An Introduction To Consistent Hashing

杨帆

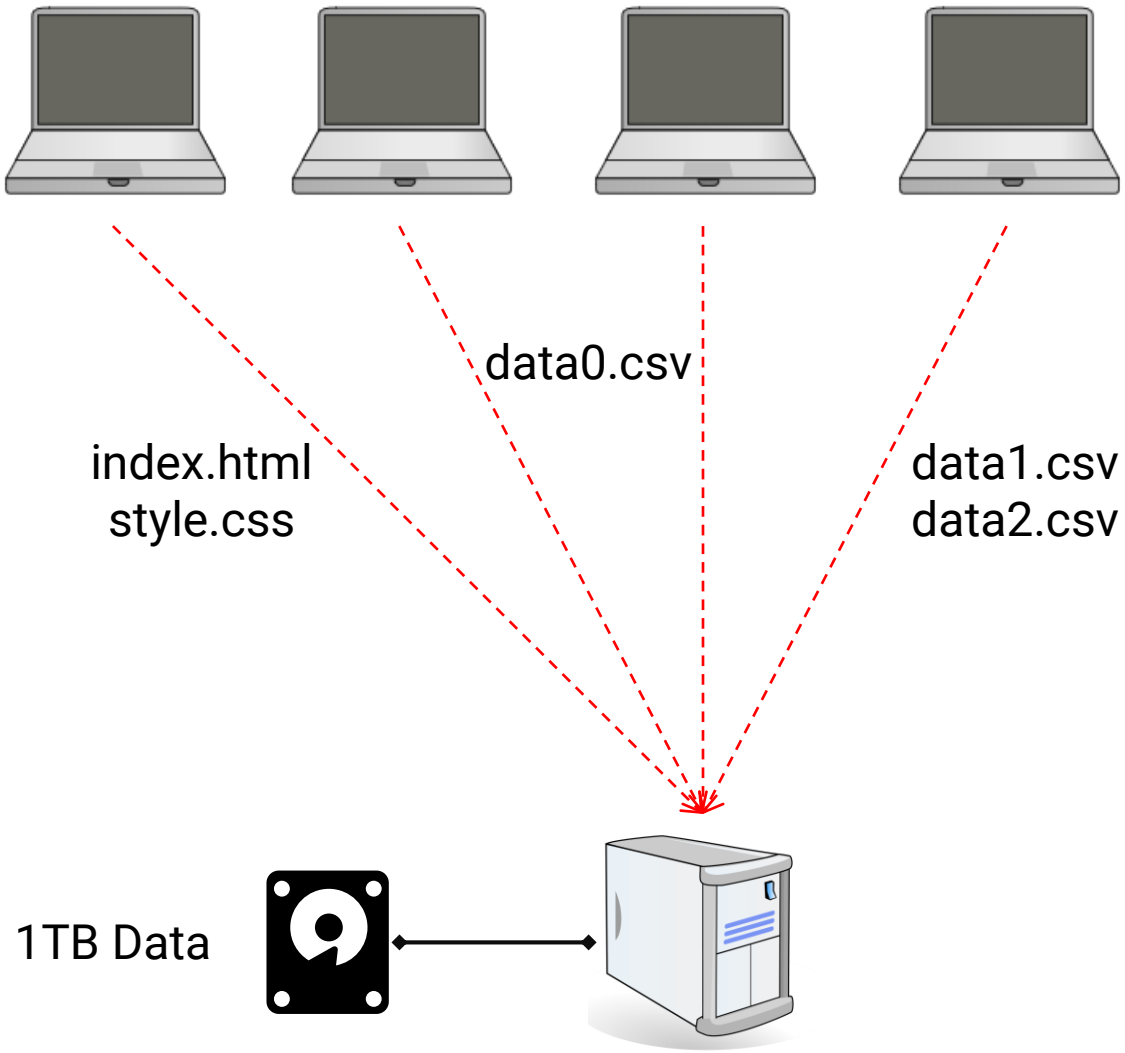
2018年1月9日

Some Clarifications On This Talk

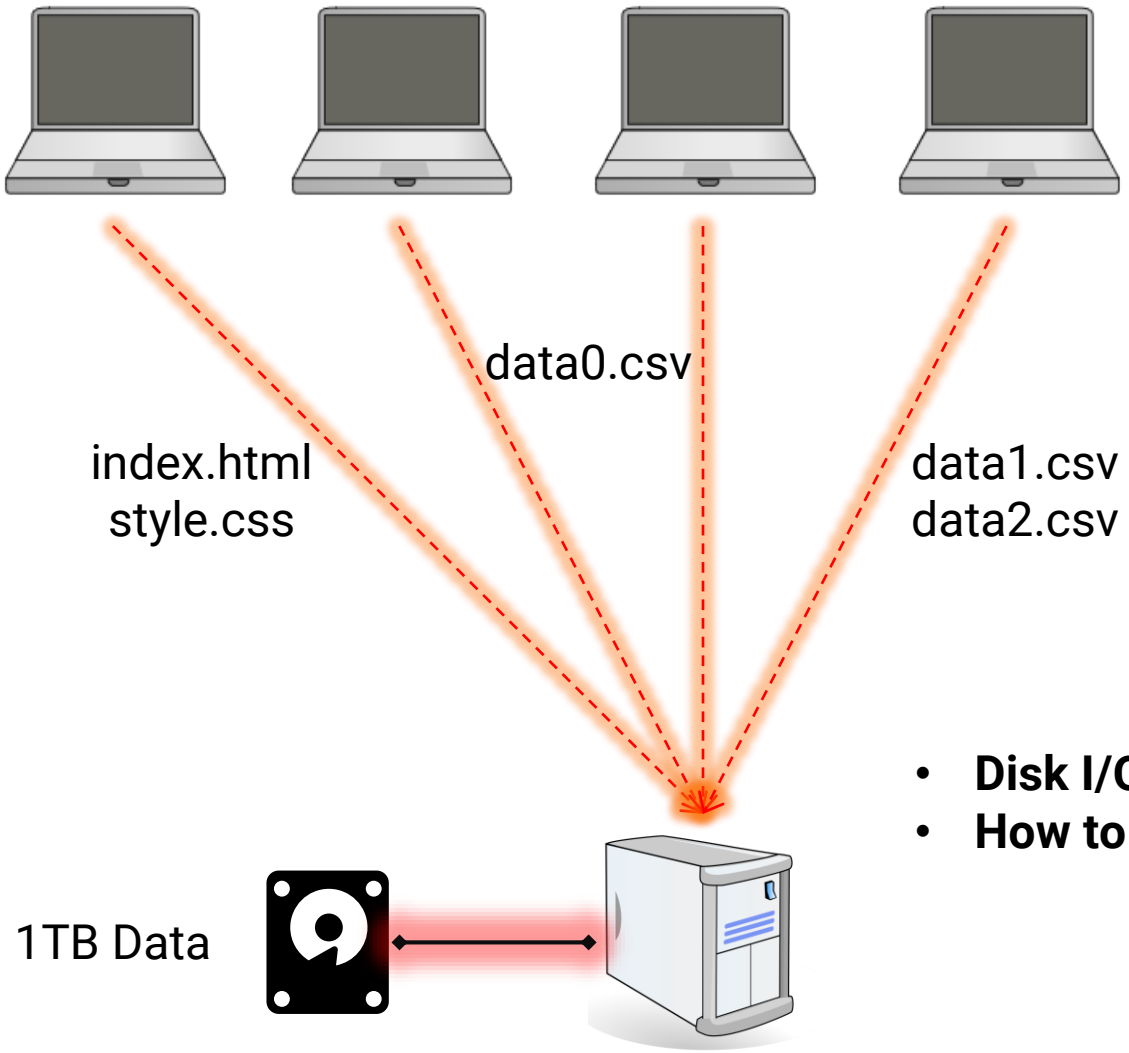
- Consistent hashing is an algorithm that is widely used in distributed systems

- We share it here because:
  - It's simple enough to be finished in 10 minutes
  - It's a fundamental algorithm that has many real-world applications
    - CDN, P2P network, load balancing, data partitioning, high availability
    - Amazon DynamoDB, Openstack Swift, Memcached, HAProxy, Linux LVS
  - Recently Google released an improved version of the algorithm, which was quickly adopted by industry community

