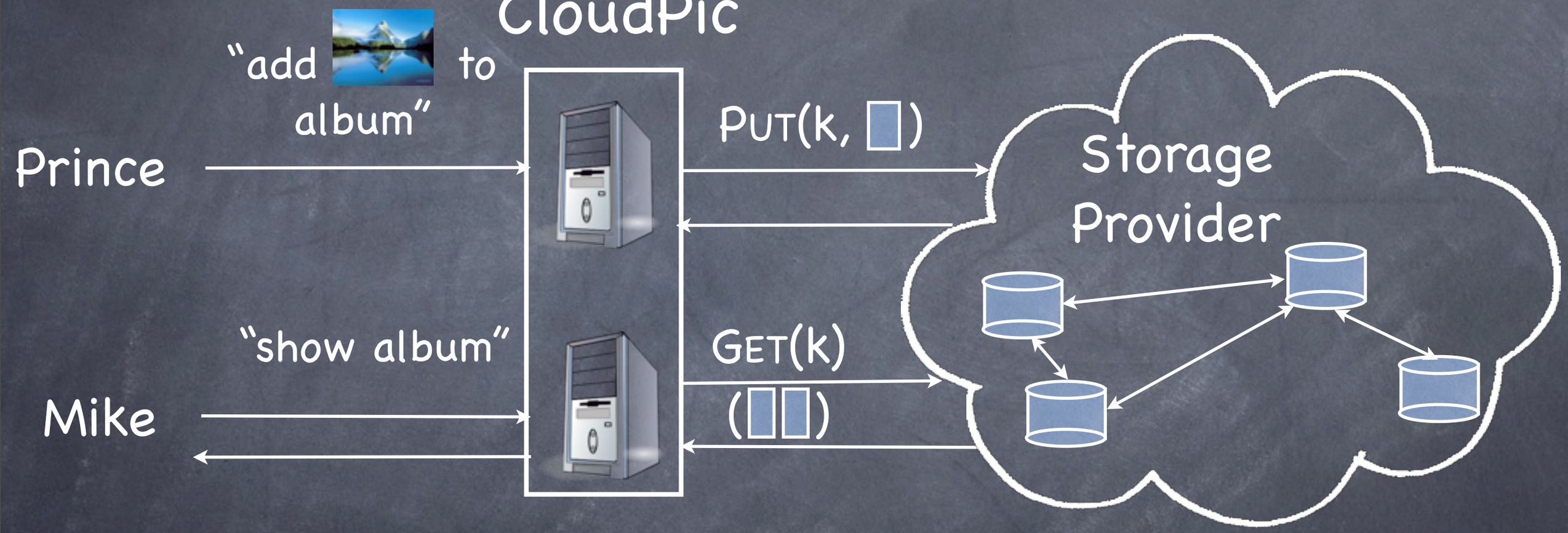
Depot

Cloud storage with minimal trust

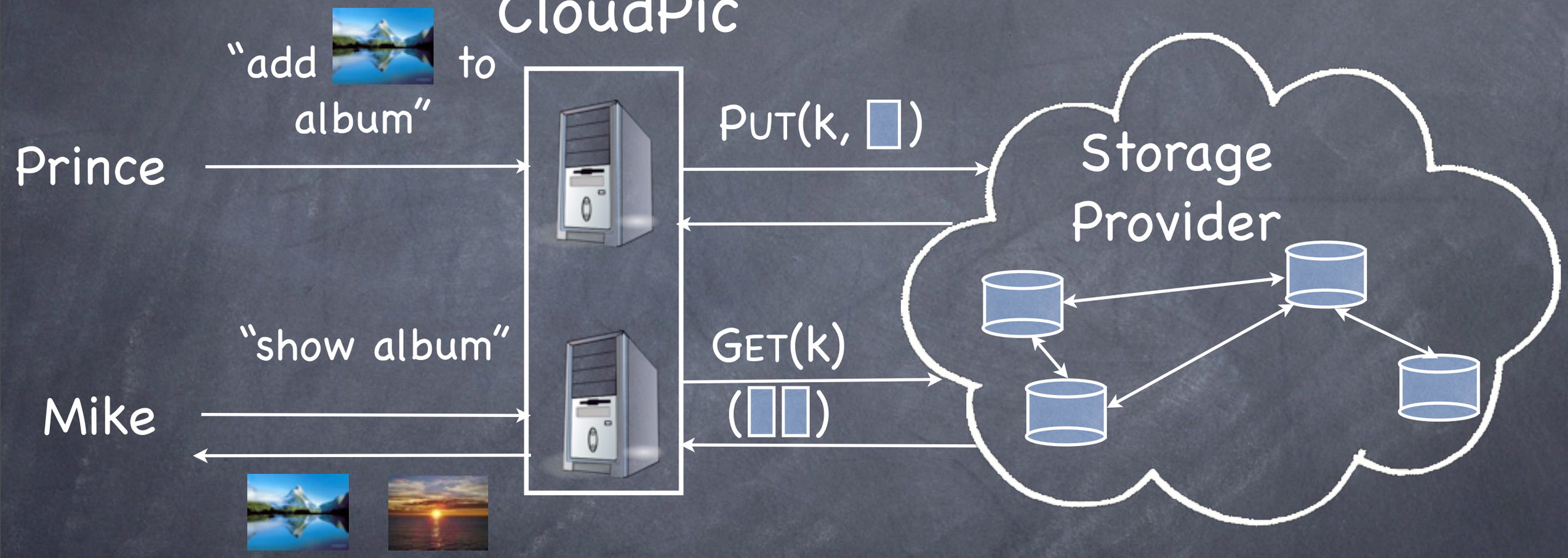
Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement,
Lorenzo Alvisi, Mike Dahlin, Michael Walfish

The University of Texas at Austin

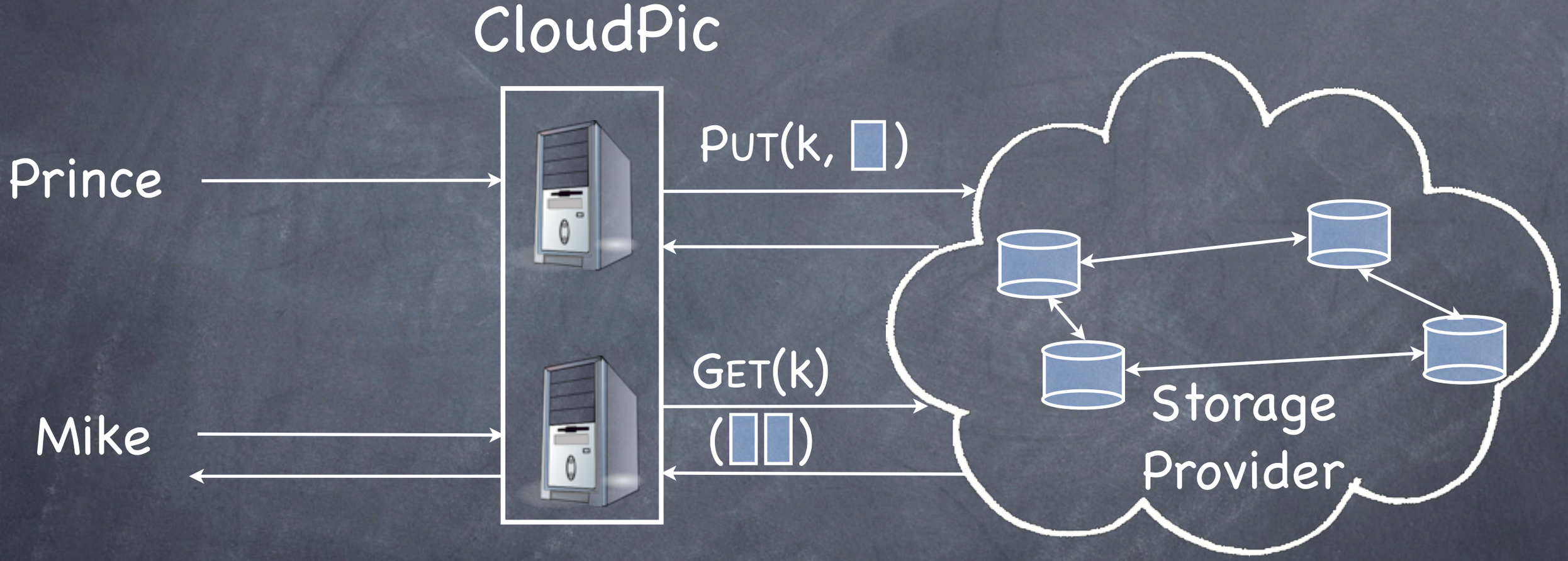
Cloud storage is appealing



Cloud storage is appealing



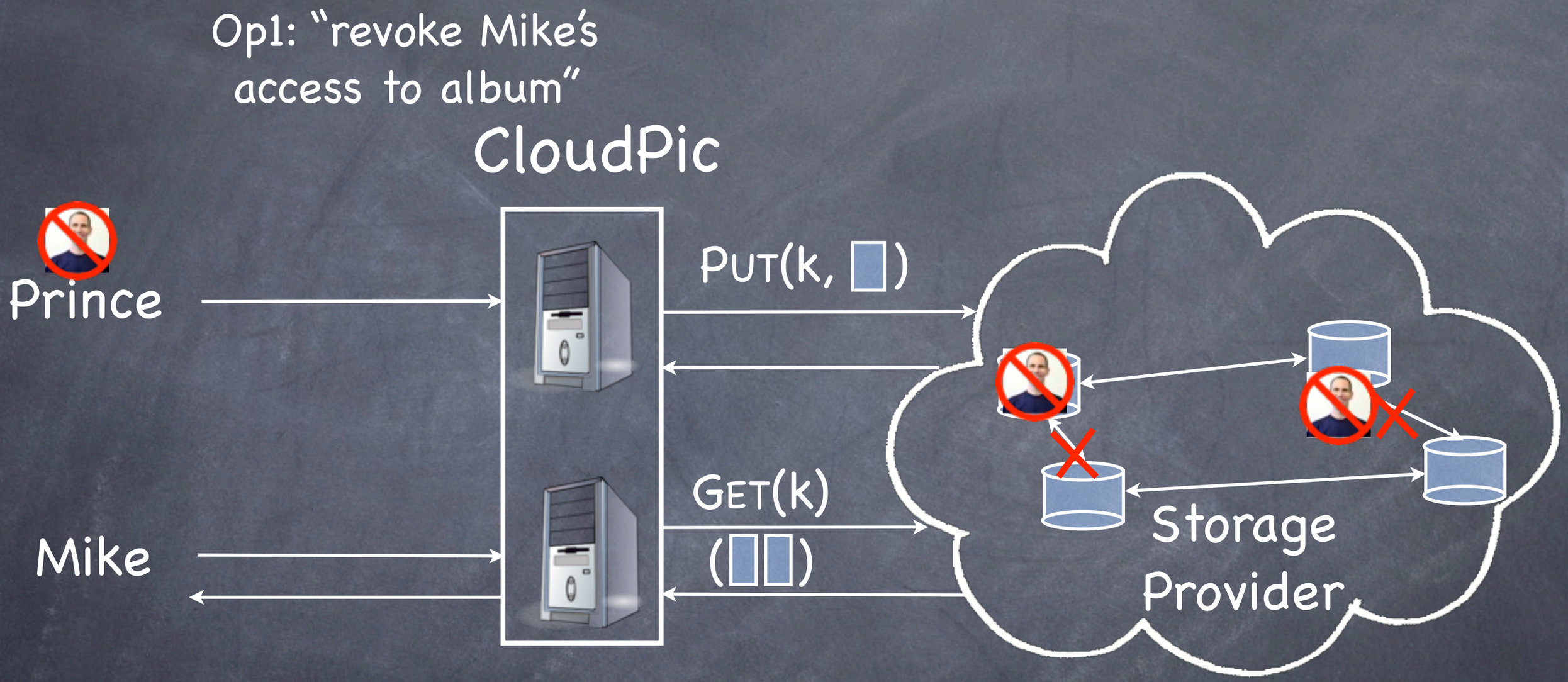
Risks of cloud storage



Failures cause undesired behavior

Risks of cloud storage

Op1: "revoke Mike's access to album"

