Pancake: Frequency Smoothing for Encrypted Data Stores

Paul Grubbs[*,2], **Anurag Khandelwal**[*,1], Marie-Sarah Lacharité[*,2], Lloyd Brown[4], Lucy Li[2], Rachit Agrawal[3], Thomas Ristenpart[2]

1 Yale
2 Cornell Tech
3 Cornell University
4 Berkeley, University of California

*Equal contribution authors

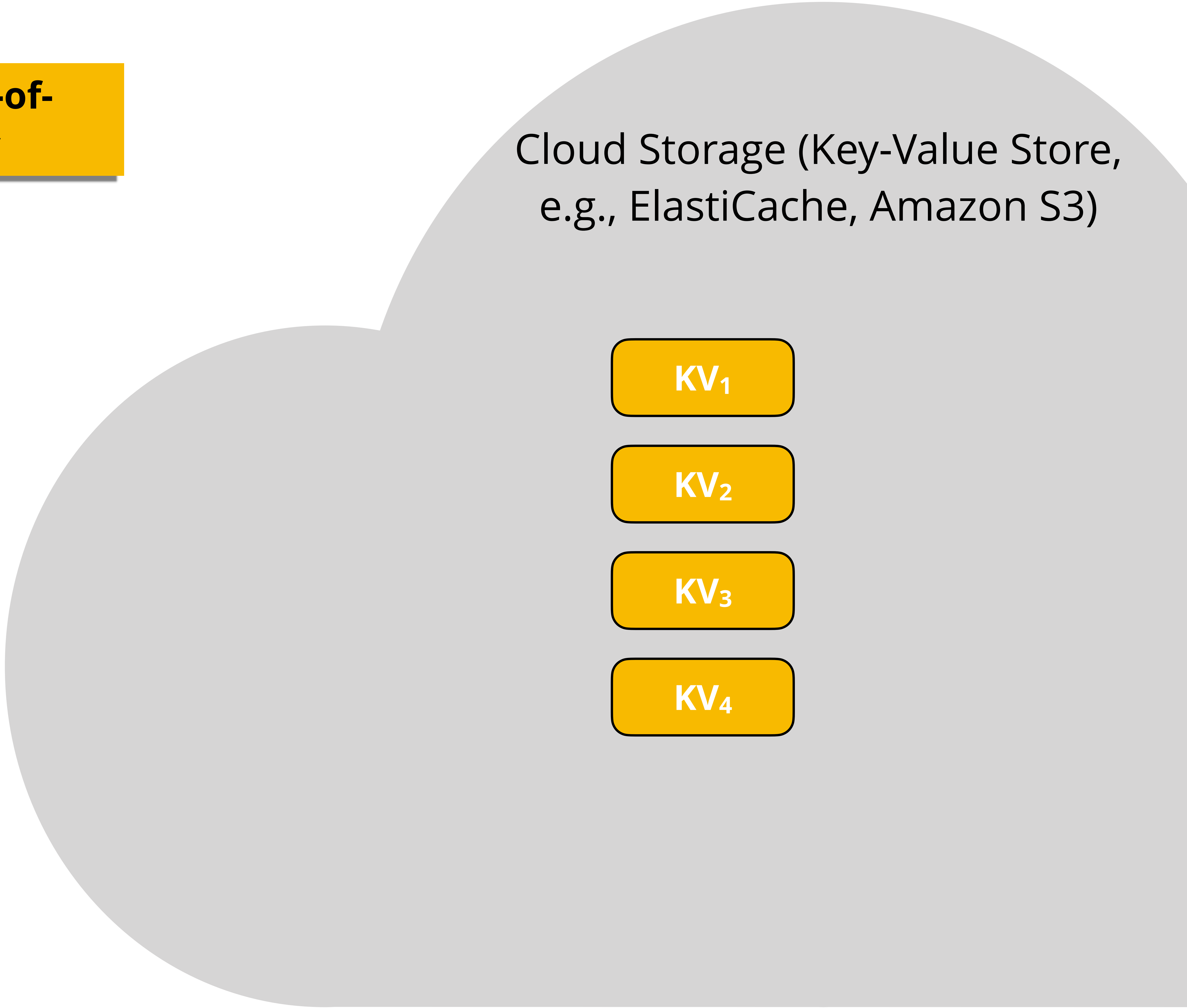
Cloud Data Stores

Cloud Data Stores

Transition to cloud hosted data stores for **ease-of-management**, **scalability** & **cost-efficiency**

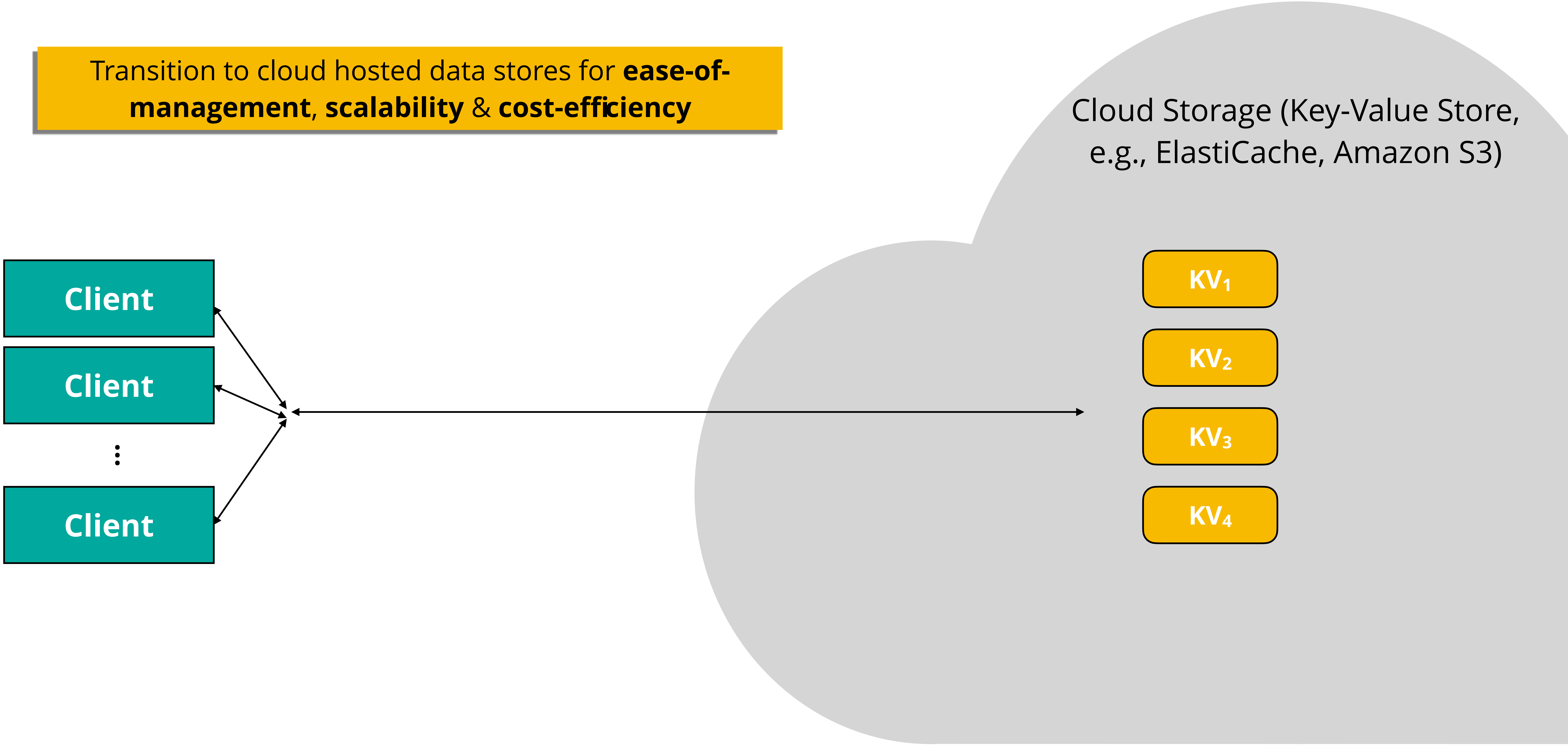
Cloud Data Stores

Transition to cloud hosted data stores for **ease-of-management**, **scalability** & **cost-efficiency**



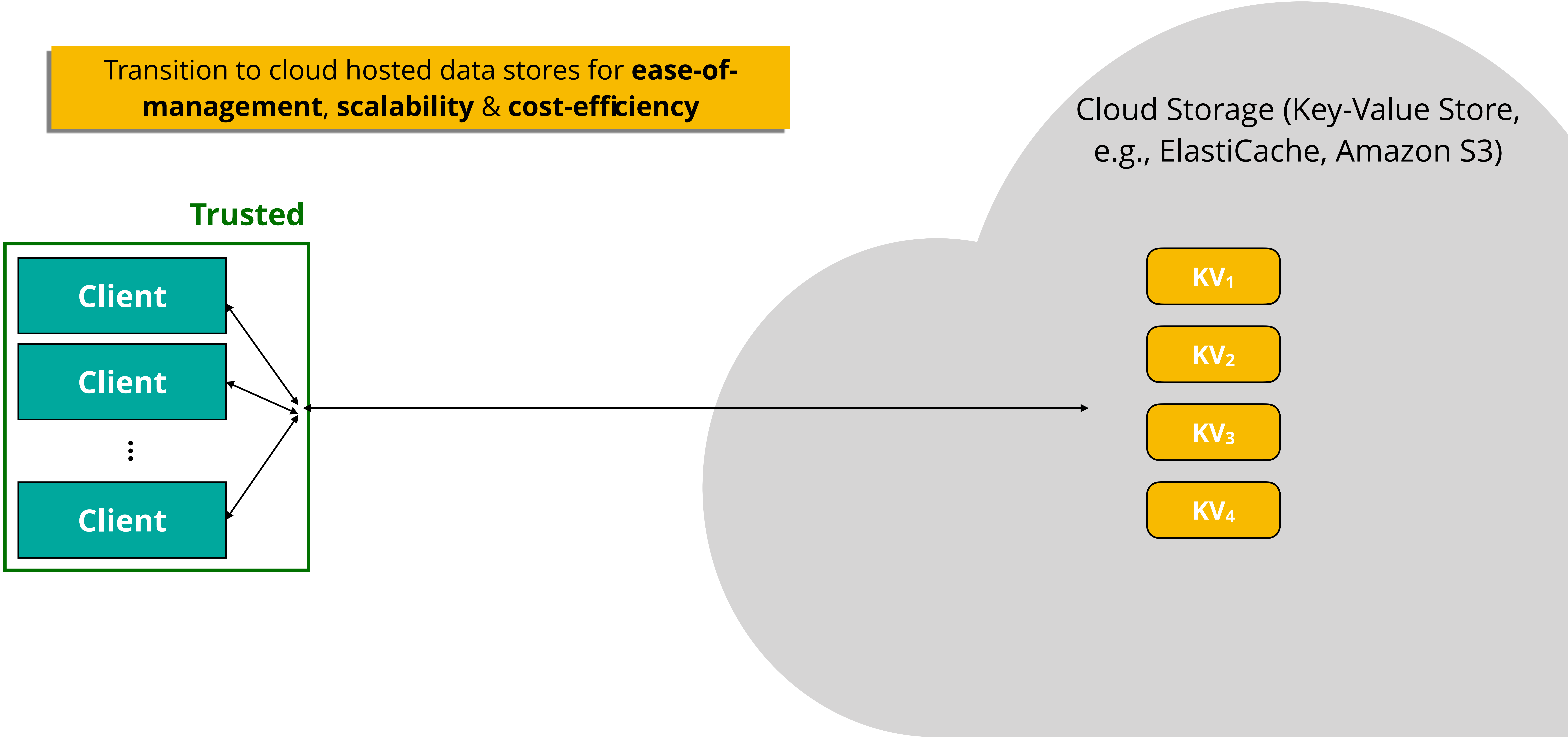
Cloud Data Stores

Transition to cloud hosted data stores for **ease-of-management**, **scalability** & **cost-efficiency**



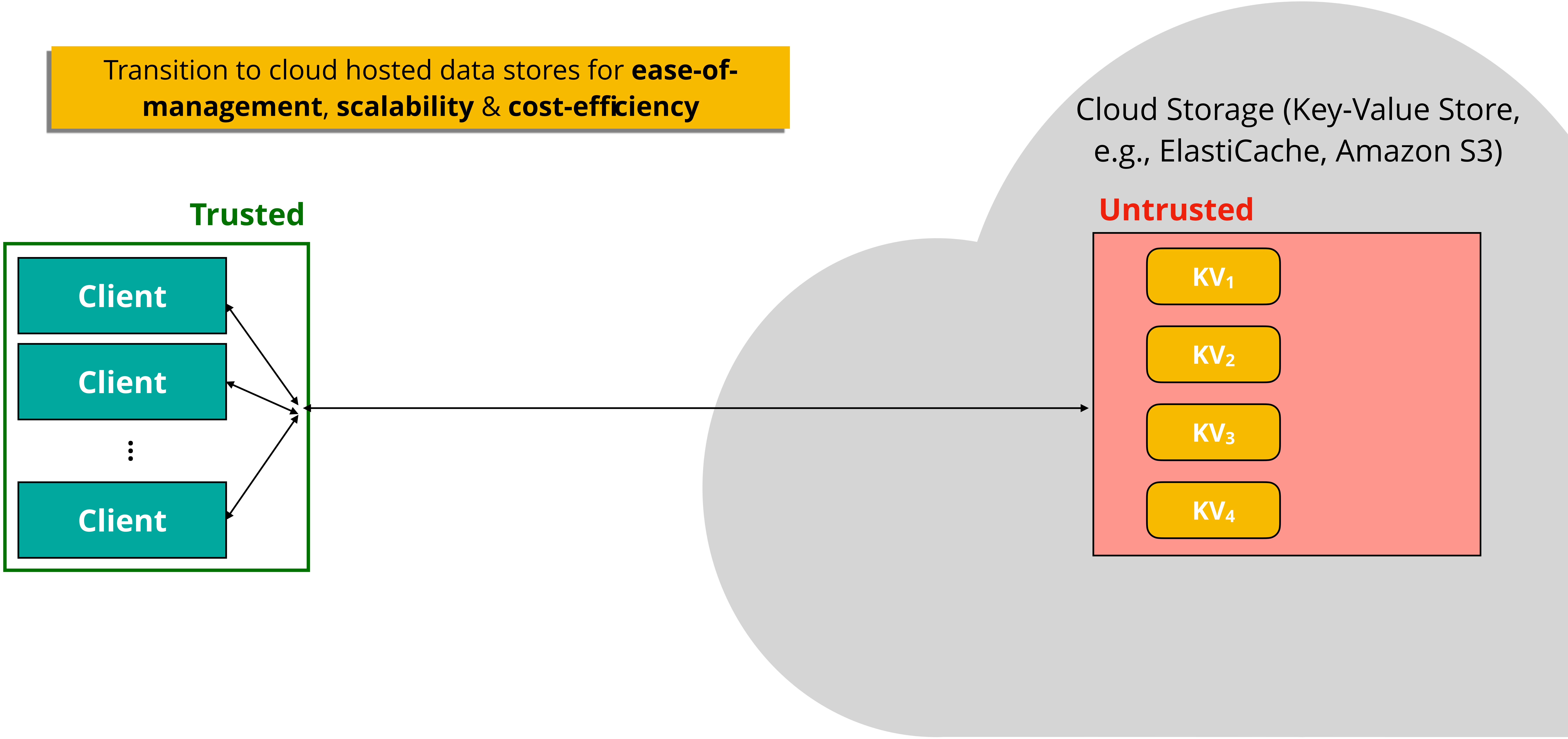
Cloud Data Stores

Transition to cloud hosted data stores for **ease-of-management**, **scalability** & **cost-efficiency**



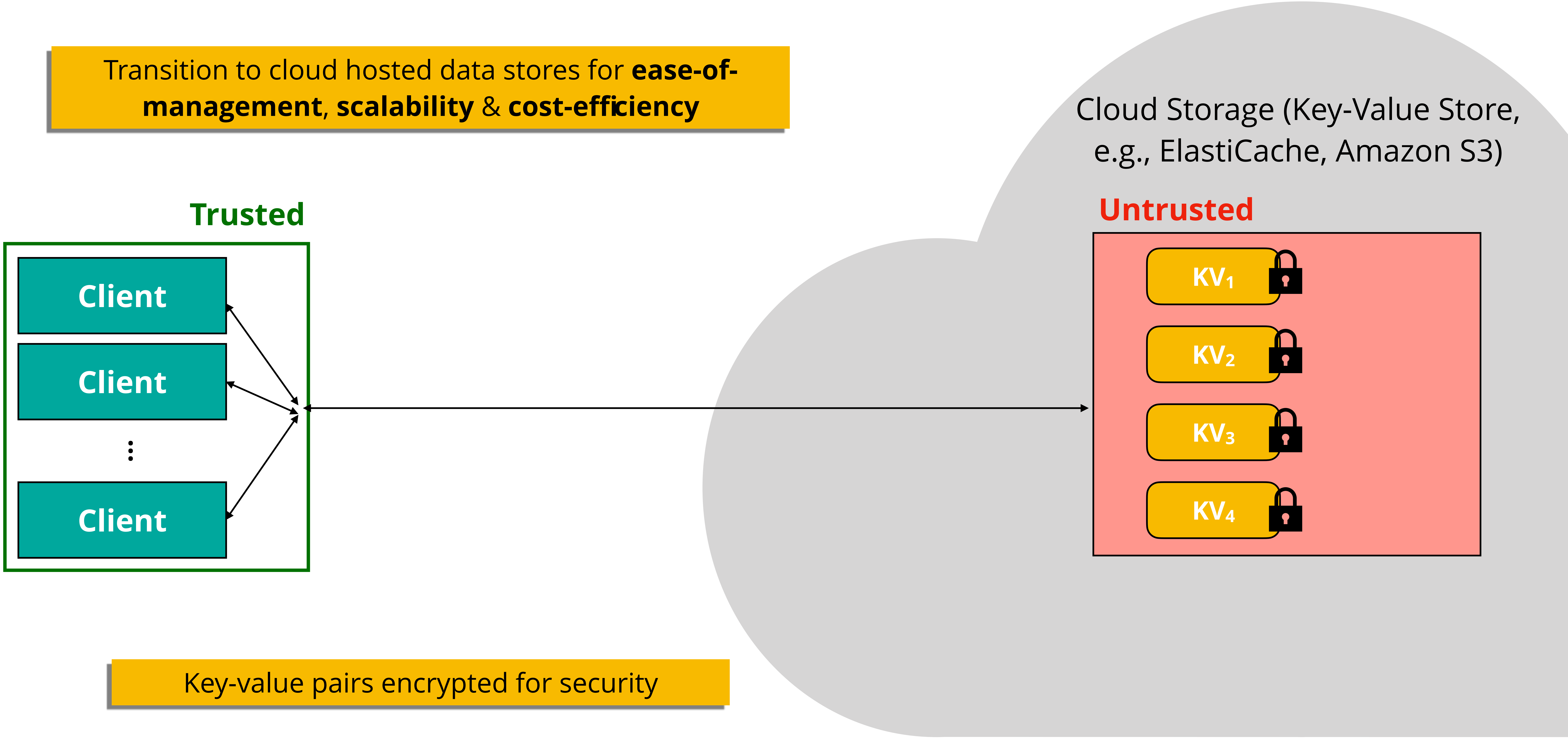
Cloud Data Stores

Transition to cloud hosted data stores for **ease-of-management**, **scalability** & **cost-efficiency**



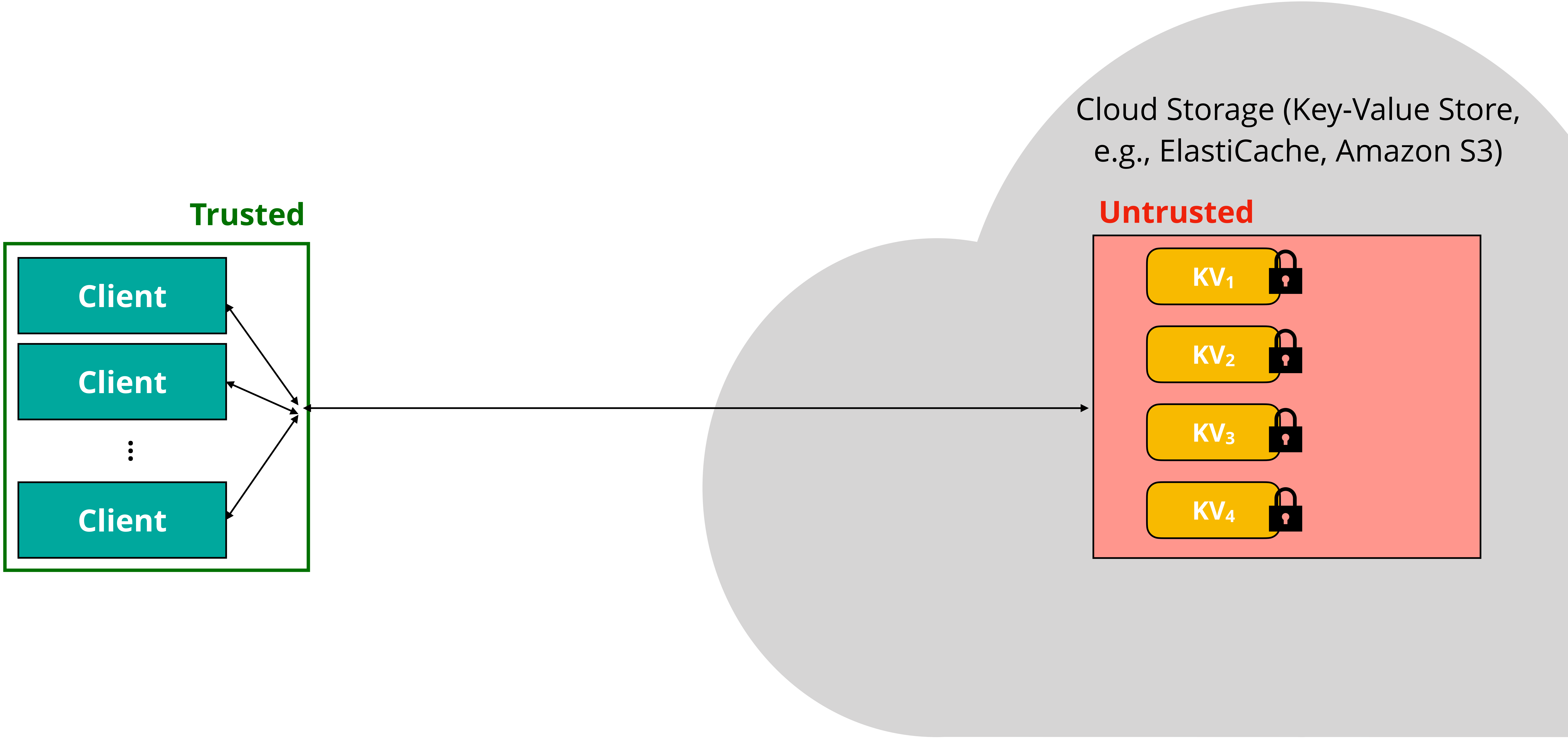
Cloud Data Stores

Transition to cloud hosted data stores for **ease-of-management**, **scalability** & **cost-efficiency**

