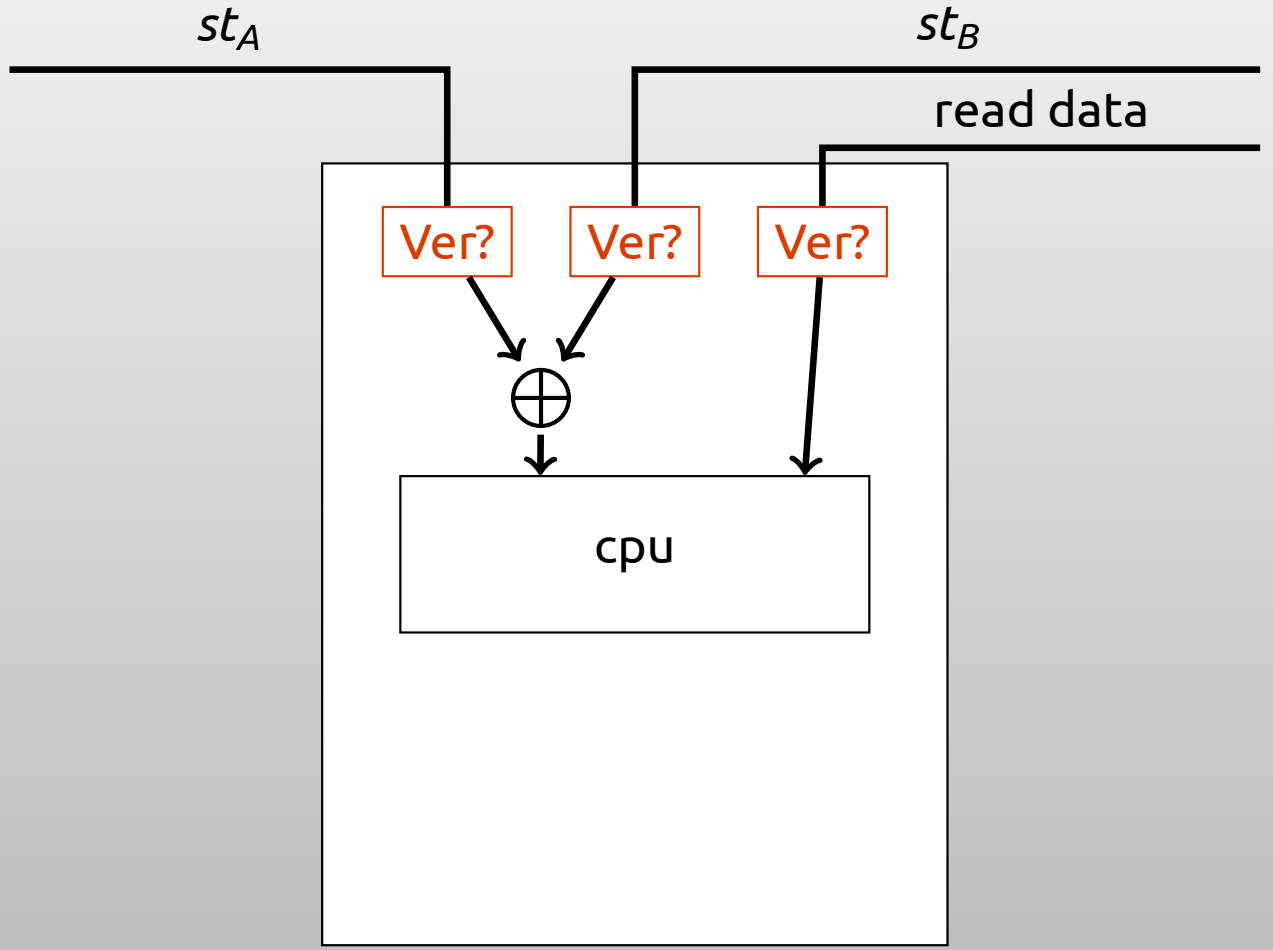
malicious-security pitfalls



Integrity of state and memory!

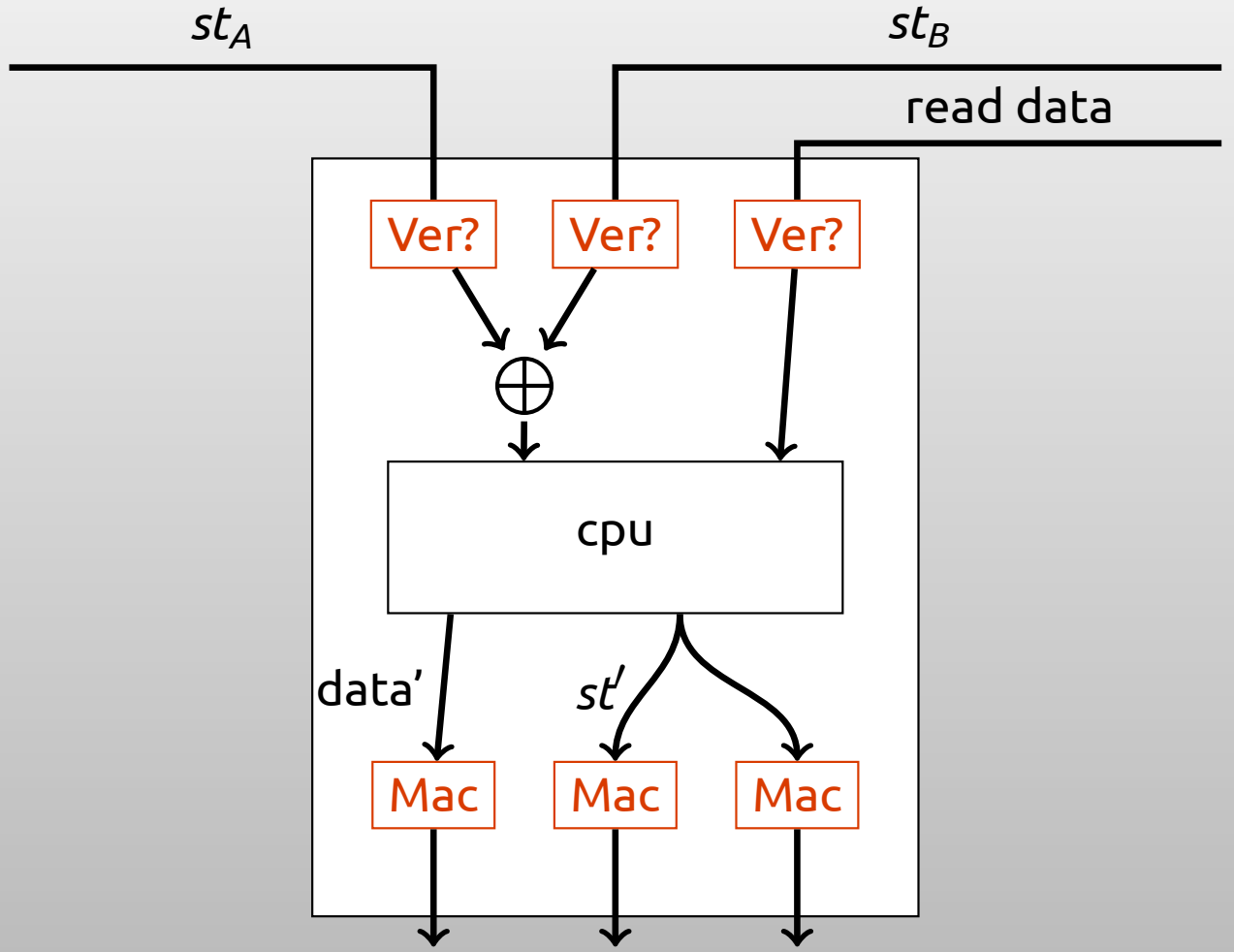
use a MAC?

use a MAC?



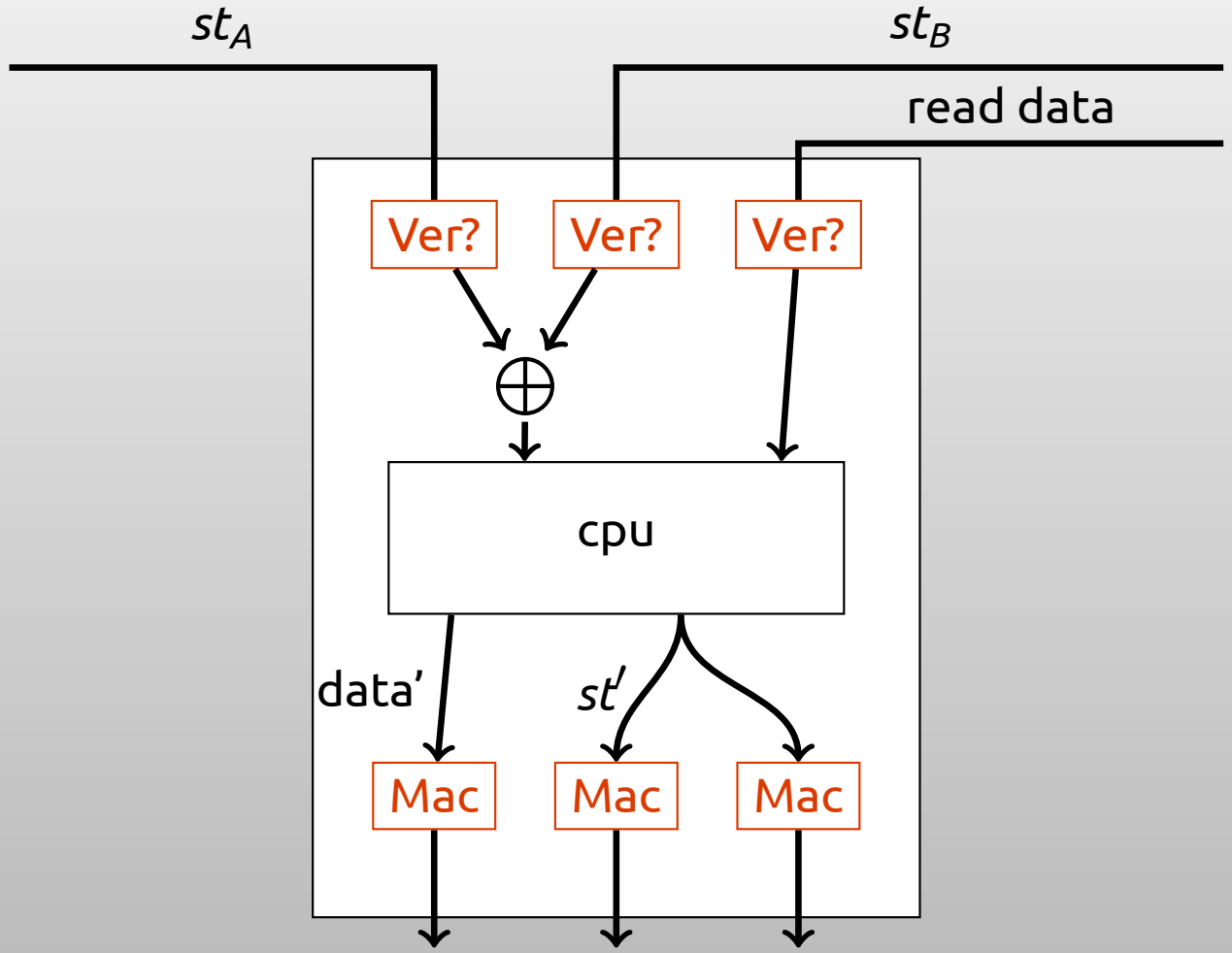
This will work, but ...

use a MAC?



This will work, but ...

use a MAC?



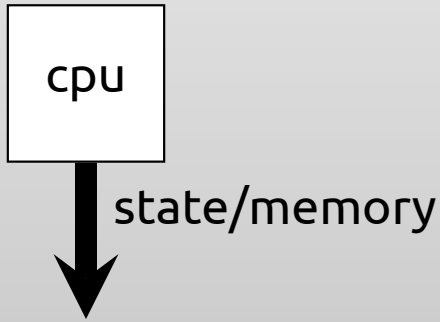
This will work, but ...

- Crypto circuitry inside garbled circuit
- Many inputs (MAC tags & keys) to garbled circuit

a better idea. . .

State & memory information should be:

1. Kept private
2. Protected from tampering



a better idea. . .

State & memory information should be:

1. Kept private
2. Protected from tampering

a better idea. . .

State & memory information should be:

1. Kept private ✓
2. Protected from tampering ✓

a better idea. . .

State & memory information should be:

1. Kept private ✓
2. Protected from tampering ✓

Key Idea

Directly reuse garbled values for state & memory!

overview

overview

Goals:

- malicious security
- no extra overhead inside garbled circuits
- efficiency matching state-of-the-art for circuit-based 2PC

overview

Goals:

- malicious security
- no extra overhead inside garbled circuits
- efficiency matching state-of-the-art for circuit-based 2PC

Results:

style	cost	technique
streaming	$s \times$ semi-honest	blind cut-and-choose, forge-and-lose
online/offline	$\sim 2s/\log T \times$ semi-honest	batched cut-and-choose, LEGO

for security 2^{-s}

T = ORAM running time

overview

Goals:

- malicious security
- no extra overhead inside garbled circuits
- efficiency matching state-of-the-art for circuit-based 2PC

Results:

style	cost	technique
streaming	$s \times$ semi-honest	blind cut-and-choose, forge-and-lose
online/offline	$\sim 2s/\log T \times$ semi-honest	batched cut-and-choose, LEGO

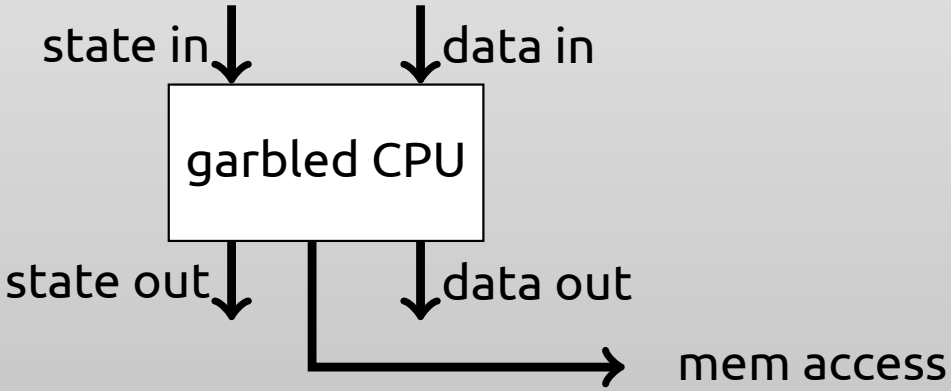
for security 2^{-s}

T = ORAM running time

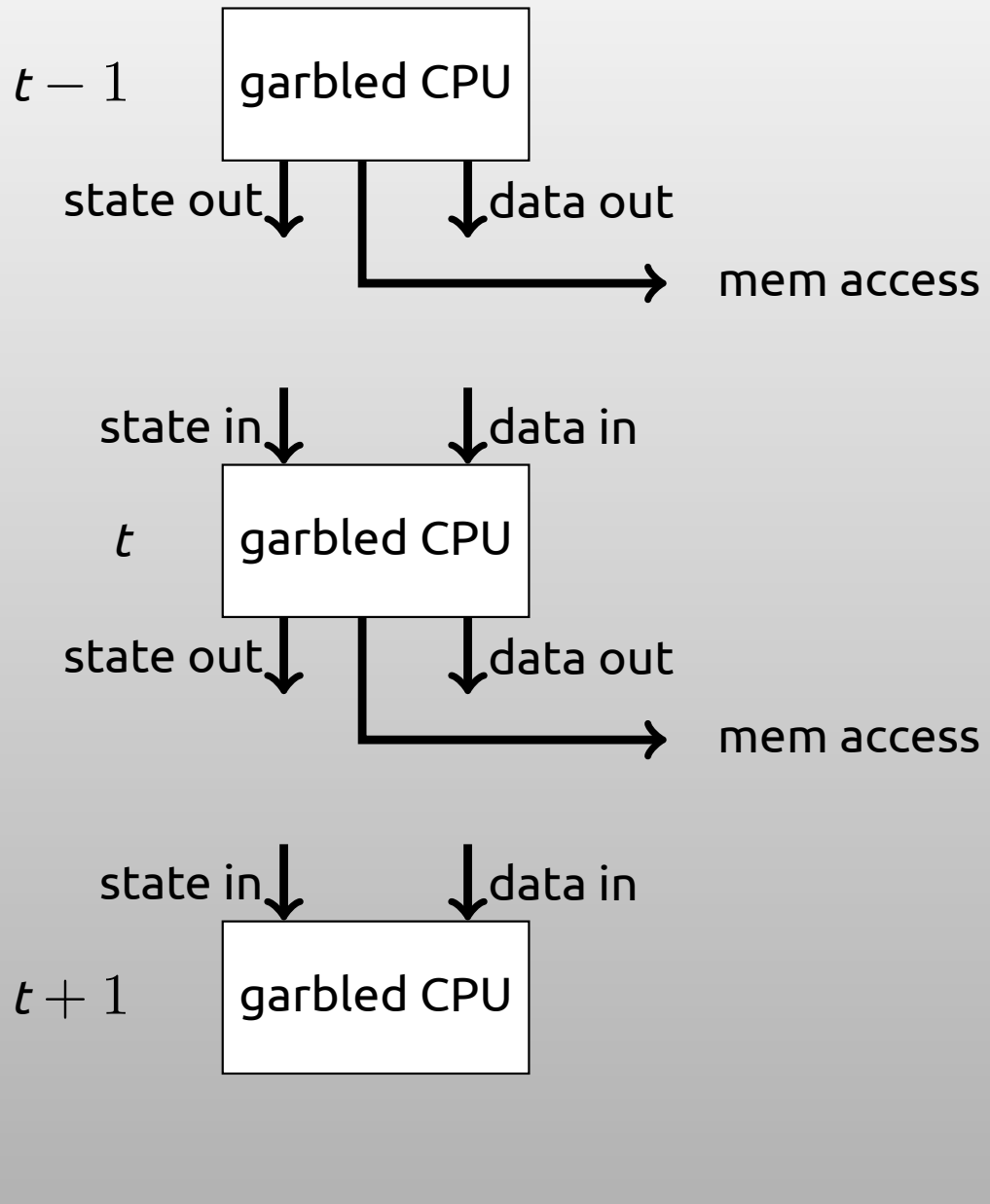
Theme:

