Adaptively Secure Computation For RAM Programs

Laasya Bangalore, Rafail Ostrovsky, Oxana Poburinnaya, and

Muthuramkrishnan Venkitasubramaniam

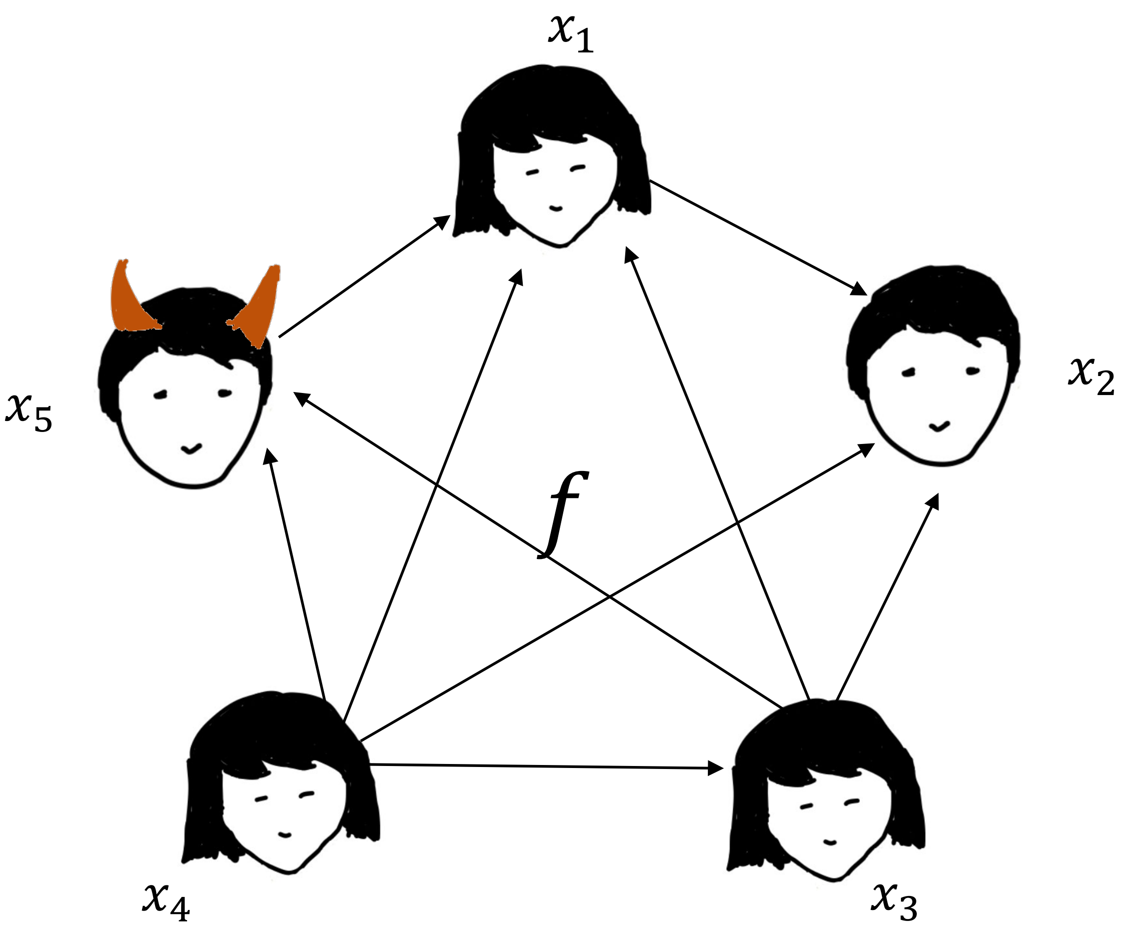
GEORGETOWN UNIVERSITY UCLA LIGERO

Our Result in a Nutshell

Only depends on the RAM complexity of a function

We construct a communication-efficient constant-round 2PC protocol with full adaptive security under minimal assumptions.

Secure Multiparty Computation



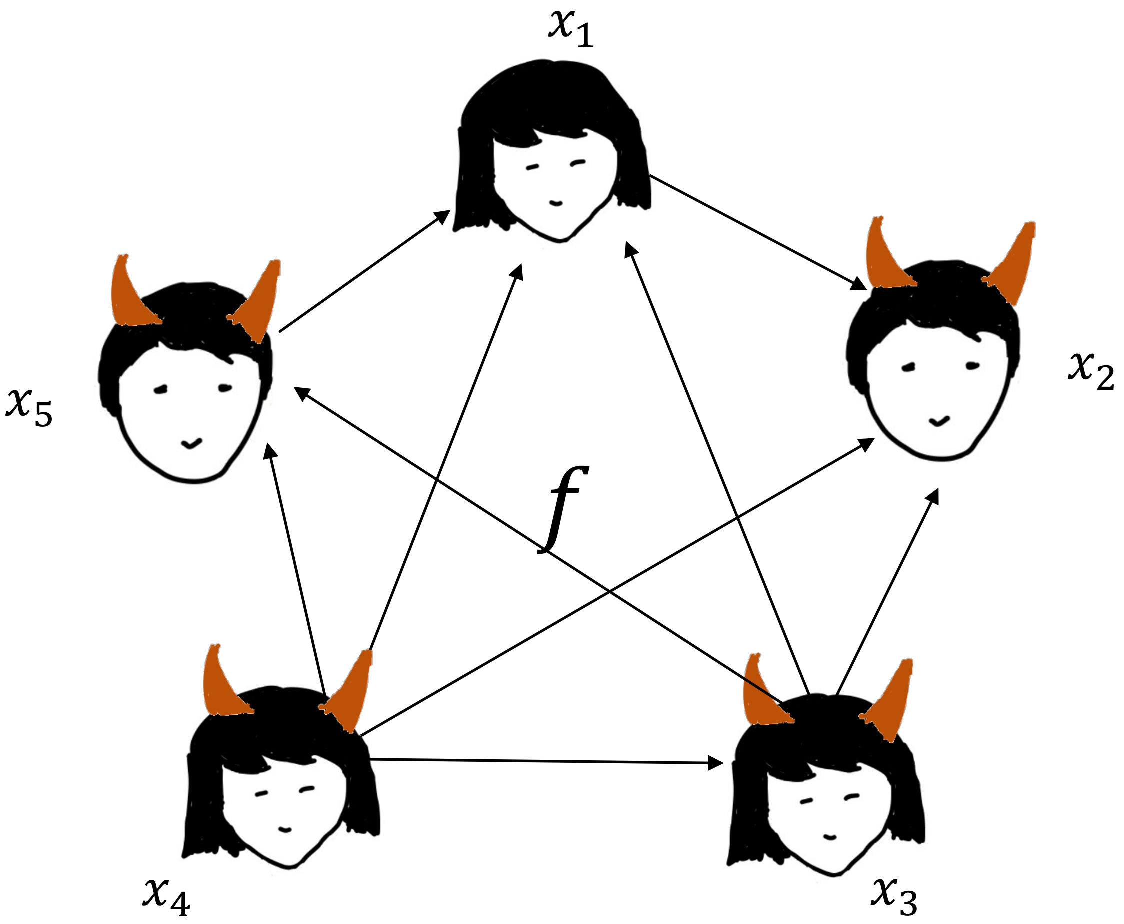
Parties learn *nothing more than* $z = f(x_1, \ldots, x_n)$

<u>Adversarial Corruption Strategies</u>

- Static security

  The set of adversarial parties is fixed in advance before the protocol begins
- Adaptive security

Secure Multiparty Computation

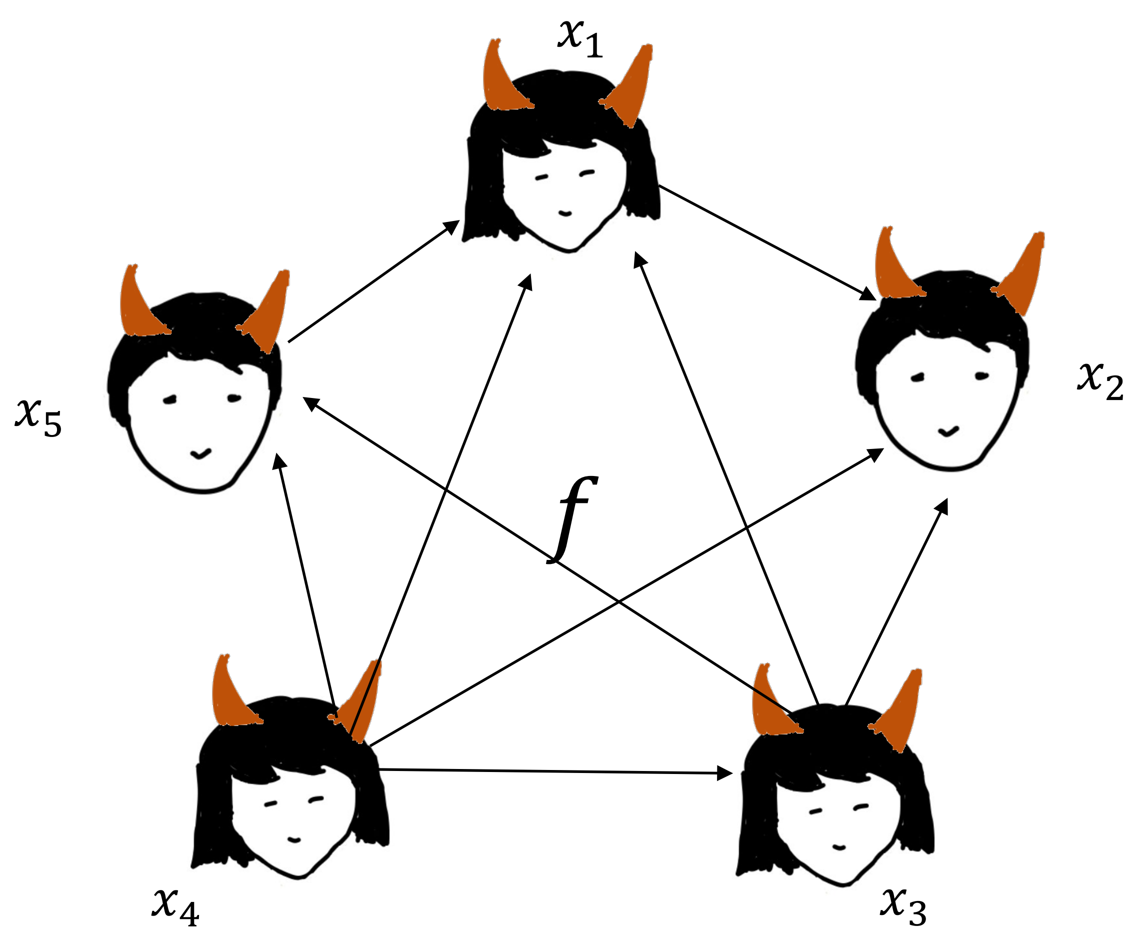


Parties learn *nothing more than* $z = f(x_1, \dots, x_n)$

<u>Adversarial Corruption Strategies</u>

- Static security
  The set of adversarial parties is fixed in advance before the protocol begins
- Adaptive security
  The adversary can choose whom to corrupt during the execution of the protocol.

