Hiding Patterns from the Database Owner
- Private Information Retrieval (PIR)
- Oblivious RAM (ORAM)

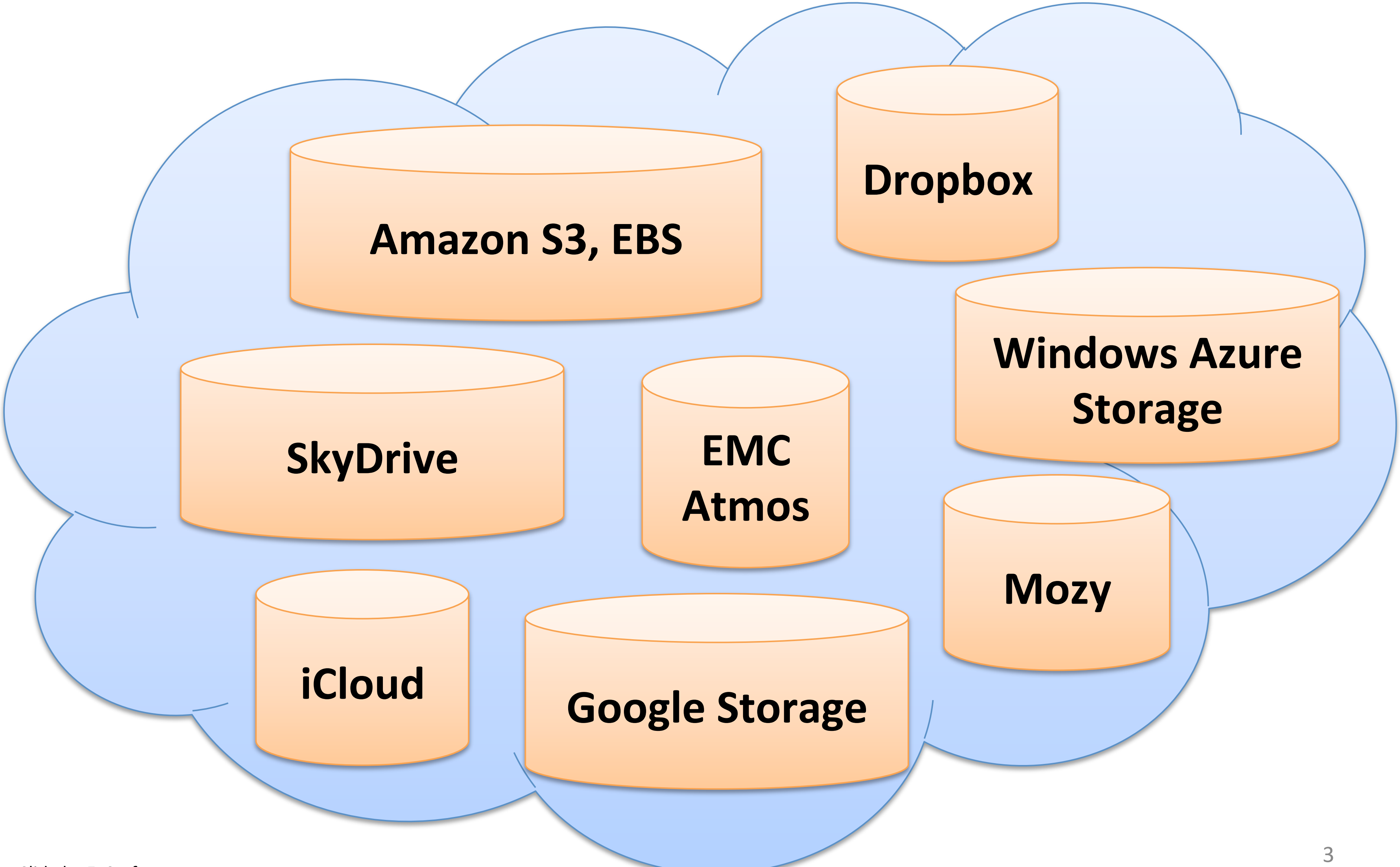
Erman Ayday

with gratitude to Jean Louis Raisaro and J.P Hubaux

Outline

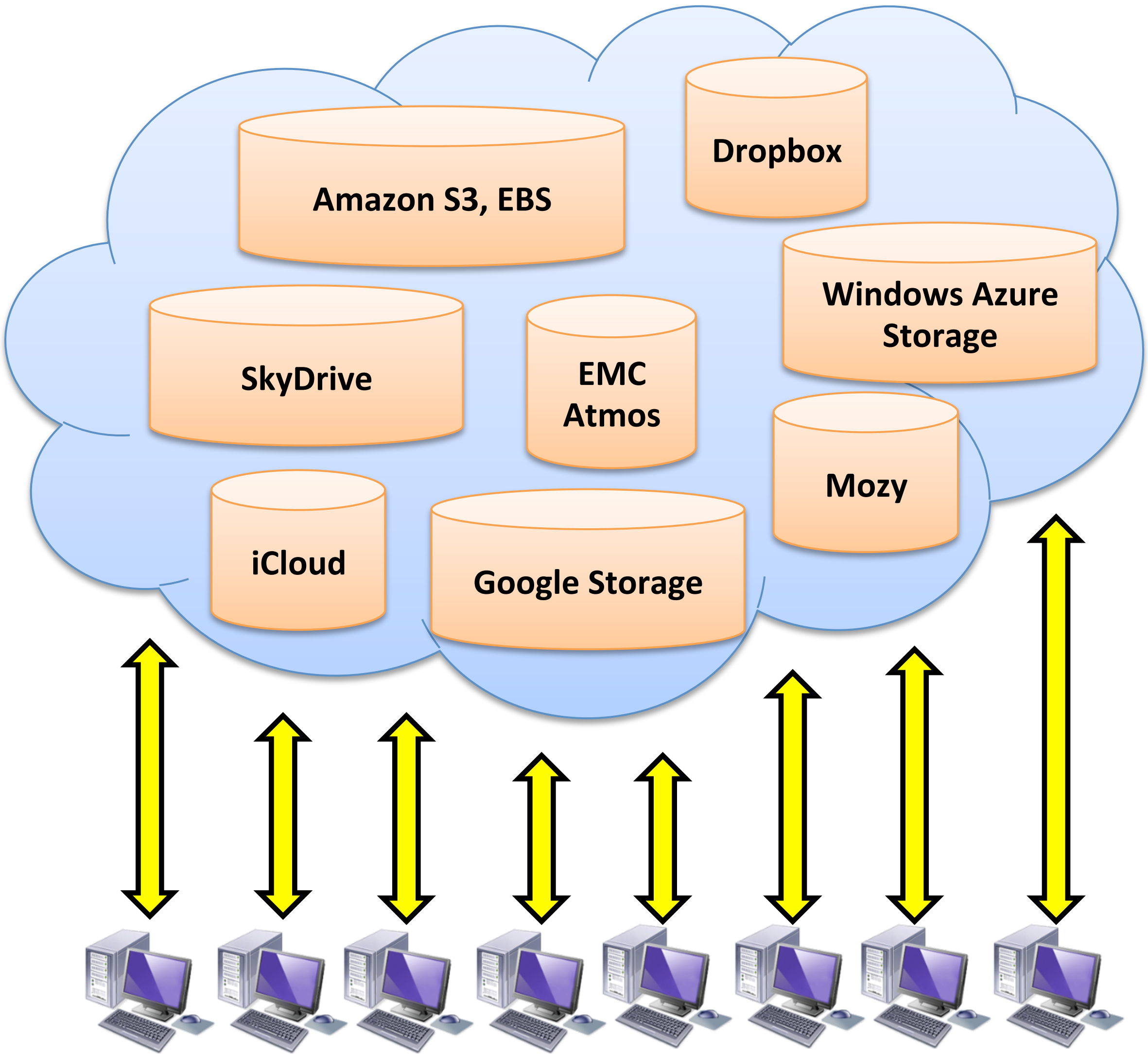
- Motivation
- Private Information Retrieval (PIR)
  – IT-PIR
  – cPIR
  – SPIR
- Oblivious RAM (ORAM)
  – Practical ORAM
  – PathORAM (Stefanov et al. CCS13)
- PIR + ORAM (Huang and Goldberg WPES13)

Cloud Storage



Slide by E. Stefanov

Cloud Storage



Can we **TRUST** the cloud?

Slide by E. Stefanov

Data Privacy

- ***Data privacy*** is a growing concern.
  - Large attack surface (possibly hundreds of servers)
  - Infrastructure bugs
  - Malware
  - Disgruntled employees
  - Big brother

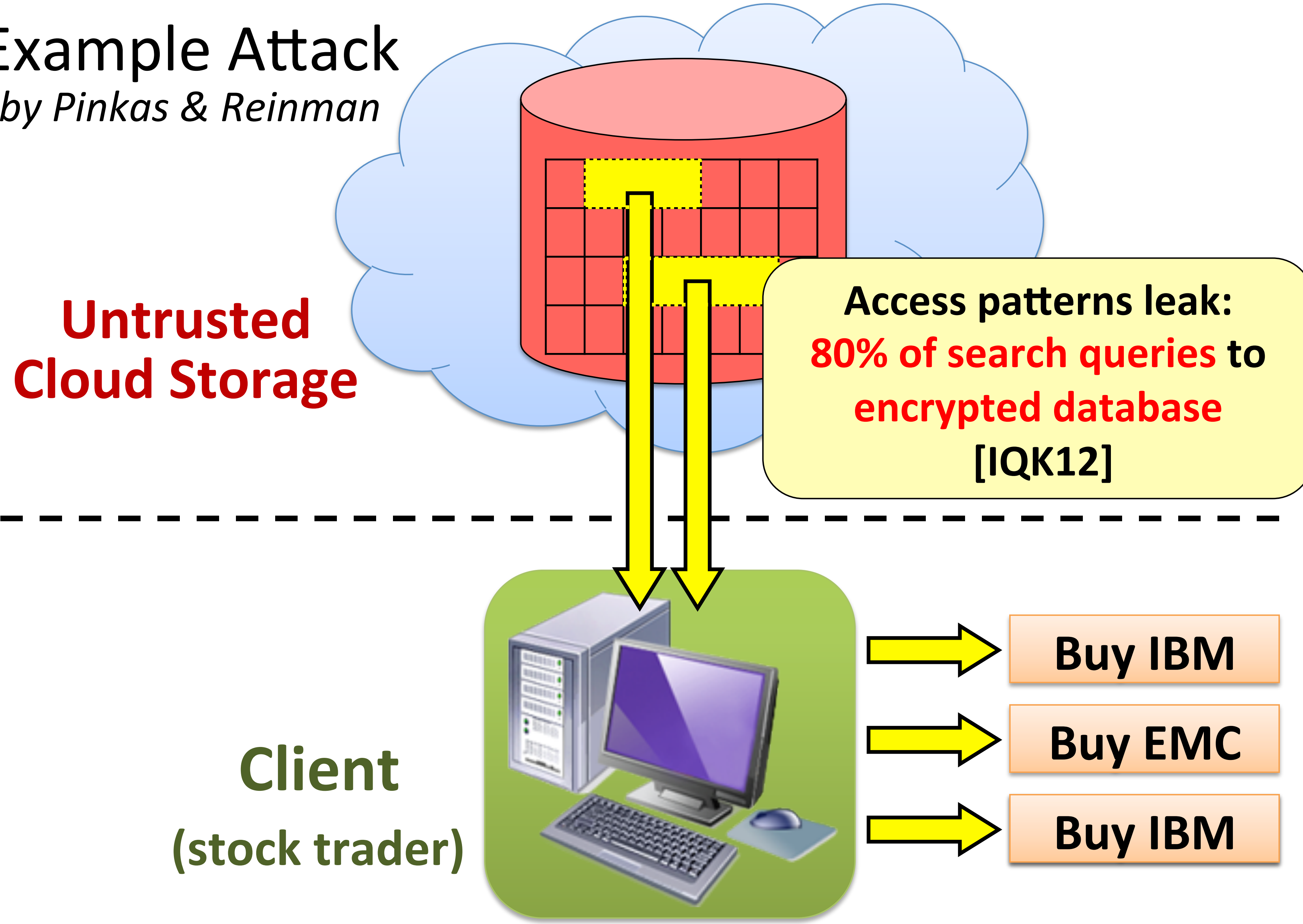
- So, many organizations ***encrypt*** their data.

But, encryption is not always enough.

Access patterns

can leak sensitive information.

Example Attack
by Pinkas & Reinman



If a sequence of data access requests is always followed by a stock exchange operation, the server can gain sensitive information even when the data is encrypted

Slide by E. Stefanov

Security for Outsourced Storage

- **Confidentiality**
  - Encrypt
- **Integrity**
  - MAC / Sign
  - Merkle tree
- **Reliability**
  - Redundancy
  - Proofs of retrievability (PoR)
- **Access privacy?**
  - **Private Information Retrieval (PIR)**
  - **Oblivious RAM (ORAM)**

Privacy and Cloud Computing

- Cloud computing infrastructures enable companies to cut IT costs by outsourcing storage and computations on-demand
- YET, clients of cloud computing services currently have **no means to control the privacy of their data** (data availability is also an issue, not addressed in this course)
- The lack of trust has fostered the design of new sophisticated technologies to ensure privacy against cloud service providers:
  - **Private Information Retrieval (PIR)**
  - **Oblivious RAM (ORAM)**
- These techniques bring **substantial computational and storage overhead**
- Privacy vs. Efficiency

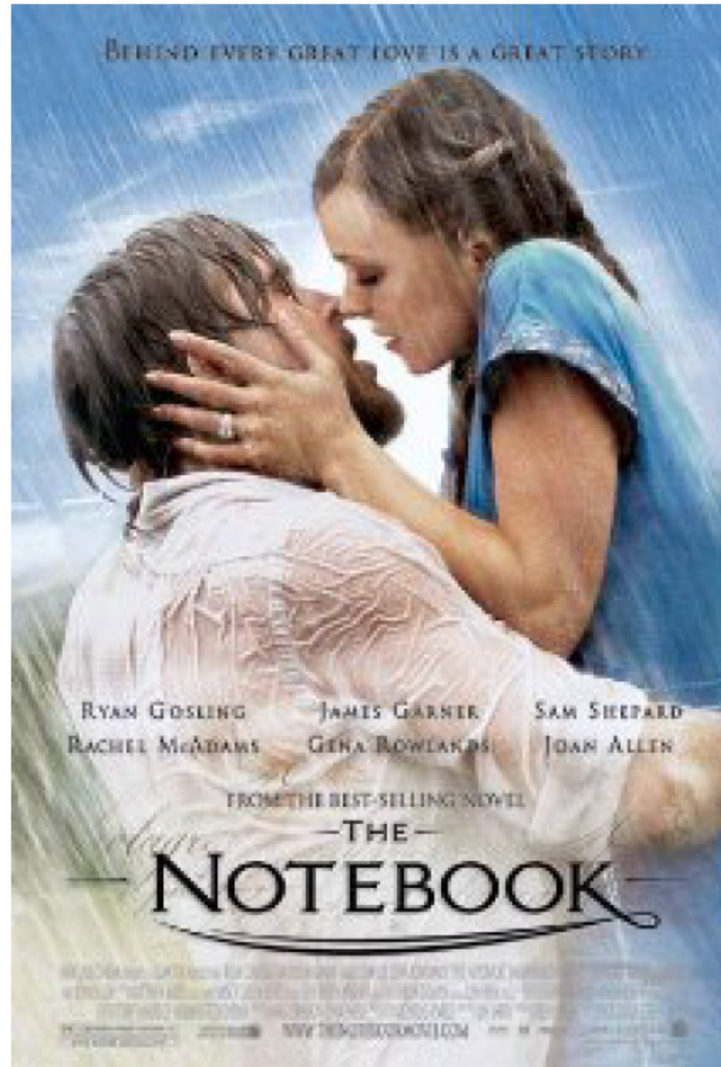
Private Information Retrieval (PIR)

Credits: some slides by Casey Devet

A Real-World Example

Suppose there is a movie database and I want to find information on the movie *The Notebook*.

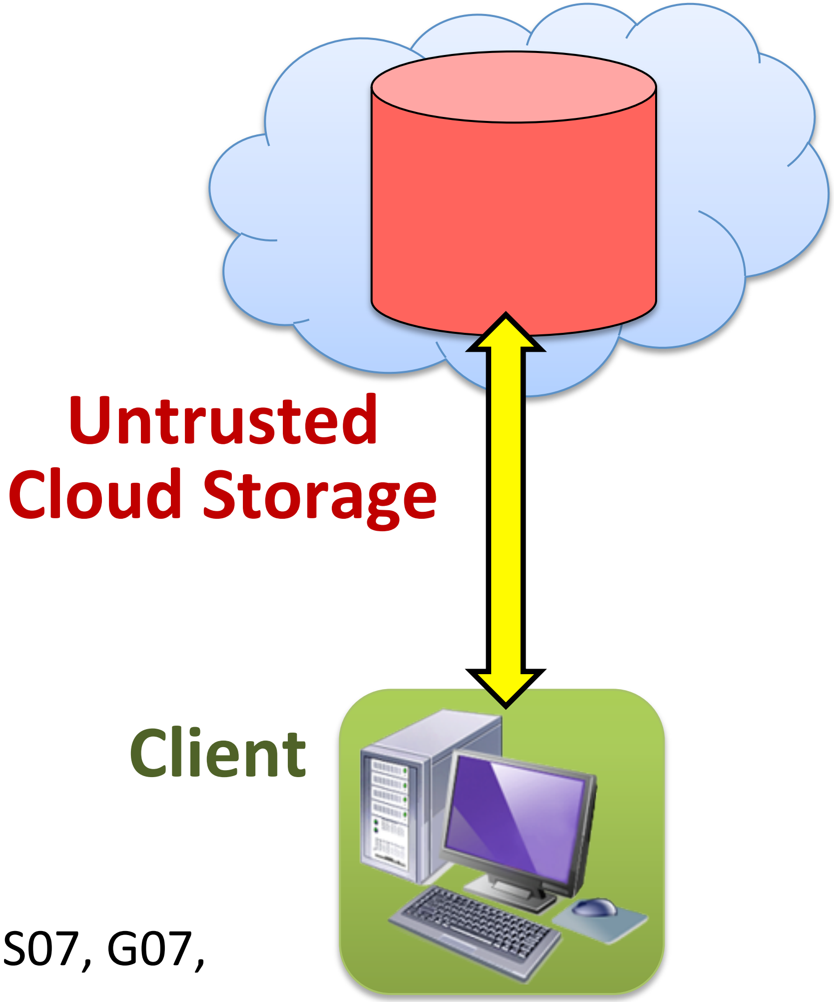
I don't want **the database operator** to know about my interest in this movie.



Private Information Retrieval (PIR)

- **Goal:** Protect privacy of **user's queries**.

- The database does not learn the query terms or responses.



Proposed by Chor et al. [CKGS95]

Recently: [KO97, CG97, CKGS98, BS02, AG07, BS07, G07, OG11, DGH12, HHG13, HG13, MBC14, …]

But…

How to do this?

Download the entire

Database?