- Re-use wire labels between evaluations of garbled CPU circuit

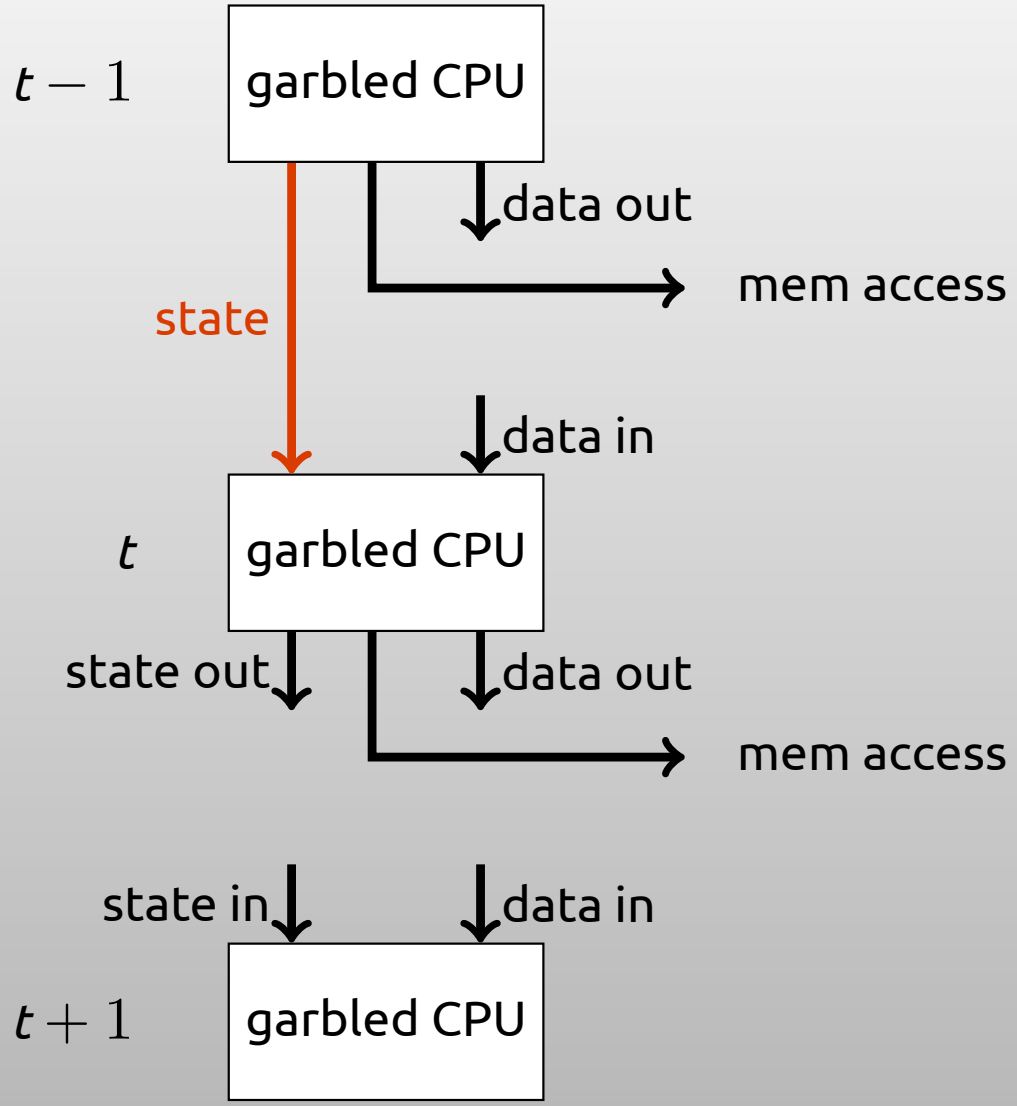
reusing wire labels: overview



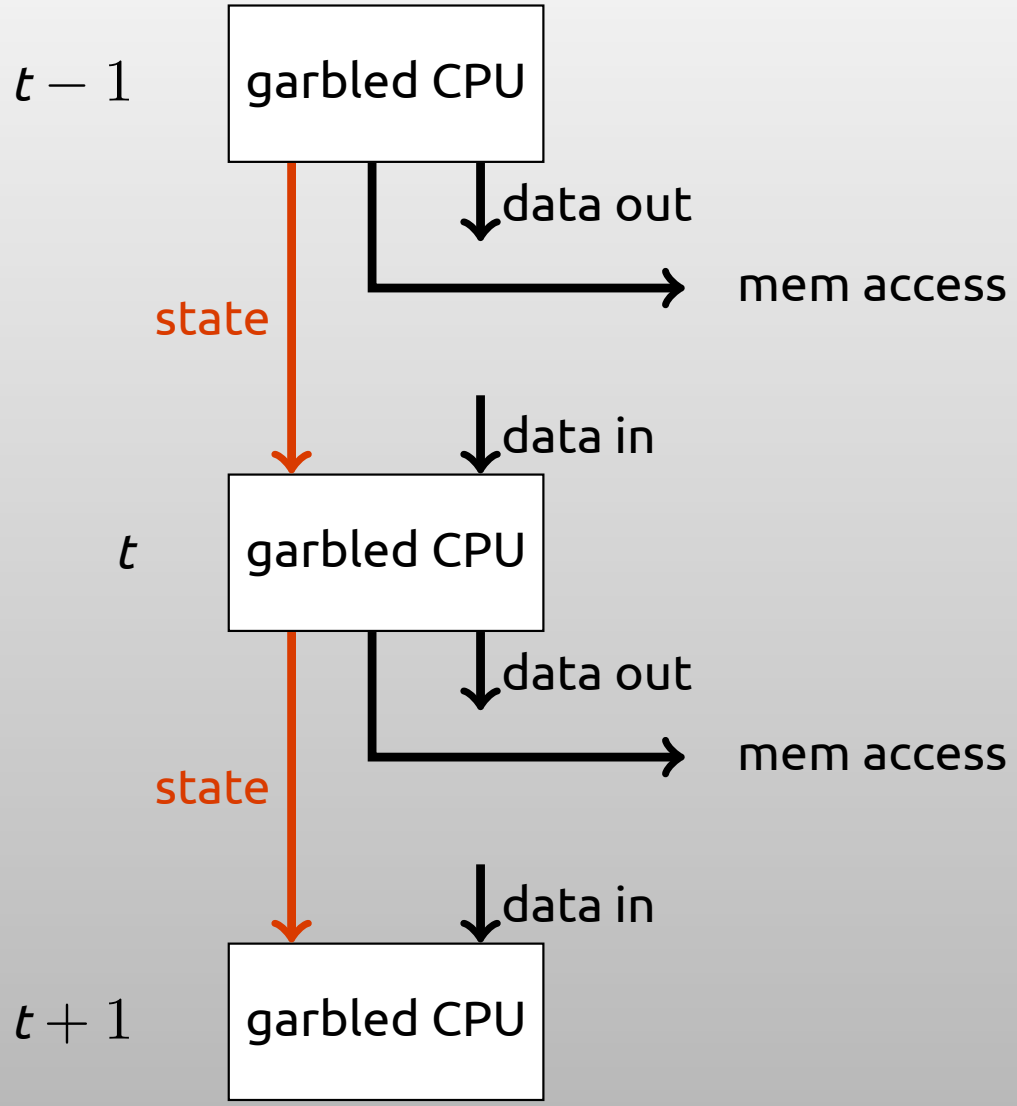
reusing wire labels: overview



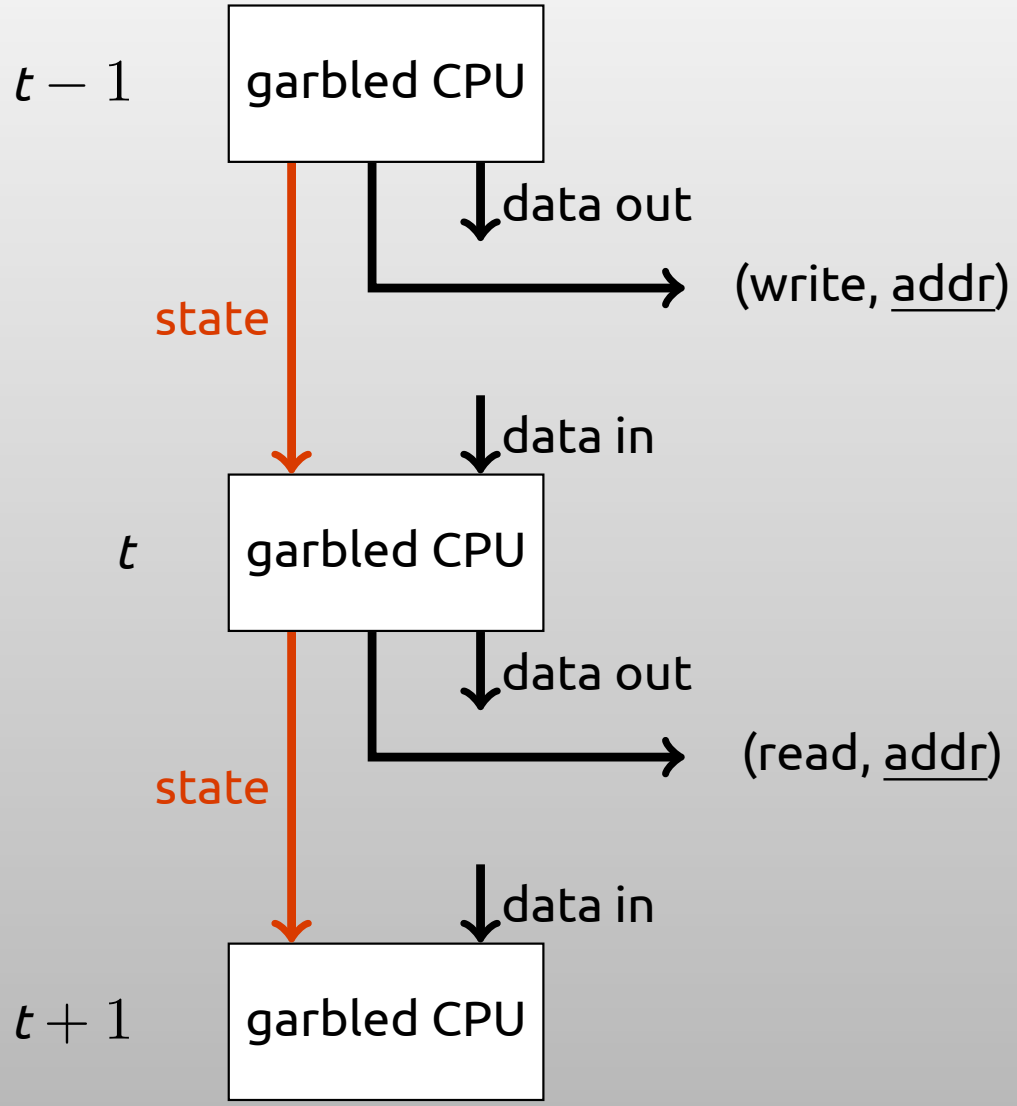
reusing wire labels: overview



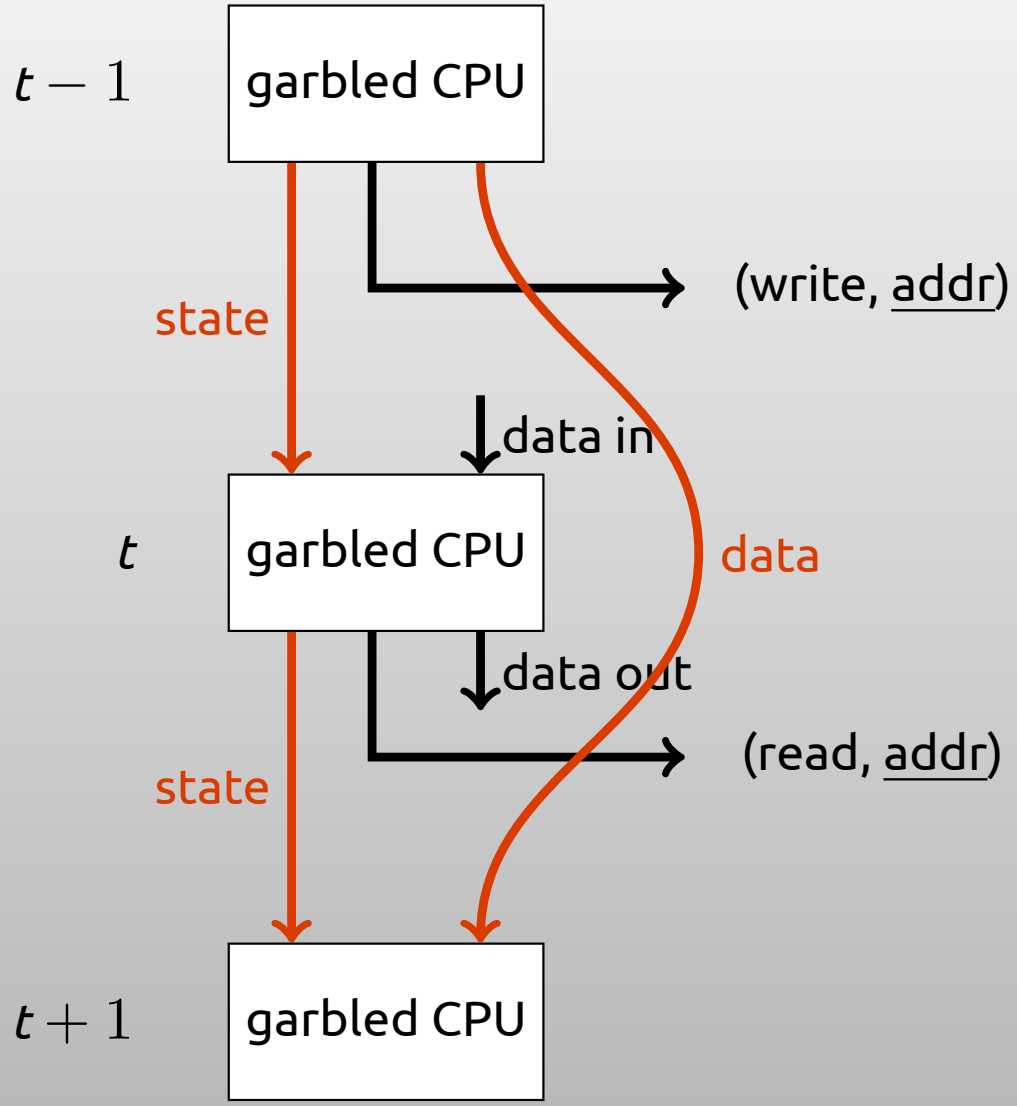
reusing wire labels: overview



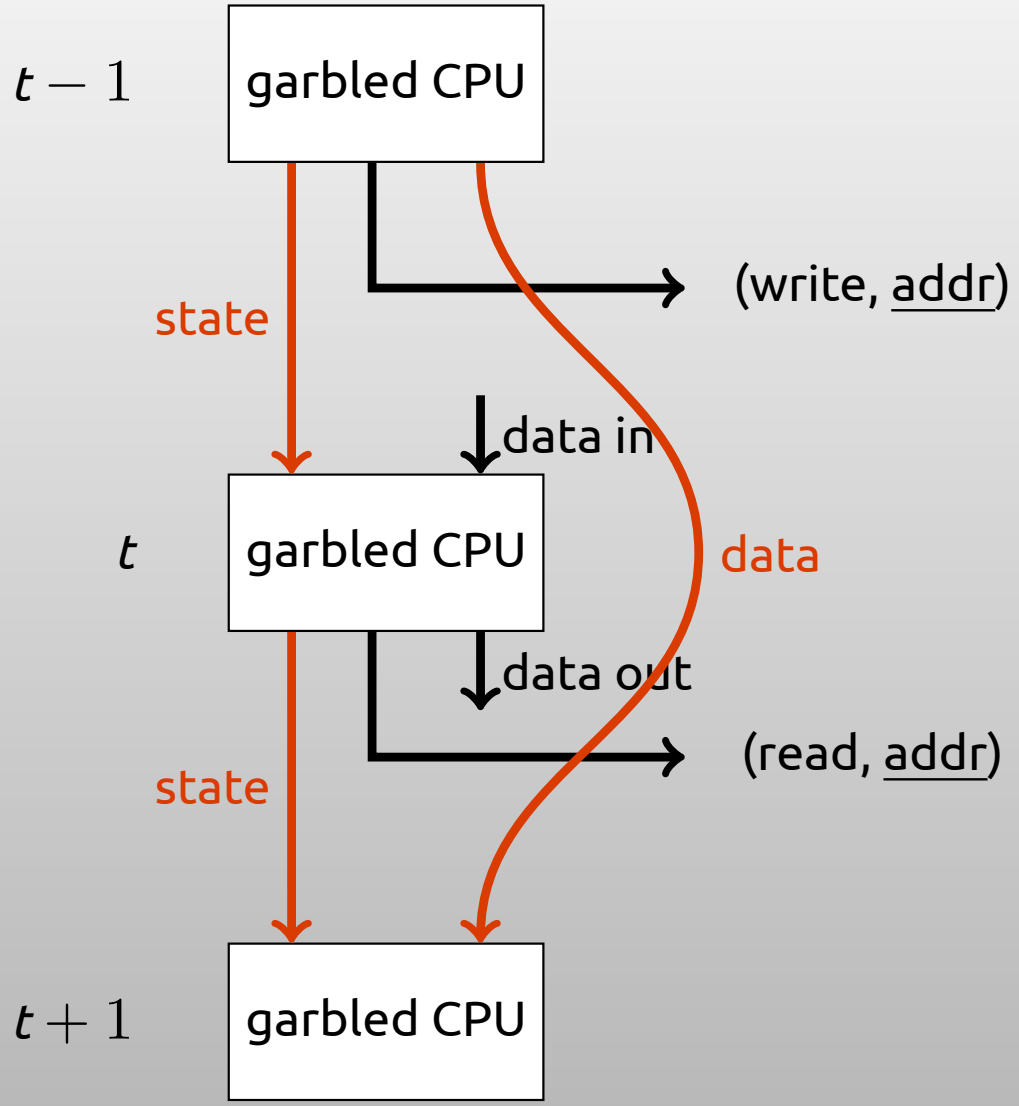
reusing wire labels: overview



reusing wire labels: overview

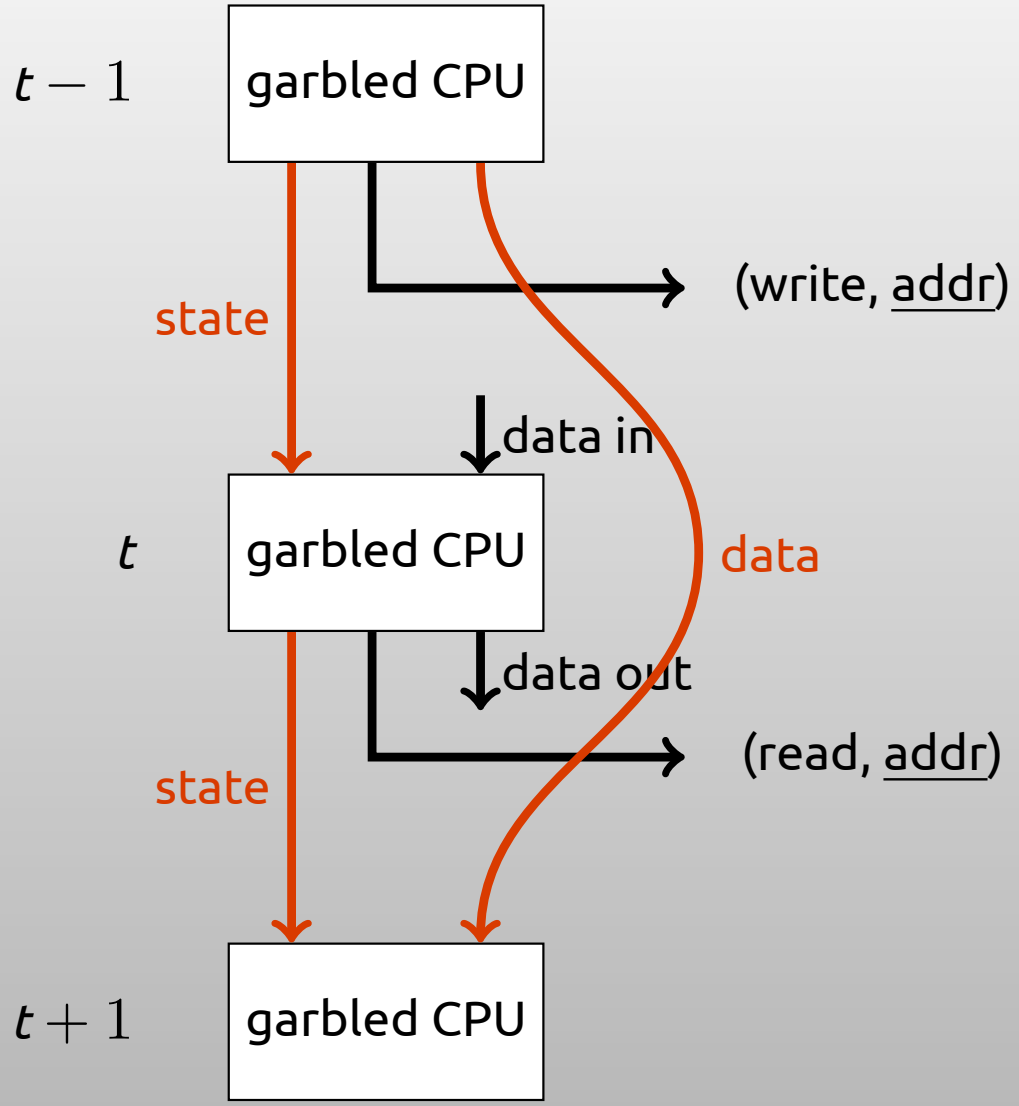


reusing wire labels: overview



Quiz: isn't this the same as making a monolothic garbled circuit for unrolled RAM computation?

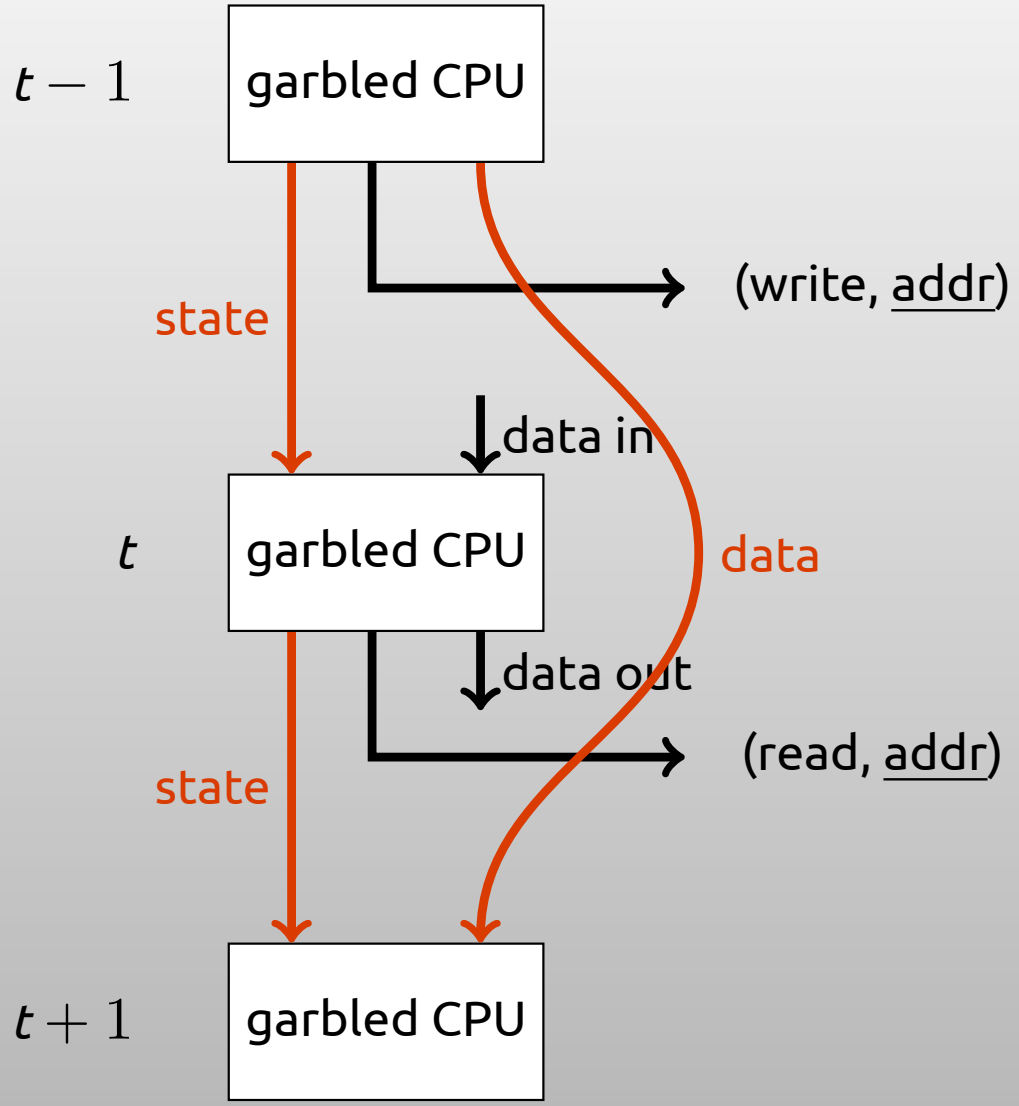
reusing wire labels: overview



Quiz: isn't this the same as making a monolothic garbled circuit for unrolled RAM computation?

