Distributed Systems

CS422/522 Lecture17

17 November 2014

Lecture Outline

- Introduction
- Hadoop
- Chord

What’s a distributed system?

What's a distributed system?

A distributed system is a collection of loosely coupled nodes interconnected by a communication network.

What's a distributed system?

A distributed system is a collection of loosely coupled nodes interconnected by a communication network.

In a distributed system, users access remote resources in the same way they access local resources.

Why distributed systems?

Scalability

PC

Scalability

PC

- What if one computer is not enough?

Scalability

PC

Server

- What if one computer is not enough?
  - Buy a bigger (server-class) computer

Scalability

PC

Server

- What if one computer is not enough?
  - Buy a bigger (server-class) computer
- What if the biggest computer is not enough?

Scalability

PC

Server

Cluster

- What if one computer is not enough?
  - Buy a bigger (server-class) computer
- What if the biggest computer is not enough?
  - Buy many computers

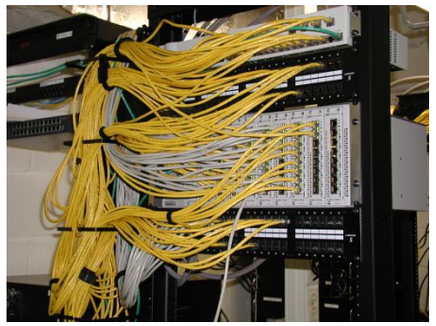
Scalability

Scalability

Rack

Scalability

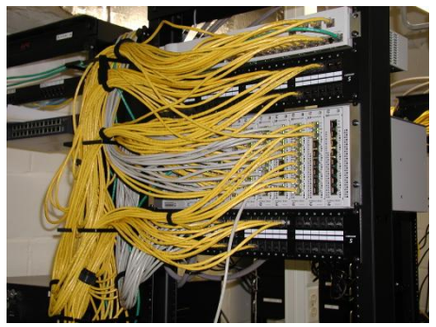
Rack



Network switches
(connects nodes with each other and with other racks)

Scalability

Rack

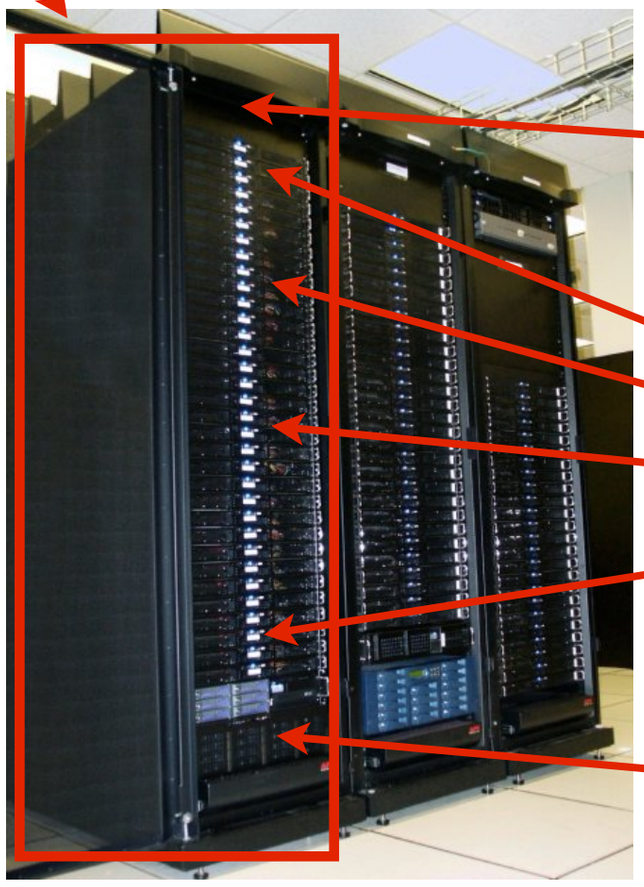


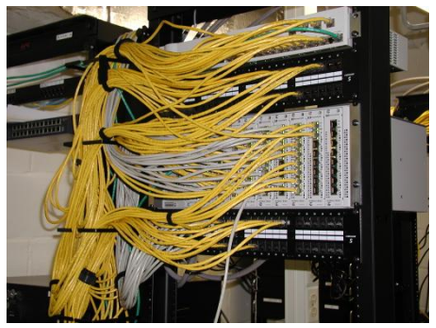
Network switches
(connects nodes with each other and with other racks)

Many nodes/blades
(often identical)

Scalability

Rack





Network switches
(connects nodes with each other and with other racks)

Many nodes/blades
(often identical)

Storage device(s)

Scalability

PC

Server

Cluster

- What if cluster is too big to fit into machine room?

Scalability

PC

Server

Cluster

- What if cluster is too big to fit into machine room?
  - Build a separate building for the cluster
  - Building can have lots of cooling and power

Scalability

PC

Server

Cluster

Data center

- What if cluster is too big to fit into machine room?
  - Build a separate building for the cluster
  - Building can have lots of cooling and power
  - Result: Data center

Google Data Center in Oregon

Google Data Center in Oregon

Data centers (size of a football field)

Google Data Center in Oregon

Data centers (size of a football field)

- A warehouse-sized computer
  - A single data center can easily contain 10,000 racks with 100 cores in each rack (1,000,000 cores total)

Google Data Centers World Wide

Why distributed systems?

- Resource sharing
  - E.g., Napster

Why distributed systems?

- Resource sharing
  - E.g., Napster
- Computation speedup
  - E.g., Hadoop

Why distributed systems?

- Resource sharing
  - E.g., Napster
- Computation speedup
  - E.g., Hadoop
- Reliability
  - E.g., Amazon S3

Why distributed systems?

- Resource sharing
  - E.g., Napster
- Computation speedup
  - E.g., Hadoop
- Reliability
  - E.g., S3

Issues?

Lecture Outline

- Introduction
- Hadoop
- Chord

Lecture Outline

- Introduction
- Hadoop
- Chord

What's the Hadoop

- A project based on GFS and MapReduce
  - Distribute data across machines
  - Try to achieve the reliability and scalability

The Google File System

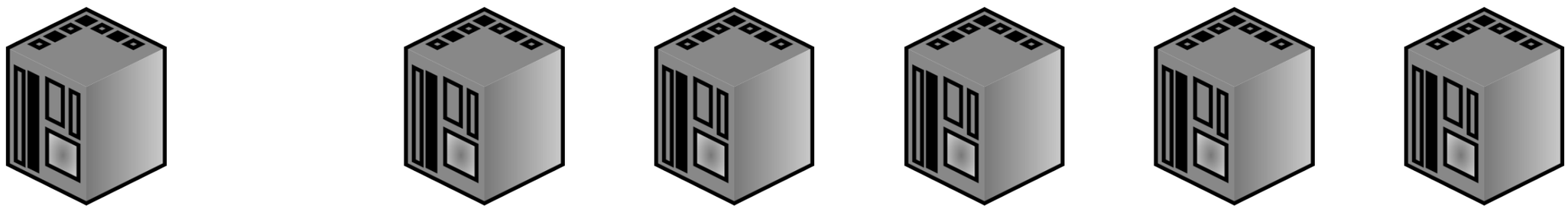
What's the Hadoop

- A project based on GFS and MapReduce
  - Distribute data across machines
  - Try to achieve the reliability and scalability

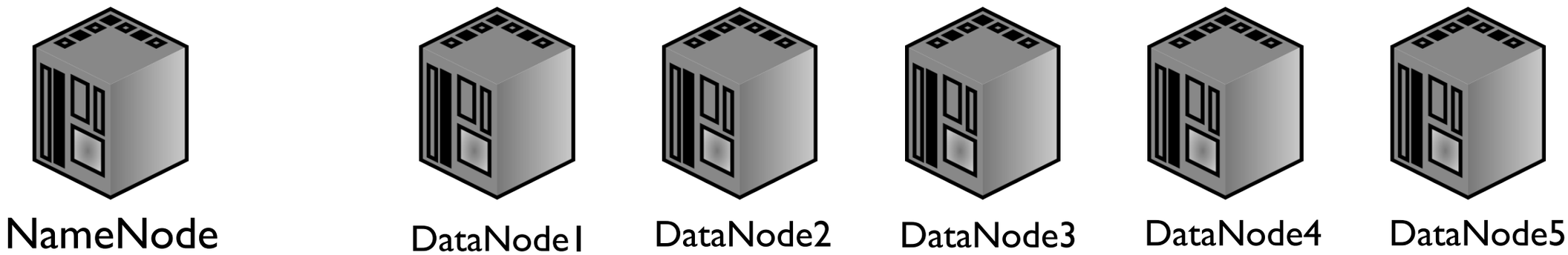
- An open-source software framework for big data
  - Distributed storage
  - Distributed processing

Why Hadoop?

Hadoop Components

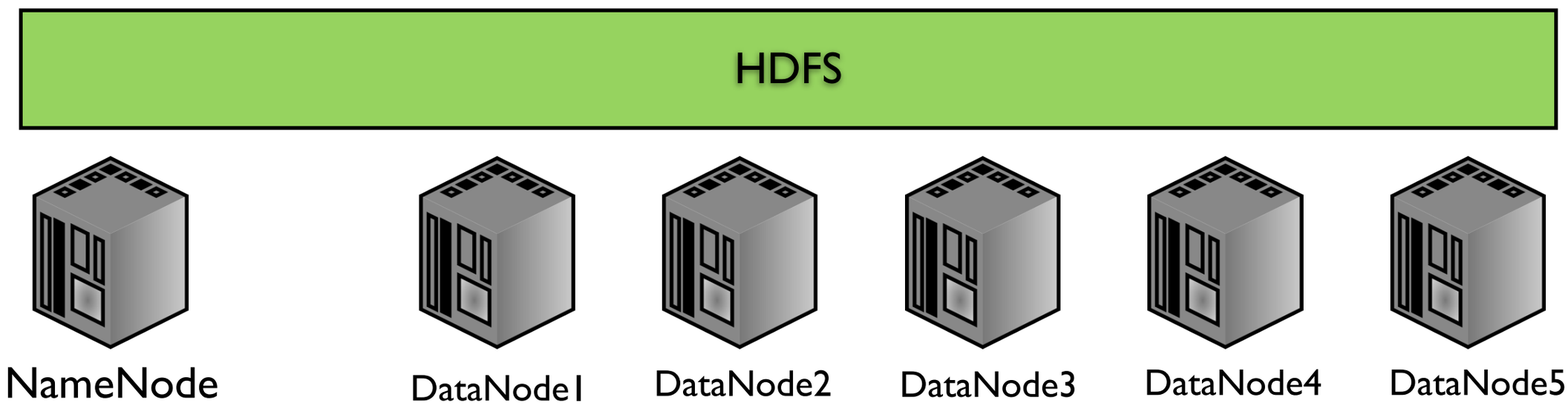


Hadoop Components



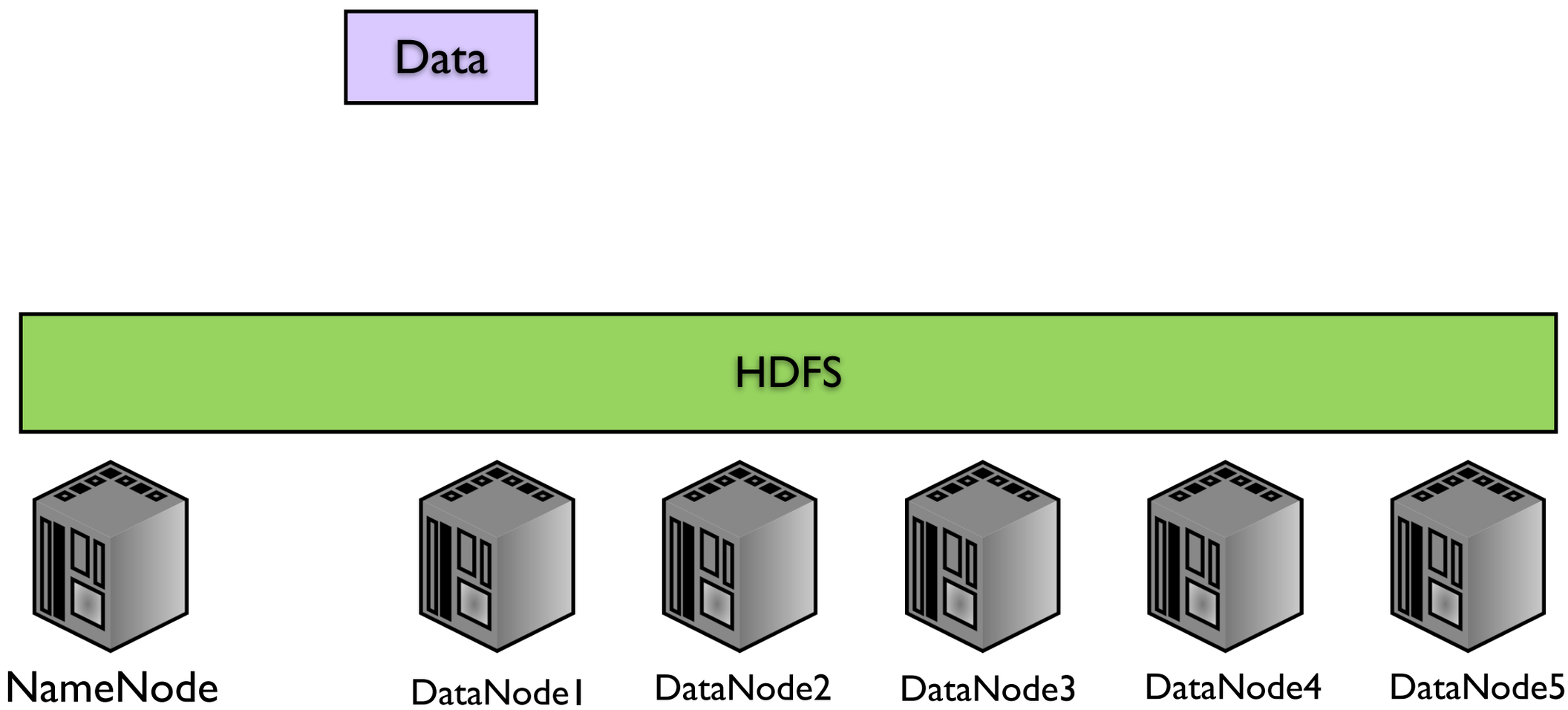
Hadoop Components

- Two core components
  - The Hadoop distributed file system (HDFS)



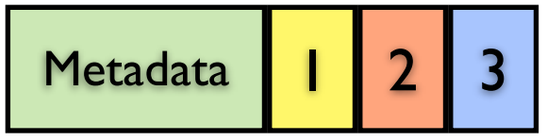
Hadoop Components

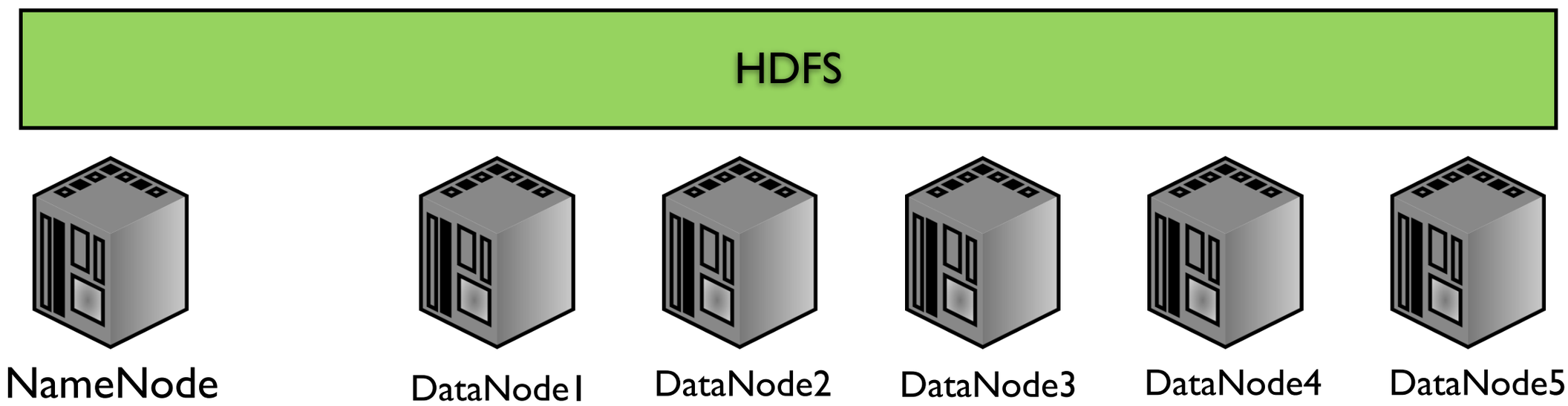
- Two core components
  - The Hadoop distributed file system (HDFS)



Hadoop Components

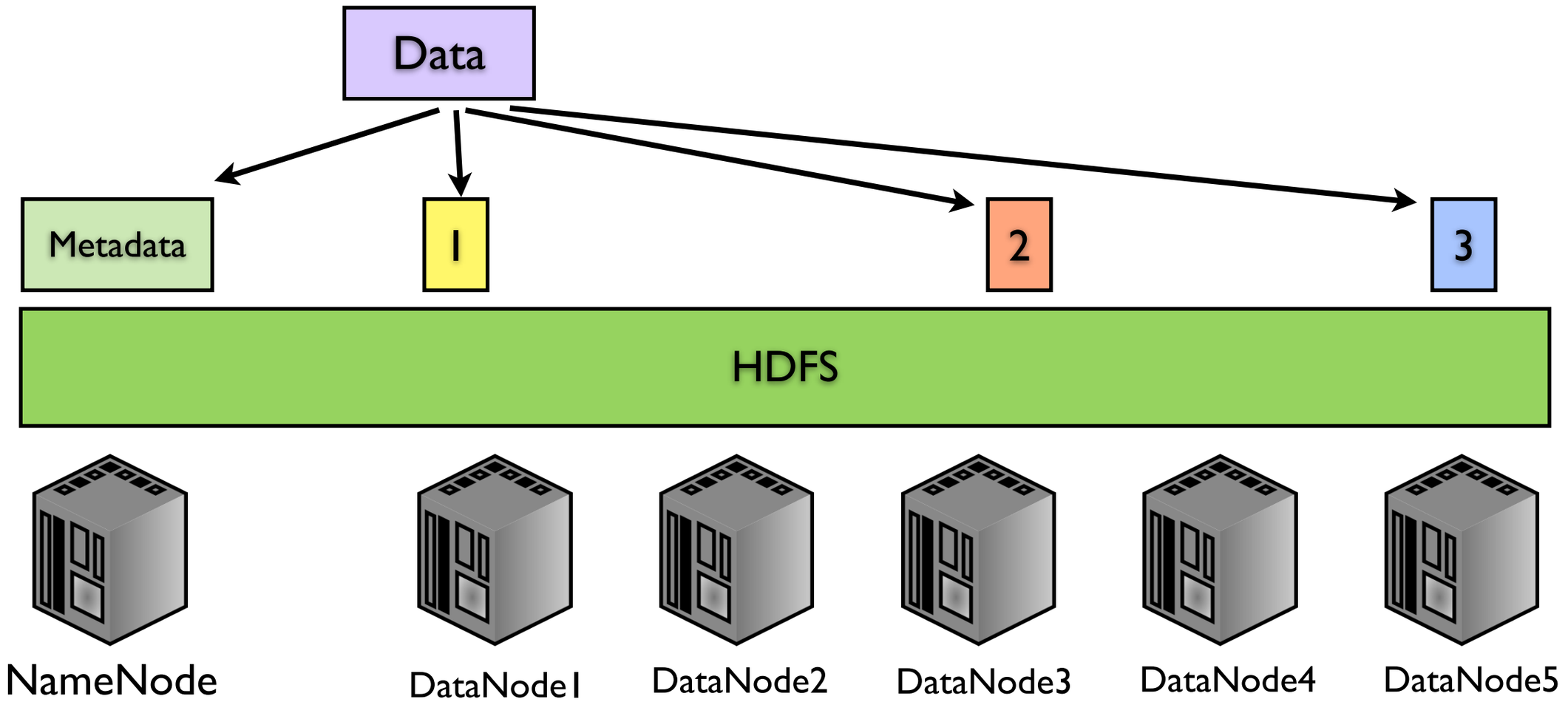
- Two core components
  - The Hadoop distributed file system (HDFS)





