Practicing Oblivious Access on Cloud Storage: the Gap, Fallacy, and the New Way Forward

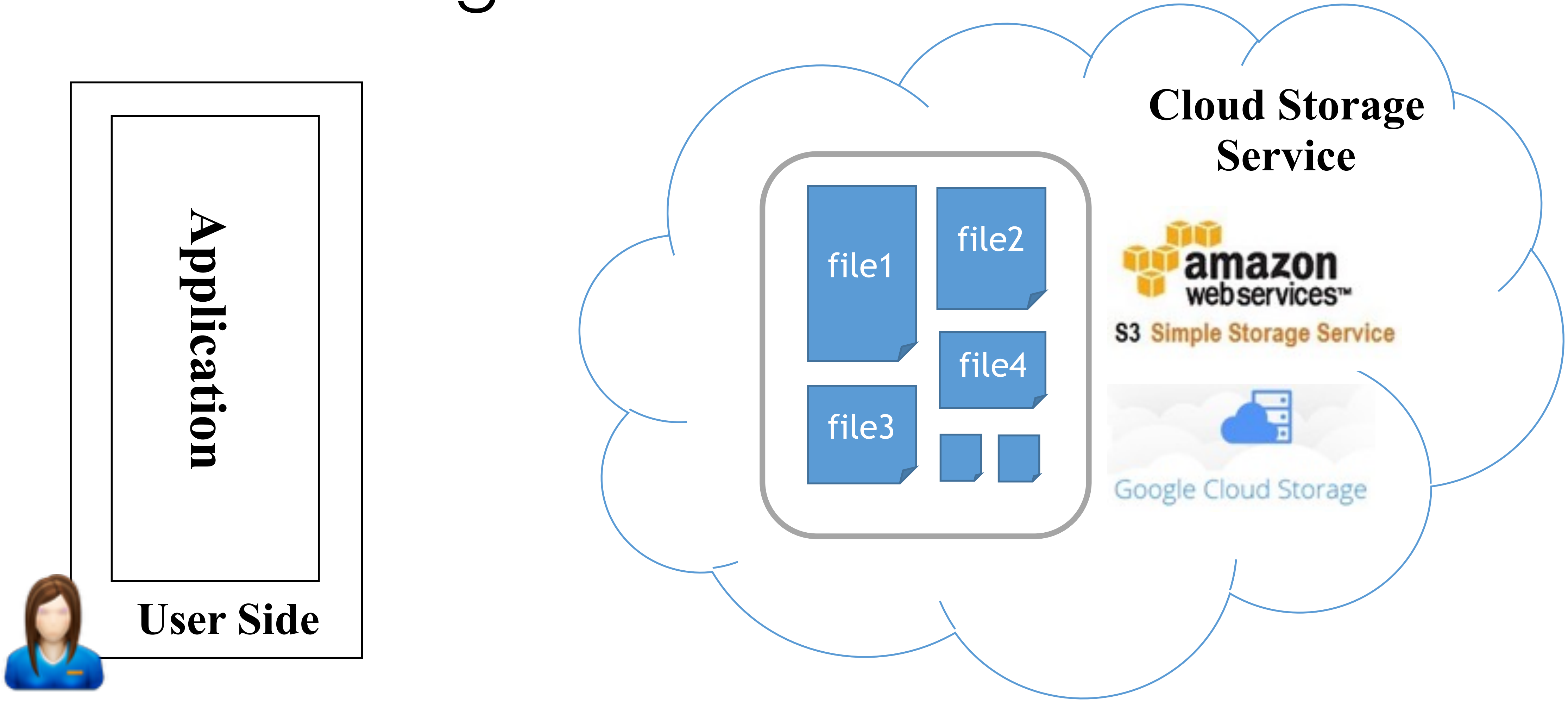
Vincent Bindschaedler[1], Muhammad Naveed[1,3], Xiaorui Pan[2], XiaoFeng Wang[2], and Yan Huang[2]