Content Delivery

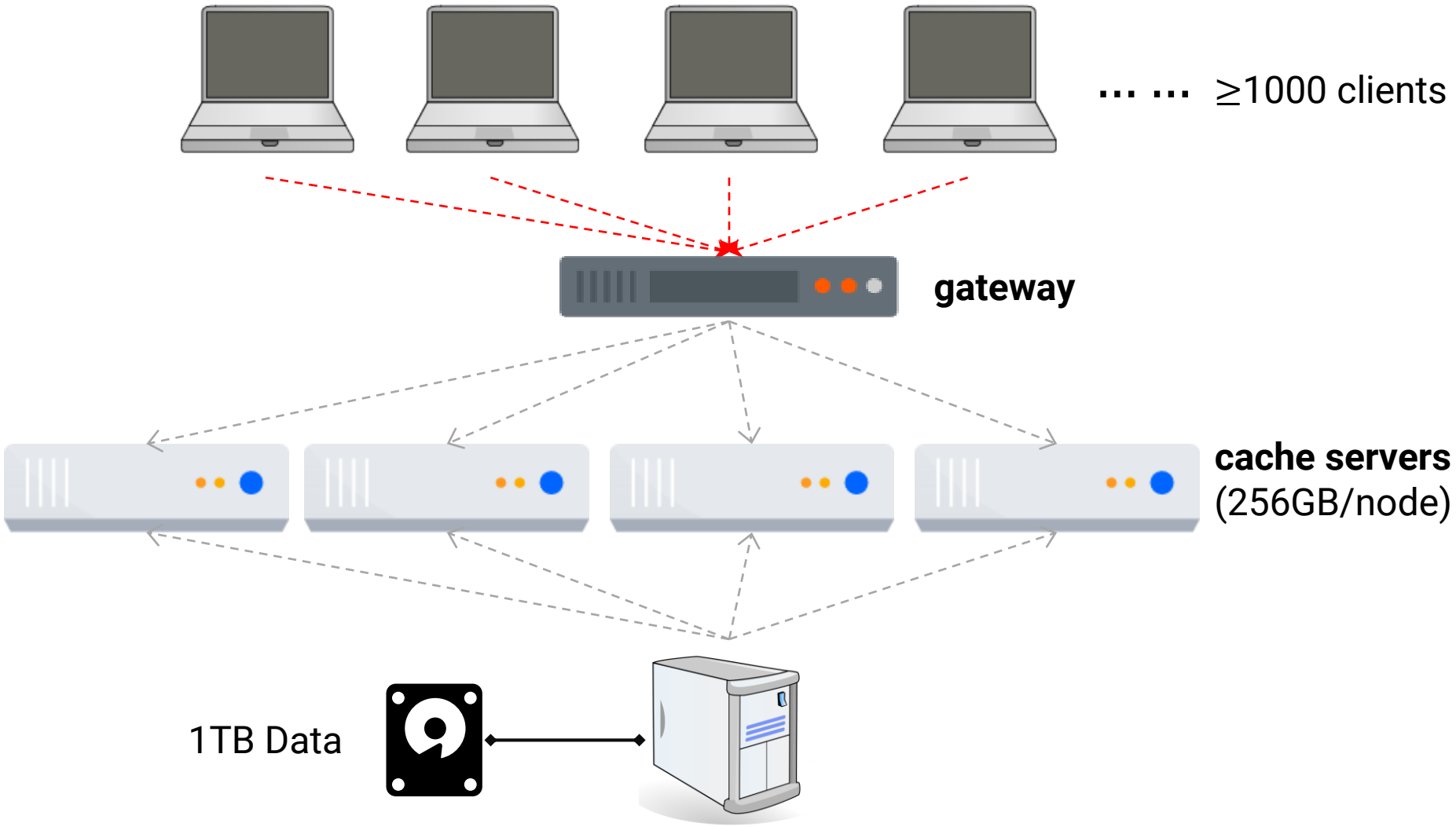


Content Delivery: The Need for Caching

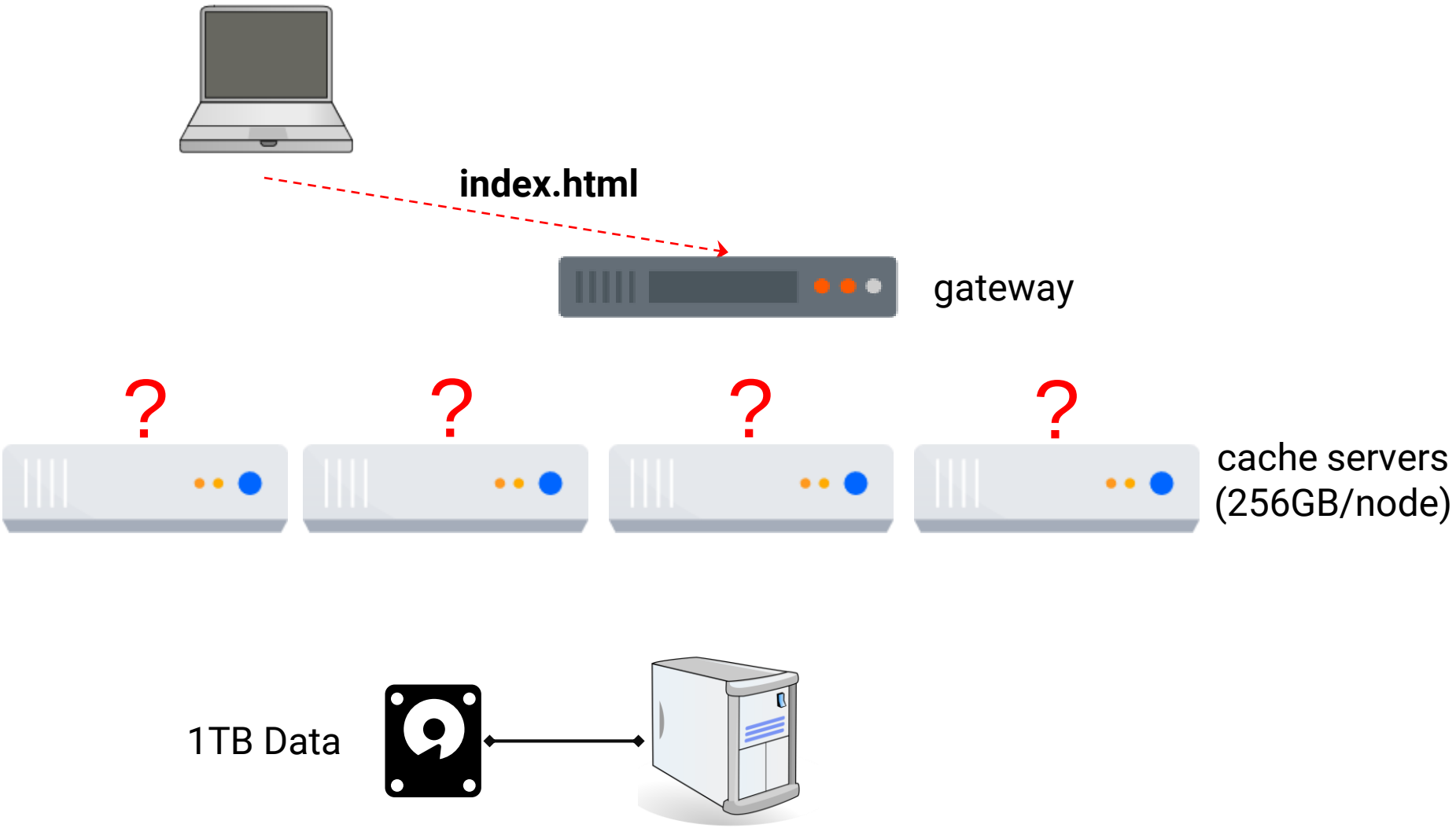


- **Disk I/O is the bottleneck**
- **How to avoid it? Caching!!!**

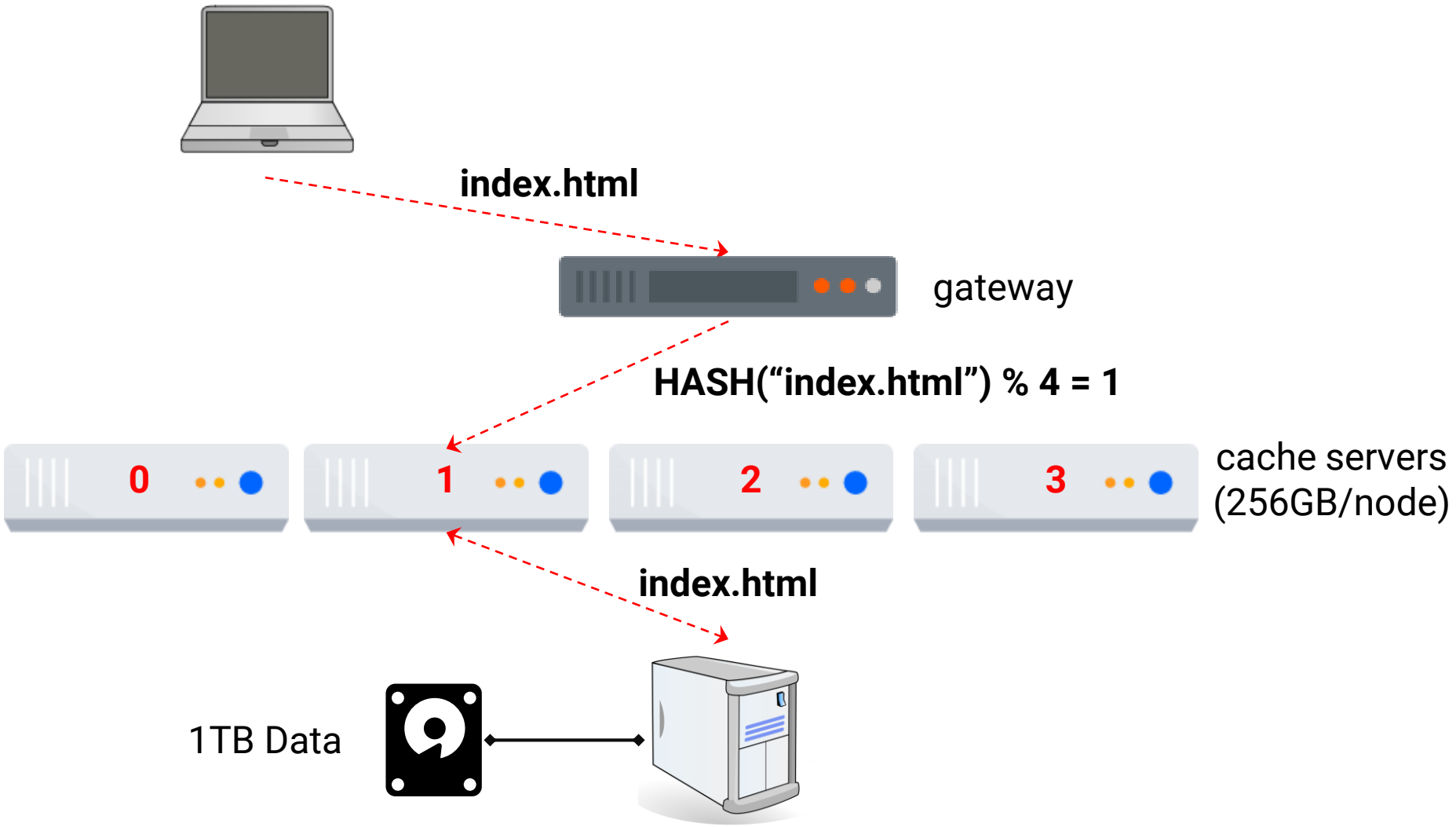
Distributed In-Memory Caching



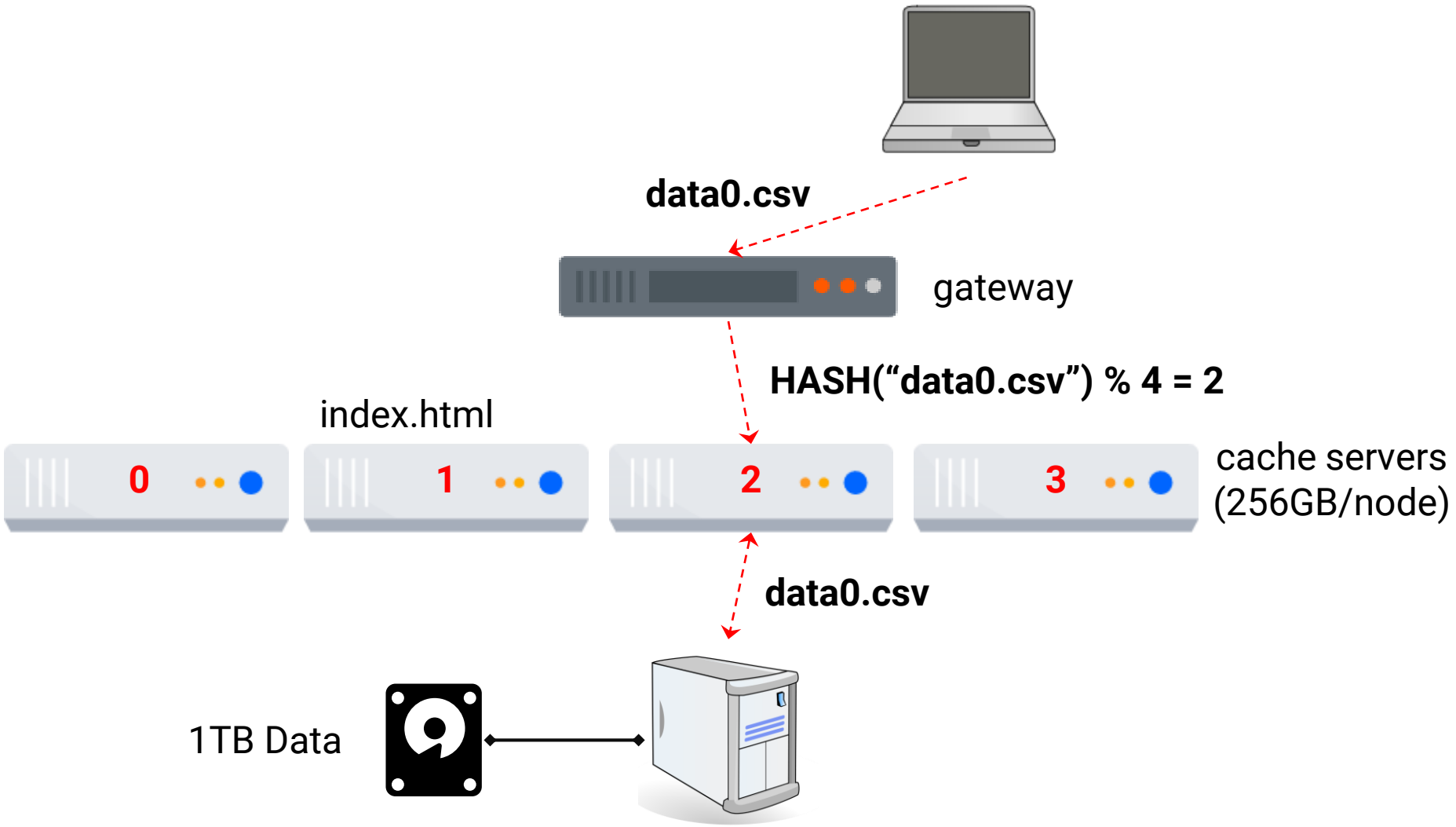
Question: How to Find the Right Cache Server?



