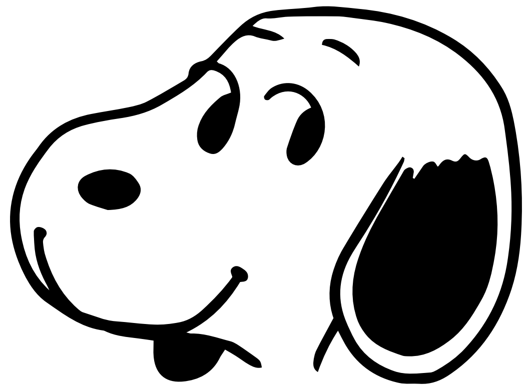
Snoopy: Surpassing the Scalability Bottleneck of Oblivious Storage

Emma Dauterman*, Vivian Fang*,
Ioannis Demertzis[†], Natacha Crooks, and Raluca Ada Popa
UC Berkeley, [†] UC Santa Cruz



SOSP 2021

** denotes equal contribution*

End-to-end encryption provides confidentiality

Attacker can’t see data contents

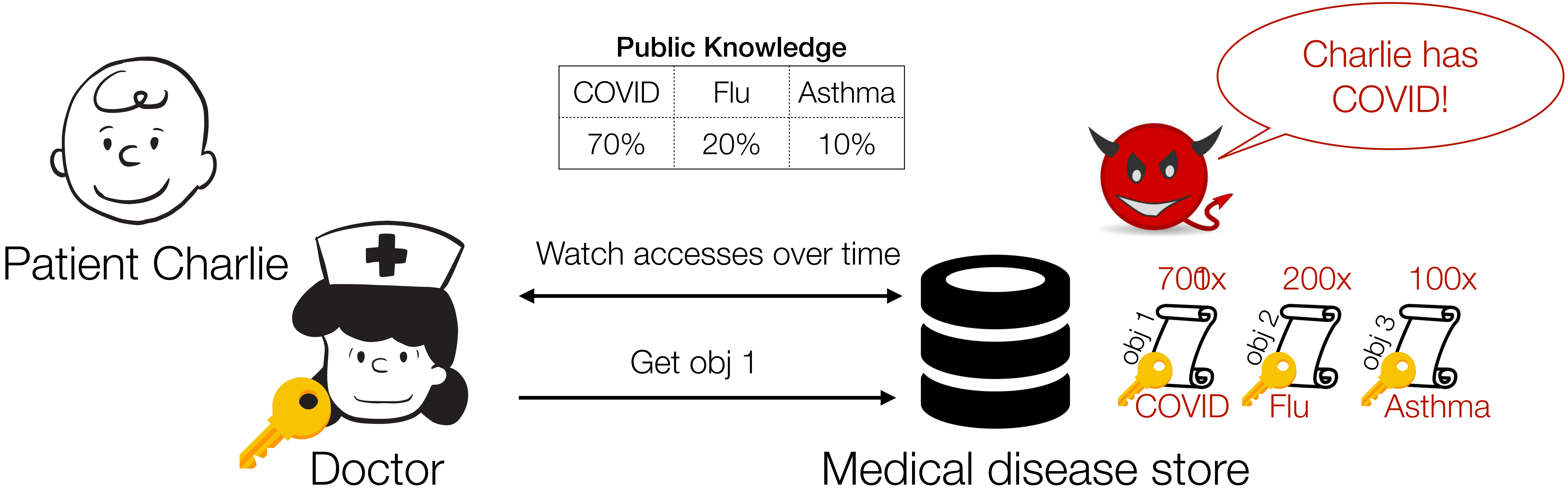
Access patterns reveal private data

[IKK12], [CGPR15], [KKNO16], [GLMP19], [KPT19]

Access patterns: how user accesses data.

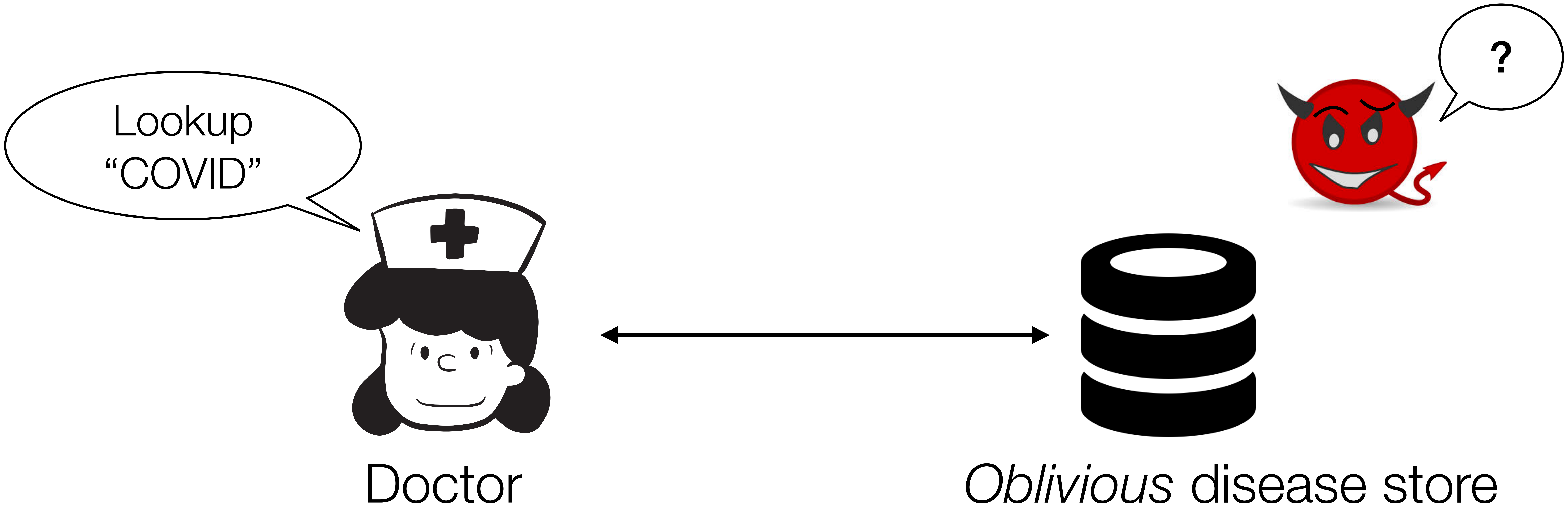
Public Knowledge

COVID	Flu	Asthma
70%	20%	10%



Oblivious storage (ORAM) protects access patterns

[GO96], SSS ORAM, PathORAM, RingORAM, Oblix, Shroud, TaoStore, Obladi, PrivateFS, …

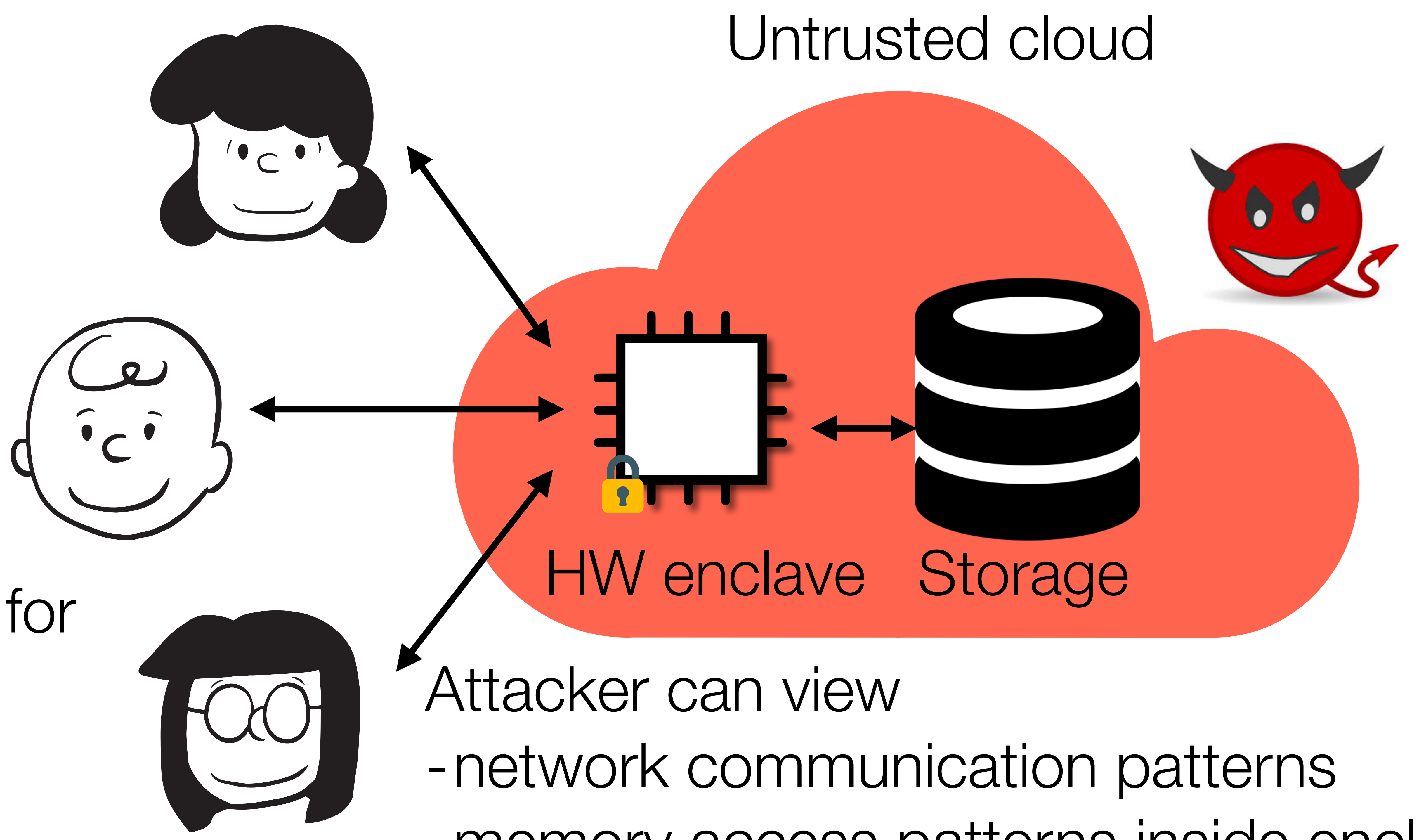


System setup

Oblix, ZeroTrace, Obliviate

HW enclave setup supports multiple users and reduces network interaction

Clients trust each other for simplicity (see paper)



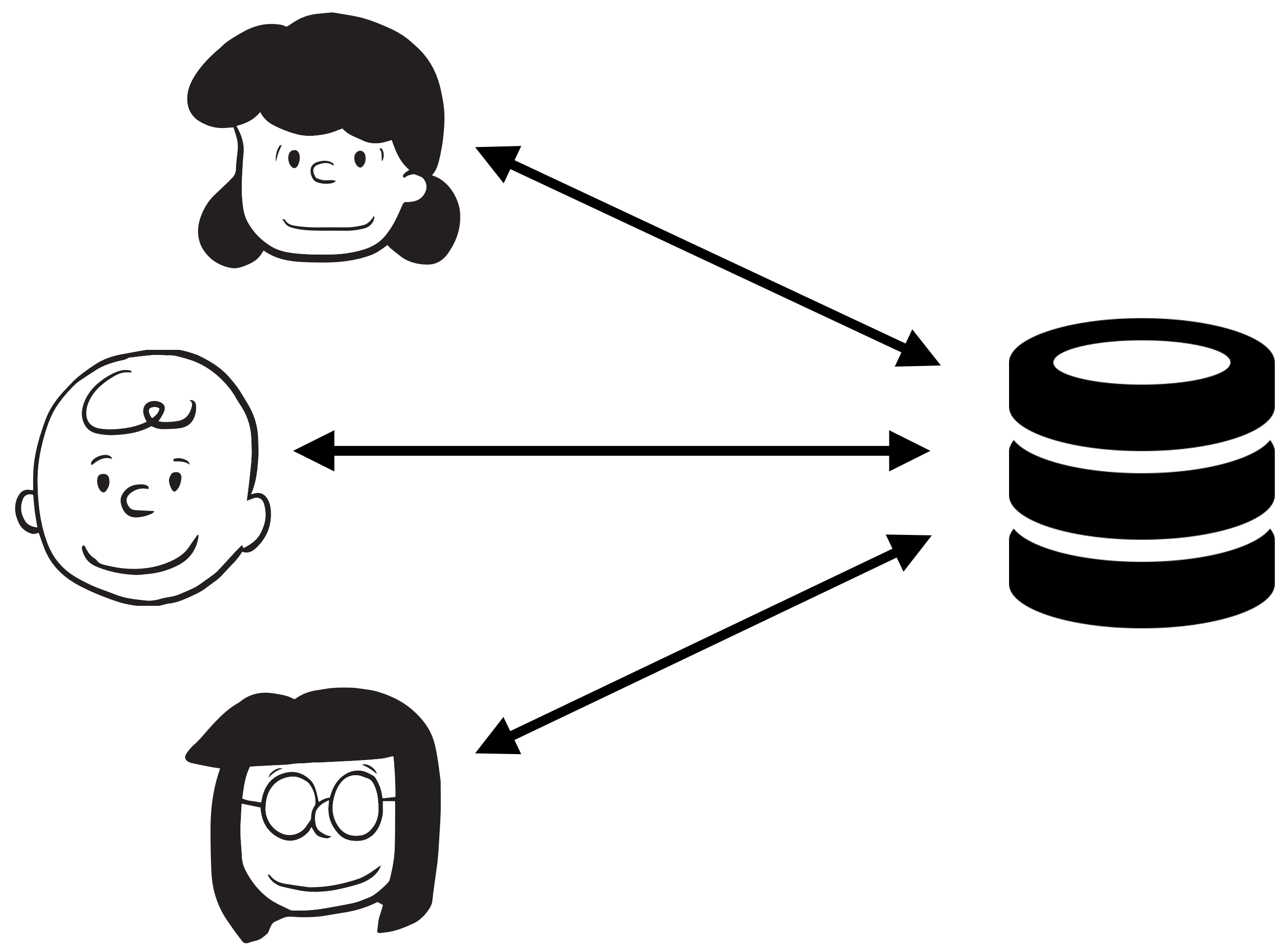
Attacker can view
- network communication patterns
- memory access patterns inside enclave

but *not* enclave contents

Existing systems have scalability bottlenecks

Scalability bottleneck:

- Coordination required for every request
- Cannot securely distribute



Existing systems have scalability bottlenecks

Scalability bottleneck:

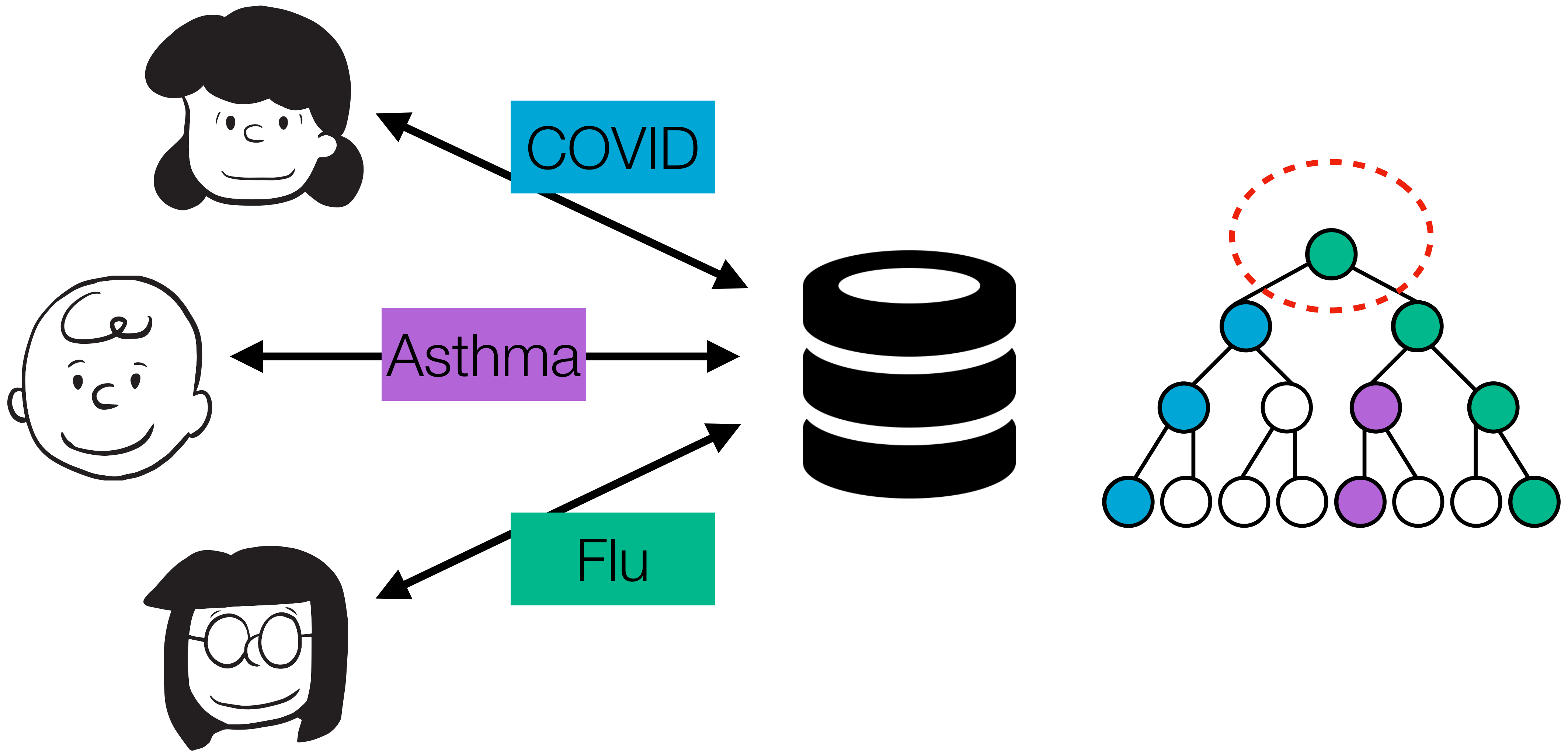
- Coordination required for every request
- Cannot securely distribute

Most systems are tree-based and hide locations of objects in the tree.