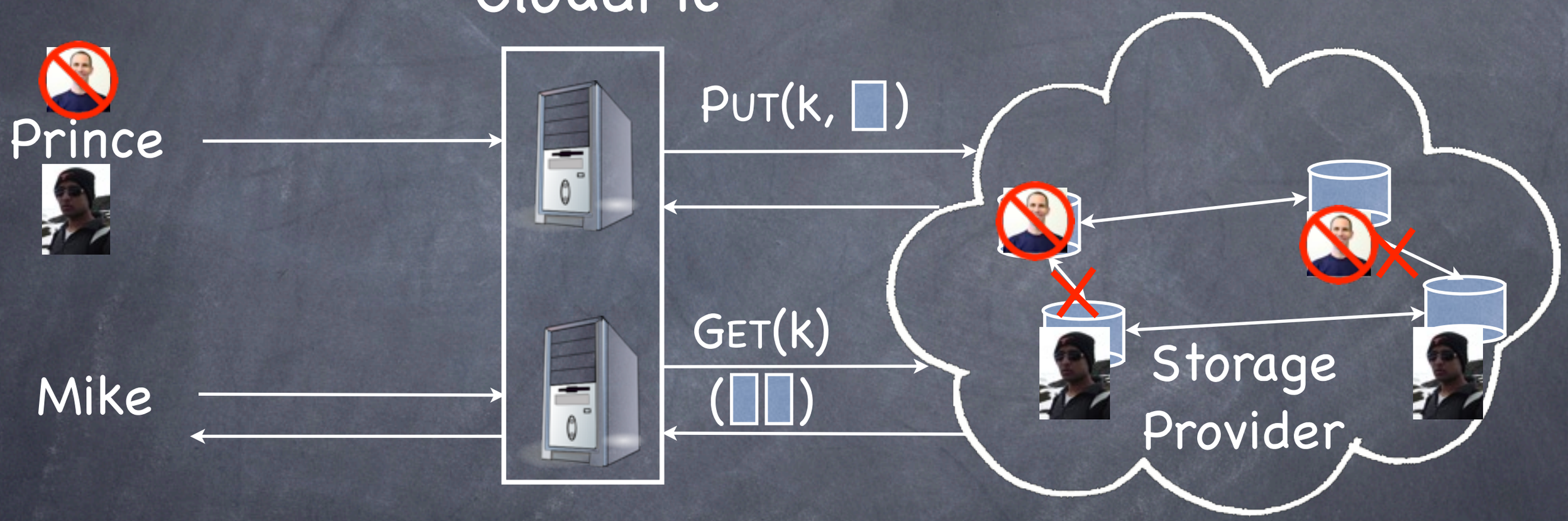


Failures cause undesired behavior

Risks of cloud storage

Op1: "revoke Mike's access to album"

Op2:"add  to album"



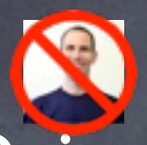
Failures cause undesired behavior

Risks of cloud storage

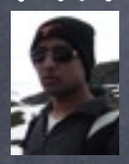
Op1: "revoke Mike's access to album"

Op2:"add  to album"

CloudPic

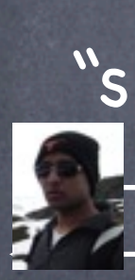


Prince



PUT(k,)

"show album"



Mike

GET(k)

( )

Storage Provider

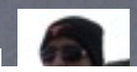
Failures cause undesired behavior

Risks of cloud storage

FAST 2007

Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?
Bianca Schroeder and Garth A. Gibson, *Carnegie Mellon University*

Failure Trends in a Large Disk Drive Population
Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso, *Google Inc.*



in IT / Why San Francisco's network admin went rogue

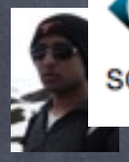
Why San Francisco's network admin went rogue

An inside source reveals details of missteps and misunderstandings in the curious case of Terry Childs, network

Amazon S3 Availability Event: July 20, 2008

We wanted to provide some additional detail about the problem we experienced

Prin



IRON file systems
Vijayan Prabhakaran, Lakshmi N. Bairavasundaram, Nitin Agrawal, Haryadi S. Gunawi, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau *University of Wisconsin*
SOSP'05

InfoWorld

Amazon S3 Issues: Load Balancers and MD5

ne 27th, 2008 : Rich Miller

Amazon's S3 storage system had some issues last week with data corruption on files using MD5 to perform integrity checks. After some investigation, Amazon confirmed the problems and identified the cause:

We've isolated this issue to a single load balancer that was brought into service at 10:55pm PDT on Friday, 6/20. It was taken out of service at 11am PDT Sunday, 6/22. While it was in service it handled a small fraction of

in the US. Intermittently,

single bytes in the byte

estigation with both internal

cnet news

May 12, 2009 9:00 AM PDT

What happens to data when a Web start-up dies?

Google Data Center Fire Returns Worldwide 404 Errors

Travis Wright | Apr 01, 2010 | Comments (3)

Gmail Disaster: Reports Of Mass Email Deletions

Michael Arrington

Dec 28, 2006

Like 5 Buzz 3

OSDI 06
7th USENIX Symposium on Operating Systems Design and Implementation

Rethink the Sync
Edmund B. Nightingale, Kaushik Veeraraghavan, Peter M. Chen, and Jason Flinn, *University of Michigan*

EXPLODE: A Lightweight, General System for Finding Serious Storage System Errors
Junfeng Yang, Can Sar, and Dawson Engler, *Stanford University*

We have a conflict

Much to like

- Geographic replication
- Professional management
- Low cost

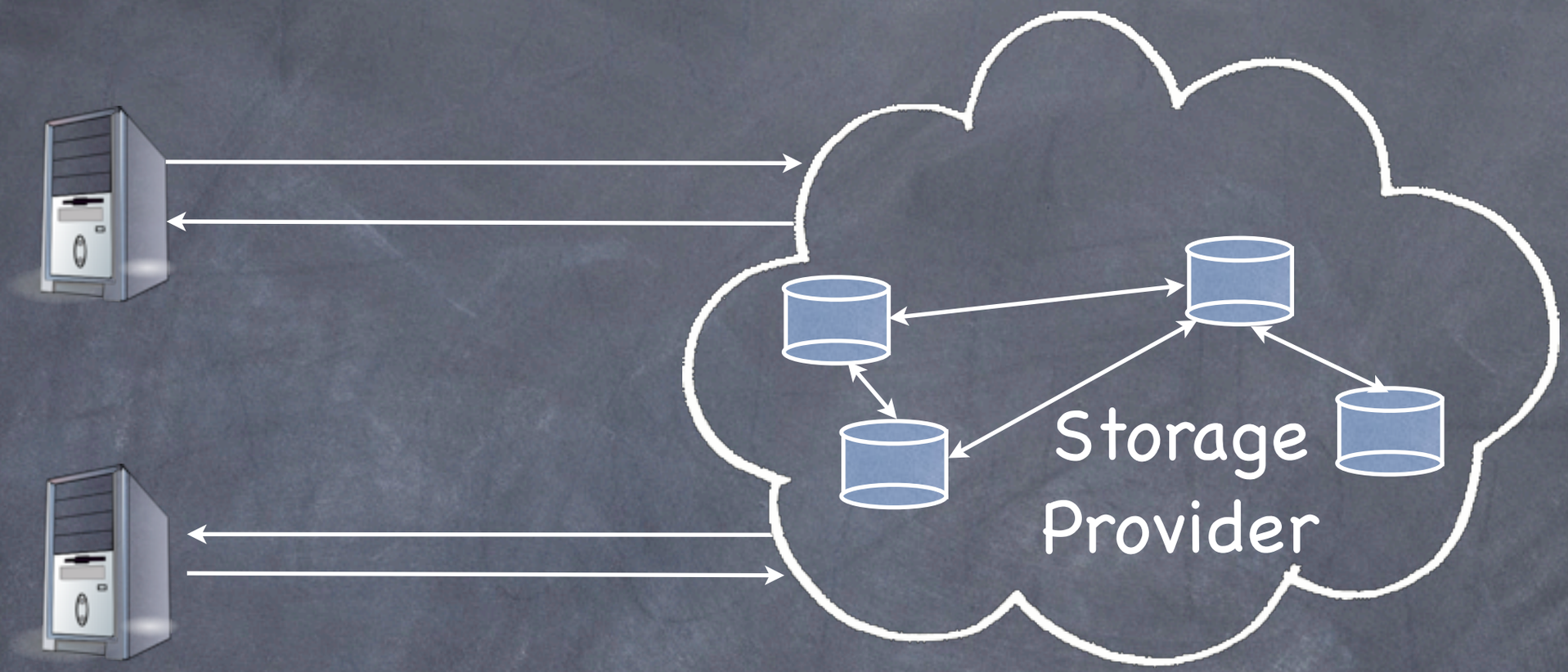
Much to give pause

- Black box
- Complex
- Error-prone

Our approach:

