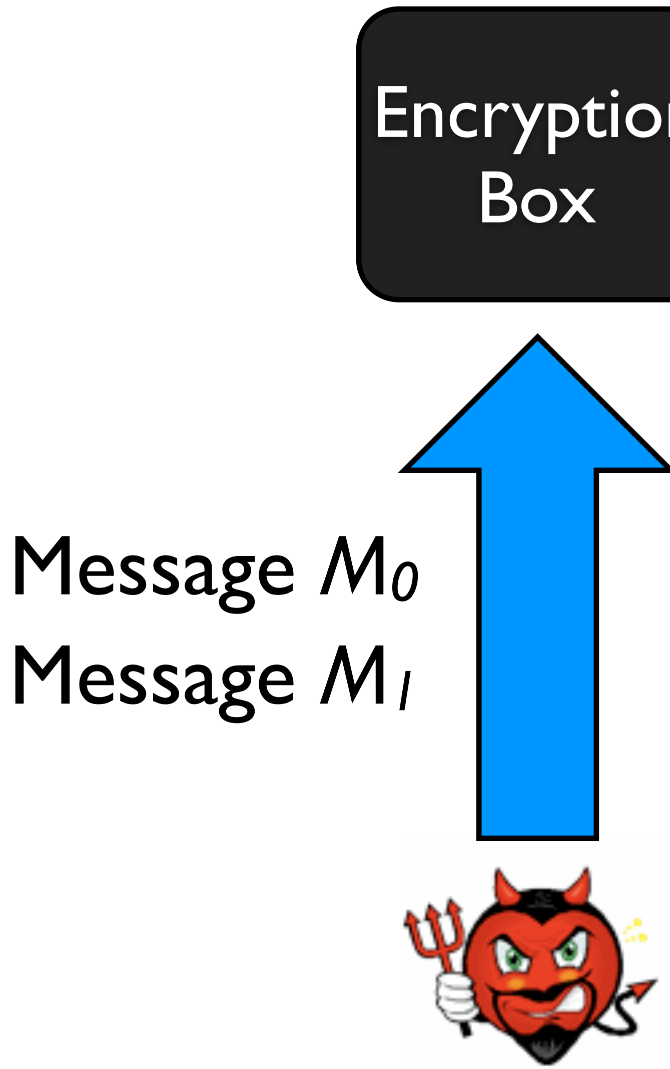
Leakage Resilient Computation

Yevgeniy Vahlis
joint work with Ali Juma, Charles Rackoff

Crypto

Encryption
Box

Crypto



Crypto

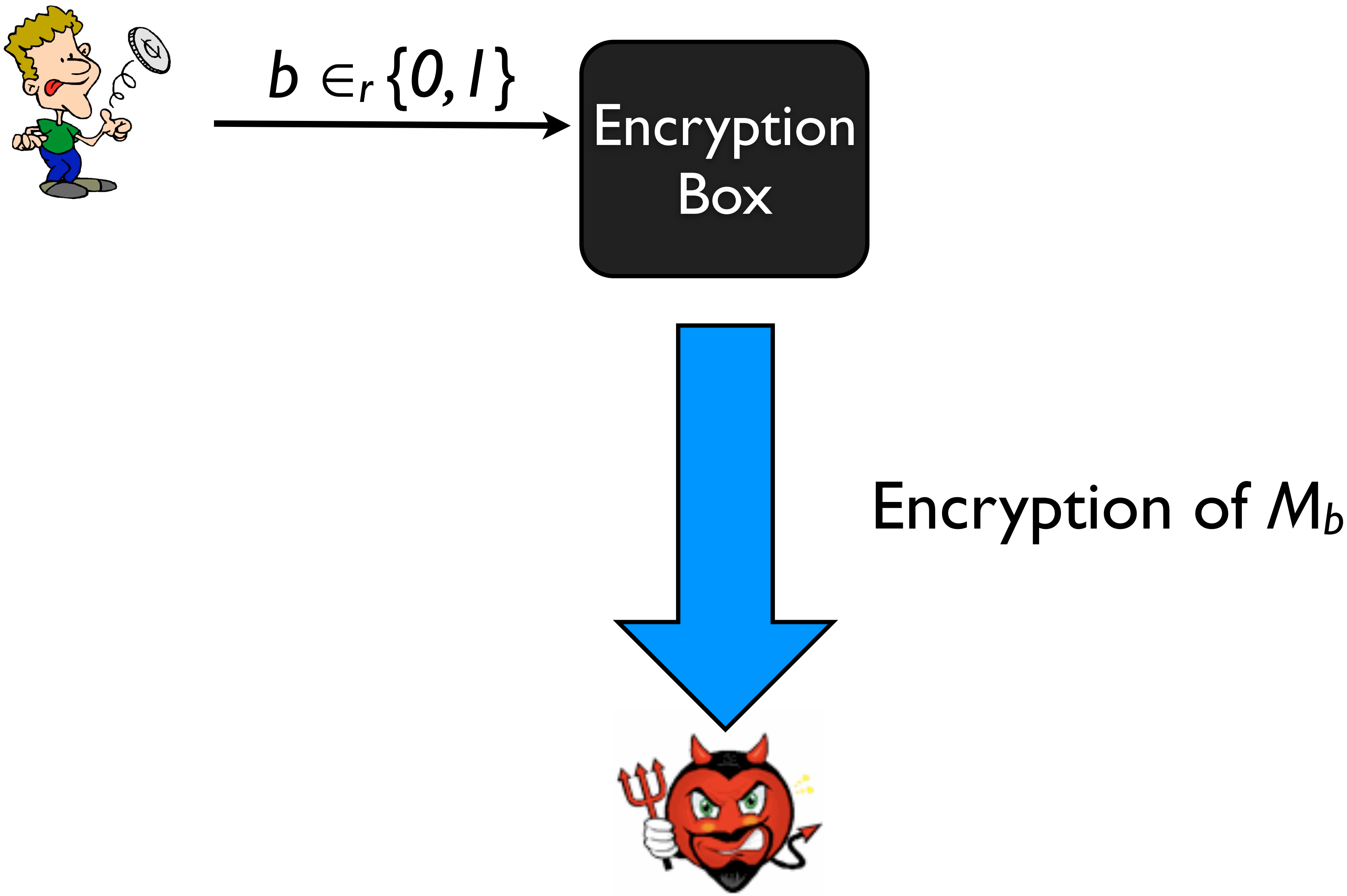
$b \in_r \{0,1\}$

Encryption Box

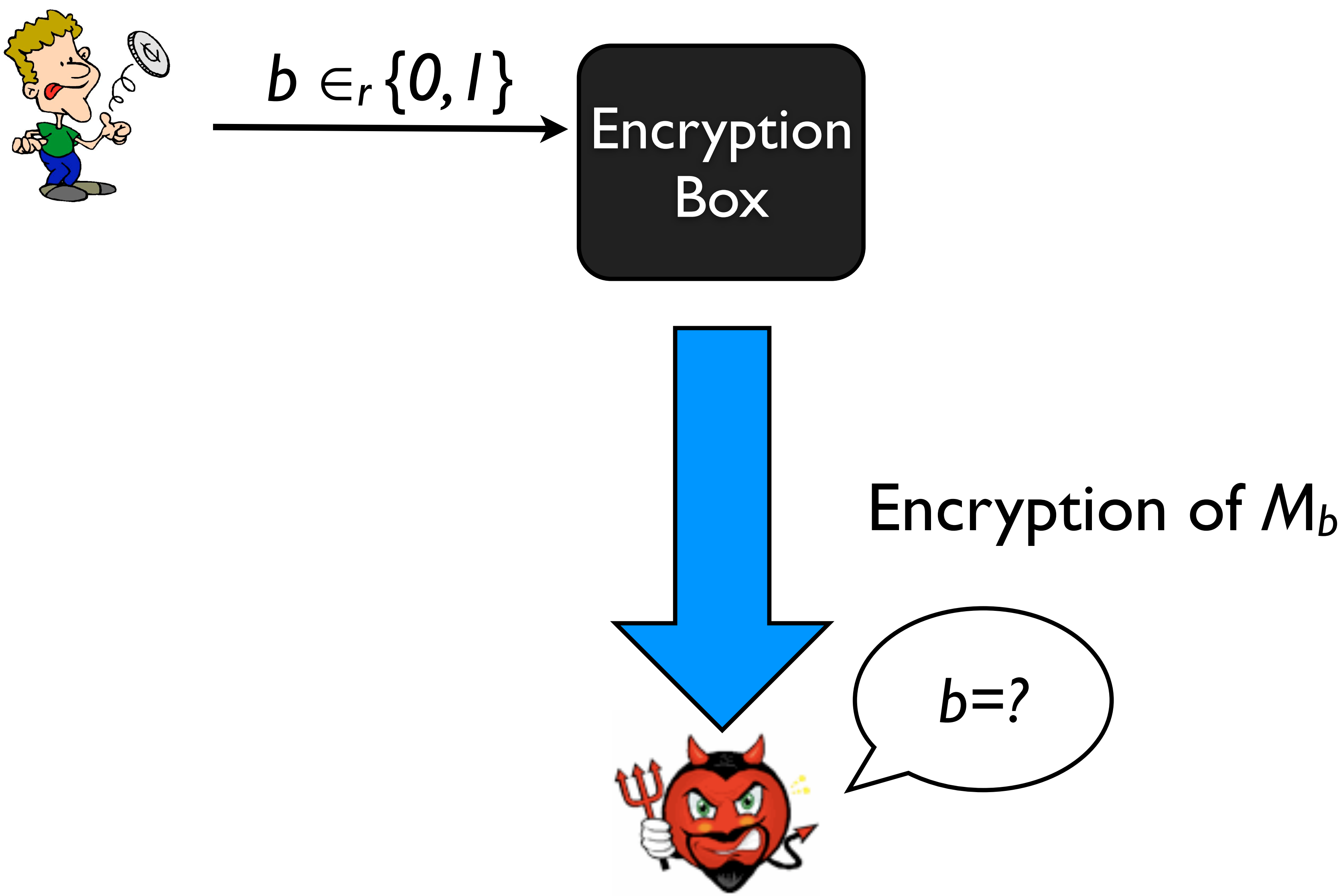
Message M_0

Message M_1

Crypto



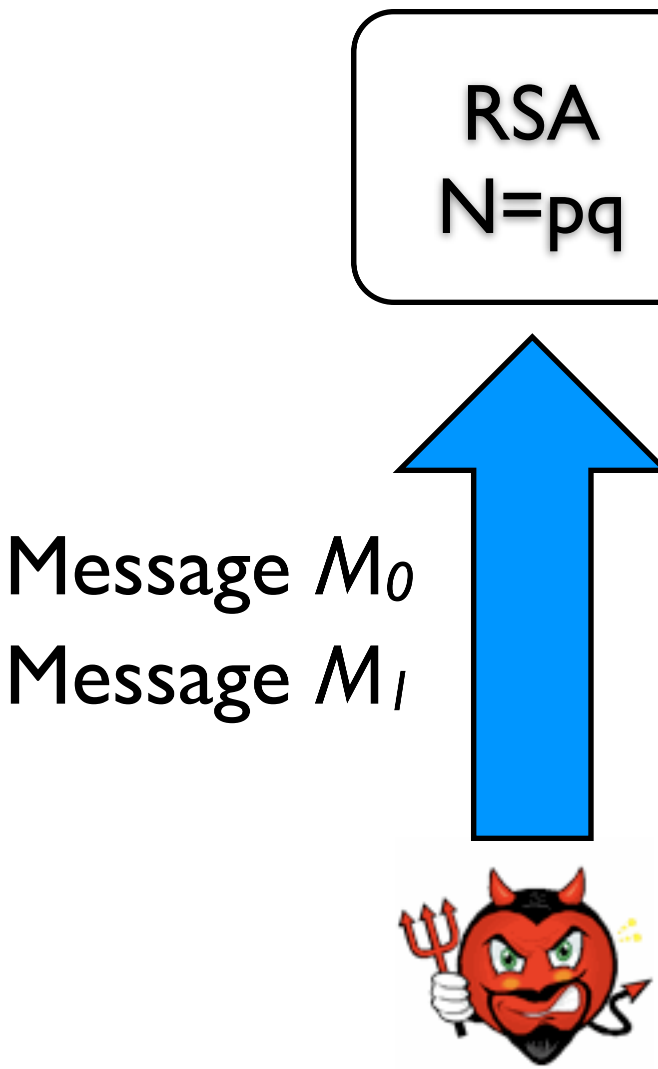
Crypto



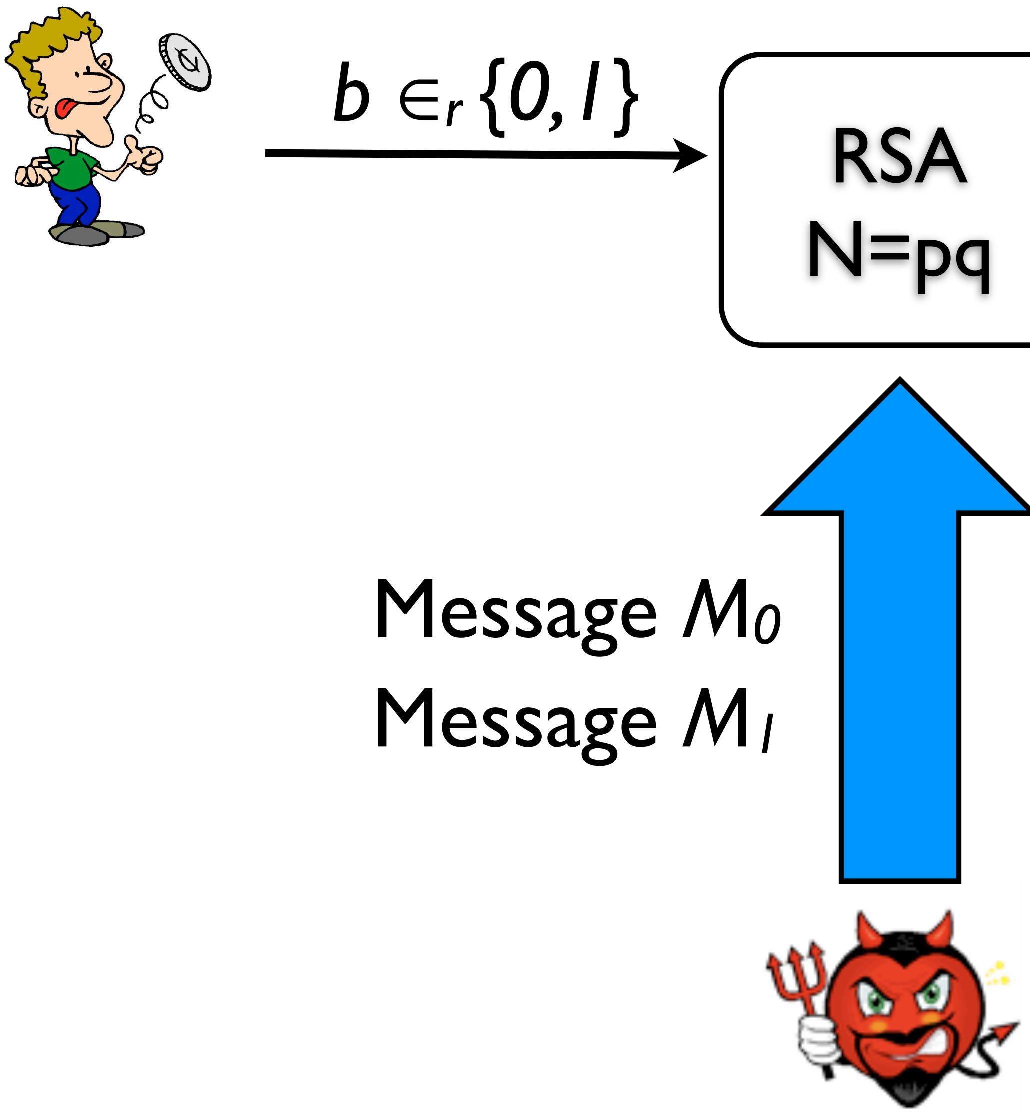
Crypto in Real Life

RSA
$N=pq$

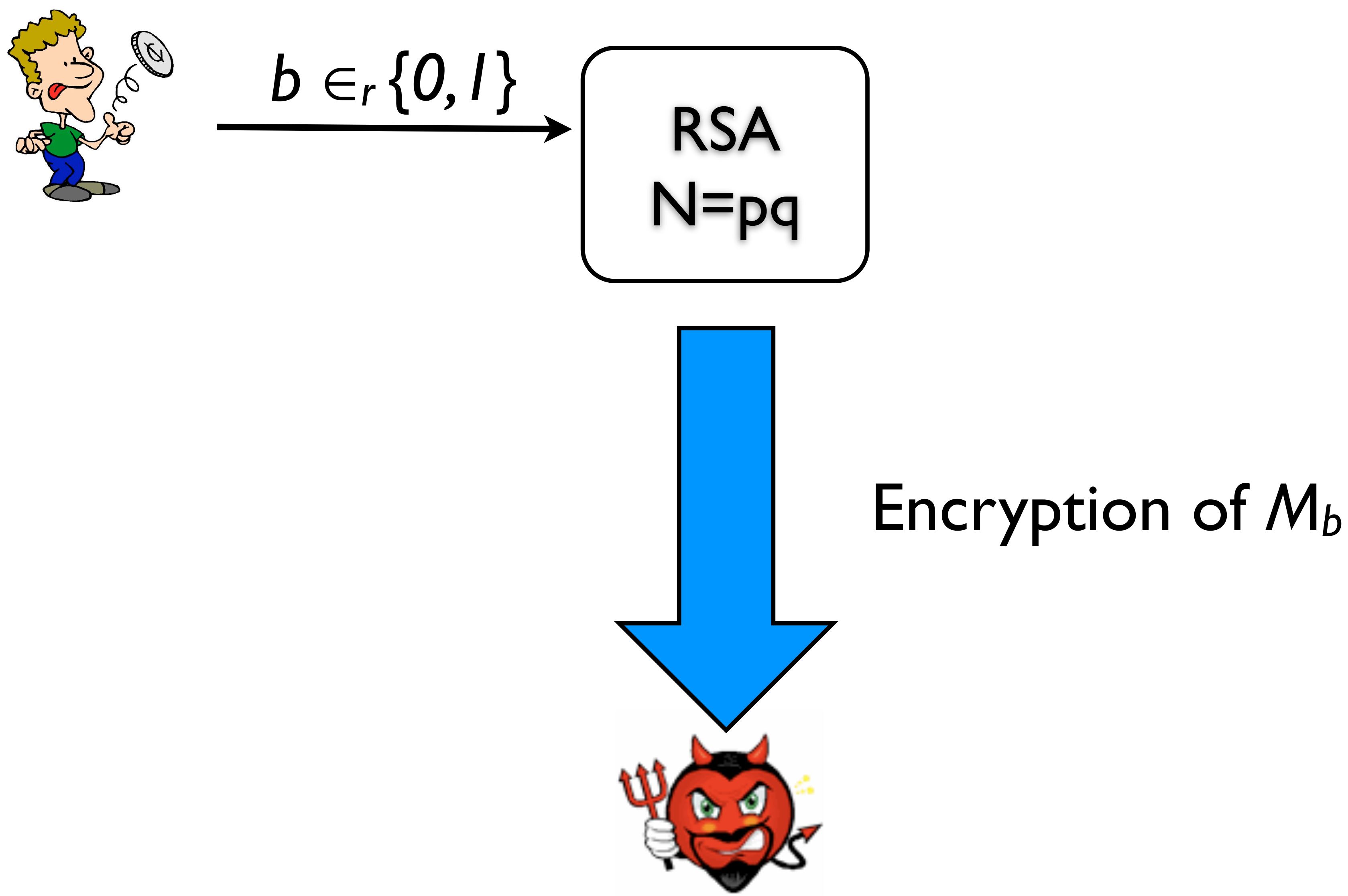
Crypto in Real Life



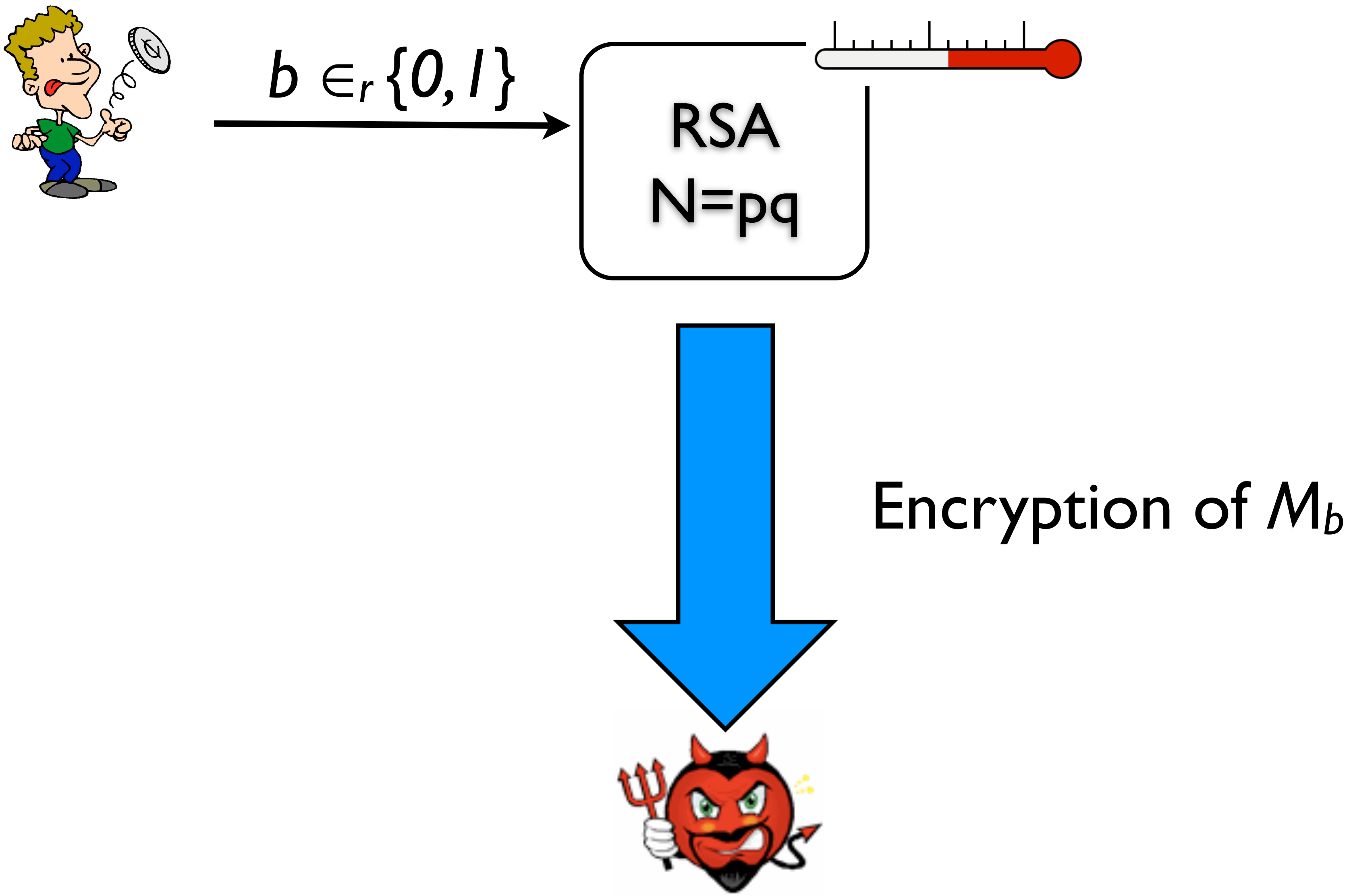
Crypto in Real Life



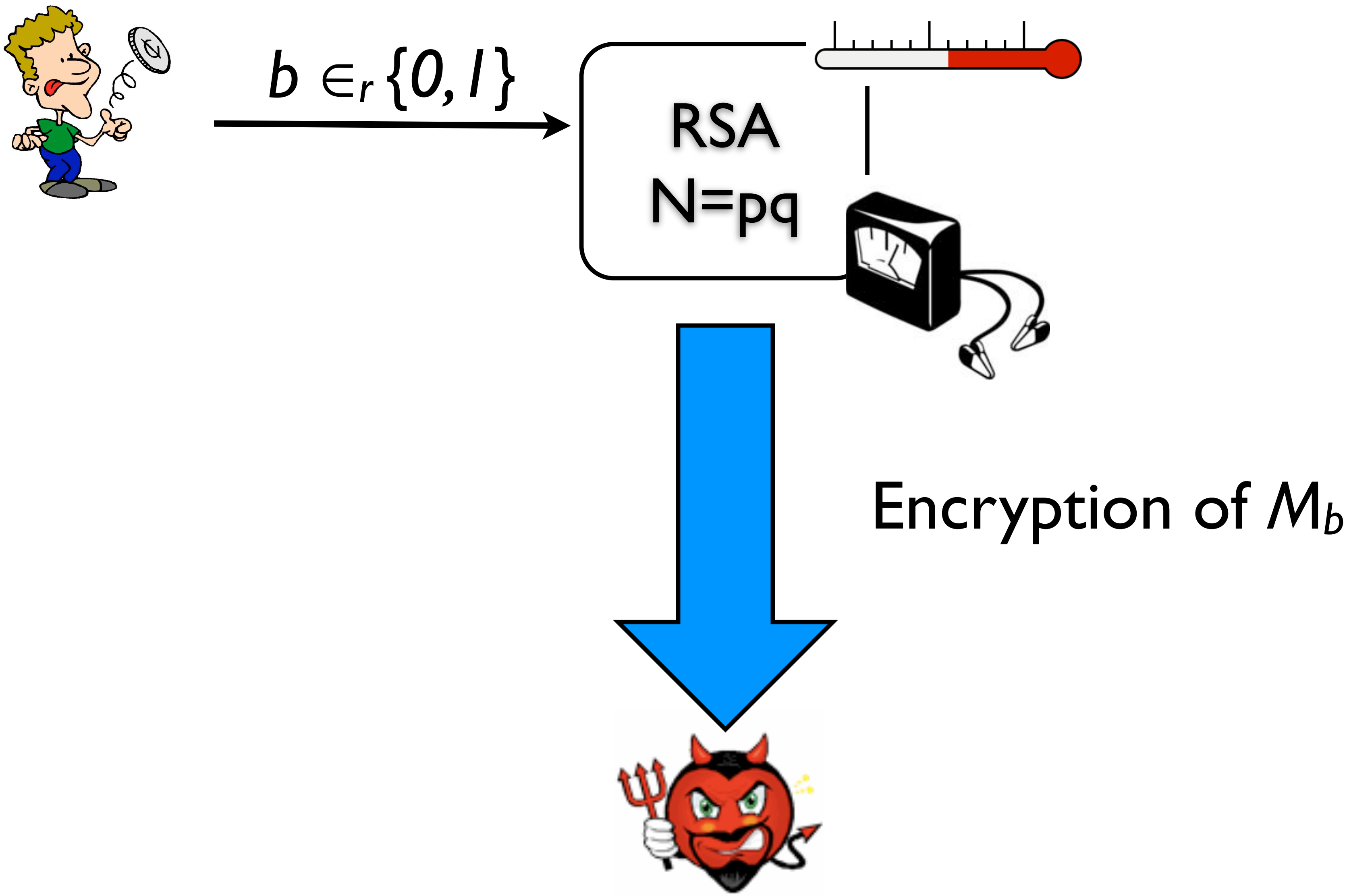
Crypto in Real Life



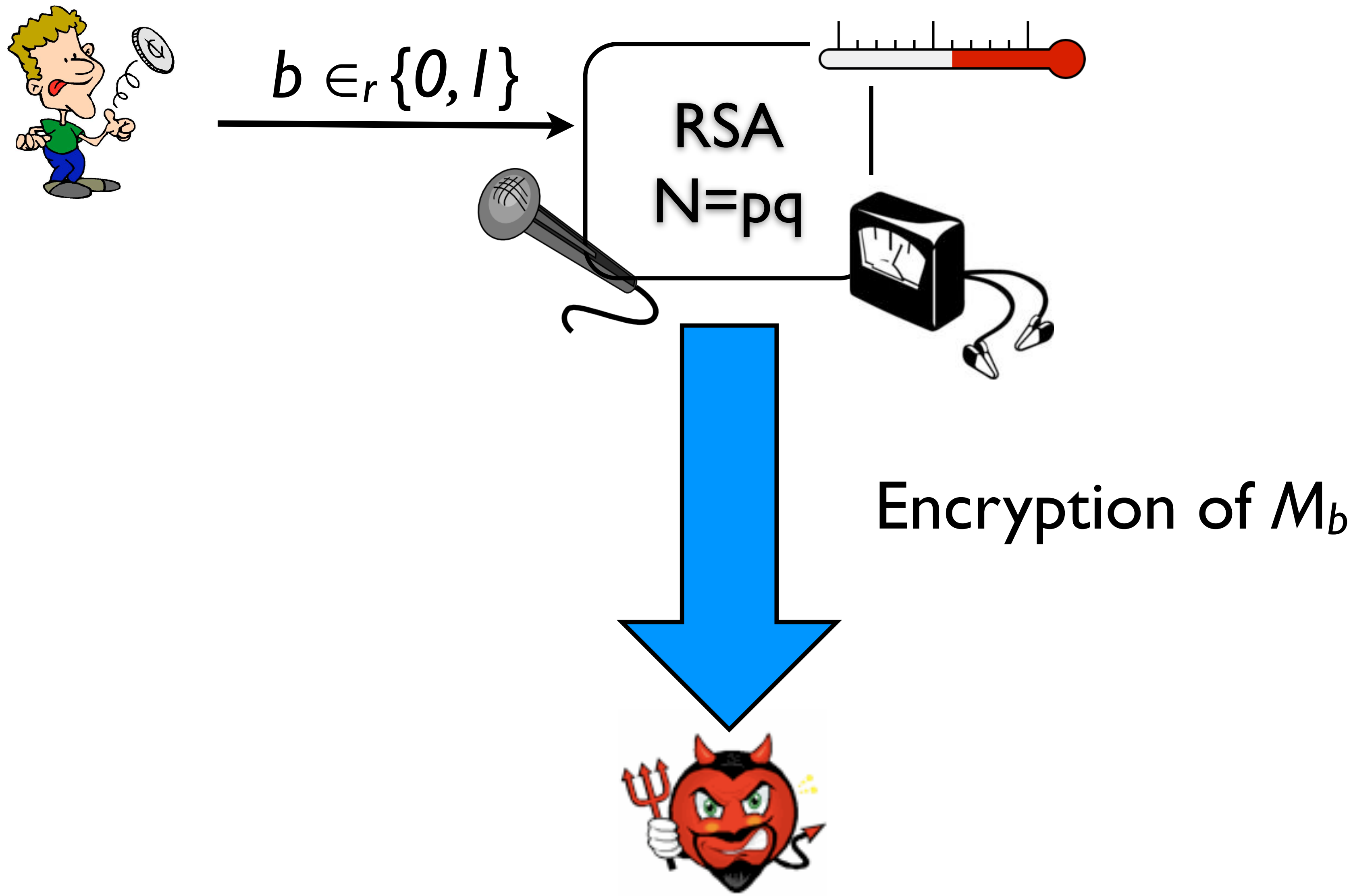
Crypto in Real Life



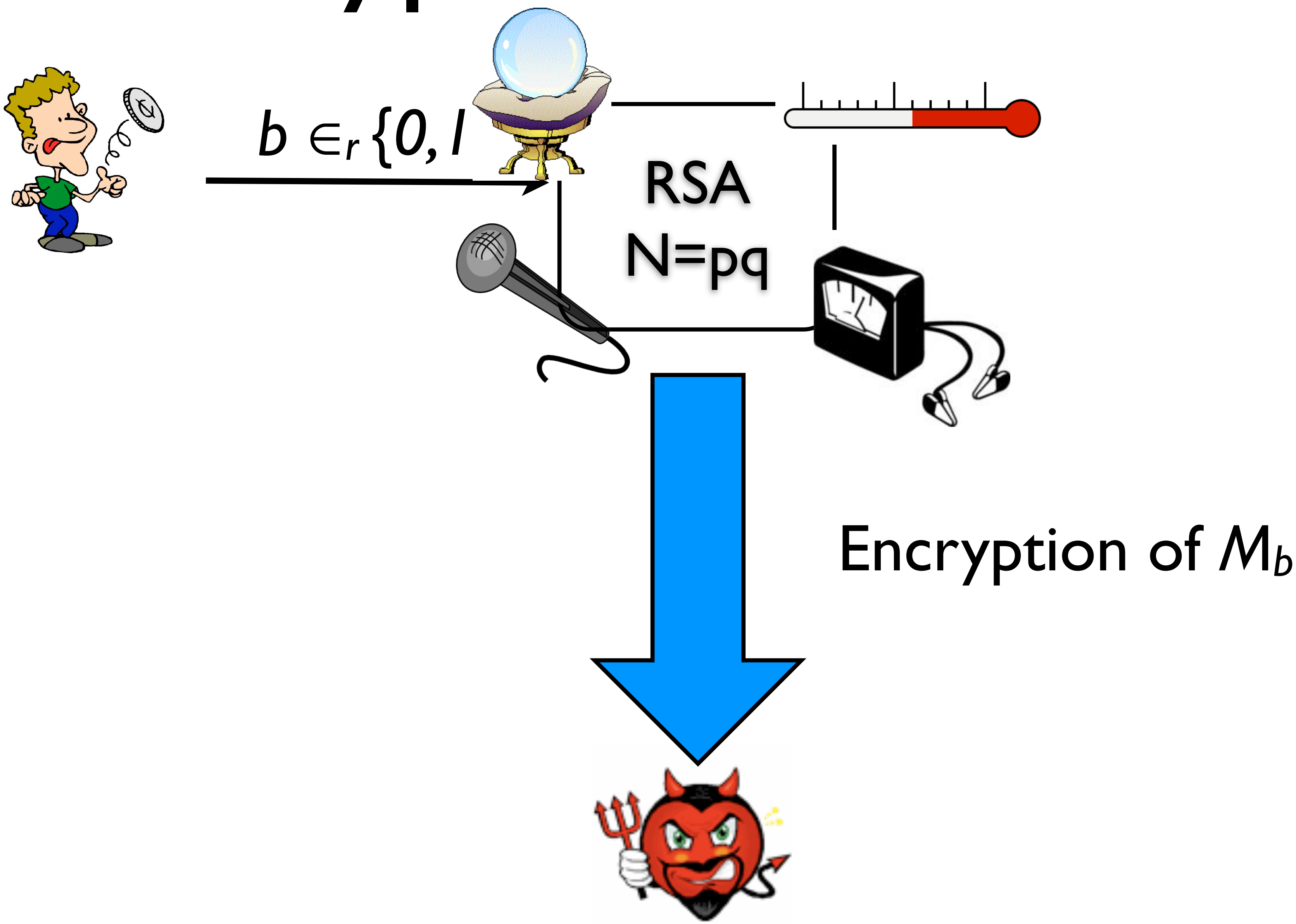
Crypto in Real Life



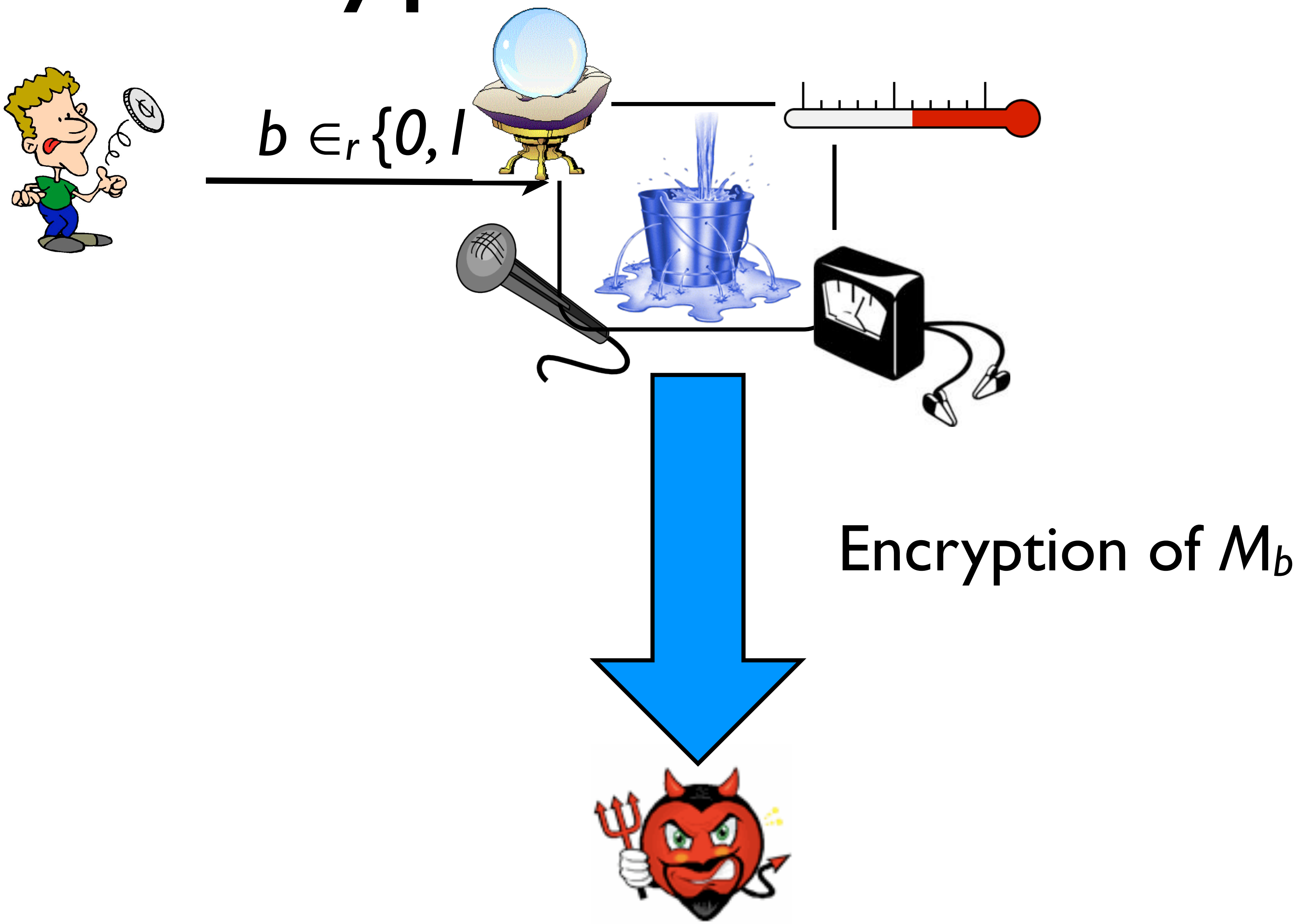
Crypto in Real Life



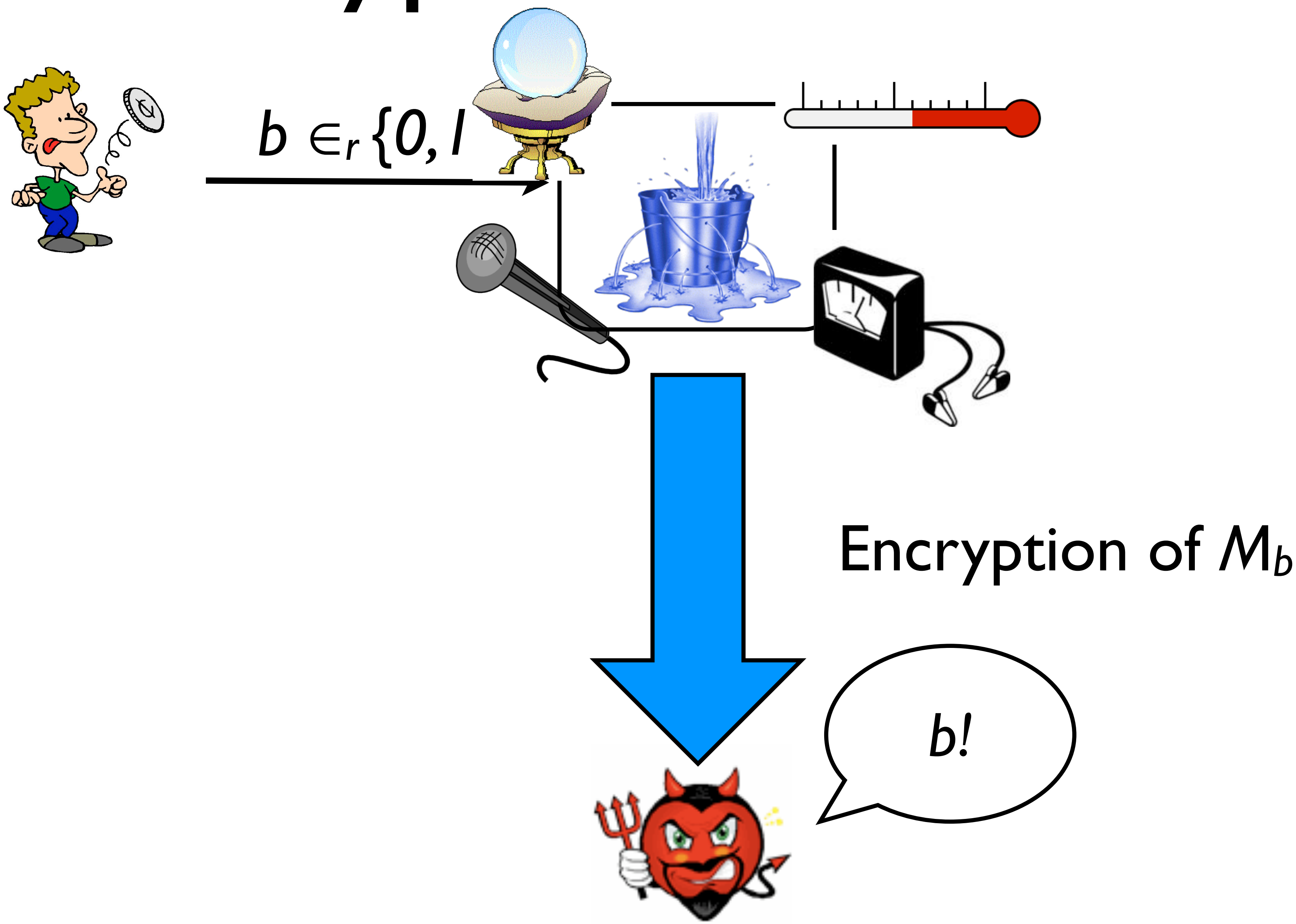
Crypto in Real Life



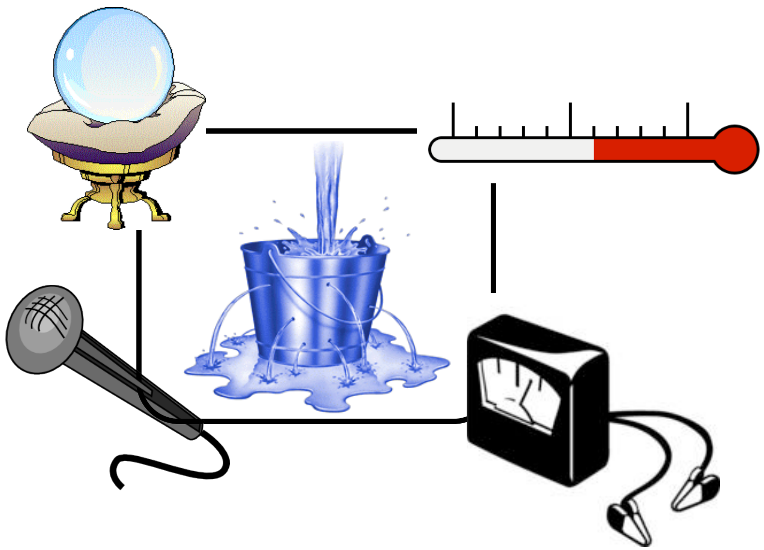
Crypto in Real Life



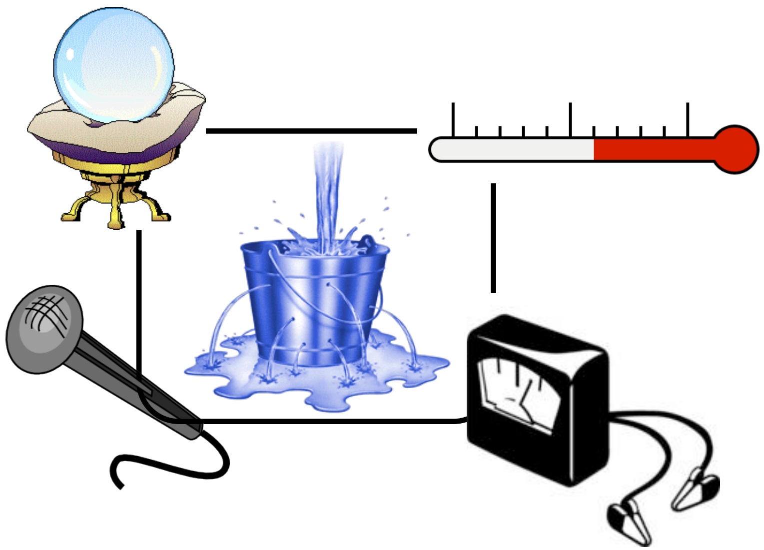
Crypto in Real Life



Crypto in Real Life

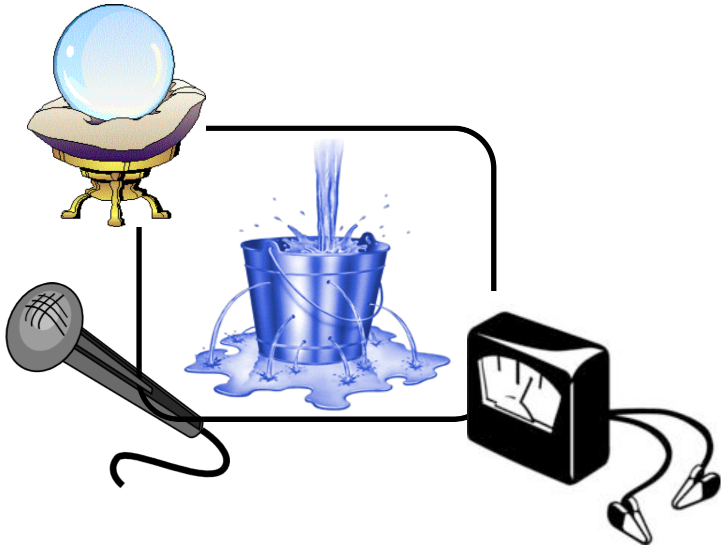


Crypto in Real Life



Solutions:

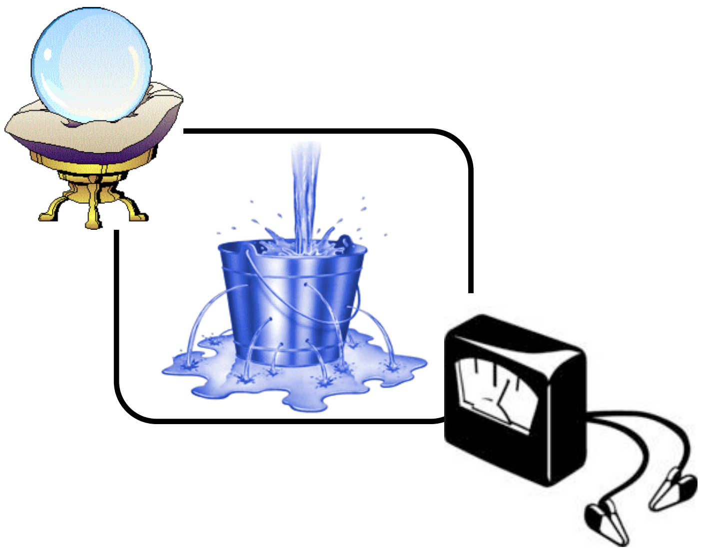
Crypto in Real Life



Solutions:

- Heat isolation

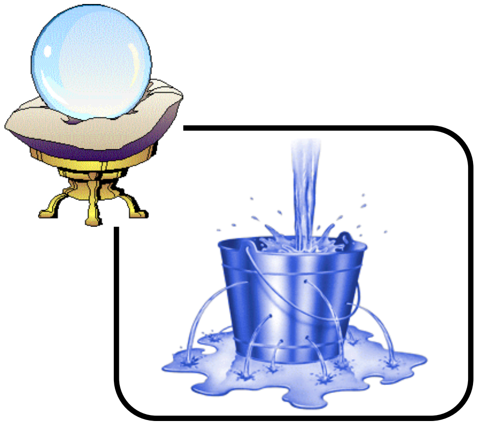
Crypto in Real Life



Solutions:

- Heat isolation
- Uniform cooling and quite fans

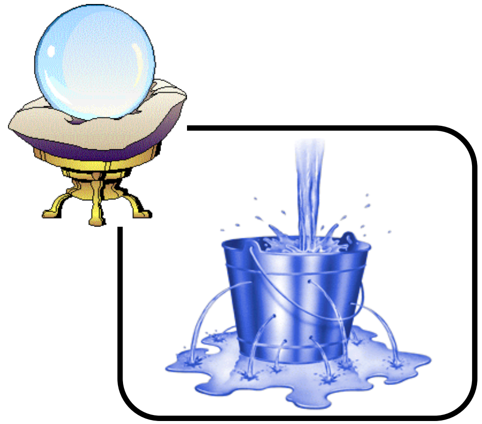
Crypto in Real Life



Solutions:

- Heat isolation
- Uniform cooling and quite fans
- Electromagnetic isolation

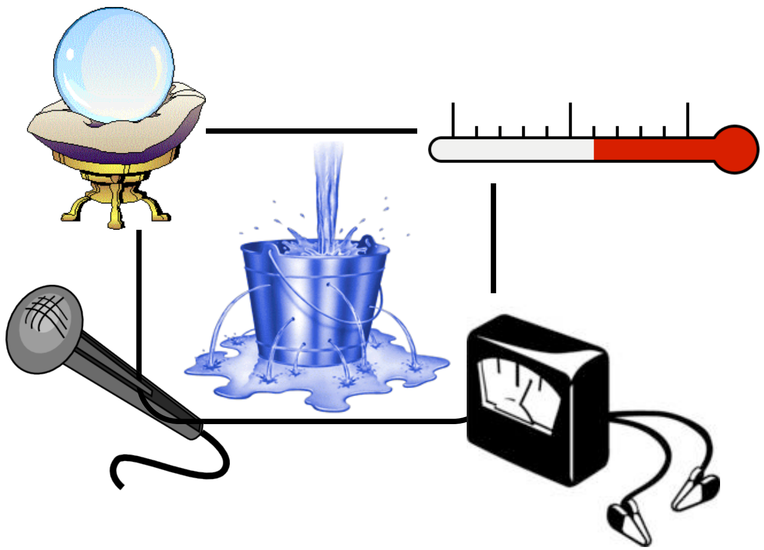
Crypto in Real Life



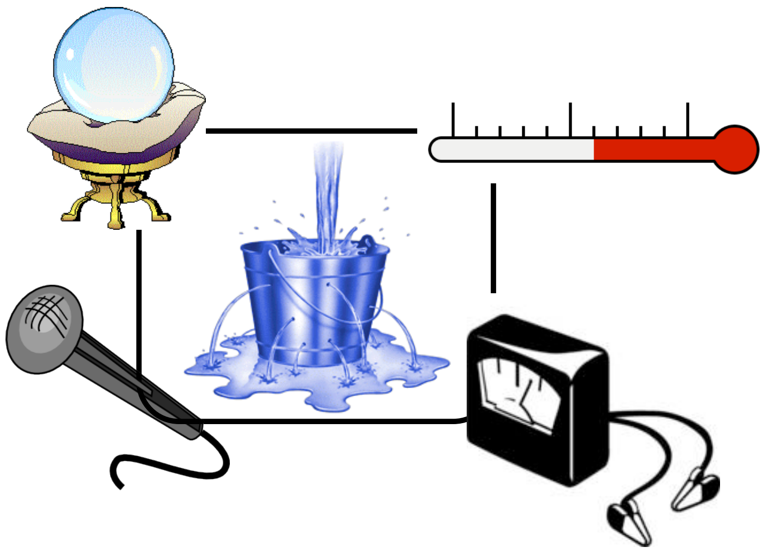
Solutions:

- Heat isolation
- Uniform cooling and quite fans
- Electromagnetic isolation
- **???**

Algorithmic Protection

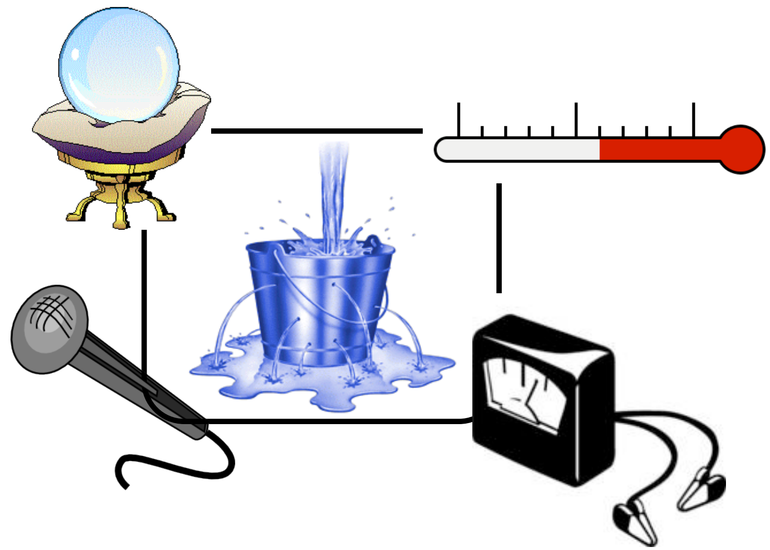


Algorithmic Protection



In recent years:

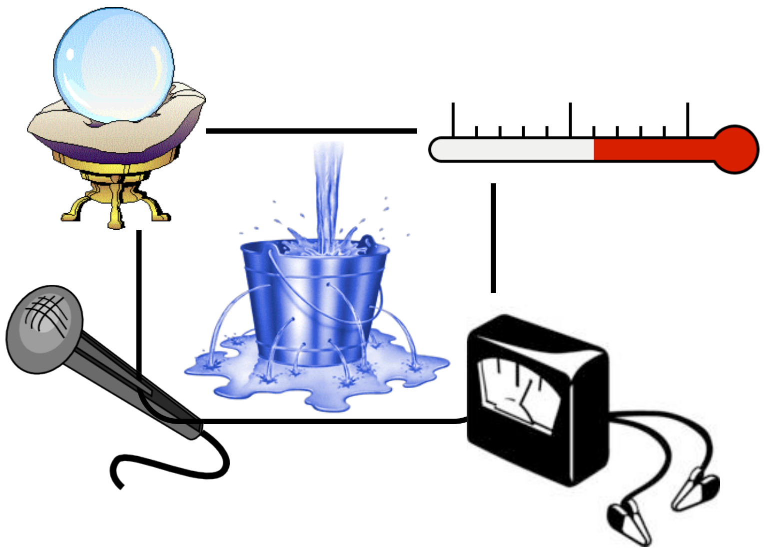
