Distributed Systems

Spring Semester 2020

Lecture 23: Distributed Training (Parameter Server)

John Liagouris
liagos@bu.edu

Why this paper?

- Very influential architecture for distributed training
- Flexible consistency model — users can choose
- Puts together common techniques in distributed systems:
  - Chain replication — for fault-tolerance
  - Consistent hashing — for fast recovery & load balancing
  - Vector clocks — also in TreadMarks, Bayou, and Dynamo
- System scales to 100K cores — billions of parameters

ML Basics

- Model Training
  - Build the model using training data and a loss function
  - The goal is to "learn" an unknown function, e.g. f(user profile) → likelihood of ad click
    - An iterative process — typically offline
- Model Serving
  - Make predictions using the trained model (inference), e.g. Q:"What's in this image?" A:"A cat"
  - Typically online

ML Basics

- Model Training

  Parameter Server

  - Build the model using training data and a loss function
  - The goal is to "learn" an unknown function, e.g. f(user profile) → likelihood of ad click
  - An iterative process — typically offline

- Model Serving
  - Make predictions using the trained model (inference), e.g. Q: "What's in this image?" A: "A cat"
  - Typically online

Model Training

- Supervised
  - There is some "ground truth" (e.g. images labeled by humans) — check model against the truth to verify accuracy
- Unsupervised
  - The is no ground truth, e.g. topic modeling: given a collection of documents, infer the topics contained in the documents
- Semi-supervised
  - Small amount of labeled data

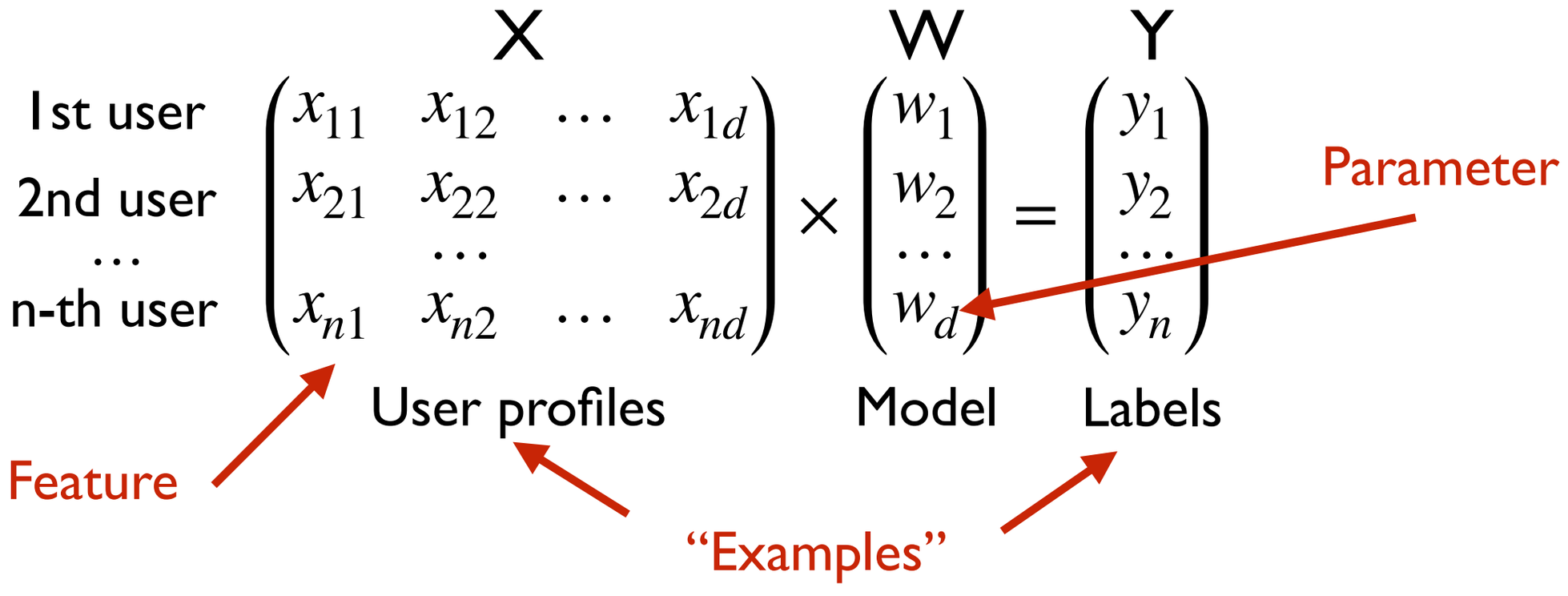
Model Training

Parameter Server

- Supervised
  - There is some "ground truth" (e.g. images labeled by humans) — check model against the truth to verify accuracy
- Unsupervised
  - The is no ground truth, e.g. topic modeling: given a collection of documents, infer the topics contained in the documents
- Semi-supervised
  - Small amount of labeled data

Example

- Supervised learning
  - Training data: Labeled user profiles (age, sex, zip, job, …)
  - Labels: "clicked on Ad" (1) or "not clicked on Ad" (-1)
  - Goal: Find a model W that predicts "clickiness" of each profile attribute ("feature")



Example

- How to find the model W?
  - Use a loss function F(X, W, Y) — represents the prediction error
  - Regularizer Ω(W) — penalizes the model complexity
- Iterative process
  - At each step, compute the gradient of F — ∇F
  - Update model based on the gradient and the regularizer
  - Repeat until the "prediction error" is acceptable

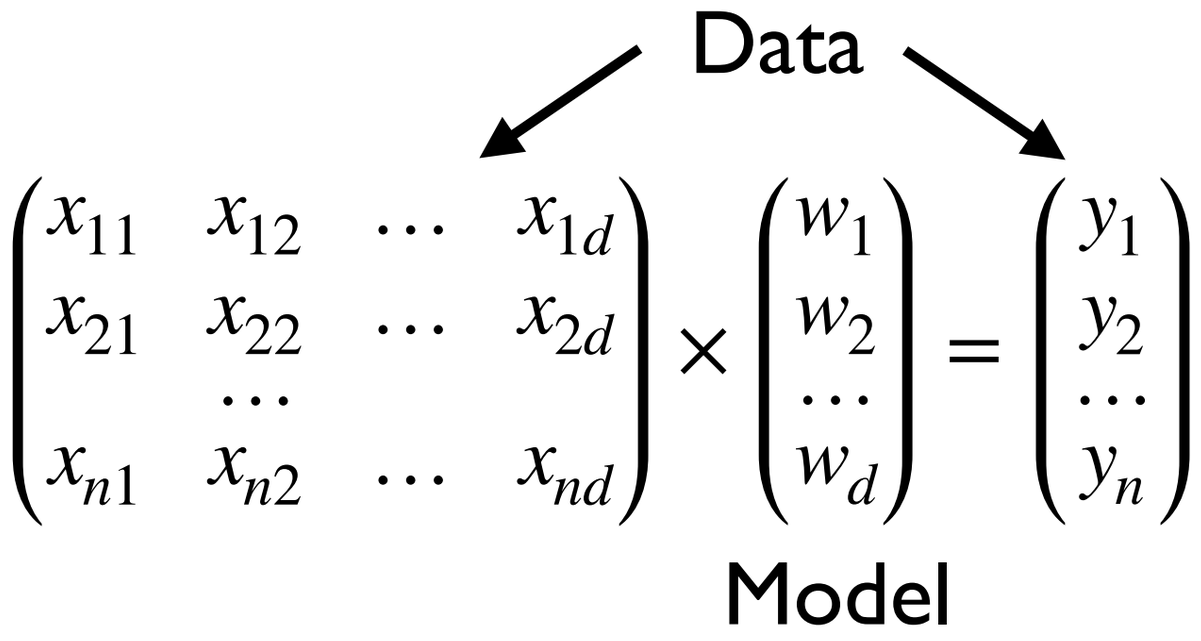
Training Challenges

- More data result in better models (up to a point)
  - Training data in the order of 100s TB - few PB
  - Model parameters in the order of billions
- BUT training process cannot be distributed easily
  - Workers must “share” state
  - Exchange updates with all other workers
  - Coordination is required

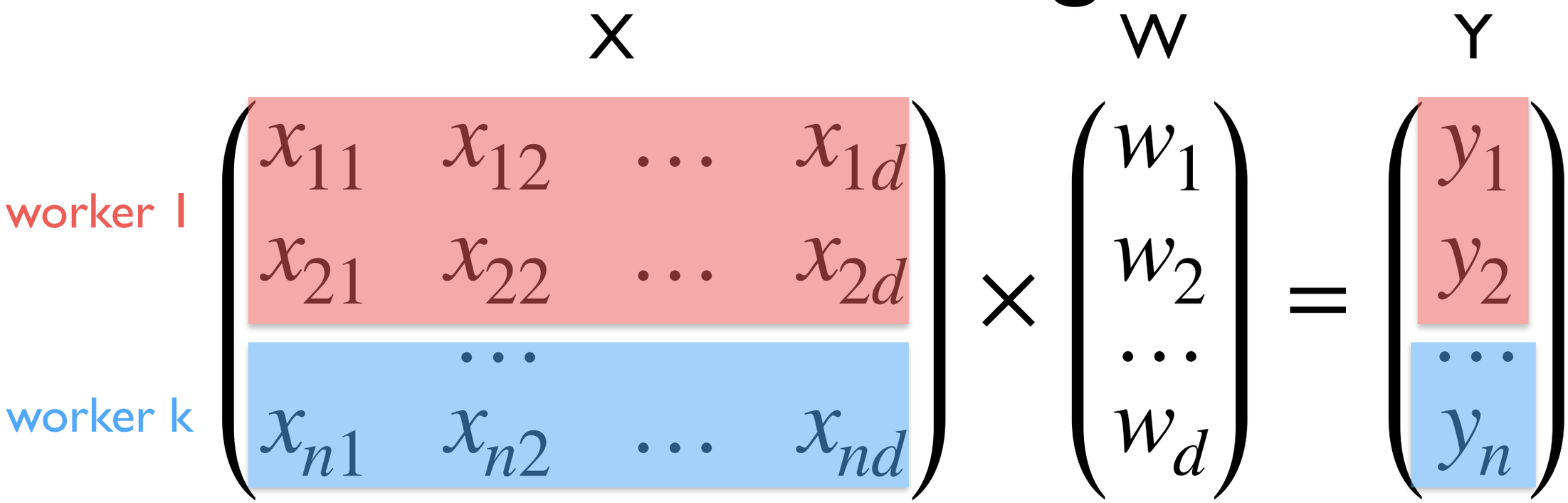
Two approaches

This paper

- Data Parallelism
  - Partition data - share model
- Model parallelism
  - Partition model - share data

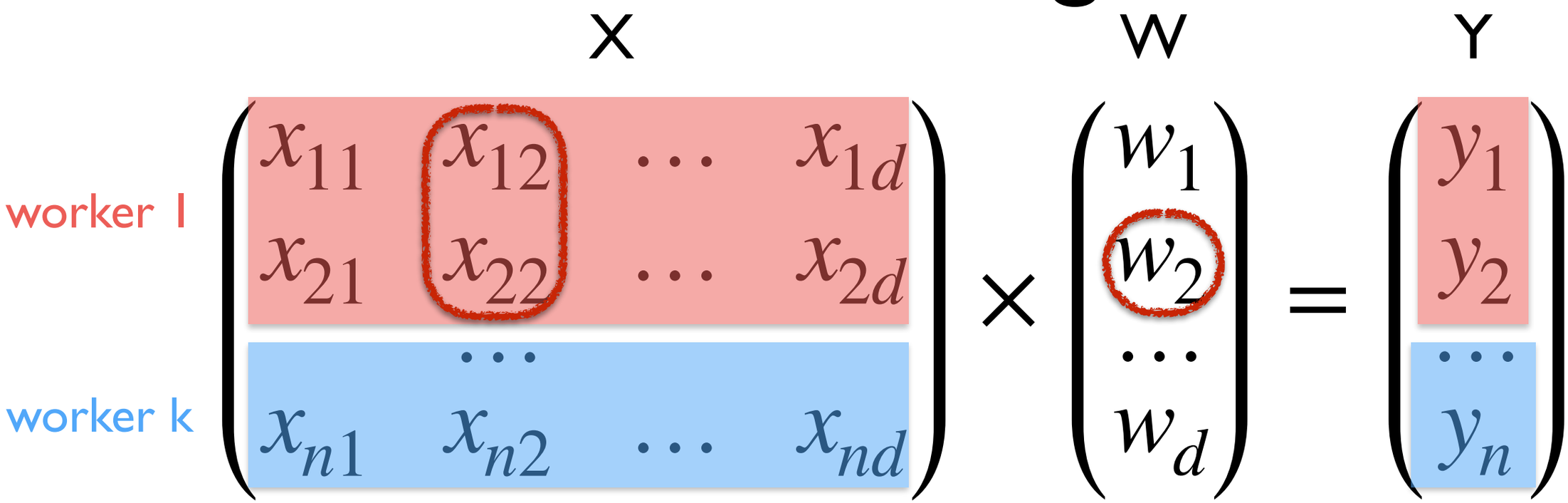


Data Partitioning in PS



- Model is "shared", i.e., replicated to all workers
- Data are often sparse — many features are zero or "null"
- In this case, each worker will only store the model parameters relevant to its partition - the "working set" of W