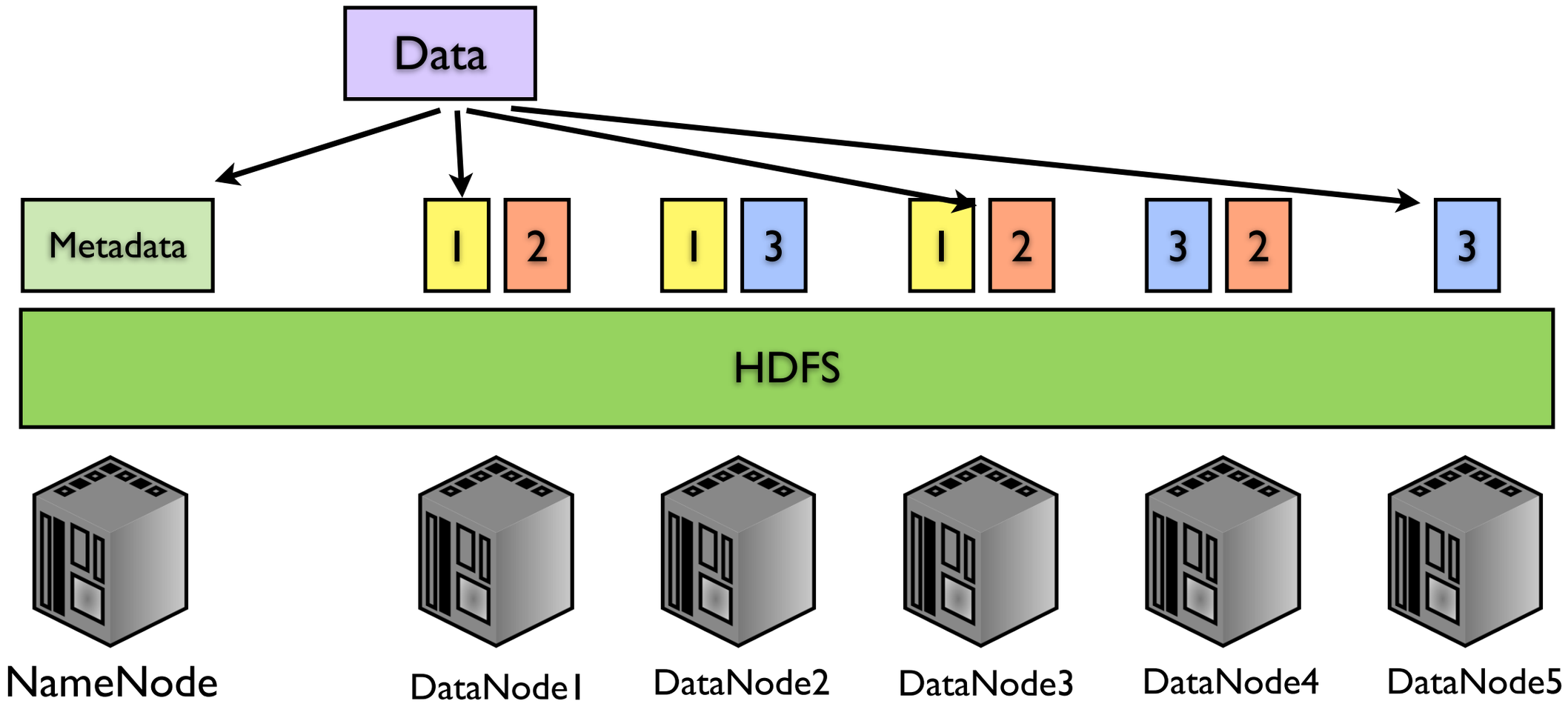
Hadoop Components

- Two core components
  - The Hadoop distributed file system (HDFS)



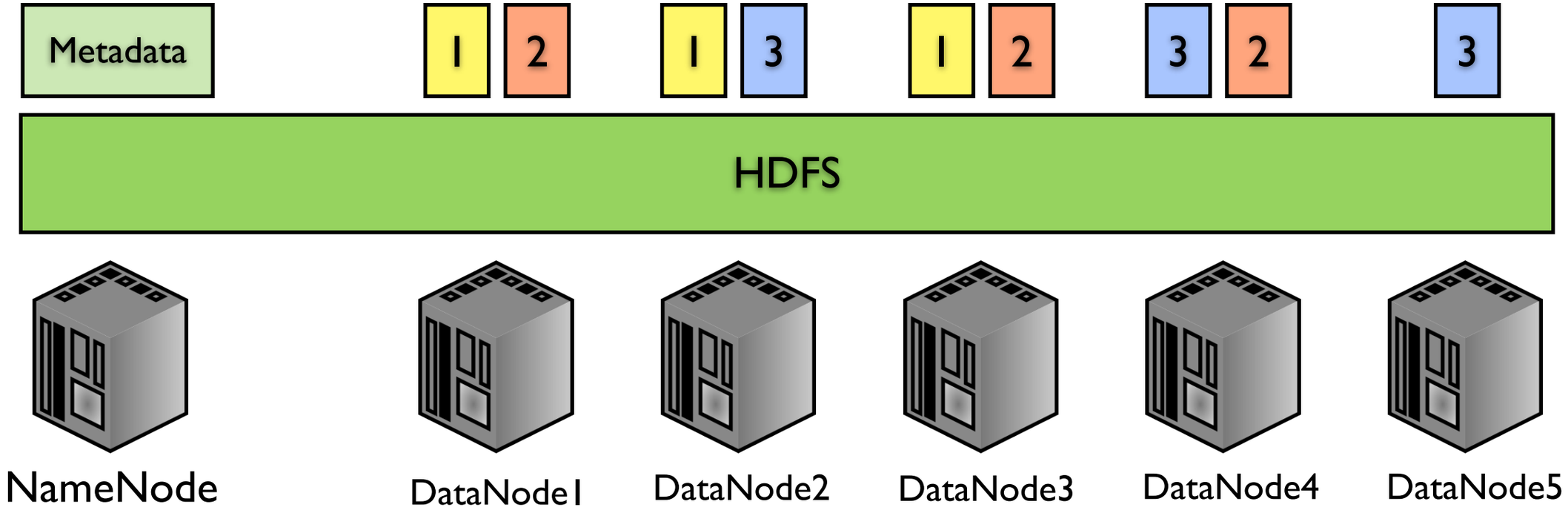
Hadoop Components

- Two core components
  - The Hadoop distributed file system (HDFS)



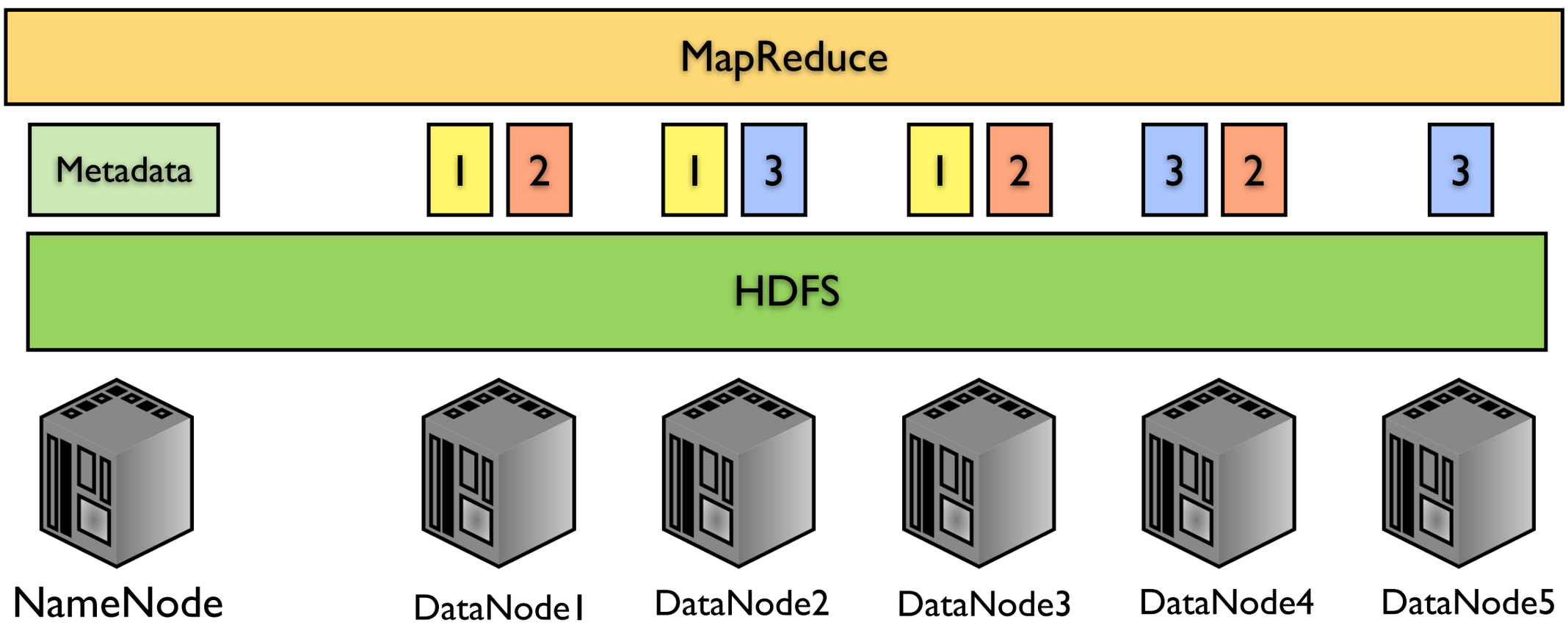
Hadoop Components

- Two core components
  - The Hadoop distributed file system (HDFS)
  - MapReduce software framework



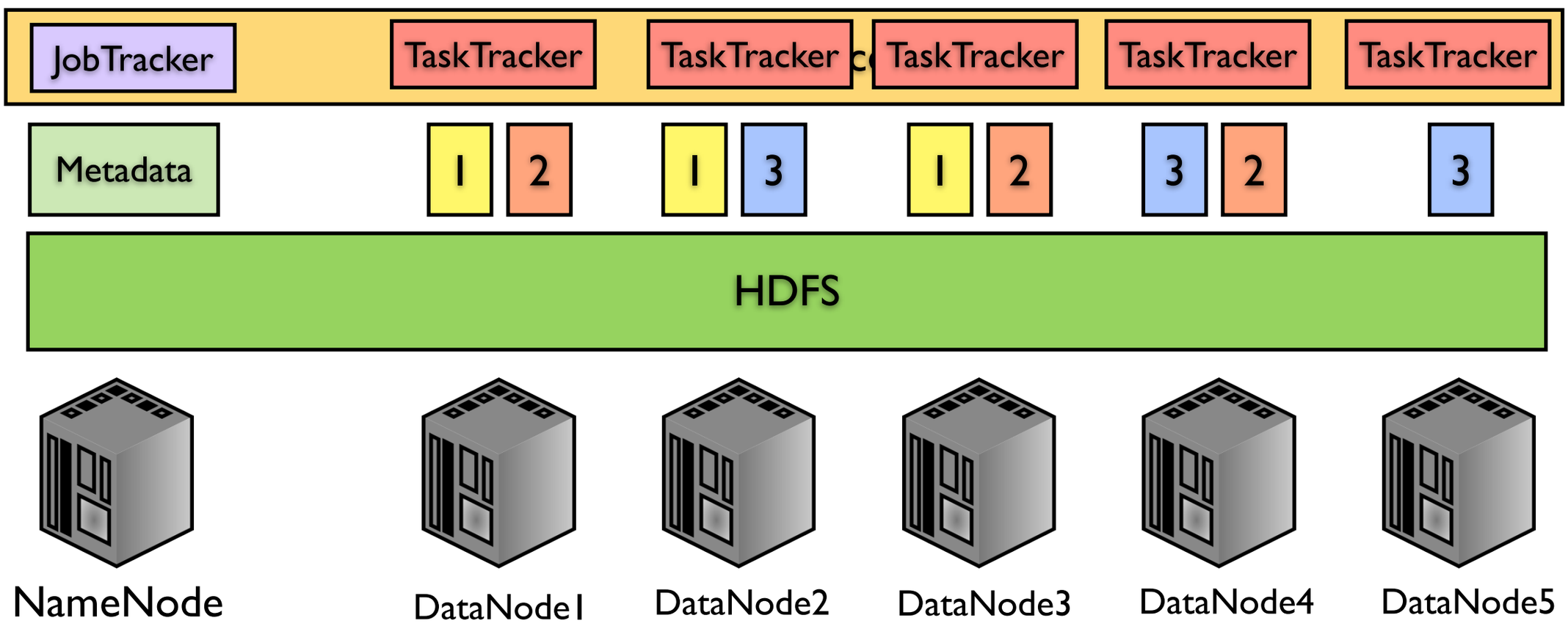
Hadoop Components

- Two core components
  - The Hadoop distributed file system (HDFS)
  - MapReduce software framework



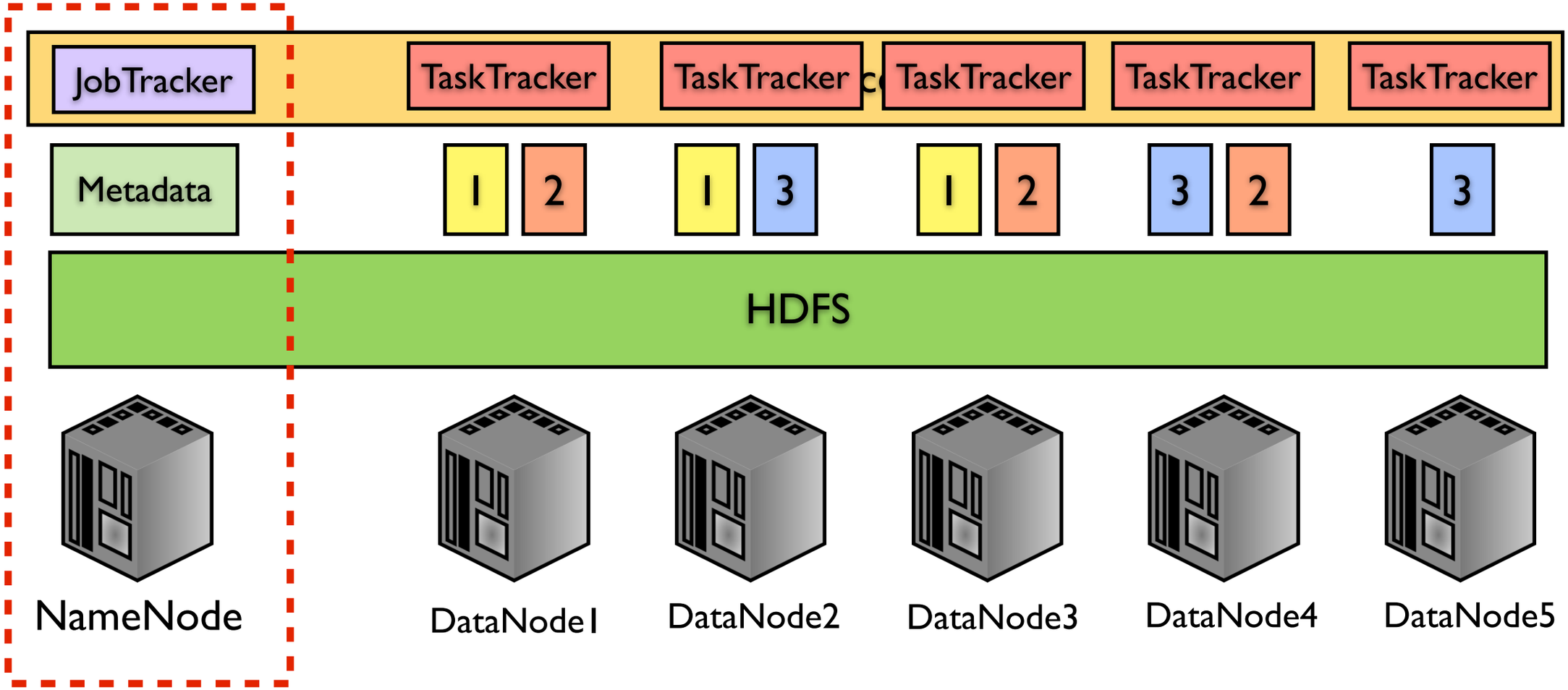
Hadoop Components

- Two core components
  - The Hadoop distributed file system (HDFS)
  - MapReduce software framework



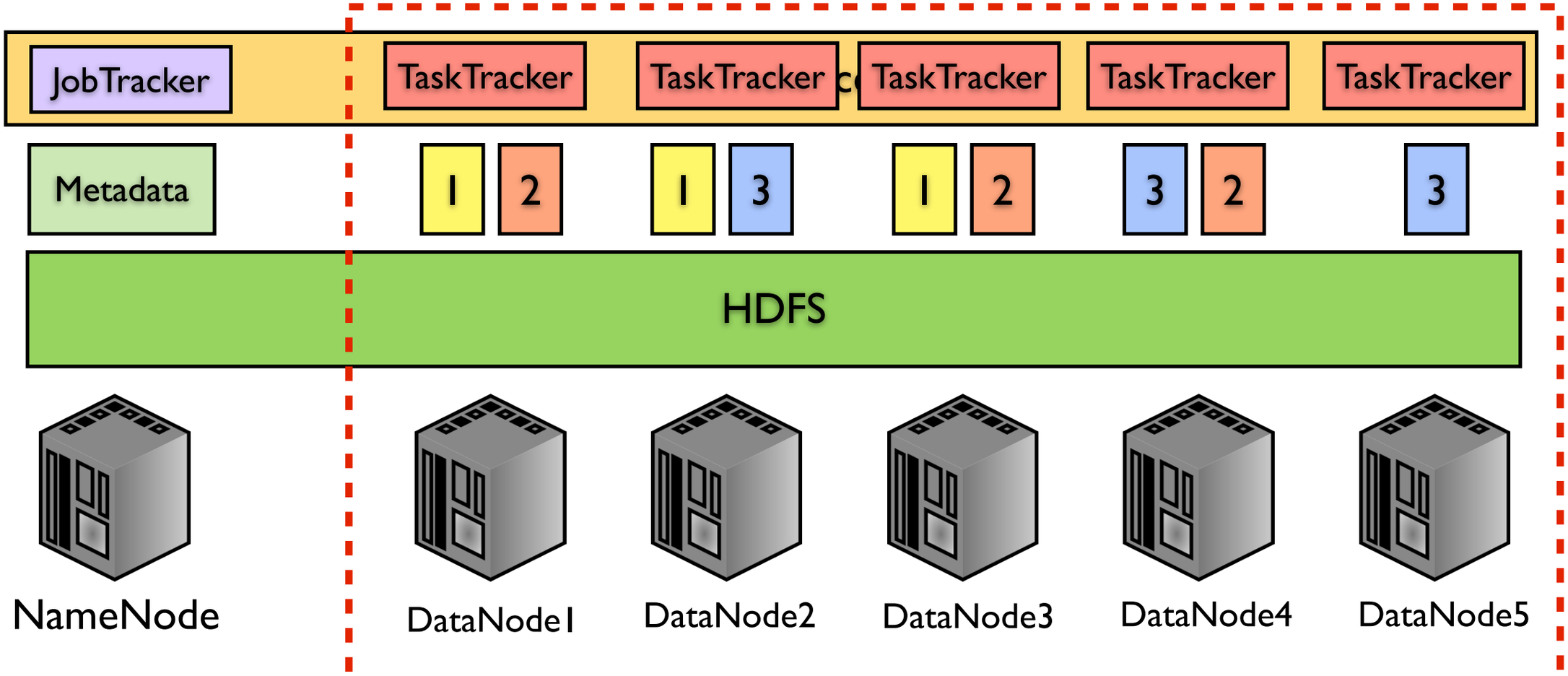
Hadoop Components

- Two core components
  - The Hadoop distributed file system (HDFS)
  - MapReduce software framework