Algorithmic Protection



In recent years:

- Reduce assumptions about hardware

Algorithmic Protection



In recent years:

- Reduce assumptions about hardware
- Design algorithms that are secure under leakage

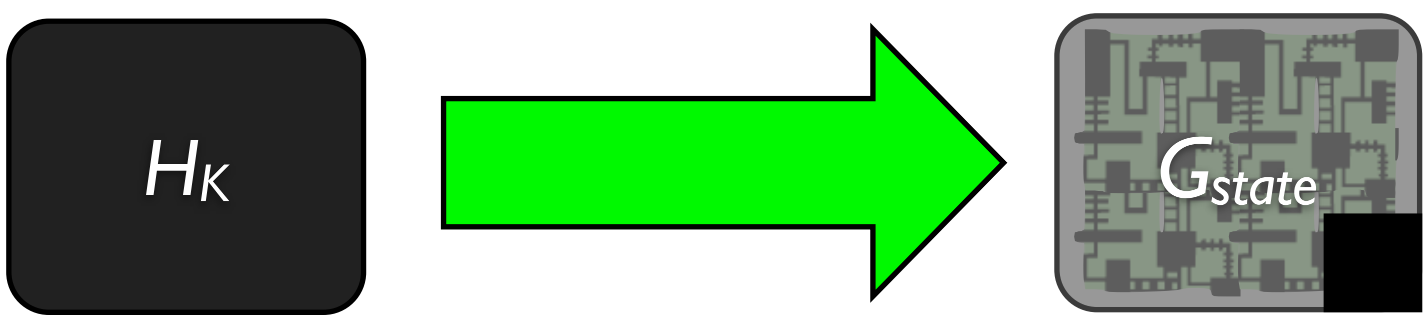
Previous Work

- [Goldreich & Ostrovsky 96] protect against complete leakage of memory when CPU is secure.
- [Ishai & Sahai & Wagner 2003] assume adversary leaks value of a fixed number of wires
- [Micali & Reyzin 04] introduce axioms and framework
- [Goldwasser & Kalai & Rothblum 2008] one-time programs
- [Dziembowski & Pietrzak 2008] Leakage resilient stream cipher in the split state model
- [Faust & Reyzin & Tromer 2009] protect against AC0 leakage functions (needs secure component)

Our Results

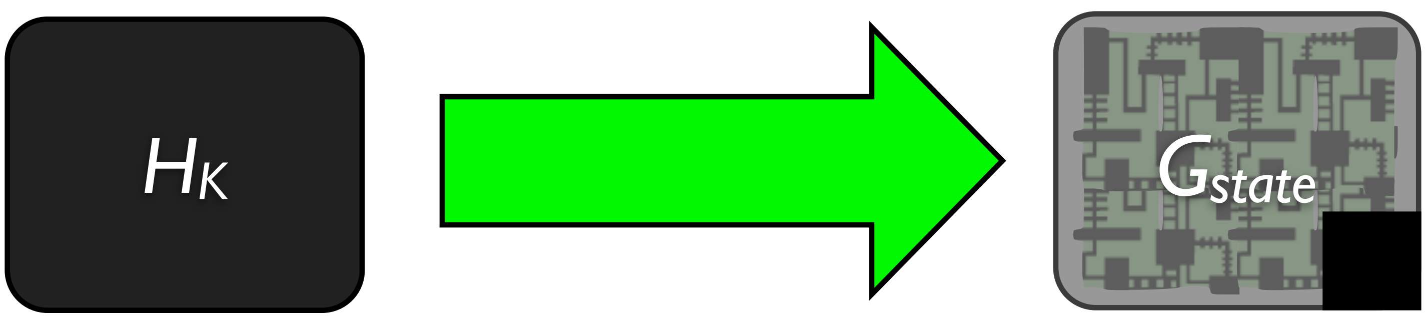
Our Results

- A compiler that transforms any keyed primitive H_K into a stateful algorithm G_{state}
- G_{state} is resilient against length bounded leakage in each invocation
- Need a fixed size, memory-less secure component



Our Results

- A compiler that transforms any keyed primitive H_K into a stateful algorithm G_{state}
- G_{state} is resilient against length bounded leakage in each invocation
- Need a fixed size, memory-less secure component

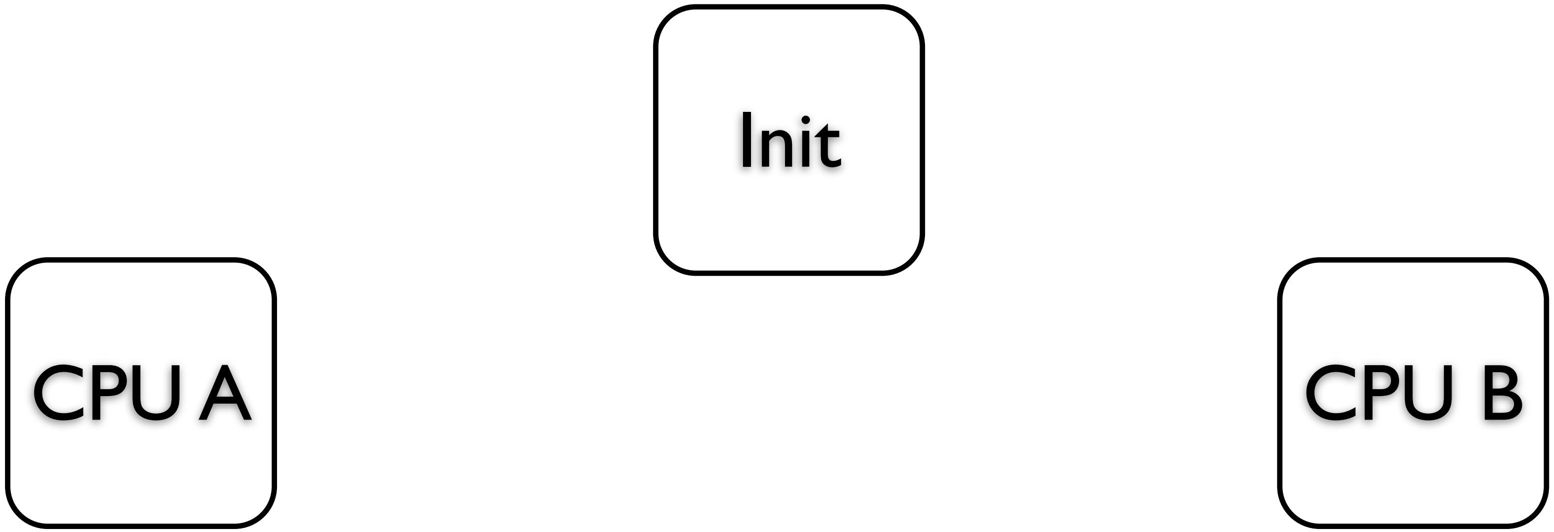


Achieve leak-resilience for arbitrary complexity from leak-resilience for fixed complexity

Model

Want to protect $H_K(x)$

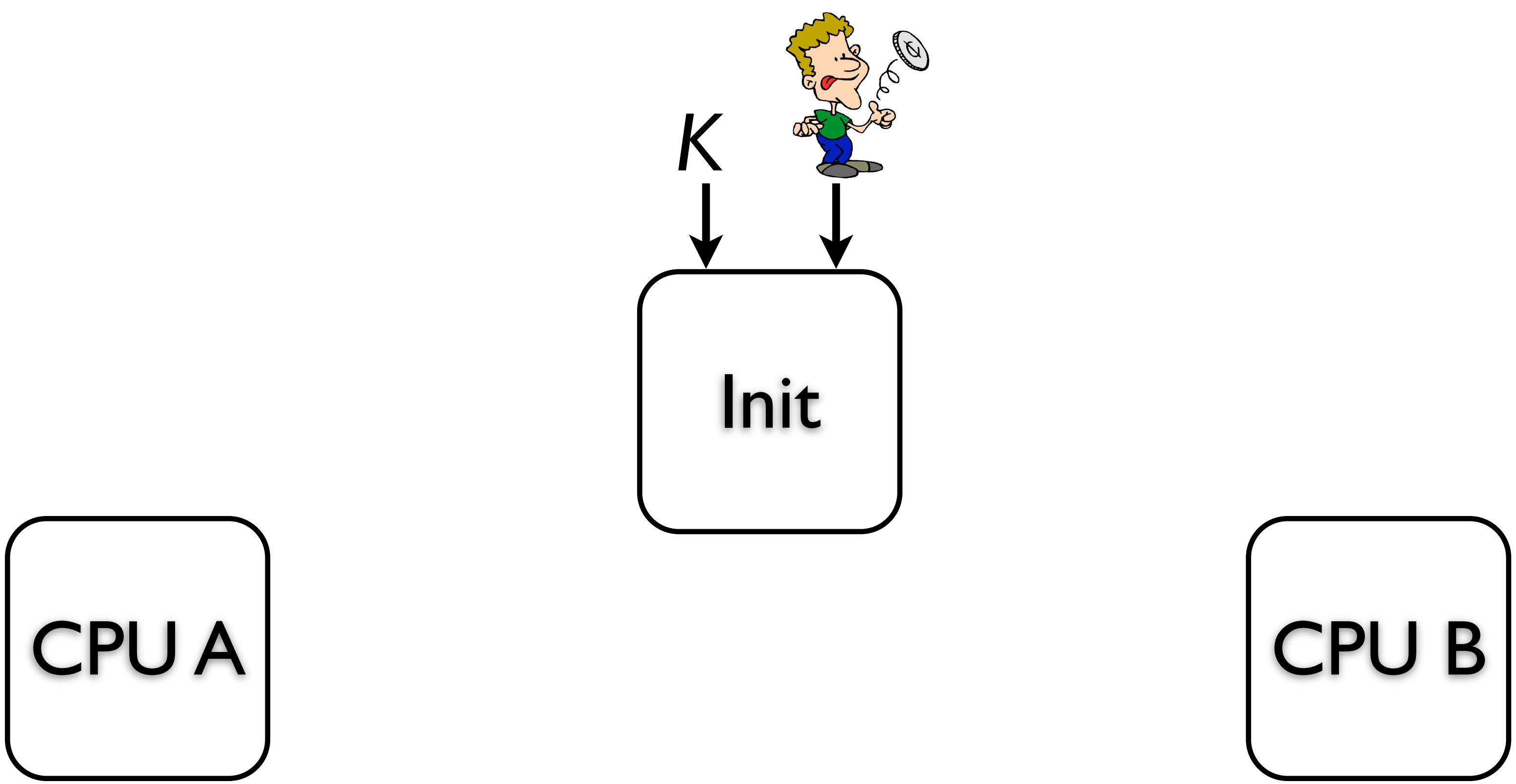
Two computers that communicate over a *public* channel. Initialization is secure.



Model

Want to protect $H_K(x)$

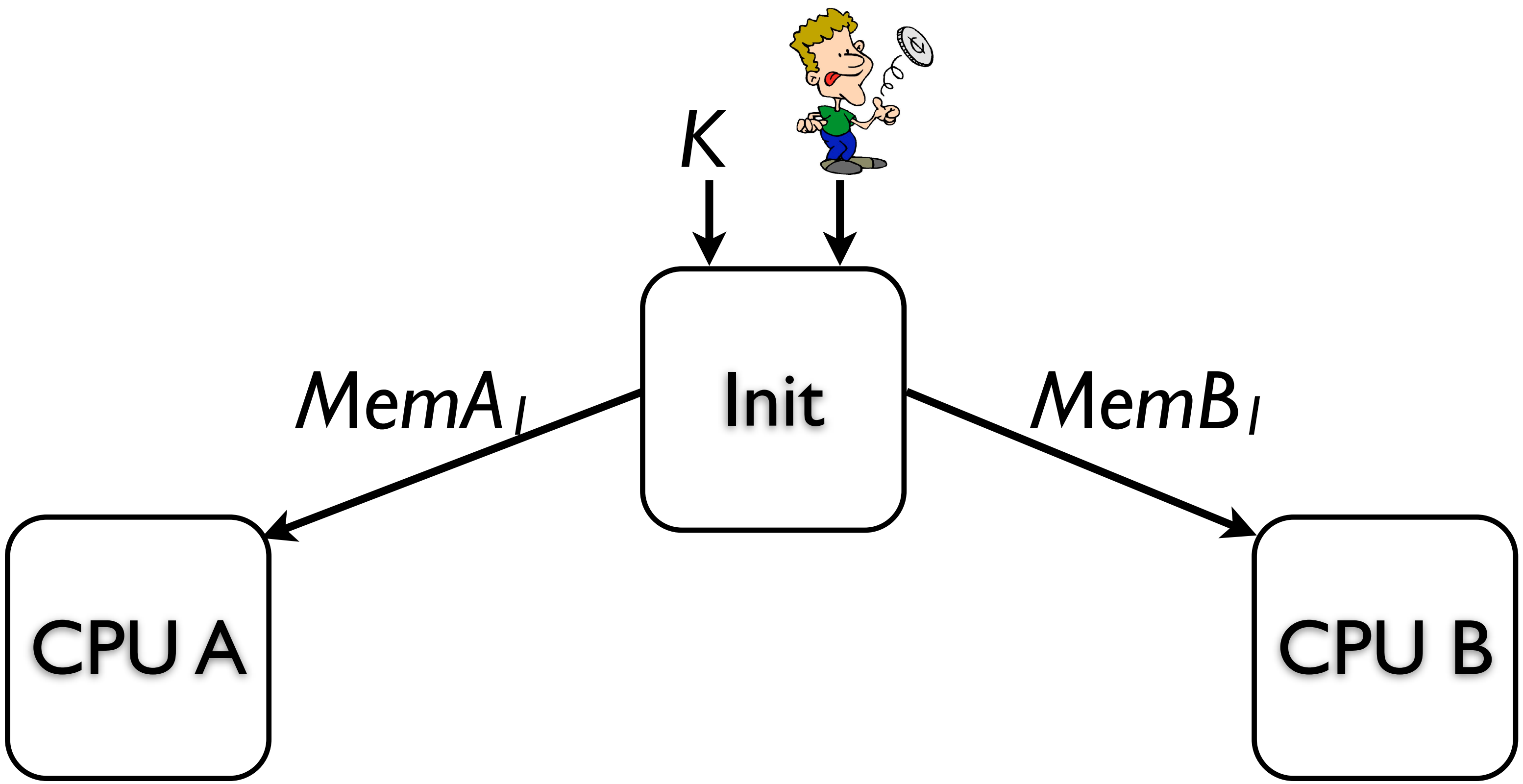
Two computers that communicate over a *public* channel. Initialization is secure.



Model

Want to protect $H_K(x)$

Two computers that communicate over a *public* channel. Initialization is secure.

