Memory Checking for Parallel RAMs

Surya Mathialagan
MIT

Quick Overview

Quick Overview

- We define a new notion of **memory checking** in the **parallel RAM** model.

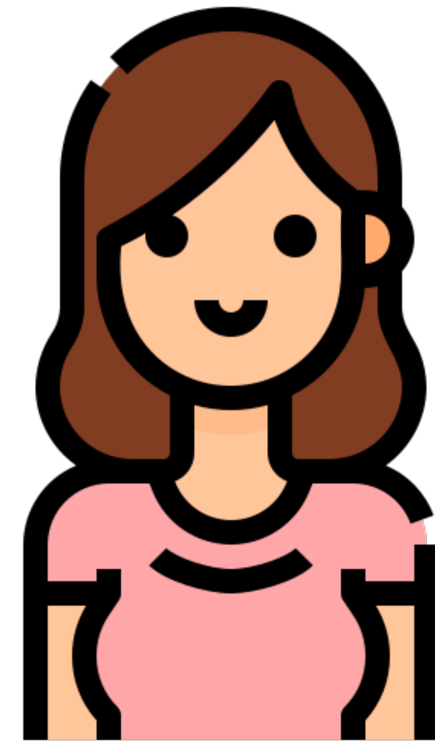
Quick Overview

- We define a new notion of **memory checking** in the **parallel RAM** model.
- We construct memory checkers for PRAMs matching the asymptotic efficiency of memory checkers for the RAM setting, while achieving optimal parallel depth.

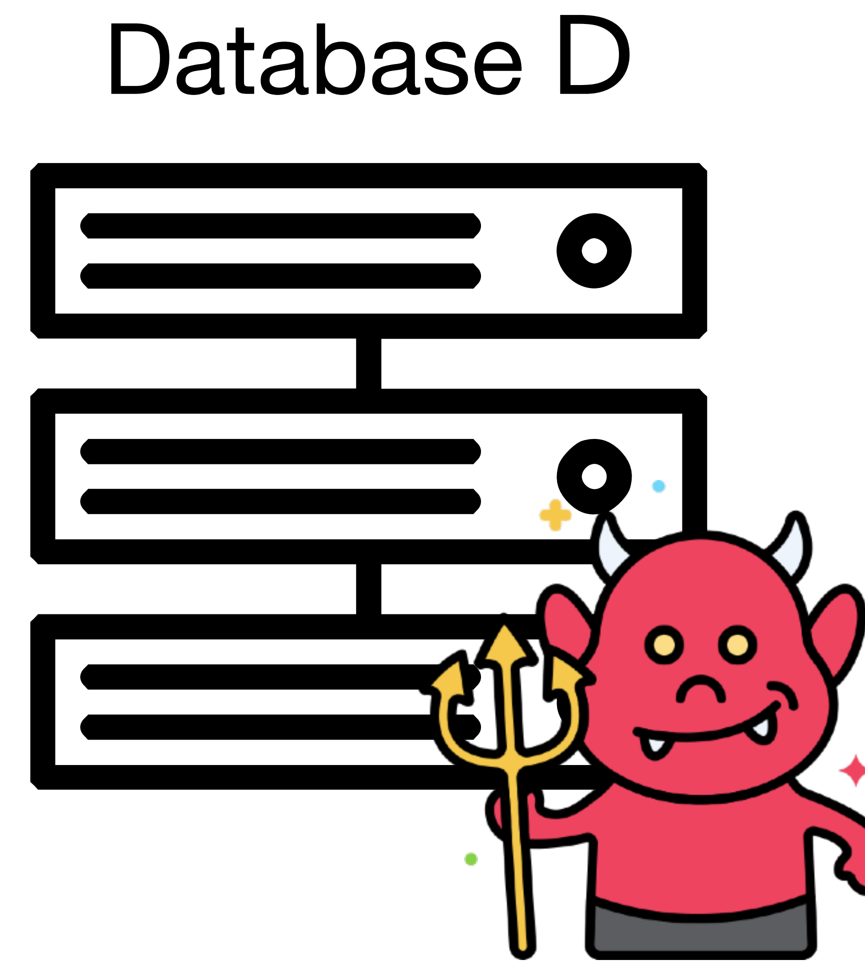
Quick Overview

- We define a new notion of **memory checking** in the **parallel RAM** model.
- We construct memory checkers for PRAMs matching the asymptotic efficiency of memory checkers for the RAM setting, while achieving optimal parallel depth.
- As an application, we construct maliciously secure Oblivious Parallel RAM with polylog overhead.

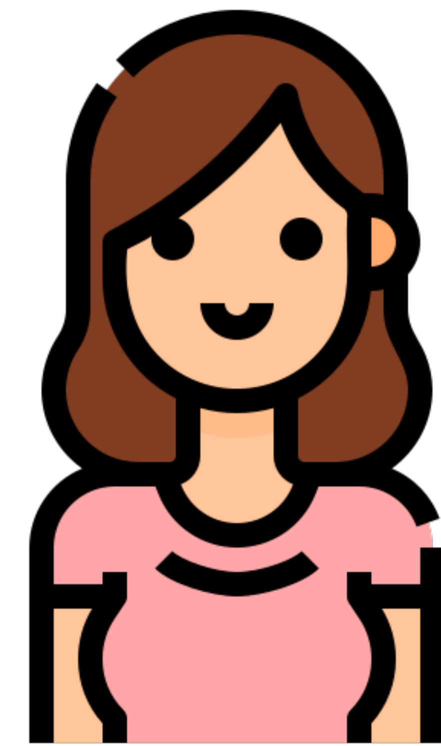
Integrity of Cloud Servers



Database D

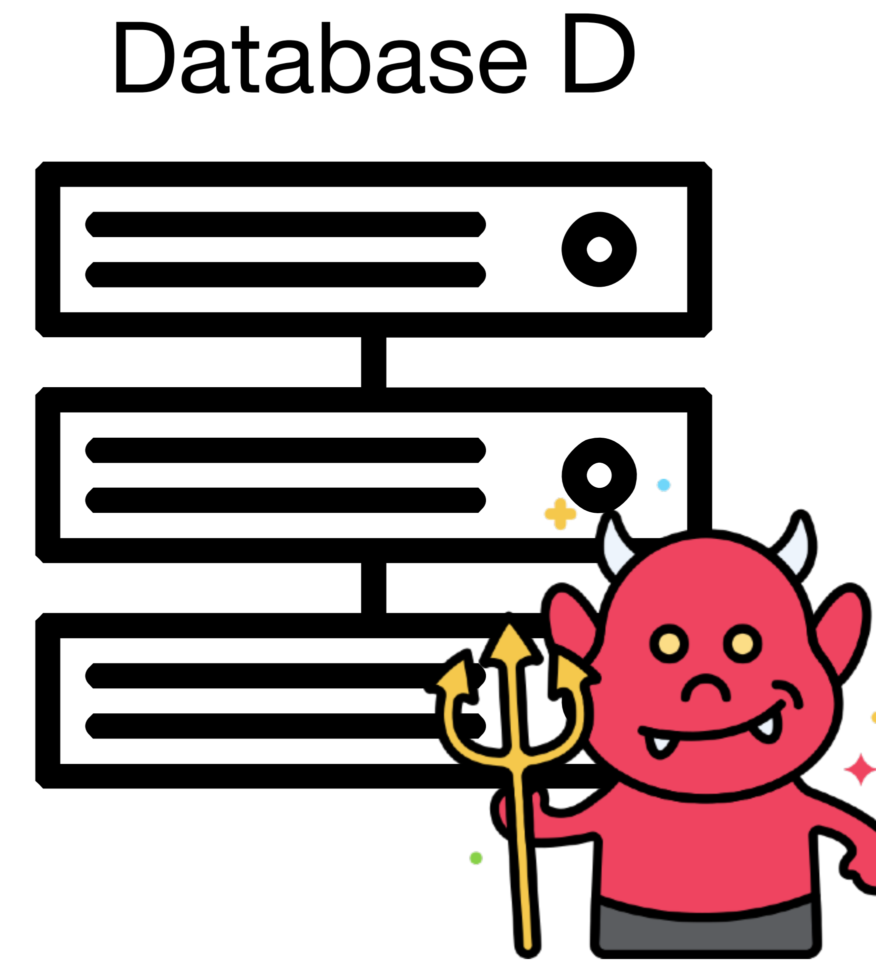


Integrity of Cloud Servers

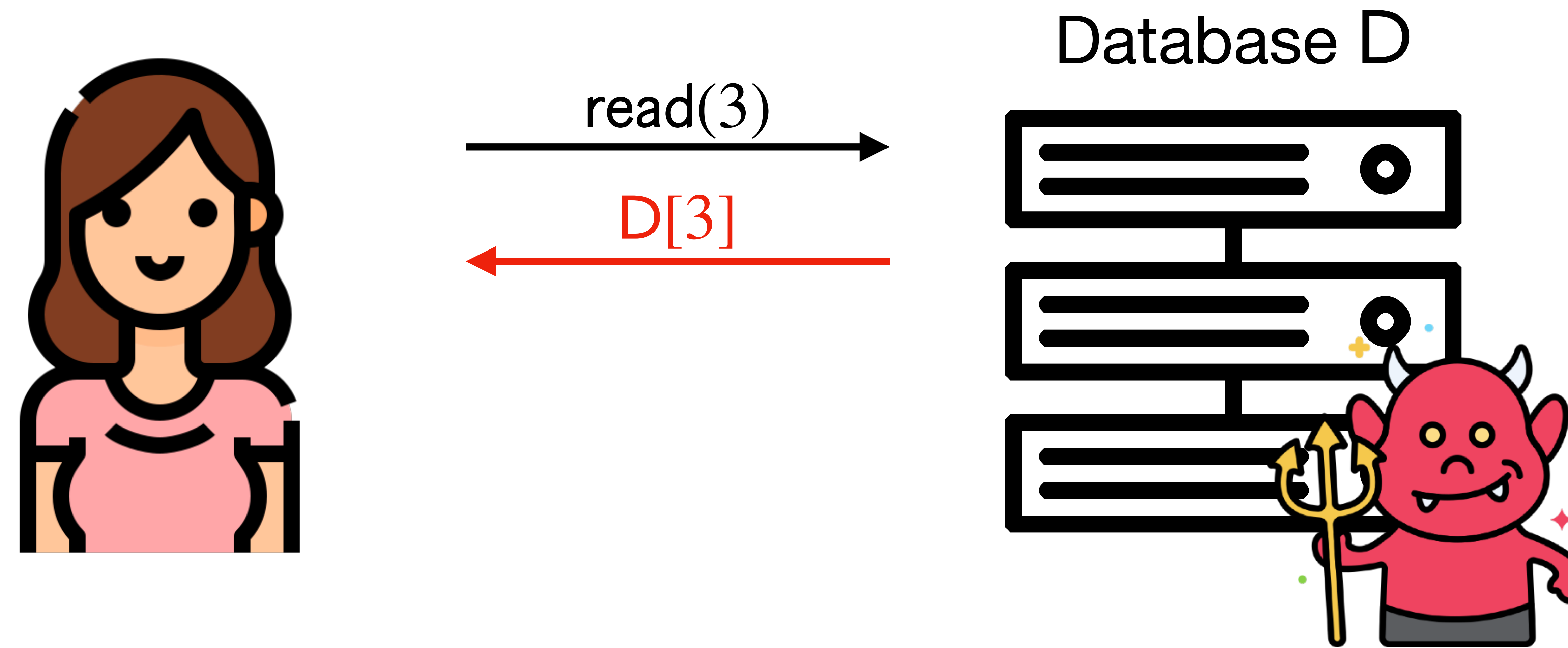


read(3)

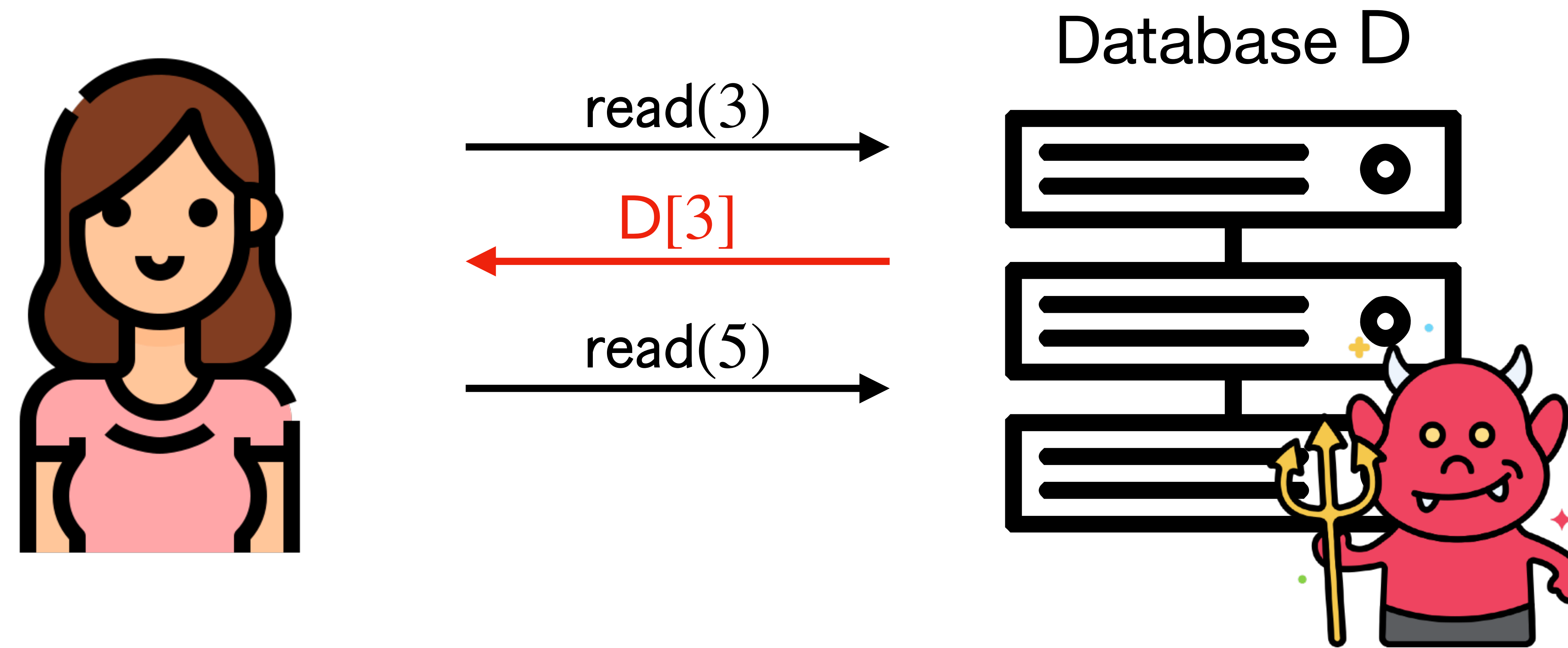
Database D



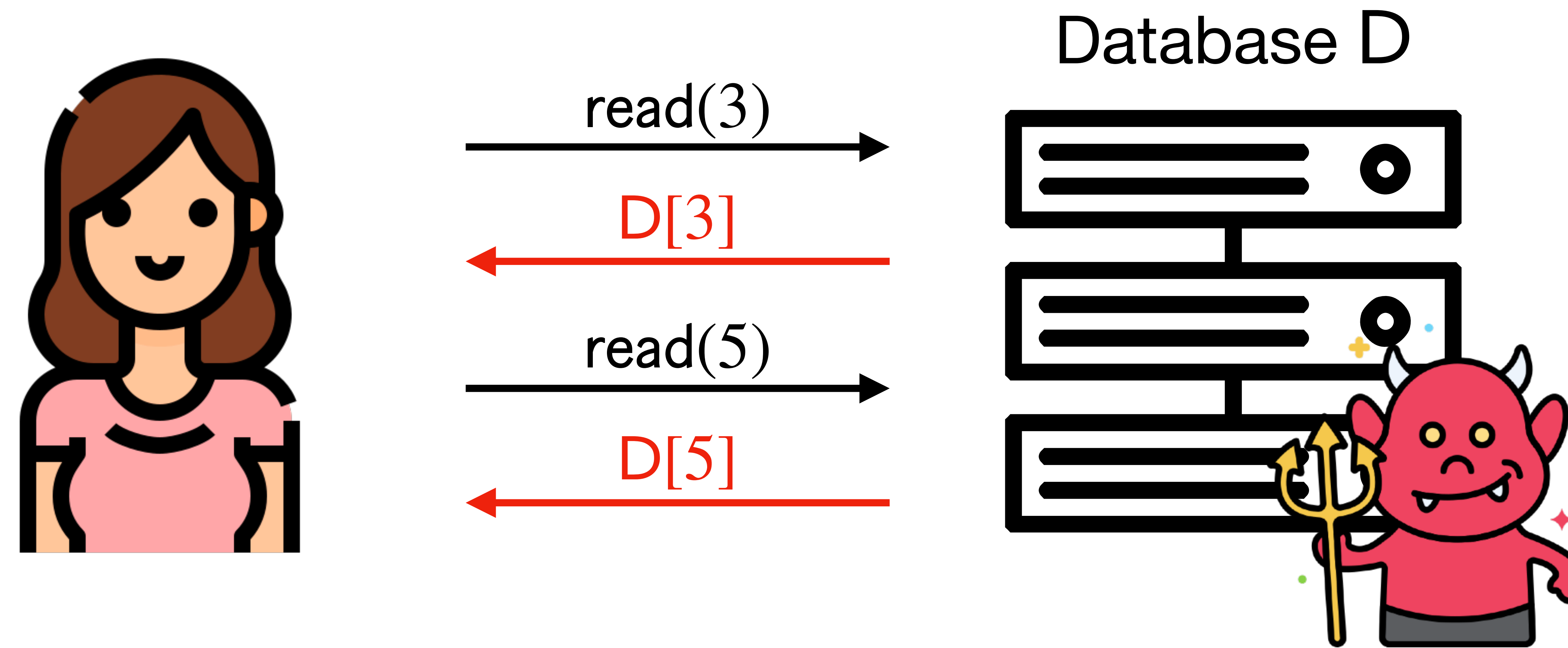
Integrity of Cloud Servers



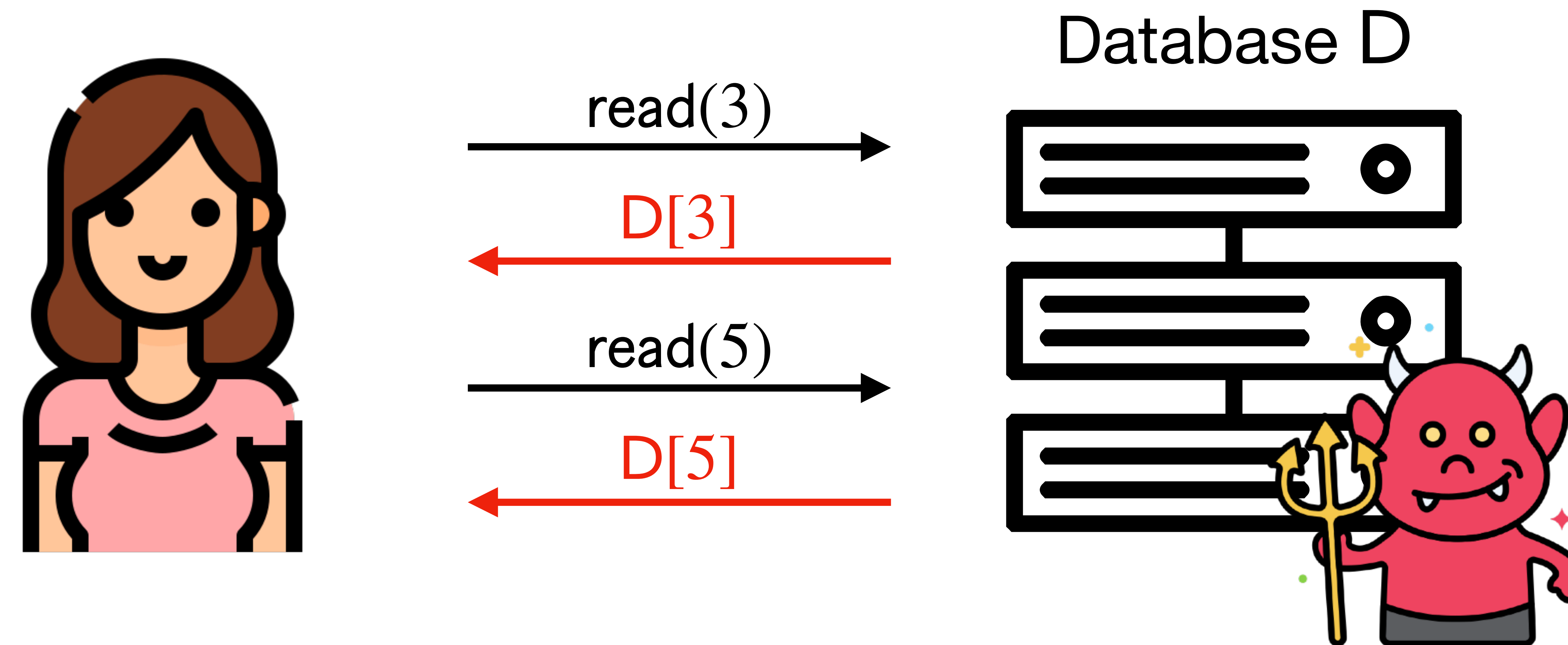
Integrity of Cloud Servers



Integrity of Cloud Servers

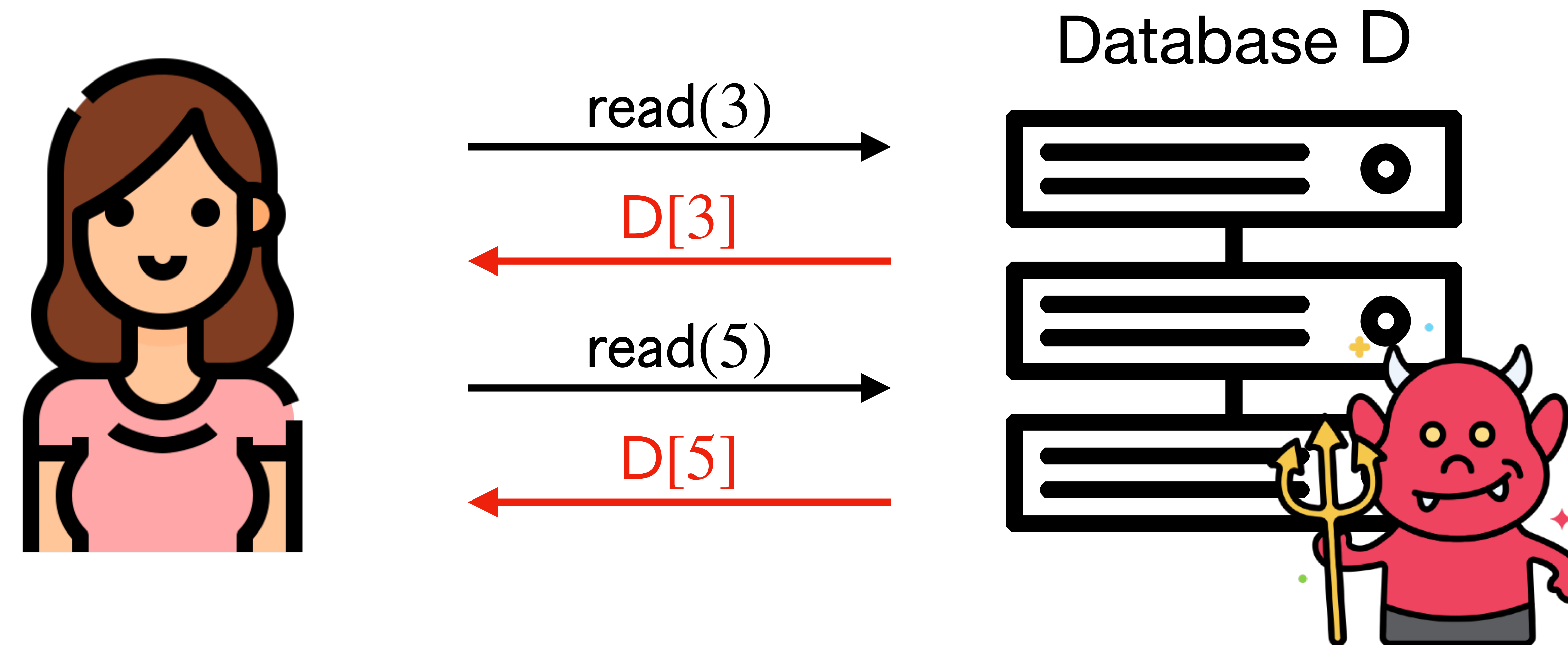


Integrity of Cloud Servers



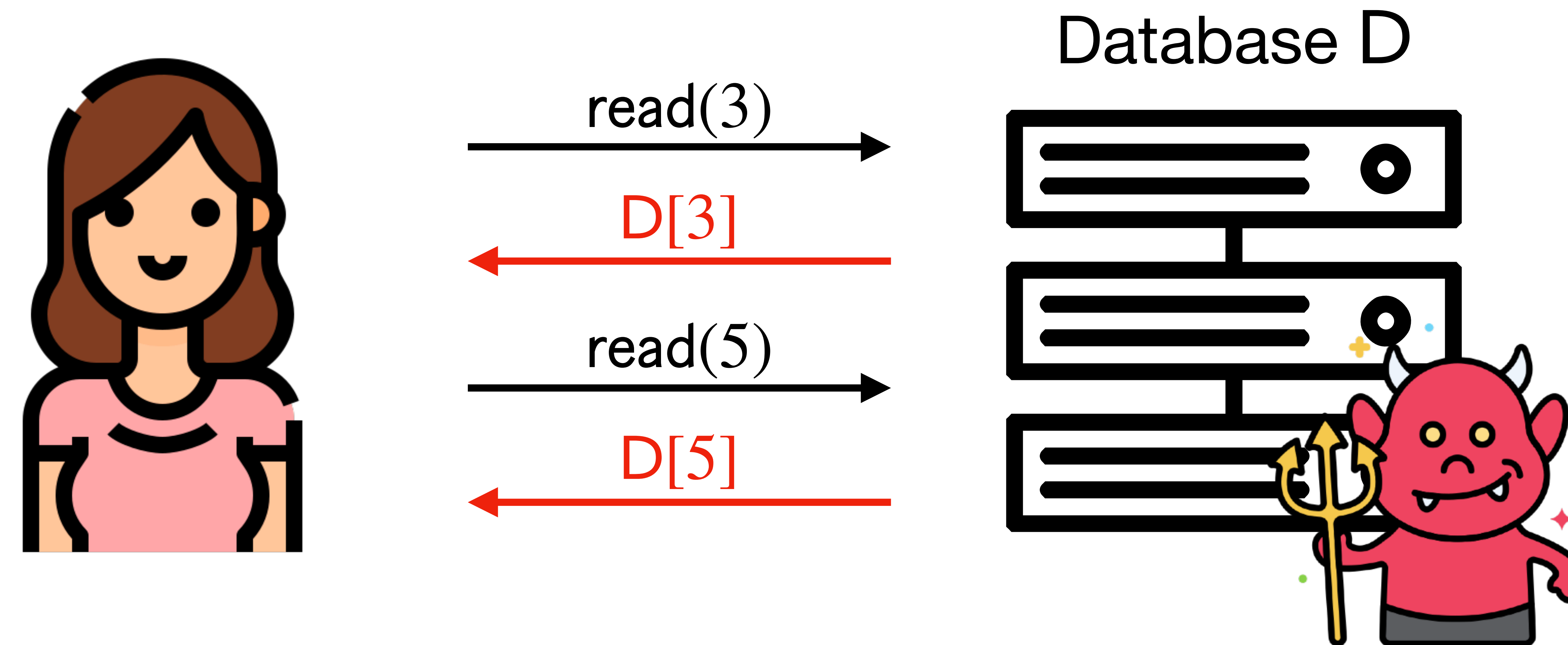
- How can a client use her **small** but **trusted** local memory to ensure that server is sending back correct responses?

Integrity of Cloud Servers



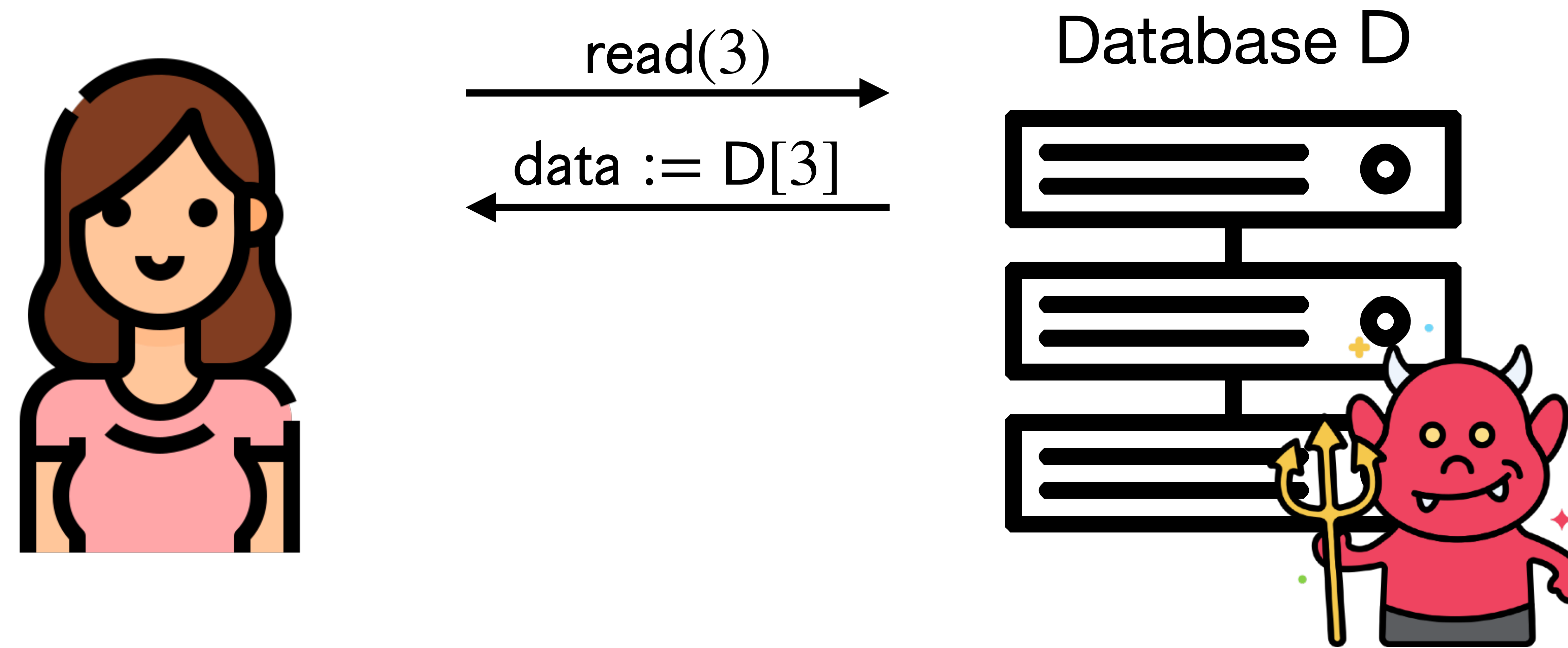
- How can a client use her **small** but **trusted** local memory to ensure that server is sending back correct responses?
- Answer: **Authentication**

Integrity of Cloud Servers

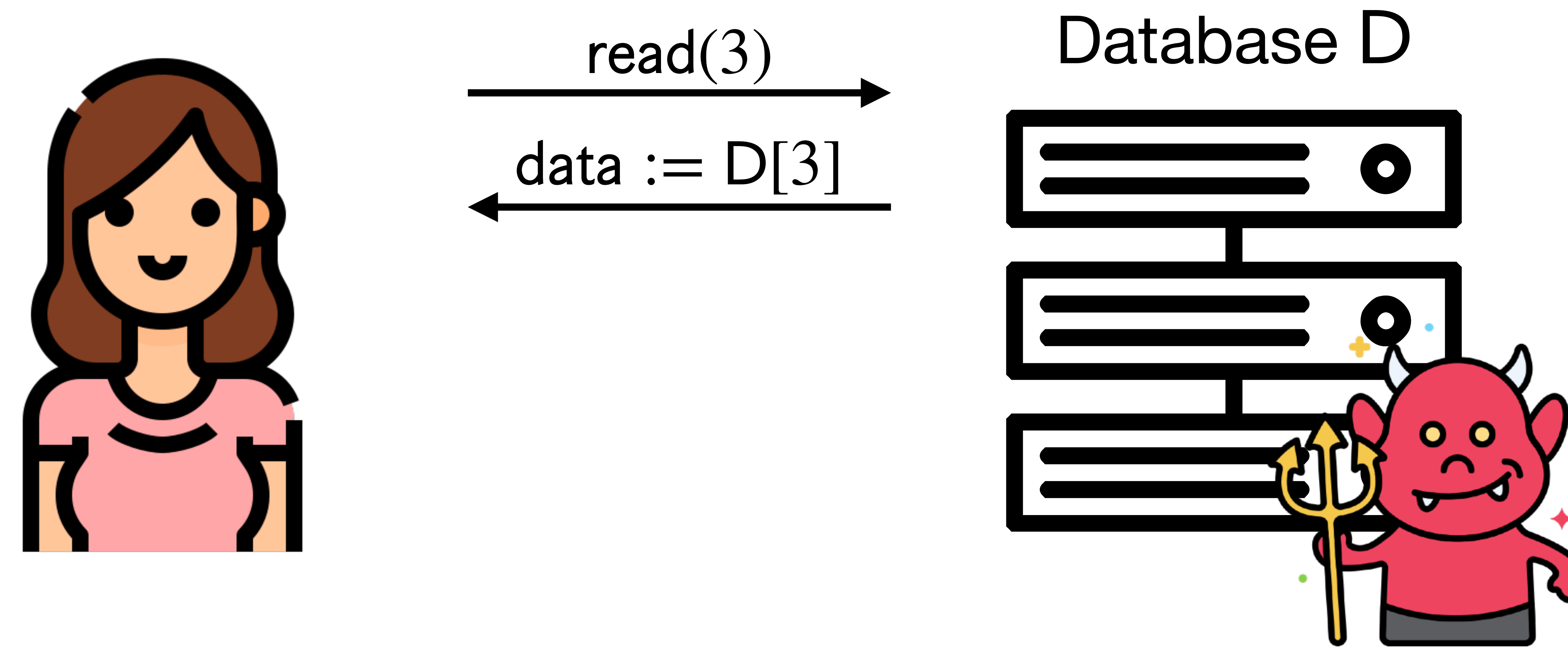


- How can a client use her **small** but **trusted** local memory to ensure that server is sending back correct responses?
- Answer: **Authentication**… if the database is static

Integrity of Cloud Servers

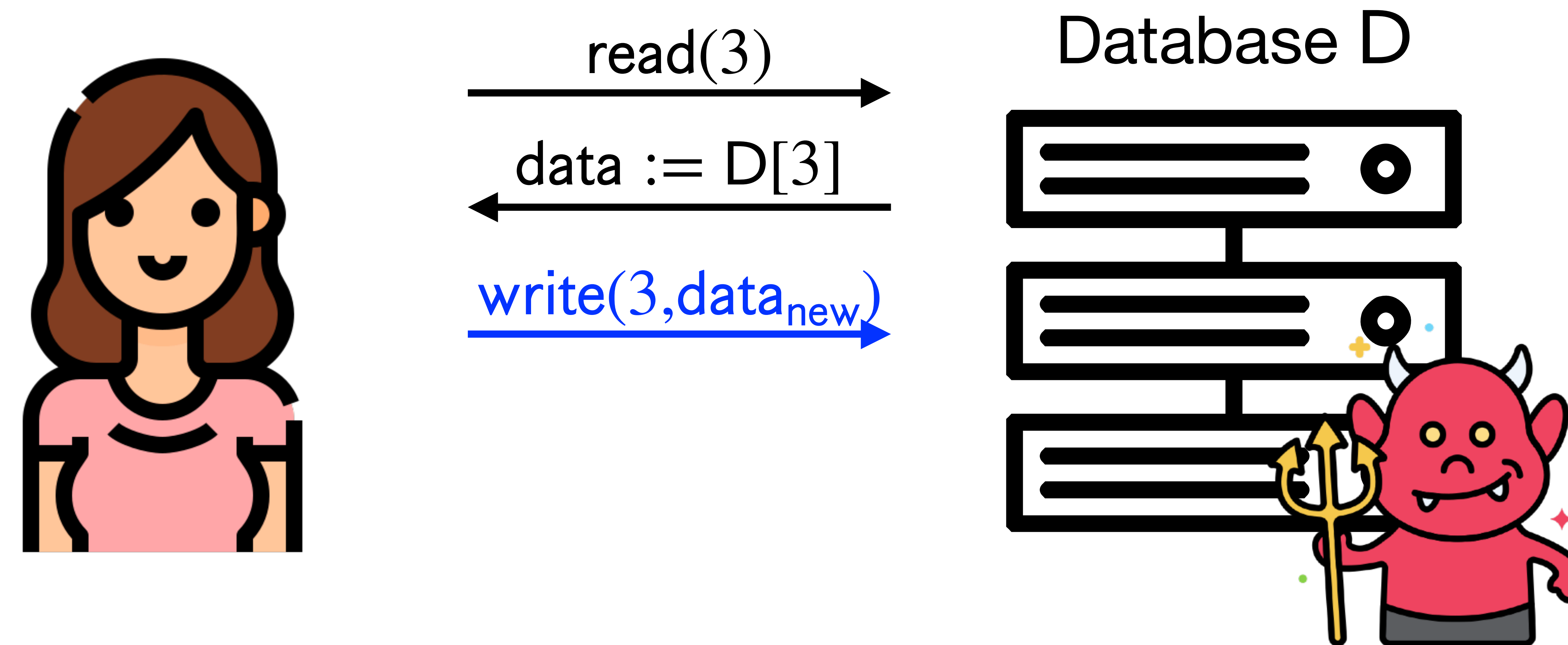


Integrity of Cloud Servers



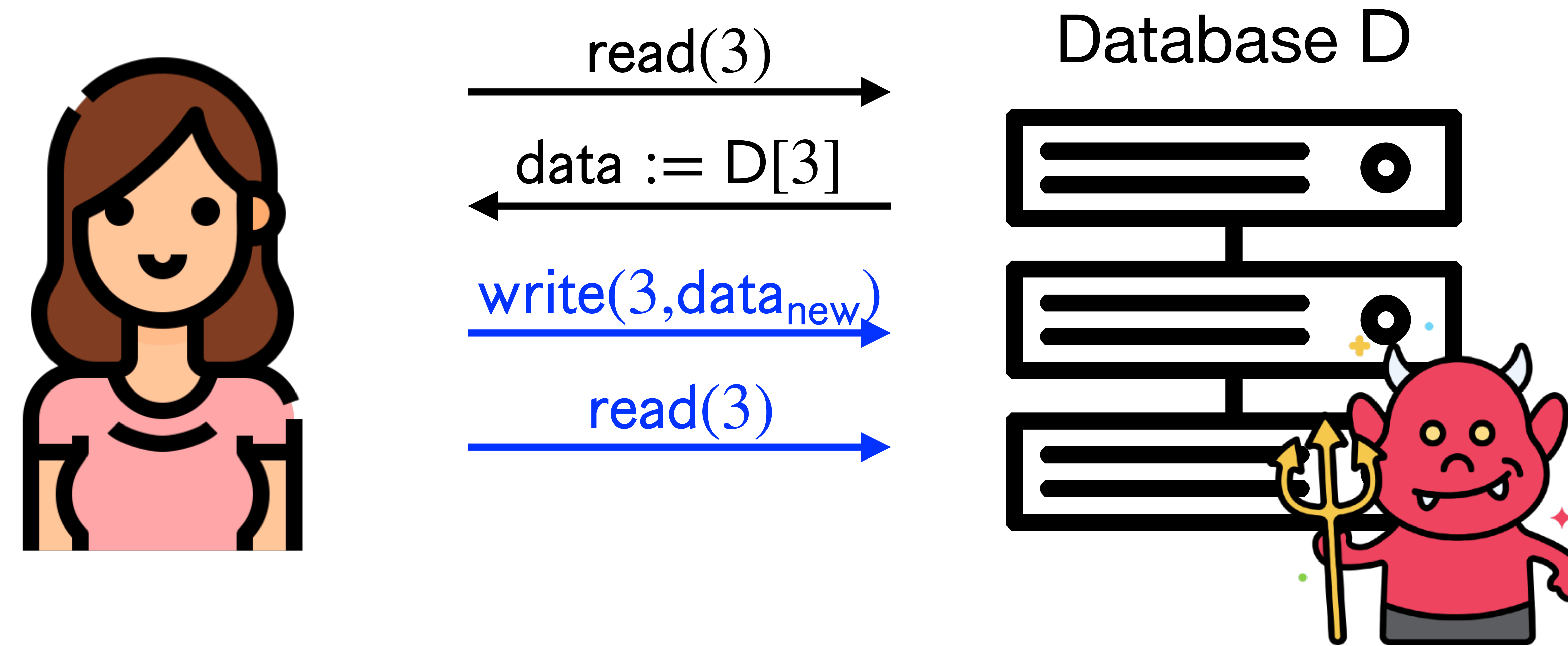
- What if the database contents are **dynamically** updated?

Integrity of Cloud Servers



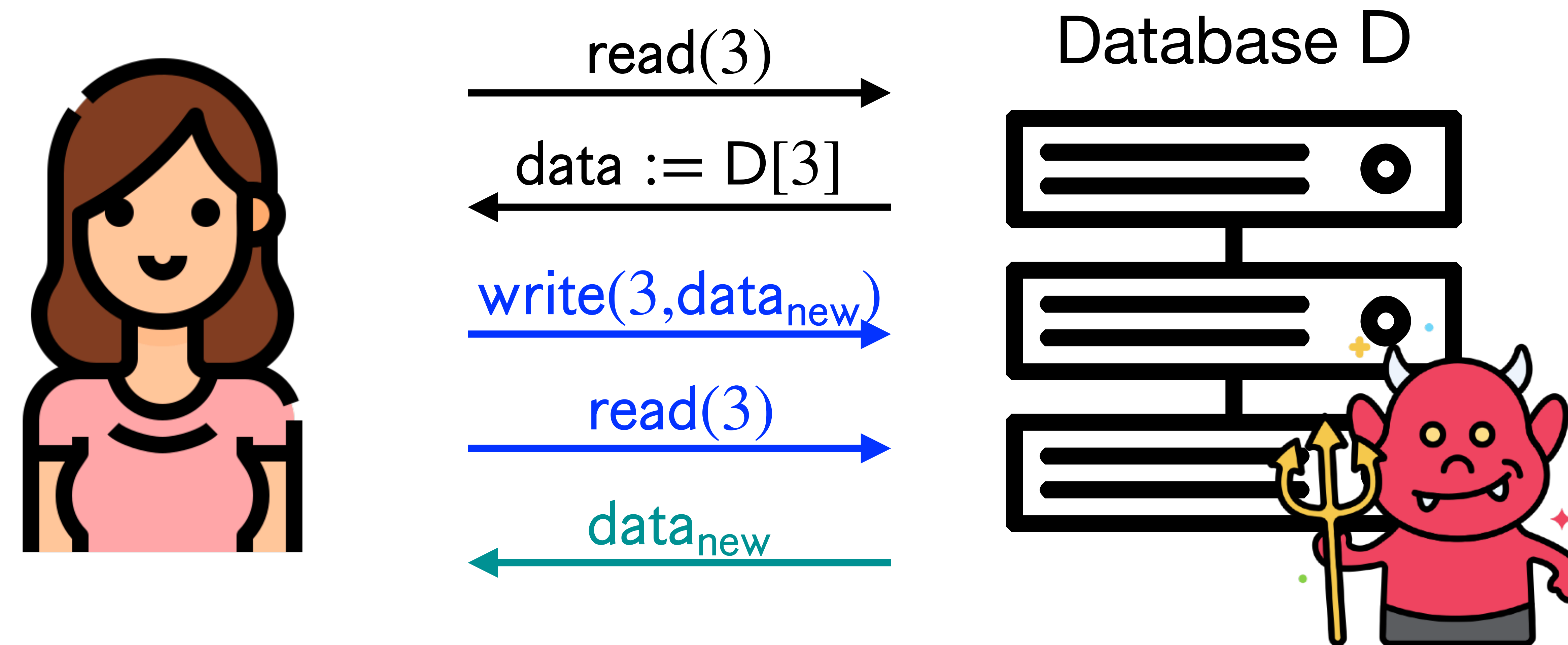
- What if the database contents are **dynamically** updated?

Integrity of Cloud Servers



- What if the database contents are **dynamically** updated?

Integrity of Cloud Servers



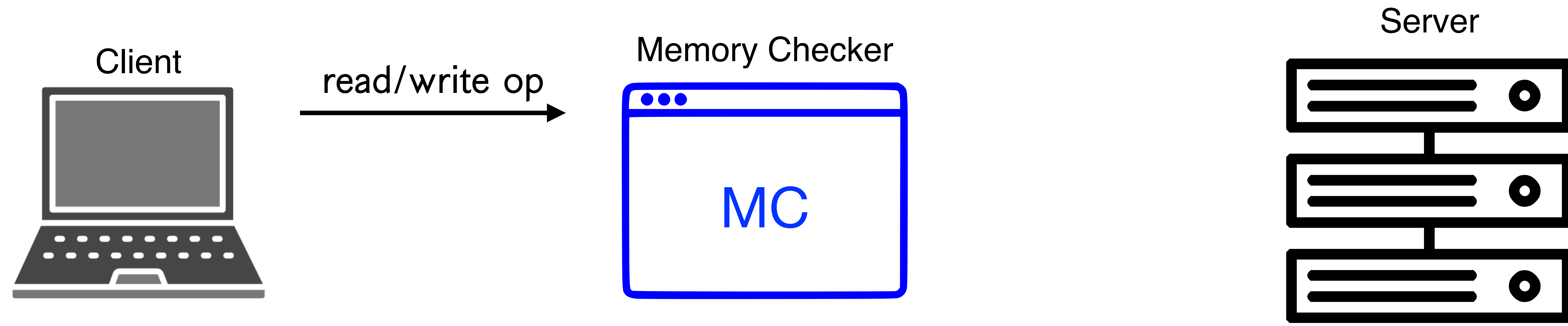
- What if the database contents are **dynamically** updated?
- We want reads to correspond to most recent version! (i.e. data_{new} not data)

Memory Checking

[Blum, Evans, Gemmel, Kannan, Naor ’94]

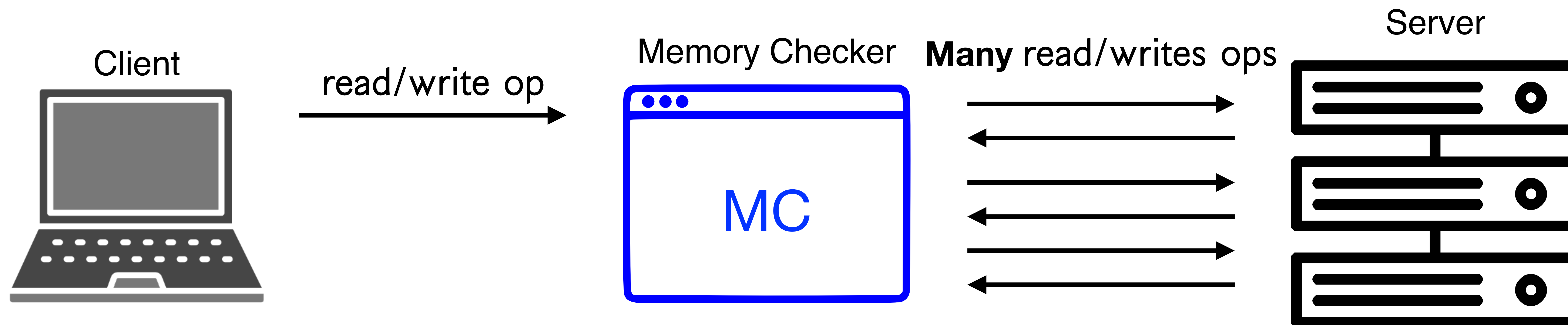
Memory Checking

[Blum, Evans, Gemmel, Kannan, Naor ’94]



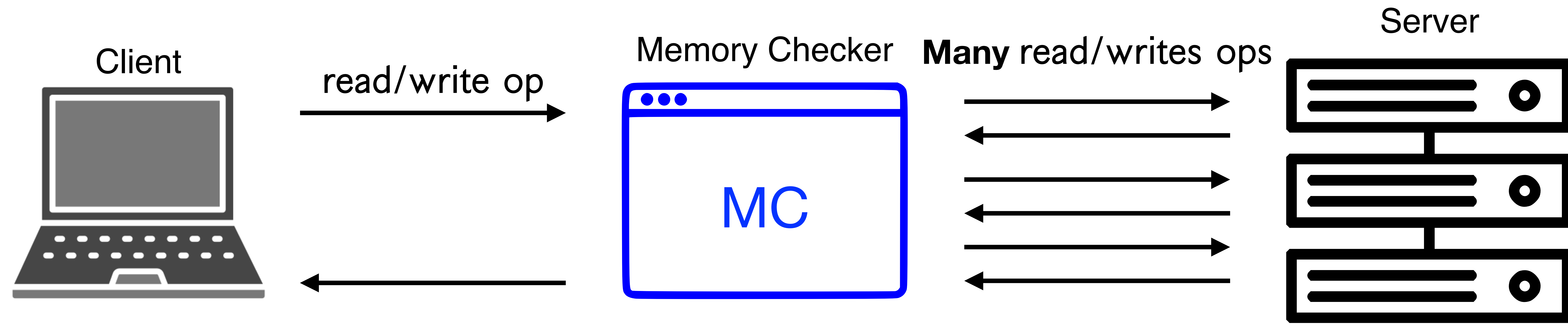
Memory Checking

[Blum, Evans, Gemmel, Kannan, Naor ’94]



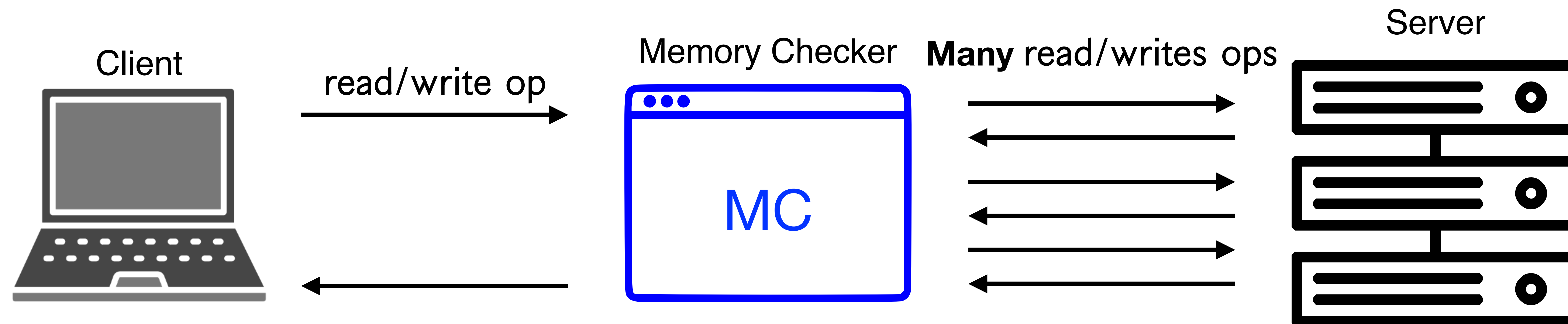
Memory Checking

[Blum, Evans, Gemmel, Kannan, Naor '94]



Memory Checking

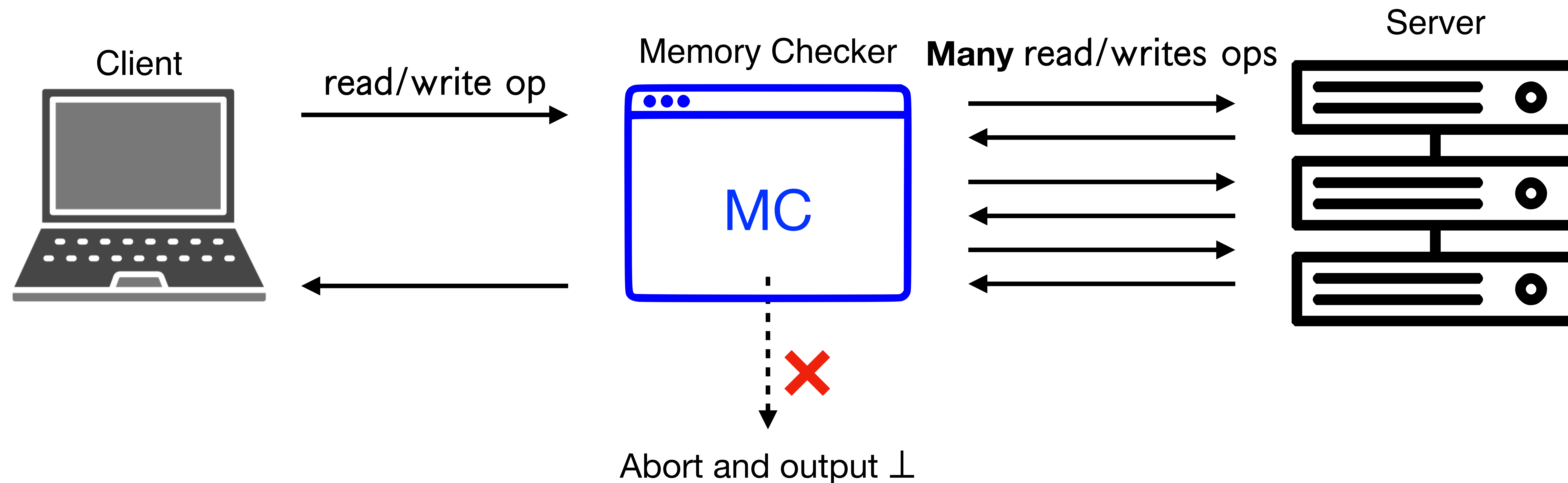
[Blum, Evans, Gemmel, Kannan, Naor '94]



- **Correctness**: For any PPT malicious server, MC either **aborts** or gives correct (i.e. most recent version of address) responses.

Memory Checking

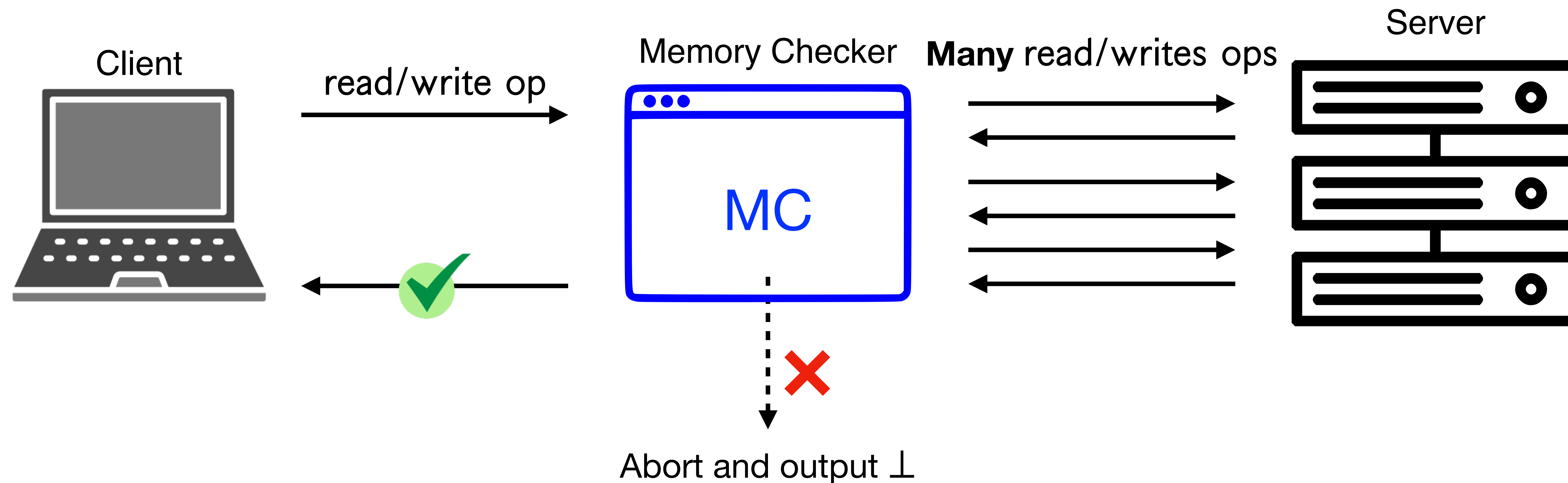
[Blum, Evans, Gemmel, Kannan, Naor ’94]



- **Correctness**: For any PPT malicious server, MC either **aborts** or gives correct (i.e. most recent version of address) responses.

Memory Checking

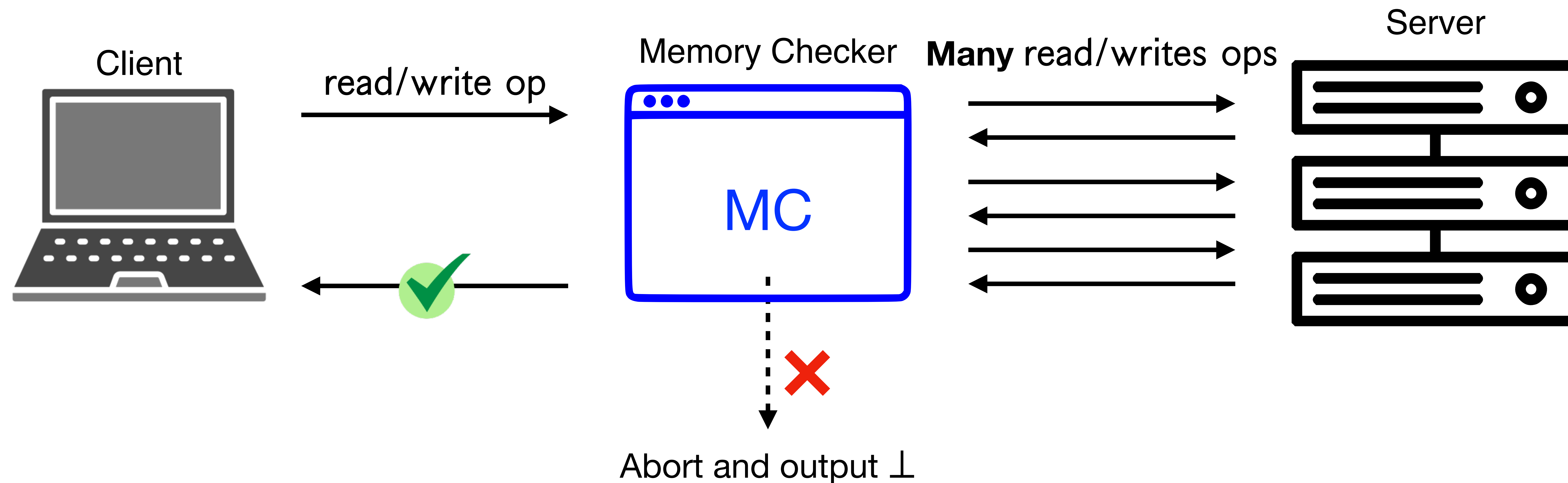
[Blum, Evans, Gemmel, Kannan, Naor ’94]



- **Correctness**: For any PPT malicious server, MC either **aborts** or gives correct (i.e. most recent version of address) responses.

