ECS656U/ECS796P

Distributed Systems

What this lecture is about

- Joining/Leaving in DHTs
- Key-Value store
- Memcached

(thanks to prof. Stoica)

What have we seen so far?

- P2P networks introduction

- Three basic architectures for locating and distributing content
    - Centralized directory (Napster, early BitTorrent)
    - Query flooding (Gnutella)
    - Hierarchical and non-hierarchical overlay designs (Kazaa, BT DHT)

- We finished looking at DHTs and how to locate content

This was the last slide: Chord (finger tables) analysis on lookup

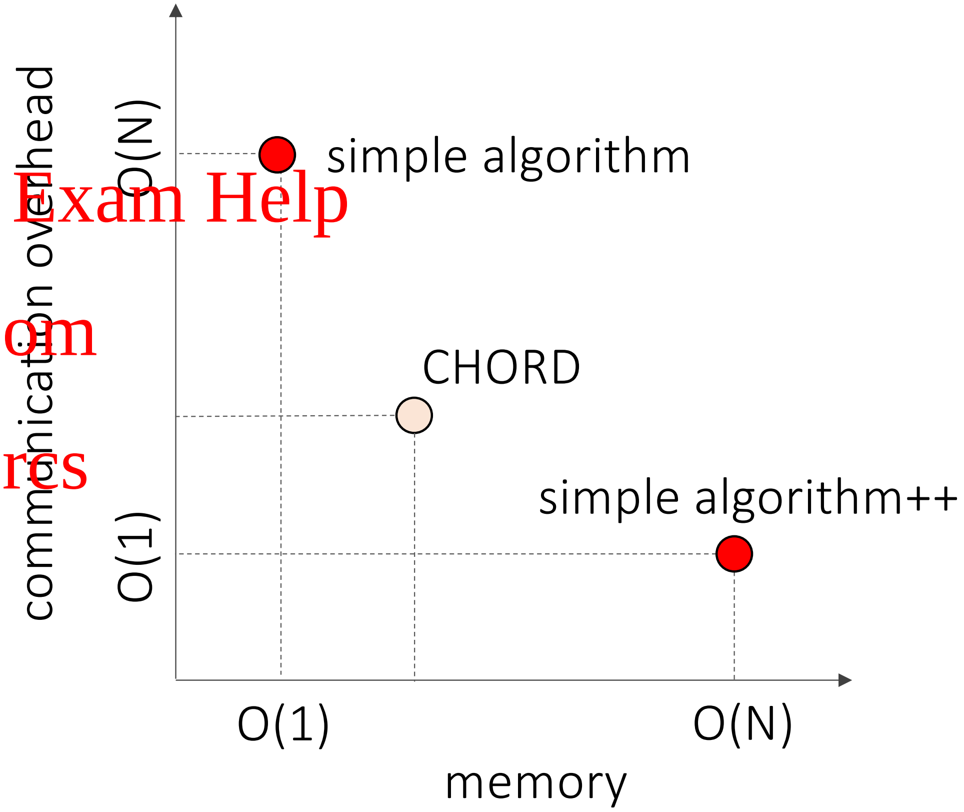
Each node stores a subset of successors:

- O(log N) memory

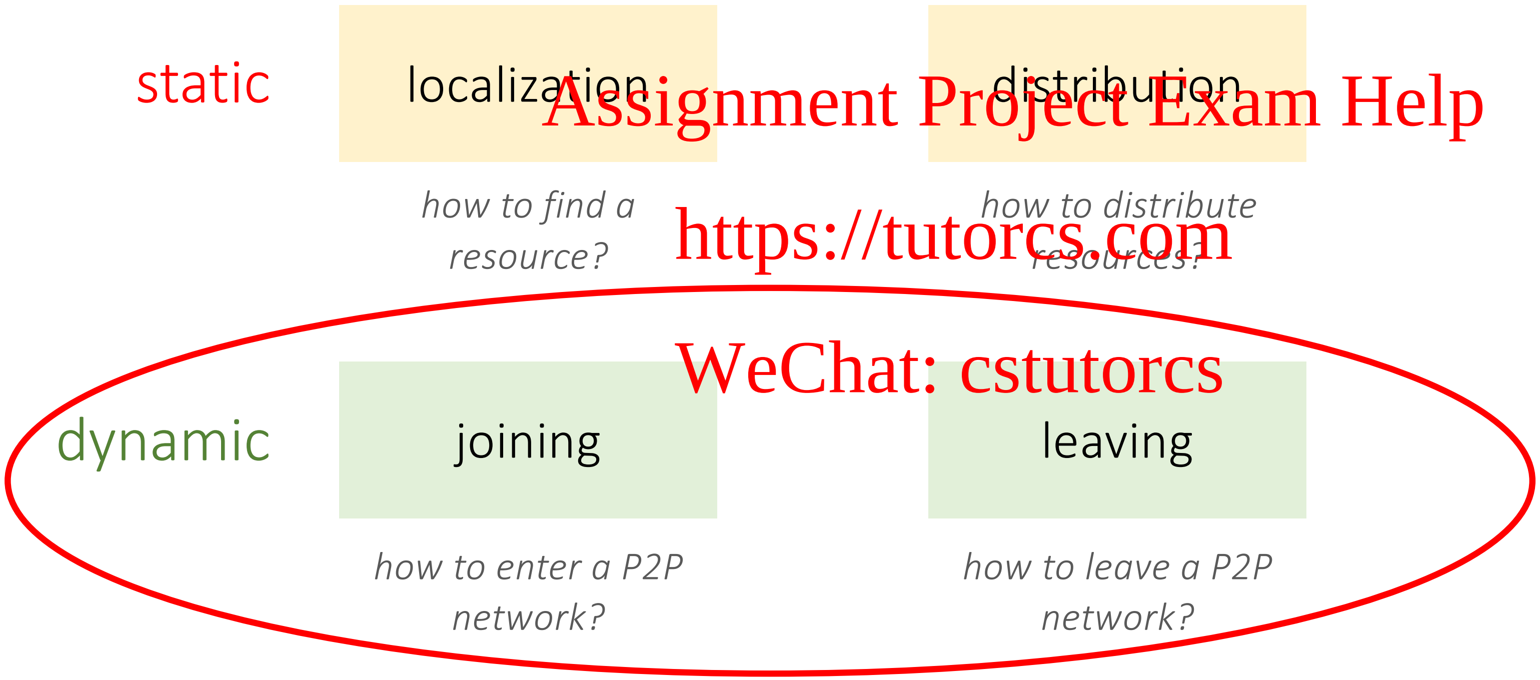
The search space is halved at each hop:

- O(log N) communication

More robust: unless the authority peer of the key ID fails, lookup operations work correctly



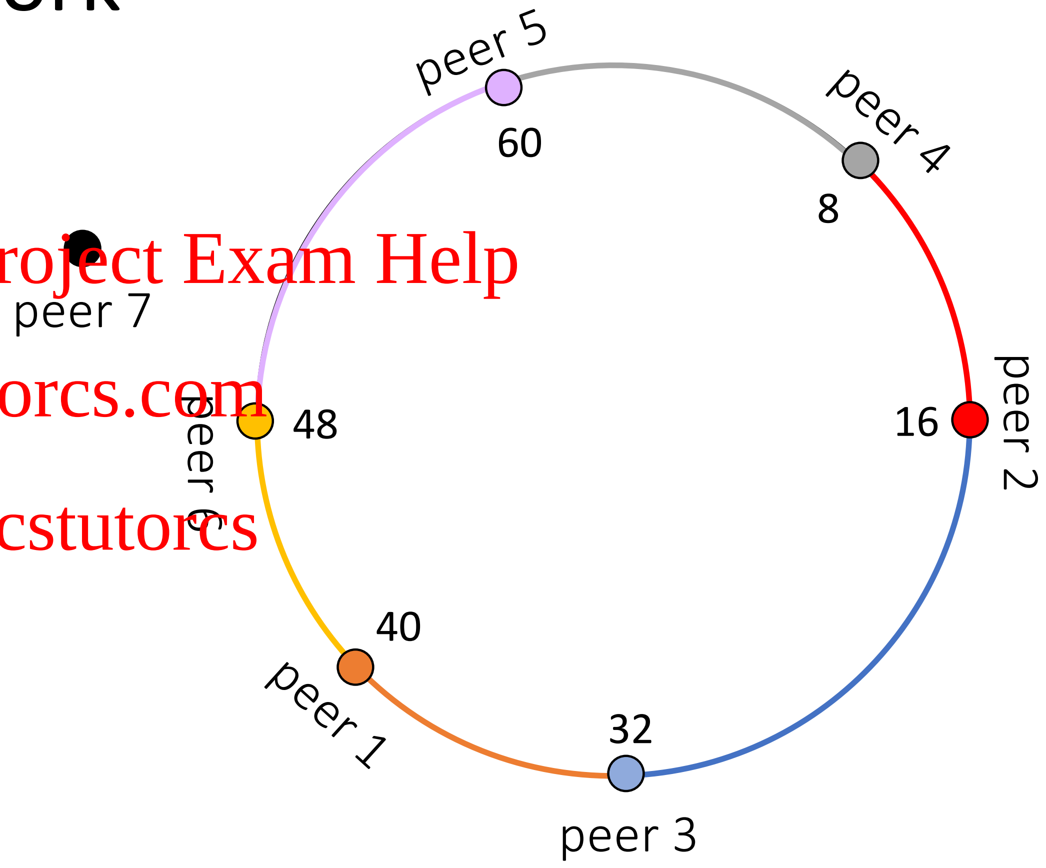
The fundamentals of P2P: the Chord example



static	localization *how to find a resource?*	distribution *how to distribute resources?*
dynamic	joining *how to enter a P2P network?*	leaving *how to leave a P2P network?*

Chord: joining the network

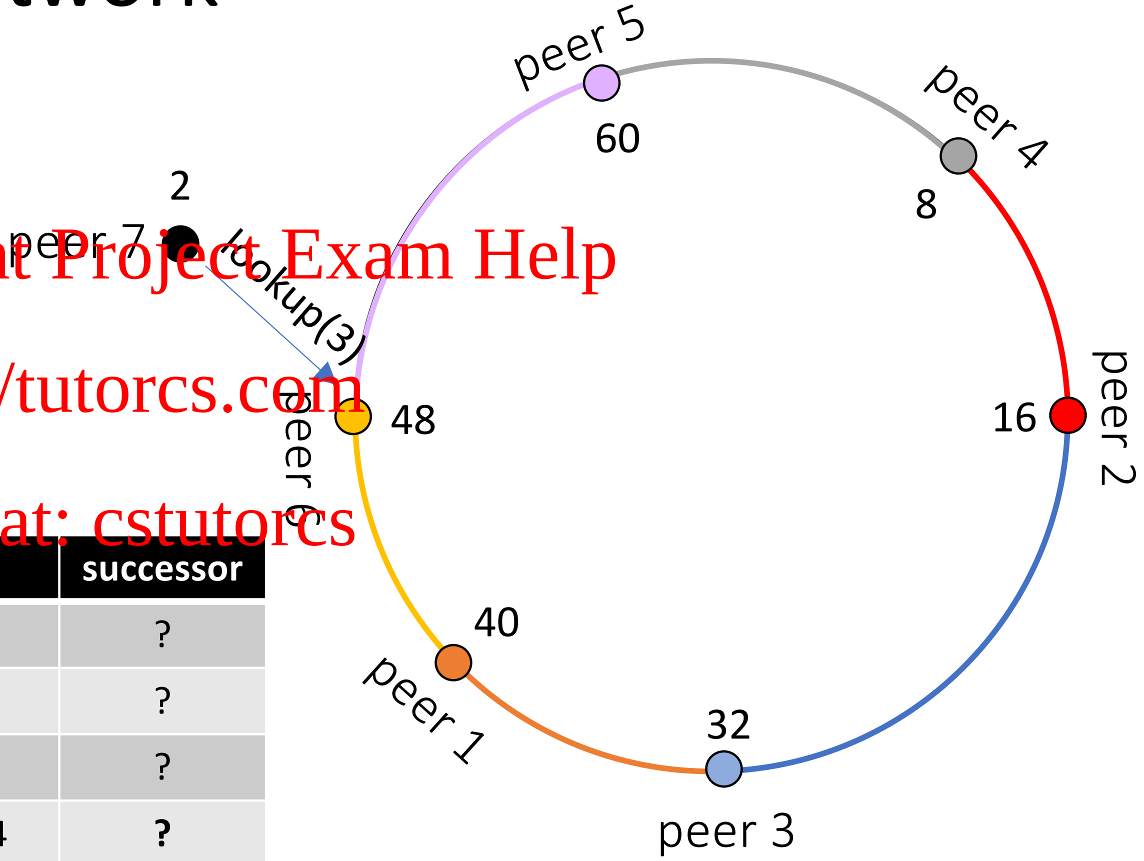
A peer that wants to joint the DHT:



Chord: joining the network

A peer that wants to joint the DHT:

- computes its own *id*
- computes its own **finger table**

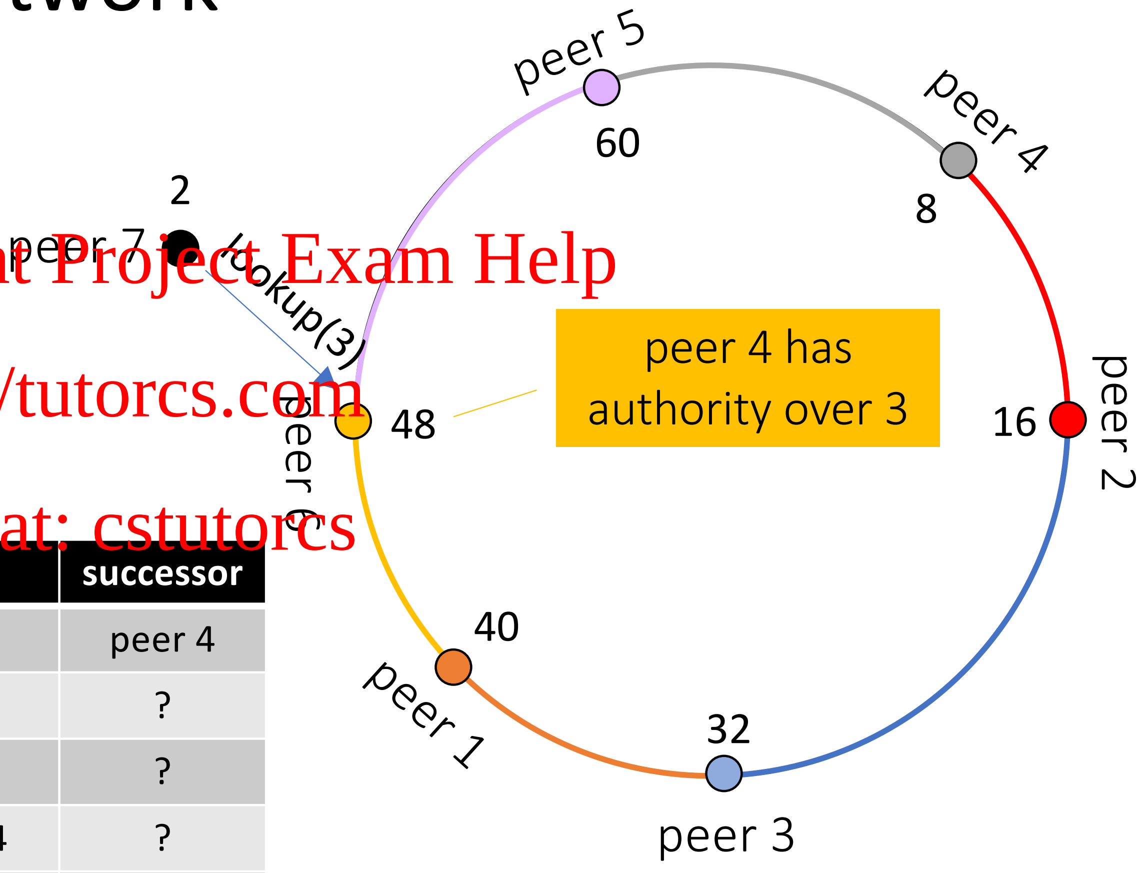


i	key id	successor
0	$2 + 2^0 \bmod 64 = 3$	?
1	$2 + 2^1 \bmod 64 = 4$	?
2	$2 + 2^2 \bmod 64 = 6$	?
3	$2 + 2^3 \bmod 64 = 14$	**?**
4	$2 + 2^4 \bmod 64 = 30$	?
5	$2 + 2^5 \bmod 64 = 46$	?

Chord: joining the network

A peer that wants to joint the DHT:

- computes its own *id*
- computes its own **finger table**



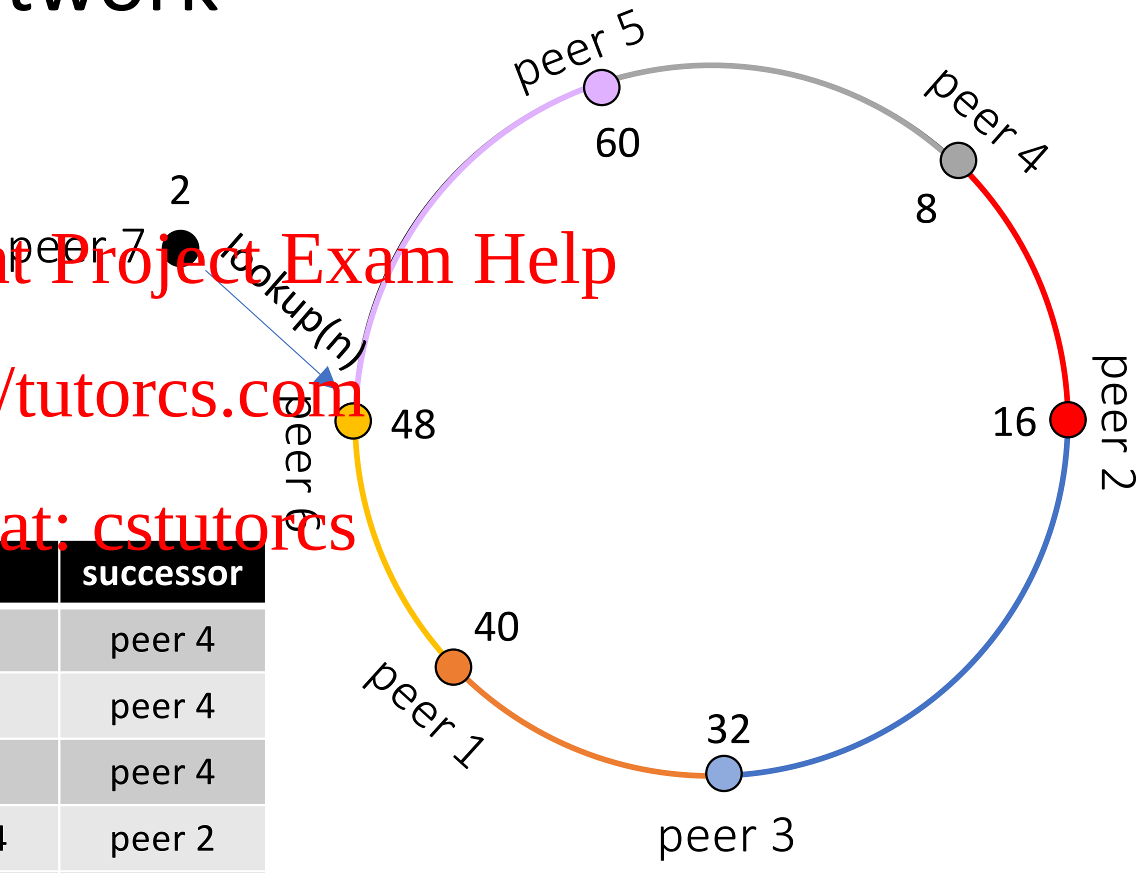
peer 4 has authority over 3

i	key id	successor
0	$2 + 2^0 \bmod 64 = 3$	peer 4
1	$2 + 2^1 \bmod 64 = 4$	?
2	$2 + 2^2 \bmod 64 = 6$	?
3	$2 + 2^3 \bmod 64 = 14$	?
4	$2 + 2^4 \bmod 64 = 30$	?
5	$2 + 2^5 \bmod 64 = 46$	?

Chord: joining the network

A peer that wants to joint the DHT:

- computes its own *id*
- computes its own **finger table**

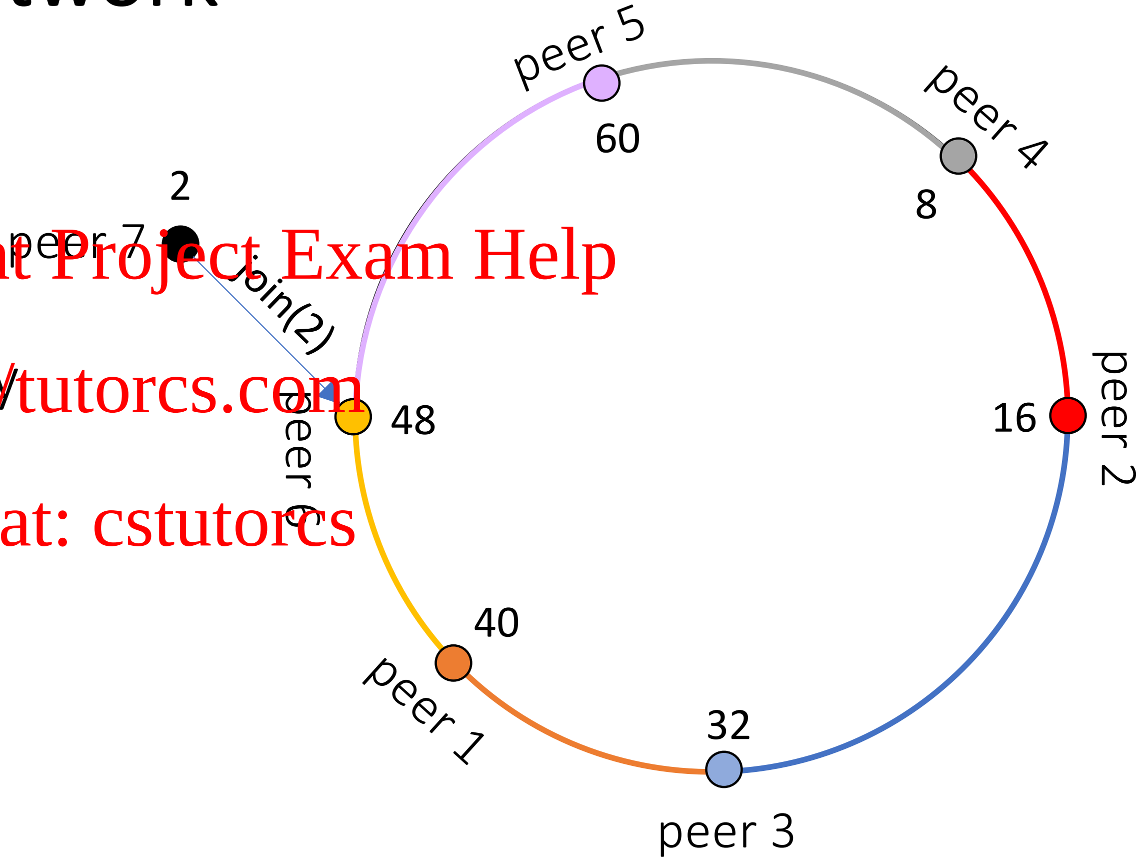


i	key id	successor
0	$2 + 2^0 \bmod 64 = 3$	peer 4
1	$2 + 2^1 \bmod 64 = 4$	peer 4
2	$2 + 2^2 \bmod 64 = 6$	peer 4
3	$2 + 2^3 \bmod 64 = 14$	peer 2
4	$2 + 2^4 \bmod 64 = 30$	peer 3
5	$2 + 2^5 \bmod 64 = 46$	peer 6

Chord: joining the network

A peer that wants to joint the DHT:

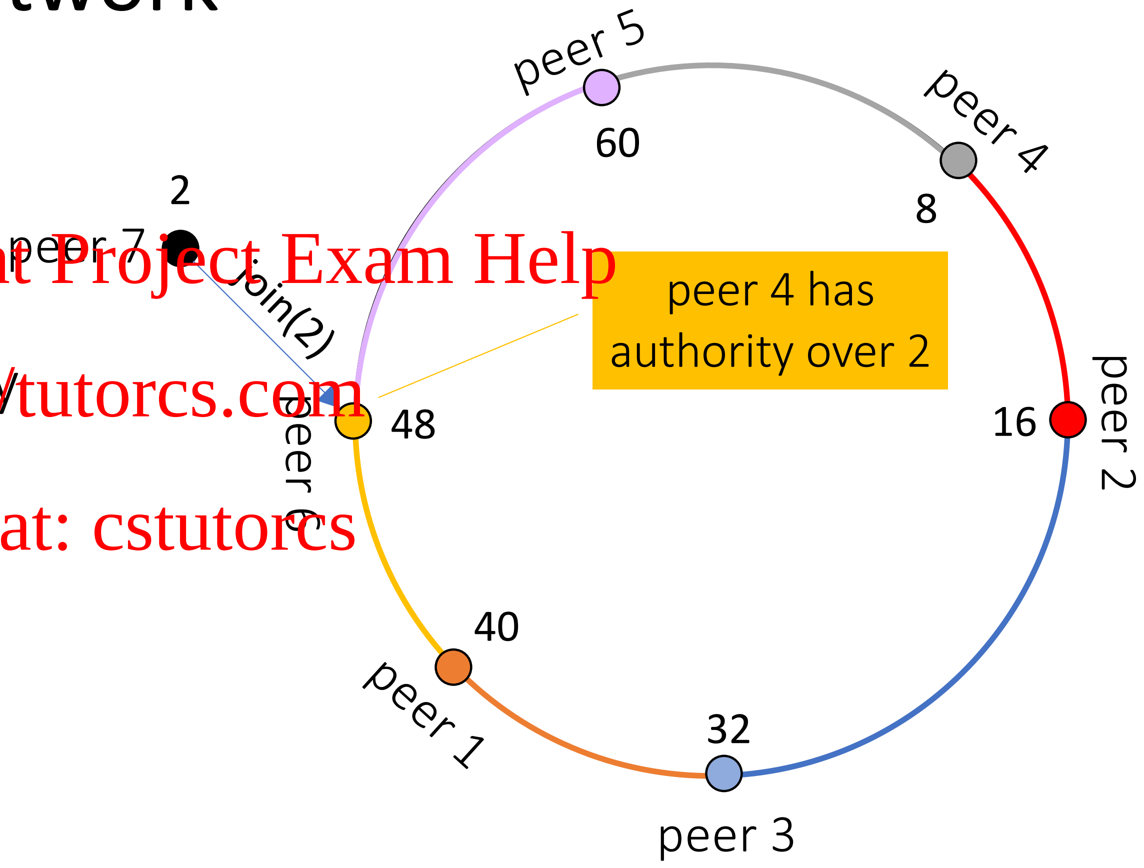
- computes its own *id*
- computes its own **finger table**
- ask any peer who has authority over *id*



Chord: joining the network

A peer that wants to joint the DHT:

- computes its own *id*
- computes its own **finger table**
- ask any peer who has authority over *id*

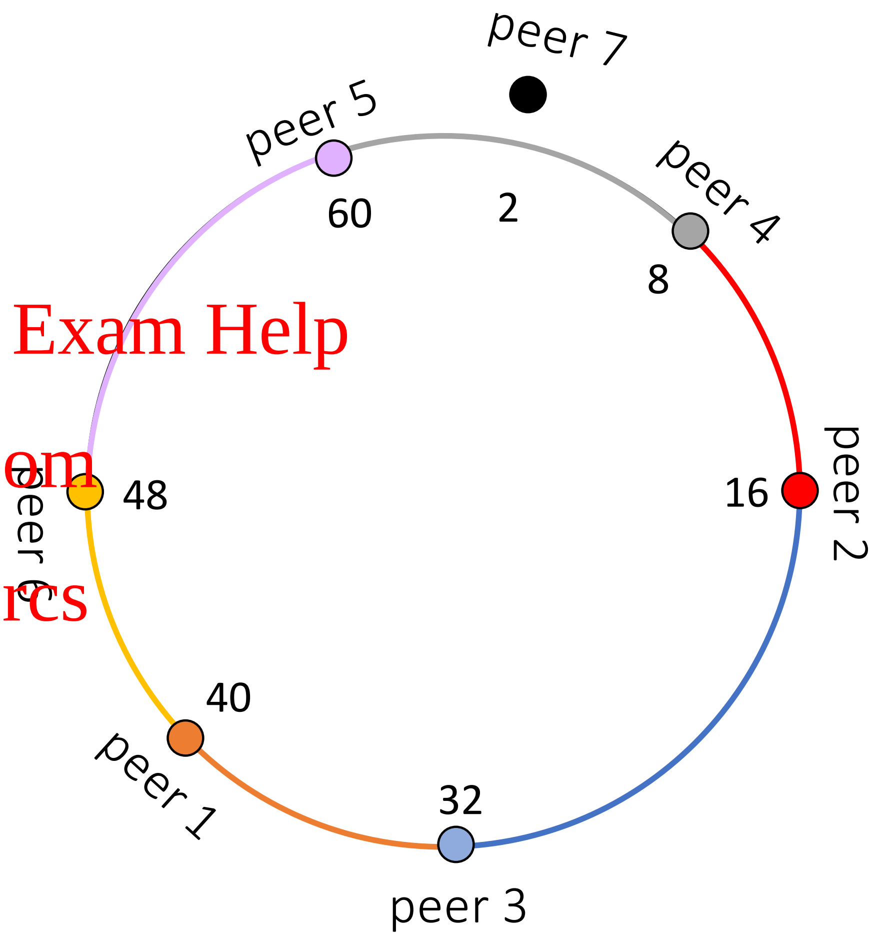


peer 4 has authority over 2

Chord: joining the network

A peer that wants to joint the DHT:

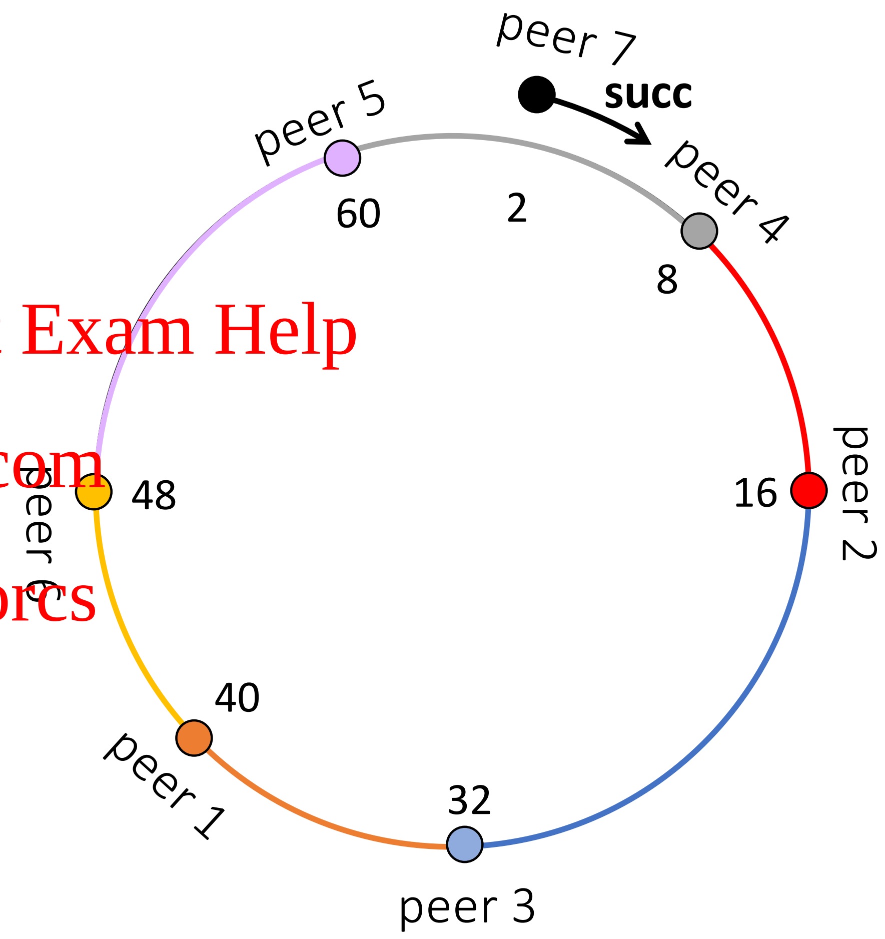
- computes its own *id*
- computes its own **finger table**
- ask any peer who has authority over *id*
- Trigger updates of the others' tables without creating anomalities!