[1]University of Illinois at Urbana-Champaign

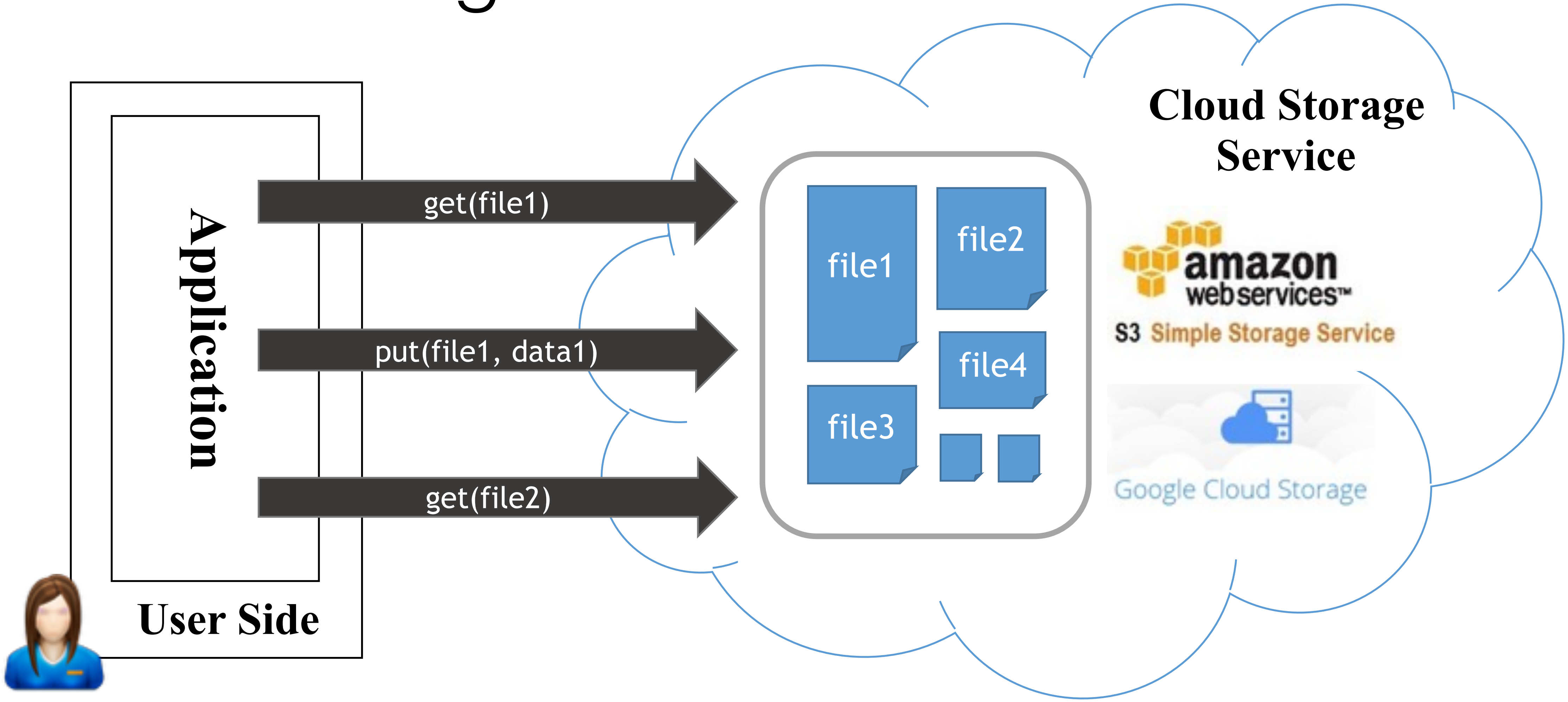
[2]Indiana University Bloomington

[3]Cornell University

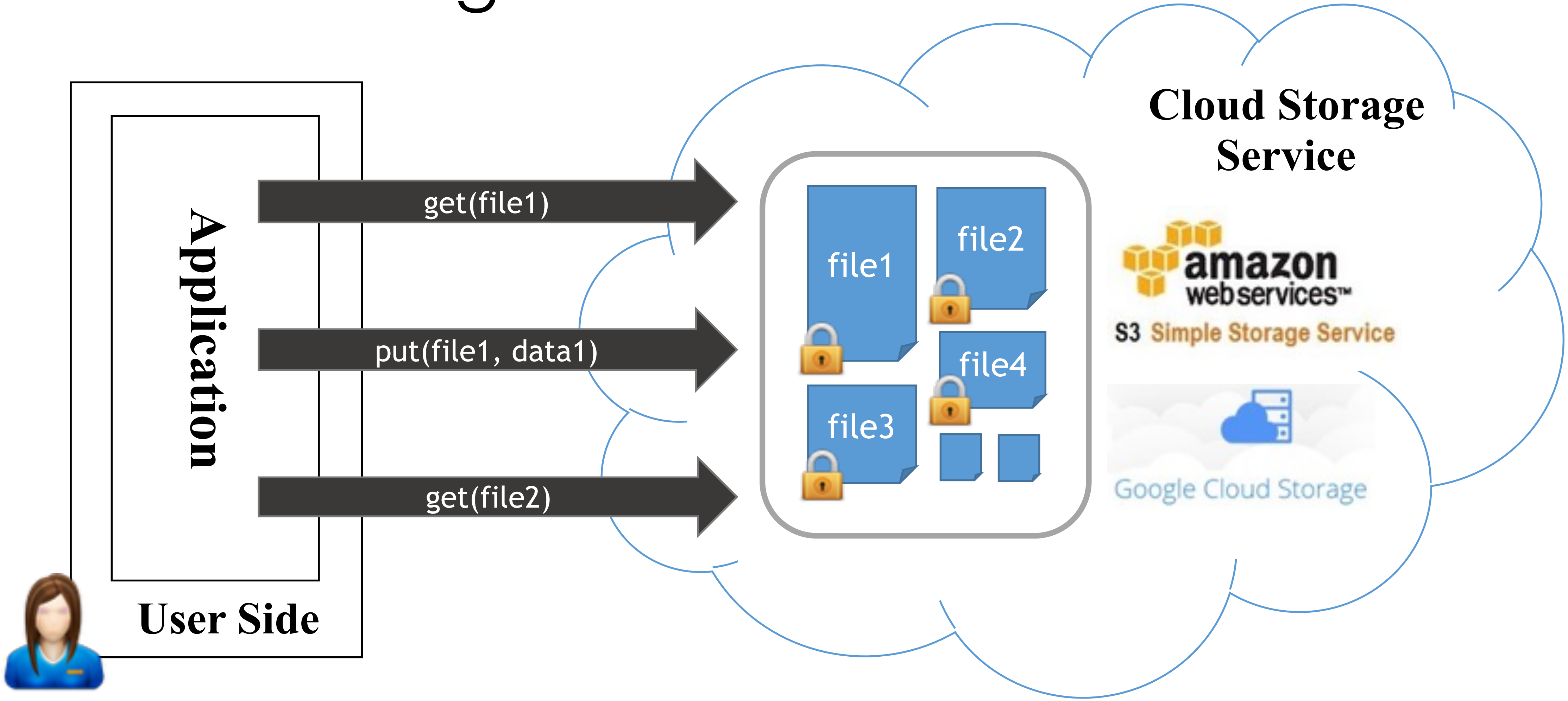
Cloud Storage



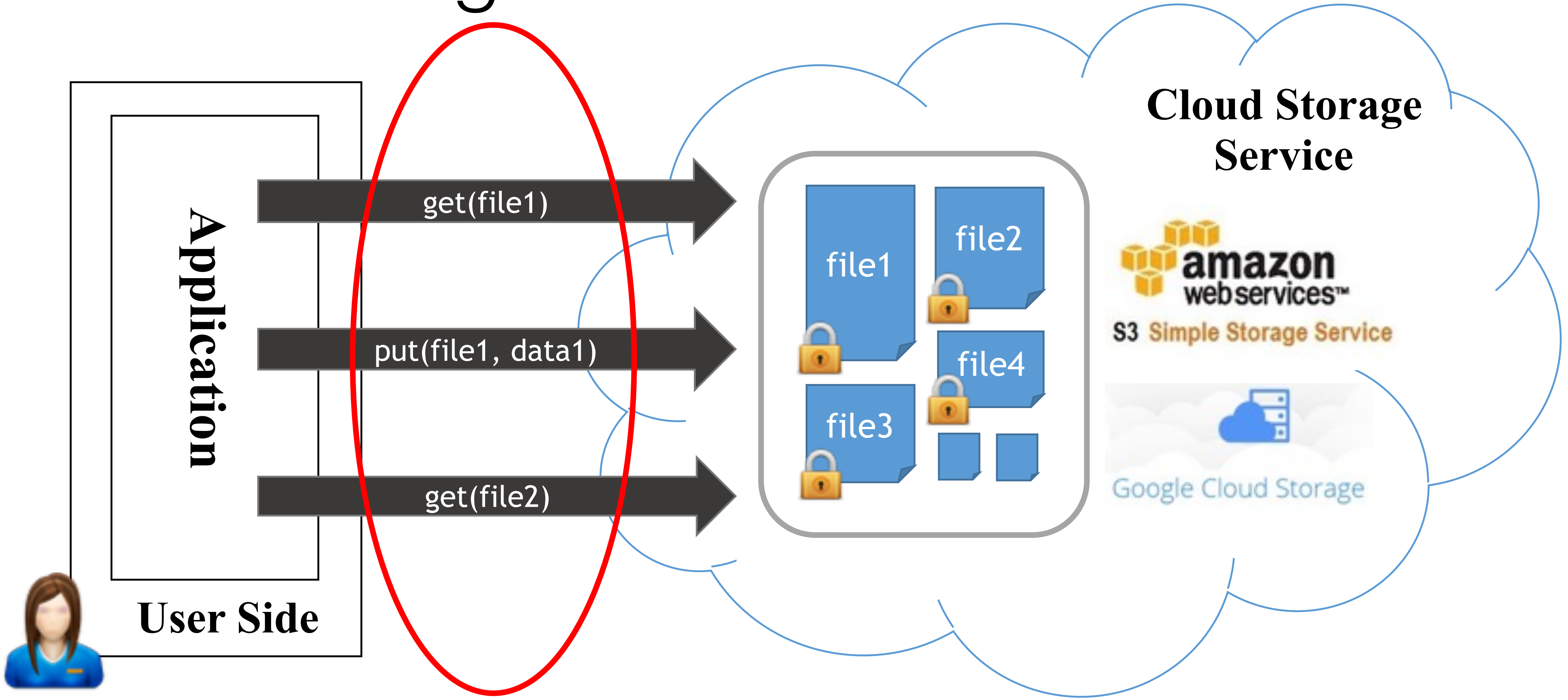
Cloud Storage



Cloud Storage



Cloud Storage



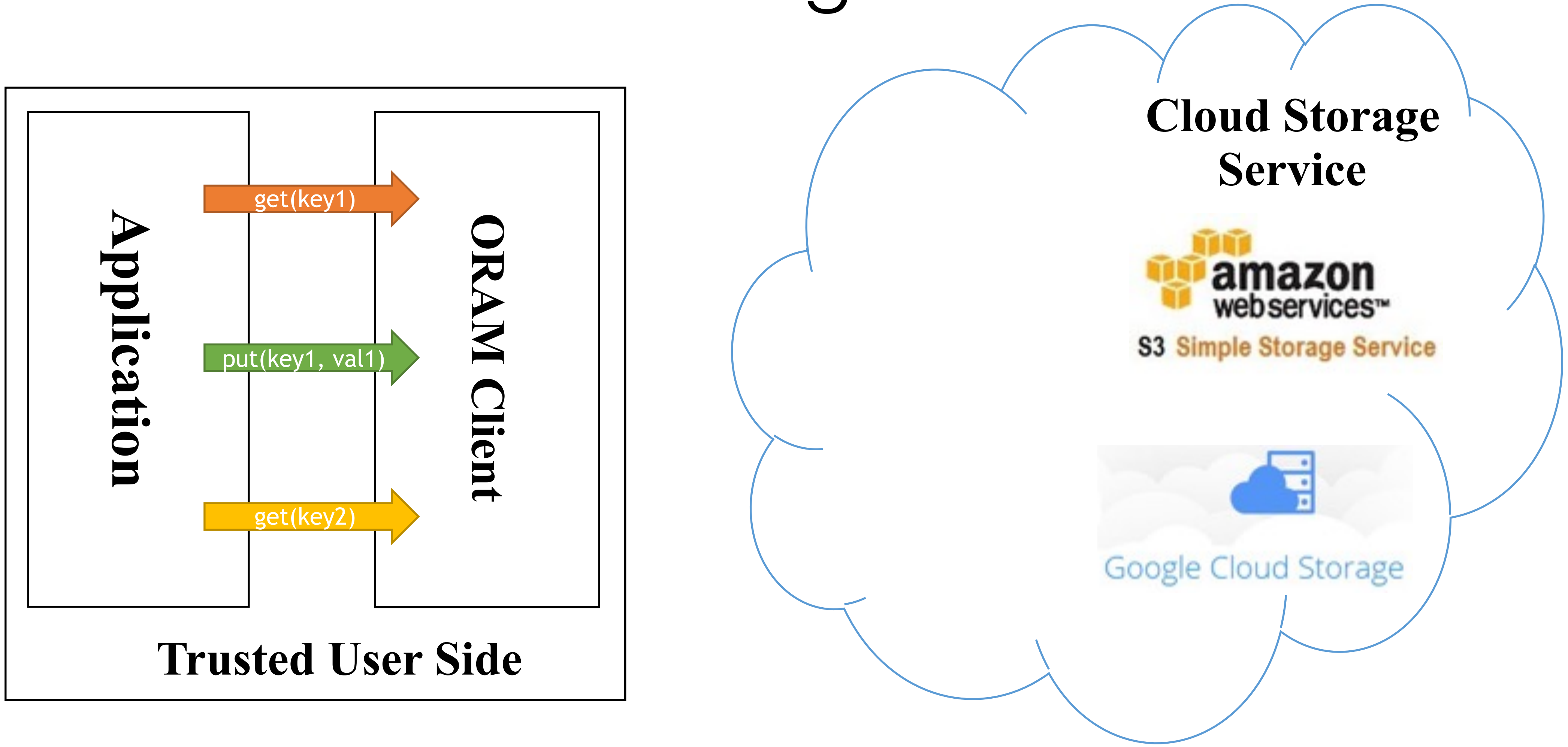
Leaks access pattern

Background: Oblivious RAM

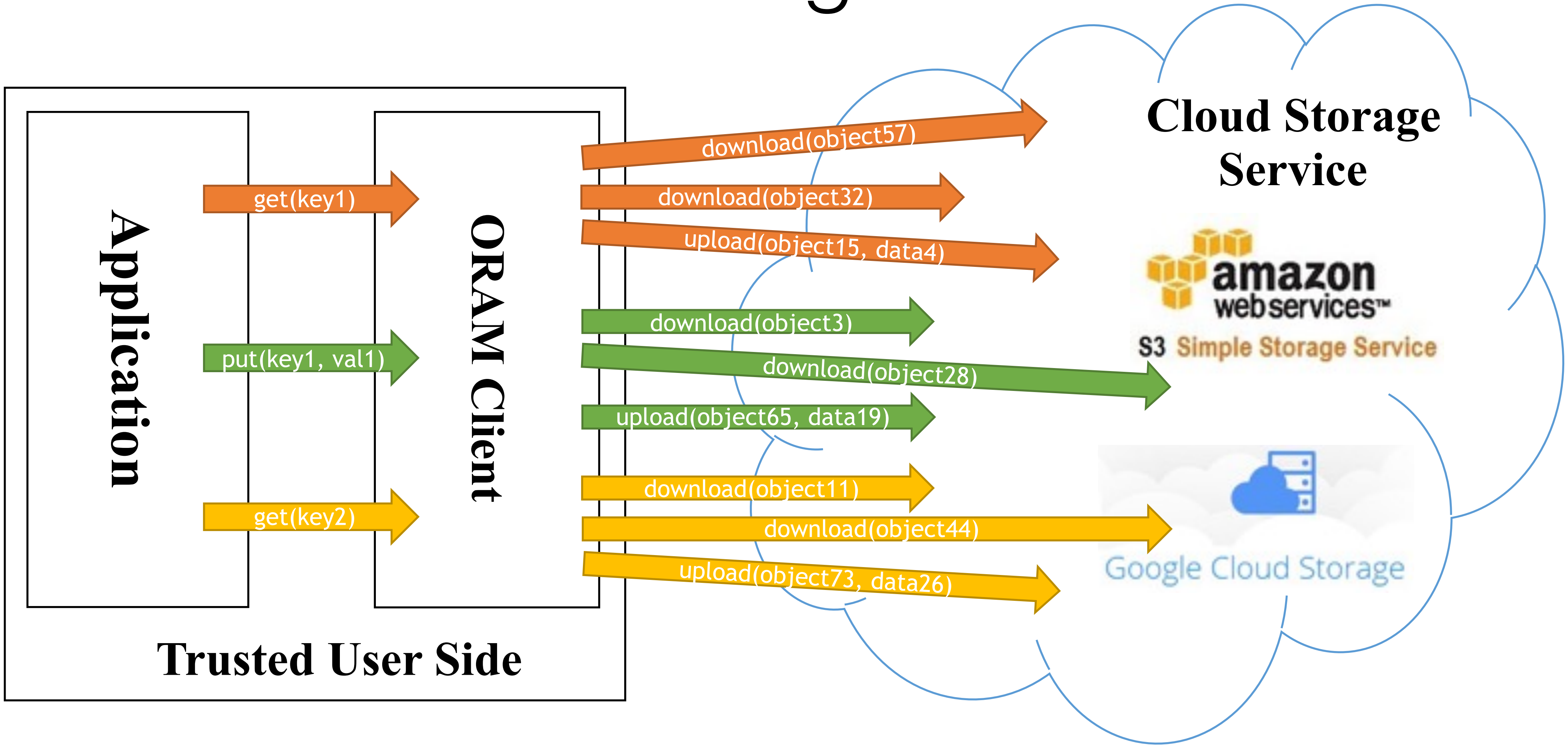
- Obliviousness:
  - For any fixed size request sequence, the associated storages accesses observed (by the cloud) are statistically independent of the requests
- Techniques
  - Operates on fixed size data blocks
  - Encrypt blocks with ciphertext indistinguishability
  - Dummy accesses, re-encryption, shuffling, etc.

