Proof concurrency in *Aardvark*

Derek Leung
MIT CSAIL

Yossi Gilad
Hebrew University of Jerusalem

Sergey Gorbunov
University of Waterloo

Leonid Reyzin
Boston University

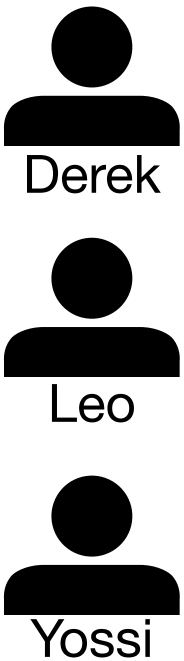
Nickolai Zeldovich
MIT CSAIL

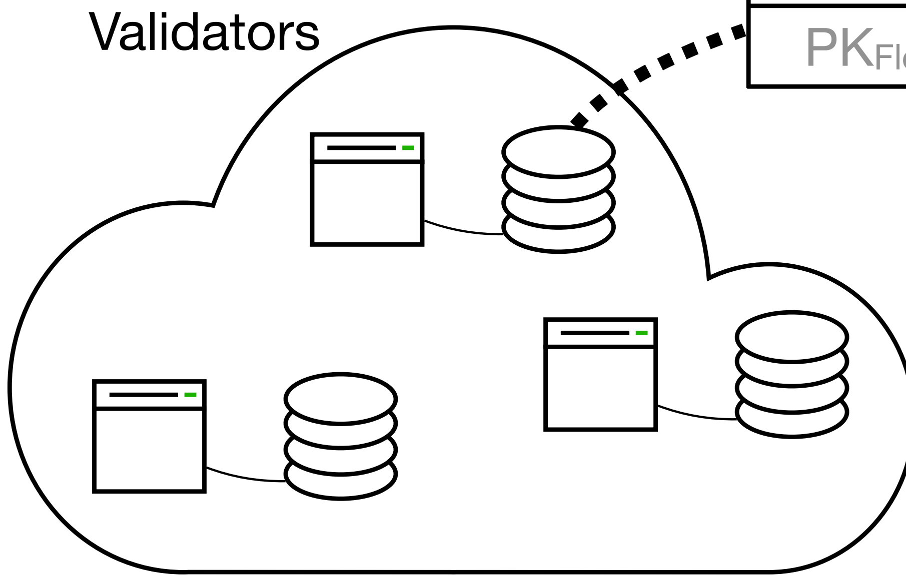
Aardvark: An Asynchronous Authenticated Dictionary with Applications to Account-based Cryptocurrencies

Part of work done at Algorand, Inc.

