An Information-Theoretic Perspective of Consistent Distributed Storage

Viveck R. Cadambe

Pennsylvania State University

Joint with Prof. Zhiying Wang (UCI) Prof. Nancy Lynch (MIT), Prof. Muriel Medard (MIT) and Dr. Peter Musial (EMC Corporation)

Distributed Storage Systems

Distributed Storage Systems

- Failure tolerance, Low storage costs, Fast reads and writes

Distributed Storage Systems

- Failure tolerance, Low storage costs, Fast reads and writes
- This talk: Consistency
  - High-level principle: read the “latest” value stored in the system

Distributed Storage Systems

- Failure tolerance, Low storage costs, Fast reads and writes

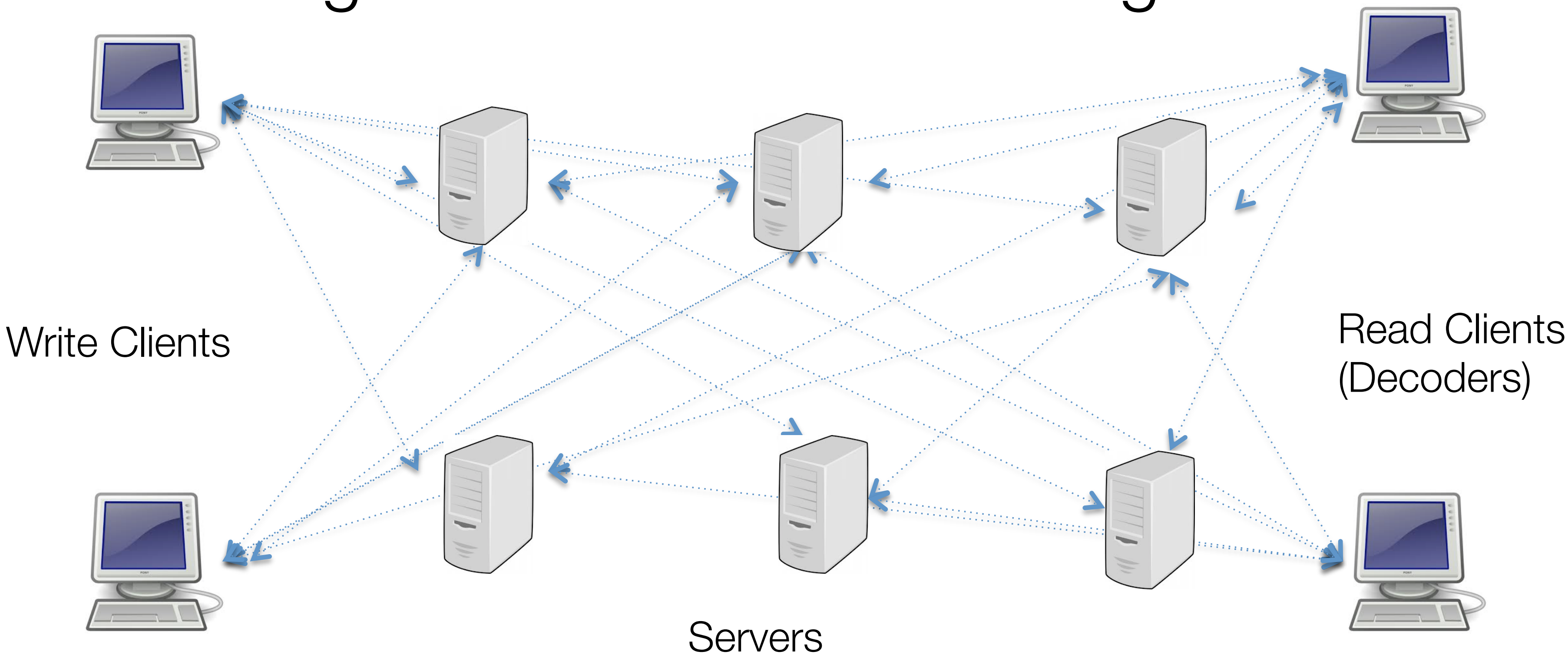
- This talk: Consistency

  - High-level principle: read the “latest” value stored in the system

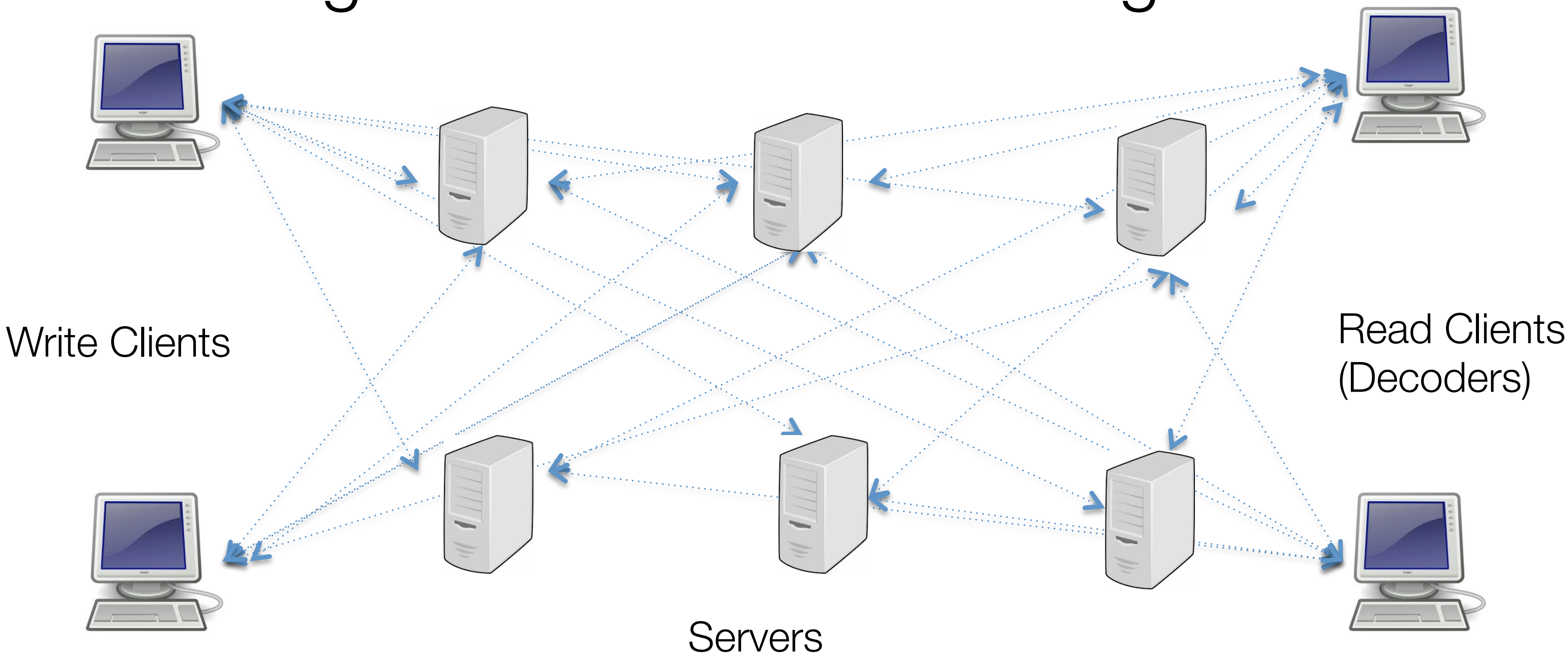
  - Modern key-value stores - Amazon Dynamo DB, Couch DB, Apache Cassandra DB, Google Spanner, Voldermort DB …..

  - Used for transactions, reservation systems, multi-player gaming, social networks, news feeds, distributed computing tasks etc.

High level Distributed Storage Model

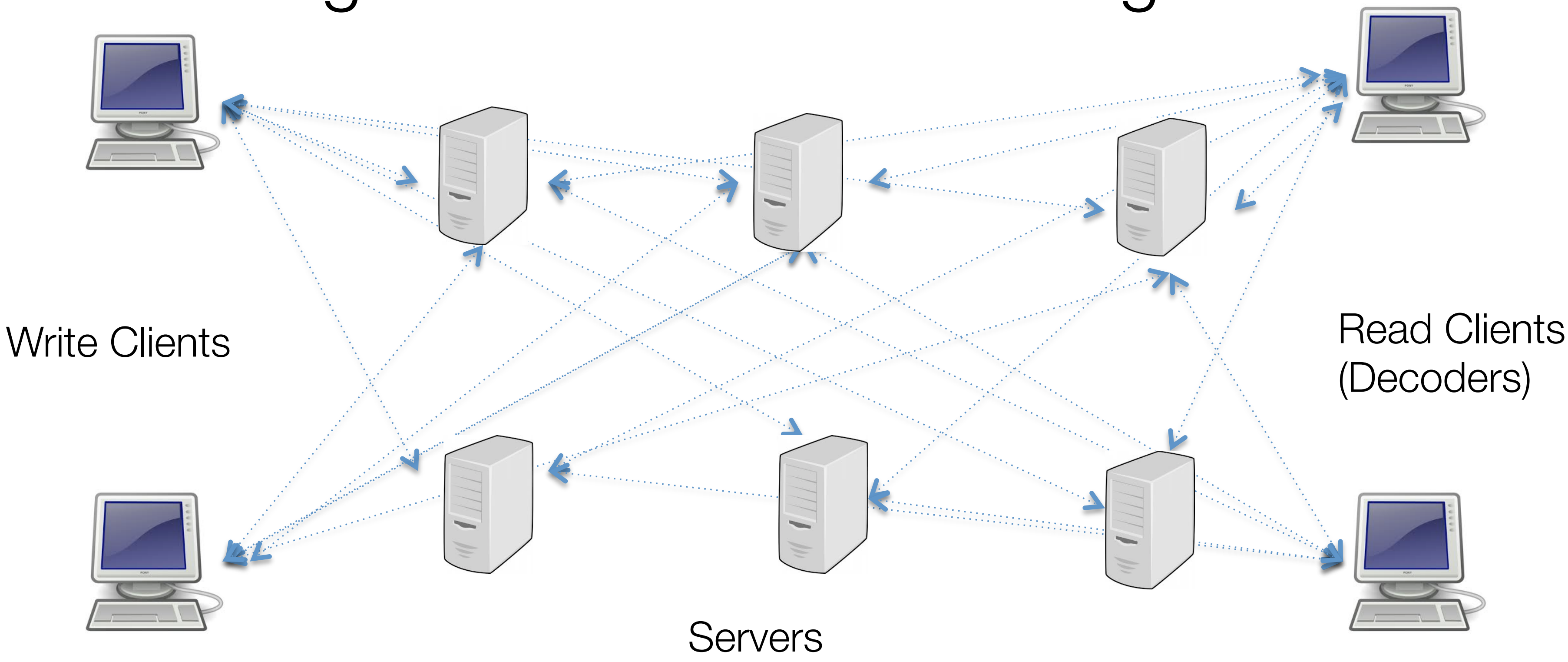


High level Distributed Storage Model



- Asynchrony – packets don’t arrive at all the servers simultaneously
- Distributed nature - nodes do not know which packets have been received by other nodes, or if they have failed.
- Consistency – the reader/decoder needs the latest “possible” version.

High level Distributed Storage Model



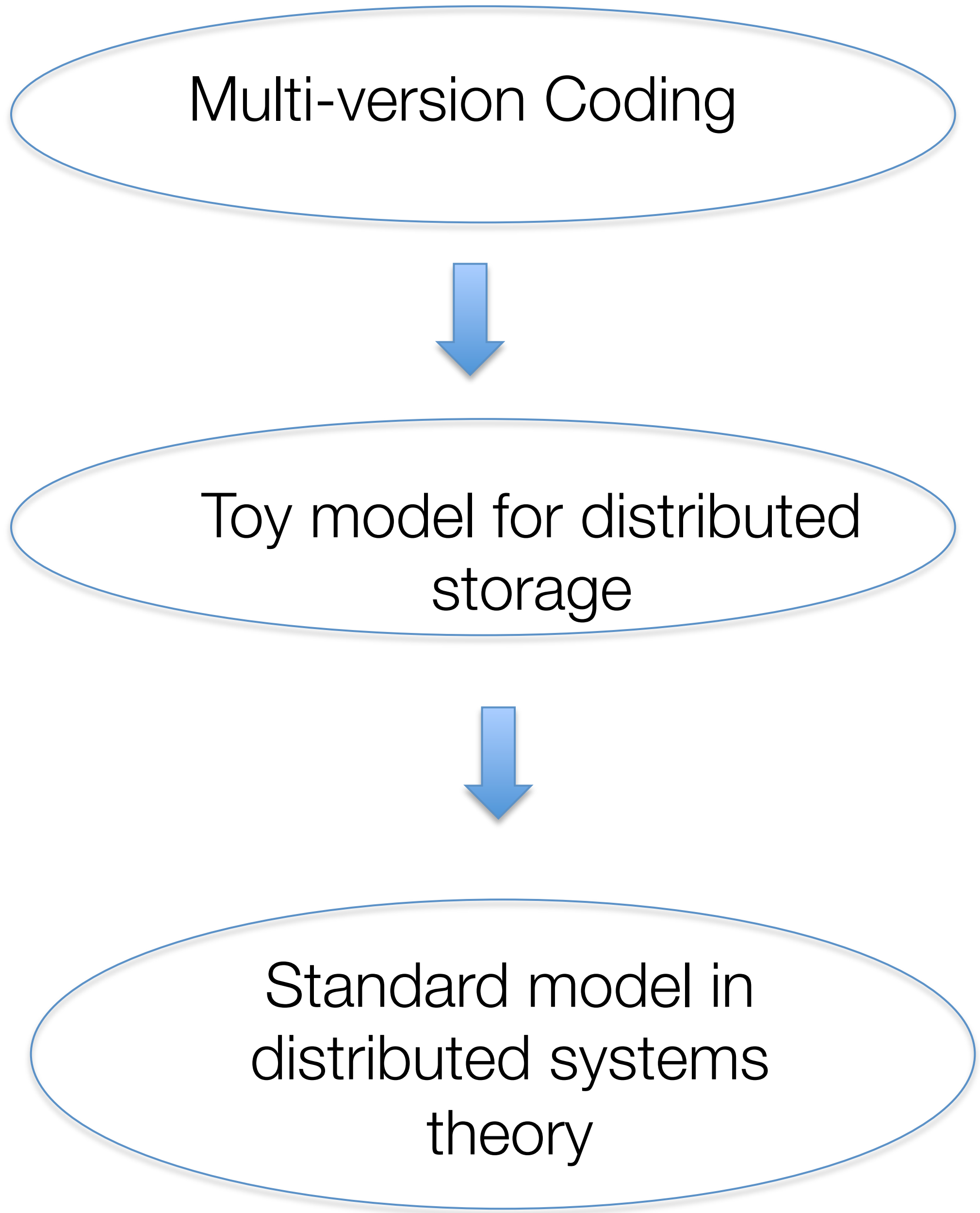
- Asynchrony, Distributed Nature, Consistency

Analytical understanding of storage costs, latency, is very limited
Replication is used in every commercial solution to provide fault tolerance

Standard model in distributed systems theory

Multi-version Coding

Standard model in distributed systems theory


Multi-version Coding
Toy model for distributed storage
Standard model in distributed systems theory