Trivial Solution

U
Clo

Impractical

$O(N)$ bandwidth overhead

***N* is the number of records in the database**

IBM

EMC

Buy IBM

(stock trader)

IT-PIR vs. cPIR

- **Information Theoretic PIR (IT-PIR)**
  - Non-colluding *L* servers
  - Each server holds a copy of the database
  - **Perfectly secure** if some number of these servers are not colluding

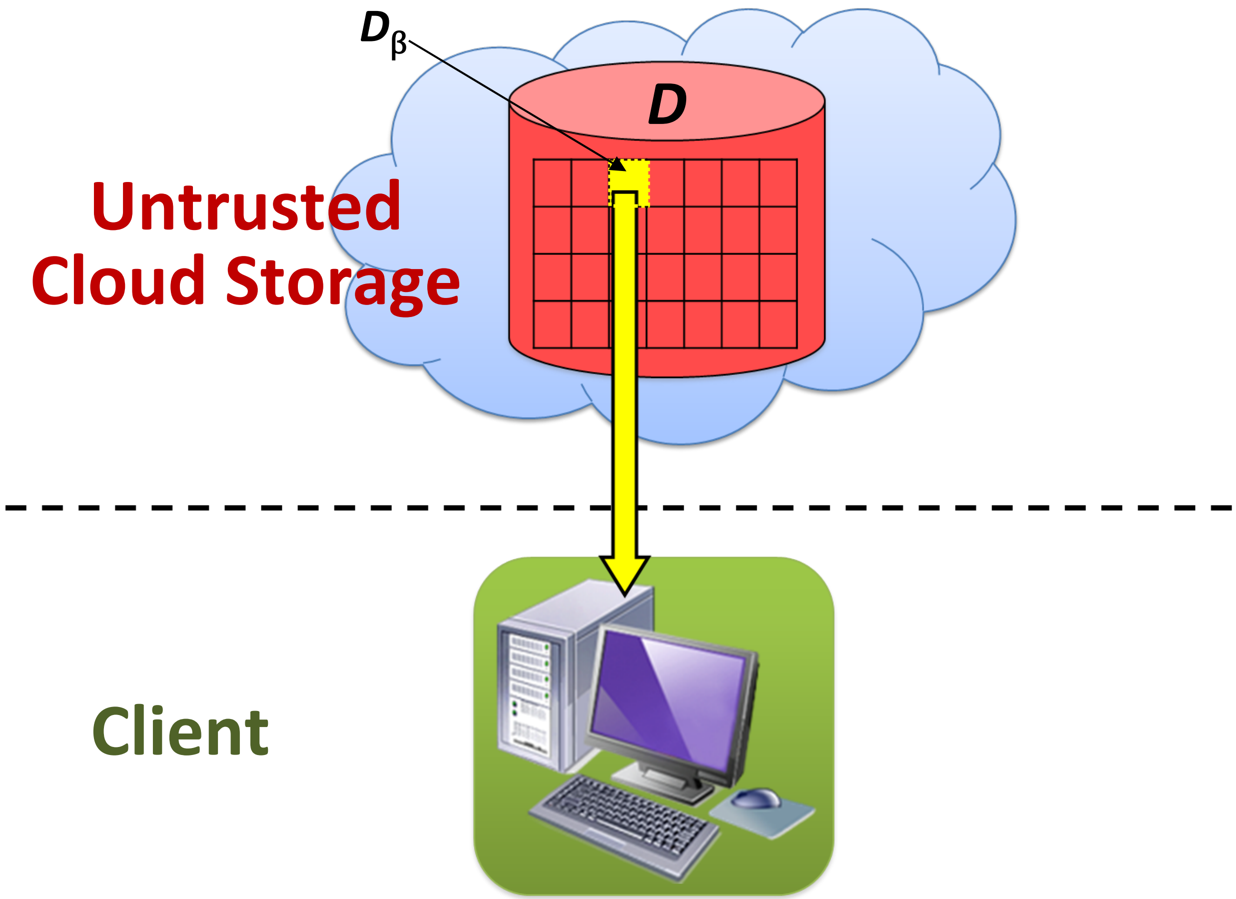
- **Computational PIR (cPIR)**
  - Single database-server
  - Uses cryptographic techniques to encrypt the user's query
  - The security of cPIR relies on the security of the underling encryption
  - Privacy is ensured **only against computationally-bounded attackers**

IT-PIR: the Goal

Database D with blocks $D_1,\ldots,D_r$

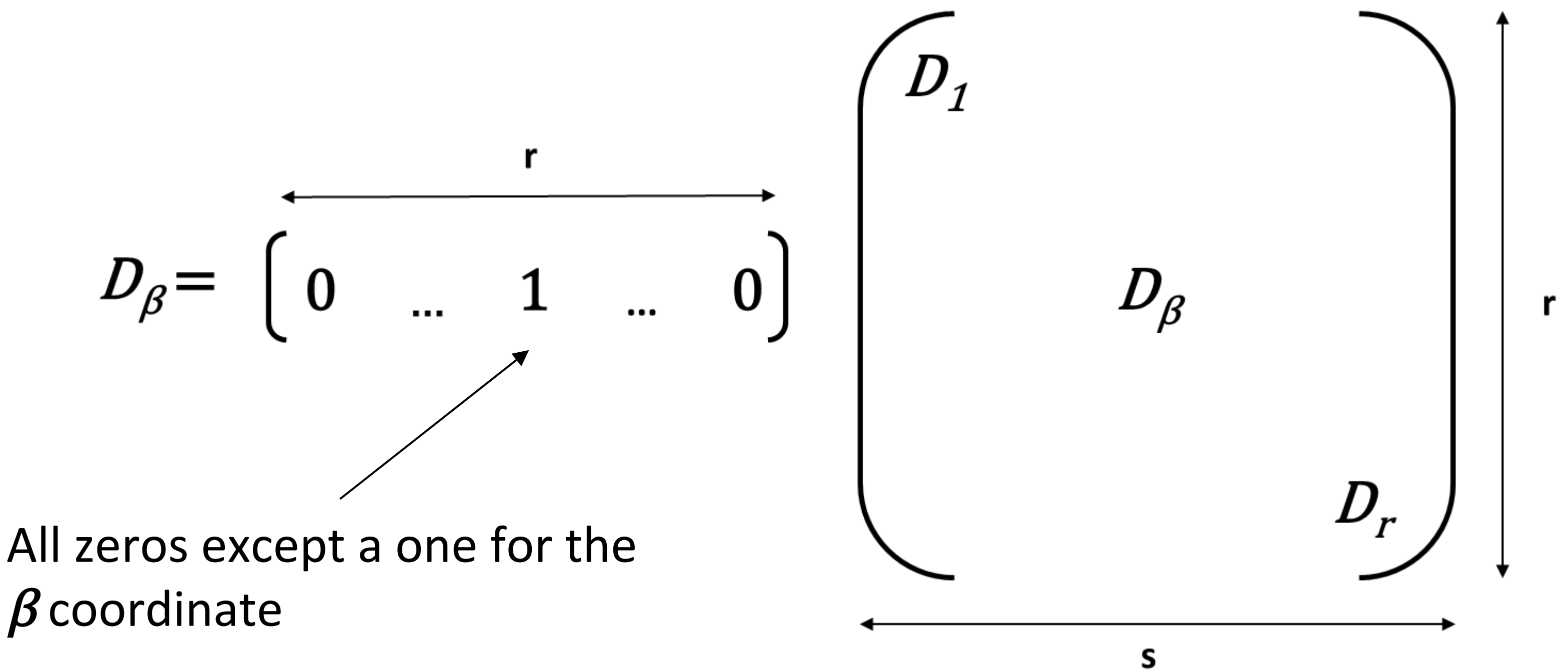
Goals:

- Retrieve D_β from the database without leaking β.
- Do this without downloading the entire database.



IT-PIR: Goldberg's Scheme [Gol07]

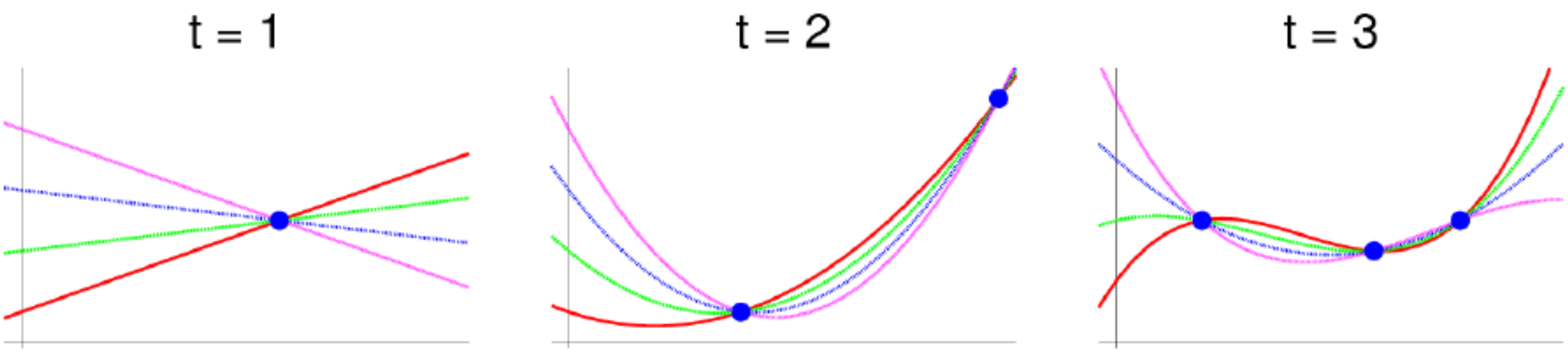
- Database D can be represented as an $\boldsymbol{r}$ x $\boldsymbol{s}$ matrix
- $\boldsymbol{D_\beta} = \boldsymbol{e_\beta} \,.\, \boldsymbol{D}$



Shamir Secret Sharing (reminder) [Sha79]

- (t+1,L) threshold scheme to share a secret S
  - Choose t random positive integers a_1, …a_t
  - Let a_0=S
  - Build polynomial f(x)=a_0 + a_1x + a_2x^2 + … + a_tx^t
  - Construct L points out of f(x) -> (i,f(i))
  - Distribute a point (share) to each participant
  - At least (t+1) shares are needed to learn S

Shamir Secret Sharing (reminder) [Sha79]



(Simplified version, just to re-convey the intuition)

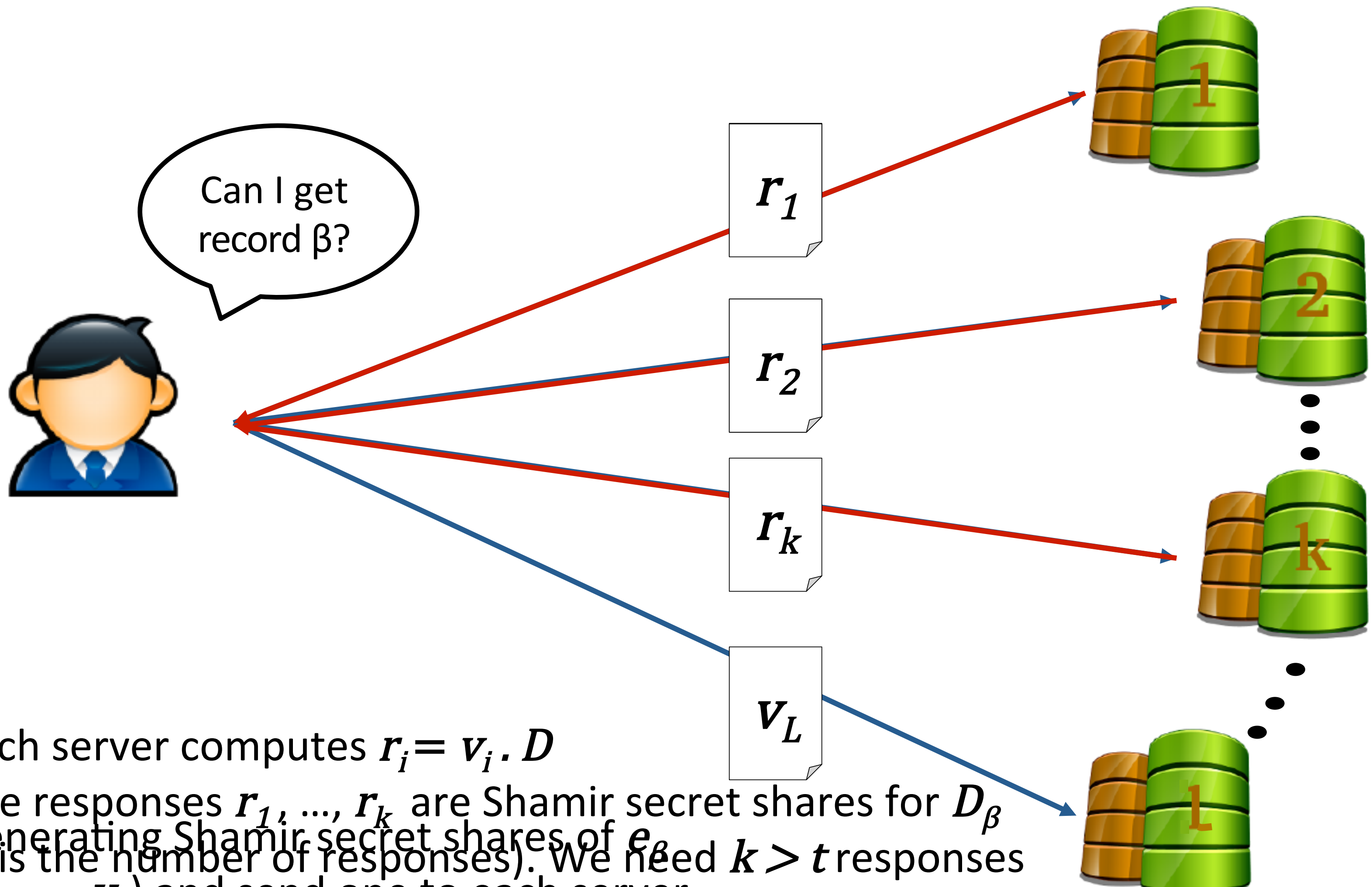
Construction:

•Assume the presence of *L* parties, we pick a random point (the secret) in a field and a polynomial of degree *t* such that the secret is the *y-axis* intercept of that polynomial and $L \geq t+1$

•We then pick *L* random points on this polynomial and each party is provided with one

•If we know at most *t* points we cannot reconstruct the secret. There is only 1 polynomial of order *t* going through the *t*+1 points

IT-PIR: Goldberg's Scheme (ctd.)



Each server computes $r_i = v_i \,.\, D$

The responses $r_1, \ldots, r_k$ are Shamir secret shares for D_β (k is the number of responses). We need $k > t$ responses to reconstruct the secret

Generating Shamir secret shares of e_β $(v_1, \ldots, v_L)$ and send one to each server

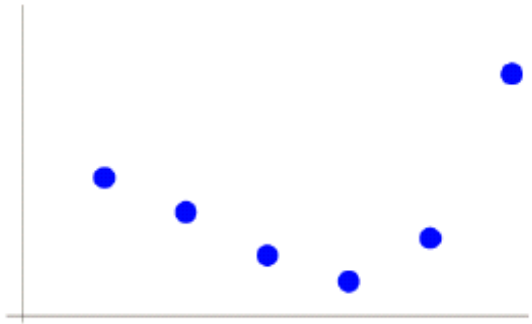
IT-PIR: Robustness

- **Robustness problem**: how many servers' responses do we need to be able to recover a database block?
- Multi-server PIR protocols tolerant of **non-responsive** or **malicious/colluding** server are called **robust** or **Byzantine robust**
- An ***L***-server system that can operate where only ***k*** of the servers respond, ***v*** of the servers respond incorrectly, and which can support up to ***t*** colluding server without revealing the client's query is called "***t-private v-byzantine robust k-out-of-L PIR***" [DGH 2012]

IT-PIR: Robustness

What happens if some of the responses (say v of k) are wrong? Ex. v = 1 and k = 5 :

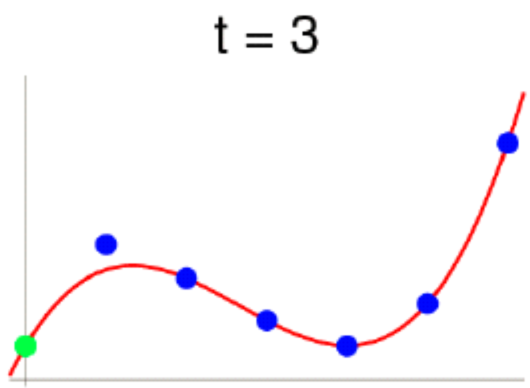
t = 3



The Shamir secret shares are a **Reed-Solomon codeword** encoding the polynomial.

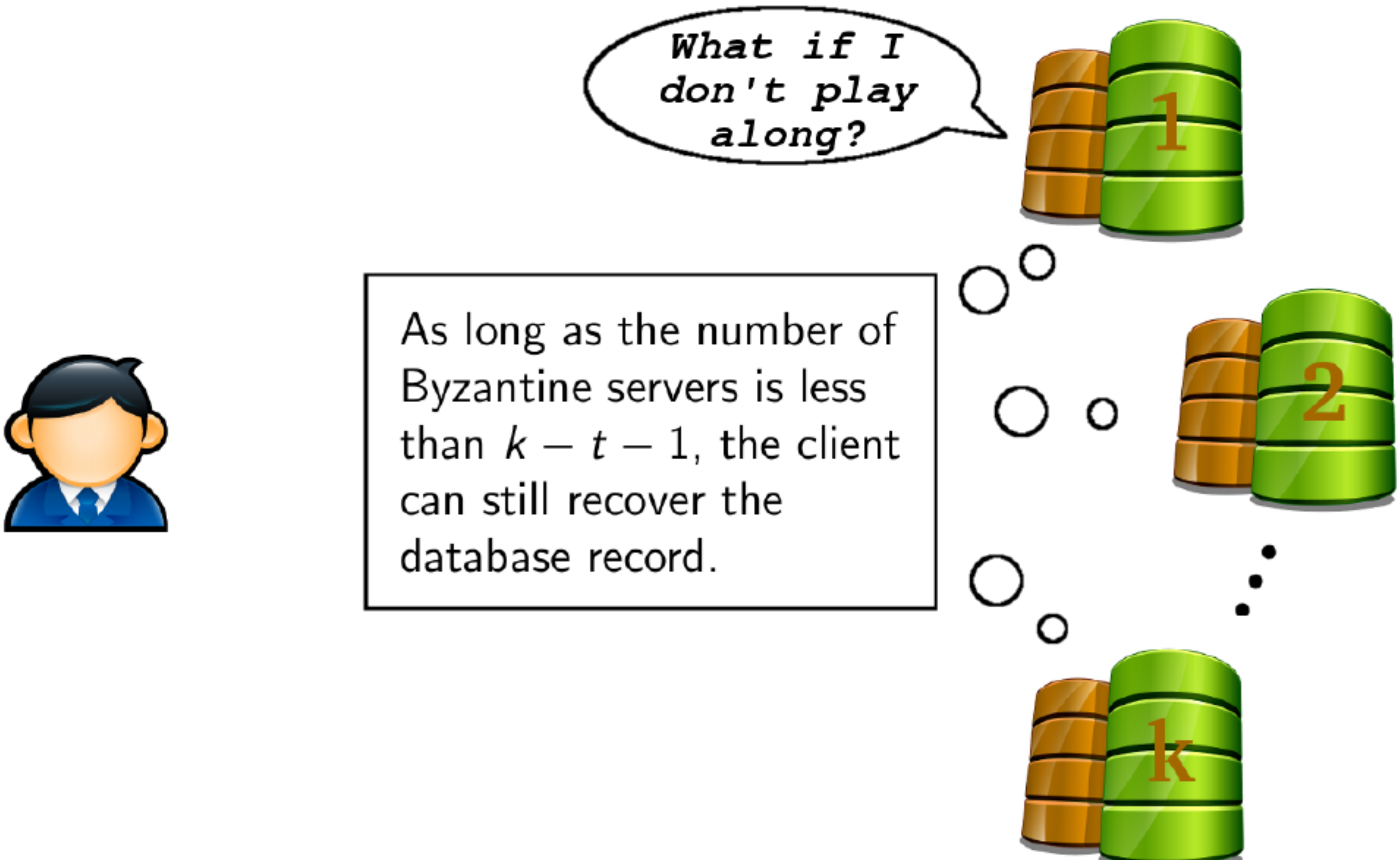
IT-PIR: Robustness

We can use **Reed-Solomon decoding algorithms** to find all polynomials of degree at most t that miss at most v of the responses. One of these polynomials is the correct one.