Common bottlenecks:

-Location metadata

-Tree root



Existing systems have scalability bottlenecks

Scalability bottleneck:

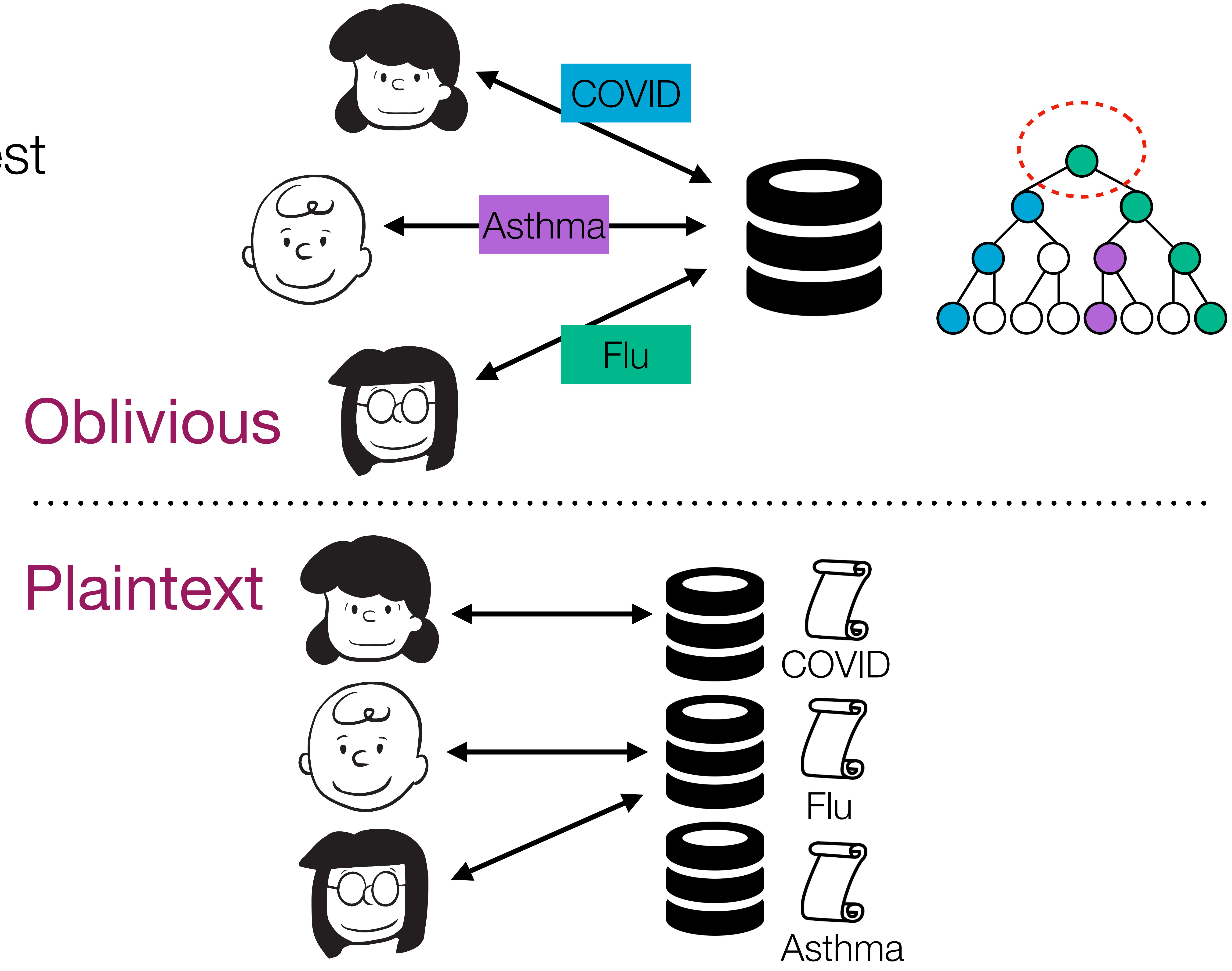
- Coordination required for every request
- Cannot securely distribute

Most systems are tree-based and hide locations of objects in the tree.

Common bottlenecks:

-Location metadata

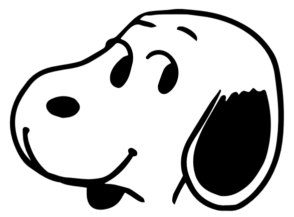
-Tree root

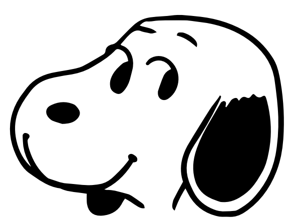


Goal of this talk:

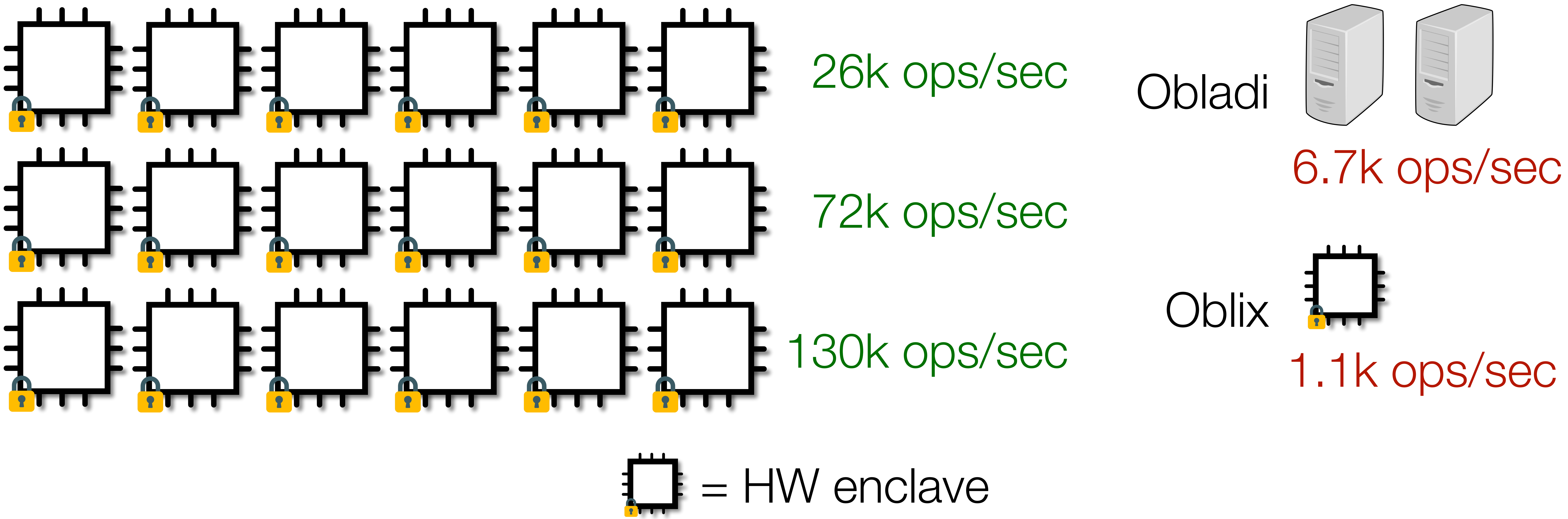
How can we build an **oblivious** object store that handles **high throughput** by **scaling like a plaintext object store**?

This talk: **S**calable **n**odes for **o**blivious **o**bject re**p**ositor**y**

This talk: Snoopy 

This talk: Snoopy 

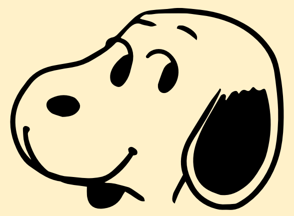
Snoopy is an oblivious object store that scales like plaintext storage.



Outline

1. Design idea
2. Load balancer
    A. Batch structure
    B. Oblivious algorithms
3. SubORAM
4. Evaluation

Outline

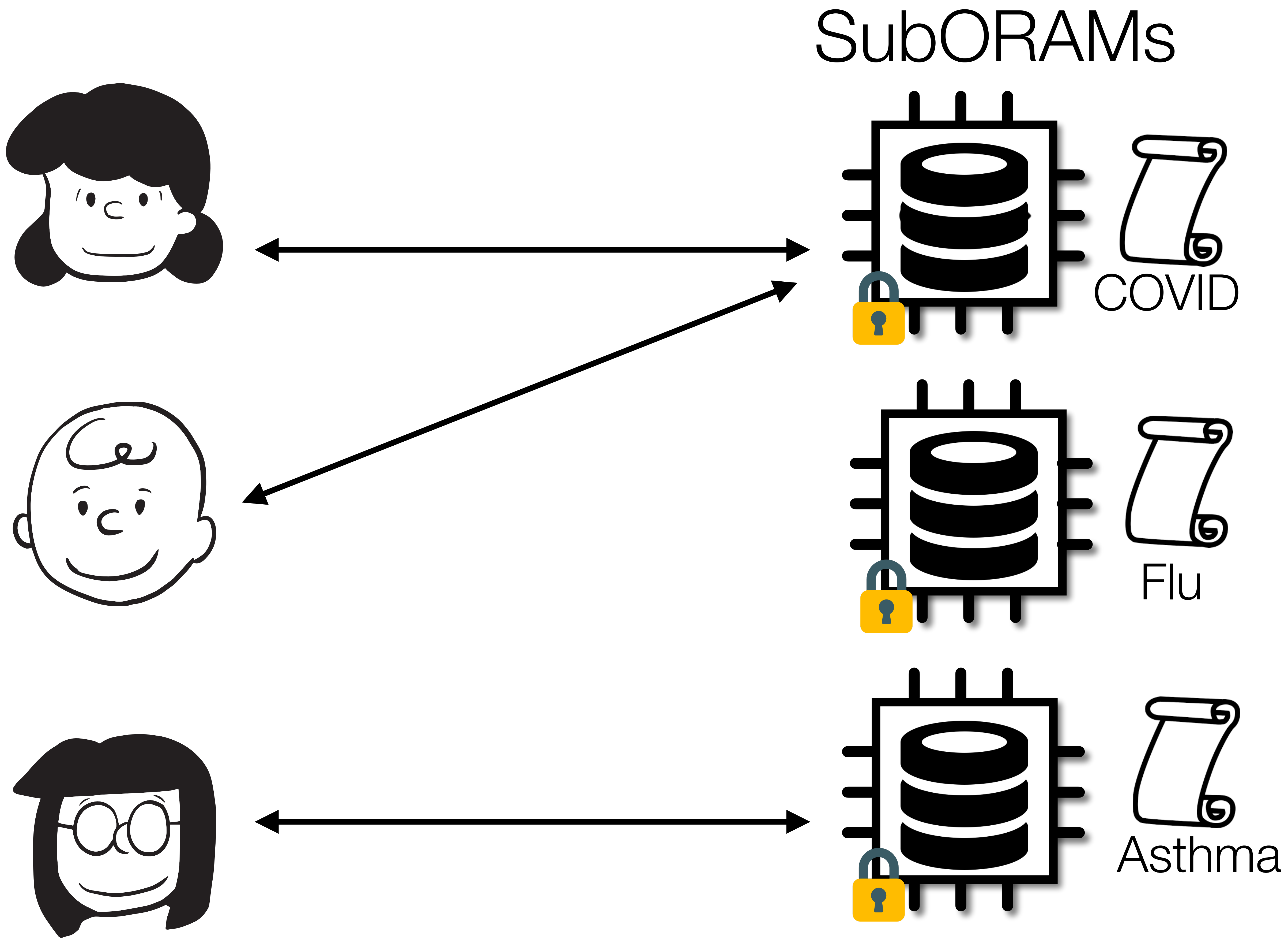
1. Design idea 
2. Load balancer
   A. Batch structure
   B. Oblivious algorithms
3. SubORAM
4. Evaluation

