How to *Privately* Access Remote Data

Rafail Ostrovsky (UCLA)

PRESENTATION AT NIST

November 8, 2011

Contact: rafail@cs.ucla.edu

Joint works/presentation credits:
Eyal Kushilevitz, Steve Lu, William E. Skeith III.

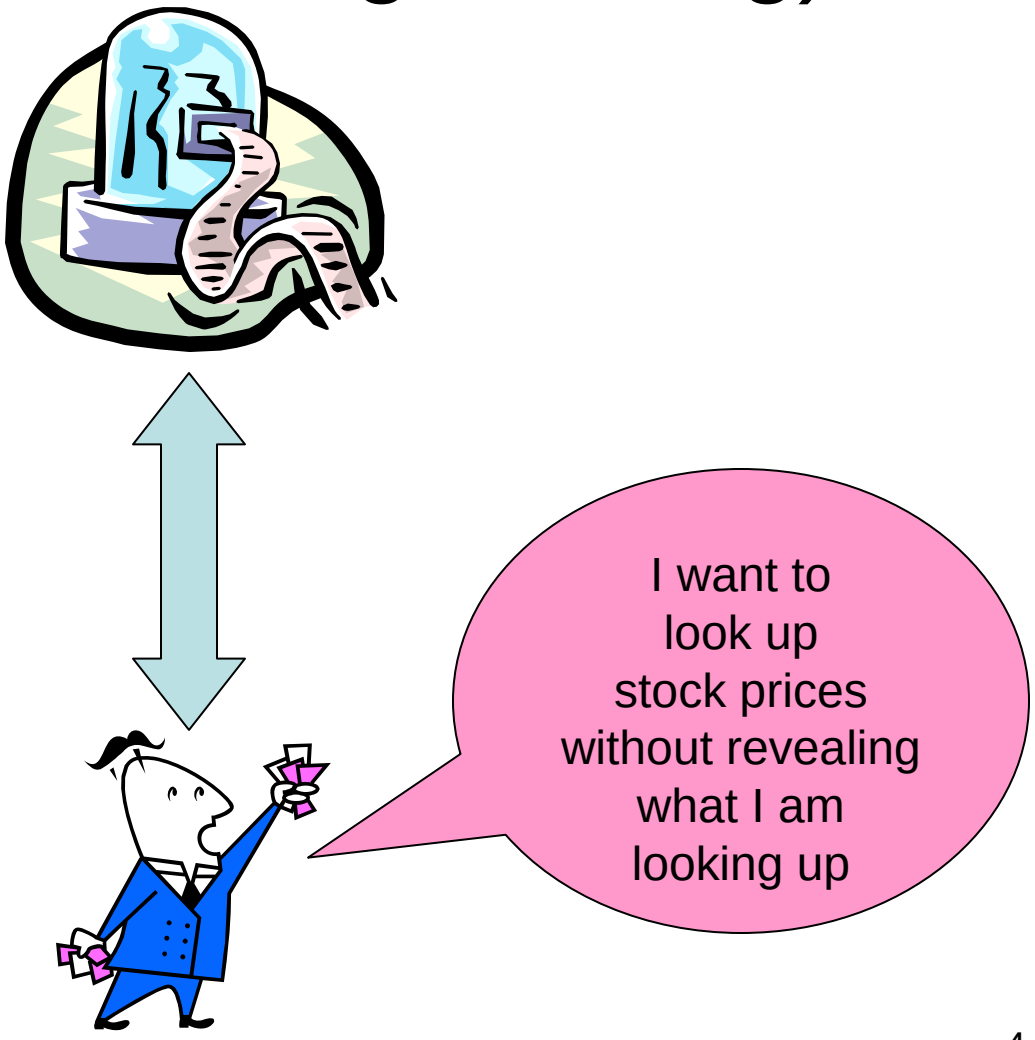
Overview

- Motivation
- Problem Statement
- Review of PIR and ORAM
- New Results
- Conclusion

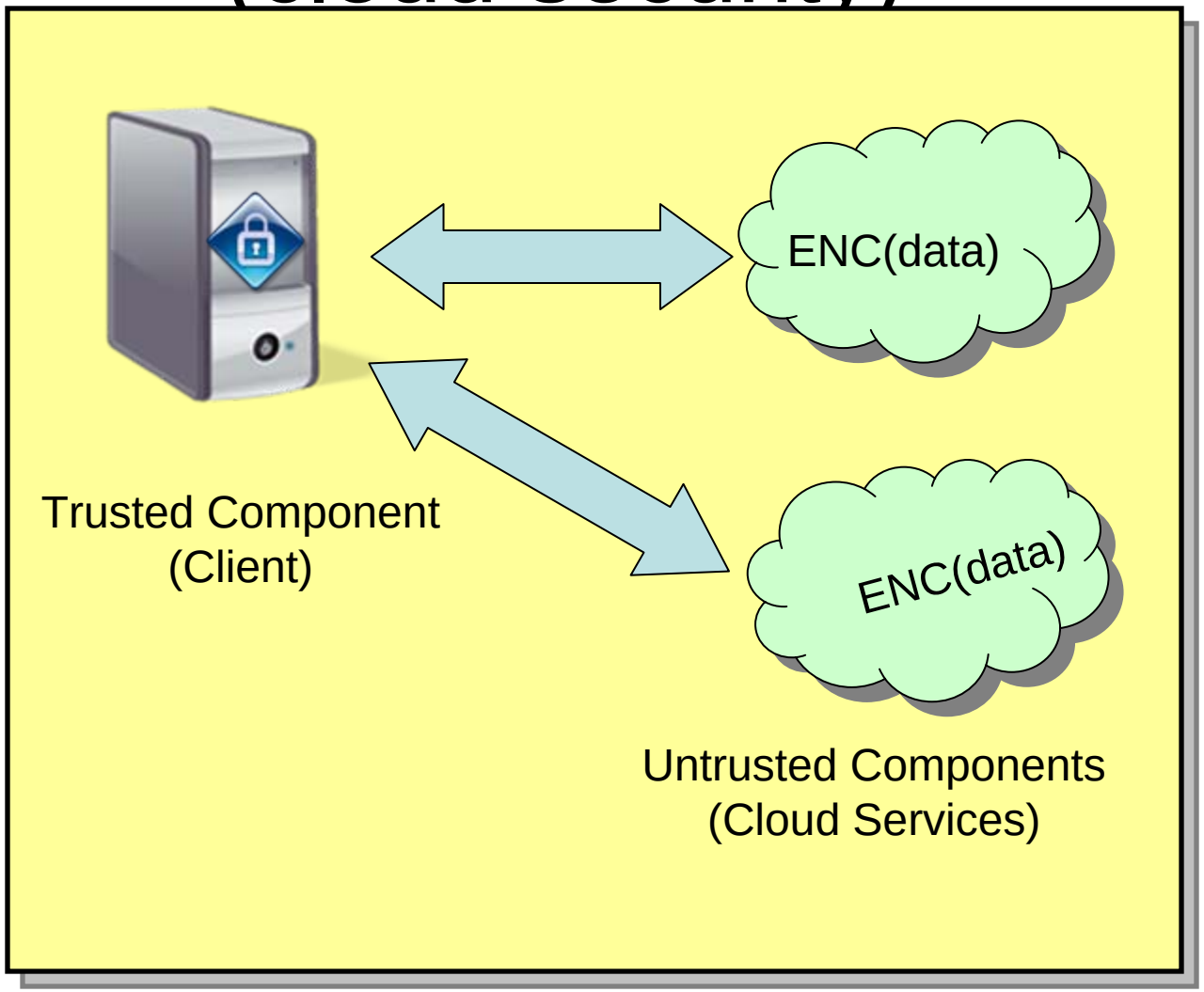
Overview

- Motivation
- Problem Statement
- Review of PIR and ORAM
- New Results
- Conclusion