Hadoop Components

- Two core components
  - The Hadoop distributed file system (HDFS)
  - MapReduce software framework



HDFS

- HDFS is responsible for storing data
  - Data is split into blocks and distributed across nodes
  - Each block is replicated

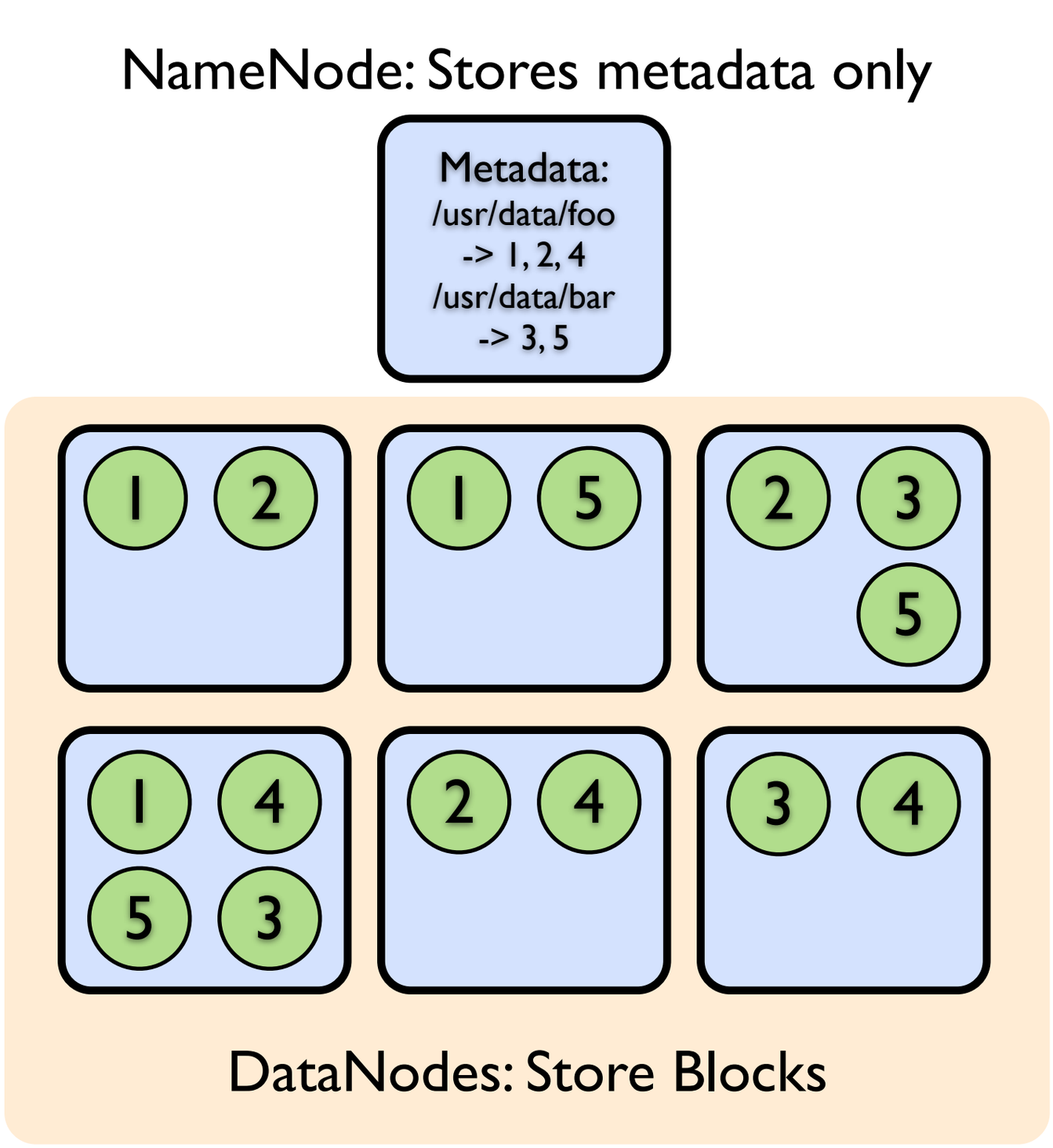
- Implementation of HDFS:
  - Based on Google's GFS
  - Offers redundant storage for massive amounts of data

Getting Data in/out of HDFS

- Hadoop API:
  - Use `hadoop fs` to work with data in HDFS

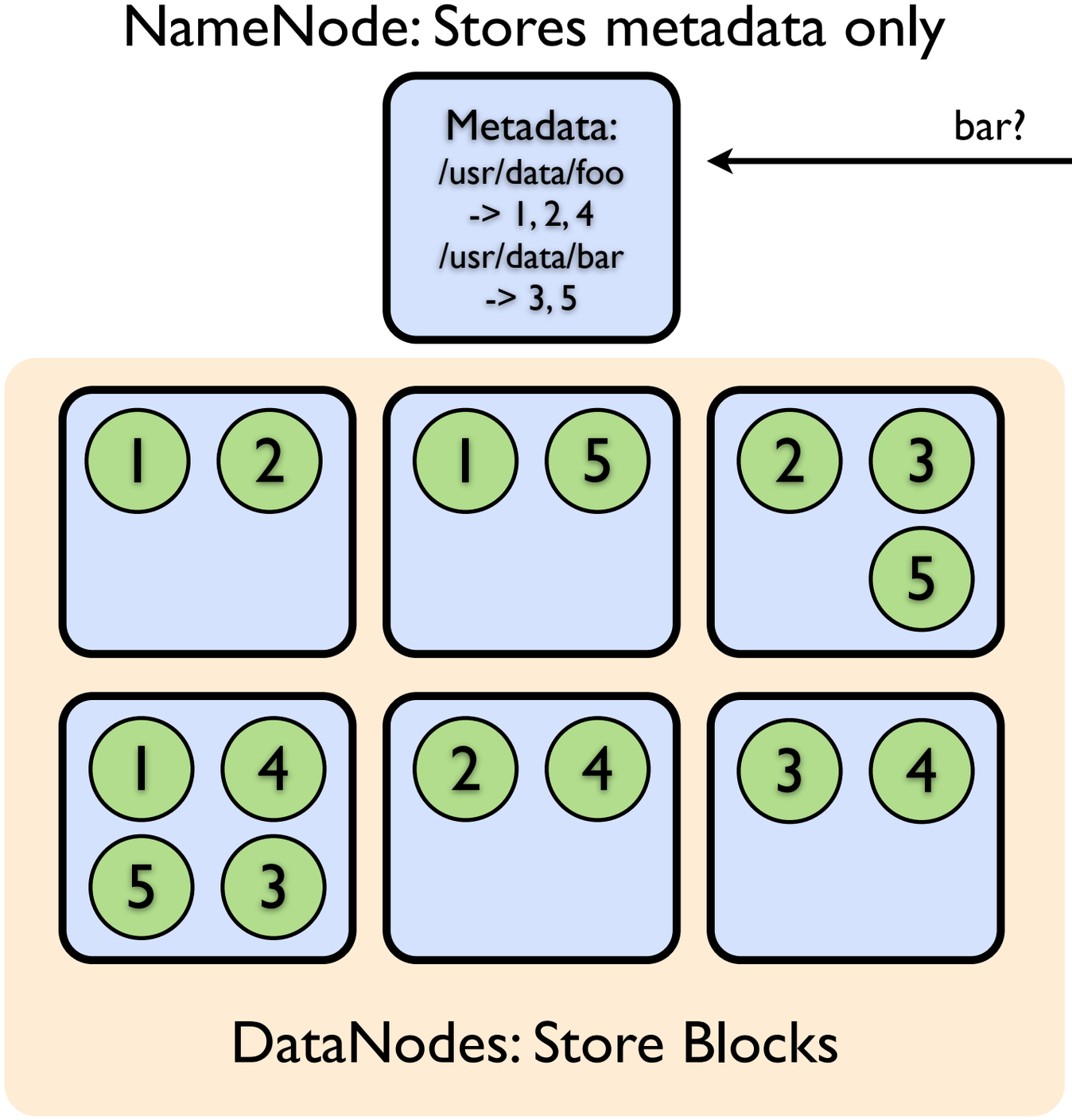
```
- hadoop fs -copyFromLocal local_dir /hdfs_dir
- hadoop fs -copyToLocal /hdfs_dir local_dir
```

HDFS Example



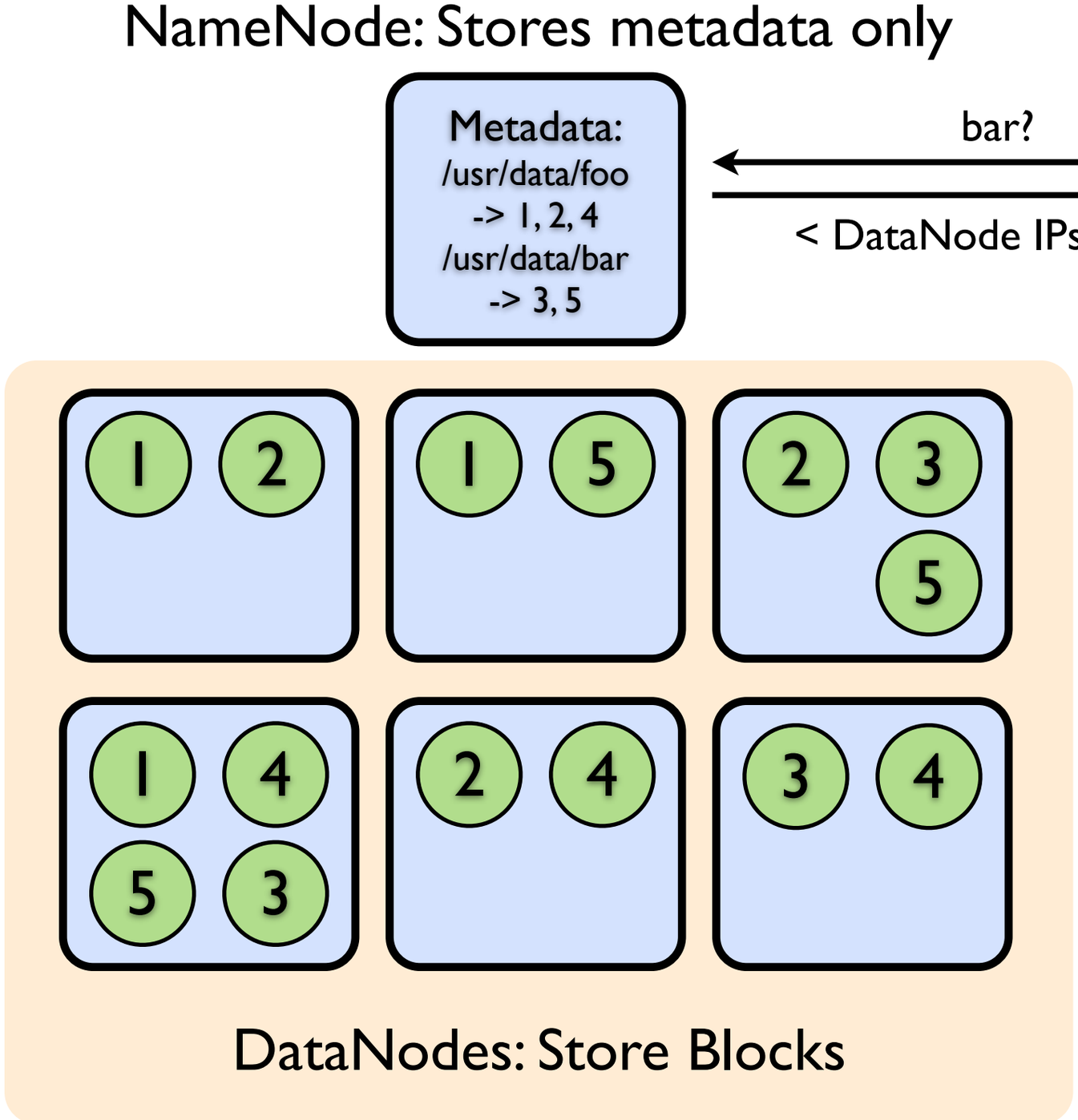
- NameNode holds metadata for the data files
- DataNodes hold the actual blocks
  - Each block is replicated three times

HDFS Example



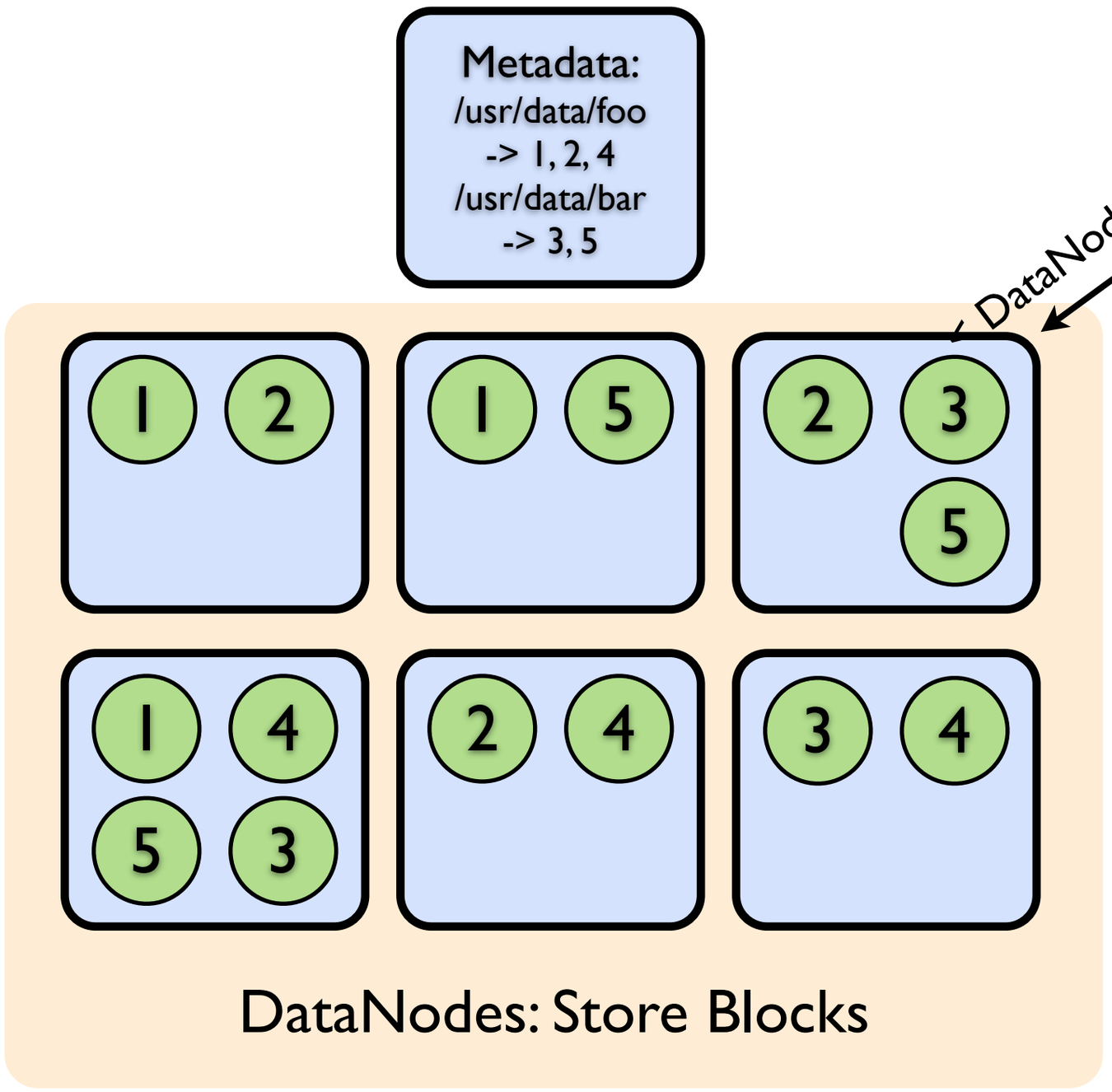
- NameNode holds metadata for the data files
- DataNodes hold the actual blocks
  - Each block is replicated three times
- When a client wants to read file

HDFS Example



- NameNode holds metadata for the data files
- DataNodes hold the actual blocks
  - Each block is replicated three times
- When a client wants to read file

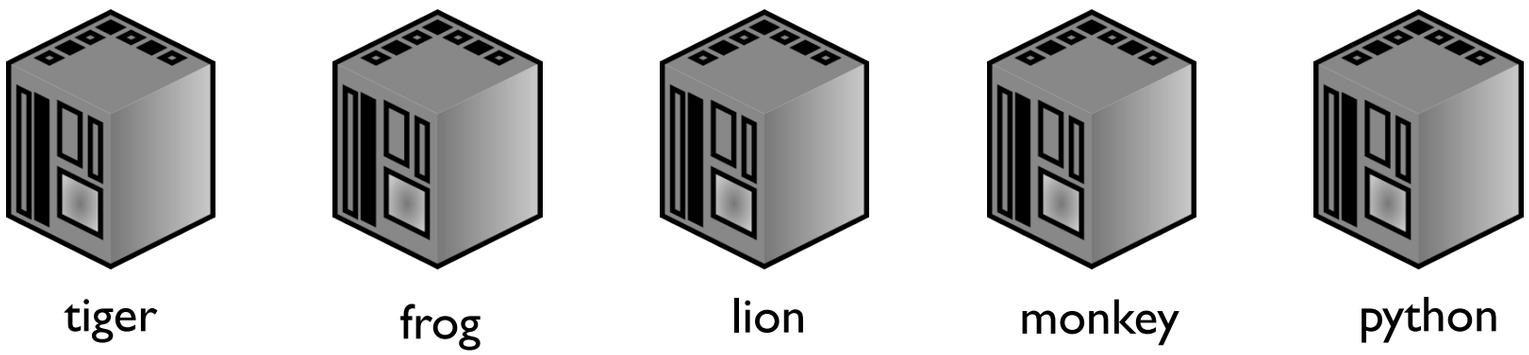
HDFS Example



- NameNode holds metadata for the data files
- DataNodes hold the actual blocks
  - Each block is replicated three times
- When a client wants to read file