Building Snoopy

Classic techniques

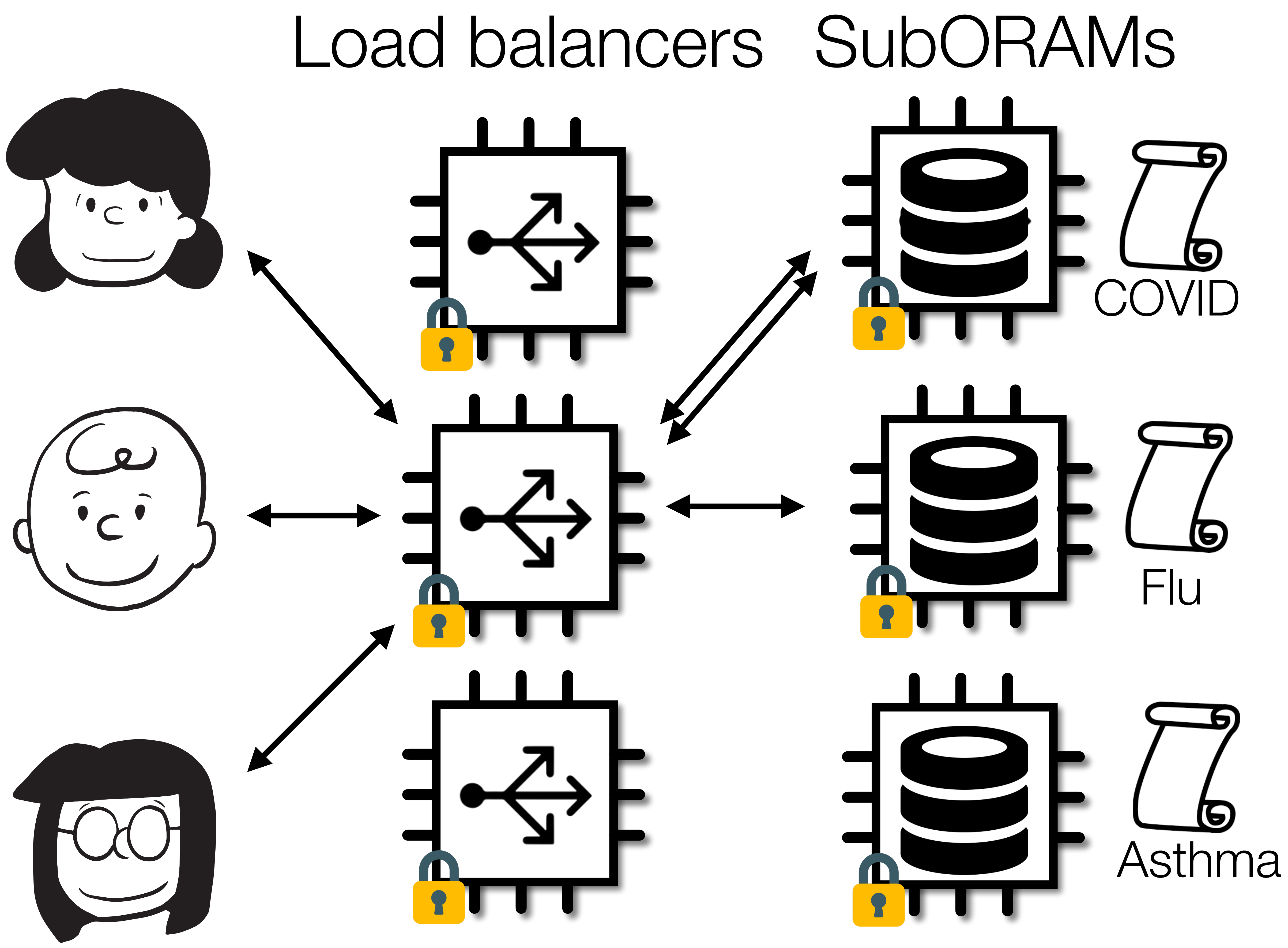
Building Snoopy

Classic techniques
Partitioning



Building Snoopy

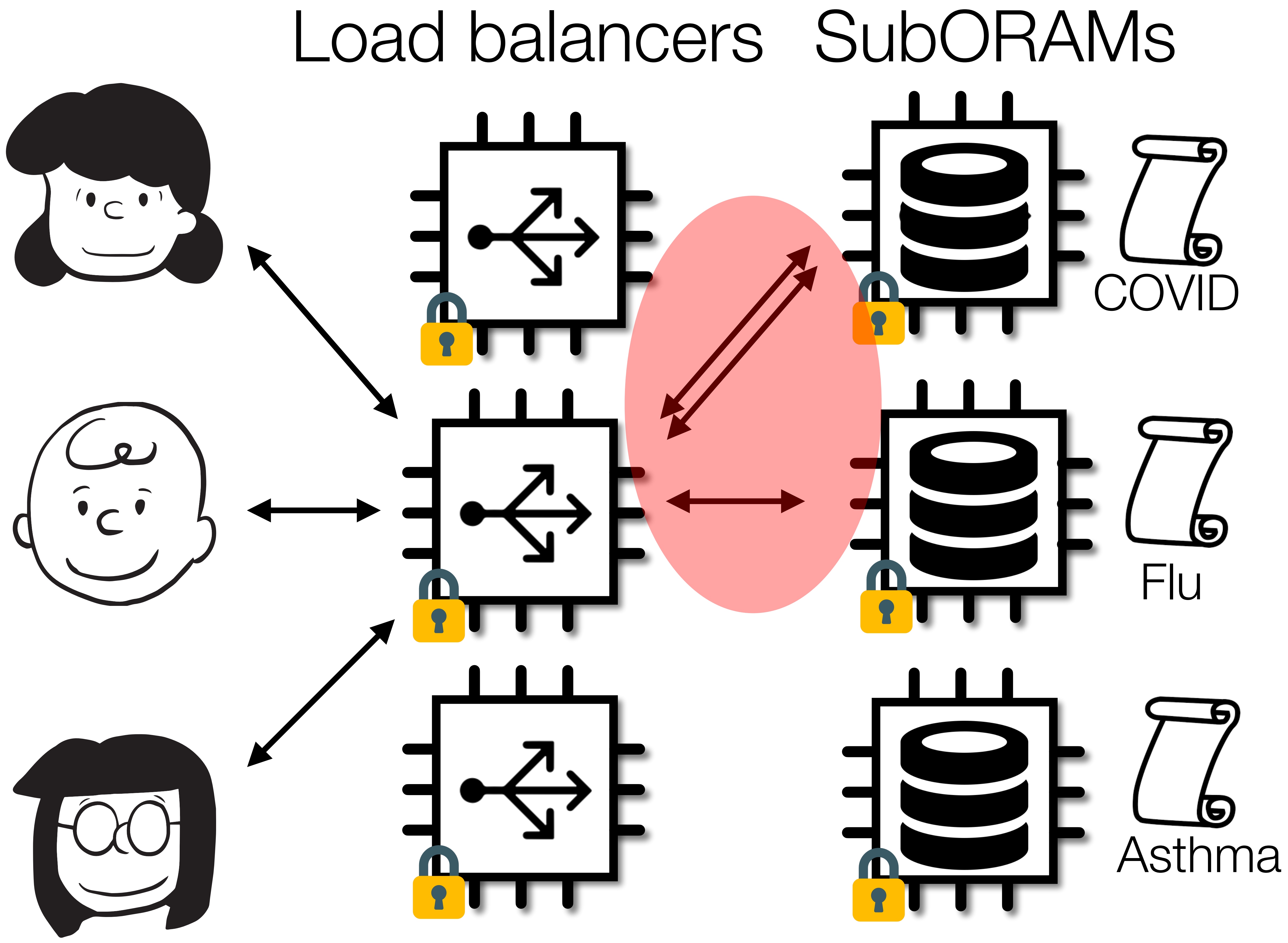
Classic techniques
Partitioning
Batching



Building Snoopy

Classic techniques
Partitioning
Batching

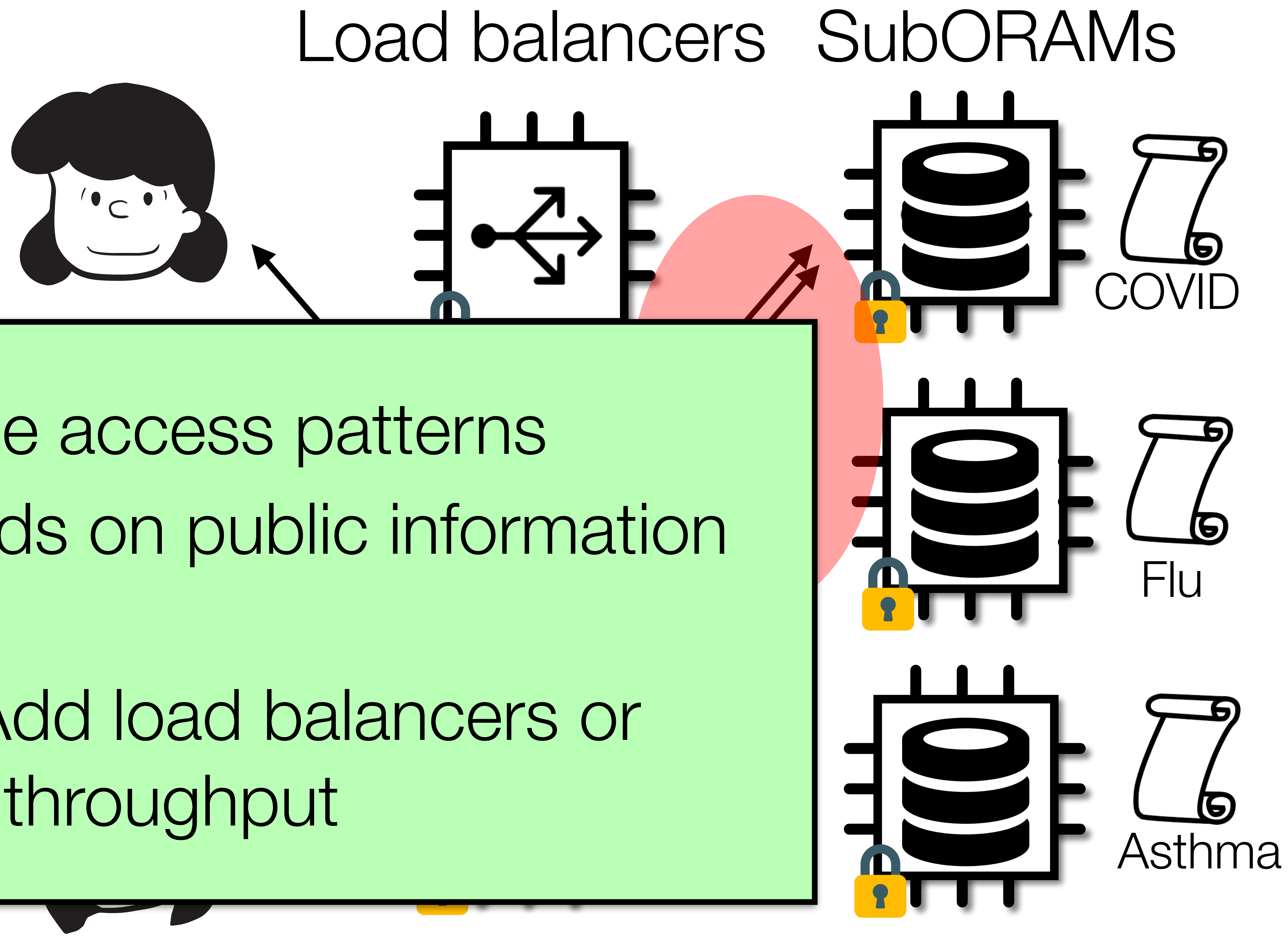
😨 **Naively insecure**
Batches sent to subORAMs reveals request distribution



Building Snoopy

Classic techniques
Partitioning
Batching

😨 **Naively insecu**
Batches sent to su
request distributio



Goal #1 (Security): Hide access patterns
-Batch size only depends on public information

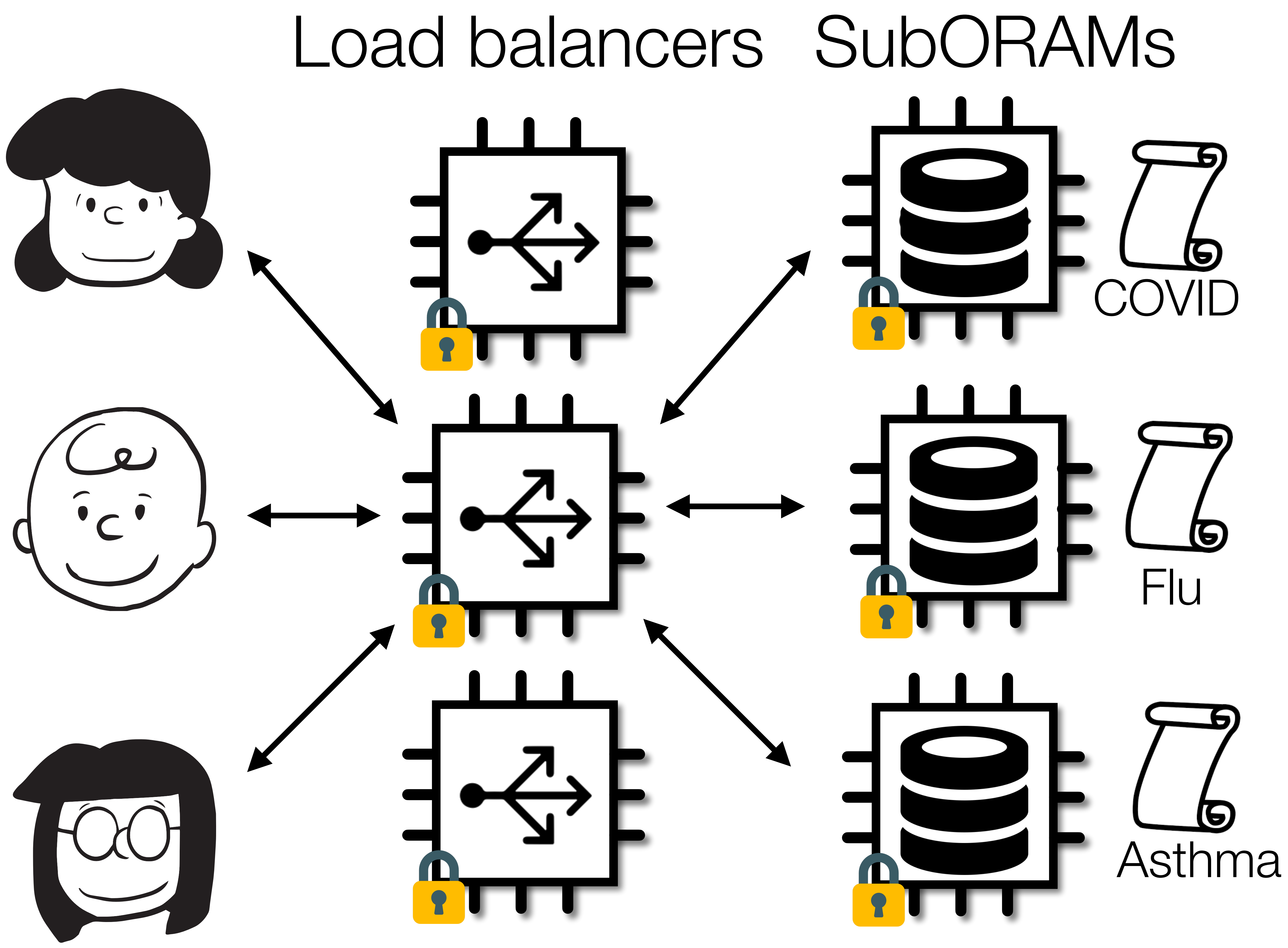
Goal #2 (Scalability): Add load balancers or subORAMs to increase throughput

Building Snoopy

Classic techniques
Partitioning
Batching

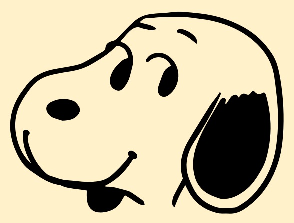
😨 **Naively insecure**
Batches sent to subORAMs reveals request distribution

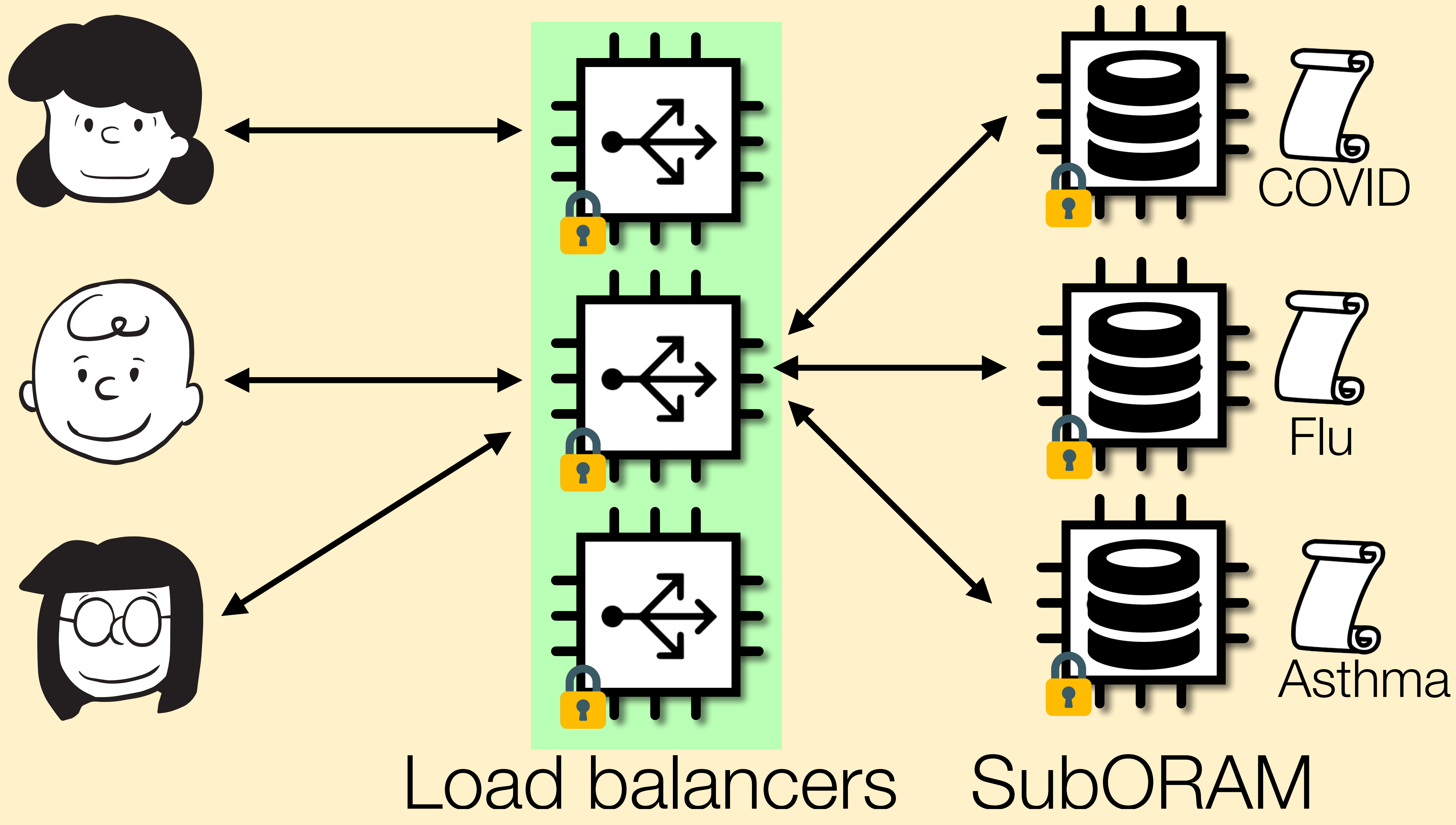
💡 **Our contributions**
Techniques that enable *batching* + *partitioning* with *security* + *scalability*



Outline

1. ~~Design idea~~
2. Load balancer
   A. **Batch structure**
   B. Oblivious algorithms
3. SubORAM
4. Evaluation



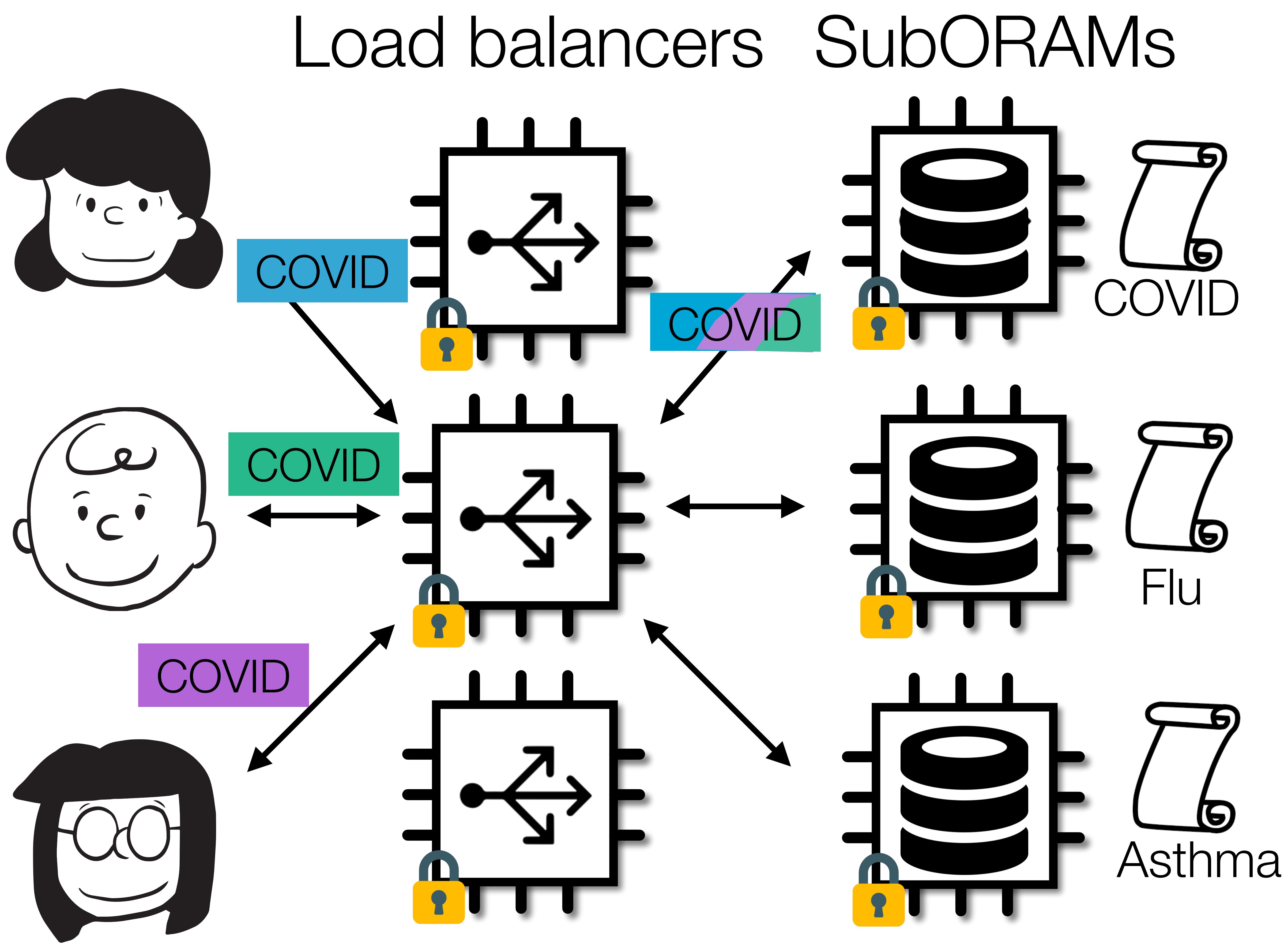


Handling skewed workloads

If every client requests the same object, then batch size = total requests
→ not scalable!