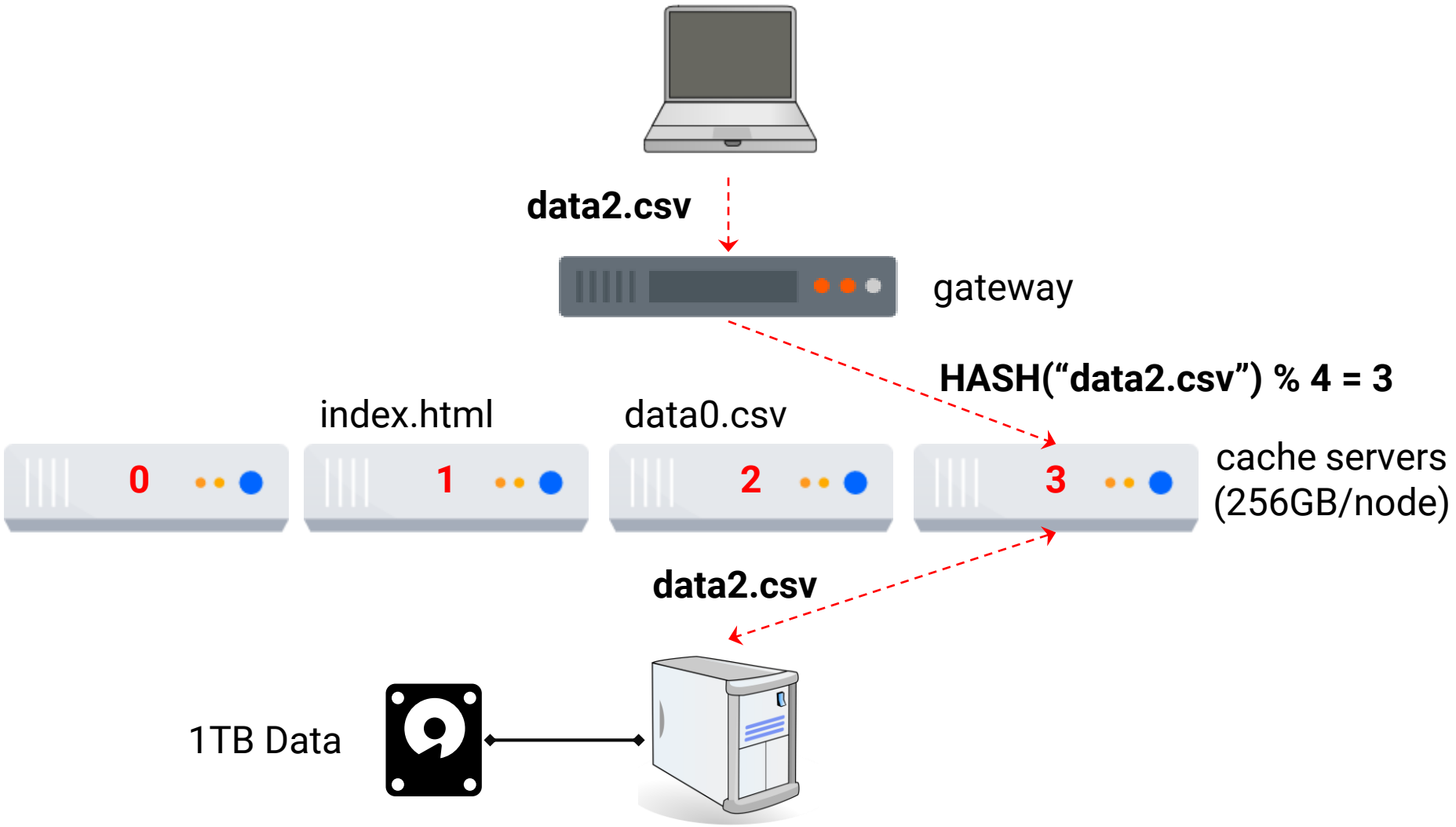
A Simple Solution: Hashing



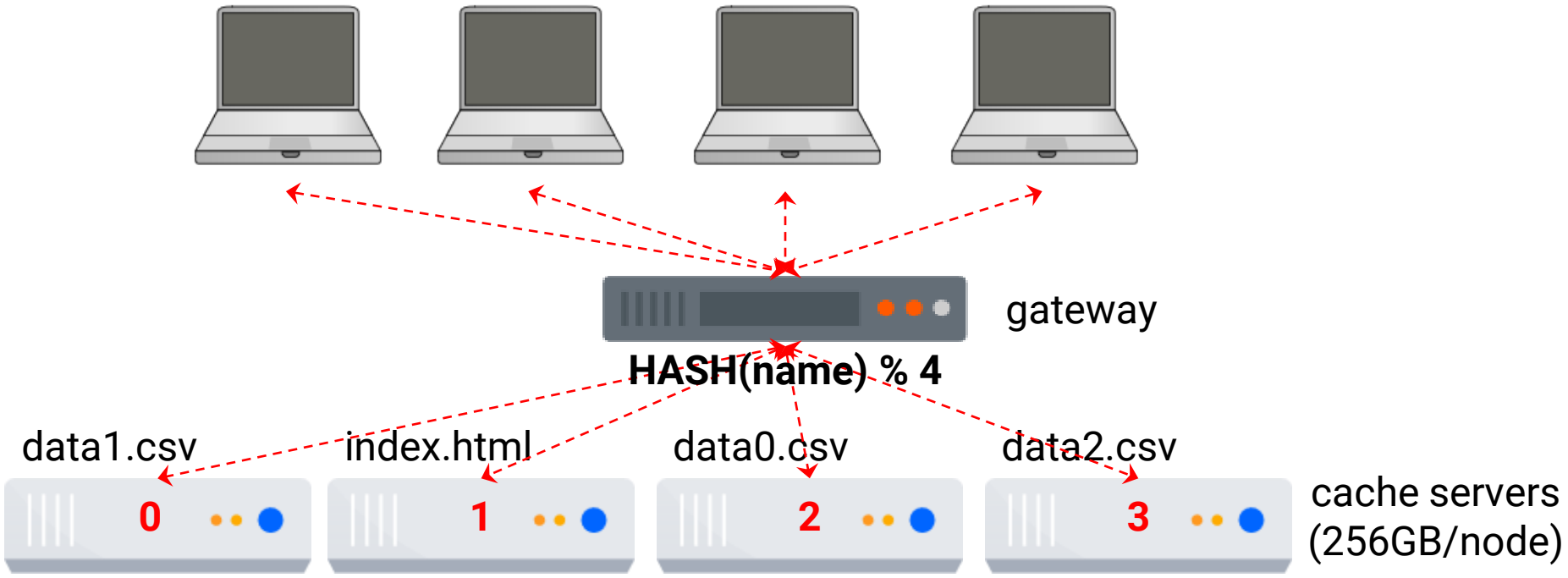
A Simple Solution: Hashing

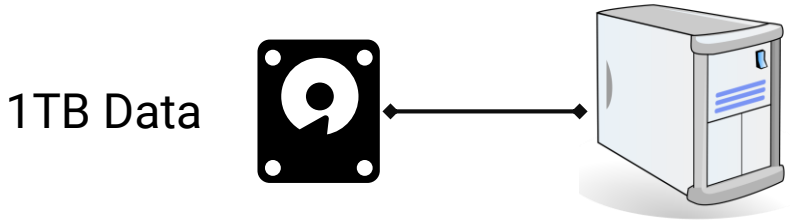


A Simple Solution: Hashing