- connections between "data in/out" determined at runtime!
- later inputs can depend on prior outputs

reusing wire labels: overview



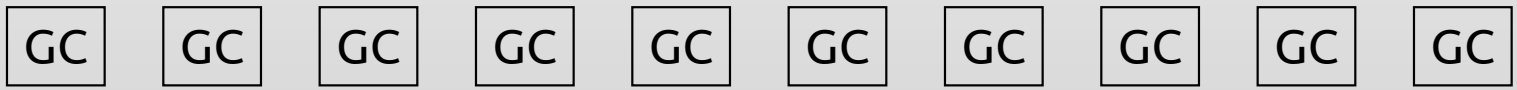
Quiz: isn't this the same as making a monolothic garbled circuit for unrolled RAM computation?

- connections between "data in/out" determined at runtime!
- later inputs can depend on prior outputs

Note: CPU need not encrypt data!

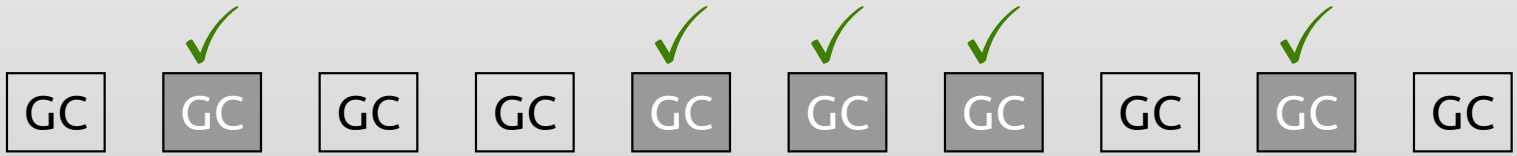
how to do cut-and-choose?

generate many garbled circuits



how to do cut-and-choose?

open & check some fraction of them; abort if any are bad



how to do cut-and-choose?

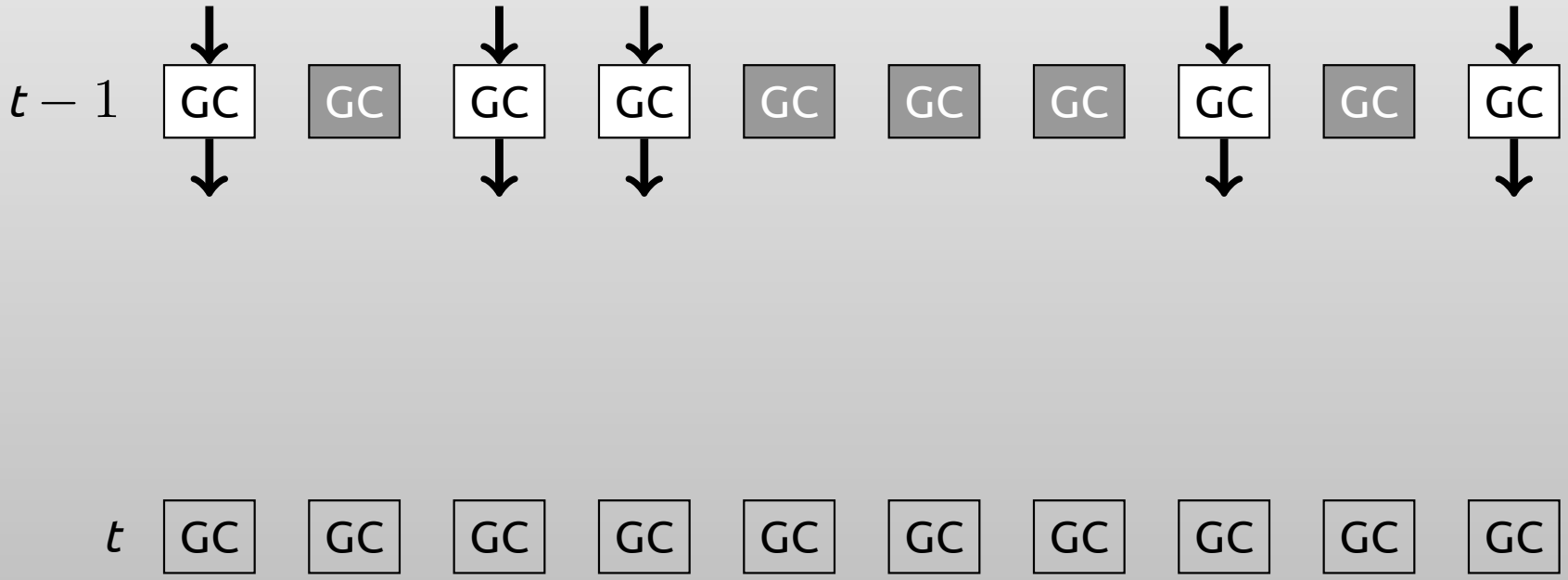
evaluate remaining circuits; take majority output

how to do cut-and-choose?

what about in the RAM setting?

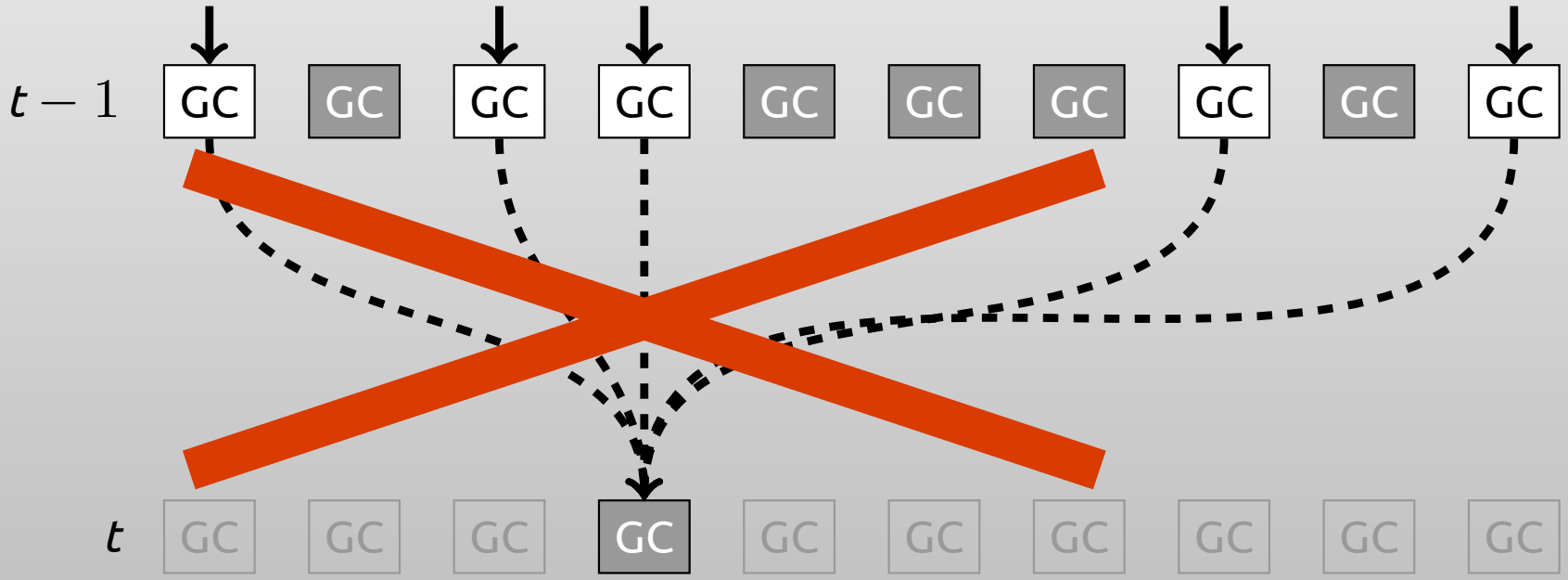
t

how to do cut-and-choose?