Chord: joining the network

Steps to trigger update:

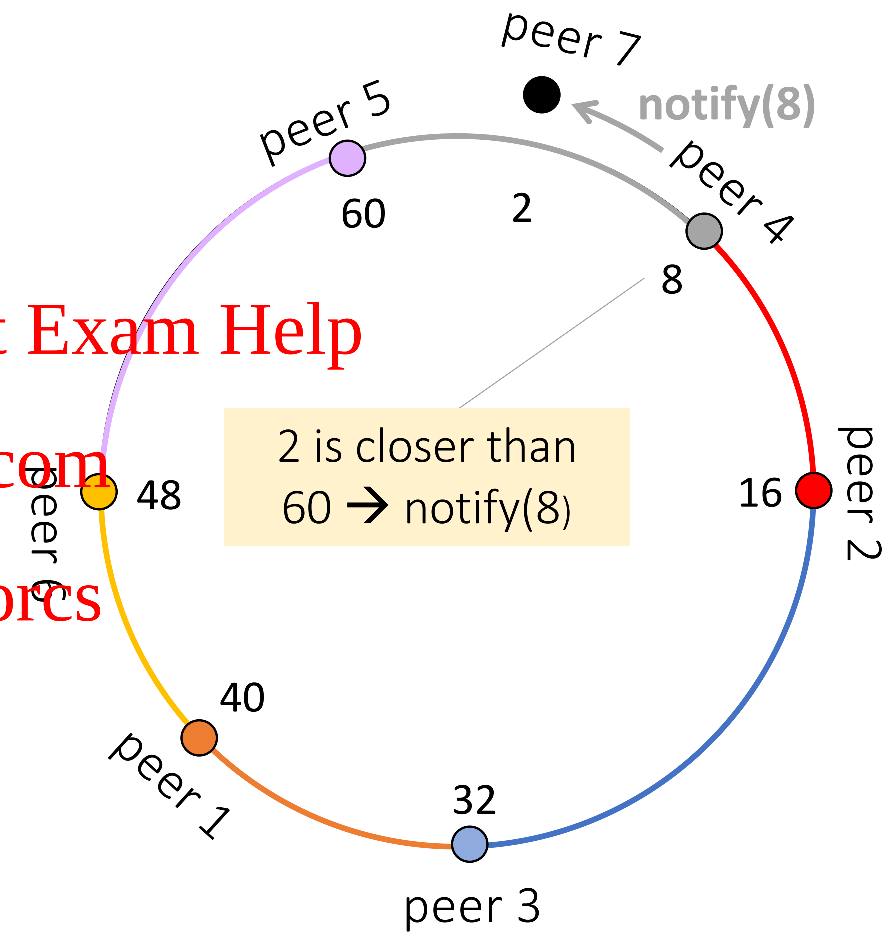
- Notify succ/pred pointers



Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers

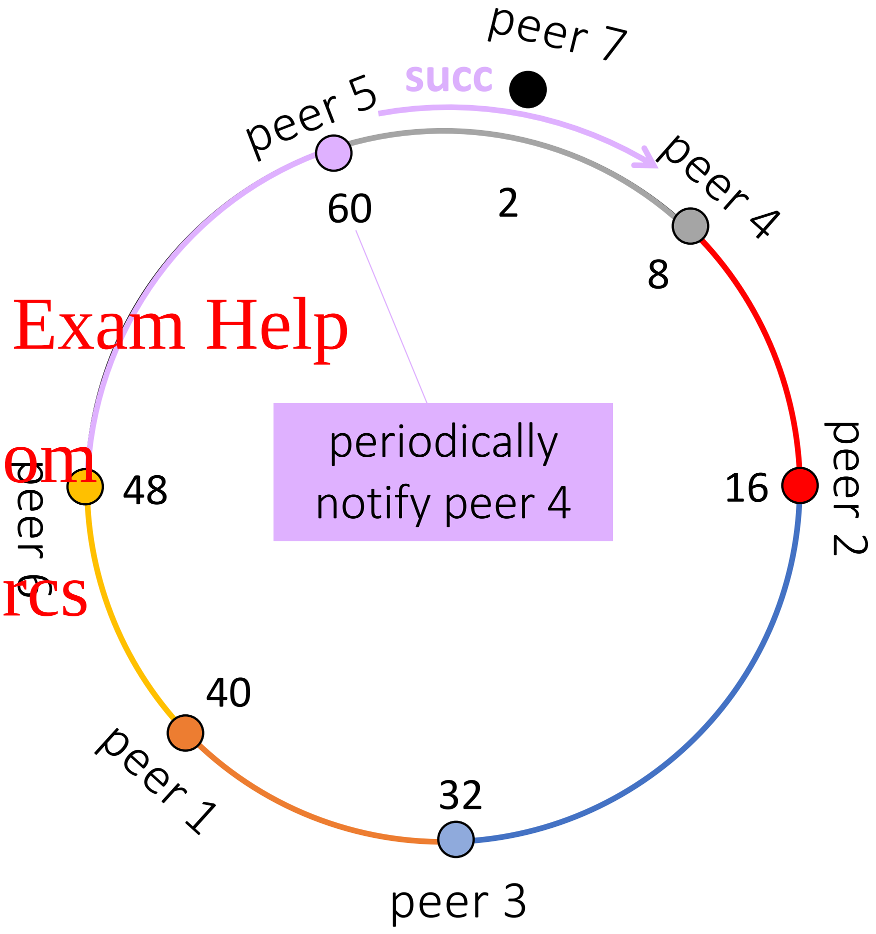


2 is closer than 60 → notify(8)

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers

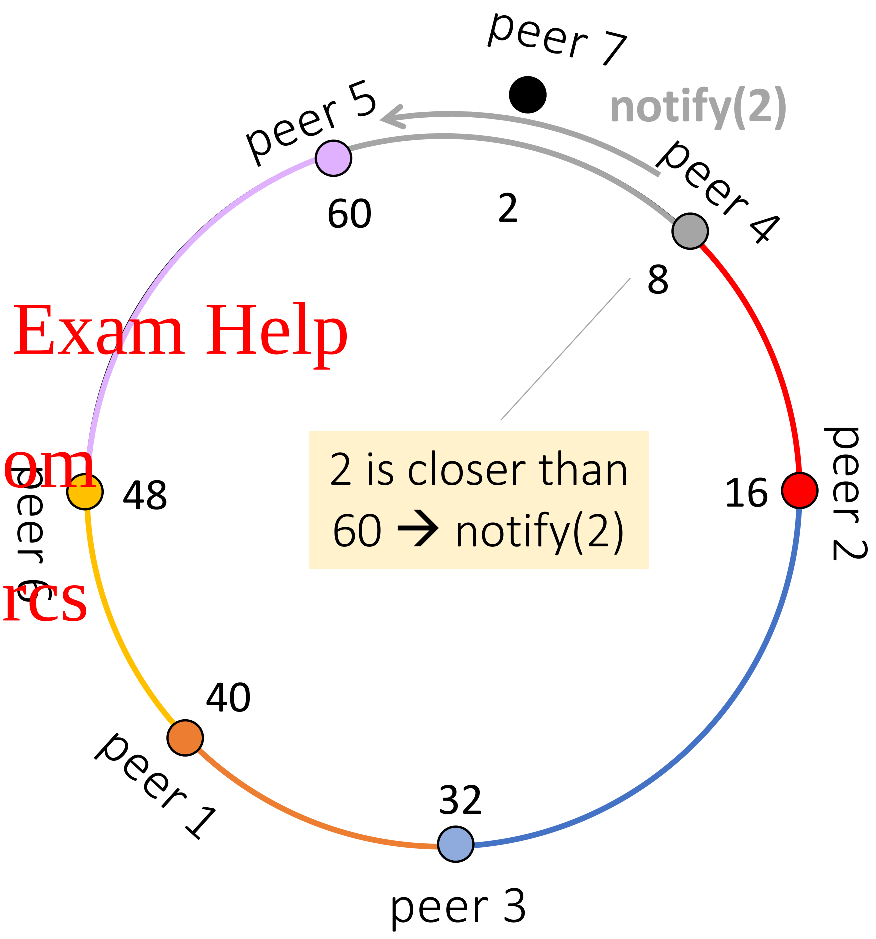


periodically notify peer 4

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers

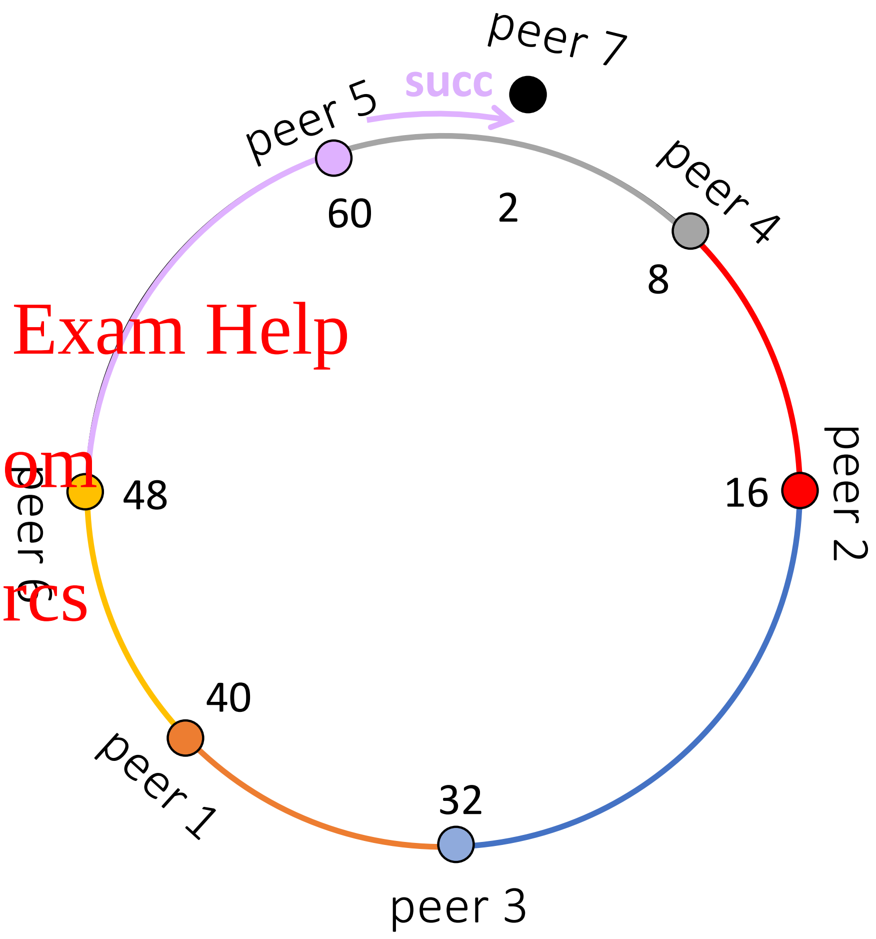


2 is closer than 60 → notify(2)

Chord: joining the network

Steps to trigger update:

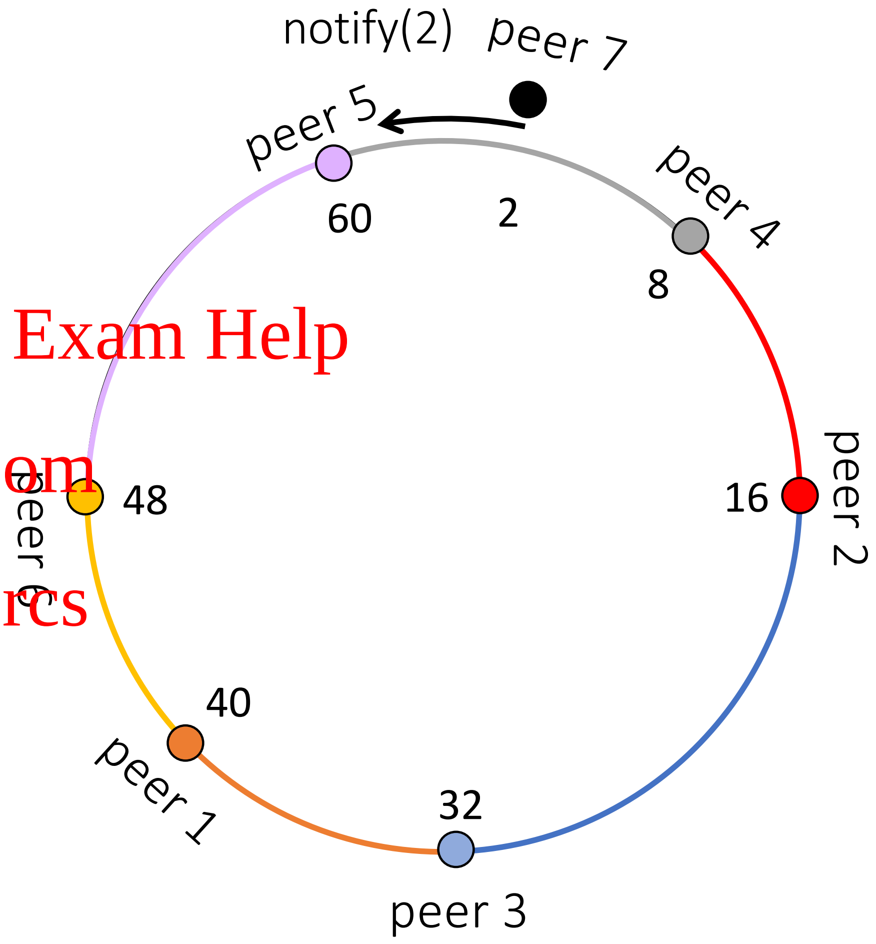
- Notify succ/pred pointers



Chord: joining the network

Steps to trigger update:

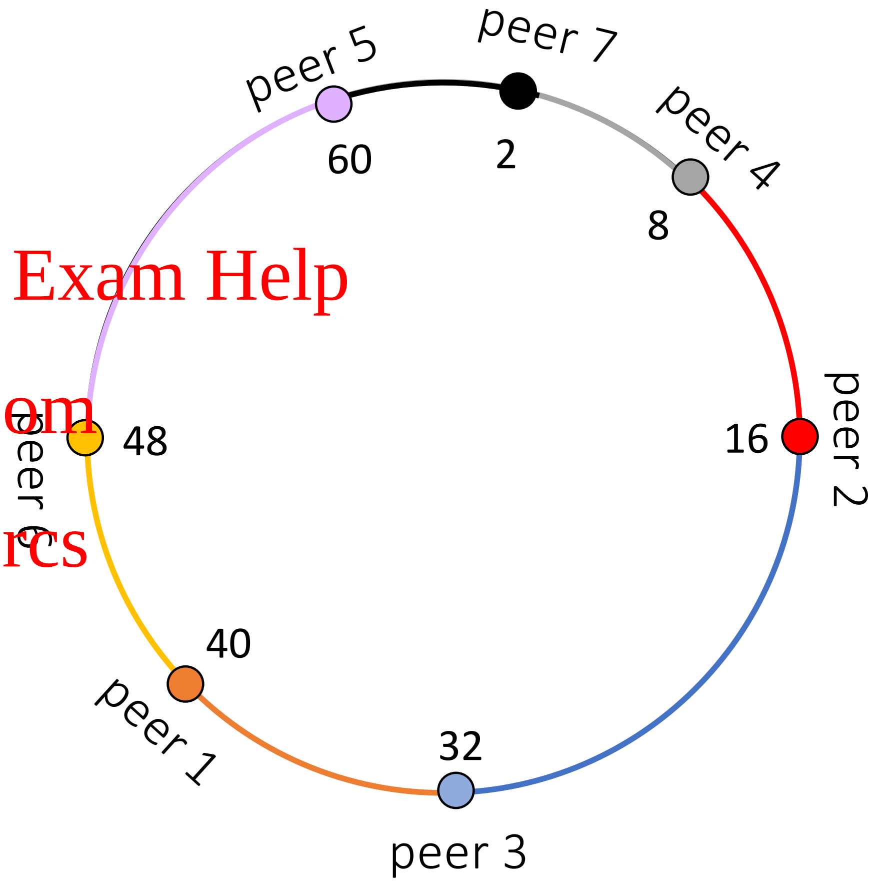
- Notify succ/pred pointers



Chord: joining the network

Steps to trigger update:

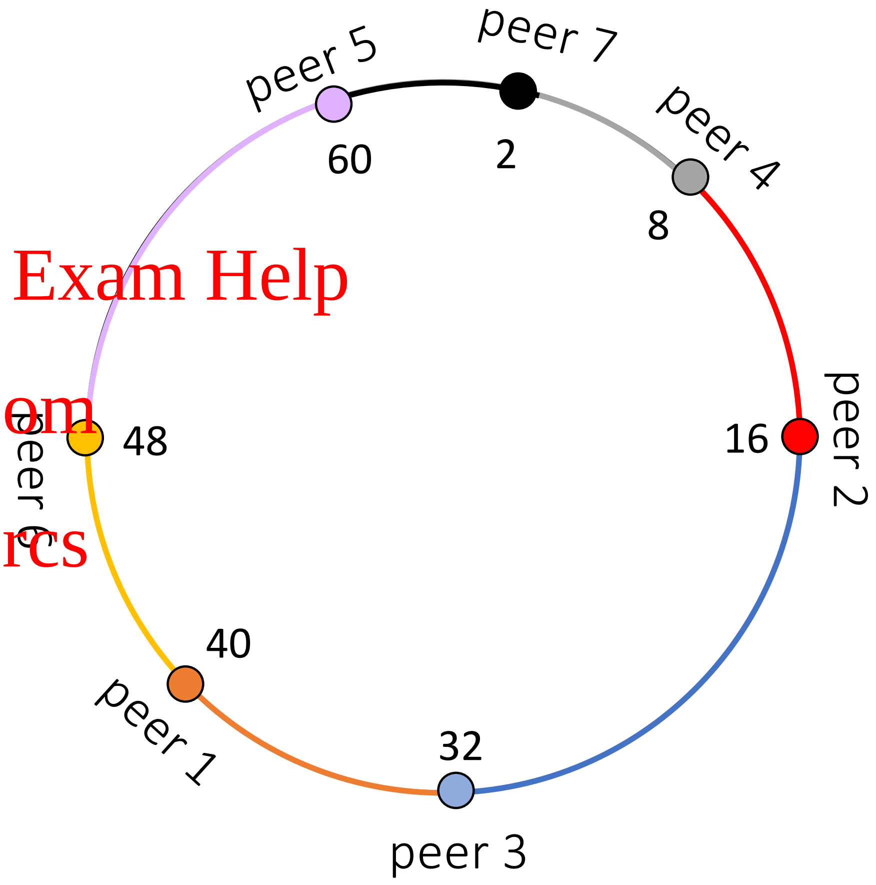
- Notify succ/pred pointers
- Safe to move resource mapping



Chord: joining the network

Steps to trigger update:

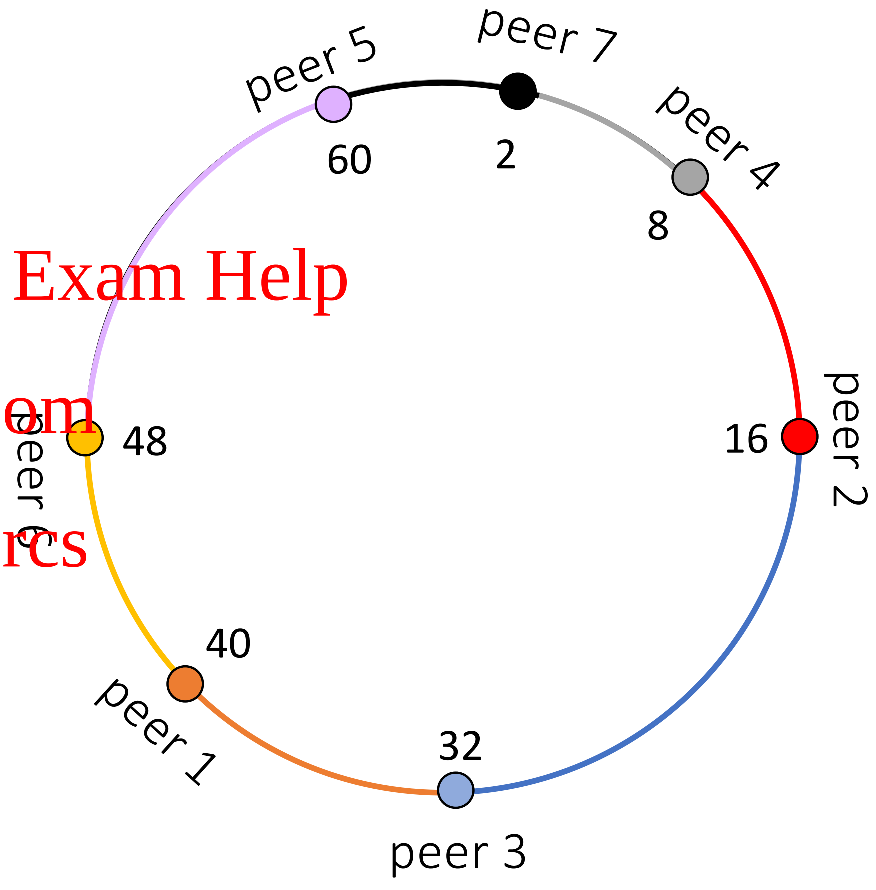
- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



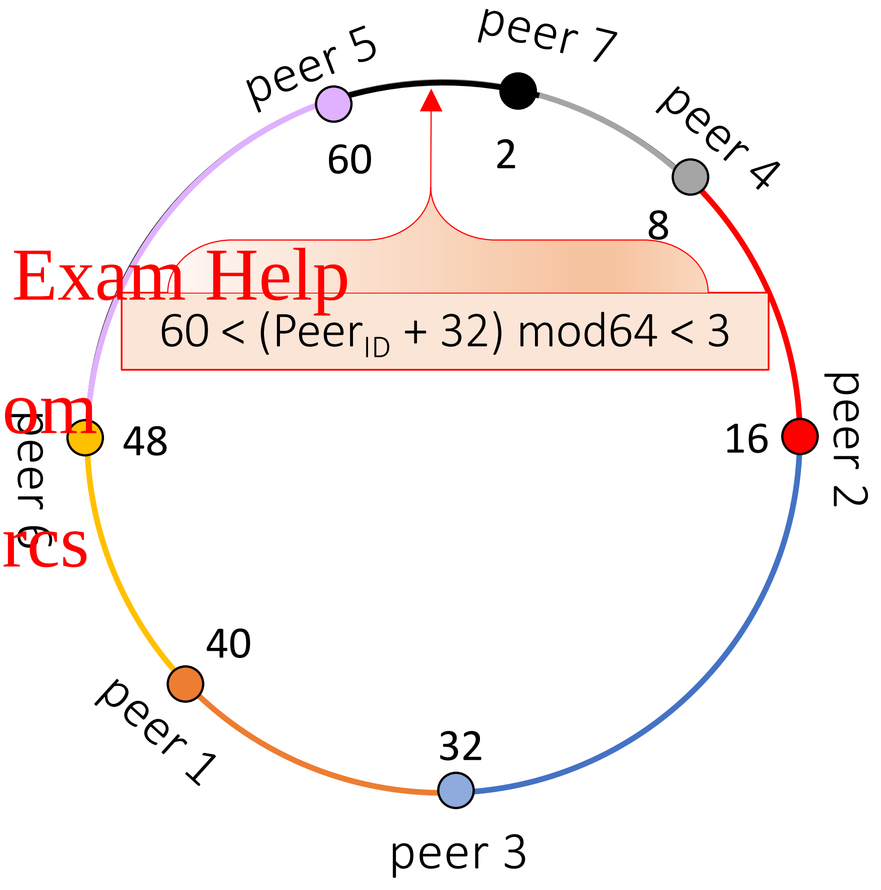
Notes:

- Peer 7 has authority over [61,2]
- Finger table has 6 entries ($0 < i < 5$)
- i= 5 means $\text{Peer}_{ID}+2^5 = \text{Peer}_{ID} + 32$
- Who are the Peers that might fall in Peer 7 authority field for i= 5 ?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



$60 < (\text{Peer}_{\text{ID}} + 32) \bmod 64 < 3$

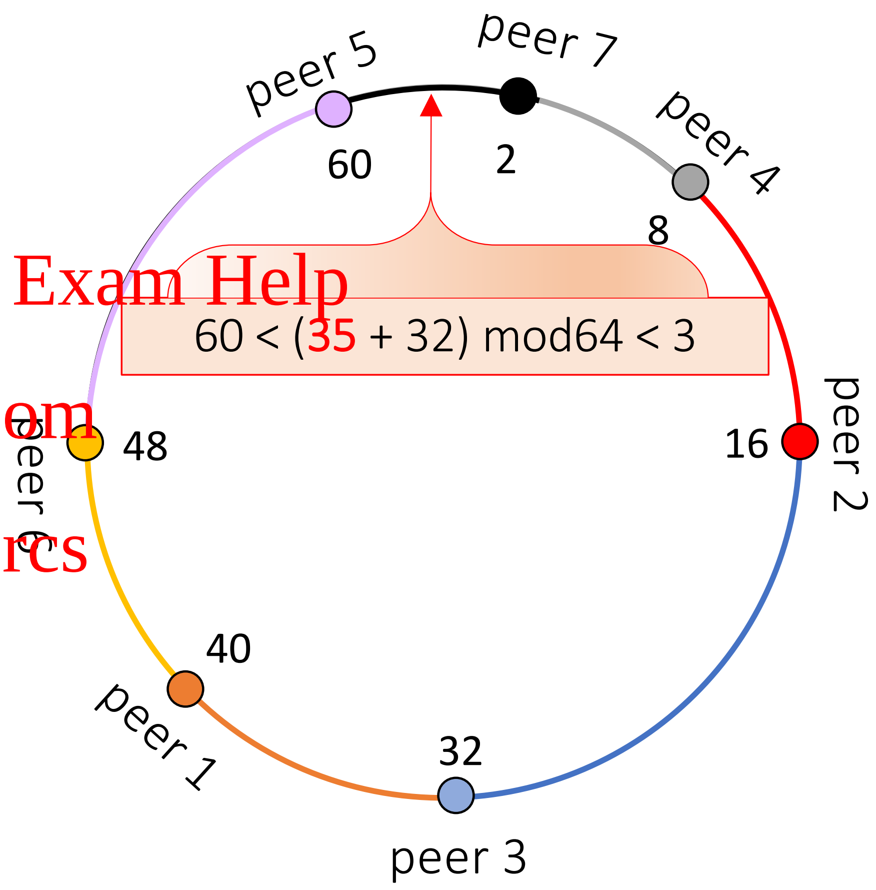
Notes:

- Peer 7 has authority over [61,2]
- Finger table has 6 entries ($0 < i < 5$)
- $i = 5$ means $\text{Peer}_{\text{ID}} + 2^5 = \text{Peer}_{\text{ID}} + 32$
- Who are the Peers that might fall in Peer 7 authority field for $i = 5$?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



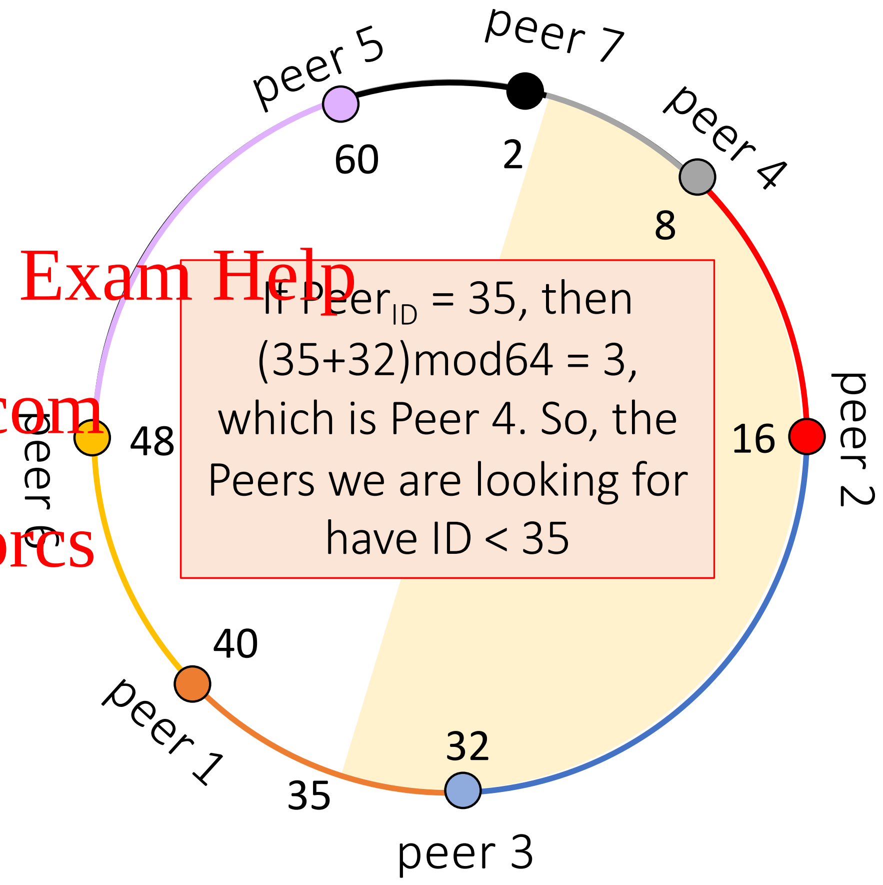
Notes:

- Peer 7 has authority over [61,2]
- Finger table has 6 entries (0 < *i* < 5)
- *i*= 5 means Peer$_{ID}$+2^5 = Peer$_{ID}$ + 32
- Who are the Peers that might fall in Peer 7 authority field for *i*= 5 ?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



If $Peer_{ID}$ = 35, then (35+32)mod64 = 3, which is Peer 4. So, the Peers we are looking for have ID < 35

Notes:

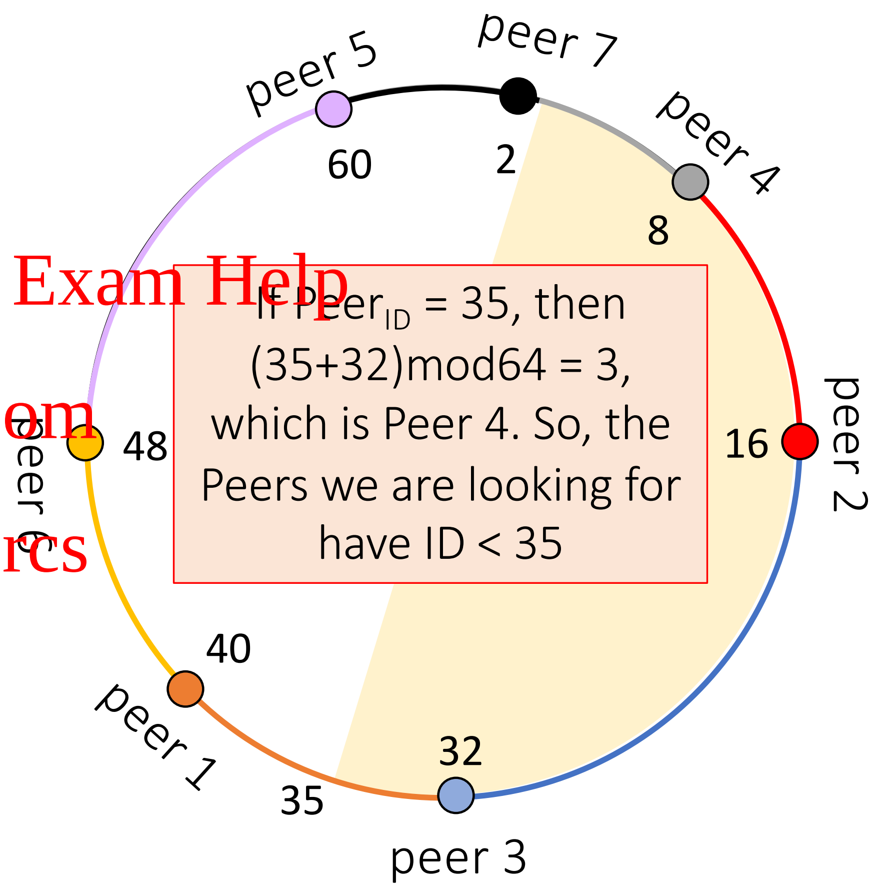
- Peer 7 has authority over (61,2)
- Finger table has 6 entries ($0 < i < 5$)
- i= 5 means $Peer_{ID}+2^5$ = $Peer_{ID}$ + 32
- Who are the Peers that might fall in Peer 7 authority field for i= 5 ?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Who is the closest Peer with ID < 35 ?