Data Partitioning in PS



- Model is "shared", i.e., replicated to all workers
- Data are often sparse — many features are zero or "null"
- In this case, each worker will only store the model parameters relevant to its partition - the "working set" of W

Strawman #1

- Each worker broadcasts updated parameters after each iteration and waits for updates from other workers
- Possible problems?
  - Ridiculous network traffic
  - Idle time — workers wait for all others (sensitive to stragglers)

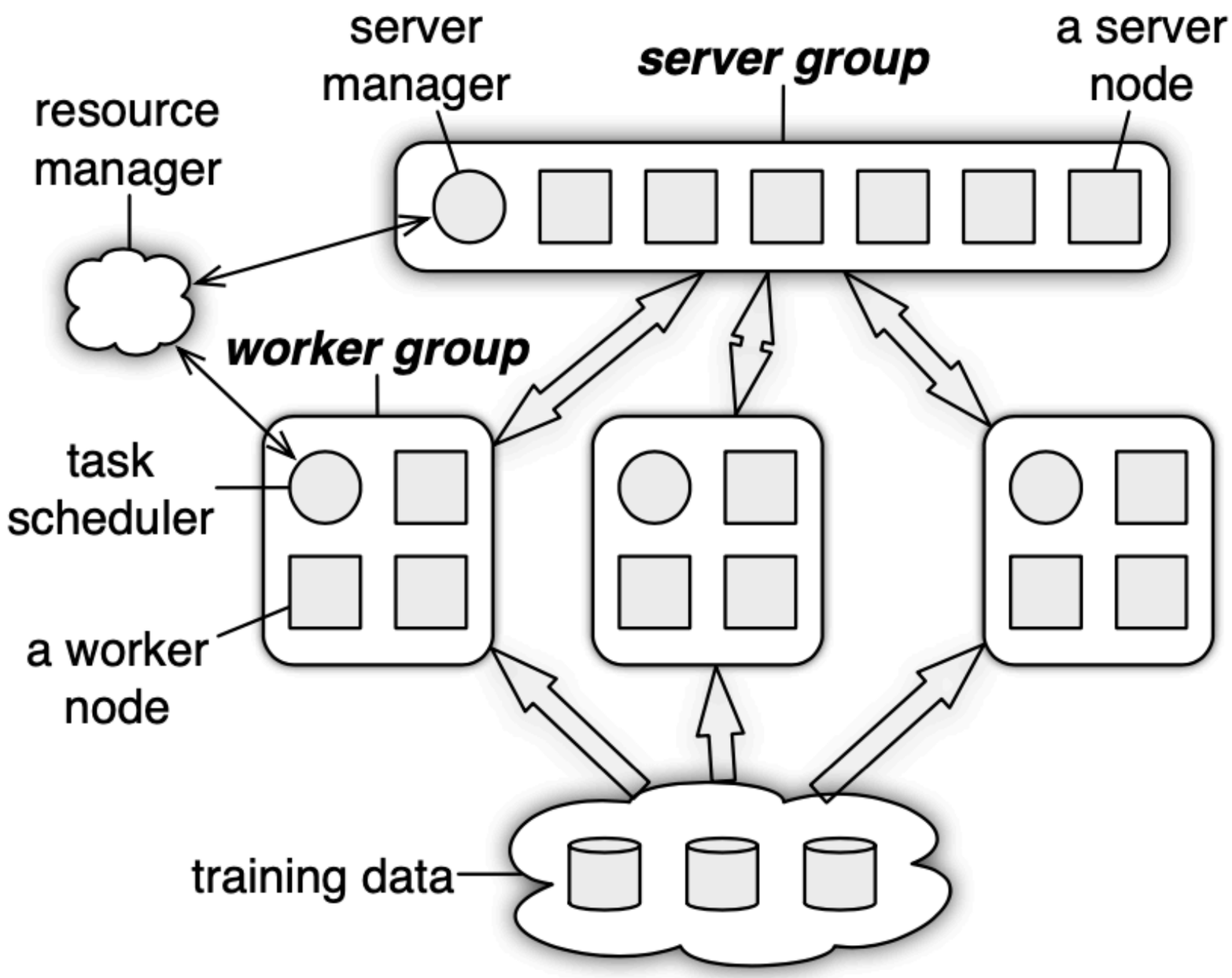
Strawman #2

- Use a single coordinator (master) to collect updates, aggregate gradients, and broadcast updates to all workers
- Possible problems?
  - Coordinator becomes bottleneck
  - Coordinator is a single point of failure

Strawman #3

- Use a big data system like Spark
  - Partition data to workers
  - Each worker computes a gradient in the form of an RDD
  - RDDs are shuffled to workers before the next iteration
- Possible problems?
  - Shuffling introduces high network traffic and is sensitive to stragglers (slow workers)
  - Cannot leverage narrow RDD dependencies for fast recovery

Parameter Server

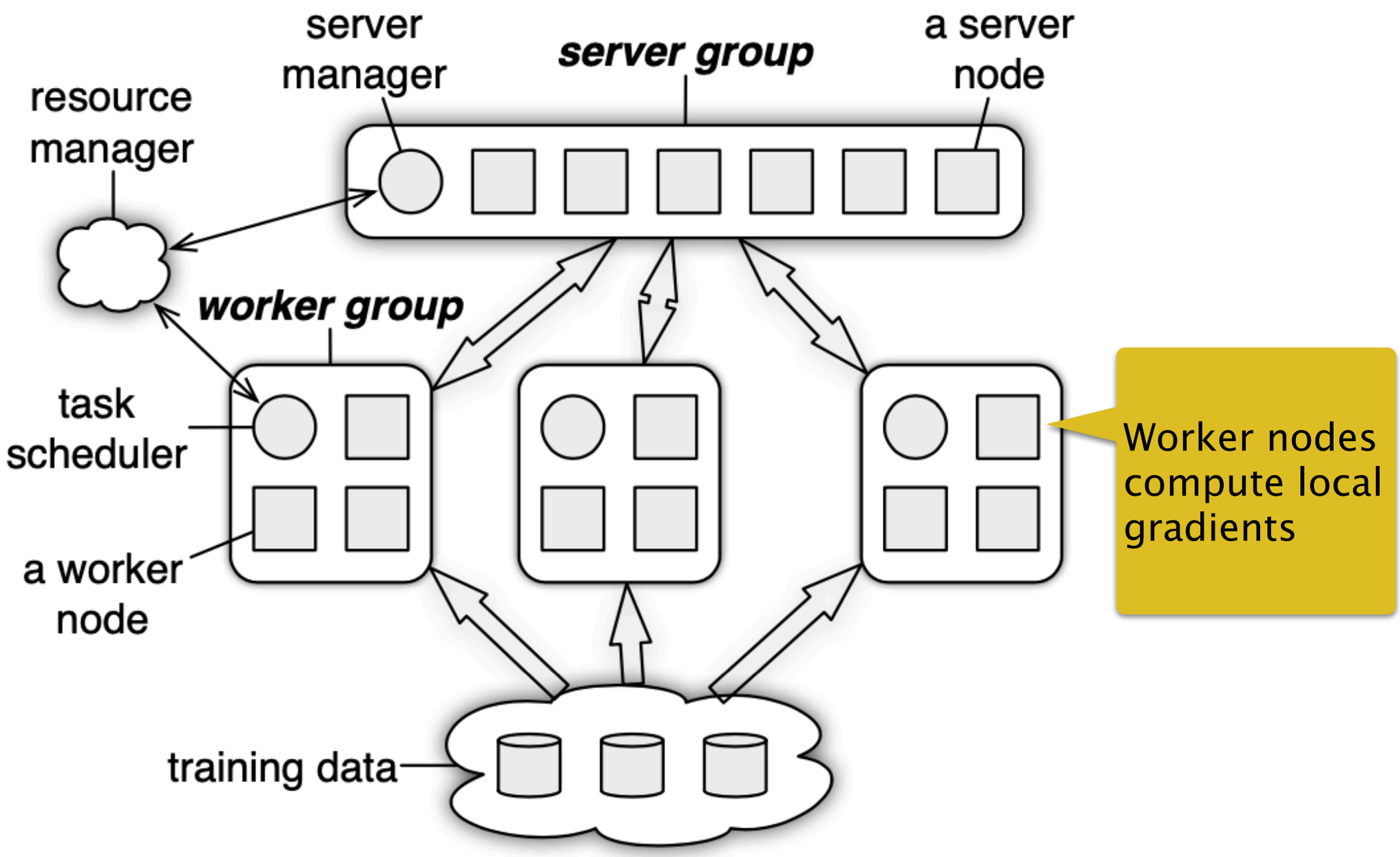


Training Data

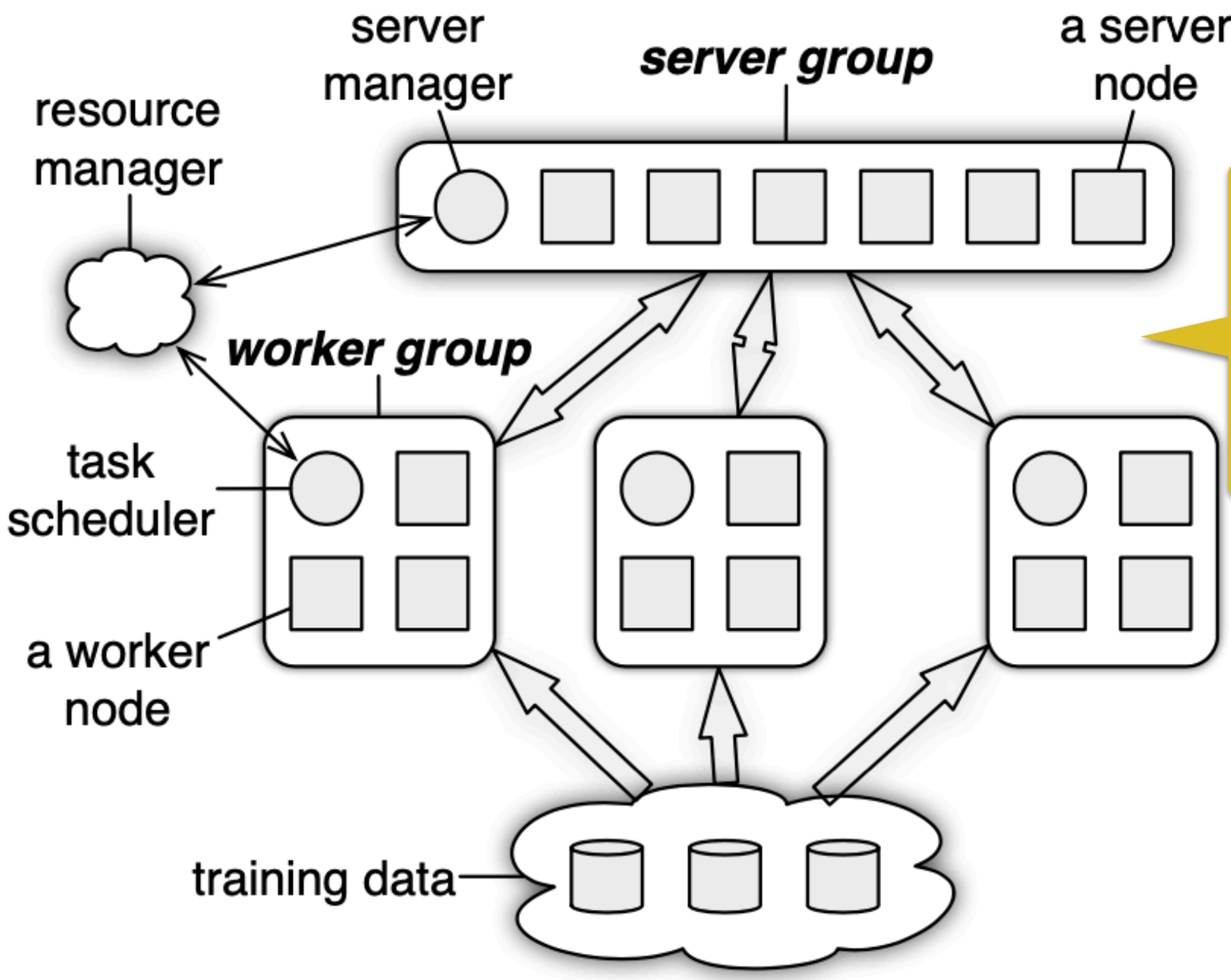
$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ & \cdots & & \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix} \times \begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_d \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix}$$

- Data and model parameters are represented as key-value pairs
  - $x_{11} \rightarrow (key : 11, value : x_{11})$
  - $w_2 \rightarrow (key : 2, value : w_2)$
  - Good for sparse data — no need to reserve space for non-existing features or parameters