The multi-version coding (MVC) problem

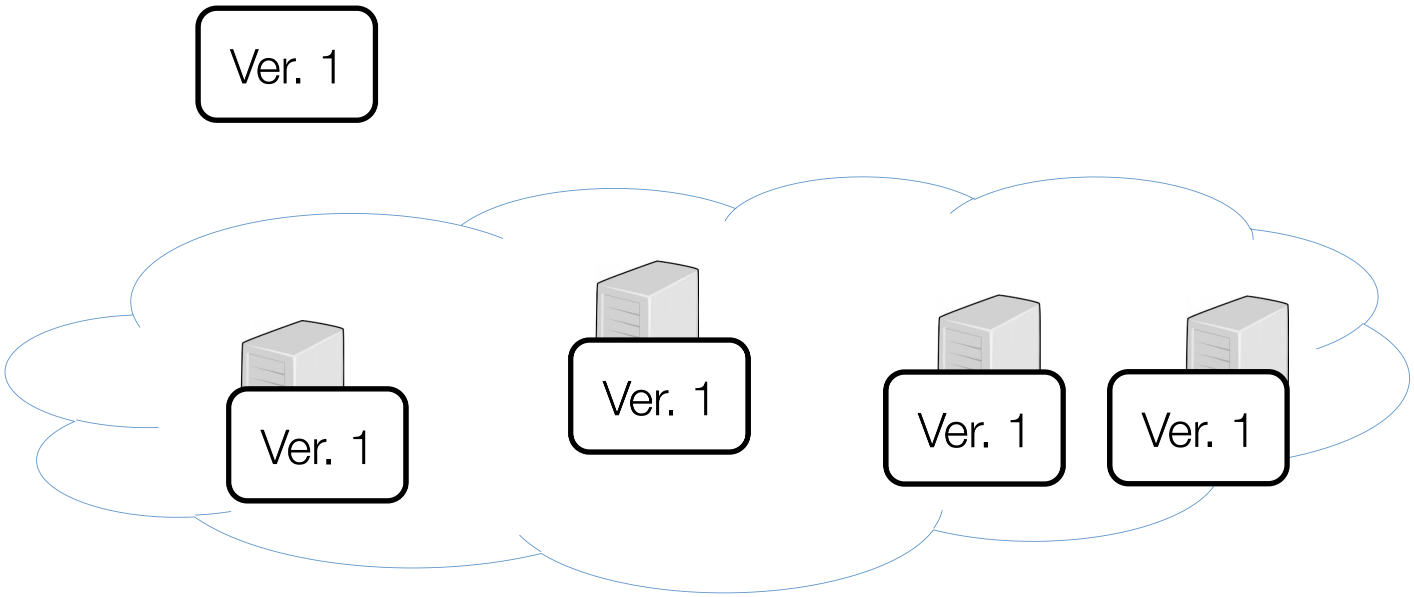
[Wang-C, ISIT, Allerton 2014, arxiv 2015]

As the data gets updated

- *Asynchrony:* all servers may not simultaneously get the new version of the data
- *Distributed nature*: each node is unaware of the versions received by the other nodes
- *Consistency:* A decoder must get the latest possible version of the data

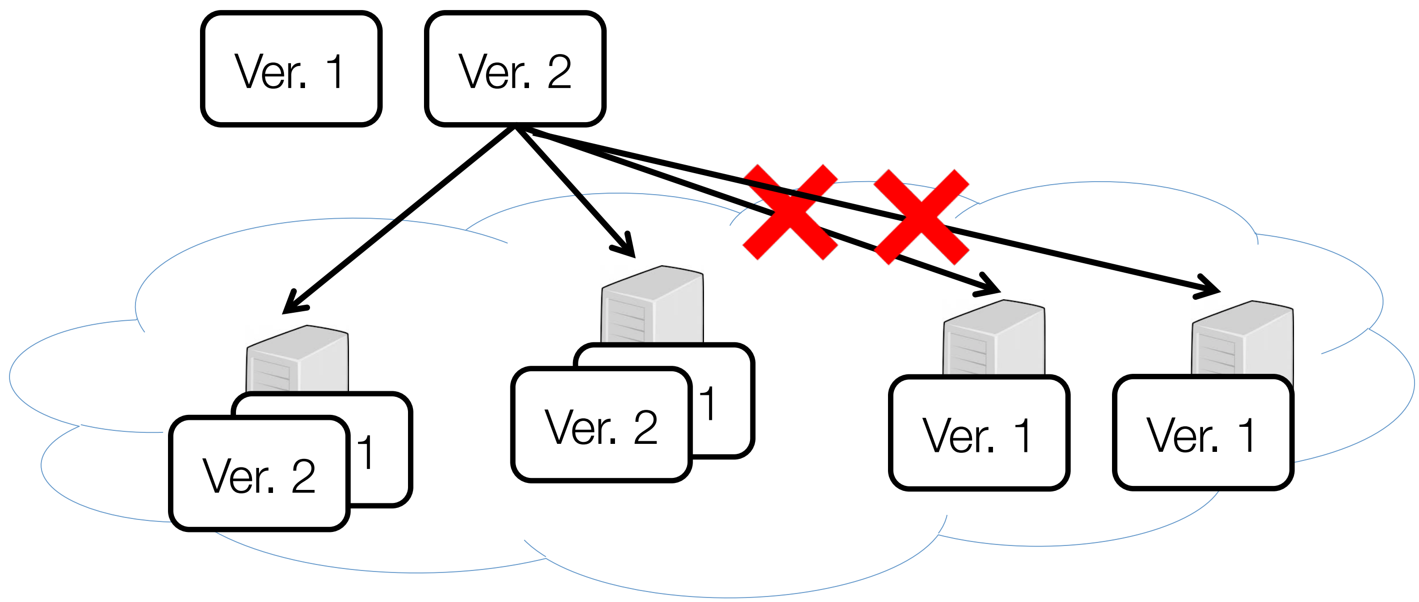
The multi-version coding problem

[Wang-C, ISIT, Allerton 2014, arxiv 2015]



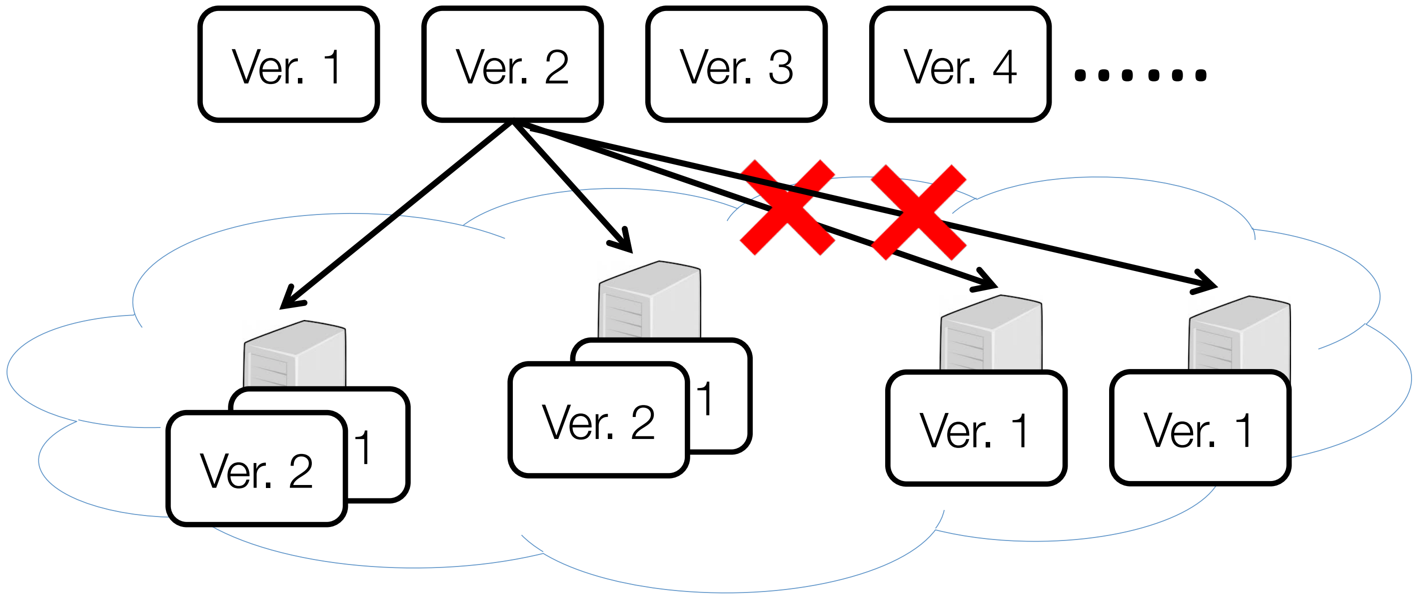
The multi-version coding problem

[Wang-C, ISIT, Allerton 2014, arxiv 2015]



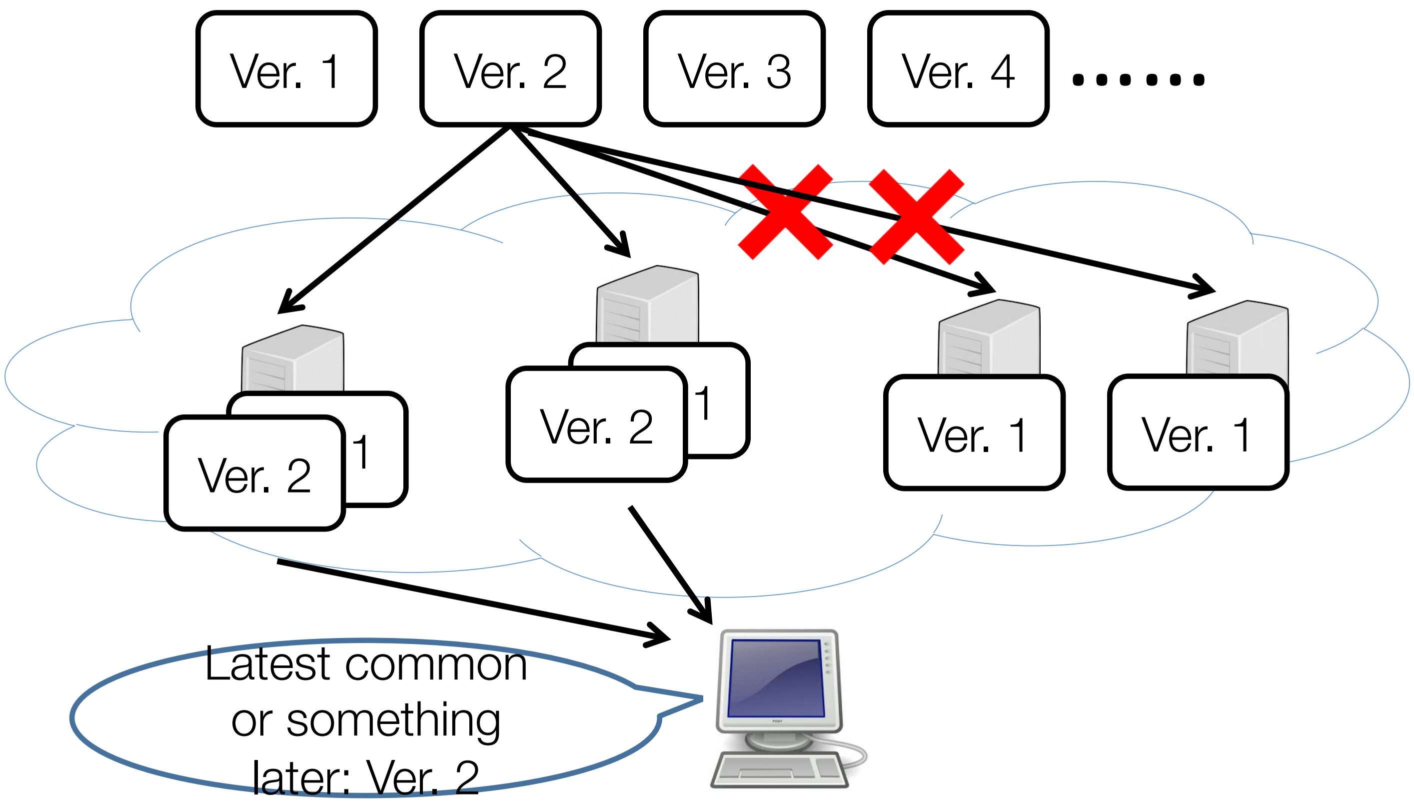
The multi-version coding problem

[Wang-C, ISIT, Allerton 2014, arxiv 2015]



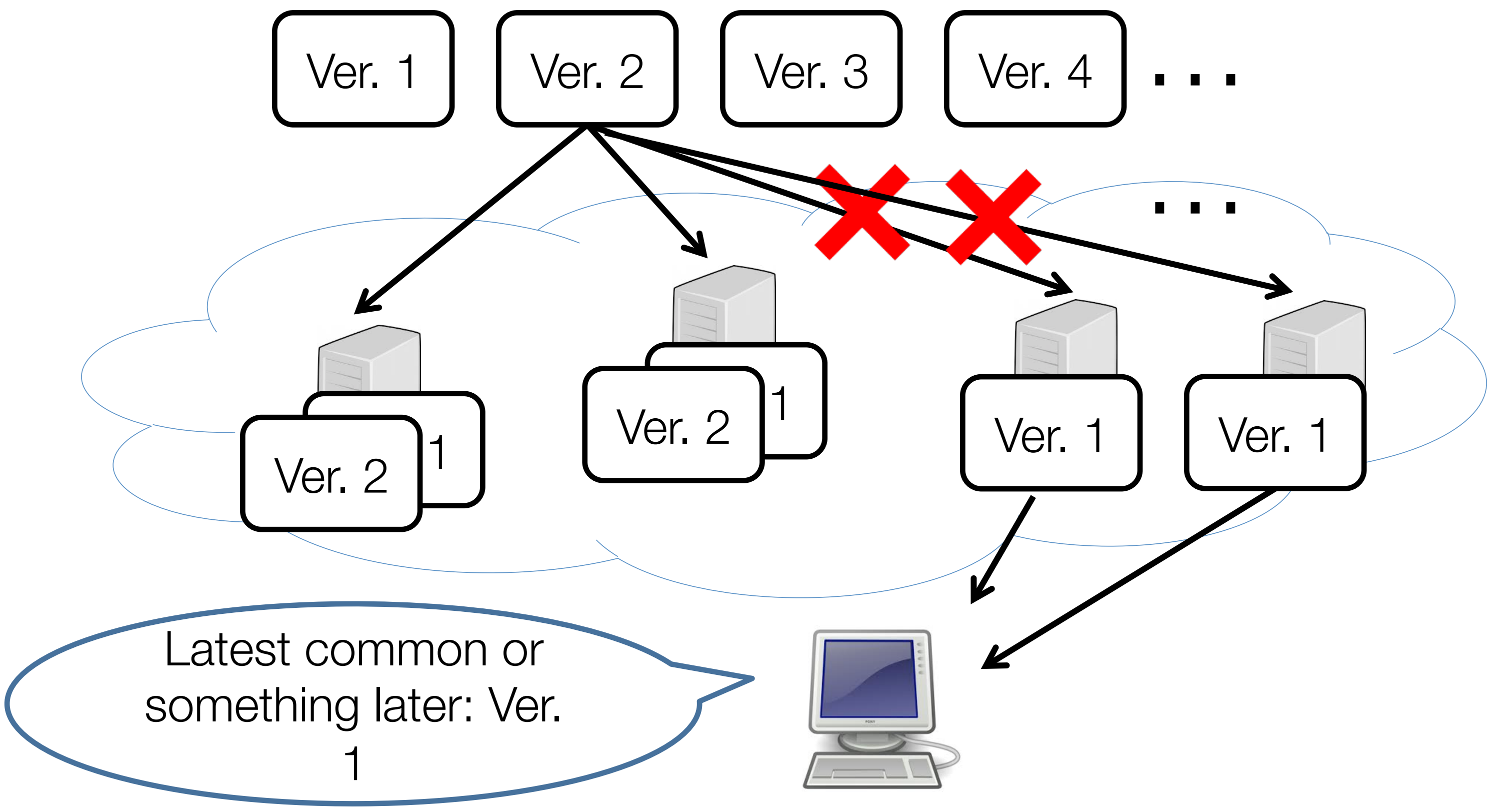
The multi-version coding problem

[Wang-C, ISIT, Allerton 2014, arxiv 2015]



