How to Efficiently Evaluate RAM Programs with Malicious Security

Arash Afshar, Zhangxiang Hu, Payman Mohassel, and Mike Rosulek

April 29, 2015

Background

- Secure Two-Party Computation (2PC)
  - Secure evaluation of "any" function
  - Preserve input privacy and correctness
  - Majority use Boolean/arithmetic circuits

Background

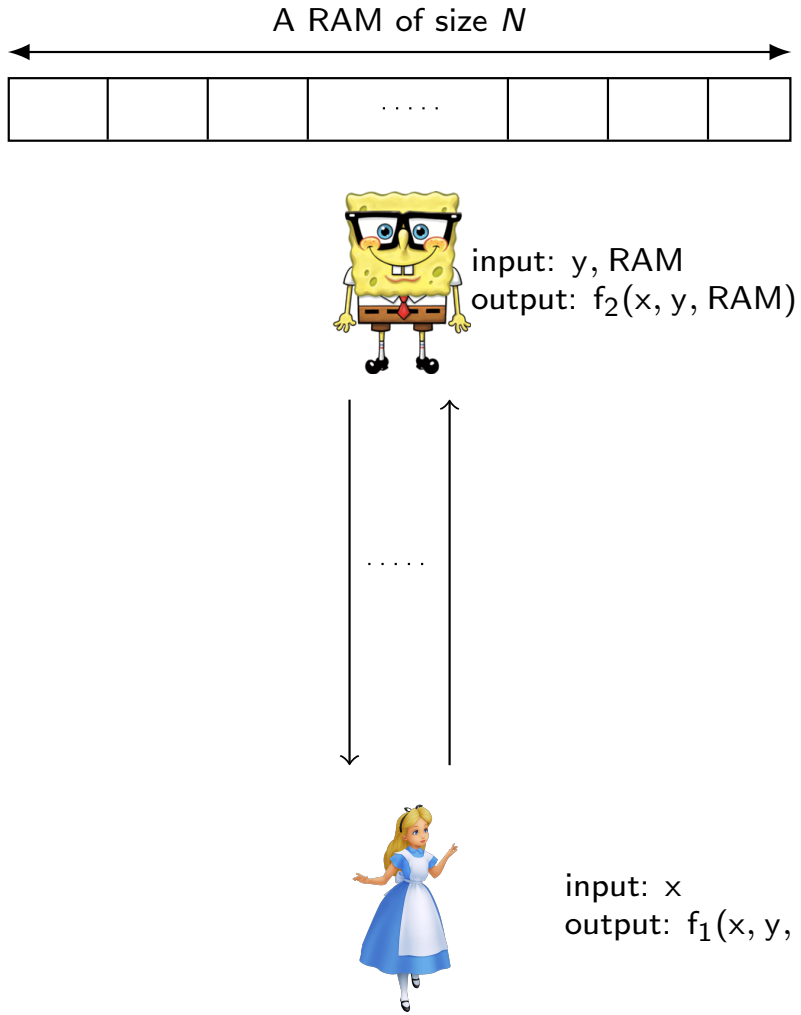
- Secure Two-Party Computation (2PC)
  - Secure evaluation of "any" function
  - Preserve input privacy and correctness
  - Majority use Boolean/arithmetic circuits
- The problem
  - Not efficient for Random-Access Memory (*RAM*) programs
  - Accessing a portion of the RAM.

Background

- Secure Two-Party Computation (2PC)
  - Secure evaluation of "any" function
  - Preserve input privacy and correctness
  - Majority use Boolean/arithmetic circuits
- The problem
  - Not efficient for Random-Access Memory (*RAM*) programs
  - Accessing a portion of the RAM.
- Solution [GKK+12]
  - Combine "Oblivious RAM" (ORAM) with 2PC

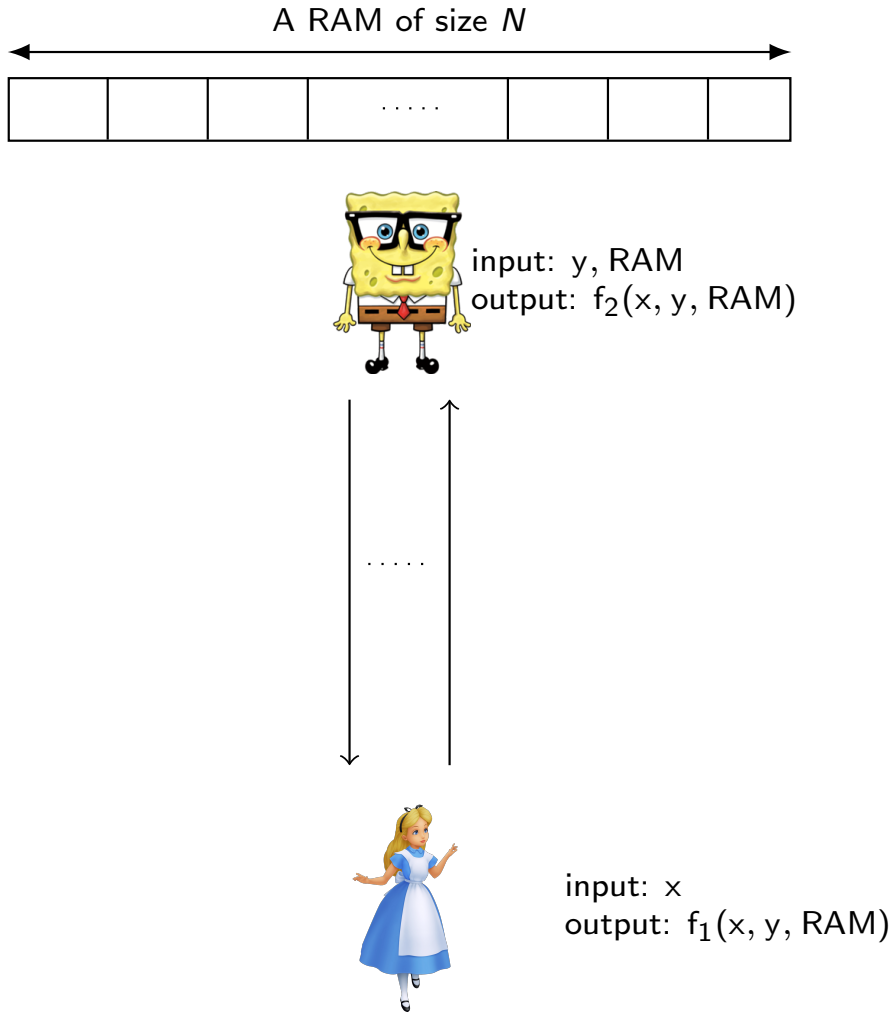
Oblivious RAM (ORAM)

- Privacy of Alice input
  - *ORAM*: $O(\log N)$
- Privacy of Bob input
  - No guarantee



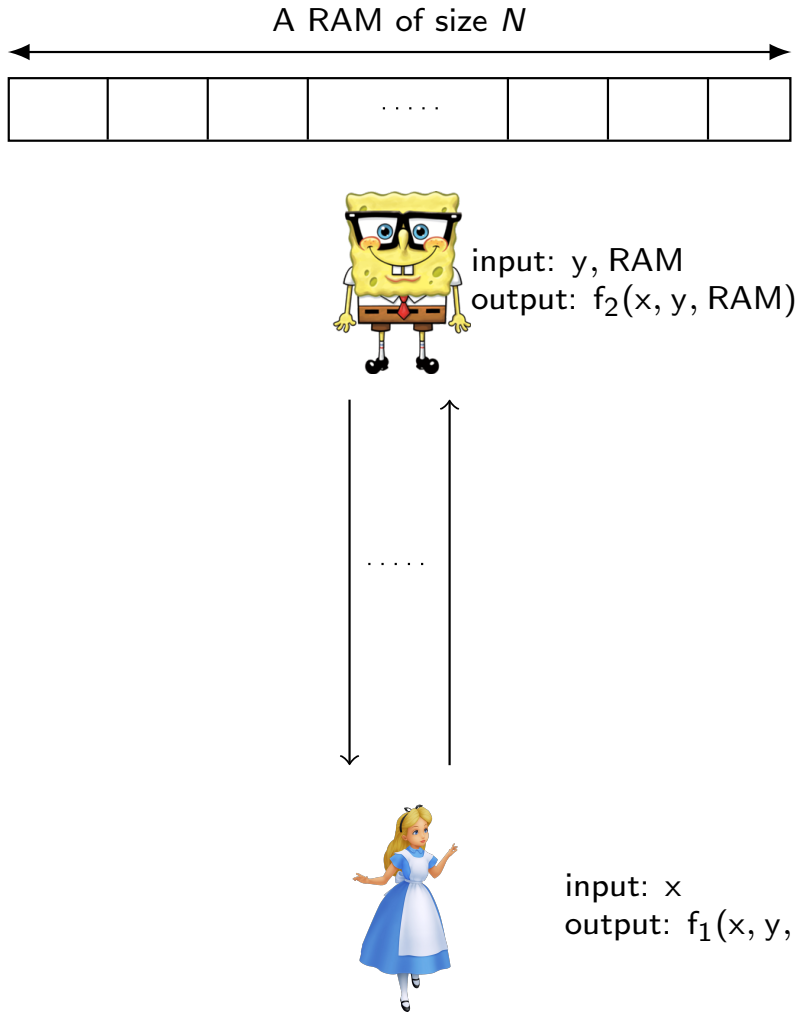
Oblivious RAM (ORAM)