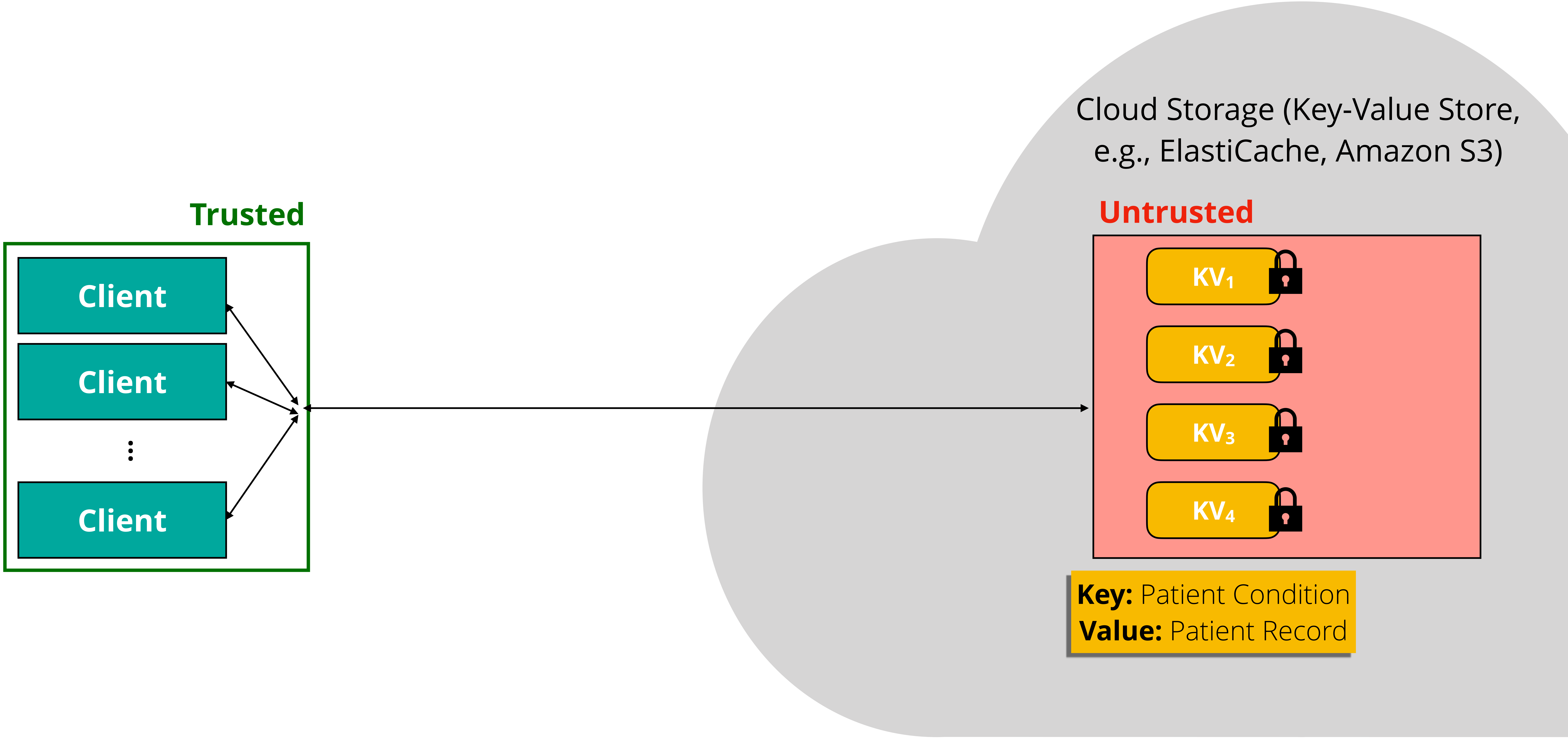


Key-value pairs encrypted for security

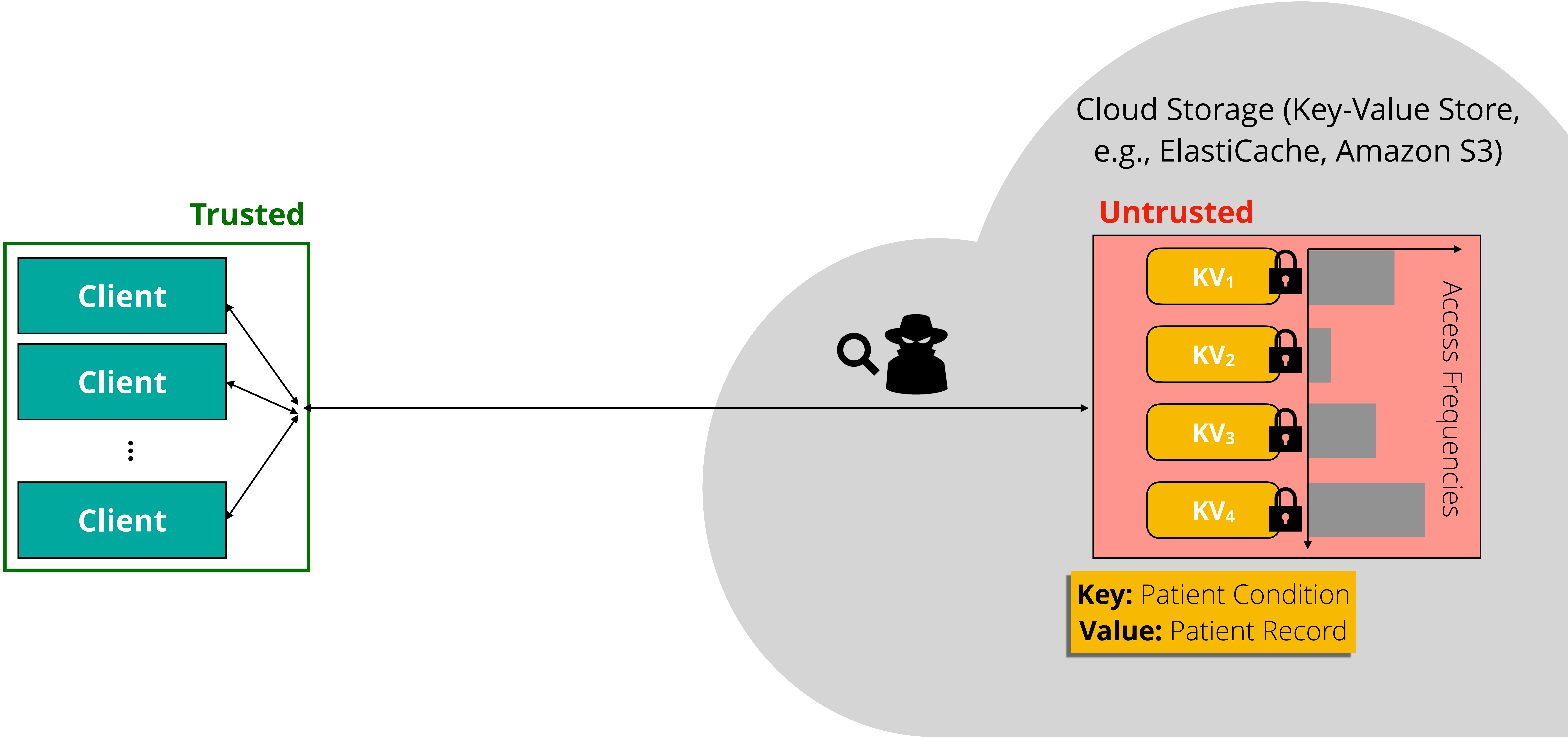
Access Pattern Attacks



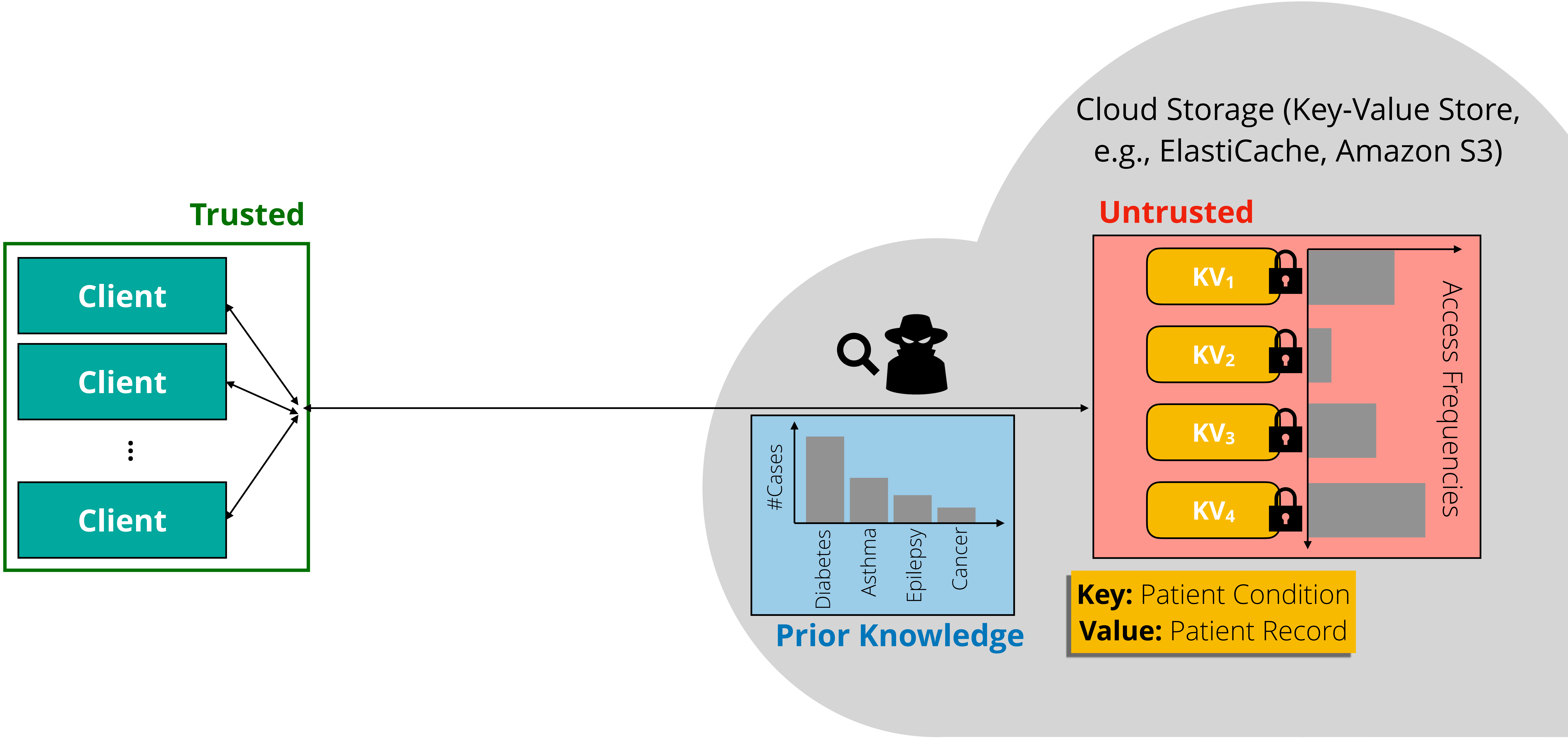
Access Pattern Attacks



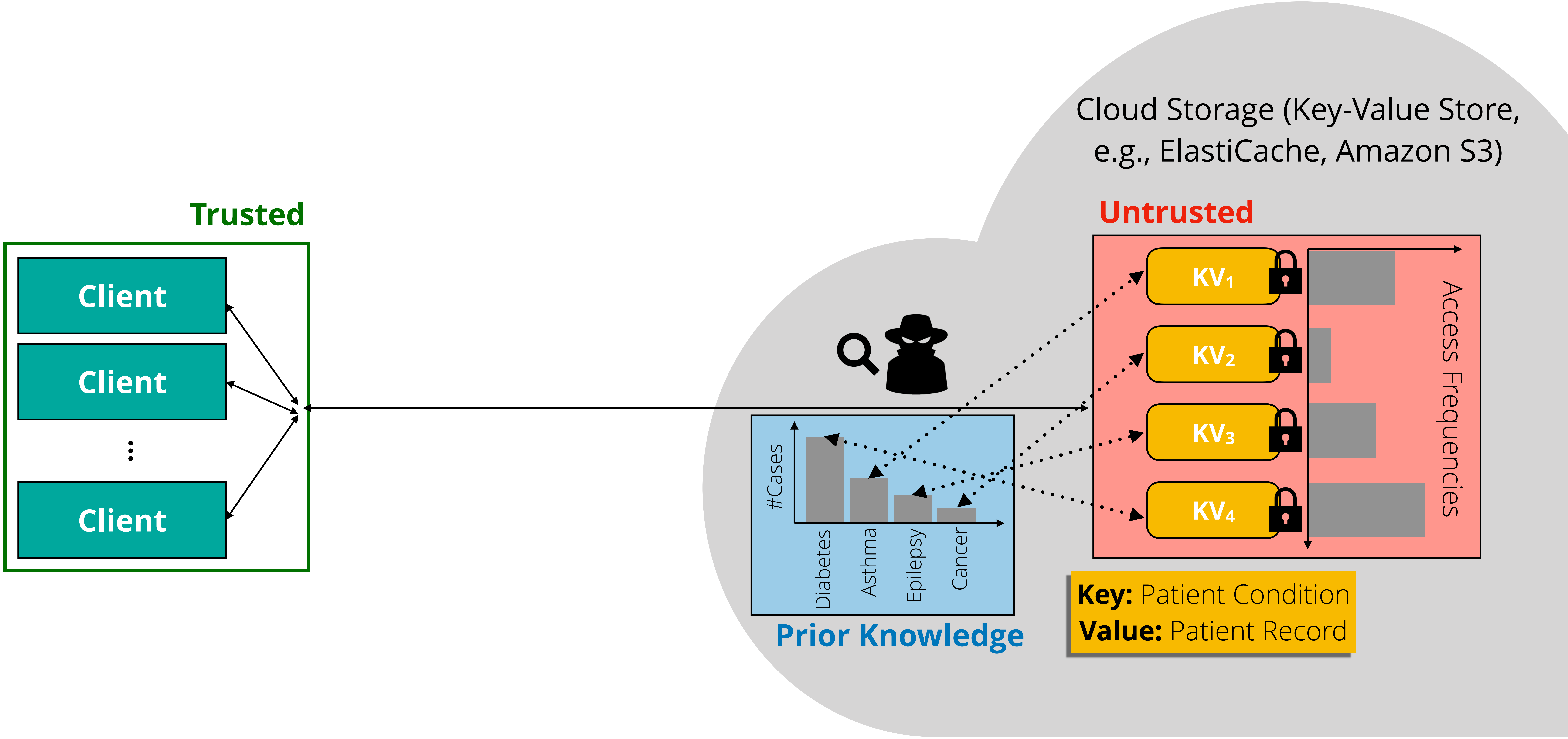
Access Pattern Attacks



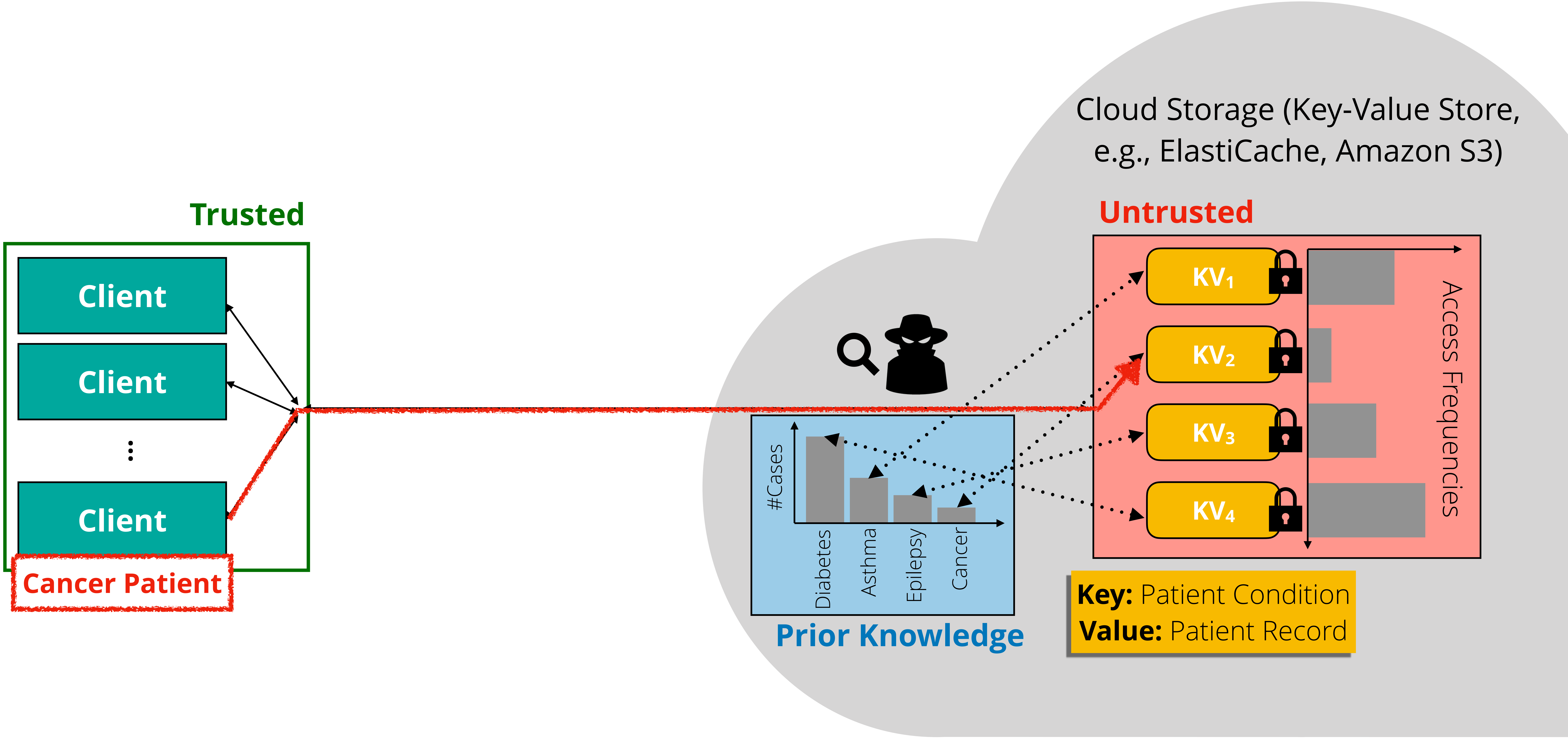
Access Pattern Attacks



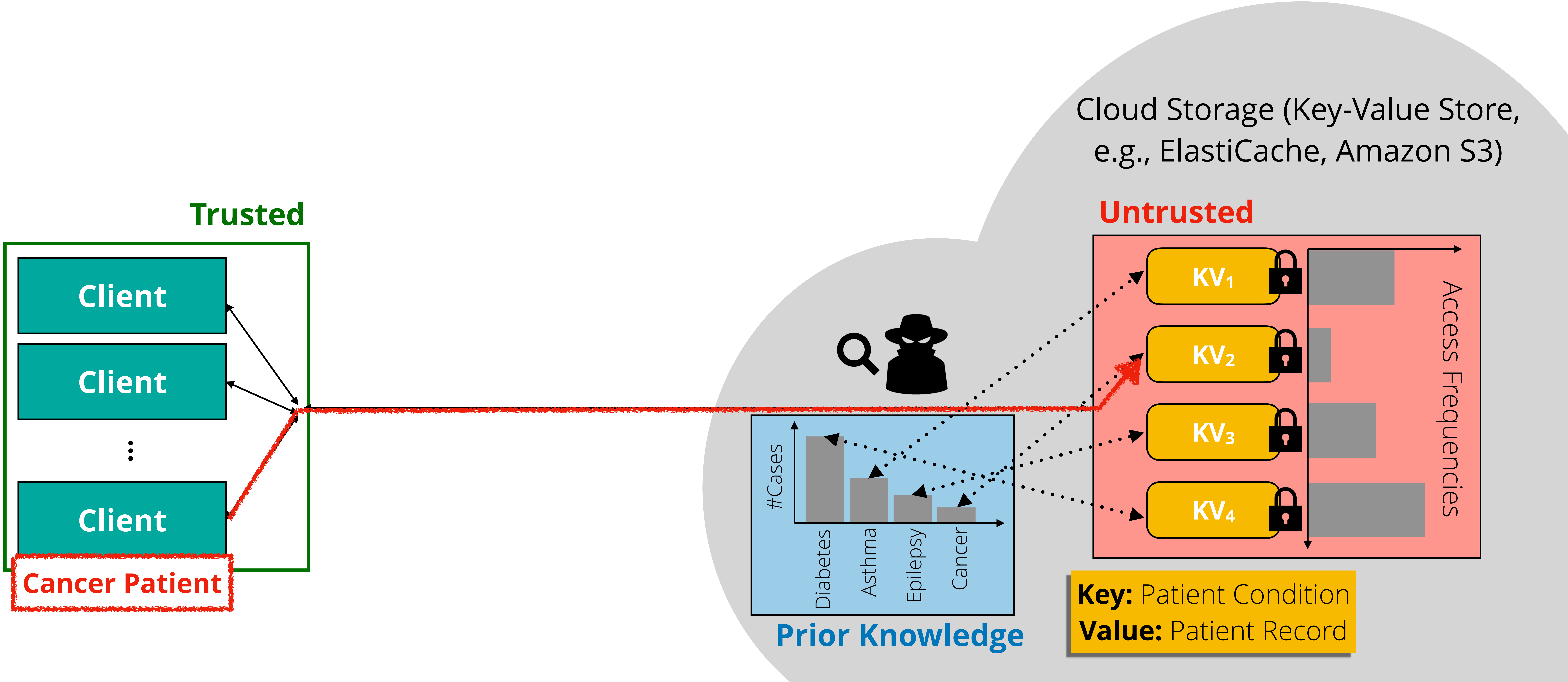
Access Pattern Attacks



Access Pattern Attacks

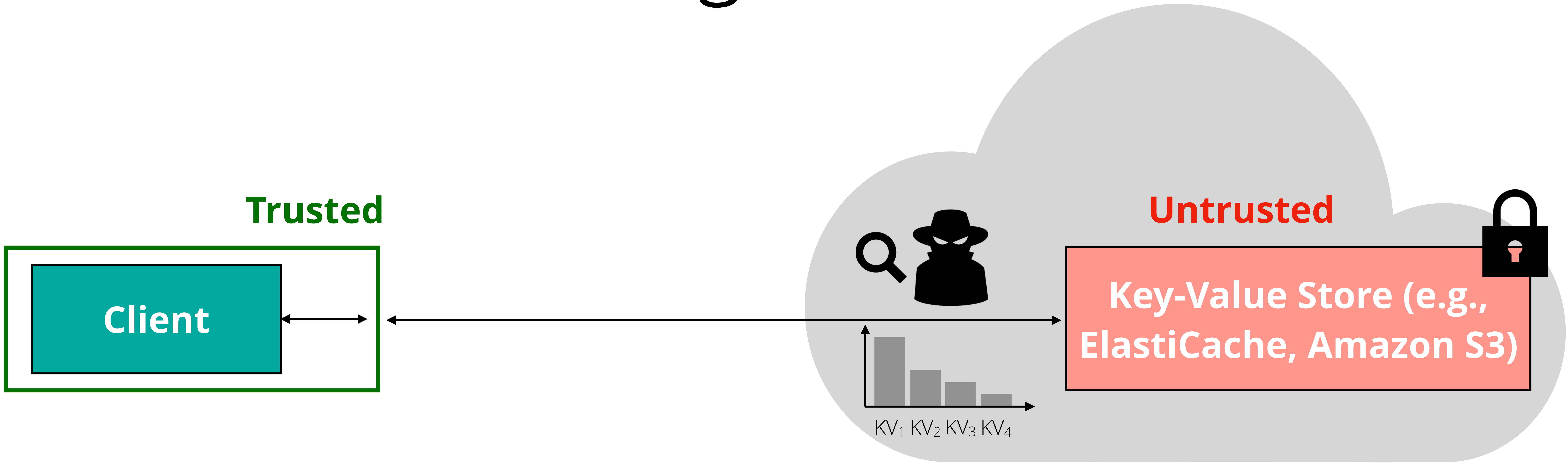


Access Pattern Attacks

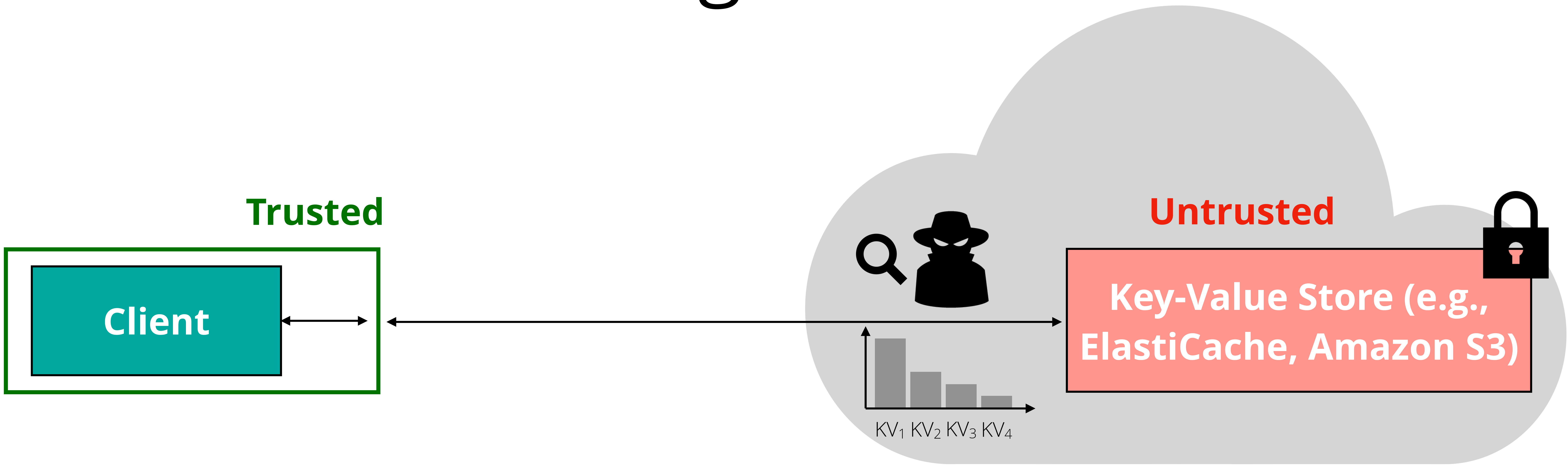


Many practical attacks: [IKK NDSS'12], [CGPR CCS'15], [KKNO CCS'16], [GLMP S&P'19], [KPT S&P'19]

Existing Solutions

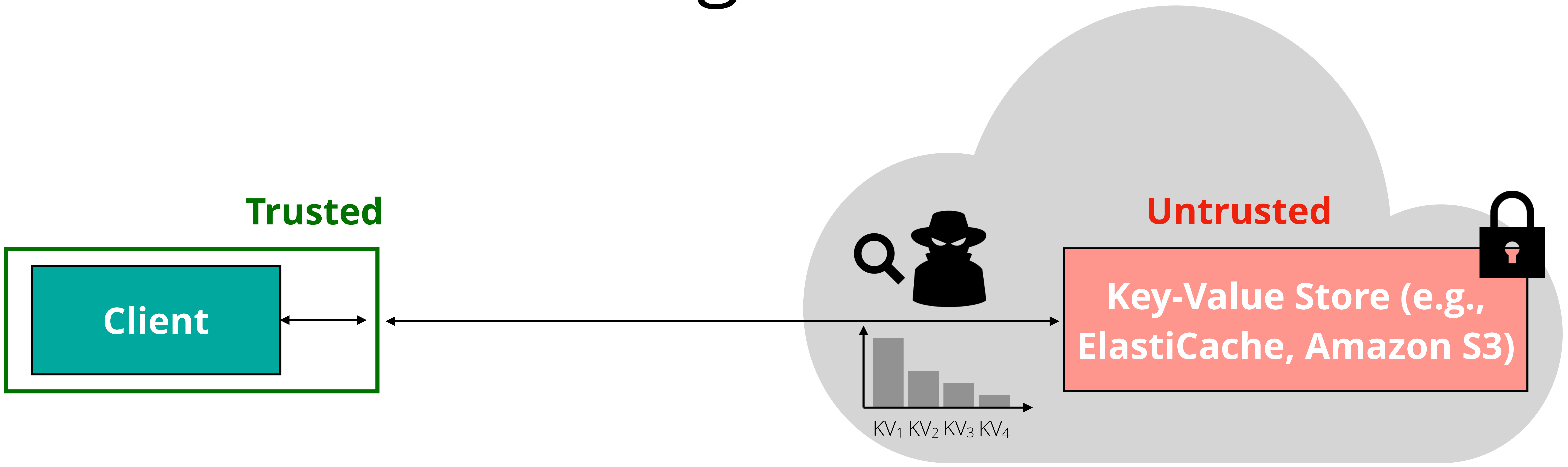


Existing Solutions



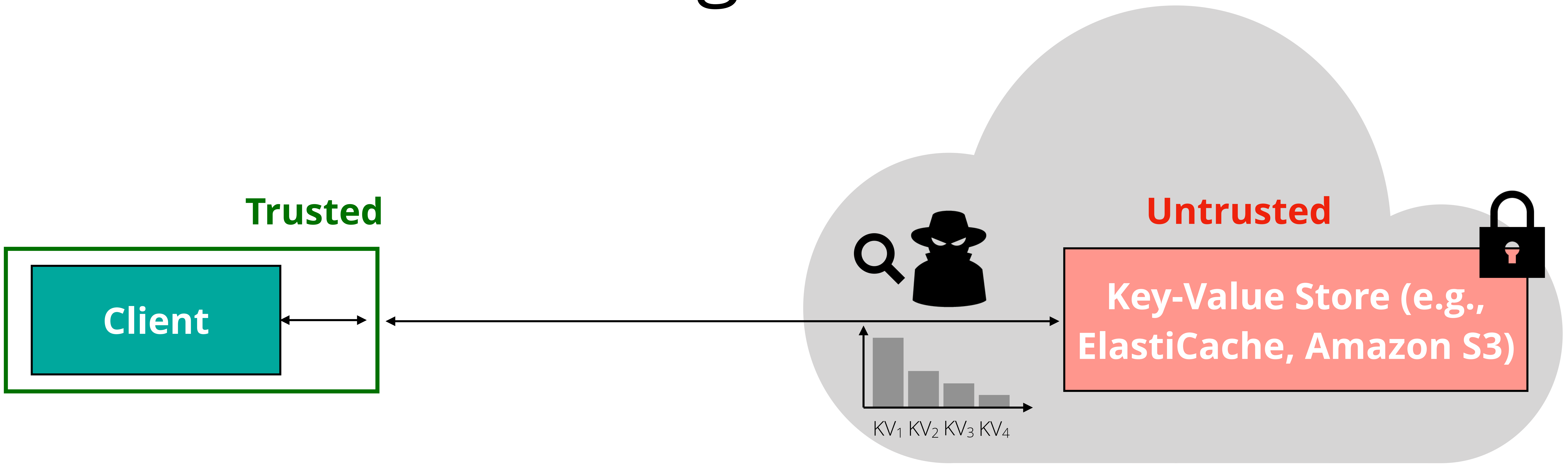
Threat model	Active Adversary
Approach	ORAM, PIR

Existing Solutions



Threat model	Active Adversary
Approach	ORAM, PIR
Security	Strong
Performance	Poor

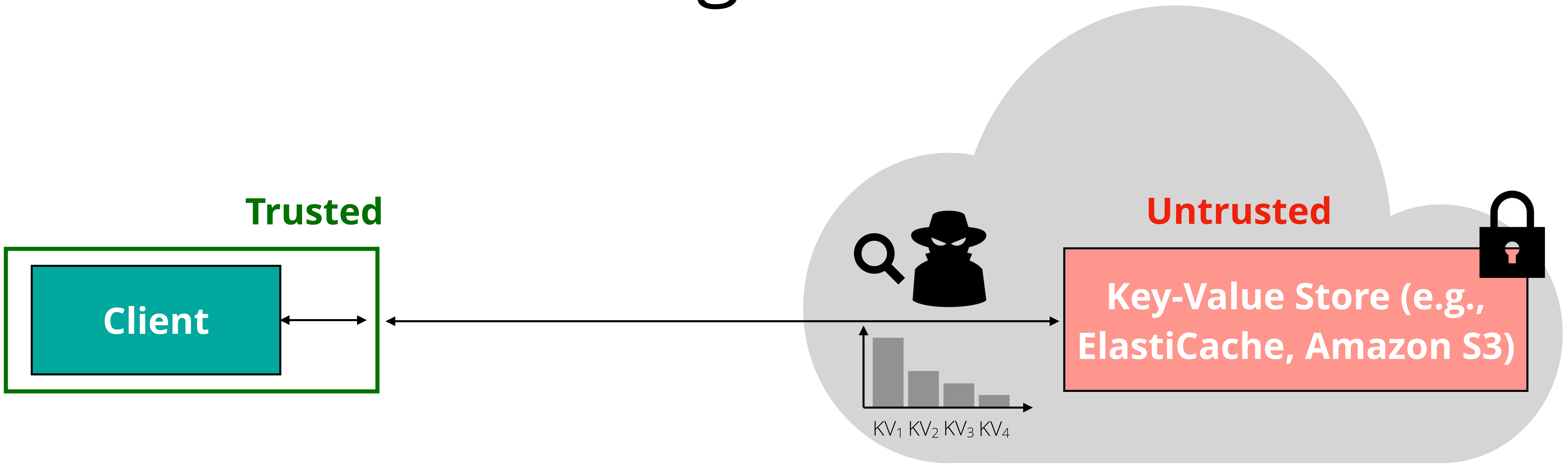
Existing Solutions