Parameter Server

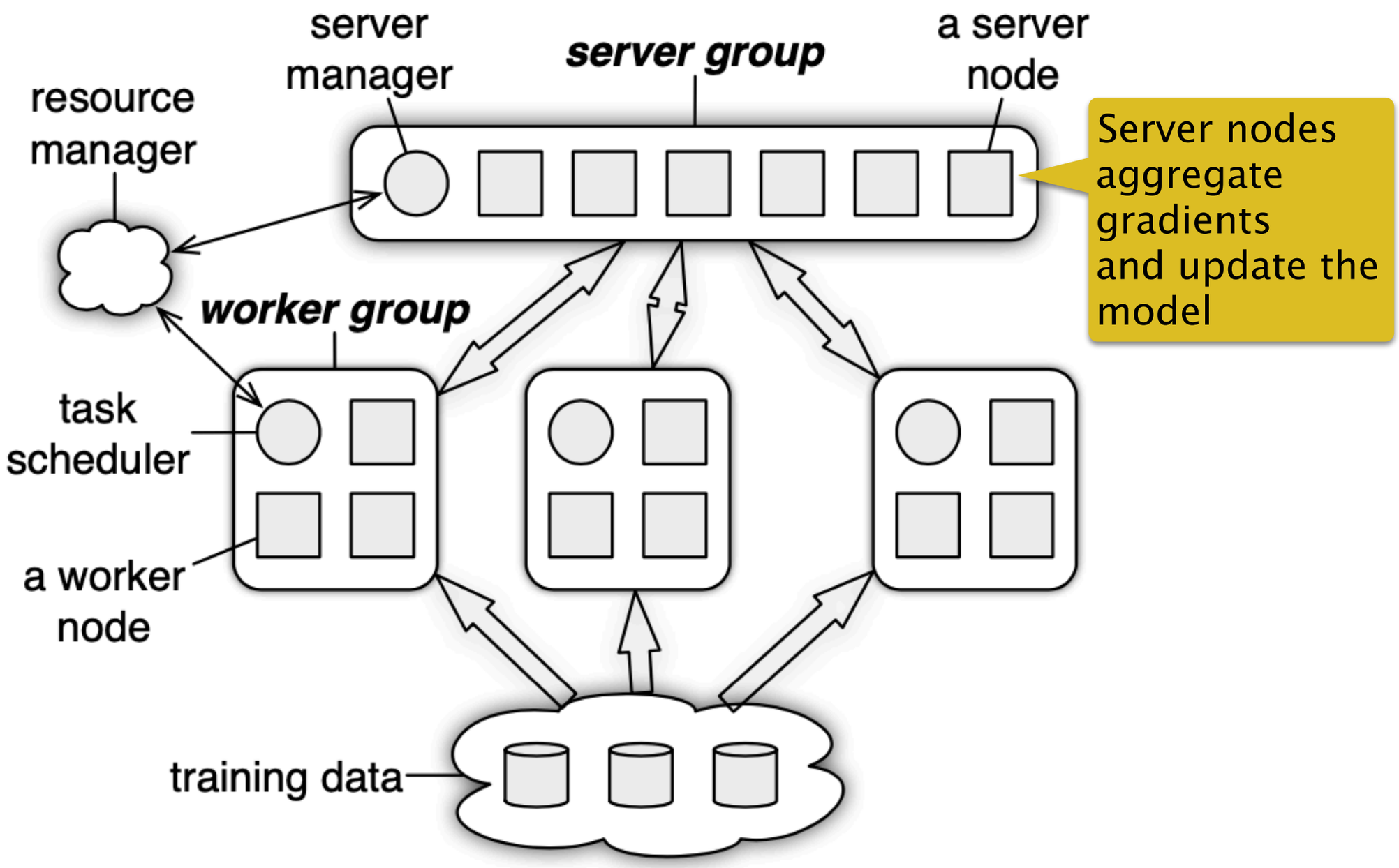


Parameter Server

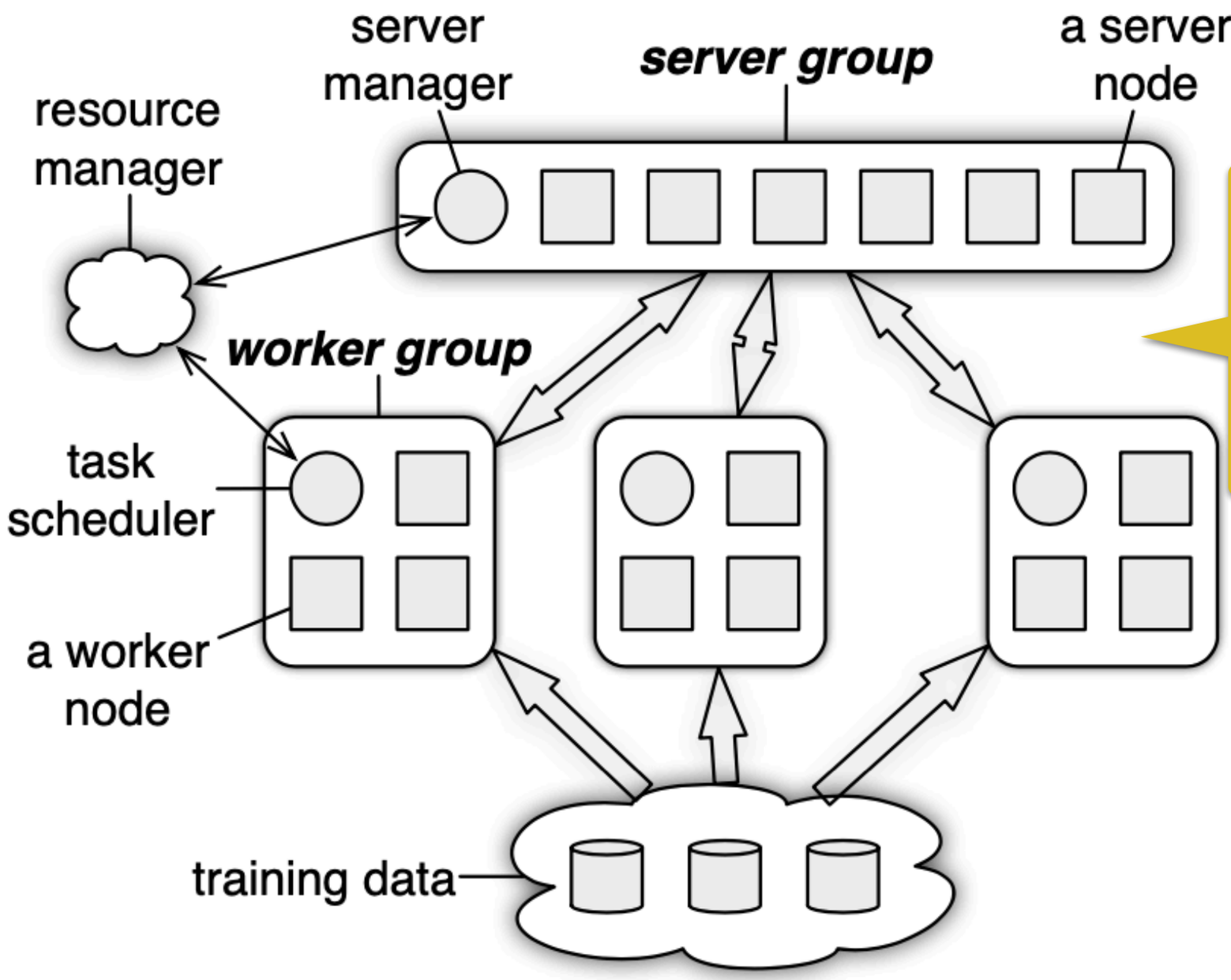


Workers send key-value pairs (gradient elements) to servers

Parameter Server



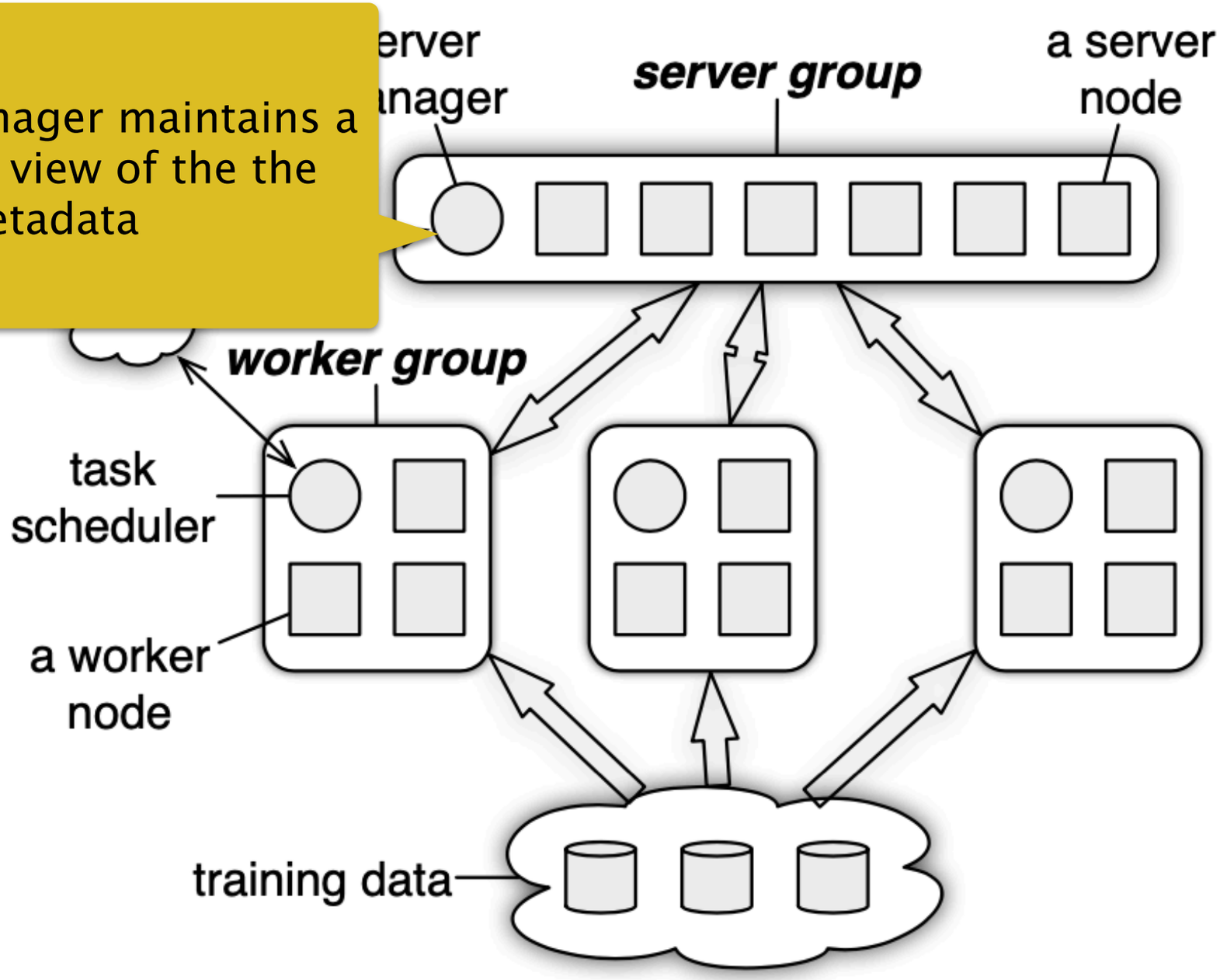
Parameter Server



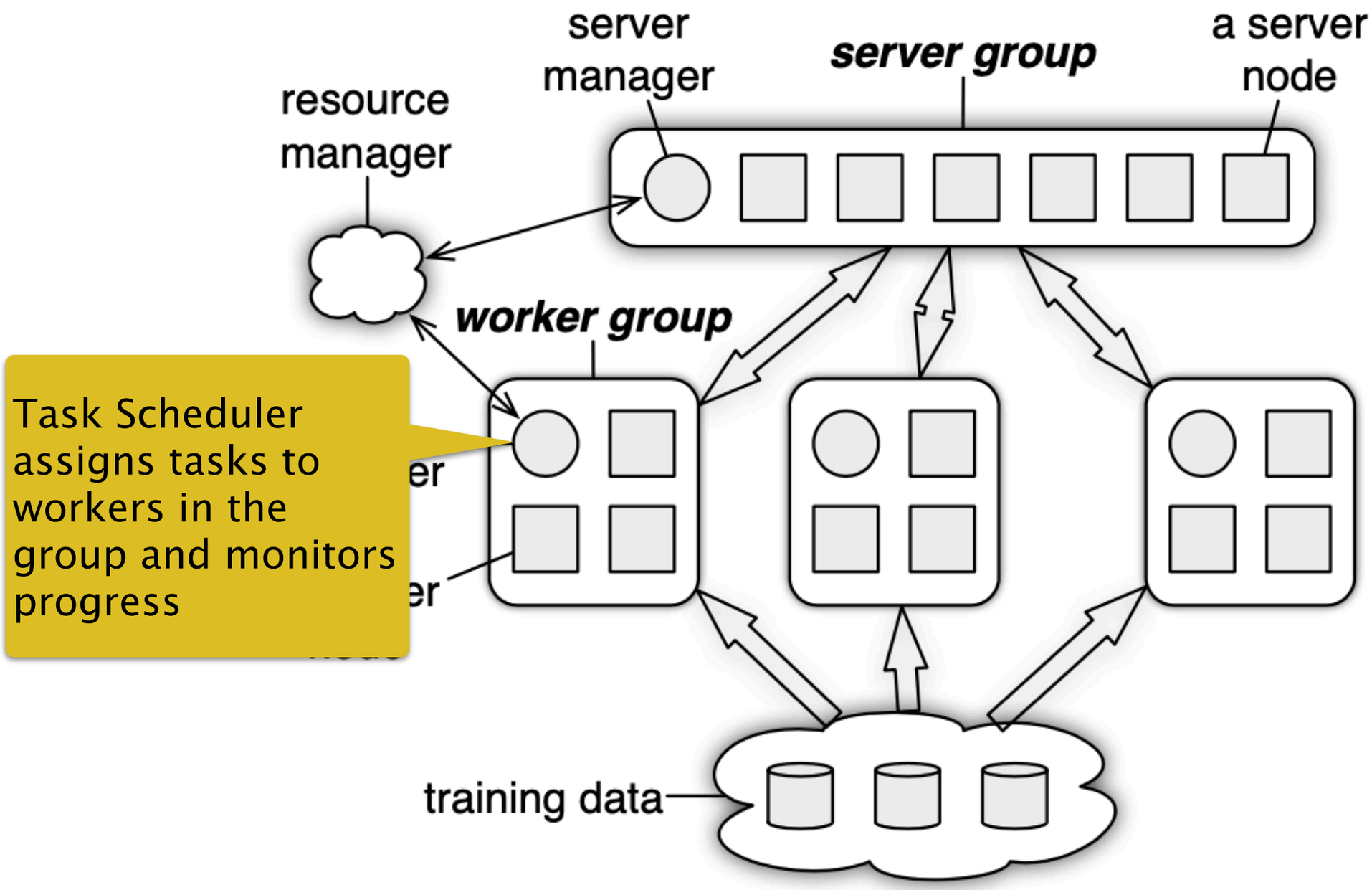
Servers send key-value pairs (updated model parameters) to workers

