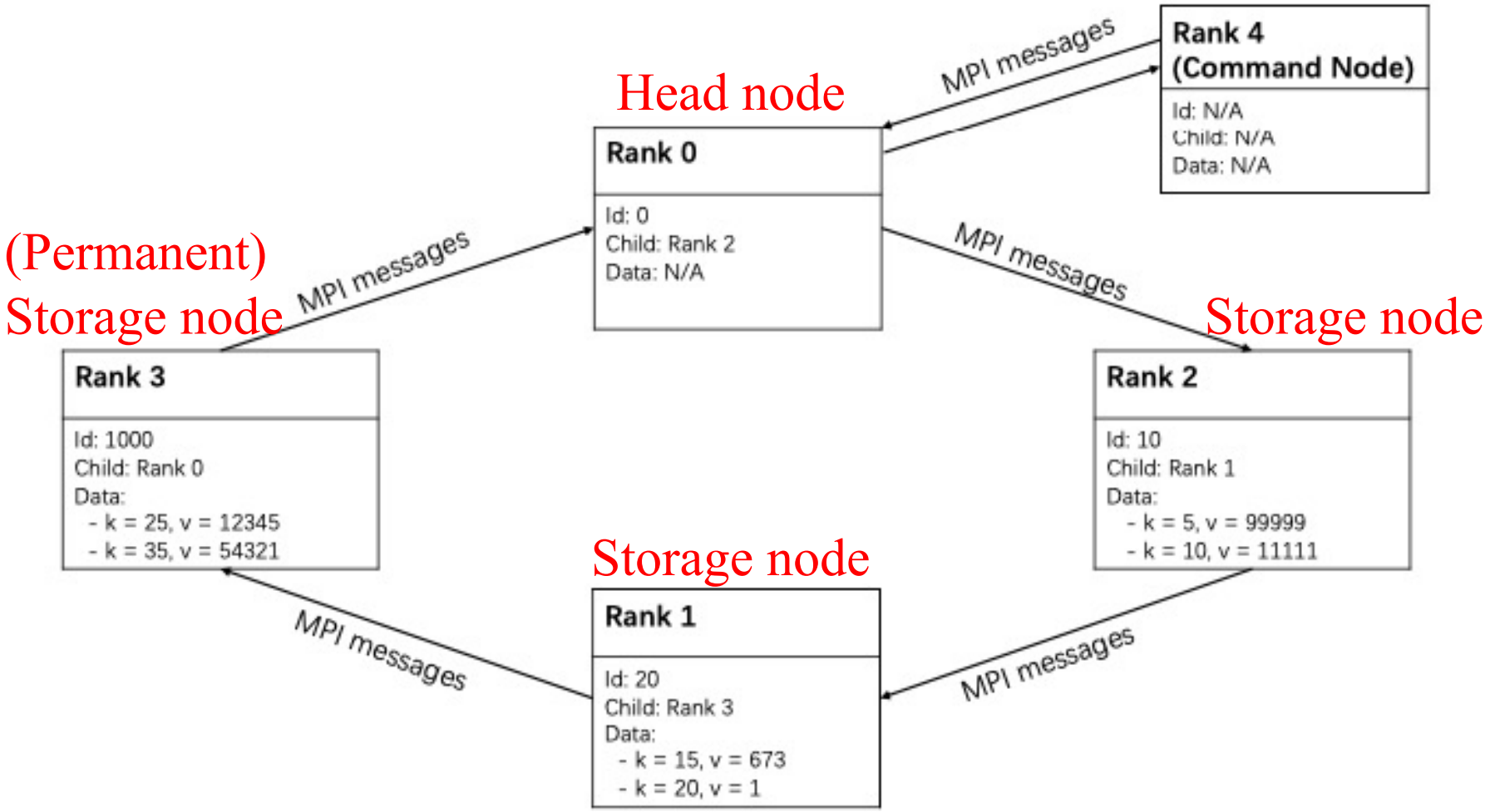