Motivation

Cryptocurrency

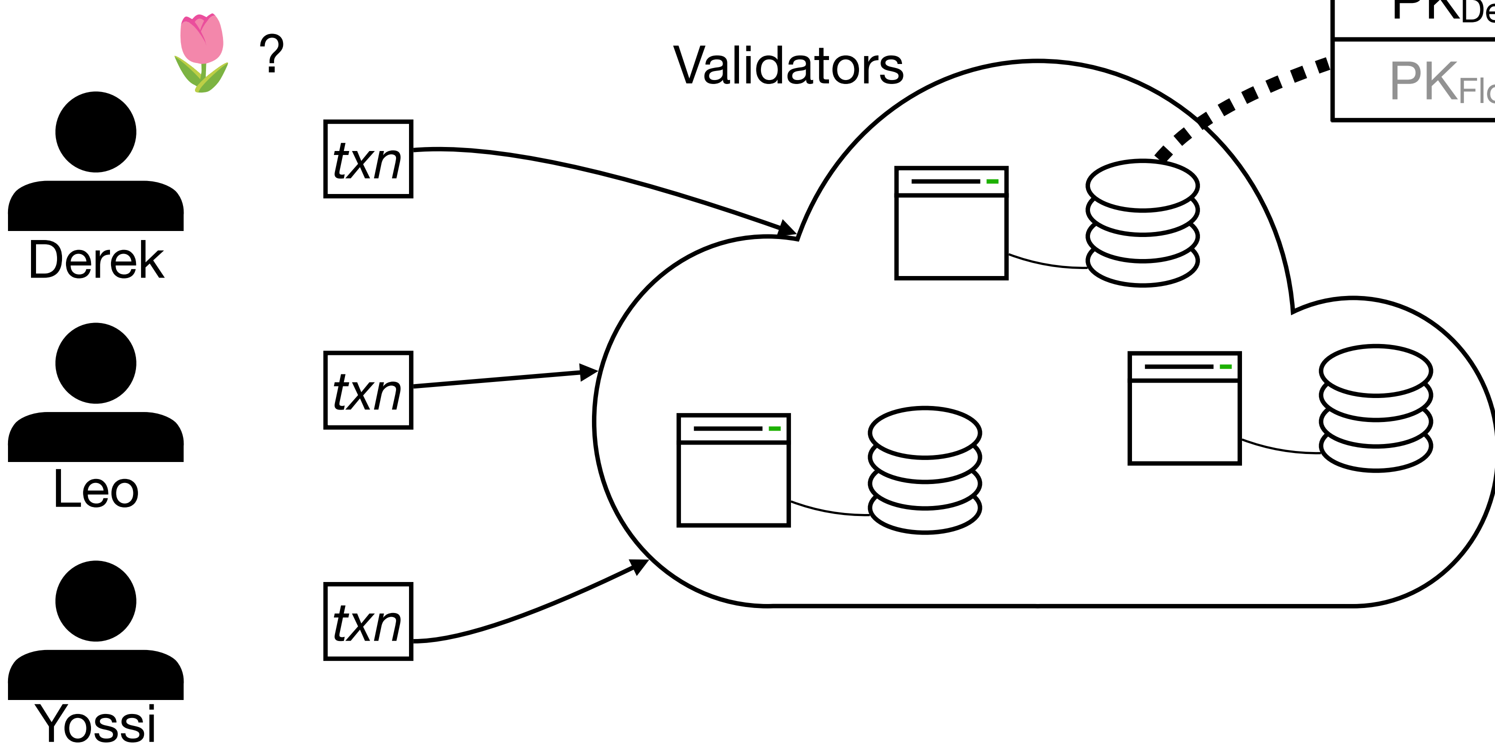




PK_{Leo}	\$100
PK_{Derek}	\$75
$PK_{Florist}$	\$0

Motivation

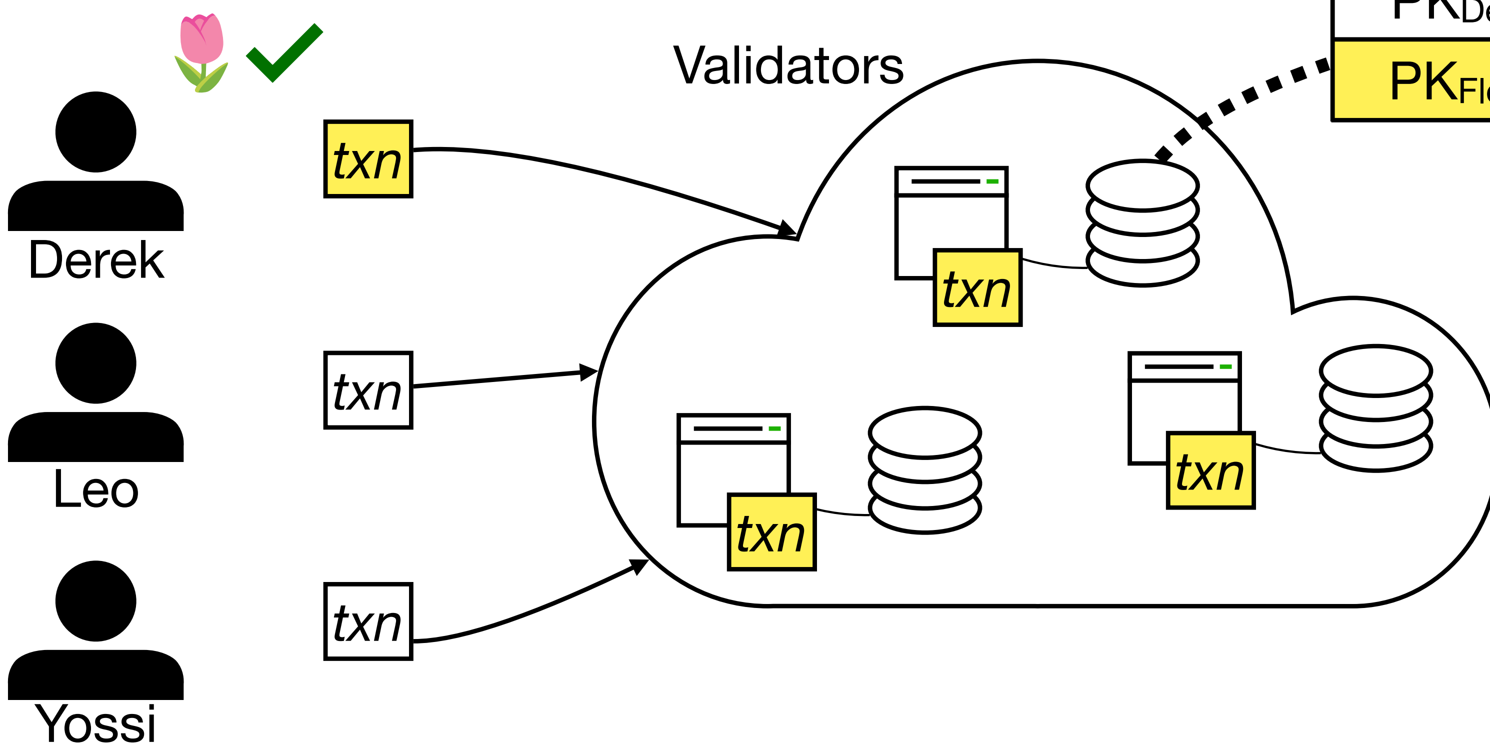
Cryptocurrency



PK_{Leo}	\$100
PK_{Derek}	\$75
$PK_{Florist}$	\$0

Motivation

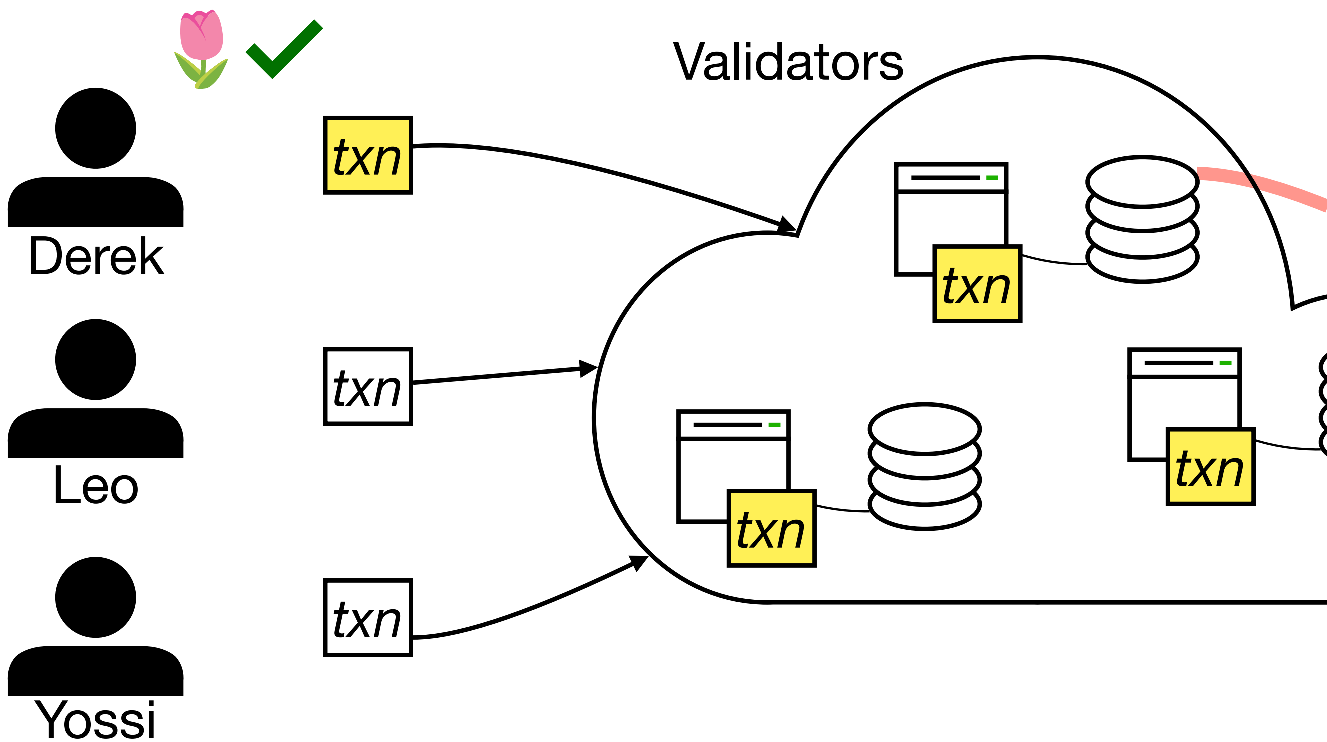
Cryptocurrency



PK_{Leo}	\$100
PK_{Derek}	\$72
$PK_{Florist}$	\$3

Motivation

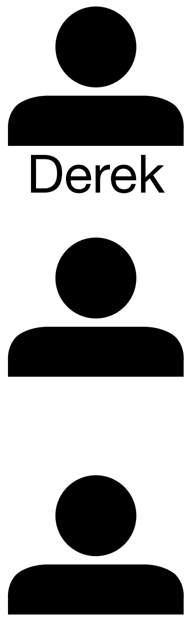
Large state

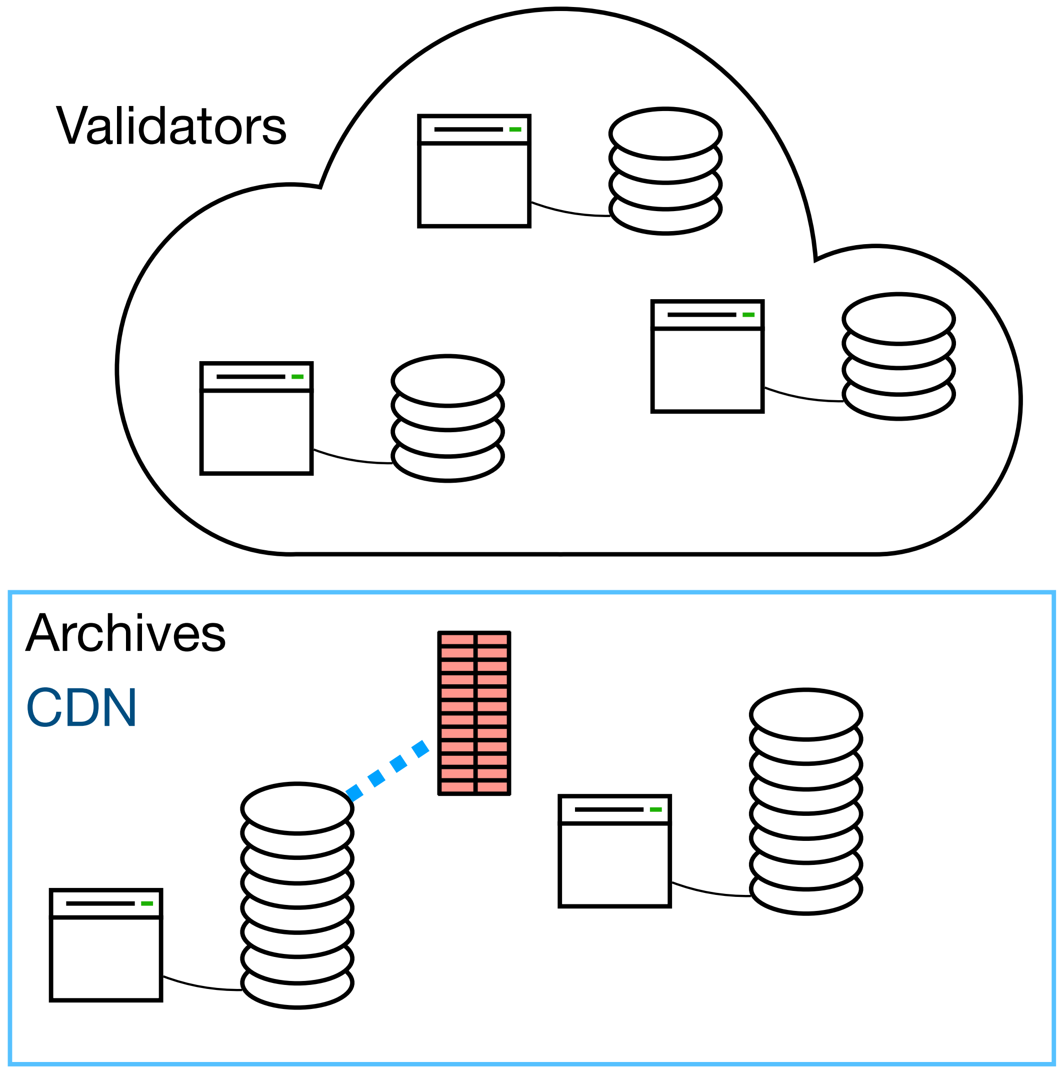


PK_{Leo}	$100
PK_{Derek}	$72
$PK_{Florist}$	$3
…	…
…	…
…	…
…	…
…	…
…	…
…	…
…	…
…	…