Memory Checking

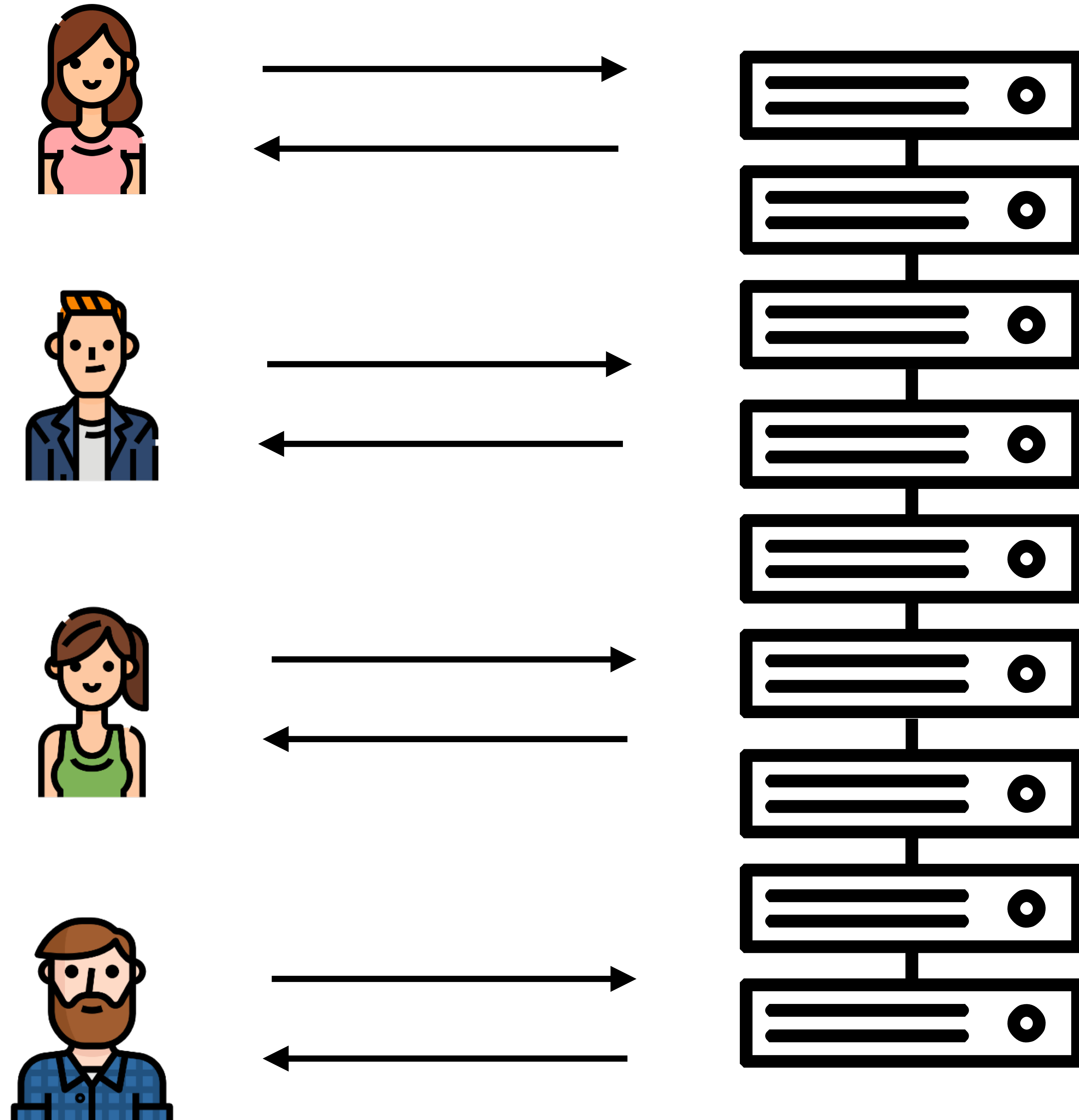
[Blum, Evans, Gemmel, Kannan, Naor '94]



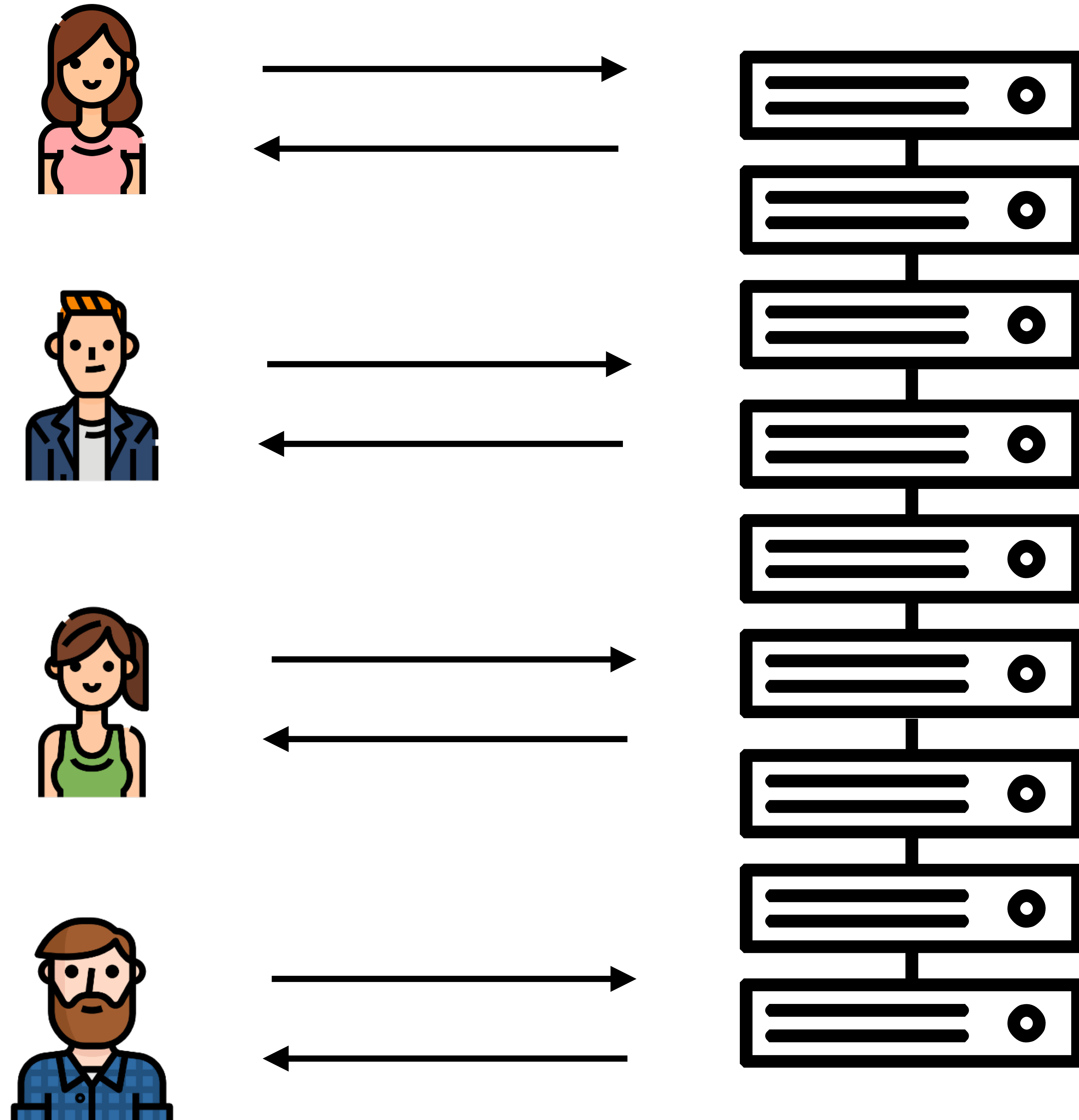
- **Correctness**: For any PPT malicious server, MC either **aborts** or gives correct (i.e. most recent version of address) responses.

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

Multiple Users Sharing Memory

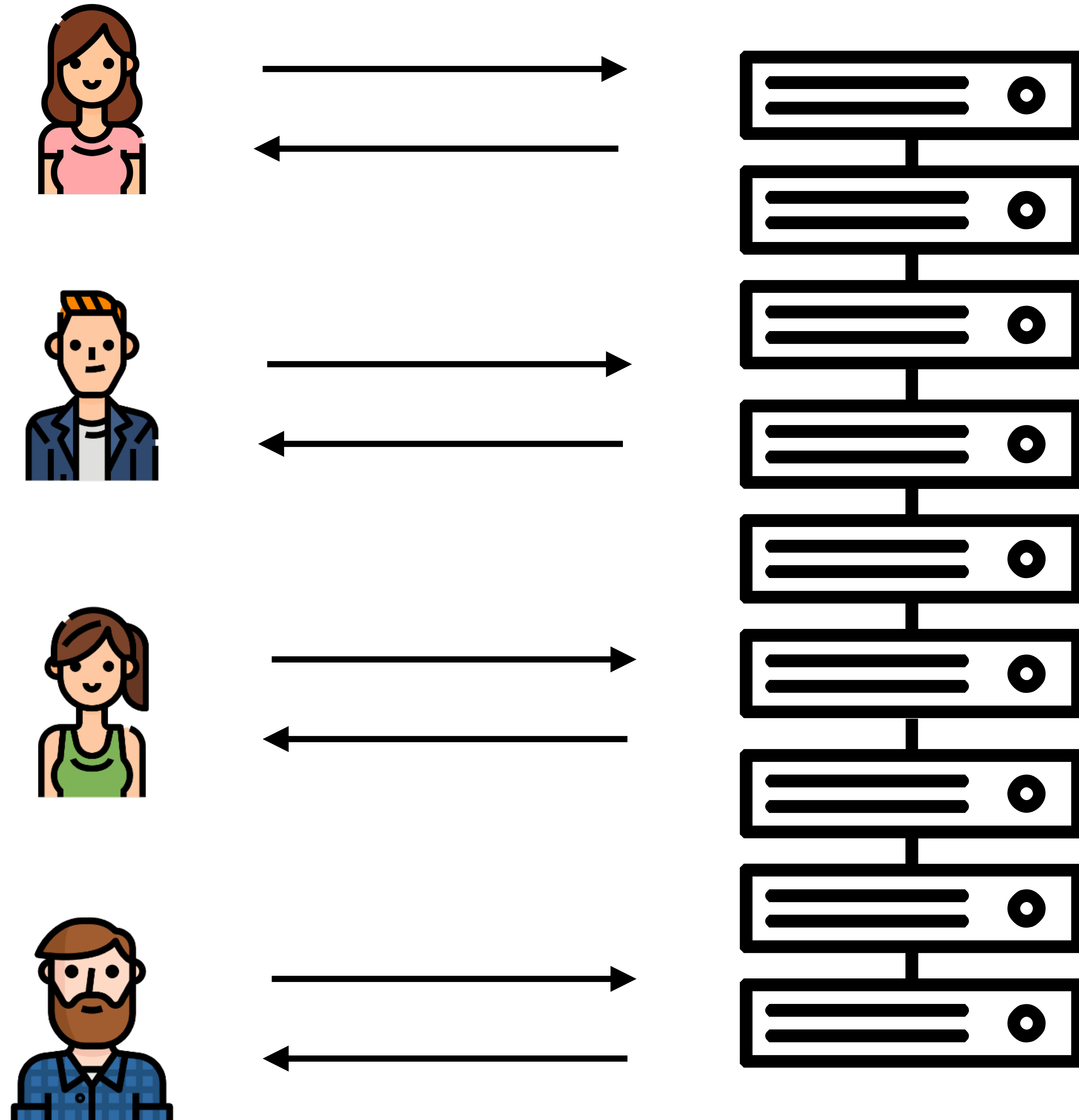


Multiple Users Sharing Memory



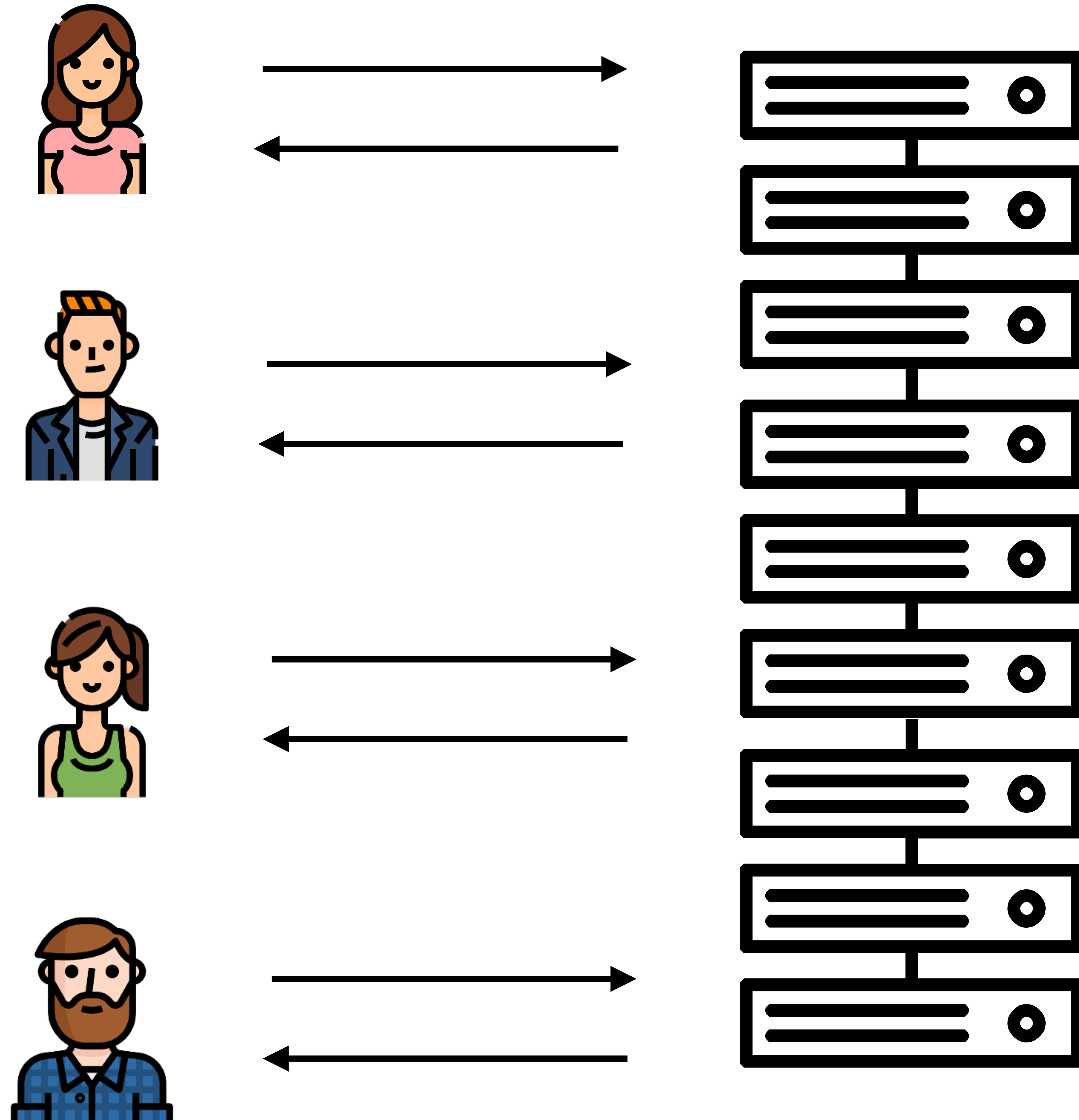
- **Examples**

Multiple Users Sharing Memory



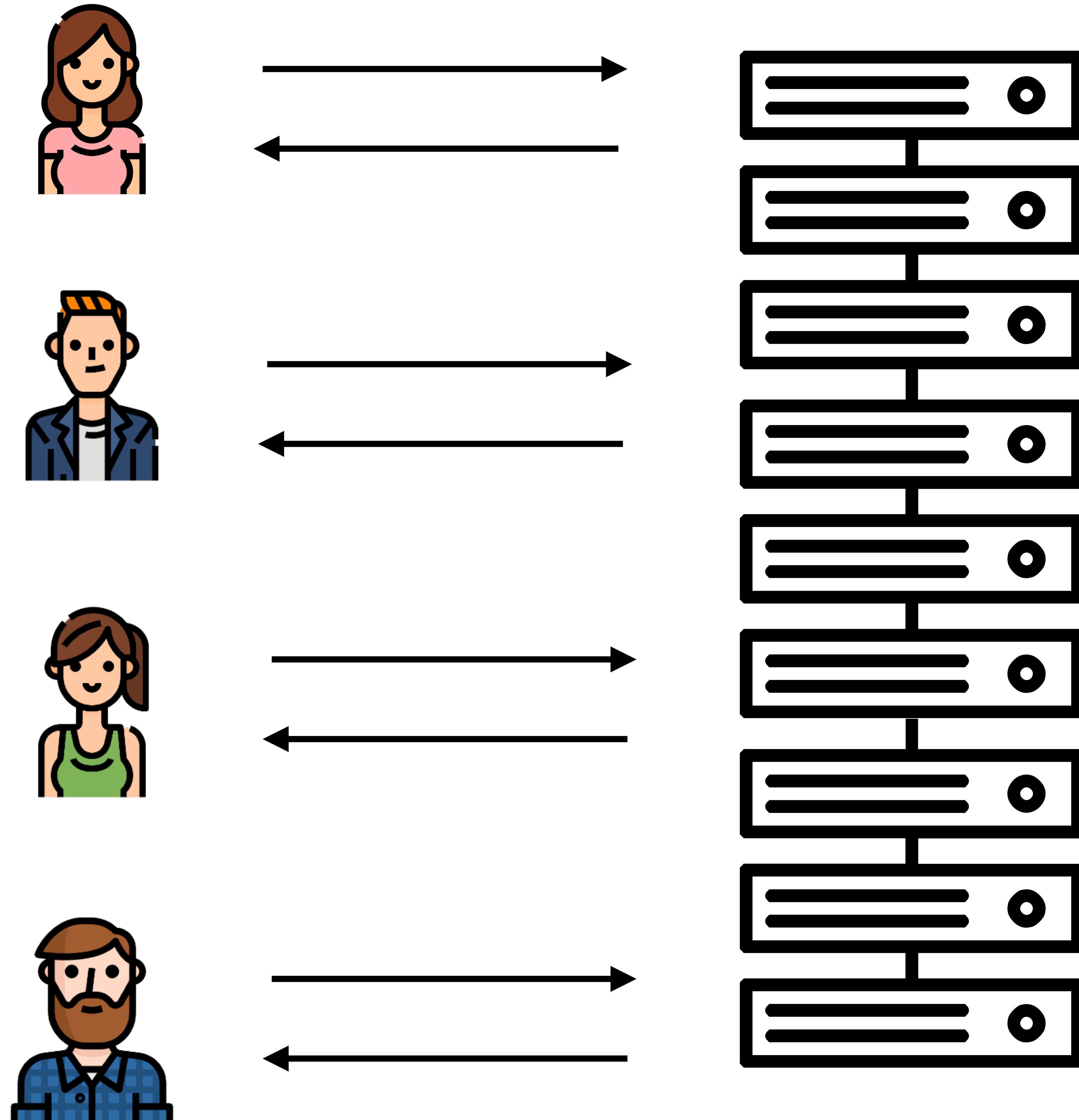
- **Examples**
  - Shared database across many clients

Multiple Users Sharing Memory



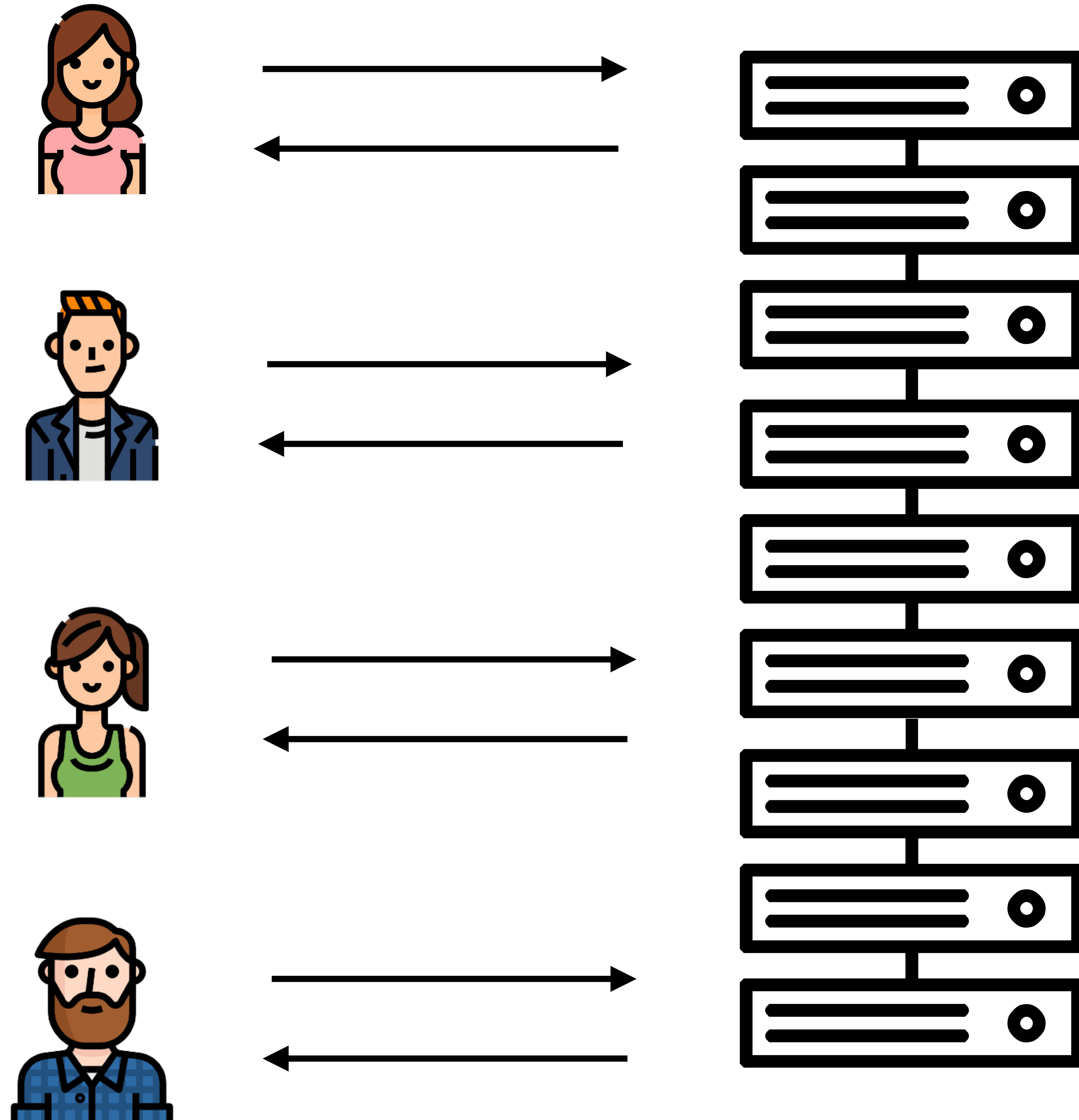
- **Examples**
  - Shared database across many clients
  - CPUs with shared memory

Multiple Users Sharing Memory



- **Examples**
  - Shared database across many clients
  - CPUs with shared memory
  - Distributed computing

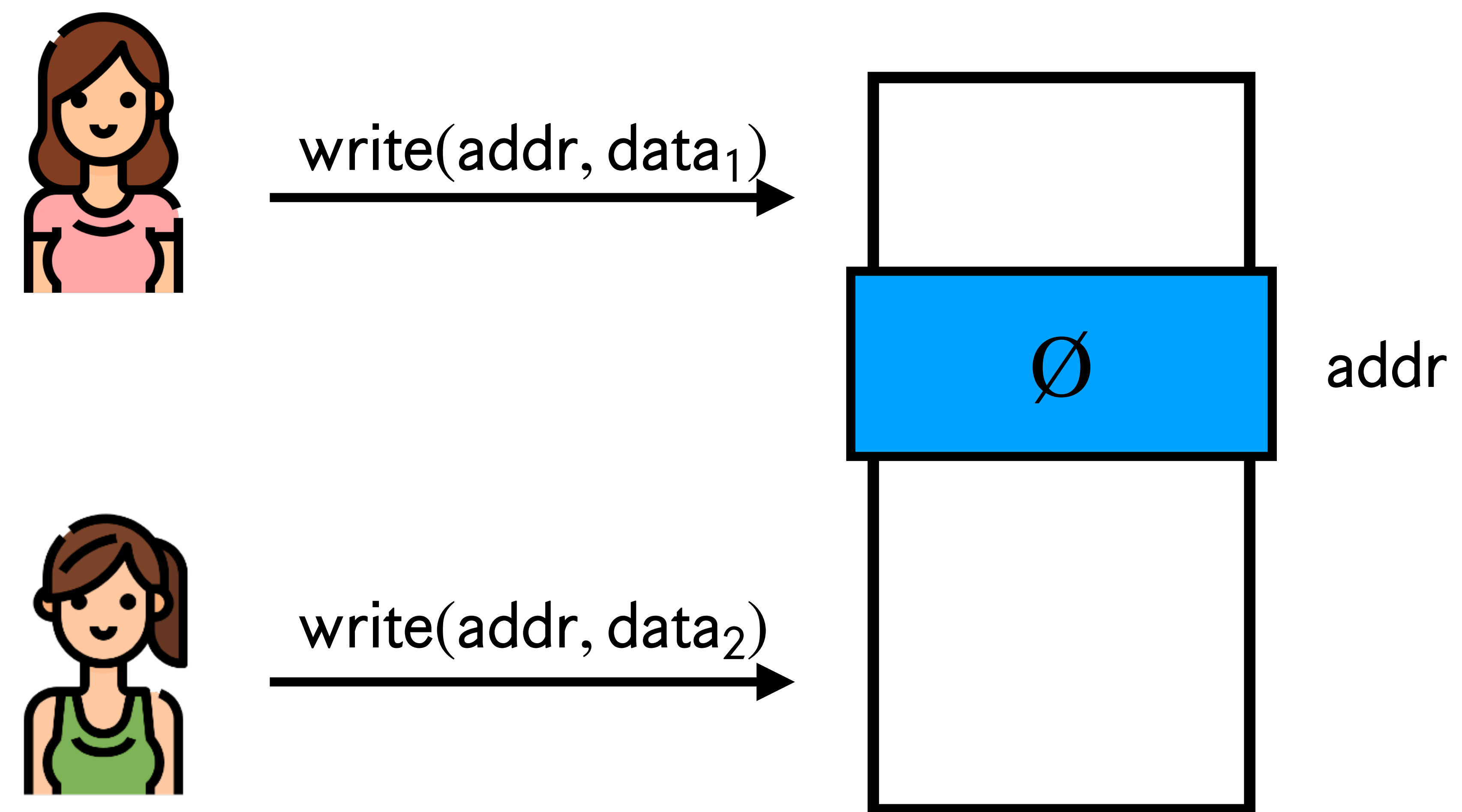
Multiple Users Sharing Memory



- **Examples**
  - Shared database across many clients
  - CPUs with shared memory
  - Distributed computing
- **Integrity verification is very useful here too!**

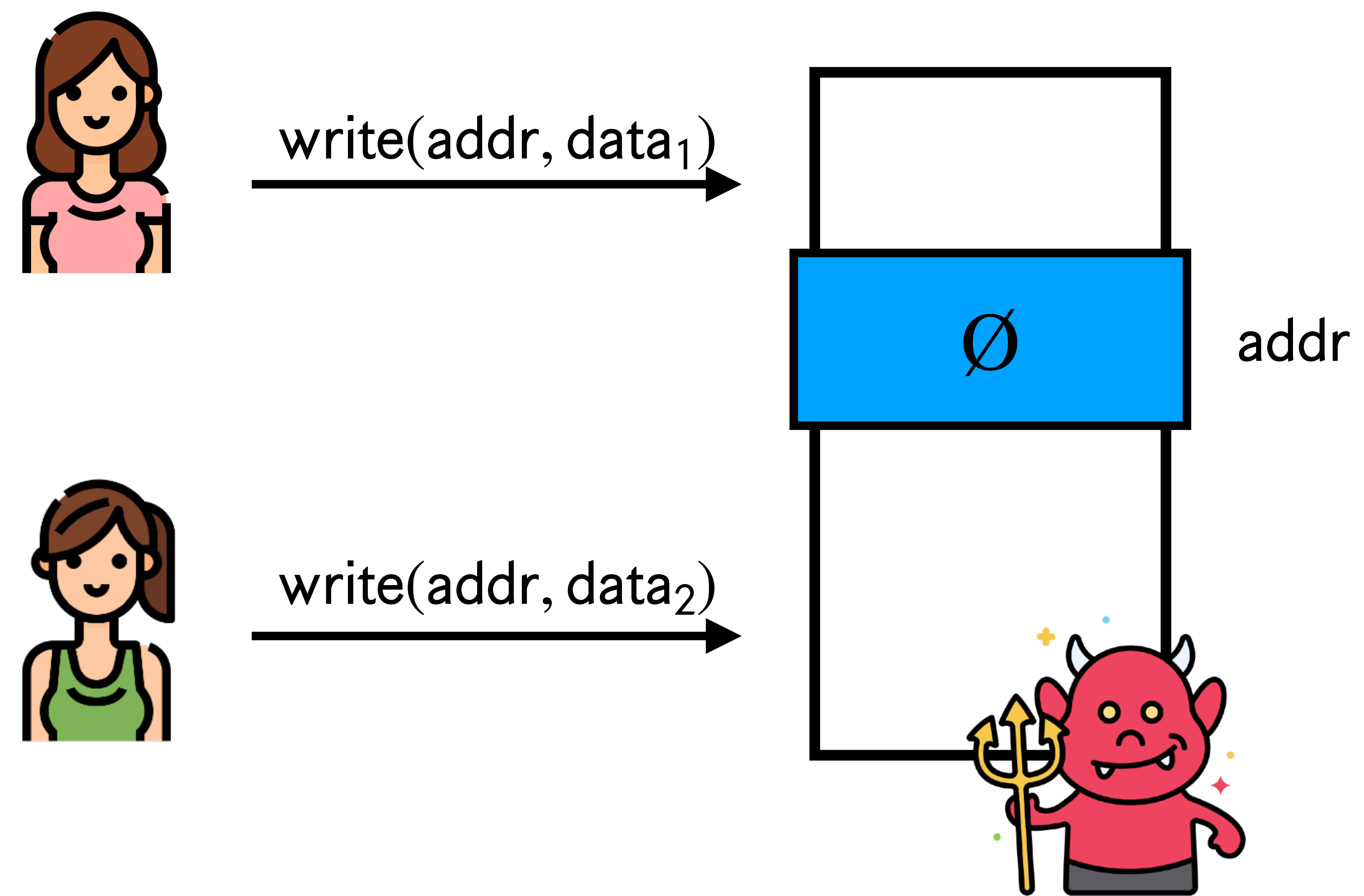
More Modes of Failure!

Consider two concurrent writes:



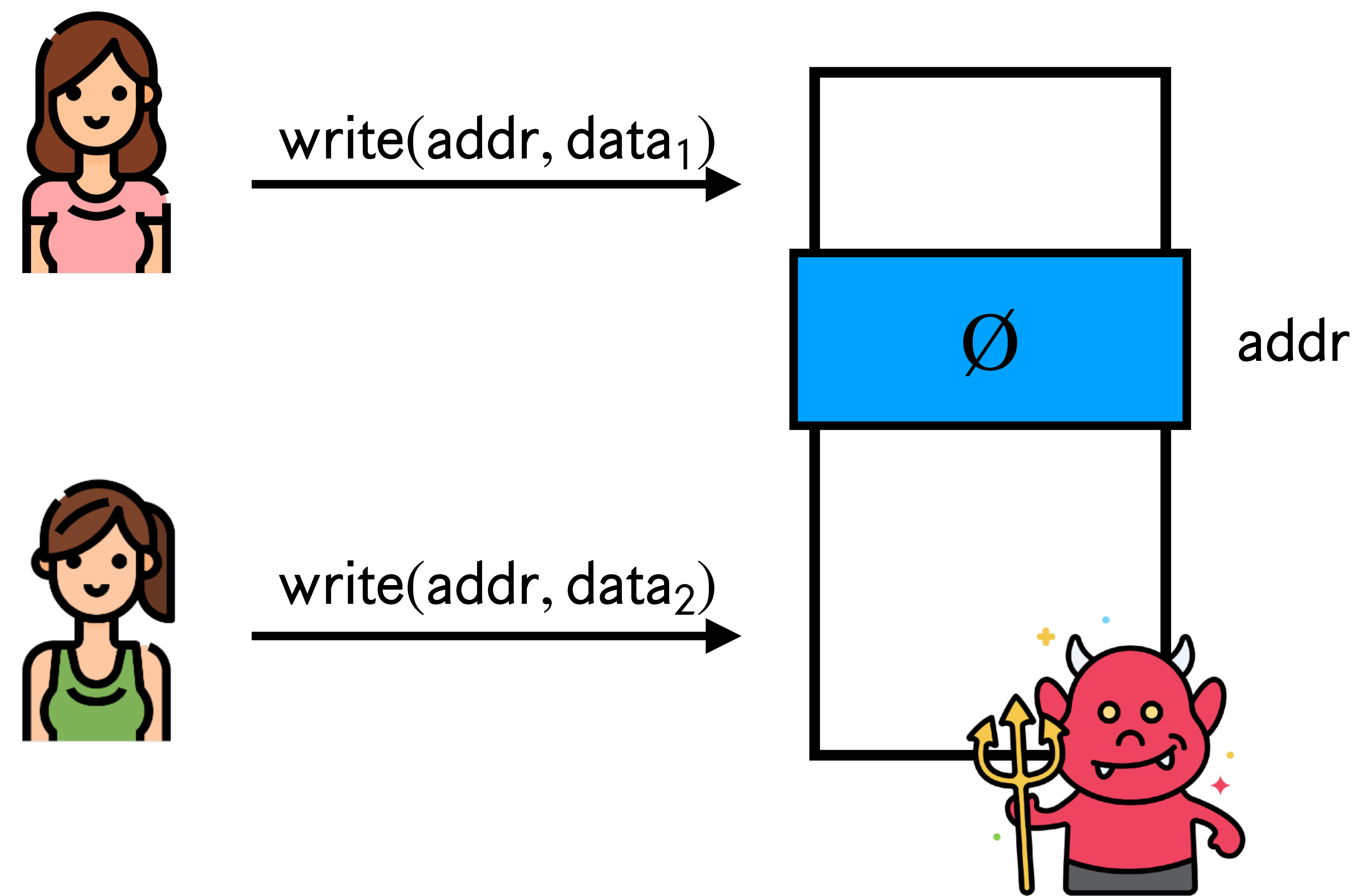
More Modes of Failure!

Consider two concurrent writes:



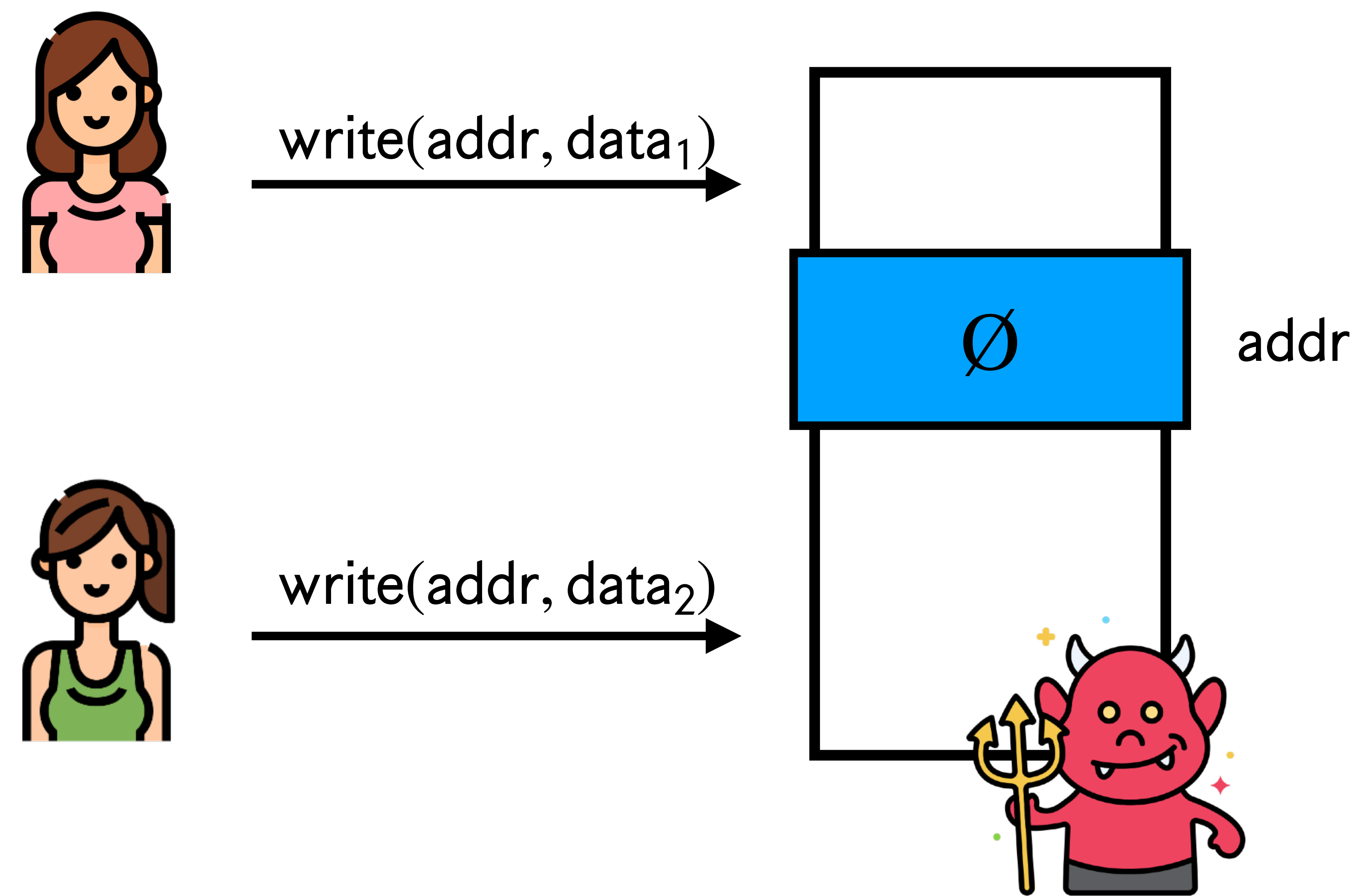
More Modes of Failure!

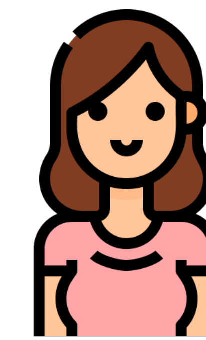
Consider two concurrent writes:

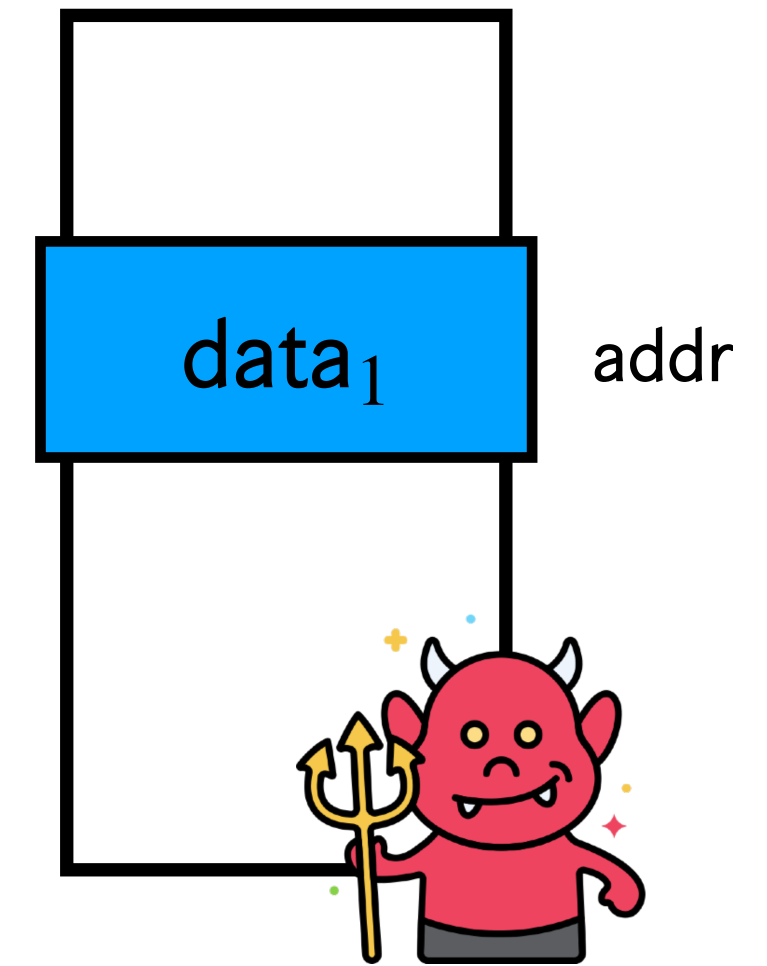


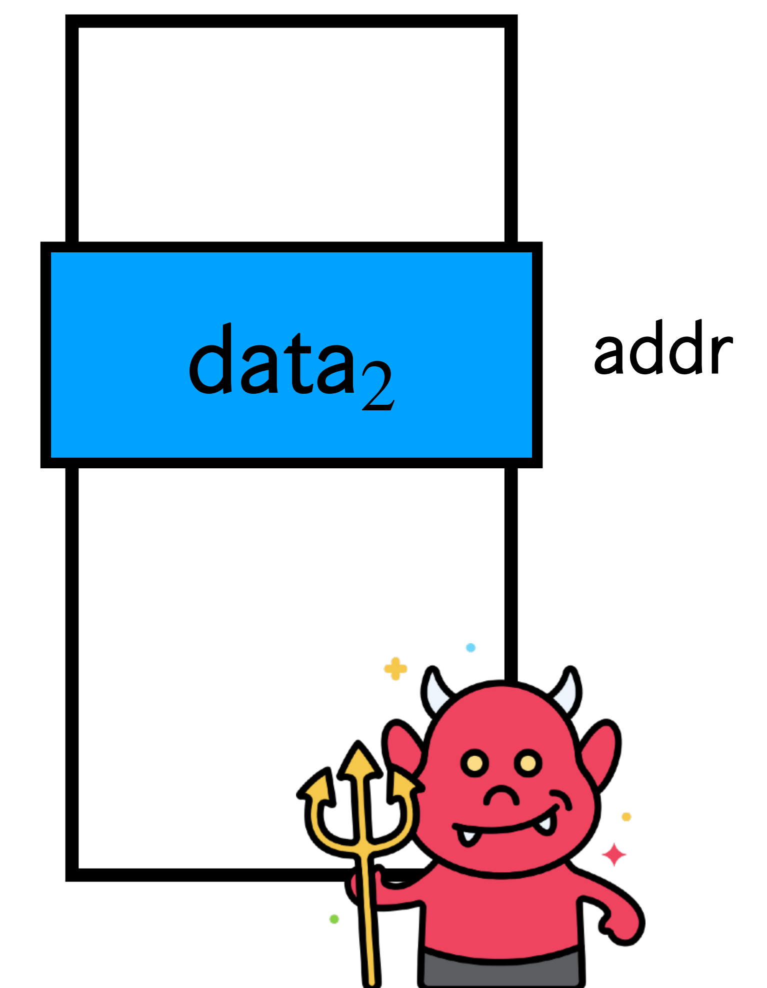
More Modes of Failure!

Consider two concurrent writes:



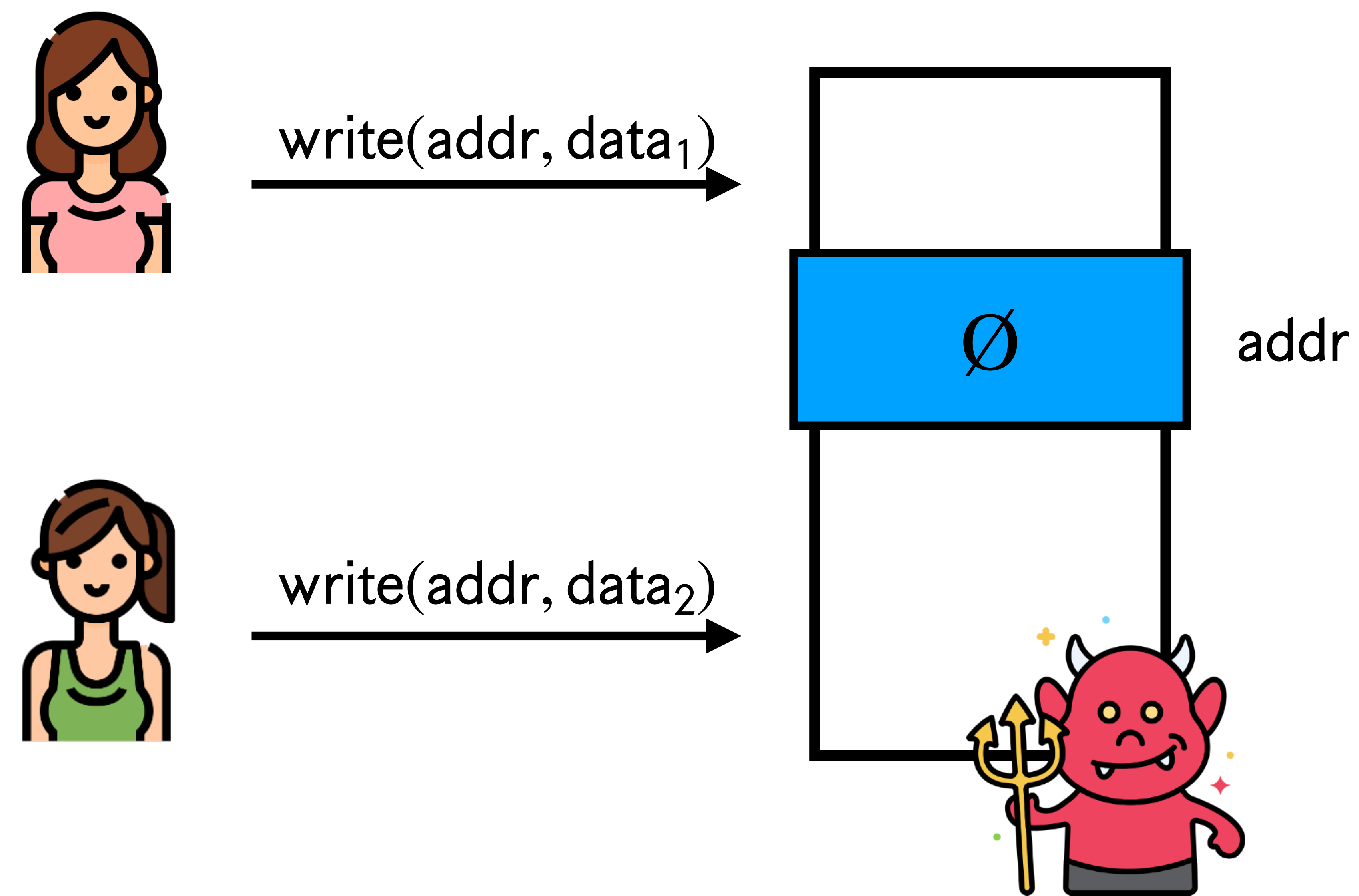


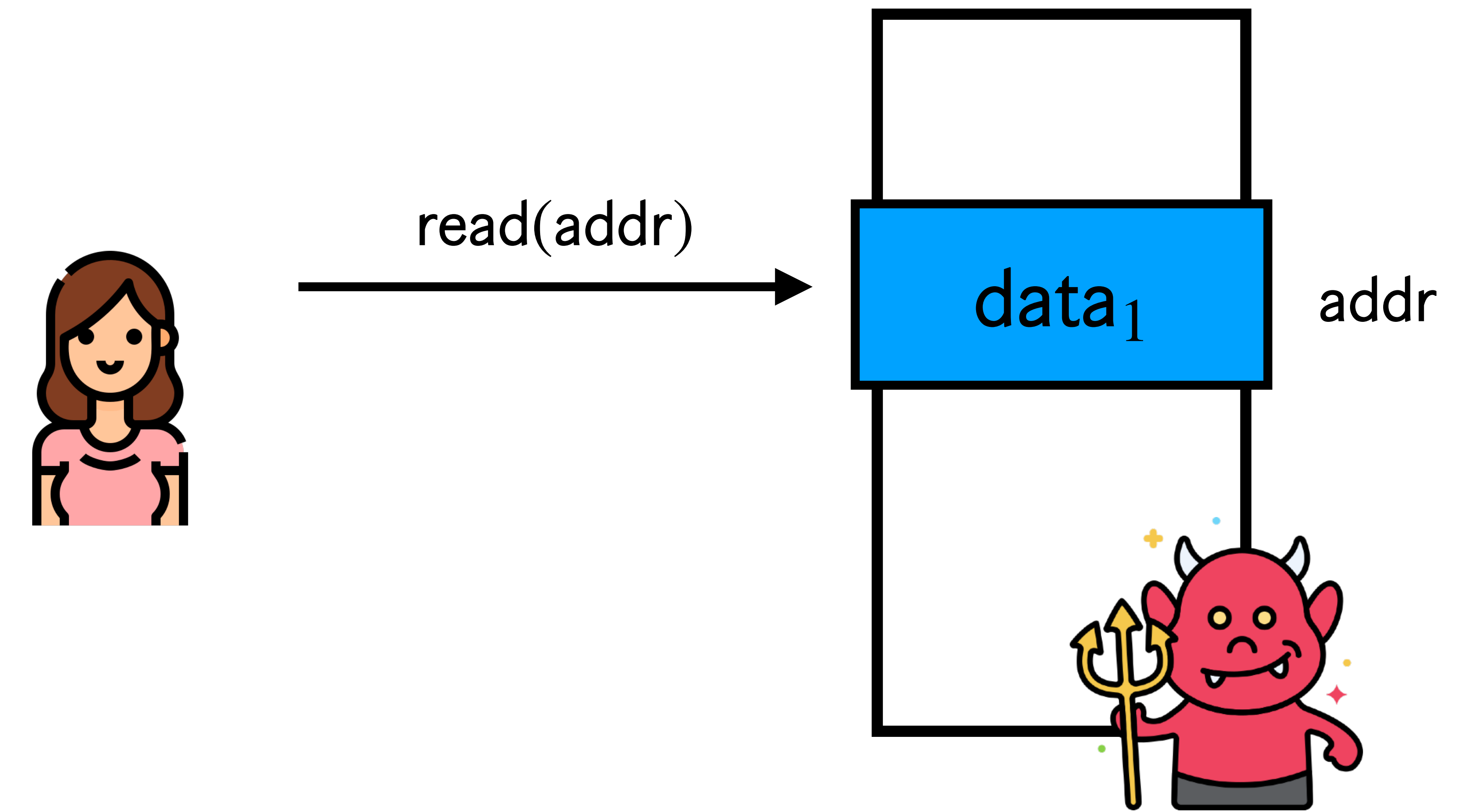


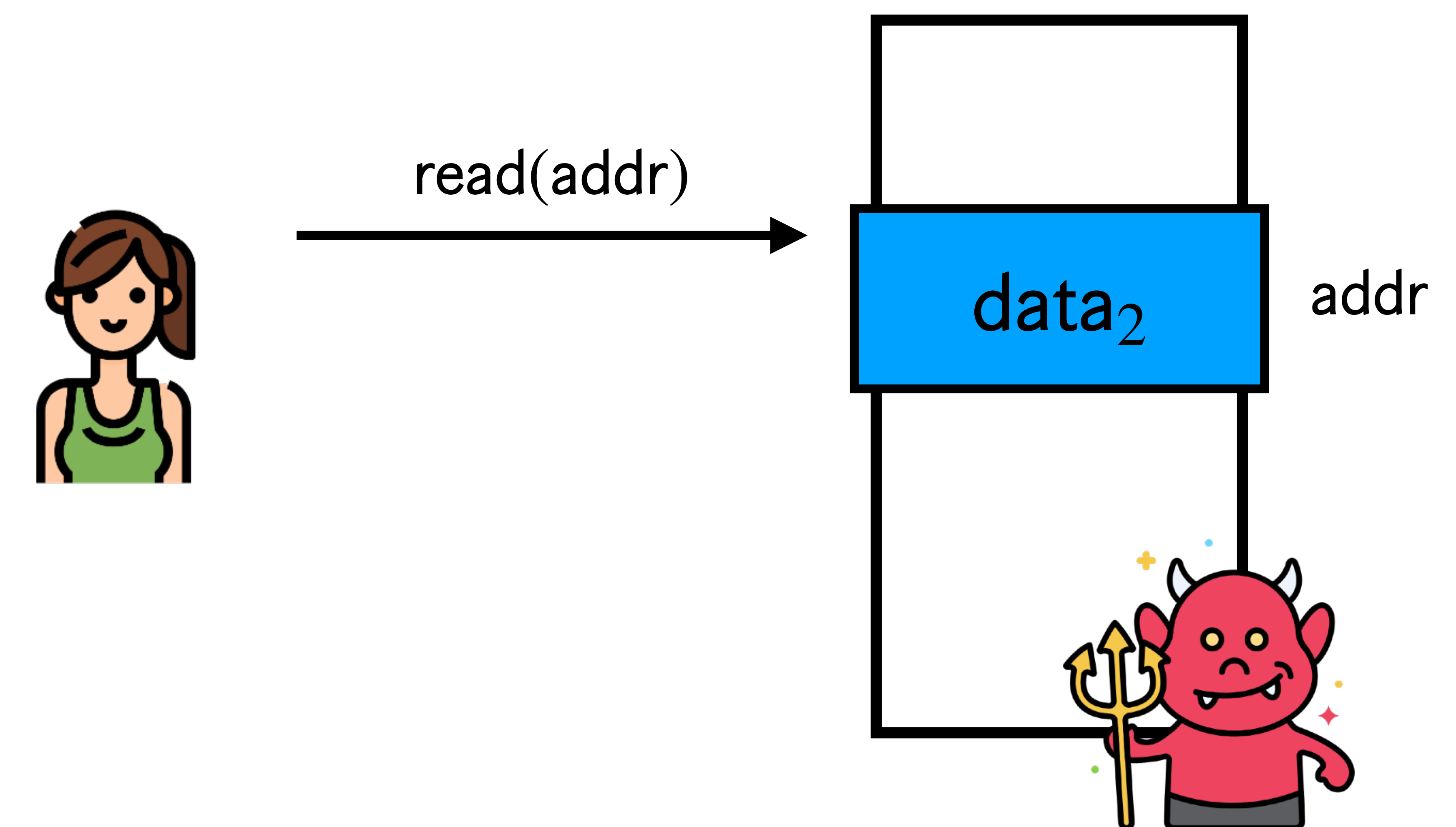


More Modes of Failure!

