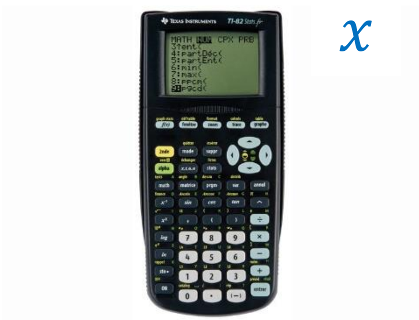
Garbling and Outsourcing Private RAM Computation

Daniel Wichs
Northeastern University

Based on :

- Garbled RAM, Revisited [Gentry-Halevi-Lu-Ostrovsky-Raykova-W]
- Outsourcing Private RAM Computation [Gentry-Halevi-Raykova-W]

Problem Overview



x

$y = P(x)$

Client

Server

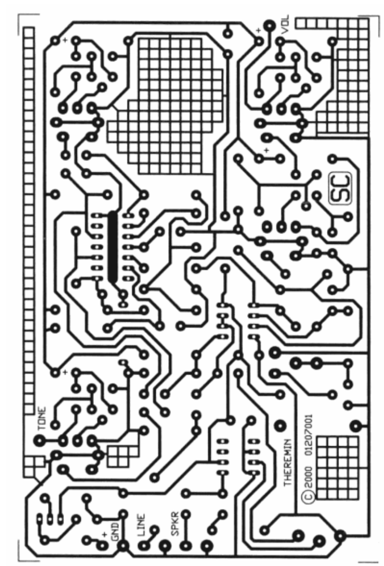
- Weak client wants to leverage resources of a powerful server to compute $P(x)$ without revealing x.

- Efficiency Requirements:
  - Client does much less work than computing $P(x)$
  - Server does about as much work as computing $P(x)$

Circuits vs. RAM

- Private outsourcing is possible using Fully Homomorphic Encryption (FHE). [RAD78,Gen09,…]

- But FHE works over *circuits* rather than *RAM programs*.

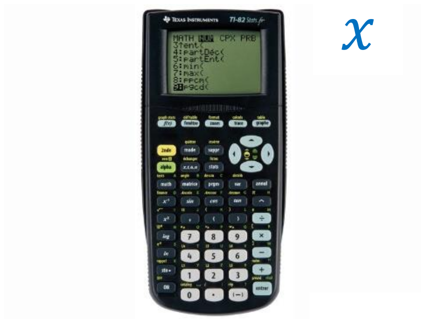


Circuits vs. RAM

- Private outsourcing is possible using Fully Homomorphic Encryption (FHE). [RAD78,Gen09,…]

- But FHE works over *circuits* rather than *RAM programs*.
  - RAM complexity T $\Rightarrow$ circuit or TM complexity T^2
  - For programs with initial "data in memory", efficiency gap can be exponential (e.g., Google search).

Goals



Client

x →

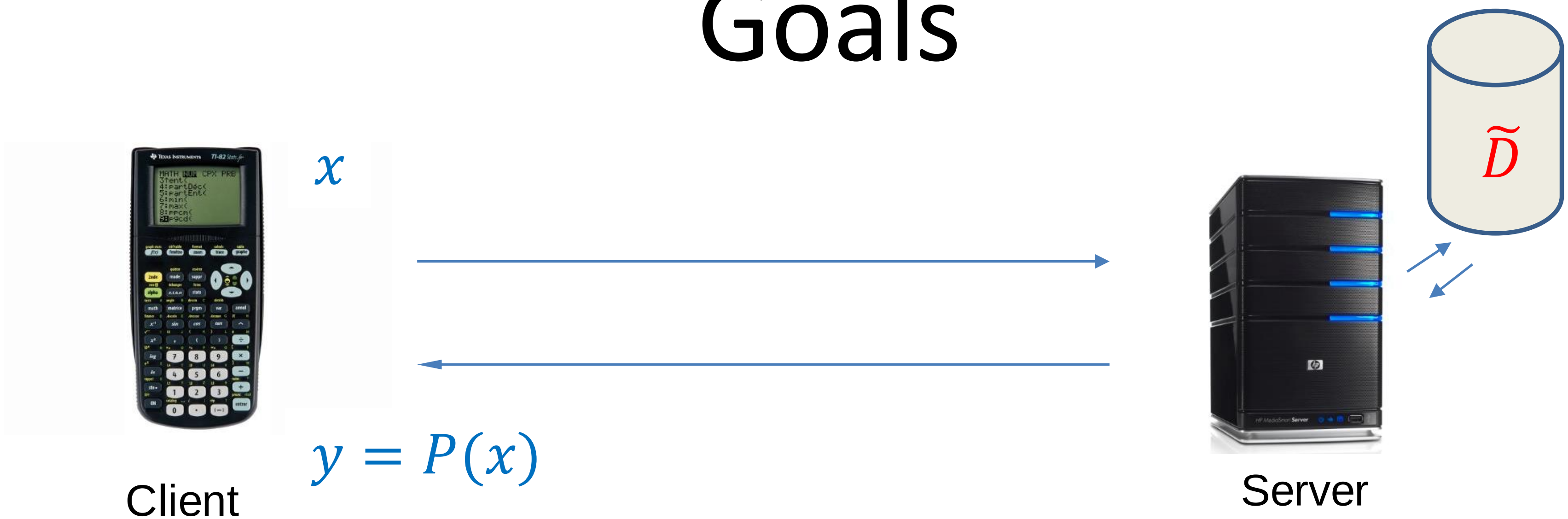
← $y = P(x)$

Server

- Client's work: $O(\,|x| + |y|)$
- Server's work: O(RAM run-time of P).

- May allow client pre-processing of P.
  - Client does one-time computation in O(RAM run-time of P).
  - Later, outsource many executions of P. Amortized efficiency.

Goals



- Basic scenario: client wants to run independent executions of P on inputs $x_1, x_2, x_3, \ldots$

- Persistent Memory Data:
  - Client initially outsources large private ‘memory data’ D.
  - Program executions $P^D(x_i)$ can read/write to D.
  - Generalizes oblivious RAM.

Goals

Client

Server

- Non-interactive solution: “reusable garbled RAM”.

Garbled Computation

Garbled Circuits

[Yao82]

Garble circuit: $C \rightarrow \tilde{C}$

Garble input: $x \rightarrow \tilde{x}$

Given $\tilde{C}, \tilde{x}$ only reveals $C(x)$

Secure on one input x.

Reusable Garbled Circuits

[GKPVZ 13a,b]

Can garble many inputs per circuit.

Efficiently outsource circuit comp.

Extension to TM.

Garbled RAM

[LO13, GHLORW14]

Garble RAM: $P \rightarrow \tilde{P}$

Garble input: $x \rightarrow \tilde{x}$

Size of $\tilde{P}$, run-time $\tilde{P}(\tilde{x})$ is O(RAM run-time P).

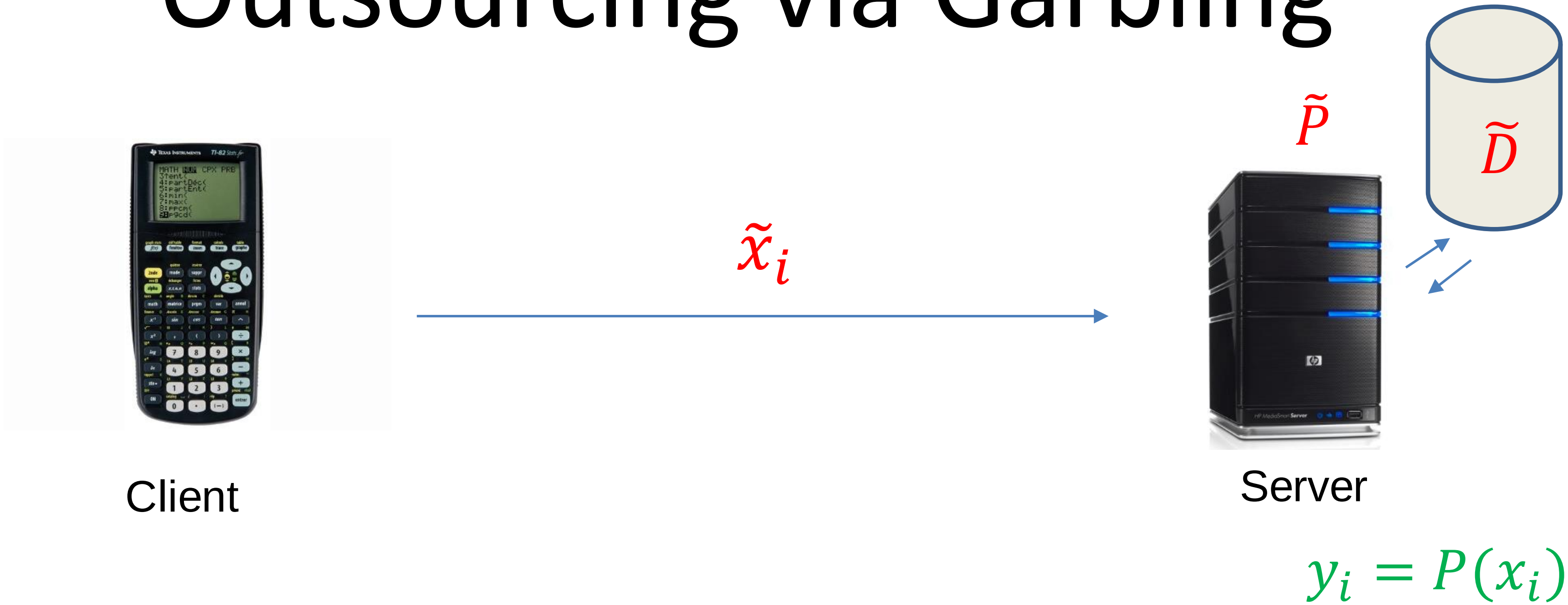
Reusable Garbled RAM

[GHRW14]

Can garble many inputs per program.