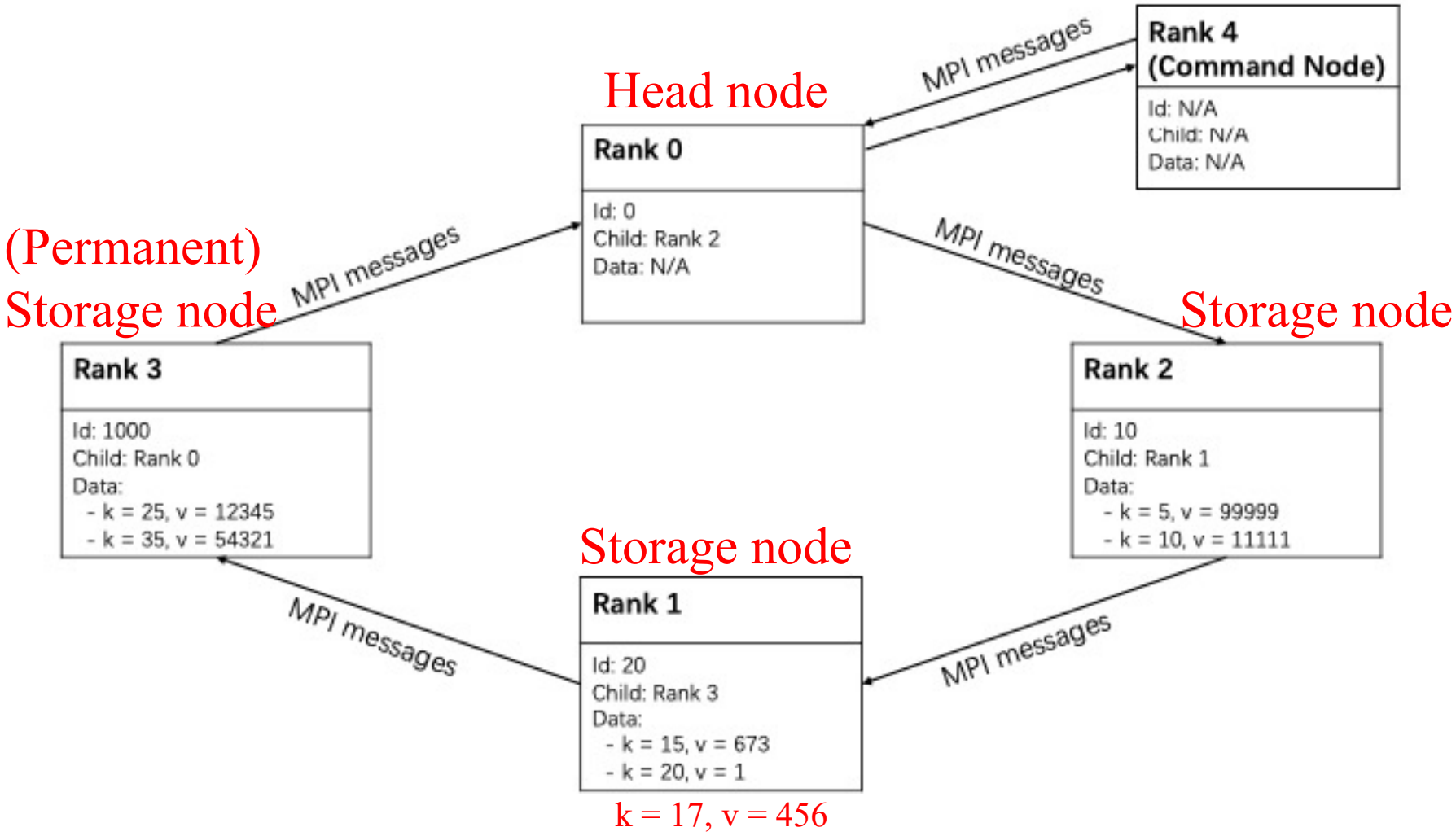
Distributed Hash Table Structure