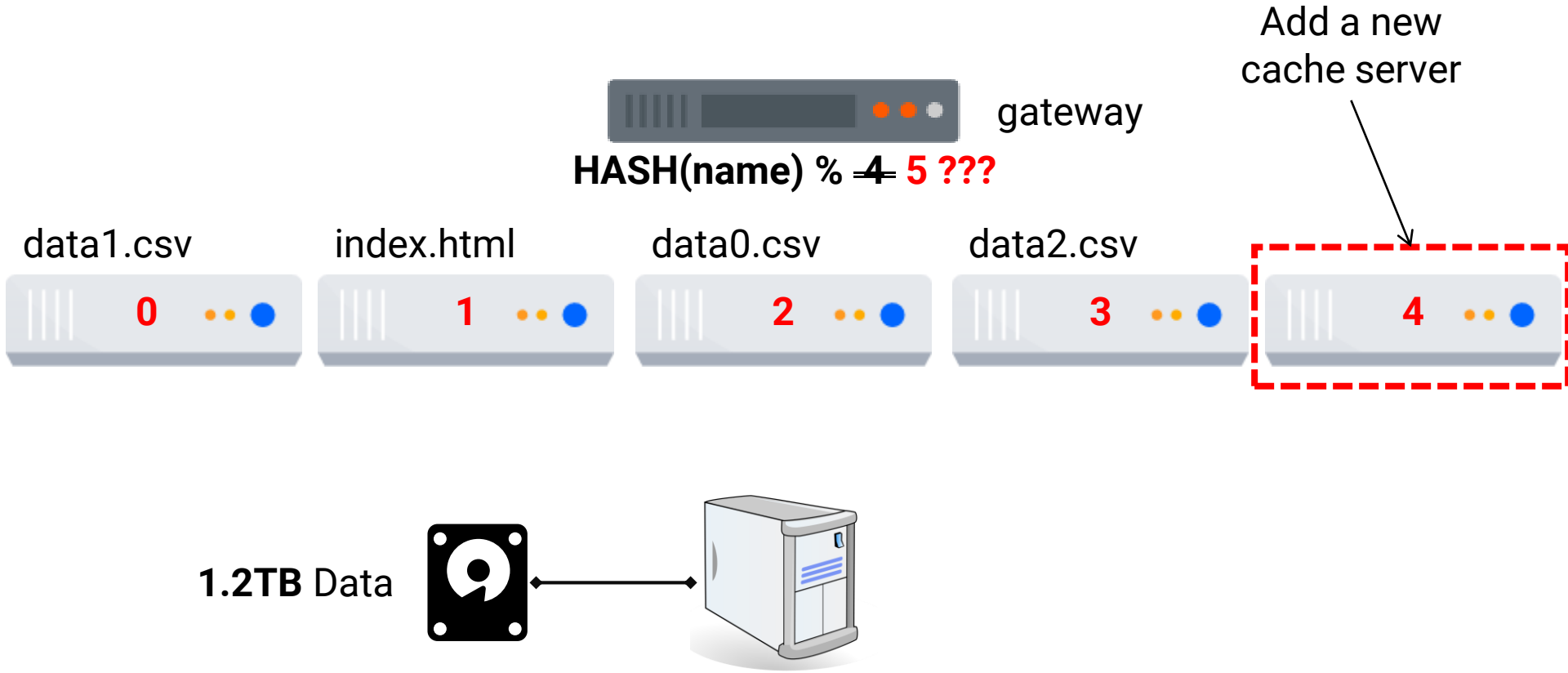


A Simple Solution: Hashing

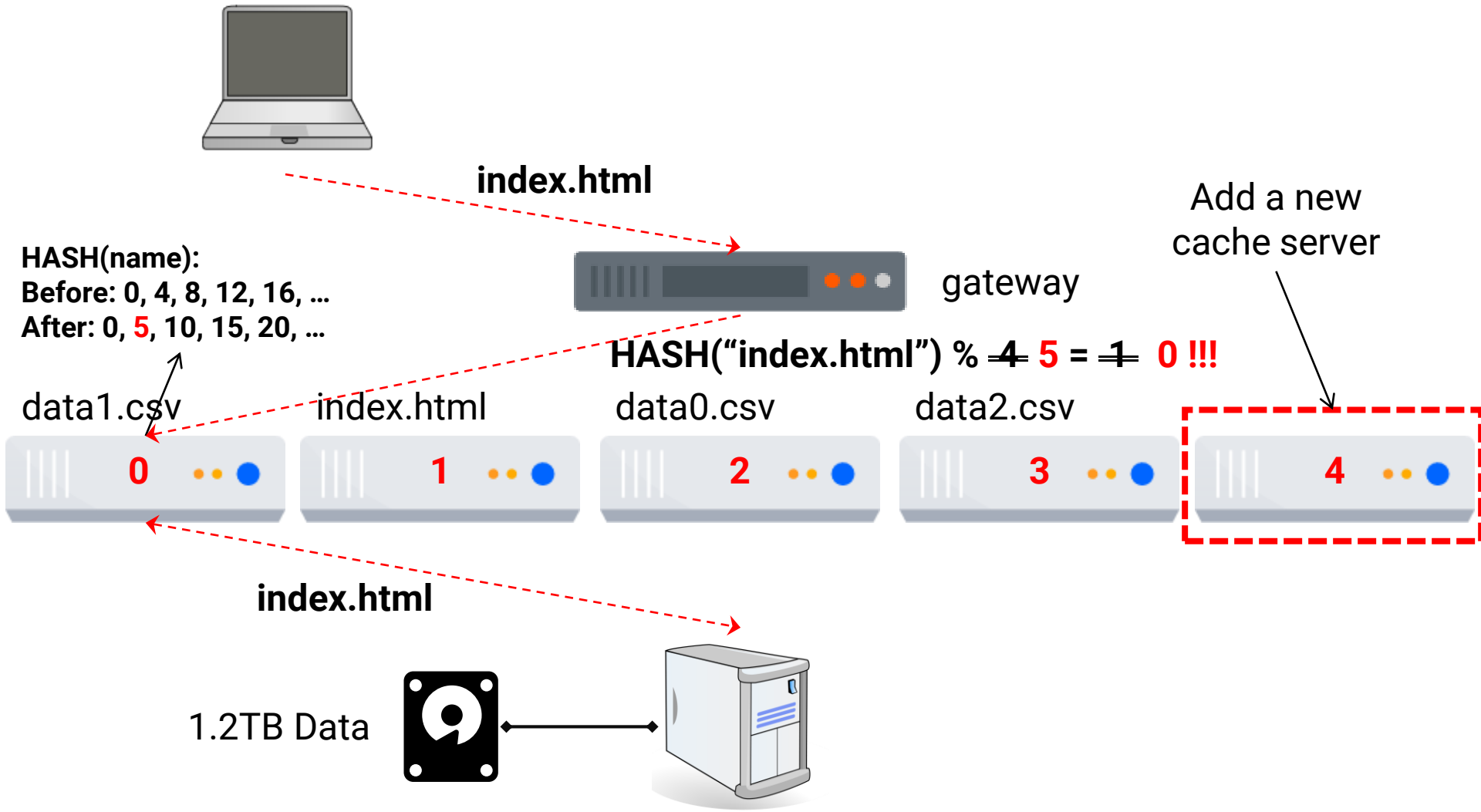




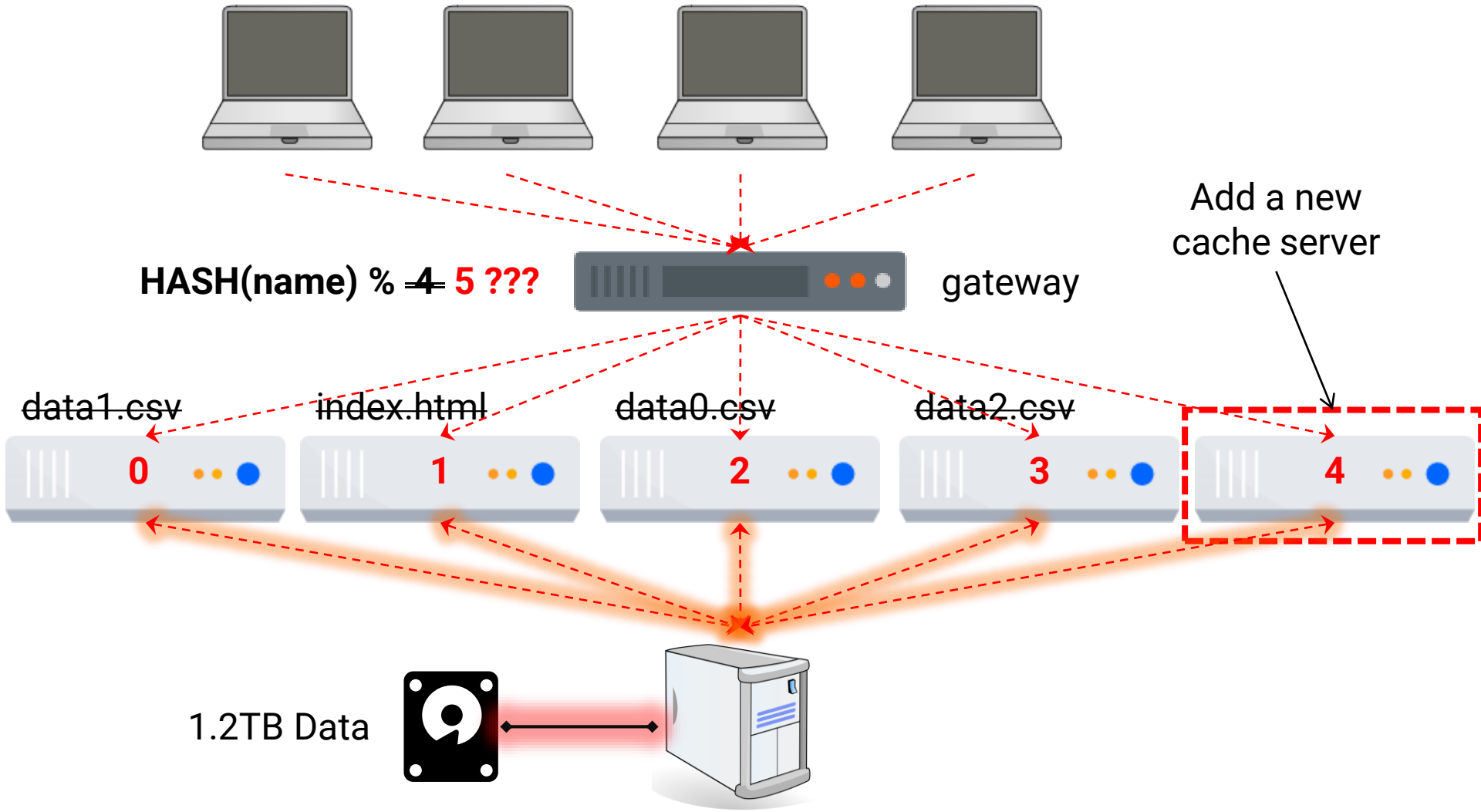
Simple Hashing in Dynamic Environment?