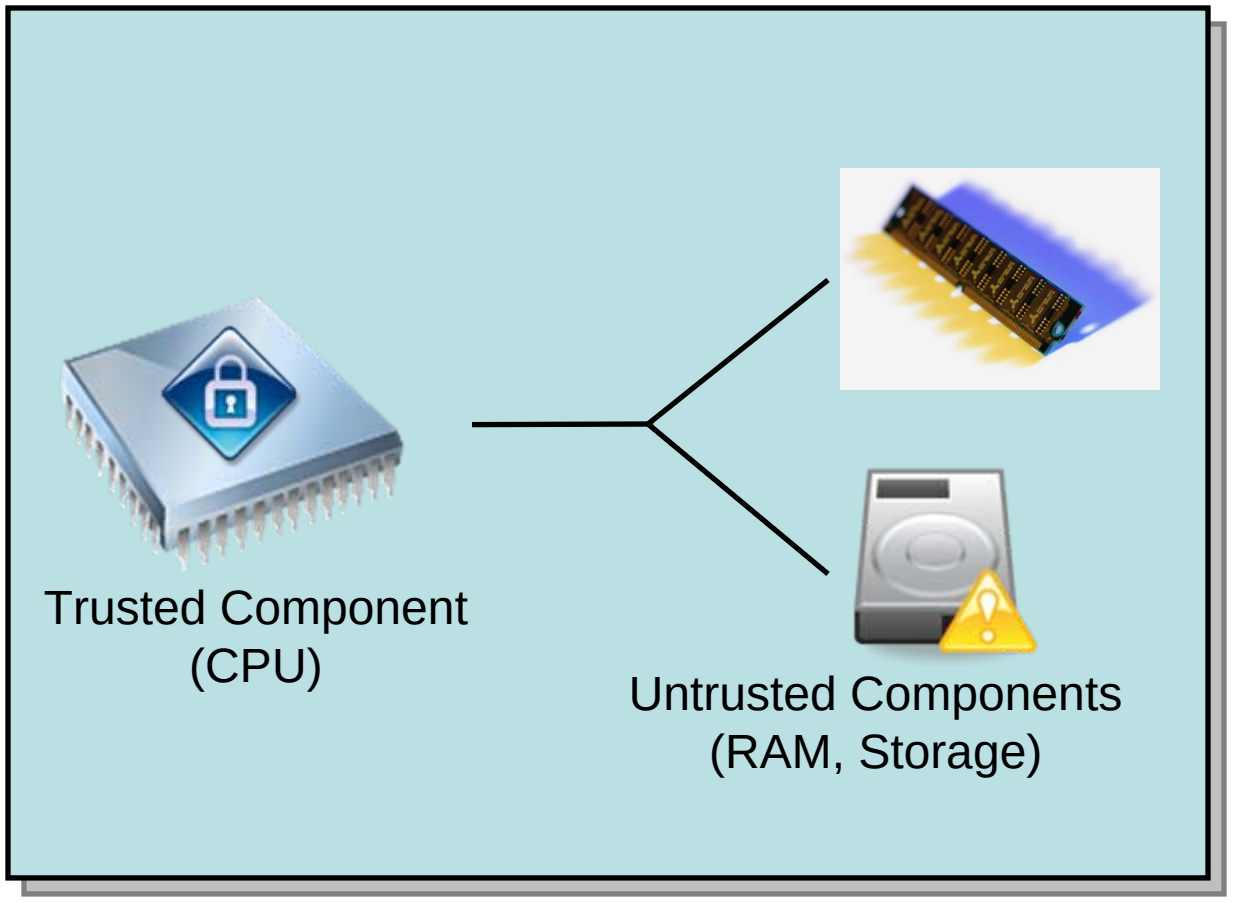
Motivating Example #1 (private monitoring/reading)



Motivating Example #2 (cloud security)



Motivating Example #3 (anti-tamper systems)



Overview

- Motivation
- Problem Statement
- Review of PIR and ORAM
- New Results
- Conclusion

Main Problem

- Encryption/Authentication protects data contents, but does *not* protect which physical locations are accessed.
- Access *remote data* without revealing the so-called *"access pattern"* of both reading and writing:
  - E.g.
    - Public data – stock ticker
    - Private data – cloud storage
- What do we mean by access pattern?
  - Want to hide everything about probed locations. It should look the same no matter:
    - If I ask for the same thing twice
    - If I ask for two adjacent locations
    - If I ask for random locations
  - More formally: given any two sequences of access locations, the server cannot distinguish between them