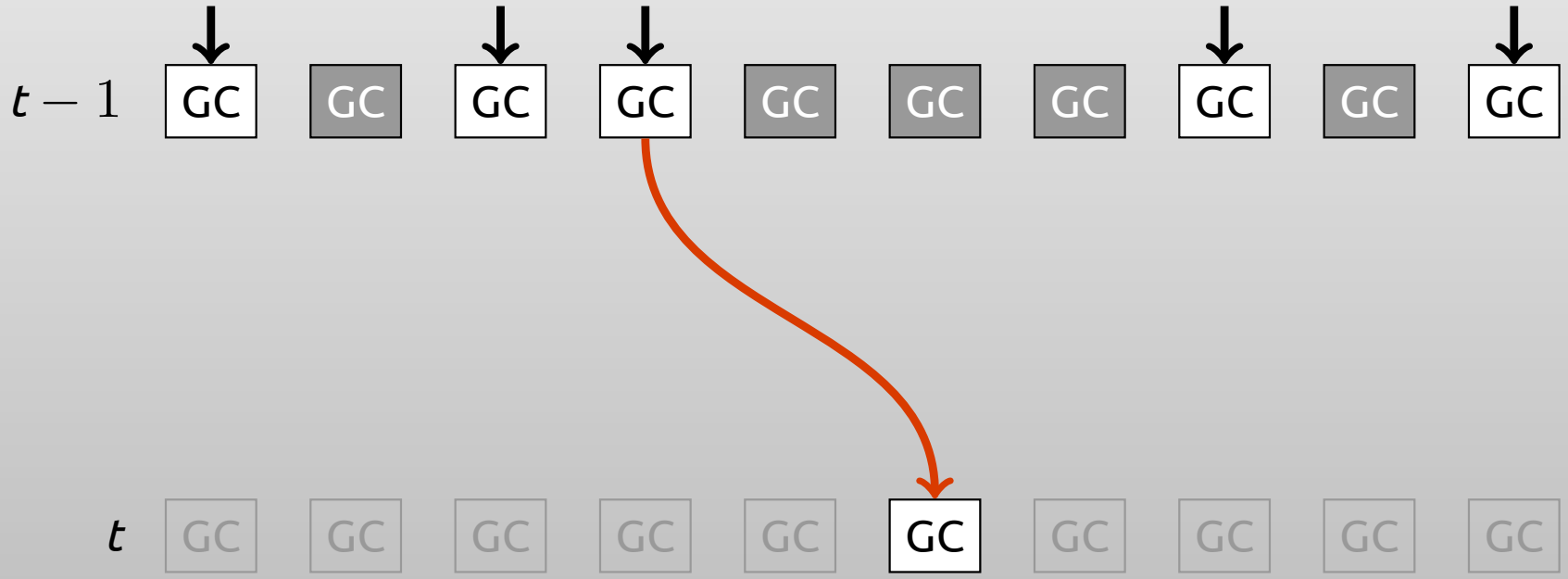
how to do cut-and-choose?

check circuit **cannot** share wire labels! (secrets revealed)



how to do cut-and-choose?

eval circuit **must** share wire labels!

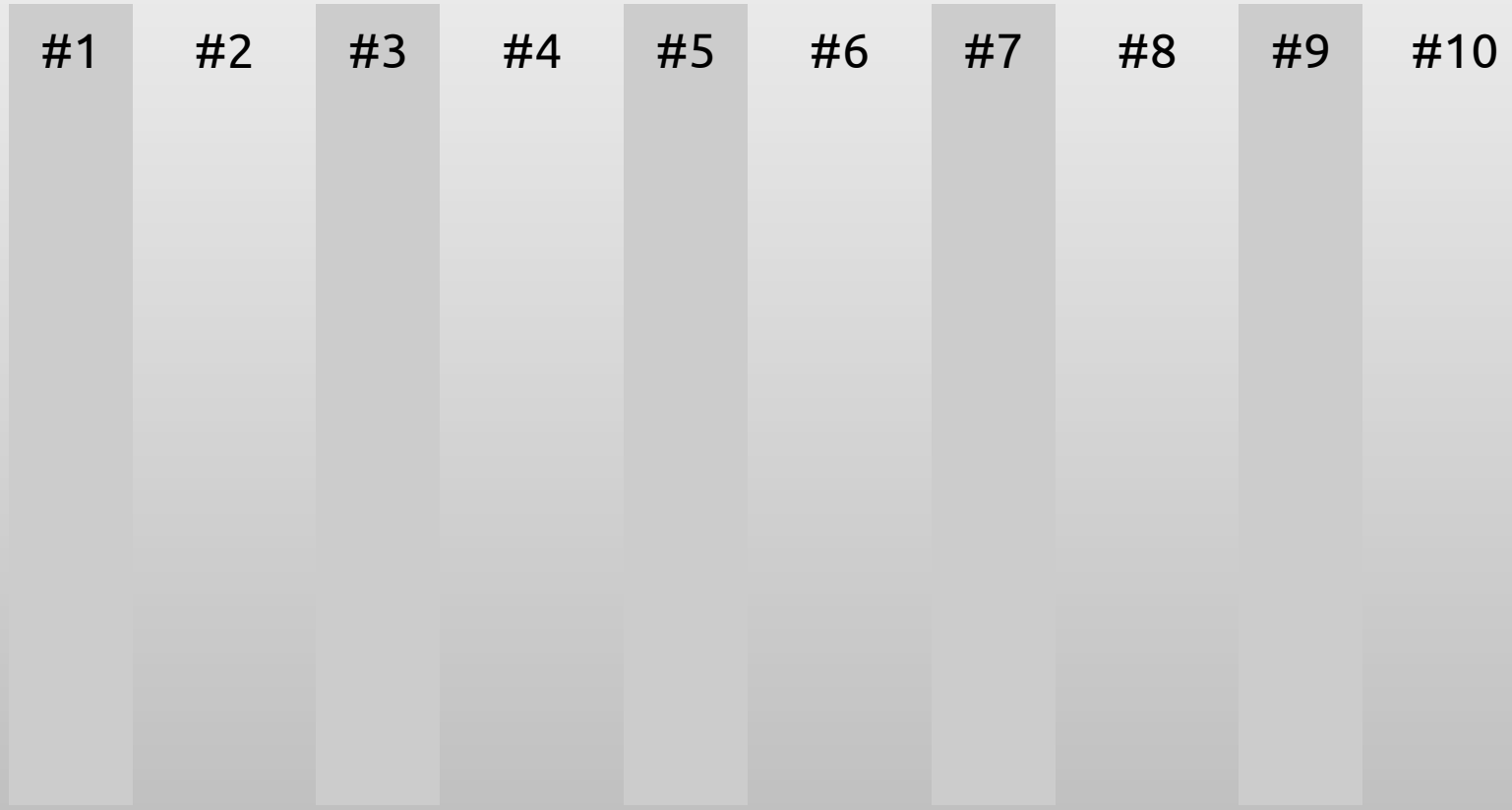


how to do cut-and-choose?

can't predict check/eval when generating garbled circuits!

our approach

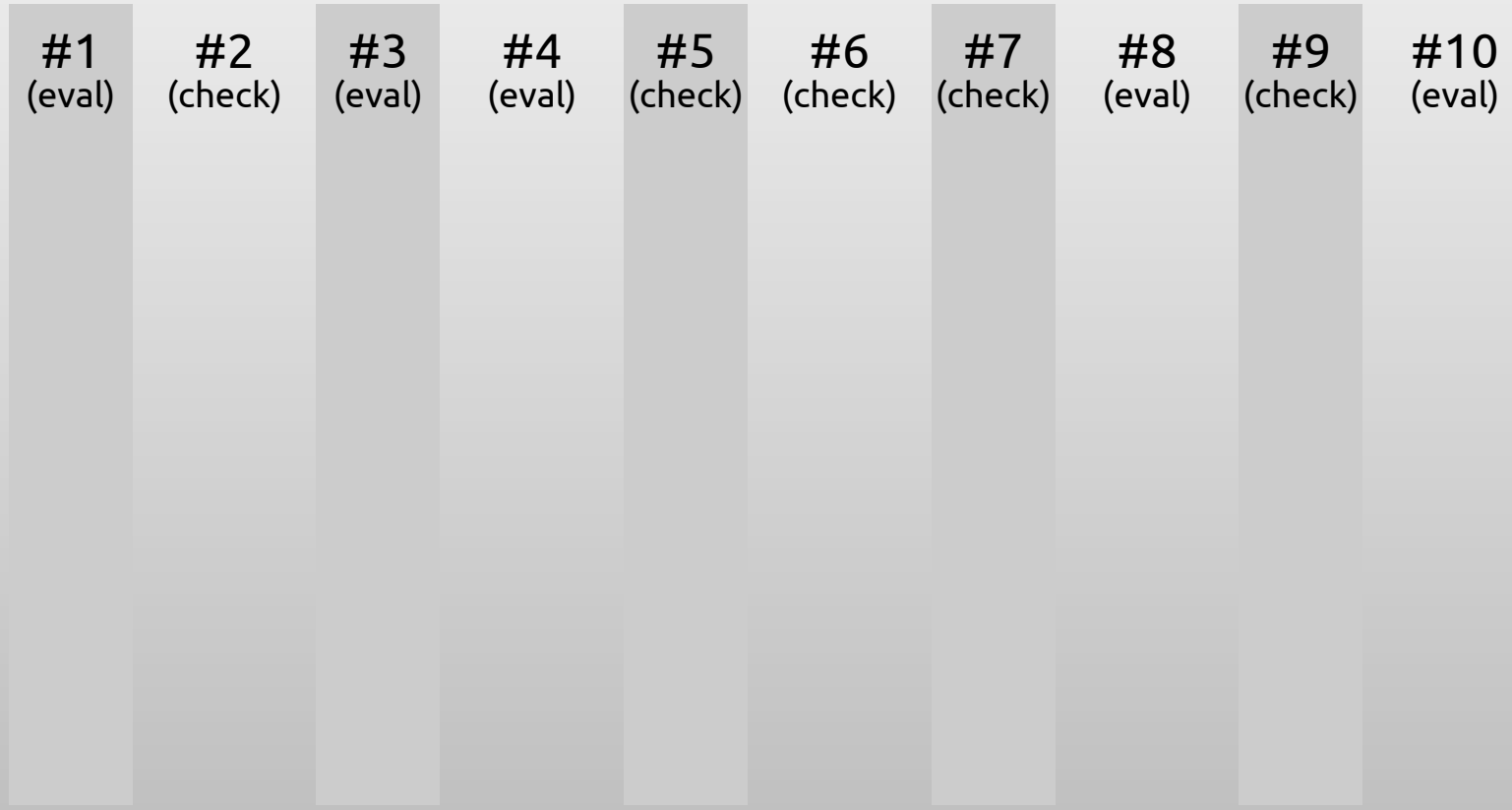
Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]



establish many threads of computation

our approach

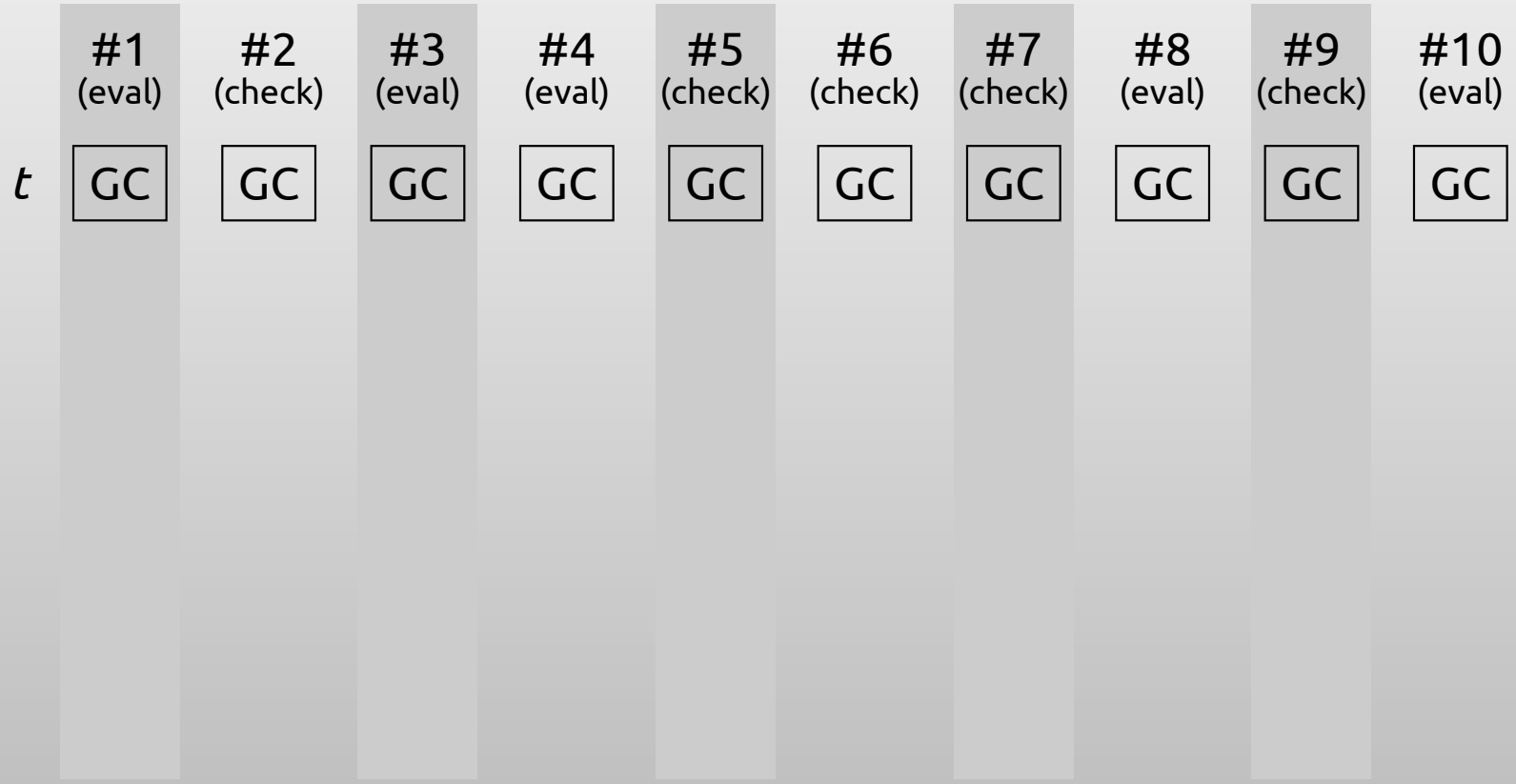
Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]