Motivation

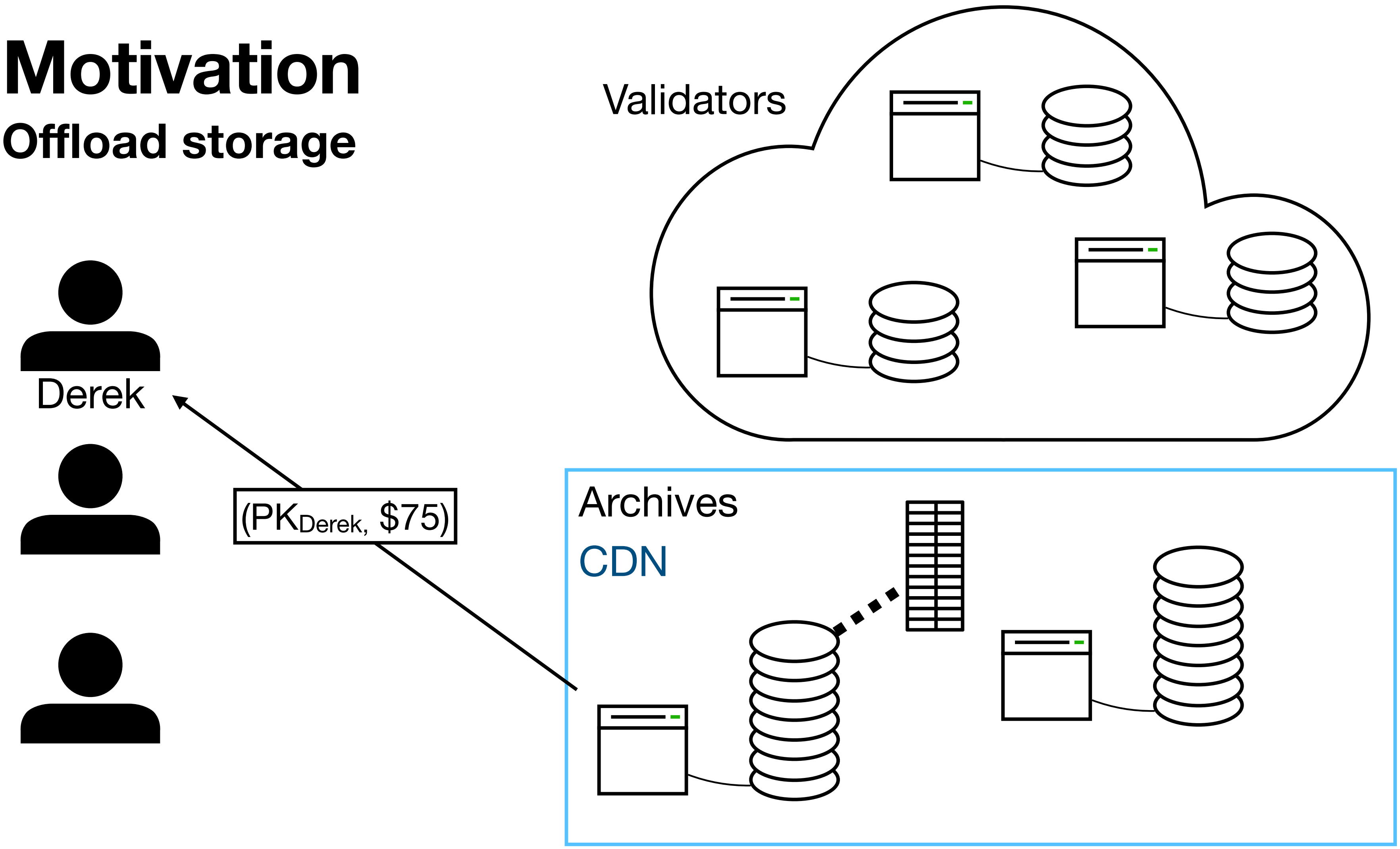
Offload storage





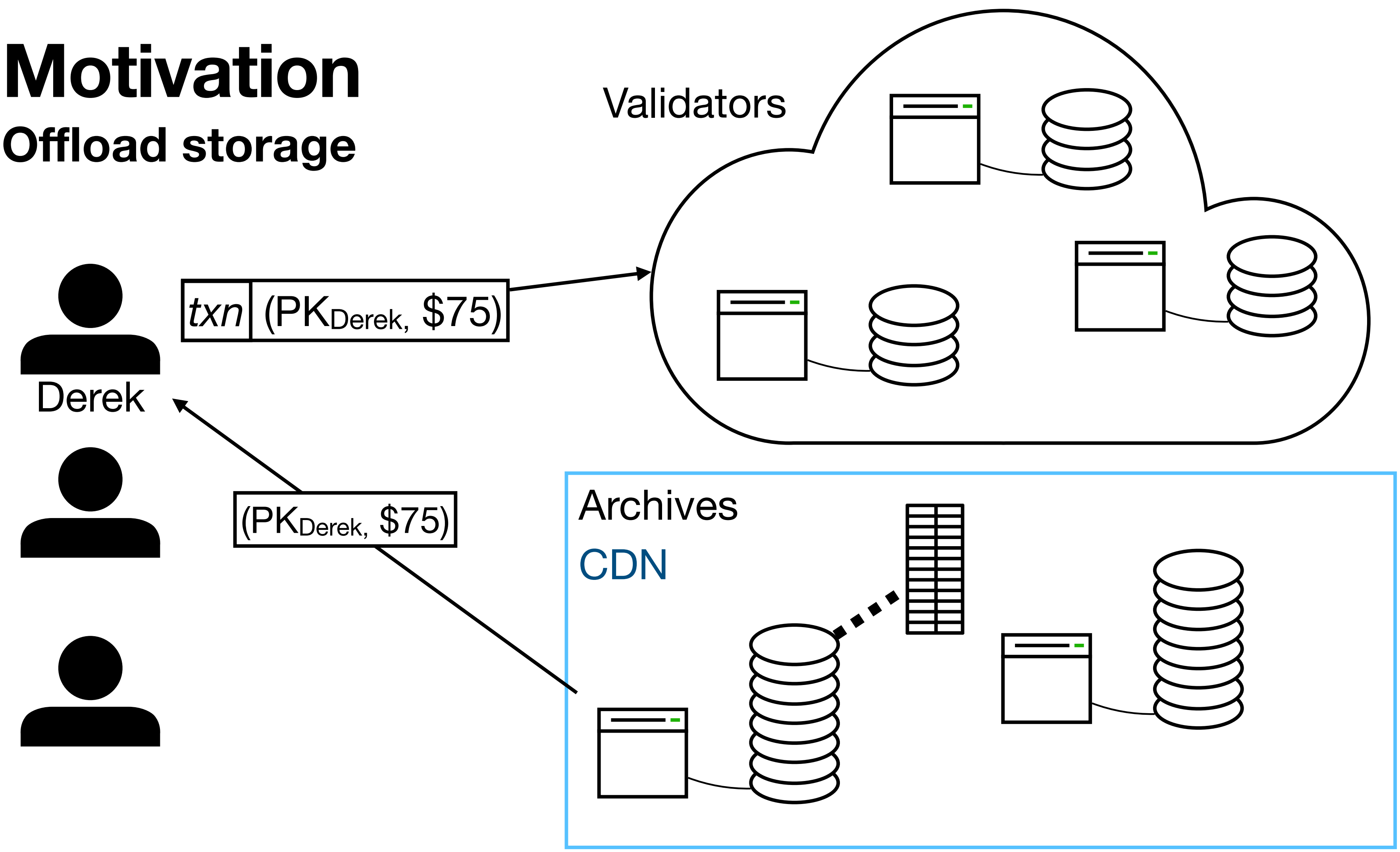
Motivation

Offload storage



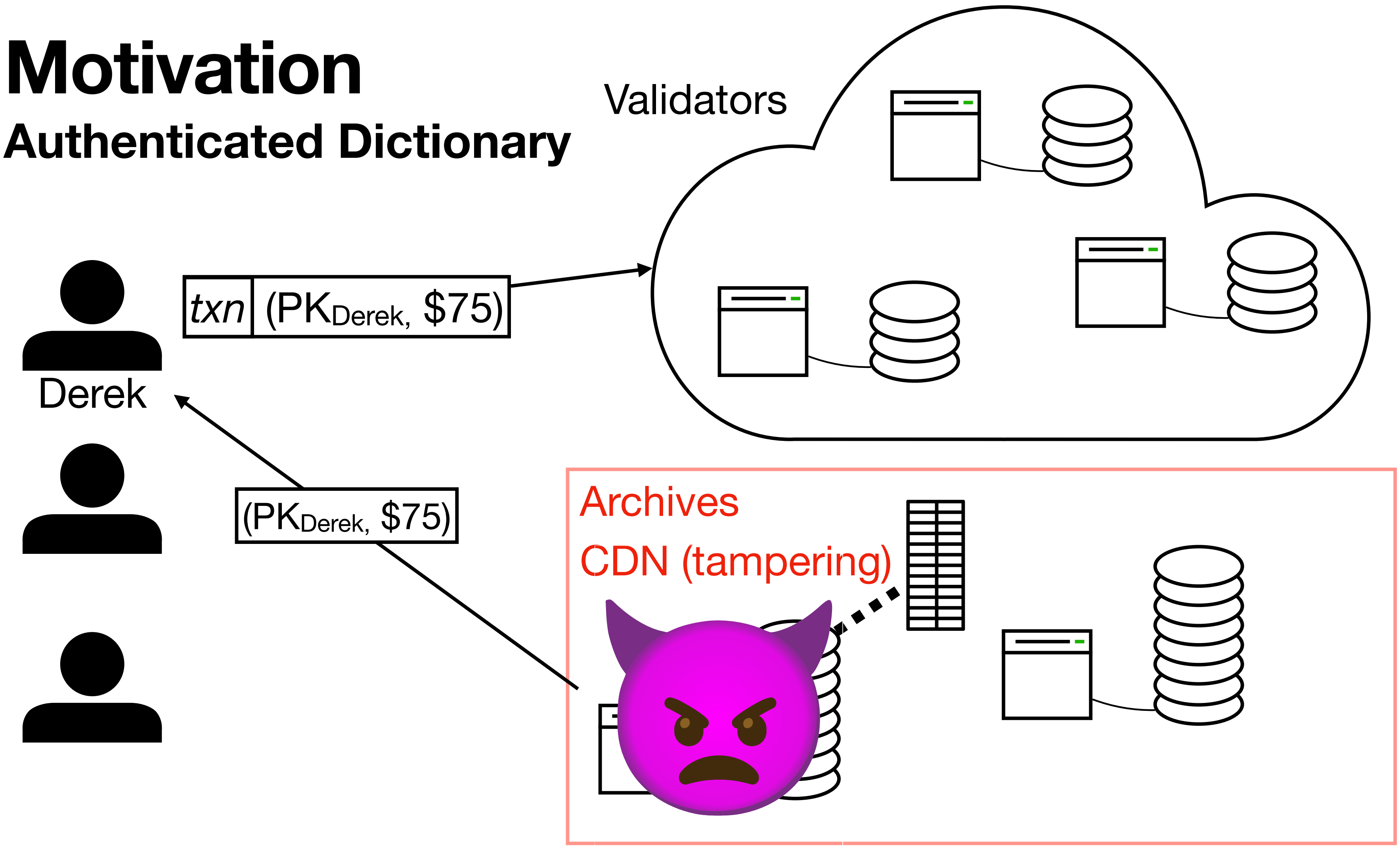
Motivation

Offload storage



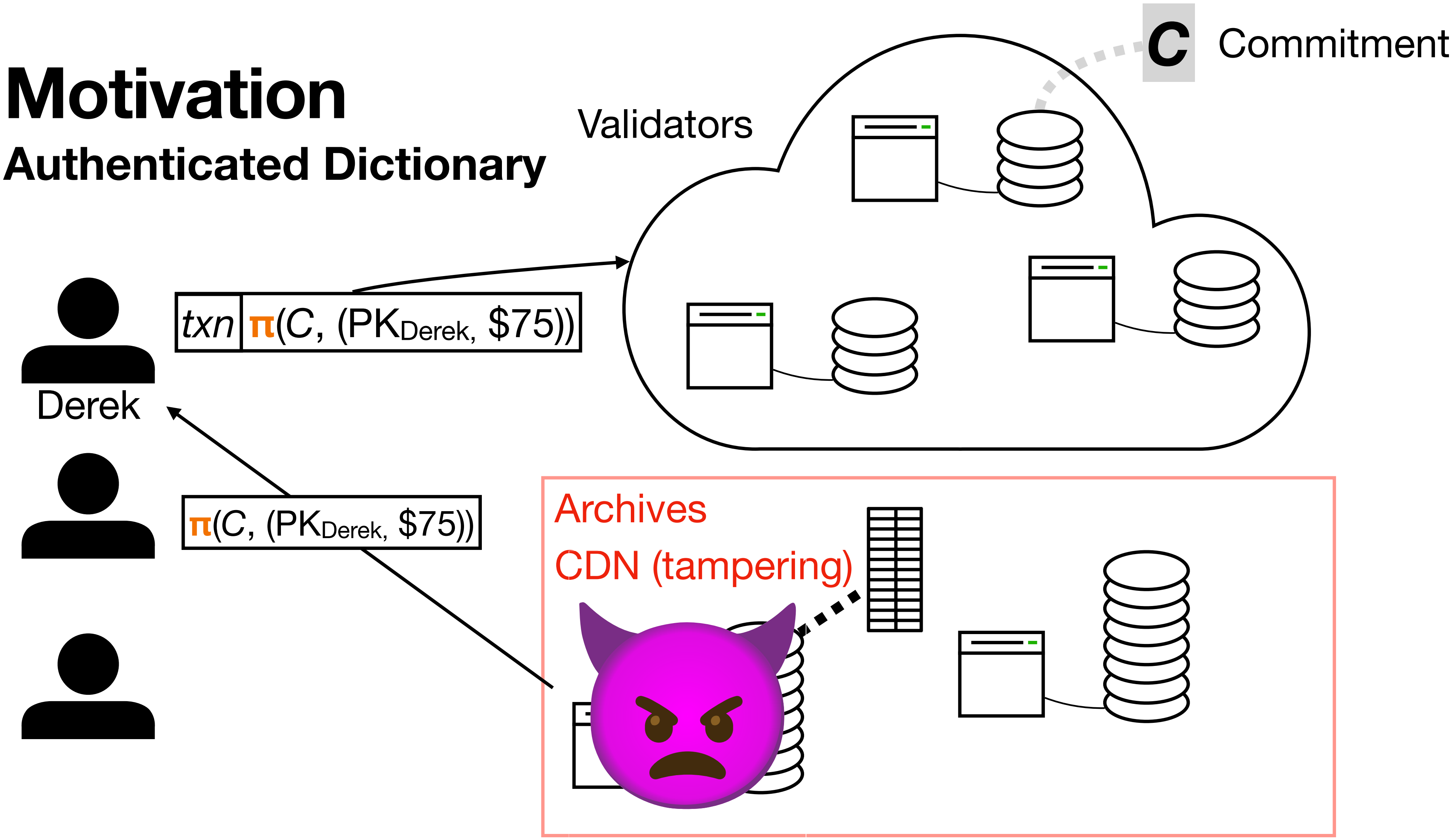
Motivation

Authenticated Dictionary



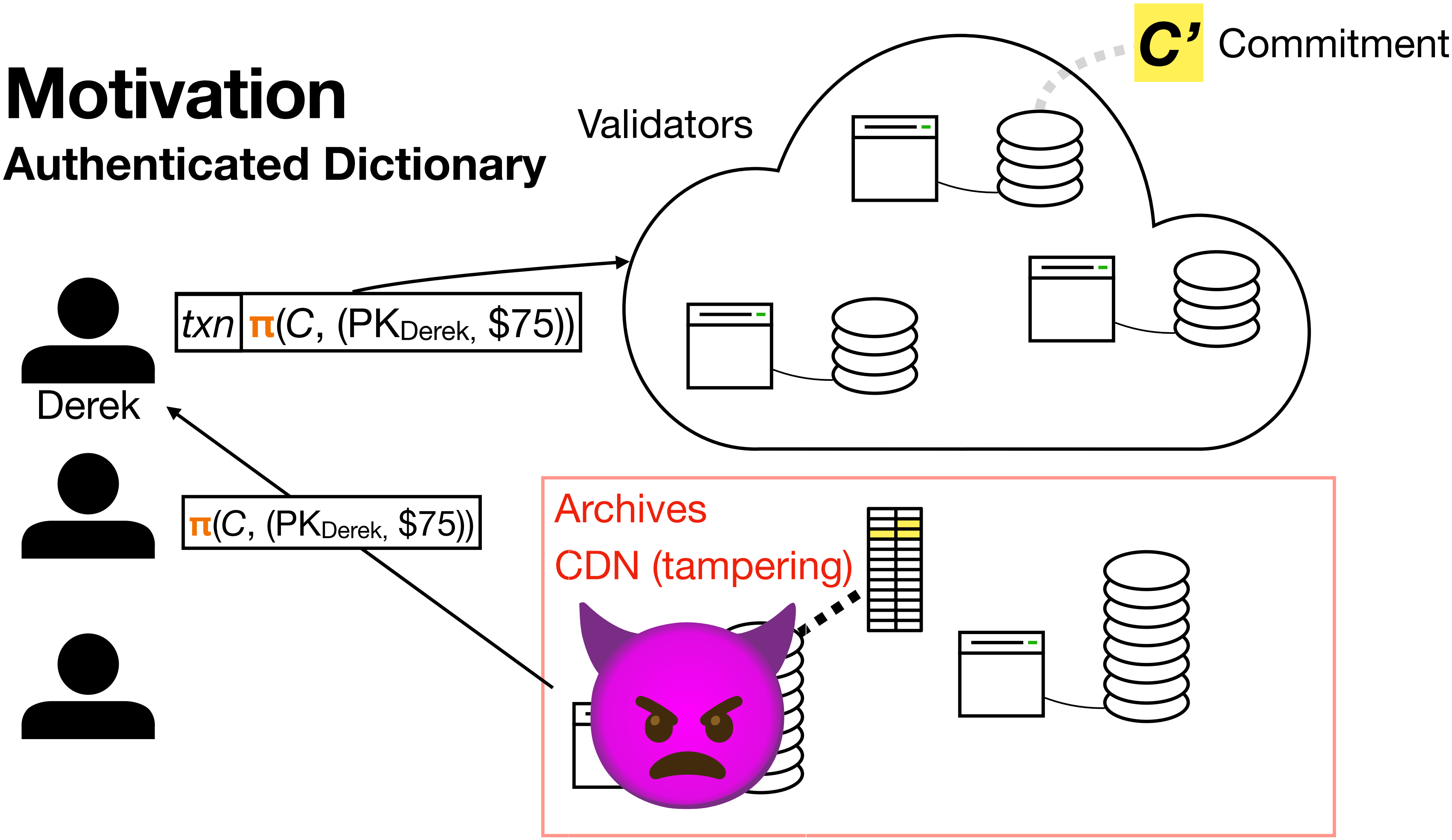
Motivation

Authenticated Dictionary



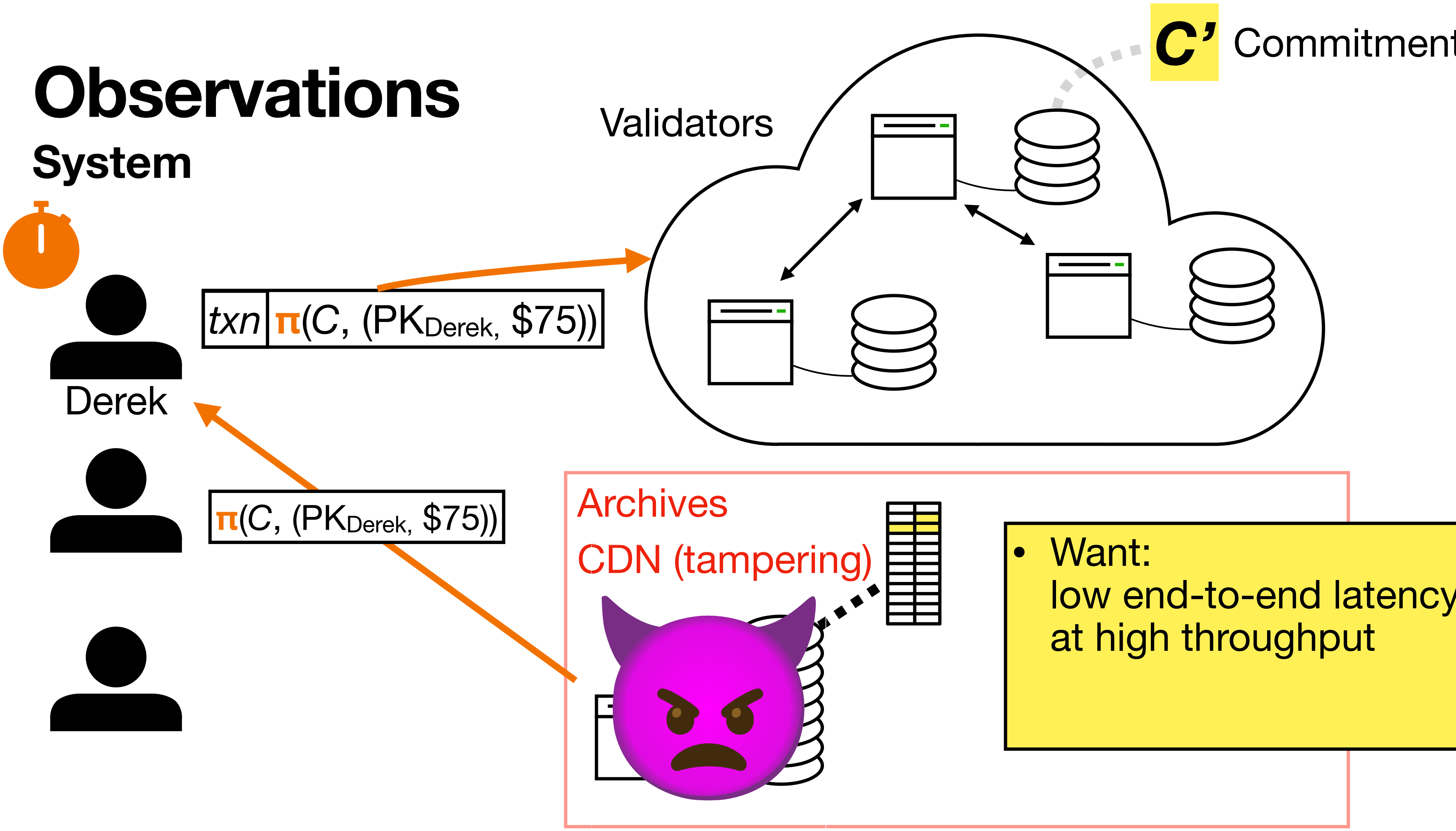
Motivation

Authenticated Dictionary



Observations

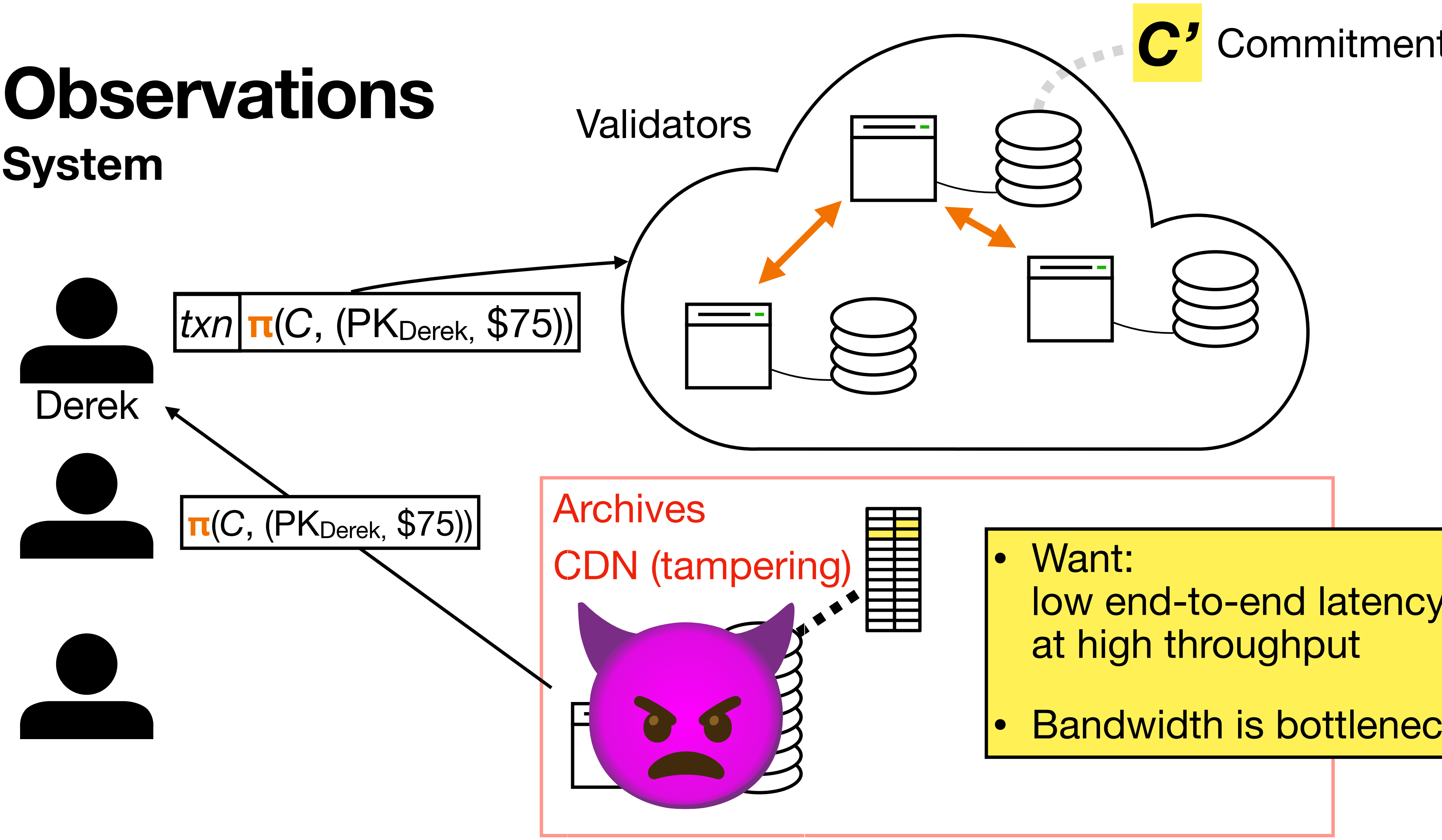
System



- Want:
  low end-to-end latency
  at high throughput

Observations

System



- Want:
  low end-to-end latency
  at high throughput
- Bandwidth is bottleneck

Requirements

Authenticated Dictionary

Proof overhead

Compute (vs. signature) | Transmit (bandwidth)

Requirements

Authenticated Dictionary

	Proof overhead	
	Compute (vs. signature)	Transmit (bandwidth)
Hash-based (e.g., Merkle Tree)	Almost free	10x overhead

Requirements

Authenticated Dictionary

	Proof overhead	
	Compute (vs. signature)	Transmit (bandwidth)
Hash-based (e.g., Merkle Tree)	Almost free	10x overhead
Pairing-based (e.g., Vector Commitment)	Substantial	~1x overhead

Requirements

Authenticated Dictionary

	Proof overhead	
	Compute (vs. signature)	Transmit (bandwidth)
Hash-based (e.g., Merkle Tree)	Almost free	10x overhead
Pairing-based (e.g., Vector Commitment)	Substantial	~1x overhead
???	Almost free?	~1x overhead?

Requirements

Authenticated Dictionary

	Proof overhead	
	Compute (vs. signature)	Transmit (bandwidth)
Hash-based (e.g., Merkle Tree)	Almost free	10x overhead
Pairing-based (e.g., Vector Commitment)	Substantial	~1x overhead
Pairing-based + Versioning	~~Almost~~ Not quite free: **Enable parallelism**	1—2x overhead