Oblivious Cloud Storage

Oblivious Cloud Storage



Oblivious Cloud Storage



How close is ORAM to practice?

- Are ORAM designs in line with the constraints of real-world cloud services?
- How close are ORAM techniques to offering practical support to cloud applications?
- Are we on the right track to narrow the gap?

Assumptions in ORAM literature

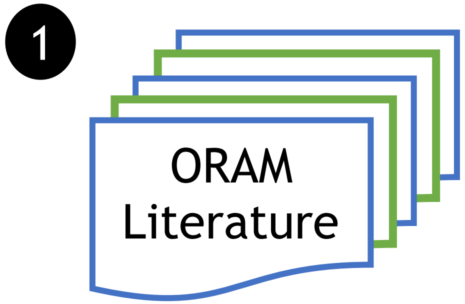
1. Bandwidth overhead is a good proxy metric
    - So, minimizing it optimizes application performance

2. Application is **not** taken into account
    - Implicit assumption that application has no impact on performance

Assumptions influence the way the problem is thought about and guide the research agenda.

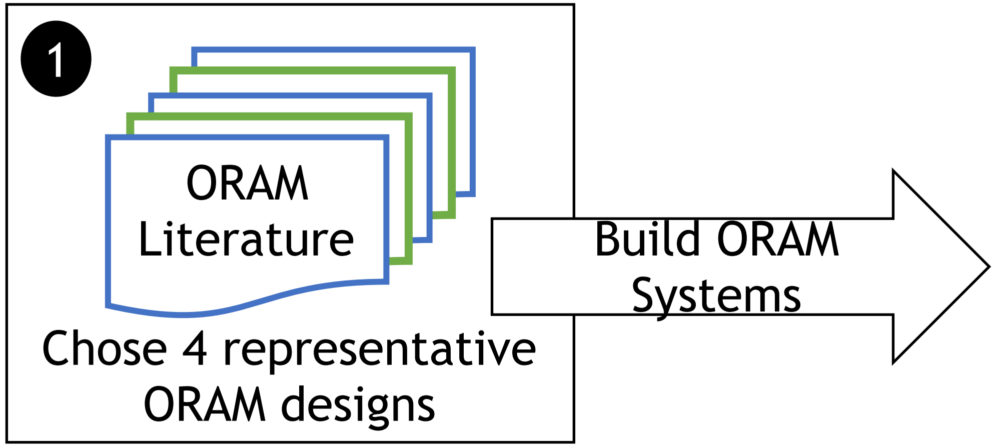
Contribution

Contribution

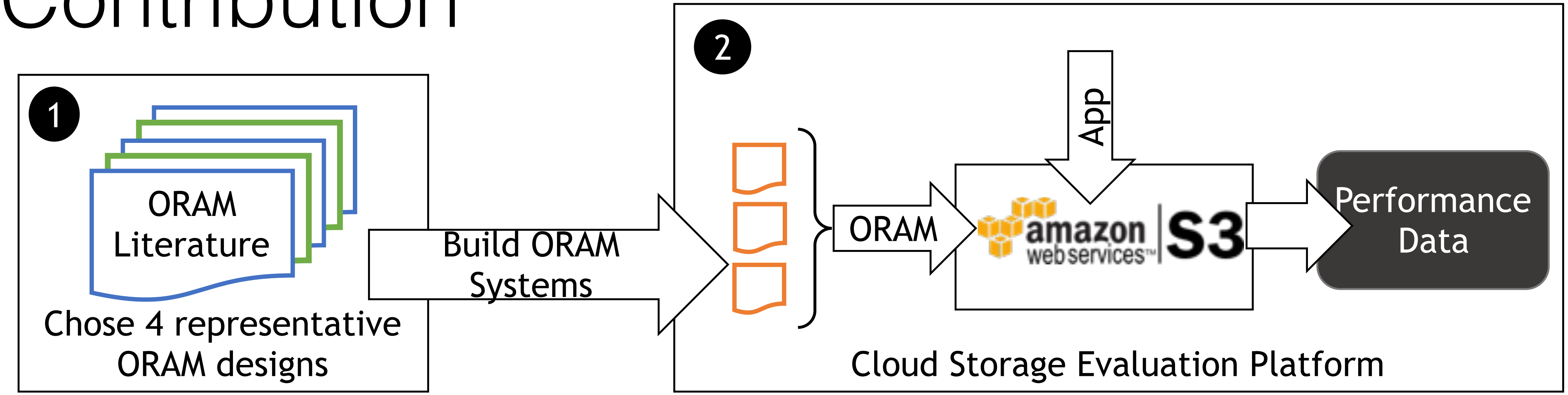


Chose 4 representative ORAM designs

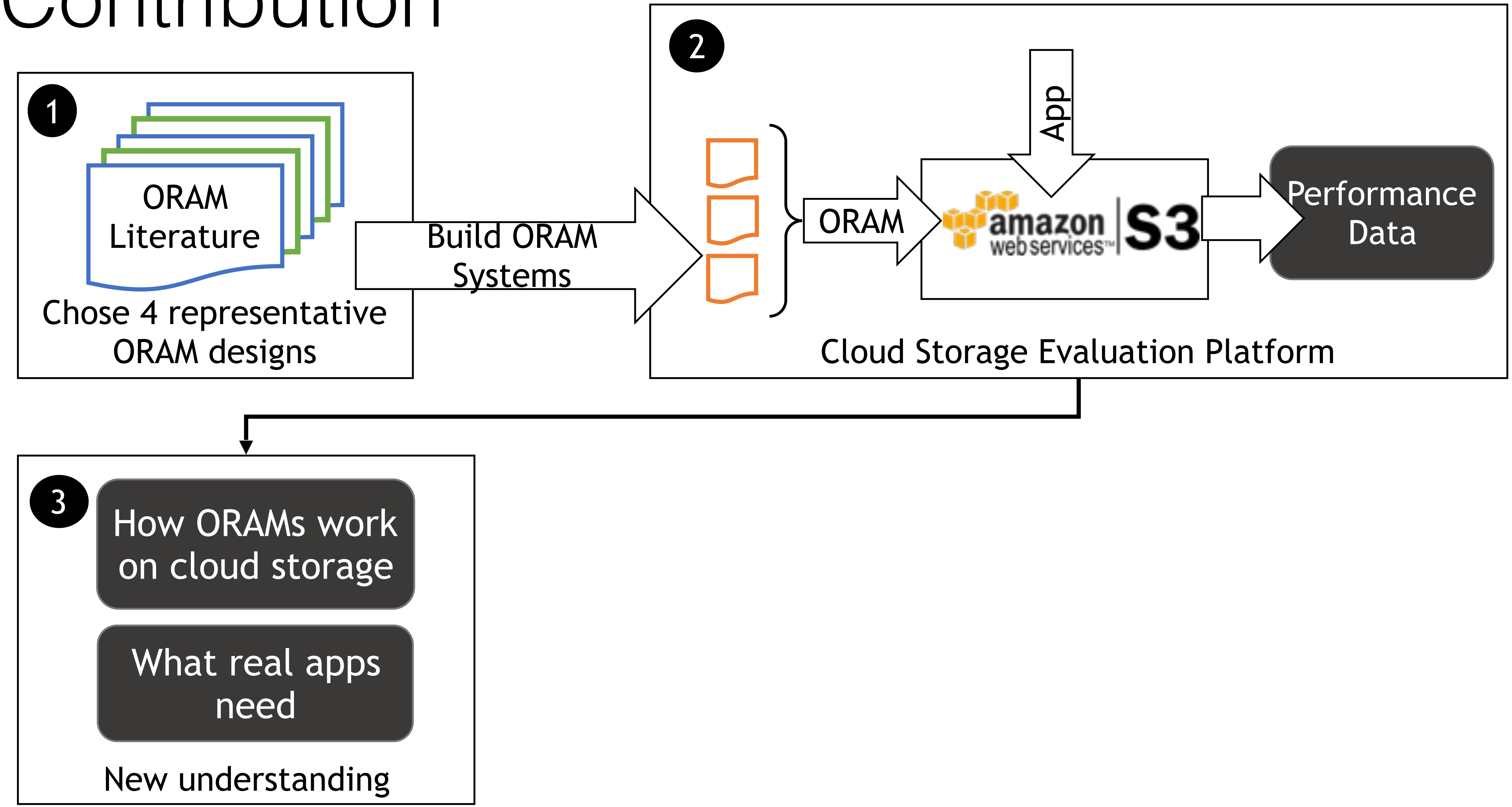
Contribution



