MacORAMa:
Optimal Oblivious RAM with Integrity

Crypto 2023

Surya Mathialagan
MIT

Neekon Vafa
MIT

Remote RAM Computation

Remote RAM Computation

- User wants to perform RAM computation, but doesn't have enough local space.

Remote RAM Computation

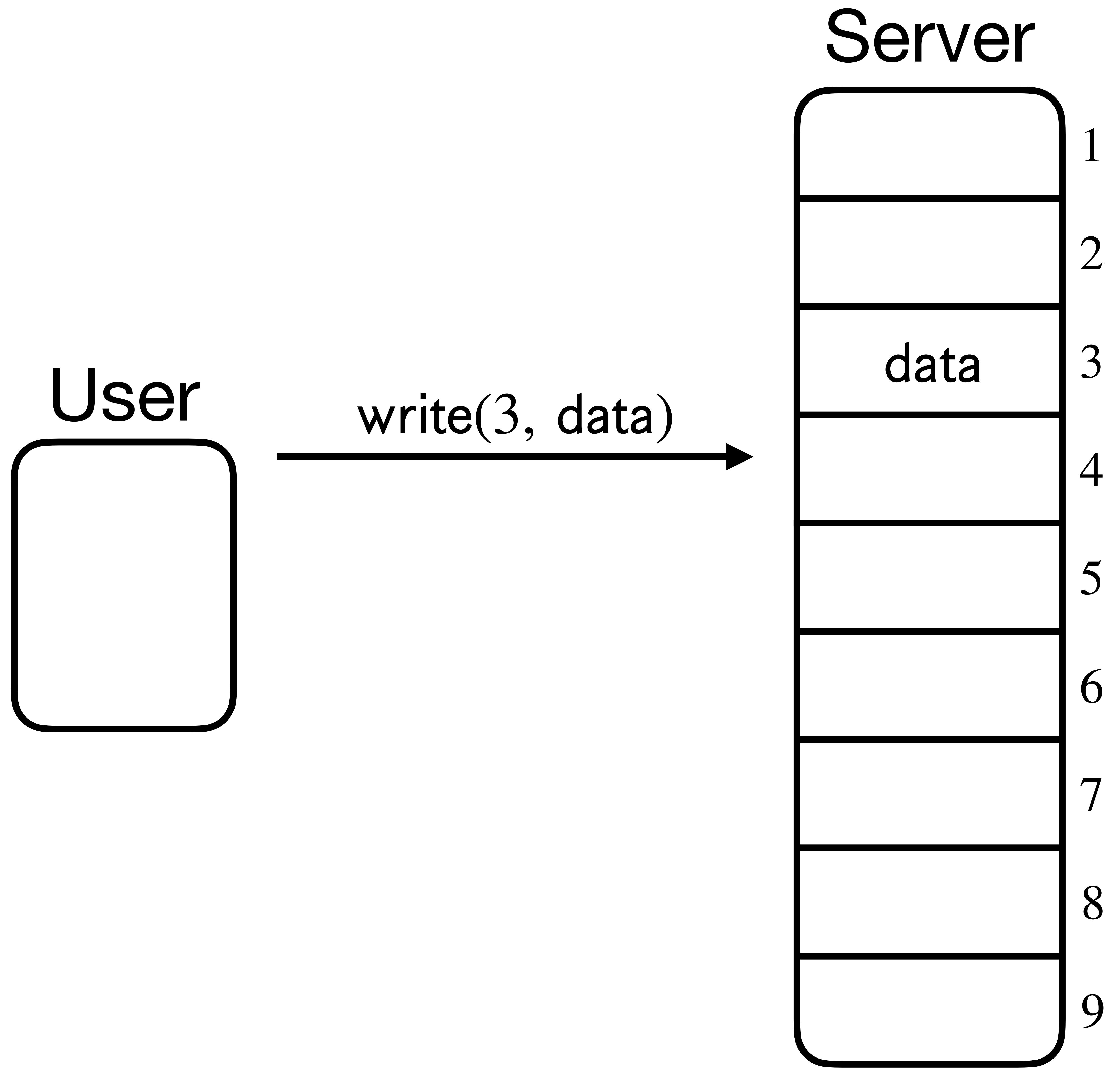
- User wants to perform RAM computation, but doesn't have enough local space.

- Solution: Use remote RAM server.

Remote RAM Computation

- User wants to perform RAM computation, but doesn't have enough local space.

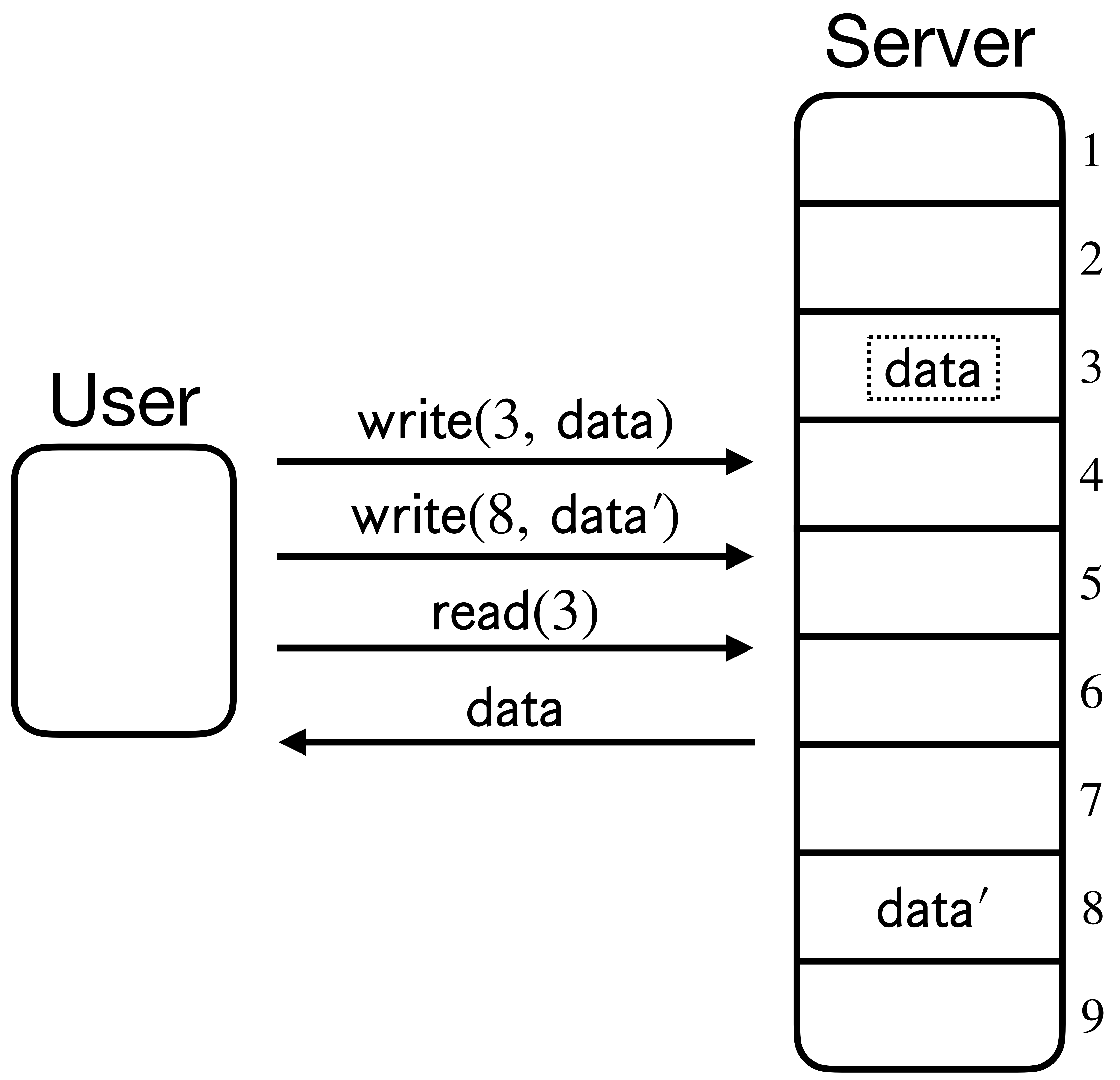
- Solution: Use remote RAM server.



Remote RAM Computation

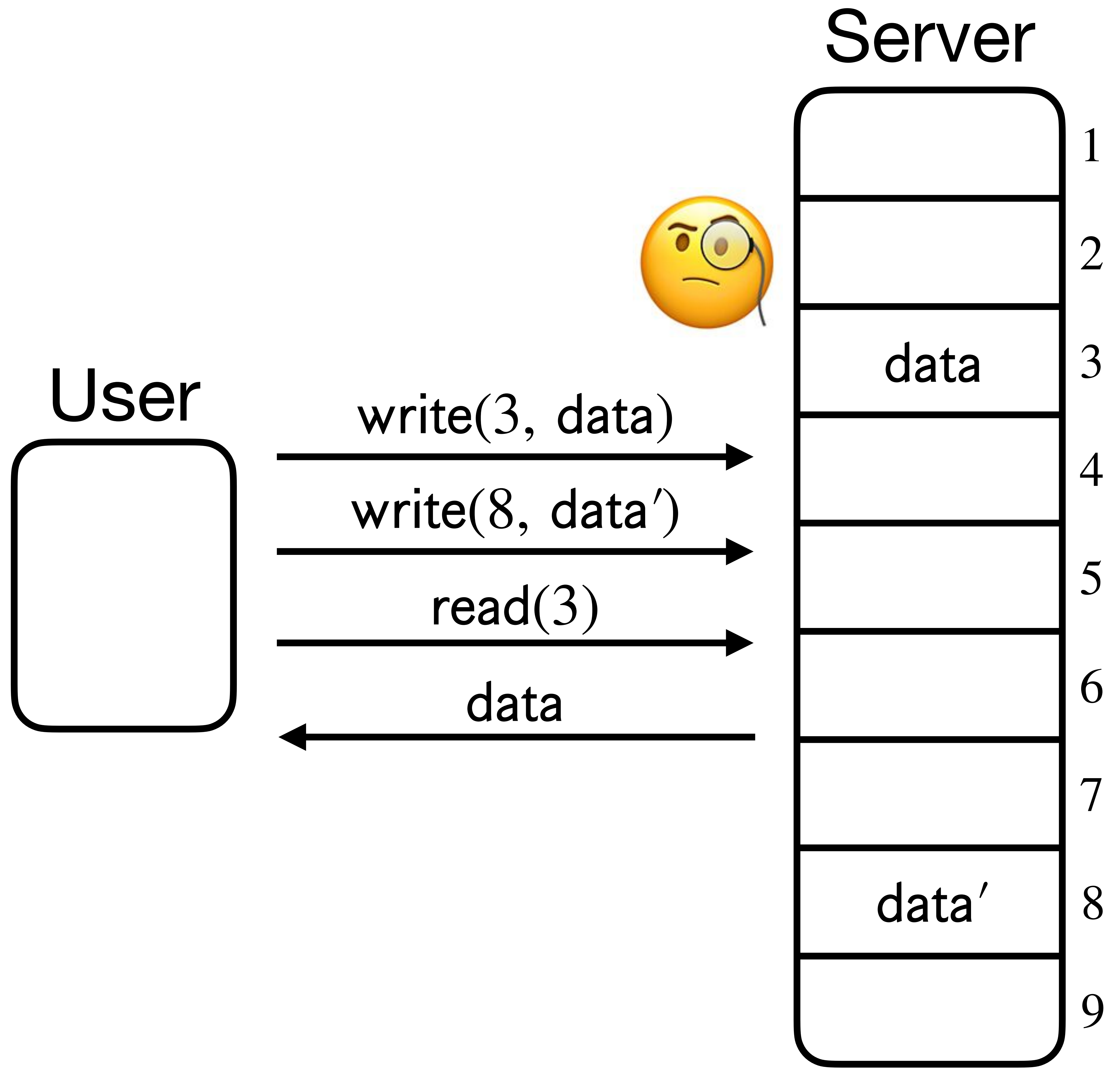
- User wants to perform RAM computation, but doesn't have enough local space.

- Solution: Use remote RAM server.



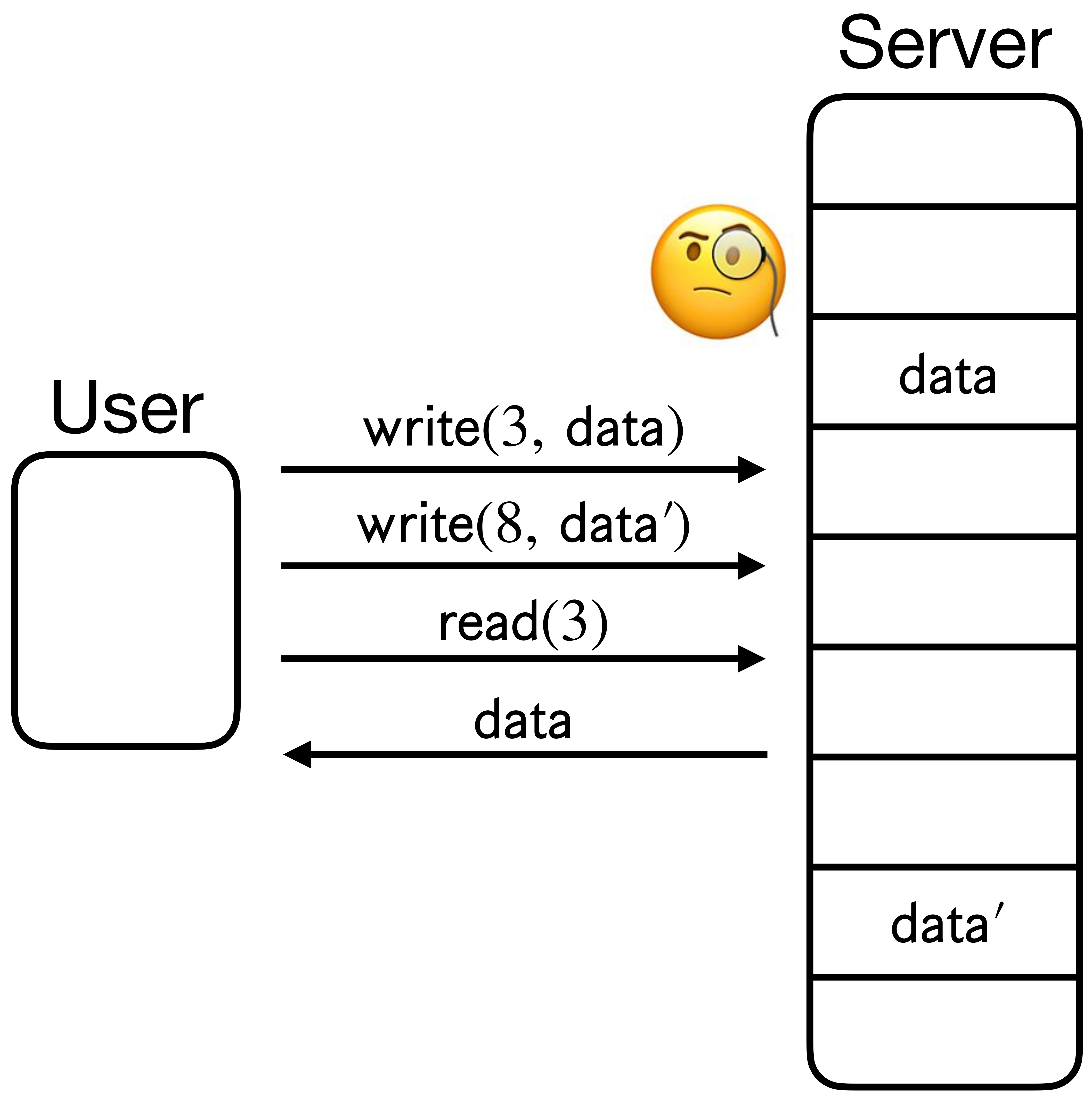
Remote RAM Computation

- User wants to perform RAM computation, but doesn't have enough local space.
- Solution: Use remote RAM server.
- How can the user ensure privacy of its computation against a curious server?



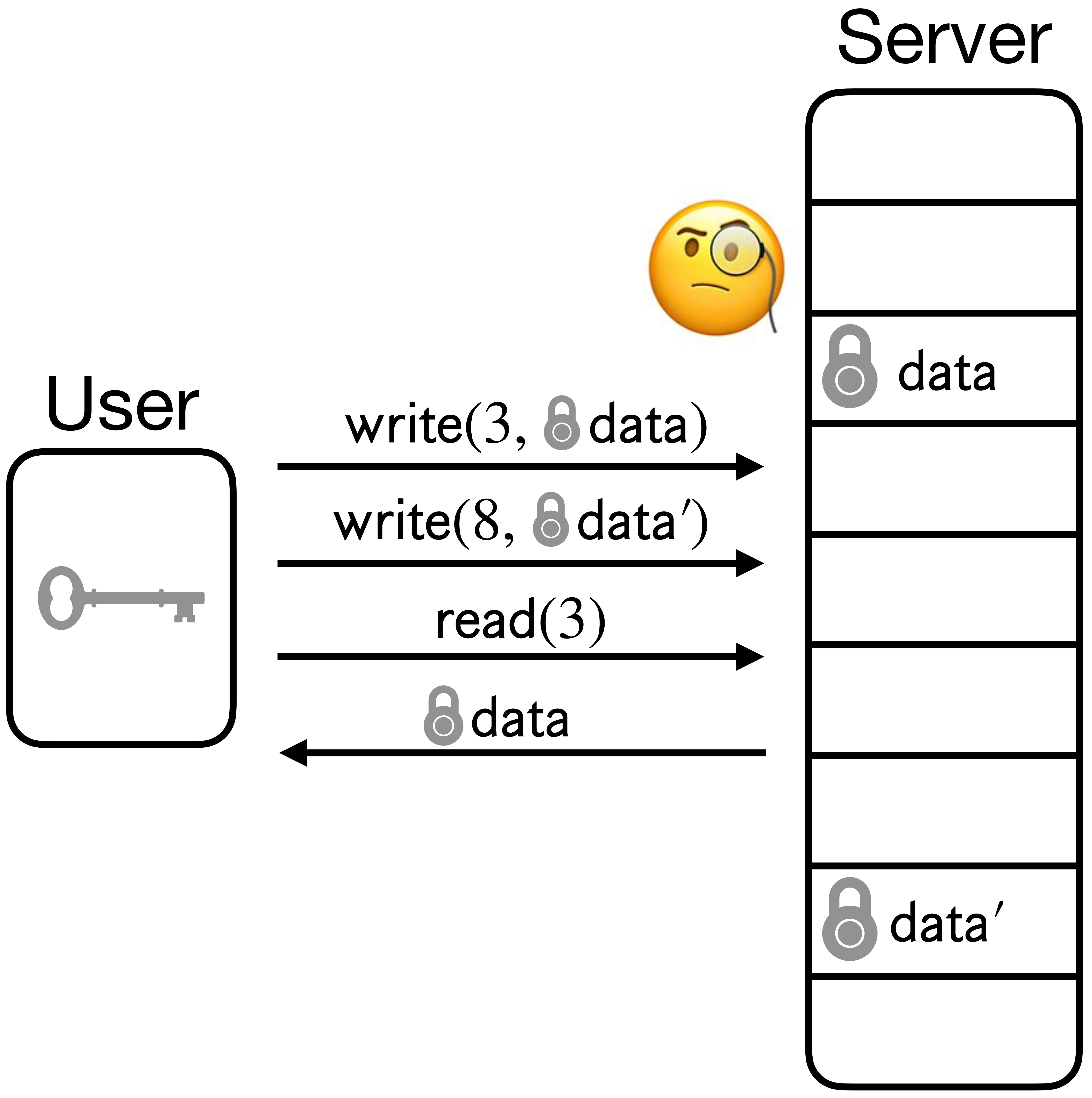
Remote RAM Computation

- One idea to ensure privacy: Encrypt the data (private key)



Remote RAM Computation

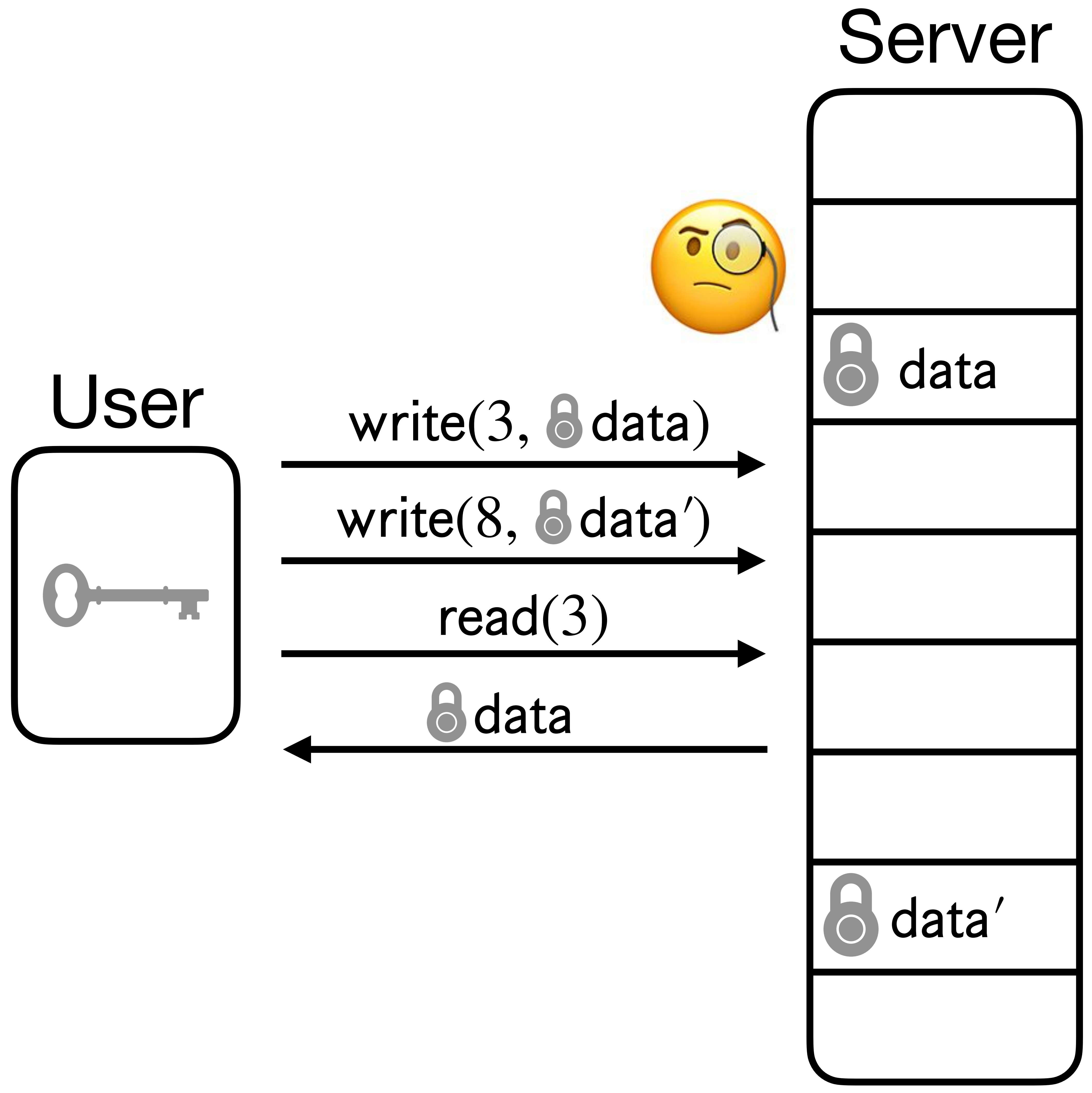
- One idea to ensure privacy: Encrypt the data (private key)



Remote RAM Computation

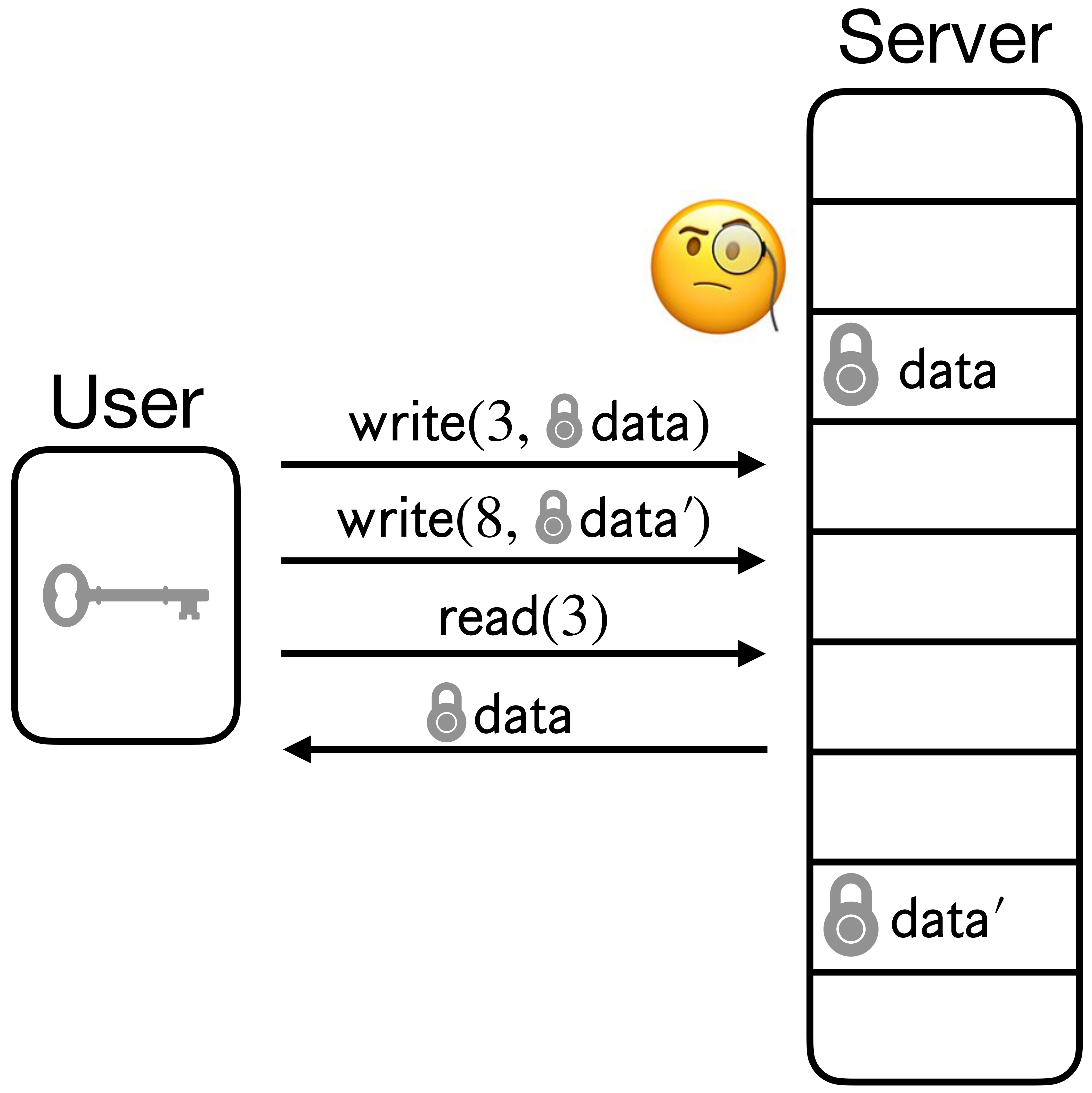
- One idea to ensure privacy: Encrypt the data (private key)

- Problem: Encryption is insufficient (access patterns reveal private information!)



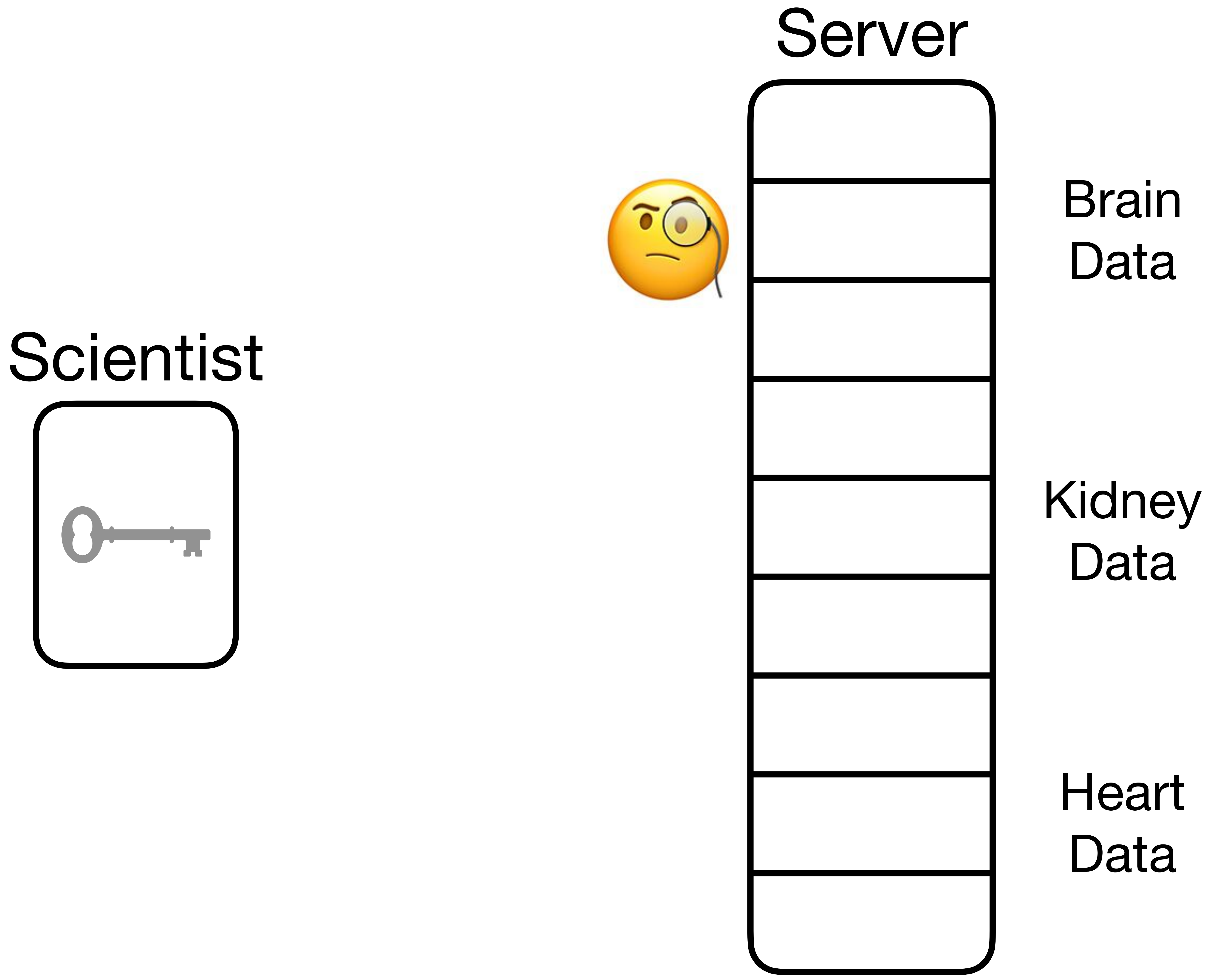
Remote RAM Computation

- One idea to ensure privacy: Encrypt the data (private key)
- Problem: Encryption is insufficient (access patterns reveal private information!)
- Example: Medical study



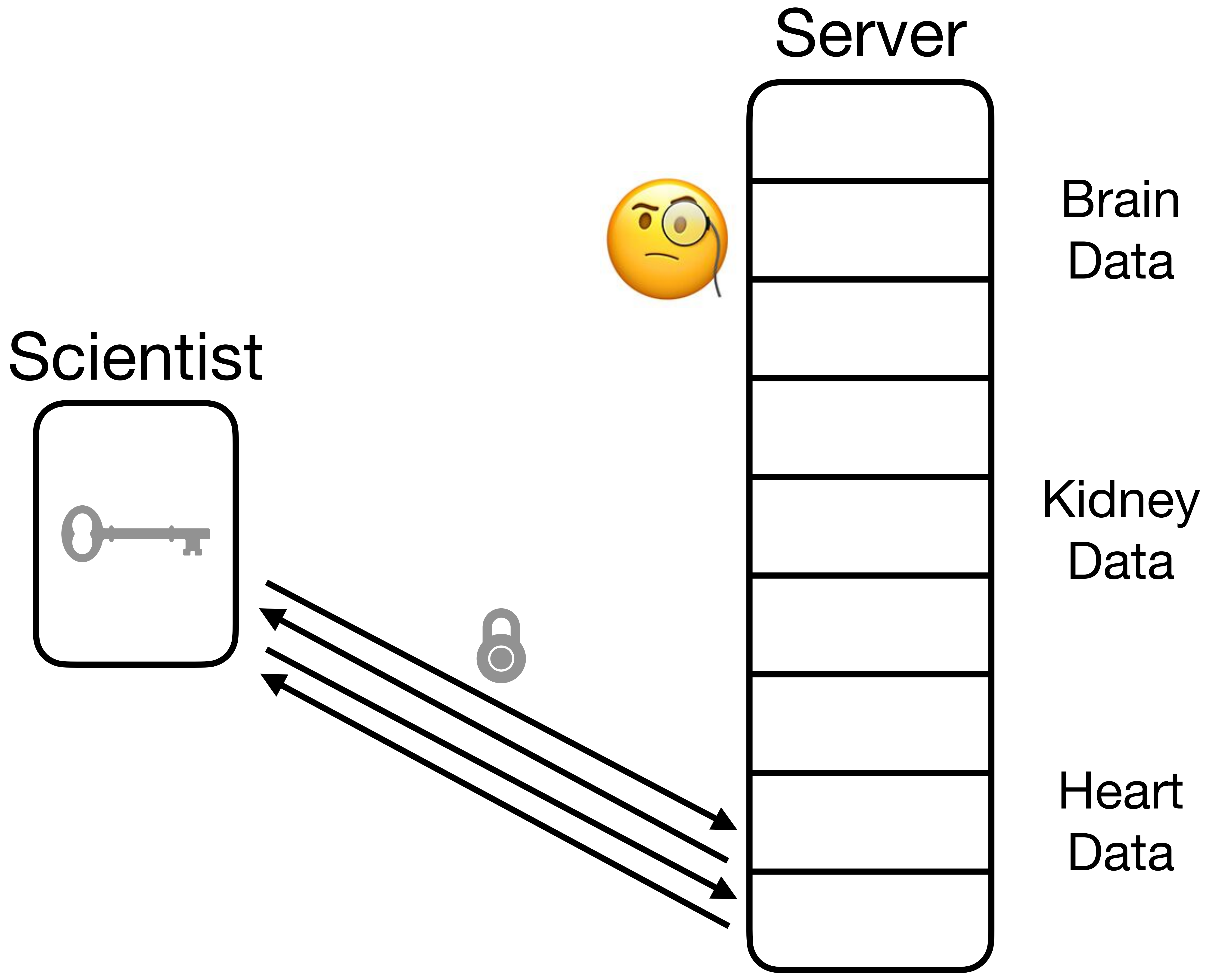
Remote RAM Computation

- One idea to ensure privacy: Encrypt the data (private key)
- Problem: Encryption is insufficient (access patterns reveal private information!)
- Example: Medical study



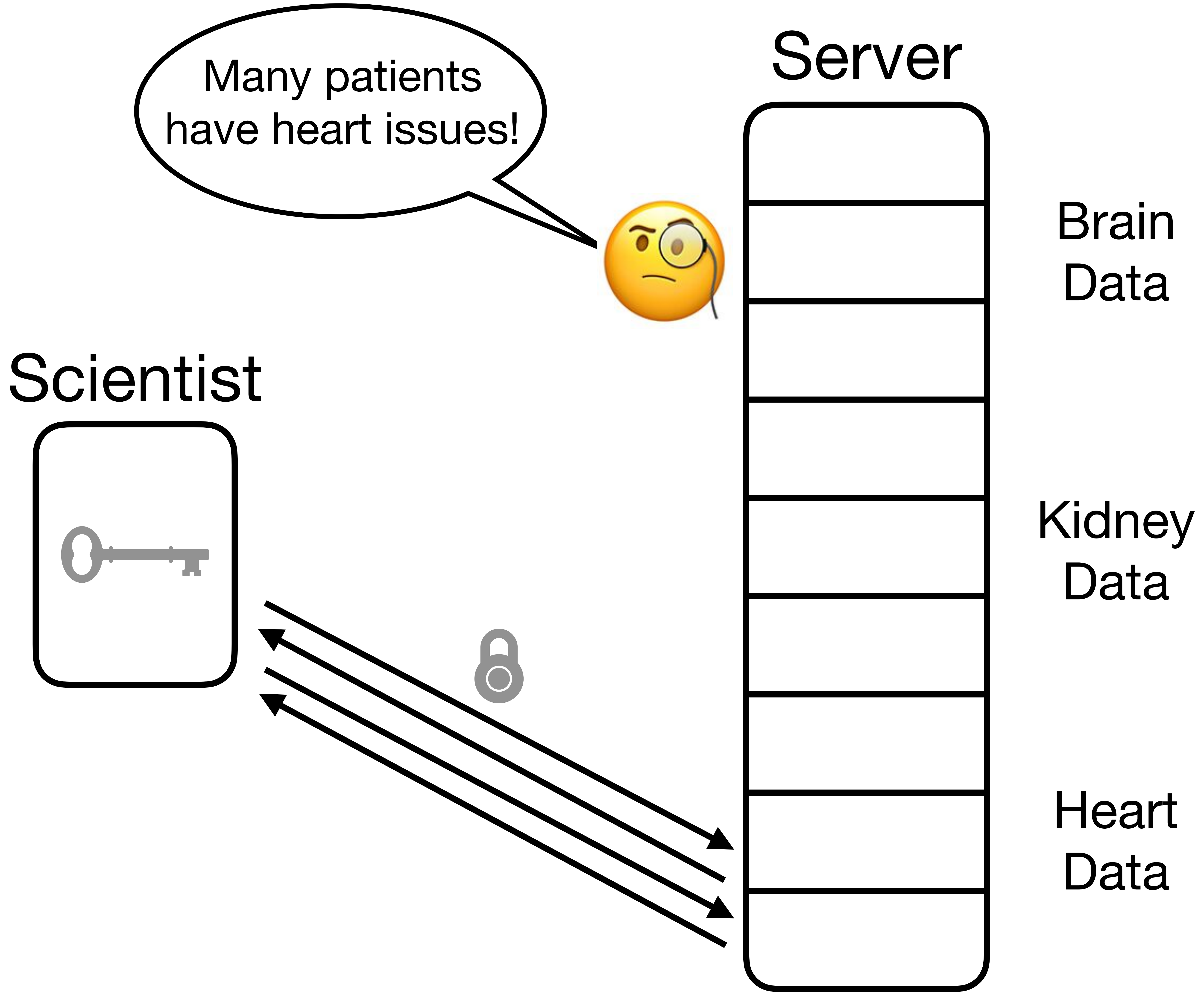
Remote RAM Computation

- One idea to ensure privacy: Encrypt the data (private key)
- Problem: Encryption is insufficient (access patterns reveal private information!)
- Example: Medical study



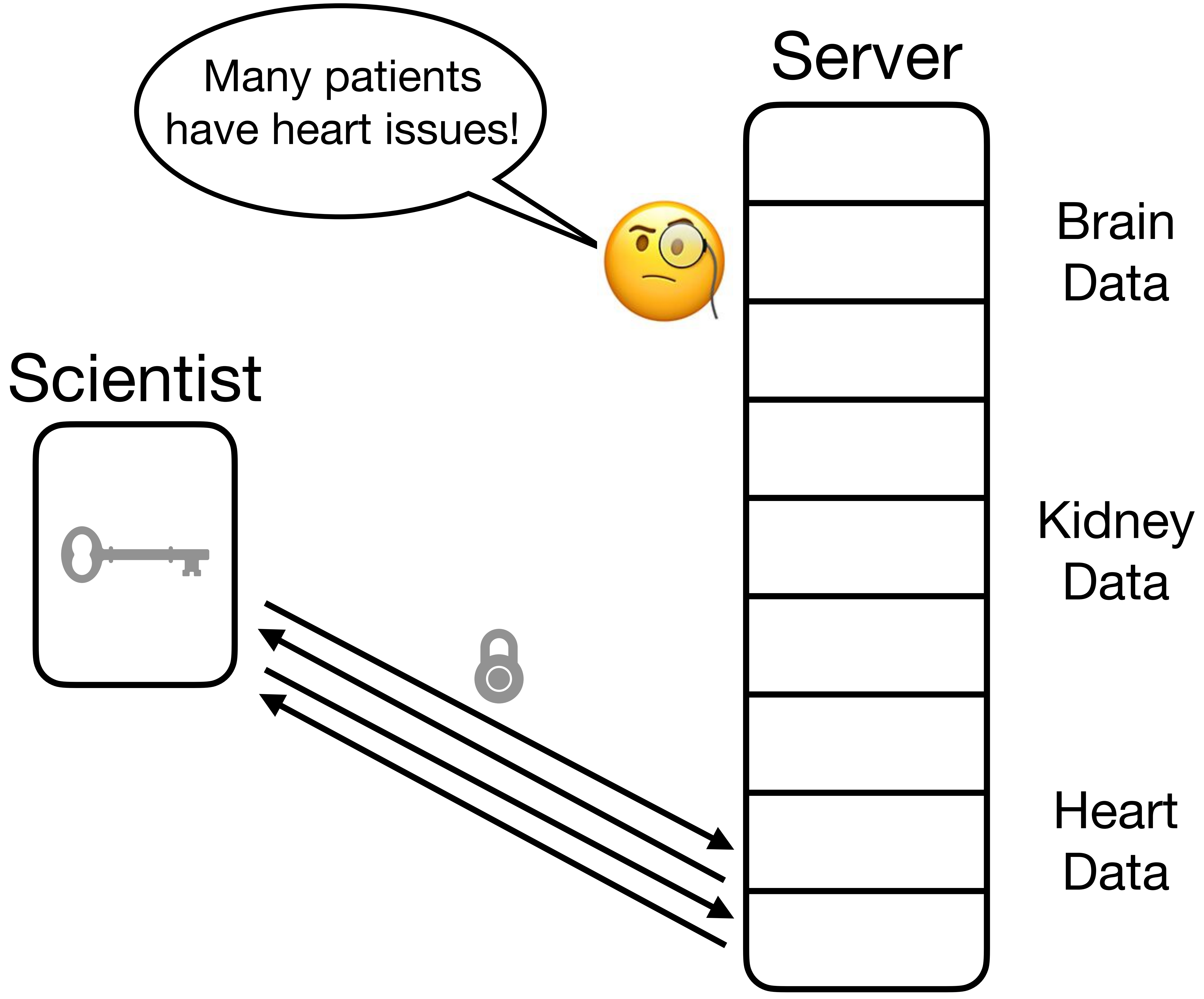
Remote RAM Computation

- One idea to ensure privacy: Encrypt the data (private key)
- Problem: Encryption is insufficient (access patterns reveal private information!)
- Example: Medical study



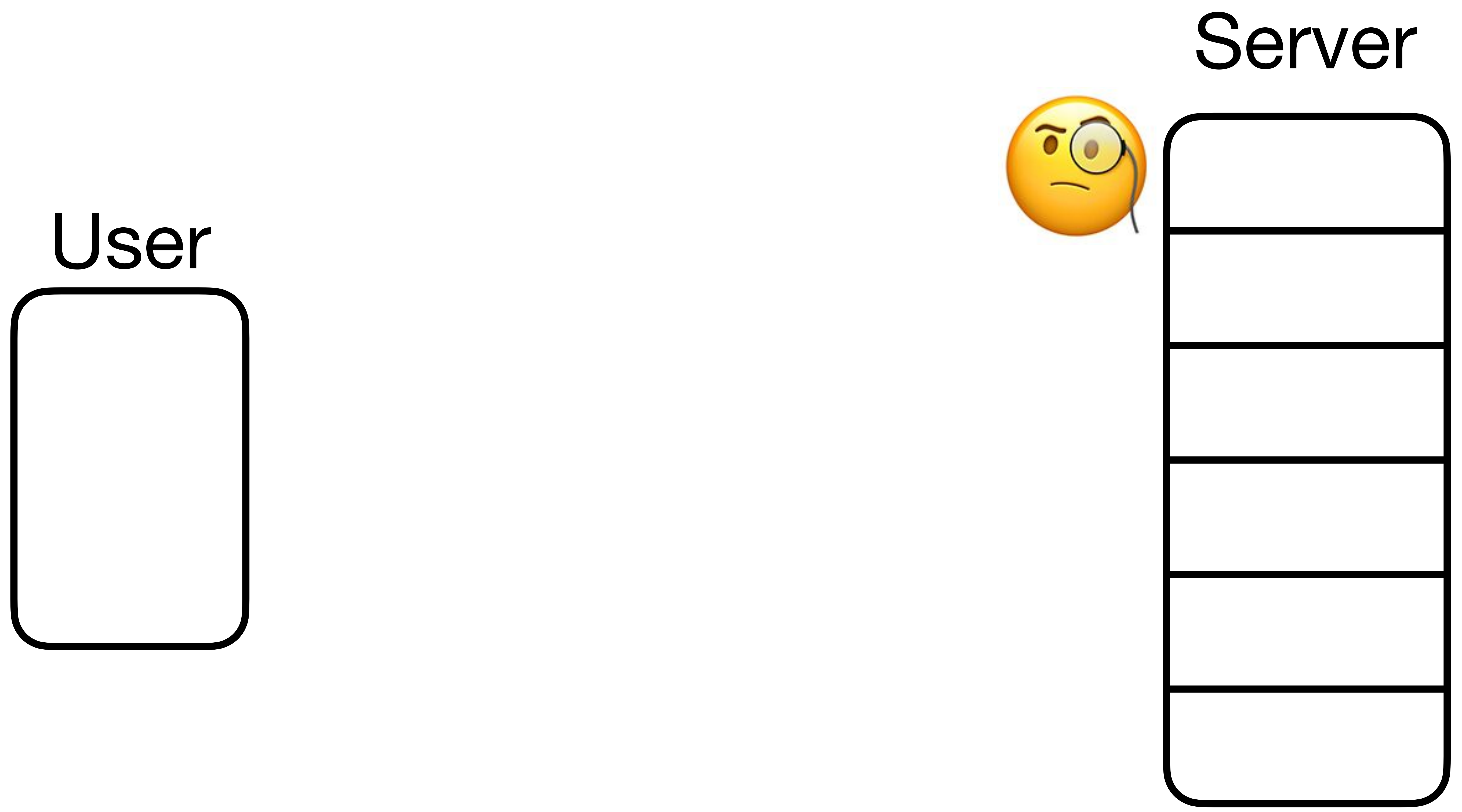
Remote RAM Computation

- One idea to ensure privacy: Encrypt the data (private key)
- Problem: Encryption is insufficient (access patterns reveal private information!)
- Example: Medical study
- RAM addresses in accesses can reveal private information!



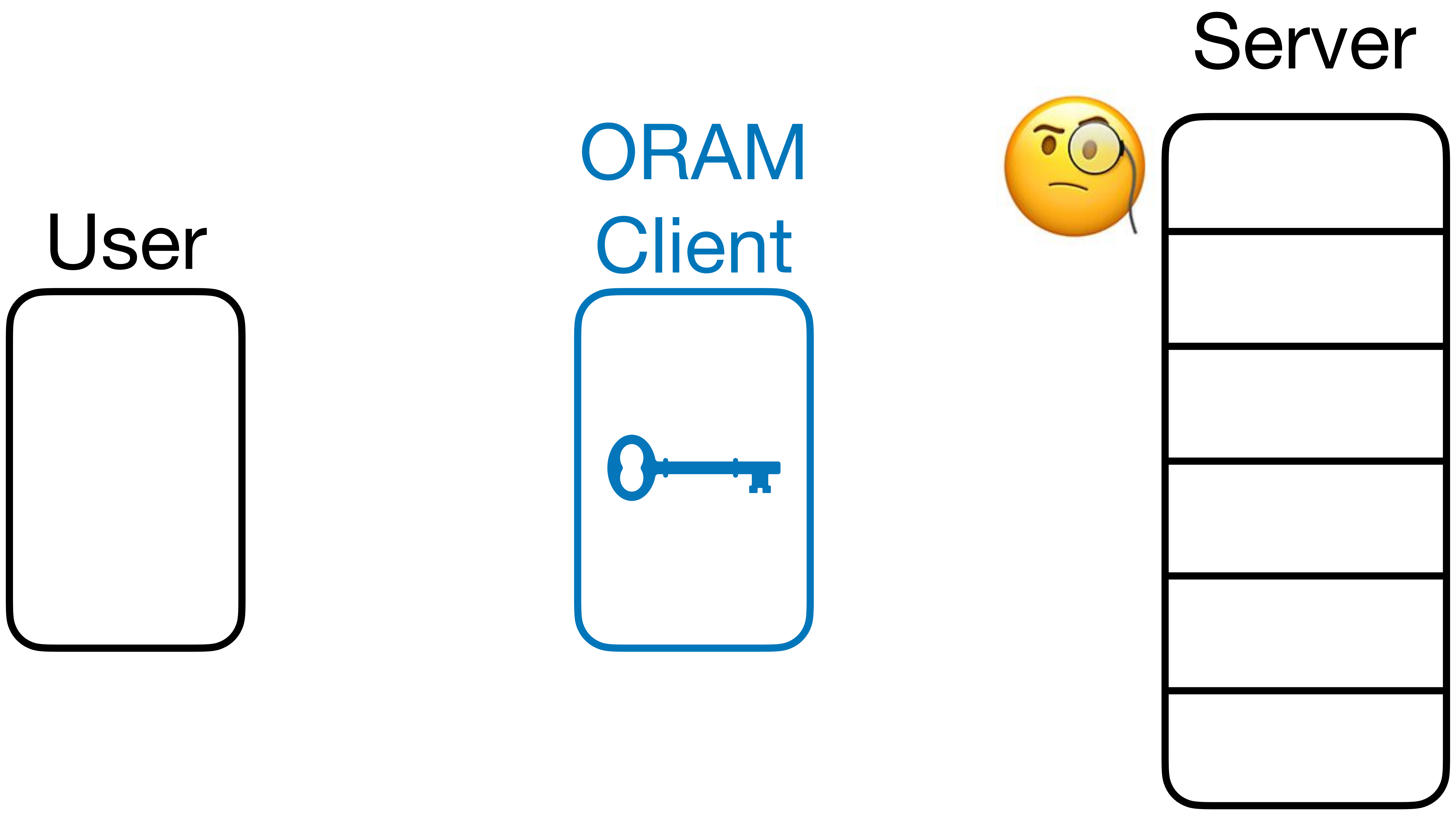
Oblivious RAM (ORAM)

[Goldreich ’87, Ostrovsky ’90, Goldreich-Ostrovsky ‘96]



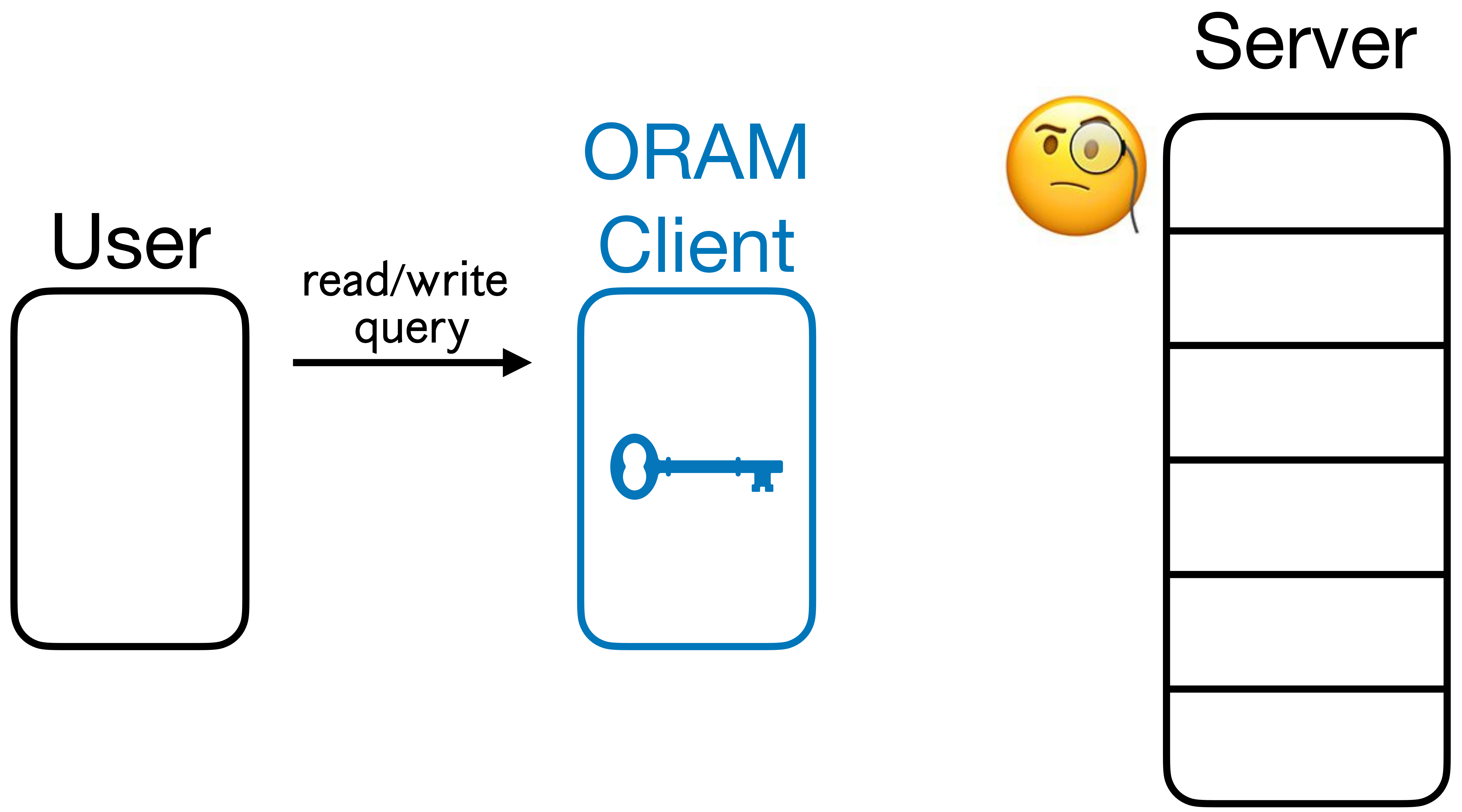
Oblivious RAM (ORAM)

[Goldreich ’87, Ostrovsky ’90, Goldreich-Ostrovsky ‘96]



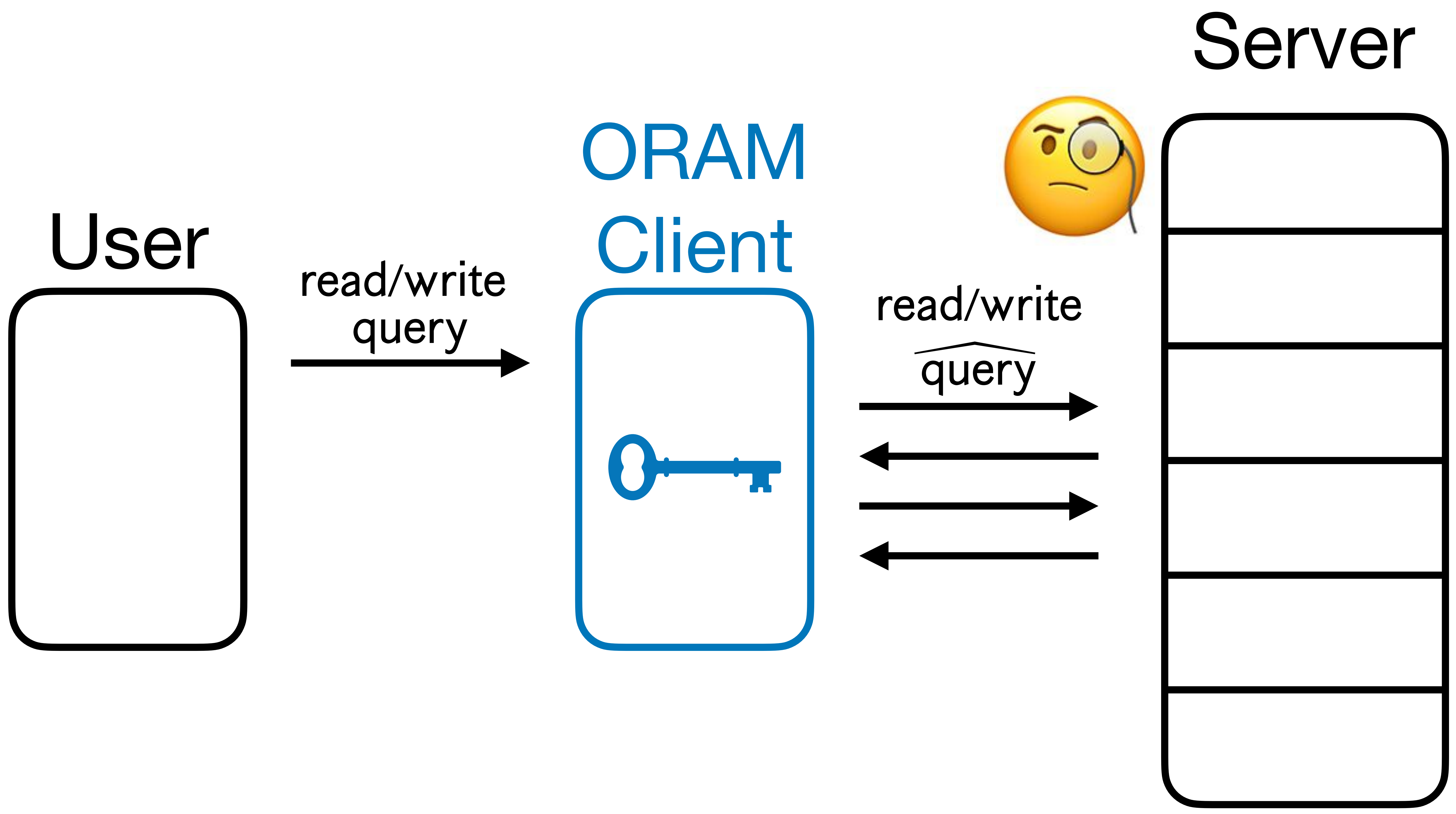
Oblivious RAM (ORAM)

[Goldreich ’87, Ostrovsky ’90, Goldreich-Ostrovsky ‘96]



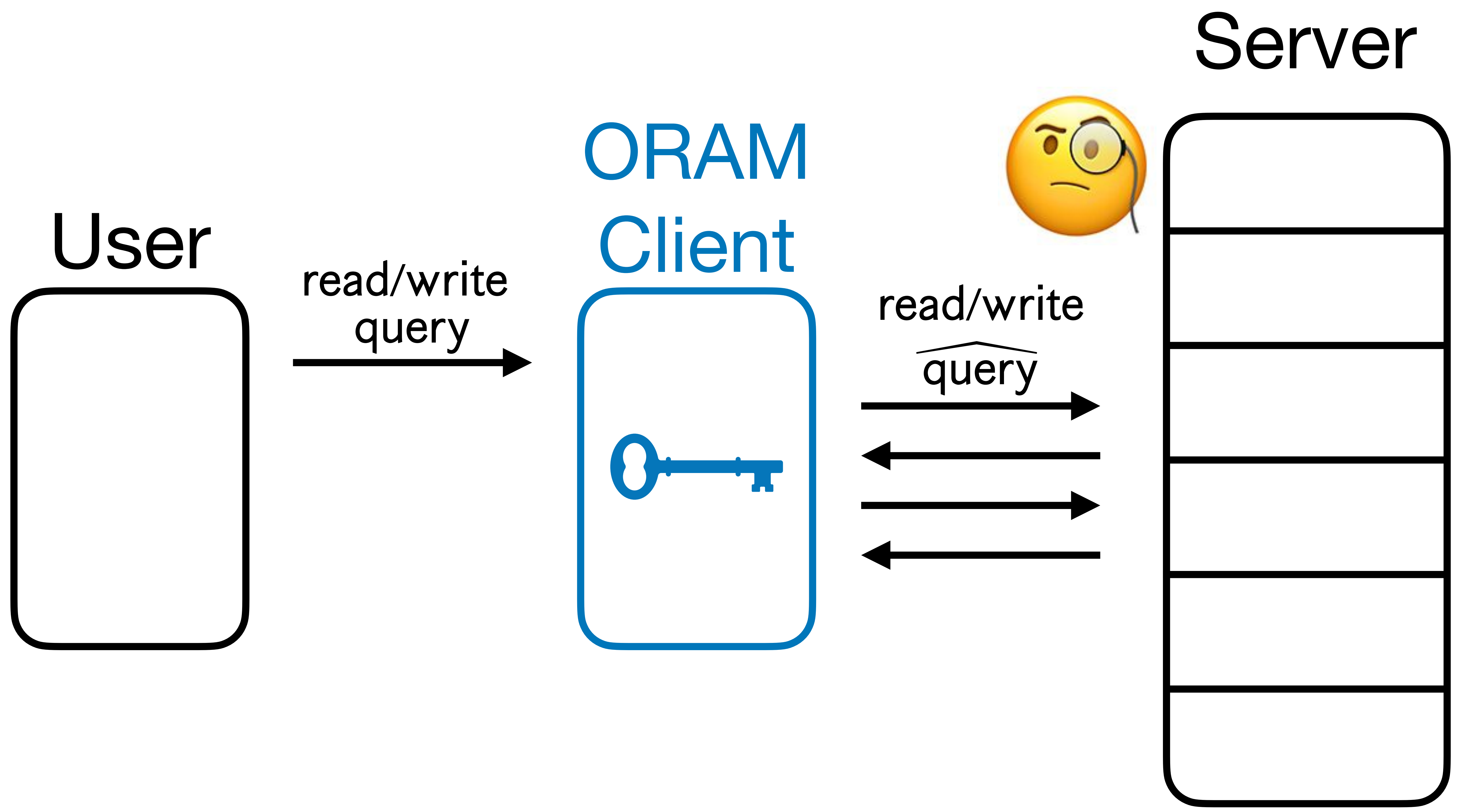
Oblivious RAM (ORAM)

[Goldreich ’87, Ostrovsky ’90, Goldreich-Ostrovsky ‘96]



Oblivious RAM (ORAM)

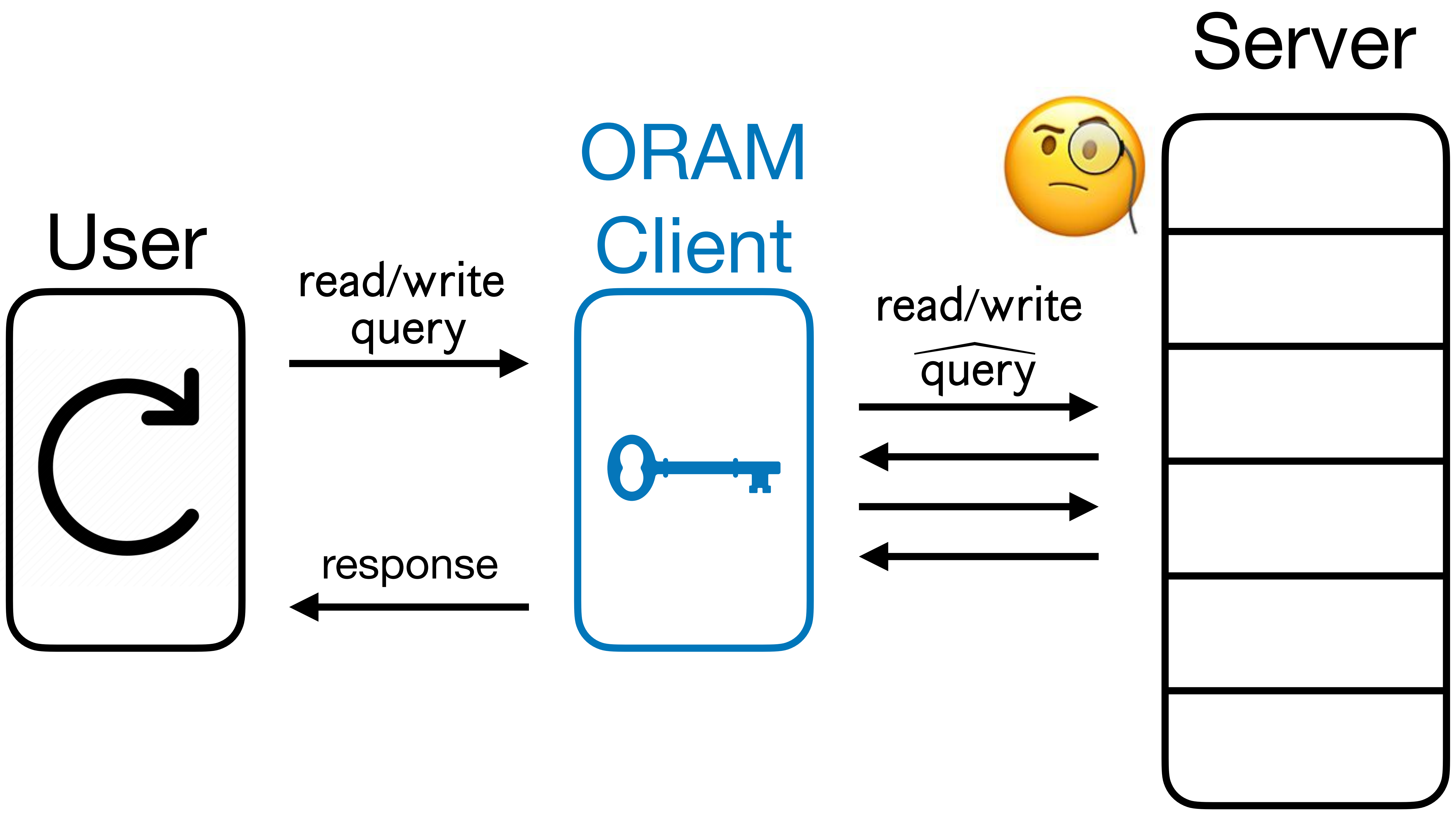
[Goldreich ’87, Ostrovsky ’90, Goldreich-Ostrovsky ‘96]



Server is a *passive storage* which does no **additional** work.

Oblivious RAM (ORAM)

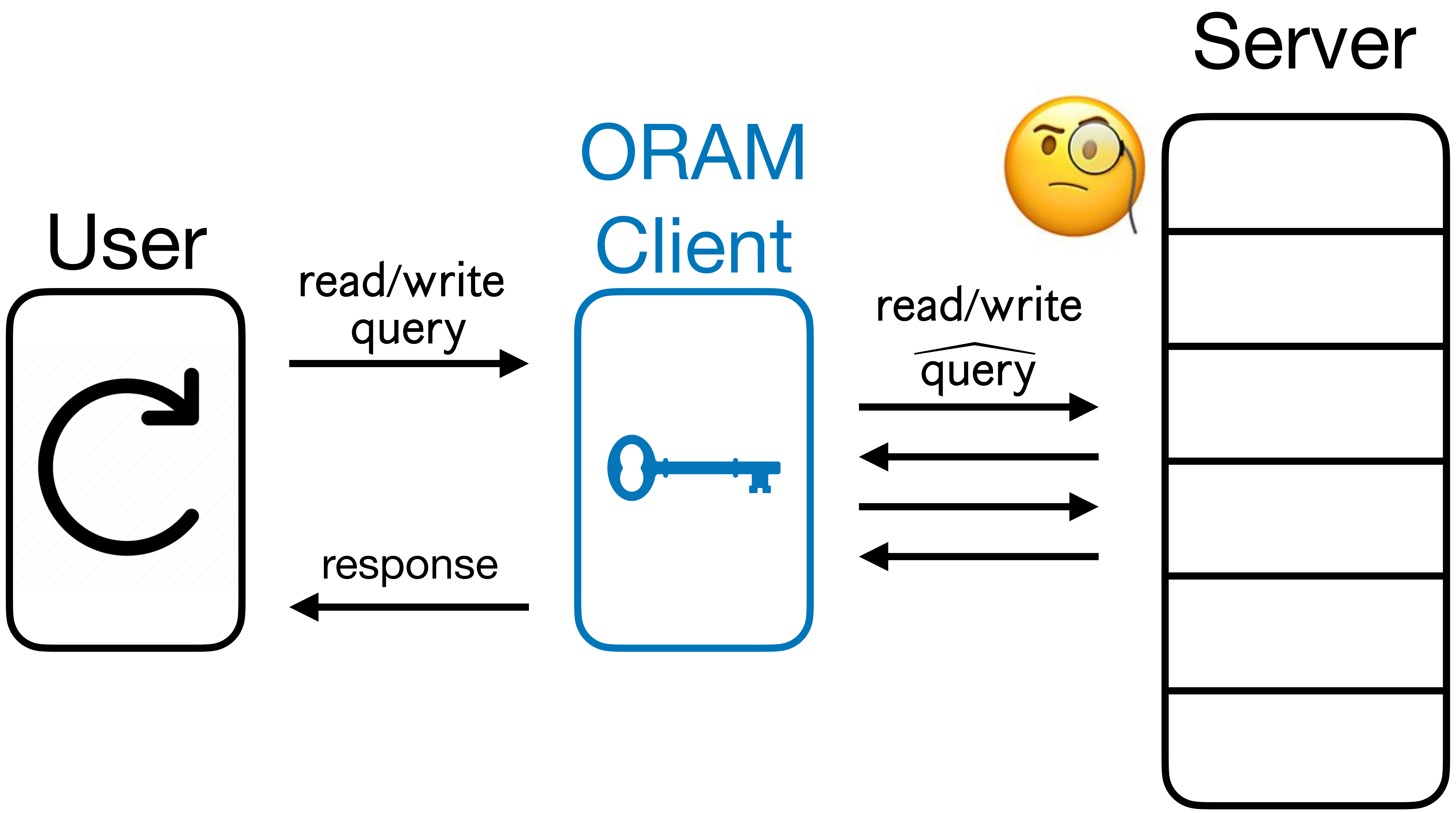
[Goldreich ’87, Ostrovsky ’90, Goldreich-Ostrovsky ‘96]



Server is a *passive storage* which does no **additional** work.

Oblivious RAM (ORAM)

[Goldreich ’87, Ostrovsky ’90, Goldreich-Ostrovsky ‘96]

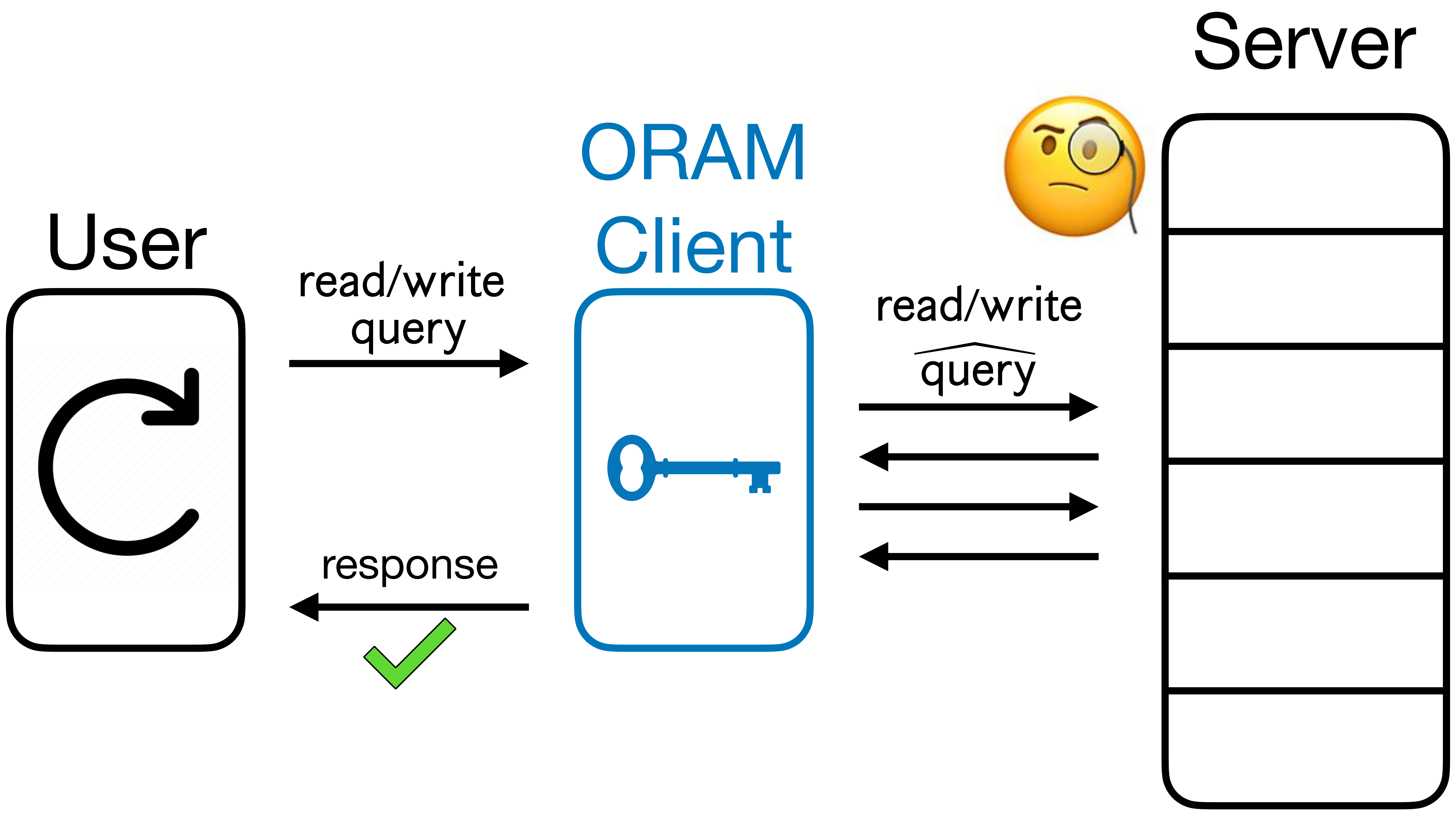


Server is a *passive storage* which does no **additional** work.

- **Correctness:** For any user queries, the ORAM responses to the user are correct.

Oblivious RAM (ORAM)

[Goldreich '87, Ostrovsky '90, Goldreich-Ostrovsky '96]

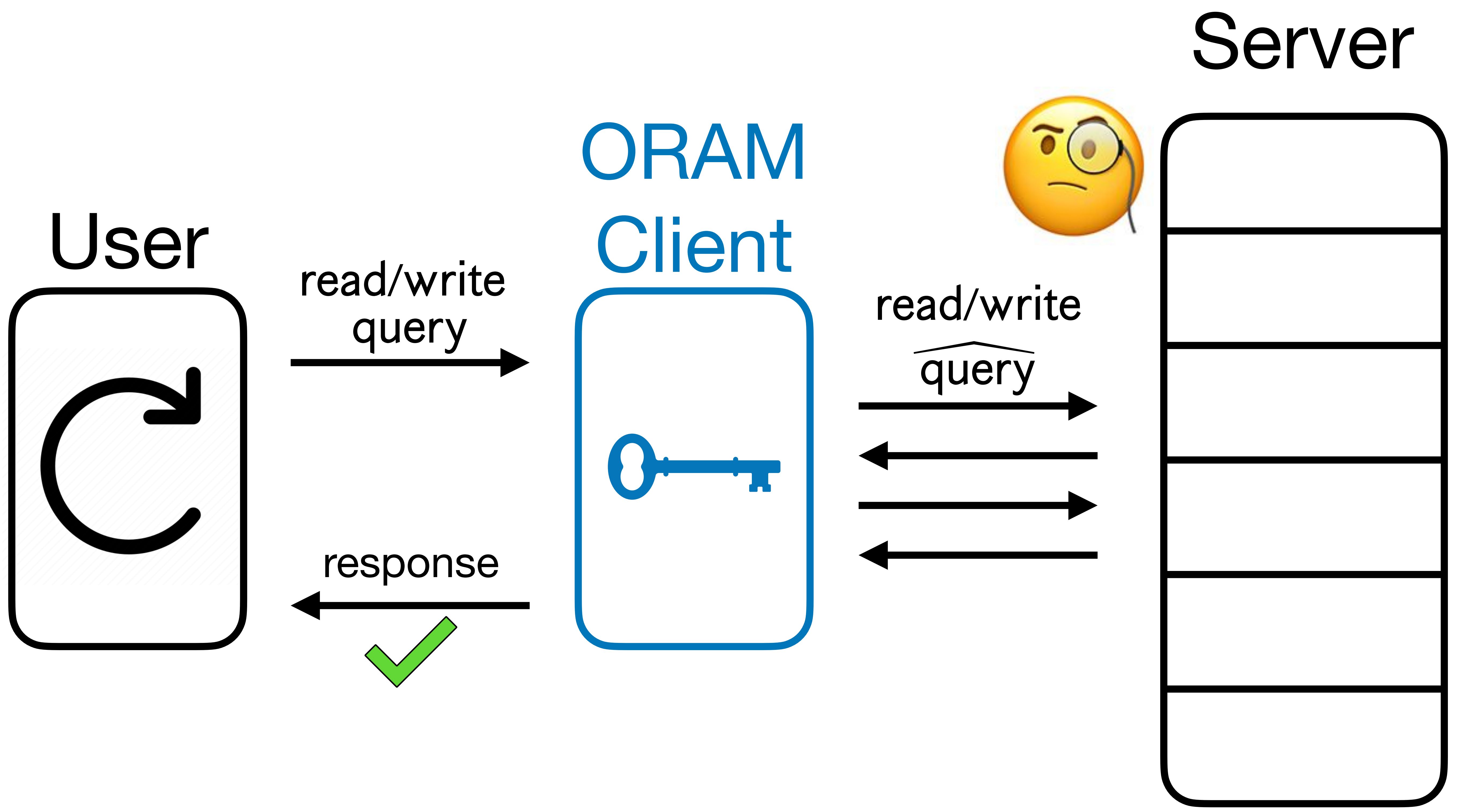


Server is a *passive storage* which does no **additional** work.

- **Correctness:** For any user queries, the ORAM responses to the user are correct.

Oblivious RAM (ORAM)

[Goldreich ’87, Ostrovsky ’90, Goldreich-Ostrovsky ‘96]

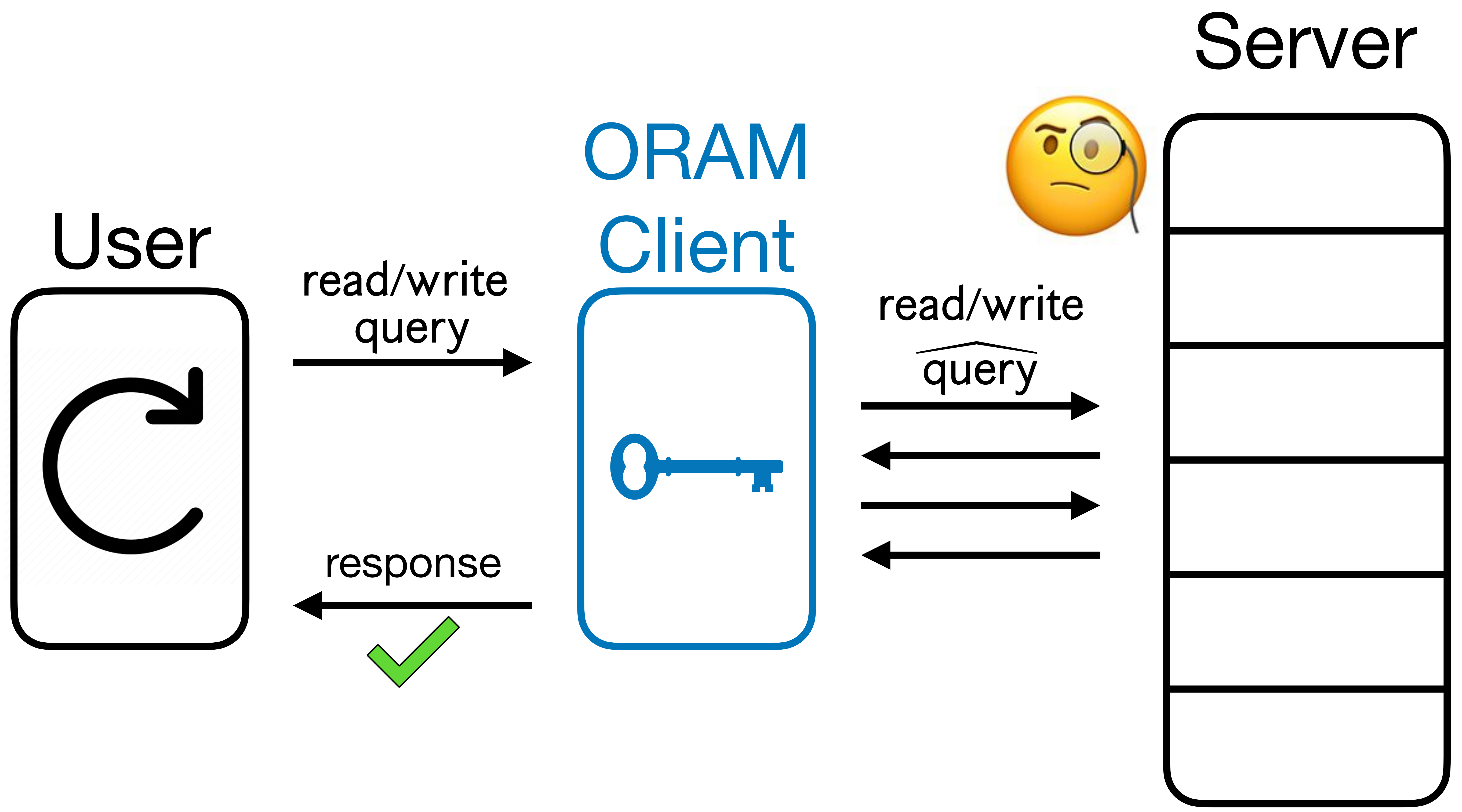


Server is a *passive storage* which does no **additional** work.

- **Correctness**: For any user queries, the ORAM responses to the user are correct.
- **Obliviousness**: Compiled queries leak *nothing* about the user queries (except for the number of queries):

Oblivious RAM (ORAM)

[Goldreich '87, Ostrovsky '90, Goldreich-Ostrovsky '96]



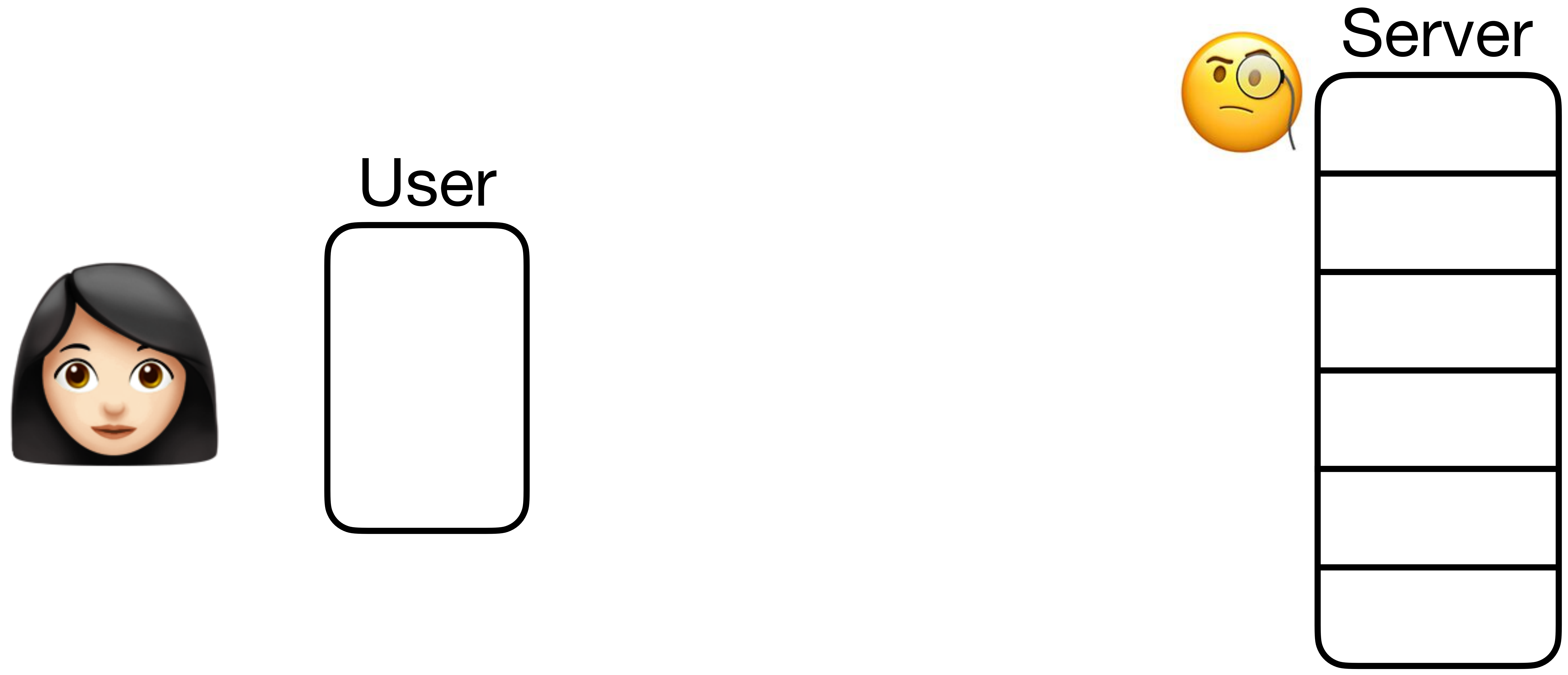
Server is a *passive storage* which does no **additional** work.

- **Correctness**: For any user queries, the ORAM responses to the user are correct.

- **Obliviousness**: Compiled queries leak *nothing* about the user queries (except for the number of queries):

$$\text{“}\left\{\widehat{\text{query}}\right\} \approx_{\text{comp}} \text{Sim}\left(1^{|\overrightarrow{\text{query}}|}\right)\text{”}$$

Application: File Storage Platforms

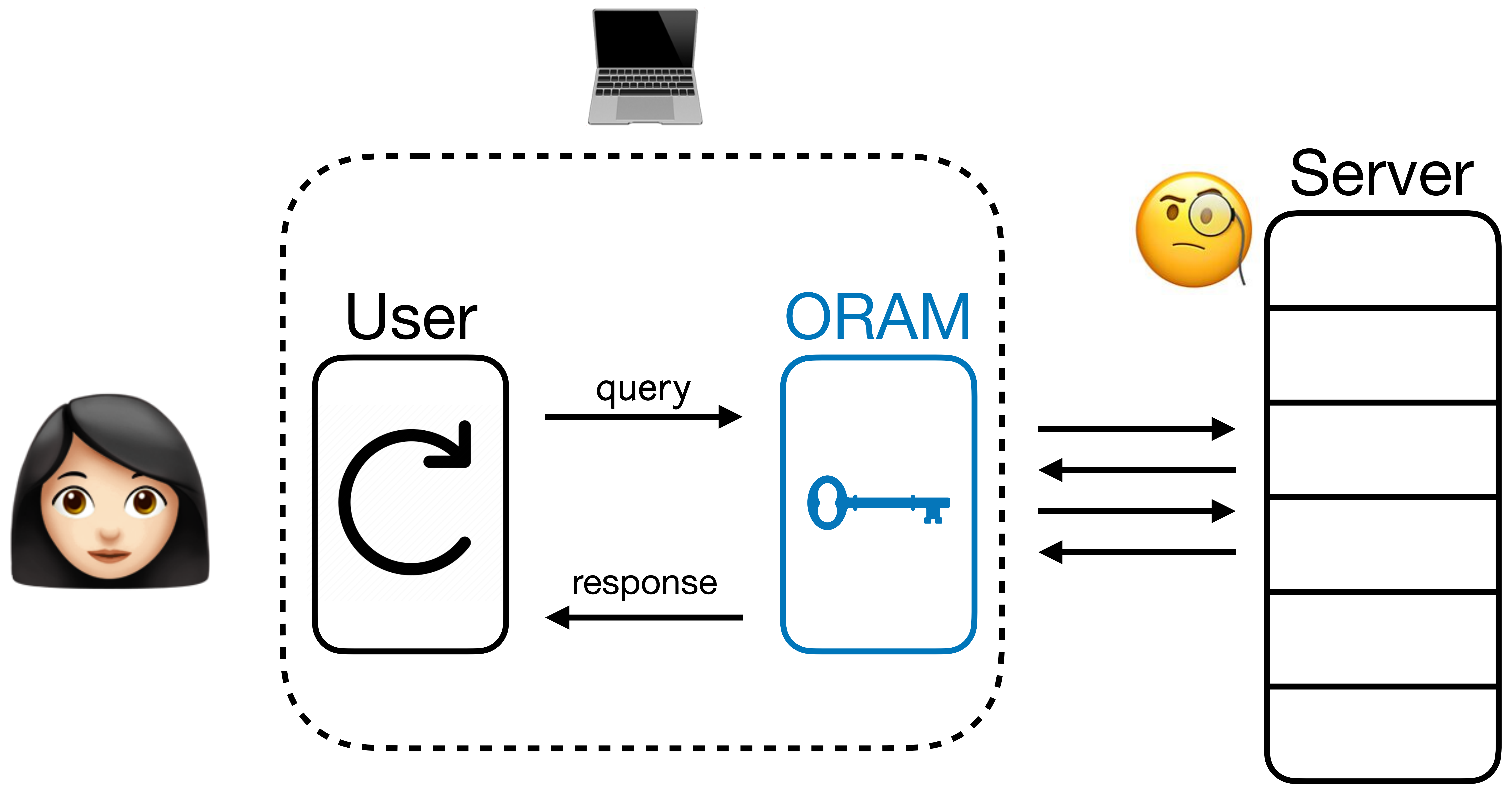


Dropbox

Google Drive

OneDrive

Application: File Storage Platforms

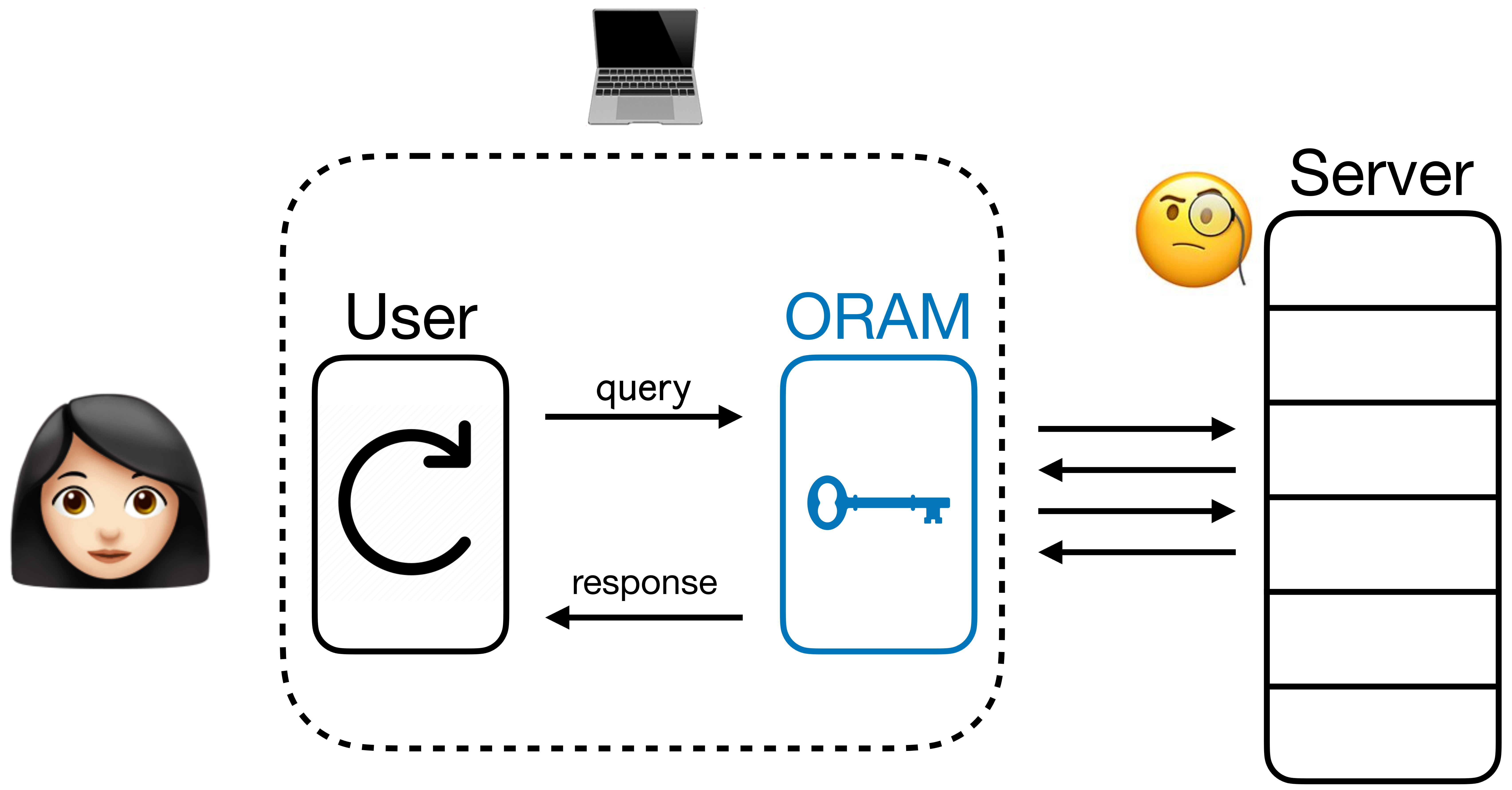


Dropbox

Google Drive

OneDrive

Application: File Storage Platforms



Dropbox

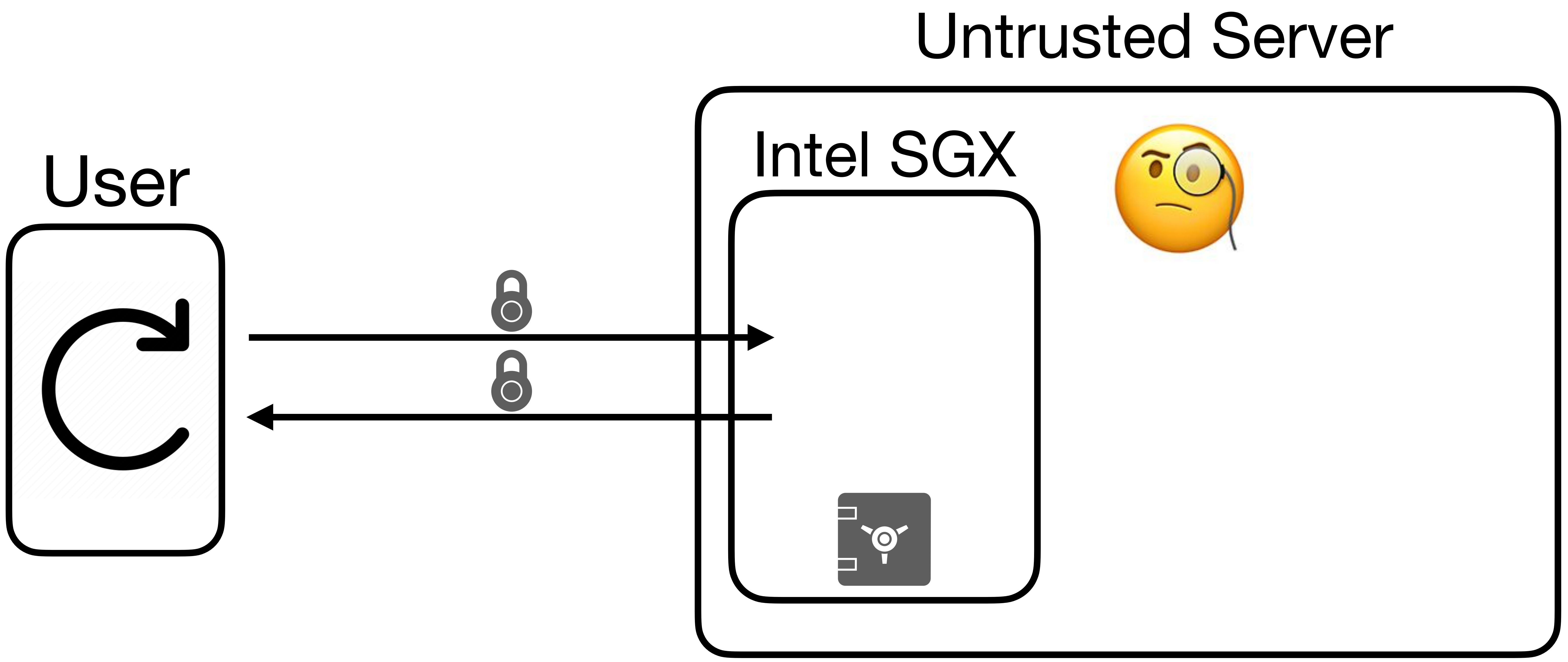
Google Drive

OneDrive

With ORAM, storage platform can't learn anything.