Threat model	Active Adversary
Approach	ORAM, PIR
Security	Strong
Performance	Poor

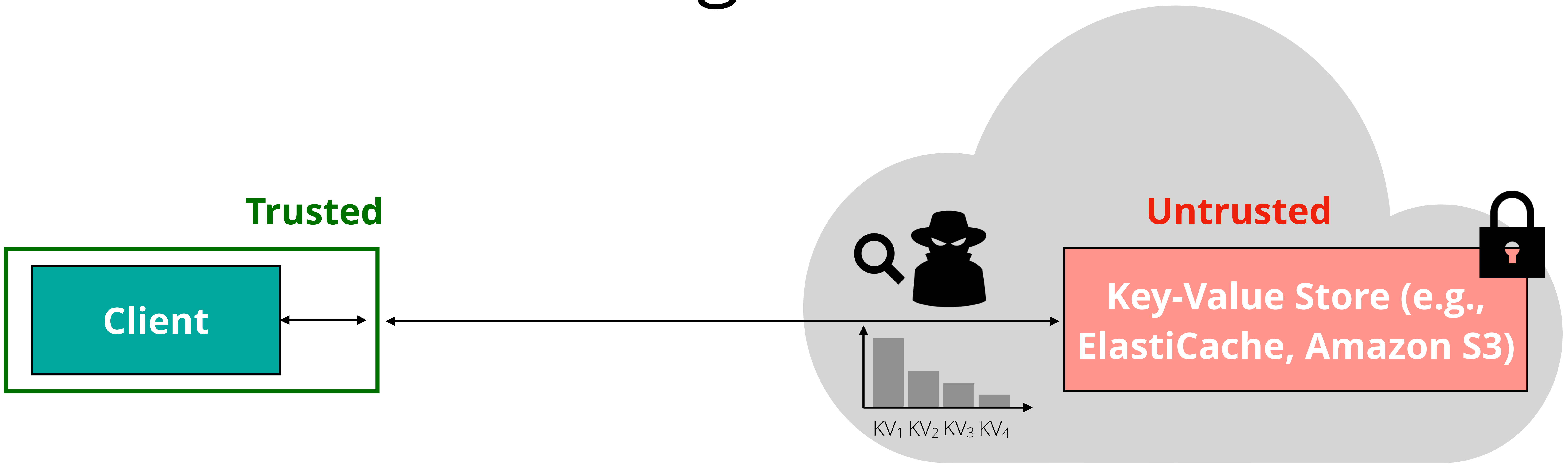
O(log n) bandwidth lower bound [BN ITCS'16, LN CRYPTO'19, …]
8x storage & 1600x bandwidth for real workloads!

Existing Solutions



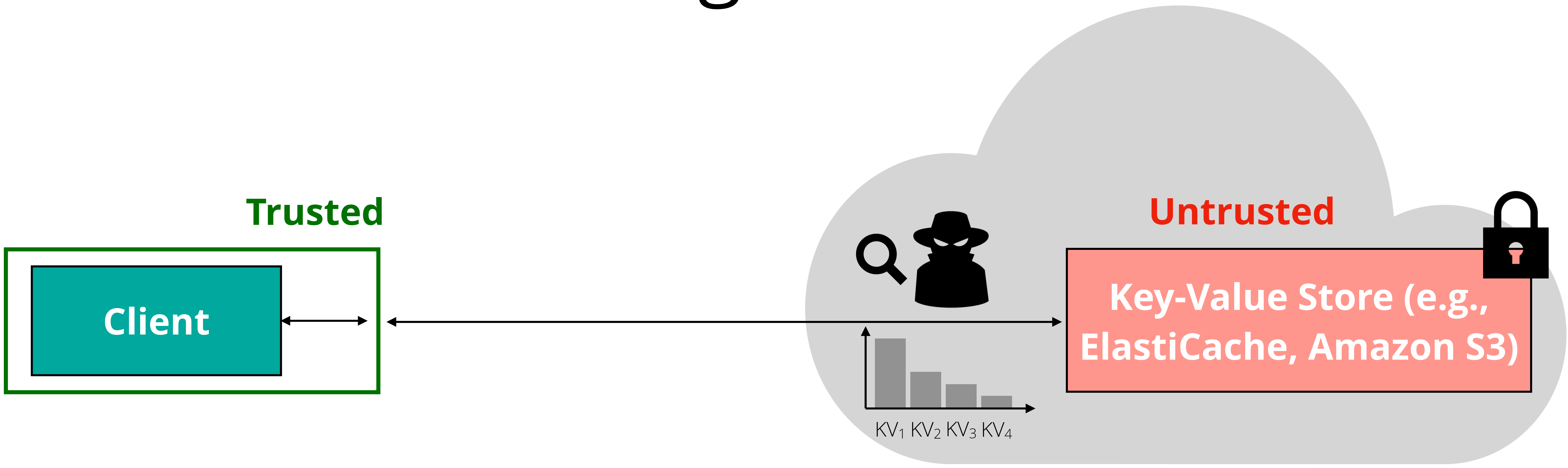
Threat model	Active Adversary	Snapshot Adversary
Approach	ORAM, PIR	SSE
Security	Strong	
Performance	Poor	

Existing Solutions



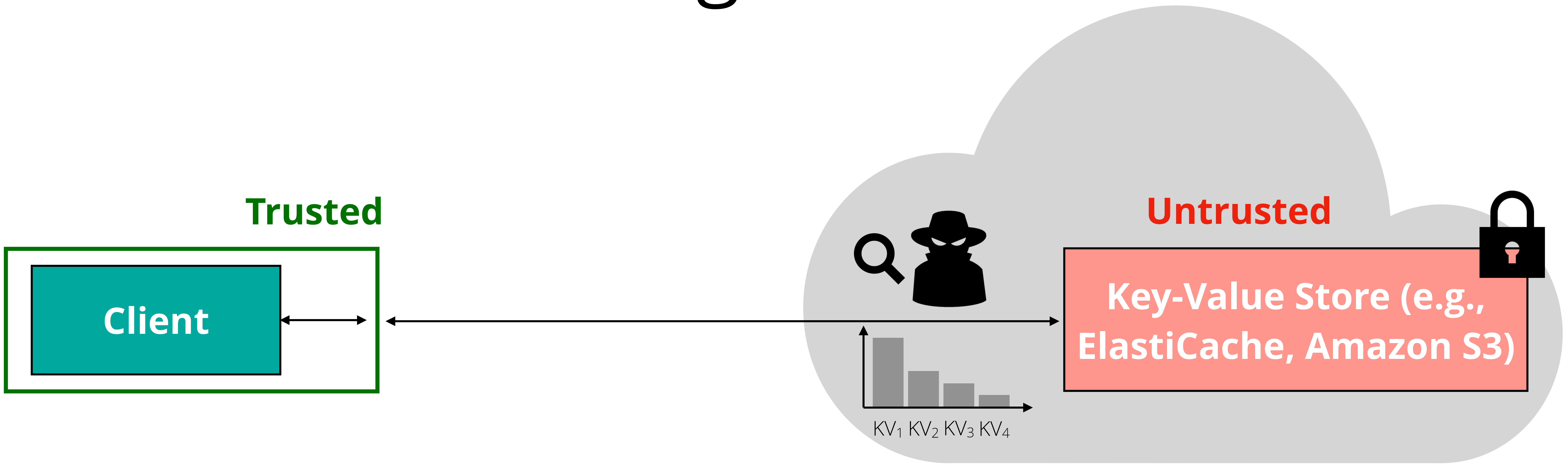
Threat model	Active Adversary	Snapshot Adversary
Approach	ORAM, PIR	SSE
Security	Strong	Weak
Performance	Poor	High

Existing Solutions



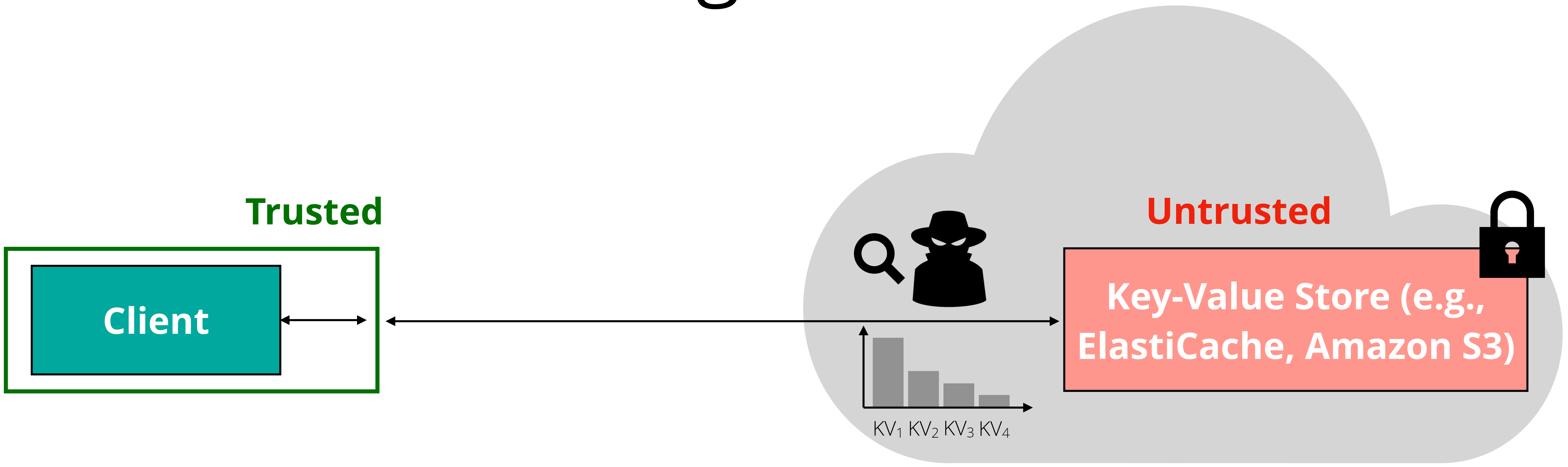
Threat model	Active Adversary		Snapshot Adversary
Approach	ORAM, PIR		SSE
Security	Strong	Unrealistic threat model [GRS HotOS'17]	Weak
Performance	Poor		High

Existing Solutions



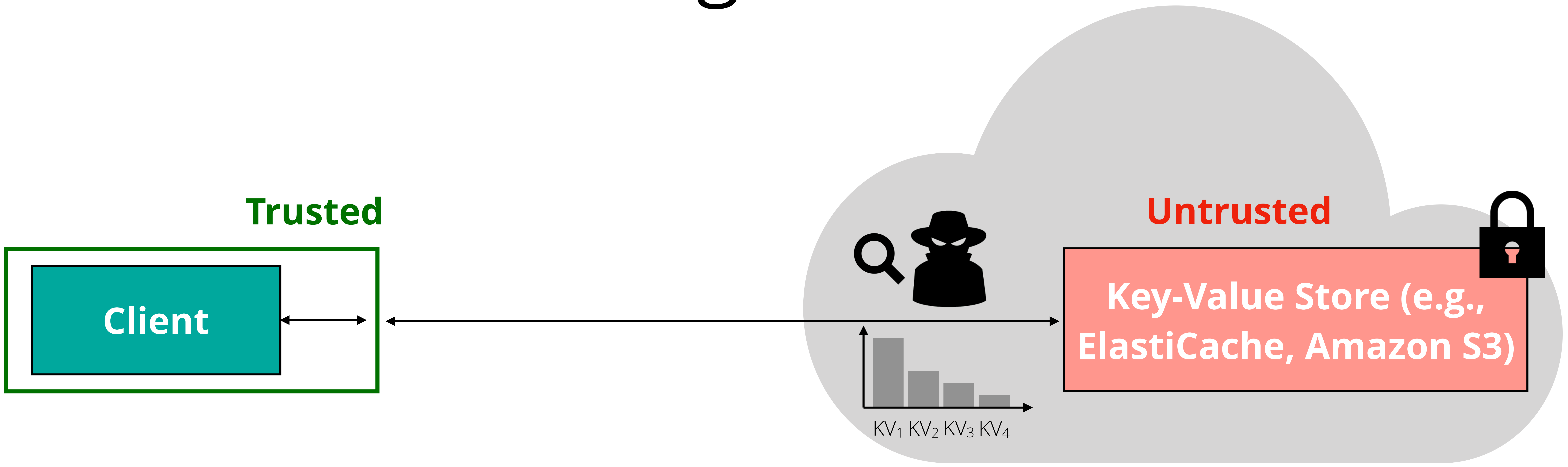
Threat model	Active Adversary	Persistent Passive Adversary	Snapshot Adversary
Approach	ORAM, PIR		SSE
Security	Strong		Weak
Performance	Poor		High