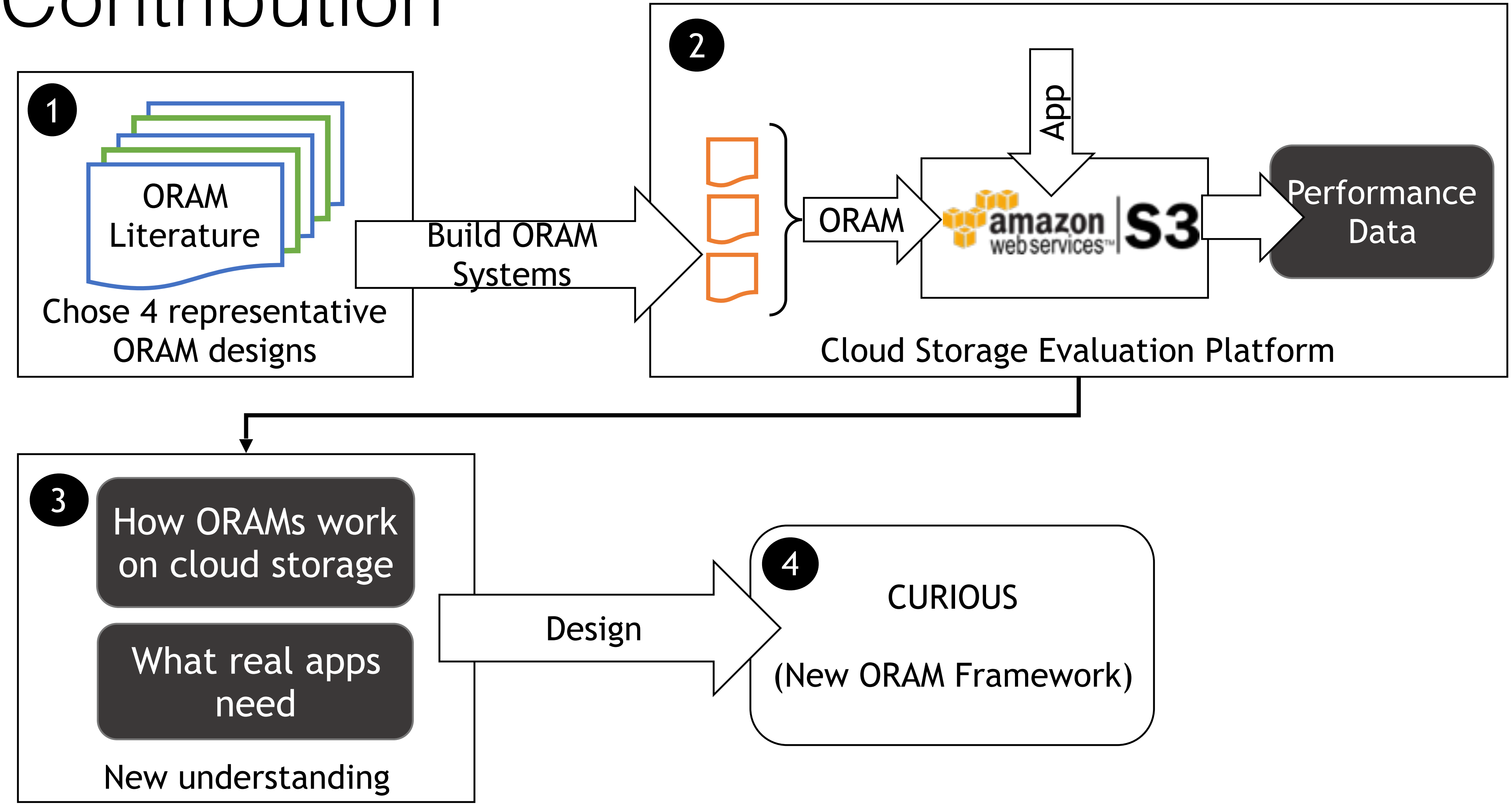
Contribution



Contribution



Contribution



ORAM Systems We Built

1. Tree-based: PathORAM
2. Layered-based: LayeredORAM
3. Large messages-based: PracticalOS
4. Partition-based: ObliviStore

1. [PathORAM] Stefanov, Emil, et al. "Path ORAM: An Extremely Simple Oblivious RAM Protocol." CCS 2013.
2. [LayeredORAM] Goodrich, Michael, et al. "Oblivious RAM simulation with efficient worst-case access overhead." CCSW 2011.
3. [PracticalOS] Goodrich, Michael, et al. "Practical oblivious storage." CODASPY 2012.
4. [ObliviStore] Stefanov, Emil, and Elaine Shi. "Oblivistore: High performance oblivious cloud storage." S&P 2013.

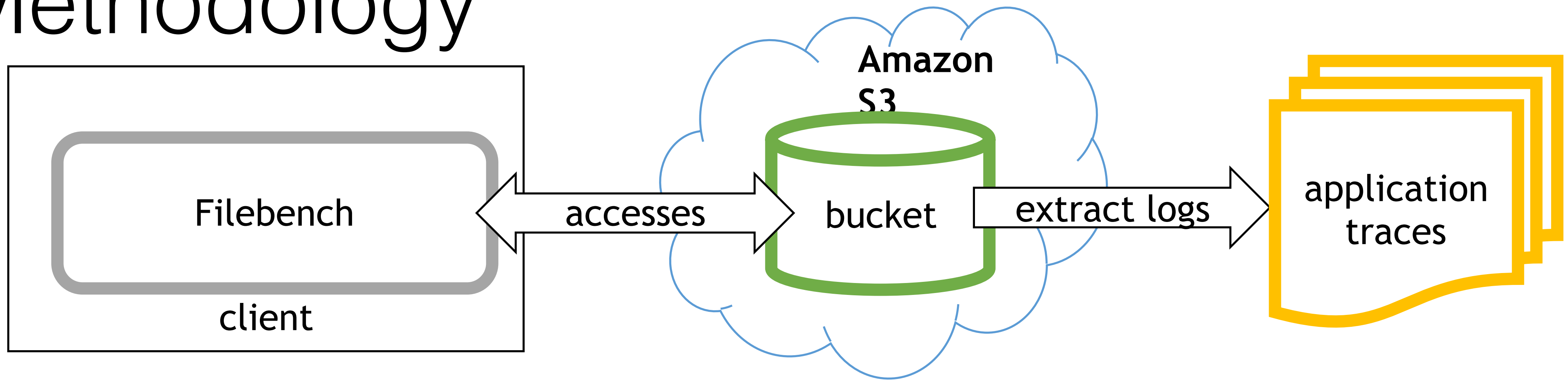
Application Selection

- We use Filebench: filesystem benchmarking too

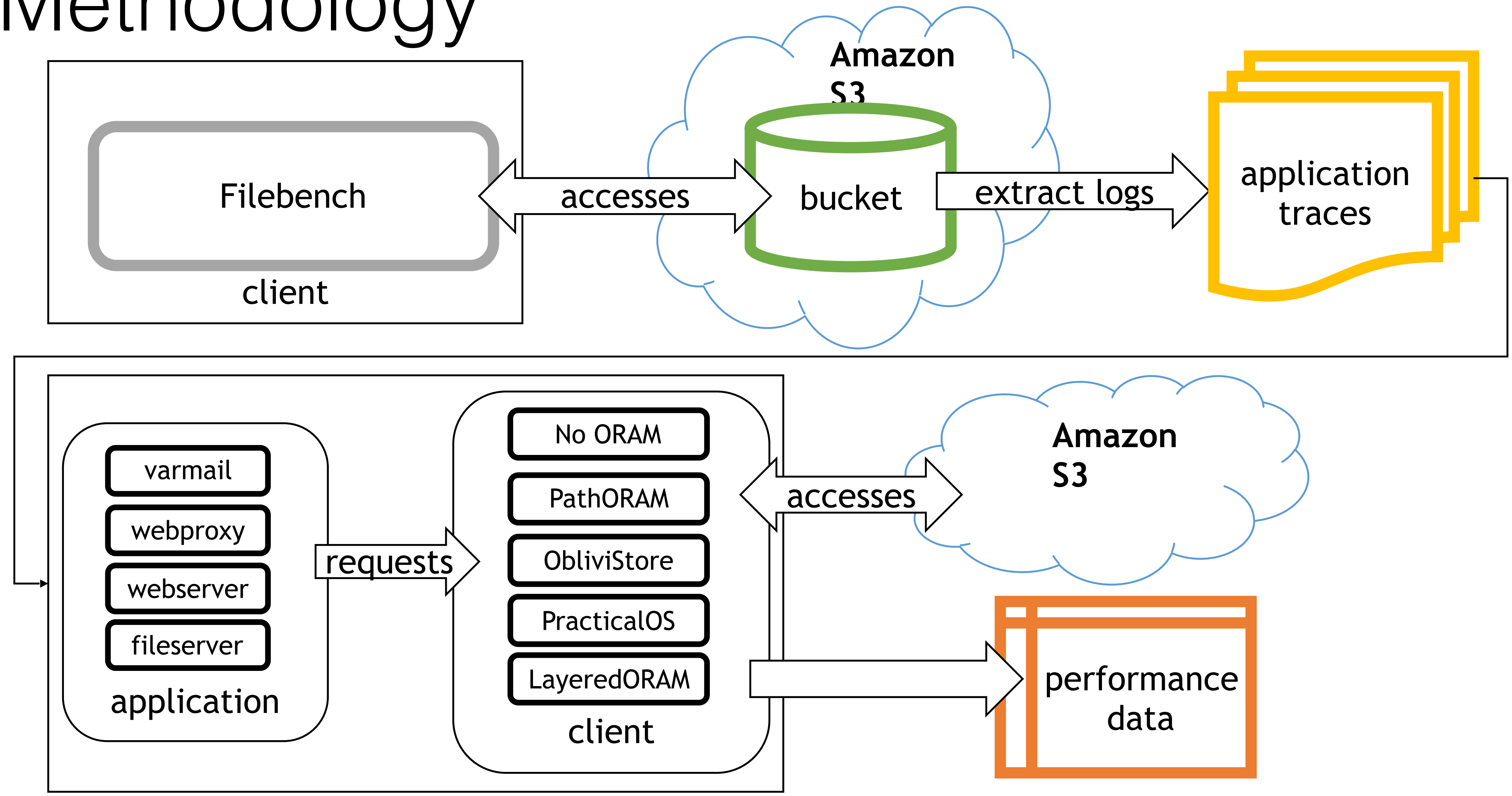
- Able to emulate several applications, e.g.:
  - Mail server
  - File server
  - Web proxy
  - Web server