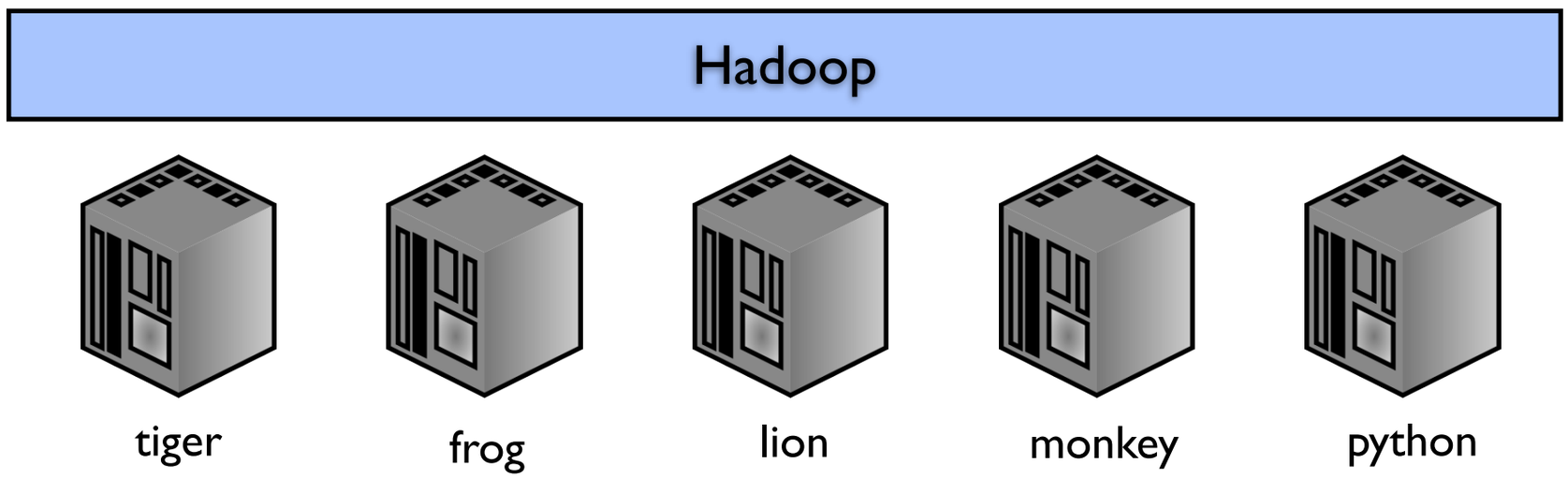
Case Study

- Policy: Assigning blocks based on disk space



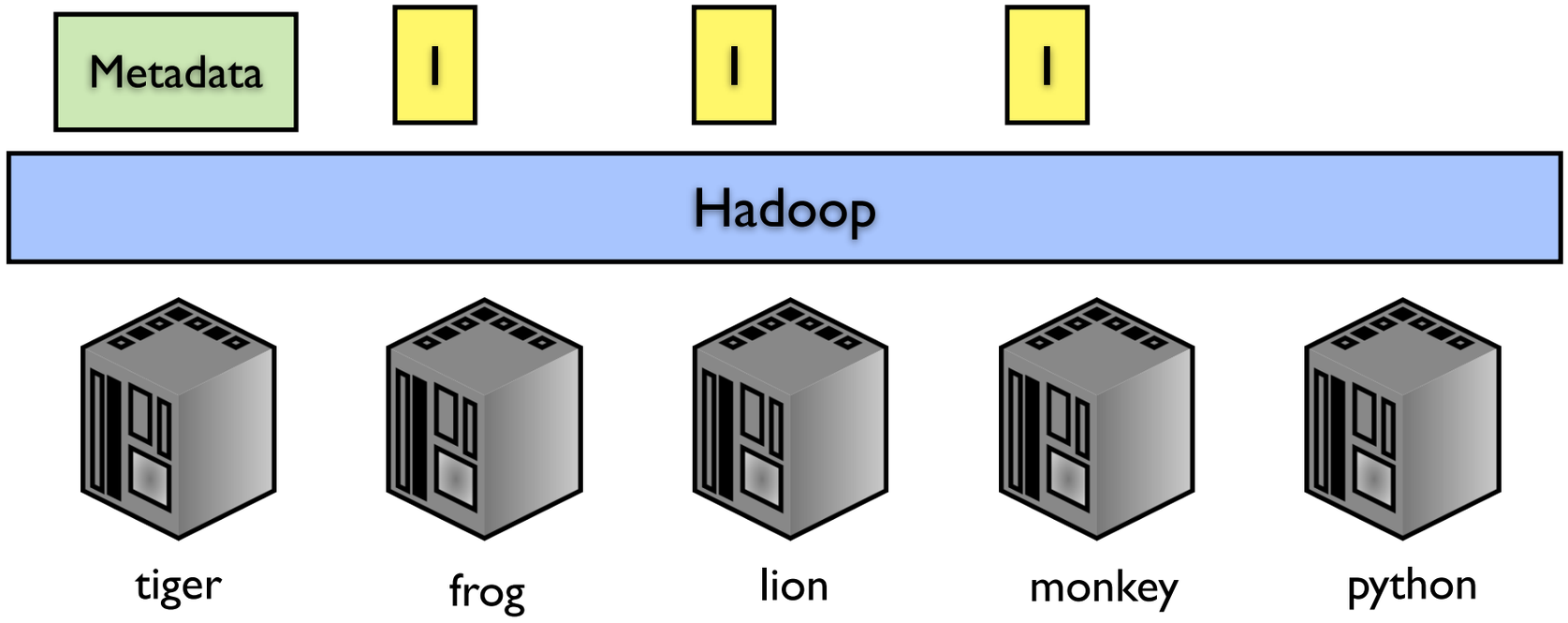
Case Study

- Policy: Assigning blocks based on disk space



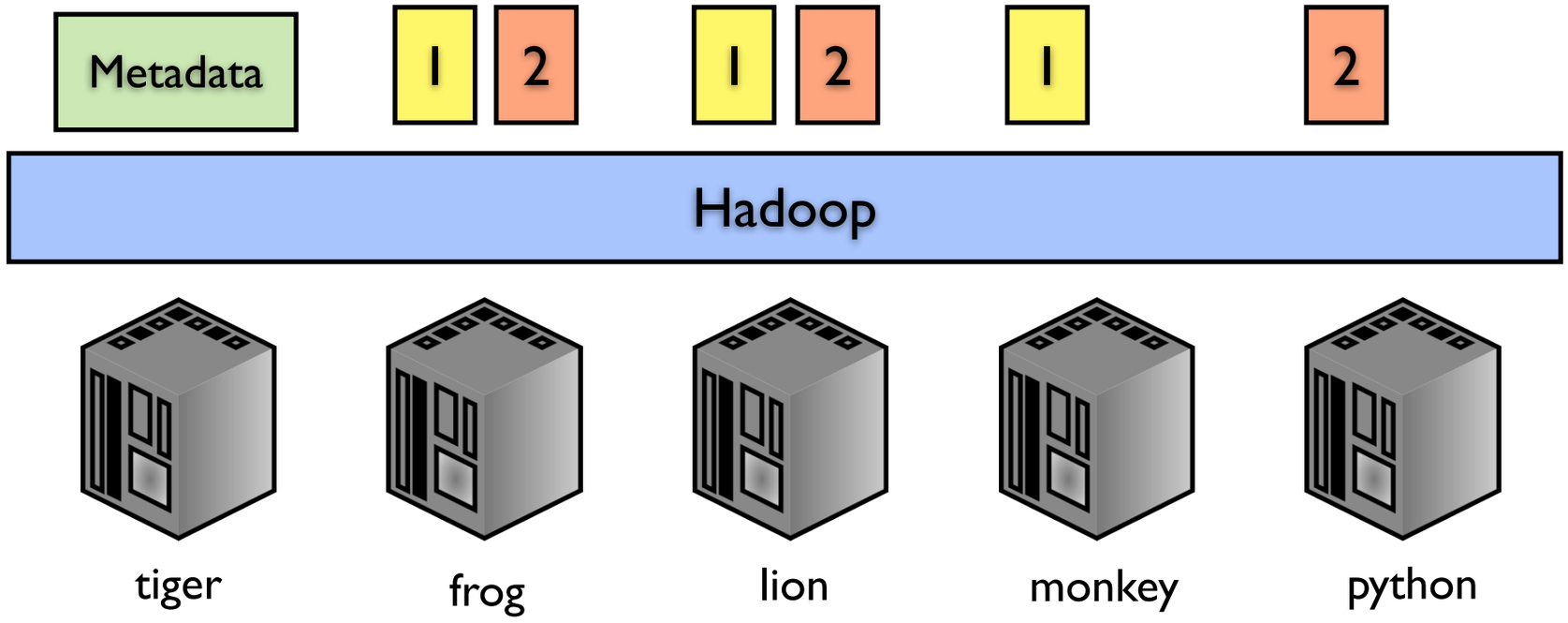
Case Study

- Policy: Assigning blocks based on disk space



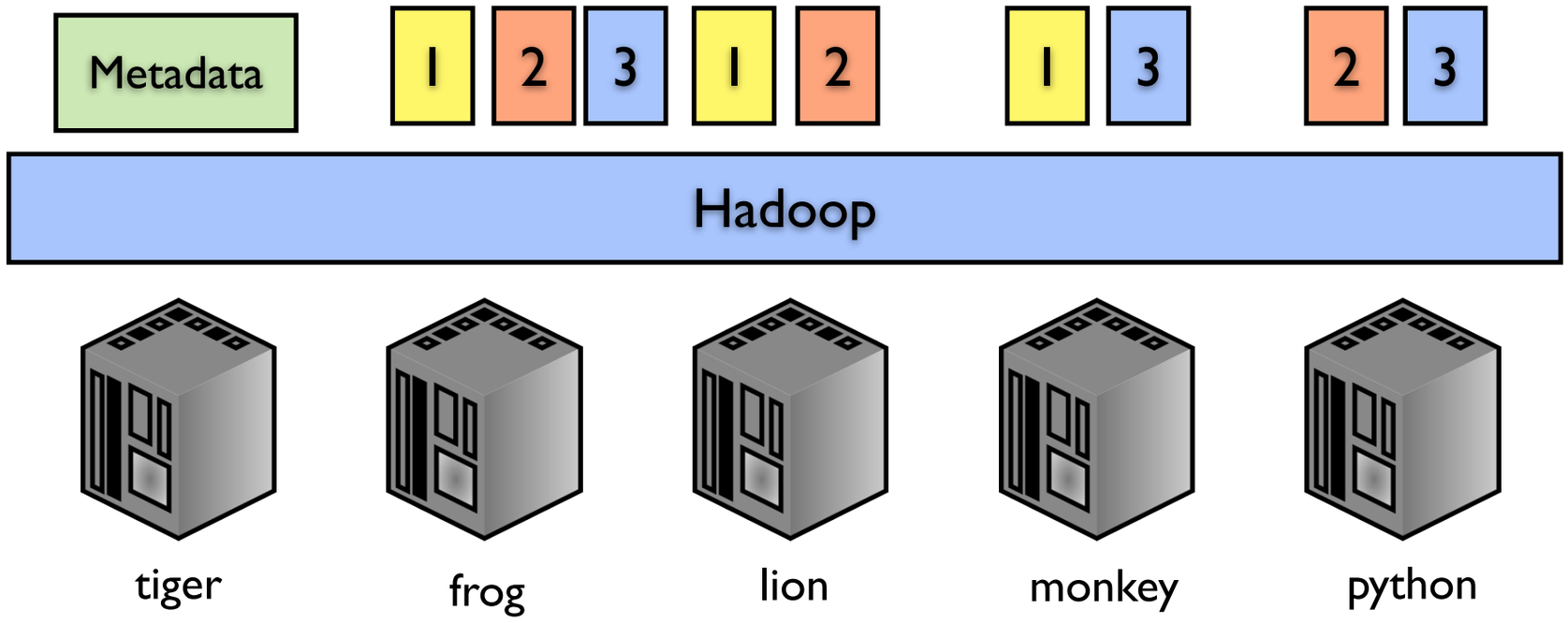
Case Study

- Policy: Assigning blocks based on disk space



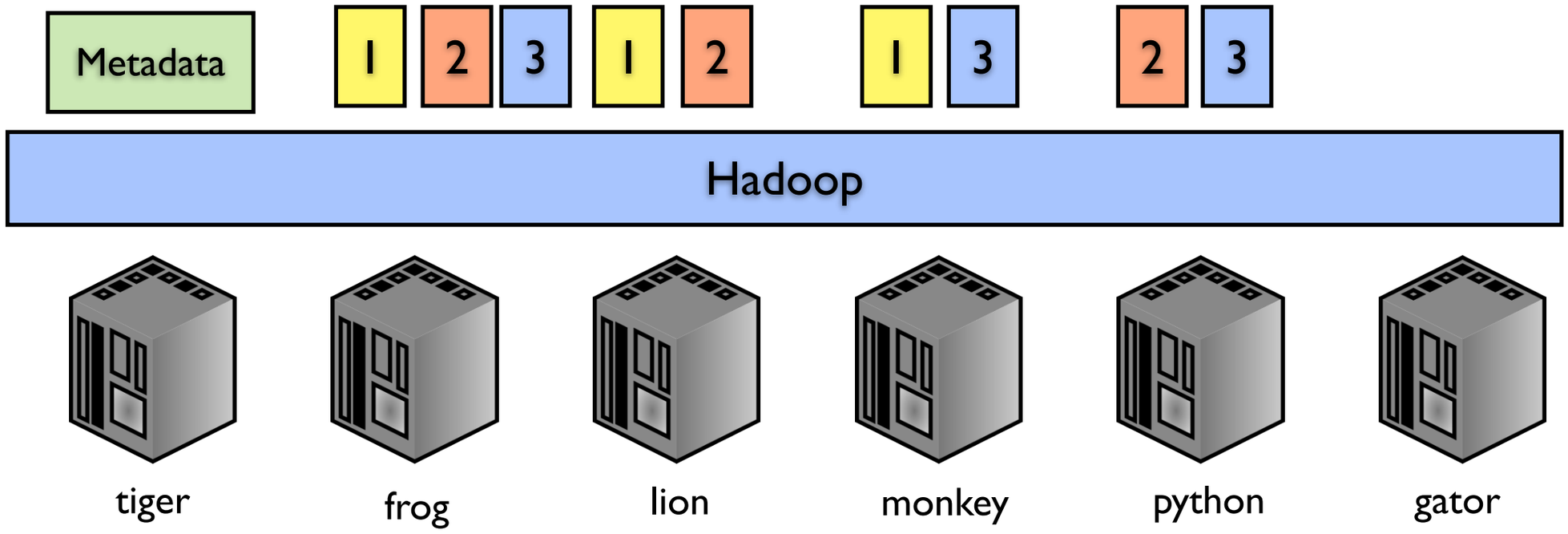
Case Study

- Policy: Assigning blocks based on disk space



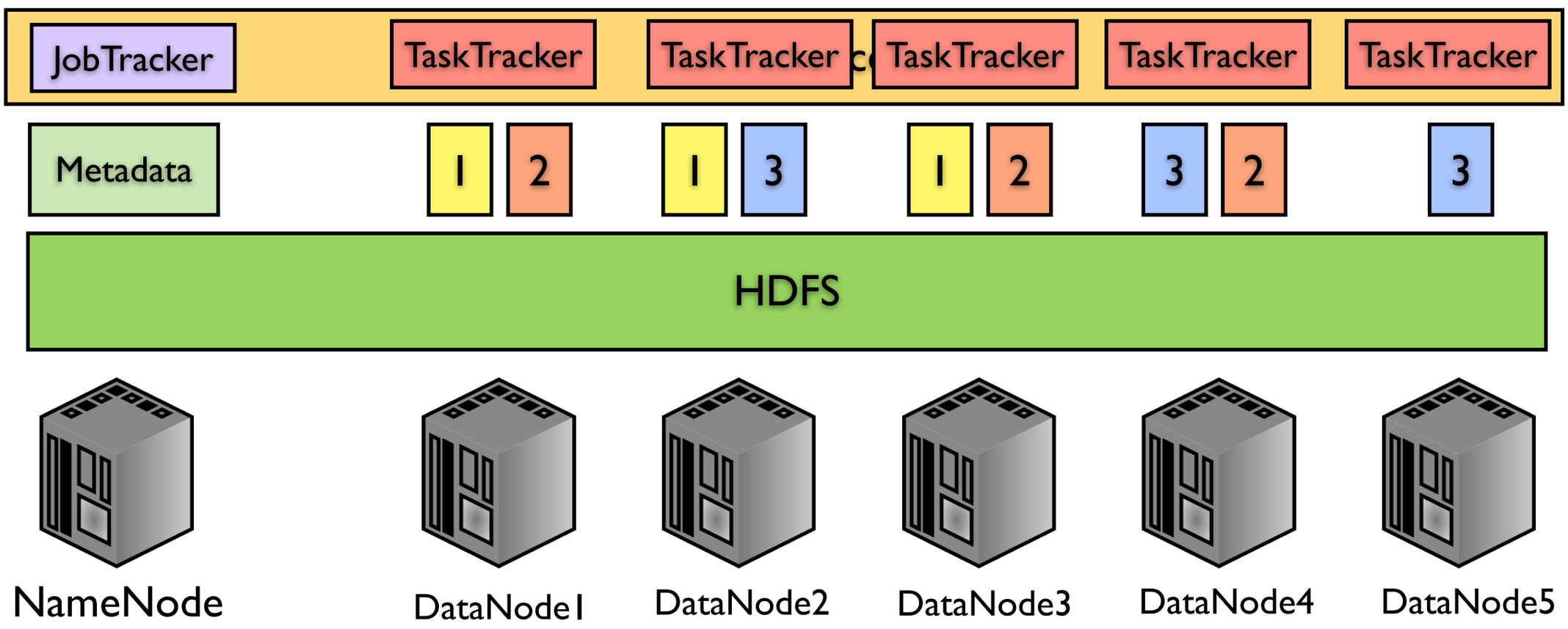
Case Study

- Policy: Assigning blocks based on disk space



Recall: Hadoop Components

- Two core components
  - The Hadoop distributed file system (HDFS)
  - MapReduce software framework



MapReduce

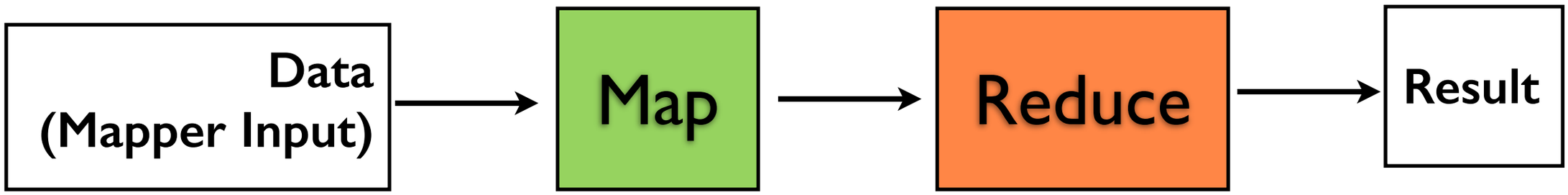
- A method distributing a task across nodes:
  - Each node processes data stored on that node

- Consists of two phases:
  - Map
  - Reduce

MapReduce

- A method distributing a task across nodes:
  - Each node processes data stored on that node

- Consists of two phases:
  - Map
  - Reduce



Features of MapReduce

- Automatic parallelization and distribution
- A clean abstraction for programmers
  - MapReduce programs are usually written in Java
- MapReduce abstracts all the housekeeping away from the developer:
  - Developers concentrate on Map and Reduce functions

MapReduce Example: Word Count

Count the # of occurrences of each word in a large amount of input data

```
Map(input_key, input_value) {
  foreach word w in input_value:
    emit(w, 1);
}
```

MapReduce Example: Word Count

Count the # of occurrences of each word in a large amount of input data

```
Map(input_key, input_value) {
  foreach word w in input_value:
    emit(w, 1);
}
```

- Input to the Mapper

```
(3414, ‘the cat sat on the mat’)
(3437, ‘the aardvark sat on the sofa’)
```

MapReduce Example: Word Count

Count the # of occurrences of each word in a large amount of input data

```
Map(input_key, input_value) {
  foreach word w in input_value:
    emit(w, 1);
}
```

- Input to the Mapper

```
(3414, ‘the cat sat on the mat’)
(3437, ‘the aardvark sat on the sofa’)
```