- Privacy of Alice input
  - *ORAM*: $O(\log N)$
- Privacy of Bob input
  - No guarantee
- Construction
  - Imagine the original data in your head



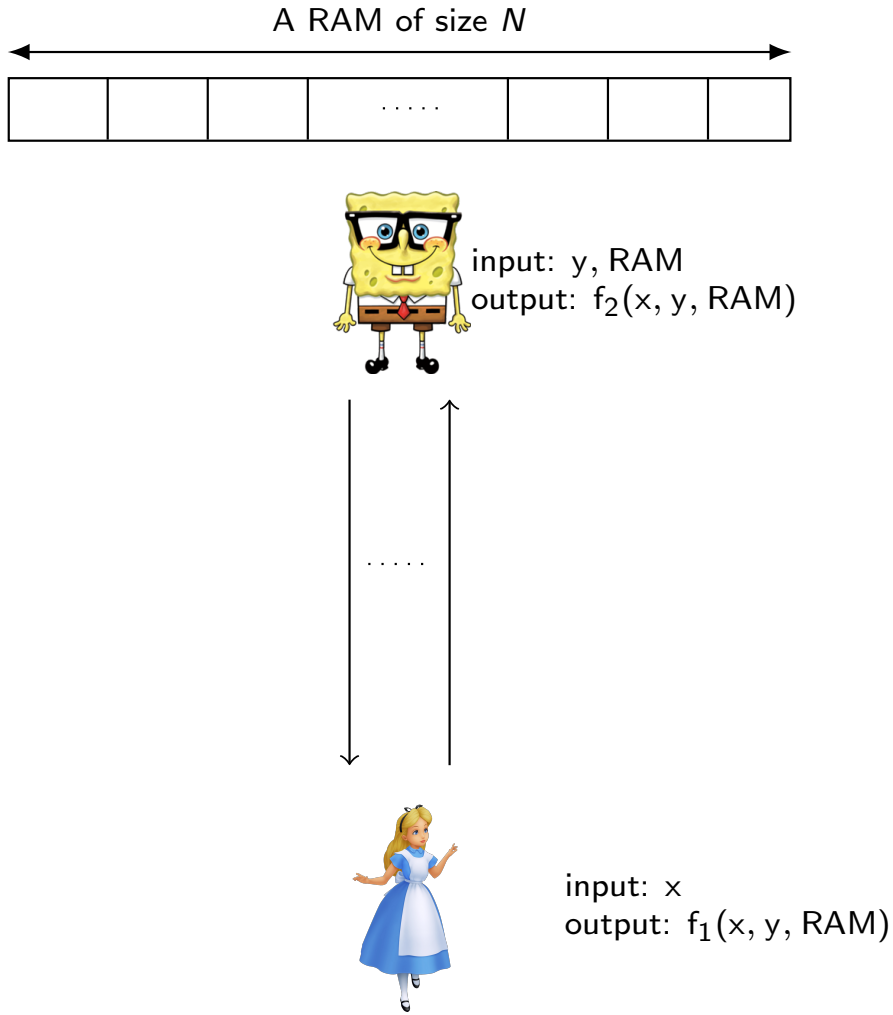
Oblivious RAM (ORAM)

- Privacy of Alice input
  - *ORAM*: $O(\log N)$
- Privacy of Bob input
  - No guarantee
- Construction
  - Imagine the original data in your head
    - Store it in a data structure in RAM



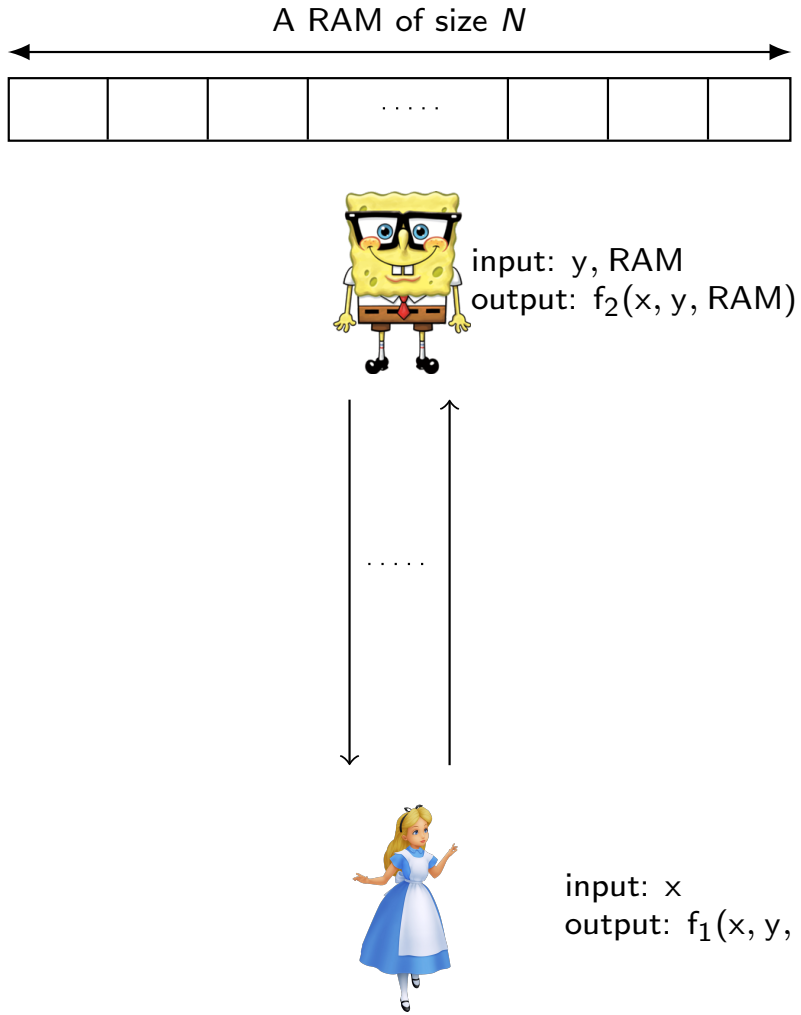
Oblivious RAM (ORAM)

- Privacy of Alice input
  - *ORAM*: $O(\log N)$
- Privacy of Bob input
  - No guarantee
- Construction
  - Imagine the original data in your head
  - Translate access to original data