Efficiently outsource RAM comp.

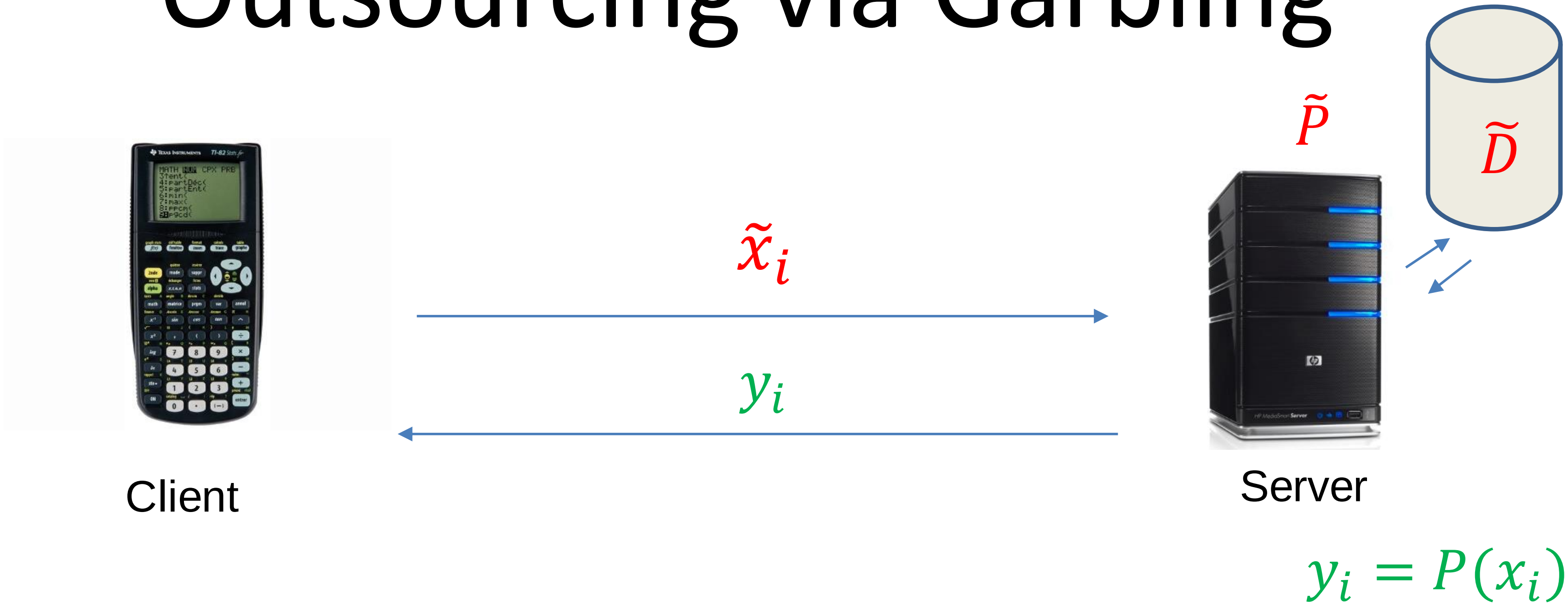
Outsourcing via Garbling



- Client garbles program $P \rightarrow \tilde{P}$ [data $D \rightarrow \tilde{D}$].
  - Pre-processing = O(run-time P)

- Client repeatedly garbles inputs $x_i \rightarrow \tilde{x}_i$ in time $O(|x_i|)$.
- Server evaluates $\tilde{P}$ on $\tilde{x}_i$ to get y_i. [using $\tilde{D}$]
  - Evaluation time = O(run-time P)

Outsourcing via Garbling



- *Output privacy:* set y_i = encryption of real output. Server sends back y_i.
- *Verifiability:* y_i includes (one-time) MAC of real output.
- *Program Privacy:*
  - P is universal RAM, code is given as part of input.
  - P has hard-coded encryption of code. x includes decryption key.

Garbled RAM

Garbled RAM
[LO13, GHLORW14]

PART I

- Overview of [LO13].
- Circularity issue, and fix.

Reusable Garbled RAM
[GHRW14]

PART II

Combine:
- Non-reusable garbled RAM.
- New type of reusable garbled circuits.

- Constructions based on obfuscation.

PART I

One-Time Garbled RAM

Garbled RAM Syntax

- **GData**$(D) \rightarrow \widetilde{D},\ k_{data}$ garble data
- **GProg**$(P) \rightarrow \tilde{P},\ k_{prog}$ garble program
- **GInput**$(x, k_{prog}) \rightarrow \tilde{x}$ garble input
- **Eval**$^{\widetilde{D}}(\tilde{P}, \tilde{x}) \rightarrow y$ evaluate program

One-Time Garbled RAM

- Basic Security: Can simulate $(\tilde{P}, \tilde{x})$ given y.

- Persistent data: Can reuse garbled data, but not garbled programs.

  Simulate $(\widetilde{D}, \quad (\tilde{P}_1, \tilde{x}_1), \ (\tilde{P}_2, \tilde{x}_2), \ \ldots)$

  Given $y_1, y_2, \ldots$

  – Note: changes to data persist, order matters.

One-Time Garbled RAM

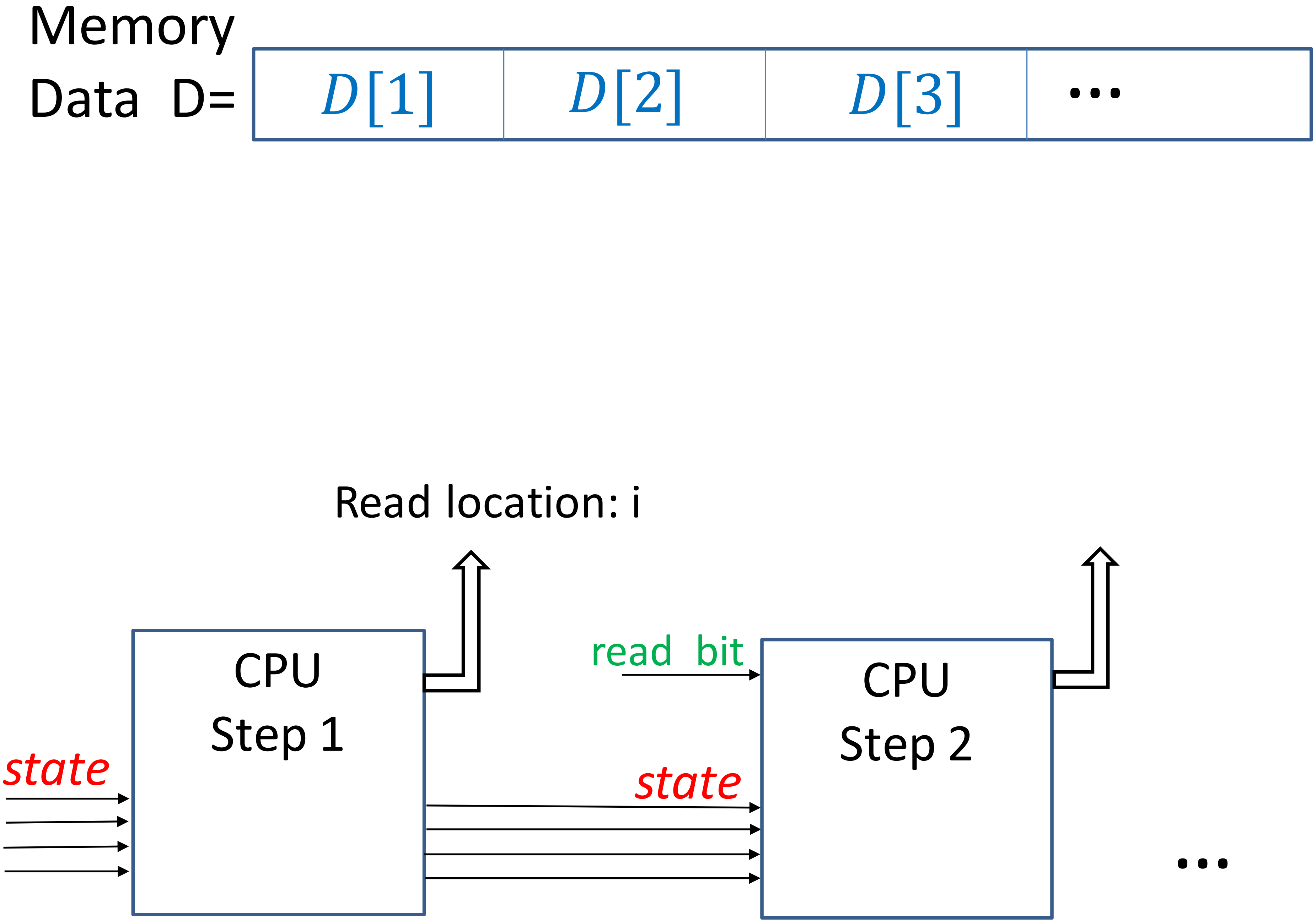
- *Unprotected memory access:* may also reveal D, and the *access pattern* of $P^D(x)$.
  - Locations of memory accessed in each step.
  - Values read and written to memory.

- Compiler: unprotected $\Rightarrow$ full security:
  - Use oblivious RAM [GO96,…] to access memory.