Parameter Server

Server Manager maintains a consistent view of the the servers metadata



Parameter Server



API

- Very simple:
  - push(R, dest): put a range of keys (with their values) to dest
  - pull(R, dest): get a range of keys (with their vsalues) from dest
- Why ranges?
  - Batch many keys in a single message — reduce traffic and I/O overhead

Flexible Consistency

- User can choose between:
  - Sequential consistency — the stricter model — equivalent to a sequential execution
  - Eventual consistency — reading stale data is not that bad for training — it increases the number of iterations
  - Bounded staleness — more flexible than the previous
- Consistency can change dynamically during execution
  - For example, the delay bound may decrease to decrease the number of iterations

Vector Clocks

- We've seen them in TreadMarks and Bayou
- Used to track progress of each individual worker and enable coordination
- Each key-value pair is associated with a vector clock (logically)
- In practice, there is one vector clock per range of keys
- Vector clocks are updated when ranges are exchanged between server and worker nodes

Vector Clocks

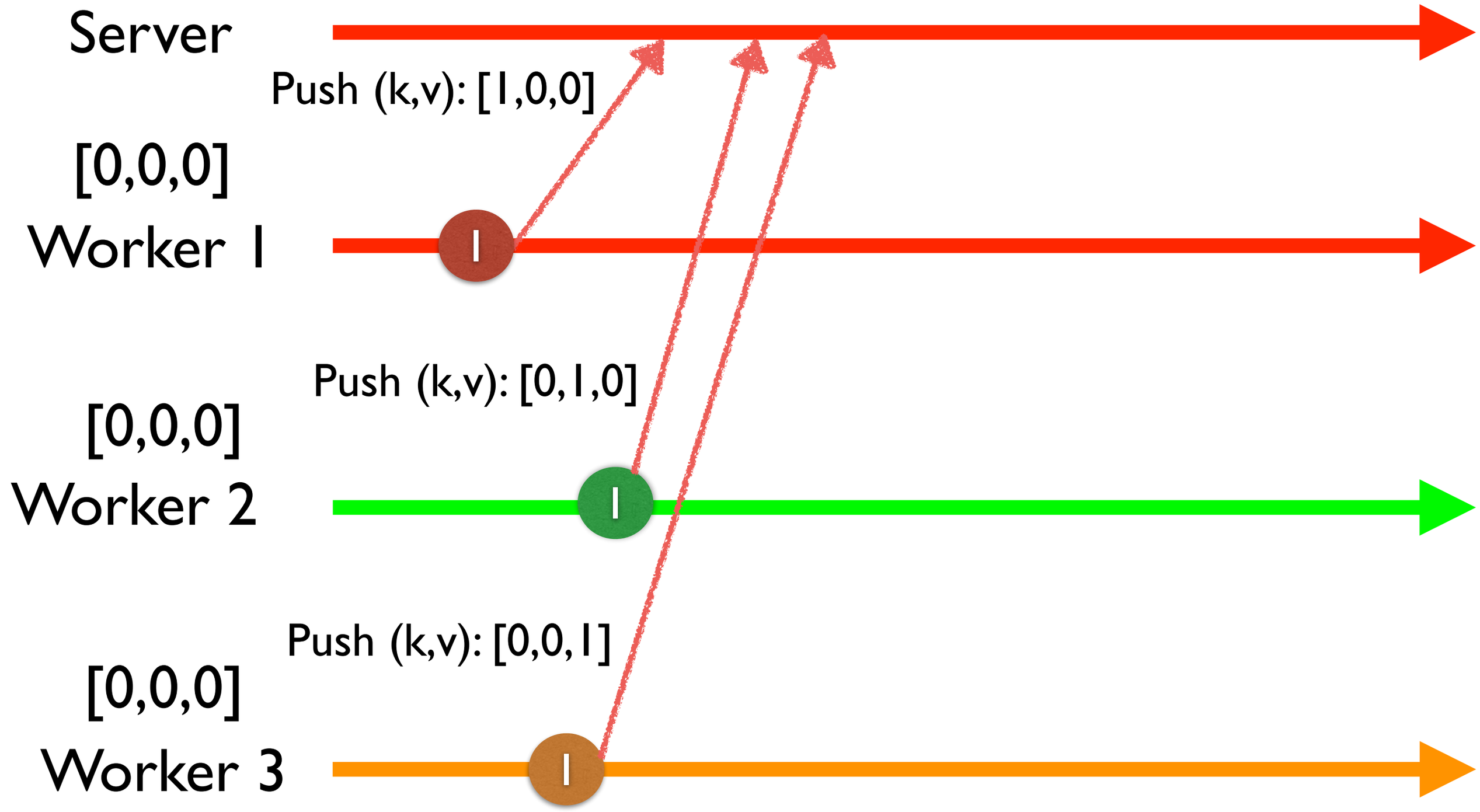
Server

[0,0,0]
Worker 1

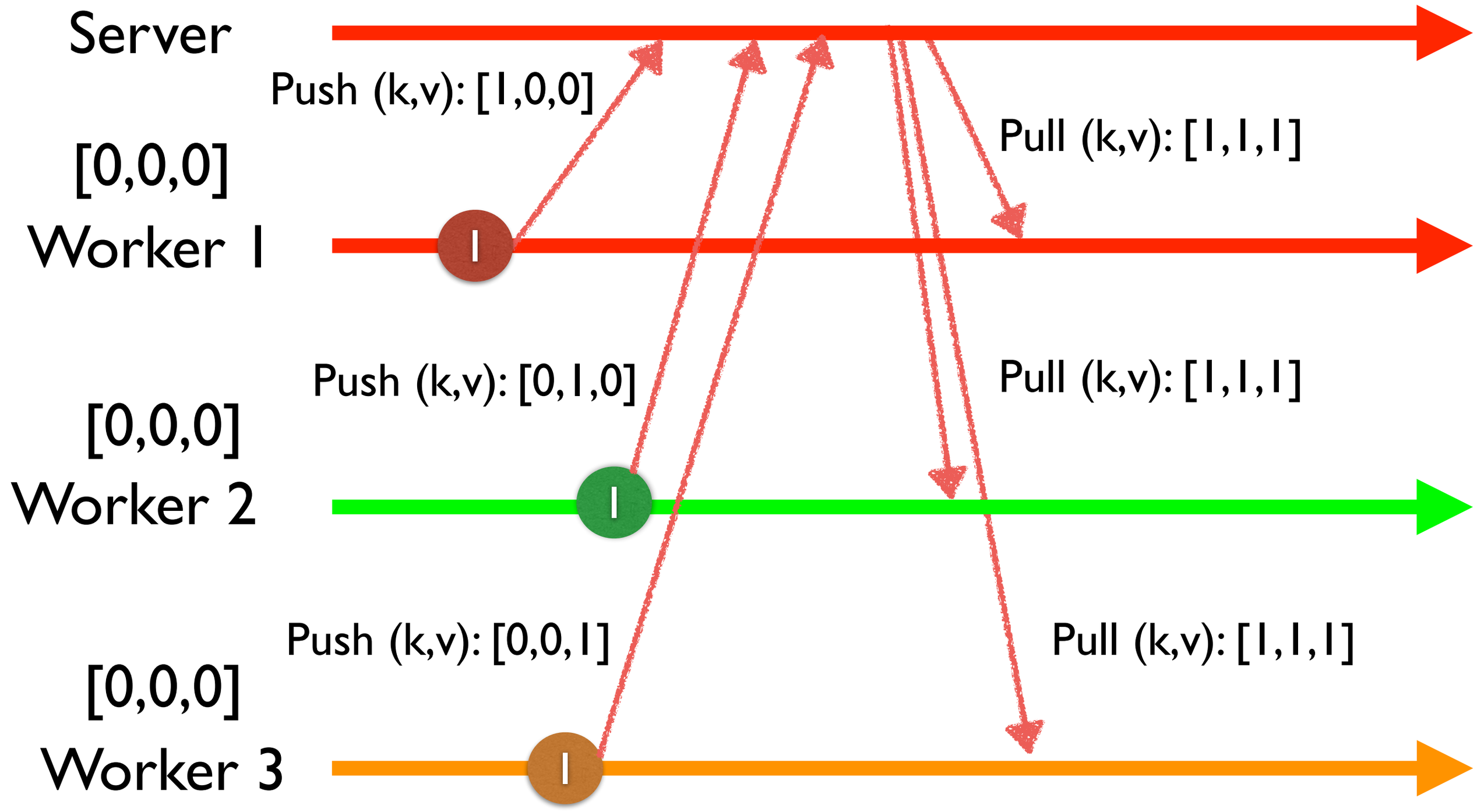
[0,0,0]
Worker 2

[0,0,0]
Worker 3

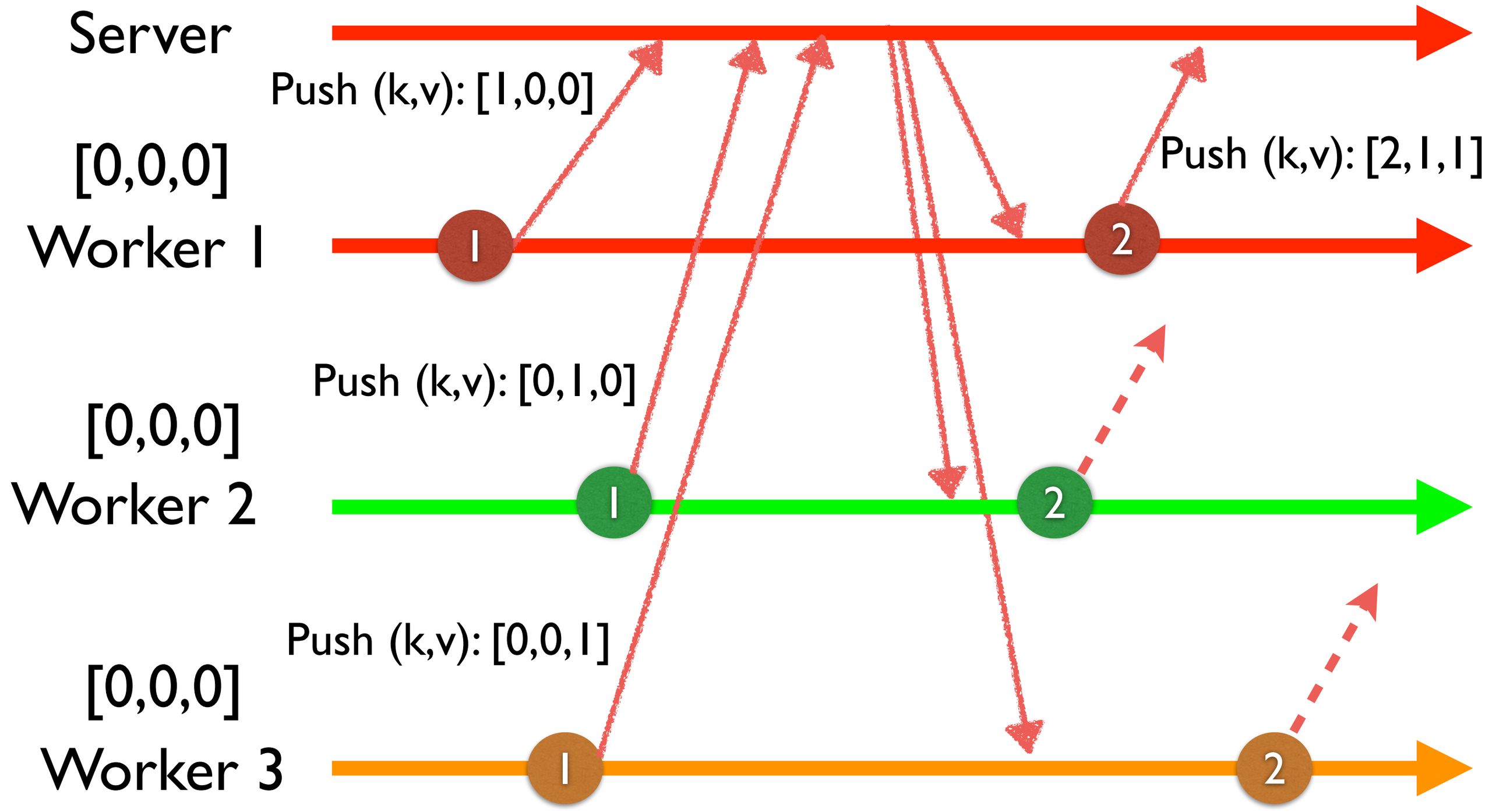
Vector Clocks



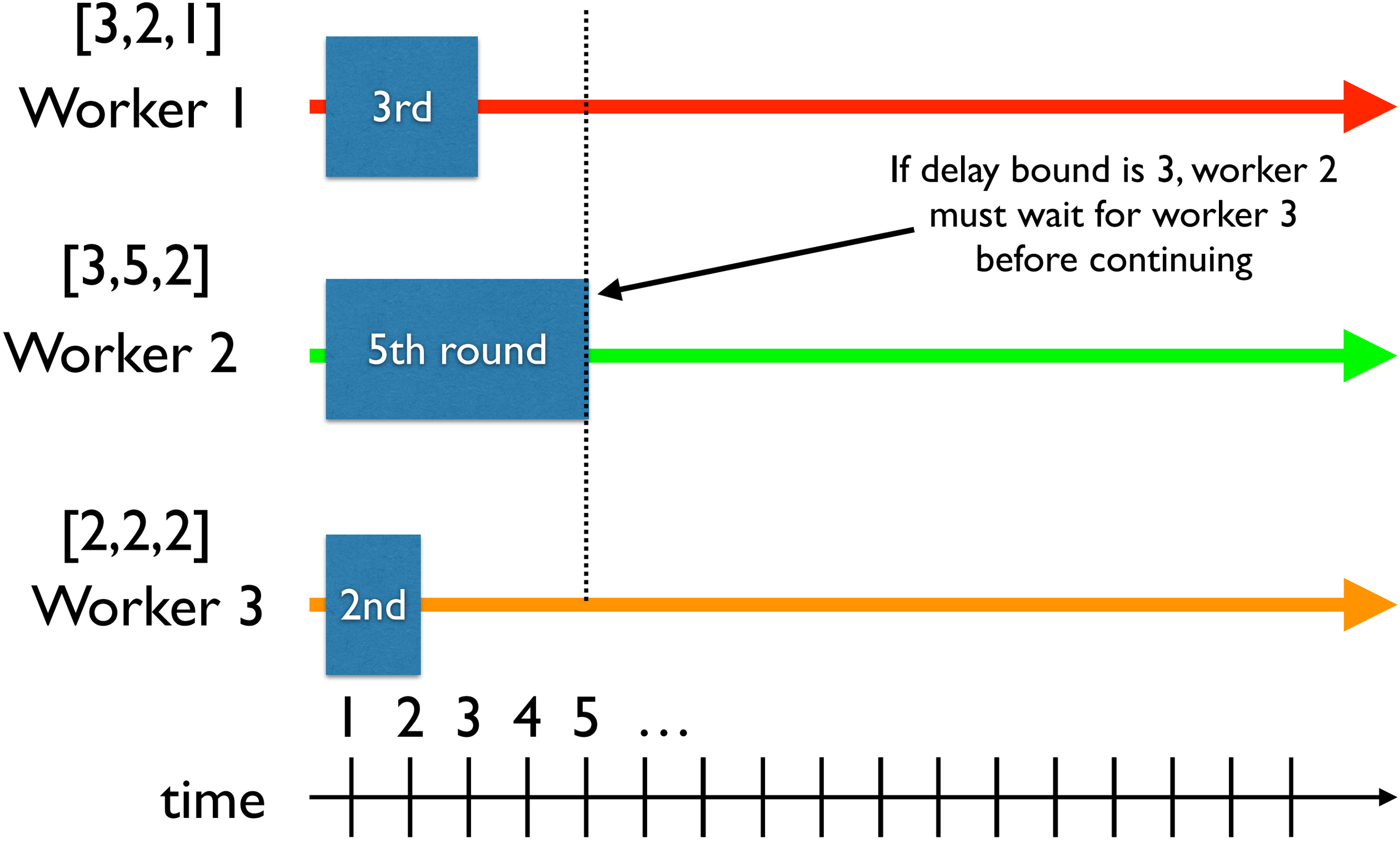
Vector Clocks



Vector Clocks

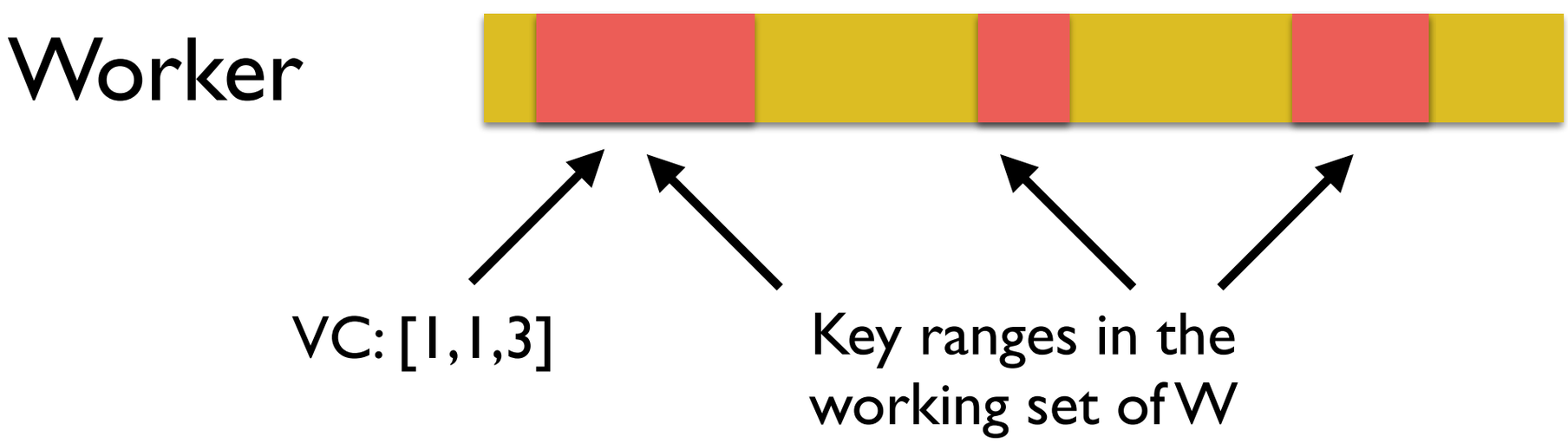


Vector Clocks

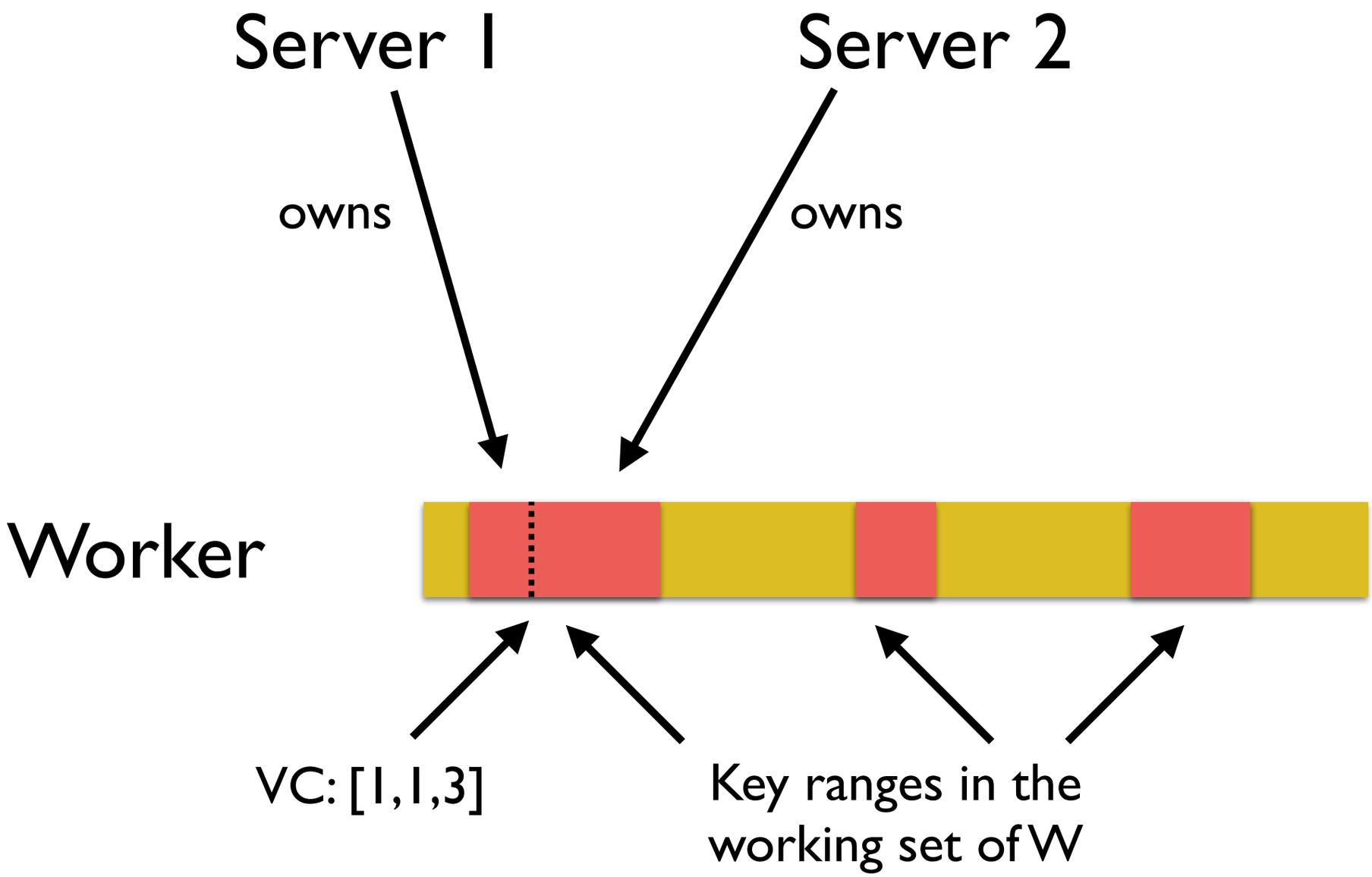


Range “Splits”