Fully Adaptive MPC



Parties learn *nothing more than* $z = f(x_1, \ldots, x_n)$

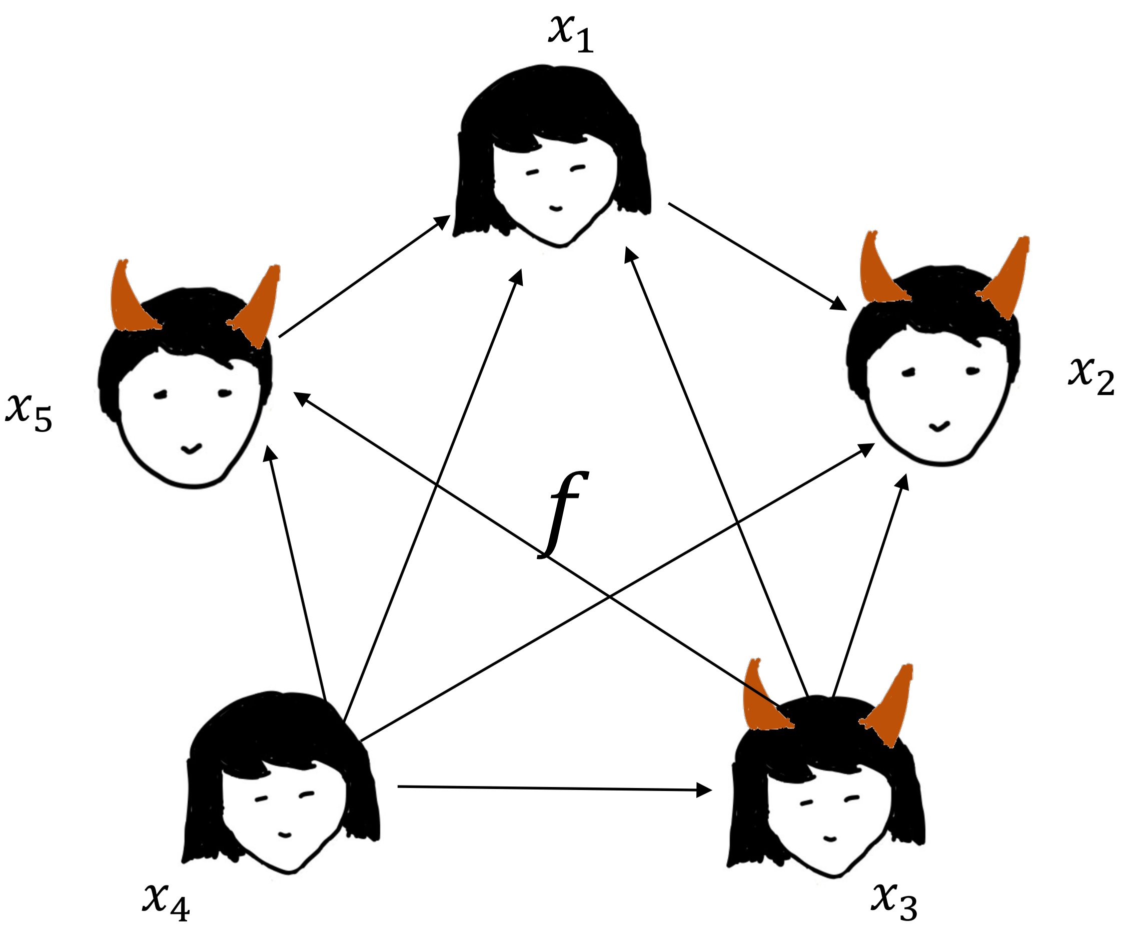
Adversarial Corruption Strategies

- Static security
  The set of adversarial parties is fixed in advance before the protocol begins
- Adaptive security
  The adversary can choose whom to corrupt during the execution of the protocol.

Fully adaptive MPC

- All parties can be corrupted eventually
- Important protocol is used within larger protocol
- Trivial in the static case
- Hard for the adaptive case

Adaptive MPC (Definition)



<u>Adaptive Security:</u> The adversary can choose whom to corrupt during the execution of the protocol.

<u>Simulator:</u>

- Simulate the communication (without knowing the inputs $x_1, x_2, \ldots, x_n$)
- Simulate the randomness of corrupted parties consistent with the communication and its inputs (Equivocation)

Function f can be encoded as either a Circuit or RAM Program

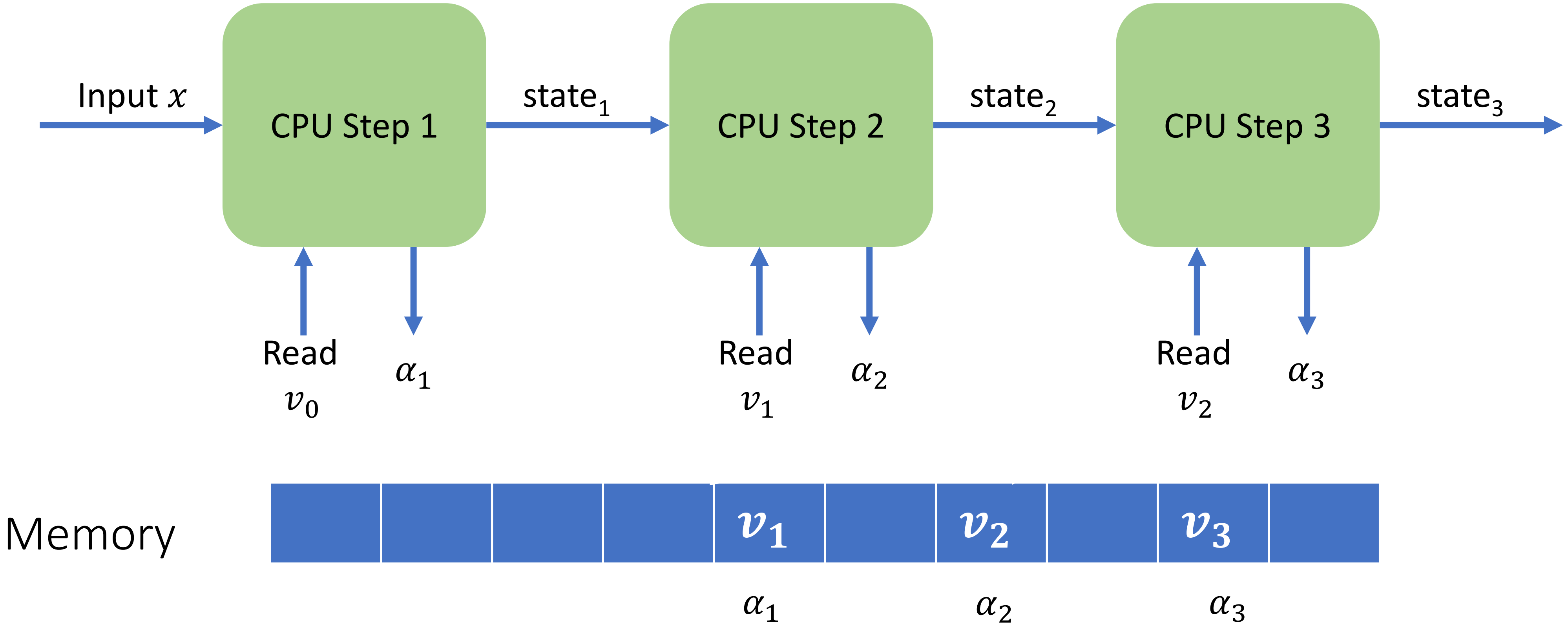
Circuits

- Standard Boolean circuits
- Well suited for highly-structured computation (such as FFT)
- Circuit complexity is expressed in terms of the #gates (say s) in the circuit.

RAM

- Circuits augmented with memory accesses.
- High-level languages are easily reduced to RAM programs.
- RAM complexity is expressed in terms of the running time (say T) of the RAM program.

RAM Model



- A memory access is made at every CPU step.

Prior Work: Adaptive MPC (for Circuits)

Feasibility

- [CLOS02] established the feasibility of fully adaptive protocols (in O(d) rounds)
- Next, we focus on constant round protocols.
- Known for specific assumptions:
  - Reliable erasures Garg and Sahai [GS12]
  - CRS model + iO [CPP15, DKR15, GP15] where CRS size is O(|C|)
- [CPV17] Constant-round protocol under minimal assumptions
- [BLPV18] (Precise rounds) 2-round MPC

d is the depth of the circuit.

Prior Work: Adaptive MPC (for Circuits)

Communication

- [CGP15, DKR15, GP15] (Optimal) Comm. independent of the size of the circuit, but CRS as large as circuit size.
    - Bound on the size of the circuit was required at the time of CRS generation
- [CsW12] Improved both comm. and CRS size is O(d) and assumes CRS + iO
- **Minimal assumptions**: [CPV17, BLPV18] Communication grows quadratically in circuit size.

d is the depth of the circuit.

Can we improve the communication of a constant-round fully adaptive secure computation under minimal assumptions?

YES!

Communication is proportional to square of the RAM complexity of the function

Prior Work: Static/Adaptive MPC (for RAM)

Static MPC

- [LO13, GHORW14, GLOS14, GLO15] Communication prop. to RAM complexity*

*ignoring polylog factors.

Adaptive MPC

- [CPV16, CP16] Communication is dependent to RAM complexity, but required strong assumptions.

The current state of affairs

- [CPV17] Communication prop. to the square of the Boolean complexity but with minimal assumptions.
- [CGP15, CPV16, CsW19, DKR15] Strong assumptions and huge CRS but better communication.

Main Theorem

Theorem: There exists a fully adaptively-secure constant-round garbled RAM with communication proportional to the square of the RAM complexity of the function under minimal assumptions, which is constructed from

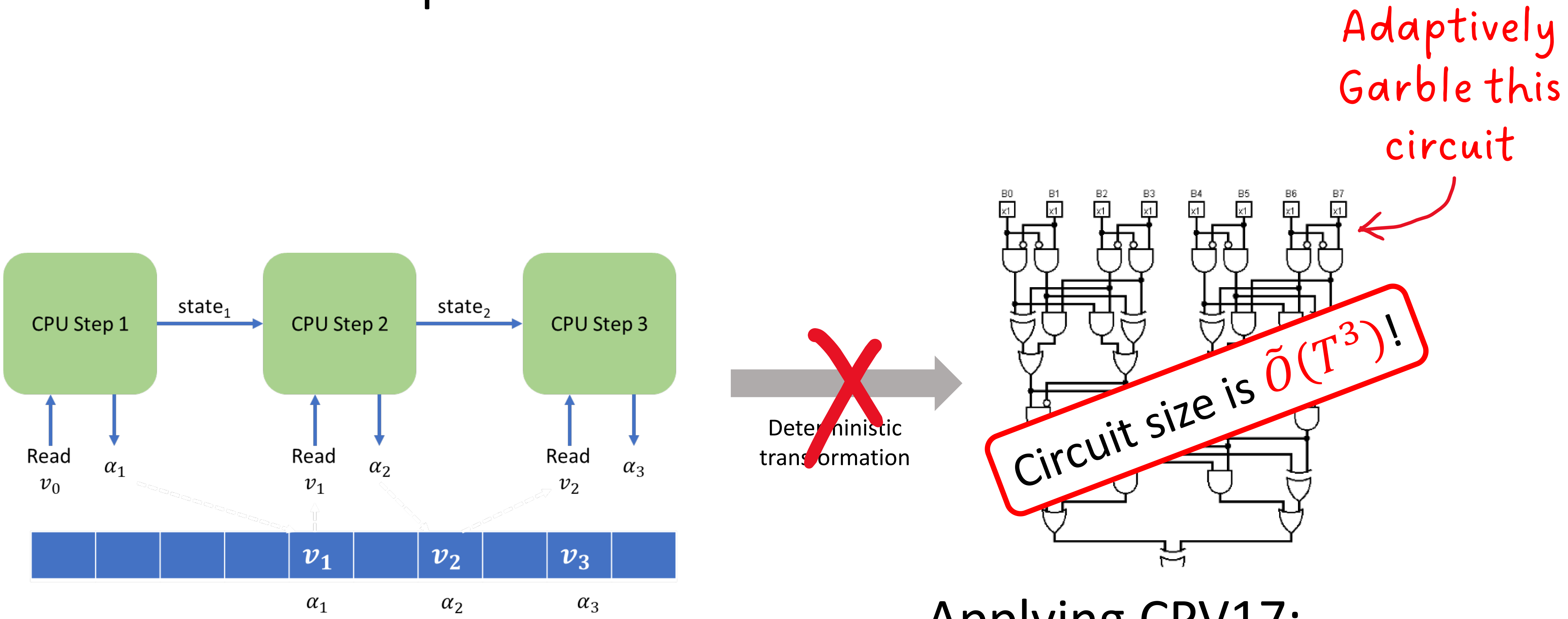
- Equivocal garbed RAM + Equivocal ORAM
- Adaptively secure OT
- non-committing encryption

Focus on 2 PC, Semi-honest setting

Main Ideas:

Challenges Towards Constructing Adaptive Garbled RAM and How To Overcome Them

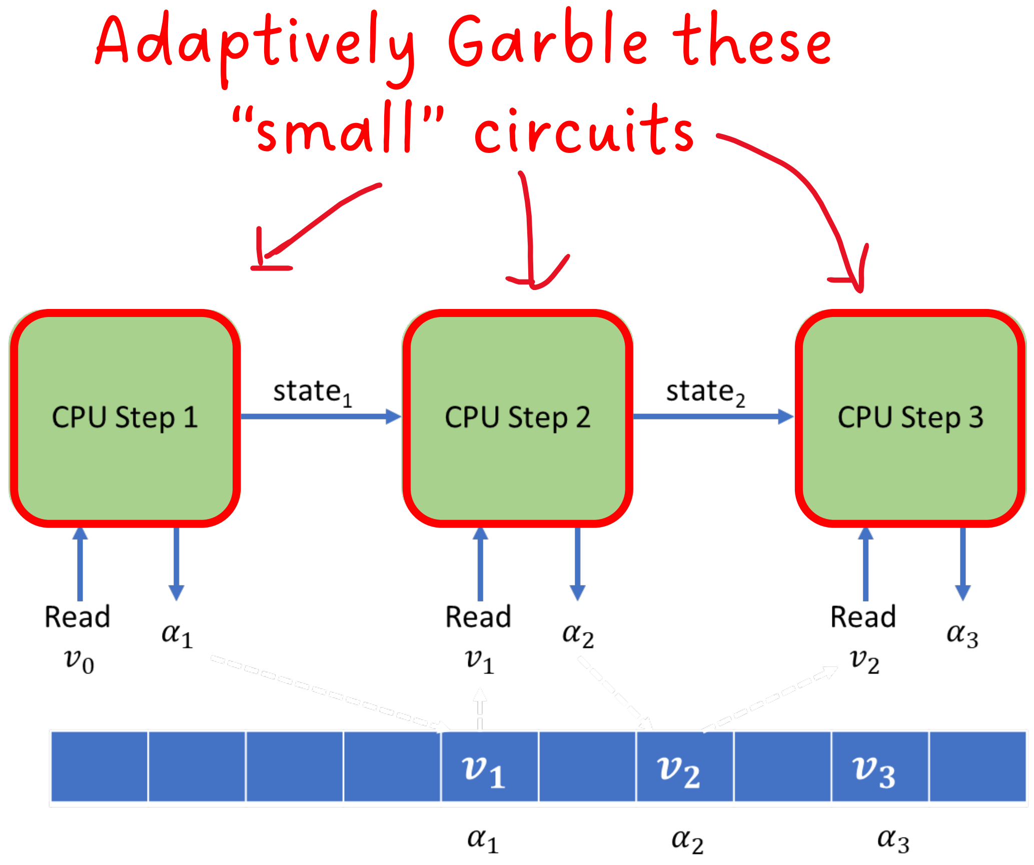
Naïve Attempt: RAM to Circuit Conversion

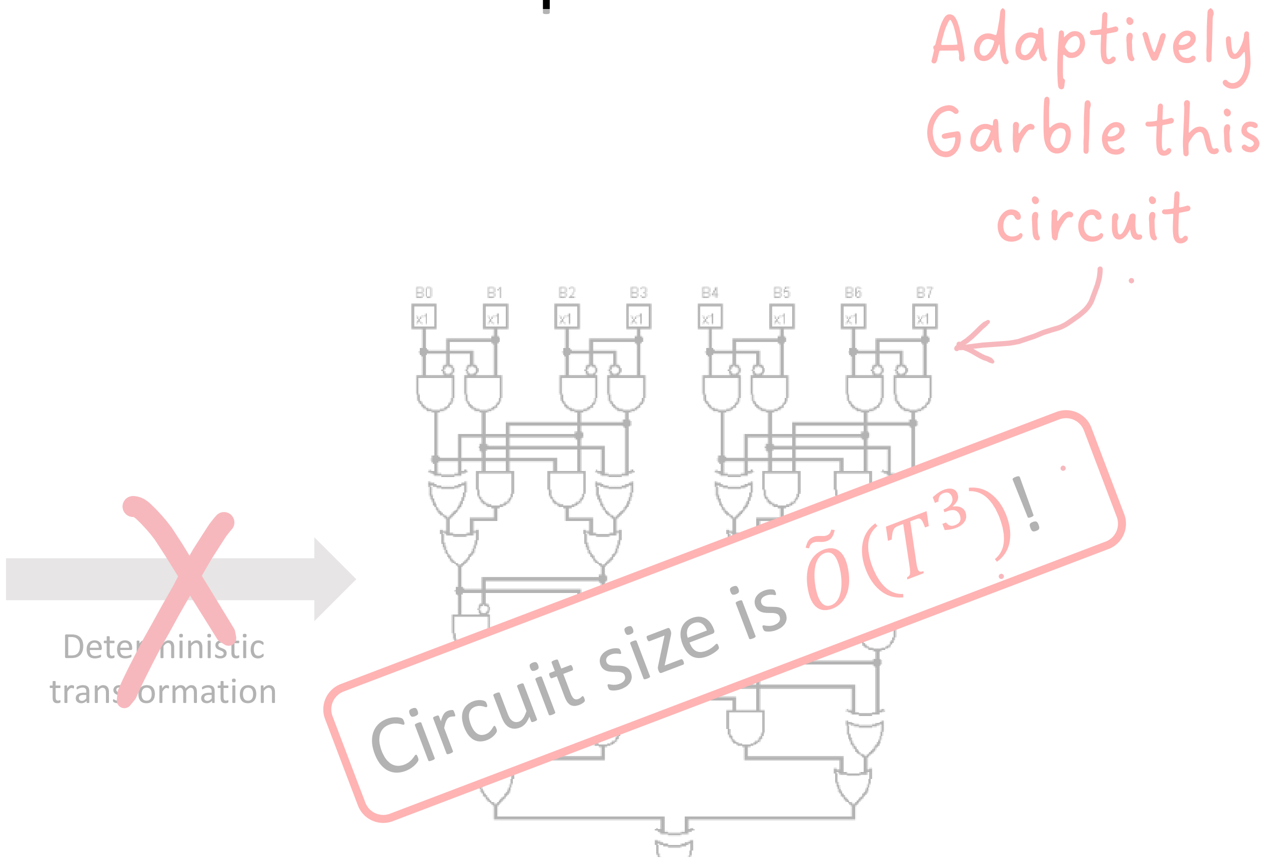


Applying CPV17:
communication = $\tilde{O}(T^6)$!

https://codegolf.stackexchange.com/questions/24834/building-circuit-for-divisibility-by-3

Smarter Attempt: Garble each step circuit…

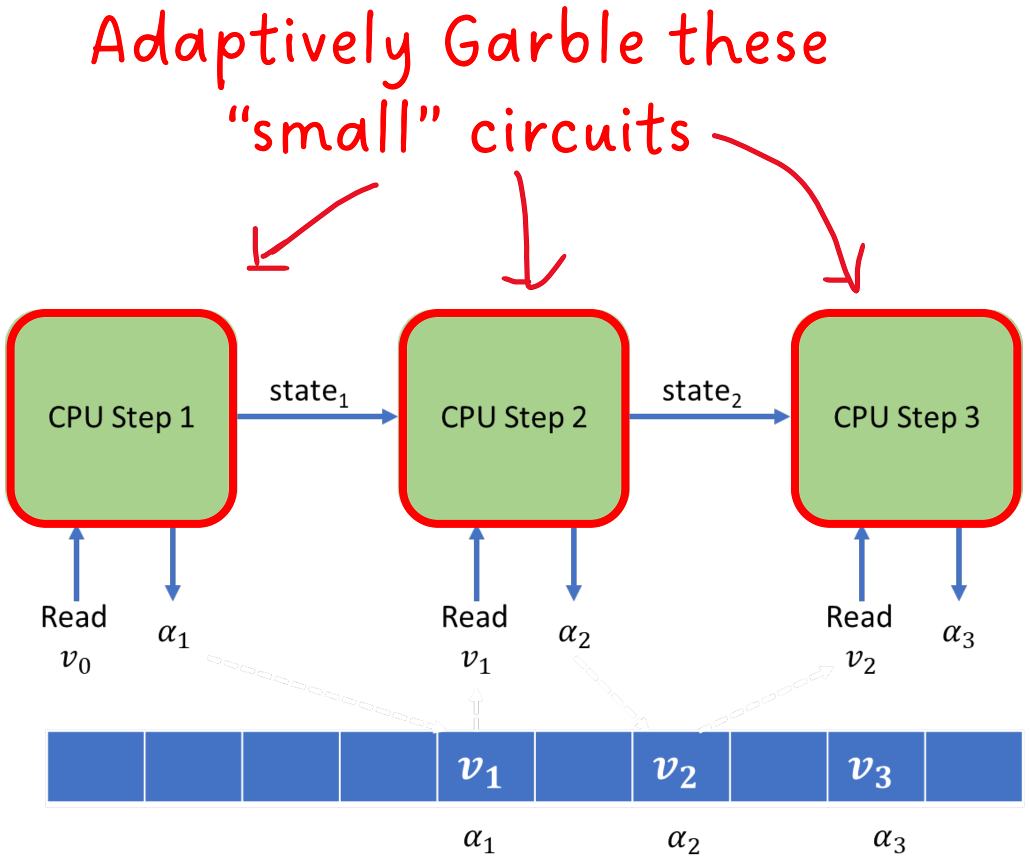




Applying CPV17:
communication = $\tilde{O}(T^6)$!

https://codegolf.stackexchange.com/questions/24834/building-circuit-for-divisibility-by-3

Smarter Attempt: Garble each step circuit…



Challenge I

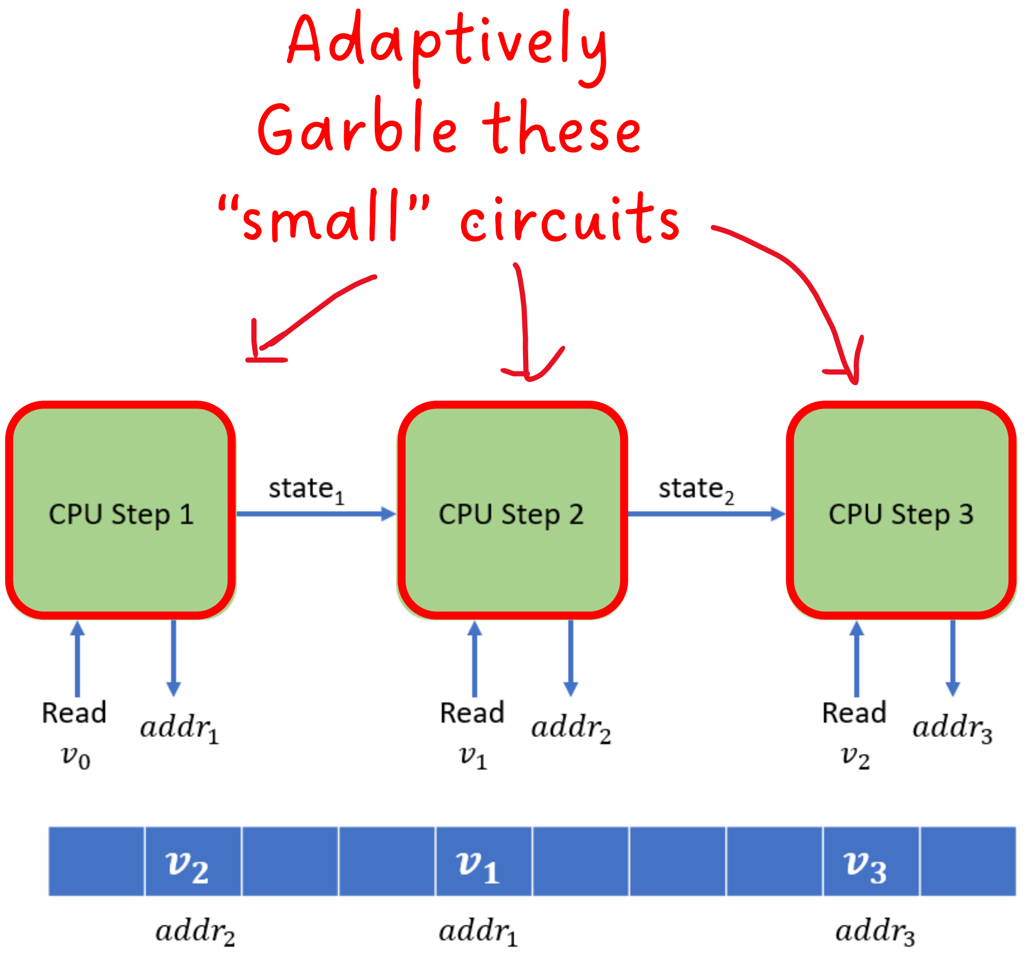
- Memory access patterns may leak information.
- ORAM resolves this issue for static garbled RAM.
- For adaptive security, we require ORAM with additional properties.

Challenge II

- [CPV17] is designed for stand-alone circuits.
- It does not handle external memory accesses.

Other Challenges…

Smarter Attempt: Garble each step circuit…



Challenge I

- Memory access patterns may leak information.
- ORAM resolves this issue for static garbled RAM.
- For adaptive security, we require ORAM with additional properties.