Overview of [Lu-Ostrovsky 13]

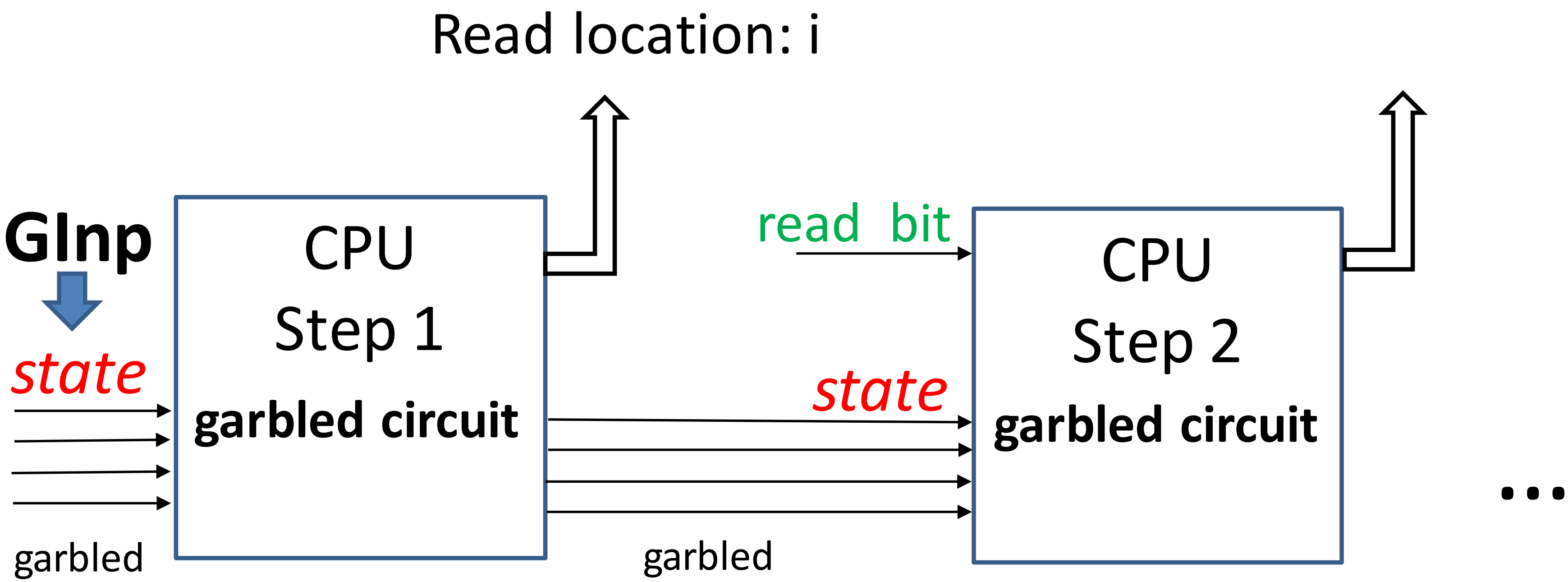
As a first step:

- read-only computation
- unprotected memory access


Memory
Data D=
$D[1]$
$D[2]$
$D[3]$
…
Read location: i
CPU
Step 1
state
read bit
CPU
Step 2
state
…


Memory
Data D=
$D[1]$
$D[2]$
$D[3]$
...

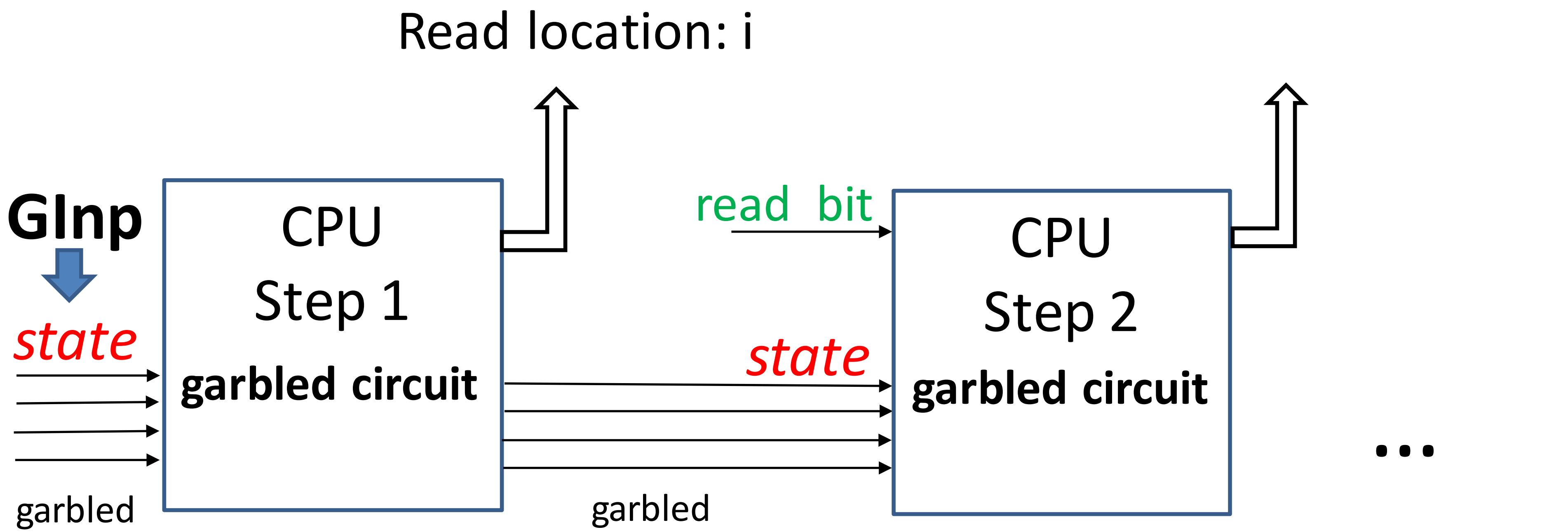
GProg:

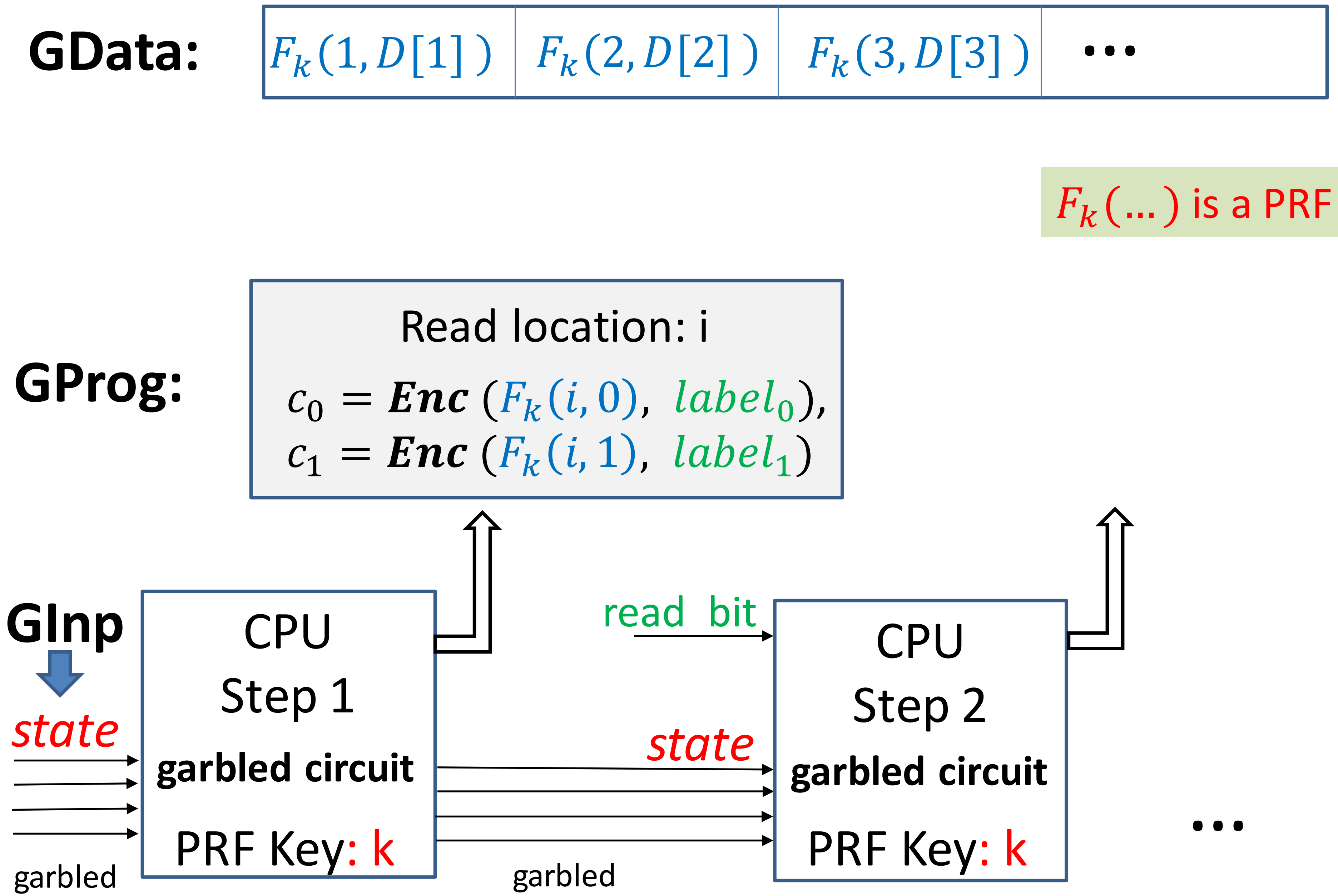

Read location: i
GInp
state
garbled
CPU
Step 1
garbled circuit
read bit
state
garbled
CPU
Step 2
garbled circuit
...