Methodology

Methodology

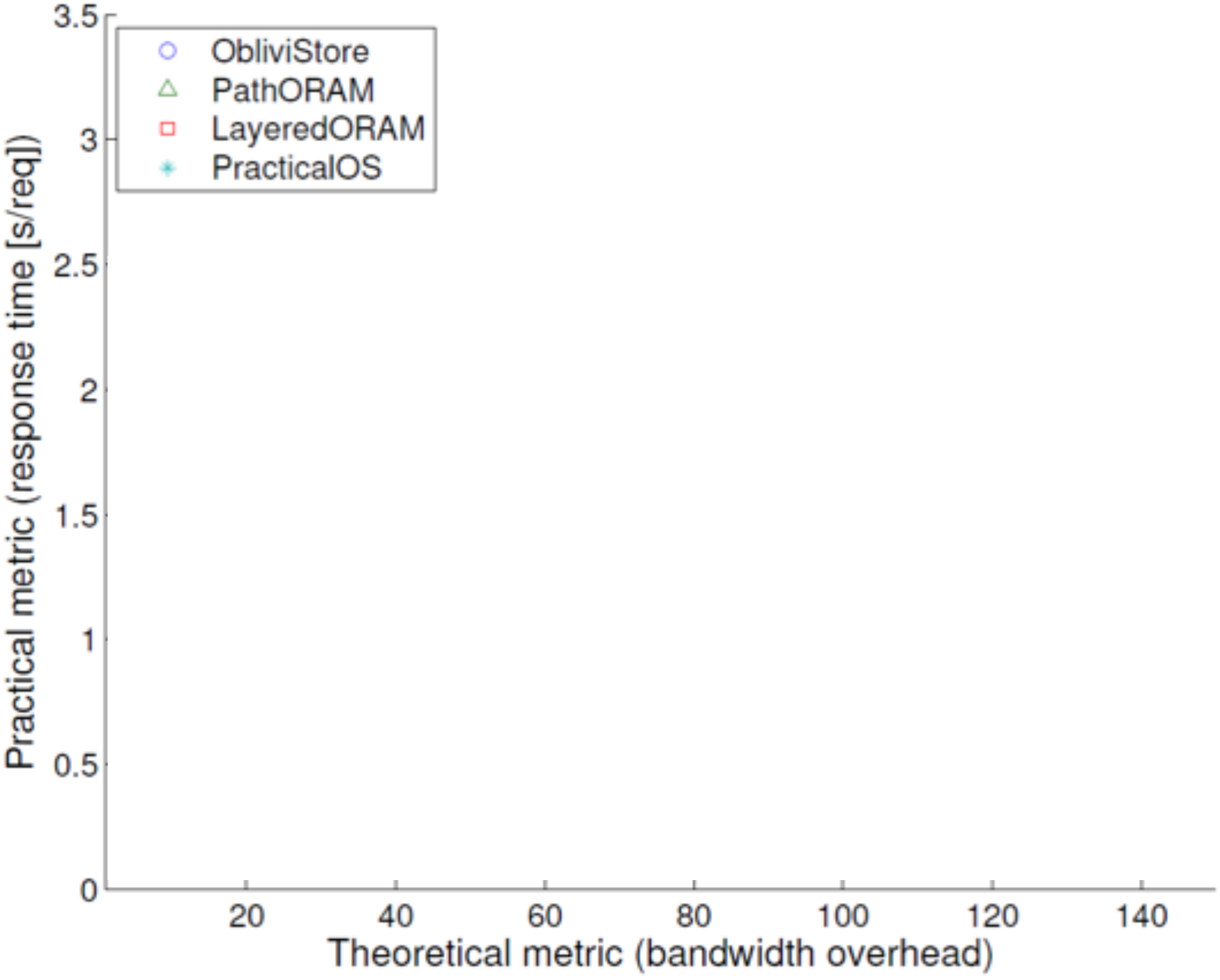


Methodology

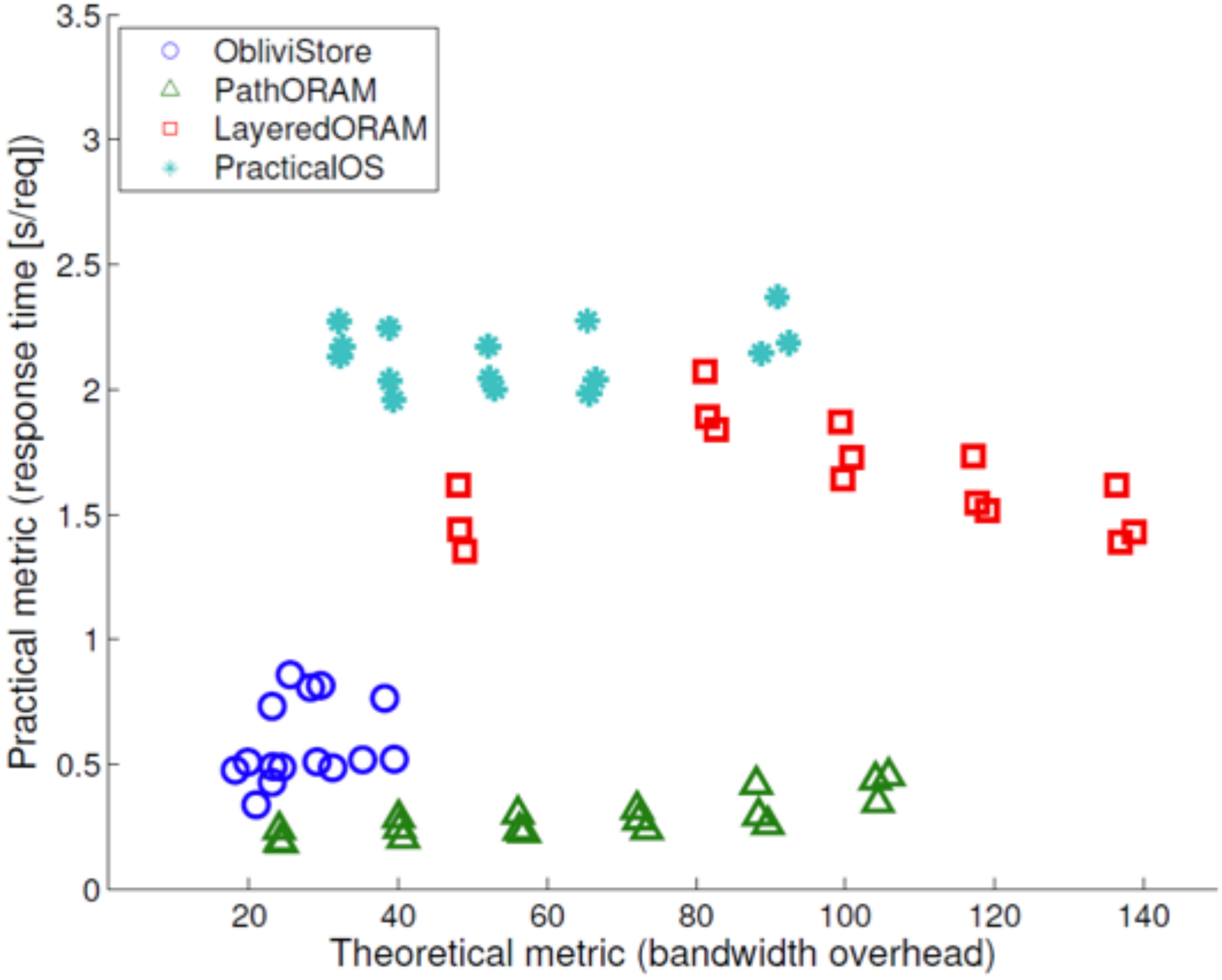


Findings

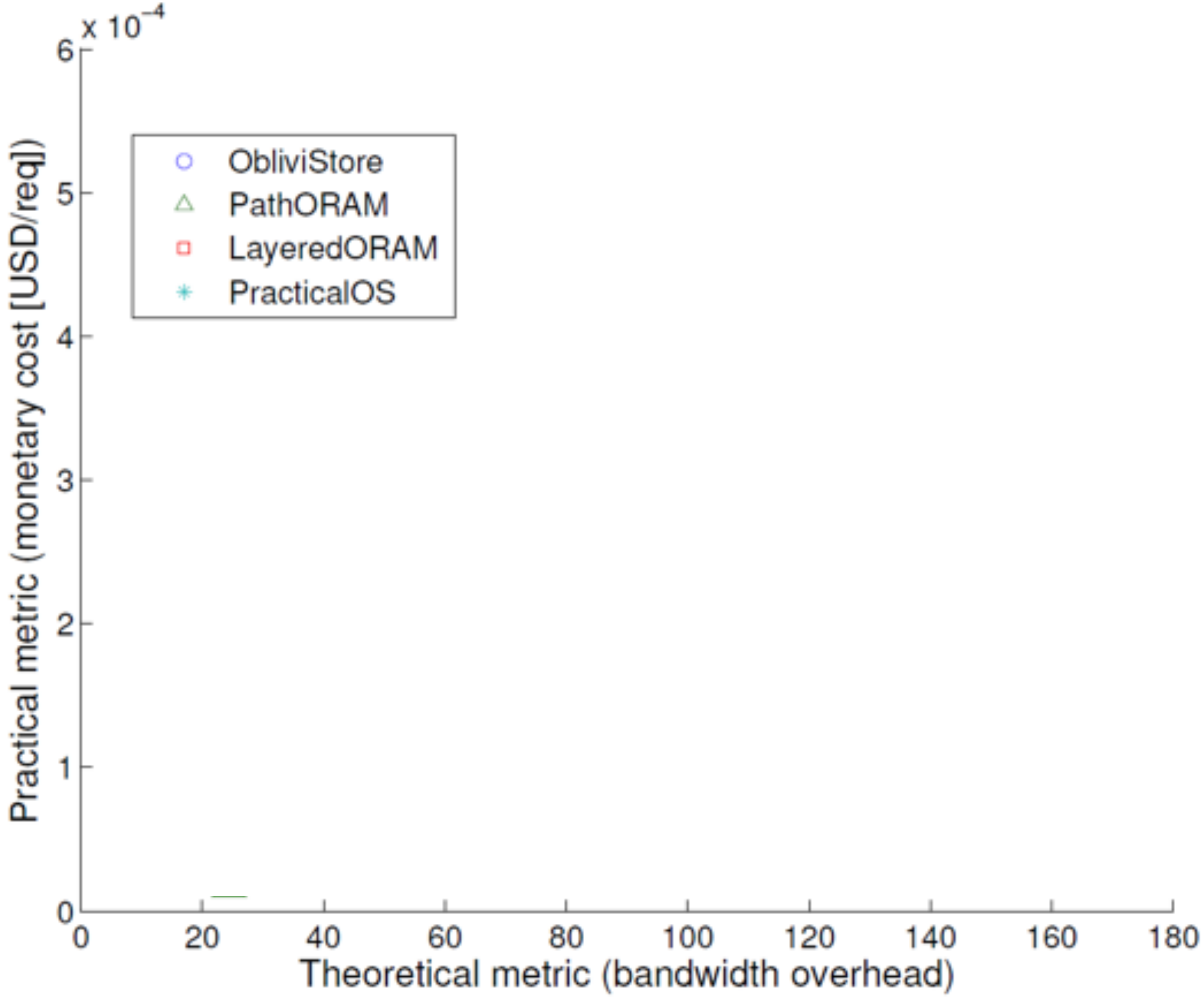
Bandwidth overhead as a proxy for response time