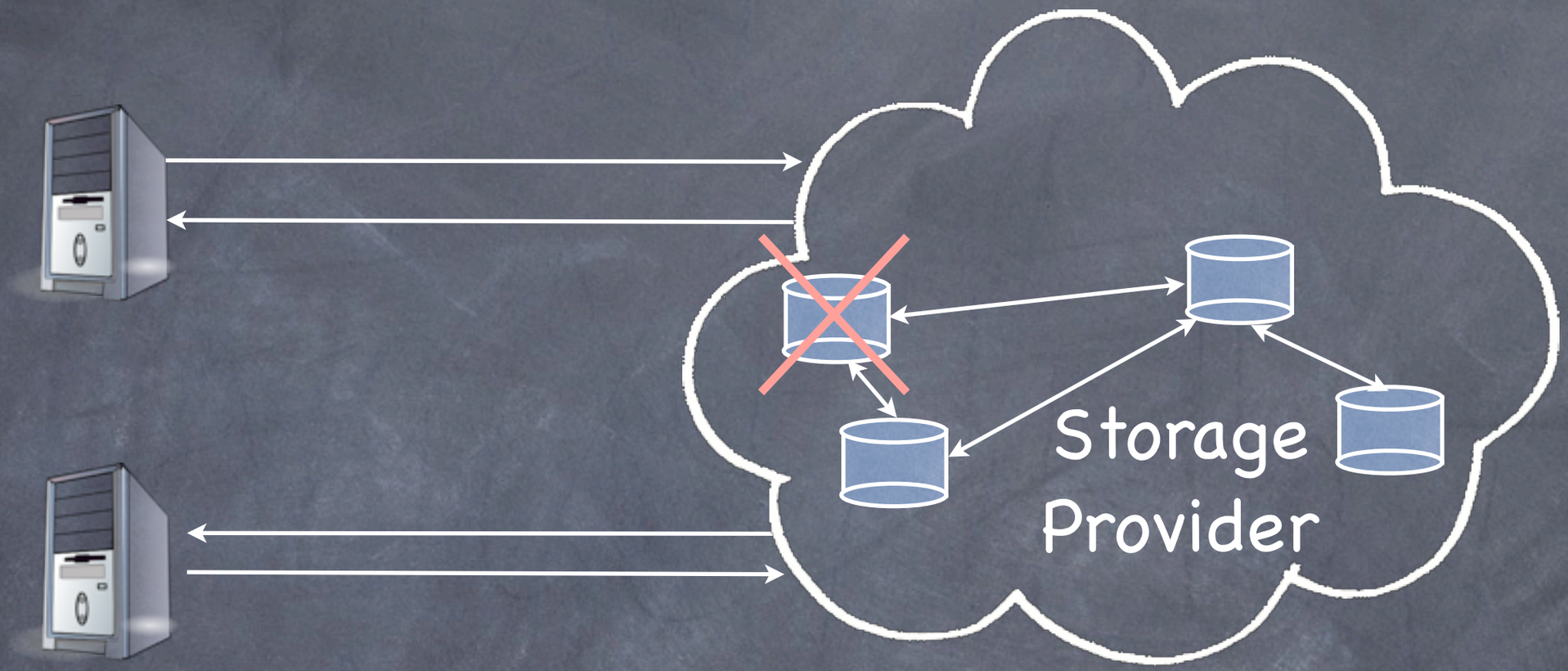
A radical fault-tolerance stance

Cloud storage with minimal trust



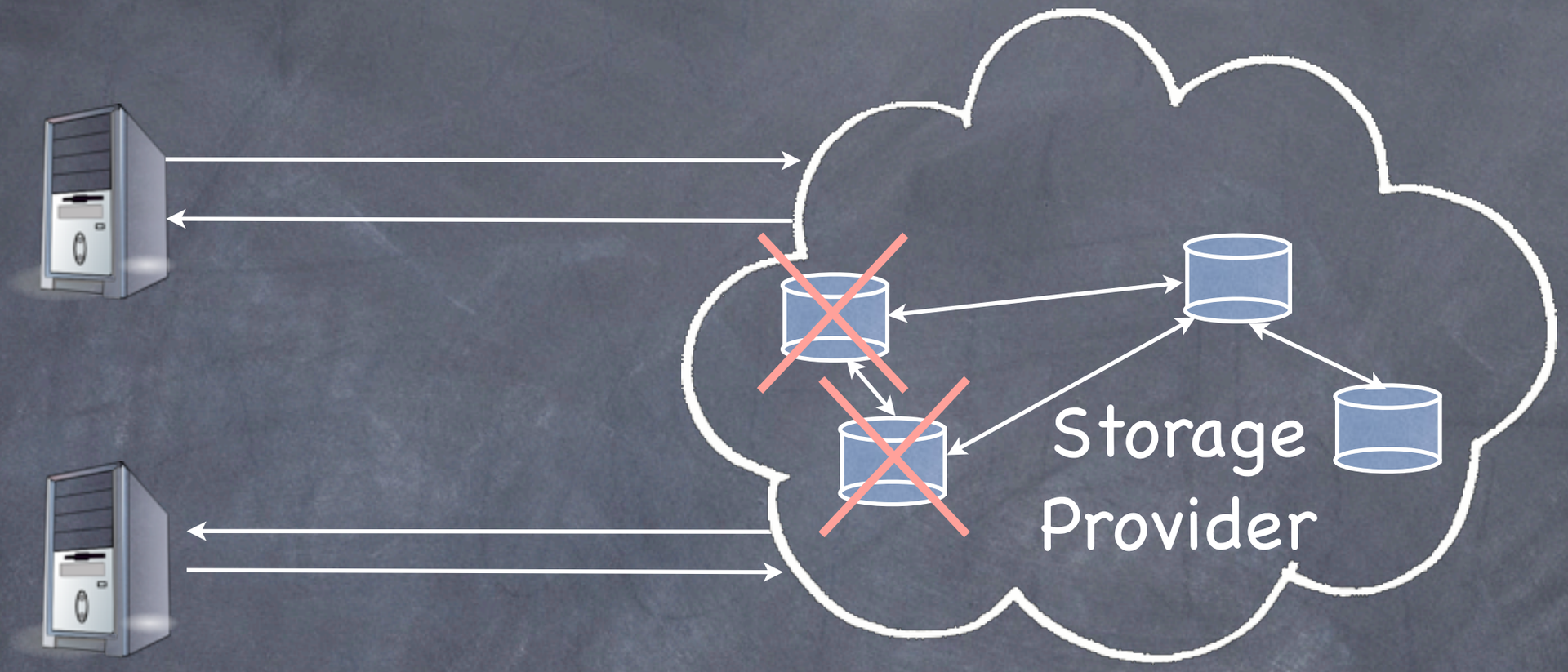
- Eliminates trust for
  - PUT availability
  - Eventual consistency
  - Staleness detection
  - Dependency preservation
- Minimizes trust for
  - GET availability
  - Durability

Cloud storage with minimal trust



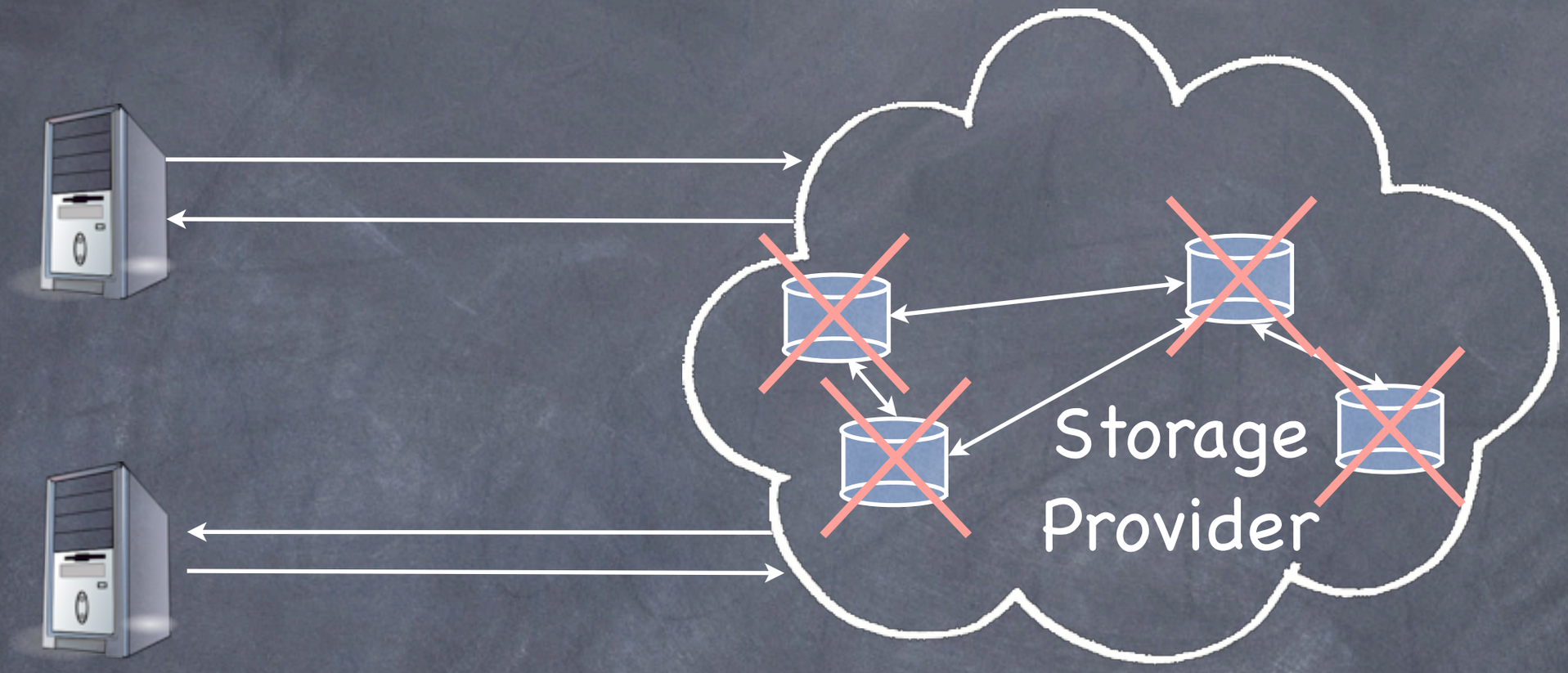
- Eliminates trust for
  - PUT availability
  - Eventual consistency
  - Staleness detection
  - Dependency preservation
- Minimizes trust for
  - GET availability
  - Durability

Cloud storage with minimal trust



- Eliminates trust for
  - PUT availability
  - Eventual consistency
  - Staleness detection
  - Dependency preservation
- Minimizes trust for
  - GET availability
  - Durability

Cloud storage with minimal trust



- Eliminates trust for
  - PUT availability
  - Eventual consistency
  - Staleness detection
  - Dependency preservation
- Minimizes trust for
  - GET availability
  - Durability

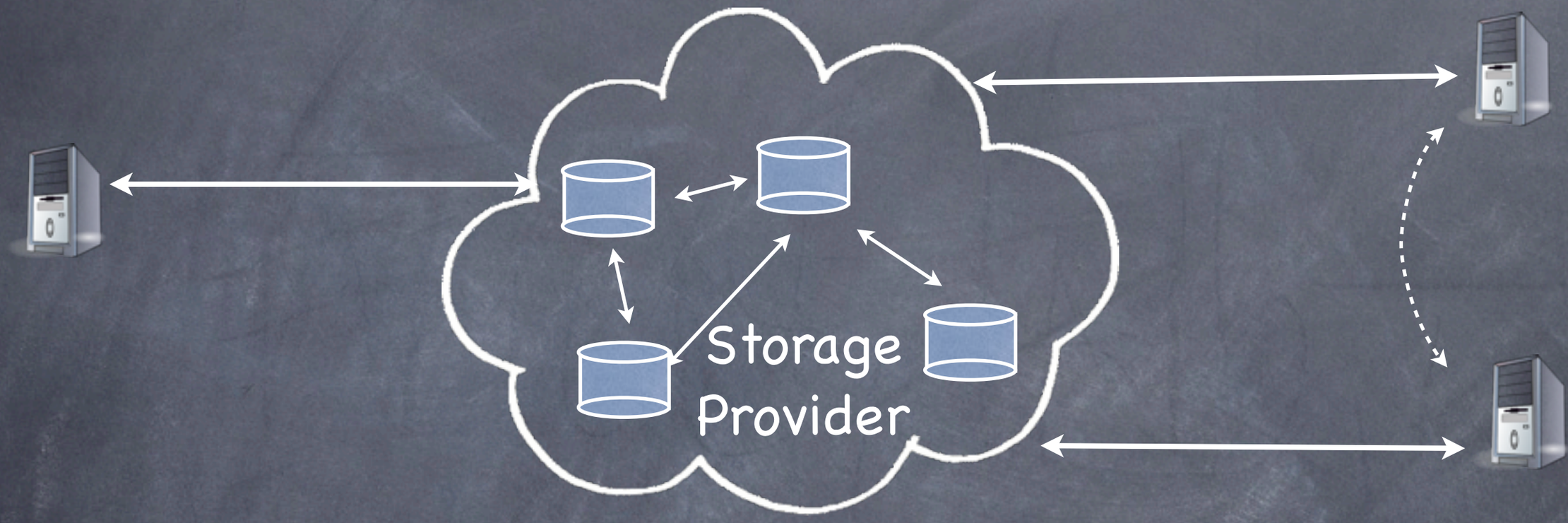
Rest of the talk

I. How does Depot work?

II. What properties does it provide?

III. How much does it cost?

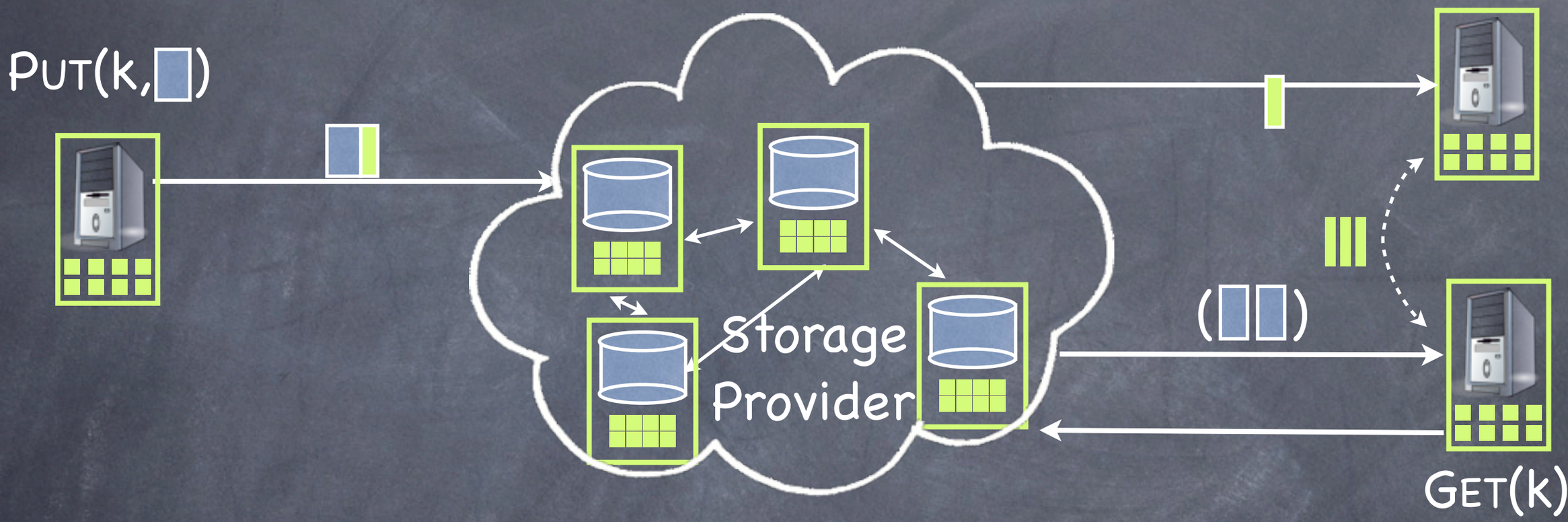
Depot in a nutshell



Ensuring high availability

- Multiple servers
- Don't enforce sequential (CAP tradeoff)
- Fall back on client-client communication