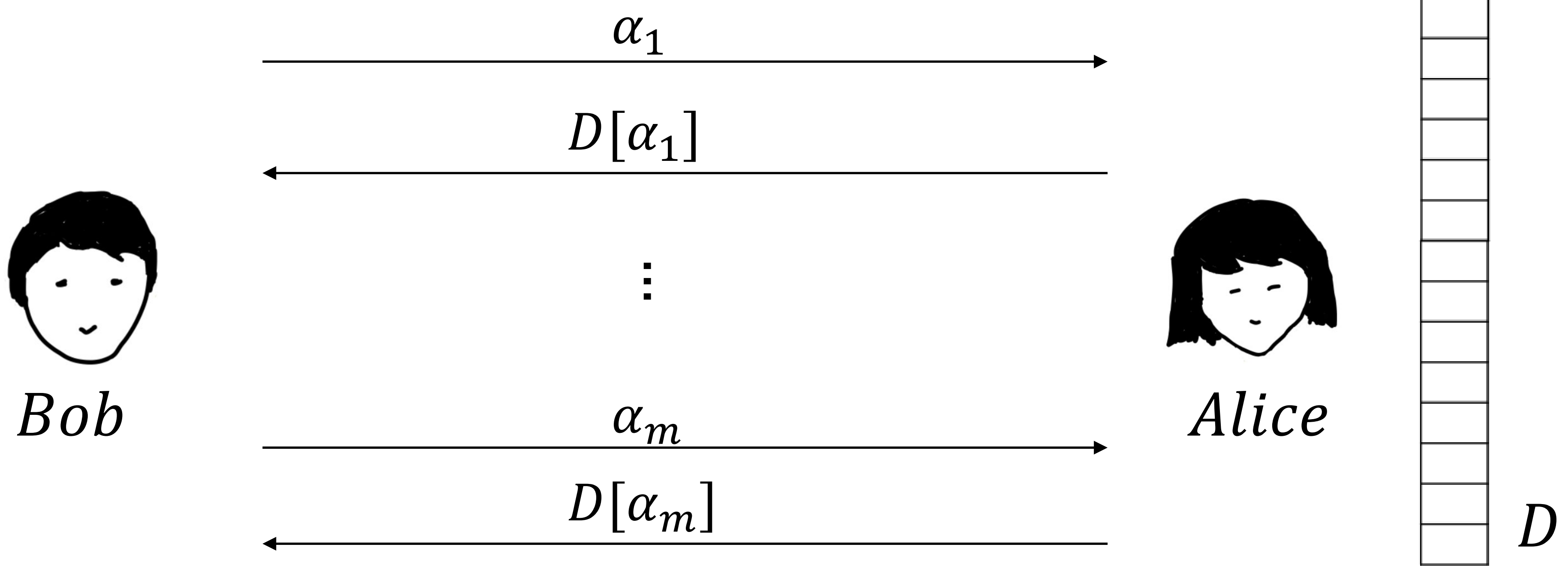
Challenge II

- [CPV17] is designed for stand-alone circuits.
- It does not handle external memory accesses.

Other Challenges…

Addressing Challenge I

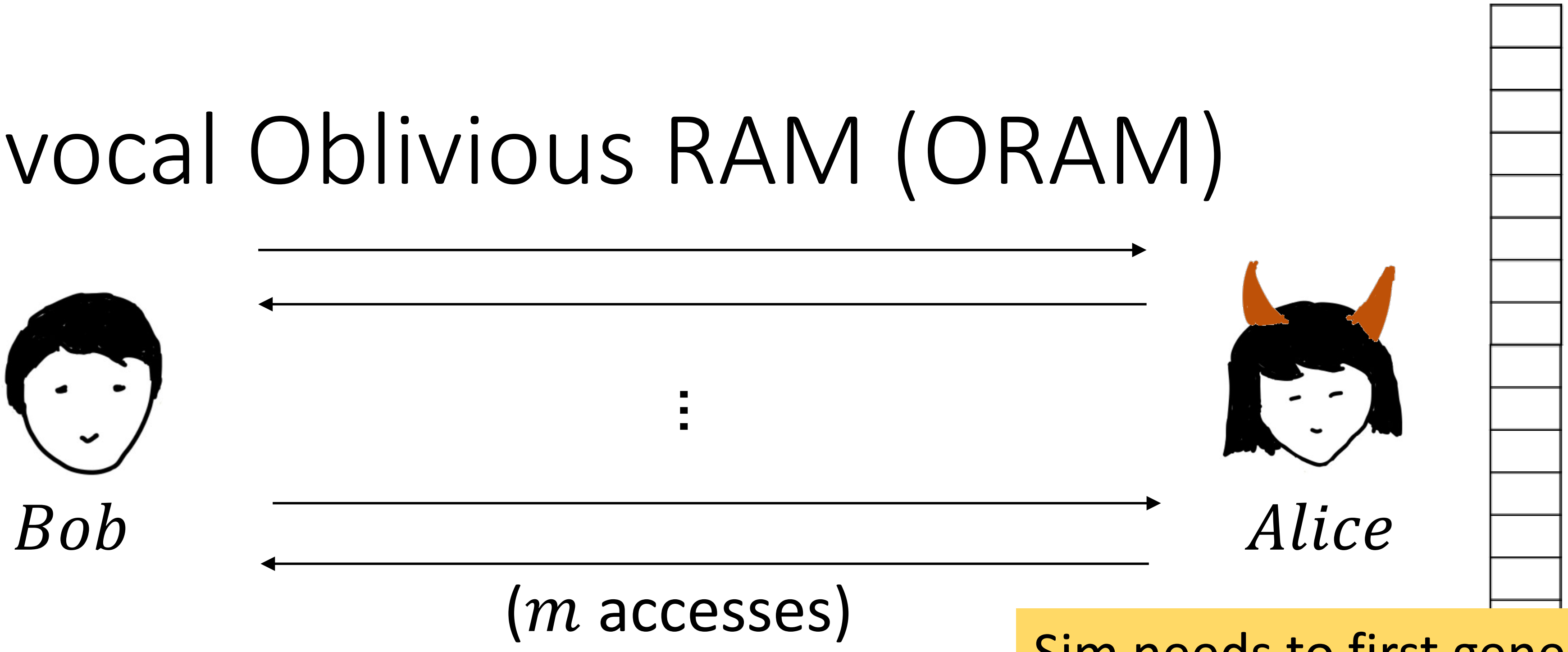
Oblivious RAM (ORAM)



- The memory locations accessed by RAM are input-dependent.
- This leaks information about Bob's input!

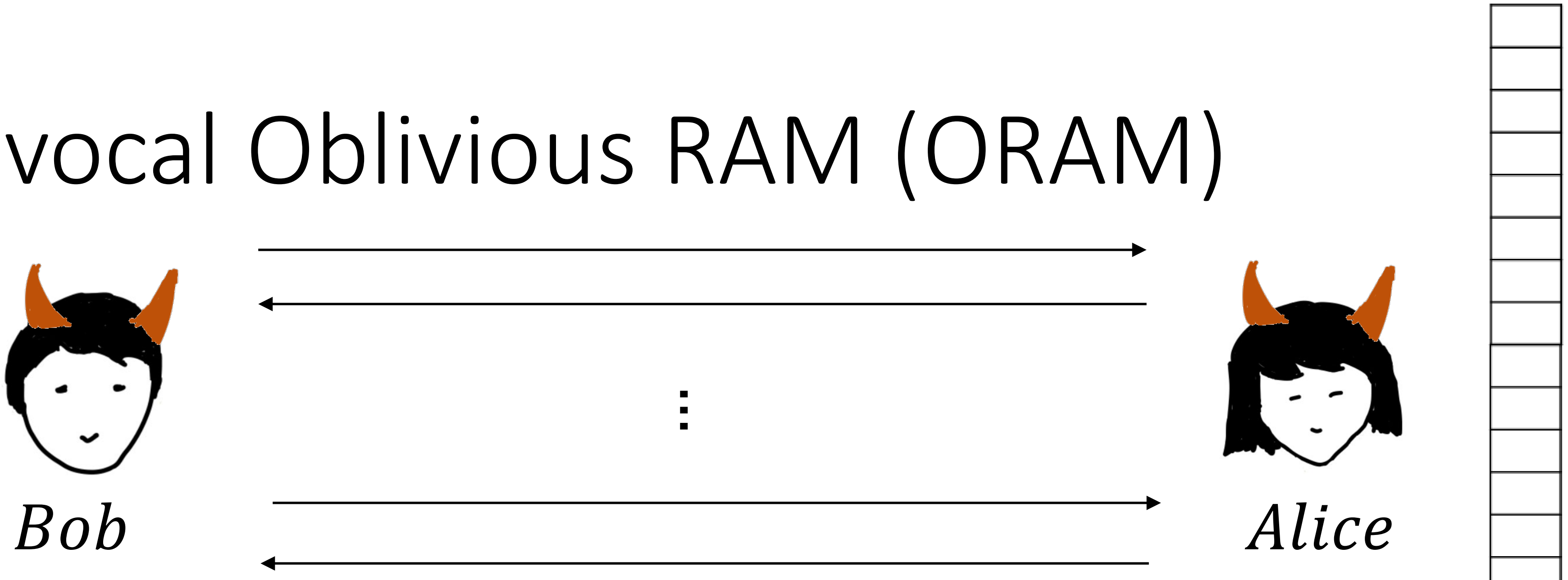
Use ORAM!

Equivocal Oblivious RAM (ORAM)



Sim needs to first generate "fake" oblivious memory accesses without knowing Bob's inputs.

Equivocal Oblivious RAM (ORAM)

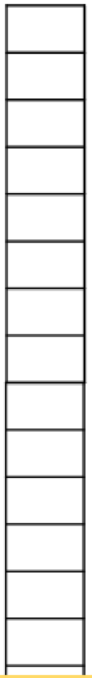


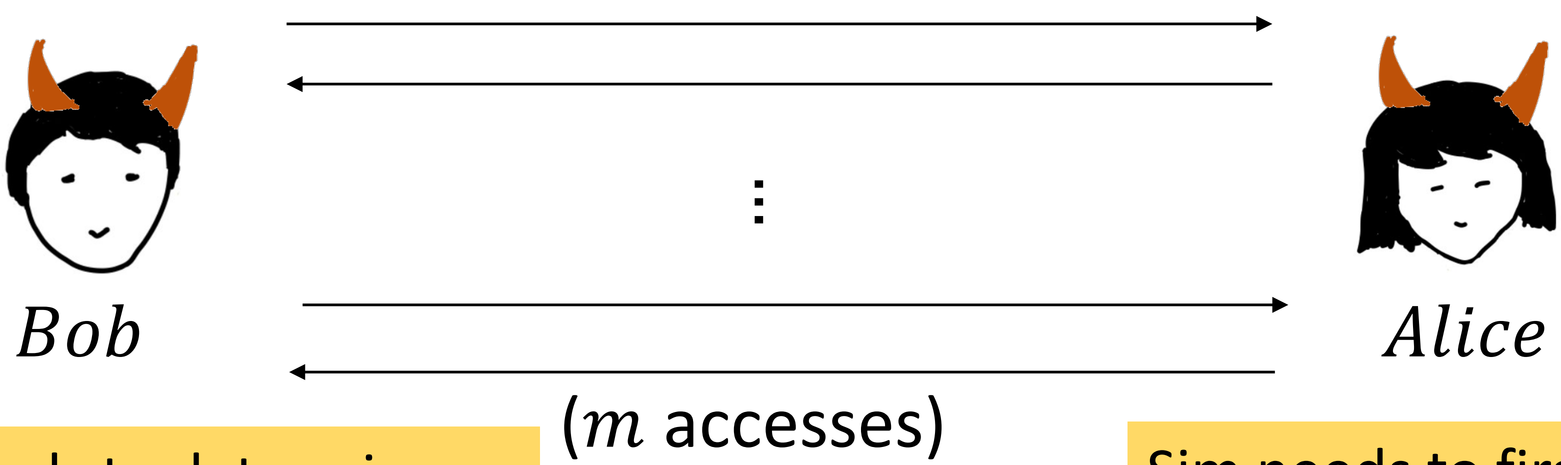
(m accesses)

Then, Sim needs to determine the randomness to justify that the “fake” oblivious memory access pattern is consistent with Bob’s inputs.

Sim needs to first generate “fake” oblivious memory accesses without knowing Bob’s inputs.