💡 Deduplication

Now we only need to handle **distinct** requests.



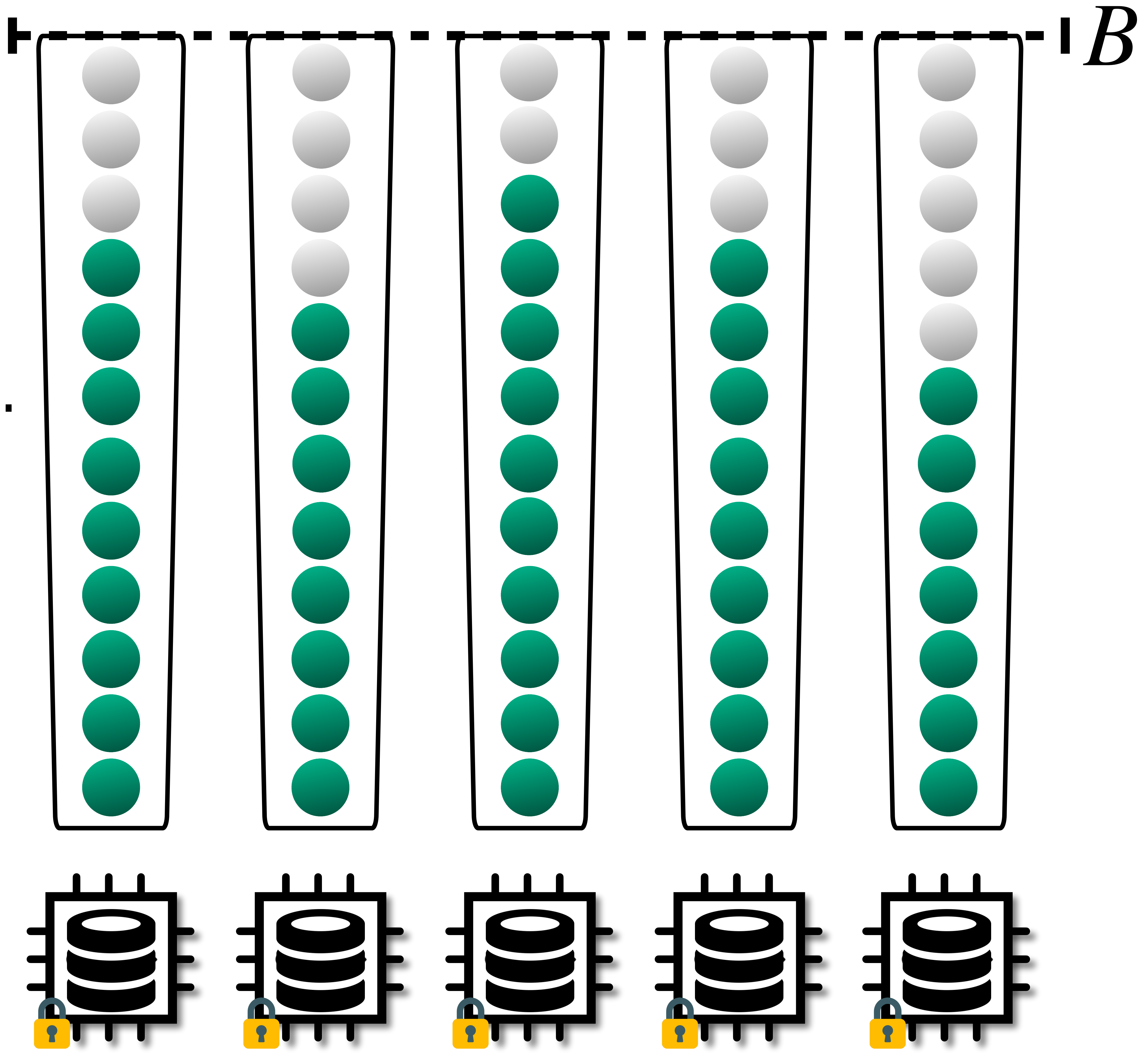
Securely setting batch size B

Requirements
-Computable with public information
-Negligible overflow probability

High-throughput → many concurrent requests.

After deduplication, requests are spread across subORAMs (≈) evenly.

💡 Don't need to add many dummy requests to have secure batch size.



Securely setting batch size B

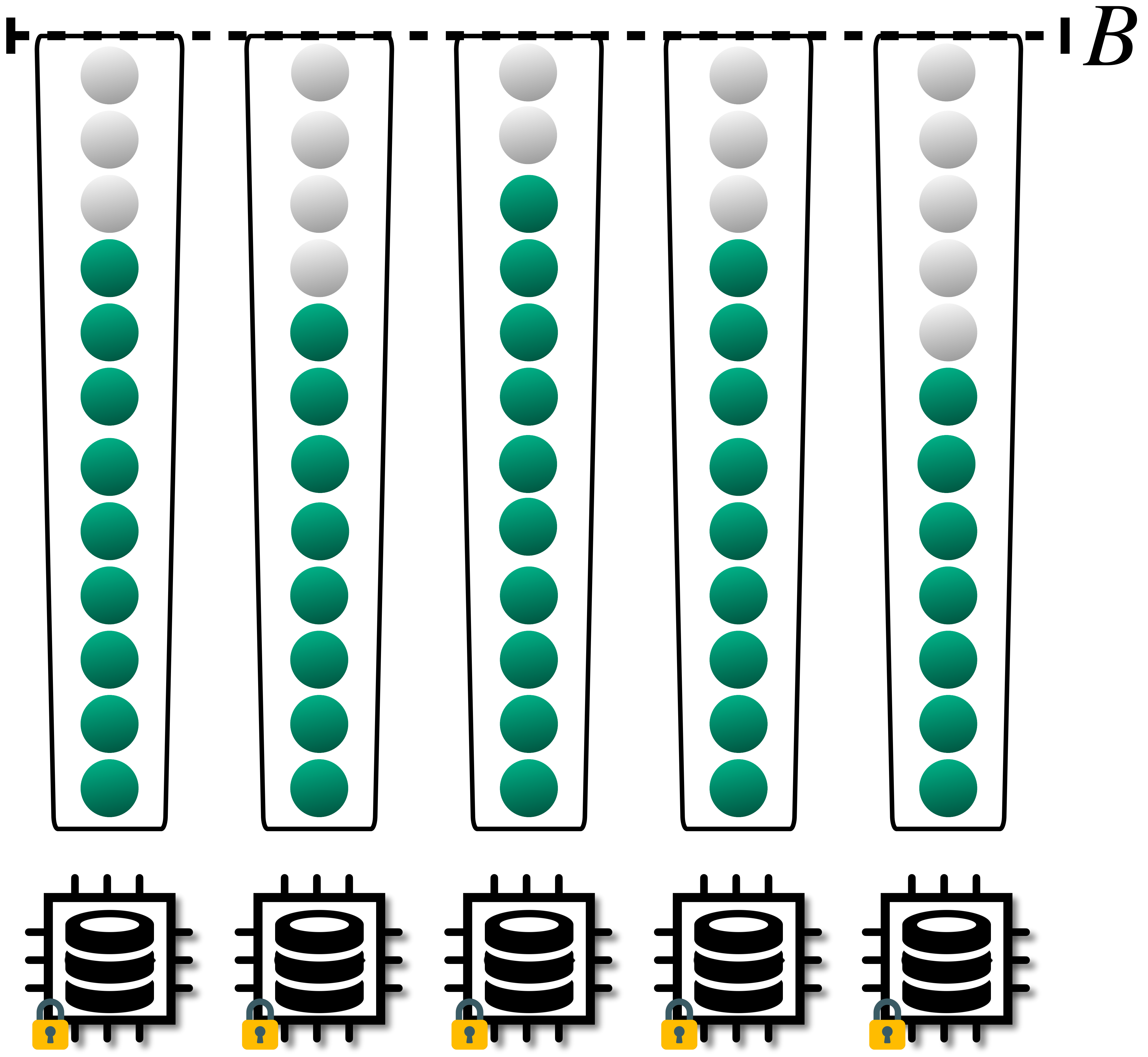
Requirements

-Computable with public information

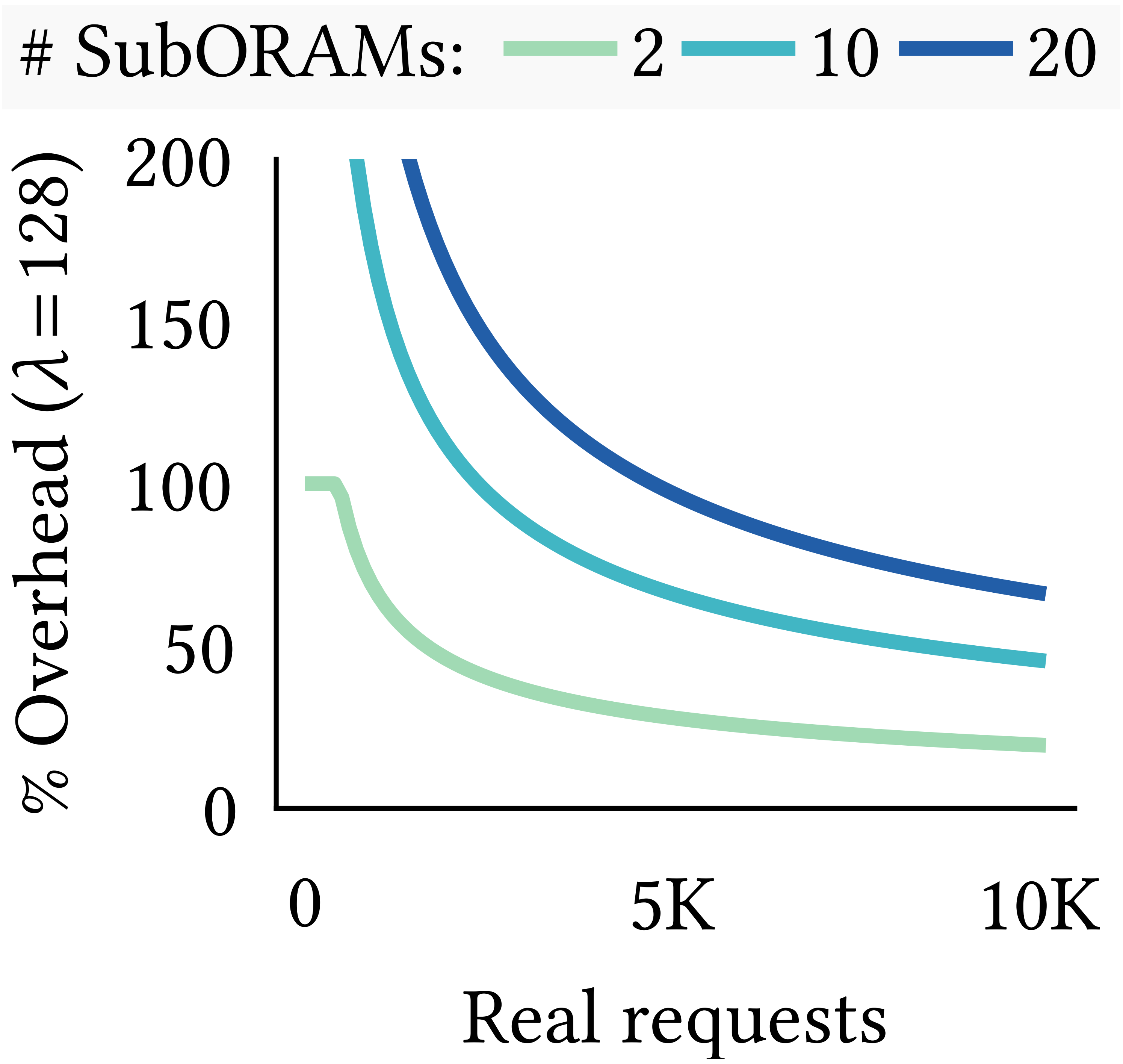
-Negligible overflow probability

Can model as a balls-into-bins problem.

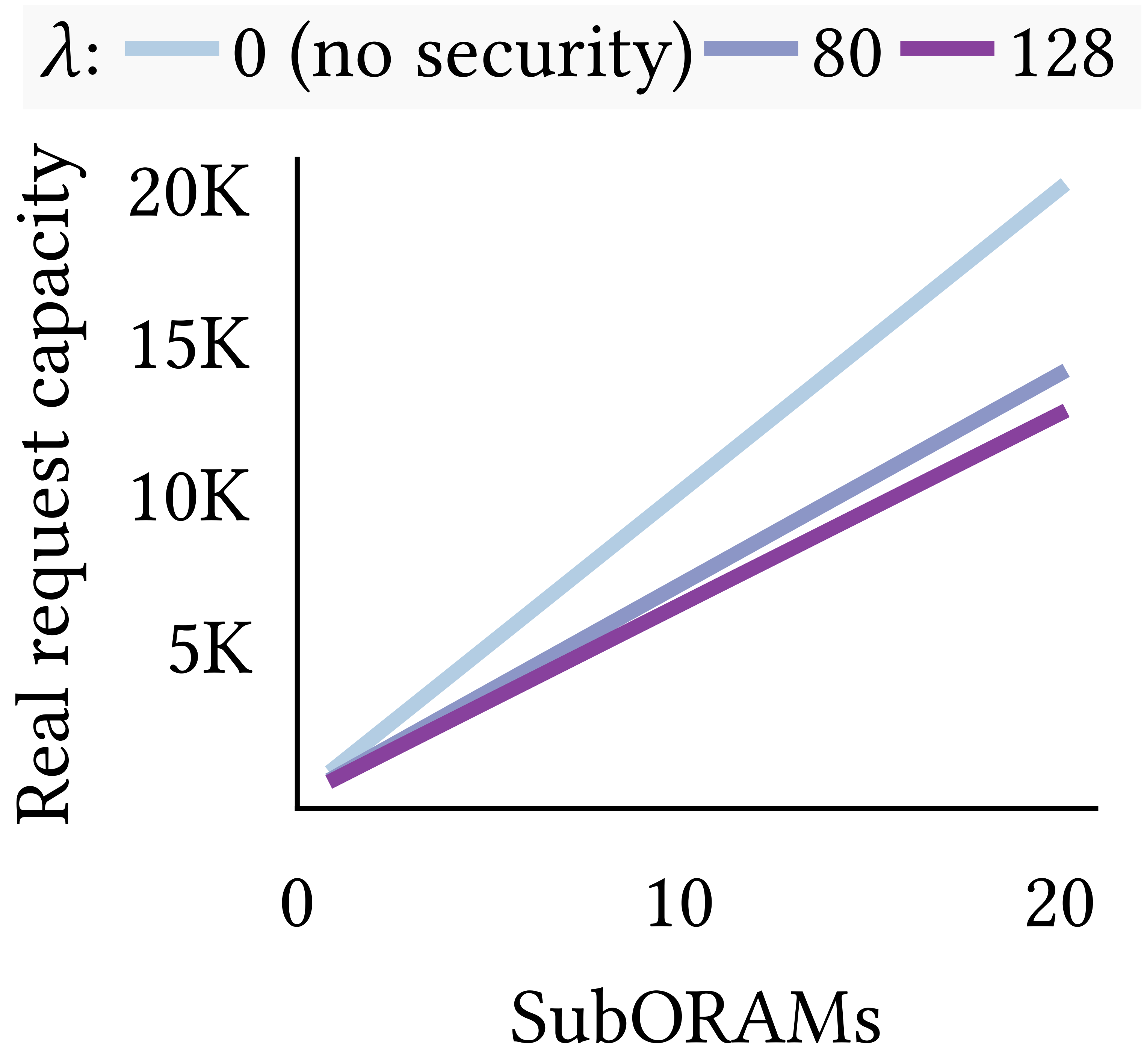
💡We contribute a bound that meets both requirements and provides scalability.