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

our approach

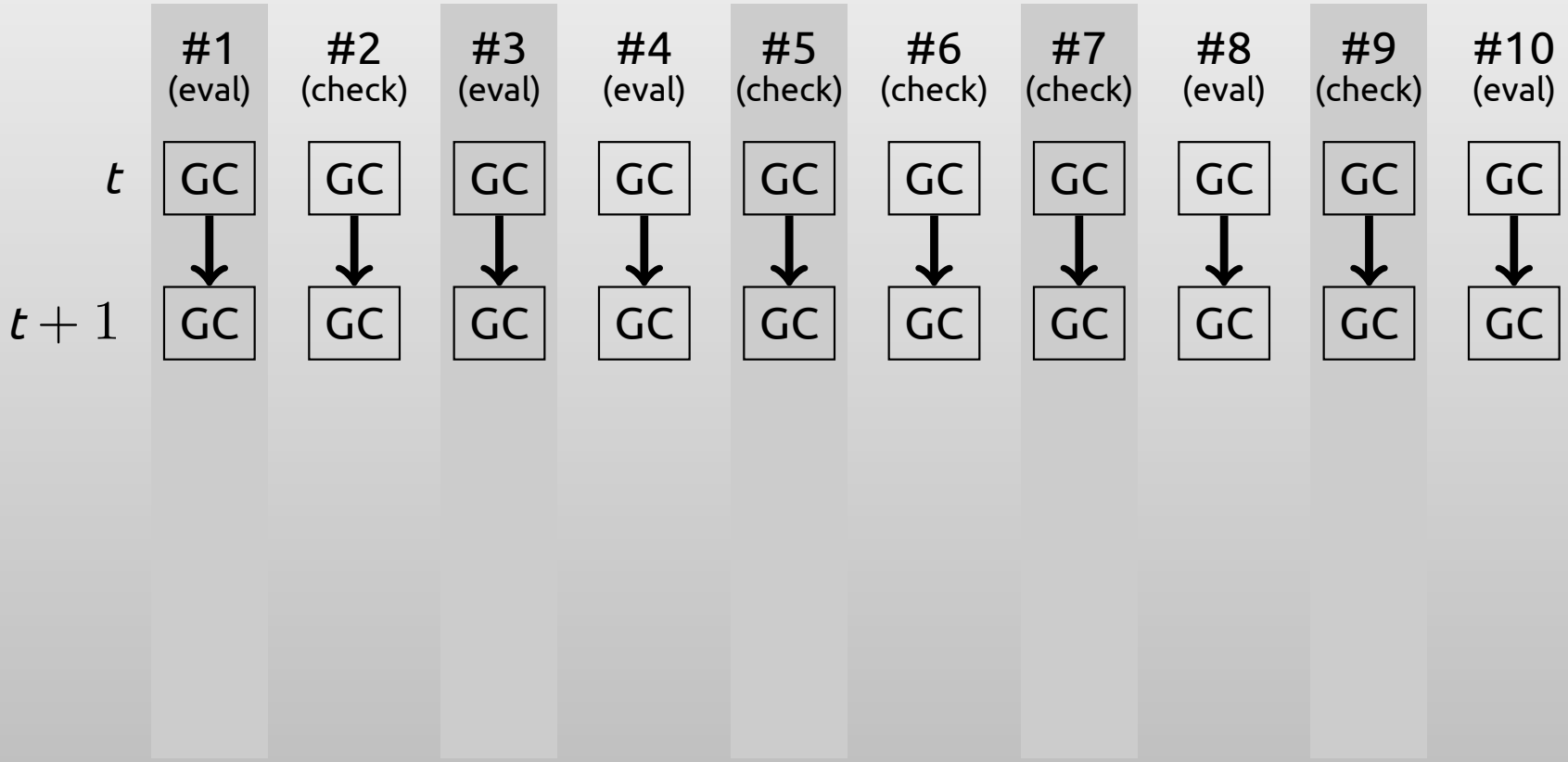
Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]



sender generates garbled circuits, reusing wire labels within each thread

our approach

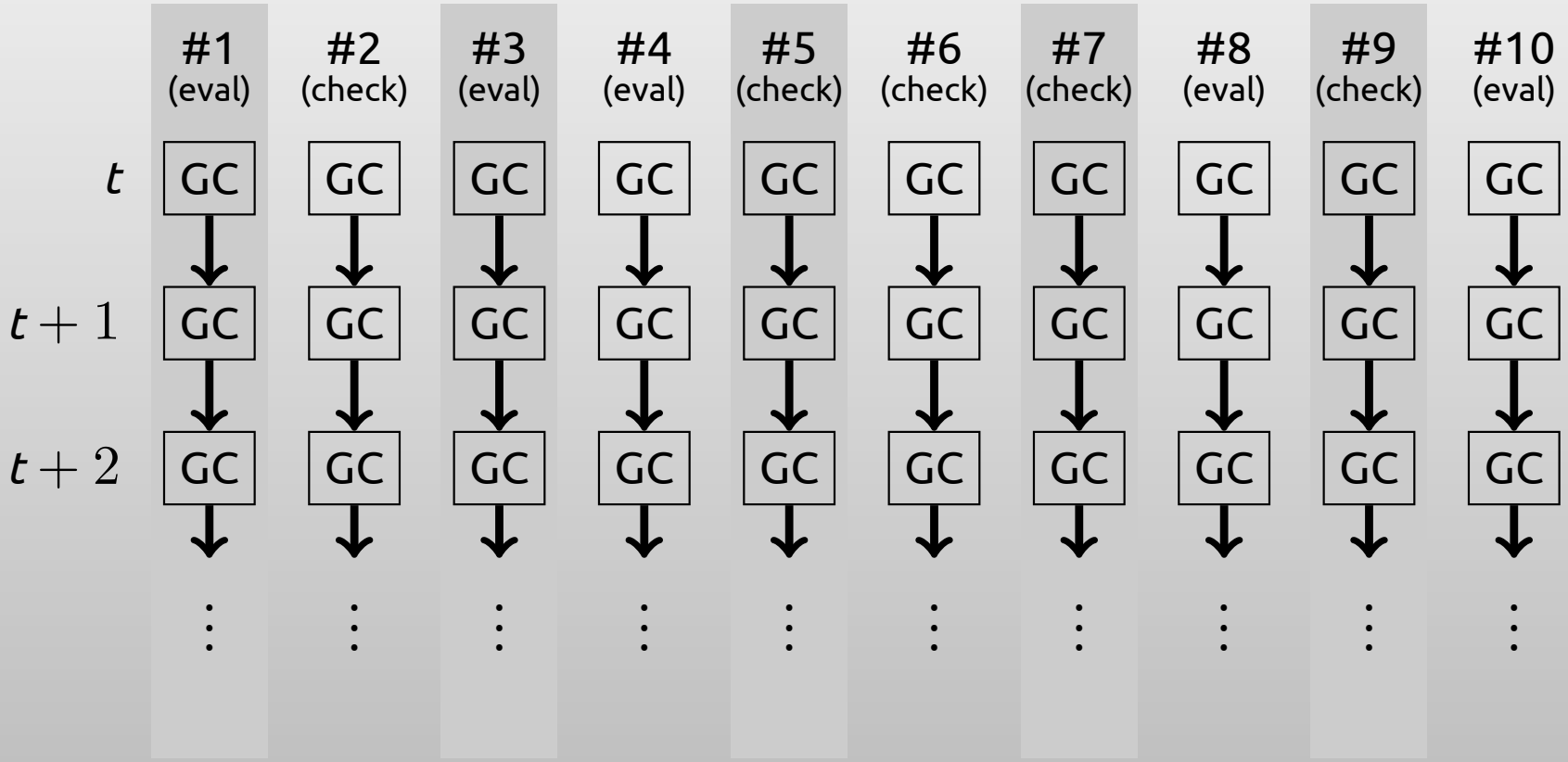
Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]



sender generates garbled circuits, reusing wire labels within each thread

our approach

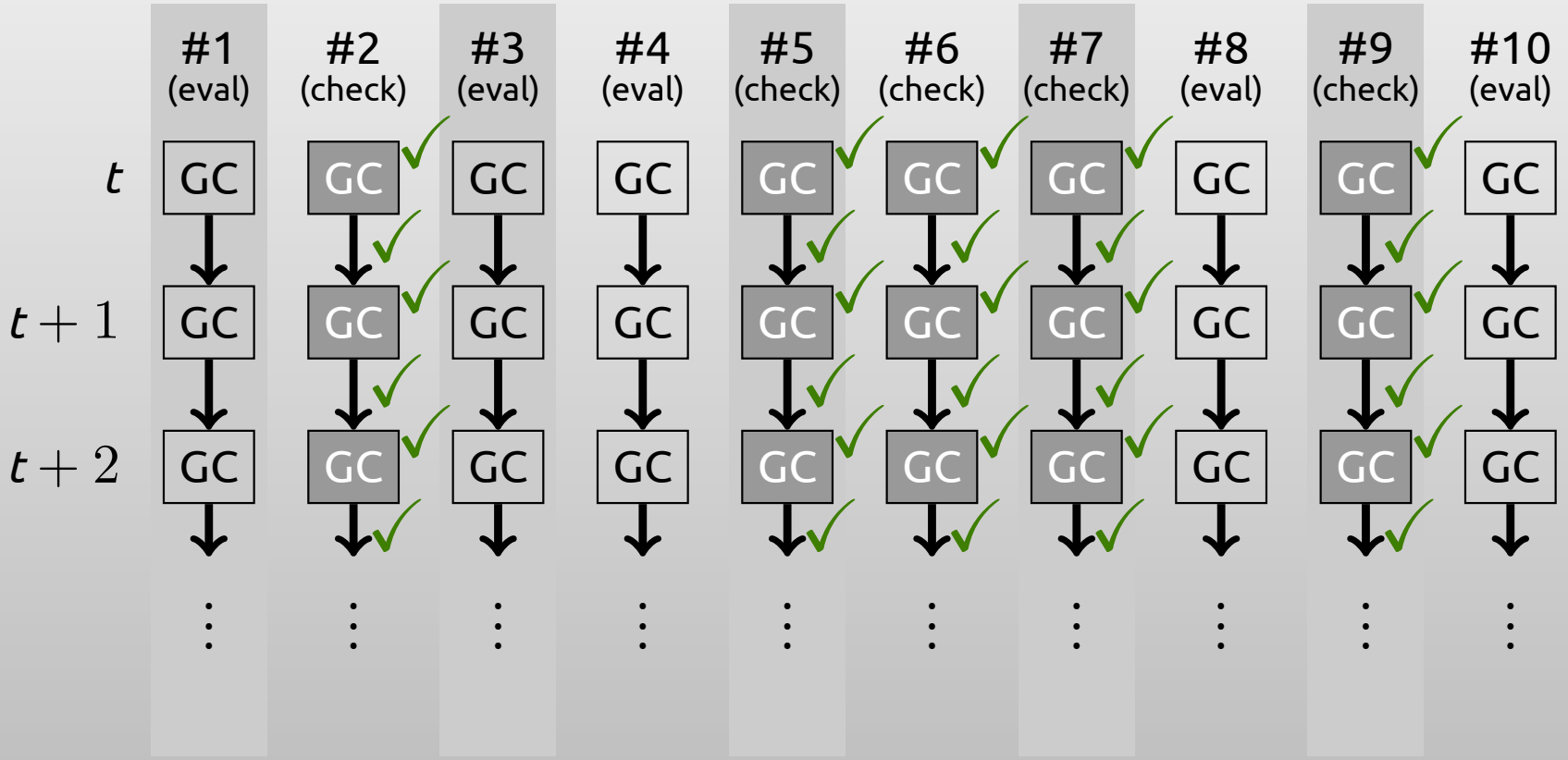
Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]



sender generates garbled circuits, reusing wire labels within each thread

our approach

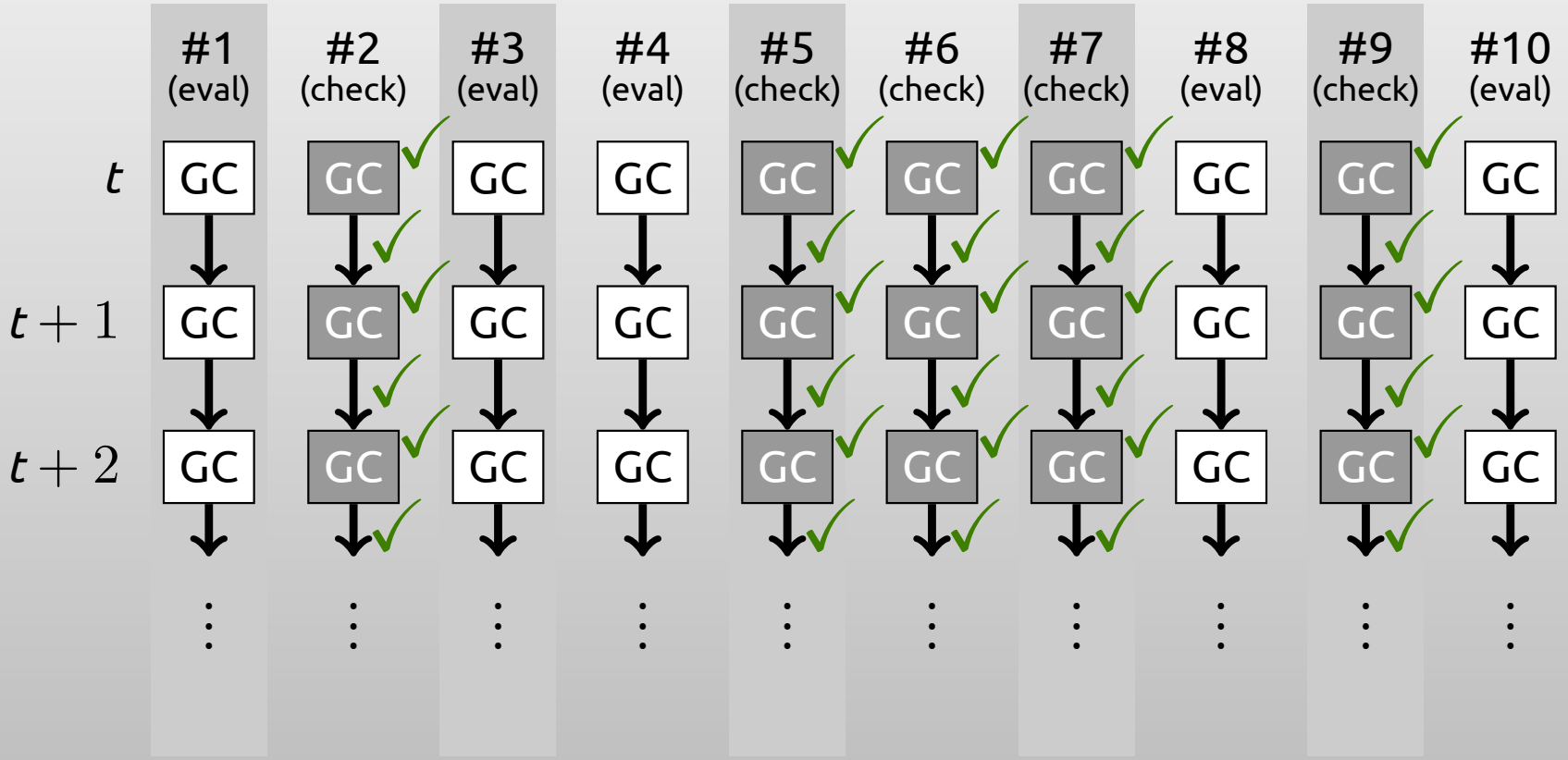
Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]



check-threads: receiver gets only enough to check

our approach

Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]



eval-threads: receiver gets only enough to eval on sender's input

overview of protocol #1

Cost of protocol = (# of threads) $\times$ (cost of semi-honest)

- with traditional cut-and-choose: $\sim 3s$ threads for security 2^{-s}
- with [Lindell13] cheating-recovery trick: only s threads
- we show how to perform [Lindell13] trick only once at the end; communication independent of RAM running time!

overview of protocol #1

Cost of protocol = (# of threads) $\times$ (cost of semi-honest)

- with traditional cut-and-choose: $\sim 3s$ threads for security 2^{-s}
- with [Lindell13] cheating-recovery trick: only s threads
- we show how to perform [Lindell13] trick only once at the end; communication independent of RAM running time!

Preprocessing, streaming?

- need to remember wire labels of previous circuits!
- can't pre-process garbled circuits (wire labels have runtime dependence)

preprocessing: batched cut-and-choose

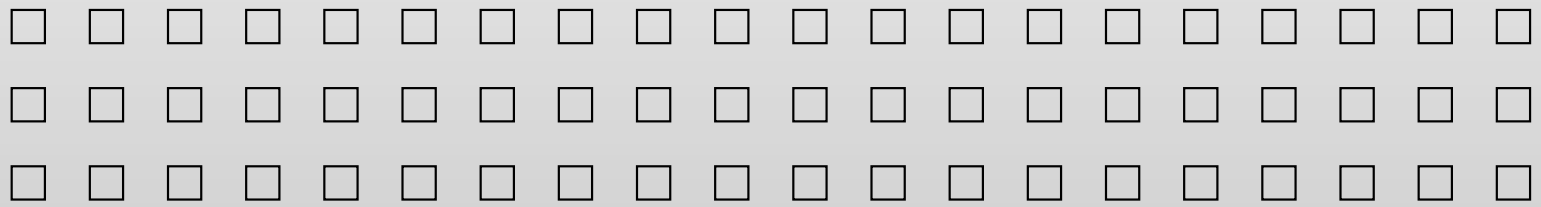
Want to do 2PC of same circuit *N* times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

preprocessing: batched cut-and-choose

Want to do 2PC of same circuit N times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]



generate a lot of garbled circuits

preprocessing: batched cut-and-choose

Want to do 2PC of same circuit *N* times?