If Peer_{ID} = 35, then (35+32)mod64 = 3, which is Peer 4. So, the Peers we are looking for have ID < 35

Notes:

- Peer 7 has authority over (61,2)
- Finger table has 6 entries ($0 < i < 5$)
- i = 5 means $\text{Peer}_{\text{ID}} + 2^5 = \text{Peer}_{\text{ID}} + 32$
- Who are the Peers that might fall in Peer 7 authority field for i = 5 ?

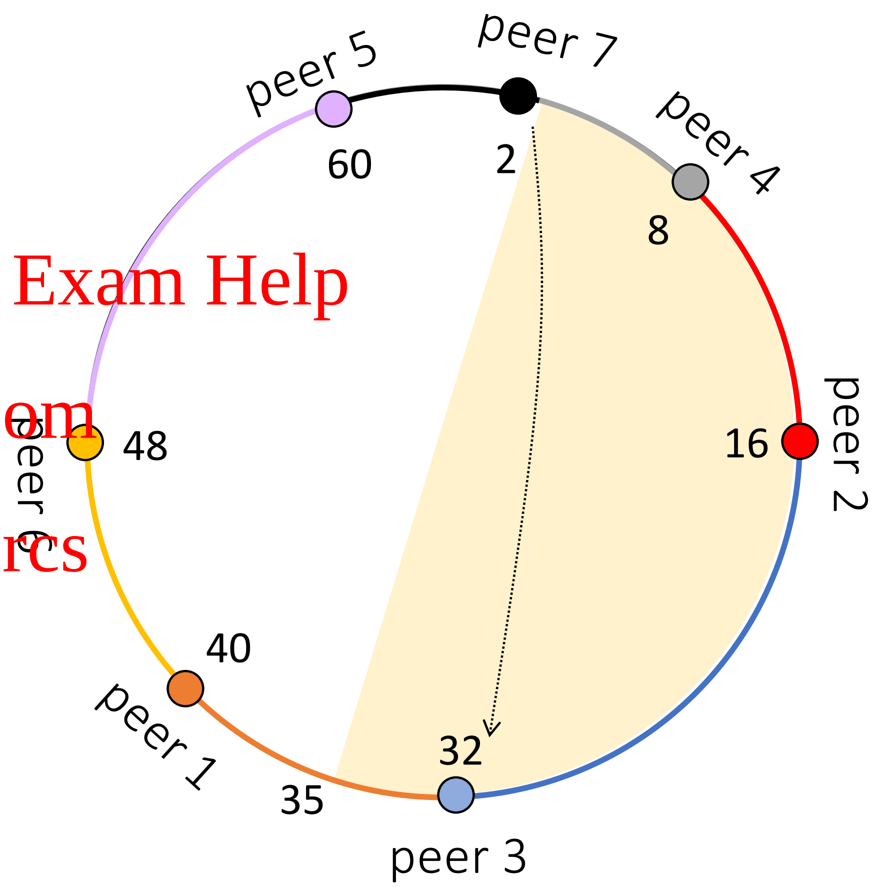
Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Message:

update(target= 32, new-peer=2)



Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

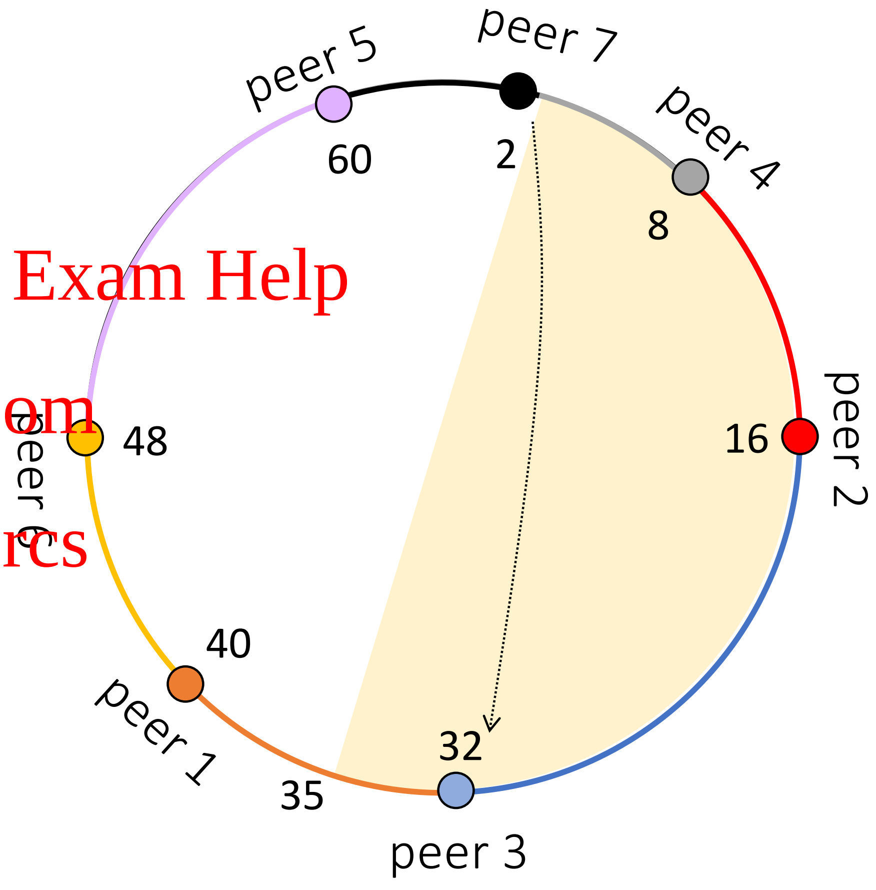
peer 3

i	key id	successor
0	32 + 2^0 mod 64 = 33	peer 1
1	32 + 2^1 mod 64 = 34	peer 1
2	32 + 2^2 mod 64 = 36	peer 1
3	32 + 2^3 mod 64 = 40	peer 1
4	32 + 2^4 mod 64 = 48	peer 6
5	32 + 2^5 mod 64 = 0	peer 4

to update

Message:

update(target= 32, new-peer=2)



Chord: joining the network

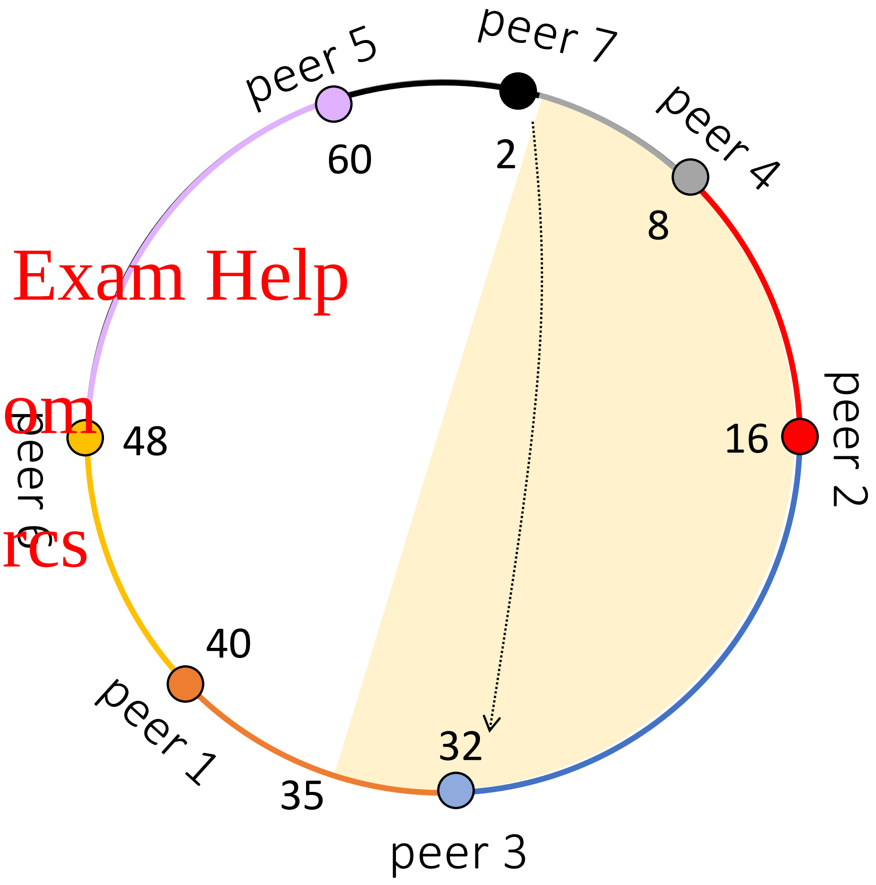
Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

peer 3

i	key id	successor
0	32 + 2^0 mod 64 = 33	peer 1
1	32 + 2^1 mod 64 = 34	peer 1
2	32 + 2^2 mod 64 = 36	peer 1
3	32 + 2^3 mod 64 = 40	peer 1
4	32 + 2^4 mod 64 = 48	peer 6
5	32 + 2^5 mod 64 = 0	**peer 7**

Message:
update(target= 32, new-peer=2)



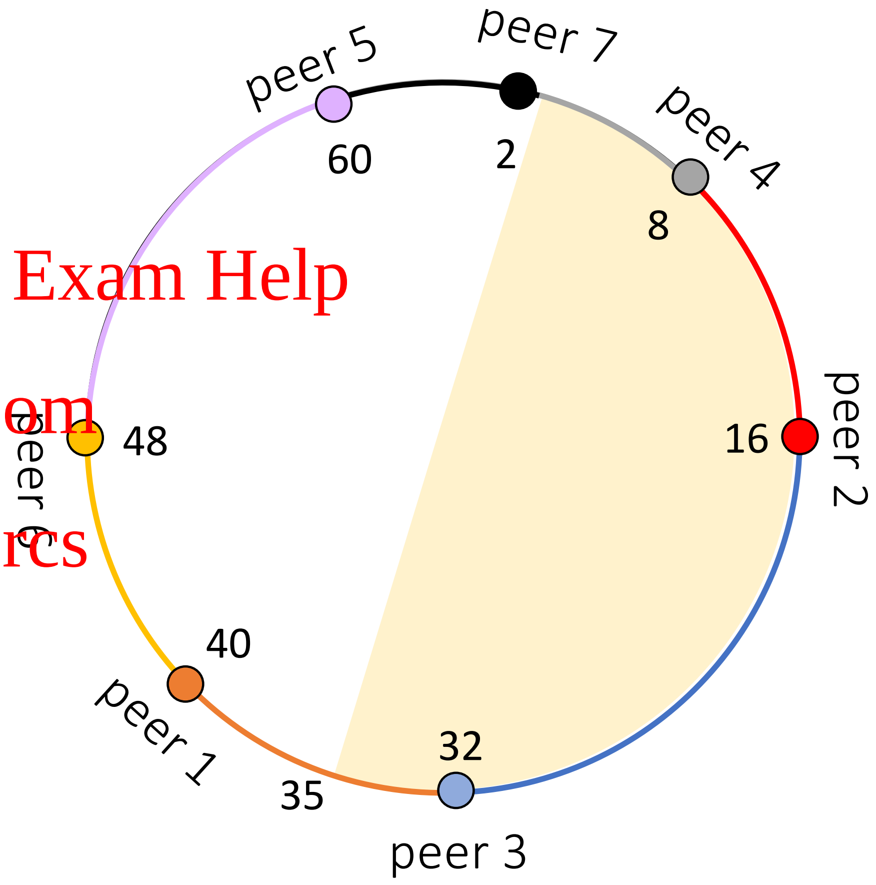
Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Notes:

- Now Peer 3 is fine!
- But, Peer 2 might not be!



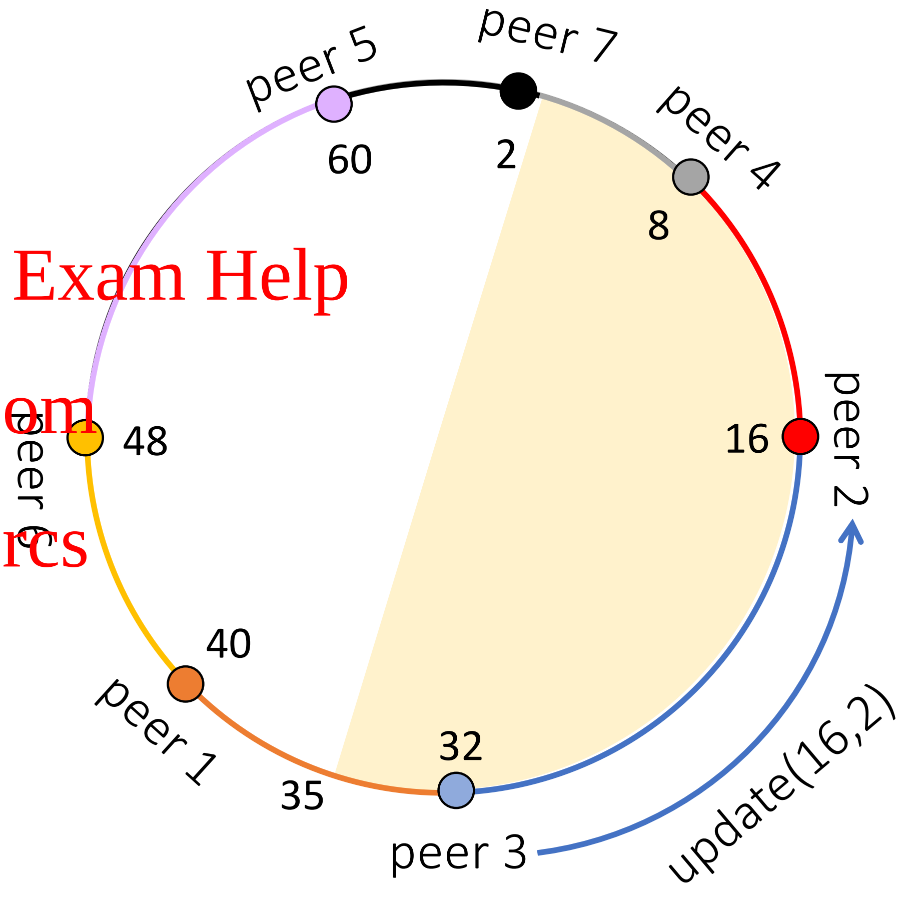
Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Notes:

- Now Peer 3 is fine!
- But, Peer 2 might not be!
- Peer 3 sends a message to Peer 2 to warn a potential Finger table update!



Chord: joining the network

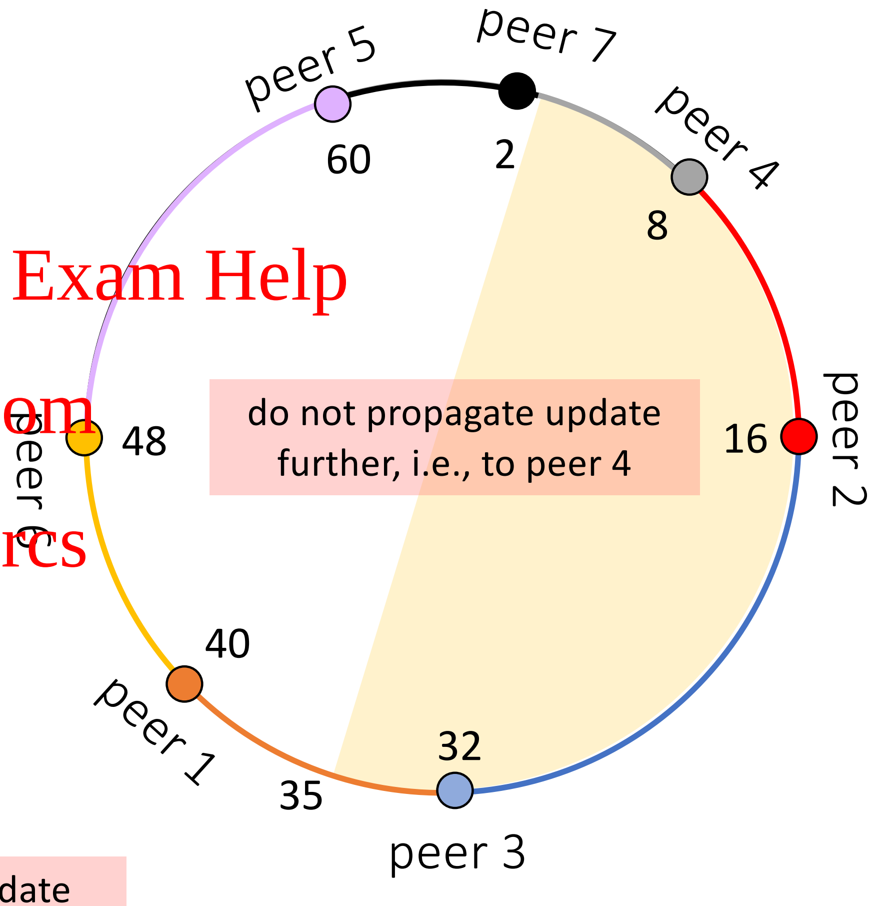
Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

peer 2

i	key id	successor
0	16 + 2^0 mod 64 = 17	peer 3
1	16 + 2^1 mod 64 = 18	peer 3
2	16 + 2^2 mod 64 = 20	peer 3
3	16 + 2^3 mod 64 = 24	peer 3
4	16 + 2^4 mod 64 = **32**	peer 3
5	16 + 2^5 mod 64 = 48	peer 6

no update needed

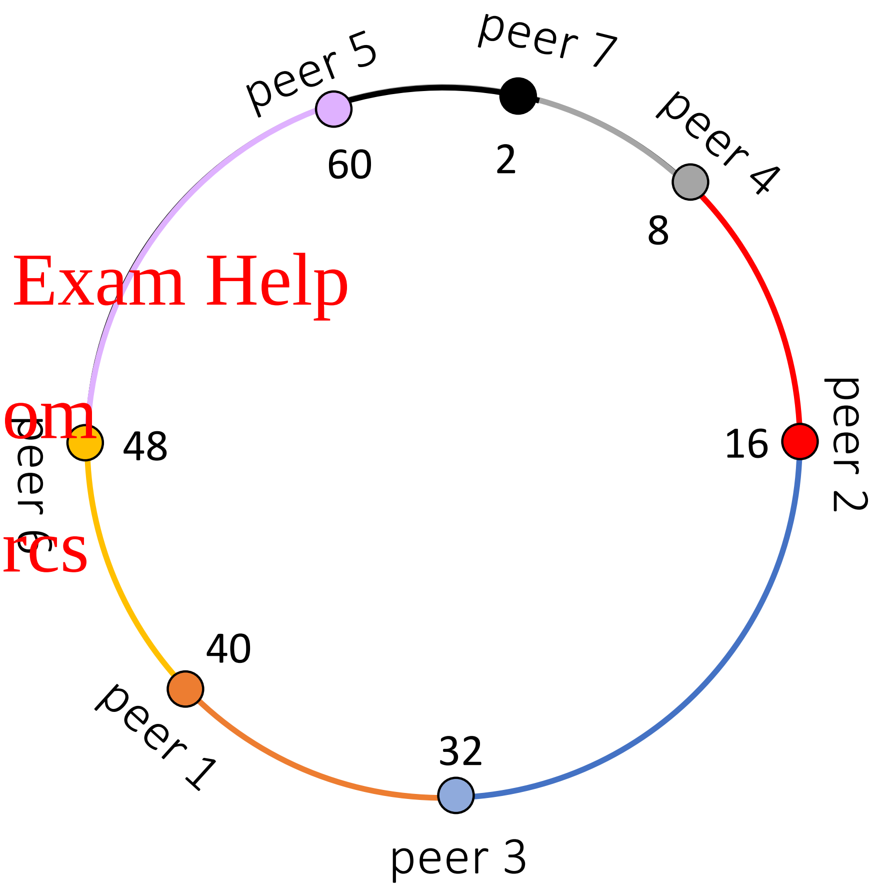


do not propagate update further, i.e., to peer 4

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



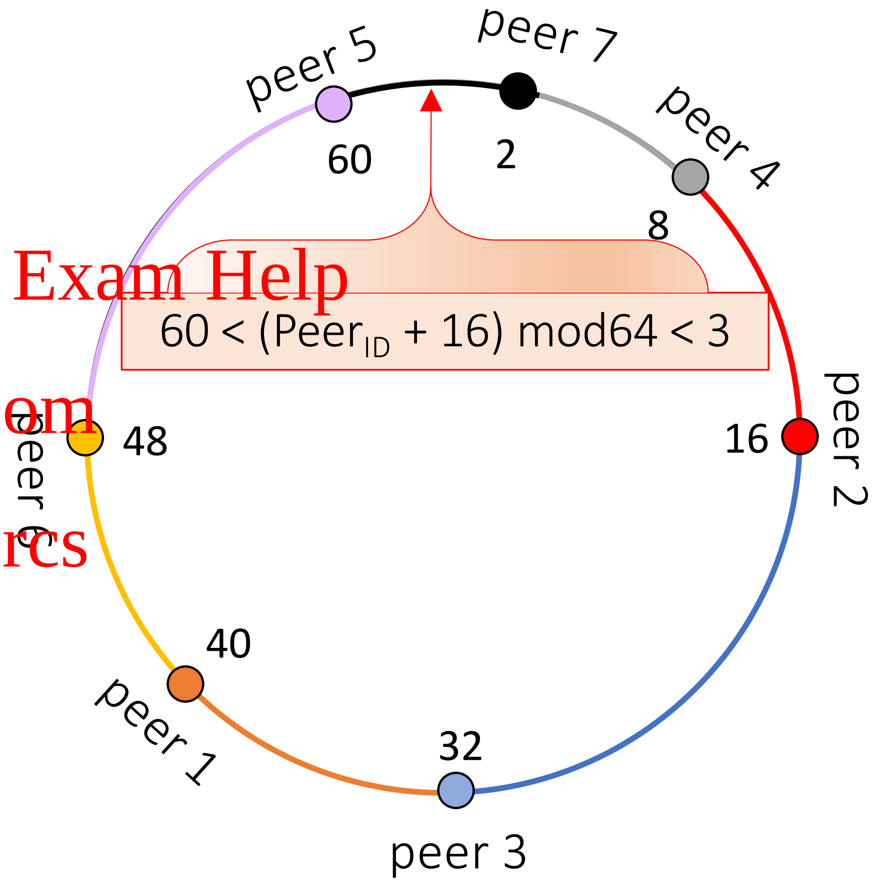
Notes:

- The case i = 5 now completed!!!
- What about i = 4 now?
- i = 4 means $\text{Peer}_{\text{ID}} + 2^4 = \text{Peer}_{\text{ID}} + 16$
- Who are the Peers that might fall in Peer 7 authority field for i= 4 ?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



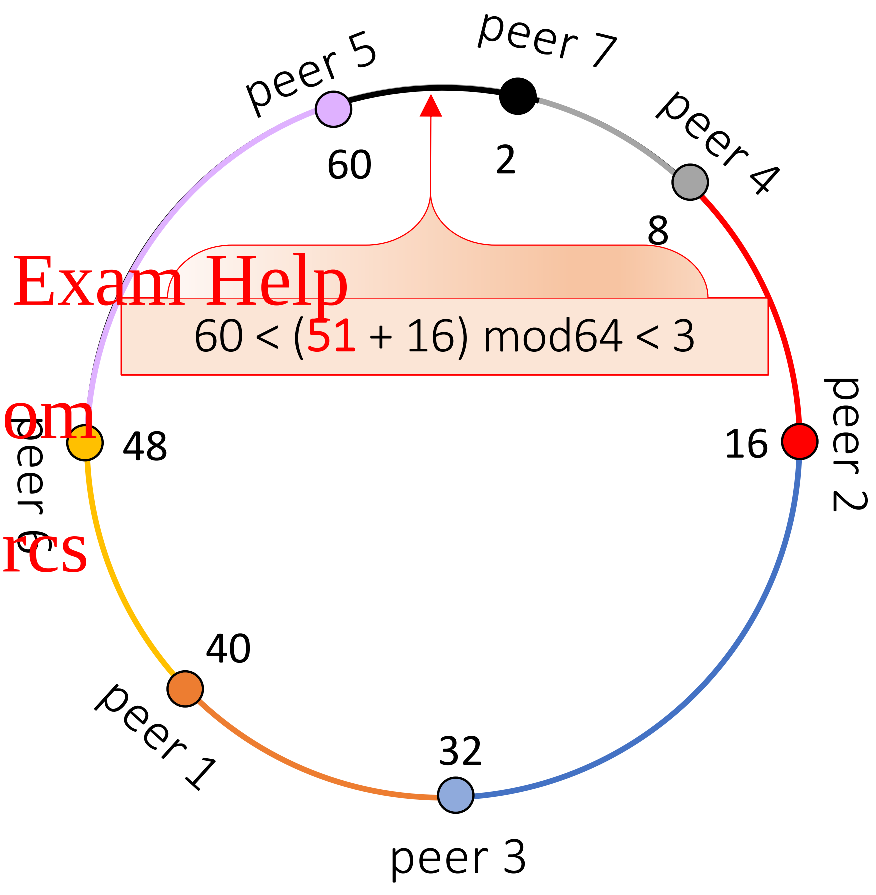
Notes:

- The case i = 5 now completed!!!
- What about i = 4 now?
- i = 4 means $\text{Peer}_{ID} + 2^4 = \text{Peer}_{ID} + 16$
- Who are the Peers that might fall in Peer 7 authority field for i= 4 ?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



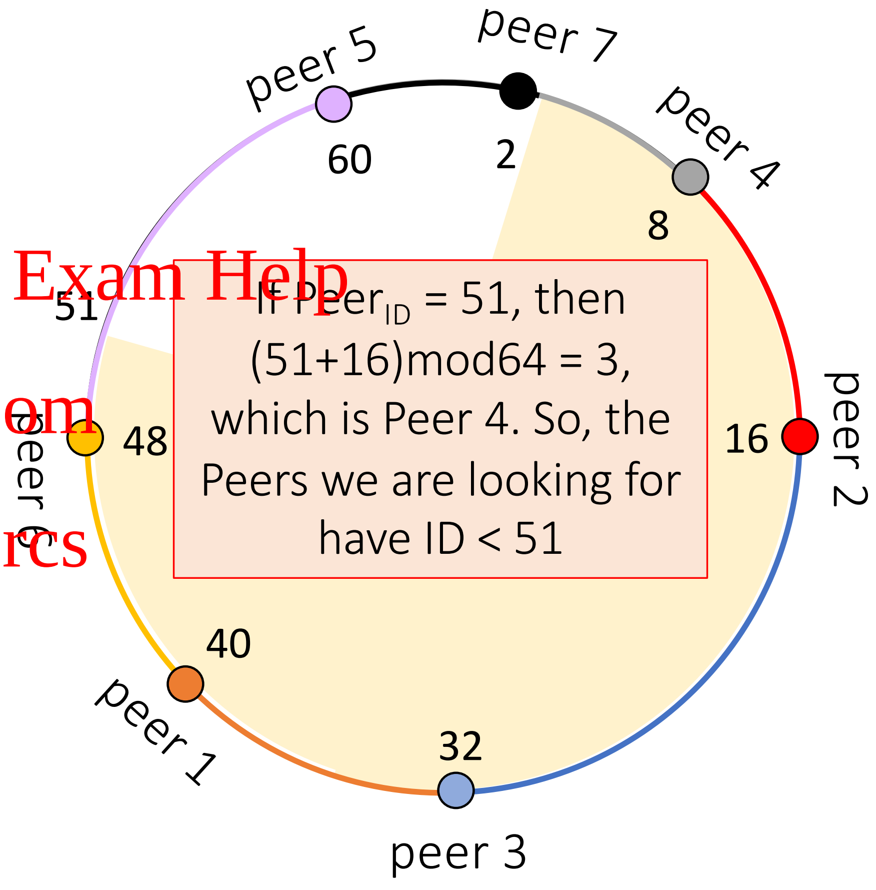
Notes:

- The case i = 5 now completed!!!
- What about i = 4 now?
- i = 4 means $\text{Peer}_{\text{ID}} + 2^4 = \text{Peer}_{\text{ID}} + 16$
- Who are the Peers that might fall in Peer 7 authority field for *i*= 4 ?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



If Peer$_{ID}$ = 51, then (51+16)mod64 = 3, which is Peer 4. So, the Peers we are looking for have ID < 51