Existing Solutions



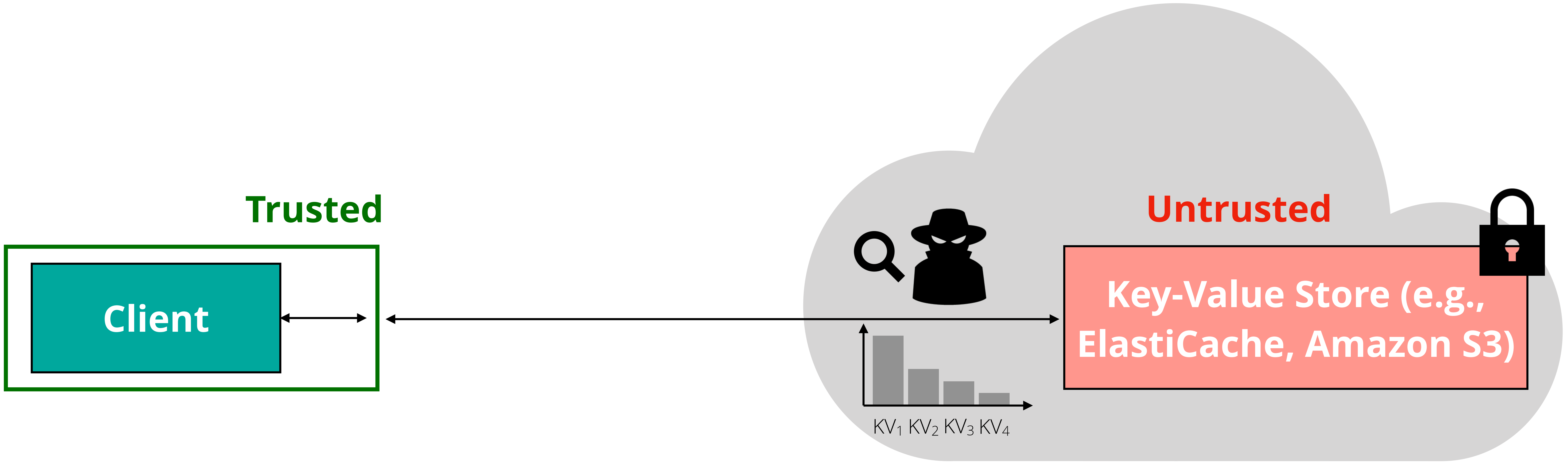
Threat model	Active Adversary	Persistent Passive Adversary	Snapshot Adversary
Approach	ORAM, PIR		SSE
Security	Strong	Captures many real-world cloud deployments	Weak
Performance	Poor		High

Existing Solutions

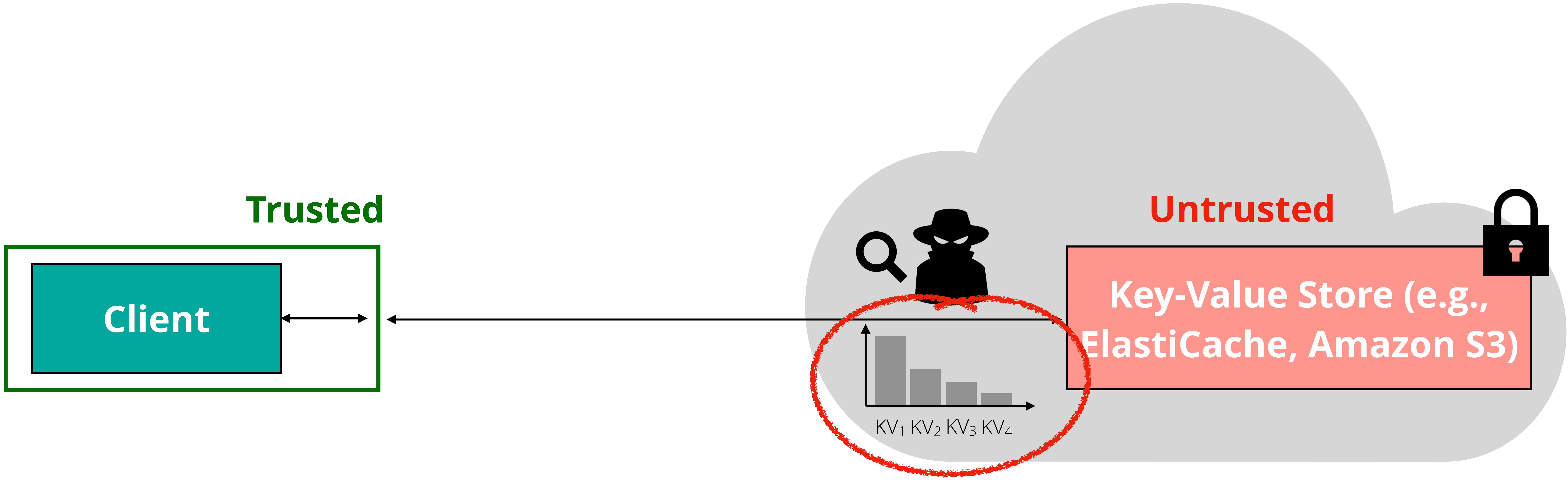


Threat model	Active Adversary	Persistent Passive Adversary	Snapshot Adversary
Approach	ORAM, PIR	?	SSE
Security	Strong	Captures many real-world cloud deployments	Weak
Performance	Poor	Can we achieve low performance overheads?	High

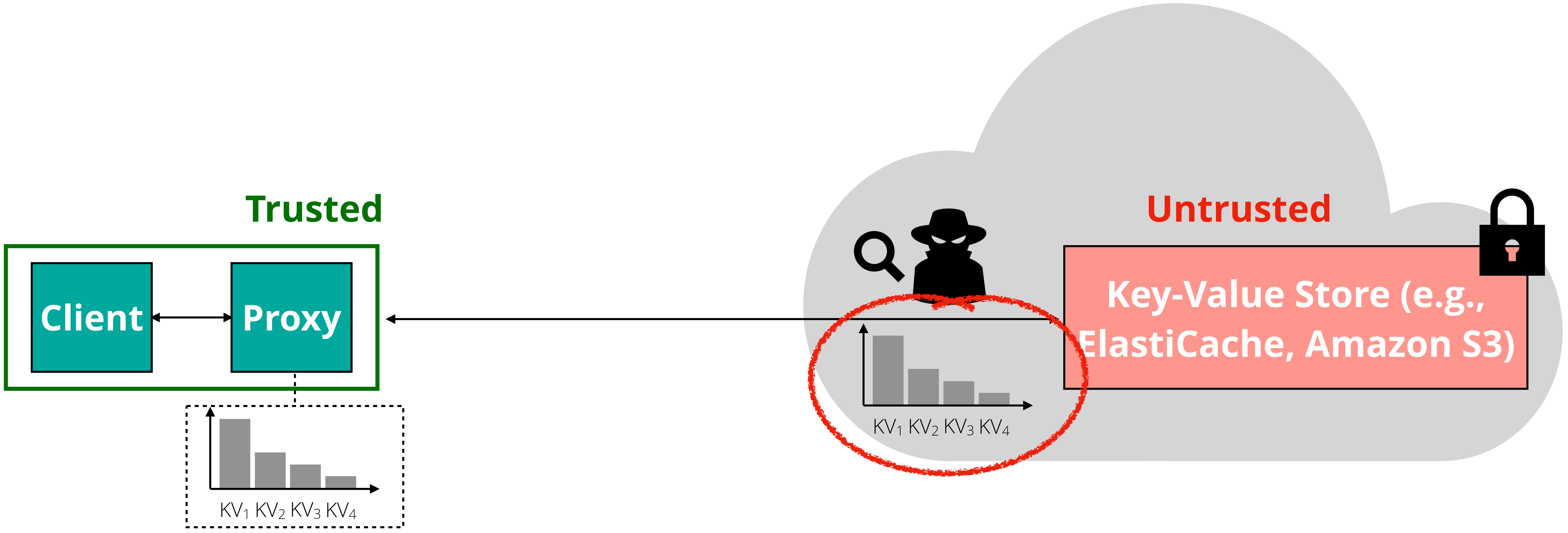
Can we do better?



Can we do better?

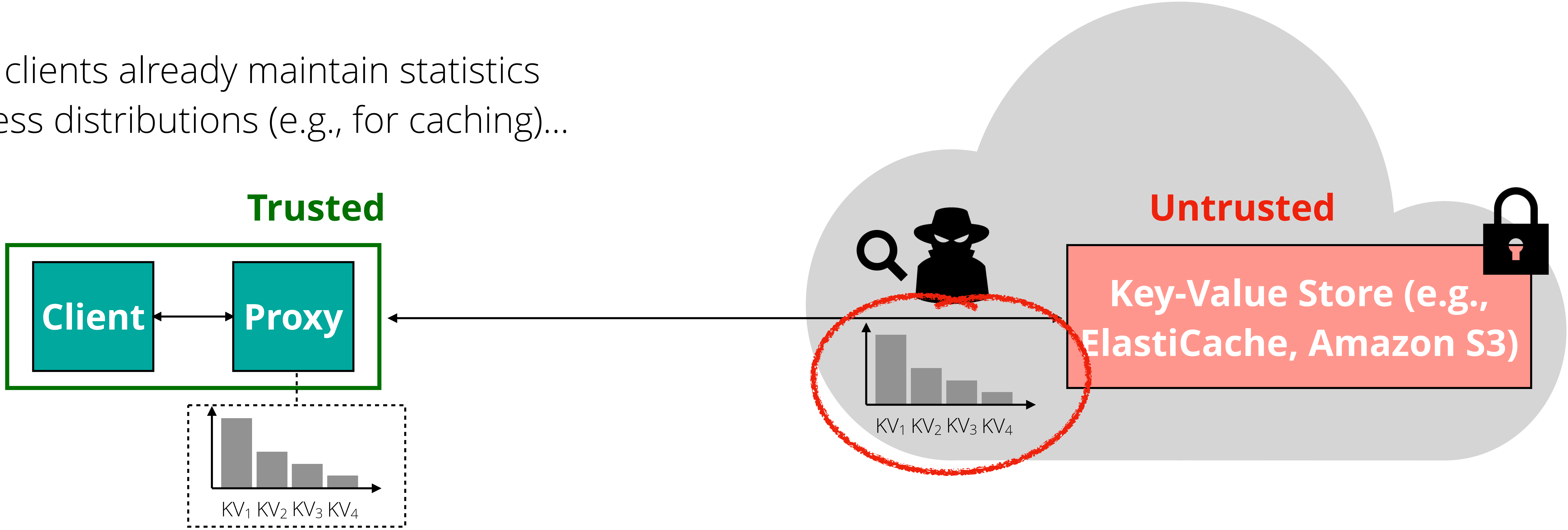


Can we do better?

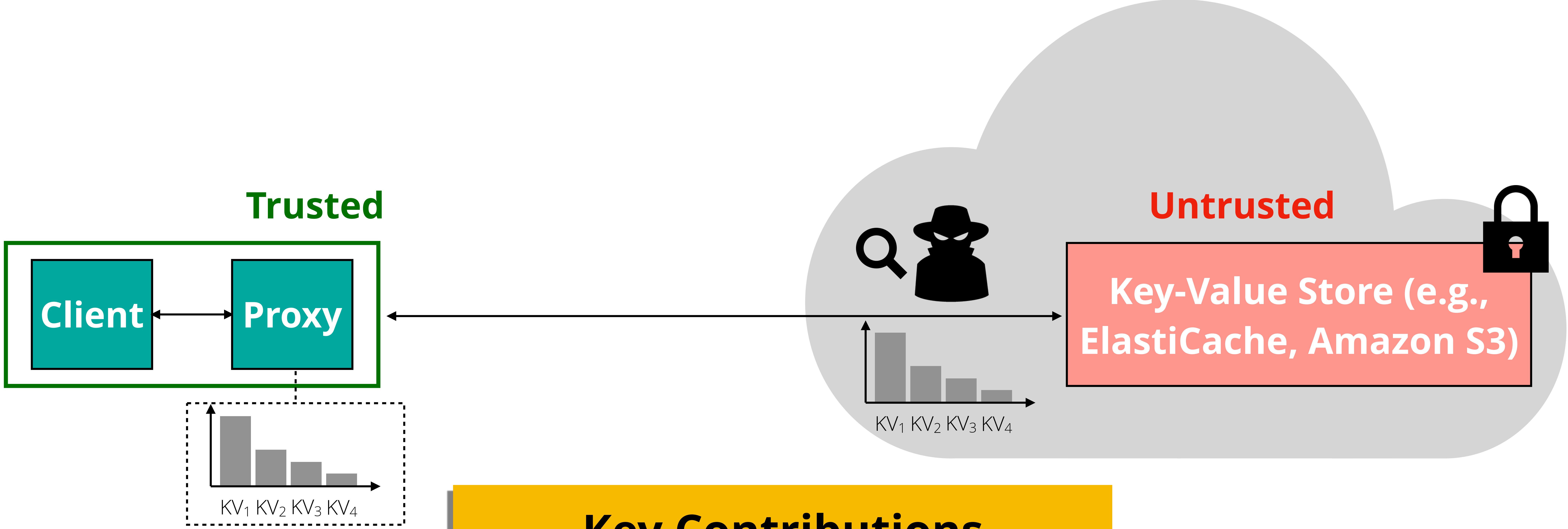


Can we do better?

KV store clients already maintain statistics about access distributions (e.g., for caching)...

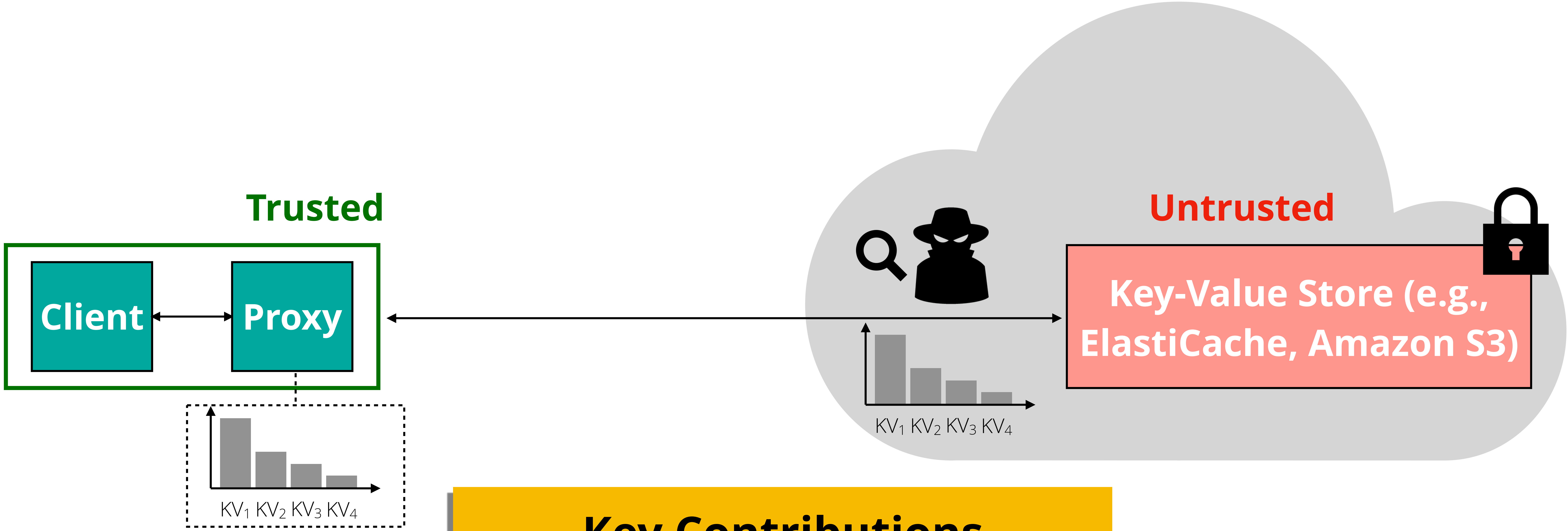


Can we do better?



Key Contributions

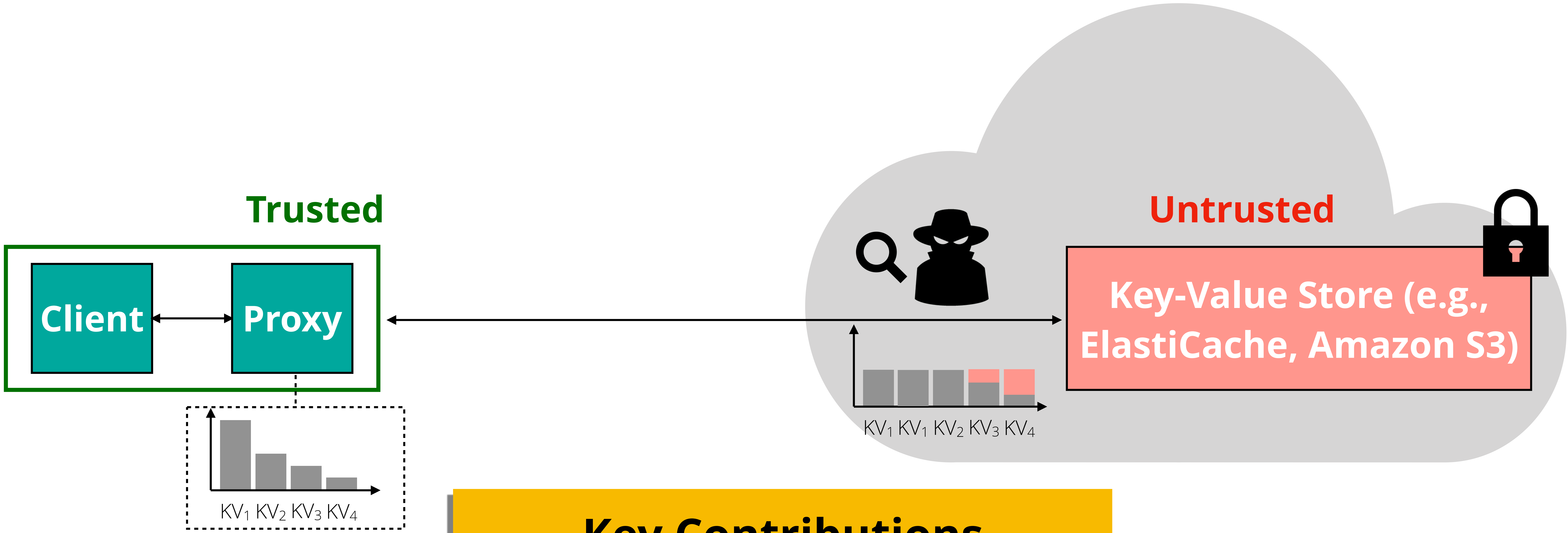
Can we do better?



Key Contributions

- **Pancake:** use **frequency smoothing** over known access distribution to provide security against access pattern attacks with **constant server storage & bandwidth overheads**

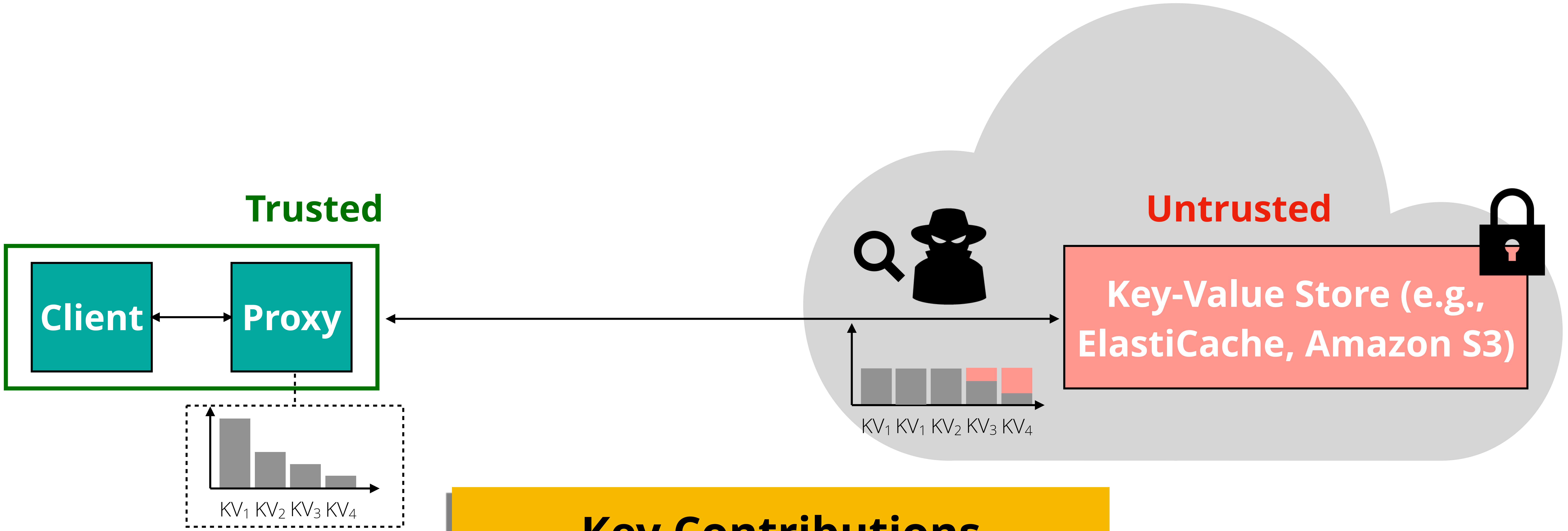
Can we do better?



Key Contributions

- **Pancake:** use **frequency smoothing** over known access distribution to provide security against access pattern attacks with **constant server storage & bandwidth overheads**

Can we do better?



Key Contributions

- **Pancake:** use **frequency smoothing** over known access distribution to provide security against access pattern attacks with **constant server storage & bandwidth overheads**
- **Formal security analysis** showing passive persistent security
- Comprehensive evaluation shows throughput **> 2 orders of magnitude higher than state-of-the art** (PathORAM)!