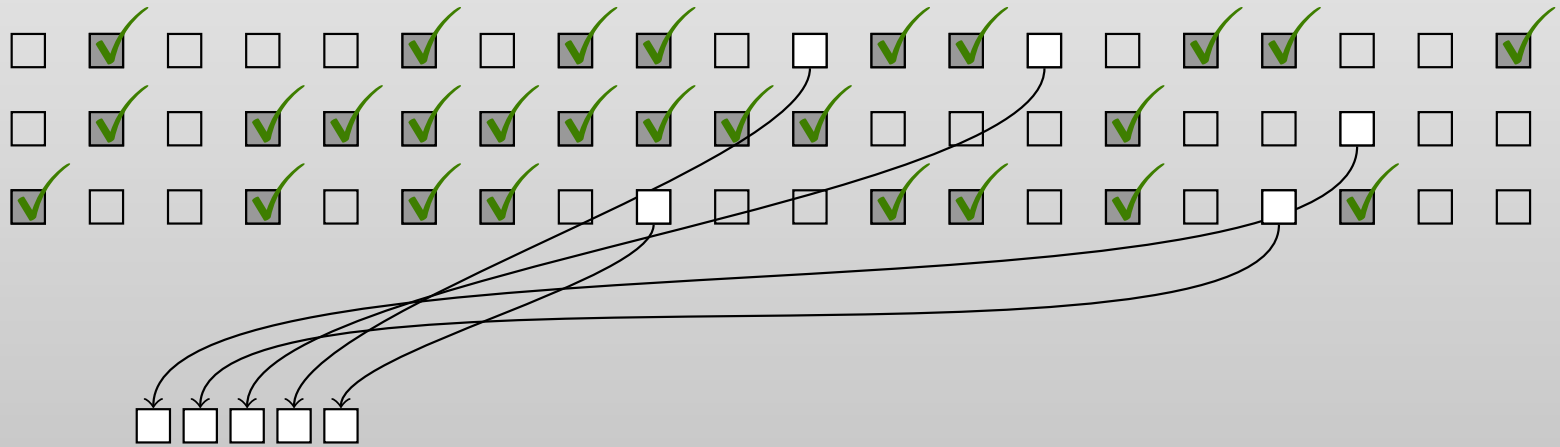
[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

open and check some fraction of them

preprocessing: batched cut-and-choose

Want to do 2PC of same circuit *N* times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

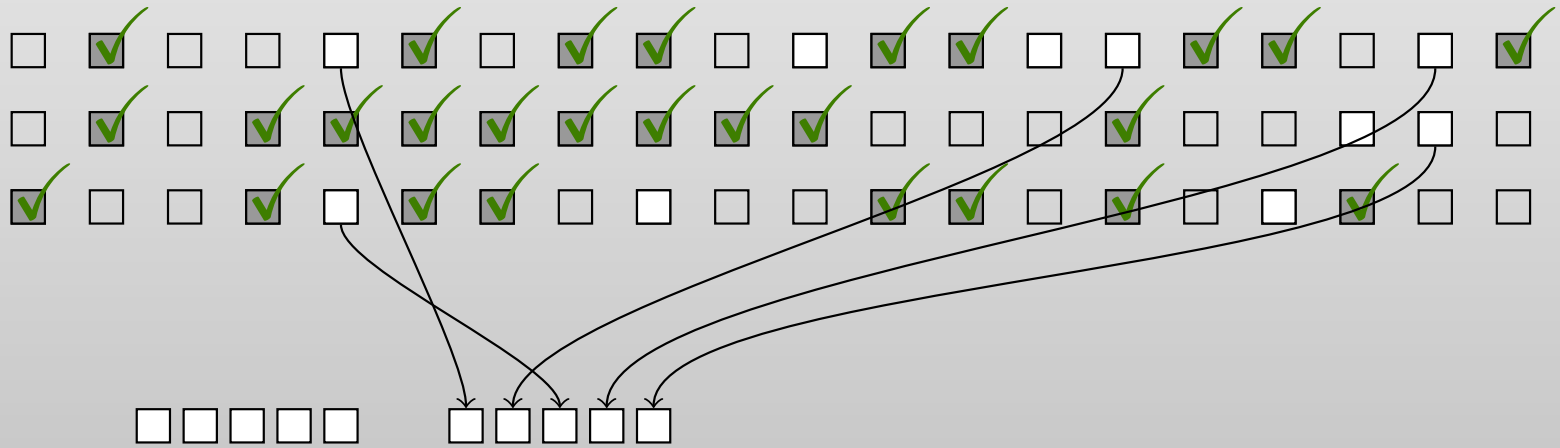


pick a random “bucket” of available circuits and evaluate them

preprocessing: batched cut-and-choose

Want to do 2PC of same circuit *N* times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

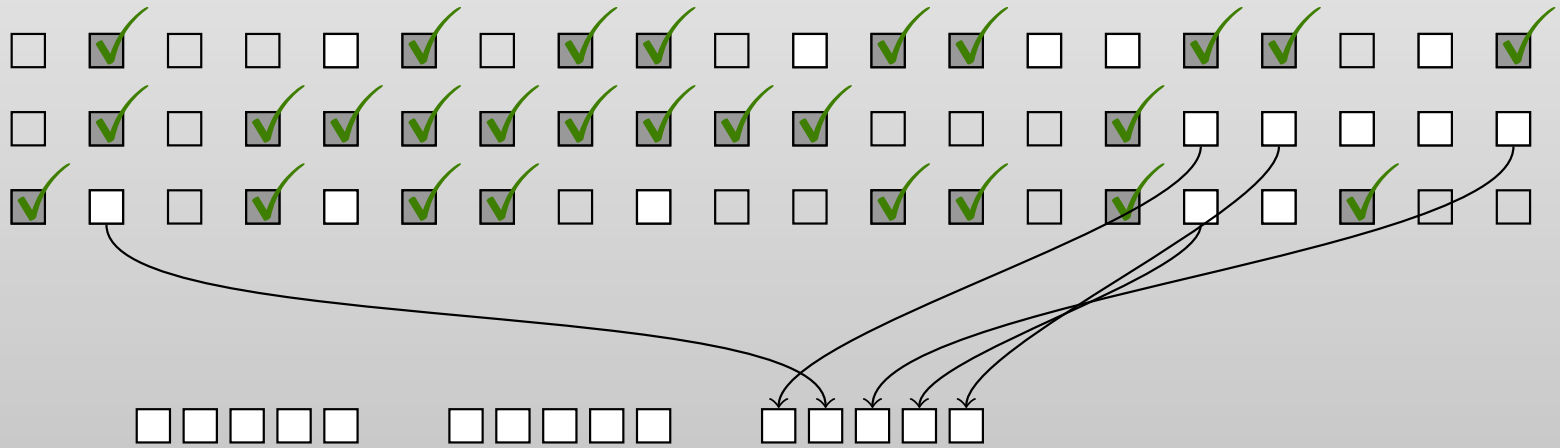


pick a random “bucket” of available circuits and evaluate them

preprocessing: batched cut-and-choose

Want to do 2PC of same circuit *N* times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

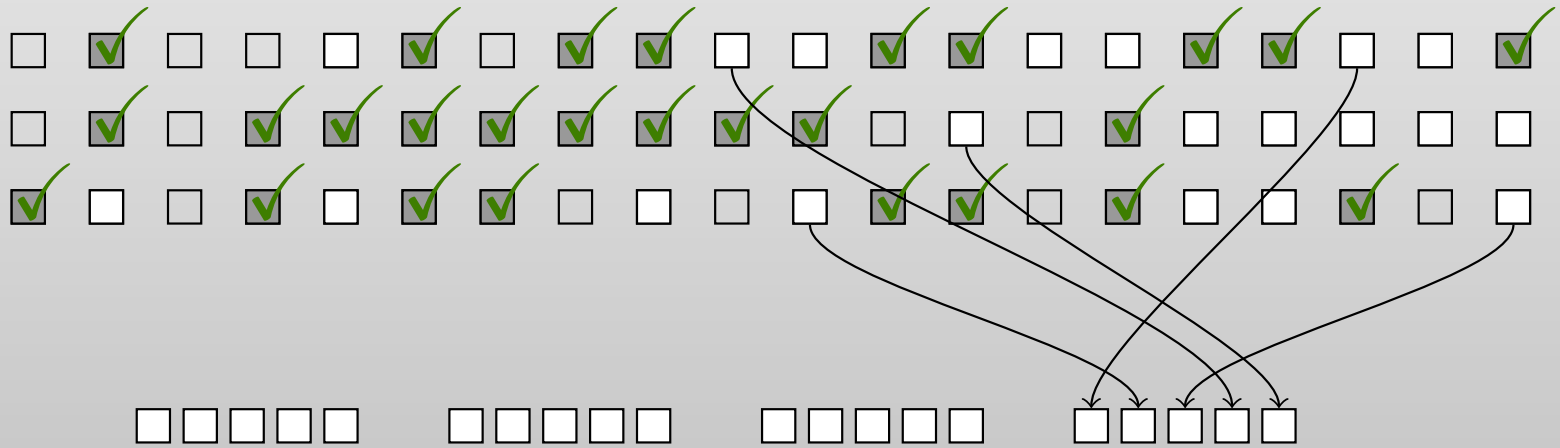


pick a random “bucket” of available circuits and evaluate them

preprocessing: batched cut-and-choose

Want to do 2PC of same circuit *N* times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

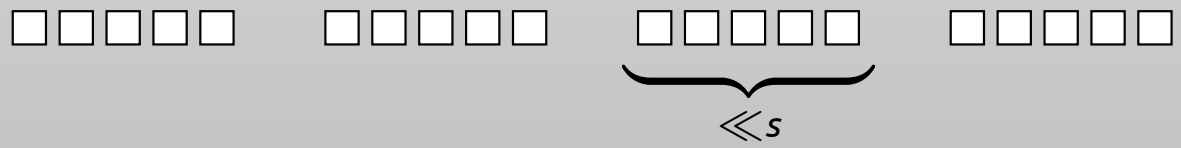


pick a random “bucket” of available circuits and evaluate them

preprocessing: batched cut-and-choose

Want to do 2PC of same circuit N times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]



buckets of size $O(s/\log N)$ give security 2^{-s}

preprocessing for RAM-2PC?

Pros:

- RAM CPU circuit evaluated over and over!
- Batched cut-and-choose would reduce number of garbled circuits needed (in online phase)
- Pre-processing already inherent for ORAM

preprocessing for RAM-2PC?

Pros:

- RAM CPU circuit evaluated over and over!
- Batched cut-and-choose would reduce number of garbled circuits needed (in online phase)
- Pre-processing already inherent for ORAM

Cons:

- Wire-label dependence determined at runtime!
- Cannot re-use wire labels if all circuits garbled beforehand

preprocessing for RAM-2PC?

Pros:

- RAM CPU circuit evaluated over and over!
- Batched cut-and-choose would reduce number of garbled circuits needed (in online phase)
- Pre-processing already inherent for ORAM

Cons:

- Wire-label dependence determined at runtime!
- Cannot re-use wire labels if all circuits garbled beforehand

If only we had a way to "connect wires on the fly" in existing garbled circuits!

the LEGO approach!

Garble individual gates and connect them later

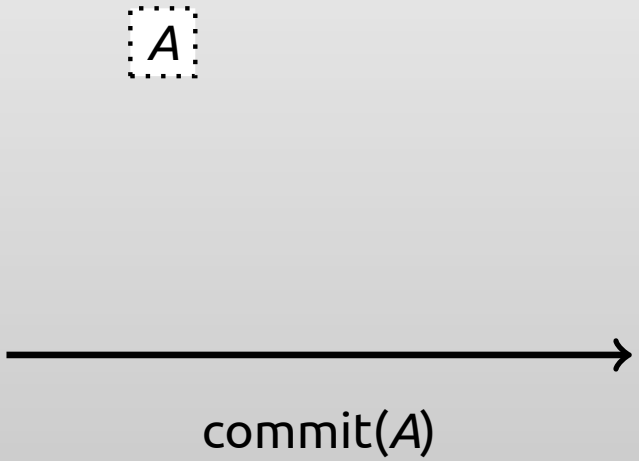
[NielsenOrlandi09,FrederiksenJakobsenNielsenNordholdOrlandi13]

- We extend the technique to circuits
- Some careful modifications are necessary

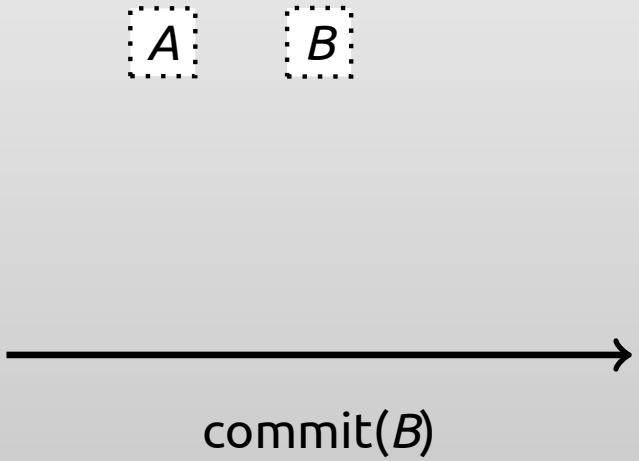
our “LEGO RAM” approach

LEGO set 6062 “Battering Ram”

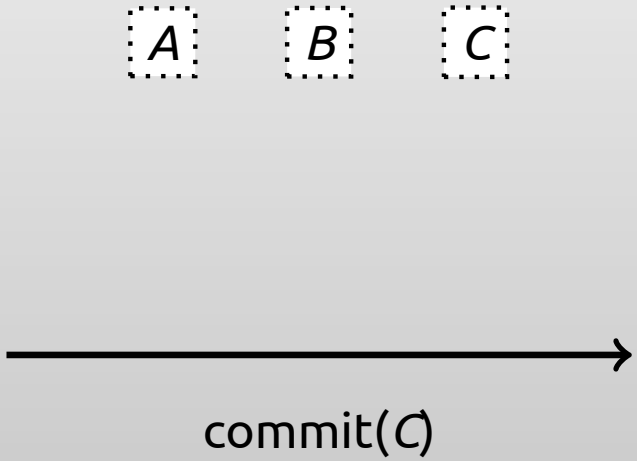
xor-homomorphic commitment



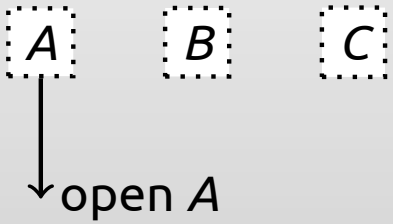
xor-homomorphic commitment



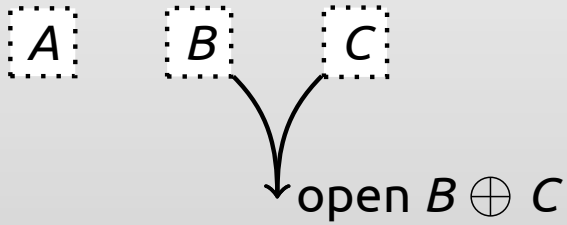
xor-homomorphic commitment



xor-homomorphic commitment

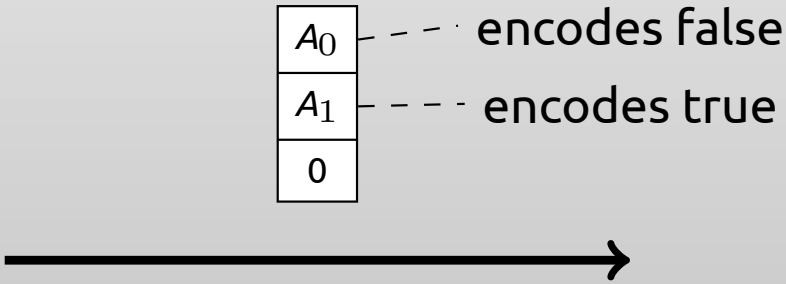


xor-homomorphic commitment



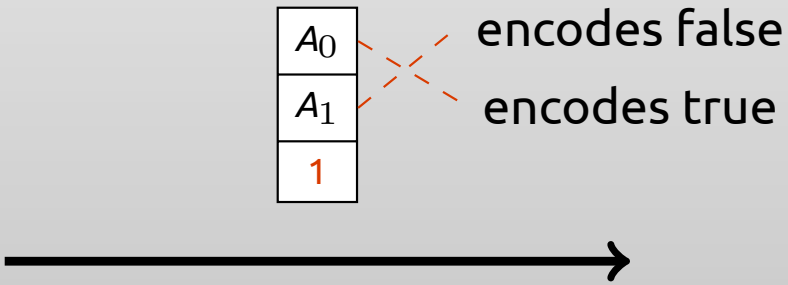
garbled circuit, wire labels

Each wire has a secret “parity bit”



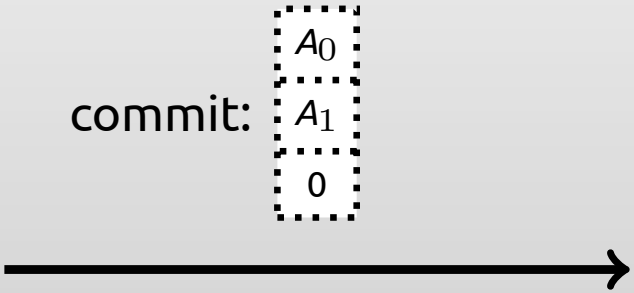
garbled circuit, wire labels

Each wire has a secret “parity bit”



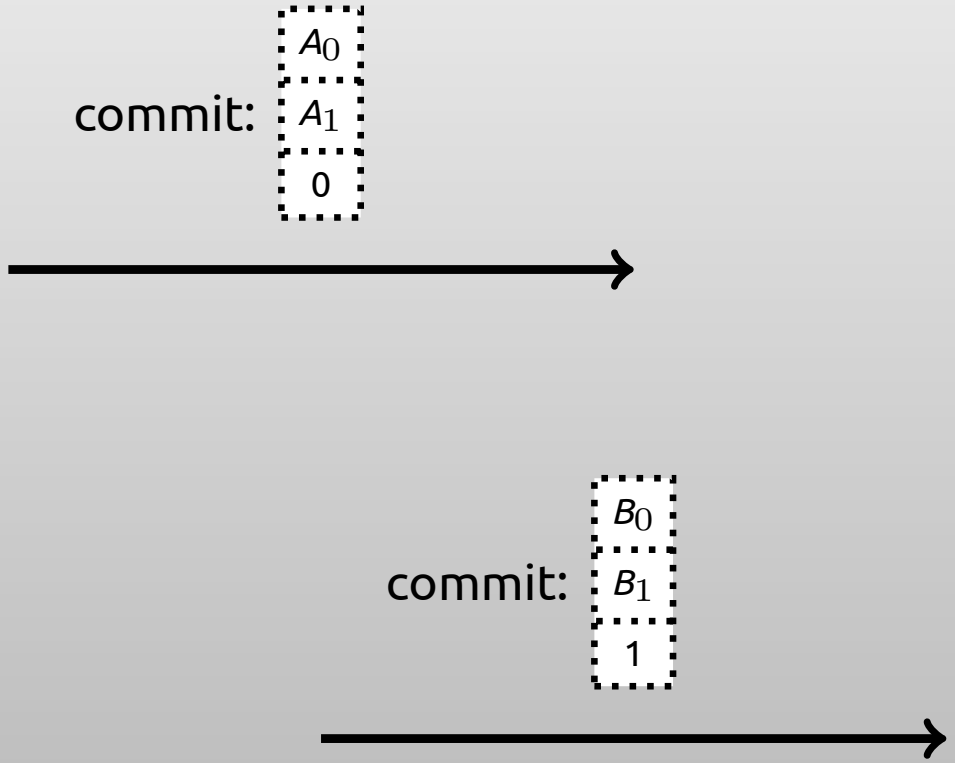
soldering

connect wires of garbled circuits “on the fly”



