Garbled RAM, Revisited

Daniel Wichs

(Northeastern University)

Joint work with: Craig Gentry, Shai Halevi, Seteve Lu,

Rafail Ostrovsky, Mariana Raykova

Goals of Garbled RAM

- An analogue of Yao garbled circuits [Yao82] that directly garbles Random Access Machines (RAM).
- Avoid efficiency loss of converting a RAM to a circuit.
  - Google search vs. reading the Internet.

- First proposed/constructed by [Lu-Ostrovsky 13].
  - Proof of security contains subtle flaw (circularity problem).

- This works: new constructions with provable security.

Garbled RAM Definition

Client

secret: k

$\mathbf{GData}(D, k) \rightarrow \tilde{D}$

$\mathbf{GProg}(P, k) \rightarrow \tilde{P}$

$\mathbf{GInput}(x, k) \rightarrow \tilde{x}$

Server

$\mathbf{Eval}^{\tilde{D}}(\tilde{P}, \tilde{x}) \rightarrow y$

Garbled RAM Definition

Client

secret: k

GData$(D, k) \rightarrow \tilde{D}$

GProg$(P_i, k, i) \rightarrow \tilde{P}_i$ $\approx$ O(run-time)

GInput$(x_i, k, i) \rightarrow \tilde{x}_i$

Server

- **Security:** server only learns $y_1, y_2, \ldots$ (even data access pattern is hidden!)

Eval$^{\tilde{D}}(\tilde{P}_i, \tilde{x}_i) \rightarrow y_i$

$\approx$ O(run-time)

Weak vs. Full Security

- Weak security: May reveal data D, and *data-access pattern* of computations.
  - Locations of memory accessed in each step.
  - Values read and written to memory.

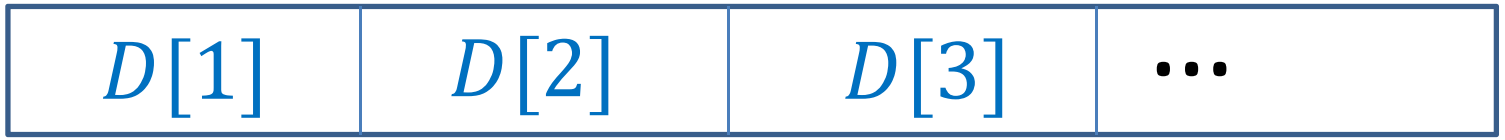
- Compiler: weak ⇒ full security:
  - Use oblivious RAM [GO96,…] to encode/access memory.

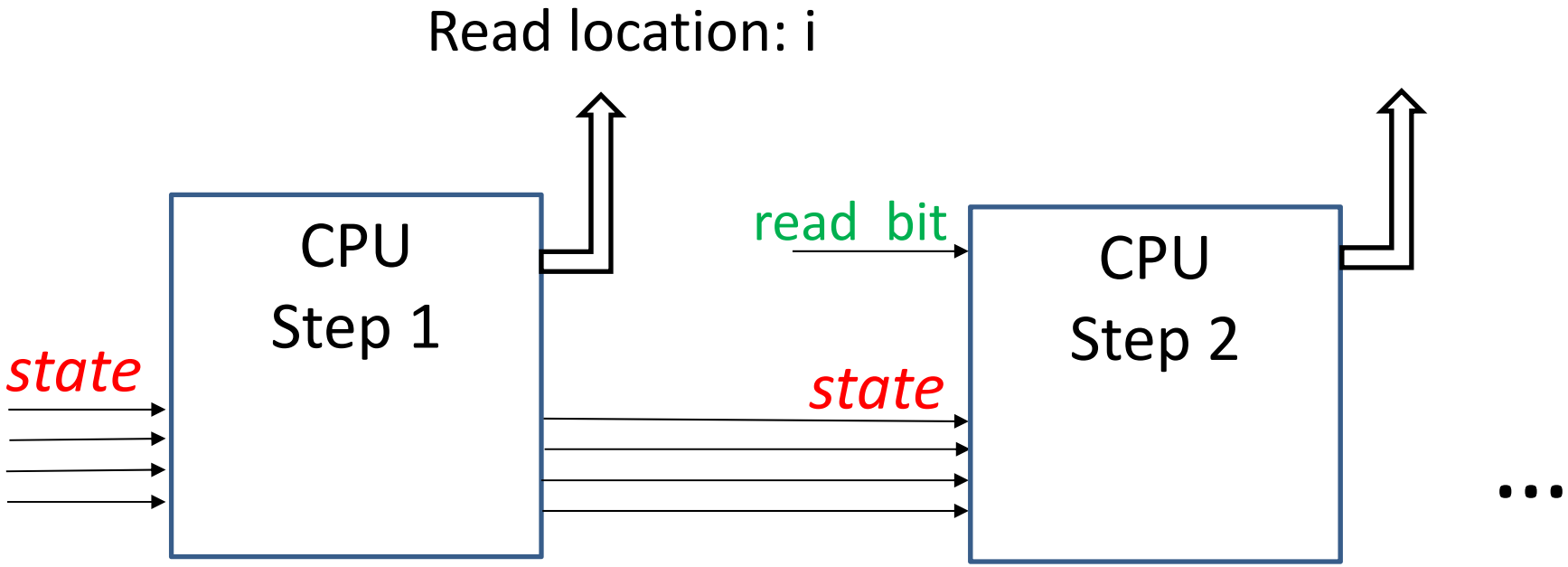
Overview of [Lu-Ostrovsky 13]

For now, read-only computation.