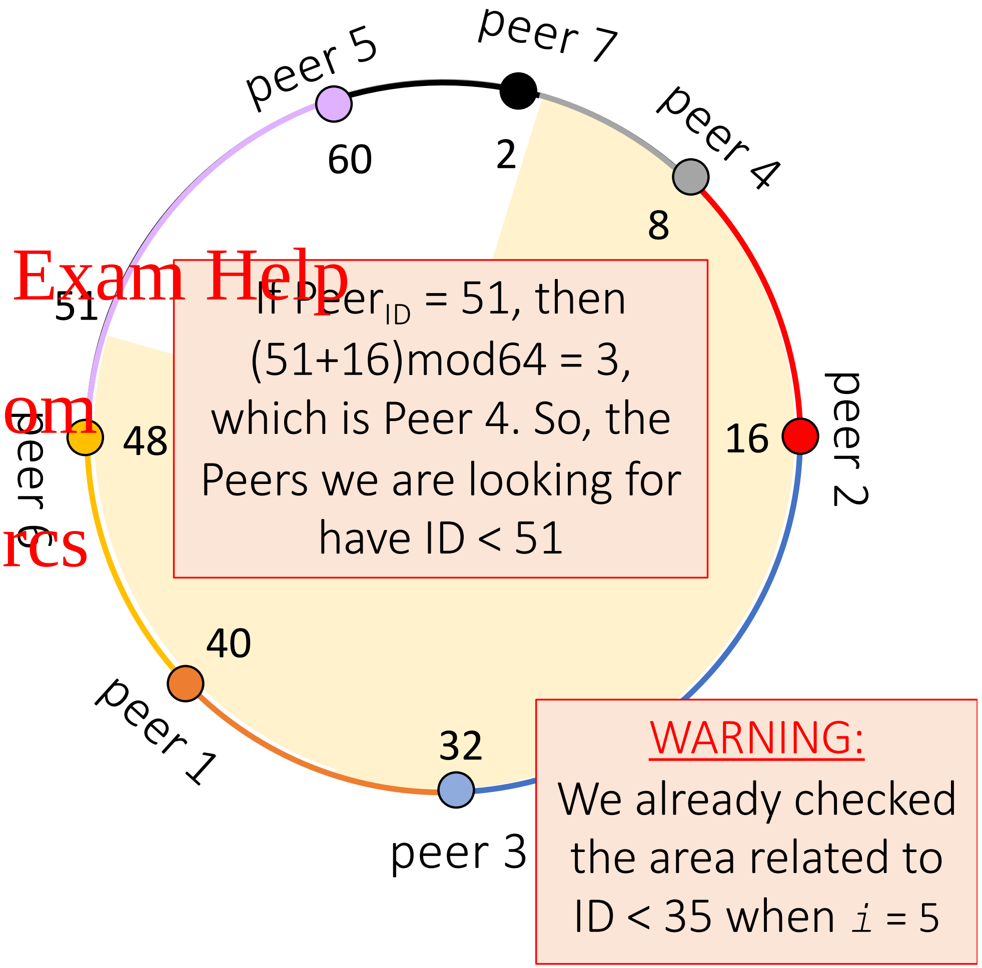
Notes:

- The case i = 5 now completed!!!
- What about i = 4 now?
- i = 4 means Peer$_{ID}$ + 2^4 = Peer$_{ID}$ + 16
- Who are the Peers that might fall in Peer 7 authority field for i= 4 ?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



If Peer_{ID} = 51, then (51+16)mod64 = 3, which is Peer 4. So, the Peers we are looking for have ID < 51

WARNING: We already checked the area related to ID < 35 when i = 5

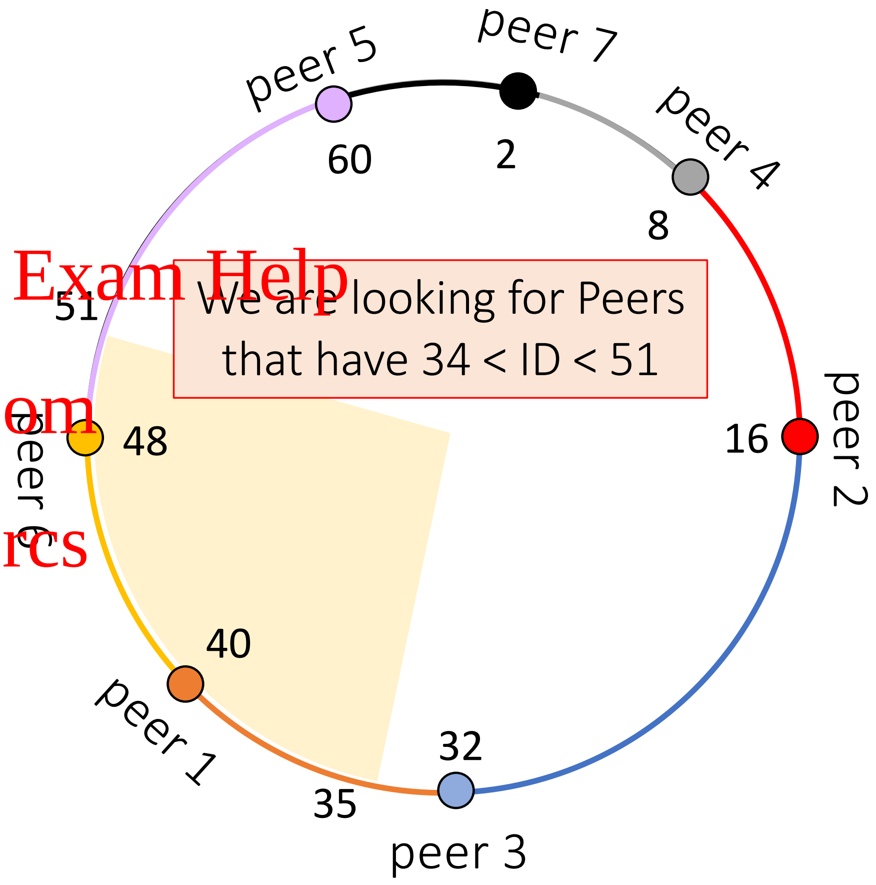
Notes:

- The case i = 5 now completed!!!
- What about i = 4 now?
- i = 4 means $\text{Peer}_{\text{ID}} + 2^4 = \text{Peer}_{\text{ID}} + 16$
- Who are the Peers that might fall in Peer 7 authority field for i= 4 ?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



We are looking for Peers that have 34 < ID < 51

Notes:

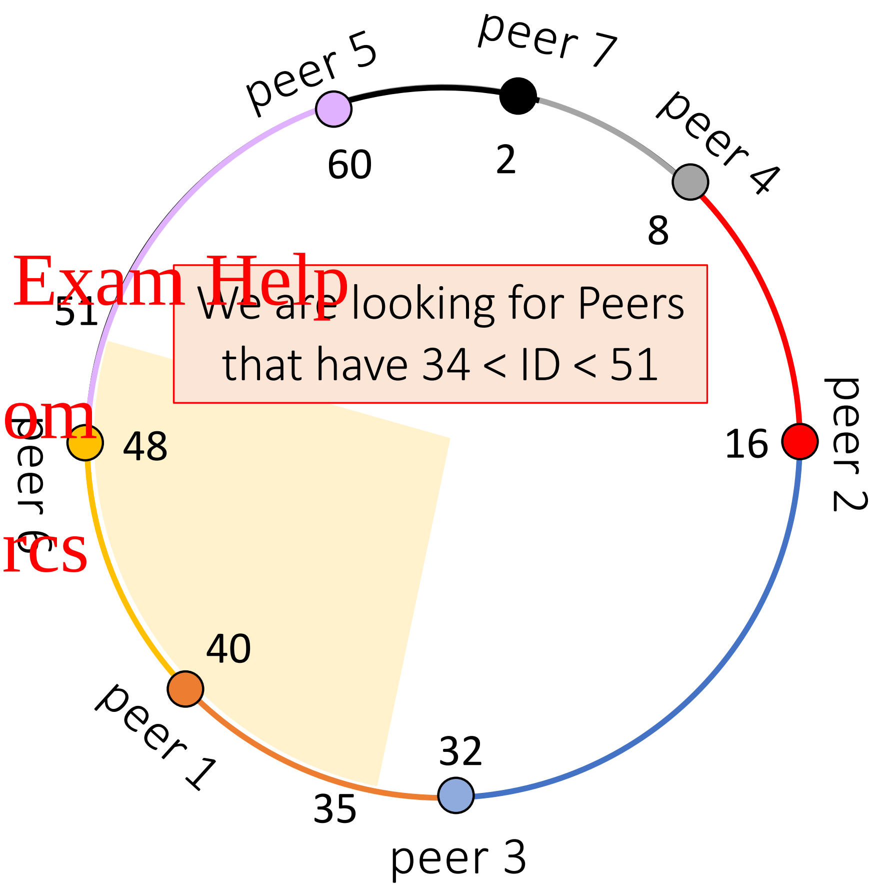
- The case i = 5 now completed!!!
- What about i = 4 now?
- i = 4 means $\text{Peer}_{\text{ID}} + 2^4 = \text{Peer}_{\text{ID}} + 16$
- Who are the Peers that might fall in Peer 7 authority field for i= 4 ?

Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Who is the closest Peer with ID < 51 ?



We are looking for Peers that have 34 < ID < 51

Notes:

- The case i = 5 now completed!!!
- What about i = 4 now?
- i = 4 means $\text{Peer}_{\text{ID}} + 2^4 = \text{Peer}_{\text{ID}} + 16$
- Who are the Peers that might fall in Peer 7 authority field for i= 4 ?

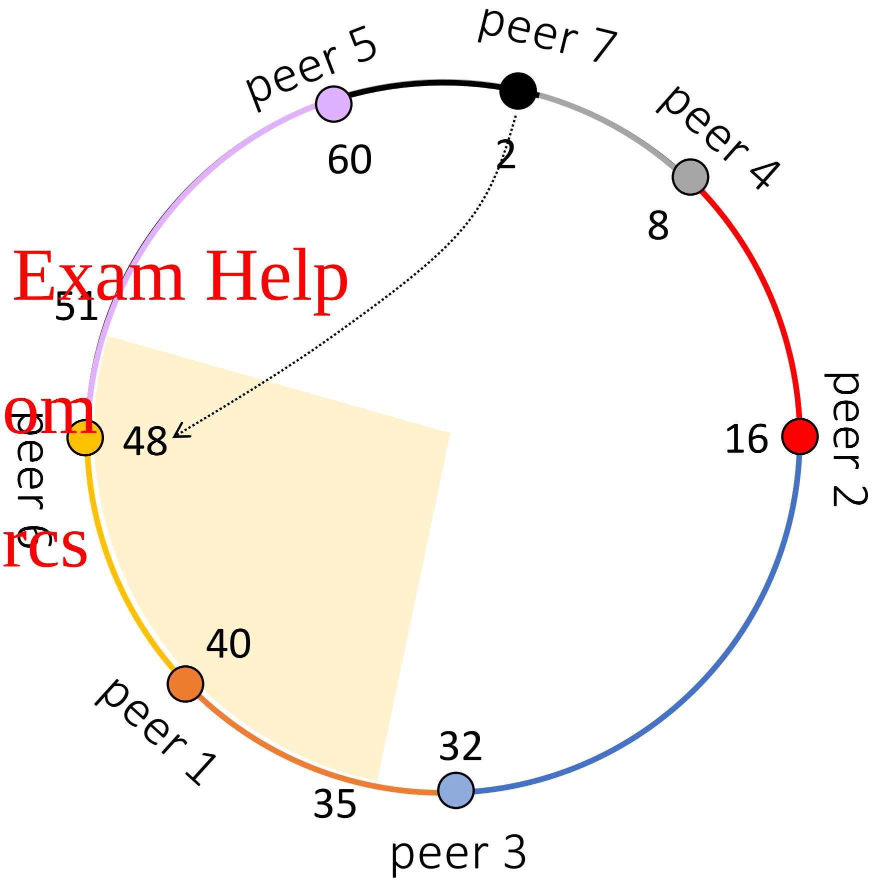
Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Message:

update(target= 48, new-peer=2)



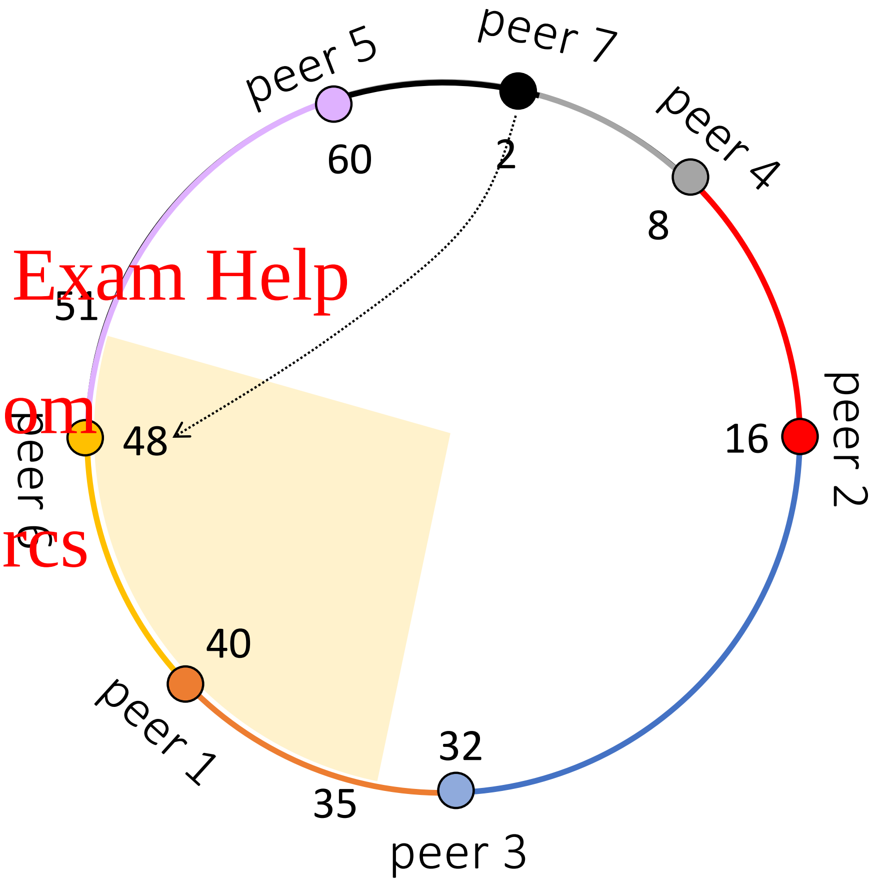
Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Message:

update(target= 48, new-peer=2)



peer 6

i	key id	successor
0	48 + 2^0 mod 64 = 49	peer 5
1	48 + 2^1 mod 64 = 50	peer 5
2	48 + 2^2 mod 64 = 52	peer 5
3	48 + 2^3 mod 64 = 56	peer 5
4	48 + 2^4 mod 64 = 0	peer 4
5	48 + 2^5 mod 64 = 16	peer 2

to update

Chord: joining the network

Steps to trigger update:

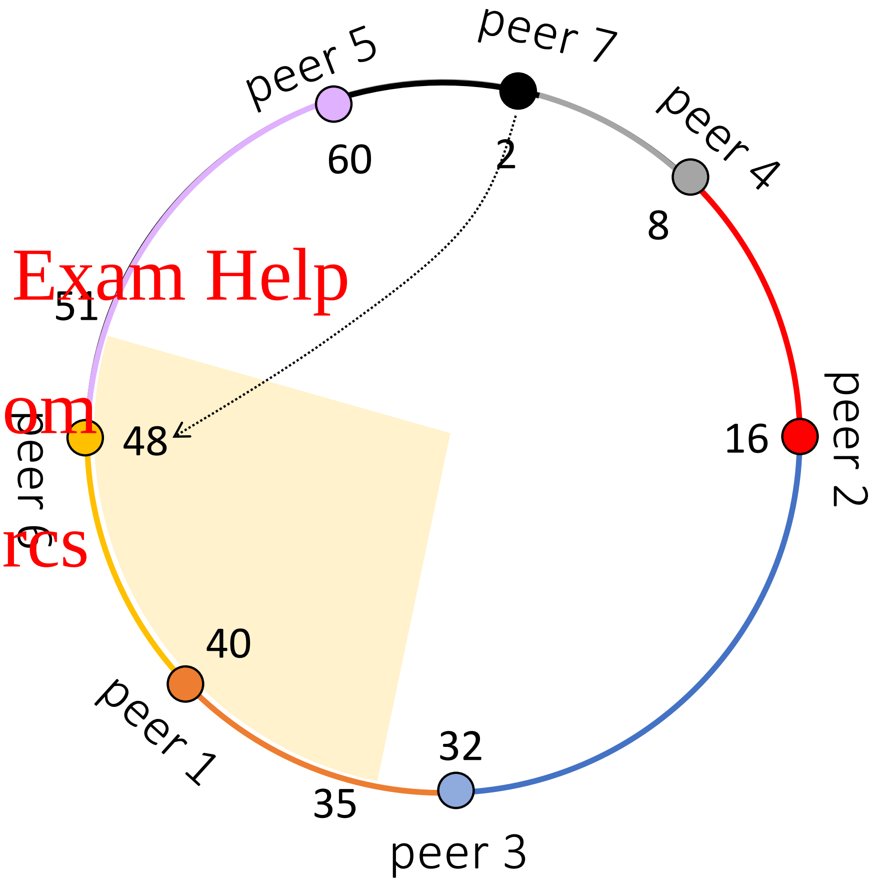
- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

peer 6

i	key id	successor
0	48 + 2^0 mod 64 = 49	peer 5
1	48 + 2^1 mod 64 = 50	peer 5
2	48 + 2^2 mod 64 = 52	peer 5
3	48 + 2^3 mod 64 = 56	peer 5
4	48 + 2^4 mod 64 = 0	**peer 7**
5	48 + 2^5 mod 64 = 16	peer 2

Message:

update(target= 48, new-peer=2)



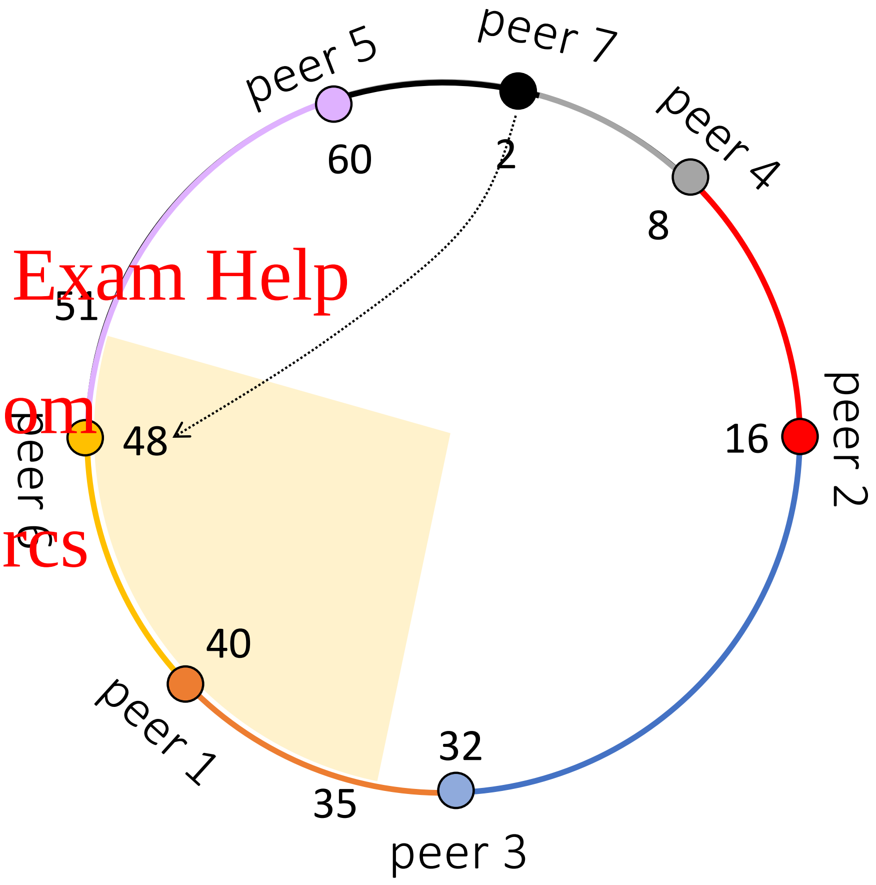
Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Notes:

- Now Peer 6 is fine!
- But, Peer 1 might not be!



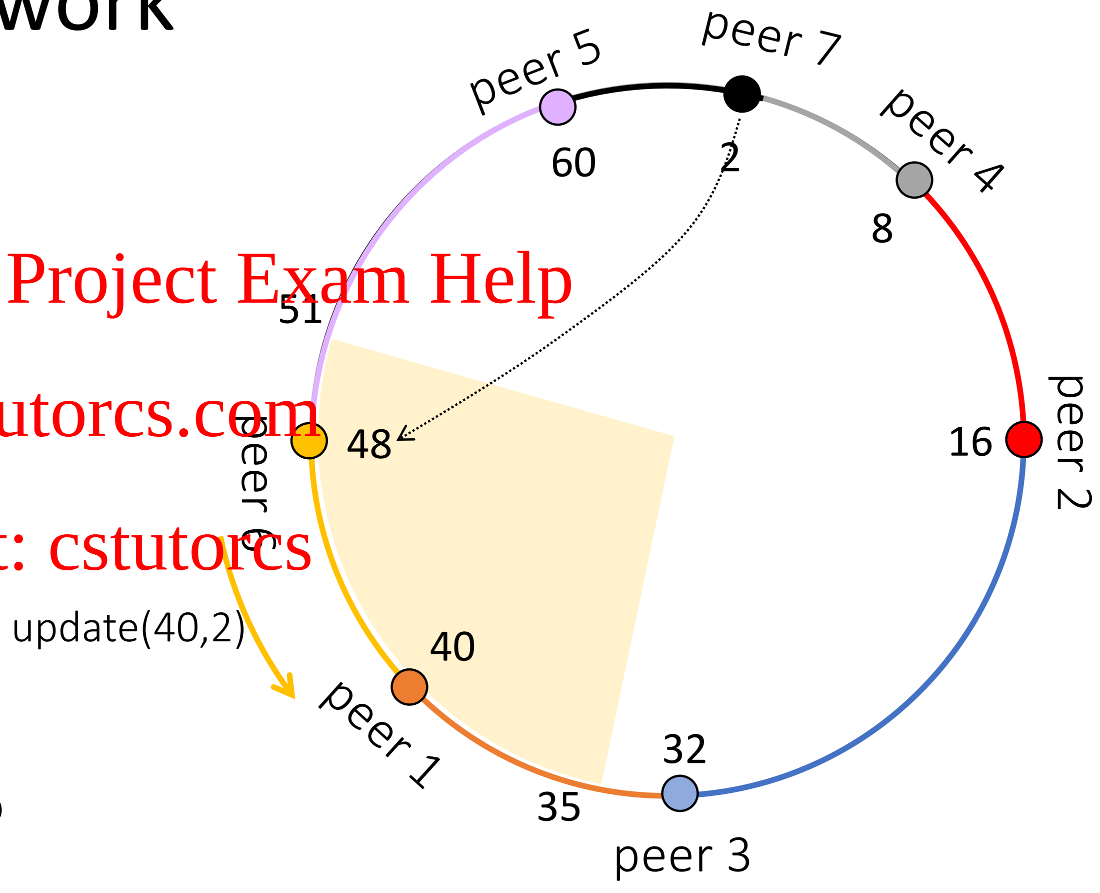
Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Notes:

- Now Peer 6 is fine!
- But, Peer 1 might not be!
- Peer 6 sends a message to Peer 1 to warn a potential Finger table update!



Chord: joining the network

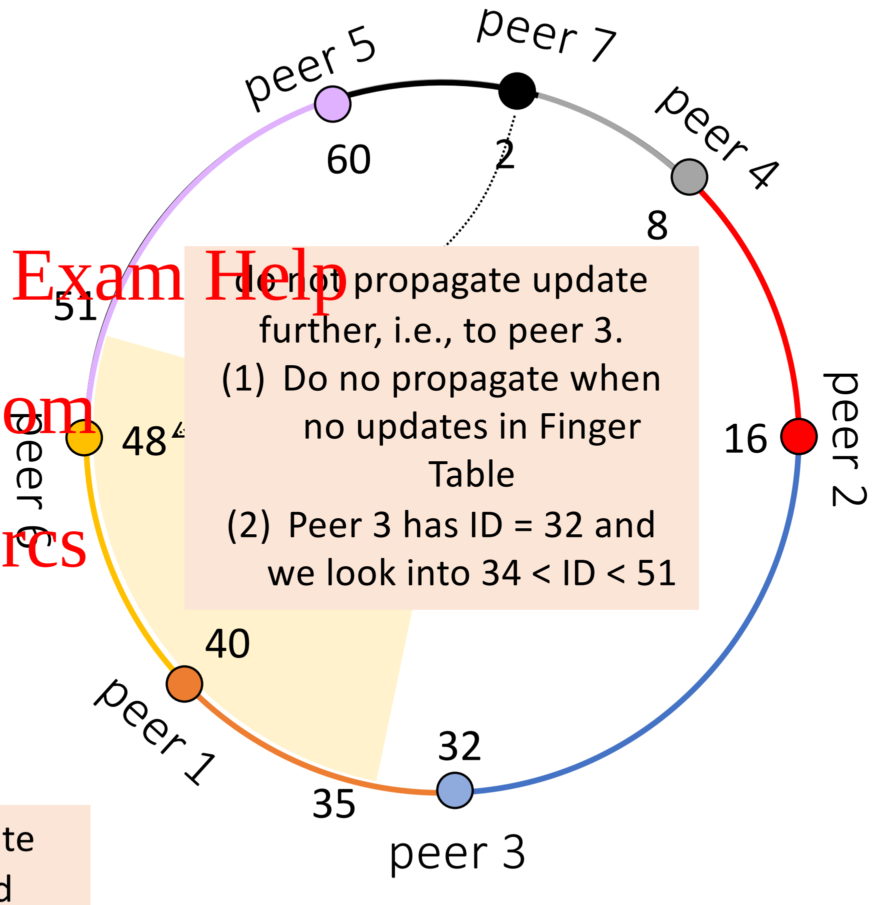
Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

peer 1

i	key id	successor
0	$40 + 2^0$ mod 64 = 41	peer 6
1	$40 + 2^1$ mod 64 = 42	peer 6
2	$40 + 2^2$ mod 64 = 44	peer 6
3	$40 + 2^3$ mod 64 = 48	peer 6
4	$40 + 2^4$ mod 64 = **56**	peer **5**
5	$40 + 2^5$ mod 64 = 8	peer 4

no update needed



do not propagate update further, i.e., to peer 3.

(1) Do no propagate when no updates in Finger Table

(2) Peer 3 has ID = 32 and we look into 34 < ID < 51

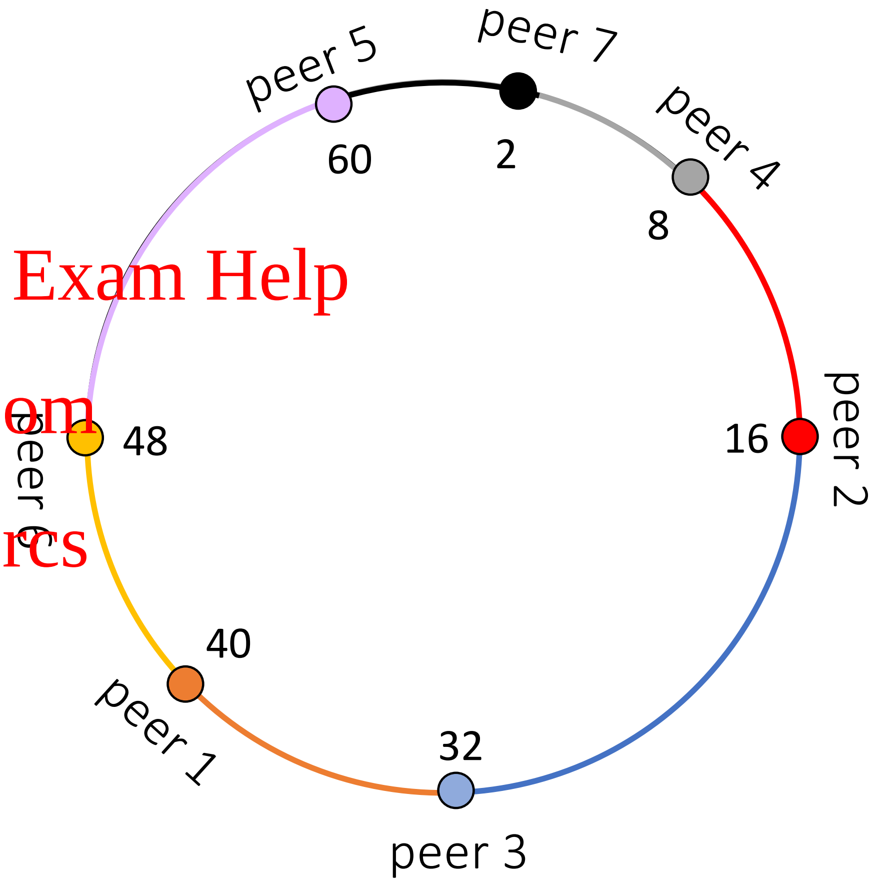
Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!

Notes:

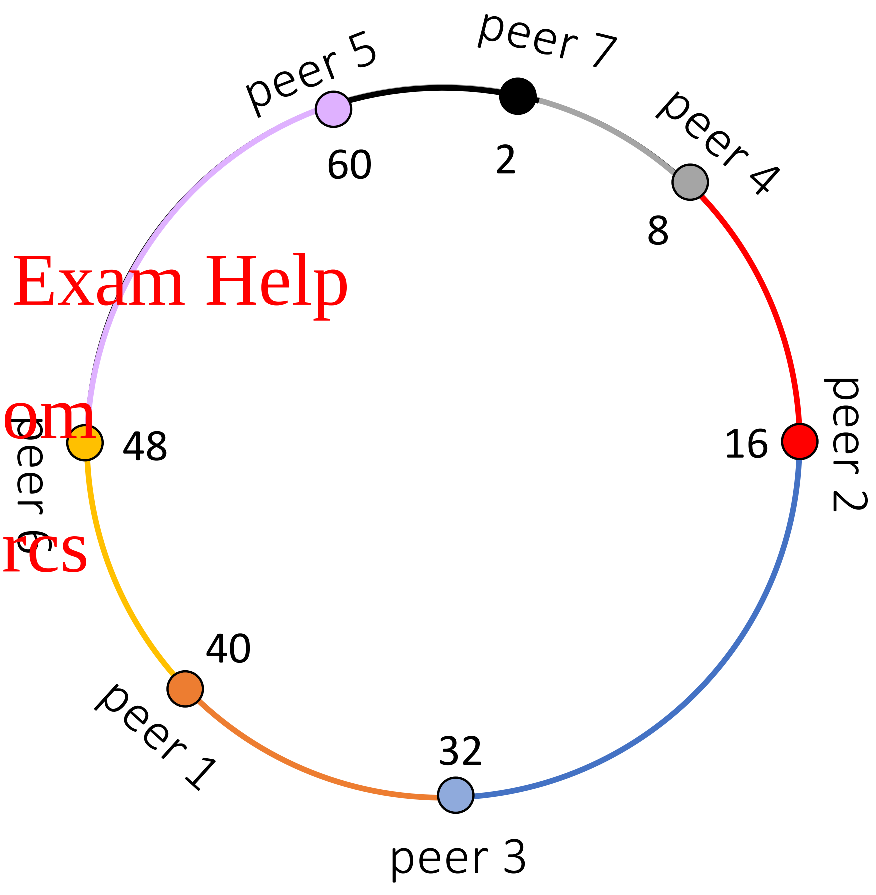
- The cases i = (5,4) now completed!
- What about i = 3 now?
- i = 3 means $\text{Peer}_{\text{ID}} + 2^3 = \text{Peer}_{\text{ID}} + 8$
- Who are the Peers that might fall in Peer 7 authority field for i= 3 ?