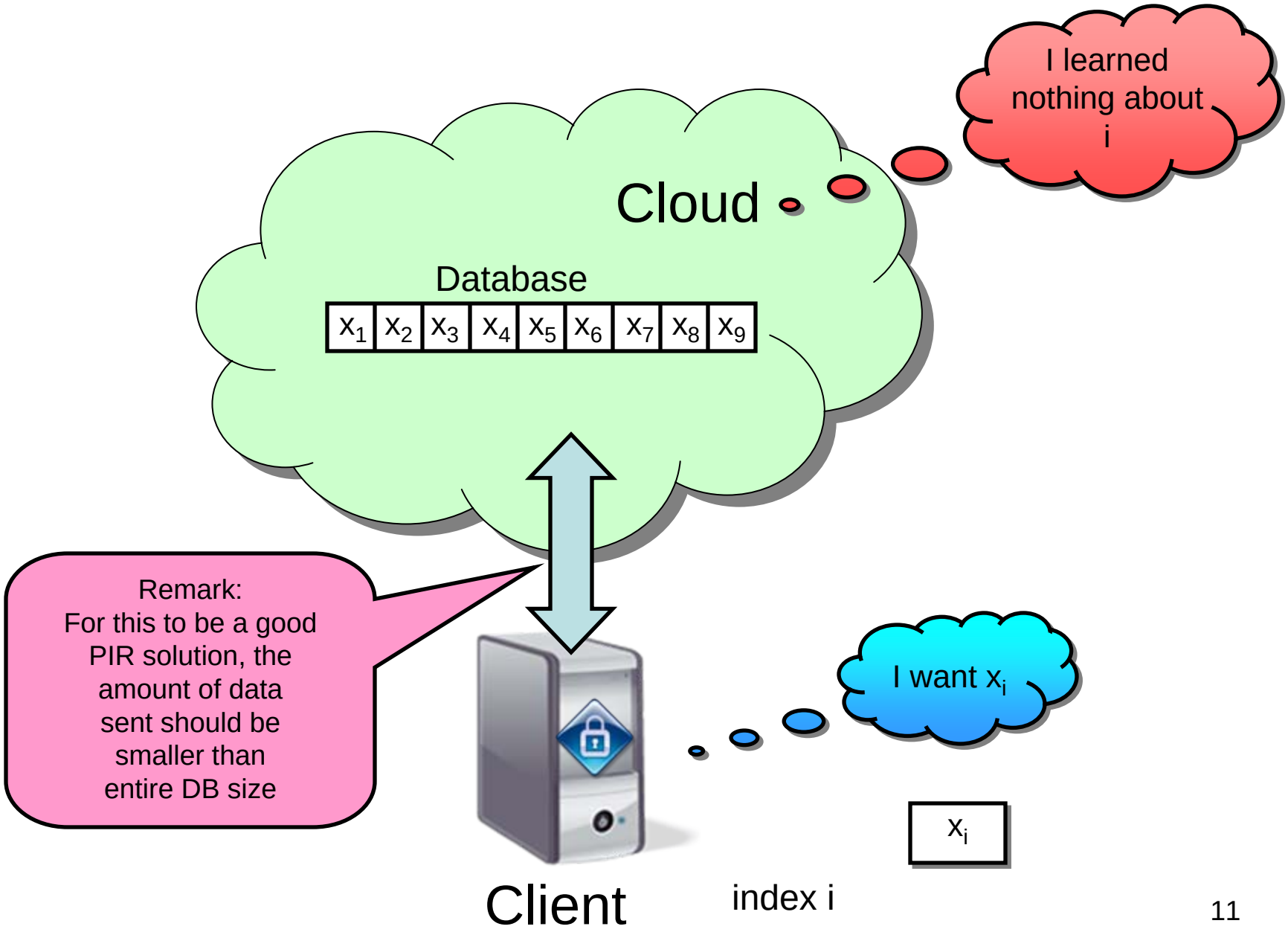
Overview

- Motivation
- Problem Statement
- Review of PIR and ORAM
- New Results
- Conclusion

One Approach: Private Information Retrieval

- Private Information Retrieval (PIR)
  - Allows client to fetch some index *i* without revealing to the server what was retrieved
    - Information Theoretic PIR solution with *replicated* non-communicating servers [CGKS95]
    - [CGKS95] proved single-DB is impossible (in the info-theoretic setting)
    - Computational PIR solution with *single* server [KO97]
  - For now, lets talk about single-DB PIR

What is PIR?


I learned nothing about i
Cloud
Database
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
Remark:
For this to be a good PIR solution, the amount of data sent should be smaller than entire DB size
I want x_i
x_i
Client
index i

Beyond PIR: distributed monitoring

- Instead of reading a single database want to monitor evolving data-sources.

- [OS95]: Searching on Streaming data.
  - (UCLA patent application, licensed to Stealth Software Technologies, Inc.)

Motivating example: “No-fly” list

- Search for classified names and aliases of suspected terrorists
- Knowledge of aliases must be kept secret
  - If not, the advantage derived from this intelligence may again become void.
- Until now, this precludes a distributed search
  - Without our technology, one must rely on an “import, then process” method

Problems with Import, then Process

- Expensive in processing
  - Processing must be done centrally
- Expensive in communication
  - Averse to dynamic data
  - Difficult to manage and synchronize data from vast and disparate sources
- Takes more information into classified setting than needed
  - sometimes can not do this (multi-agency or coalition operations)

Searching on Steaming Data

1. Classified machine: given secret search criteria, create an encrypted search and a decryption key.
2. Migrate encrypted search to multiple machines on any network (or unclassified Server Farm).
3. Every machine runs encrypted search on (local) data, writing output into a small encrypted buffer.
4. Send encrypted buffers back to a classified machine at a regular intervals (minute/hour/day).
5. Classified machine: Decrypt buffers using decryption key from step 1.

Advantages

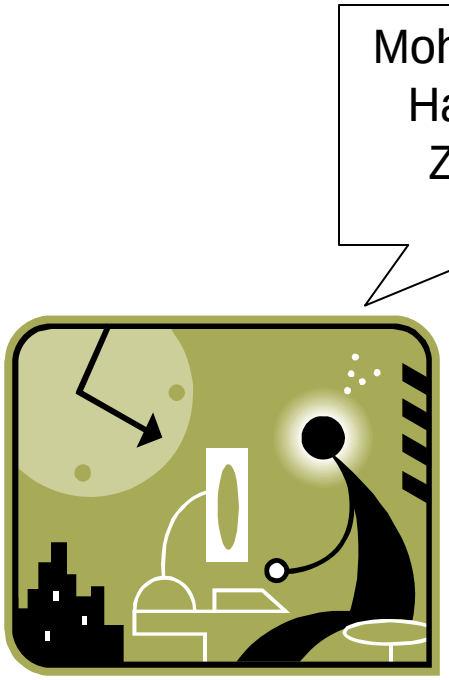
- Attractive alternative to the import, then process paradigm
- Ideal for dynamic, distributed, streaming data
- Creates savings in communication and processing
- Enables low-latency, low-complexity monitoring

Technology at 40,000 feet

- Homomorphic encryption:
  - E(x)*E(y) = E(x+y)
- Stealth discovery (done at UCLA)
  - E(hidden keyword) * DOCUMENT =
    - **=E(0)** **or**
    - **=E(DOCUMENT)** *(only when there is a match!)*
    - Can not tell which outcome happened, just an equation
- Now can use this to “collect” only matching documents into a small encrypted buffer

Our process in detail

Step 1: Create Encrypted Search



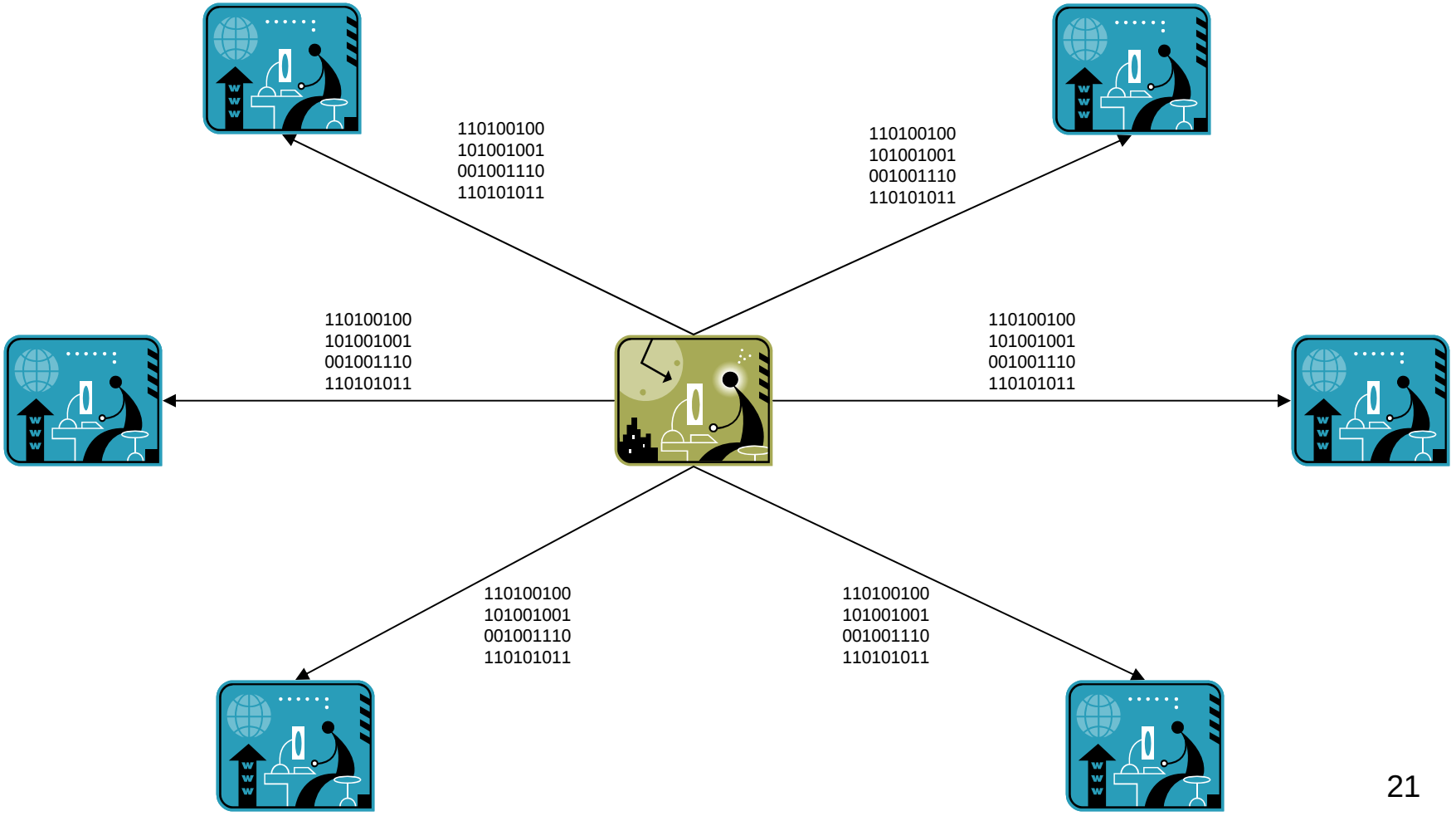
1010101010111000001101010110001001001010101010000101111110100100110100110101011101011001001000011101101011010110101100100100111001001010111010101001010100001011110