Consider two concurrent writes:

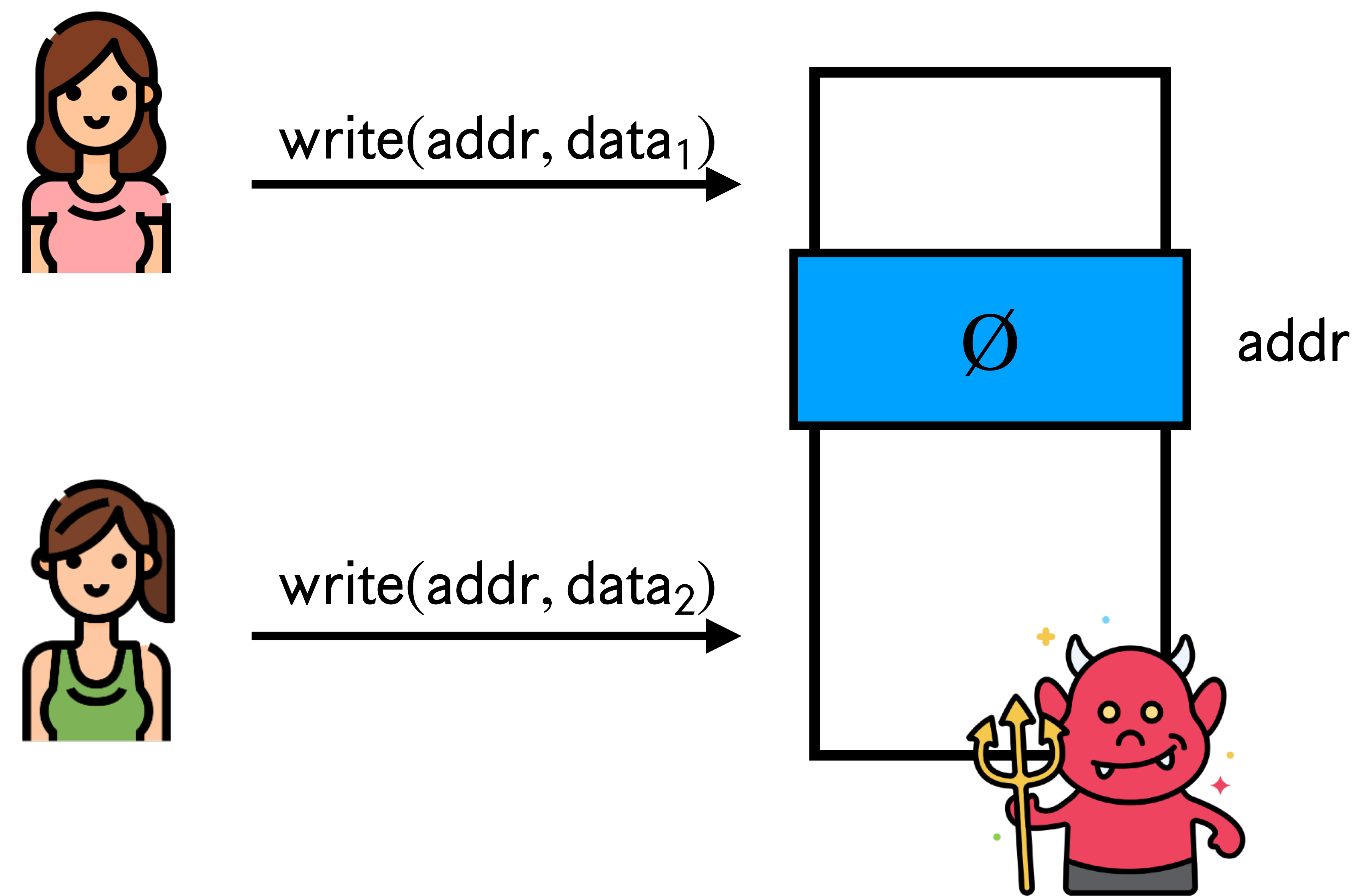


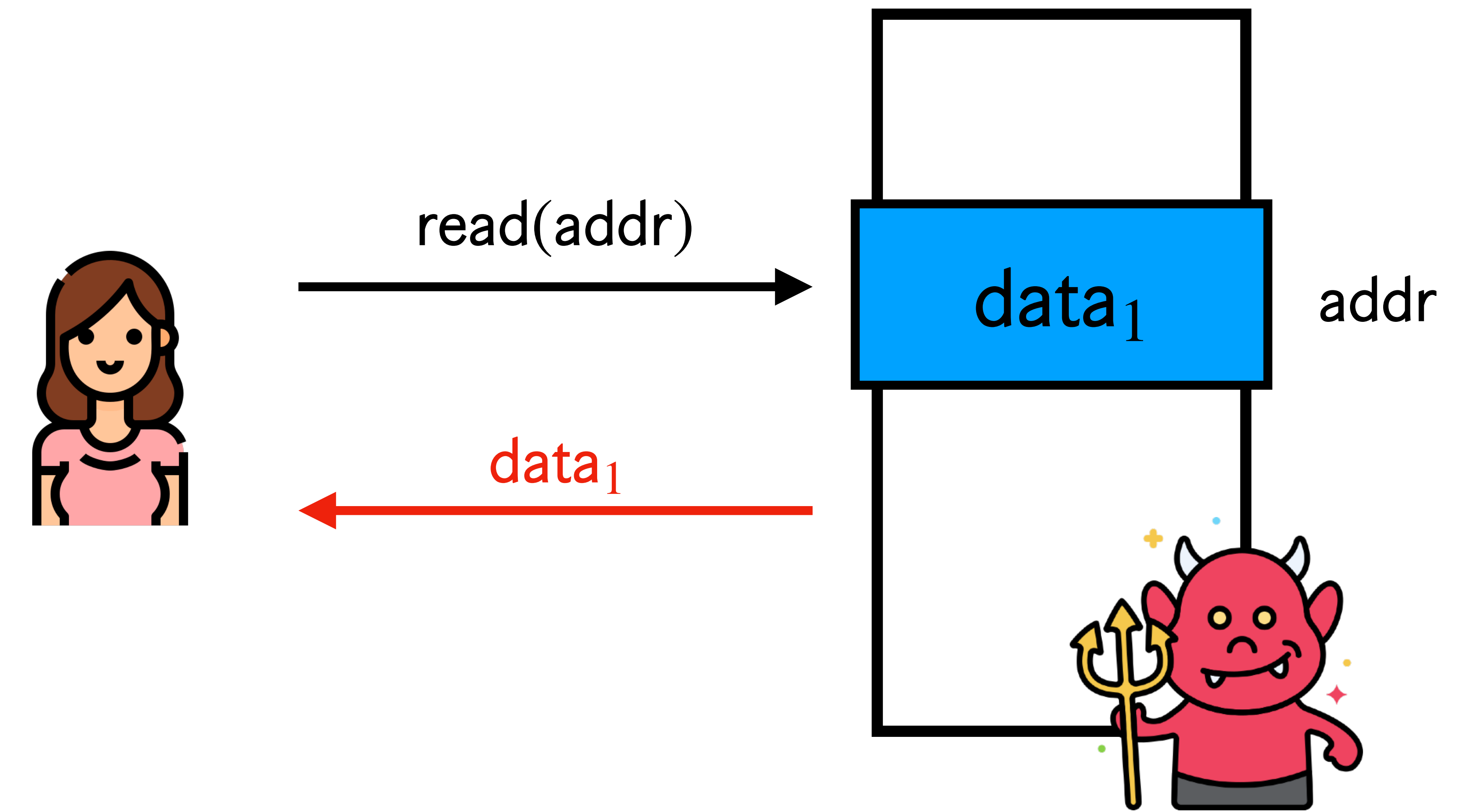


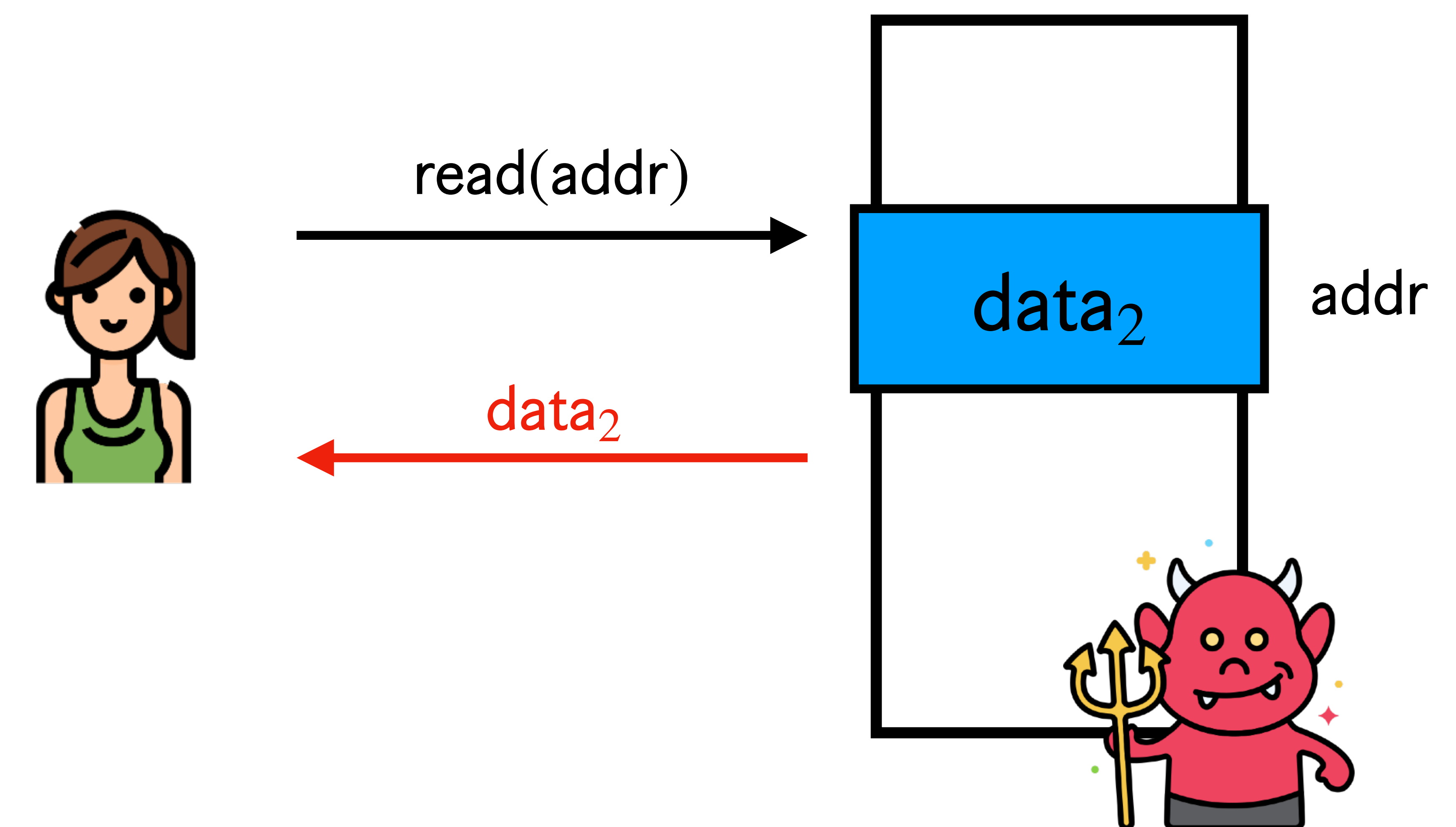


More Modes of Failure!

Consider two concurrent writes:

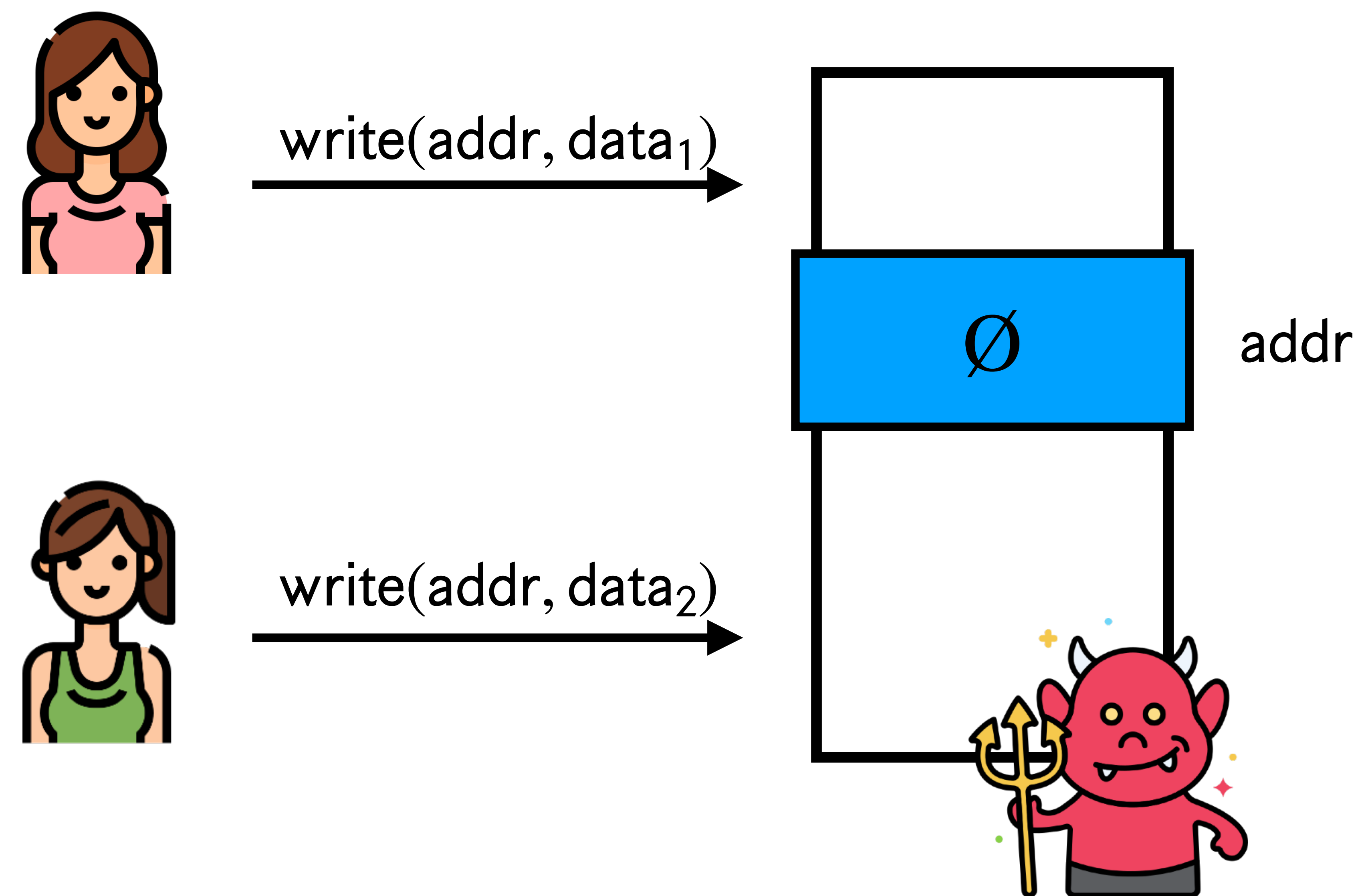


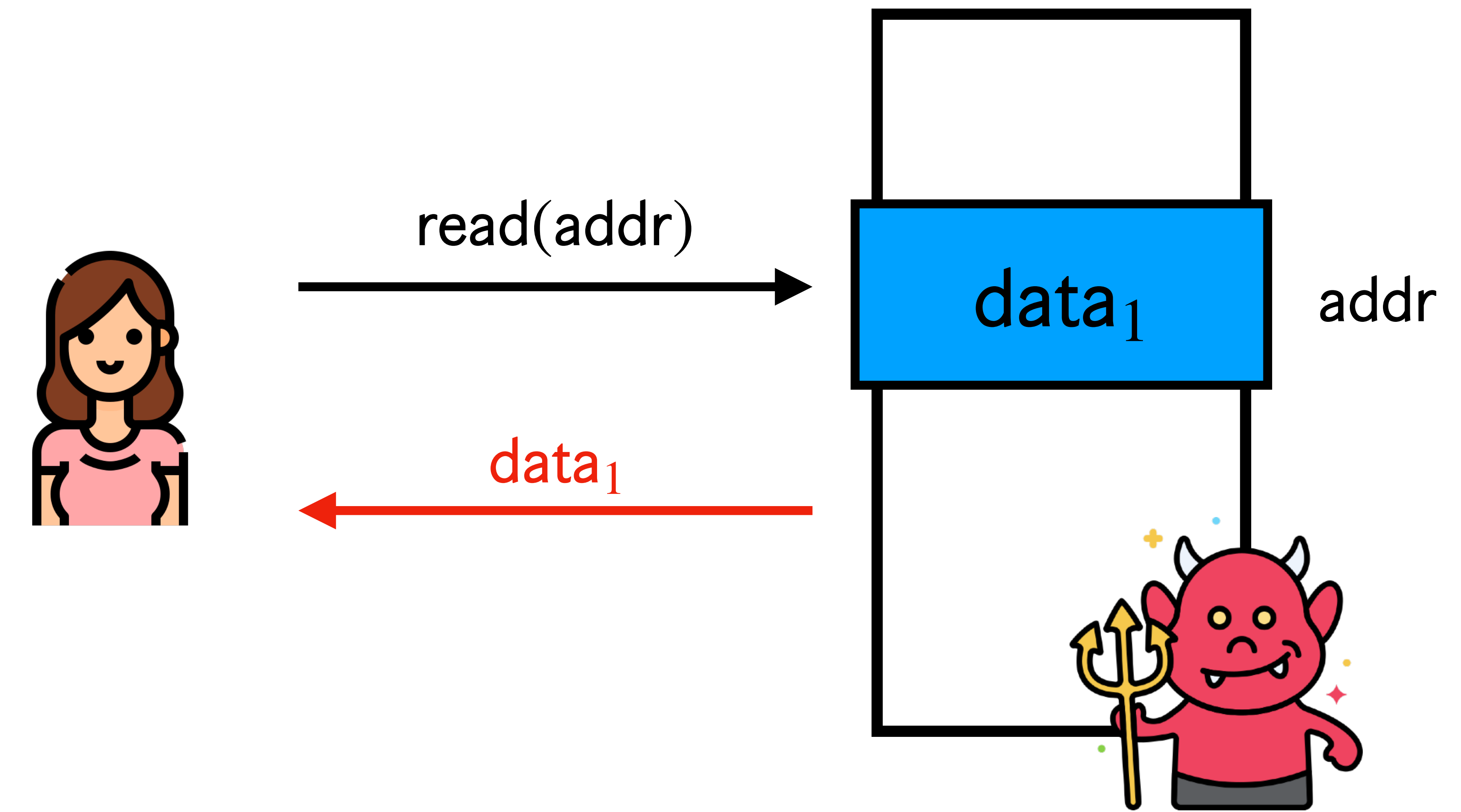


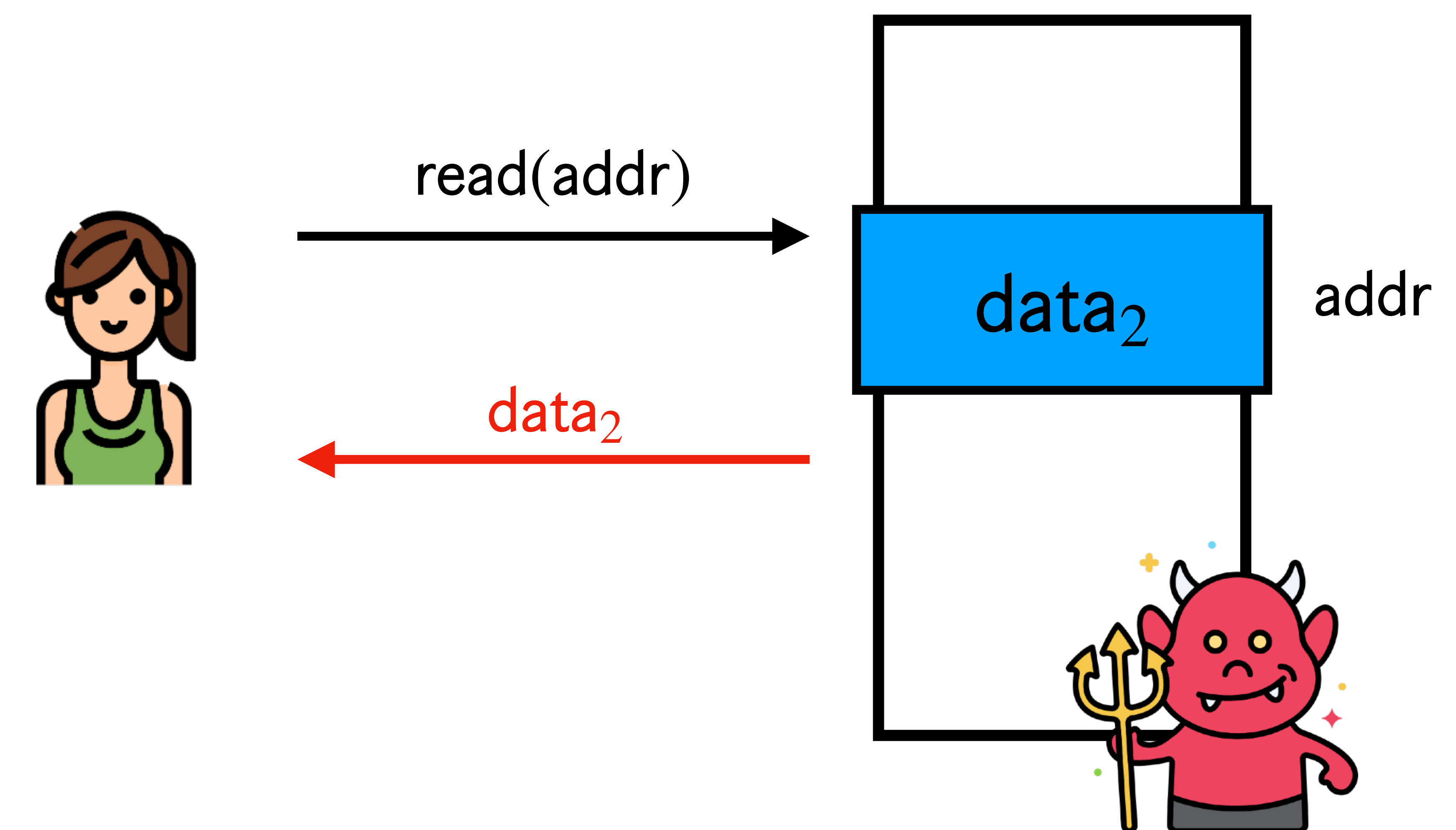


More Modes of Failure!

Consider two concurrent writes:



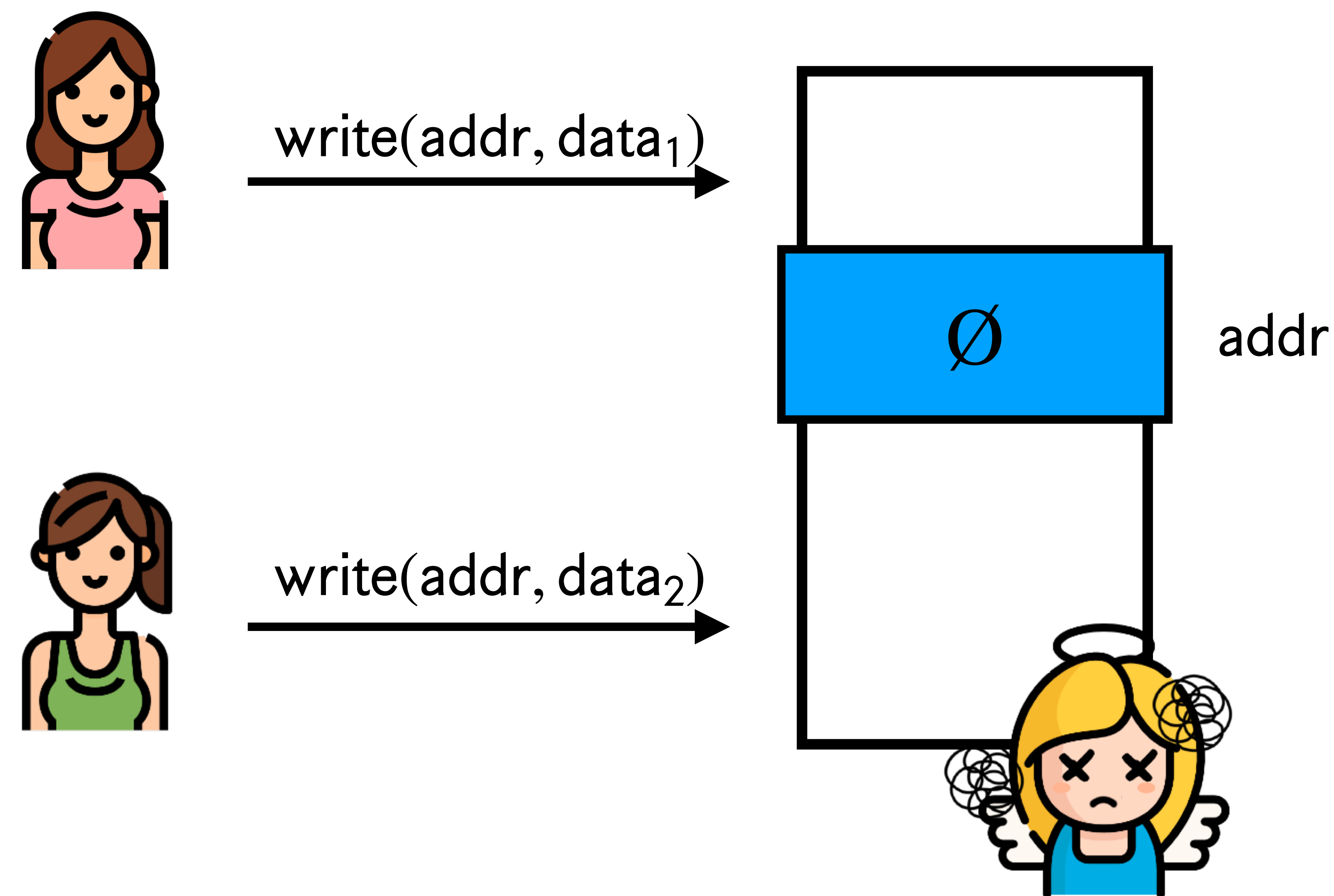


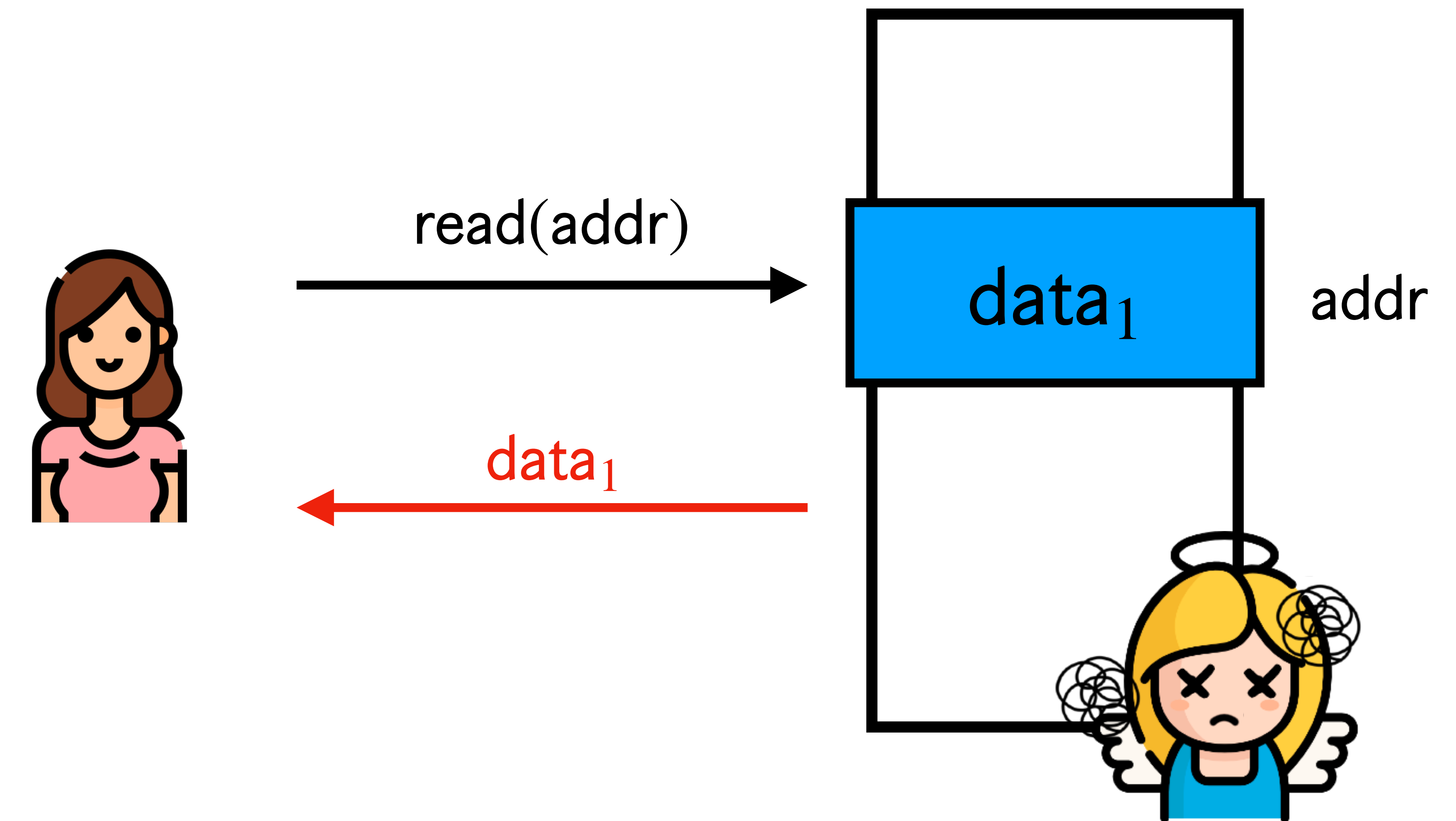


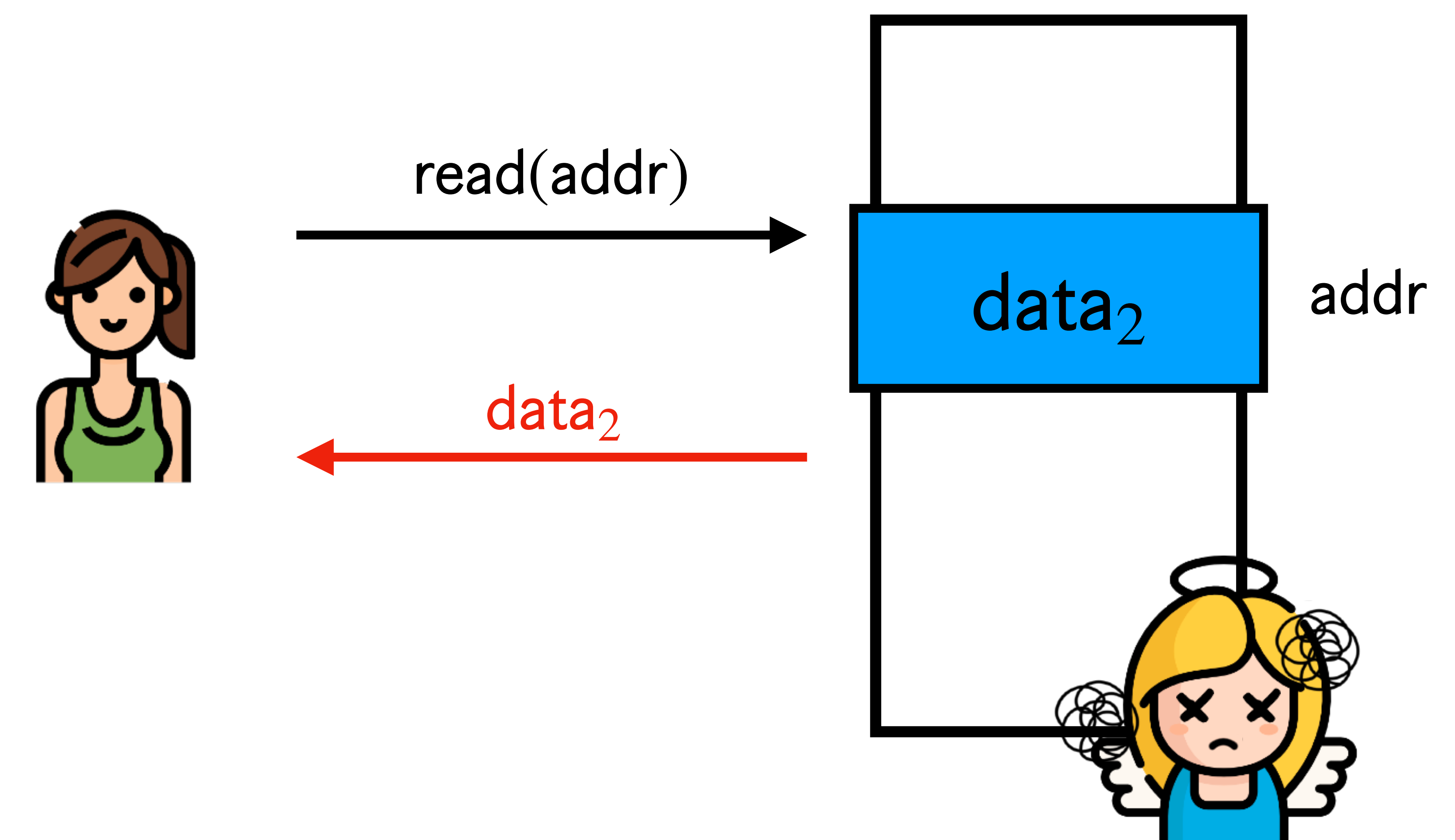
This “branching” might happen even when the server makes an “**honest** mistake”!!

More Modes of Failure!

Consider two concurrent writes:





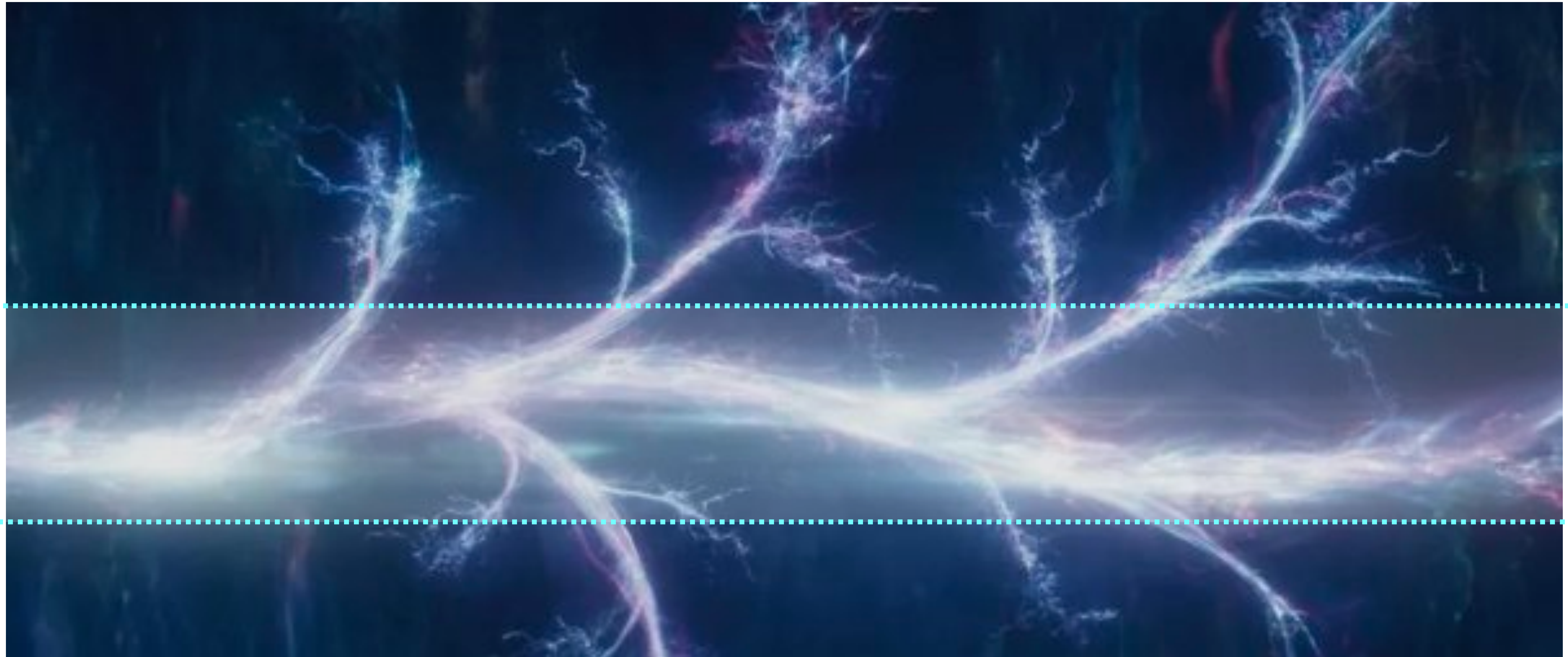


This “branching” might happen even when the server makes an “**honest** mistake”!!

Branching Timelines

From Marvel TV Series: Loki

Branching Timelines



From Marvel TV Series: Loki

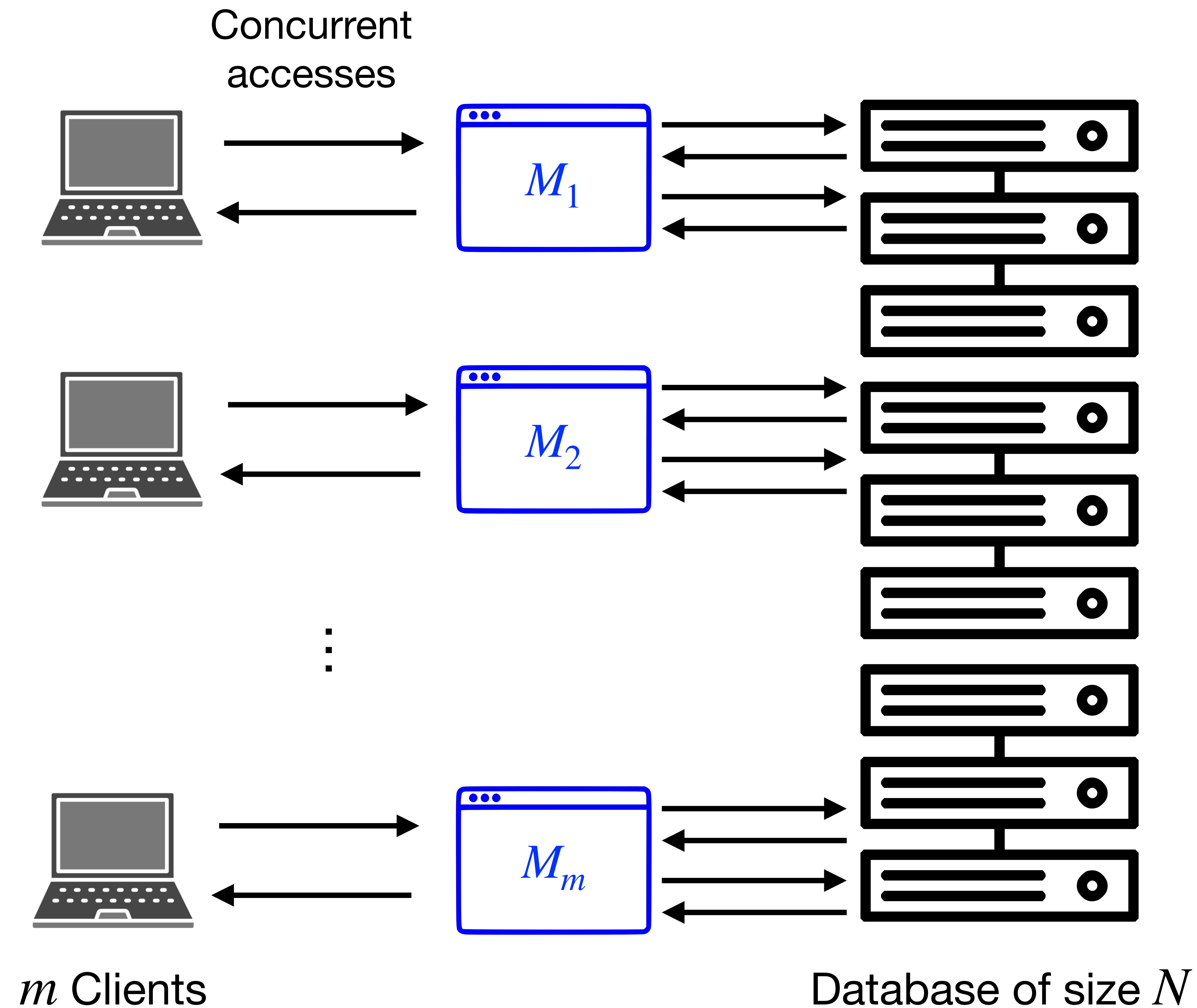
Branching Timelines

From Marvel TV Series: Loki

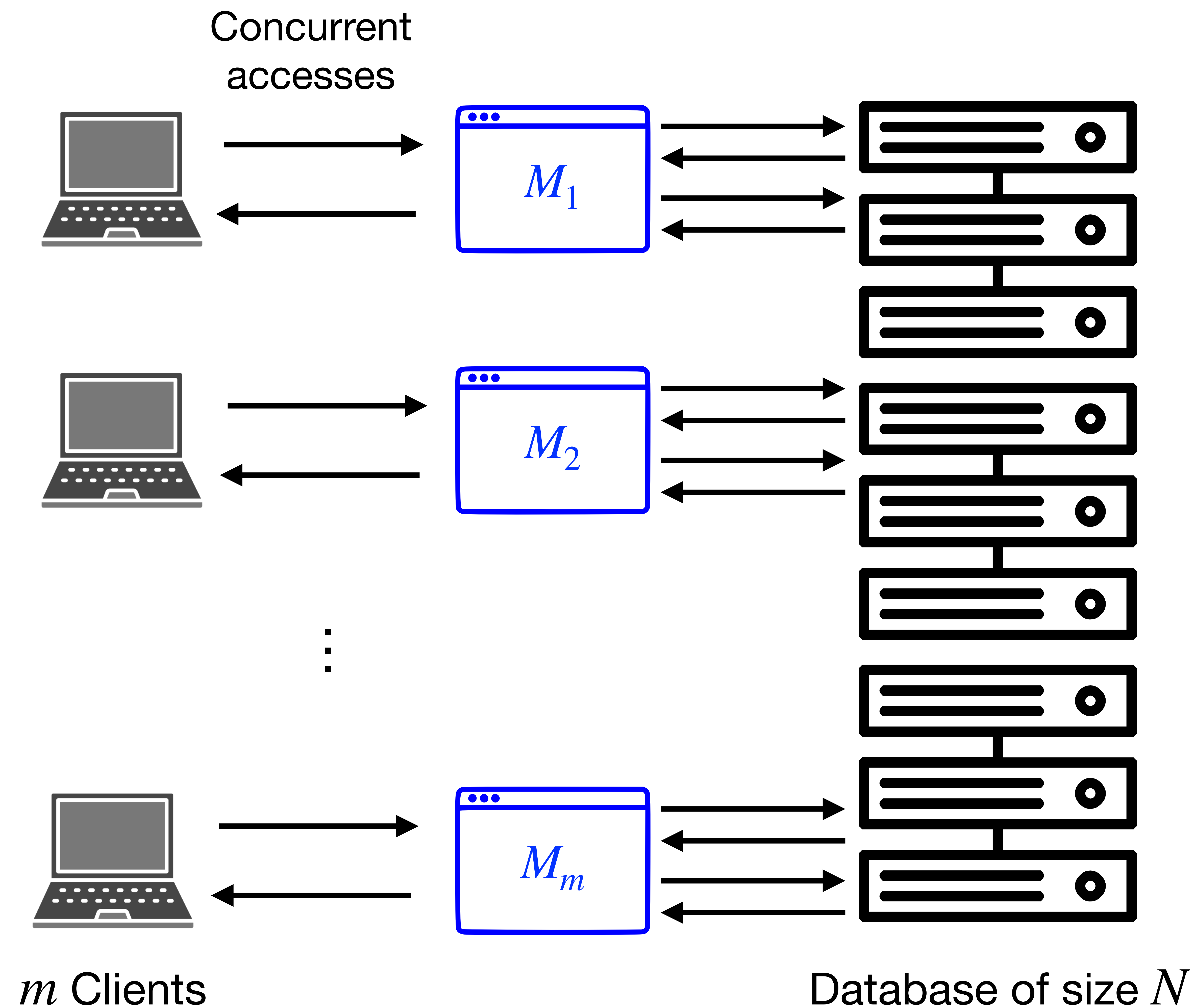
We need to maintain a single consistent version of the database (or *sacred timeline*) across all the clients!

Our Definition

Memory Checking for Parallel RAMs

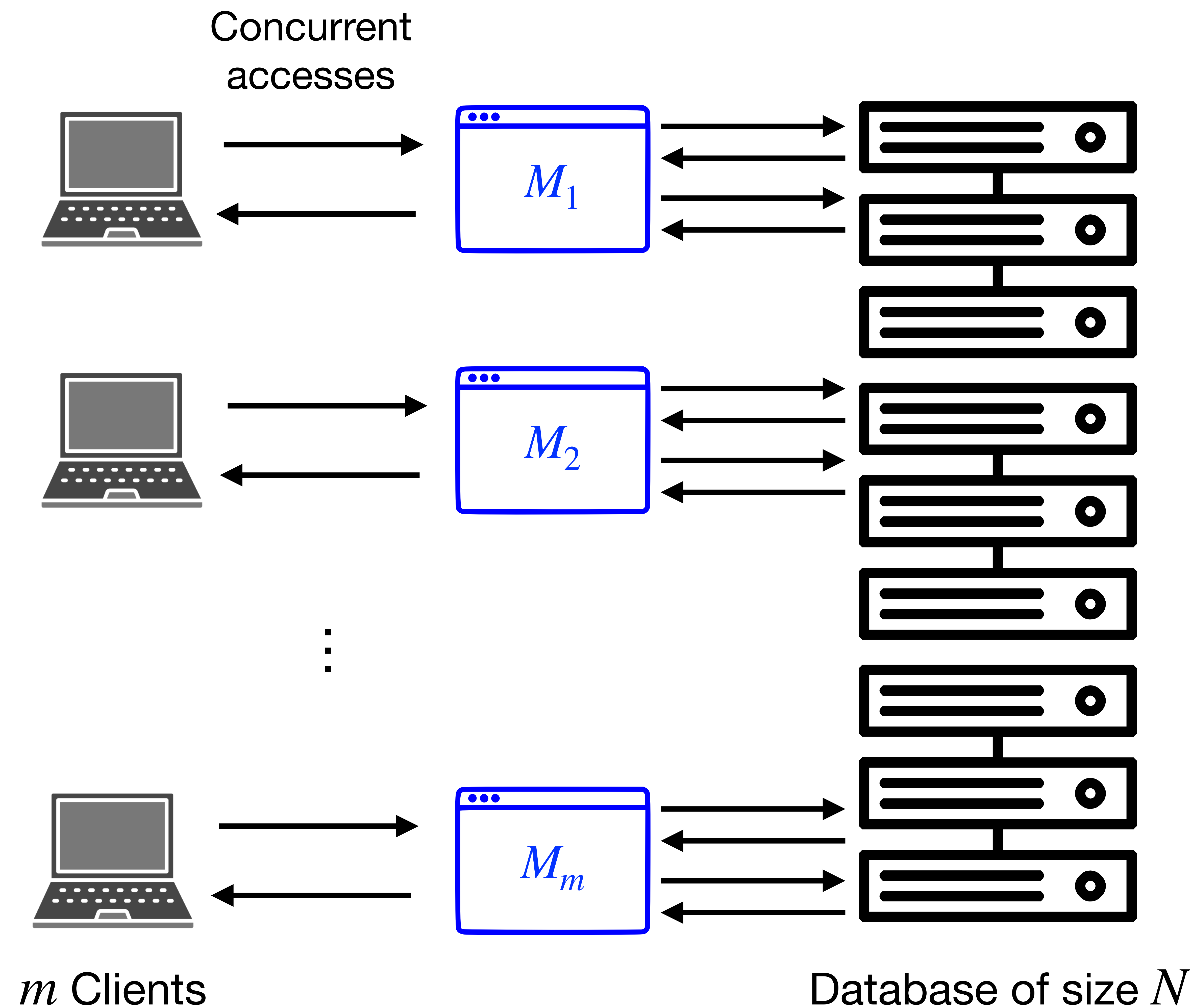


Memory Checking for Parallel RAMs



We now define a notion of **memory checking** for **parallel RAMs.**

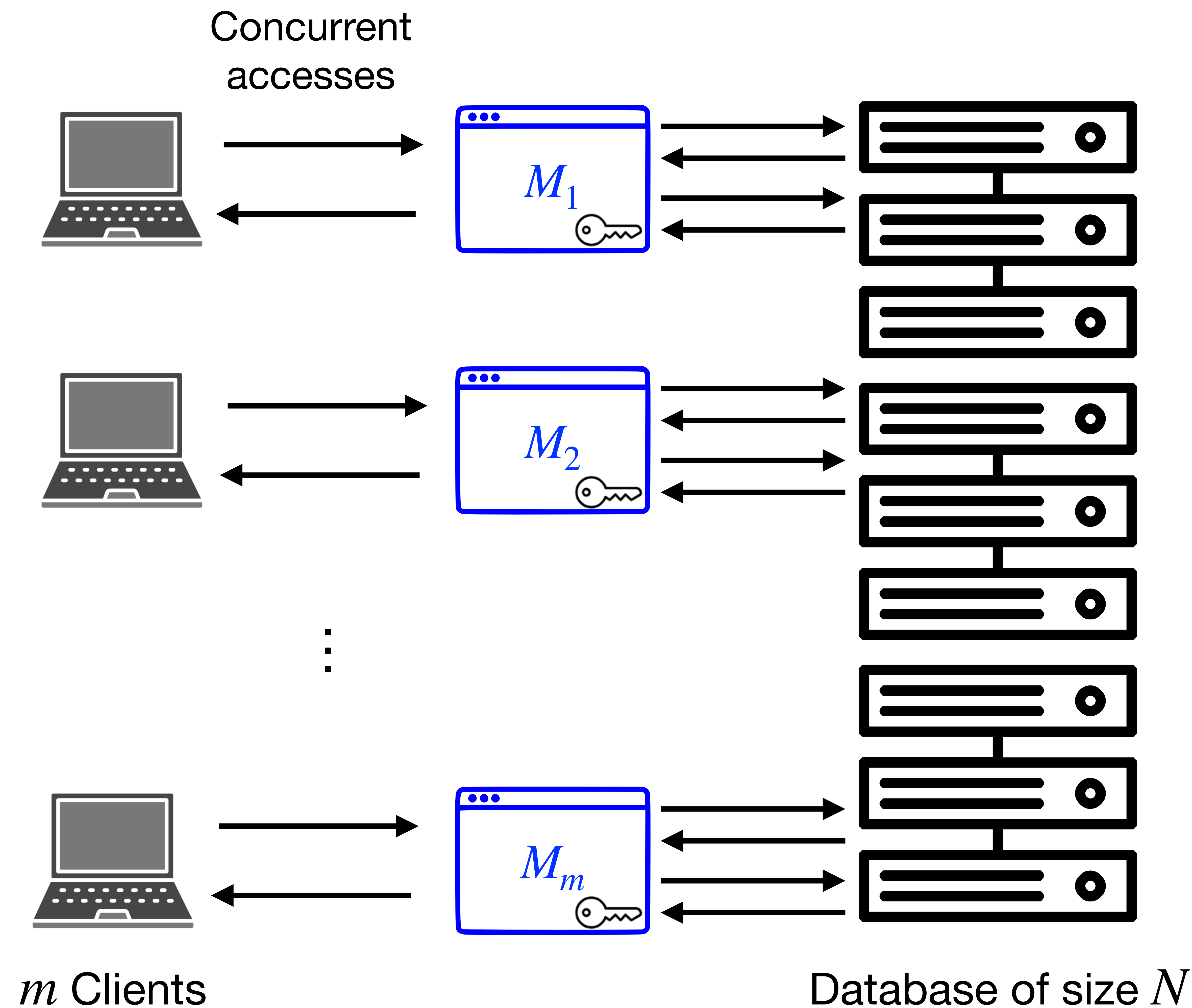
Memory Checking for Parallel RAMs



We now define a notion of **memory checking** for **parallel RAMs.**

Set-up phase:

Memory Checking for Parallel RAMs

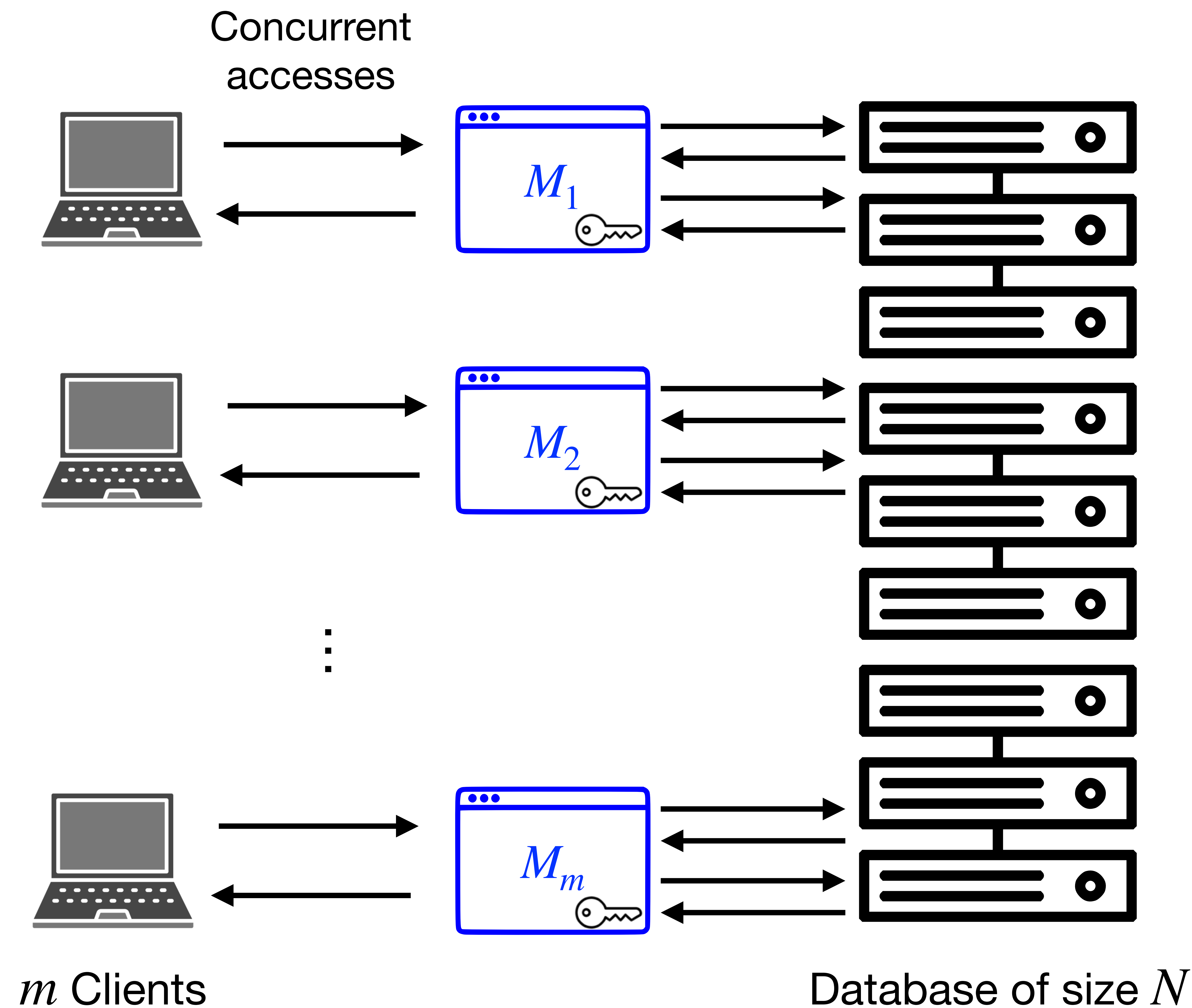


We now define a notion of **memory checking** for **parallel RAMs.**

Set-up phase:

- $\{M_i\}$ initialize their states together.

Memory Checking for Parallel RAMs

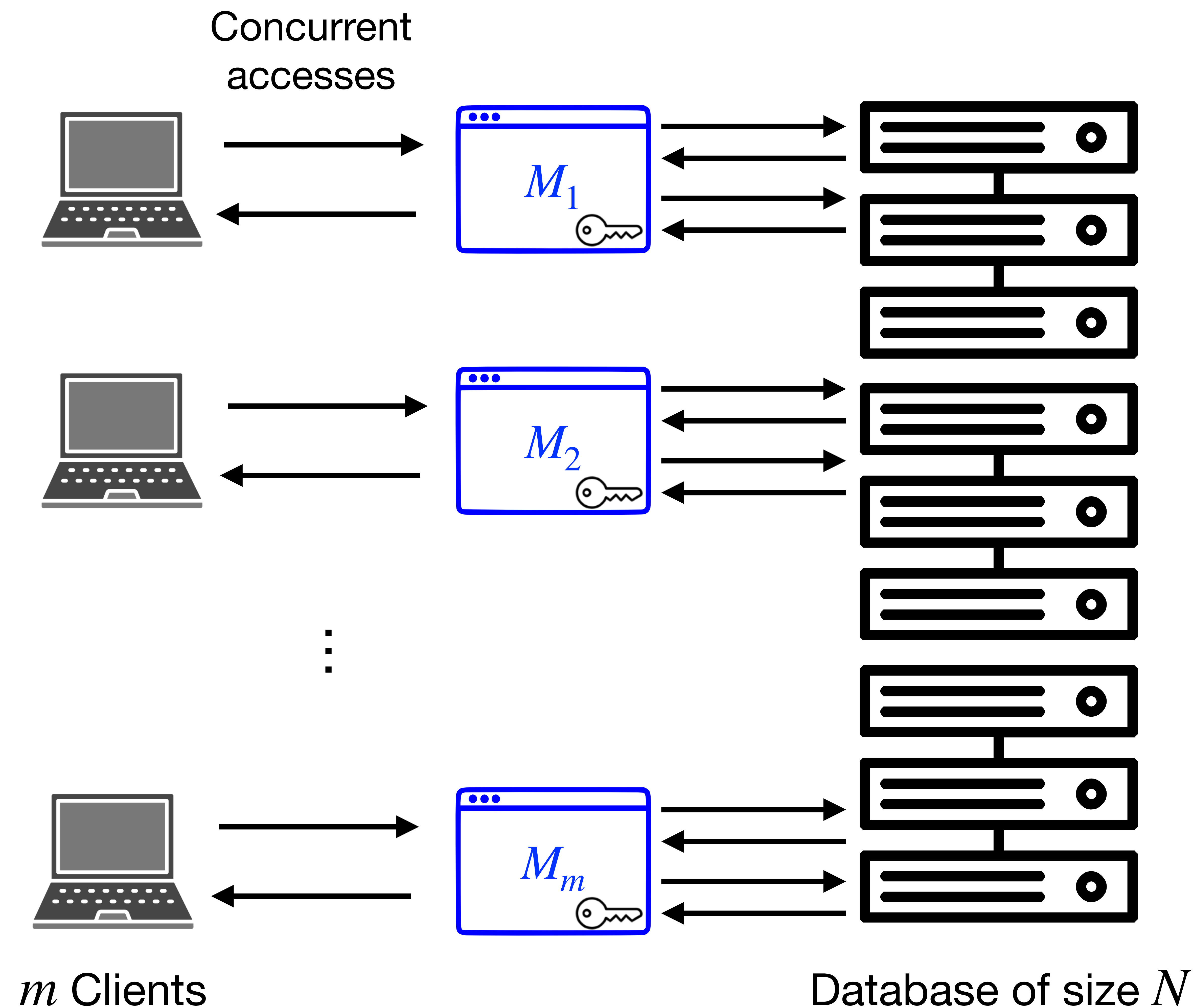


We now define a notion of **memory checking** for **parallel RAMs.**

Set-up phase:

- $\{M_i\}$ initialize their states together.
- No ***direct*** communication after (except through the server).