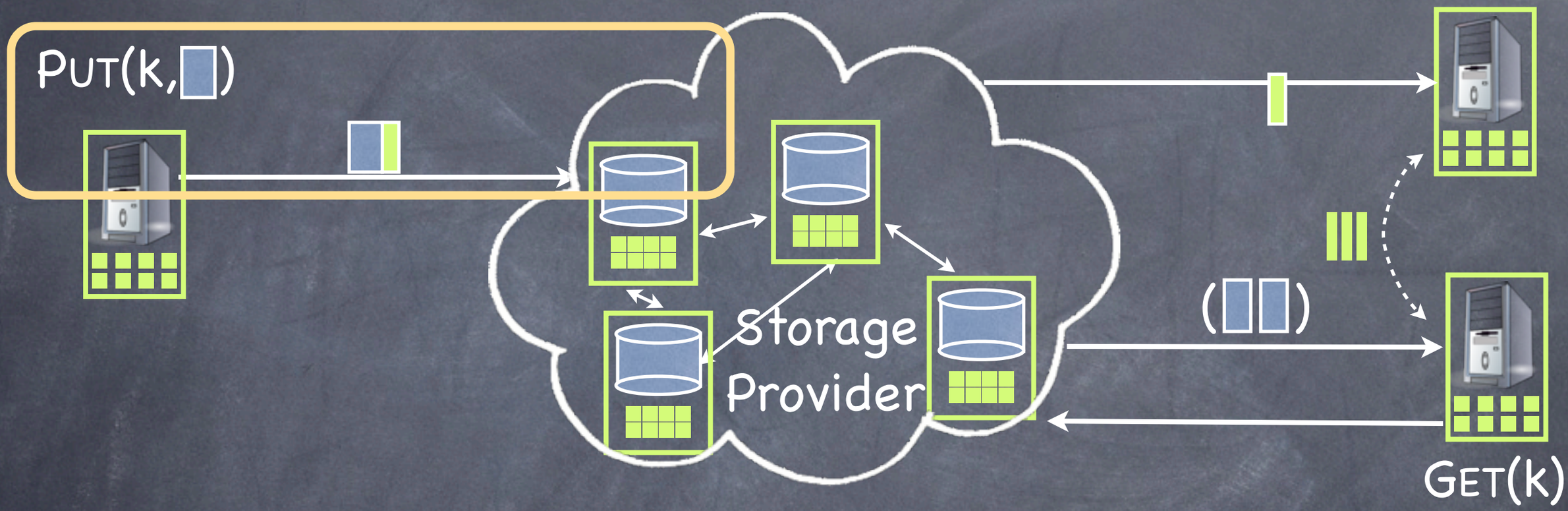
Depot in a nutshell



Preventing omission, reordering

- Add metadata to PUTs
- Add local state to nodes
- Add checks on received metadata

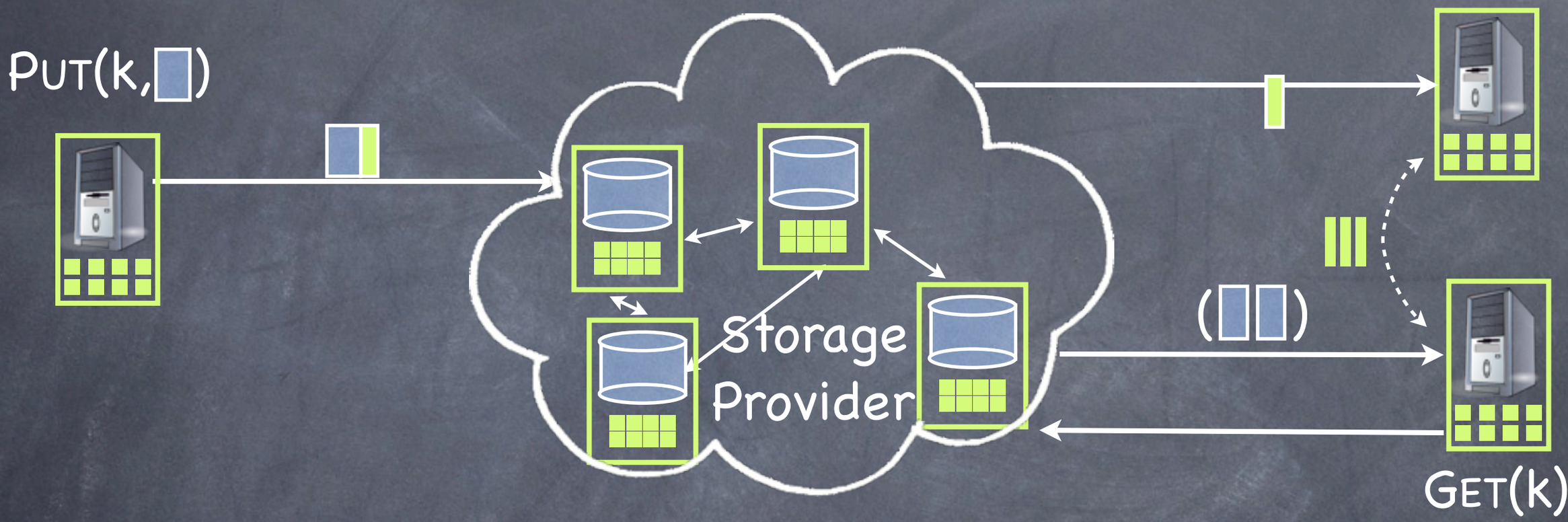
Depot in a nutshell



Preventing omission, reordering

- Add metadata to PUTs
- Add local state to nodes
- Add checks on received metadata

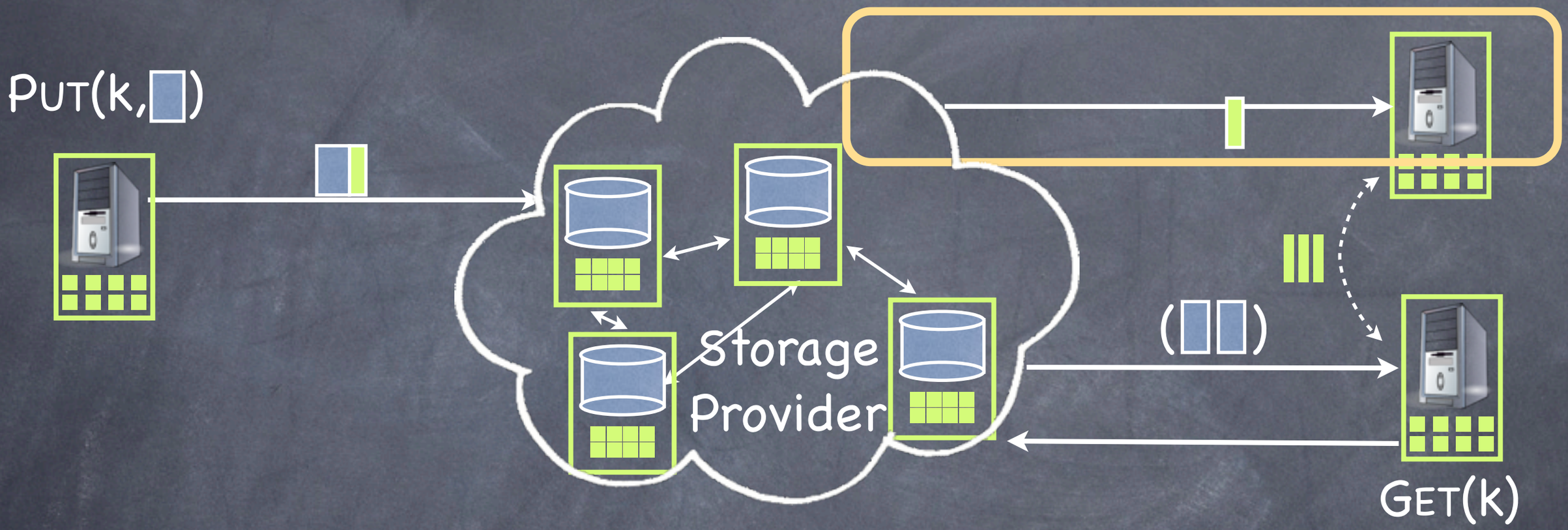
Depot in a nutshell



Preventing omission, reordering

- Add metadata to PUTs
- Add local state to nodes
- Add checks on received metadata

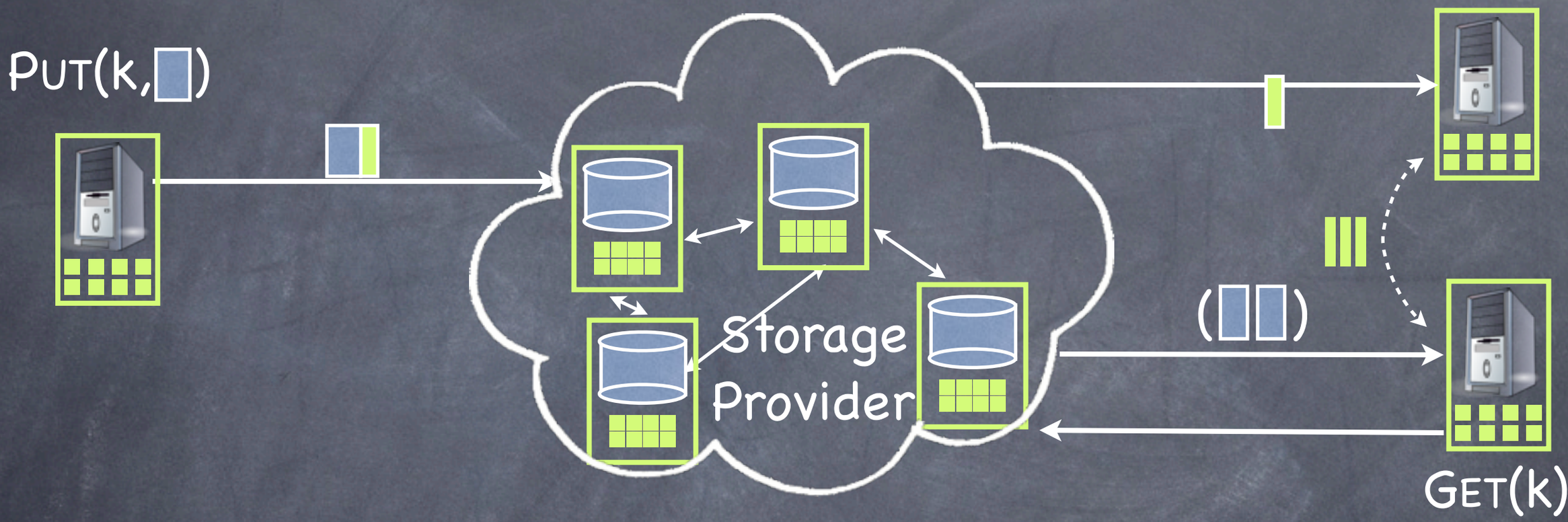
Depot in a nutshell



Preventing omission, reordering

- Add metadata to PUTs
- Add local state to nodes
- Add checks on received metadata

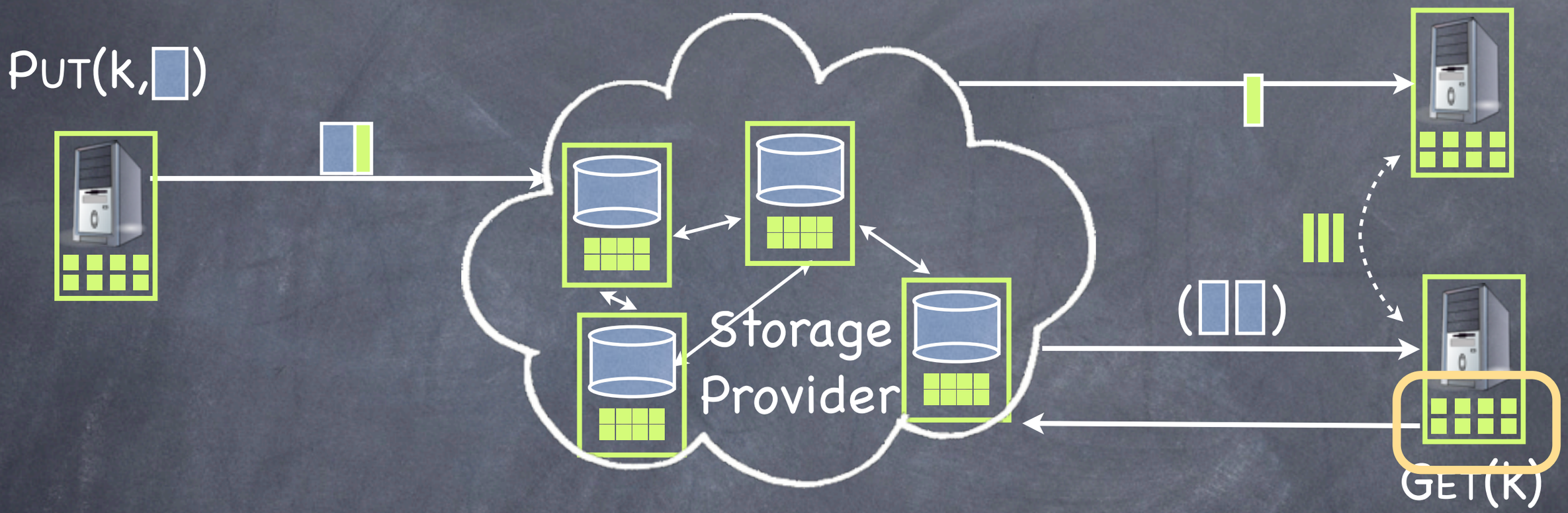
Depot in a nutshell



Preventing omission, reordering

- Add metadata to PUTs
- Add local state to nodes
- Add checks on received metadata