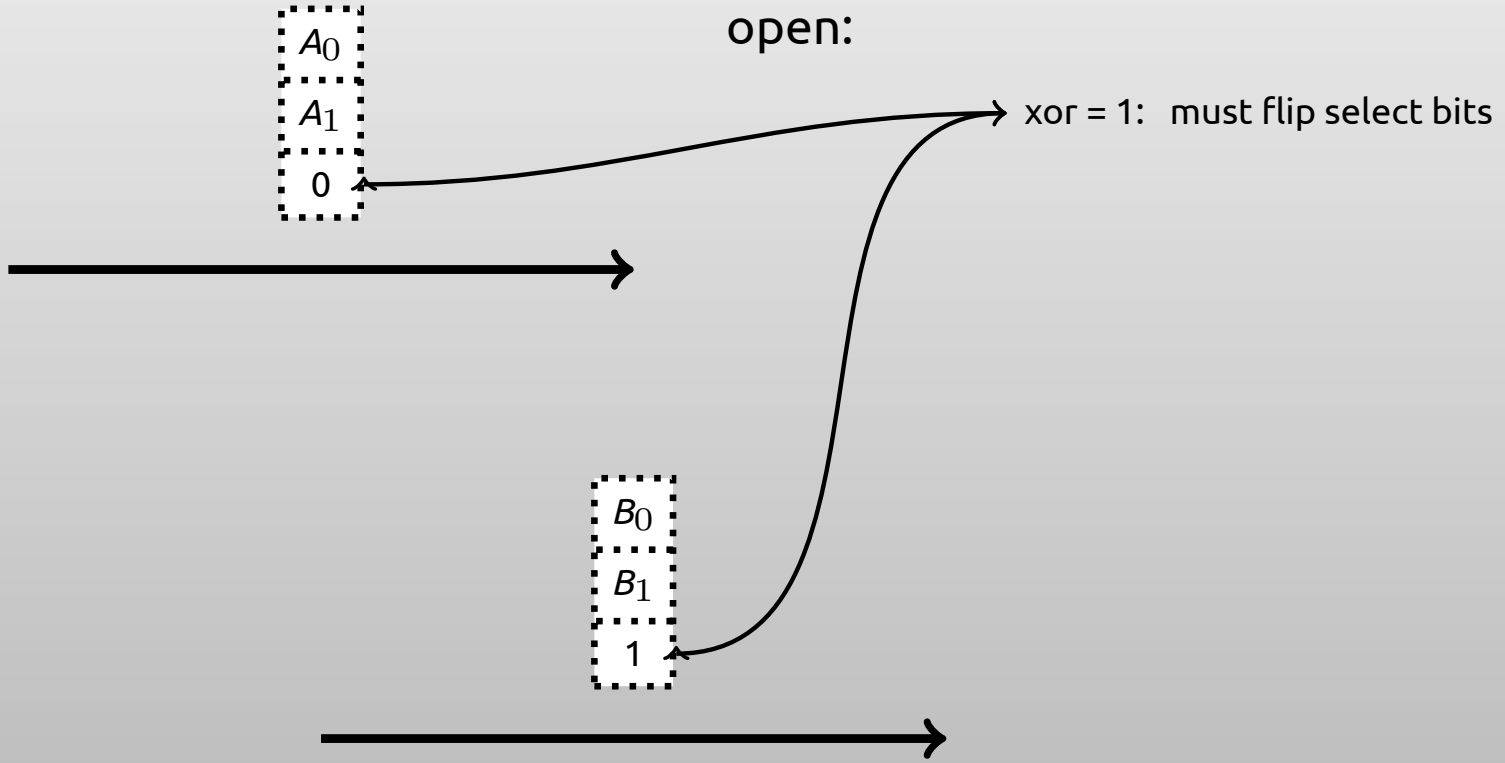
soldering

connect wires of garbled circuits “on the fly”



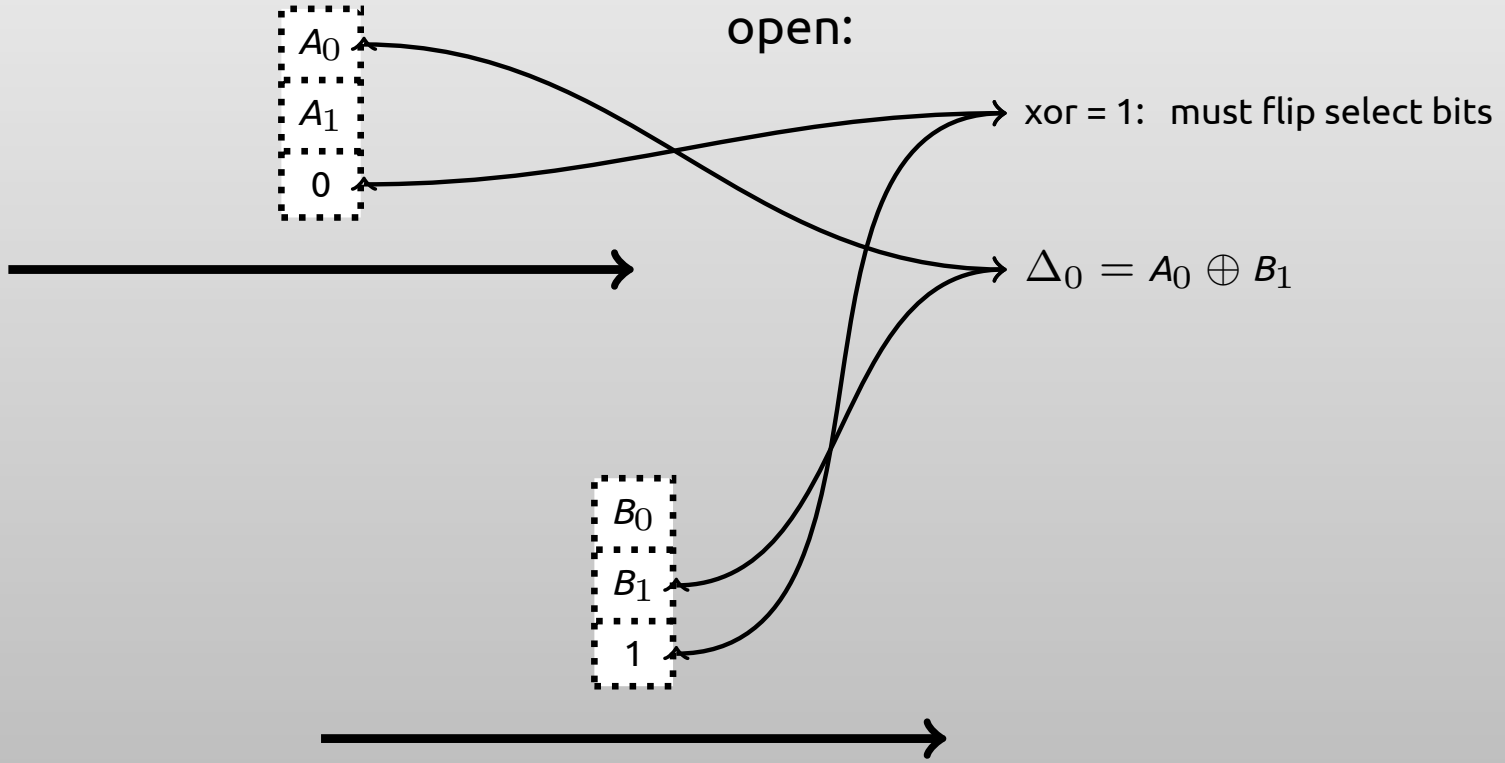
soldering

connect wires of garbled circuits “on the fly”



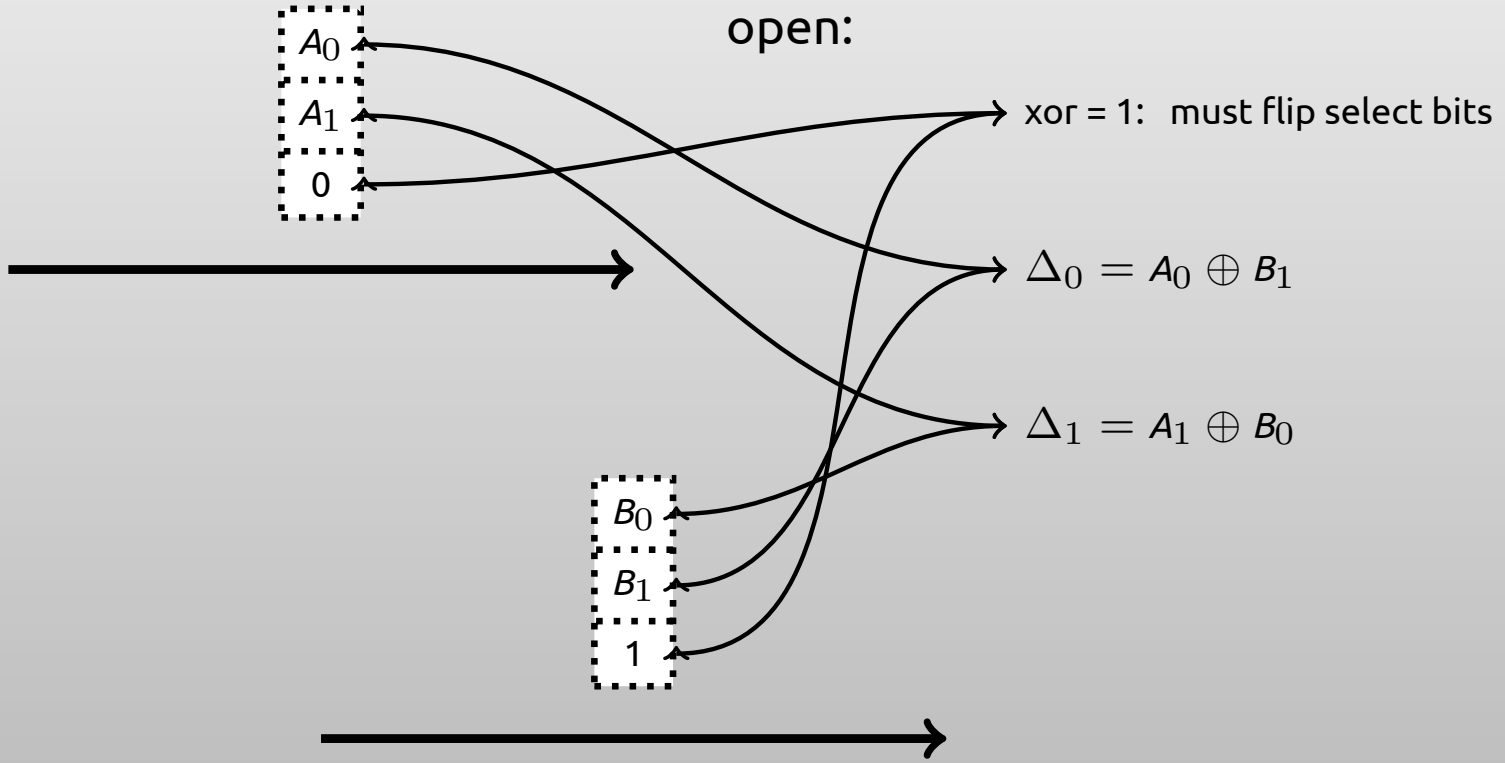
soldering

connect wires of garbled circuits “on the fly”



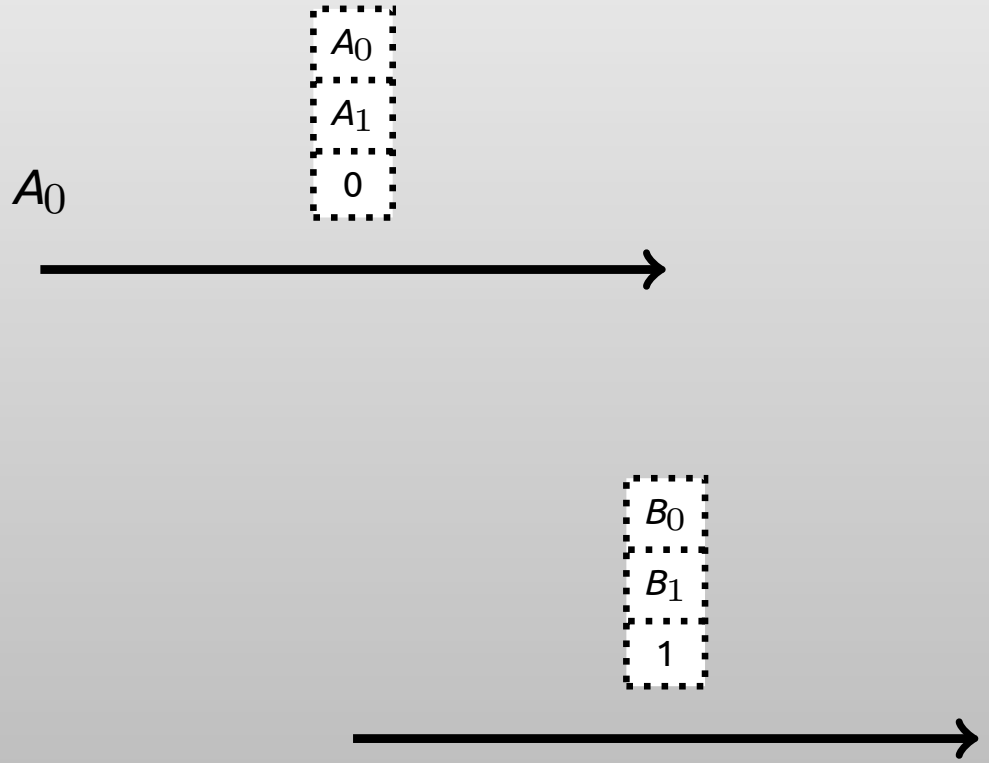
soldering

connect wires of garbled circuits “on the fly”



soldering

connect wires of garbled circuits “on the fly”



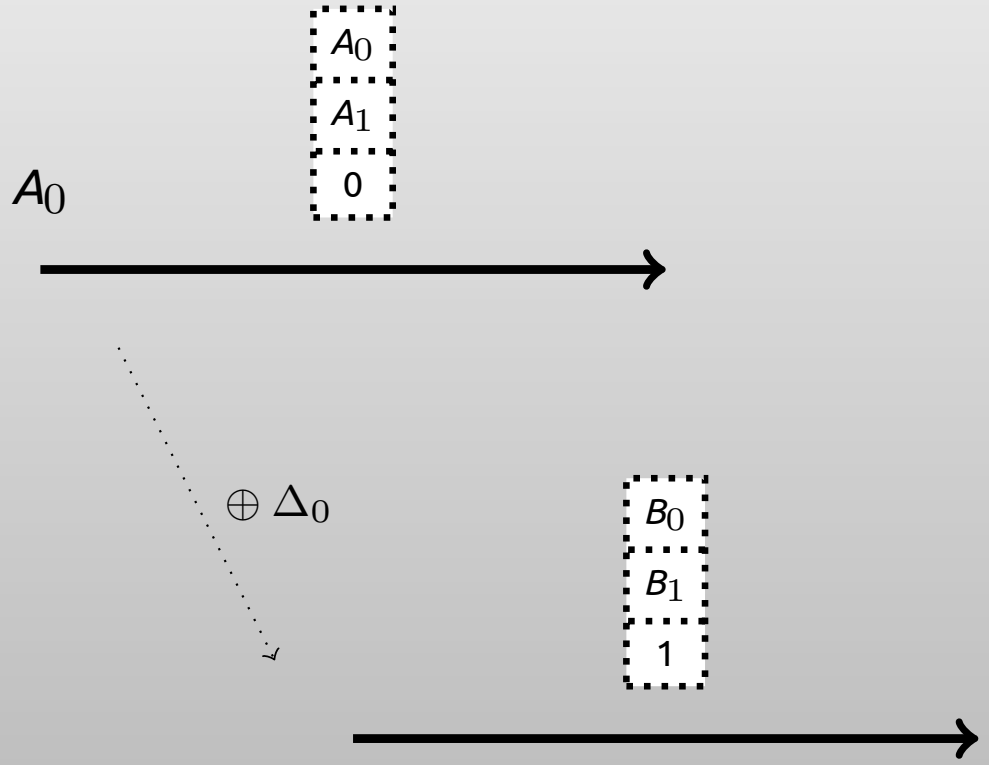
xor = 1: must flip select bits

$$\Delta_0 = A_0 \oplus B_1$$

$$\Delta_1 = A_1 \oplus B_0$$

soldering

connect wires of garbled circuits “on the fly”