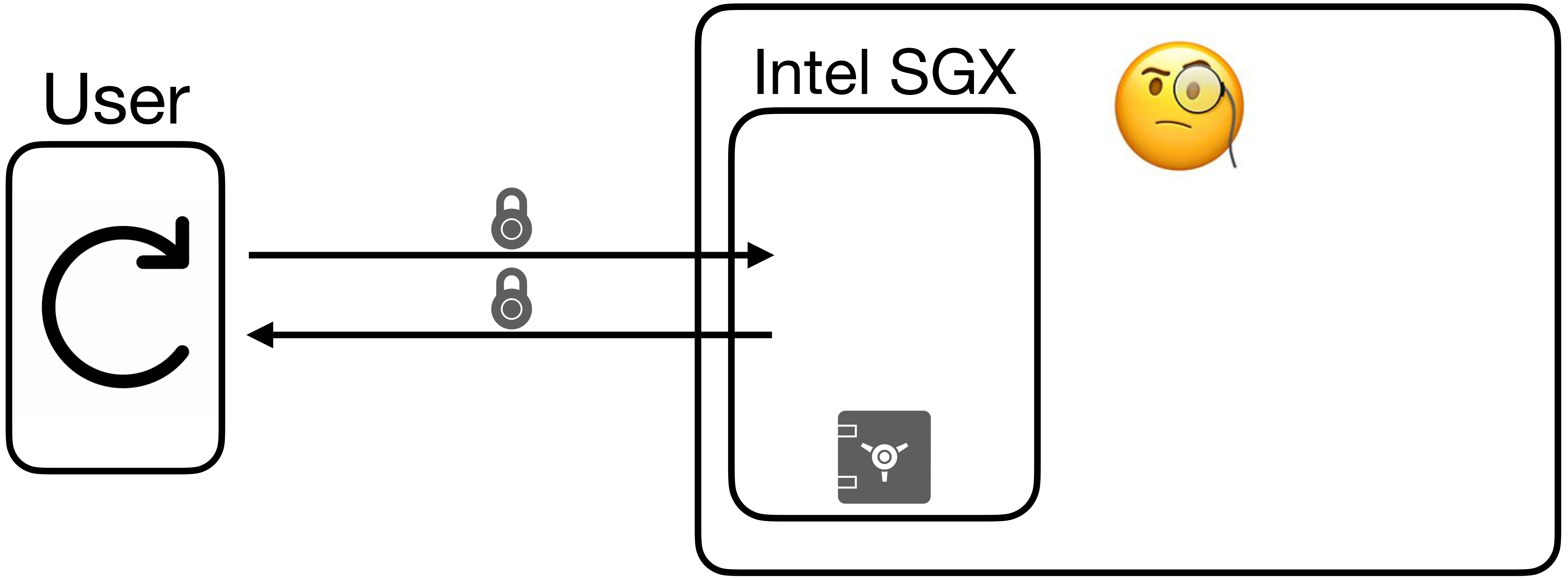
Application: Secure Hardware Enclaves

- **Secure Hardware Enclaves** (e.g., Intel SGX) allow users to execute programs securely on untrusted remote servers.



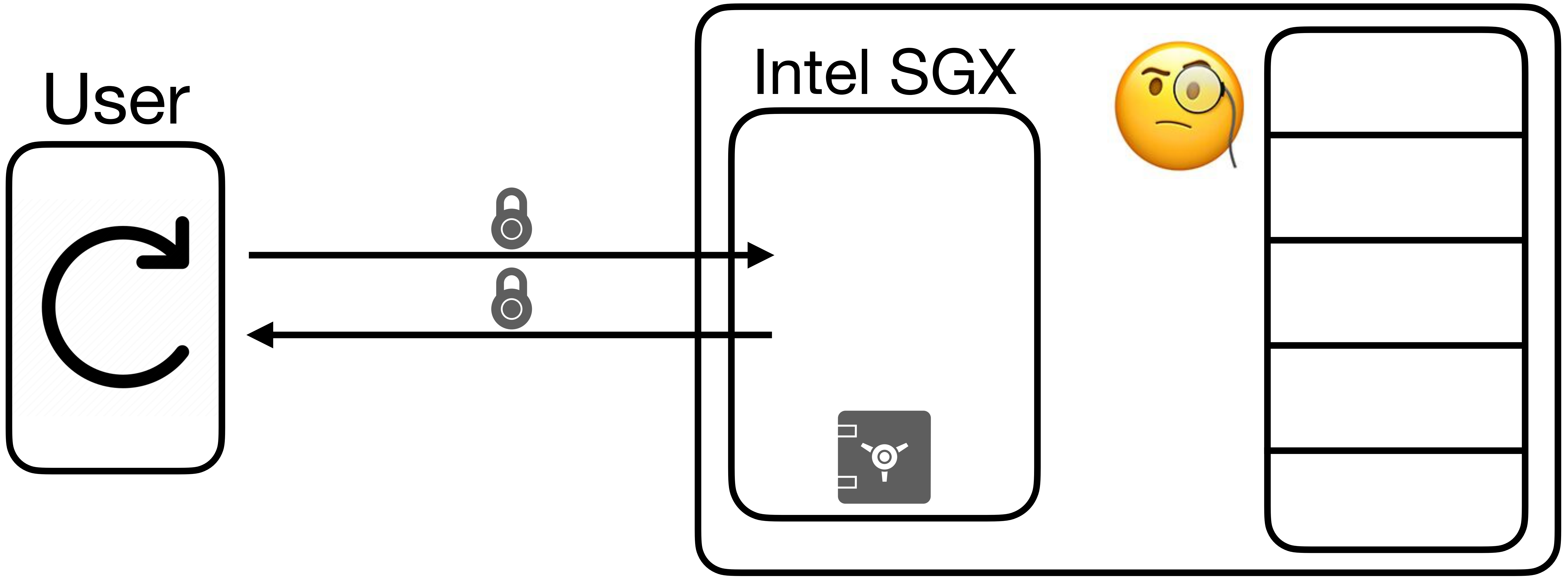
Application: Secure Hardware Enclaves

- **Secure Hardware Enclaves** (e.g., Intel SGX) allow users to execute programs securely on untrusted remote servers.
- Some enclaves have tiny internal space. Use untrusted memory within the server!



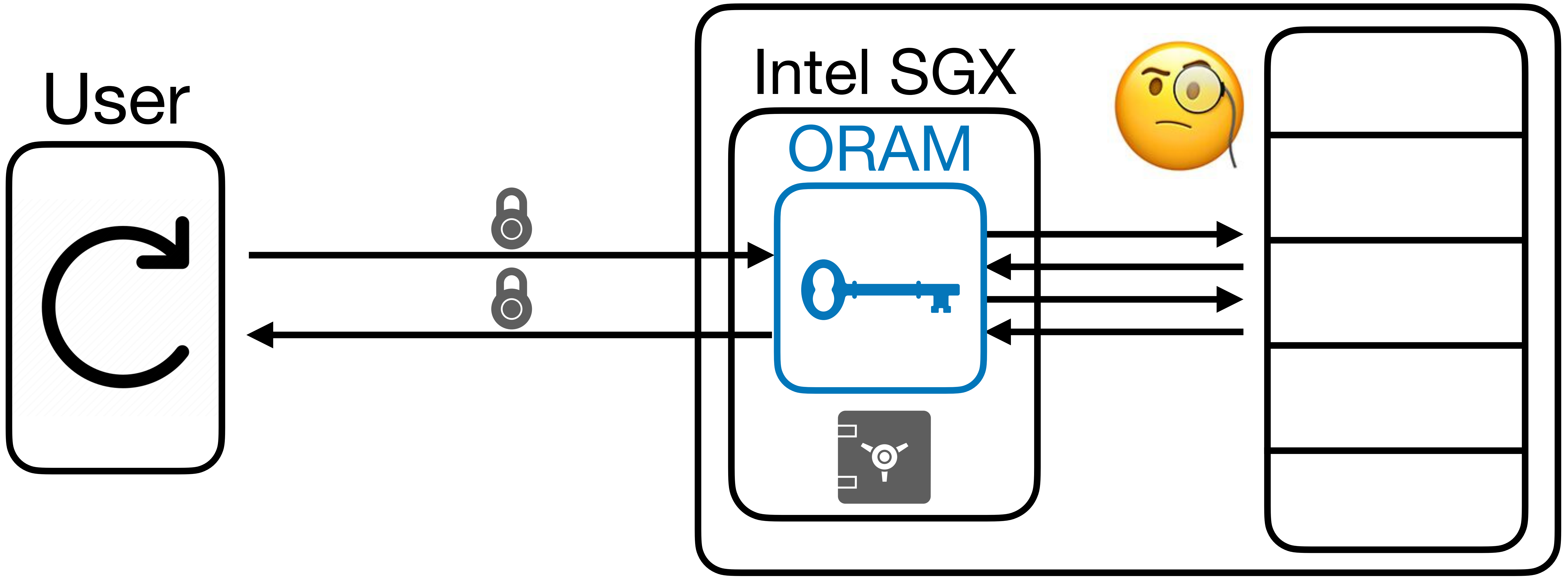
Application: Secure Hardware Enclaves

- **Secure Hardware Enclaves** (e.g., Intel SGX) allow users to execute programs securely on untrusted remote servers.
- Some enclaves have tiny internal space. Use untrusted memory within the server!



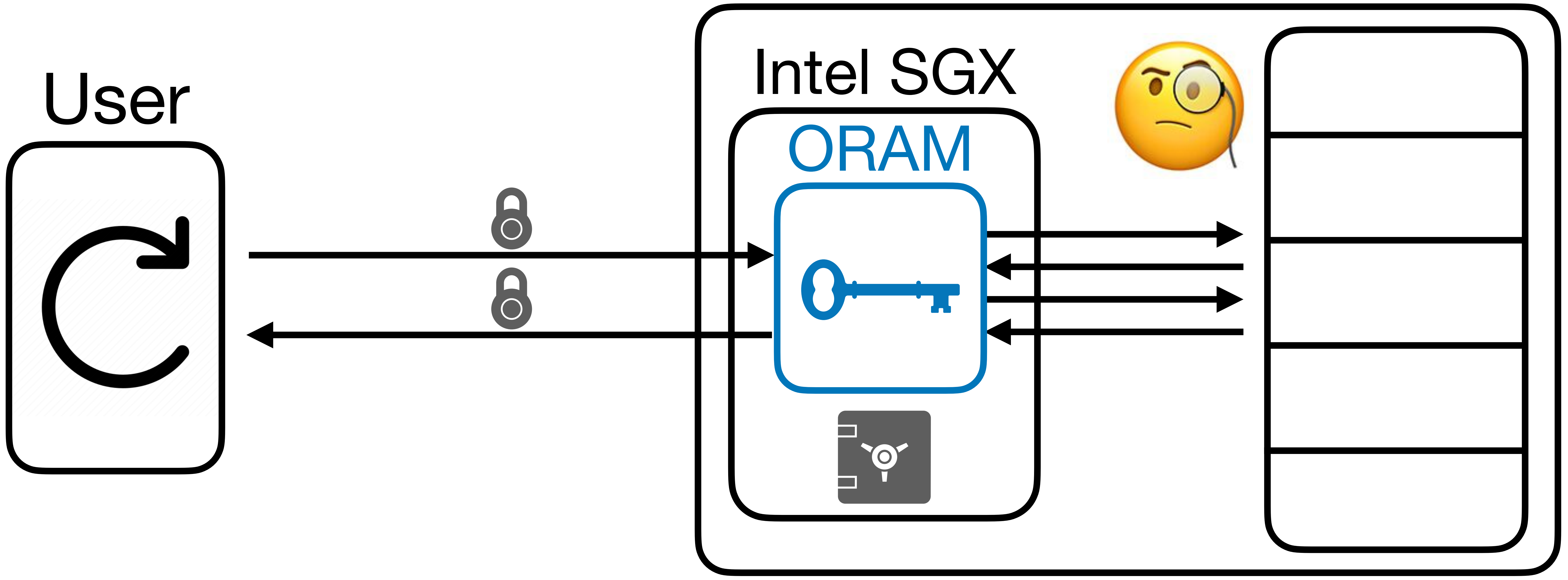
Application: Secure Hardware Enclaves

- **Secure Hardware Enclaves** (e.g., Intel SGX) allow users to execute programs securely on untrusted remote servers.
- Some enclaves have tiny internal space. Use untrusted memory within the server!



Application: Secure Hardware Enclaves

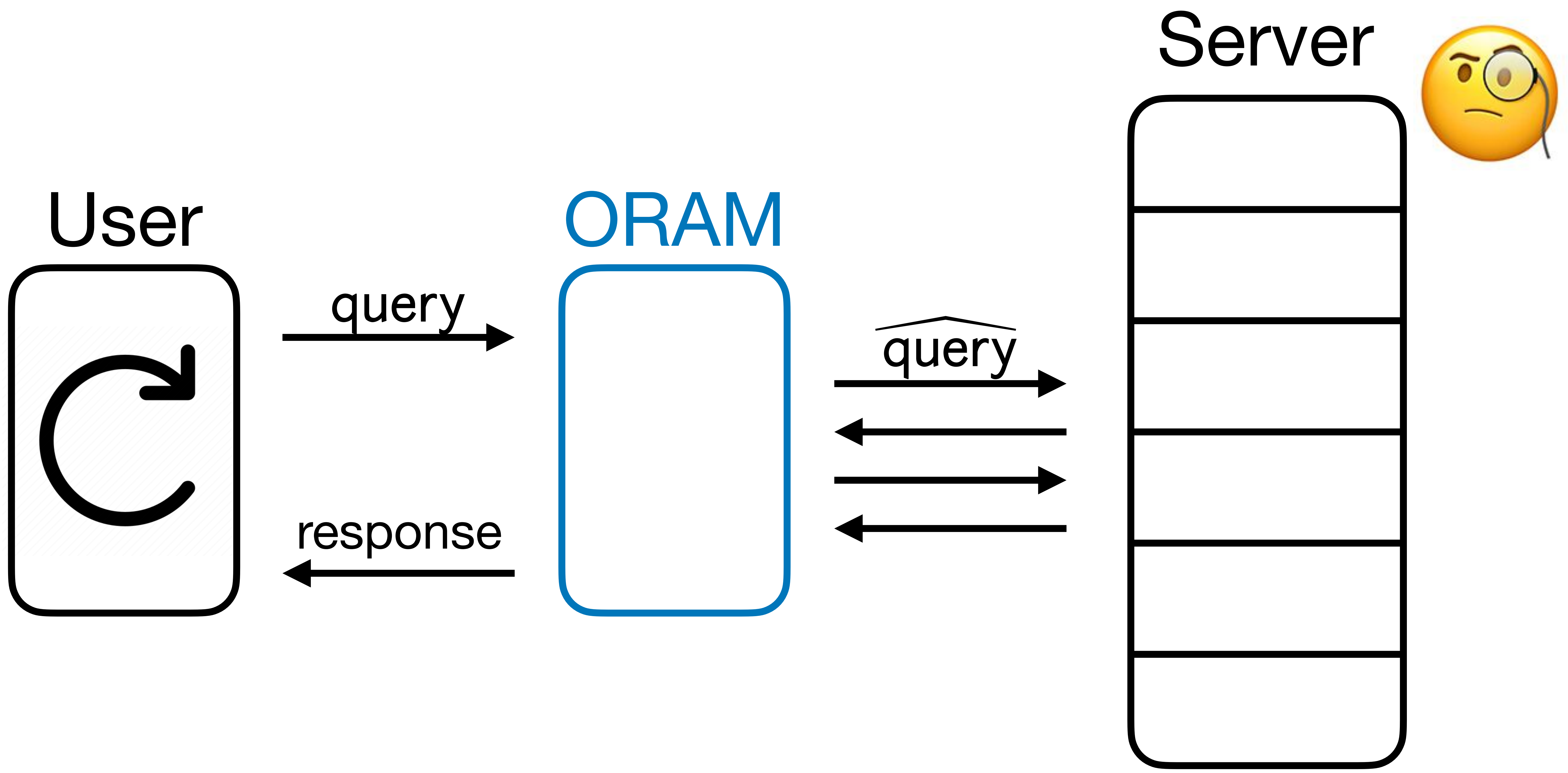
- **Secure Hardware Enclaves** (e.g., Intel SGX) allow users to execute programs securely on untrusted remote servers.
- Some enclaves have tiny internal space. Use untrusted memory within the server!



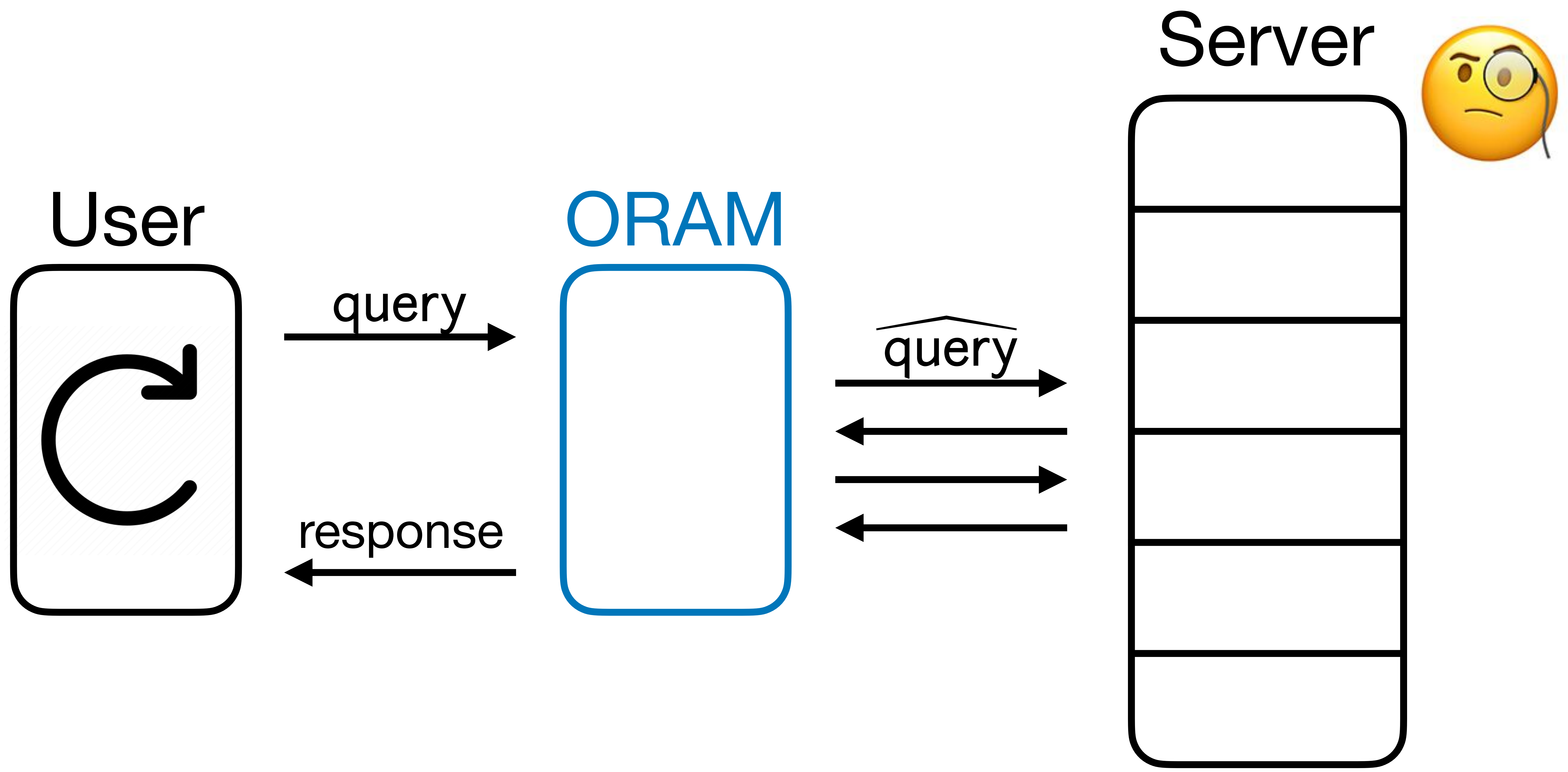
- **Real World**: Signal very recently implemented ORAM for private contact discovery!

Signal

ORAM Efficiency

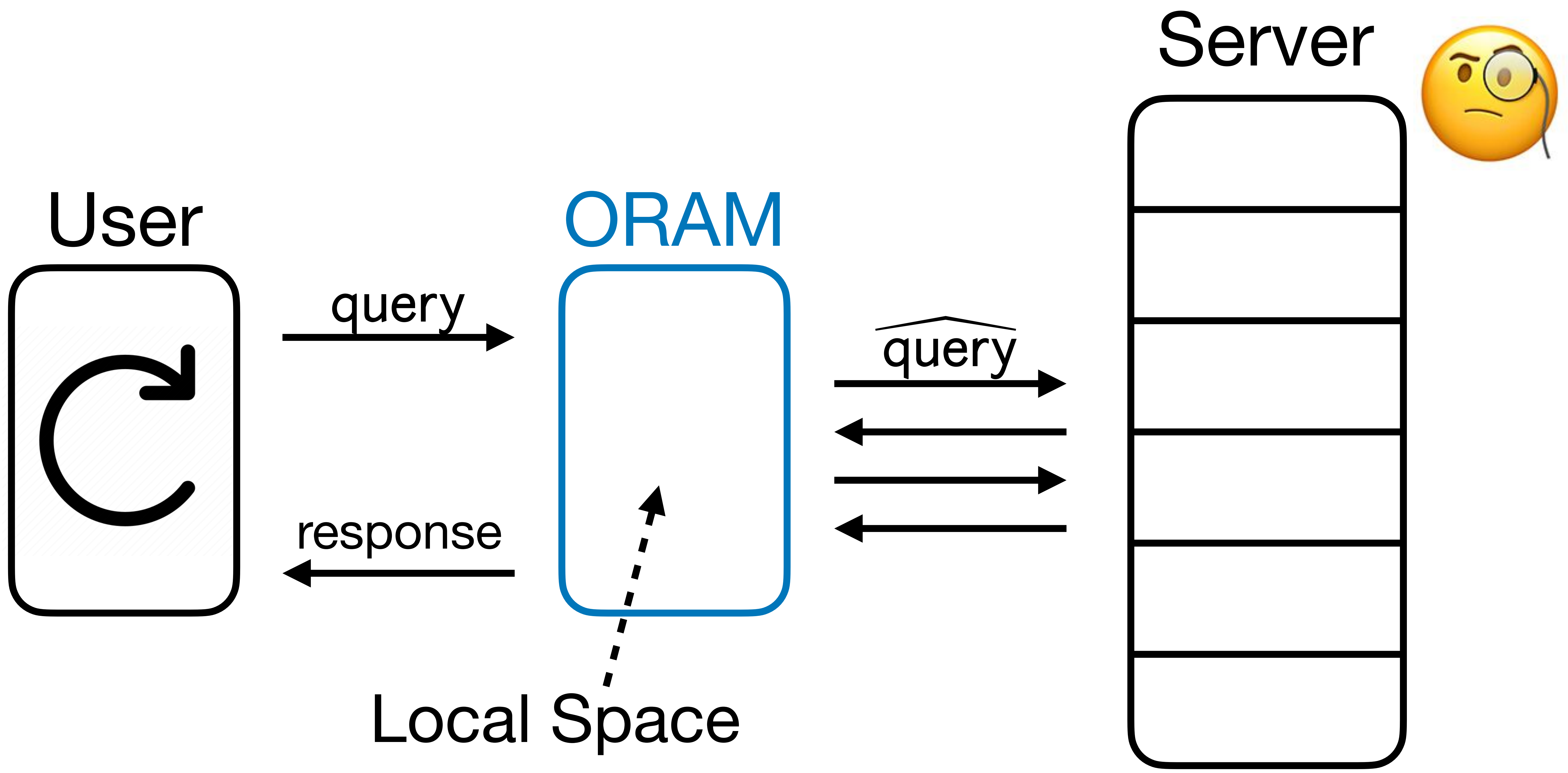


ORAM Efficiency



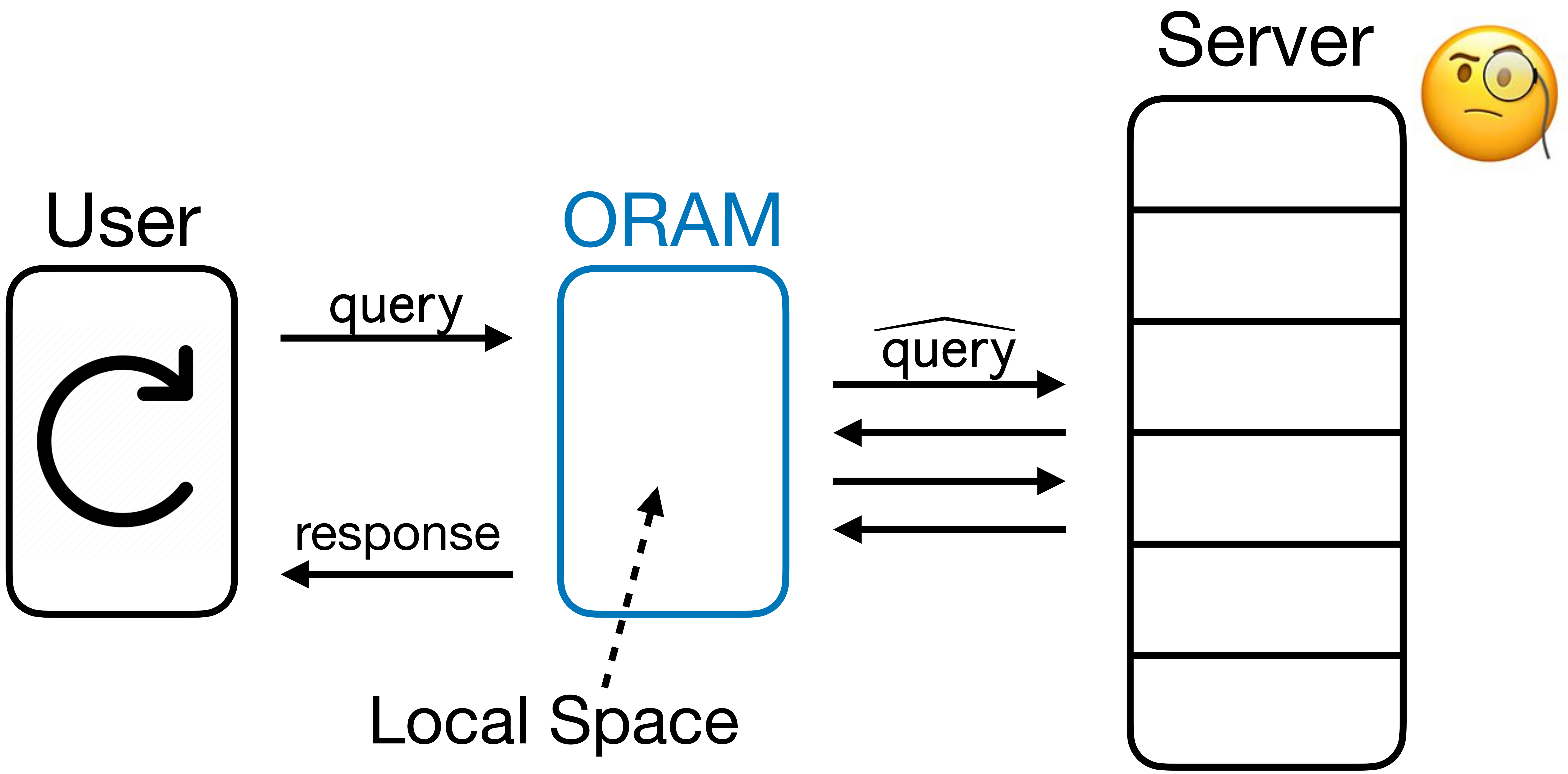
1. **Local Space**: Amount of space the ORAM can store locally (trusted & private). For talk, think of $O(1)$ words ($\approx \log N$ bits).

ORAM Efficiency



1. **Local Space**: Amount of space the ORAM can store locally (trusted & private). For talk, think of $O(1)$ words ($\approx \log N$ bits).

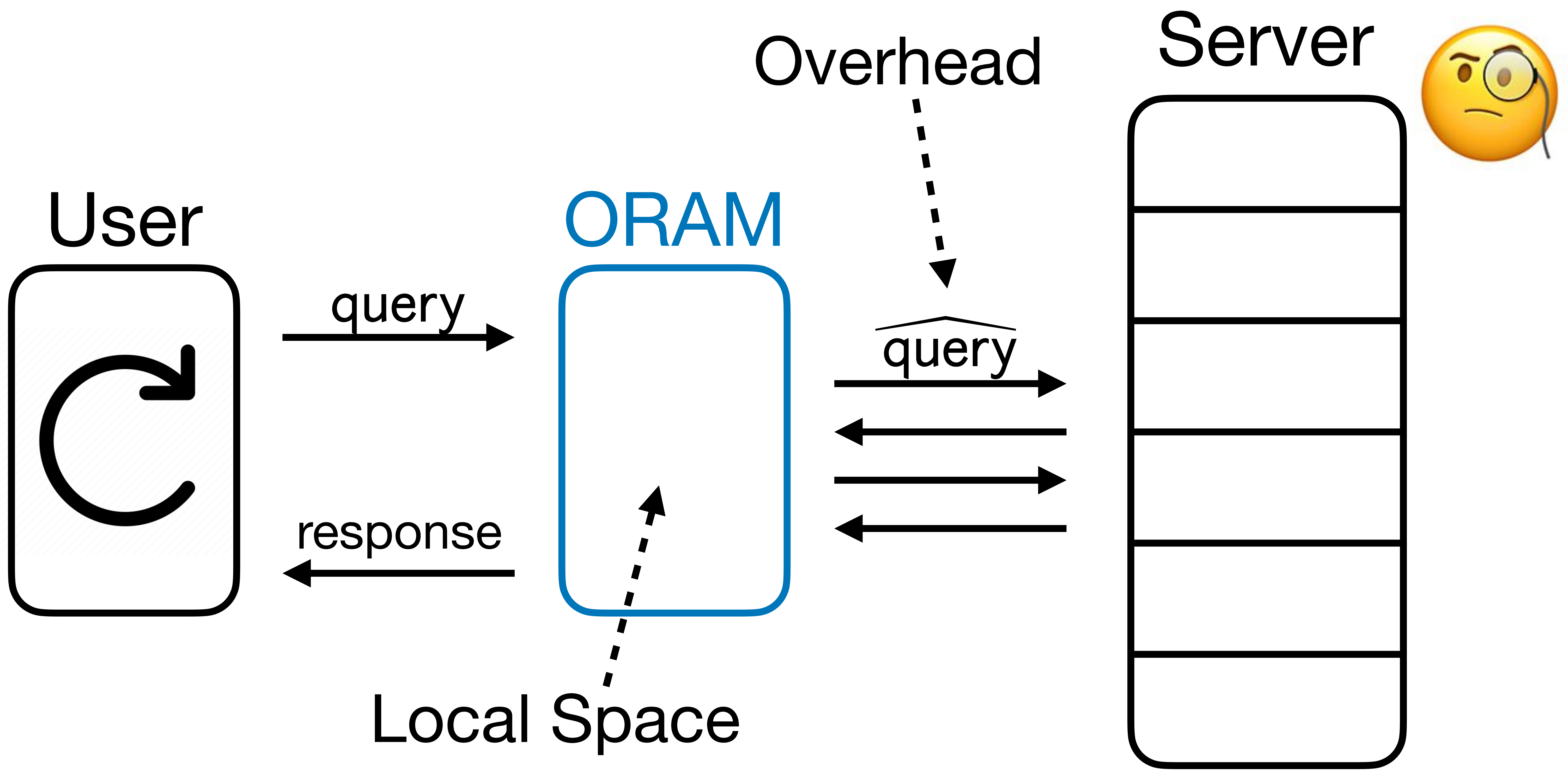
ORAM Efficiency



1. **Local Space**: Amount of space the ORAM can store locally (trusted & private). For talk, think of $O(1)$ words ($\approx \log N$ bits).

2. **Overhead**: Number of queries made to the server per user query.

ORAM Efficiency



1. **Local Space**: Amount of space the ORAM can store locally (trusted & private). For talk, think of $O(1)$ words ($\approx \log N$ bits).

2. **Overhead**: Number of queries made to the server per user query.

ORAM History

RAM of size N, word size $\Theta(\log N)$, local space size $O(1)$

Work	**Overhead**

ORAM History

RAM of size N, word size $\Theta(\log N)$, local space size $O(1)$

Work	**Overhead**
[Goldreich '87]	$\sqrt{N} \log N$

ORAM History

RAM of size N, word size $\Theta(\log N)$, local space size $O(1)$

Work	**Overhead**
[Goldreich ’87]	$\sqrt{N} \log N$
[Ostrovsky ’90, Goldreich-Ostrovsky ’96]	$\log^3 N$
Path ORAM [SvDSCFRYD ’12]	$\log^2 N$
PanORAMa [Patel-Persiano-Raykova-Yeo ’18]	$\log N \log \log N$

ORAM History

RAM of size N, word size $\Theta(\log N)$, local space size $O(1)$

Work	**Overhead**
[Goldreich ’87]	$\sqrt{N} \log N$
[Ostrovsky ’90, Goldreich-Ostrovsky ’96]	$\log^3 N$
Path ORAM [SvDSCFRYD ’12]	$\log^2 N$
PanORAMa [Patel-Persiano-Raykova-Yeo ’18]	$\log N \log \log N$
OptORAMa [AKLNPS ‘20, AKLS ’21]	$\log N$

ORAM History

RAM of size N, word size $\Theta(\log N)$, local space size $O(1)$

Work	**Overhead**
[Goldreich ’87]	$\sqrt{N}\log N$
[Ostrovsky ’90, Goldreich-Ostrovsky ’96]	$\log^3 N$
Path ORAM [SvDSCFRYD ’12]	$\log^2 N$
PanORAMa [Patel-Persiano-Raykova-Yeo ’18]	$\log N \log\log N$
OptORAMa [AKLNPS ‘20, AKLS ’21]	$\log N$

Lower Bound: [Goldreich ’87, Larsen-Nielsen ’18, Komargodski-Lin ’21]	$\Omega\left(\log N\right)$

ORAM History

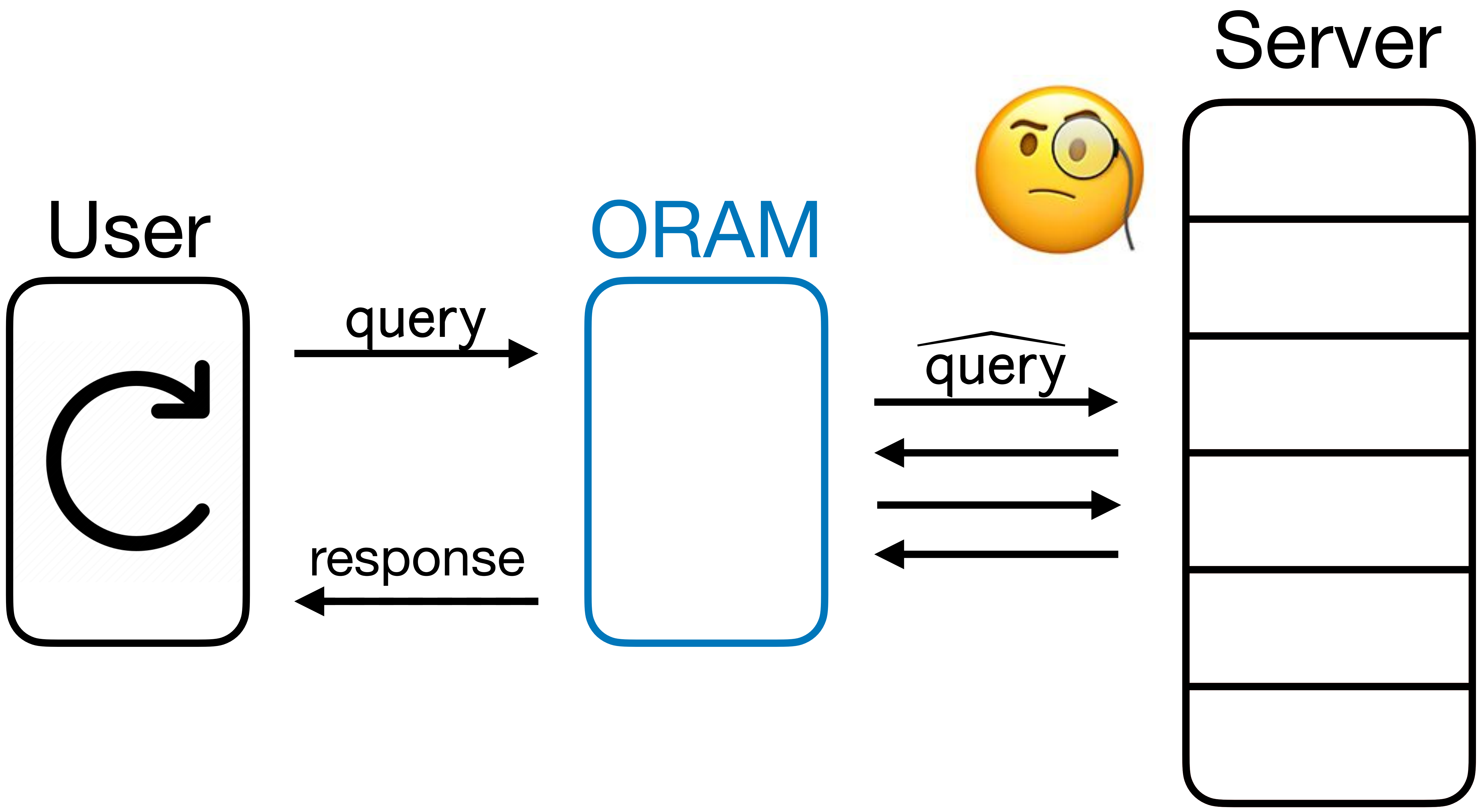
RAM of size N, word size $\Theta(\log N)$, local space size $O(1)$

Work	**Overhead**
[Goldreich '87]	$\sqrt{N}\log N$
[Ostrovsky '90, Goldreich-Ostrovsky '96]	$\log^3 N$
Path ORAM [SvDSCFRYD '12]	$\log^2 N$
PanORAMa [Patel-Persiano-Raykova-Yeo '18]	$\log N \log\log N$
OptORAMa [AKLNPS '20, AKLS '21]	$\log N$ ← Optimal!

Lower Bound: [Goldreich '87, Larsen-Nielsen '18, Komargodski-Lin '21]	$\Omega\left(\log N\right)$

Power of the Adversary

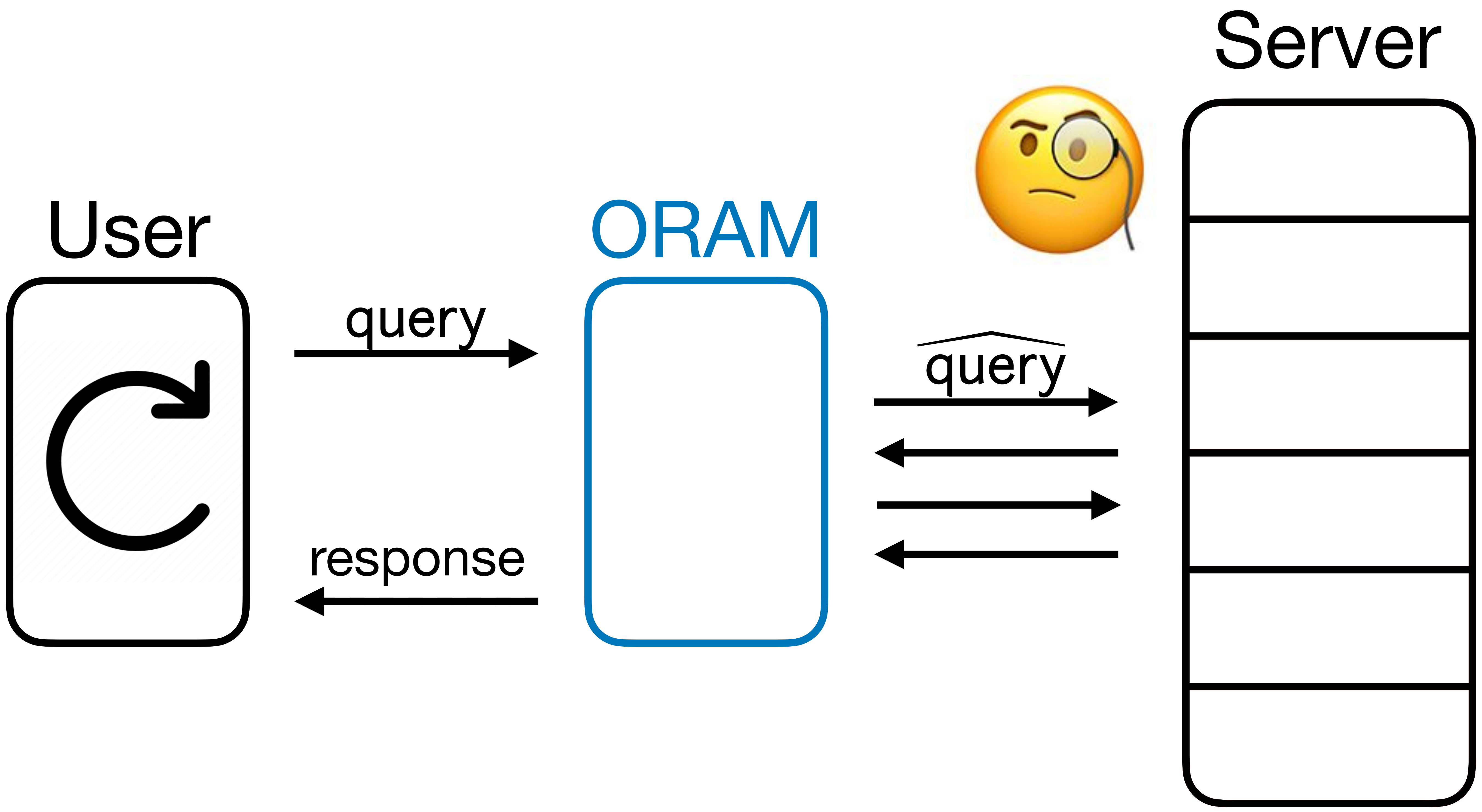
- But up until now, we have assumed a **passive**, **honest-but-curious** RAM server that can try to learn something about the queries.



Power of the Adversary

- But up until now, we have assumed a **passive**, **honest-but-curious** RAM server that can try to learn something about the queries.

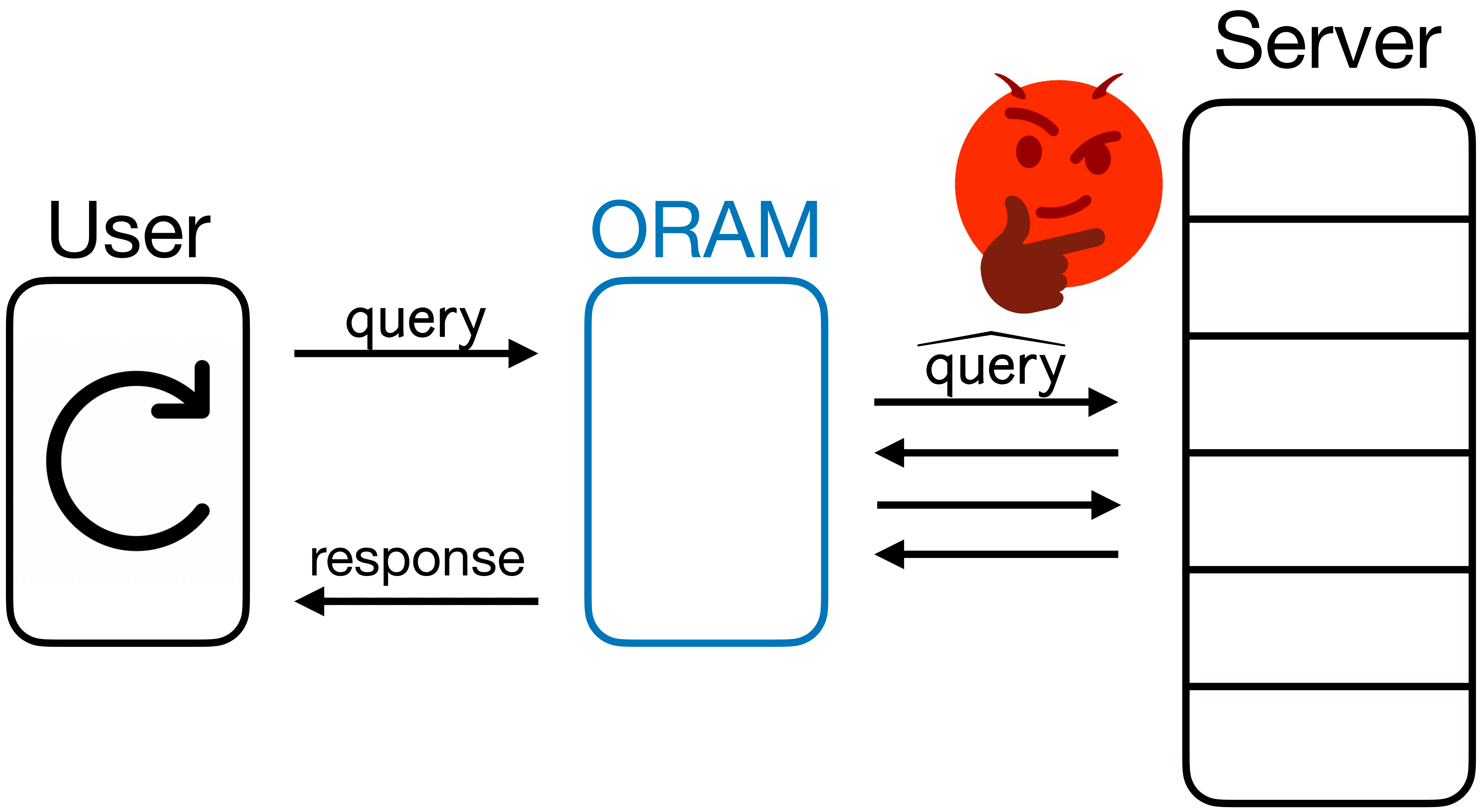
- In reality, ***an adversary can do more***! What about an **active, malicious** adversary that can **modify** the contents in the RAM?



Power of the Adversary

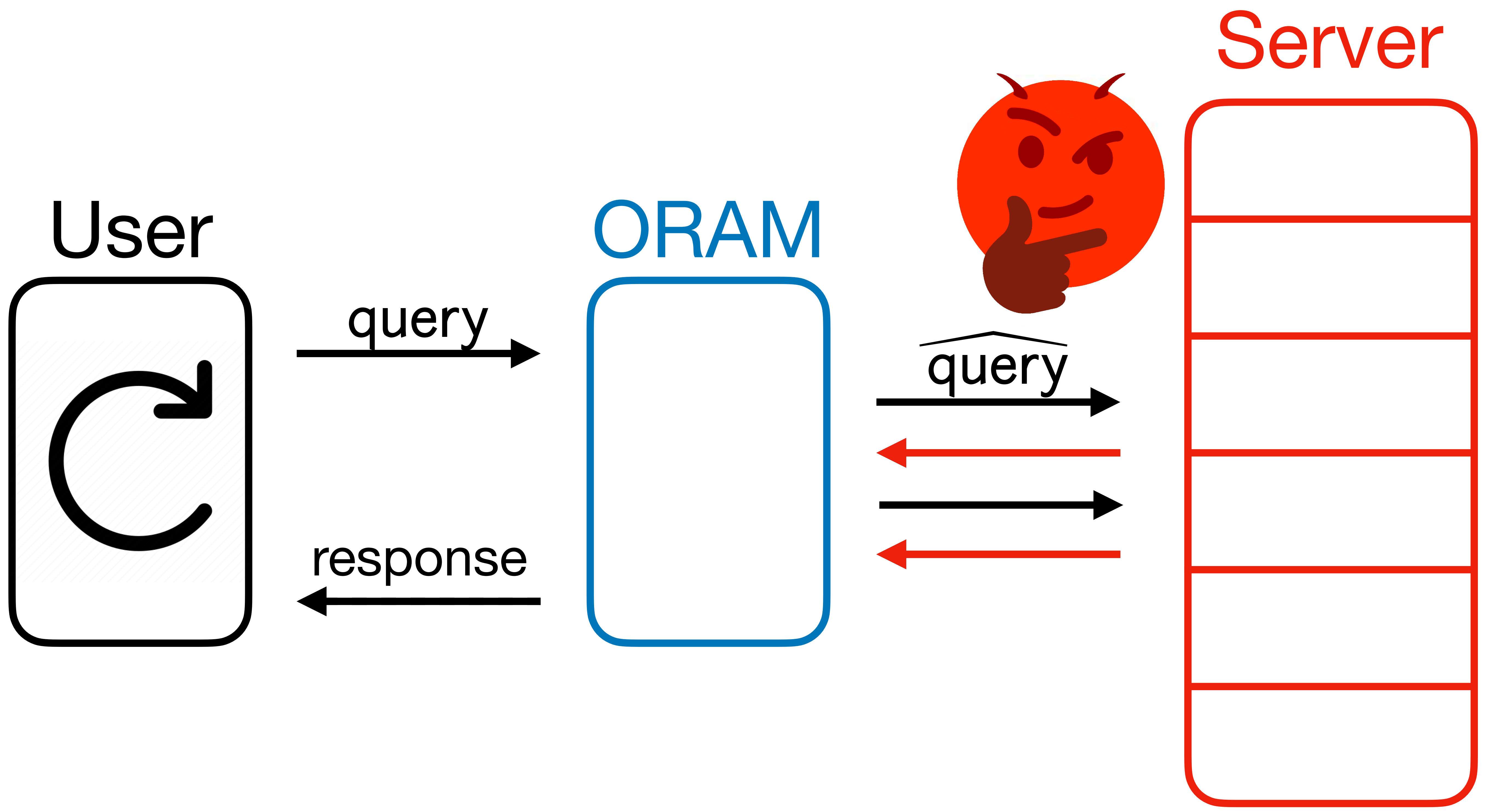
- But up until now, we have assumed a **passive**, **honest-but-curious** RAM server that can try to learn something about the queries.

- In reality, ***an adversary can do more***! What about an **active, malicious** adversary that can **modify** the contents in the RAM?



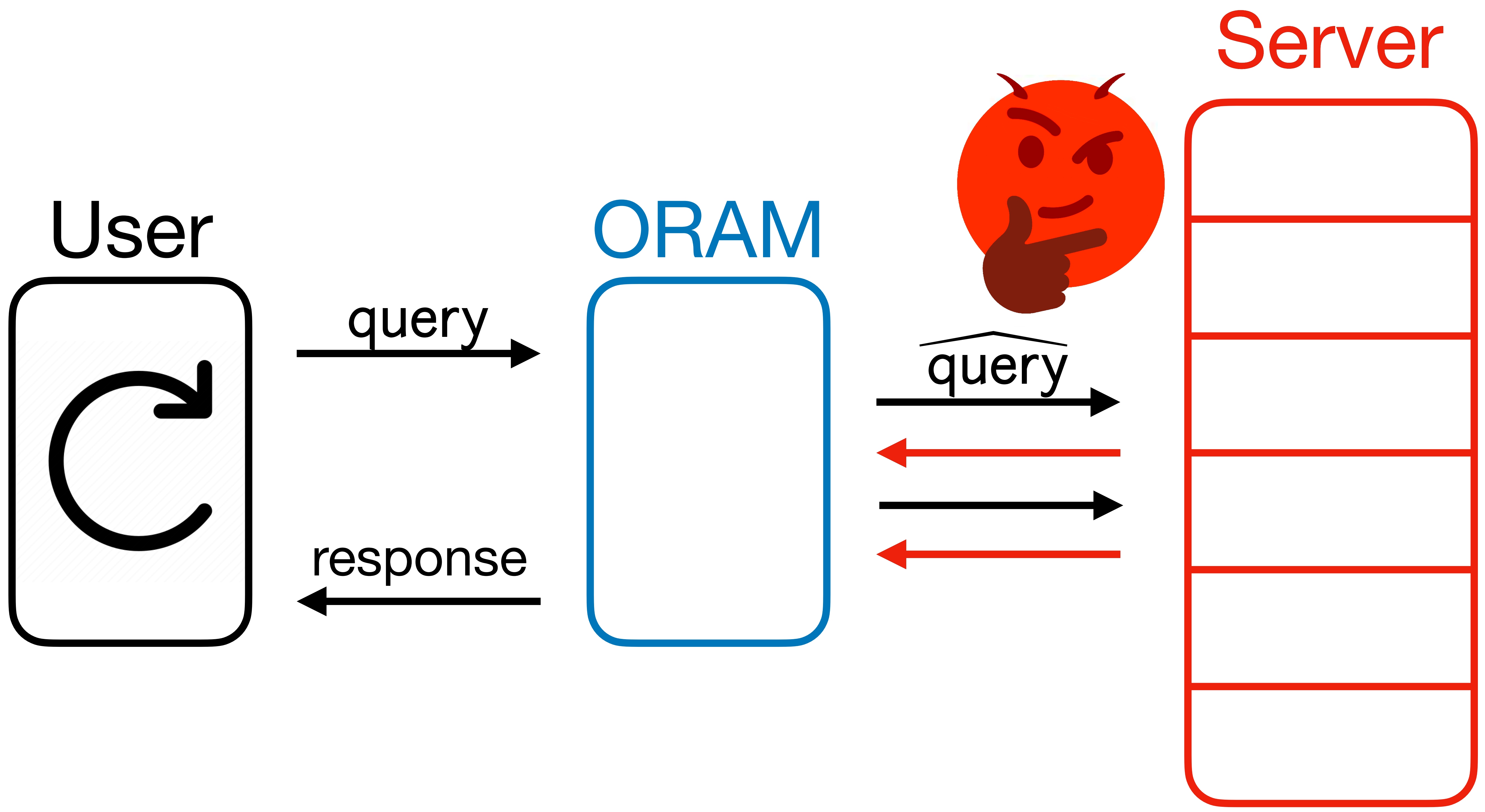
Power of the Adversary

- But up until now, we have assumed a **passive**, **honest-but-curious** RAM server that can try to learn something about the queries.
- In reality, ***an adversary can do more***! What about an **active, malicious** adversary that can **modify** the contents in the RAM?



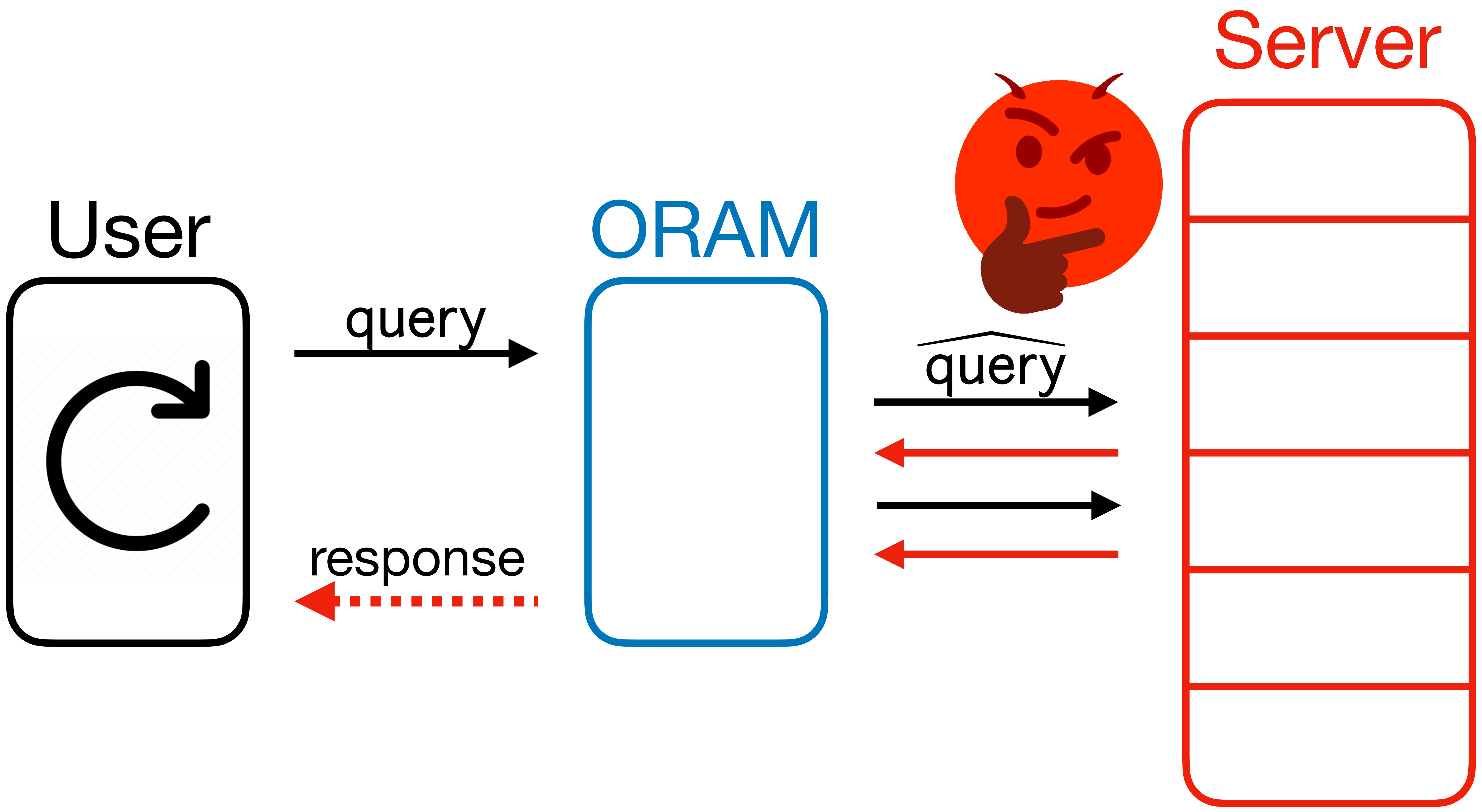
Malicious Security

- A malicious server breaks correctness



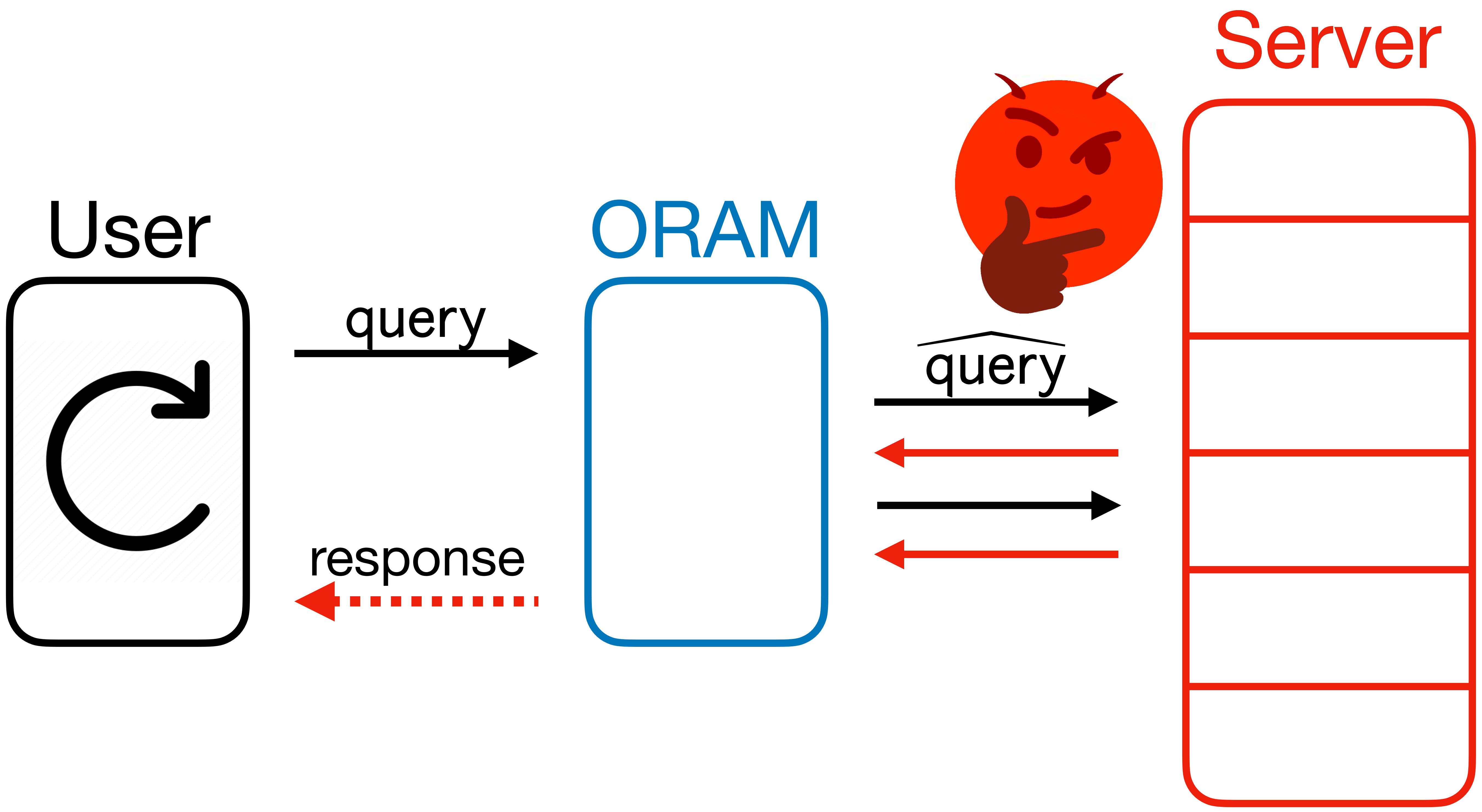
Malicious Security

- A malicious server breaks correctness



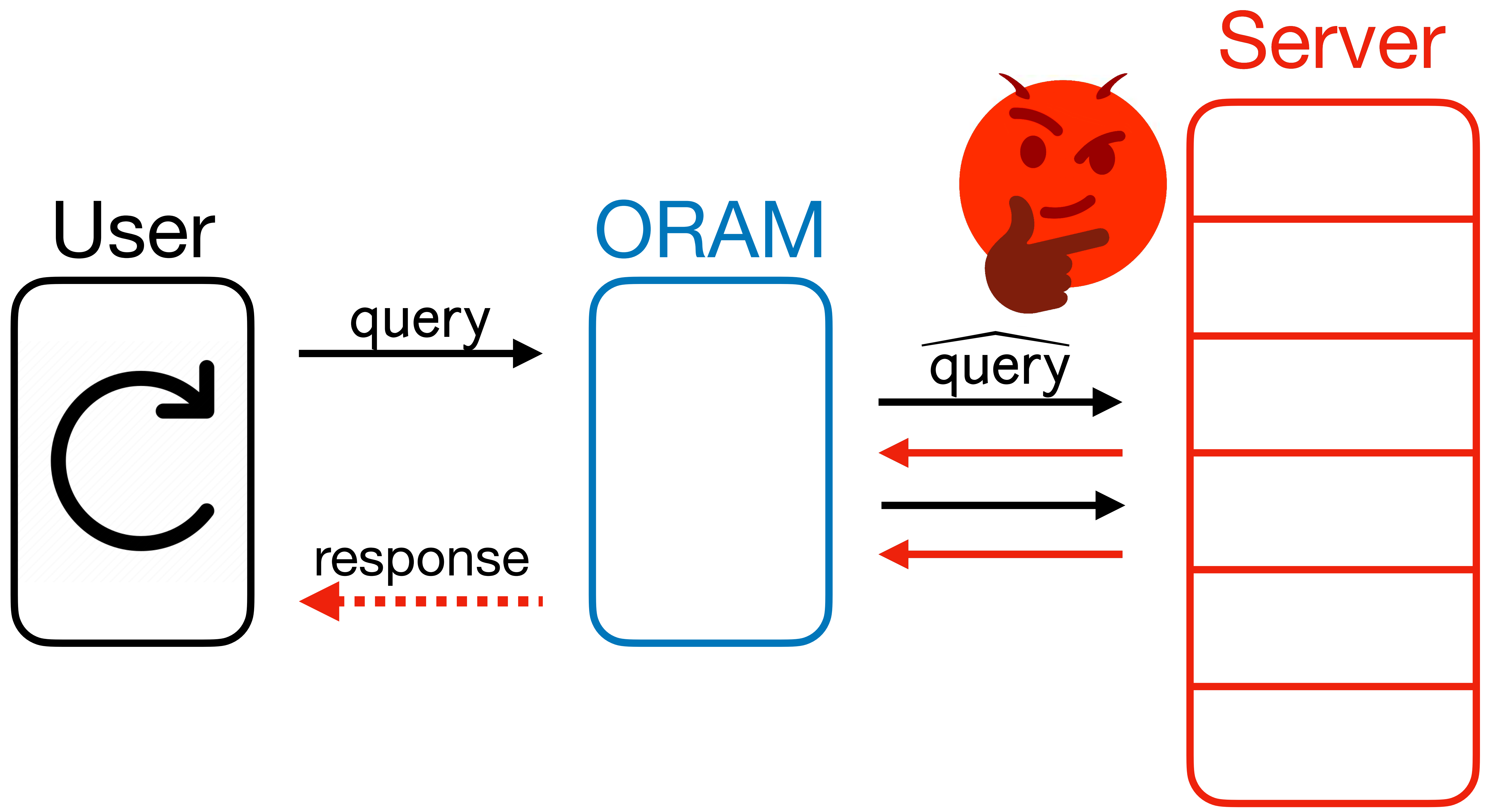
Malicious Security

- A malicious server breaks correctness *and also obliviousness*.



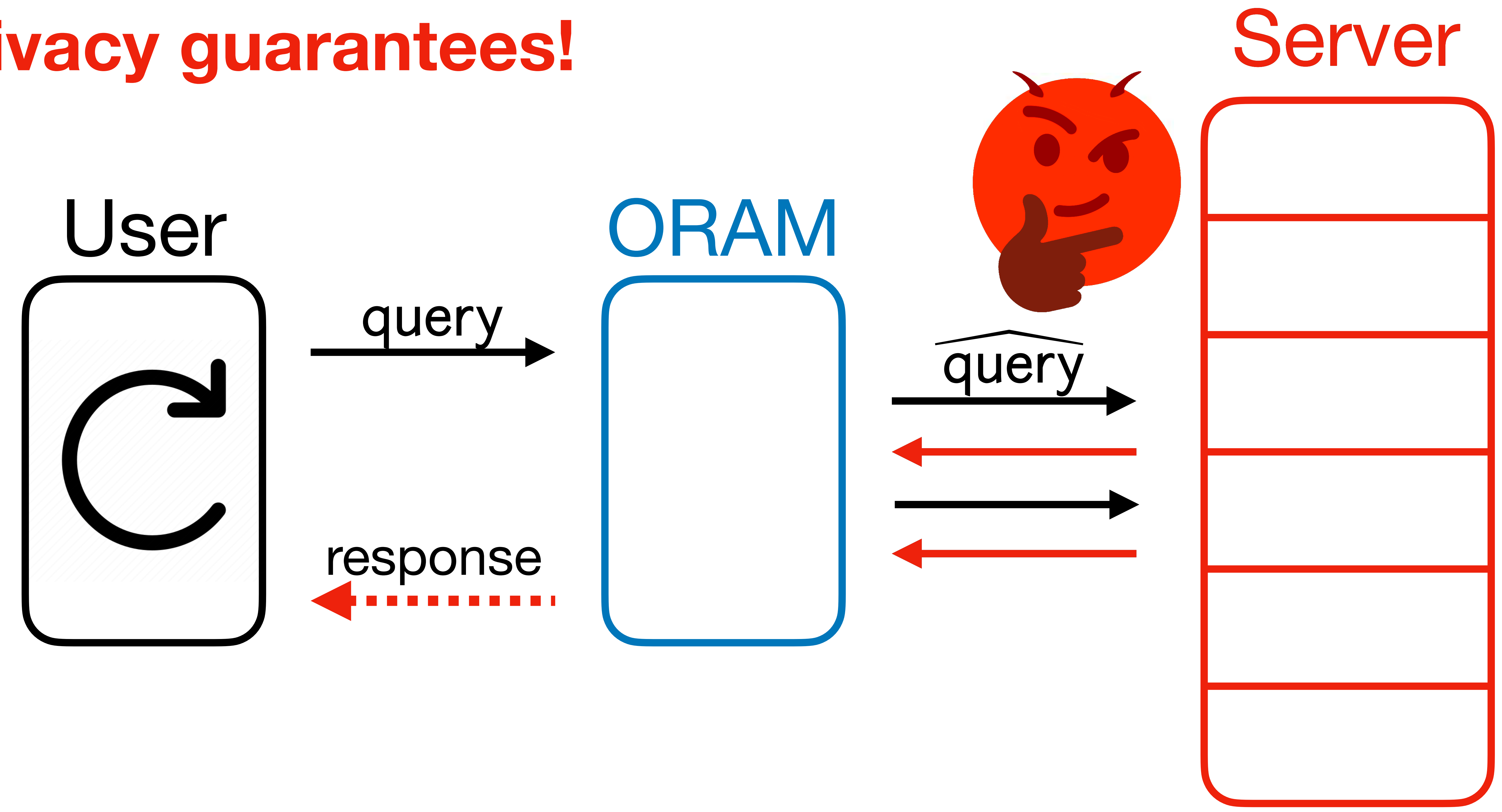
Malicious Security

- A malicious server breaks correctness *and also obliviousness*.
  - Why? After a corrupted server response, a standard ORAM has no obliviousness guarantee anymore. (This will be a big issue!)



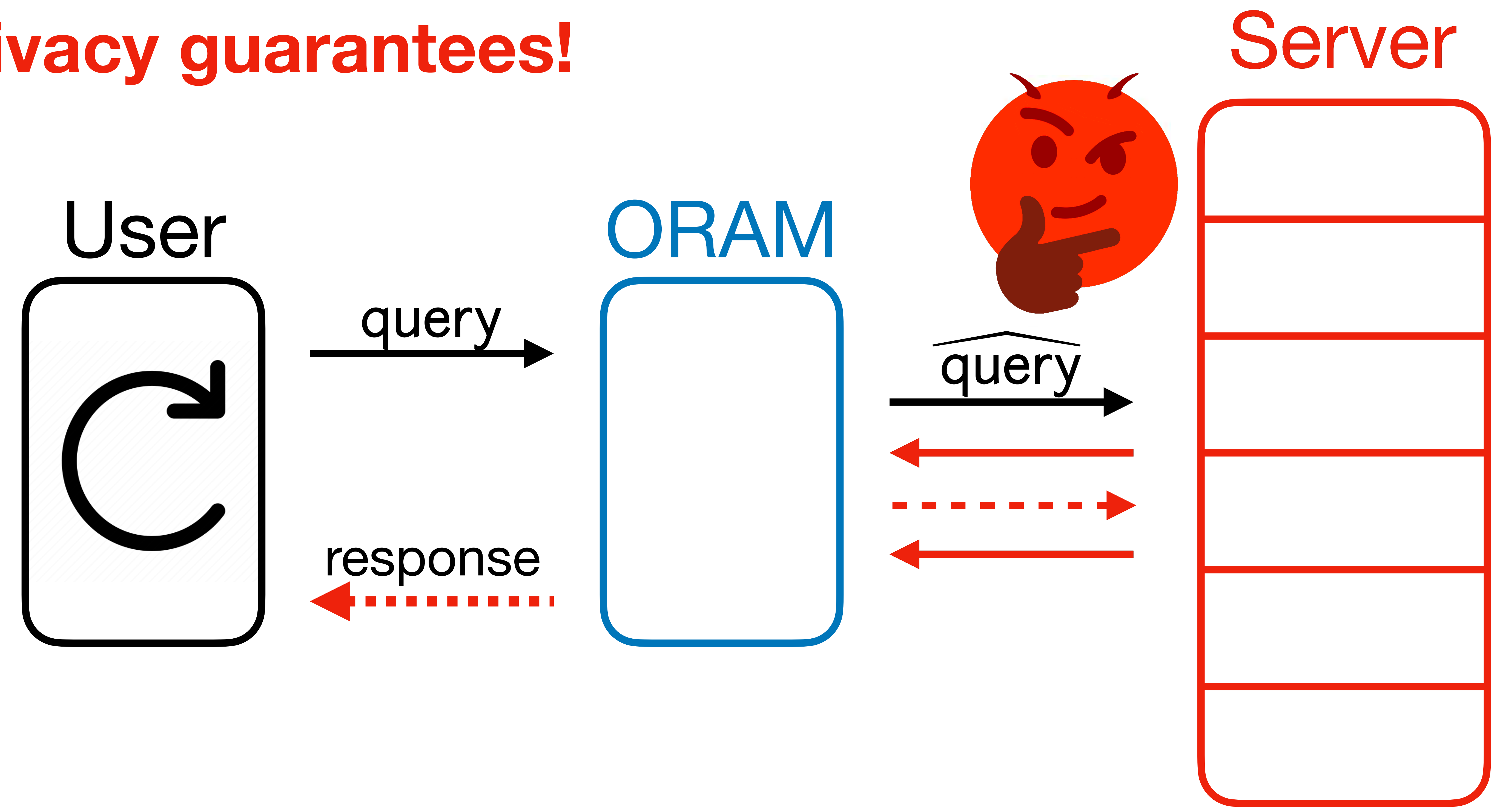
Malicious Security

- A malicious server breaks correctness *and also obliviousness*.
  - Why? After a corrupted server response, a standard ORAM has no obliviousness guarantee anymore. (This will be a big issue!)
- **No more privacy guarantees!**



Malicious Security

- A malicious server breaks correctness *and also obliviousness*.
  - Why? After a corrupted server response, a standard ORAM has no obliviousness guarantee anymore. (This will be a big issue!)
- **No more privacy guarantees!**



ORAM History (Malicious)

RAM of size N, word size $\omega(\log N)$, local space $O(1)$, assuming OWF

Work	**Overhead**	**Malicious?**
[Goldreich ’87]		
[Ostrovsky ’90, Goldreich-Ostrovsky ’96]		
Path ORAM [SvDSCFRYD ’12]		
PanORAMa [Patel-Persiano-Raykova-Yeo ’18]		
OptORAMa [AKLNPS ‘20, AKLS ’21]		

ORAM History (Malicious)

RAM of size N, word size $\omega(\log N)$, local space $O(1)$, assuming OWF

Work	Overhead	Malicious?
[Goldreich ’87]	$\sqrt{N} \log N$	
[Ostrovsky ’90, Goldreich-Ostrovsky ’96]	$\log^3 N$	
Path ORAM [SvDSCFRYD ’12]	$\log^2 N$	
PanORAMa [Patel-Persiano-Raykova-Yeo ’18]	$\log N \log \log N$	
OptORAMa [AKLNPS ‘20, AKLS ’21]	$\log N$	

Lower Bound: [Goldreich ’87, LN ’18, KL ’21]	$\Omega\left(\log N\right)$	

ORAM History (Malicious)

RAM of size N, word size $\omega(\log N)$, local space $O(1)$, assuming OWF

Work	Overhead	Malicious?
[Goldreich '87]	$\sqrt{N} \log N$	Yes
[Ostrovsky '90, Goldreich-Ostrovsky '96]	$\log^3 N$	Yes
Path ORAM [SvDSCFRYD '12]	$\log^2 N$	Yes
PanORAMa [Patel-Persiano-Raykova-Yeo '18]	$\log N \log \log N$	
OptORAMa [AKLNPS '20, AKLS '21]	$\log N$	

Lower Bound: [Goldreich '87, LN '18, KL '21]	$\Omega\left(\log N\right)$	

ORAM History (Malicious)

RAM of size N, word size $\omega(\log N)$, local space $O(1)$, assuming OWF

Work	Overhead	Malicious?
[Goldreich '87]	$\sqrt{N} \log N$	Yes
[Ostrovsky '90, Goldreich-Ostrovsky '96]	$\log^3 N$	Yes
Path ORAM [SvDSCFRYD '12]	$\log^2 N$	Yes
PanORAMa [Patel-Persiano-Raykova-Yeo '18]	$\log N \log \log N$	No
OptORAMa [AKLNPS '20, AKLS '21]	$\log N$	No

Lower Bound: [Goldreich '87, LN '18, KL '21]	$\Omega\left(\log N\right)$	

ORAM History (Malicious)

RAM of size N, word size $\omega(\log N)$, local space $O(1)$, assuming OWF

Work	Overhead	Malicious?
[Goldreich '87]	$\sqrt{N} \log N$	Yes
[Ostrovsky '90, Goldreich-Ostrovsky '96]	$\log^3 N$	Yes
Path ORAM [SvDSCFRYD '12]	$\log^2 N$	Yes
PanORAMa [Patel-Persiano-Raykova-Yeo '18]	$\log N \log \log N$	No
OptORAMa [AKLNPS '20, AKLS '21]	$\log N$	No

Attacks!

Lower Bound: [Goldreich '87, LN '18, KL '21]	$\Omega\left(\log N\right)$	

ORAM History (Malicious)

RAM of size N, word size $\omega(\log N)$, local space $O(1)$, assuming OWF

Work	Overhead	Malicious?
[Goldreich ’87]	$\sqrt{N}\log N$	Yes
[Ostrovsky ’90, Goldreich-Ostrovsky ’96]	$\log^3 N$	Yes
Path ORAM [SvDSCFRYD ’12]	$\log^2 N$	Yes
PanORAMa [Patel-Persiano-Raykova-Yeo ’18]	$\log N \log\log N$	No
OptORAMa [AKLNPS ‘20, AKLS ’21]	$\log N$	No

Attacks!

Lower Bound: [Goldreich ’87, LN ’18, KL ’21]	$\Omega\left(\log N\right)$	Any stronger?

Optimal Maliciously Secure ORAM

Optimal Maliciously Secure ORAM

Question: Is there a maliciously secure ORAM with $O(\log N)$ overhead?

Optimal Maliciously Secure ORAM

Question: Is there a maliciously secure ORAM with $O(\log N)$ overhead?

Theorem [M.-Vafa ’23]: Yes!

Optimal Maliciously Secure ORAM

Question: Is there a maliciously secure ORAM with $O(\log N)$ overhead?

Theorem [M.-Vafa ’23]: Yes!

Assuming one-way functions, we construct **MacORAMa**, a maliciously secure ORAM with $O(\log N)$ overhead and $O(1)$ local space*.

Optimal Maliciously Secure ORAM?

<u>Theorem</u> <u>[M.-Vafa '23]</u>: Assuming one-way functions, there is a maliciously secure ORAM with $O(\log N)$ overhead and $O(1)$ word local space*.

- As before, $O(\log N)$ overhead is optimal – malicious security for free!

Optimal Maliciously Secure ORAM?

Theorem [M.-Vafa '23]: Assuming one-way functions, there is a maliciously secure ORAM with $O(\log N)$ overhead and $O(1)$ word local space*.

- As before, $O(\log N)$ overhead is optimal – malicious security for free!
- Maliciously secure ORAM still in passive storage model! No **extra work** for honest server.

Optimal Maliciously Secure ORAM?

Theorem [M.-Vafa '23]: Assuming one-way functions, there is a maliciously secure ORAM with $O(\log N)$ overhead and $O(1)$ word local space*.

- As before, $O(\log N)$ overhead is optimal – malicious security for free!
- Maliciously secure ORAM still in passive storage model! No **extra work** for honest server.
- OWFs are also *necessary* for maliciously secure ORAM. [Naor, Rothblum '05]

ORAM History (Malicious)

RAM of size N, word size $\omega(\log N)$, local space $O(1)$

Work	Overhead	Malicious?
[Goldreich '87]	$\sqrt{N} \log N$	Yes
[Ostrovsky '90, Goldreich-Ostrovsky '96]	$\log^3 N$	Yes
Path ORAM [SvDSCFRYD '12]	$\log^2 N$	Yes
PanORAMa [Patel-Persiano-Raykova-Yeo '18]	$\log N \log \log N$	No
OptORAMa [AKLNPS '20, AKLS '21]	$\log N$	No

Lower Bound: [Goldreich '87, LN '18, KL '21]	$\Omega\left(\log N\right)$	Any stronger?

ORAM History (Malicious)

RAM of size N, word size $\omega(\log N)$, local space $O(1)$

Work	**Overhead**	**Malicious?**
[Goldreich '87]	$\sqrt{N} \log N$	Yes
[Ostrovsky '90, Goldreich-Ostrovsky '96]	$\log^3 N$	Yes
Path ORAM [SvDSCFRYD '12]	$\log^2 N$	Yes
PanORAMa [Patel-Persiano-Raykova-Yeo '18]	$\log N \log \log N$	No
OptORAMa [AKLNPS '20, AKLS '21]	$\log N$	No
MacORAMa [M.-Vafa '22]	$\log N$	**Yes**
Lower Bound: [Goldreich '87, LN '18, KL '21]	$\Omega\left(\log N\right)$	$\Omega\left(\log N\right)$

Our Construction

We start with **OptORAMa** [Asharov, Komargodski, Lin, Nayak, Peserico, Shi] - a **honest-but-curious** ORAM with **optimal** $O(\log N)$ overhead.

Our Construction

We start with **OptORAMa** [Asharov, Komargodski, Lin, Nayak, Peserico, Shi] - a **honest-but-curious** ORAM with **optimal** $O(\log N)$ overhead.

Our Construction

We start with **OptORAMa** [Asharov, Komargodski, Lin, Nayak, Peserico, Shi] - a **honest-but-curious** ORAM with **optimal** $O(\log N)$ overhead.

Our Construction

We start with **OptORAMa** [Asharov, Komargodski, Lin, Nayak, Peserico, Shi] - a **honest-but-curious** ORAM with **optimal** $O(\log N)$ overhead.

- **Technique #1:** Memory checking (e.g. Merkle trees).

Our Construction

We start with **OptORAMa** [Asharov, Komargodski, Lin, Nayak, Peserico, Shi] - a **honest-but-curious** ORAM with **optimal** $O(\log N)$ overhead.

- **Technique #1:** Memory checking (e.g. Merkle trees).
- **Technique #2:** Authenticate all writes with MACs.

Our Construction

We start with **OptORAMa** [Asharov, Komargodski, Lin, Nayak, Peserico, Shi] - a **honest-but-curious** ORAM with **optimal** $O(\log N)$ overhead.

- **Technique #1:** Memory checking (e.g. Merkle trees).
- **Technique #2:** Authenticate all writes with MACs.
- **Technique #3:** MAC and Cheese!

Technique #1: Memory Checking

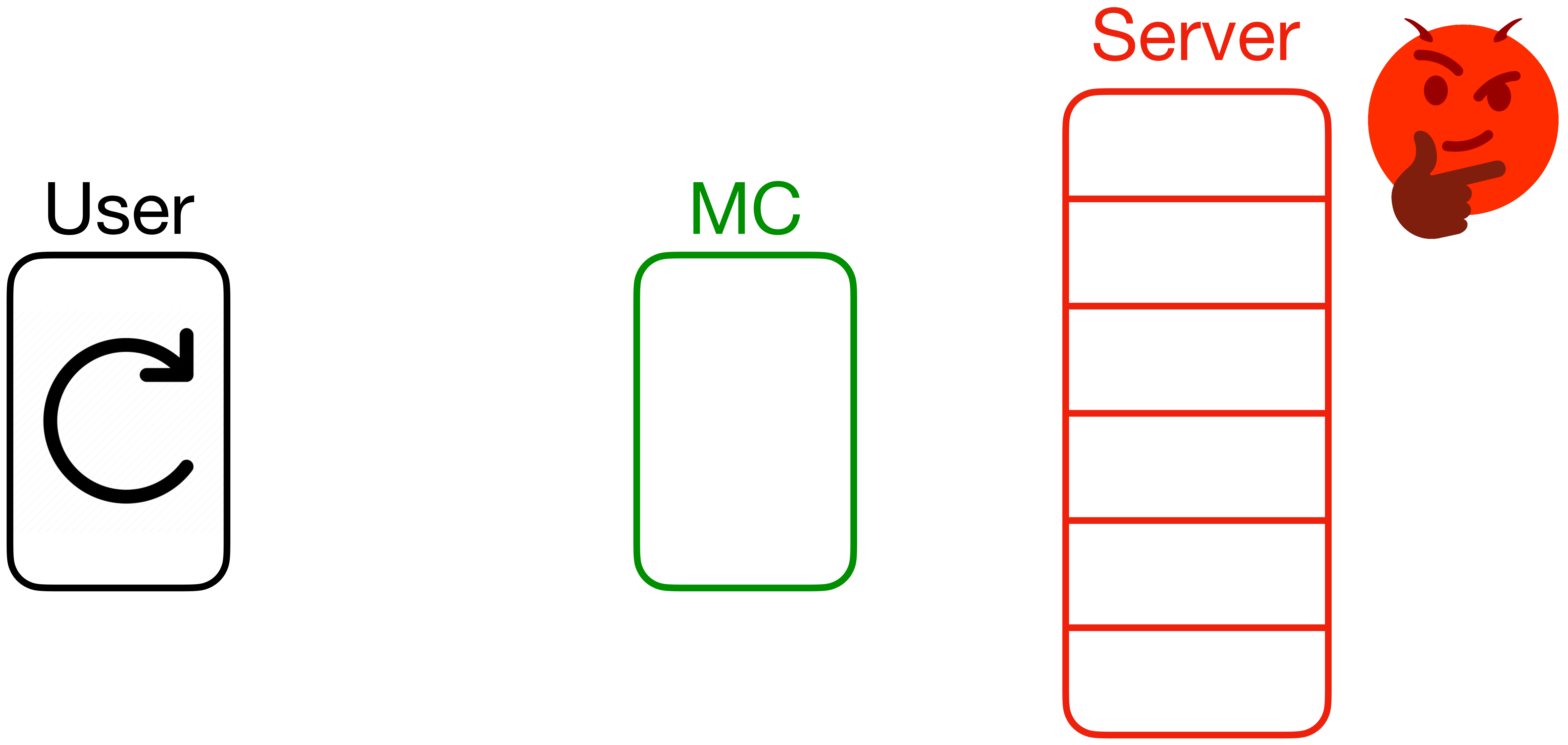
Technique #1: Memory Checking

- Overlay a **Merkle** tree, or more generally a **Memory Checker** (MC).

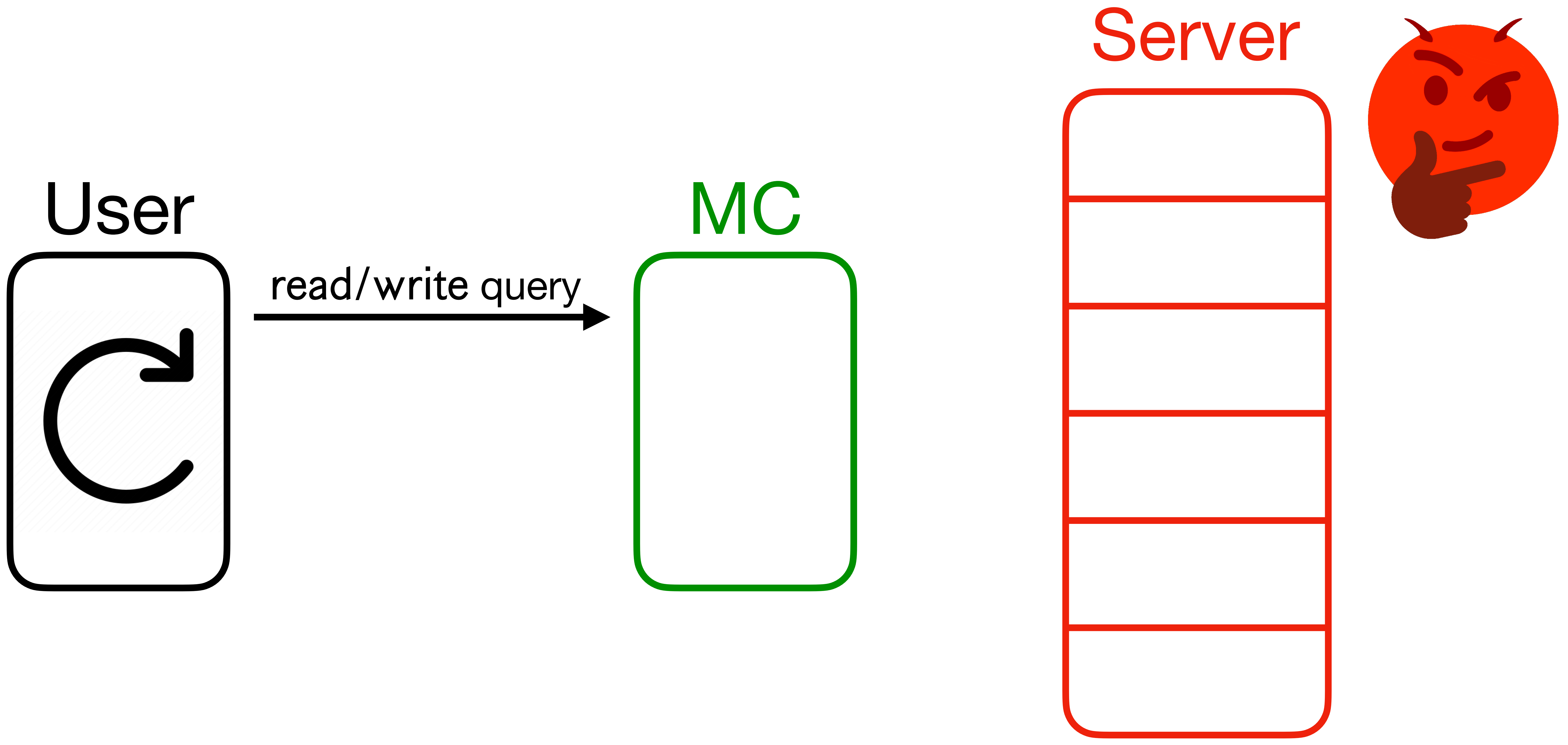
Technique #1: Memory Checking

- Overlay a **Merkle** tree, or more generally a **Memory Checker** (MC).
- A memory checker is a protocol that detects whether a malicious server tampered with RAM. [Blum, Evans, Gemmell, Kannan, Naor ’94]

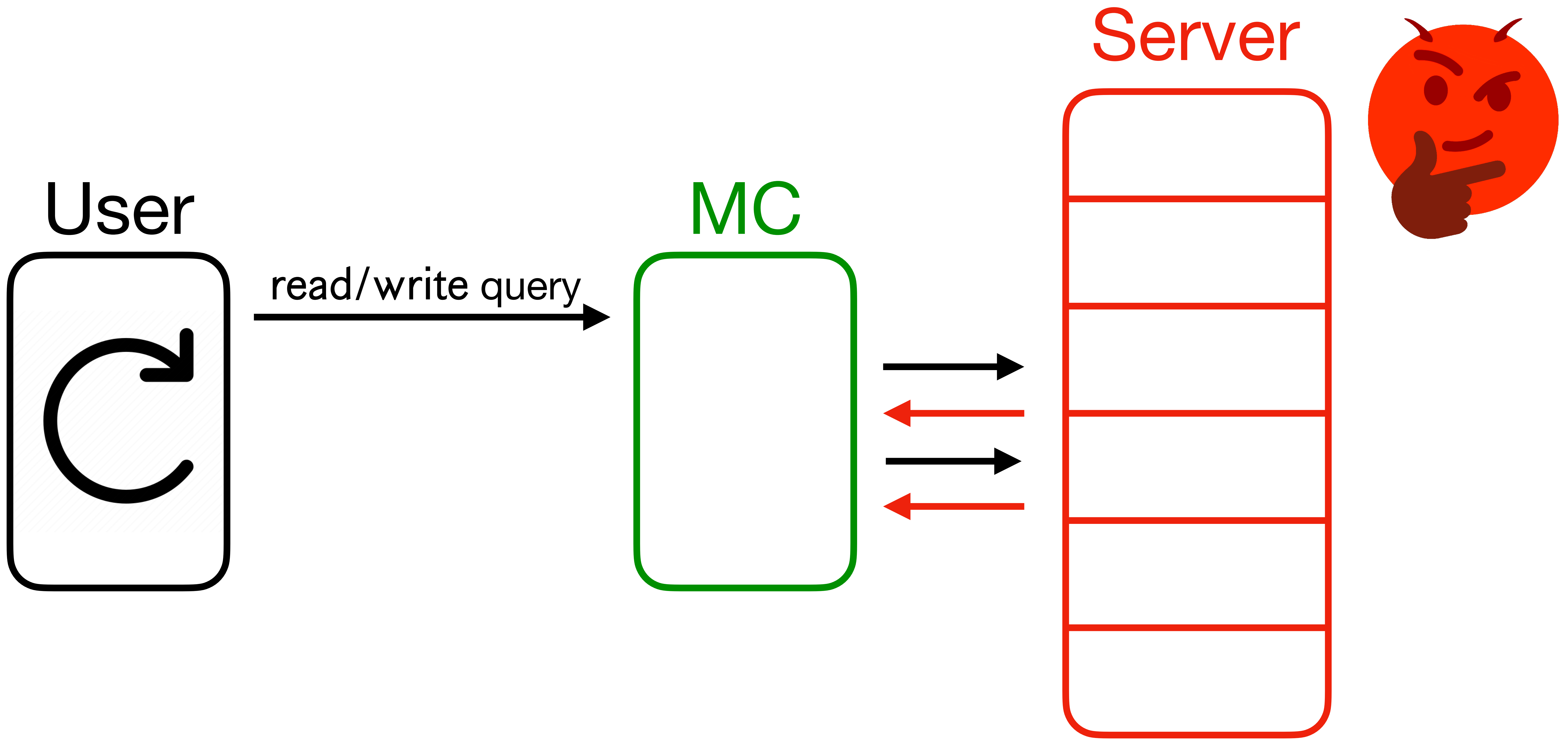
Technique #1: Memory Checking



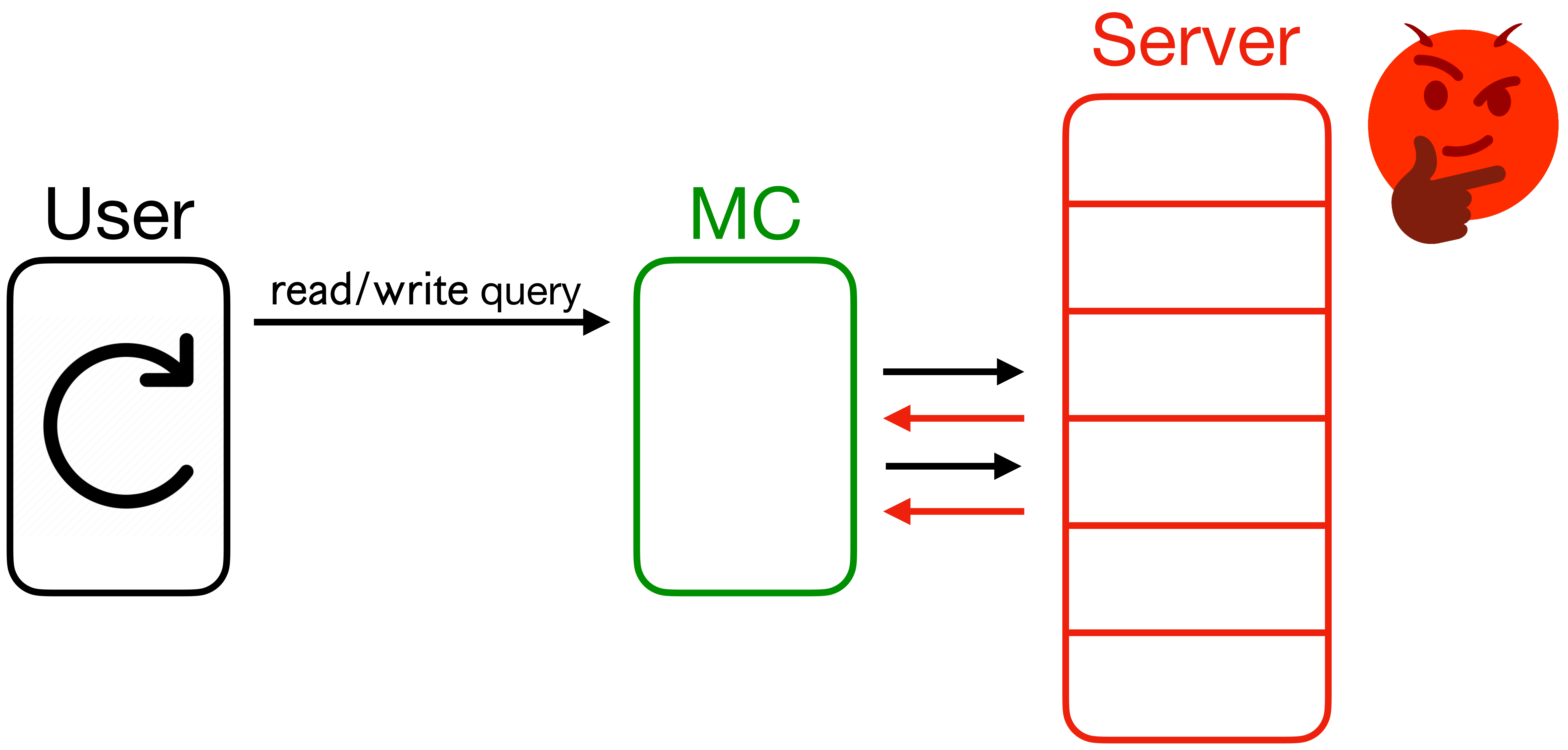
Technique #1: Memory Checking



Technique #1: Memory Checking

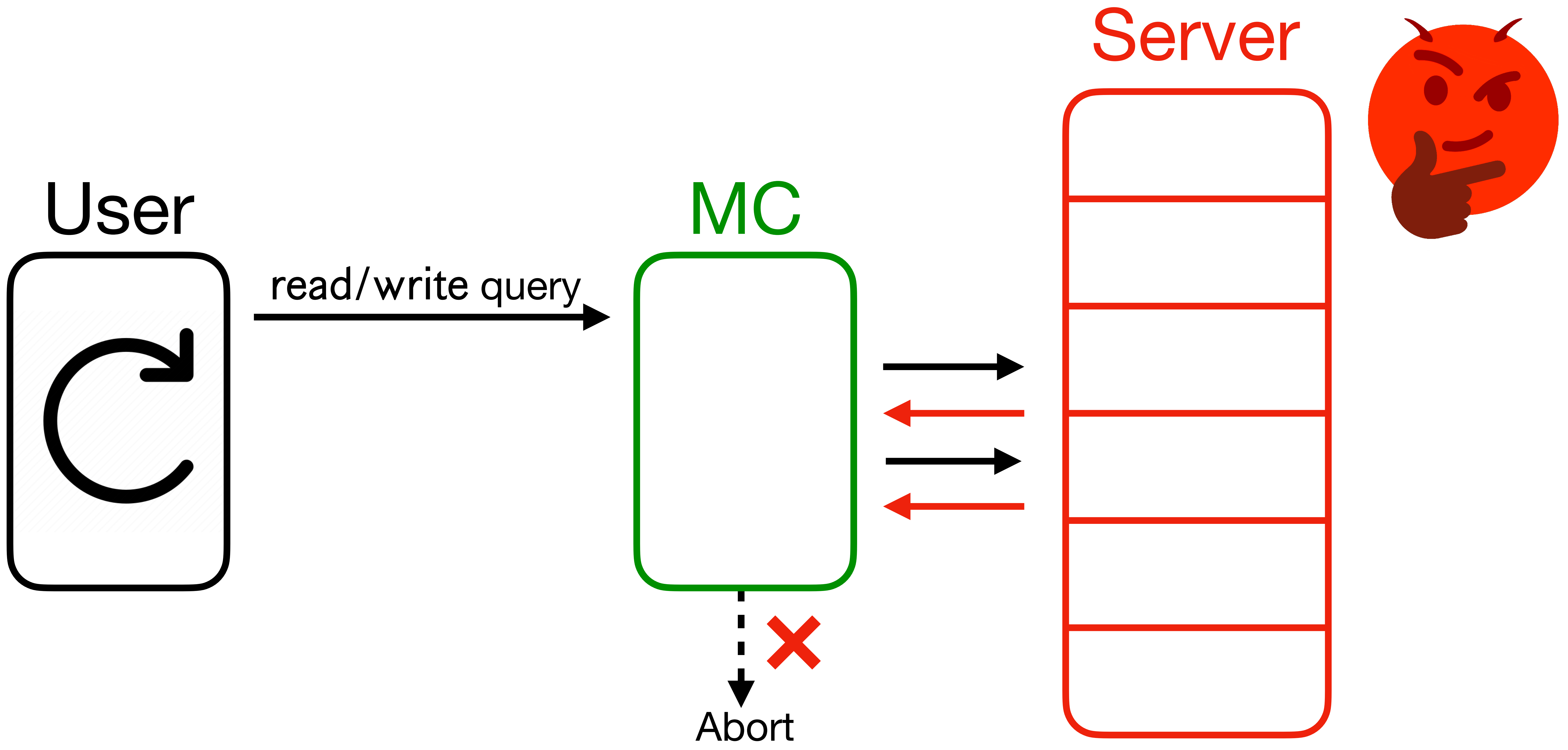


Technique #1: Memory Checking



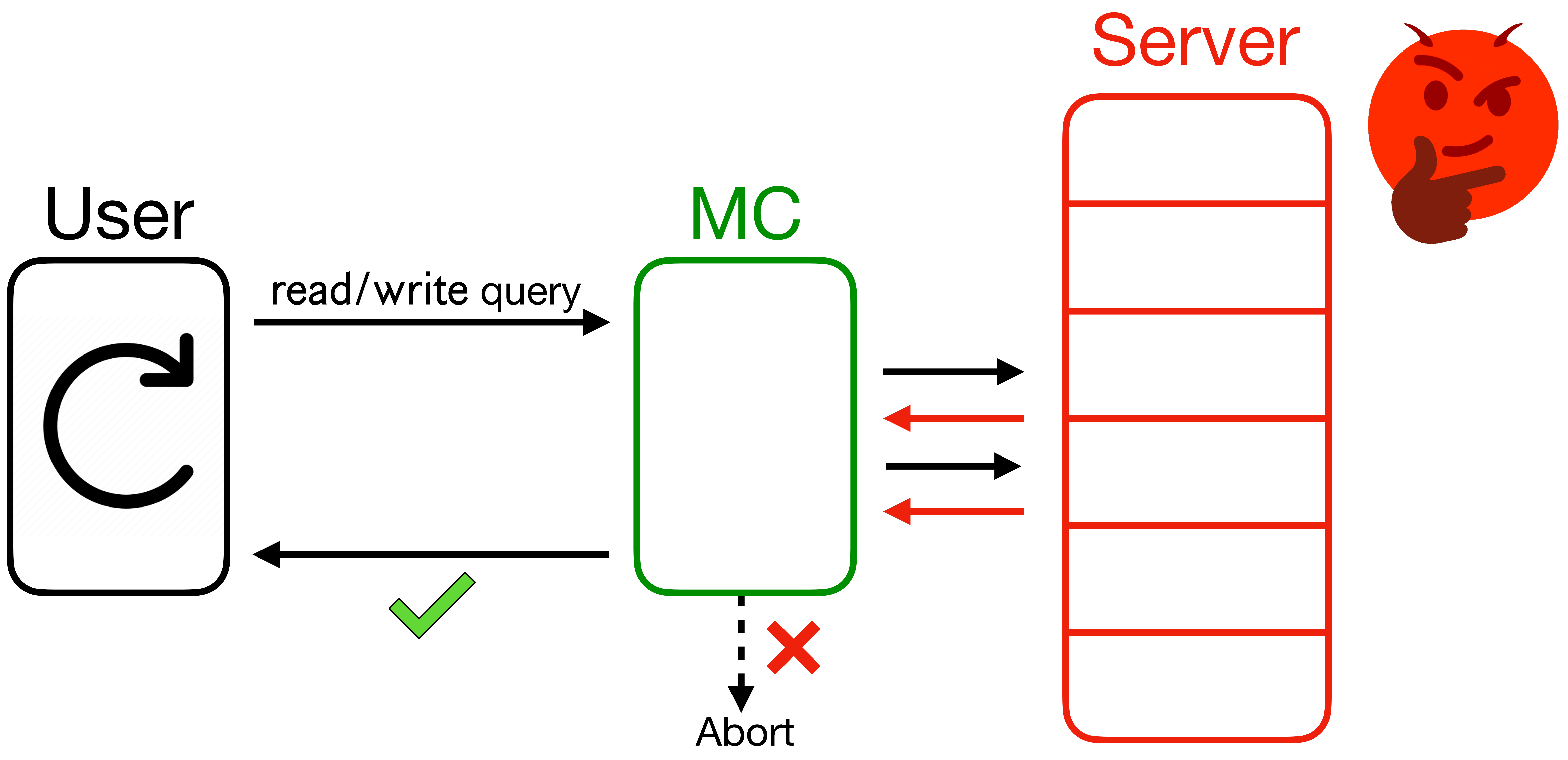
- **Correctness**: For any PPT malicious server, MC either **aborts** or gives correct responses.