Aardvark contributions

An authenticated dictionary with

- Short proofs + commitments

    Built from vector commitments, via short proofs of nonmembership

Aardvark contributions

An authenticated dictionary with

- Short proofs + commitments

  Built from vector commitments, via short proofs of nonmembership

- Low latency at high throughput

  Using transactional concurrency control, via versioning state

This talk

An authenticated dictionary with

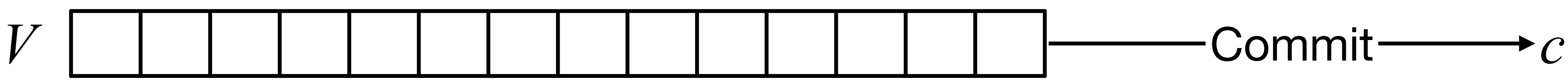
- Short proofs + commitments

  Built from vector commitments, via short proofs of nonmembership

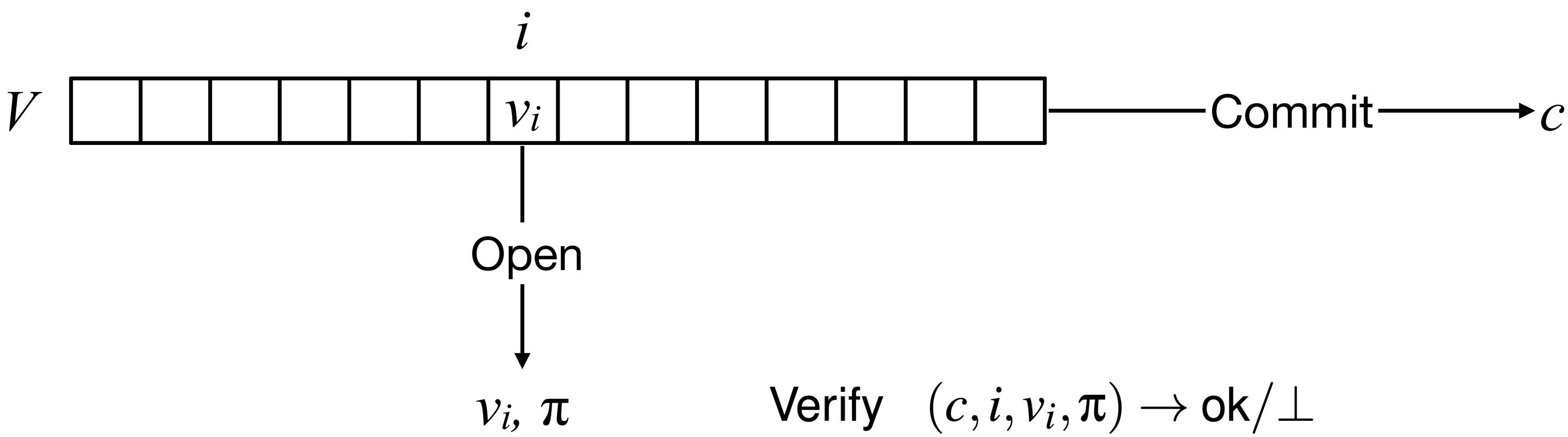
- Low latency at high throughput

  Using transactional concurrency control, via versioning state

Background: Vector commitments

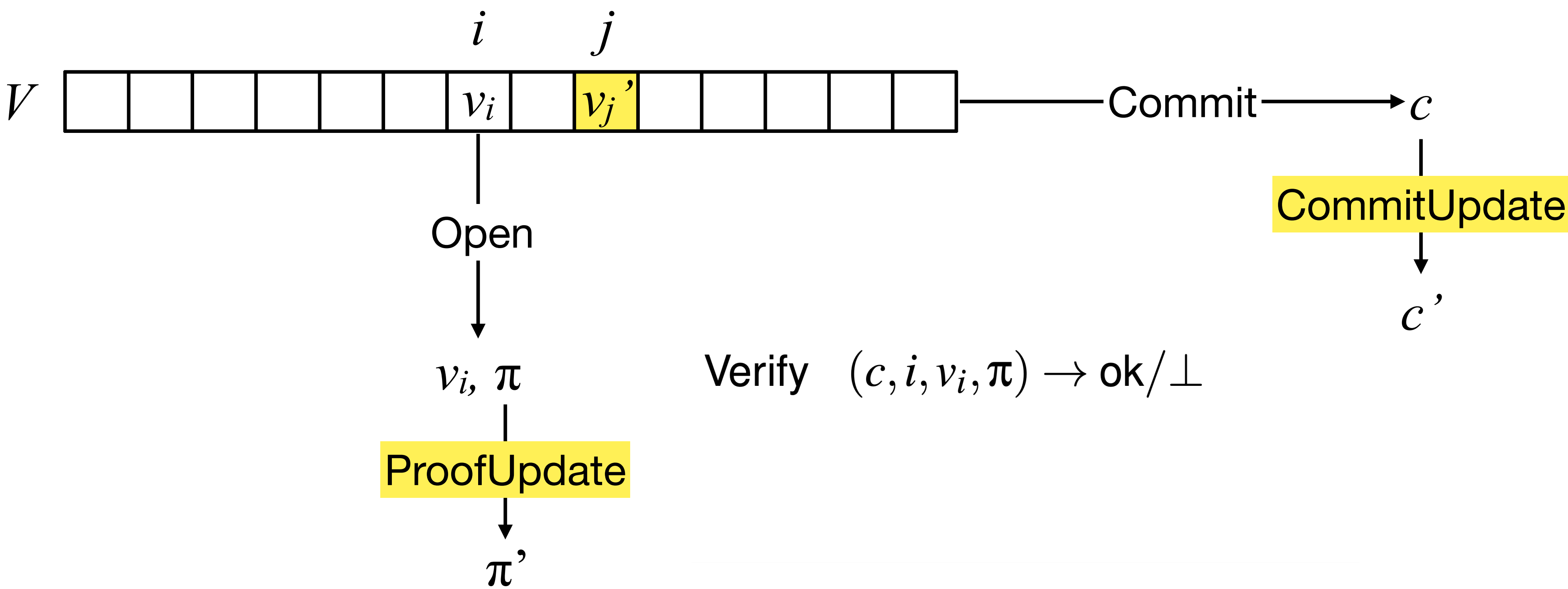


Background: Vector commitments



Verify $(c, i, v_i, \pi) \rightarrow$ ok$/\perp$

Background: Vector commitments



Background: Vector commitments

vs. Merkle Trees: Proof compactness

Only 48B!

Commit $(V) \rightarrow c$

Open $(V, i) \rightarrow v_i, \boxed{\pi}$

Verify $(c, i, v_i, \pi) \rightarrow \text{ok}/\perp$

CommitUpdate $(c, (i, v_i, v_i')) \rightarrow c'$

ProofUpdate $(\pi, j, (i, v_i, v_i')) \rightarrow \pi'$

Background: Vector commitments

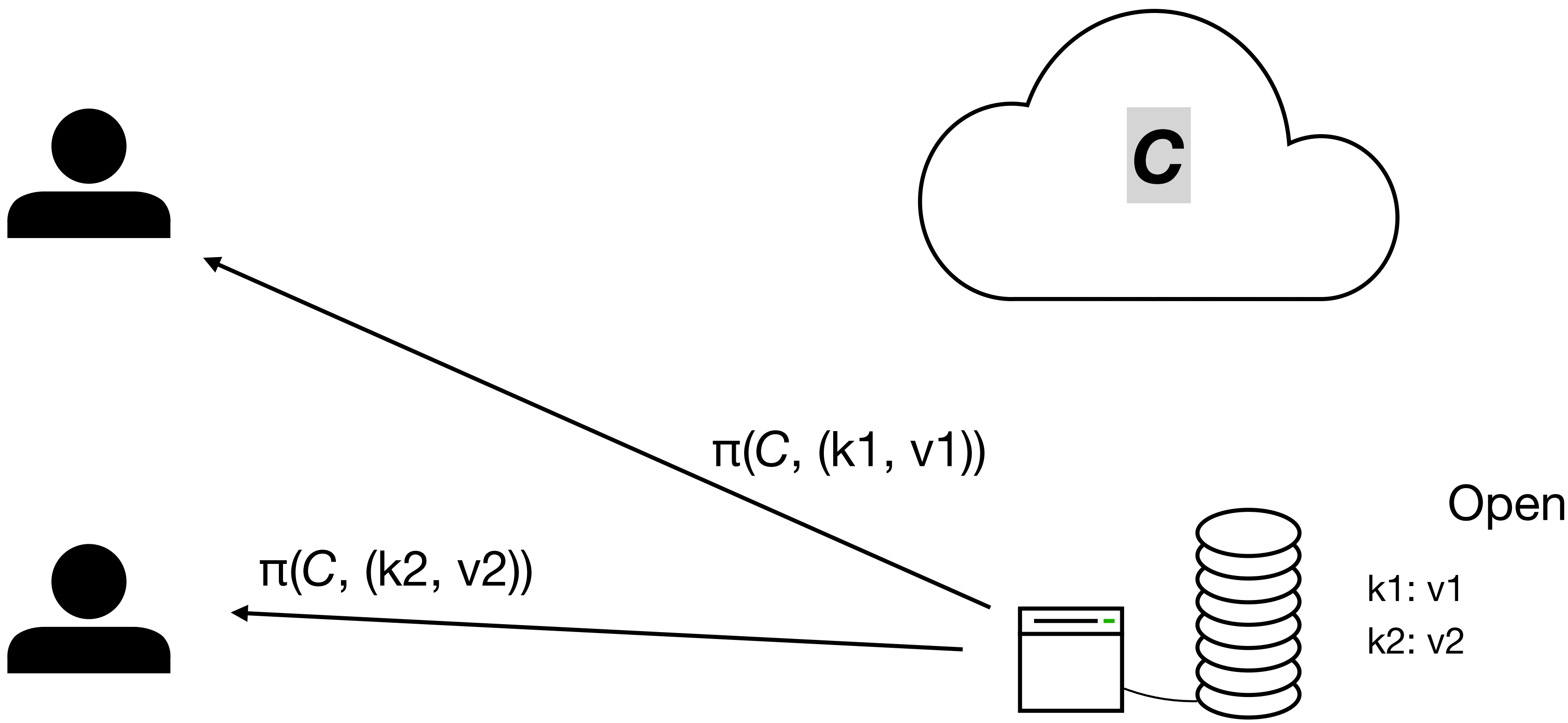
vs. Merkle Trees: Slowdown

Only 48B!

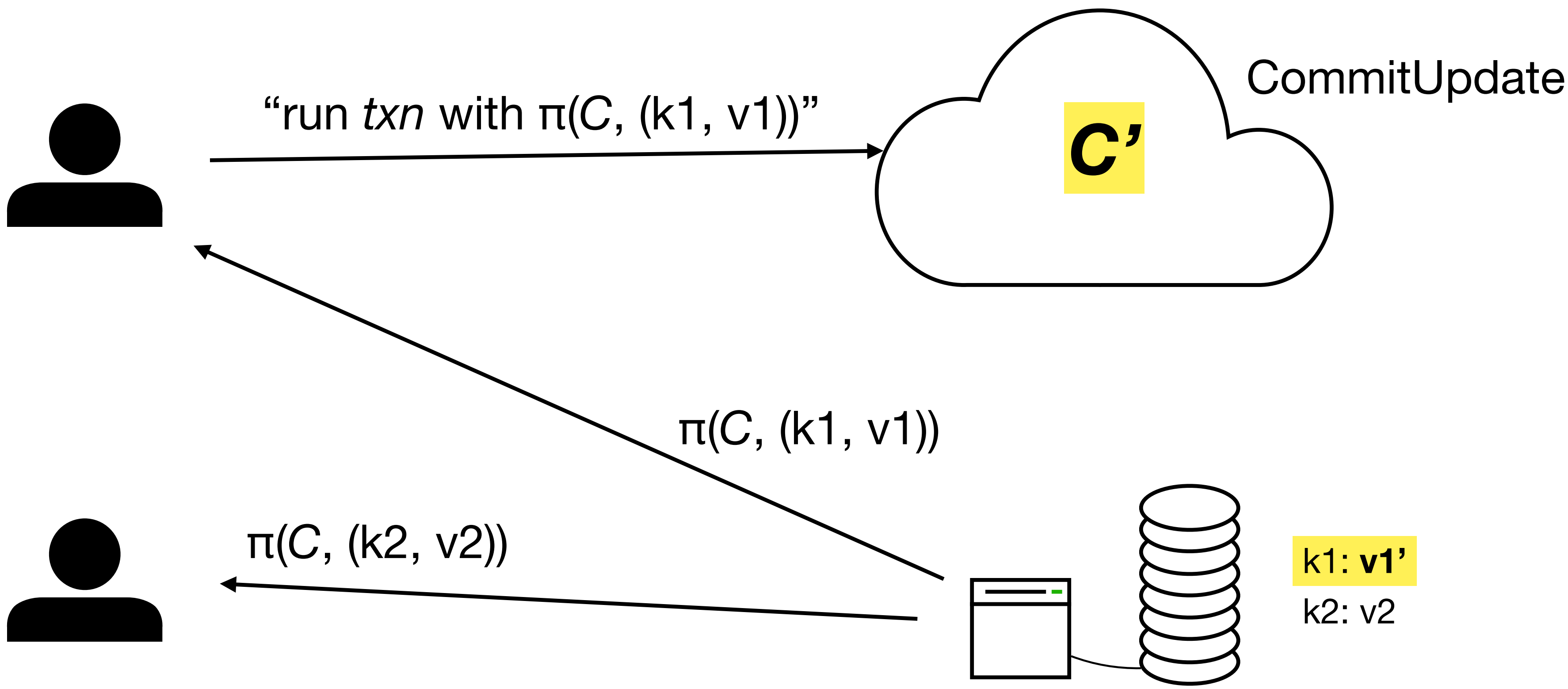
30x	Commit	$(V) \to c$
>40,000x	Open	$(V, i) \to v_i, \boxed{\pi}$
400x	Verify	$(c, i, v_i, \pi) \to \text{ok}/\perp$
7x	CommitUpdate	$(c, (i, v_i, v_i')) \to c'$
>60x	ProofUpdate	$(\pi, j, (i, v_i, v_i')) \to \pi'$