- Output from the Mapper

```
(‘the’, 1), (‘cat’, 1), (‘sat’, 1), (‘on’, 1),
(‘the’, 1), (‘mat’, 1), (‘the’, 1), (‘aardvark’, 1),
(‘sat’, 1), (‘on’, 1), (‘the’, 1), (‘sofa’, 1)
```

Map Phase

Mapper Input

```
the cat sat on the mat
the aardvark sat on the sofa
```

Map Phase

Mapper Input

```
the cat sat on the mat
the aardvark sat on the sofa
```

Mapper Output

```
('the', 1),
('cat', 1),
('sat', 1),
('on', 1),
('the', 1),
('mat', 1),
('the', 1),
('aardvark', 1),
('sat', 1),
('on', 1),
('the', 1),
('sofa', 1)
```

Reducer

- After the Map, all the intermediate values for a given intermediate key are combined together into a list

Reducer

- After the Map, all the intermediate values for a given intermediate key are combined together into a list

Mapper Output

```
('the', 1),
('cat', 1),
('sat', 1),
('on', 1),
('the', 1),
('mat', 1),
('the', 1),
('aardvark', 1),
('sat', 1),
('on', 1),
('the', 1),
('sofa', 1)
```

Reducer Input

```
aardvark, 1
cat, 1
mat, 1
on [1, 1]
sat [1, 1]
sofa, 1
the [1, 1, 1, 1]
```

Reducer

Add up all the values associated with each intermediate key:

```
Reduce(output_key, intermediate_vals) {
  set count = 0;
  foreach v in intermediate_vals:
    count += v;
  emit(output_key, count);
}
```

Reducer

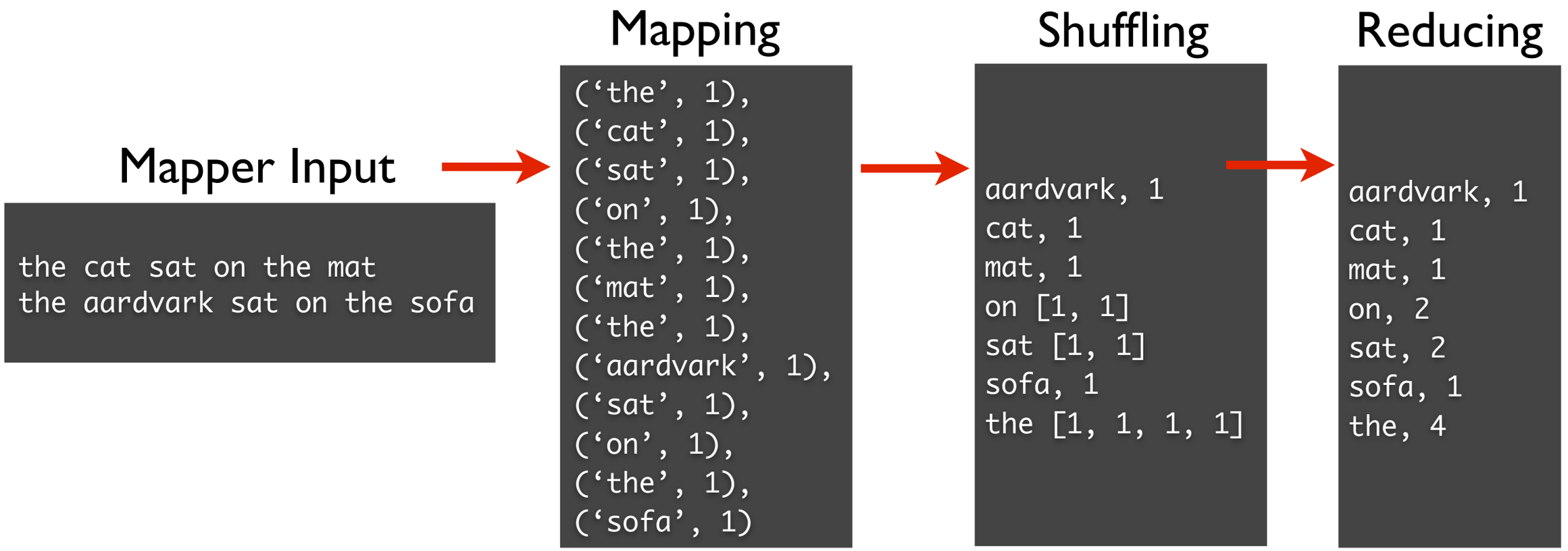
Add up all the values associated with each intermediate key:

```
Reduce(output_key, intermediate_vals) {
  set count = 0;
  foreach v in intermediate_vals:
    count += v;
  emit(output_key, count);
}
```

- Output from the Reducer

```
('the', 4), ('sat', 2), ('on', 2), ('sofa', 1),
('mat', 1), ('cat', 1), ('aardvark', 1)
```

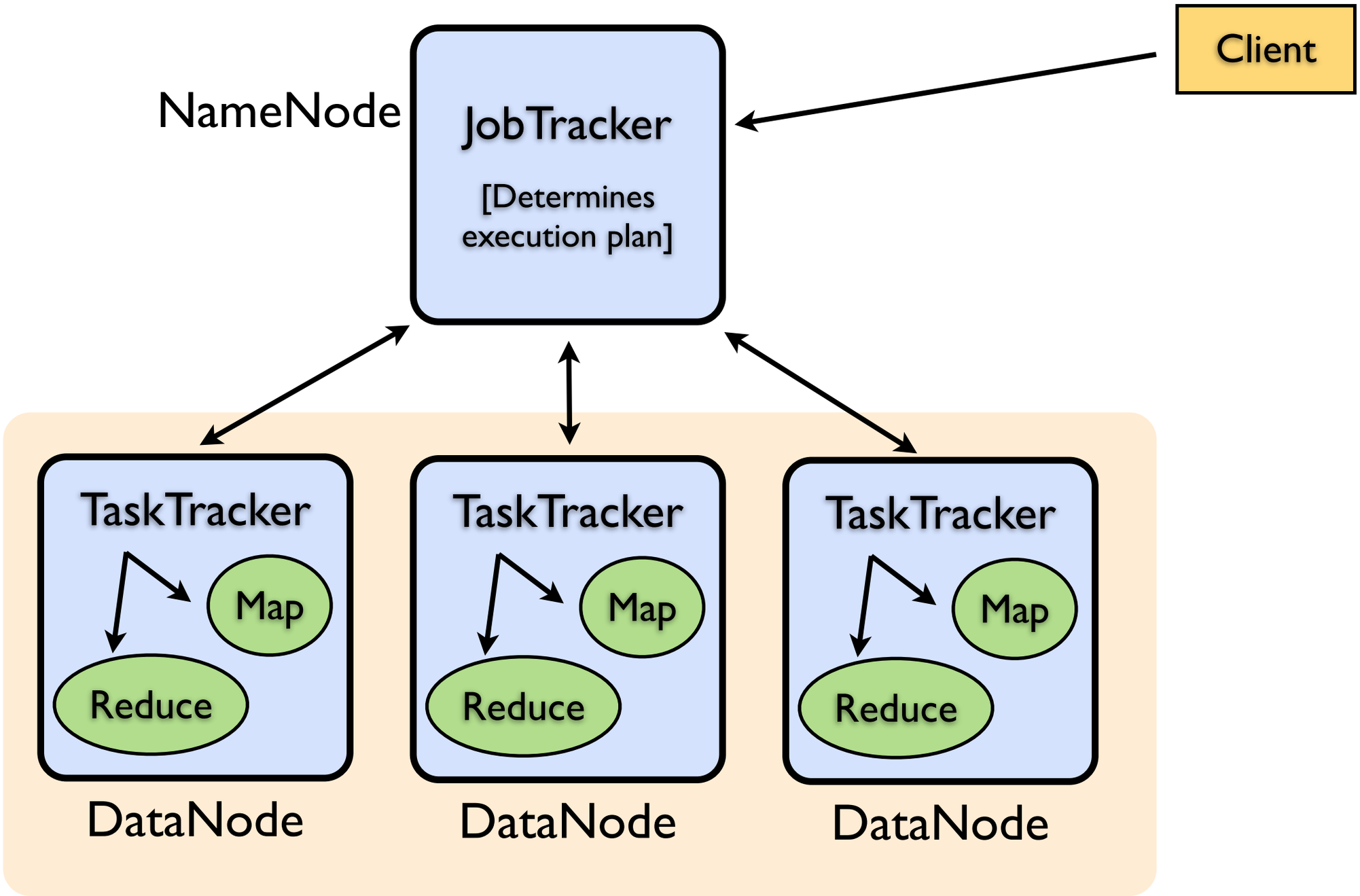
Map + Reduce


Mapper Input
the cat sat on the mat
the aardvark sat on the sofa
Mapping
('the', 1),
('cat', 1),
('sat', 1),
('on', 1),
('the', 1),
('mat', 1),
('the', 1),
('aardvark', 1),
('sat', 1),
('on', 1),
('the', 1),
('sofa', 1)
Shuffling
aardvark, 1
cat, 1
mat, 1
on [1, 1]
sat [1, 1]
sofa, 1
the [1, 1, 1, 1]
Reducing
aardvark, 1
cat, 1
mat, 1
on, 2
sat, 2
sofa, 1
the, 4

Why we care about counting words

- Word count is challenging over massive amounts of data
- Fundamentals of statistics often are aggregate functions
- Most aggregation functions have distributive nature
- MapReduce breaks complex tasks into smaller pieces which can be executed in parallel

Deployment


NameNode
JobTracker
[Determines execution plan]
Client
TaskTracker
Map
Reduce
DataNode
TaskTracker
Map
Reduce
DataNode
TaskTracker
Map
Reduce
DataNode

Discussions

Lecture Outline

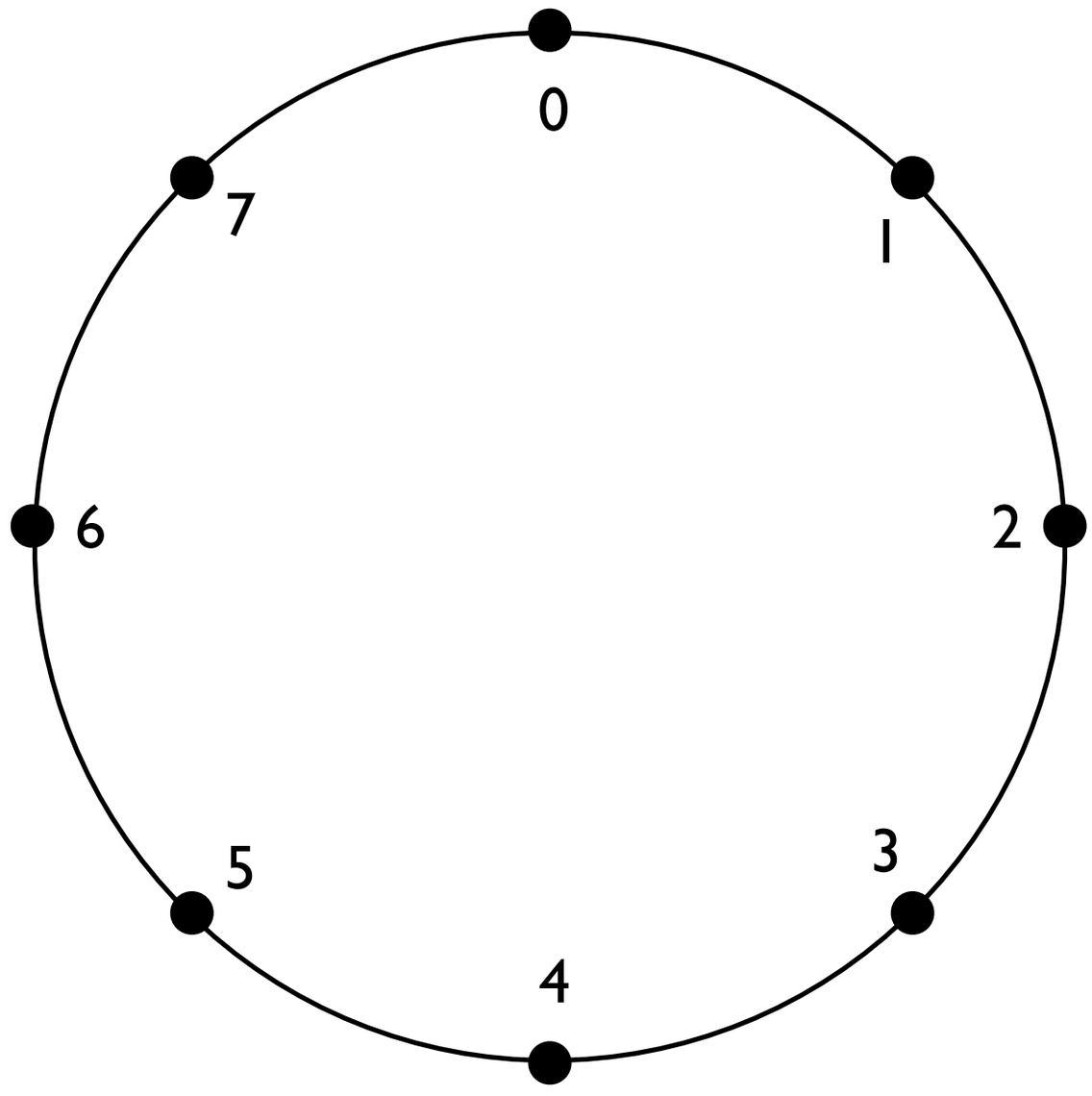
- Introduction
- Hadoop
- Chord

Lecture Outline

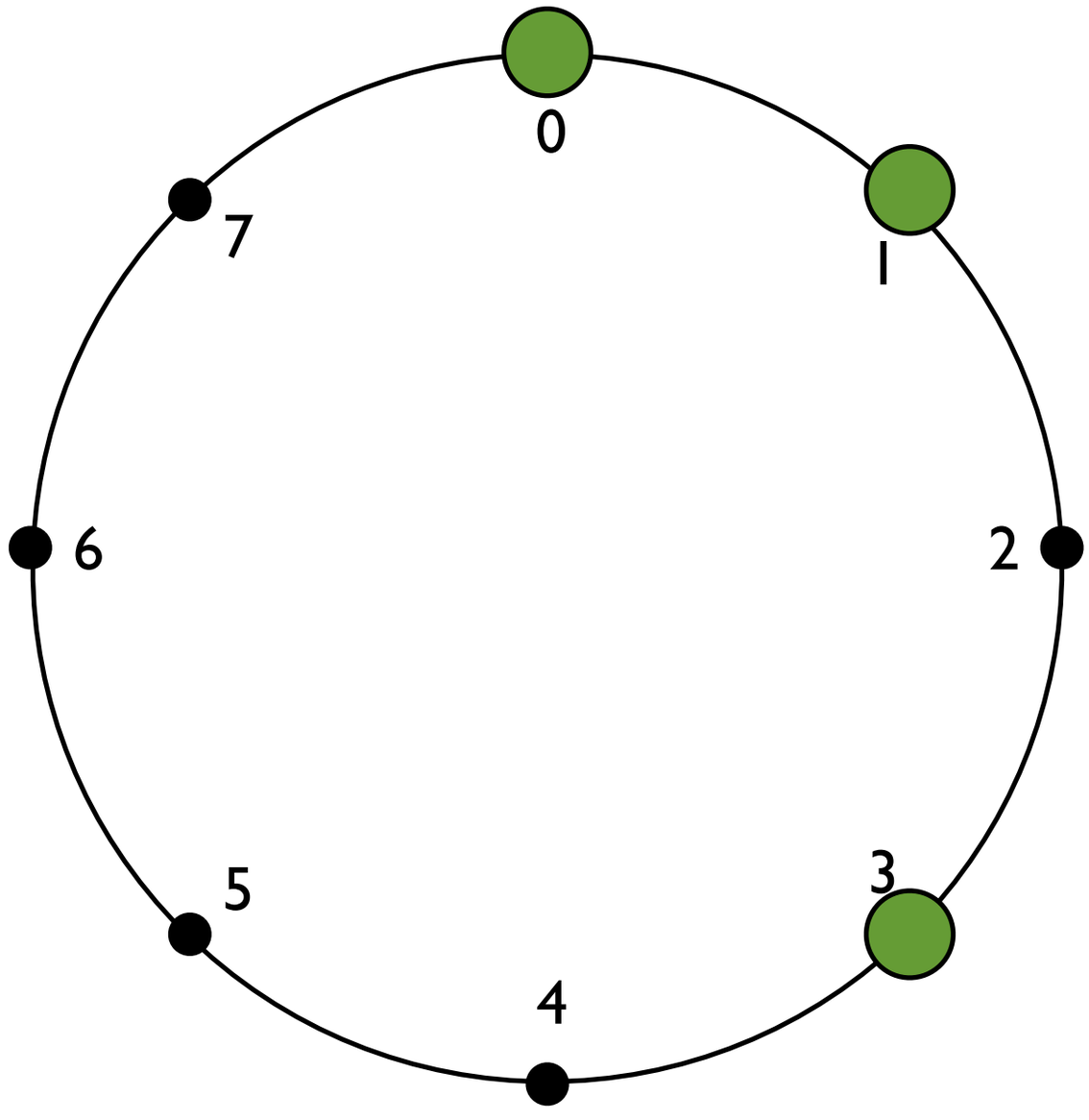
- Introduction
- Hadoop
- Chord

Why Chord?