GData: $F_k(1, D[1])$ | $F_k(2, D[2])$ | $F_k(3, D[3])$ | …

$F_k(\ldots)$ is a PRF

GProg:





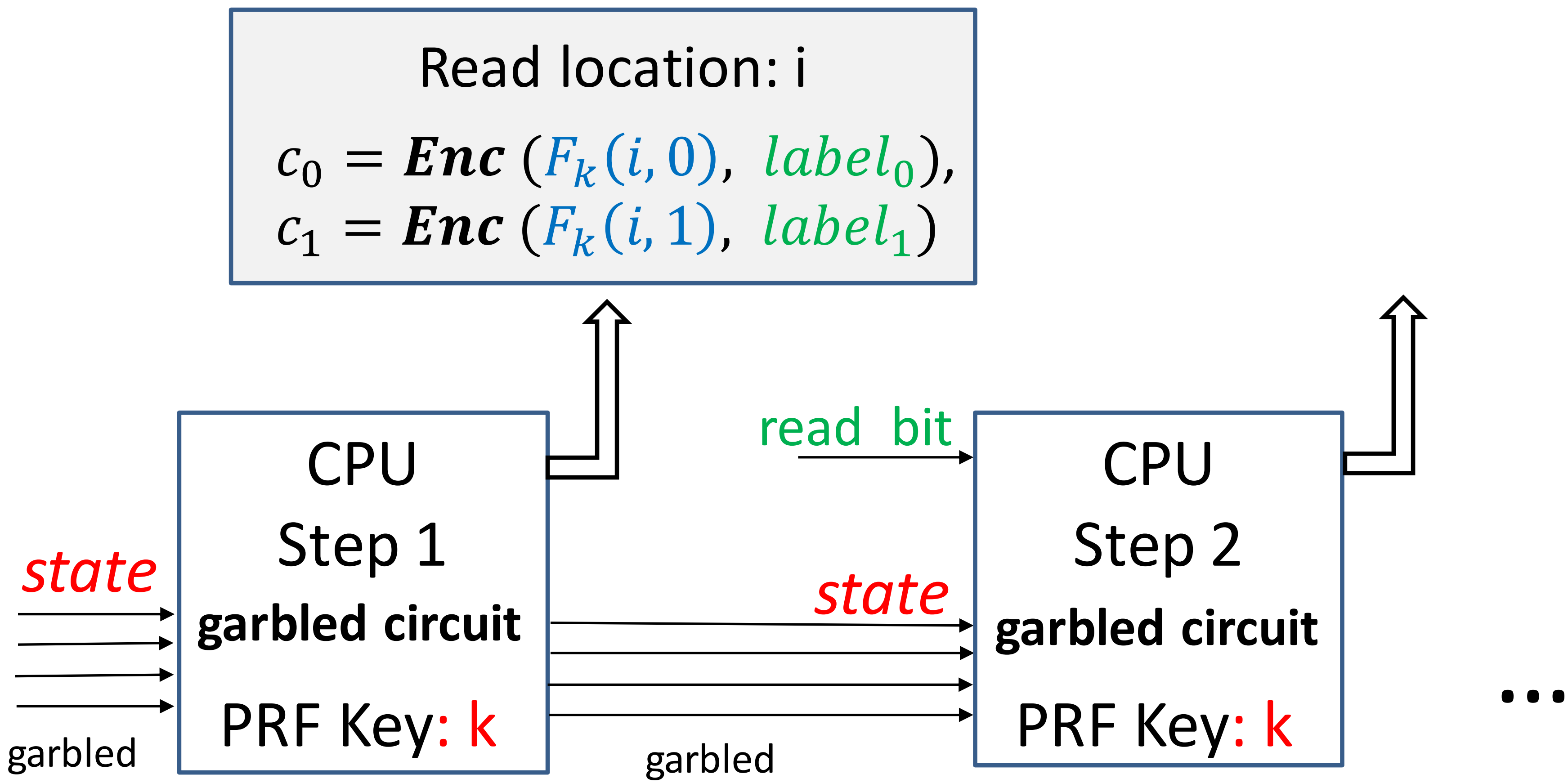
GData:
$F_k(1, D[1])$
$F_k(2, D[2])$
$F_k(3, D[3])$
...
$F_k(\ldots)$ is a PRF
GProg:
Read location: i
$c_0 = \boldsymbol{Enc}\,(F_k(i, 0),\ label_0),$
$c_1 = \boldsymbol{Enc}\,(F_k(i, 1),\ label_1)$
GInp
state
garbled
CPU
Step 1
garbled circuit
PRF Key: k
read bit
state
garbled
CPU
Step 2
garbled circuit
PRF Key: k
...

Let’s try to prove security…

Should rely on:

1. Security of garbled circuits
2. Security of PRF/Encryption.

Use security of 1st garbled circuit…



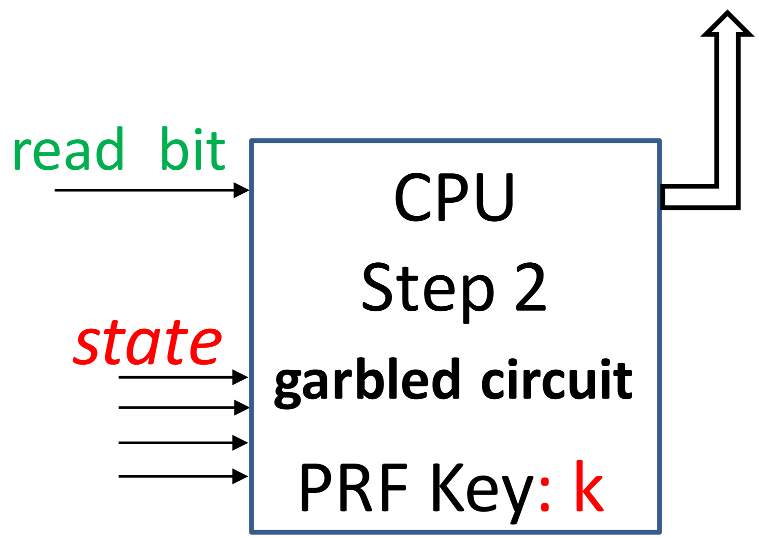
Use security of 1st garbled circuit

only learn output

Read location: i

$c_0 = \boldsymbol{Enc}\,(F_k(i,0),\ label_0),$
$c_1 = \boldsymbol{Enc}\,(F_k(i,1),\ label_1)$

labels for *state*



…

Use security of 1^{st} garbled circuit

only learn output

Assume D[i]=0

Read location: i

$label_0$

$c_1 = \boldsymbol{Enc}\,(F_k(i,1),\ label_1)$

labels for $state$

read bit

$state$

CPU

Step 2

garbled circuit

PRF Key: k

…

Use security of 2nd garbled circuit

don't learn
$label_1$ for read bit

don't learn
PRF key k

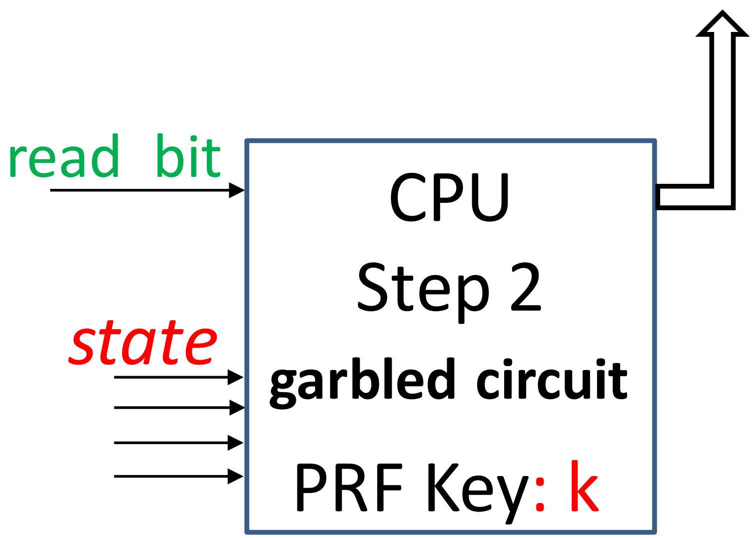
Use security of Encryption/PRF

Read location: i

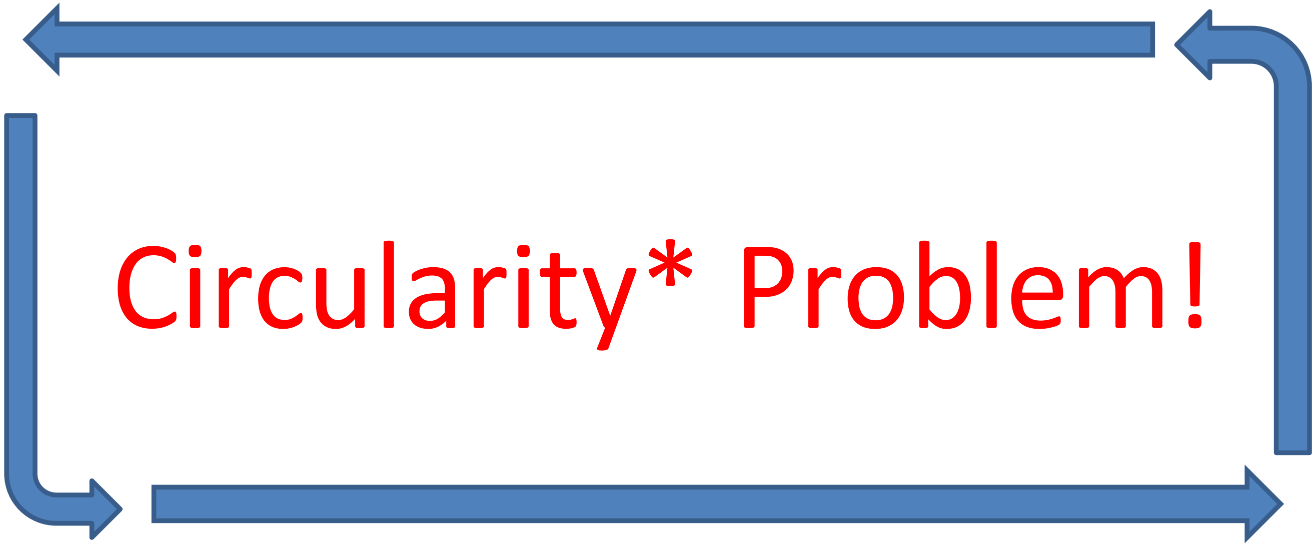
$label_0$

$c_1 = \boldsymbol{Enc}\,(F_k(i, 1),\ \ label_1)$

labels for $state$



...



* May appear rectangular

So is it secure?