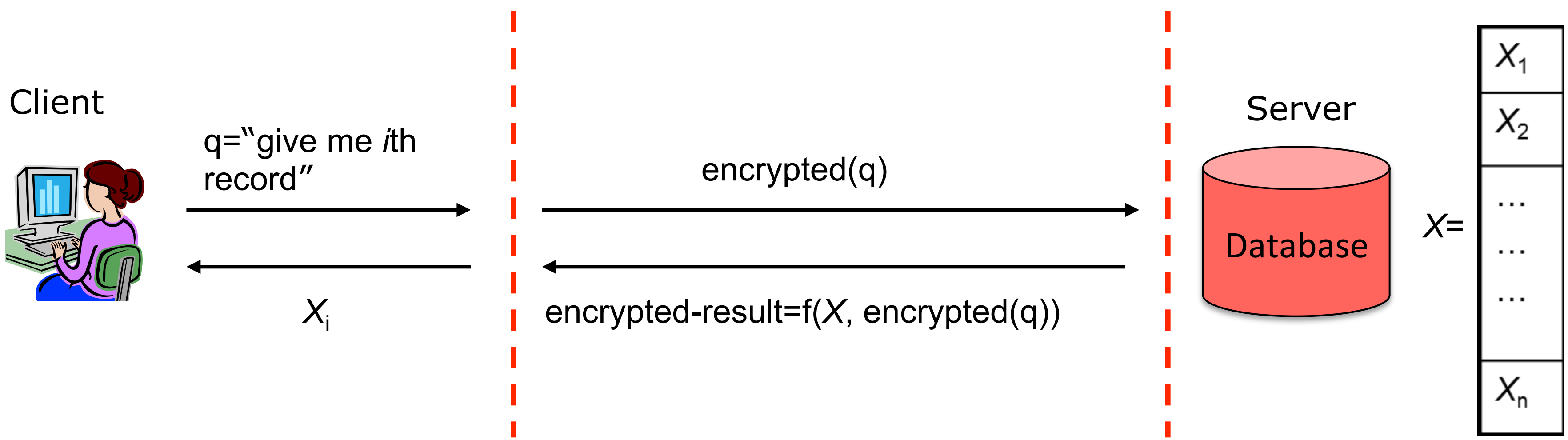
The **Byzantine robustness** of Goldberg's scheme is the bound on v. ($v < k$-t-1 is the theoretical max value)

Example


What if I don't play along?
1
As long as the number of Byzantine servers is less than $k - t - 1$, the client can still recover the database record.
2
k

Computational PIR (cPIR)

- User privacy is related to the (assumed) intractability of a mathematical problem.
- **Principle:** Achieve computationally complete privacy by applying cryptographic computations over the entire public data



cPIR: Theoretical Background

Quadratic Residue (QR)

- x is a *quadratic residue* (QR) mod N if

$$\exists\, y \in Z^*_N \; s.t.\; y^2 = x \bmod N$$

- E.g. $N = 35$, *11 is QR* ($9^2 = 11 \bmod 35$), *3 is QNR (no y exists for* $y^2 = 3 \bmod 35$*)*
- Essential properties:
  - *QR x QR = QR*
  - *QR x QNR = QNR*

- Let $N = p_1 \times p_2$, p_1 and p_2 are large primes of *m/2 bits (m* is the number of bits of the modulo *N)*

Quadratic Residuosity Assumption (QRA)

- Determining if a number is QR or QNR is computationally hard if p_1 and p_2 are not given.

Slide by S.Wang, D.Agrawal, and A. El Abbadi

cPIR: The Basic Scheme [KO97]

Slide by S.Wang, D.Agrawal, and A. El Abbadi

- Public data size: $n = 16$
- Organize data in an s x t (4 x 4) binary matrix M

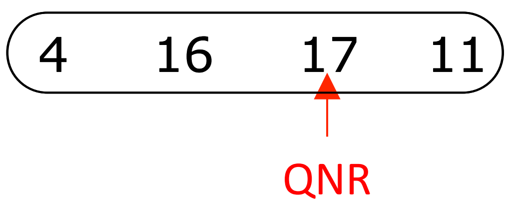
Client

Get $M_{2,3}$

$e=2$, $g=3$, $N=35$, $m=6$

QR={1,4,9,11,16,29}

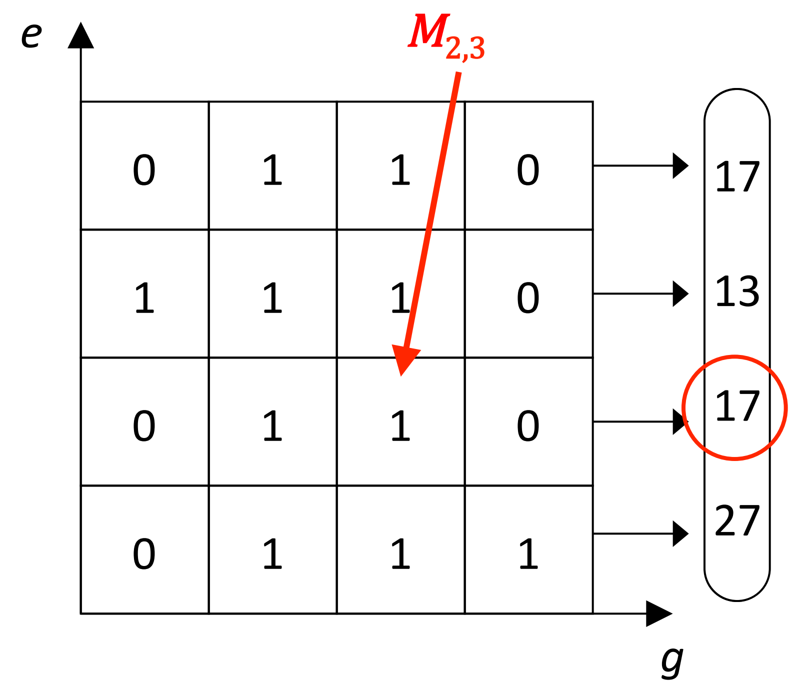
QNR = Z^*_{35} / QR



$z_3 = \text{QNR} \Rightarrow M_{2,3} = 1$

$z_3 = \text{QR} \Rightarrow M_{2,3} = 0$

Server



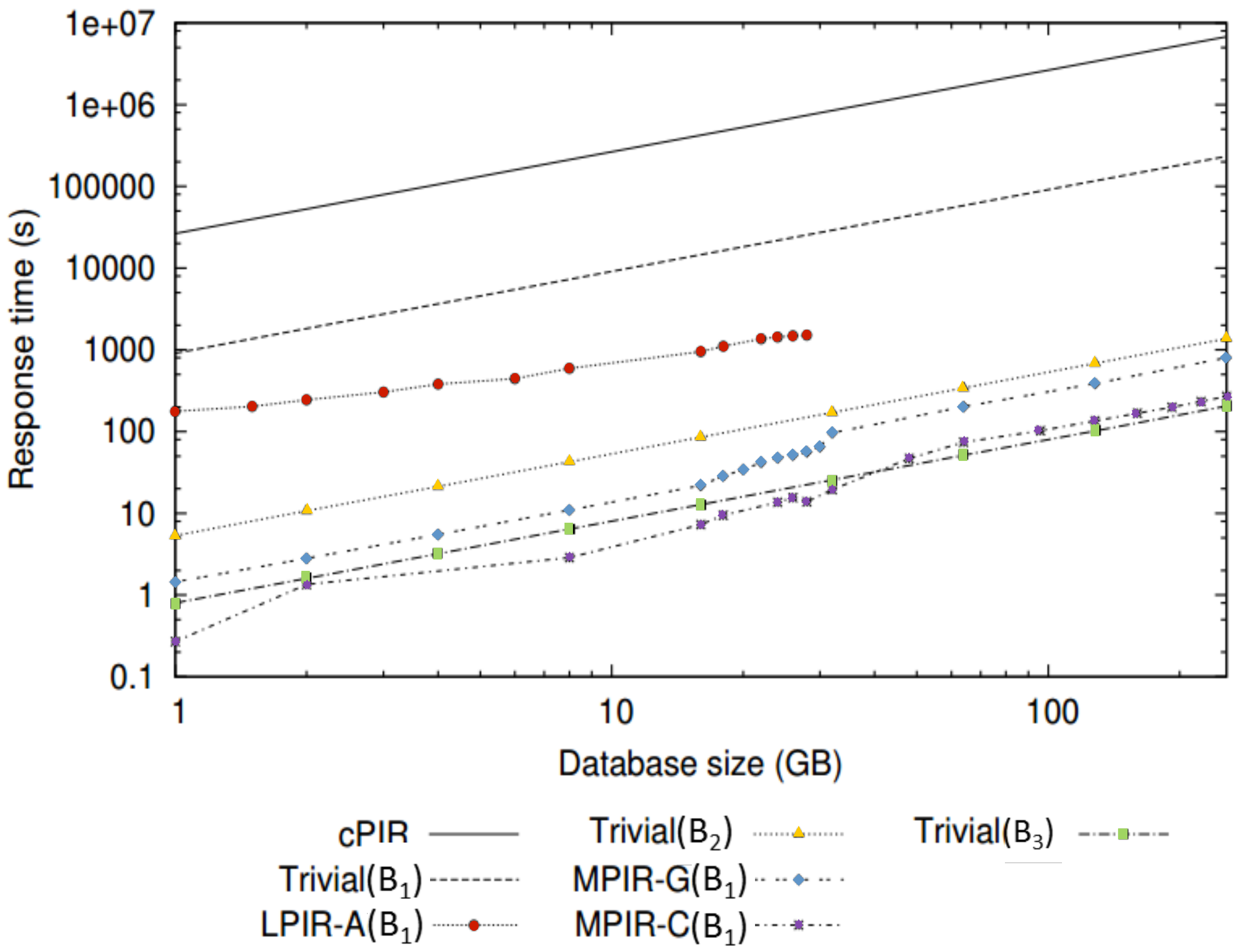
$\{z_i\}$

$$z_i = \prod_{j=1}^{t} y_j \,.\, y_j^{1-M_{i,j}}$$

Note: result leaks information about other rows to the client

Some Numbers [OG11]

Comparison of the response times of different PIR schemes over 3 current network bandwidths



cPIR [KO97]
LPIR-A [AM08]
MPIR-G [G07]
MPIR-C [CGKS95]

End User (B_1) – 9/2 Mbps
Commercial (B_2) – 1.5 Gbps
Ethernet LAN (B_3) – 10 Gbps

Symmetric PIR (SPIR)

- PIR protects the privacy of the user's query
  - The client can learn more than a single record
- **Symmetric PIR (SPIR)** protects the privacy of the database server [GIKM98, CDN09, CDN10, HOG11]:
  - The database client learns **only one** record per access request
  - The privacy of both the client and the server is preserved that's why it is called "symmetric"
  - Symmetric PIR implies Oblivious transfer (OT) [DMO00]

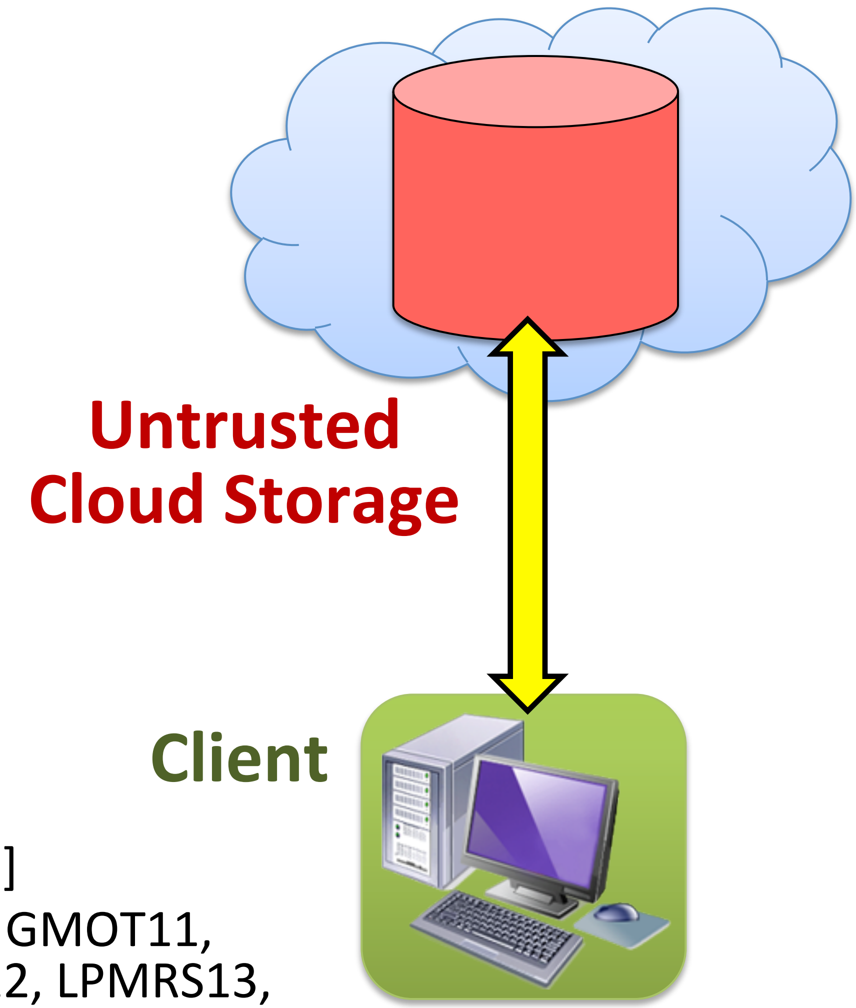
Oblivious RAM (ORAM)

Credits: some slides by Emil Stefanov

Oblivious RAM (O-RAM)

- **Goal:** Conceal **access patterns to remote storage** from the database owner.

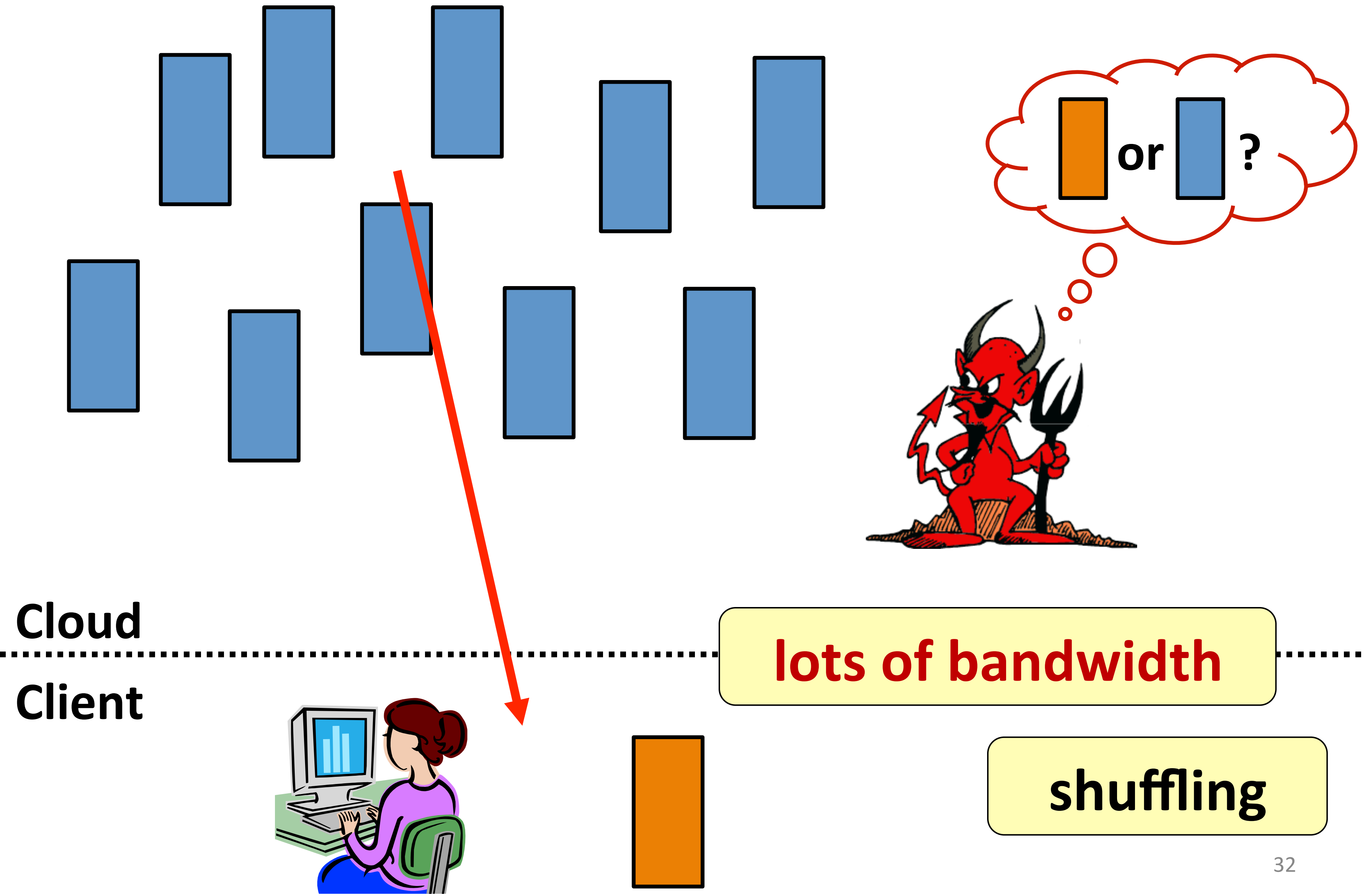
- An observer cannot distinguish a sequence of read/write operations from random operations.