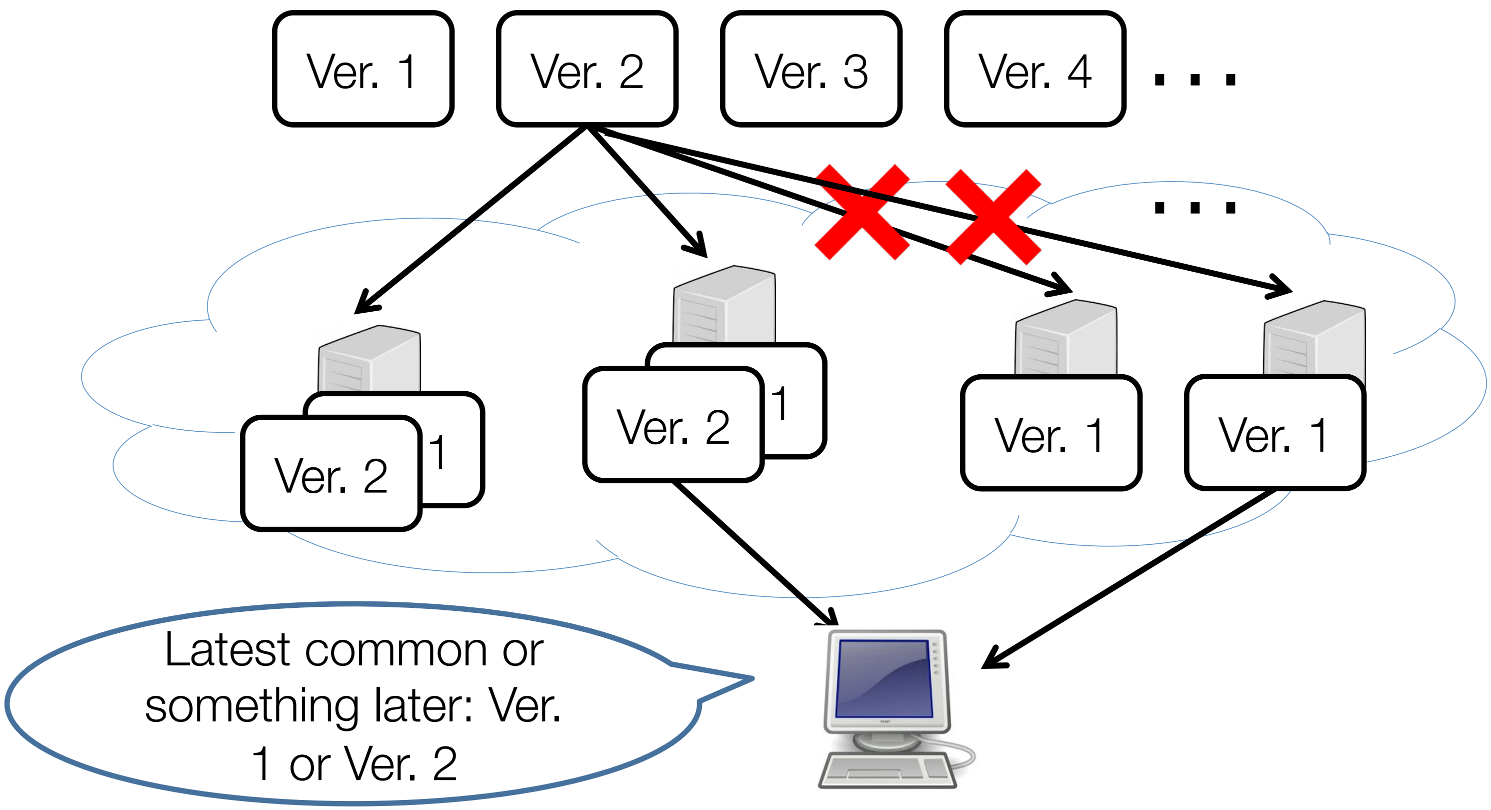
The multi-version coding problem

[Wang-C, ISIT, Allerton 2014, arxiv 2015]



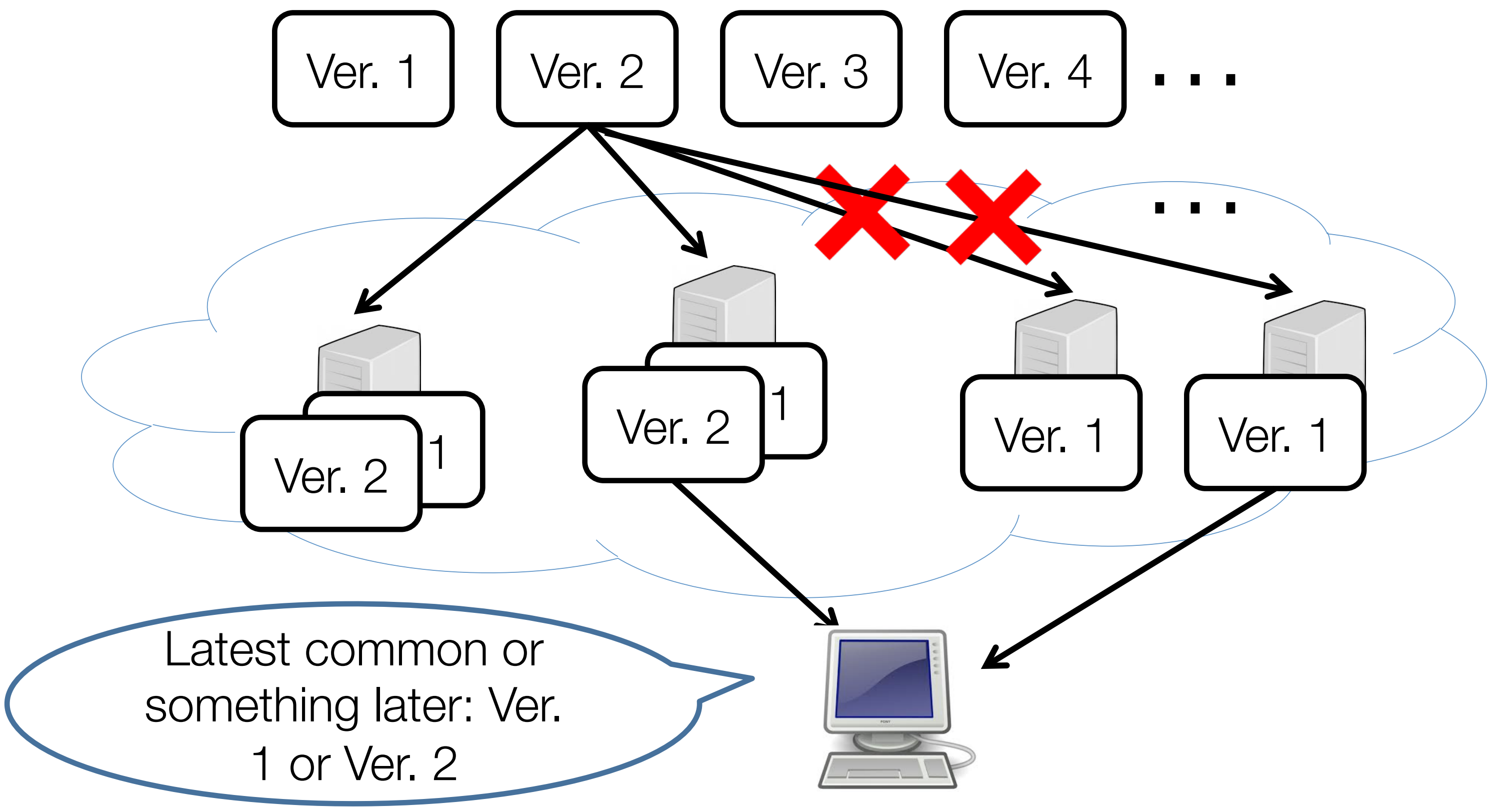
The multi-version coding problem

[Wang-C, ISIT, Allerton 2014, arxiv 2015]



The multi-version coding problem

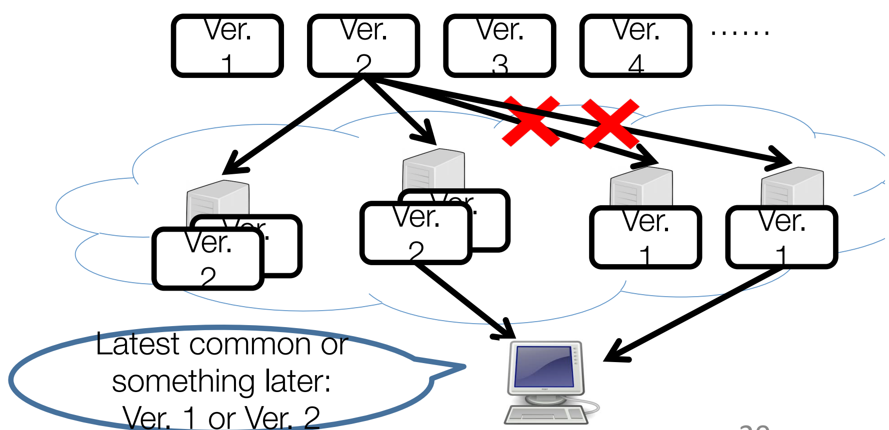
[Wang-C, ISIT, Allerton 2014, arxiv 2015]



In general, client connects to c servers, demands the latest common version among v versions

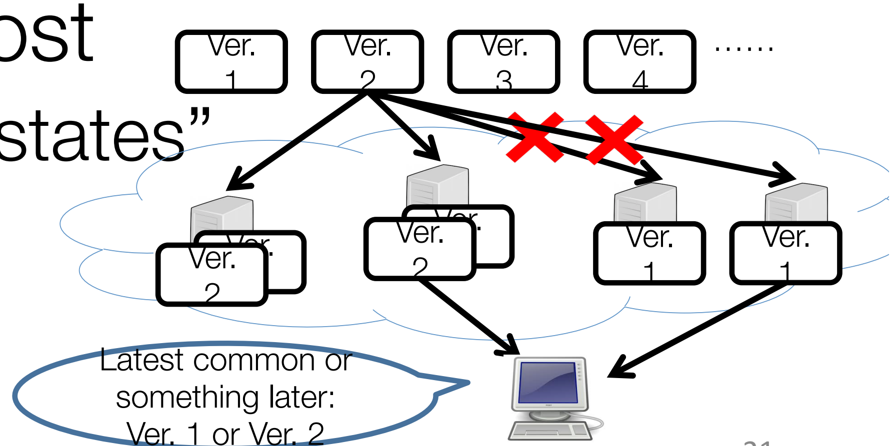
The multi-version coding problem

- n servers
- v versions
- c connectivity



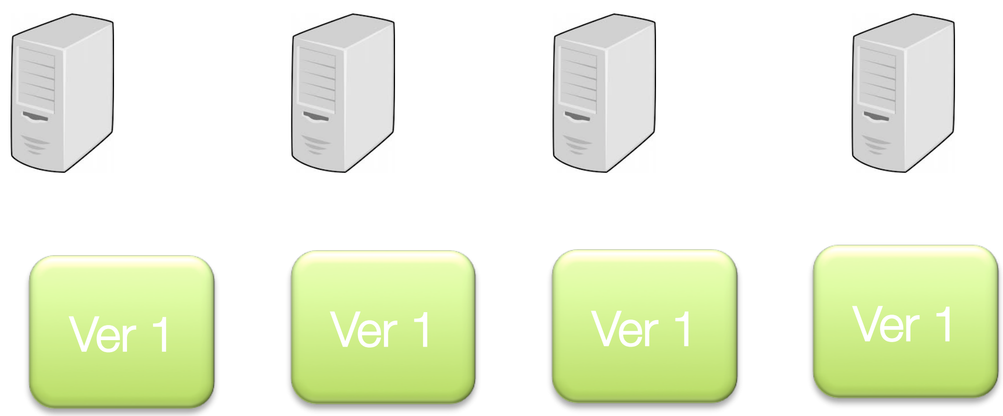
The multi-version coding problem

- *n* servers
- *v* versions
- *c* connectivity
- Goal: decode the latest common version among the c servers
- Minimize the storage cost
  - Worst case, across all “states”
  - across all servers