Equivocal Oblivious RAM (ORAM)



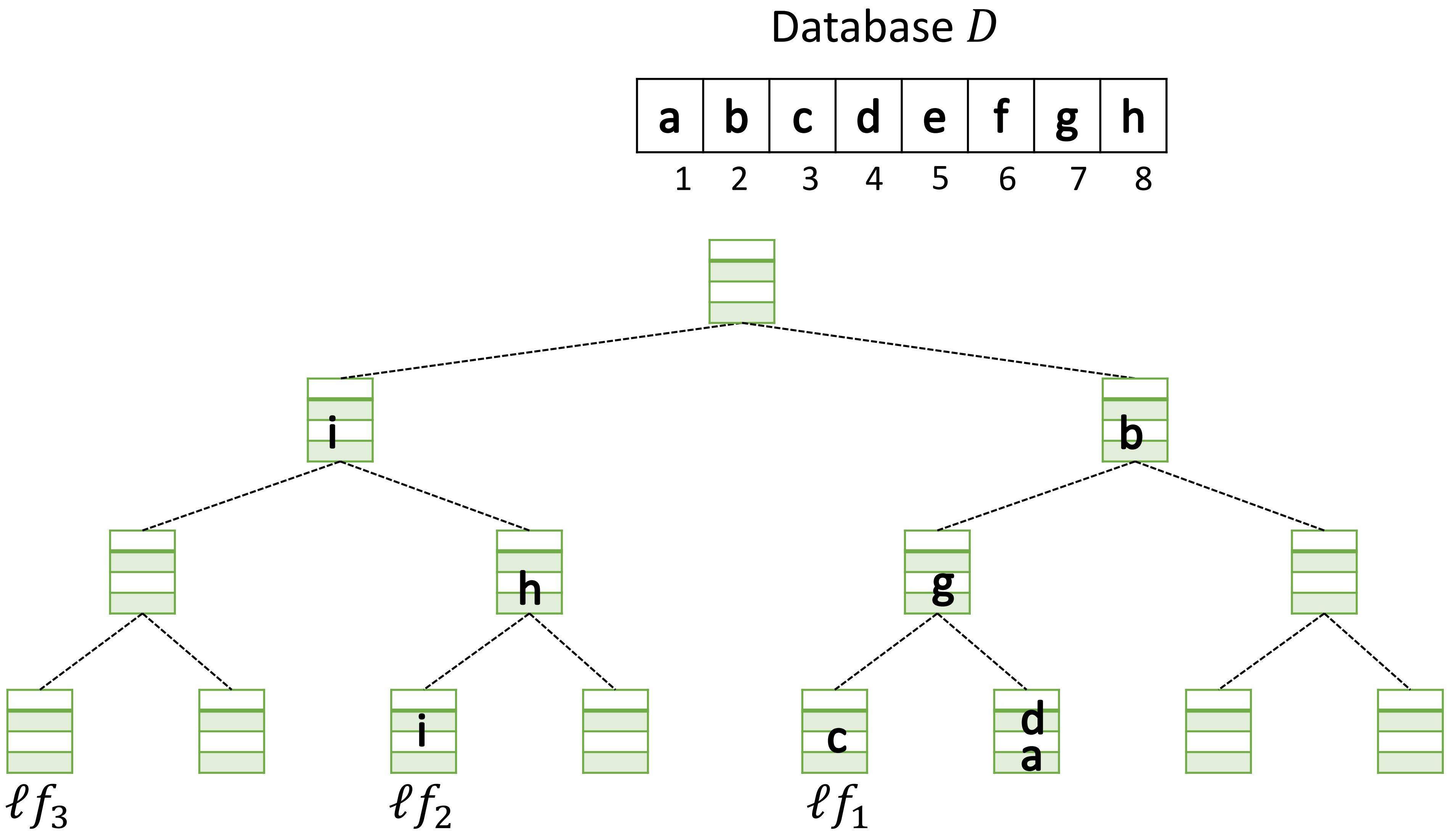


Then, Sim needs to determine consistent randomness (Equivocation)

Sim needs to first generate "fake" oblivious memory accesses.

- For statistical ORAMs, such consistent randomness exists.
  - Can the randomness be extracted efficiently?
- Stronger requirement: the cost to determine consistent randomness should be proportional to the RAM complexity of the function.
  - This algorithm is incorporated within Equivocal Garbled RAM
- Next, we show how to determine randomness for a specific tree-based ORAM.

Quick Review of Tree-Based ORAM [CP13]

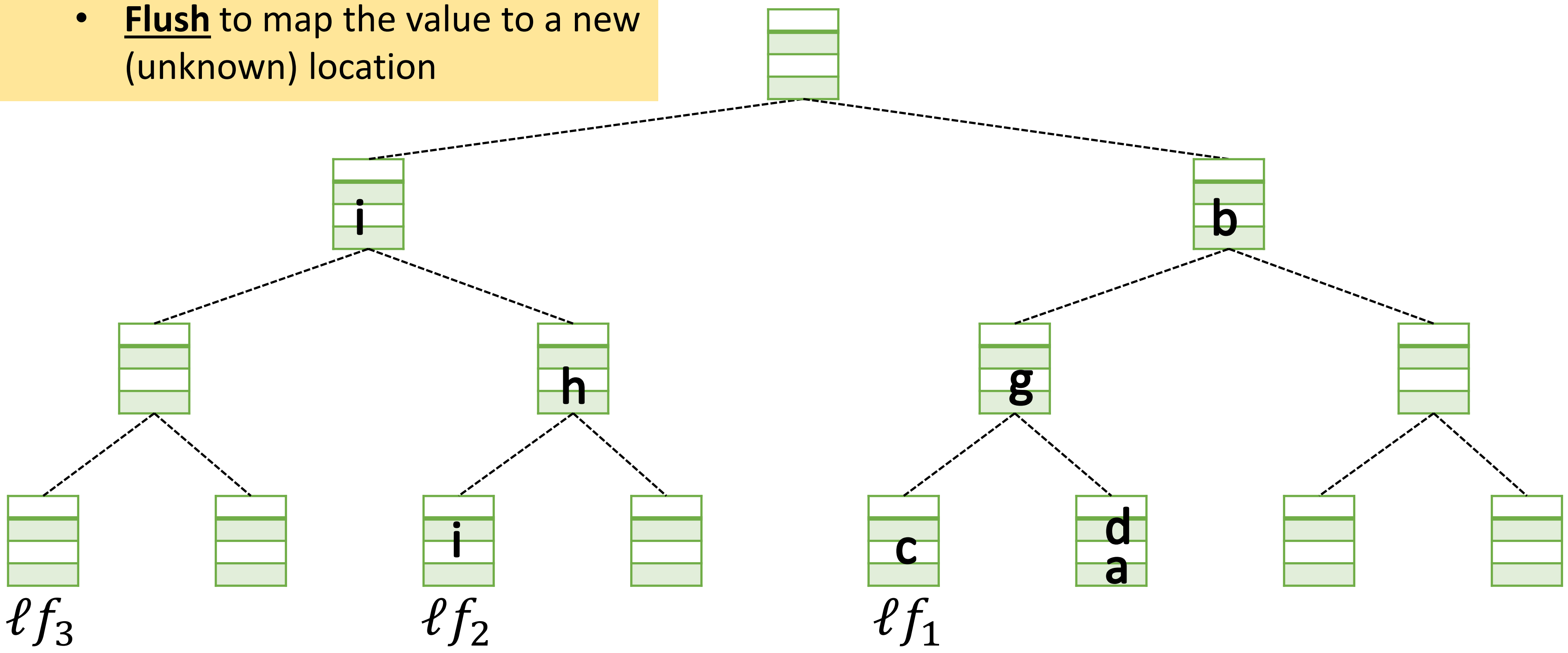


Quick Review of Tree-Based ORAM [CP13]

- Each memory location in D is associated with a leaf node.
- For every read operation, two passes are made from the root to the leaf:
  - **<u>Access</u>** the location to read
  - **<u>Flush</u>** to map the value to a new (unknown) location

Database D

a	b	c	d	e	f	g	h
1	2	3	4	5	6	7	8



Quick Review of Tree-Based ORAM [CP13]

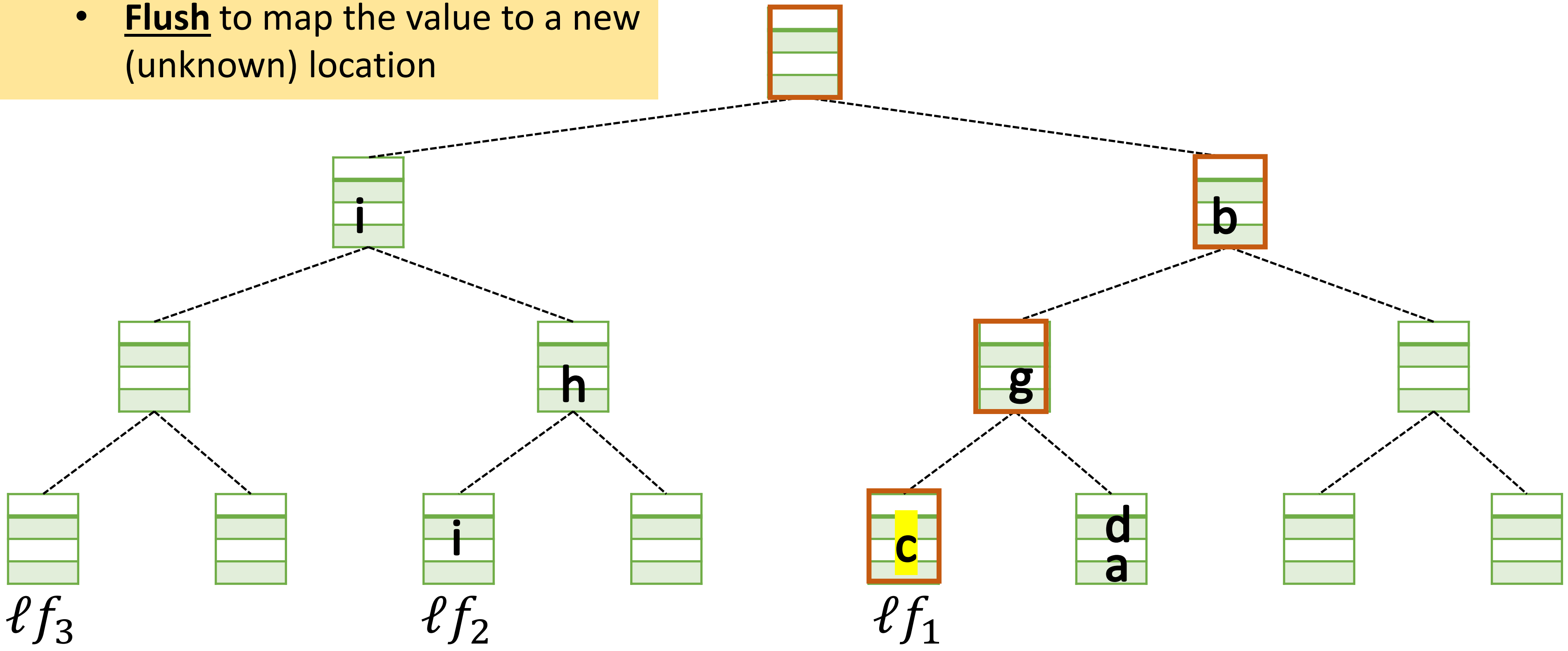
- Each memory location in D is associated with a leaf node.
- For every read operation, two passes are made from the root to the leaf:
  - **Access** the location to read
  - **Flush** to map the value to a new (unknown) location

Database D

a	b	c	d	e	f	g	h
1	2	3	4	5	6	7	8

Instructions

1. Read location 3



Quick Review of Tree-Based ORAM [CP13]

- Each memory location in D is associated with a leaf node.
- For every read operation, two passes are made from the root to the leaf:
  - **Access** the location to read
  - **Flush** to map the value to a new (unknown) location