Chord: joining the network

Steps to trigger update:

- Notify succ/pred pointers
- Safe to move resource mapping
- Now? Trigger finger tables update for the other peers!



Notes:

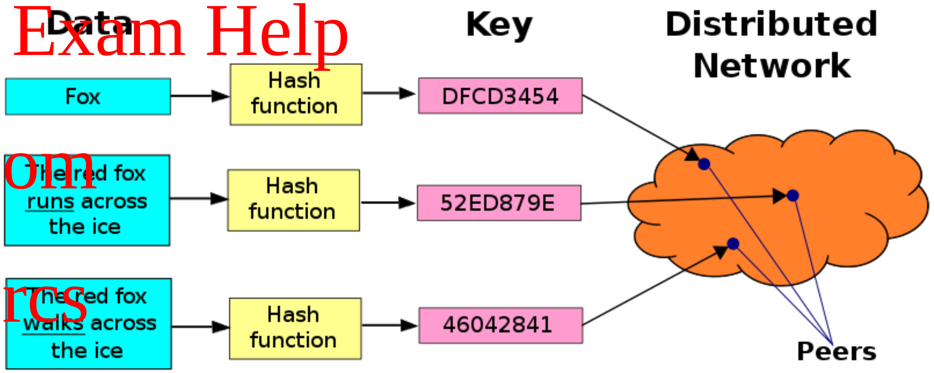
- The cases i = (5,4) nc mpleted!
- What about i =
- i = 3 mea = Peer$_{ID}$ + 8
- Who s that might fall in Peer 7 rity field for i= 3 ?

This is an iterative process!!

DHT recap

- DHT is a class of a decentralized distributed system that provides a lookup service like a hash table: (key, value) pairs are stored in a DHT

- Keys are unique identifiers which map to values, which in turn can be anything from addresses, to documents, to arbitrary data.

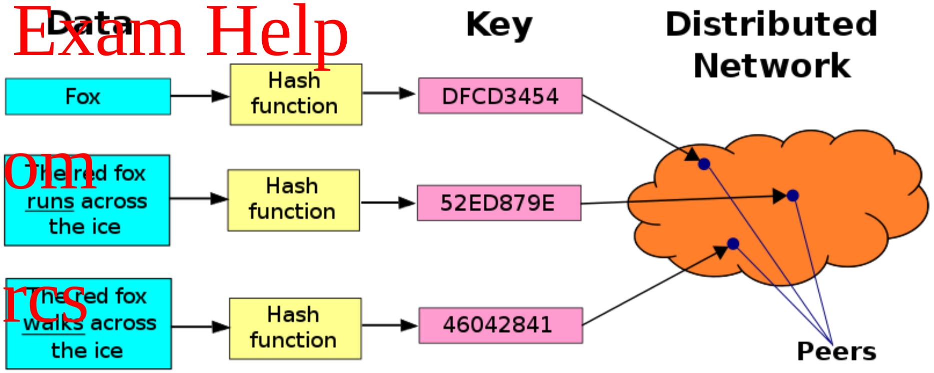


DHT recap

- DHTs can form the infrastructure that can be used to build complex services like P2P

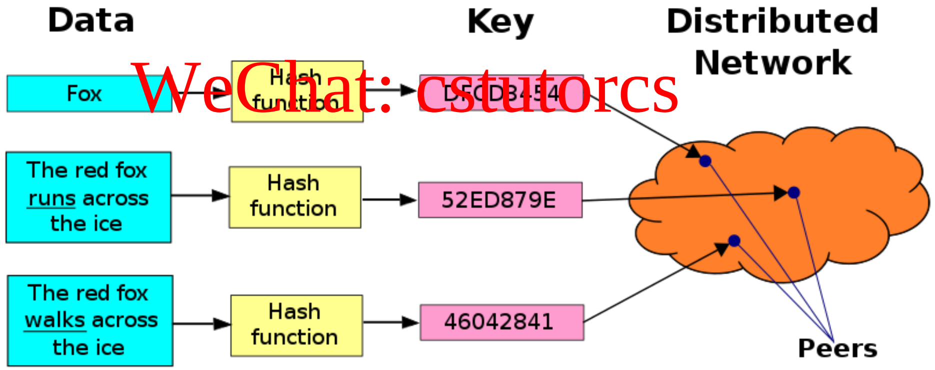
- WARNING: not only that!

- Do not associate DHT to only P2P!



DHT recap

- It is an approach for Key-Value store --> The value is stored in a database in the form of a two-value tuple. One is the identifier(key) and other is the actual data(Value), and hence it is called as Key Value Store.



The key-value abstraction

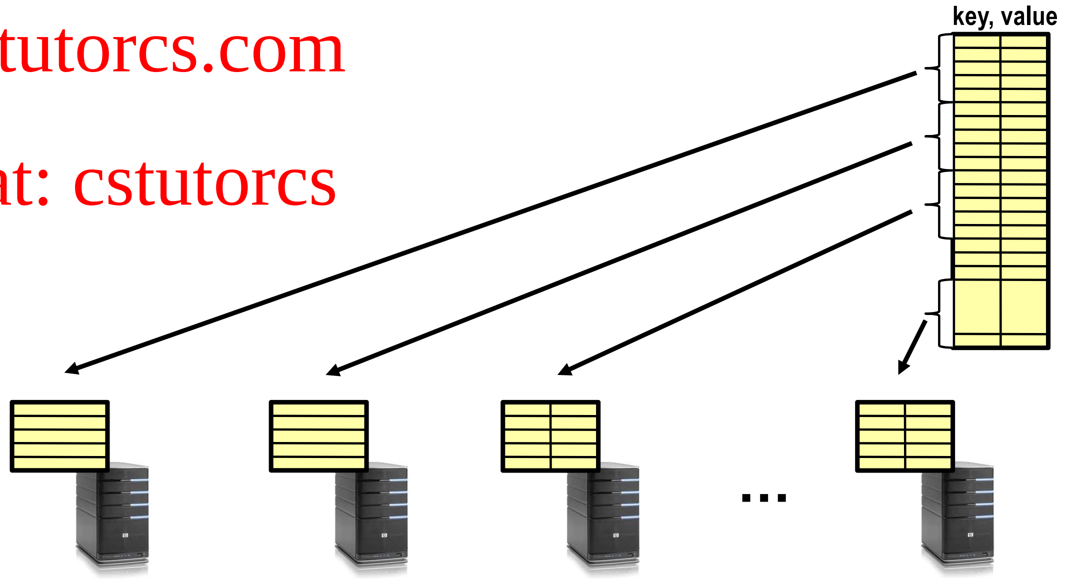
- (twitter.com): user ID --> user profile (e.g., posting history, photos, friends..)
- (amazon.com): item number --> information about it
- (kayak.com): flight number --> information about flight (e.g., availability)
- (yourbank.com): account number --> information about it

The key-value abstraction (cont’d)

- It’s a dictionary data-structure
- But distributed. (Too much data, you can maintain them in a single server)
- Sound familiar? Here the connection with DHTs!
- It is not surprising that key-value stores reuse many techniques from DHTs

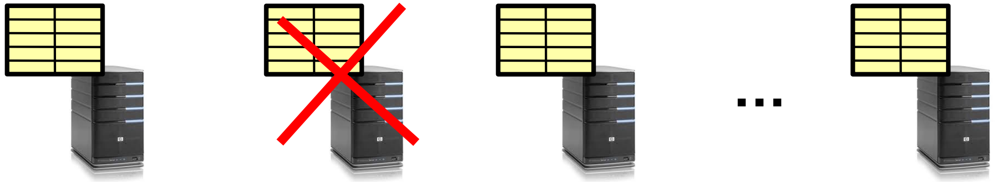
Too much data to maintain in a single server

Key Idea: partition set of key-values across many machines



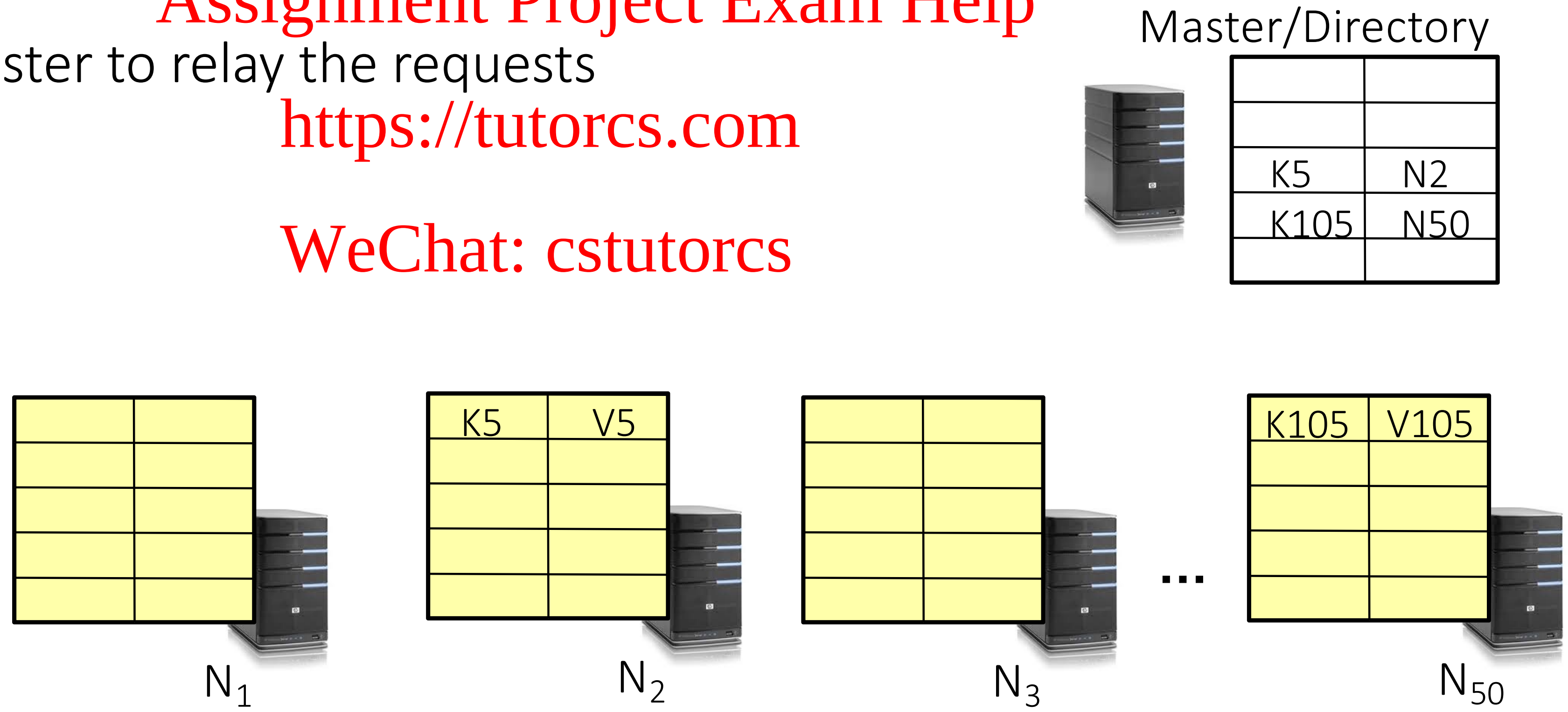
Challenges

- **Fault Tolerance:** handle machine failures without losing data and without degradation in performance
- **Scalability:**
  - Need to scale to thousands of machines
  - Need to allow easy addition of new machines
- **Consistency:** maintain data consistency in face of node failures and message losses



Directory-based architecture: recursive query

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys.**
- Having the master to relay the requests



Master/Directory

K5	N2
K105	N50

N_1

N_2

K5	V5

N_3

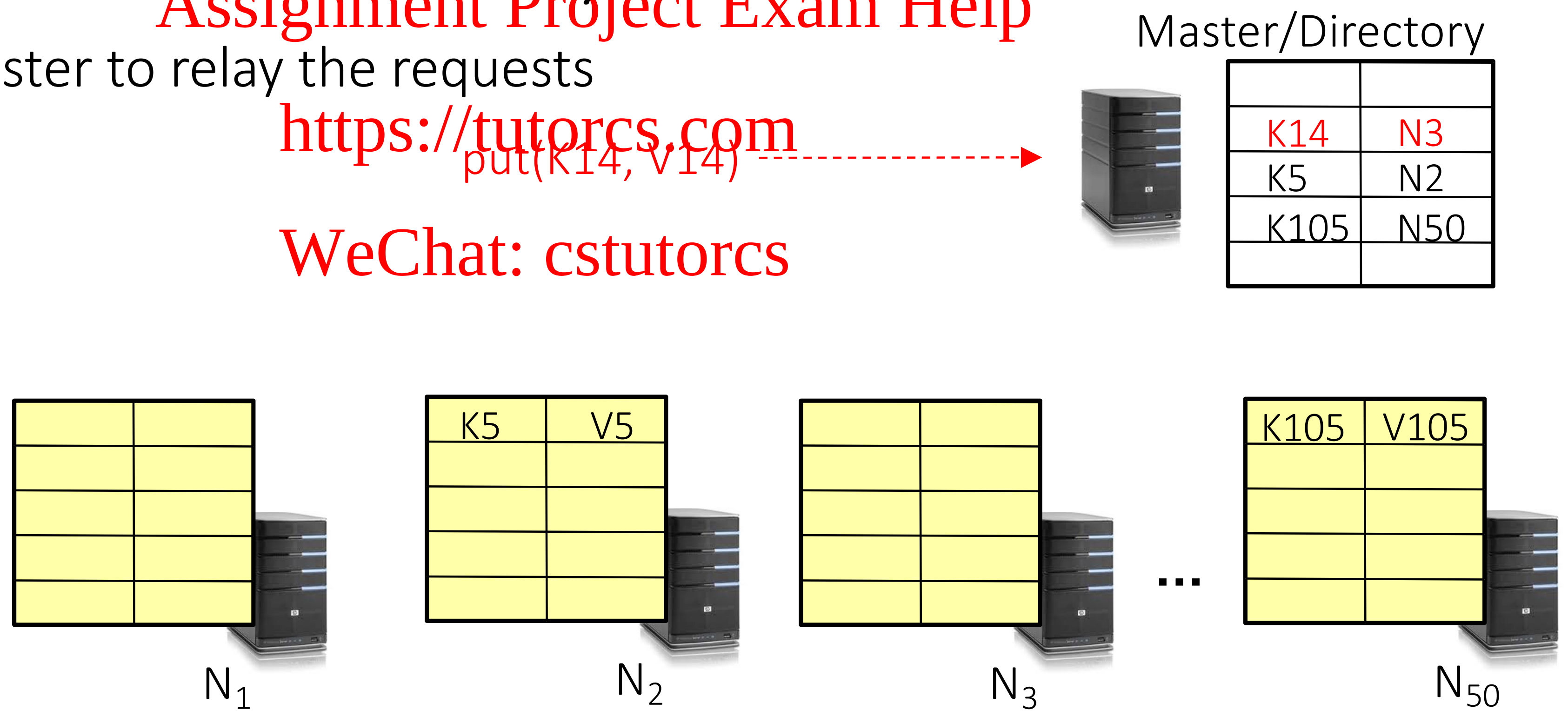
…

N_{50}

K105	V105

Directory-based architecture: recursive query

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys**.
- Having the master to relay the requests

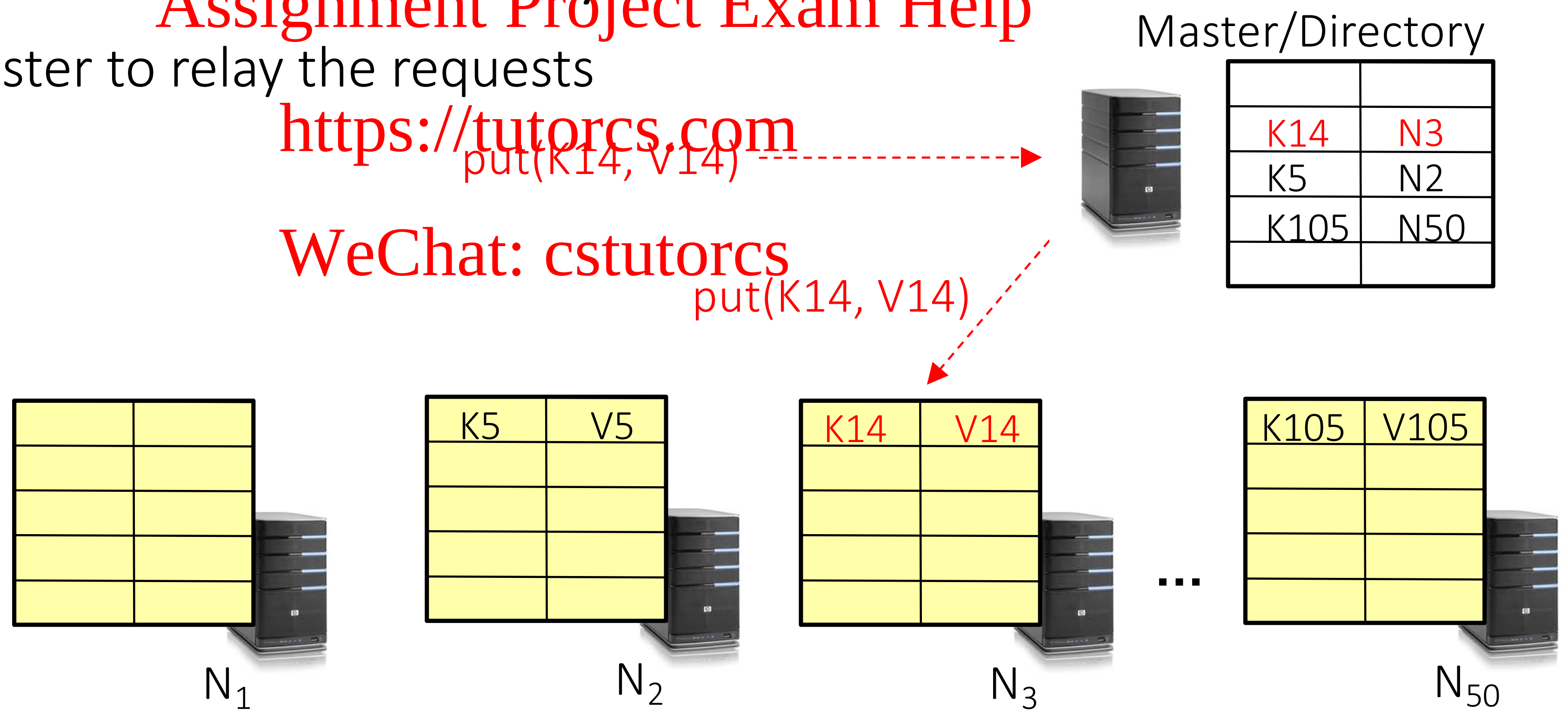


Master/Directory

K14	N3
K5	N2
K105	N50

Directory-based architecture: recursive query

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys.**
- Having the master to relay the requests



Master/Directory

K14	N3
K5	N2
K105	N50

N_1

N_2

K5	V5

N_3

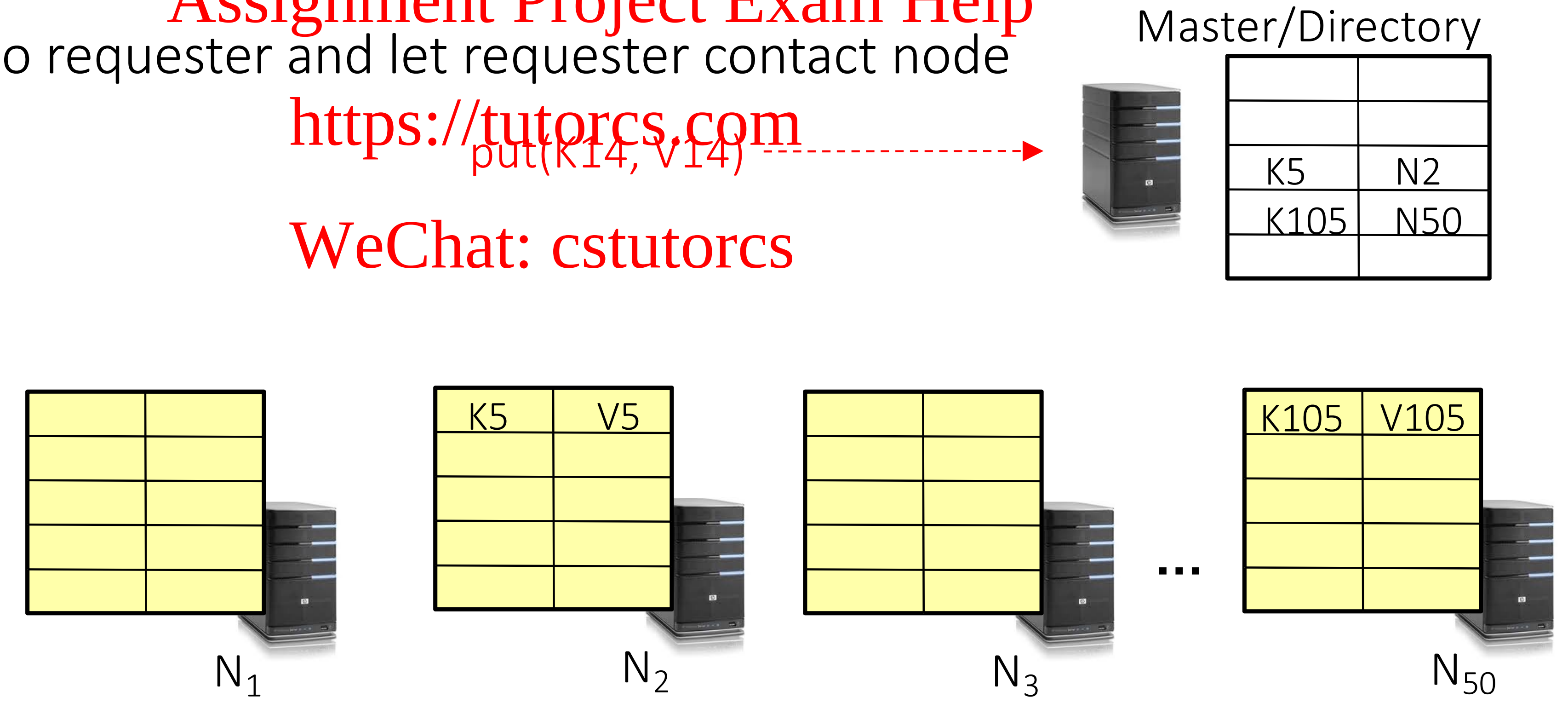
K14	V14

N_{50}

K105	V105

Directory-based architecture: iterative query

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys**.
- Return node to requester and let requester contact node



Master/Directory

K5	N2
K105	N50

N2

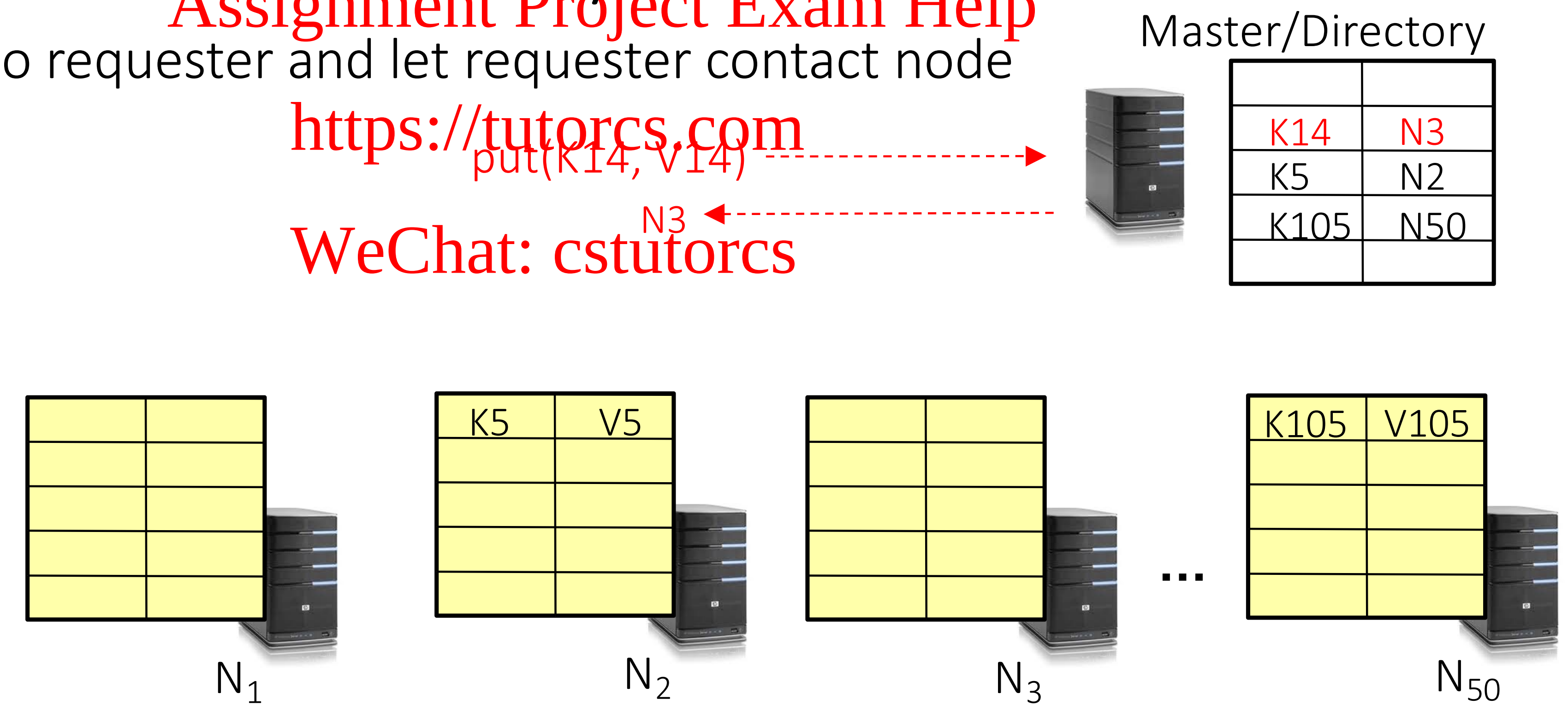
K5	V5

N50

K105	V105

Directory-based architecture: iterative query

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys**.
- Return node to requester and let requester contact node



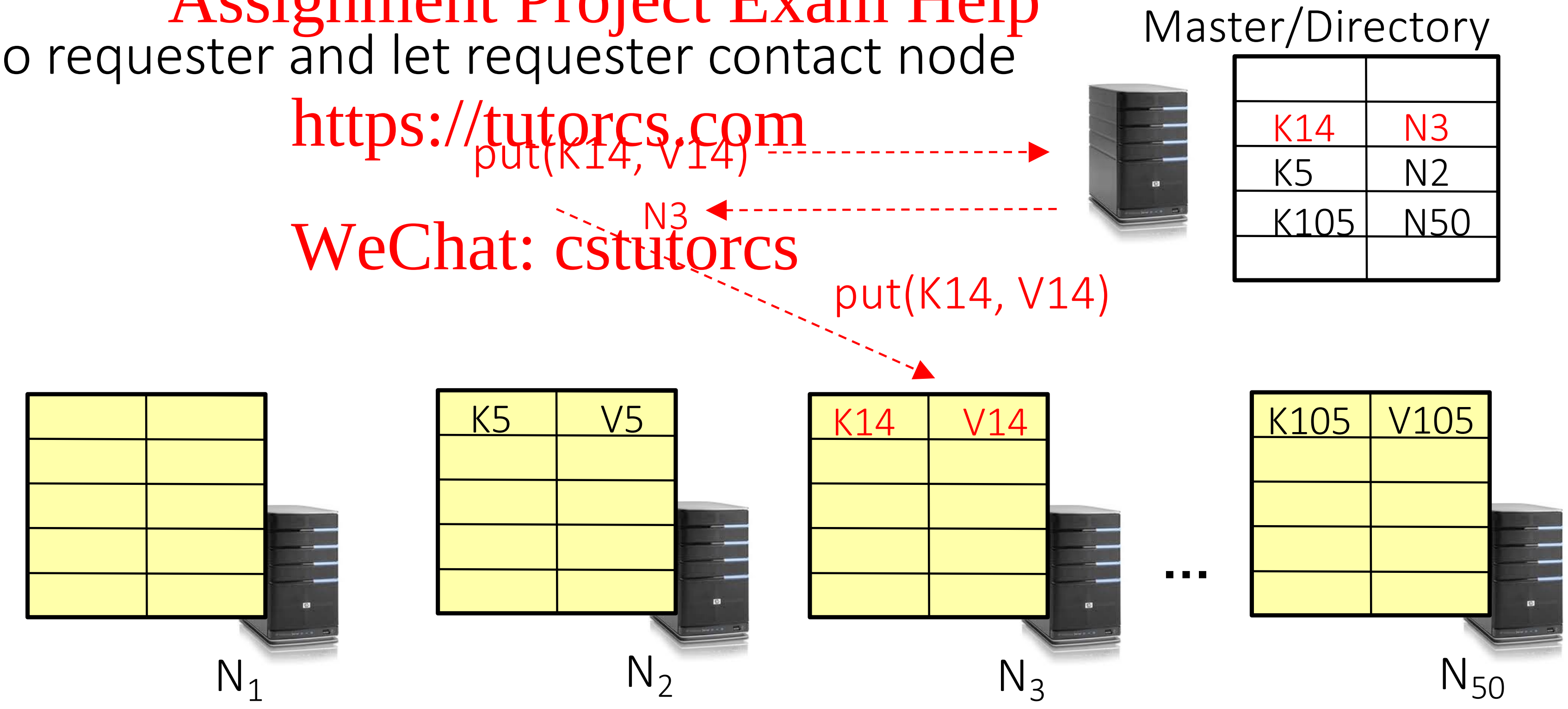
K14	N3
K5	N2
K105	N50

K5	V5

K105	V105

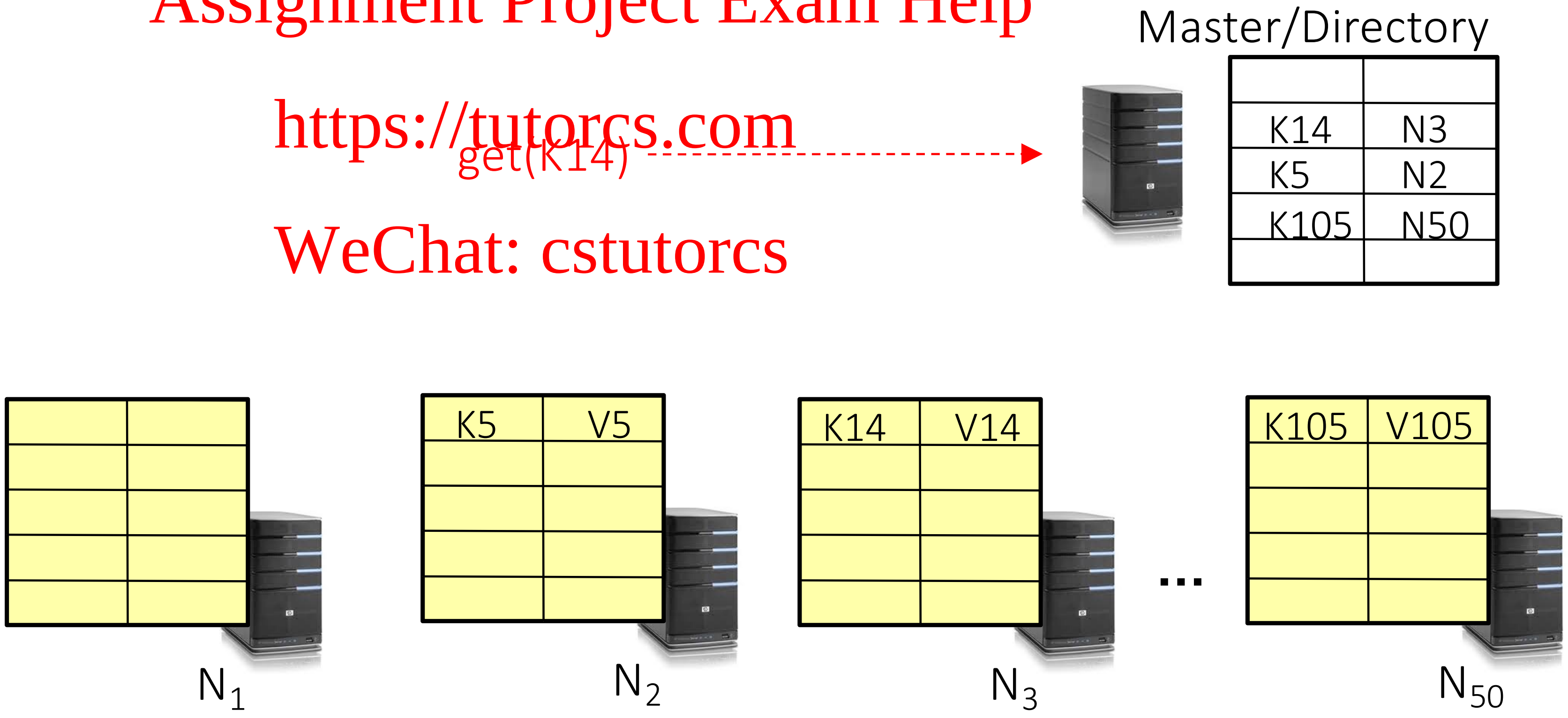
Directory-based architecture: iterative query