Technique #1: Memory Checking



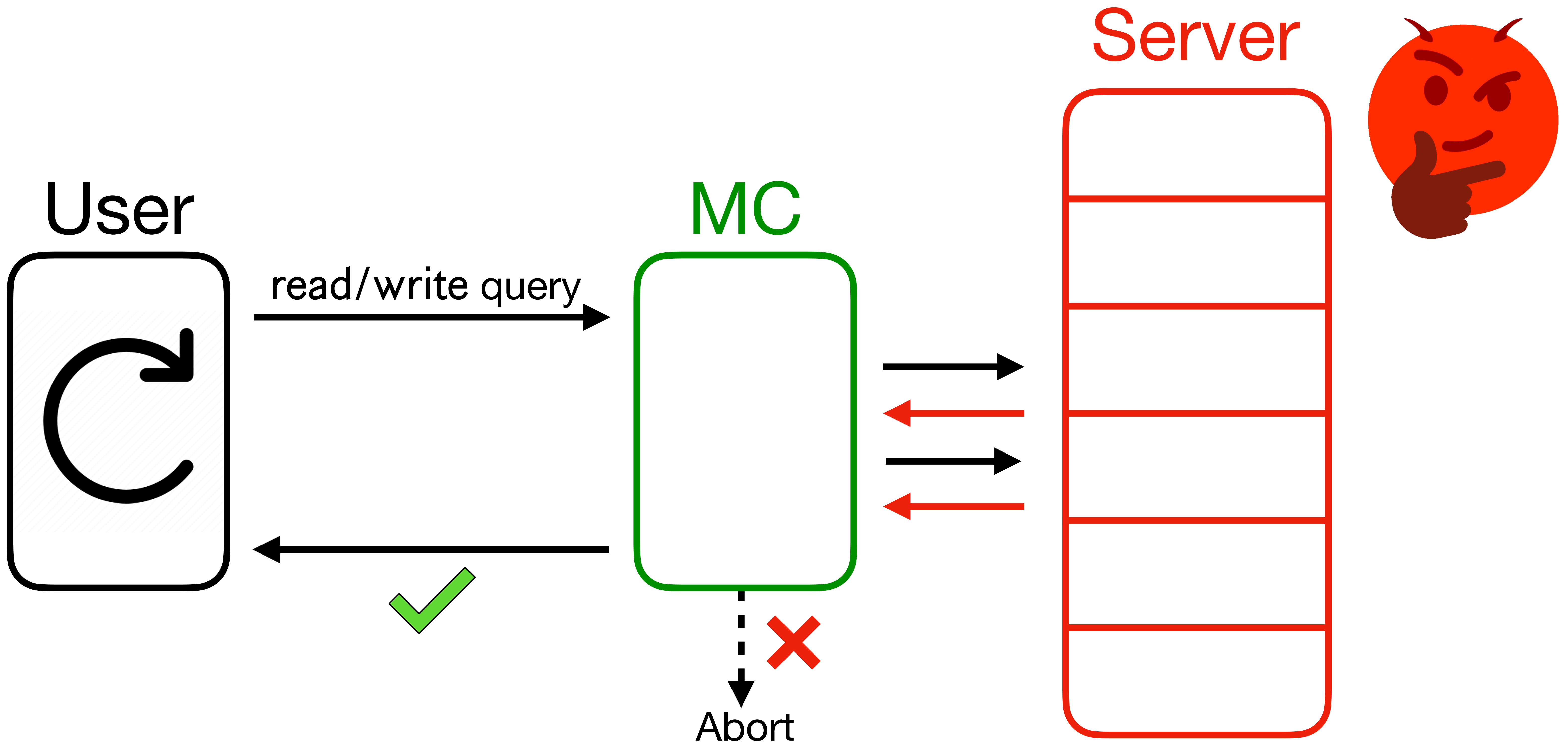
- **Correctness**: For any PPT malicious server, MC either **aborts** or gives correct responses.

Technique #1: Memory Checking



- **Correctness**: For any PPT malicious server, MC either **aborts** or gives correct responses.

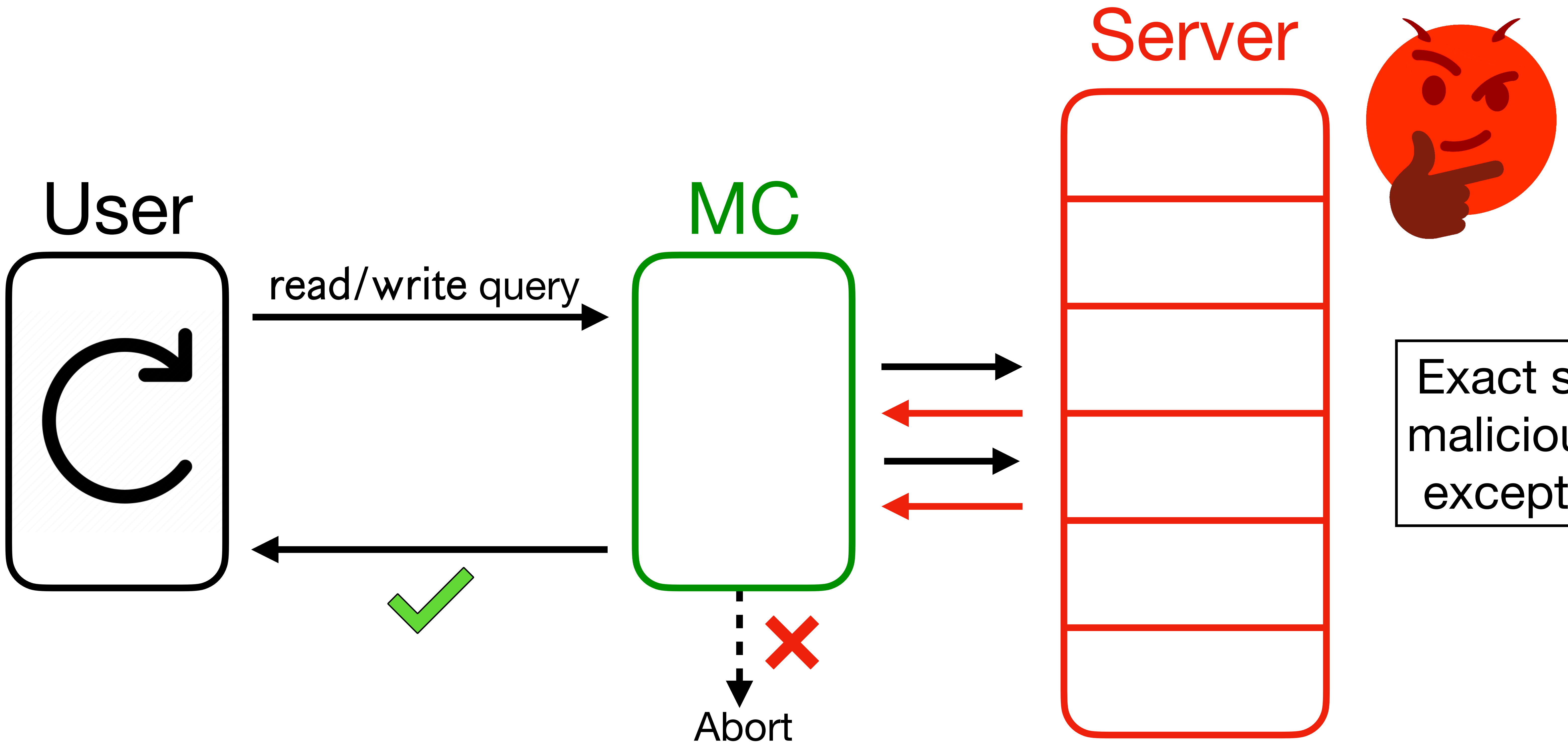
Technique #1: Memory Checking



- **Correctness**: For any PPT malicious server, MC either **aborts** or gives correct responses.

- **Completeness**: If the server behaved honestly, MC doesn't abort.

Technique #1: Memory Checking



Exact same properties as maliciously secure ORAM, except no obliviousness.

- **Correctness**: For any PPT malicious server, MC either **aborts** or gives correct responses.
- **Completeness**: If the server behaved honestly, MC doesn't abort.

Memory Checking Efficiency

Memory Checking Efficiency

- Just like ORAM, **local space** and **overhead** are two main efficiency metrics (local space N trivial). For $O(1)$ local space:

Memory Checking Efficiency

- Just like ORAM, **local space** and **overhead** are two main efficiency metrics (local space N trivial). For $O(1)$ local space:
  - Memory checking with $o(N)$ overhead implies OWF. [Naor-Rothblum '05]

Memory Checking Efficiency

- Just like ORAM, **local space** and **overhead** are two main efficiency metrics (local space N trivial). For $O(1)$ local space:
  - Memory checking with $o(N)$ overhead implies OWF. [Naor-Rothblum '05]
  - Best known constructions have $O(\log N)$ overhead.* [Blum et al. '94]

*More accurately, bandwidth (in terms of bits), not overhead (in case word sizes differ).

Memory Checking Efficiency

- Just like ORAM, **local space** and **overhead** are two main efficiency metrics (local space N trivial). For $O(1)$ local space:
  - Memory checking with $o(N)$ overhead implies OWF. [Naor-Rothblum ’05]
  - Best known constructions have $O(\log N)$ overhead.* [Blum et al. ’94]
    - E.g., Merkle trees. Store Merkle root and access paths in binary tree.

*More accurately, bandwidth (in terms of bits), not overhead (in case word sizes differ).

Memory Checking Efficiency

- Just like ORAM, **local space** and **overhead** are two main efficiency metrics (local space N trivial). For $O(1)$ local space:
  - Memory checking with $o(N)$ overhead implies OWF. [Naor-Rothblum ’05]
  - Best known constructions have $O(\log N)$ overhead.* [Blum et al. ’94]
    - E.g., Merkle trees. Store Merkle root and access paths in binary tree.
  - **Lower bound** of $\Omega(\log N / \log \log N)$ overhead for deterministic, non-adaptive memory checkers (which the existing constructions are).

[Dwork et al. ’09]

*More accurately, bandwidth (in terms of bits), not overhead (in case word sizes differ).

Technique #1: Memory Checking

Technique #1: Memory Checking

- Intuitively, memory checking seems to solve the issue of a tampering adversary.

Technique #1: Memory Checking

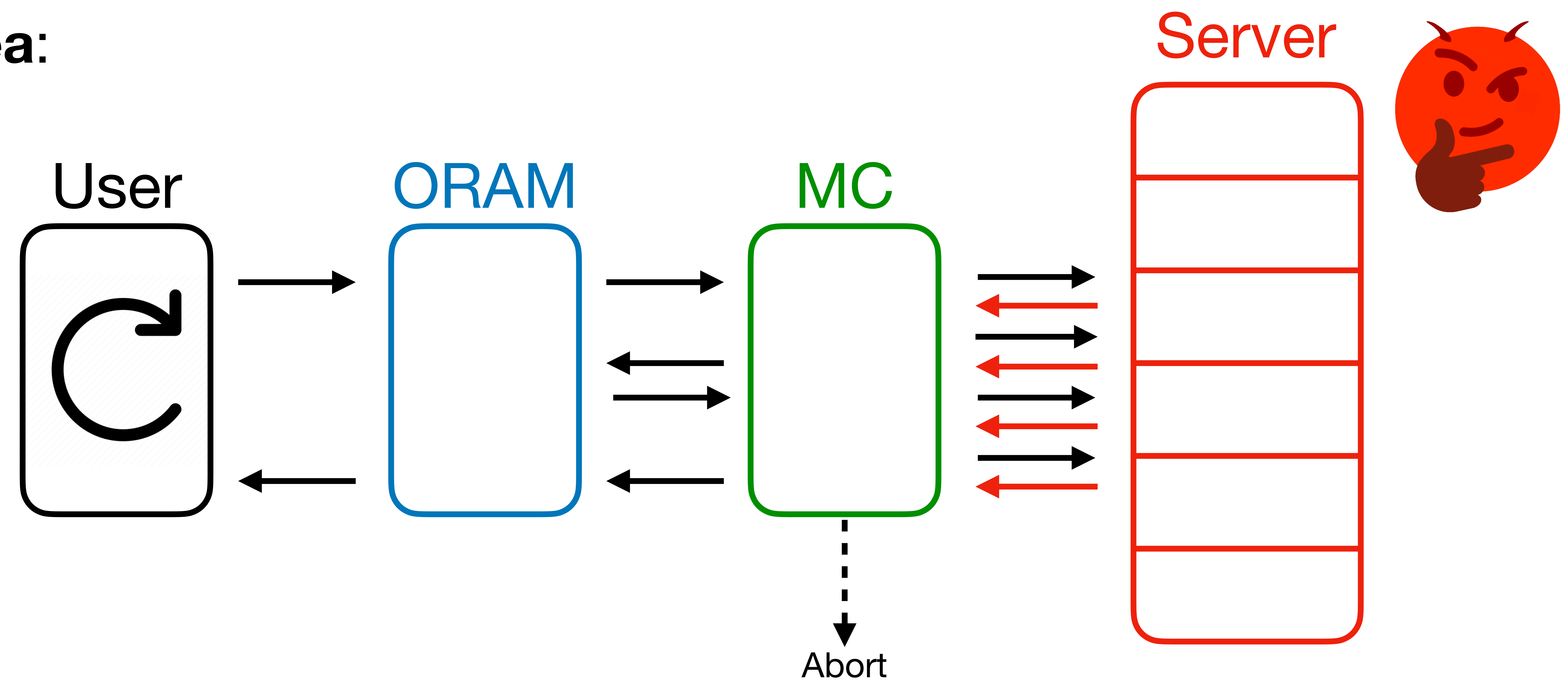
- Intuitively, memory checking seems to solve the issue of a tampering adversary.
- **Theorem**: Honest-but-curious ORAM + MC = maliciously secure ORAM.

Technique #1: Memory Checking

- Intuitively, memory checking seems to solve the issue of a tampering adversary.
- **Theorem**: Honest-but-curious ORAM + MC = maliciously secure ORAM.
- **Idea**:

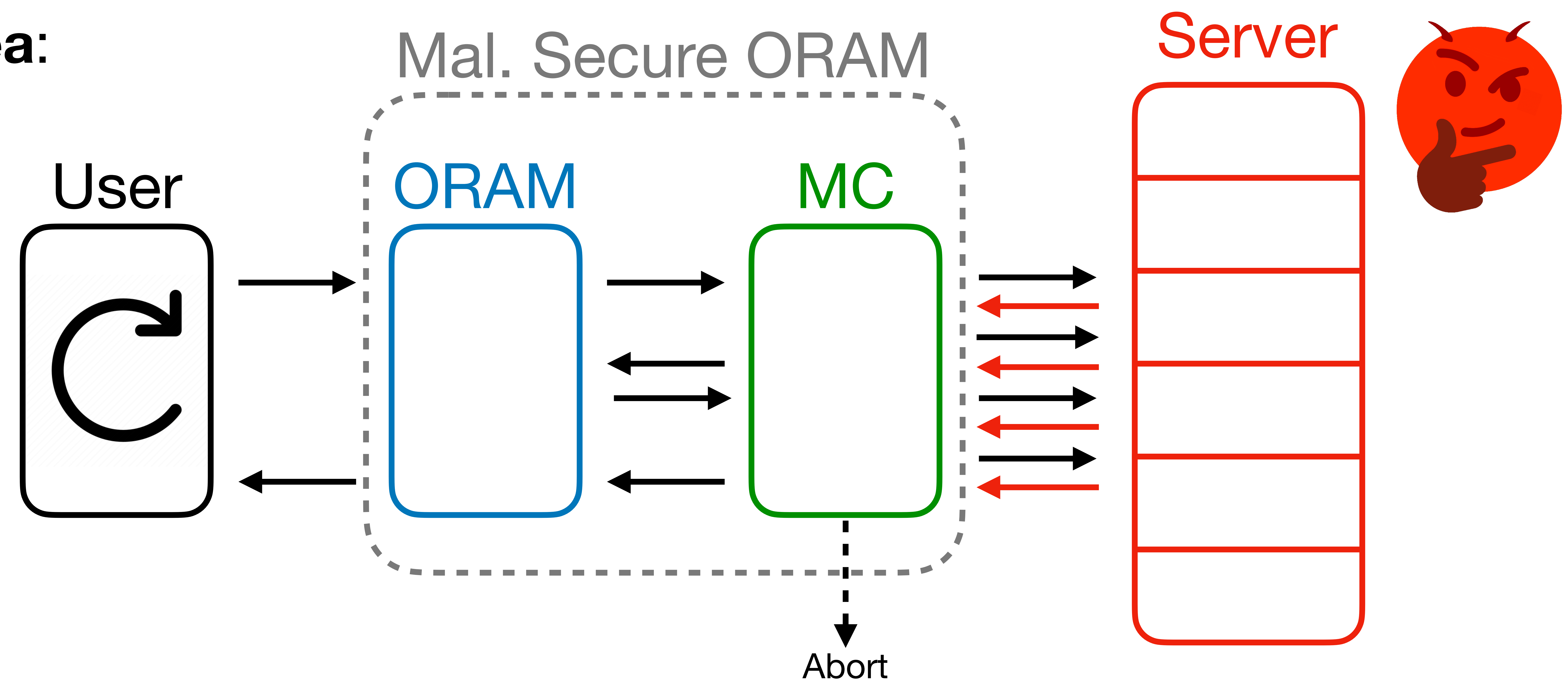
Technique #1: Memory Checking

- Intuitively, memory checking seems to solve the issue of a tampering adversary.
- **Theorem**: Honest-but-curious ORAM + MC = maliciously secure ORAM.
- **Idea**:



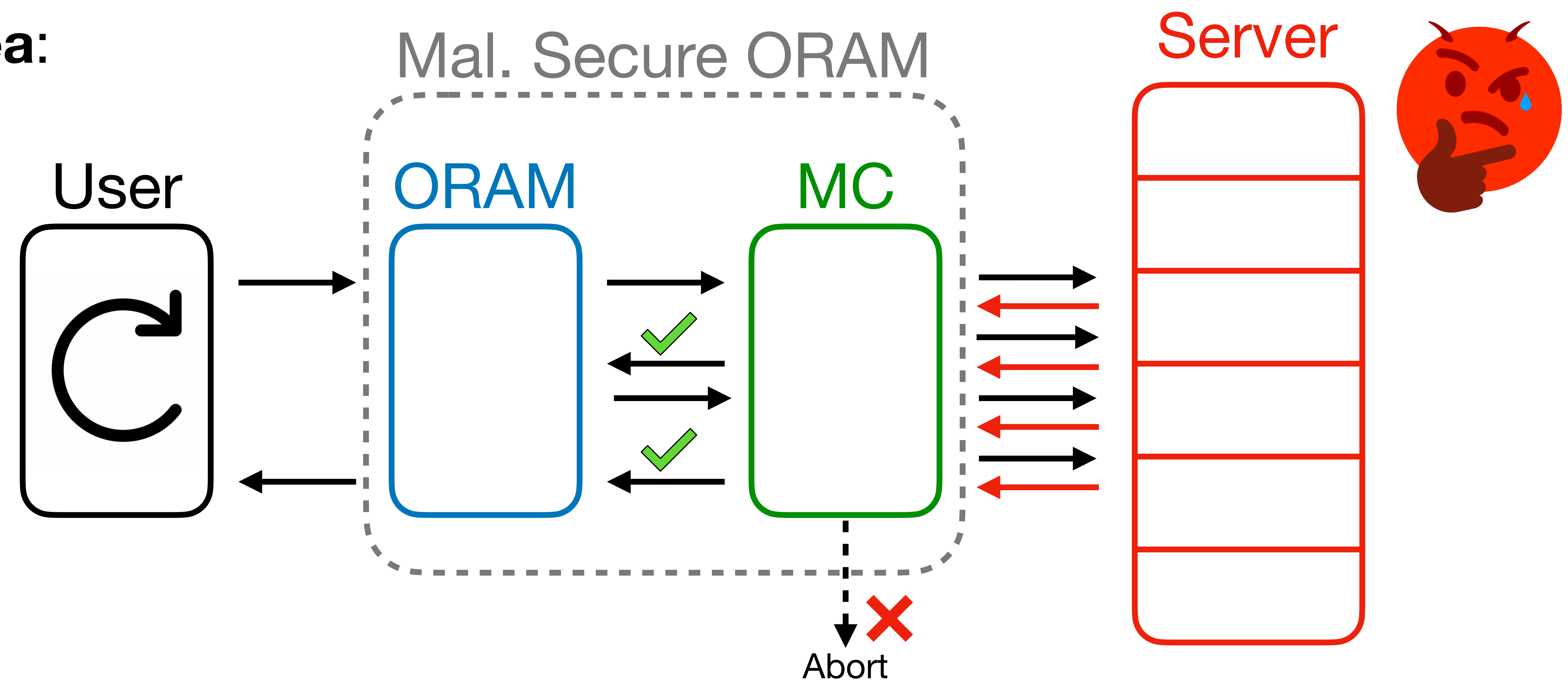
Technique #1: Memory Checking

- Intuitively, memory checking seems to solve the issue of a tampering adversary.
- **Theorem**: Honest-but-curious ORAM + MC = maliciously secure ORAM.
- **Idea**:



Technique #1: Memory Checking

- Intuitively, memory checking seems to solve the issue of a tampering adversary.
- **Theorem**: Honest-but-curious ORAM + MC = maliciously secure ORAM.
- **Idea**:



Technique #1: Memory Checking

- Great! But this isn’t efficient enough.

Technique #1: Memory Checking

- Great! But this isn’t efficient enough.

Technique #1: Memory Checking

- Great! But this isn't efficient enough.

$$\mathsf{Overhead}(\mathsf{ORAM}_{\mathsf{Mal}}) = \mathsf{Overhead}(\mathsf{ORAM}_{\mathsf{HBC}}) \cdot \mathsf{Overhead}(\mathsf{MC})$$

Technique #1: Memory Checking

- Great! But this isn't efficient enough.

$$\text{Overhead}(\text{ORAM}_{\text{Mal}}) = \underset{\substack{\uparrow \\ \log N}}{\text{Overhead}(\text{ORAM}_{\text{HBC}})} \cdot \text{Overhead}(\text{MC})$$

Technique #1: Memory Checking

- Great! But this isn't efficient enough.

$$\mathsf{Overhead}(\mathsf{ORAM}_{\mathsf{Mal}}) = \underset{\substack{\uparrow \\ \log N}}{\mathsf{Overhead}(\mathsf{ORAM}_{\mathsf{HBC}})} \cdot \underset{\substack{\uparrow \\ \log N}}{\mathsf{Overhead}(\mathsf{MC})}$$

Technique #1: Memory Checking

- Great! But this isn't efficient enough.

$$\underset{\substack{\Big\uparrow \\ \log^2(N)}}{\text{Overhead}(\text{ORAM}_{\text{Mal}})} = \underset{\substack{\Big\uparrow \\ \log N}}{\text{Overhead}(\text{ORAM}_{\text{HBC}})} \cdot \underset{\substack{\Big\uparrow \\ \log N}}{\text{Overhead}(\text{MC})}$$

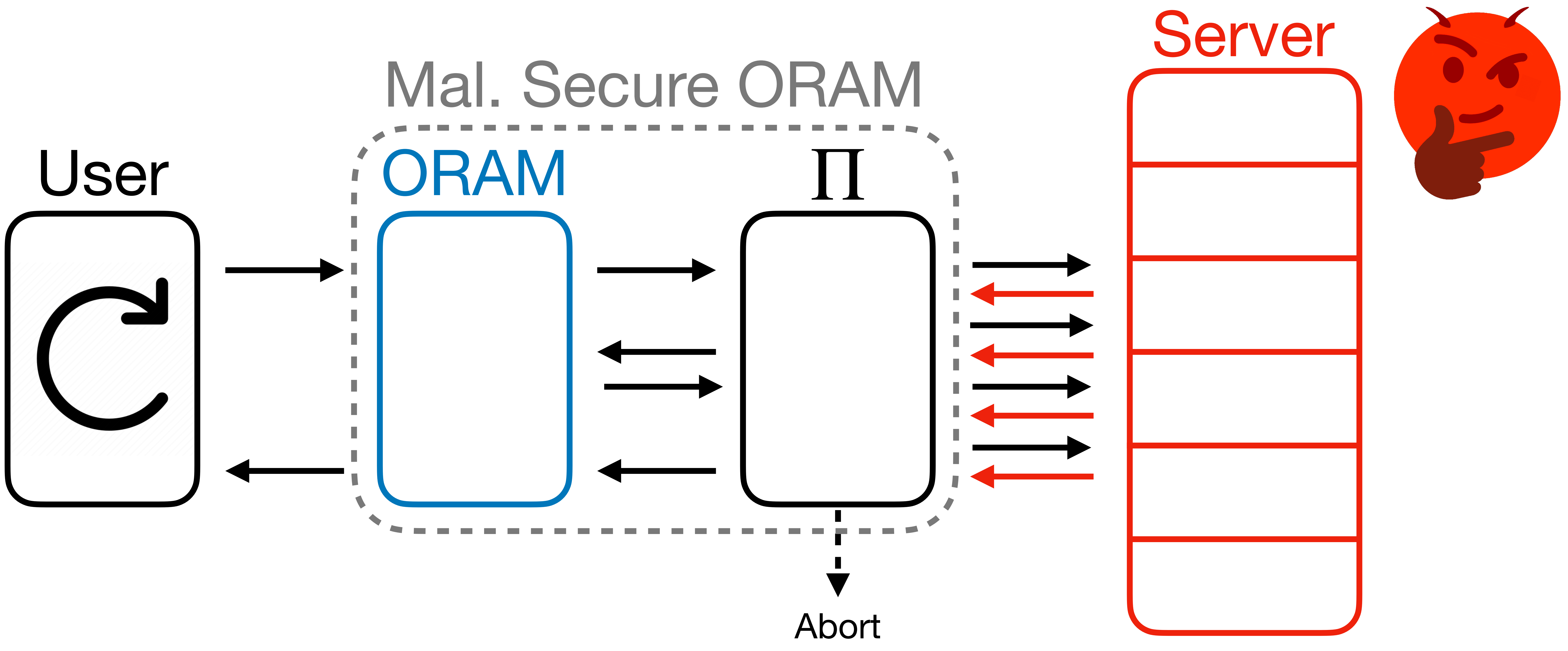
Technique #1: Memory Checking

- Great! But this isn't efficient enough.

$$\underset{\overset{\uparrow}{\log^2(N)}}{\text{Overhead}(\text{ORAM}_{\text{Mal}})} = \underset{\overset{\uparrow}{\log N}}{\text{Overhead}(\text{ORAM}_{\text{HBC}})} \cdot \underset{\overset{\uparrow}{\log N}}{\text{Overhead}(\text{MC})}$$

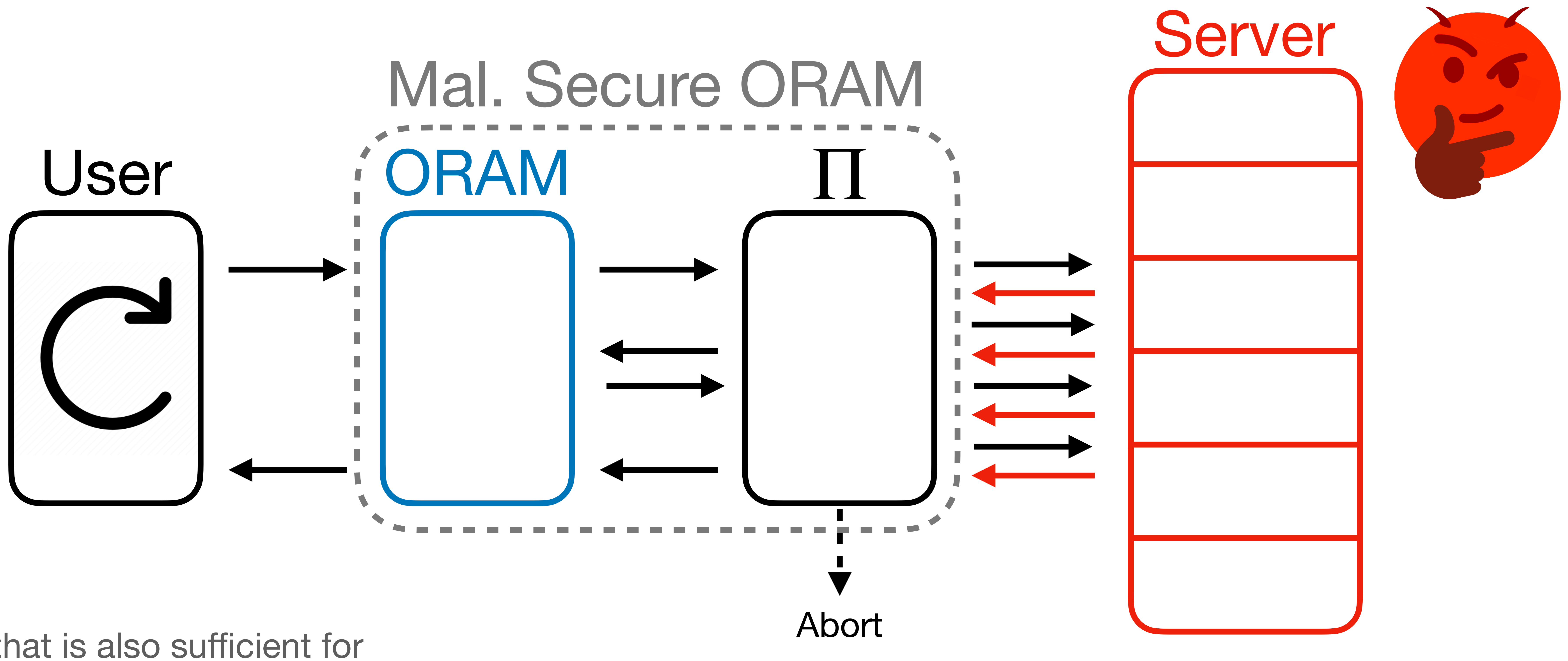
- Do we really need a memory checker? Does a weaker compiler suffice?

Technique #1: Memory Checking



Technique #1: Memory Checking

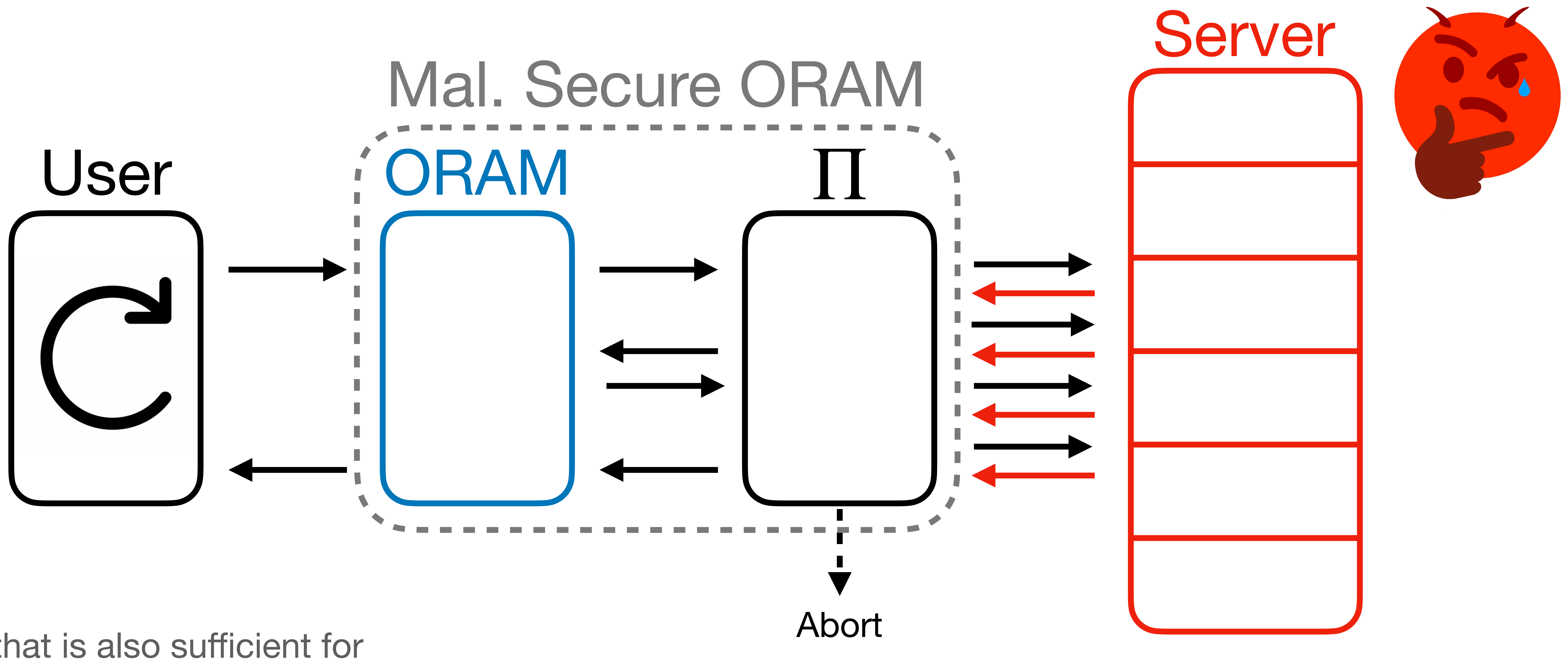
<u>Theorem</u> <u>[M.-Vafa '23]</u>: If Π compiles **any** honest-but-curious ORAM into a maliciously secure ORAM with overhead blowup ℓ in this way, then Π is a memory checker* with overhead ℓ.



*Slight weakening that is also sufficient for converting honest-but-curious to malicious ORAM.

Technique #1: Memory Checking

<u>Theorem</u> <u>[M.-Vafa '23]</u>: If Π compiles **any** honest-but-curious ORAM into a maliciously secure ORAM with overhead blowup ℓ in this way, then Π is a memory checker* with overhead ℓ.



*Slight weakening that is also sufficient for converting honest-but-curious to malicious ORAM.

Summary

Summary

Memory Checking (MC)

Summary

Memory Checking (MC)

- $O(1)$-blowup post-compiler is **equivalent** to an $O(1)$-overhead memory checker.

Summary

Memory Checking (MC)

- $O(1)$-blowup post-compiler is **equivalent** to an $O(1)$-overhead memory checker.
- Best memory checkers have $O(\log N)$ overhead, so seems unlikely.

Summary

Memory Checking (MC)

- $O(1)$-blowup post-compiler is **equivalent** to an $O(1)$-overhead memory checker.
- Best memory checkers have $O(\log N)$ overhead, so seems unlikely.

How can we proceed?

Summary

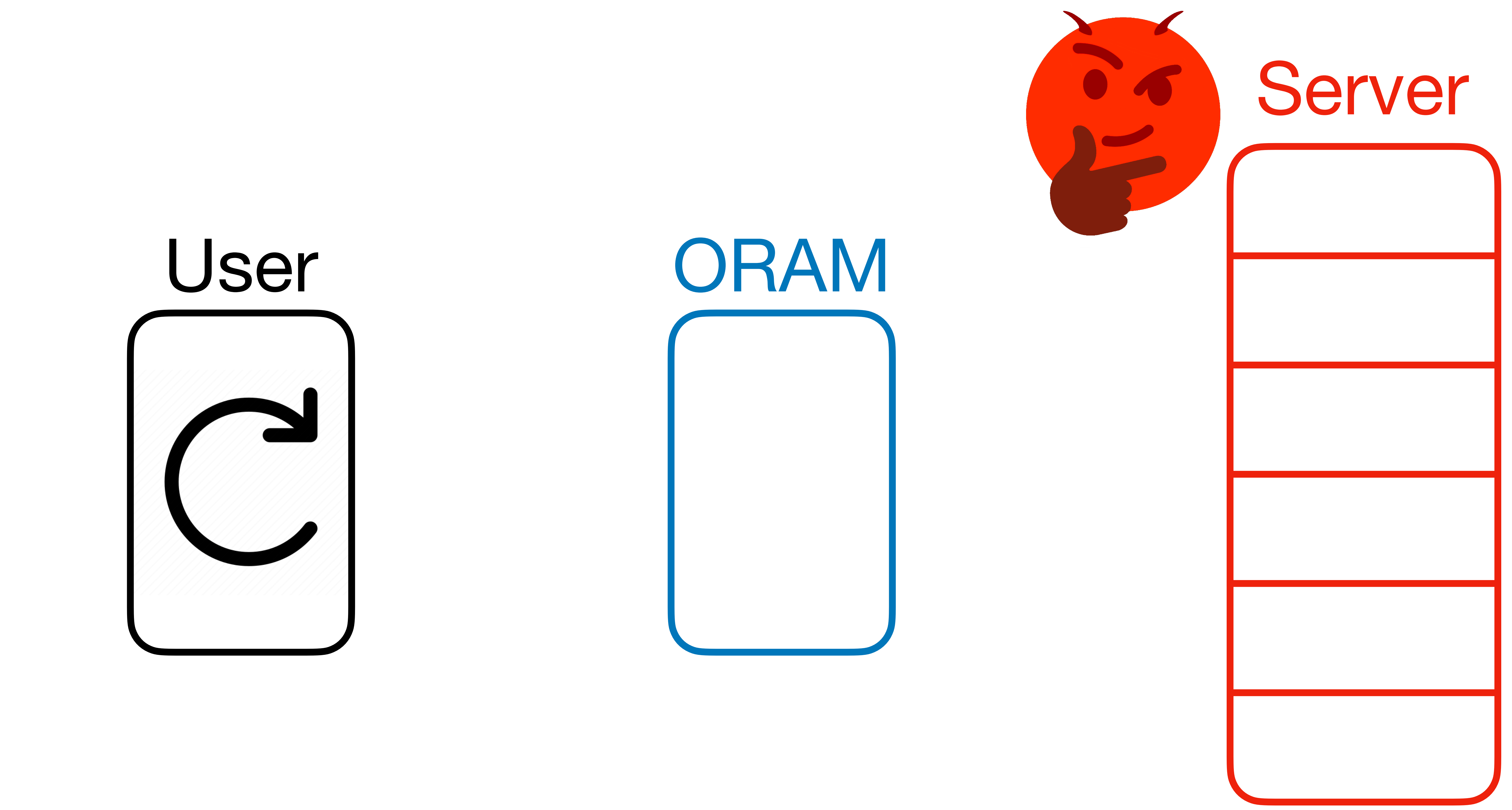
Memory Checking (MC)

- $O(1)$-blowup post-compiler is **equivalent** to an $O(1)$-overhead memory checker.
- Best memory checkers have $O(\log N)$ overhead, so seems unlikely.

How can we proceed?

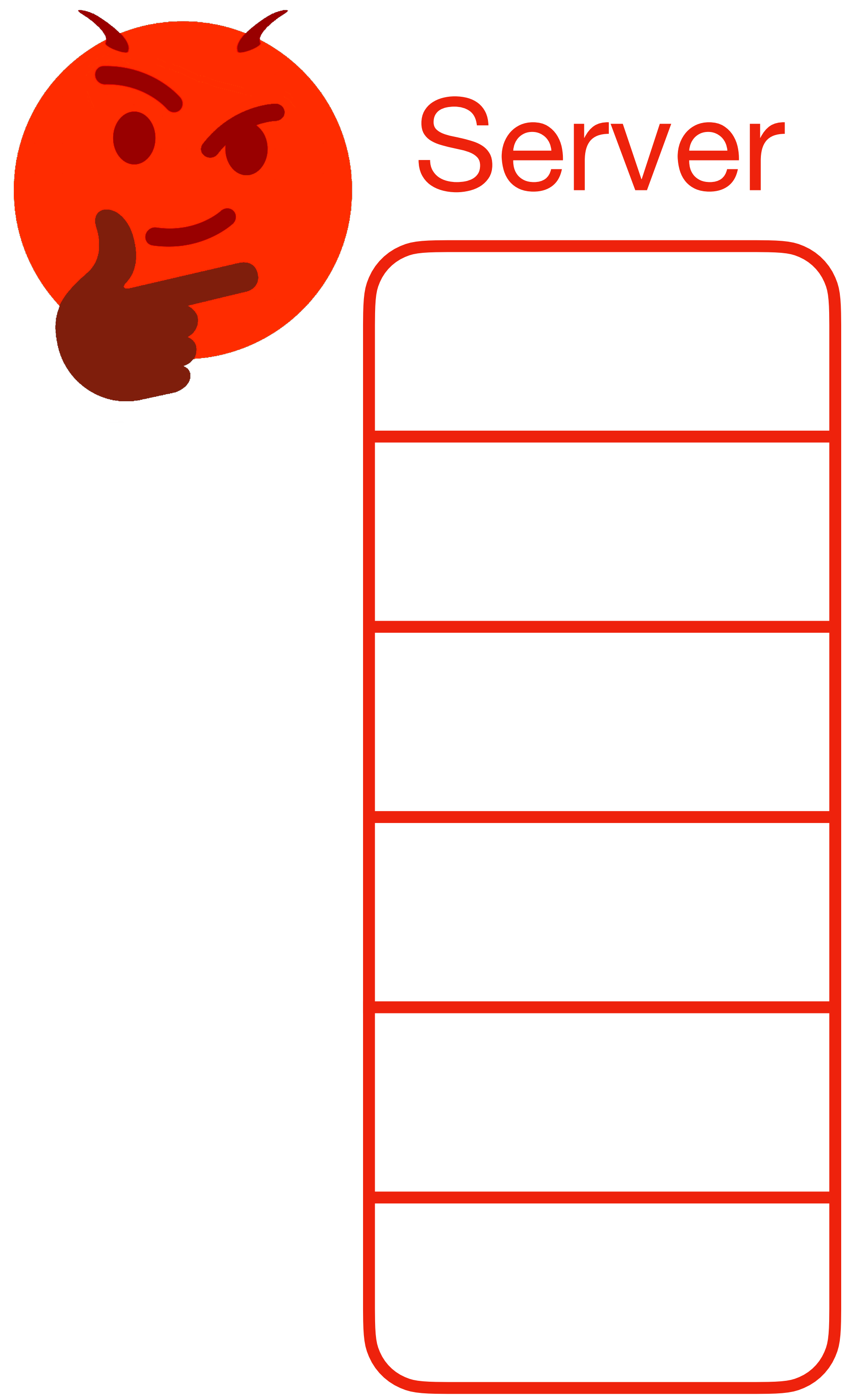
We have to handle OptORAMa in a white-box way!

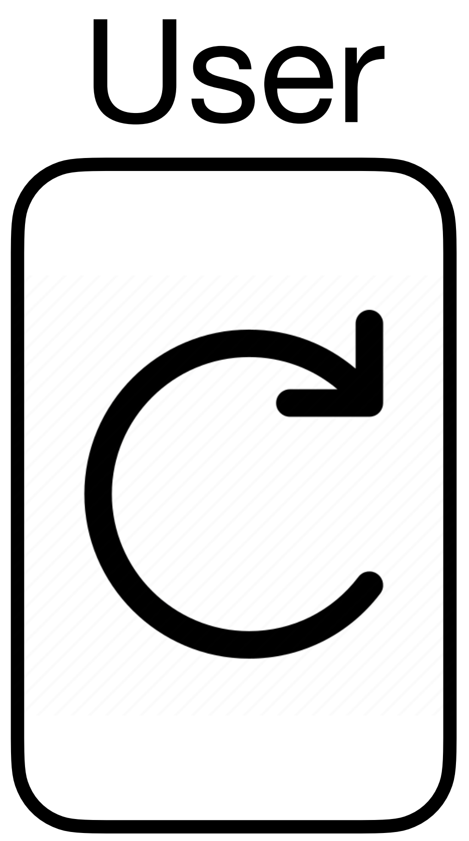
Does OptORAMa really need memory checking?

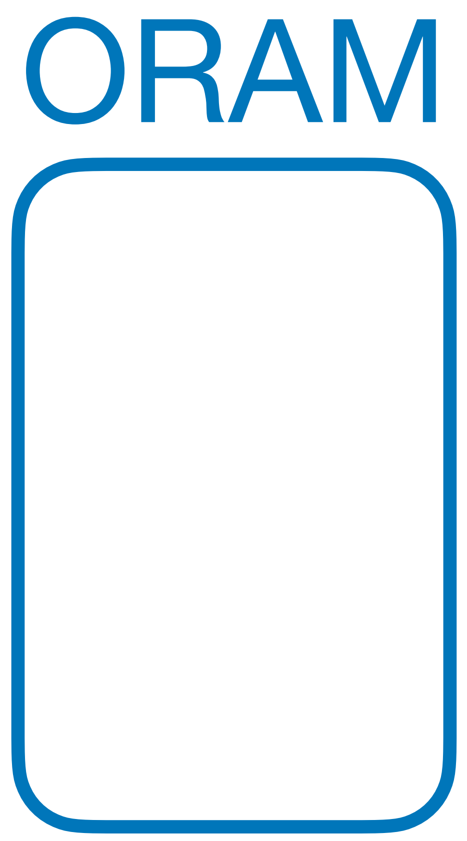


Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate ***some*** lies from the server?

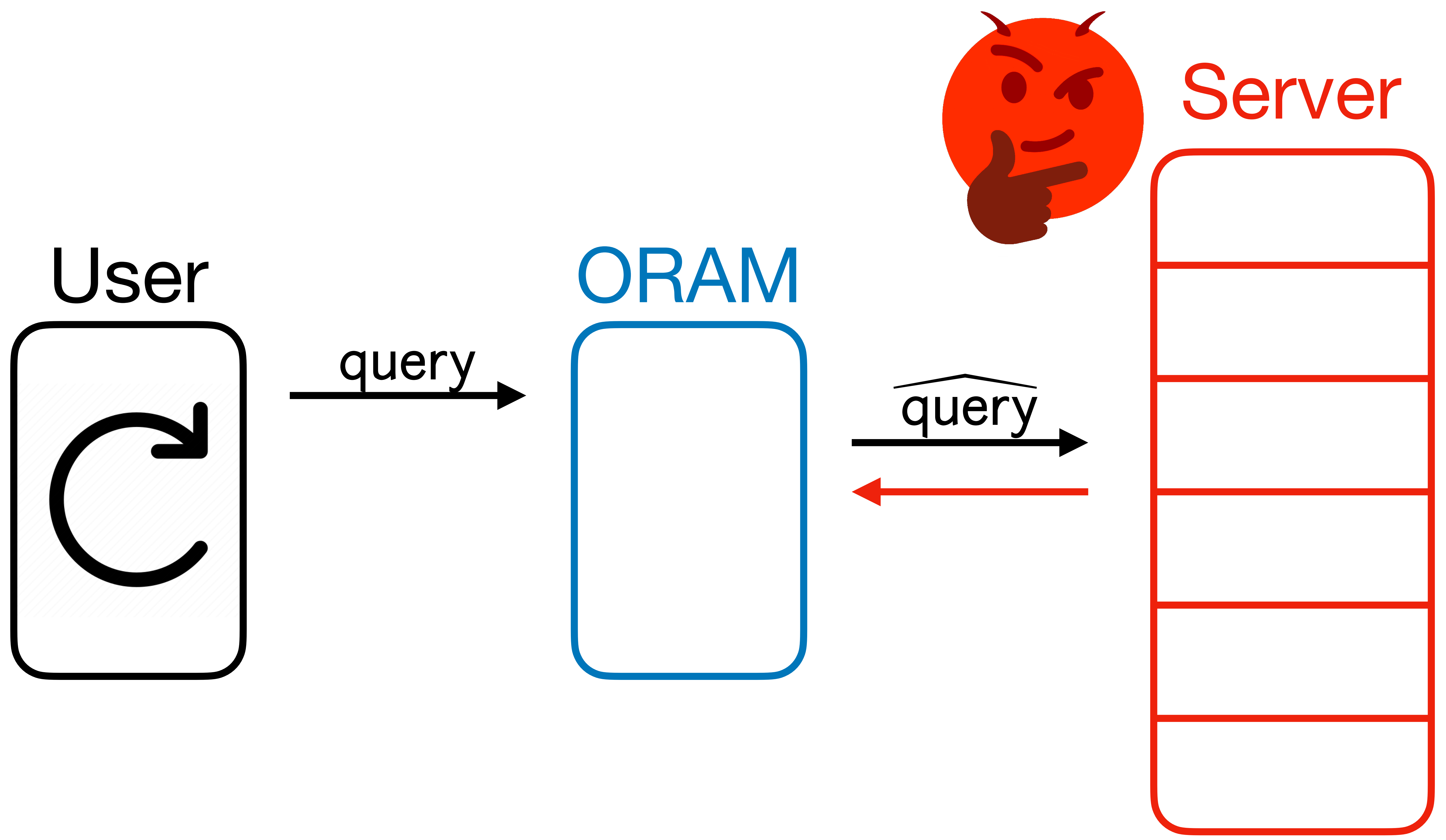






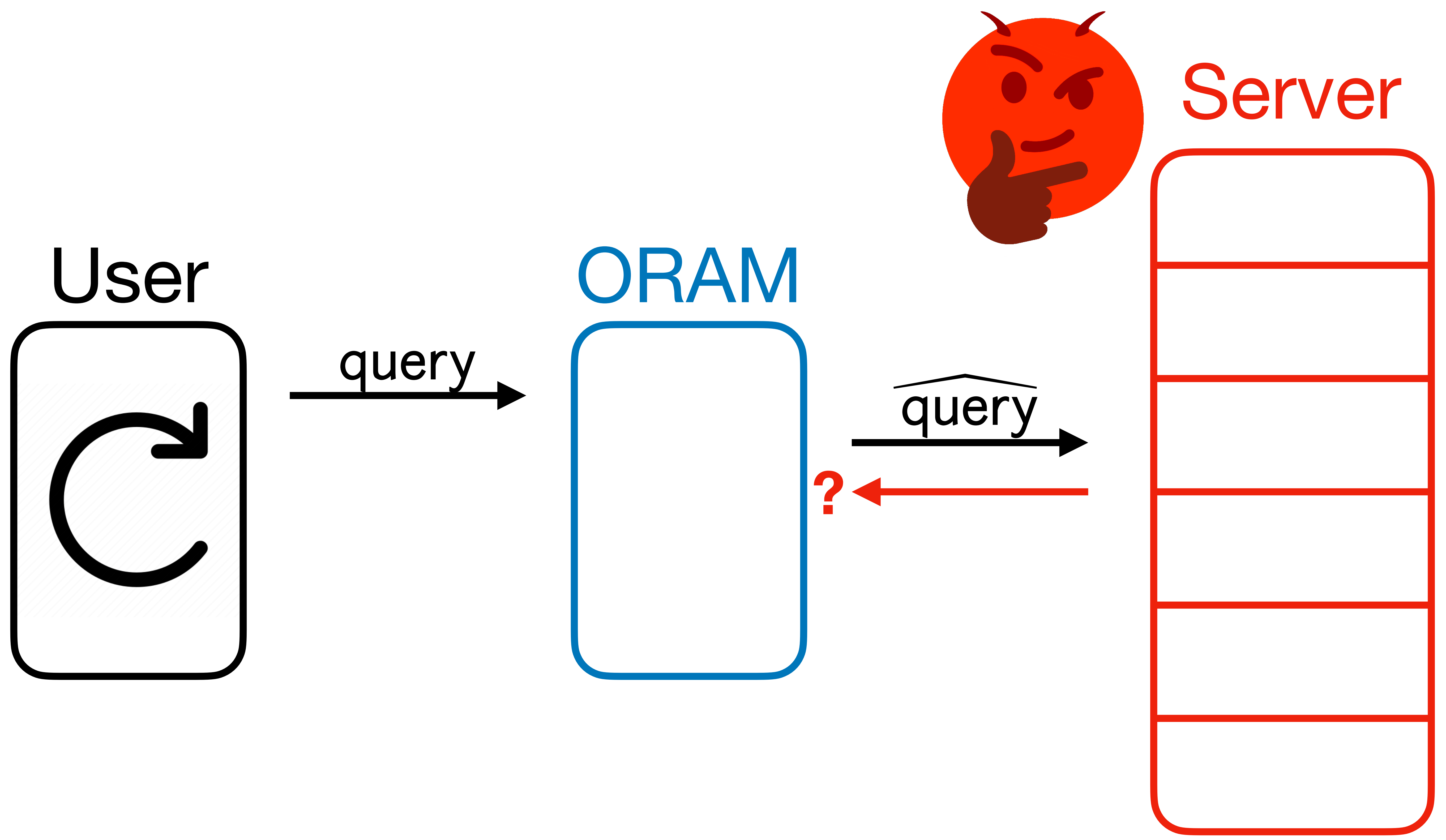
Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate ***some*** lies from the server?



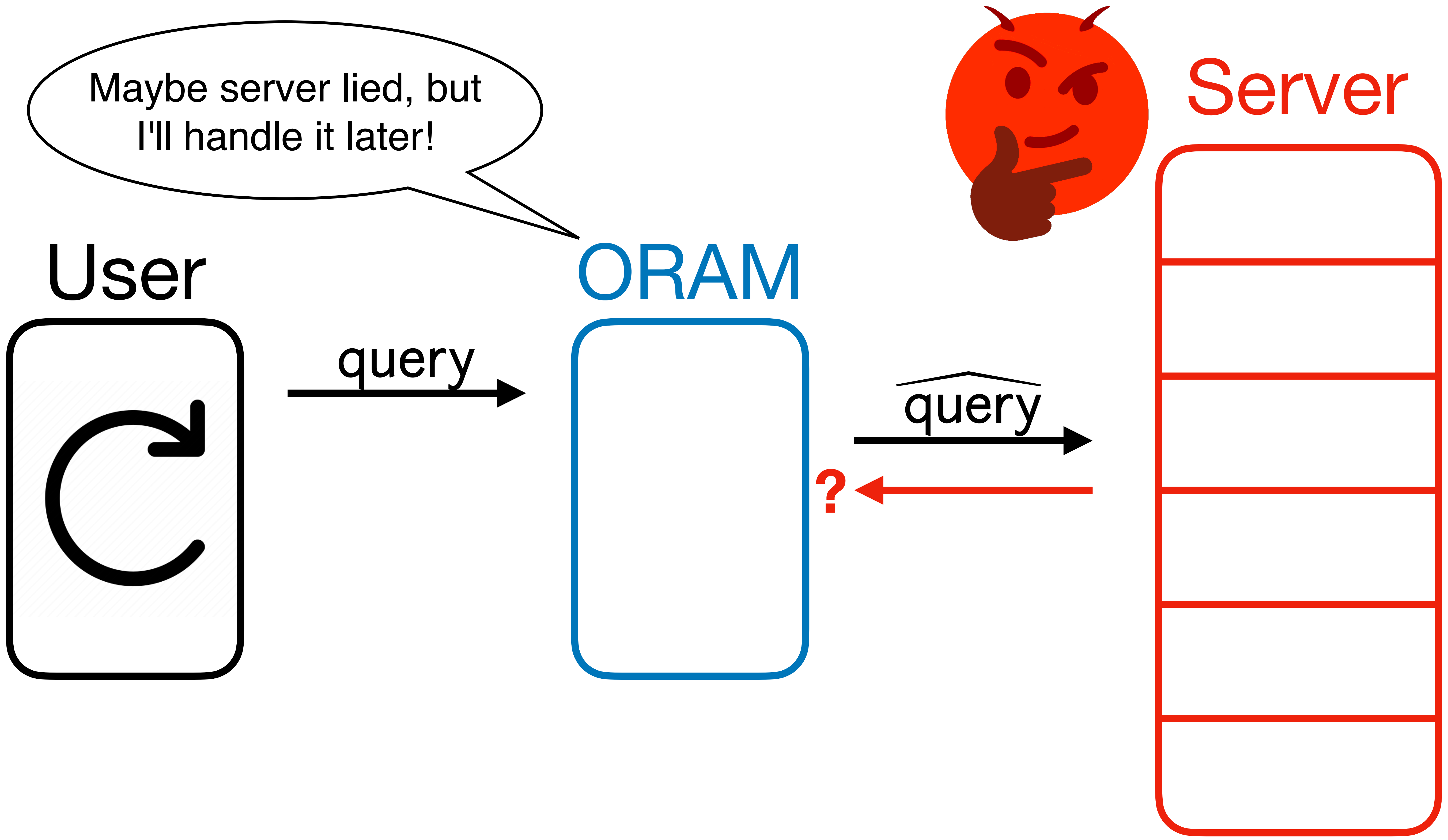
Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate ***some*** lies from the server?



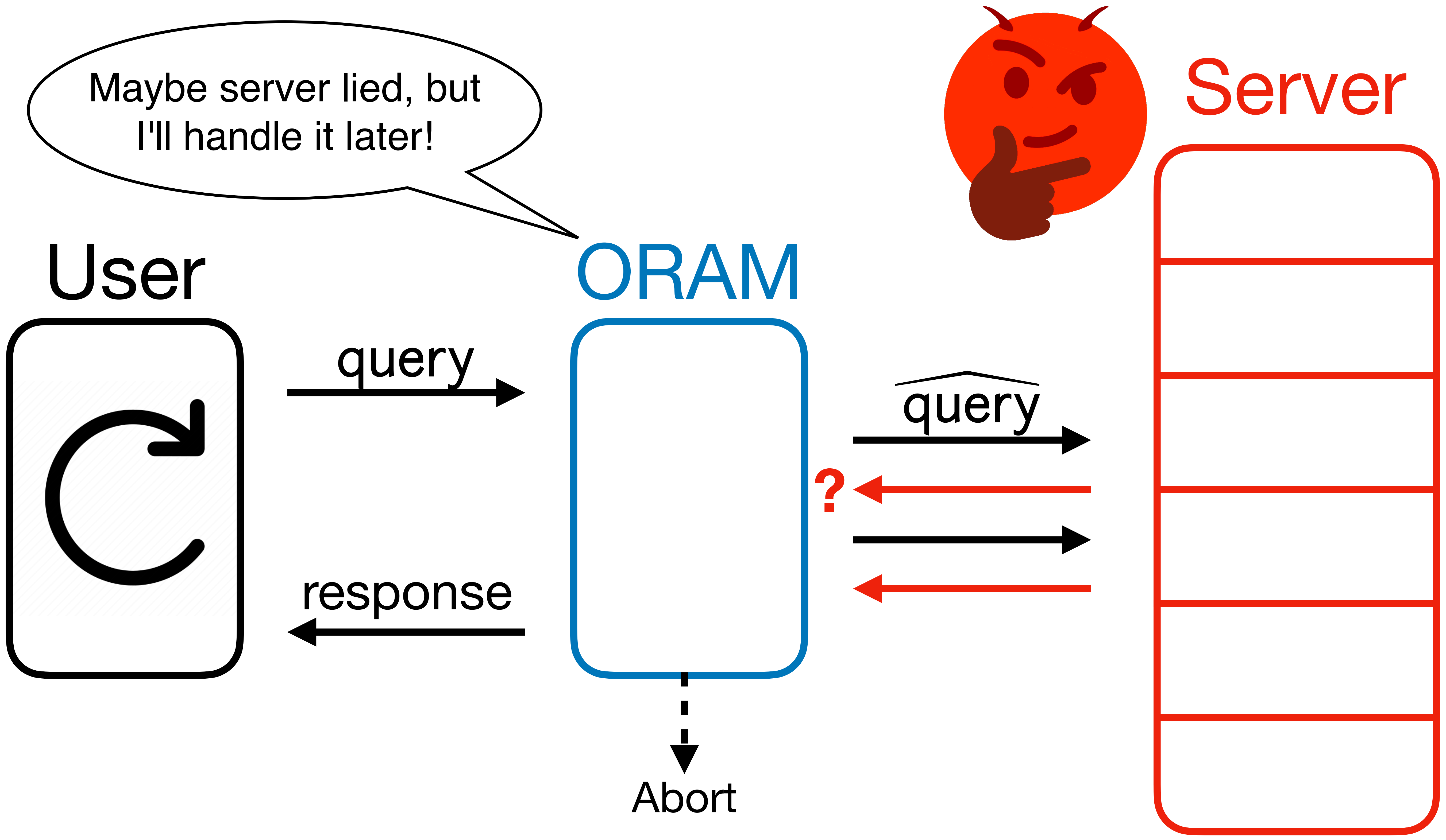
Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate ***some*** lies from the server?



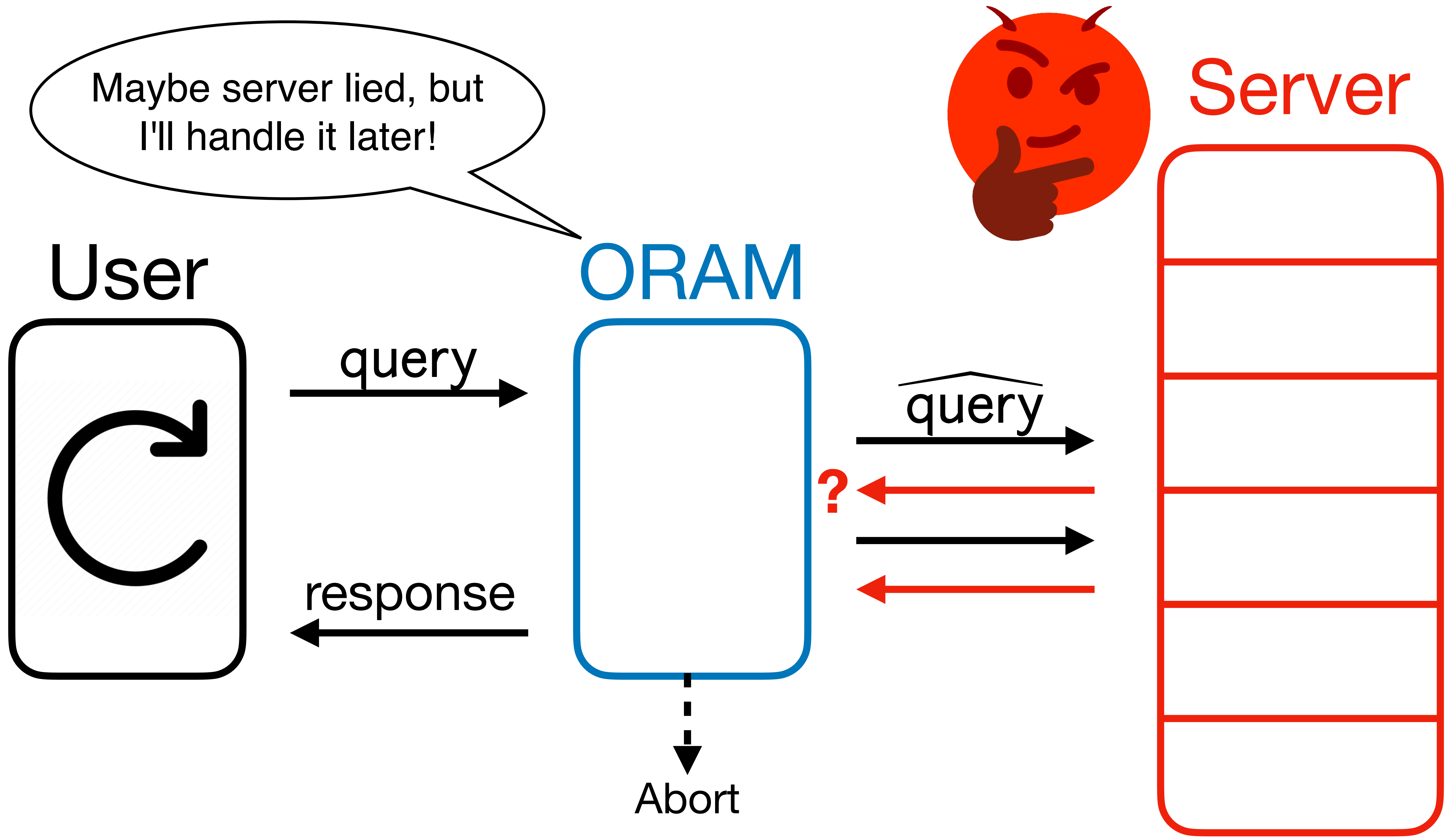
Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate ***some*** lies from the server?



Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate ***some*** lies from the server?
- **Our Idea**: Use weaker, more efficient **"batched"** notion of memory checking to capitalize on this!



Offline Memory Checking

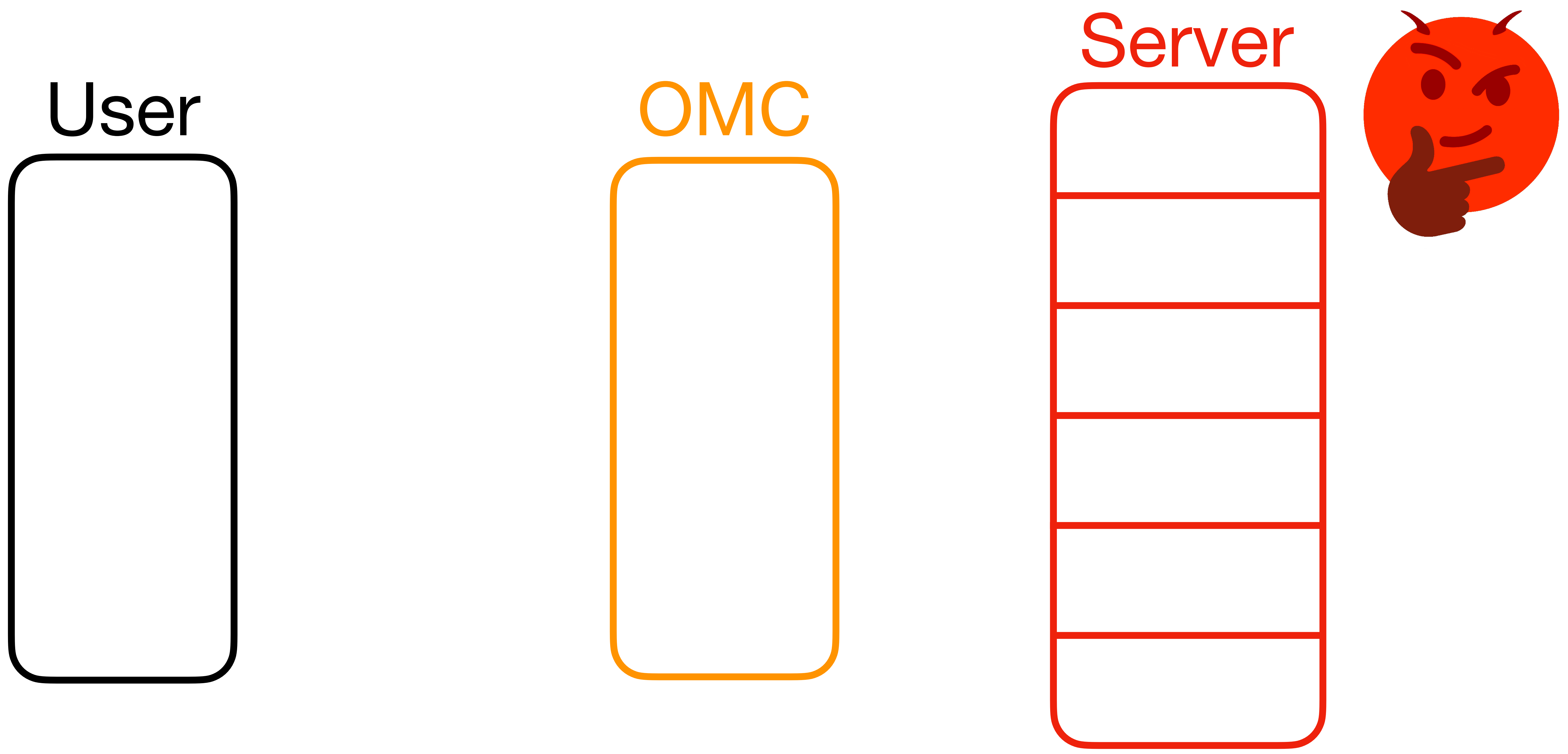
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition:

[Blum et al. '94]

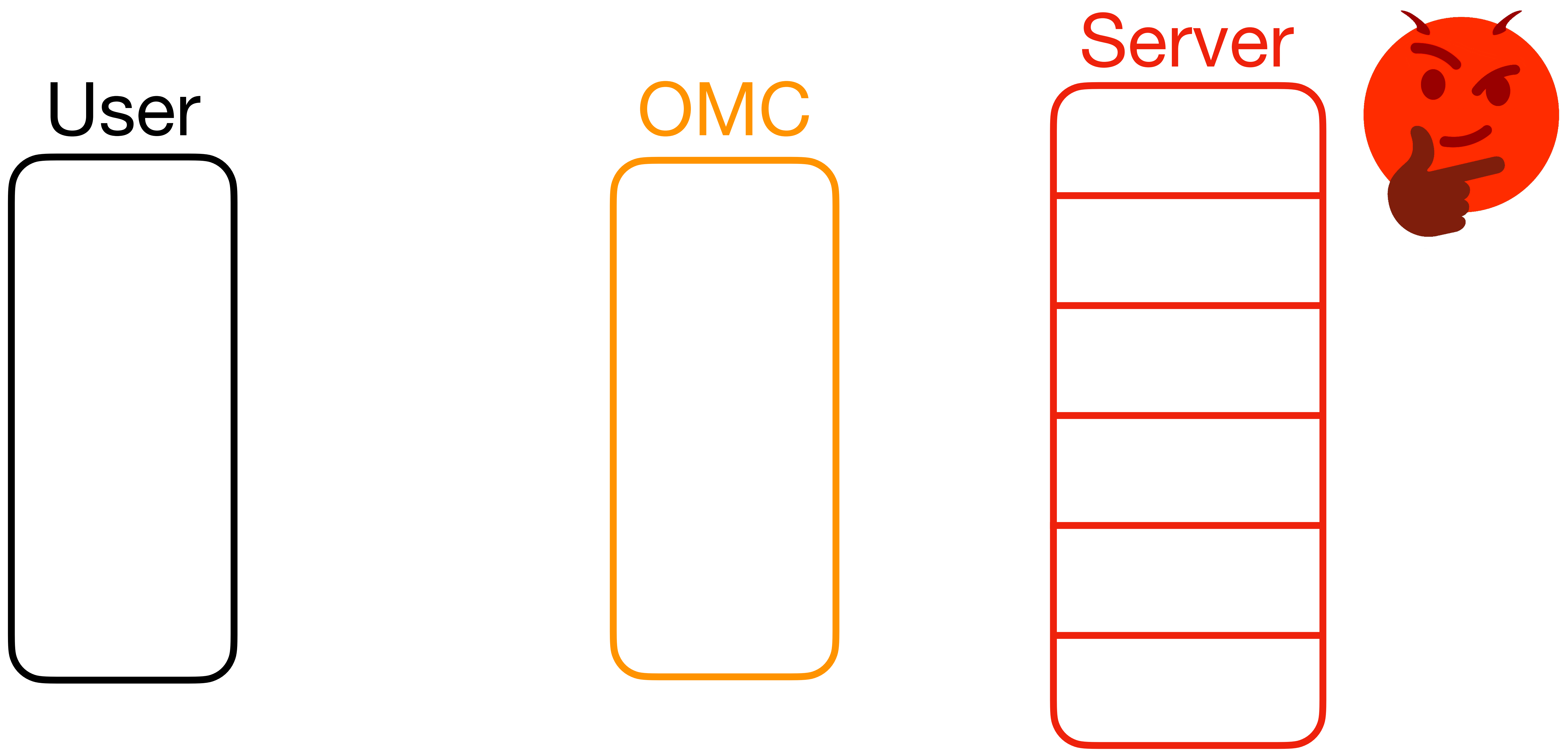
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]



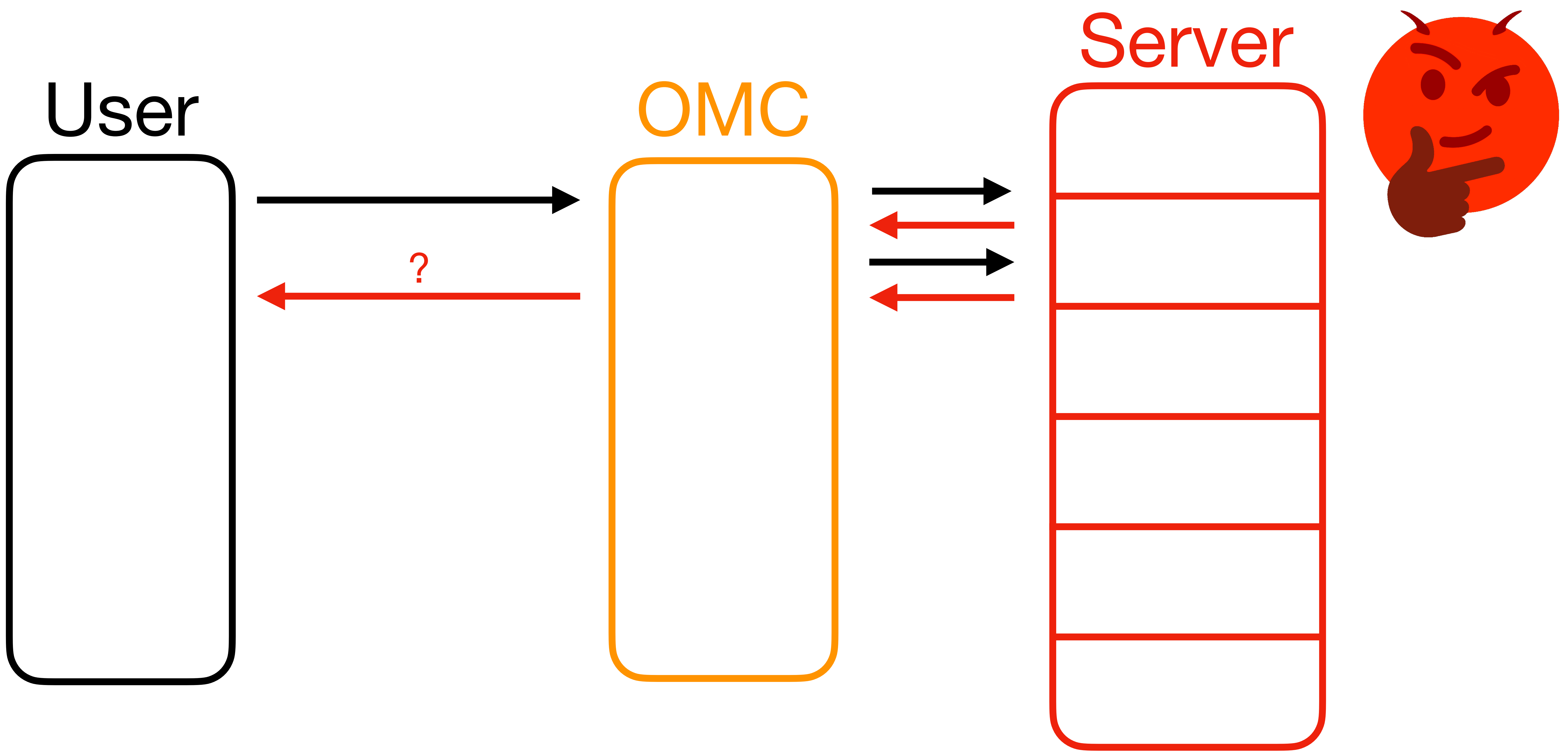
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think “batching” a regular memory checker.)



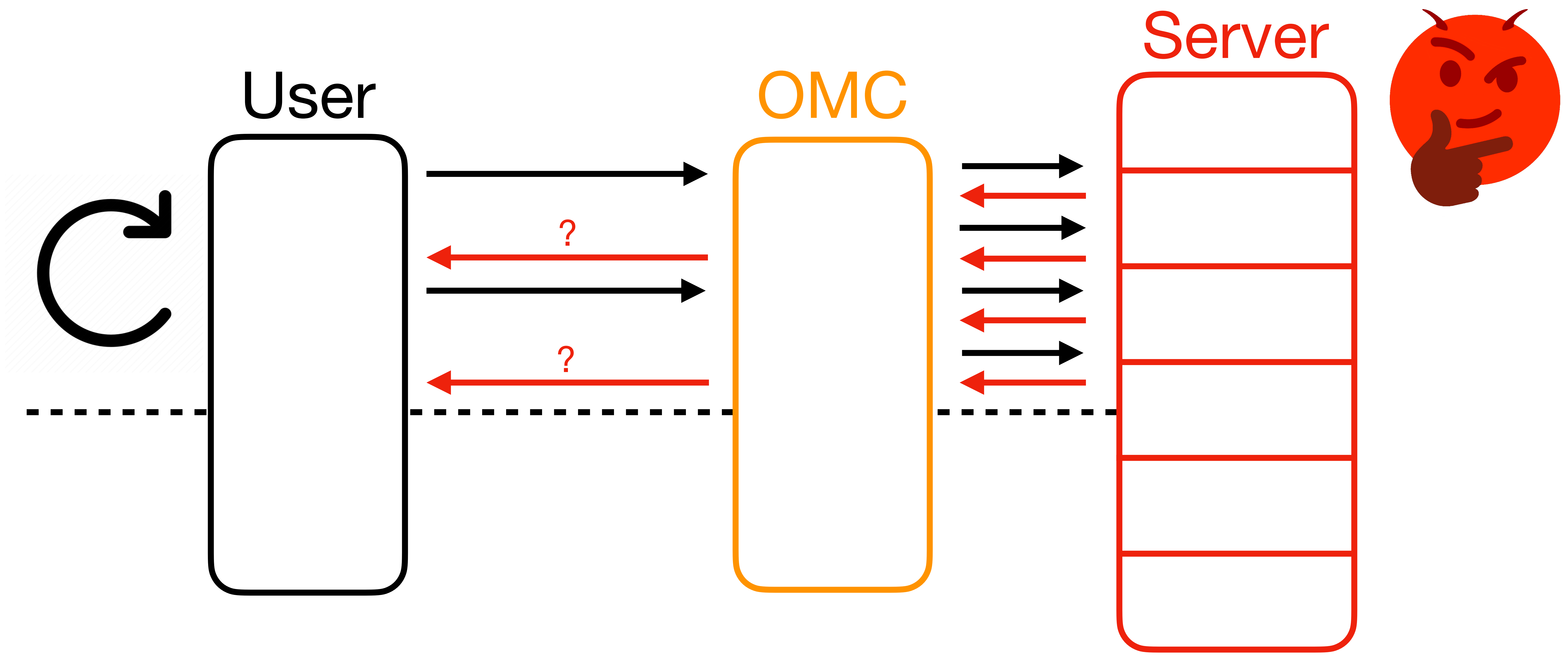
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



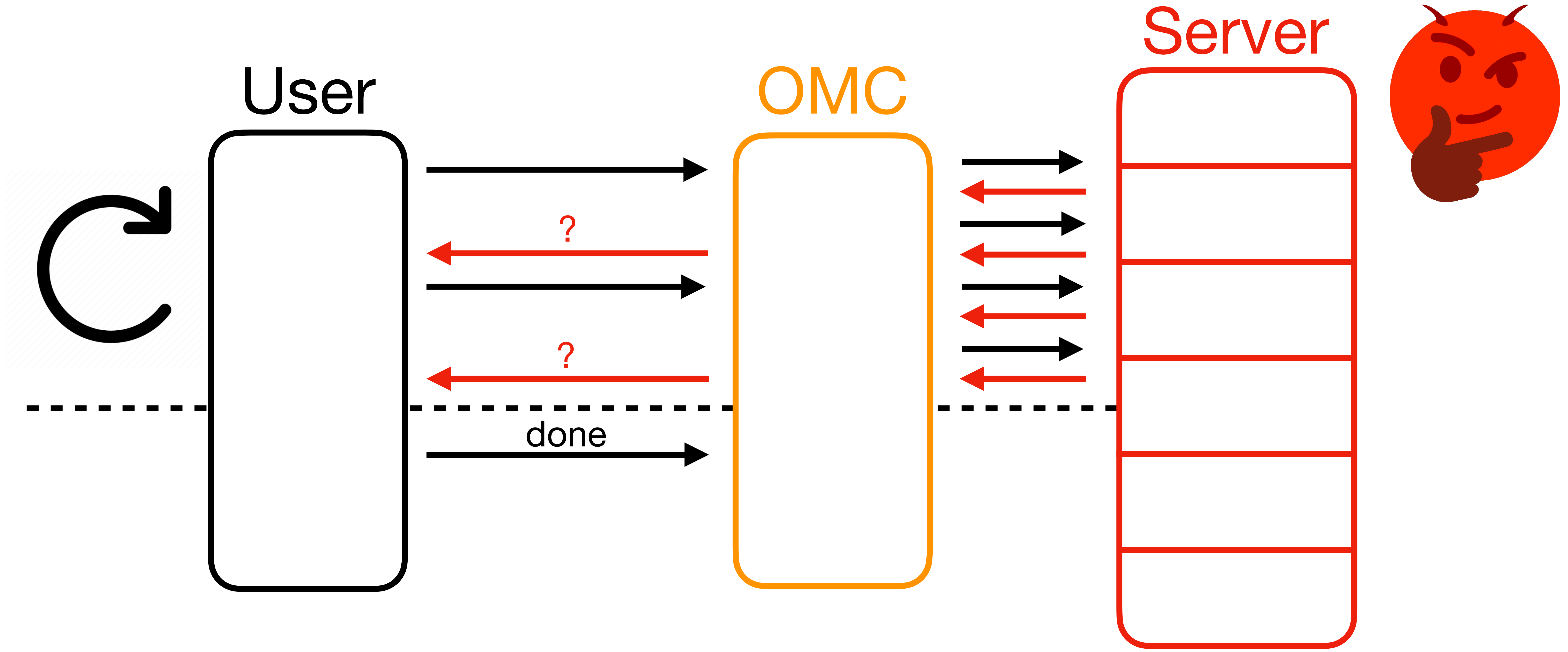
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



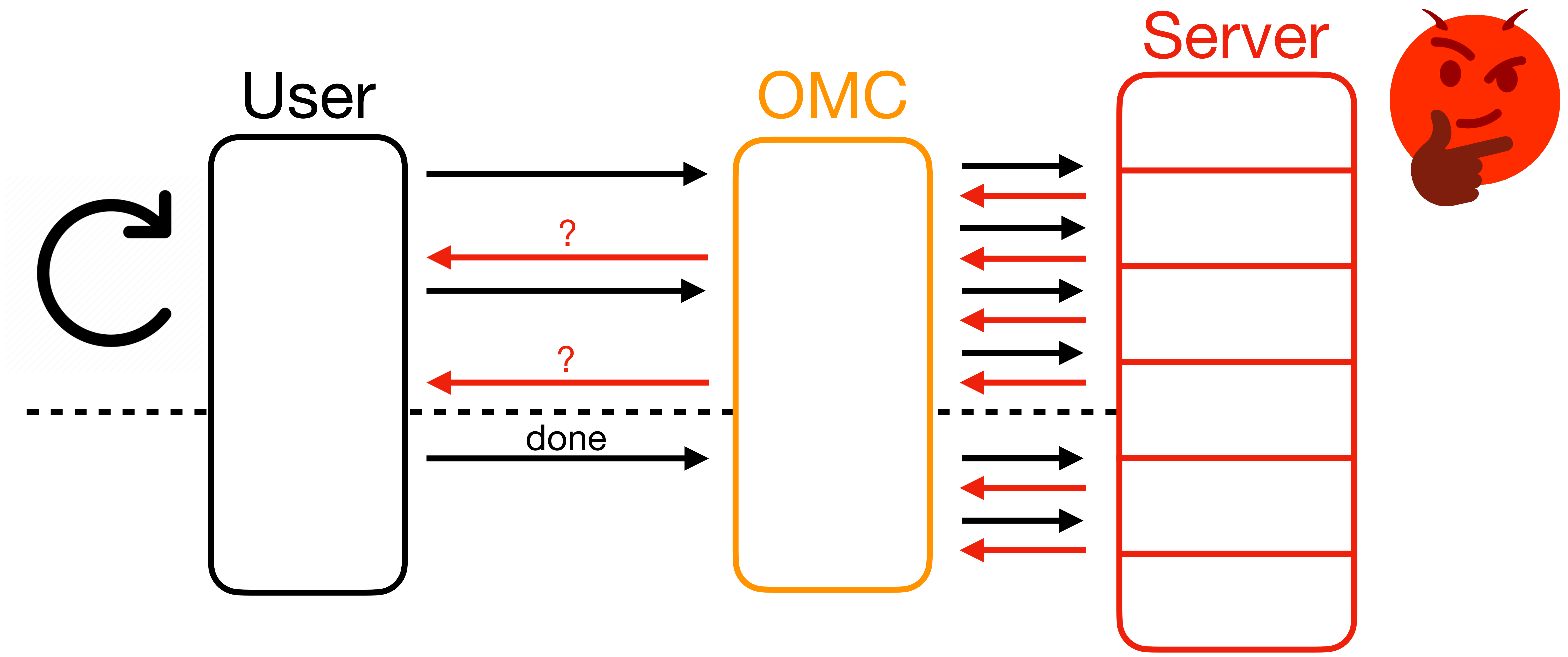
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



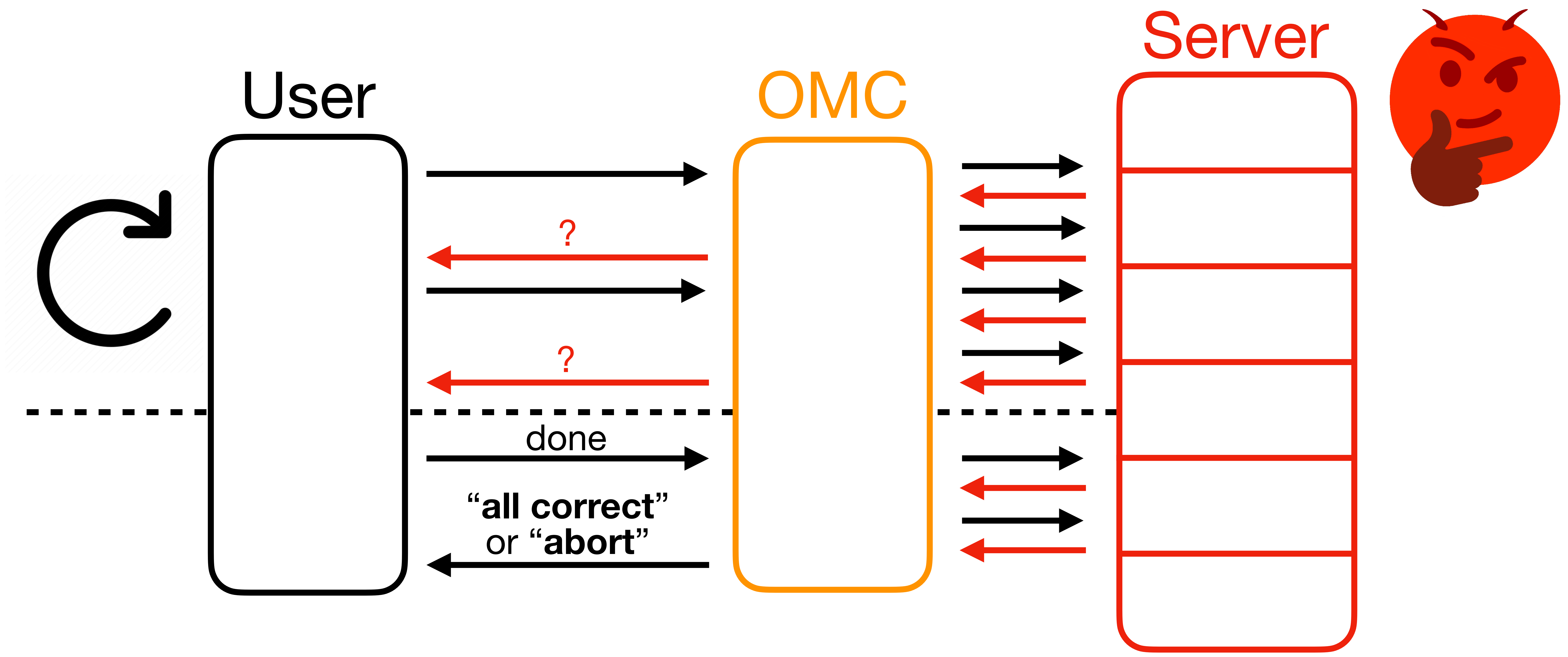
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think “batching” a regular memory checker.)



Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**

[Blum et al. '94]
[Dwork et al. '09]

Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**
    - Many subroutines in OptORAMa can be offline-checked.
- Con of offline memory checking: **insufficient!** Does not work for **all** subroutines.

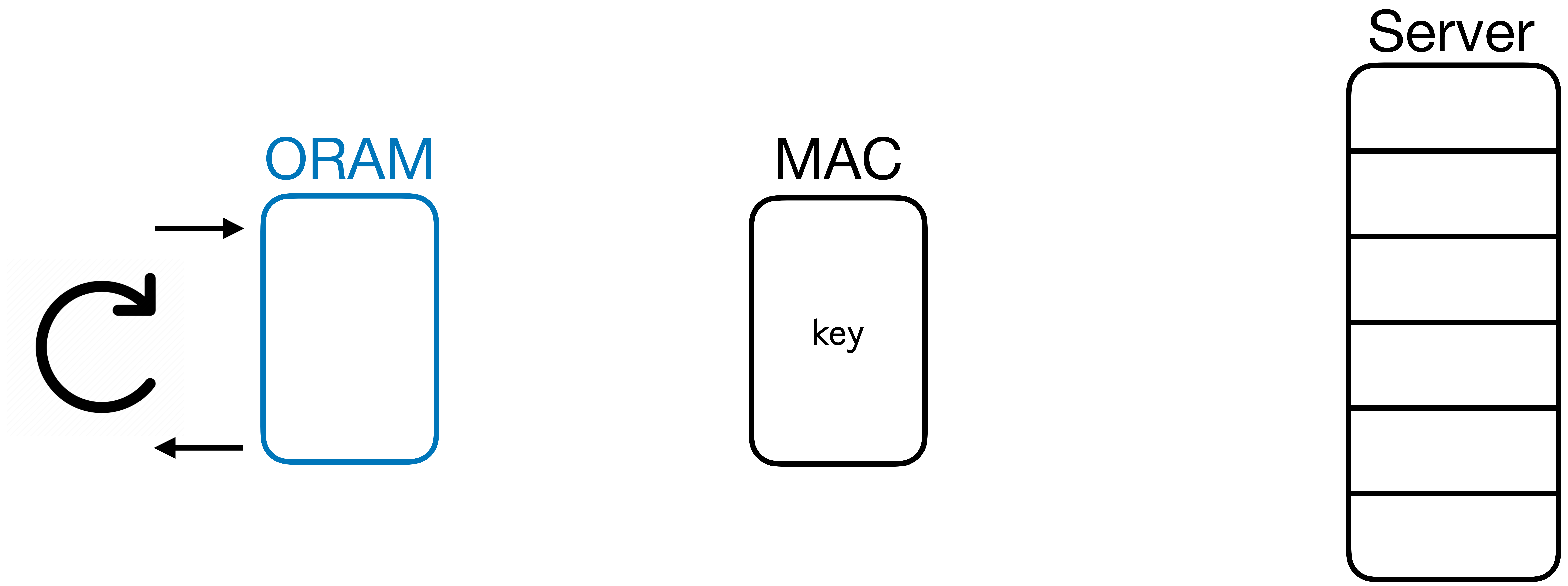
Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**
  - Many subroutines in OptORAMa can be offline-checked.
- Con of offline memory checking: **insufficient!** Does not work for **all** subroutines.

We need another technique!

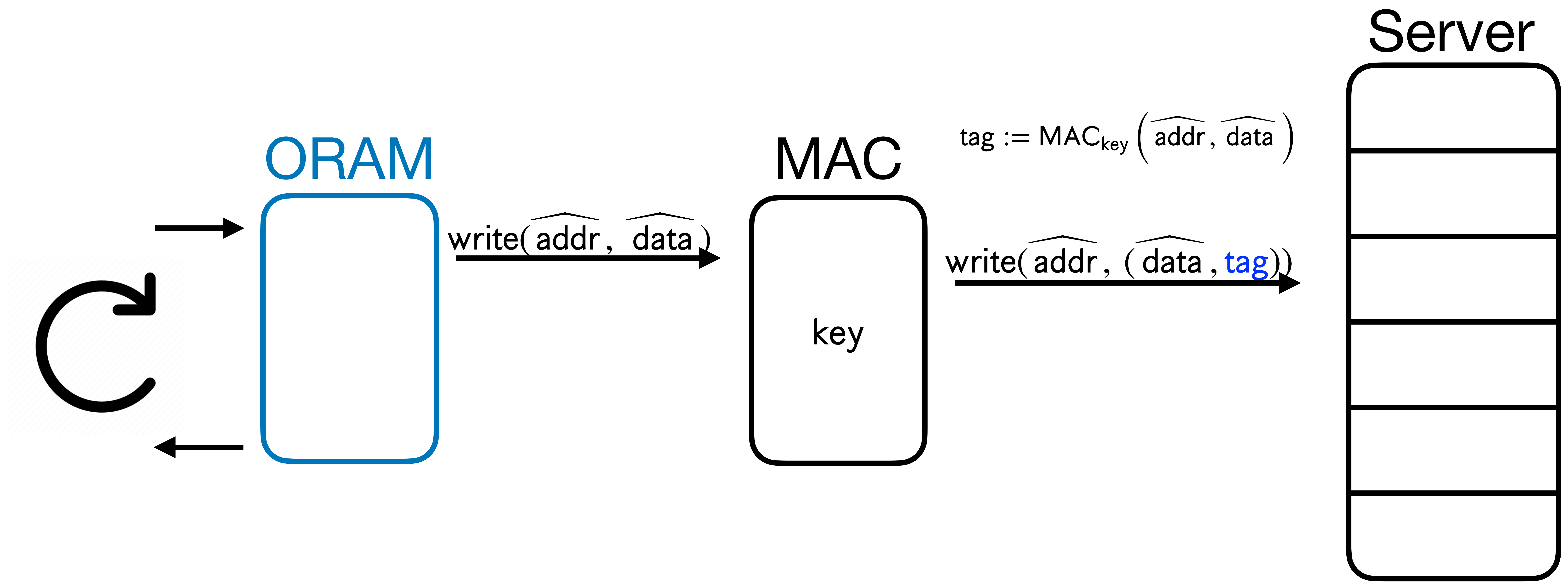
Technique #2: MACs

- What about Message Authentication Codes (MACs)?