Securely setting batch size B



Requests ↑, dummy overhead ↓



SubORAMs ↑, request capacity ↑
(and dummy overhead ↑)

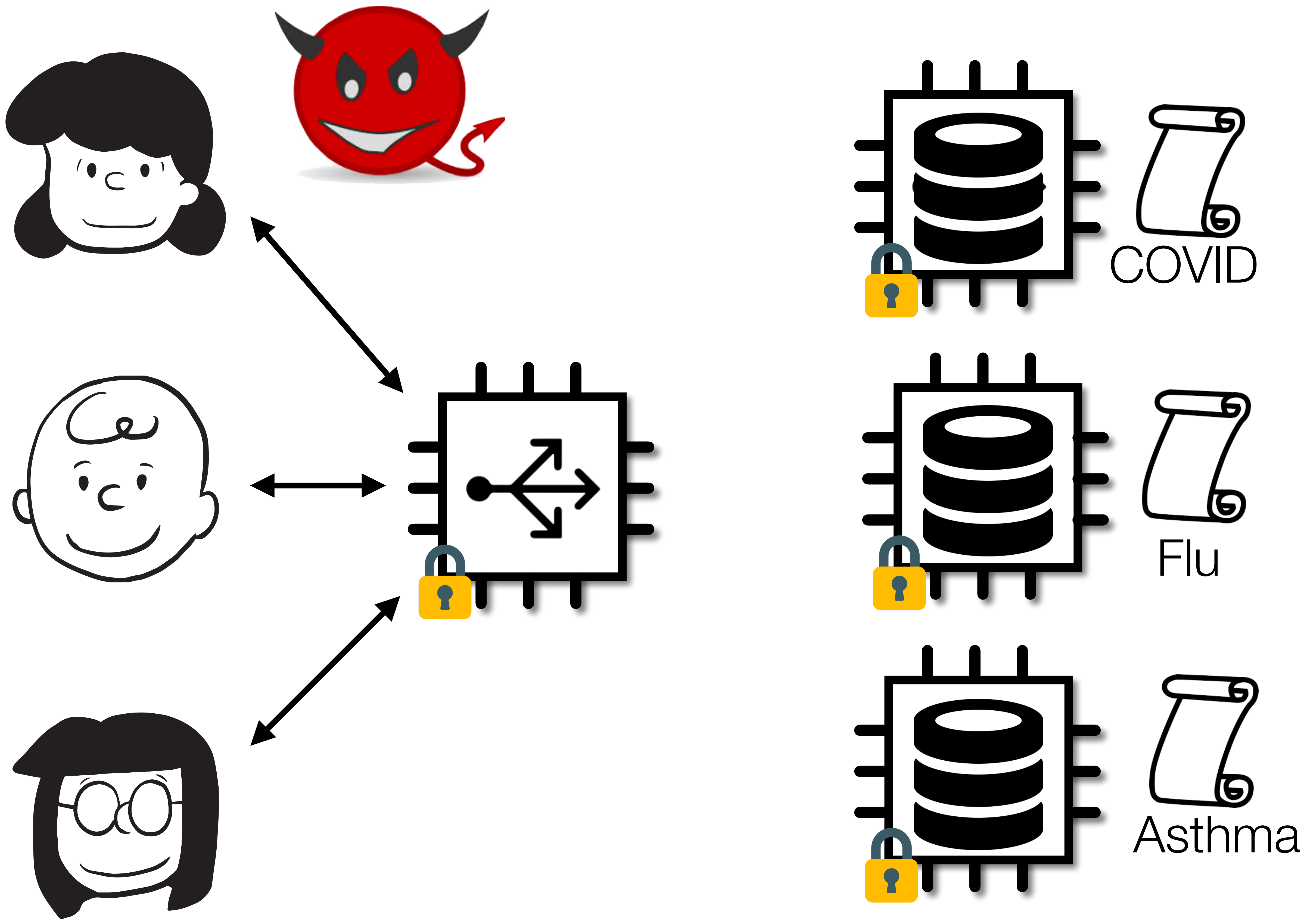
Attacker cannot cause overflow (with high probability)

Attacker's goal: Overflow request batch

Snoopy's defenses:

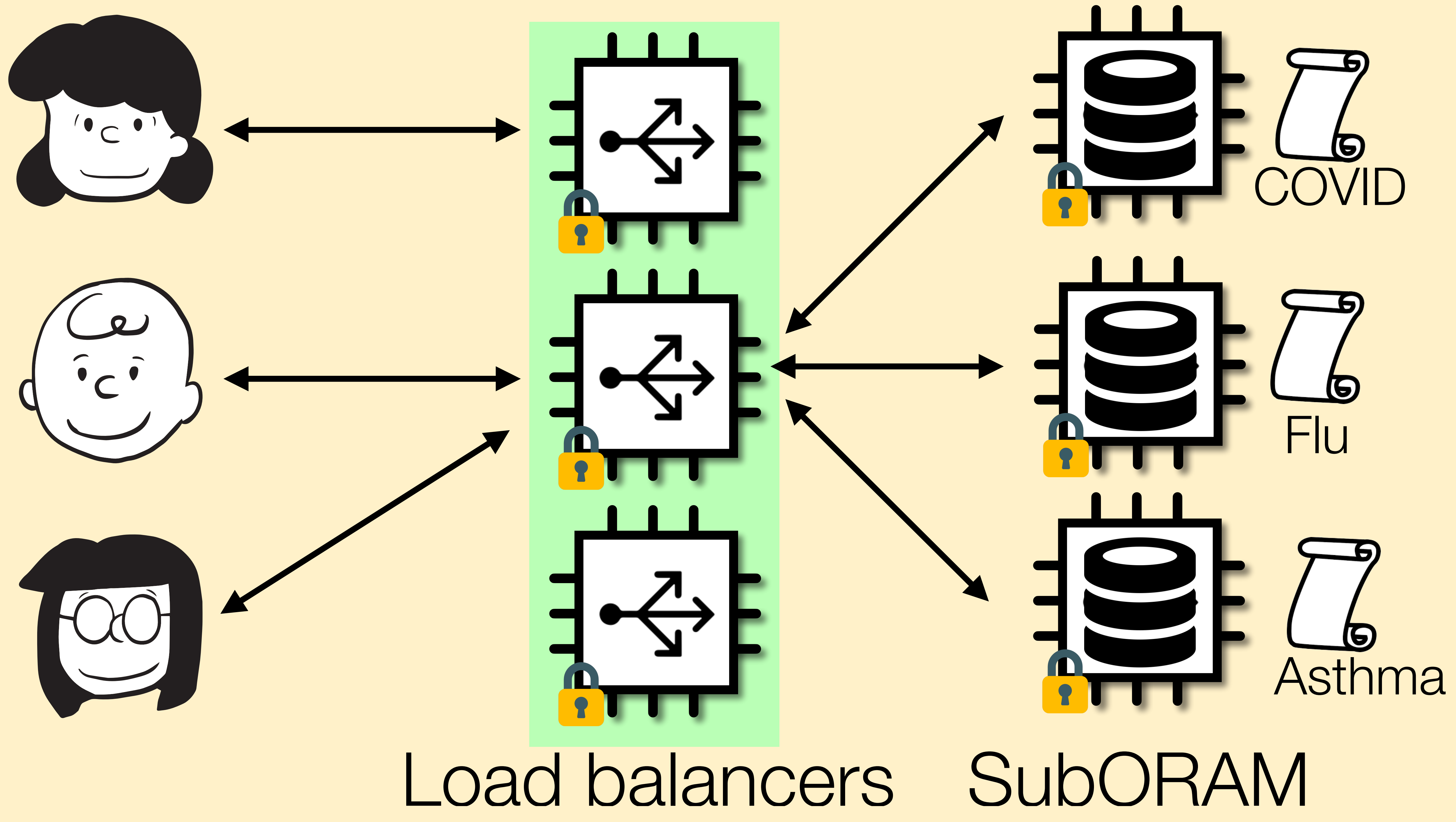
- Deduplication (identical requests $\not\Rightarrow$ overflow)
- Hidden mapping of requests to subORAMs (keyed hash)
- Oblivious request routing

By balls-into-bins analysis, attacker cannot overflow with high probability.

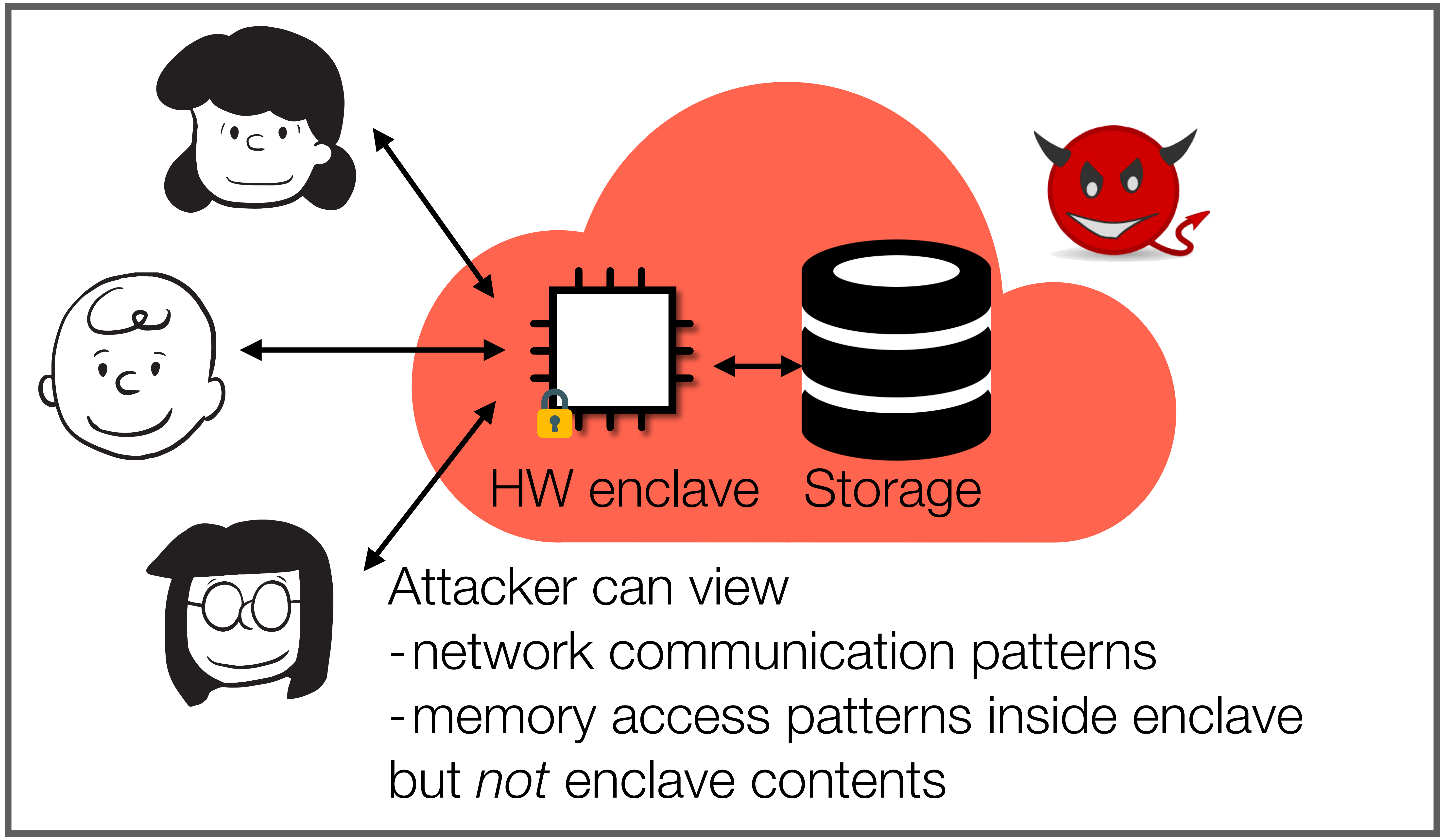


Outline

1. ~~Design idea~~
2. Load balancer
   A. ~~Batch structure~~
   B. Oblivious algorithms
3. SubORAM
4. Evaluation



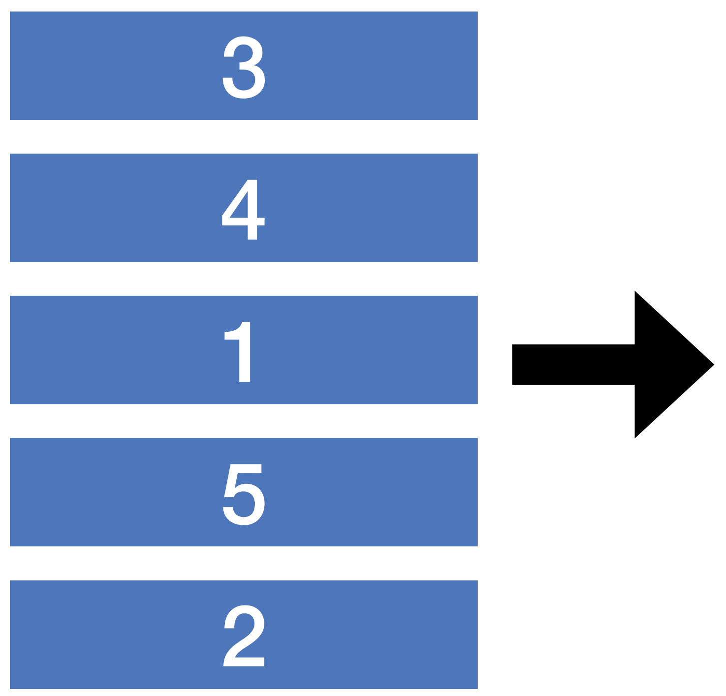
Designing oblivious algorithms



Memory access patterns should not leak information about requests.

Oblivious building blocks

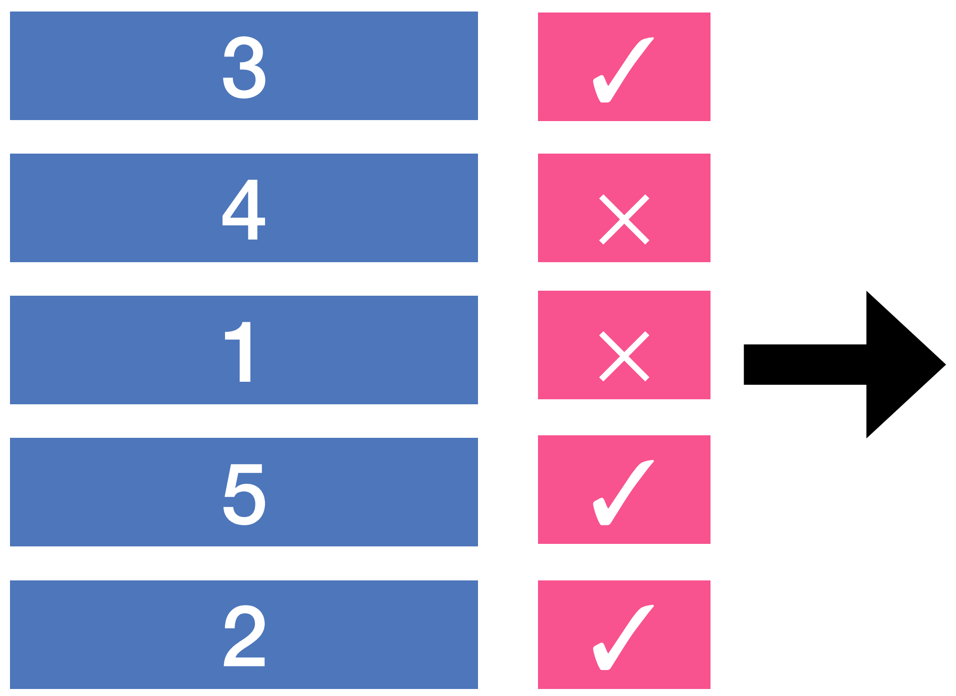
Perform compare-and-swaps in fixed, predefined order



Oblivious sort

$O(n \log^2 n)$

[Batcher68]

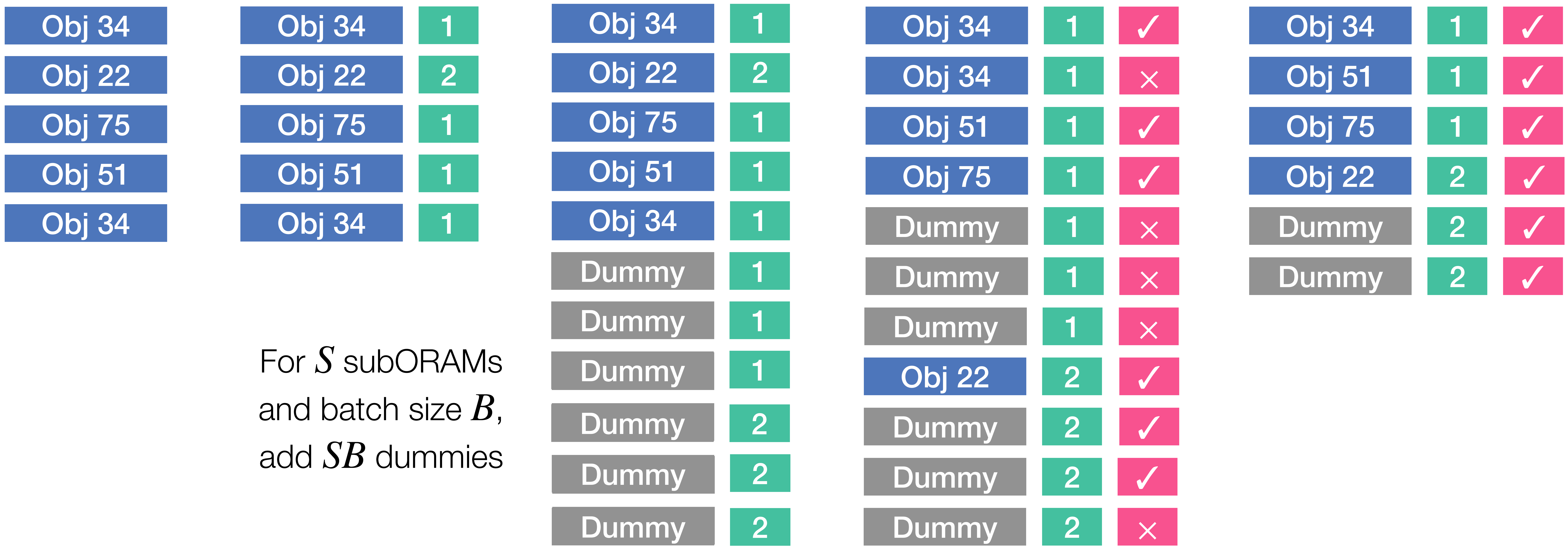


Oblivious compaction

$O(n \log n)$

[Goodrich11]