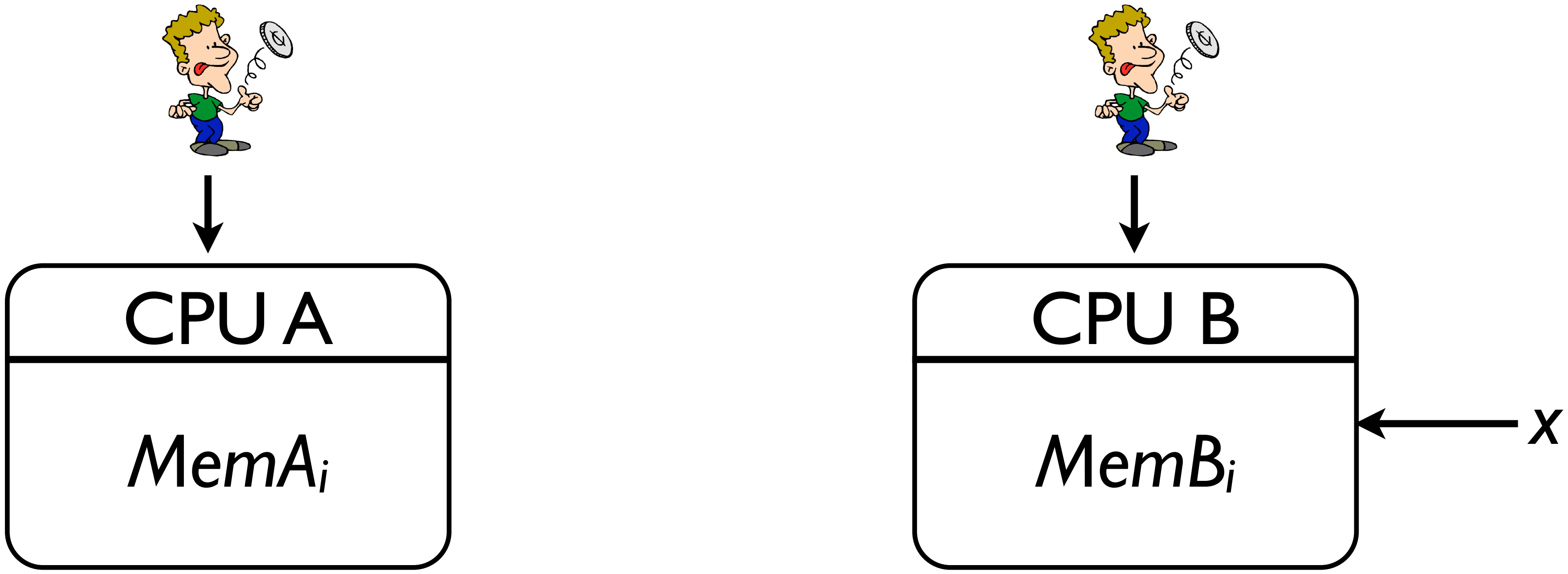


Model

Evaluating $H_K(x)$

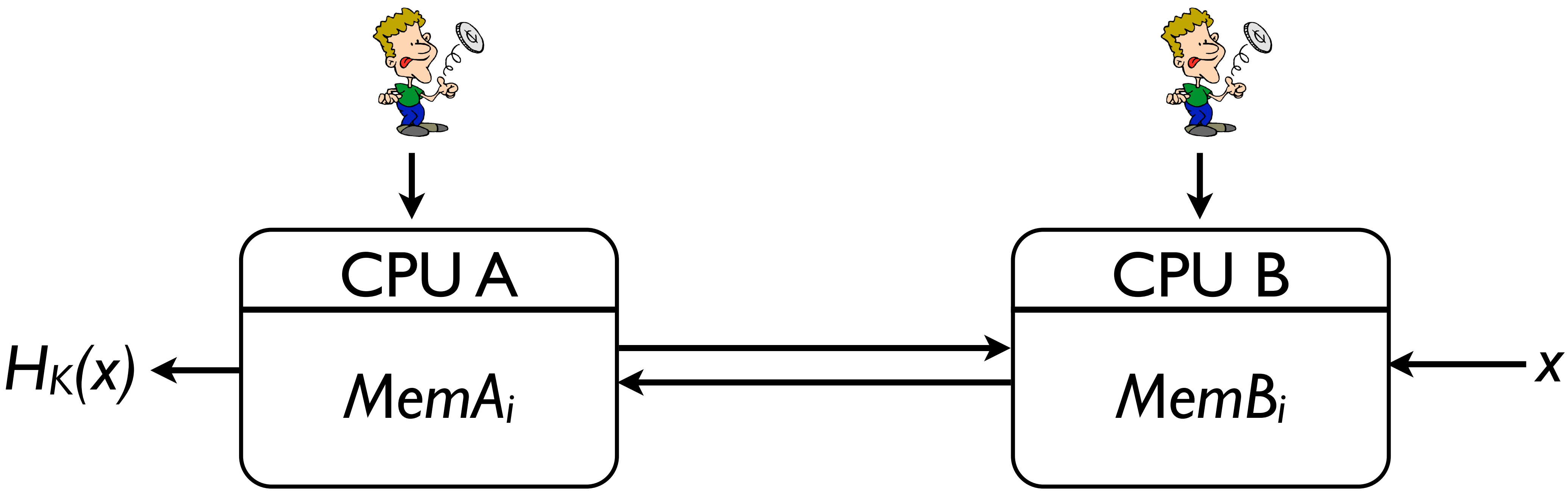
Model

Evaluating $H_K(x)$



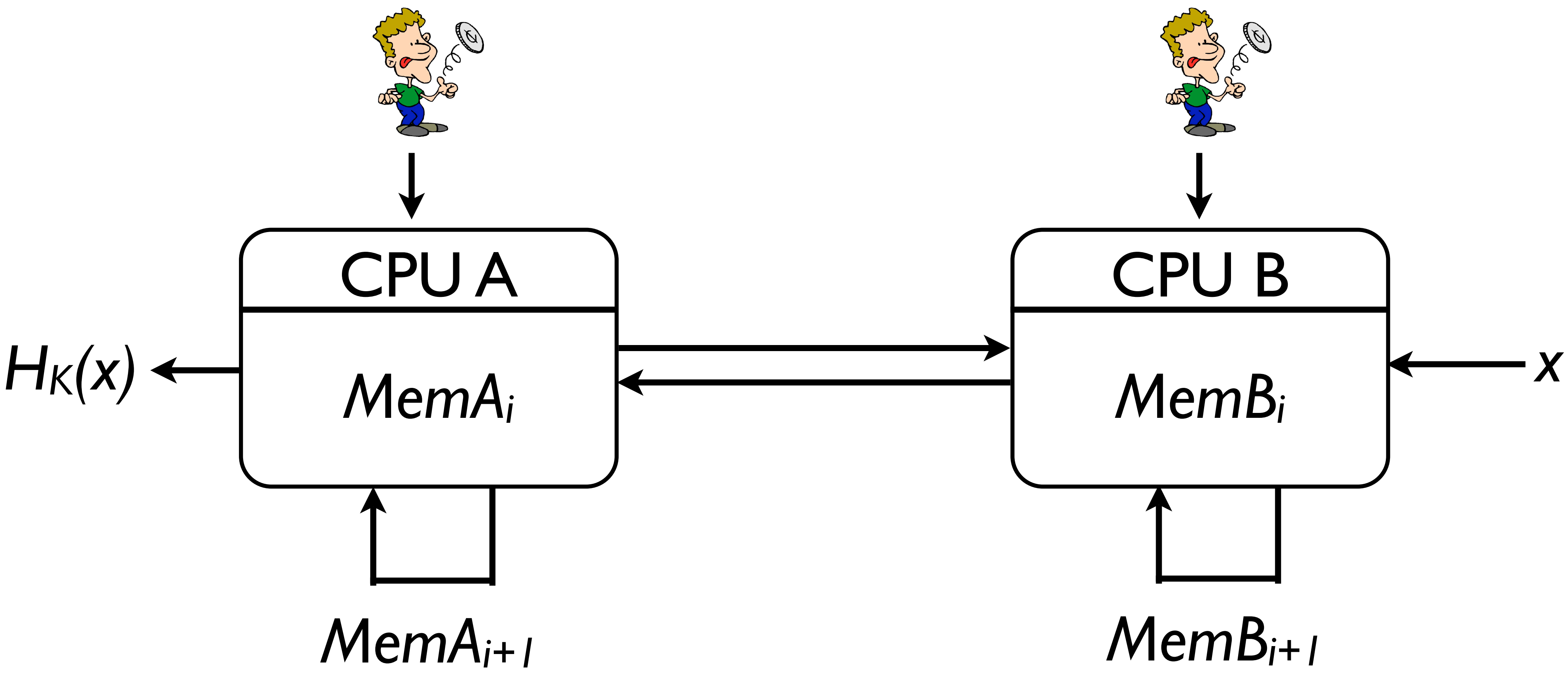
Model

Evaluating $H_K(x)$

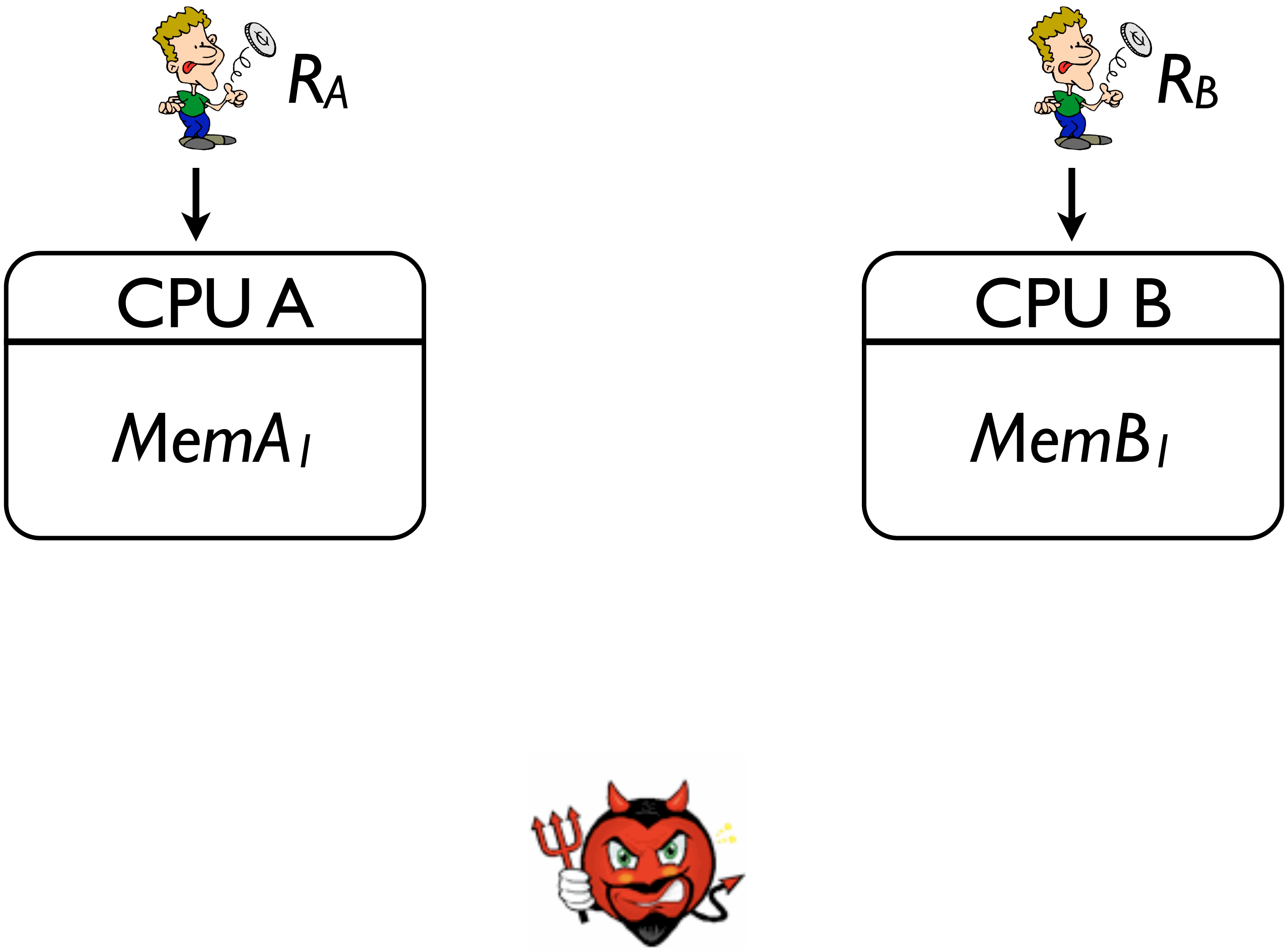


Model

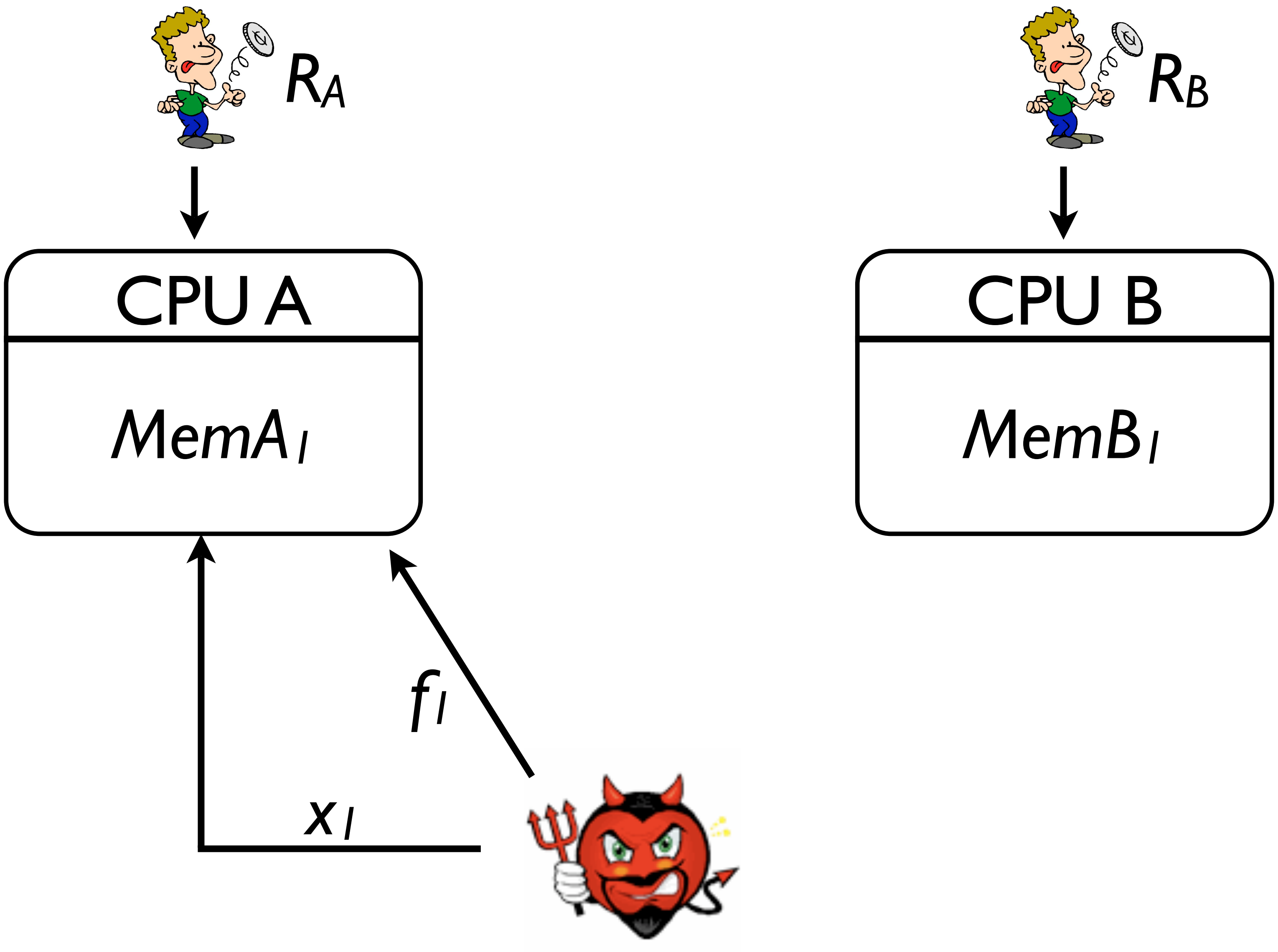
Evaluating $H_K(x)$



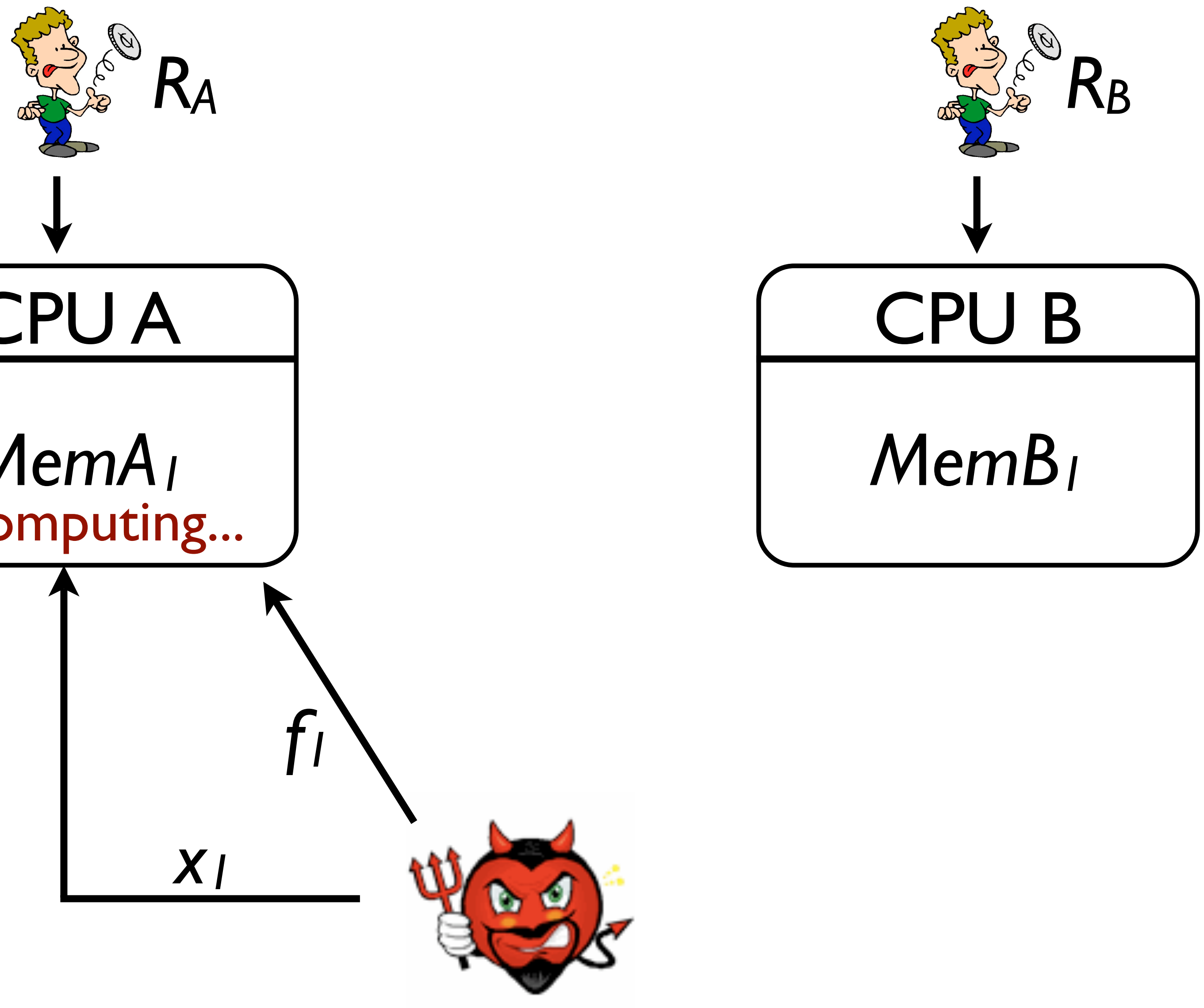
Leakage



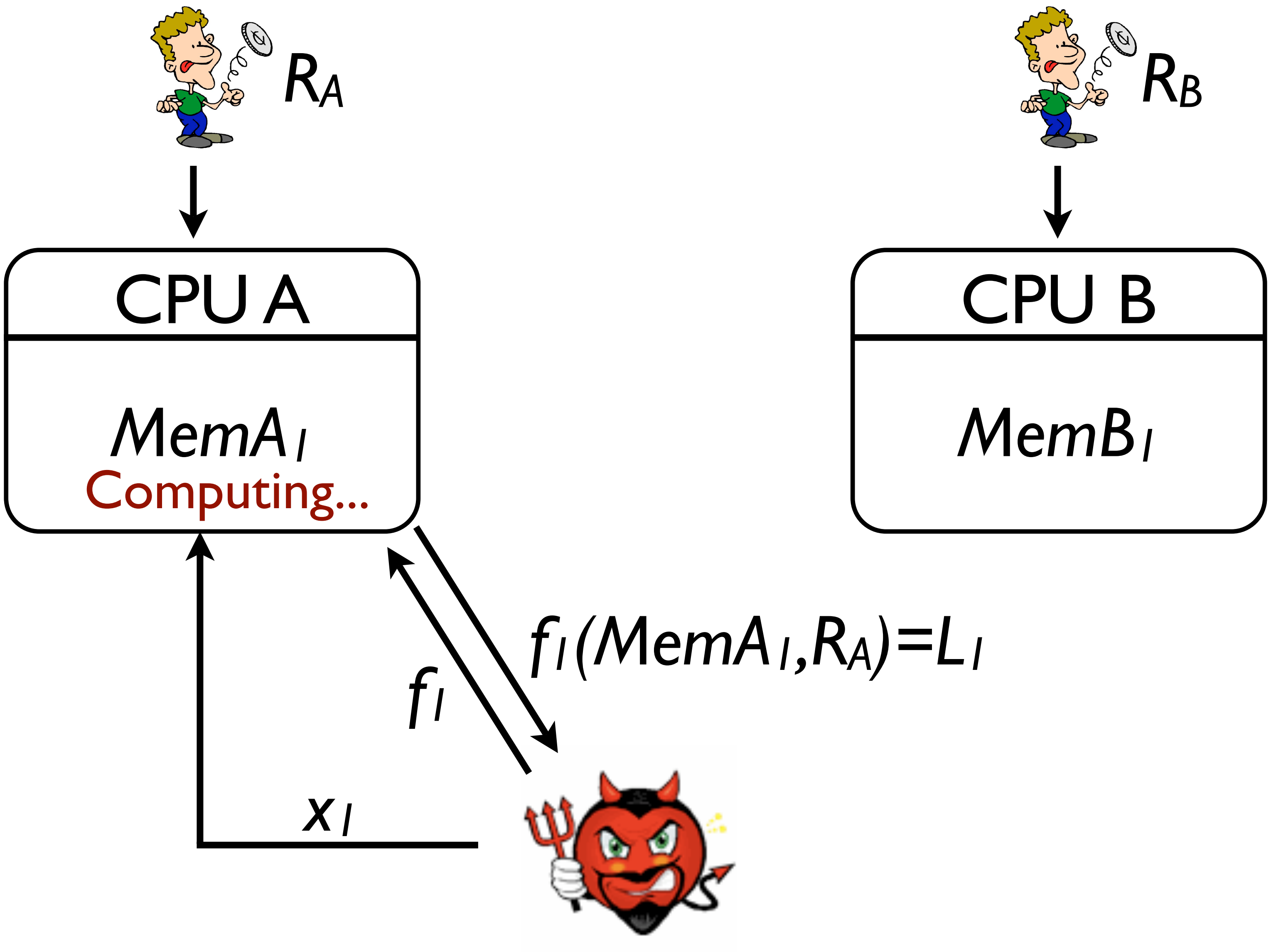
Leakage



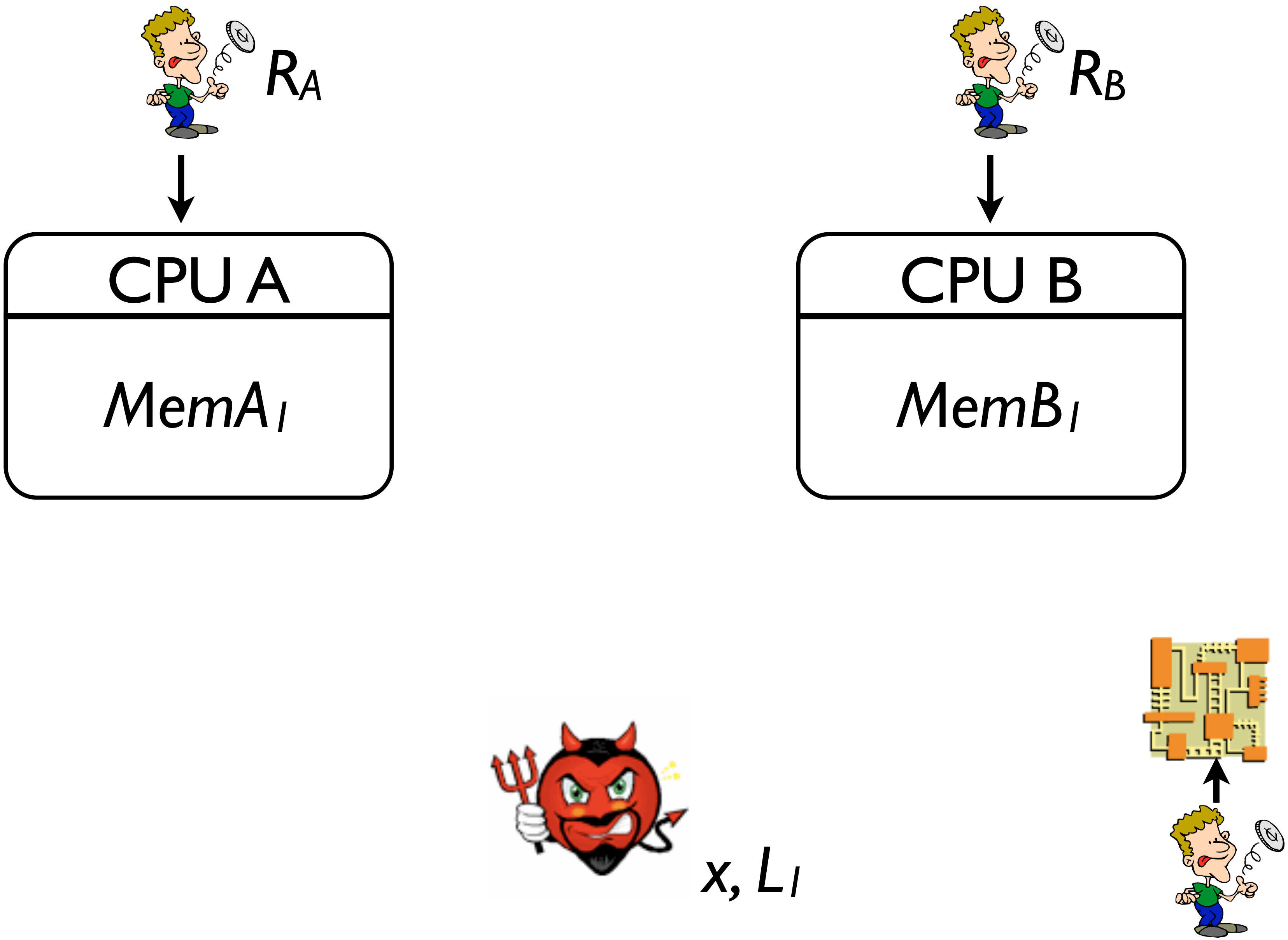
Leakage



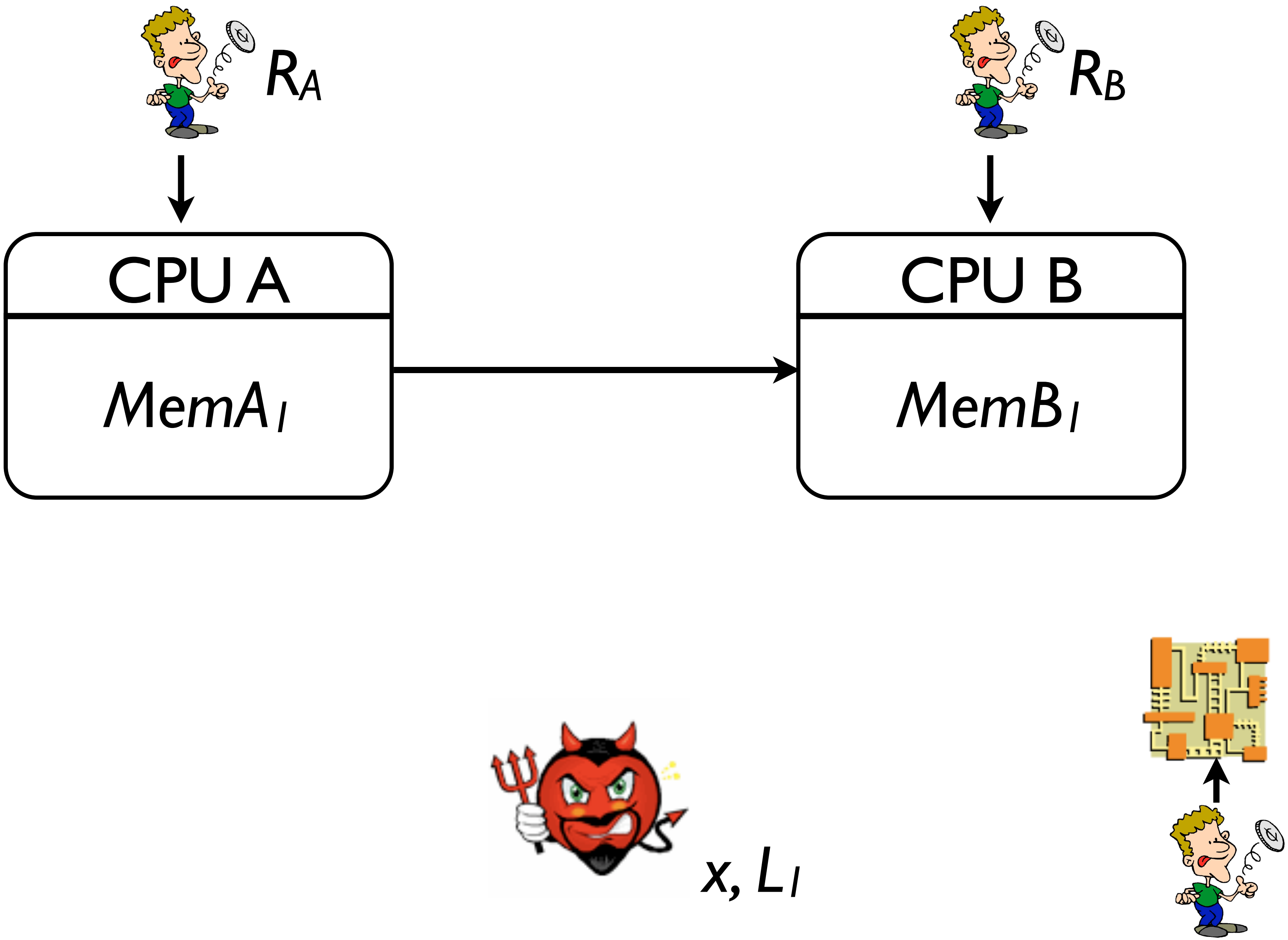
Leakage



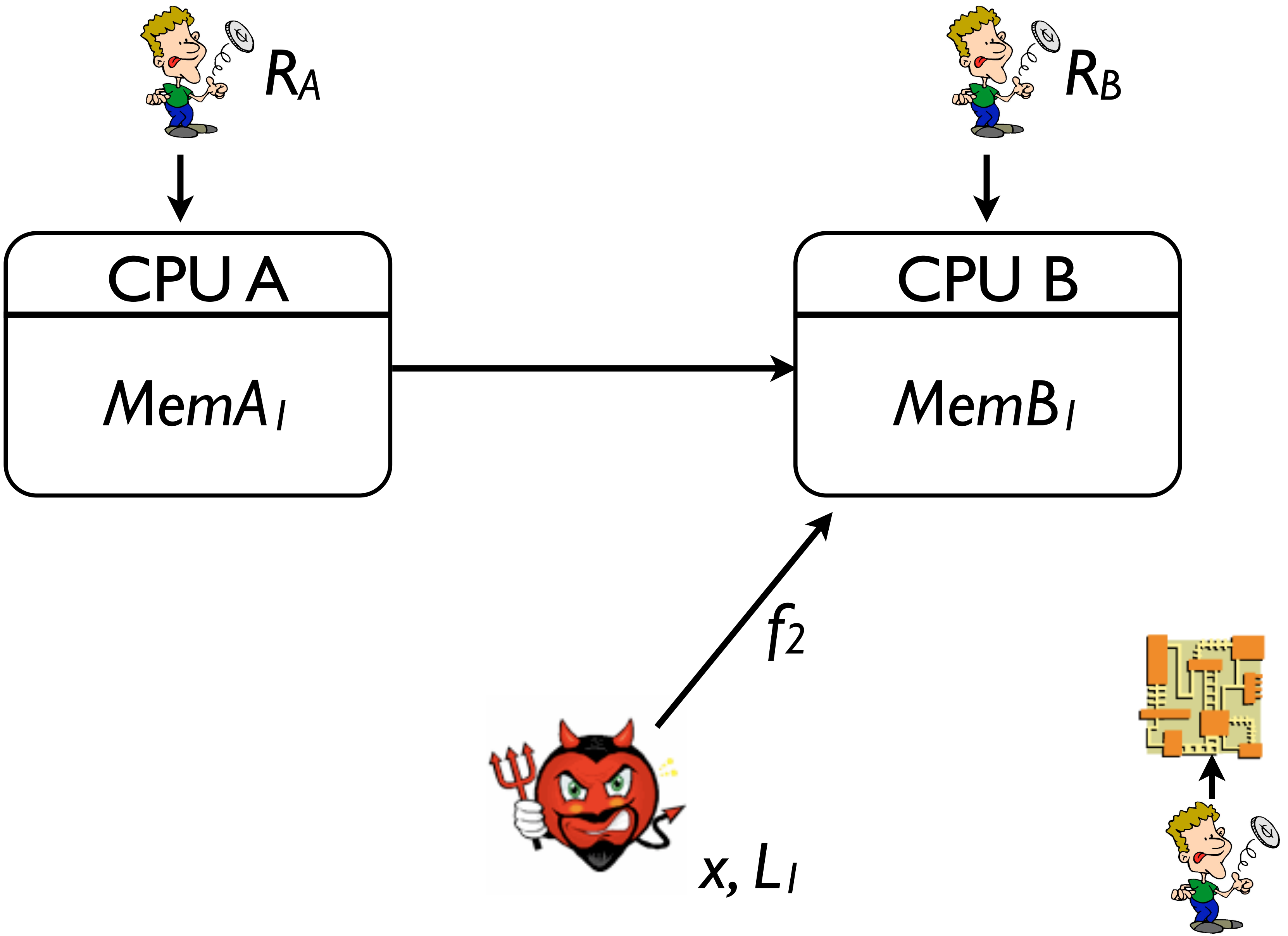
Leakage



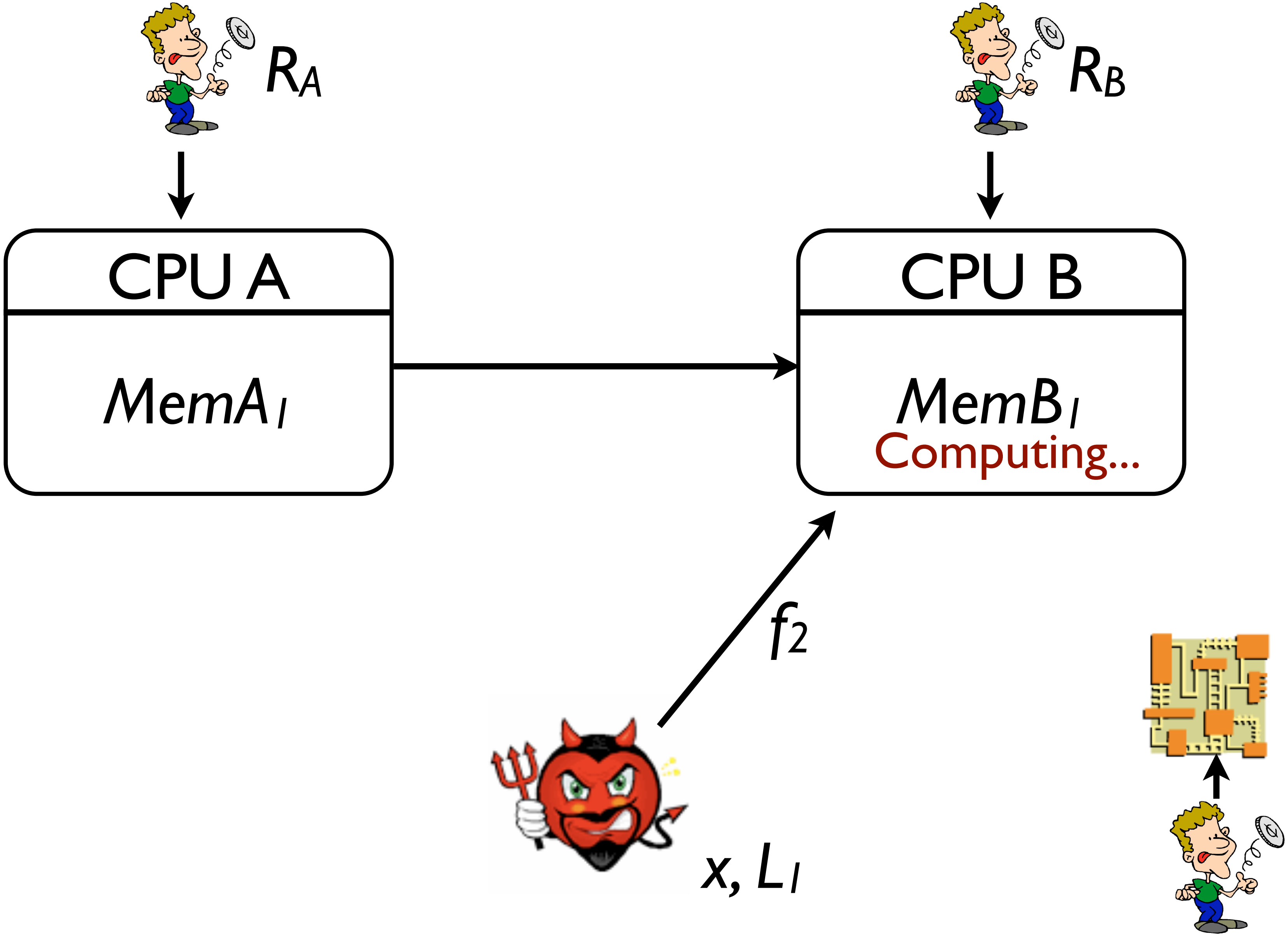
Leakage



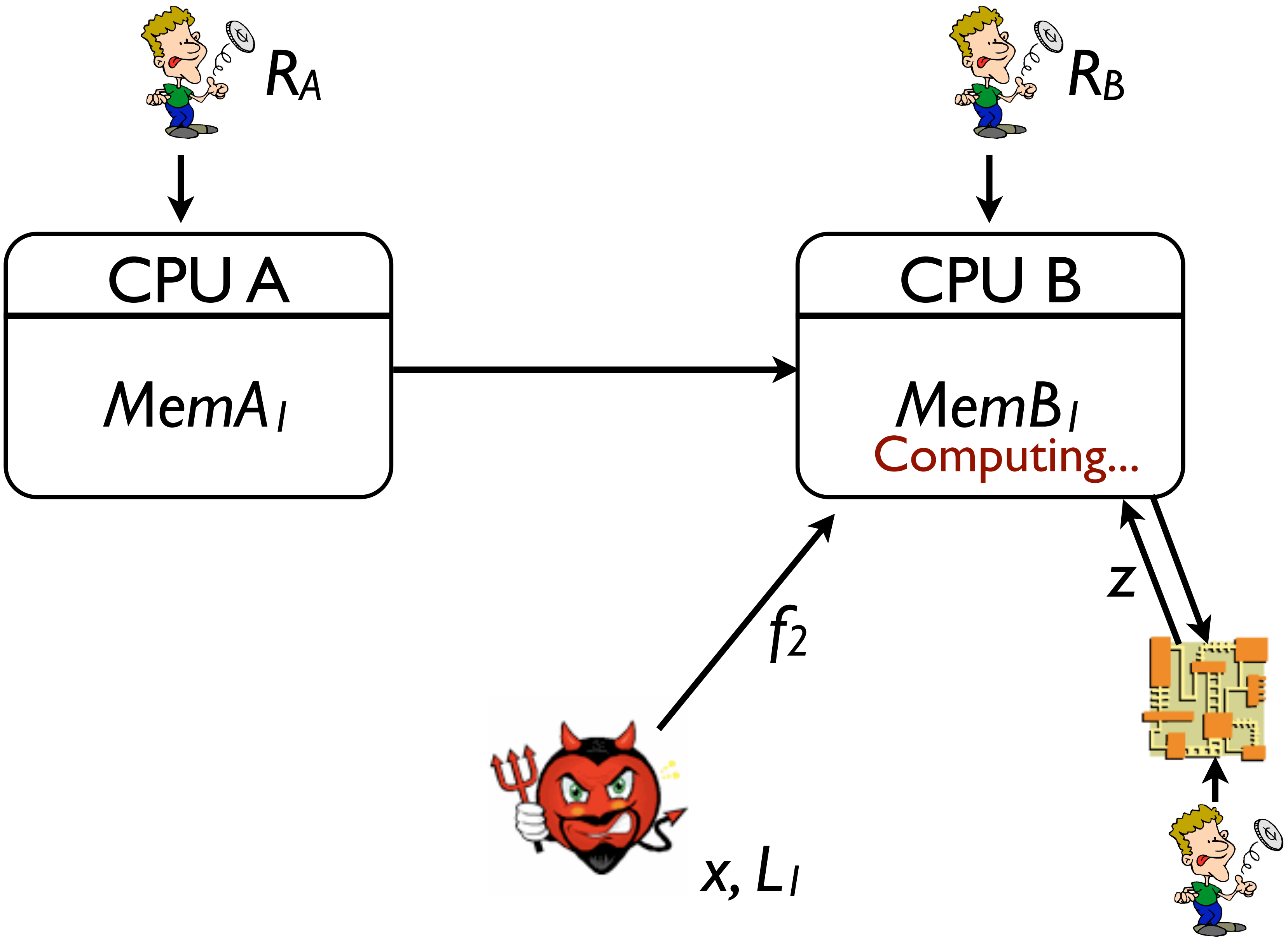
Leakage



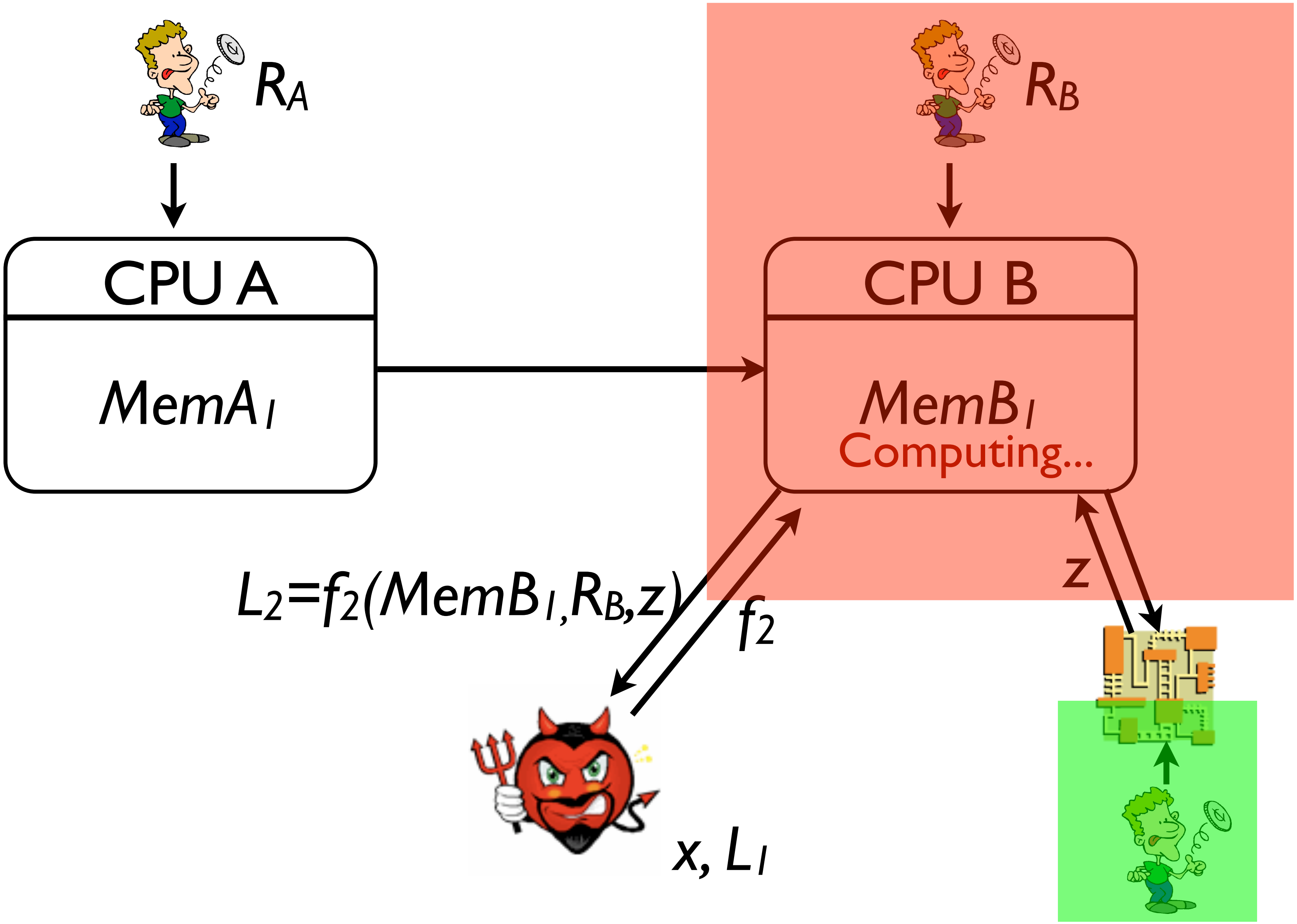