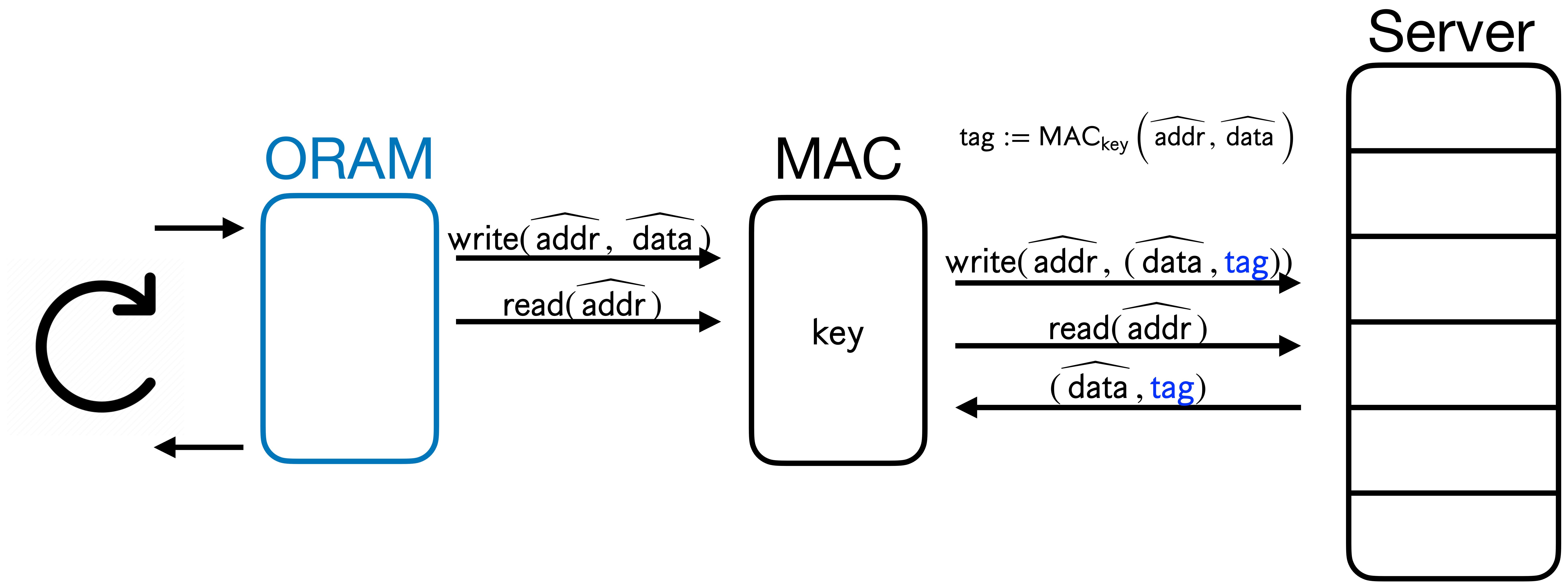
Technique #2: MACs

- What about Message Authentication Codes (MACs)?


ORAM
$\mathrm{write}(\widehat{\mathrm{addr}}, \widehat{\mathrm{data}})$
MAC
key
$\mathrm{tag} := \mathrm{MAC}_{\mathrm{key}}\left(\widehat{\mathrm{addr}}, \widehat{\mathrm{data}}\right)$
$\mathrm{write}(\widehat{\mathrm{addr}}, (\widehat{\mathrm{data}}, \mathrm{tag}))$
Server

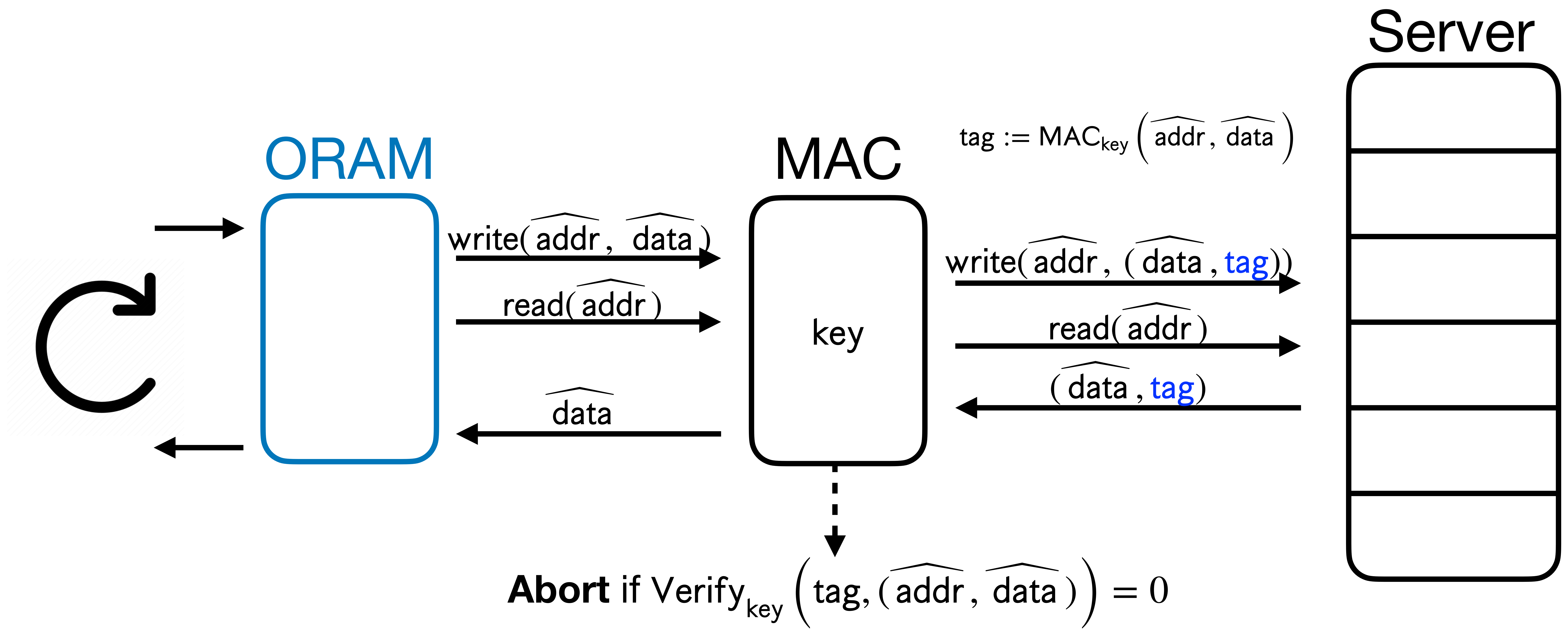
Technique #2: MACs

- What about Message Authentication Codes (MACs)?


ORAM
MAC
Server
key
$\widehat{\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})}$
$\text{read}(\widehat{\text{addr}})$
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}\right)$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{tag})$

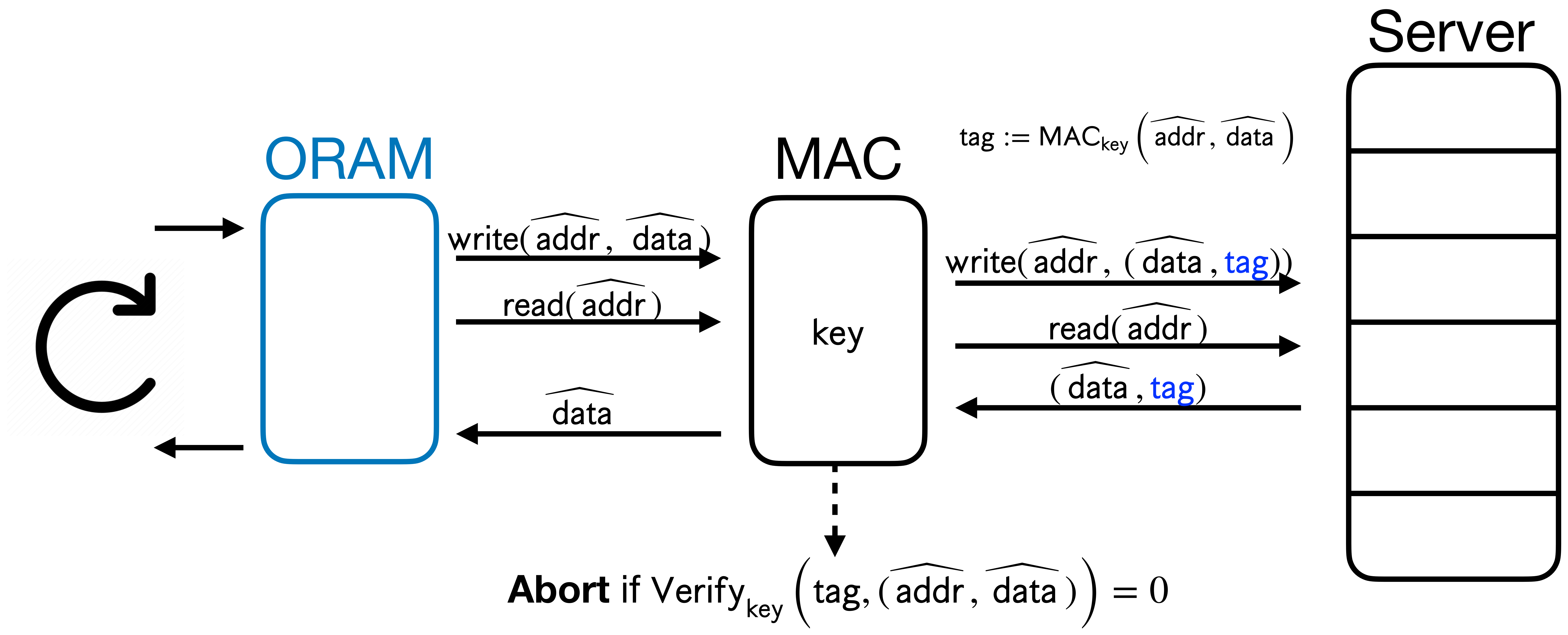
Technique #2: MACs

- What about Message Authentication Codes (MACs)?


ORAM
MAC
Server
key
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{tag})$
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}})\right) = 0$

Technique #2: MACs

- What about Message Authentication Codes (MACs)?
- MACs force the server to only send back values it has already seen.

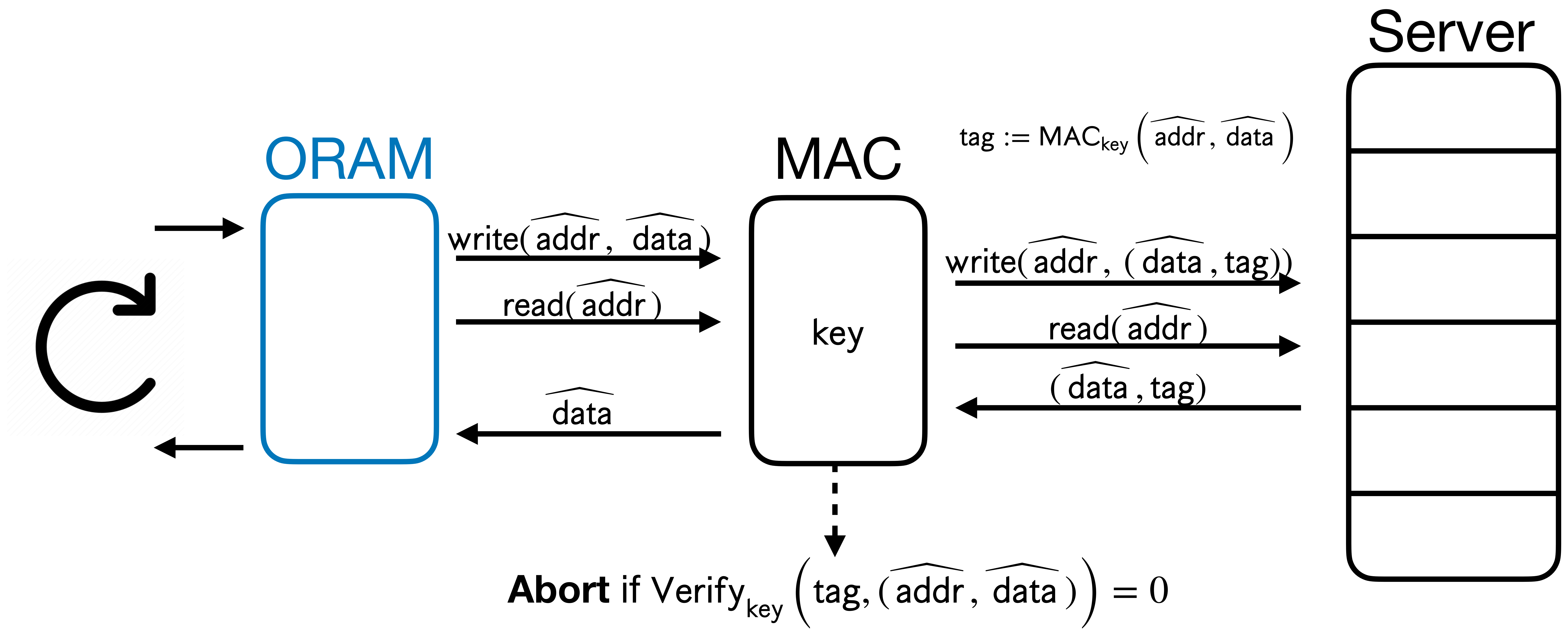


Server
ORAM
MAC
key
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{tag})$

Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}})\right) = 0$

Technique #2: MACs

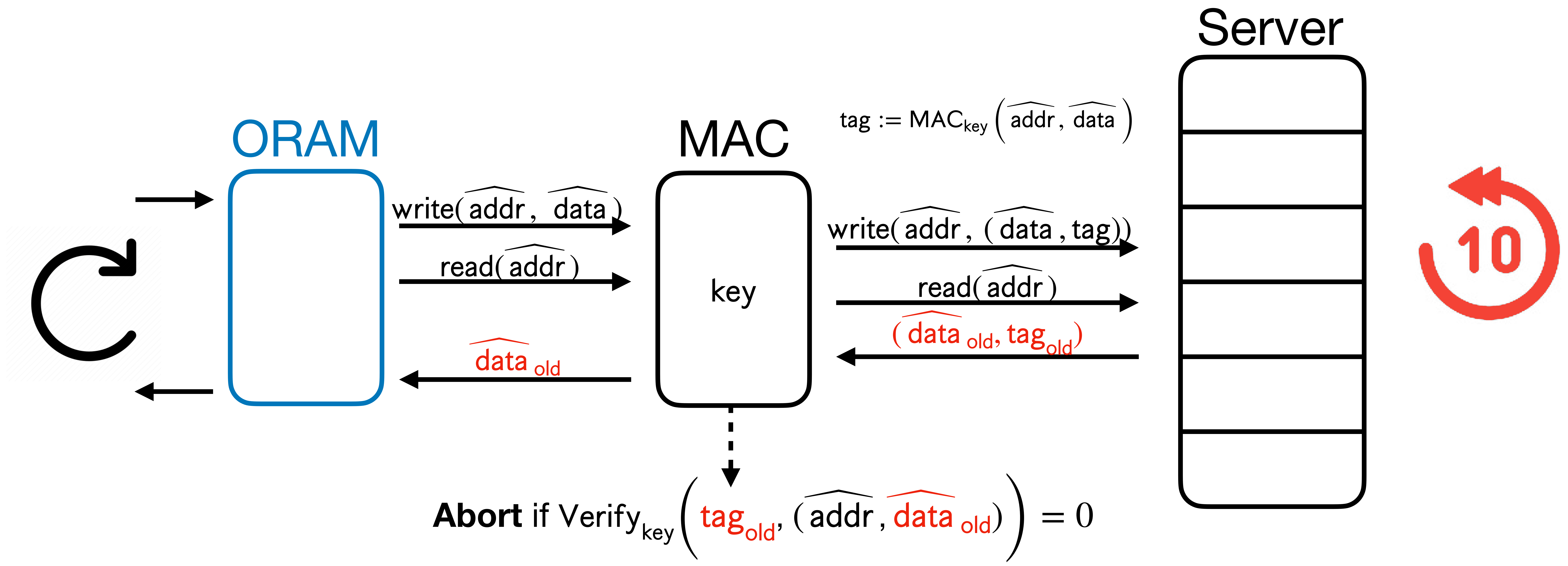
- MACs are **insufficient** because the server can do *replay attacks*.



ORAM
MAC
Server
key
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}\right)$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{tag})$
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}})\right) = 0$

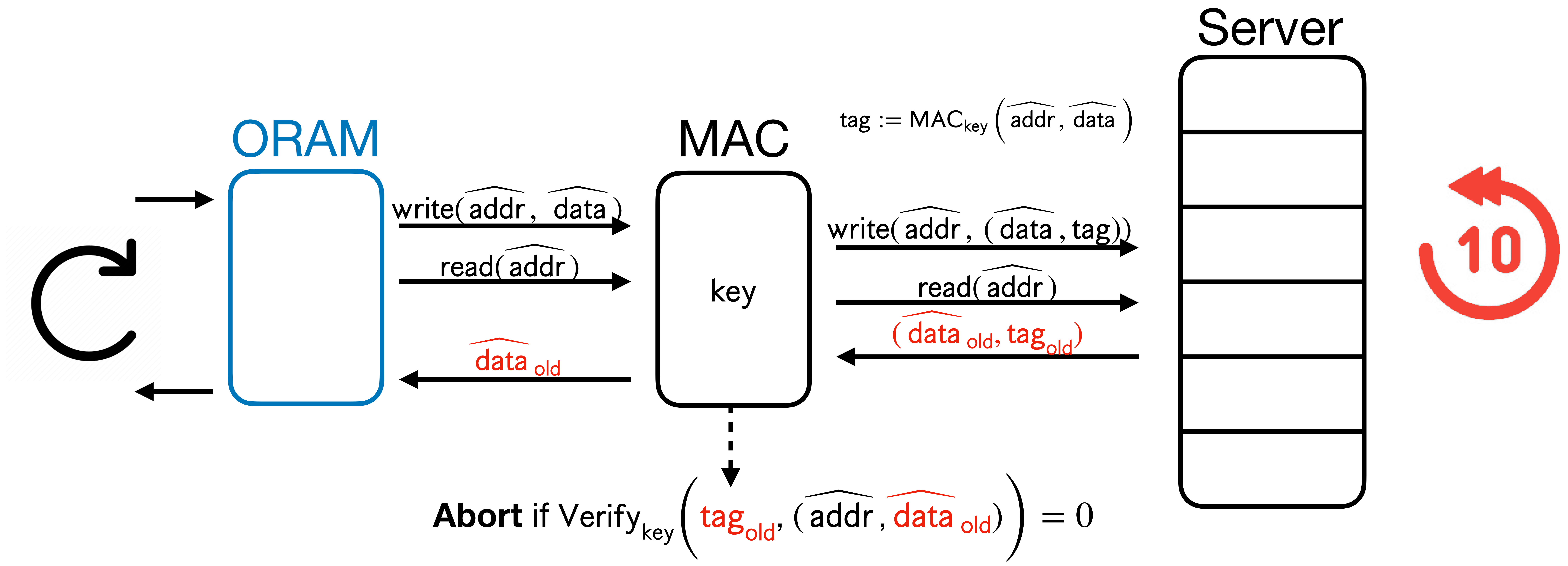
Technique #2: MACs

- MACs are **insufficient** because the server can do *replay attacks*.


ORAM
MAC
Server
key
$\mathrm{write}(\widehat{\mathrm{addr}}, \widehat{\mathrm{data}})$
$\mathrm{read}(\widehat{\mathrm{addr}})$
$\widehat{\mathrm{data}}_{\mathrm{old}}$
$\mathrm{tag} := \mathrm{MAC}_{\mathrm{key}}(\widehat{\mathrm{addr}}, \widehat{\mathrm{data}})$
$\mathrm{write}(\widehat{\mathrm{addr}}, (\widehat{\mathrm{data}}, \mathrm{tag}))$
$\mathrm{read}(\widehat{\mathrm{addr}})$
$(\widehat{\mathrm{data}}_{\mathrm{old}}, \mathrm{tag}_{\mathrm{old}})$
Abort if $\mathrm{Verify}_{\mathrm{key}}\left(\mathrm{tag}_{\mathrm{old}}, (\widehat{\mathrm{addr}}, \widehat{\mathrm{data}}_{\mathrm{old}})\right) = 0$
10

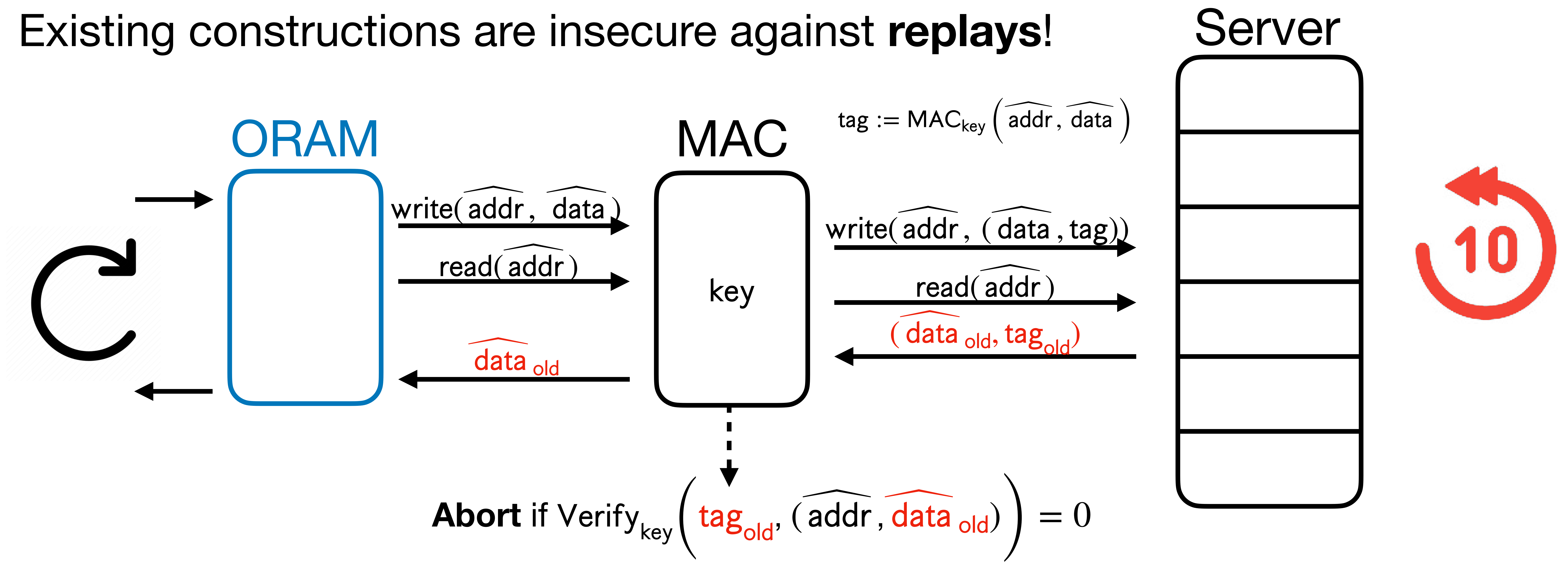
Technique #2: MACs

- MACs are **insufficient** because the server can do *replay attacks*.
- Affects correctness *and obliviousness*!


ORAM
MAC
Server
key
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}_{\text{old}}$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}_{\text{old}}, \text{tag}_{\text{old}})$
10
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}_{\text{old}}, (\widehat{\text{addr}}, \widehat{\text{data}}_{\text{old}})\right) = 0$

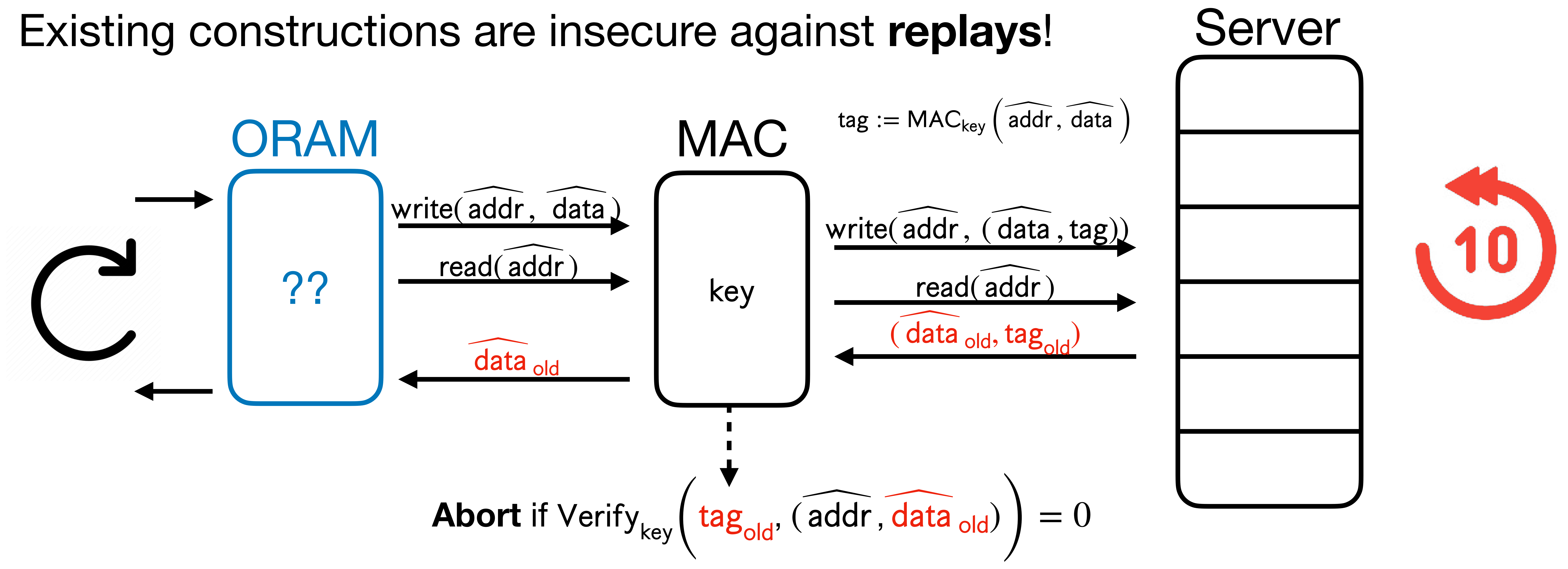
Technique #2: MACs

- MACs are **insufficient** because the server can do *replay attacks*.
- Affects correctness *and obliviousness*!
- Existing constructions are insecure against **replays**!

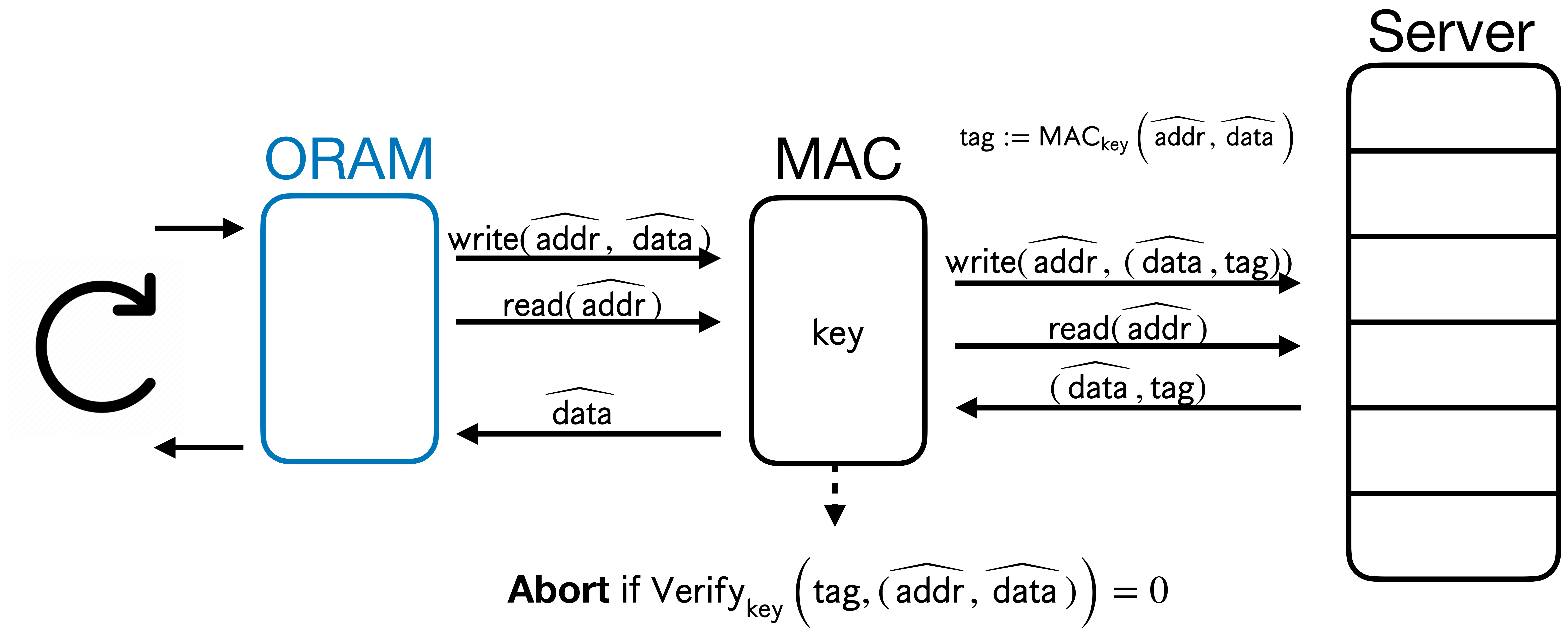

Server
ORAM
MAC
key
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}_{\text{old}}$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}_{\text{old}}, \text{tag}_{\text{old}})$
10
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}_{\text{old}}, (\widehat{\text{addr}}, \widehat{\text{data}}_{\text{old}})\right) = 0$

Technique #2: MACs

- MACs are **insufficient** because the server can do *replay attacks*.
- Affects correctness *and obliviousness*!
- Existing constructions are insecure against **replays**!

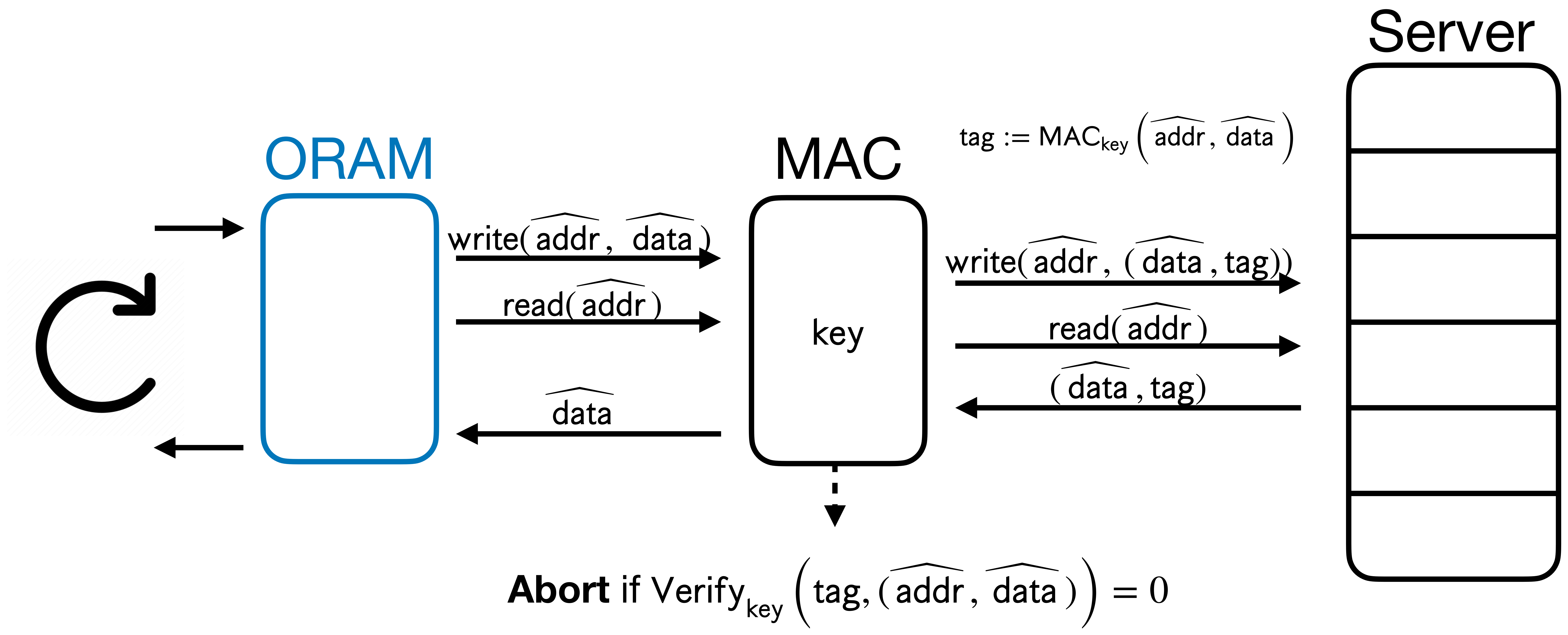

Server
ORAM
??
MAC
key
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}_{\text{old}}$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}_{\text{old}}, \text{tag}_{\text{old}})$
10
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}_{\text{old}}, (\widehat{\text{addr}}, \widehat{\text{data}}_{\text{old}})\right) = 0$

Technique #2: MACs


ORAM
MAC
key
Server
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{tag})$
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}})\right) = 0$

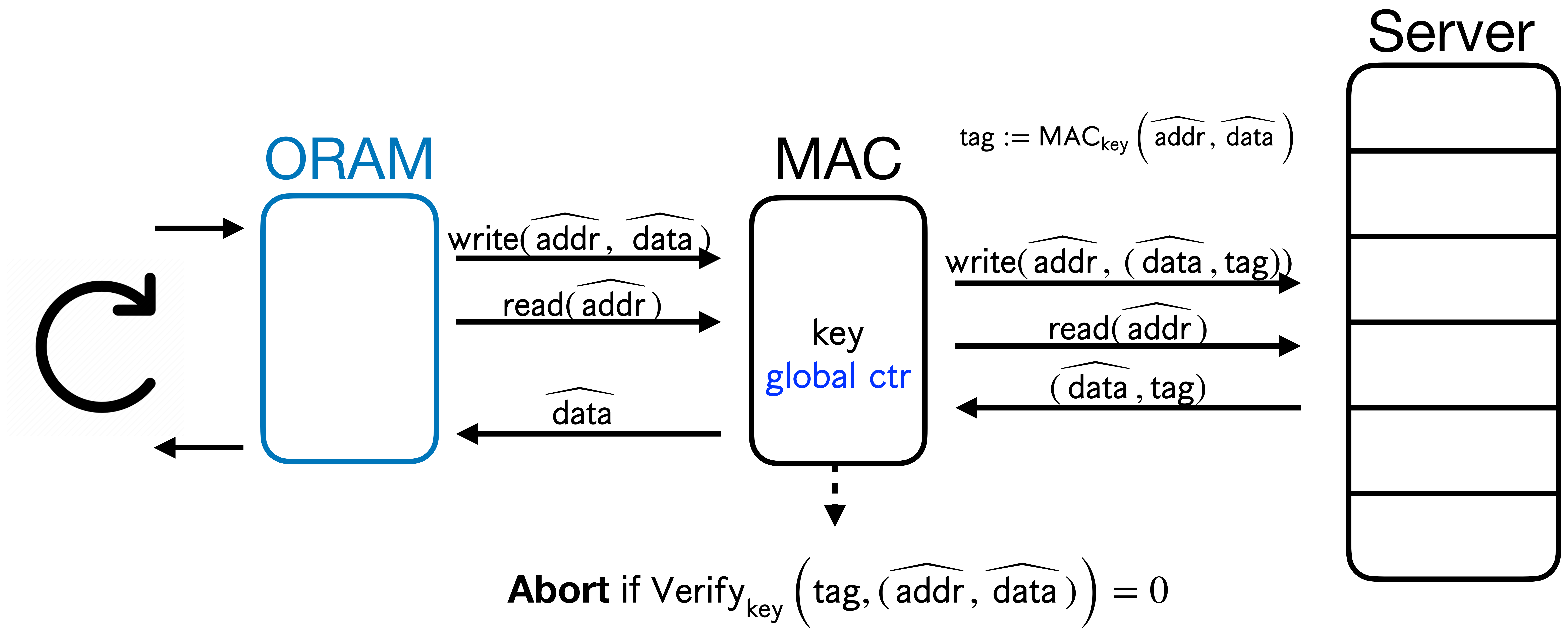
Technique #2: MACs

- Standard fix: Append a **time-stamp!**


ORAM
MAC
key
Server
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
$\text{tag} := \text{MAC}_{\text{key}}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{tag})$
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}})\right) = 0$

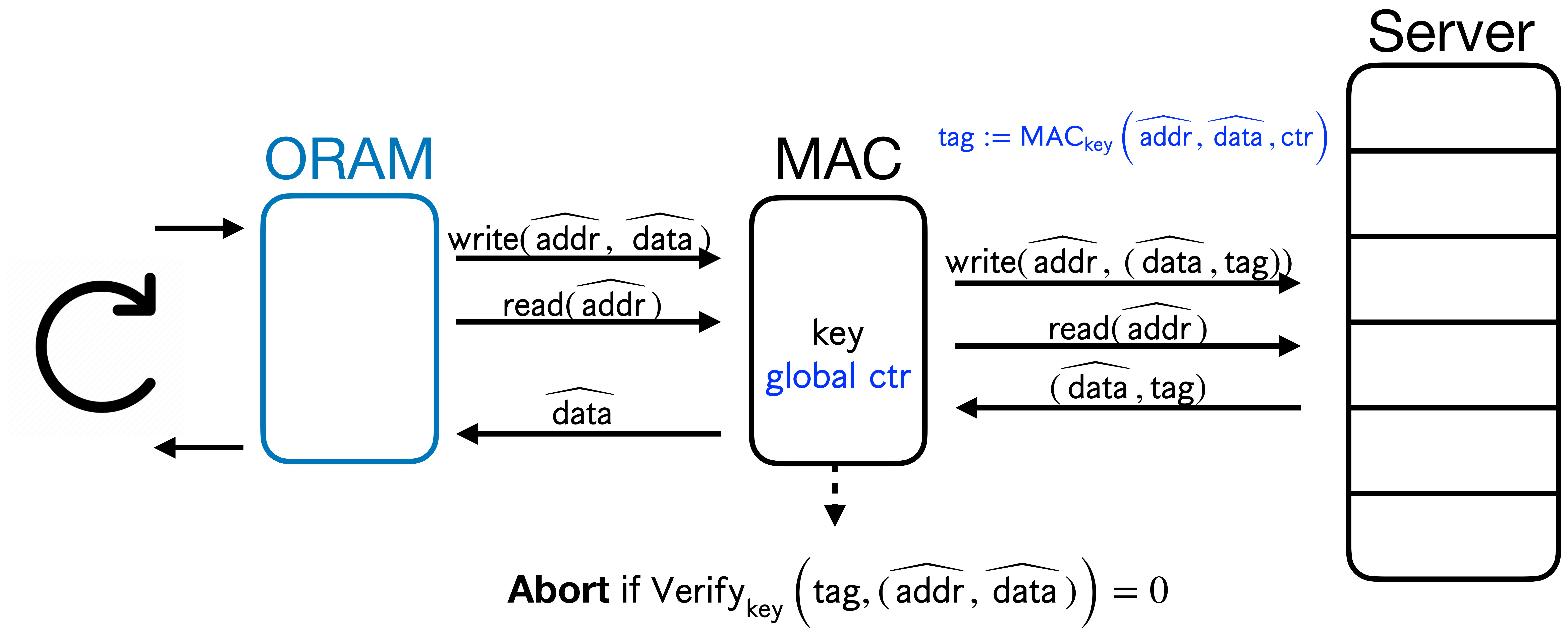
Technique #2: MACs

- Standard fix: Append a **time-stamp!**


ORAM
MAC
Server
key
global ctr
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}\right)$
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{tag})$
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}})\right) = 0$

Technique #2: MACs

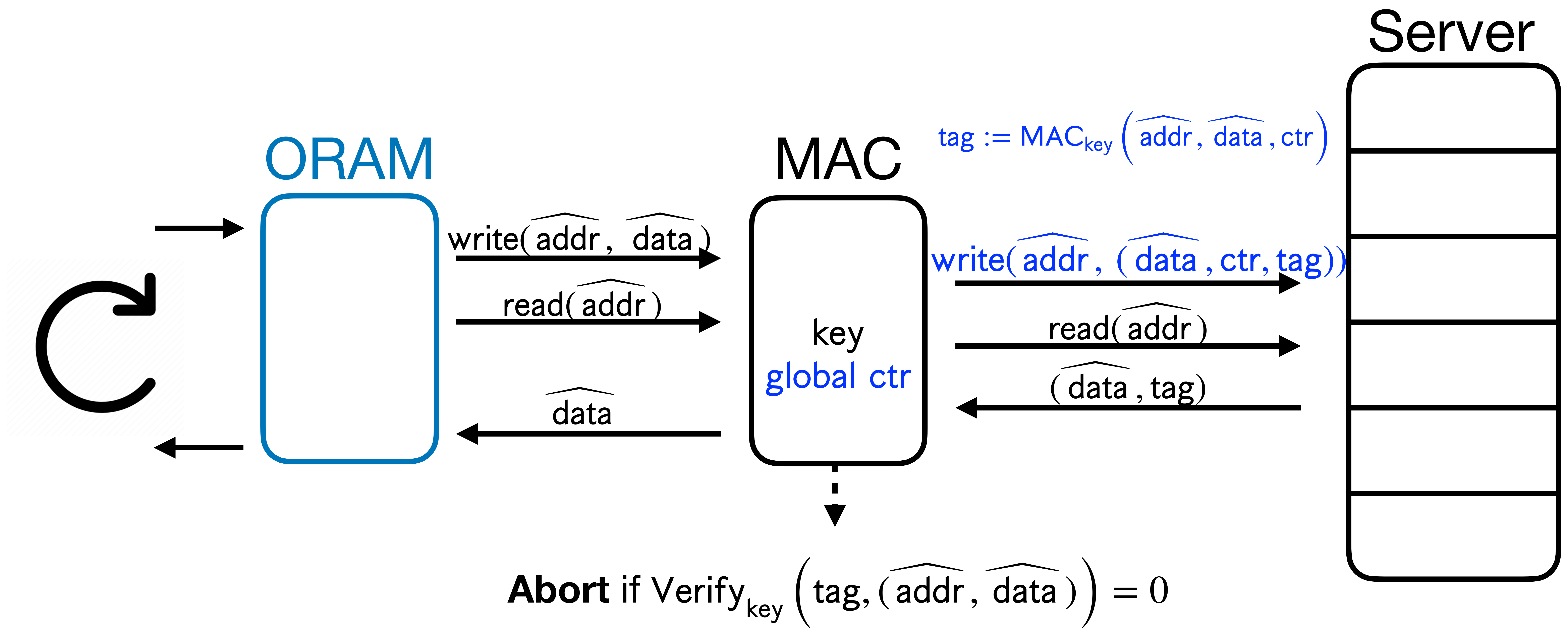
- Standard fix: Append a **time-stamp!**



Server
ORAM
MAC
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}, \text{ctr}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
key
global ctr
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{tag})$
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}})\right) = 0$

Technique #2: MACs

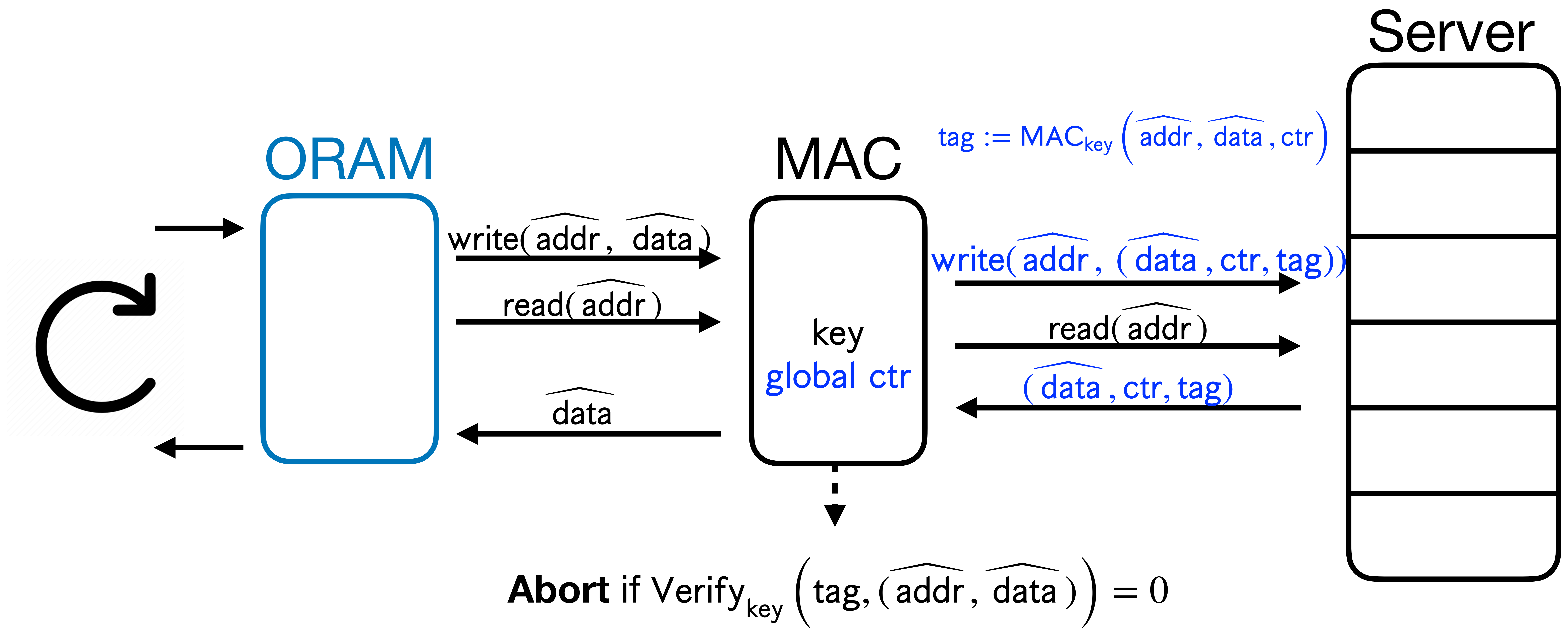
- Standard fix: Append a **time-stamp!**



Server
ORAM
MAC
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}, \text{ctr}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
key
global ctr
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{ctr}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{tag})$
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}})\right) = 0$

Technique #2: MACs

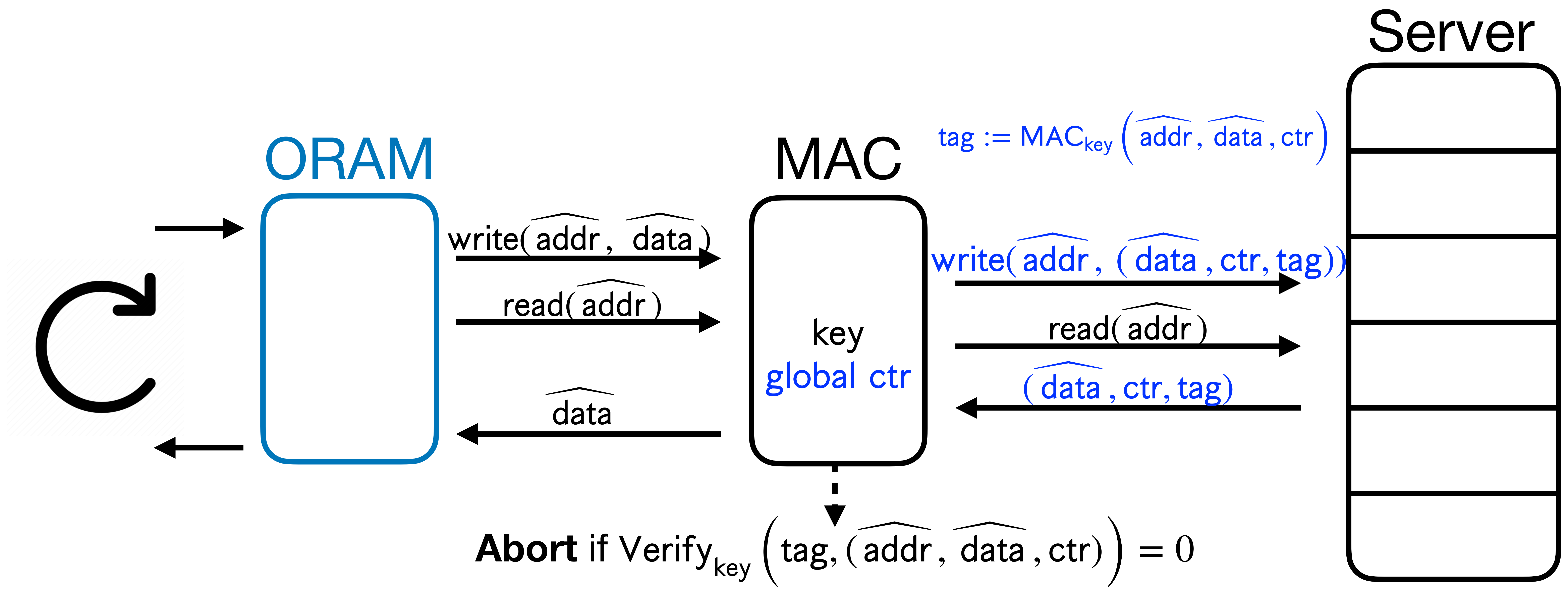
- Standard fix: Append a **time-stamp!**



Server
ORAM
MAC
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}, \text{ctr}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
key
global ctr
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{ctr}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{ctr}, \text{tag})$
Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}})\right) = 0$

Technique #2: MACs

- Standard fix: Append a **time-stamp!**



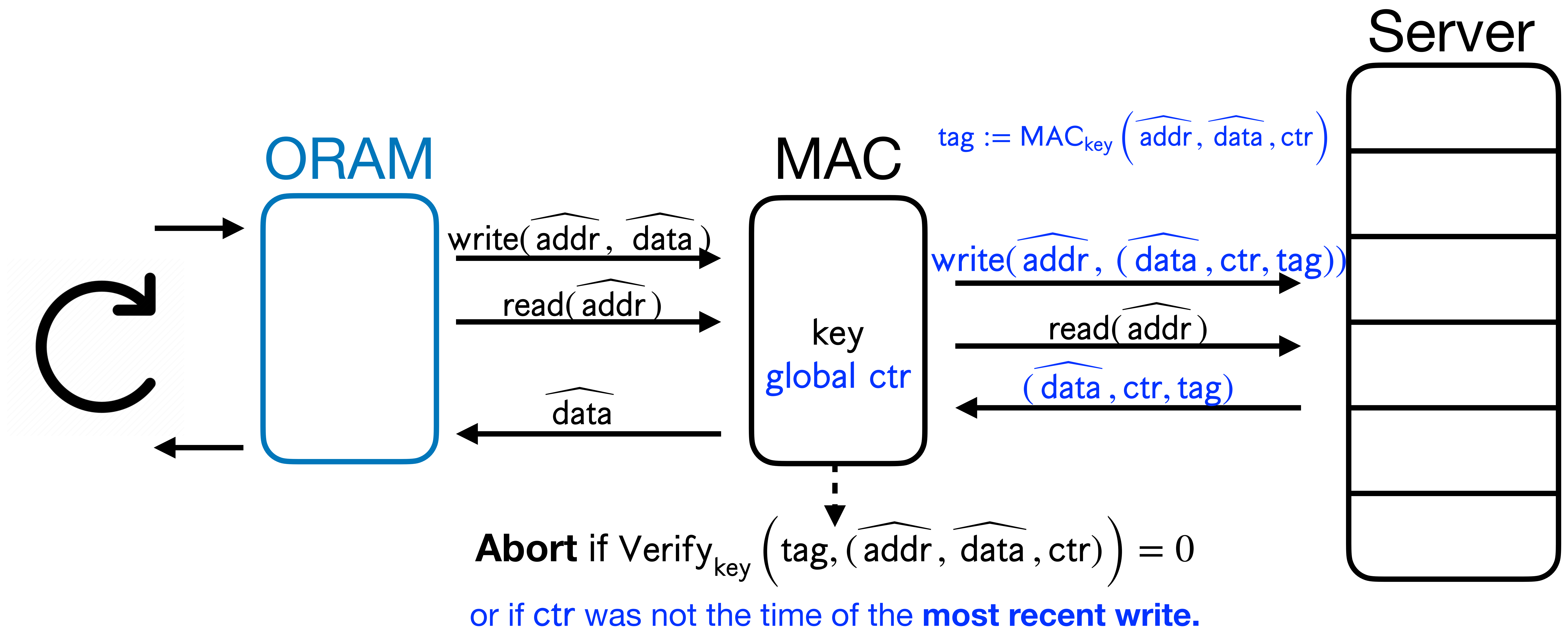
Server
ORAM
MAC
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}, \text{ctr}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
key
global ctr
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{ctr}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{ctr}, \text{tag})$

Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}}, \text{ctr})\right) = 0$

or if ctr was not the time of the **most recent write.**

Technique #2: MACs

- Standard fix: Append a **time-stamp!**
- Some subroutines in OptORAMa require $\Omega(N)$ bits of local space to verify the counter of the most recent write.


Server
ORAM
MAC
tag := $\mathrm{MAC}_{\mathrm{key}}\left(\widehat{\mathrm{addr}}, \widehat{\mathrm{data}}, \mathrm{ctr}\right)$
write($\widehat{\mathrm{addr}}$, $\widehat{\mathrm{data}}$)
read($\widehat{\mathrm{addr}}$)
$\widehat{\mathrm{data}}$
key
global ctr
write($\widehat{\mathrm{addr}}$, ($\widehat{\mathrm{data}}$, ctr, tag))
read($\widehat{\mathrm{addr}}$)
($\widehat{\mathrm{data}}$, ctr, tag)

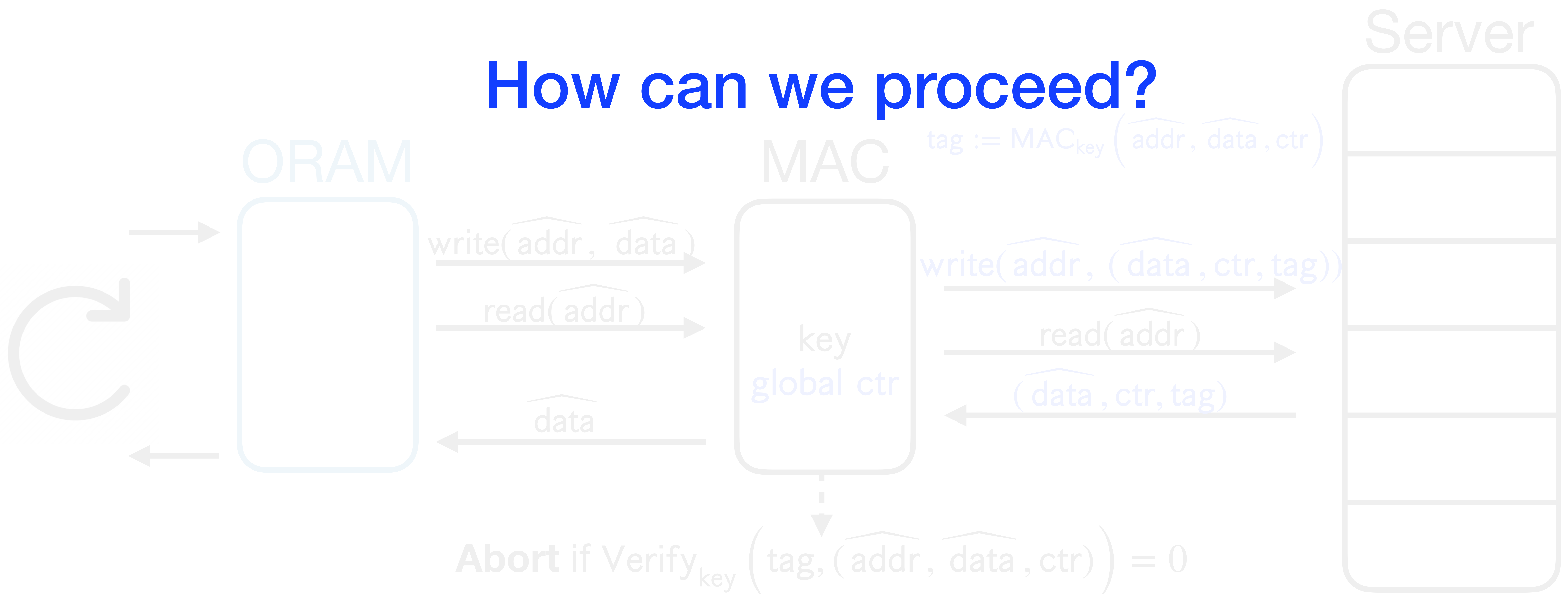
Abort if $\mathrm{Verify}_{\mathrm{key}}\left(\mathrm{tag}, (\widehat{\mathrm{addr}}, \widehat{\mathrm{data}}, \mathrm{ctr})\right) = 0$

or if ctr was not the time of the **most recent write.**

Technique #2: MACs

- Standard fix: Append a **time-stamp!**
- Some subroutines in OptORAMa require $\Omega(N)$ bits of local space to verify the counter of the most recent write.

How can we proceed?



Server
ORAM
MAC
$\text{tag} := \text{MAC}_{\text{key}}\left(\widehat{\text{addr}}, \widehat{\text{data}}, \text{ctr}\right)$
$\text{write}(\widehat{\text{addr}}, \widehat{\text{data}})$
$\text{read}(\widehat{\text{addr}})$
$\widehat{\text{data}}$
key
global ctr
$\text{write}(\widehat{\text{addr}}, (\widehat{\text{data}}, \text{ctr}, \text{tag}))$
$\text{read}(\widehat{\text{addr}})$
$(\widehat{\text{data}}, \text{ctr}, \text{tag})$

Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}}, \text{ctr})\right) = 0$

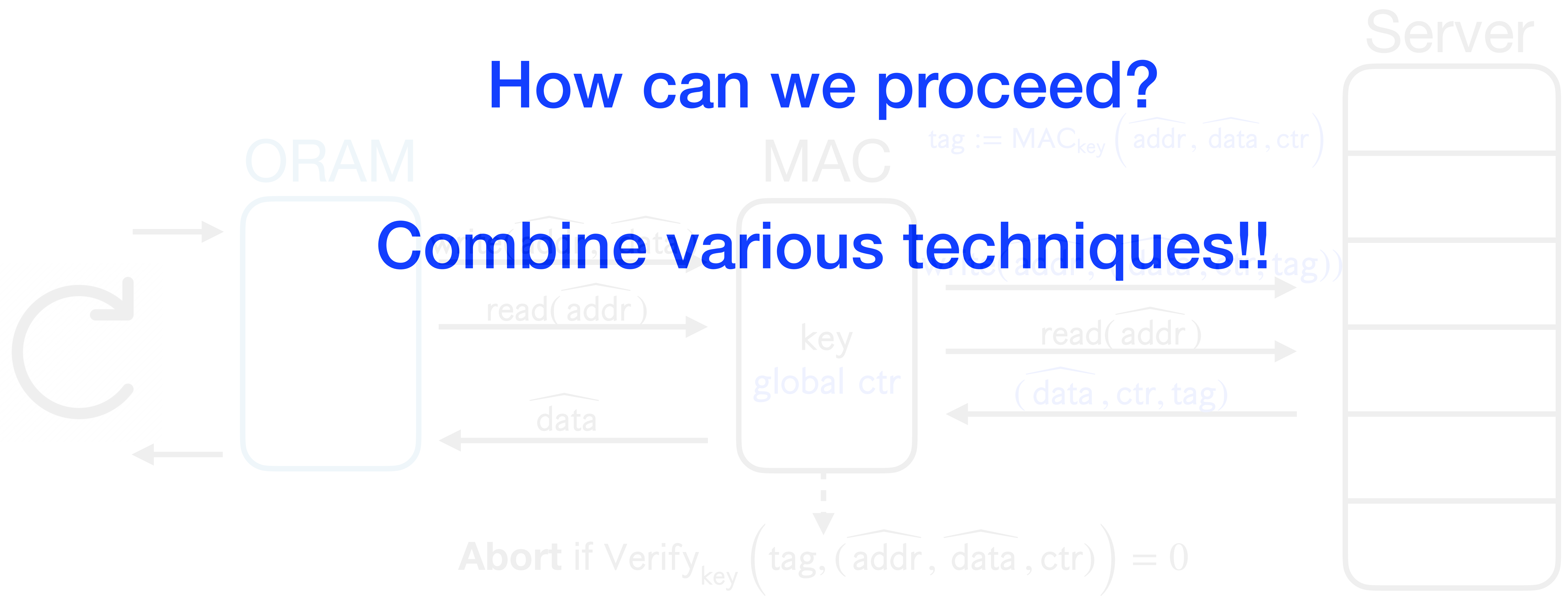
or if ctr was not the time of the **most recent write.**

Technique #2: MACs

- Standard fix: Append a **time-stamp!**
- Some subroutines in OptORAMa require $\Omega(N)$ bits of local space to verify the counter of the most recent write.

How can we proceed?

Combine various techniques!!



Abort if $\text{Verify}_{\text{key}}\left(\text{tag}, (\widehat{\text{addr}}, \widehat{\text{data}}, \text{ctr})\right) = 0$

or if ctr was not the time of the **most recent write.**

Overarching Ideas

Overarching Ideas

- **MAC**s

Overarching Ideas

- **MAC**s
  - For portions of the scheme where access pattern is **time-stampable**, we can do this with $O(1)$ blowup.

Overarching Ideas

- **MAC**s
    - For portions of the scheme where access pattern is **time-stampable**, we can do this with $O(1)$ blowup.
- And **Che**cking **E**fficiently and **Se**curely

Overarching Ideas

- **MAC**s
  - For portions of the scheme where access pattern is **time-stampable**, we can do this with $O(1)$ blowup.
- And **Che**cking **E**fficiently and **Se**curely
  - Use a **offline memory checker** with amortized $O(1)$ blowup, and use it in a careful and secure way.

Overarching Ideas

- **MAC**s
  - For portions of the scheme where access pattern is **time-stampable**, we can do this with $O(1)$ blowup.
- And **Che**cking **E**fficiently and **Se**curely
  - Use a **offline memory checker** with amortized $O(1)$ blowup, and use it in a careful and secure way.
- In some cases, we have to **interleave** time-stamping and offline checking!

Overarching Ideas

- **MAC**s
    - For portions of the scheme where access pattern is **time-stampable**, we can do this with $O(1)$ blowup.
- And **Che**cking **E**fficiently and **Se**curely
    - Use a **offline memory checker** with amortized $O(1)$ blowup, and use it in a careful and secure way.
- In some cases, we have to **interleave** time-stamping and offline checking!
- i.e. MAC and CheESe

Overarching Ideas

- **MAC**s
  - For portions of the scheme where access pa
    do this with $O(1)$ blowup.
- And **Che**cking **E**fficiently and **Se**curely
  - Use a **offline memory checker** with amortiz
    careful and secure way.
- In some cases, we have to **interleave** time-star
- i.e. MAC and CheESe

Overarching Ideas

- **MACs**
  - For portions of the scheme where access pa do this with $O(1)$ blowup.
- And **Che**cking **E**fficiently and **Se**curely
  - Use a **offline memory checker** with amortiz careful and secure way.
- In some cases, we have to **interleave** time-star
- i.e. MAC and CheESe

Created by Dall-E

Summary & Conclusion

Summary & Conclusion

- We construct **MacORAMa**, a **maliciously** secure ORAM with **optimal** $O(\log N)$ overhead and $O(1)$ local space.

Summary & Conclusion

- We construct **MacORAMa**, a **maliciously** secure ORAM with **optimal** $O(\log N)$ overhead and $O(1)$ local space.
    - **Another interpretation**: First *oblivious* memory checker with $O(\log N)$ overhead, matching best *non-oblivious* memory checker overhead.

Summary & Conclusion

- We construct **MacORAMa**, a **maliciously** secure ORAM with **optimal** $O(\log N)$ overhead and $O(1)$ local space.
  - **Another interpretation**: First *oblivious* memory checker with $O(\log N)$ overhead, matching best *non-oblivious* memory checker overhead.
  - Assumptions are **provably minimal** (OWF necessary and sufficient).

Summary & Conclusion

- We construct **MacORAMa**, a **maliciously** secure ORAM with **optimal** $O(\log N)$ overhead and $O(1)$ local space.
  - **Another interpretation**: First *oblivious* memory checker with $O(\log N)$ overhead, matching best *non-oblivious* memory checker overhead.
  - Assumptions are **provably minimal** (OWF necessary and sufficient).
- An overhead-preserving black-box compiler from honest-but-curious to malicious security has a barrier.

Summary & Conclusion

- We construct **MacORAMa**, a **maliciously** secure ORAM with **optimal** $O(\log N)$ overhead and $O(1)$ local space.
    - **Another interpretation**: First *oblivious* memory checker with $O(\log N)$ overhead, matching best *non-oblivious* memory checker overhead.
    - Assumptions are **provably minimal** (OWF necessary and sufficient).
- An overhead-preserving black-box compiler from honest-but-curious to malicious security has a barrier.
- Instead, we develop **memory checking** techniques in the ORAM setting that should generalize to future constructions.

Open Questions

Open Questions

- Any maliciously secure ORAM with $O(\log N)$ overhead with better constant factors? **OptORAMa** has large constant factors.

Open Questions

- Any maliciously secure ORAM with $O(\log N)$ overhead with better constant factors? **OptORAMa** has large constant factors.
- Any memory checker with $O(1)$ overhead? Any lower bounds? (Best constructions have $O(\log N)$ overhead.)

Thanks!

Bonus Slides

Dall-E's Attempts

ORAM vs. PIR

ORAM vs. PIR

- Private Information Retrieval (PIR) is similar to ORAM but has crucial differences:

ORAM vs. PIR

- Private Information Retrieval (PIR) is similar to ORAM but has crucial differences:
  - In PIR, the database is typically **public**.

ORAM vs. PIR

- Private Information Retrieval (PIR) is similar to ORAM but has crucial differences:
  - In PIR, the database is typically **public**.
  - Unlike ORAM, PIR allows **many clients** to access database.

ORAM vs. PIR

- Private Information Retrieval (PIR) is similar to ORAM but has crucial differences:
    - In PIR, the database is typically **public**.
    - Unlike ORAM, PIR allows **many clients** to access database.
    - PIR clients typically do not perform writes.

ORAM vs. PIR

- Private Information Retrieval (PIR) is similar to ORAM but has crucial differences:
    - In PIR, the database is typically **public**.
    - Unlike ORAM, PIR allows **many clients** to access database.
    - PIR clients typically do not perform writes.
- Because ORAMs can be **stateful**, we have better constructions under minimal assumptions.

Ideal Malicious Security

Ideal Malicious Security

- What guarantee do we want?

Ideal Malicious Security

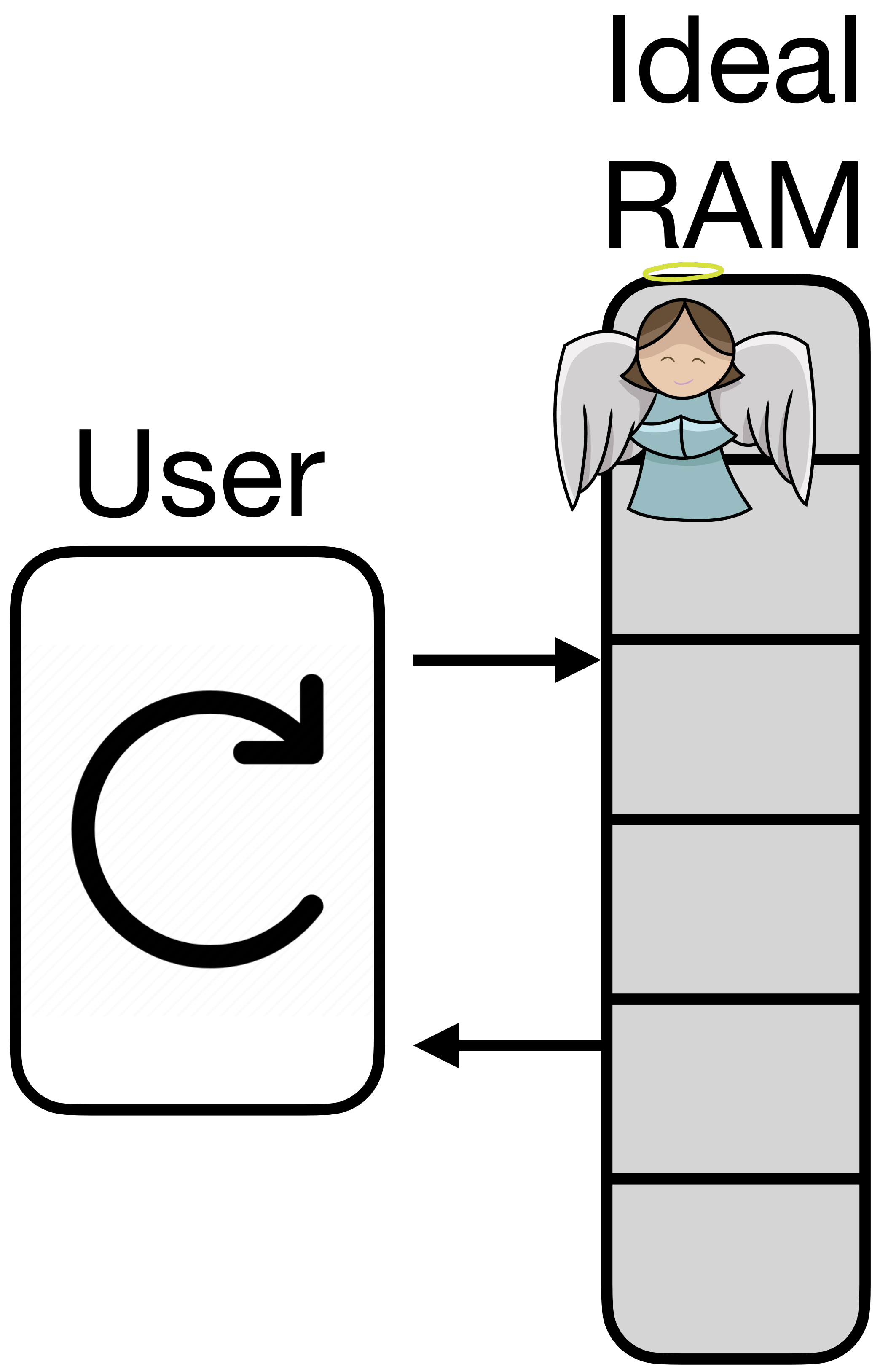
- What guarantee do we want?

    1. **Correctness**: If no abort, user should never get incorrect responses from ORAM, even if server tampers.

Ideal Malicious Security

- What guarantee do we want?

  1. **Correctness**: If no abort, user should never get incorrect responses from ORAM, even if server tampers.

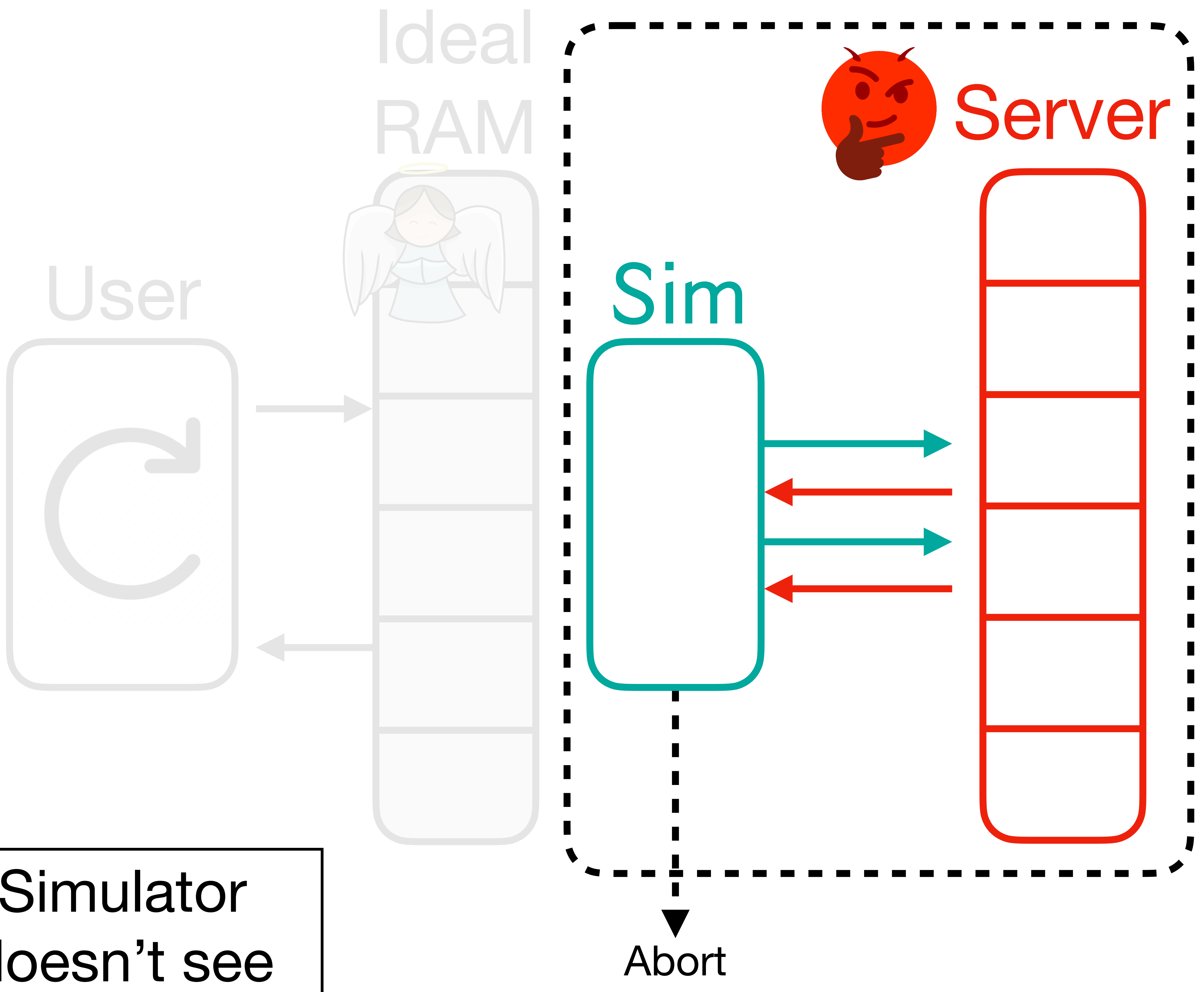


Ideal Malicious Security

- What guarantee do we want?

  1. **Correctness**: If no abort, user should never get incorrect responses from ORAM, even if server tampers.

  2. **Obliviousness**: Server shouldn't be able to learn ***anything, even by tampering***. Server should **only** be able to:



Simulator doesn't see user queries!

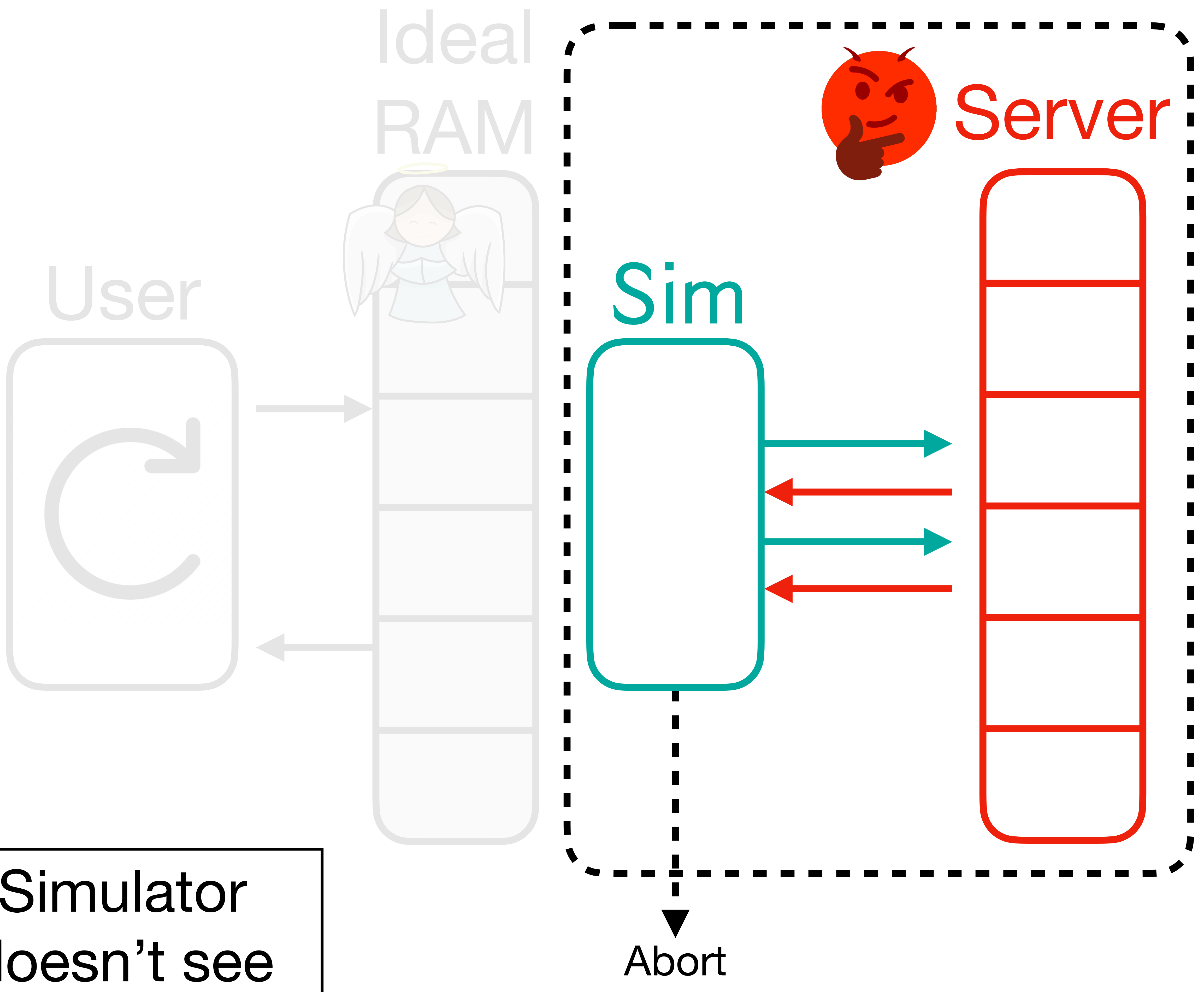
Ideal Malicious Security

- What guarantee do we want?

  1. **Correctness**: If no abort, user should never get incorrect responses from ORAM, even if server tampers.

  2. **Obliviousness**: Server shouldn't be able to learn ***anything, even by tampering***. Server should **only** be able to:

     A. Learn number of queries.



Simulator doesn't see user queries!

Ideal Malicious Security

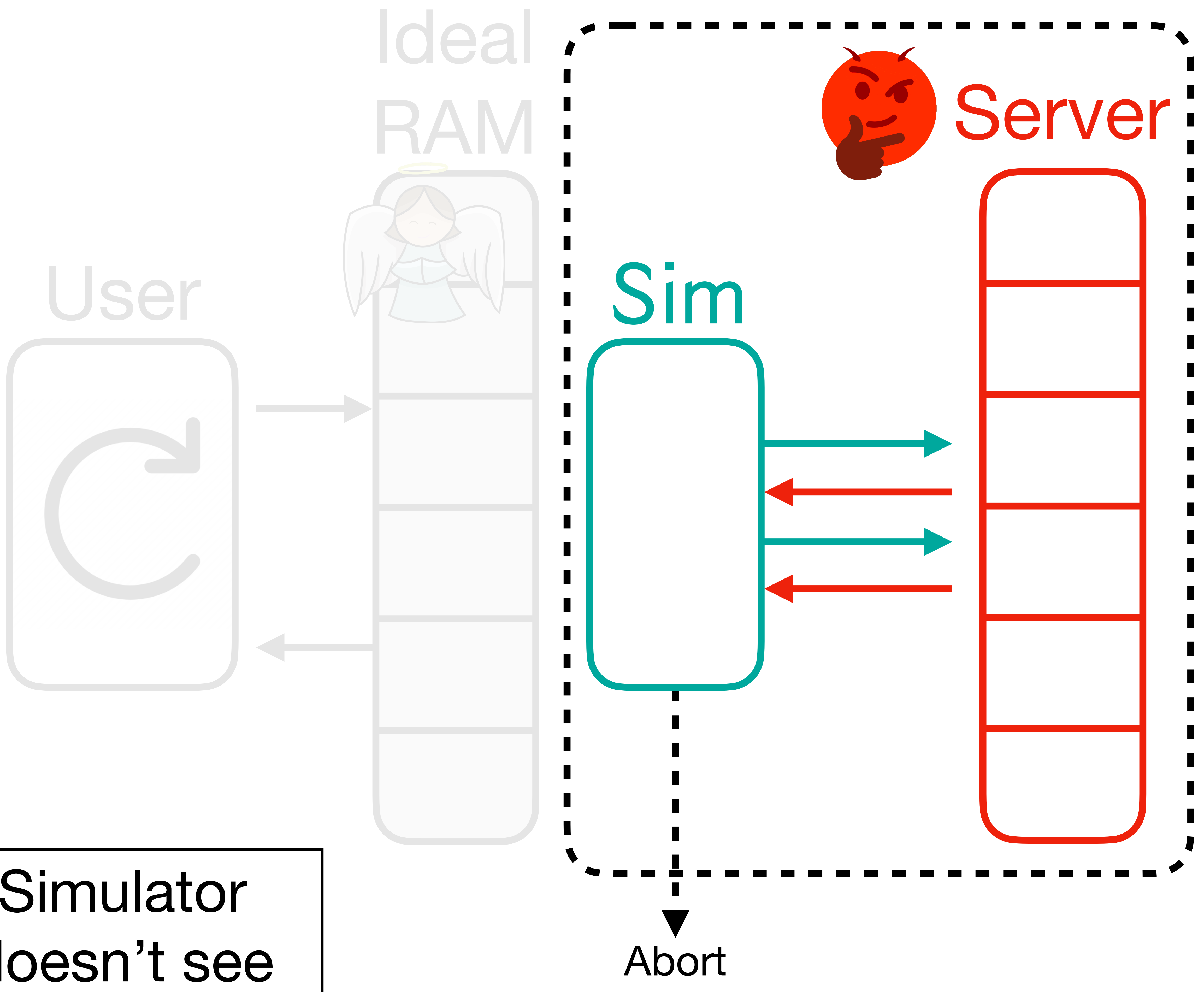
- What guarantee do we want?

  1. **Correctness**: If no abort, user should never get incorrect responses from ORAM, even if server tampers.

  2. **Obliviousness**: Server shouldn't be able to learn ***anything, even by tampering***. Server should **only** be able to:

     A. Learn number of queries.

     B. Decide whether to abort.



Simulator doesn't see user queries!

Ideal Malicious Security

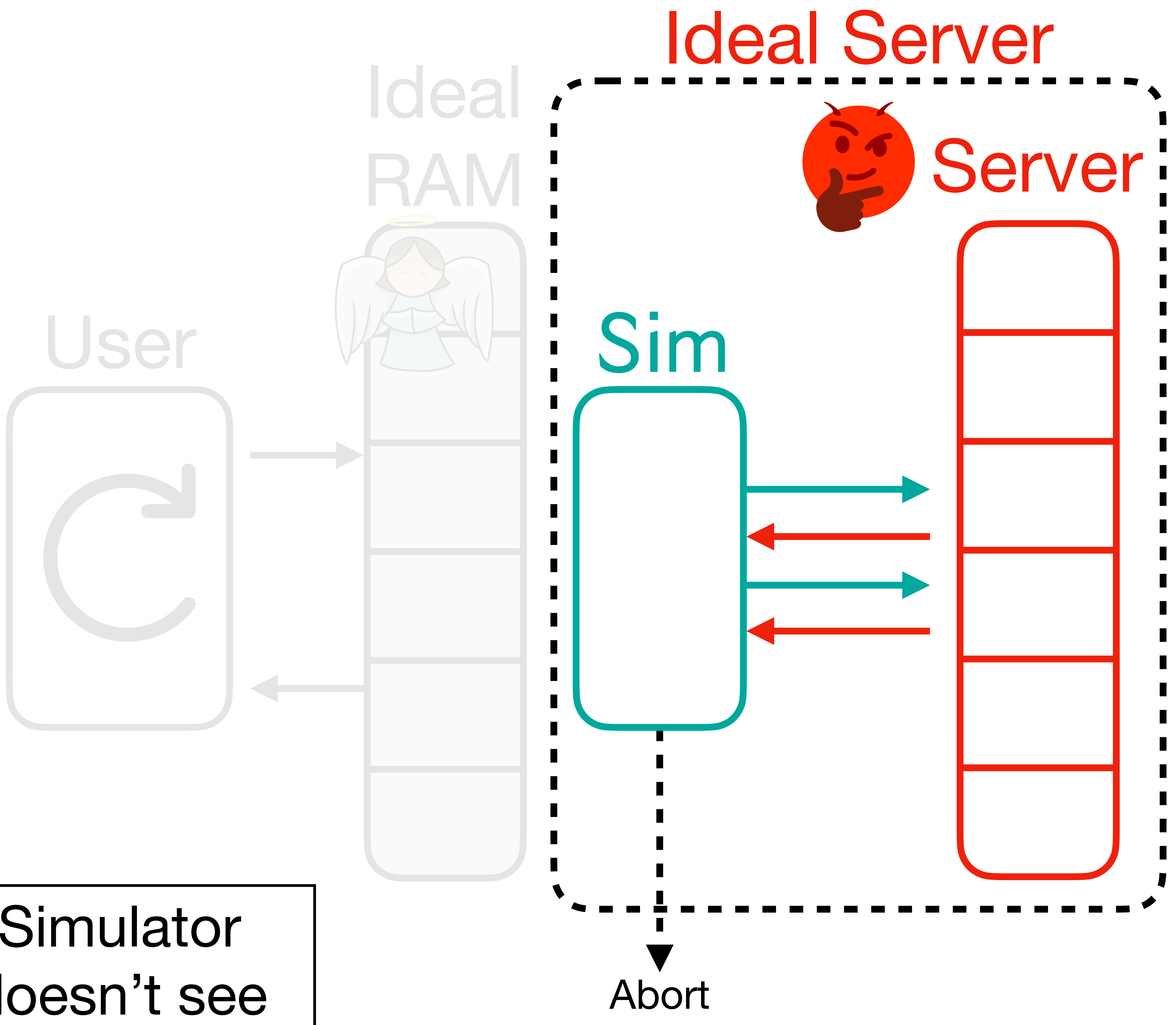
- What guarantee do we want?

  1. **Correctness**: If no abort, user should never get incorrect responses from ORAM, even if server tampers.

  2. **Obliviousness**: Server shouldn't be able to learn ***anything, even by tampering***. Server should **only** be able to:

     A. Learn number of queries.

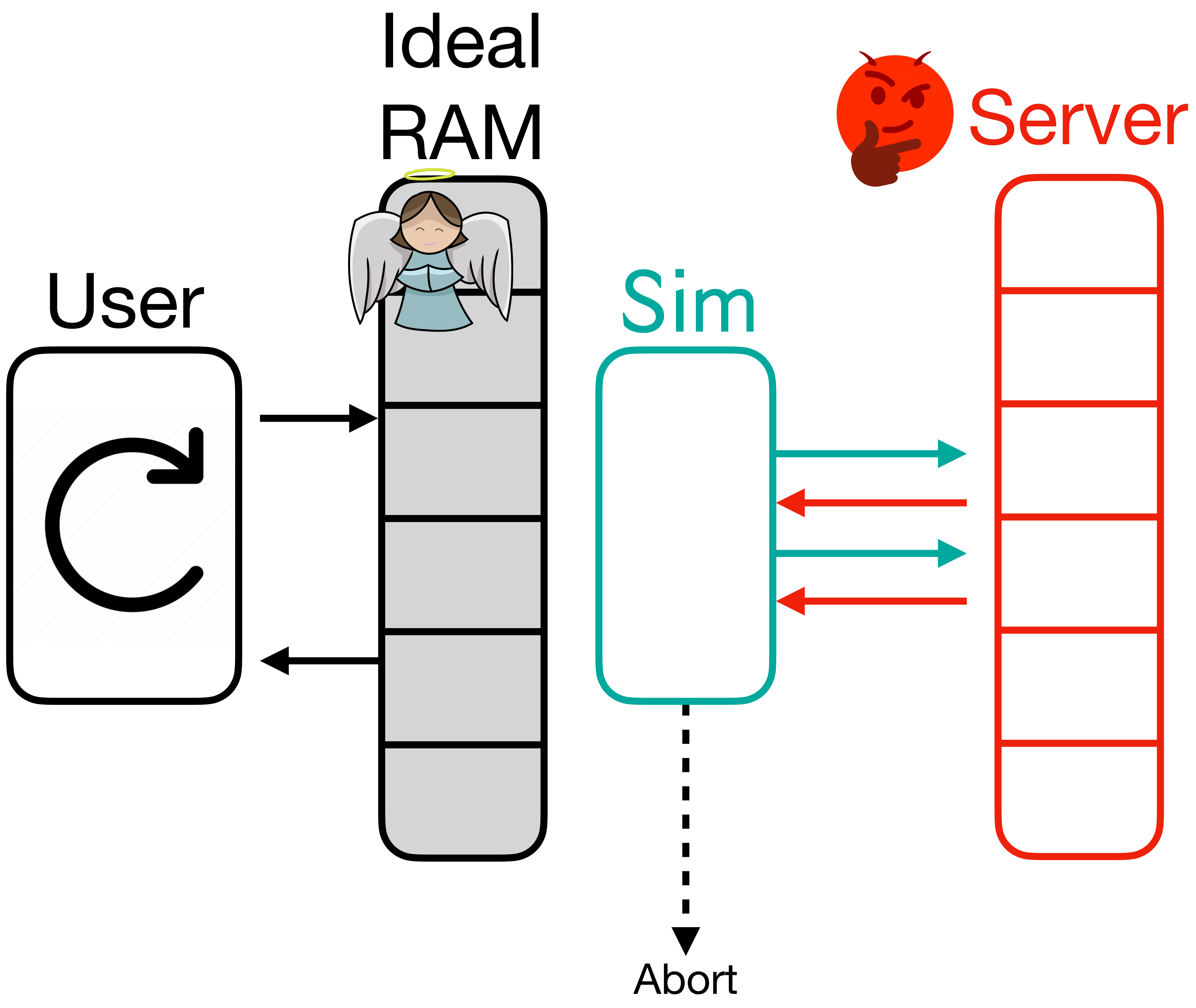
     B. Decide whether to abort.



Simulator doesn't see user queries!