Encrypted version of search is indistinguishable from a random distribution

Encrypted Search:

- Provably reveals no information about search terms!
  - Therefore, it can be distributed outside of a classified environment

Step 2: Distribute Search



Step 3: Run Distributed Search

- Any willing and able parties (or a server farm) may now participate.
  - The outside participants know they are helping with a search, but remain oblivious as to what they are searching for and if there are any hits.

- Generic Interface (distributed only once) runs data “through” the encrypted search.
  - Results are collected in small encrypted buffers.

Real-Time Monitoring

- Traditional methods are unpleasant- typically complex and communication-intensive
- Constant downloads / synchronization
  - High complexity, high communication
- Waiting for batches
  - Reduces complexity, but increases latency and still involves un-necessary communication

Real-Time Monitoring

Small 0/1 flag

(Encrypted)

A small encrypted flag can be frequently transmitted indicating the presence or absence of any search results. This provides a simple mechanism for real-time monitoring.

Real-Time Monitoring

The encrypted flags can be aggregated so that one small value can indicate the presence or absence of results for an entire airport, if desired.

Rather than monitoring a constant stream of thousands of names, one small value can be frequently checked on a high side.

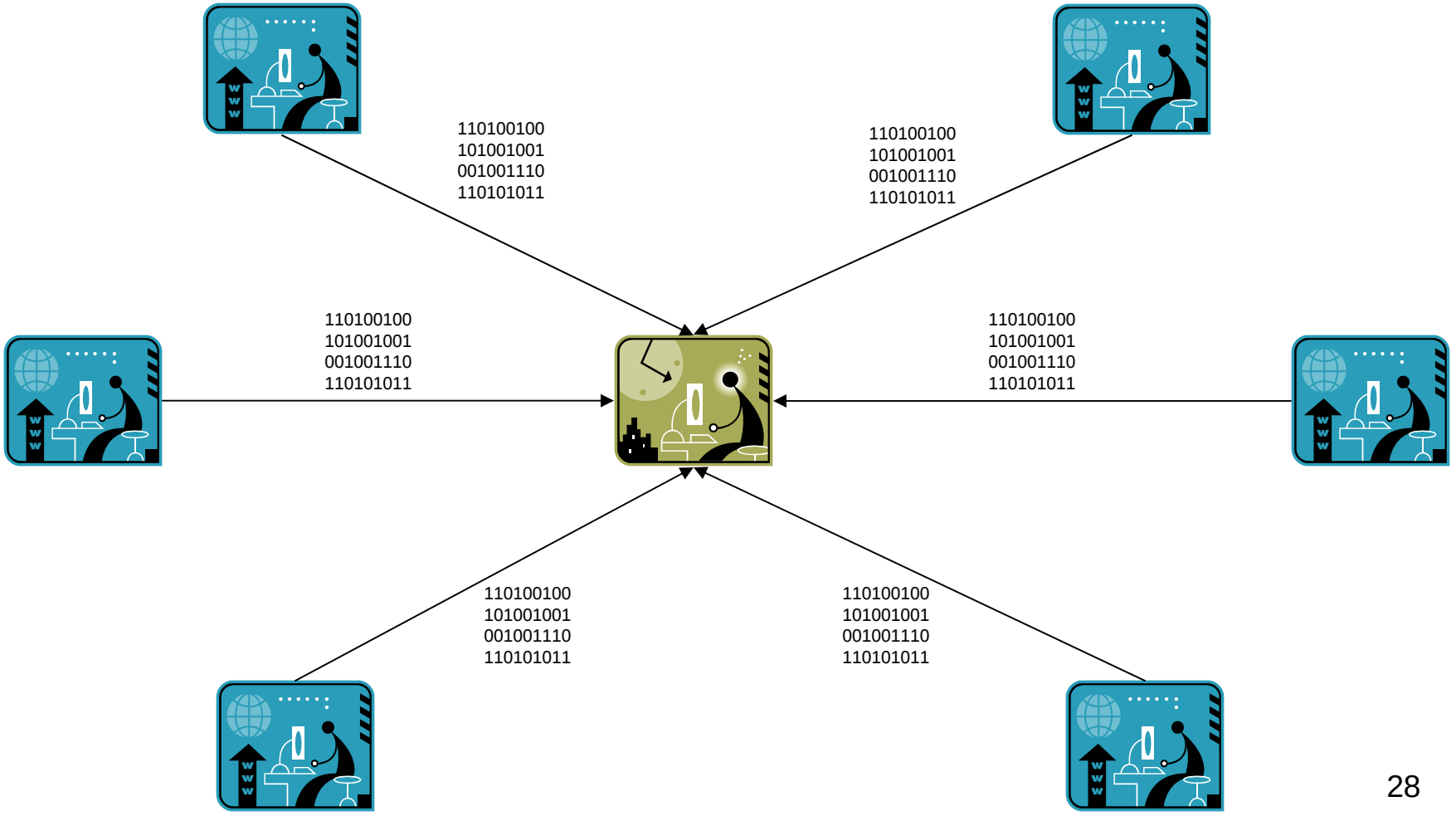
Real-Time Monitoring

- Saves communication- only download critical data,
- Furthermore, you only download *what you were looking for, nothing else*
- Low-overhead, low-complexity method for monitoring vast data sources
- Ideal for highly dynamic data
- Ideal for situations where long knowledge latency is unacceptable

A Note on Encrypted Flags

- Encrypted flags can contain a lot, or only a little information, depending on the application
- They can give additional information, e.g. a more specific location where a hit was found and the number of hits
- If desired, it can be guaranteed to only take values of “yes” or “no”
  - Example: In coalition or multi-agency operations, one can assure that flags reveal found/not found only, and nothing else.

Step 4 and 5: upload & decrypt



Steps 4 & 5: Upload & Decrypt

- Collection of “interesting data”:
  - Transfer small buffers to a classified environment
  - Then, decrypt buffers to obtain results

- Decryption key is NEVER given to the low (i.e., unclassified) side, everything on the unclassified side is **encrypted**.

