Final project

- Deadlines updated on the syllabus
- **Topic & literature review due this Friday (September 24)!**
  - A writeup of the existing work in the area, categorized into sub-topics
  - Should have a brief description (2-3 sentences) of what each paper does
  - A compressed version will become the related work section of the final project writeup

Oblivious RAM

Slides adapted from here, here

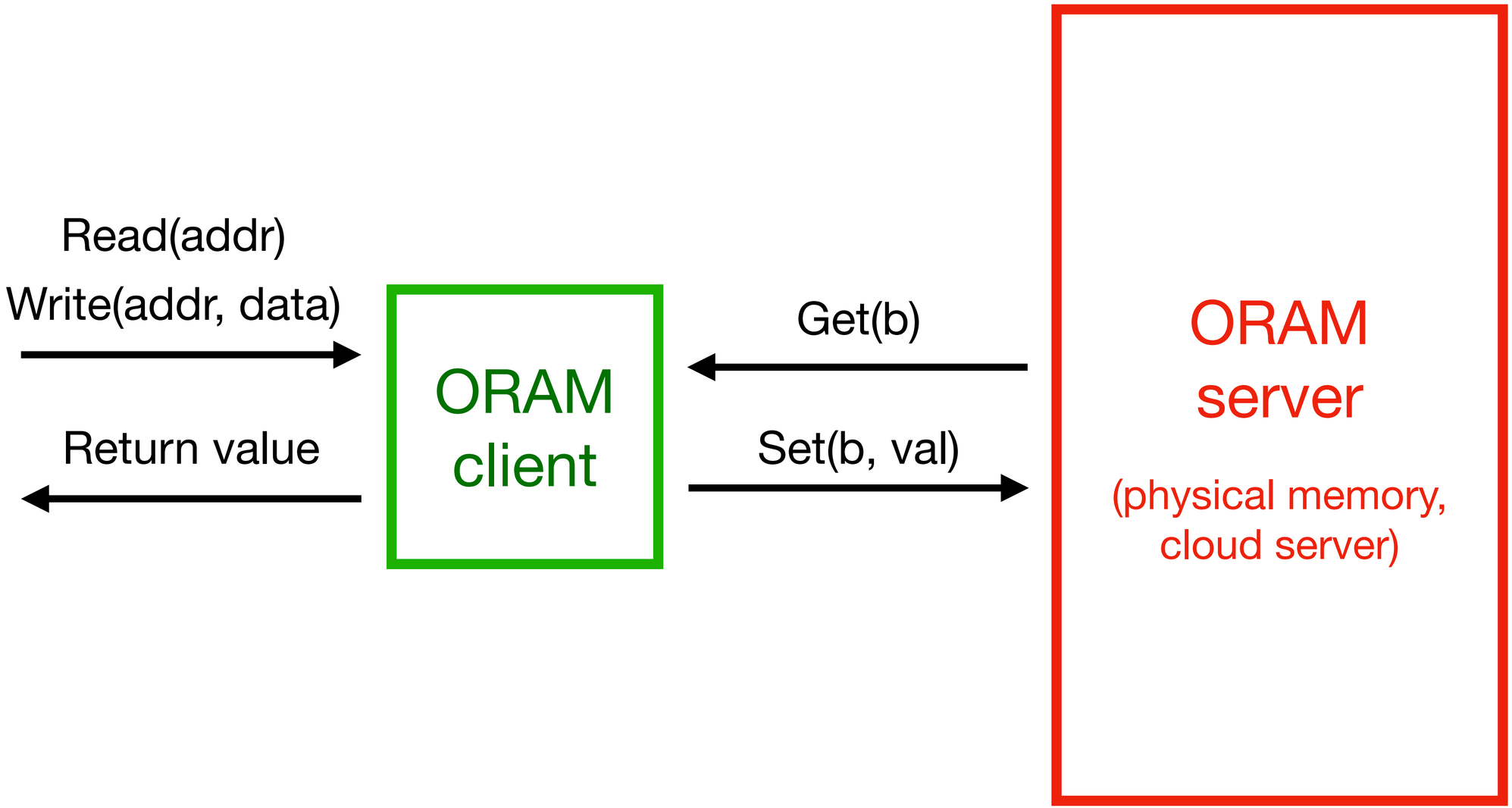
Motivation (GO93)

- Software protection - how to sell programs without allowing redistribution of the programs by the buyer to other users?
- Suggested physically shielded CPU
- Problem
  - Addresses to the memory cells are not kept secret
  - Could reveal essential properties about the program
- **How to execute a RAM program while hiding access pattern to the memory?**

Oblivious RAM

- Formulated under the RAM model
- Efficient simulation of an arbitrary RAM program on a probabilistic oblivious RAM
- The following should not be leaked about the original RAM program:
  - Which data block is accessed
  - The age of the data block (when it was last accessed)
  - Frequency of access to a block
  - Whether data blocks are being accessed together
  - Whether the access is a read or a write