$|V| = 1000$

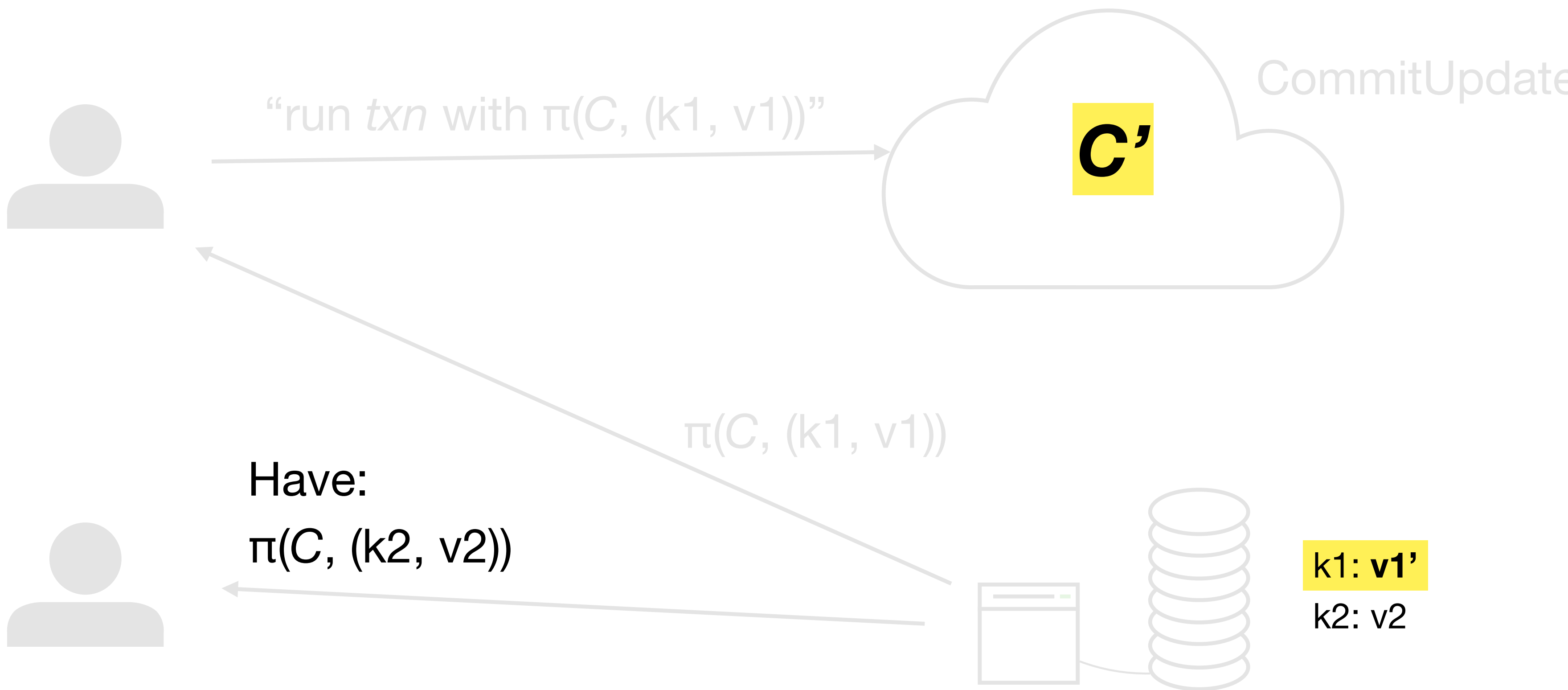
Problem: Concurrent Updates



Problem: Concurrent Updates

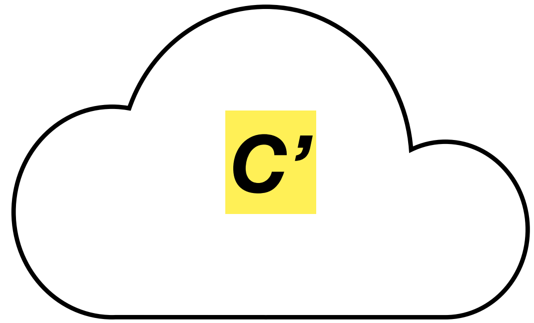


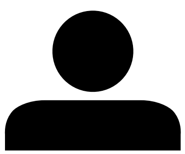
Problem: Concurrent Updates



Updating π: Quadratic Scaling

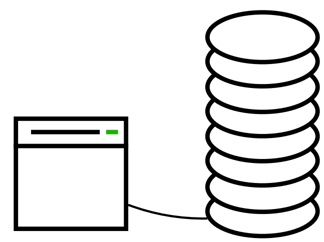
Throughput bottleneck





Have:
$\pi(C, (k2, v2))$

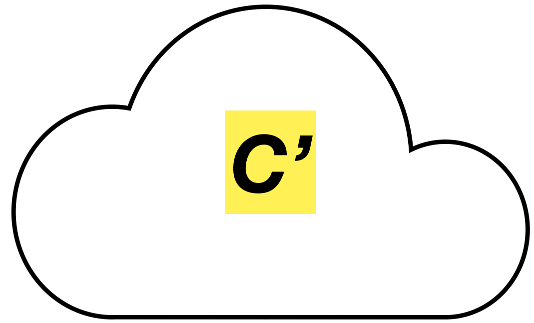
Need:
$\pi(C', (k2, v2))$



k1: **v1’**
k2: v2

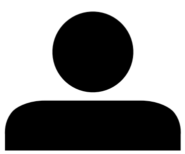
Updating π: Quadratic Scaling

Throughput bottleneck



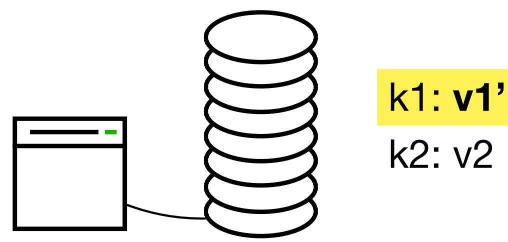
Options:

- Archive re-runs Open
- Client runs ProofUpdate
- Validator runs ProofUpdate



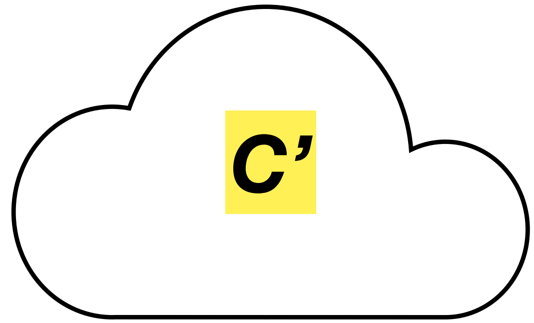
Have:
π(C, (k2, v2))

Need:
π(C', (k2, v2))



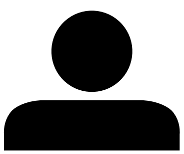
Updating π: Quadratic Scaling

Throughput bottleneck



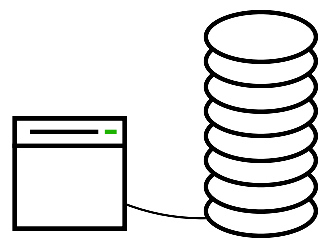
Options:

- ~~Archive re-runs Open~~ **Many round trips**
- Client runs ProofUpdate
- Validator runs ProofUpdate



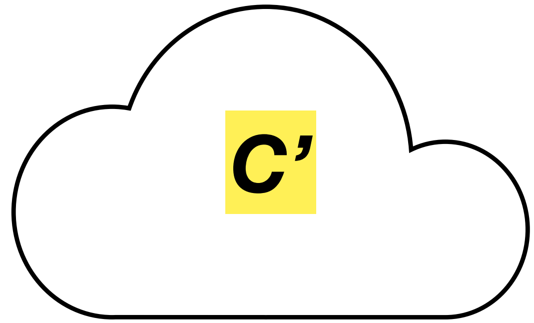
Have:
$\pi(C, (k2, v2))$

Need:
$\pi(C', (k2, v2))$



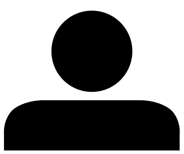
Updating π: Quadratic Scaling

Throughput bottleneck



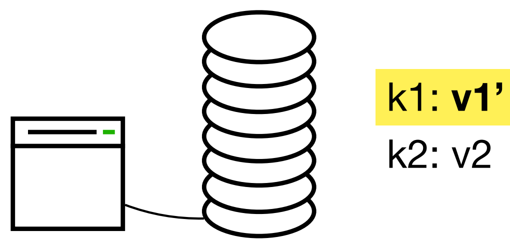
Options:

- ~~Archive re-runs Open~~ **Many round trips**
- ~~Client runs ProofUpdate~~ **Expensive**
- Validator runs ProofUpdate



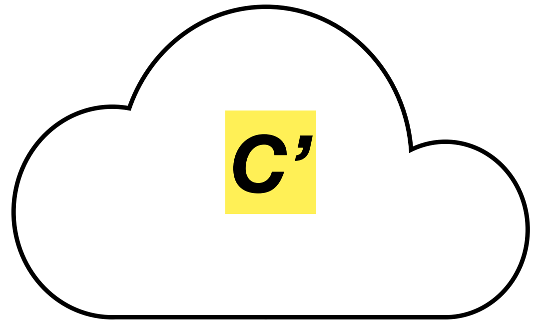
Have:
π(*C*, (k2, v2))

Need:
π(*C’*, (k2, v2))



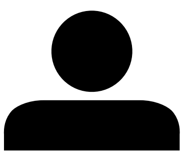
Updating π: Quadratic Scaling

Throughput bottleneck



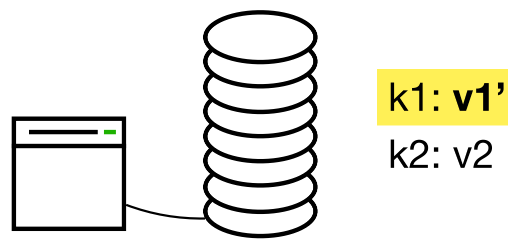
Options:

- ~~Archive re-runs Open~~ **Many round trips**
- ~~Client runs ProofUpdate~~ **Expensive**
- ~~Validator runs ProofUpdate~~ **Congestion collapse**