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys**.
- Return node to requester and let requester contact node



Directory-based architecture: iterative query

- The same solution applies also to retrieve a value…



Master/Directory

K14	N3
K5	N2
K105	N50

N1

N2

K5	V5

N3

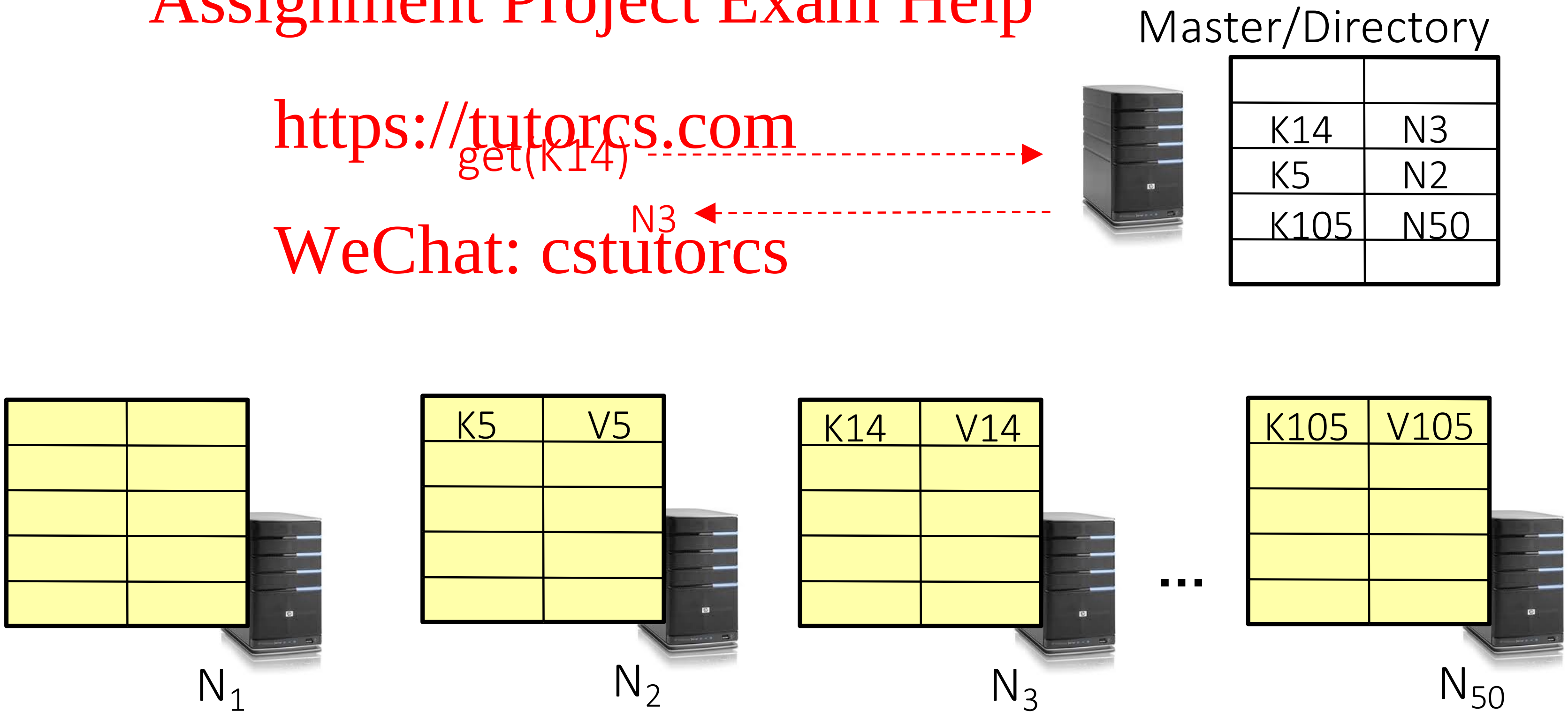
K14	V14

N50

K105	V105

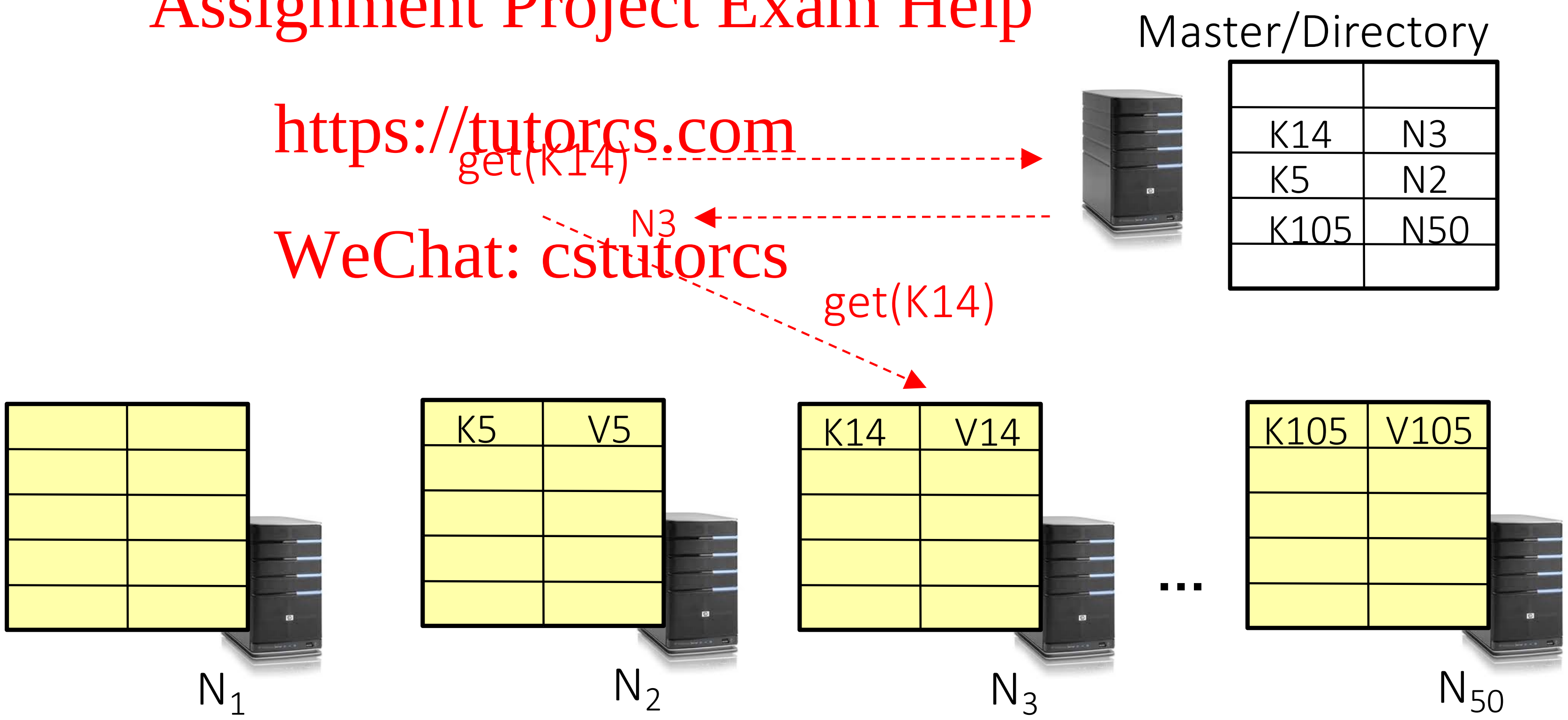
Directory-based architecture: iterative query

- The same solution applies also to retrieve a value…



Directory-based architecture: iterative query

- The same solution applies also to retrieve a value…



Master/Directory

K14	N3
K5	N2
K105	N50

N_1

N_2

K5	V5

N_3

K14	V14

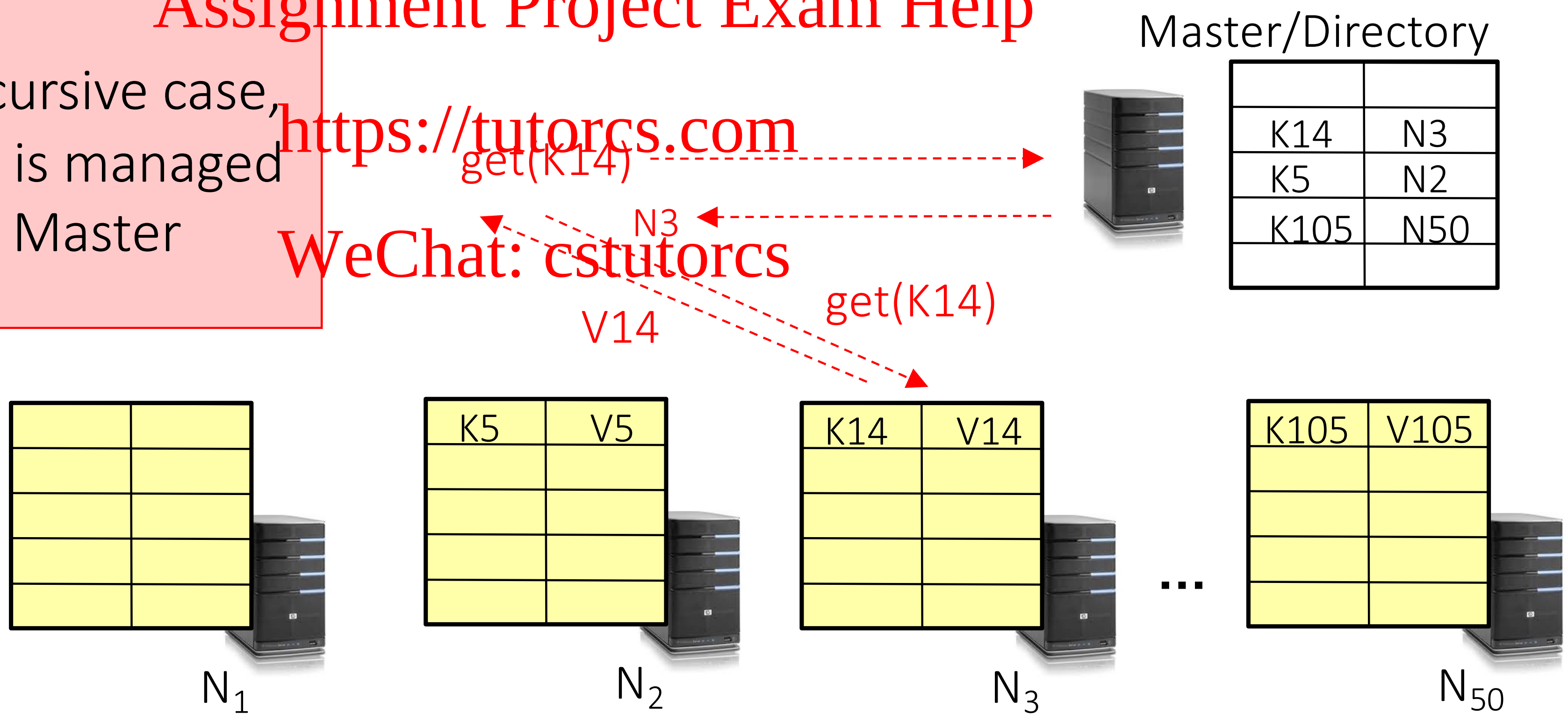
N_{50}

K105	V105

Directory-based architecture: iterative query

- The same solution applies also to retrieve a value…

For the recursive case, everything is managed by the Master



Master/Directory

K14	N3
K5	N2
K105	N50

N1

N2

K5	V5

N3

K14	V14

N50

K105	V105

Iterative vs recursive query

- Recursive Query (Master in charge):
  - Advantages:
    - Faster, as typically master/directory closer to nodes
    - Easier to maintain consistency, as master/directory can serialize puts()/gets()
  - Disadvantages: scalability bottleneck, as all “Values” go through master

- Iterative Query
  - Advantages: more scalable
  - Disadvantages: slower, harder to enforce data consistency

Key questions

- put(key, value): where do you store a new (key, value) tuple?
- get(key): where is the value associated with a given "key" stored?

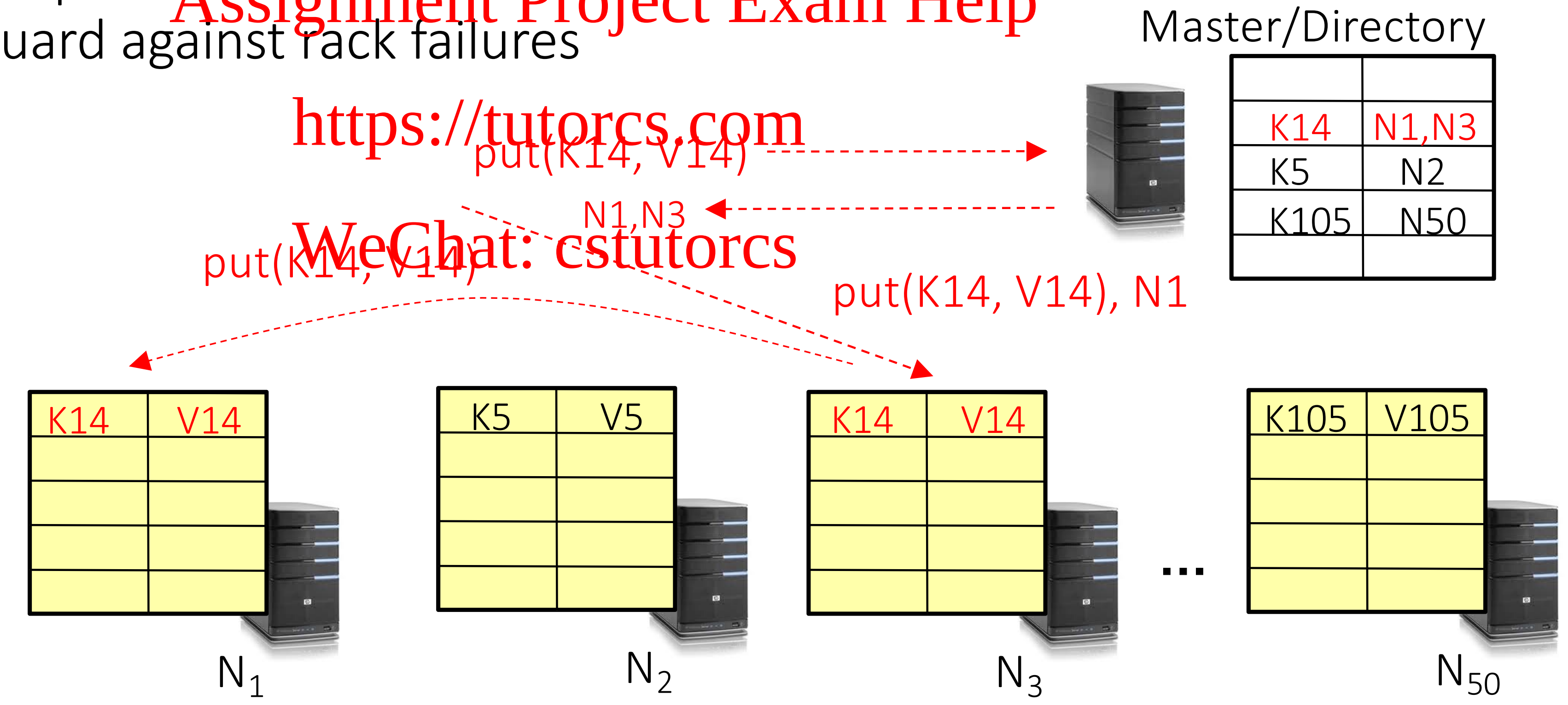
- …do the above while providing
  - Fault Tolerance
  - Scalability
  - Consistency

Key questions

- put(key, value): where do you store a new (key, value) tuple?
- get(key): where is the value associated with a given "key" stored?
- ...do the above while providing
  - Fault Tolerance
  - Scalability
  - Consistency

Fault tolerance: recursive

- Replicate value on several nodes
- Usually, place replicas on different racks in a datacenter to guard against rack failures



Master/Directory

K14	N1,N3
K5	N2
K105	N50

N1

K14	V14

N2

K5	V5

N3

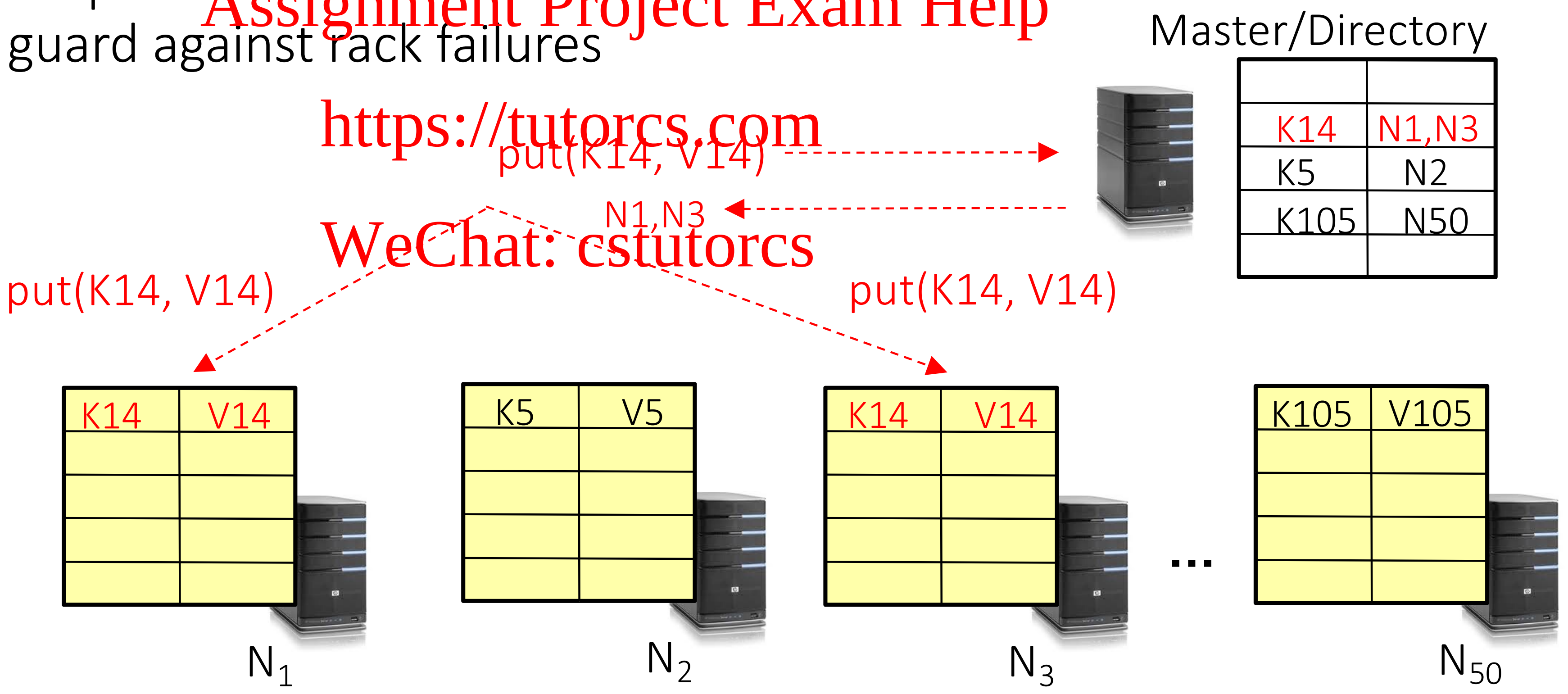
K14	V14

N50

K105	V105

Fault tolerance: iterative

- Replicate value on several nodes
- Usually, place replicas on different racks in a datacenter to guard against rack failures



Master/Directory

K14	N1,N3
K5	N2
K105	N50

N_1

K14	V14

N_2

K5	V5

N_3

K14	V14

N_{50}

K105	V105

Replication challenges

- Need to make sure that a value is replicated correctly

- How do you know a value has been replicated on every node?
  - Wait for acknowledgements from every node

- What happens if a node fails during replication?
  - Pick another node and try again

- What happens if a node is slow?
  - Slow down the entire put()? Pick another node?

Replication challenges

- Need to make sure that a value is replicated correctly
- How do you know a value has been replicated on every node?
  - Wait for acknowledgements from every node
- What happens if a node fails during replication?
  - Pick another node and try again
- What happens if a node is slow?
  - Slow down the entire put()? Pick another node?

In general, with multiple replicas:

Slow puts and fast gets

Key questions

- put(key, value): where do you store a new (key, value) tuple?
- get(key): where is the value associated with a given “key” stored?

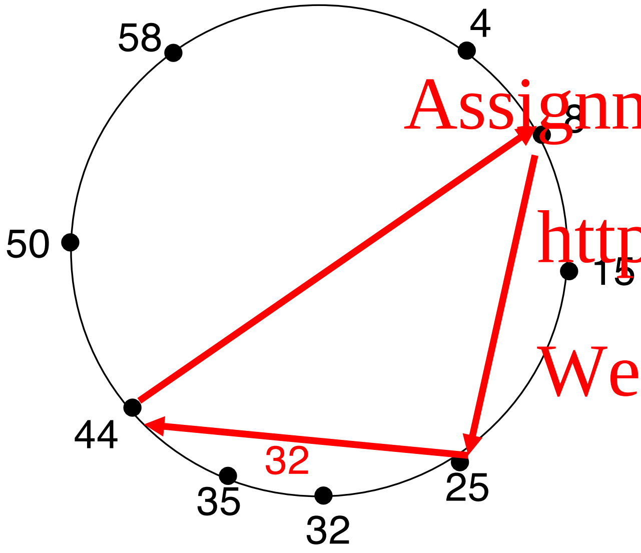
- …do the above while providing
  - Fault Tolerance
  - Scalability
  - Consistency

Scalability

- **Storage:** use more nodes

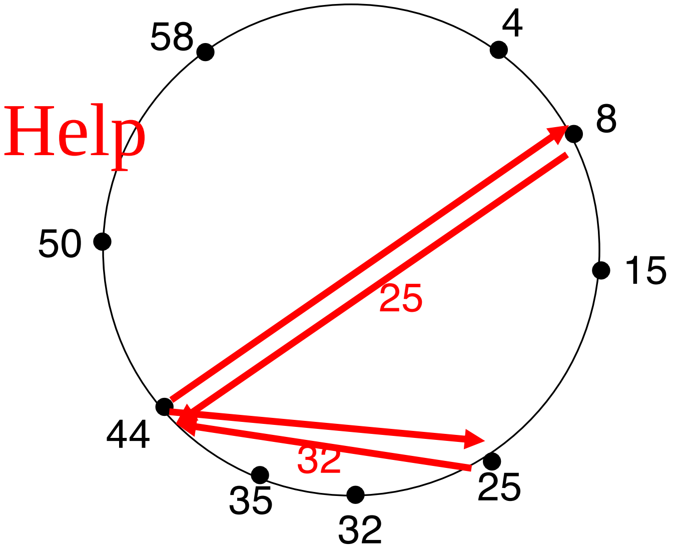
- **Request throughput:**
  - Can serve requests from all nodes on which a value is stored in parallel
  - Master can replicate a popular value on more nodes
- **Master/directory scalability:**
  - Replicate it
  - Partition it, so different keys are served by different masters/directories (do you remember Chord? ☺)

Scalability with Chord



Recursively example:

node 44 issue query(31)



Iteratively example:

node 44 issue query(31)

Scalability: load balancing

- Directory keeps track of the storage availability at each node
    - Preferentially insert new values on nodes with more storage available

- What happens when a new node is added?

    - Move values from the heavy loaded nodes to the new node

- What happens when a node fails?
    - Need to replicate values from fail node to other nodes

Key questions

- put(key, value): where do you store a new (key, value) tuple?
- get(key): where is the value associated with a given “key” stored?

- …do the above while providing
  - Fault Tolerance
  - Scalability
  - Consistency

Consistency

- How close does a distributed system emulate a single machine in terms of read and write semantics?

Consistency

- How close does a distributed system emulate a single machine in terms of read and write semantics?

- Q: Assume **put(K14, V14’)** and **put(K14, V14’’)** are concurrent, what value ends up being stored?

Consistency

- How close does a distributed system emulate a single machine in terms of read and write semantics?

- Q: Assume **put(K14, V14’)** and **put(K14, V14’’)** are concurrent, what value ends up being stored?
- A: assuming **put()** is atomic, then either V14’ or V14’’, right?

Consistency

- How close does a distributed system emulate a single machine in terms of read and write semantics?

- Q: Assume a client calls **put(K14, V14)** and then **get(K14)**, what is the result returned by **get()**?

Consistency

- How close does a distributed system emulate a single machine in terms of read and write semantics?

- **Q:** Assume a client calls **put(K14, V14)** and then **get(K14)**, what is the result returned by **get()**?
- **A:** It should be V14, right?

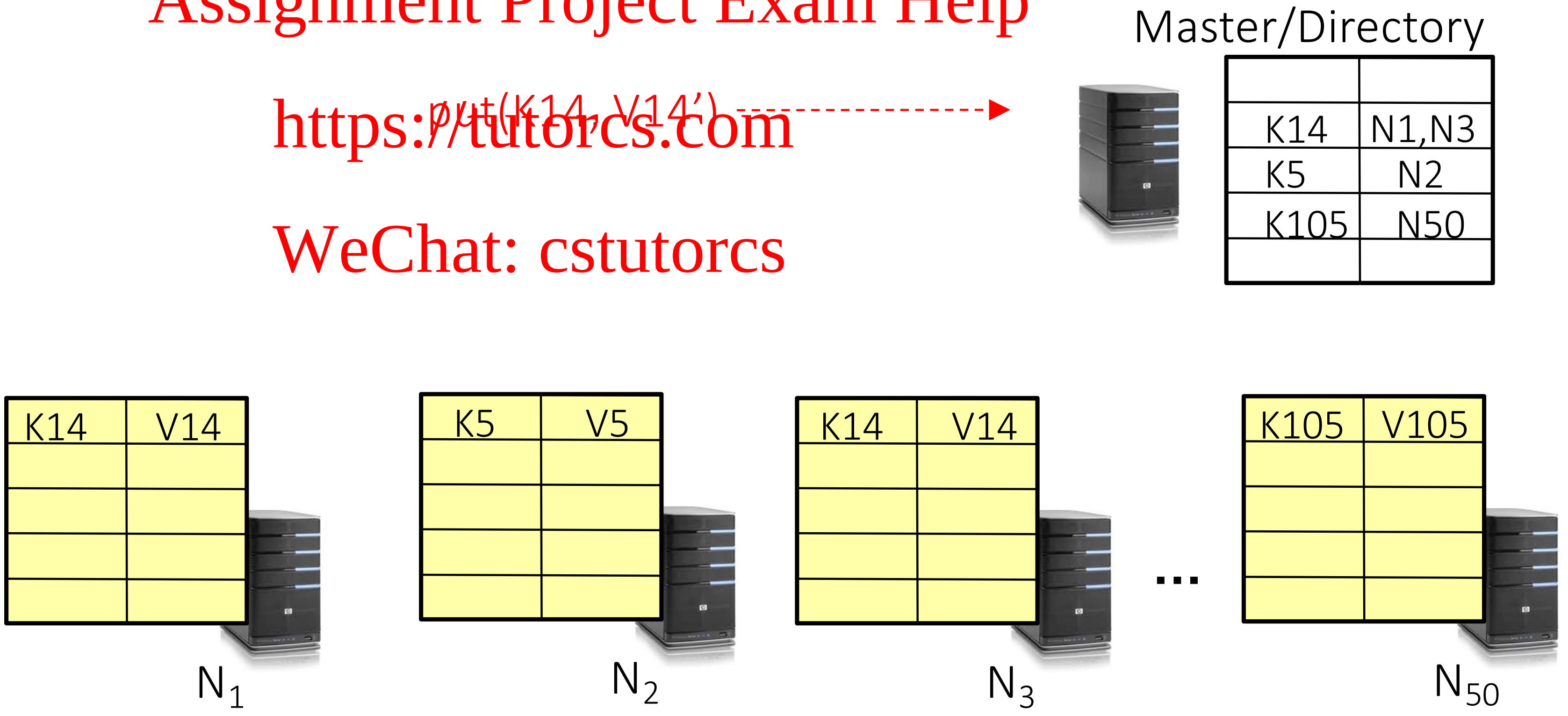
Consistency

- How close does a distributed system emulate a single machine in terms of read and write semantics?