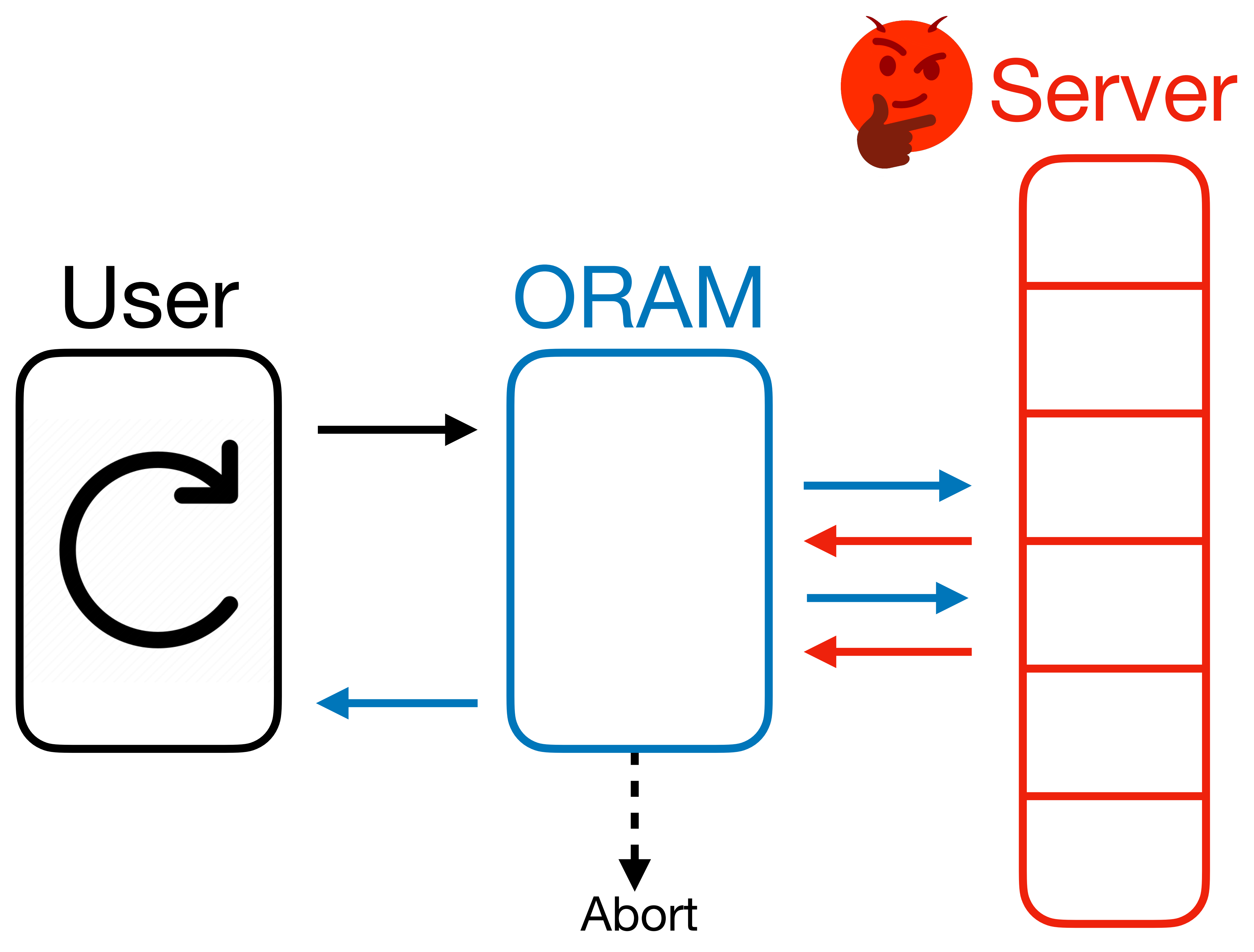
`

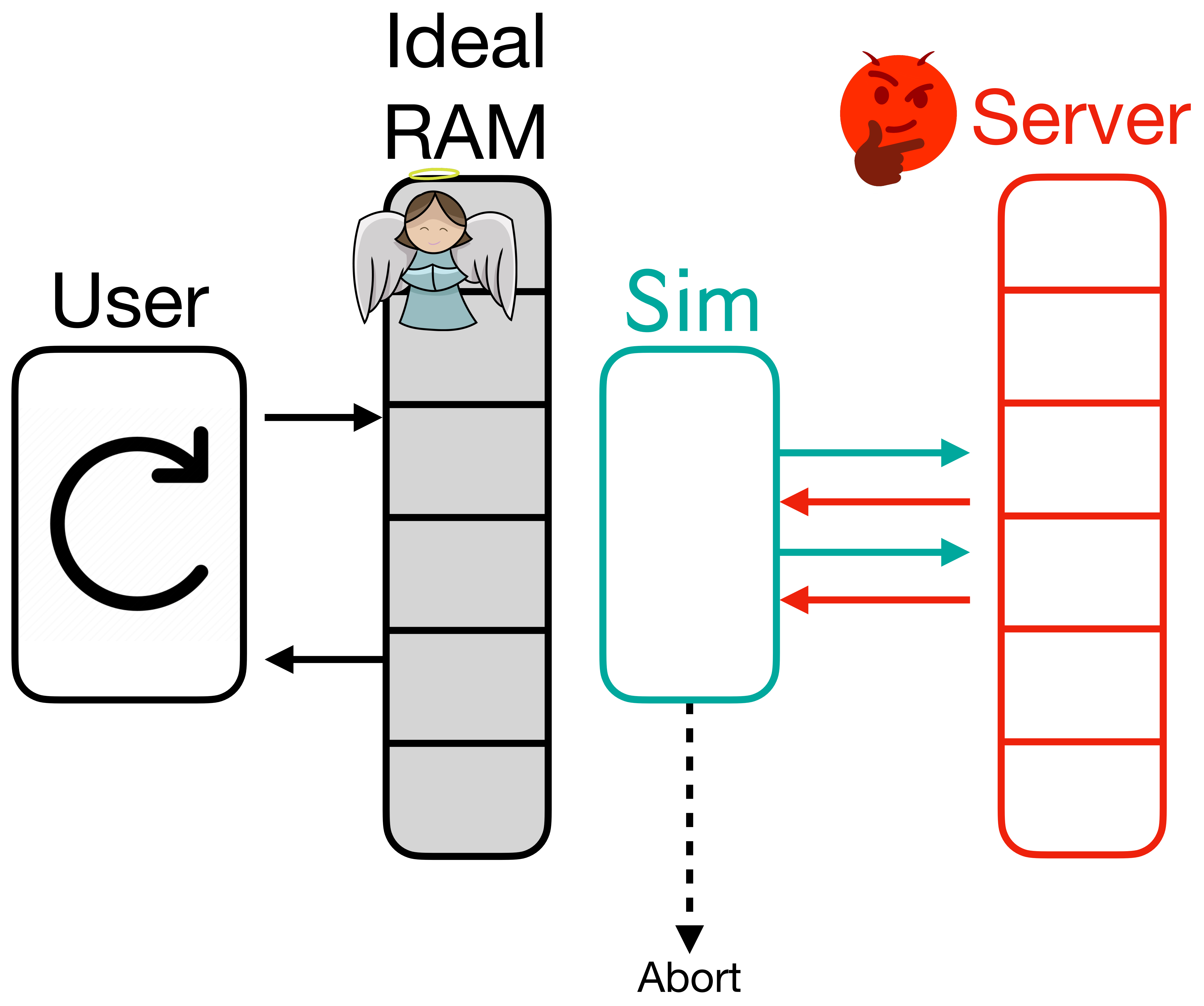
Ideal



`

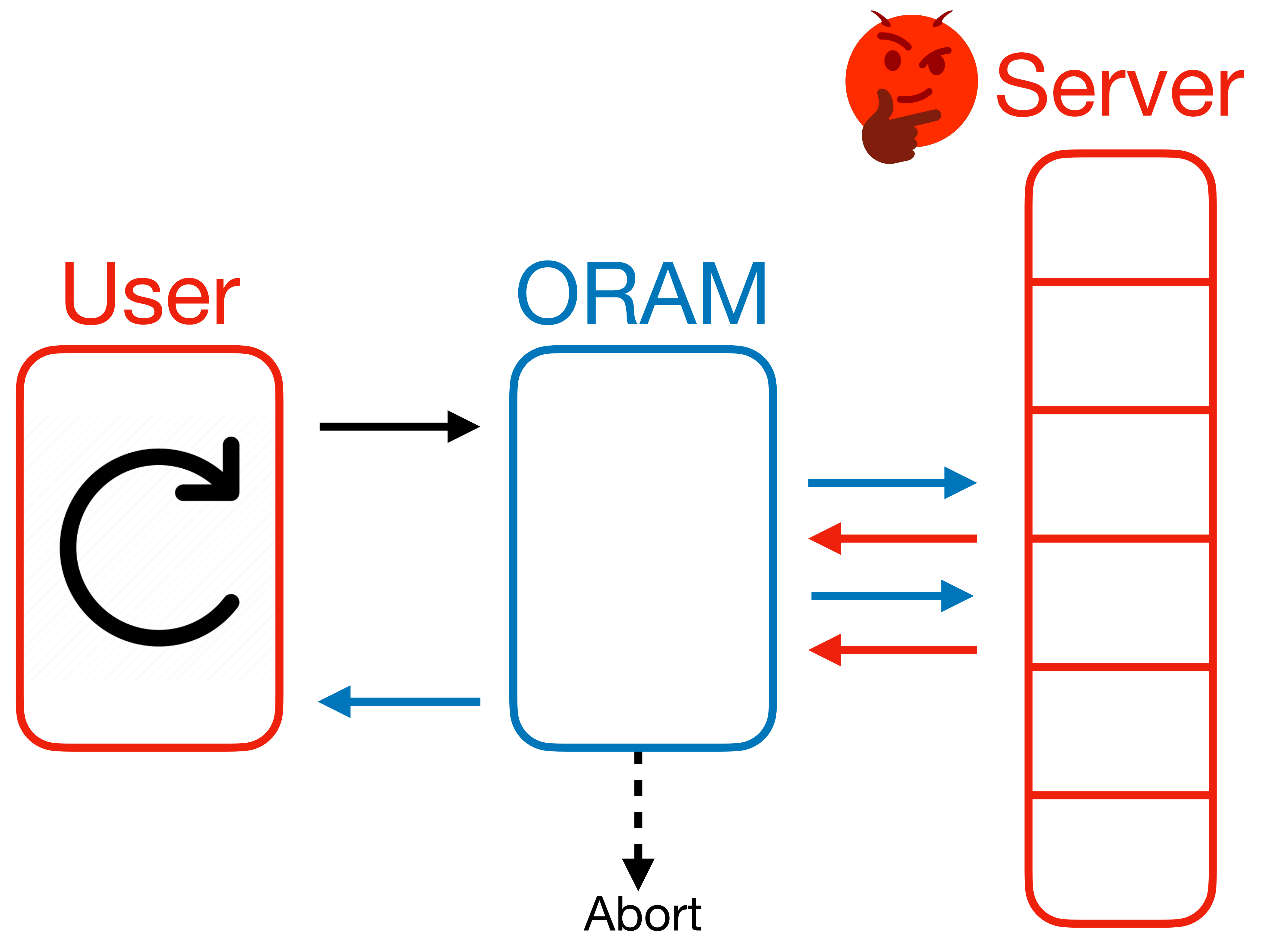
Ideal

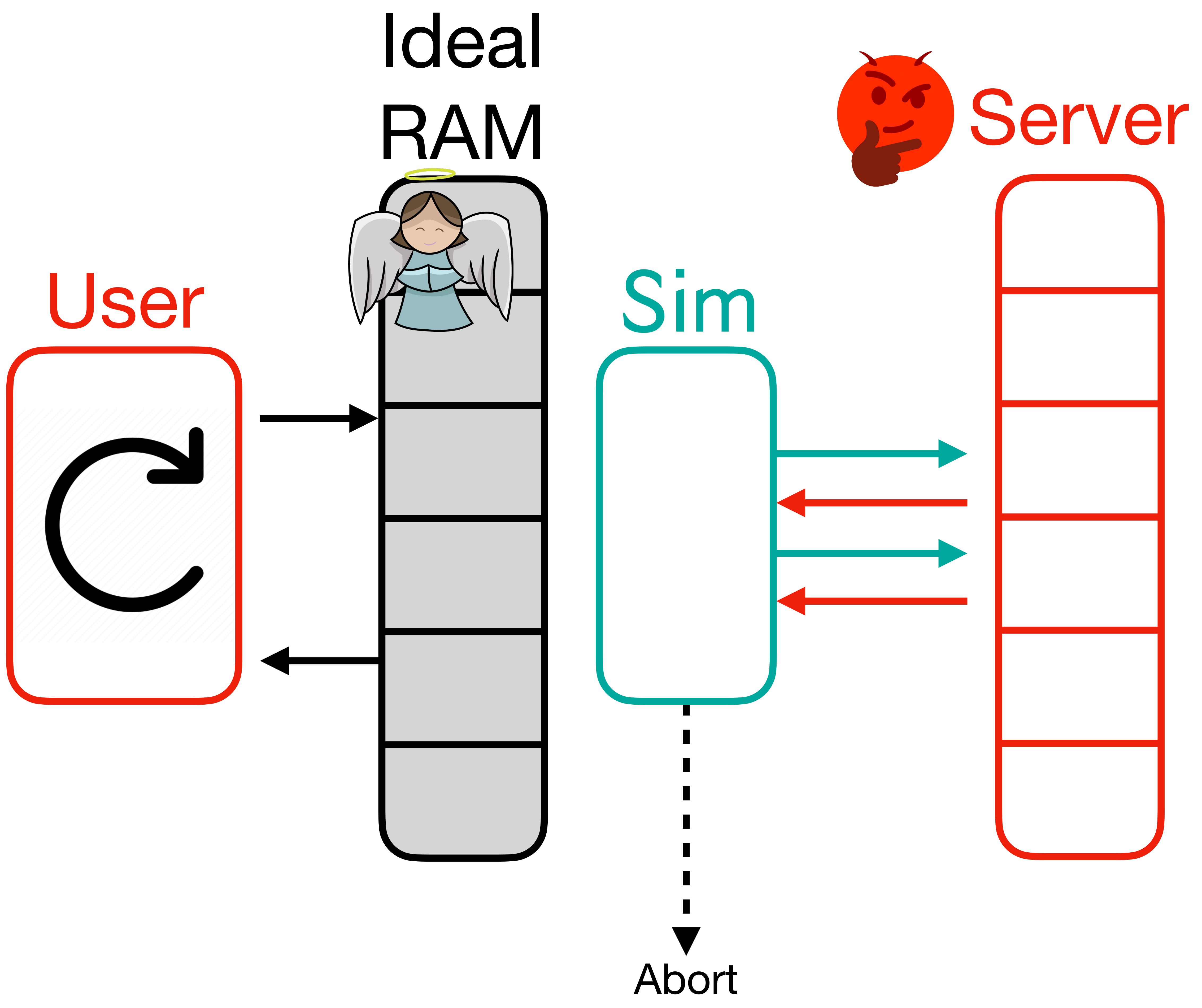




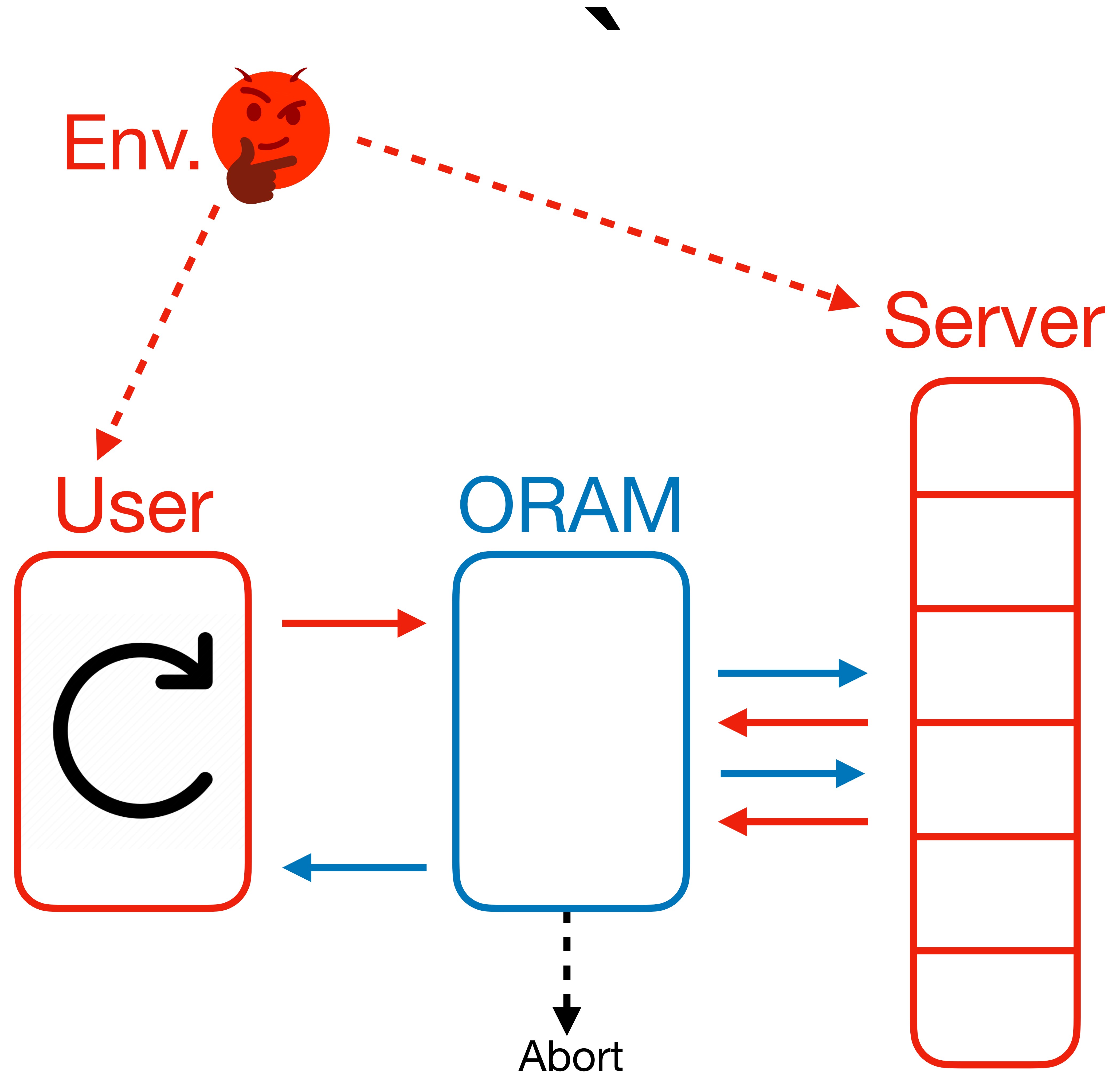
`

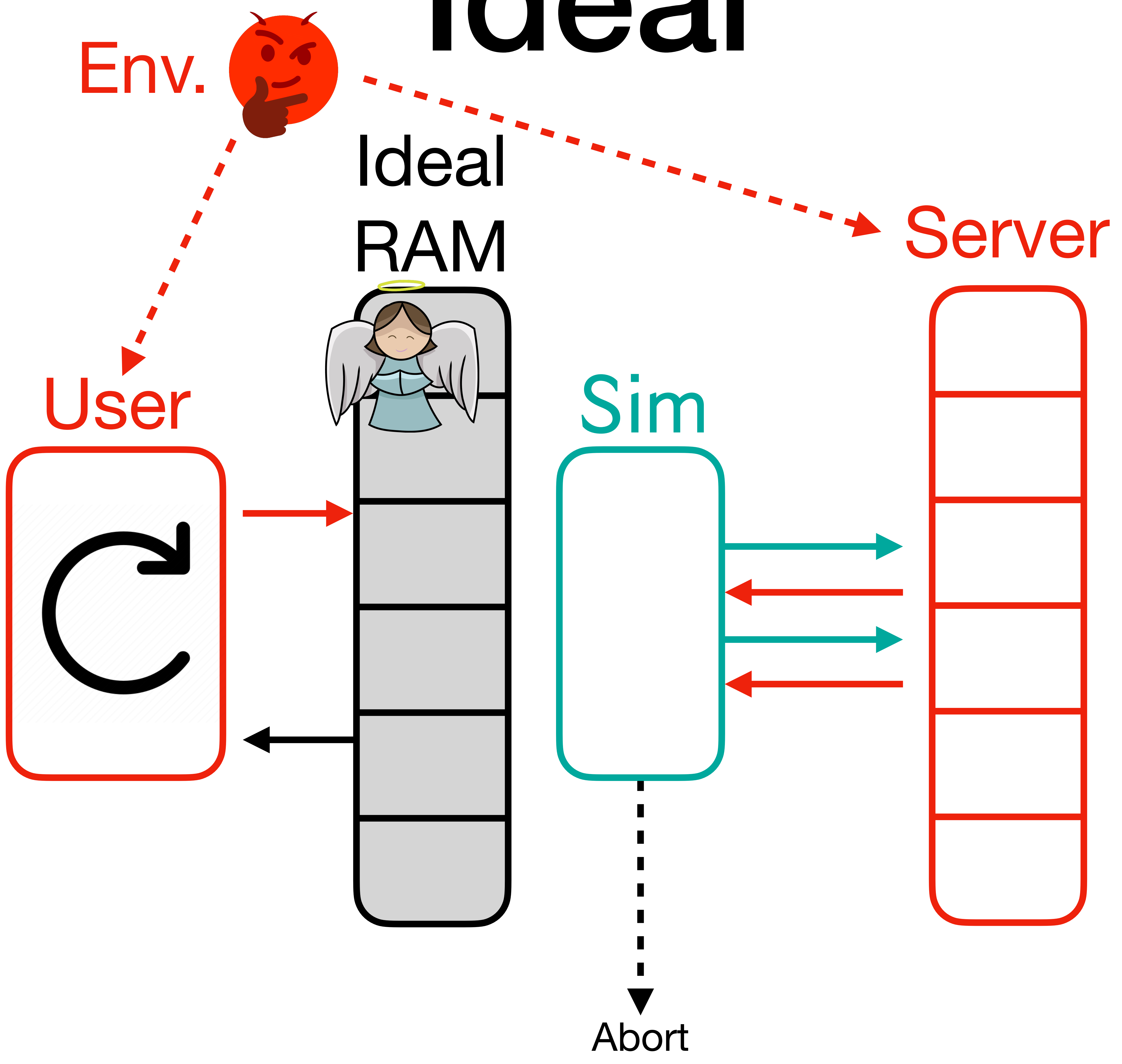
Ideal





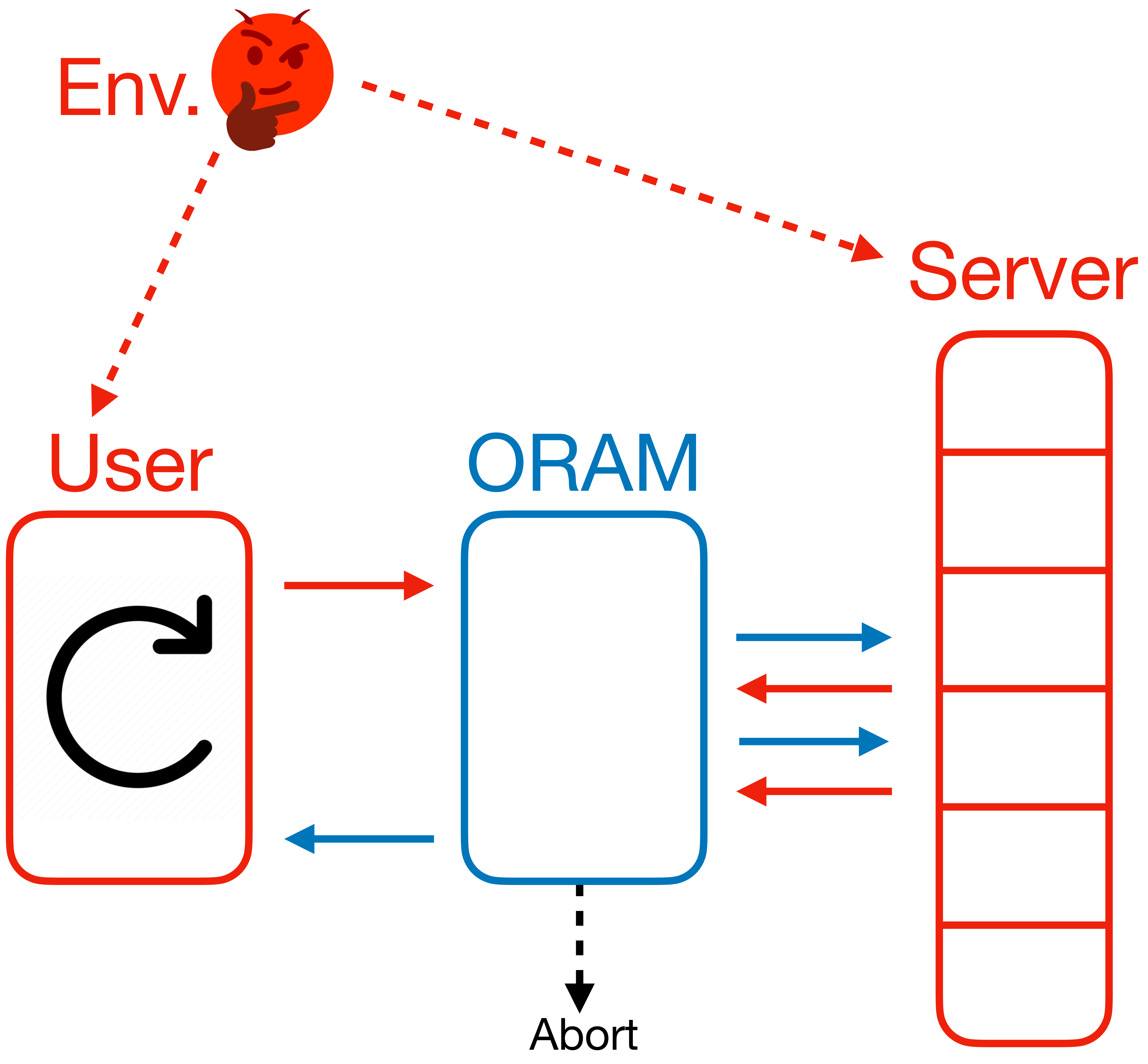
Ideal

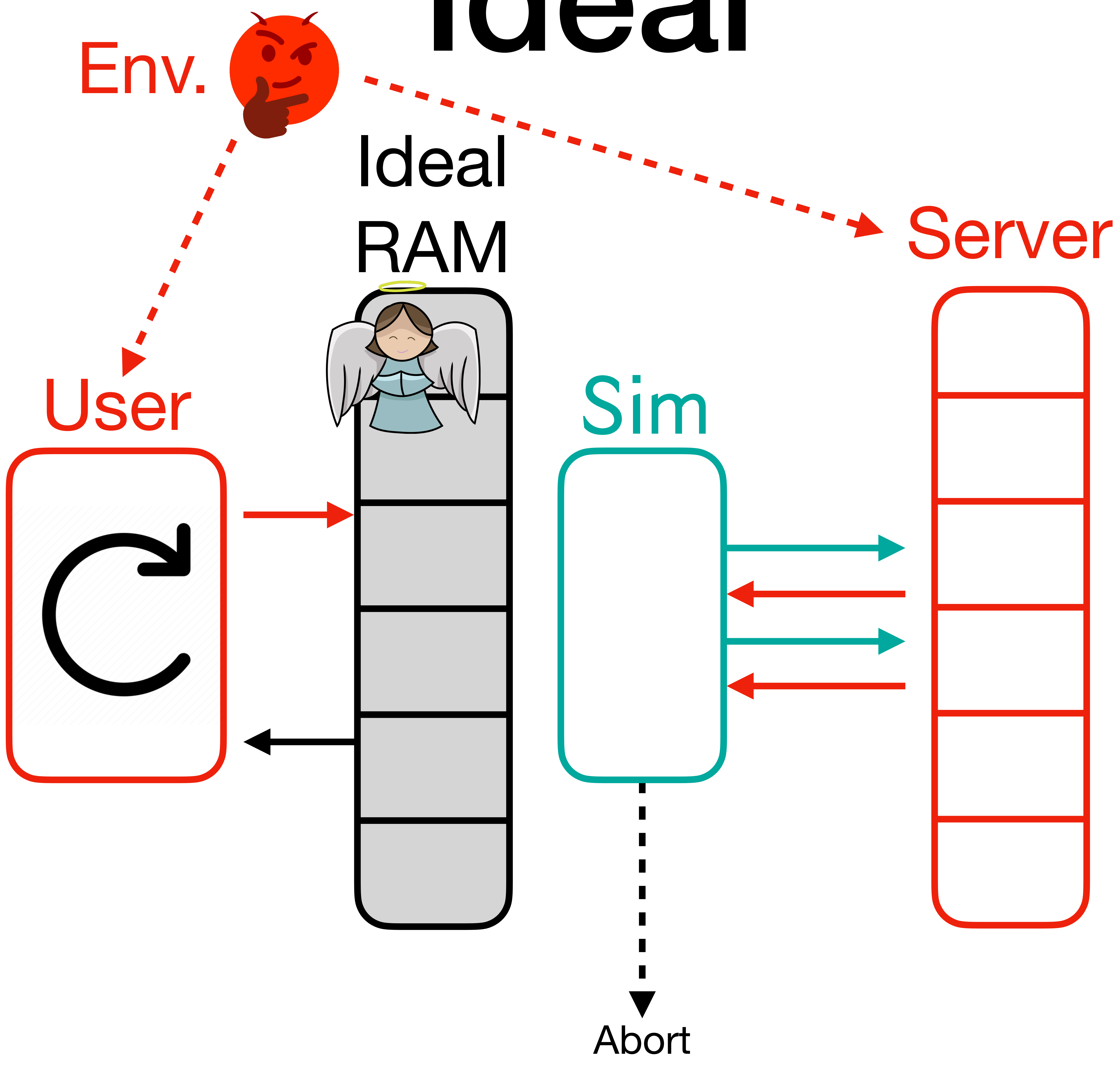

Env.
Server
User
ORAM
Abort


Env.
Ideal RAM
Server
User
Sim
Abort

`

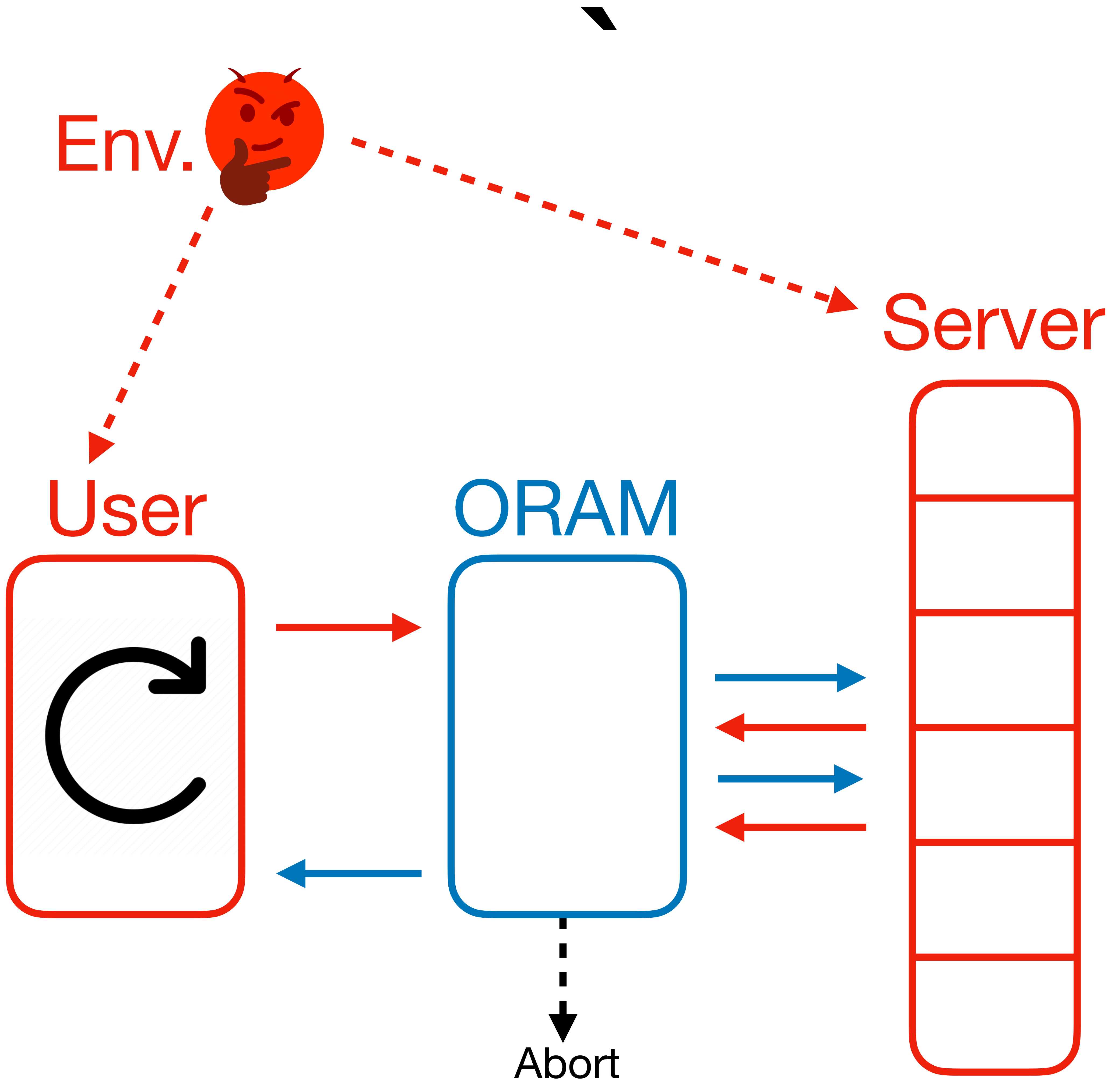
Ideal

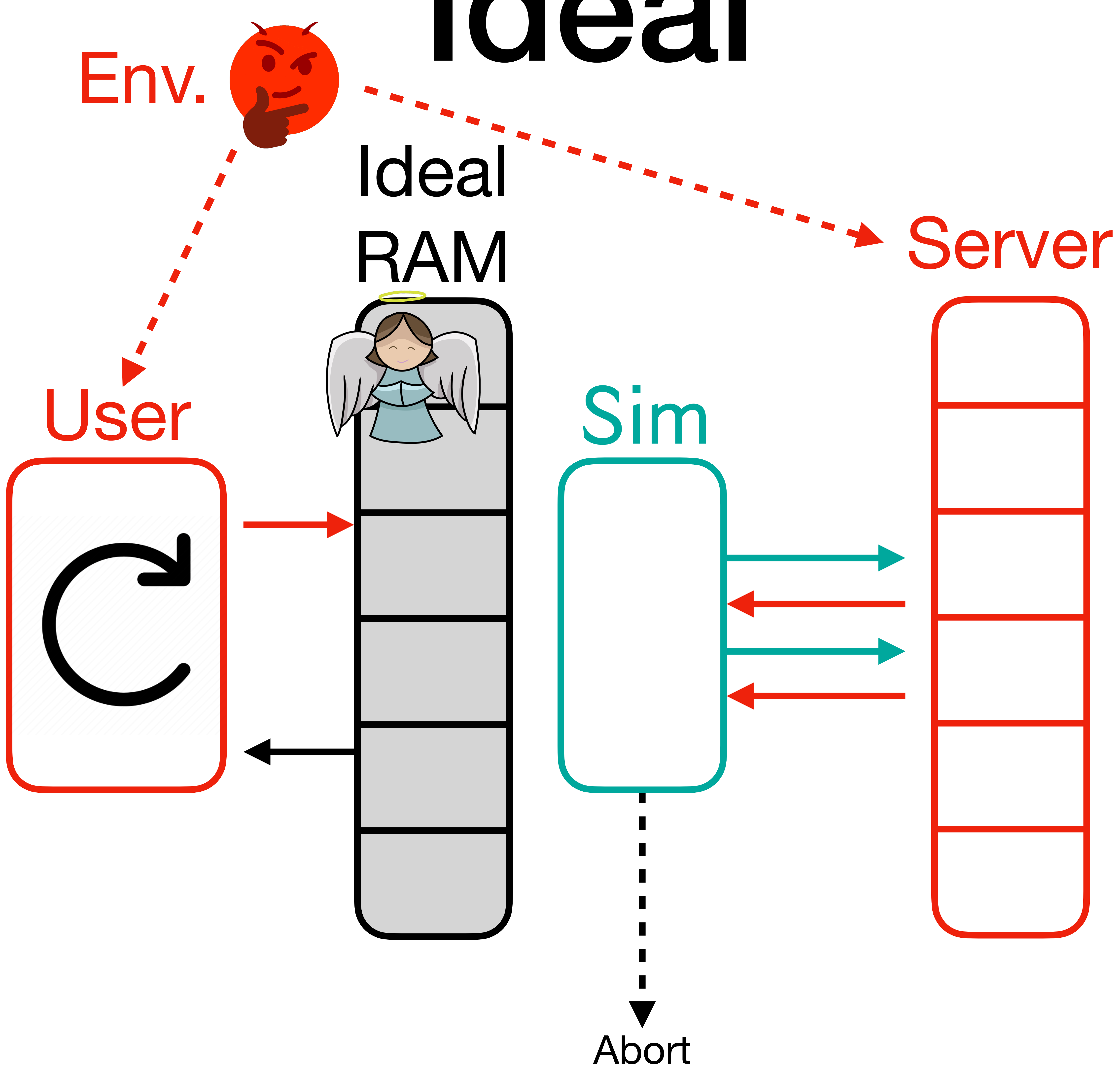




Our Definition: ORAM is *maliciously secure* if ∃Sim such that for all 🤔, Real $\approx_{\text{comp}}$ Ideal

Ideal





Our Definition: ORAM is *maliciously secure* if $\exists$Sim such that for all 🤔, Real $\approx_{\text{comp}}$ Ideal (and ORAM doesn't abort against an honest server).

Background: Hierarchical ORAM

Background: Hierarchical ORAM

- Many ORAM constructions, starting with [Ostrovsky ’90, Goldreich-Ostrovsky ’96] and including **OptORAMa** [AKLNPS ’21], follow the **hierarchical paradigm.**

Background: Hierarchical ORAM

- Many ORAM constructions, starting with [Ostrovsky ’90, Goldreich-Ostrovsky ’96] and including **OptORAMa** [AKLNPS ’21], follow the **hierarchical paradigm.**

- For each $i \in [\log_2(N)]$, there’s an *oblivious* hash table H_i of size 2^i.

Background: Hierarchical ORAM

- Many ORAM constructions, starting with [Ostrovsky ’90, Goldreich-Ostrovsky ’96] and including **OptORAMa** [AKLNPS ’21], follow the **hierarchical paradigm**.

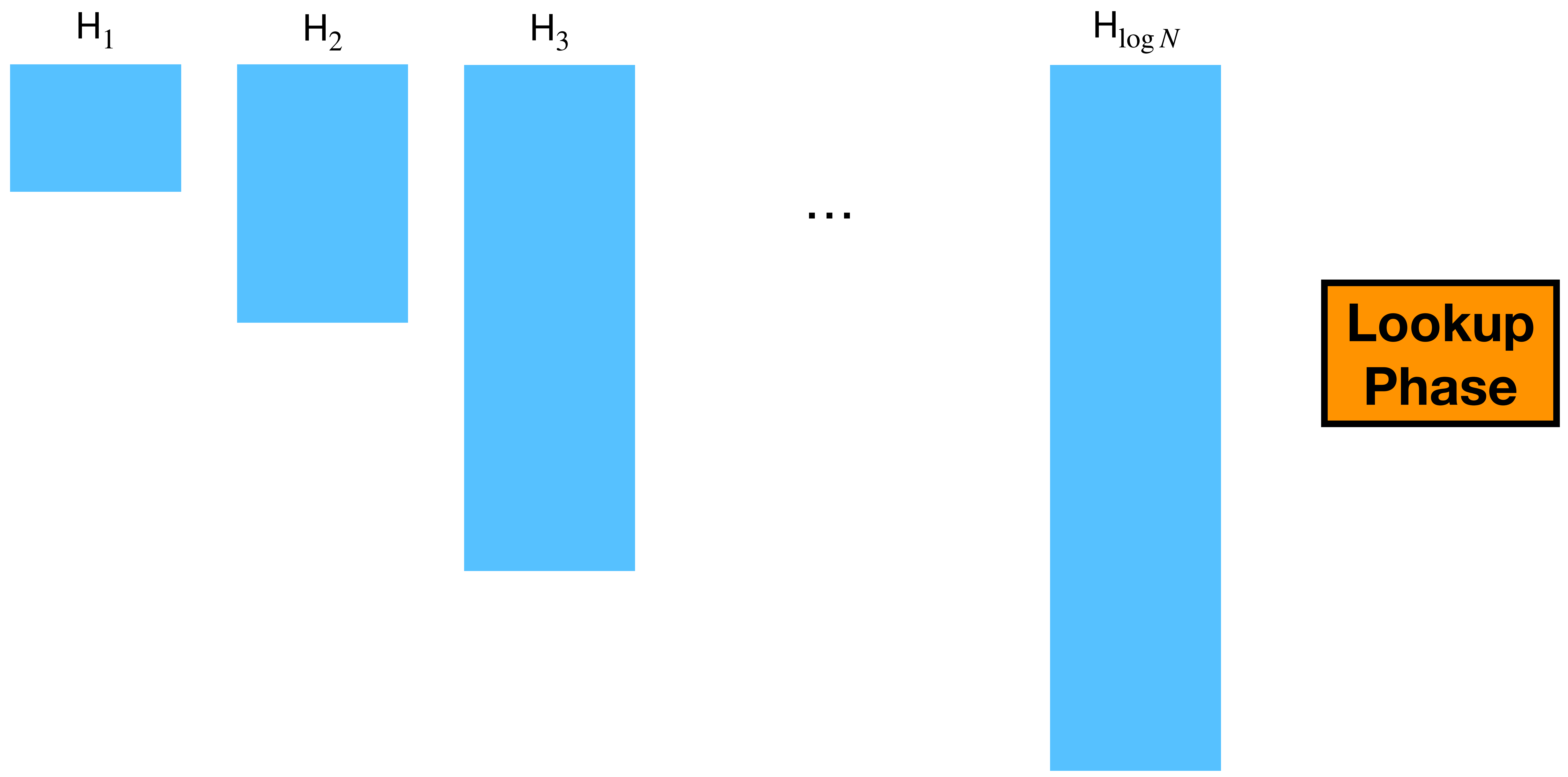
- For each $i \in [\log_2(N)]$, there’s an *oblivious* hash table H_i of size 2^i.

  - **Lookup Phase**: Given a query to addr, lookup addr in $\mathsf{H}_1, \mathsf{H}_2, \ldots$ until found. Lookup dummy elements for the subsequent tables, and write updated addr back to H_1.

Background: Hierarchical ORAM

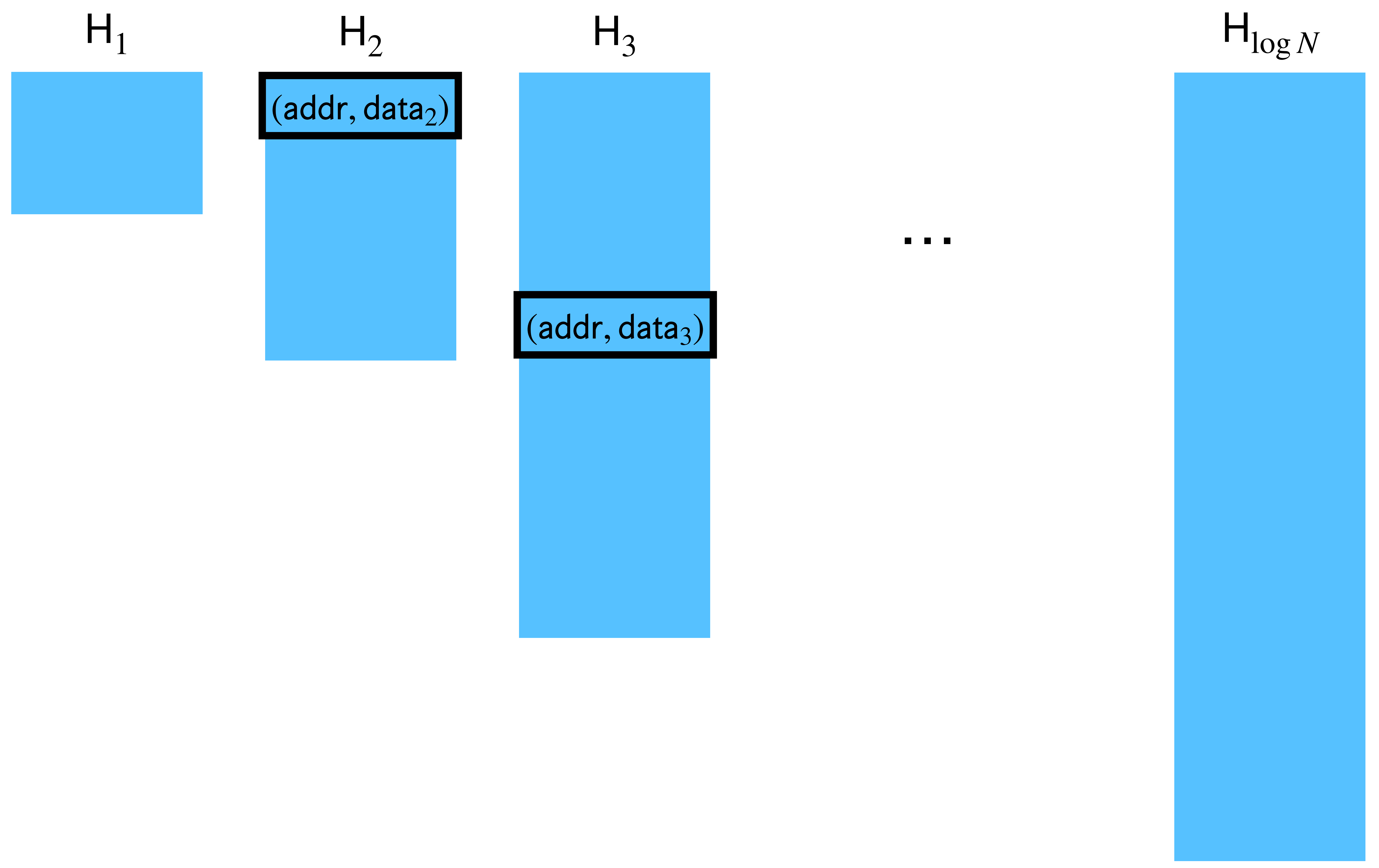
- Many ORAM constructions, starting with [Ostrovsky ’90, Goldreich-Ostrovsky ’96] and including **OptORAMa** [AKLNPS ’21], follow the **hierarchical paradigm**.
- For each $i \in [\log_2(N)]$, there’s an *oblivious* hash table H_i of size 2^i.
  - **Lookup Phase**: Given a query to addr, lookup addr in $\mathsf{H}_1, \mathsf{H}_2, \ldots$ until found. Lookup dummy elements for the subsequent tables, and write updated addr back to H_1.
  - **Rebuild Phase**: Every 2^i queries, obliviously merge $\mathsf{H}_1 \to \mathsf{H}_2 \to \cdots \to \mathsf{H}_{i+1}$ into new H_{i+1}, removing duplicate addresses by keeping the version from the smaller H_j.

Hierarchical Construction: Lookup



Hierarchical Construction: Lookup

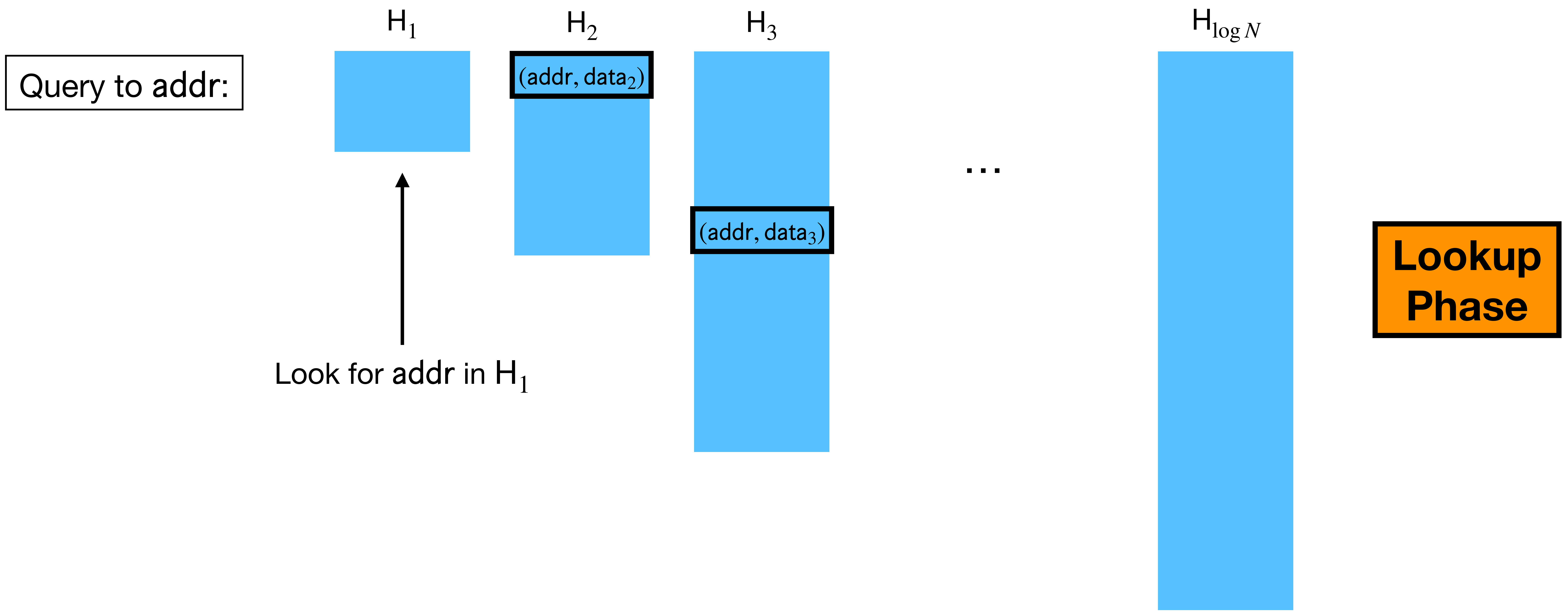
Query to addr:



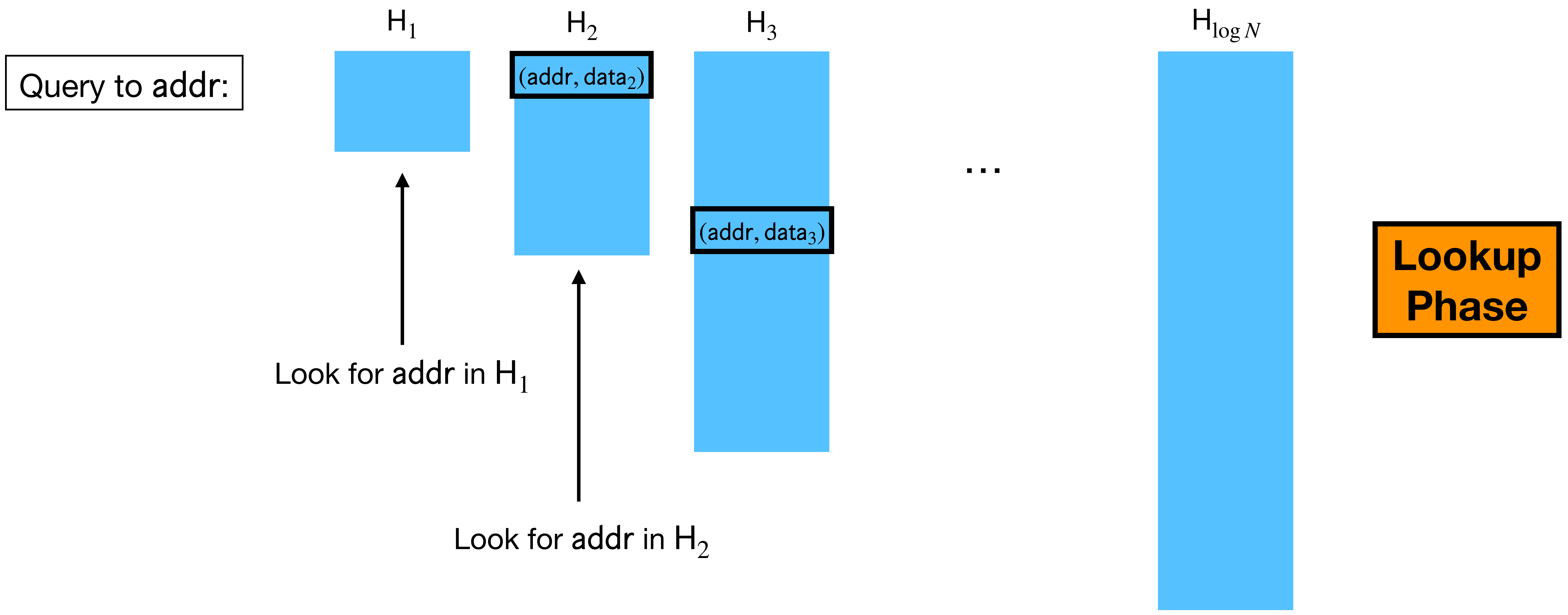
H_1
H_2
H_3
$H_{\log N}$
$(\text{addr}, \text{data}_2)$
$(\text{addr}, \text{data}_3)$
…

Lookup Phase

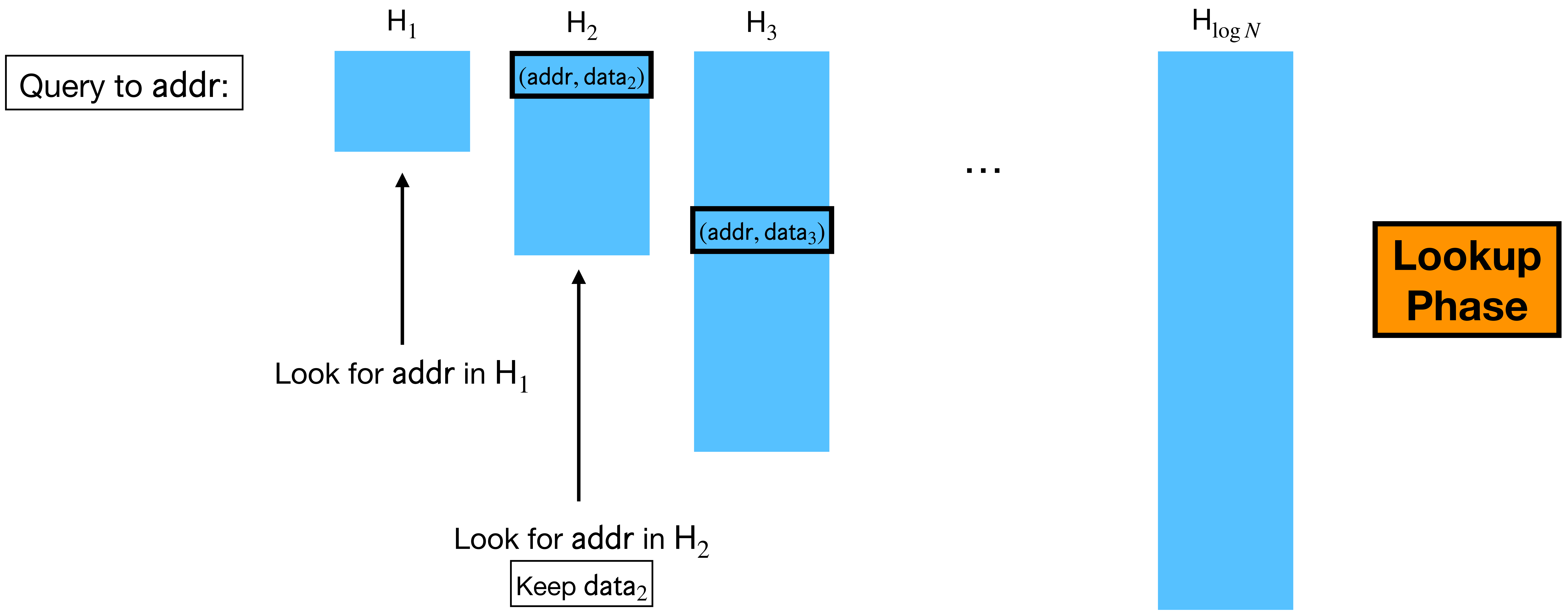
Hierarchical Construction: Lookup



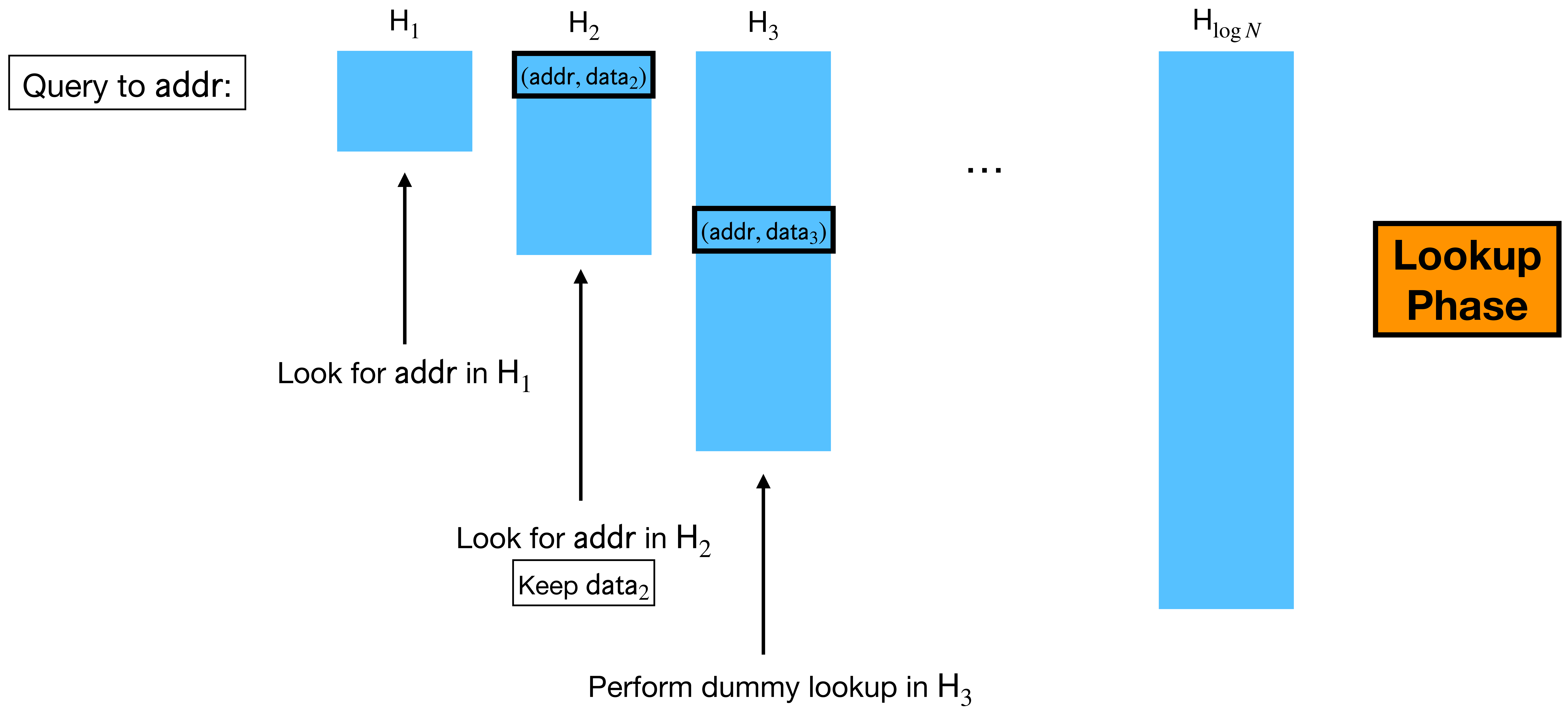
Hierarchical Construction: Lookup



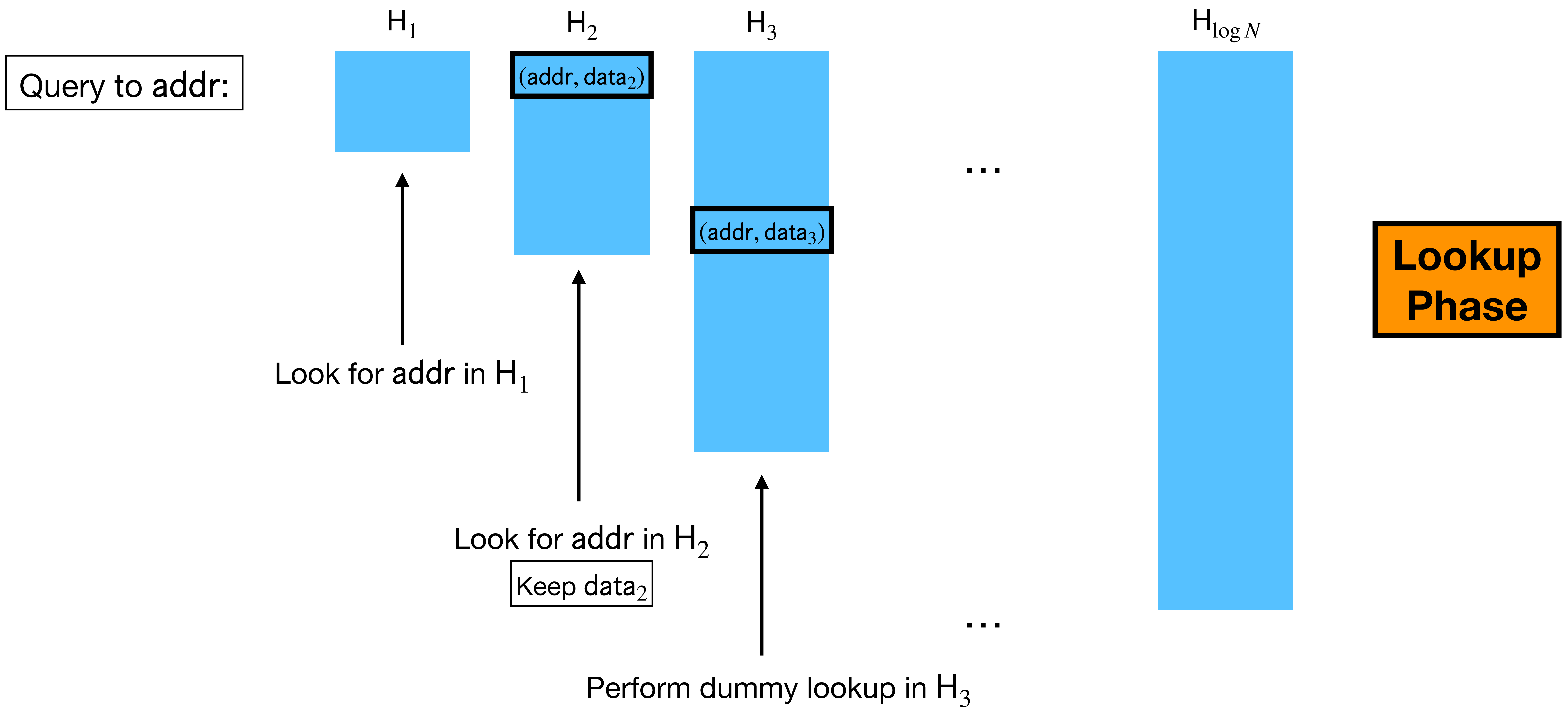
Hierarchical Construction: Lookup



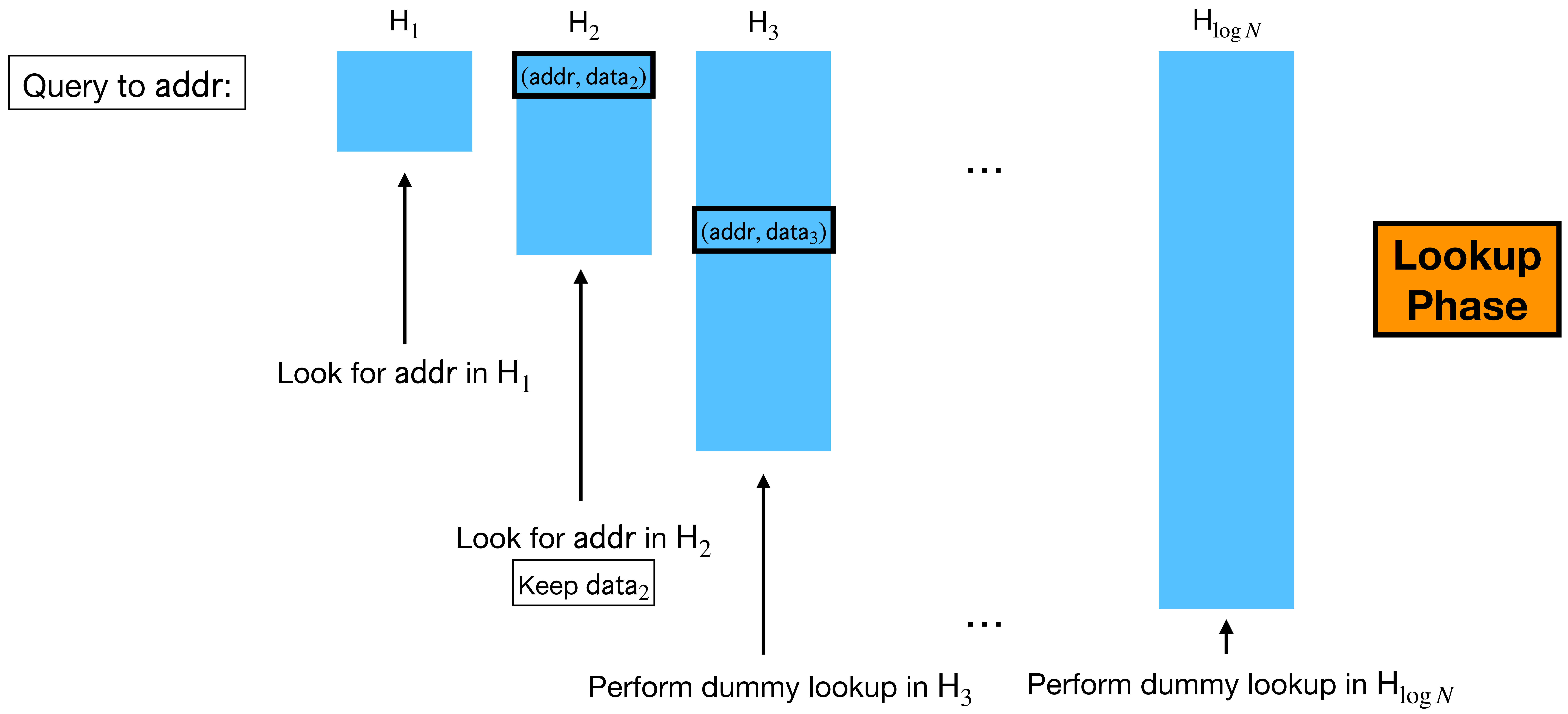
Hierarchical Construction: Lookup



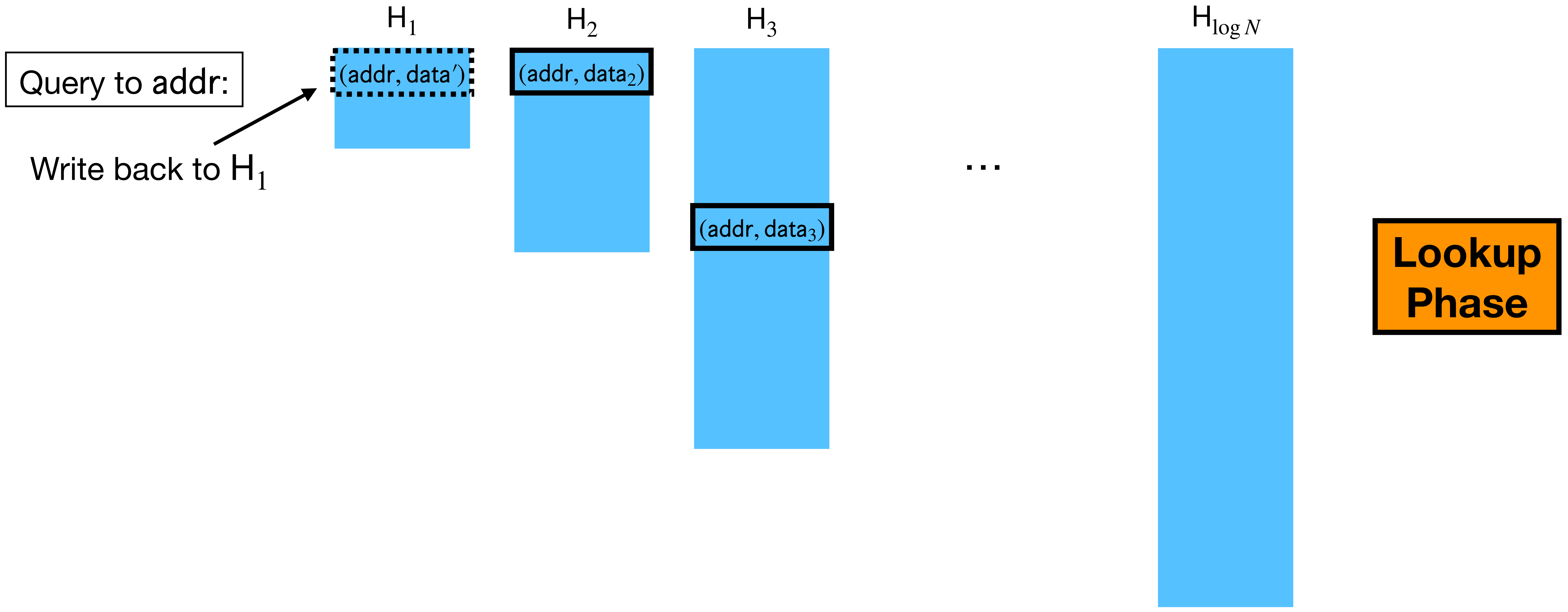
Hierarchical Construction: Lookup



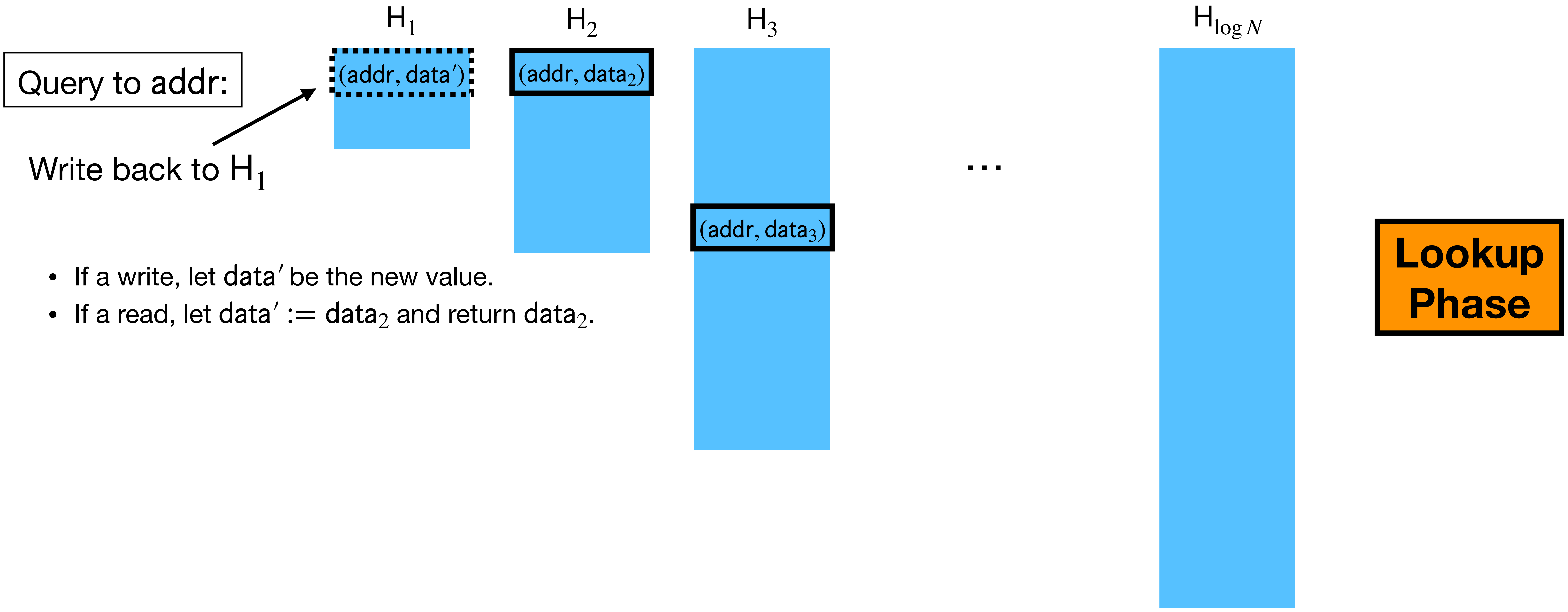
Hierarchical Construction: Lookup



Hierarchical Construction: Lookup

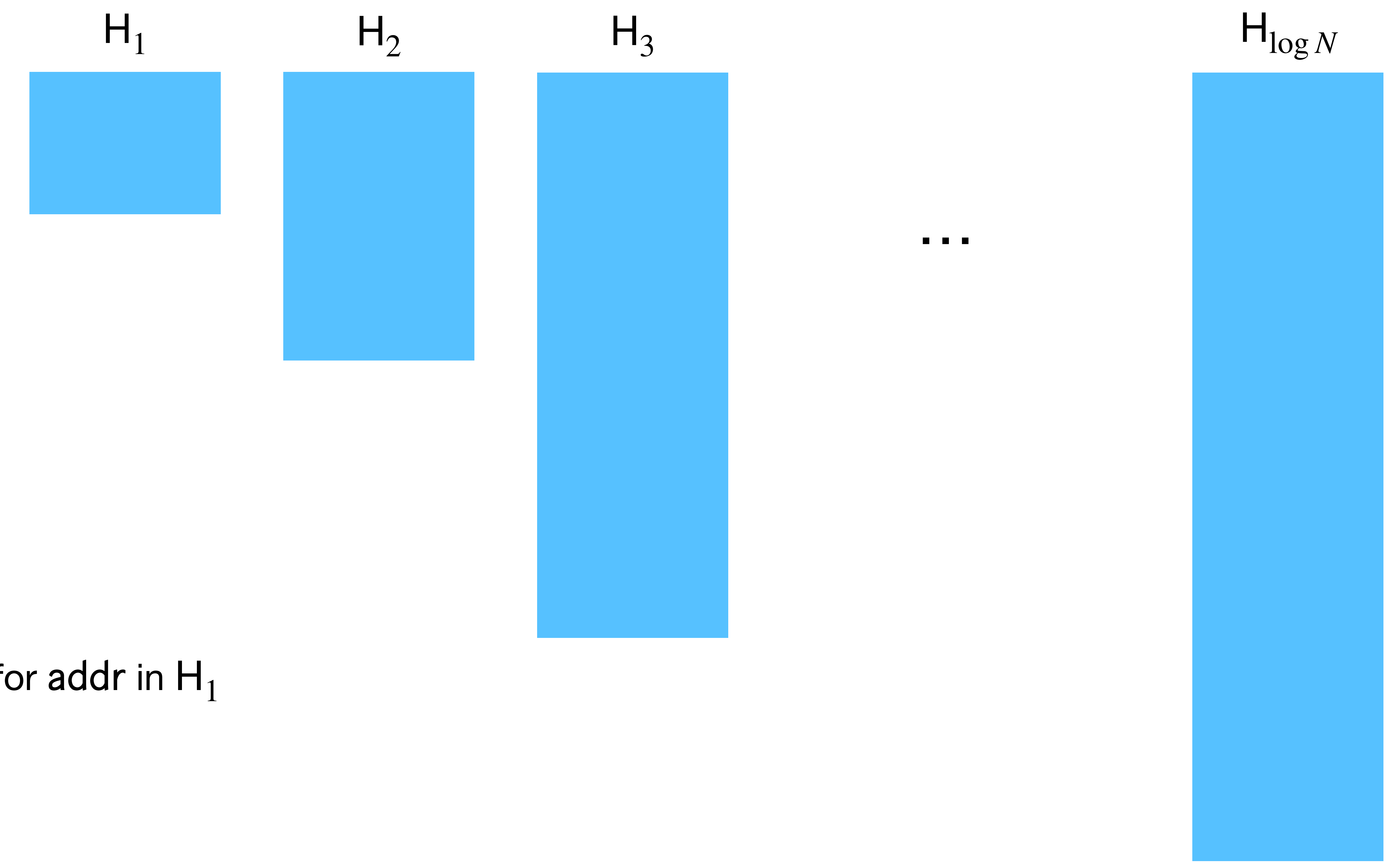


Hierarchical Construction: Lookup


Query to addr:
Write back to H_1
H_1
$(\mathsf{addr}, \mathsf{data}')$
H_2
$(\mathsf{addr}, \mathsf{data}_2)$
H_3
$(\mathsf{addr}, \mathsf{data}_3)$
$\cdots$
$H_{\log N}$
Lookup Phase

- If a write, let data' be the new value.
- If a read, let $\mathsf{data}' := \mathsf{data}_2$ and return data_2.

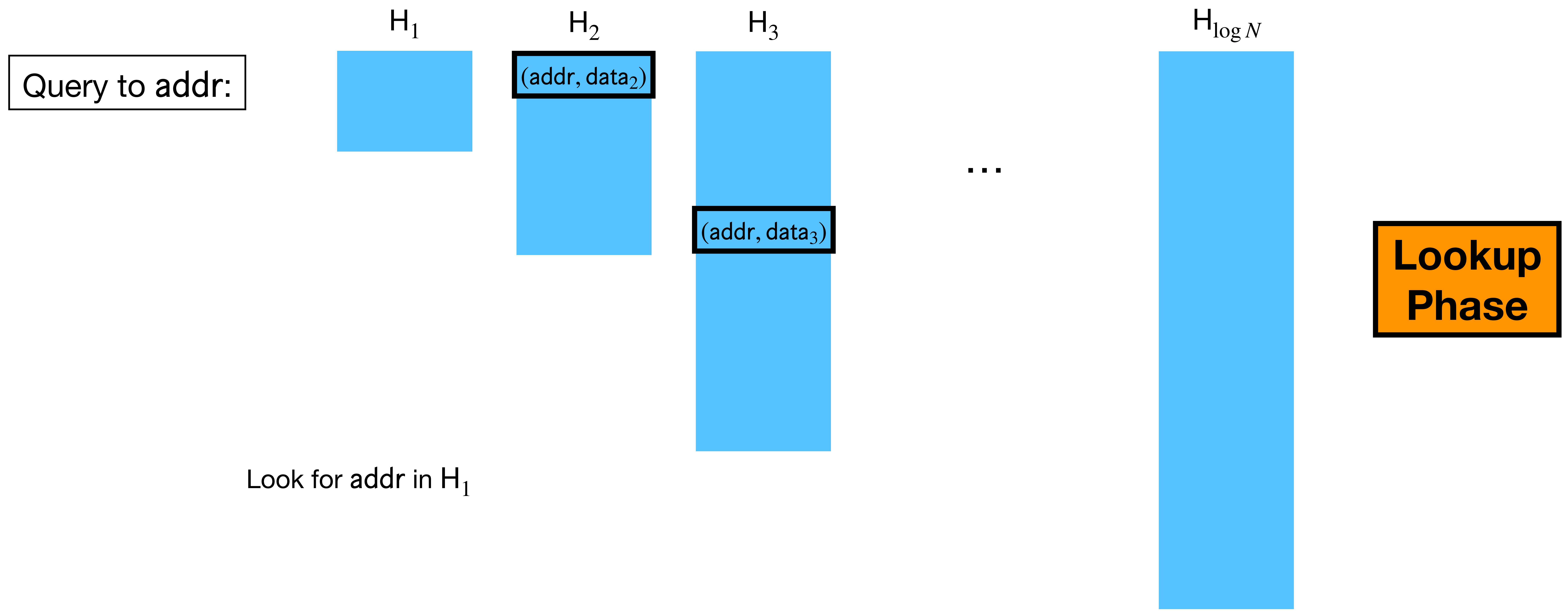
Hierarchical Construction: Lookup



Lookup Phase

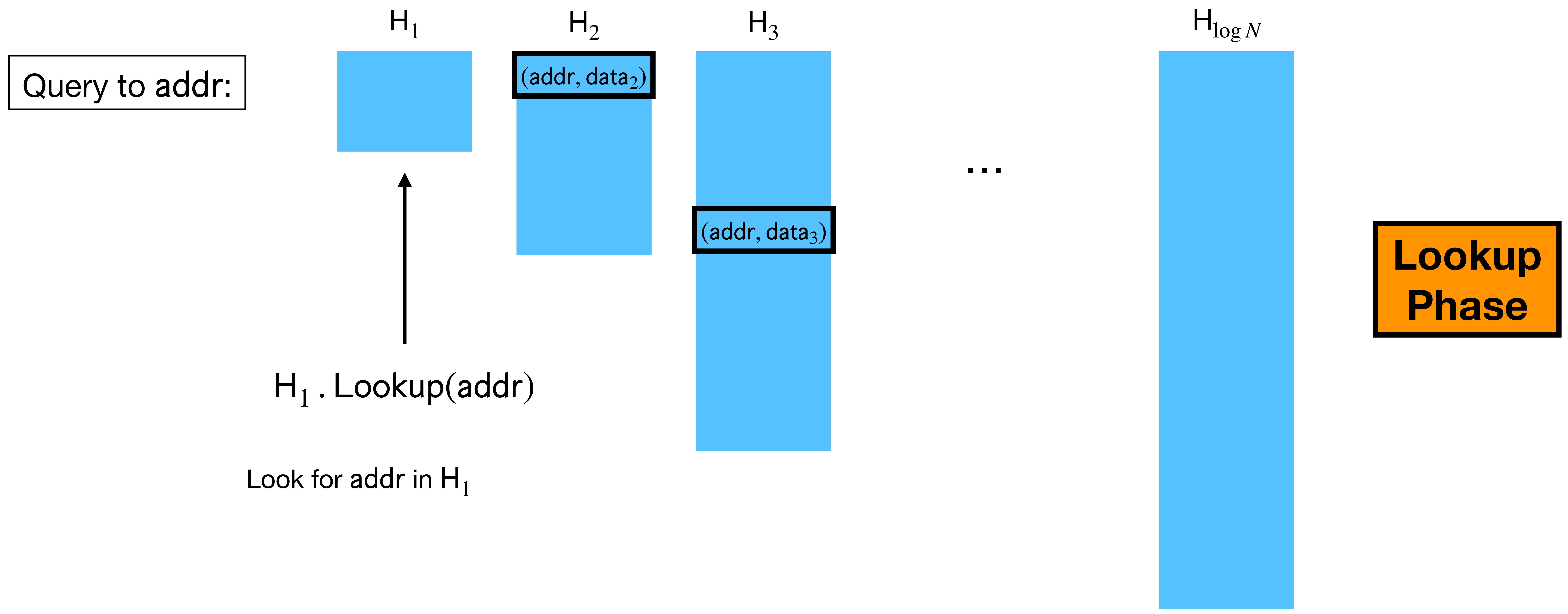
Look for addr in H_1

Hierarchical Construction: Lookup

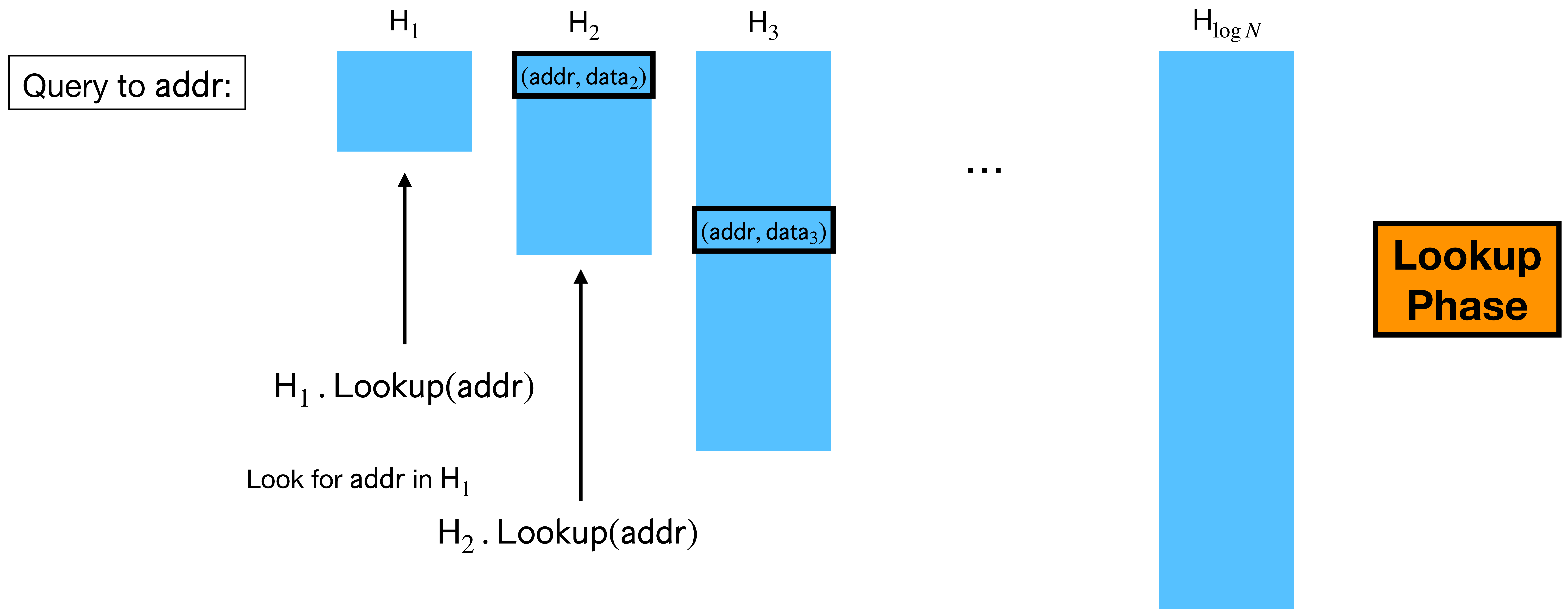

Query to addr:
H_1
H_2
H_3
$H_{\log N}$
$(\text{addr}, \text{data}_2)$
$(\text{addr}, \text{data}_3)$
…
Lookup Phase

Look for addr in H_1

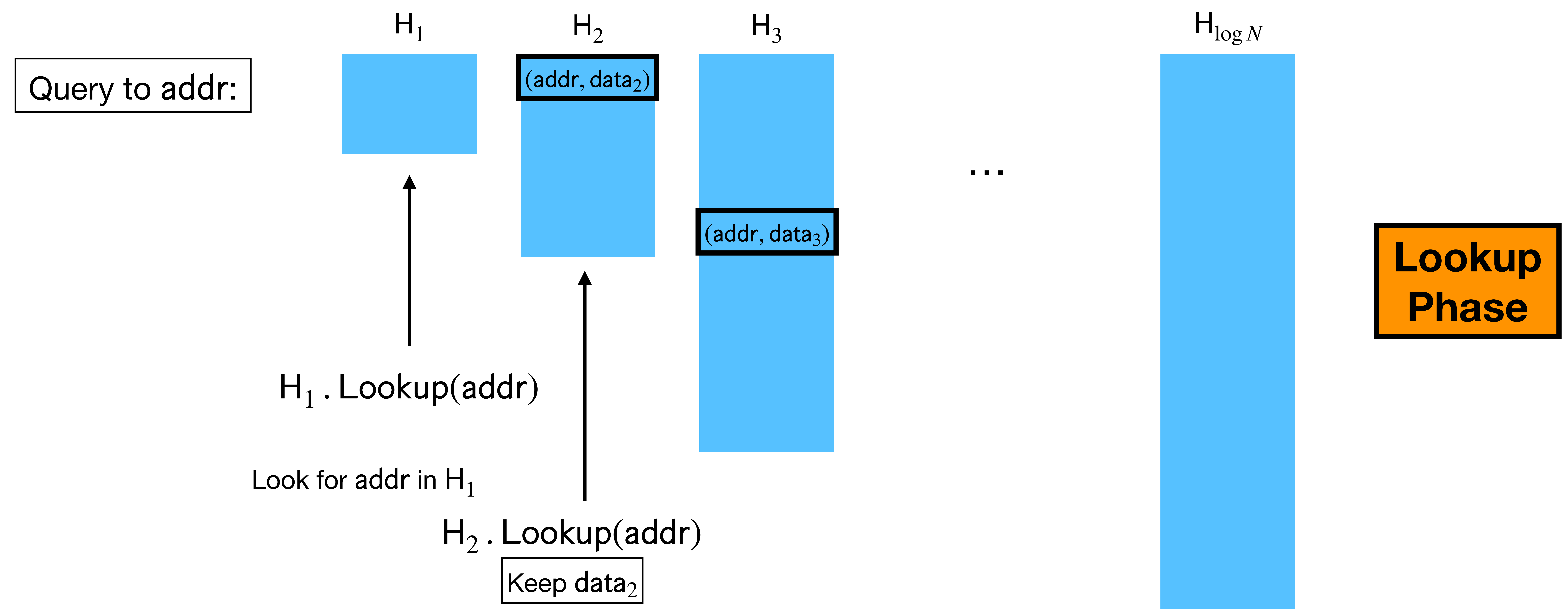
Hierarchical Construction: Lookup



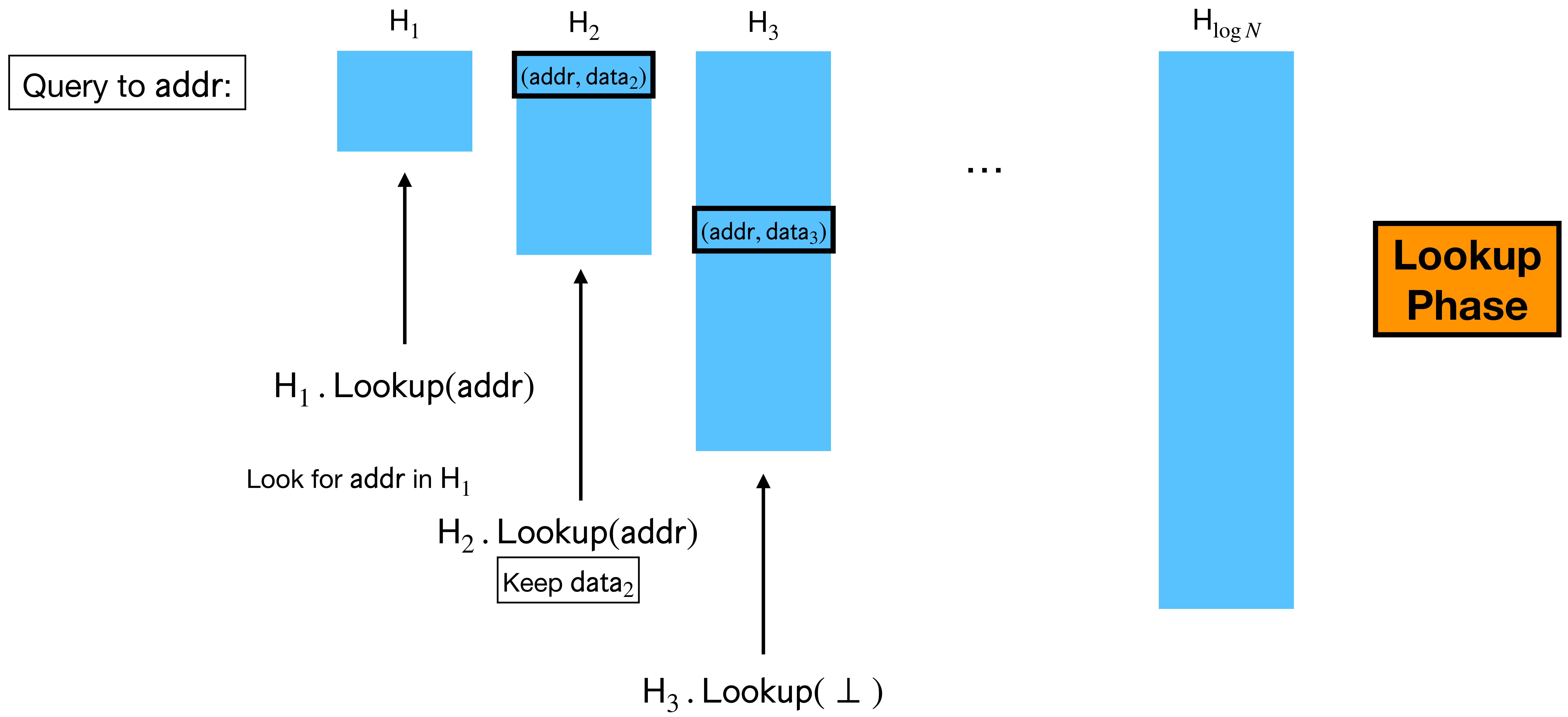
Hierarchical Construction: Lookup



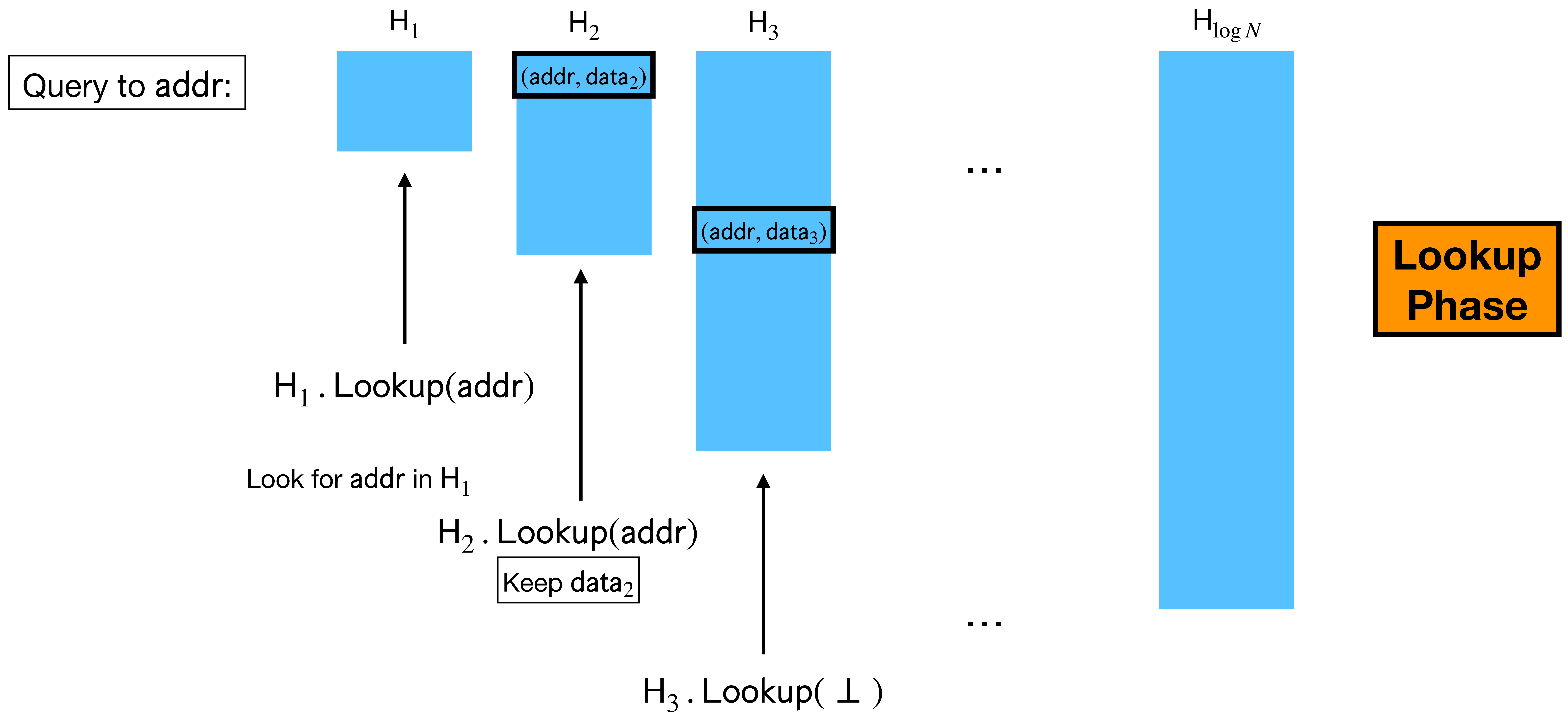
Hierarchical Construction: Lookup



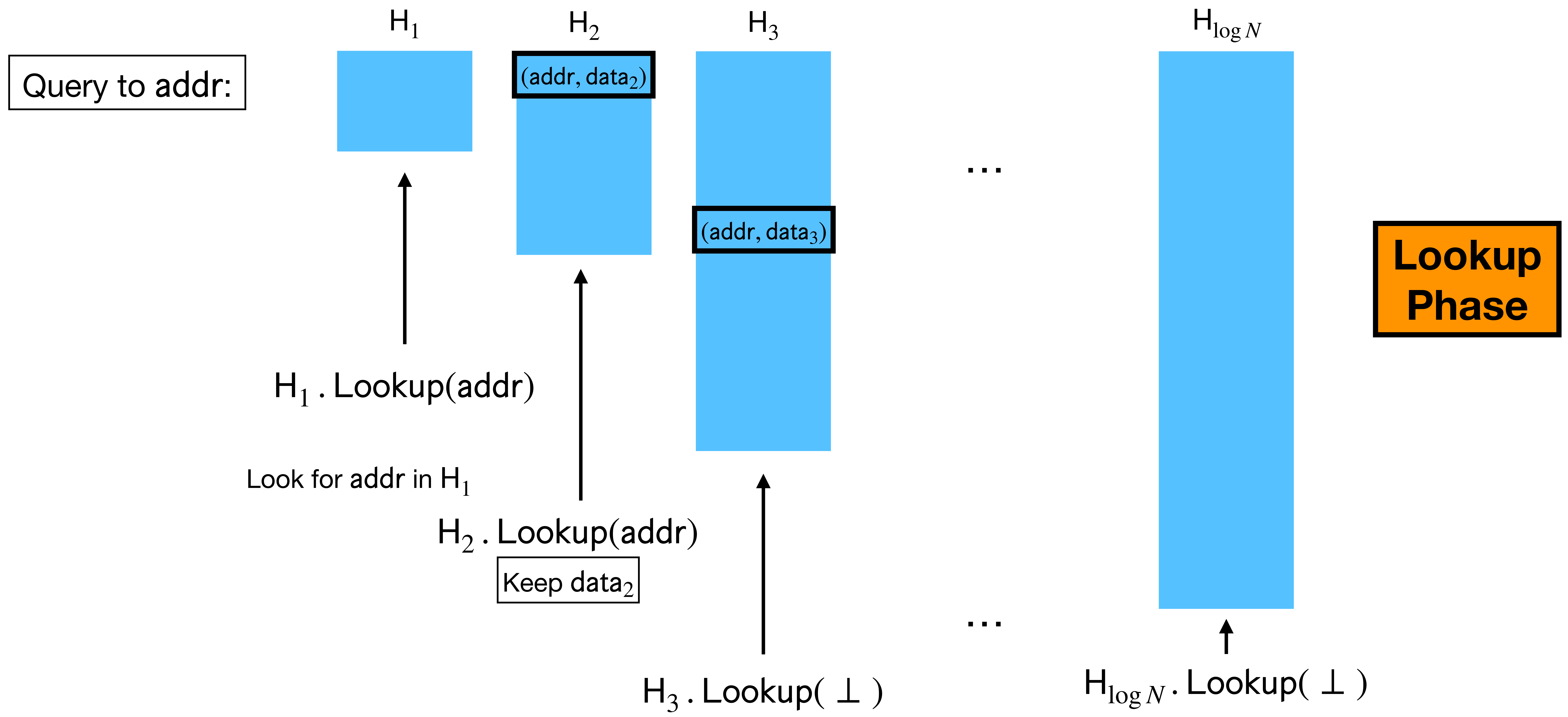
Hierarchical Construction: Lookup



Hierarchical Construction: Lookup

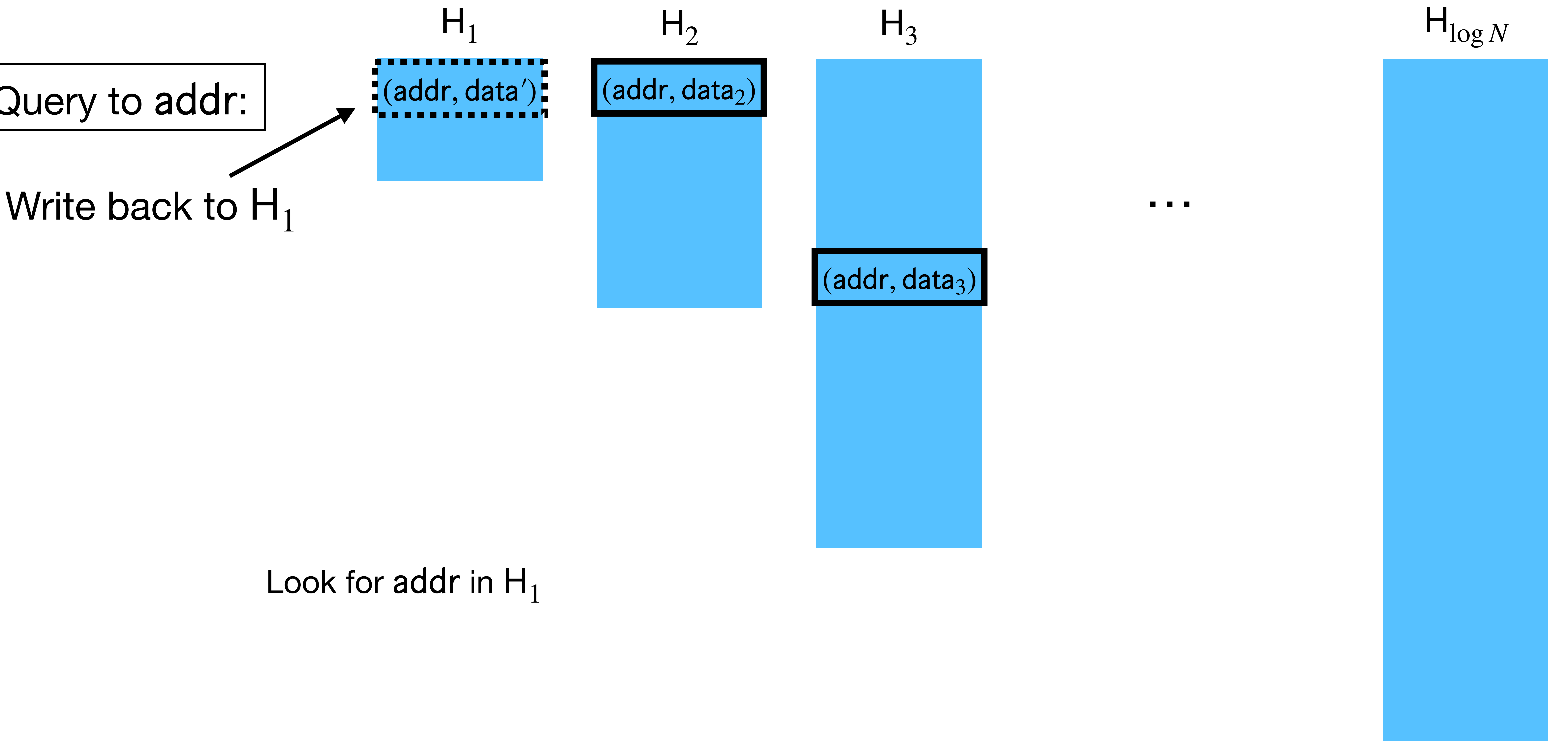

Query to addr:
H_1
H_2
H_3
$H_{\log N}$
$(\text{addr}, \text{data}_2)$
$(\text{addr}, \text{data}_3)$
...
$H_1.\text{Lookup}(\text{addr})$
Look for addr in H_1
$H_2.\text{Lookup}(\text{addr})$
Keep data_2
$H_3.\text{Lookup}(\perp)$
...
Lookup Phase

Hierarchical Construction: Lookup



Hierarchical Construction: Lookup

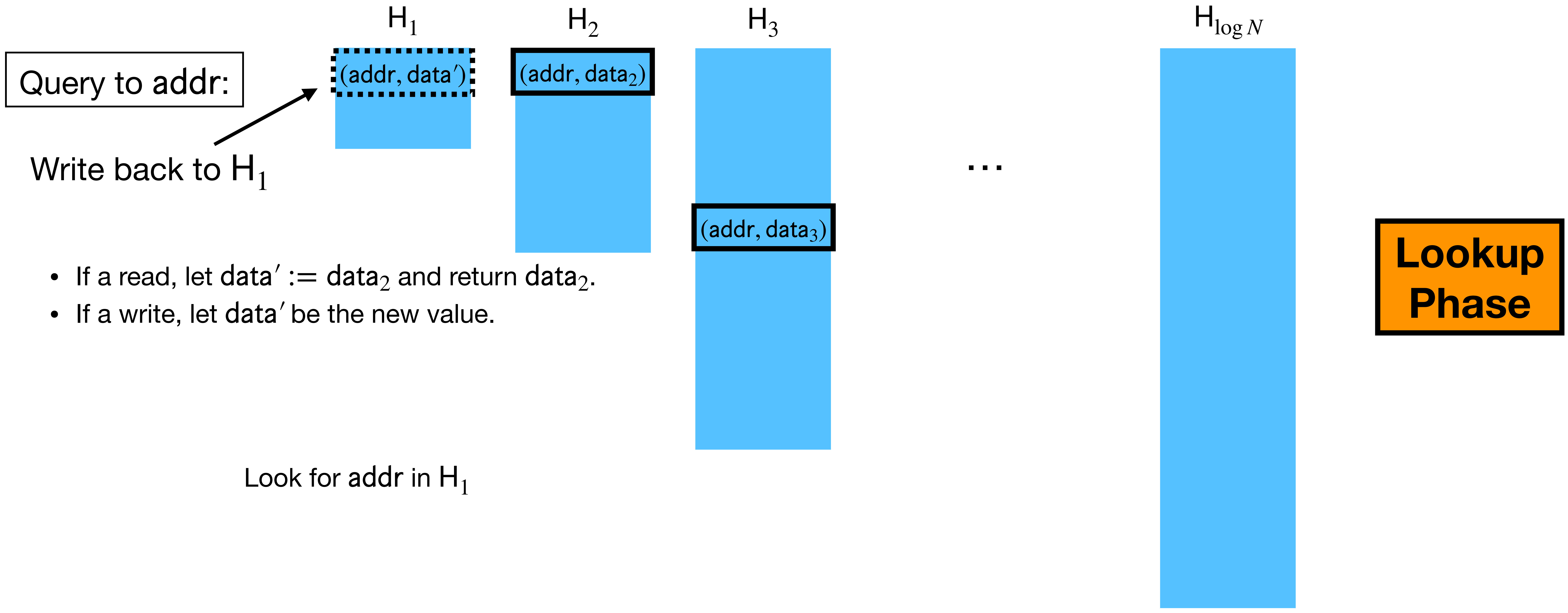
Query to addr:



Look for addr in H_1

Lookup Phase

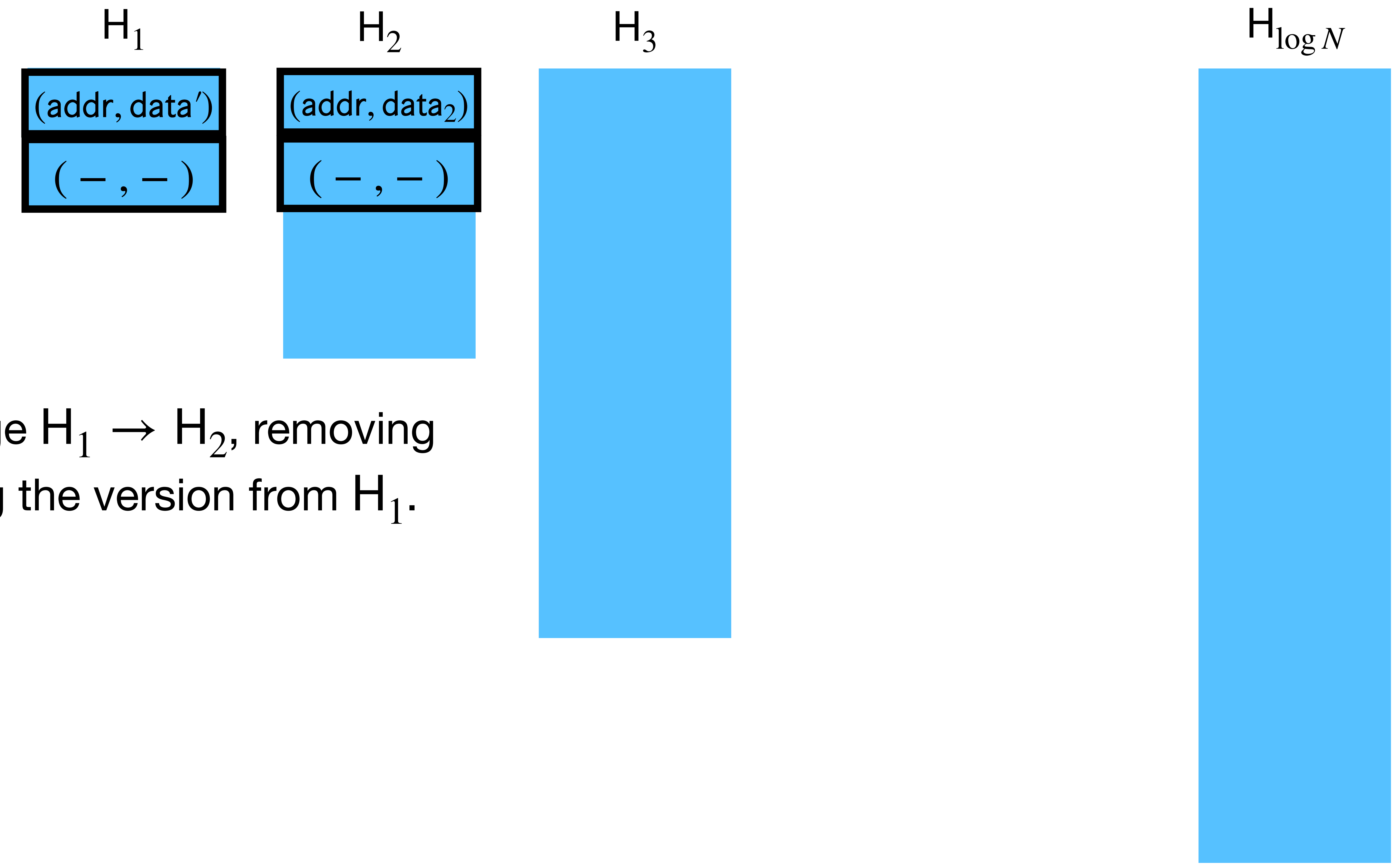
Hierarchical Construction: Lookup


Query to addr:
H_1
H_2
H_3
$H_{\log N}$
$(\text{addr}, \text{data}')$
$(\text{addr}, \text{data}_2)$
$(\text{addr}, \text{data}_3)$
Write back to H_1
$\cdots$
Lookup Phase

- If a read, let $\text{data}' := \text{data}_2$ and return data_2.
- If a write, let data' be the new value.