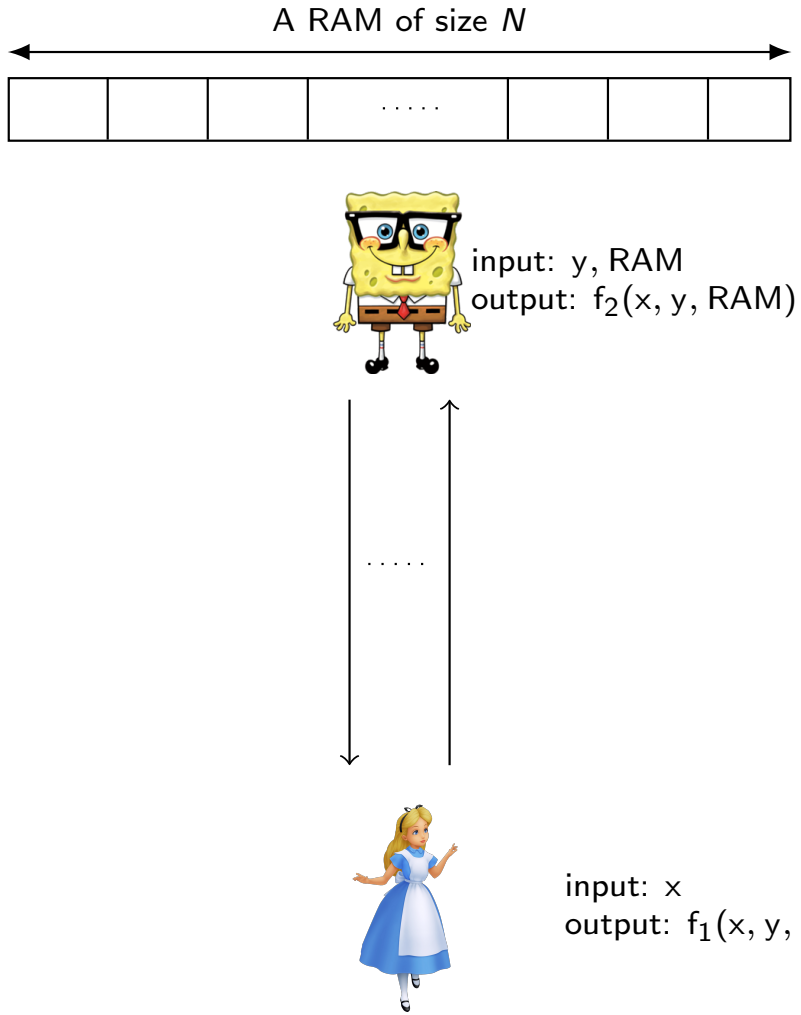
Oblivious RAM (ORAM)

- Privacy of Alice input
  - *ORAM*: $O(\log N)$
- Privacy of Bob input
  - No guarantee
- Construction
  - Imagine the original data in your head
  - Translate access to original data
    - multiple actual memory access



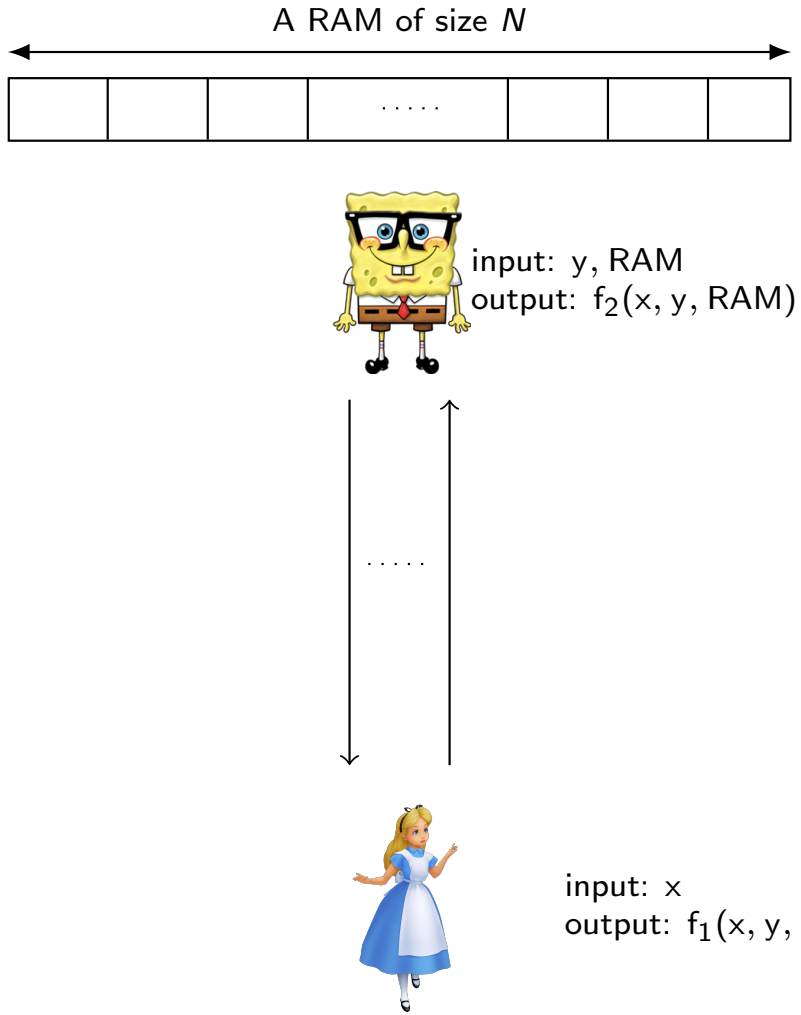
Oblivious RAM (ORAM)

- Privacy of Alice input
  - *ORAM*: $O(\log N)$
- Privacy of Bob input
  - No guarantee
- Construction
  - Imagine the original data in your head
  - Translate access to original data
  - Re-position the original data within the data structure



Oblivious RAM (ORAM)

- Privacy of Alice input
  - *ORAM*: $O(\log N)$
- Privacy of Bob input
  - No guarantee
- Construction
  - Imagine the original data in your head
  - Translate access to original data
  - Re-position the original data within the data structure
  - Update *state* for next iteration



Combine ORAM with 2PC

- ORAM is not enough for 2PC

Combine ORAM with 2PC

- ORAM is not enough for 2PC
- 2PC to initialize ORAM

Combine ORAM with 2PC

- ORAM is not enough for 2PC
- 2PC to initialize ORAM
- 2PC for *ORAM step*

Combine ORAM with 2PC

- ORAM is not enough for 2PC
- 2PC to initialize ORAM
- 2PC for *ORAM step*
  - compute next instruction
    - Given the "state" & last read "data"

Combine ORAM with 2PC

- ORAM is not enough for 2PC
- 2PC to initialize ORAM
- 2PC for *ORAM step*
  - compute next instruction
    - Given the "state" & last read "data"
  - perform next Instruction
    - Access to original array $\rightarrow$ many actual instructions

Combine ORAM with 2PC

- ORAM is not enough for 2PC
- 2PC to initialize ORAM
- 2PC for *ORAM step*
  - compute next instruction
    - Given the "state" & last read "data"
  - perform next Instruction
    - Access to original array $\rightarrow$ many actual instructions
  - Secret share "state" & last read "data"

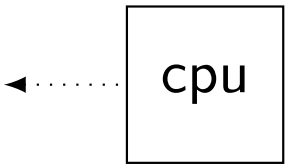
Combine ORAM with 2PC

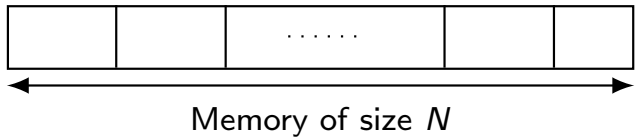
- ORAM is not enough for 2PC
- 2PC to initialize ORAM
- 2PC for *ORAM step*
  - compute next instruction
    - Given the "state" & last read "data"
  - perform next Instruction
    - Access to original array $\rightarrow$ many actual instructions
  - Secret share "state" & last read "data"
  - Compute until "state" equals "halt"

Semi-honest RAM-2PC [GKK+12]