Memory

Data D=



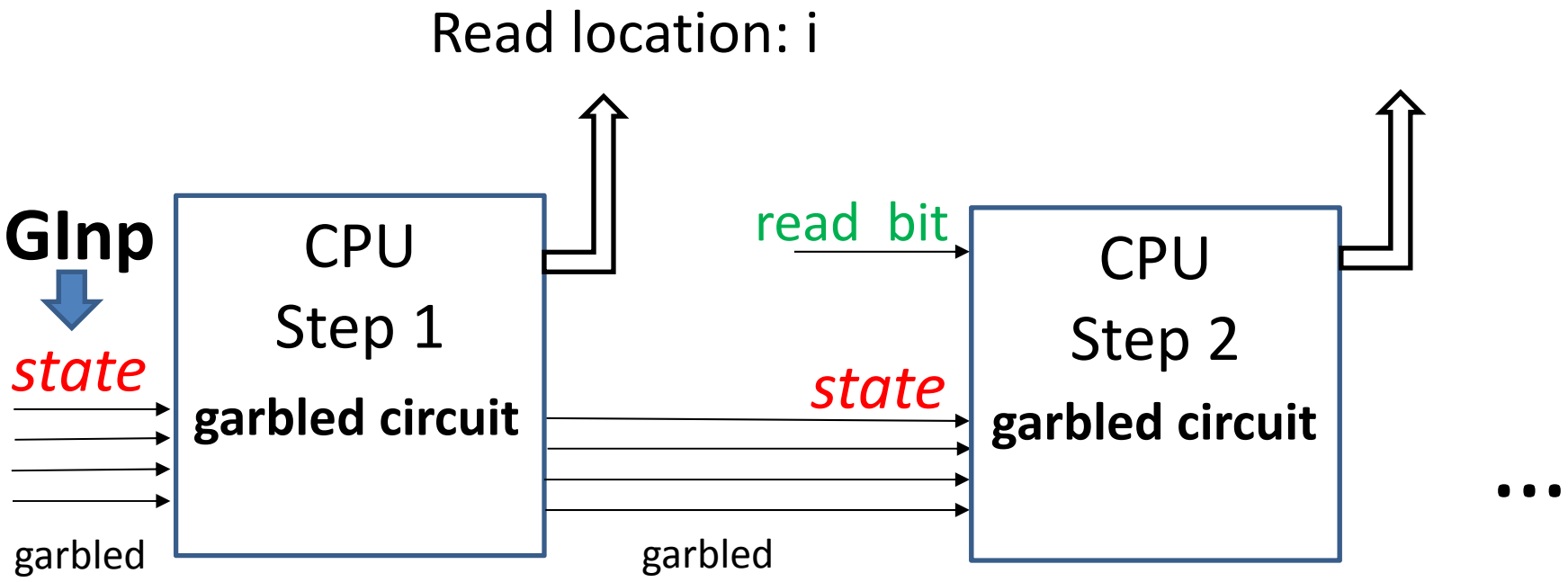


Memory

Data D=

$D[1]$	$D[2]$	$D[3]$	...

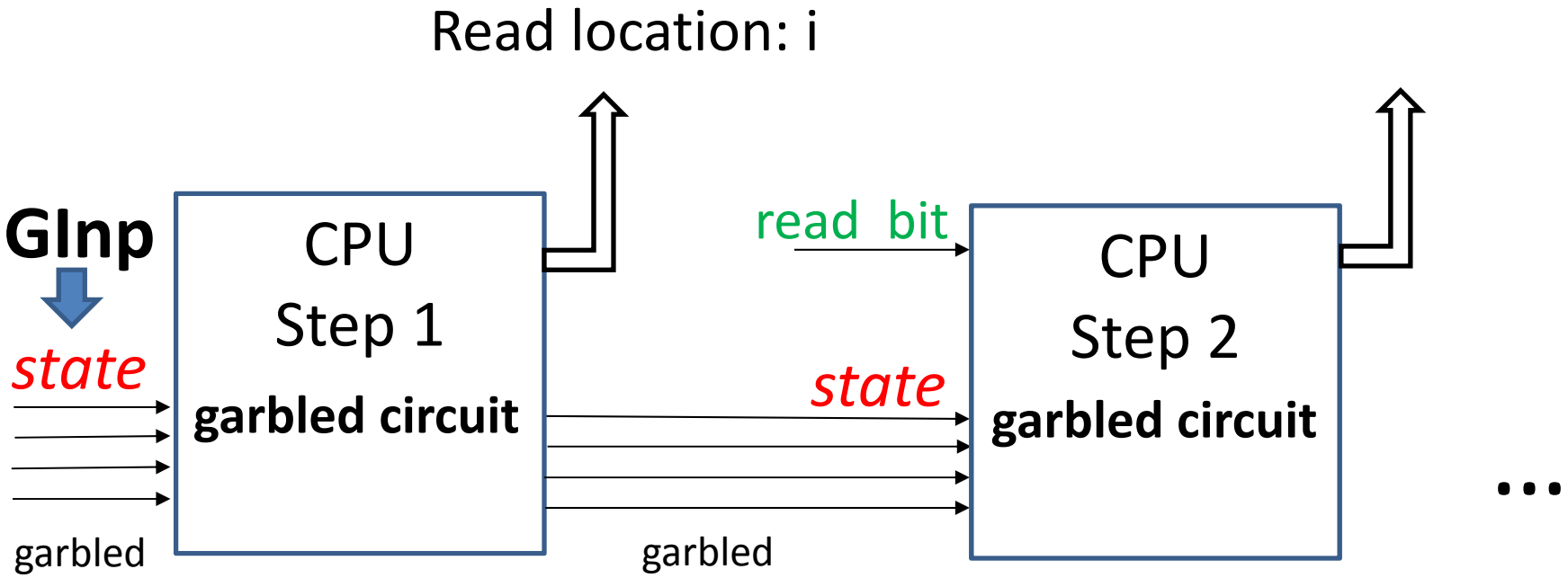
GProg:



GData:

$F_k(\ldots)$ is a PRF

GProg:



GData:

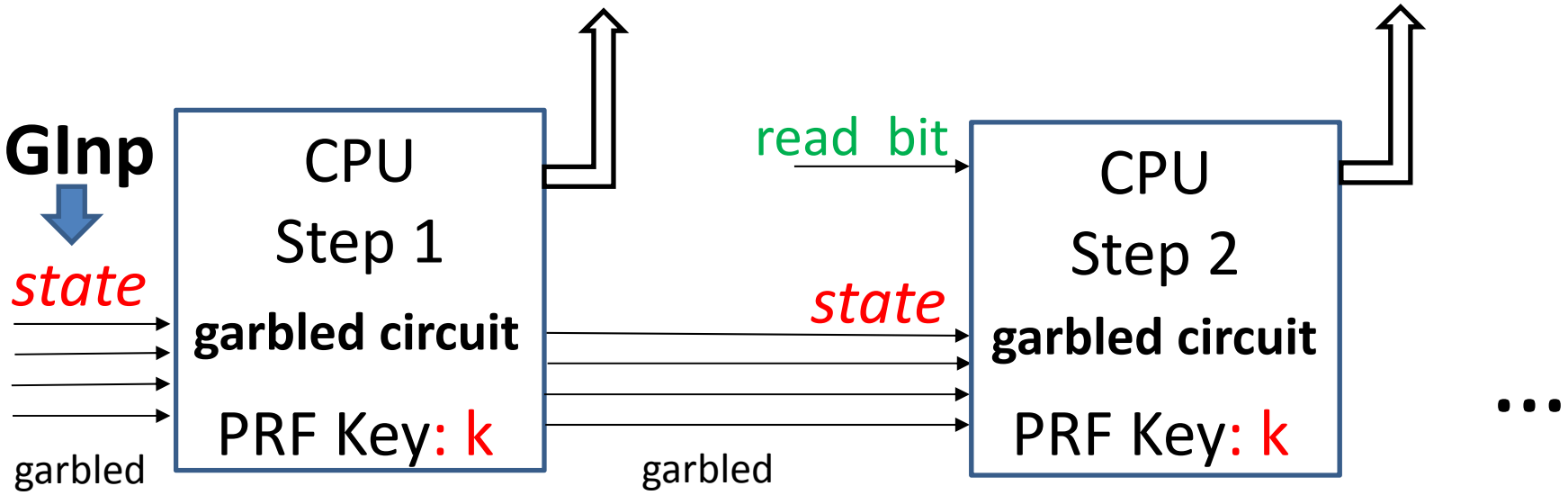
$F_k(1, D[1])$	$F_k(2, D[2])$	$F_k(3, D[3])$	...