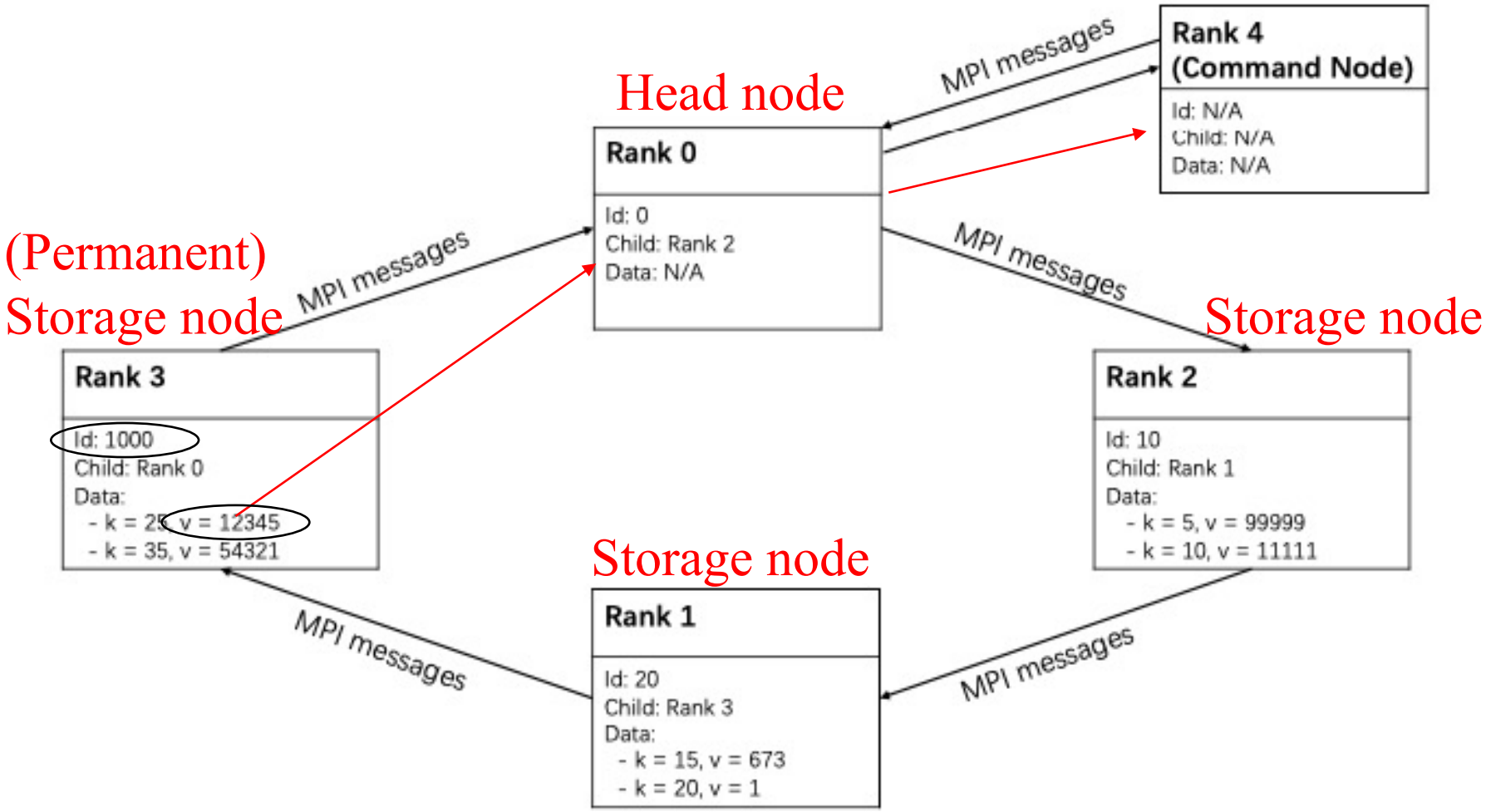
Distributed Hash Table Structure

Command: PUT 17 456

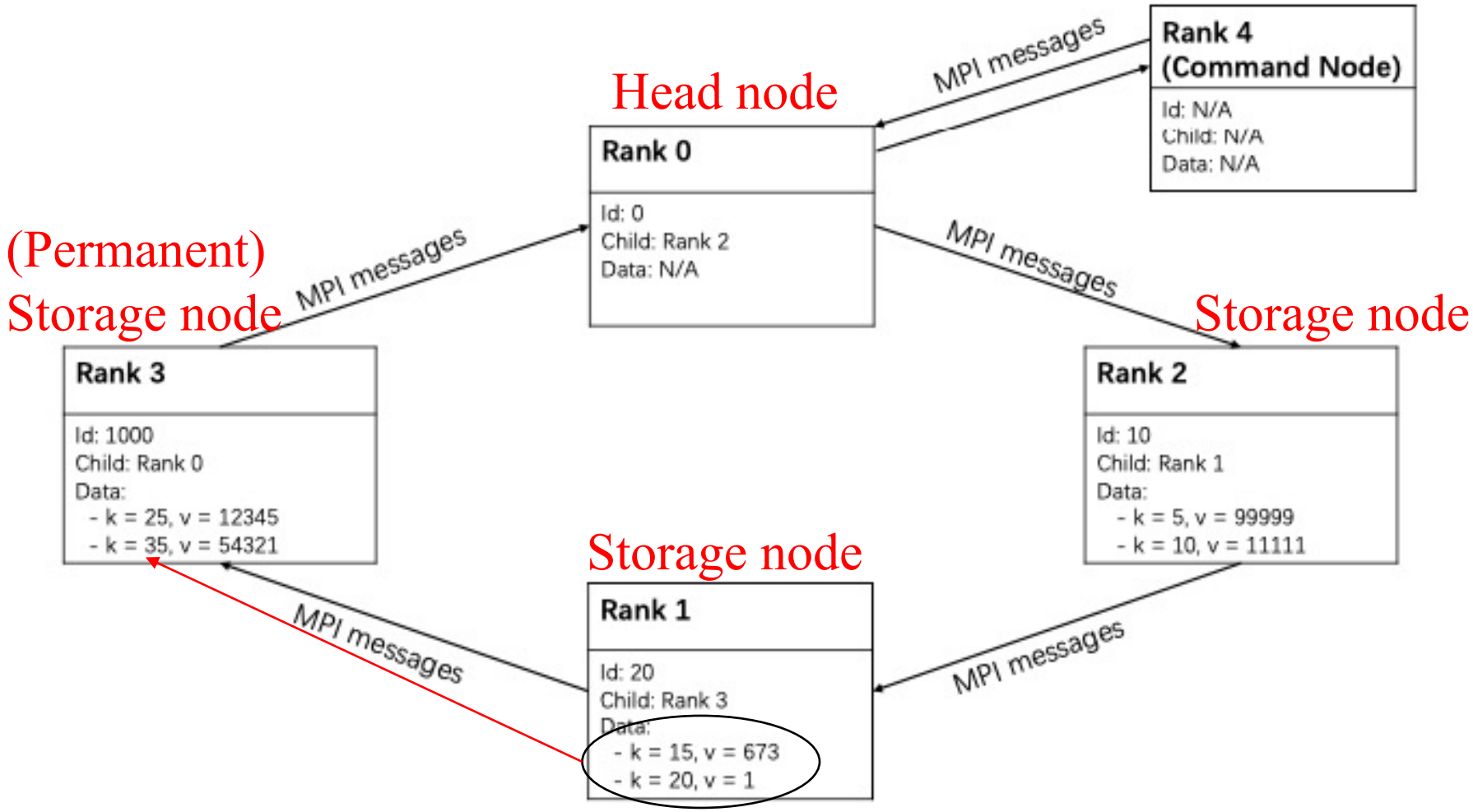


Distributed Hash Table Structure

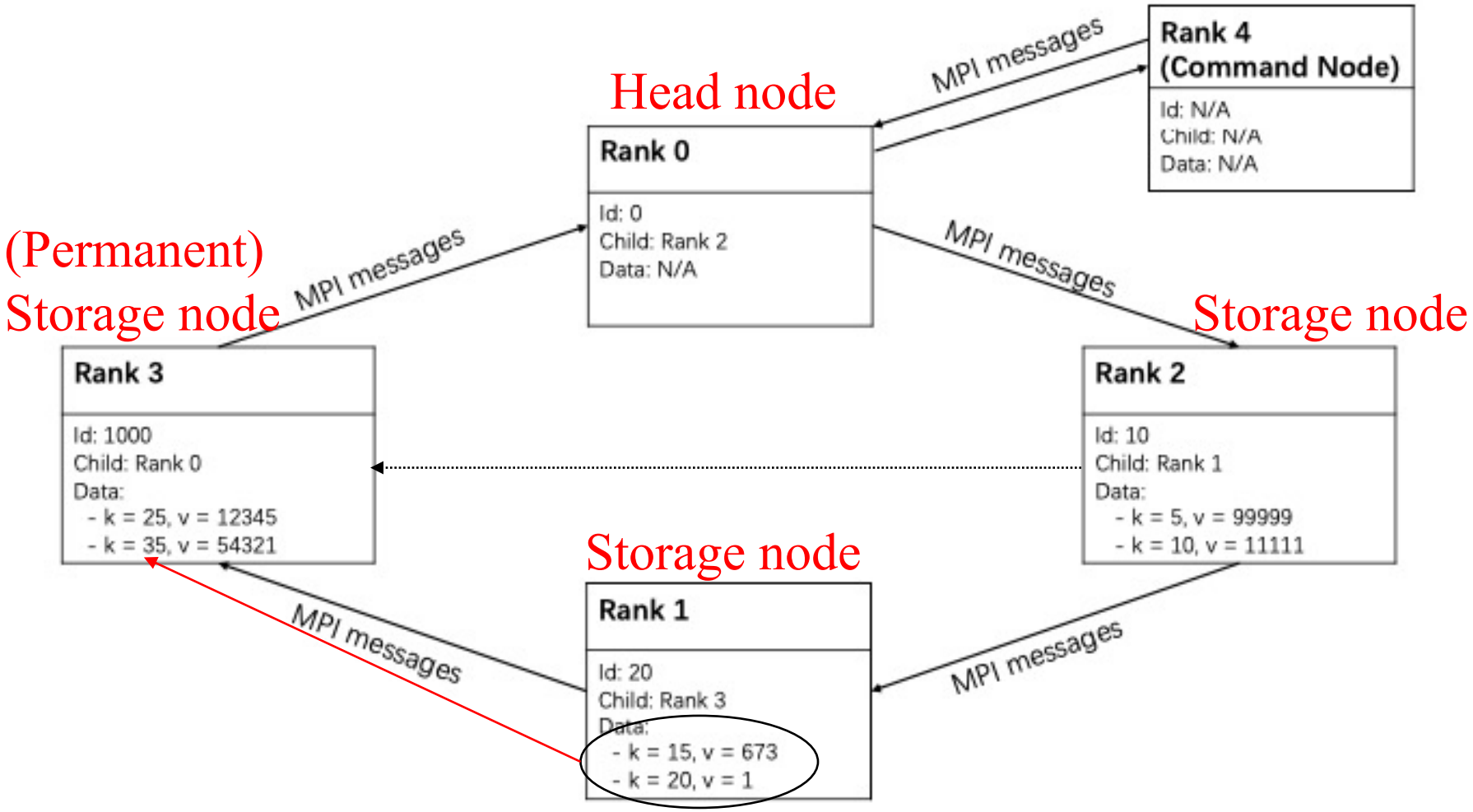
Command: GET 25



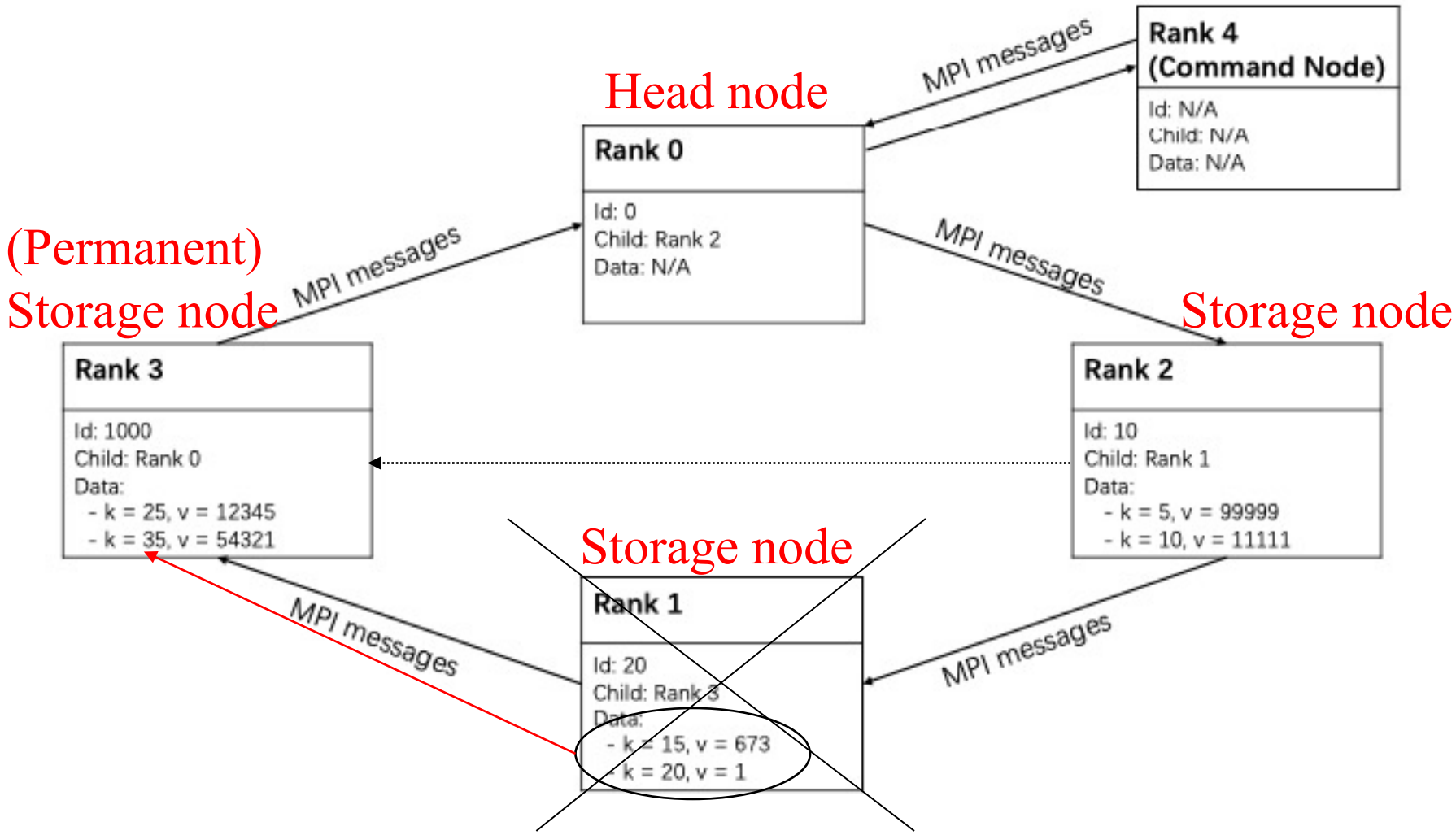
Distributed Hash Table Structure



Distributed Hash Table Structure



Distributed Hash Table Structure



DHT Program Notes

- When does an MPI_Send match an MPI_Recv?
  - (1) source of receive matches the sender
  - (2) the tags match