Frequency Smoothing

Model: Queries drawn from **distribution** π over keys, known to both system & adversary

Frequency Smoothing

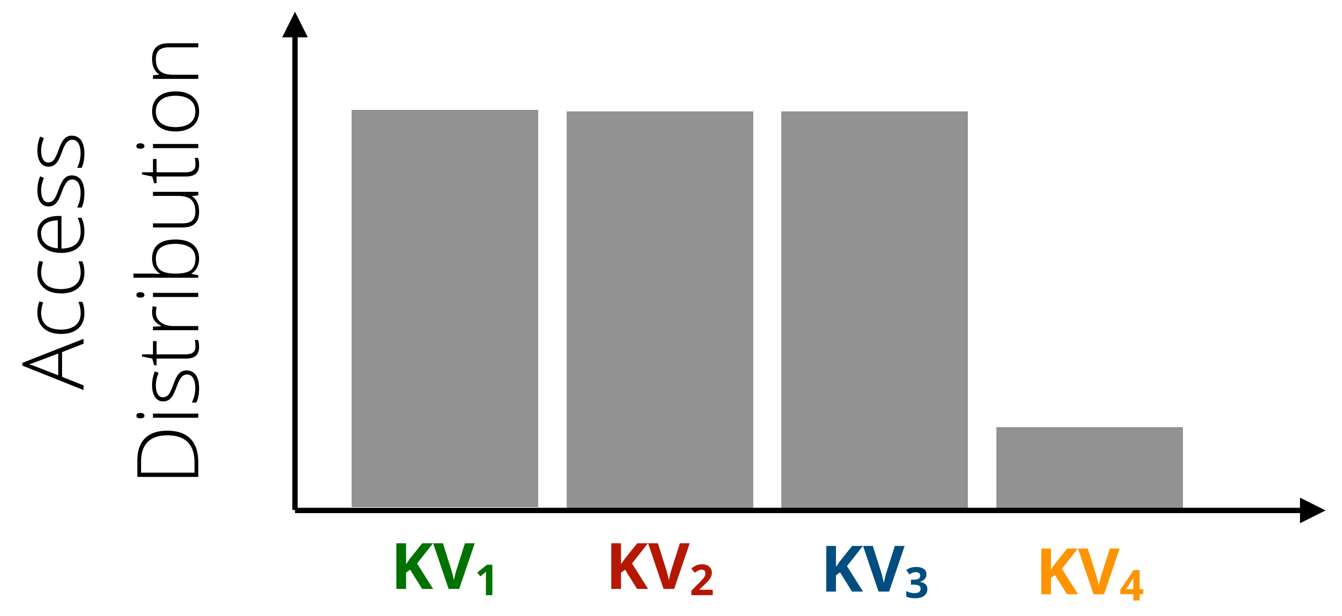
Model: Queries drawn from **distribution** π over keys, known to both system & adversary

Approach: Transform π to a "smooth" distribution over (potentially larger set of) encrypted items

Frequency Smoothing

Model: Queries drawn from **distribution** π over keys, known to both system & adversary

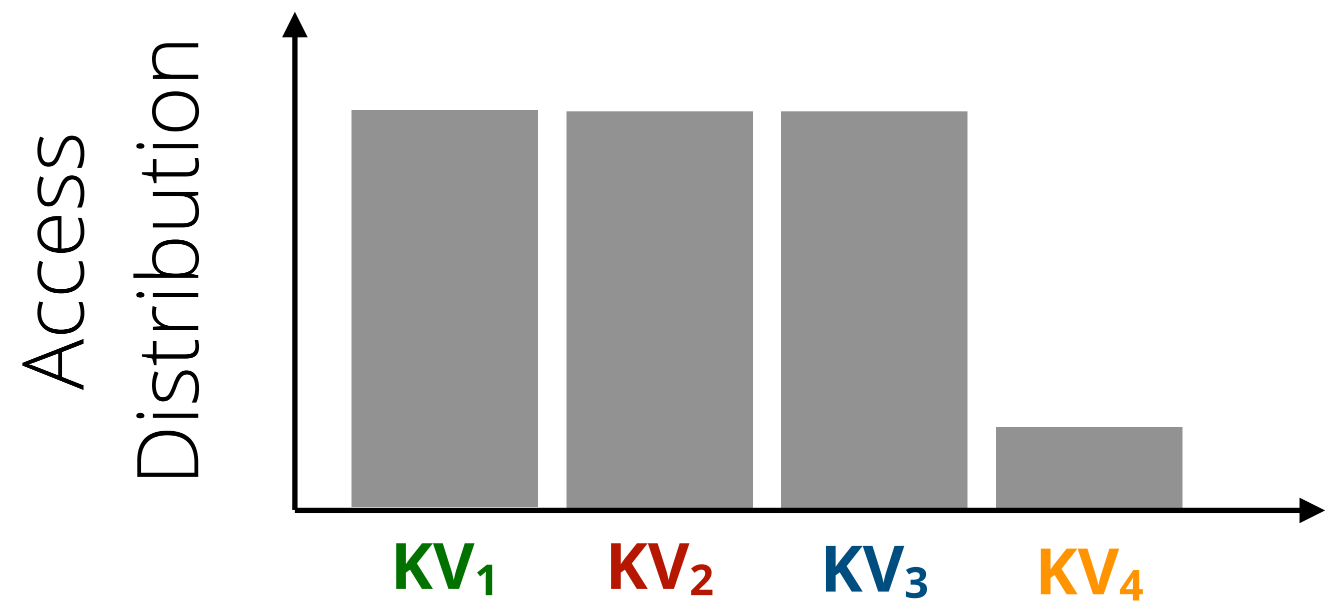
Approach: Transform π to a "smooth" distribution over (potentially larger set of) encrypted items



Frequency Smoothing

Model: Queries drawn from **distribution** π over keys, known to both system & adversary

Approach: Transform π to a "smooth" distribution over (potentially larger set of) encrypted items



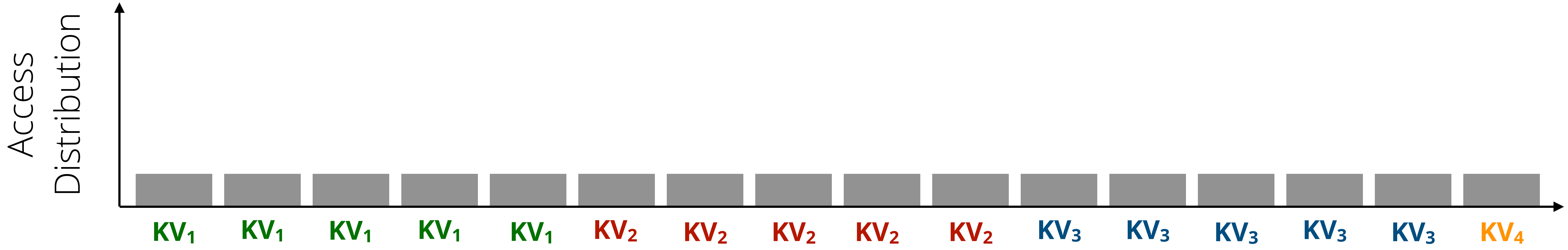
Idea#1: Replication

Replicate popular items → uniform distribution across replicas

Frequency Smoothing

Model: Queries drawn from **distribution** π over keys, known to both system & adversary

Approach: Transform π to a "smooth" distribution over (potentially larger set of) encrypted items



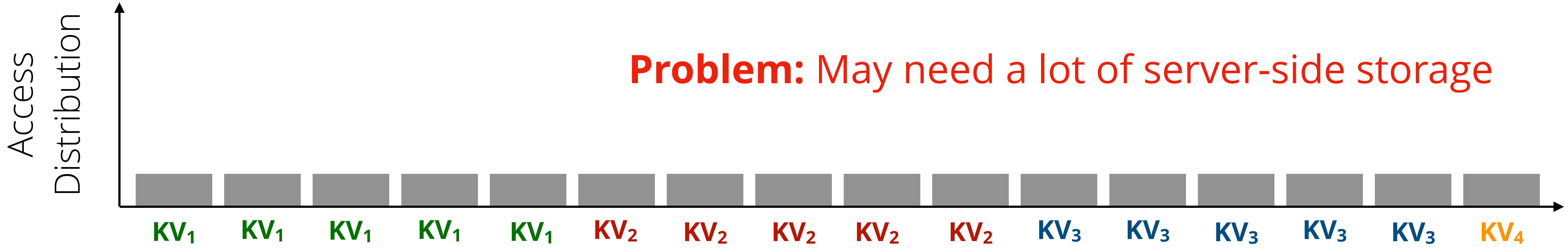
Idea#1: Replication

Replicate popular items → uniform distribution across replicas

Frequency Smoothing

Model: Queries drawn from **distribution** π over keys, known to both system & adversary

Approach: Transform π to a "smooth" distribution over (potentially larger set of) encrypted items



Problem: May need a lot of server-side storage

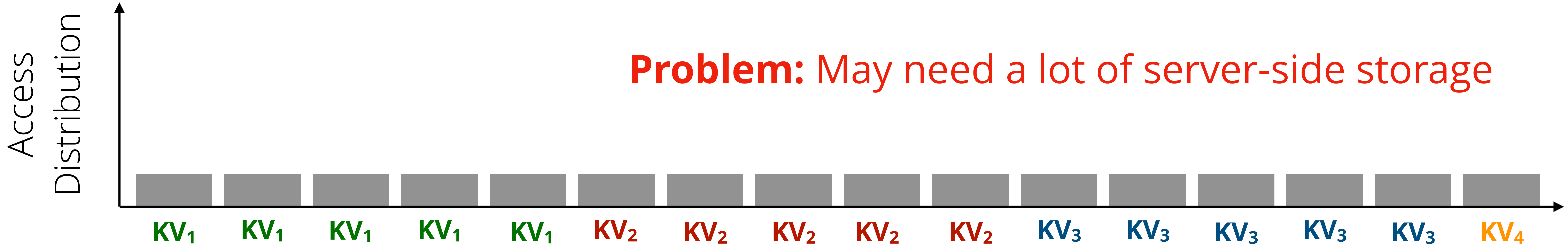
Idea#1: Replication

Replicate popular items → uniform distribution across replicas

Frequency Smoothing

Model: Queries drawn from **distribution π** over keys, known to both system & adversary

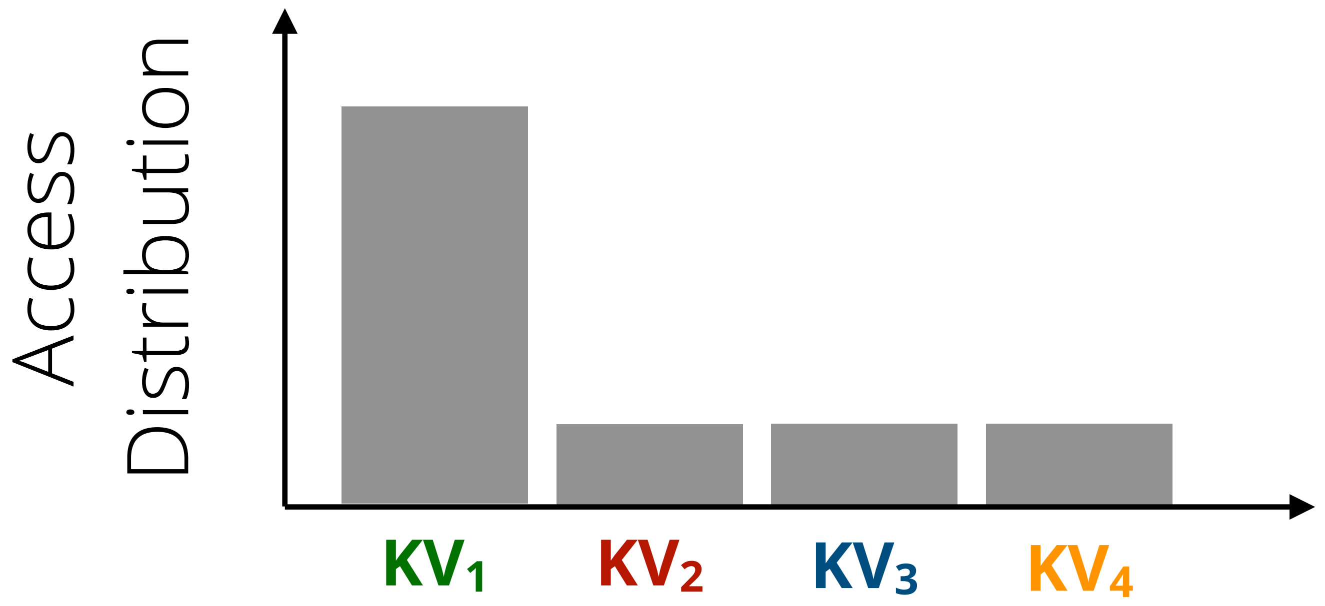
Approach: Transform π to a "smooth" distribution over (potentially larger set of) encrypted items



Problem: May need a lot of server-side storage

Idea#1: Replication

Replicate popular items → uniform distribution across replicas



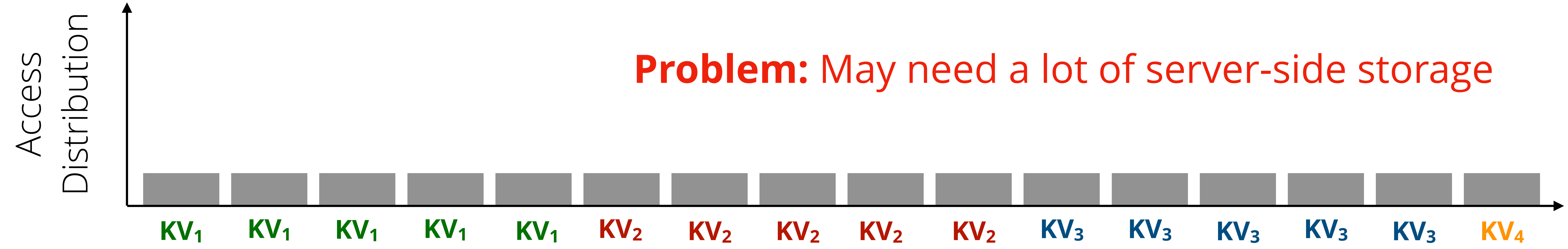
Idea#2: Fake Accesses

Fake accesses to unpopular items → uniform distribution across items

Frequency Smoothing

Model: Queries drawn from **distribution π** over keys, known to both system & adversary

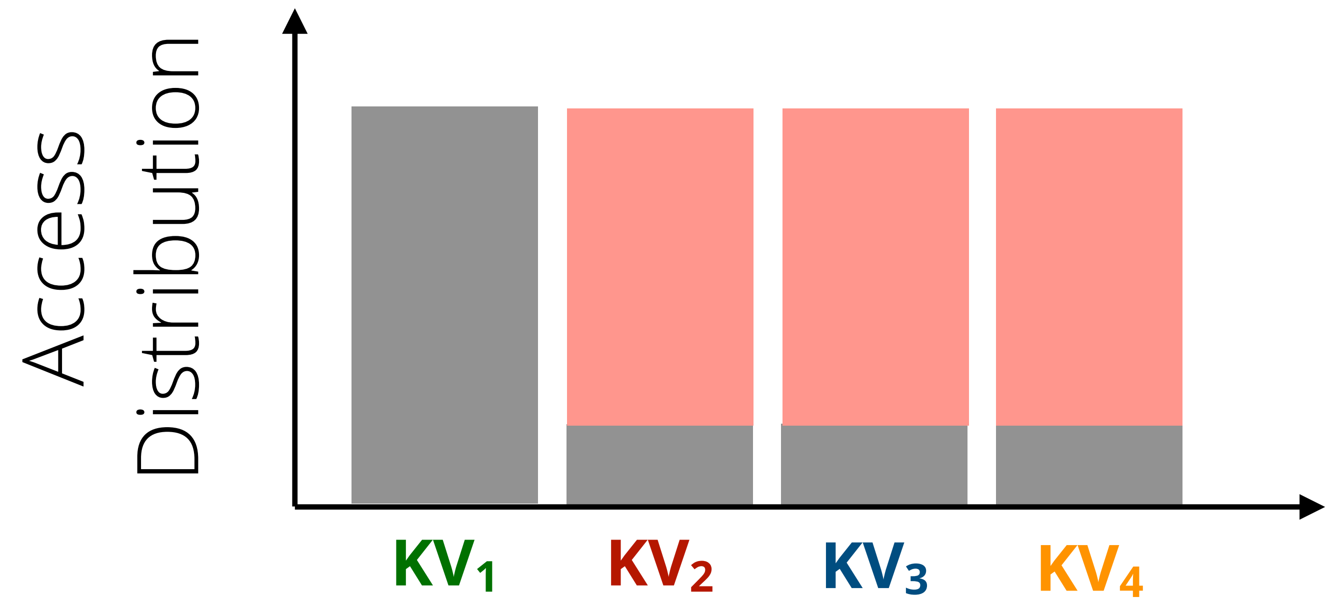
Approach: Transform π to a "smooth" distribution over (potentially larger set of) encrypted items



Problem: May need a lot of server-side storage

Idea#1: Replication

Replicate popular items → uniform distribution across replicas



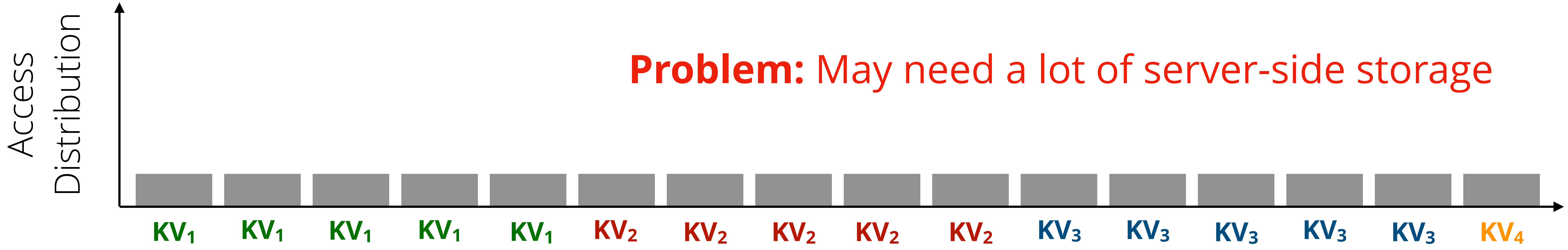
Idea#2: Fake Accesses

Fake accesses to unpopular items → uniform distribution across items

Frequency Smoothing

Model: Queries drawn from **distribution** π over keys, known to both system & adversary

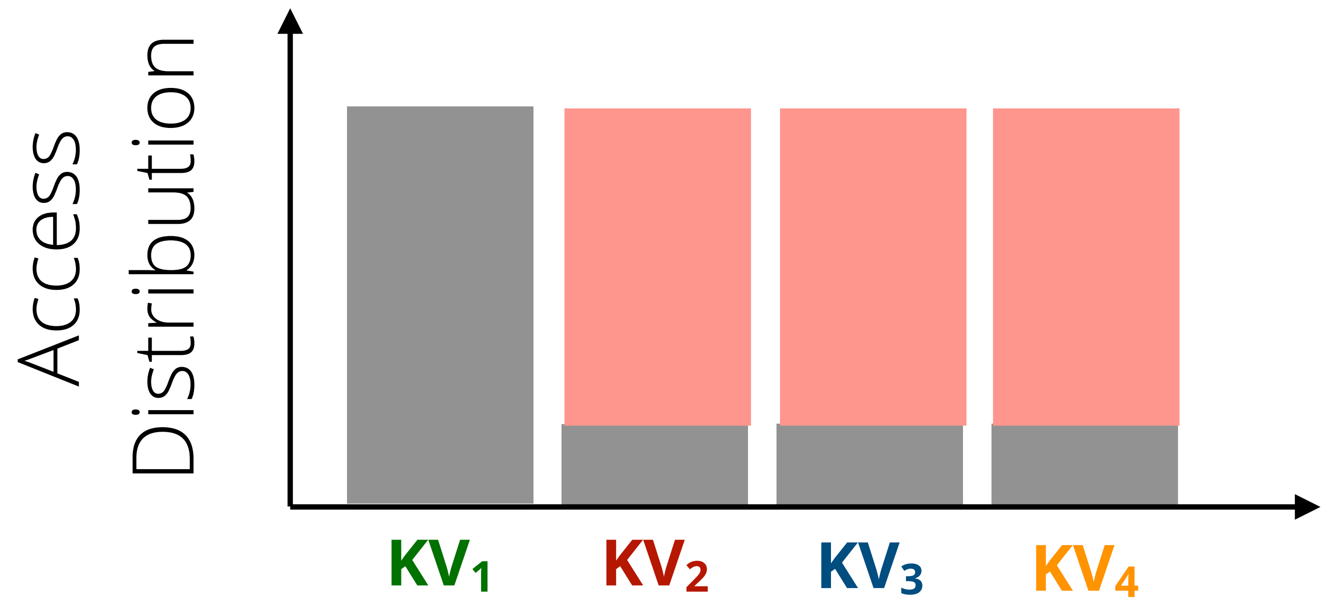
Approach: Transform π to a "smooth" distribution over (potentially larger set of) encrypted items



Problem: May need a lot of server-side storage

Idea#1: Replication

Replicate popular items → uniform distribution across replicas



Problem: May add a lot of bandwidth overheads

Idea#2: Fake Accesses

Fake accesses to unpopular items → uniform distribution across items

Pancake

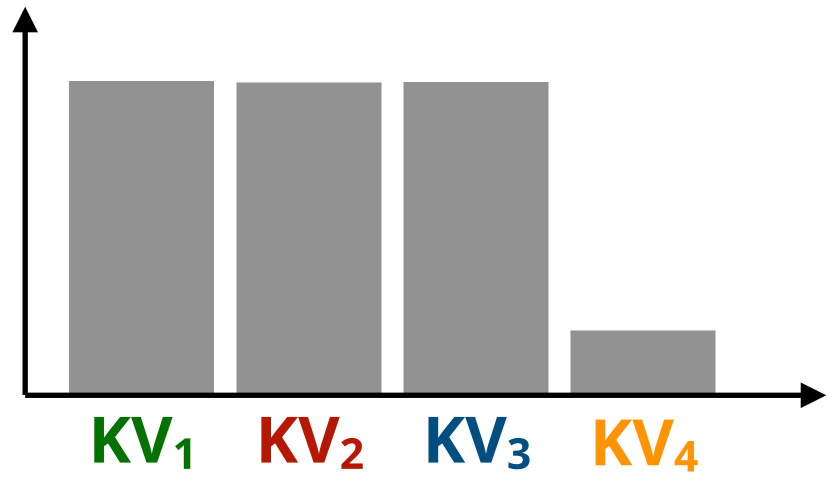
Model: Queries drawn from **distribution** π over keys, known to both system & adversary

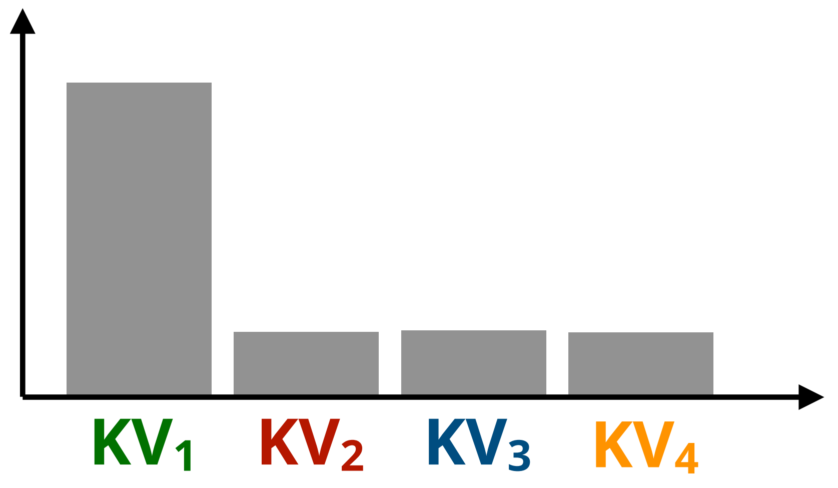
Combine replication and fake accesses!

Pancake

Model: Queries drawn from **distribution π** over keys, known to both system & adversary

Combine replication and fake accesses!