Constructing batches obliviously



For S subORAMs
and batch size B,
add SB dummies

1. Assign requests to subORAMs

2. Add dummy requests

3. OSort to construct batches with extra dummies

4. OCompact out extra dummies.

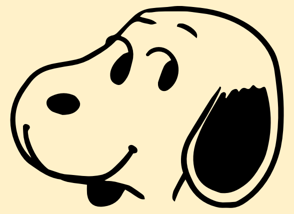
Matching subORAM responses to client requests

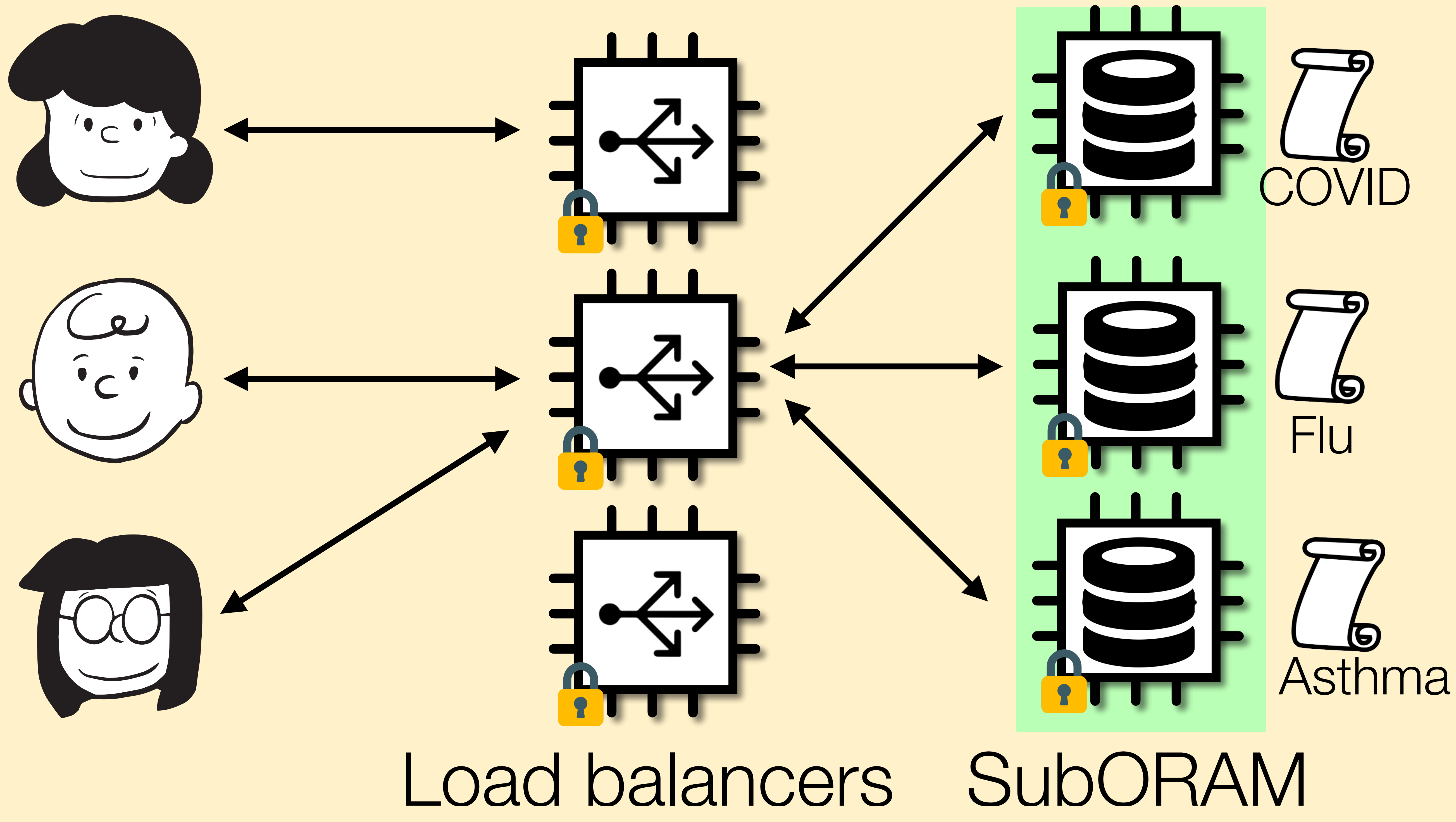
Same key ideas from constructing batches (see paper for details)

Need to:

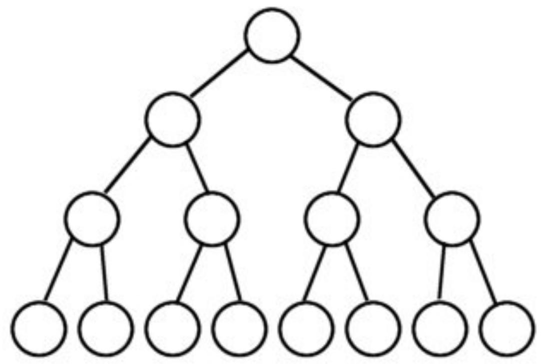
- Filter out dummies
- Propagate subORAM responses to potentially multiple client requests

Outline

1. ~~Design idea~~
2. ~~Load balancer~~
   A. ~~Batch structure~~
   B. ~~Oblivious algorithms~~
3. SubORAM 
4. Evaluation



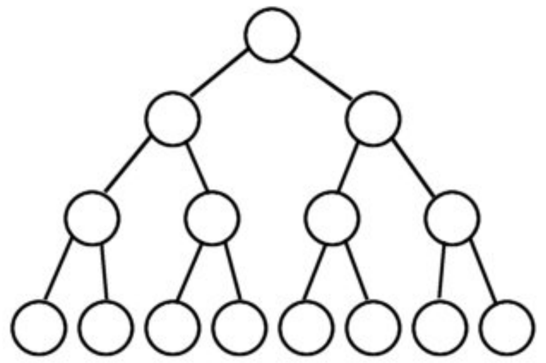
Designing the SubORAM



ORAMs traditionally prioritize latency/communication for individual requests in the client-server model.

- Trees or hierarchical structures support logarithmic access times.
- Making client algorithms oblivious adds overhead [Oblix, CircuitORAM]

Designing the SubORAM

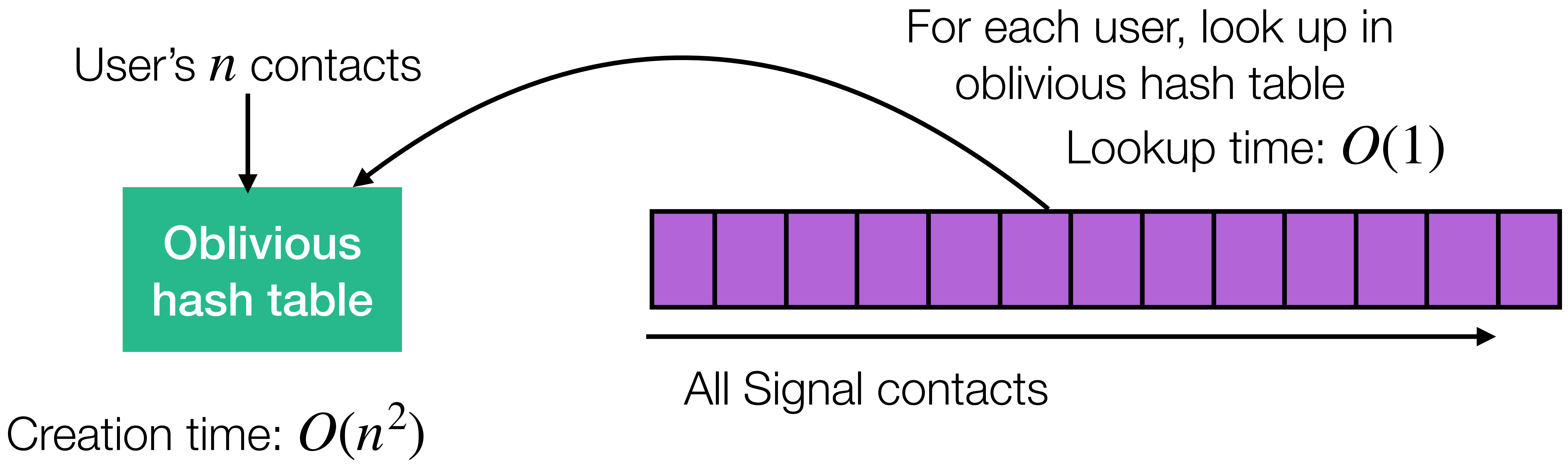


ORAMs traditionally prioritize latency/communication for individual requests in the client-server model.

- Trees or hierarchical structures support logarithmic access times.
- Making client algorithms oblivious adds overhead [Oblix, CircuitORAM]

We instead prioritize throughput for batches of distinct requests in the hardware-enclave setting.

Signal's private contact discovery



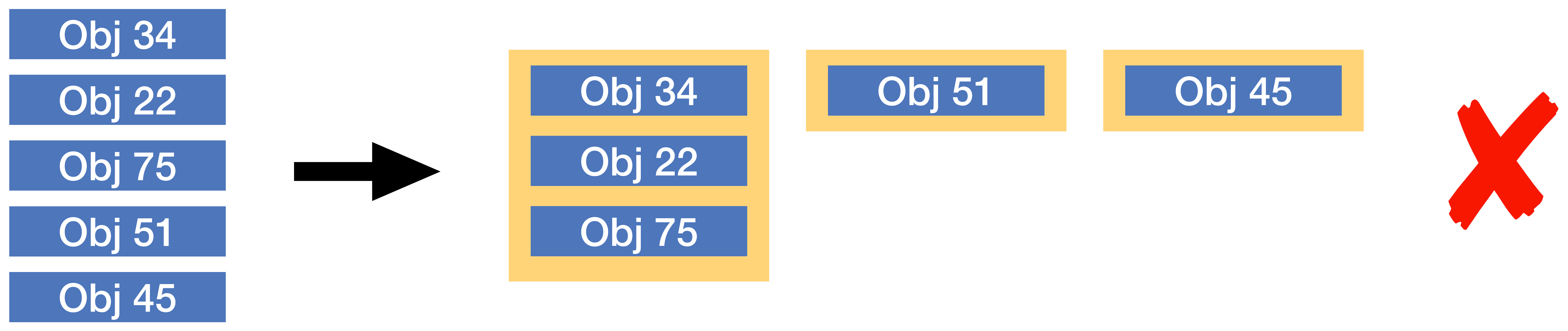
Maps to our setting: contacts = requests, only possible with distinct requests

- Performance: oblivious hash table construction slow for many requests
- Security: Do not size buckets to prevent overflow

Choosing an oblivious hash table

Attempt #1: Fix overflow problem by dynamically sizing hash buckets.

😨 Insecure: Object more likely to be requested if hashed to big bucket.



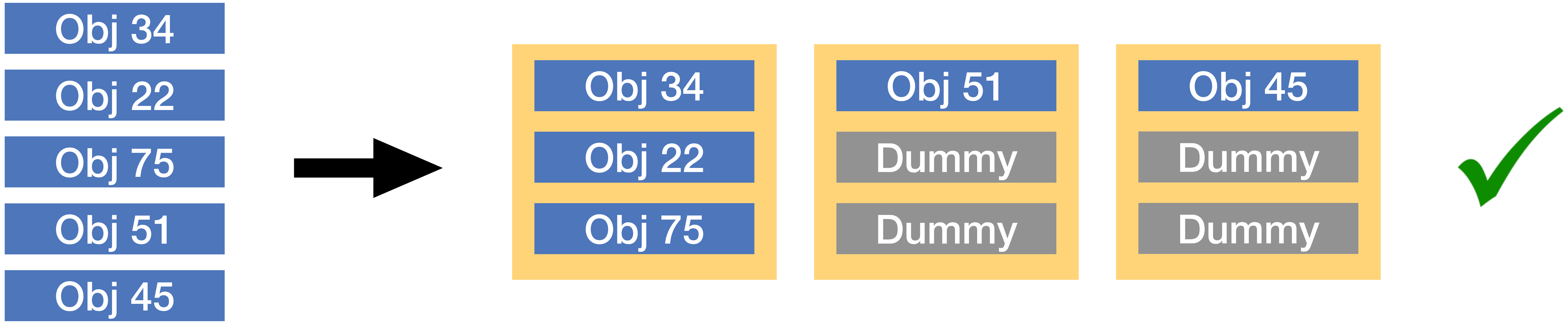
Choosing an oblivious hash table

Attempt #1: Fix overflow problem by dynamically sizing hash buckets.

😨 Insecure: Object more likely to be requested if hashed to big bucket.

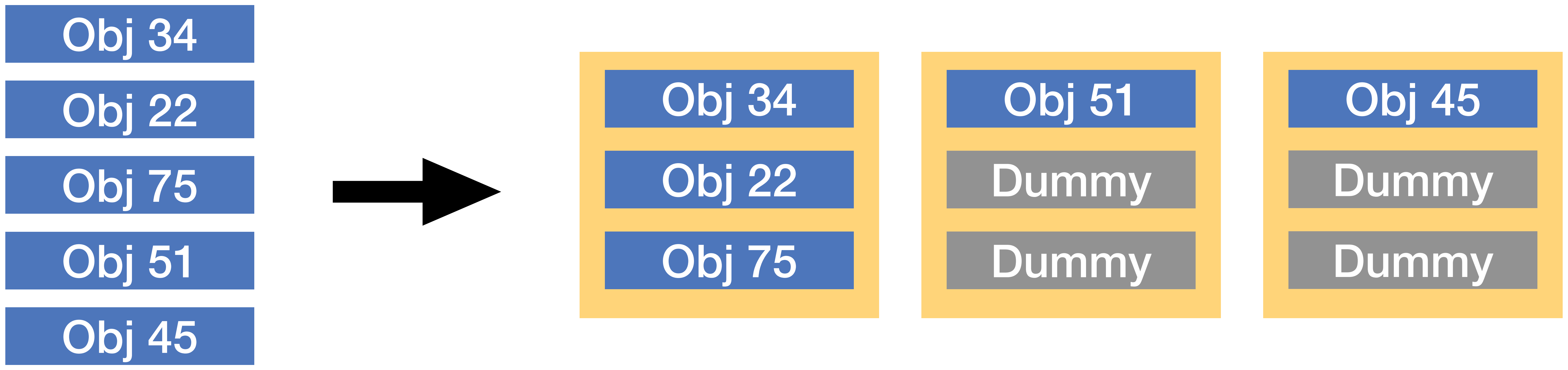
We need a bucket size such that the overflow probability is negligible.

… *wait, didn't we already do this?*



Choosing an oblivious hash table

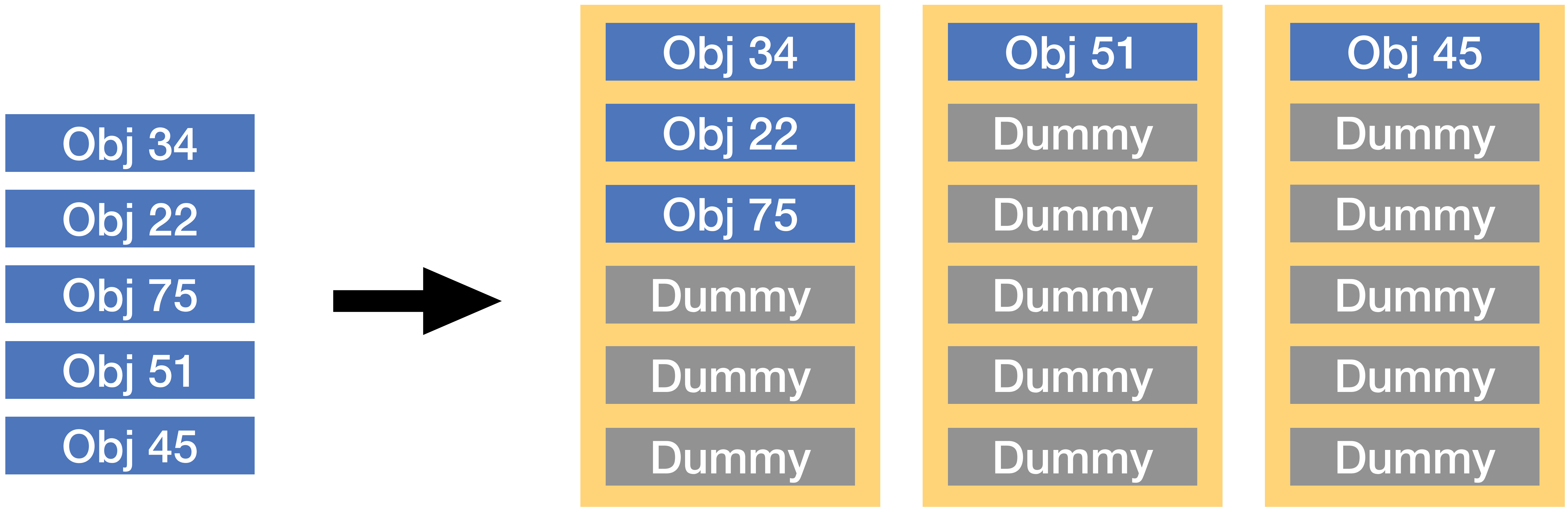
Attempt #2: Set hash bucket size using our bound for the load balancer.