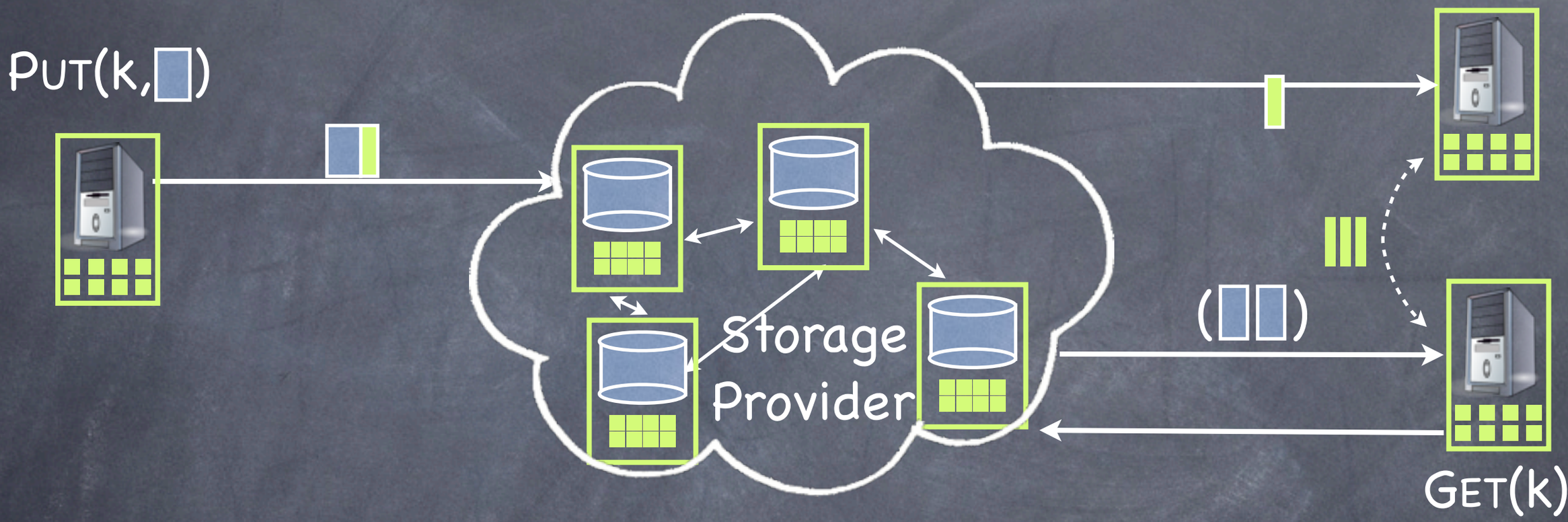
Depot in a nutshell



Preventing omission, reordering

- Add metadata to PUTs
- Add local state to nodes
- Add checks on received metadata

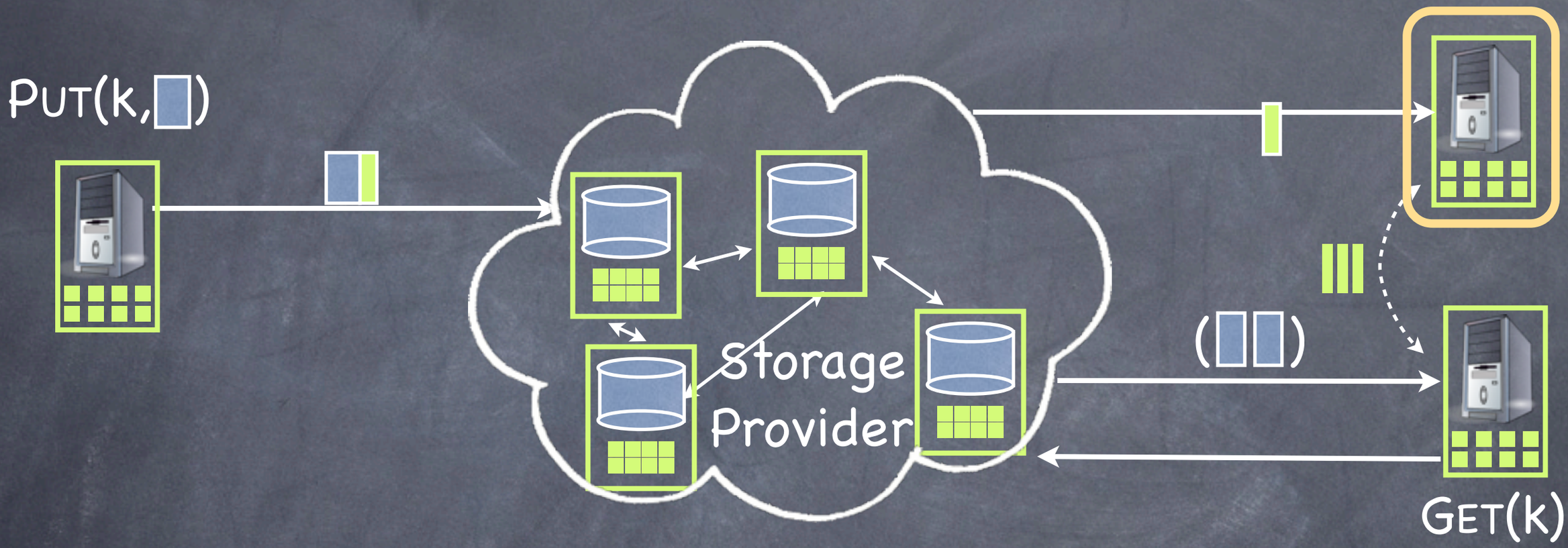
Depot in a nutshell



Preventing omission, reordering

- Add metadata to PUTs
- Add local state to nodes
- Add checks on received metadata

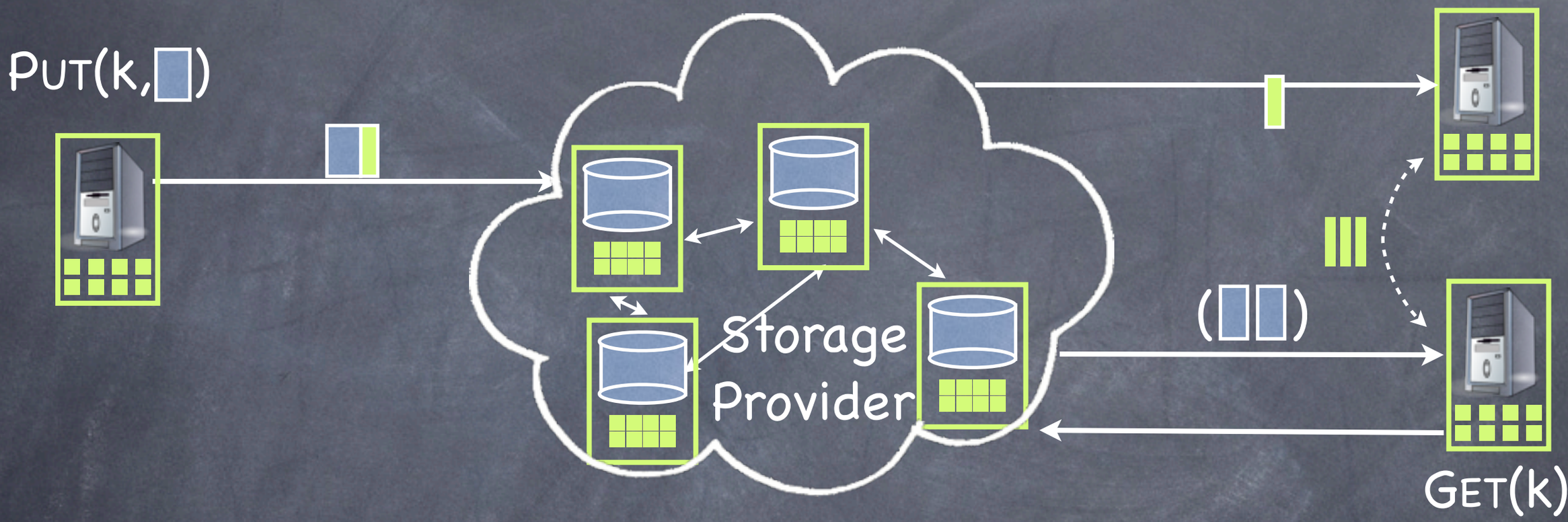
Depot in a nutshell



Preventing omission, reordering

- Add metadata to PUTs
- Add local state to nodes
- Add checks on received metadata

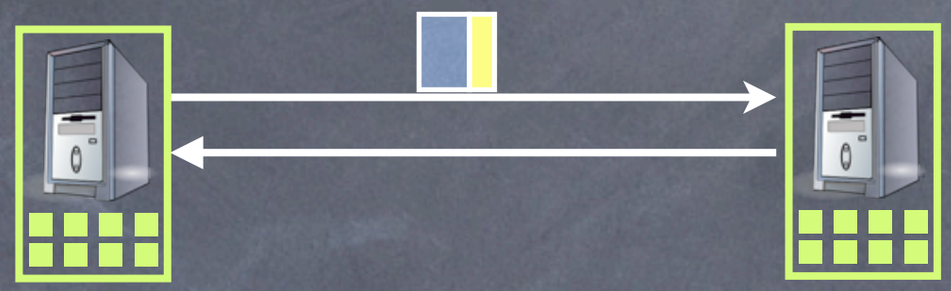
Depot in a nutshell

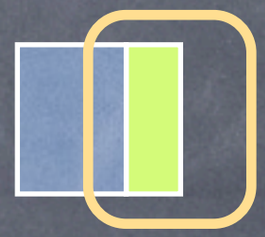


Preventing omission, reordering

- Add metadata to PUTs
- Add local state to nodes
- Add checks on received metadata

Protecting Consistency





(1) Update metadata

- {nodeID, key, H(value), LocalClock, History}$_{nodeID}$

(2) Nodes store update metadata

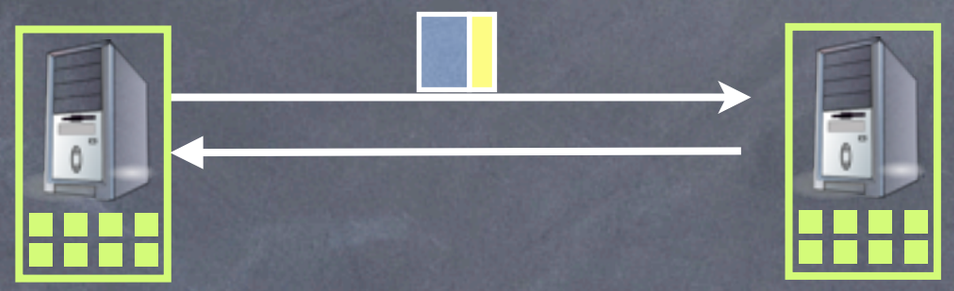
- Logically: Store all previous updates
  [See paper for garbage collection]

Protecting Consistency

(3) Local checks

- Accept an update u created by N if
  - No omissions
    - All updates in u's History are also in local state
  - Don't modify history
    - u is newer than any prior update by N