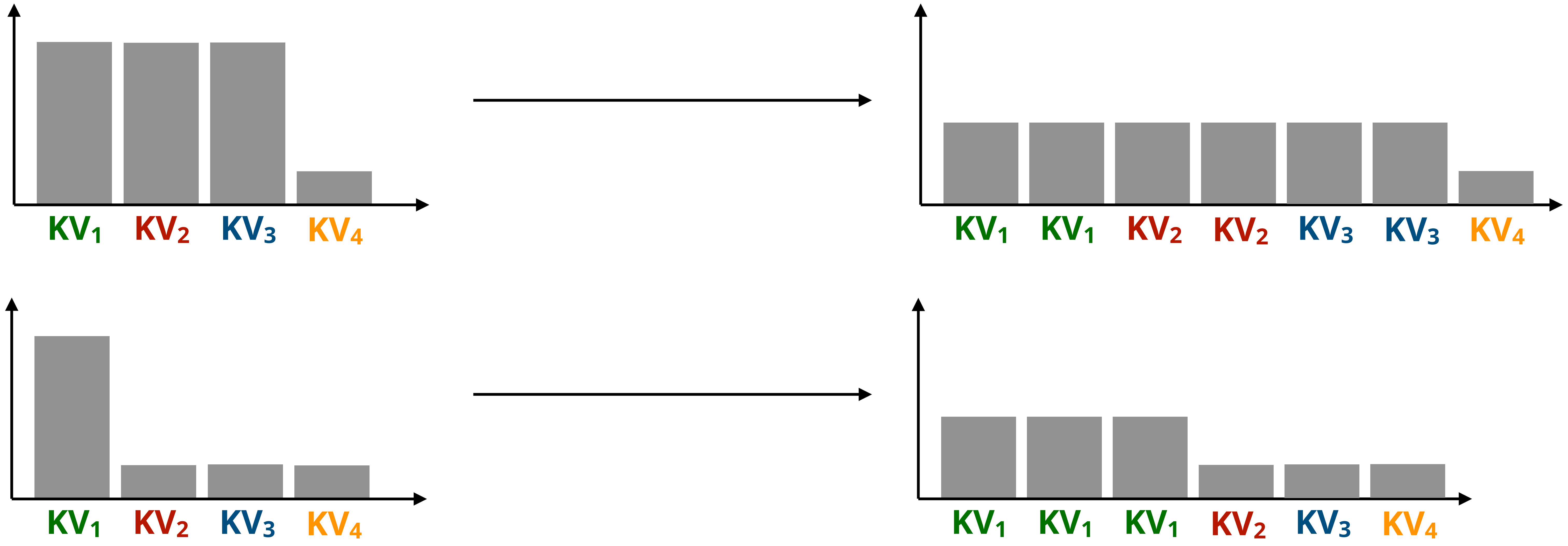




Pancake

Model: Queries drawn from **distribution π** over keys, known to both system & adversary

Combine replication and fake accesses!

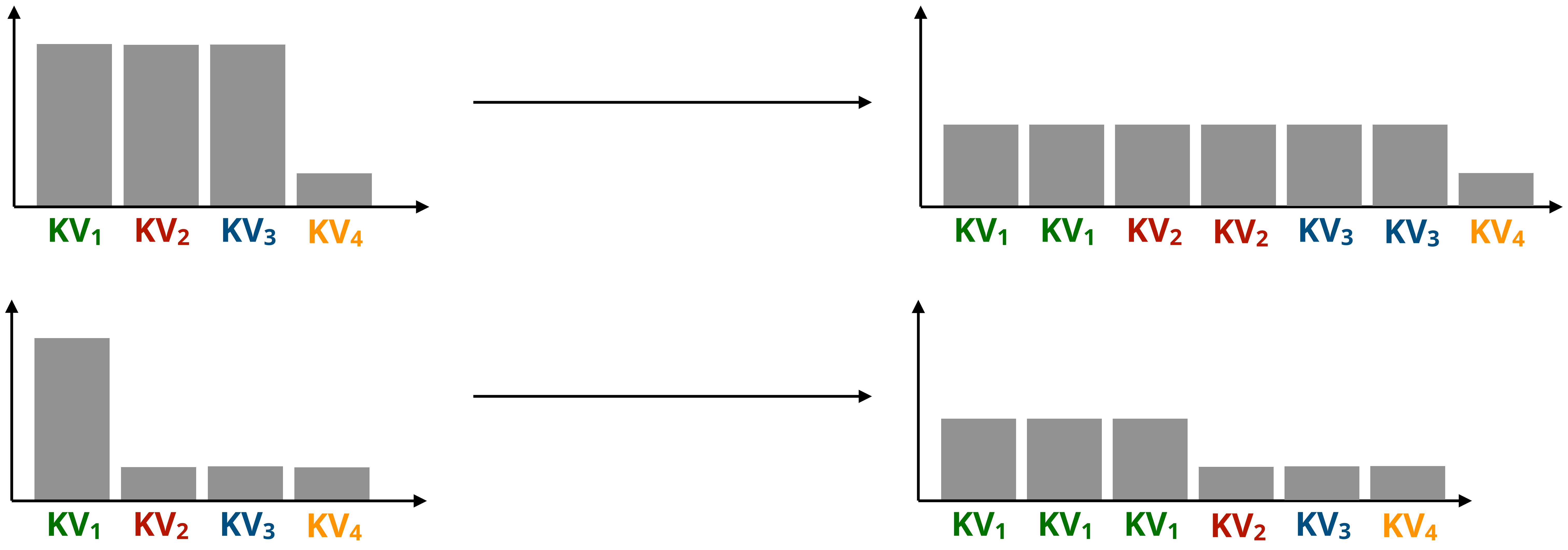


Step 1: Replication - Create "just enough" replicas to *partially* smooth out distribution π with bounded storage

Pancake

Model: Queries drawn from **distribution π** over keys, known to both system & adversary

Combine replication and fake accesses!



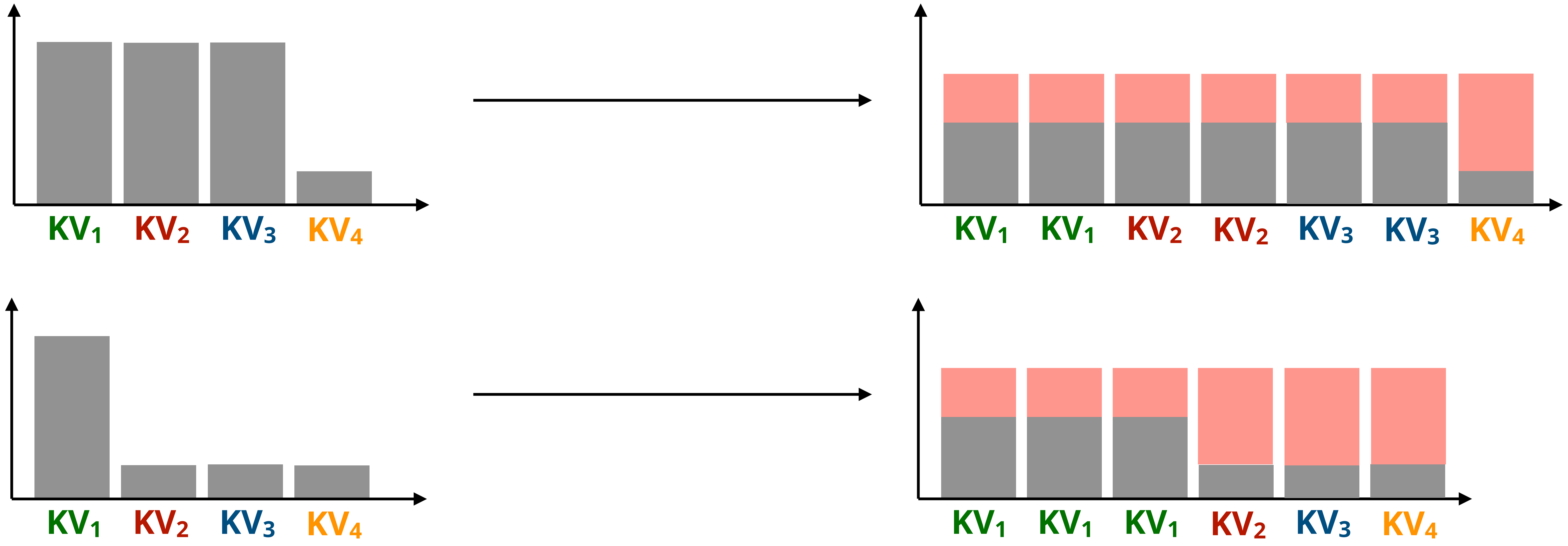
Step 1: Replication - Create "just enough" replicas to *partially* smooth out distribution π with bounded storage

At most 2x total KV pairs

Pancake

Model: Queries drawn from **distribution π** over keys, known to both system & adversary

Combine replication and fake accesses!



Step 1: Replication - Create "just enough" replicas to *partially* smooth out distribution π with bounded storage

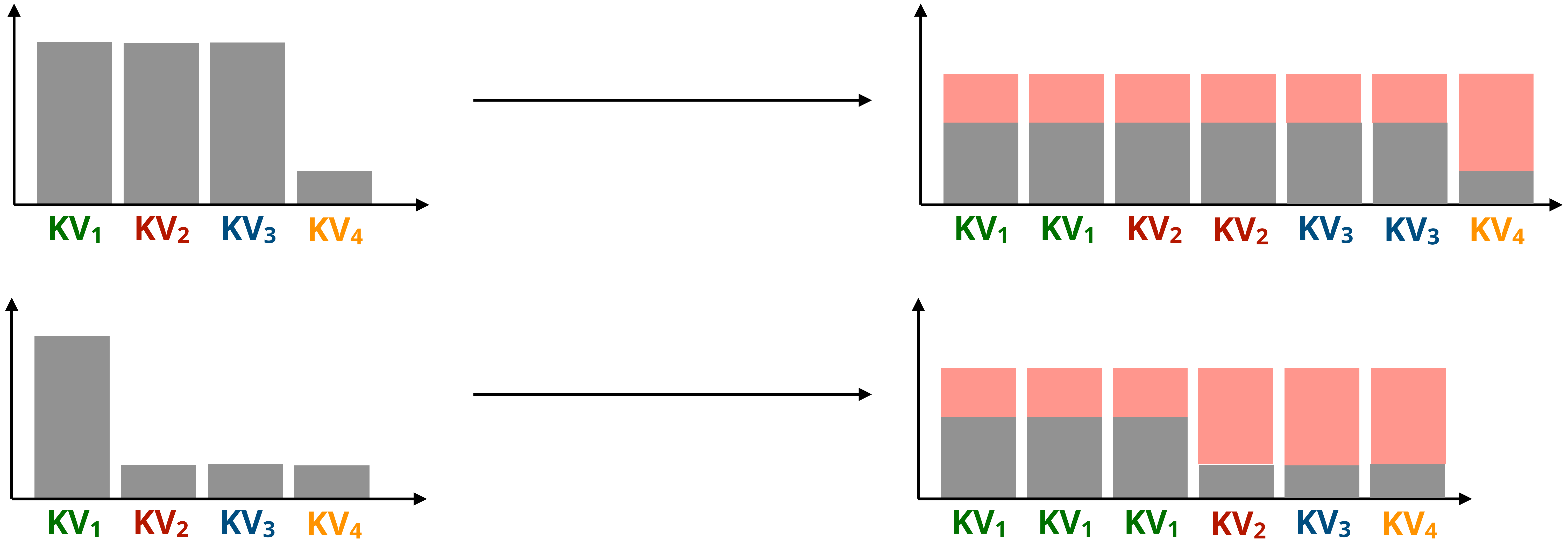
At most 2x total KV pairs

Step 2: Add fake access distribution π_f to smooth out the resulting distribution completely

Pancake

Model: Queries drawn from **distribution π** over keys, known to both system & adversary

Combine replication and fake accesses!



Step 1: Replication - Create "just enough" replicas to *partially* smooth out distribution π with bounded storage

At most 2x total KV pairs

Step 2: Add fake access distribution π_f to smooth out the resulting distribution completely

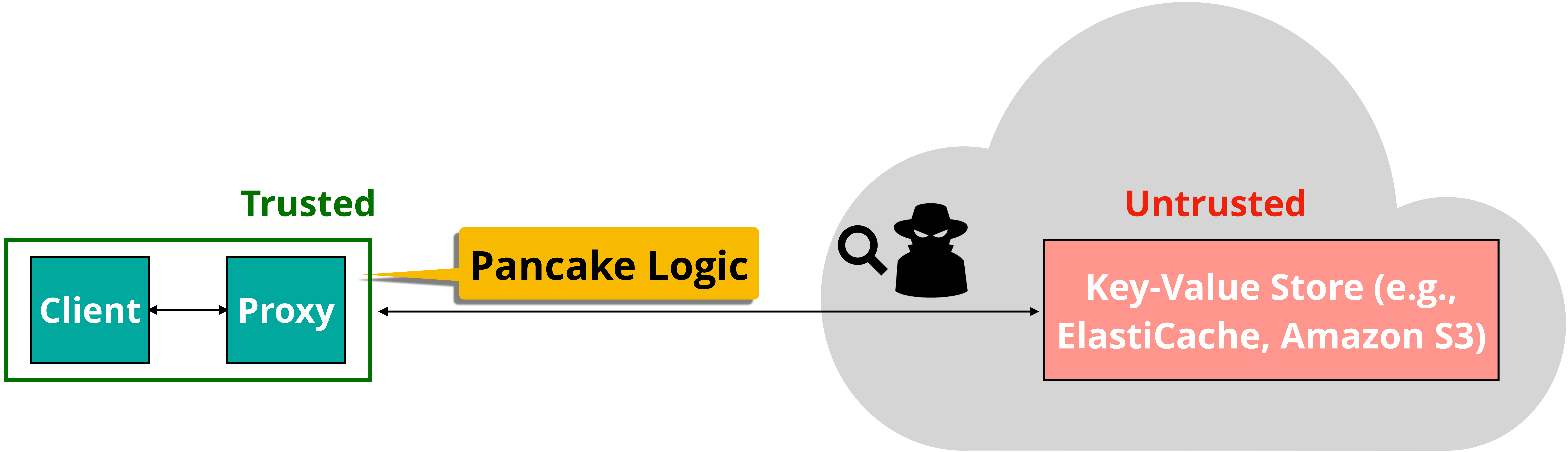
At most one fake access (from π_f) per real access (from π)

Pancake

Combine replication and fake accesses!

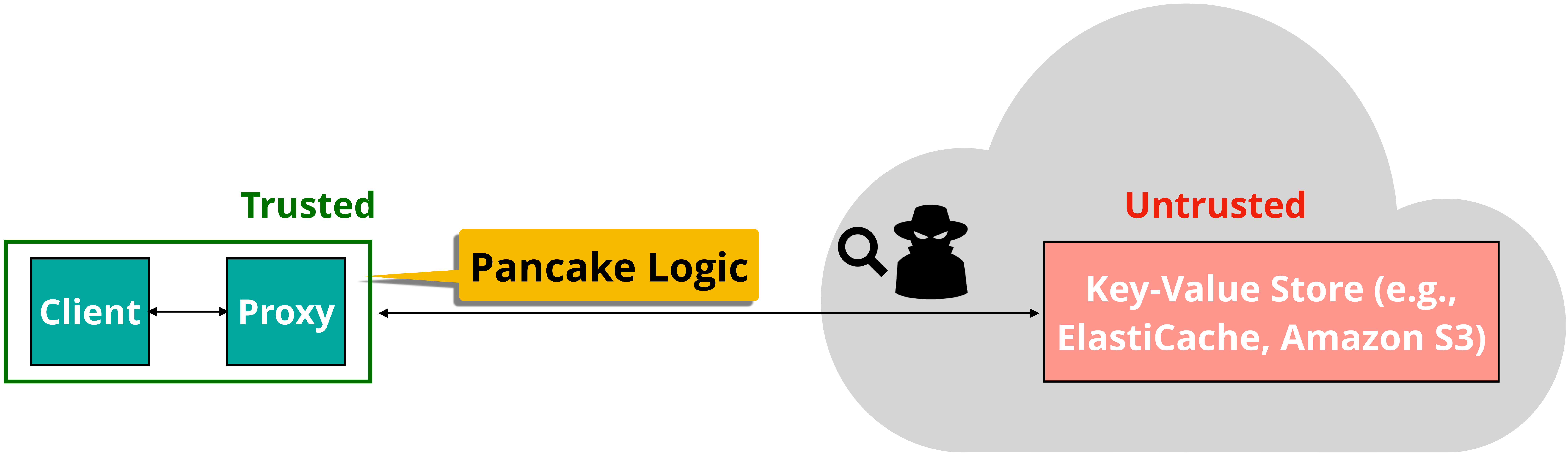
Pancake

Combine replication and fake accesses!



Pancake

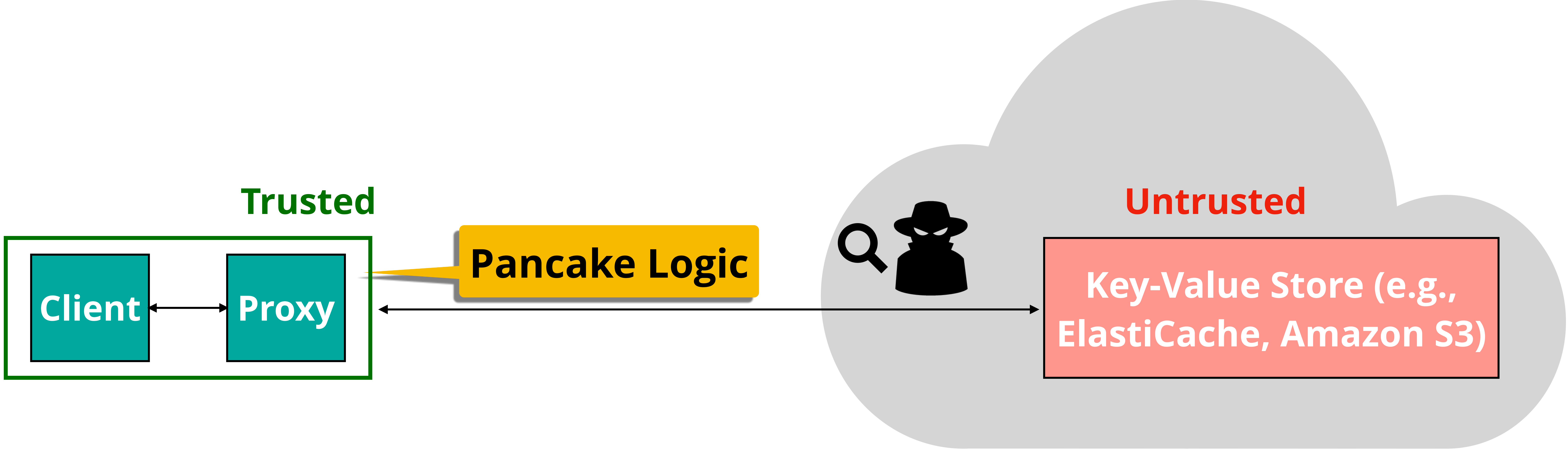
Combine replication and fake accesses!



Challenge: How to issue fake+real accesses without revealing which is which?

Pancake

Combine replication and fake accesses!

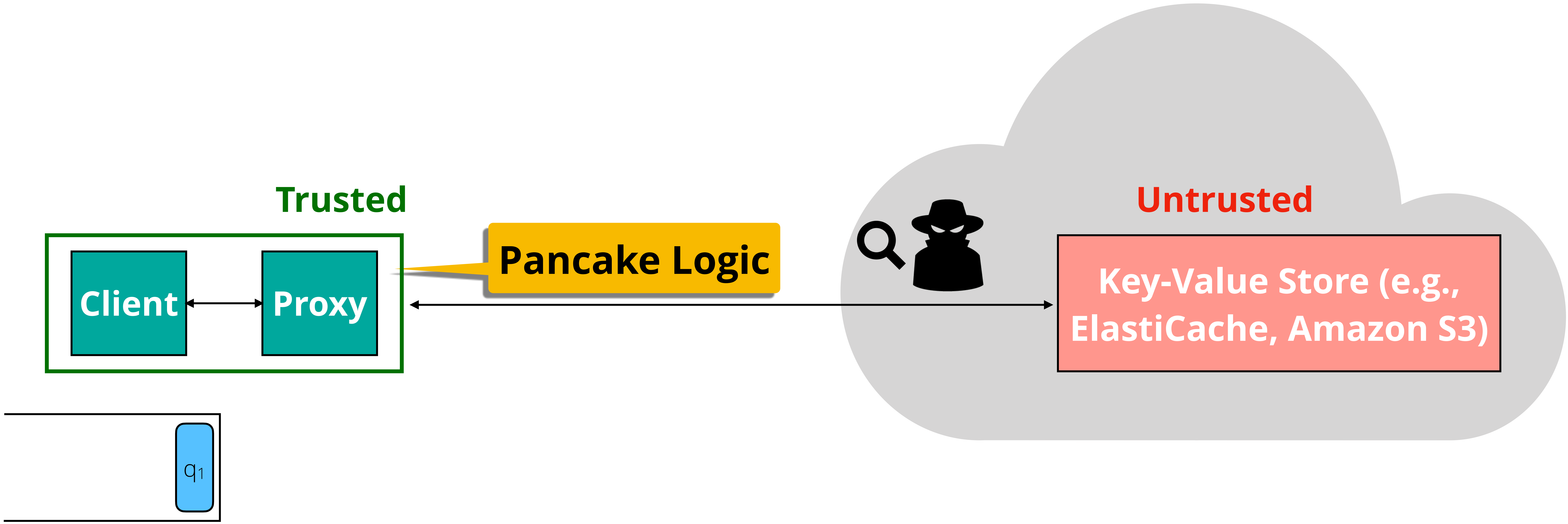


Challenge: How to issue fake+real accesses without revealing which is which?

Approach: Send fixed-size batches of real+fake accesses per query

Pancake

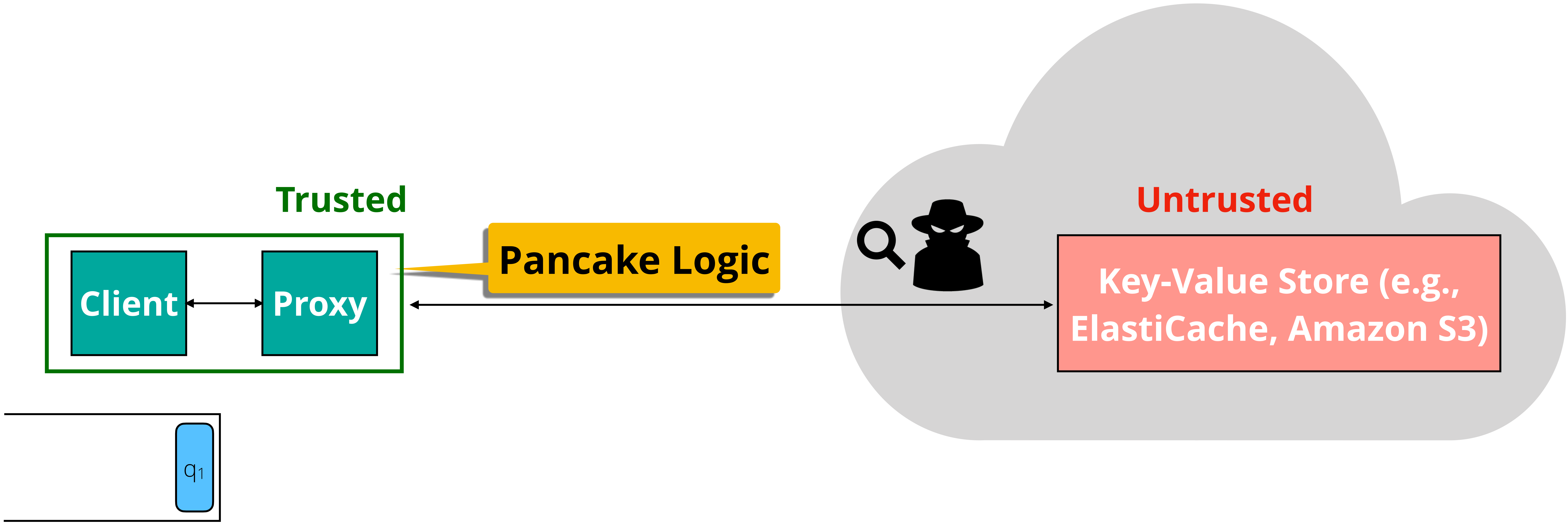
Combine replication and fake accesses!