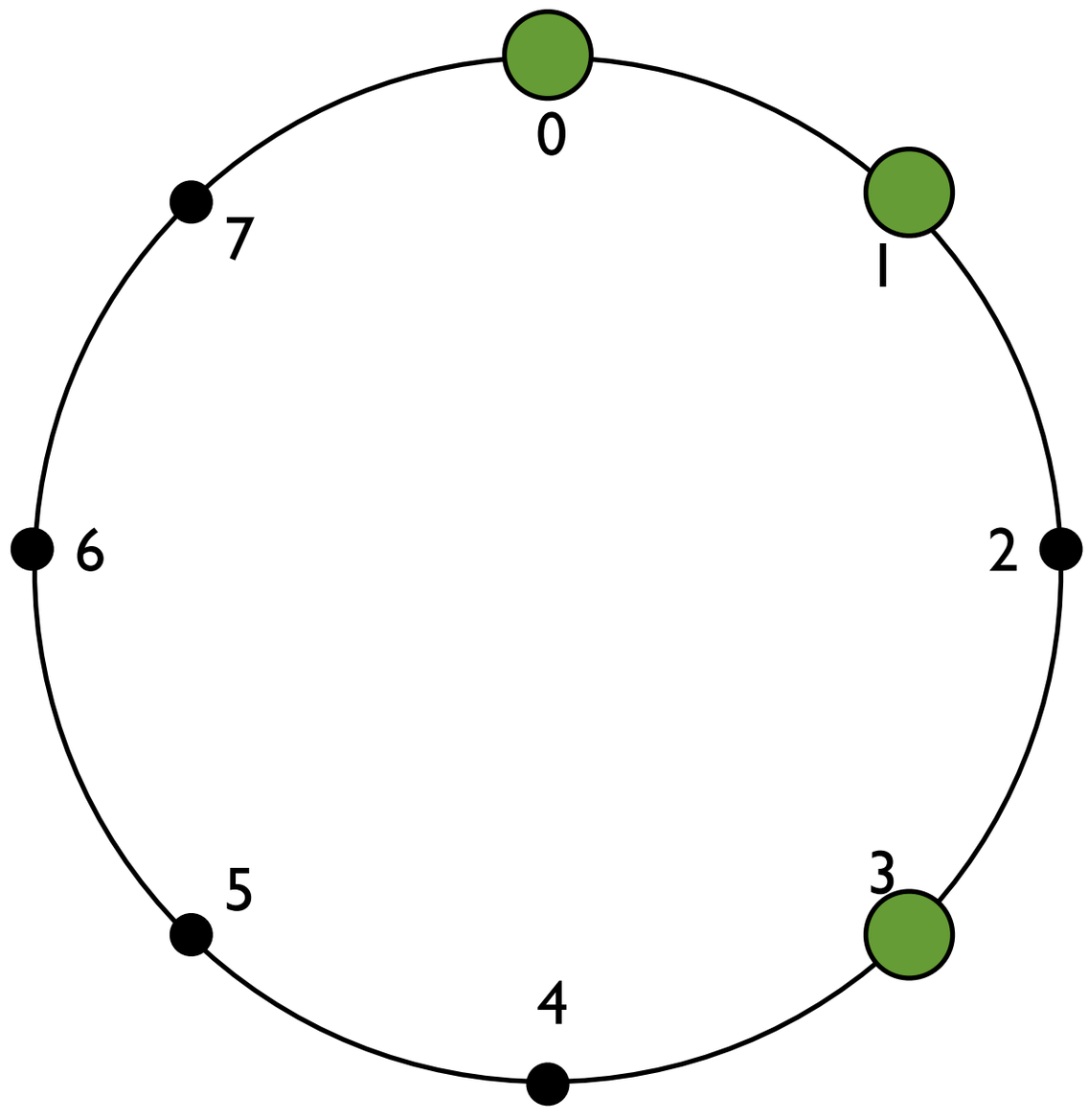
Chord



Chord



Chord

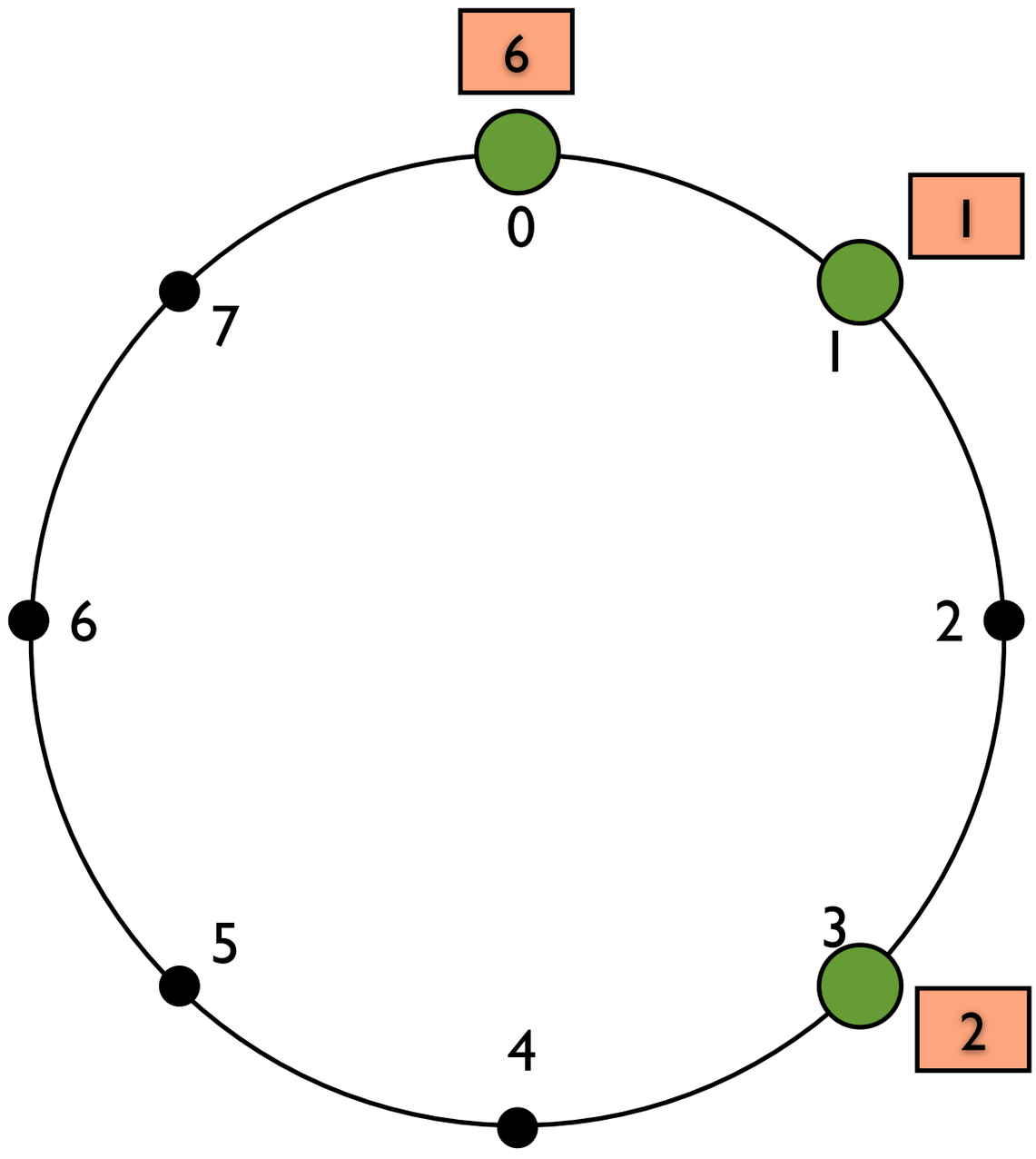


1

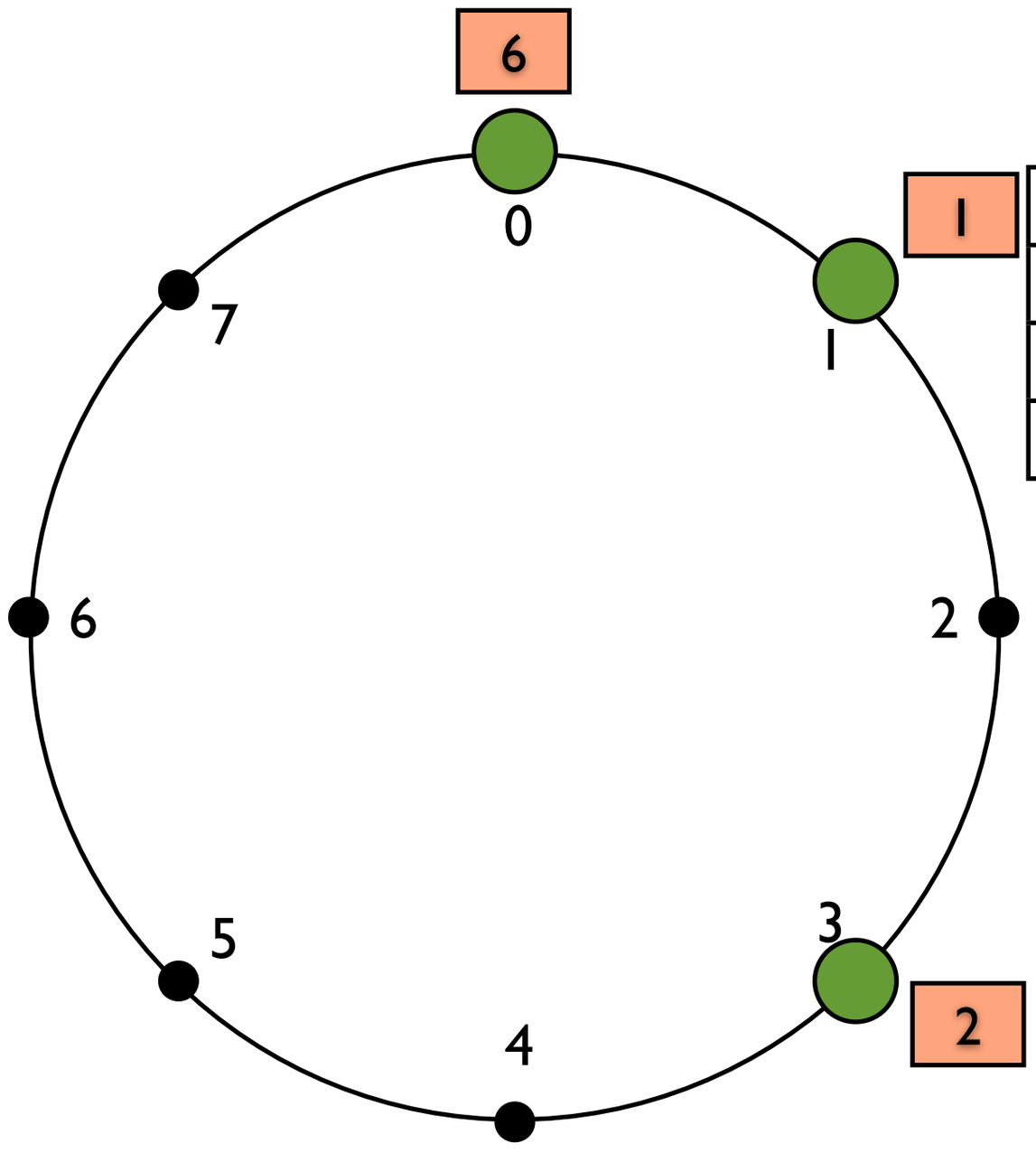
2

6

Chord

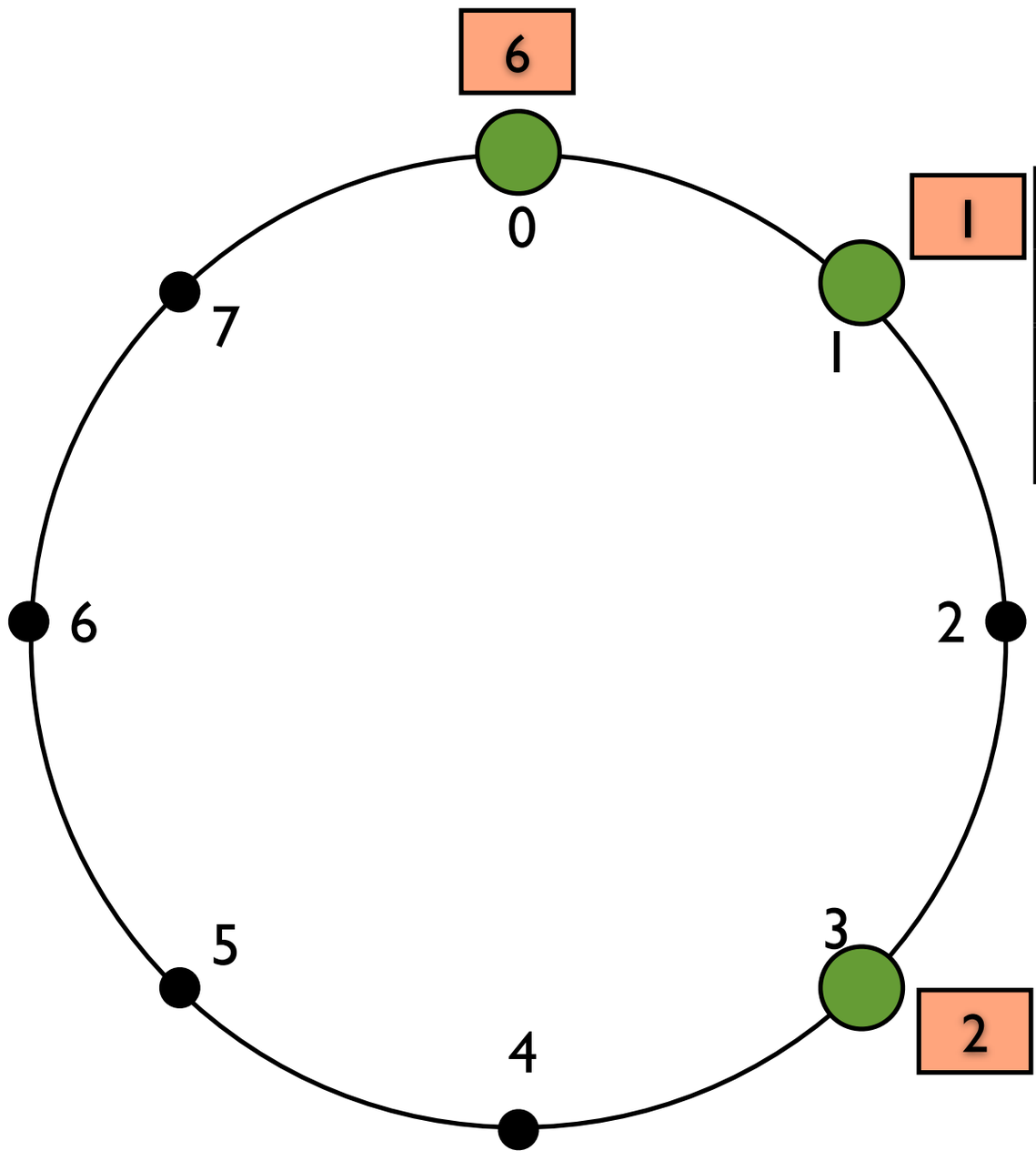


Chord



start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

Chord

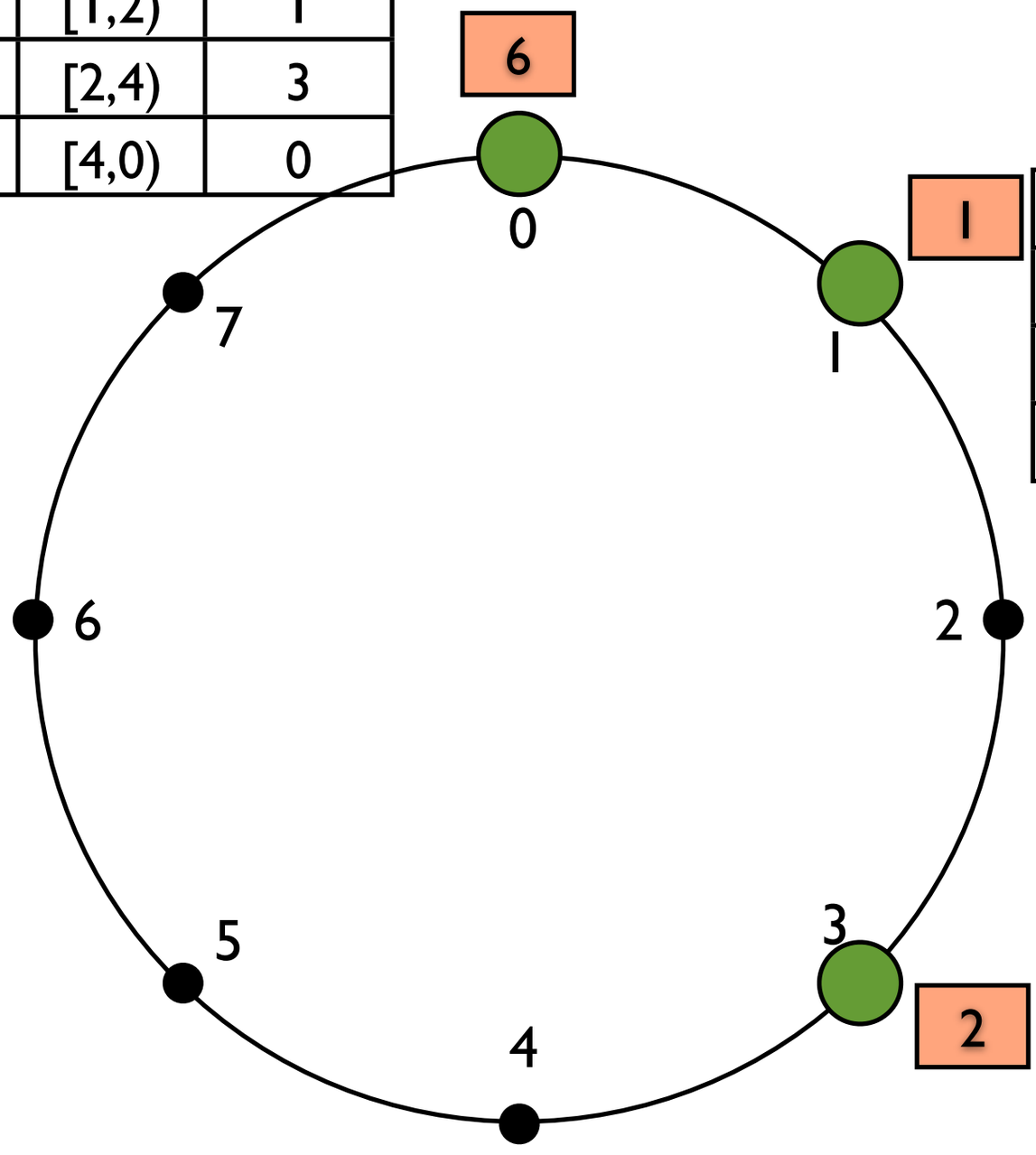


start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Chord

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0



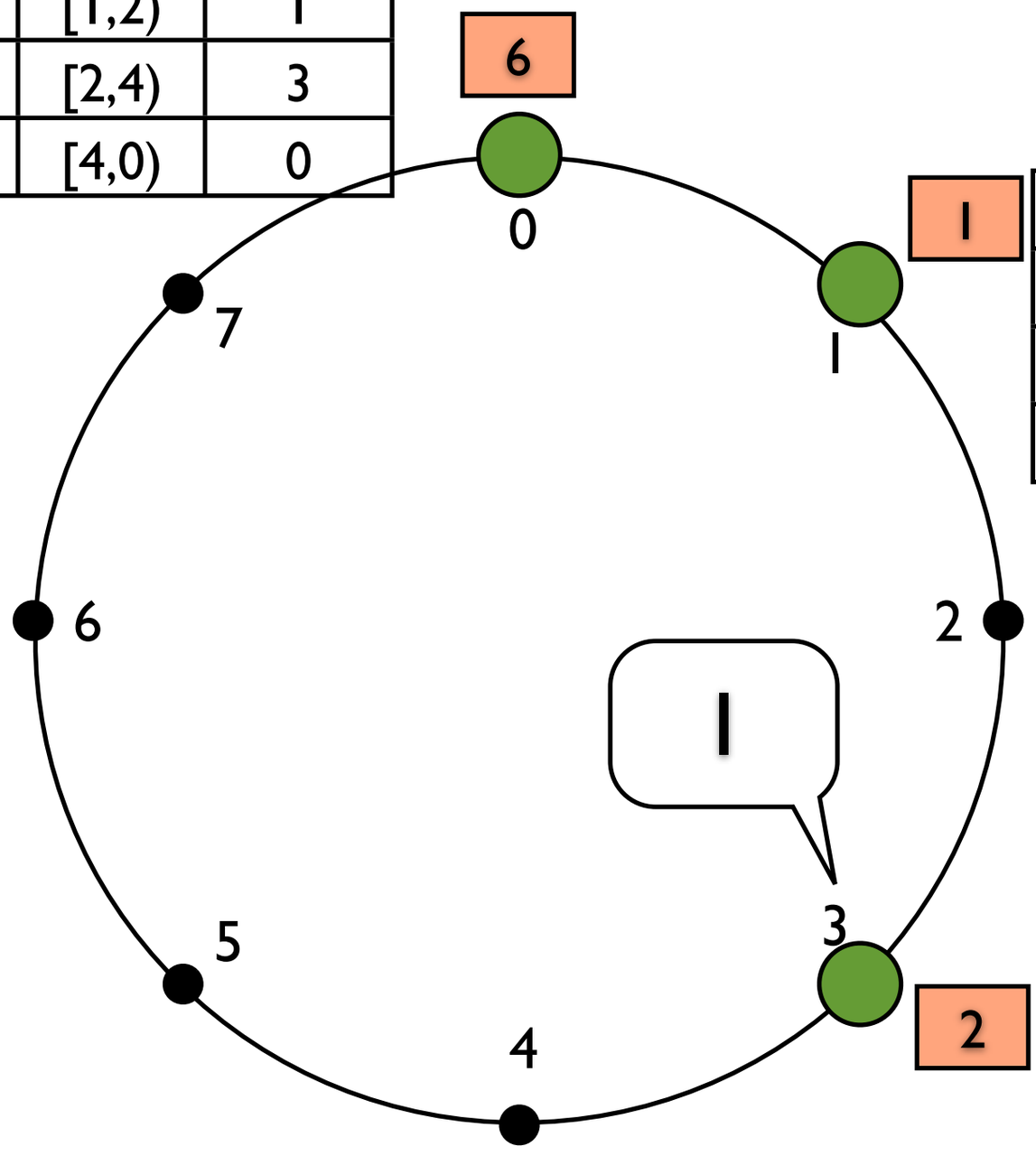
1

start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Chord

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0

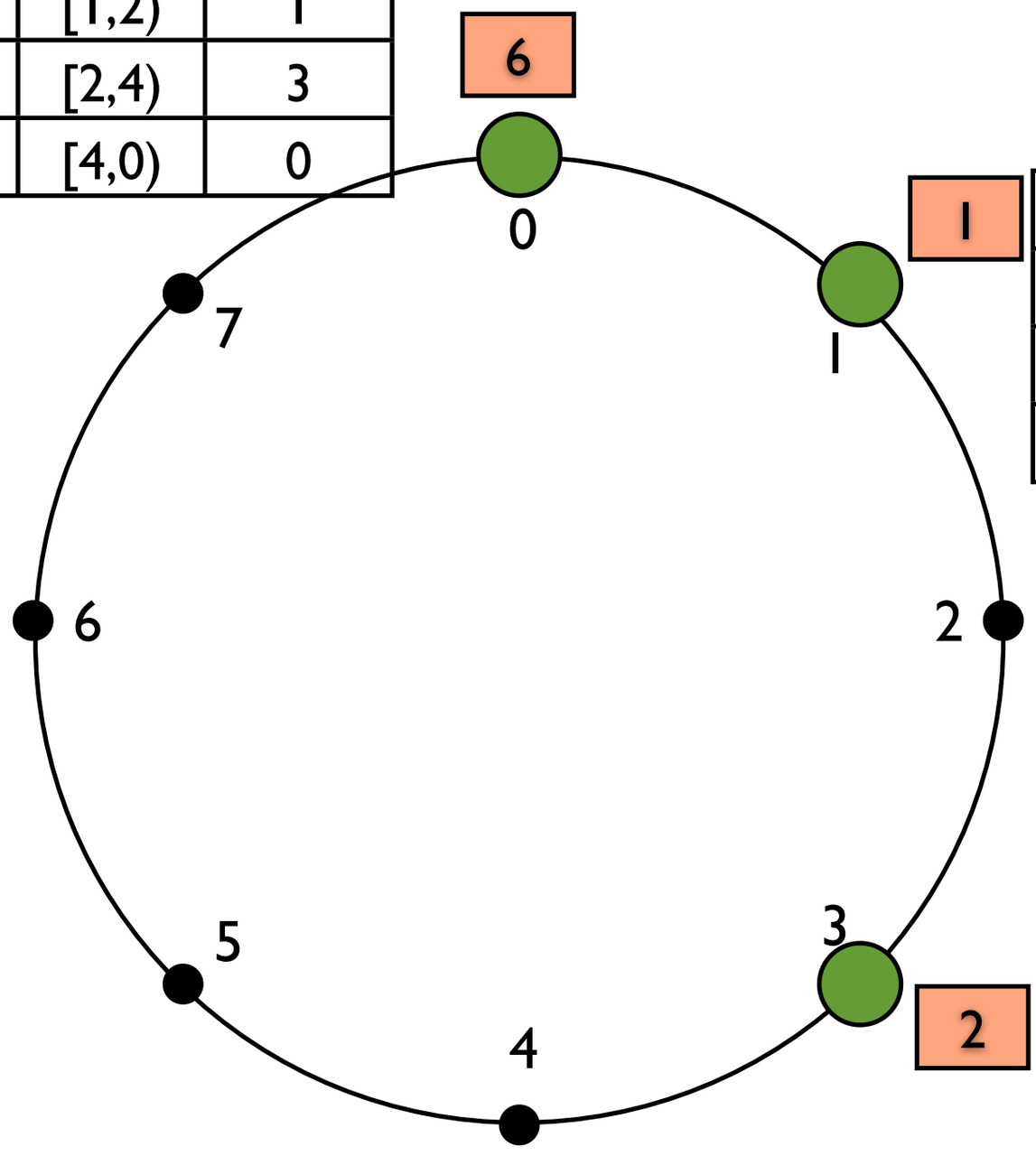


start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Importance: Complexity!