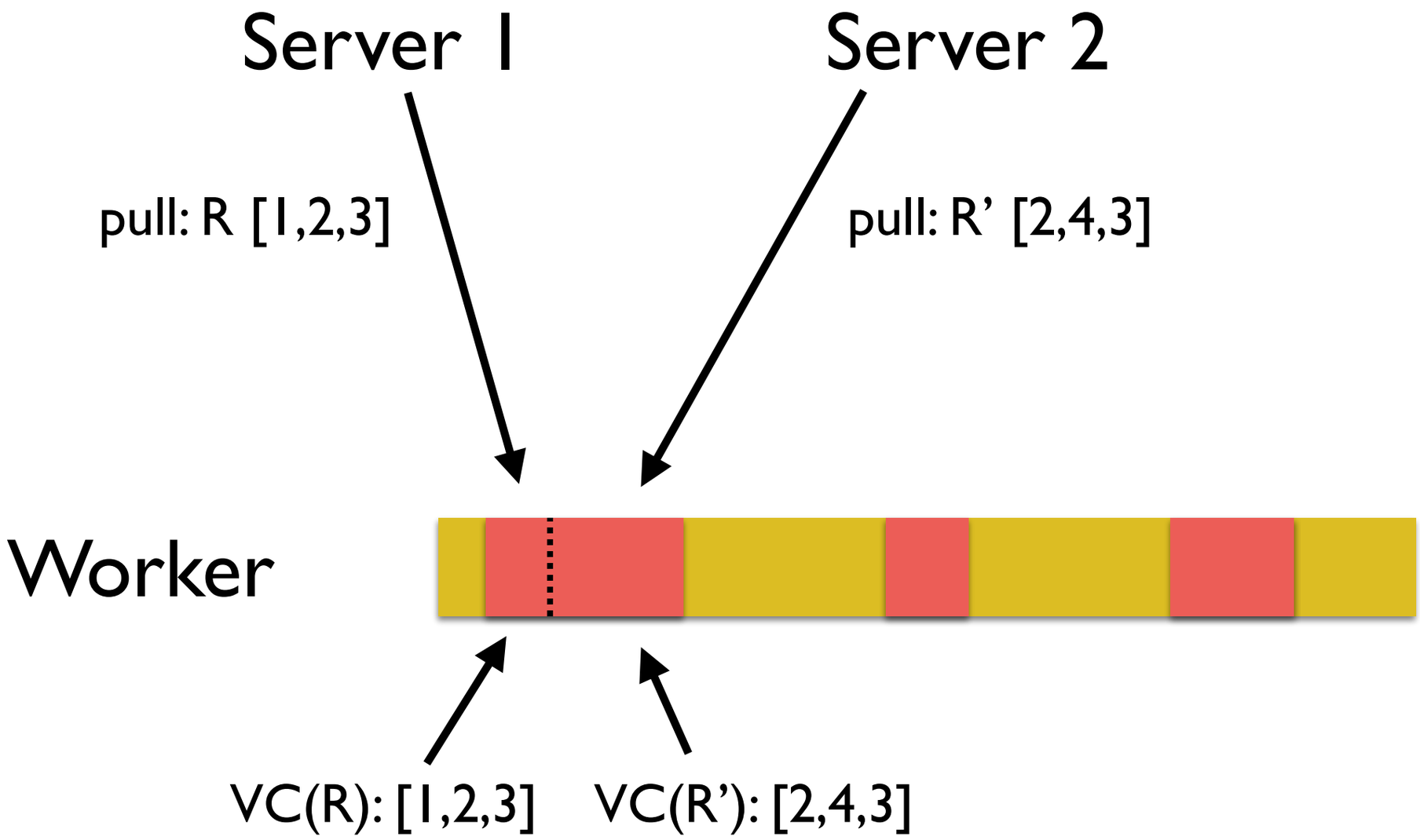
Server 1 Server 2



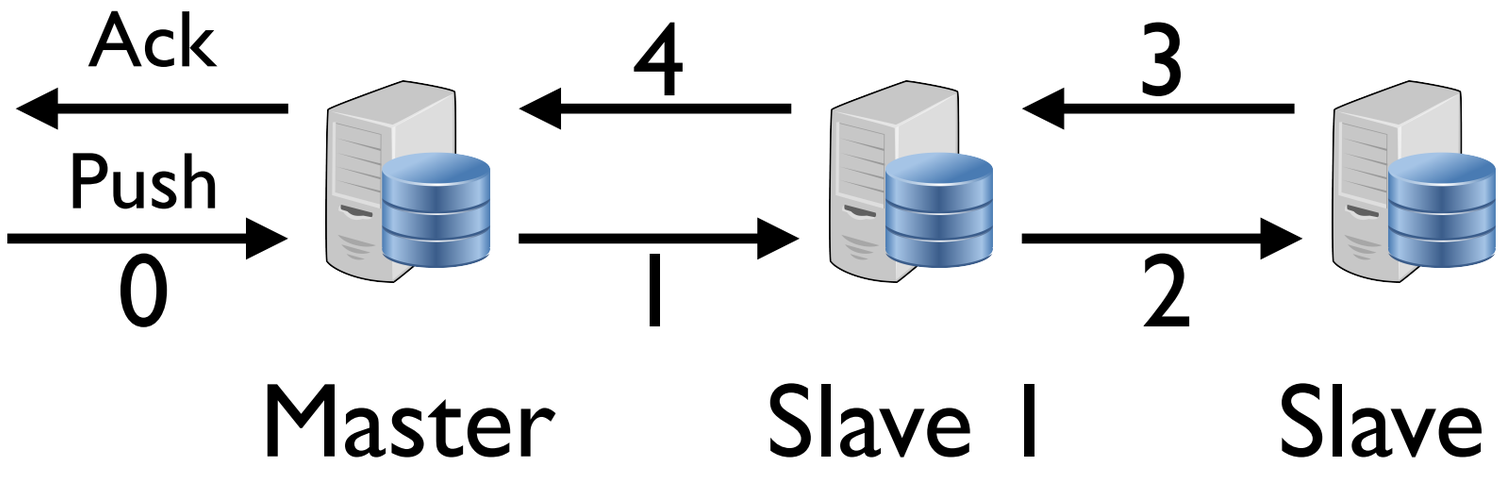
Range “Splits”



Range “Splits”



Chain Replication



All message delays are added to
the critical path of the request

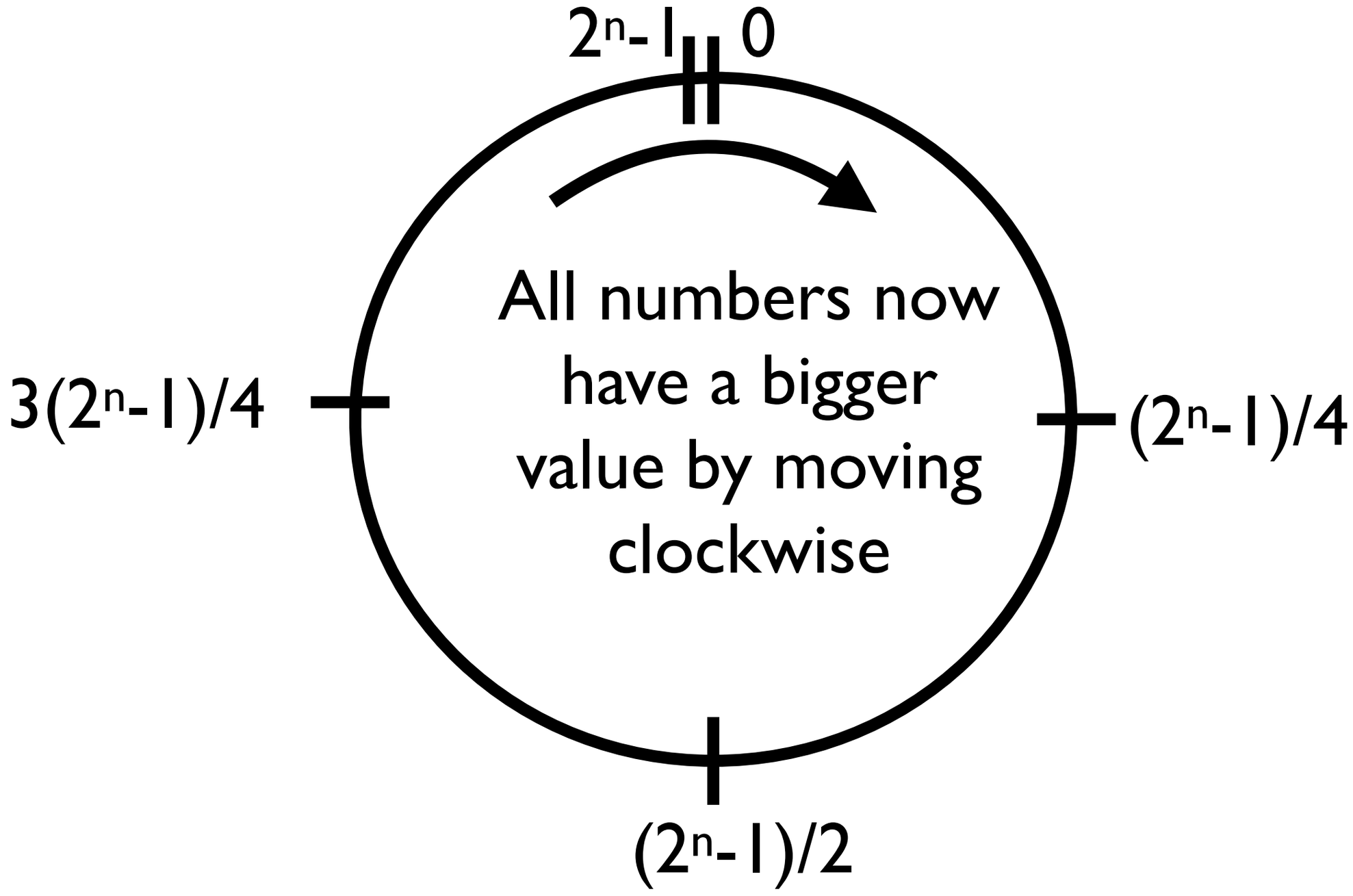
Fewer concurrent messages compared to
the typical Primary-Backup scheme

Consistent Hashing :The Ring

Given a fixed number of bits n we can represent value inclusively between 0 and 2^n-1