Leakage



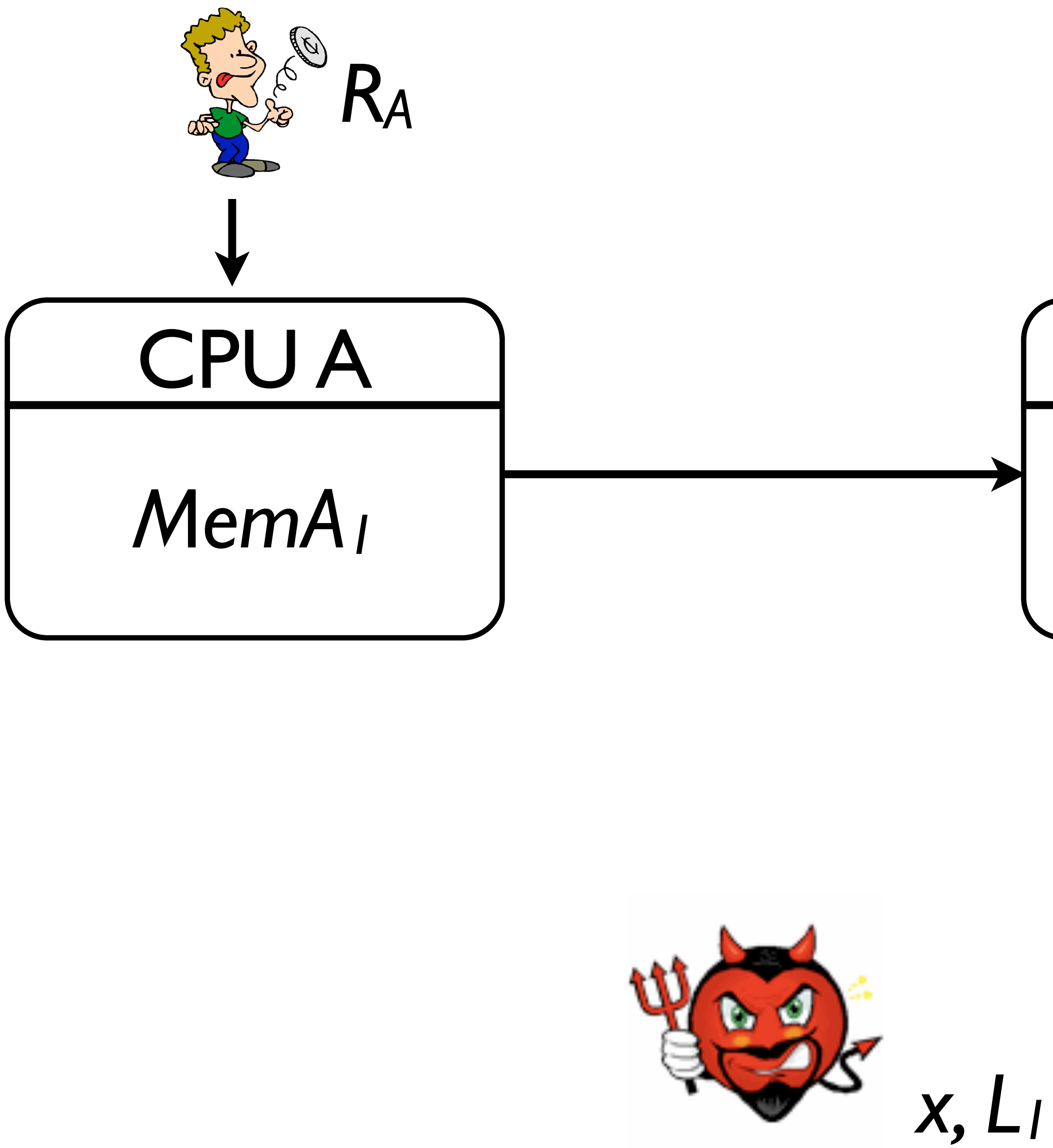
Leakage



Leakage

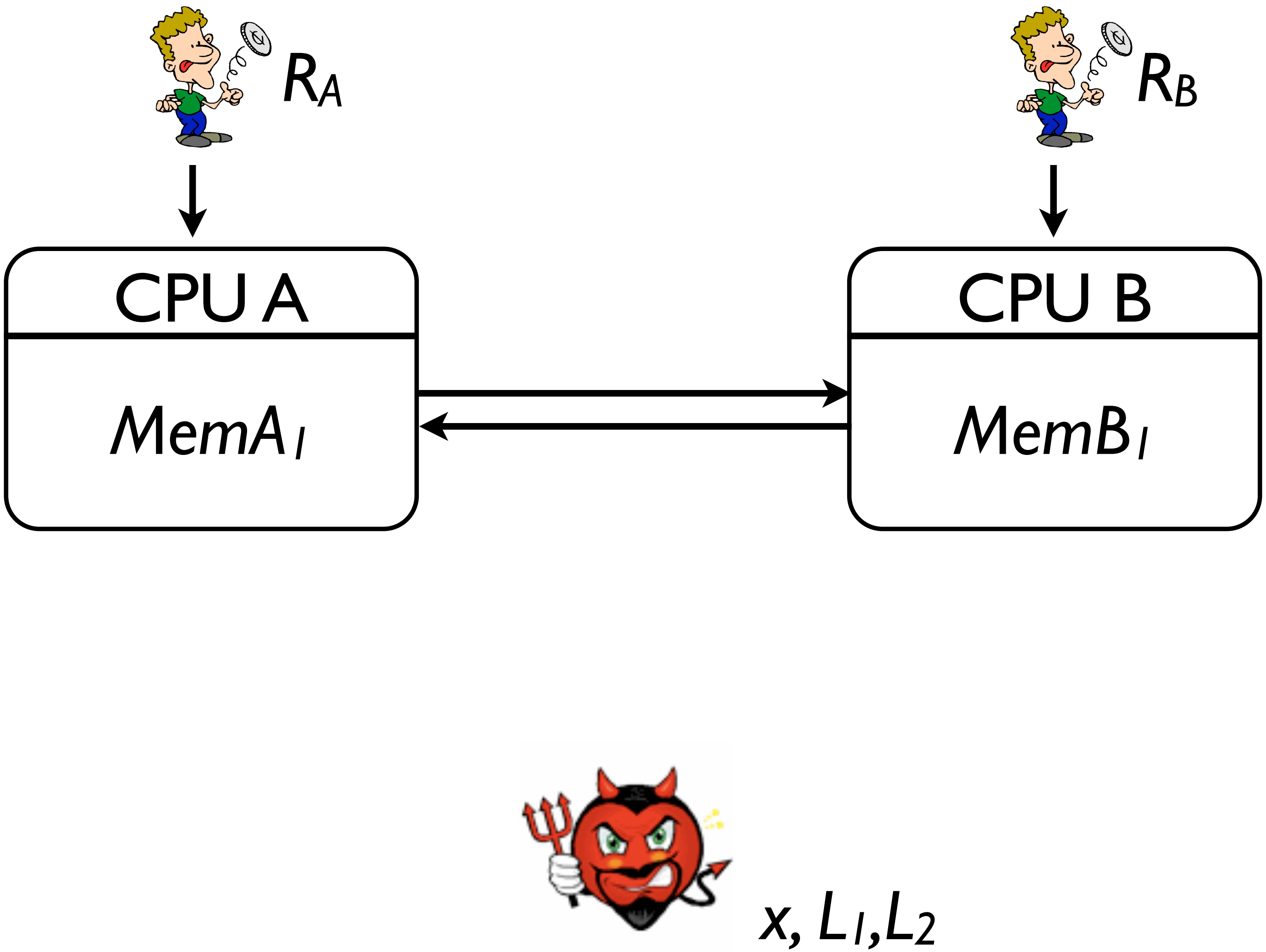


Leakage



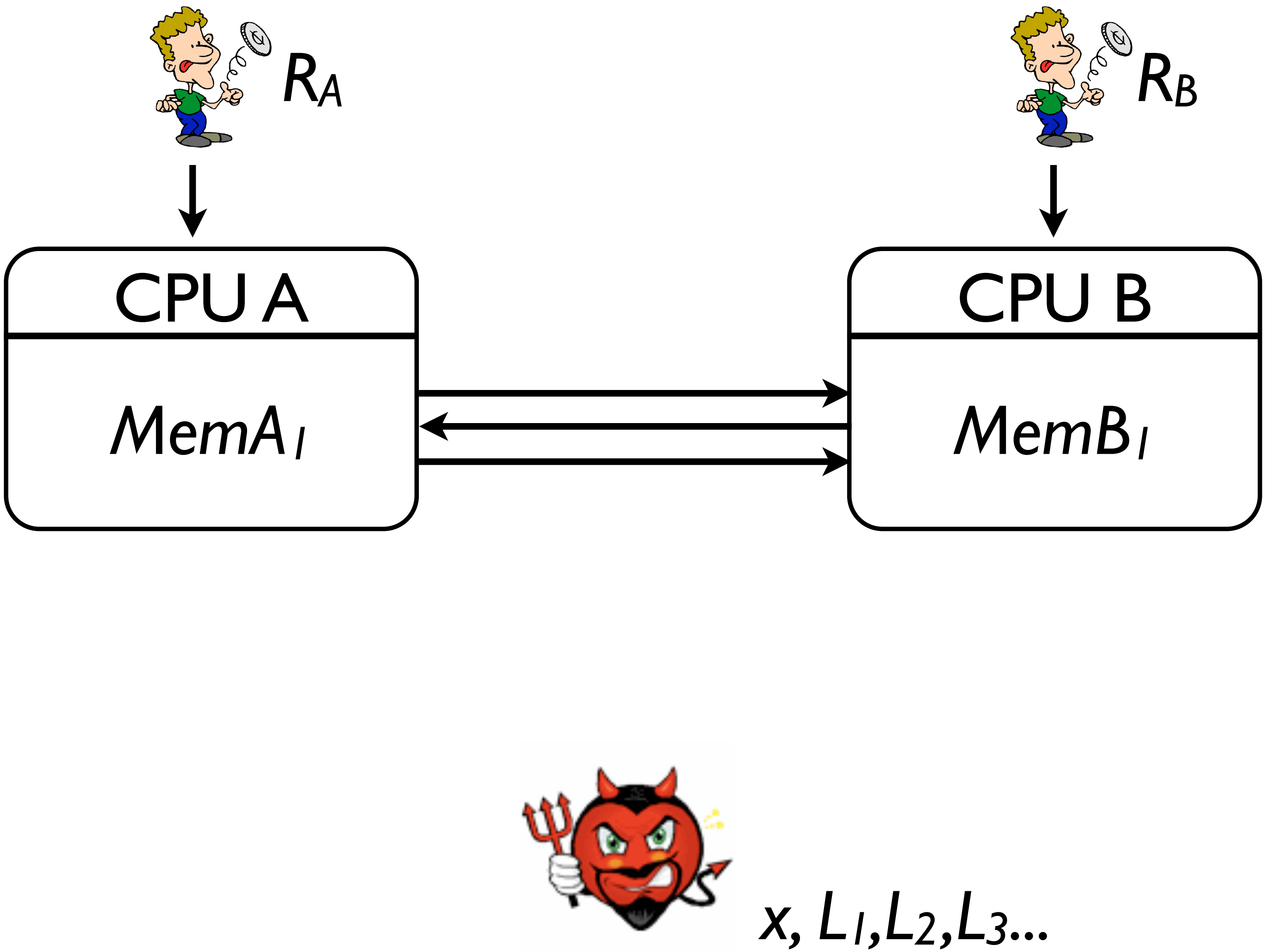
help

Leakage



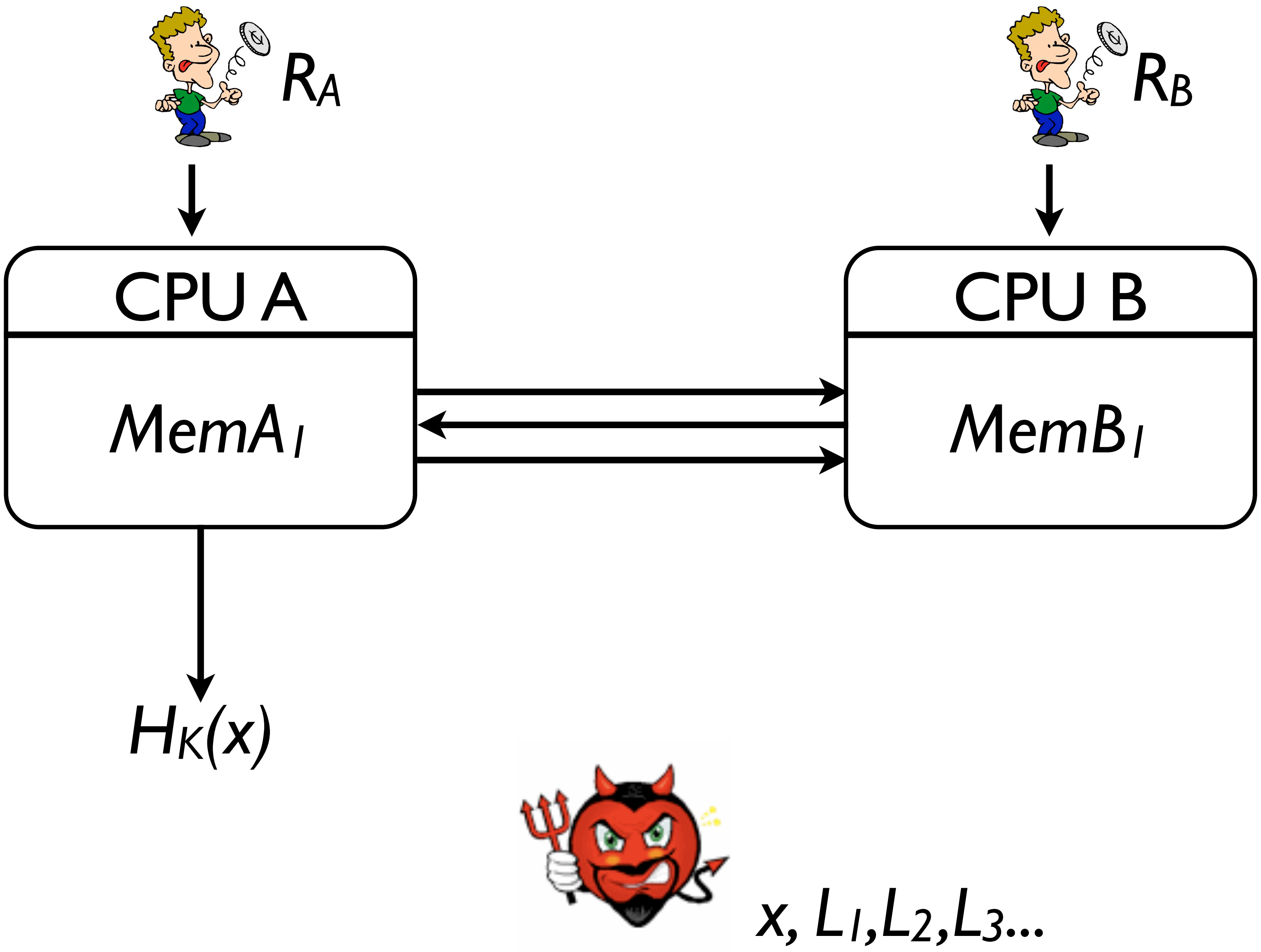
help

Leakage



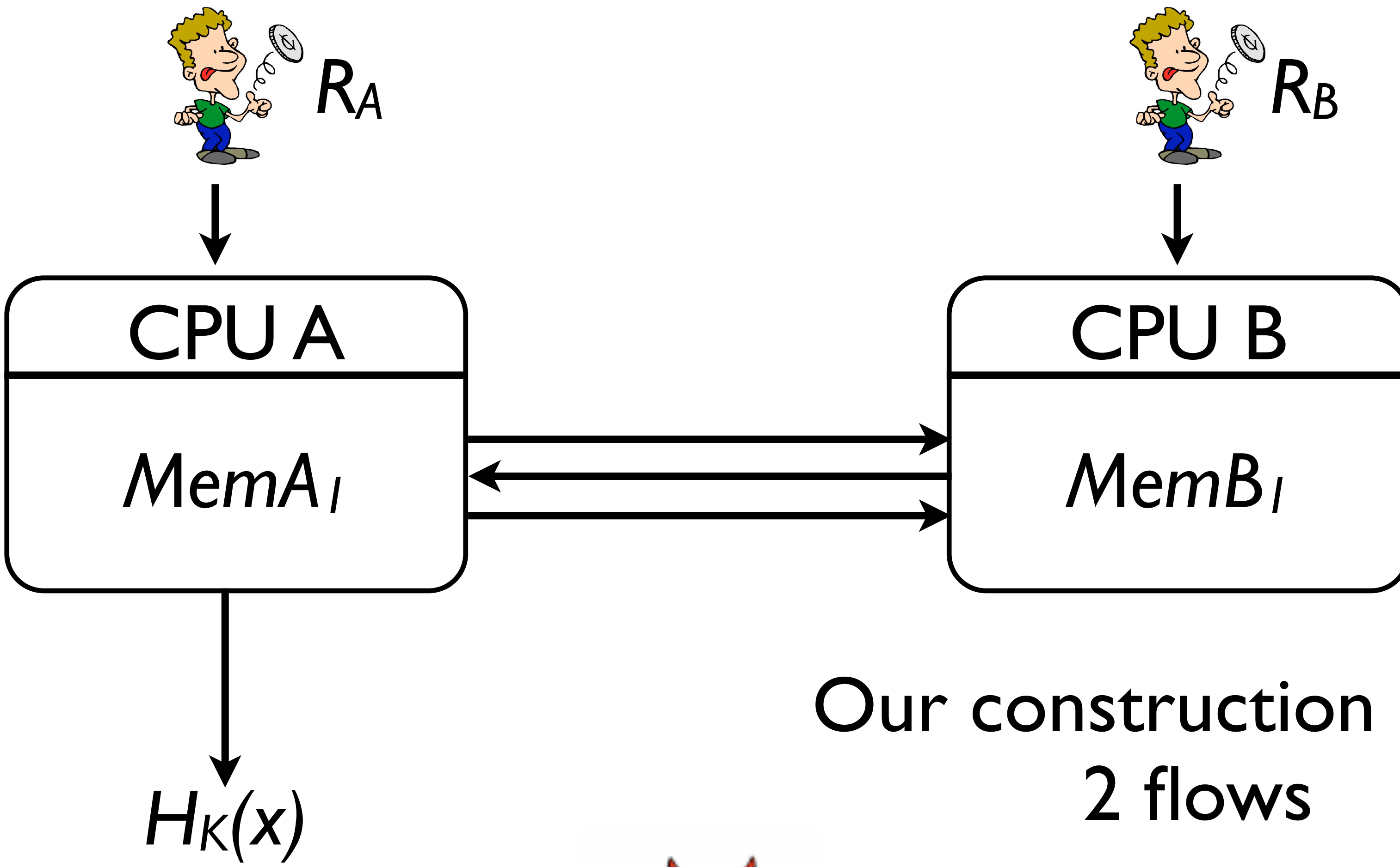
help

Leakage



help

Leakage



Our construction needs
2 flows

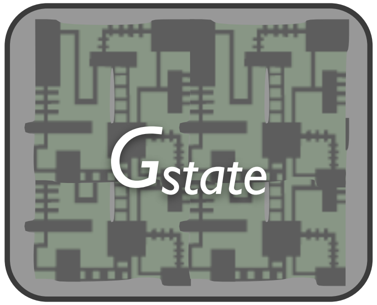
$x, L_1, L_2, L_3...$

help

Definition of Security

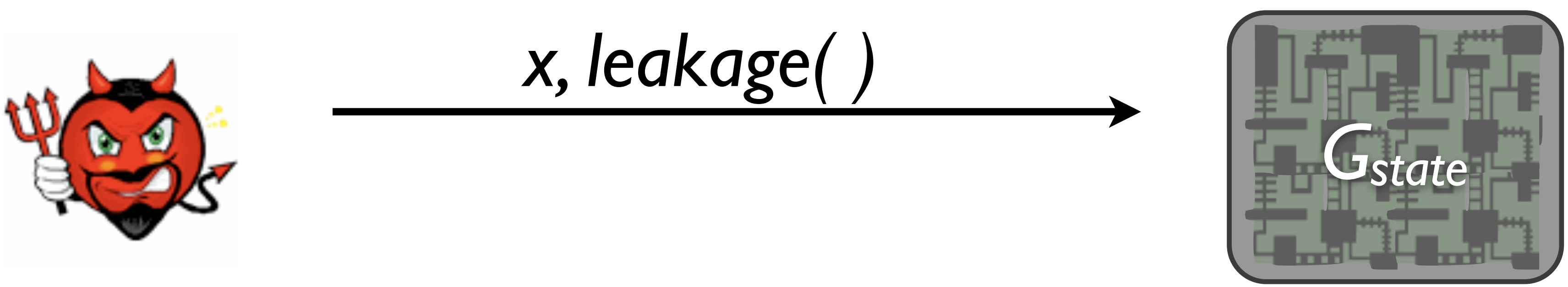
Definition of Security

Real World



Definition of Security

Real World

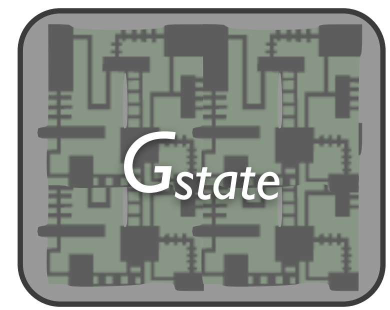


Definition of Security

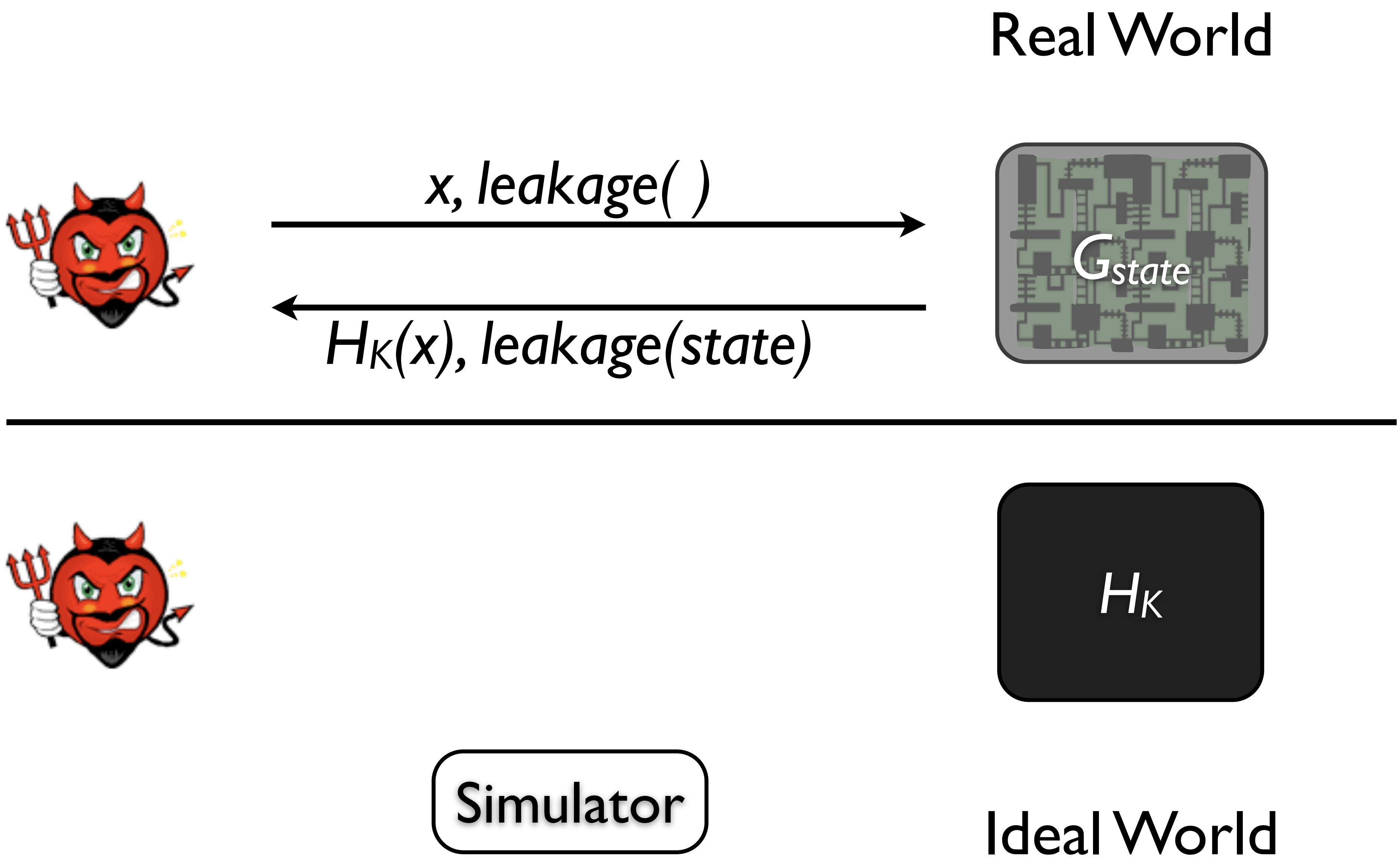
Real World

x, leakage( )