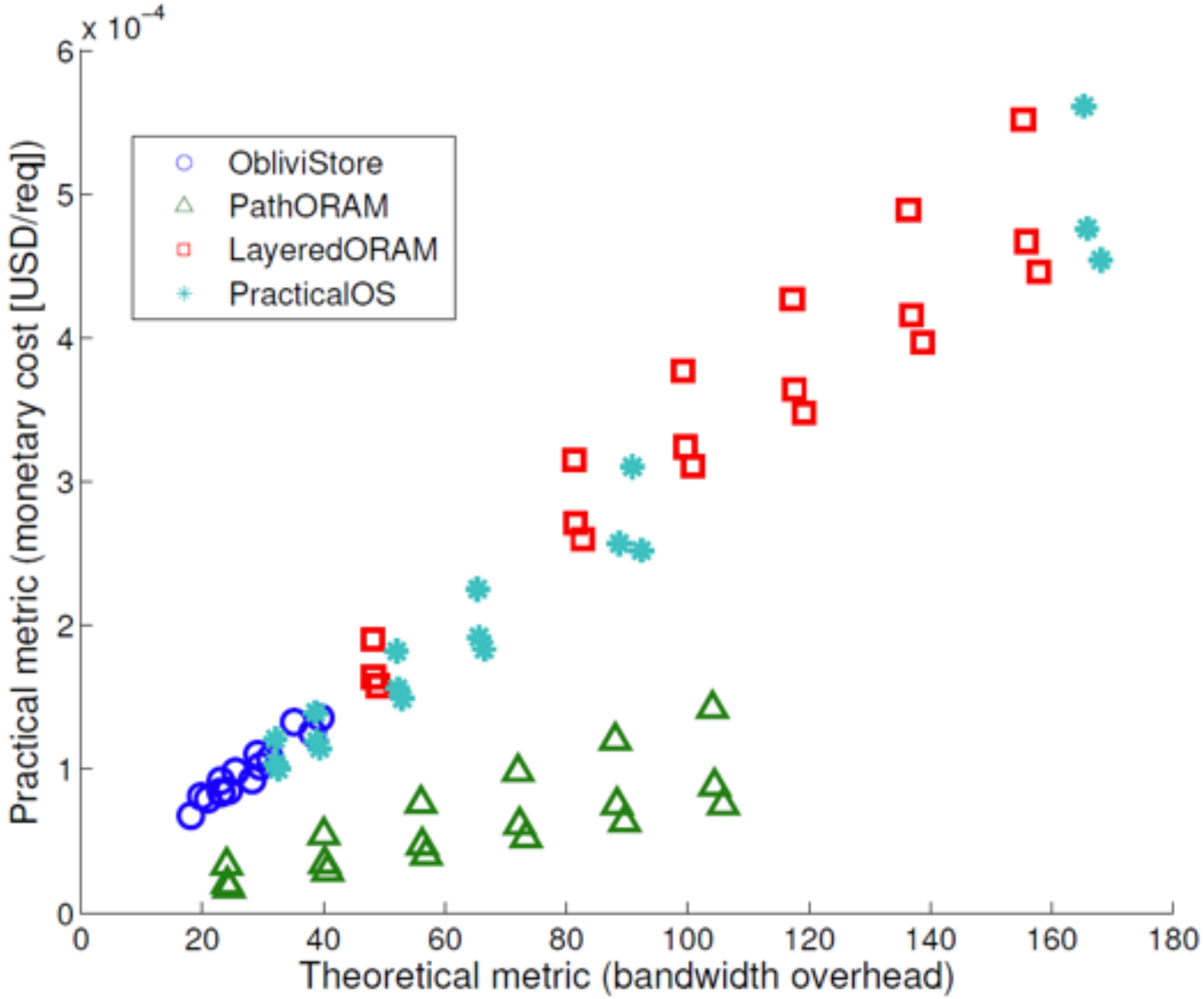
Bandwidth overhead as a proxy for response time



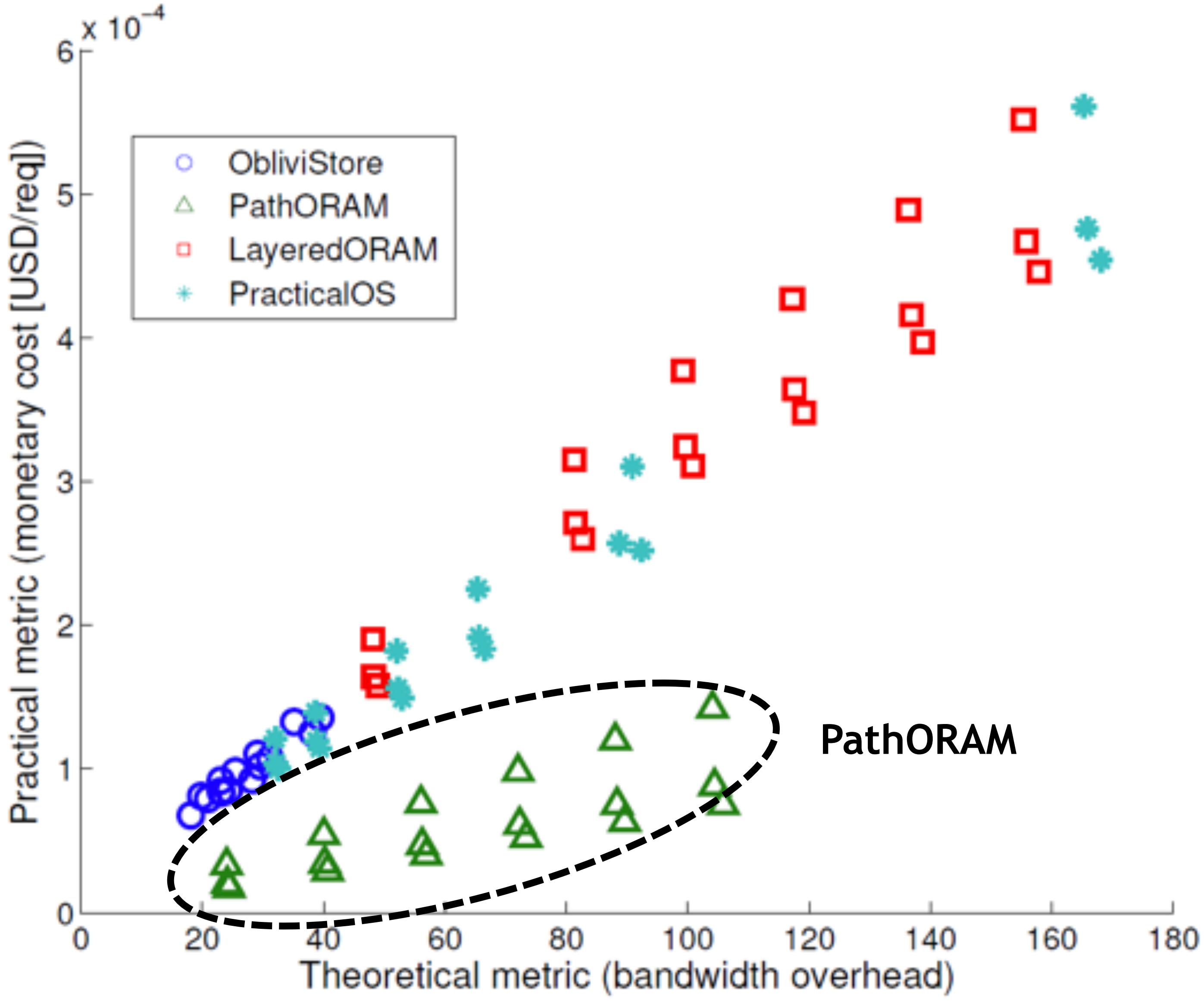
Bandwidth overhead as a proxy for monetary cost



Bandwidth overhead as a proxy for monetary cost

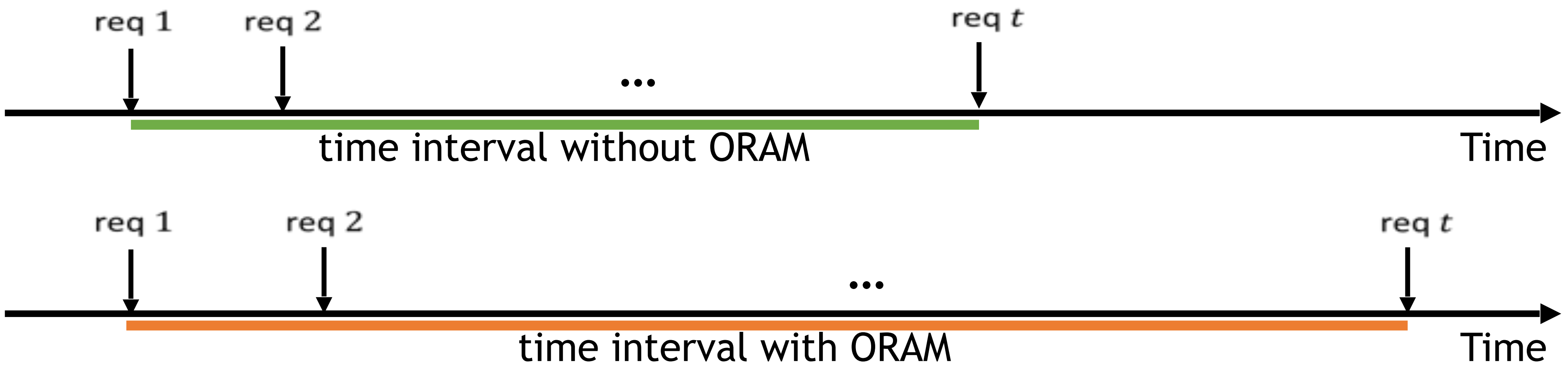


Bandwidth overhead as a proxy for monetary cost



Application traces

- What metric should be used?
  - Throughput?
  - Response time?
- We propose to use the *slowdown*:
  - Measures how much an ORAM scheme slows down an application
  - A slowdown of x means the time to replay an application trace on top of ORAM is x times that of without ORAM
- Slowdown := time with ORAM / time without ORAM



Application Traces

- According to slowdown measurements:
  - ObliviStore could easily handle two applications (i.e., varmail and webproxy), but could not handle the other two (i.e., webserver and fileserver)
  - PathORAM could not handle any of the four applications (it experienced slowdowns ranging from 3 to 92)

- In all cases, the monetary cost of running on top of ORAM was roughly 100 times (or more) than running without ORAM

PracticalOS & LayeredORAM

- Neither of the two schemes could support any of the applications
- PracticalOS has a low response time for requests
  - but a long and expensive reshuffling phase
- The cost of operating PracticalOS for varmail is roughly 15 USD / min

Main Findings

- Bandwidth overhead is **not** the bottleneck
- Network latency is the bottleneck

- Many real applications **require** the ORAM to process requests concurrently

- Downloads and uploads do **not** have the same cost

Asynchronicity & Concurrent Request Processing

- ObliviStore can process multiple requests concurrently and offer an asynchronous interface

- Others (e.g., PathORAM) are fundamentally synchronous
  - The current request must be fully completed before the processing of the next request can start

- ORAM schemes do not appear to consider **asynchronicity** as a crucial property
  - 3 out of 39 published papers have this property

Asynchronicity is a MUST!