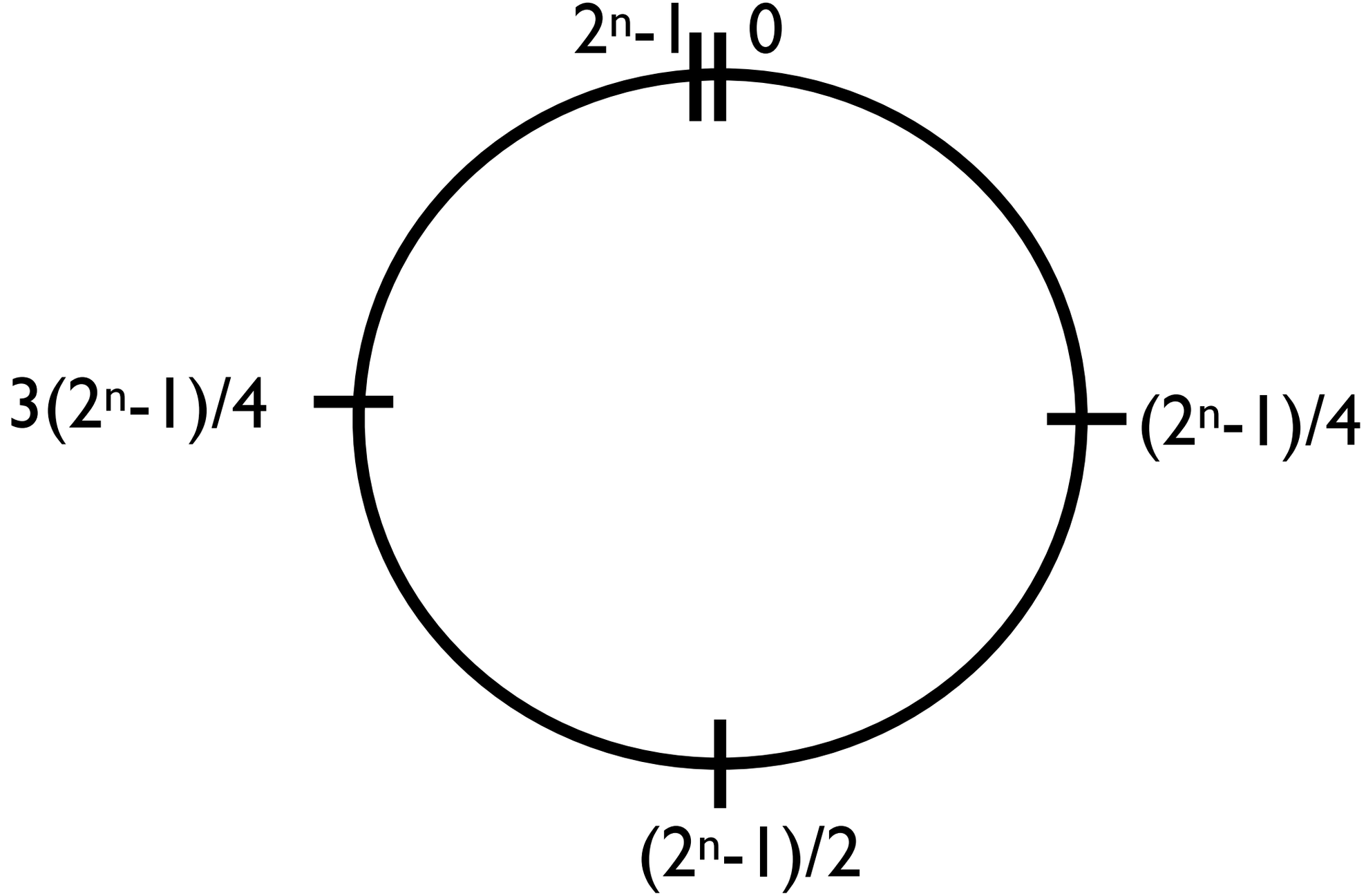
The Ring

Wrap numbers around and place them on a ring



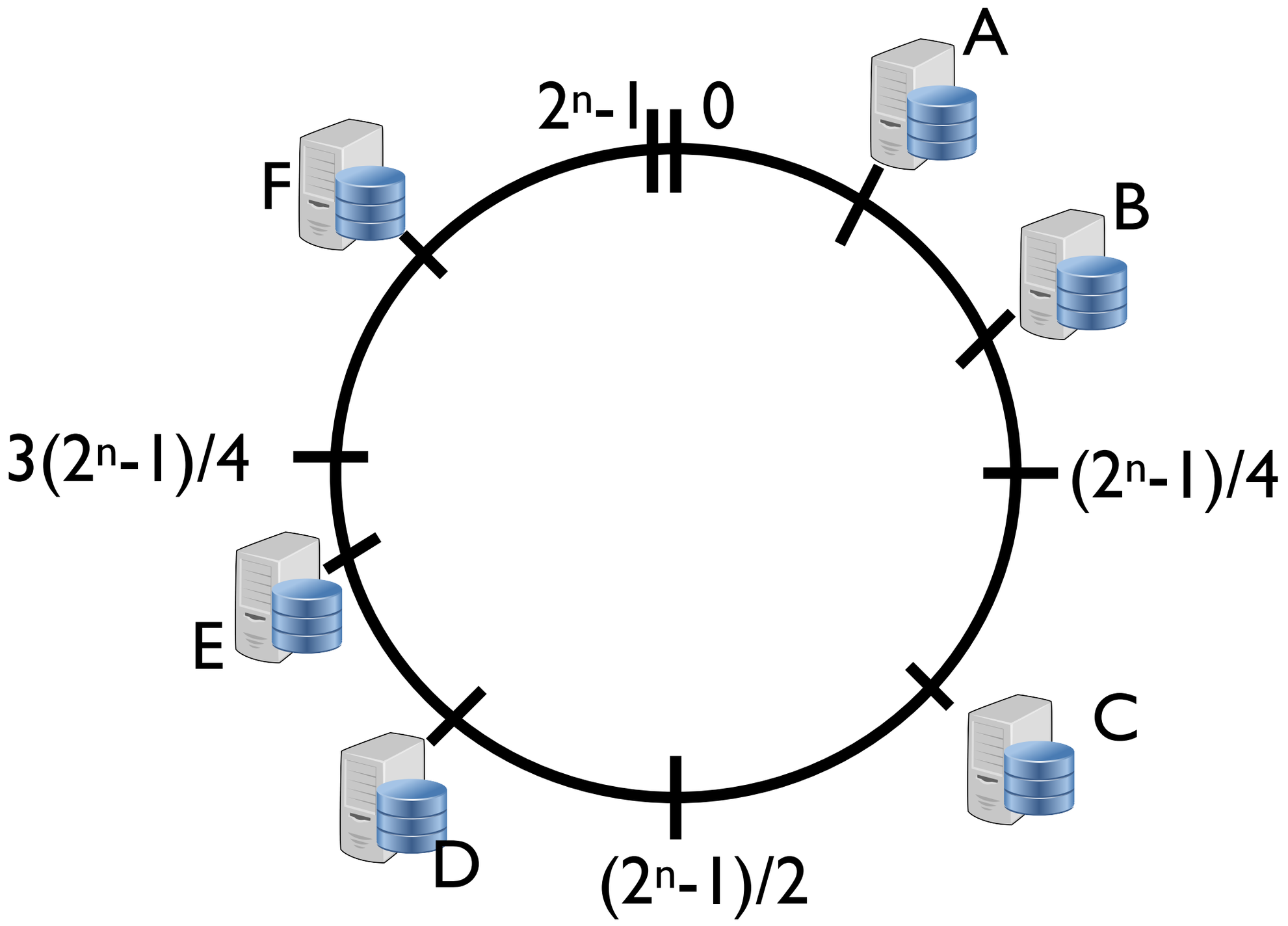
The Ring

Now use a hash function like MD5 to map values to n bit ids — positions on the ring



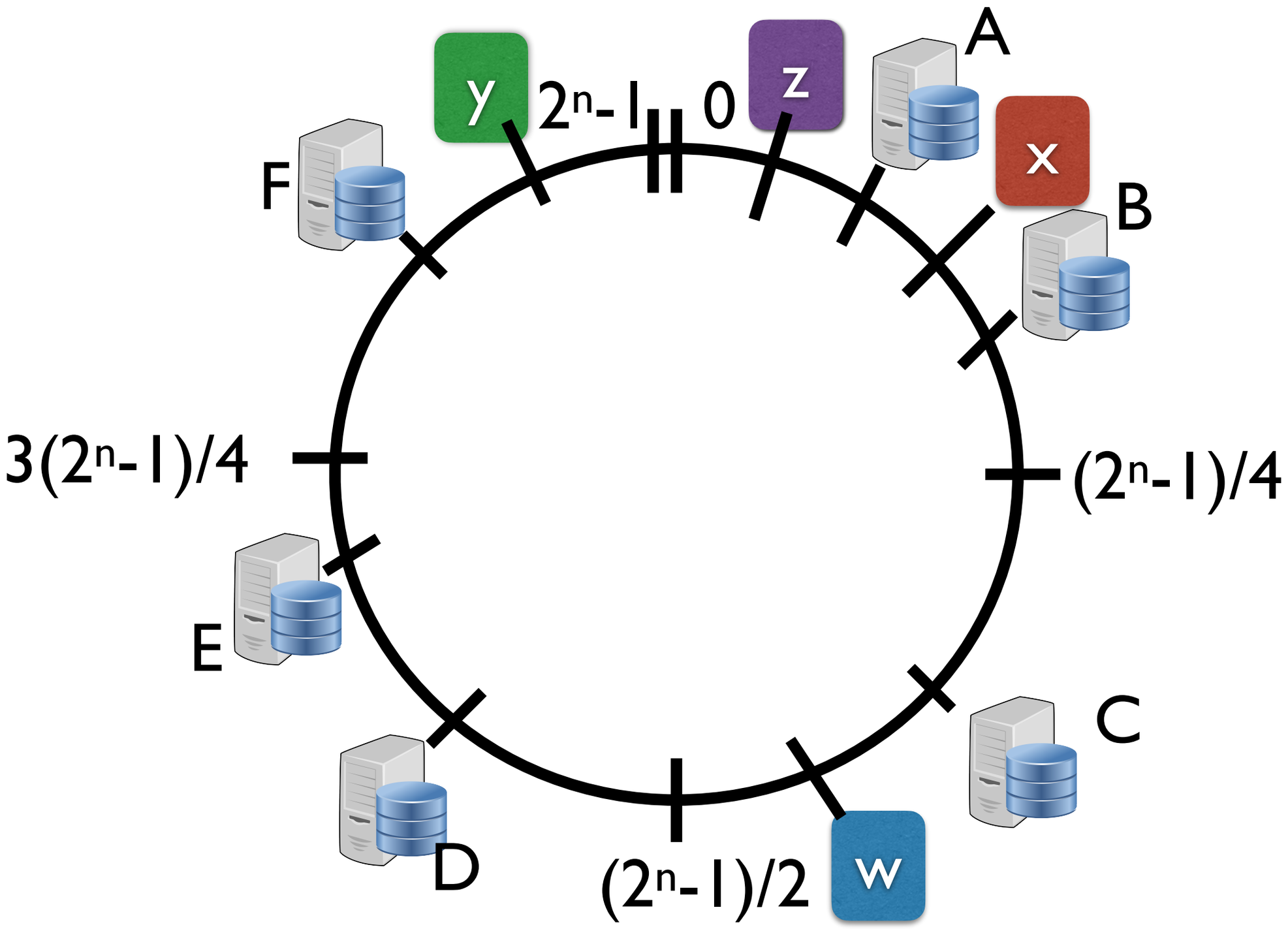
The Ring

Assign each server a random position on the ring
— eg. MD5(HostID) to map servers to ring



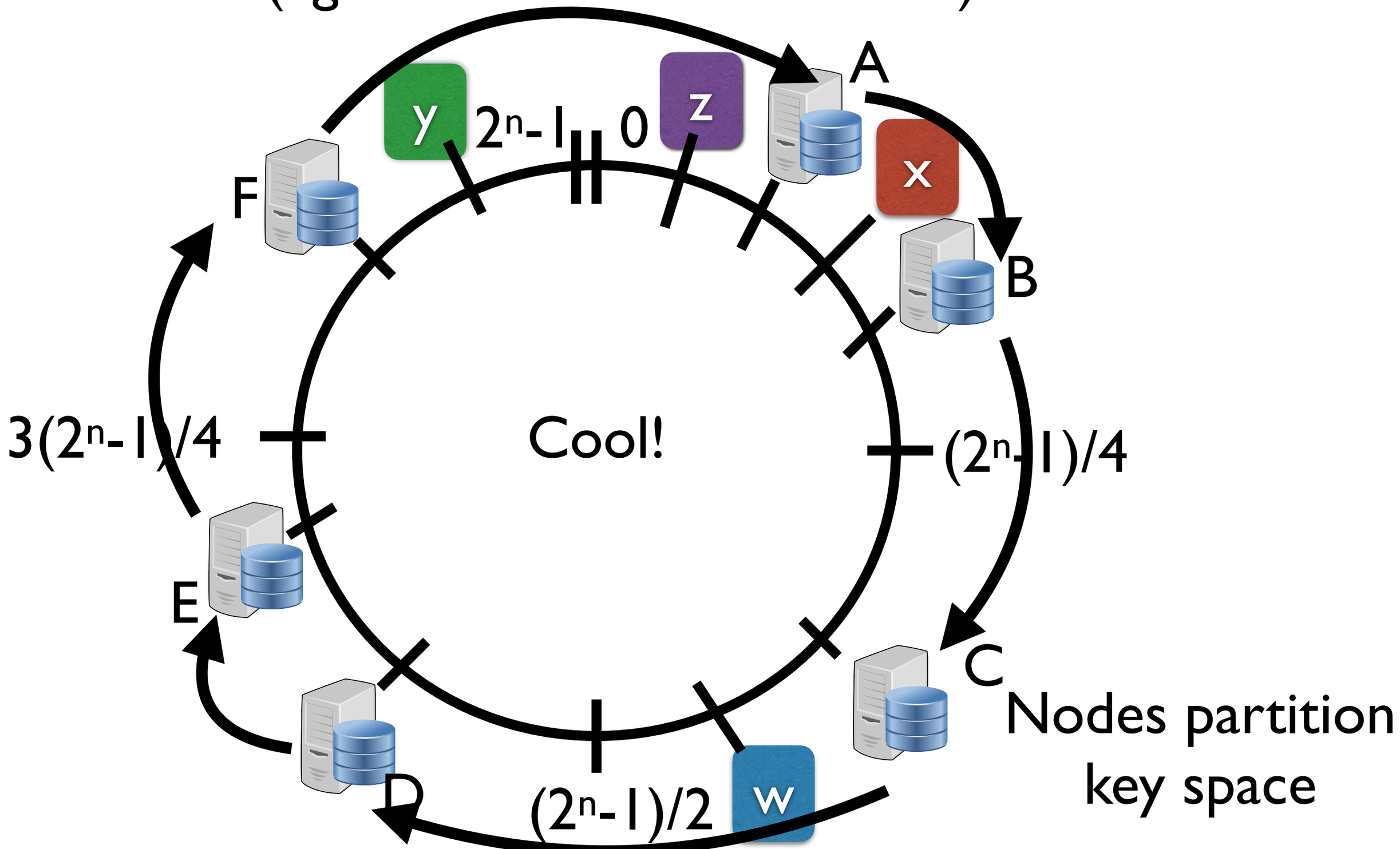
The Ring

Use MD5(key) to map keys to ring too!



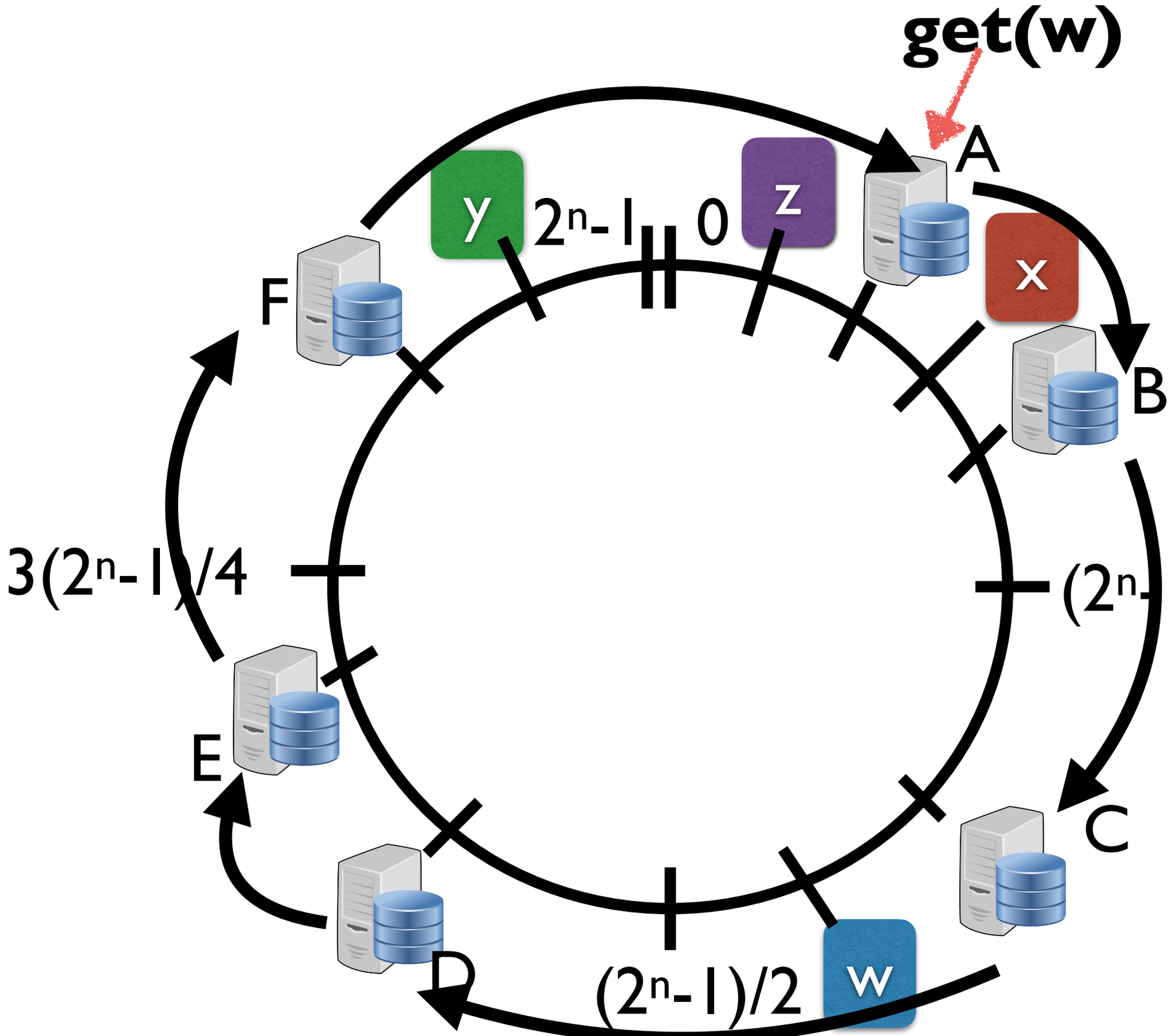
Consistent Hashing

First node to the right of key is owner
(eg first node with id >= mk)