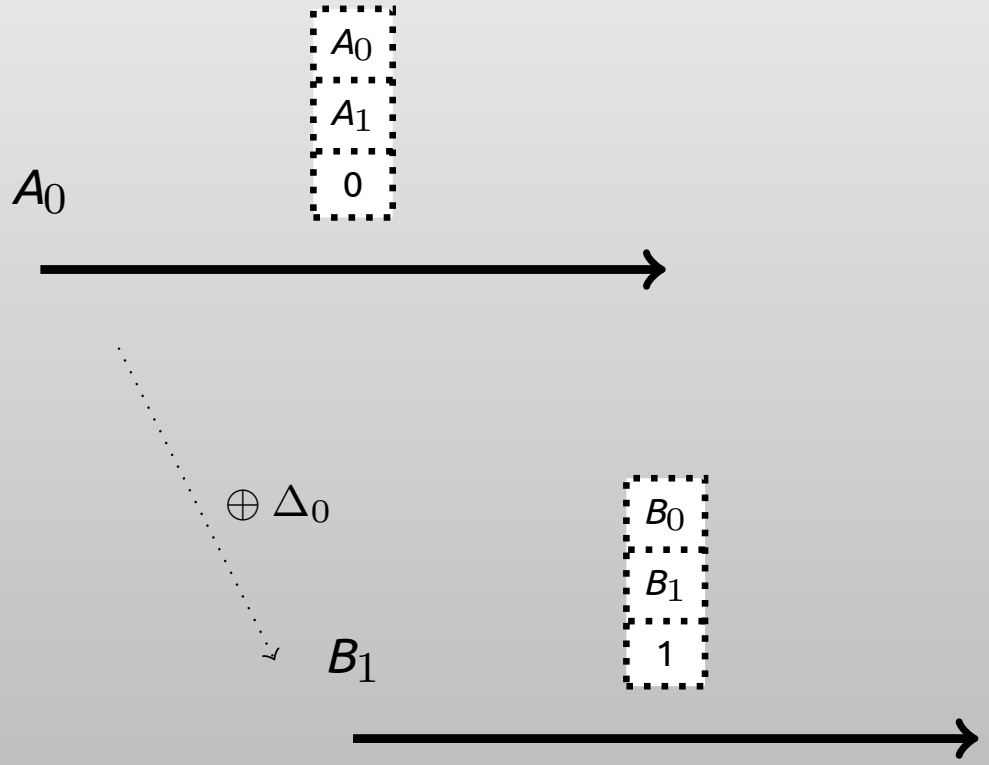
xor = 1: must flip select bits

$\Delta_0 = A_0 \oplus B_1$

$\Delta_1 = A_1 \oplus B_0$

soldering

connect wires of garbled circuits “on the fly”



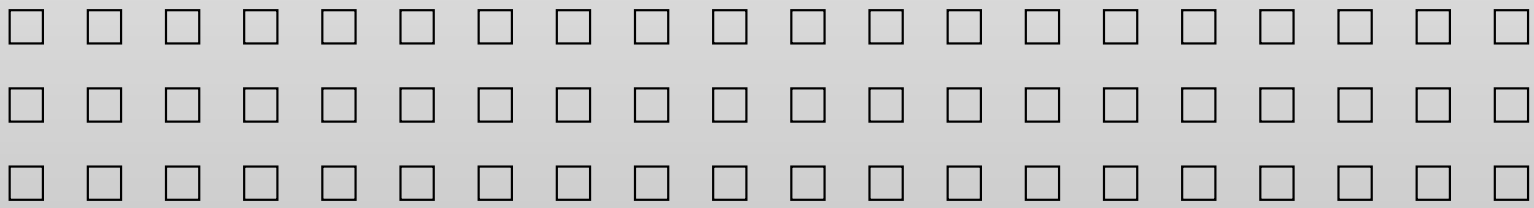
xor = 1: must flip select bits

$\Delta_0 = A_0 \oplus B_1$

$\Delta_1 = A_1 \oplus B_0$

putting it all together

putting it all together

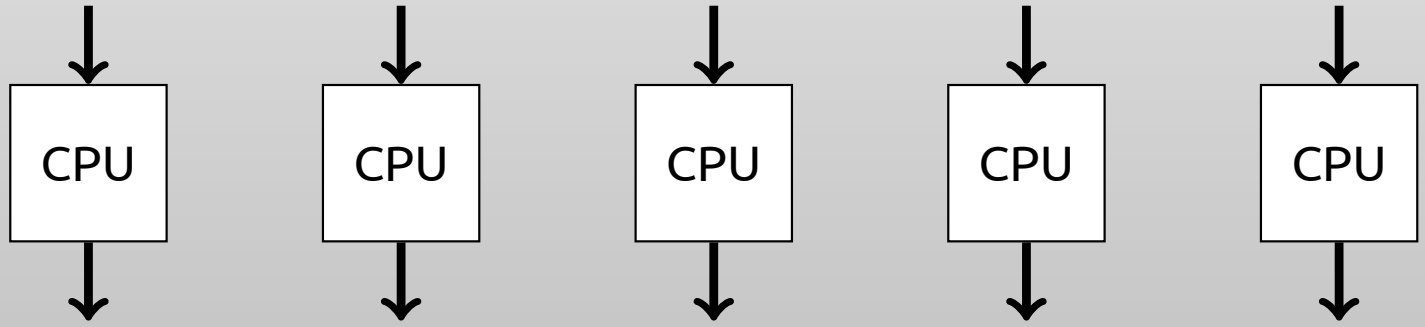


generate lots of garbled CPU circuits + homomorphic commitments

putting it all together

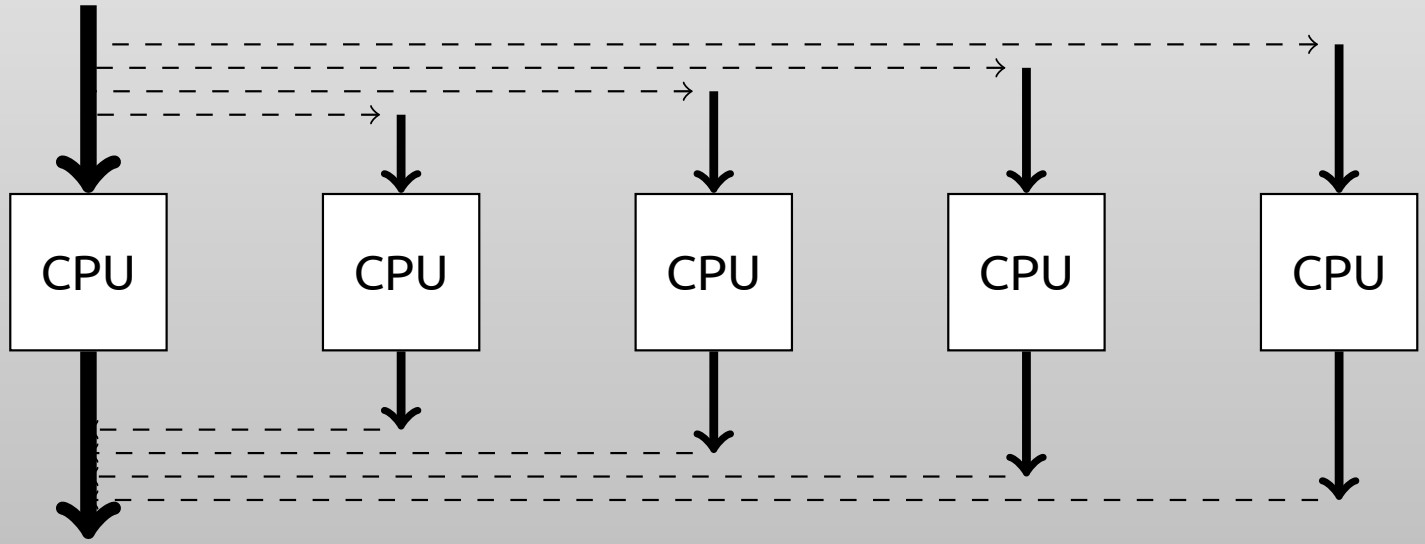
open and check some fraction of them

putting it all together



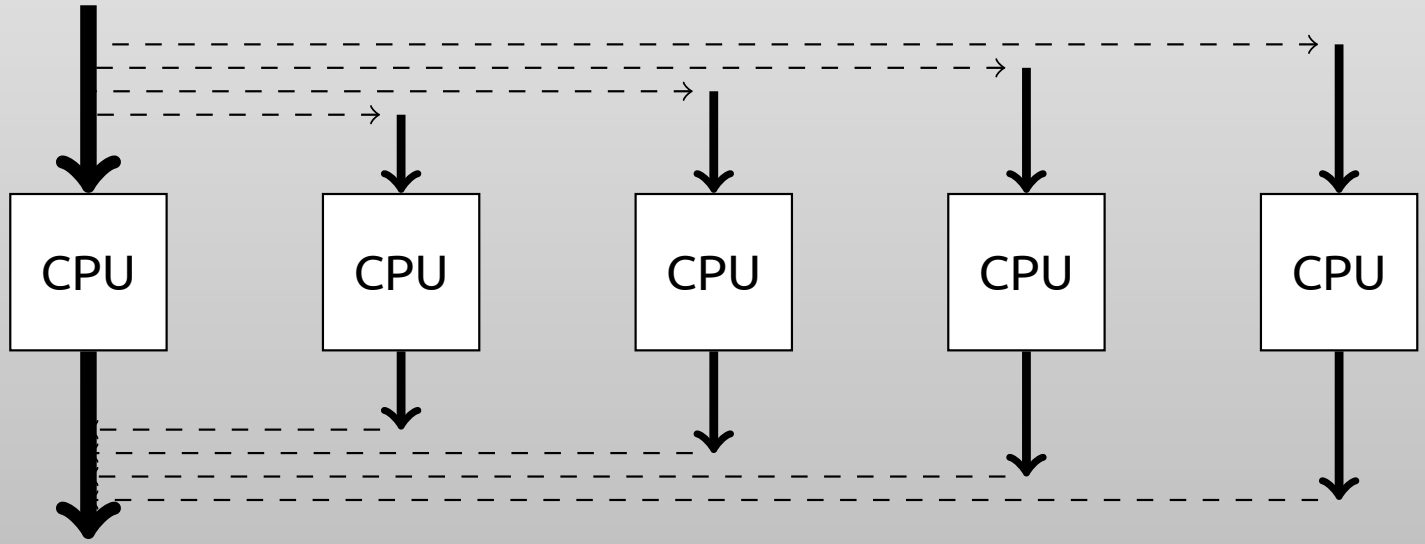
each timestep, select random “bucket” of circuits to evaluate

putting it all together



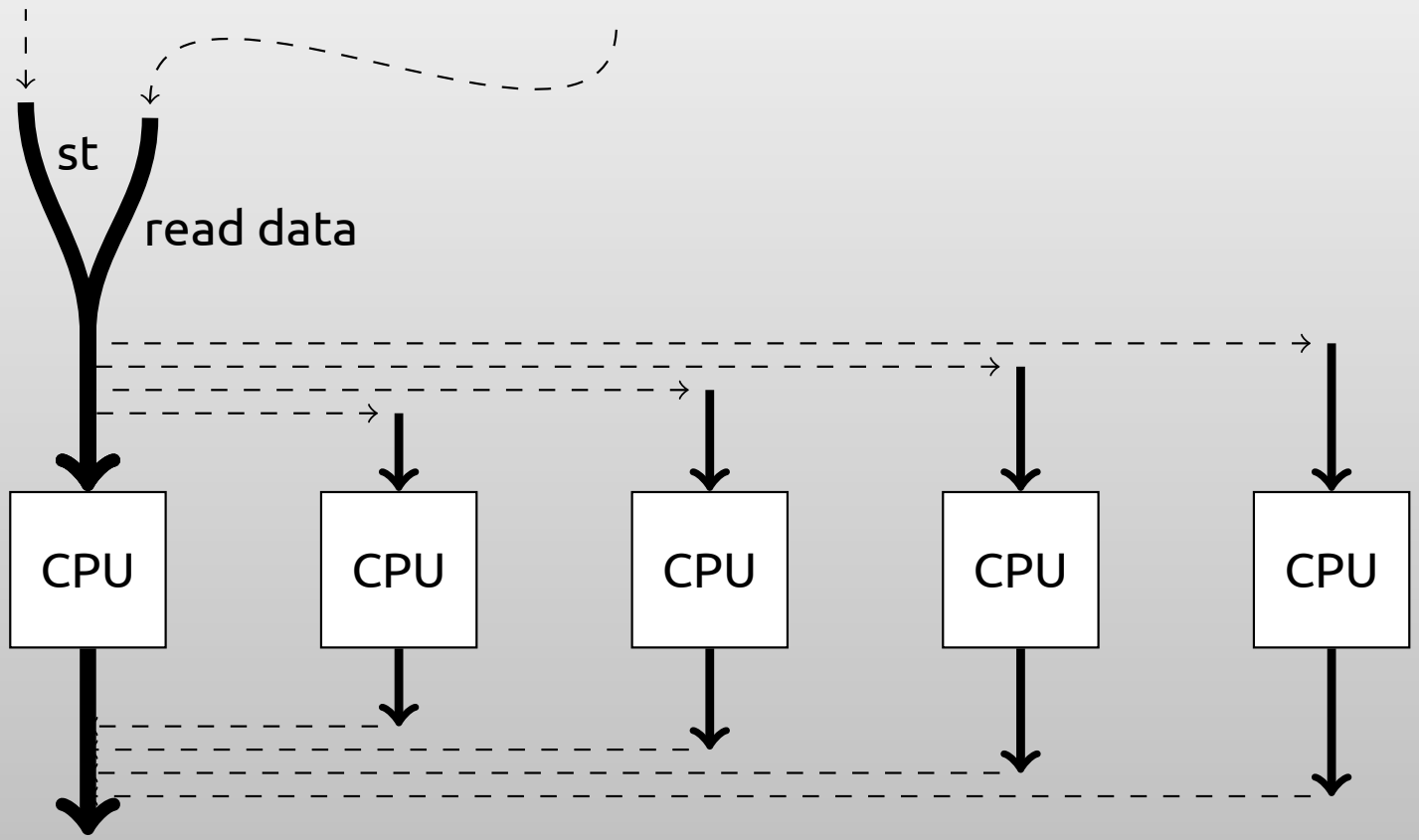
solder input/output wires together by opening commitments

putting it all together



(evaluate by taking majority wire labels of each computation path)

putting it all together



solder input wires from previous garbled circuits

overview of protocol #2

Two phase protocol:

- offline phase = circuit generation & batch cut-and-choose
- online phase = soldering & evaluation

overview of protocol #2

Two phase protocol:

- offline phase = circuit generation & batch cut-and-choose
- online phase = soldering & evaluation

Online cost of protocol = (bucket size) $\times$ (cost of semi-honest)

- bucket size = $O(s/\log T)$ where $T =$ RAM running time
- concretely, for $s = 40$:
  - $T = 5{,}000 \Rightarrow$ bucket size = 7
  - $T = 500{,}000 \Rightarrow$ bucket size = 5

overview of protocol #2

Two phase protocol:

- offline phase = circuit generation & batch cut-and-choose
- online phase = soldering & evaluation

Online cost of protocol = (bucket size) $\times$ (cost of semi-honest)

- bucket size = $O(s/\log T)$ where $T =$ RAM running time
- concretely, for $s = 40$:
  - $T = 5{,}000 \Rightarrow$ bucket size = 7
  - $T = 500{,}000 \Rightarrow$ bucket size = 5

Other cool features:

- oblivious RAM = (original RAM) + (ORAM construction steps)
- pre-process different circuits separately: smaller bucket size for (common) ORAM steps

lies and omissions

Lots of other standard tools from circuit-2PC:

- Input consistency checks
- Output authenticity checks
- Preventing selective aborts
- Cheating recovery techniques

RAM stuff I didn't mention:

- Elephant in the room: ORAM initialization!
- Getting inputs into the RAM
- ORAM needs randomness
- Safe to run many RAM invocations with same memory

summary

Goals:

- malicious security
- no extra overhead inside garbled circuits
- efficiency matching state-of-the-art for circuit-based 2PC

Results:

style	cost	technique
streaming	$s \times$ semi-honest	blind cut-and-choose, forge-and-lose
online/offline	$\sim 2s/\log T \times$ semi-honest	batched cut-and-choose, LEGO

for security 2^{-s}

T = ORAM running time

Theme:

- Leverage existing security properties of wire labels in garbled circuits!

the end!

Mike Rosulek : `rosulekm@eecs.oregonstate.edu`

From Circuits to RAM Programs in Malicious-2PC

Arash Afshar, Zhangxiang Hu, Payman Mohassel, Mike Rosulek

appearing on eprint soon