Proposed by Goldreich and Ostrovsky. [GO96]

Recently: [OS97, WS08, WSC08, PR10, GM10, GMOT11, BMP11, SCSL11, SSS12, GMOT12, KLO12, WS12, LPMRS13, SS13, MBC14 …]

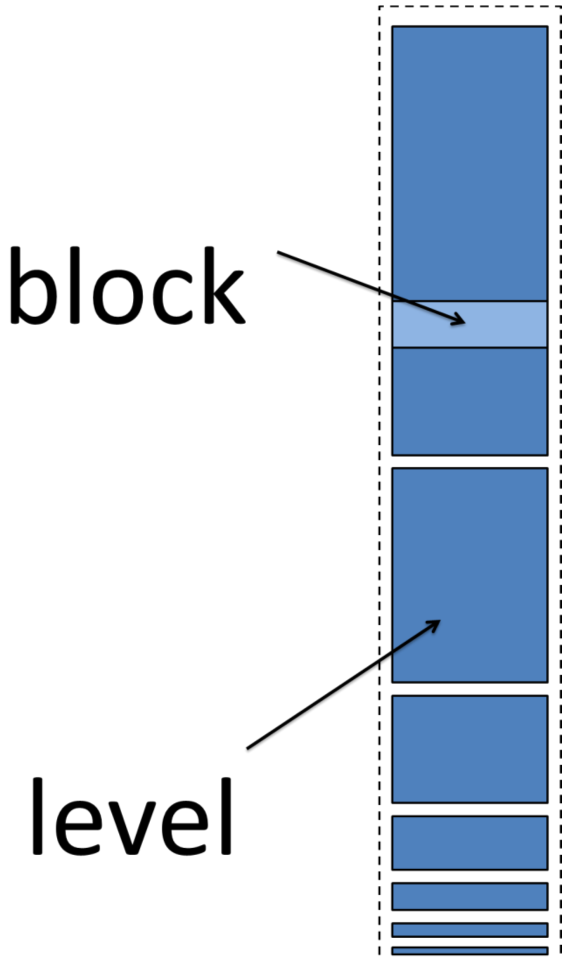
ORAM in a Nutshell



Goldreich's ORAM

- Provides access pattern privacy to a *single client*
- Database is considered to be a set of N semantically-secure encrypted blocks
- Data is organized into several levels as a pyramid
- Goal: Server should be unable to distinguish between reads, writes, and inserts
- Continuously shuffling and re-encrypting data as they are accessed

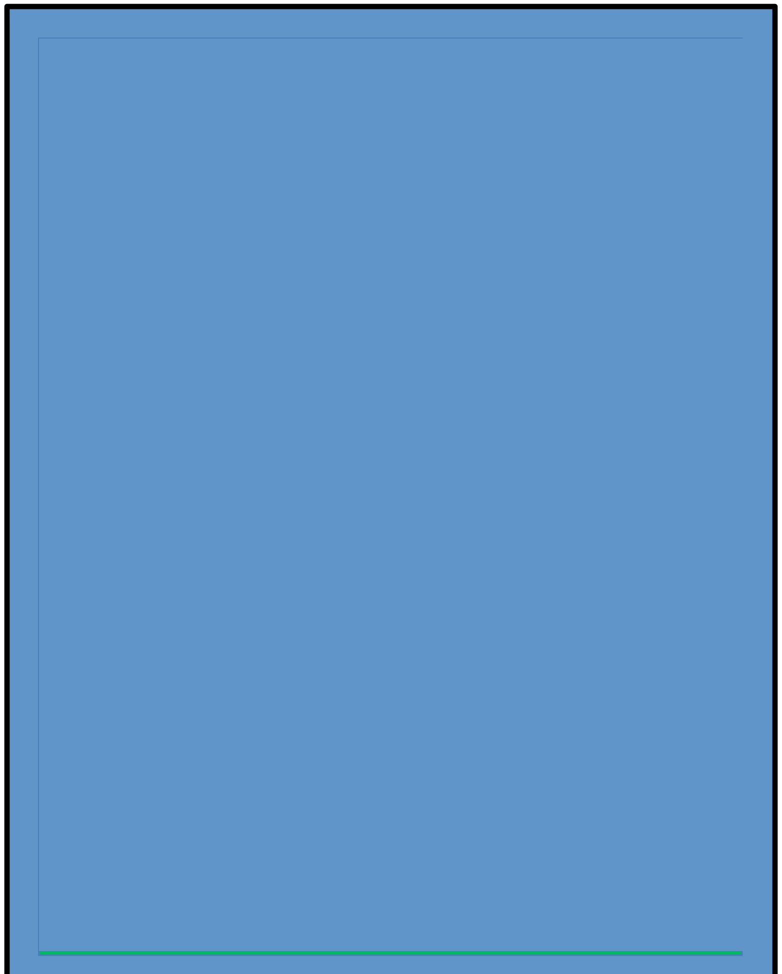
Existing Approaches



- Based on Goldreich-Ostrovsky scheme.
- $\log_2 N+1$ levels
  - Sizes: $1, 2, 4, \ldots, N$

[GO96, OS97, WS08, PR10, GM10, GMOT11, BMP11, GMOT12, KLO12…]

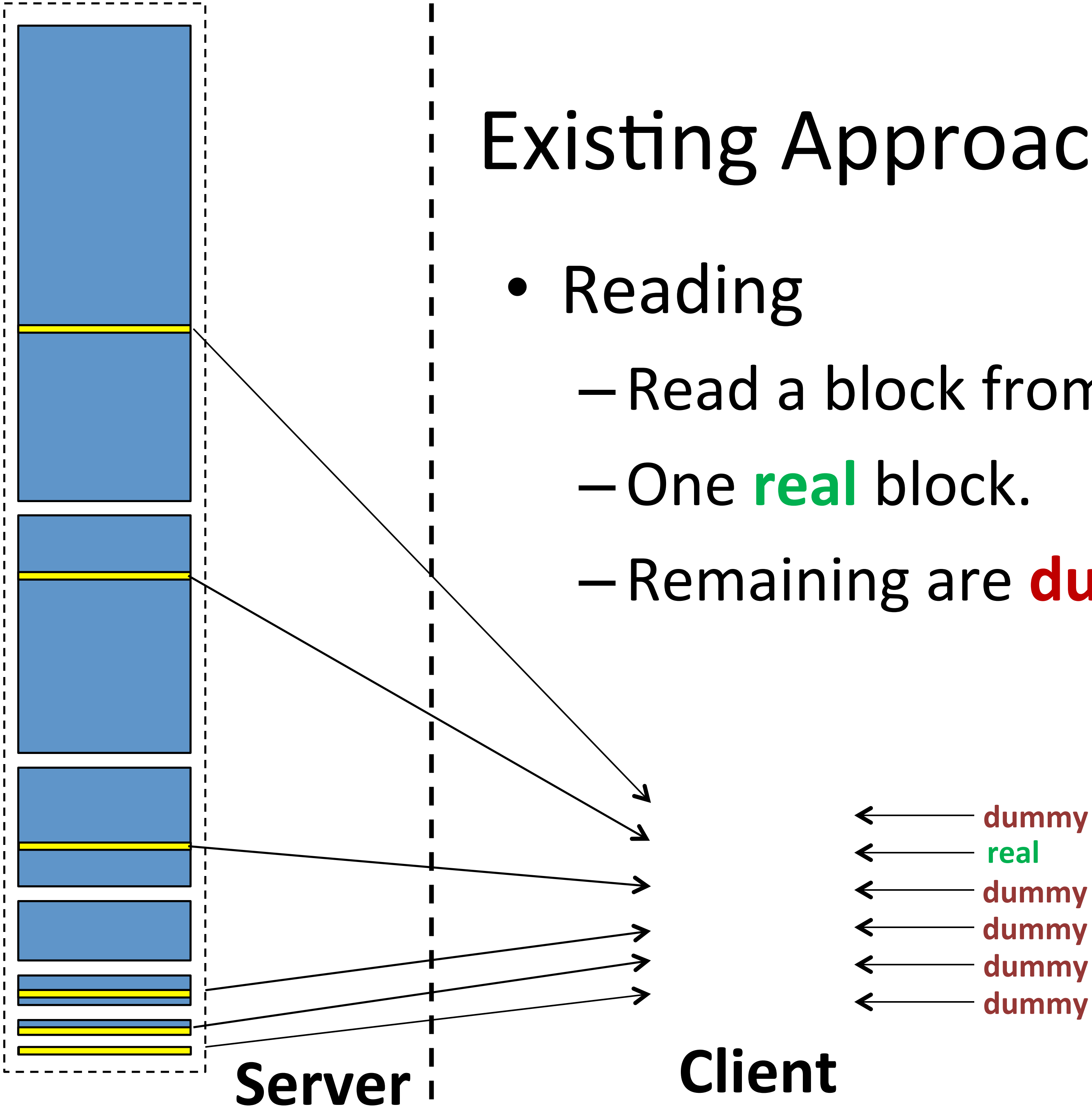
Existing Approaches



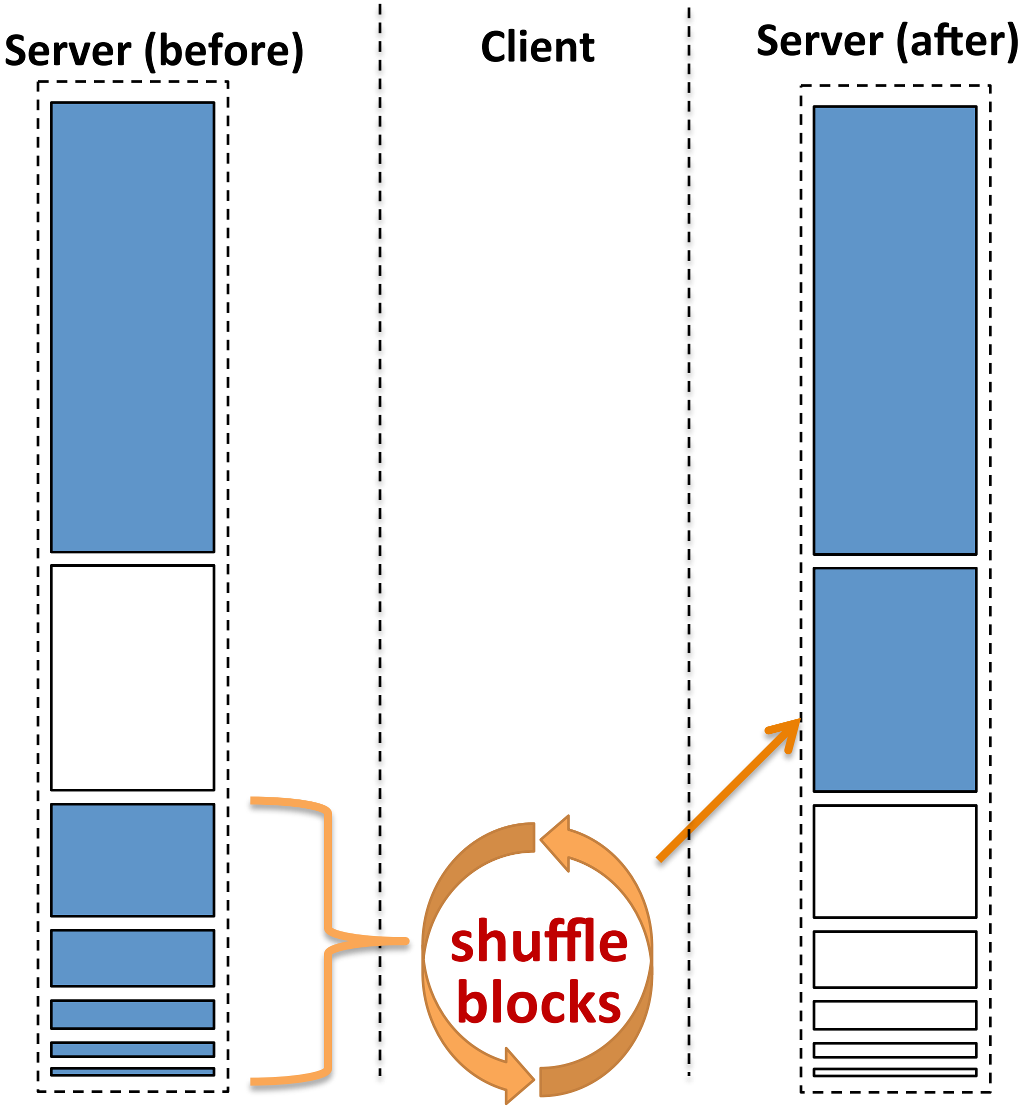
- Inside a level
  - Some real blocks
    - Useful data
  - Some dummy blocks
    - Random data
  - Randomly permuted
    - Only the client knows the permutation

Existing Approaches

- Reading
  - Read a block from each level
  - One **real** block.
  - Remaining are **dummy** blocks

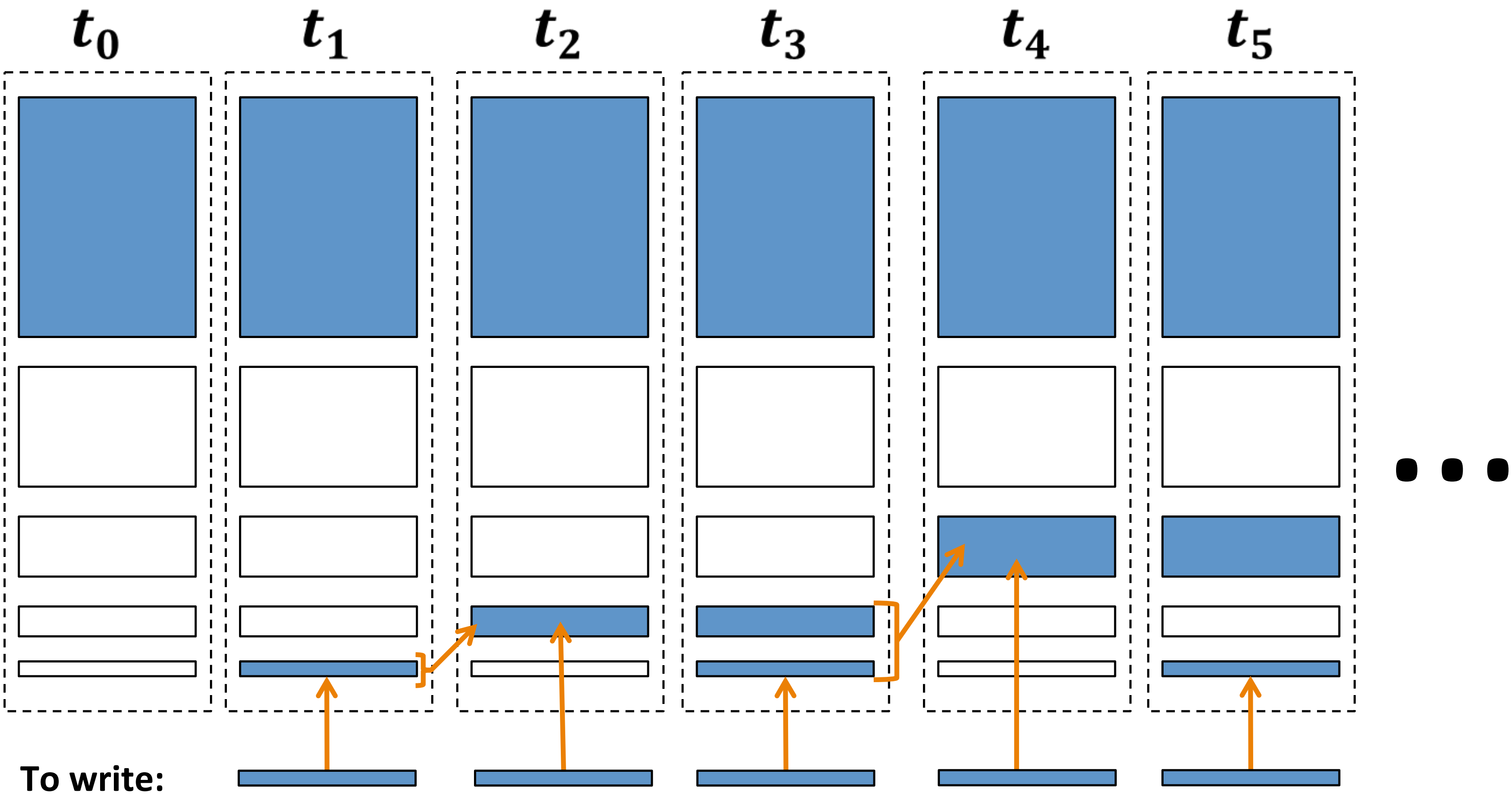


Existing Approaches



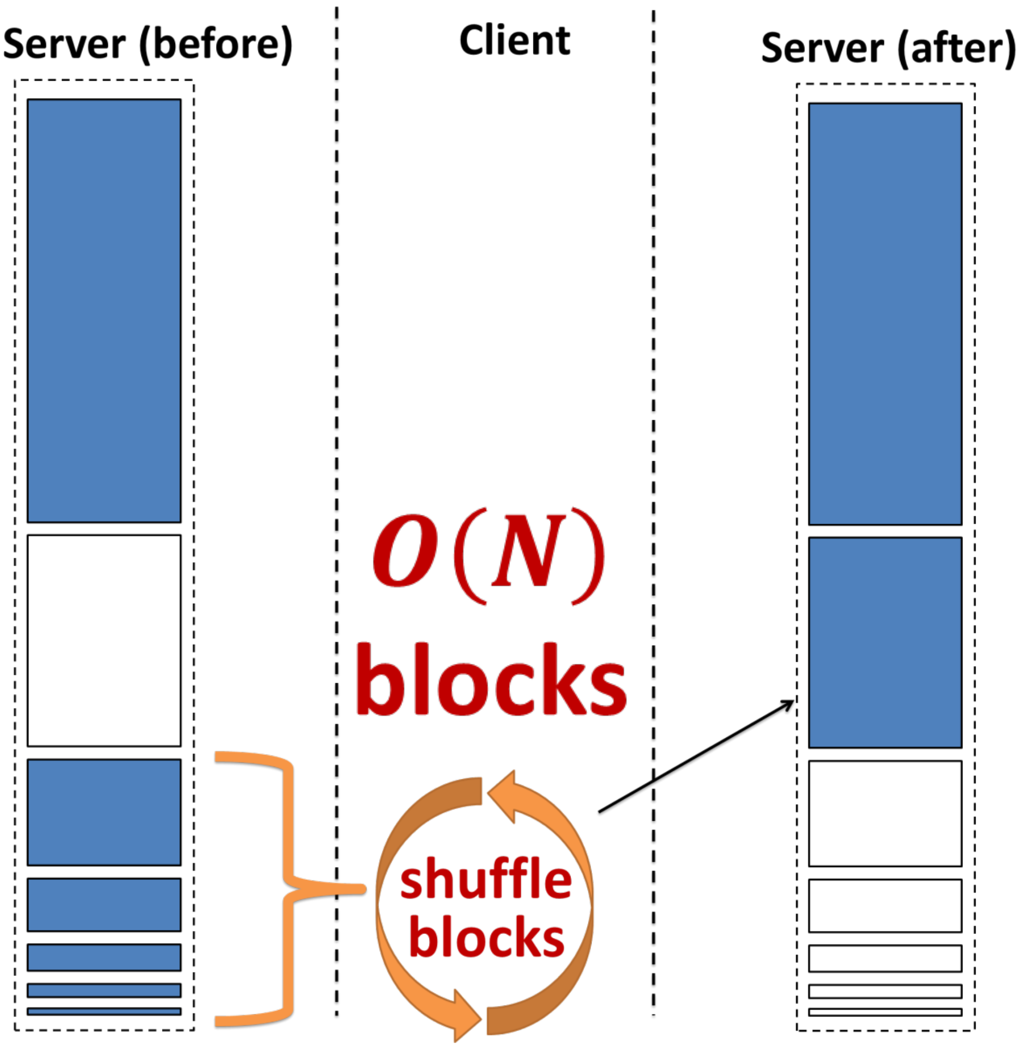
- Writing
  - Shuffle consecutively filled levels.
  - Write into next unfilled level.
  - Clear the source levels

Continuous Shuffling



- Cost per operation (amortized): $O(\log N)$ or $O(\log^2 N)$
  - Depending on shuffling algorithm

The Problem with Existing Approaches



- Writing is expensive.
- Sometimes need to shuffle $O(N)$ blocks.
- Cannot store them all locally.
- Needs oblivious shuffling algorithm.
  - Very expensive!