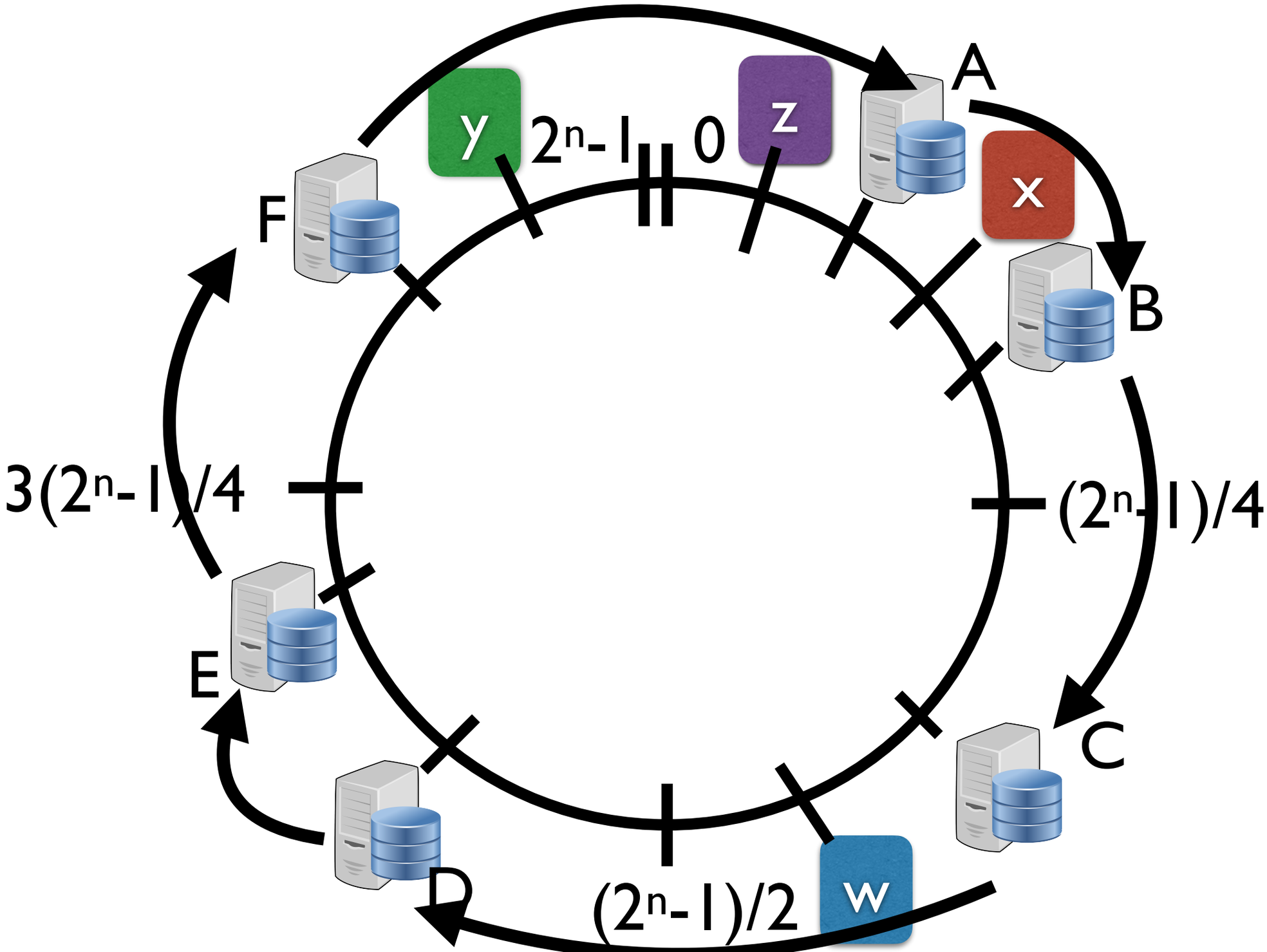
Nodes partition key space

Basic approach works by contacting any server and walking the ring (each node keeps next pointer)

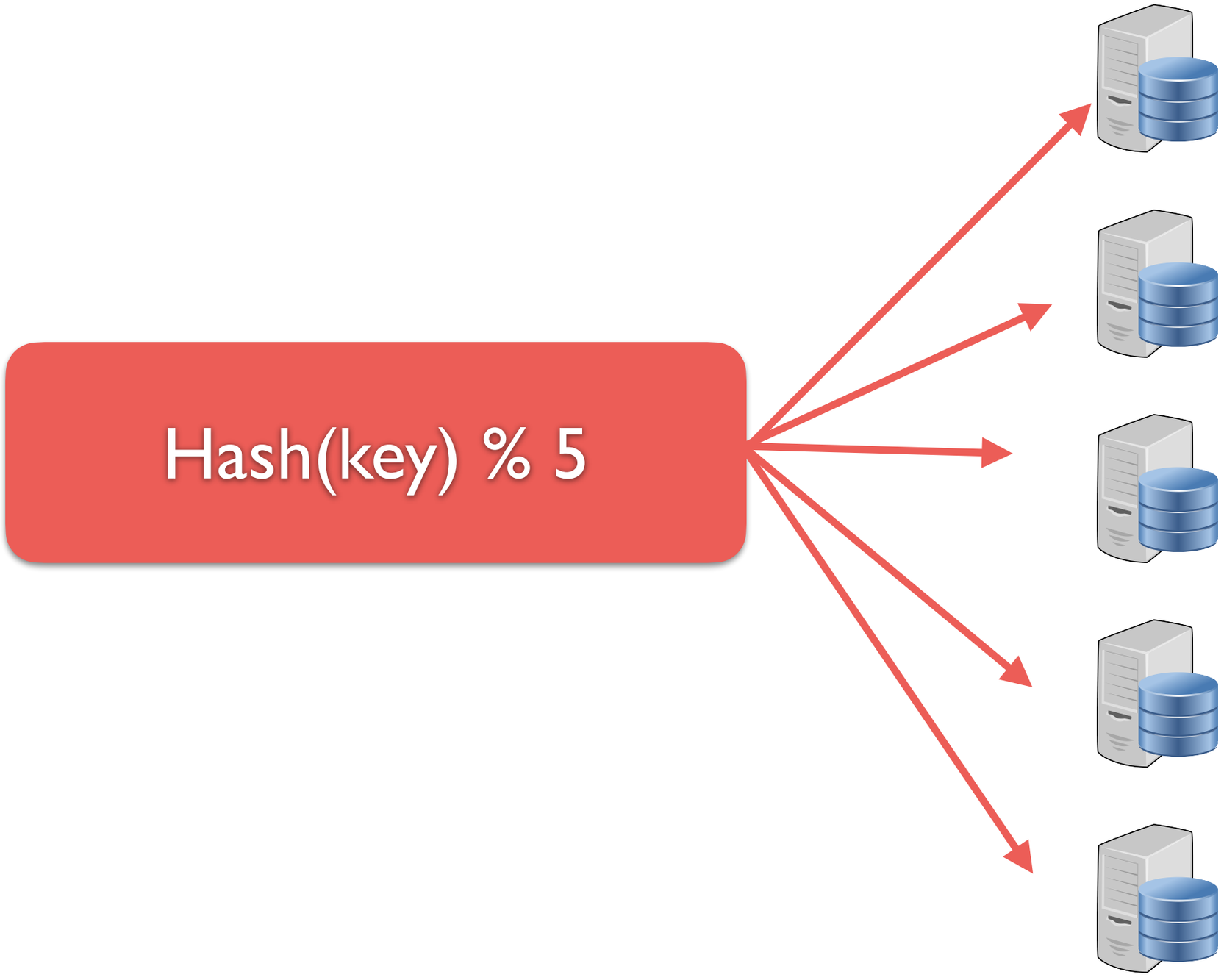


Operations are carried out by a master: The node that is identified as the key's successor

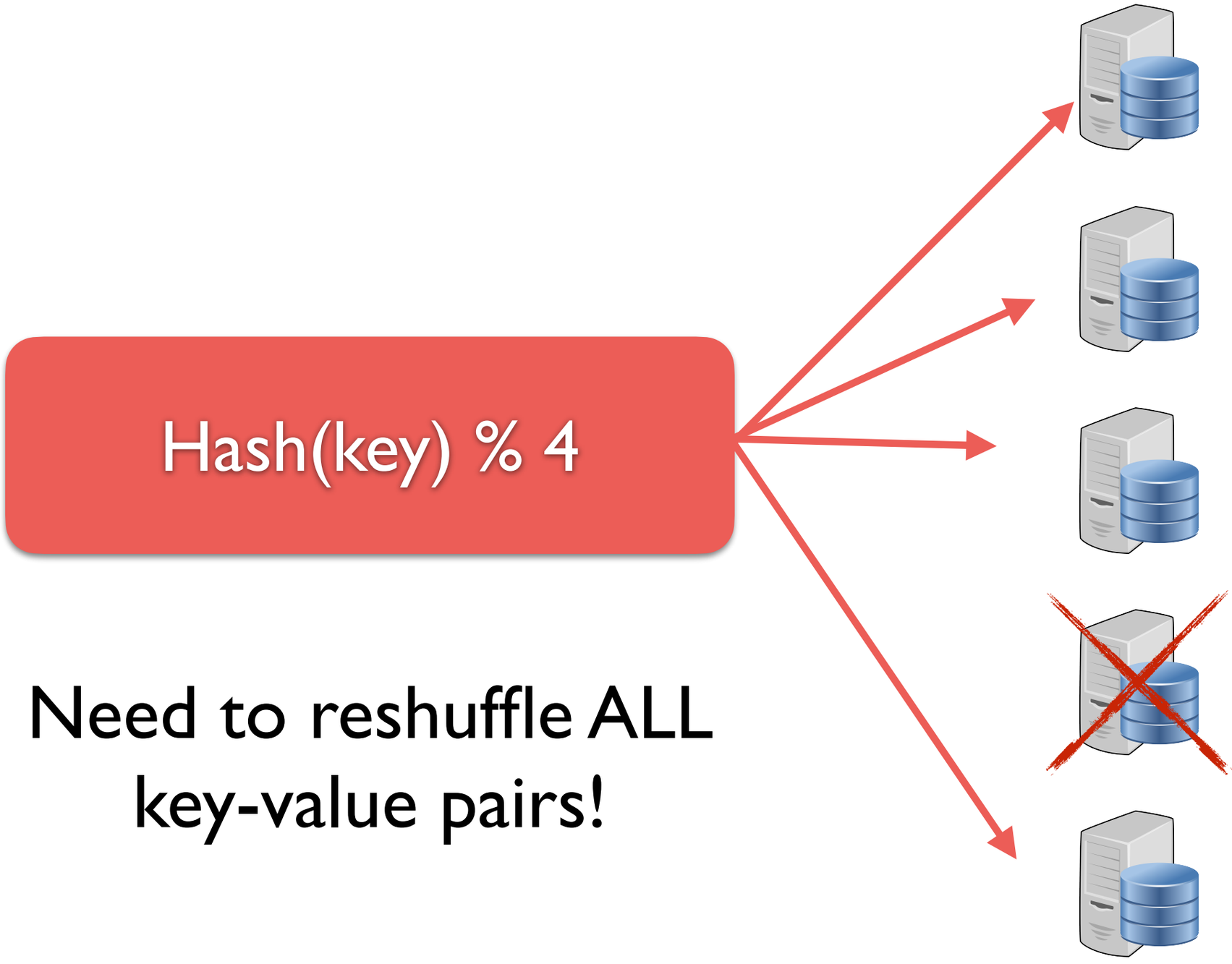
Big idea is that nodes can be inserted and removed by only talking to neighbors to exchange data rest of ring is untouched!



If mod was used…



If mod was used…



Need to reshuffle ALL key-value pairs!