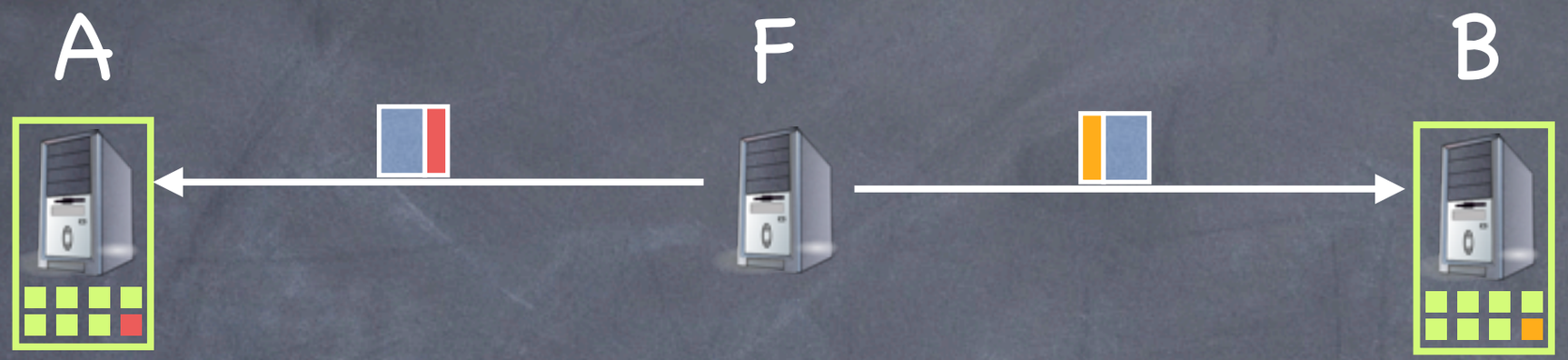
Protecting Consistency



(3) Local checks

- Accept an update u created by N if
  - No omissions
    - All updates in u's History are also in local state
  - Don't modify history
    - u is newer than any prior update by N

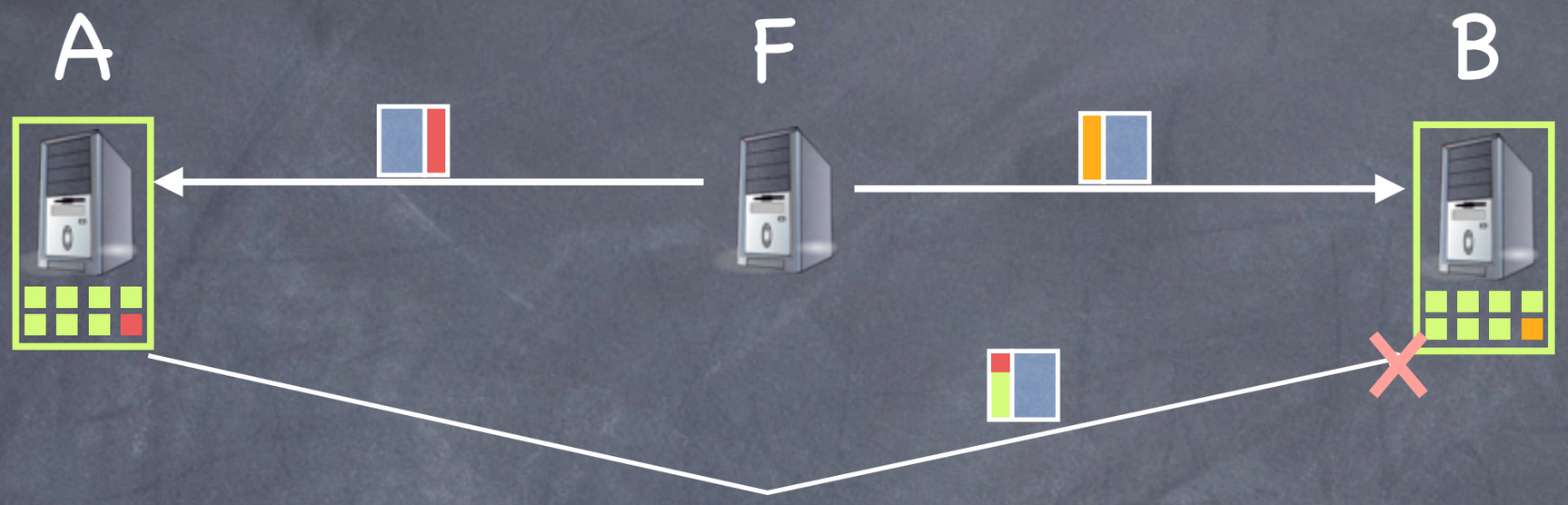
Faults can cause forks



Fork:

- Expose inconsistent views to different nodes
- Each node's view locally consistent

Faults can cause forks



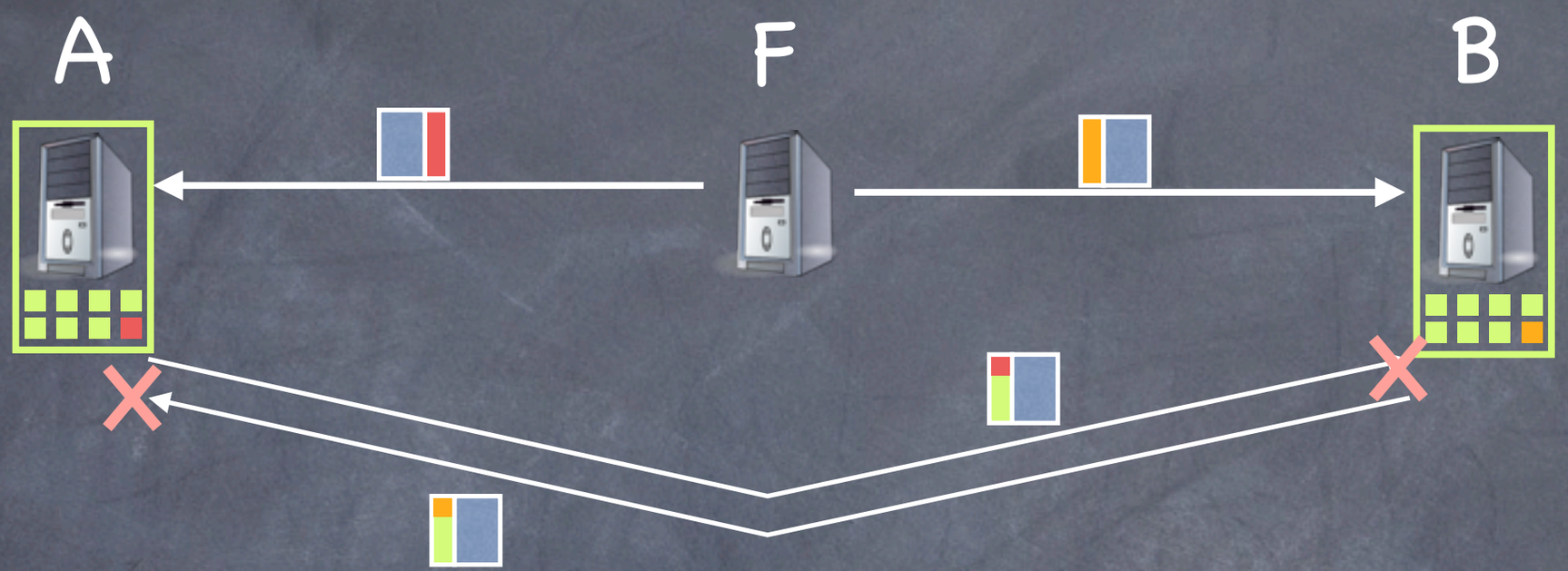
Forks partition correct nodes

- Correct nodes' future updates tainted
- Receiver's update checks fail

Forks prevent eventual consistency

- Inconsistently tainted nodes cannot communicate

Faults can cause forks



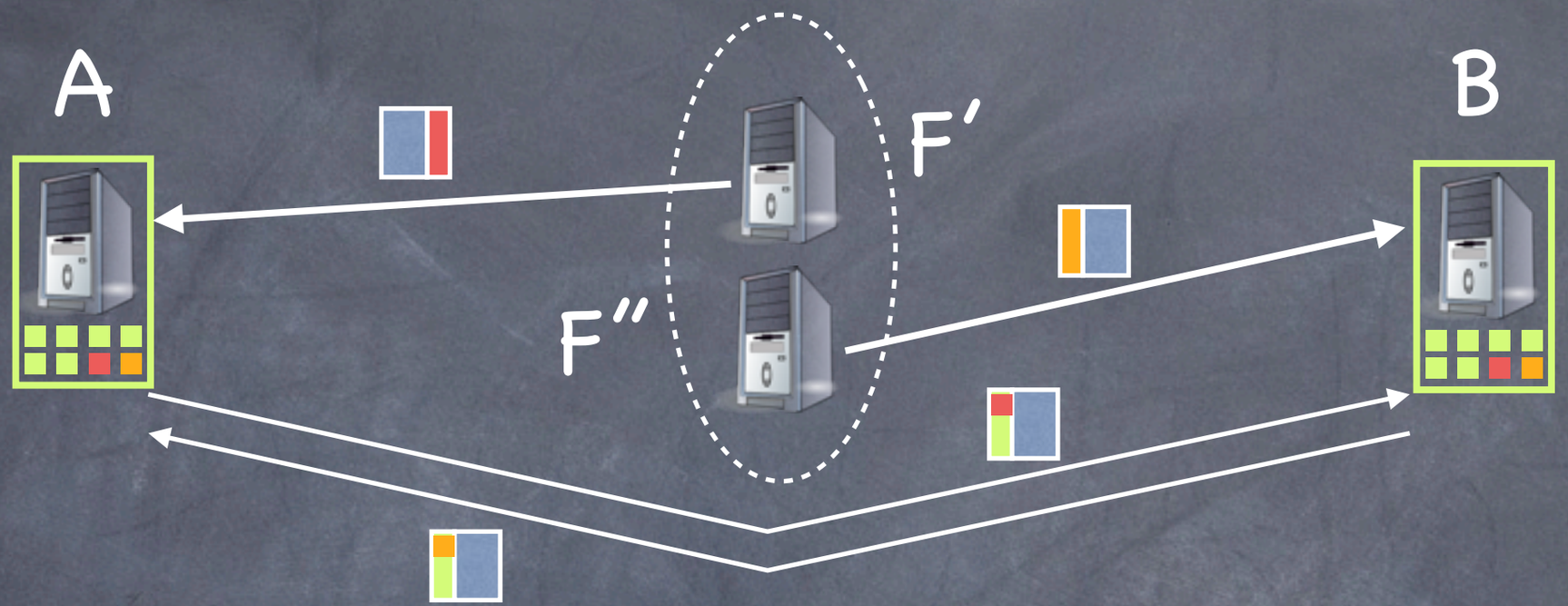
Forks partition correct nodes

- Correct nodes' future updates tainted
- Receiver's update checks fail

Forks prevent eventual consistency

- Inconsistently tainted nodes cannot communicate

Join forks for eventual consistency



Convert faults into concurrency

- Faulty node --> Two (correct) virtual nodes
- Correct nodes can accept subsequent updates
- Correct nodes can evict faulty node

Faults v. Concurrency

Converting faults into concurrency

- Allows correct nodes to converge

Concurrency can introduce conflicts

- Conflict: Concurrent updates to same object
- Problem not introduced by Depot
  - Already possible due to decentralized server
  - Applications built for high availability (such as Amazon S3) allow concurrent writes
- Depot exposes conflicts to applications
  - GET returns set of most recent concurrent updates

Summary: Basic Protocol

- Protect safety
  - Local checks
- Protect liveness
  - Joining forks
  - Reduce failures to concurrency
- Fork-join-causal consistency
  - A novel consistency semantics
  - Suitable for environments with minimal trust

Rest of the talk

I. How does Depot work?

II. What properties does Depot provide?

III. How much does it cost?

Depot Properties

Dimension	Safety/ Liveness	Property	Correct Nodes Required
Consistency	Safety	Fork-Join Causal	Any Subset
	Safety	Bounded Staleness	Any Subset
	Safety	Eventual Consistency (s)	Any Subset
Availability	Liveness	Eventual consistency (l)	Any Subset
	Liveness	Always write	Any Subset
	Liveness	Always exchange	Any Subset
	Liveness	Read availability/ durability	A correct node has data
Integrity	Safety	Only auth. PUT	Any Subset
Eviction	Safety	Valid eviction	Any Subset

GET Availability, Durability

Ideal "Trust Only Yourself"