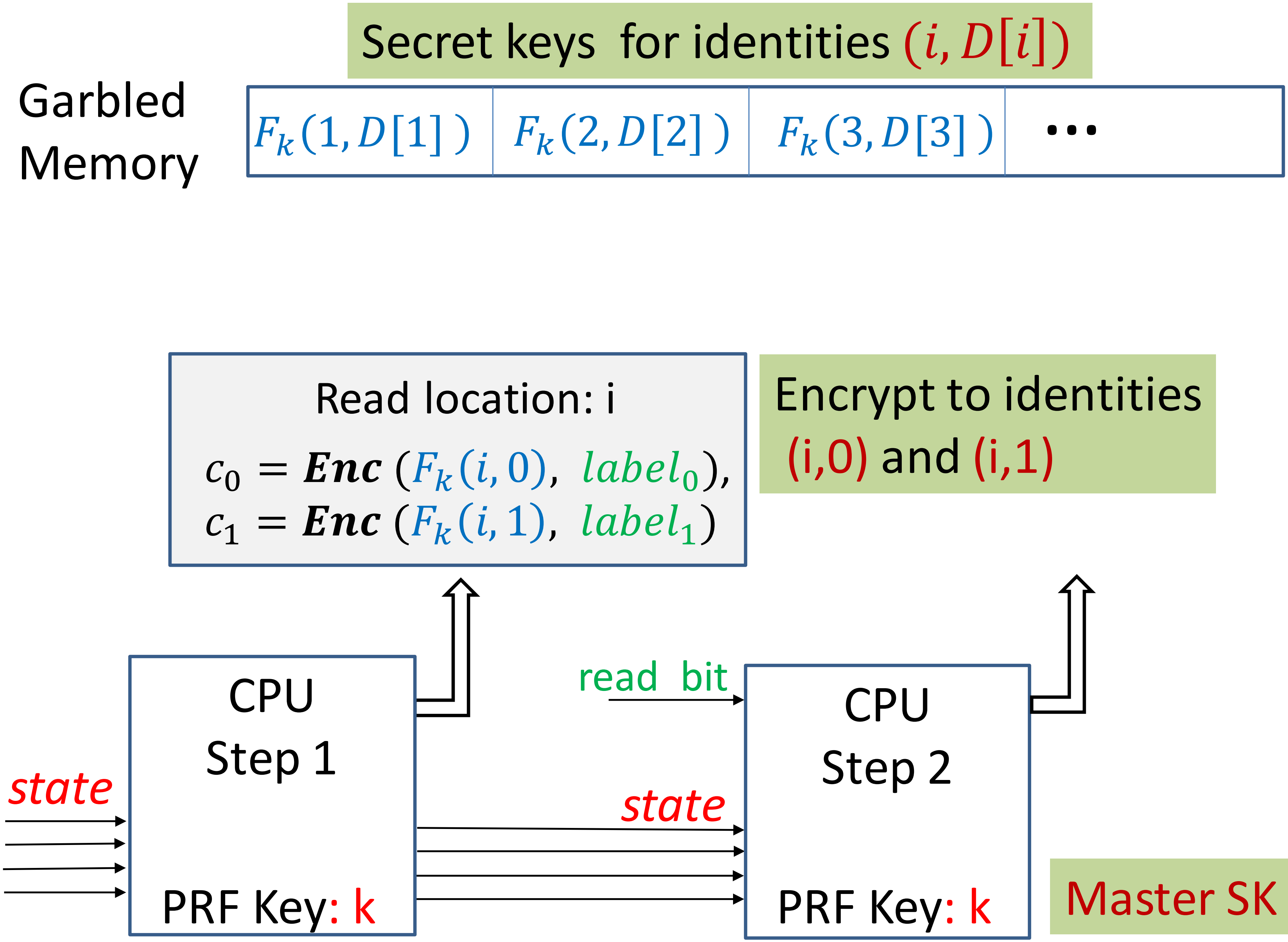
- Perhaps secure if instantiated with a "good" encryption, PRF, circuit garbling.
  - No proof.
  - No "simple" circularity assumption on one primitive.

Can we fix it? Yes! [Gentry-Halevi-Raykova- Lu-Ostrovsky-W]

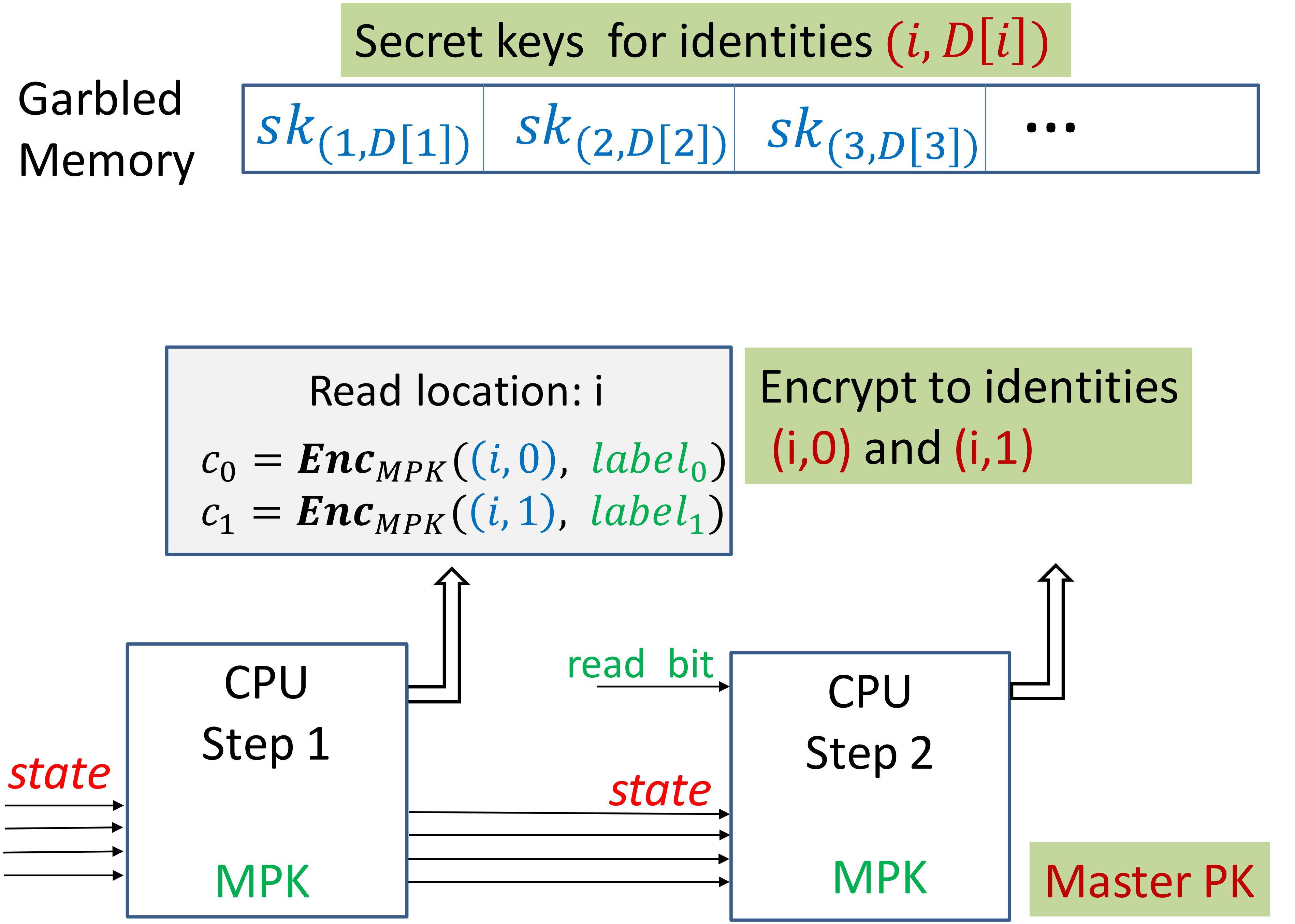
- Fix 1 :
  - Using identity-based encryption (IBE).
  - Polylogarithmic overhead
- Fix 2 :
  - Only use one-way functions.
  - Overhead n^{ε}.

The Fix

- Public-key instead of symmetric-key encryption.
  - Garbled circuits have hard-coded public key. No secrets.
  - Semantic security of ciphertexts holds even given public-key which is hard-coded in all garbled circuits.
- Caveat: need *identity-based encryption (IBE)*
  - Original solution used “Sym-key IBE” = PRF + Sym-Enc.



Secret keys for identities $(i, D[i])$
Garbled Memory
$F_k(1, D[1])$
$F_k(2, D[2])$
$F_k(3, D[3])$
…
Read location: i
$c_0 = \boldsymbol{Enc}\,(F_k(i,0),\ label_0),$
$c_1 = \boldsymbol{Enc}\,(F_k(i,1),\ label_1)$
Encrypt to identities (i,0) and (i,1)
read bit
CPU Step 1
PRF Key: k
state
state
CPU Step 2
PRF Key: k
Master SK


Secret keys for identities $(i, D[i])$
Garbled Memory
$sk_{(1,D[1])}$
$sk_{(2,D[2])}$
$sk_{(3,D[3])}$
...
Read location: i
$c_0 = \boldsymbol{Enc}_{MPK}((i,0),\ label_0)$
$c_1 = \boldsymbol{Enc}_{MPK}((i,1),\ label_1)$
Encrypt to identities (i,0) and (i,1)
read bit
CPU Step 1
CPU Step 2
state
state
MPK
MPK
Master PK

- **Theorem:** Assuming IBE, get **garbled RAM**:
  For any RAM program w. run-time T , data of size N
  - Garbled memory-data is of size: $O(N)$.
  - Garbled program size, creation/evaluation-time: $O\big(T \cdot polylog(N)\big)$.
  - Supports "persistent memory data".