$F_k(\dots)$ is a PRF

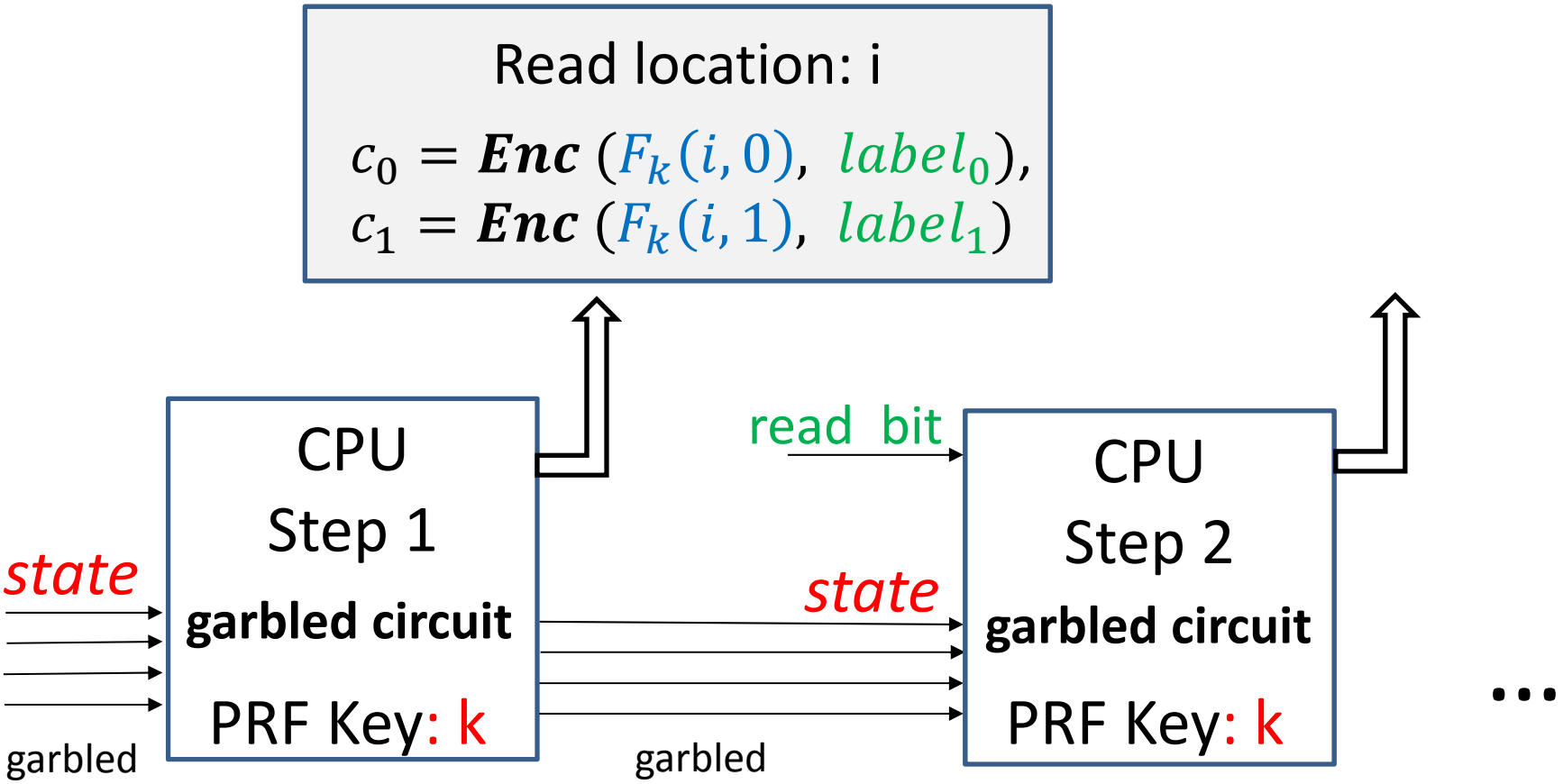
GProg:

Read location: i

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

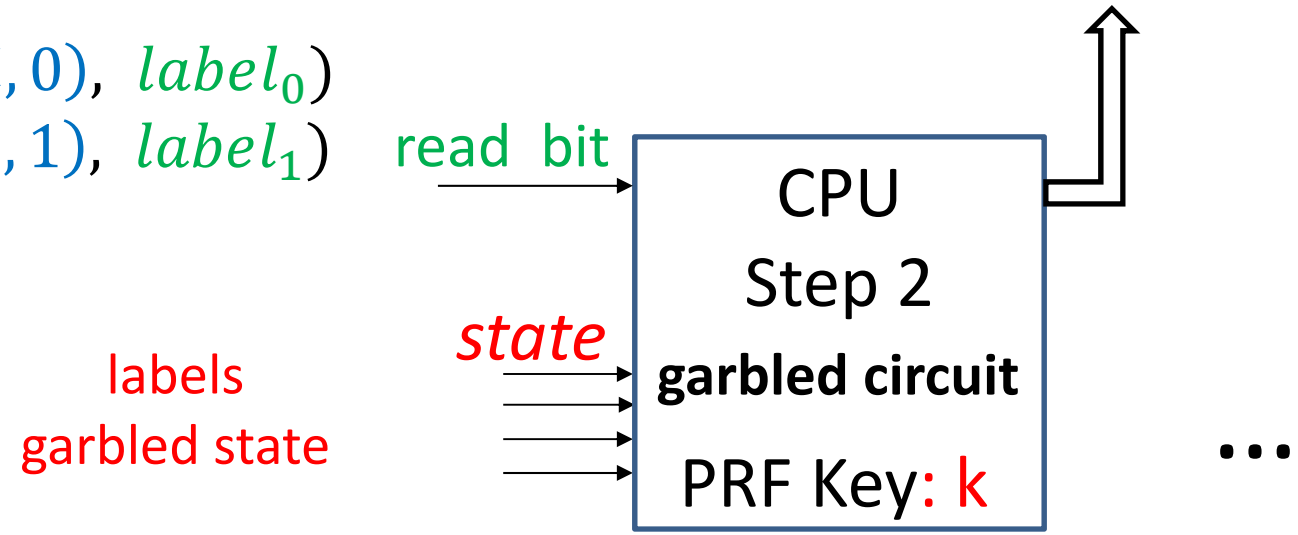


Let's try to prove security…



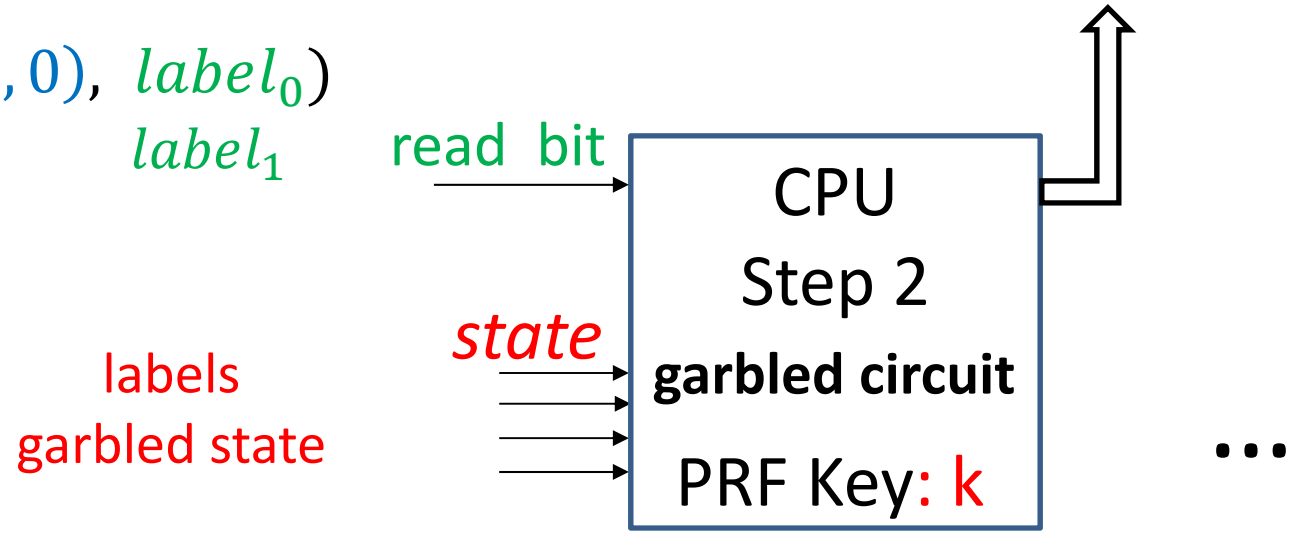
Use security of 1st garbled circuit
only learn output

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



Use security of 1st garbled circuit
only learn output (assume D[i]=1)