- Can't reach that goal

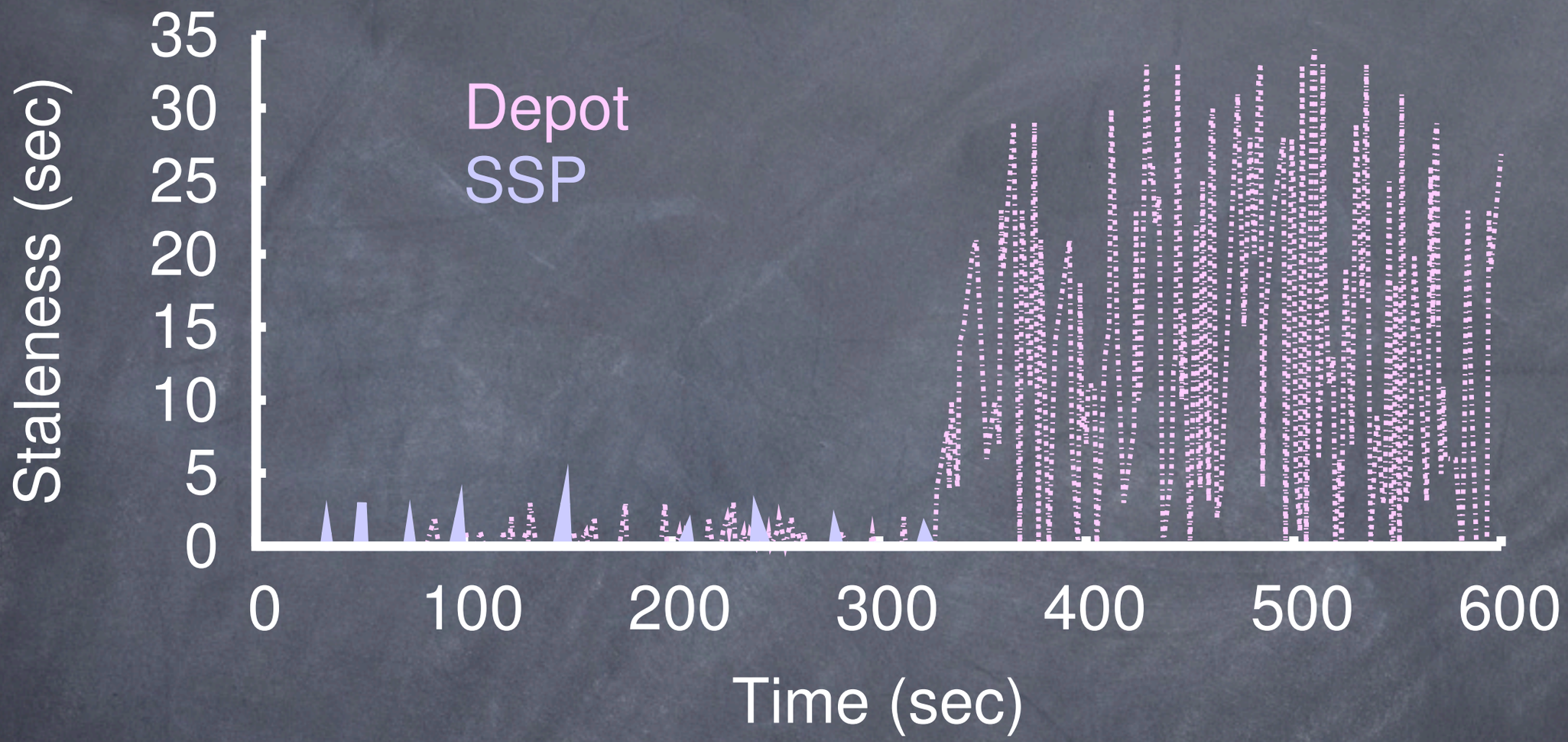
Depot

1. Minimize required number of correct nodes
   - Data can safely flow via any path
   - If any correct node has data, GET eventually succeeds
2. Make it likely a correct node has data
   - SSP replicates to multiple servers
   - Additional replication to protect against total SSP failure

Contingency Plan

- Protect against correlated SSP failure
  - Availability event or permanent failure
- Key: Storage servers are untrusted
  - Pick any node with low correlation to SSP
  - Prototype:
    - Client that issues PUT keeps copy of data
    - Gossiped update metadata sufficient to route GET requests when SSP unavailable
  - Alternatives:
    - Private cloud storage node (e.g., Eucalyptus/Walrus)
    - Another external SSP

Depot Tolerates SSP Failure



Complete cloud failure at 300s

- Depot's GET, PUT continue
- Depot's staleness increases

Rest of the talk

I. How does Depot work?

II. What properties does Depot provide?

III. How much does Depot cost?

- Latency, resources, dollars

How much does it cost?

Latency cost

- Compare GET and PUT latencies

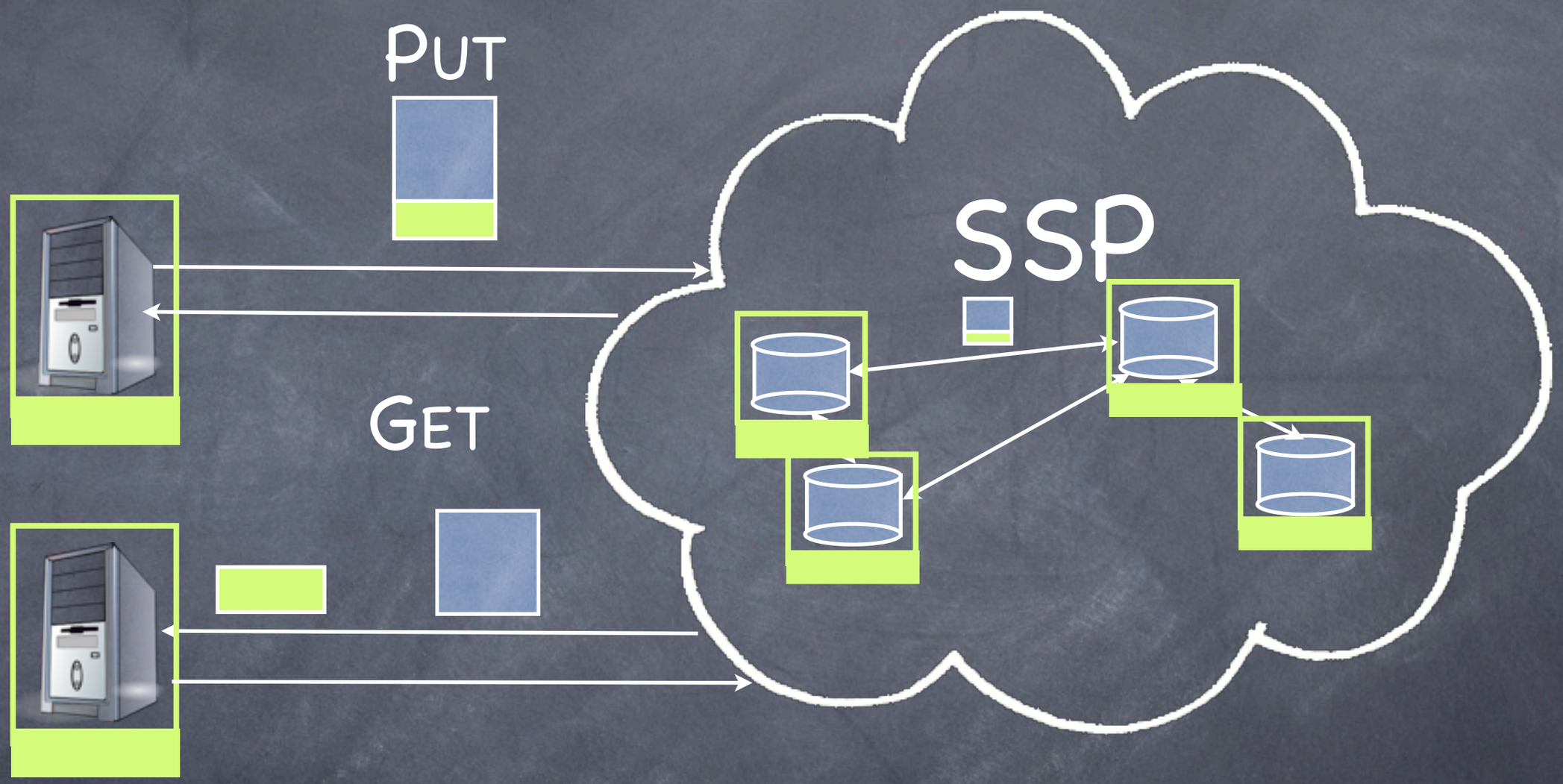
Resource cost

- Processing (client and server)
- Network (client-server and server-server)
- Storage (client and server)

Dollar cost

- Weighted Processing + Network + Storage

Sources of overhead in Depot



Sources of overhead in Depot

metadata =
signature
+ partial VV
+ history hash

PUT

SSP

GET

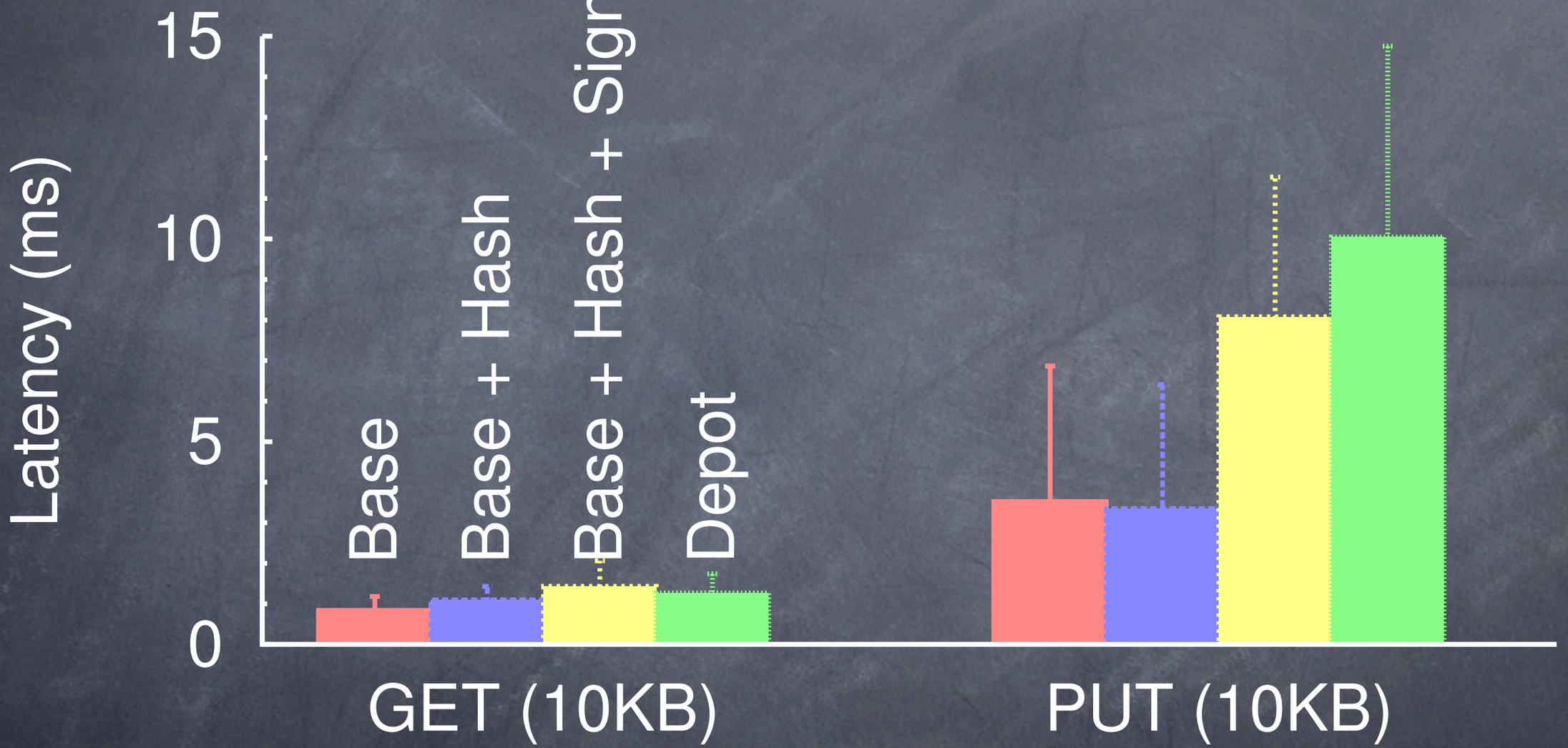
data check =
SHA256 check

metadata check =
SHA256 check
+ RSA verify
+ history check

Setup

- 12 nodes on local Emulab
  - 8 clients + 4 servers
    - Quad core Intel Xeon X3220 2.40 GHz processor
    - 8 GB RAM
    - two local 7200 RPM disk
    - 1 Gbps link
- Each client issues 1 request/sec
  - Measure latency, per-request cost
- Emulate traditional cloud storage
  - Servers implemented Depot without any checks
  - Clients don't receive any metadata