Choosing an oblivious hash table

Attempt #2: Set hash bucket size using our bound for the load balancer.

- Inefficient: Load balancer bound optimized for large batch sizes.
- We want small bucket sizes (an access requires scanning entire bucket).



Choosing an oblivious hash table

💡**Solution:** Oblivious two-tier hash table [CGLS17]

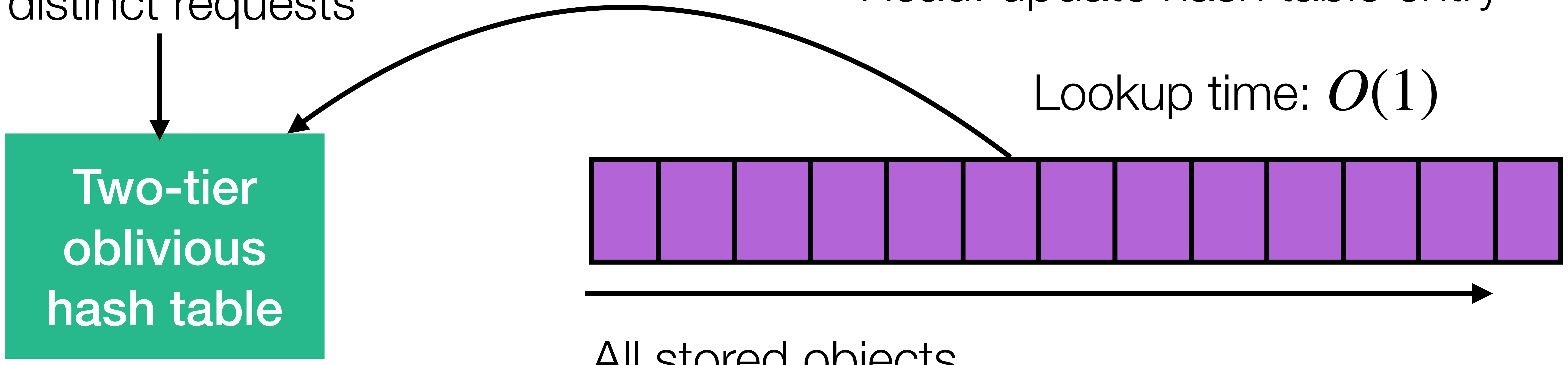
- Overflow requests placed into second hash table → smaller buckets!

Our subORAM design

For stored object, look up in oblivious hash table
- Write: update stored object
- Read: update hash table entry

n distinct requests

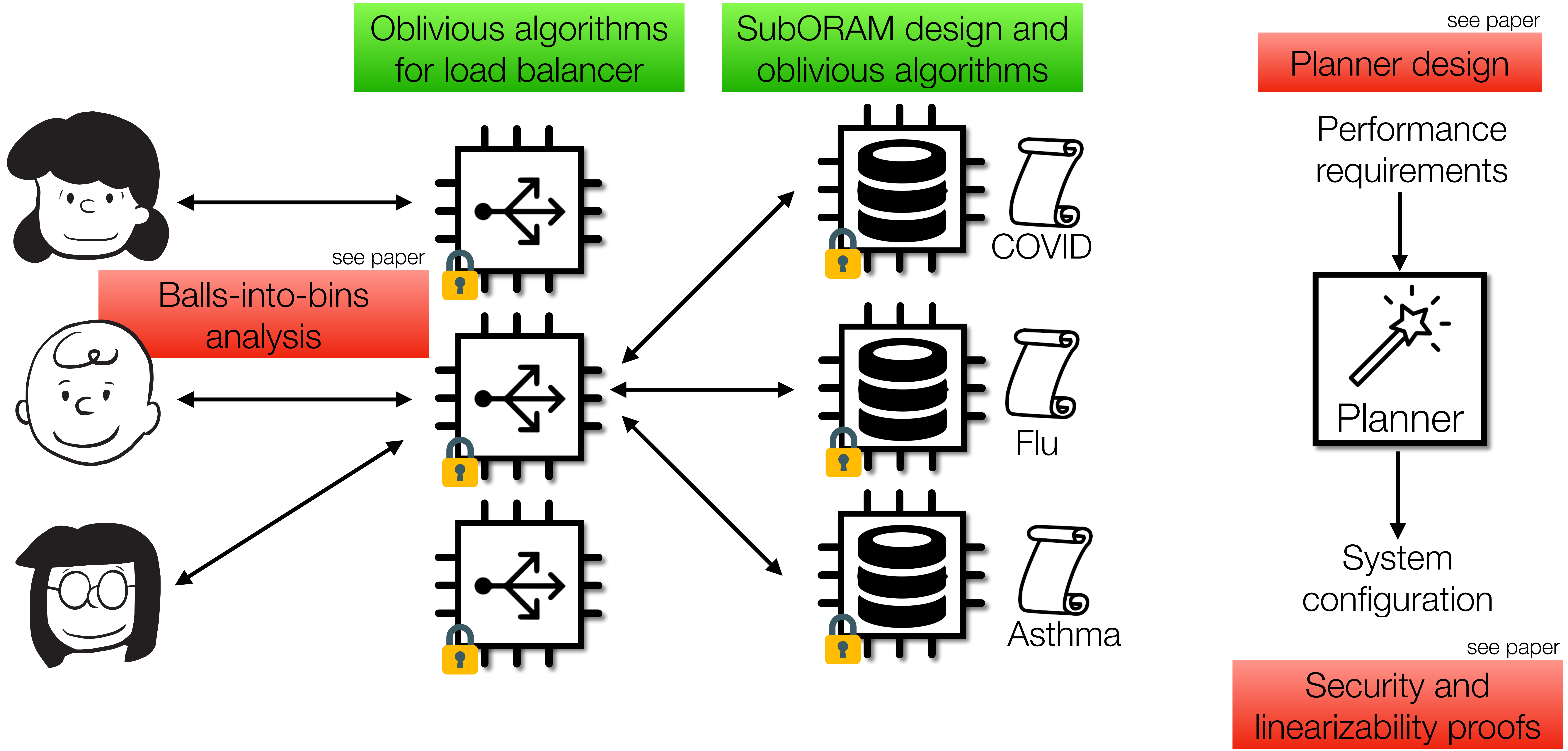
Lookup time: $O(1)$



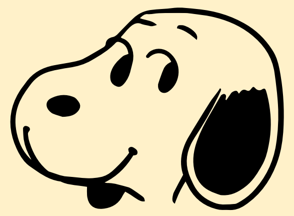
All stored objects

Creation time: $O(n \log^2 n)$

Contributions



Outline

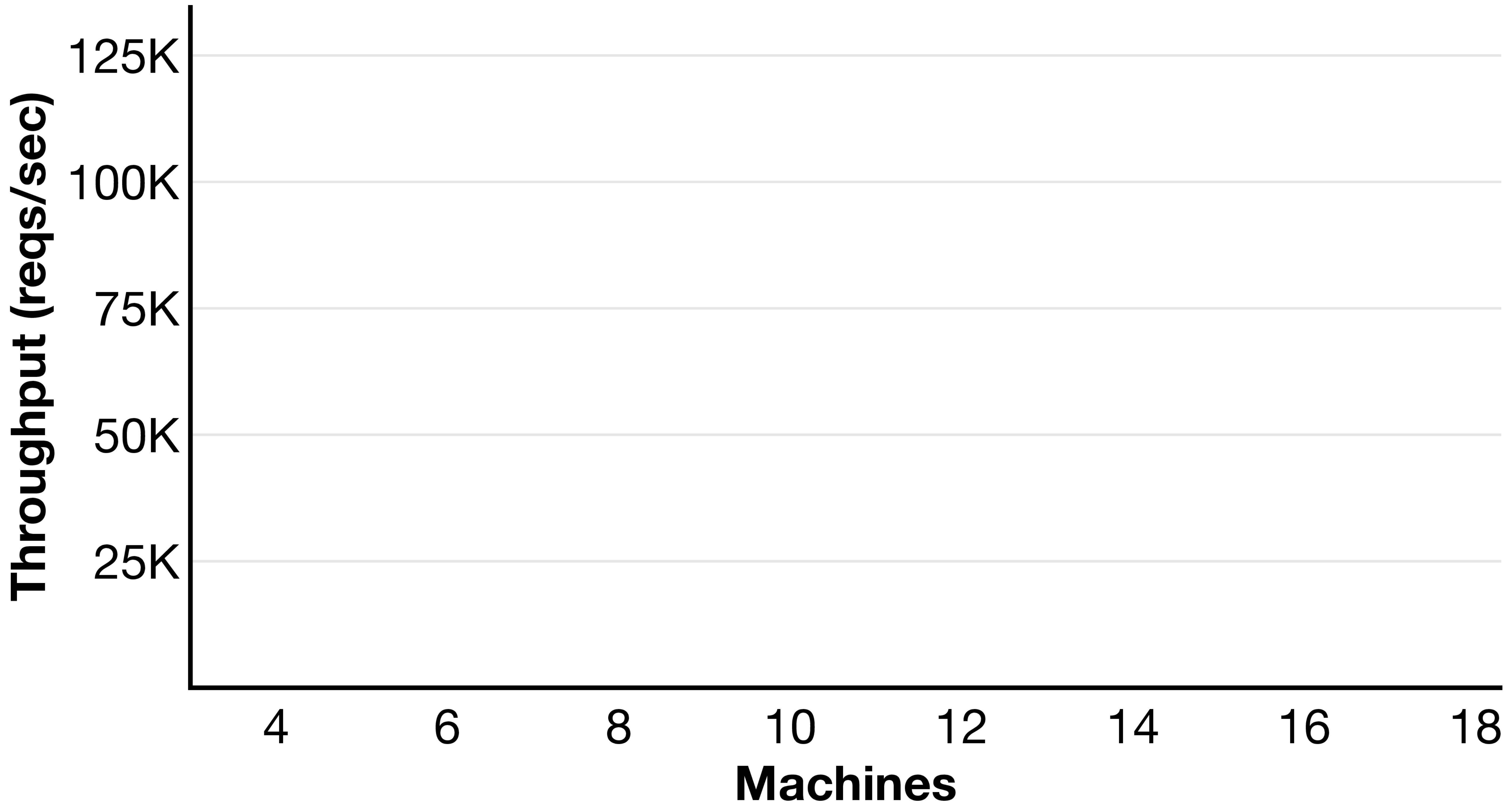
1. ~~Design idea~~
2. ~~Load balancer~~
   A. ~~Batch structure~~
   B. ~~Oblivious algorithms~~
3. ~~SubORAM~~
4. Evaluation 

Evaluation

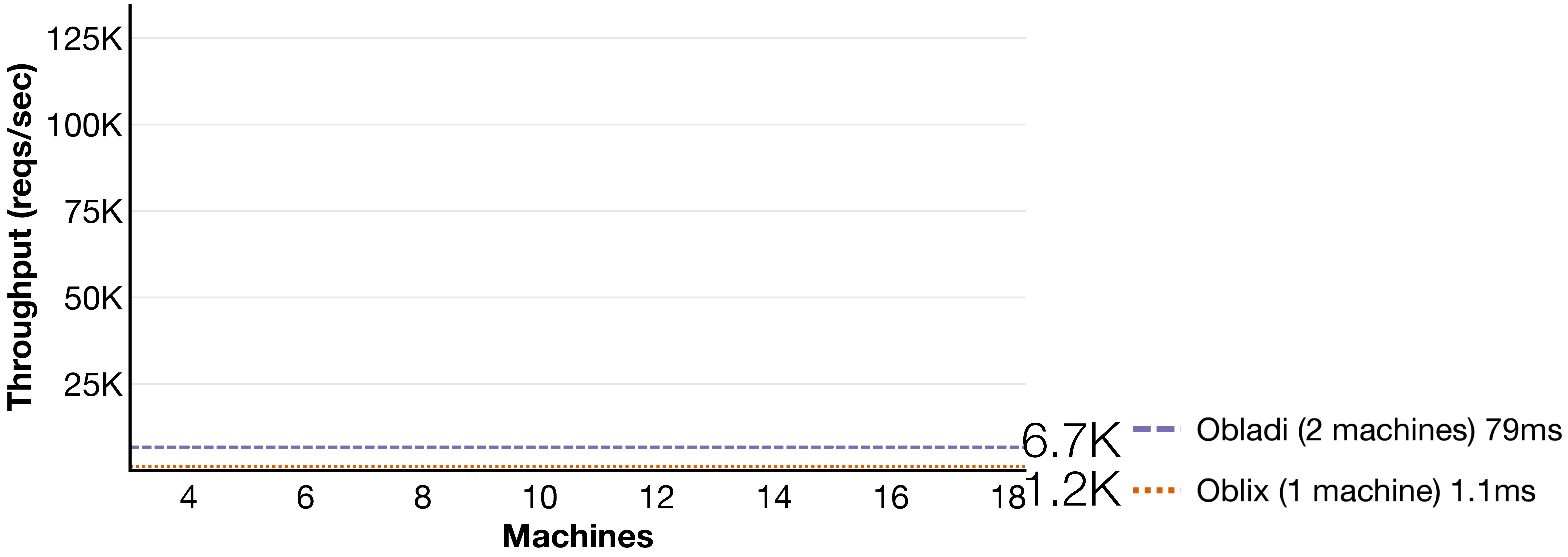
-18 Azure DCsv2 machines
  4-core Intel Xeon CPUs with Intel SGX
-2M objects, 160B object size

Compare to:
-Obladi: ORAM with trusted proxy, optimized for throughput (batch size 500)
-Oblix: ORAM for hardware enclaves