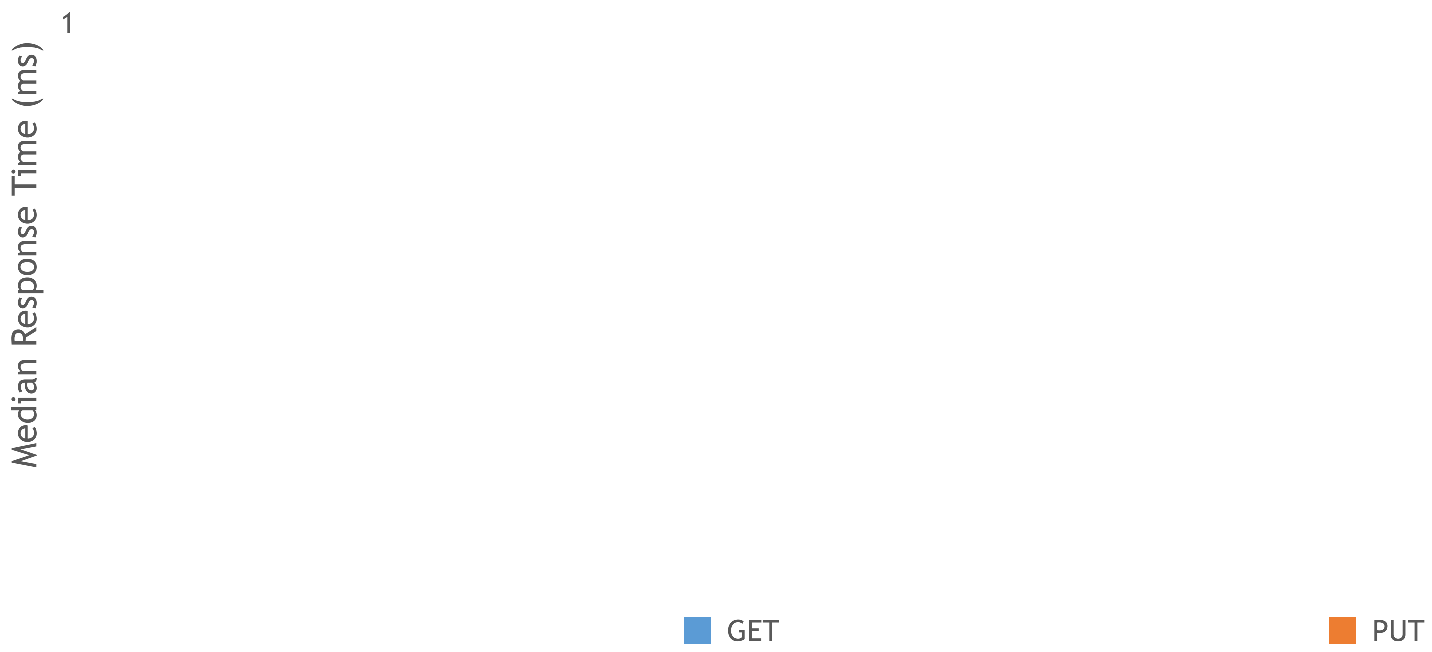
- **Asynchronicity has never been a main design goal.**
- **But**, we found that:
  1. Asynchronicity is not only desirable but actually **necessary**
     - No synchronous ORAM scheme can fully support cloud applications
  2. Asynchronicity is difficult
     - E.g., the implementation of ObliviStore did not get it right

Bandwidth Asymmetricity

- S3: the monetary cost of an upload is 12.5 times that of a download

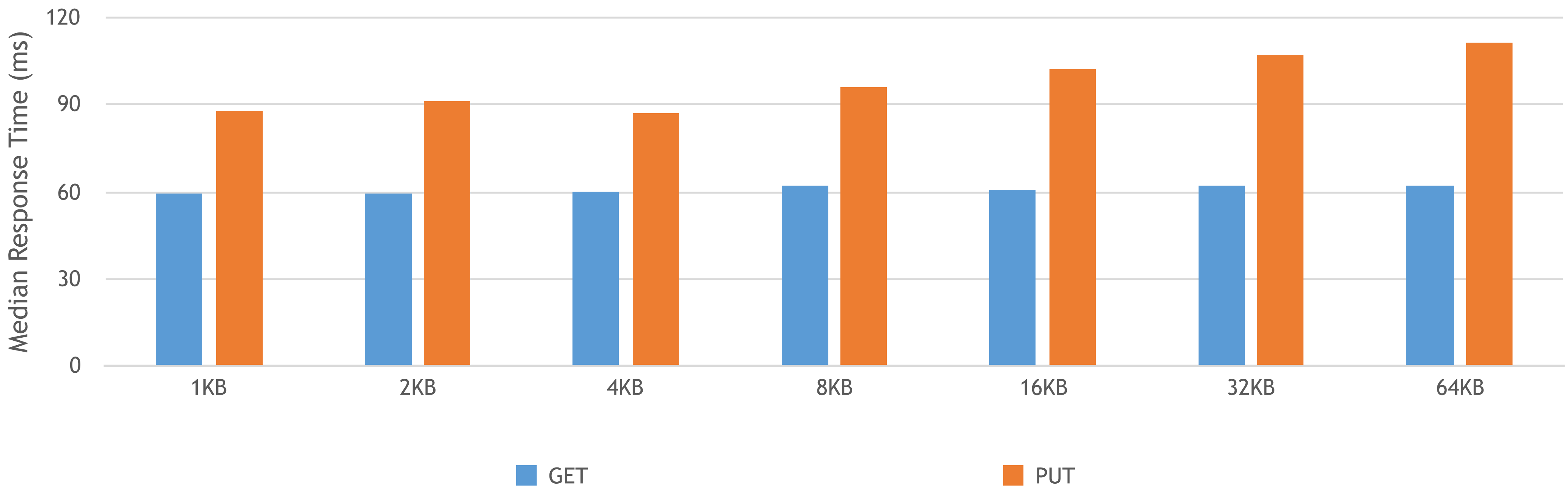
Bandwidth Asymmetricity

- S3: the monetary cost of an upload is 12.5 times that of a download



Bandwidth Asymmetricity

- S3: the monetary cost of an upload is 12.5 times that of a download



Bandwidth-only evaluation is INACCURATE!

- **Overhead evaluation: total bandwidth only in existing literature**
  - Bandwidth overhead := download overhead + upload overhead

- **But**, experimentally, their performance and monetary cost are different
  - Failure to incorporate this experimental insight in our thinking could lead us to make incorrect conclusions about how schemes perform in practice
  - Example: which is better?
    - Scheme 1: 20 download overhead, 20 upload overhead
    - Scheme 2: 40 download overhead, 10 upload overhead

CURIOUS

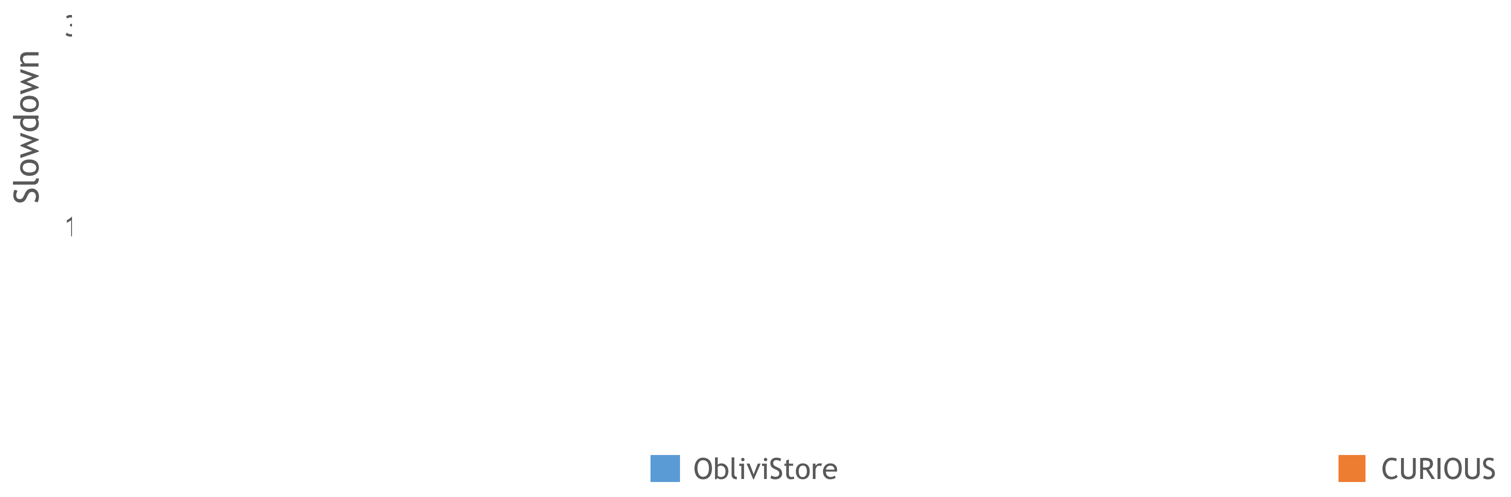
Novel ORAM Framework: CURIOUS

- Based on our findings, we propose CURIOUS

- **Simple design:**
  - Flexible due to modular design
  - Simple concurrency model