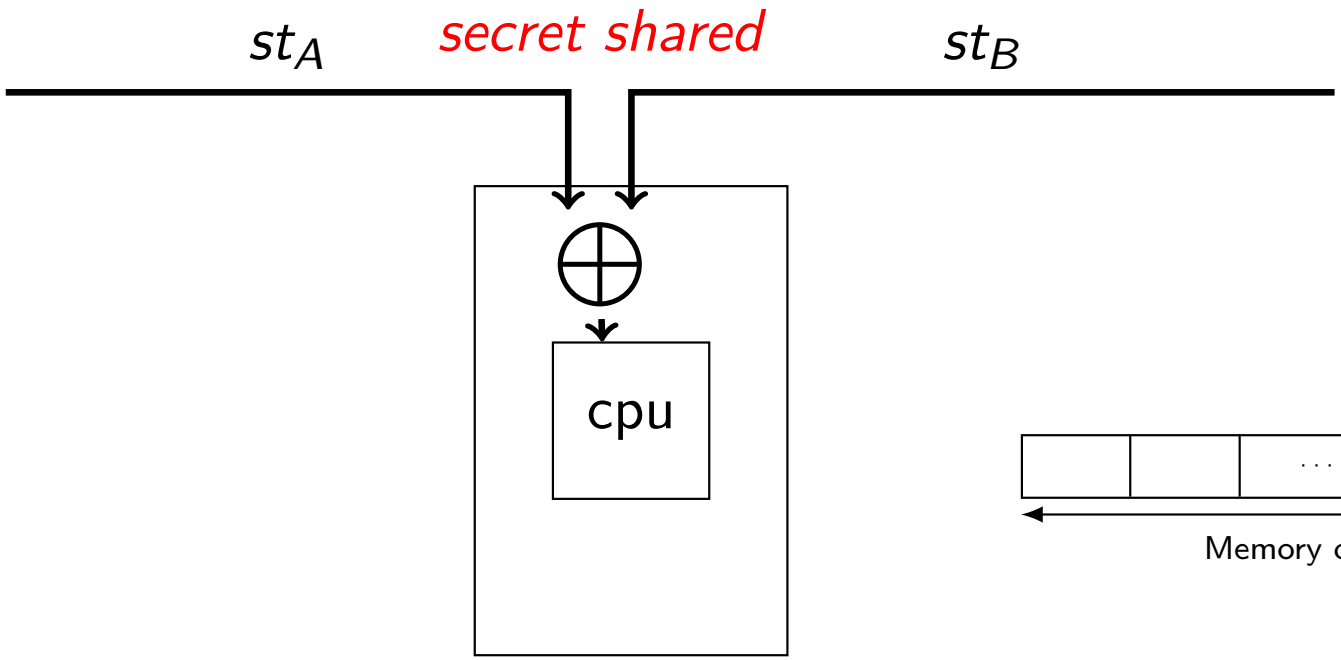
Garbled Circuits

for a *Single* ORAM Step.

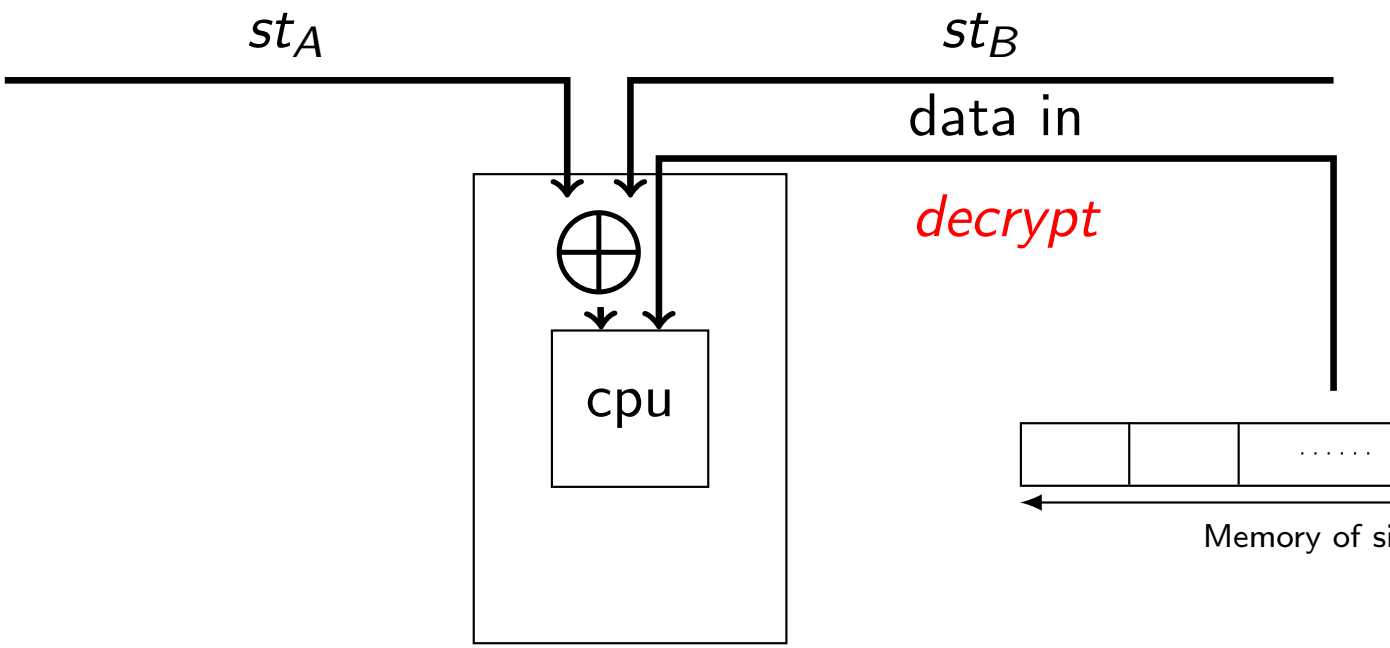




Semi-honest RAM-2PC [GKK+12]



Semi-honest RAM-2PC [GKK+12]



Semi-honest RAM-2PC [GKK+12]

Possible attacks

- Garbler may garble different circuits
  - Needs *cut-and-choose*

Possible attacks

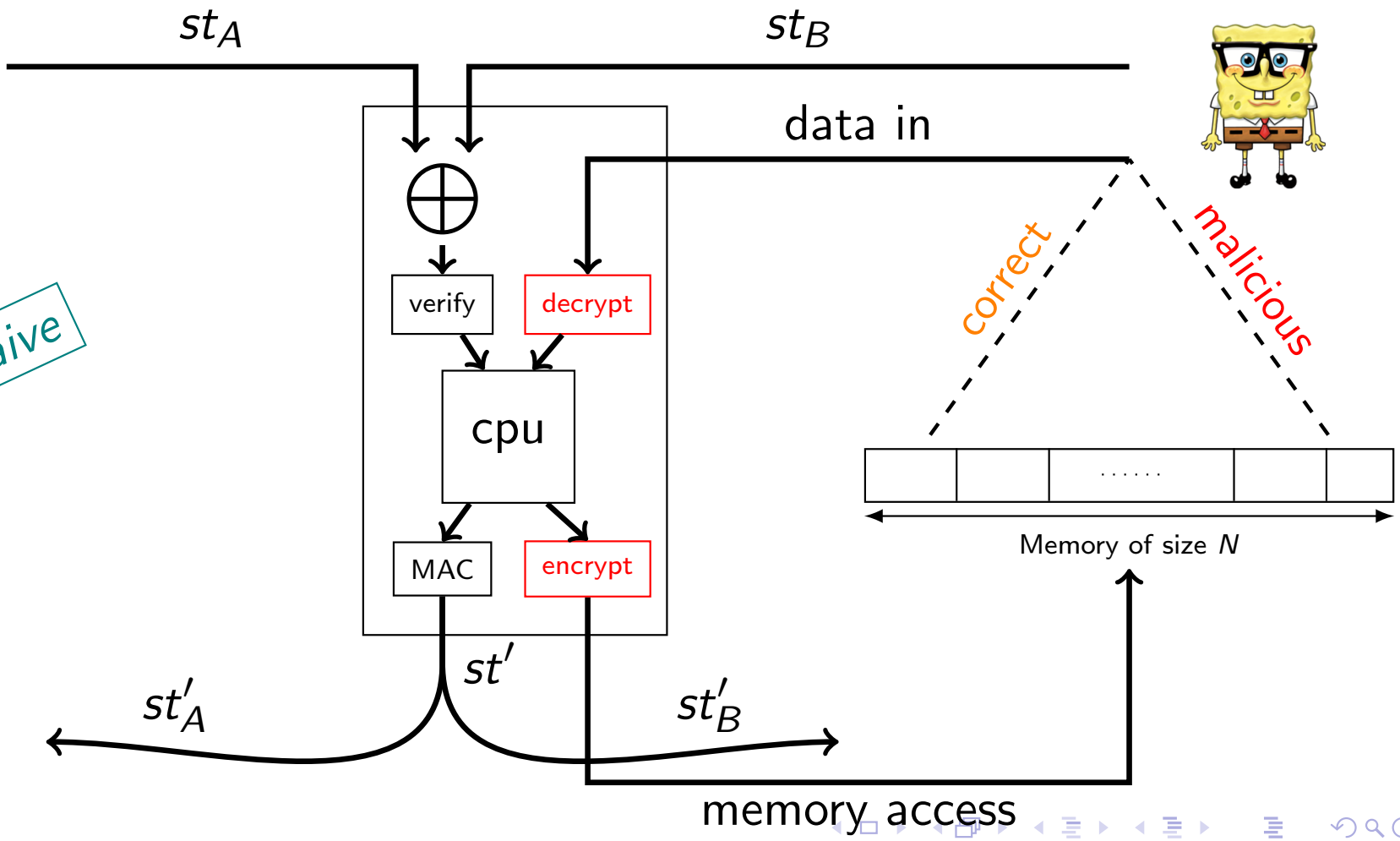
- Garbler may garble different circuits
  - Needs *cut-and-choose*
- Parties use incorrect inputs to the circuit, i.e.
  - Original inputs
  - Shares of the state
  - Memory contents
  - Needs *consistency* and *authenticity* checks