Design and Performance

- Designed for parallel architectures.
- Based on independently developed high-performance library for long integers and number theory.
  - In single processor, 32-bit mode, already outperforms well-established and respected libraries (e.g. NTL) on an Intel Core 2 by more than a factor of 2.
  - 64 bit mode outperforms 64 bit optimized NTL by a factor of 7 for multiplication of 1024 bit integers.
  - On a 2GHz core 2 duo in 64-bit mode can process data at 100KB/sec (small files) and 120KB/sec for large files.
  - This is about 100x faster than where we started.
- Makes use of special purpose arithmetic algorithms, ideal for the task

Another Approach: Oblivious RAM

- Oblivious RAM (ORAM)
  - Introduced by Goldreich and Ostrovsky
  - Allows client to write and read to untrusted storage *encrypted data* without revealing what or where it is being accessed
- We focus on this solution for the remainder of the talk

Overview

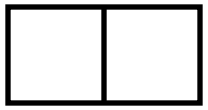
- Motivation
- Problem Statement
- Review of PIR and ORAM
- New Results
- Conclusion

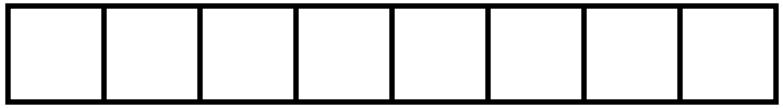
Model of Oblivious RAM

- Small, trusted component
  - CPU
  - User
- Large, untrusted component
  - RAM
  - Server Farm
- Goal: Protect the *contents* and the *access pattern* of the small CPU from the large RAM/Cloud storage

- PIR & ORAM Models are *different*: ORAM hides *encrypted* data of CPU/User instead of reading *public data* (as in PIR).

Review: Hierarchical Solution [O90]