Memory Checking for Parallel RAMs



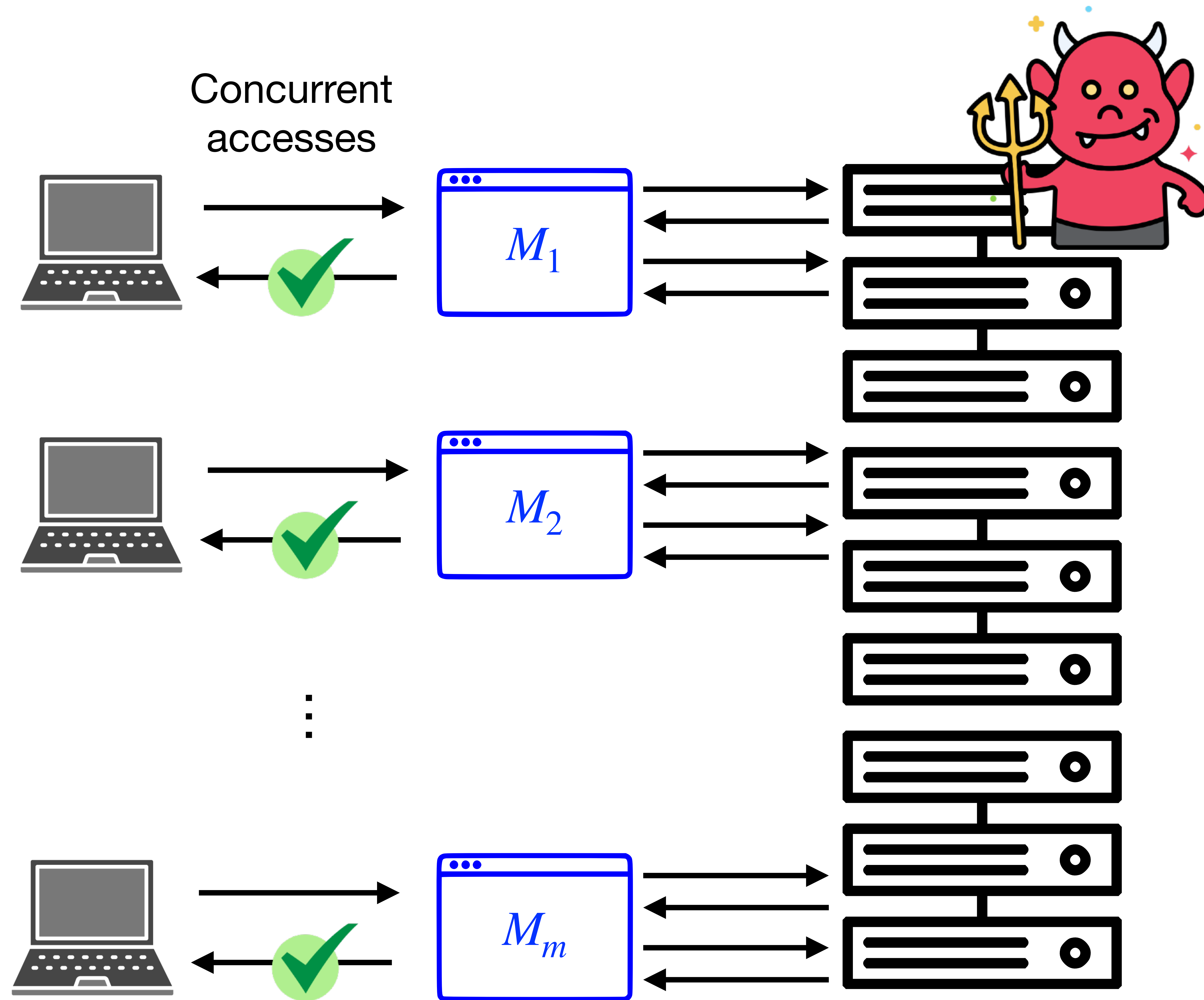
We now define a notion of **memory checking** for **parallel RAMs.**

Set-up phase:

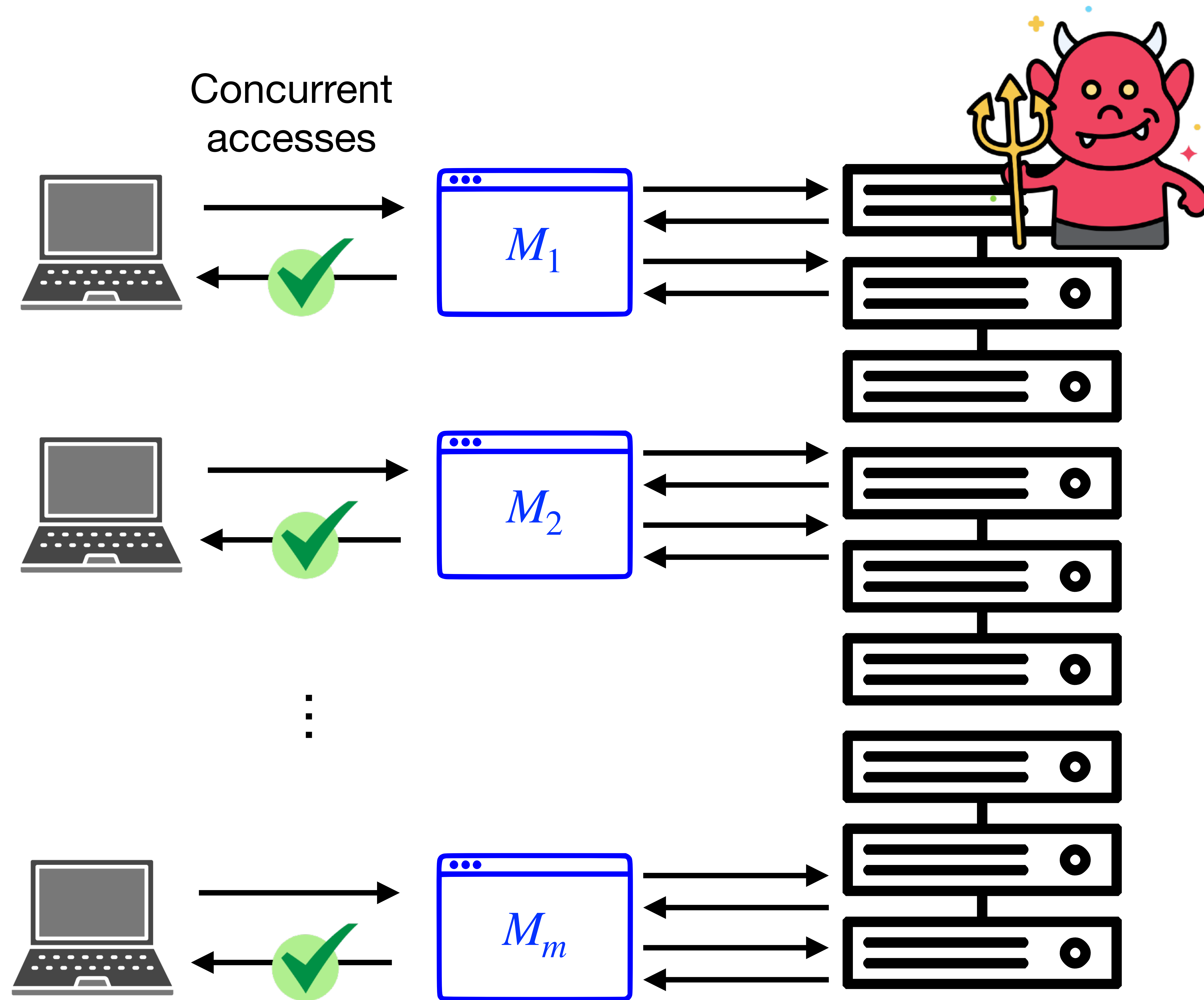
- $\{M_i\}$ initialize their states together.
- No ***direct*** communication after (except through the server).

Note: It is possible that the clients have secure channels, but we want to make no assumptions.

Memory Checking for Parallel RAMs

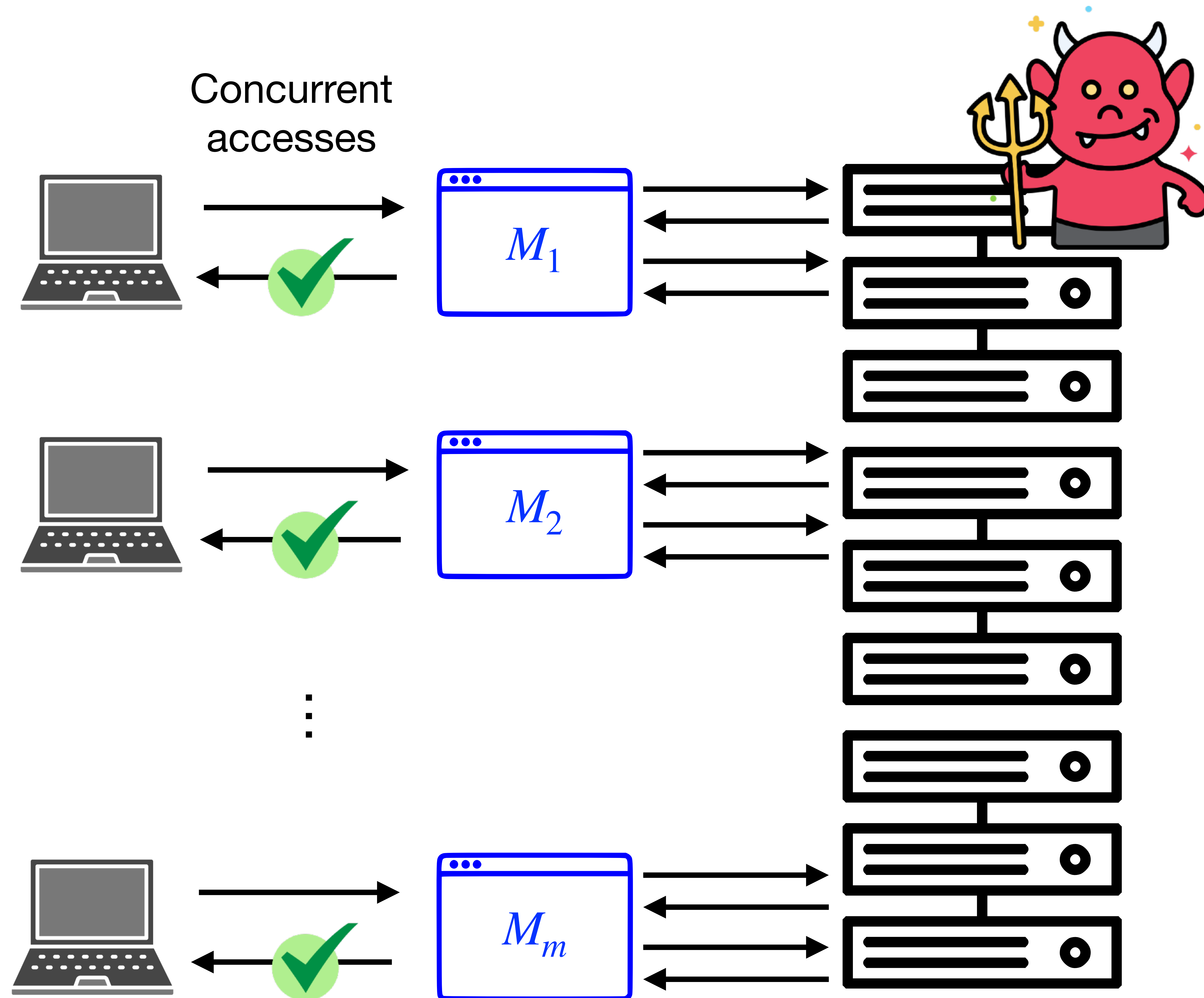


Memory Checking for Parallel RAMs



- **Correctness:** All $\{M_i\}_i$ send back correct responses, or some M_j aborts.

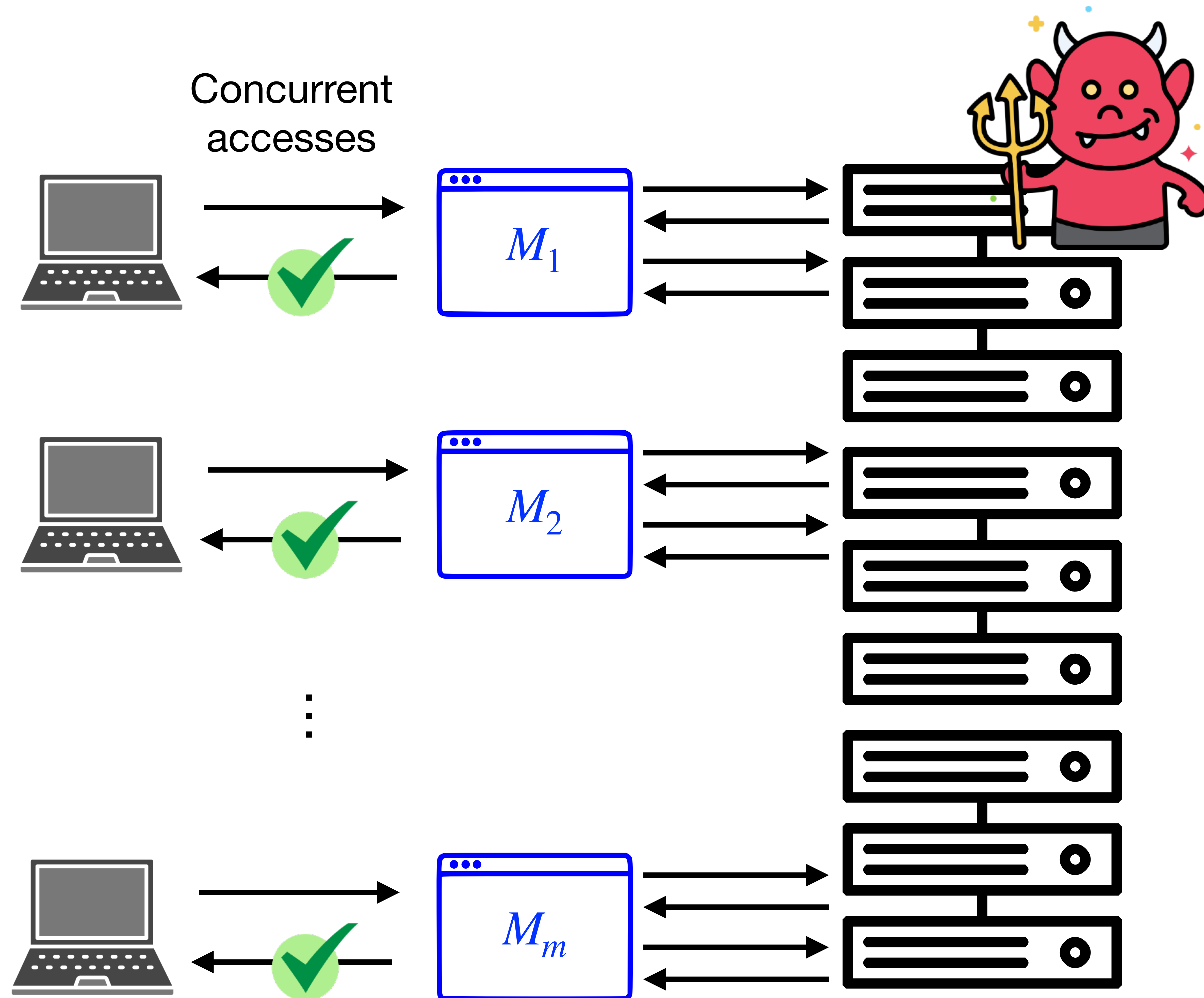
Memory Checking for Parallel RAMs



- **Correctness:** All $\{M_i\}_i$ send back correct responses, or some M_j aborts.

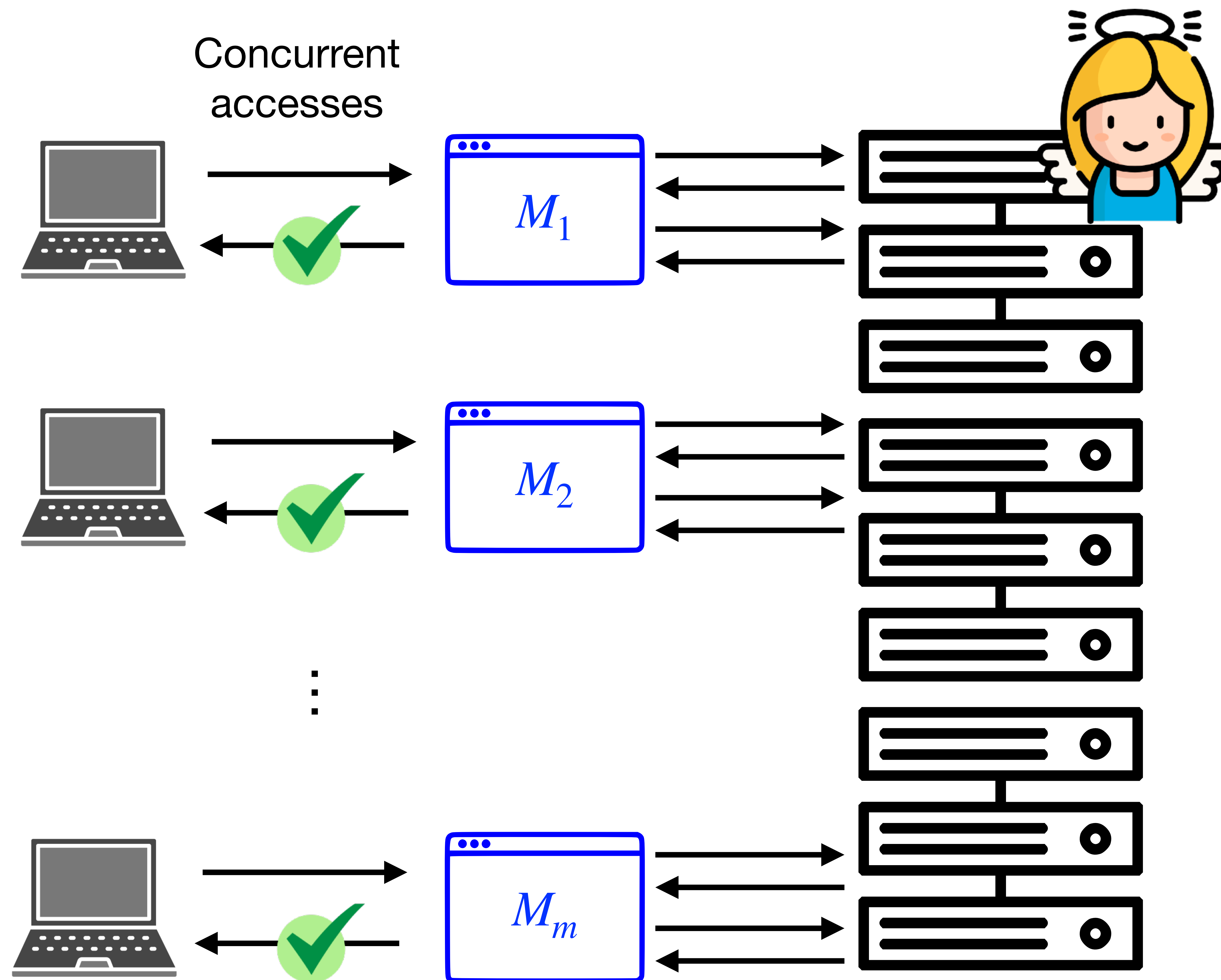
  - All reads are correct

Memory Checking for Parallel RAMs



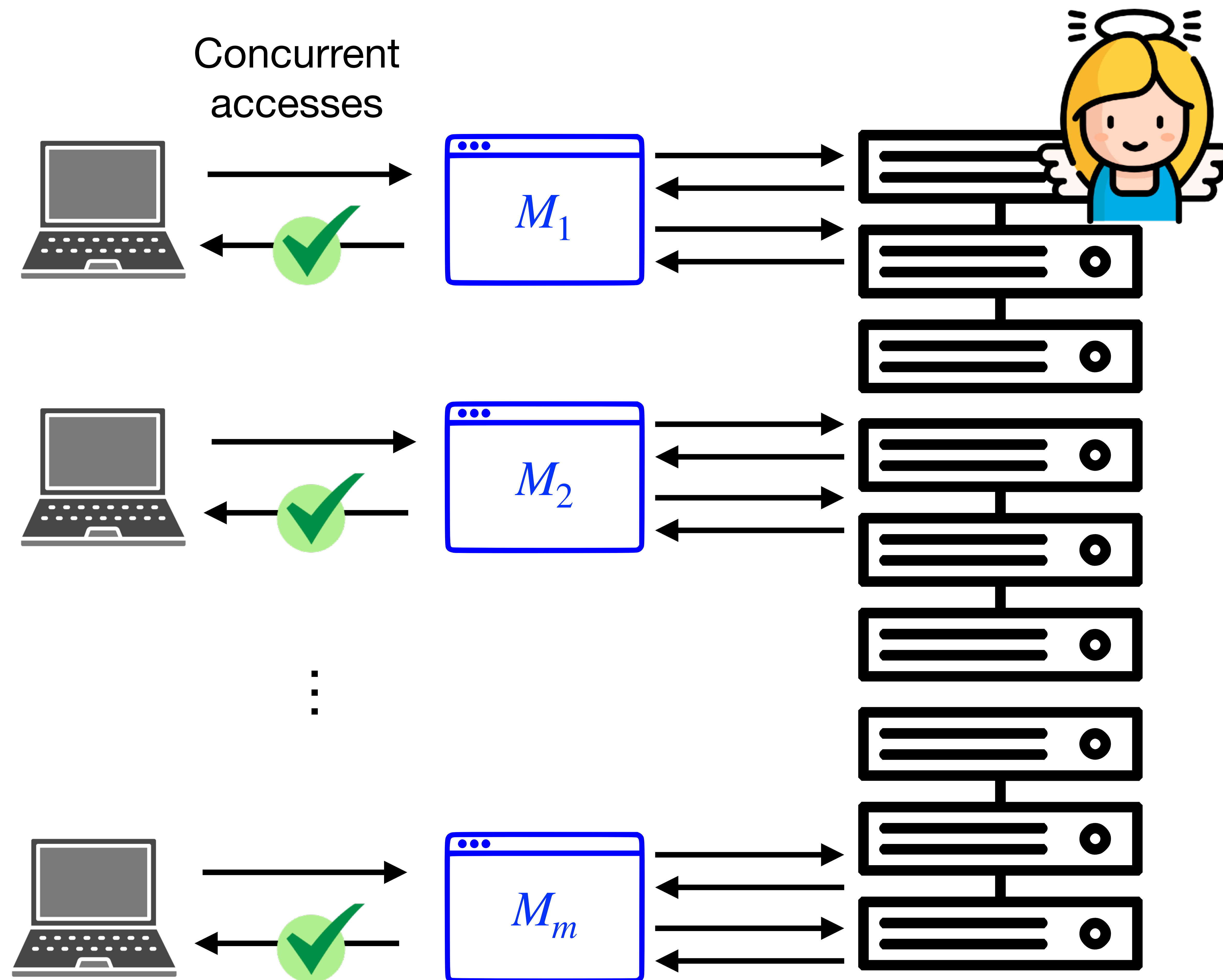
- **Correctness:** All $\{M_i\}_i$ send back correct responses, or some M_j aborts.
  - All reads are correct
  - Concurrent writes are tie-broken (arbitrarily chosen by server)

Memory Checking for Parallel RAMs



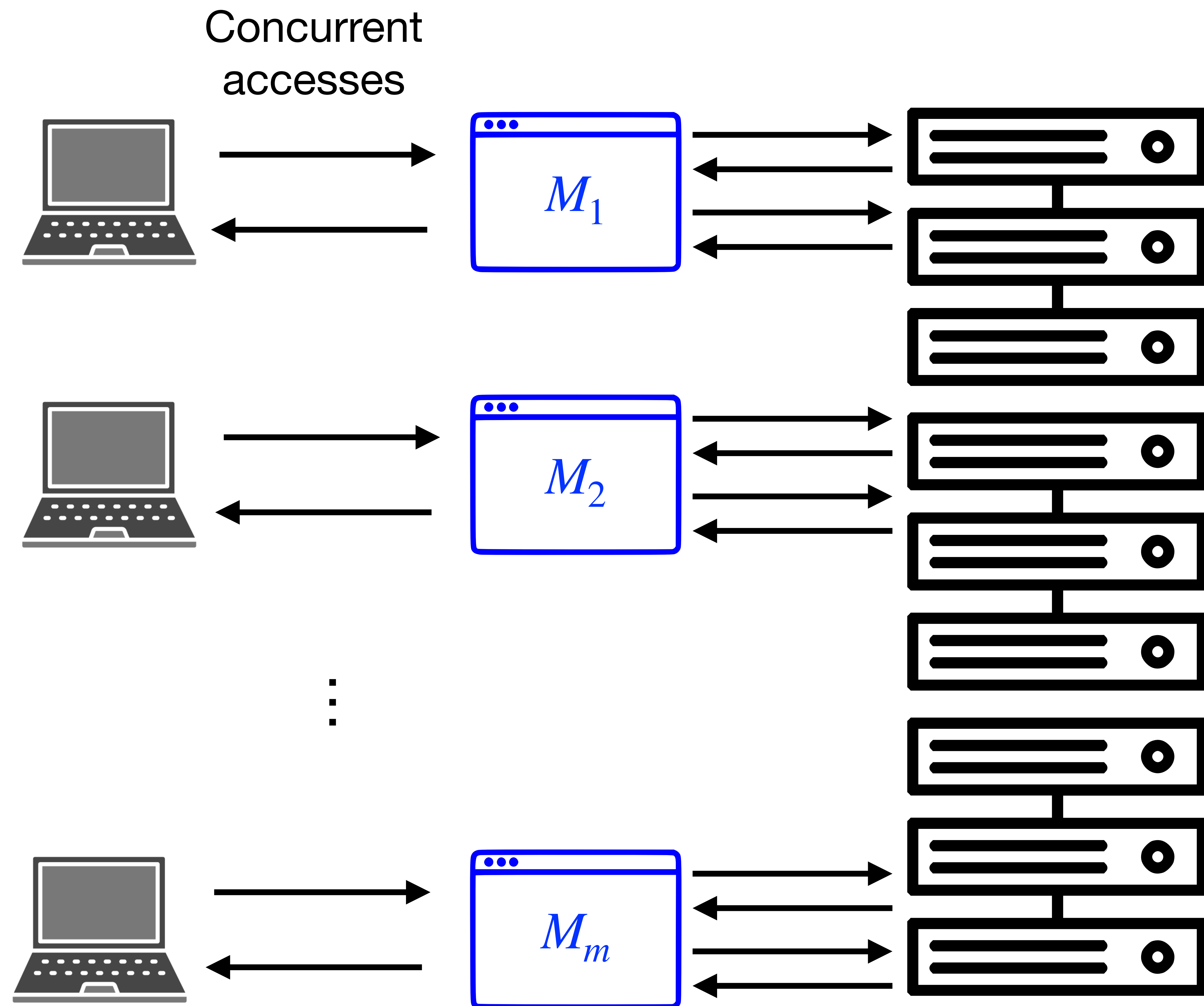
- **Correctness:** All $\{M_i\}_i$ send back correct responses, or some M_j aborts.
  - All reads are correct
  - Concurrent writes are tie-broken (arbitrarily chosen by server)

Memory Checking for Parallel RAMs

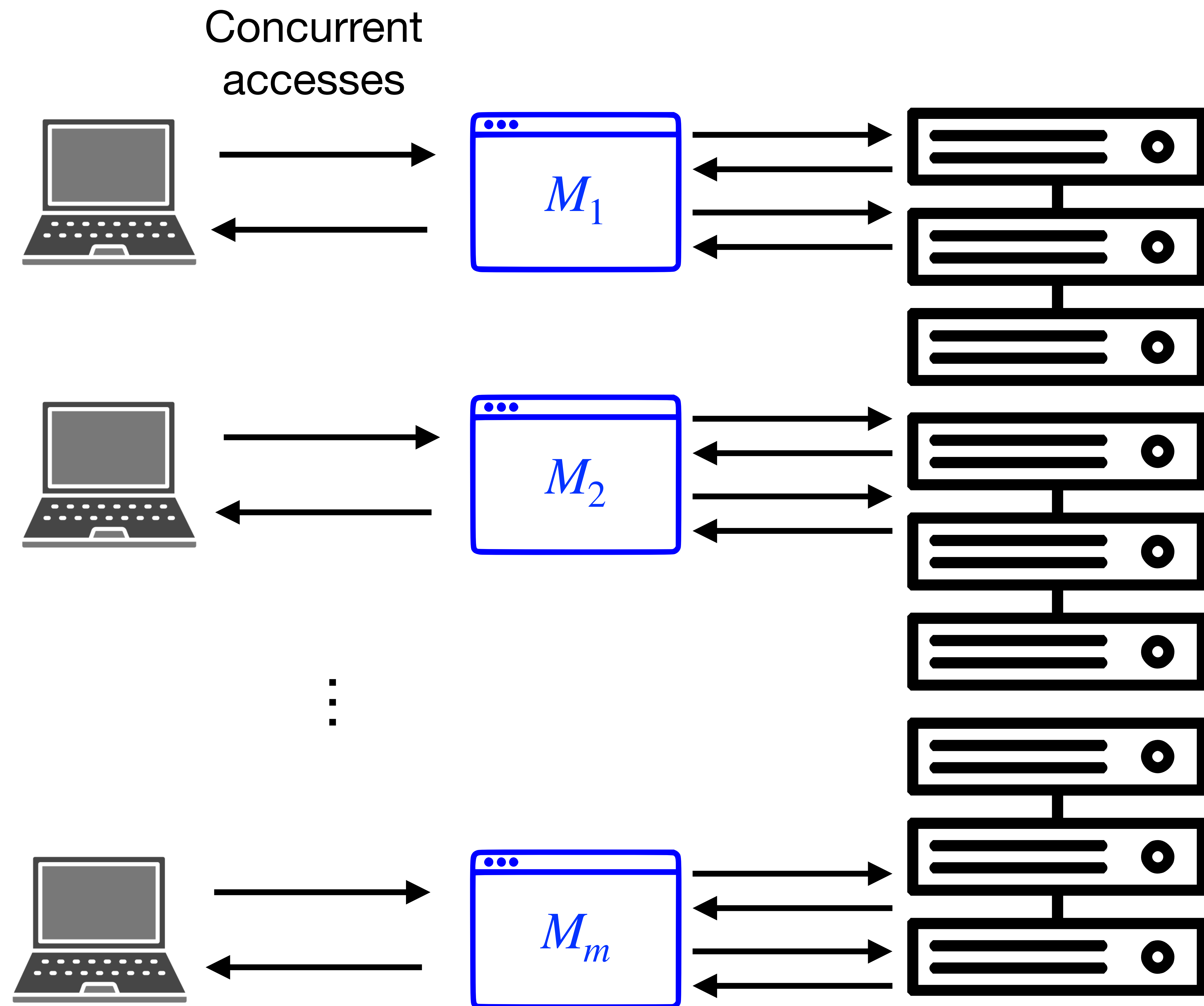


- **Correctness:** All $\{M_i\}_i$ send back correct responses, or some M_j aborts.
  - All reads are correct
  - Concurrent writes are tie-broken (arbitrarily chosen by server)
- **Completeness:** No M_j aborts if server is not malicious.

Memory Checking for Parallel RAMs

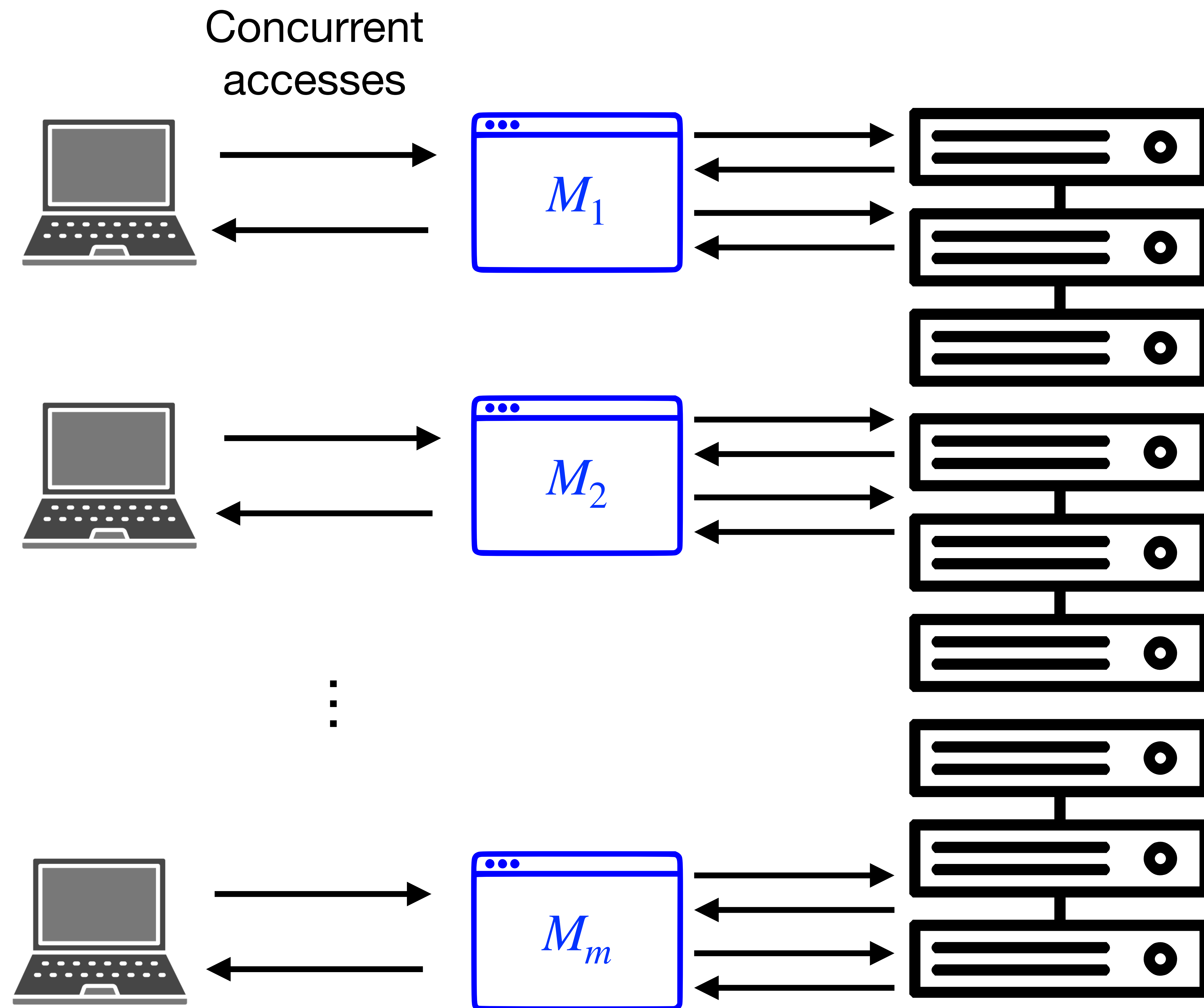


Memory Checking for Parallel RAMs



Efficiency metrics

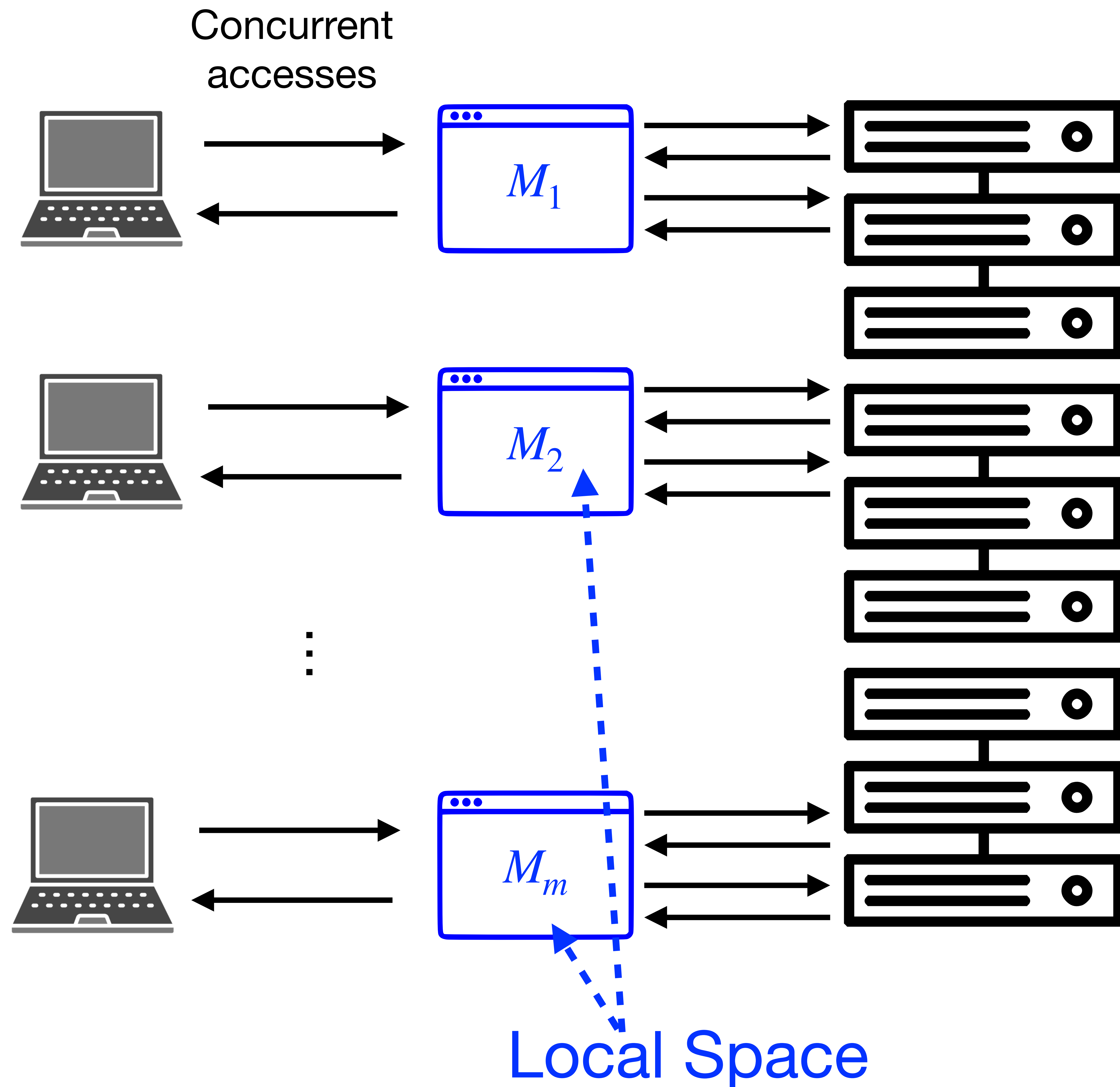
Memory Checking for Parallel RAMs



Efficiency metrics

- **Local Space:** Space per checker. This talk: $O(1)$ words/$O(\lambda)$ bits.

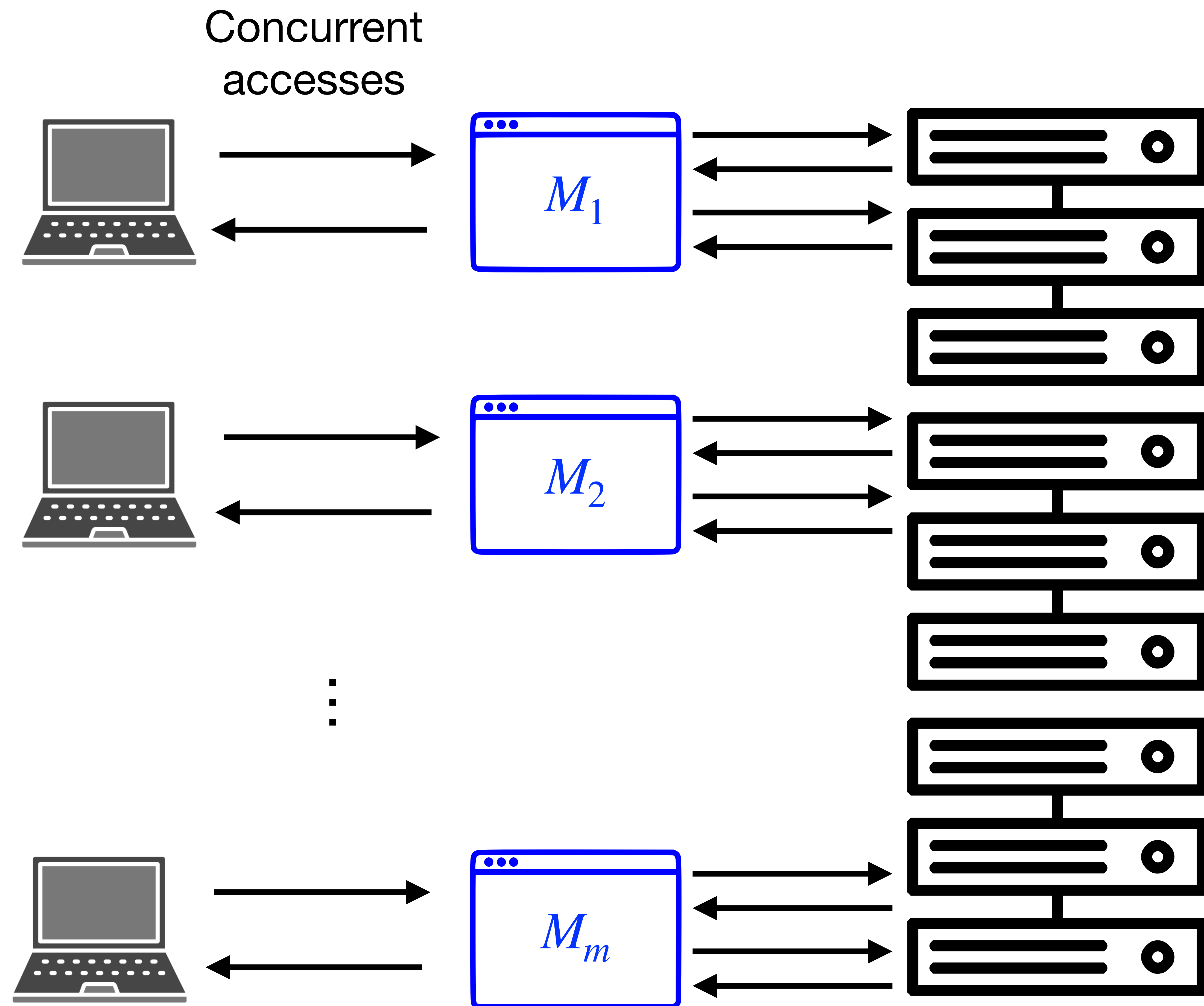
Memory Checking for Parallel RAMs



Efficiency metrics

- **Local Space:** Space per checker. This talk: $O(1)$ words/$O(\lambda)$ bits.

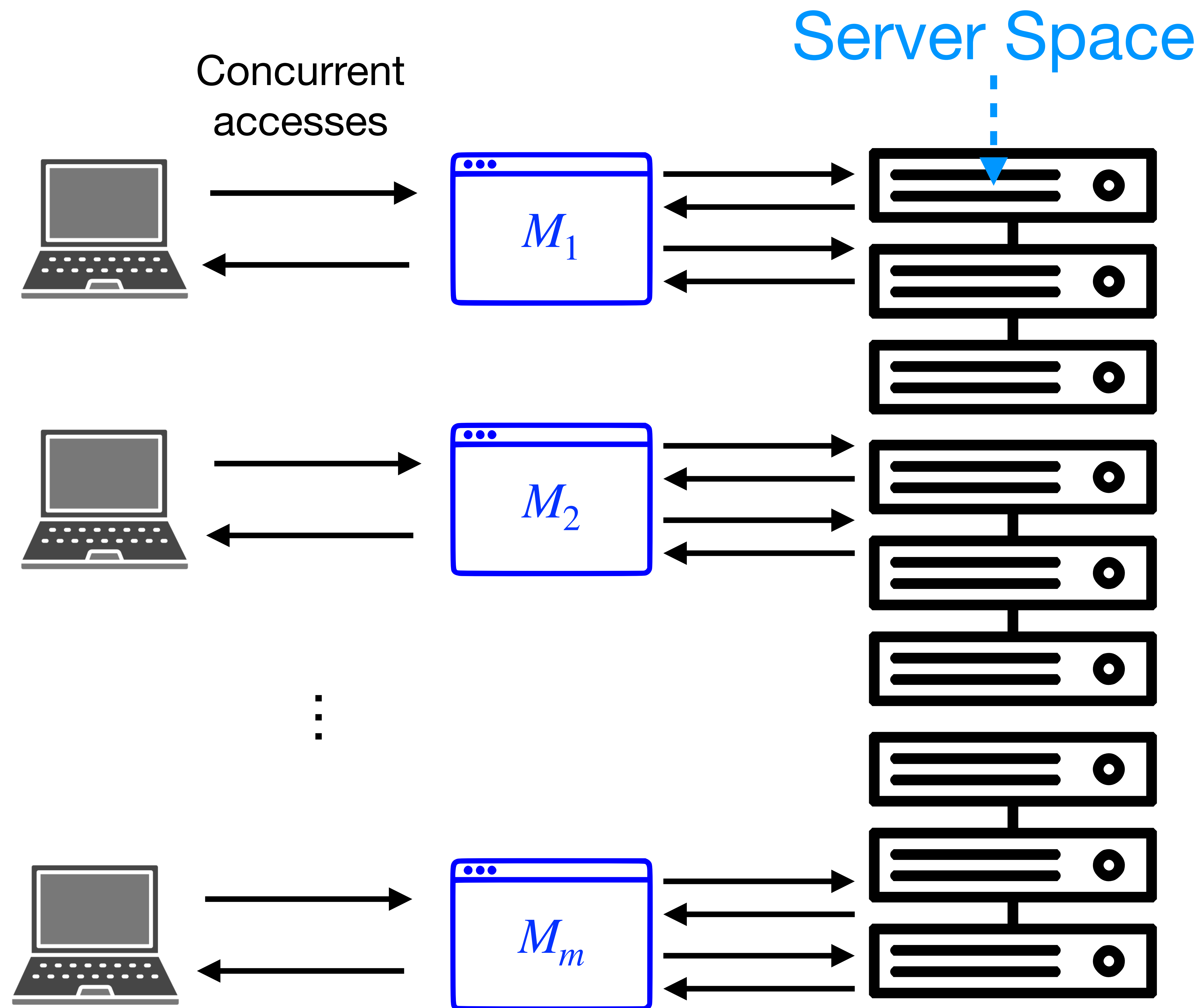
Memory Checking for Parallel RAMs



Efficiency metrics

- **Local Space:** Space per checker. This talk: $O(1)$ words/$O(\lambda)$ bits.
- **Server Space:** Server storage size. This talk: $O(N)$ words.

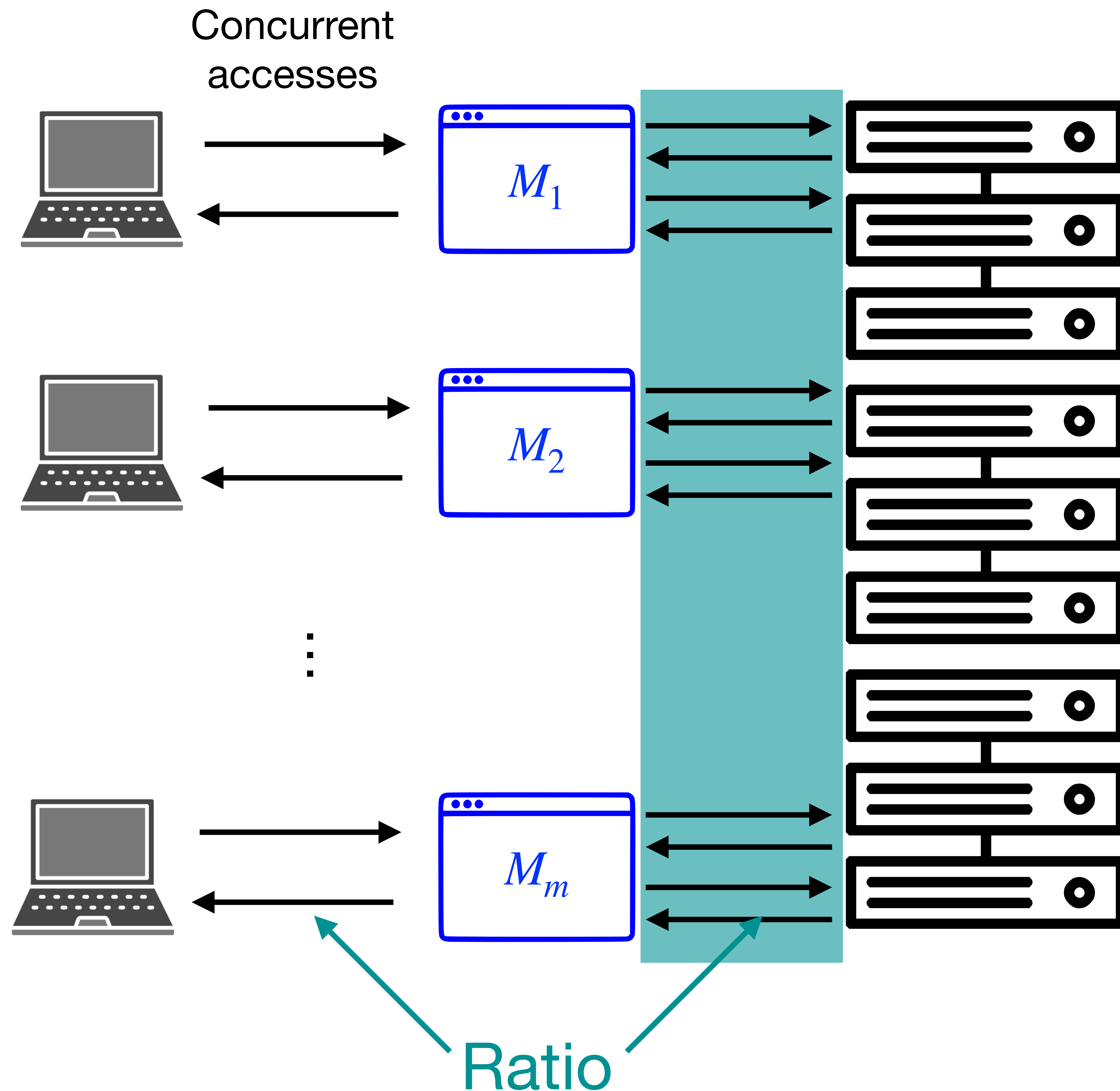
Memory Checking for Parallel RAMs



Efficiency metrics

- **Local Space:** Space per checker. This talk: $O(1)$ words/$O(\lambda)$ bits.
- **Server Space:** Server storage size. This talk: $O(N)$ words.

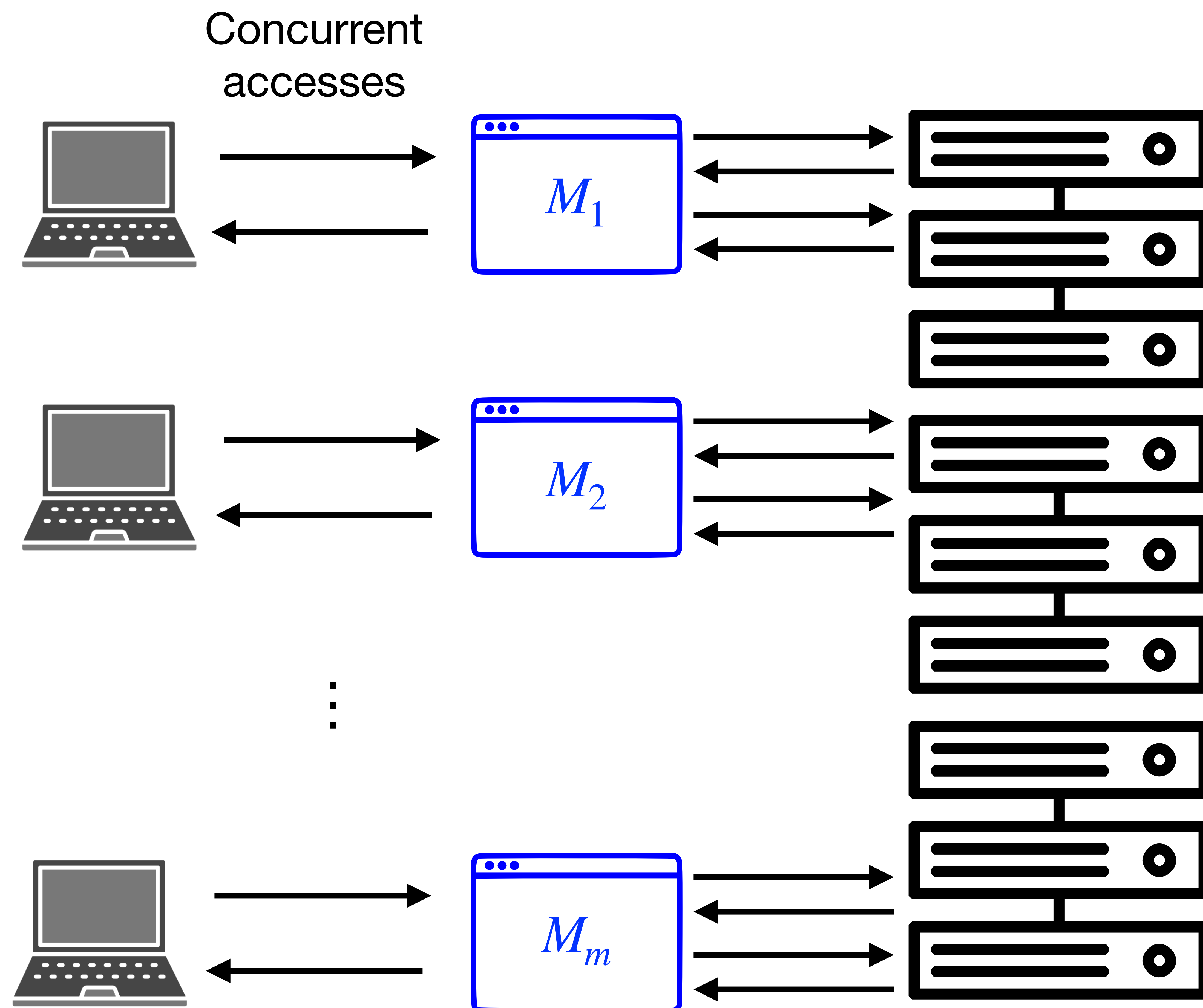
Memory Checking for Parallel RAMs



Efficiency metrics

- **Local Space:** Space per checker. This talk: $O(1)$ words/$O(\lambda)$ bits.
- **Server Space:** Server storage size. This talk: $O(N)$ words.
- **Work blowup:** Ratio of server accesses per underlying PRAM access.

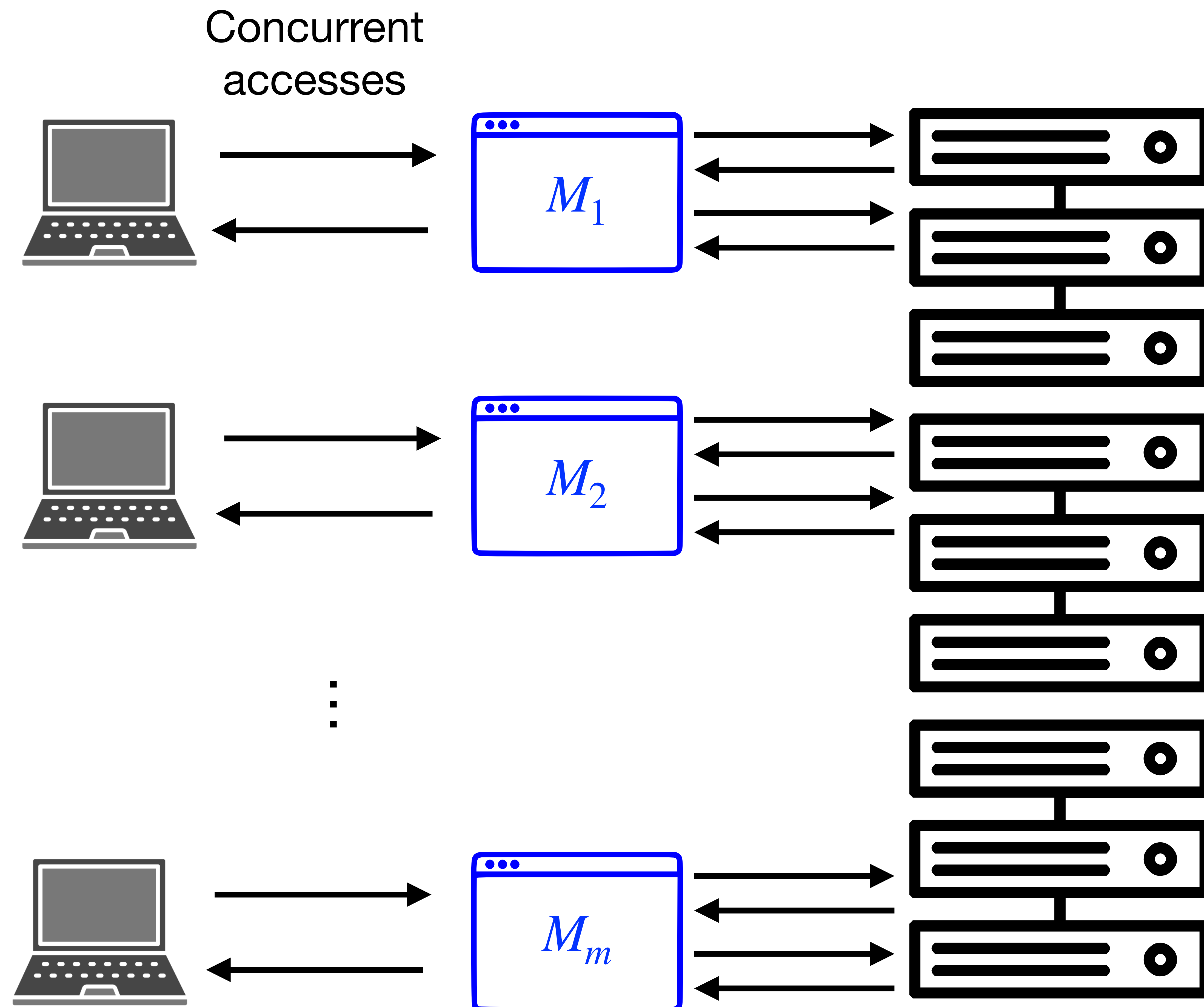
Memory Checking for Parallel RAMs



Efficiency metrics

- **Local Space:** Space per checker. This talk: $O(1)$ words/$O(\lambda)$ bits.
- **Server Space:** Server storage size. This talk: $O(N)$ words.
- **Work blowup:** Ratio of server accesses per underlying PRAM access.

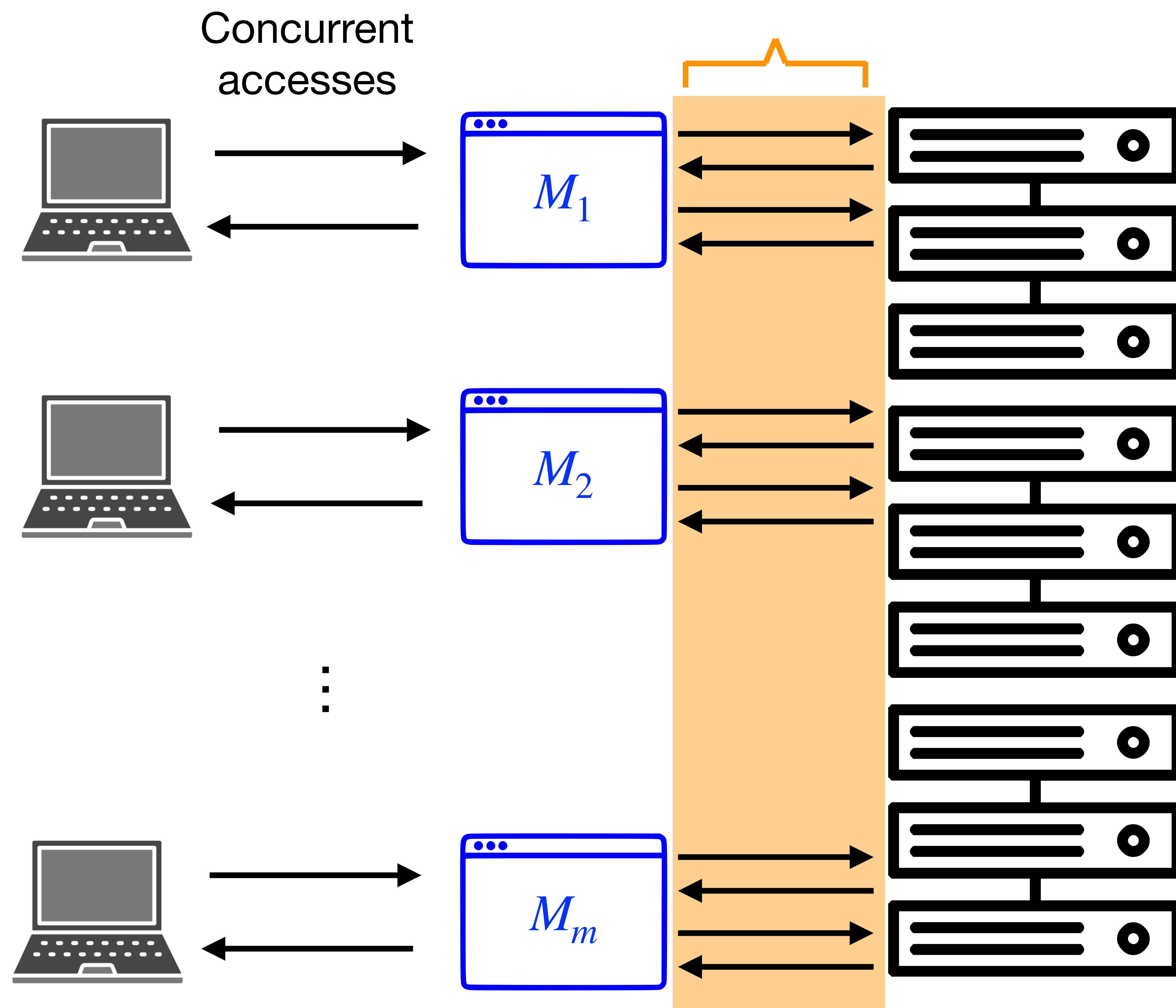
Memory Checking for Parallel RAMs



Efficiency metrics

- **Local Space:** Space per checker. This talk: $O(1)$ words/$O(\lambda)$ bits.
- **Server Space:** Server storage size. This talk: $O(N)$ words.
- **Work blowup:** Ratio of server accesses per underlying PRAM access.
- **Depth blowup:** Number of parallel steps to support a single batch of requests.