Naive Solution for Malicious RAM-2PC

- *Integrity and consistency of state*

Naive Solution for Malicious RAM-2PC

- *Memory privacy/consistency*



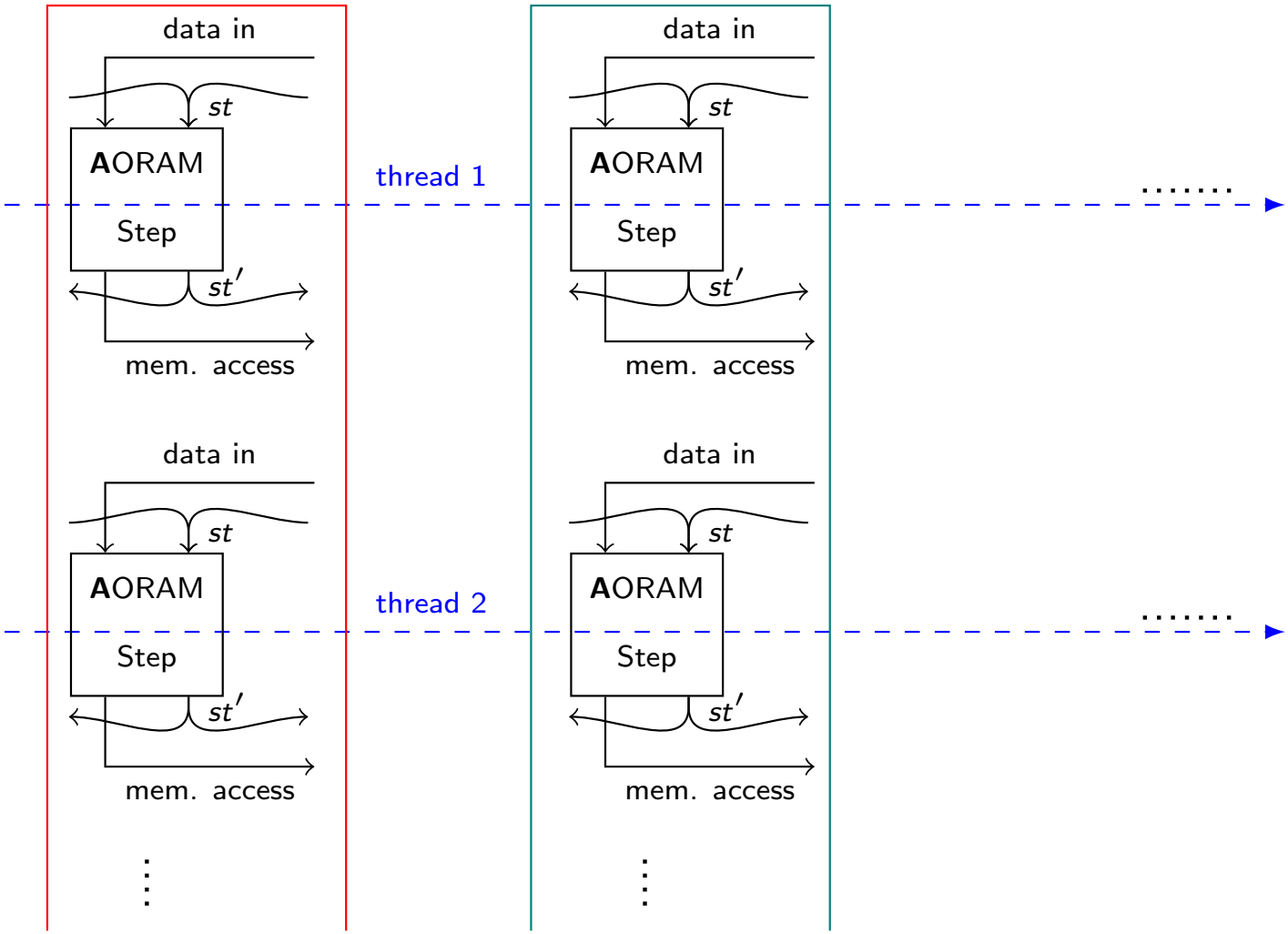
Naive Solution for Malicious RAM-2PC

Augmented ORAM Step.

(*AORAM* Step)

Naive Cut-and-Choose Approach

- Separate Malicious 2PCs



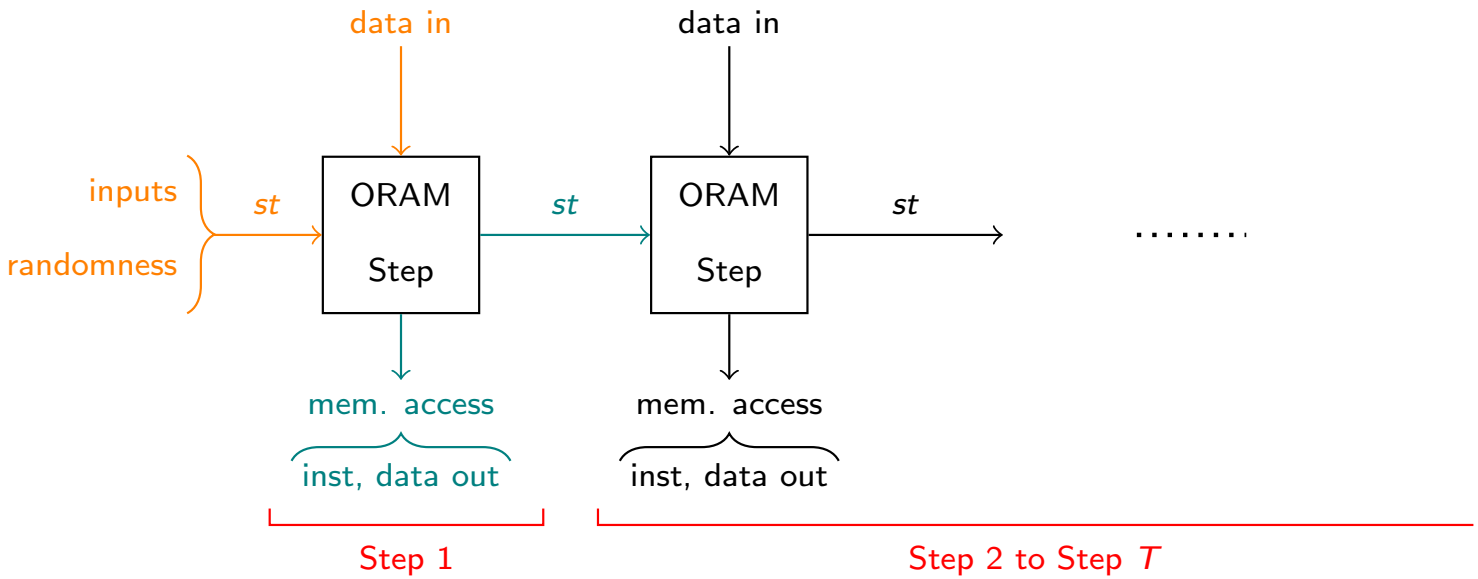
Our Approach: Two Protocols

- Batching Protocol, based on LEGO idea [FJN+13, NO09]
- Streaming Cut-and-Choose Protocol
  - This talk