Solution 1: Replication

Storage size = size-of-one-version

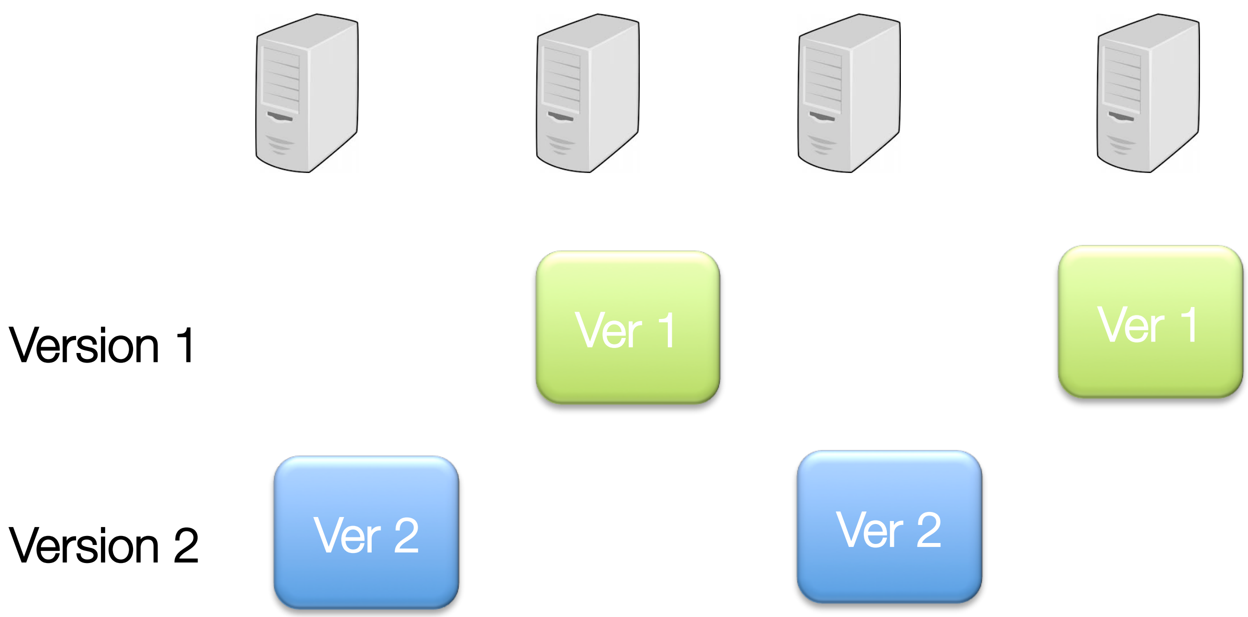


Version 1

Version 2

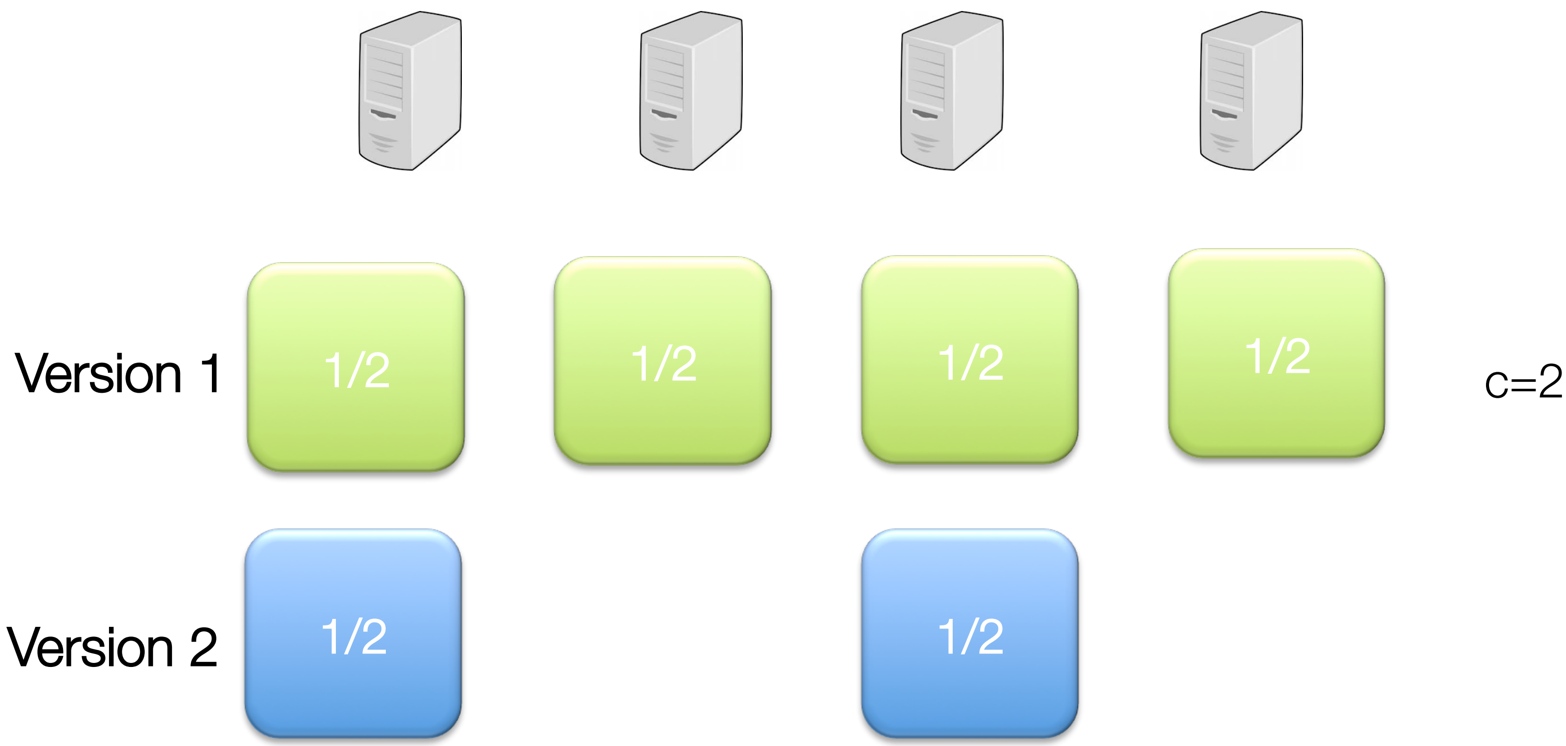
Solution 1: Replication

Storage size = size-of-one-version



Solution 2: MDS code

Storage size = (Number of versions / c)*size-of-one-version
= *v*/c*size-of-one-version



Separate coding across versions.
Each server stores all the versions received.

	Storage Cost Normalized by size-of-value
Replication	1
Naïve MDS codes	v/c

$$v = \text{Number of Versions}$$

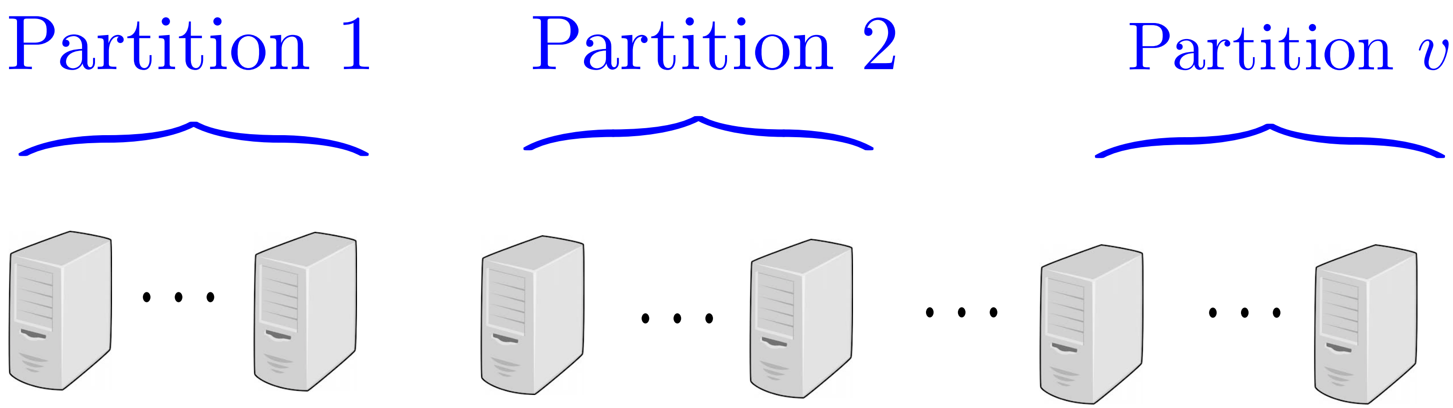
$$c = \text{Connectivity}$$

	Storage Cost Normalized by size-of-value
Replication	1
Naïve MDS codes	v/c
Constructions	$\frac{1}{\lceil c/v \rceil}$
Lower bound	$\frac{v}{c+v-1} - o(\text{size-of-value})$

$$v = \text{Number of Versions}$$

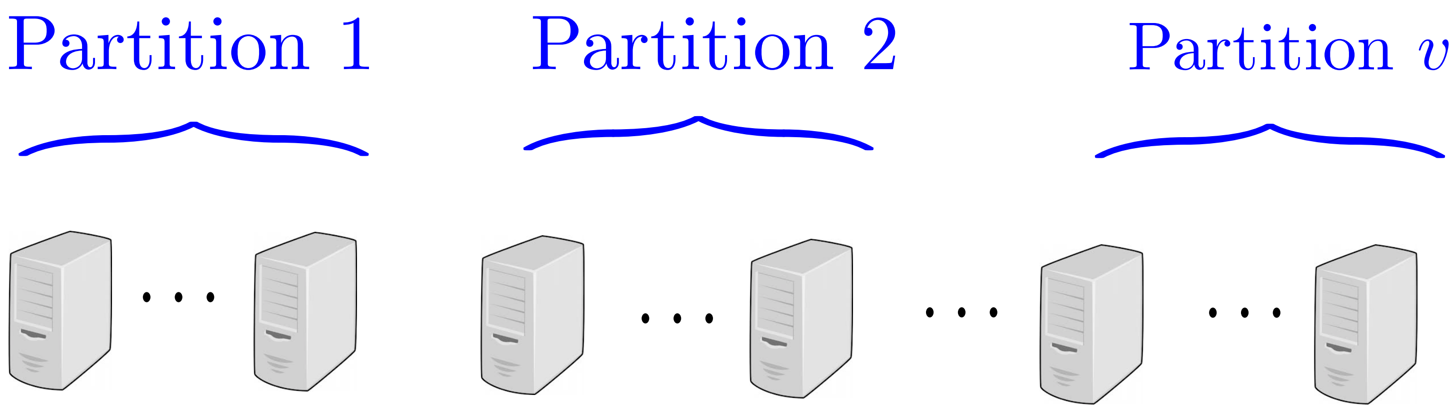
$$c = \text{Connectivity}$$

Achievability



Partition i: Version i is the latest version

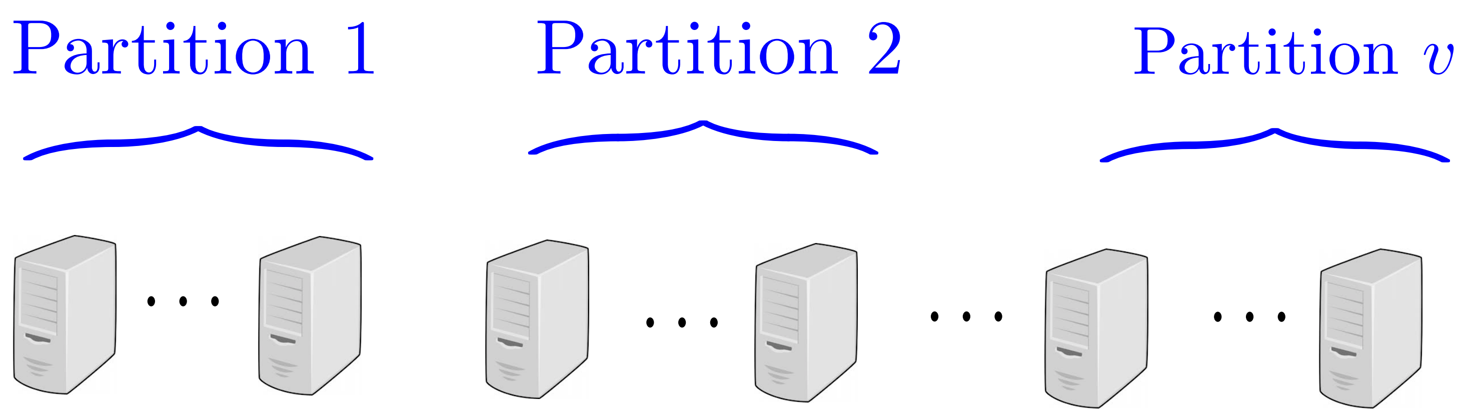
Achievability



Partition i: Version i is the latest version

There is at least one partition with $\lceil c/v \rceil$ servers

Achievability



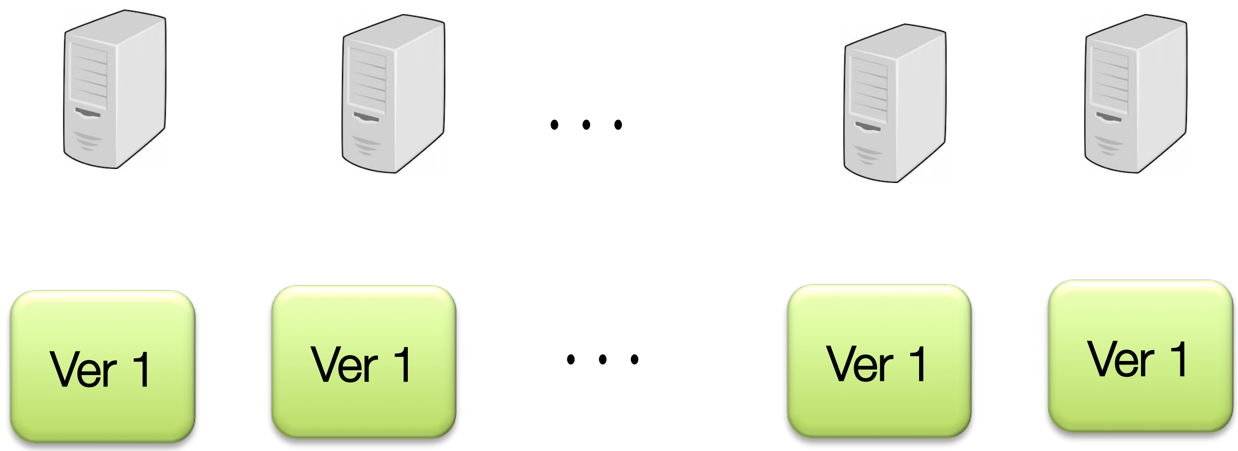
Partition i: Version i is the latest version

There is at least one partition with $\lceil c/v \rceil$ servers

Simple achievable scheme:
Server in partition i stores $\frac{1}{\lceil c/v \rceil}$ of version i

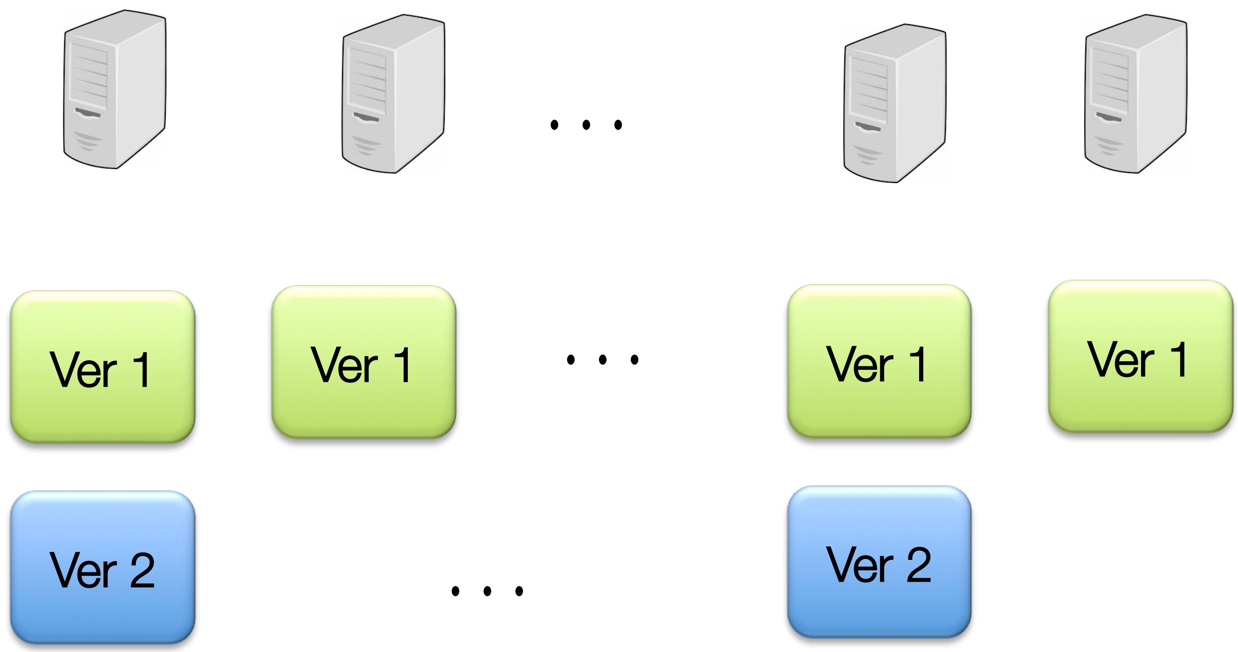
Converse: $v = 2$, storage cost $\gtrsim \frac{2}{c+1}$