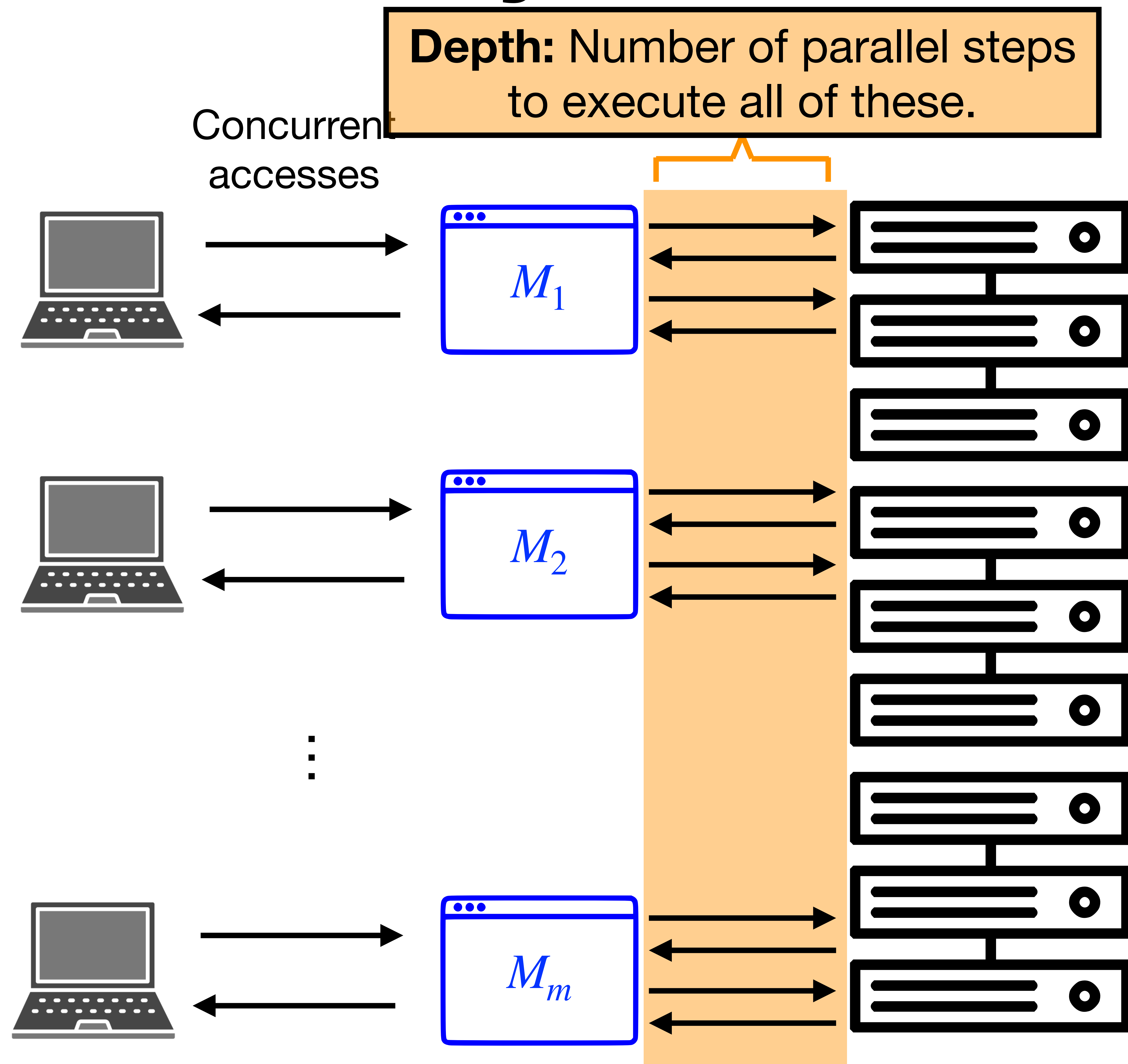
Memory Checking for Parallel RAMs



Efficiency metrics

- **Local Space:** Space per checker. This talk: $O(1)$ words/$O(\lambda)$ bits.
- **Server Space:** Server storage size. This talk: $O(N)$ words.
- **Work blowup:** Ratio of server accesses per underlying PRAM access.
- **Depth blowup:** Number of parallel steps to support a single batch of requests.

Memory Checking for Parallel RAMs



Efficiency metrics

- **Local Space:** Space per checker. This talk: $O(1)$ words/$O(\lambda)$ bits.
- **Server Space:** Server storage size. This talk: $O(N)$ words.
- **Work blowup:** Ratio of server accesses per underlying PRAM access.
- **Depth blowup:** Number of parallel steps to support a single batch of requests.

Our Results

Our Results

Our Results

[Blum-Evans-Gemmel-Kannan-Naor '91]

- For RAM setting, the best constructions have $O(\log N)$ work blow-up. Lower bound of $\Omega(\log N/\log \log N)$ known for special cases.

Our Results

[Blum-Evans-Gemmel-Kannan-Naor ’91]

- For RAM setting, the best constructions have $O(\log N)$ work blow-up. Lower bound of $\Omega(\log N/\log \log N)$ known for special cases.

[Dwork-Naor-Rothblum-Vaikuntanathan ’09]

Our Results

- For RAM setting, the best constructions have $O(\log N)$ work blow-up. Lower bound of $\Omega(\log N / \log \log N)$ known for special cases.

Our Results

- For RAM setting, the best constructions have $O(\log N)$ work blow-up. Lower bound of $\Omega(\log N/\log\log N)$ known for special cases.
  - Immediately gives: m-CPU PRAM memory checker with $O(\log N)$ work blow-up by serialising the algorithm. **But** $O(m\log N)$ depth blowup.

Our Results

- For RAM setting, the best constructions have $O(\log N)$ work blow-up. Lower bound of $\Omega(\log N / \log \log N)$ known for special cases.
    - Immediately gives: m-CPU PRAM memory checker with $O(\log N)$ work blow-up by serialising the algorithm. **But** $O(m \log N)$ depth blowup.
- We show how to also obtain $O(\log N)$ **depth**.

Our Results

- For RAM setting, the best constructions have $O(\log N)$ work blow-up. Lower bound of $\Omega(\log N / \log \log N)$ known for special cases.
  - Immediately gives: m-CPU PRAM memory checker with $O(\log N)$ work blow-up by serialising the algorithm. **But** $O(m \log N)$ depth blowup.
- We show how to also obtain $O(\log N)$ **depth**.

Theorem 1. Assuming OWFs, there exists a memory checking protocol for PRAM programs with $O(\log N)$ **worst-case work** and **depth** blowup.

Our Results

- For RAM setting, the best constructions have $O(\log N)$ work blow-up. Lower bound of $\Omega(\log N / \log\log N)$ known for special cases.
  - Immediately gives: m-CPU PRAM memory checker with $O(\log N)$ work blow-up by serialising the algorithm. **But** $O(m \log N)$ depth blowup.
- We show how to also obtain $O(\log N)$ **depth**.

Assumption is minimal [Naor-Rothblum '05]

Theorem 1. Assuming OWFs, there exists a memory checking protocol for PRAM programs with $O(\log N)$ **worst-case work** and **depth** blowup.

Application to Oblivious Parallel RAM

Application to Oblivious Parallel RAM

- Oblivious Parallel RAMs (OPRAMs) are access-pattern hiding PRAM compilers.

[Boyle-Chung-Pass ’16]

Application to Oblivious Parallel RAM

- Oblivious Parallel RAMs (OPRAMs) are access-pattern hiding PRAM compilers.
  [Boyle-Chung-Pass ’16]
- A recent work constructed an **honest-but-curious** OPRAM constructions with $O(\log N)$ blowup in both **work** and **depth.** Optimal!
  [Asharov-Komargodski-Lin-Peserico-Shi ’22]

Application to Oblivious Parallel RAM

- Oblivious Parallel RAMs (OPRAMs) are access-pattern hiding PRAM compilers.

  [Boyle-Chung-Pass ’16]

- A recent work constructed an **honest-but-curious** OPRAM constructions with $O(\log N)$ blowup in both **work** and **depth.** Optimal!

  [Asharov-Komargodski-Lin-Peserico-Shi ’22]

- We obtain the **first** construction of **maliciously secure** OPRAM with **polylogarithmic overhead.**

Application to Oblivious Parallel RAM

- Oblivious Parallel RAMs (OPRAMs) are access-pattern hiding PRAM compilers.

[Boyle-Chung-Pass '16]

- A recent work constructed an **honest-but-curious** OPRAM constructions with $O(\log N)$ blowup in both **work** and **depth.** Optimal!

[Asharov-Komargodski-Lin-Peserico-Shi '22]

- We obtain the **first** construction of **maliciously secure** OPRAM with **polylogarithmic overhead.**

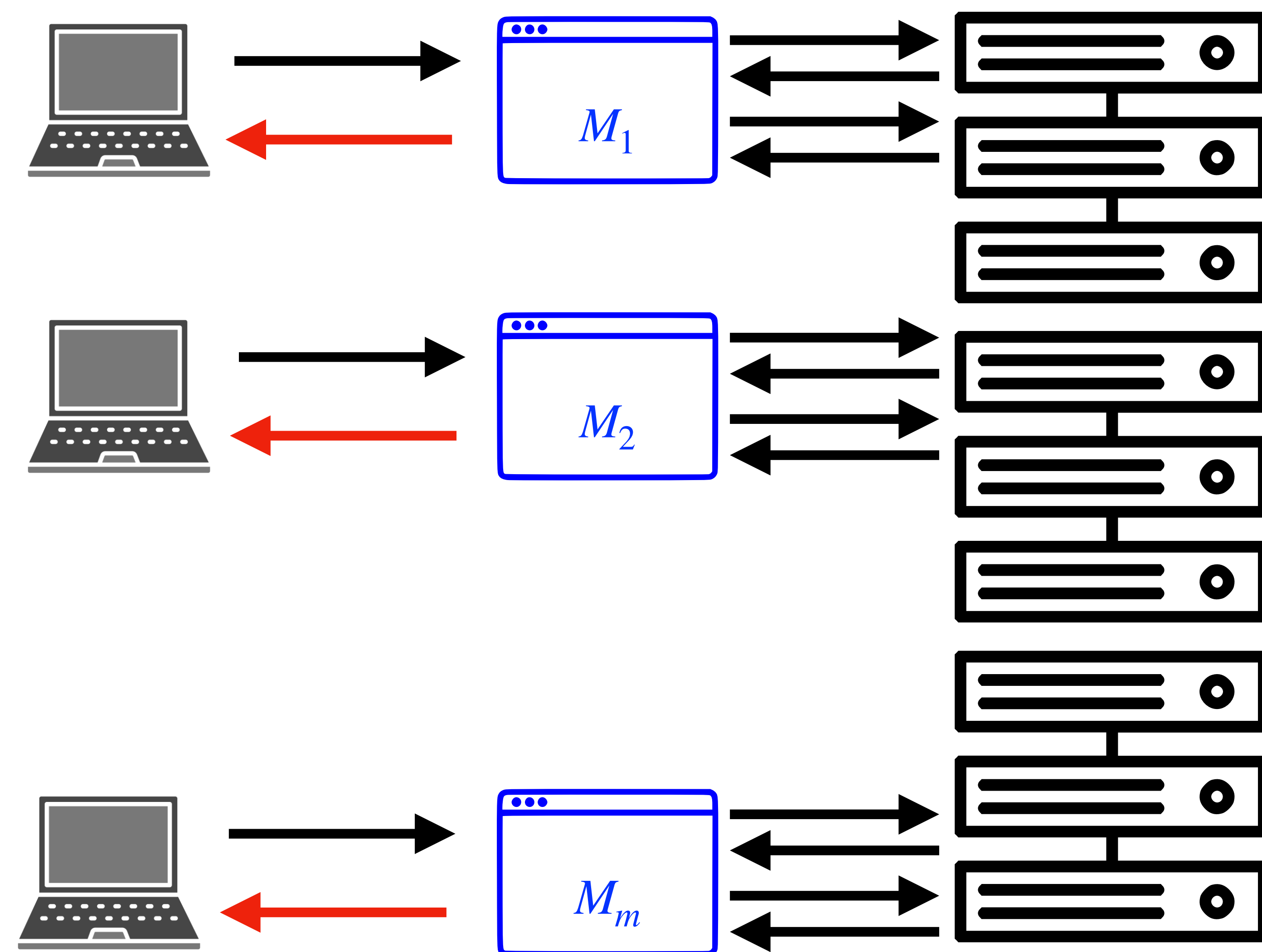
Theorem 2. Assuming OWFs, there exists an **maliciously secure OPRAM** compiler with $O(\log^2 N)$ **work** and **depth** blowup*.

Our Results

Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.

Our Results

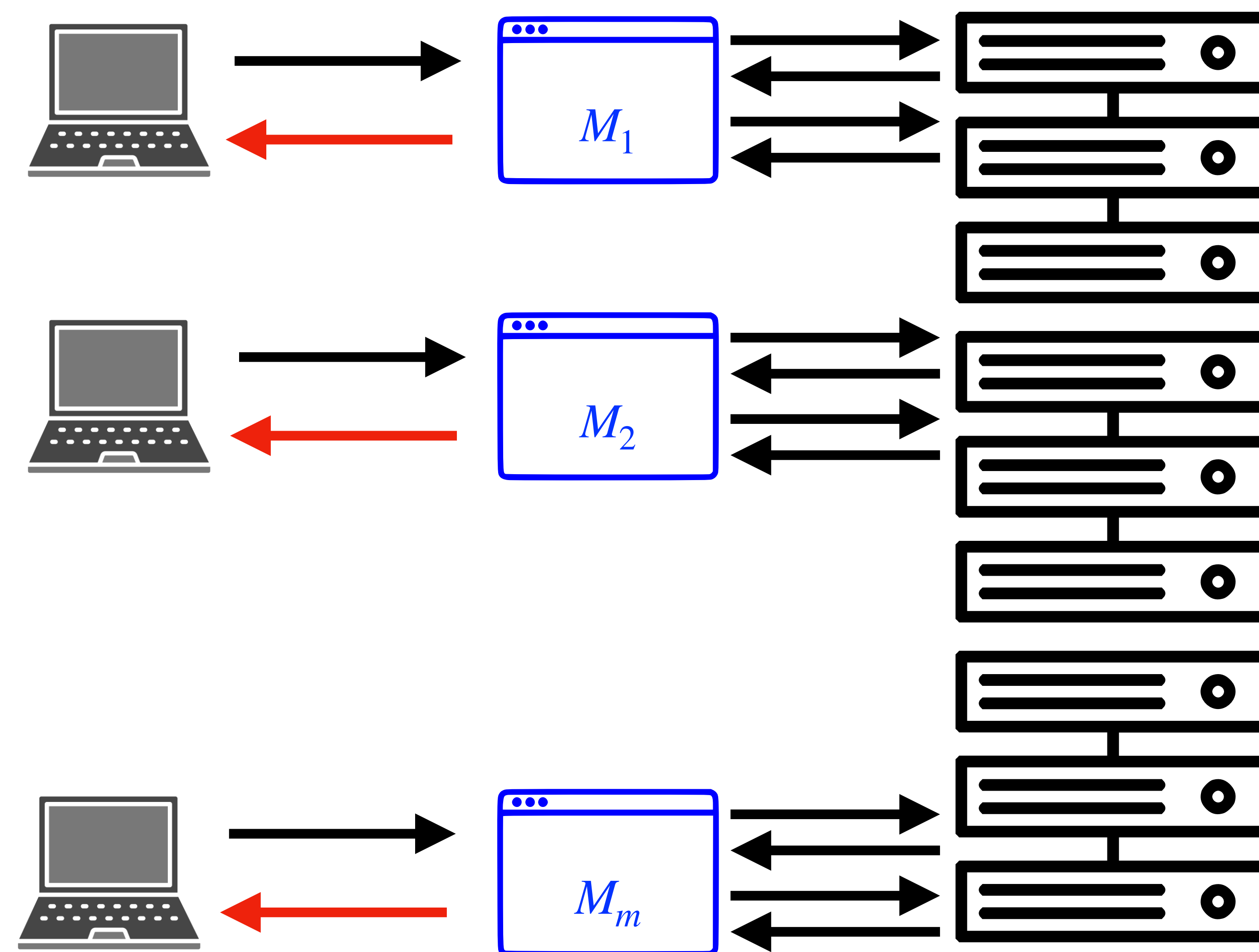
Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.



Query phase: Answers can be wrong!
Repeat until clients say done.

Our Results

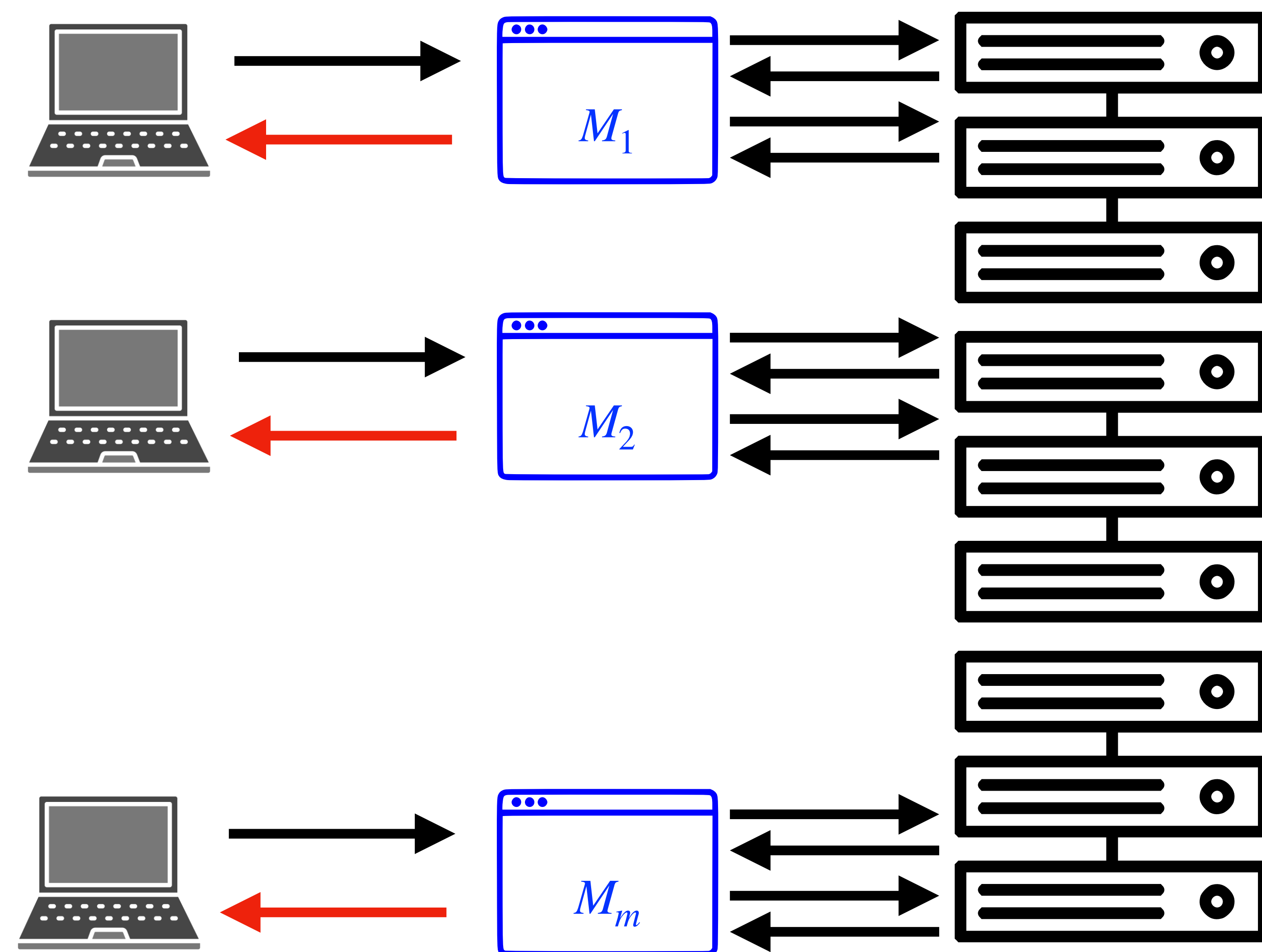
Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.



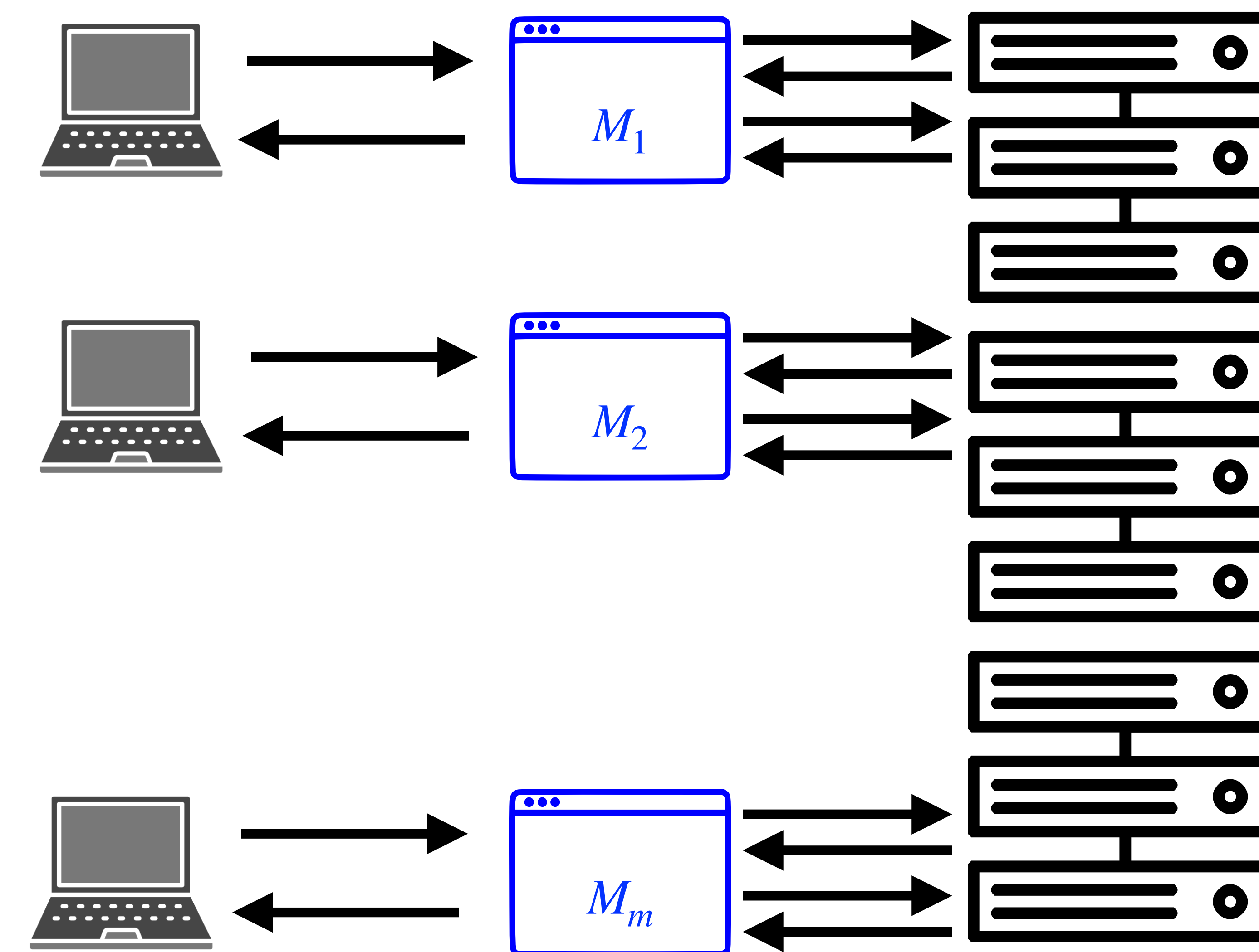
Query phase: Answers can be wrong!
Repeat until clients say done.

Our Results

Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.



Query phase: Answers can be wrong!
Repeat until clients say done.



Verification phase: Reports if all correct or some mistake.

Our Results

Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.

Our Results

Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** bl

Checks if any mistake happened after a large batch of concurrent requests

Our Results

Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.

Our Results

Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.

- **Idea:** Adapt the counting technique from a previous work [**M.**-Vafa '23]

Our Results

Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.

- **Idea:** Adapt the counting technique from a previous work [**M.**-Vafa '23]
- Memory checkers maintain local counters T_i of the number of updates.

Our Results

Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.

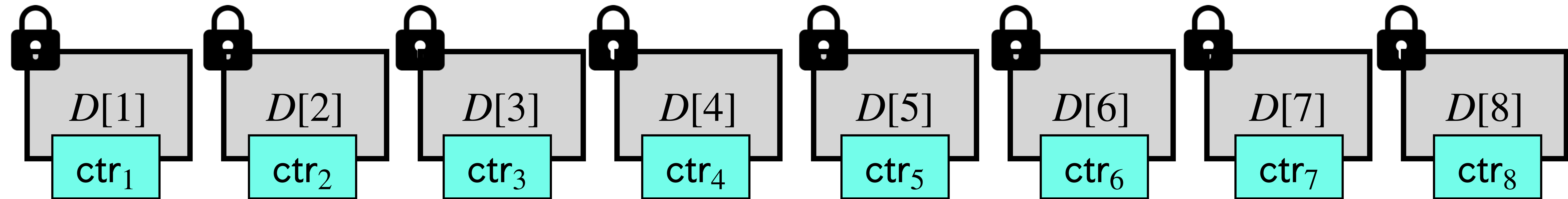
- **Idea:** Adapt the counting technique from a previous work [**M.**-Vafa ’23]
- Memory checkers maintain local counters T_i of the number of updates.
- Every database entry is tagged with counters ctr_i

Our Results

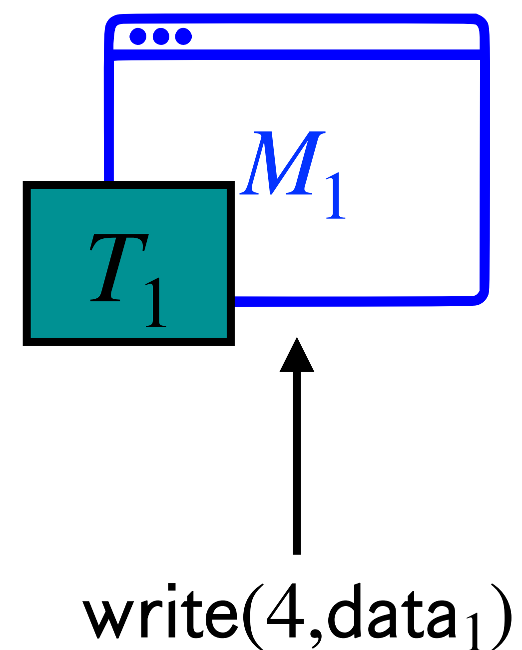
Theorem 3. Assuming OWFs, there exists an *offline* memory checking protocol for PRAM programs with $O(1)$ amortised **work** and **depth** blowup.

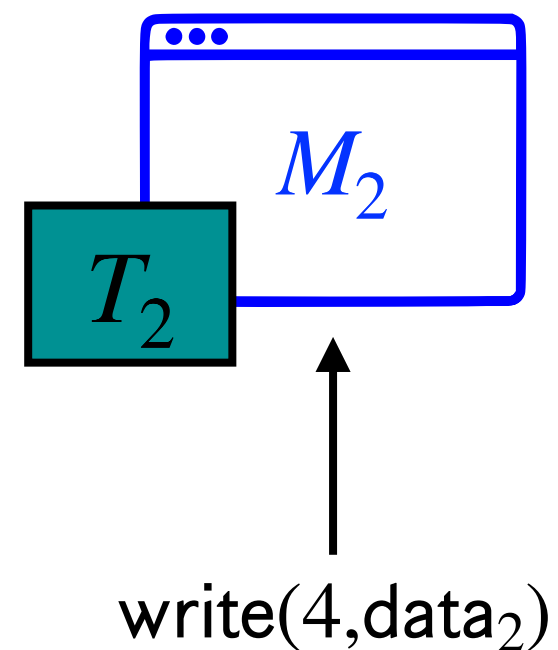
- **Idea:** Adapt the counting technique from a previous work [**M.**-Vafa '23]
- Memory checkers maintain local counters T_i of the number of updates.
- Every database entry is tagged with counters ctr_i
- Verification phase: Check if $\sum_i \mathsf{ctr}_i = \sum_j T_j$.

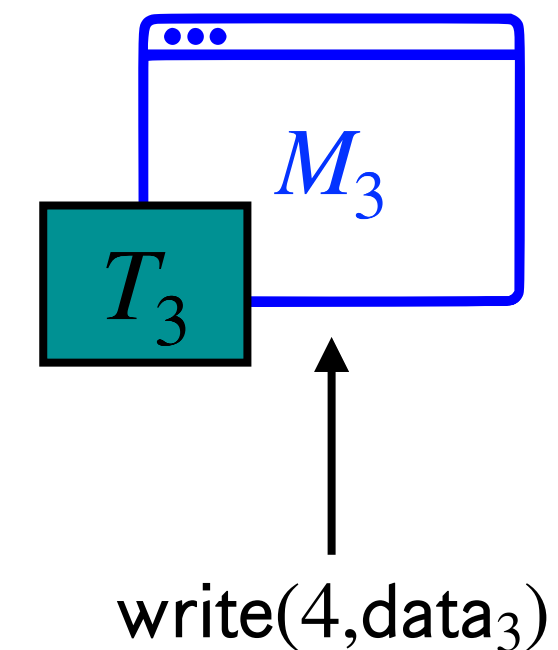
New: Offline-Checking for PRAMs



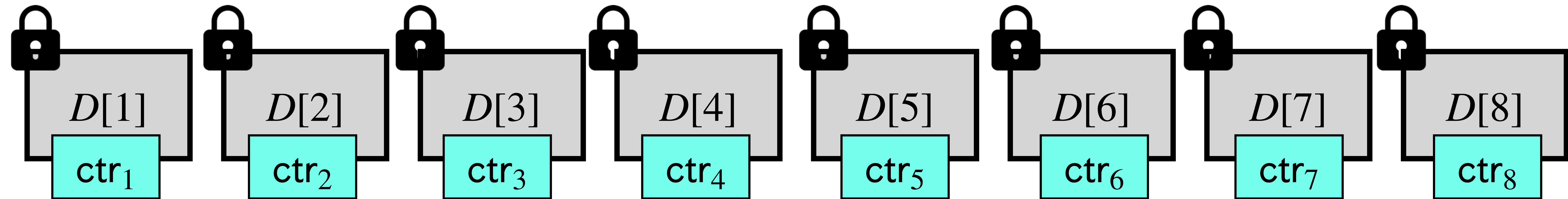
ctr_i: #times $D[i]$ updated
T_i : #times M_i wrote



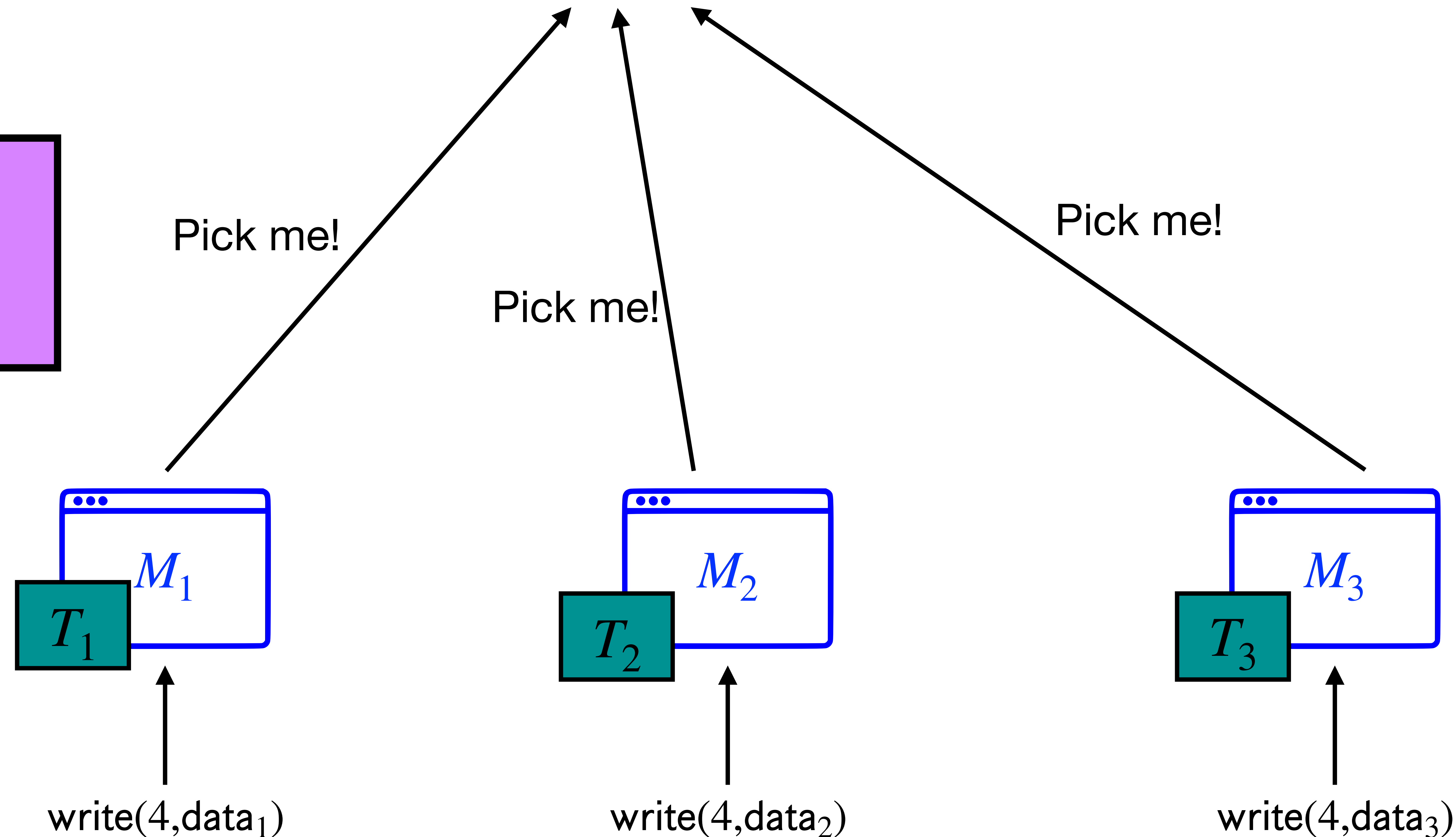




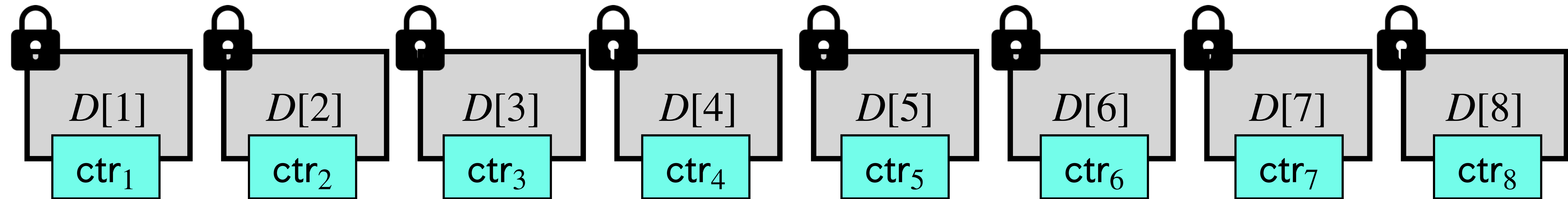
New: Offline-Checking for PRAMs



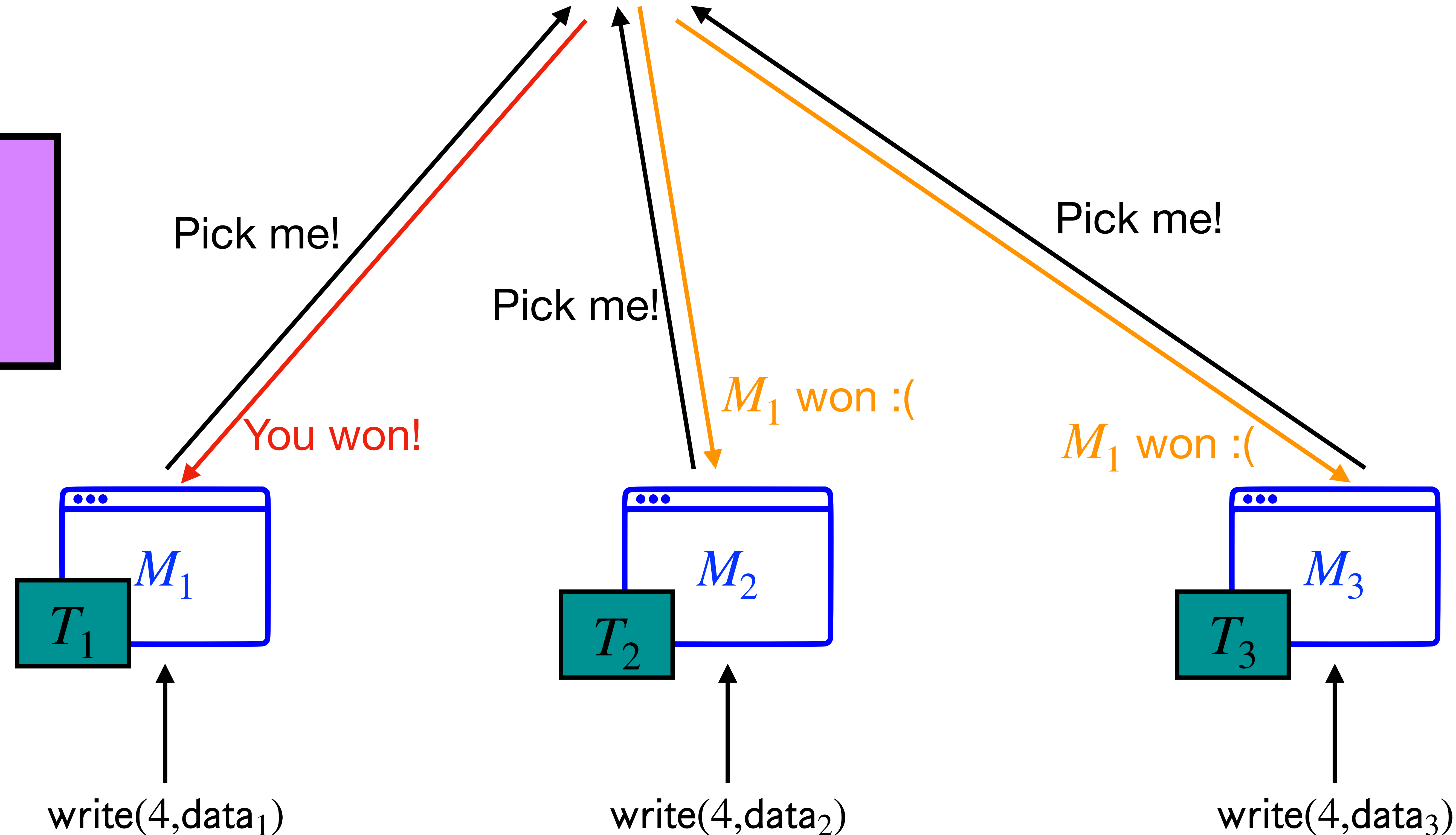
ctr_i: #times $D[i]$ updated
T_i : #times M_i wrote



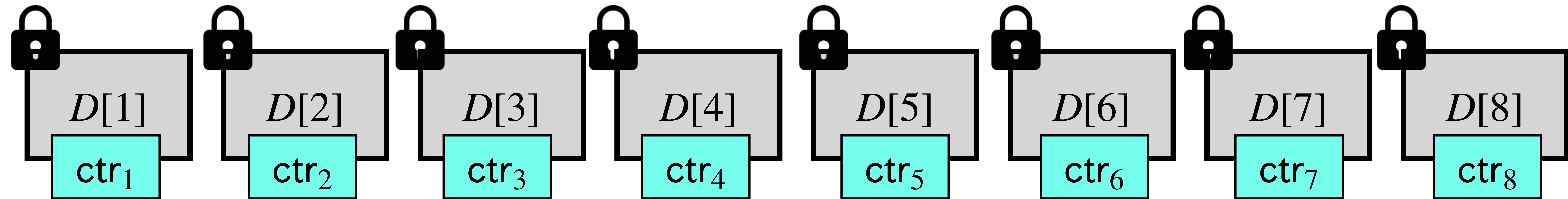
New: Offline-Checking for PRAMs



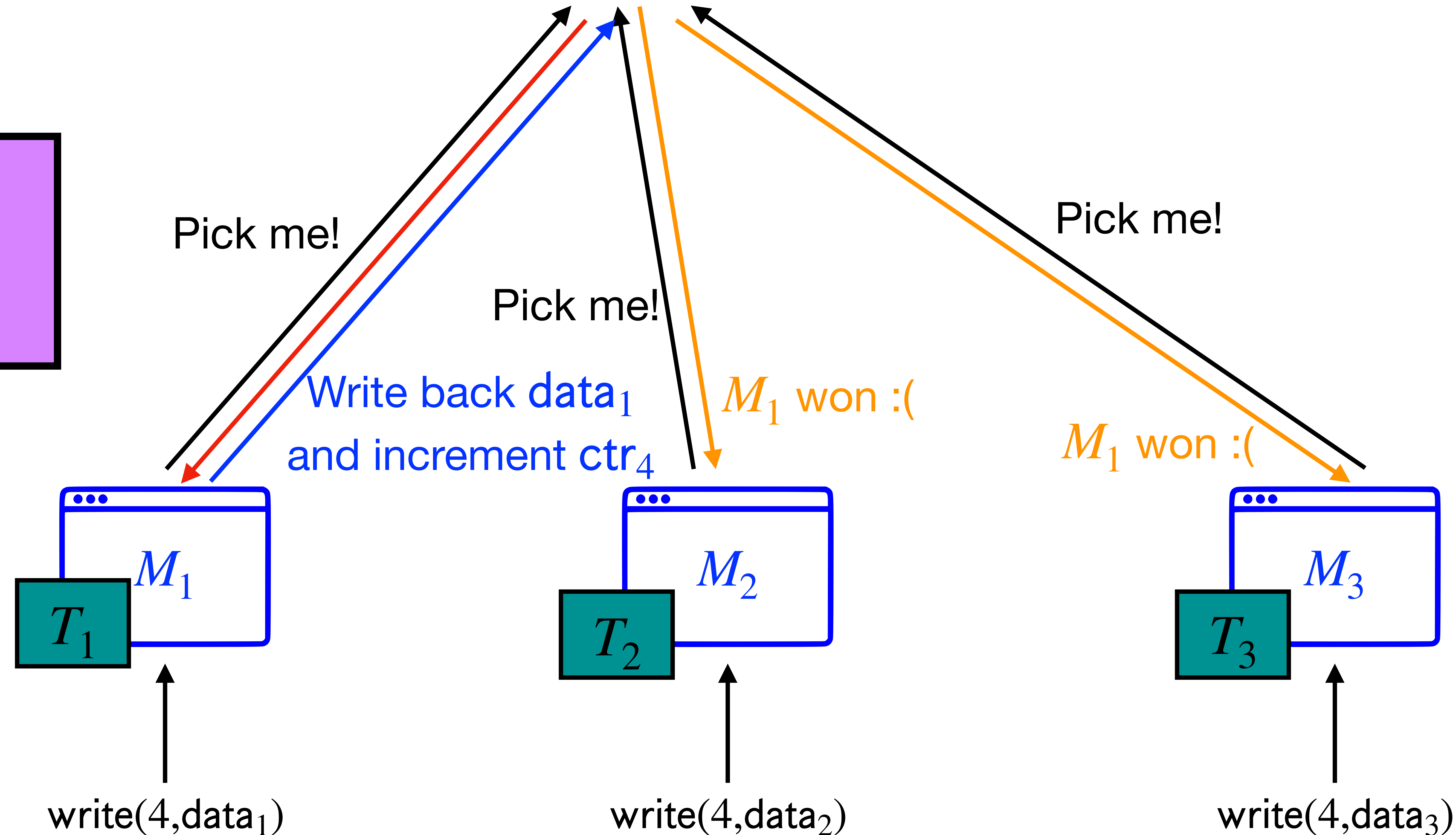
ctr_i: #times $D[i]$ updated
T_i : #times M_i wrote



New: Offline-Checking for PRAMs



ctr_i: #times $D[i]$ updated
T_i : #times M_i wrote



New: Offline-Checking for PRAMs

$D[1]$ $D[2]$ $D[3]$ $D[4]$ $D[5]$ $D[6]$ $D[7]$ $D[8]$

ctr_1 ctr_2 ctr_3 ctr_4 ctr_5 ctr_6 ctr_7 ctr_8

Incremented!

ctr_i: #times $D[i]$ updated
T_i : #times M_i wrote

Pick me!

Pick me!

Pick me!

Write back data_1
and increment ctr_4

M_1 won :(

M_1 won :(

Incremented!

M_1 M_2 M_3

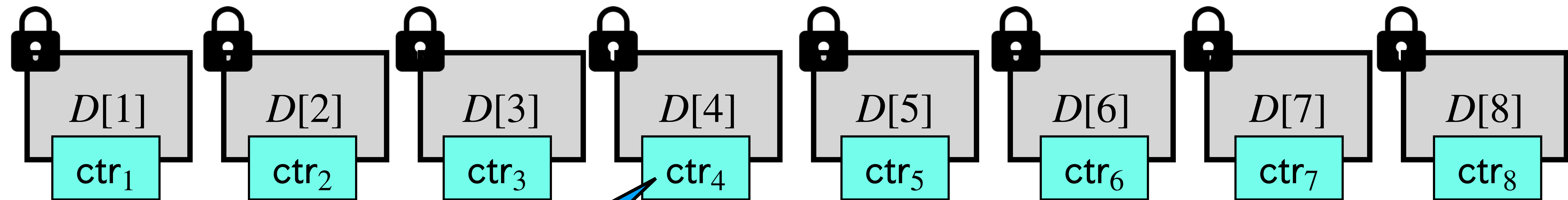
T_1 T_2 T_3

write(4,data_1)

write(4,data_2)

write(4,data_3)

New: Offline-Checking for PRAMs



Incremented!

ctr_i: #times $D[i]$ updated
T_i : #times M_i wrote

Pick me!

Pick me!

Pick me!

Write back data_1 and increment ctr_4

M_1 won :(

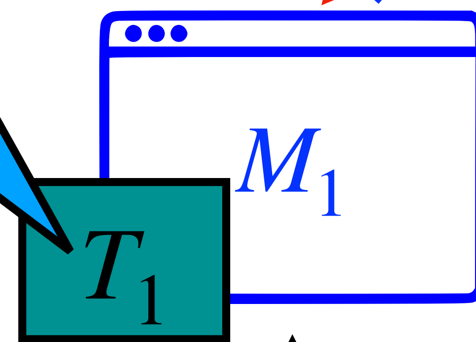
M_1 won :(

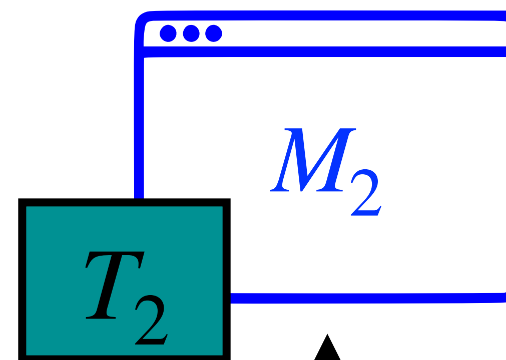
Note that this maintains:

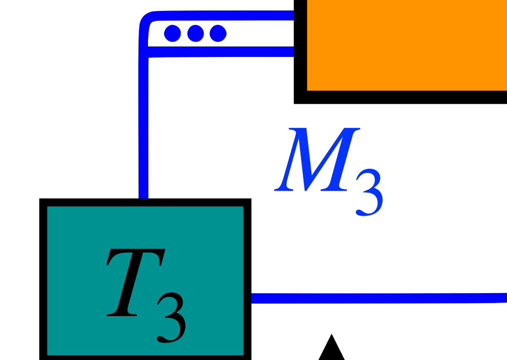
$$\sum_i \text{ctr}_i = \sum_j T_j$$

since both sides are incremented

Incremented!





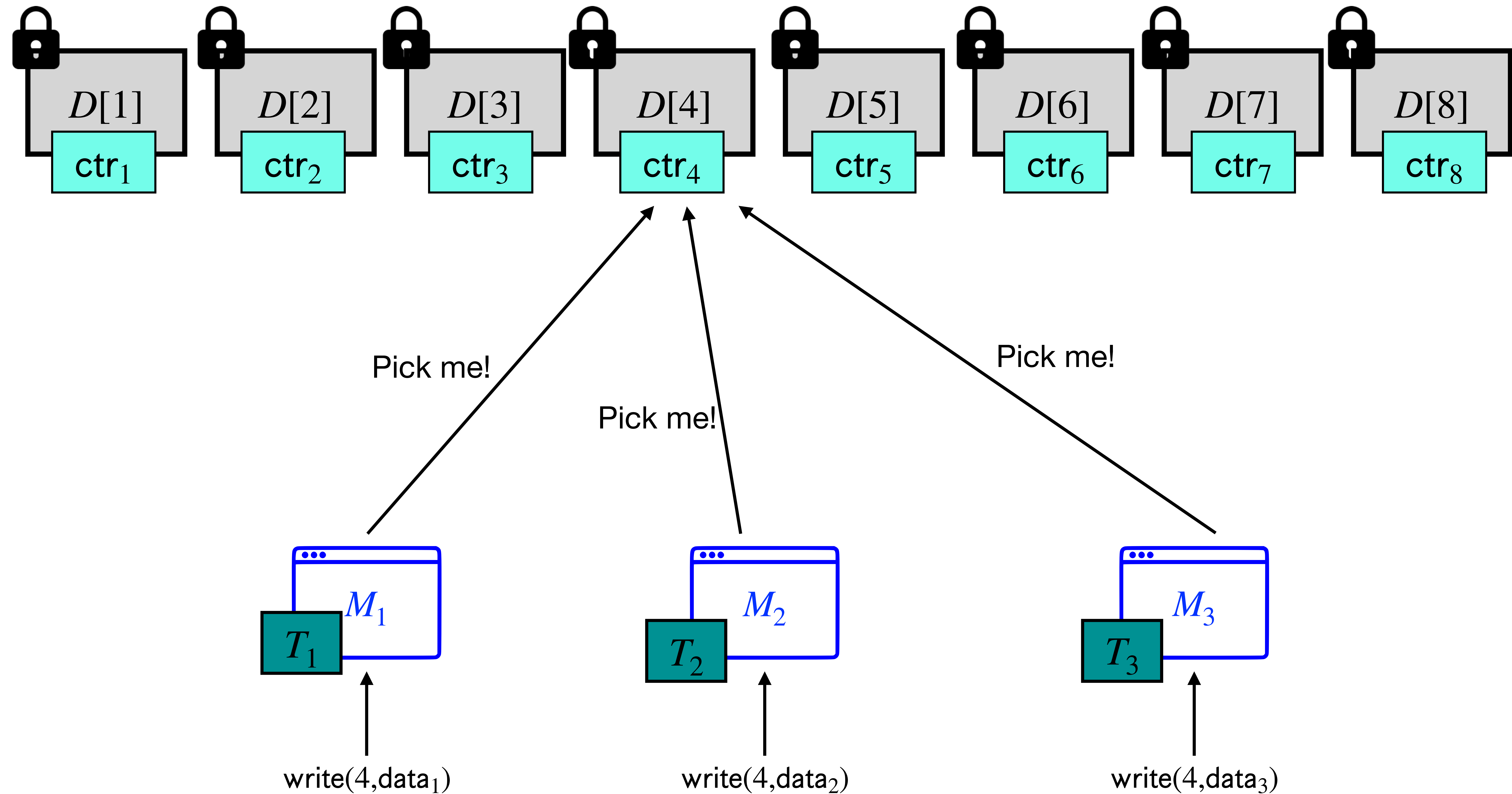


write(4,data_1)

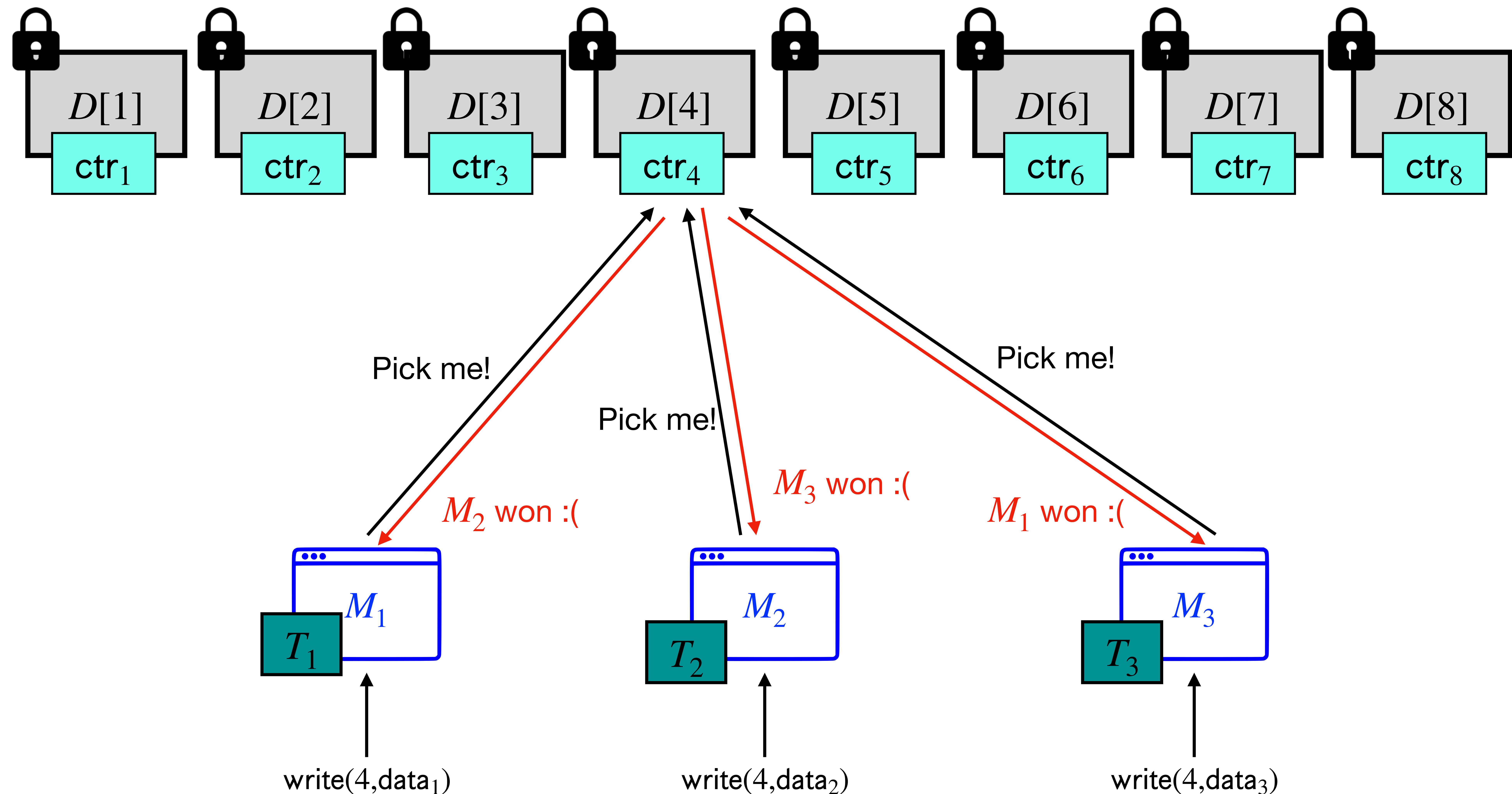
write(4,data_2)

write(4,data_3)

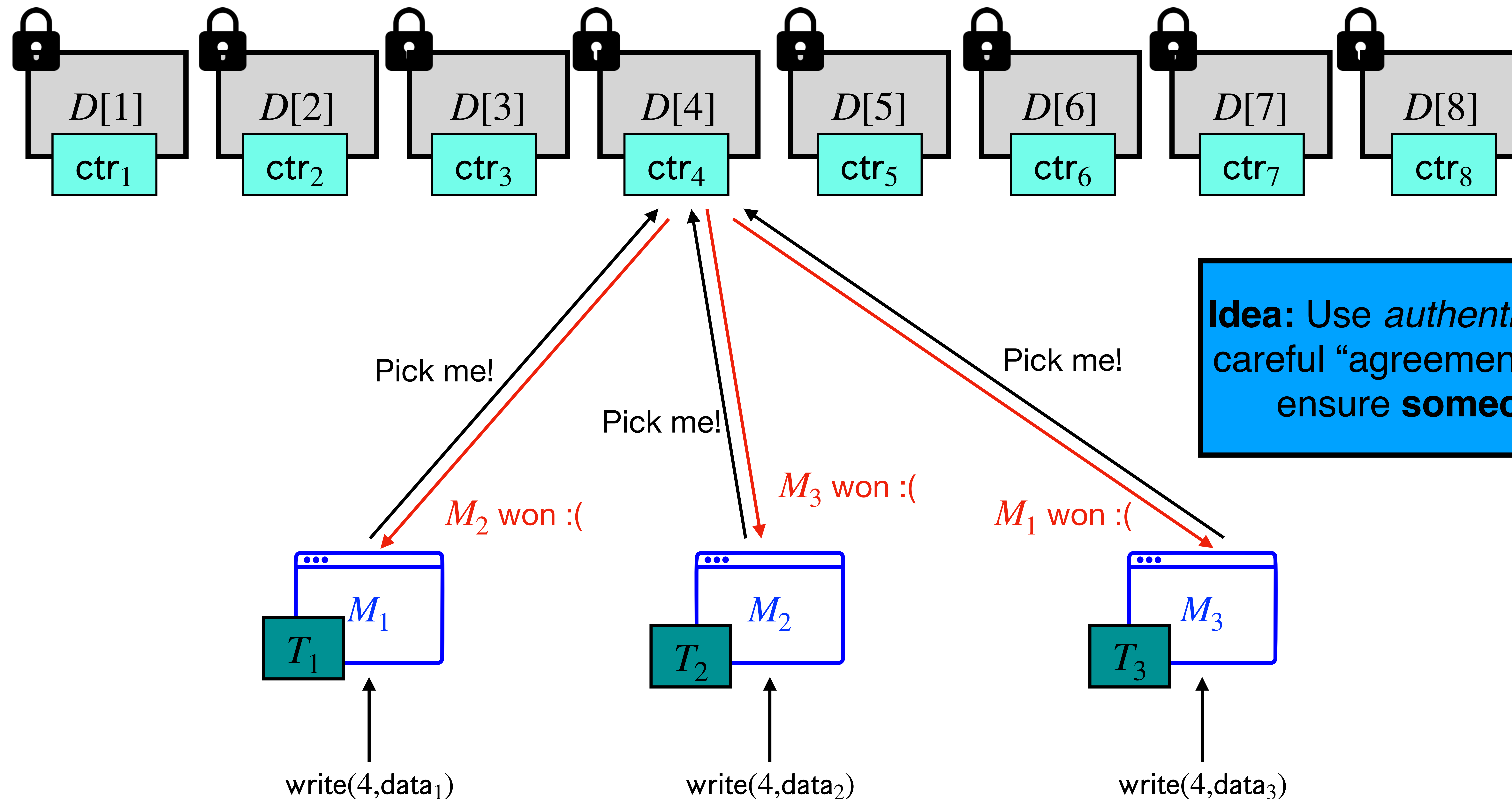
Issue 1: Spoofing Attack



Issue 1: Spoofing Attack

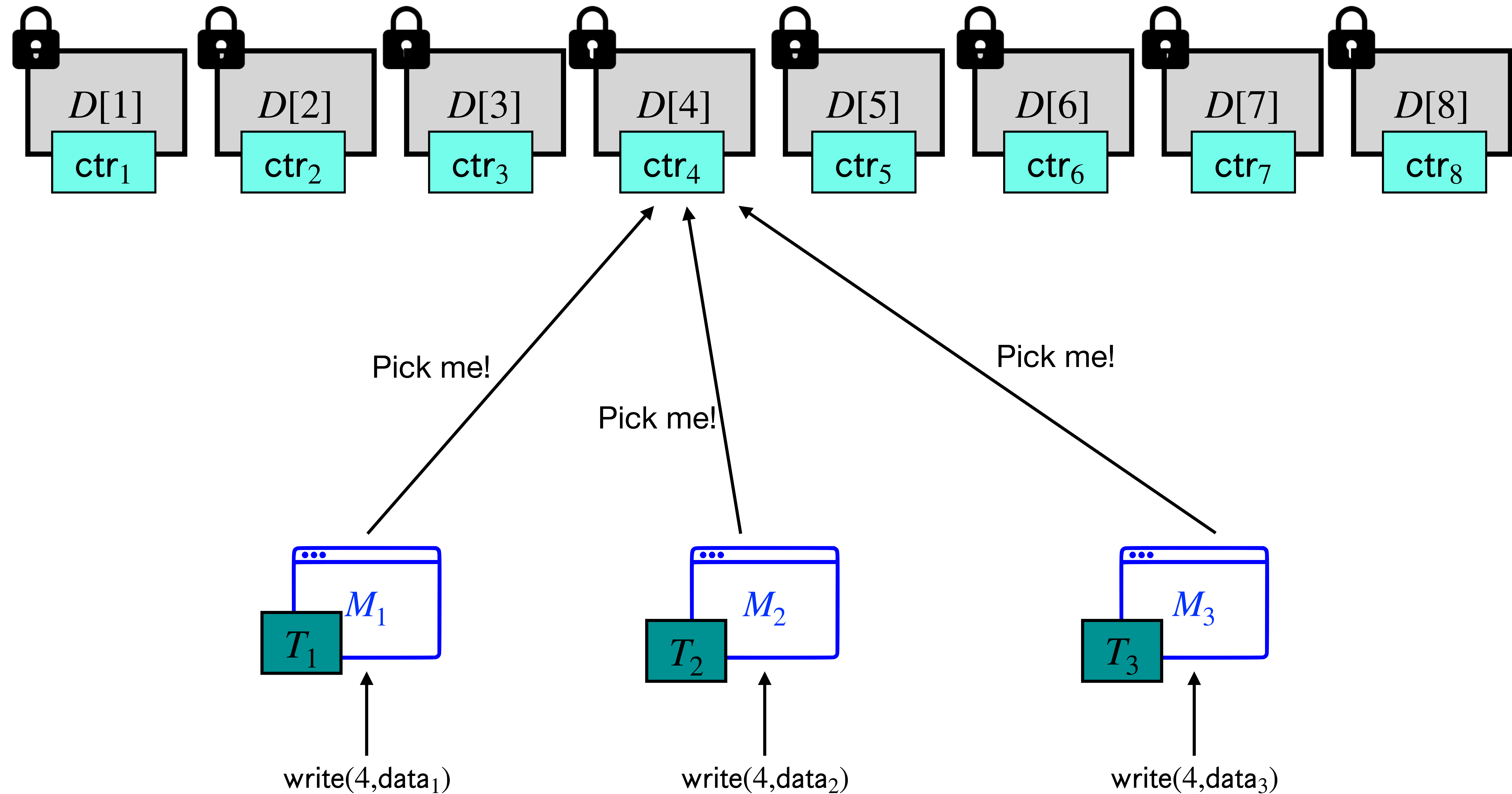


Issue 1: Spoofing Attack

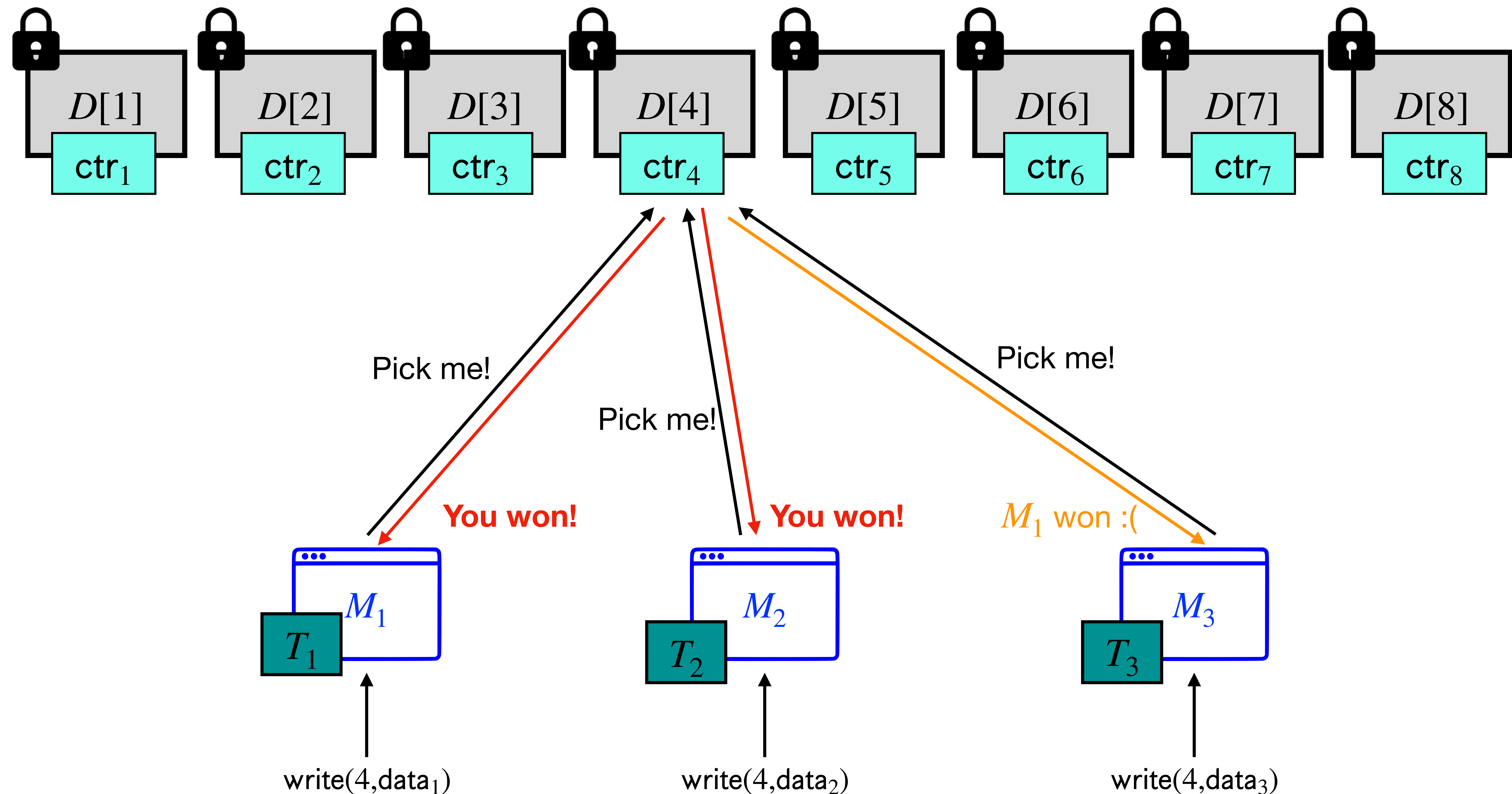


Idea: Use *authentication*, and a careful "agreement" protocol to ensure **someone** wins.