Problem #1: Almost All Objects Are Relocated



Problem #2: Original Server Can Be Overloaded



Background: Summary

- Hashing is awesome
  - simple, fast, and space-efficient
  - behaves like a totally random function
    - mapping objects almost uniformly to cache servers
- However, simple hashing works badly in dynamic environment
  - almost all objects are relocated
  - original server can be overloaded
- Is there an algorithm that:
  - keeps the advantages of simple hashing, and
  - works well in dynamic environment

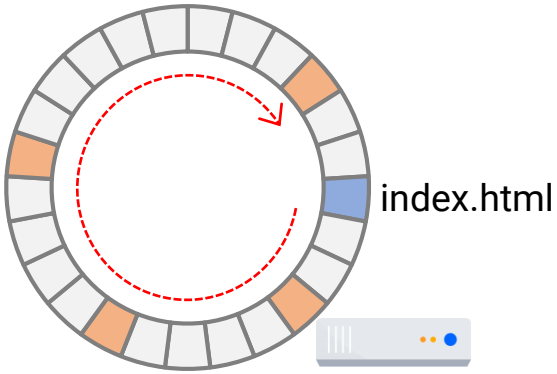
The Need for the *Consistent* Property

Almost all objects **stay assigned to the same cache** even as the number of caches changes.

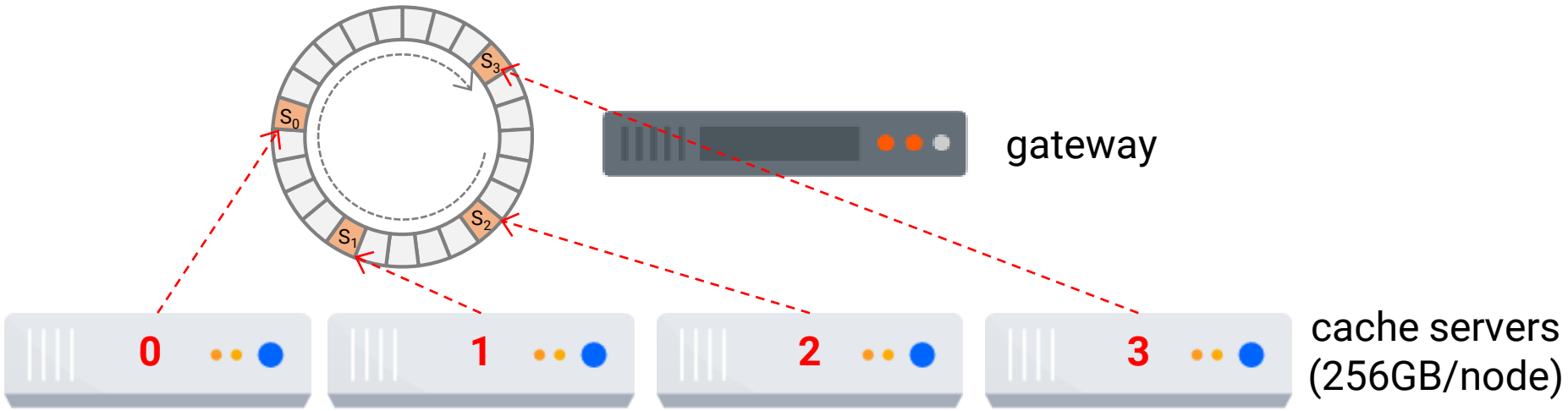
Consistent Hashing

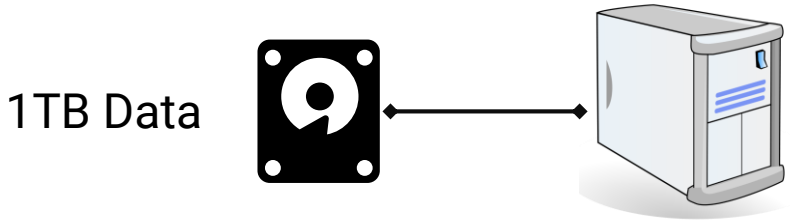
- D Karger, et al. **Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web**. STOC 1997

- Basic idea:
  - Hashing ***both*** **servers and objects** to the same **ring**, and
  - Exploiting the relative position of servers to achieve consistency

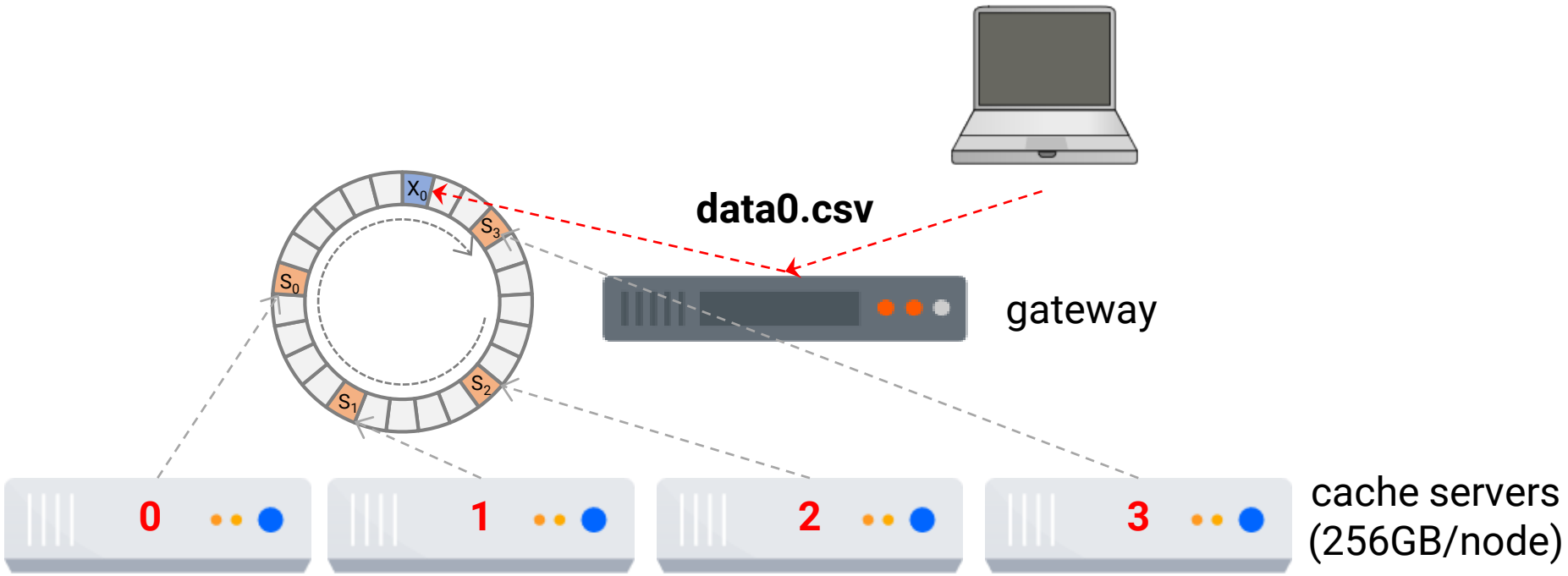


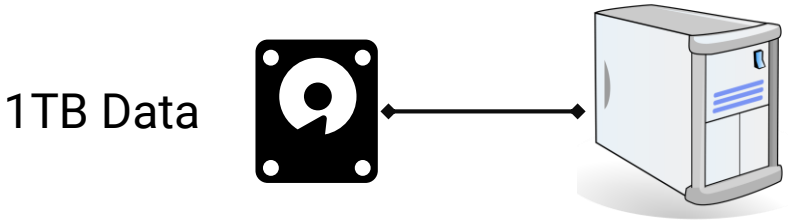
Q: How to Find the Right Cache Server?