Stream Cut-and-Choose Protocol

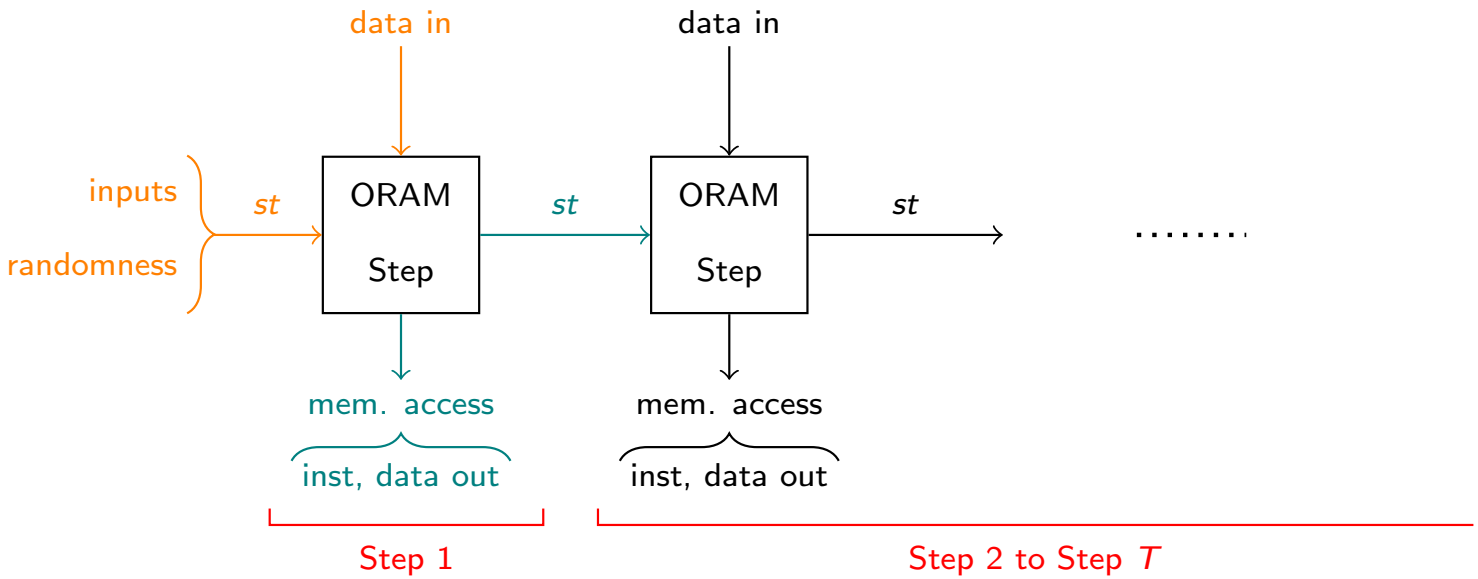
Garbled Values

- Memory and state as *garbled values*
  - No encryption
  - No MAC/Verification



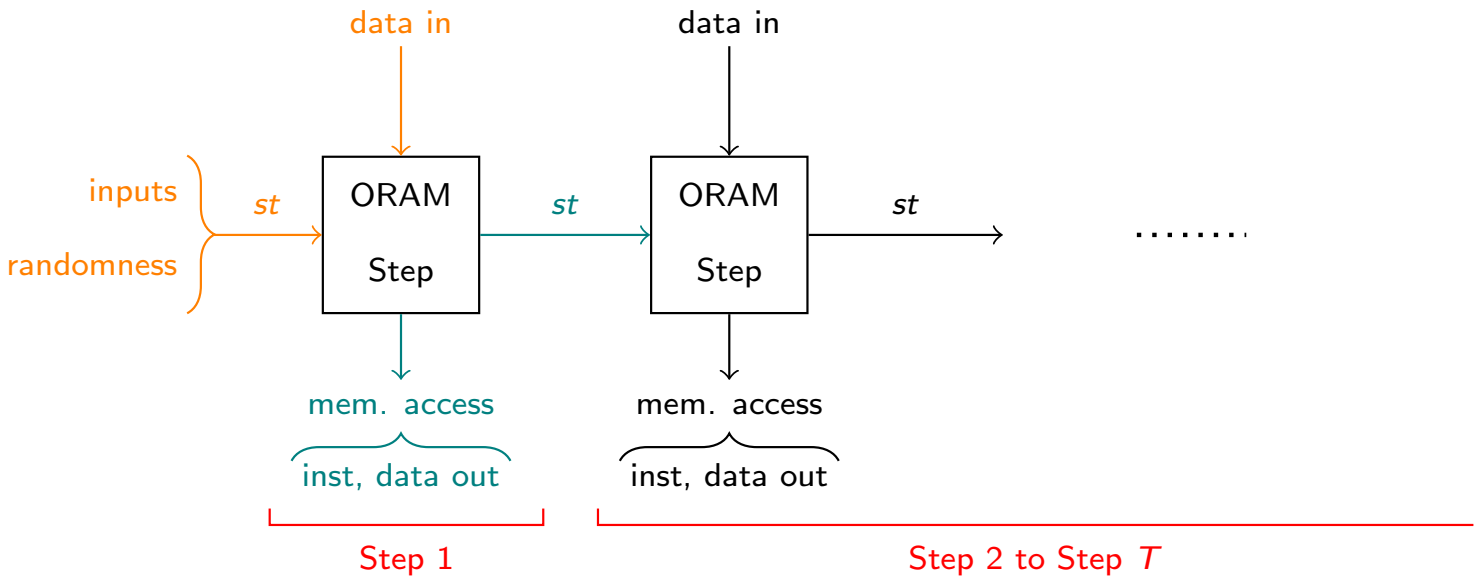
Garbled Values

- Memory and state as *garbled values*
  - No encryption
  - No MAC/Verification
    - *Authenticity*
    - *Privacy*



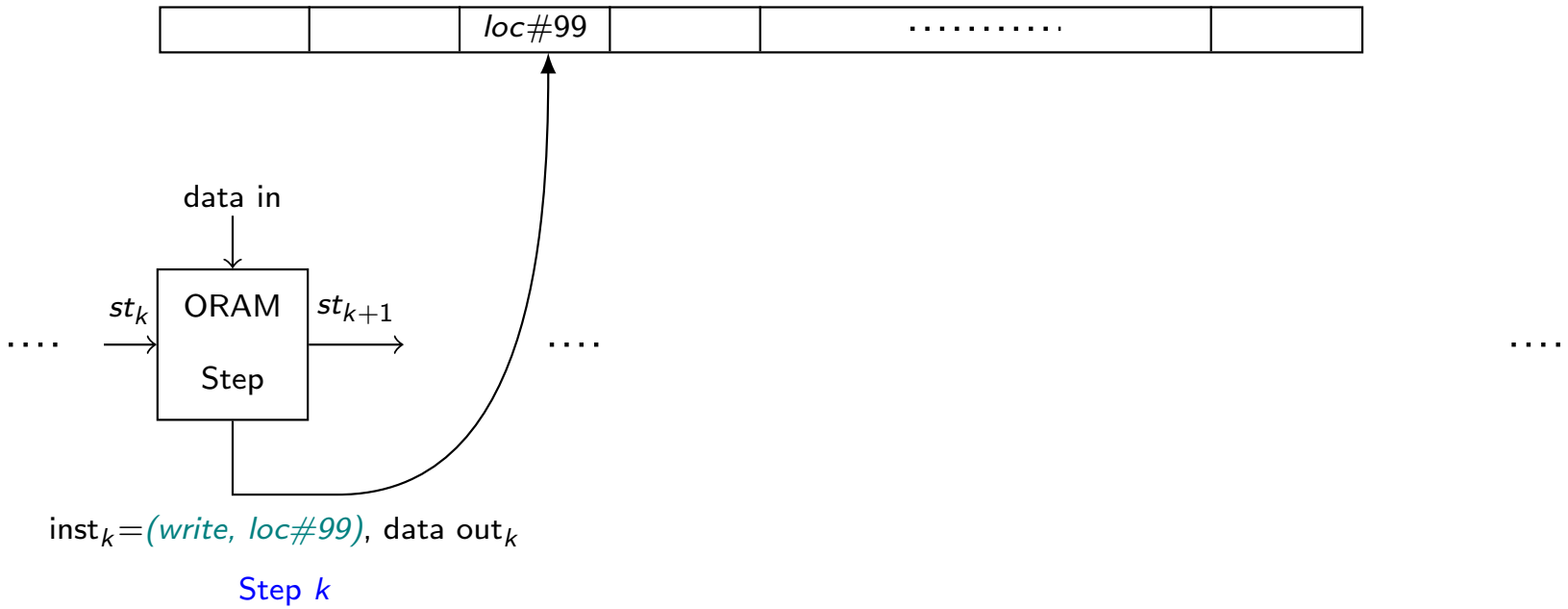
Garbled Values

- Memory and state as *garbled values*
  - No encryption
  - No MAC/Verification
    - *Authenticity*
    - *Privacy*
- *Re-use* garbled values
  - No input consistency checks in intermediate circuits