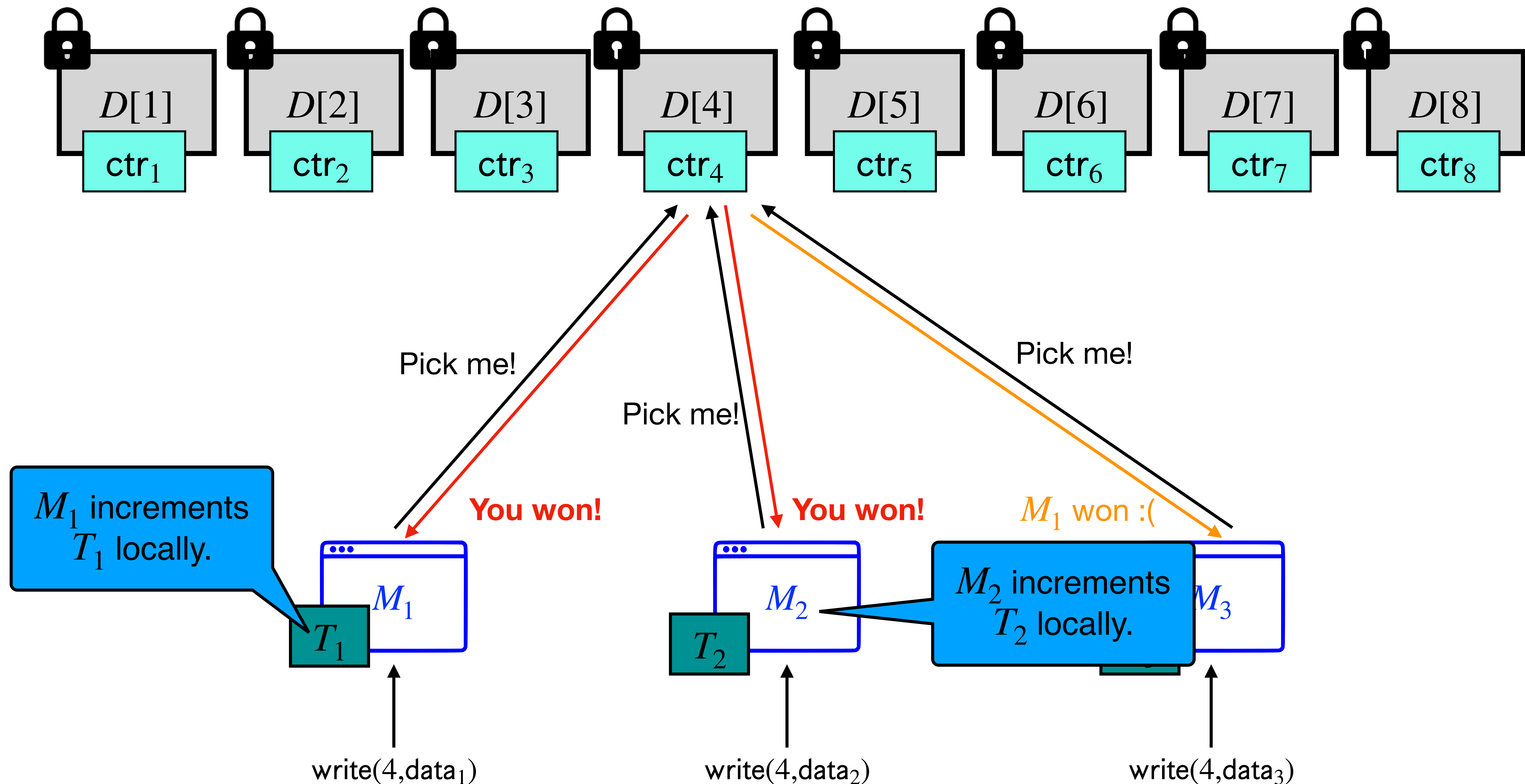
Issue 2: Branching attack



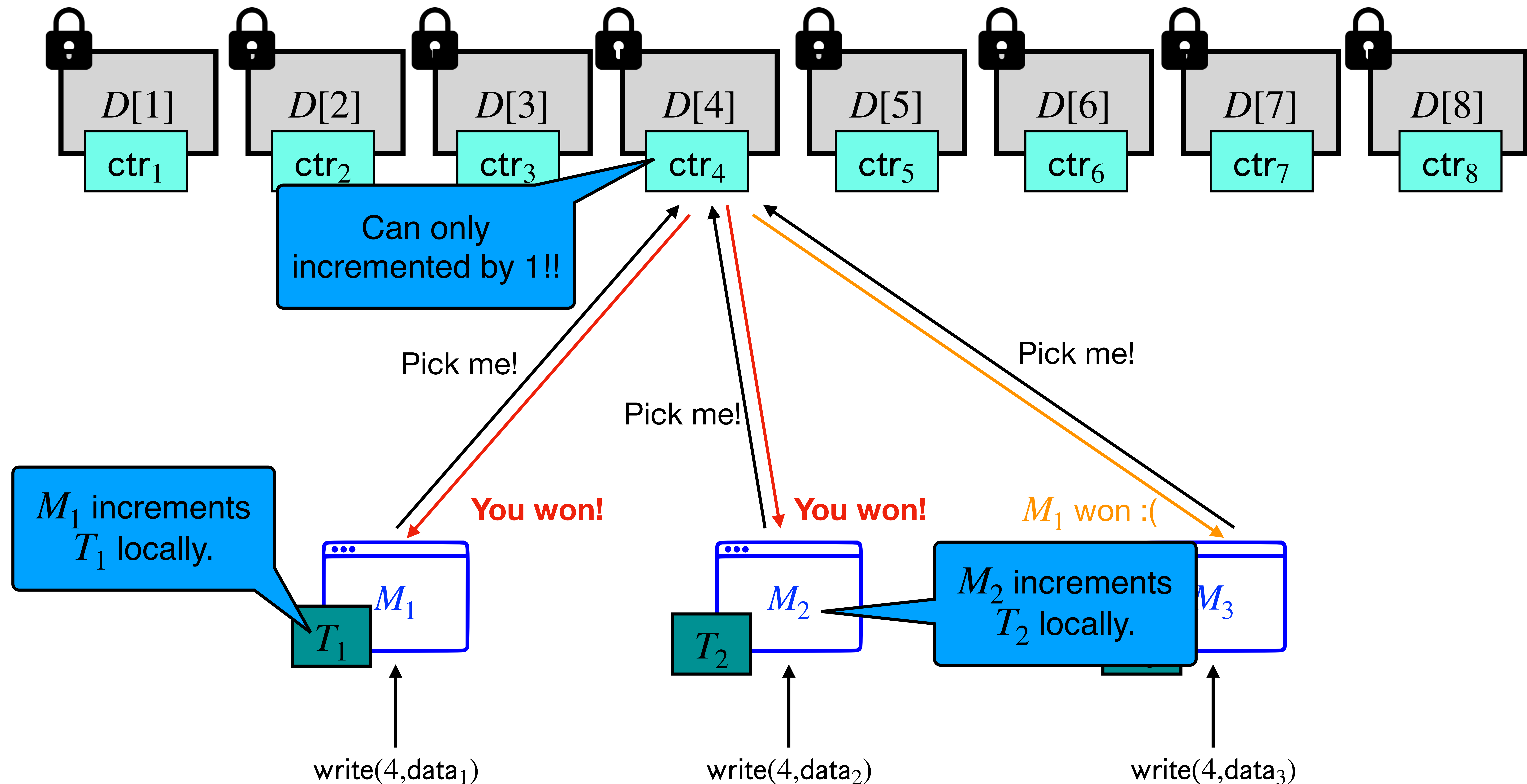
Issue 2: Branching attack



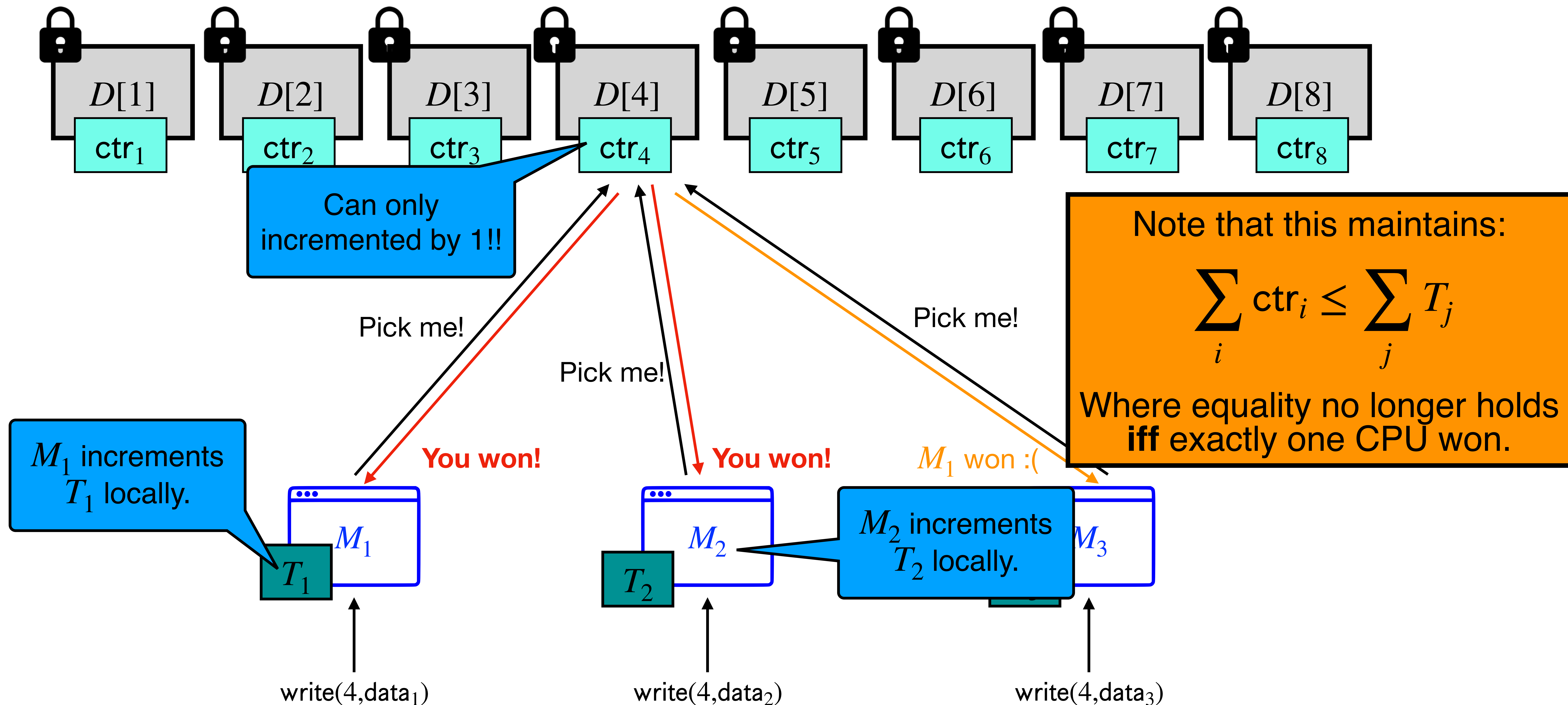
Issue 2: Branching attack



Issue 2: Branching attack



Issue 2: Branching attack



Open Problems

Open Problems

- Can we obtain a **statistically secure** *offline* memory checker for CRCW programs with $O(1)$ amortised overhead?

Open Problems

- Can we obtain a **statistically secure** *offline* memory checker for CRCW programs with $O(1)$ amortised overhead?
    - Known for single RAM setting, and we show for PRAM setting **without** concurrent read/writes.

Open Problems

- Can we obtain a **statistically secure** *offline* memory checker for CRCW programs with $O(1)$ amortised overhead?
  - Known for single RAM setting, and we show for PRAM setting **without** concurrent read/writes.
- Can we use these memory checkers to obtain an **optimal maliciously secure OPRAM** with $O(\log N)$ work and depth blow-up?

Open Problems

- Can we obtain a **statistically secure** *offline* memory checker for CRCW programs with $O(1)$ amortised overhead?
    - Known for single RAM setting, and we show for PRAM setting **without** concurrent read/writes.
- Can we use these memory checkers to obtain an **optimal maliciously secure OPRAM** with $O(\log N)$ work and depth blow-up?
    - [**M**-Vafa ’23] shows an $O(\log N)$ **maliciously secure** ORAM construction by interleaving *offline* and *online* memory checking.

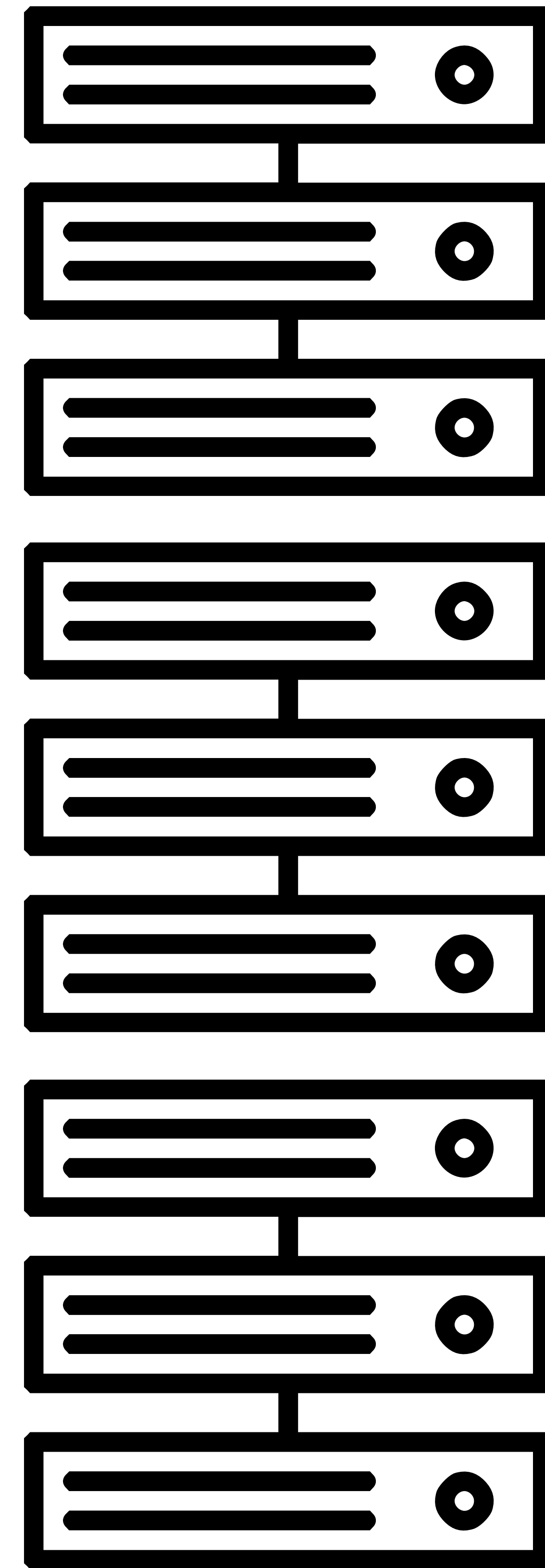
Open Problems

- Can we obtain a **statistically secure** *offline* memory checker for CRCW programs with $O(1)$ amortised overhead?
  - Known for single RAM setting, and we show for PRAM setting **without** concurrent read/writes.
- Can we use these memory checkers to obtain an **optimal maliciously secure OPRAM** with $O(\log N)$ work and depth blow-up?
  - [**M**-Vafa '23] shows an $O(\log N)$ **maliciously secure** ORAM construction by interleaving *offline* and *online* memory checking.
  - Can we do the same?

Bonus Slides

CRCW Parallel RAM Model

⋮

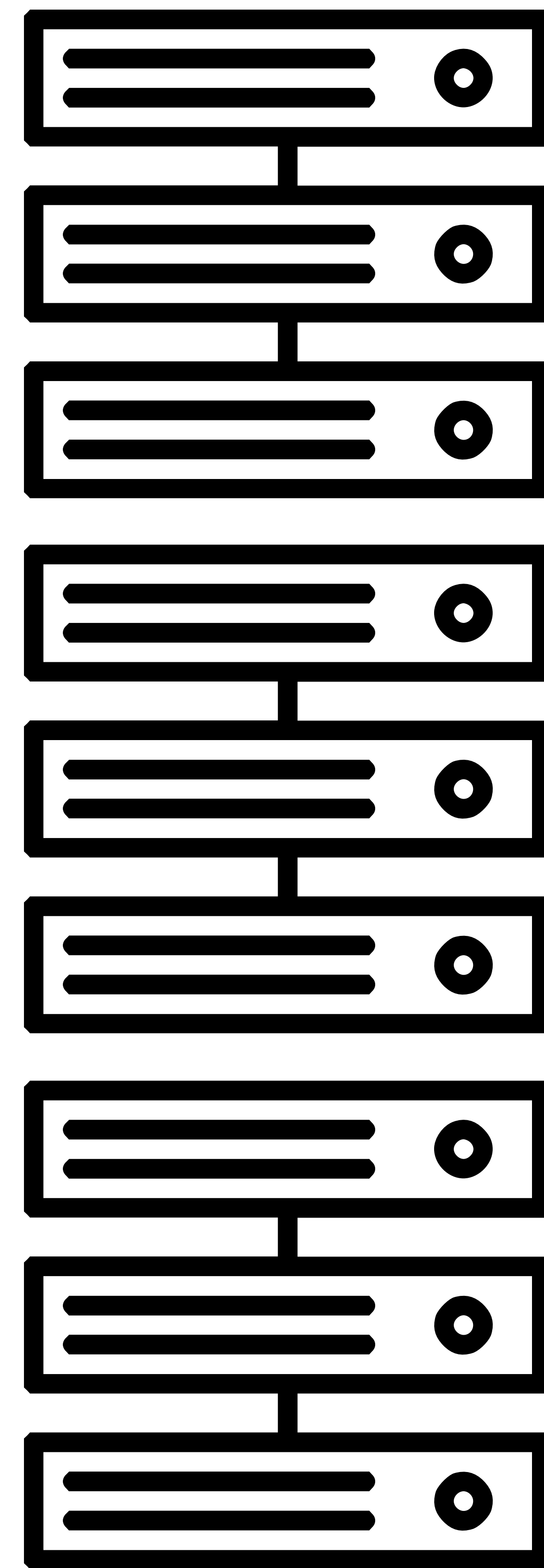


m Clients

Database of size N

CRCW Parallel RAM Model

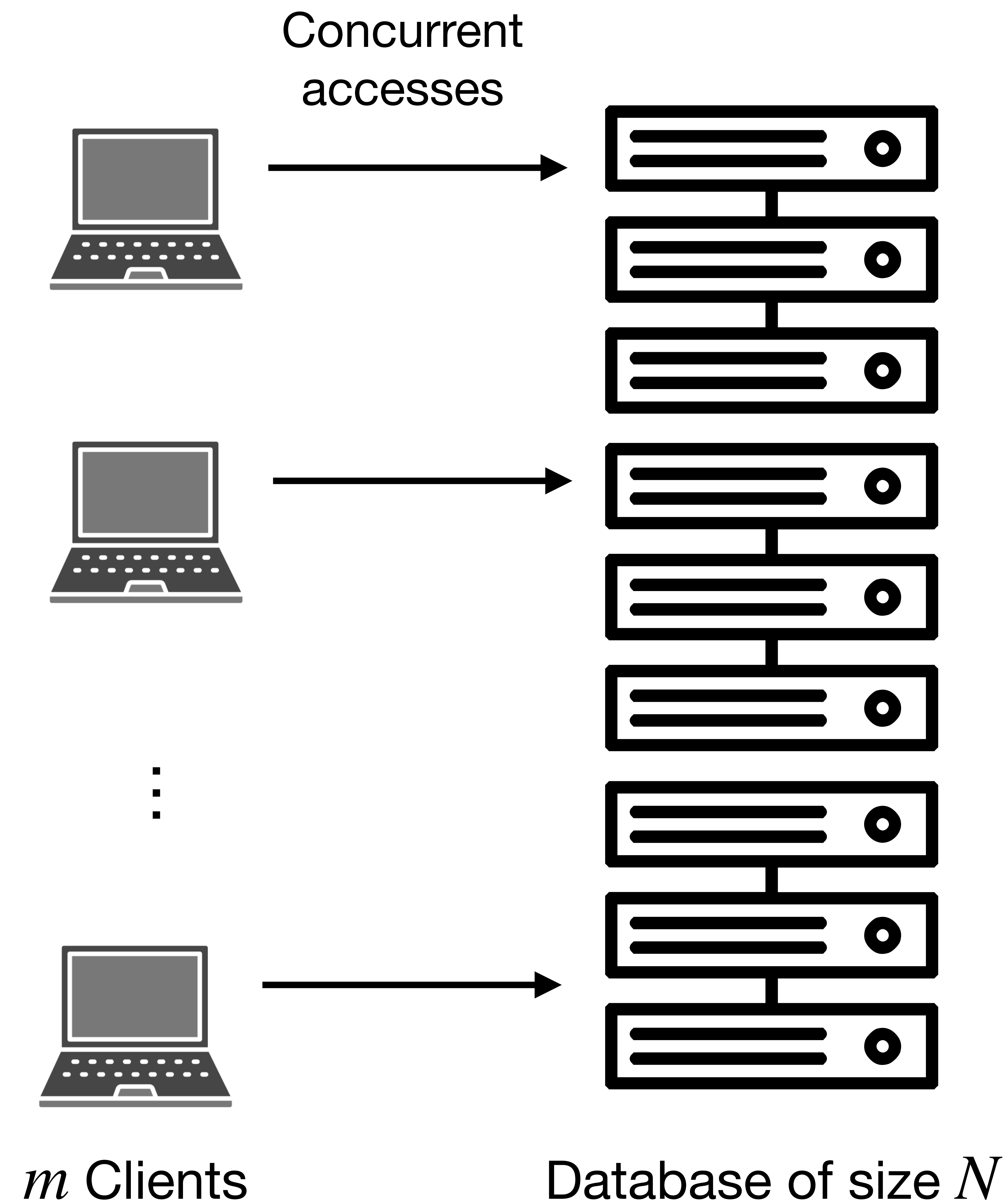
m Clients



Database of size N

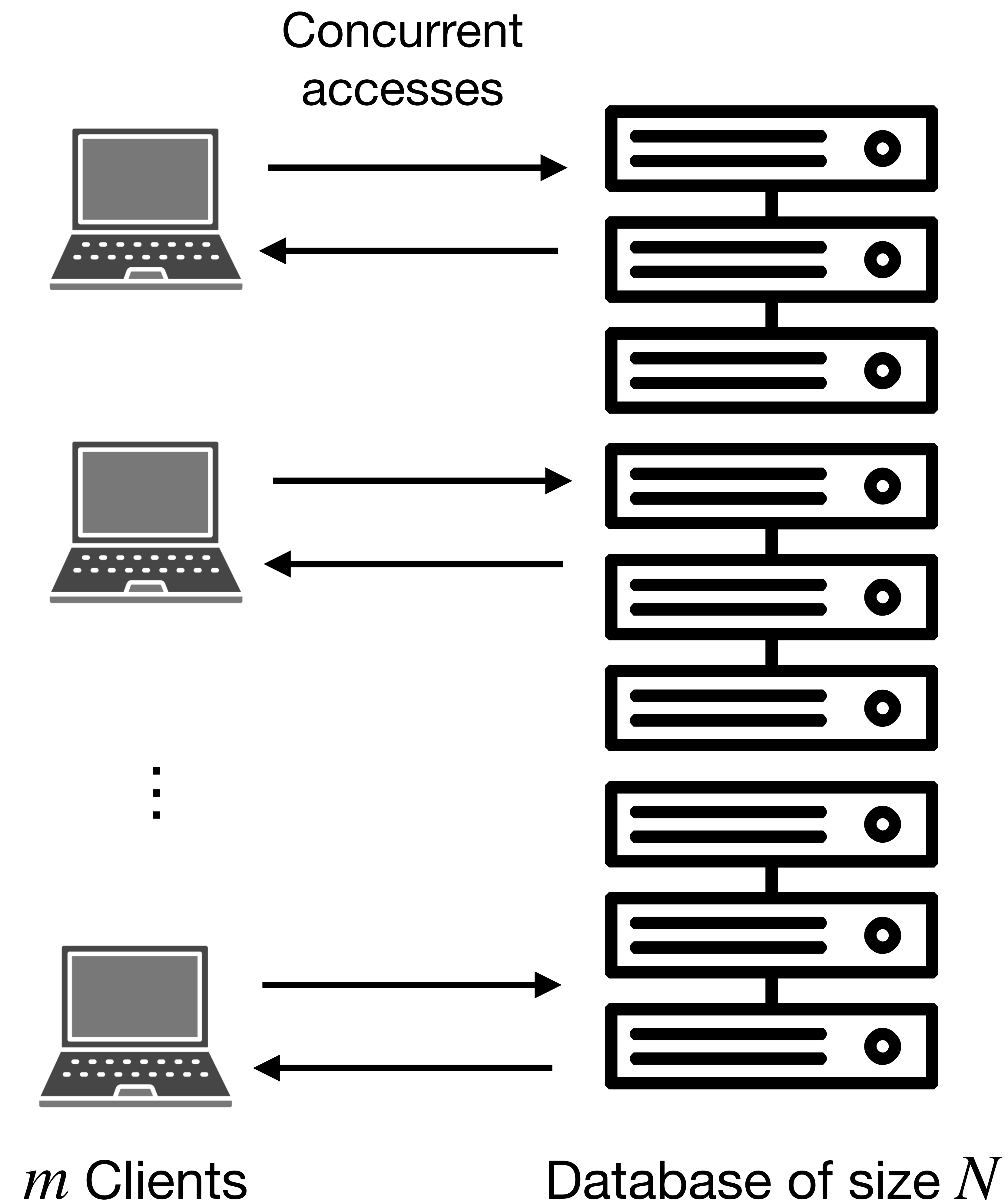
- **PRAM:** Multiple CPUs accessing shared memory

CRCW Parallel RAM Model



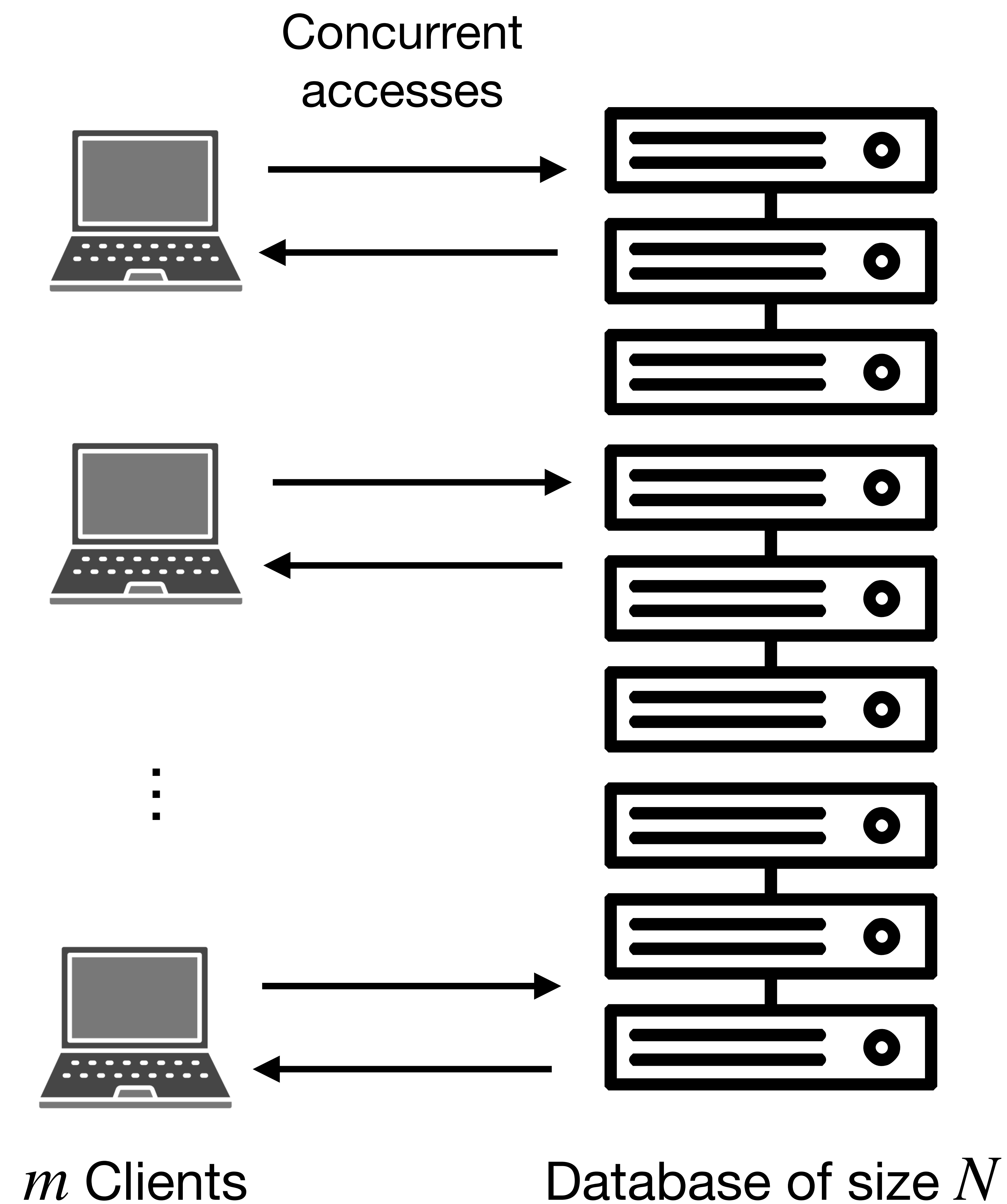
- **PRAM:** Multiple CPUs accessing shared memory

CRCW Parallel RAM Model



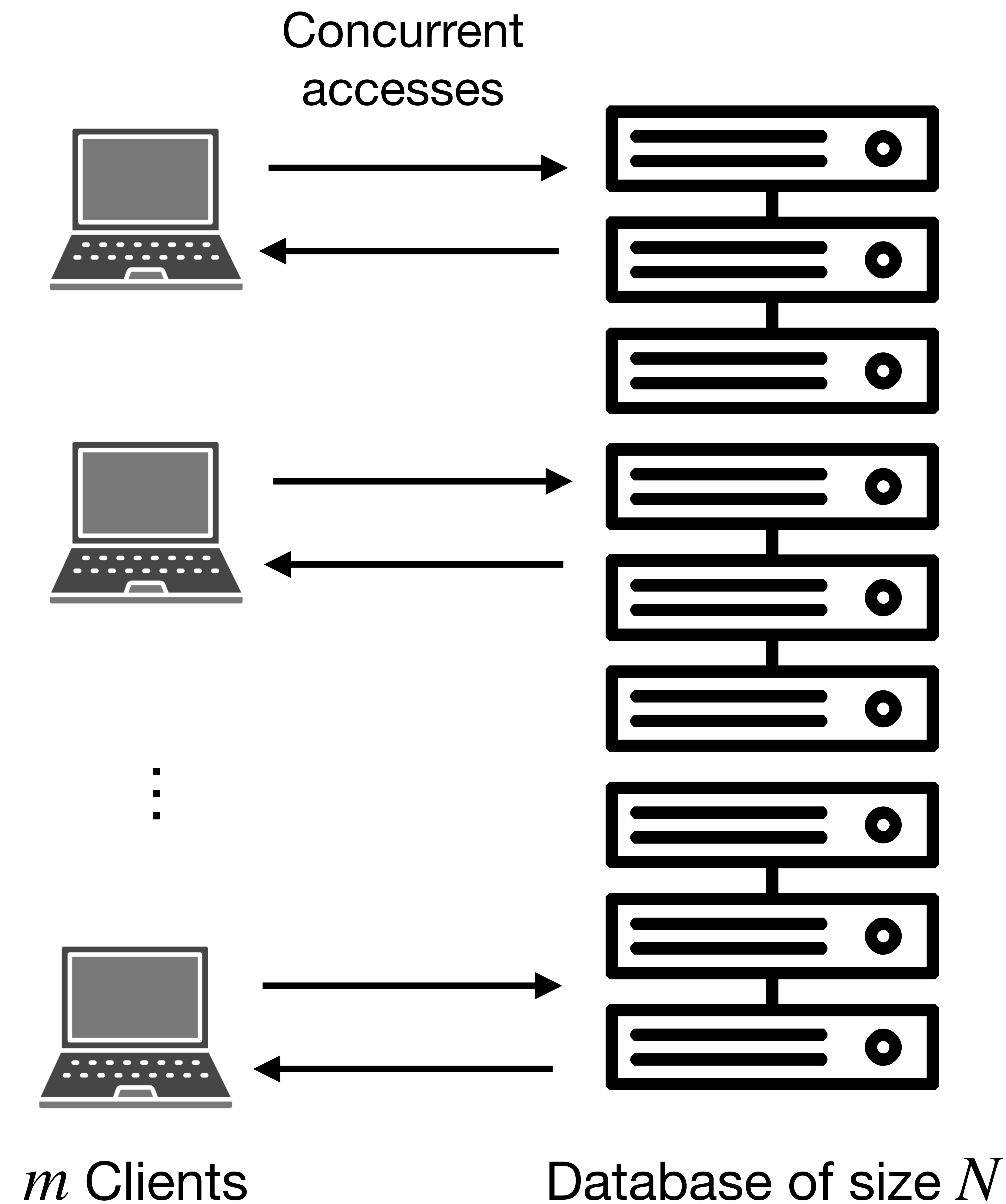
- **PRAM:** Multiple CPUs accessing shared memory

CRCW Parallel RAM Model



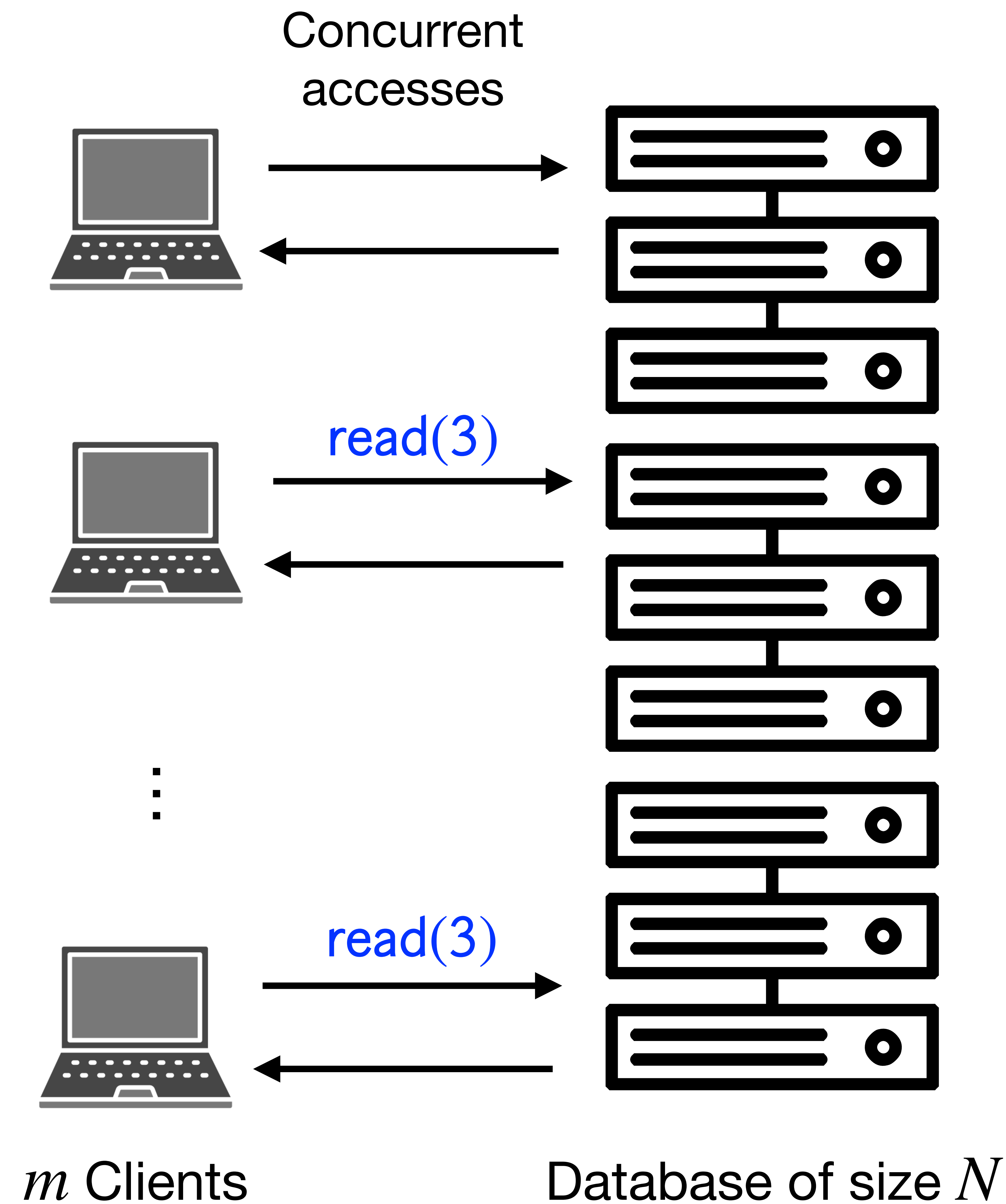
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model: C**oncurrent **R**ead **C**oncurrent **W**rite

CRCW Parallel RAM Model



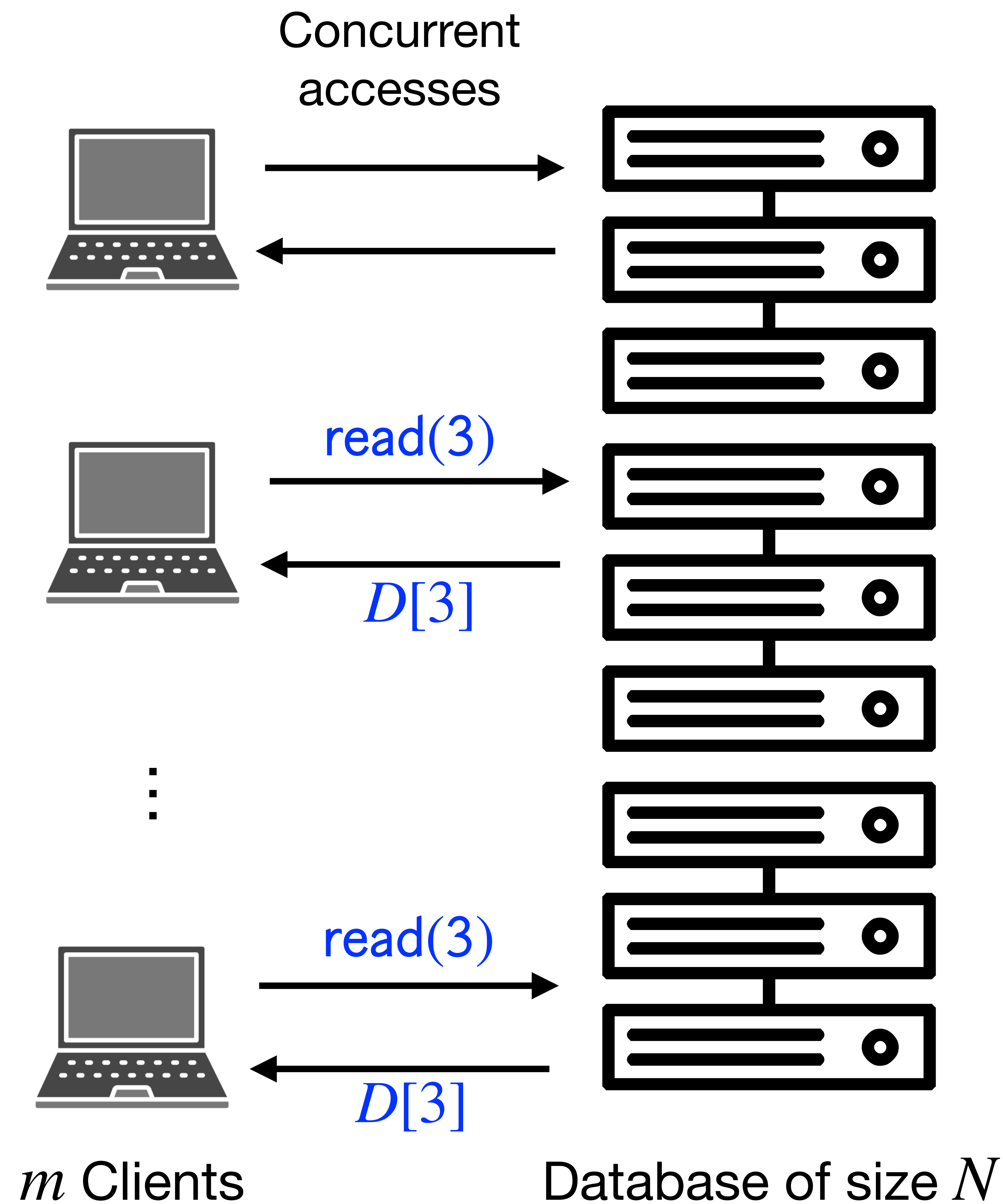
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** Concurrent Read Concurrent Write
  - **Concurrent reads** to any location are allowed.

CRCW Parallel RAM Model



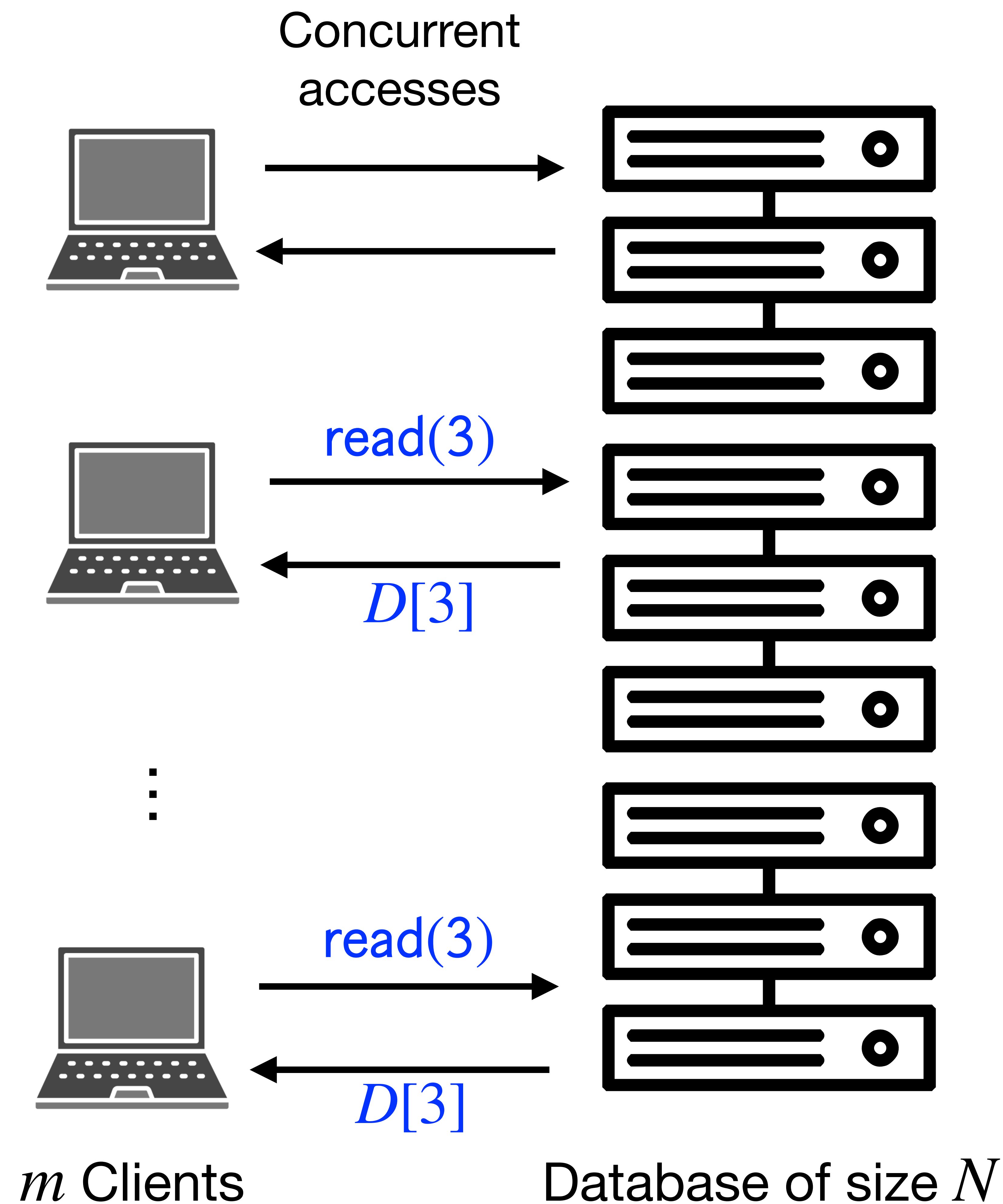
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** **C**oncurrent **R**ead **C**oncurrent **W**rite
  - **Concurrent reads** to any location are allowed.

CRCW Parallel RAM Model



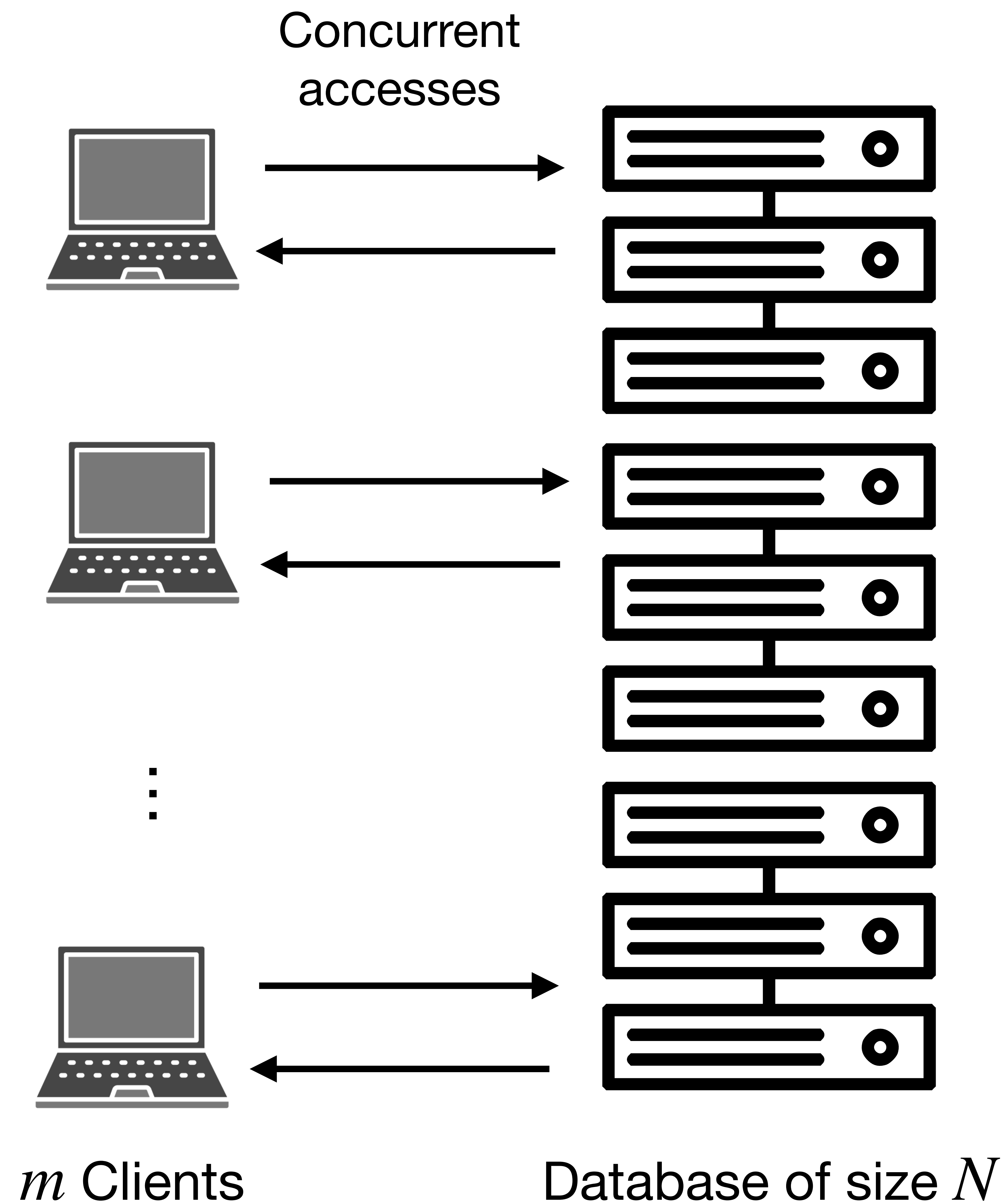
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model: C**oncurrent **R**ead **C**oncurrent **W**rite
  - **Concurrent reads** to any location are allowed.