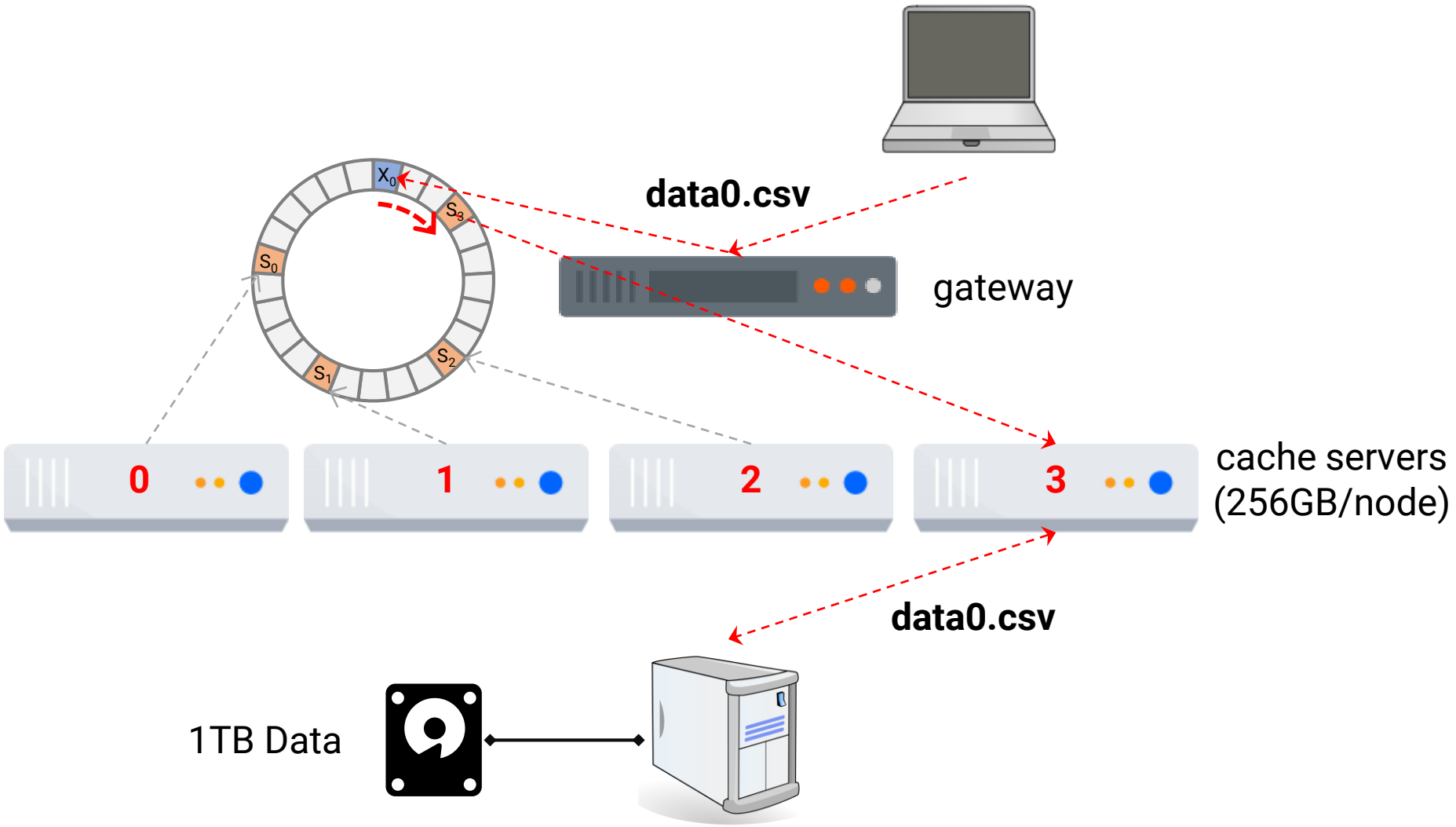


Q: How to Find the Right Cache Server?

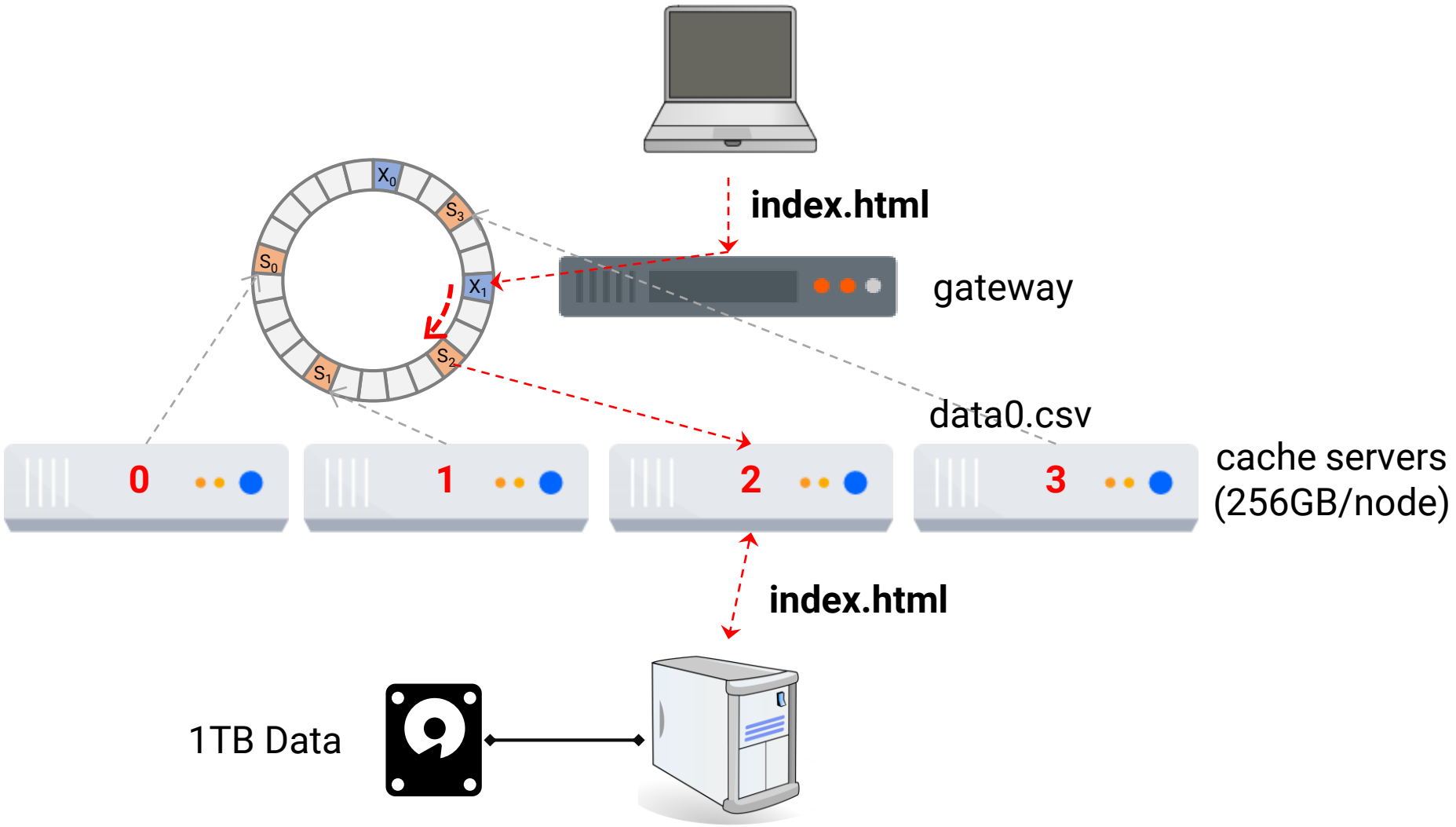




Q: How to Find the Right Cache Server?



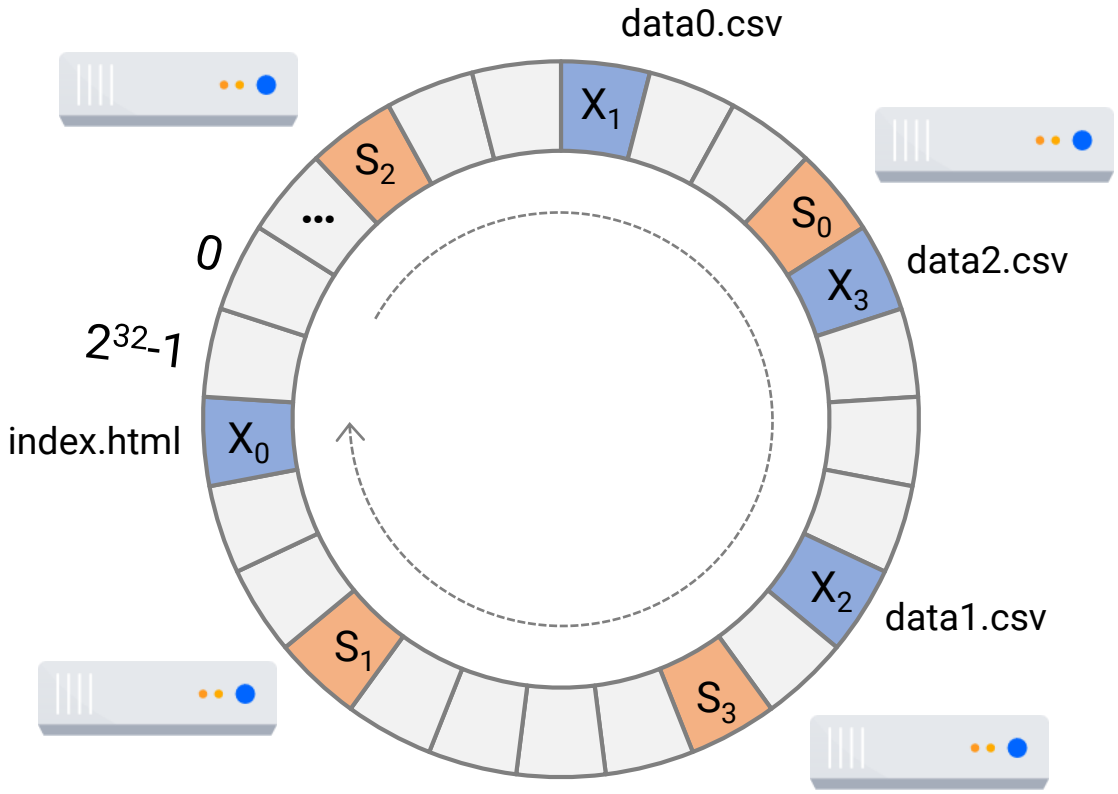
Q: How to Find the Right Cache Server?



Why is This Algorithm *Consistent*?