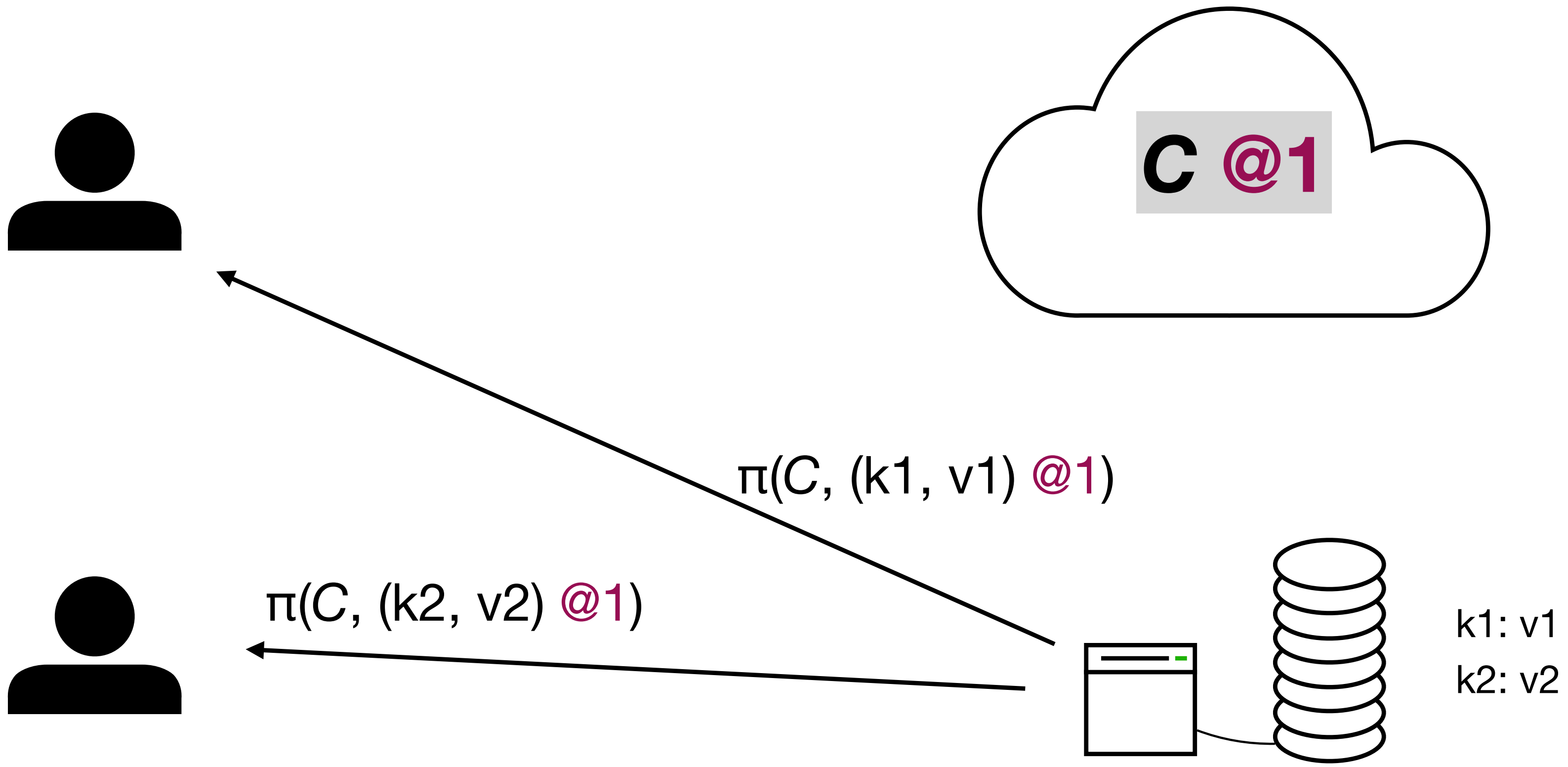
Have:
π(*C*, (k2, v2))

Need:
π(*C’*, (k2, v2))