CRCW Parallel RAM Model



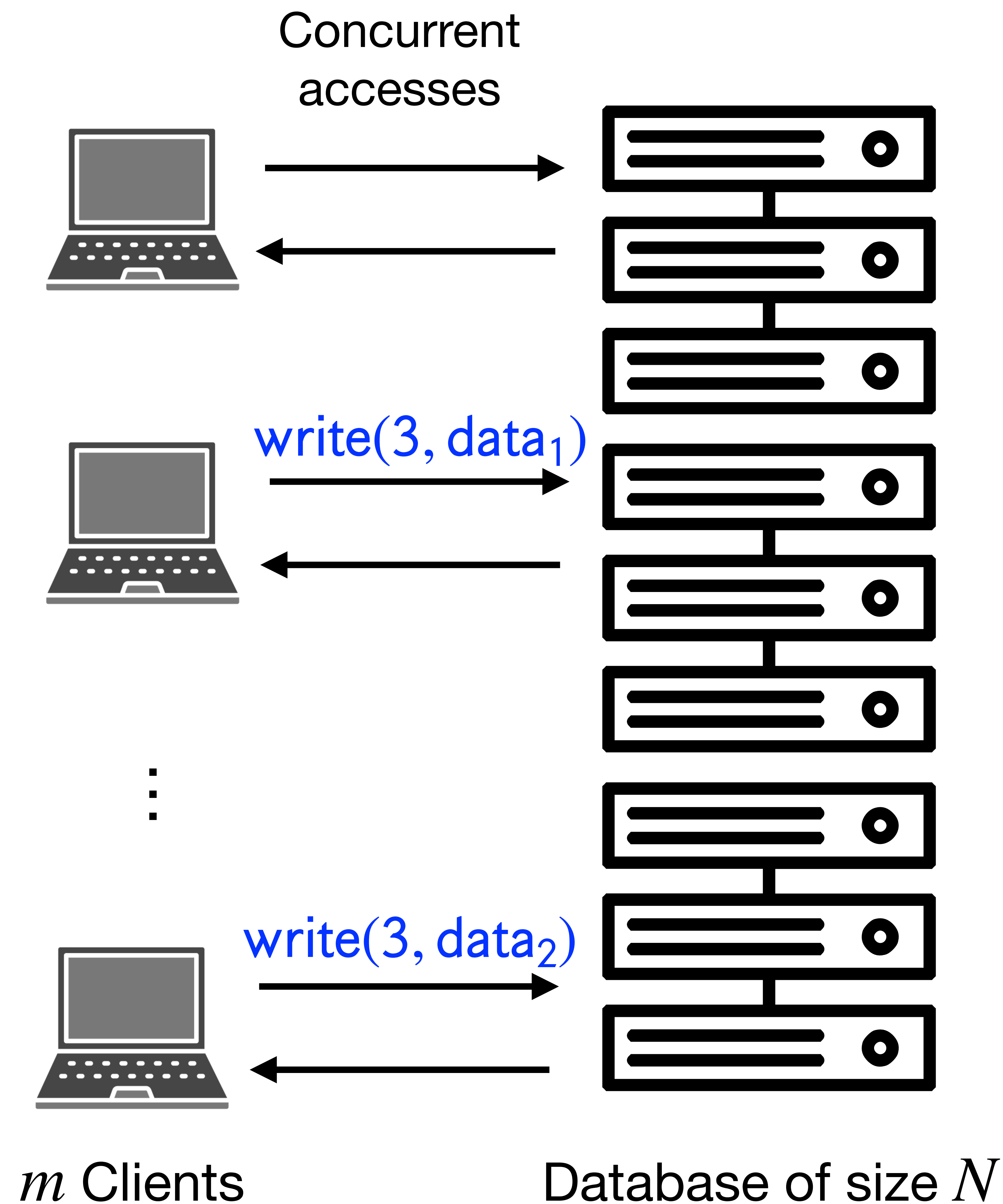
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** <u>**C**</u>oncurrent <u>**R**</u>ead <u>**C**</u>oncurrent <u>**W**</u>rite
  - **Concurrent reads** to any location are allowed.
  - **Concurrent writes** to any location are <u>*arbitrarily*</u> <u>tie-broken</u>.

CRCW Parallel RAM Model



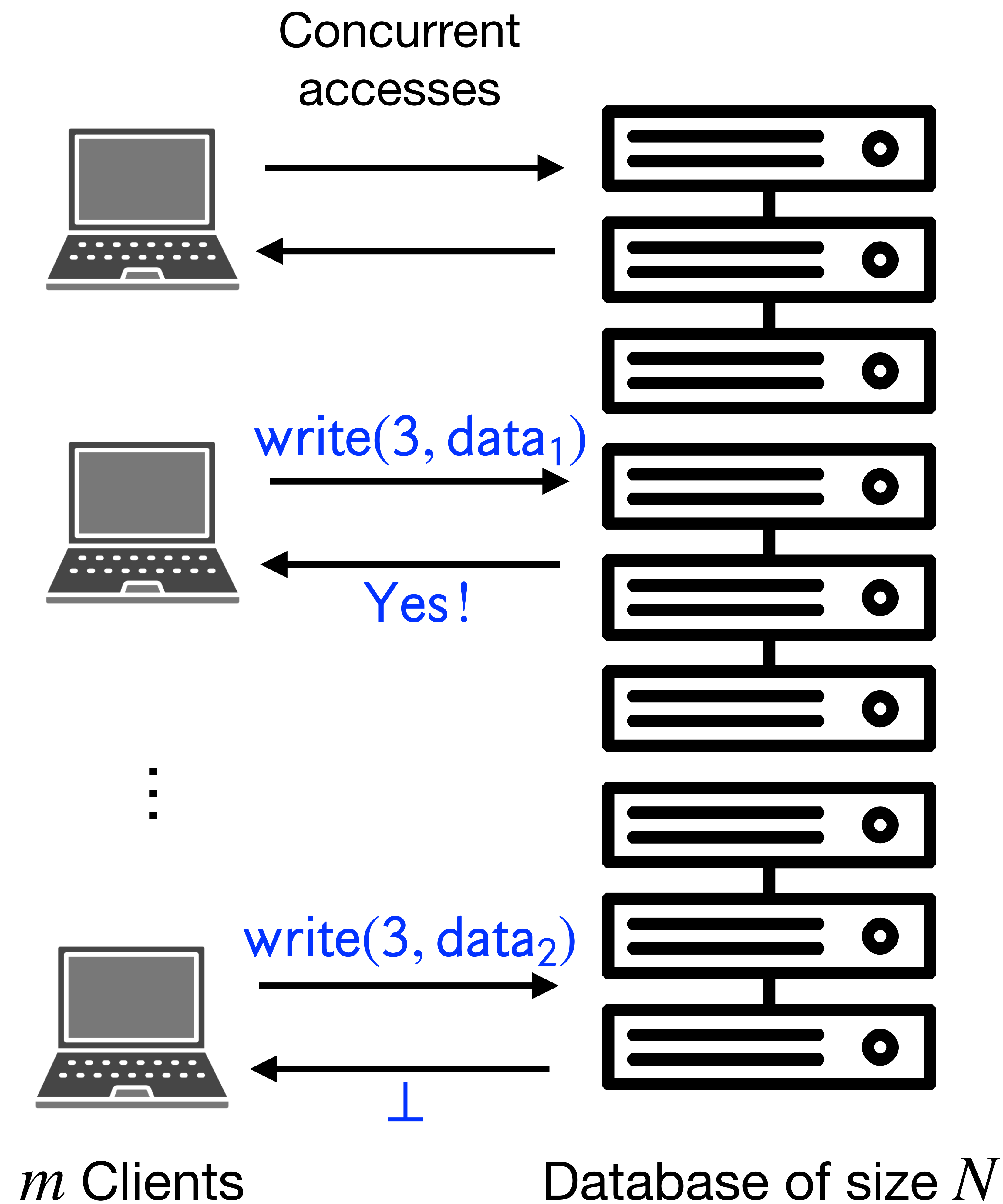
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** **C**oncurrent **R**ead **C**oncurrent **W**rite
  - **Concurrent reads** to any location are allowed.
  - **Concurrent writes** to any location are *arbitrarily* tie-broken.

CRCW Parallel RAM Model



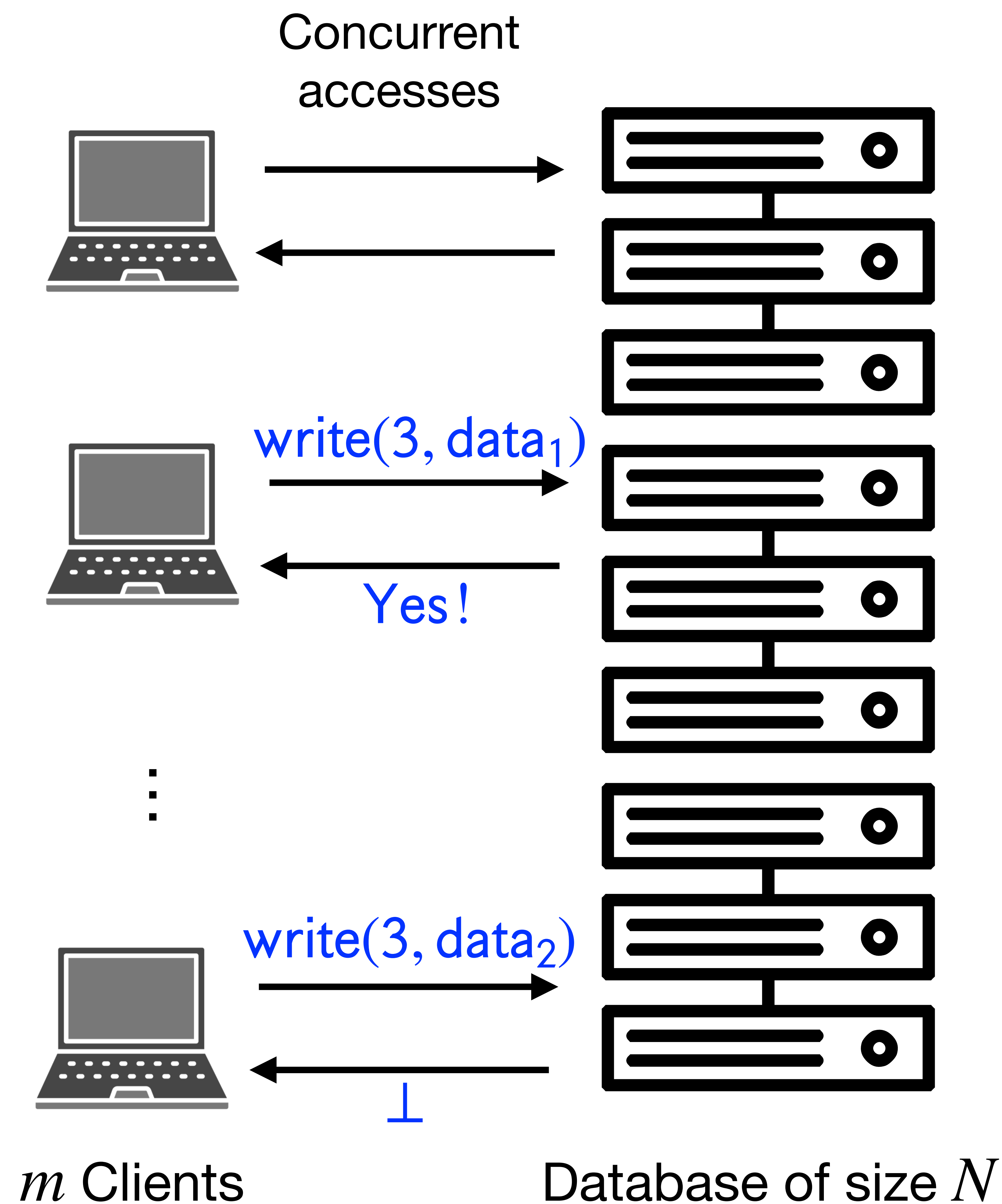
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** <u>**C**</u>oncurrent <u>**R**</u>ead <u>**C**</u>oncurrent <u>**W**</u>rite
  - **Concurrent reads** to any location are allowed.
  - **Concurrent writes** to any location are <u>*arbitrarily*</u> <u>tie-broken</u>.

CRCW Parallel RAM Model



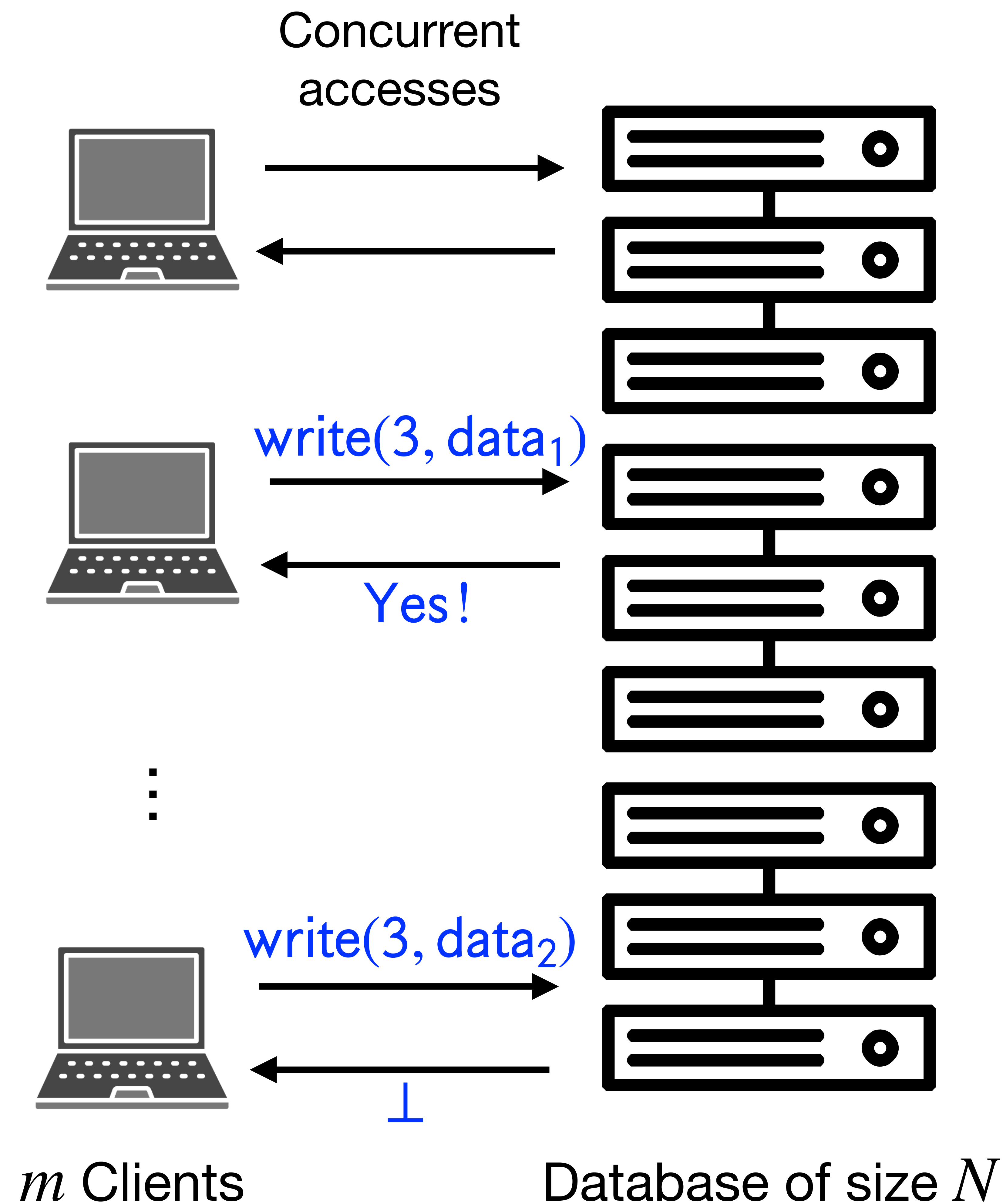
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** Concurrent Read Concurrent Write
  - **Concurrent reads** to any location are allowed.
  - **Concurrent writes** to any location are *arbitrarily* tie-broken.

CRCW Parallel RAM Model



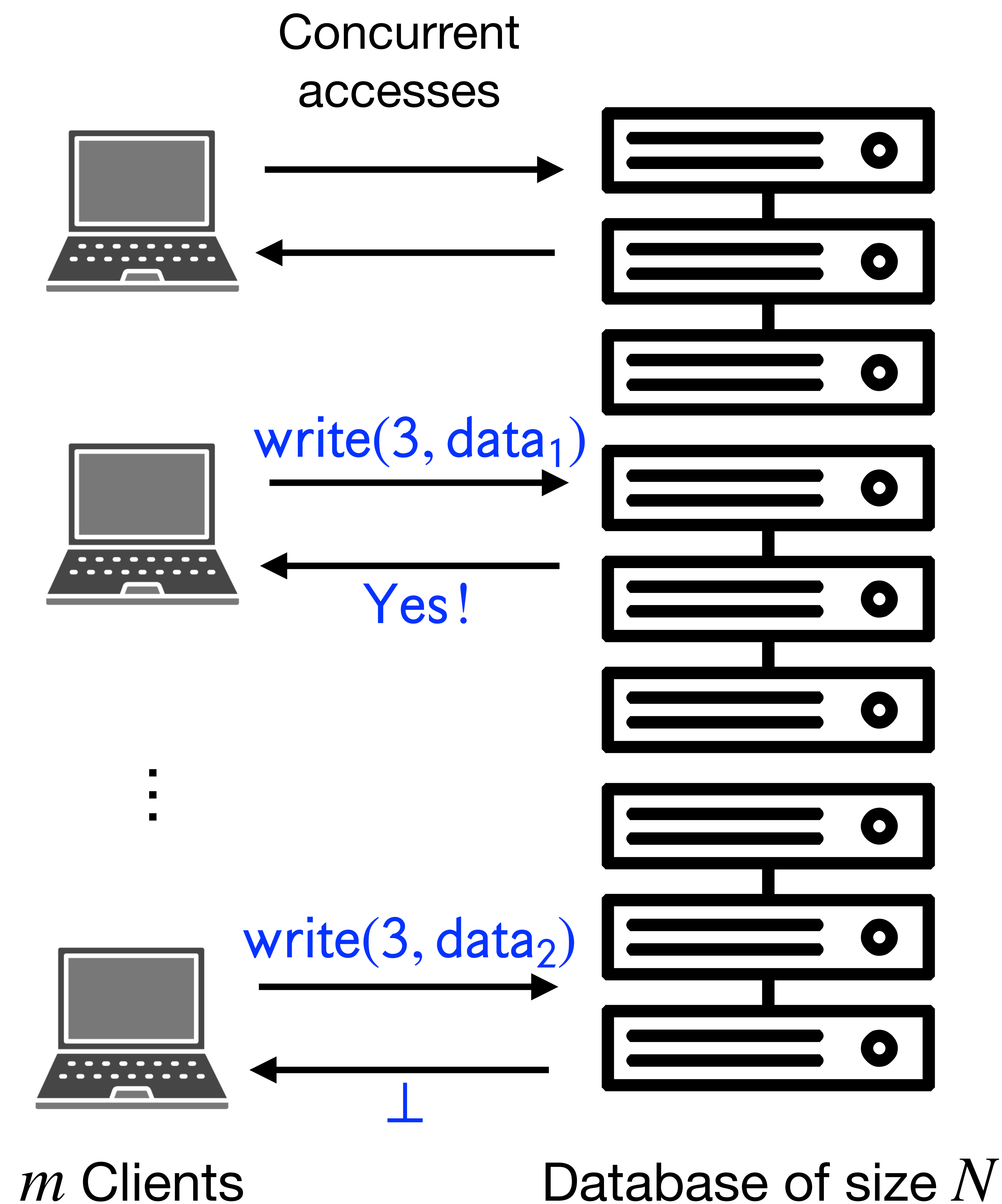
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** **C**oncurrent **R**ead **C**oncurrent **W**rite
  - **Concurrent reads** to any location are allowed.
  - **Concurrent writes** to any location are *arbitrarily* tie-broken.
  - Note: Not every client has to perform an operation.

CRCW Parallel RAM Model



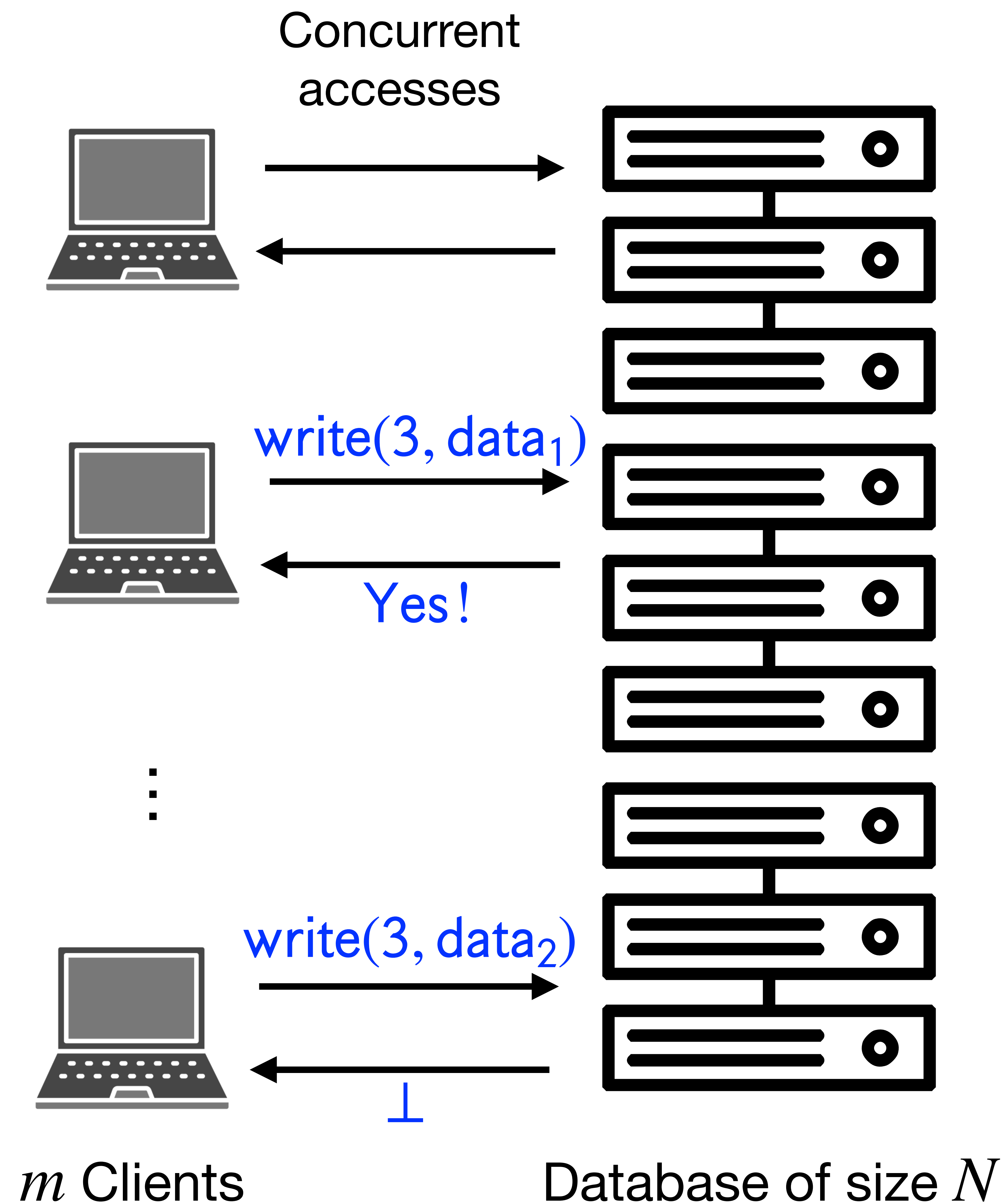
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** Concurrent Read Concurrent Write
  - **Concurrent reads** to any location are allowed.
  - **Concurrent writes** to any location are *arbitrarily* tie-broken.
  - Note: Not every client has to perform an operation.
- **Two efficiency metrics for PRAM algorithms:**

CRCW Parallel RAM Model



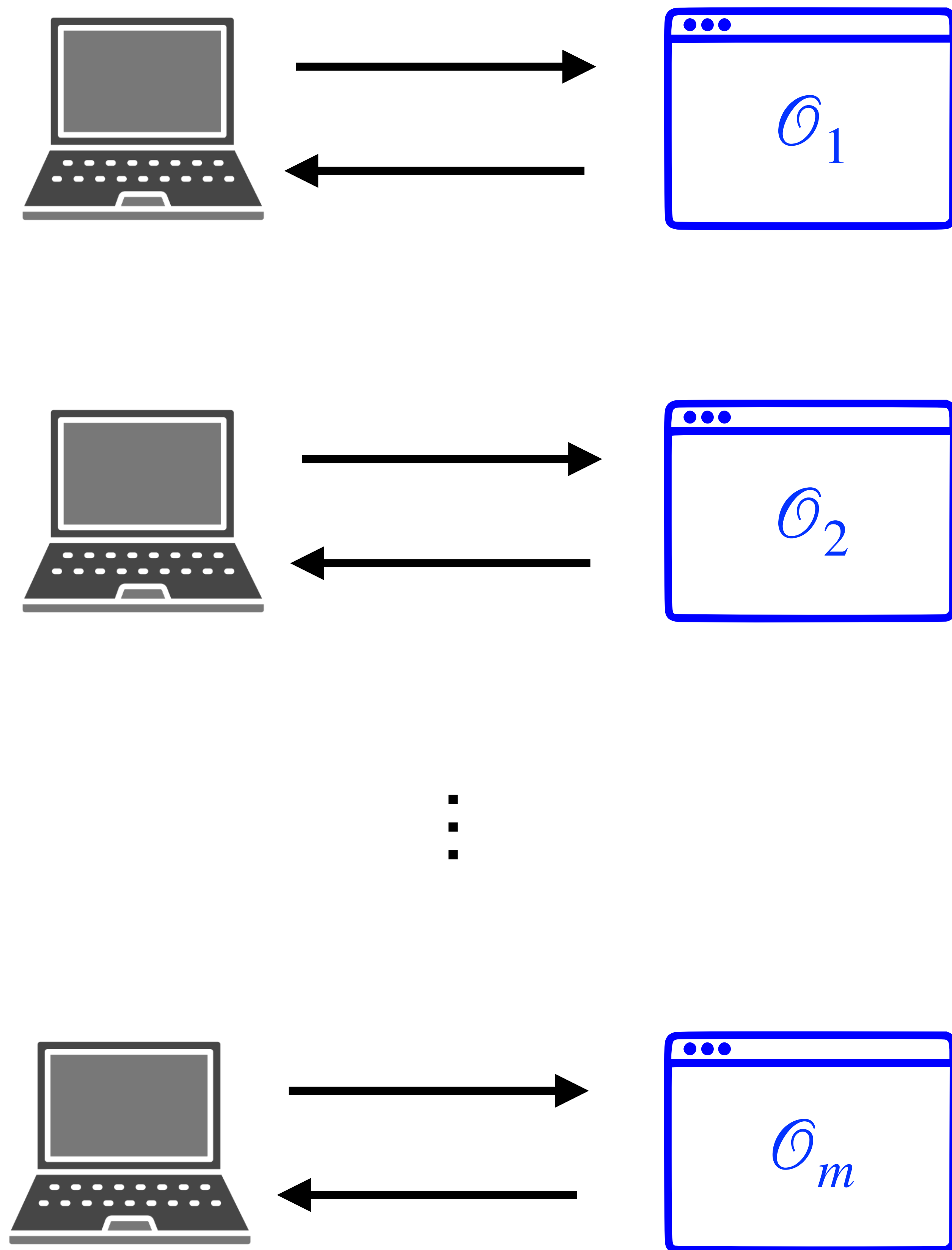
- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** Concurrent Read Concurrent Write
  - **Concurrent reads** to any location are allowed.
  - **Concurrent writes** to any location are *arbitrarily* tie-broken.
  - Note: Not every client has to perform an operation.
- **Two efficiency metrics for PRAM algorithms:**
  - Work: Number of read/write operations

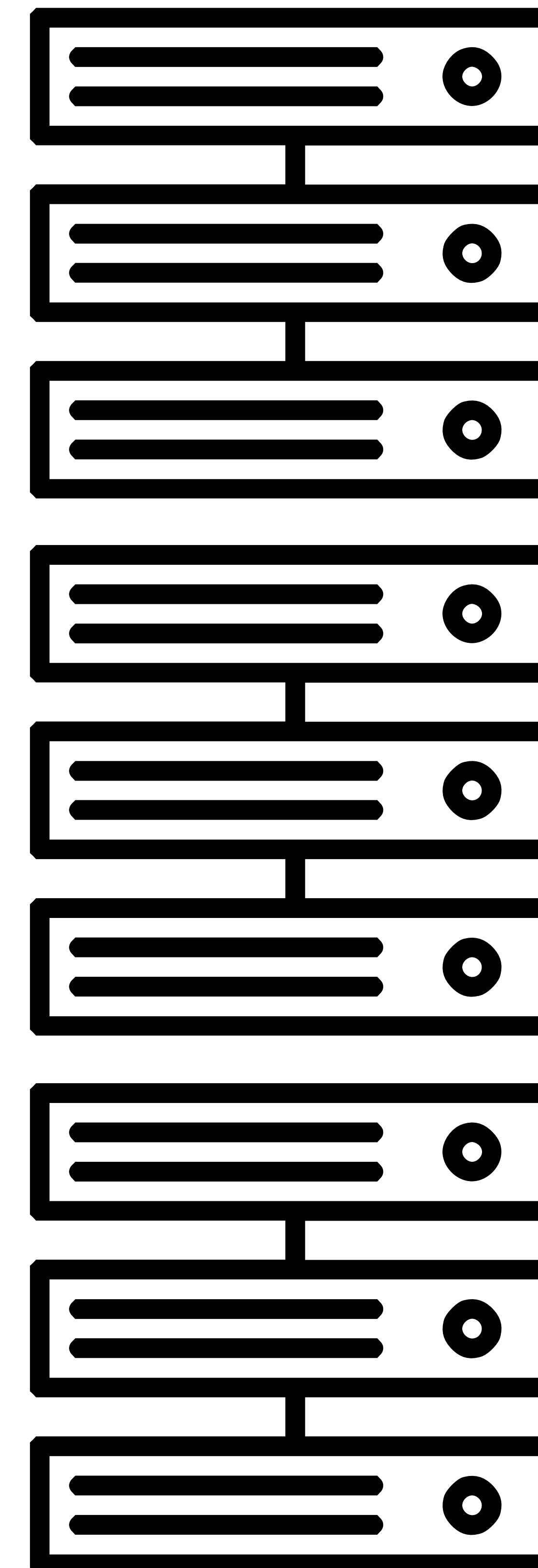
CRCW Parallel RAM Model



- **PRAM:** Multiple CPUs accessing shared memory
- **CRCW Model:** Concurrent Read Concurrent Write
  - **Concurrent reads** to any location are allowed.
  - **Concurrent writes** to any location are *arbitrarily* tie-broken.
  - Note: Not every client has to perform an operation.
- **Two efficiency metrics for PRAM algorithms:**
  - Work: Number of read/write operations
  - Depth: Number of parallel steps