A practical Approach [SSS12]

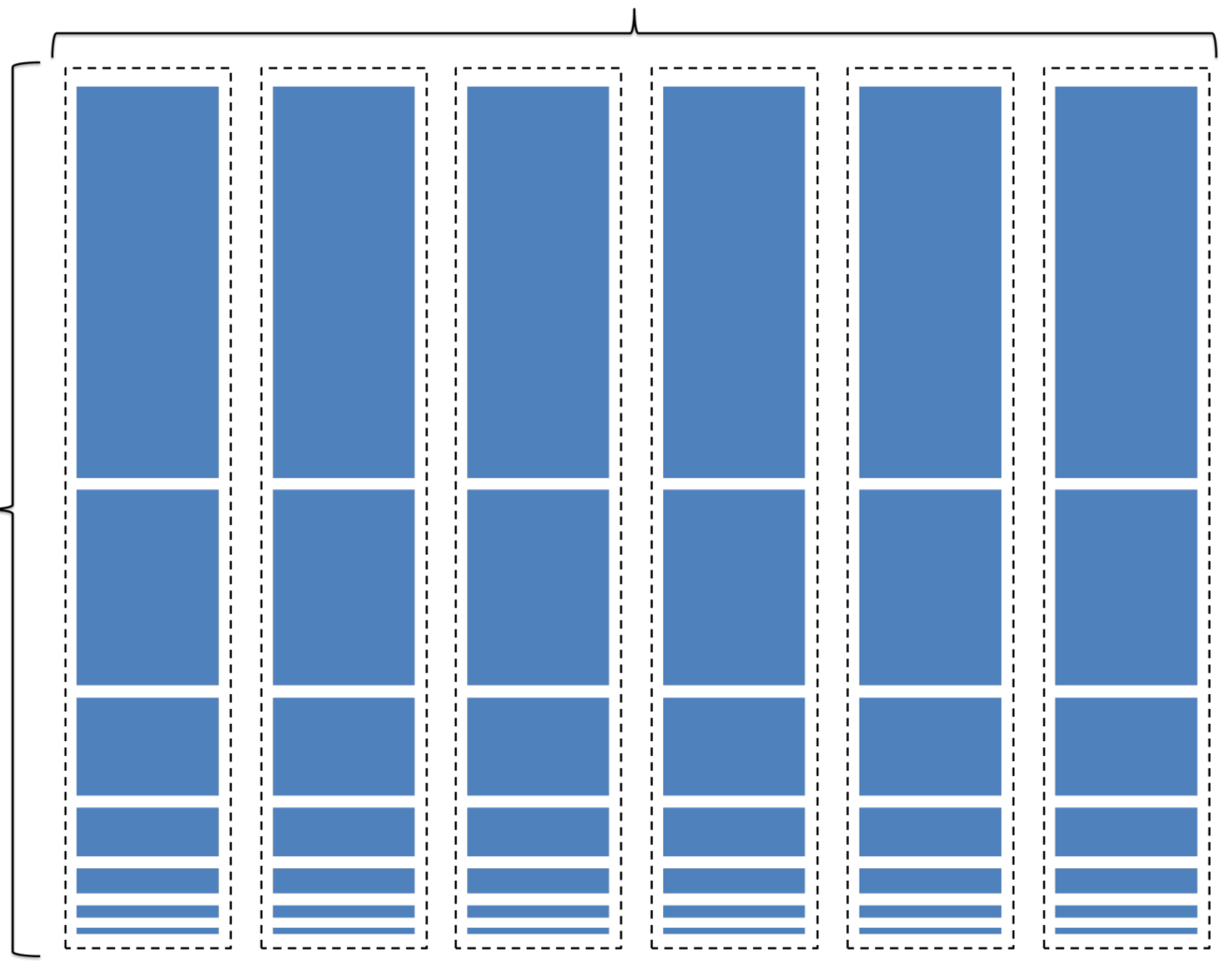
- Make shuffling cheaper.
- Reduce the worst-case cost.

But, how?

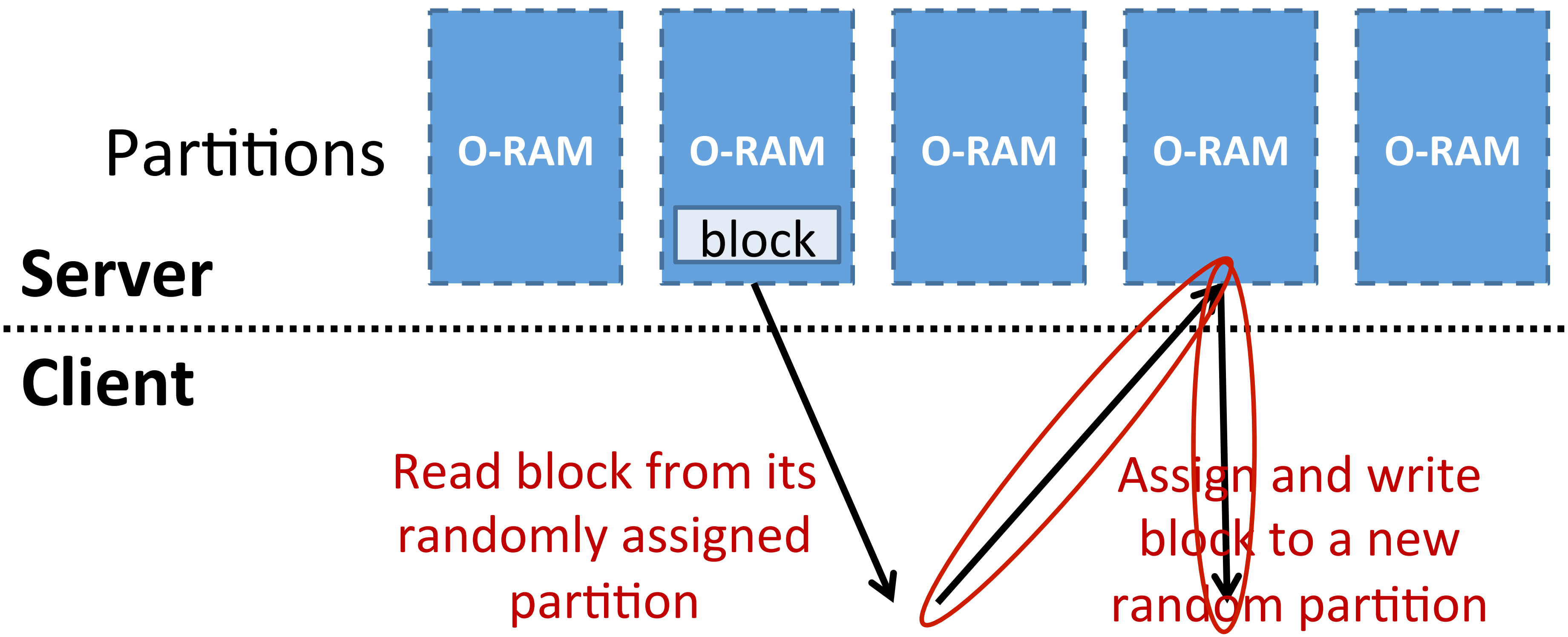
Answer: Partition the Storage

$\sqrt{N}$ partitions

Partitions of size $O(\sqrt{N})$



Challenge: Partitioning Breaks Security

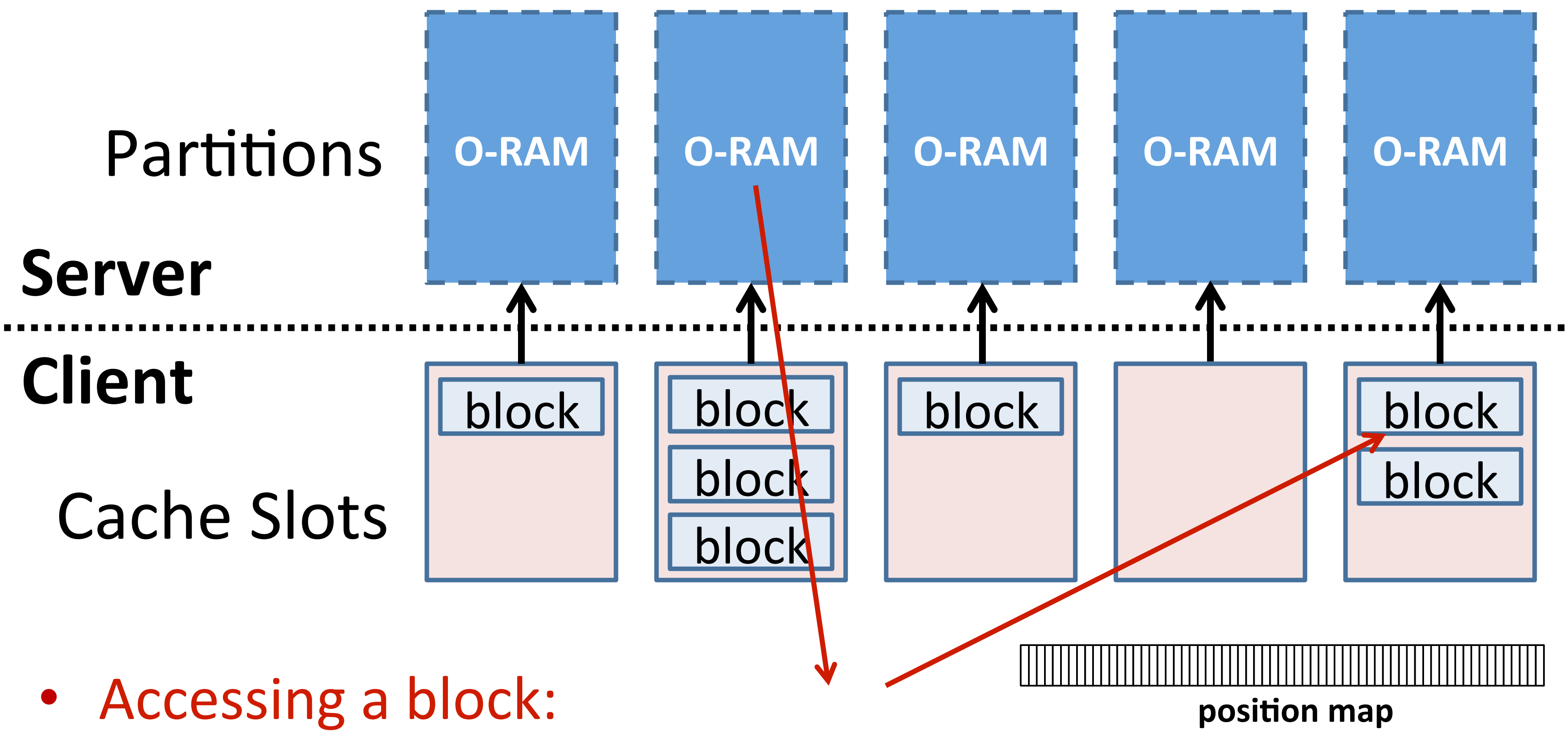


Not privacy preserving!

Read block from its previously assigned random partition

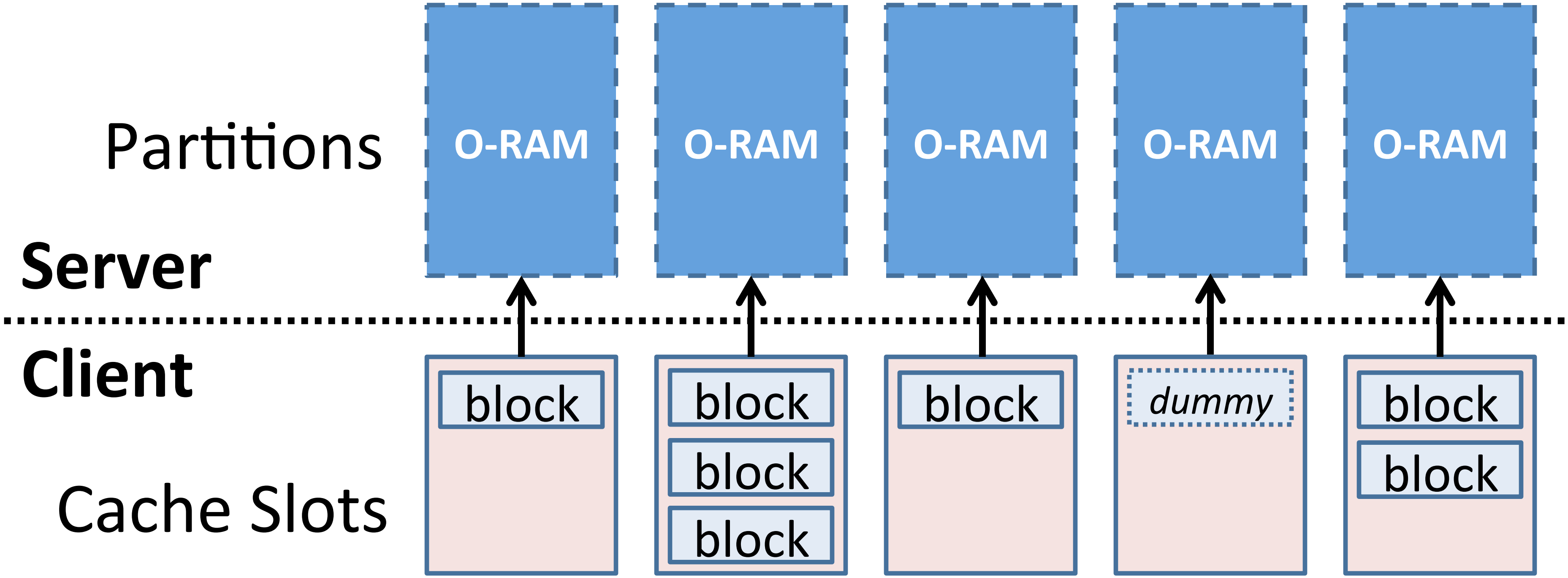
There is **linkability** between reads and writes.

Solution: Partitioning Framework



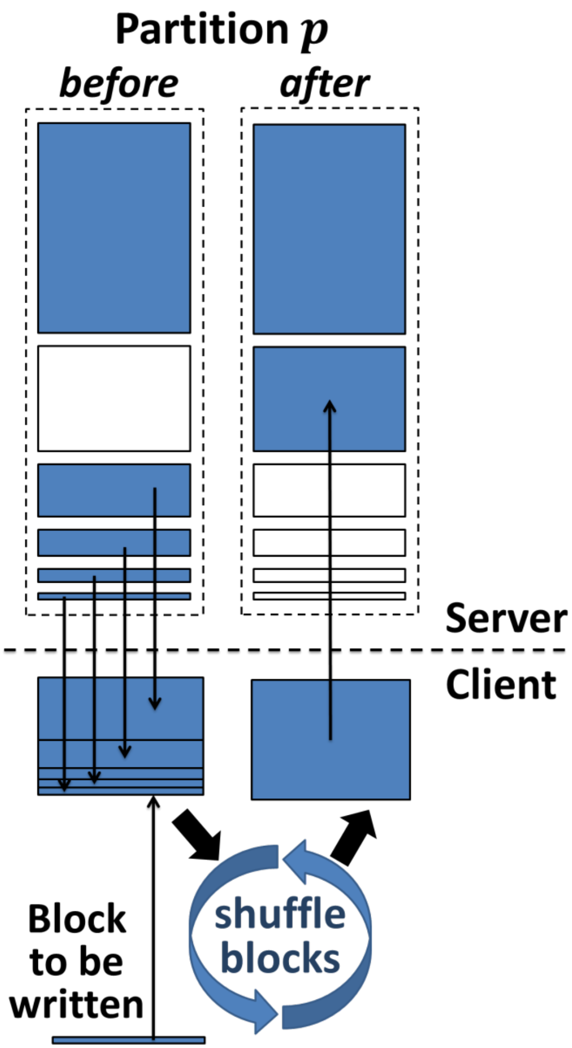
- Accessing a block:
  1. Read from partition (previously randomly assigned).
  2. Read/modify block data.
  3. Write to random cache slot (don't write to server yet).

Solution: Partitioning Framework



- Background eviction:
  - Sequentially scan the cache slots.
  - Evict one block if possible.
  - Evict dummy block otherwise.

Partition O-RAM



- Local shuffling
  - ***No expensive oblivious shuffling.***
- No cuckoo hashing.
  - ***2X speedup***
- Matrix compression algorithm for uploading levels
  - ***1.5X speedup***
- Constant latency:
  - $O(\log N)$ ➔ ***1 round trip***

Path ORAM
[SVSF13]
CCS13

Path ORAM

- Problem statement:
  - Client wishes to store data at a remote untrusted server while preserving its privacy
  - Server is untrusted, and the client is trusted, including the client's processor, memory, and disk
- No information should be leaked about:
  - Which data is being accessed
  - When was it last accessed
  - Whether the same data is being accessed
  - Access pattern
  - Whether the access is read or write

Goals

- **Small client storage**
  - Constant or logarithmic
  - Not $O(\sqrt{N})$
- **Simple**
  - No oblivious sorting, oblivious cuckoo hash table construction, etc
- **Performance**
  - Improved asymptotic bounds
  - Small constants → practical

Security Definition

- Privacy:
  - For any two data request sequences of the same length, their access patterns are computationally indistinguishable by anyone but the client

- Correctness:
  - ORAM construction is correct in that it returns the requested data with high probability

Path ORAM - Overview

- Client stores a small amount of local data in a stash
- Server-side storage is treated as a binary tree where each node is a bucket that can hold up to a fixed number of blocks
- Each block is mapped to a uniformly random leaf bucket in the tree
- Unstashed blocks are always placed in some bucket along the path to the mapped leaf

Path ORAM – Read/Write

- When a block is read from the server, the entire path to the mapped leaf is read into the stash