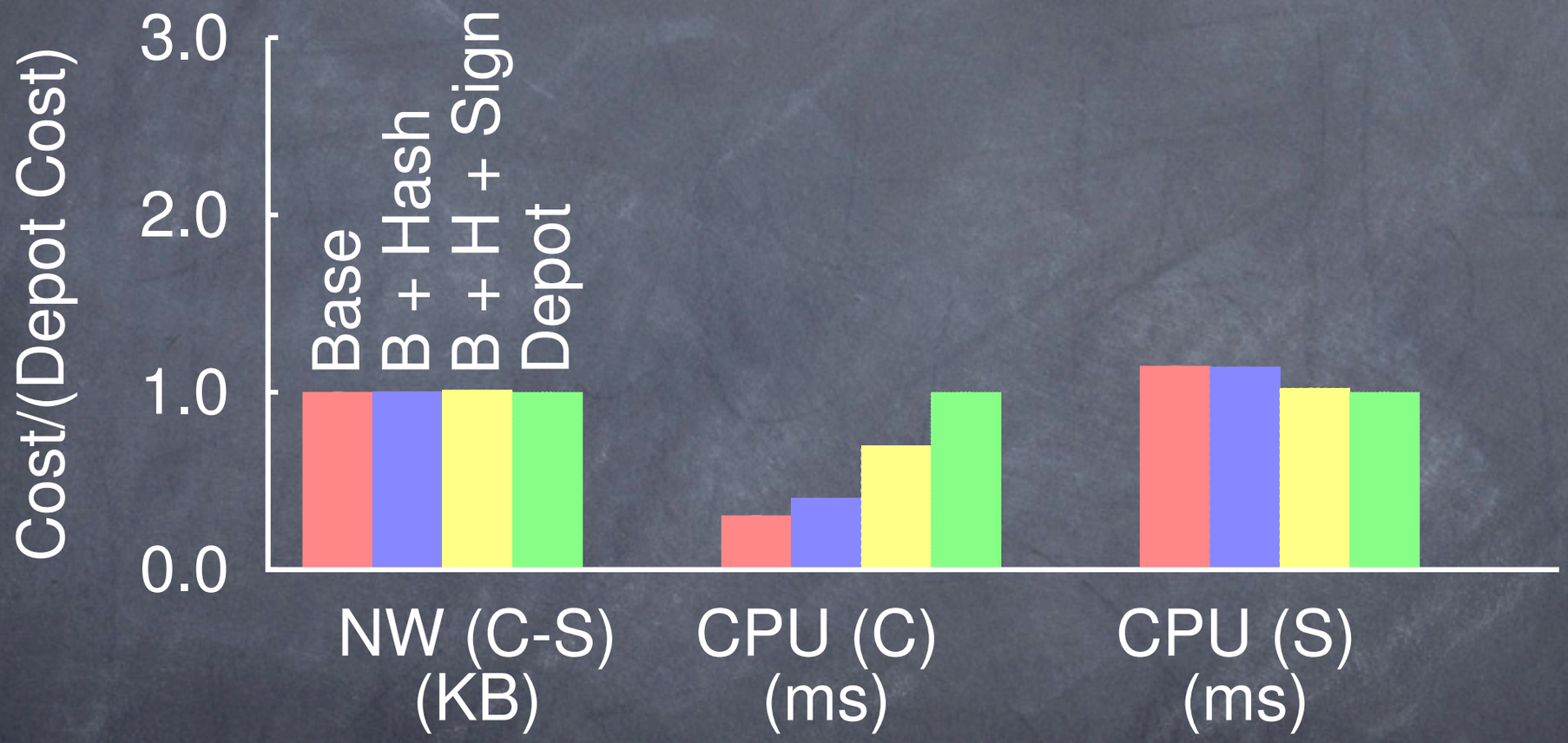
Depot adds little latency



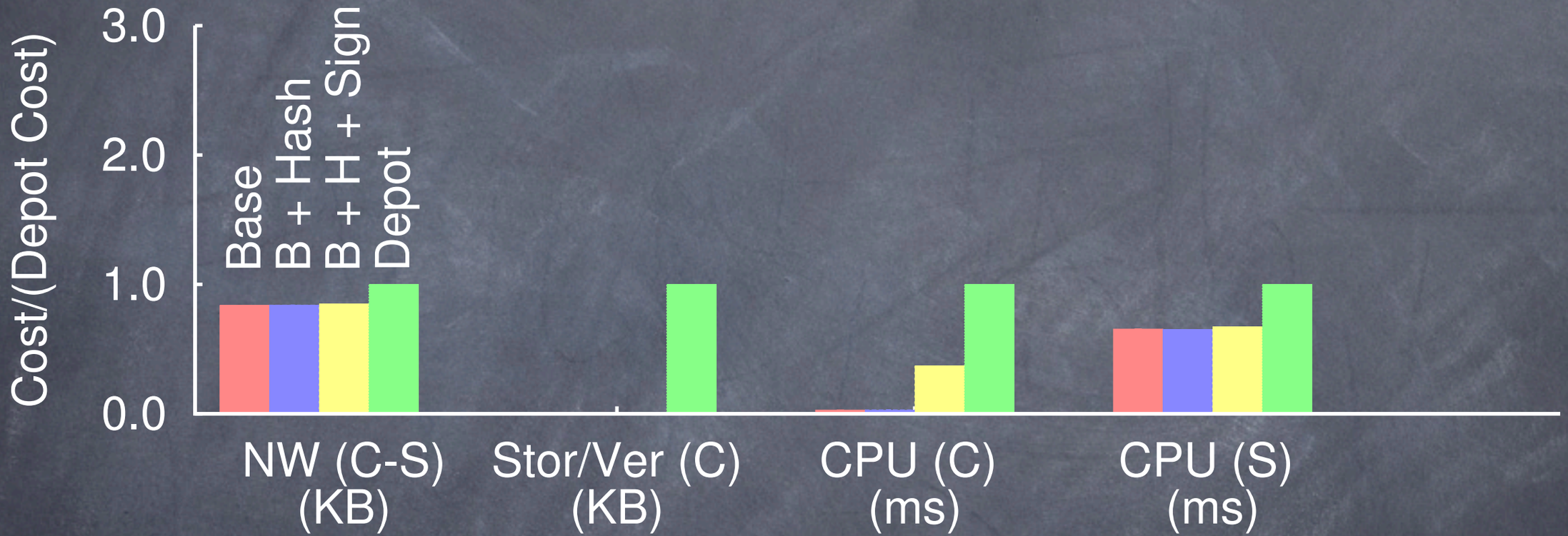
Depot overheads on GETs are very small

Overheads on PUTs are modest

Depot GET overheads are modest



Depot PUT overheads are modest



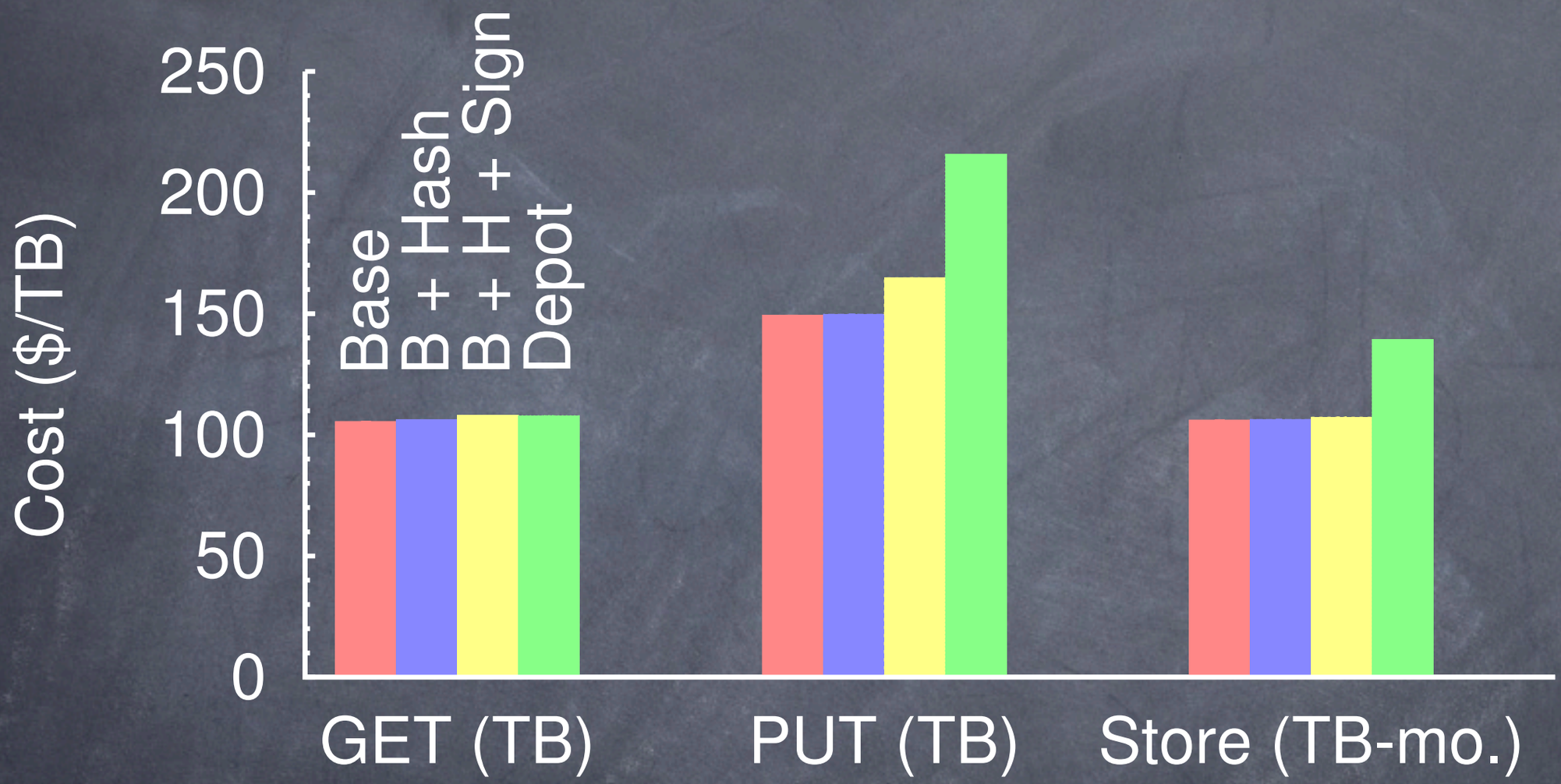
- Metrics that didn't change are omitted.
  - E.g. Storage(S), NW(S-S)
- Metadata transfer=>NW cost
- Metadata verification=>CPU cost
- Metadata store=>Storage cost

Cost Model

Client-Server NW Bandwidth	$0.10/GB
Server-Server NW Bandwidth	$0.01/GB
Disk Storage	$0.025 GB/month
CPU Processing	$0.10/hour

Based (loosely) on current cloud pricing

Depot dollar costs are small



Related Work

- Fork-based systems
  - SUNDR [Li et al. OSDI 2004]
  - BFT2F [Li and Mazieres NSDI 2007]
  - SPORC [Feldman et al. OSDI 2010]
  - Venus [Shraer et al. CCSW 2010]
- Quorums and state machines
  - BQS [Malkhi and Reiter Dist. Comp. 1998]
  - PBFT [Castro and Liskov TOCS 2002]
  - Q/U [El-Malek et al. SOSP 2005]
  - HQ [Cowling and Liskov OSDI 2006]
  - Zyzzyva [Kotla et al. SOSP 2007]
- Many others

Conclusion

- Depot: Cloud storage with minimal trust
- Radical fault tolerance
  - Any node could fail in any way
  - Eliminate trust for consistency, staleness, update exchange, eviction, ...
    - Any subset of correct clients get these properties
  - Minimize trust for GET availability, durability
    - GET succeeds if any correct, reachable node has data
    - Protocol hooks to make this likely