Idea: Compose ORAM and MC



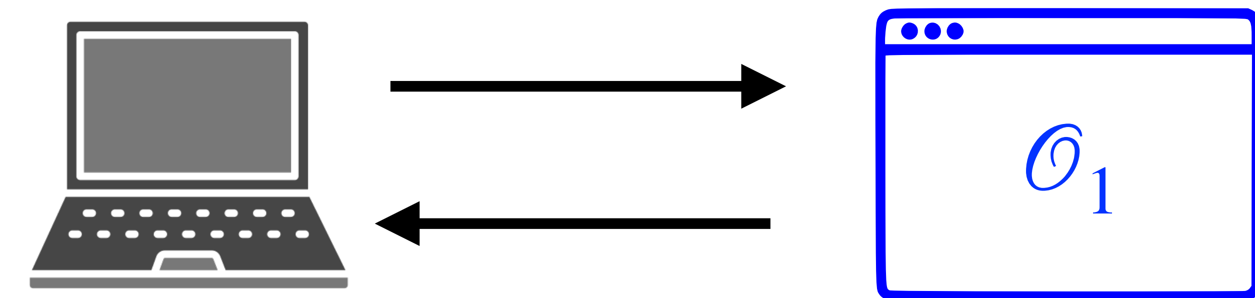


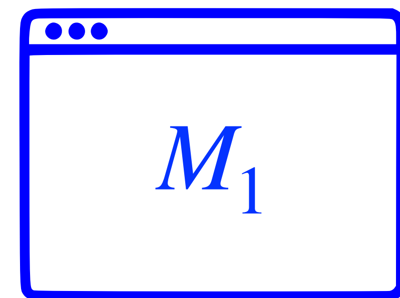
OPRAM

Idea: Compose ORAM and MC

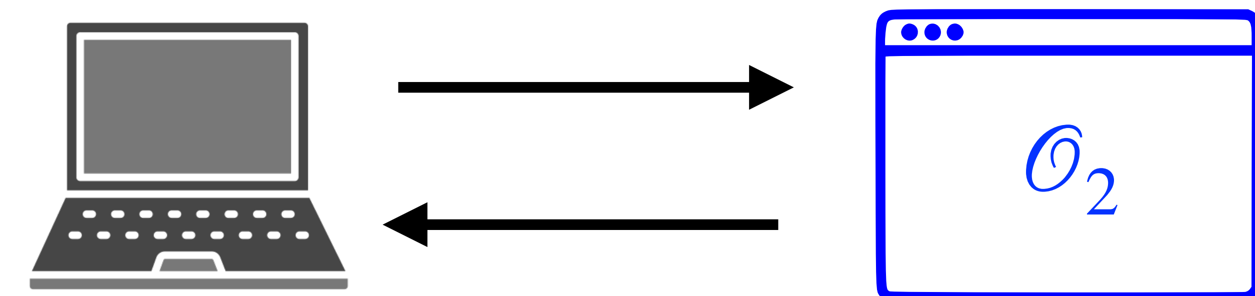
m Clients

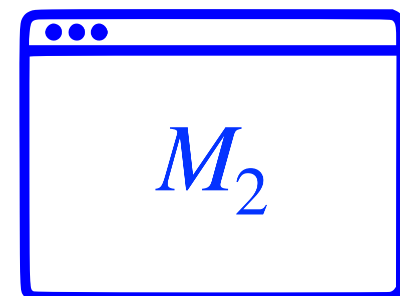
Database of size N





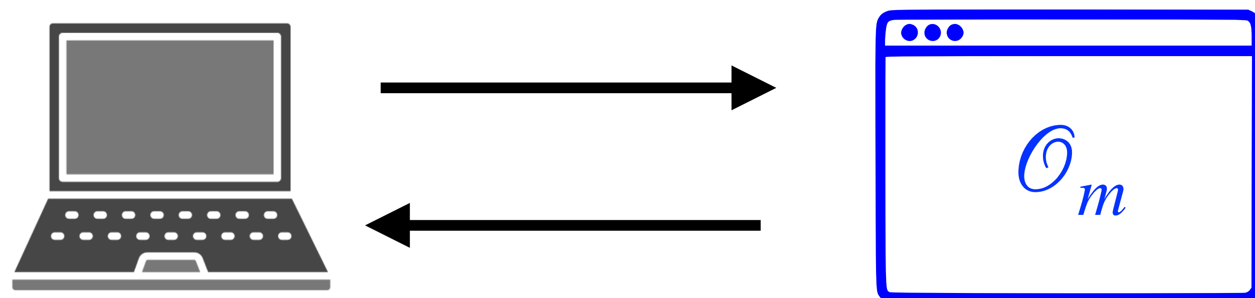
M_1

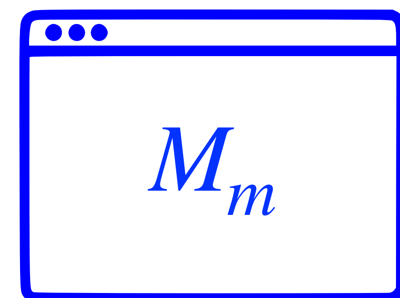




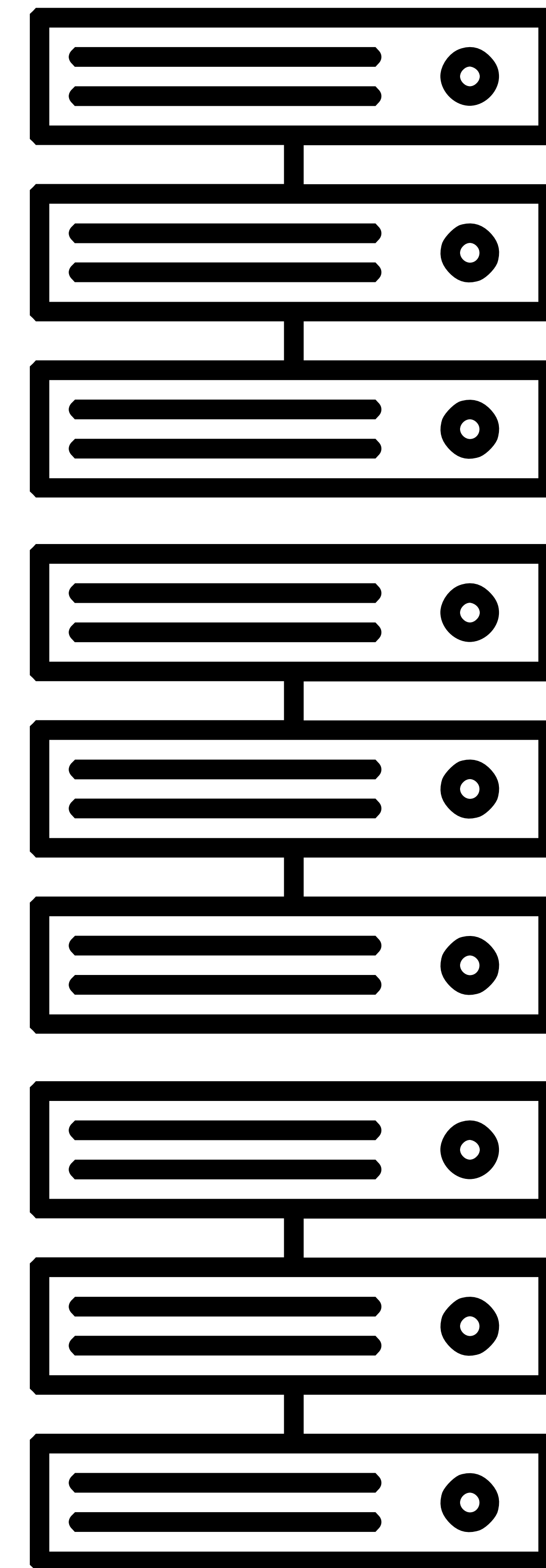
M_2

⋮





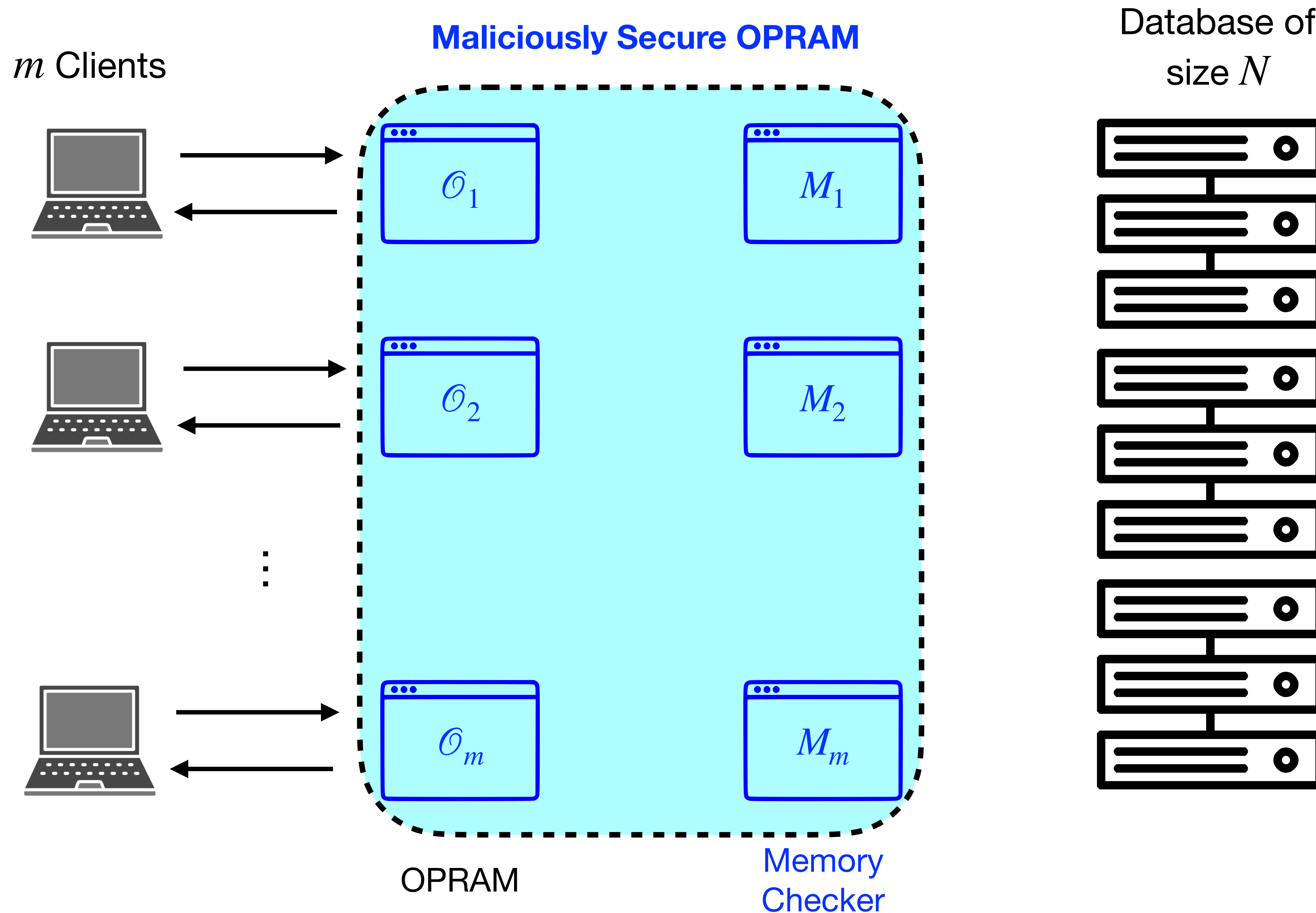
M_m



OPRAM

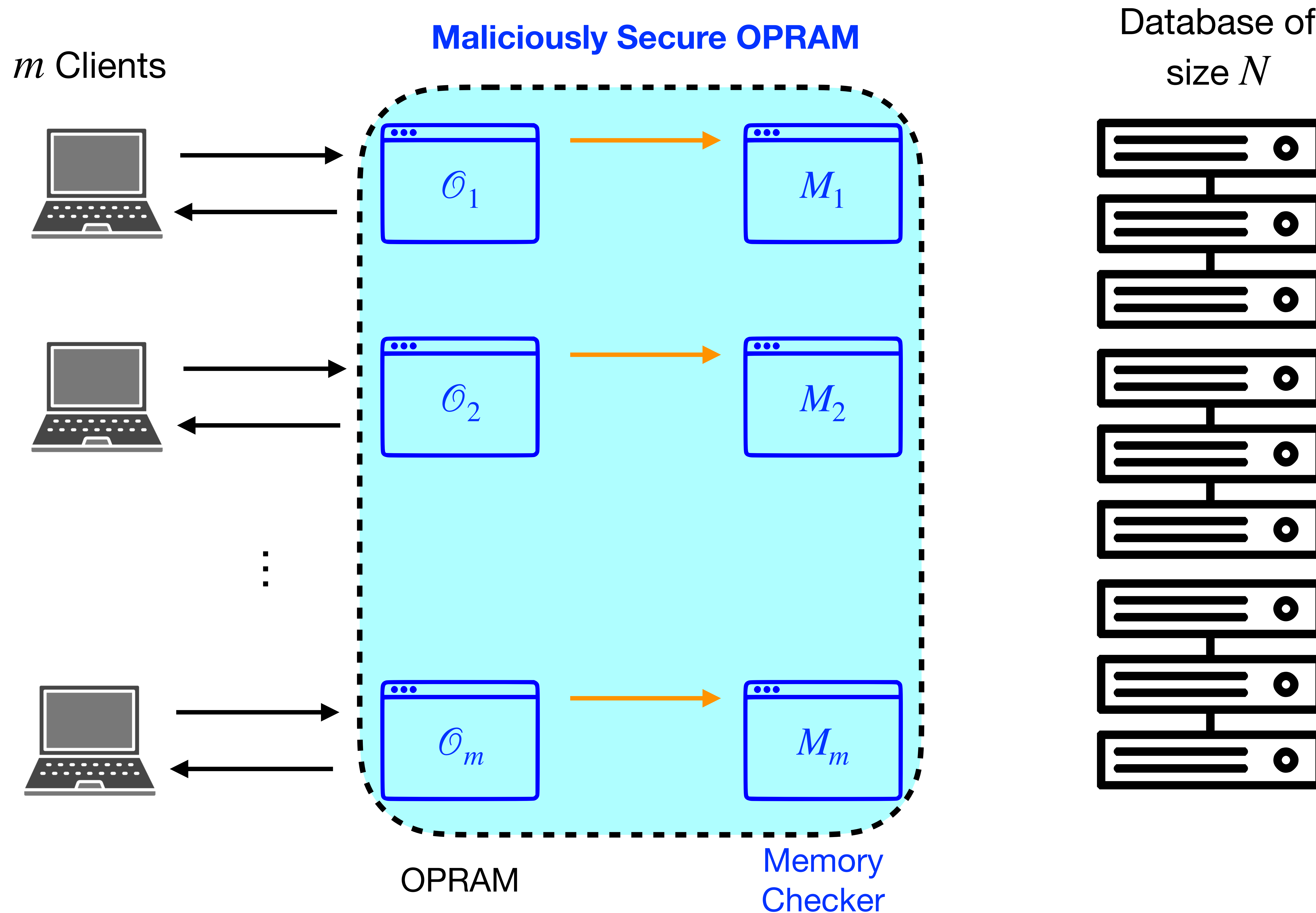
Memory Checker

Idea: Compose ORAM and MC

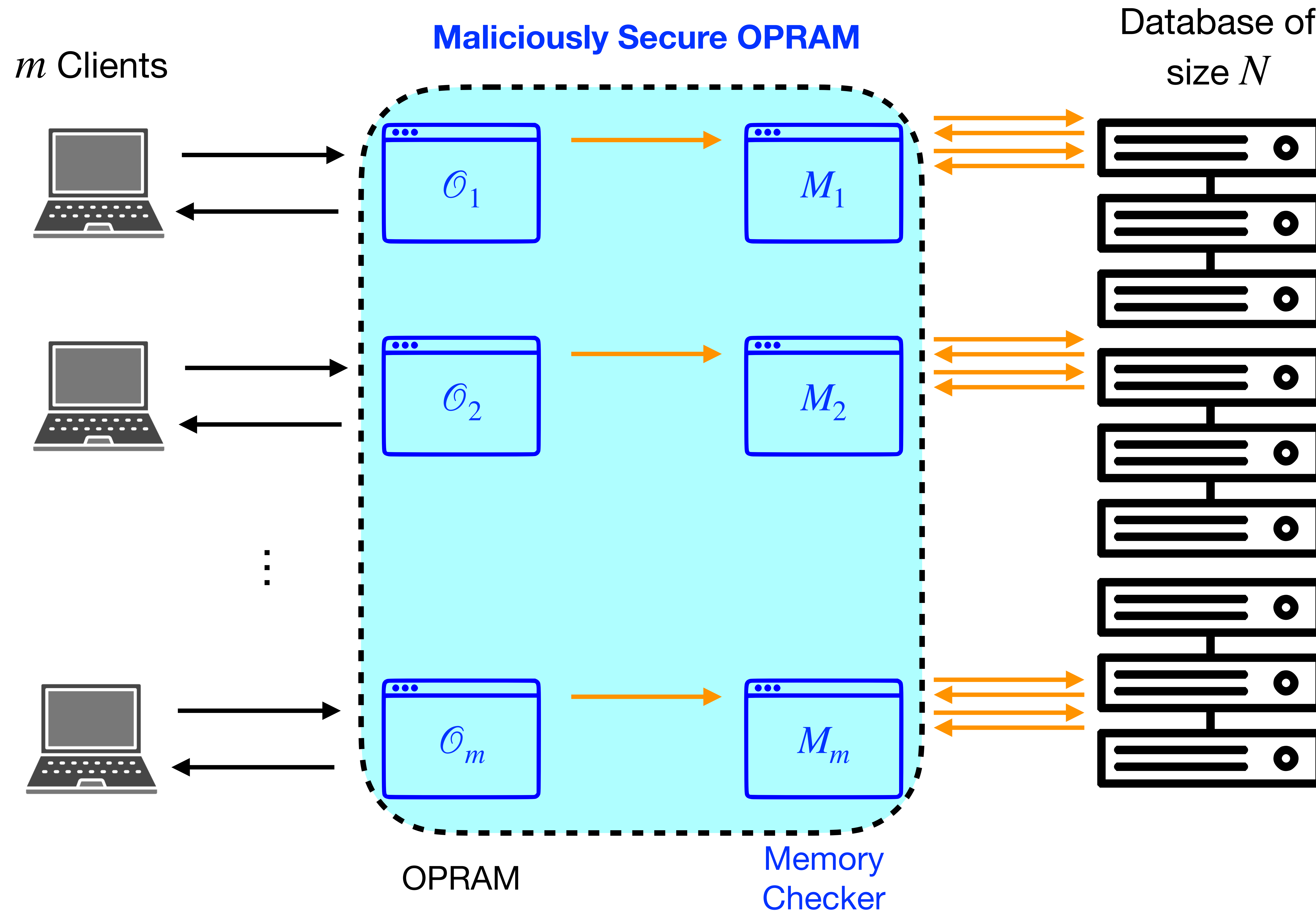


Database of size N

Idea: Compose ORAM and MC

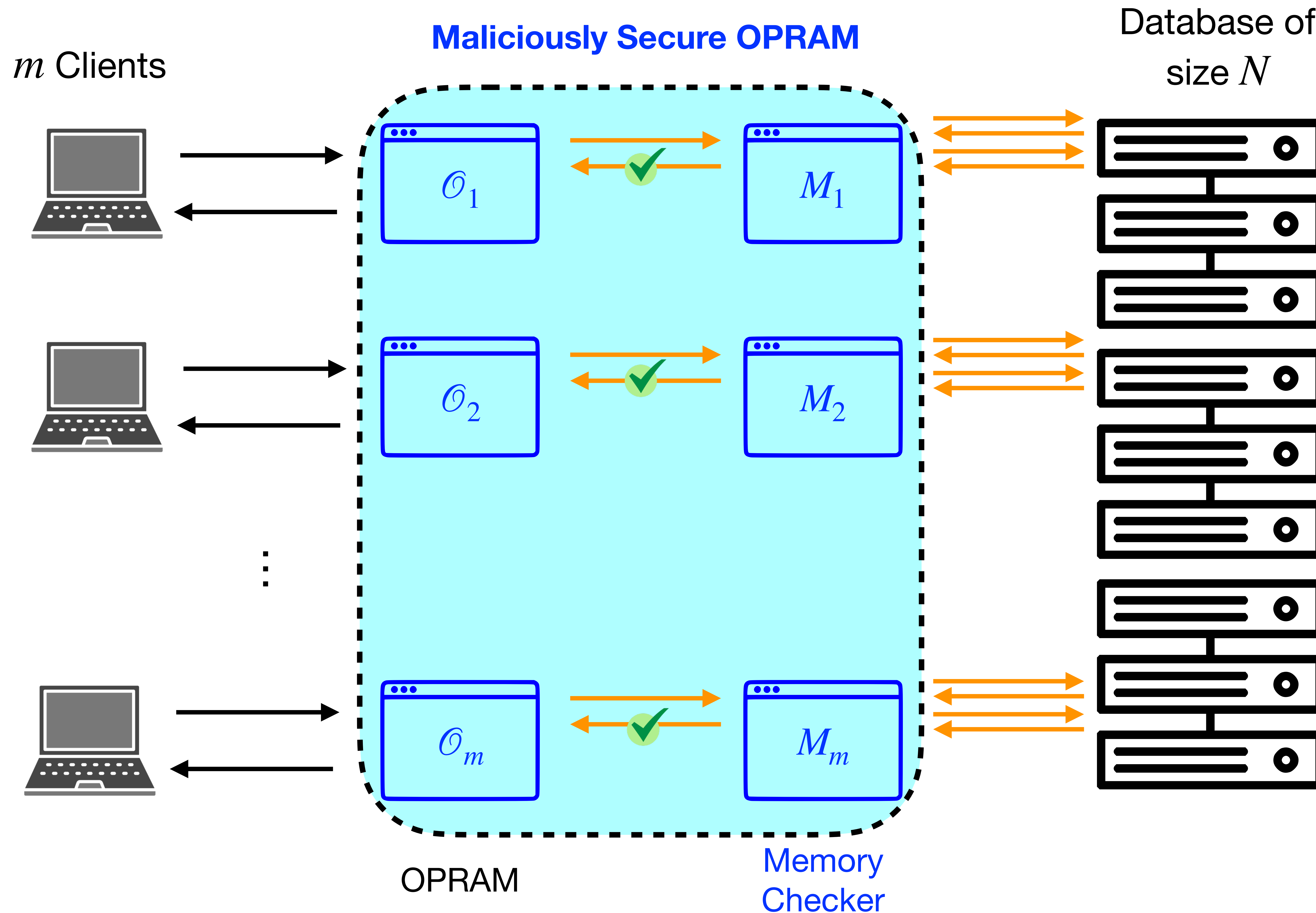


Idea: Compose ORAM and MC



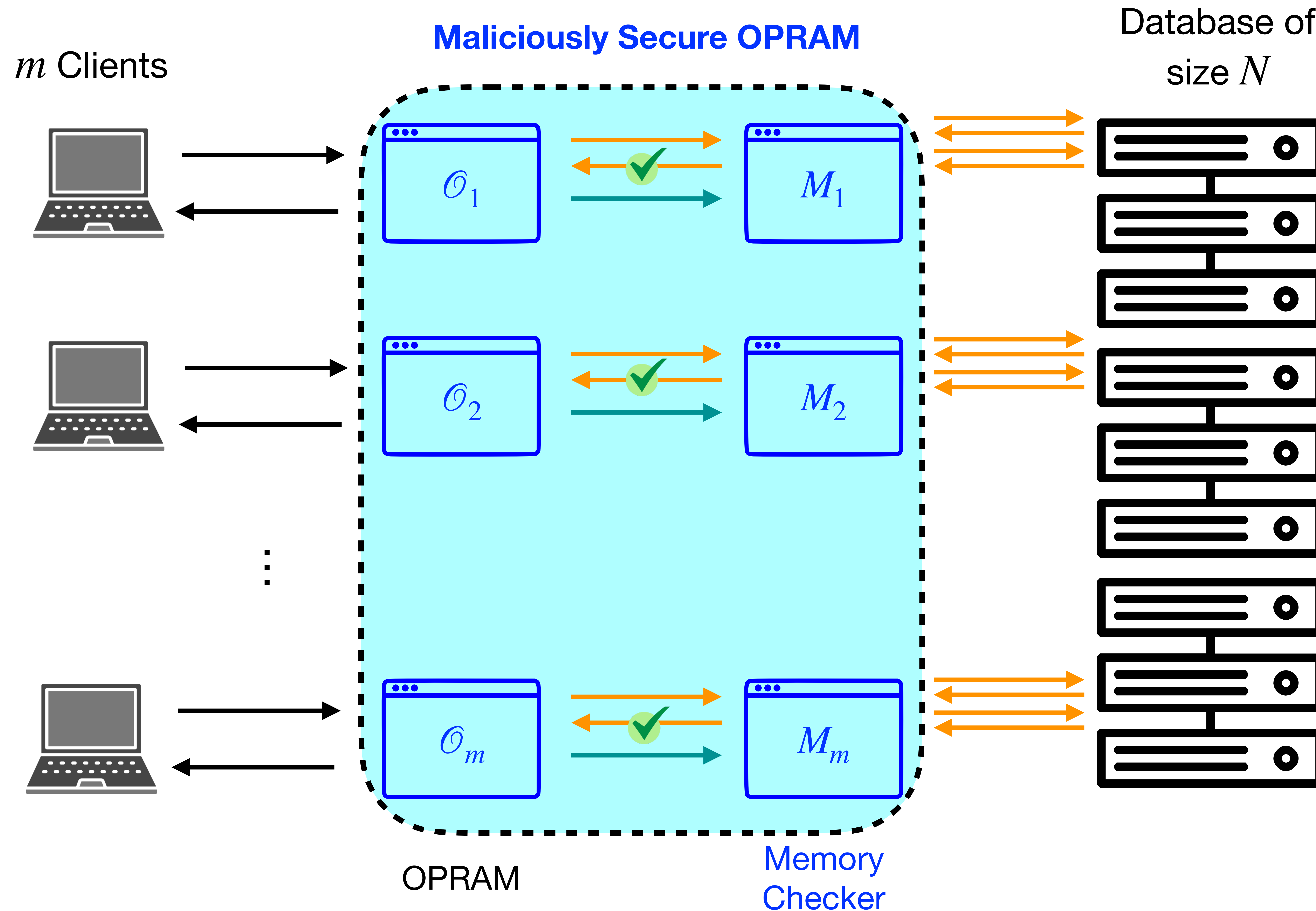
Database of size N

Idea: Compose ORAM and MC

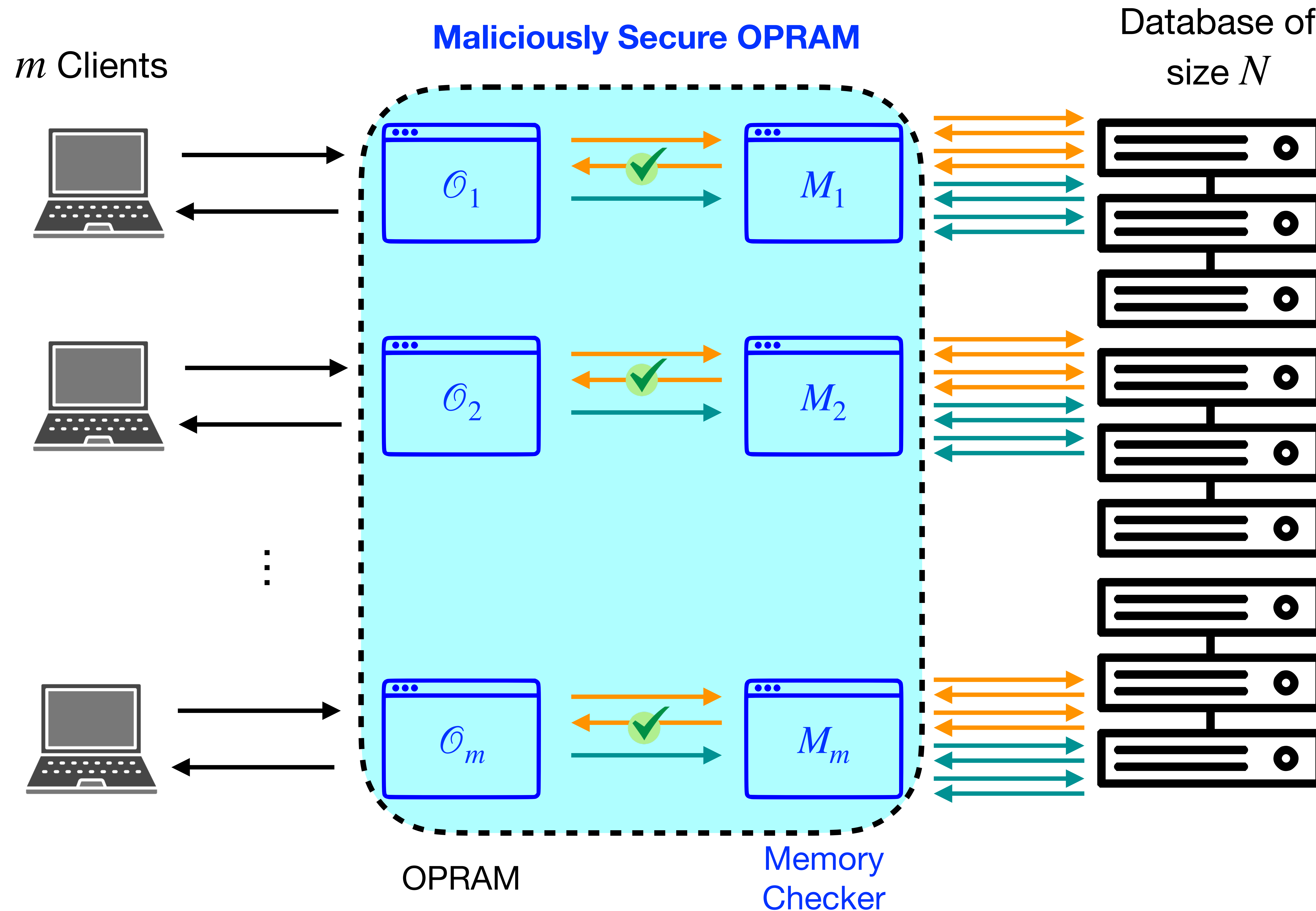


Database of size N

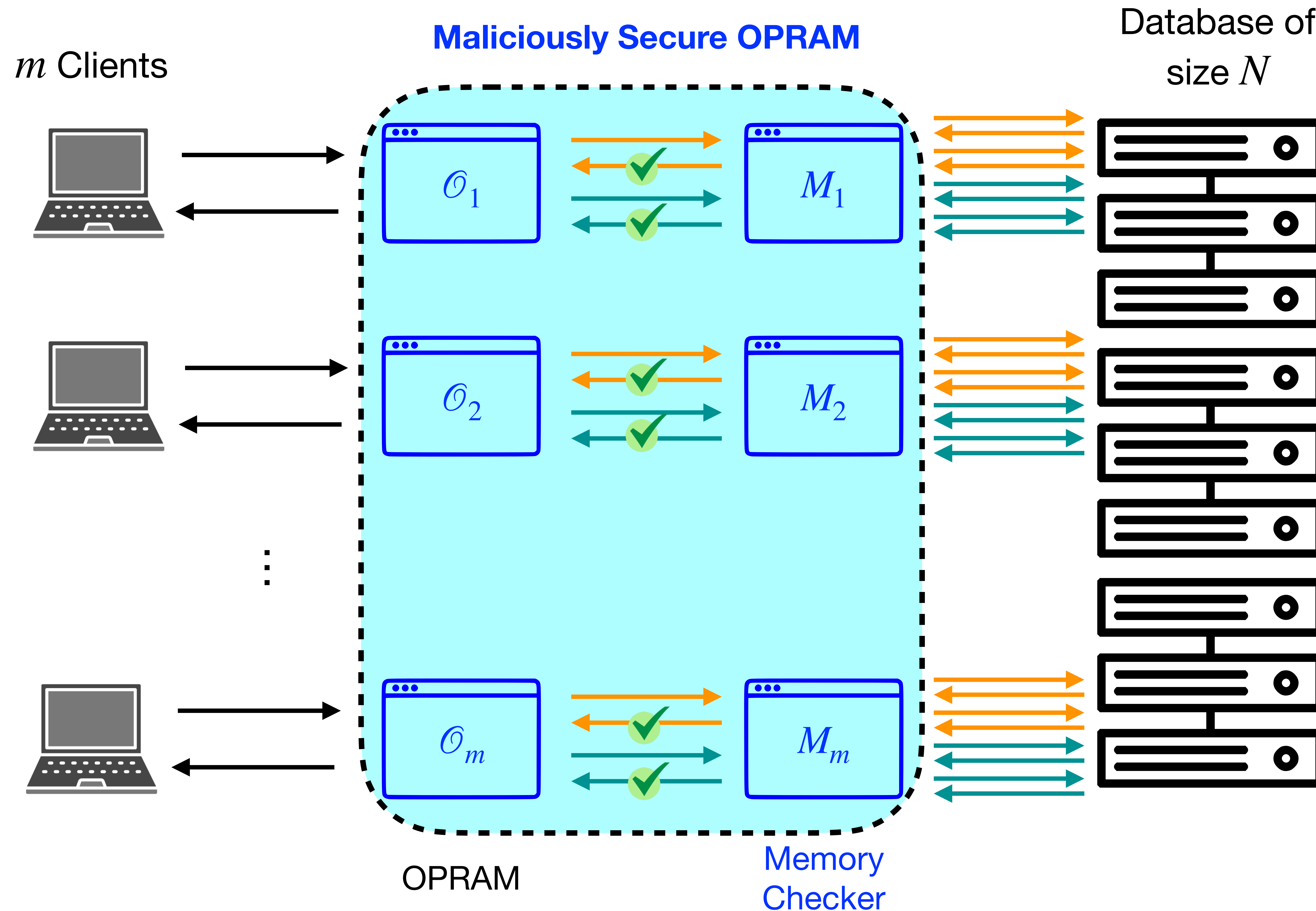
Idea: Compose ORAM and MC



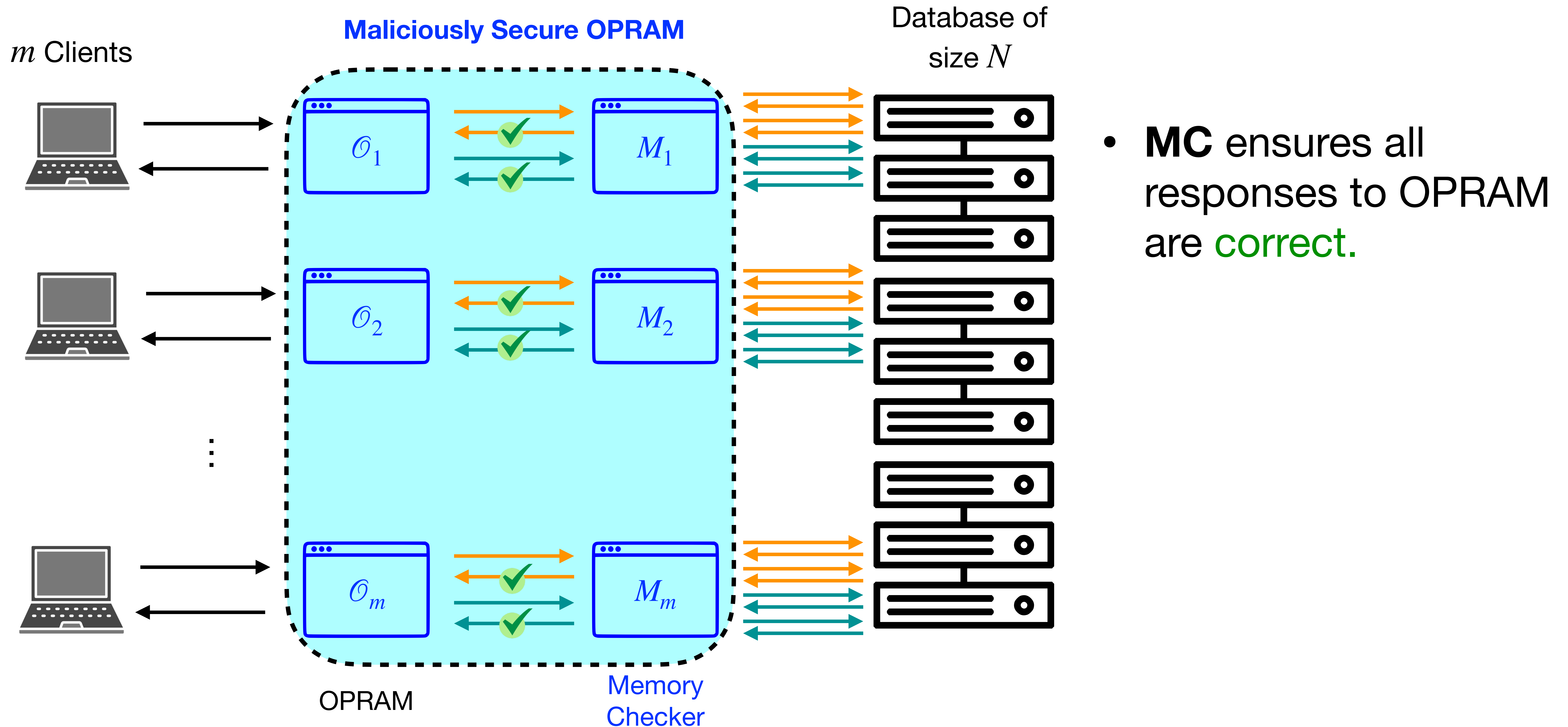
Idea: Compose ORAM and MC



Idea: Compose ORAM and MC

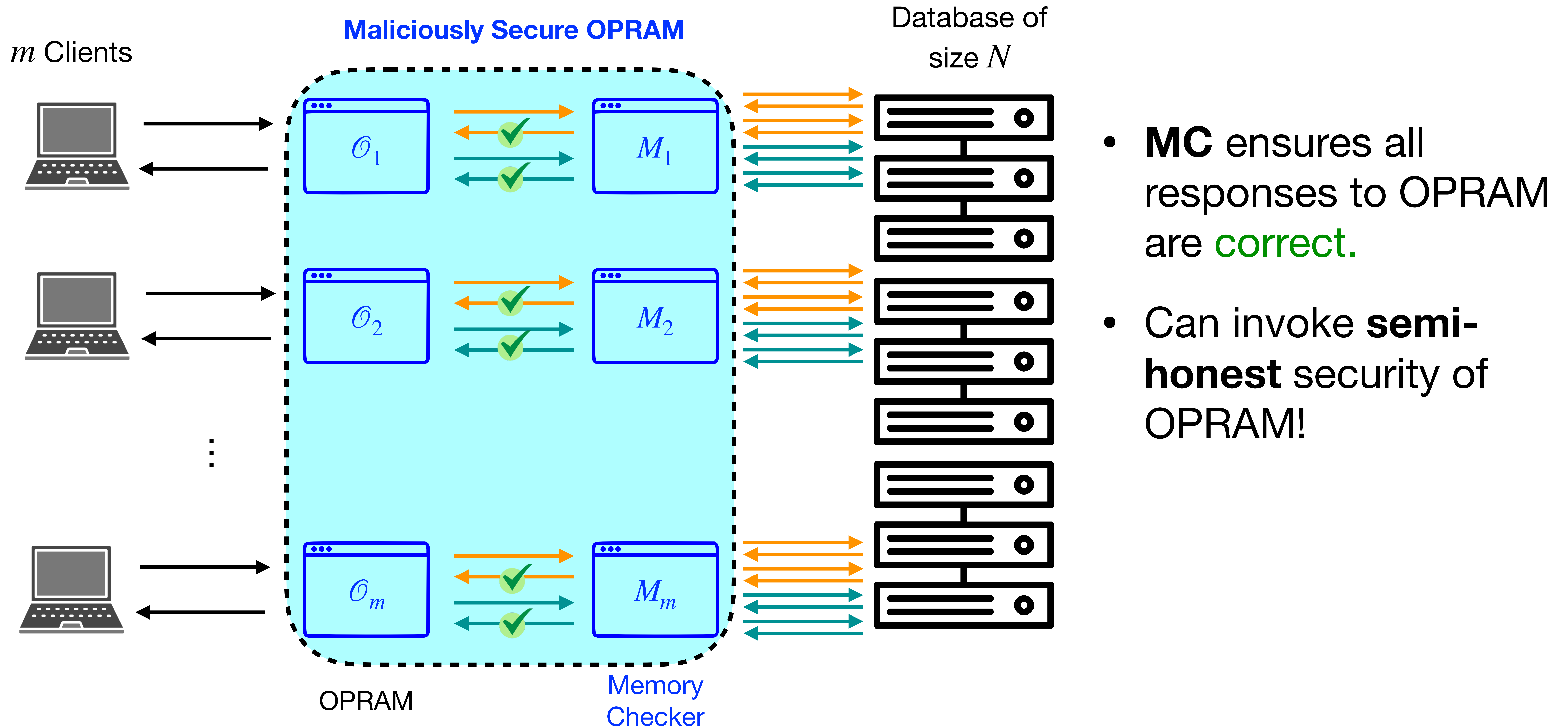


Idea: Compose ORAM and MC



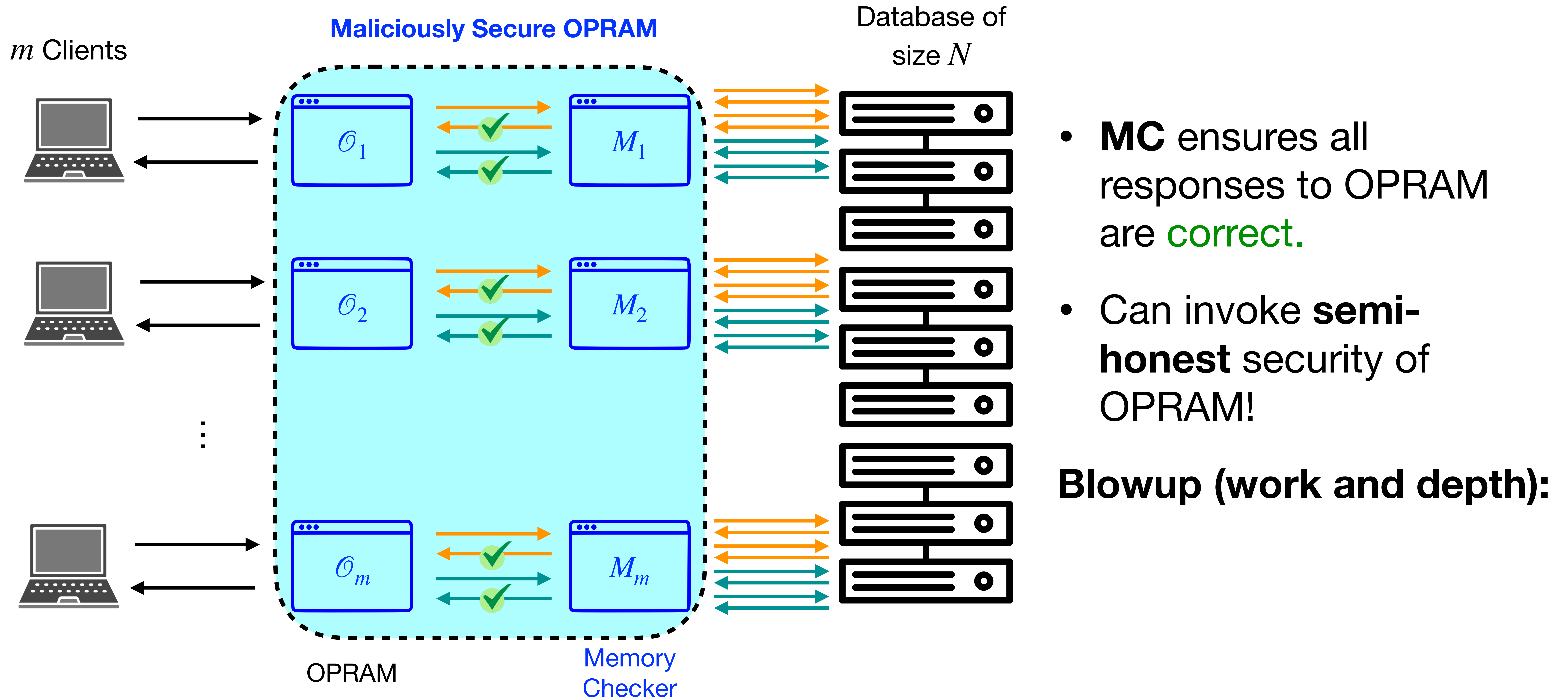
- **MC** ensures all responses to OPRAM are correct.

Idea: Compose ORAM and MC



- **MC** ensures all responses to OPRAM are correct.
- Can invoke **semi-honest** security of OPRAM!

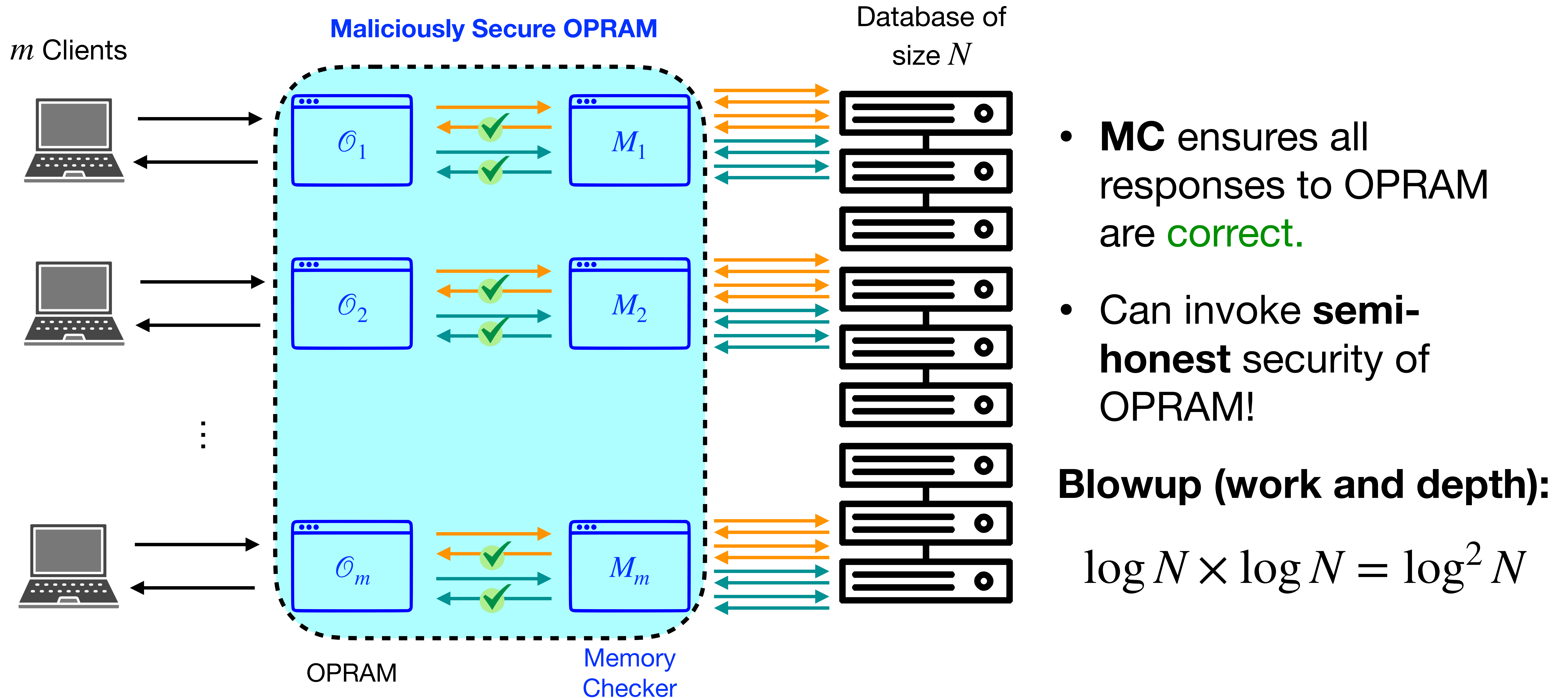
Idea: Compose ORAM and MC



- **MC** ensures all responses to OPRAM are correct.
- Can invoke **semi-honest** security of OPRAM!

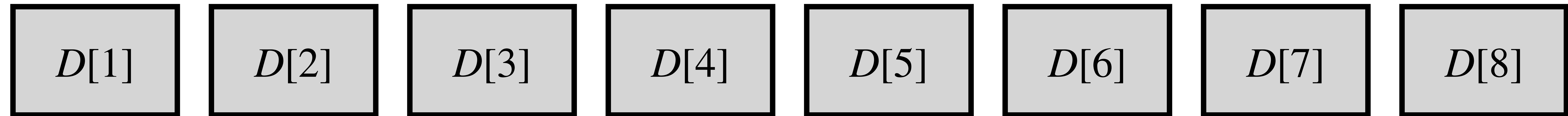
Blowup (work and depth):

Idea: Compose ORAM and MC



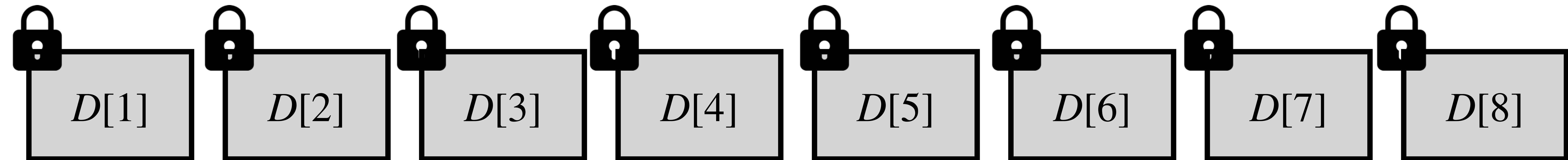
Offline-Checking for RAMs

[Blum, Evans, Gemmel, Kannan, Naor ’94], this construction is based on [**M**, Vafa `23]



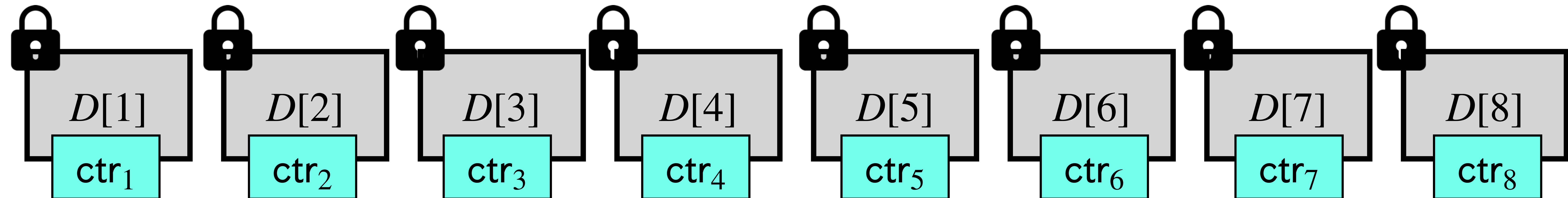
Offline-Checking for RAMs

[Blum, Evans, Gemmel, Kannan, Naor ’94], this construction is based on [**M**, Vafa `23]



Offline-Checking for RAMs

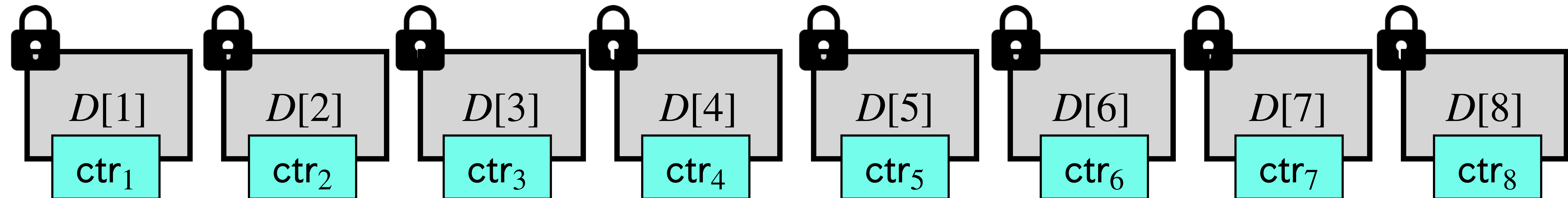
[Blum, Evans, Gemmel, Kannan, Naor ’94], this construction is based on [**M**, Vafa `23]



- For entry $D[i]$, let $\mathsf{ctr}_i =$ number of times location i was ***read or written*** to. Initialise to 0.

Offline-Checking for RAMs

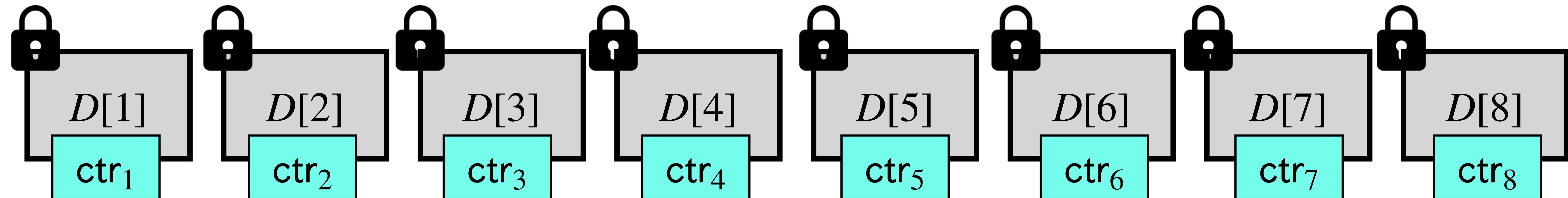
[Blum, Evans, Gemmel, Kannan, Naor ’94], this construction is based on [**M**, Vafa `23]



- For entry $D[i]$, let $\mathsf{ctr}_i =$ number of times location i was ***read or written*** to. Initialise to 0.
- Memory checker local stores a counter T initialised to 0.

Offline-Checking for RAMs

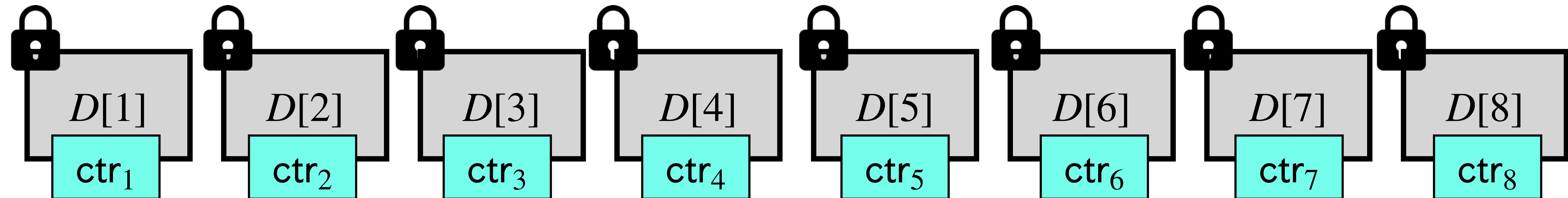
[Blum, Evans, Gemmel, Kannan, Naor ’94], this construction is based on [**M**, Vafa `23]



- For entry $D[i]$, let $\mathsf{ctr}_i =$ number of times location i was ***read or written*** to. Initialise to 0.
- Memory checker local stores a counter T initialised to 0.
- For every access to $D[i]$, increment T locally, and increment ctr_i.

Offline-Checking for RAMs

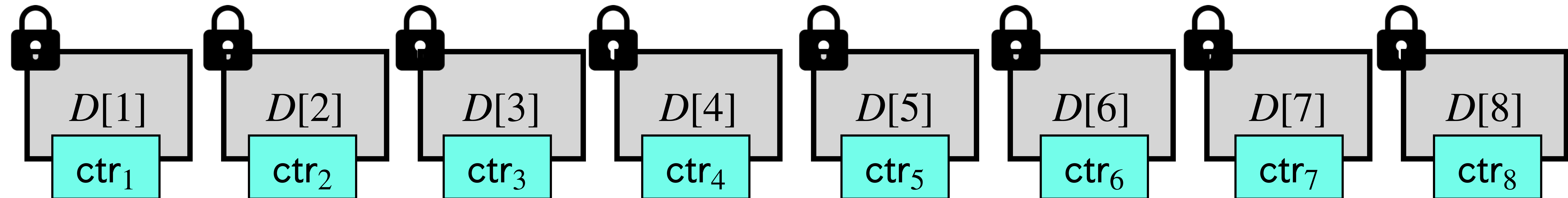
[Blum, Evans, Gemmel, Kannan, Naor '94], this construction is based on [**M**, Vafa `23]



- For entry $D[i]$, let ctr$_i$ $=$ number of times location i was ***read or written*** to. Initialise to 0.
- Memory checker local stores a counter T initialised to 0.
- For every access to $D[i]$, increment T locally, and increment ctr$_i$.
- At the end, the memory checker iterates over the array and verifies $\sum_i \text{ctr}_i = T$.

Offline-Checking for RAMs

[Blum, Evans, Gemmel, Kannan, Naor '94], this construction is based on [**M**, Vafa `23]



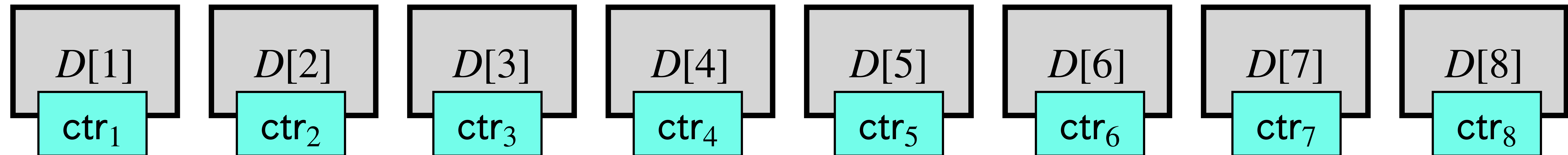
- For entry $D[i]$, let $\mathsf{ctr}_i =$ number of times location i was
- Memory checker local stores a counter T initialised to 0.
- For every access to $D[i]$, increment T locally, and incre
- At the end, the memory checker iterates over the array and verifies $\sum_i \mathsf{ctr}_i = T$.

Idea: Let t_i be the number of times $D[i]$ was actually accessed.

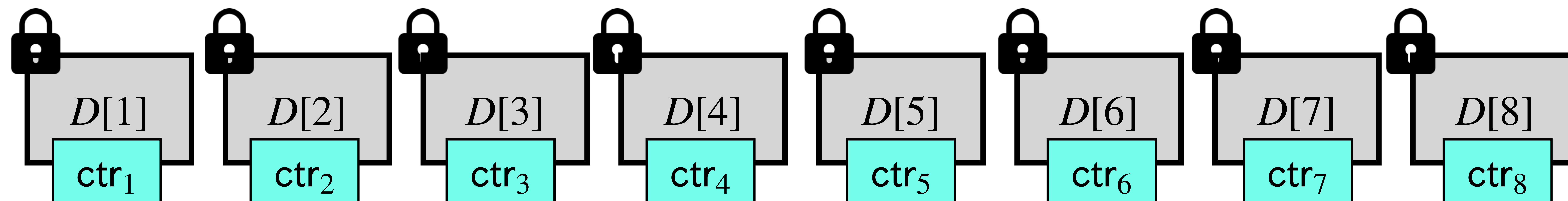
$$\sum_i \mathsf{ctr}_i \leq \sum_i t_i = T$$

where equality holds **iff** there was no replay attack.

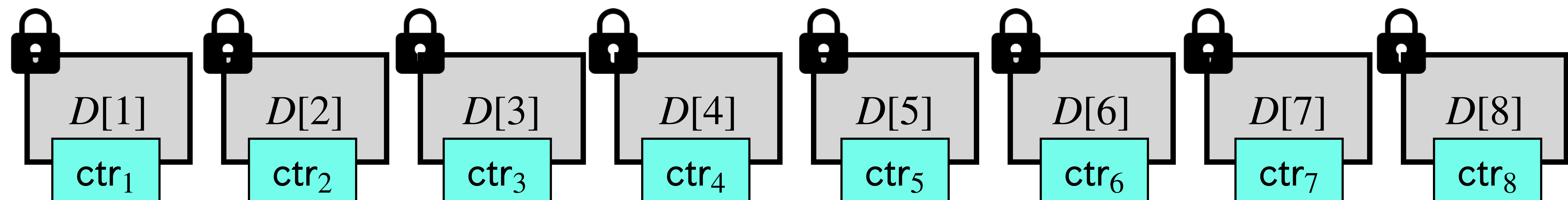
New: Offline-Checking for PRAMs



New: Offline-Checking for PRAMs



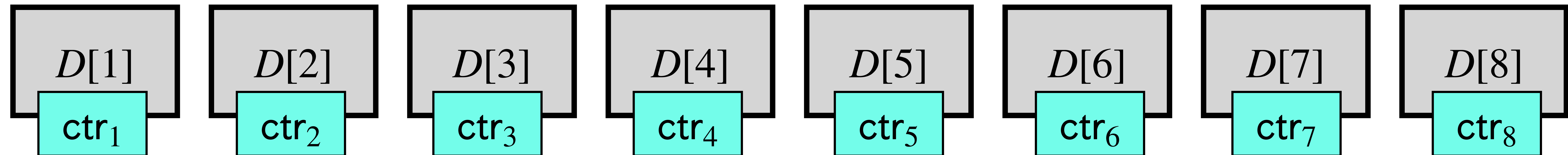
New: Offline-Checking for PRAMs

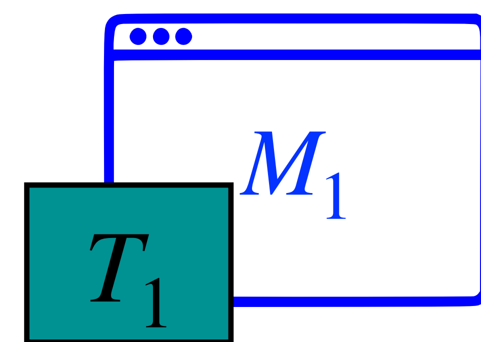


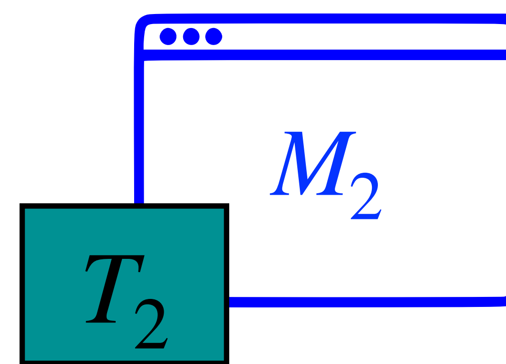
- **As before:** Initialise all entries with $\mathsf{ctr}_i = 0$ and authenticate all entries.

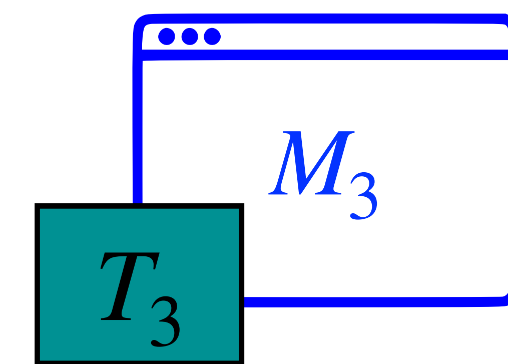
- Each M_i keeps a local count T_i.

New: Offline-Checking for PRAMs

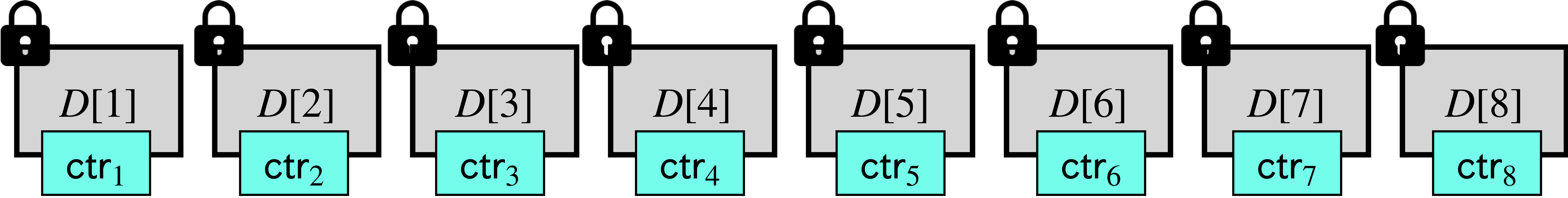


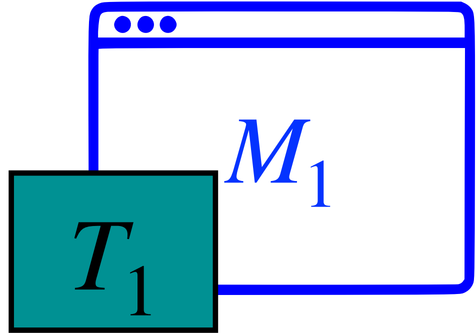


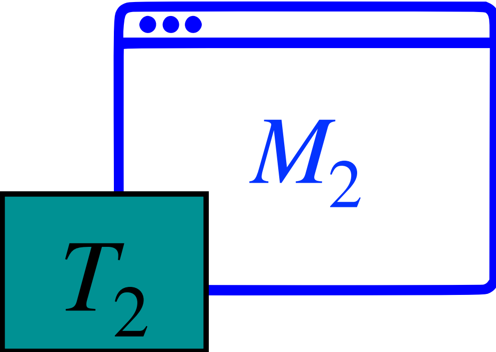


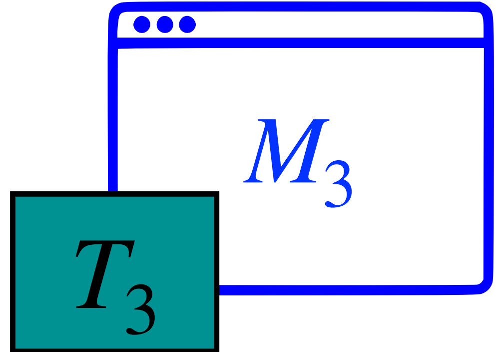


New: Offline-Checking for PRAMs

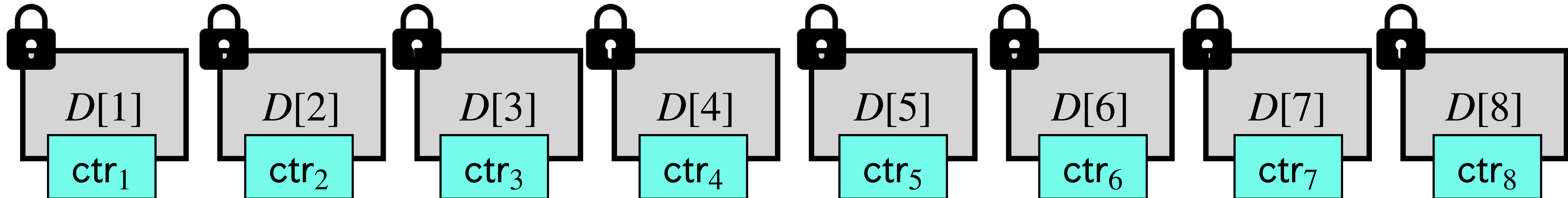








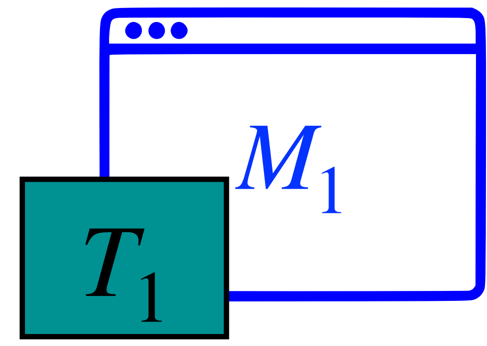
New: Offline-Checking for PRAMs

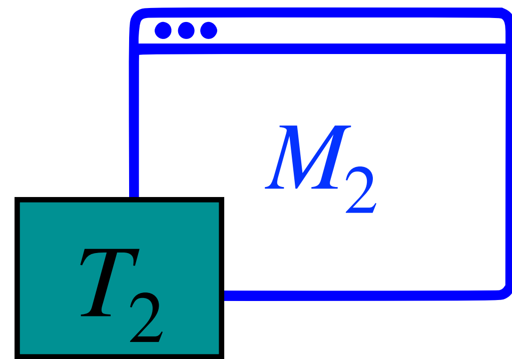


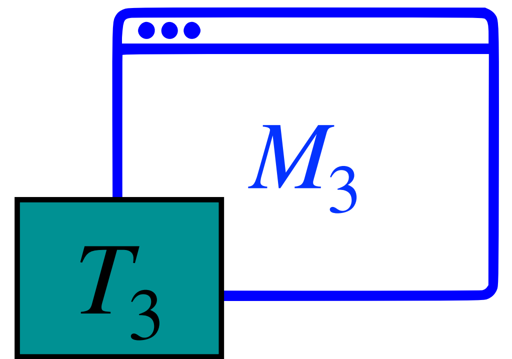
Idea: Want to maintain

$$\sum_i \mathsf{ctr}_i \leq \sum_j T_j$$

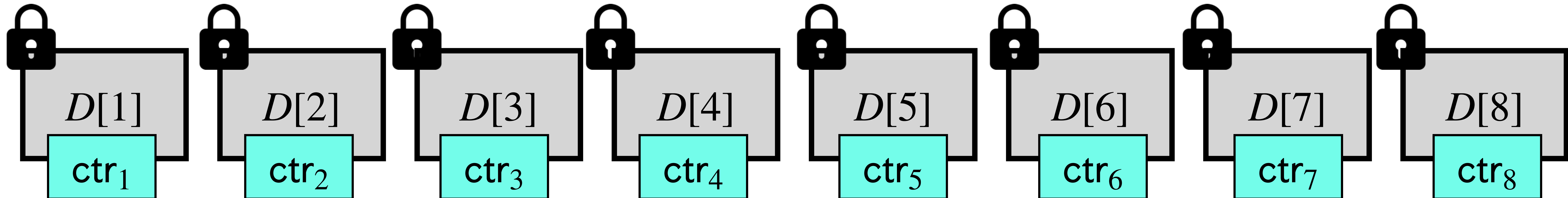
where equality holds **iff** server acts honestly

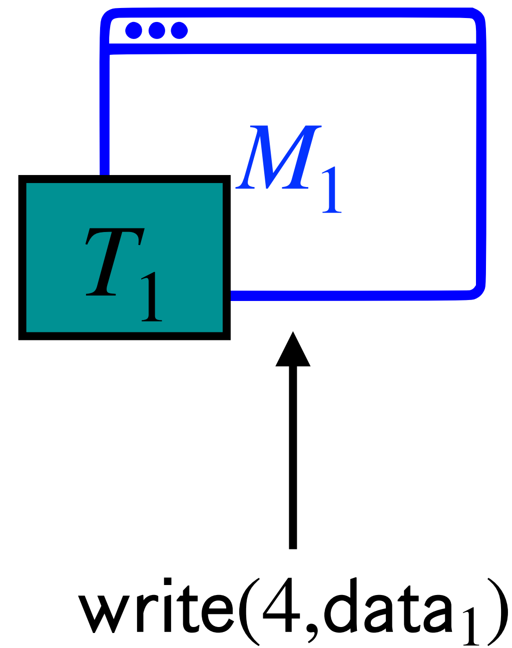


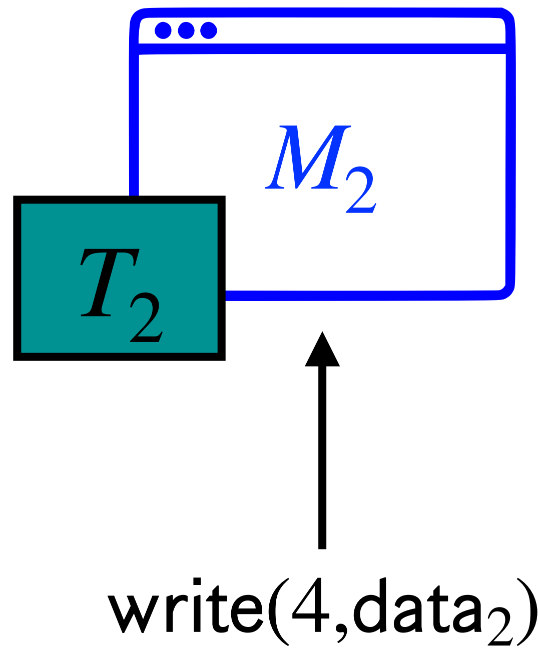


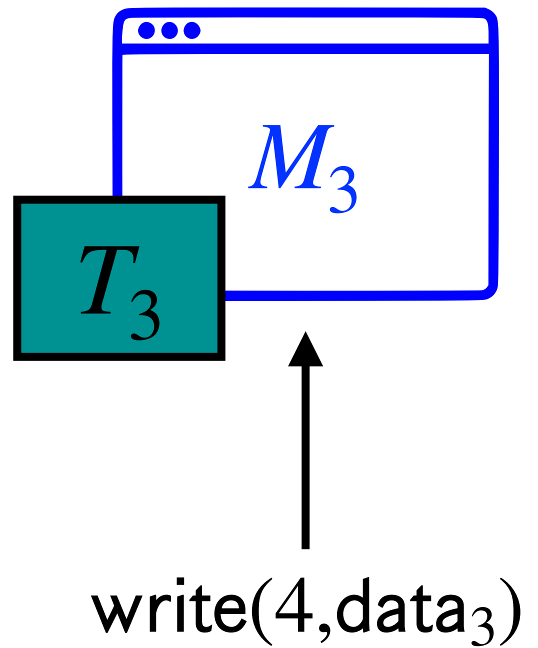


New: Offline-Checking for PRAMs

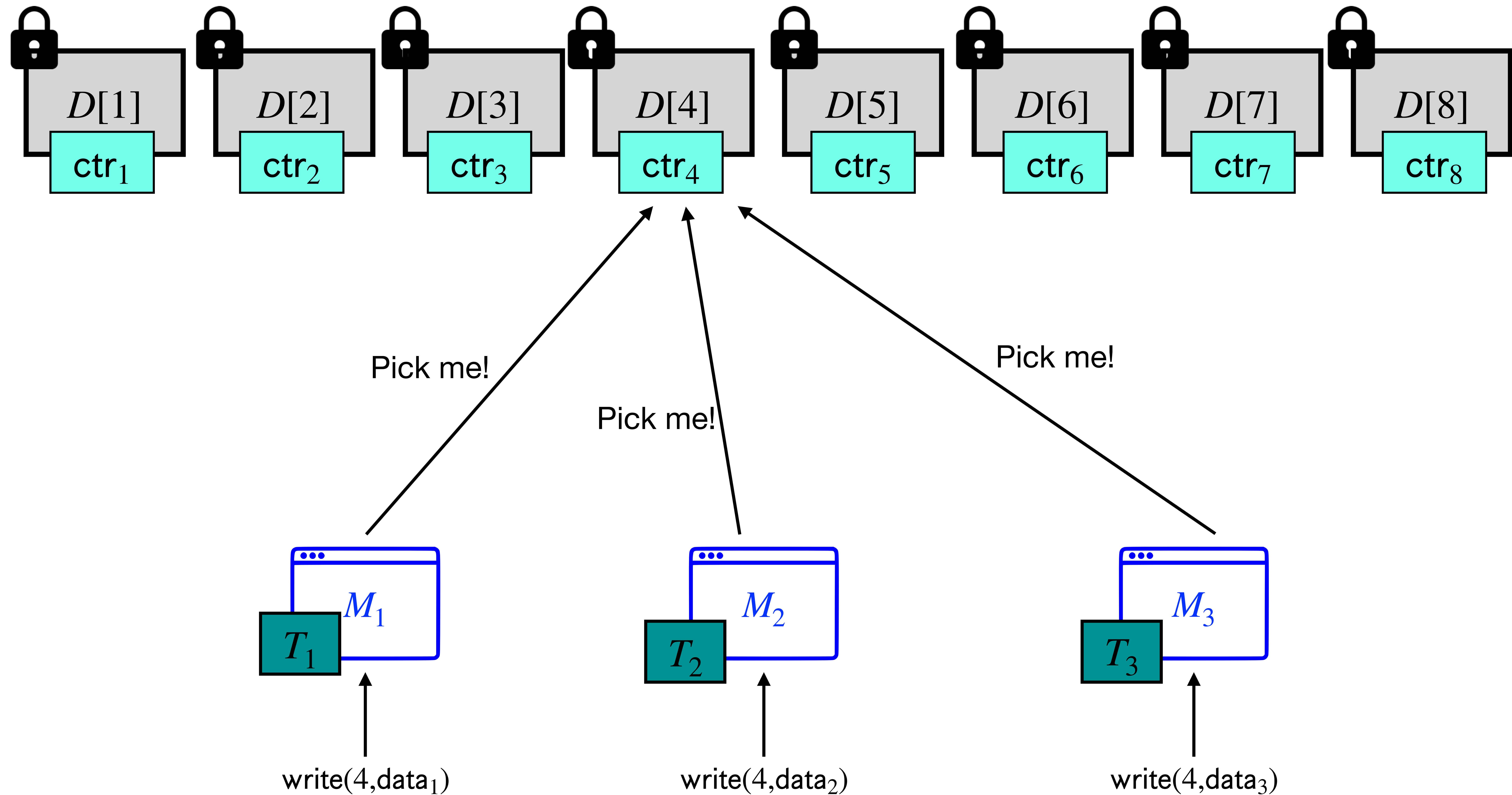




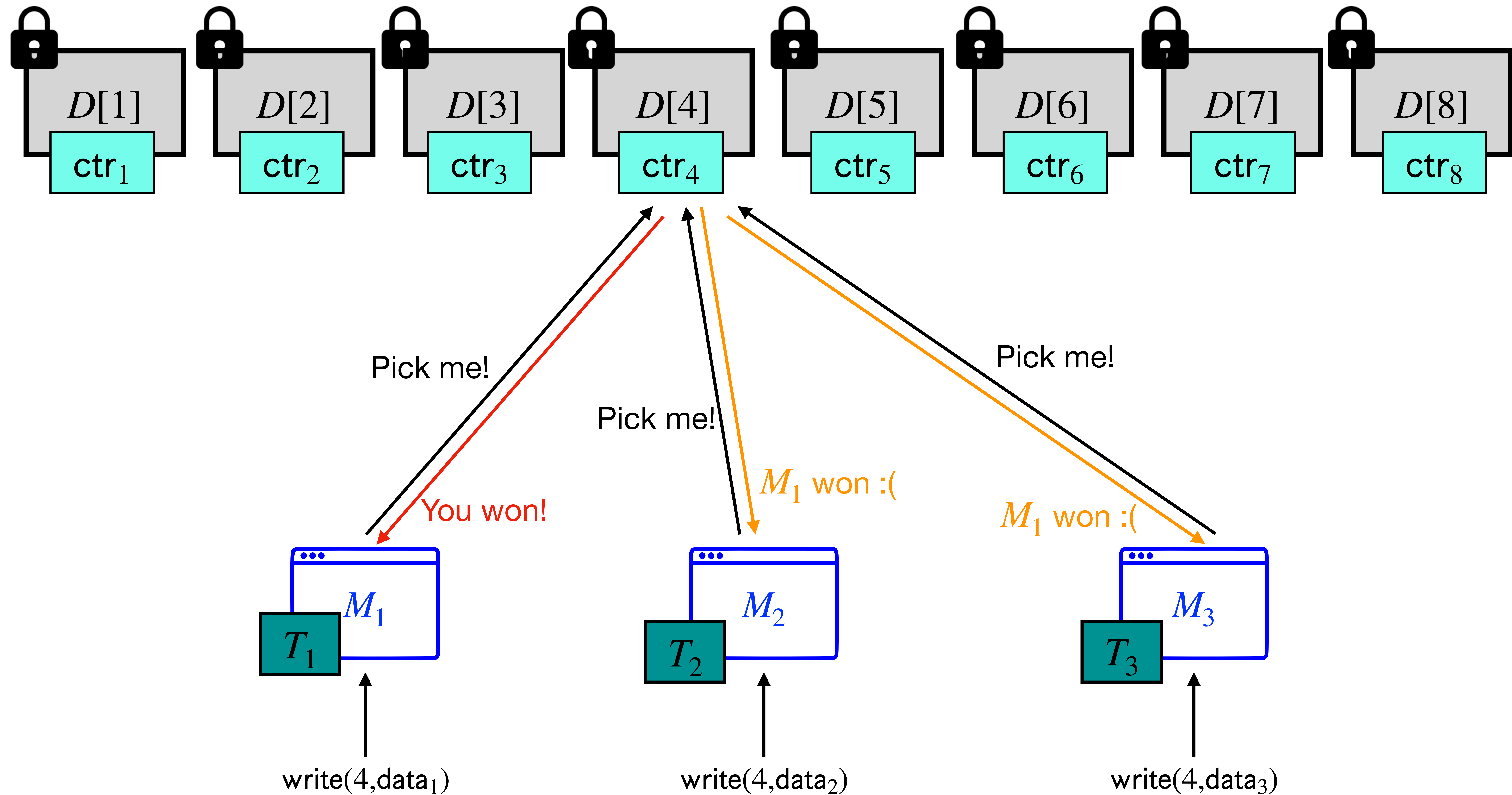




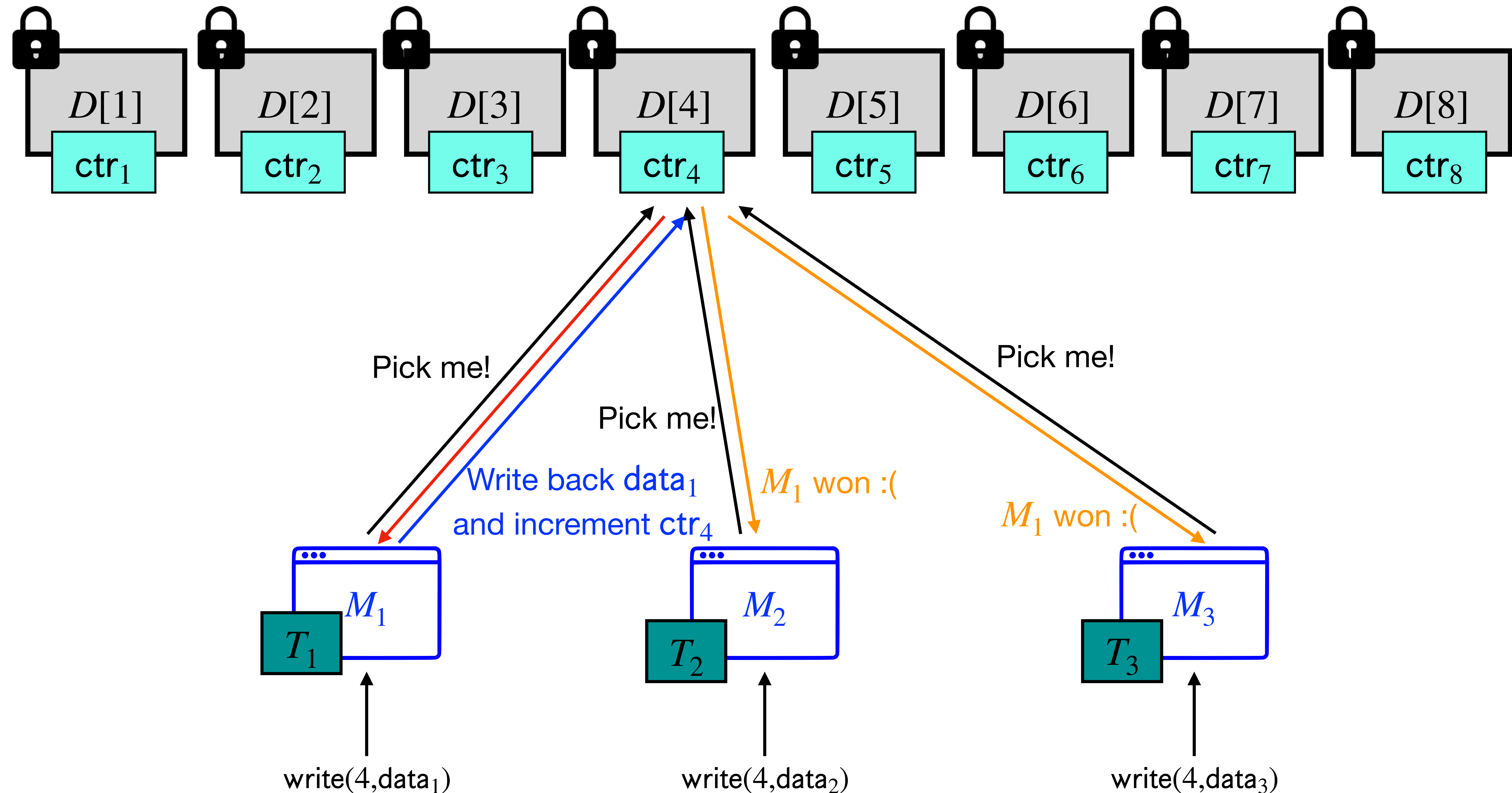
New: Offline-Checking for PRAMs



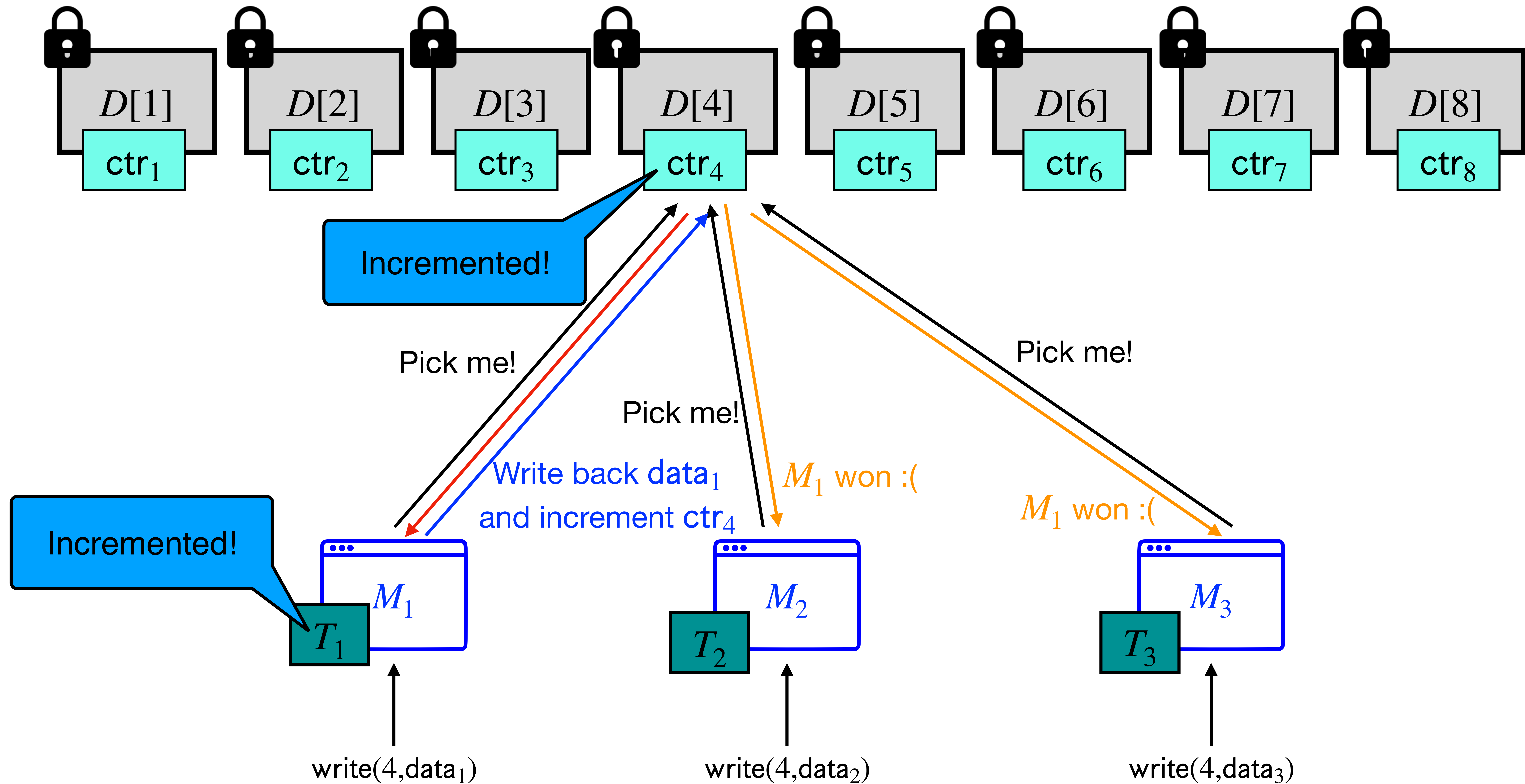
New: Offline-Checking for PRAMs



New: Offline-Checking for PRAMs



New: Offline-Checking for PRAMs



New: Offline-Checking for PRAMs