Start with c servers



Converse: $v = 2$, storage cost $\gtrsim \frac{2}{c+1}$

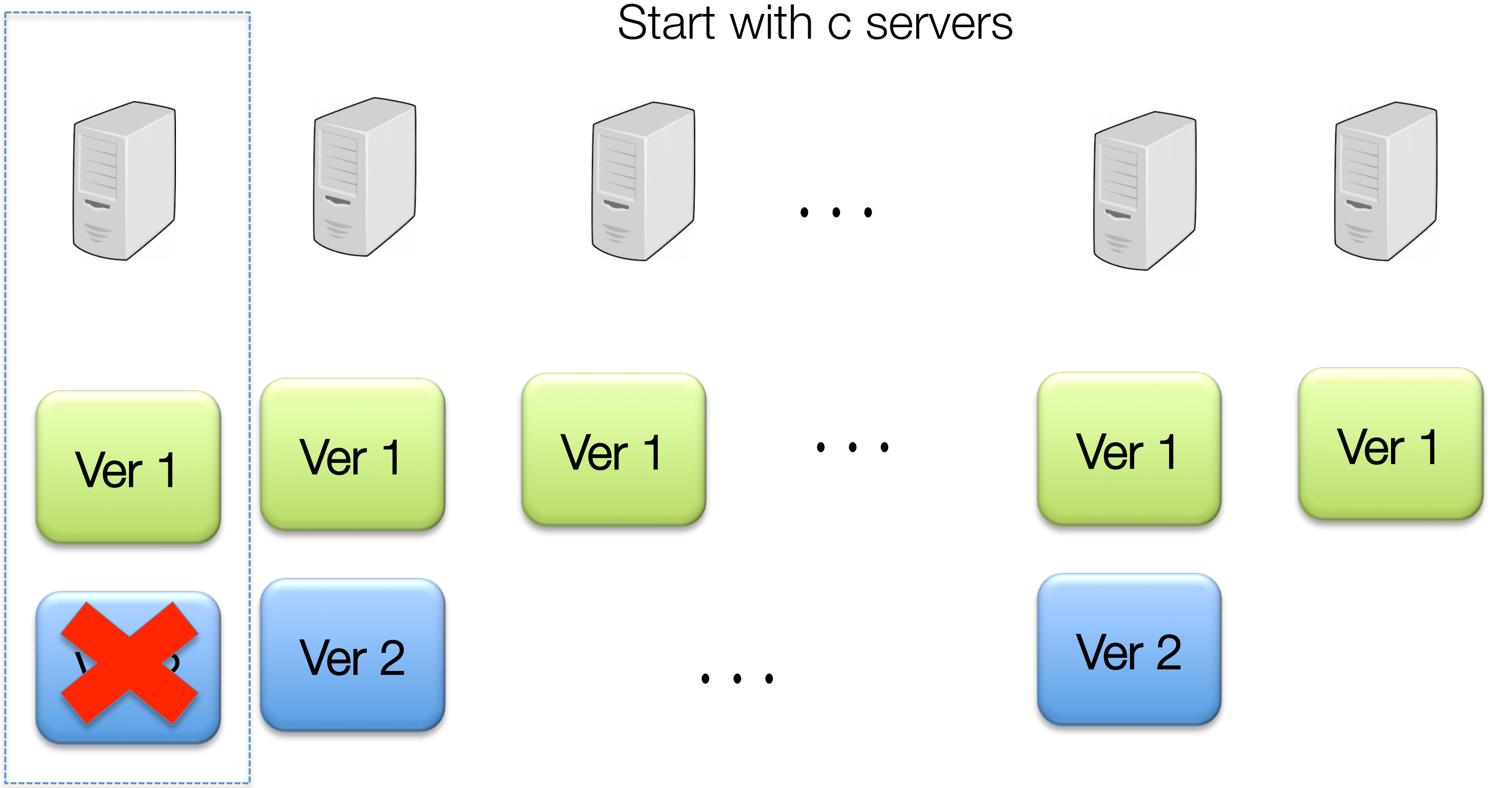
Start with c servers



Propagate version 2 to a minimal set of servers
such that it is decodable

Converse: $v = 2$, storage cost $\gtrsim \frac{2}{c+1}$

Start with c servers



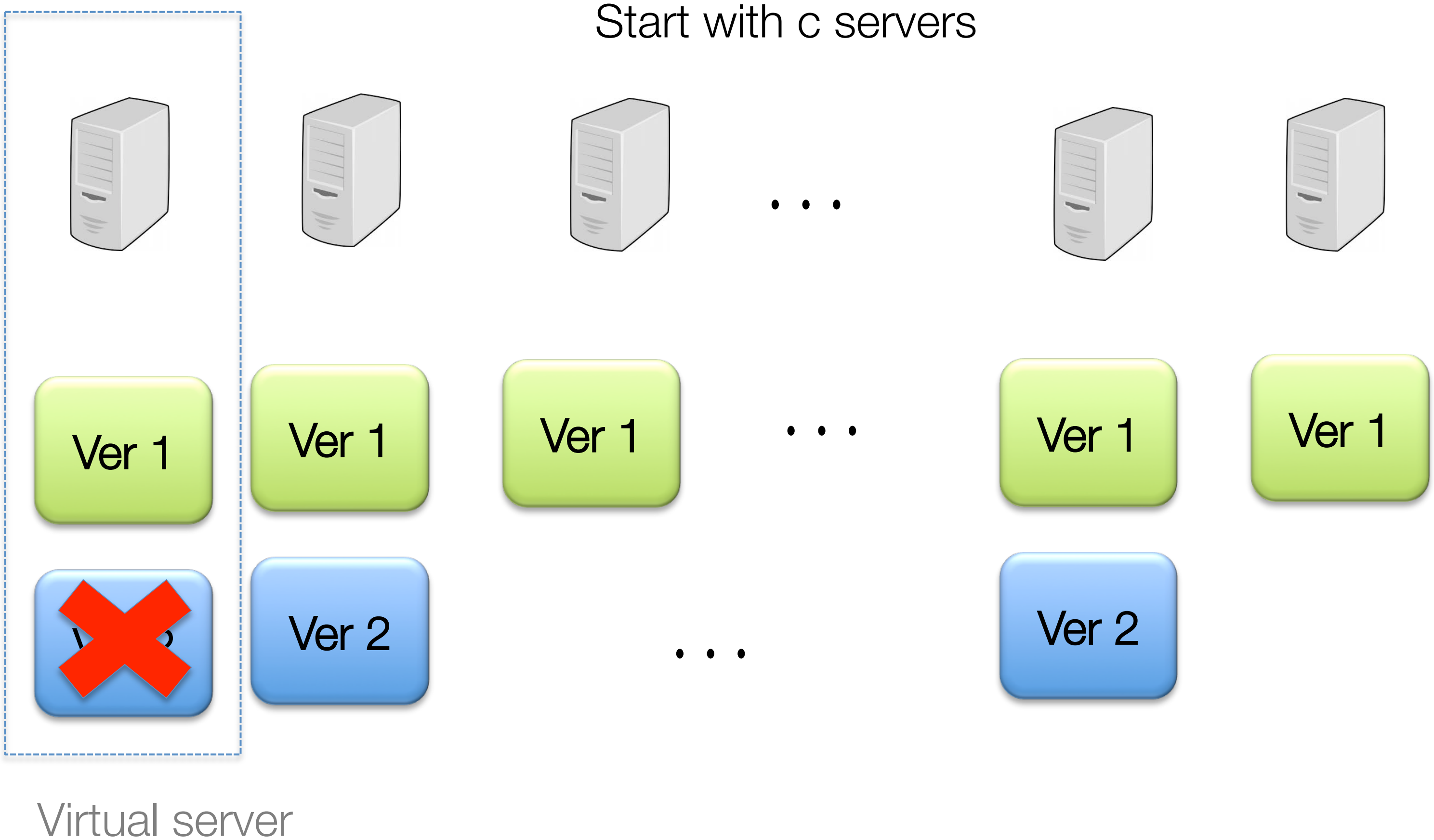
Virtual server

Propagate version 2 to a minimal set of servers such that it is decodable

Versions 1 and 2 decodable from c+1 symbols

Converse: $v = 2$, storage cost $\gtrsim \frac{2}{c+1}$

Start with c servers



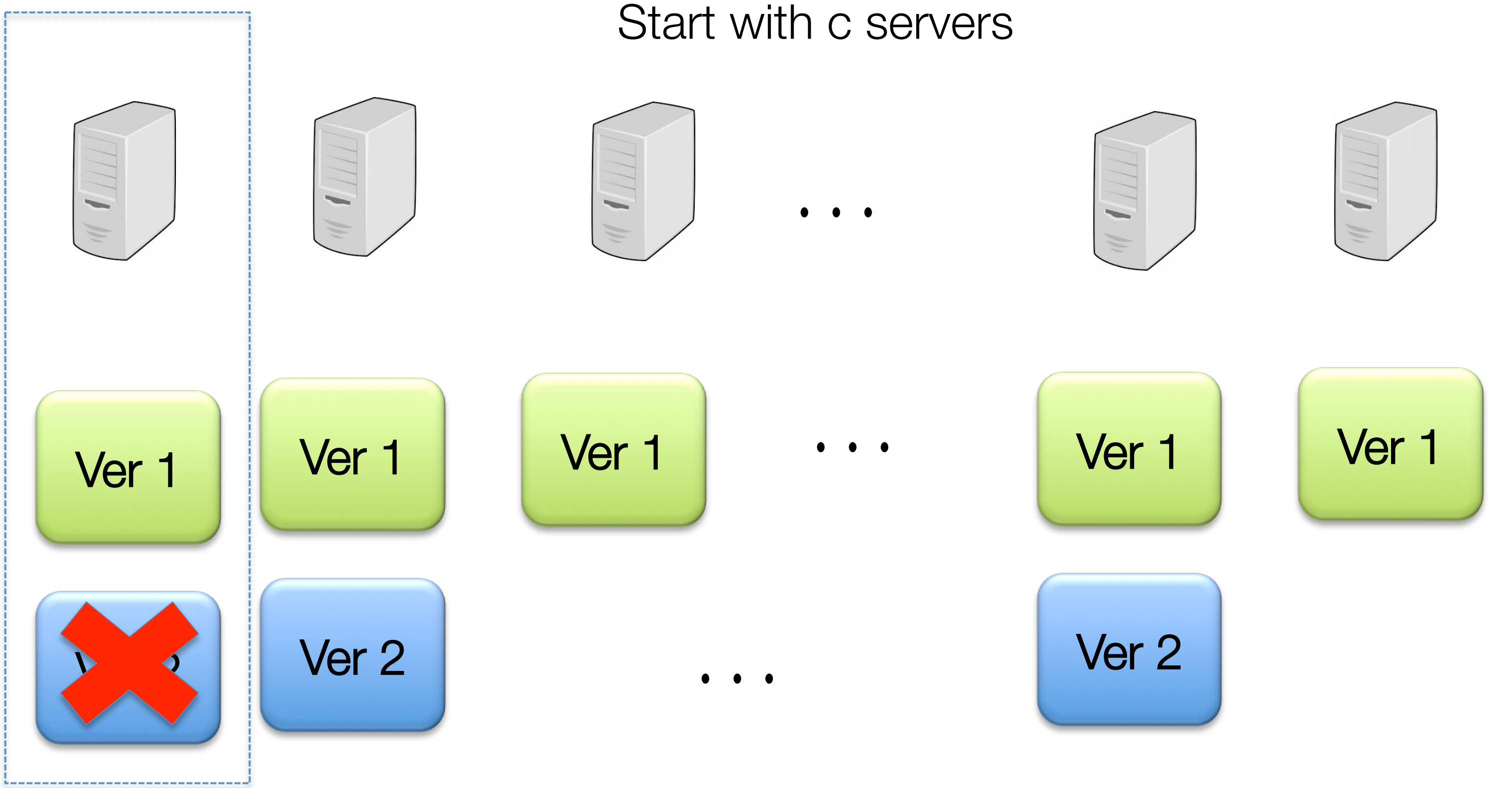
Propagate version 2 to a minimal set of servers such that it is decodable

Versions 1 and 2 decodable from c+1 symbols

$$\implies \text{Storage} \geq \frac{2}{c+1}$$

Converse: $v = 2$, storage cost $\gtrsim \frac{2}{c+1}$

Start with c servers



Virtual server

Propagate version 2 to a minimal set of servers
such that it is decodable

Versions 1 and 2 decodable from c+1 symbols

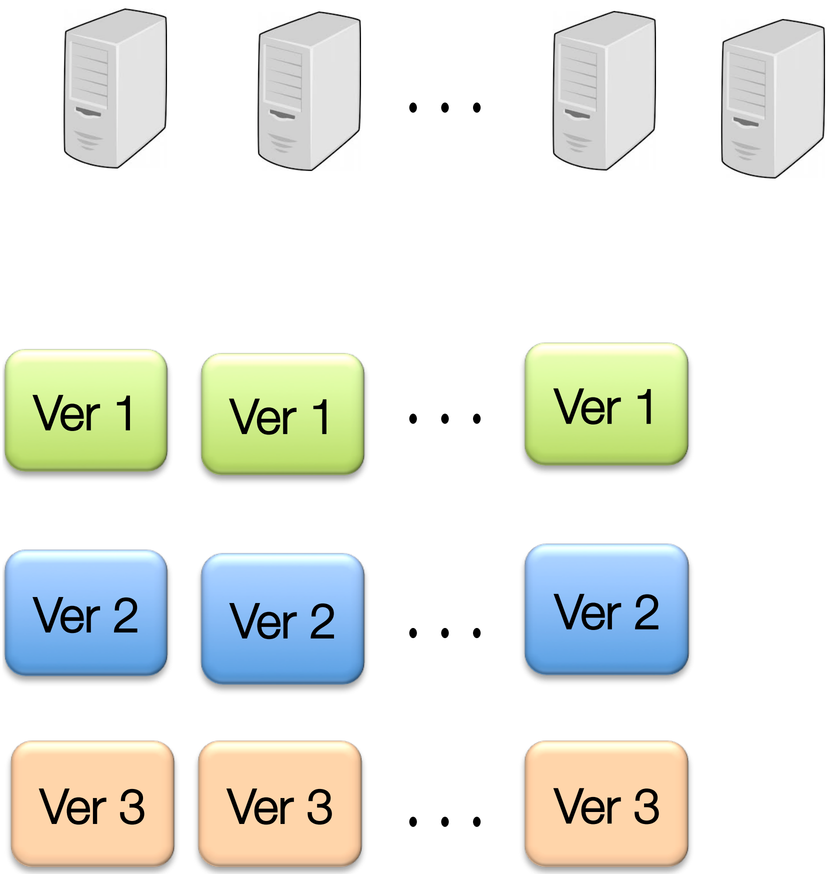
$\Longrightarrow$ Storage $\geq \frac{2}{c+1}$ $-o(\text{size-of-one-version})$