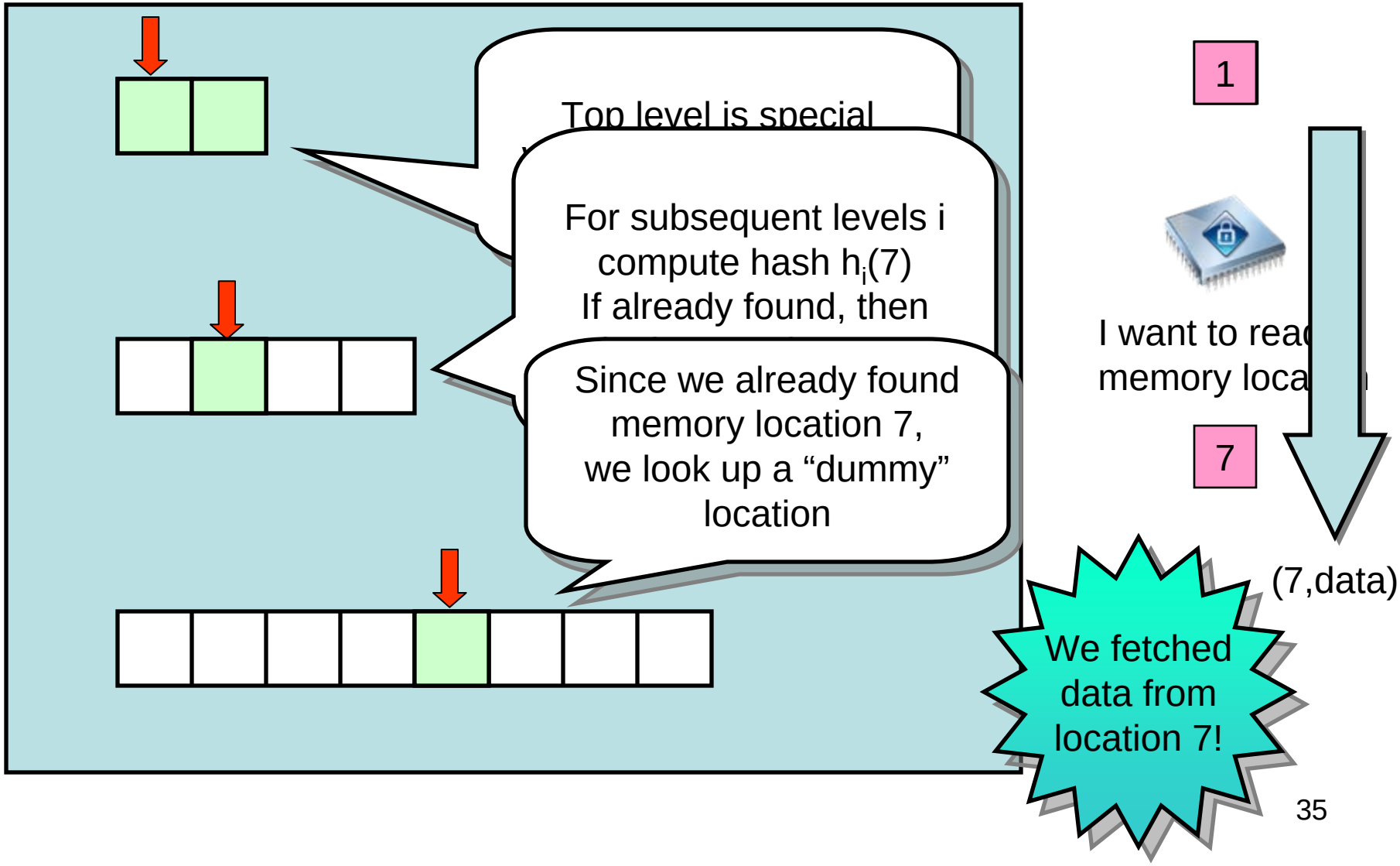




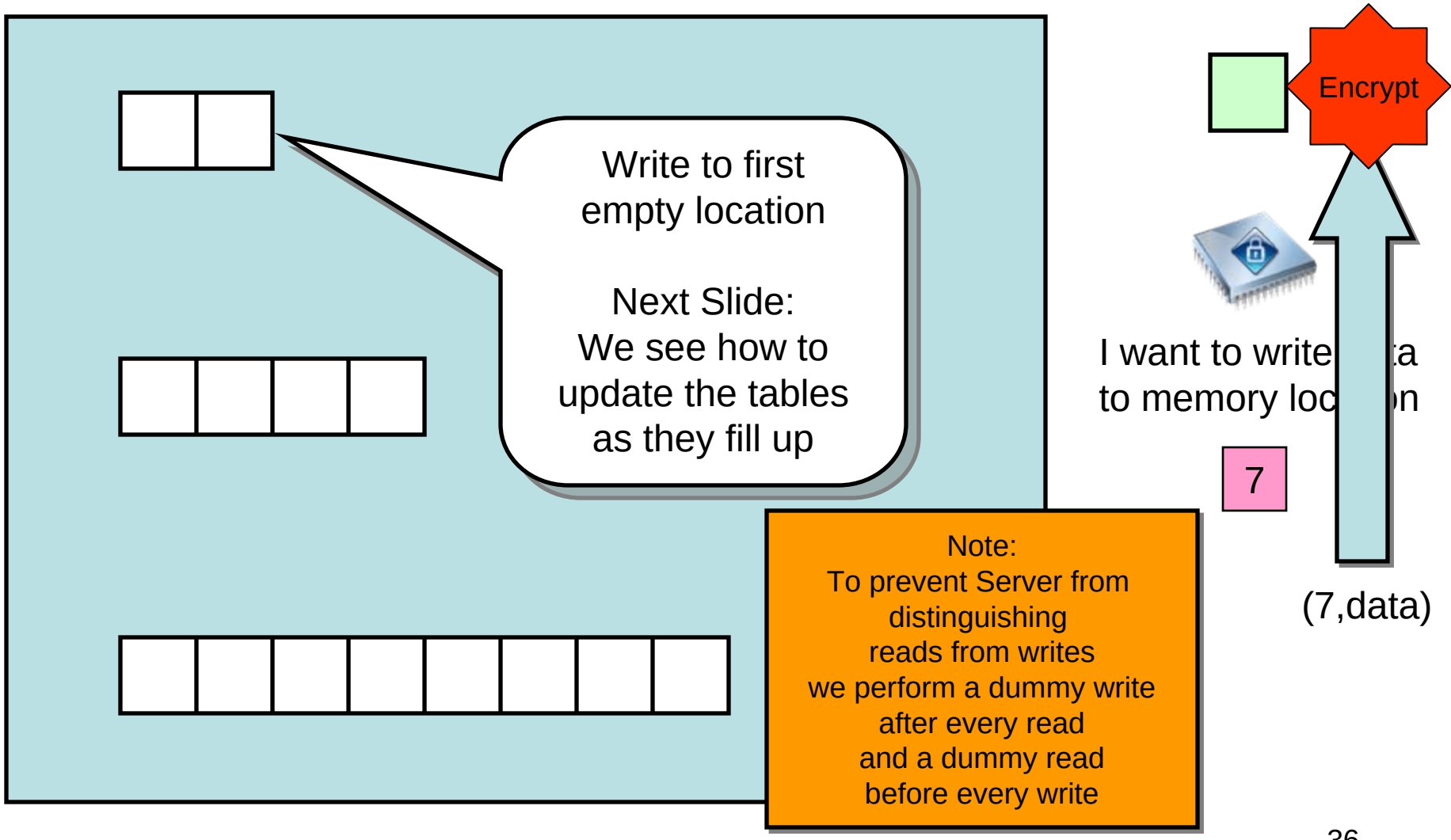
- Set up the Server/RAM in a hierarchy of tables
- Tables with sizes in geometric progression
  - E.g. each table is twice the size of the previous one
- Hash tables
  - Bucketed hash tables with log sized buckets
  - Cuckoo hash (need to be careful with these)
- Main property: a pair (x,v) where x is a memory location and v is the contents will reside *encrypted* on the server and shall appear in a level i in table position $h_i(x)$

We drill down to the details to see how this happens

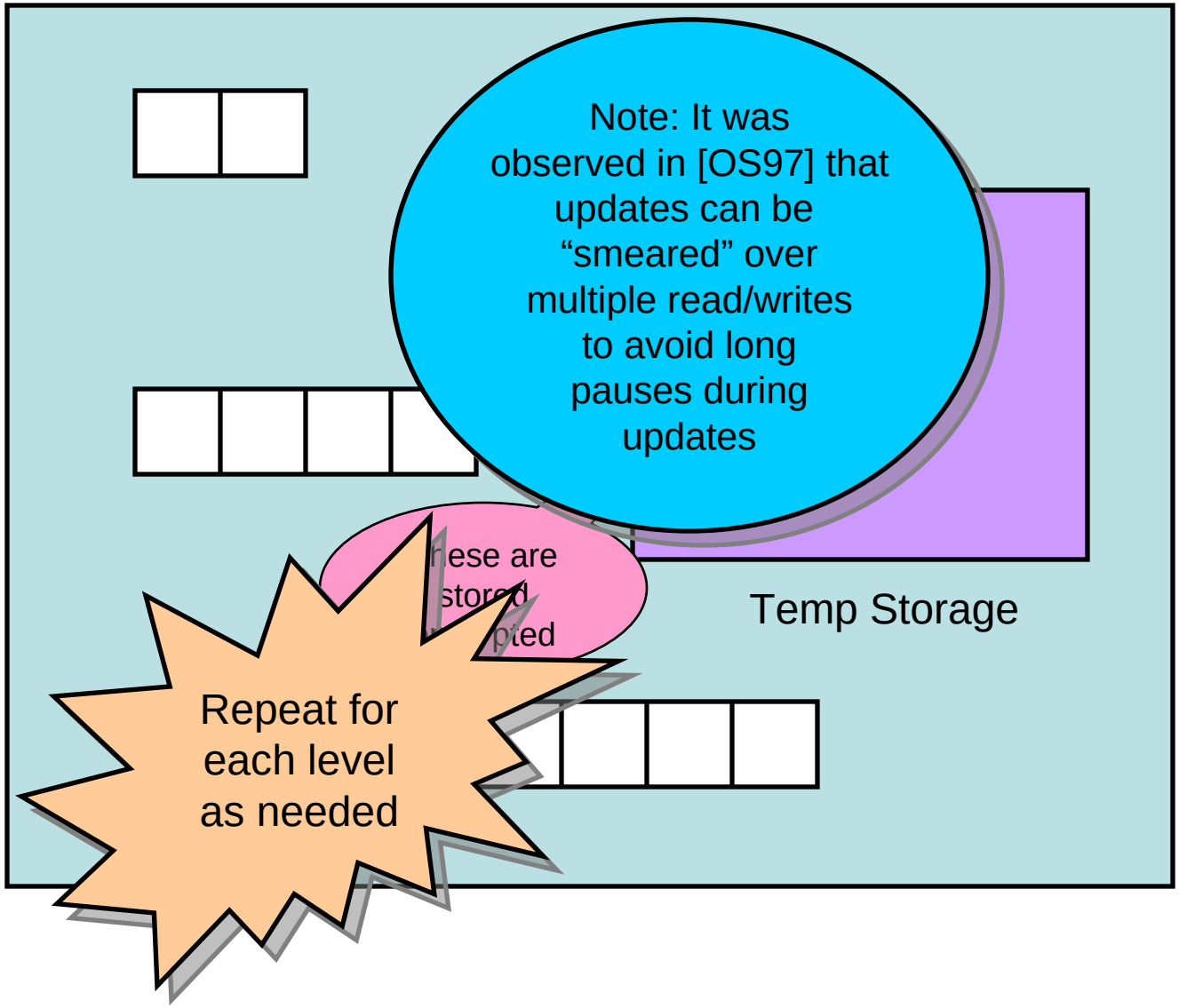
Review: Hierarchical Solution [O90] Reading an element