Every time a new query arrives, enqueue it

Pancake

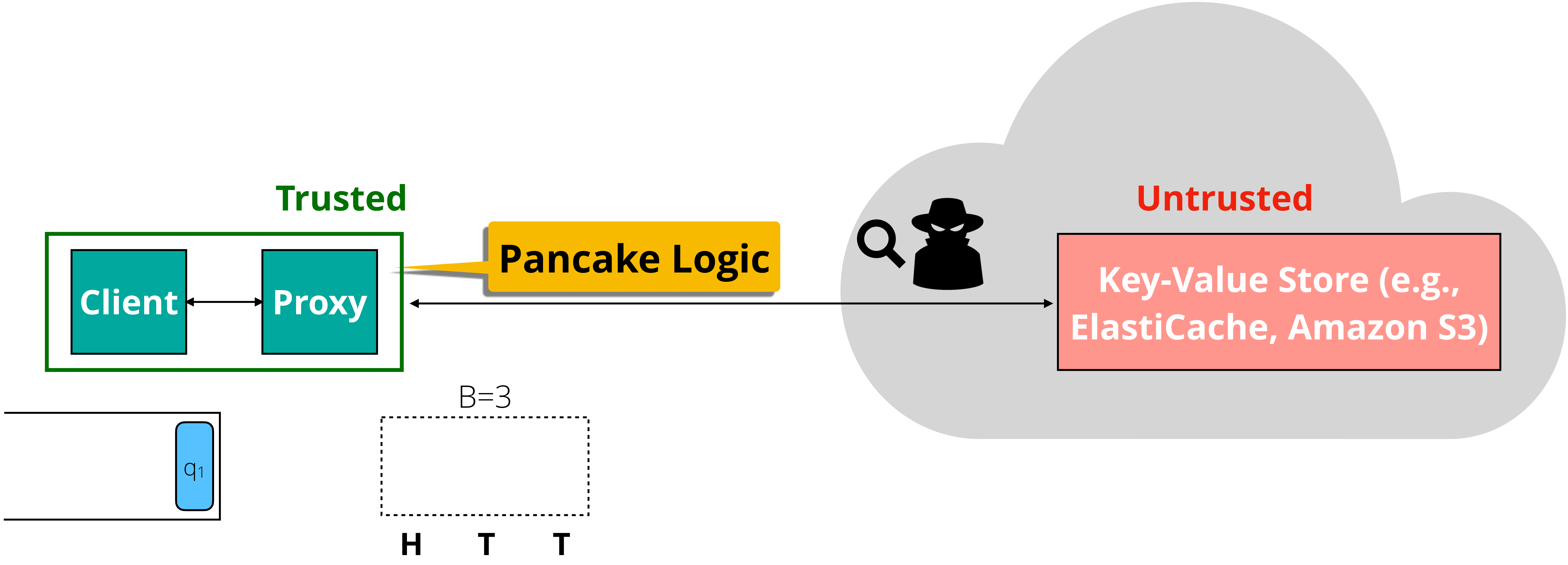
Combine replication and fake accesses!



Every time a new query arrives, enqueue it and flip B coins

Pancake

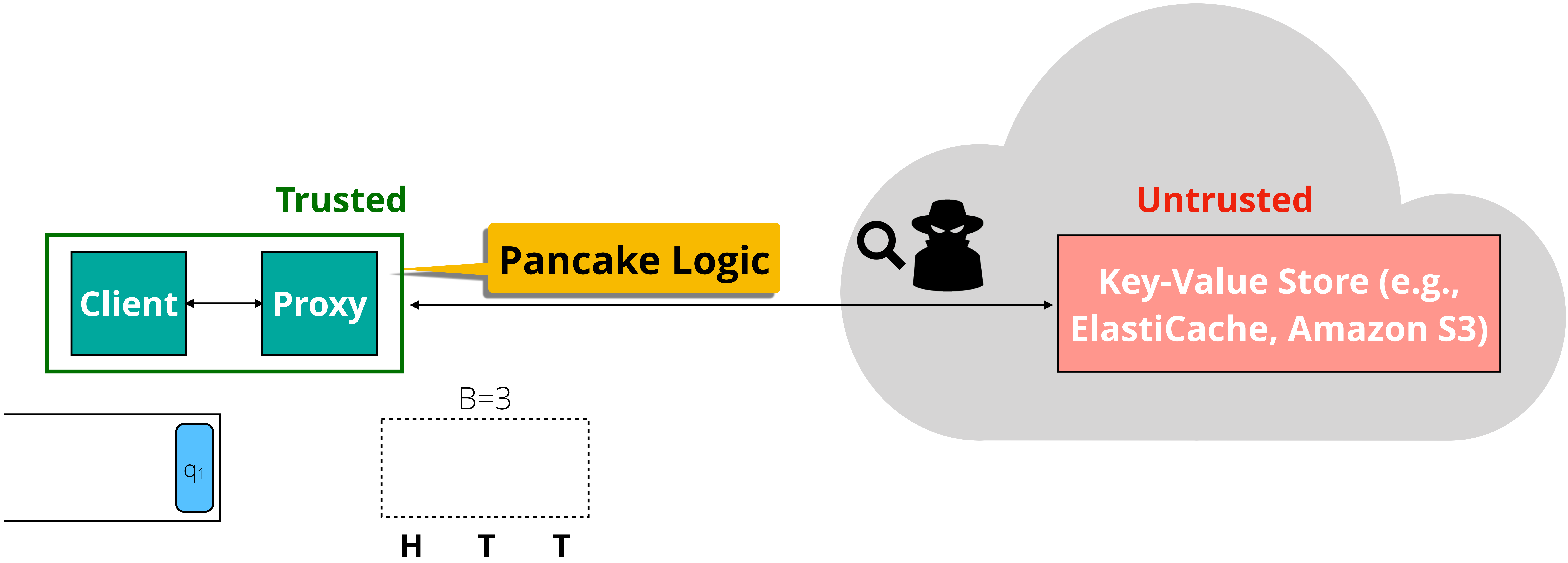
Combine replication and fake accesses!



Every time a new query arrives, enqueue it and flip B coins

Pancake

Combine replication and fake accesses!

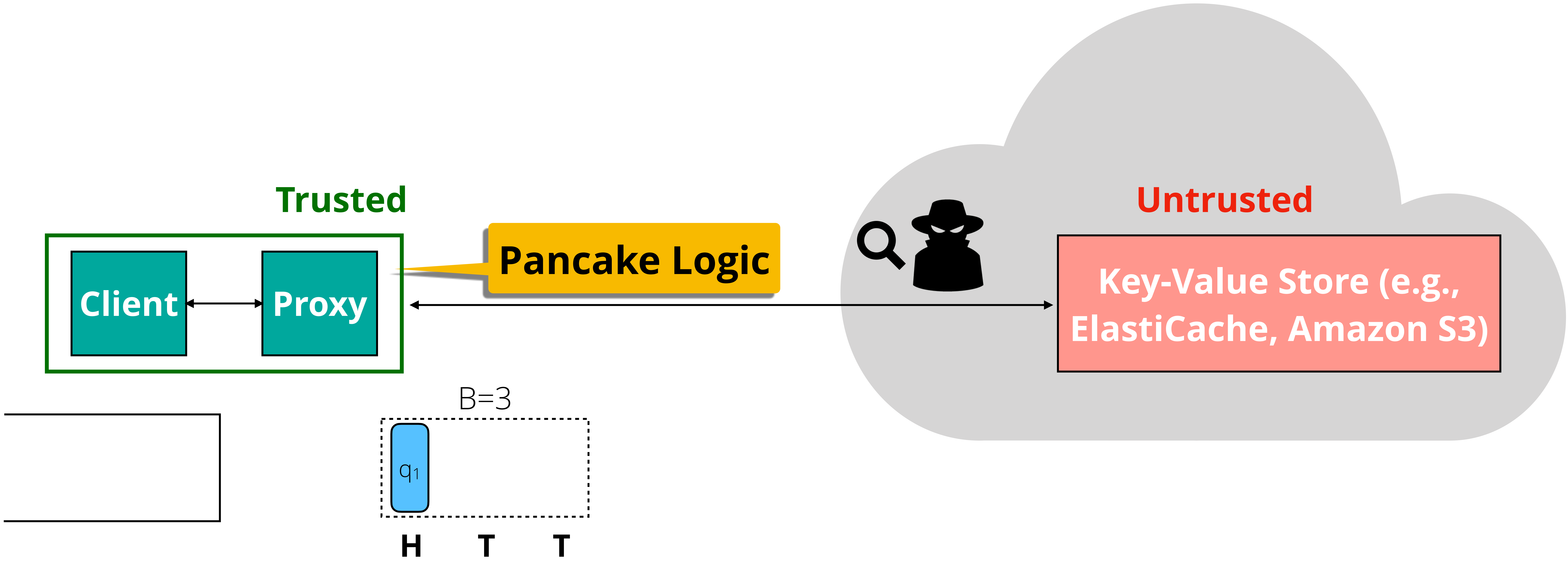


Every time a new query arrives, enqueue it and flip B coins

If heads, dequeue a real query (or draw from π)

Pancake

Combine replication and fake accesses!

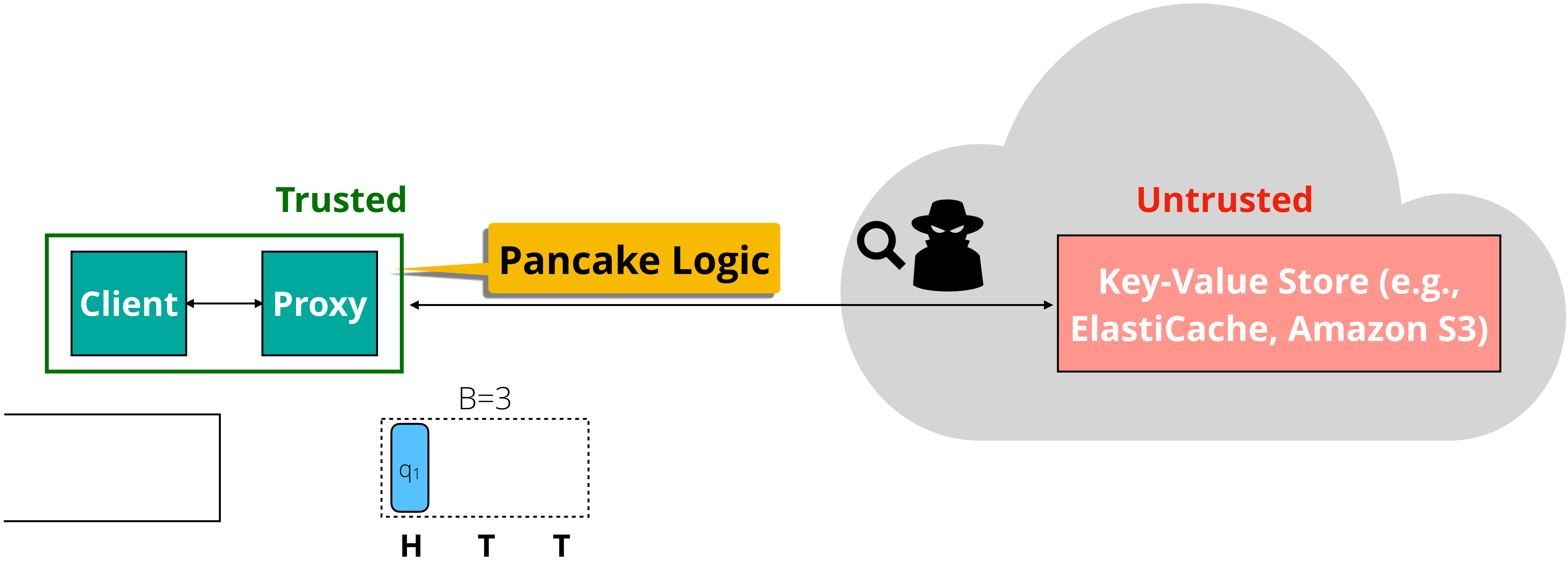


Every time a new query arrives, enqueue it and flip B coins

If heads, dequeue a real query (or draw from π)

Pancake

Combine replication and fake accesses!

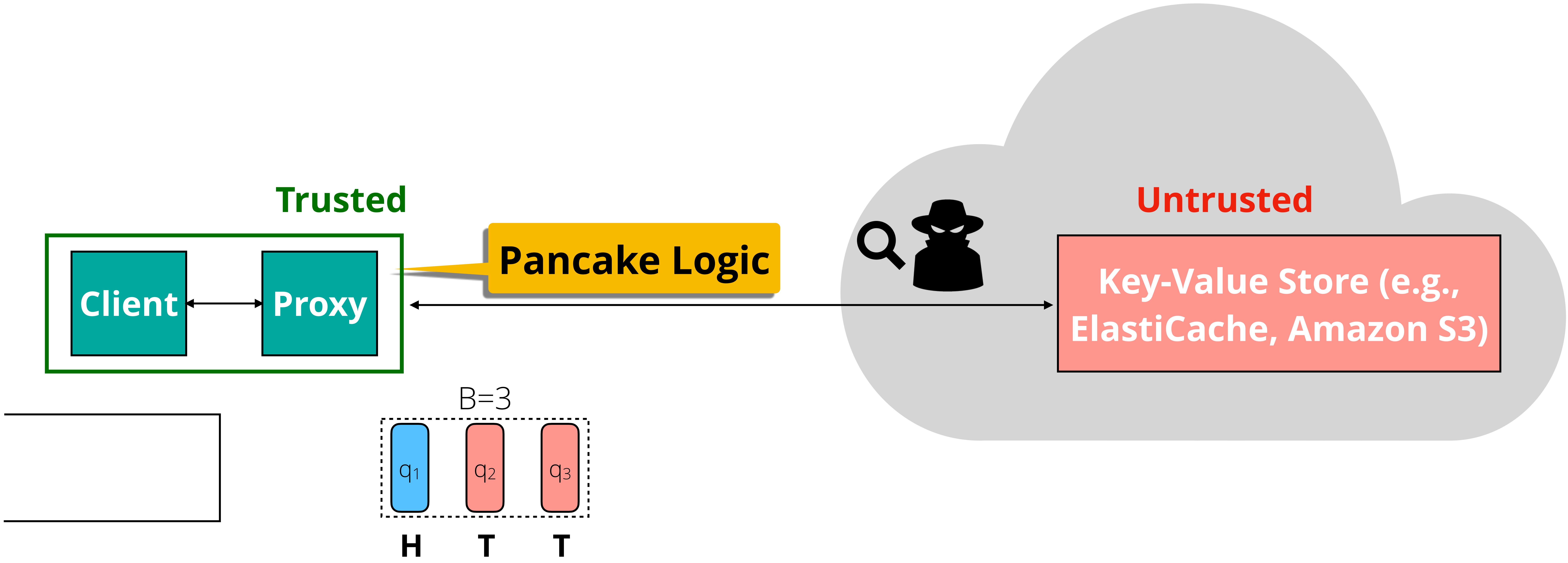


Every time a new query arrives, enqueue it and flip B coins

- If heads, dequeue a real query (or draw from π)
- Else, draw a fake access from π_f

Pancake

Combine replication and fake accesses!



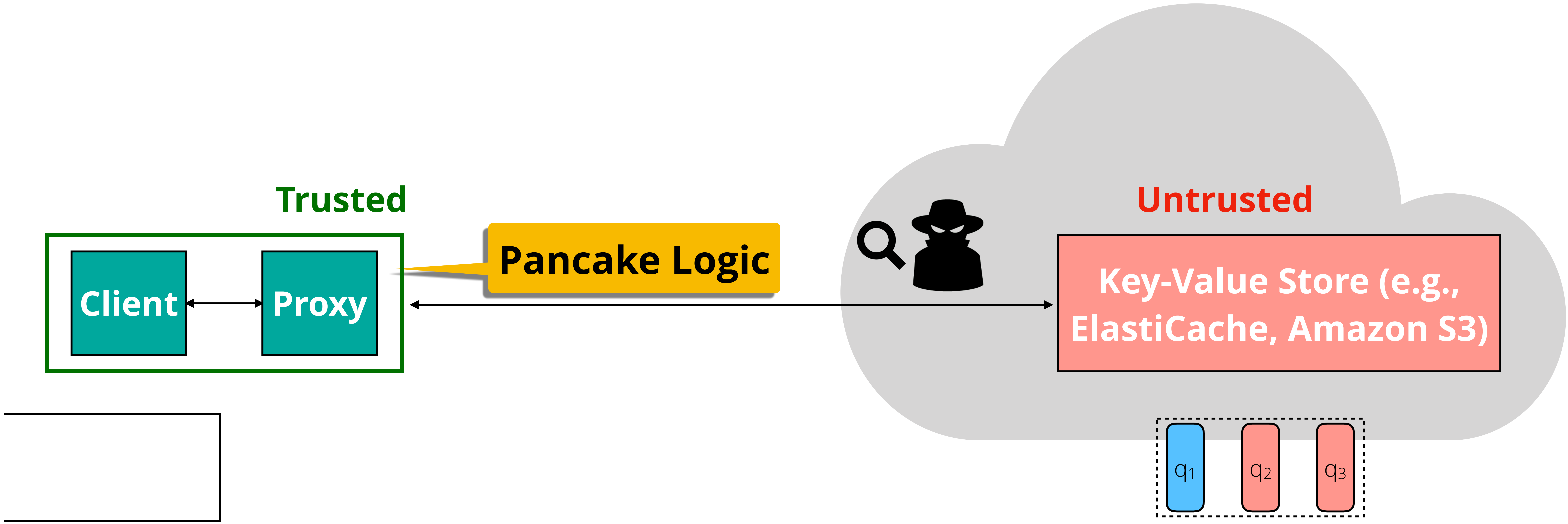
Every time a new query arrives, enqueue it and flip B coins

If heads, dequeue a real query (or draw from π)

Else, draw a fake access from π_f

Pancake

Combine replication and fake accesses!

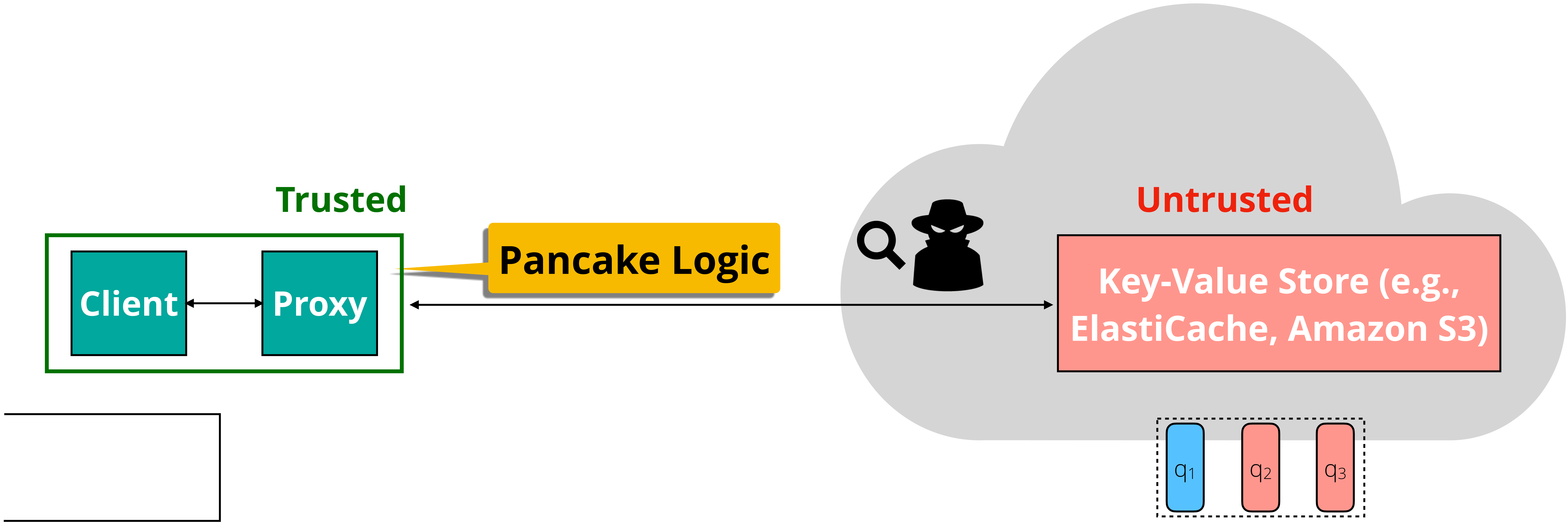


Every time a new query arrives, enqueue it and flip B coins

- If heads, dequeue a real query (or draw from π)
- Else, draw a fake access from π_f

Pancake

Combine replication and fake accesses!