Converse: $v > 2$

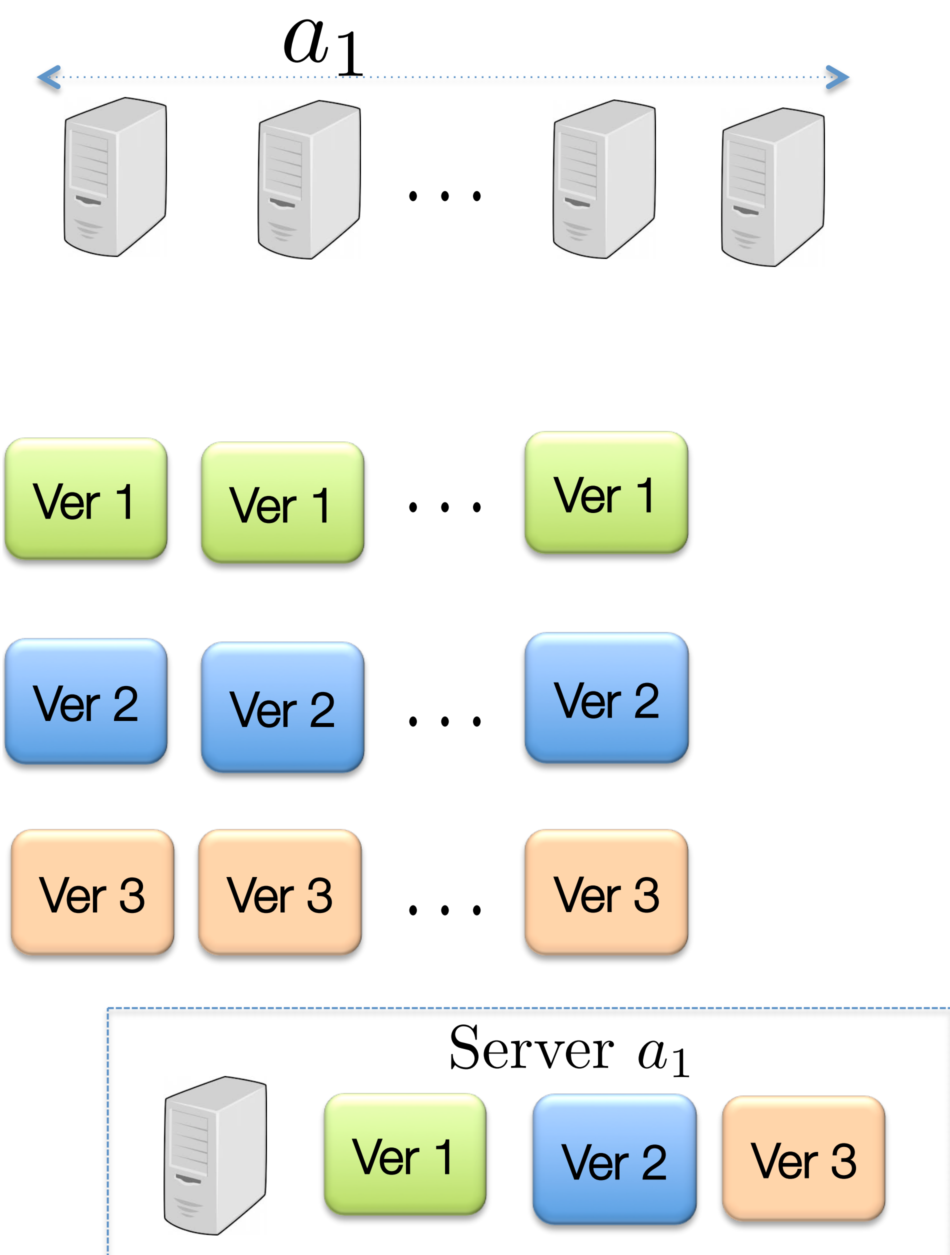
- Intuition: Find c+v-1 virtual servers, where all v versions can be decoded
- A more intricate puzzle as compared to v=2.
- Multi-version coding problem related to index-coding/ multiple-unicast/non-multicast network coding
  - More precisely, it is related to *pliable index coding*

[Brahma-Fragouli 12]

Converse: v=3



Converse: v=3


a_1
. . .
Ver 1
Ver 1
. . .
Ver 1
Ver 2
Ver 2
. . .
Ver 2
Ver 3
Ver 3
. . .
Ver 3
Server a_1
Ver 1
Ver 2
Ver 3

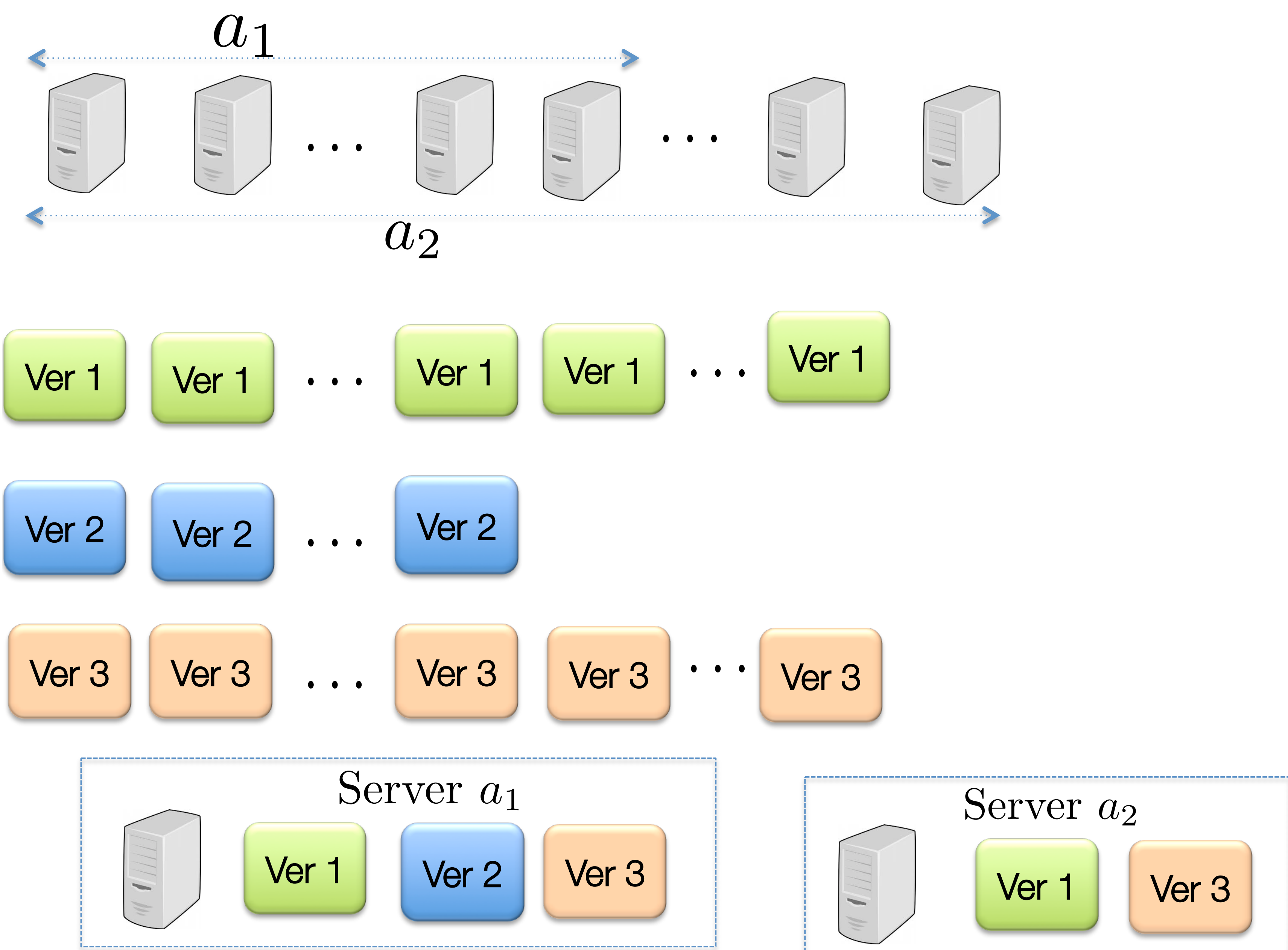
Converse: v=3

a_1 is the smallest number such that, there is a version x, such that

Version x is decodable, given the symbols of the first a_1 servers with all 3 versions

and the messages of versions $\{1, 2, 3\} - \{x\}$

Converse: v=3



a_1
a_2
Ver 1
Ver 1
Ver 1
Ver 1
Ver 1
Ver 2
Ver 2
Ver 2
Ver 3
Ver 3
Ver 3
Ver 3
Ver 3
Server a_1
Ver 1
Ver 2
Ver 3
Server a_2
Ver 1
Ver 3

Converse: v=3

a_1 is the smallest number such that, there is a version x, such that

Version x is decodable, given the symbols of the first a_1 servers with all 3 versions

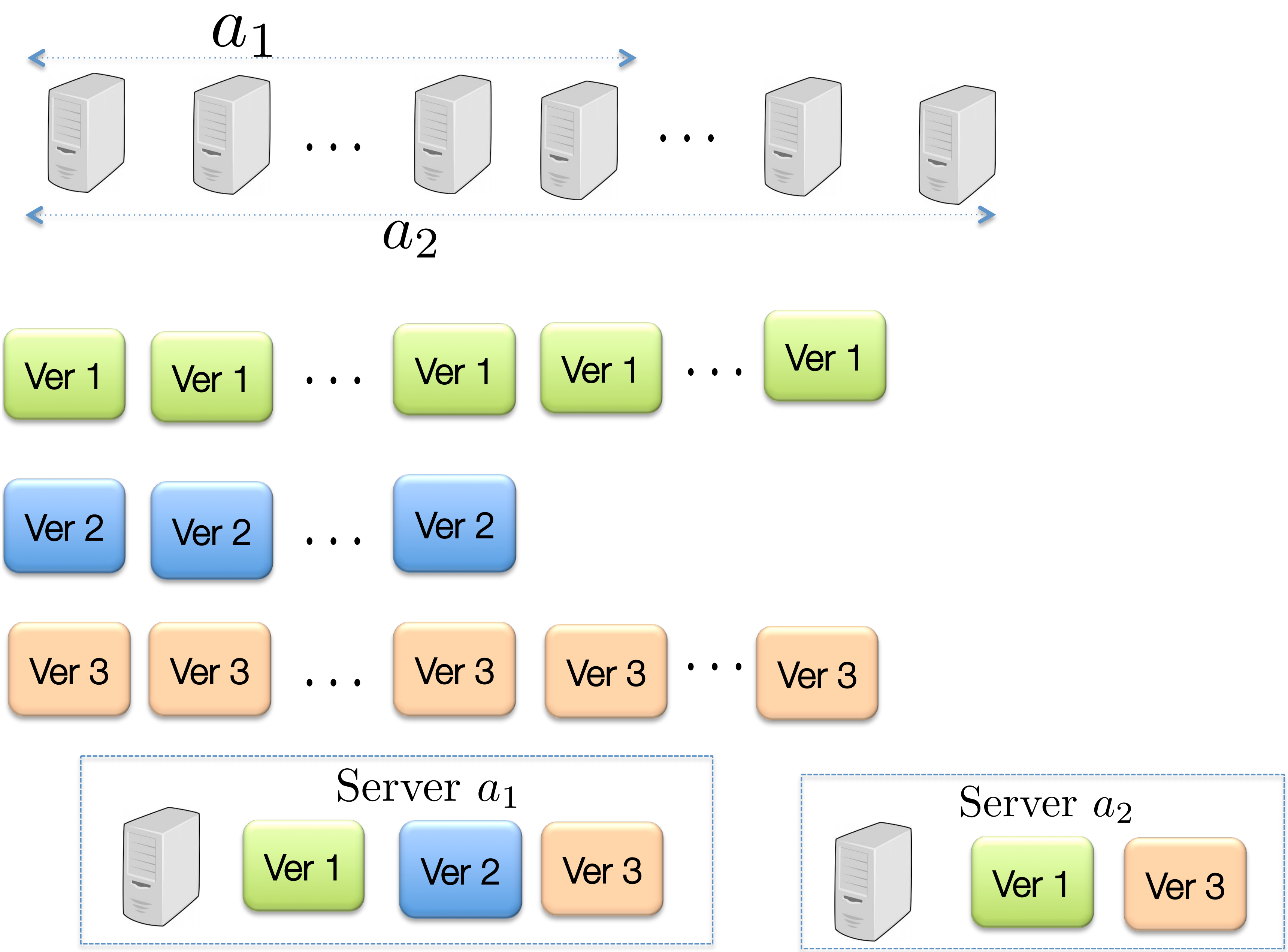
and the messages of versions $\{1, 2, 3\} - \{x\}$

a_2 is the smallest number such that, there is a version $y \in \{1, 2, 3\} - \{x\}$, such that

Version y is decodable, given the symbols of the first $a_1 - 1$ servers with all 3 versions and the remaining $a_2 - (a_1 - 1)$ servers with versions $\{1, 2, 3\} - \{x\}$

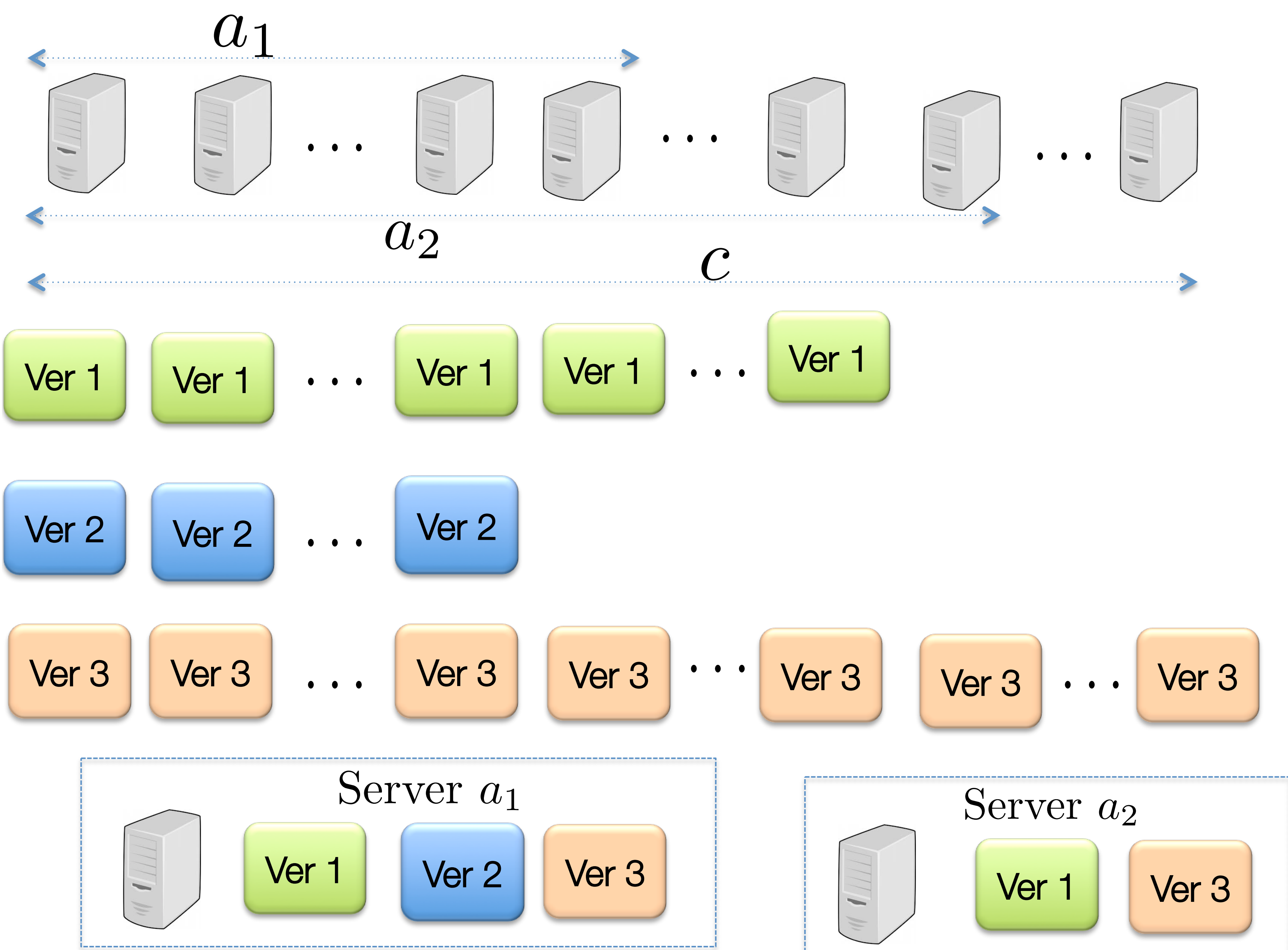
and the message of version $\{1, 2, 3\} - \{x, y\}$