Above semantics, not trivial to achieve in distributed systems!!!!

Concurrent writes

- If concurrent updates (i.e., puts to same key) may need to make sure that updates happen in the same order



K14	N1,N3
K5	N2
K105	N50

K14	V14

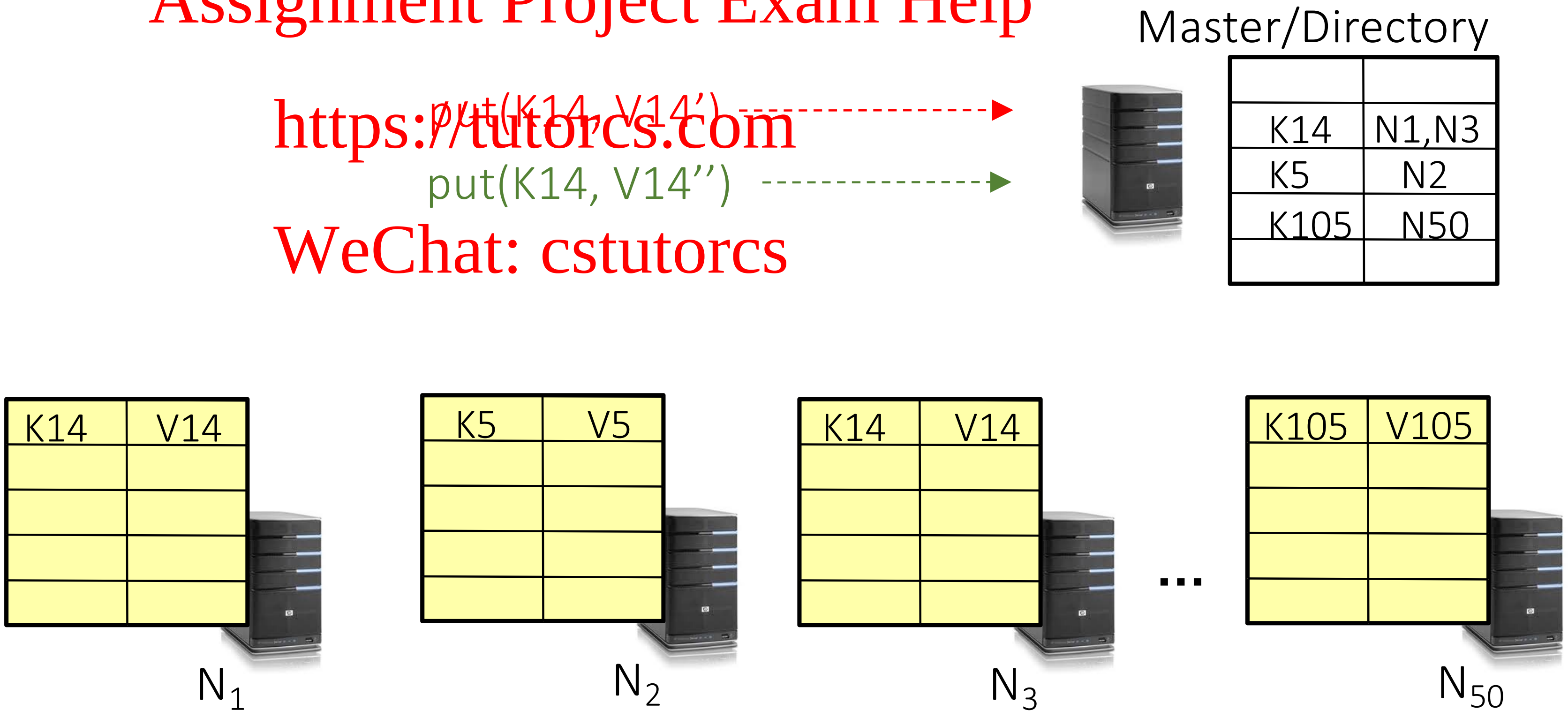
K5	V5

K14	V14

K105	V105

Concurrent writes

- If concurrent updates (i.e., puts to same key) may need to make sure that updates happen in the same order



Master/Directory

K14	N1,N3
K5	N2
K105	N50

N1

K14	V14

N2

K5	V5

N3

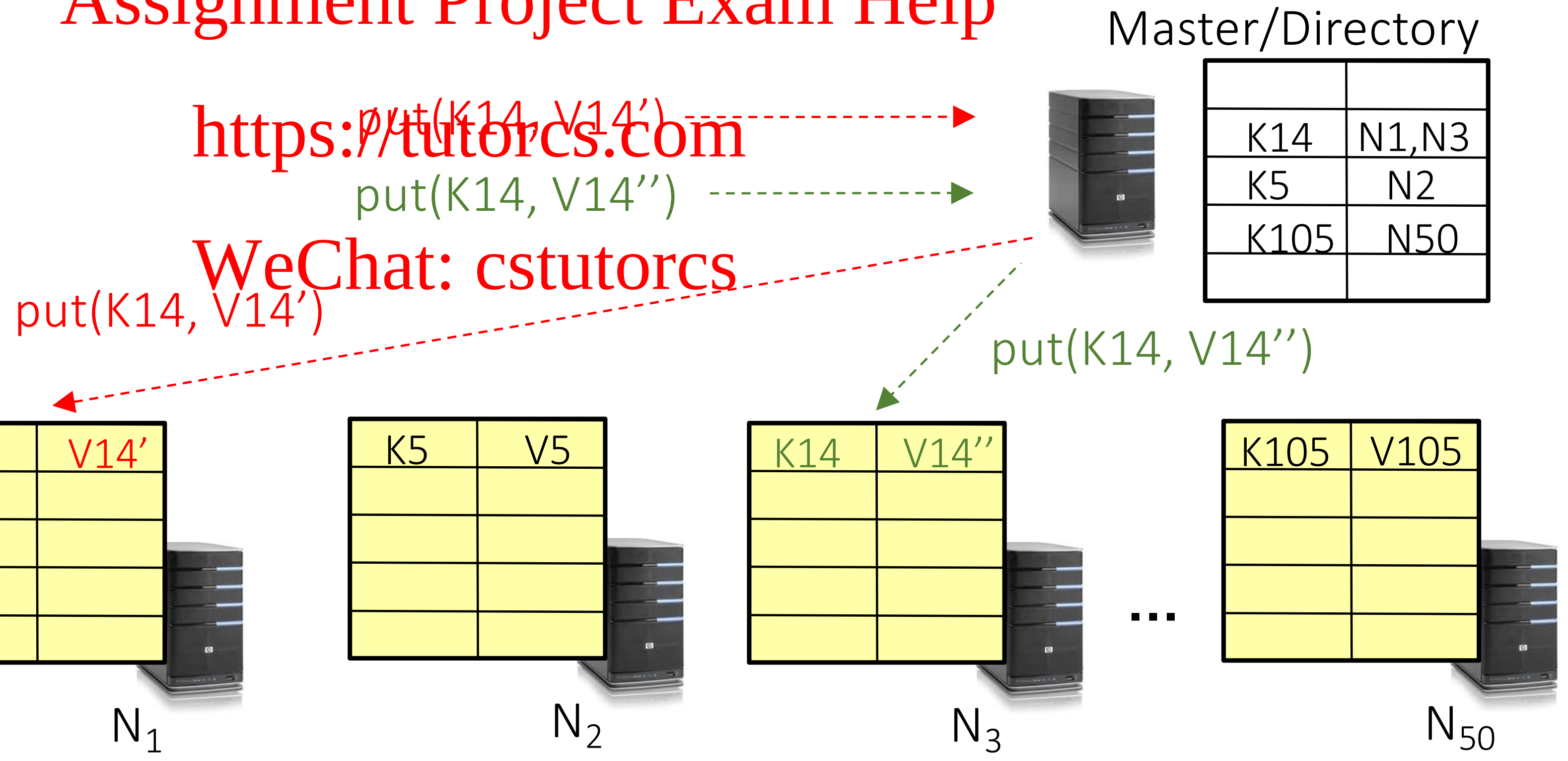
K14	V14

N50

K105	V105

Concurrent writes

- If concurrent updates (i.e., puts to same key) may need to make sure that updates happen in the same order



K14	N1,N3
K5	N2
K105	N50

K14	V14'

K5	V5

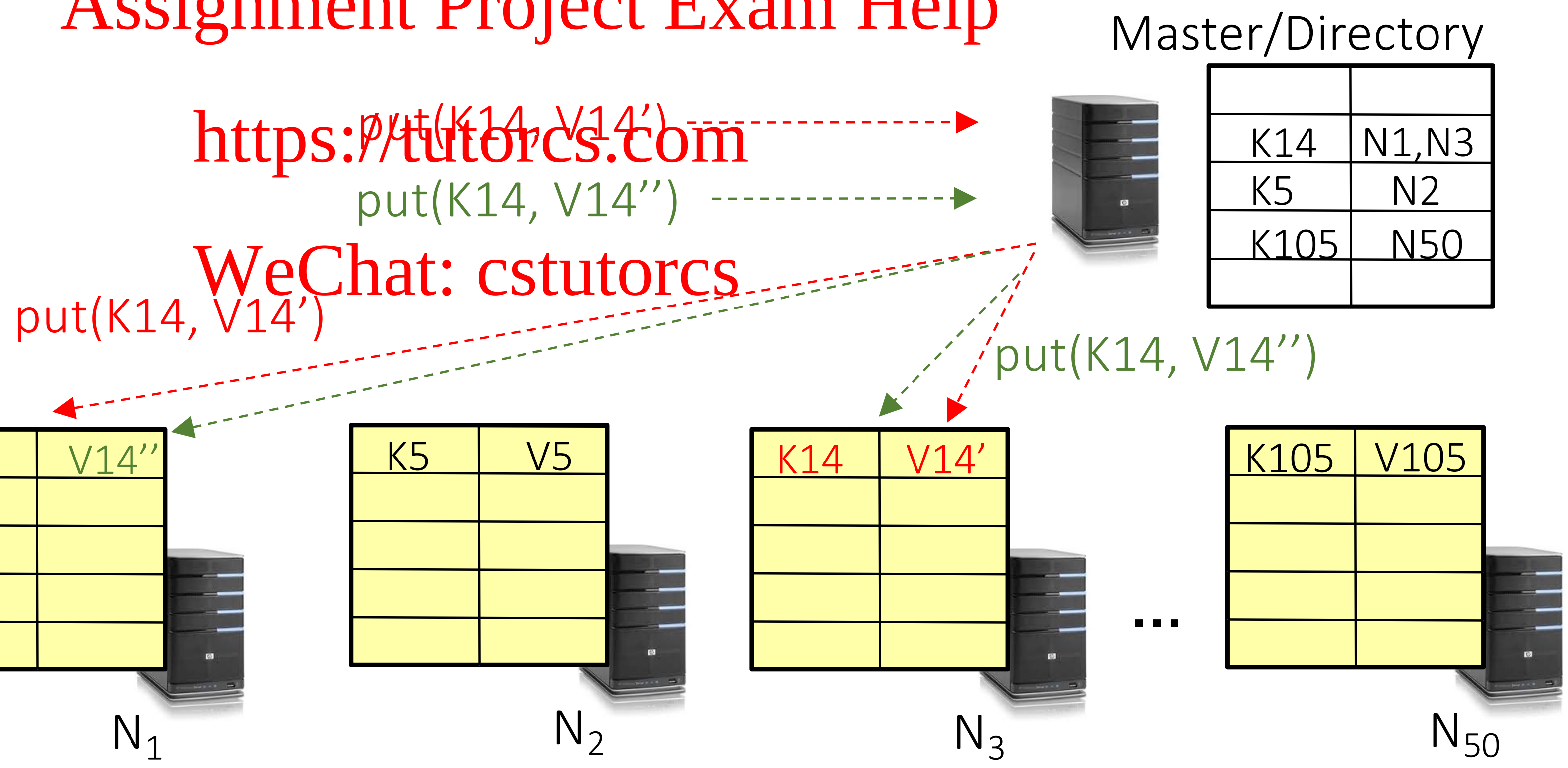
K14	V14''

K105	V105

Concurrent writes

- put(K14, V14’) and put(K14, V14’’) reach N1 and N3 in reverse order
- What does get(K14) return?
  - Undefined!

- If concurrent updates (i.e., puts to same key) may need to make sure that updates happen in the same order



K14	N1,N3
K5	N2
K105	N50

K14	V14’’

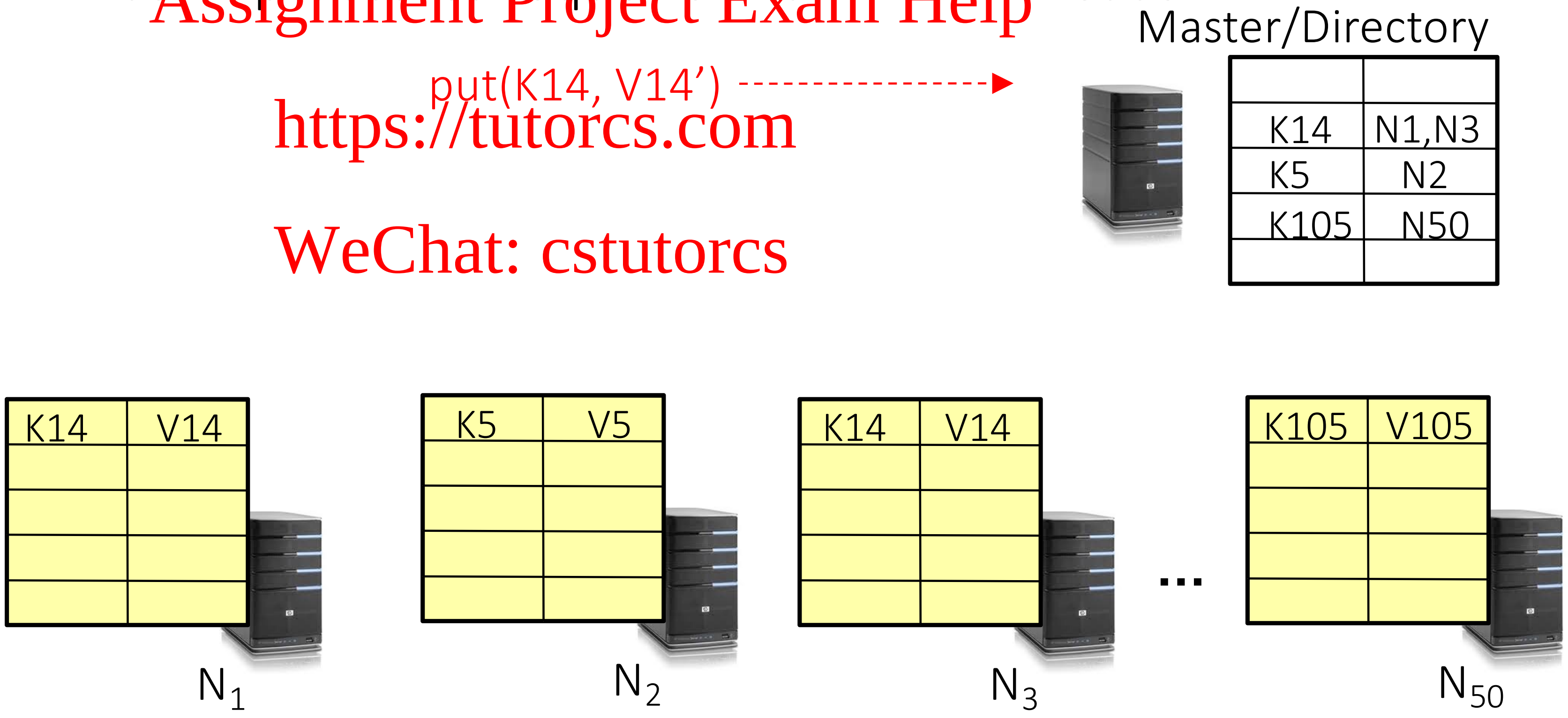
K5	V5

K14	V14’

K105	V105

Read after write

- Read not guaranteed to return value of latest write
  - Can happen if Master processes requests in different threads



Master/Directory

K14	N1,N3
K5	N2
K105	N50

N_1

K14	V14

N_2

K5	V5

N_3

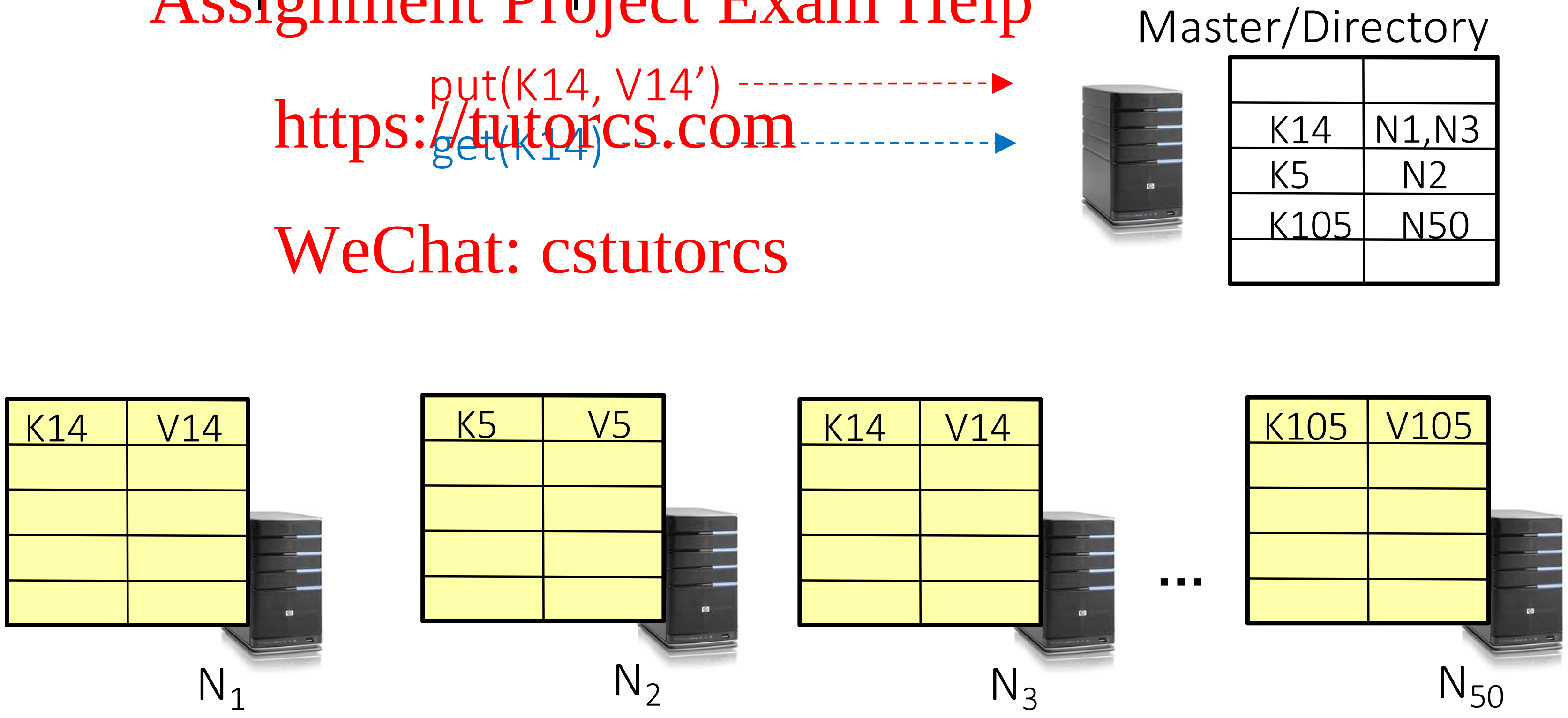
K14	V14

N_{50}

K105	V105

Read after write

- Read not guaranteed to return value of latest write
  - Can happen if Master processes requests in different threads



K14	N1,N3
K5	N2
K105	N50

K14	V14

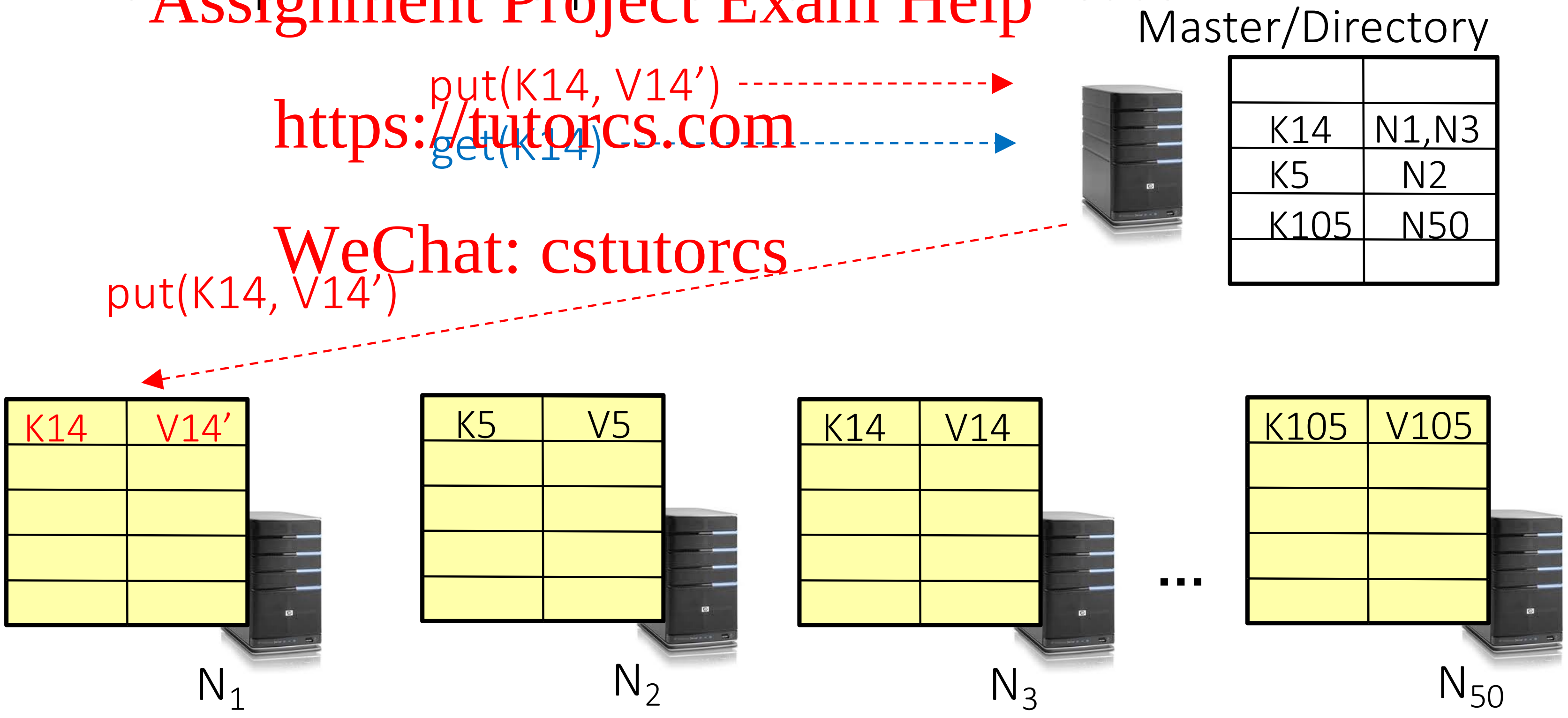
K5	V5

K14	V14

K105	V105

Read after write

- Read not guaranteed to return value of latest write
  - Can happen if Master processes requests in different threads



Master/Directory

K14	N1,N3
K5	N2
K105	N50

N1

K14	V14'

N2

K5	V5

N3

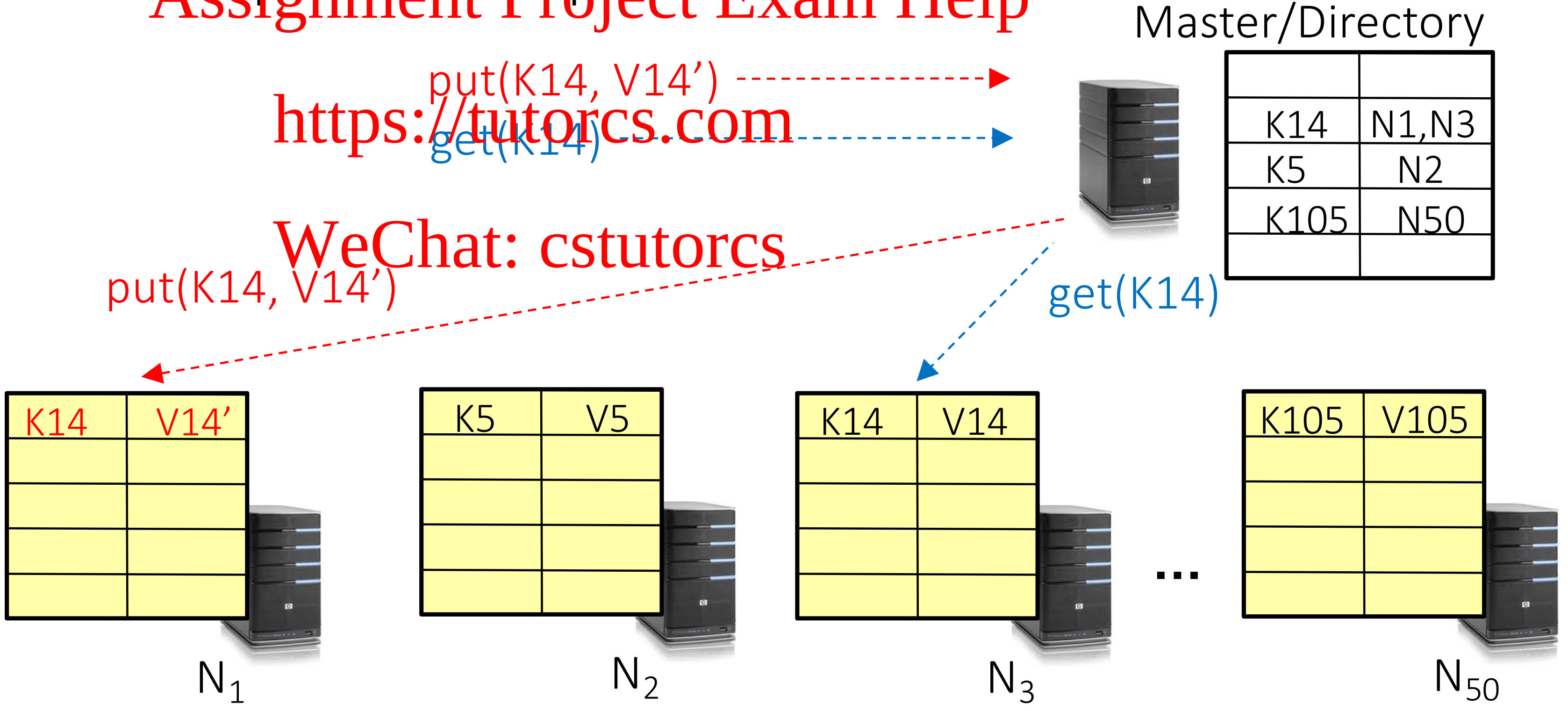
K14	V14

N50

K105	V105

Read after write

- Read not guaranteed to return value of latest write
  - Can happen if Master processes requests in different threads



K14	N1,N3
K5	N2
K105	N50

K14	V14'

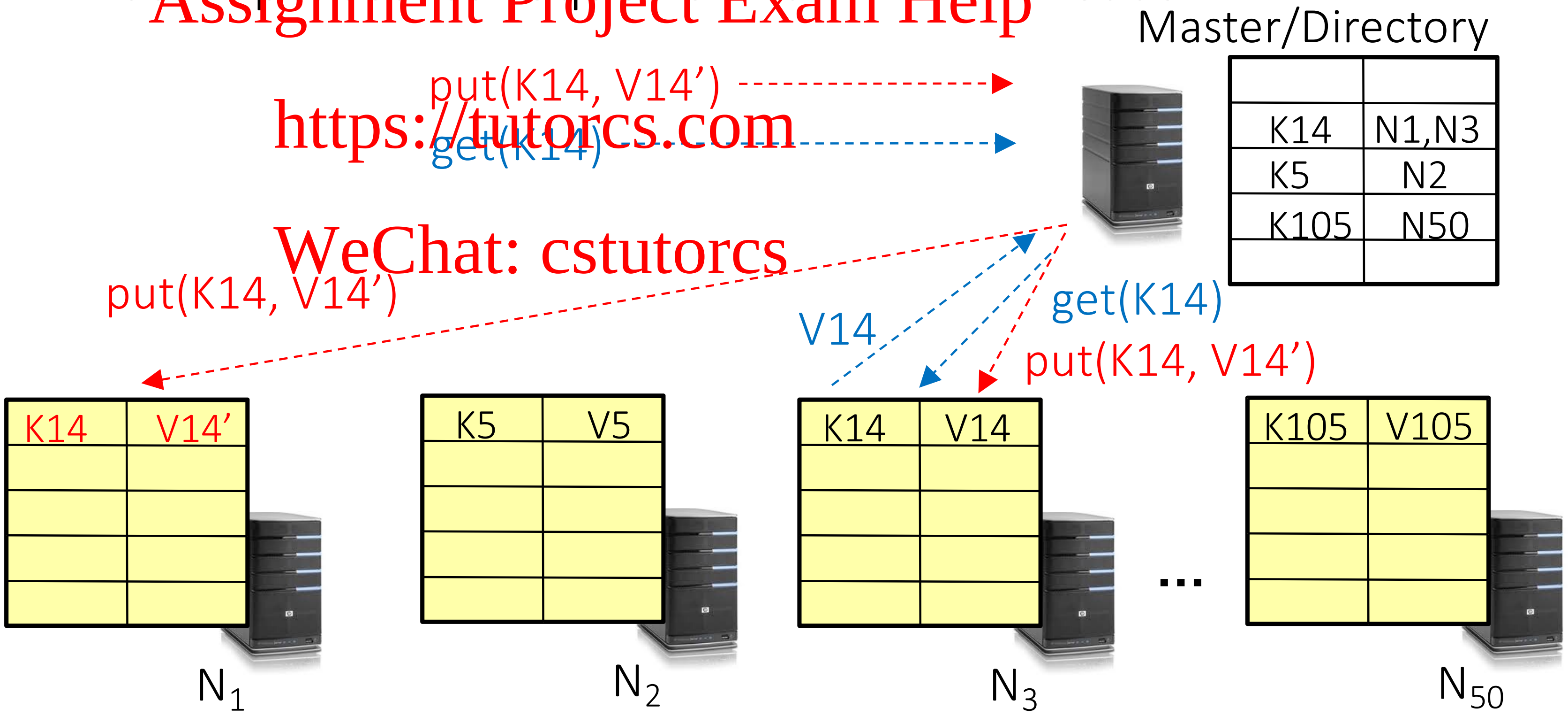
K5	V5

K14	V14

K105	V105

Read after write

- Read not guaranteed to return value of latest write
  - Can happen if Master processes requests in different threads



K14	N1,N3
K5	N2
K105	N50

K14	V14'