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



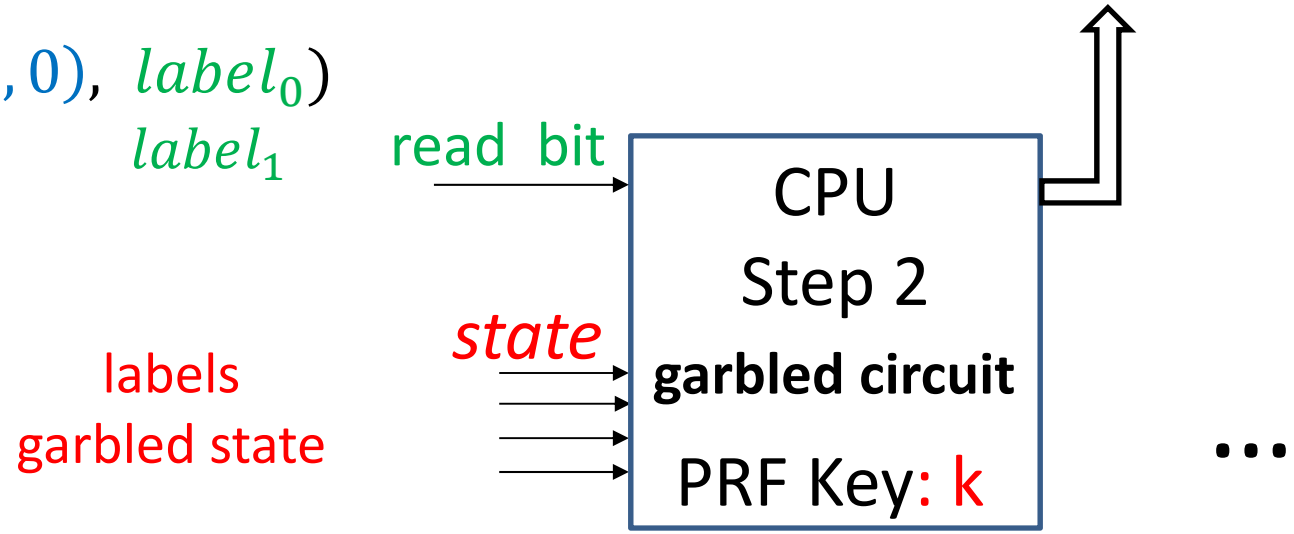
Use security of 2nd garbled circuit

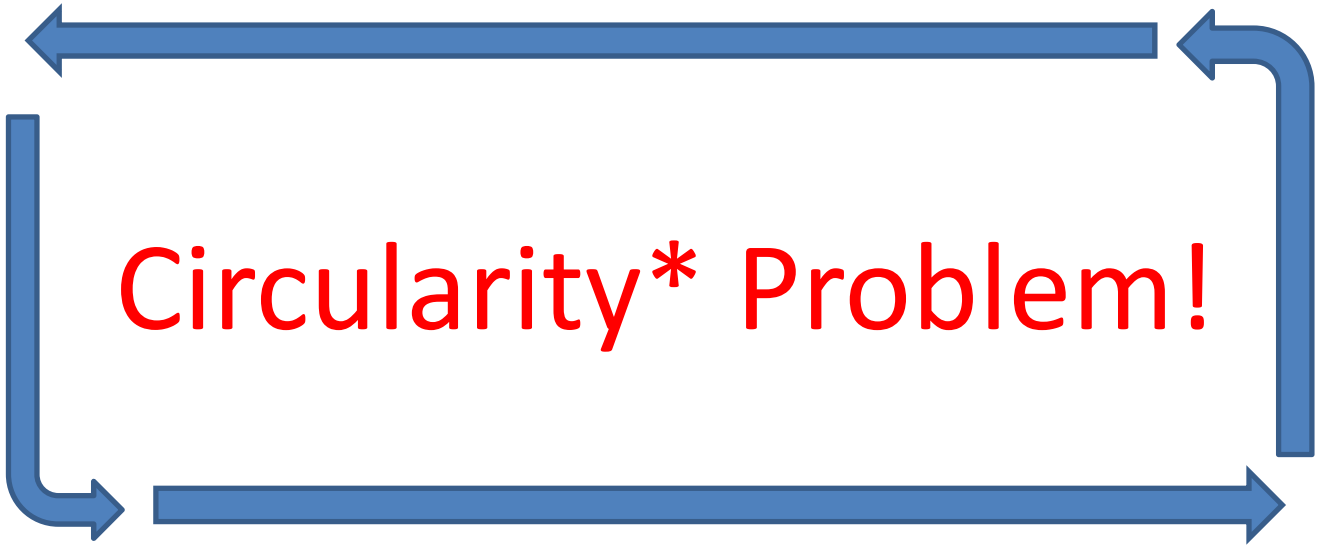
don't learn
$label_0$ for read bit

don't learn
PRF key k

Use security of Encryption/PRF

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





* May appear rectangular

So is it secure?

- Perhaps, but…
  - No proof.
  - No “simple” circularity assumption on one primitive.

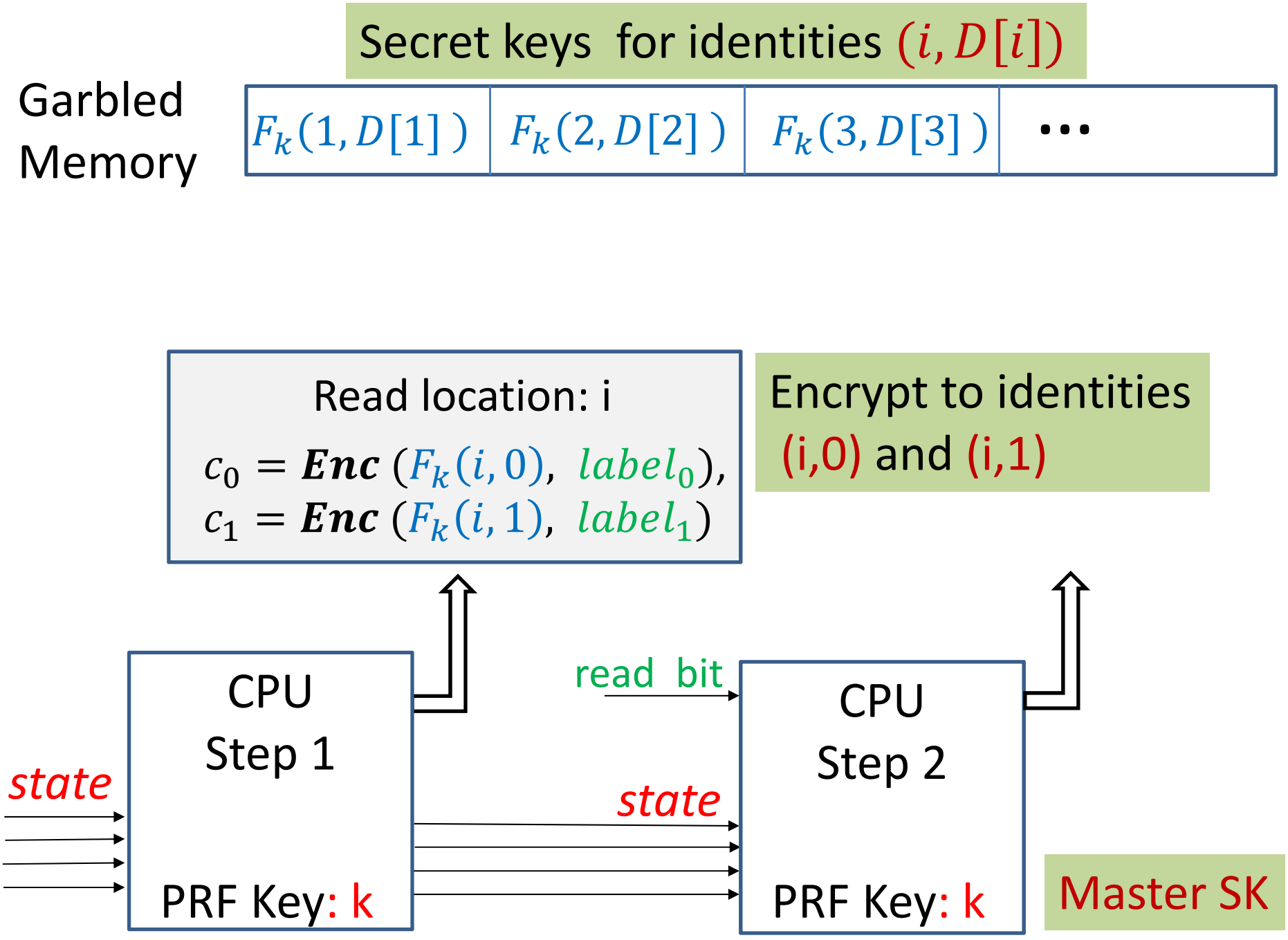
Can we fix it? Yes!