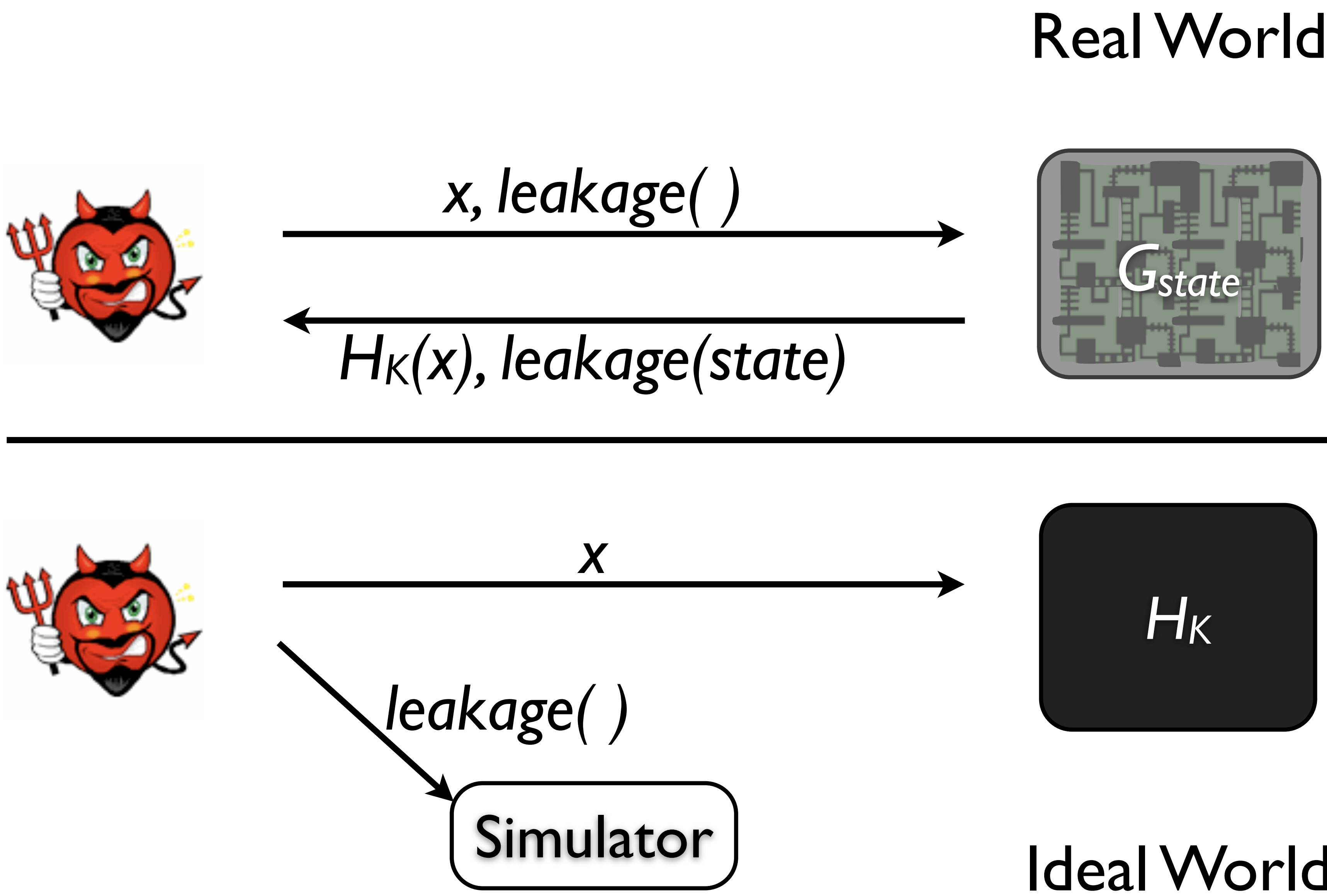
$H_K(x)$, *leakage(state)*



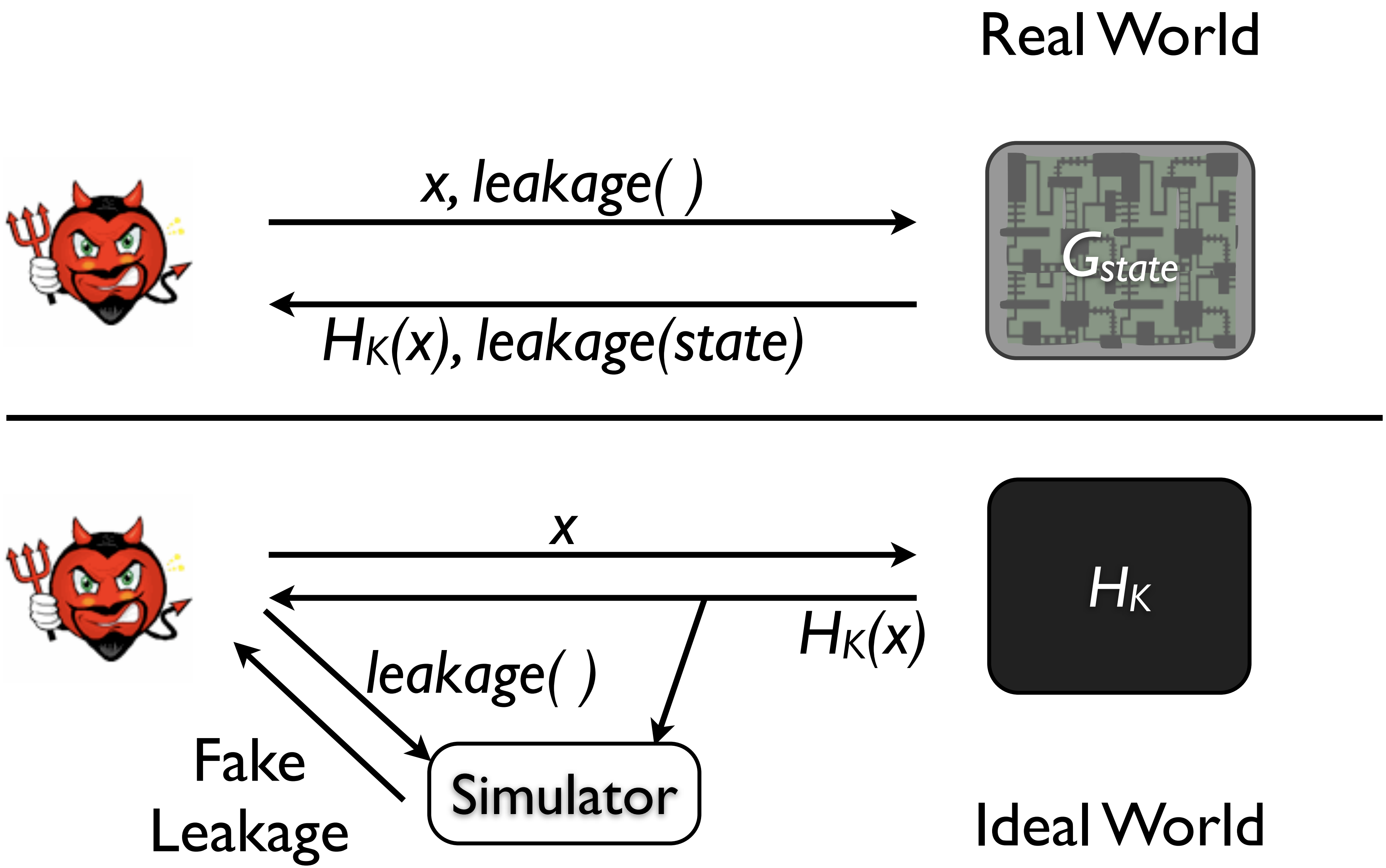
Definition of Security



Definition of Security



Definition of Security



Main Tool

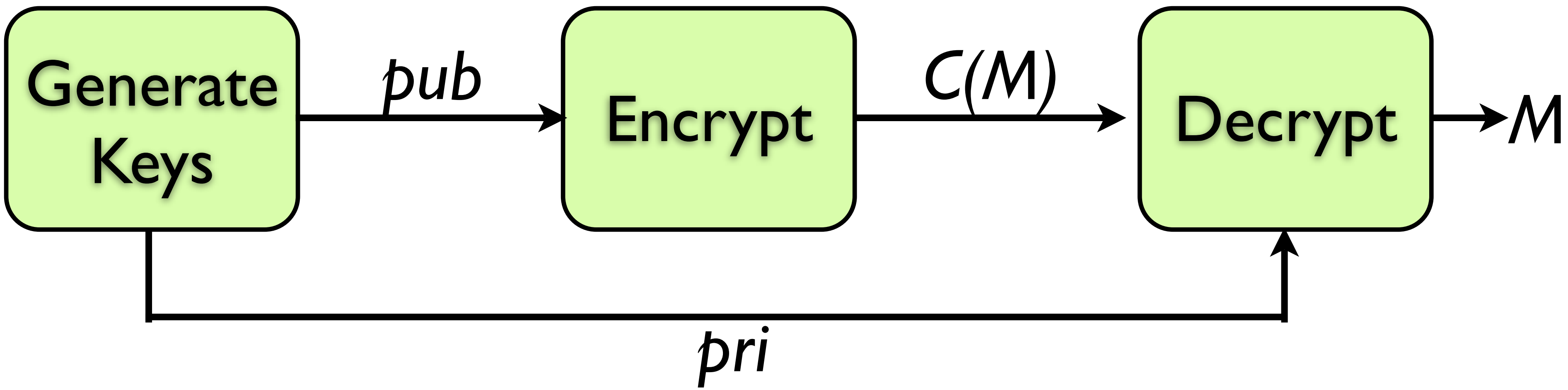
Fully Homomorphic Encryption (FHE)

First construction by Gentry at STOC 09
Based on Ideal Lattices

Other restricted constructions are known
[Boneh Goh Nissim 2005]
[Melchor Gaborit Herranz 2008]

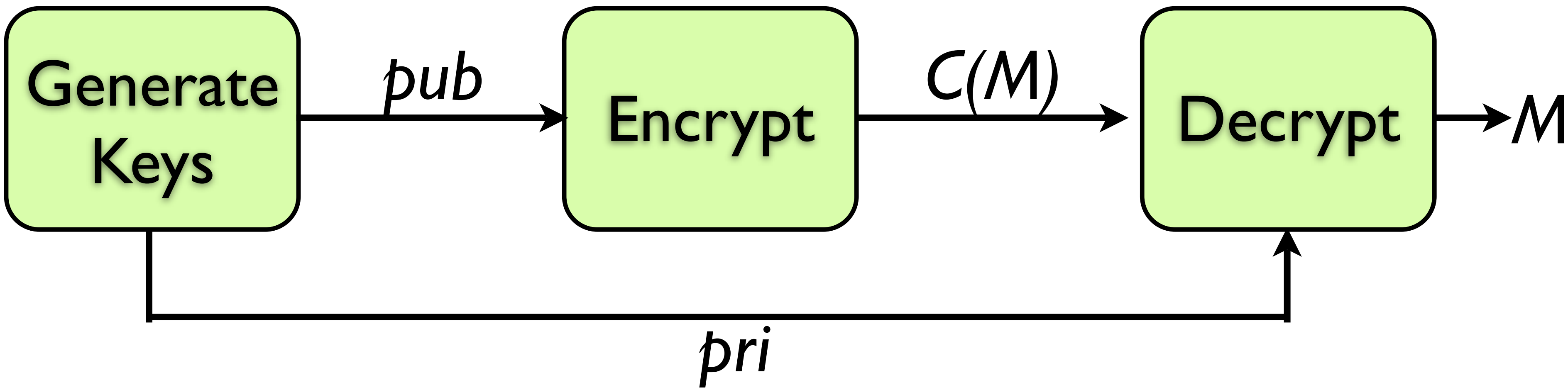
Homomorphic Encryption

Regular Public Key Encryption



Homomorphic Encryption

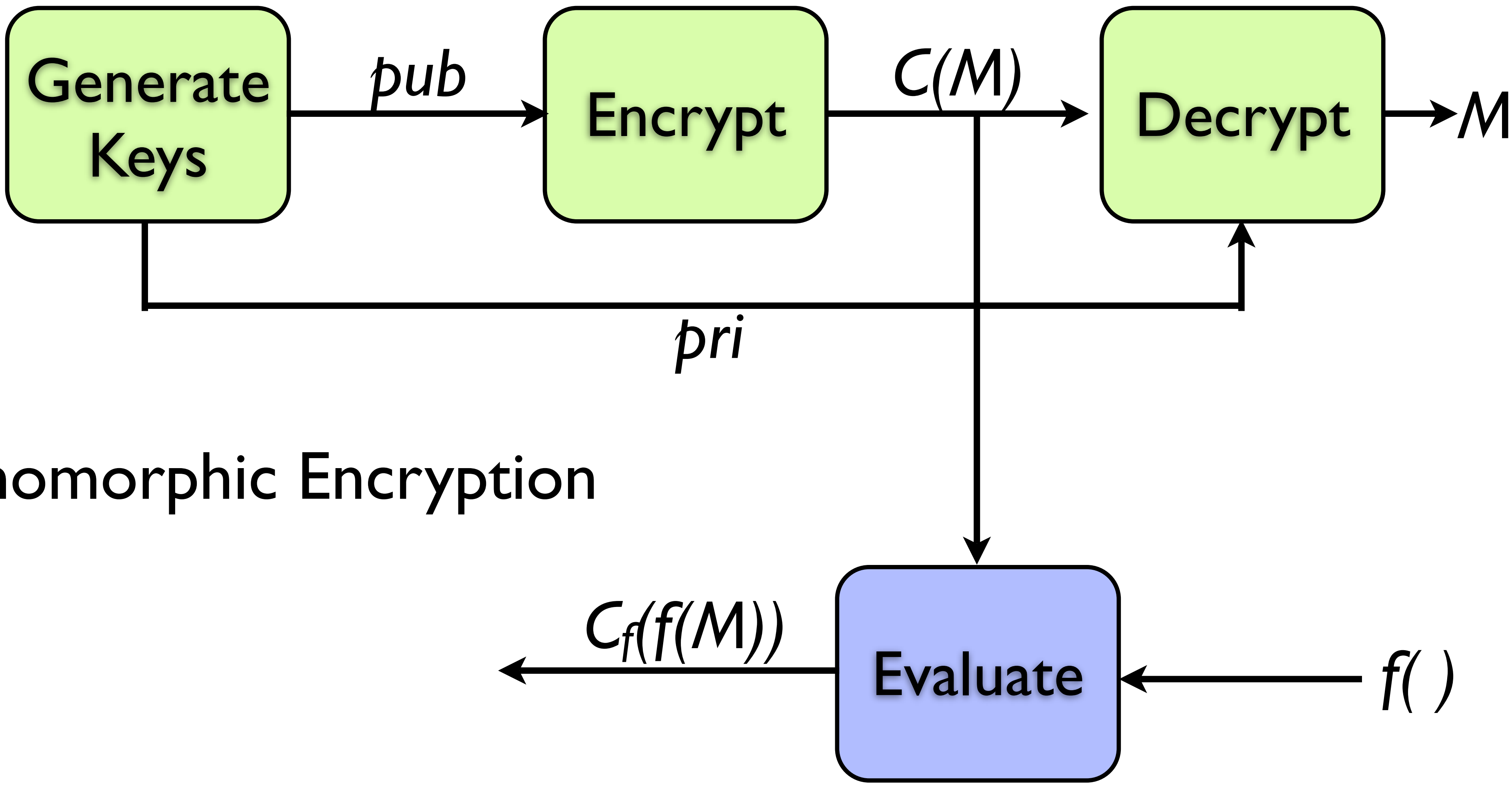
Regular Public Key Encryption



Homomorphic Encryption

Homomorphic Encryption

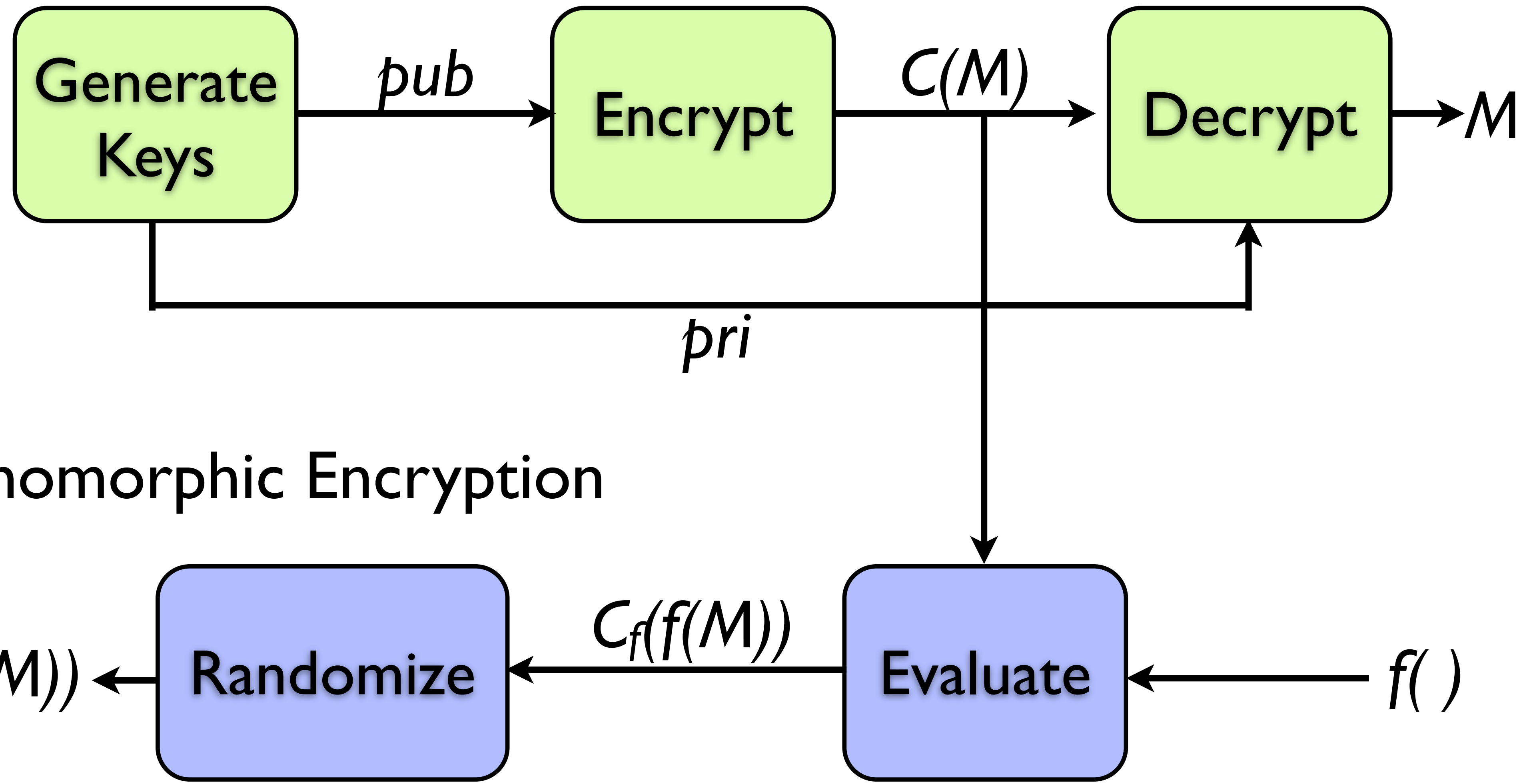
Regular Public Key Encryption



Homomorphic Encryption

Homomorphic Encryption

Regular Public Key Encryption



Homomorphic Encryption

Our Construction

Initialization(K):

Generate keys *pri*,*pub*

$C(K) = Enc_{pub}(K)$

Our Construction

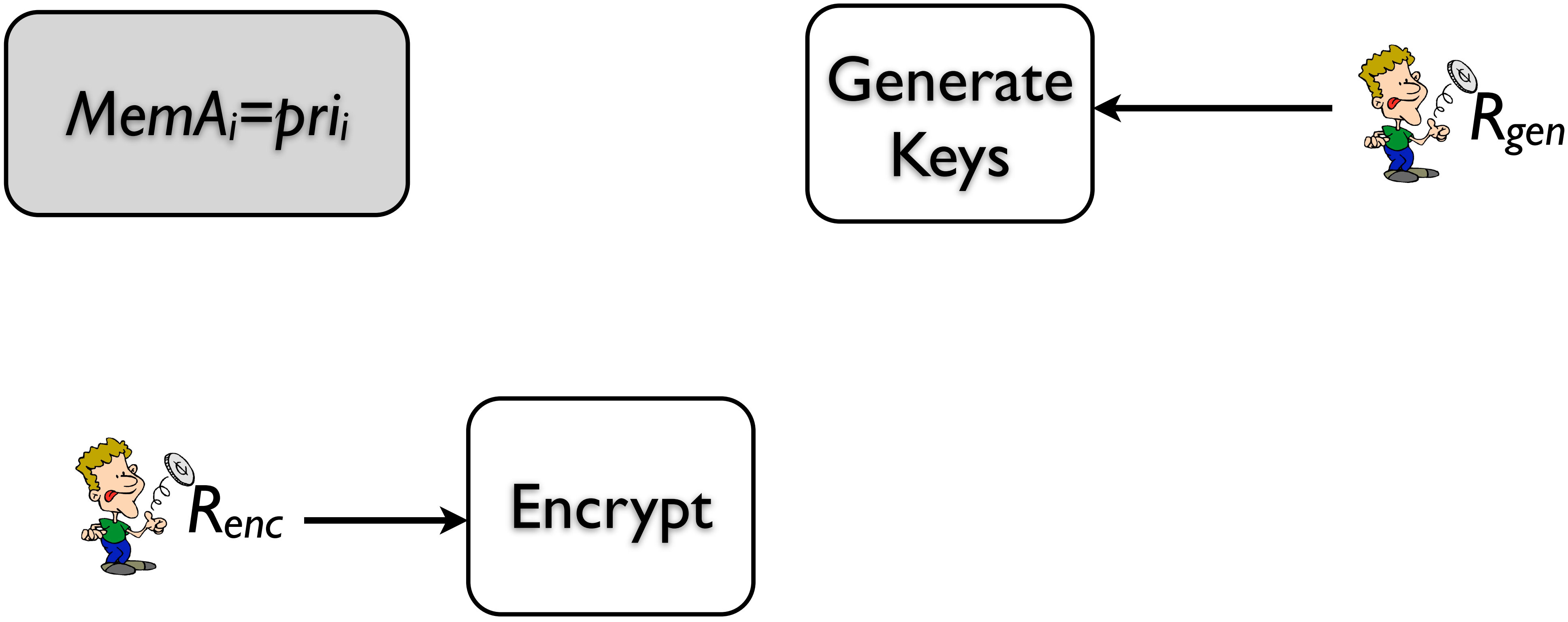
Initialization(K):