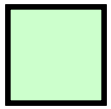


Review: Hierarchical Solution [O90]
Writing an element



Review: Hierarchical Solution [O90] Updating the Hierarchy





2 1

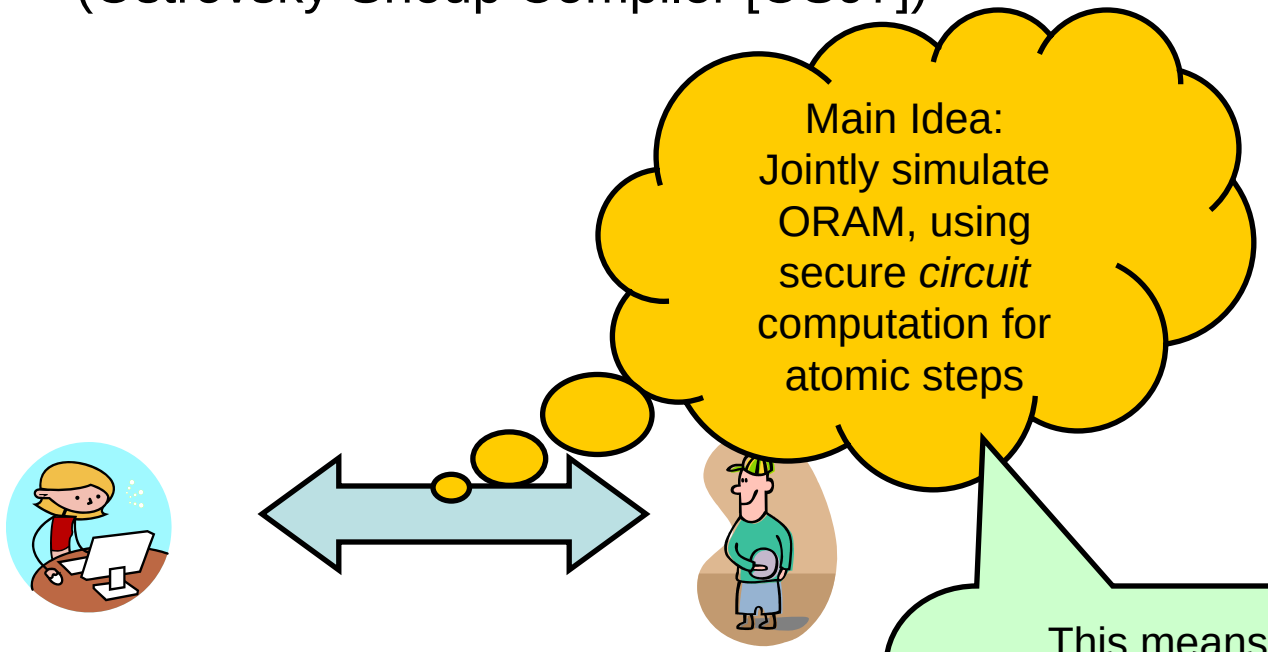
Add Dummy Elements as needed

Compute Hash Locations

Oblivious Sort Store in level

Application to Secure Computation

(Ostrovsky-Shoup Compiler [OS97])



Wish to compute some program P(A,B) without revealing inputs

Note that many existing secure computation solutions work on circuits rather than programs. We are able to “bootstrap” this!

Overview

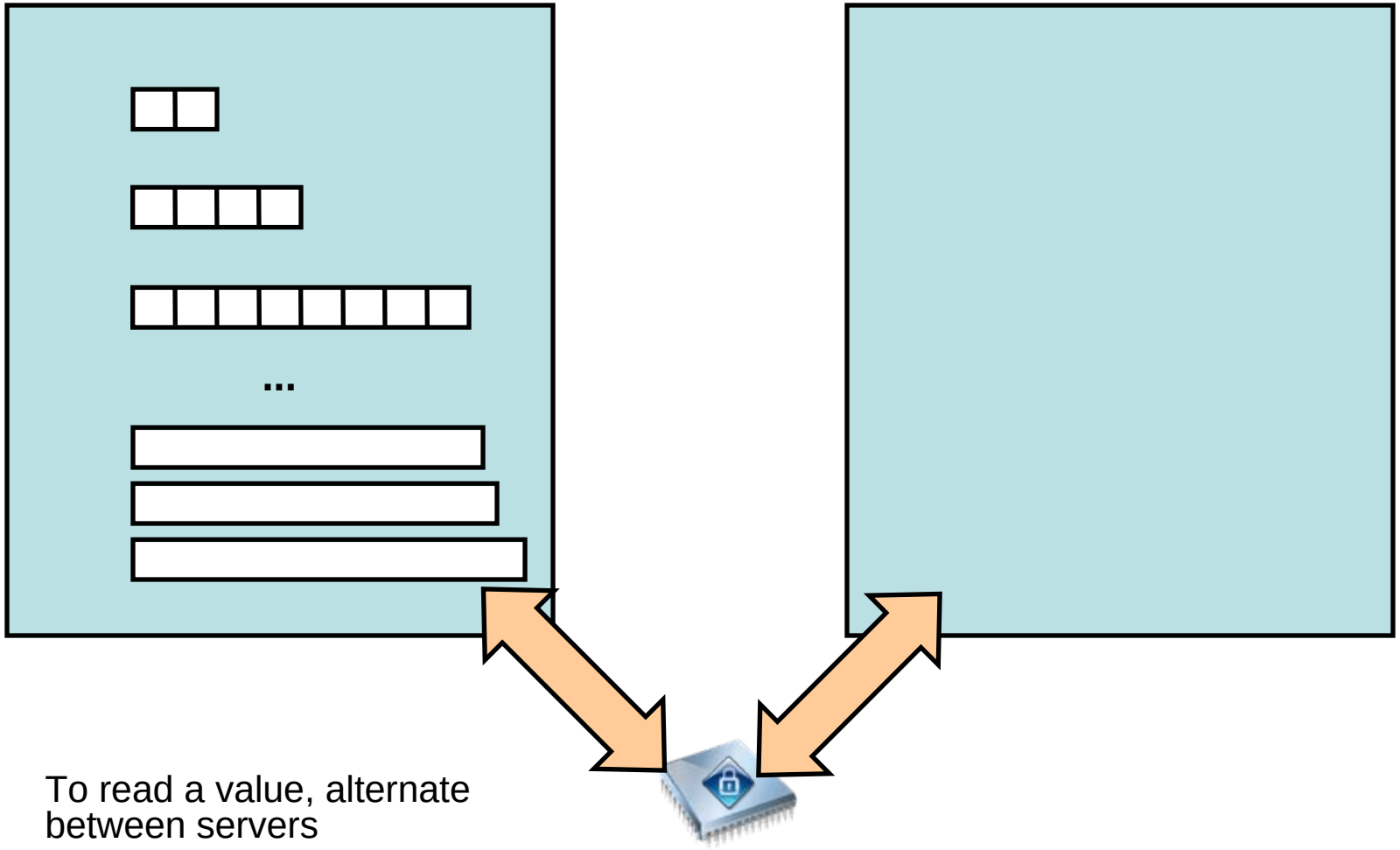
- Motivation
- Problem Statement
- Review
- New Results
- Conclusion

New Results

- New Insight: in Ostrovsky-Shoup complier, Alice-Bob can afford to have *two* non-communicating servers.
- Multi-Server Oblivious RAM [LO11]
  - Joint work with Steve Lu
  - Two (or more) non-communicating servers
    - E.g. multiple cloud services
  - O(log n) access overhead with constant client memory
    - Matches lower bound in the *single-server* case
  - Bypasses the expensive “oblivious sort” during updates
- Balancing Oblivious RAM [KLO11] (to appear in SODA-12)
  - Joint work with Eyal Kushilevitz and Steve Lu
  - Reduces the total overhead by balancing accesses with updates.