Converse: v=3



a_1
a_2
Ver 1
Ver 1
Ver 1
Ver 1
Ver 1
Ver 2
Ver 2
Ver 2
Ver 3
Ver 3
Ver 3
Ver 3
Ver 3
Server a_1
Ver 1
Ver 2
Ver 3
Server a_2
Ver 1
Ver 3

Converse: v=3



a_1
a_2
c
Ver 1
Ver 1
Ver 1
Ver 1
Ver 1
Ver 2
Ver 2
Ver 2
Ver 3
Ver 3
Ver 3
Ver 3
Ver 3
Ver 3
Ver 3
Server a_1
Ver 1
Ver 2
Ver 3
Server a_2
Ver 1
Ver 3

Summary

	Storage Cost Normalized by size-of-one-version
Replication	1
Naïve MDS codes	v/c
Constructions	$\frac{1}{\lceil c/v \rceil}$ *
Lower bound	$\frac{v}{c+v-1}$ * $-o(\text{size-of-one-version})$

$$v = \text{Number of Versions}$$

$$c = \text{Connectivity}$$

*These bounds can be improved.

See "Multi-version Coding – An Information Theoretic Perspective of Distributed Storage ", Wang-Cadambe, arxiv, 2015

Multi-version codes – Main Insights

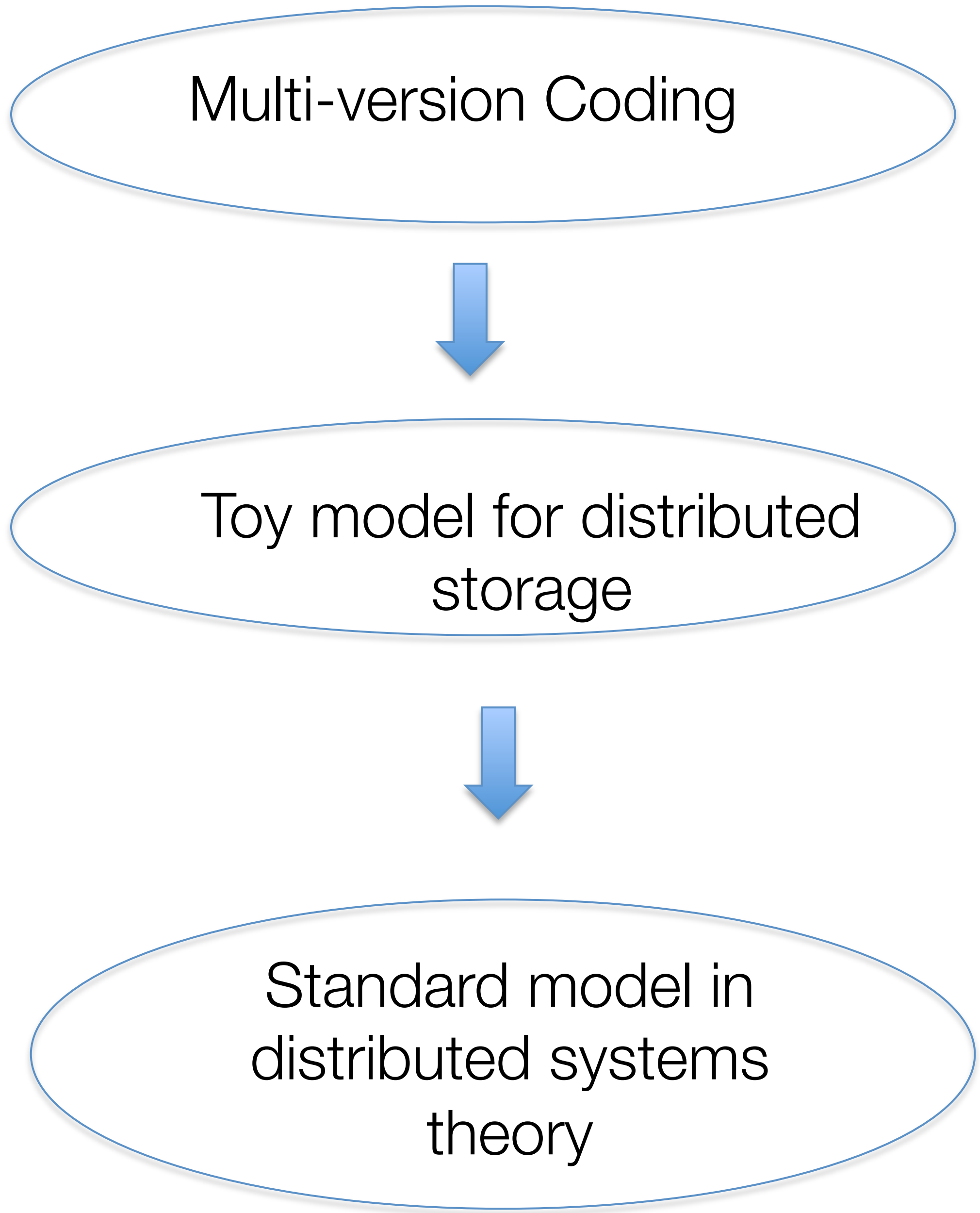
- Redundancy required to ensure consistency in an asynchronous environment
  - Redundancy increases with the number of parallel versions in the system

See "Multi-version Coding – An Information Theoretic Perspective of Distributed Storage ", Wang-Cadambe, arxiv, 2015

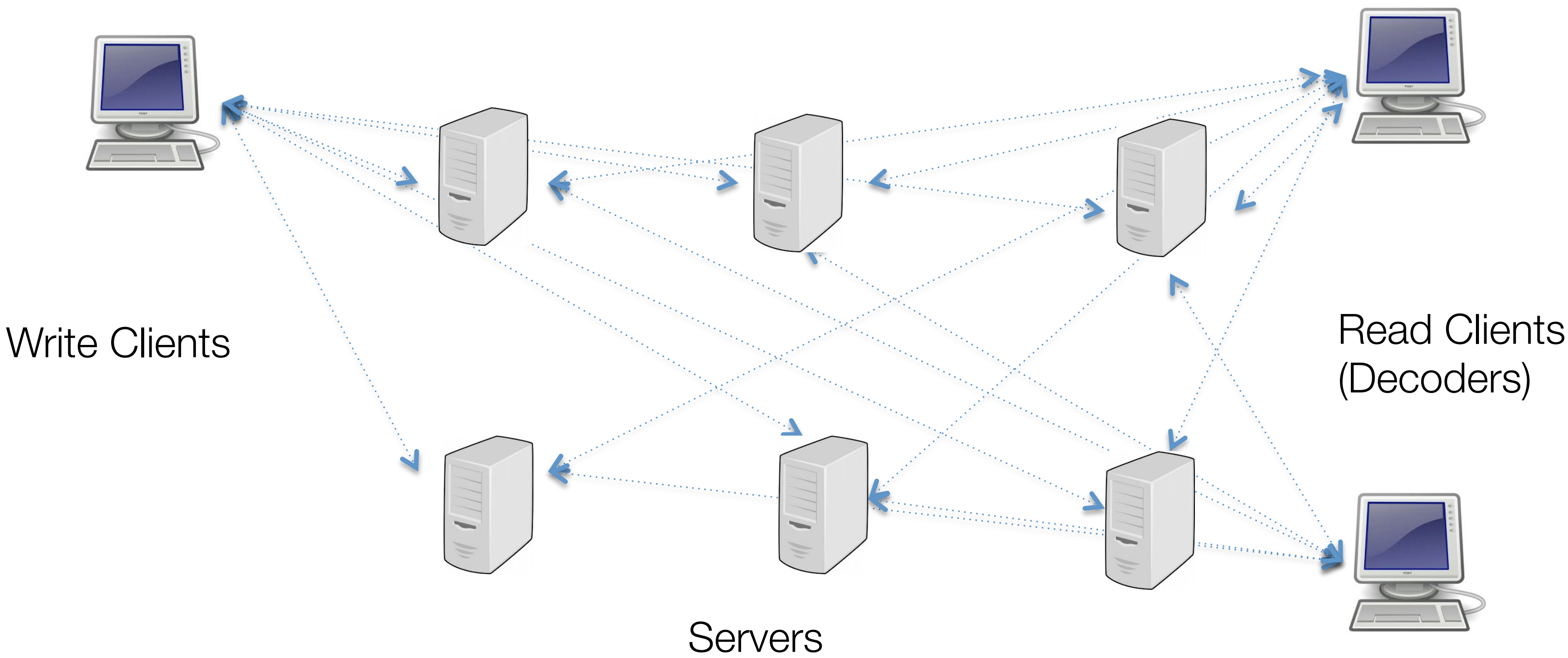
Multi-version codes – Main Insights

- Redundancy required to ensure consistency in an asynchronous environment
  - Redundancy increases with the number of parallel versions in the system
- Simple codes are (approximately) optimal
  - Separate coding across versions
  - Random linear codes within versions

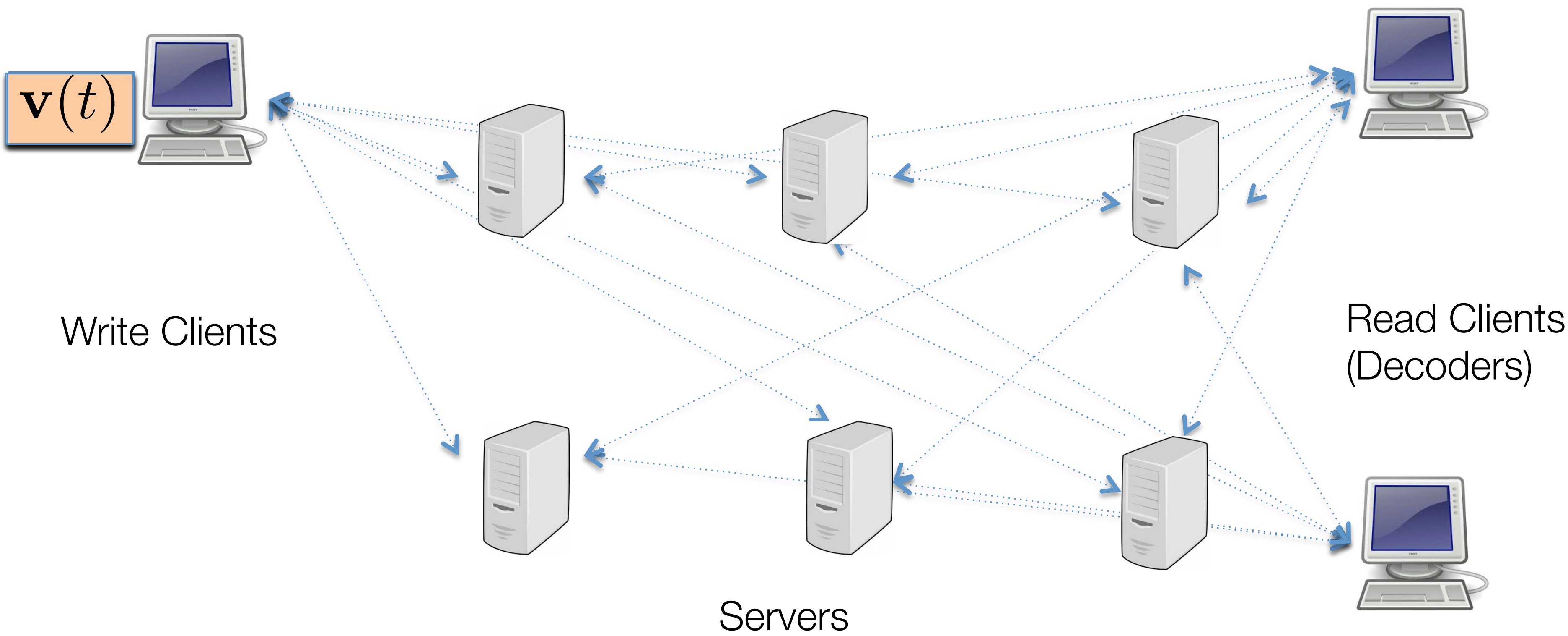
See "Multi-version Coding – An Information Theoretic Perspective of Distributed Storage ", Wang-Cadambe, arxiv, 2015


Multi-version Coding
Toy model for distributed storage
Standard model in distributed systems theory

Toy Model for packet arrivals, links

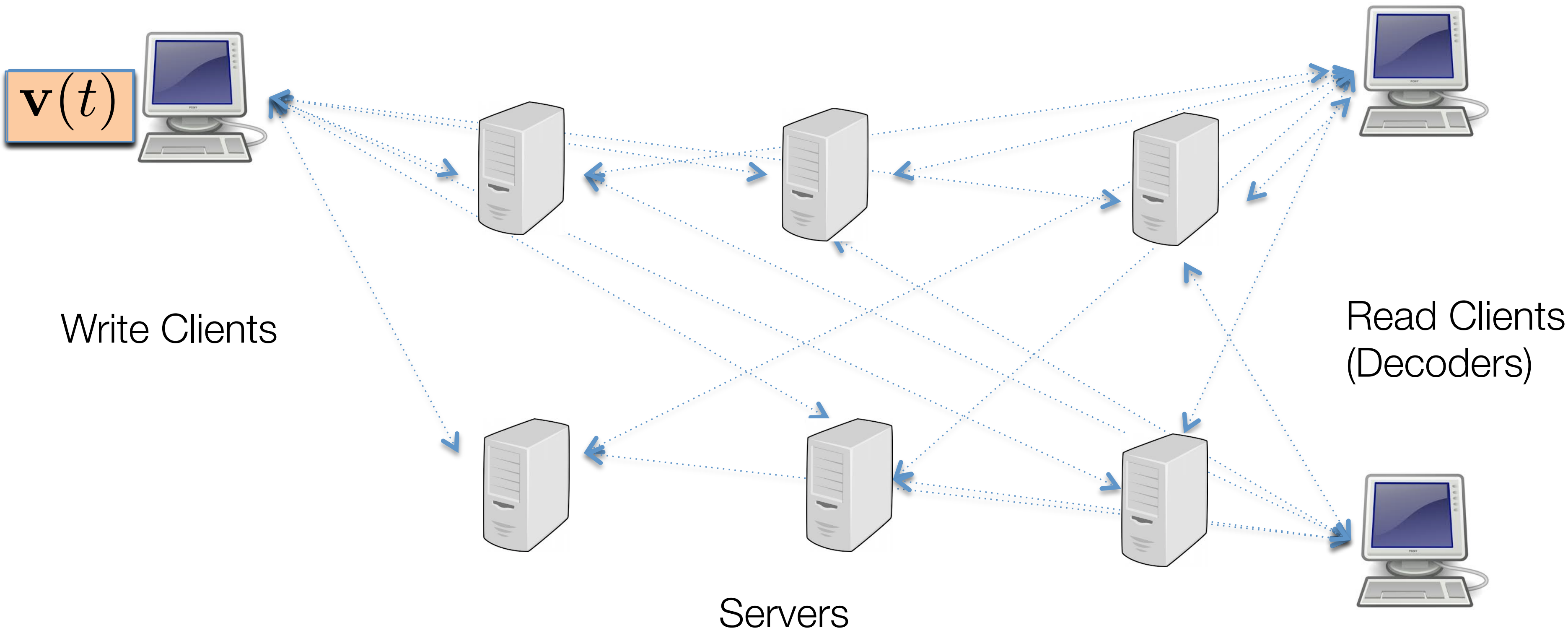


Toy Model for packet arrivals, links



- Arrival at client: One packet in every time slot. Sent immediately to the servers.
- Channel from the write client to the server: Delay is an integer in [0,T-1].