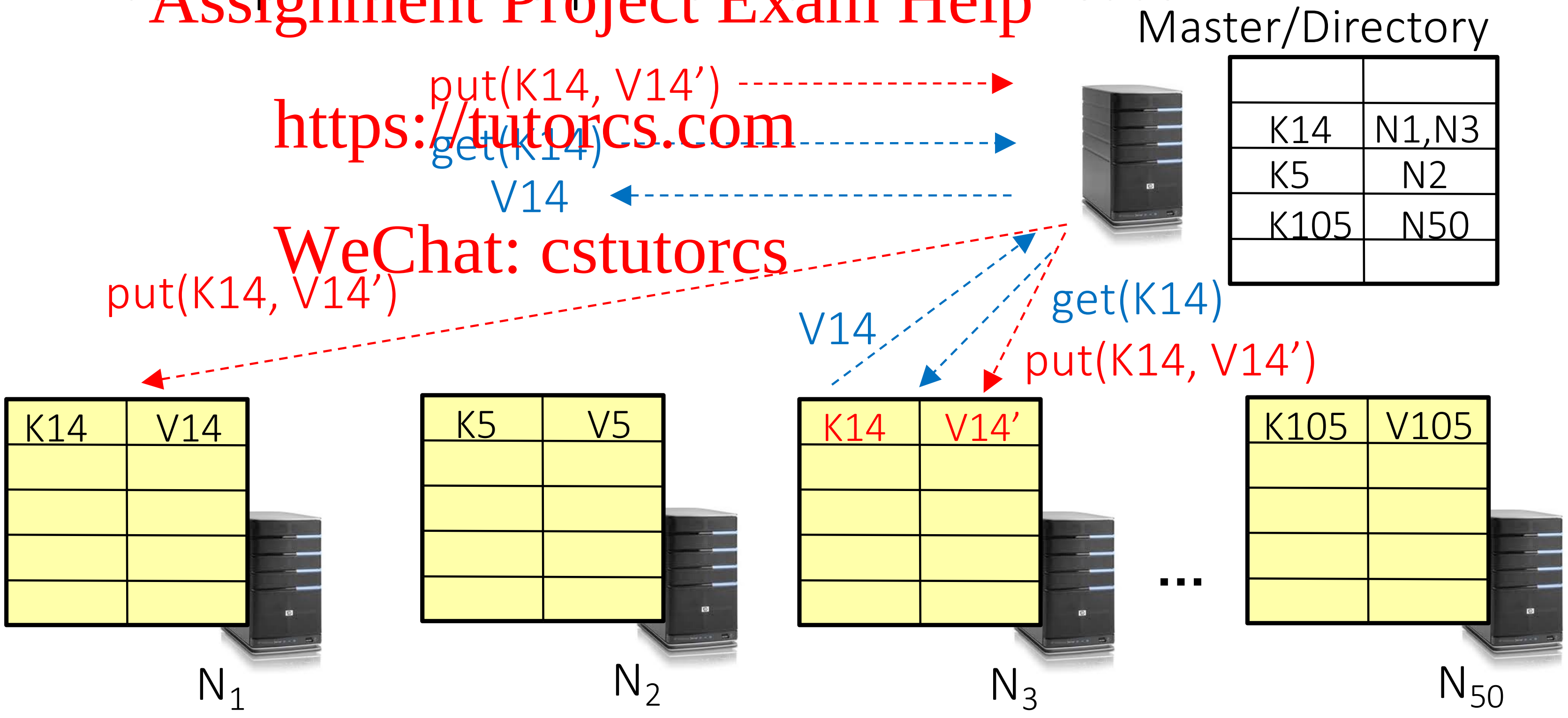
K5	V5

K14	V14

K105	V105

Read after write

- Read not guaranteed to return value of latest write
    - Can happen if Master processes requests in different threads



K14	N1,N3
K5	N2
K105	N50

K14	V14

K5	V5

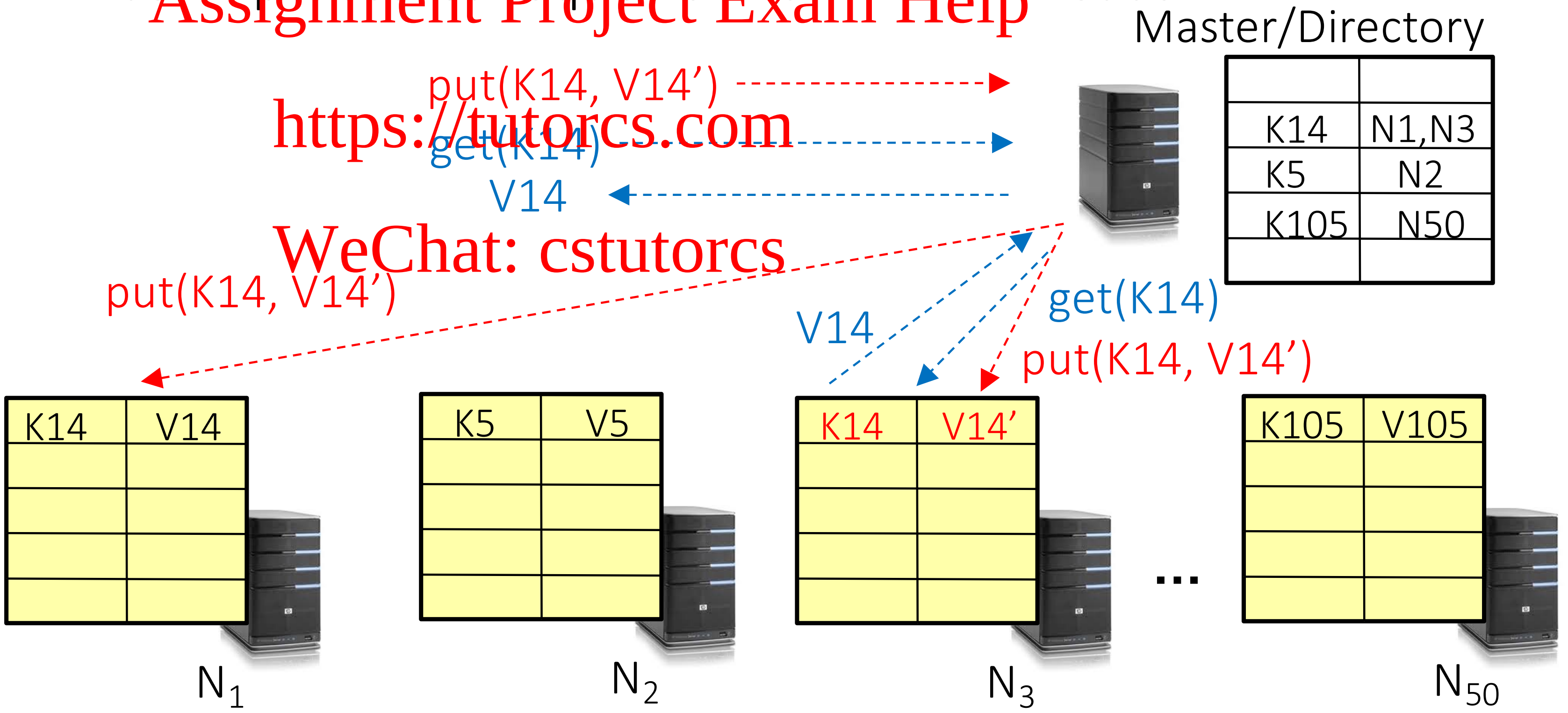
K14	V14'

K105	V105

Read after write

- get(K14) happens right after put(K14, V14’)
- get(K14) reaches N3 before put(K14, V14’)!

- Read not guaranteed to return value of latest write
  - Can happen if Master processes requests in different threads



K14	N1,N3
K5	N2
K105	N50

K14	V14

K5	V5

K14	V14’

K105	V105

The return of an old friend

- Does this remind you something? ☺

The return of an old friend

- Does this remind you something? ☺

- Yes, all the consistency models/protocols we have seen apply also here!

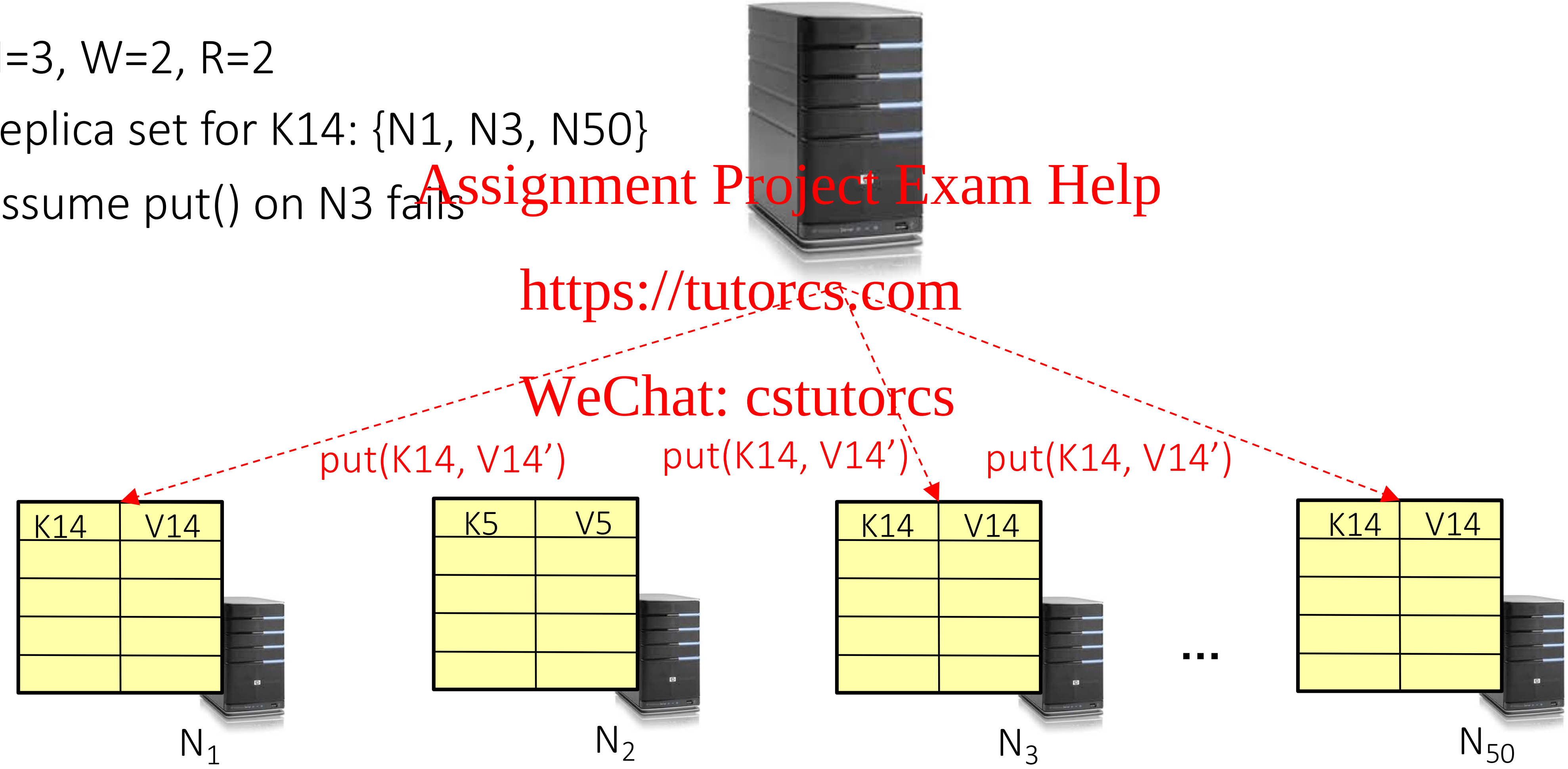
Quorum consensus

- Define a replica set of size N
  - **put()** waits for acks from at least W replicas
  - **get()** waits for responses from at least R replicas

- Why may you use W+R > N+1?

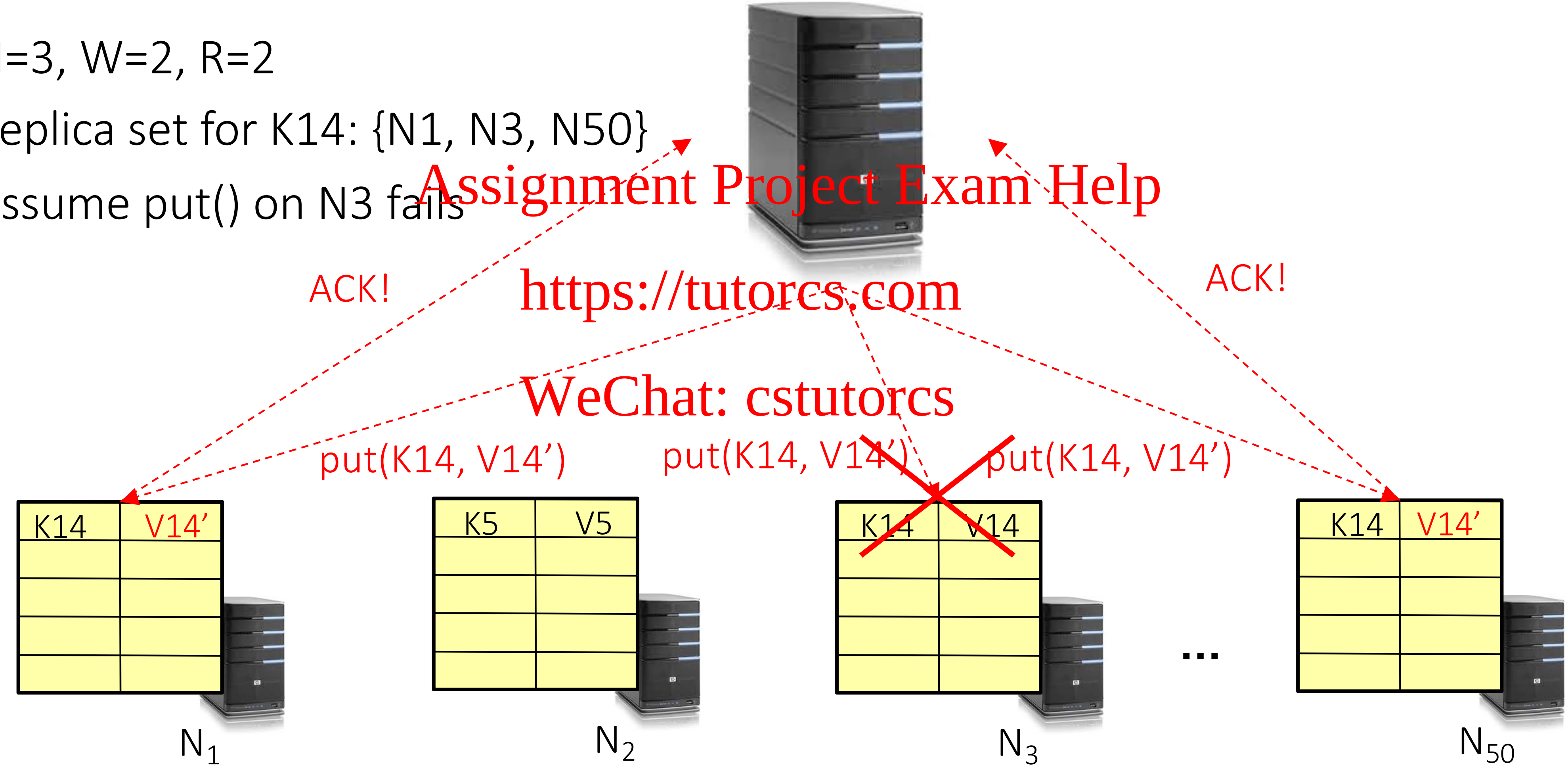
Quorum consensus

- N=3, W=2, R=2
- Replica set for K14: {N1, N3, N50}
- Assume put() on N3 fails



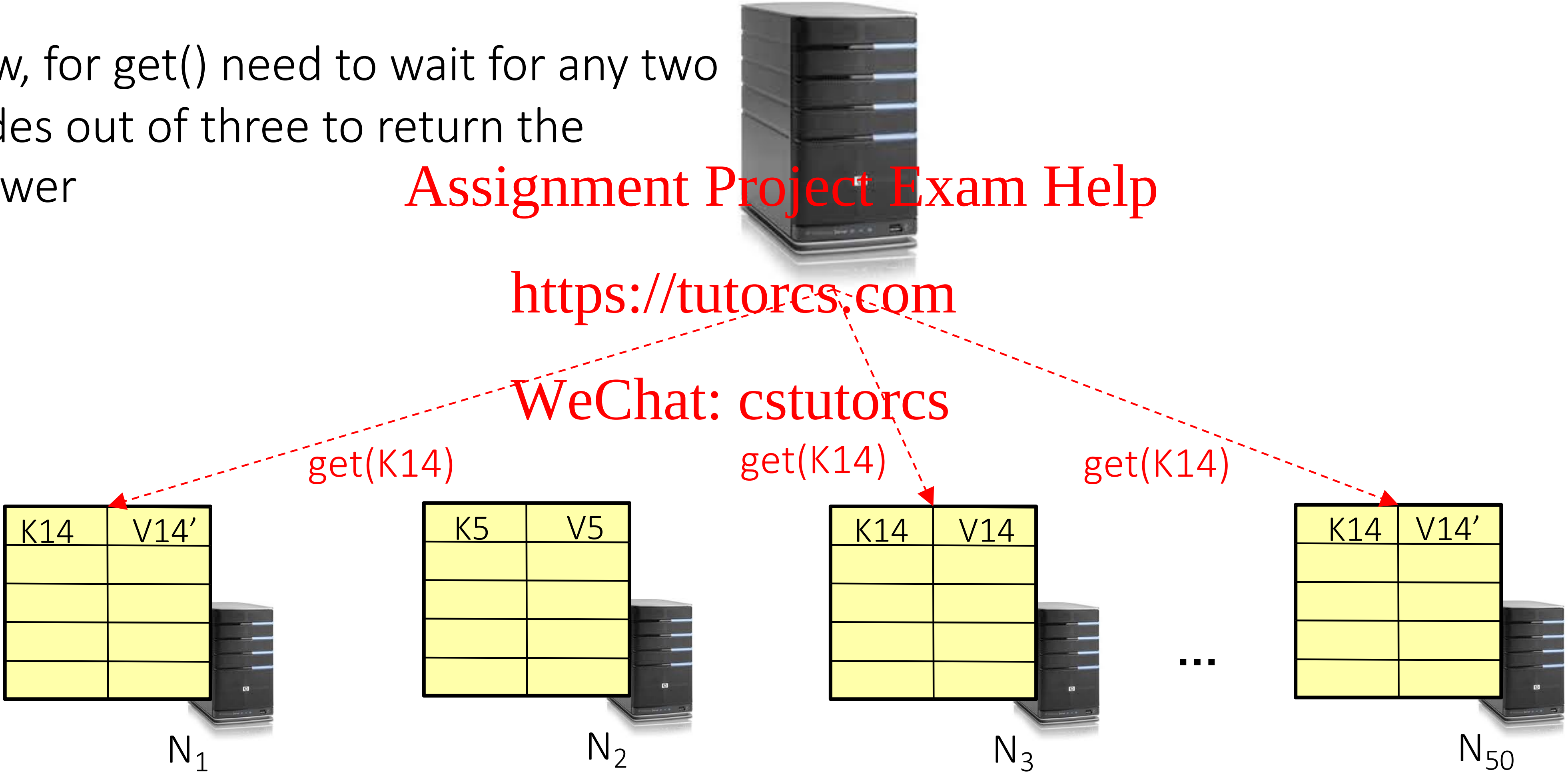
Quorum consensus

- N=3, W=2, R=2
- Replica set for K14: {N1, N3, N50}
- Assume put() on N3 fails



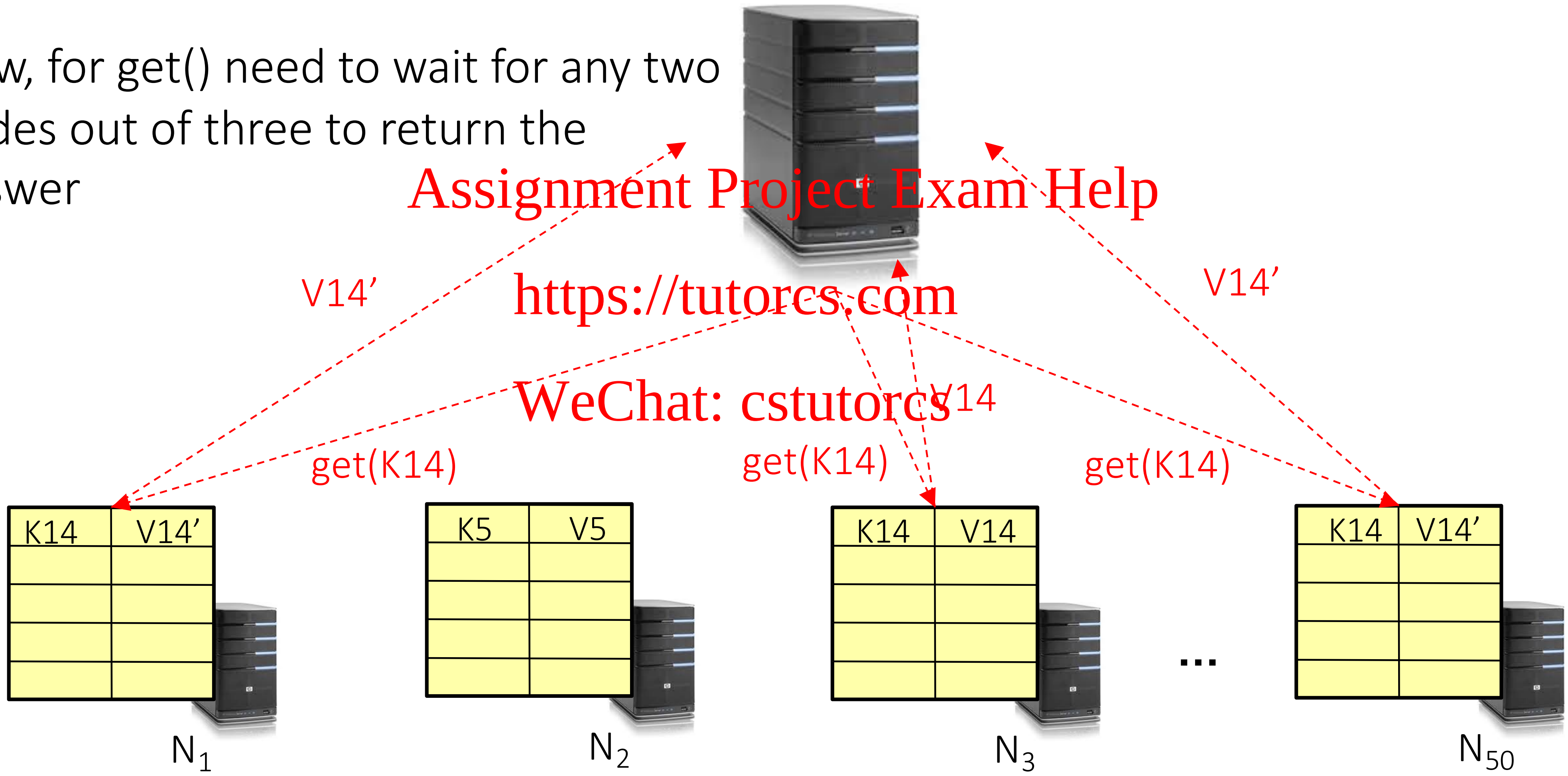
Quorum consensus

Now, for get() need to wait for any two nodes out of three to return the answer



Quorum consensus

Now, for get() need to wait for any two nodes out of three to return the answer



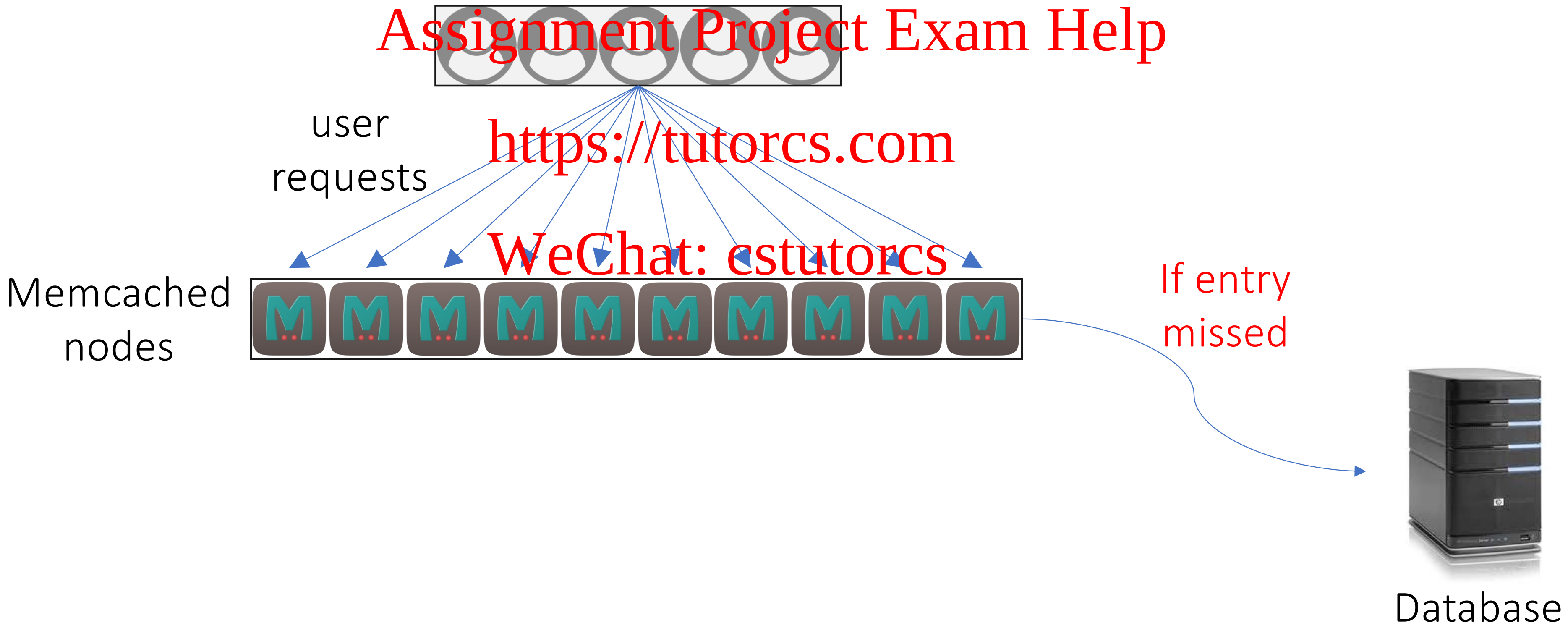
Memcached: a Key-Value Store example

- Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering

- Memcached's APIs provide a very large hash table distributed across multiple machines.
  - If table is full: subsequent inserts cause older data to be purged in least recently used (LRU) order.

- Applications using Memcached typically layer requests and additions into RAM before falling back on a slower backing store, such as a database.

Memcached in a figure



Memcached: when to use it?

- Often used for small objects

- Anything what is more expensive to fetch from elsewhere, and has sufficient hitrate, can be placed in memcached
  - How often will object or data be used?
  - How expensive is it to generate the data?
  - What is the expected hitrate?

Memcached: trade-offs

- Why YES:
  1. to reduce the load on the database by caching data <u>BEFORE</u>
  2. improve the entire application response time (much faster hitting the RAM than the disk or the database)

- Why NO:
  1. Memcache is held in RAM. This is a finite resource.
  2. Adding complexity to a system just for complexities sake is a waste. If the system can respond within the requirements without it - leave it alone

Memcached: software architecture

- Client–server architecture

- **The servers** maintain a key–value associative array and do not communicate each other

- **The clients** populate this array and query it by key. They know all servers

- If a client wishes to set or read the value corresponding to a certain key, the client's library first computes a hash of the key to determine which server to use.
- The servers keep the values in RAM; if a server runs out of RAM, it discards the oldest values.

Memcached: software architecture

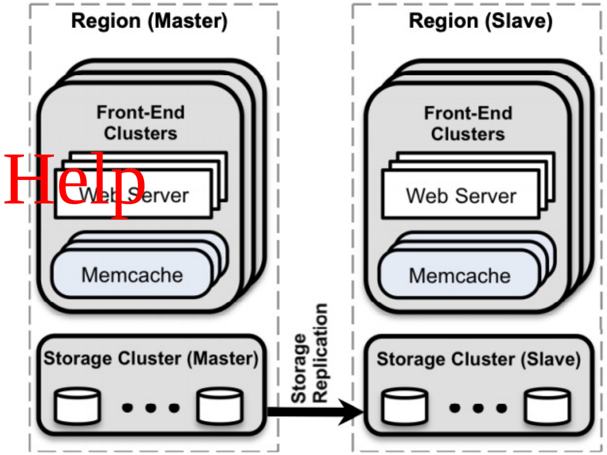
- Clients must treat Memcached as a transitory cache

- They cannot assume that data stored in Memcached is still there when they need it.

- Other databases, such as MemcacheDB, Couchbase Server, provide persistent storage while maintaining Memcached protocol compatibility.

Facebook: a real-world scenario

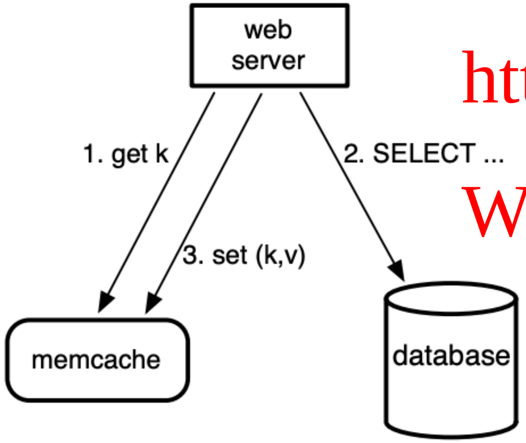
- Need to support very heavy read load
  - Over 1 billion reads/second
- Geographically distributed
- Support a constantly evolving product



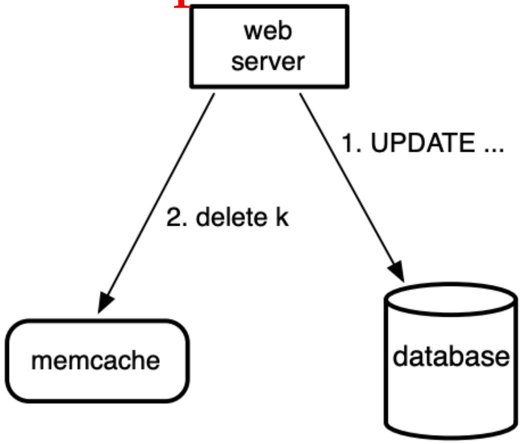
Scaling Memcache at Facebook, USENIX NSDI 2013

Facebook: a real world scenario

- Memcache as a demand-filled look-aside cache



Read path for a web server on a cache miss



The write path

Scaling Memcache at Facebook, USENIX NSDI 2013

Recap on Key-Value Stores

- Very large-scale storage systems

- Two operations
  - put(key,value)
  - value = get(key)

- Challenges
  - Fault tolerance → replication
  - Scalability → serve get()'s in parallel, replicate/cache hot tuples
  - Consistency → quorum consensus