PART II

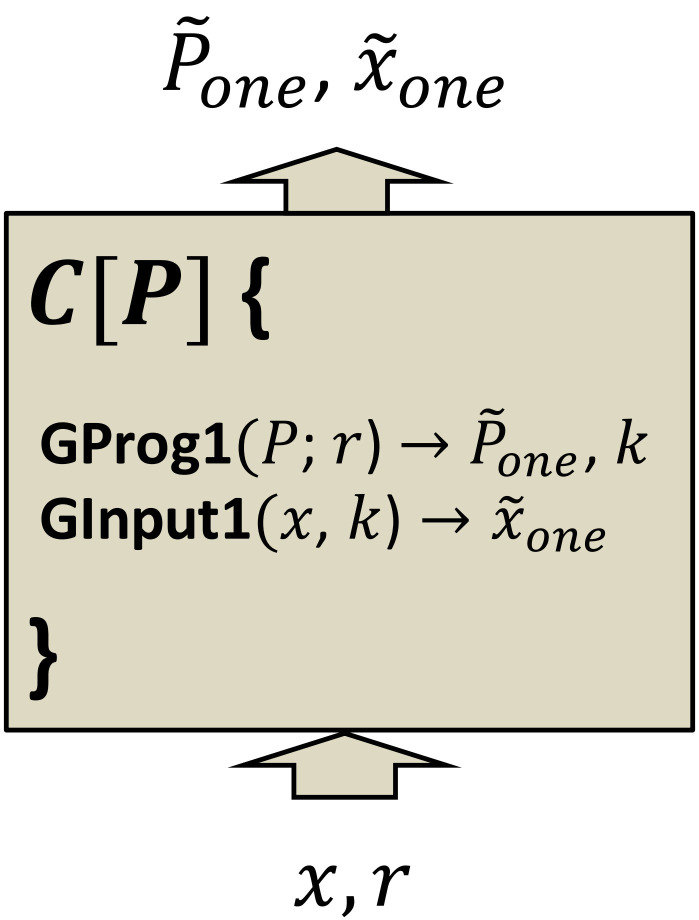
Reusable Garbled RAM

Security of Reusable Garbled RAM (without persistent data)

Simulate $\tilde{P}, \tilde{x}_1, \tilde{x}_2, \ldots$

given $P, y_1 = P(x_1), y_2 = P(x_2) \ldots$

- **Construction idea** by combining:
  - one-time garbled RAM (GProg1, GInp1, GEval1)
  - reusable garbled circuits



Reusable GProg: $\tilde{P}_{reuse}$
reusable circuit-garbling of $C[P]$

Reusable GInput: $\tilde{x}_i$
garbled input for $\tilde{C}[P]$

- Size of $C[P]$ = (RAM run-time of P)
- |input| = O(|x|)
- |output| = (RAM run-time of P)

- **Construction idea** by combining:
  - one-time garbled RAM (GProg1, GInp1, GEval1)
  - reusable garbled circuits

$\tilde{P}_{one}, \tilde{x}_{one}$

$\boldsymbol{C[P]}$ **{**

GProg1$(P; r) \rightarrow \tilde{P}_{one}, k$
GInput1$(x, k) \rightarrow \tilde{x}_{one}$

}

x, r

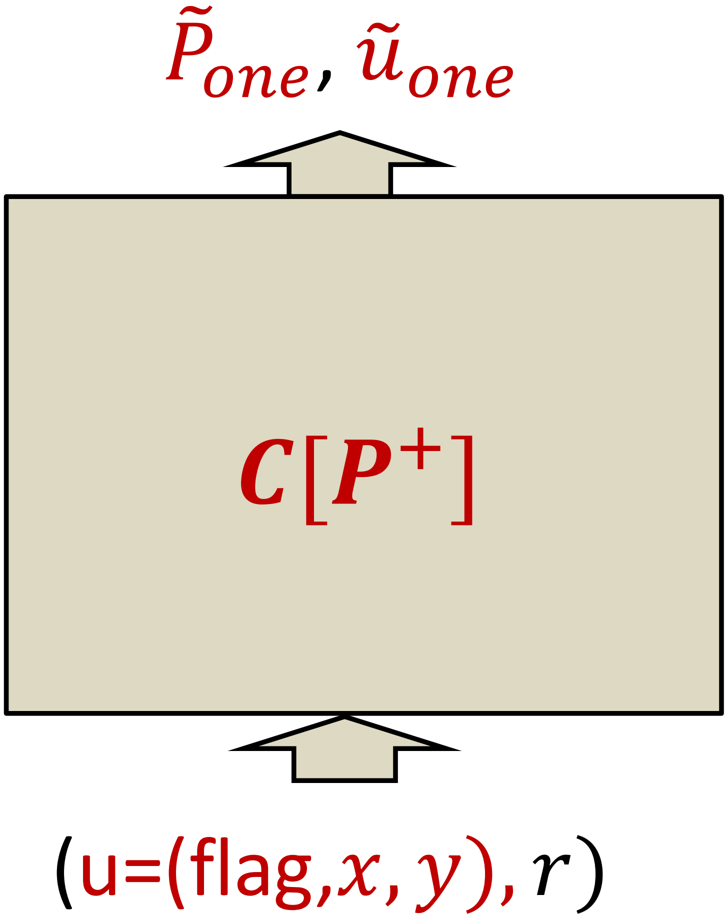
Problem: In reusable garbled circuits of [GKPVZ13], size of garbled input always exceeds size of circuit output.

Unfortunately: This is inherent. Cannot do better if want simulation security.

- Size of $C[P]$ = (RAM run-time of P)
- |input| = O(|x|)
- |output| = (RAM run-time of P)

Distributional Indistinguishability

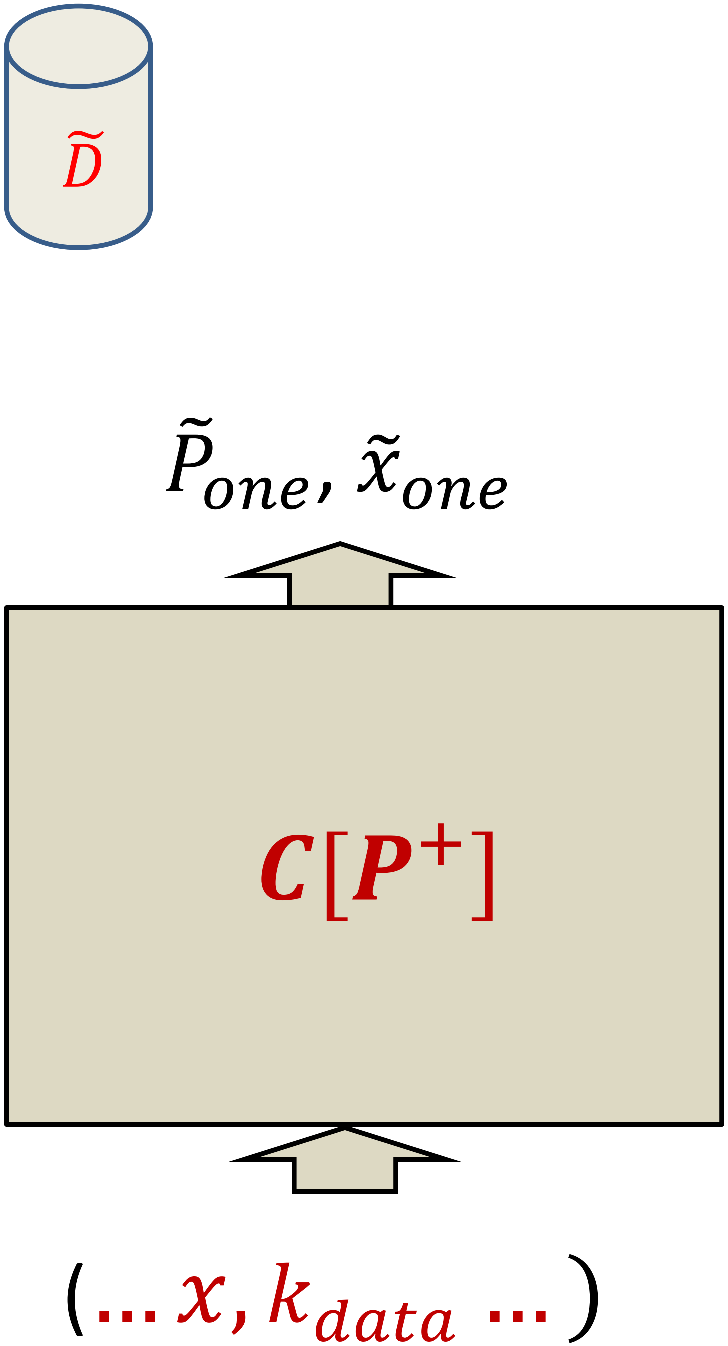
- **Solution idea:** new/weaker security notion for garbled circuits.
  - For circuit C and independent distributions $\{w_i\}$, $\{w_i'\}$ s.t.
    $$C(w_i) \approx C(w_i')$$
    we get
    $$[\widetilde{\boldsymbol{C}}, \widetilde{w}_1, \widetilde{w}_2, \dots, \widetilde{w}_n] \approx [\widetilde{\boldsymbol{C}}, \widetilde{w}'_1, \widetilde{w}'_2, \dots, \widetilde{w}'_n]$$
- Follows from indistinguishability obfuscation, or functional encryption for circuits.
  - Can garble circuits with huge output, small garbled input.
- Stronger variant “correlated distributional ind.”: the distributions are not necessarily independent.
  - Follows from stronger notions of obfuscation.



- "Real-or-Dummy" program

```
P+( flag, x, y ) {
    if flag=1    // real
        output P(x)
    else         //dummy
        output  y
}
```

- **User:** garbles inputs ((flag=1, $x, y = \perp$) , r)
- **Simulator:** garbles inputs ((flag=0,$x = \perp, y$), r)



- **Persistent memory:** Use 1-time garbled RAM to compute: $\tilde{D}, k_{data} \leftarrow$ GData(D)

- **Problem:** inputs to $\boldsymbol{C}[\boldsymbol{P}^{+}]$ have a common secret k_{data}.
  – Need "correlated distributional ind." security.