Look for addr in H_1

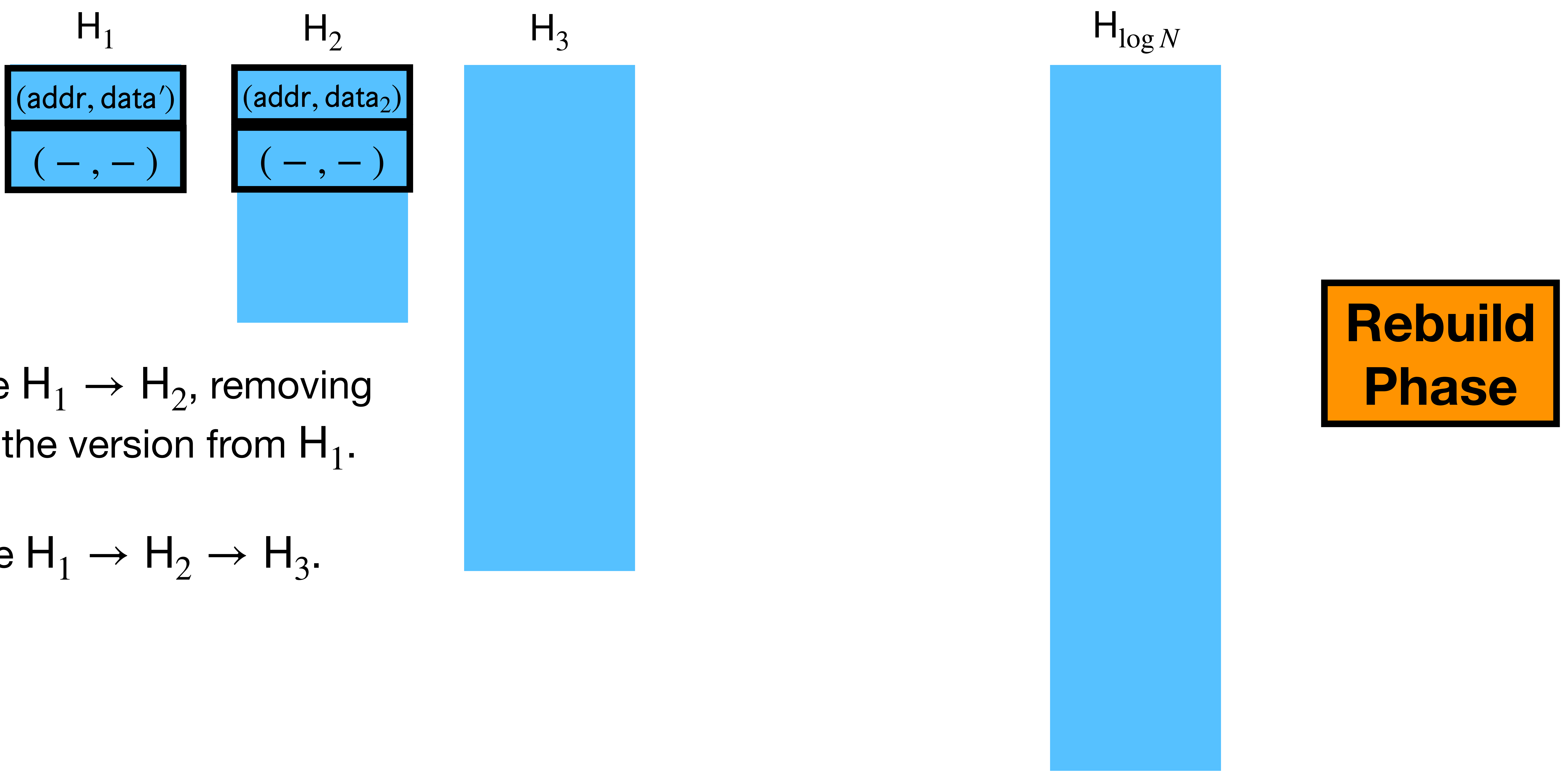
Hierarchical Construction: Rebuild



- Every 2 queries, merge $\mathsf{H}_1 \to \mathsf{H}_2$, removing duplicates by keeping the version from H_1.

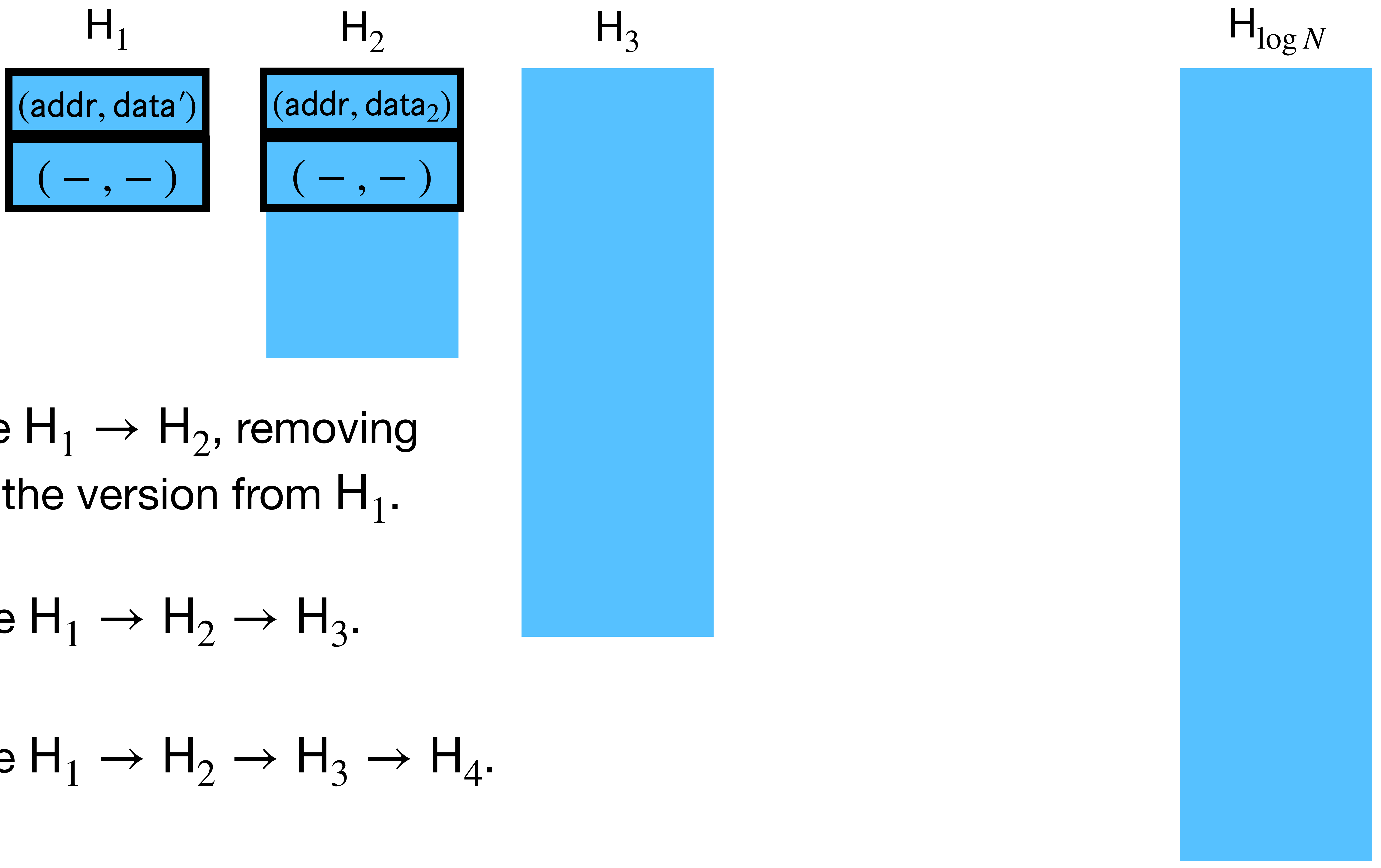
Rebuild Phase

Hierarchical Construction: Rebuild



- Every 2 queries, merge $H_1 \rightarrow H_2$, removing duplicates by keeping the version from H_1.
- Every 4 queries, merge $H_1 \rightarrow H_2 \rightarrow H_3$.

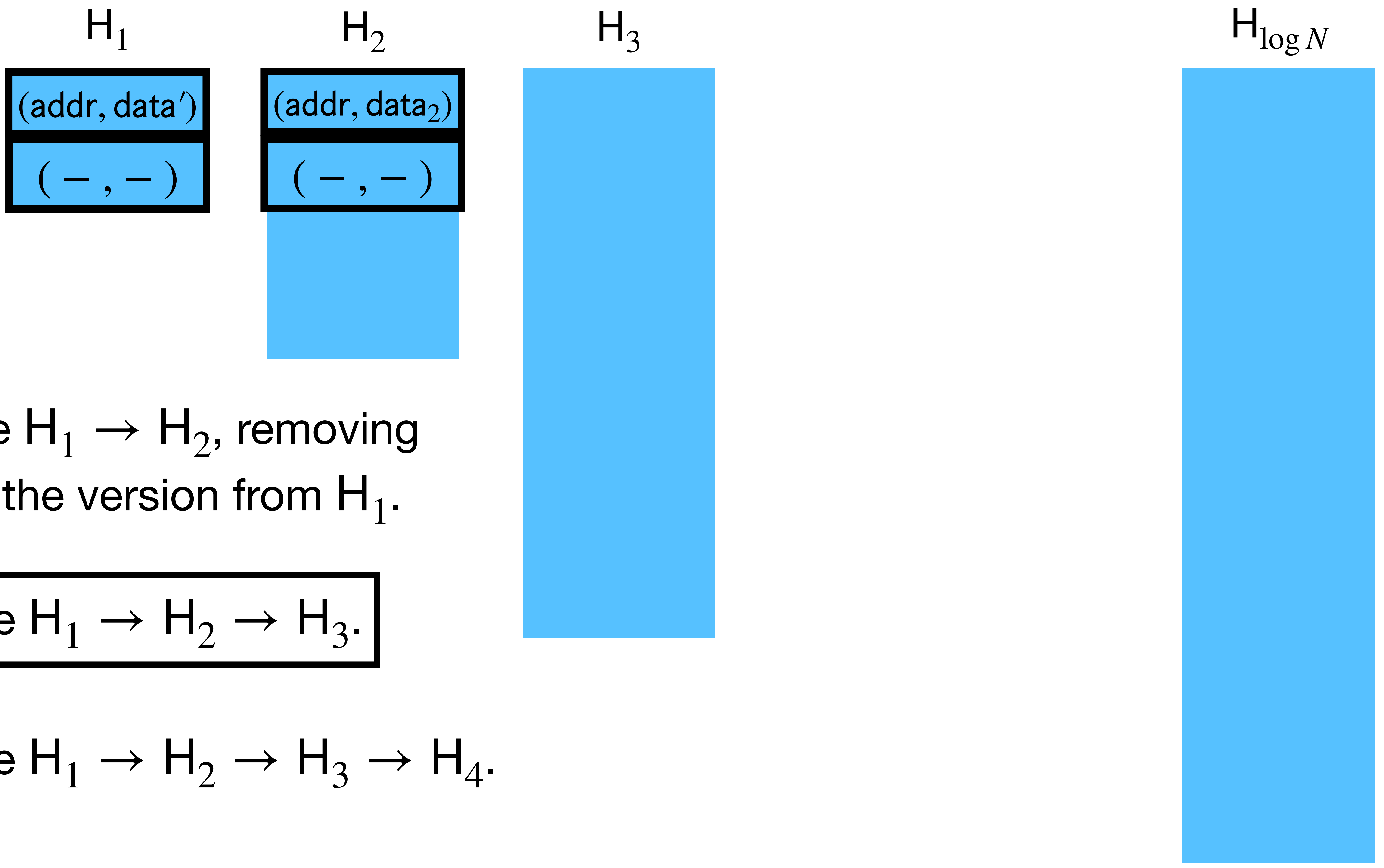
Hierarchical Construction: Rebuild



Rebuild Phase

- Every 2 queries, merge $H_1 \to H_2$, removing duplicates by keeping the version from H_1.
- Every 4 queries, merge $H_1 \to H_2 \to H_3$.
- Every 8 queries, merge $H_1 \to H_2 \to H_3 \to H_4$.
- …

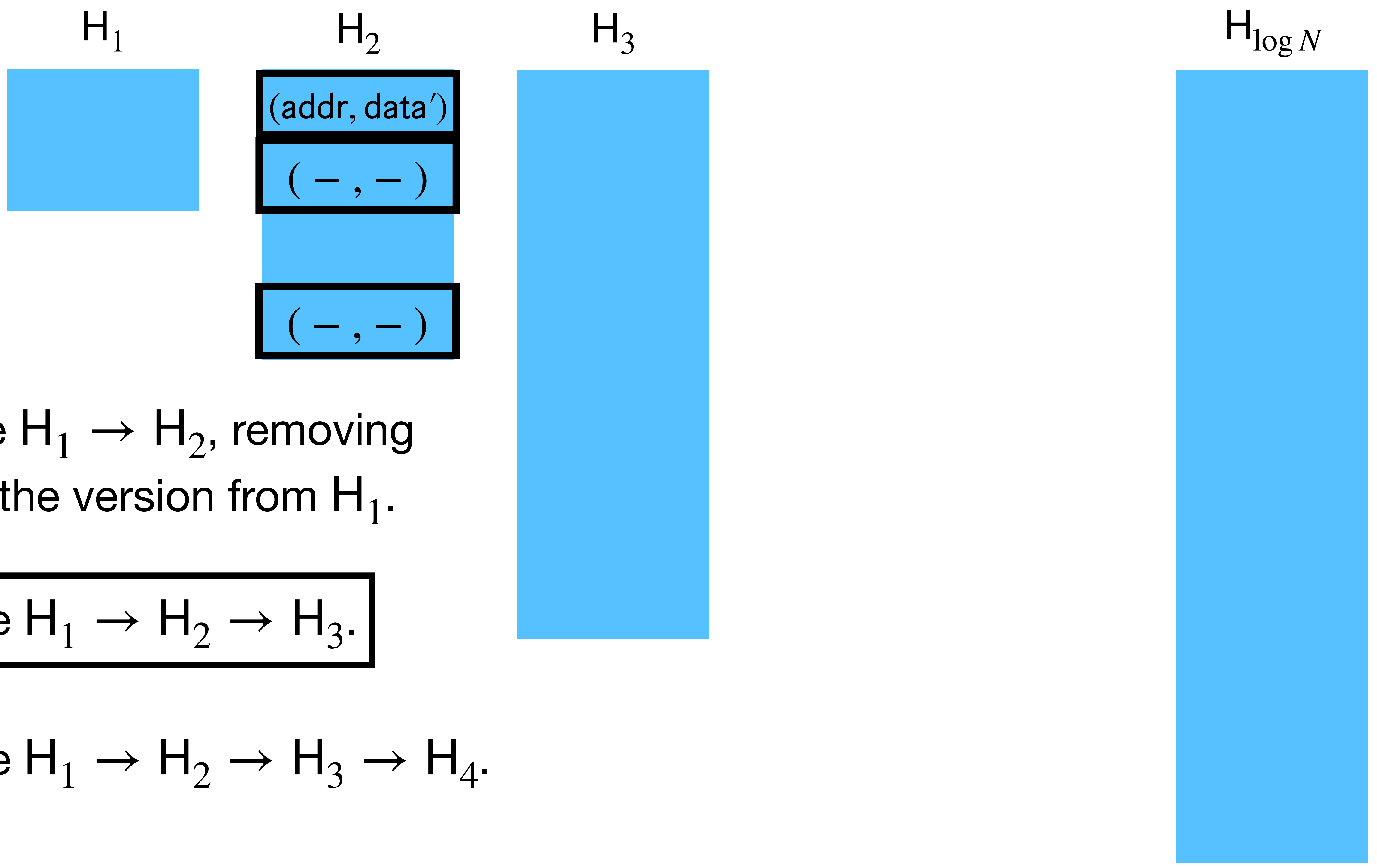
Hierarchical Construction: Rebuild



Rebuild Phase

- Every 2 queries, merge $H_1 \to H_2$, removing duplicates by keeping the version from H_1.
- Every 4 queries, merge $H_1 \to H_2 \to H_3$.
- Every 8 queries, merge $H_1 \to H_2 \to H_3 \to H_4$.
- …

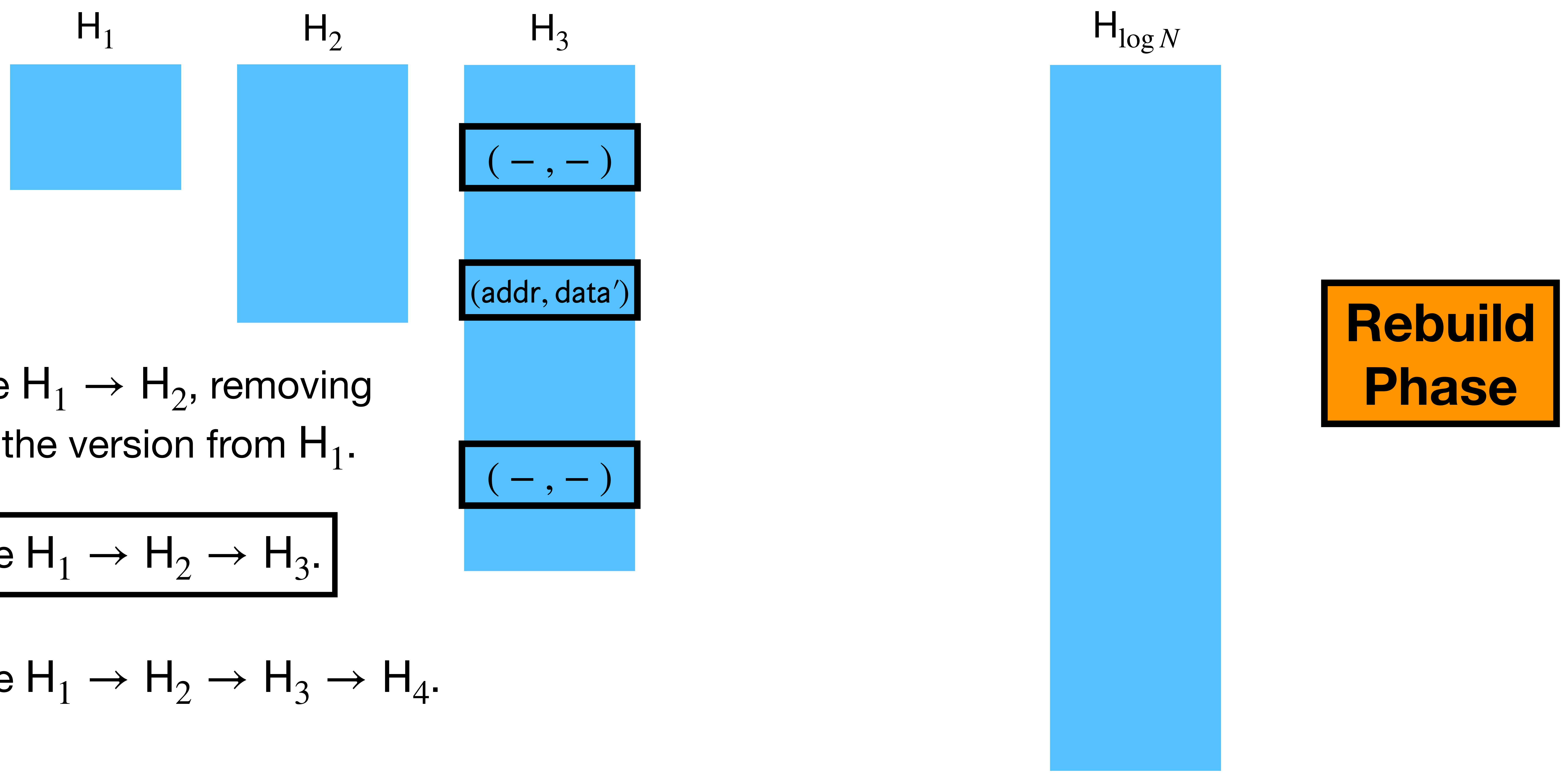
Hierarchical Construction: Rebuild



Rebuild Phase

- Every 2 queries, merge $H_1 \rightarrow H_2$, removing duplicates by keeping the version from H_1.
- Every 4 queries, merge $H_1 \rightarrow H_2 \rightarrow H_3$.
- Every 8 queries, merge $H_1 \rightarrow H_2 \rightarrow H_3 \rightarrow H_4$.
- …

Hierarchical Construction: Rebuild



- Every 2 queries, merge $H_1 \to H_2$, removing duplicates by keeping the version from H_1.
- Every 4 queries, merge $H_1 \to H_2 \to H_3$.
- Every 8 queries, merge $H_1 \to H_2 \to H_3 \to H_4$.
- …

Hierarchical Efficiency

Hierarchical Efficiency

- Each H_i lookup takes $O(1)$ $\widehat{\text{query}}$'s using oblivious cuckoo hashing.*

[Goodrich-Mitzenmacher '11]

*Ignoring cuckoo hash-table stashes.

Hierarchical Efficiency

- Each H_i lookup takes $O(1)$ $\widehat{\mathsf{query}}$'s using oblivious cuckoo hashing.*
    - Iterating over $i \in [\log N]$, the Lookup Phase takes $O(\log N)$ $\widehat{\mathsf{query}}$'s.

Hierarchical Efficiency

- Each H_i lookup takes $O(1)$ $\widehat{\mathsf{query}}$'s using oblivious cuckoo hashing.*
    - Iterating over $i \in [\log N]$, the Lookup Phase takes $O(\log N)$ $\widehat{\mathsf{query}}$'s.
- Suppose the Rebuild Phase happening every 2^i steps takes $T\left(2^i\right)$ $\widehat{\mathsf{query}}$'s.

Hierarchical Efficiency

- Each H_i lookup takes $O(1)$ $\widehat{\text{query}}$'s using oblivious cuckoo hashing.*
    - Iterating over $i \in [\log N]$, the Lookup Phase takes $O(\log N)$ $\widehat{\text{query}}$'s.
- Suppose the Rebuild Phase happening every 2^i steps takes $T\left(2^i\right)$ $\widehat{\text{query}}$'s.
- Amortized ORAM overhead over $\geq N$ queries:

$$O\left(\log N\right) + \sum_{i \in [\log N]} \frac{1}{2^i} \cdot T\left(2^i\right)$$

Hierarchical Efficiency

- Each H_i lookup takes $O(1)$ $\widehat{\text{query}}$'s using oblivious cuckoo hashing.*
  - Iterating over $i \in [\log N]$, the Lookup Phase takes $O(\log N)$ $\widehat{\text{query}}$'s.
- Suppose the Rebuild Phase happening every 2^i steps takes $T\left(2^i\right)$ $\widehat{\text{query}}$'s.
- Amortized ORAM overhead over $\geq N$ queries:

$$O\left(\log N\right) + \sum_{i \in [\log N]} \frac{1}{2^i} \cdot T\left(2^i\right)$$

Lookup

Hierarchical Efficiency

- Each H_i lookup takes $O(1)$ $\widehat{\text{query}}$'s using oblivious cuckoo hashing.*
  - Iterating over $i \in [\log N]$, the Lookup Phase takes $O(\log N)$ $\widehat{\text{query}}$'s.
- Suppose the Rebuild Phase happening every 2^i steps takes $T\left(2^i\right)$ $\widehat{\text{query}}$'s.
- Amortized ORAM overhead over $\geq N$ queries:

$$\underbrace{O\left(\log N\right)}_{\text{Lookup}} + \underbrace{\sum_{i \in [\log N]} \frac{1}{2^i} \cdot T\left(2^i\right)}_{\text{Rebuild}}$$

Hierarchical Efficiency

- Each H_i lookup takes $O(1)$ $\widehat{\text{query}}$'s using oblivious cuckoo hashing.*
  - Iterating over $i \in [\log N]$, the Lookup Phase takes $O(\log N)$ $\widehat{\text{query}}$'s.
- Suppose the Rebuild Phase happening every 2^i steps takes $T\left(2^i\right)$ $\widehat{\text{query}}$'s.
- Amortized ORAM overhead over $\geq N$ queries:

$$O\left(\log N\right) + \sum_{i \in [\log N]} \frac{1}{2^i} \cdot T\left(2^i\right)$$

- If $T\left(2^i\right) = O\left(2^i\right)$, then this becomes $O(\log N)$! [**OptORAMa**, AKLNPS '20]

Hierarchical Efficiency

- Each H_i lookup takes $\boxed{O(1)}$ $\widehat{\text{query}}$'s using oblivious cuckoo hashing.*
    - Iterating over $i \in [\log N]$, the Lookup Phase takes $O(\log N)$ $\widehat{\text{query}}$'s.
- Suppose the Rebuild Phase happening every 2^i steps takes $T\left(2^i\right)$ $\widehat{\text{query}}$'s.
- Amortized ORAM overhead over $\geq N$ queries:

$$O\left(\log N\right) + \sum_{i \in [\log N]} \frac{1}{2^i} \cdot T\left(2^i\right)$$

- If $T\left(2^i\right) = \boxed{O\left(2^i\right)}$, then this becomes $O(\log N)$! [**OptORAMa**, AKLNPS '20]

Quite difficult! Long line of work to get this efficiency.

Replay Attack for Hierarchical Framework

Replay Attack for Hierarchical Framework

- **Key fact**: Oblivious hash tables are oblivious only if lookups are *non-recurrent*.

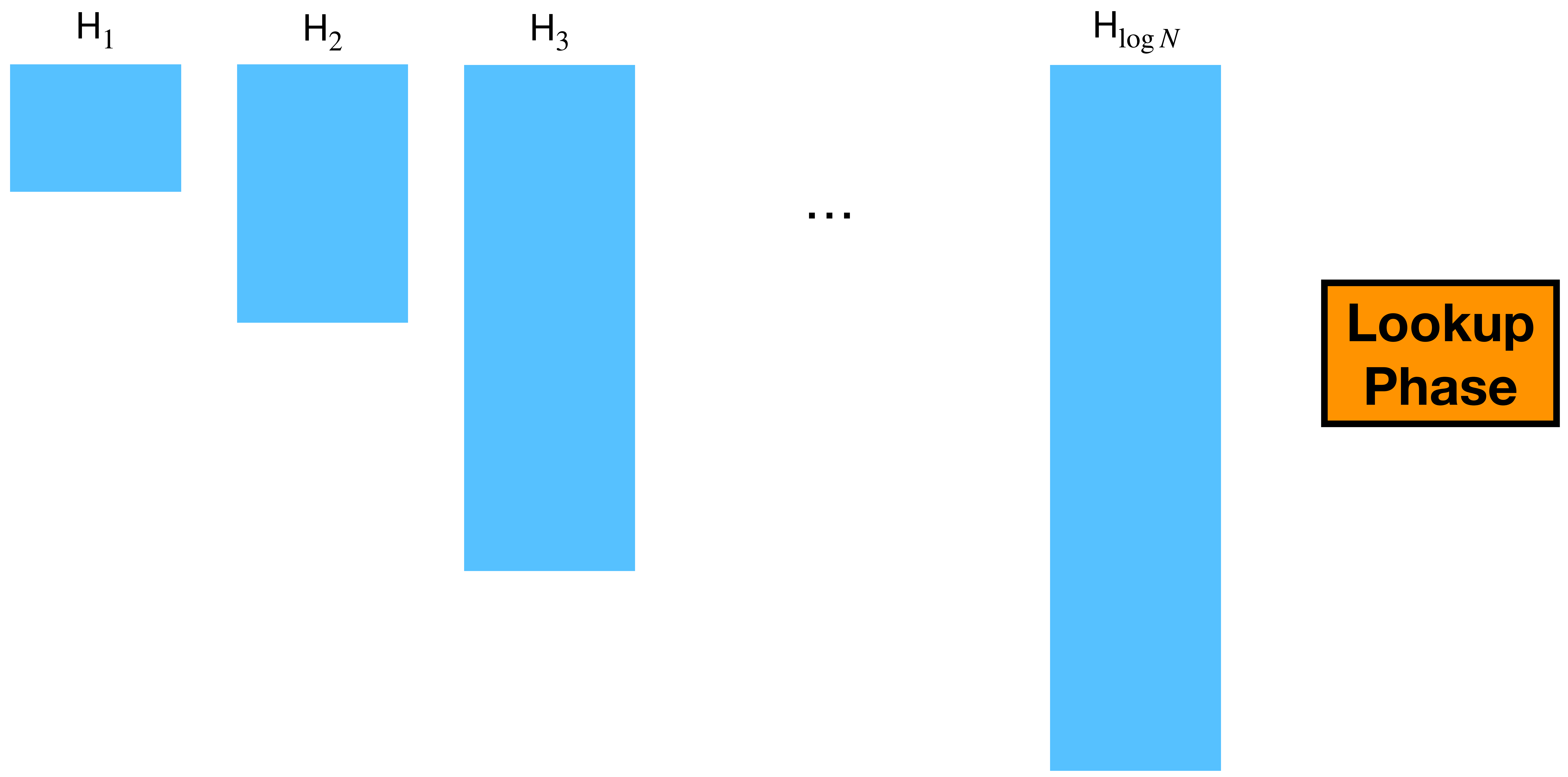
Replay Attack for Hierarchical Framework

- **Key fact**: Oblivious hash tables are oblivious only if lookups are *non-recurrent*.
  - If you look up the same addr twice in some H_i without rebuilding in between, **access pattern to H_i will be identical – not oblivious**.

Replay Attack for Hierarchical Framework

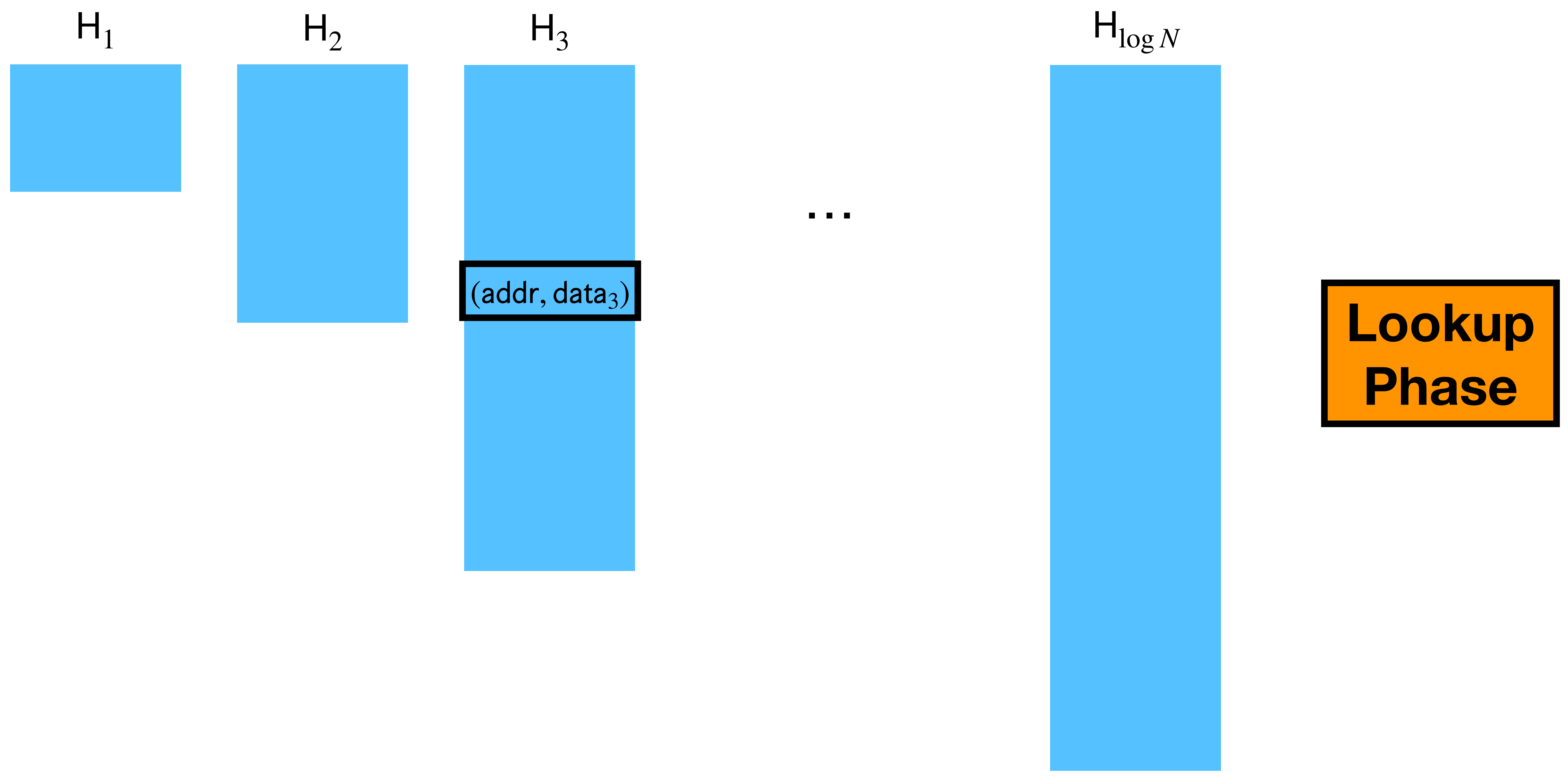
- **Key fact**: Oblivious hash tables are oblivious only if lookups are *non-recurrent*.
  - If you look up the same addr twice in some H_i without rebuilding in between, **access pattern to H_i will be identical – not oblivious***.*
  - In honest-but-curious setting, looking up **dummies** and rebuilding hash tables ensures reads will be non-recurrent.

Replay Attack



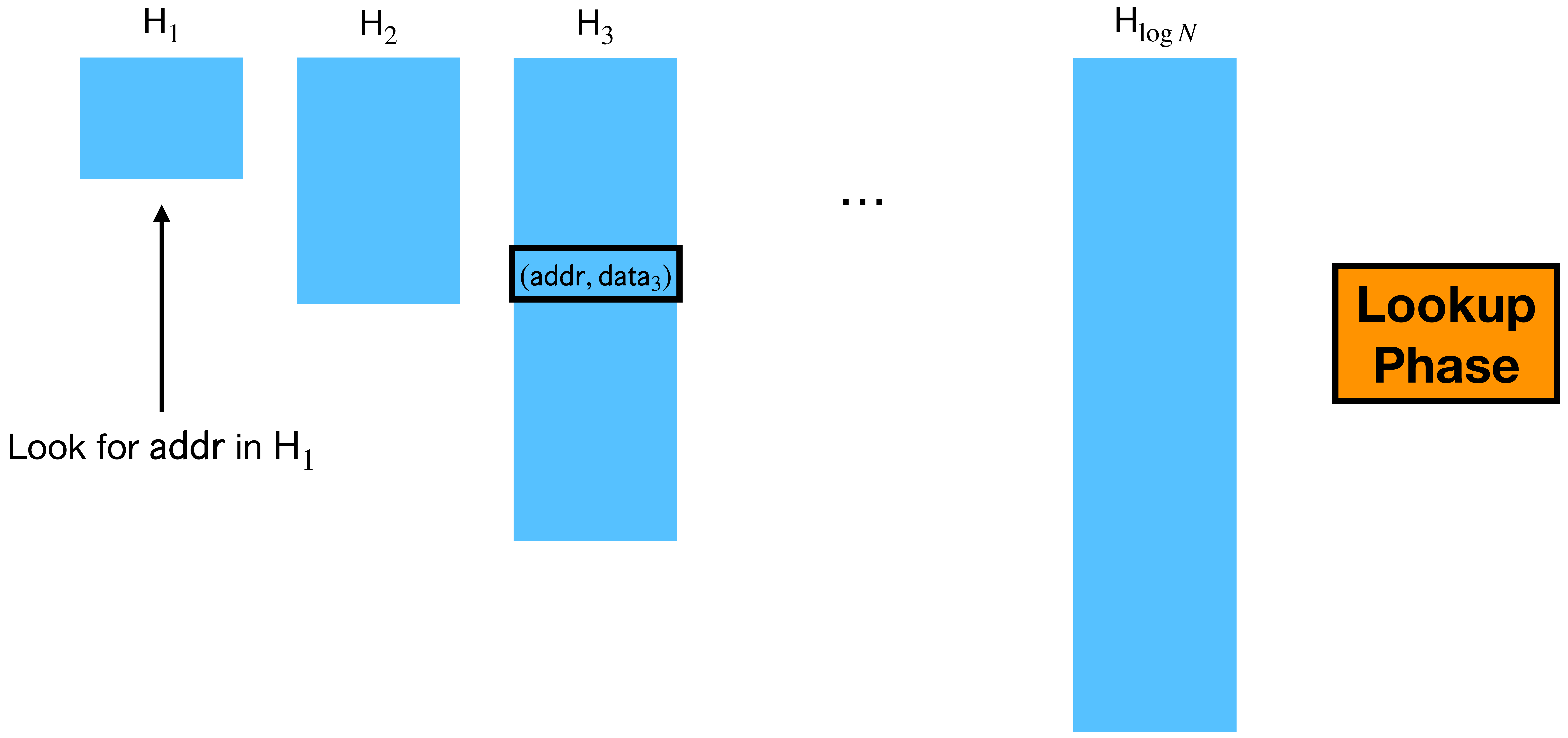
Replay Attack

Read addr:



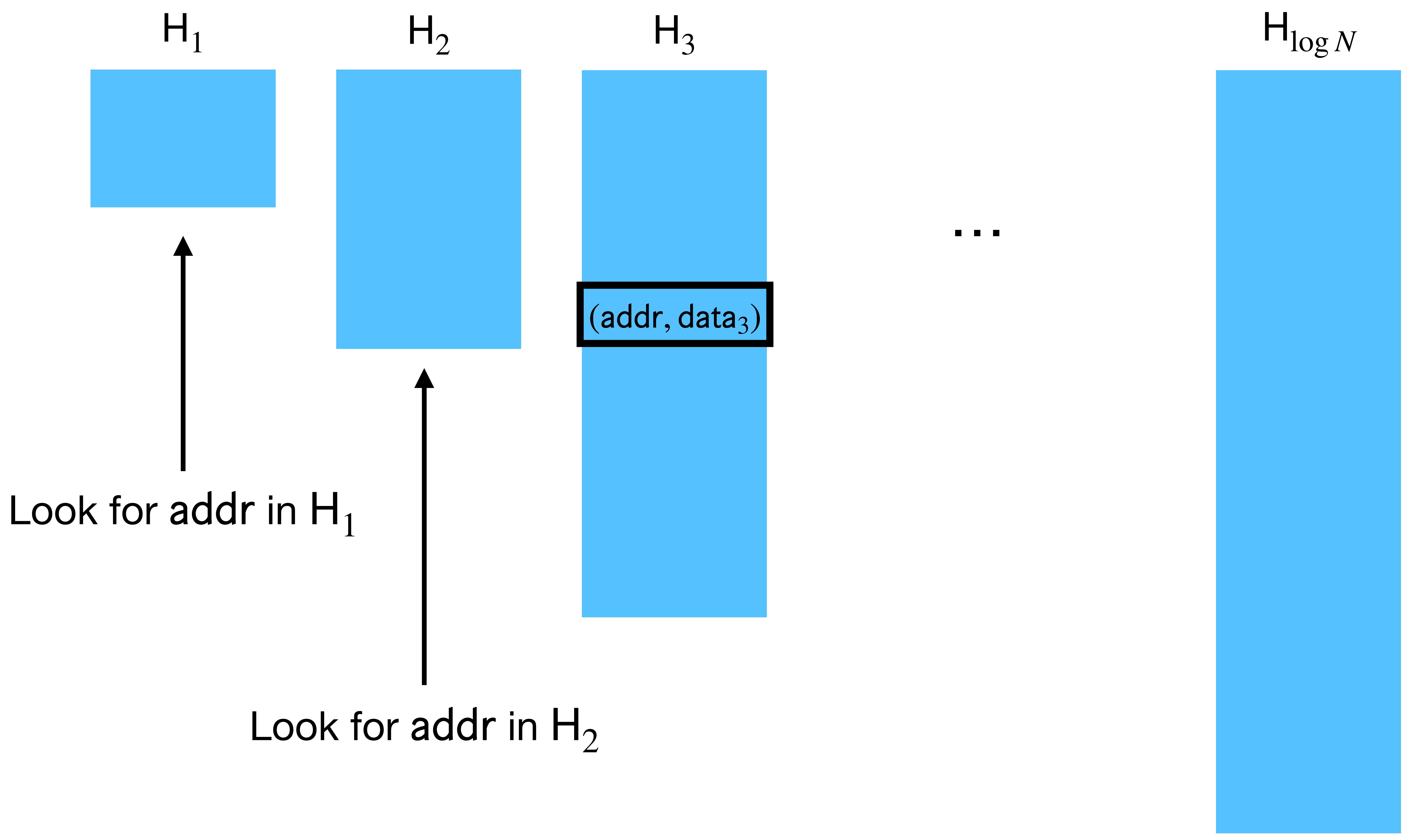
Replay Attack

Read addr:



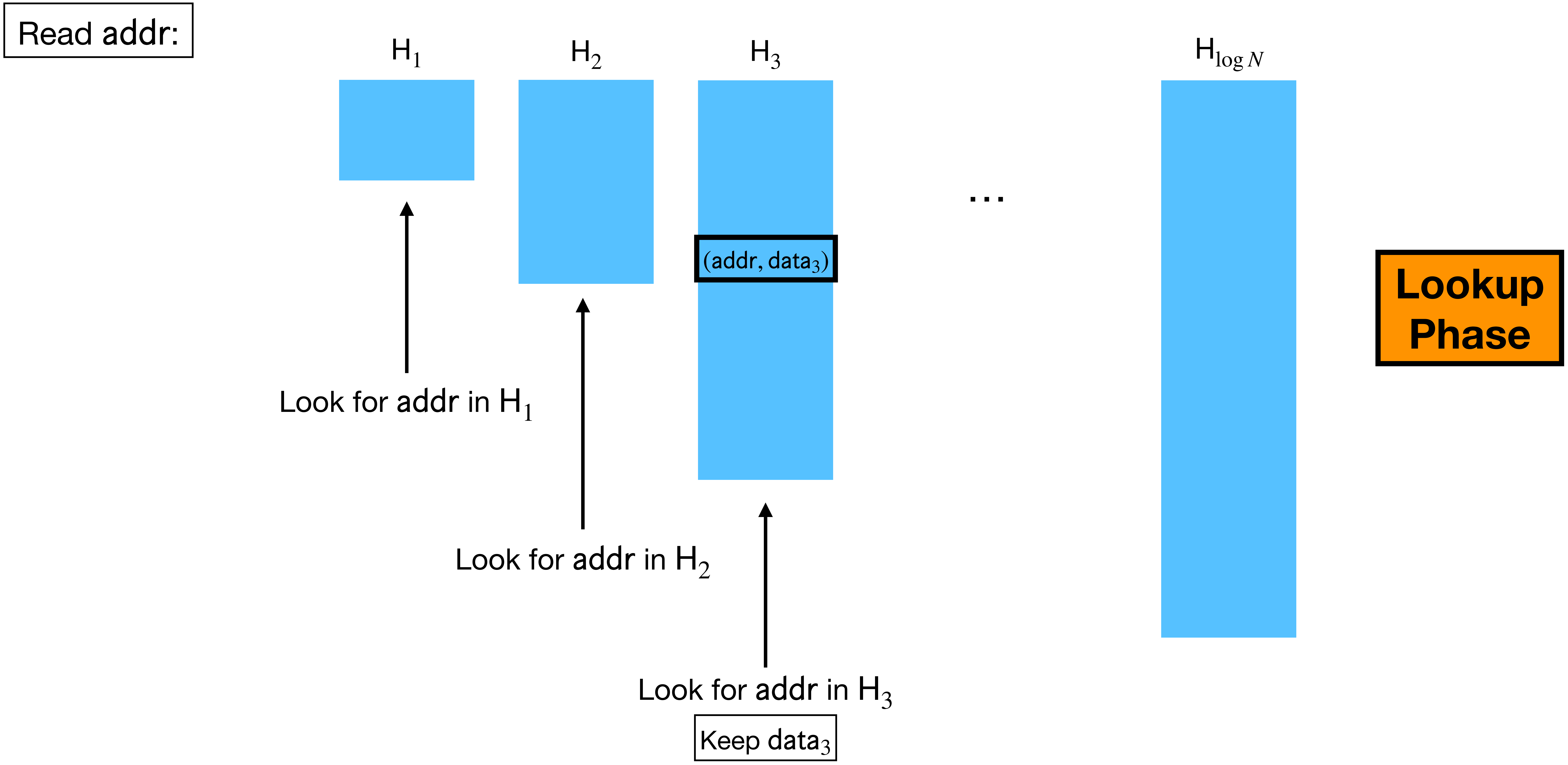
Replay Attack

Read addr:

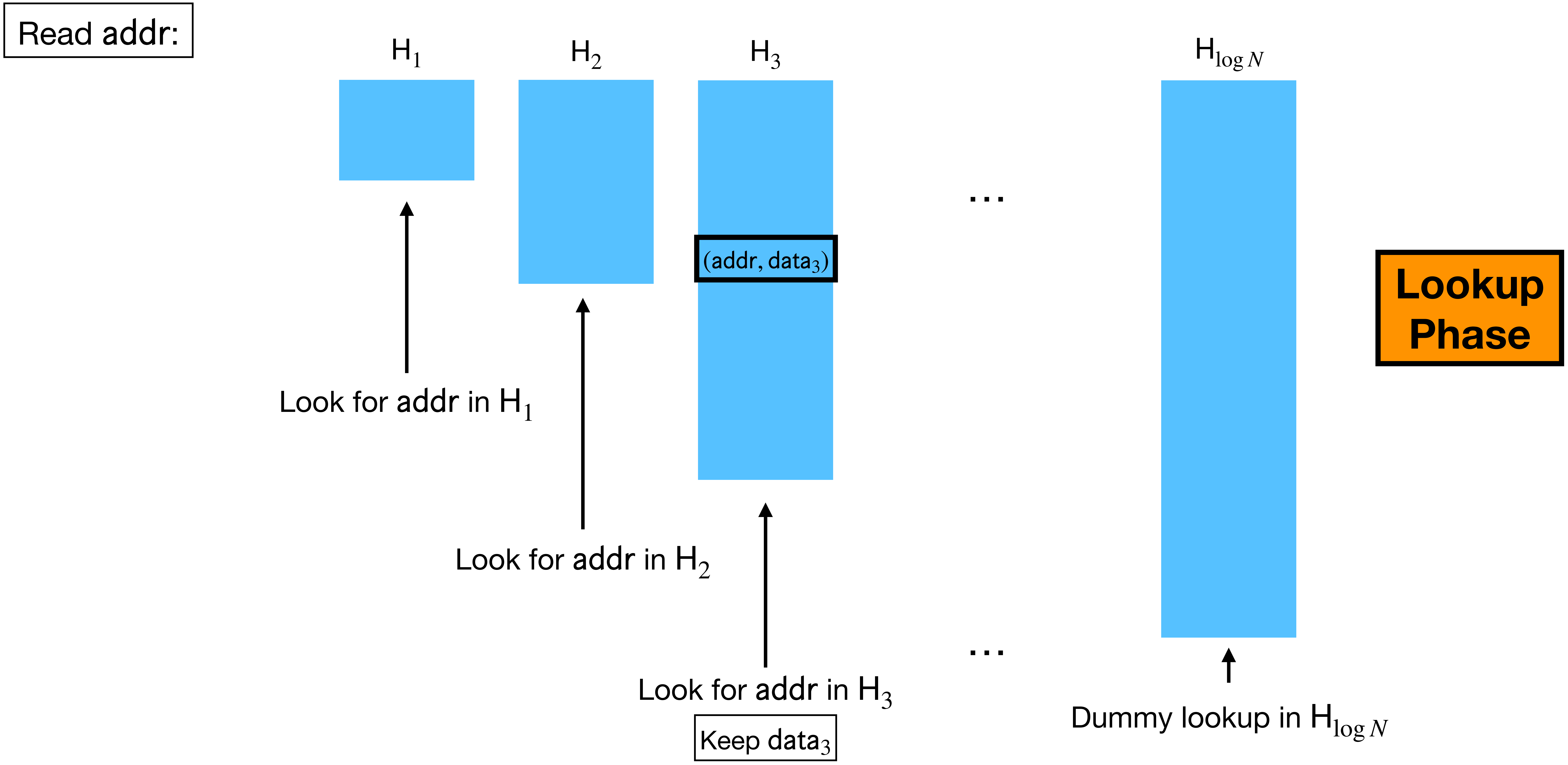


Lookup Phase

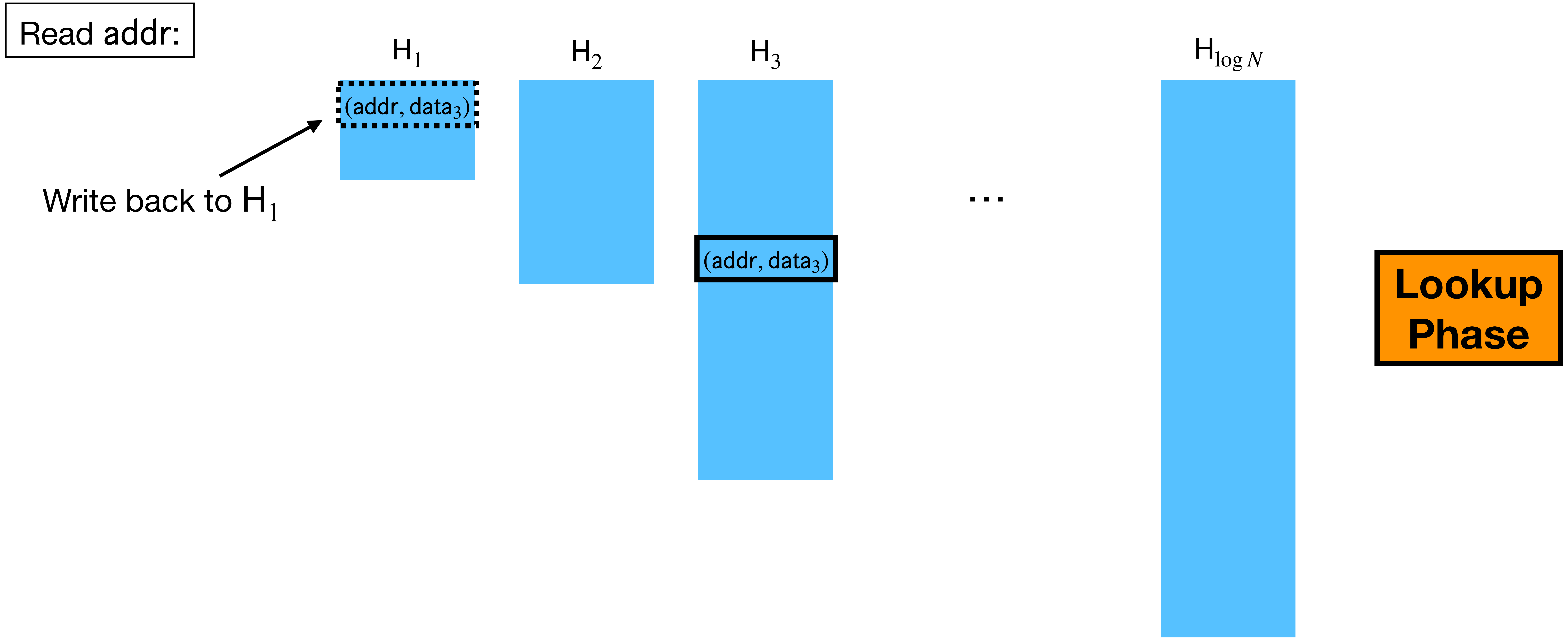
Replay Attack



Replay Attack



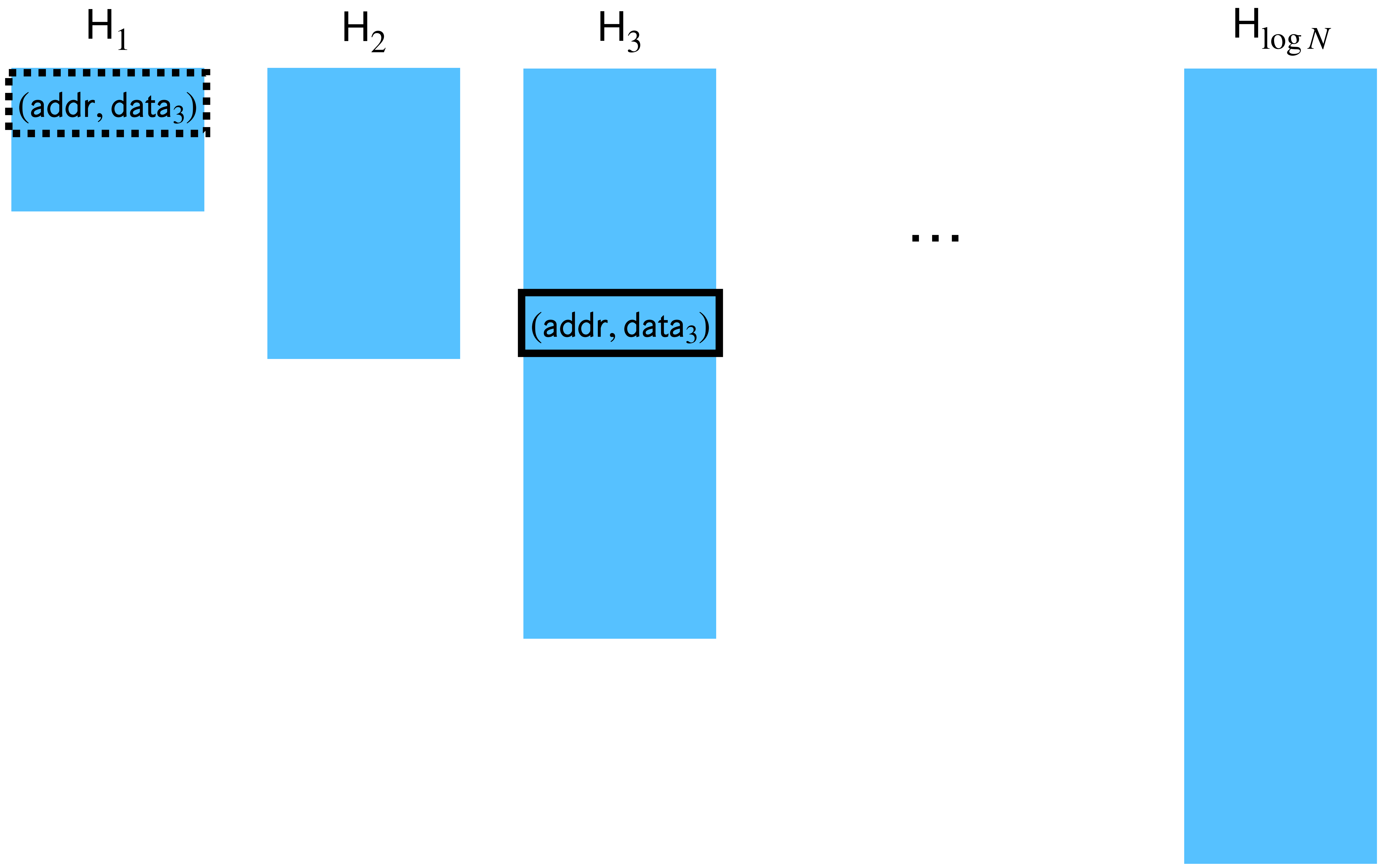
Replay Attack



Replay Attack

Read addr:

Write to addr:

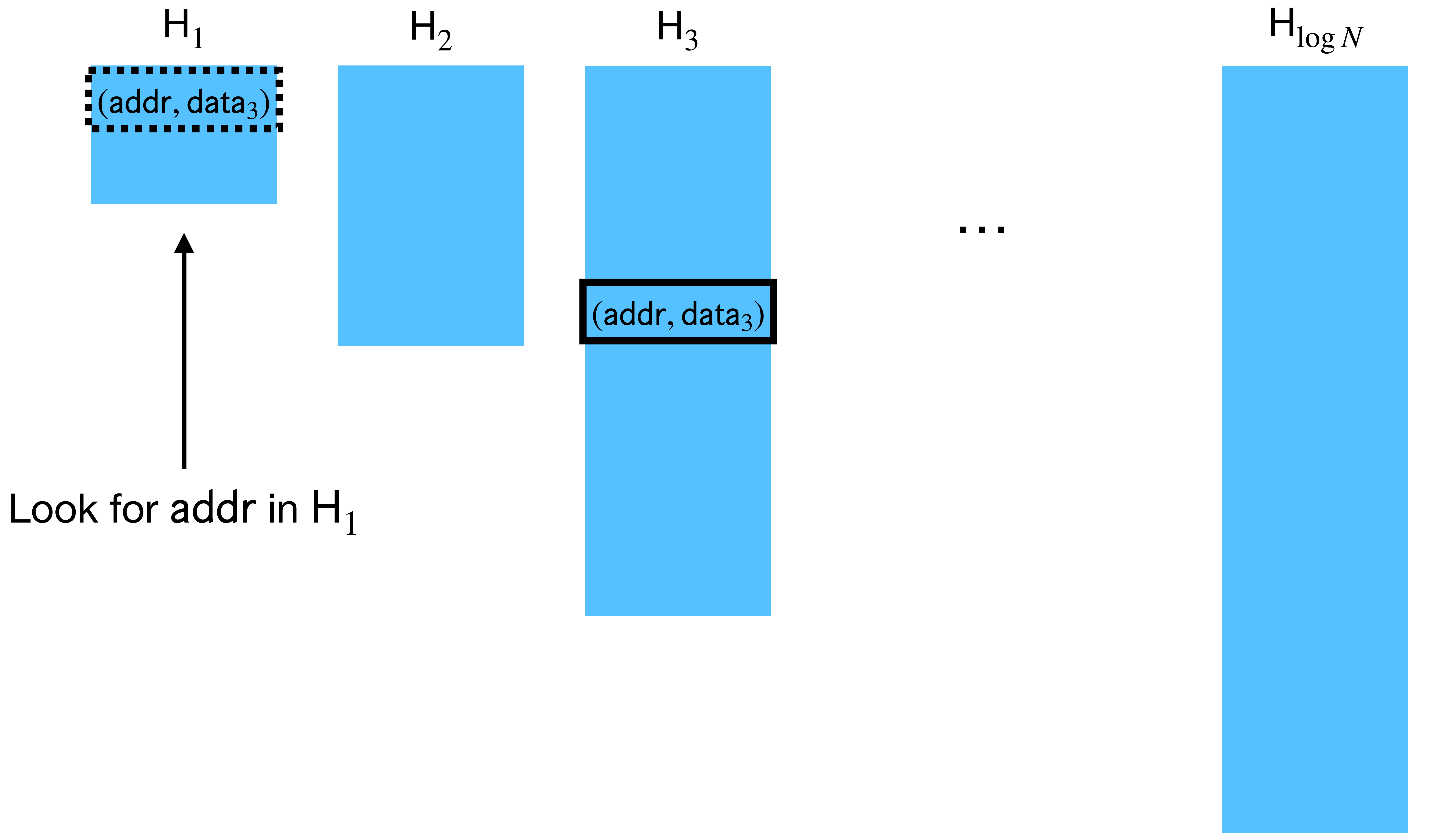


Lookup Phase

Replay Attack

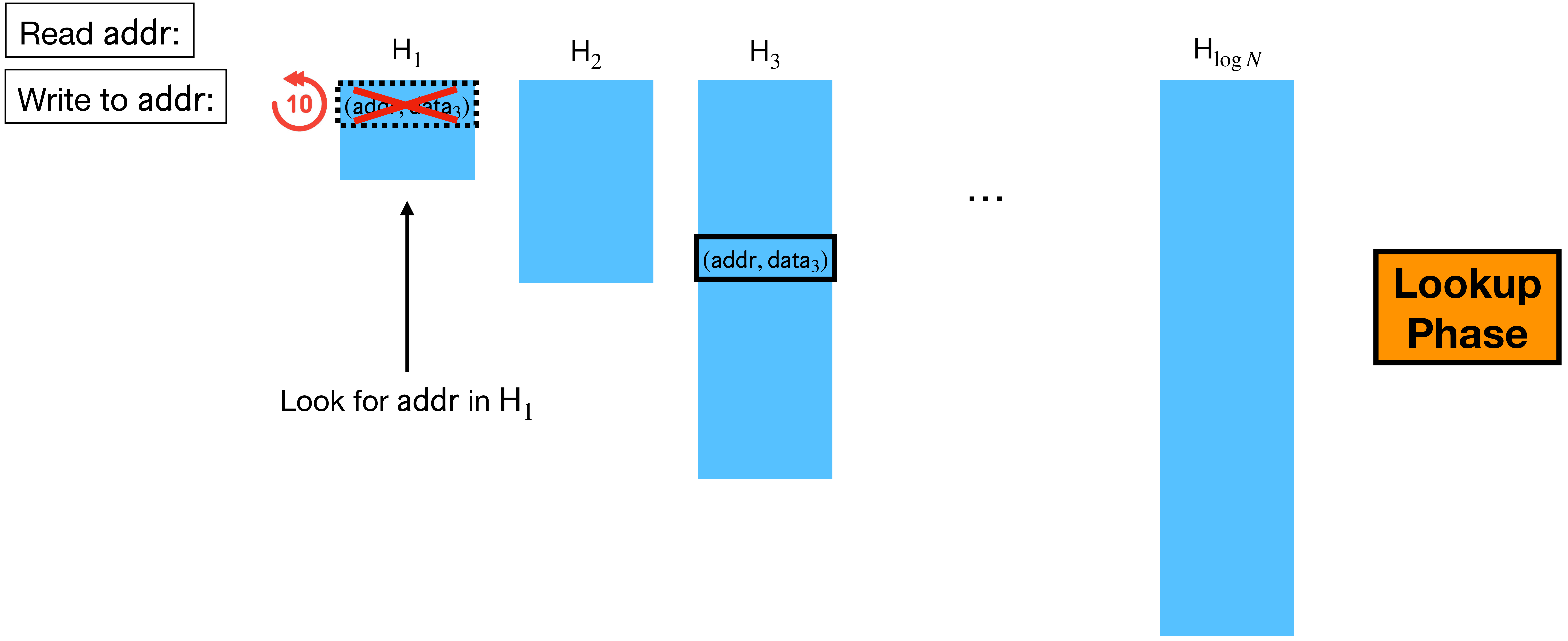
Read addr:

Write to addr:



Lookup Phase

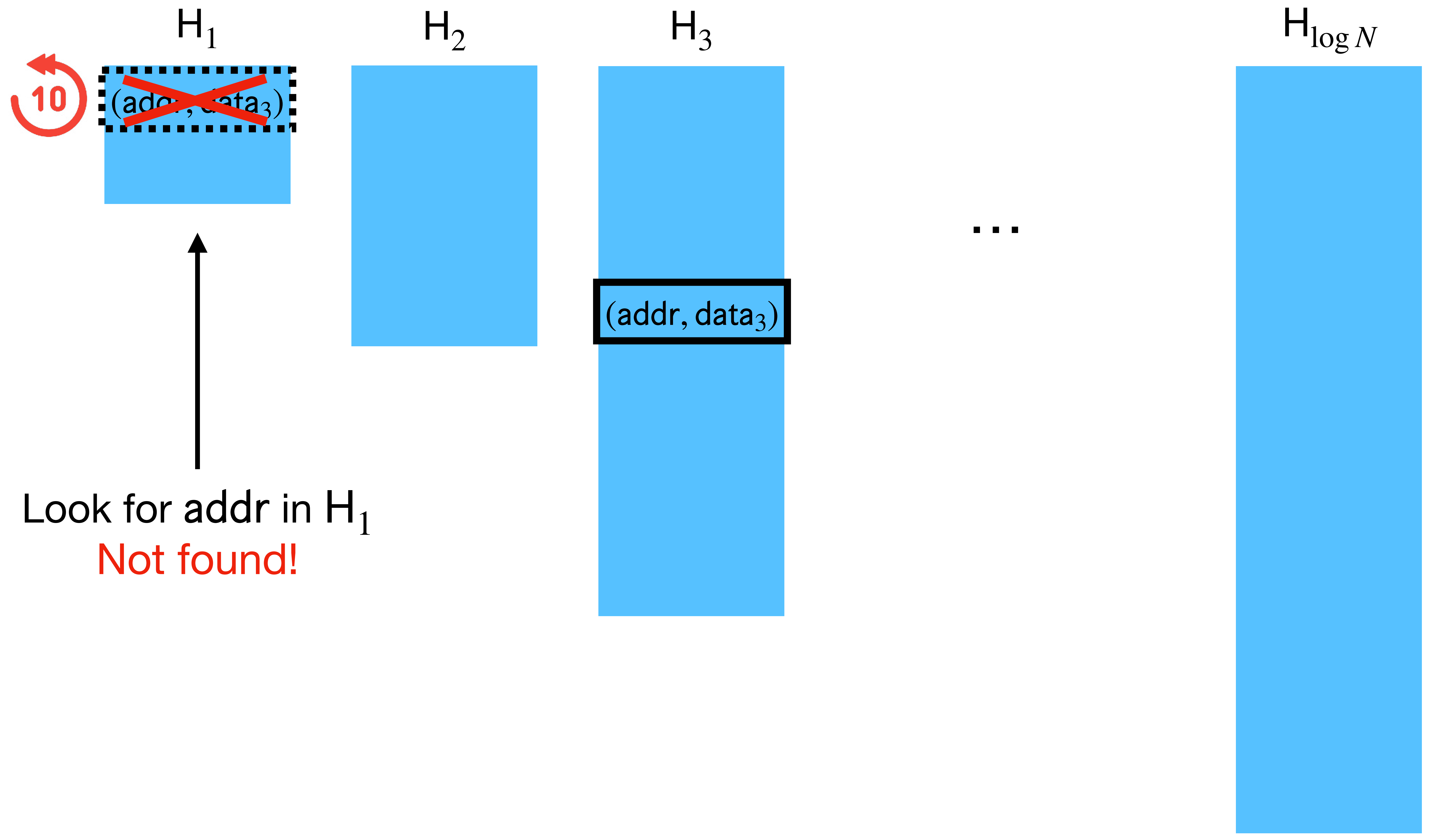
Replay Attack



Replay Attack

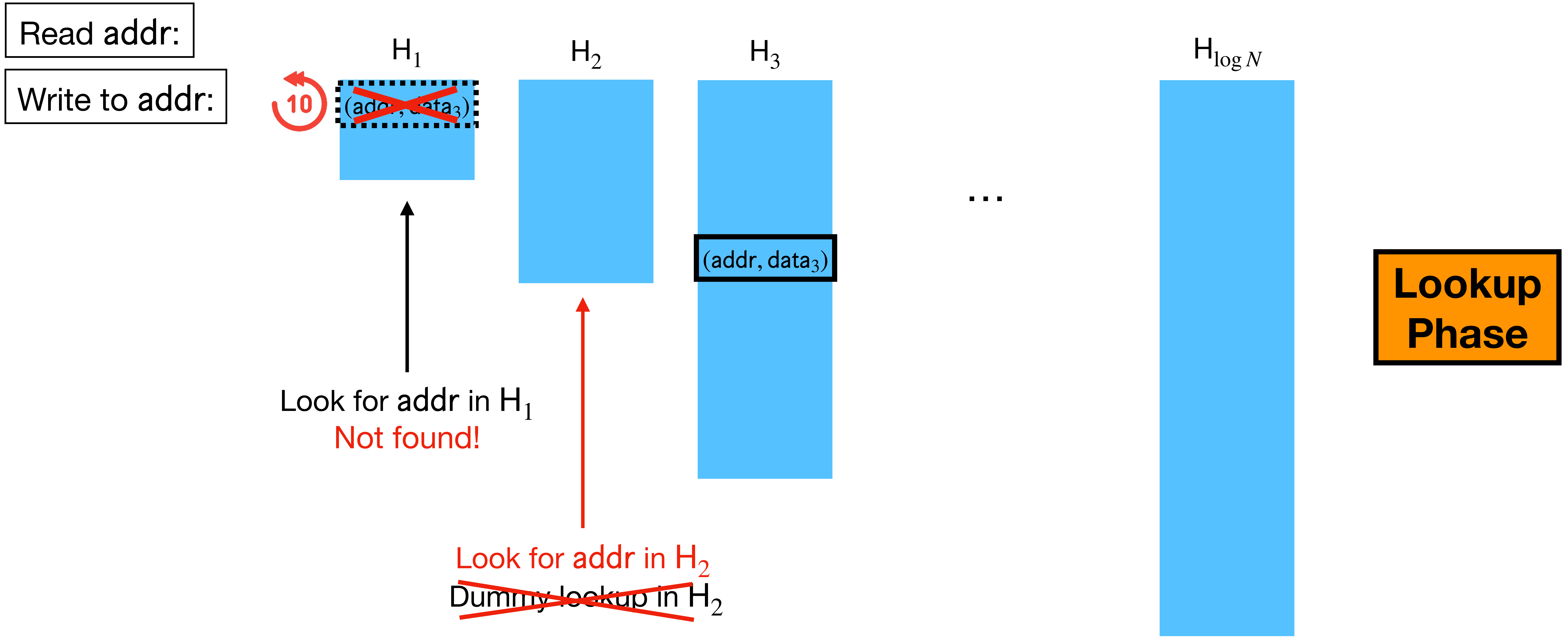
Read addr:

Write to addr:

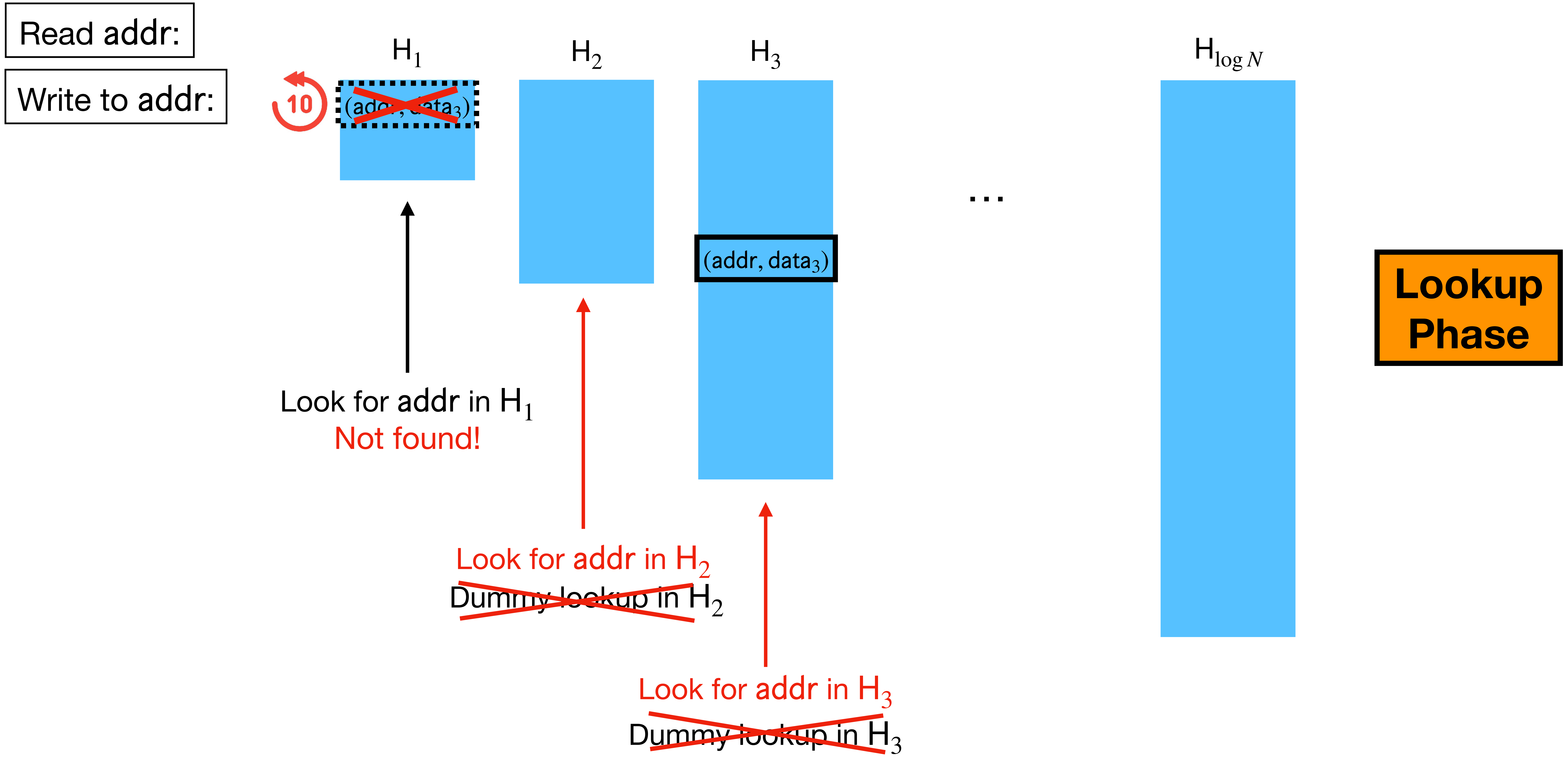


Lookup Phase

Replay Attack



Replay Attack



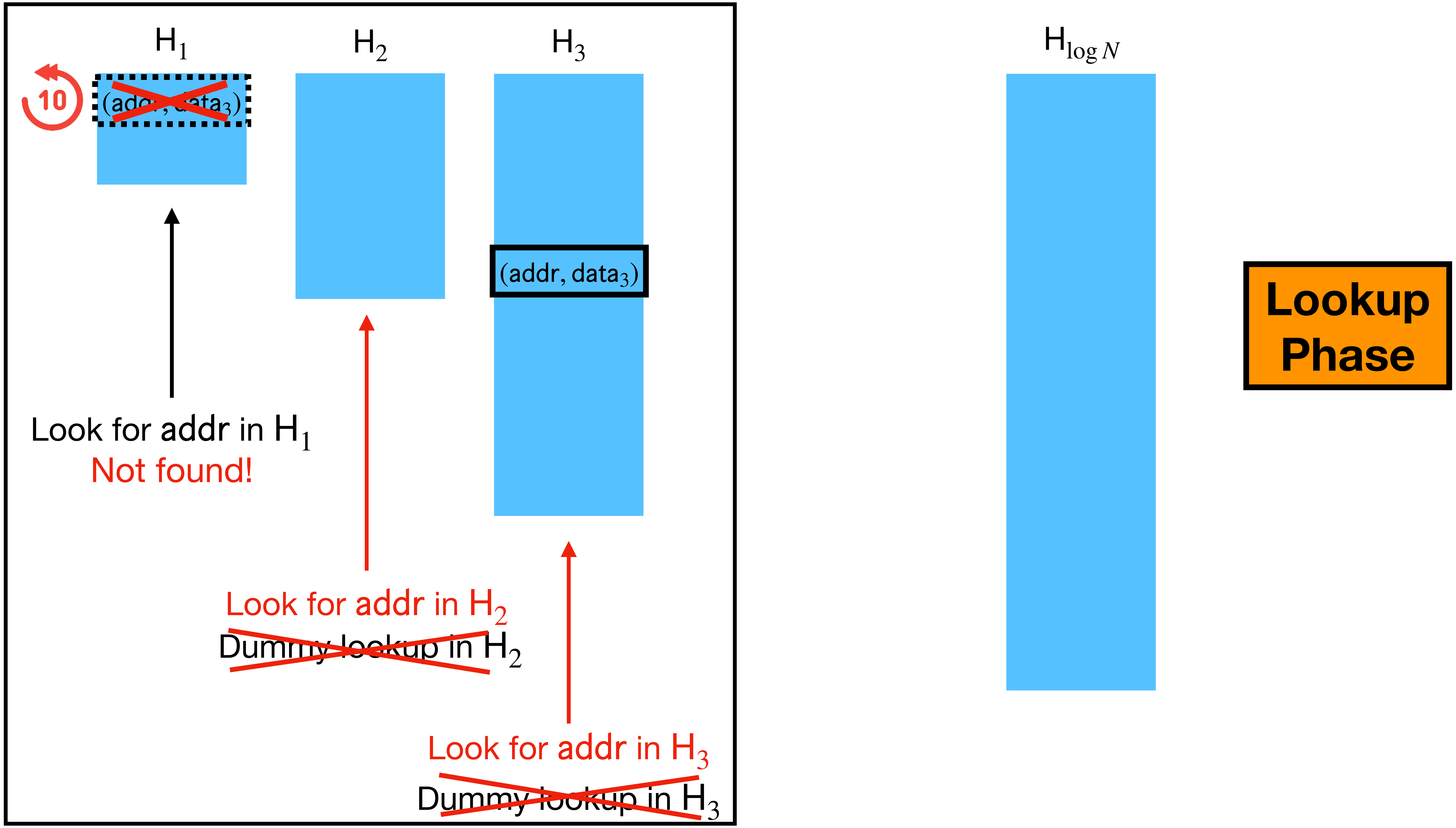
Replay Attack

Read addr:

Write to addr:

Exact same access pattern as first query!

Leaks repeated address.



Replay Attack

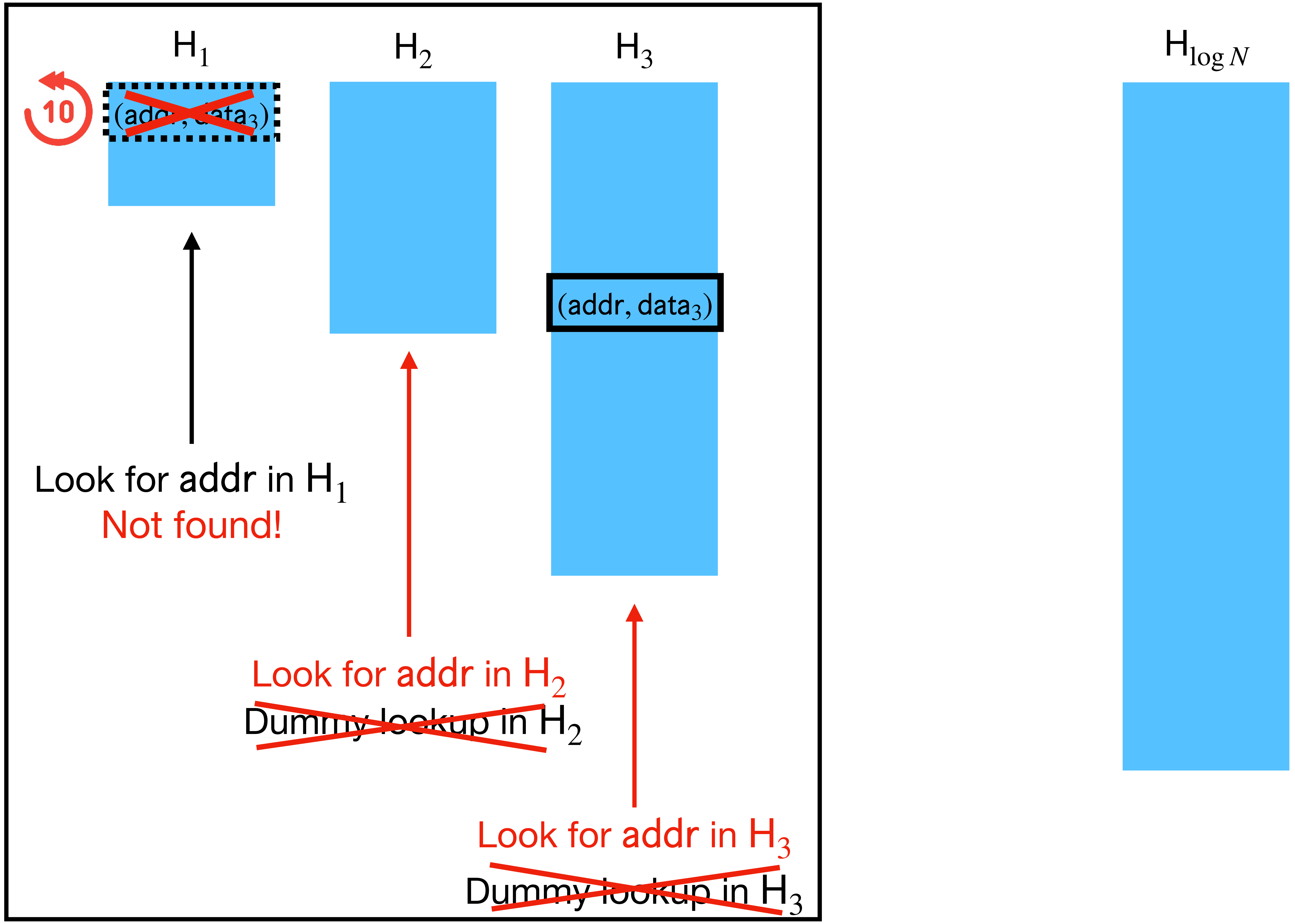
Read addr:

Write to addr:

Exact same access pattern as first query!

Leaks repeated address.

Obliviousness of H_i lookups depends on **correctness** of $H_{<i}$ lookups!



Lookup Phase

General Fix For Replay Attacks: Time-Stamping

- [Ostrovsky ’90, Goldreich-Ostrovsky ’96] noticed that *time-stamping* is sufficient to prevent replay attacks with MACs (in their $O(\log^3 N)$ ORAM).

General Fix For Replay Attacks: Time-Stamping

- [Ostrovsky ’90, Goldreich-Ostrovsky ’96] noticed that *time-stamping* is sufficient to prevent replay attacks with MACs (in their $O(\log^3 N)$ ORAM).
- **Time-stamping**:

General Fix For Replay Attacks: Time-Stamping

- [Ostrovsky ’90, Goldreich-Ostrovsky ’96] noticed that *time-stamping* is sufficient to prevent replay attacks with MACs (in their $O(\log^3 N)$ ORAM).
- **Time-stamping**:
    - Keep track of global counter ctr, counting the number of $\widehat{\mathsf{query}}$’s so far.

General Fix For Replay Attacks: Time-Stamping

- [Ostrovsky '90, Goldreich-Ostrovsky '96] noticed that *time-stamping* is sufficient to prevent replay attacks with MACs (in their $O(\log^3 N)$ ORAM).
- **Time-stamping**:
  - Keep track of global counter ctr, counting the number of $\widehat{\mathsf{query}}$'s so far.

$$\boxed{\mathsf{PrevTime}\left(\mathsf{ctr}, \widehat{\mathsf{addr}}\right) := \begin{array}{c}\text{most recent time (up until } \mathsf{ctr}\text{)} \\ \text{when } \widehat{\mathsf{addr}} \text{ has been written to.}\end{array}}$$

General Fix For Replay Attacks: Time-Stamping

- [Ostrovsky '90, Goldreich-Ostrovsky '96] noticed that *time-stamping* is sufficient to prevent replay attacks with MACs (in their $O(\log^3 N)$ ORAM).
- **Time-stamping**:
    - Keep track of global counter ctr, counting the number of $\widehat{\mathsf{query}}$'s so far.

$$\boxed{\mathsf{PrevTime}\left(\mathsf{ctr}, \widehat{\mathsf{addr}}\right) := \begin{array}{c}\text{most recent time (up until ctr)}\\ \text{when } \widehat{\mathsf{addr}} \text{ has been written to.}\end{array}}$$

- **Theorem** [GO '96]: If ORAM has **local, low-space** computable $\mathsf{PrevTime}$, then MACs + time-stamping converts honest-but-curious ORAM to maliciously secure ORAM with the same asymptotic overhead.

Time-Stamping

$\widehat{\text{addr}}_1$	$\widehat{\text{addr}}_2$	$\widehat{\text{addr}}_3$	$\widehat{\text{addr}}_4$	$\widehat{\text{addr}}_5$	$\widehat{\text{addr}}_6$	$\widehat{\text{addr}}_7$
$\text{data}_1, \text{ctr}_1$	$\text{data}_2, \text{ctr}_2$	$\text{data}_3, \text{ctr}_3$	$\text{data}_4, \text{ctr}_4$	$\text{data}_5, \text{ctr}_5$	$\text{data}_6, \text{ctr}_6$	$\text{data}_7, \text{ctr}_7$

$$\boxed{\text{PrevTime}\left(\text{ctr}, \widehat{\text{addr}}\right) := \begin{array}{c}\text{most recent time (up until ctr)} \\ \text{when } \widehat{\text{addr}} \text{ has been written to.}\end{array}}$$

Time-Stamping

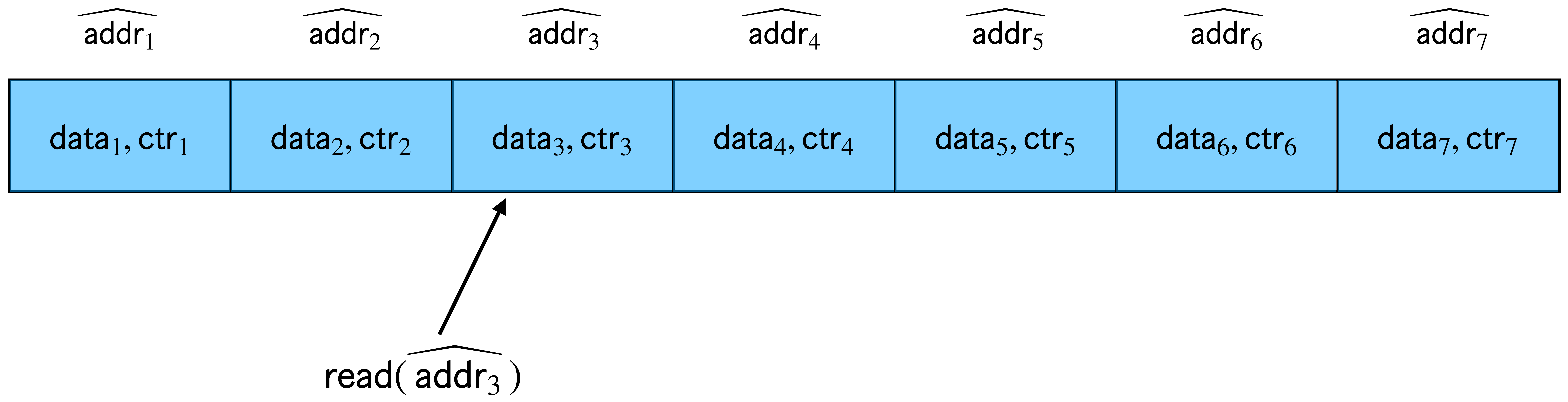
All entries are MAC'ed
Current time: ctr

$\widehat{\text{addr}}_1$	$\widehat{\text{addr}}_2$	$\widehat{\text{addr}}_3$	$\widehat{\text{addr}}_4$	$\widehat{\text{addr}}_5$	$\widehat{\text{addr}}_6$	$\widehat{\text{addr}}_7$
$\text{data}_1, \text{ctr}_1$	$\text{data}_2, \text{ctr}_2$	$\text{data}_3, \text{ctr}_3$	$\text{data}_4, \text{ctr}_4$	$\text{data}_5, \text{ctr}_5$	$\text{data}_6, \text{ctr}_6$	$\text{data}_7, \text{ctr}_7$

$\text{PrevTime}\left(\text{ctr}, \widehat{\text{addr}}\right) :=$ most recent time (up until ctr) when $\widehat{\text{addr}}$ has been written to.

Time-Stamping

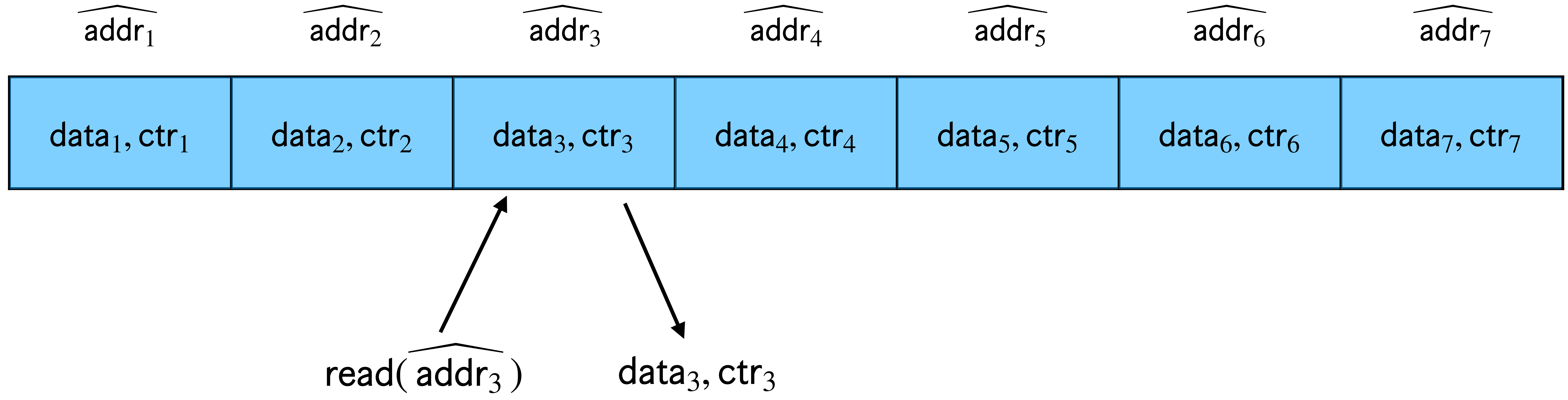
All entries are MAC'ed
Current time: ctr


$\widehat{\text{addr}}_1$
$\widehat{\text{addr}}_2$
$\widehat{\text{addr}}_3$
$\widehat{\text{addr}}_4$
$\widehat{\text{addr}}_5$
$\widehat{\text{addr}}_6$
$\widehat{\text{addr}}_7$
$\text{data}_1, \text{ctr}_1$
$\text{data}_2, \text{ctr}_2$
$\text{data}_3, \text{ctr}_3$
$\text{data}_4, \text{ctr}_4$
$\text{data}_5, \text{ctr}_5$
$\text{data}_6, \text{ctr}_6$
$\text{data}_7, \text{ctr}_7$
$\text{read}(\widehat{\text{addr}}_3)$

$\text{PrevTime}\left(\text{ctr}, \widehat{\text{addr}}\right) :=$ most recent time (up until ctr) when $\widehat{\text{addr}}$ has been written to.

Time-Stamping

All entries are MAC'ed
Current time: ctr

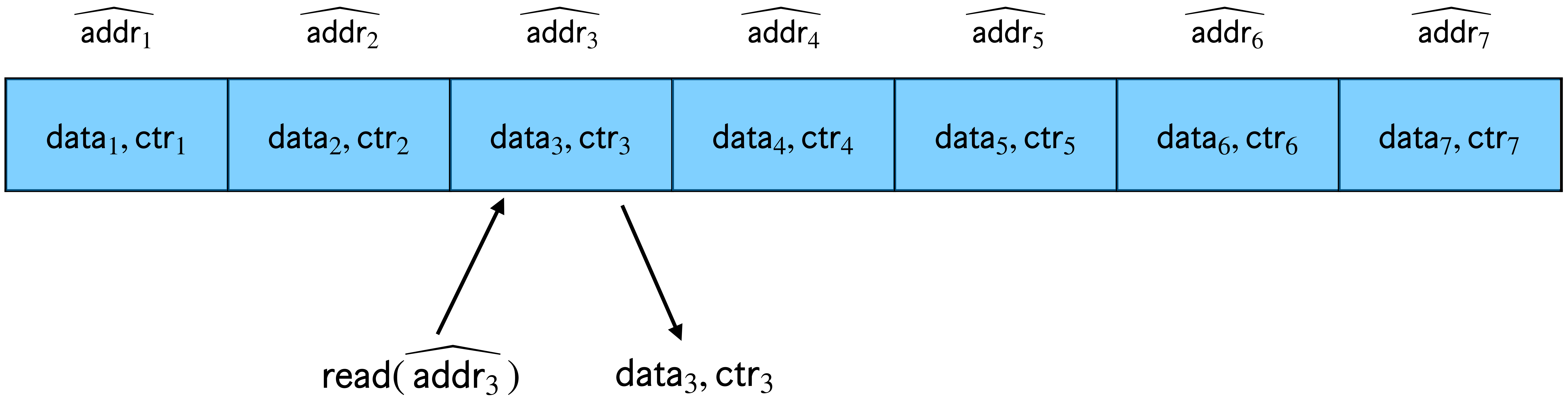


$\widehat{\text{addr}_1}$
$\widehat{\text{addr}_2}$
$\widehat{\text{addr}_3}$
$\widehat{\text{addr}_4}$
$\widehat{\text{addr}_5}$
$\widehat{\text{addr}_6}$
$\widehat{\text{addr}_7}$
$\text{data}_1, \text{ctr}_1$
$\text{data}_2, \text{ctr}_2$
$\text{data}_3, \text{ctr}_3$
$\text{data}_4, \text{ctr}_4$
$\text{data}_5, \text{ctr}_5$
$\text{data}_6, \text{ctr}_6$
$\text{data}_7, \text{ctr}_7$
$\text{read}(\widehat{\text{addr}_3})$
$\text{data}_3, \text{ctr}_3$

$\text{PrevTime}\left(\text{ctr}, \widehat{\text{addr}}\right) :=$ most recent time (up until ctr) when $\widehat{\text{addr}}$ has been written to.