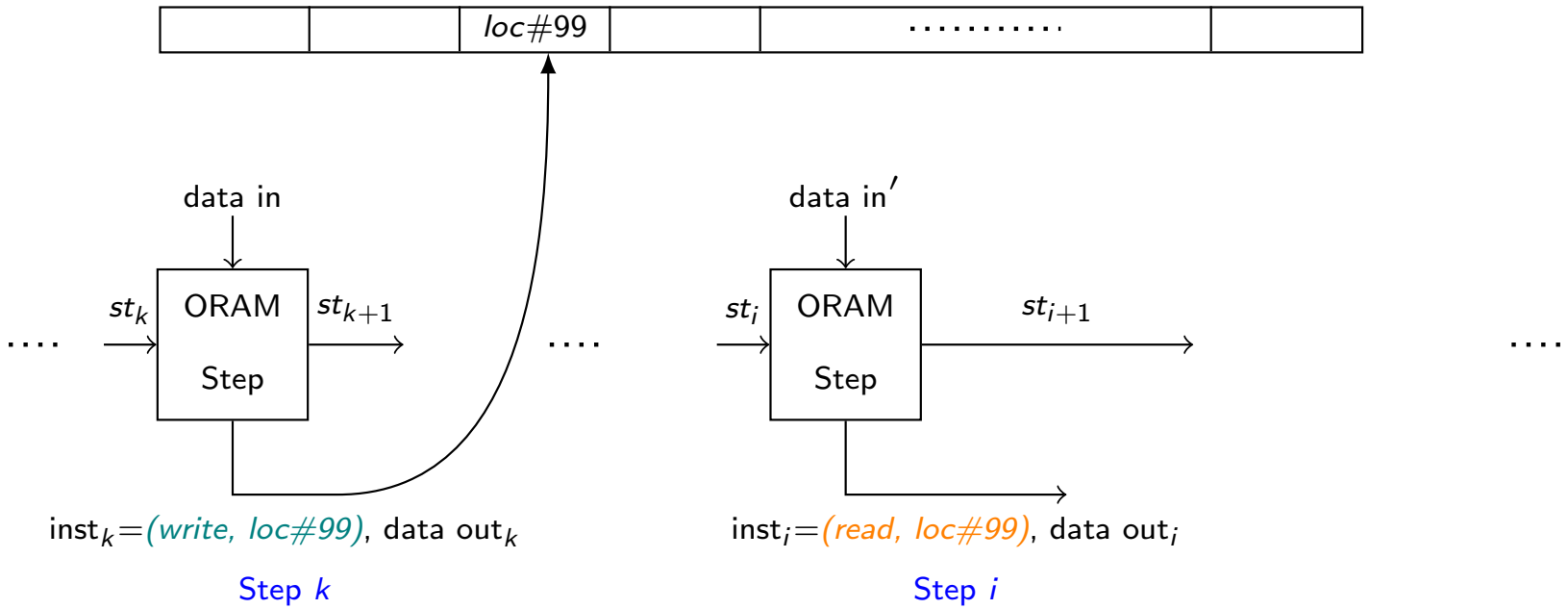
Memory Privacy/Consistency

- Memory items are garbled values
  - Bob reports the memory location
  - Correctness: *cut-and-choose*
- Consistency of memory location [MR13]



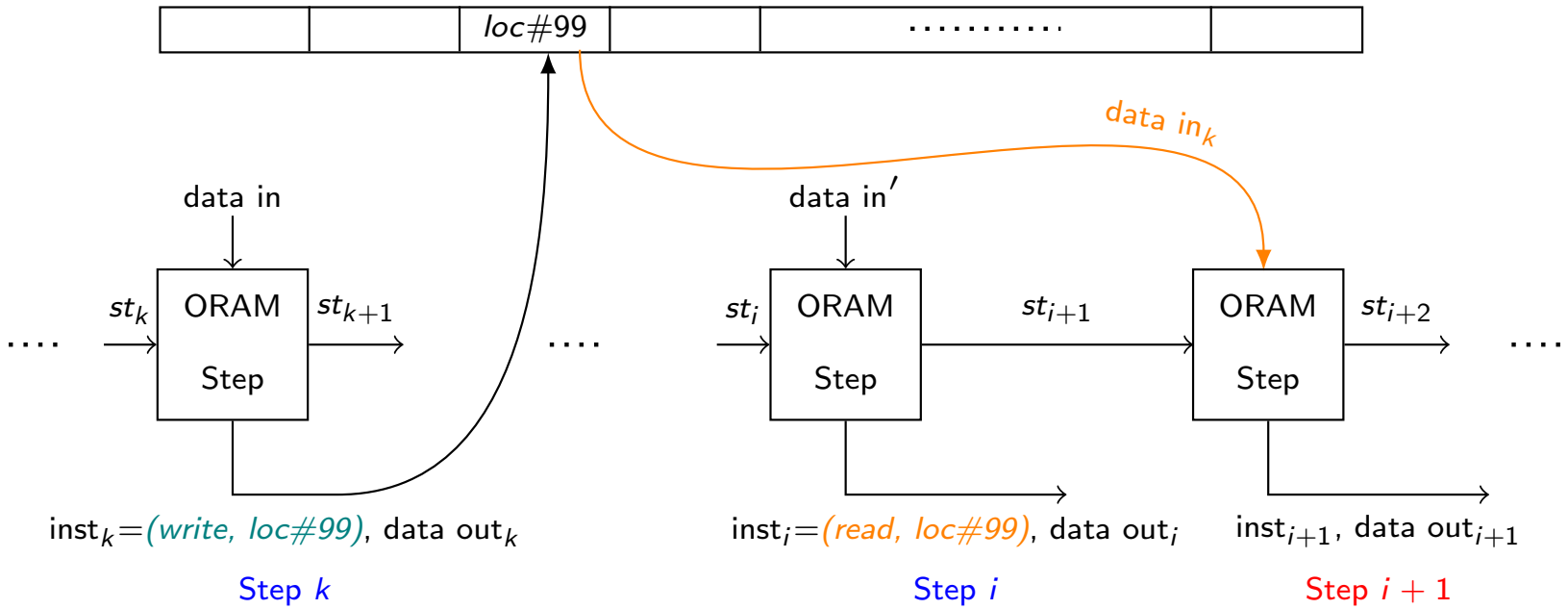
Memory Privacy/Consistency

- Memory items are garbled values
  - Bob reports the memory location
  - Correctness: *cut-and-choose*
- Consistency of memory location [MR13]



Memory Privacy/Consistency

- Memory items are garbled values
  - Bob reports the memory location
  - Correctness: *cut-and-choose*
- Consistency of memory location [MR13]