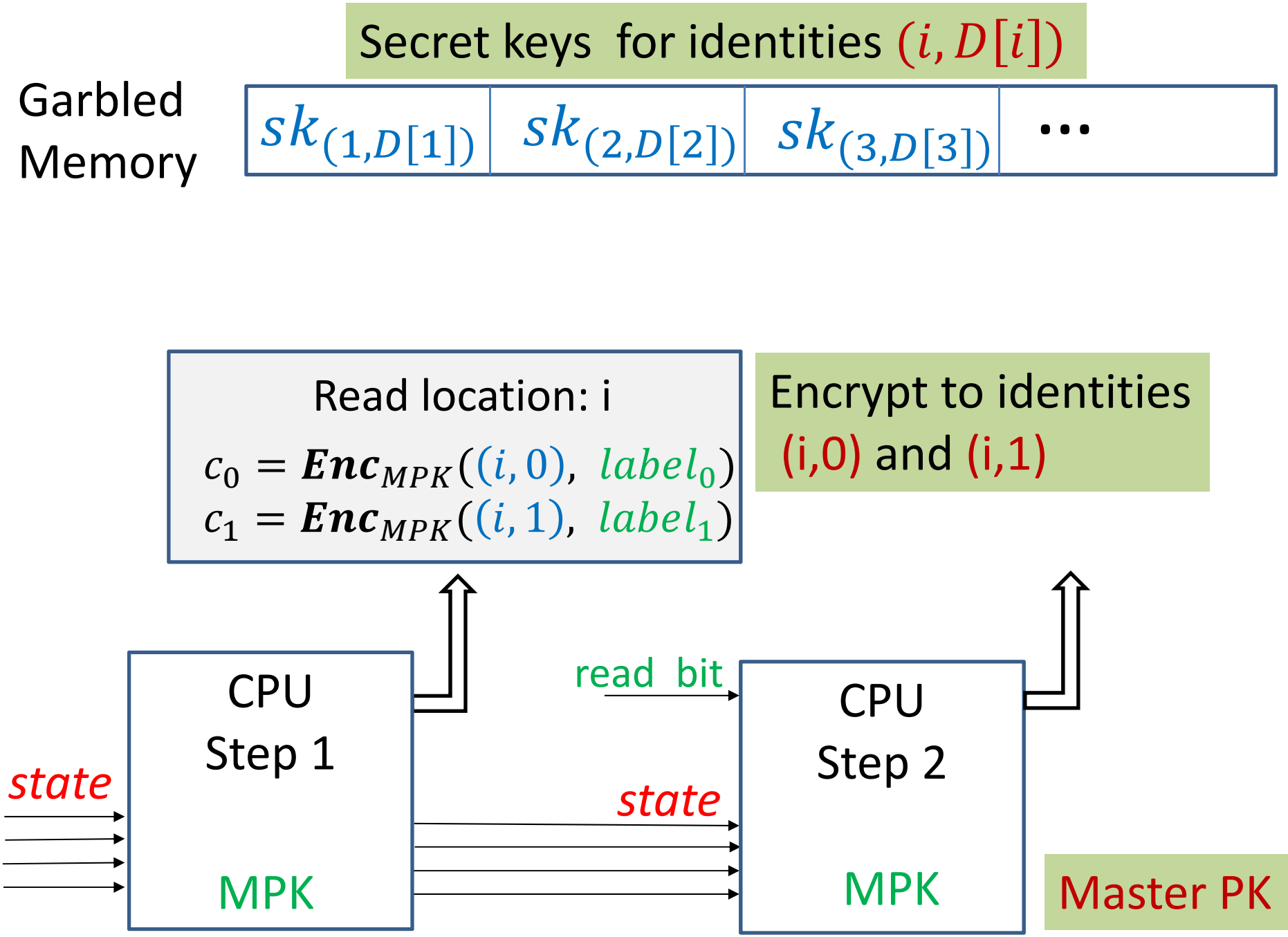
- Fix 1 :
  - Using identity-based encryption (IBE).