The Need for the *Consistent* Property

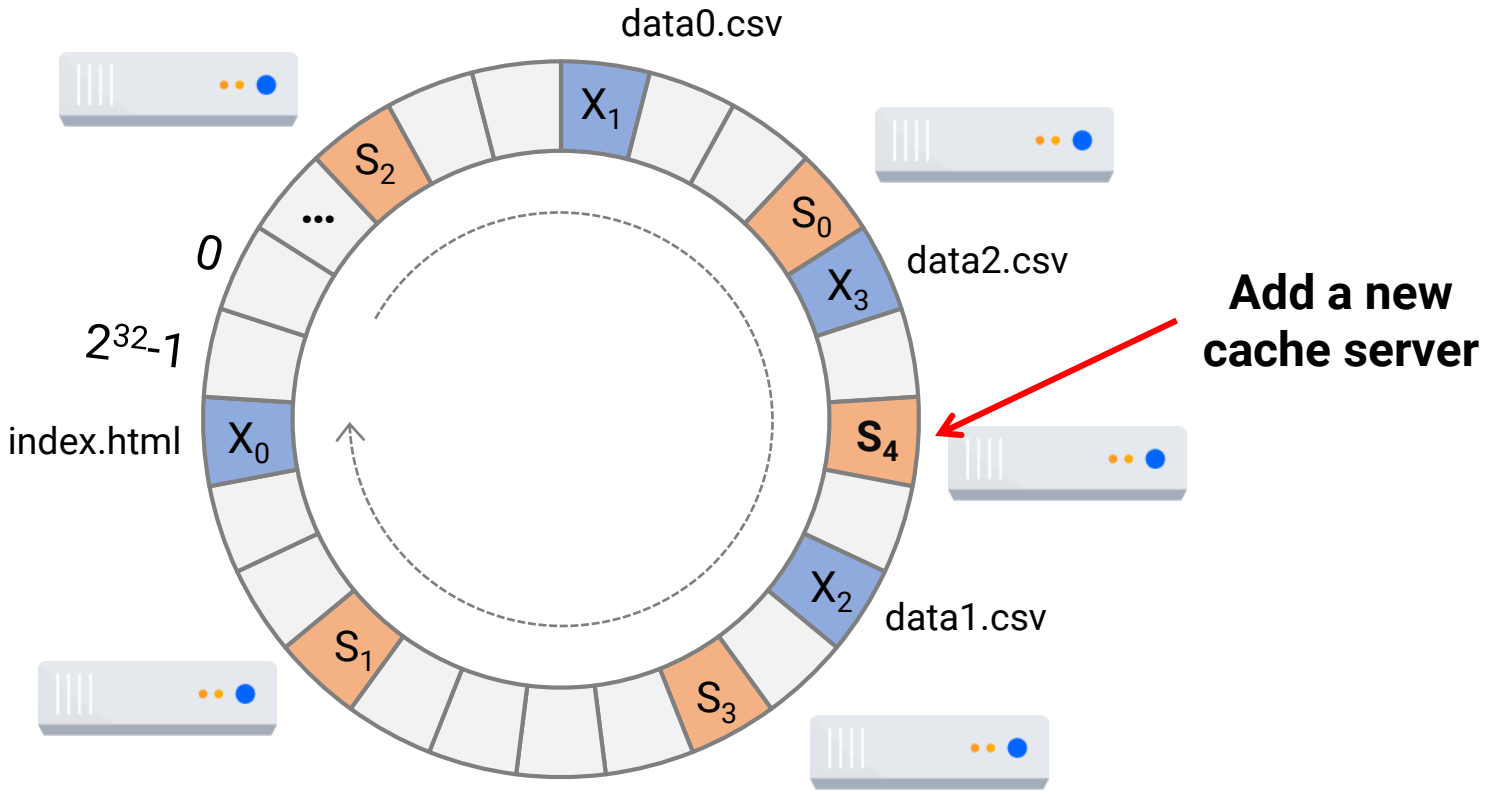
Almost all objects **stay assigned to the same cache** even as the number of caches changes.



Why is This Algorithm *Consistent*?

The Need for the *Consistent* Property

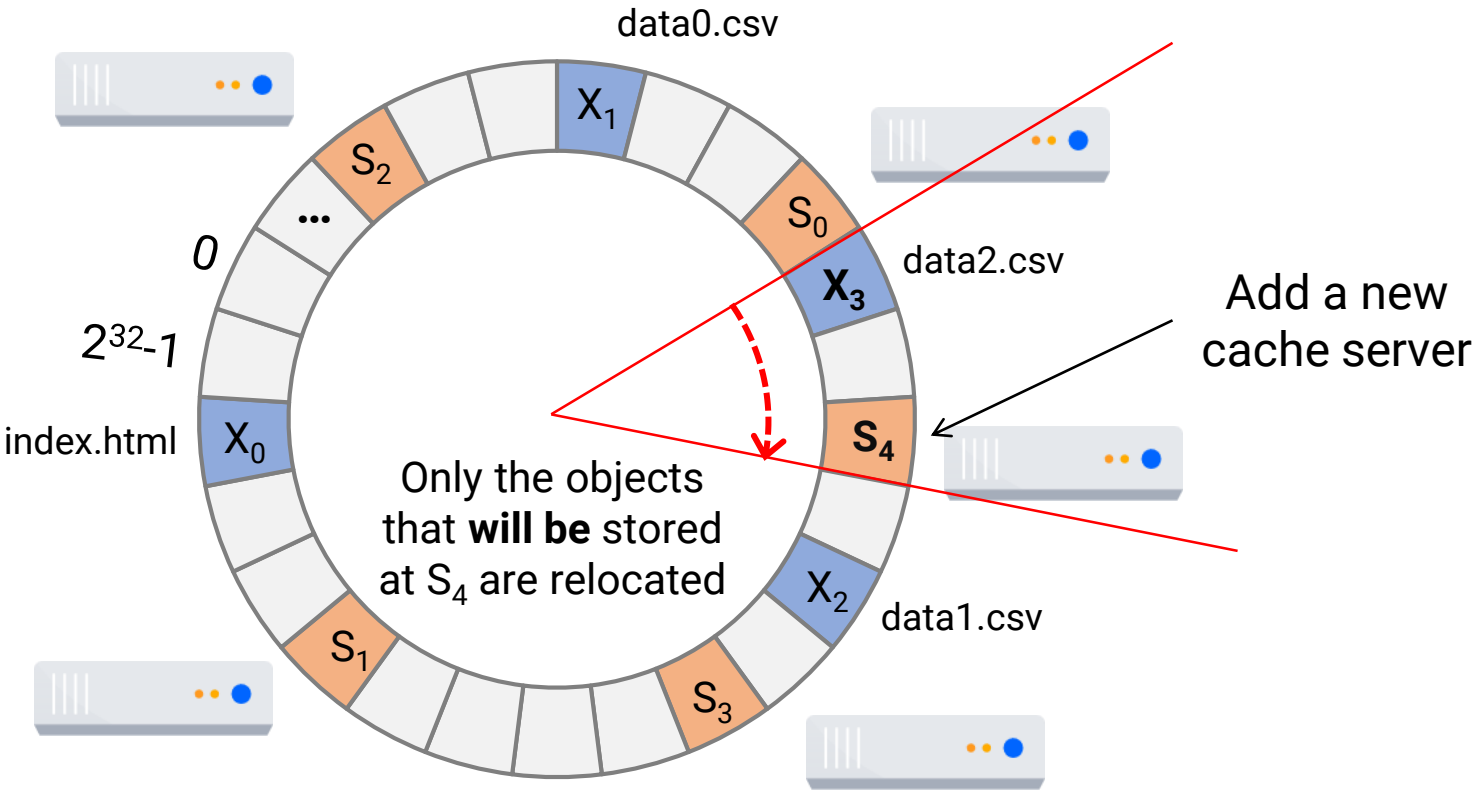
Almost all objects **stay assigned to the same cache** even as the number of caches changes.



Why is This Algorithm *Consistent*?

The Need for the *Consistent* Property

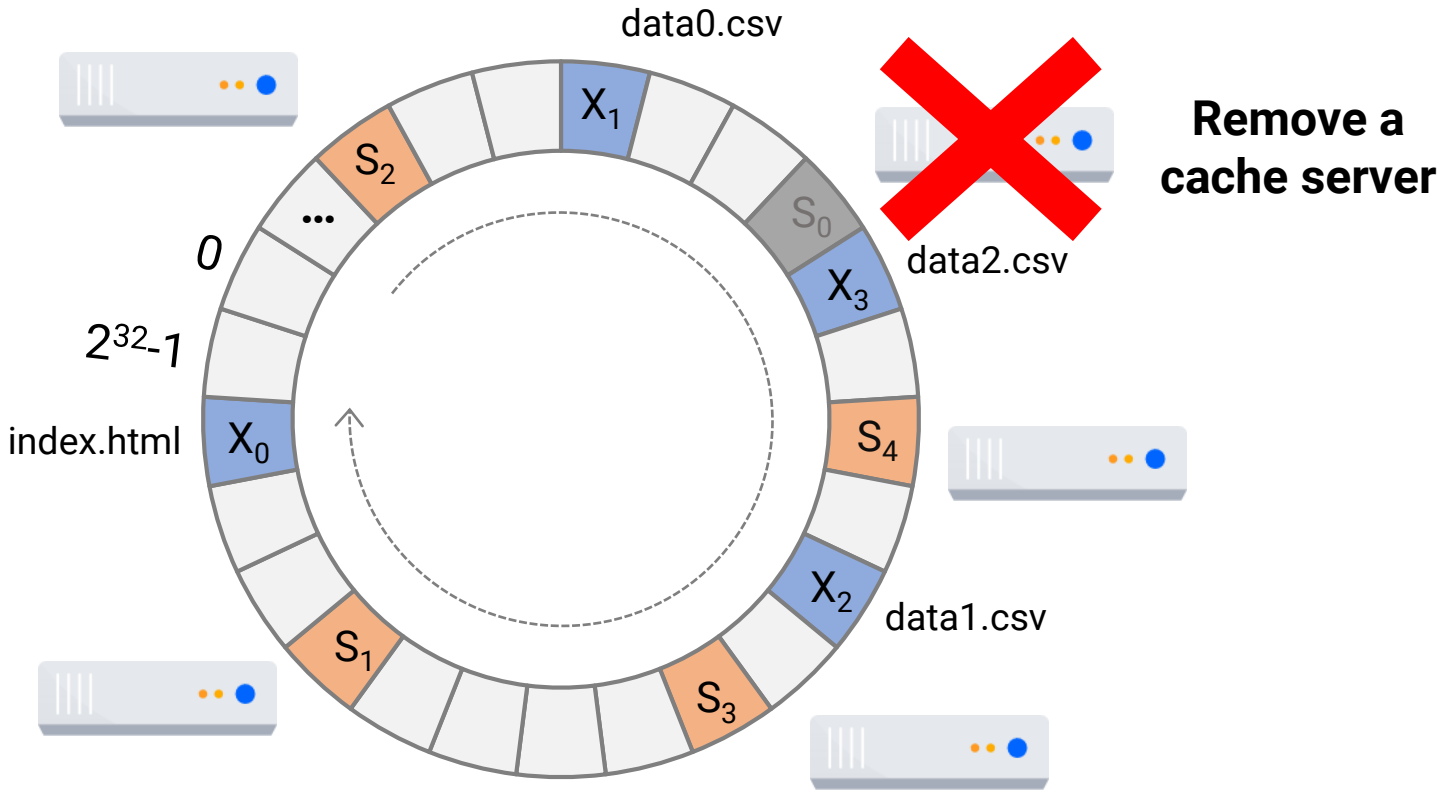
Almost all objects **stay assigned to the same cache** even as the number of caches changes.