Generate keys pri,pub

$C(K) = Enc_{pub}(K)$

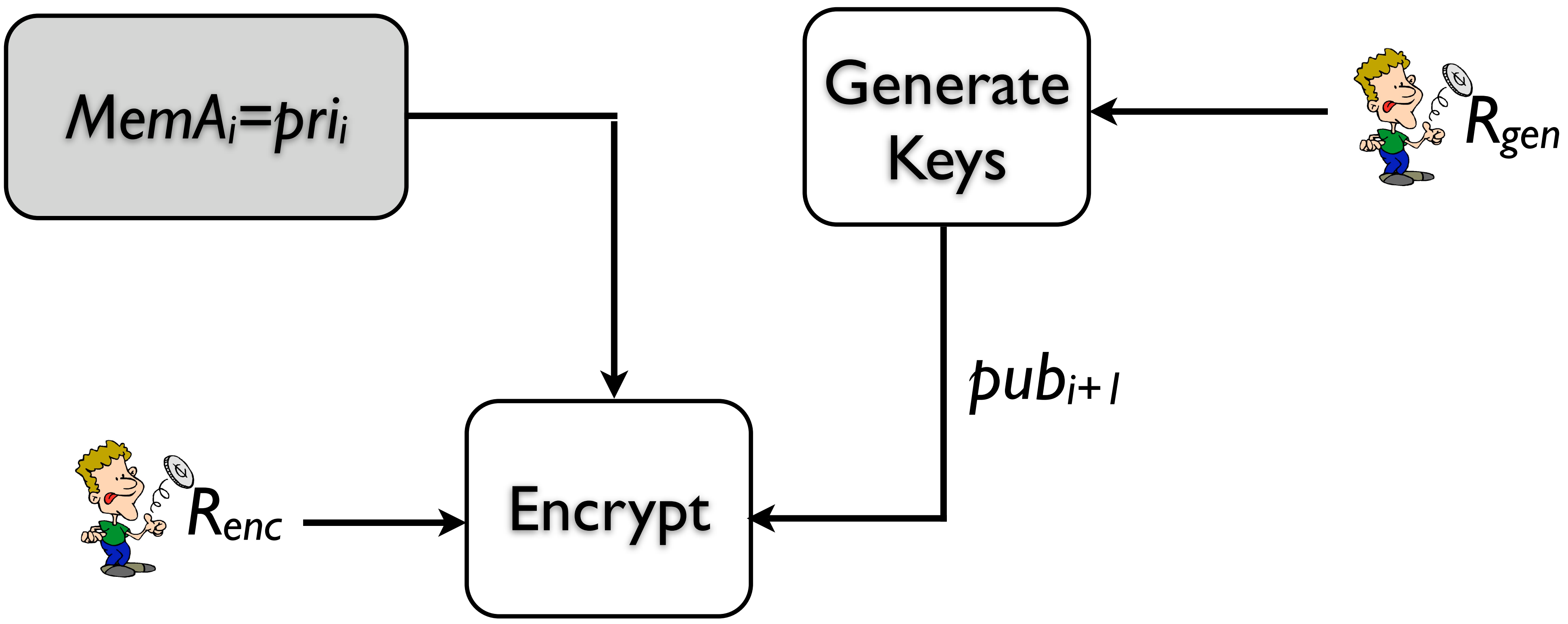
First Attempt

Step 1: CPU A



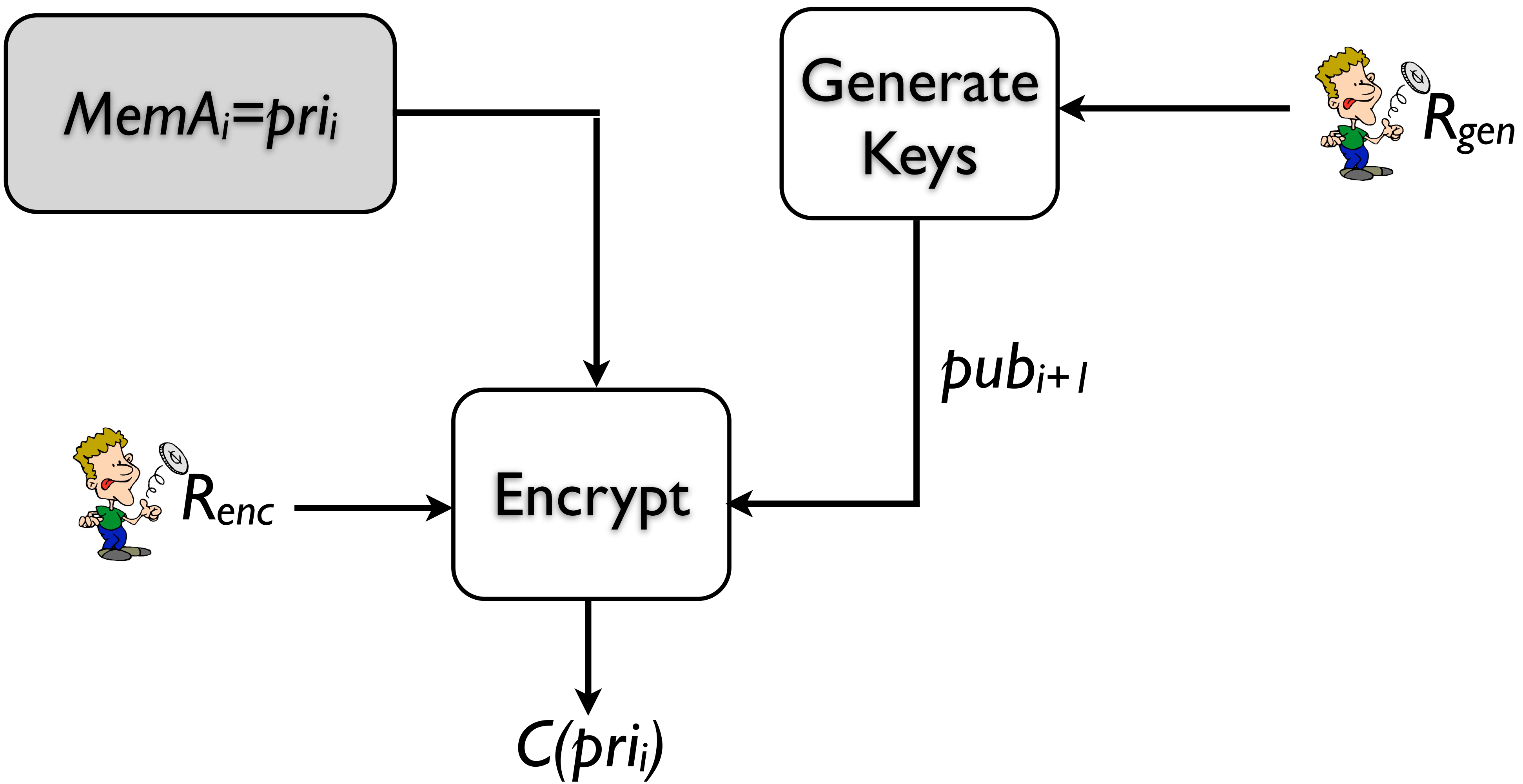
First Attempt

Step 1: CPU A



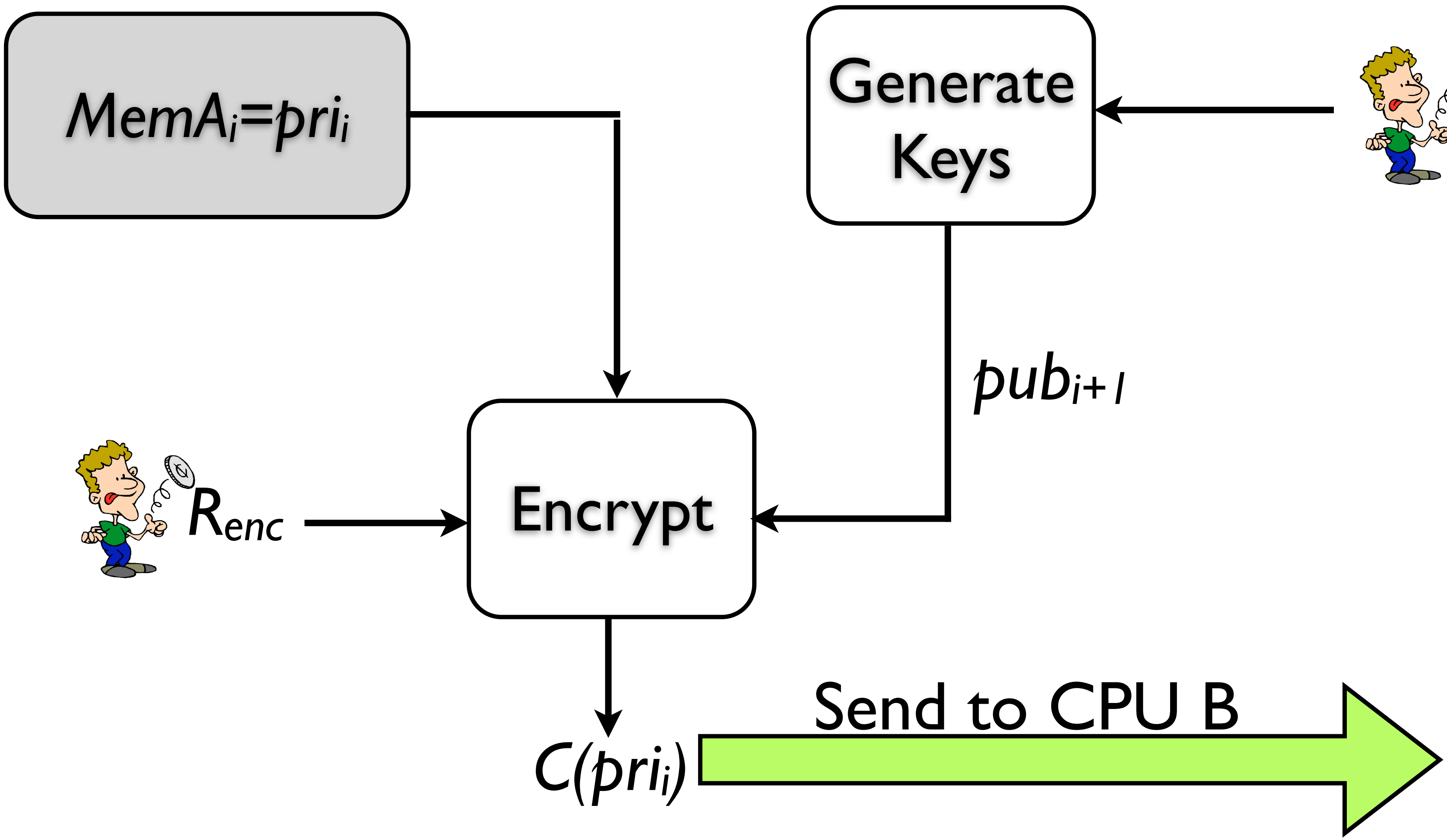
First Attempt

Step 1: CPU A



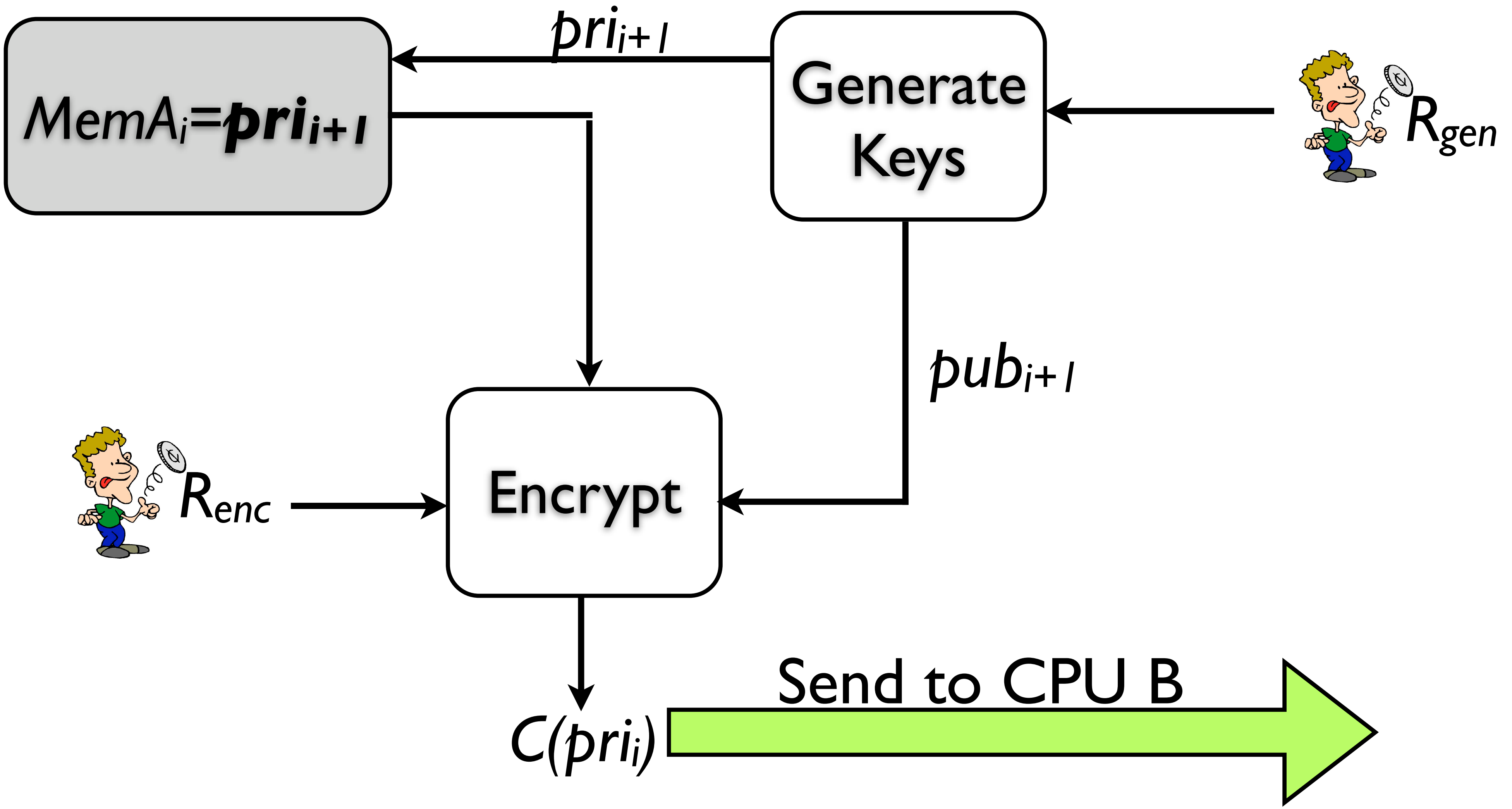
First Attempt

Step 1: CPU A



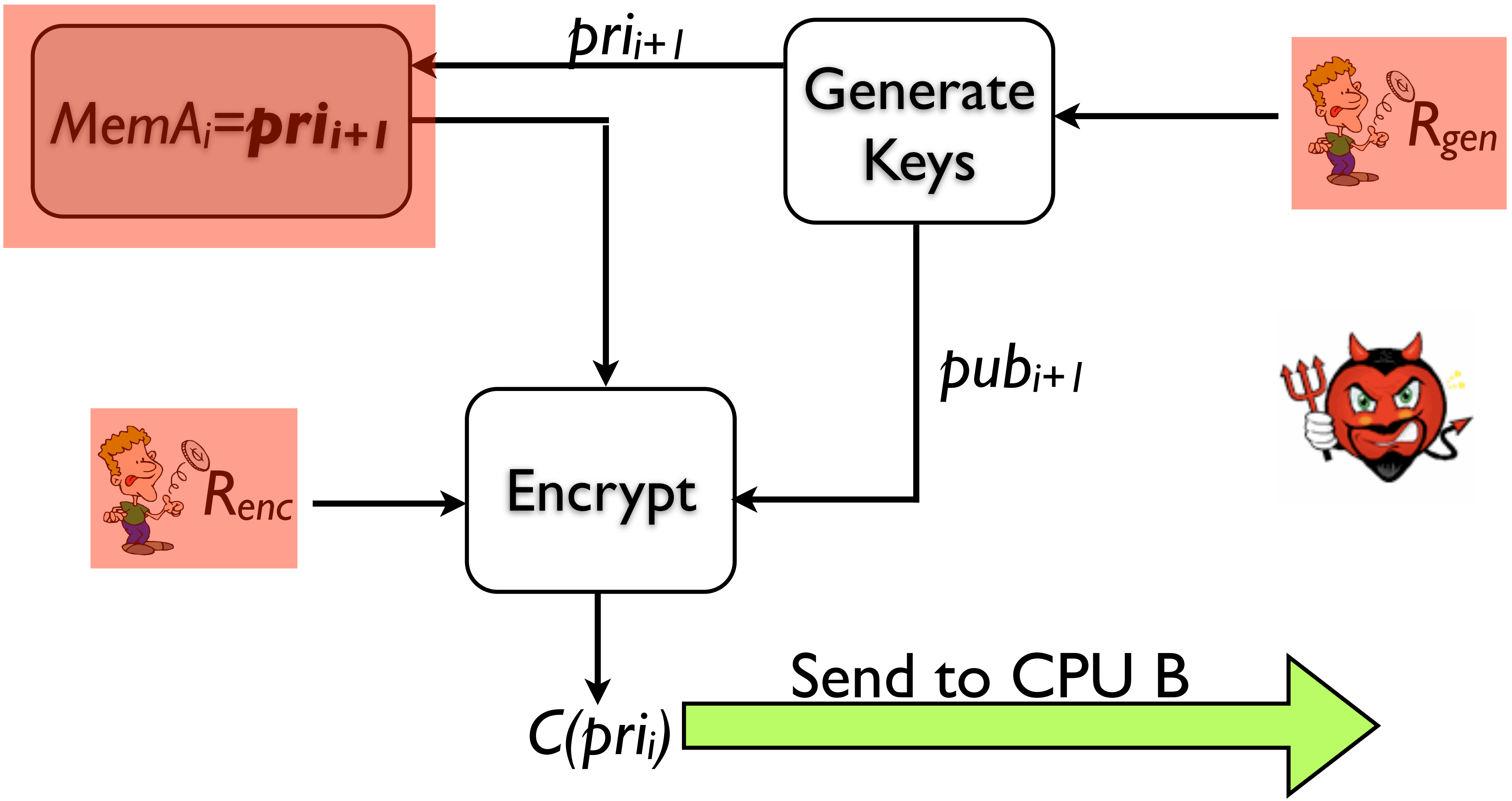
First Attempt

Step 1: CPU A



First Attempt

Step 1: CPU A



First Attempt

Step 2: CPU B

Input: *x*

$C_{i+1}(pri_i)$

Evaluate relative to pub_{i+1}

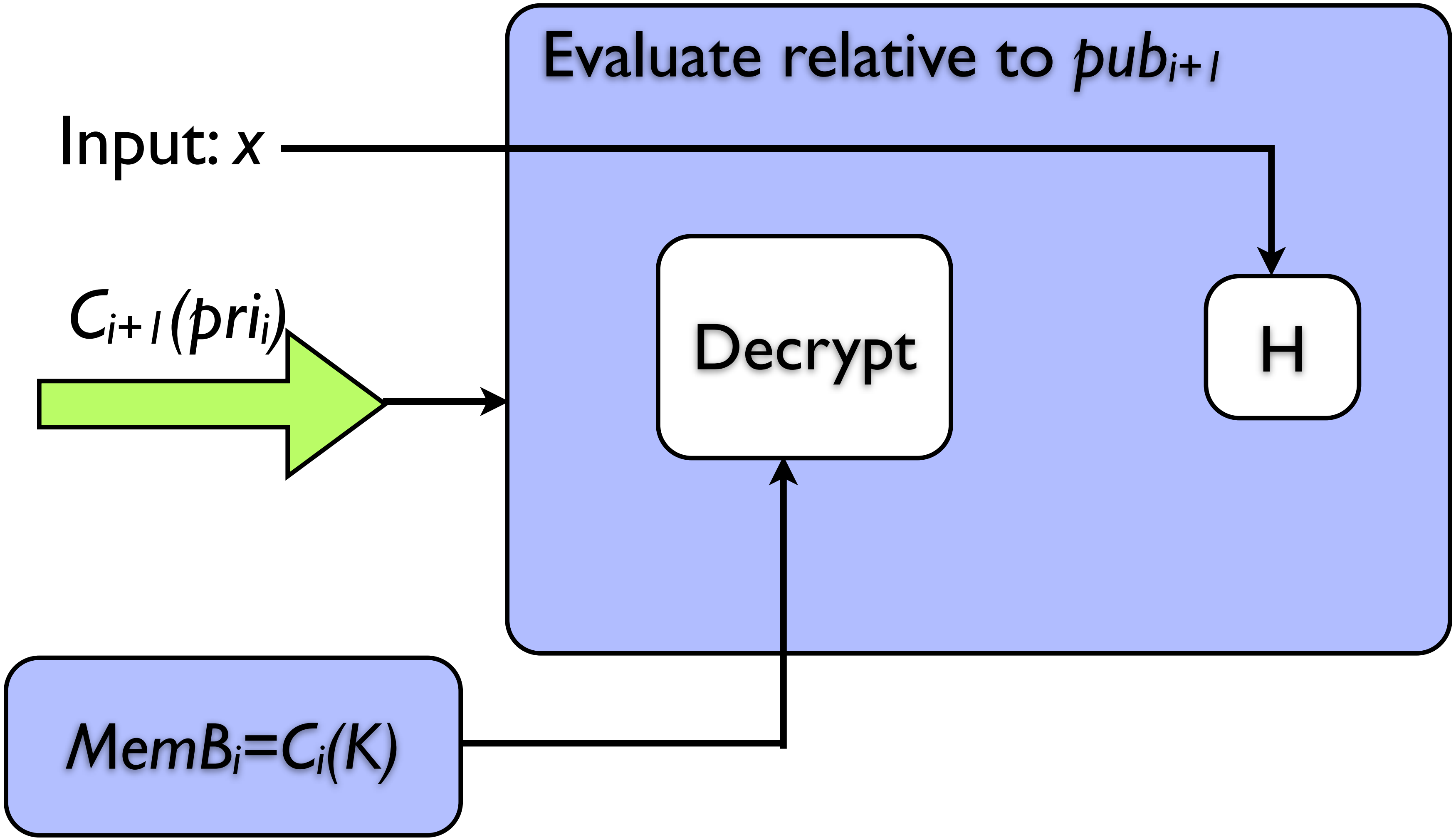
Decrypt

H

$MemB_i=C_i(K)$

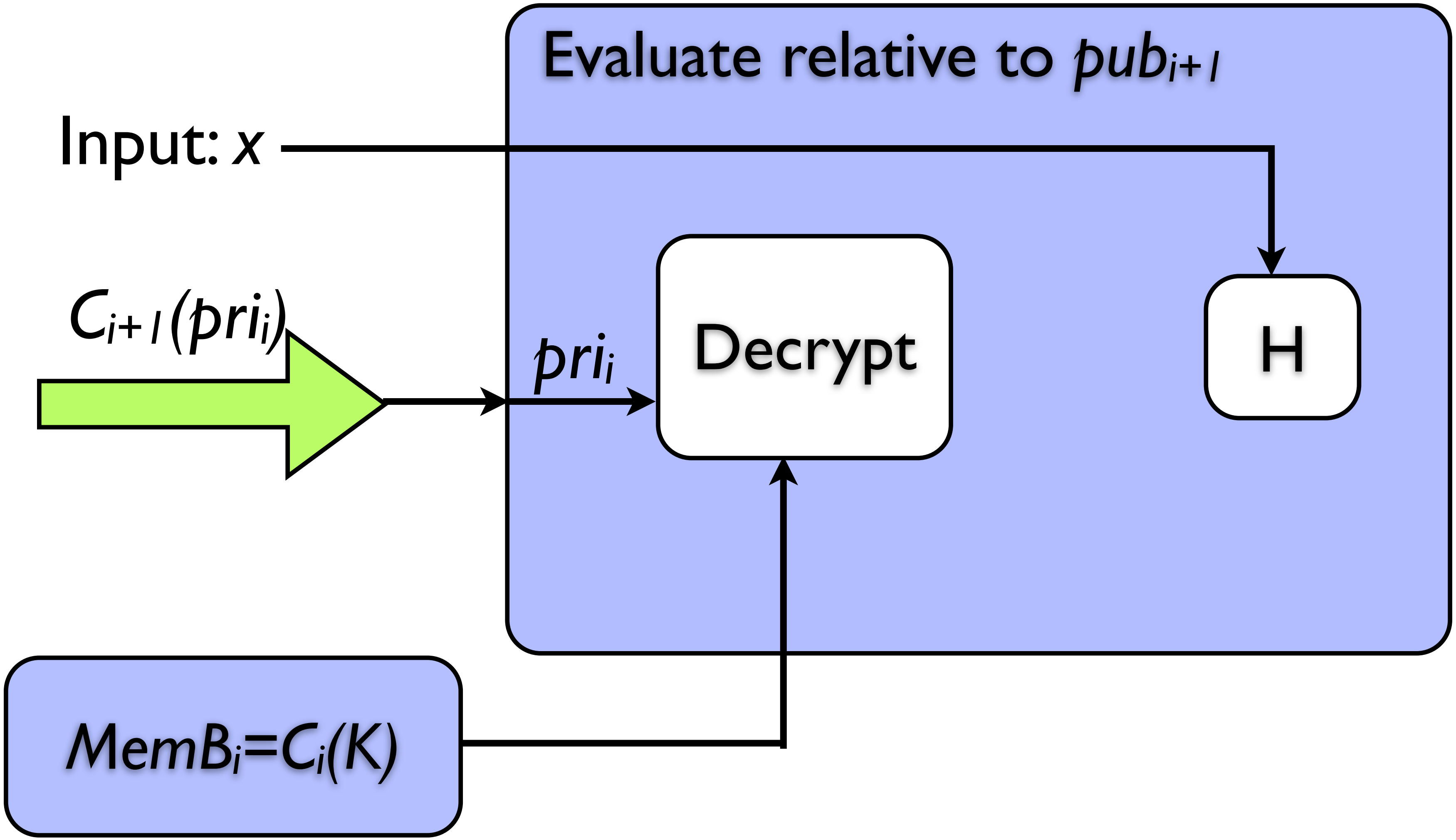
First Attempt

Step 2: CPU B



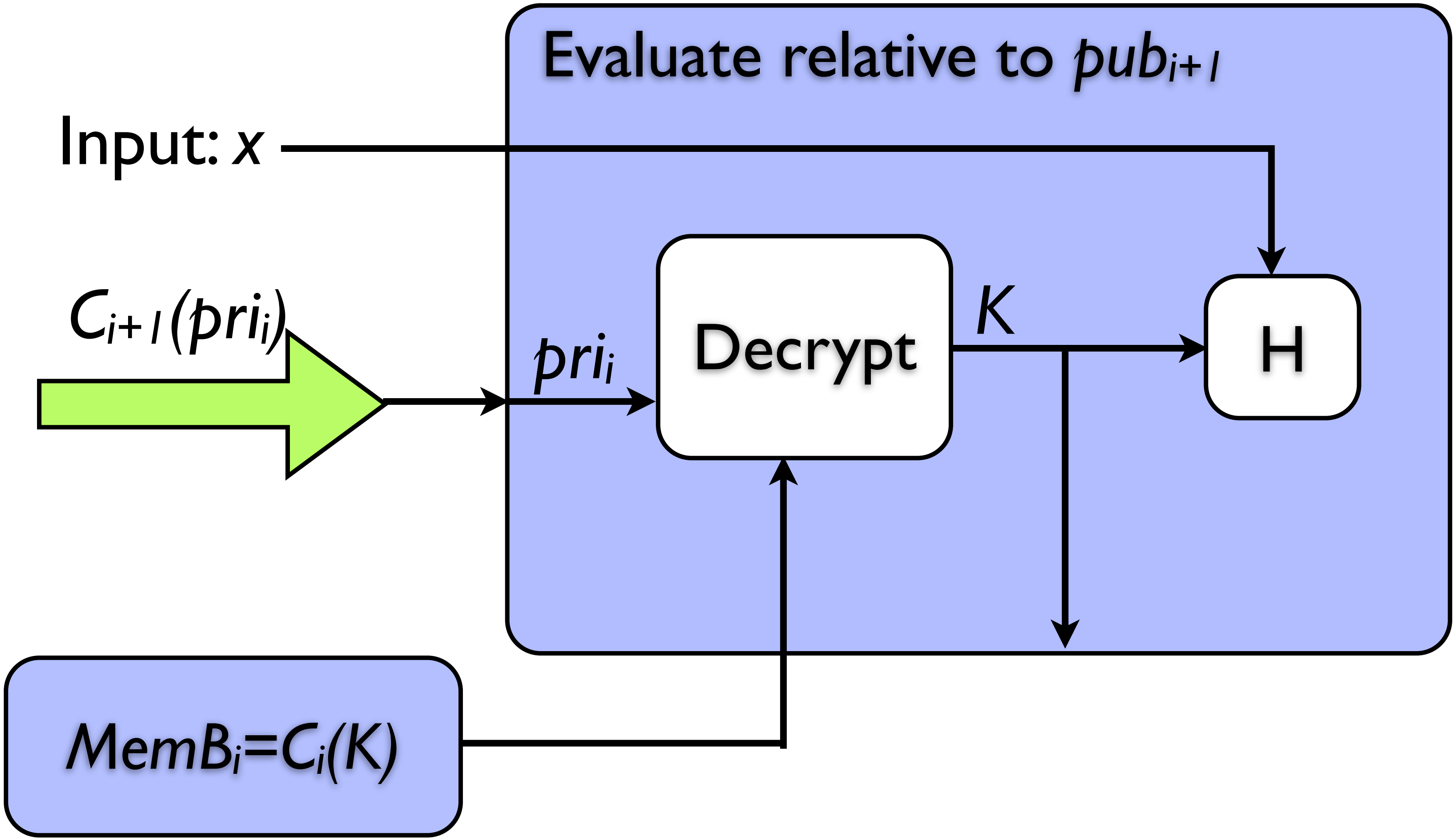
First Attempt

Step 2: CPU B



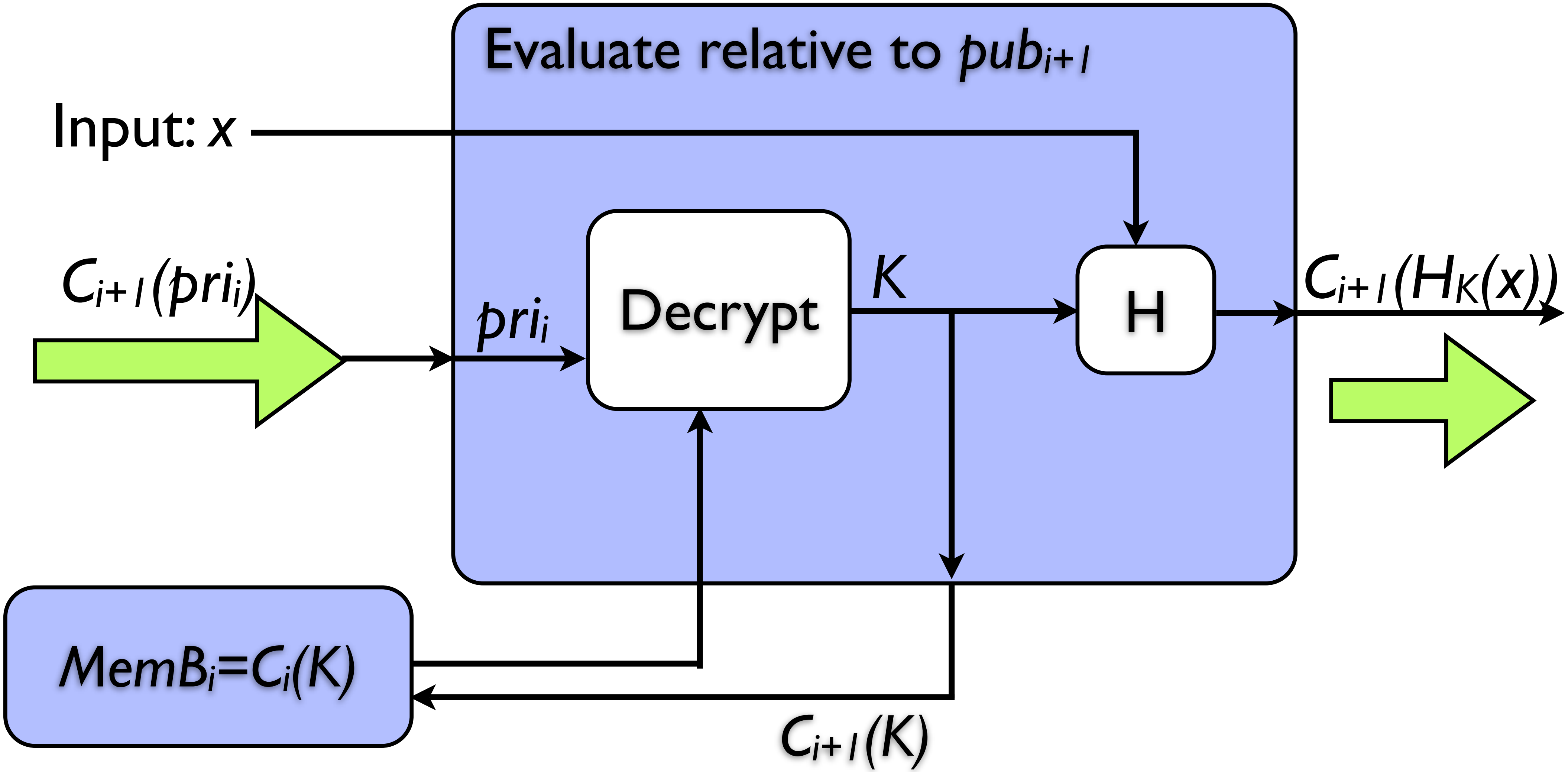
First Attempt

Step 2: CPU B



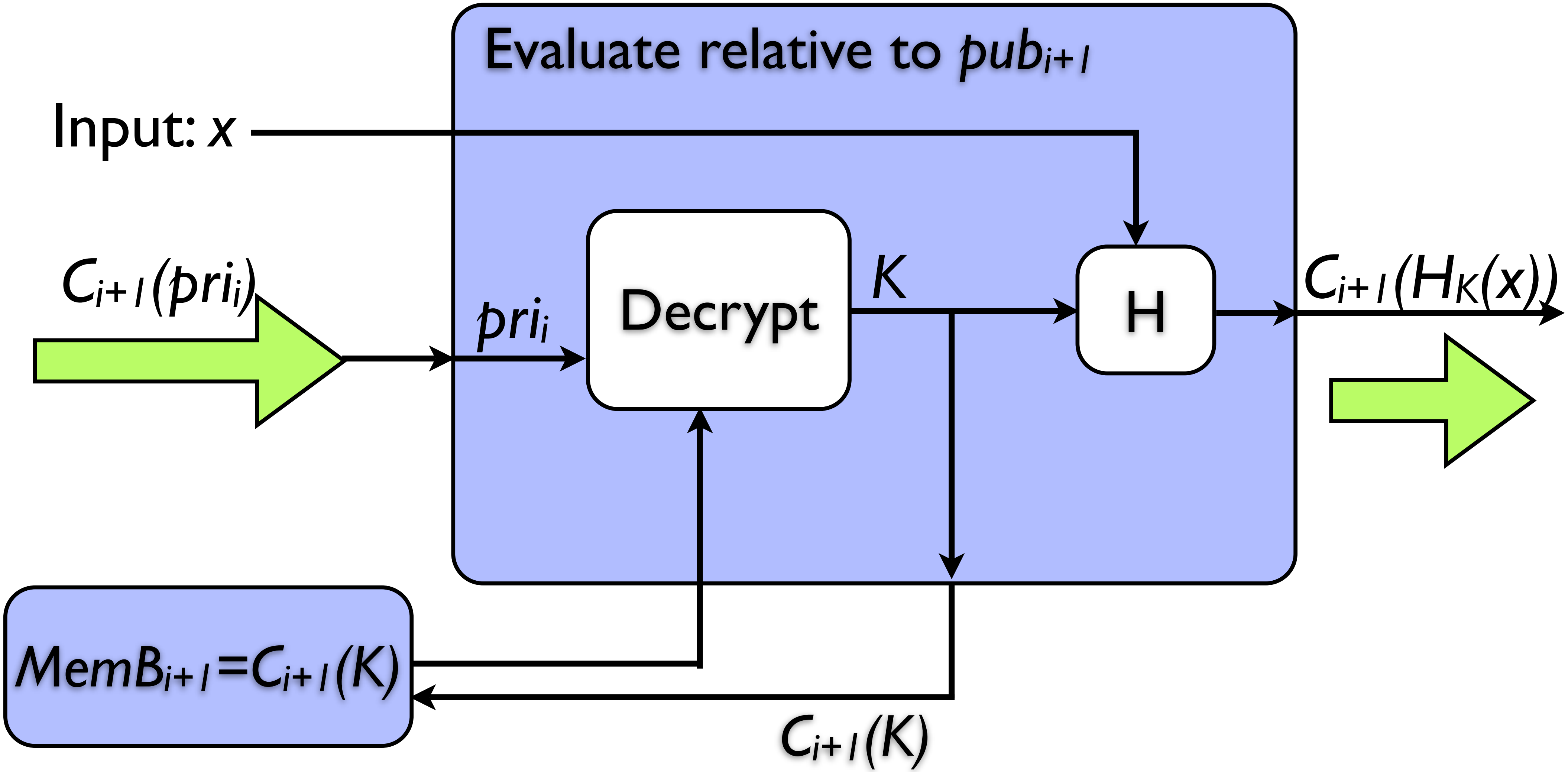
First Attempt

Step 2: CPU B

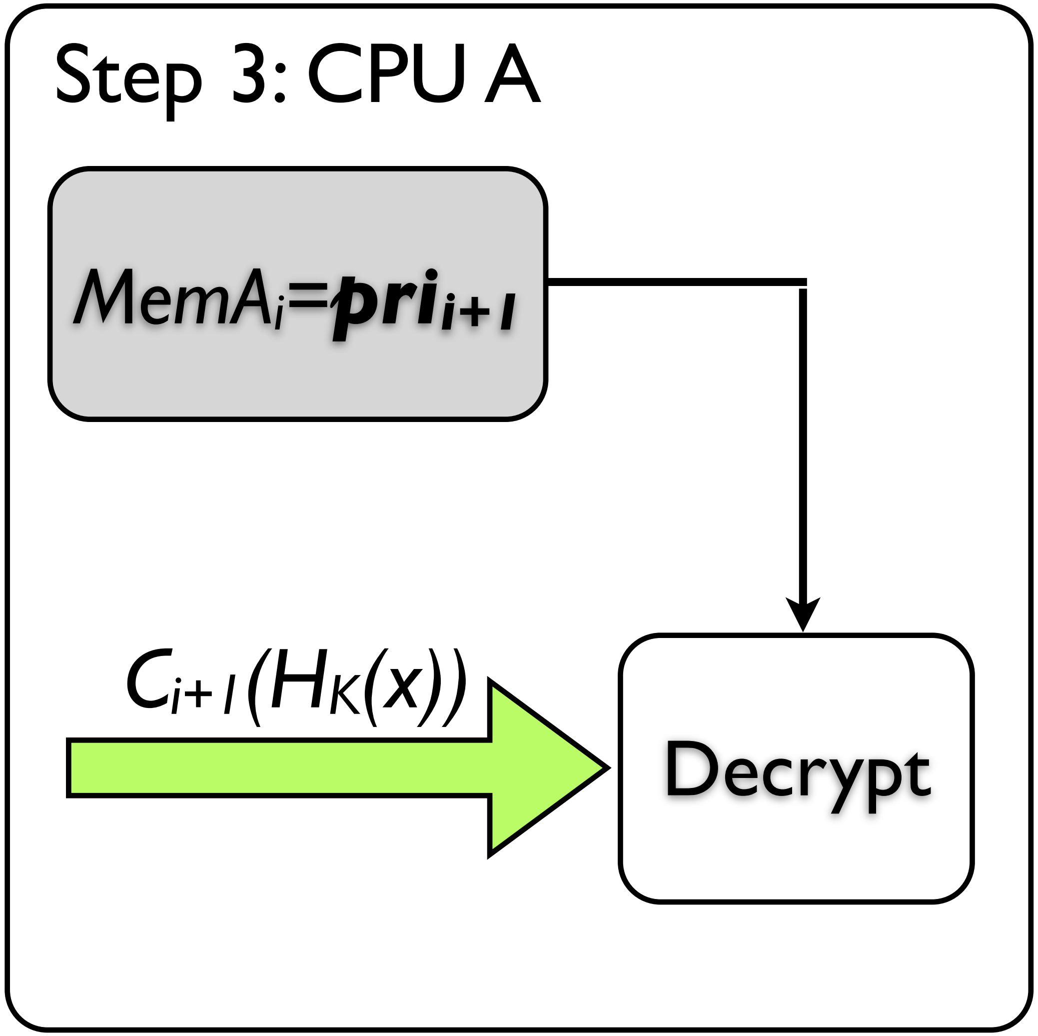


First Attempt

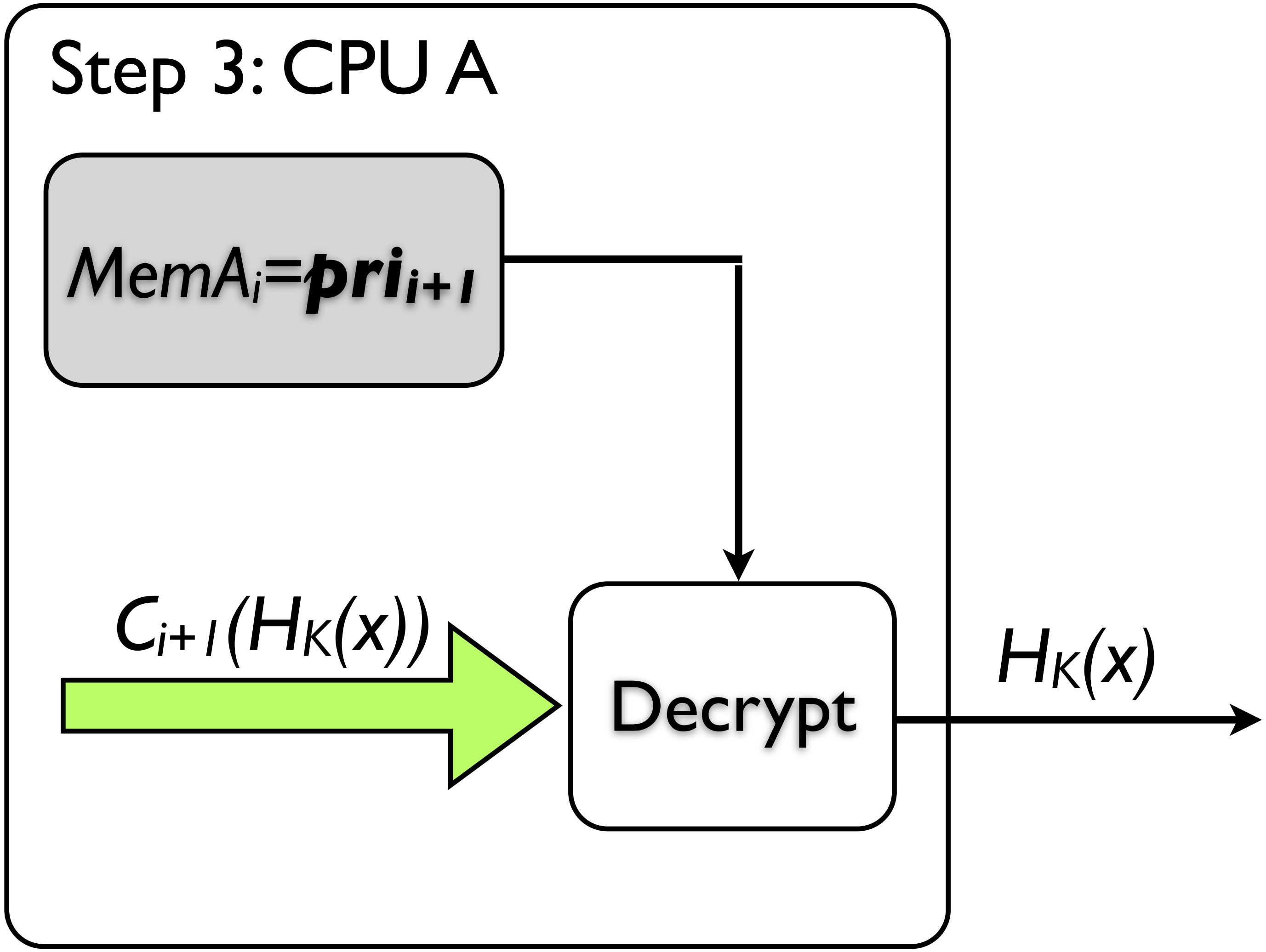
Step 2: CPU B



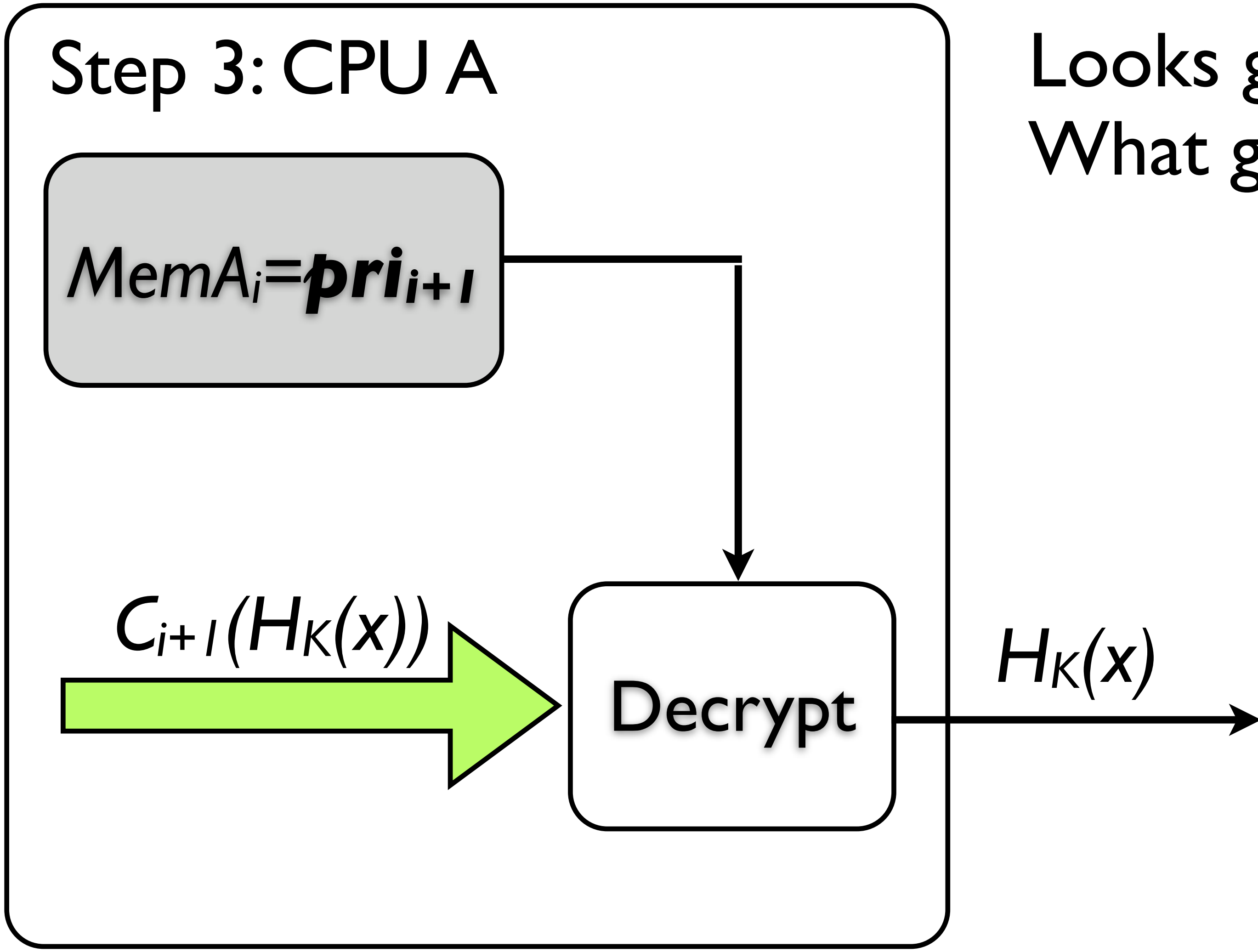
First Attempt



First Attempt

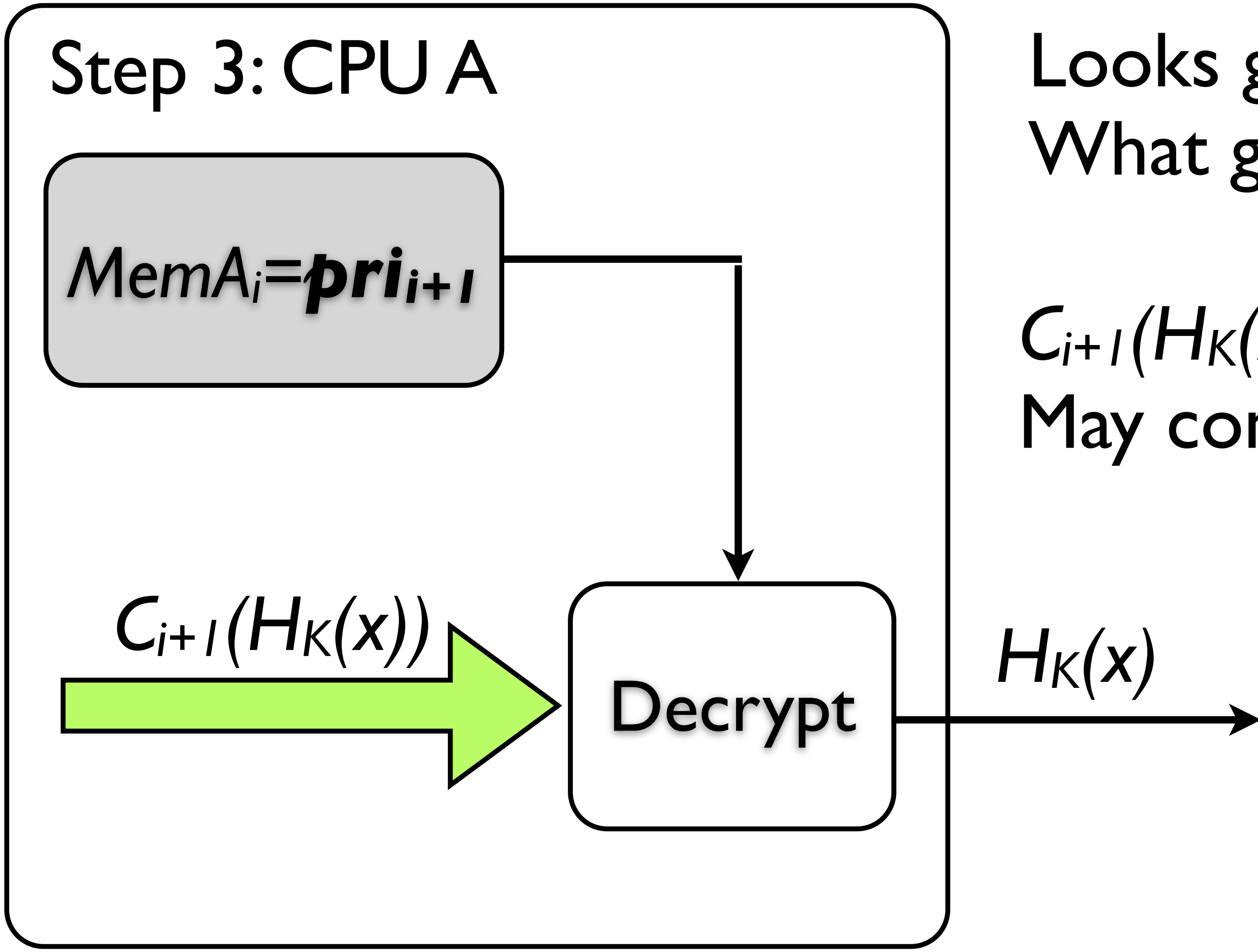


First Attempt



Looks great!
What goes wrong?