Evaluation

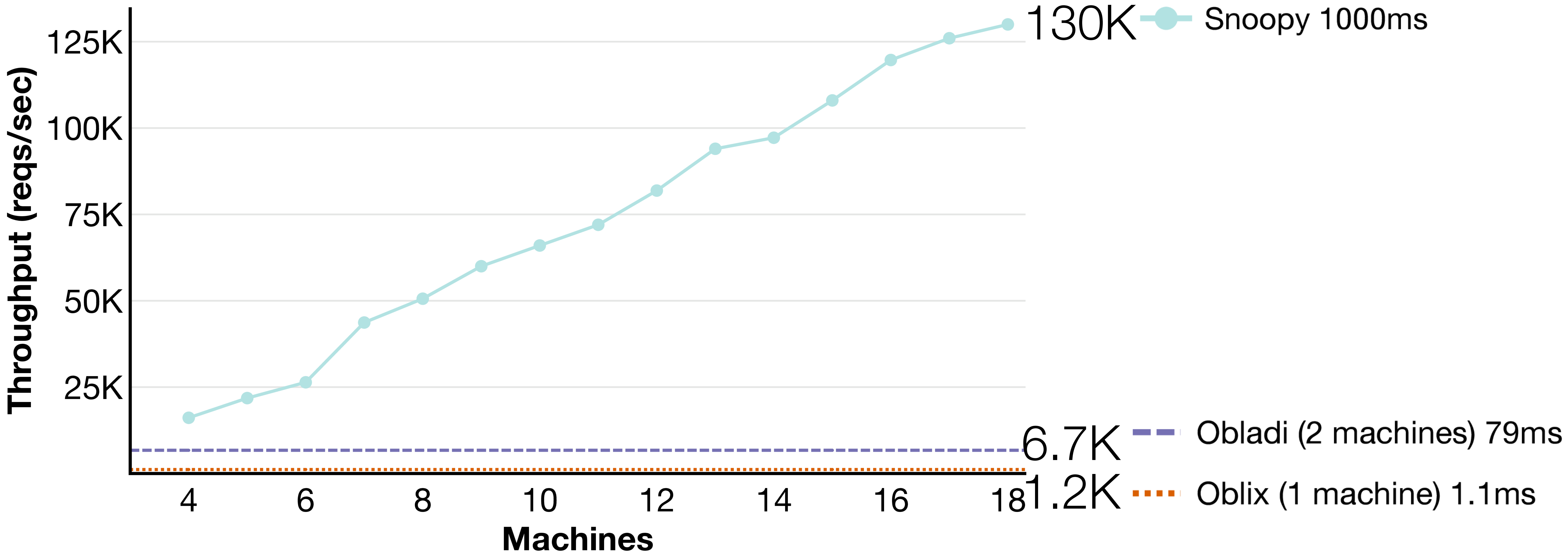

Throughput (reqs/sec)
125K
100K
75K
50K
25K
4
6
8
10
12
14
16
18
Machines

Evaluation



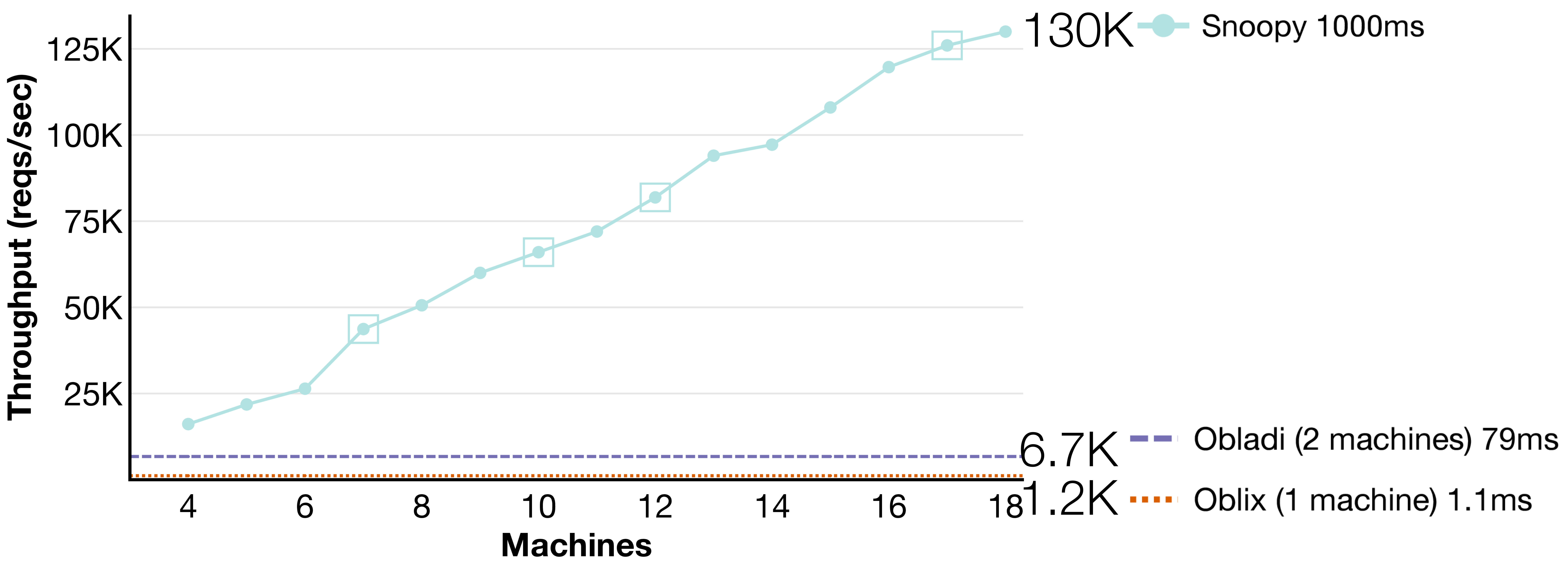
Throughput (reqs/sec)
125K
100K
75K
50K
25K
4
6
8
10
12
14
16
18
Machines
6.7K
1.2K
Obladi (2 machines) 79ms
Oblix (1 machine) 1.1ms

Evaluation



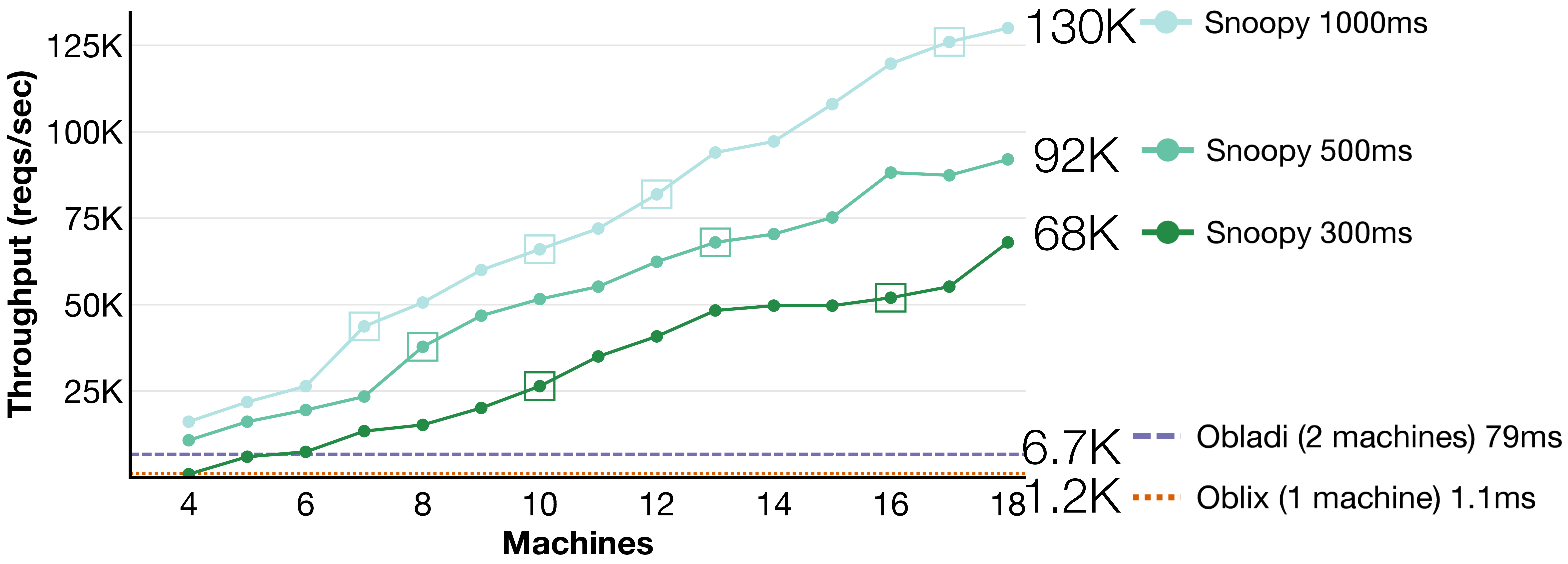
Throughput (reqs/sec)
125K
100K
75K
50K
25K
4
6
8
10
12
14
16
18
Machines
130K
Snoopy 1000ms
6.7K
Obladi (2 machines) 79ms
1.2K
Oblix (1 machine) 1.1ms

Evaluation



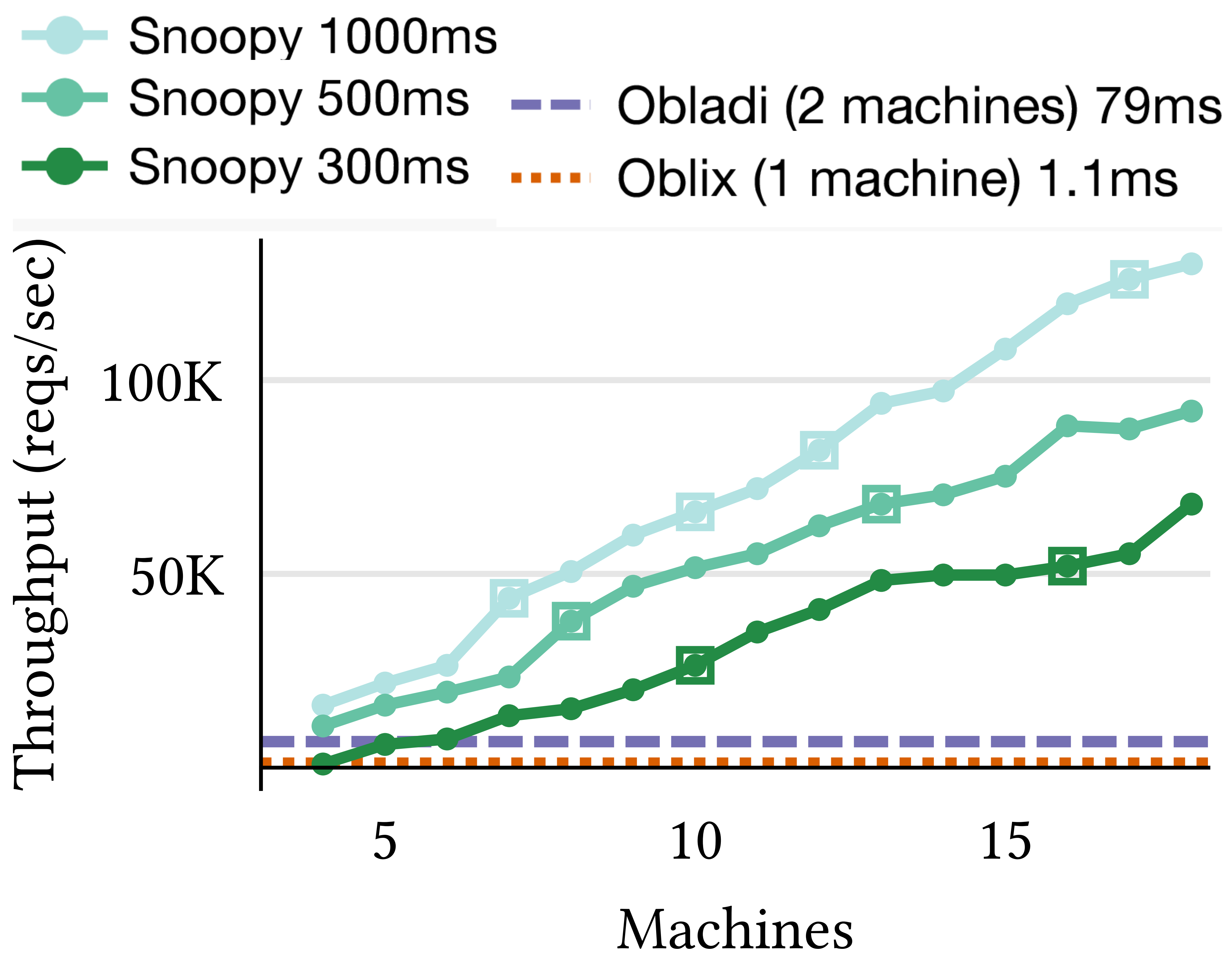
Throughput (reqs/sec)
125K
100K
75K
50K
25K
4
6
8
10
12
14
16
18
Machines
130K
Snoopy 1000ms
6.7K
Obladi (2 machines) 79ms
1.2K
Oblix (1 machine) 1.1ms

Evaluation

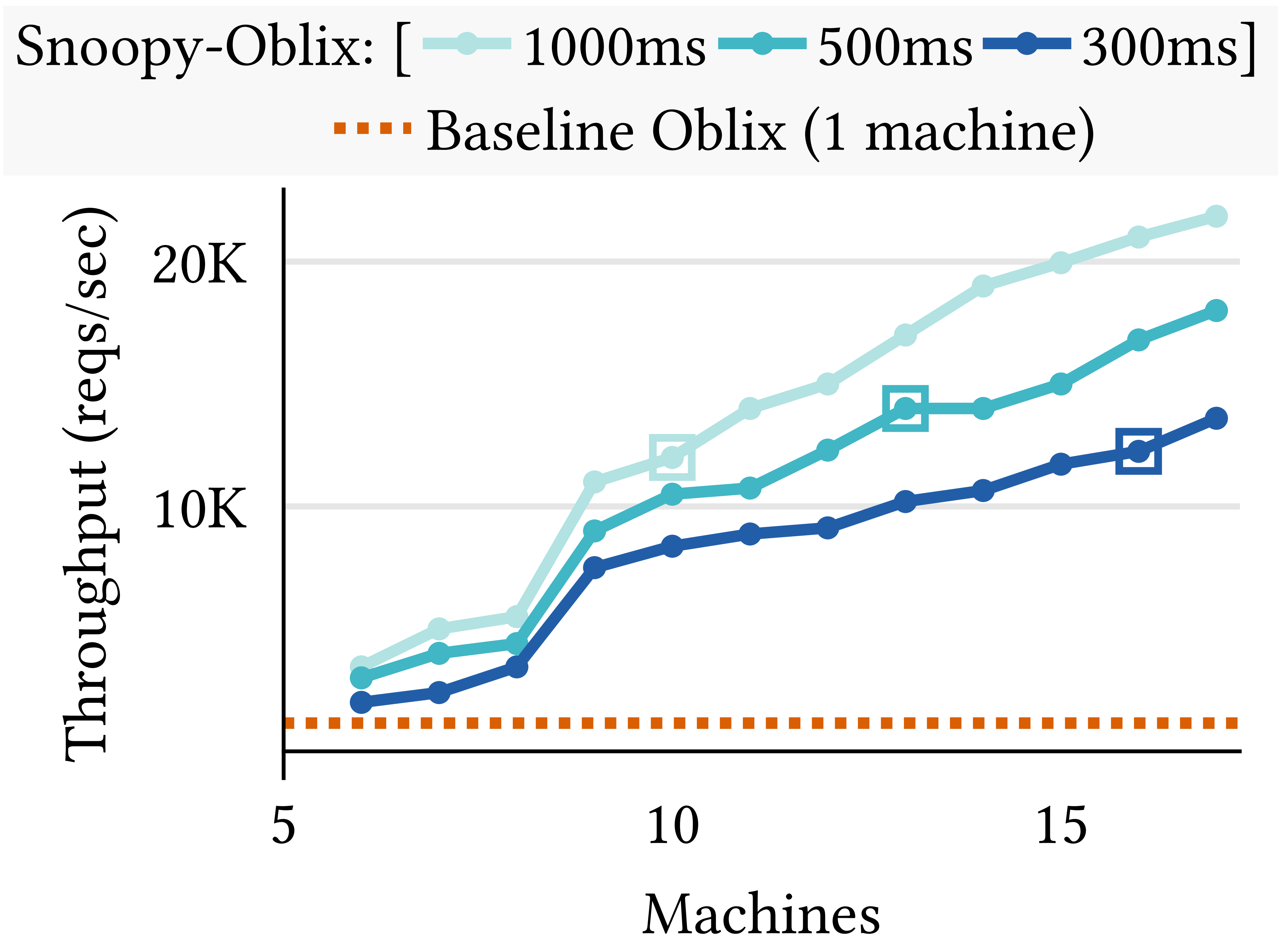


Throughput (reqs/sec)
125K
100K
75K
50K
25K
4
6
8
10
12
14
16
18
Machines
130K
Snoopy 1000ms
92K
Snoopy 500ms
68K
Snoopy 300ms
6.7K
Obladi (2 machines) 79ms
1.2K
Oblix (1 machine) 1.1ms

Evaluation



Snoopy 1000ms
Snoopy 500ms
Snoopy 300ms
Obladi (2 machines) 79ms
Oblix (1 machine) 1.1ms
Throughput (reqs/sec)
100K
50K
5
10
15
Machines



Snoopy-Oblix: [1000ms 500ms 300ms]
Baseline Oblix (1 machine)
Throughput (reqs/sec)
20K
10K
5
10
15
Machines

Conclusion

Snoopy is an oblivious object store that scales like plaintext storage.

Thanks!

Emma Dauterman
edauterman@berkeley.edu

https://eprint.iacr.org/2021/1280.pdf
https://github.com/ucbrise/snoopy

References

[AKSL18] Adil Ahmad, Kyungtae Kim, Muhammad Ihsanulhaq Sarfaraz, and Byoungyoung Lee. OBLIVIATE: A data oblivious filesystem for Intel SGX. In NDSS, 2018.

[CGLS17] T-H Hubert Chan, Yue Guo, Wei-Kai Lin, and Elaine Shi. Oblivious hashing revisited, and applications to asymptotically efficient ORAM and OPRAM. In ASIACRYPT. Springer, IACR, 2017.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In CCS. ACM, 2015.

[CBCH+18]Natacha Crooks, Matthew Burke, Ethan Cecchetti, Sitar Harel, Rachit Agarwal, and Lorenzo Alvisi. Obladi: Oblivious serializable transactions in the cloud. In OSDI. USENIX, 2018.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. Journal of the ACM (JACM), 1996

[GLMP19] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In Security & Privacy. IEEE, 2019.

[IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In NDSS. Citeseer, 2012.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'neill. Generic attacks on secure outsourced databases. In CCS. ACM, 2016.

[KPT19] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Data recovery on encrypted databases with k-nearest neighbor query leakage. In Security & Privacy). IEEE, 2019.

[LPMR+13] Jacob R Lorch, Bryan Parno, James Mickens, Mariana Raykova, and Joshua Schiffman. Shroud: Ensuring private access to large-scale data in the data center. In FAST, 2013.

[MPCC+18] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In Security & Privacy. IEEE, 2018.

[RFKS+15] Ling Ren, Christopher Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten Van Dijk, and Srinivas Devadas. Constants count: Practical improvements to oblivious RAM. In USENIX Security Symposium, 2015.

[SZAL+16] Cetin Sahin, Victor Zakhary, Amr El Abbadi, Huijia Lin, and Stefano Tessaro. TaoStore: Overcoming asynchronicity in oblivious data storage. In Security & Privacy. IEEE, 2016.

[SGF18] Sajin Sasy, Sergey Gorbunov, and Christopher W Fletcher. ZeroTrace: Oblivious memory primitives from Intel SGX. In NDSS, 2018.

[SS13] Emil Stefanov and Elaine Shi. ObliviStore: High performance oblivious cloud storage. In Security & Privacy. IEEE, 2013.

[SSS12] Emil Stefanov, Elaine Shi, and Dawn Song. Towards practical oblivious ram. In NDSS, 2012.

[SVSF+13] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In CCS. ACM, 2013.

[WST12] Peter Williams, Radu Sion, and Alin Tomescu. Privatefs: A parallel oblivious file system. In CCS. ACM, 2012.