Time-Stamping

All entries are MAC'ed
Current time: ctr



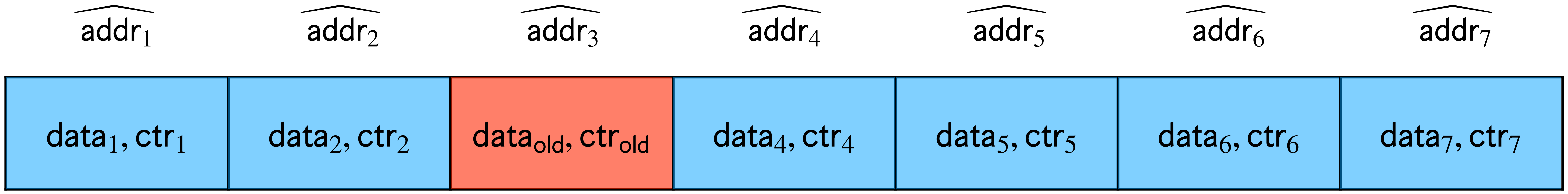
$\widehat{\text{addr}}_1$
$\widehat{\text{addr}}_2$
$\widehat{\text{addr}}_3$
$\widehat{\text{addr}}_4$
$\widehat{\text{addr}}_5$
$\widehat{\text{addr}}_6$
$\widehat{\text{addr}}_7$
$\text{data}_1, \text{ctr}_1$
$\text{data}_2, \text{ctr}_2$
$\text{data}_3, \text{ctr}_3$
$\text{data}_4, \text{ctr}_4$
$\text{data}_5, \text{ctr}_5$
$\text{data}_6, \text{ctr}_6$
$\text{data}_7, \text{ctr}_7$
$\text{read}(\widehat{\text{addr}}_3)$
$\text{data}_3, \text{ctr}_3$

$\text{PrevTime}(\text{ctr}, \widehat{\text{addr}}_3) = \text{ctr}_3$ ✓

$\text{PrevTime}\left(\text{ctr}, \widehat{\text{addr}}\right) :=$ most recent time (up until ctr) when $\widehat{\text{addr}}$ has been written to.

Time-Stamping

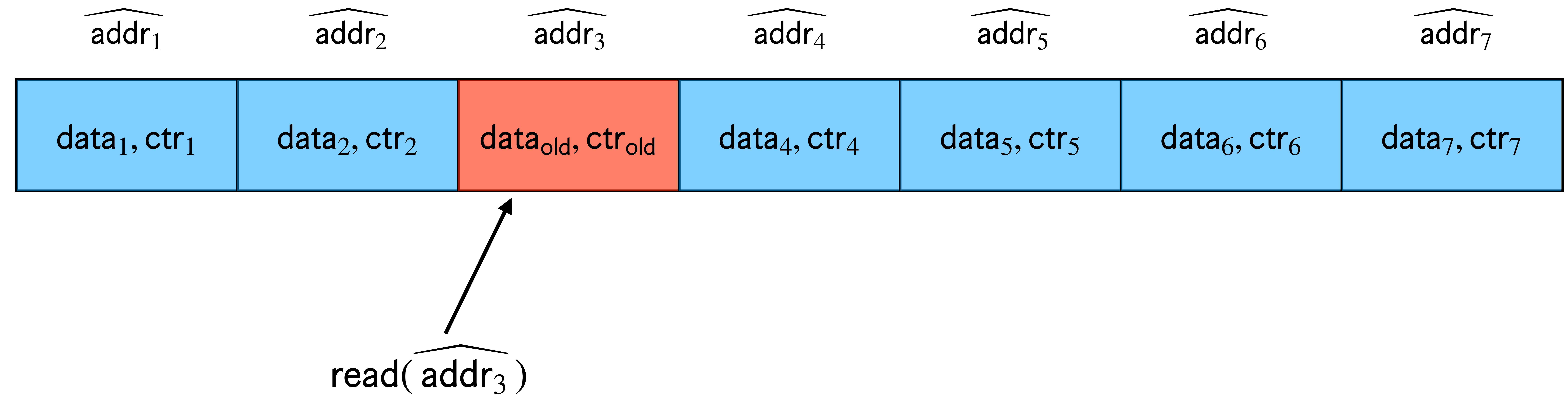
All entries are MAC'ed
Current time: ctr


$\widehat{\text{addr}_1}$
$\widehat{\text{addr}_2}$
$\widehat{\text{addr}_3}$
$\widehat{\text{addr}_4}$
$\widehat{\text{addr}_5}$
$\widehat{\text{addr}_6}$
$\widehat{\text{addr}_7}$
$\text{data}_1, \text{ctr}_1$
$\text{data}_2, \text{ctr}_2$
$\text{data}_{\text{old}}, \text{ctr}_{\text{old}}$
$\text{data}_4, \text{ctr}_4$
$\text{data}_5, \text{ctr}_5$
$\text{data}_6, \text{ctr}_6$
$\text{data}_7, \text{ctr}_7$

$\text{PrevTime}\left(\text{ctr}, \widehat{\text{addr}}\right) :=$ most recent time (up until ctr) when $\widehat{\text{addr}}$ has been written to.

Time-Stamping

All entries are MAC'ed
Current time: ctr

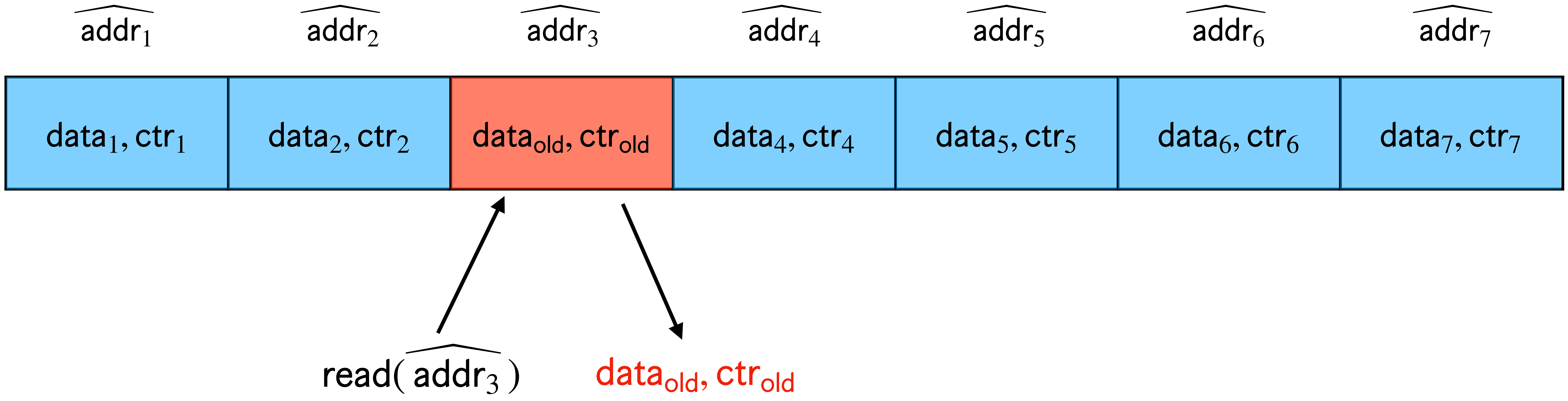


$\widehat{\text{addr}_1}$
$\widehat{\text{addr}_2}$
$\widehat{\text{addr}_3}$
$\widehat{\text{addr}_4}$
$\widehat{\text{addr}_5}$
$\widehat{\text{addr}_6}$
$\widehat{\text{addr}_7}$
$\text{data}_1, \text{ctr}_1$
$\text{data}_2, \text{ctr}_2$
$\text{data}_{\text{old}}, \text{ctr}_{\text{old}}$
$\text{data}_4, \text{ctr}_4$
$\text{data}_5, \text{ctr}_5$
$\text{data}_6, \text{ctr}_6$
$\text{data}_7, \text{ctr}_7$
$\text{read}(\widehat{\text{addr}_3})$

$\text{PrevTime}\left(\text{ctr}, \widehat{\text{addr}}\right) :=$ most recent time (up until ctr) when $\widehat{\text{addr}}$ has been written to.

Time-Stamping

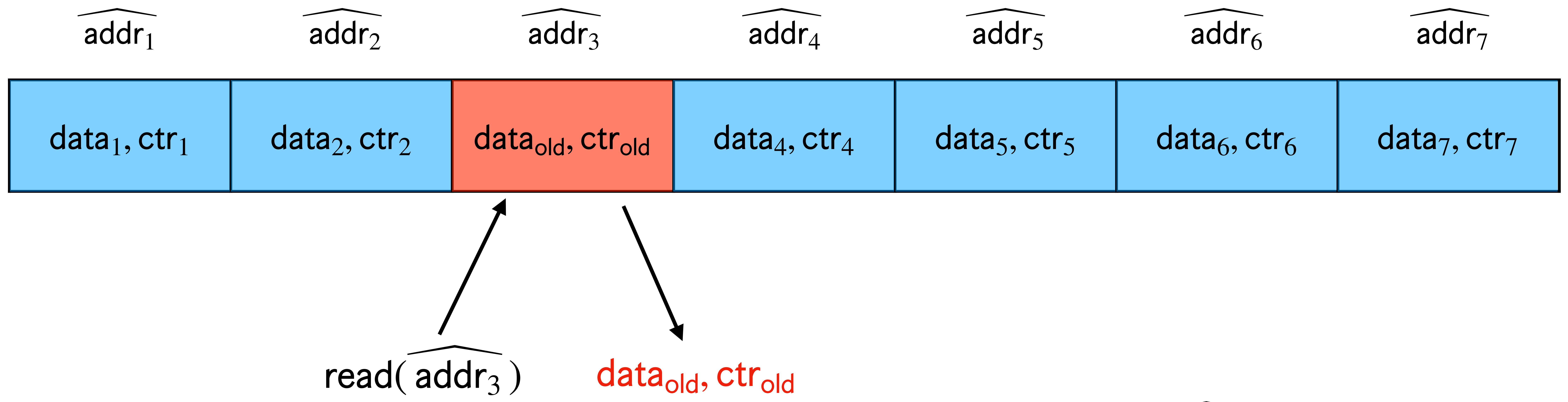
All entries are MAC'ed
Current time: ctr


$\widehat{\text{addr}}_1$
$\widehat{\text{addr}}_2$
$\widehat{\text{addr}}_3$
$\widehat{\text{addr}}_4$
$\widehat{\text{addr}}_5$
$\widehat{\text{addr}}_6$
$\widehat{\text{addr}}_7$
$\text{data}_1, \text{ctr}_1$
$\text{data}_2, \text{ctr}_2$
$\text{data}_{\text{old}}, \text{ctr}_{\text{old}}$
$\text{data}_4, \text{ctr}_4$
$\text{data}_5, \text{ctr}_5$
$\text{data}_6, \text{ctr}_6$
$\text{data}_7, \text{ctr}_7$
$\text{read}(\widehat{\text{addr}}_3)$
$\text{data}_{\text{old}}, \text{ctr}_{\text{old}}$

$$\text{PrevTime}\left(\text{ctr}, \widehat{\text{addr}}\right) := \begin{array}{c}\text{most recent time (up until ctr)} \\ \text{when } \widehat{\text{addr}} \text{ has been written to.}\end{array}$$

Time-Stamping

All entries are MAC’ed
Current time: ctr



$\widehat{\text{addr}}_1$
$\widehat{\text{addr}}_2$
$\widehat{\text{addr}}_3$
$\widehat{\text{addr}}_4$
$\widehat{\text{addr}}_5$
$\widehat{\text{addr}}_6$
$\widehat{\text{addr}}_7$
$\text{data}_1, \text{ctr}_1$
$\text{data}_2, \text{ctr}_2$
$\text{data}_{\text{old}}, \text{ctr}_{\text{old}}$
$\text{data}_4, \text{ctr}_4$
$\text{data}_5, \text{ctr}_5$
$\text{data}_6, \text{ctr}_6$
$\text{data}_7, \text{ctr}_7$
$\text{read}(\widehat{\text{addr}}_3)$
$\text{data}_{\text{old}}, \text{ctr}_{\text{old}}$

Since $\text{ctr}_{\text{old}} < \text{ctr}_3 = \text{PrevTime}(\text{ctr}, \widehat{\text{addr}}_3)$,
replay attack detected!

$$\text{PrevTime}\left(\text{ctr}, \widehat{\text{addr}}\right) := \begin{array}{c}\text{most recent time (up until ctr)} \\ \text{when } \widehat{\text{addr}} \text{ has been written to.}\end{array}$$

Time-Stamping is Hard

Time-Stamping is Hard

- Unfortunately, **the recent ORAM constructions cannot be time-stamped.**

Time-Stamping is Hard

- Unfortunately, **the recent ORAM constructions cannot be time-stamped.**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.

Replay Attack for Hierarchical

- As is, the hierarchical paradigm with MACs is susceptible to replay attacks, so it’s still maliciously insecure.

Replay Attack for Hierarchical

- As is, the hierarchical paradigm with MACs is susceptible to replay attacks, so it’s still maliciously insecure.
- Is there a fix?

Time-Stamping is Hard

Time-Stamping is Hard

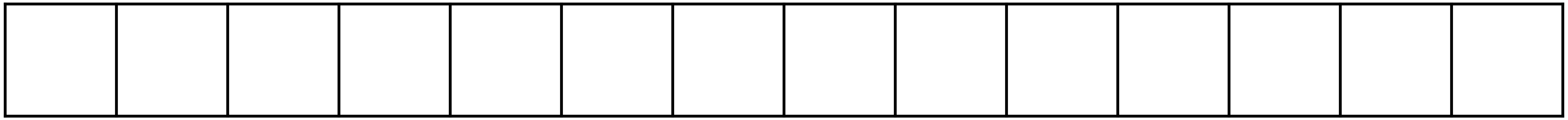
- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**

Time-Stamping is Hard

- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.

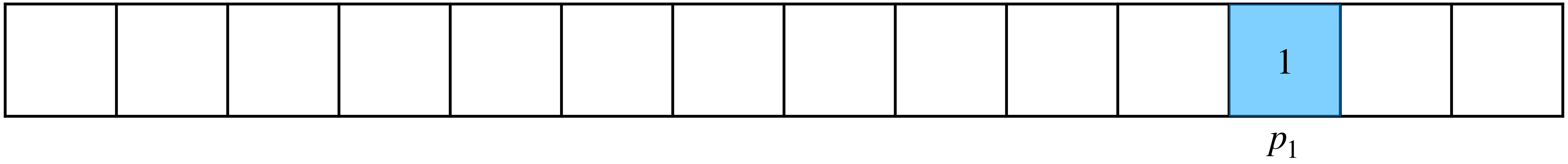
Time-Stamping is Hard

- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.
- Example: **Marking** (appears in oblivious hash tables in **PanORAMa** and **OptORAMa**)



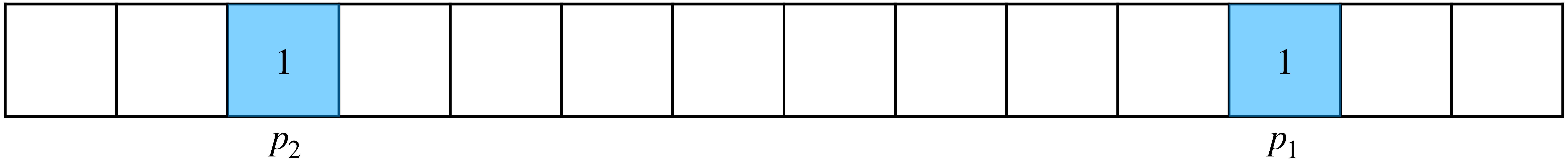
Time-Stamping is Hard

- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.
- Example: **Marking** (appears in oblivious hash tables in **PanORAMa** and **OptORAMa**)



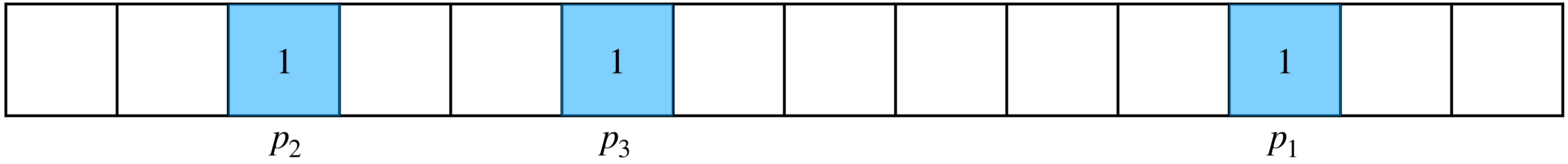
Time-Stamping is Hard

- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.
- Example: **Marking** (appears in oblivious hash tables in **PanORAMa** and **OptORAMa**)



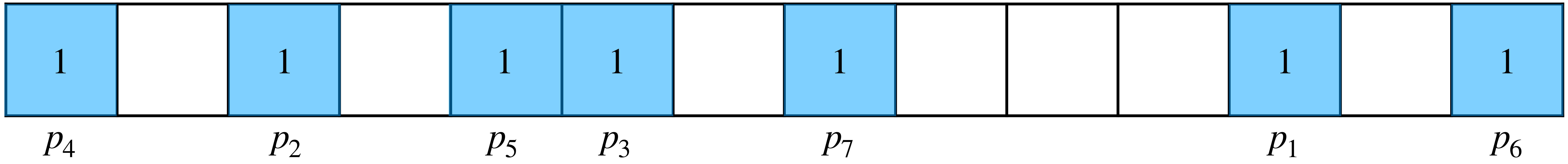
Time-Stamping is Hard

- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.
- Example: **Marking** (appears in oblivious hash tables in **PanORAMa** and **OptORAMa**)



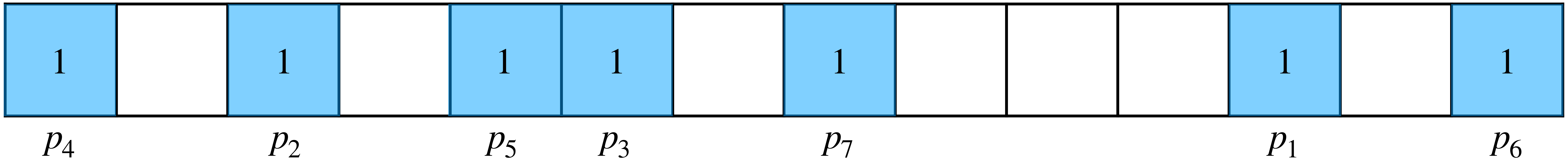
Time-Stamping is Hard

- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.
- Example: **Marking** (appears in oblivious hash tables in **PanORAMa** and **OptORAMa**)



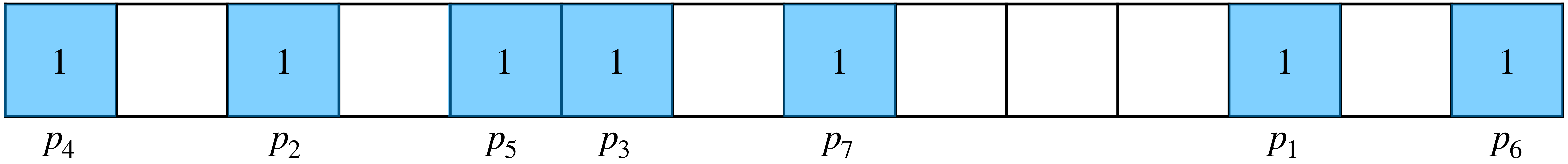
Time-Stamping is Hard

- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.
- Example: **Marking** (appears in oblivious hash tables in **PanORAMa** and **OptORAMa**)
  - Setup: Mark positions $p_i \in [N]$ as *visited* when given online way for $1 \leq i \leq N/2$.



Time-Stamping is Hard

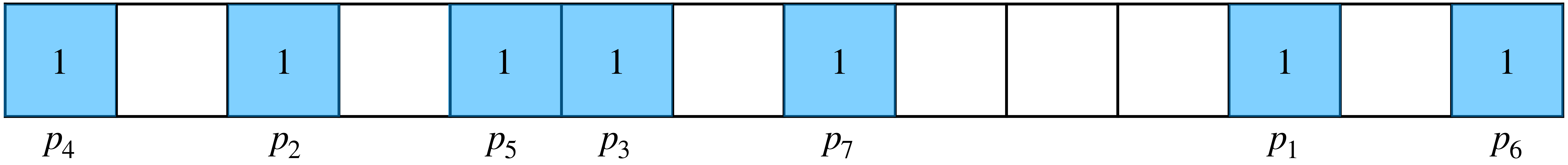
- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.
- Example: **Marking** (appears in oblivious hash tables in **PanORAMa** and **OptORAMa**)
  - Setup: Mark positions $p_i \in [N]$ as *visited* when given online way for $1 \leq i \leq N/2$.



  - If you can time-stamp this access pattern, you can recover all p_i.

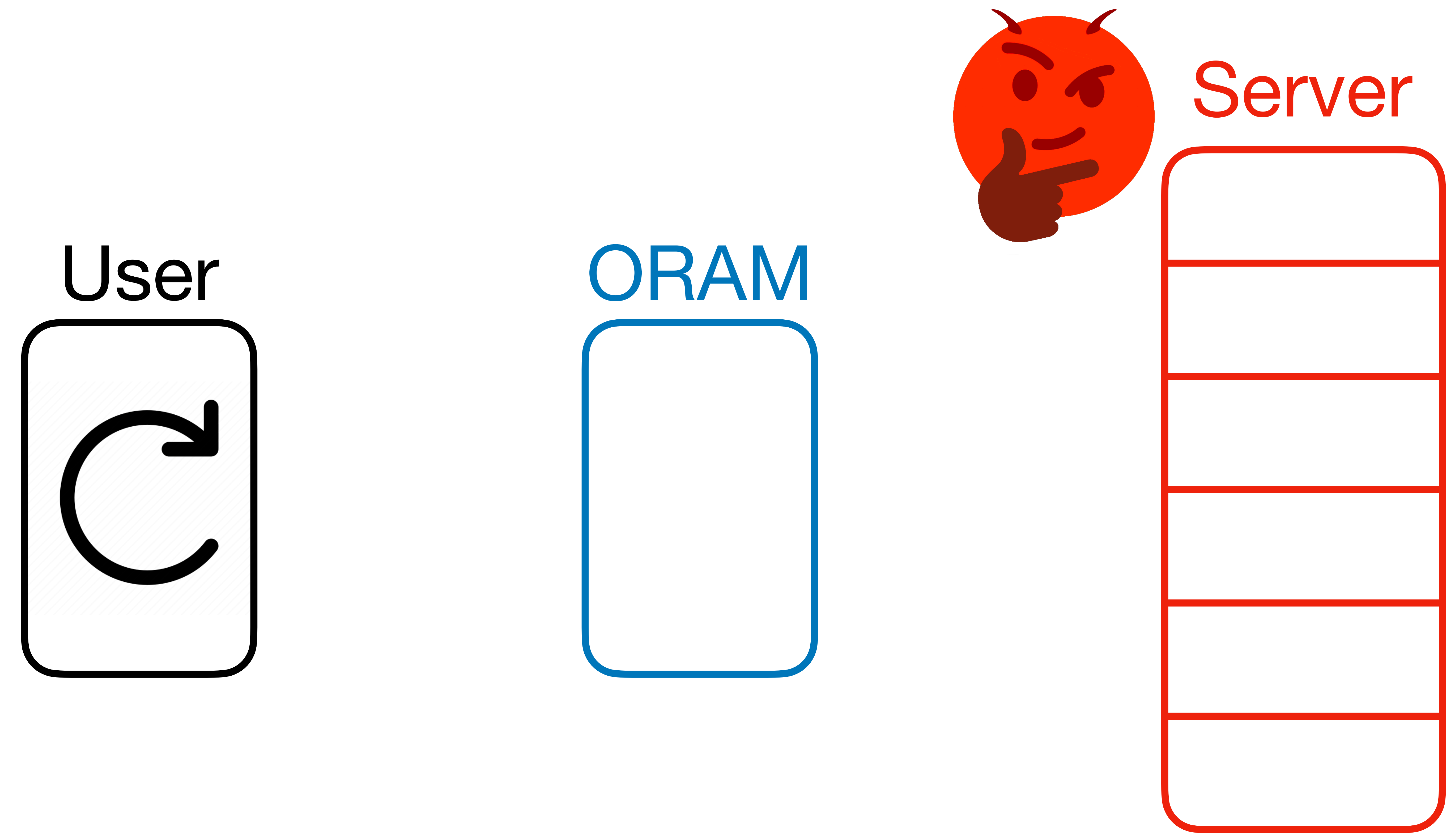
Time-Stamping is Hard

- Unfortunately, **the recent hierarchical ORAM constructions cannot be time-stamped**
- **Unconditionally** requires $\Omega(N)$ bits of local space to time-stamp **OptORAMa**.
- Example: **Marking** (appears in oblivious hash tables in **PanORAMa** and **OptORAMa**)
  - Setup: Mark positions $p_i \in [N]$ as *visited* when given online way for $1 \leq i \leq N/2$.



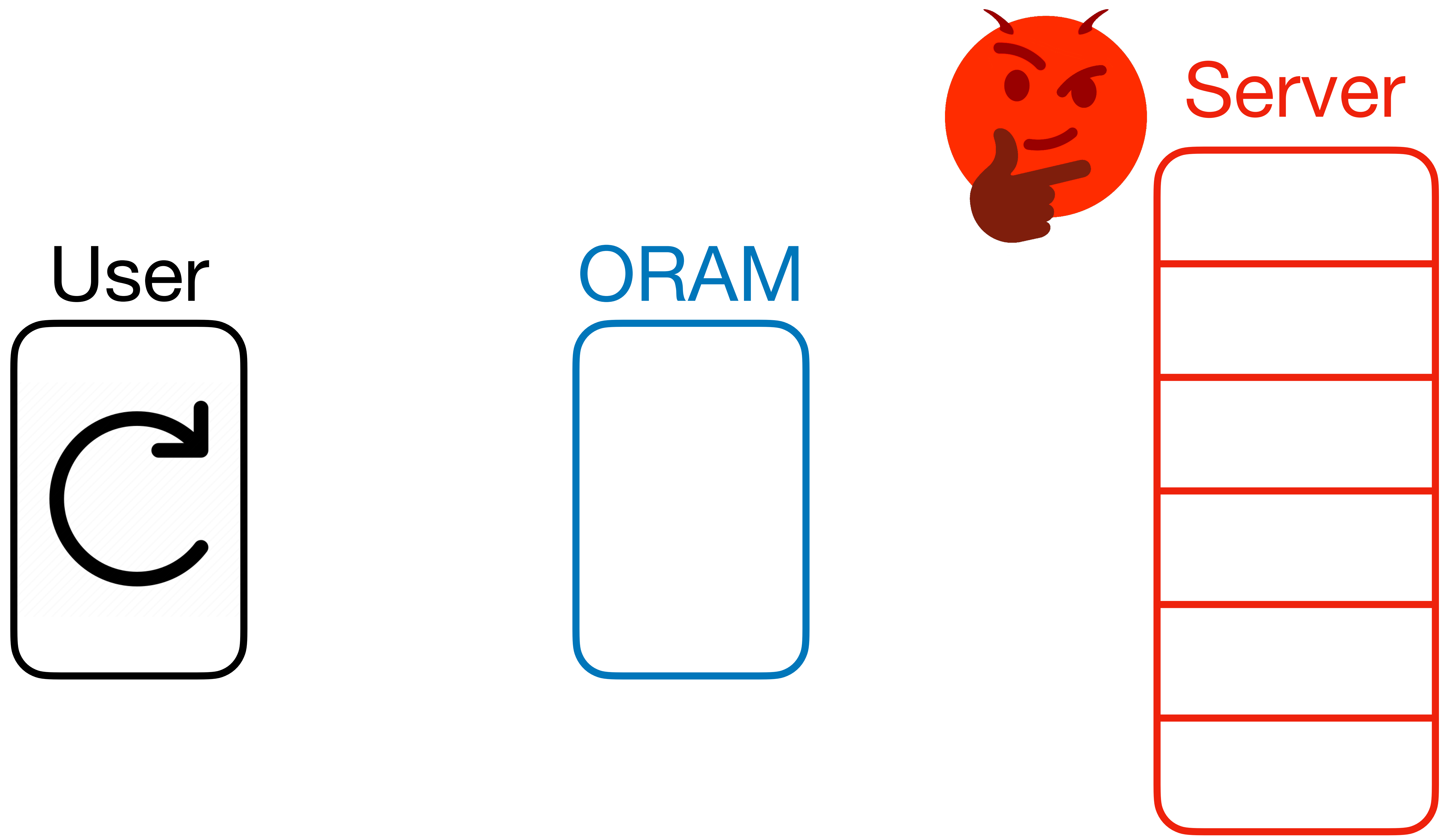
  - If you can time-stamp this access pattern, you can recover all p_i.
  - Random sequence of p_i has entropy $\Theta(N \log N)$, so no way to time-stamp with even $O(N)$ bits of space, let alone $O(\log N)$ bits.

Does OptORAMa really need memory checking?



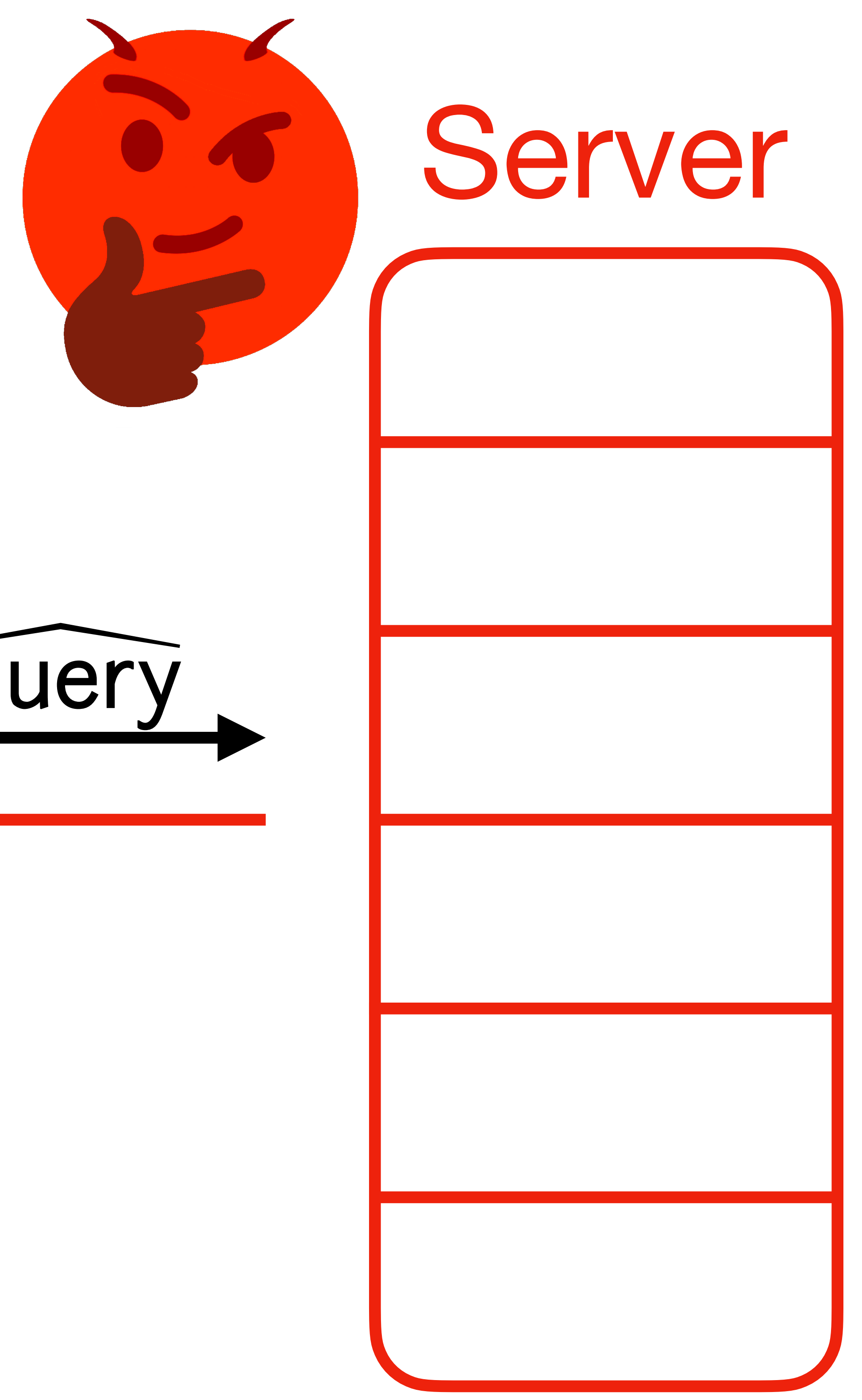
Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate *some* lies from the server?



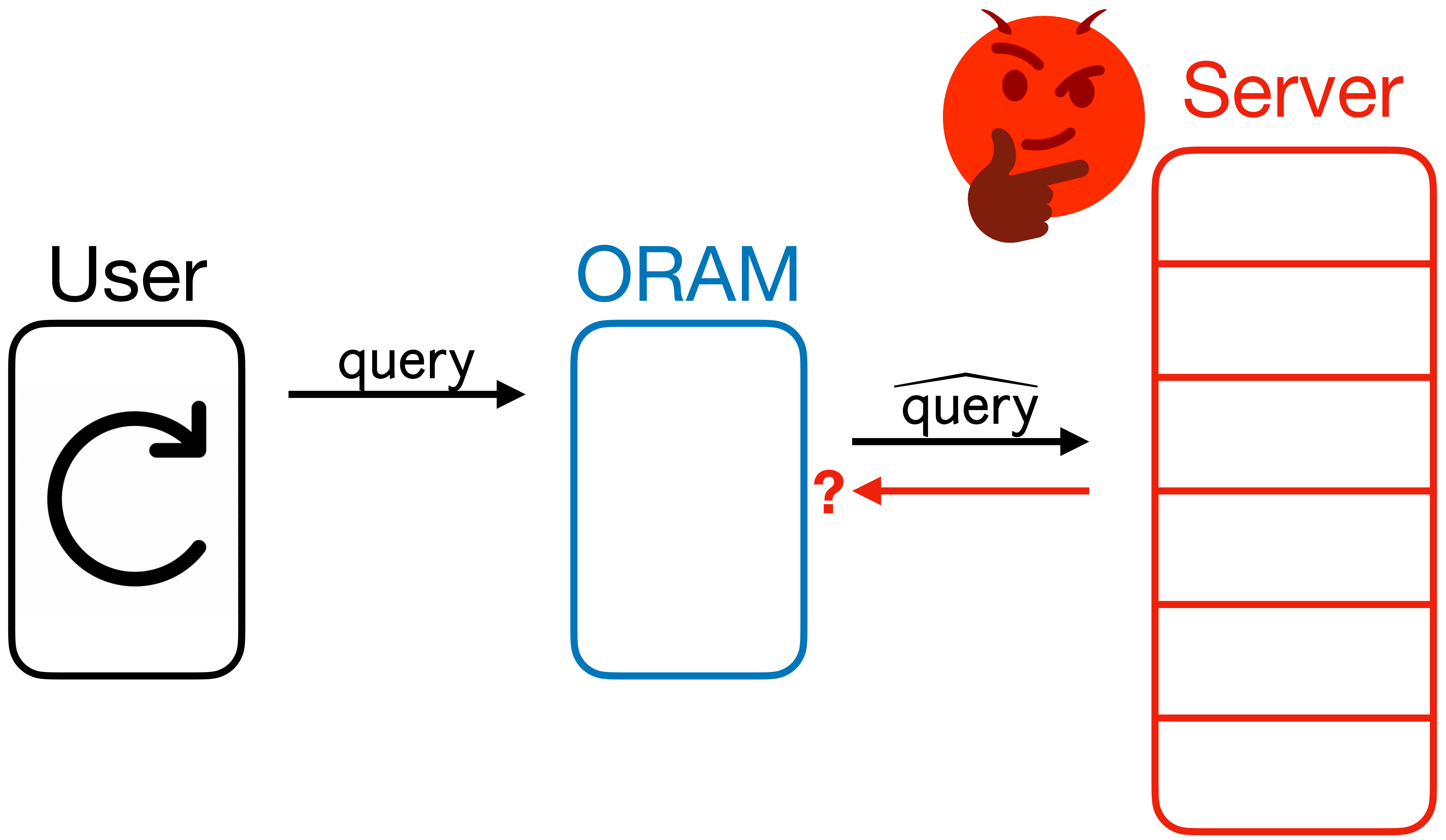
Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate *some* lies from the server?



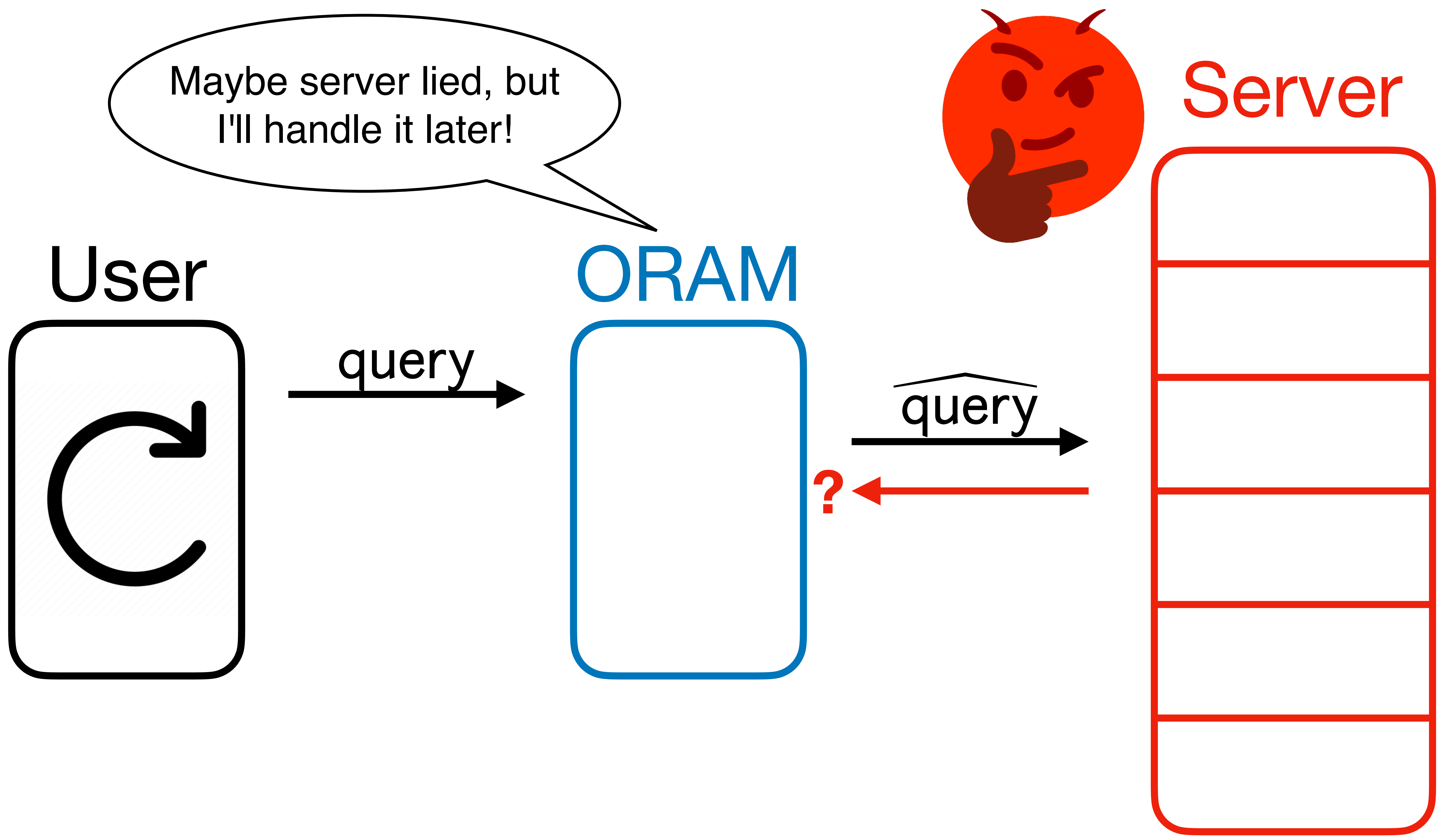
Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate *some* lies from the server?



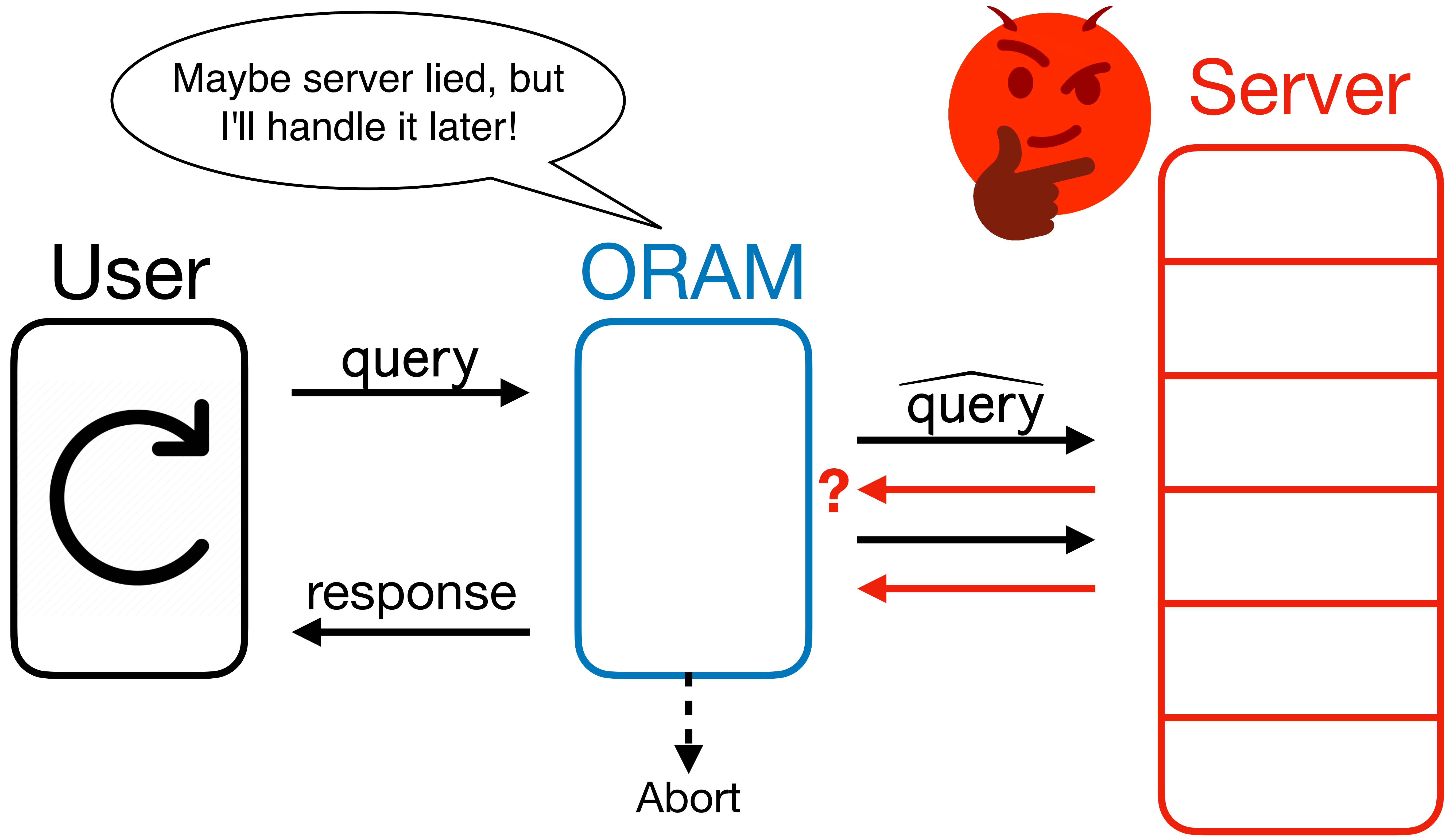
Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate *some* lies from the server?



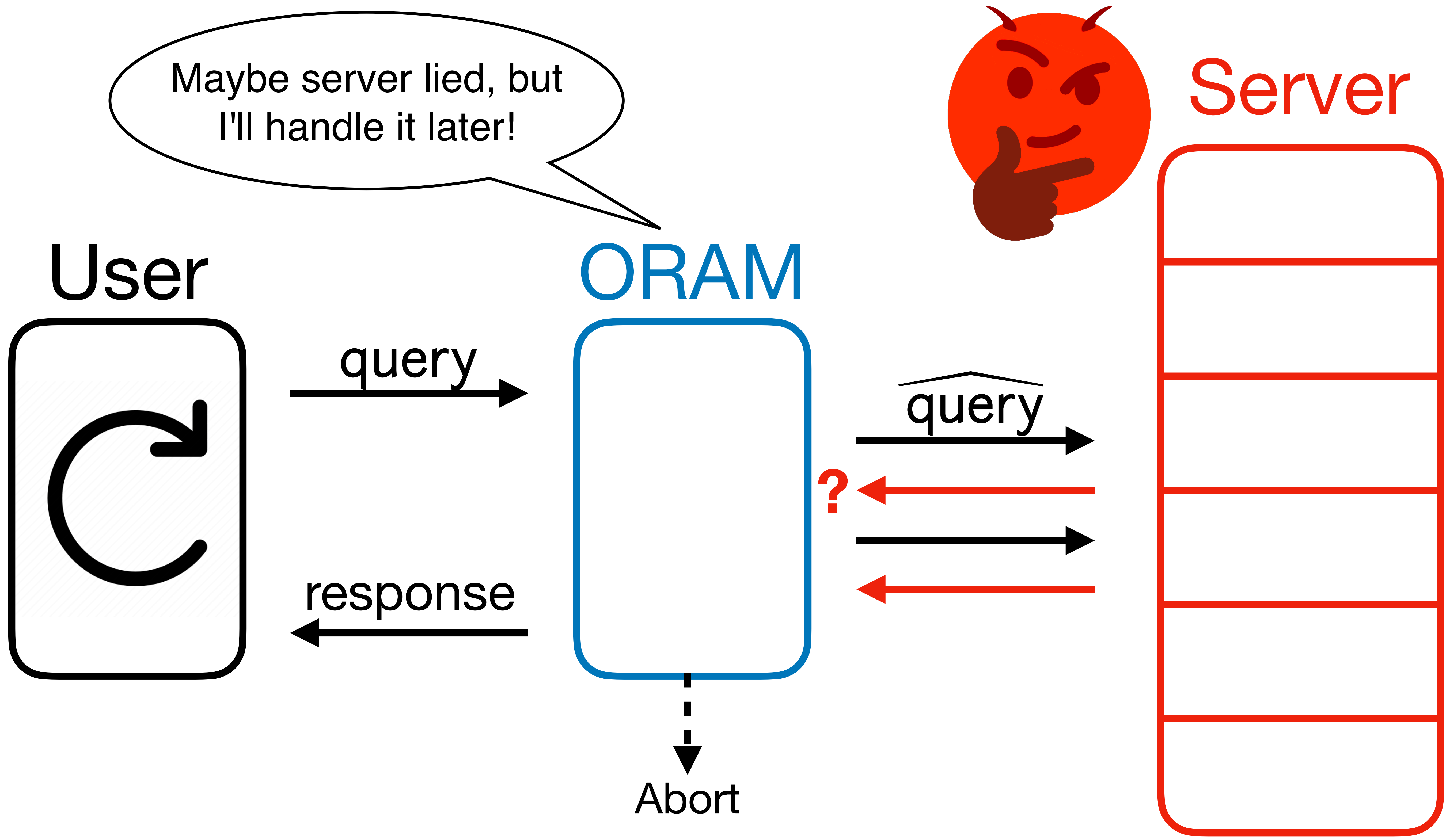
Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate *some* lies from the server?



Does OptORAMa really need memory checking?

- What if **OptORAMa** can tolerate *some* lies from the server?
- **Our Idea**: Use weaker, more efficient notion of memory checking to capitalize on this!



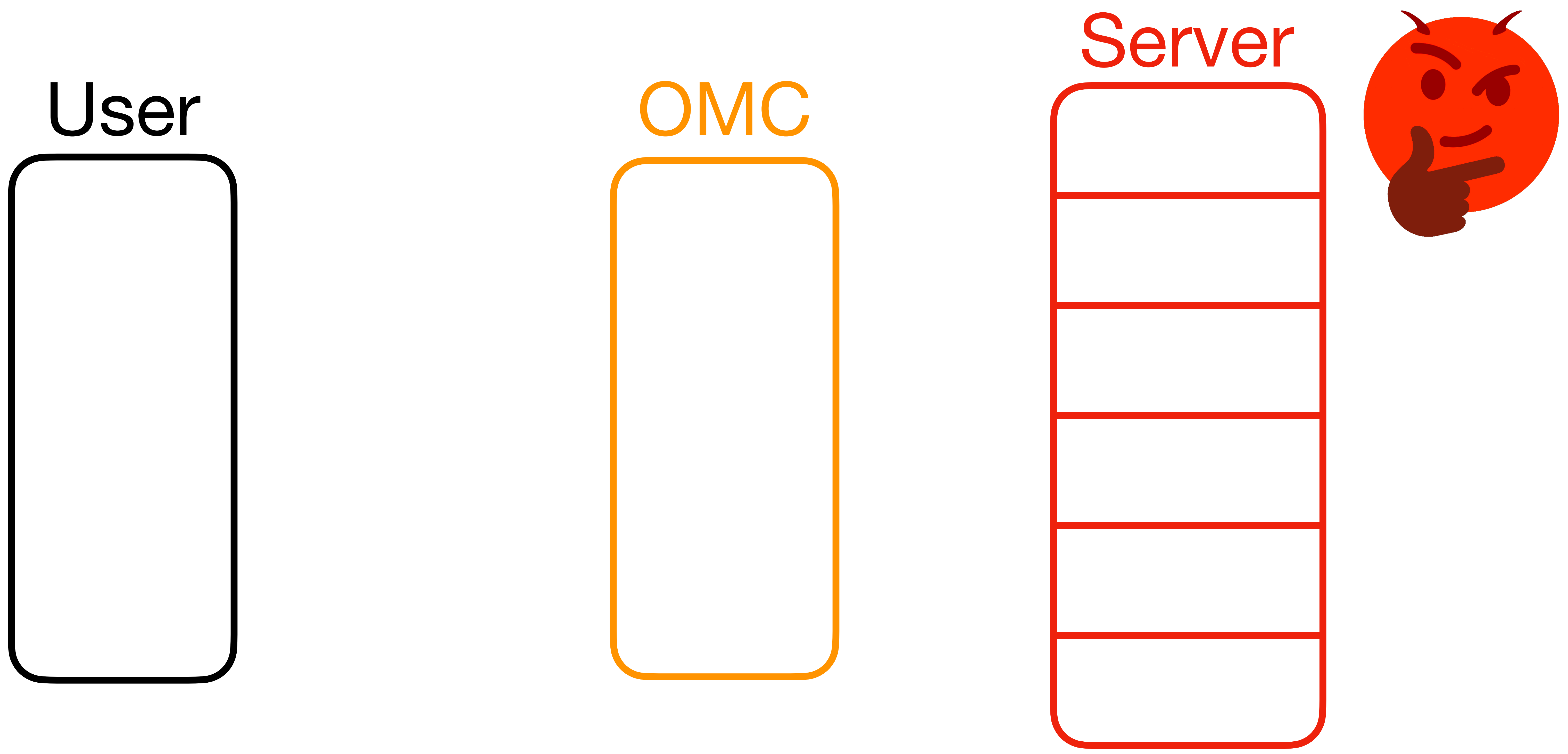
Offline Memory Checking

Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]

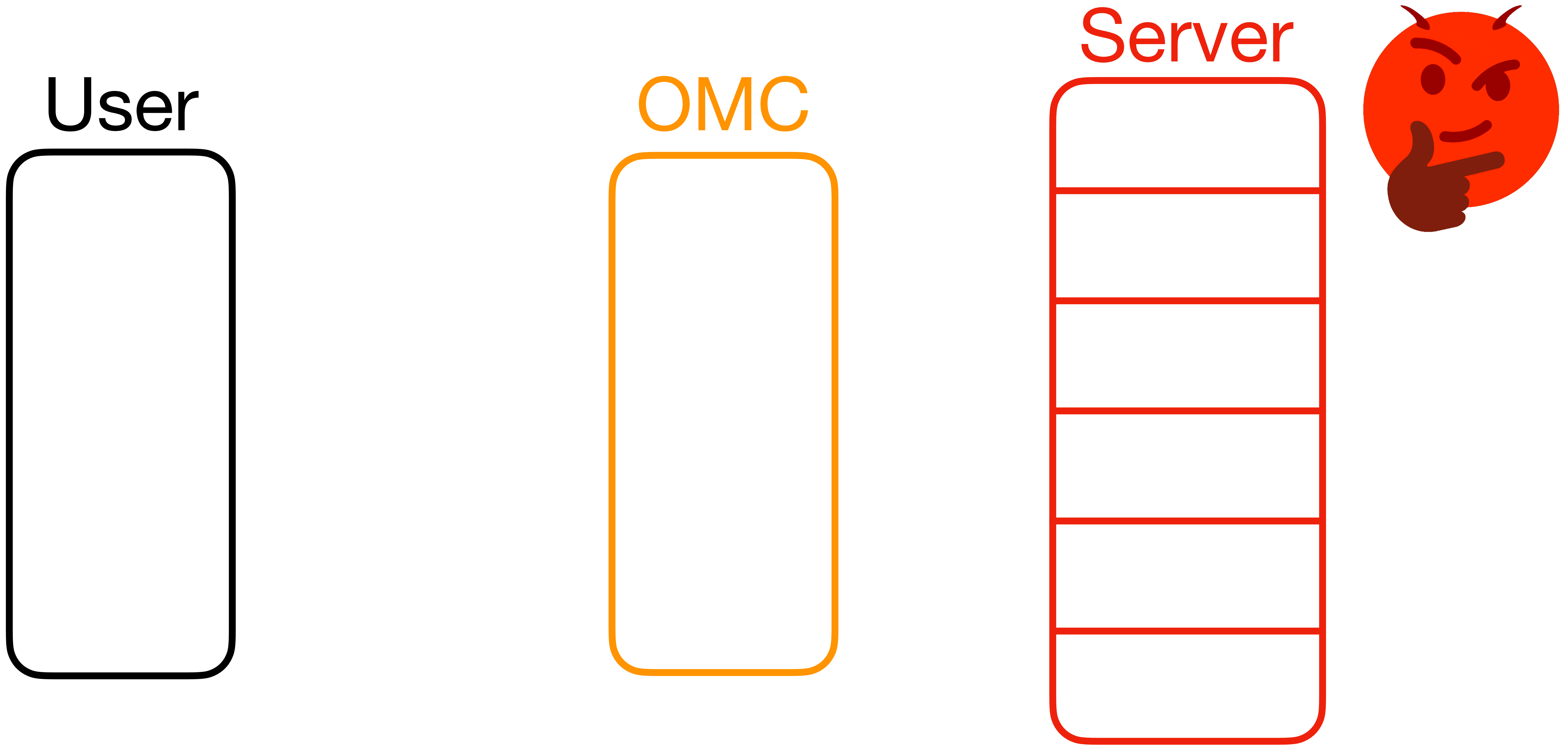
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]



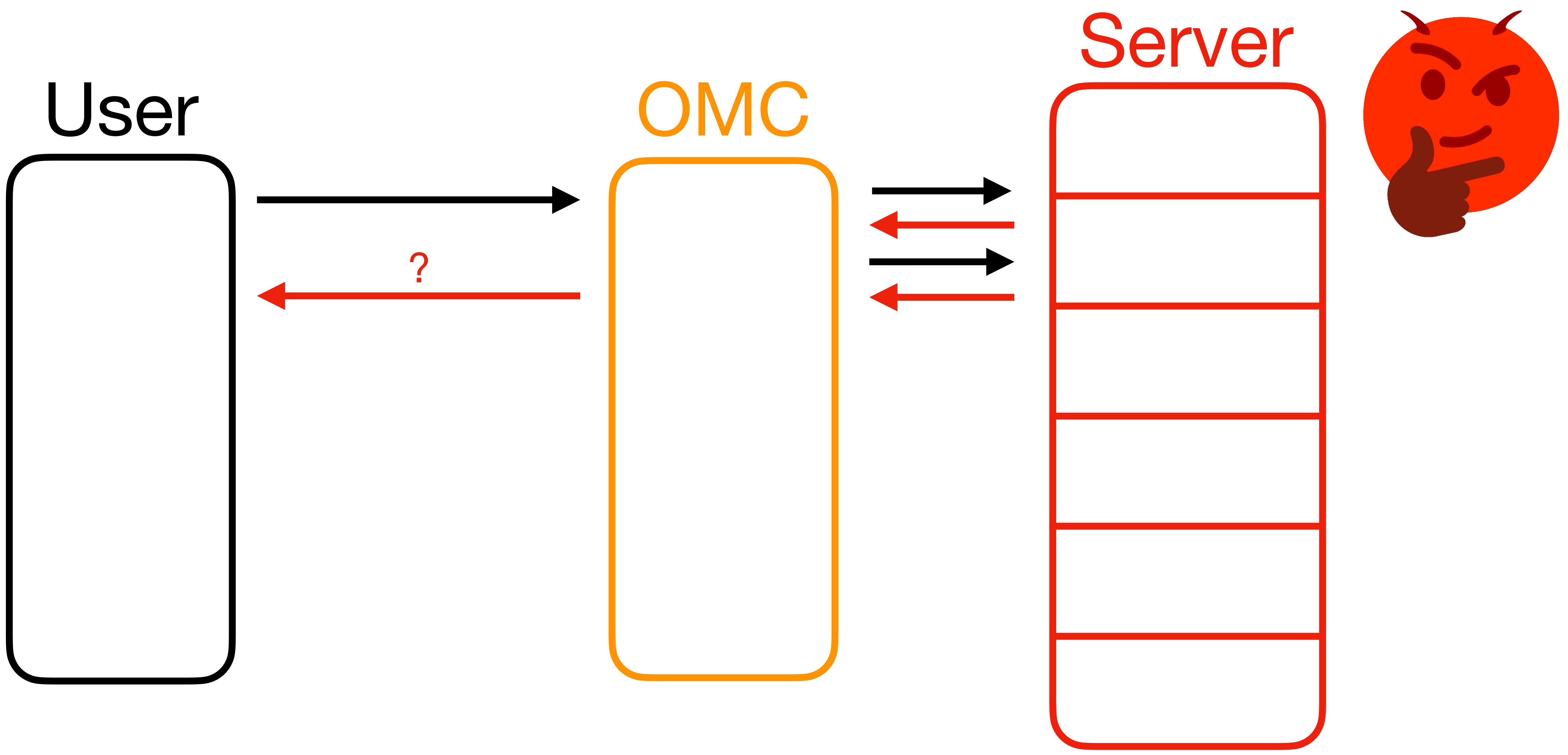
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



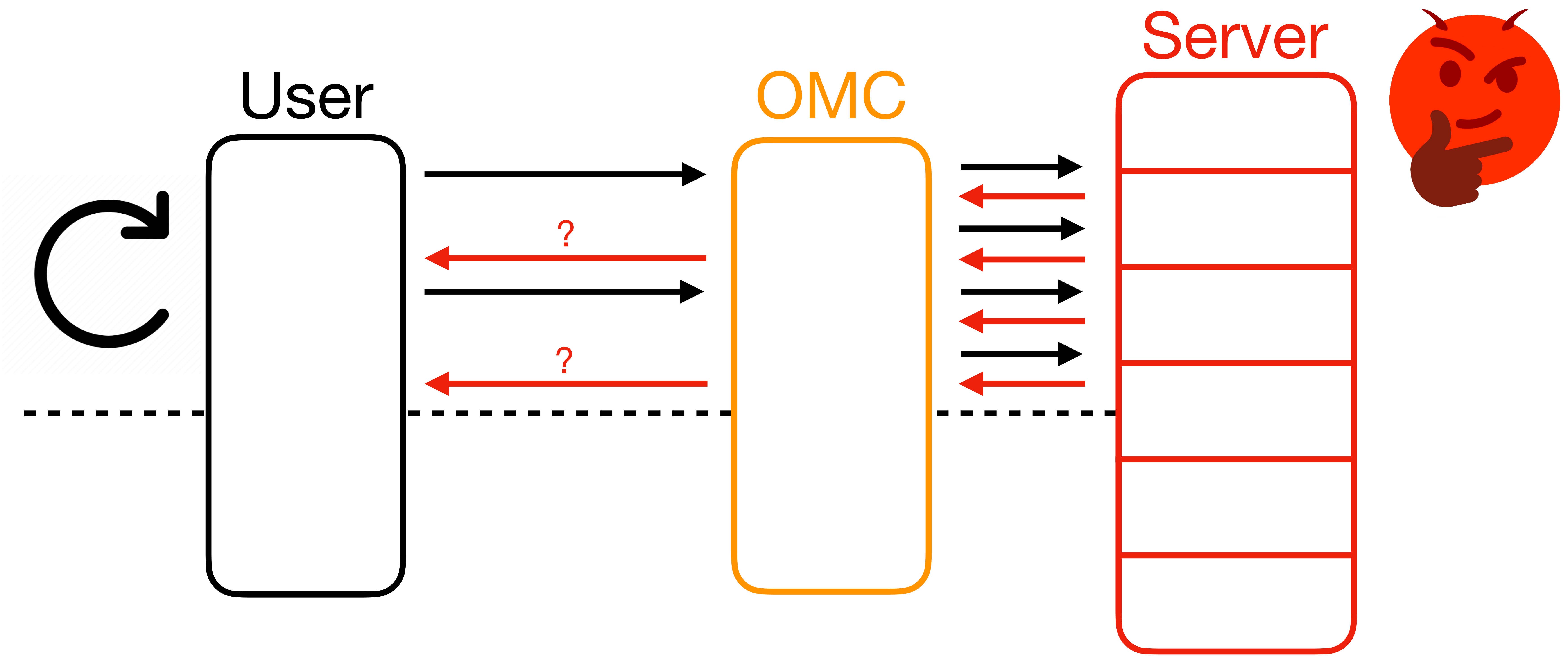
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



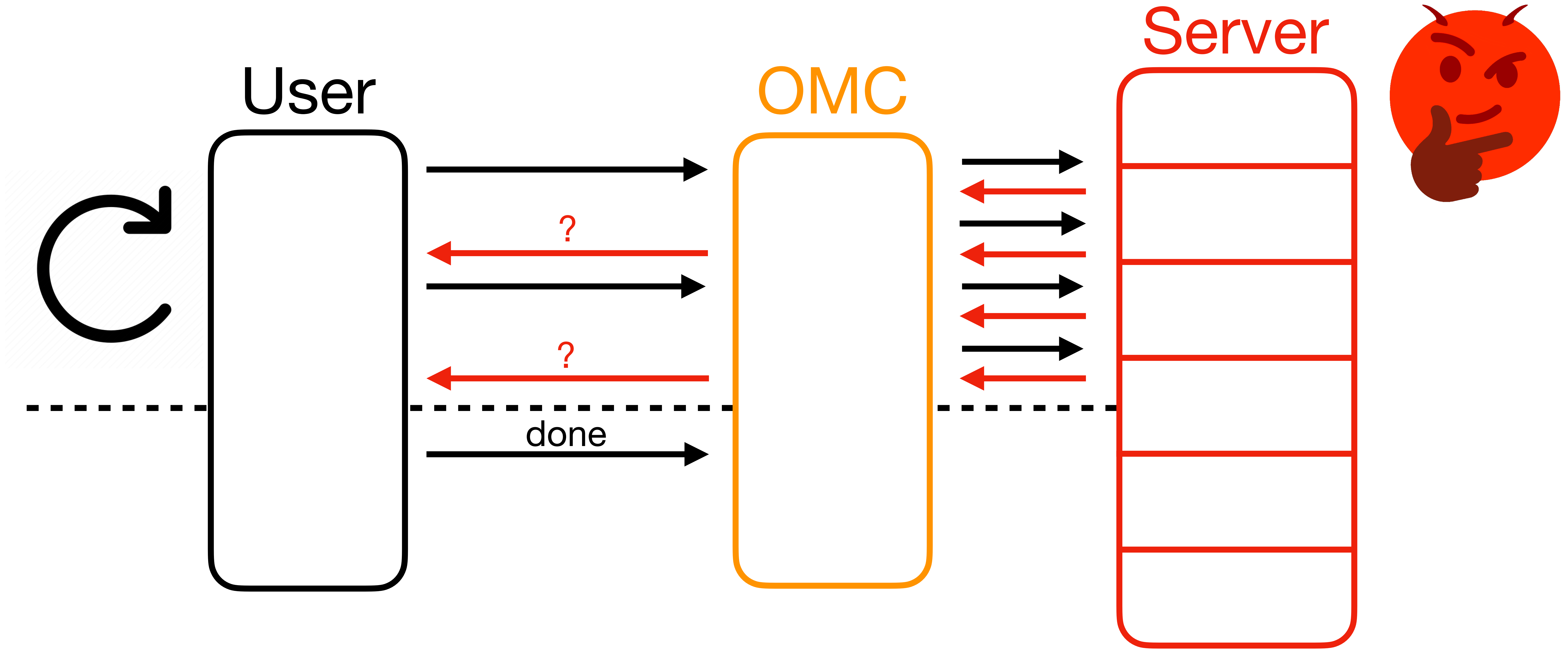
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



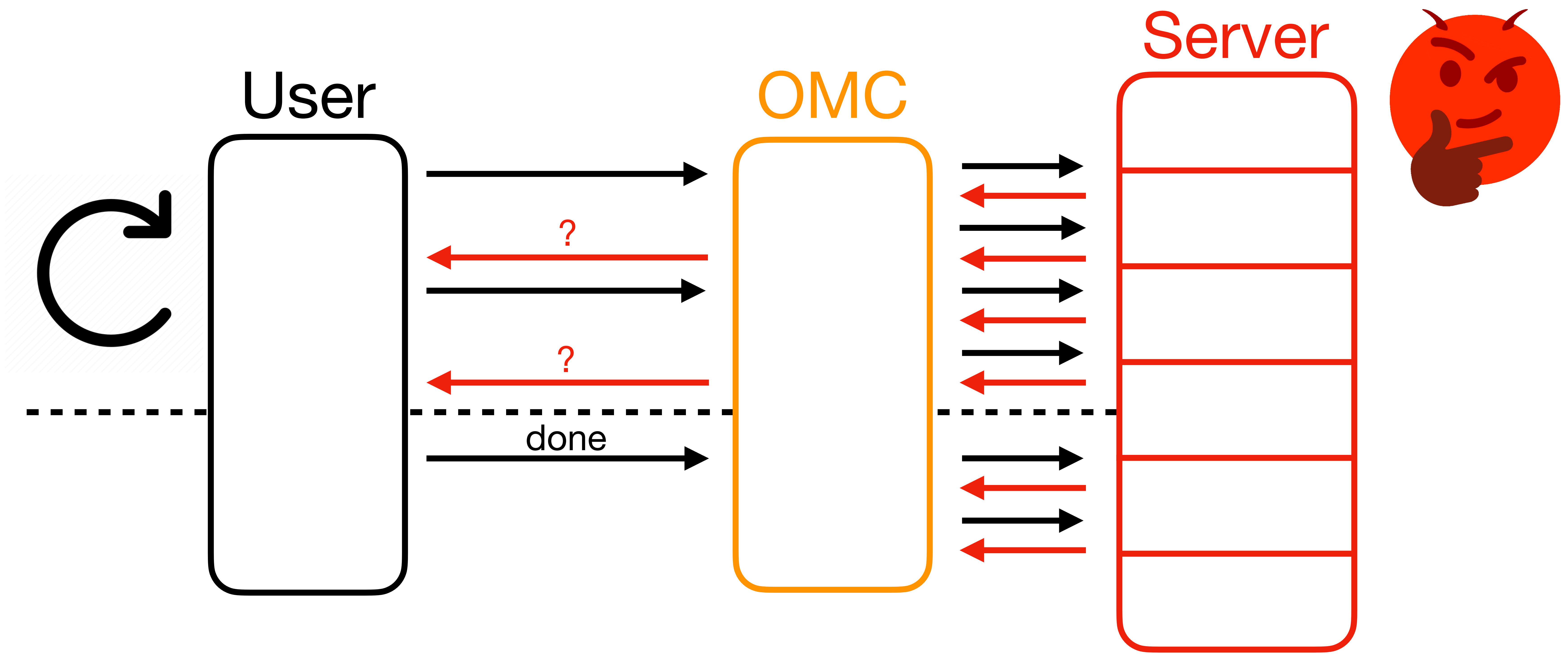
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think “batching” a regular memory checker.)



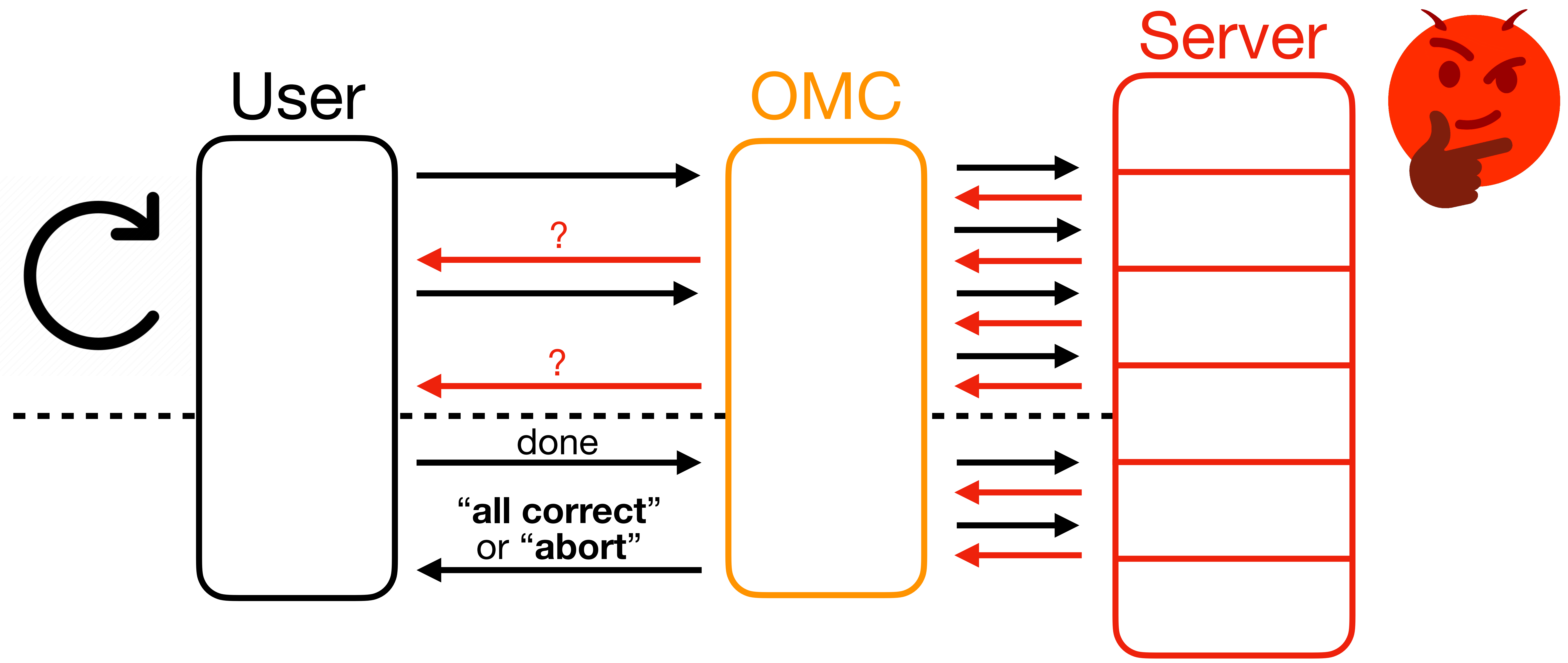
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think “batching” a regular memory checker.)



Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**

[Blum et al. ’94]
[Dwork et al. ’09]

Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**
- Con of offline memory checking: **insufficient**! **Insecure for OptORAMa.**

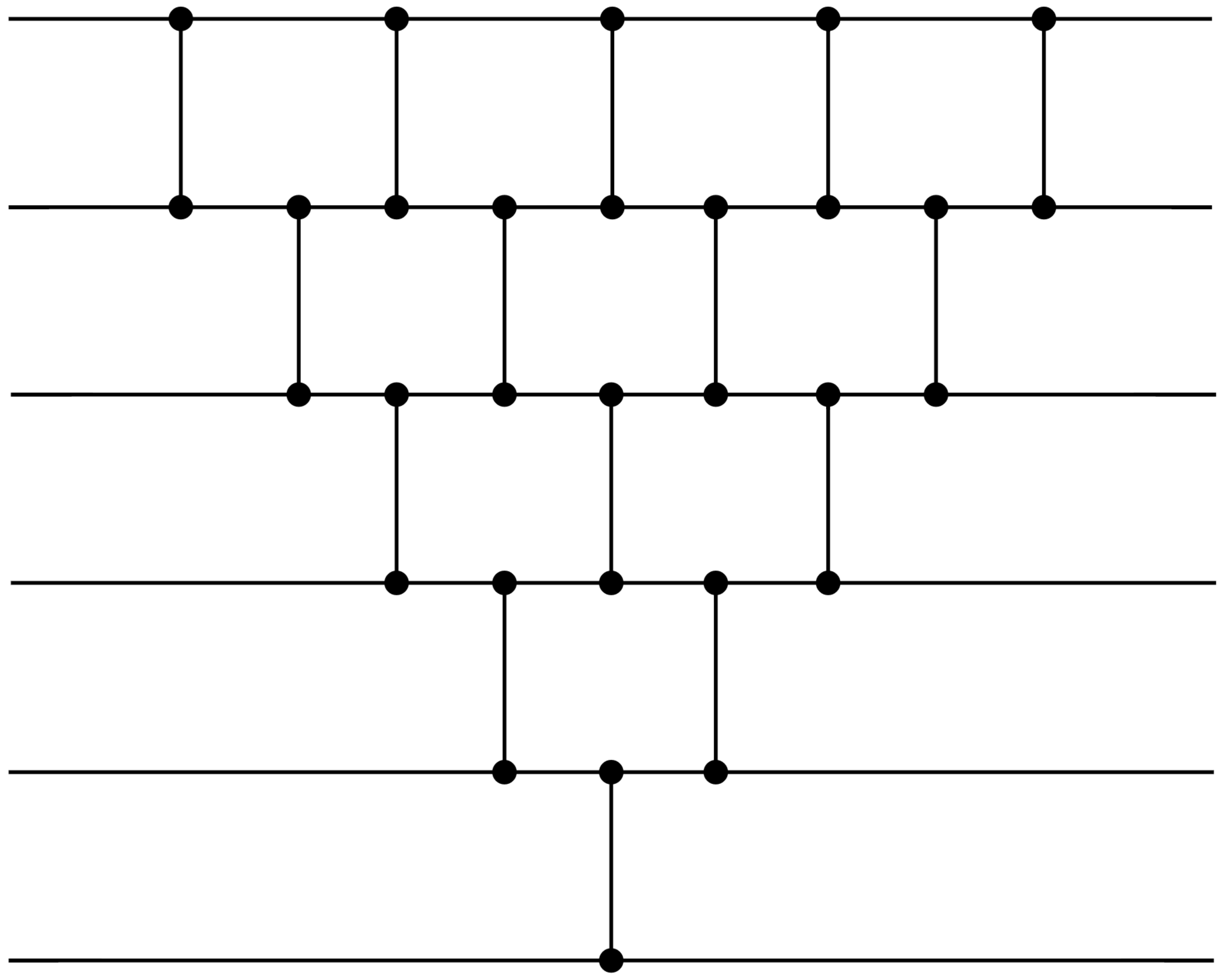
Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**
- Con of offline memory checking: **insufficient**! **Insecure for OptORAMa**.
  - Replay attack (with MACs and offline memory checking) **still applies.**

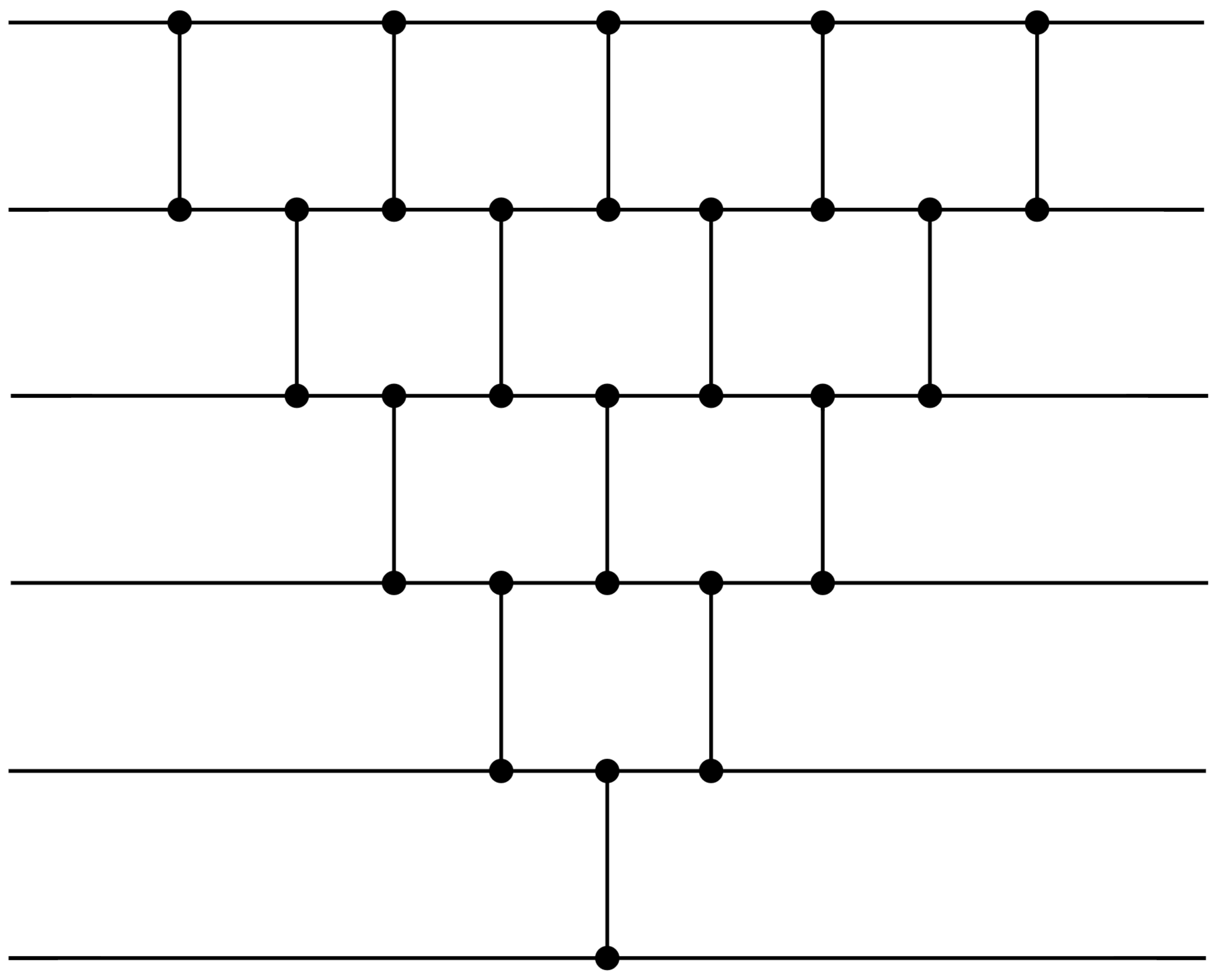
Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**
- Con of offline memory checking: **insufficient**! **Insecure for OptORAMa**.
  - Replay attack (with MACs and offline memory checking) **still applies.**
- So when is offline checking safe?

When is Offline Checking Safe?

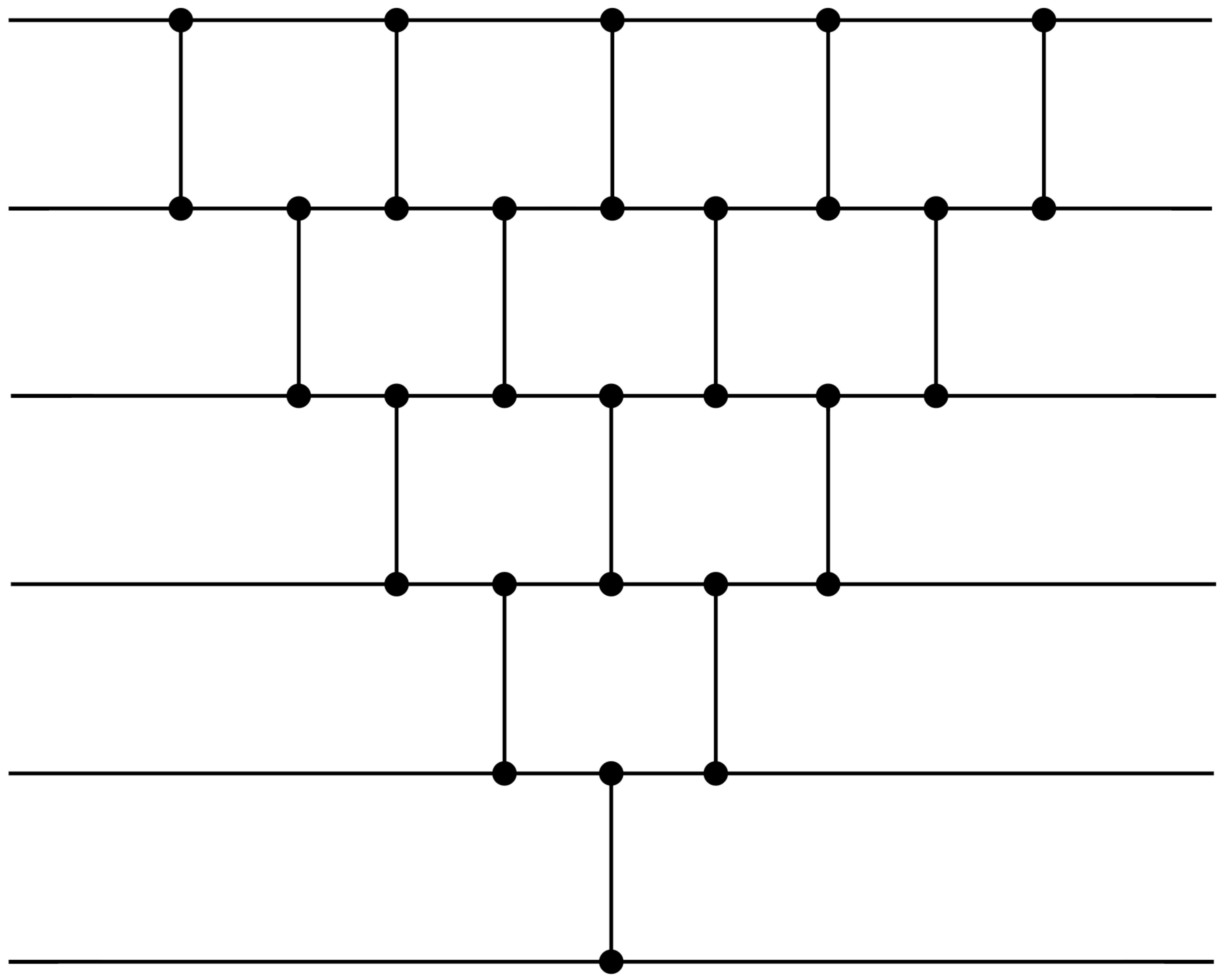


When is Offline Checking Safe?



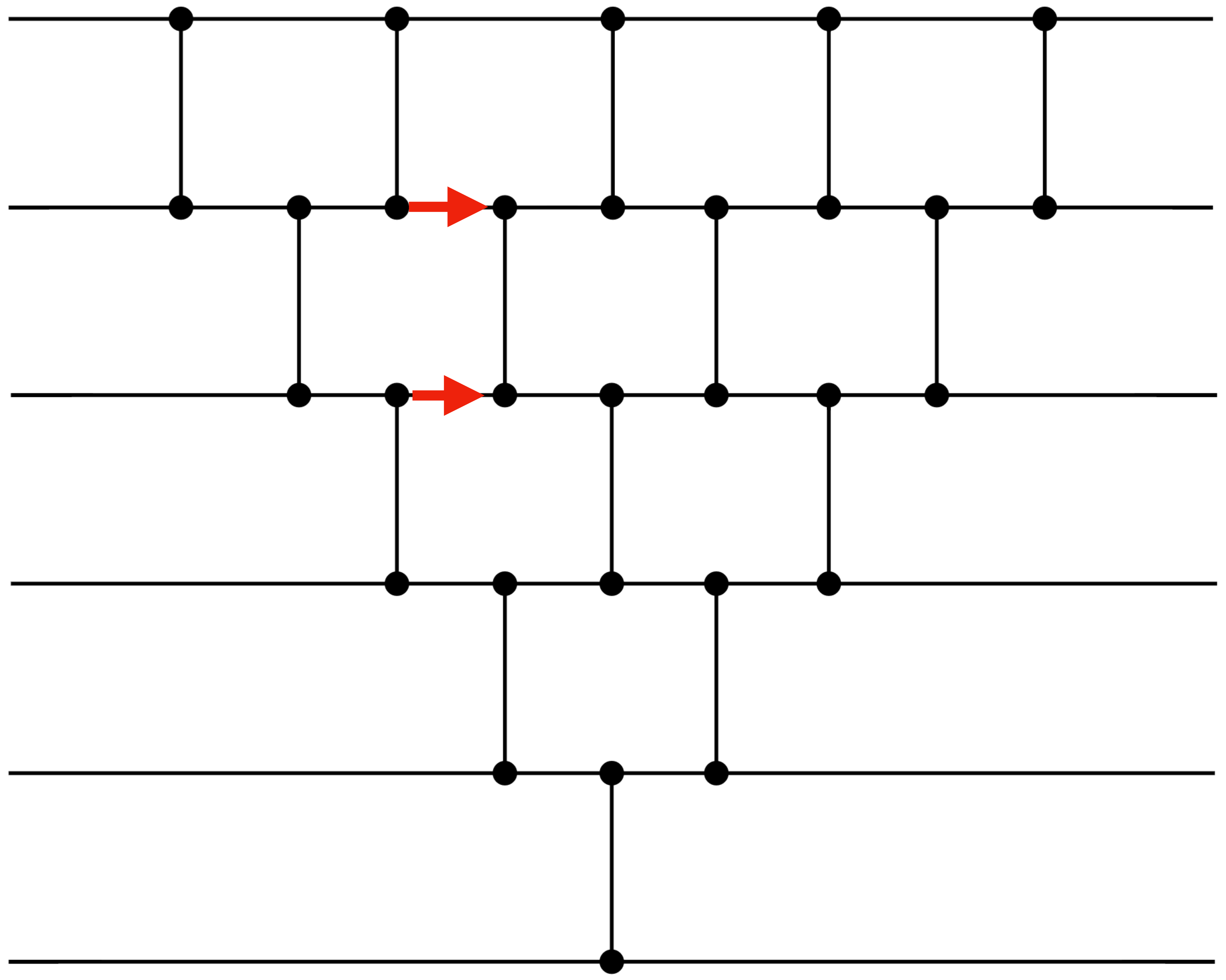
- Eg. Simple sorting networks (e.g. Batcher’s)

When is Offline Checking Safe?



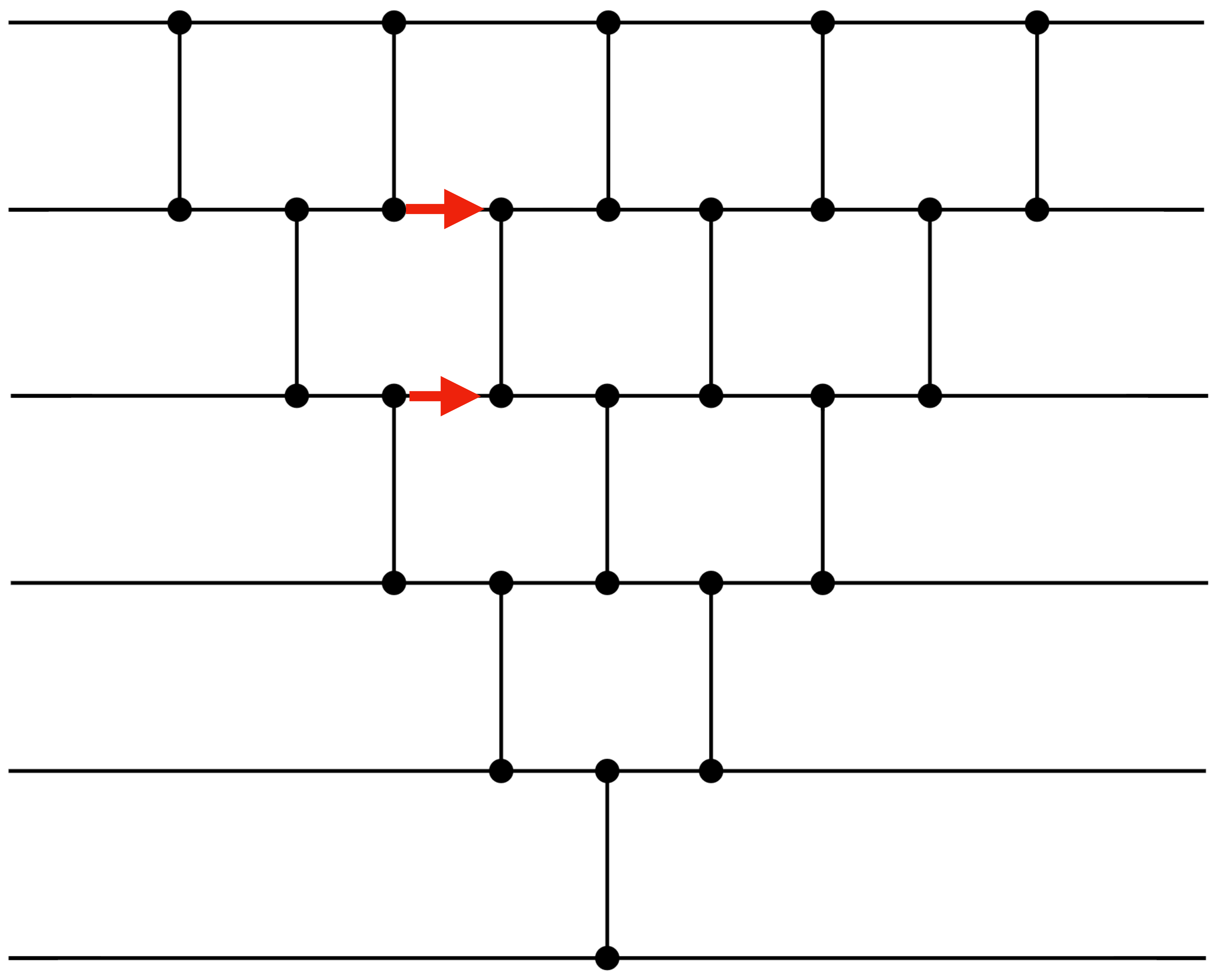
- Eg. Simple sorting networks (e.g. Batcher’s)
- Can locally compute all comparisons to be made.

When is Offline Checking Safe?



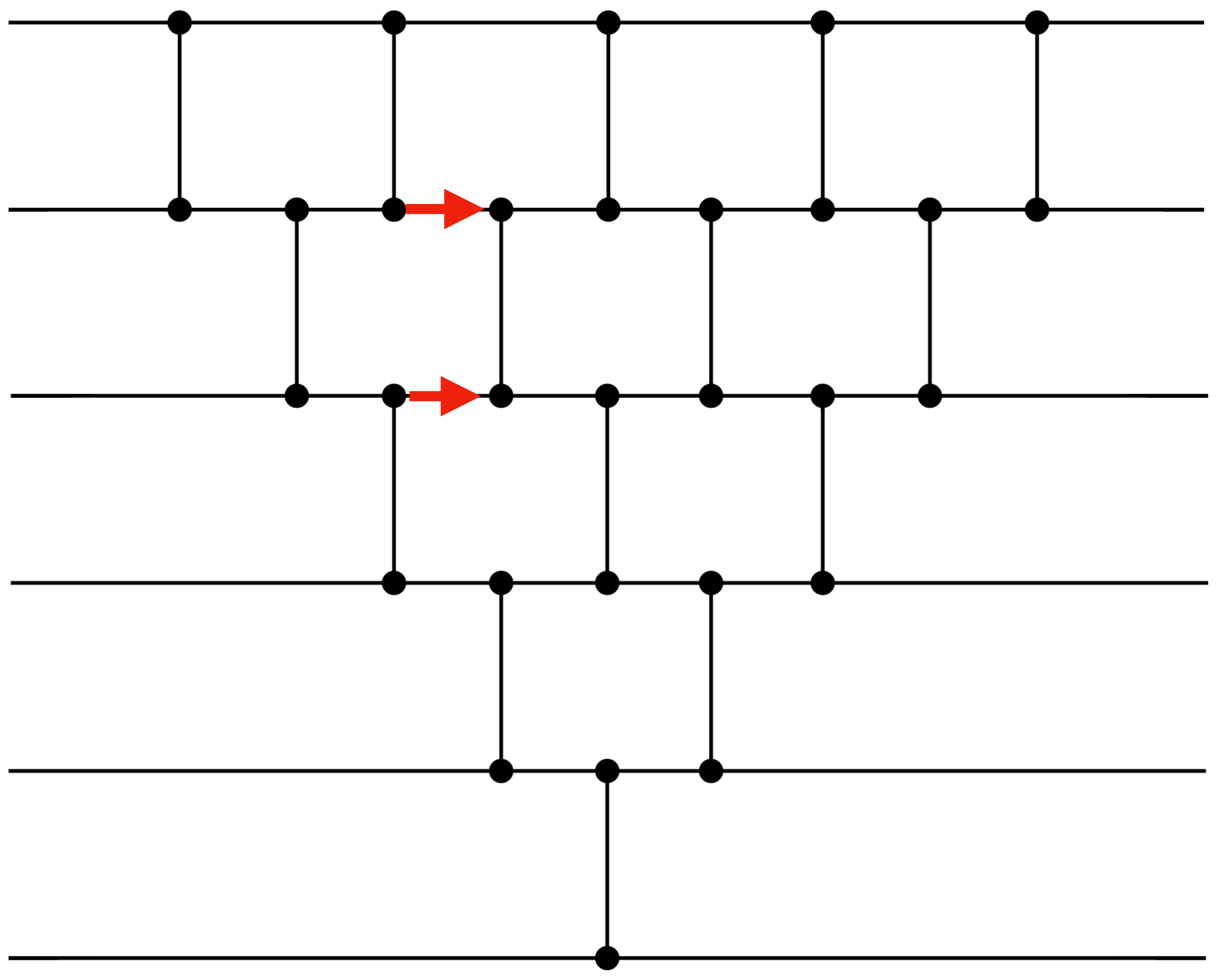
- Eg. Simple sorting networks (e.g. Batcher's)
- Can locally compute all comparisons to be made.
- Incorrect wire values do not affect the comparisons made, so access pattern is **not affected.**

When is Offline Checking Safe?



- Eg. Simple sorting networks (e.g. Batcher's)
- Can locally compute all comparisons to be made.
- Incorrect wire values do not affect the comparisons made, so access pattern is **not affected.**
- Safe to offline-check!

When is Offline Checking Safe?



- Eg. Simple sorting networks (e.g. Batcher's)
- Can locally compute all comparisons to be made.
- Incorrect wire values do not affect the comparisons made, so access pattern is **not affected.**
- Safe to offline-check!
- In our work, we generalise this further to capture more classes of algorithms.

Making OptORAMa Maliciously Secure

Making OptORAMa Maliciously Secure

- In an ideal world:

Making OptORAMa Maliciously Secure

- In an ideal world:
  - **Time-stamp** whatever you can using MACs (with no overhead).

Making OptORAMa Maliciously Secure

- In an ideal world:
  - **Time-stamp** whatever you can using MACs (with no overhead).
  - Hope that everything else in **OptORAMa** is **offline-safe** or **access-deterministic**.

Making OptORAMa Maliciously Secure

- In an ideal world:
  - **Time-stamp** whatever you can using MACs (with no overhead).
  - Hope that everything else in **OptORAMa** is **offline-safe** or **access-deterministic**.
- Unfortunately, this isn’t true.

Making OptORAMa Maliciously Secure

- In an ideal world:
  - **Time-stamp** whatever you can using MACs (with no overhead).
  - Hope that everything else in **OptORAMa** is **offline-safe** or **access-deterministic**.
- Unfortunately, this isn't true.
  - Oblivious hash table of **OptORAMa** is **not time-stampable** or **offline-safe**.