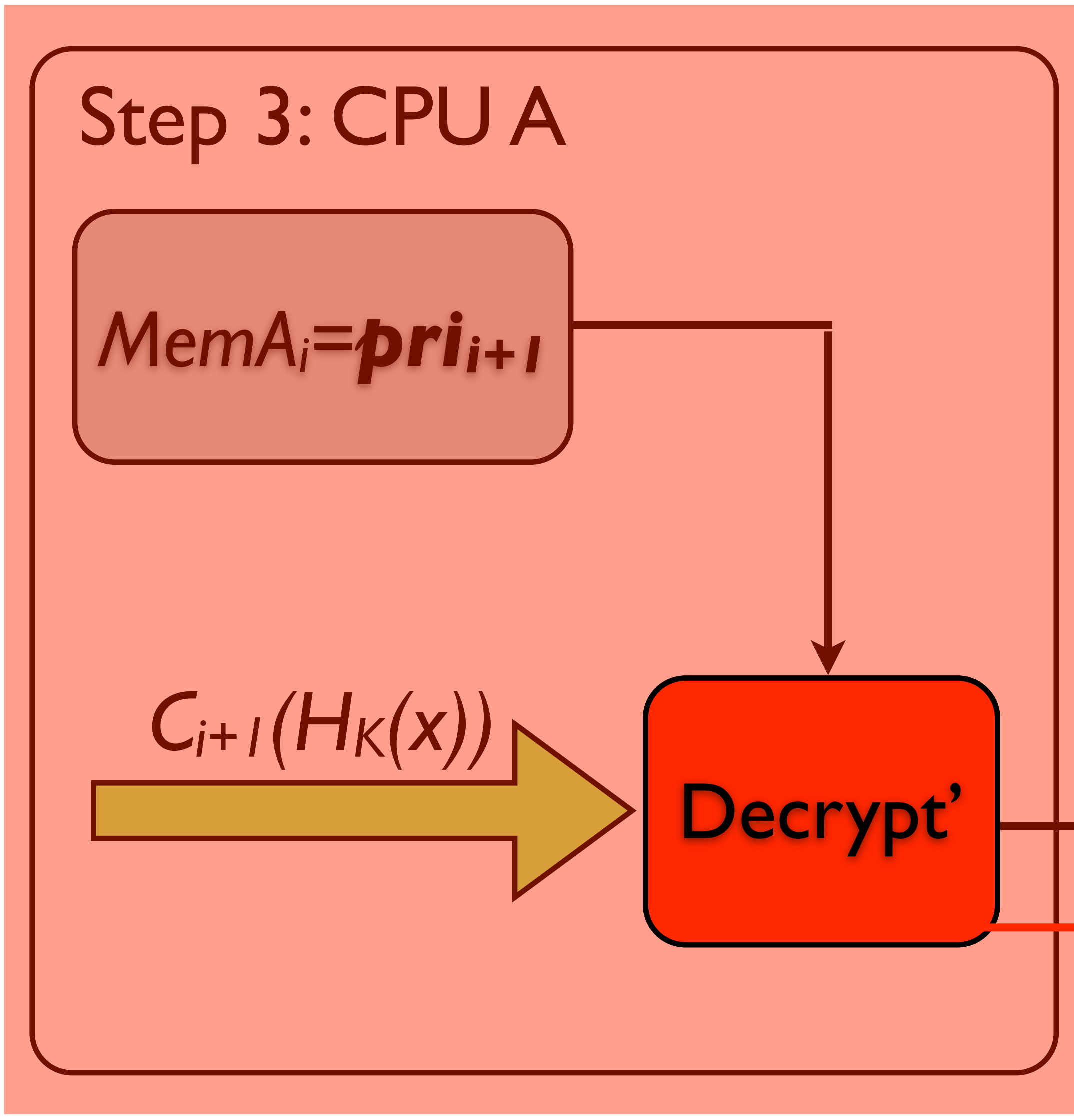
First Attempt



Looks great!
What goes wrong?

$C_{i+1}(H_K(x))$ carries history
May contain K

First Attempt

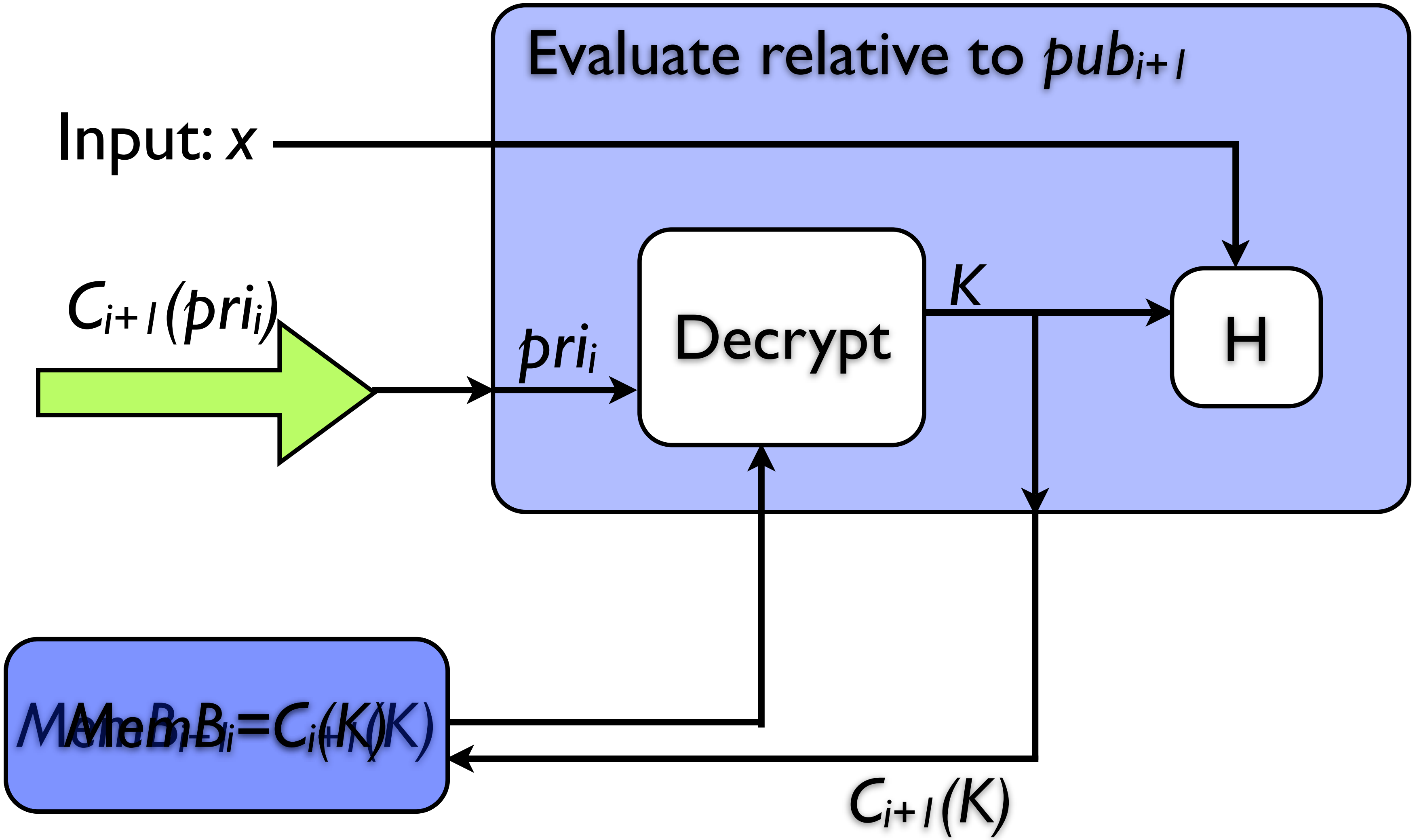


Looks great!
What goes wrong?

$C_{i+1}(H_K(x))$ carries history
May contain K

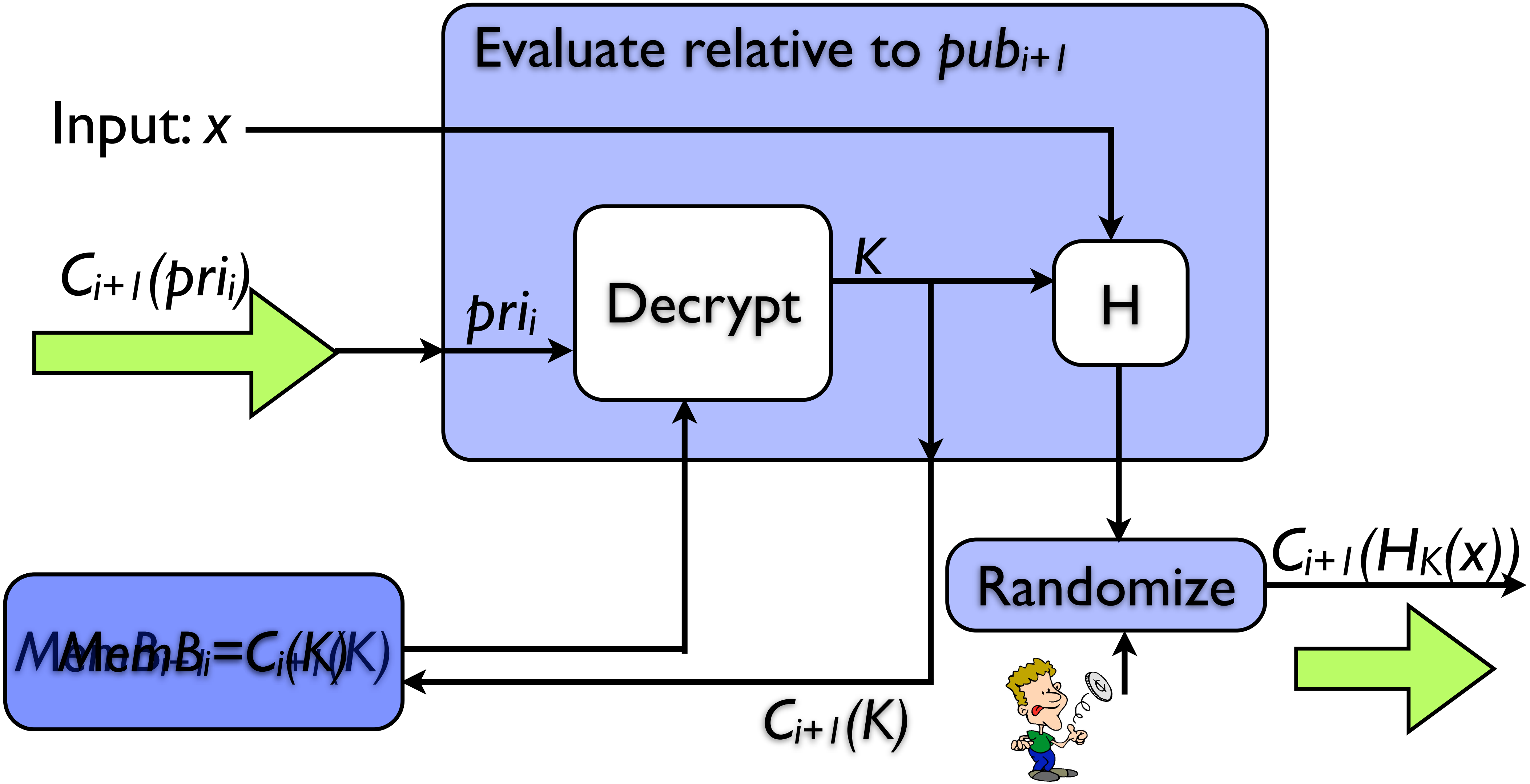
Second Attempt

Step 2’: CPU B



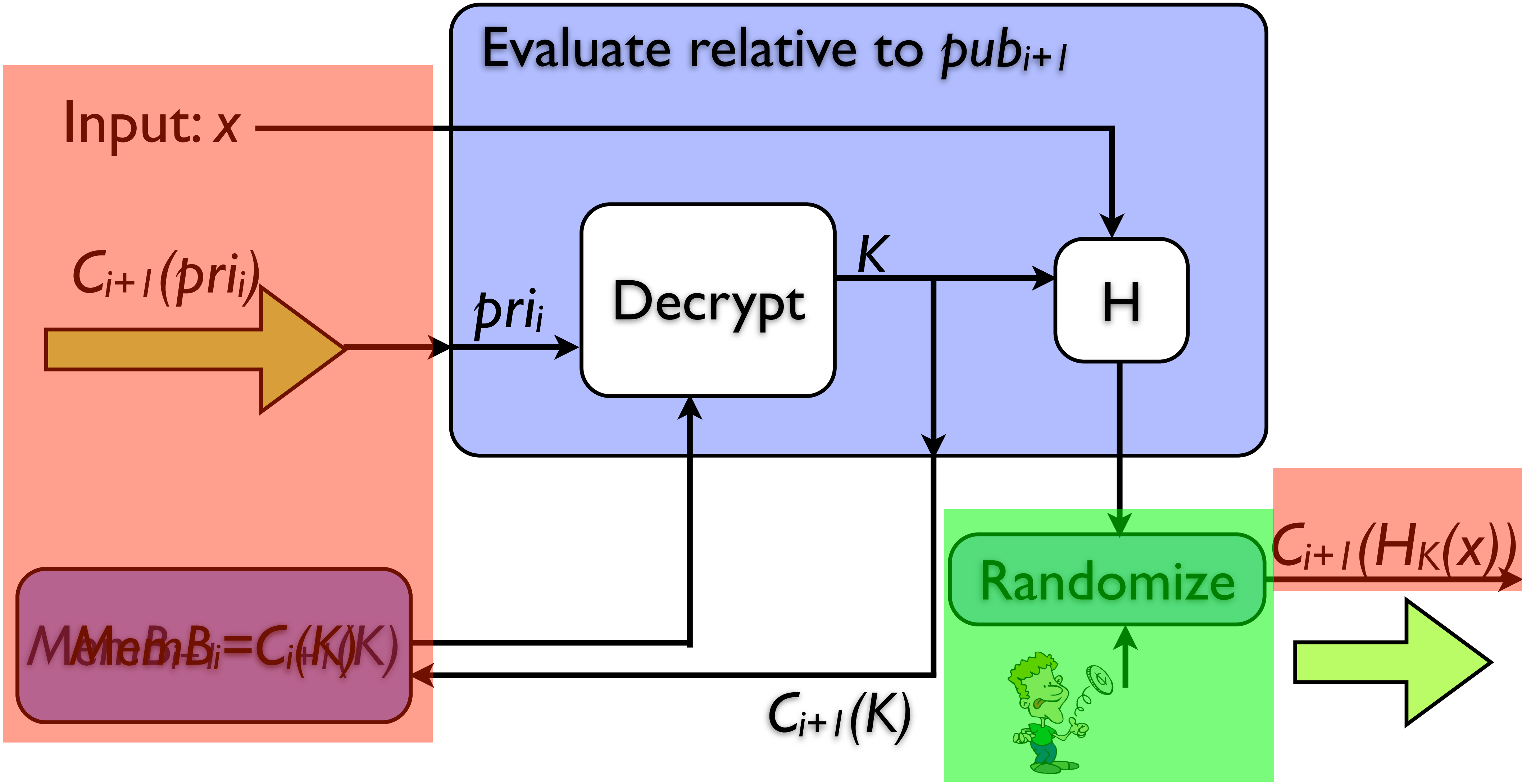
Second Attempt

Step 2': CPU B

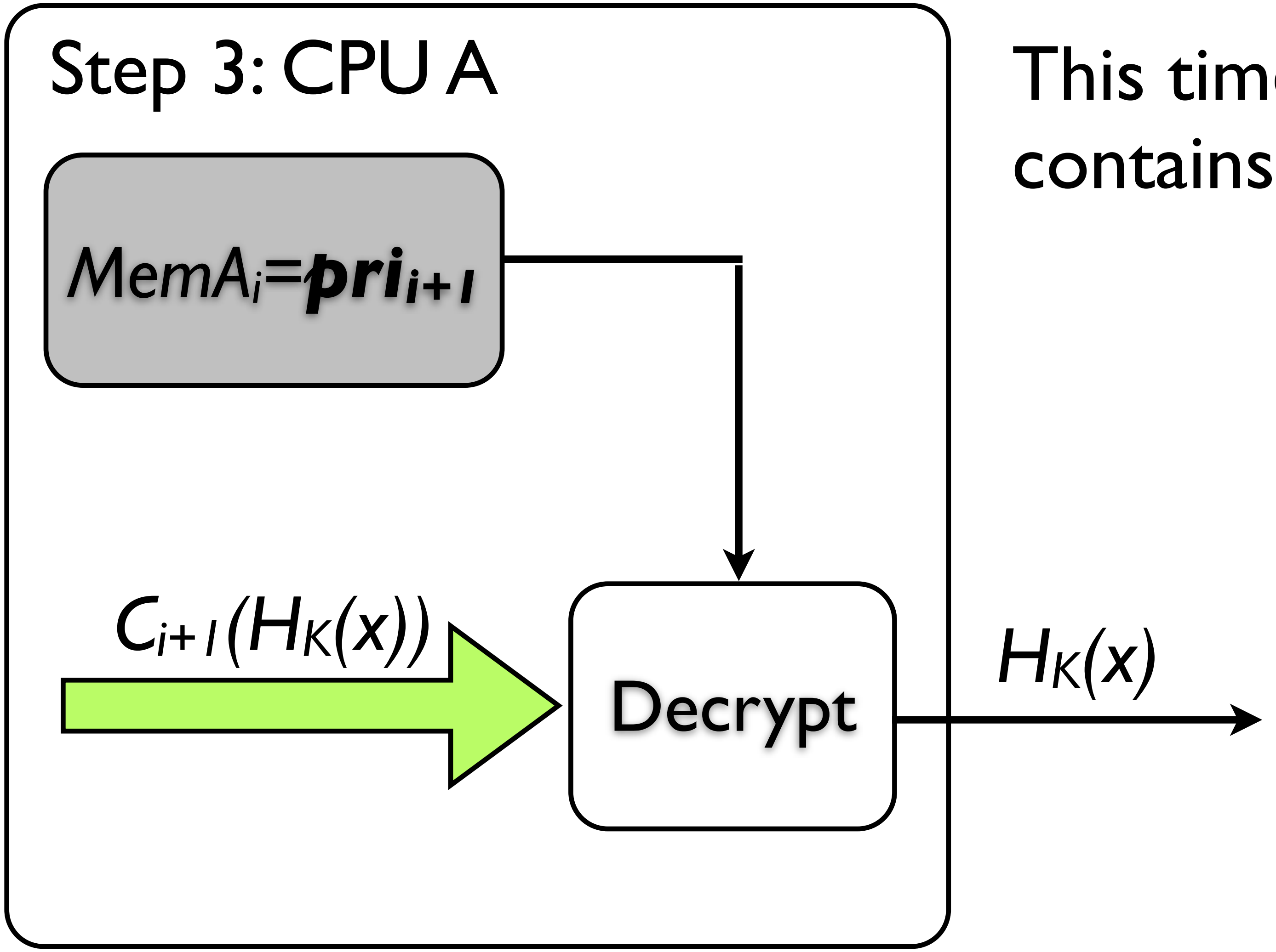


Second Attempt

Step 2': CPU B

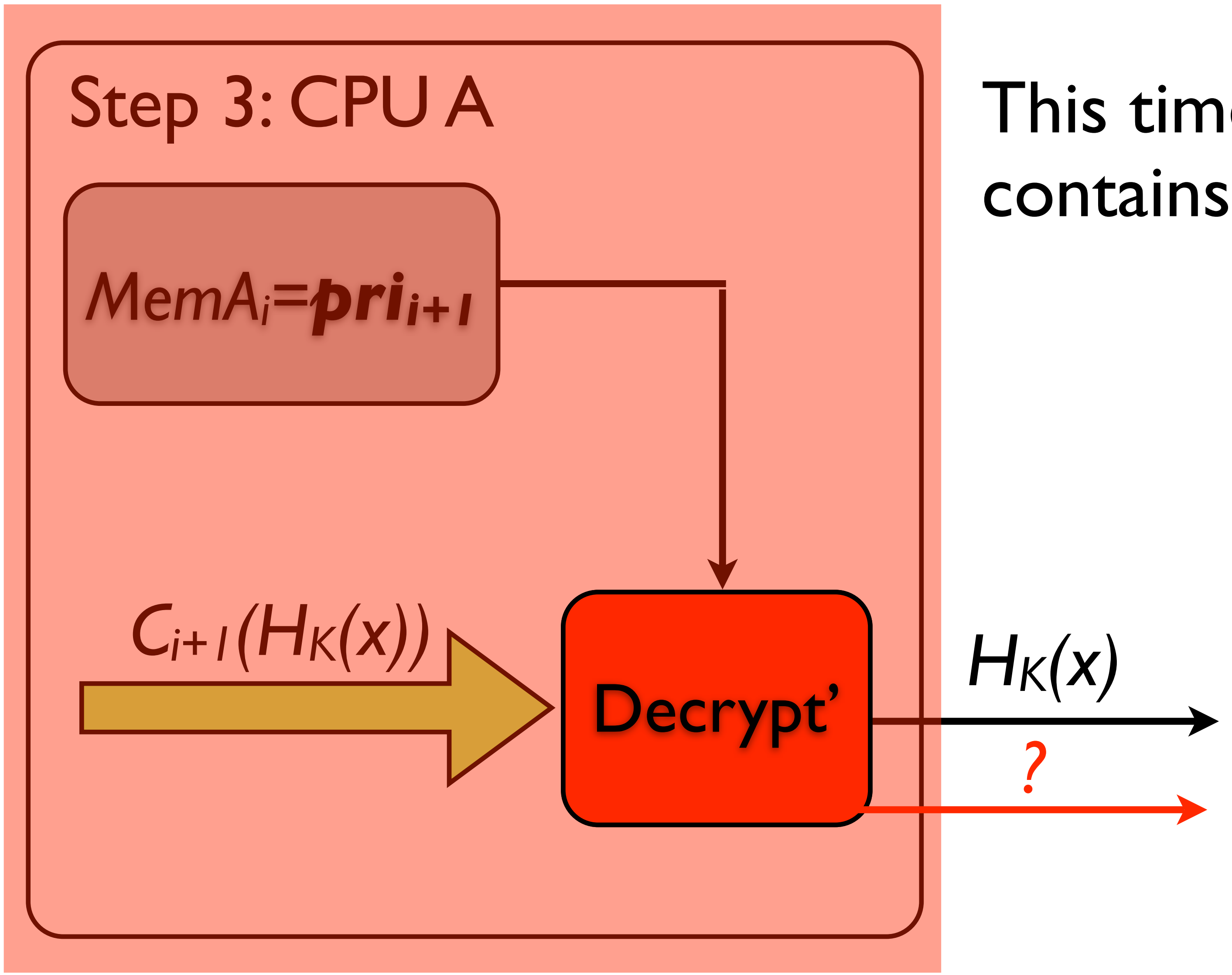


Second Attempt



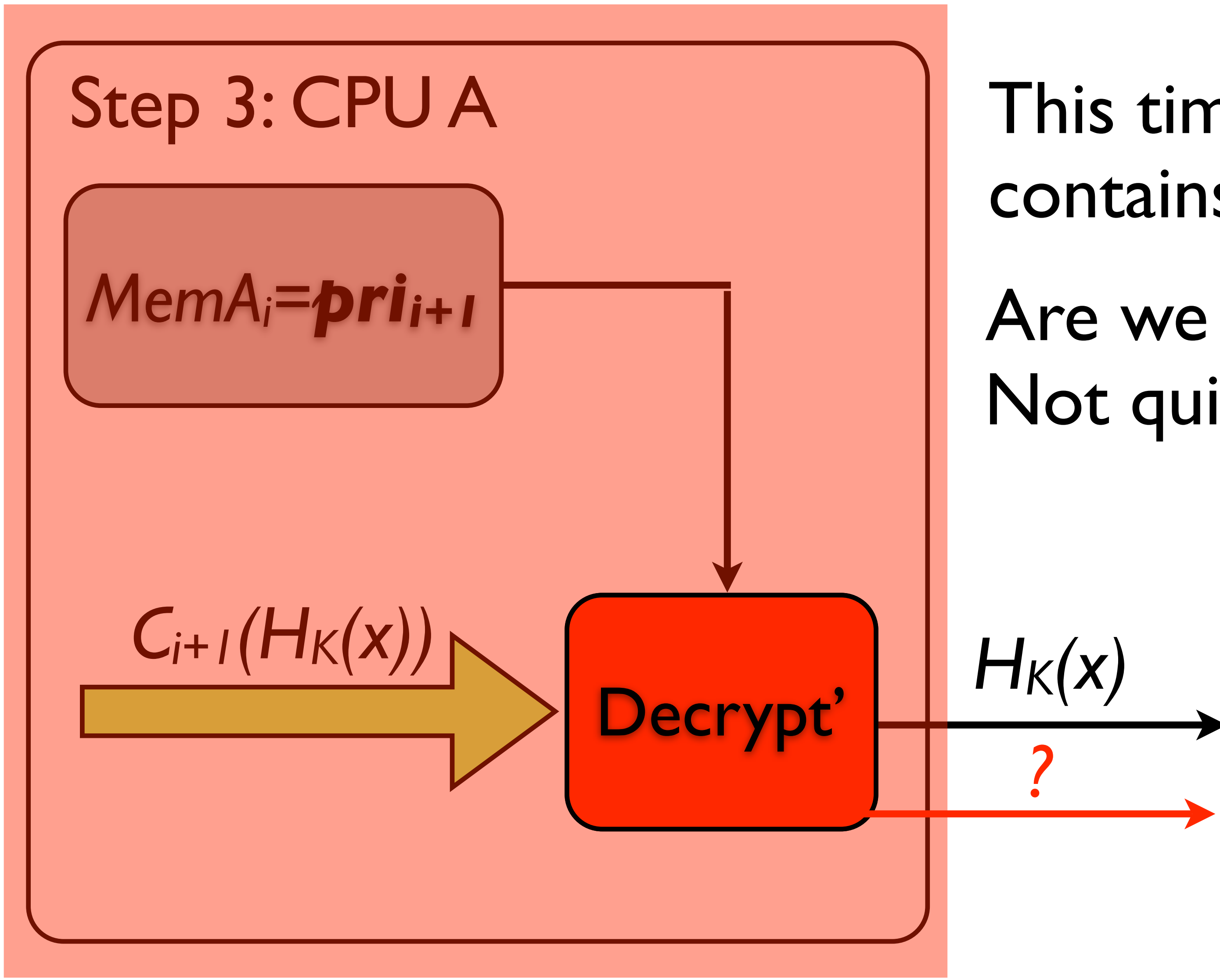
This time $C_{i+1}(H_K(x))$ only contains $H_K(x)$

Second Attempt



This time $C_{i+1}(H_K(x))$ only contains $H_K(x)$

Second Attempt

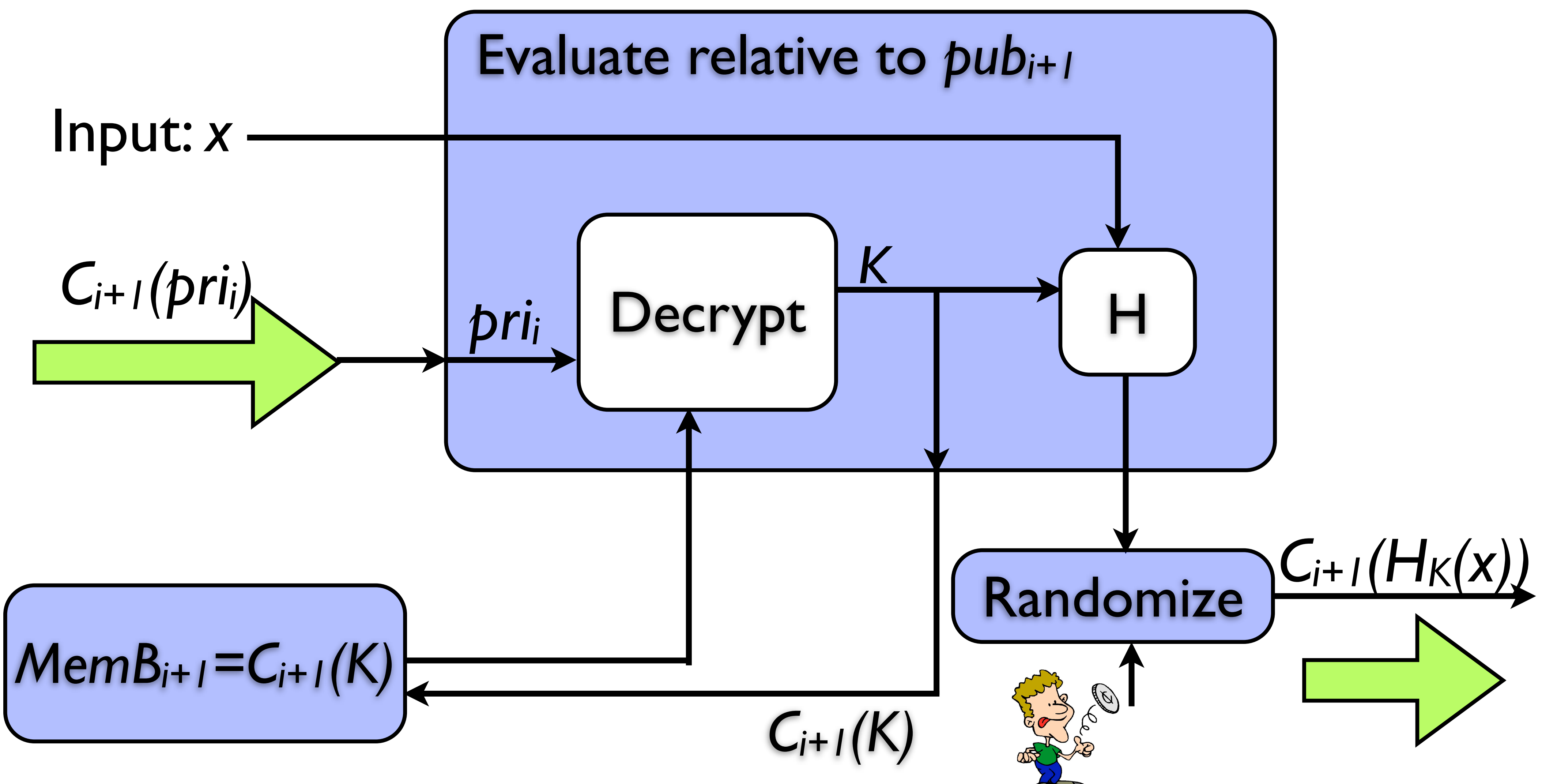


This time $C_{i+1}(H_K(x))$ only contains $H_K(x)$

Are we done?
Not quite...