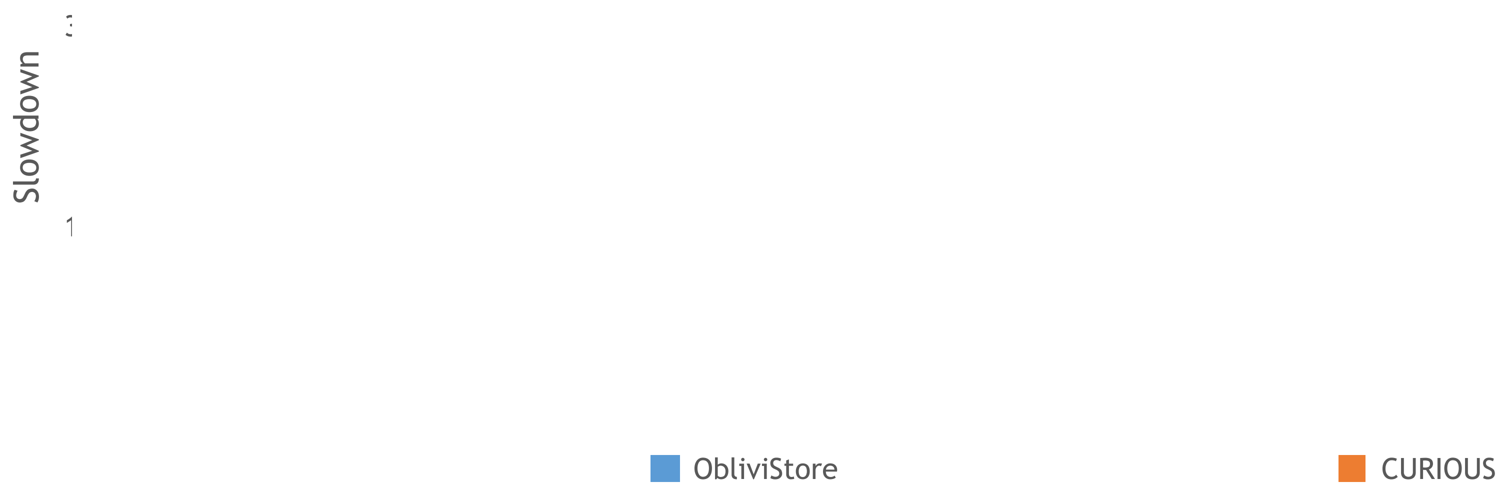
- Also, it preserves properties that applications expect from cloud
  - e.g., reliability

CURIOUS performs better than ObliviStore



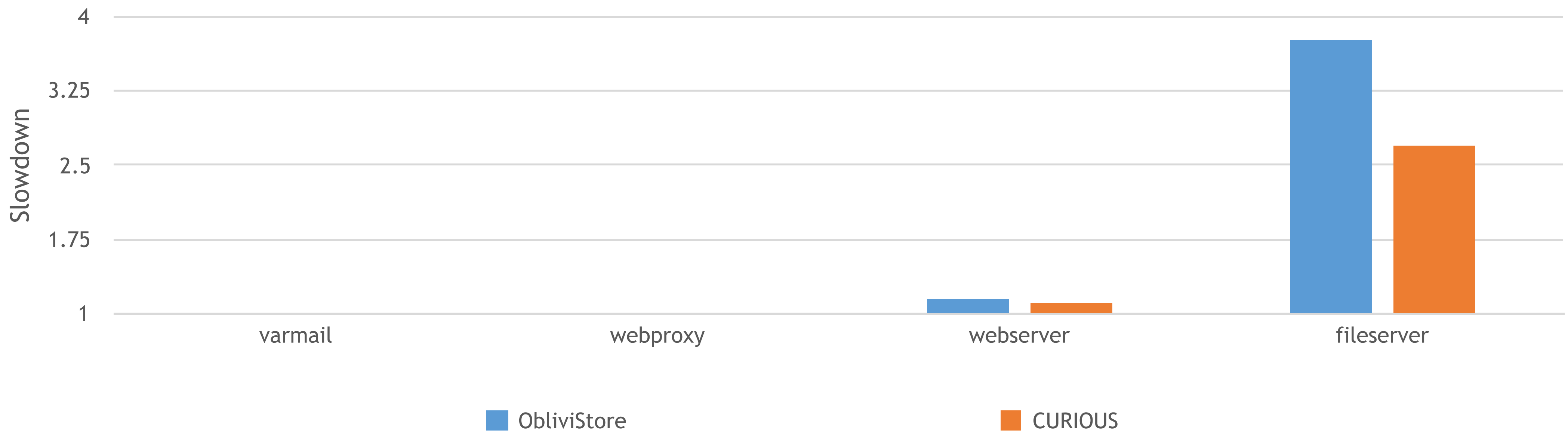
CURIOUS performs better than ObliviStore

- Monetary cost is only half to two-thirds



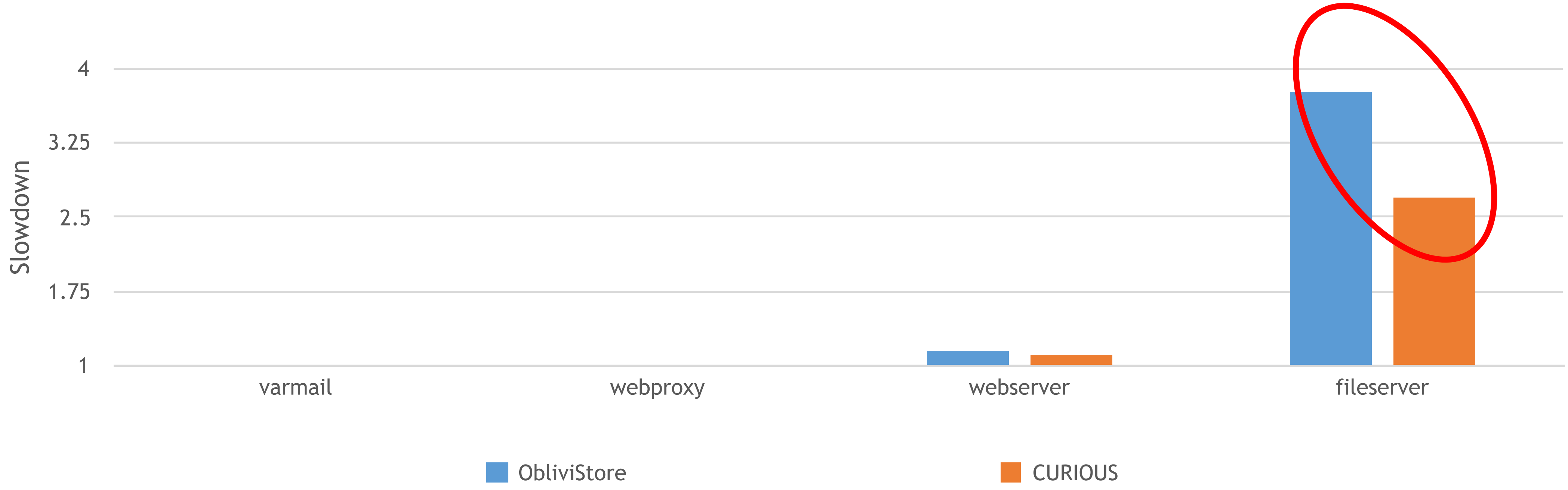
CURIOUS performs better than ObliviStore

- Monetary cost is only half to two-thirds



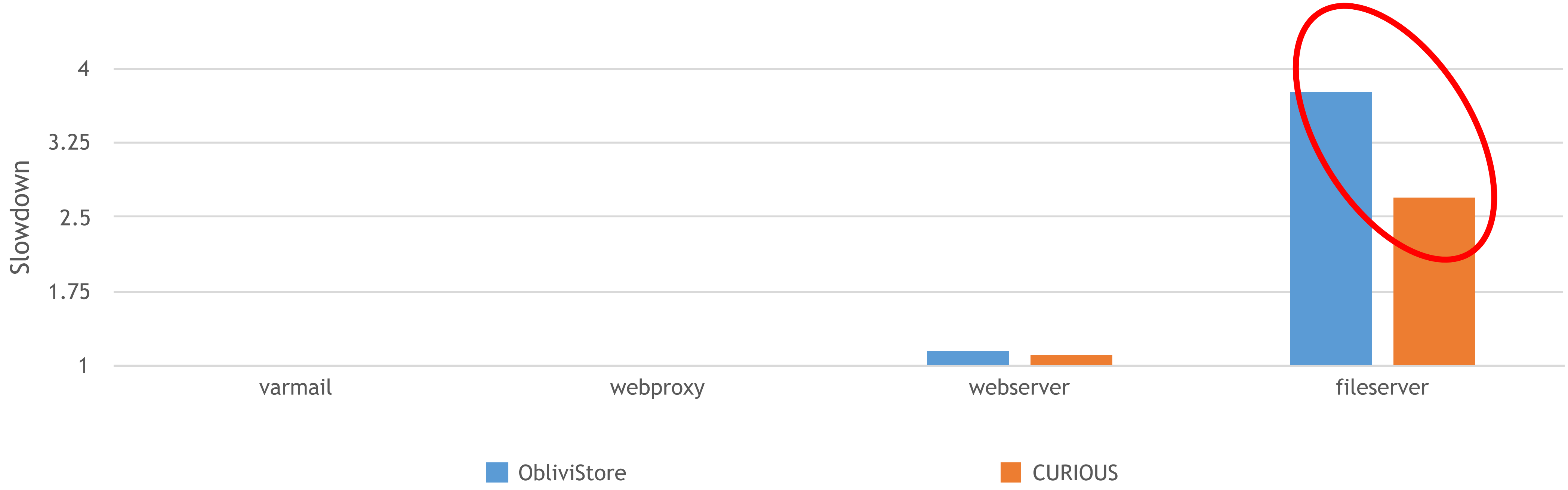
CURIOUS performs better than ObliviStore

- Monetary cost is only half to two-thirds



CURIOUS performs better than ObliviStore

- Monetary cost is only half to two-thirds
- Even though
  - CURIOUS uses **2X** the bandwidth of ObliviStore



Conclusions

- Oblivious RAM has come a long way…
- … and there is a long way to go still…
- But we found:
  - In theory there is no difference between theory and practice
  - But in practice, there is.

- Lesson:
  - **align theory to practice**
  - **evaluate theory on practical systems**

Open-Source Code (BSD license)

- Our entire system including CURIOUS, the 4 representative ORAM schemes (PathORAM, LayeredORAM, PracticalOS, ObliviStore), and our evaluation platform is open-source.
- Uses Amazon S3 as storage backend.
- Download URL: oblivious-storage.com
- Contact: bindsch2@illinois.edu