- **Theorem:** Get reusable garbled RAM where:
  - Garble, evaluate program: O(RAM run-time P).
  - Garble input = O(input + output size).

  assuming “ind. obfuscation” + stat. sound NIZK.

- **Theorem:** Get reusable garbled RAM with persistent memory where:
  - Optional: garble data = O(data size)
  - garble program = O(description size P)
  - garble input = O(input + output size)
  - evaluate = O(RAM run-time P)

  assuming “strong differing-inputs obfuscation”.

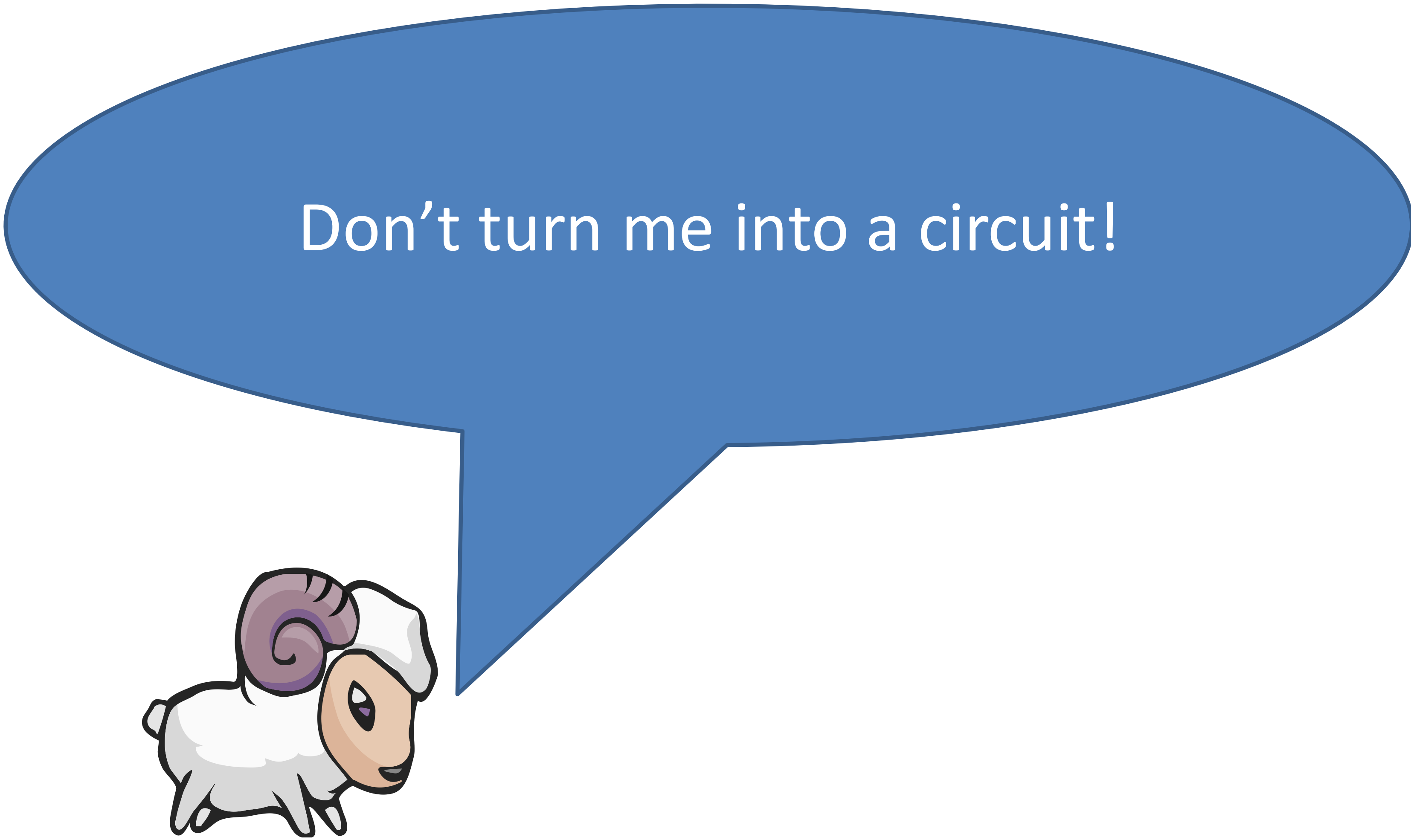
Summary

- Outsource Private RAM computation via "reusable garbled RAM".

- One-Time Garbled RAM
  - Avoid circularity issue in [LO13] via IBE
  - Can also use OWFs at the cost of higher overhead
  - Best of both worlds?

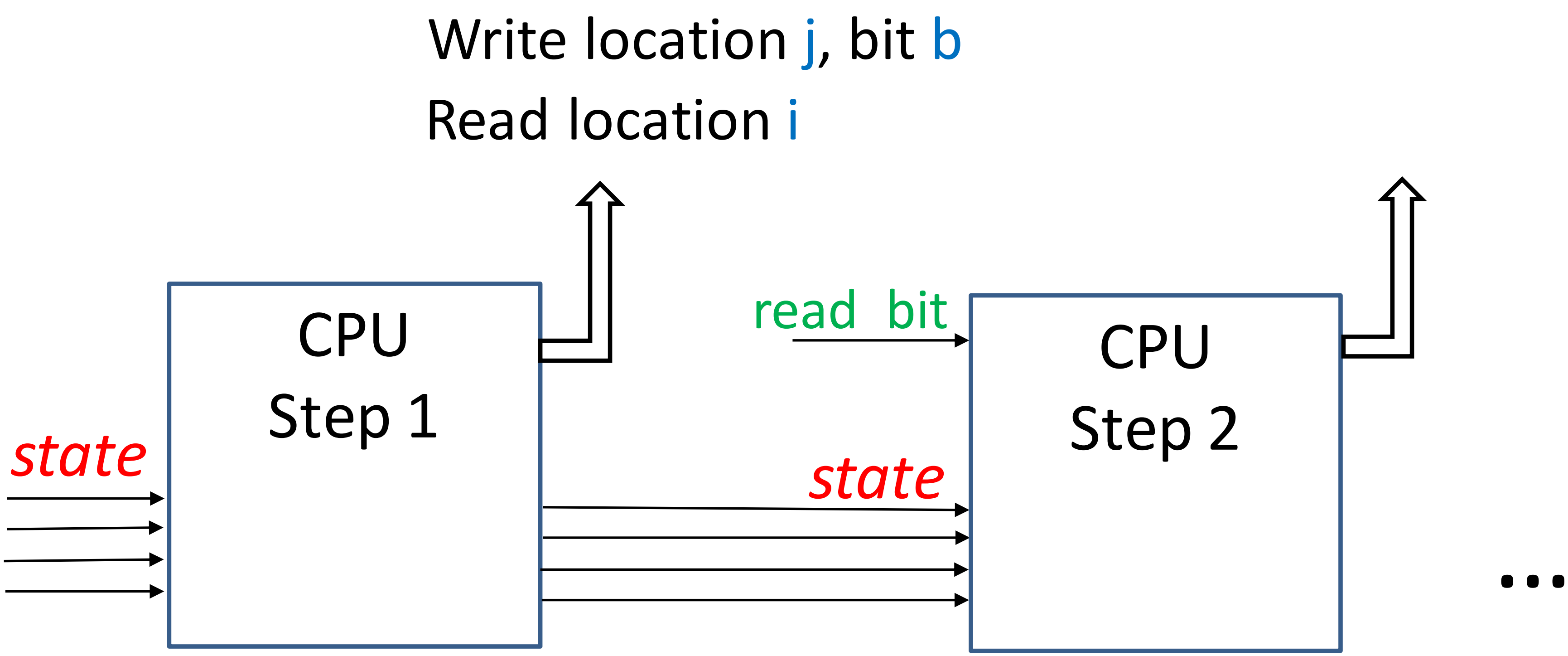
- Reusable Garbled RAM
  - Construction from one-time RAM + reusable circuits.
    - "[correlated] distributional indistinguishability"
  - Instantiations using "obfuscation" assumptions.
  - Weaker assumptions?

Thank You!



How to allow writes?

Predictably-Timed Writes:
Whenever read location i,
"know" its last-write-time u.



How to allow writes?

- Garbled memory = $\{ sk_{ID} : ID = (j,i,b)\}$
  - i = location.
  - j = last-write time of location i.
  - b = bit in location i written in step j.
- To read location i, need to know last-write time j.
  - Encrypt labels to identities $(j,i,0)$ and $(j,i,1)$
- To write location i, at time j
  - Create secret key for $ID = (j,i,b)$.
  - Need master secret key. Reintroduces circulairty!

How to allow writes?