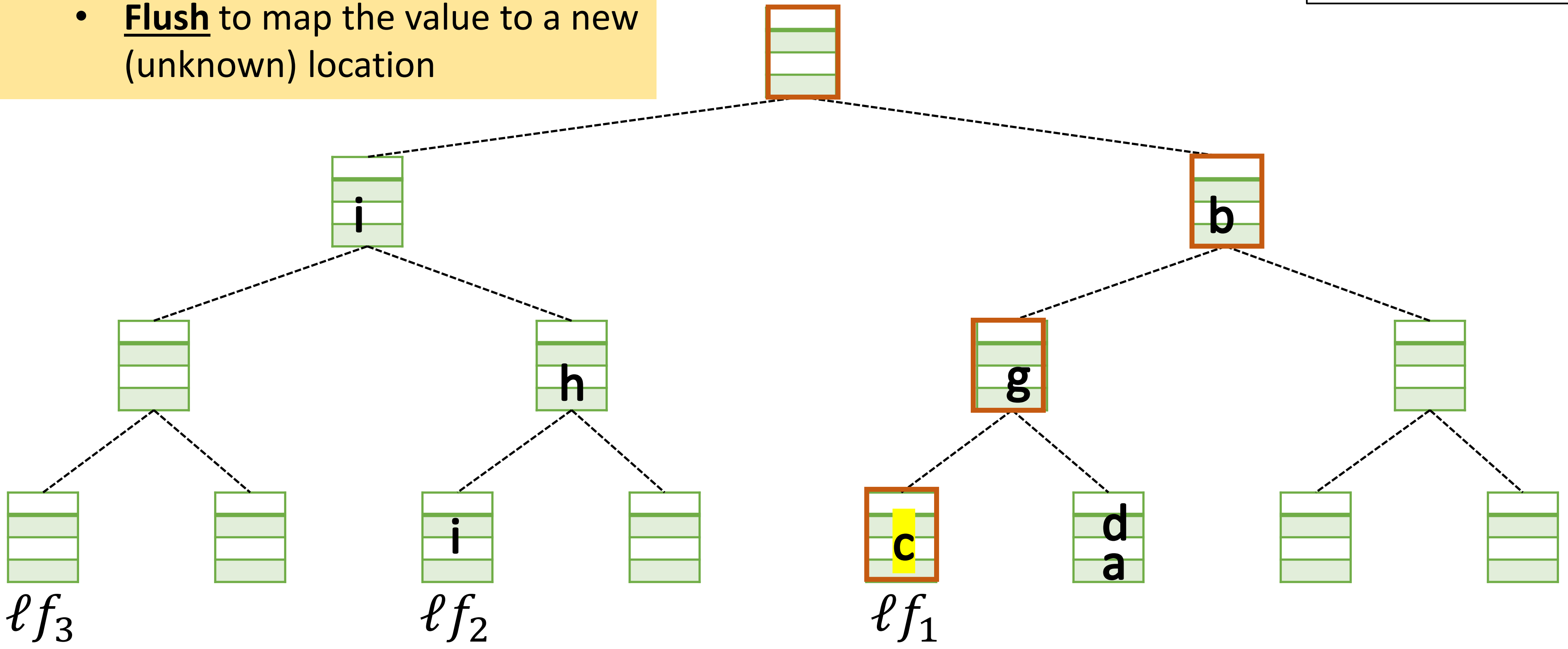
Database D

a	b	c	d	e	f	g	h
1	2	3	4	5	6	7	8

Instructions

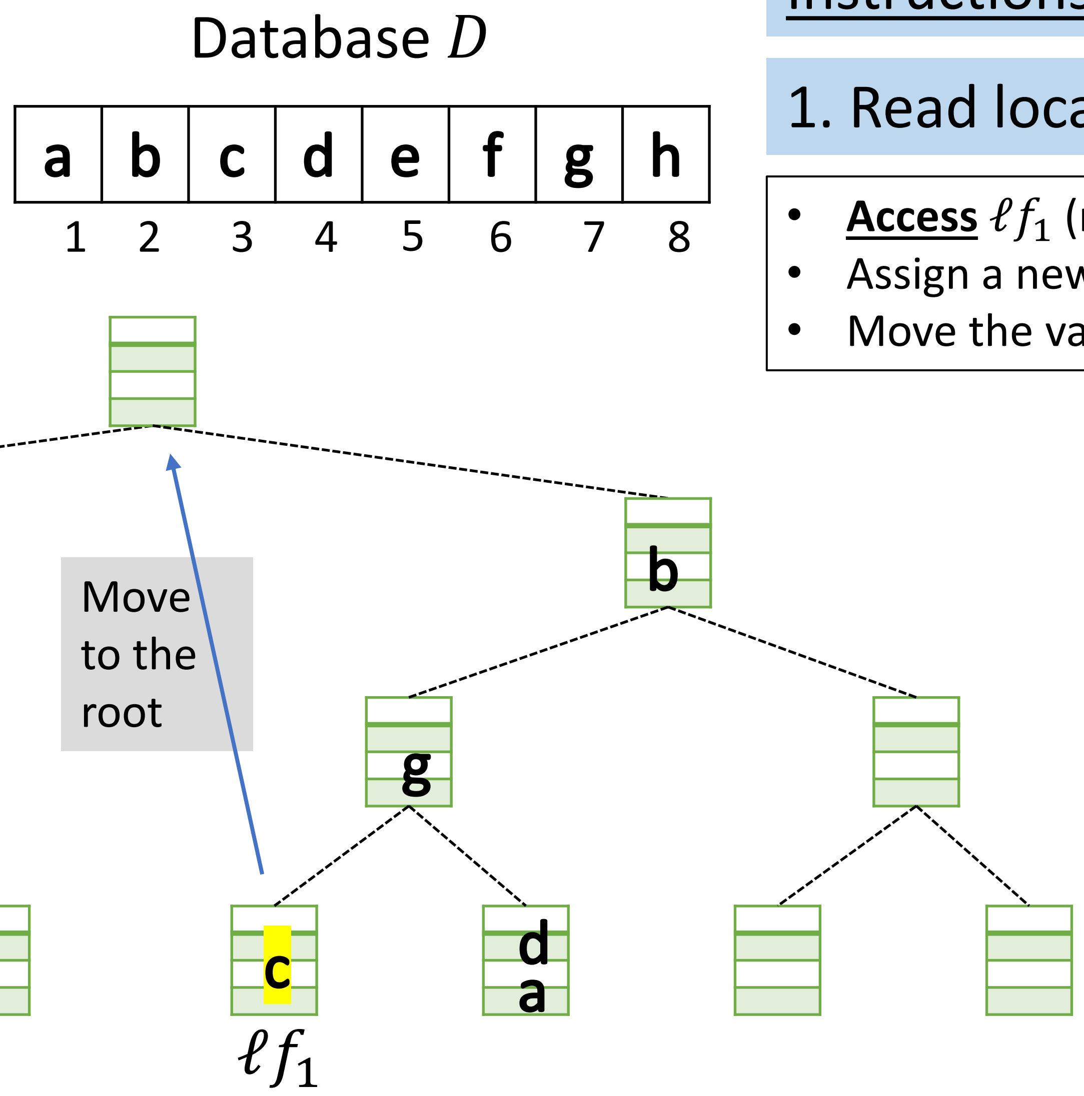
1. Read location 3

- **Access** ℓf_1 (red path)
- Assign a new leaf node ℓf_2 to location 3



Quick Review of Tree-Based ORAM [CP13]

- Each memory location in D is associated with a leaf node.
- For every read operation, two passes are made from the root to the leaf:
  - **Access** the location to read
  - **Flush** to map the value to a new (unknown) location



Instructions

1. Read location 3

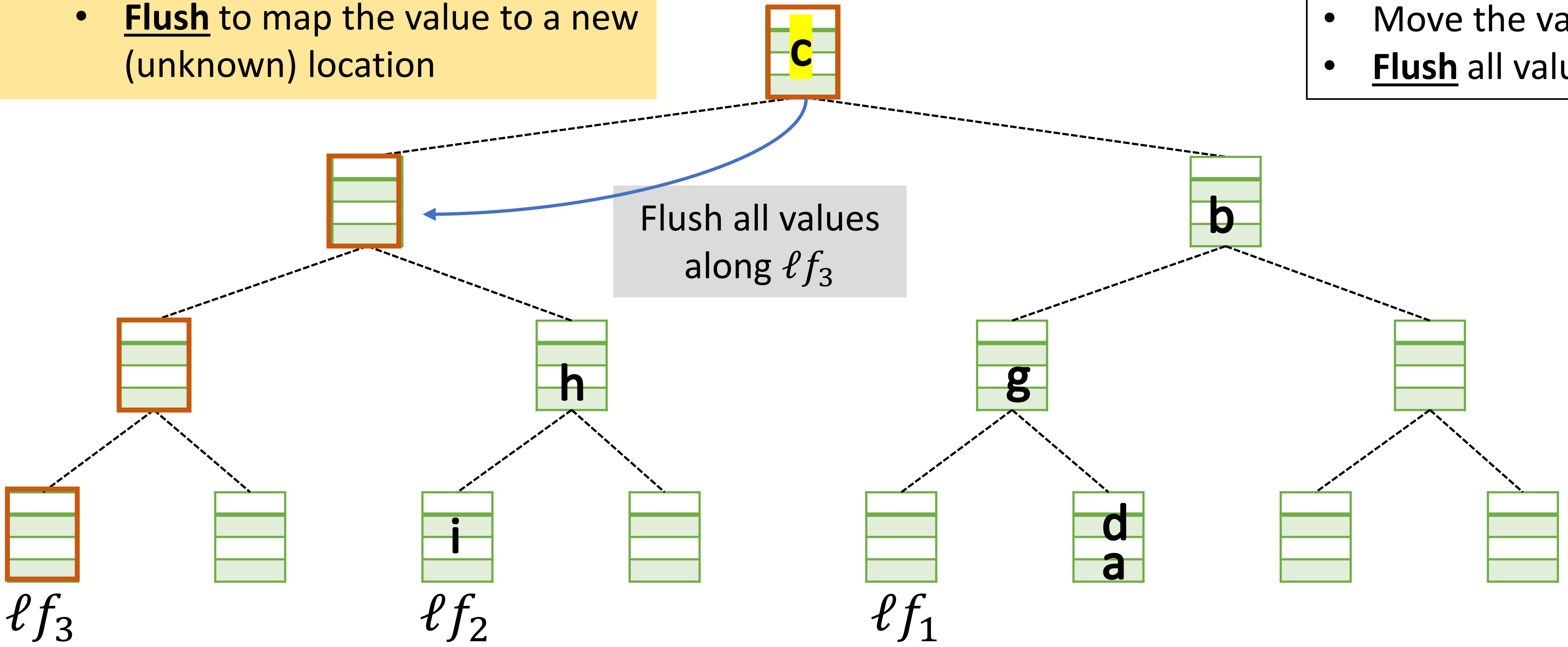
- **Access** ℓf_1 (red path)
- Assign a new leaf node ℓf_2 to location 3
- Move the value read into the root node

Quick Review of Tree-Based ORAM [CP13]

- Each memory location in D is associated with a leaf node.
- For every read operation, two passes are made from the root to the leaf:
  - **Access** the location to read
  - **Flush** to map the value to a new (unknown) location

Database D

a	b	c	d	e	f	g	h
1	2	3	4	5	6	7	8



Instructions

1. Read location 3

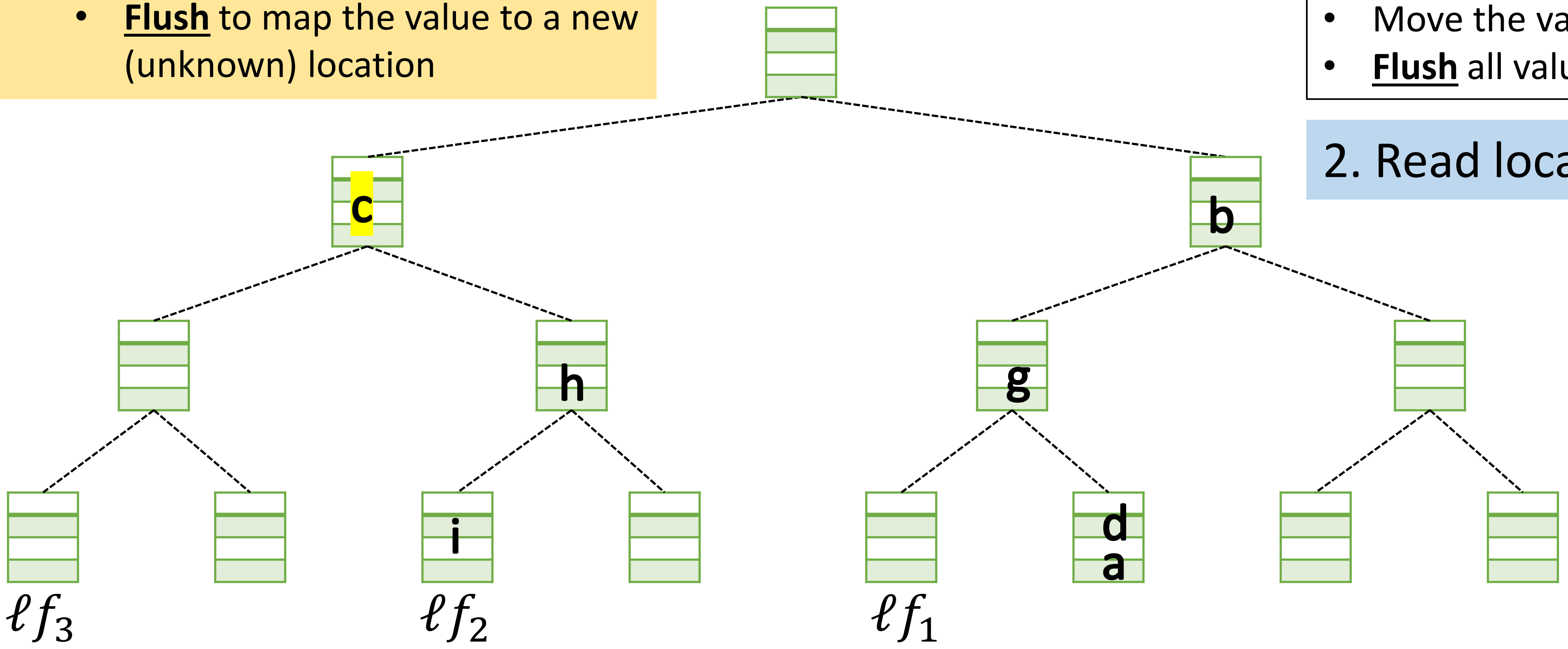
- **Access** ℓf_1 (red path)
- Assign a new leaf node ℓf_2 to location 3
- Move the value read into the root node
- **Flush** all values along with ℓf_3