Toy Model for packet arrivals, links



- Arrival at client: One packet in every time slot. Sent immediately to the servers.
- Channel from the write client to the server: Delay is an integer in [0,T-1].
- Channel from server to read client: instantaneous (no delay).
- Goal: decoder invoked at time t, gets the latest common version among c servers

Insights from multi-version codes over toy model

Achievability "Theorem":

There exists an achievable storage strategy that achieves a storage cost of

$$\frac{1}{\lceil \frac{T}{c} \rceil} \times \text{size-of-one-version}$$

Converse "Theorem":

There exists no achievable storage strategy that achieves a storage cost smaller than

$$\frac{T}{T+c-1} \times \text{size-of-one-version} - o(\text{size-of-one-version})$$

Insights from multi-version codes over toy model

<u>Achievability "Theorem":</u>

There exists an achievable storage strategy that achieves a storage cost of

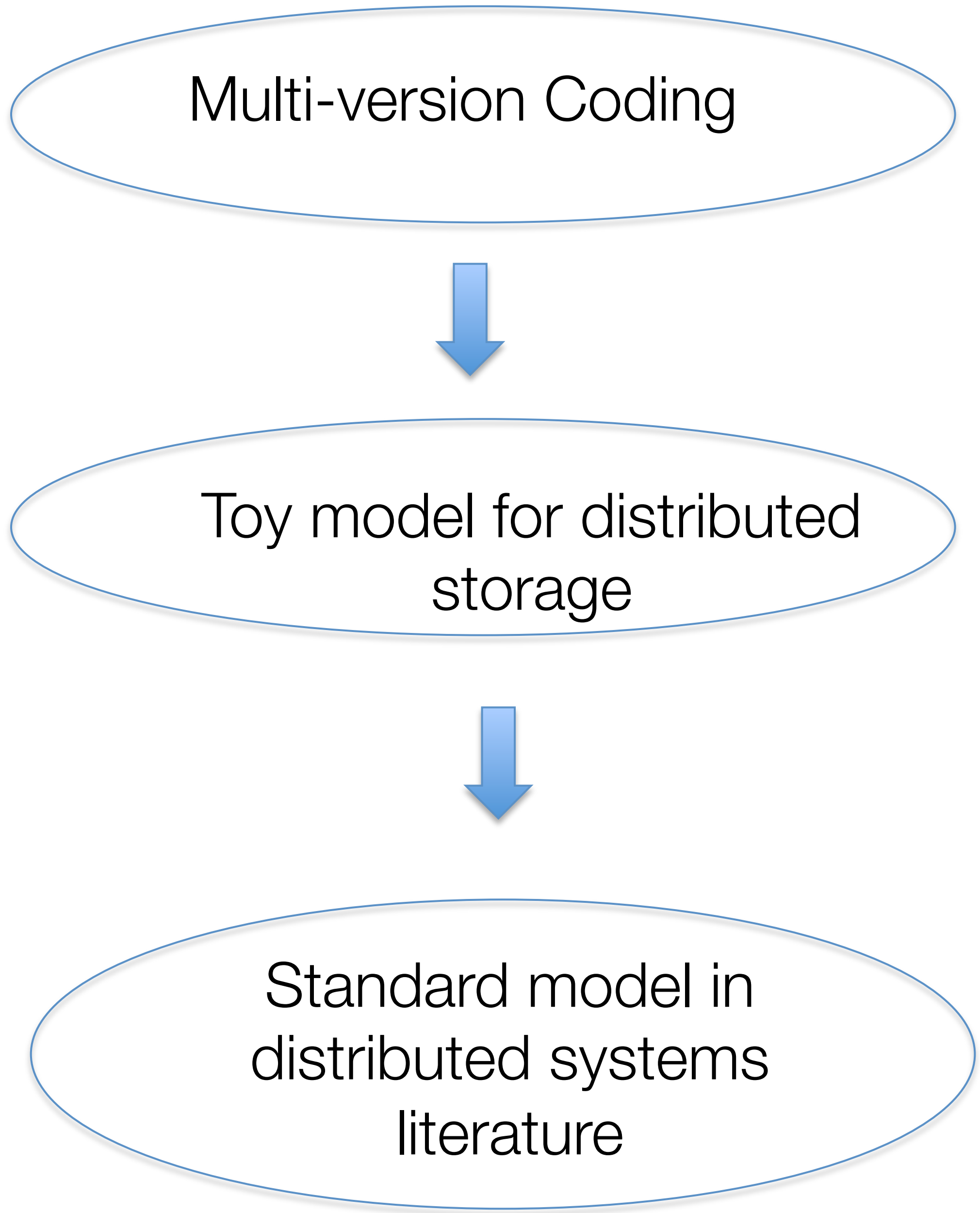
$$\frac{1}{\lceil \frac{T}{c} \rceil} \times \text{size-of-one-version}$$

<u>Converse "Theorem":</u>

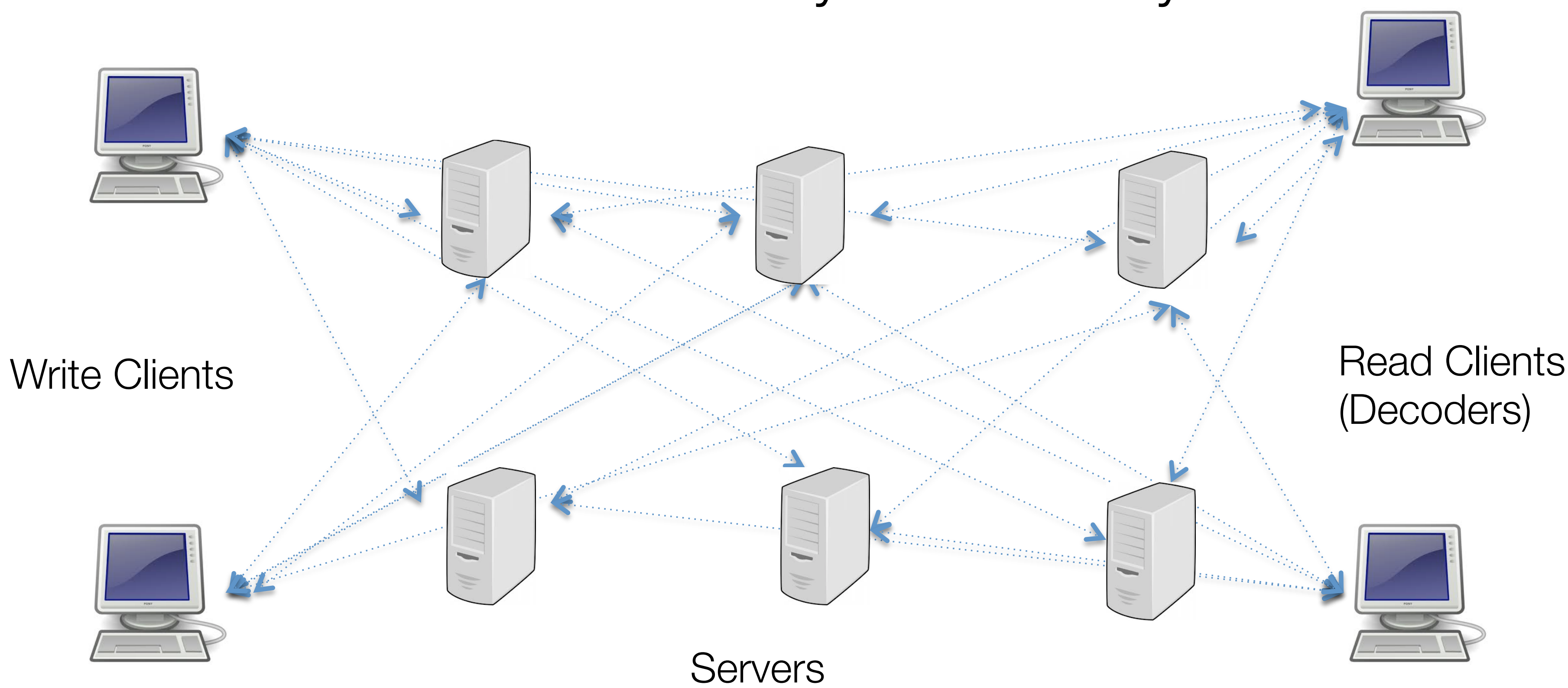
There exists no achievable storage strategy that achieves a storage cost smaller than

$$\frac{T}{T + c - 1} \times \text{size-of-one-version} - o(\text{size-of-one-version})$$

Number of versions ν, depends on degree of asynchrony T

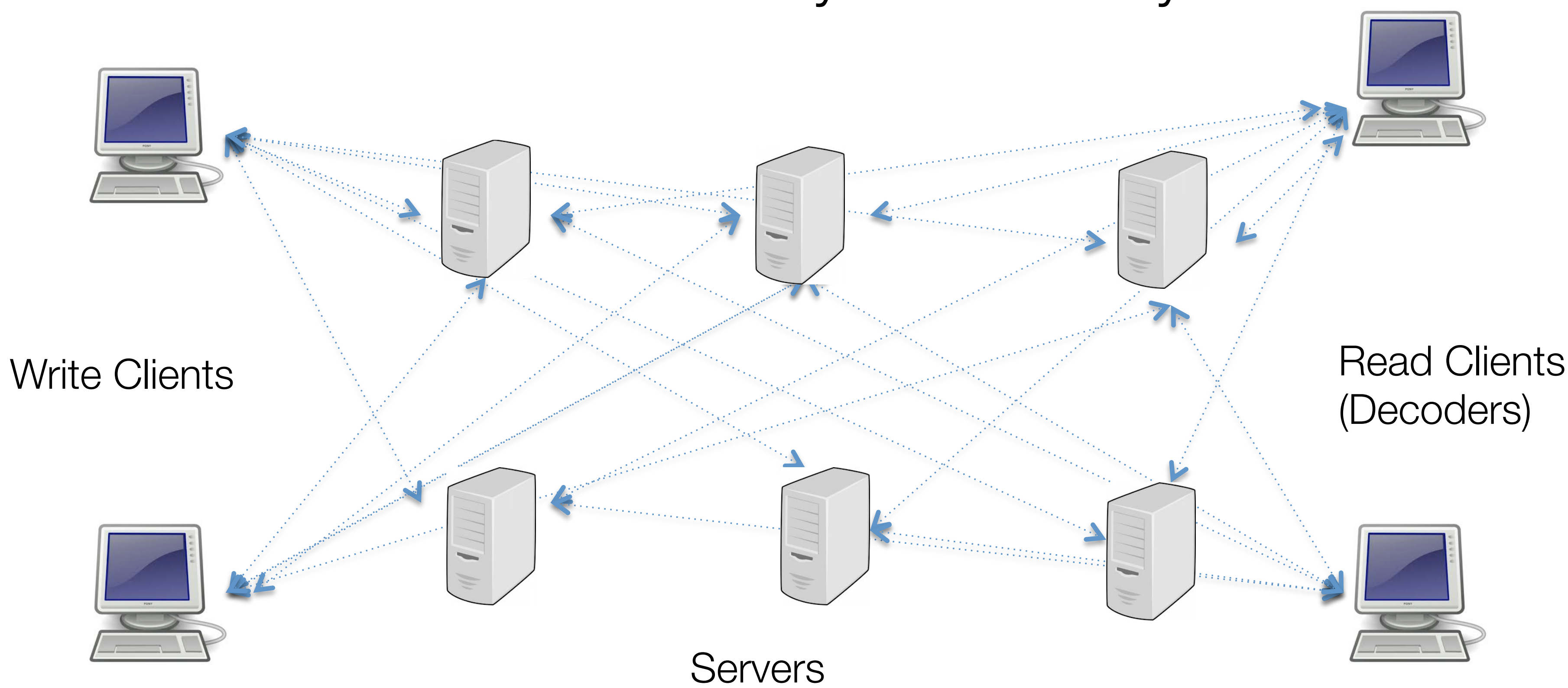

Multi-version Coding
Toy model for distributed storage
Standard model in distributed systems literature

Model studied in distributed systems – Key features



- Arrival at clients: arbitrary
- Channel from clients to servers: arbitrary delay, reliable
- Clients and servers are modeled as I/O automata, so their protocols can be designed.

Model studied in distributed systems – Key features



- Arrival at clients: arbitrary
- Channel from clients to servers: arbitrary delay, reliable
- Clients and servers are modeled as I/O automata, so their protocols can be designed.

Multi-version coding converse for v=2 can be lifted to this setting.

Future Work – Many open questions

- Less conservative modeling assumptions,
  - Exploiting correlation between versions
  - Allow for a “small” number of erroneous states
  - Less distributed, knowledge of the state of other nodes.

Future Work – Many open questions

- Less conservative modeling assumptions,
  - Exploiting correlation between versions
  - Allow for a “small” number of erroneous states
  - Less distributed, knowledge of the state of other nodes.

- Finer network and node models (beyond toy models).
  - Can lead to finer insights in to communication and storage costs
  - Allow for the design of protocols, for say, the read client (or the write client)

Future Work – Many open questions

- Less conservative modeling assumptions,
  - Exploiting correlation between versions
  - Allow for a “small” number of erroneous states
  - Less distributed, knowledge of the state of other nodes.

- Finer network and node models (beyond toy models).
  - Can lead to finer insights in to communication and storage costs
  - Allow for the design of protocols, for say, the read client (or the write client)

- Study of errors/Byzantine adversaries instead of erasures - useful assumption for ensuring security.

Thanks