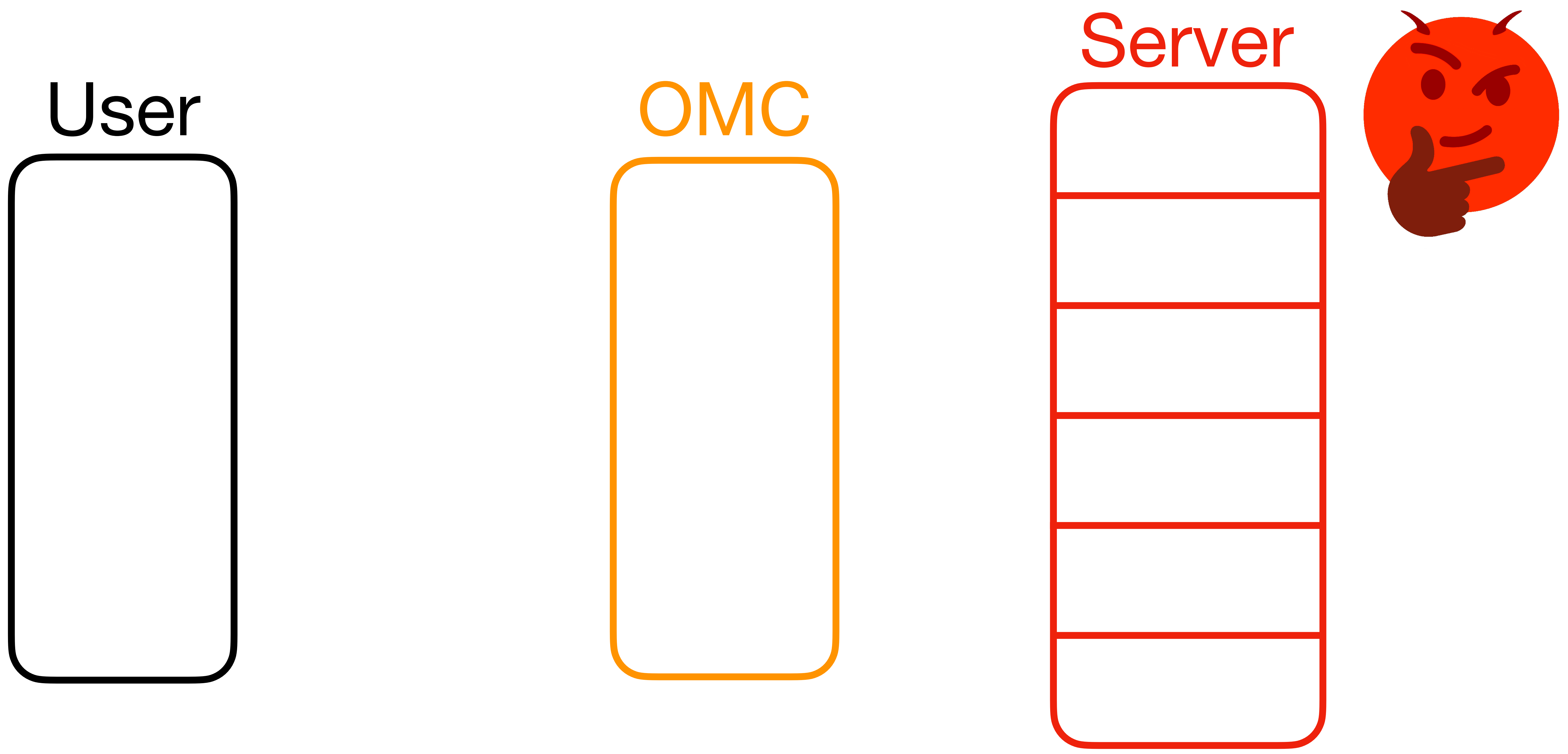
Offline Memory Checking

Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]

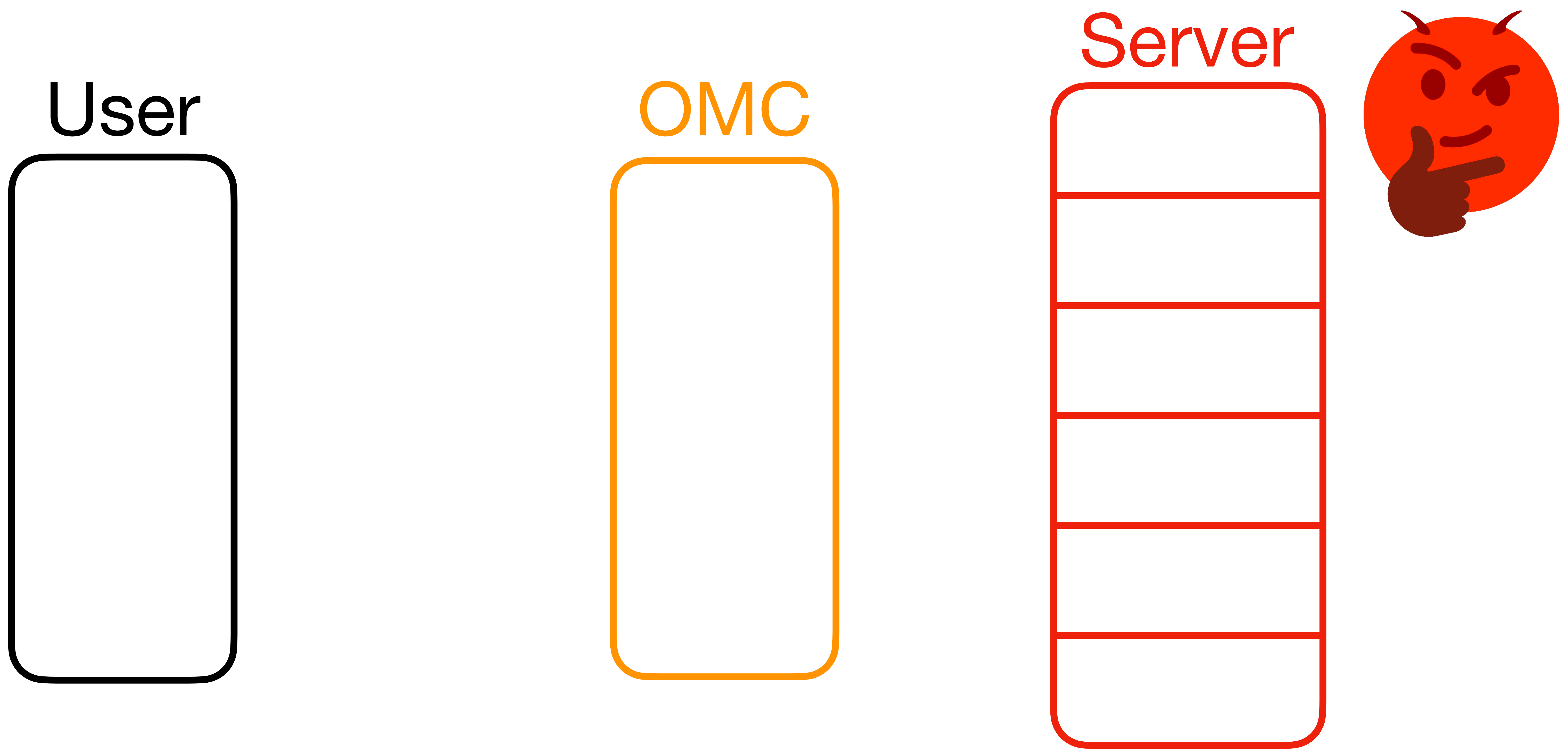
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]



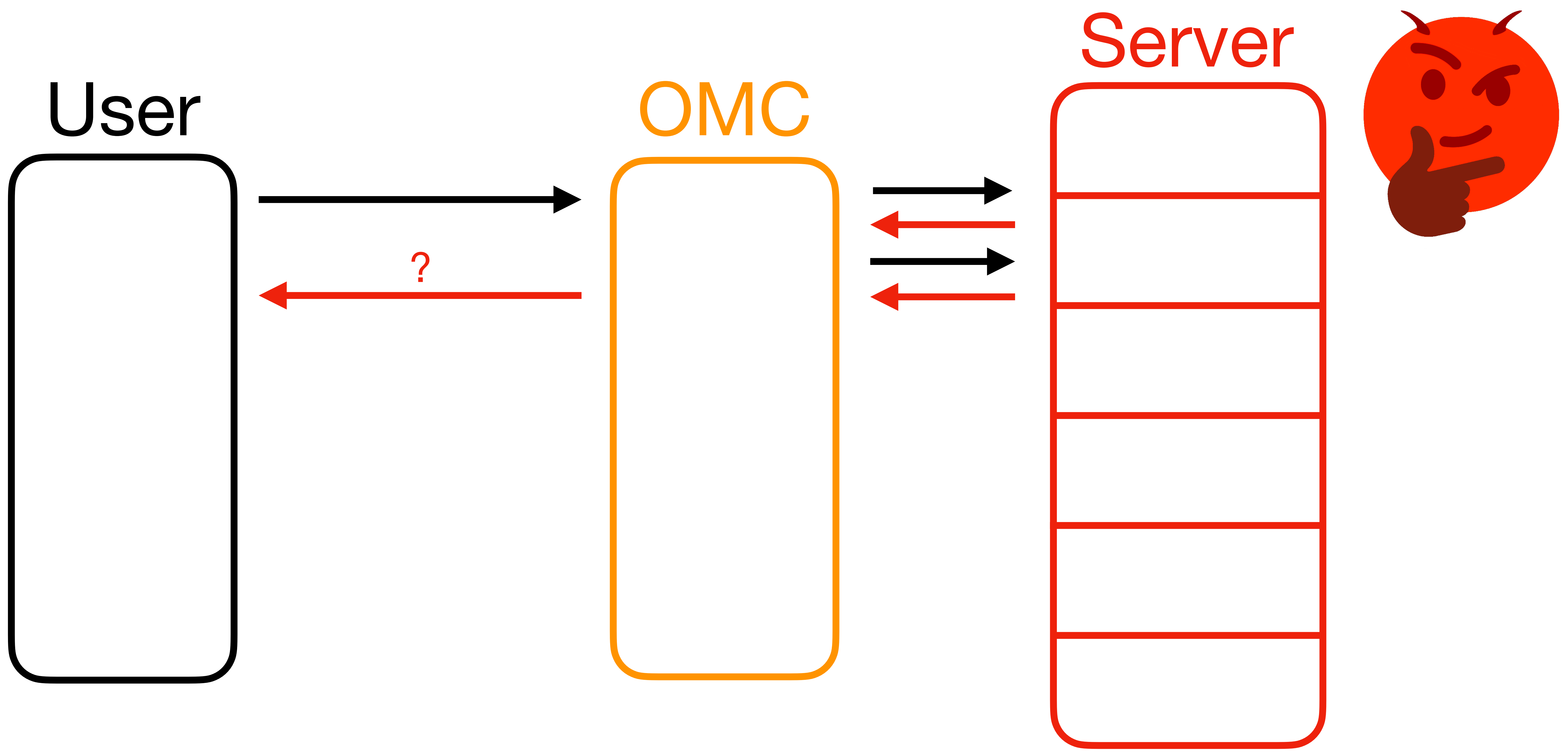
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



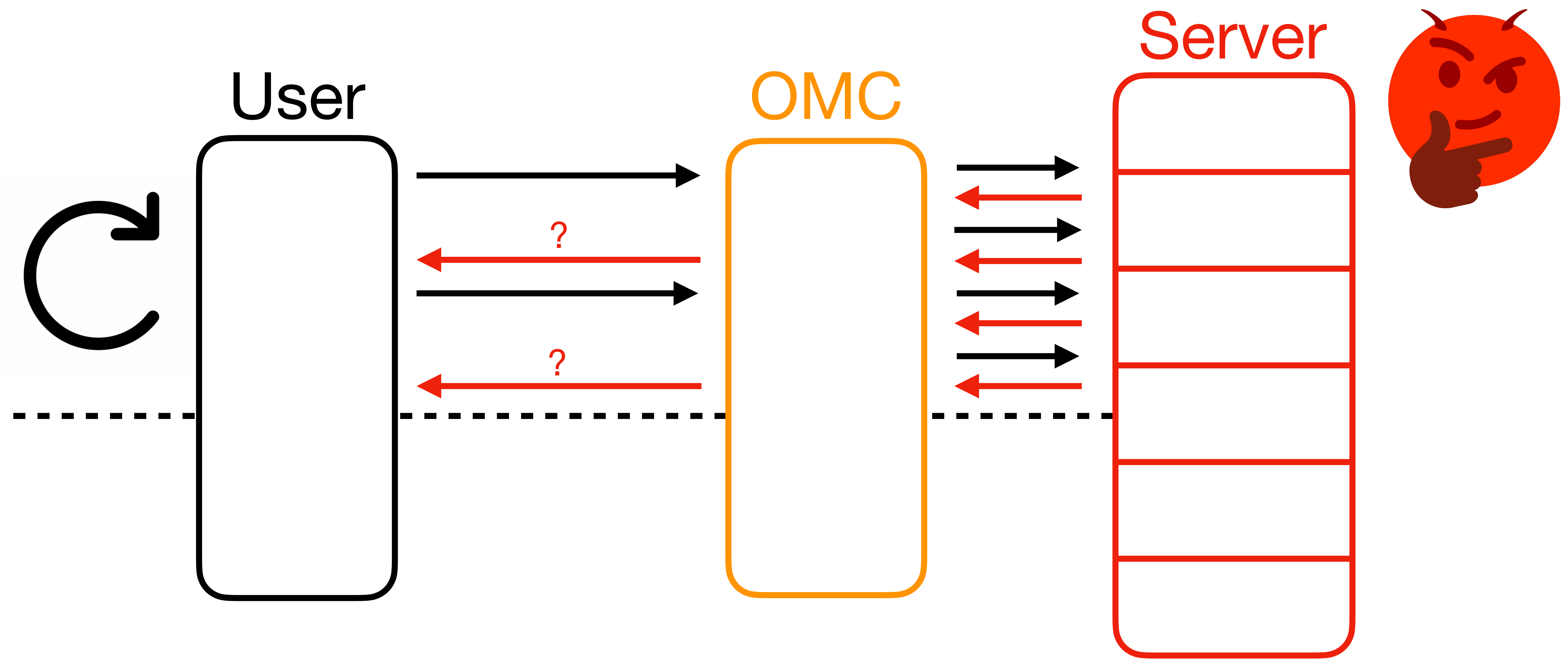
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think “batching” a regular memory checker.)



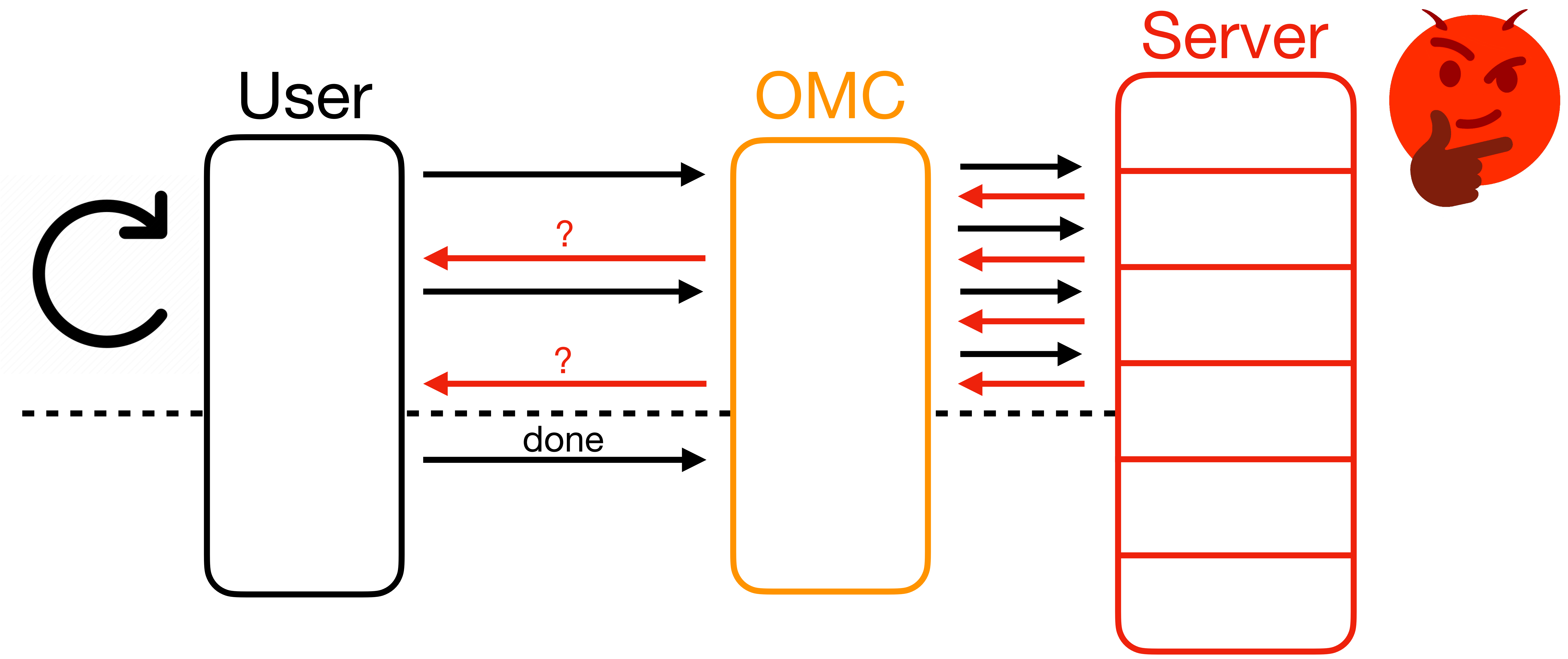
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



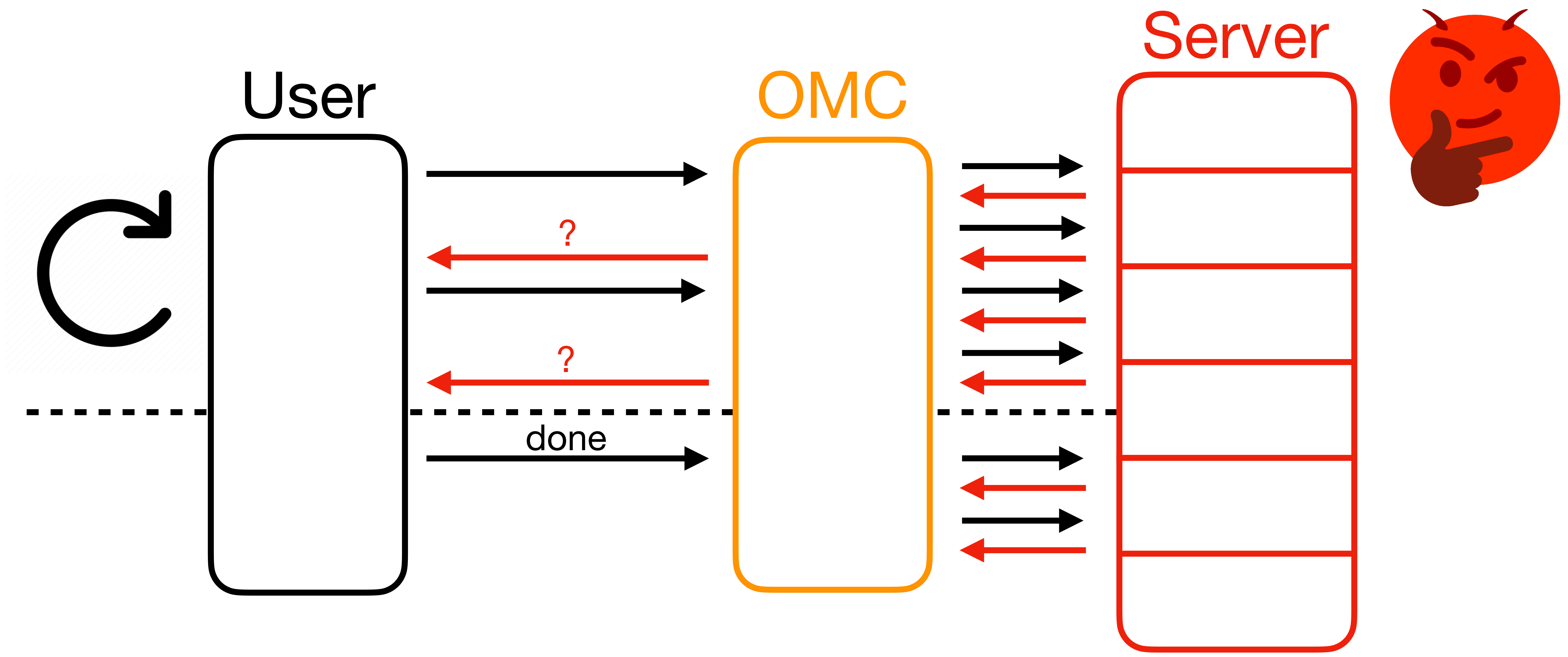
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



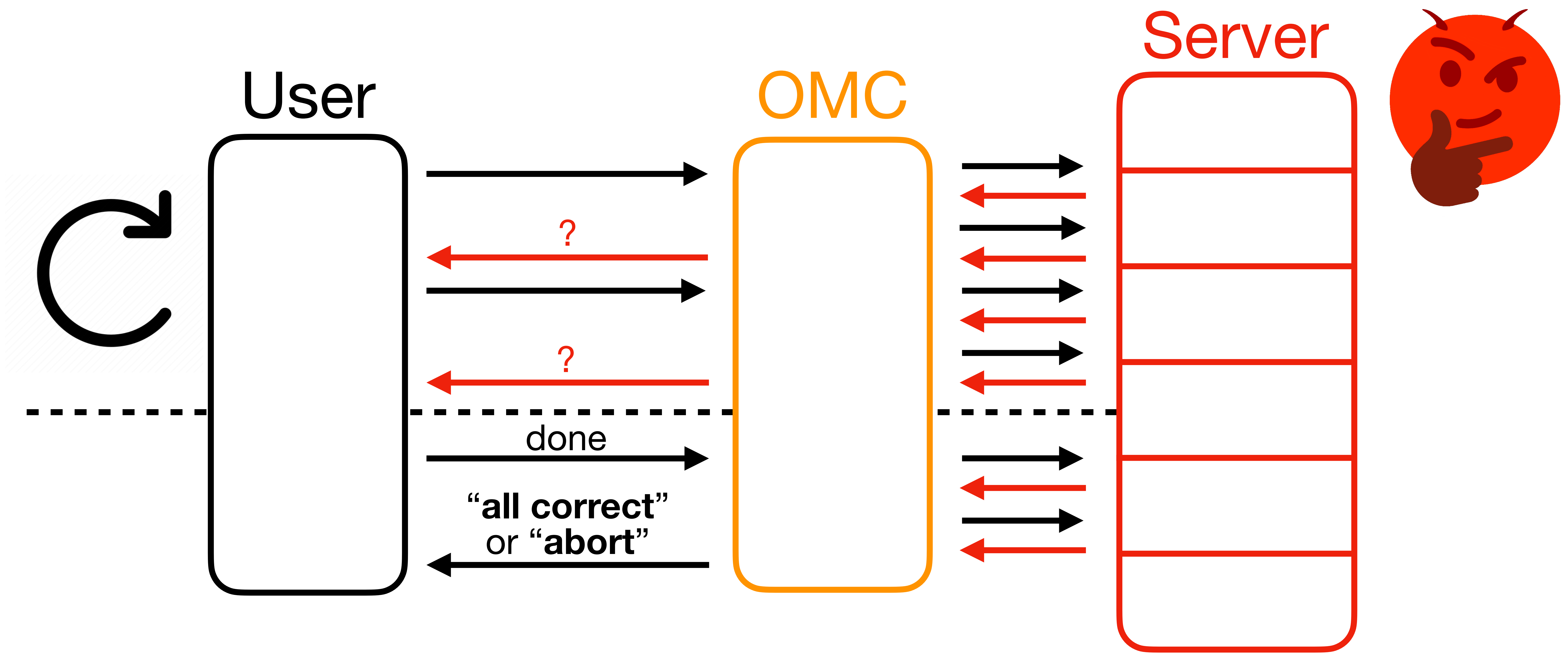
Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. ’94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think “batching” a regular memory checker.)



Offline Memory Checking

- An **Offline Memory Checker** (OMC) is a memory checker with a weaker correctness condition: [Blum et al. '94]
- Just needs to abort **by the end** – intermediate responses from the OMC may be incorrect! (Think "batching" a regular memory checker.)



Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**

[Blum et al. ’94]
[Dwork et al. ’09]

Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**
- Con of offline memory checking: **insufficient**! **Insecure for OptORAMa**.

Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**
- Con of offline memory checking: **insufficient**! **Insecure for OptORAMa**.
    - Replay attack (with MACs and offline memory checking) **still applies.**

Offline Memory Checking

- Benefit of offline memory checking: **constructions with** (amortized) $O(1)$ **overhead!**
- Con of offline memory checking: **insufficient**! **Insecure for OptORAMa**.
    - Replay attack (with MACs and offline memory checking) **still applies.**
- So when is offline checking safe?

Our Construction

Our Construction

- How do we get around this?

Our Construction

- How do we get around this?

- We combine time-stamping and offline checking **within algorithms**!

Balls-in-Bins Hashing

Balls-in-Bins Hashing

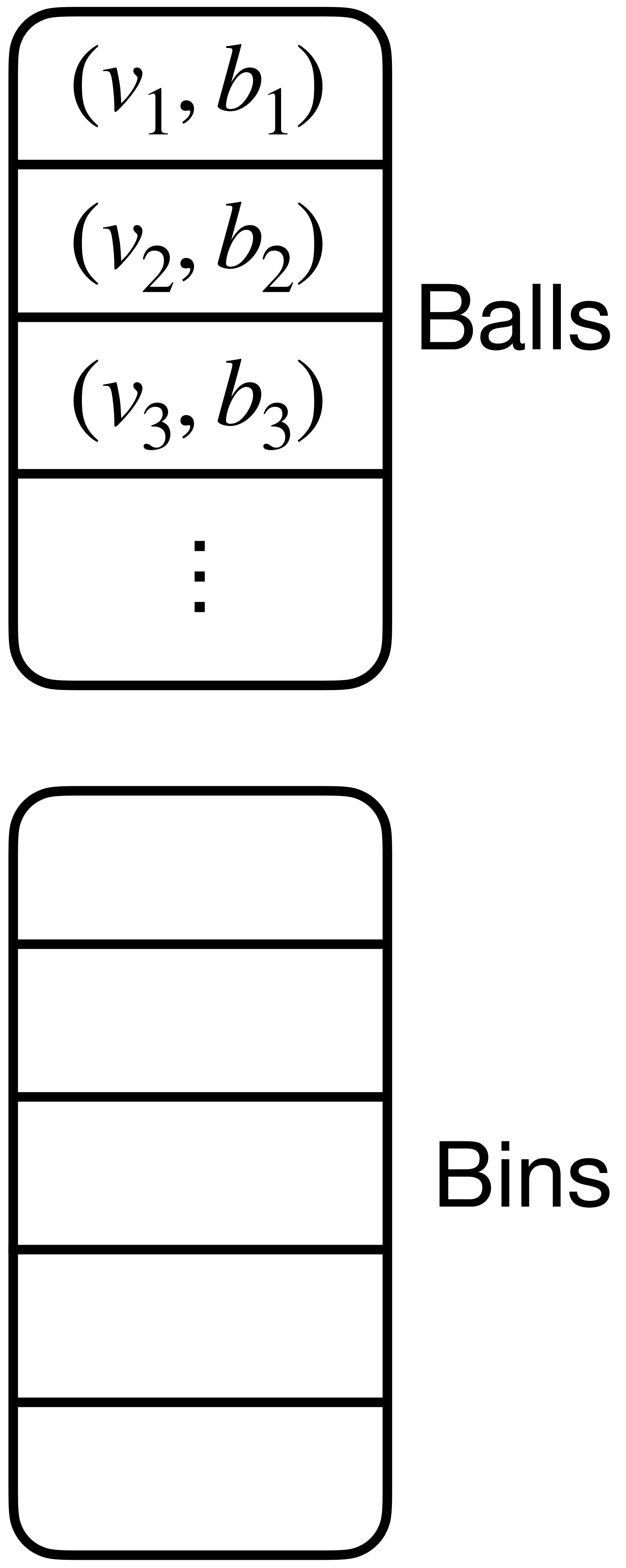
- **Example**: Hashing balls (values v_i) into bins (b_i).

Balls-in-Bins Hashing

- **Example**: Hashing balls (values v_i) into bins (b_i).
  - Used in building **OptORAMa** oblivious hash tables.

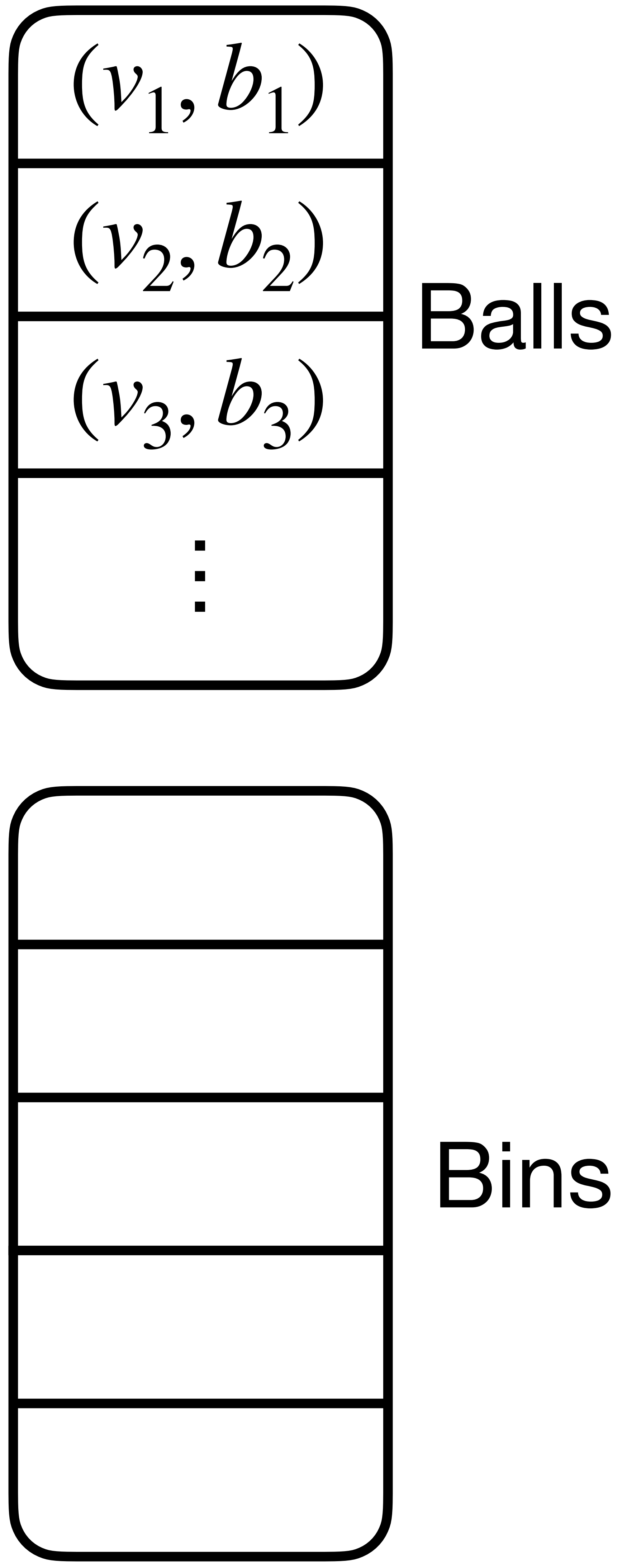
Balls-in-Bins Hashing

- **Example**: Hashing balls (values v_i) into bins (b_i).
    - Used in building **OptORAMa** oblivious hash tables.



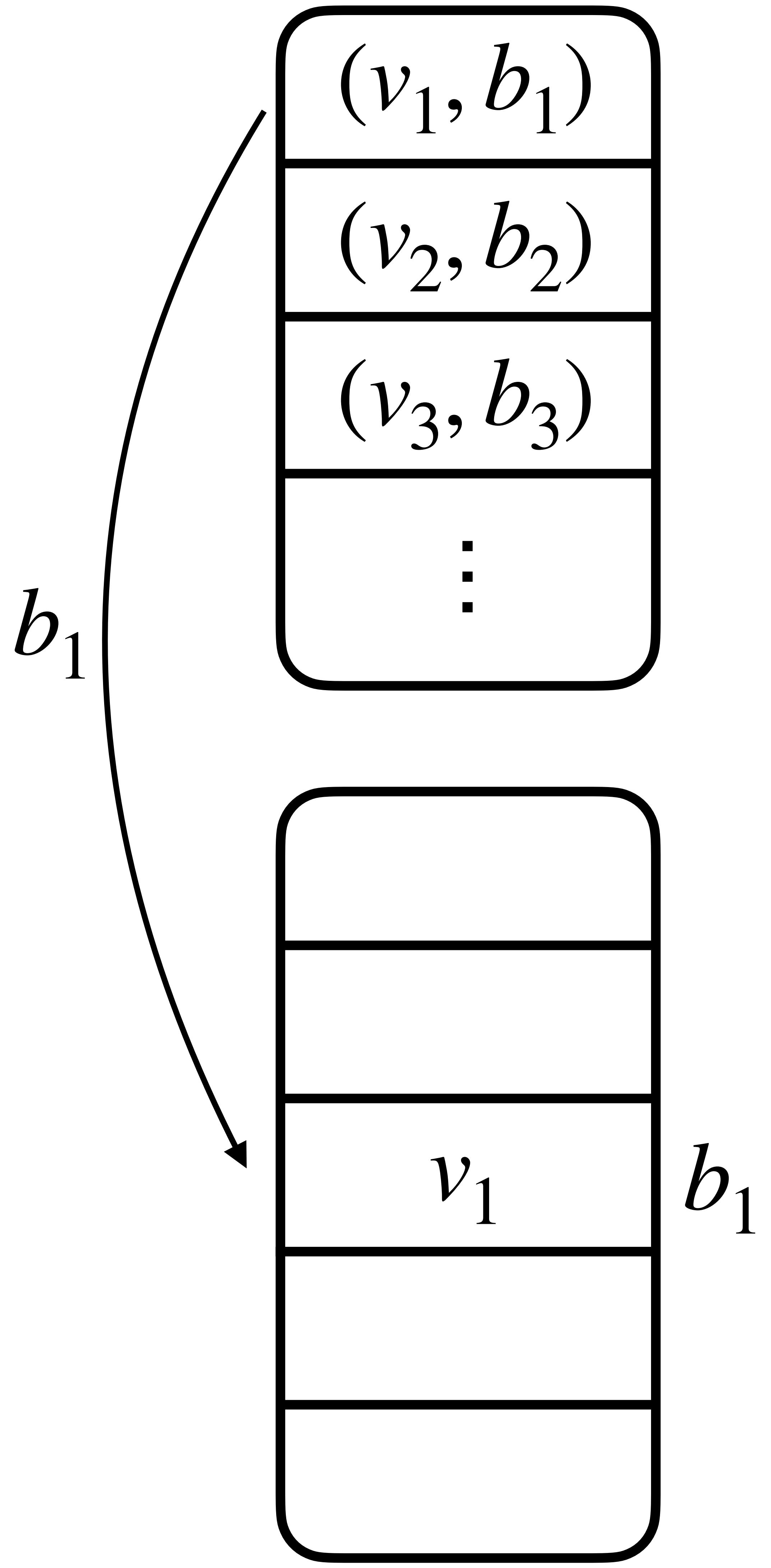
Balls-in-Bins Hashing

- **Example**: Hashing balls (values v_i) into bins (b_i).
  - Used in building **OptORAMa** oblivious hash tables.
- If b_i is safe to leak, access pattern is determined by $\{(v_i, b_i)\}$ array. Only b_i leaked.



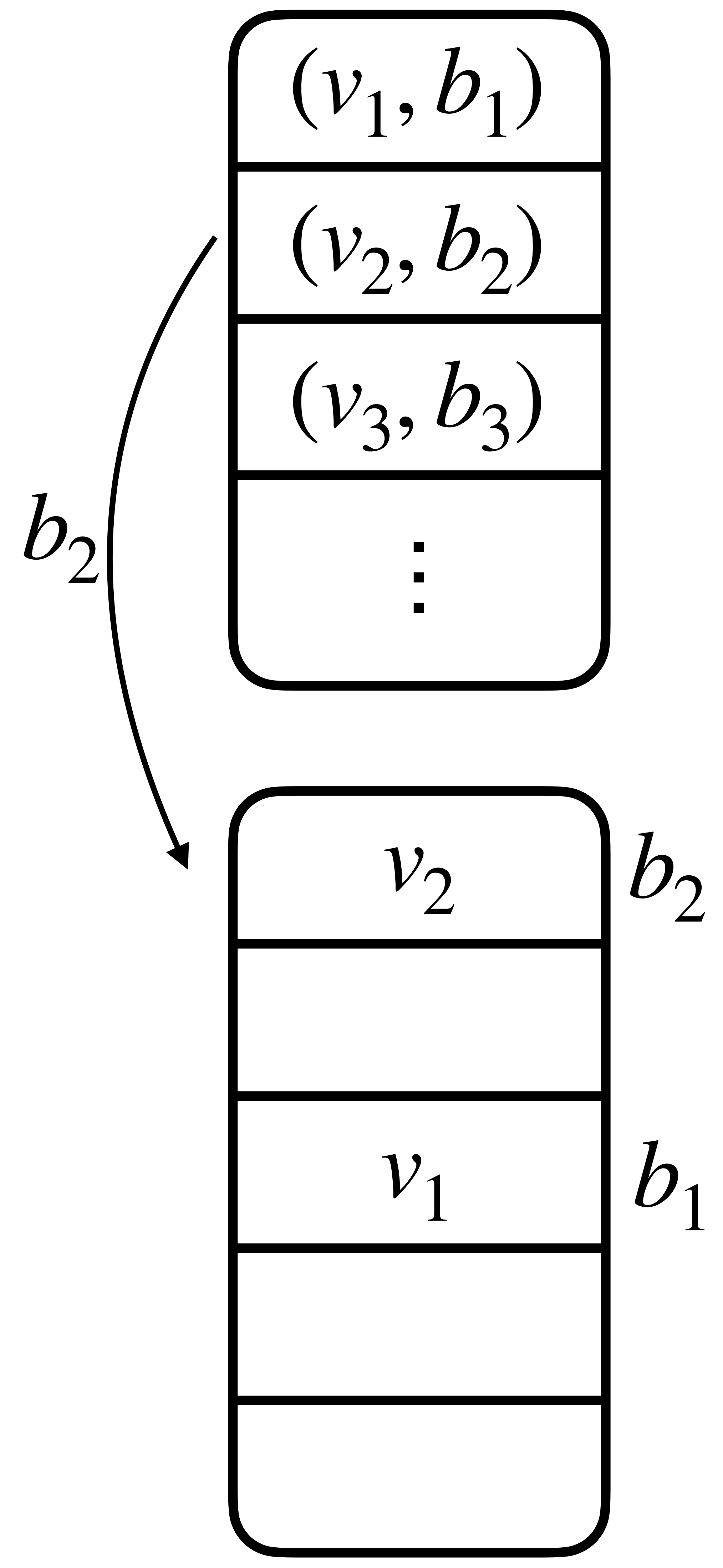
Balls-in-Bins Hashing

- **Example**: Hashing balls (values v_i) into bins (b_i).
  - Used in building **OptORAMa** oblivious hash tables.
- If b_i is safe to leak, access pattern is determined by $\{(v_i, b_i)\}$ array. Only b_i leaked.



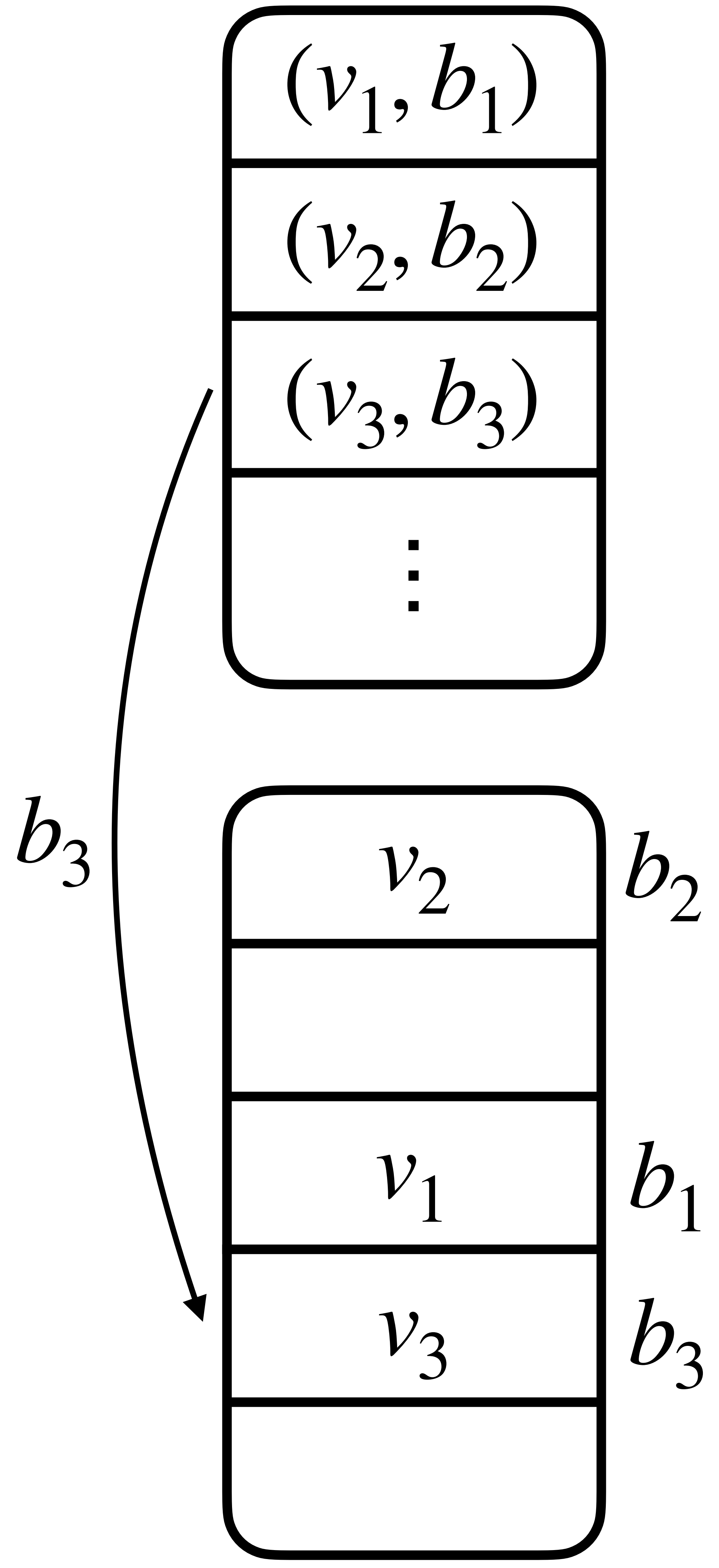
Balls-in-Bins Hashing

- **Example**: Hashing balls (values v_i) into bins (b_i).
  - Used in building **OptORAMa** oblivious hash tables.
- If b_i is safe to leak, access pattern is determined by $\{(v_i, b_i)\}$ array. Only b_i leaked.



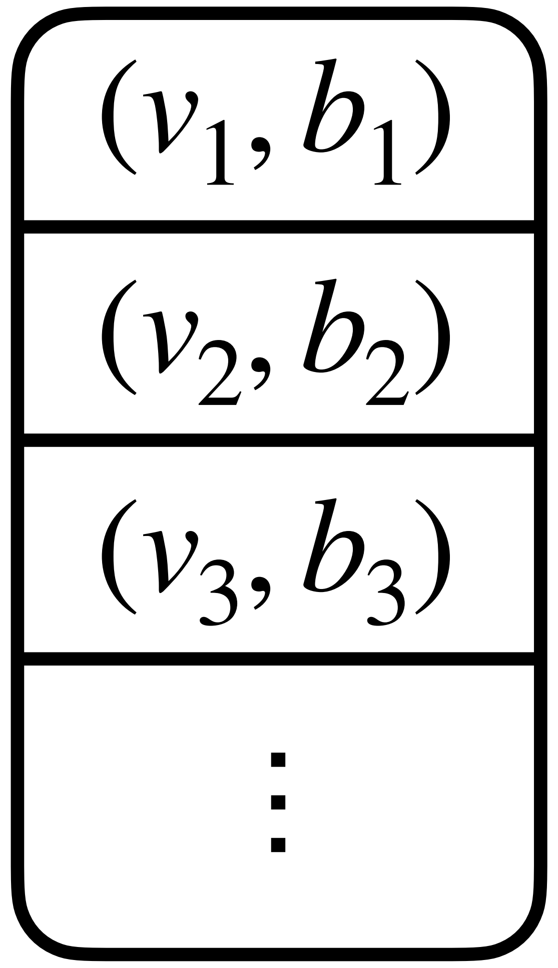
Balls-in-Bins Hashing

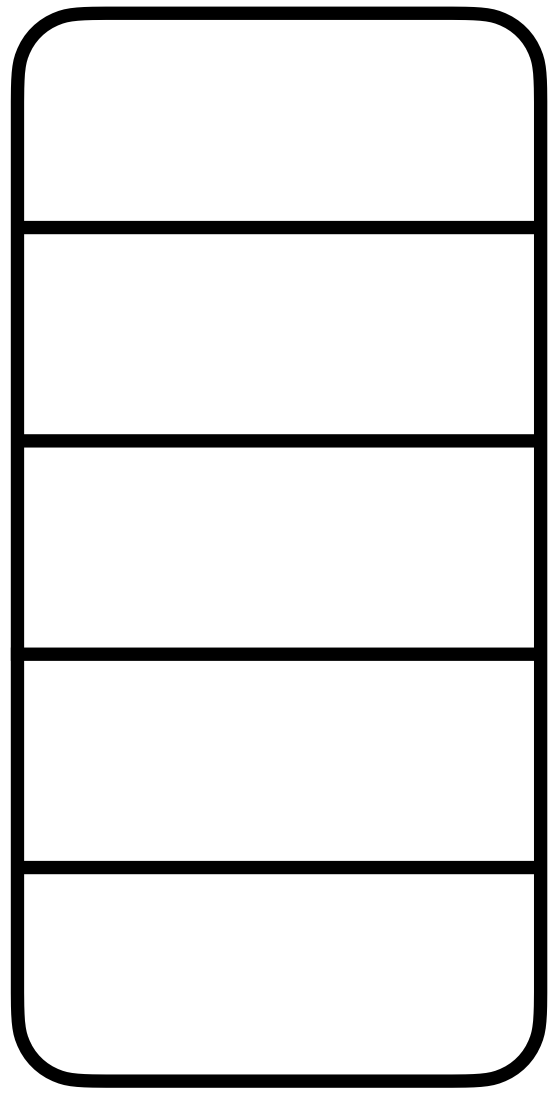
- **Example**: Hashing balls (values v_i) into bins (b_i).
    - Used in building **OptORAMa** oblivious hash tables.
- If b_i is safe to leak, access pattern is determined by $\{(v_i, b_i)\}$ array. Only b_i leaked.



Balls-in-Bins Hashing

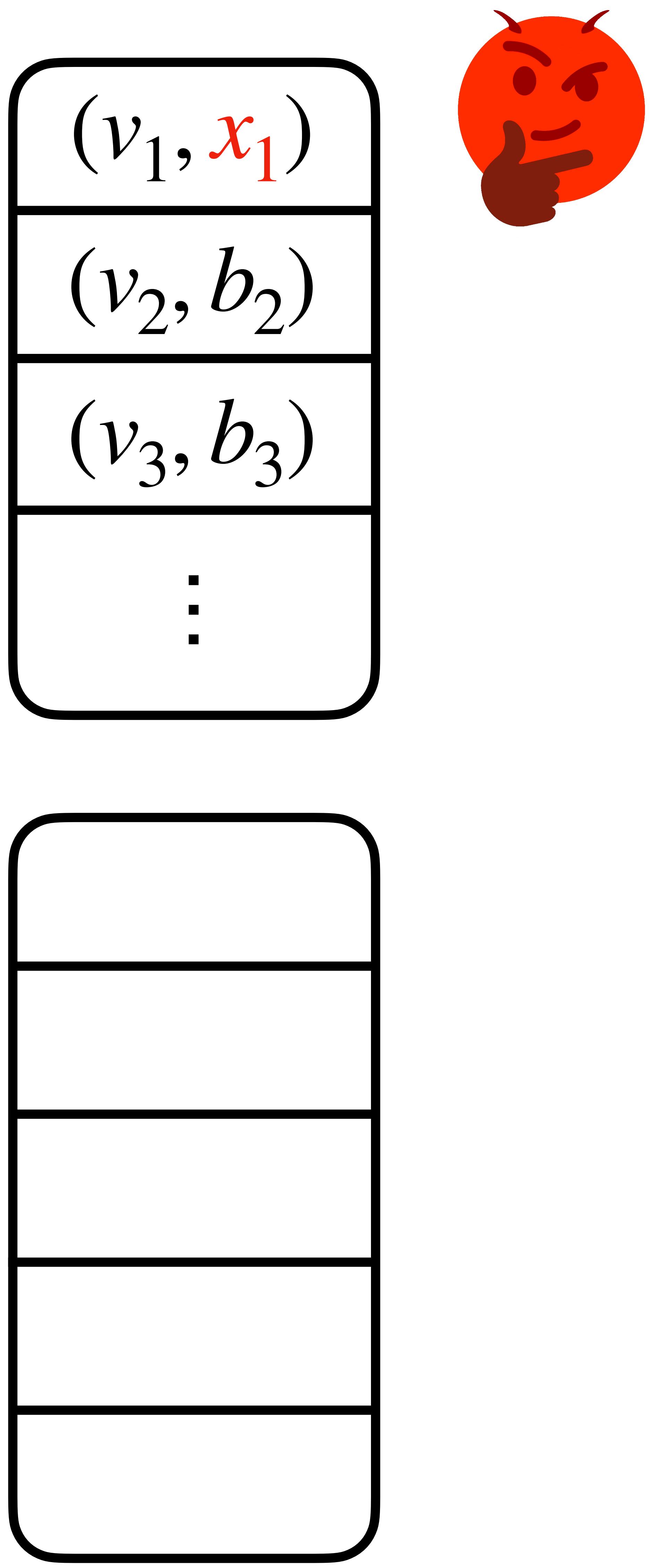
- **Example**: Hashing balls (values v_i) into bins (b_i).
  - Used in building **OptORAMa** oblivious hash tables.
- If b_i is safe to leak, access pattern is determined by $\{(v_i, b_i)\}$ array. Only b_i leaked.
- If $\{(v_i, b_i)\}$ array is tampered to include ciphertext of **private** x_i**,** then access pattern leaks x_i! Not offline-safe!





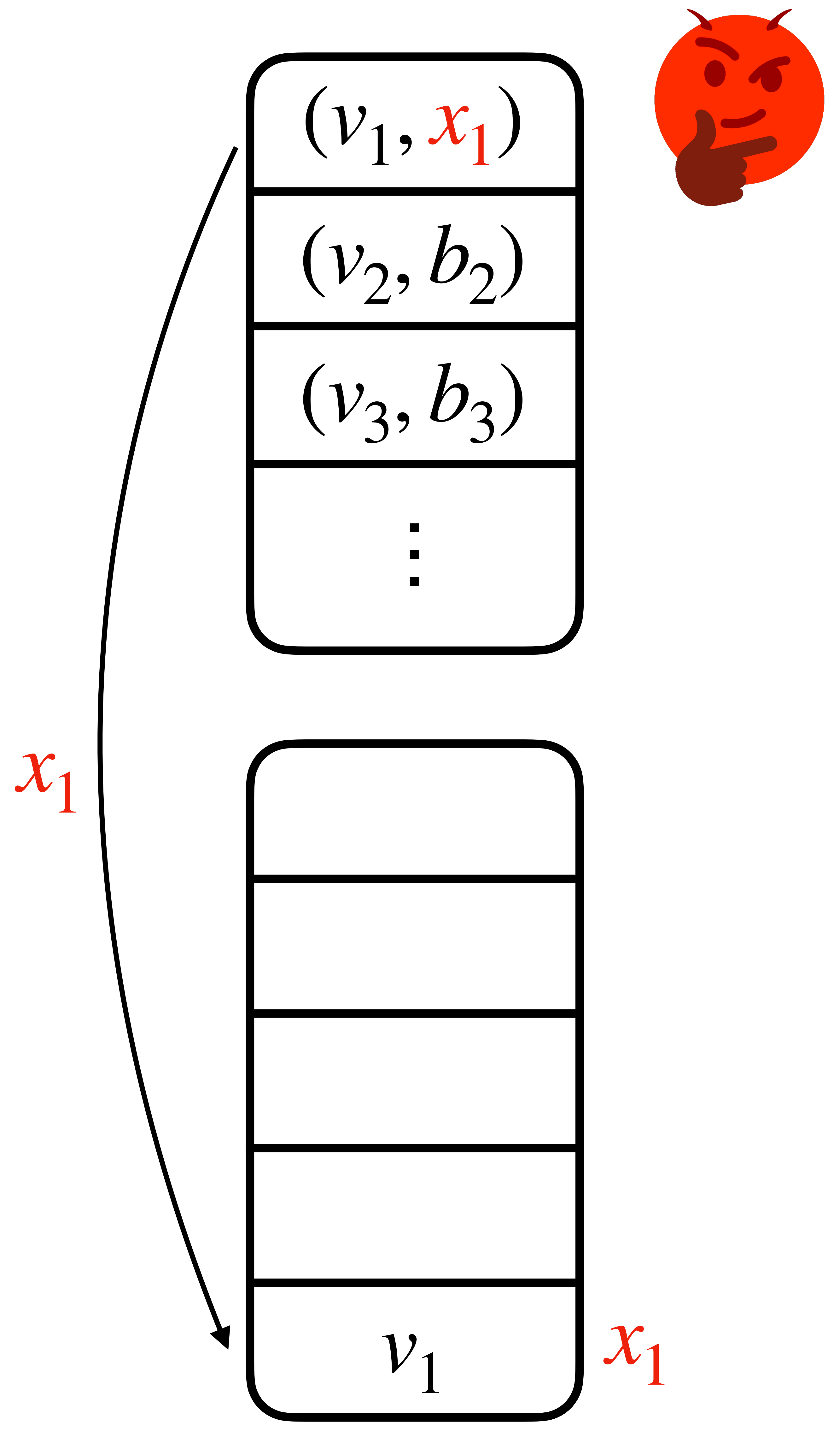
Balls-in-Bins Hashing

- **Example**: Hashing balls (values v_i) into bins (b_i).
  - Used in building **OptORAMa** oblivious hash tables.
- If b_i is safe to leak, access pattern is determined by $\{(v_i, b_i)\}$ array. Only b_i leaked.
- If $\{(v_i, b_i)\}$ array is tampered to include ciphertext of **private** x_i**,** then access pattern leaks x_i! Not offline-safe!



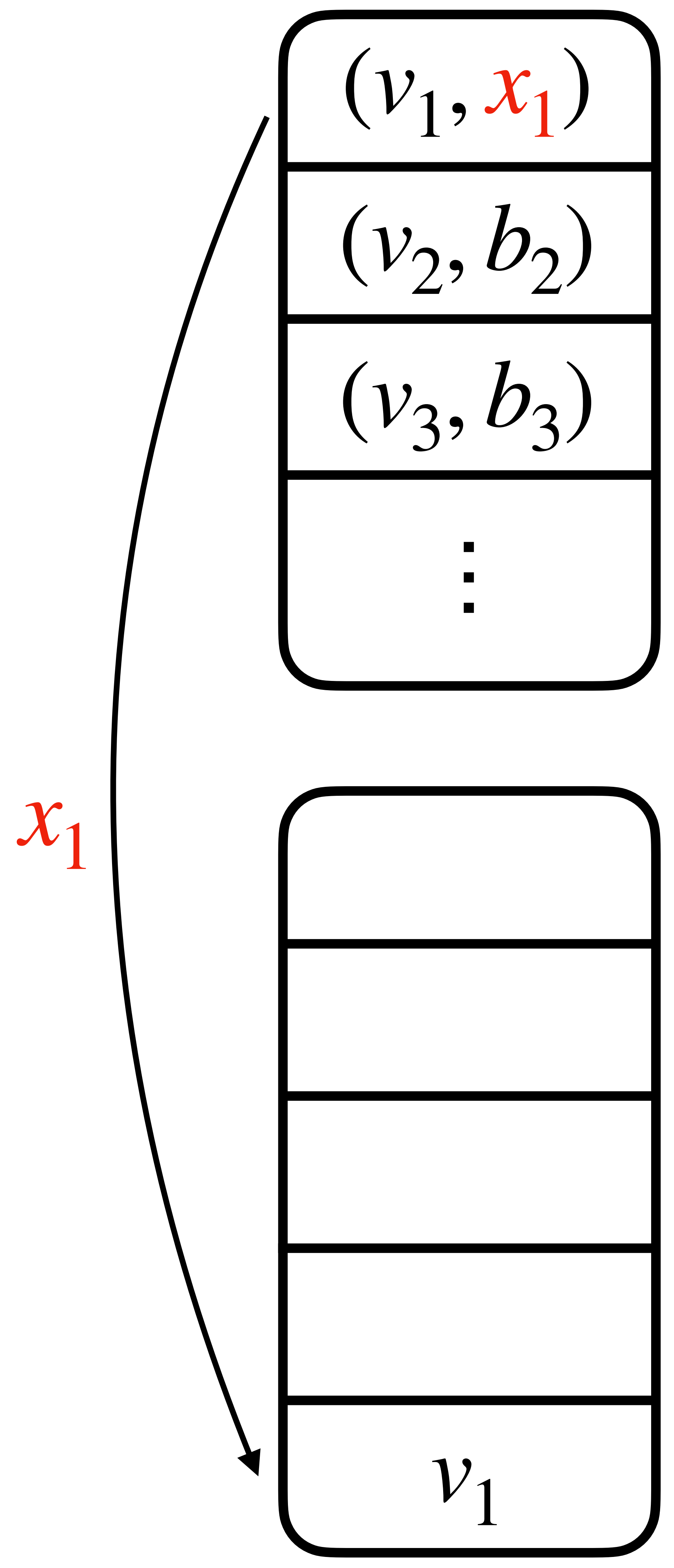
Balls-in-Bins Hashing

- **Example**: Hashing balls (values v_i) into bins (b_i).
  - Used in building **OptORAMa** oblivious hash tables.
- If b_i is safe to leak, access pattern is determined by $\{(v_i, b_i)\}$ array. Only b_i leaked.
- If $\{(v_i, b_i)\}$ array is tampered to include ciphertext of **private** x_i**,** then access pattern leaks x_i! Not offline-safe!

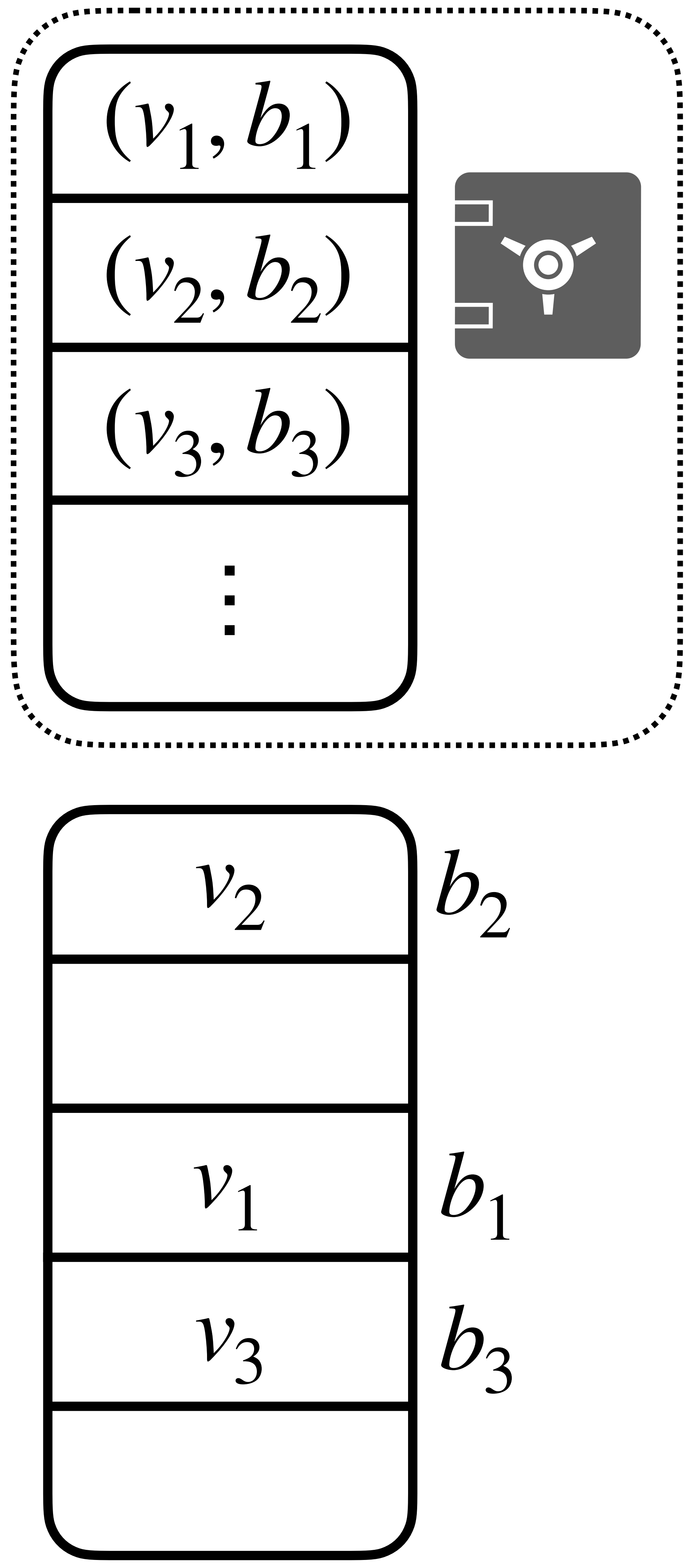


Balls-in-Bins Hashing

- **Example**: Hashing balls (values v_i) into bins (b_i).
  - Used in building **OptORAMa** oblivious hash tables.
- If b_i is safe to leak, access pattern is determined by $\{(v_i, b_i)\}$ array. Only b_i leaked.
- If $\{(v_i, b_i)\}$ array is tampered to include ciphertext of **private** x_i**,** then access pattern leaks x_i! Not offline-safe!
- But offline-safe if $\{(v_i, b_i)\}$ is not tampered with.

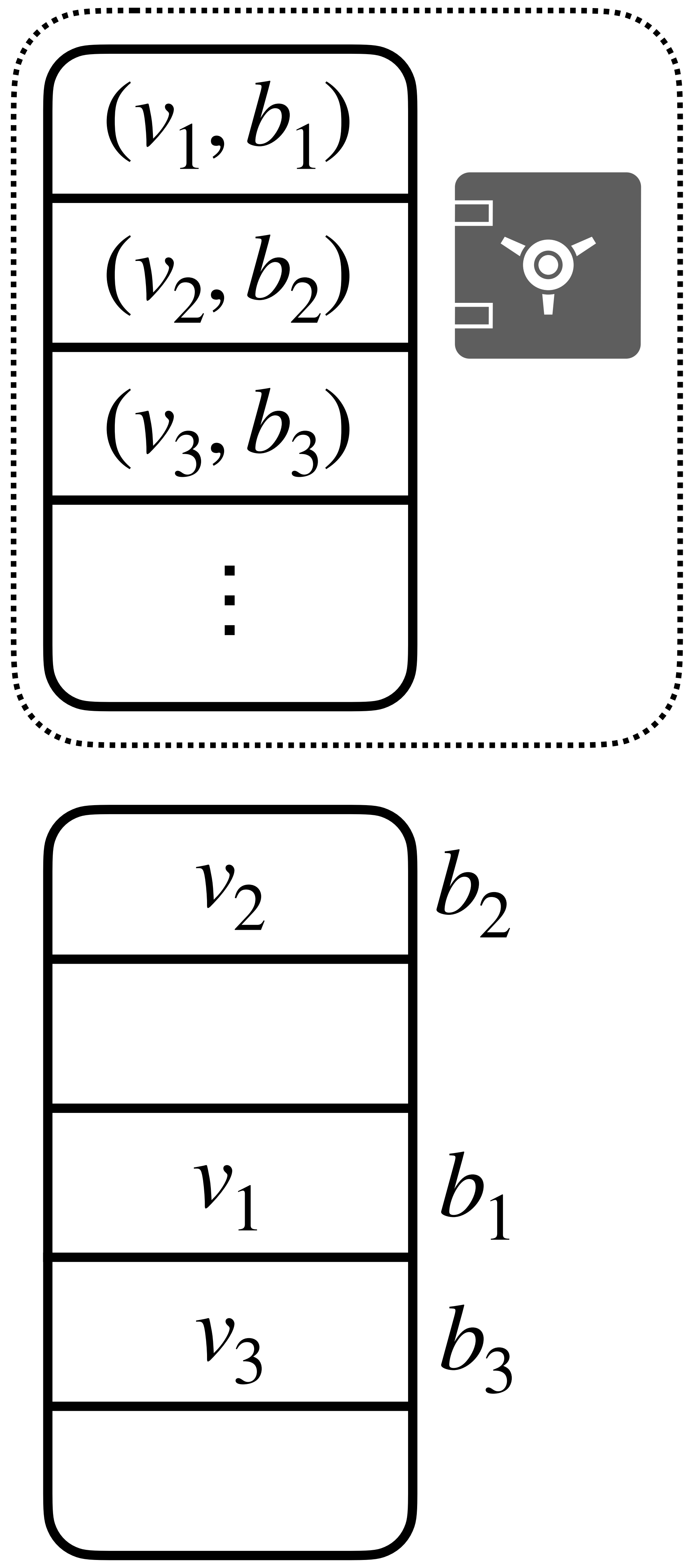


Combining Time-Stamping + Offline Checking



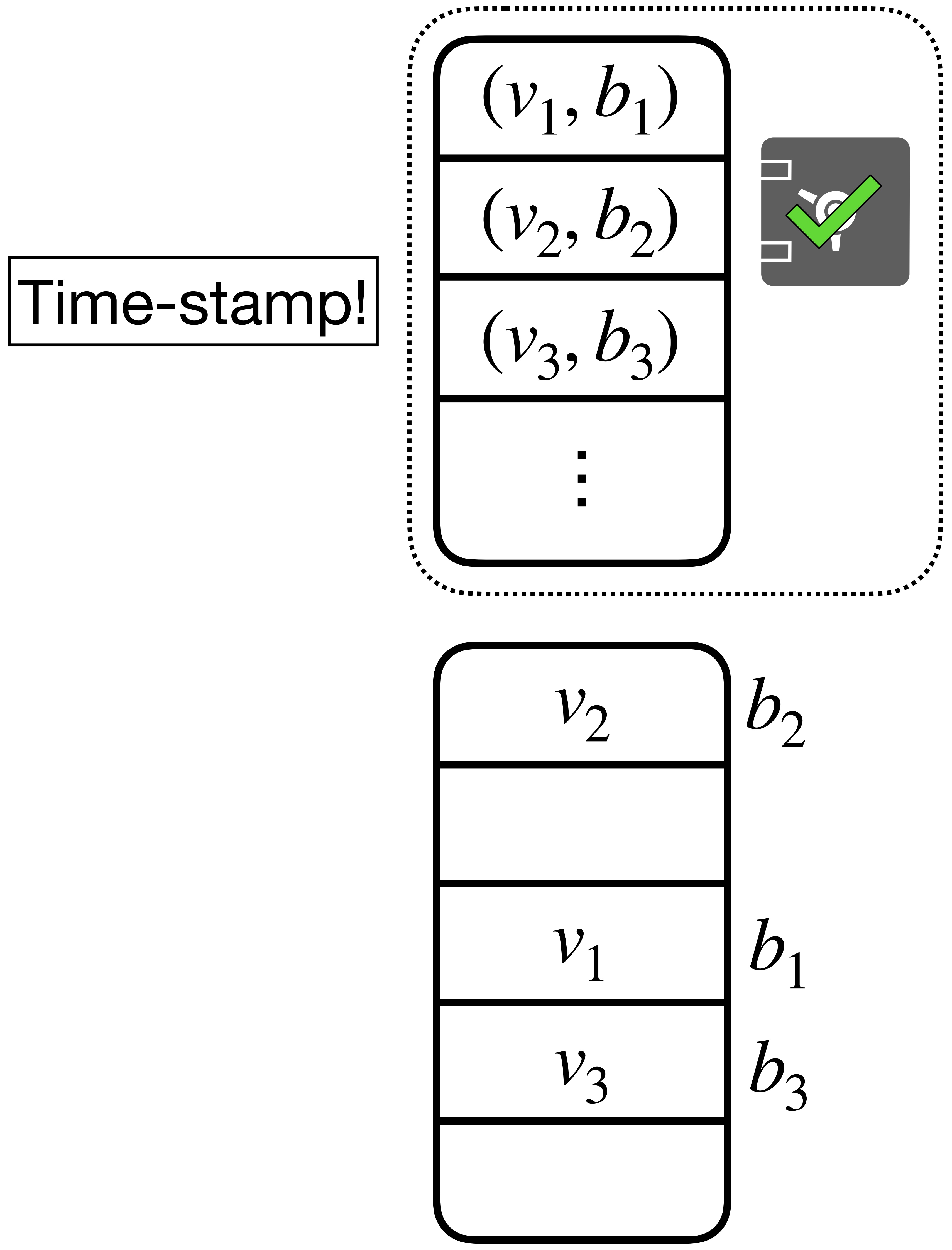
Combining Time-Stamping + Offline Checking

- **Key point**: If we can time-stamp $\{(v_i, b_i)\}$ array, the adversary can no longer tamper with it!



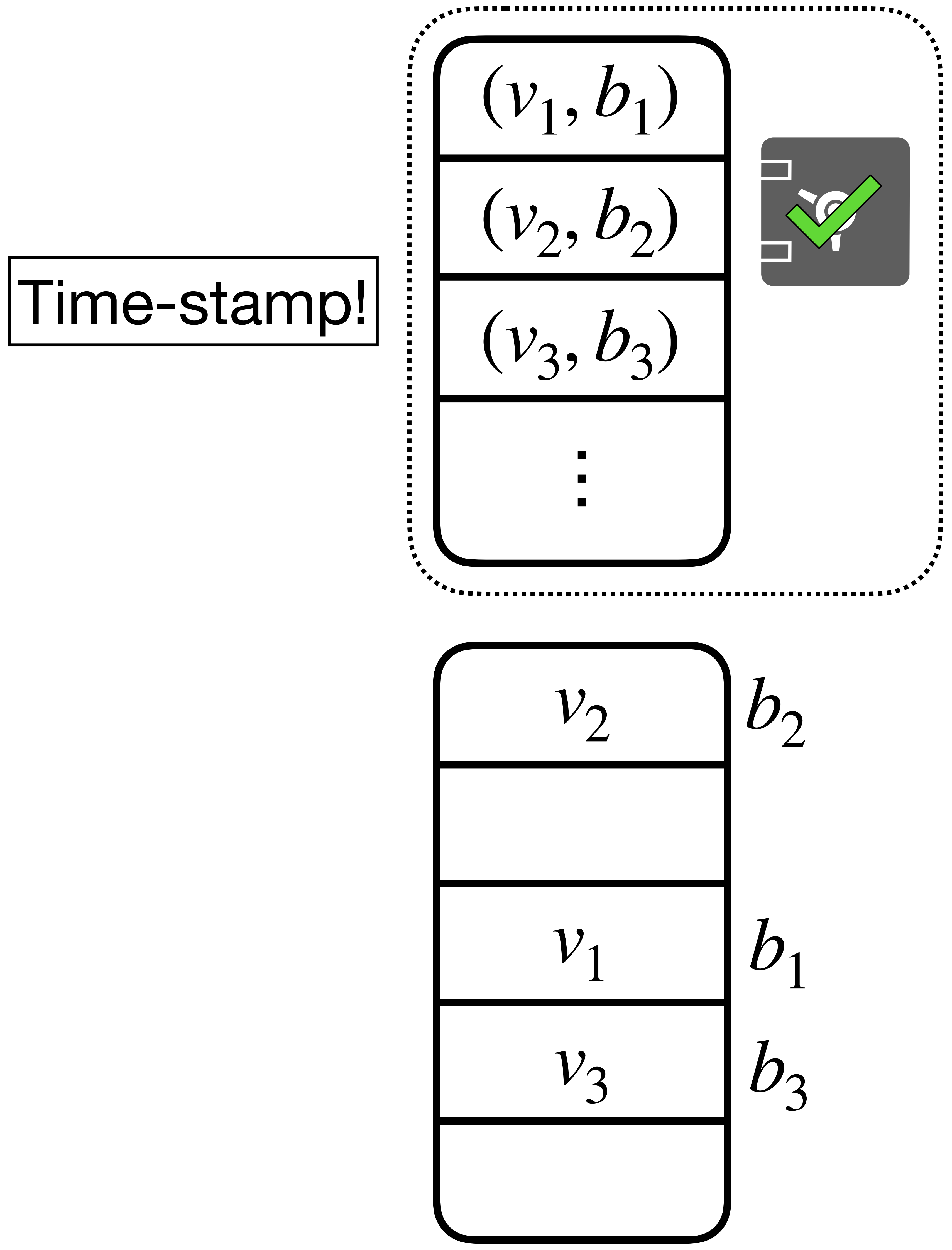
Combining Time-Stamping + Offline Checking

- **Key point**: If we can time-stamp $\{(v_i, b_i)\}$ array, the adversary can no longer tamper with it!



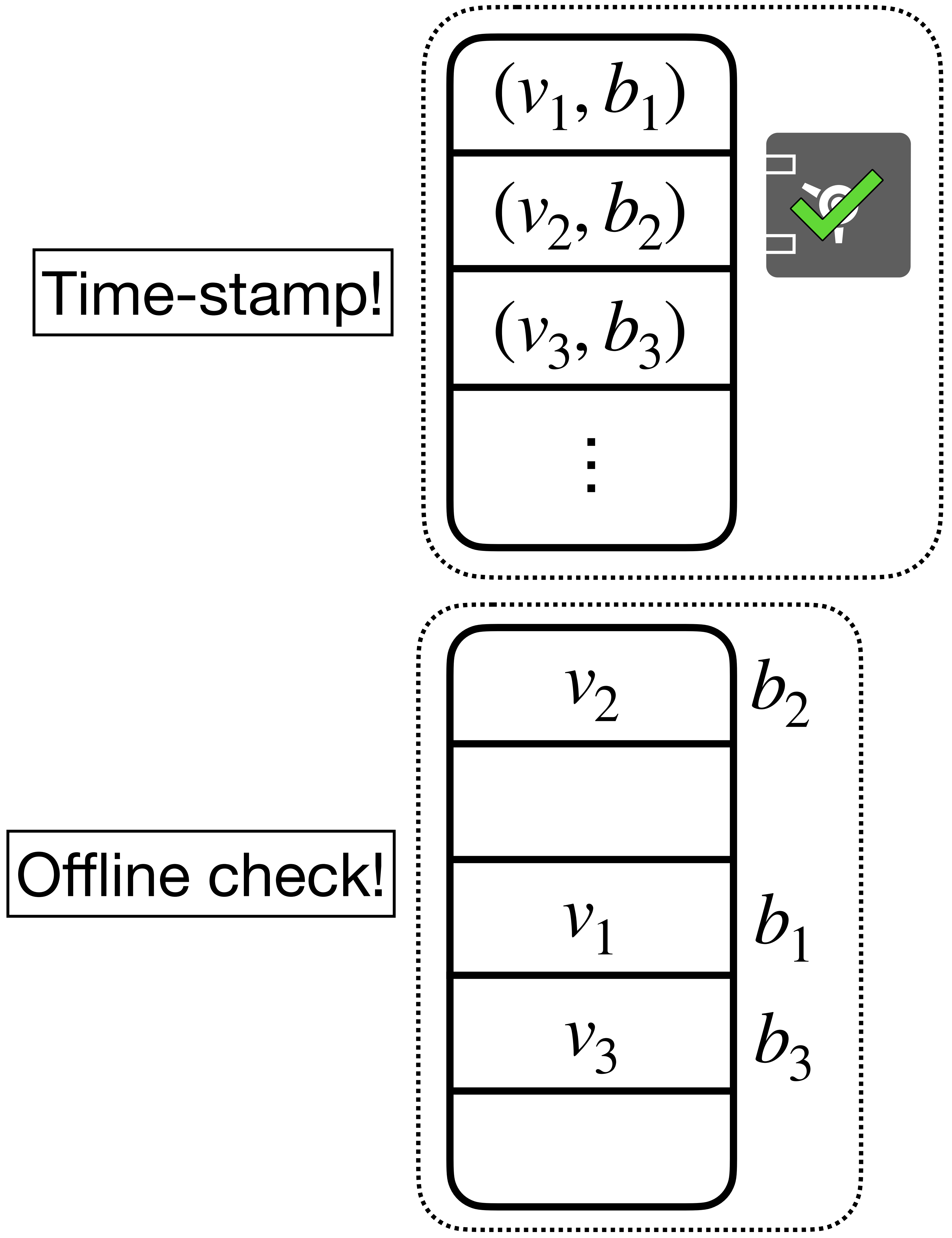
Combining Time-Stamping + Offline Checking

- **Key point**: If we can time-stamp $\{(v_i, b_i)\}$ array, the adversary can no longer tamper with it!
- Now, the hashing algorithm is offline-safe.



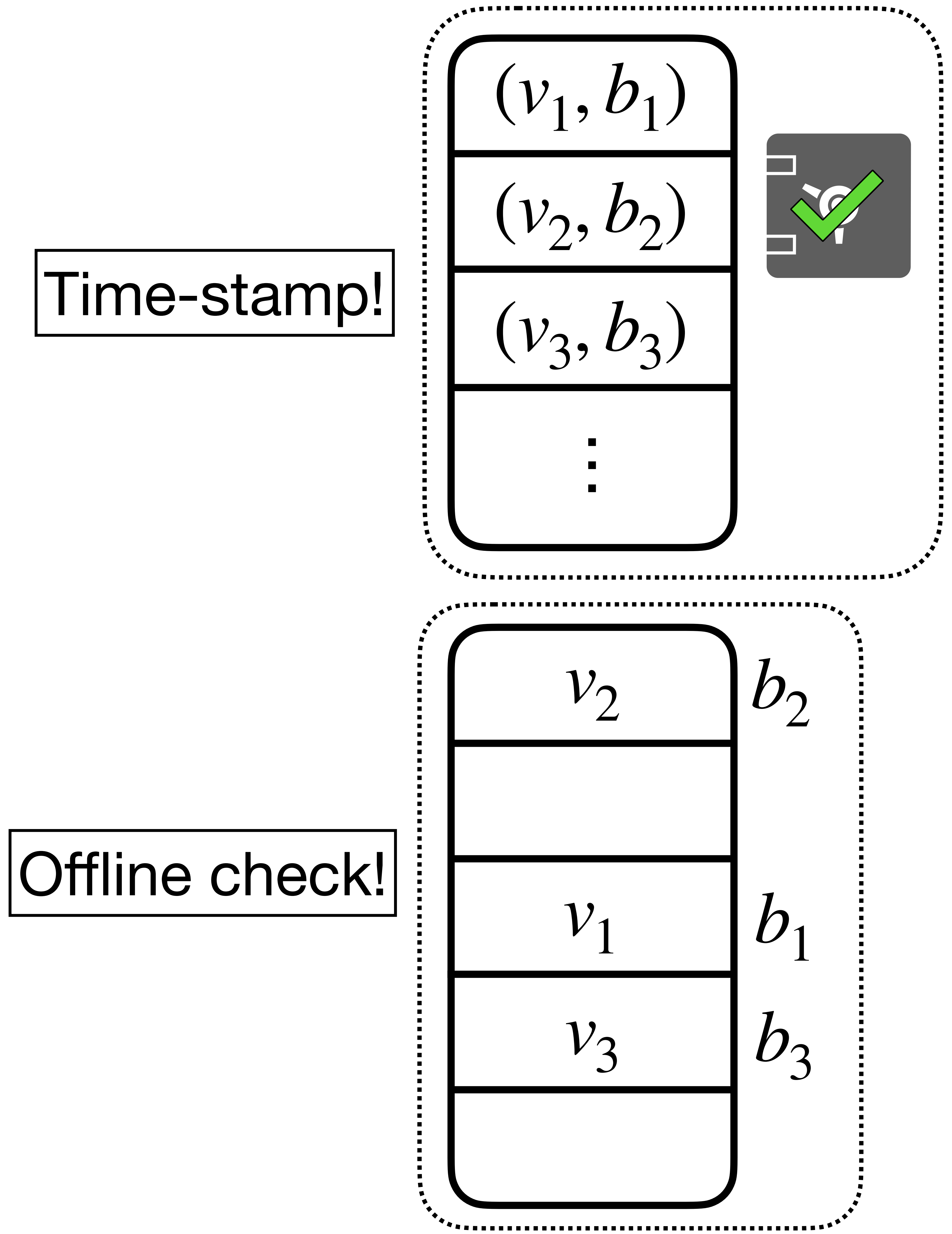
Combining Time-Stamping + Offline Checking

- **Key point**: If we can time-stamp $\{(v_i, b_i)\}$ array, the adversary can no longer tamper with it!
- Now, the hashing algorithm is offline-safe.



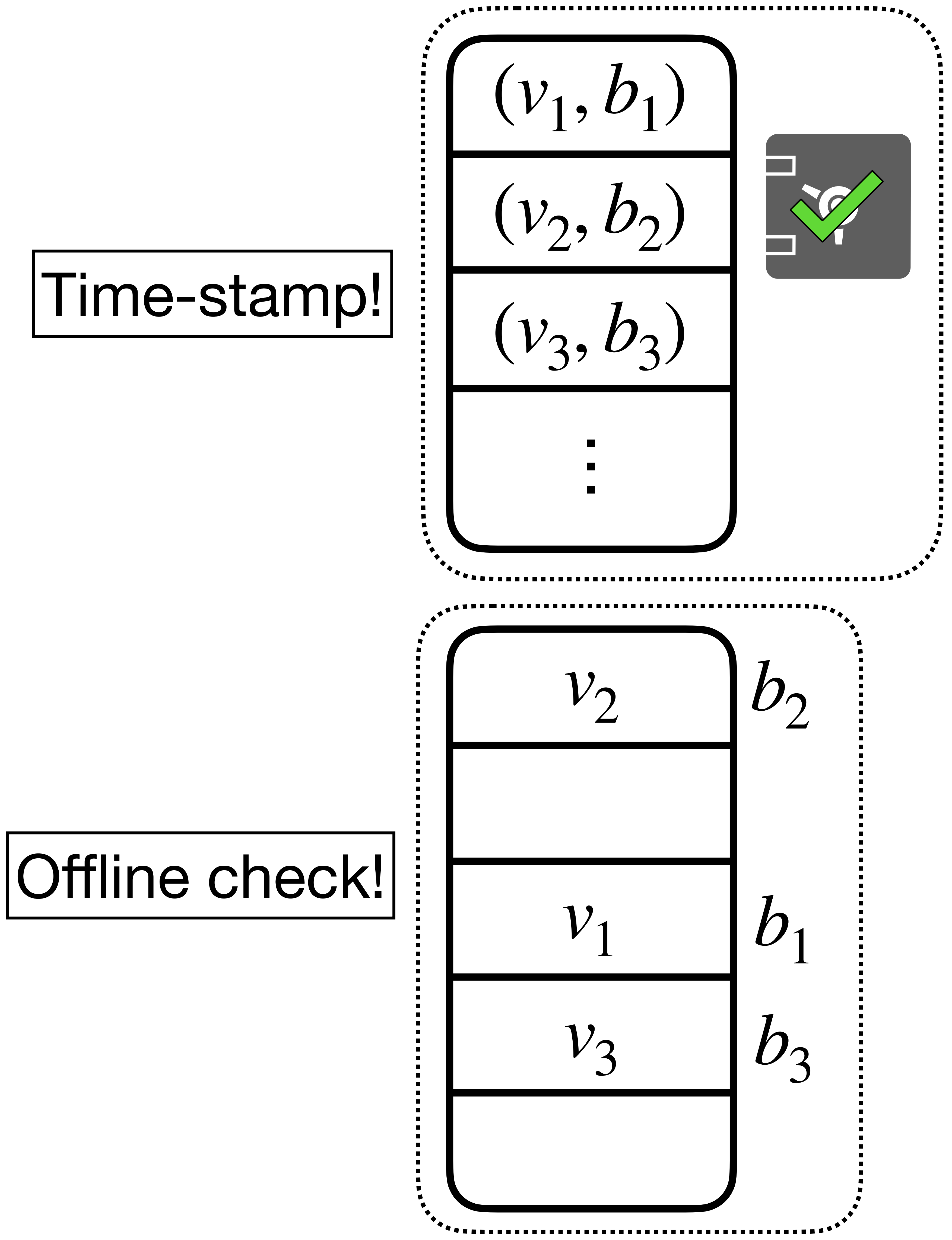
Combining Time-Stamping + Offline Checking

- **Key point**: If we can time-stamp $\{(v_i, b_i)\}$ array, the adversary can no longer tamper with it!
- Now, the hashing algorithm is offline-safe.
- **Summary**:



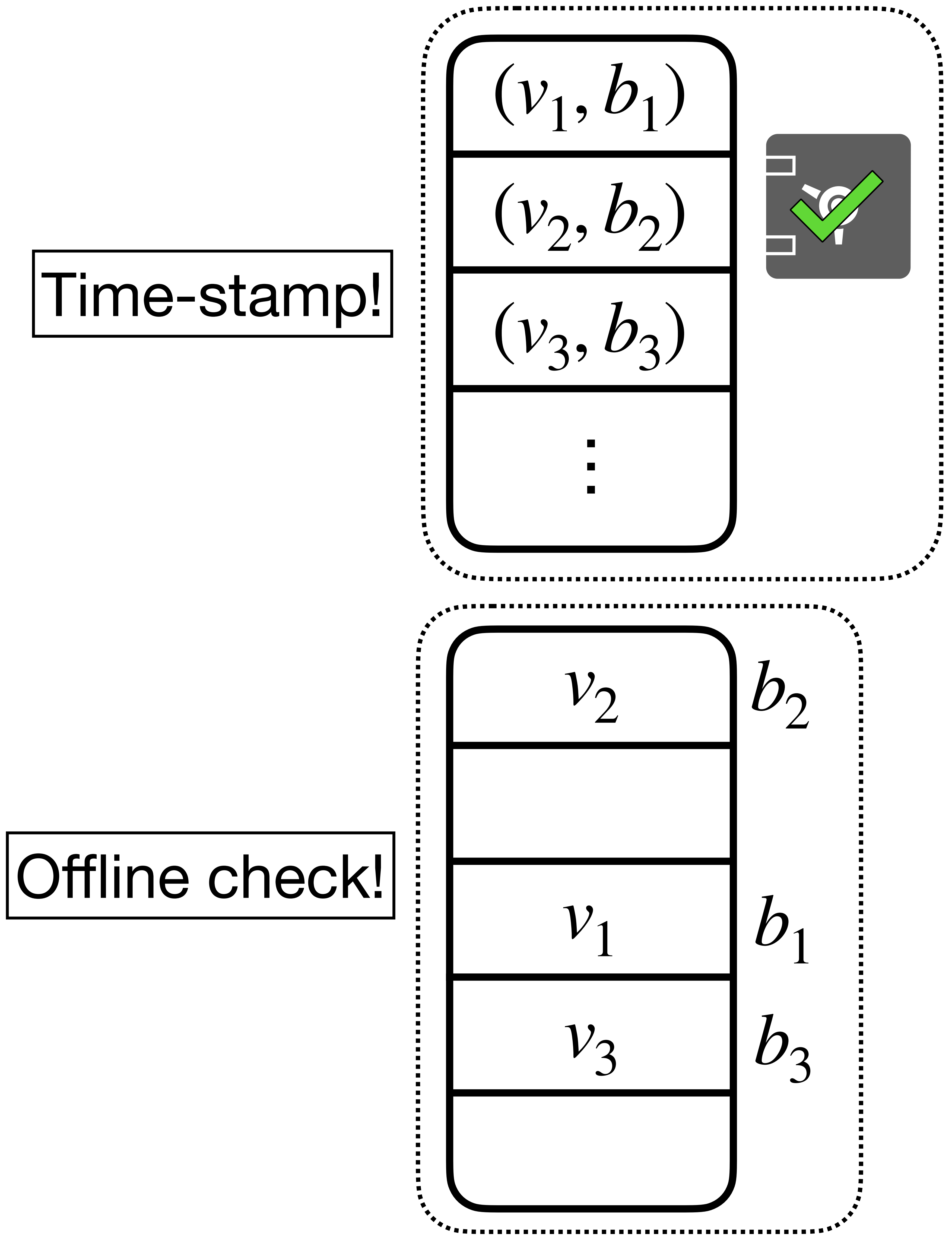
Combining Time-Stamping + Offline Checking

- **Key point**: If we can time-stamp $\{(v_i, b_i)\}$ array, the adversary can no longer tamper with it!
- Now, the hashing algorithm is offline-safe.
- **Summary**:
    - **Time-stamp** the part that needs to be tamper-proof (e.g., $\{(v_i, b_i)\}$ array).



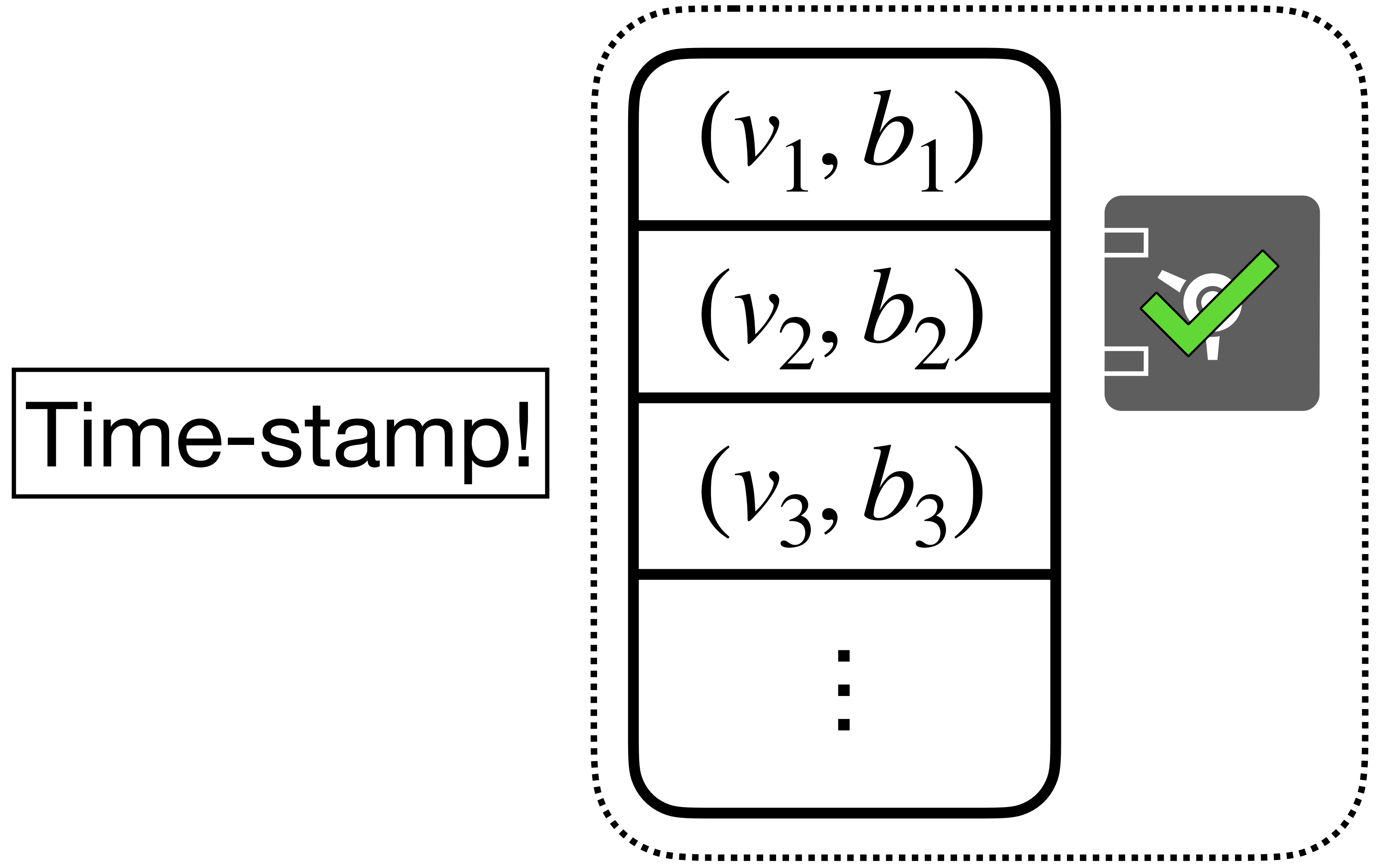
Combining Time-Stamping + Offline Checking

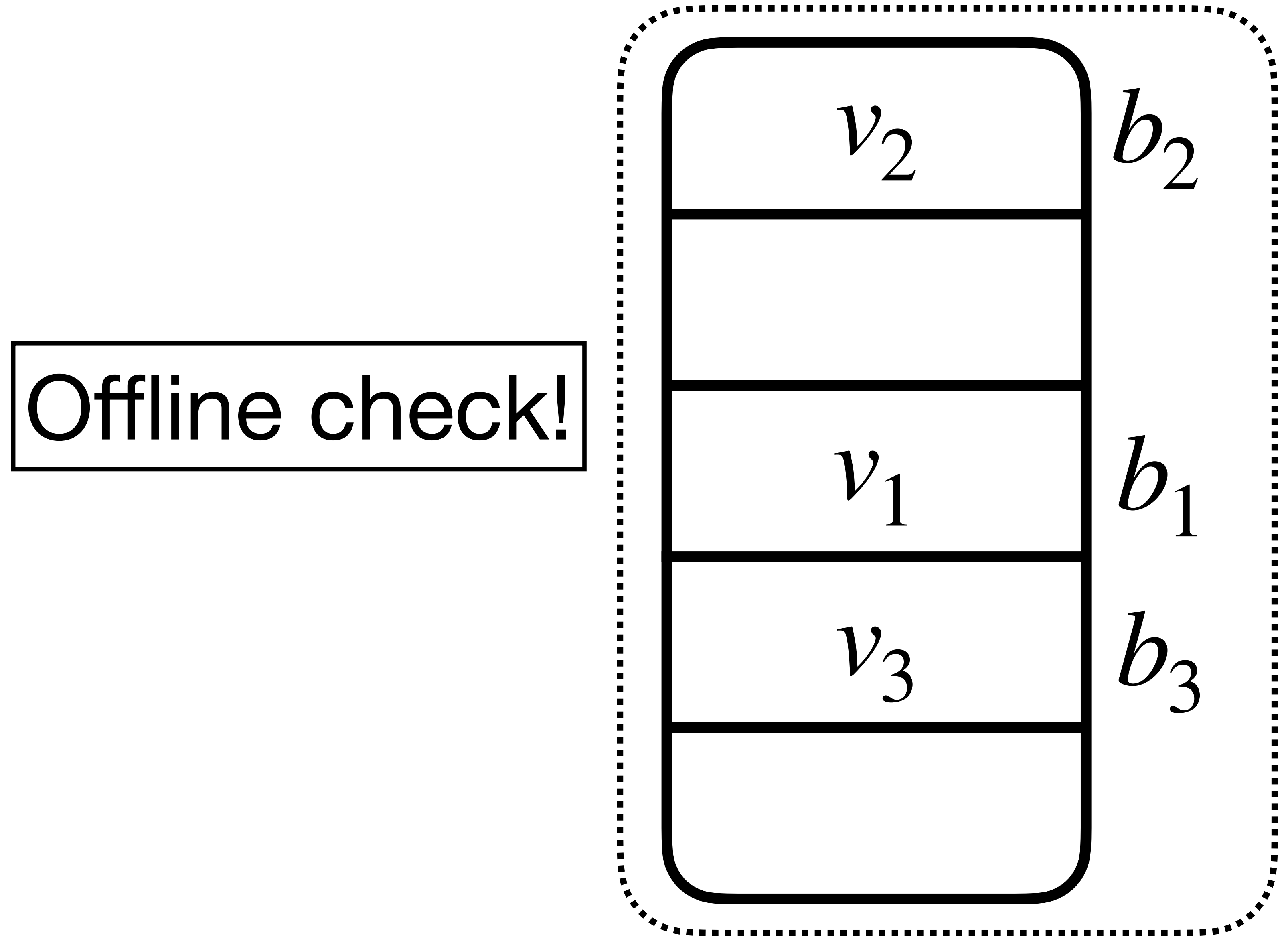
- **Key point**: If we can time-stamp $\{(v_i, b_i)\}$ array, the adversary can no longer tamper with it!
- Now, the hashing algorithm is offline-safe.
- **Summary**:
    - **Time-stamp** the part that needs to be tamper-proof (e.g., $\{(v_i, b_i)\}$ array).
    - **Offline check** the rest.



Combining Time-Stamping + Offline Checking

- **Key point**: If we can time-stamp $\{(v_i, b_i)\}$ array, the adversary can no longer tamper with it!
- Now, the hashing algorithm is offline-safe.
- **Summary**:
  - **Time-stamp** the part that needs to be tamper-proof (e.g., $\{(v_i, b_i)\}$ array).
  - **Offline check** the rest.
  - Converts **honest-but-curious** to **malicious** security!





When is Offline Checking Safe?

1. Access-Deterministic

When is Offline Checking Safe?

1. Access-Deterministic

   - **Definition**: A subroutine is **access-deterministic** if $\{\widehat{\mathsf{addr}}_i\}$ is deterministic and *perfectly independent* of the input **when interacting with an honest server**.

When is Offline Checking Safe?

1. Access-Deterministic

   - **Definition**: A subroutine is **access-deterministic** if $\{\widehat{\mathsf{addr}}_i\}$ is deterministic and *perfectly independent* of the input **when interacting with an honest server**.
   - In general, access-deterministic subroutines **may not be offline-safe** against **adversarial** servers. Nonetheless:

When is Offline Checking Safe?

1. Access-Deterministic

- **Definition**: A subroutine is **access-deterministic** if $\{\widehat{\mathrm{addr}}_i\}$ is deterministic and *perfectly independent* of the input **when interacting with an honest server**.

- In general, access-deterministic subroutines **may not be offline-safe** against **adversarial** servers. Nonetheless:

Theorem [**M**V '23]: If a subroutine is *access-deterministic*, then it can be made *maliciously secure* with the **same asymptotic overhead.**

When is Offline Checking Safe?

1. Access-Deterministic

- **Definition**: A subroutine is **access-deterministic** if $\{\widehat{\mathsf{addr}}_i\}$ is deterministic and *perfectly independent* of the input **when interacting with an honest server**.

- In general, access-deterministic subroutines **may not be offline-safe** against **adversarial** servers. Nonetheless:

Theorem [**M**V '23]: If a subroutine is *access-deterministic*, then it can be made *maliciously secure* with the **same asymptotic overhead.**

- **Idea:** Use *offline-checking* to pre-process a $\mathsf{PrevTime}$ data-structure for the algorithm, and use this to **time-stamp** the algorithm.

When is Offline Checking Safe?

1. Access-Deterministic

- **Definition**: A subroutine is **access-deterministic** if $\{\widehat{\mathsf{addr}}_i\}$ is deterministic and *perfectly independent* of the input **when interacting with an honest server**.
- In general, access-deterministic subroutines **may not be offline-safe** against **adversarial** servers. Nonetheless:

Theorem [**M**V '23]: If a subroutine is *access-deterministic*, then it can be made *maliciously secure* with the **same asymptotic overhead.**

- **Idea:** Use *offline-checking* to pre-process a $\mathsf{PrevTime}$ data-structure for the algorithm, and use this to **time-stamp** the algorithm.
- Can be viewed as a strengthening of Goldreich-Ostrovsky's time-stamping theorem!

Why Access-Deterministic Algorithms May Not Be Offline-Safe

- Consider the following implementation of an AKS sort.

  1. Use server space to compute and store a bipartite expander $G = (V, E)$.

  2. Iterate over edge set E, and make comparisons according to E.

- If the contents of E are **replaced with secret data**, the secret data will be leaked!