Why is This Algorithm *Consistent*?

The Need for the *Consistent* Property

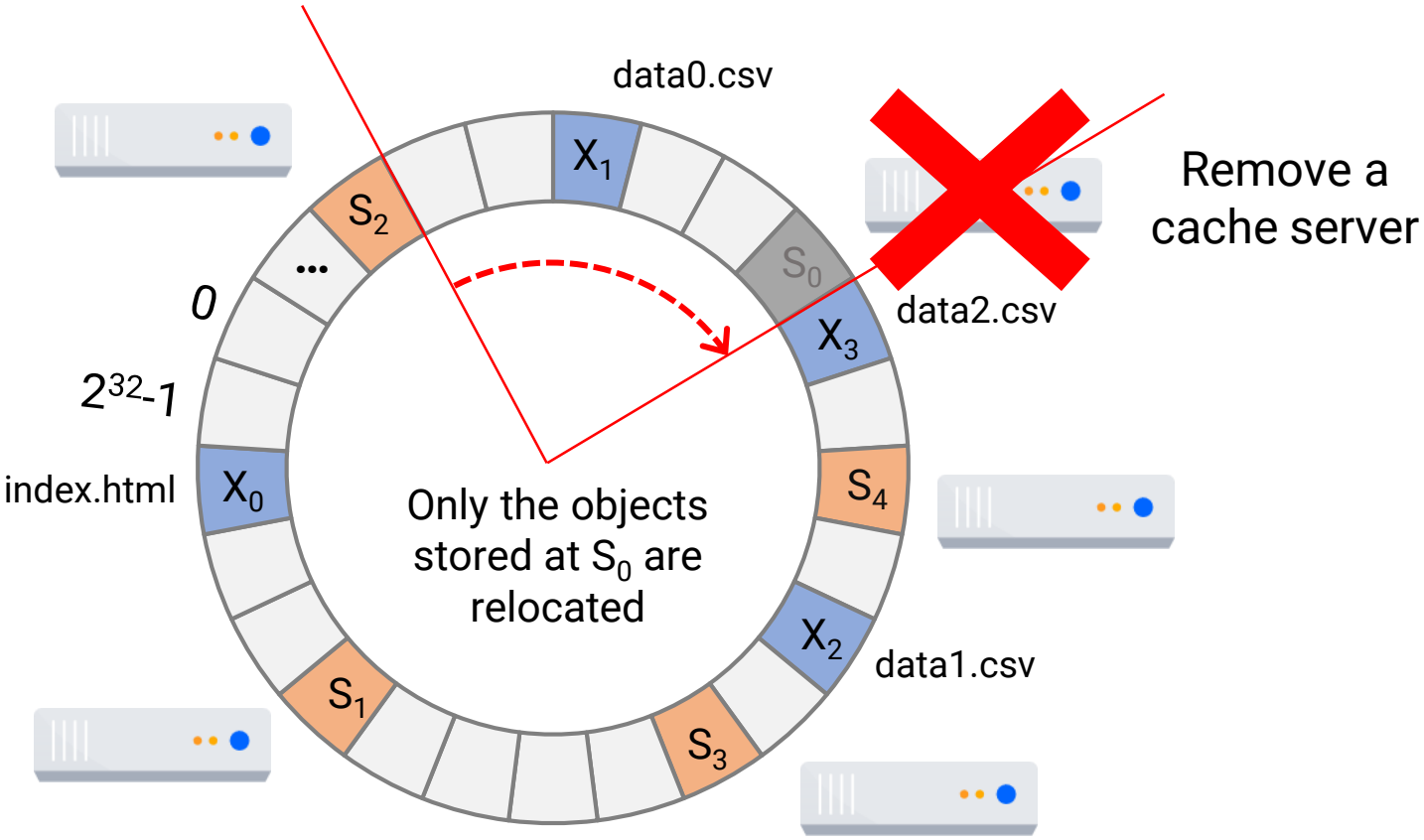
Almost all objects **stay assigned to the same cache** even as the number of caches changes.



Why is This Algorithm *Consistent*?

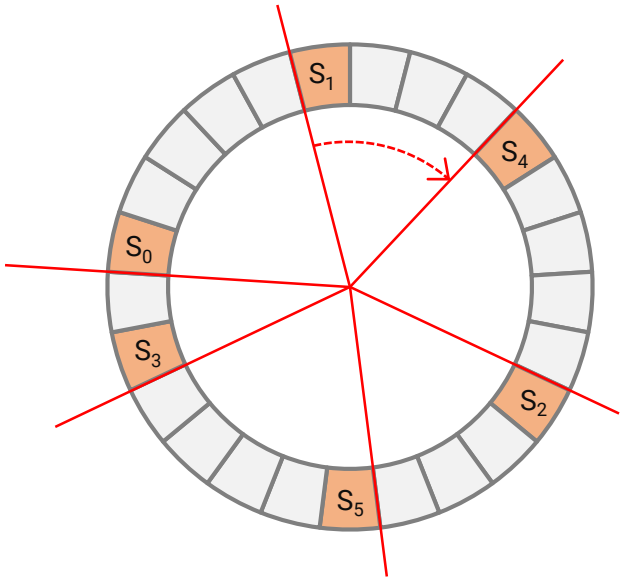
The Need for the *Consistent* Property

Almost all objects **stay assigned to the same cache** even as the number of caches changes.



Consistent Hashing: Summary

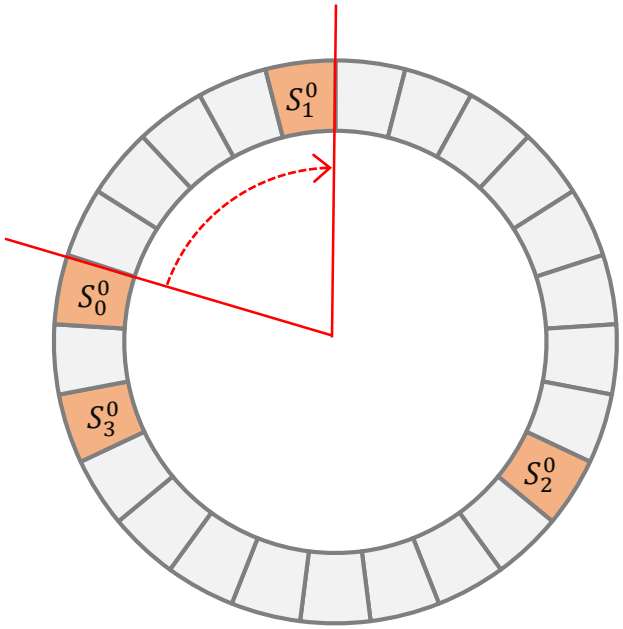
- Expected fraction of objects stored at each of n servers is $\frac{1}{n}$
- Adding/removing a server causes only a $\frac{1}{n}$ fraction of the objects to relocate
  - Simple hashing: on average only a $\frac{1}{n}$ fraction of the objects don't move

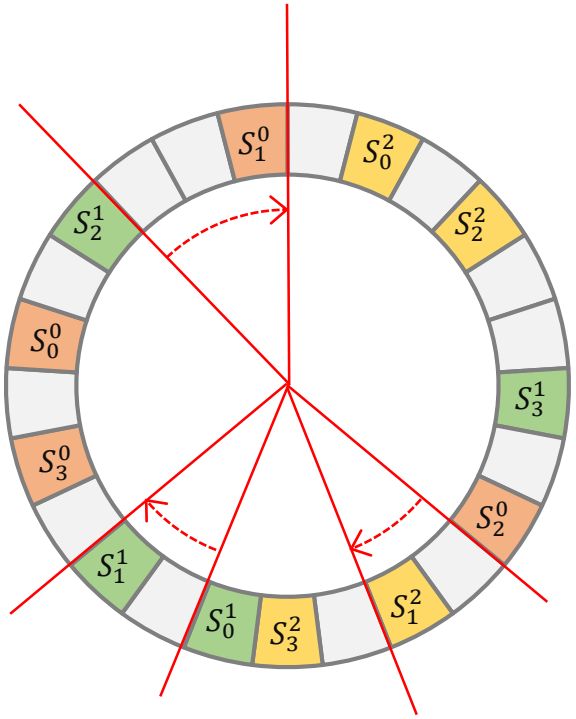


Implementation Details

- `ClockwiseSuccessor`: using an ordered dictionary, e.g., a red-black tree

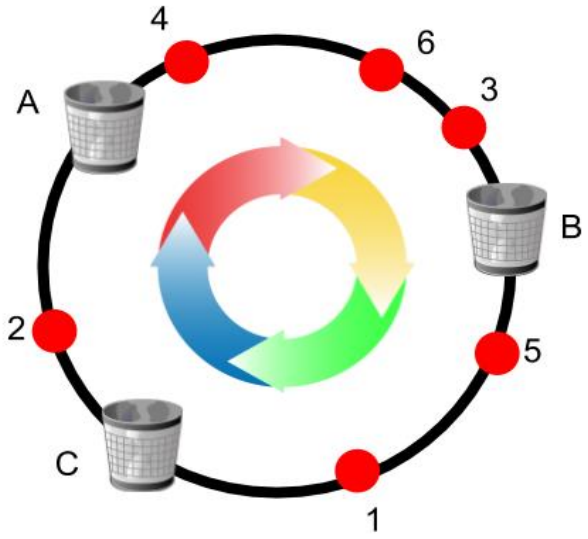
- Use ***virtual copies*** to reduce the ***variance***





Google's Improved Consistent Hashing

- V Mirrokni, et al. **Consistent Hashing with Bounded Loads**. SODA 2018
  - Let servers(bins) have ***capacities***, i.e., the maximum load they can handle
  - Hash the request to the ring, same as Consistent Hashing
  - Walk the ring clockwise until we find the first **non-full** server



Q&A