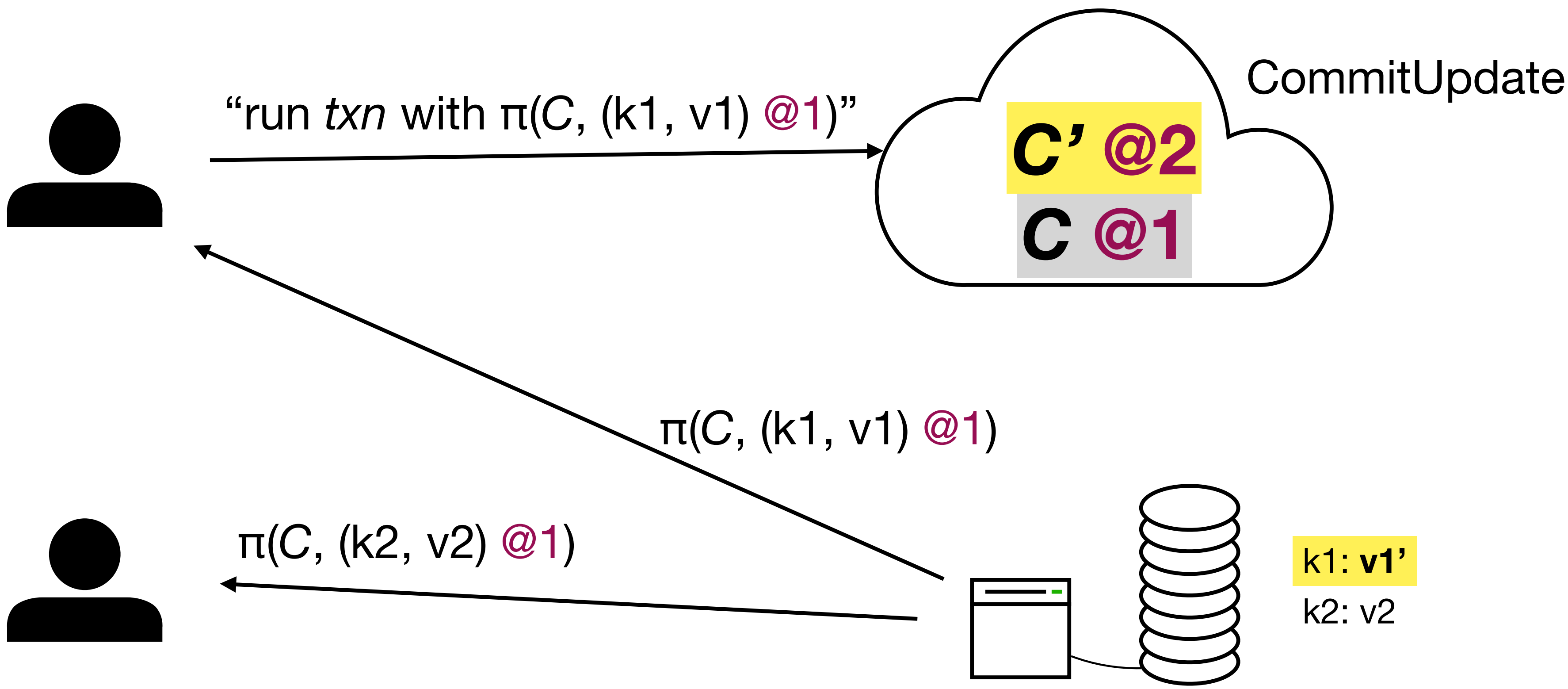


Stale proofs force serialization: Cache old state for parallelism

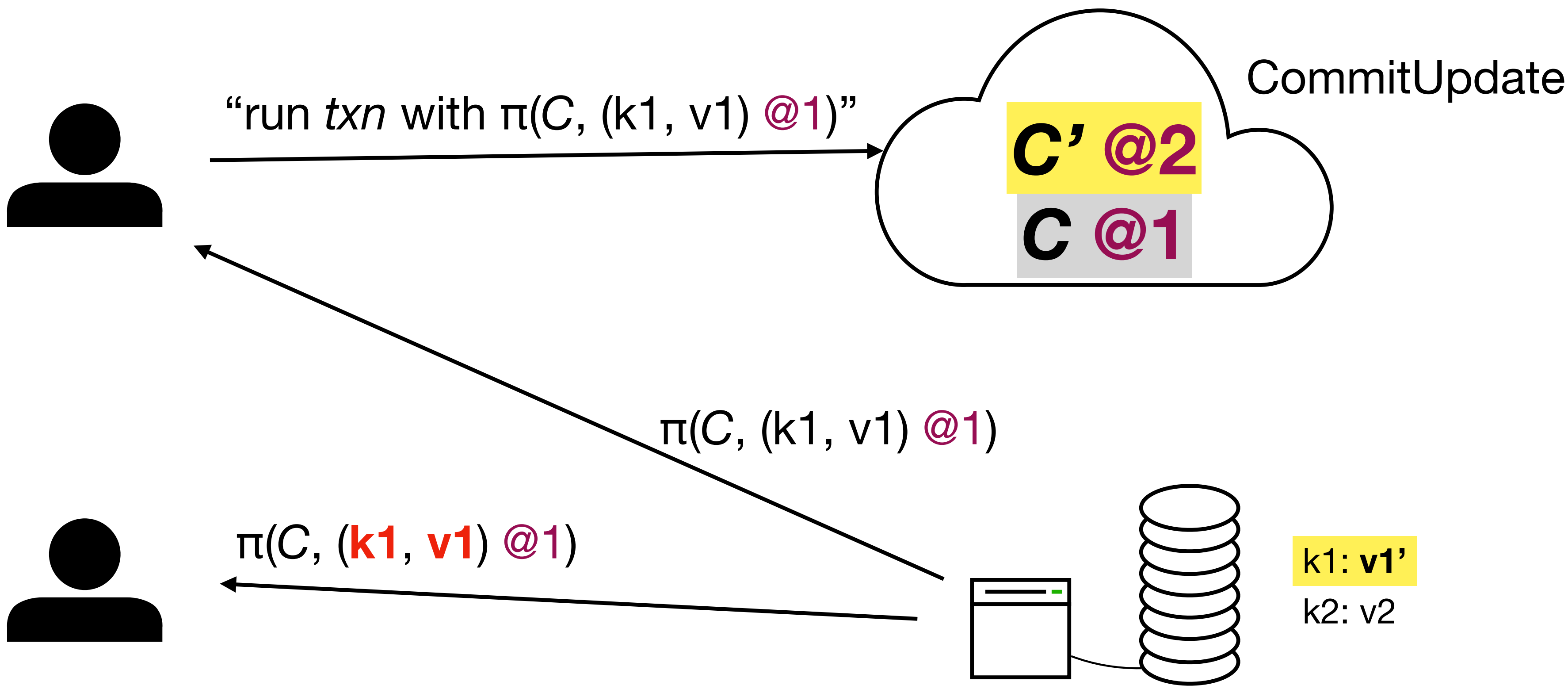
Solution: Dictionary versioning



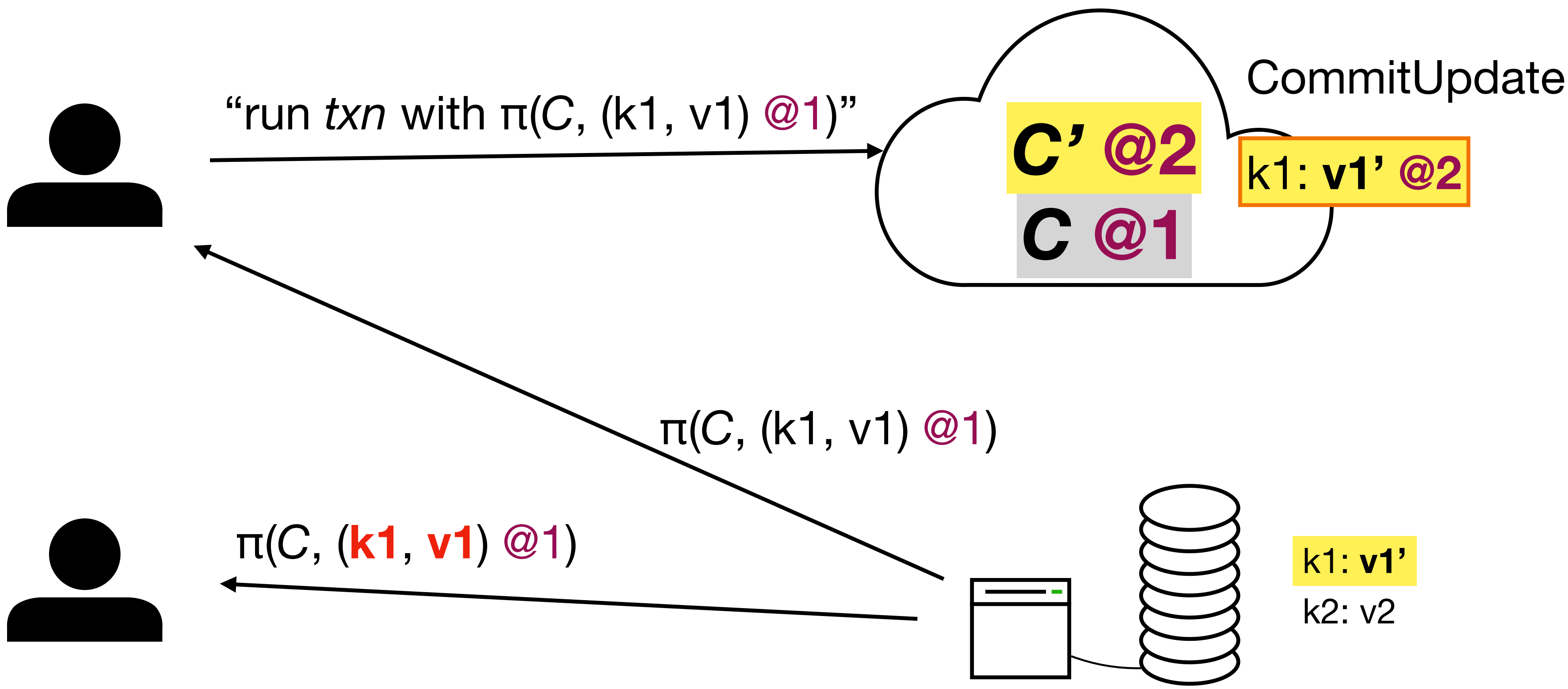
Solution: Dictionary versioning



Two transactions, one key?



Solution: Dictionary versioning + data caching



Solution: Dictionary versioning + data caching

Details

Nontrivial interaction with nonmembership proofs

- How to handle key insertion/deletion?

See paper for more details…

Evaluation

Integrated into implementation of Algorand cryptocurrency

Evaluation

Integrated into implementation of Algorand cryptocurrency

✓ Storage costs: reduced by >800x

✓ Proof size: 100–200B

Evaluation

Integrated into implementation of Algorand cryptocurrency

✓ Storage costs: reduced by >800x

✓ Proof size: 100–200B

- Effect on throughput and latency?
    - Focus: processing at validators

Evaluation: validator slowdown

AWS `c5.metal`, 100,000 {put, delete} ops / 10 blocks, 1M keys

Evaluation: validator slowdown

AWS `c5.metal`, 100,000 {put, delete} ops / 10 blocks, 1M keys

Processing overhead /
1MB block:

34–68s

Evaluation: parallelism helps

AWS `c5.metal`, 100,000 {put, delete} ops / 10 blocks, 1M keys

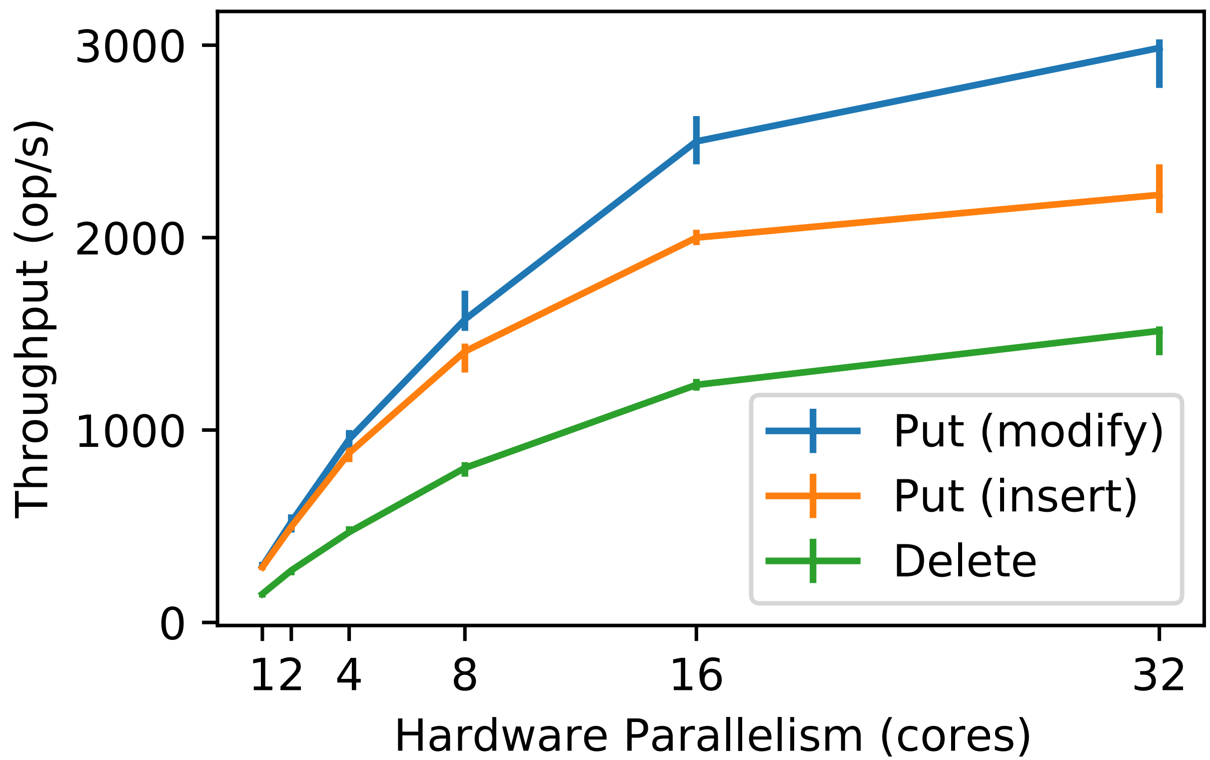
Processing overhead / 1MB block:

34–68s: **1 core**

3.4–6.7s: **32 cores**

↓

8–10x speedup



Related work

- Merkle Trees
- EDRAX (Chepurnoy *et al.*) (+ other VC-based schemes)
  - Aardvark versioning can help manage high compute costs

Related work

- Merkle Trees
- EDRAX (Chepurnoy *et al.*) (+ other VC-based schemes)
  - Aardvark versioning can help manage high compute costs
- Rollups (Ethereum community) (+ zk-SNARKs)
  - Verification: Near constant processing and transmission costs
  - Proof creation: >10x slower/txn than VCs; forced batching

Conclusion

An authenticated dictionary with

- Short proofs (**100–200B**) + commitments (**0.1%** storage cost)

    Built from vector commitments, via short proofs of nonmembership

- Low latency at high throughput

    Using transactional concurrency control, via versioning state

dtl@mit.edu https://github.com/derbear/aardvark-prototype

Backup slides

Transaction detail

```
a = txn.Get(alice)
b = txn.Get(bob)
assert a >= p
if a-p == 0:
    txn.Delete(alice)
else:
    txn.Put(alice, a-p)
txn.Put(bob, b+p)
```

Transaction restriction: static keys

```
with Transaction(alice, bob) as txn:
    a = txn.Get(alice)
    b = txn.Get(bob)
    assert a >= p
    if a-p == 0:
        txn.Delete(alice)
    else:
        txn.Put(alice, a-p)
    txn.Put(bob, b+p)
```

Transaction restriction: static keys

```
                            π     π
with Transaction(alice, bob) as txn:
    a = txn.Get(alice)
    b = txn.Get(bob)
    assert a >= p
    if a-p == 0:
        txn.Delete(alice)
    else:
        txn.Put(alice, a-p)
    txn.Put(bob, b+p)
```

Table 1: VC Operation Latency (mean $\pm$ SD μs)

	Operation	Aardvark	Merkle Tree
30x	Commit	$40\,262 \pm 129$	$1\,317 \pm 4$
>40,000x	Open	$40\,277 \pm 444$	< 1
400x	Verify	$3\,707 \pm 10$	9 ± 0
7x	CommitUpdate	62 ± 1	9 ± 0
>60x	ProofUpdate	62 ± 1	< 1

$|V| = 1000$

Table 1: VC Operation Latency (mean $\pm$ SD μs)

i.e., about	Operation	Aardvark	Merkle Tree
40 ms	Commit	$40\,262 \pm 129$	$1\,317 \pm 4$
40 ms	Open	$40\,277 \pm 444$	< 1
4 ms	Verify	$3\,707 \pm 10$	9 ± 0
60 µs	CommitUpdate	62 ± 1	9 ± 0
60 µs	ProofUpdate	62 ± 1	< 1

$|V| = 1000$

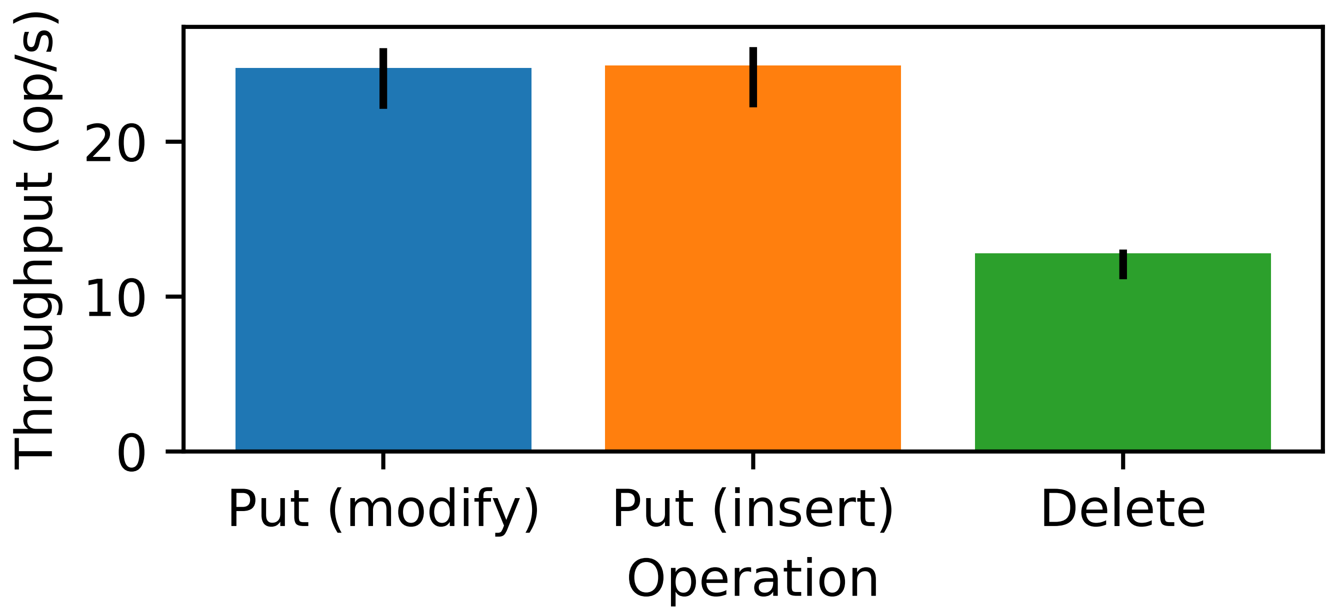

Throughput (op/s)
0
10
20
Put (modify)
Put (insert)
Delete
Operation

Table 1: VC Operation Latency (mean $\pm$ SD μs)

Operation	Aardvark	Merkle Tree	EDRAX w/o SNARK
Commit	$40\,262 \pm 129$	$1\,317 \pm 4$	—
Open	$40\,277 \pm 444$	< 1	—
Verify	$3\,707 \pm 10$	9 ± 0	$3\,131 \pm 9$
CommitUpdate	62 ± 1	9 ± 0	13 ± 1
ProofUpdate	62 ± 1	< 1	27 ± 19