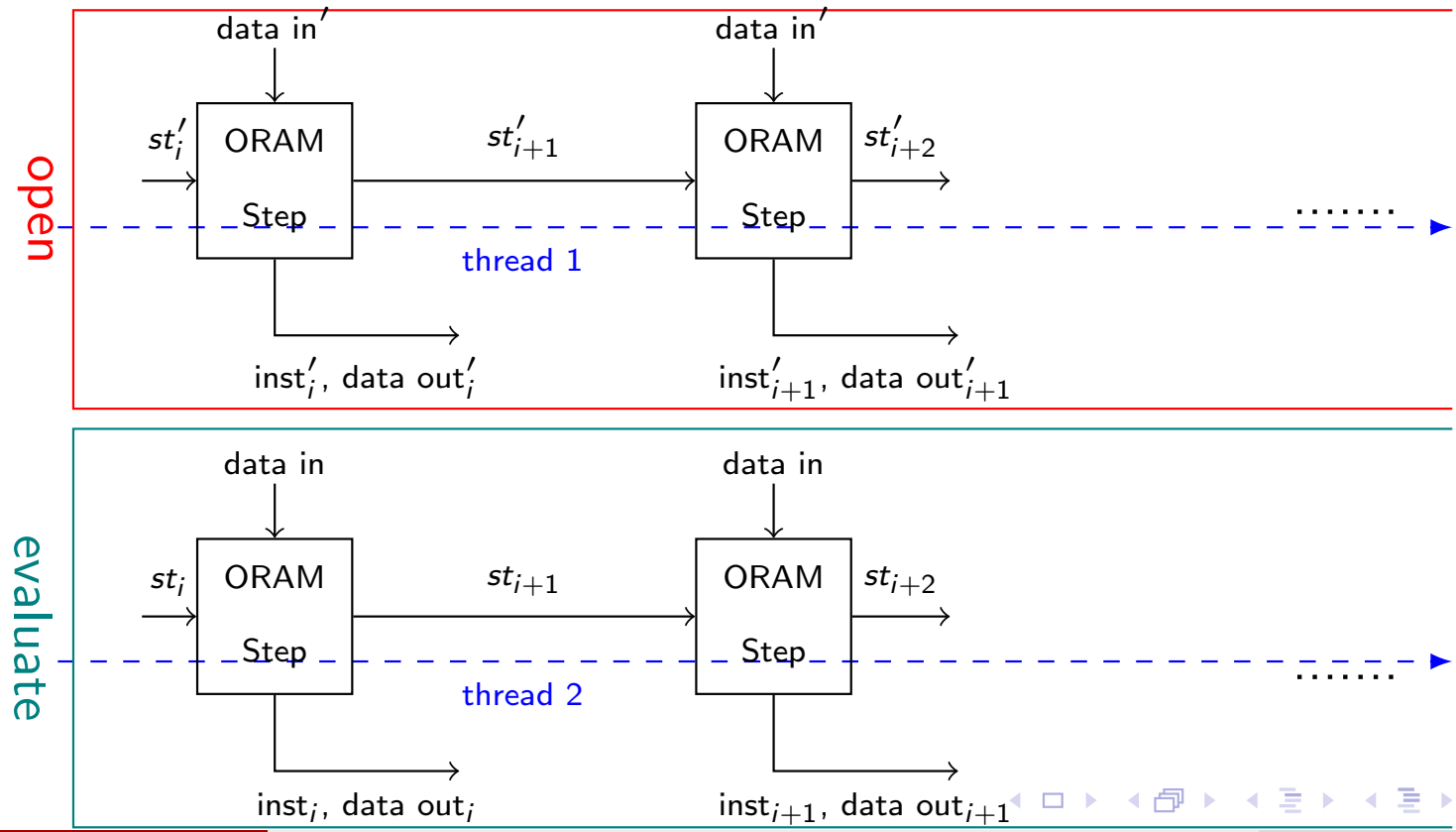


Step Generation

- ... hence:
  - Given inst_i
  - Step $i+1$ is generated *after* Step i is *evaluated*

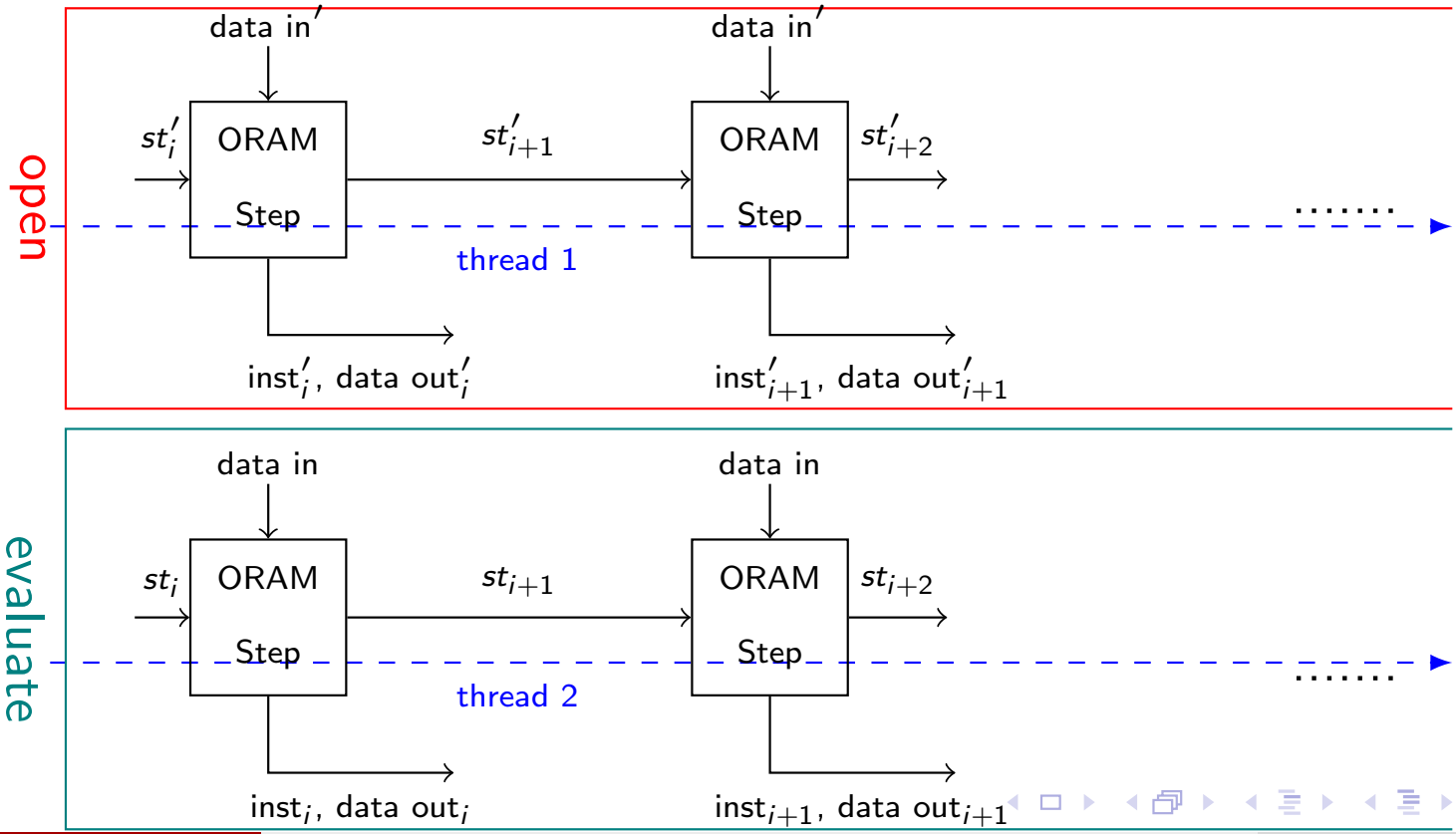
A Single Cut-and-Choose

- One 2PC vs. many 2PCs
  - Input consistency check: *Once at the beginning*



A Single Cut-and-Choose

- One 2PC vs. many 2PCs
  - Input consistency check: *Once at the beginning*
  - Cut-and-Choose rely on one correct circuit
    - Also known as *cheating recovery*
    - Once at the end vs. after each step



Comparison

Table : Overhead

	Naive implementation	Streaming cut-and-choose
Circuit Size (non-XOR gates)	$T \times 154.36 \times 2^{20}$	120
Alice Storage	0	$5MB + \log T \times 40KB$
Input Consistency Checks	$O(T \times IC \times ND)$	0

- Where
  - T is the running time of the RAM
  - IC is the overhead of input consistency check for one bit of data on "s" garbled circuits.
  - S is statistical security parameter
  - N is the length of memory
  - D is the length of ORAM metadata

References

Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi.
MiniLEGO: Efficient secure two-party computation from general assumptions.
In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 537–556. Springer, May 2013.

S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis.
Secure two-party computation in sublinear (amortized) time.
In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12: 19th Conference on Computer and Communications Security*, pages 513–524. ACM Press, October 2012.

Payman Mohassel and Ben Riva.
Garbled circuits checking garbled circuits: More efficient and secure two-party computation.
In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 36–53. Springer, August 2013.

Jesper Buus Nielsen and Claudio Orlandi.
LEGO for two-party secure computation.
In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 368–386. Springer, March 2009.

Thank You!