Multi-Server Oblivious RAM [LO11]

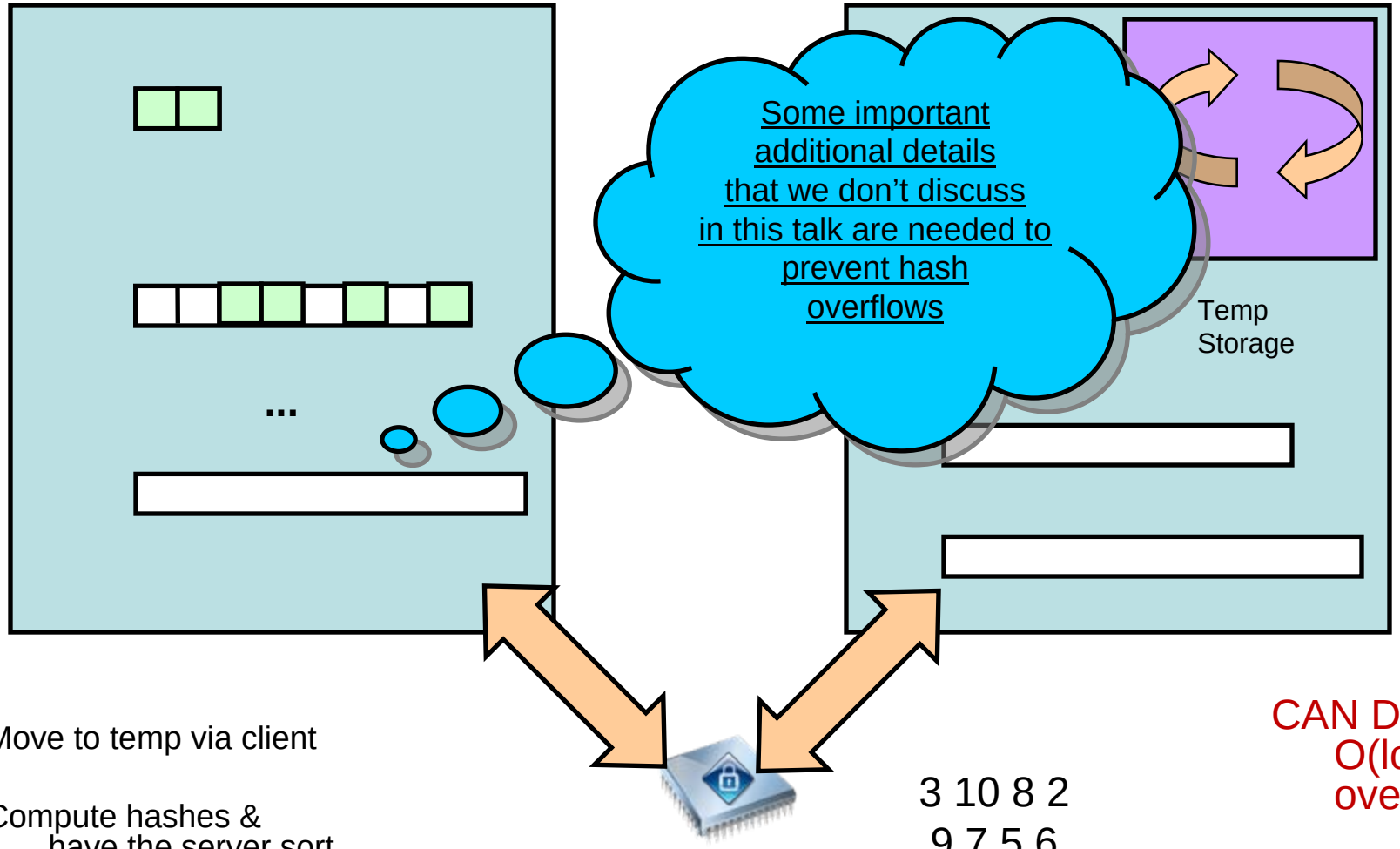
Main Idea



- To read a value, alternate between servers
- Let's see how update works

Multi-Server Oblivious RAM [LO11]

Updating the levels



Move to temp via client

Compute hashes &
have the server sort

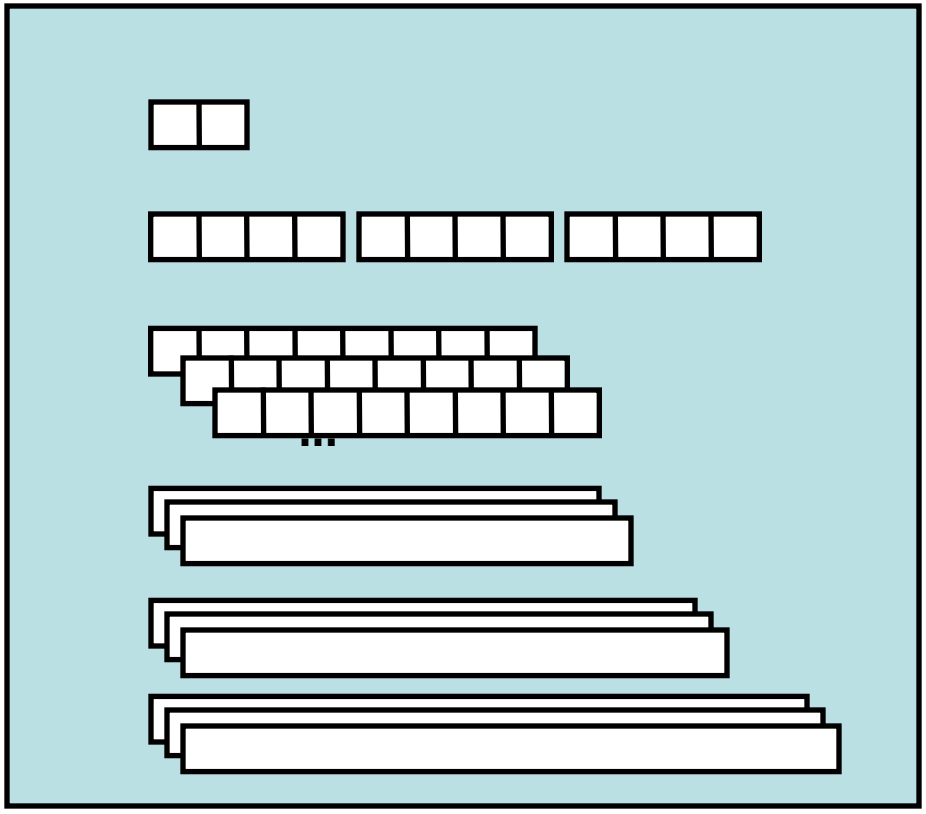
Move back to other server via client

3 10 8 2
9 7 5 6
4 1

CAN DO
O(log N)
overhead

Balancing Oblivious RAM [KLO11]

Main Idea



- Another idea to reduce overhead
- Single server model
- Increase the size of each level to reduce the frequency of updates
  - Simply increasing the growth rate does not give us enough savings!
- Main idea: each level stores multiple hash tables
  - This reduces the frequency of updates
  - But increases the cost of reading
- How many?
  - Optimization Problem
  - For our construction turns out to be log(n) tables per level
  - Reduces total overhead down to $O(\log^2 n/\log\log n)$

Overview

- Motivation
- Problem Statement
- Review
- New Results
- Conclusion

Conclusion

- In this talk, we defined the problem of hiding the access pattern from the server
- We discussed two approaches
  - Single-DB PIR [KO97]
    - and searching in streaming model [OS05]
  - Hierarchical ORAM [O90,GO96]
- We gave additional details for the ORAM approach
  - Hierarchical Solution [O90,GO96]
  - Avoiding long pauses with “worst-case” overhead [OS97]
  - Application to secure computation [OS97]
- Described new results for ORAM [KLO12,LO11]

THANK YOU!

BACKUP: References

- [CGKS95] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private Information Retrieval. In *FOCS 1995*, 41-50.
- [KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is Not Needed: Single Database, Computationally-Private Information Retrieval. In *FOCS 1997*, 364-373.
- [O90] Rafail Ostrovsky: Efficient Computation on Oblivious RAMs In STOC 1990: 514-523
- [GO96] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. In *J.ACM 43(3) 1996*, 431-473.
- [KLO] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. Cryptology ePrint Archive, Report 2011/327, 2011. To appear in *SODA 2012*.
- [LO] Steve Lu and Rafail Ostrovsky. Distributed Oblivious RAM for Secure Two-Party Computation. Cryptology ePrint Archive, Report 2011/384, 2011.
- [OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *STOC 1997*, 294-303.