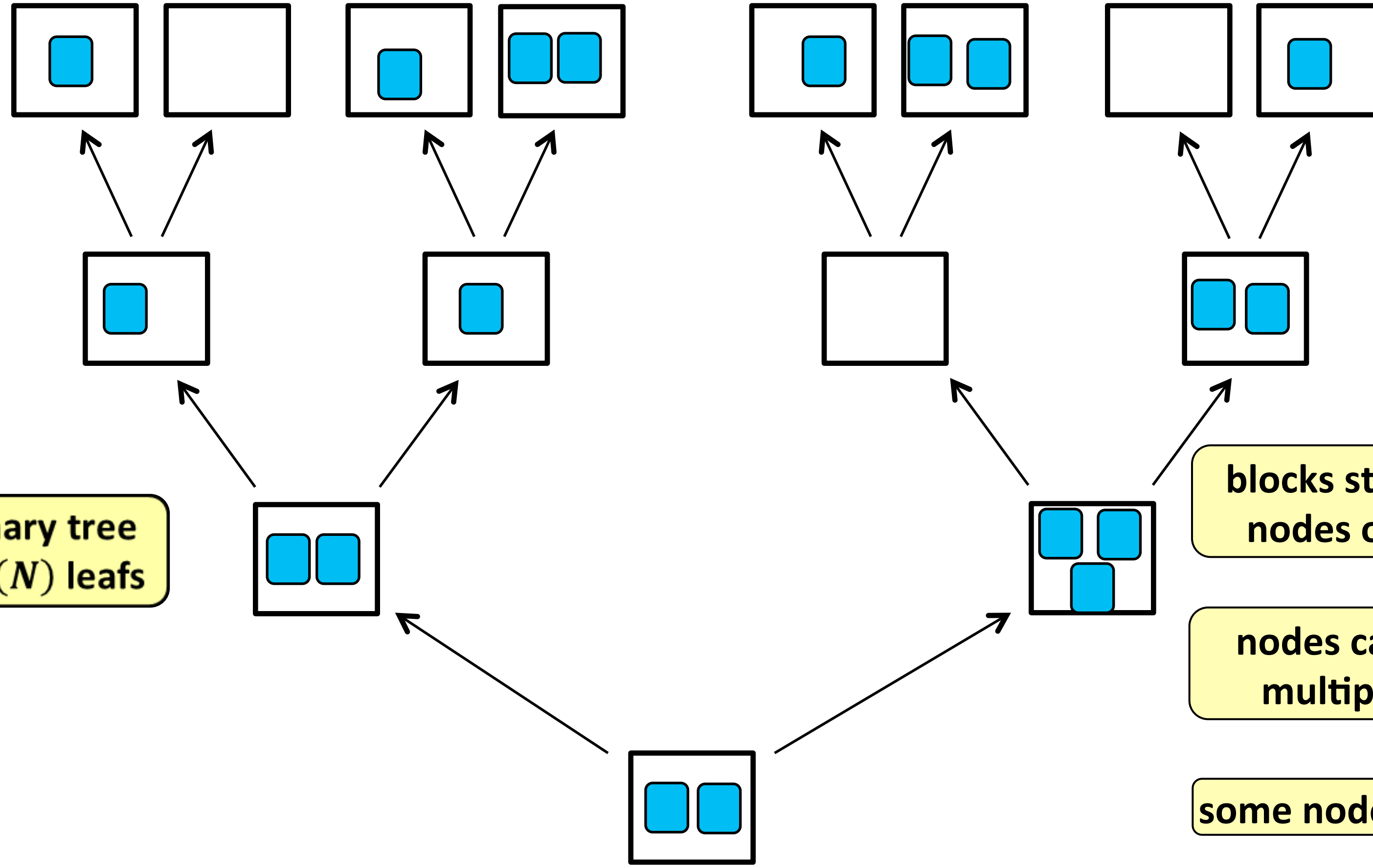
- Requested block is remapped to another leaf, and then the path that was just read is written back to the server full binary tree with $O(N)$ leafs

blocks stored in the nodes of the tree

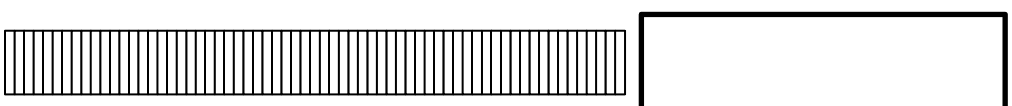
nodes can contain multiple blocks

some nodes are empty

Server



Client

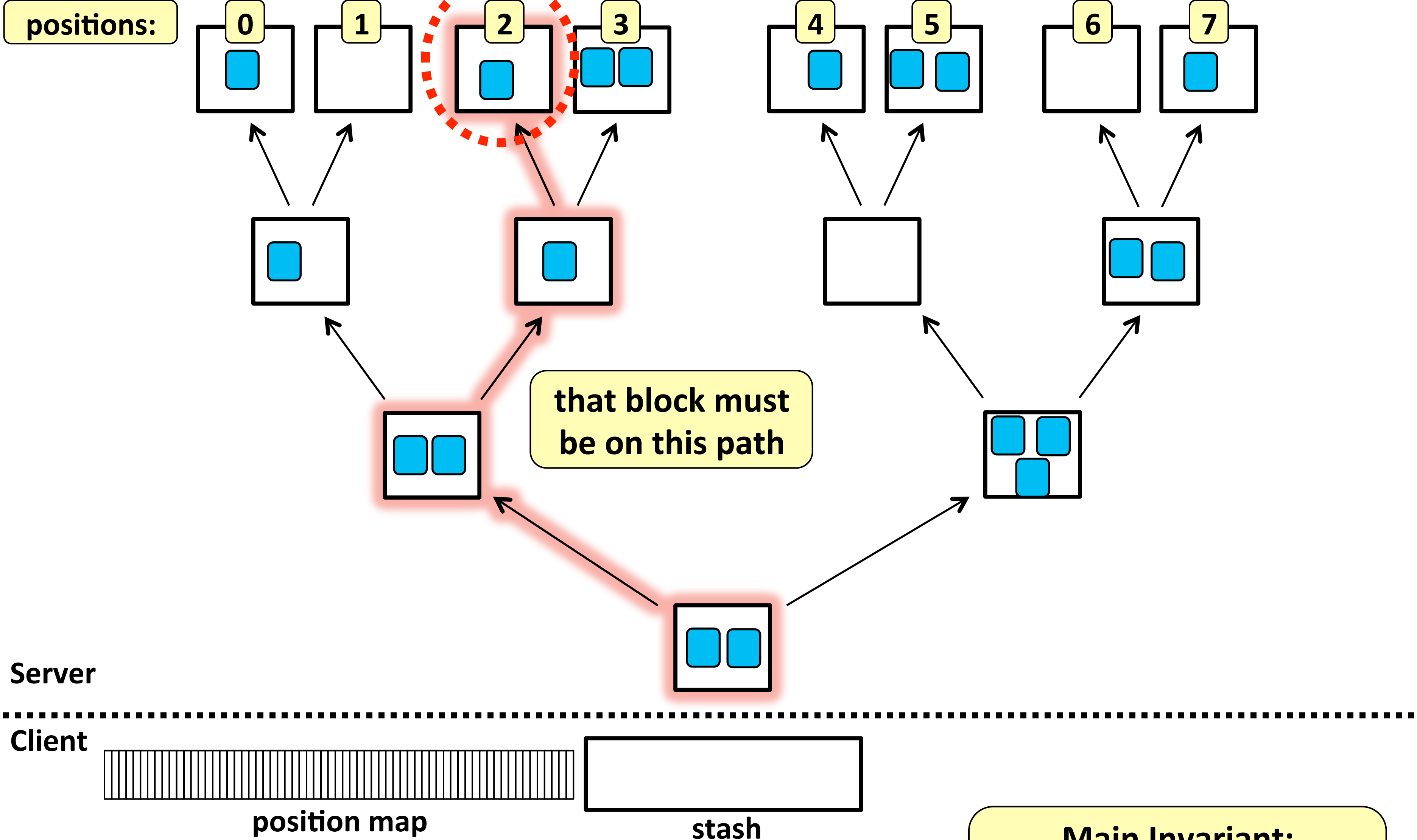


array that maps:
block id → random leaf (position)

stores blocks not currently in the tree

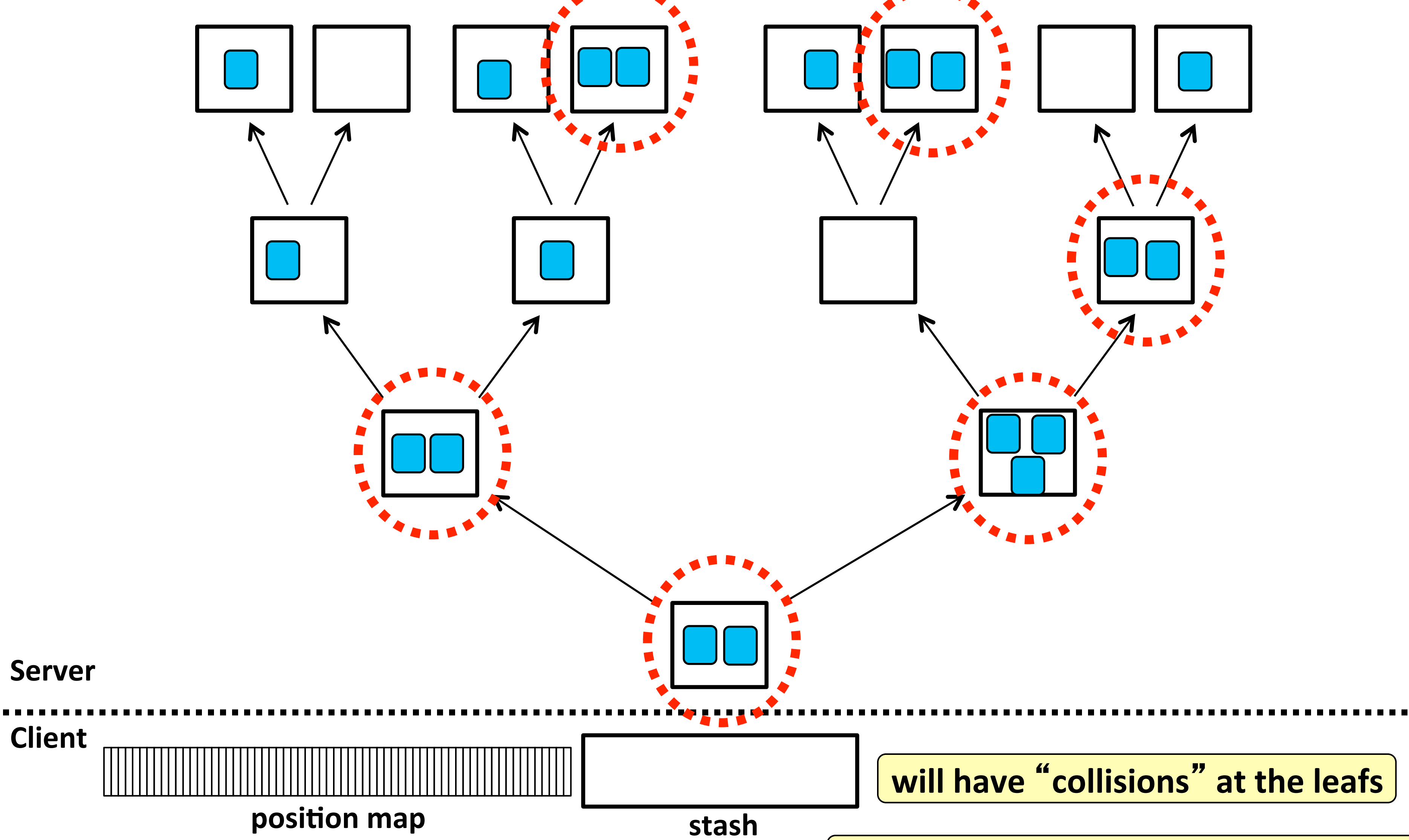
Main Invariant:
A block is always on the path to its "position".

$\log N$ bits per block



Example:
A block is mapped to position 2.

<u>Main Invariant:</u>
A block is always on the path to its "position".

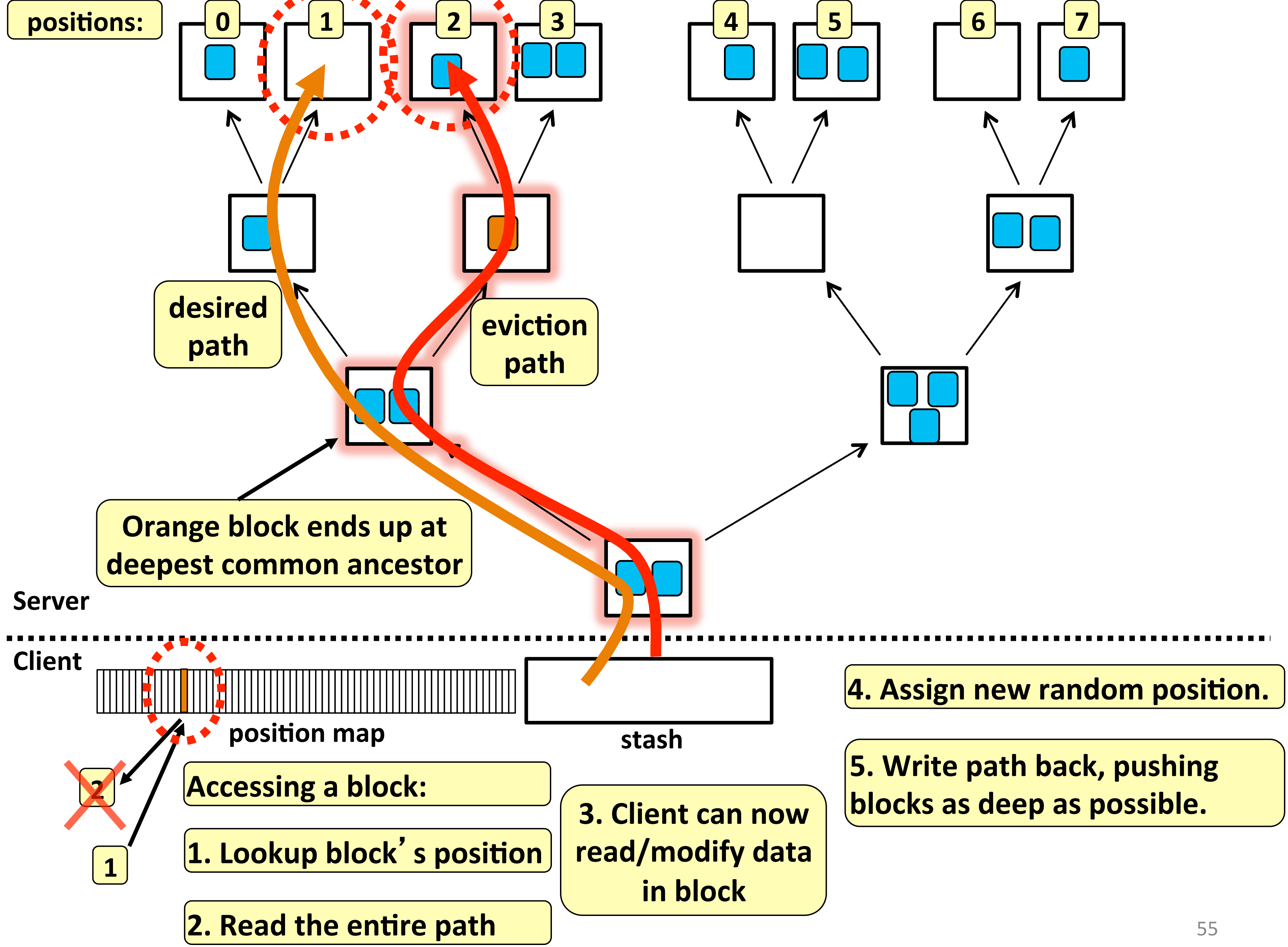


will have "collisions" at the leafs

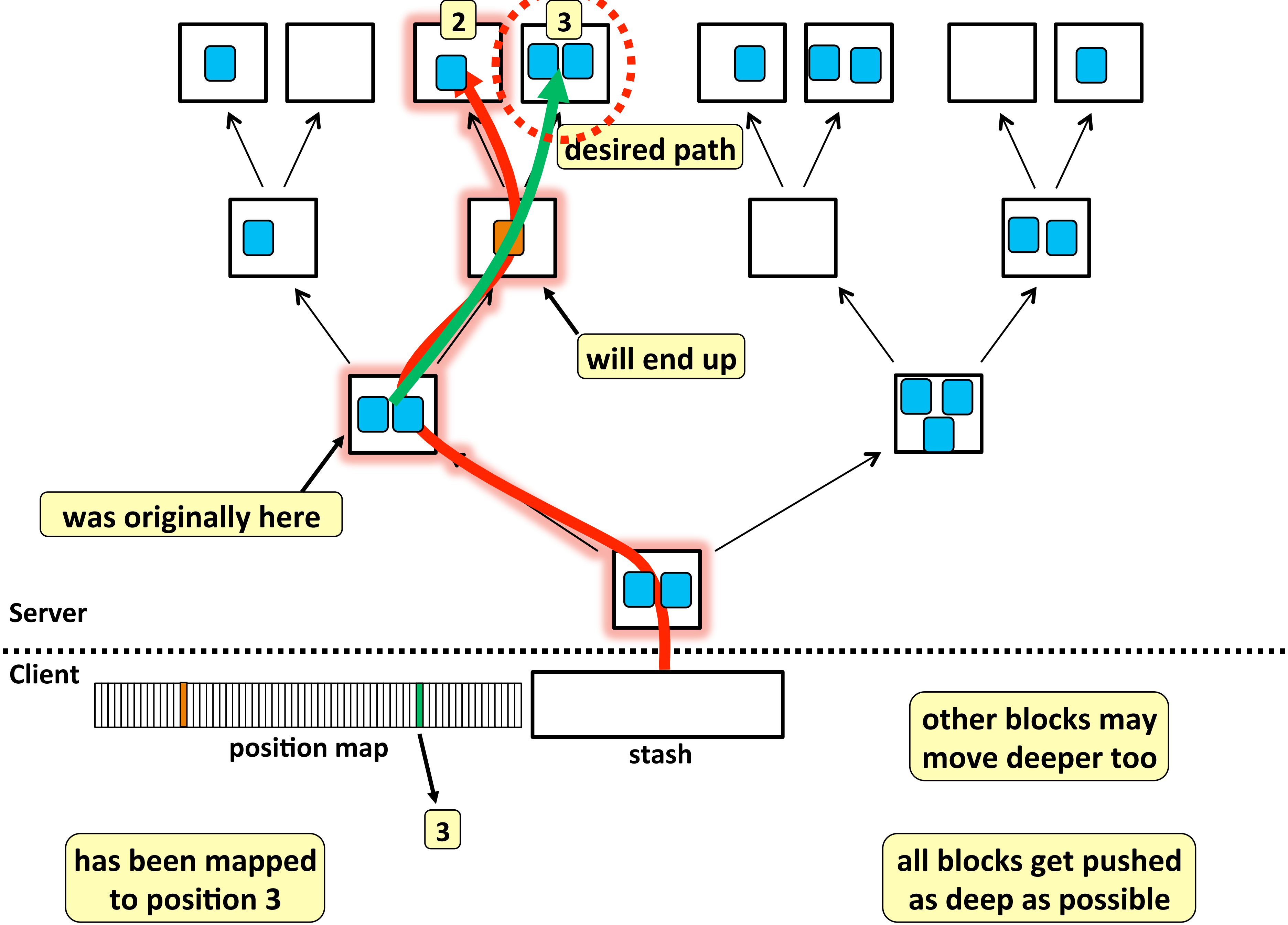
will have "collisions" at internal nodes

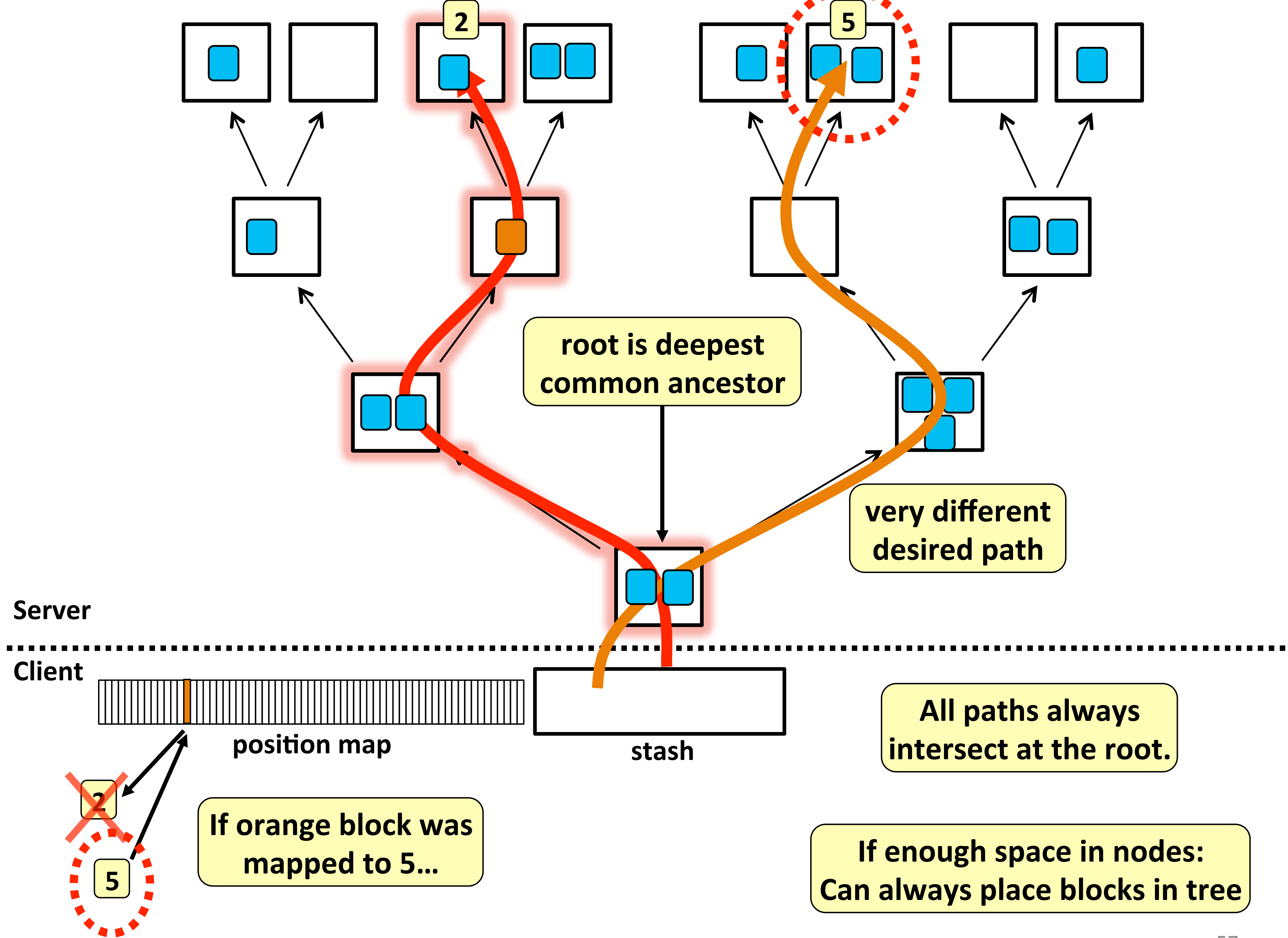
positions are uniformly independently randomly chosen

this is why some nodes might contain multiple blocks



positions:
0
1
2
3
4
5
6
7
desired path
eviction path
Orange block ends up at deepest common ancestor
Server
Client
position map
stash
2
1
Accessing a block:
1. Lookup block's position
2. Read the entire path
3. Client can now read/modify data in block
4. Assign new random position.
5. Write path back, pushing blocks as deep as possible.