ORAM abstraction



ORAM properties

- Let $y = \{y_1, y_2, \cdots, y_M\}$ where $y_i = (\text{op}_i, \text{a}_i, \text{data}_i)$, op_i denotes read/write, a_i is the location of the block, and data_i denotes the data being written,

- **Correctness:** for every access in sequence y, the ORAM client answers each operation correctly (or have a negligible probability of returning an incorrect answer)

- **Security:** for any two sequences y_1, y_2 of equal length, let $A(y)$ denote the (possibly randomized) sequence of accesses to the ORAM, then $A(y_1)$ and $A(y_2)$ are computationally indistinguishable by anyone except the ORAM client

Some naive solutions

- Solution 1: client reads all blocks from the server upon every logical request
  - $O(1)$ client storage (e.g., encryption key)
  - n RAM accesses per retrieval
- Solution 2: client stores all blocks, doesn't even need to contact the server
  - No communication
  - $O(n)$ client storage
- Solution 3: randomly permute all memory blocks through a secret permutation and store on server. Whenever the client wishes to access a block, make an access to the permuted location.
  - Security breaks if a block is accessed multiple times

ORAM design points

- Small communication cost (e.g., low amortized simulated RAM accesses per real RAM access)
- Small client storage (larger server storage is fine)
- If we want a non-trivial ORAM scheme, then it seems that a block needs to be “relocated” after it has been accessed
  - Needs probabilistic encryption so that relocation isn’t detected

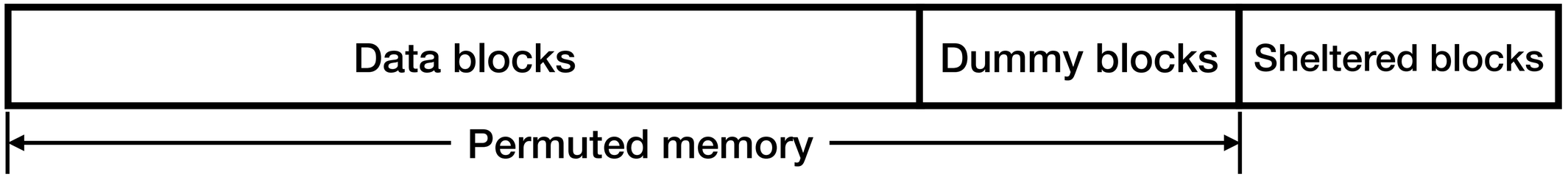
A simple ORAM construction

- Square-root ORAM from GO93
- Insight:
  - Suppose RAM holds logical blocks shuffled according to some permutation π that only the client knows
  - Need to amend the design so that any shuffled memory location is accessed at most once

A simple ORAM construction

- Initialize the ORAM server with $n + 2\sqrt{n}$ blocks
  - n are data blocks
  - $\sqrt{n}$ are dummy blocks
  - $\sqrt{n}$ are “sheltered” blocks

A simple ORAM construction



1. Randomly select a permutation π over blocks 1 through $n + \sqrt{n}$ and shuffle the data
2. Simulate $\sqrt{n}$ memory accesses of the original RAM
3. Return memory contents to the initial positions

A simple ORAM construction

1. **Randomly select a permutation π over blocks 1 through $n + \sqrt{n}$ and shuffle the data**

   1.1. Use a pseudorandom permutation to assign a new random location $\pi(i)$ to location i

   1.2. Use Batcher sorting network to oblivious sort by the tag $\pi(i)$

A simple ORAM construction

2. Simulate $\sqrt{n}$ memory accesses of the original RAM. To access location i:

2.1. Scan through sheltered blocks. If the contents of the original i-th block is in the list, then fetch it.

2.2. If value was not found from the sheltered blocks, then read the data block at location $\pi(i)$. If value is found, then read location $\pi(n + c)$ where c is the number of single accesses simulated so far in the current run. This is a read to a dummy block.

2.3. Scan through sheltered blocks again, and write the (possibly updated) value of block i.

A simple ORAM construction

3. **Return memory contents to the initial positions**

   3.1. This is done using another oblivious sort, except this time using the original block location.

   3.2. If there was any write to location i, then the updated value in the sheltered block must be shuffled to the correct location within the data blocks. This can be done in the previous step 2 by invalidating an older version of the block.

A simple ORAM construction

- Correctness
- Security
  - Oblivious sort
  - Scan over sheltered blocks
  - Permuted location is revealed, but it is random, and only one such access is made

A simple ORAM construction

- Client storage: $O(1)$
- Server storage: $n + 2\sqrt{n}$ blocks
- Simulated RAM access overhead
  - Permutation/sorting: $O(n \log^2 n)$ total, $O(\sqrt{n} \log^2 n)$ per access
  - RAM simulation: $O(\sqrt{n})$ per access due to scanning
  - Amortized $O(\sqrt{n} \log^2 n)$ simulated RAM accesses per real access

A “simple” ORAM construction with better complexities: Path ORAM