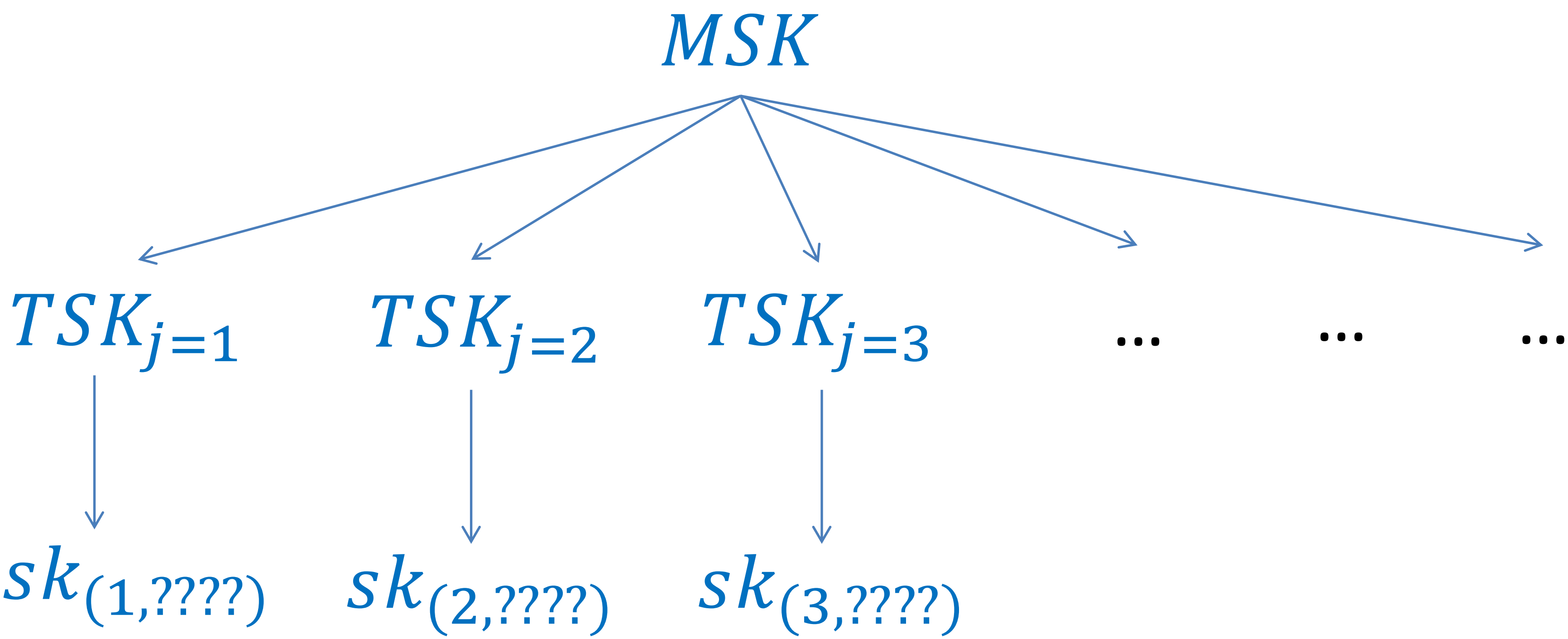
- Idea: CPU step j can create secret key for any $ID = (j, *)$ but cannot decrypt for identities $j' \neq j$.

- Prevents circularity: Translation ciphertext created by CPU step j maintain semantic security even given secrets contained in CPU steps $j+1, j+2, \ldots$

- Need "restricted MSK" for time-period j.

- Timed IBE (TIBE): restricted notion of HIBE.

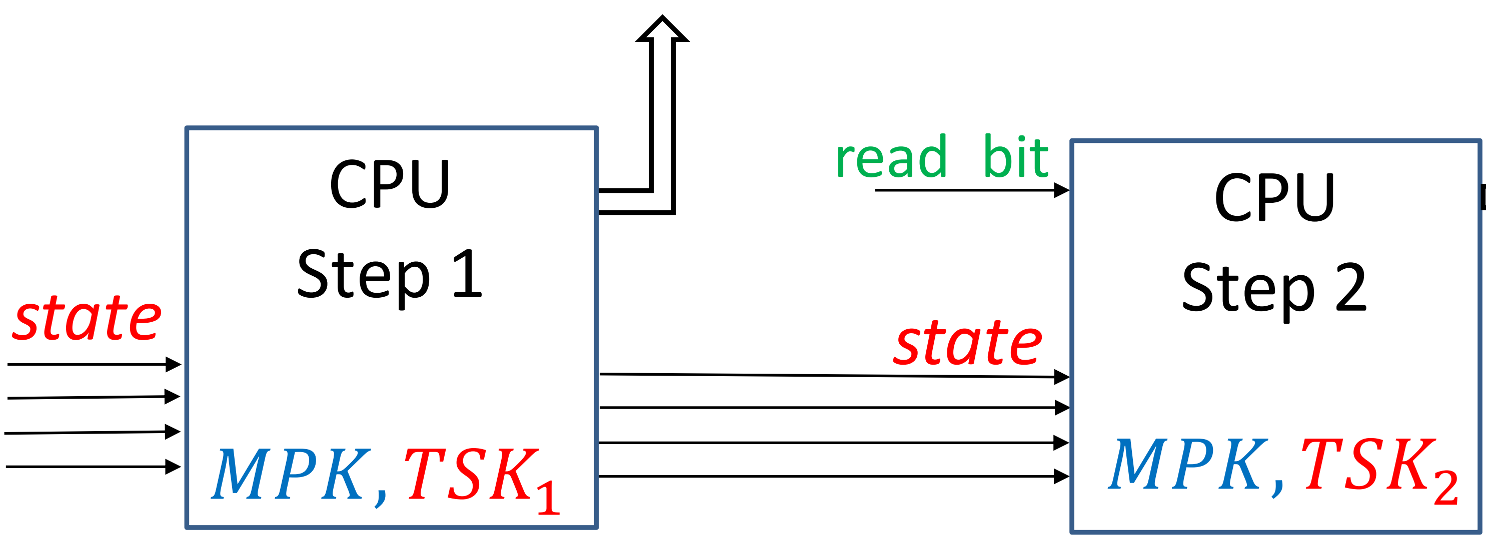
- Timed IBE (TIBE): restricted notion of HIBE.
  - Time-period key TSK_j can be used to create a *single* identity secret key for *any* identity ID = (j, *).
  - Semantic security holds for all other j.
- Can construct TIBE from any IBE. (see paper)



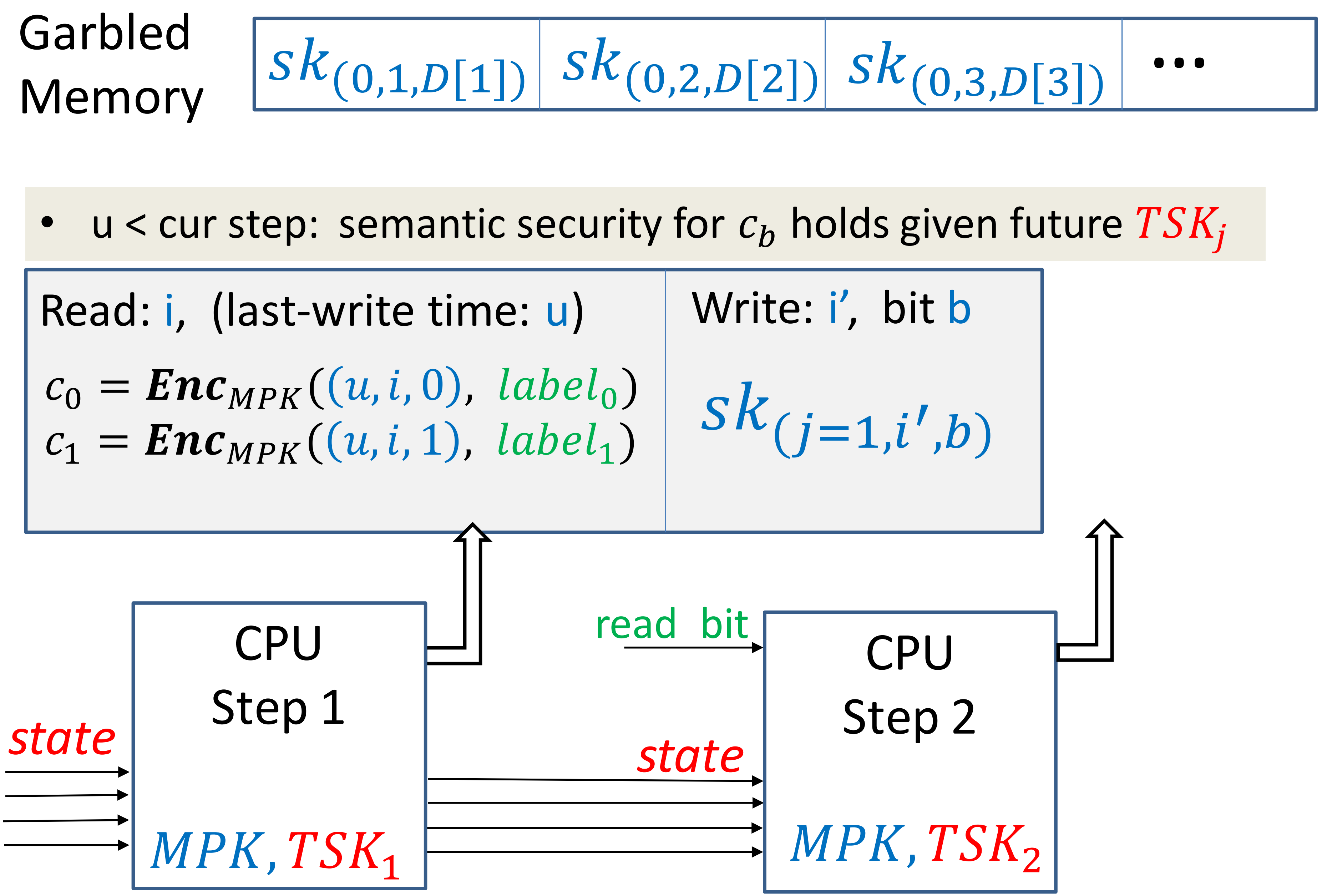
Garbled Memory

$sk_{(0,1,D[1])}$	$sk_{(0,2,D[2])}$	$sk_{(0,3,D[3])}$	…

- initially all keys have time j=0
- Invariant: always have $sk_{(j,i,b)}$ where j=last-write-time(i), and b is latest bit.



Step j has $TSKj$



Garbled Memory
$sk_{(0,1,D[1])}$
$sk_{(0,2,D[2])}$
$sk_{(0,3,D[3])}$
…
• u < cur step: semantic security for c_b holds given future TSK_j
Read: i, (last-write time: u)
$c_0 = \boldsymbol{Enc}_{MPK}((u, i, 0),\ label_0)$
$c_1 = \boldsymbol{Enc}_{MPK}((u, i, 1),\ label_1)$
Write: i’, bit b
$sk_{(j=1,i',b)}$
CPU
Step 1
state
MPK, TSK_1
read bit
state
CPU
Step 2
MPK, TSK_2