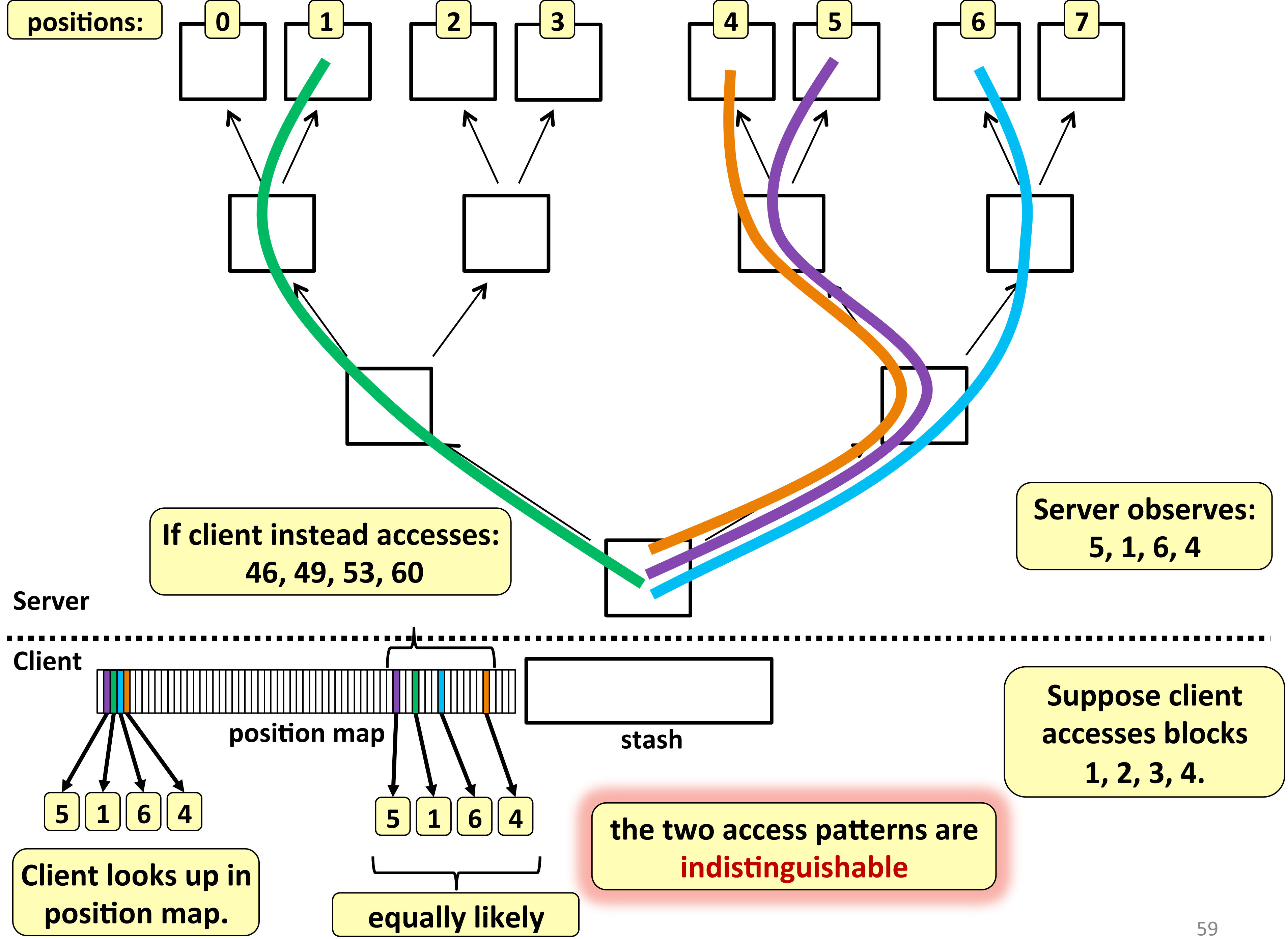

2
3
desired path
will end up
was originally here
Server
Client
position map
stash
3
other blocks may move deeper too
has been mapped to position 3
all blocks get pushed as deep as possible



2
5
root is deepest common ancestor
very different desired path
Server
Client
position map
stash
All paths always intersect at the root.
2
5
If orange block was mapped to 5…
If enough space in nodes: Can always place blocks in tree

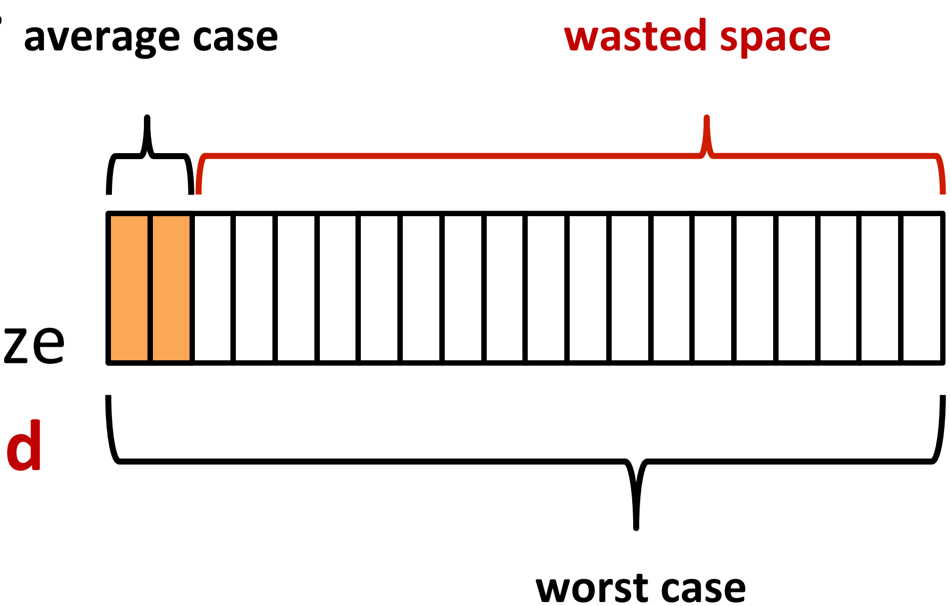
Security

- Why is this secure?
- Nodes are padded to fixed size
  - E.g., each node can contain up to Z blocks.
    - For Z large enough.
  - Nodes encrypted → server can't tell # blocks in node
- What is revealed to the server when block accessed?
  - Only the path, nothing else.
  - Determined entirely block's position
  - Positions are always uniformly independently random.
  - Block's new position is not revealed.
  - **Server only observes a single random number on each access.**
  - **Independent of the access pattern.**


positions:
0
1
2
3
4
5
6
7
If client instead accesses:
46, 49, 53, 60
Server observes:
5, 1, 6, 4
Server
Client
position map
stash
Suppose client accesses blocks 1, 2, 3, 4.
5
1
6
4
5
1
6
4
Client looks up in position map.
equally likely
the two access patterns are indistinguishable

Node Size

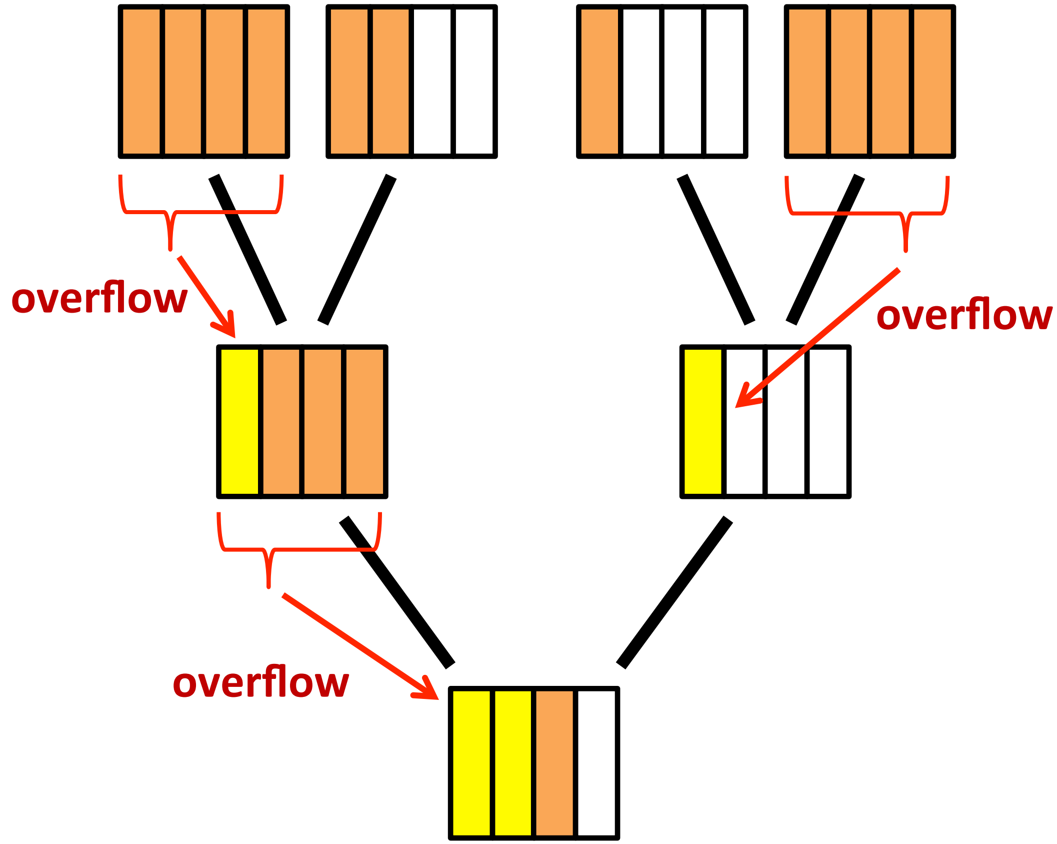
- **To preserve privacy:**
  - Must hide # blocks in node.
  - Must use worst-case size.
  - Pad with dummy blocks.
- **Problem:**
  - Large worst case size
  - Much smaller average size
  - **Lots of wasted space and bandwidth.**



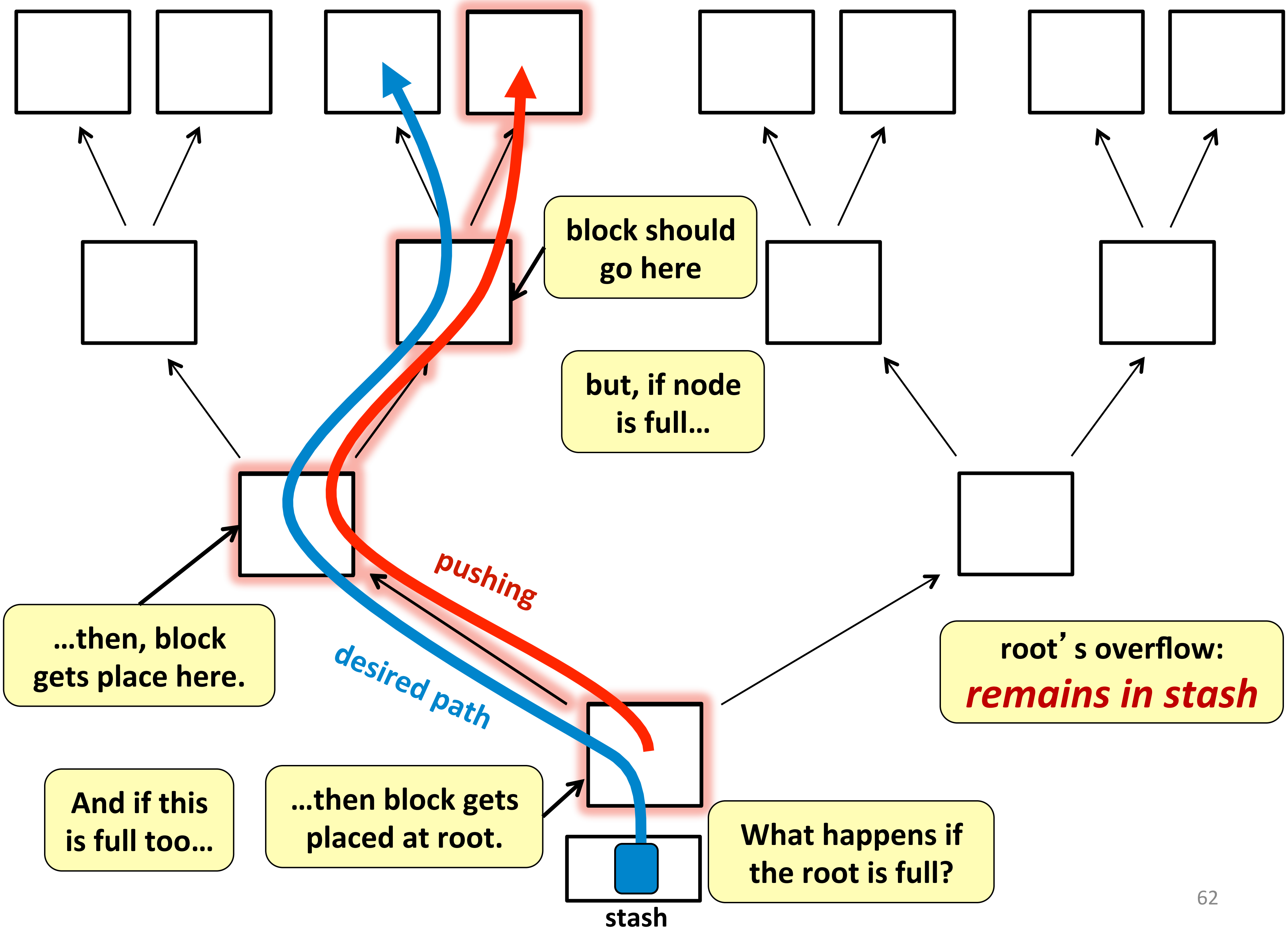
Node Size

Our solution:

- **Use small node size.**
  - Node size of 4 works well.
- **Children overflow into lower levels.**



Overflow preserves main (path) invariant.

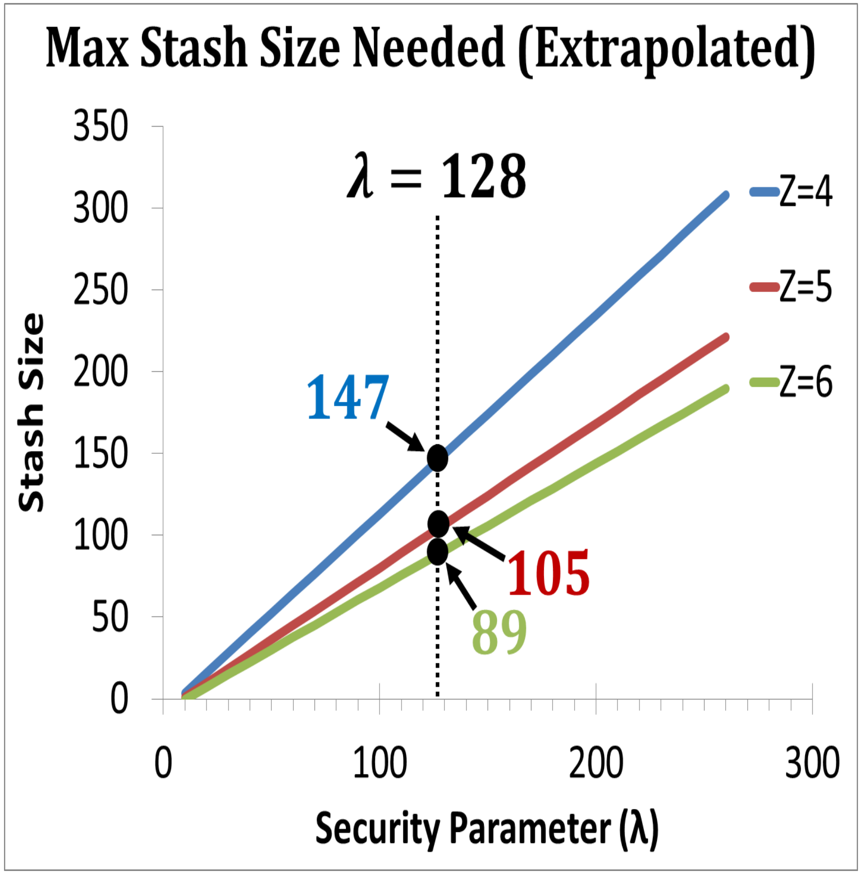


block should go here
but, if node is full…
pushing
…then, block gets place here.
desired path
root’s overflow:
remains in stash
And if this is full too…
…then block gets placed at root.
What happens if the root is full?
stash

Client’s Stash

Empirical estimation of the required stash size to achieve failure probability less than $2^{-\lambda}$

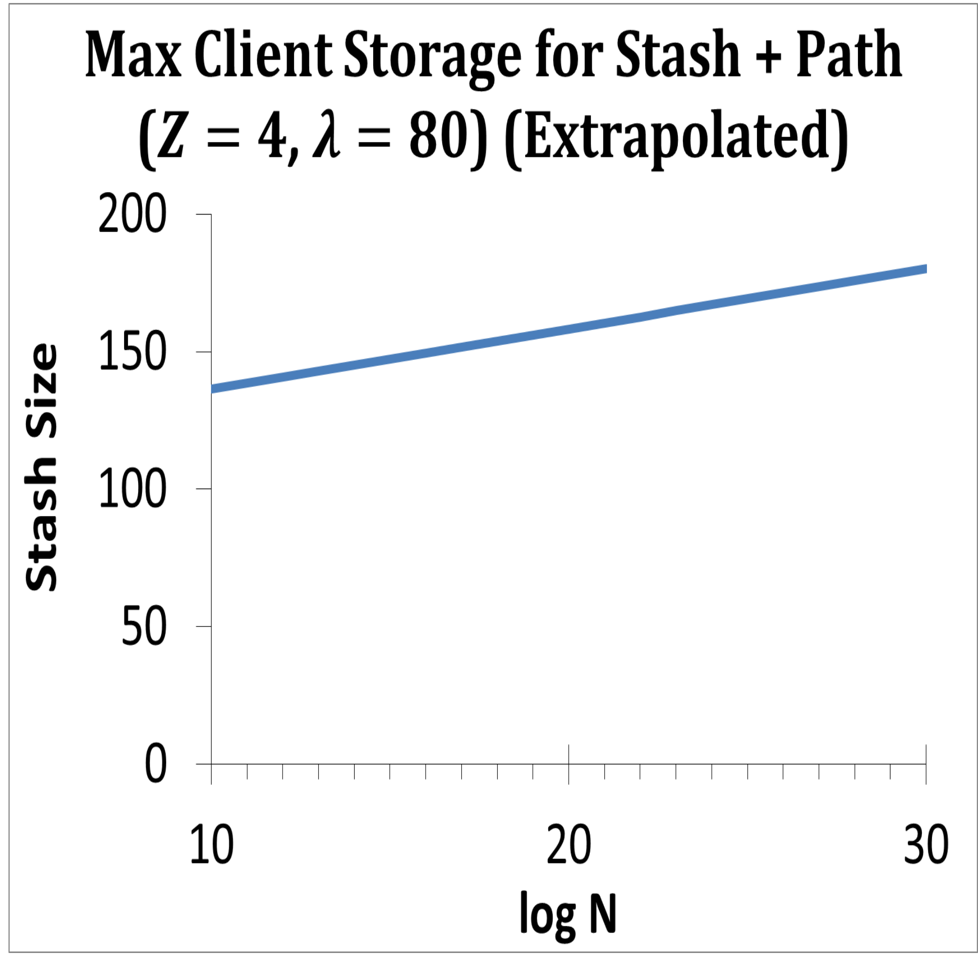
- **Very small**
  - With high probability
  - Usually empty after writing path.
- $\boldsymbol{O(\lambda)}$ **blocks**
  - λ: security parameter.
  - Does not depend on N.
- **Works well for small nodes**
  - Node size $Z = 4$ works well.
  - For $Z \geq 5$, small improvements in stash size.