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0

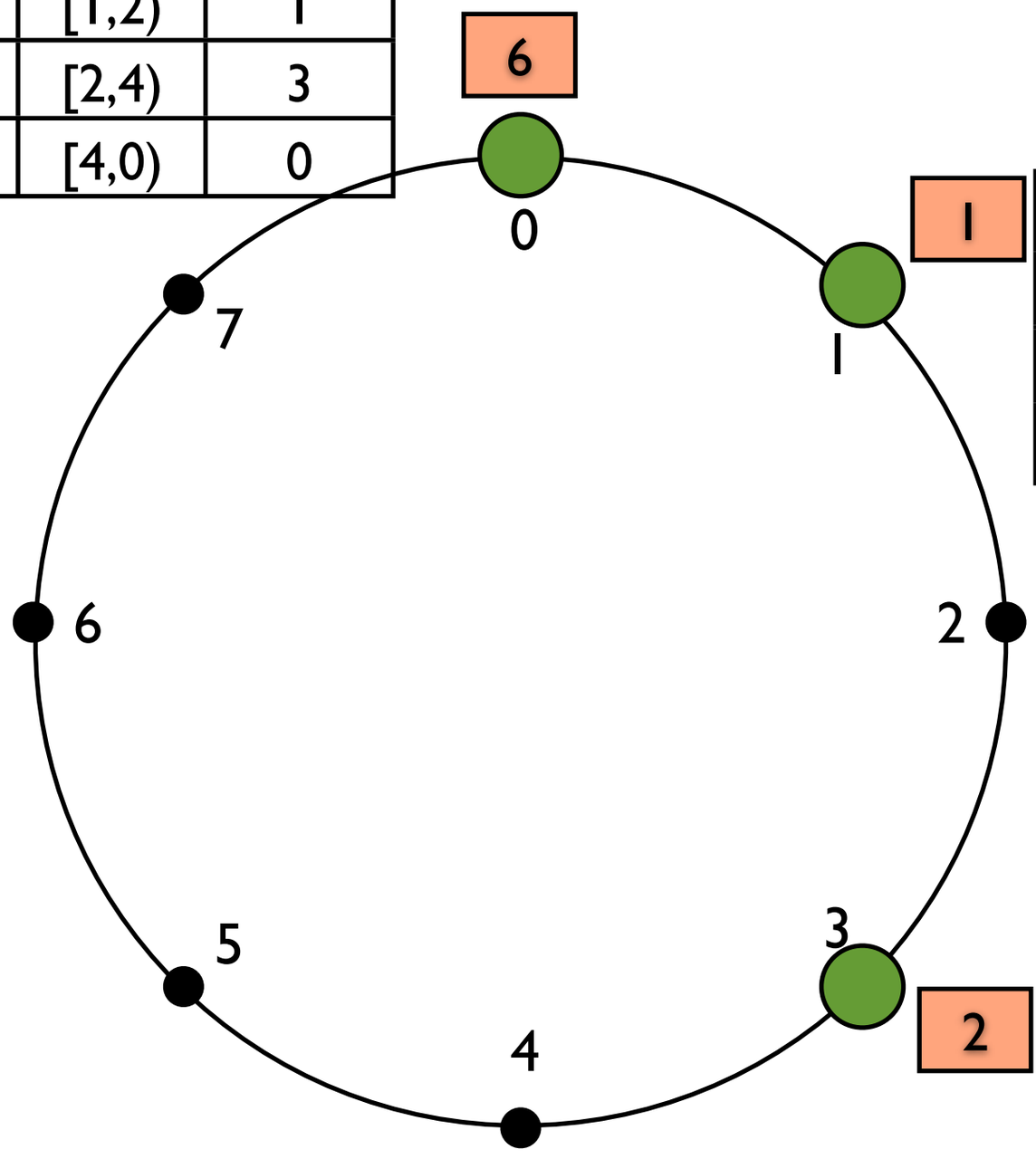


1	start	int.	succ.
	2	[2,3)	3
	3	[3,5)	3
	5	[5,1)	0

2	start	int.	succ.
	4	[4,5)	0
	5	[5,7)	0
	7	[7,3)	0

Node Joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0



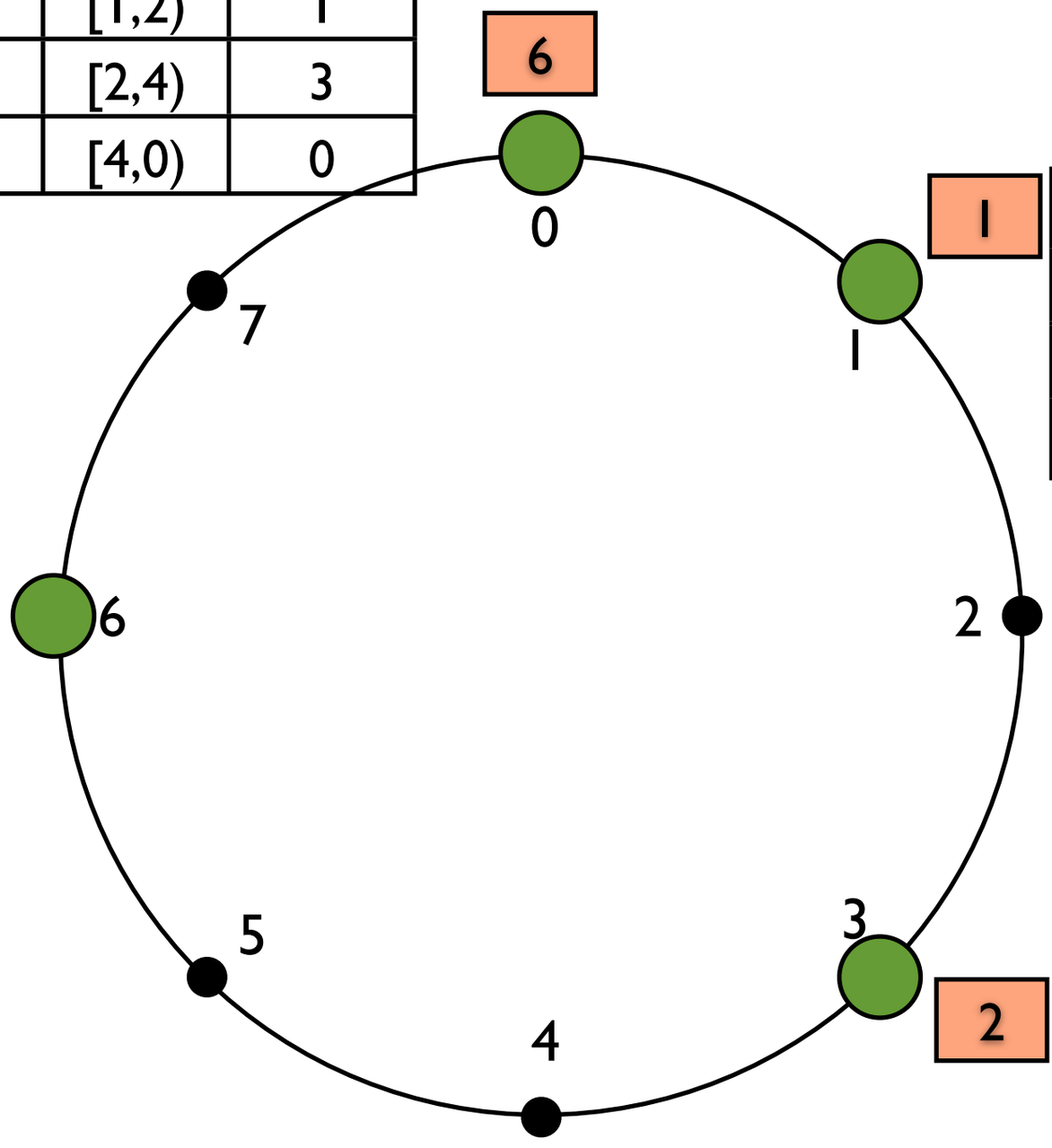
1

start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Node Joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0

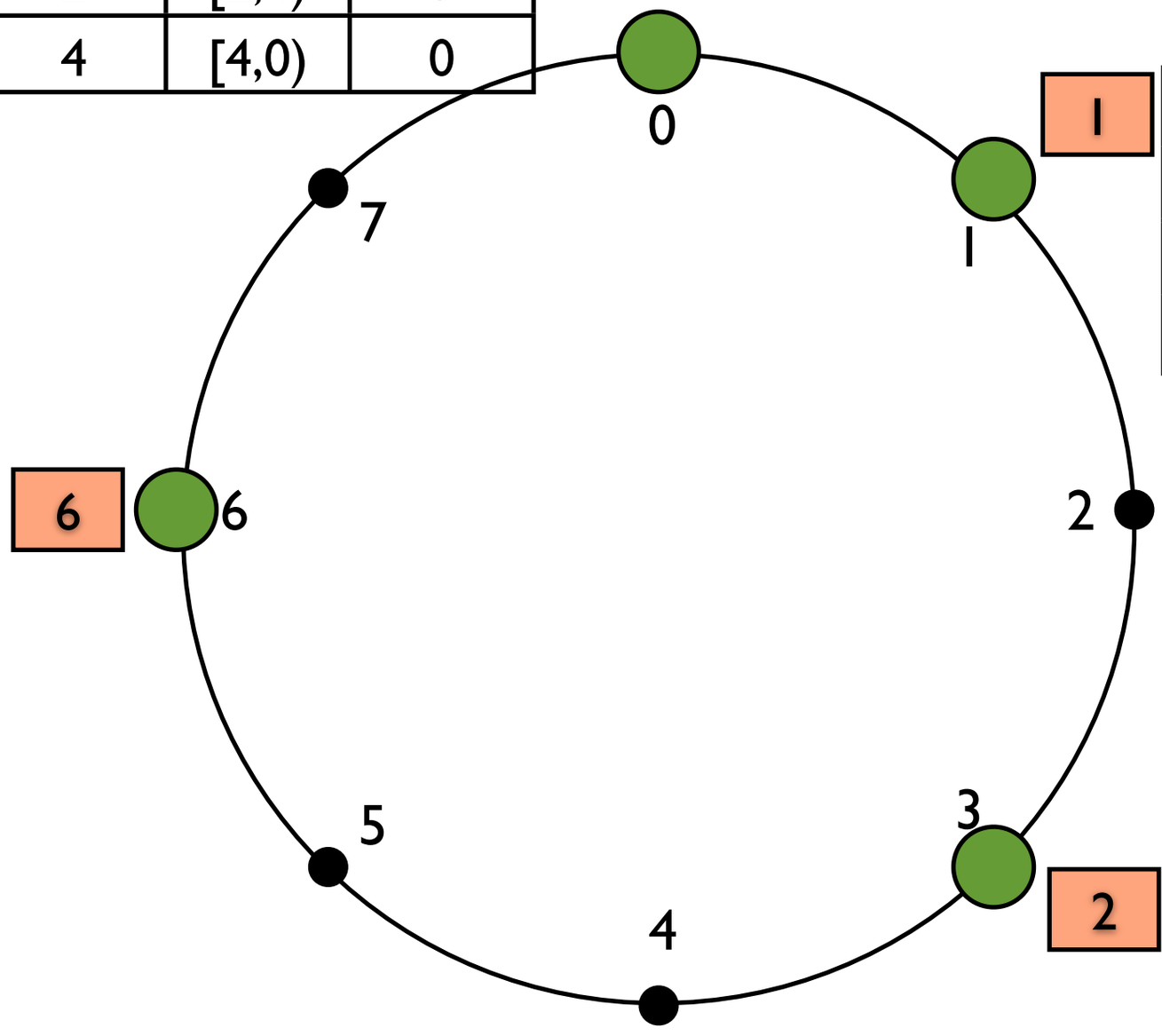


start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Node Joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0



1

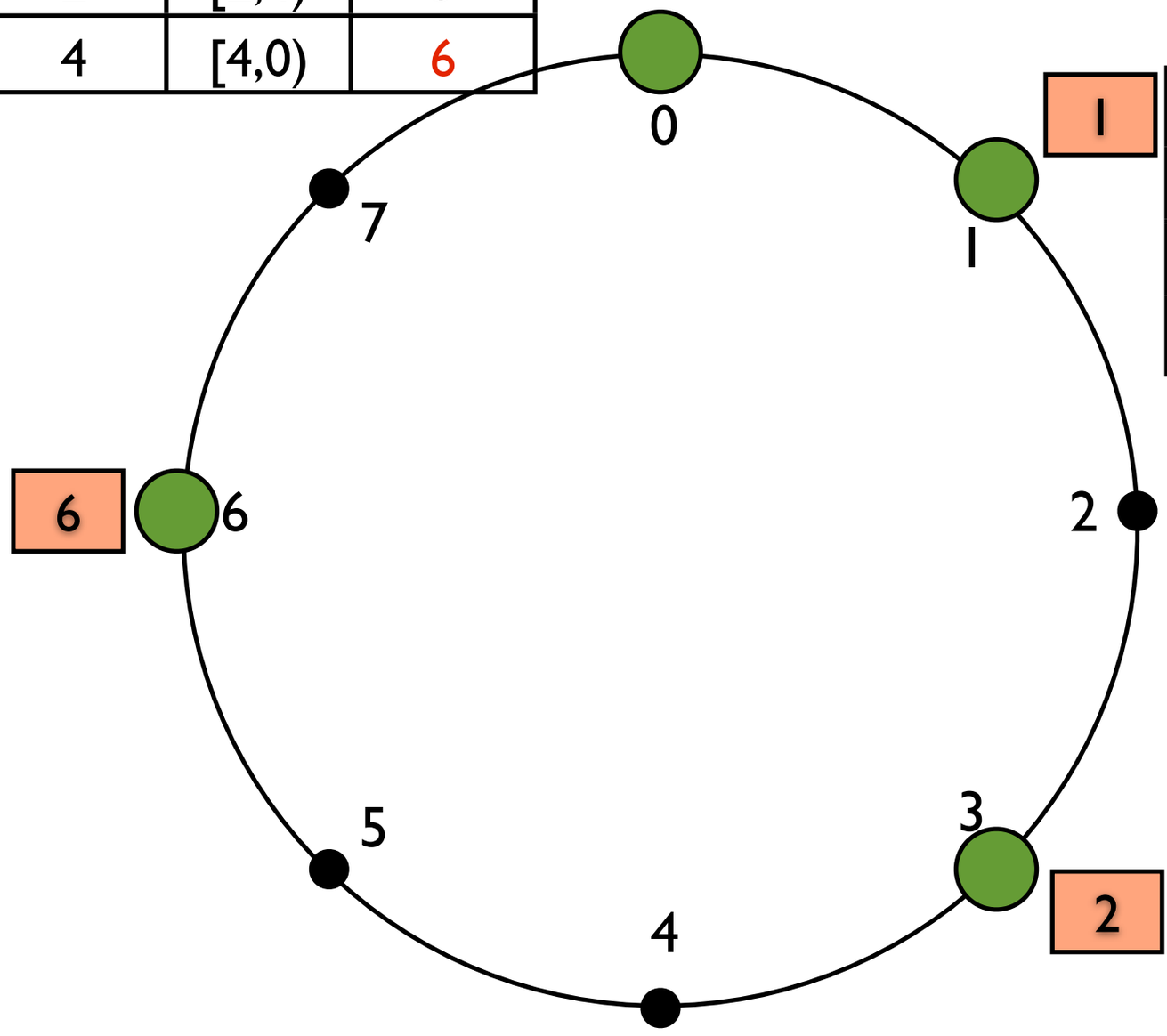
start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