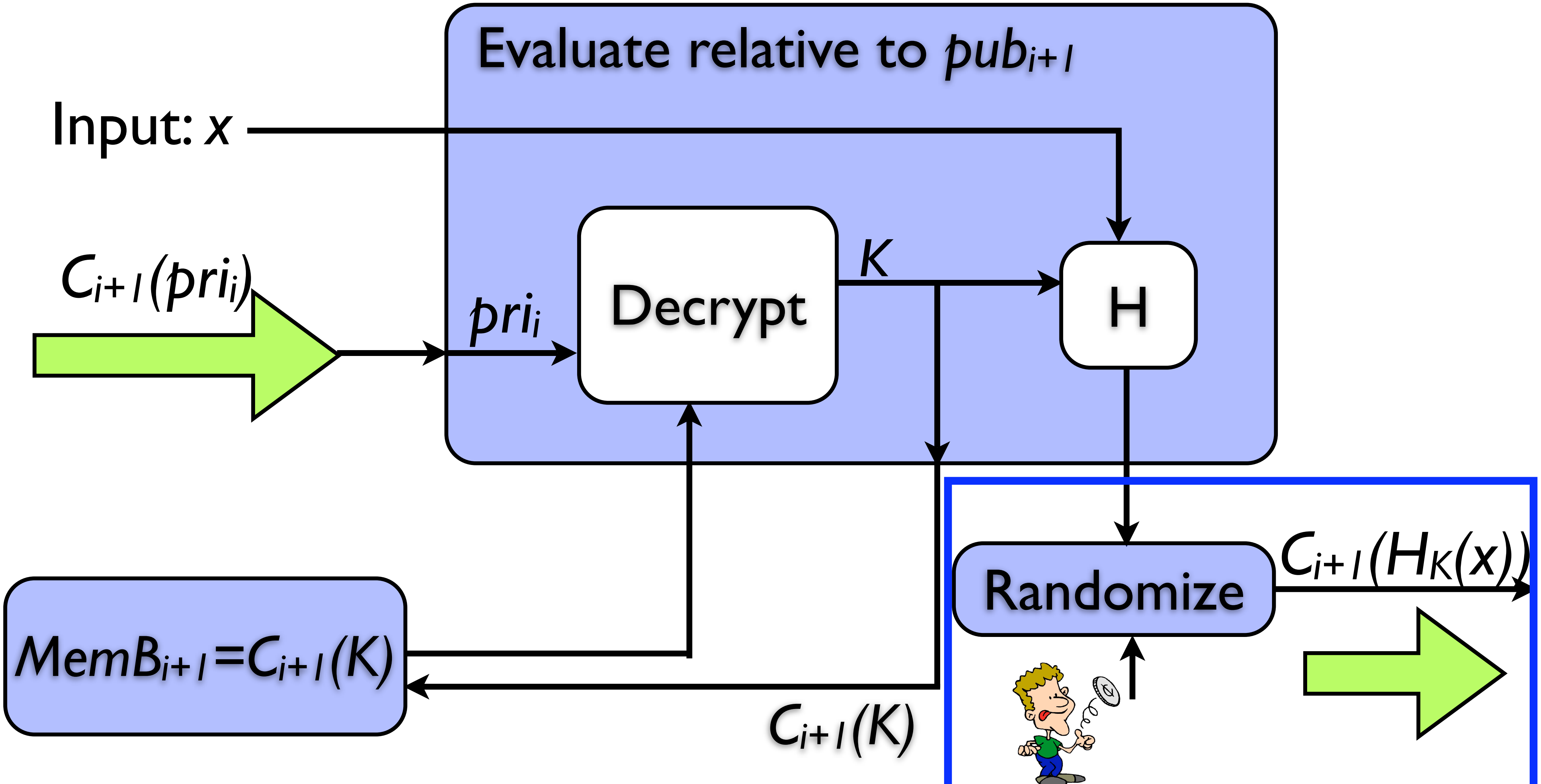
Second Attempt

Step 2': CPU B



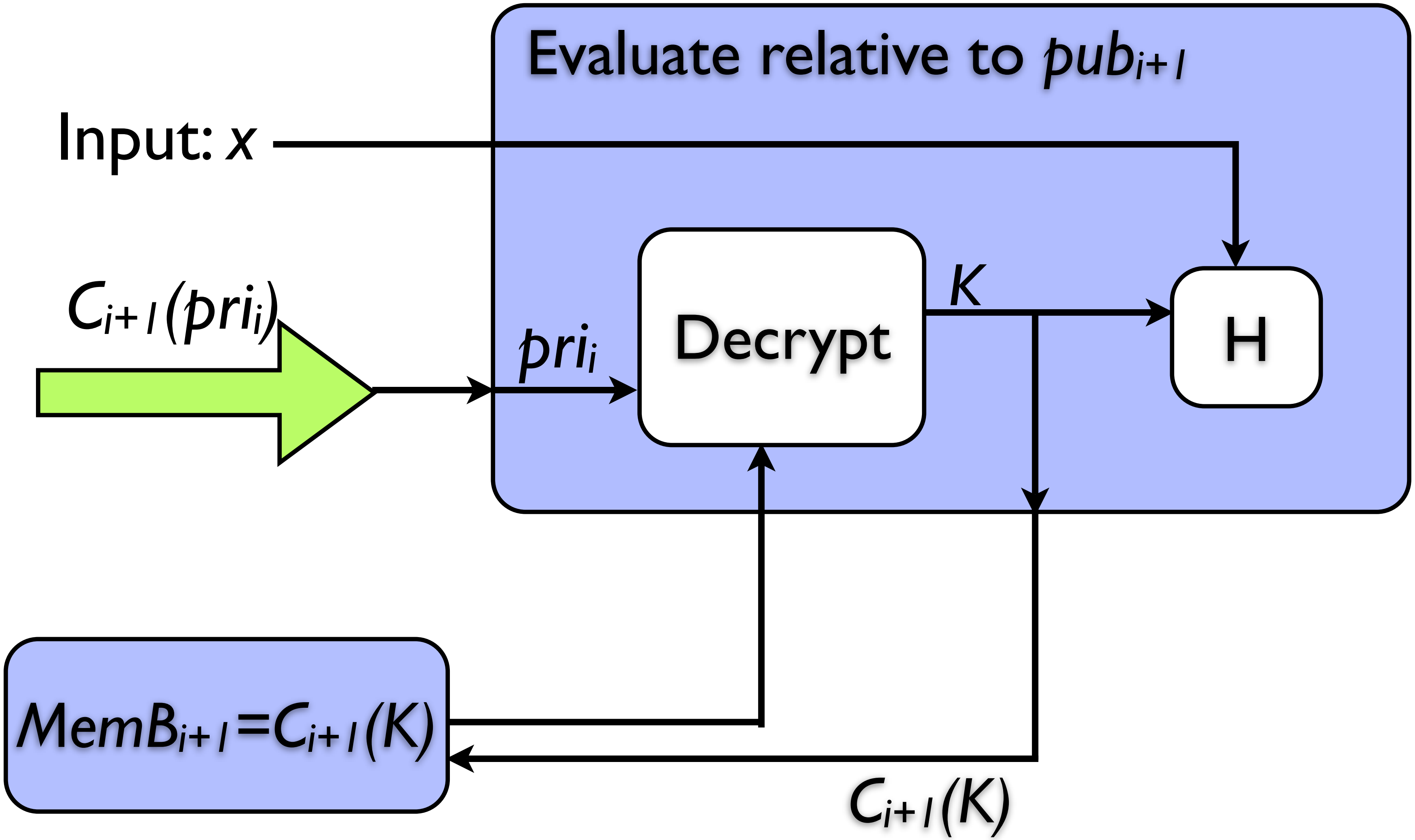
Second Attempt

Step 2': CPU B



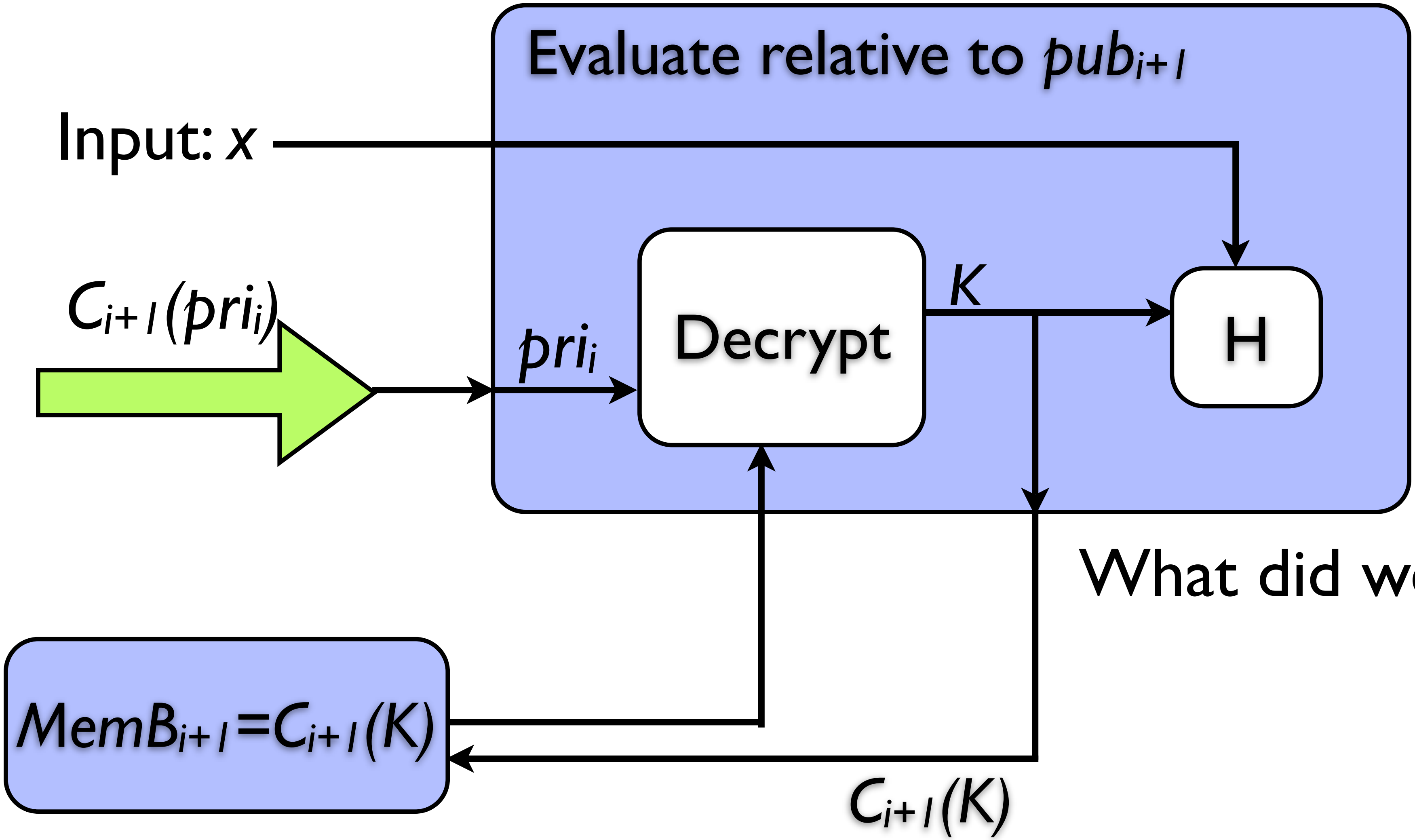
Second Attempt

Step 2': CPU B



Second Attempt

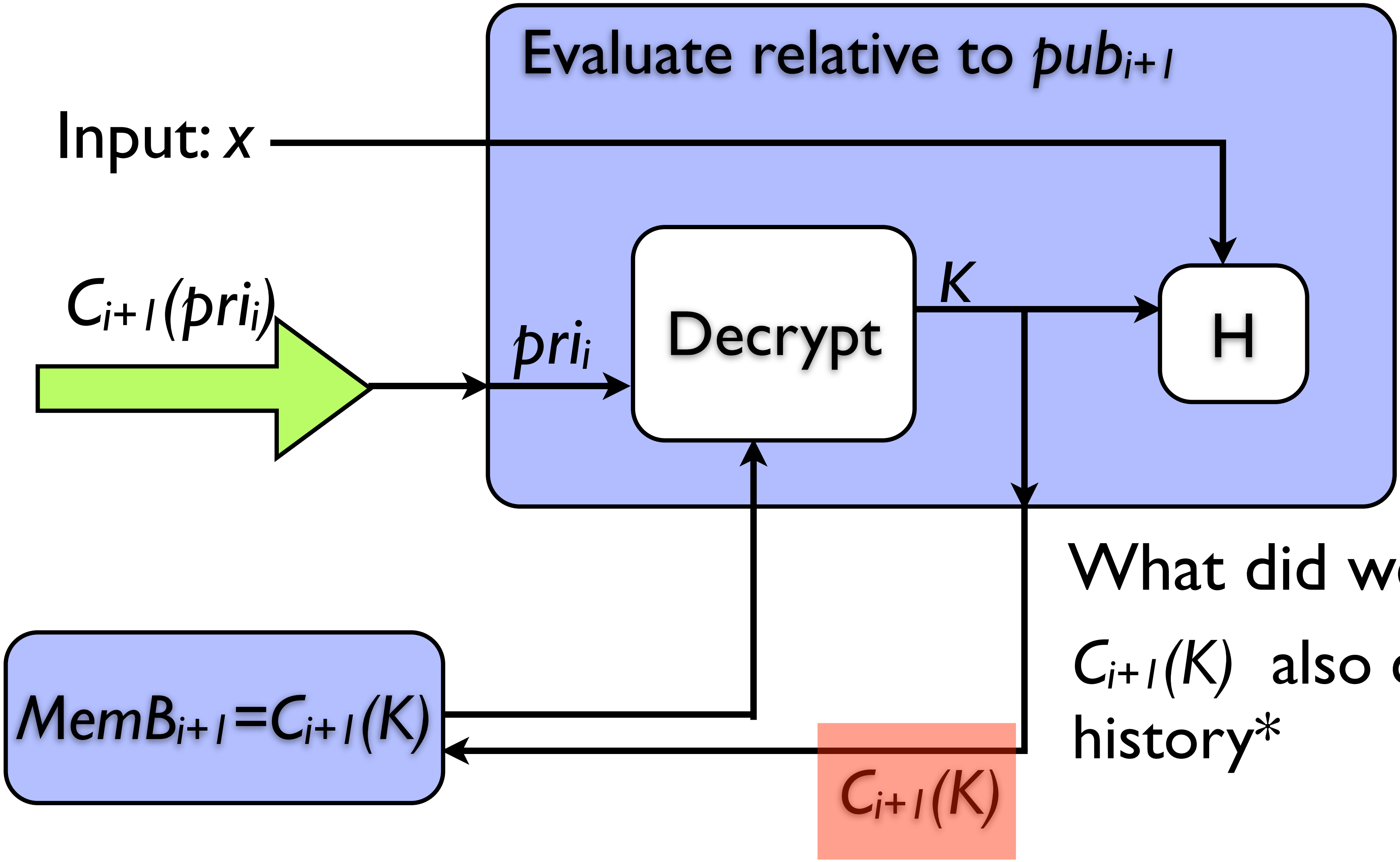
Step 2’: CPU B



What did we forget?

Second Attempt

Step 2': CPU B

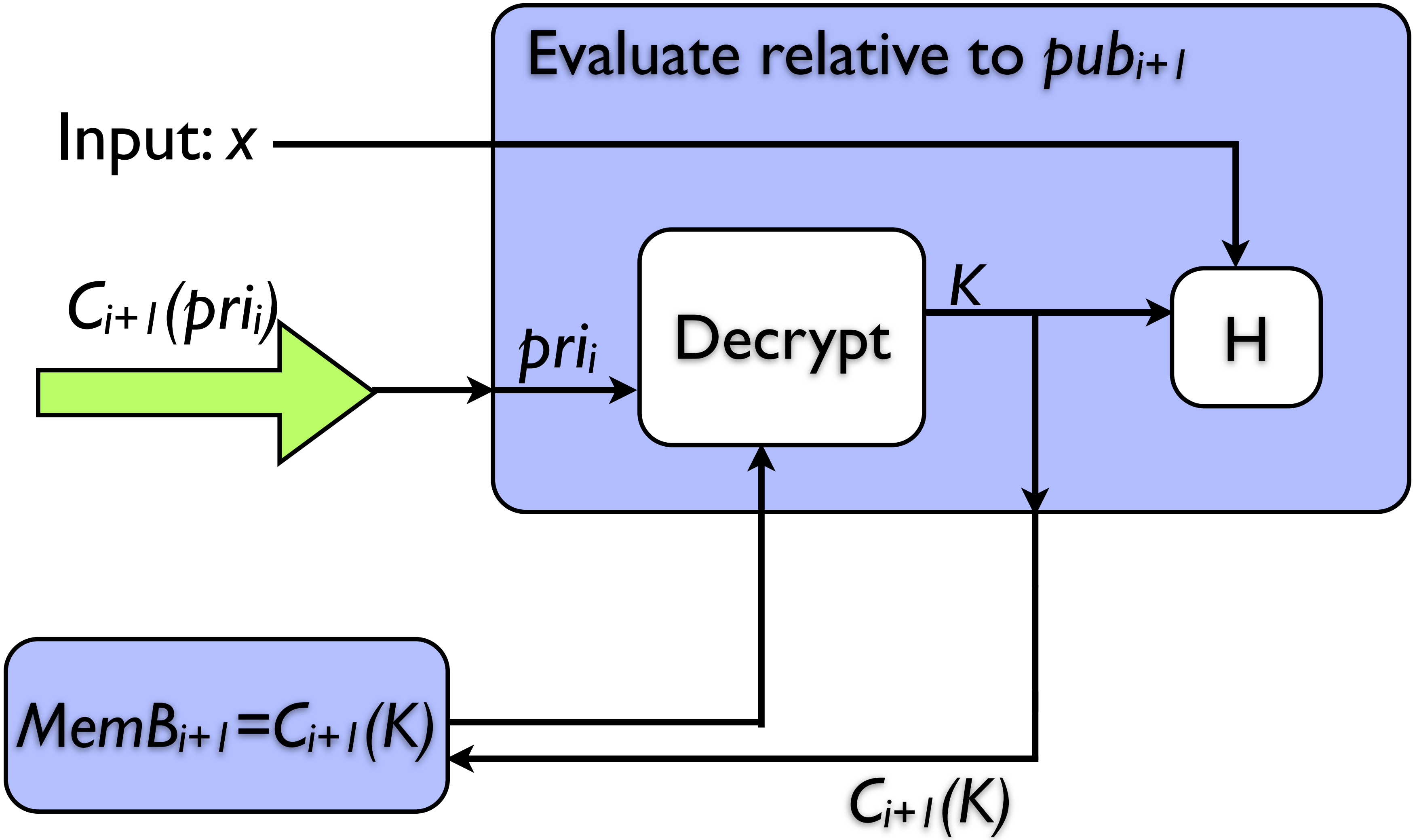


What did we forget?

$C_{i+1}(K)$ also carries history*

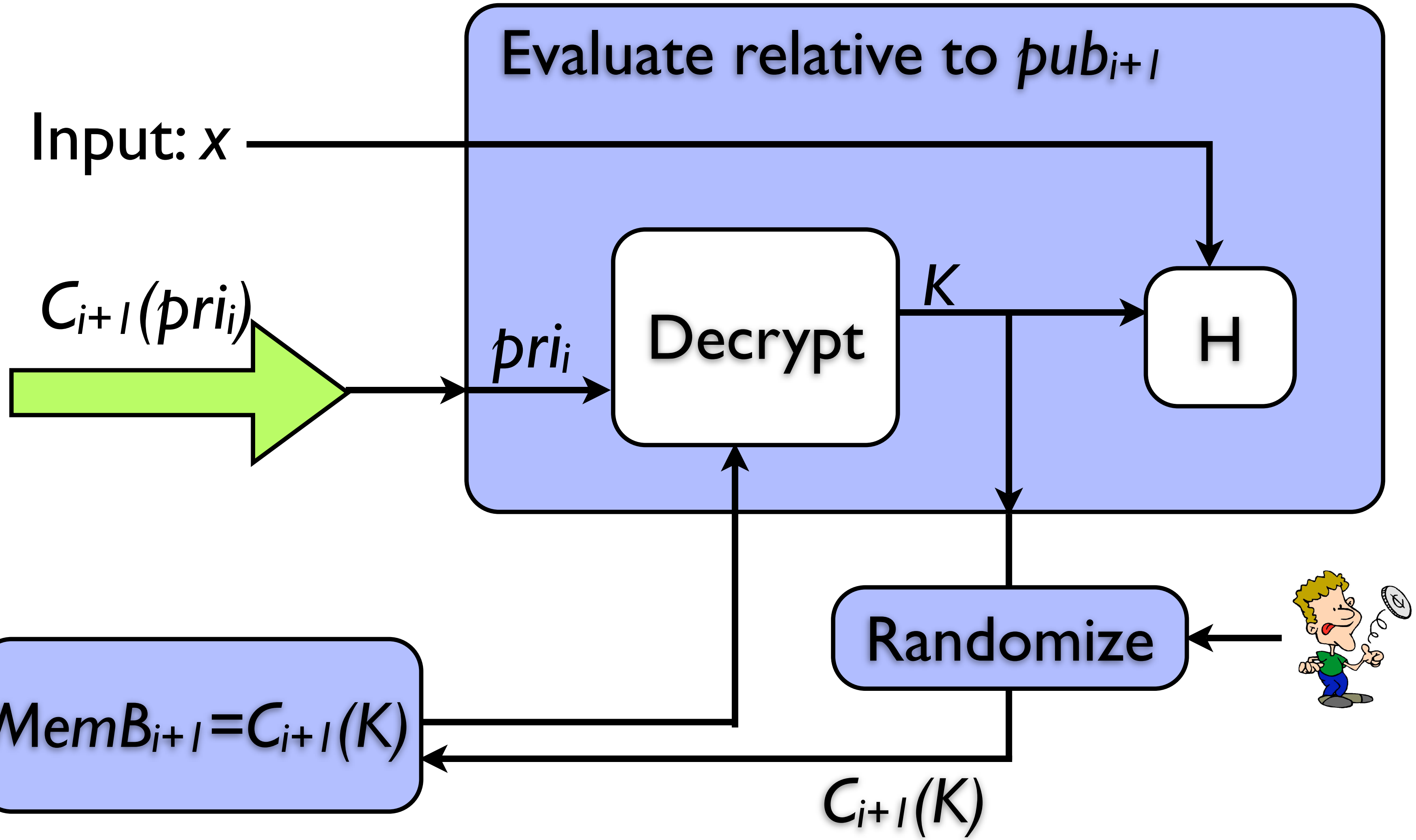
Third Attempt

Step 2'': CPU B



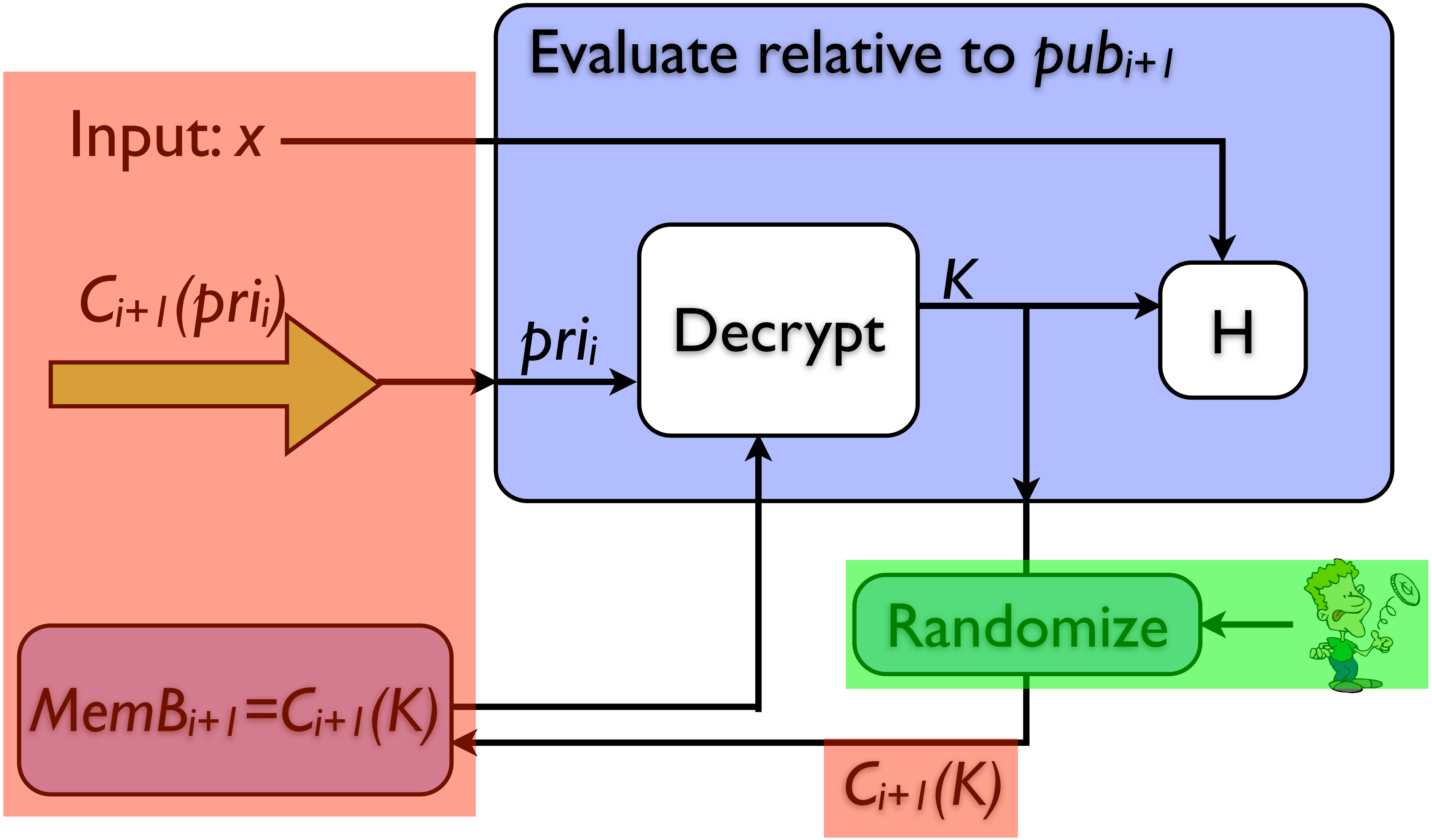
Third Attempt

Step 2’’: CPU B



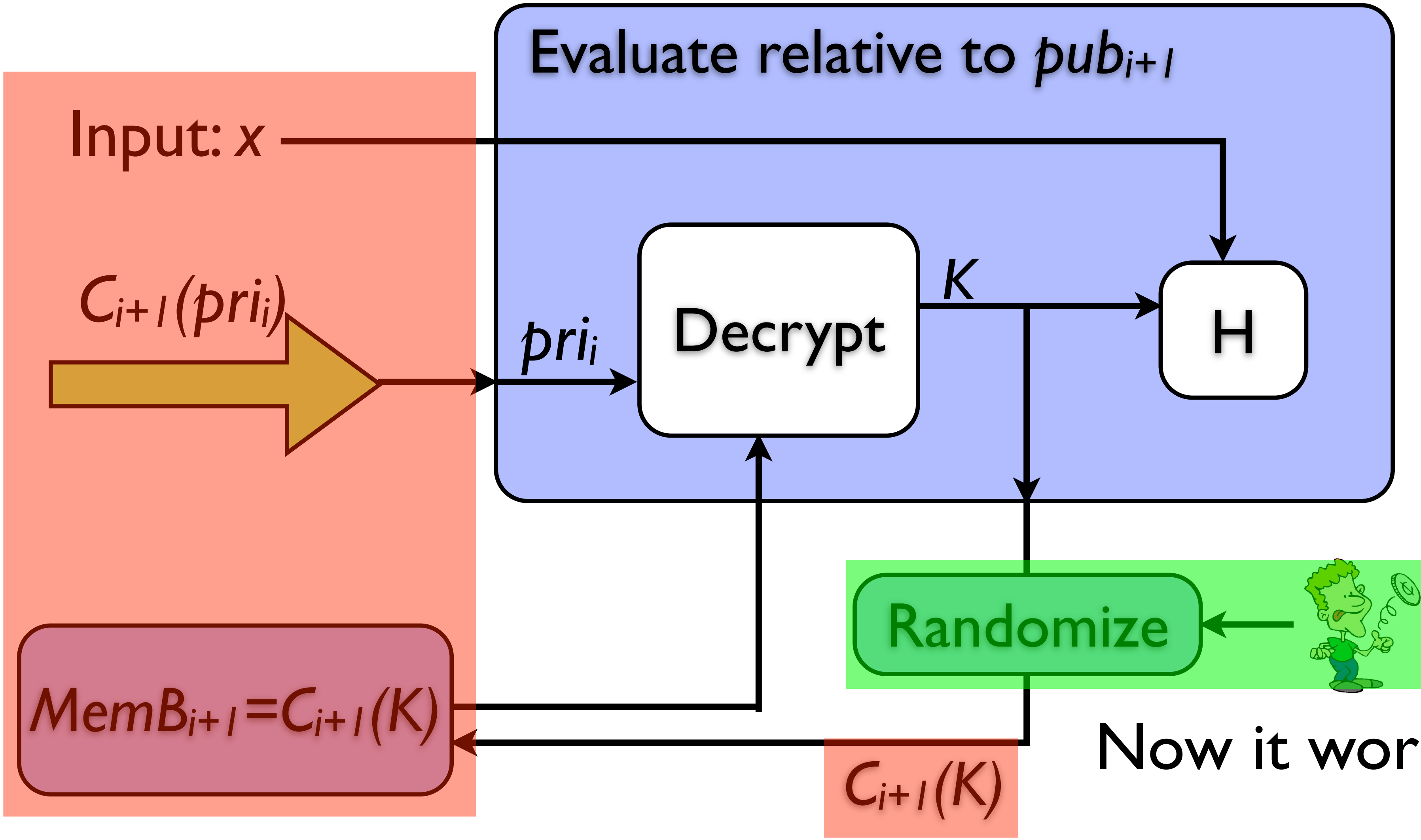
Third Attempt

Step 2'': CPU B



Third Attempt

Step 2'': CPU B



Now it works!

Complete Construction

Memory A: pri_i
Randomness: r_{gen}

Memory B: C_K
Randomness: r, r'

$pub_{i+1}, pri_{i+1} = KeyGen(r_{gen})$

$C_{pri} = Enc(pri_i, pub_{i+1}) \xrightarrow{C_{pri}}$

Set Memory A = pri_{i+1}

$C_{reply} = Evaluate(C_{pri}, C_K, x, H_K(x); r)$

$C'_K = Evaluate(C_{pri}, C_K, x, K; r')$

$\xleftarrow{C'_{reply}}$ $C'_{reply} = Randomize(C_{reply}; r)$

set Memory B = $Randomize(C'_K; r')$

$Y = Dec(C'_{reply}; pri_{i+1})$

Output Y

Complete Construction

Memory A: pri_i
Randomness: r_{gen}

Memory B: C_K
Randomness: r, r'

$pub_{i+1}, pri_{i+1} = KeyGen(r_{gen})$
$C_{pri} = Enc(pri_i, pub_{i+1}) \xrightarrow{C_{pri}}$
Set Memory A $= pri_{i+1}$

$C_{reply} = Evaluate(C_{pri}, C_K, x, H_K(x); r)$
$C'_K = Evaluate(C_{pri}, C_K, x, K; r')$

$\xleftarrow{C'_{reply}}$ $C'_{reply} = Randomize(C_{reply}\,; r)$
set Memory B $= Randomize(C'_K\,; r')$

$Y = Dec(C'_{reply}\,; pri_{i+1})$
Output Y

Proof

Hybrid 1

Memory A: pri_i
Randomness: r_{gen}

Memory B: C_K
Randomness: r, r'

$pub_{i+1}, pri_{i+1} = KeyGen(r_{gen})$

$C_{pri} = Enc(pri_i, pub_{i+1}) \xrightarrow{C_{pri}}$

Set Memory $A = pri_{i+1}$

$C_{reply} = Evaluate(C_{pri}, C_K, x, H_K(x); r)$

$C'_K = Evaluate(C_{pri}, C_K, x, K; r')$

$\xleftarrow{C'_{reply}}$ $C'_{reply} = Randomize(C_{reply}; r)$

set Memory $B = Randomize(C'_K; r')$

$Y = Dec(C'_{reply}; pri_{i+1})$

Output Y

Hybrid 1

Memory A: pri_i
Randomness: r_{gen}

Memory B: C_K
Randomness: r, r'

$pub_{i+1}, pri_{i+1} = KeyGen(r_{gen})$
$C_{pri} = Enc(pri_i, pub_{i+1})$ $\xrightarrow{C_{pri}}$
Set Memory $A = pri_{i+1}$

$C_{reply} = Evaluate(C_{pri}, C_K, x, H_K(x); r)$
$C'_K = Evaluate(C_{pri}, C_K, x, K; r')$

$\xleftarrow{C''_{reply}}$ $C'_{reply} = Randomize(C_{reply}; r)$
set Memory $B = Randomize(C'_K; r')$

$Y = Dec(C'_{reply}; pri_{i+1})$
Output Y

$C''_{reply} = Enc(H_K(x), pub_{i+1})$

Hybrid 1

Memory A: pri_i
Randomness: r_{gen}

Memory B: C_K
Randomness: r, r'

$pub_{i+1}, pri_{i+1} = KeyGen(r_{gen})$
$C_{pri} = Enc(pri_i, pub_{i+1})$ $\xrightarrow{C_{pri}}$
Set Memory $A = pri_{i+1}$

$C_{reply} = Evaluate(C_{pri}, C_K, x, H_K(x); r)$
$C'_K = Evaluate(C_{pri}, C_K, x, K; r')$

$\xleftarrow{C''_{reply}}$ $C'_{reply} = Randomize(C_{reply}\,; r)$
set Memory $B = Randomize(C'_K\,; r')$

$Y = Dec(C'_{reply}\,; pri_{i+1})$
Output Y

$C''_{reply} = Enc(H_K(x), pub_{i+1})$

Doesn't change the distribution

Hybrid 2

Memory A: pri_i
Randomness: r_{gen}

$pub_{i+1}, pri_{i+1} = KeyGen(r_{gen})$
$C_{pri} = Enc(pri_i, pub_{i+1})$ $\xrightarrow{C_{pri}}$
Set Memory A = pri_{i+1}

Memory B: C_K
Randomness: r, r'

$C_{reply} = Evaluate(C_{pri}, C_K, x, H_K(x); r)$
$C'_K = Evaluate(C_{pri}, C_K, x, K; r')$

$\xleftarrow{C''_{reply}}$ $C'_{reply} = Randomize(C_{reply}; r)$
set Memory B = $Randomize(C'_K; r')$

$Y = Dec(C'_{reply}; pri_{i+1})$
Output Y

$C''_{reply} = Enc(H_K(x), pub_{i+1})$

Hybrid 2

Memory A: pri_i
Randomness: r_{gen}

$pub_{i+1}, pri_{i+1} = KeyGen(r_{gen})$
$C_{pri} = Enc(pri_i, pub_{i+1})$ $\xrightarrow{\quad C_{pri} \quad}$
Set Memory $A = pri_{i+1}$

Memory B: C_K
Randomness: r, r'

$C_{reply} = Evaluate(C_{pri}, C_K, x, H_K(x); r)$
$C'_K = Evaluate(C_{pri}, C_K, x, K; r')$

$\xleftarrow{\quad C''_{reply} \quad}$ $C'_{reply} = Randomize(C_{reply}; r)$
set Memory $B = C''_K$

$Y = Dec(C'_{reply}; pri_{i+1})$
Output Y

$C''_{reply} = Enc(H_K(x), pub_{i+1})$

$C''_K = Enc(0...0, pub_{i+1})$

Hybrid 2

Memory A: pri_i
Randomness: r_{gen}

Memory B: C_K
Randomness: r, r'

$pub_{i+1}, pri_{i+1} = KeyGen(r_{gen})$
$C_{pri} = Enc(pri_i, pub_{i+1})$ $\xrightarrow{C_{pri}}$
Set Memory A $= pri_{i+1}$

$C_{reply} = Evaluate(C_{pri}, C_K, x, H_K(x); r)$
$C'_K = Evaluate(C_{pri}, C_K, x, K; r')$

$\xleftarrow{C''_{reply}}$ $C'_{reply} = Randomize(C_{reply}; r)$
set Memory B $= C''_K$

$Y = Dec(C'_{reply}; pri_{i+1})$
Output Y

$C''_{reply} = Enc(H_K(x), pub_{i+1})$
$C''_K = Enc(0...0, pub_{i+1})$

Changes the distribution completely

Why Should This Work?

- Very informally: Ciphertexts are incompressible.
  - This means that leakage on B can help only if Adv knows enough about *pri*
  - But Adv sees only leakage on *pri* which is insufficient to break semantic security

Open Questions

- Can we get rid of the leak-free component?
- Granularity of leakage.

Thank you!