Extrapolated based on $\lambda \leq 26$

Client Storage

- **Stash:**
  $O(\lambda)$ blocks
- **Path:**
  $O(\log N)$ blocks
- **Stash + Path:**
  $O(\lambda + \log N)$ blocks
- **Position map:**
  $\log N$ bits per block
  ➔ $O(N)$, Need to reduce it!

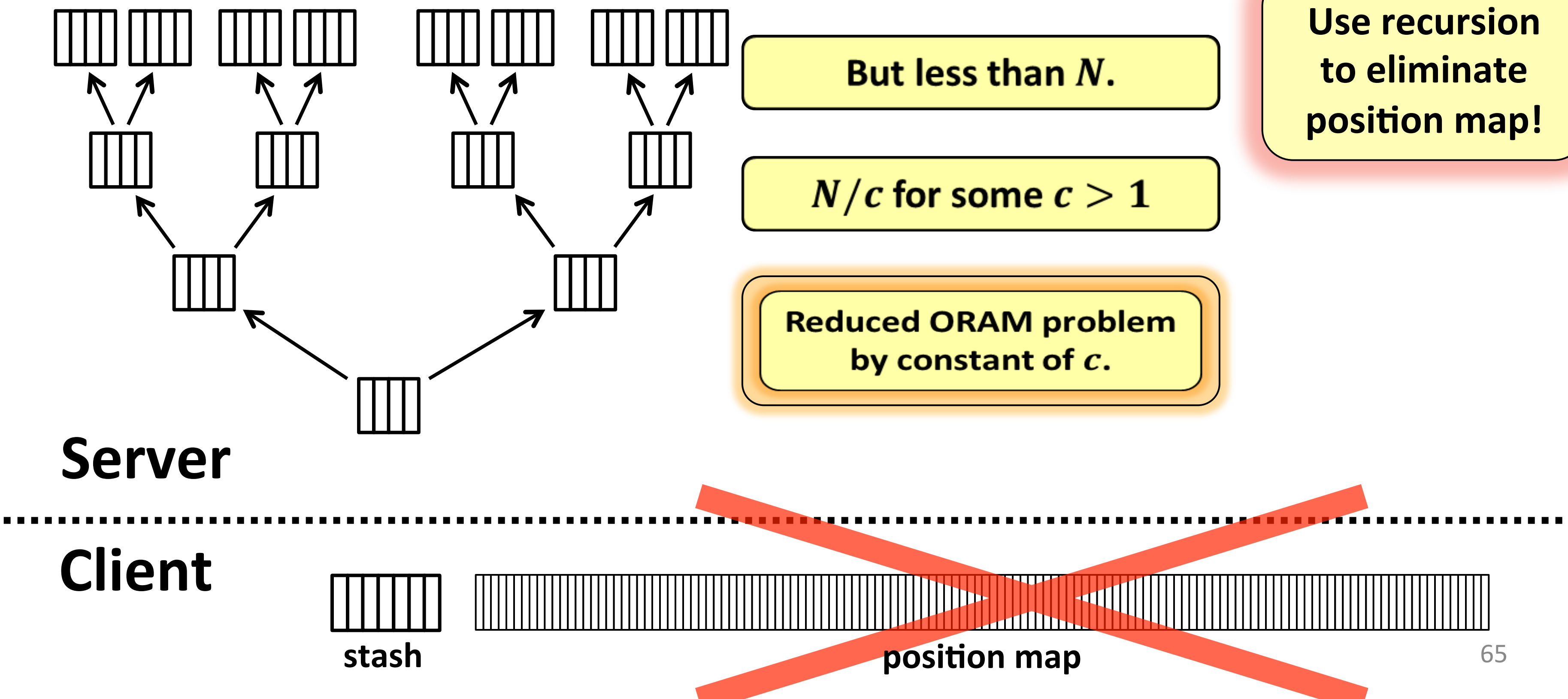


Extrapolated based on $\log N \leq 21$ and $\lambda \leq 22$

Results from empirical evaluation for the Phantom processor by [MLS13].

Position Map

- So far, the data is laid out as follows:



position map: $O(N)$ size

But less than N.

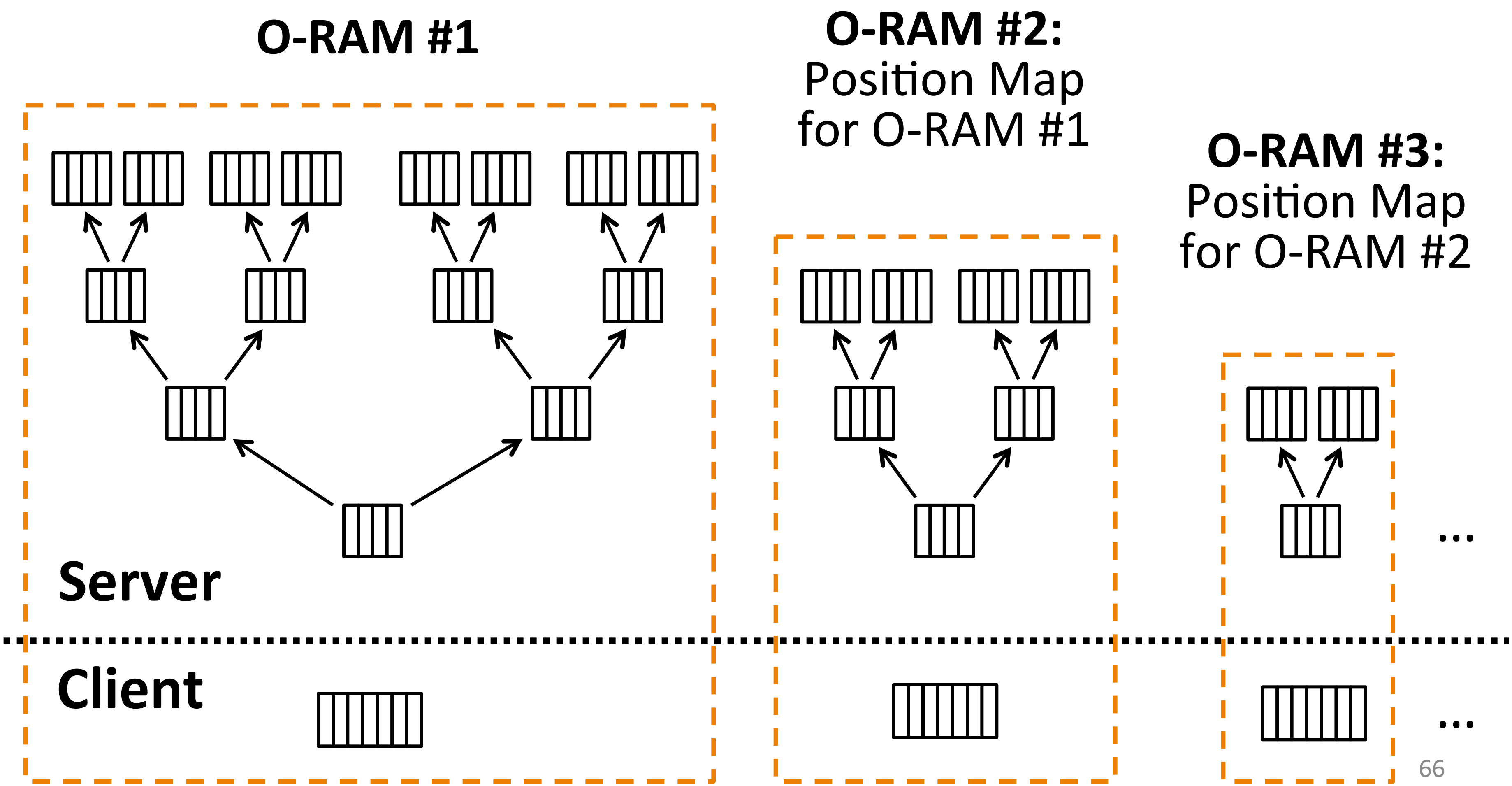
N/c for some $c > 1$

Reduced ORAM problem by constant of c.

Use recursion to eliminate position map!

Recursion

- Store the position map in another ORAM.
- Do this recursively.



Asymptotic Costs

- **Asymptotic breakthrough** for large enough blocks (e.g., 4 KB blocks).

Instantiation	Client Storage	Bandwidth
Without Recursion	$O(N)$	$O(\log N)$
With Recursion	$\sim O\left(\frac{(\log N)^2}{\log X}\right)$	$O\left(\frac{(\log N)^2}{\log X}\right)$

$$B = X \log N$$

Block size in bits

Tot # of blocks

Stateless ORAM

- ORAM is often considered in a single-client model
- It is sometimes useful to have multiple clients accessing the same ORAM
- Goodrich et al. introduce the concept of stateless ORAM [1]
  - client state is small enough
  - any client accessing the ORAM can download it before each data access and upload it afterwards

[1] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Privacy-preserving group data access via stateless oblivious ram simulation. In Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '12,

PIR vs. ORAM

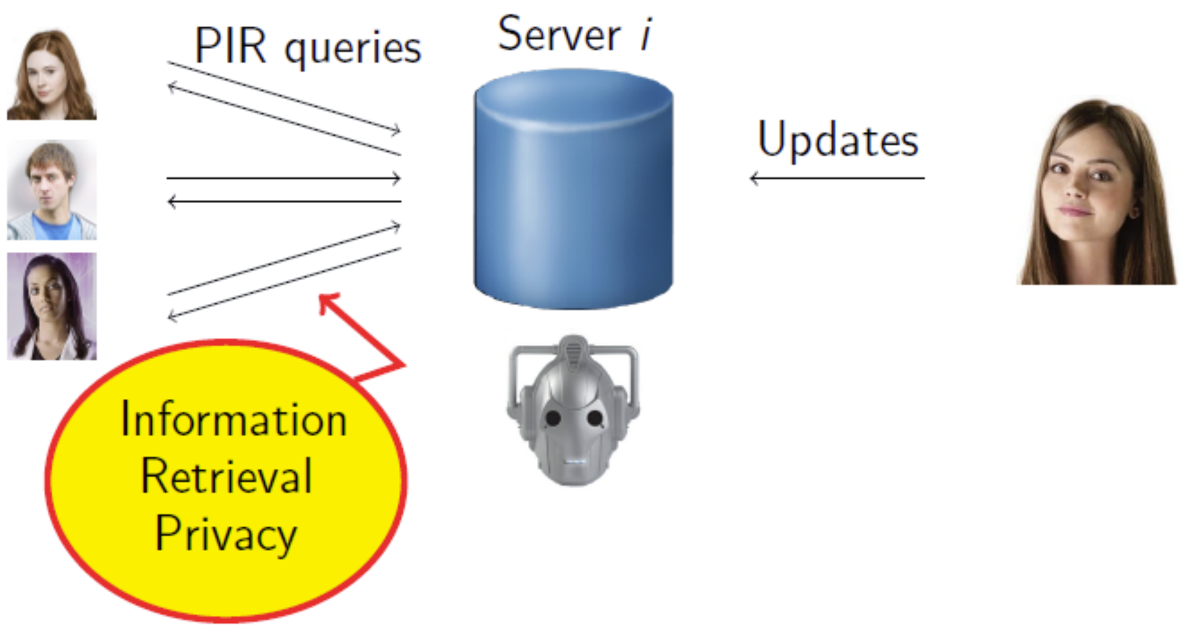
- PIR does not allow write operations on the database
- PIR scheme is single-rounded query-answer protocol (common communication pattern in context of databases)
- PIR allows multiple users to access the database
- Data in PIR is not necessarily encrypted.

- ORAM allows read/write operations on the database
- ORAM allows only the user with the crypto key to access the database
  - If several users, it requires them to coordinate their acts
- Data in ORAM is stored in encrypted form
- ORAM requires the user to keep some state information
- ORAM requires pre-processing initialization step.

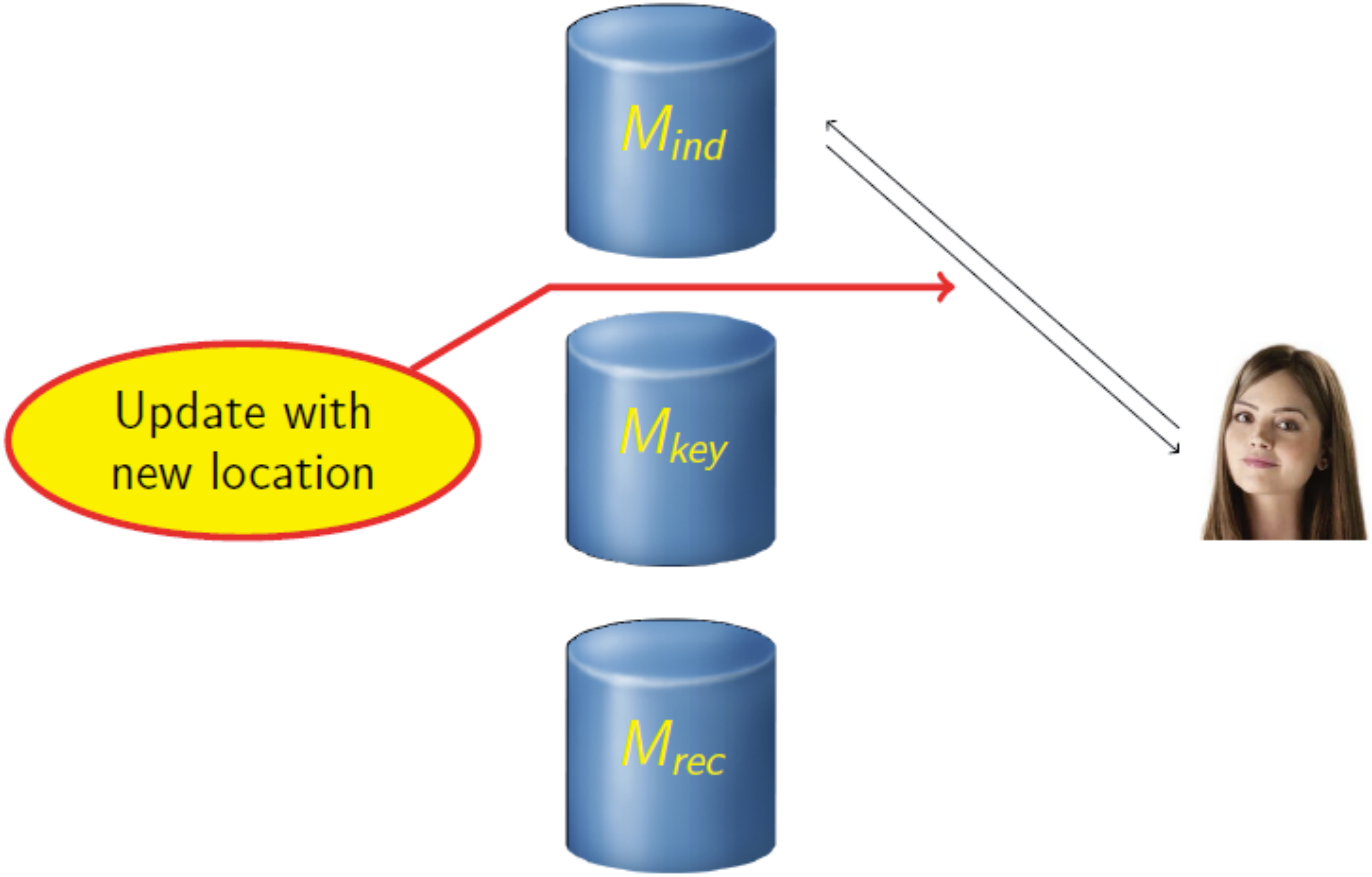
A Mixed Approach

[HG13] Outsourced Private Information Retrieval through combination of:

- Write-only ORAM
- Goldberg IT-PIR



The Construction



Takeaways

- Privacy protection against database owners is a growing concern especially with the spread of cloud computing
- Encryption is not always enough to protect data privacy since access patterns can reveal sensitive information
- Sophisticated techniques such as PIR and ORAM prevents the leakage of access patterns to the database owner
- However these techniques are computationally expensive and their usage must be evaluate carefully
- A lot of research is ongoing on this domain to make these techniques simpler and more efficient

References

[CKGS95] - Chor, B., Kushilevitz, E., Goldreich, O., and Sudan, M. . Private information retrieval. *Journal of the ACM (JACM), 1998*, *45*(6), 965-981.

[DGH12] - Devet, C., Goldberg, I., and Heninger, N. . Optimally Robust Private Information Retrieval. In *USENIX Security Symposium* (pp. 269-283). (2012, August).

[GO96] - Goldreich, O., and Ostrovsky, R.. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, *43*(3), 431-473. (1996).

[Gol07] - Goldberg, I.. Improving the robustness of private information retrieval. In *Security and Privacy, 2007. SP'07. IEEE Symposium on* (pp. 131-148). IEEE.

[GIKM98] - Gertner, Y., Ishai, Y., Kushilevitz, E., & Malkin, T.. Protecting data privacy in private information retrieval schemes. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing* (pp. 151-160). ACM 1998.

[HG13] - Huang, Y., and Goldberg, I.. Outsourced private information retrieval. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society* (pp. 119-130). ACM 2013.

References (ctd.)

[IKK12] - Islam, M. S., Kuzu, M., and Kantarcioglu, M. (2012, February). Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *NDSS'12*.

[KO97] - Kushilevitz, E., & Ostrovsky, R.. Replication is not needed: Single database, computationally-private information retrieval. IEEE Computer Society, 1997.

[MLS13] - Maas, M., et al.. Phantom: Practical oblivious computation in a secure processor. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 311-324). ACM.

[SSS12] - Stefanov, E., Shi, E., and Song, D. Towards practical oblivious RAM. *In Symposium on Network and Distributed System Security (NDSS), 2012*

[SVSF13] - Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., and Devadas, S.. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 299-310). ACM (2013, November)