Quick Review of Tree-Based ORAM [CP13]

- Each memory location in D is associated with a leaf node.
- For every read operation, two passes are made from the root to the leaf:
  - **Access** the location to read
  - **Flush** to map the value to a new (unknown) location

Database D

a	b	c	d	e	f	g	h
1	2	3	4	5	6	7	8



Instructions

1. Read location 3

- **Access** ℓf_1 (red path)
- Assign a new leaf node ℓf_2 to location 3
- Move the value read into the root node
- **Flush** all values along with ℓf_3

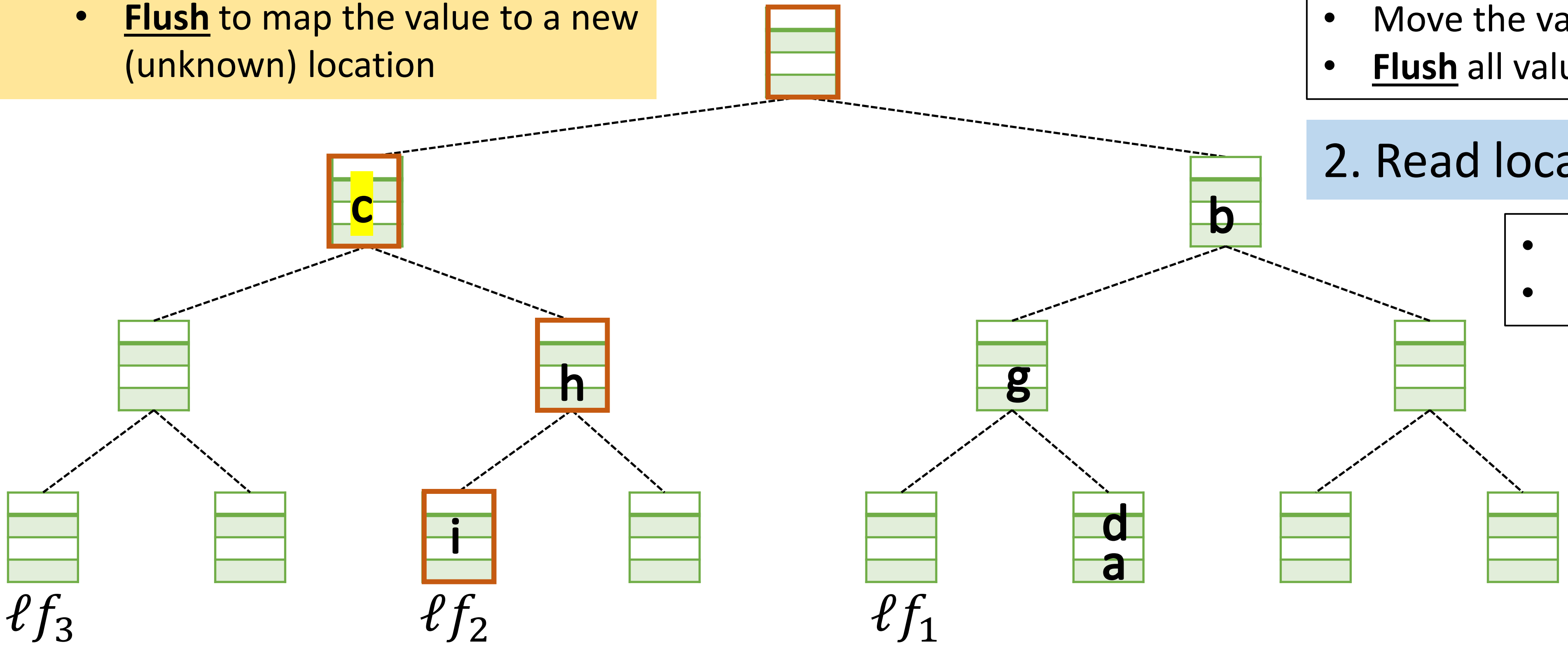
2. Read location 3

Quick Review of Tree-Based ORAM [CP13]

- Each memory location in D is associated with a leaf node.
- For every read operation, two passes are made from the root to the leaf:
  - **Access** the location to read
  - **Flush** to map the value to a new (unknown) location

Database D

a	b	c	d	e	f	g	h
1	2	3	4	5	6	7	8



Instructions

1. Read location 3

- **Access** ℓf_1 (red path)
- Assign a new leaf node ℓf_2 to location 3
- Move the value read into the root node
- **Flush** all values along with ℓf_3

2. Read location 3

- **Access** ℓf_2 (purple path)
- Repeat as above

What does it mean to show ORAM is adaptive?

Actual Memory Accesses M_1

α_1 α_2 α_3 α_4

Simulator must generate the randomness

Oblivious Memory Accesses M_2

ℓf_1 ℓf_3 ℓf_1 ℓf_3 ℓf_1 ℓf_3 ℓf_1 ℓf_3

- SimORAM samples $2m$ leaf nodes randomly as the oblivious leaf nodes.
- Generating consistent randomness for each memory access corresponds to the new leaf node assigned to a memory location after it is read.
- Essentially, the randomness corresponds to leaf nodes $\{\ell f_2\}_{i \in [m]}$
- Suppose $\alpha_1 = \alpha_2$, then $\ell f_2 = \ell f_1$
- Efficiency: $m \cdot polylog(m)$

Addressing Challenge II

Recall that:

- [CPV17] is designed for stand-alone circuits.
- It does not handle external memory accesses.
- Quick Review of Equivocal Garbling of [CPV17]

Overview: How to Garble Circuits?

Yao's 2PC Protocol

↑

Garbled Circuits

↑

CPA-secure Encryption

Overview: How to Adaptively Garble Circuits?

CPV17

Yao's 2PC Protocol	Adaptive 2PC for Circuits
↑	↑
Garbled Circuits	Equivocal Garbled Circuits
↑	↑
CPA-secure Encryption	Circuit-efficient Equivocal Encryption (CEE)

Overview: How to Adaptively Garble RAM Programs?

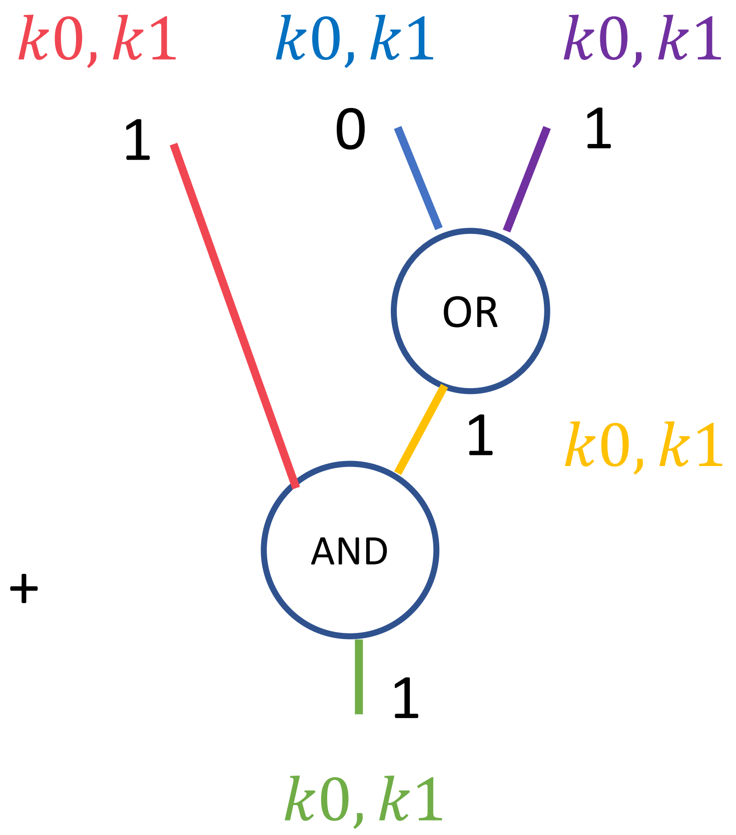
	CPV17	Our Work
Yao's 2PC Protocol	Adaptive 2PC for Circuits	Adaptive 2PC for RAM
↑	↑	↑
Garbled Circuits	Equivocal Garbled Circuits	Equivocal Garbled RAM
↑	↑	↑
CPA-secure Encryption	Circuit-efficient Equivocal Encryption (CEE)	RAM-efficient Equivocal Encryption (REE) + Equivocal ORAM

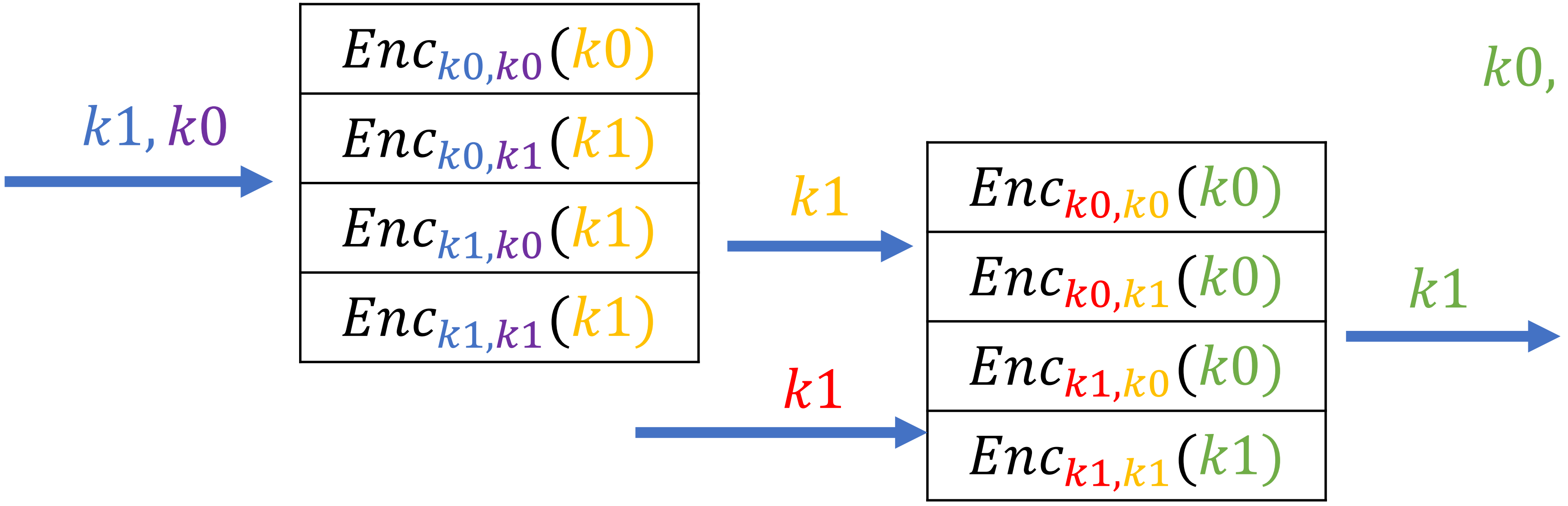
Yao's Garbling Scheme

Key Generation: Pick two keys per wire.