- Fix 2 :
  - Only use one-way functions.
  - Bigger overhead.

The Fix

- Public-key instead of symmetric-key encryption.
  - Garbled circuits have hard-coded public key.
  - Break circularity: security of ciphertexts holds even given public-key hard-coded in all garbled circuits.
- Caveat: need *identity-based encryption (IBE)*
  - Original solution used “Sym-key IBE”.


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)
CPU Step 1
PRF Key: k
state
read bit
state
CPU Step 2
PRF Key: k
Master SK

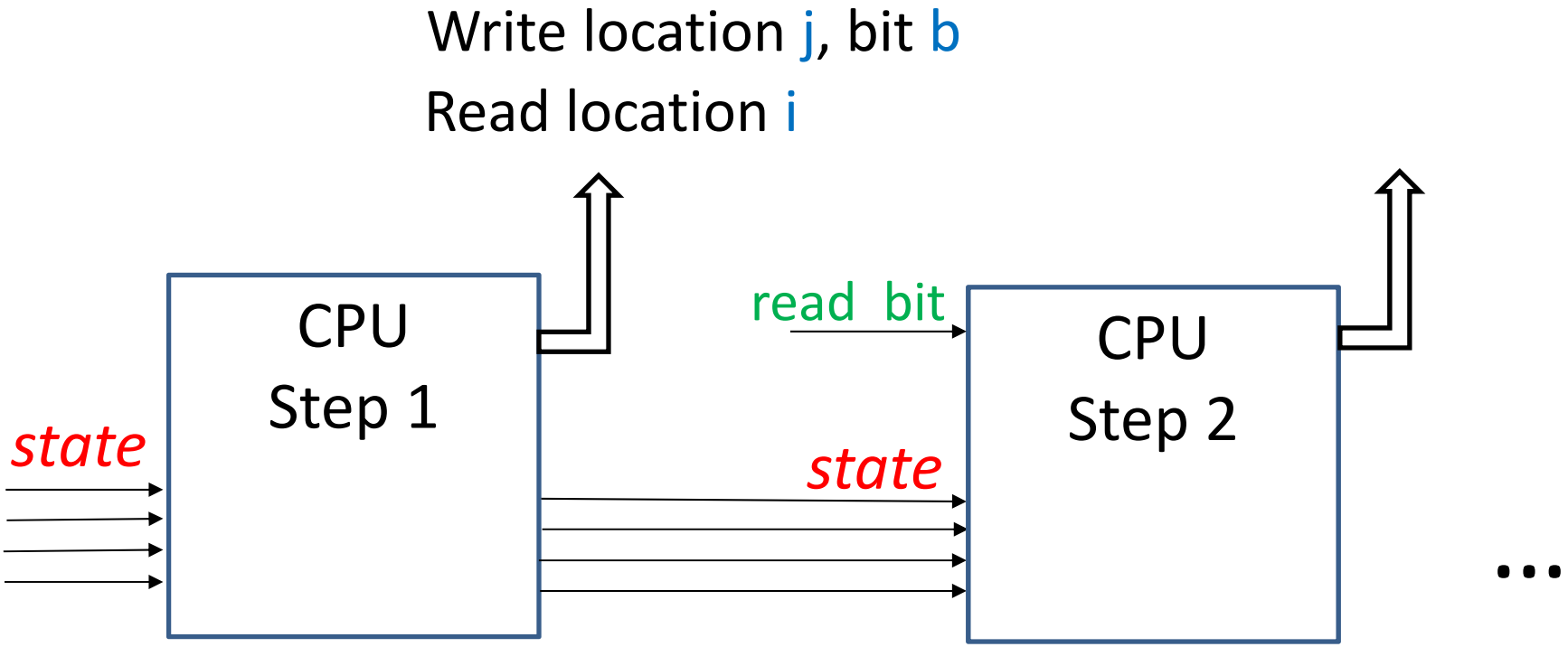

Secret keys for identities $(i, D[i])$
Garbled Memory
$sk_{(1,D[1])}$
$sk_{(2,D[2])}$
$sk_{(3,D[3])}$
...
Read location: i
$c_0 = \boldsymbol{Enc}_{MPK}((i,0),\ label_0)$
$c_1 = \boldsymbol{Enc}_{MPK}((i,1),\ label_1)$
Encrypt to identities (i,0) and (i,1)
CPU Step 1
MPK
state
read bit
state
CPU Step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

How to allow writes?

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

Compiler

Any Program



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 $j' < j$.
- Prevents circularity: Ciphertext created by CPU step j maintain semantic security even given secrets contained in all future CPU steps.
- Need "restricted MSK" for time-period j.
  - Use hierarchical IBE.
  - By being more careful, can use any IBE.

- **Theorem:** Assuming Identity Based Encryption (IBE),

  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: $O\left(T \cdot polylog(N)\right)$.

- **Theorem:** Assuming one-way functions,

  For any constant $\varepsilon > 0$:
  - Garbled memory-data is of size: $O(N)$.
  - Garbled program size, creation/evaluation-time: $O(T \cdot N^{\varepsilon})$.

Thank You!