2

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Node Joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	6

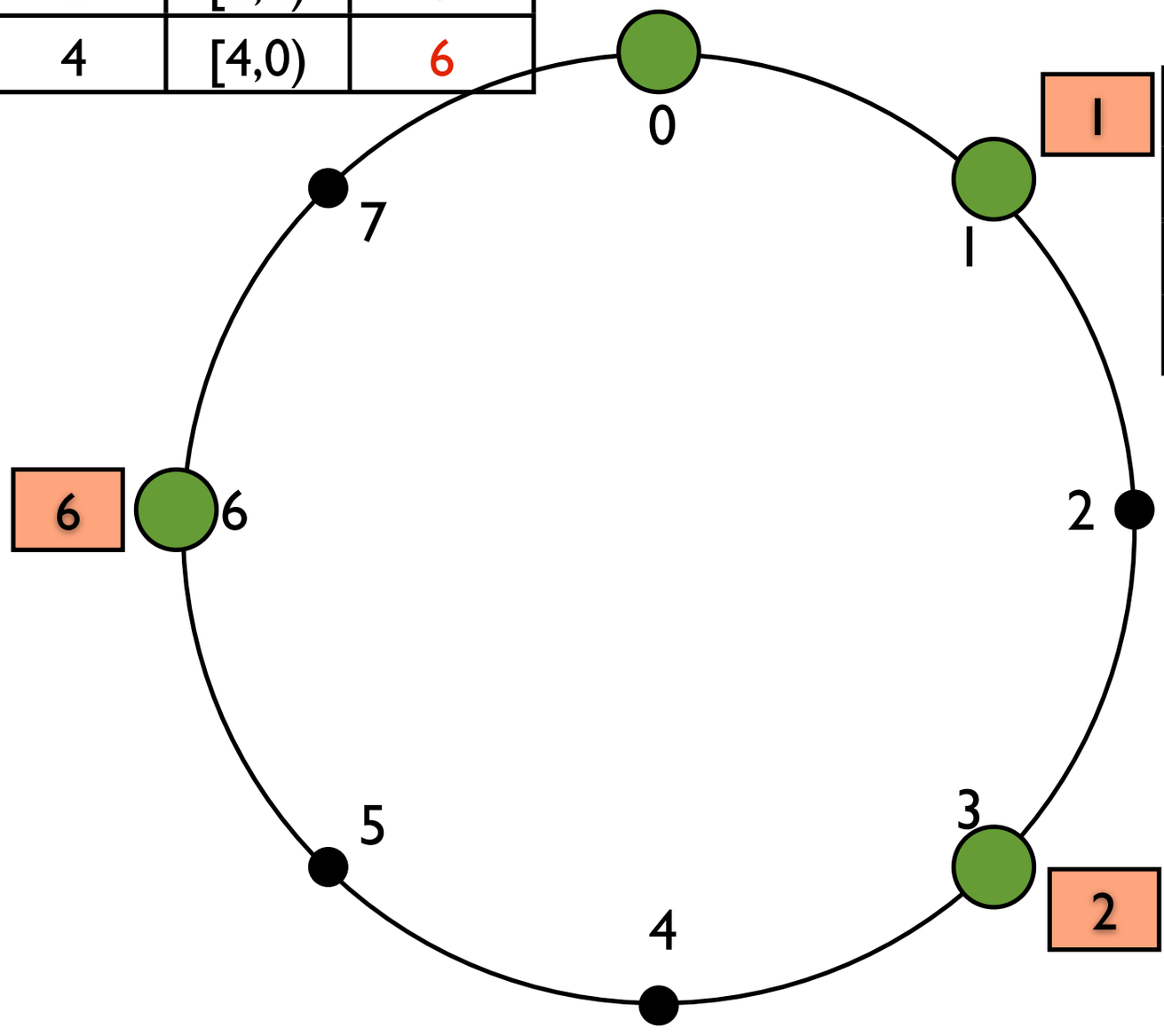


start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Node Joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	6



1

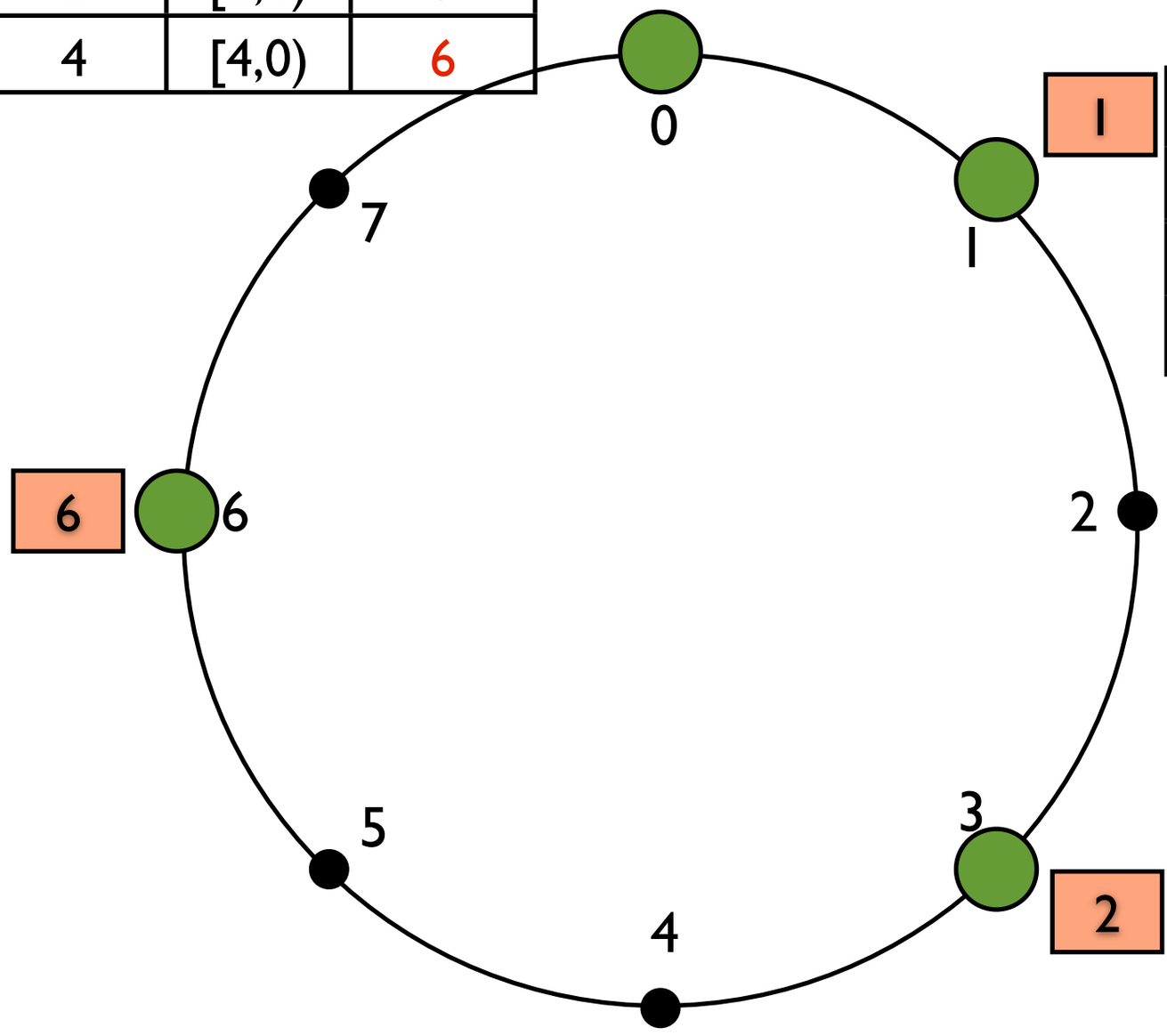
start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	6

2

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Node Joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	6



1

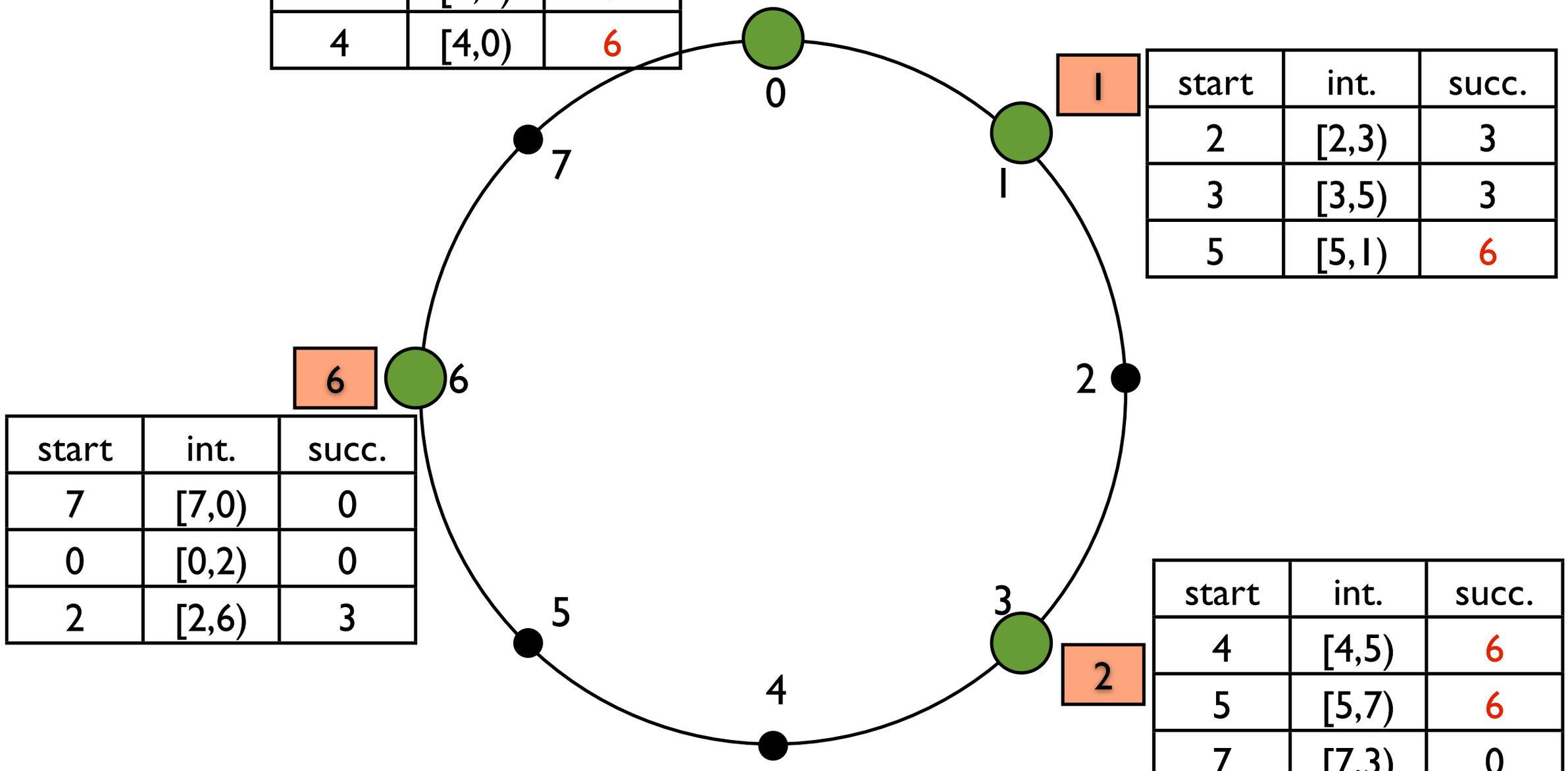
start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	6

2

start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Node Joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	6



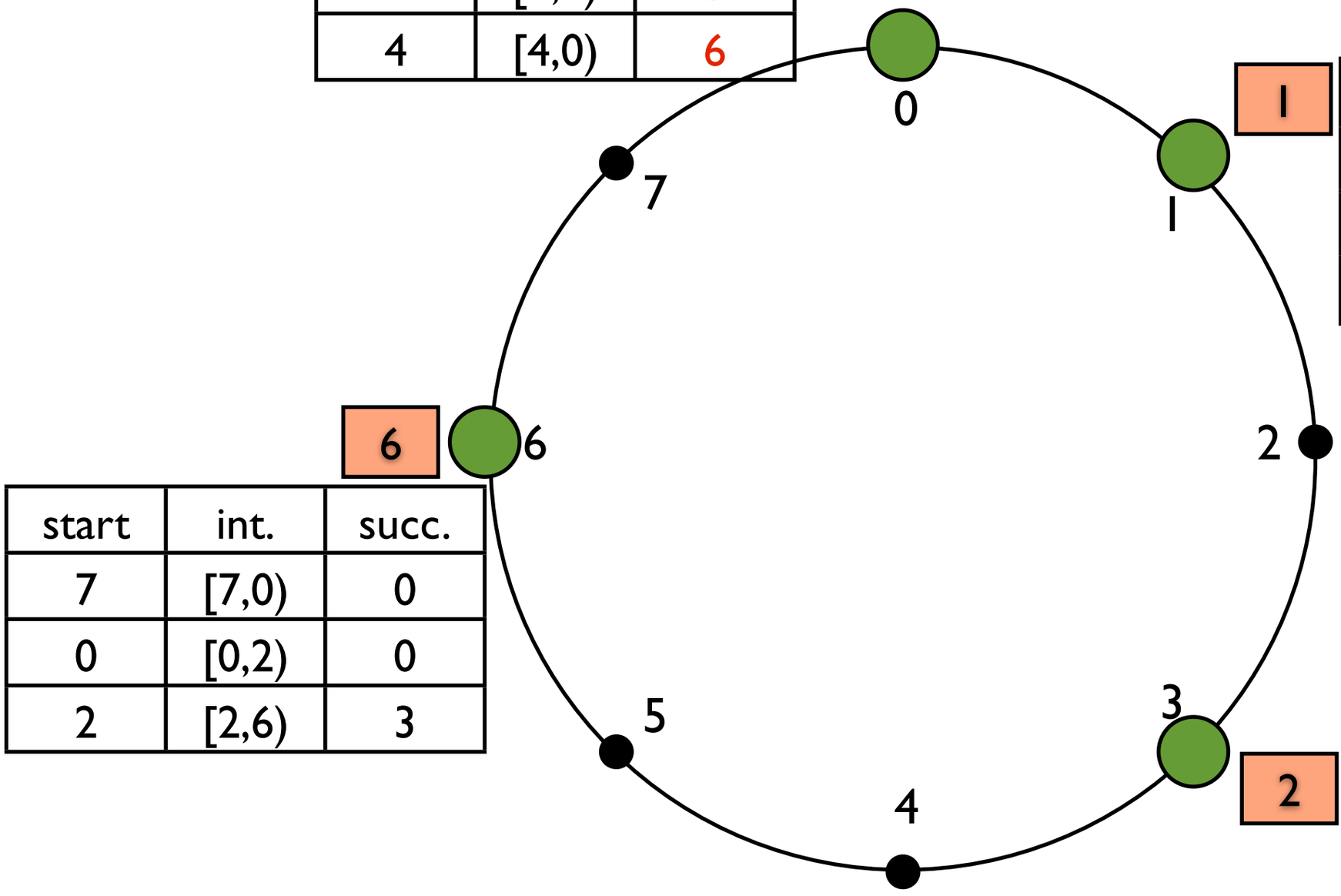
start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	6

start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3

start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Case Study: Node Leaves

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	6



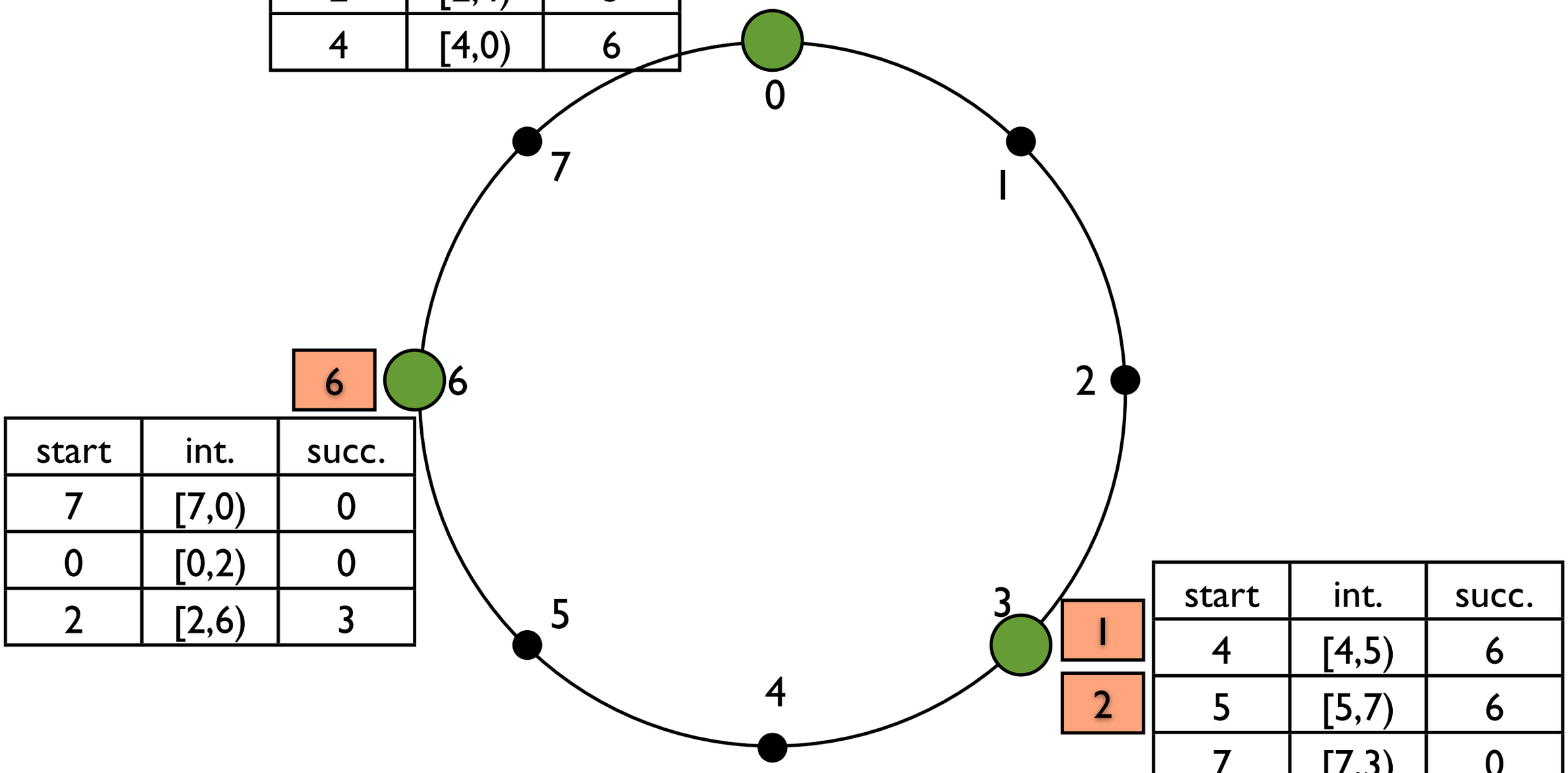
start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	6

start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3

start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Case Study: Node Leaves

start	int.	succ.
1	[1,2)	3
2	[2,4)	3
4	[4,0)	6



start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3

1
2

start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Chord Issues?