Garble Input(x): $101 \rightarrow k1, k0, k1$

Garble Circuit(C): Mechanism to evaluate the circuit +
Output translation table

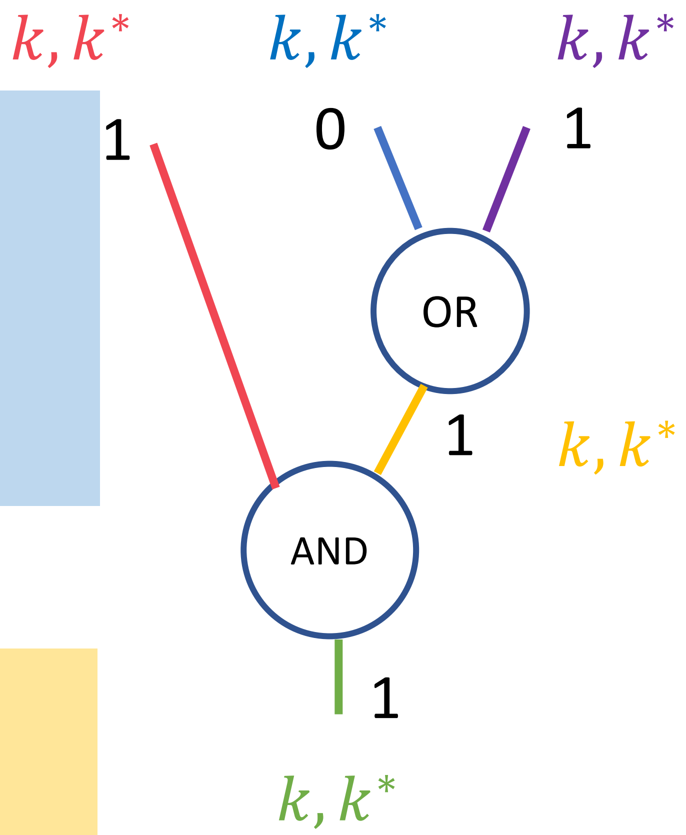




Yao's Garbling: Static Security

Simulation:

- Pick an active key for each wire
- 1 ciphertext encrypts the active key
- Other 3 ciphertexts are simulated
- Set output table to match the output $C(x)$



$??? \rightarrow k, k, k$

Given input x, show the consistent randomness generation

- Inactive keys
- Randomness for encryption

We have:

Simulated
$Enc_{k,k}(k)$
Simulated
Simulated

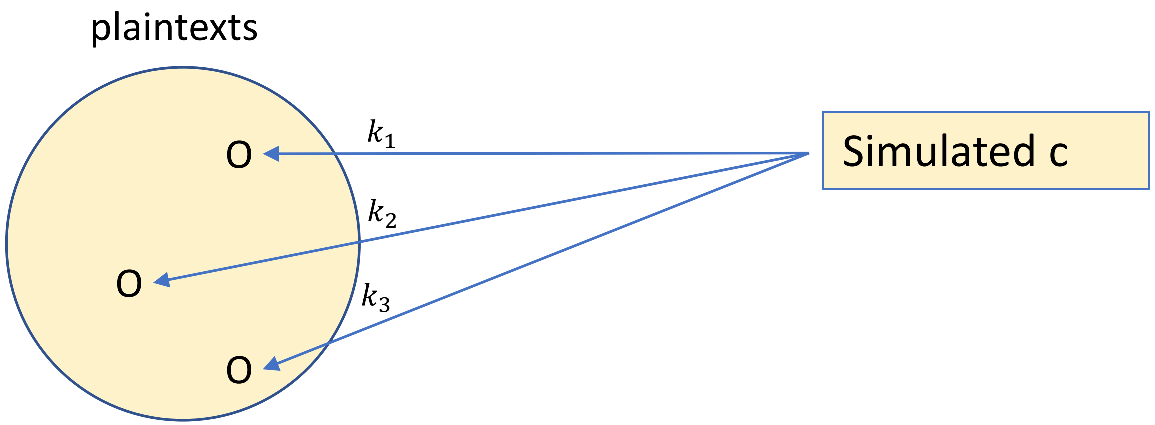
We need:

$Enc_{k^*,k}(k)$
$Enc_{k,k}(k)$
$Enc_{k,k^*}(k^*)$
$Enc_{k^*,k^*}(k)$

Which key should be encrypted is determined by the wire values of circuit C.

Non-Committing Encryption

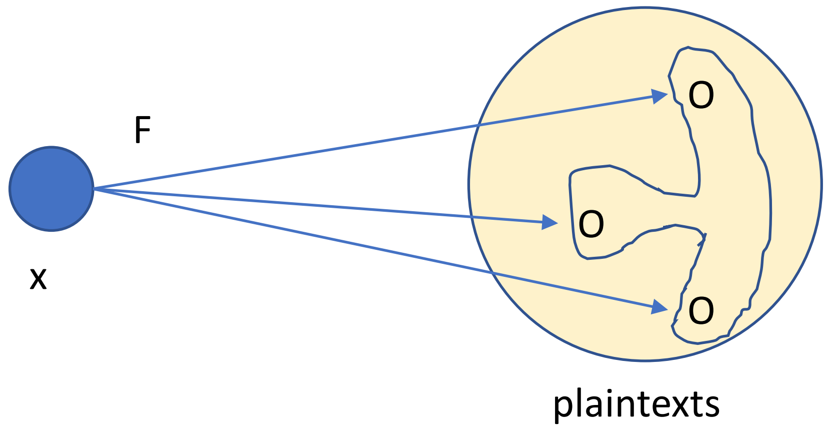
- Honestly generated cipertexts: standard correctness and security
- Simulated cipertexts can be “opened” to any plain text m_i:
  - Sim can generate k_i such that $c = Enc(k_i : m_i)$



too many options to open → too large k → Exp. Growth of keys

Circuit-Efficient Equivocal Encryption (Def.)

- Simulated cipertexts can be “opened” to **some (but exp many)** plaintexts:
  - Sim can generate k_i such that $c = Enc(k_i, m_i)$
- Only plaintexts in the image space of a function F can be equivocated.



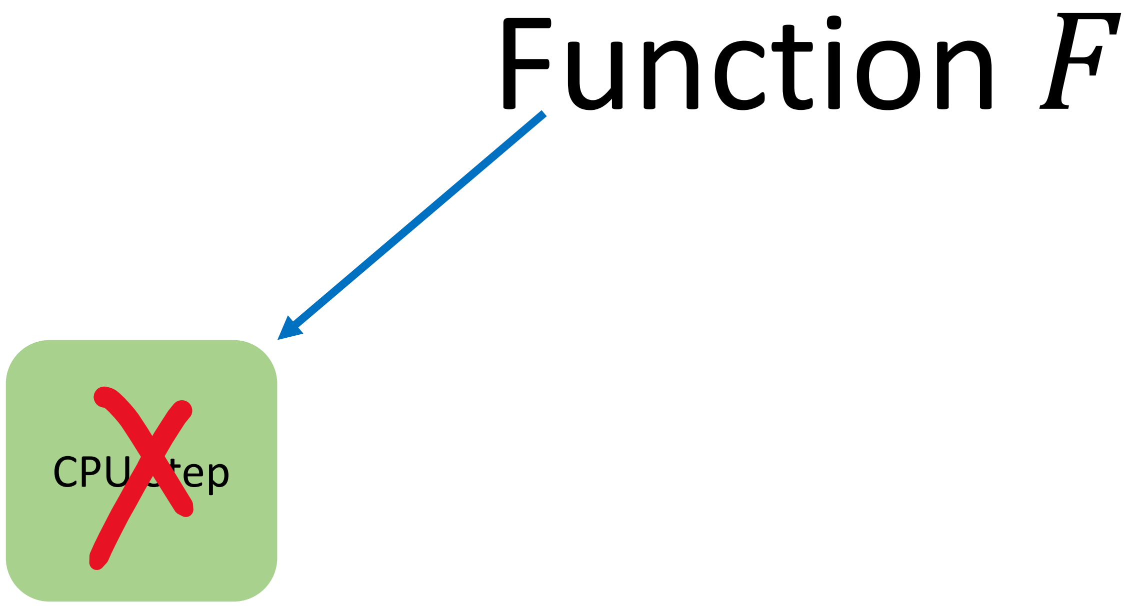
CEE Property:

$$k \leftarrow Equivocate(x)$$

$Dec(k; c) = F(x)$ for simulated c

- [CPV17] F is expressed as a circuit.
- Next, we will see how to instantiate F.

Function For Equivocal Encryption



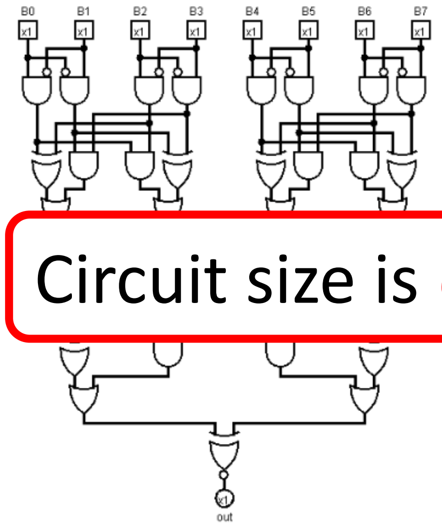
<u>But the step circuits are dependent and take additional inputs other than x.</u>

- Given just input x, it is not sufficient to compute wire values in any step circuit.
  - Require state and memory values to evaluate the wire values in intermediate step circuits
- Solution: So, we could convert the RAM to a circuit and then use this within Enc.

Function For Equivocation Encryption

Function F

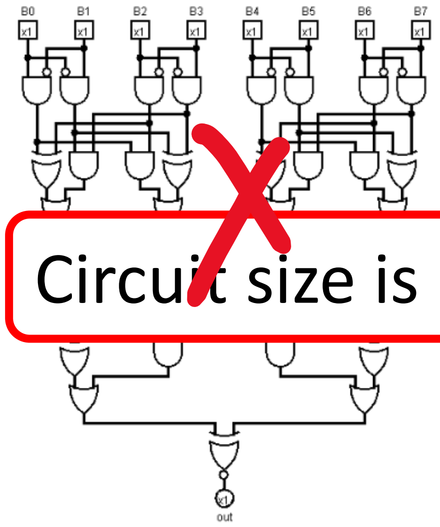
CPU step



Circuit size is $\tilde{O}(T^3)$!

Function For Equivocation Encryption

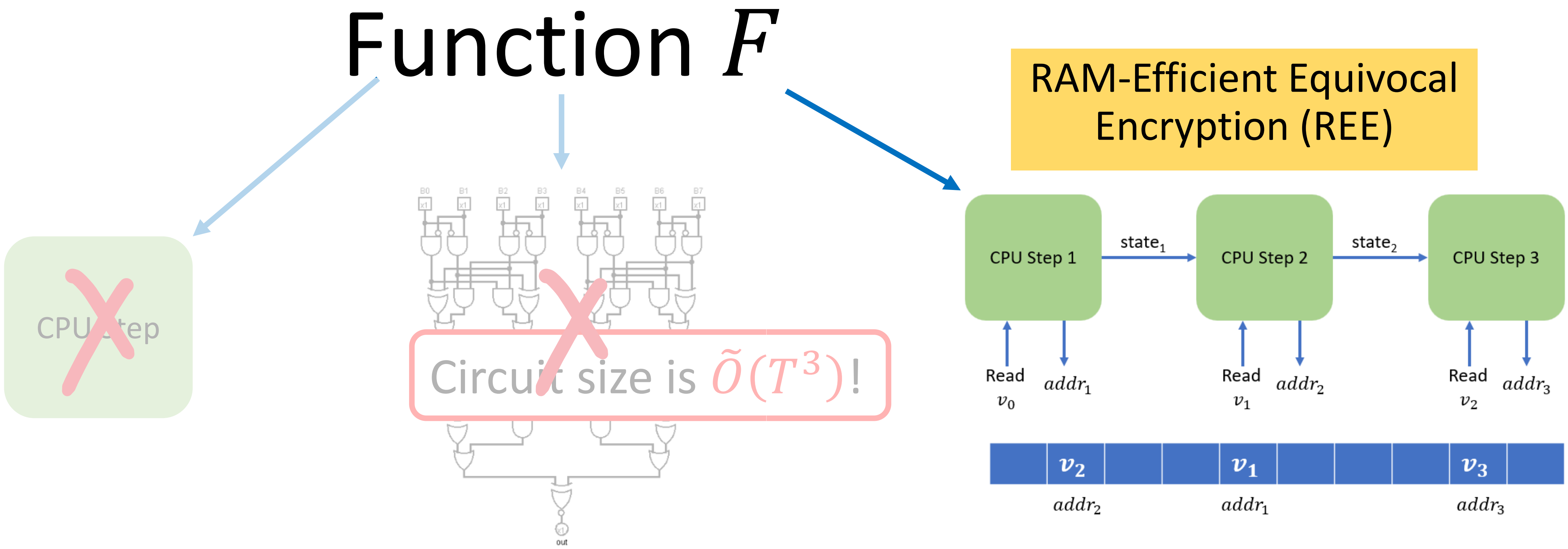
Function F



Circuit size is $\tilde{O}(T^3)$!

- Each ciphertext is of size $\tilde{O}(T^3)$
- There are $T \cdot polylog(T)$ such ciphertexts in the entire garbled RAM
- So, the communication is $\tilde{O}(T^4)$!

Function For Equivocation Encryption



- Each ciphertext is of size $\tilde{O}(T)$
- There are $T \cdot polylog(T)$ such ciphertexts in the entire garbled RAM
- So, the communication is $\tilde{O}(T^2)$

Other Challenges…

- Most Garbled RAM works are non-black-box in PRFs
  - Non-trivial to equivocate!
- However, [GLO15] fits well into our framework
  - Black-box use of underlying primitives
  - Memory is expressed as a tree of circuits
- Malicious security
  - Construct RAM-efficient adaptively-secure Zero-knowledge proofs
  - Previously based on indistinguishability obfuscation [GP15, CPV17].
  - Then apply standard transformation (GMW compiler)

Future Directions

For fully adaptive constant-round protocols, the communication is

- [CPV17] Quadratic in the circuit complexity of a function
- Our result: Quadratic in the RAM complexity of a function

Is the quadratic communication cost in the circuit/RAM complexity inherent in this regime?

THANK YOU!