- For this program, we will be determining the operation (e.g., put, get, add, remove, end) via the tag
  - So, how does a receiver know which tag to use on a receive?
- Also, a receiver may not know the sender

ANY_SOURCE and ANY_TAG

- If the receiver does not know the sender, the source can be ANY_SOURCE
  - This matches *any* sender
- If the receiver does not know what tag the sender is using but wants it to match, ANY_TAG can be used
  - This matches *any* sender tag

ANY_SOURCE and ANY_TAG

- Very useful for client/server applications
  - Server does not know who message is coming from as well as what operation that message is requesting
  - One way to handle this (in general) is to do an ANY_TAG receive and make sure that each message is of identical size
  - With the DHT assignment, messages coming from the command node ("client") to the head node ("server") are of varying sizes
    - Example: PUT sends key and value; GET only sends a key

ANY_SOURCE and ANY_TAG

- Instead, the idea is to "peek" at the message to find out what it is, and then do a receive later
  - This is necessary on the head node
    - Use MPI_Probe (the "peek") to figure out what the tag is
    - Then receive the proper message (you know the tag and the source now)
    - With MPI_Probe, you can also figure out the size of the message, which is valuable when carrying out an ADD or REMOVE command
  - Not actually necessary on the command node, because the command node knows exactly what it's receiving.