Every time a new query arrives, enqueue it and flip B coins

If heads, dequeue a real query (or draw from π)

Else, draw a fake access from π_f

$3\times$ **bandwidth overhead,** $\leq 2\times$ **storage overhead**

Pancake Security

Pancake Security

Assumptions:

Pancake Security

Assumptions:

- **Persistent passive adversary:** can observe, but not inject or tamper with accesses

Pancake Security

Assumptions:

- **Persistent passive adversary:** can observe, but not inject or tamper with accesses
- Pancake has a reasonable estimate of π

Pancake Security

Assumptions:

- **Persistent passive adversary:** can observe, but not inject or tamper with accesses
- Pancake has a reasonable estimate of π
- Fake & real accesses cannot be distinguished by server (e.g., using timing analysis)

Pancake Security

Assumptions:

- **Persistent passive adversary:** can observe, but not inject or tamper with accesses
- Pancake has a reasonable estimate of π
- Fake & real accesses cannot be distinguished by server (e.g., using timing analysis)

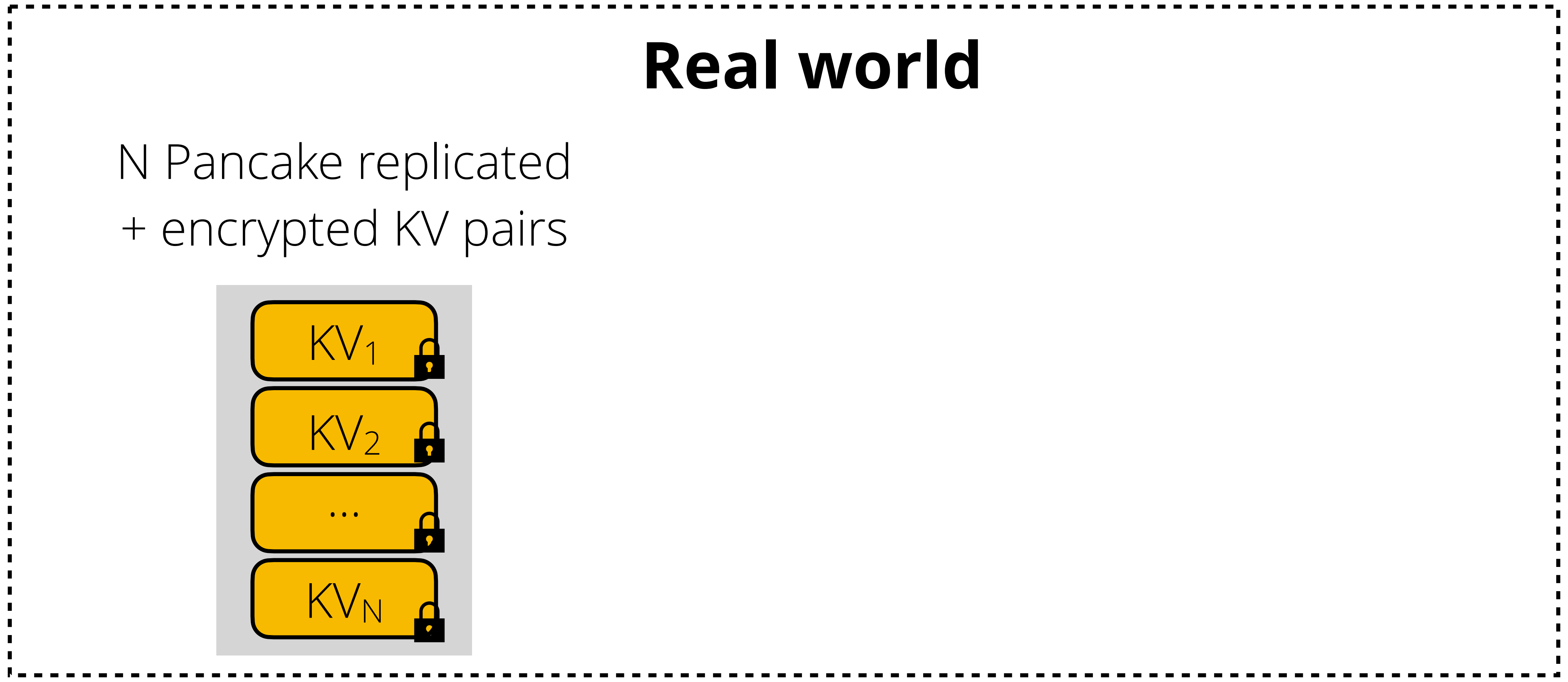
Formal guarantee: Real-versus-random indistinguishability under chosen distribution attack (ROR-CDA)

Pancake Security

Assumptions:

- **Persistent passive adversary:** can observe, but not inject or tamper with accesses
- Pancake has a reasonable estimate of π
- Fake & real accesses cannot be distinguished by server (e.g., using timing analysis)

Formal guarantee: Real-versus-random indistinguishability under chosen distribution attack (ROR-CDA)

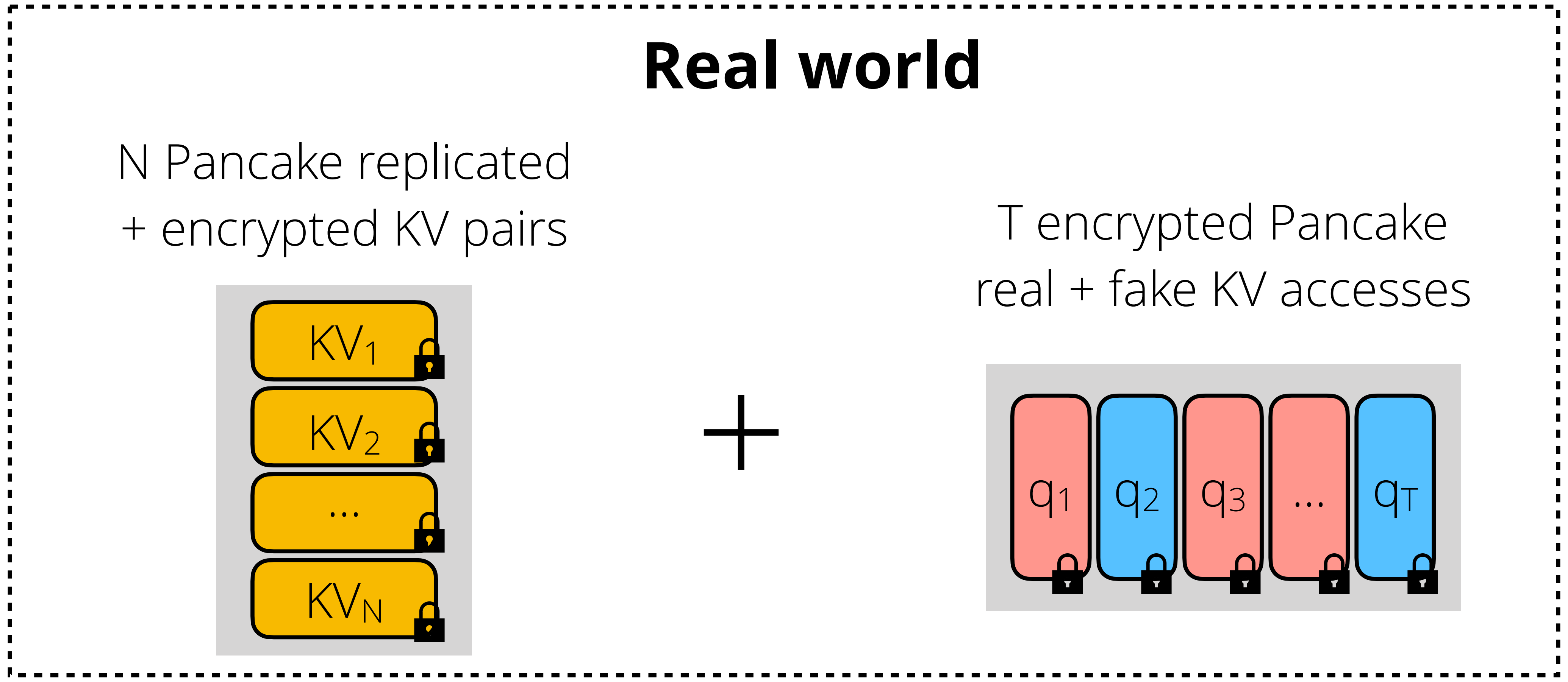


Pancake Security

Assumptions:

- **Persistent passive adversary:** can observe, but not inject or tamper with accesses
- Pancake has a reasonable estimate of π
- Fake & real accesses cannot be distinguished by server (e.g., using timing analysis)

Formal guarantee: Real-versus-random indistinguishability under chosen distribution attack (ROR-CDA)



Pancake Security

Assumptions:

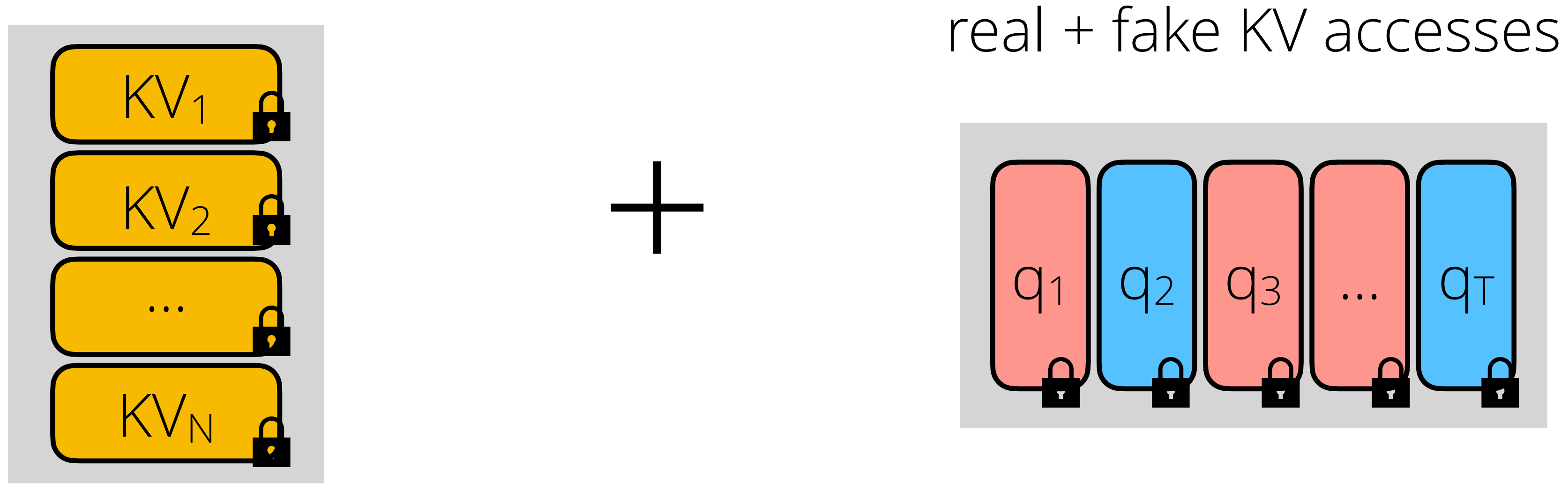
- **Persistent passive adversary:** can observe, but not inject or tamper with accesses
- Pancake has a reasonable estimate of π
- Fake & real accesses cannot be distinguished by server (e.g., using timing analysis)

Formal guarantee: Real-versus-random indistinguishability under chosen distribution attack (ROR-CDA)

Real world

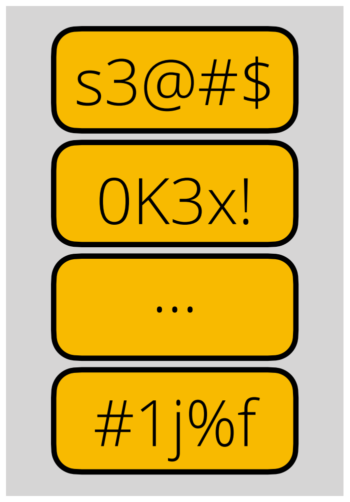
N Pancake replicated + encrypted KV pairs

T encrypted Pancake real + fake KV accesses



Ideal world

N random bit strings

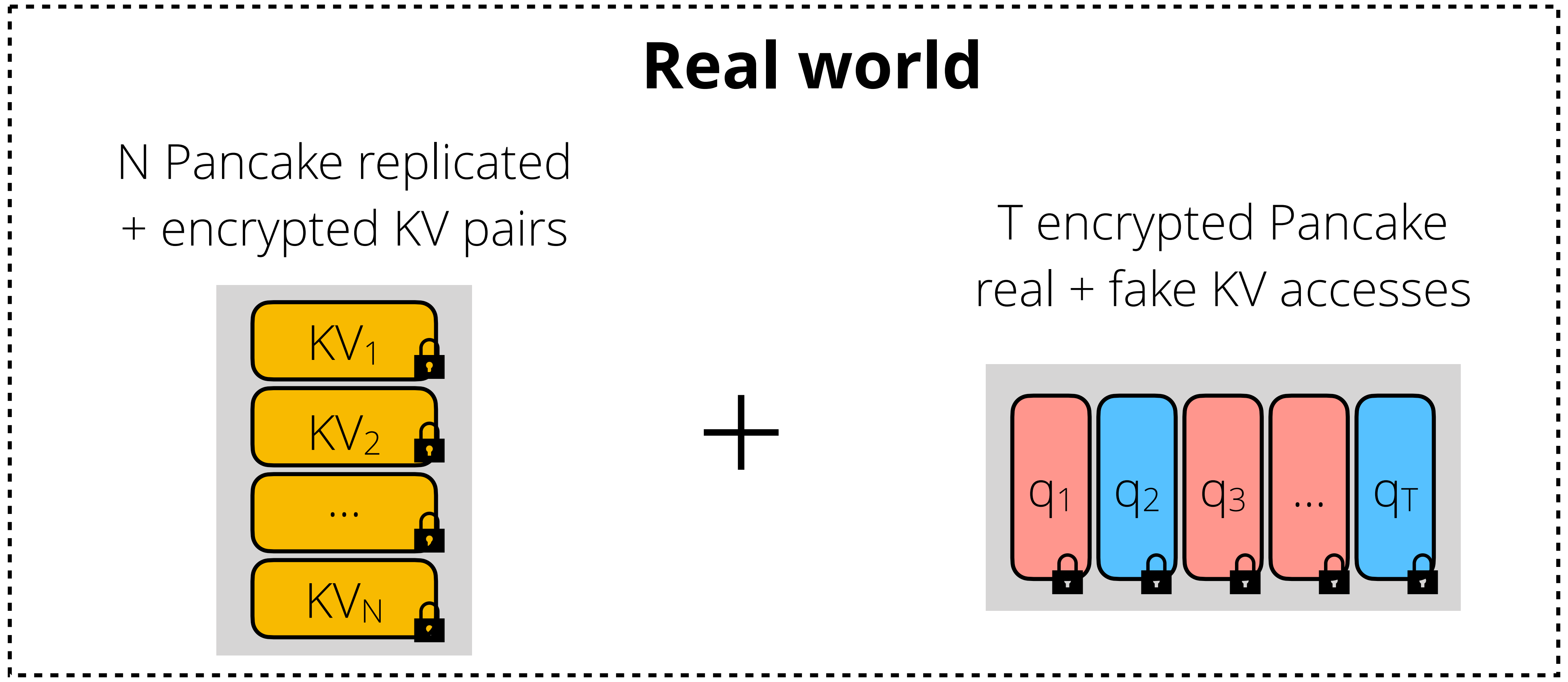


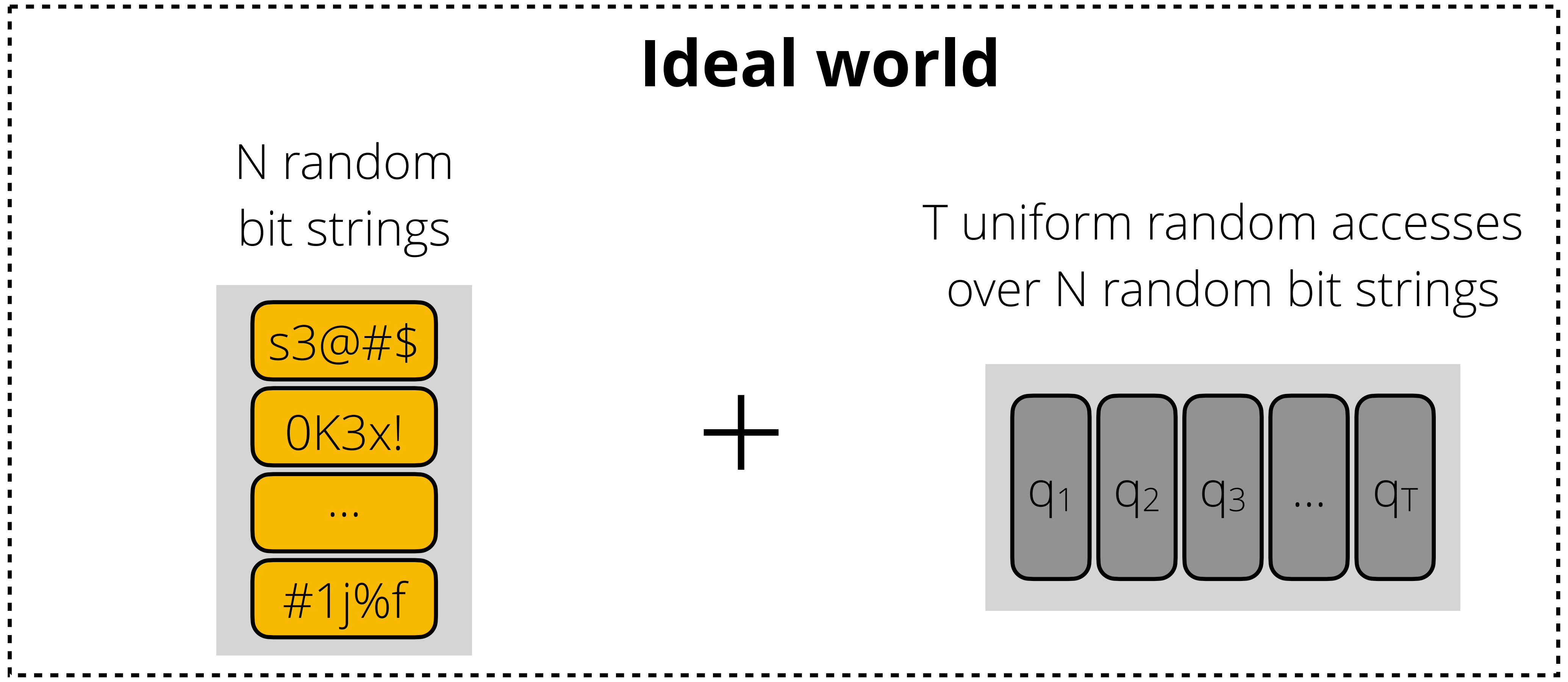
Pancake Security

Assumptions:

- **Persistent passive adversary:** can observe, but not inject or tamper with accesses
- Pancake has a reasonable estimate of π
- Fake & real accesses cannot be distinguished by server (e.g., using timing analysis)

Formal guarantee: Real-versus-random indistinguishability under chosen distribution attack (ROR-CDA)



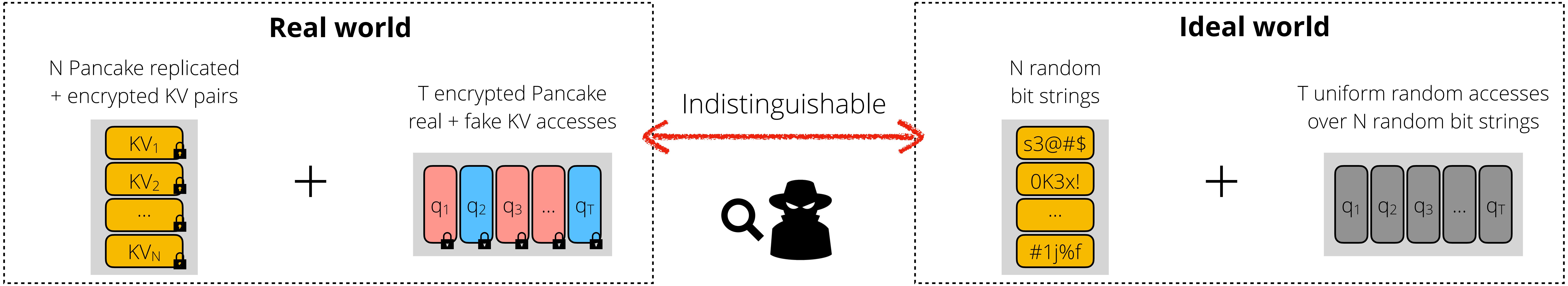


Pancake Security

Assumptions:

- **Persistent passive adversary:** can observe, but not inject or tamper with accesses
- Pancake has a reasonable estimate of π
- Fake & real accesses cannot be distinguished by server (e.g., using timing analysis)

Formal guarantee: Real-versus-random indistinguishability under chosen distribution attack (ROR-CDA)



Pancake: Additional Challenges

Pancake: Additional Challenges

Update KV pair with replicas?

Buffer updates to KV replicas until next access

Pancake: Additional Challenges

Update KV pair with replicas?	**Buffer updates to KV replicas until next access**
Dynamic access patterns?	**Adjust fake distribution & reassign replicas**

Pancake: Additional Challenges

Update KV pair with replicas?	**Buffer updates to KV replicas until next access**
Dynamic access patterns?	**Adjust fake distribution & reassign replicas**
Estimate access distribution, detect distribution changes?	**Sliding-window histograms, two-sample KS test**

Pancake: Additional Challenges

Update KV pair with replicas?	**Buffer updates to KV replicas until next access**
Dynamic access patterns?	**Adjust fake distribution & reassign replicas**
Estimate access distribution, detect distribution changes?	**Sliding-window histograms, two-sample KS test**

Details in the paper!

Pancake Performance

Pancake Performance

Cloud Storage: Redis on t3.2xlarge Amazon EC2 instances, **Client/Proxy:** Amazon EC2 r4.8xlarge instances

Pancake Performance

Cloud Storage: Redis on t3.2xlarge Amazon EC2 instances, **Client/Proxy:** Amazon EC2 r4.8xlarge instances
Dataset: 10^6 x 1KB key-value pairs, **Workload:** YCSB Workload A (50% reads + 50% writes)

Pancake Performance

Cloud Storage: Redis on t3.2xlarge Amazon EC2 instances, **Client/Proxy:** Amazon EC2 r4.8xlarge instances
Dataset: 10^6 x 1KB key-value pairs, **Workload:** YCSB Workload A (50% reads + 50% writes)

Approach →	Insecure Baseline	PathORAM	Pancake
Server Storage	1 GB	8 GB	2 GB
Proxy Storage	0 GB	8 MB	24 MB

Takeaways

Server storage **4x lower** than PathORAM, low proxy storage (~1% of server storage)

Pancake Performance

Cloud Storage: Redis on t3.2xlarge Amazon EC2 instances, **Client/Proxy:** Amazon EC2 r4.8xlarge instances
Dataset: 10^6 x 1KB key-value pairs, **Workload:** YCSB Workload A (50% reads + 50% writes)

Approach →	Insecure Baseline	PathORAM	Pancake
Server Storage	1 GB	8 GB	2 GB
Proxy Storage	0 GB	8 MB	24 MB
Latency	1.15 ms	31.32 ms	2.61 ms
Throughput	50,990 Op/s	32 Op/s	6,718 Op/s

Takeaways

Server storage **4x lower** than PathORAM, low proxy storage (~1% of server storage)
Throughput **220x higher** and latency **12x lower** than PathORAM

Pancake Performance

Cloud Storage: Redis on t3.2xlarge Amazon EC2 instances, **Client/Proxy:** Amazon EC2 r4.8xlarge instances
Dataset: 10^6 x 1KB key-value pairs, **Workload:** YCSB Workload A (50% reads + 50% writes)

Approach ⟶	Insecure Baseline	PathORAM	Pancake
Server Storage	1 GB	8 GB	2 GB
Proxy Storage	0 GB	8 MB	24 MB
Latency	1.15 ms	31.32 ms	2.61 ms
Throughput	50,990 Op/s	32 Op/s	6,718 Op/s

Takeaways

Server storage **4x lower** than PathORAM, low proxy storage (~1% of server storage)
Throughput **220x higher** and latency **12x lower** than PathORAM

Many more results in the paper!

Summary

- **Pancake:** first system that protects data stores against access pattern attacks at **constant factor server storage & bandwidth overheads**
- **Formal security analysis** showing passive persistent security
- Comprehensive evaluation shows **throughput > 2 orders of magnitude higher than state-of-the art** (PathORAM)!

Summary

- **Pancake:** first system that protects data stores against access pattern attacks at **constant factor server storage & bandwidth overheads**
- **Formal security analysis** showing passive persistent security
- Comprehensive evaluation shows **throughput > 2 orders of magnitude higher than state-of-the art** (PathORAM)!

Thank You!

Questions?