

COMP9313

Final

登登教育

-Elijah-

20T2

课程安排

	8月13日	8月14日
上	课程内容回顾	课程内容回顾
下	课程内容回顾	答疑

开卷 + 时间充足

我们复习的重点放在**理解概念、算法、流程**上

记忆性的东西到时候直接搜索 Lecture Notes

重点章节

Week	Content	
Week 1	Introduction	
Week 2	Hadoop HDFS	✓
Week 3	MapReduce	✓
Week 4	Spark	✓
Week 5	High Dimensional Similarity Search Product Quantization	✓
Week 7	Spark SQL	
Week 8	Spark ML	
Week 9	Data Stream	✓
Week 10	Recommand System	✓

没过在作业中出现过的章节

Week	Content	
Week 1	Introduction	
Week 2	Hadoop HDFS	
Week 3	MapReduce	
Week 4	Spark	
Week 5	High Dimensional Similarity Search Product Quantization	✓
Week 7	Spark SQL	
Week 8	Spark ML	
Week 9	Data Stream	✓
Week 10	Recommand System	✓

- 
- Hadoop HDFS

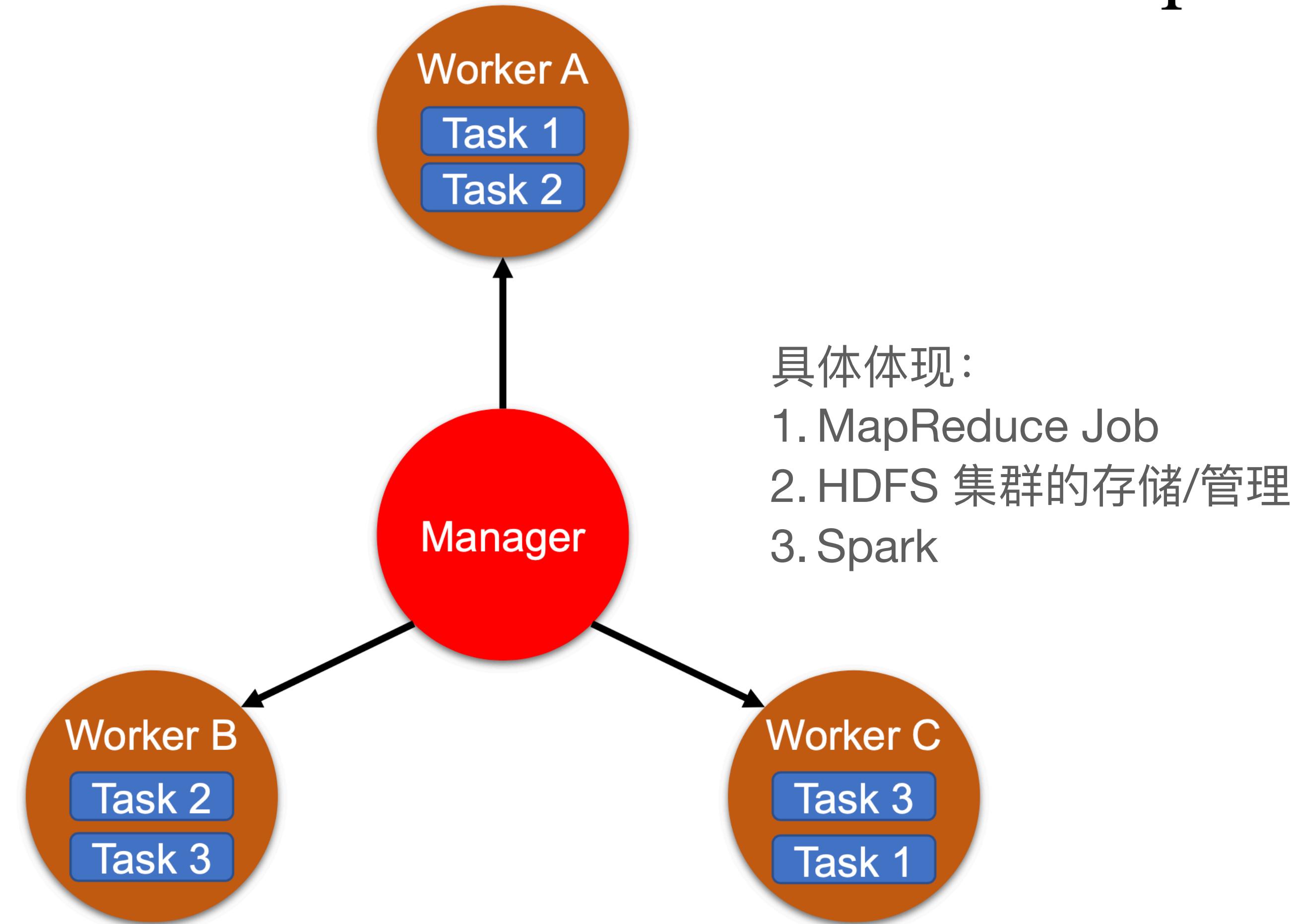
考点 1：hadoop 生态/框架的架构，重要组成部分

Hadoop Ecosystem

- Core of Hadoop
 - Hadoop distributed file system (HDFS)
 - MapReduce
 - YARN (Yet Another Resource Negotiator) (from Hadoop v2.0)
- Additional software packages
 - Pig
 - Hive
 - Spark
 - HBase
 - ...

考点 2 : Master-Slave 架构 (理解)

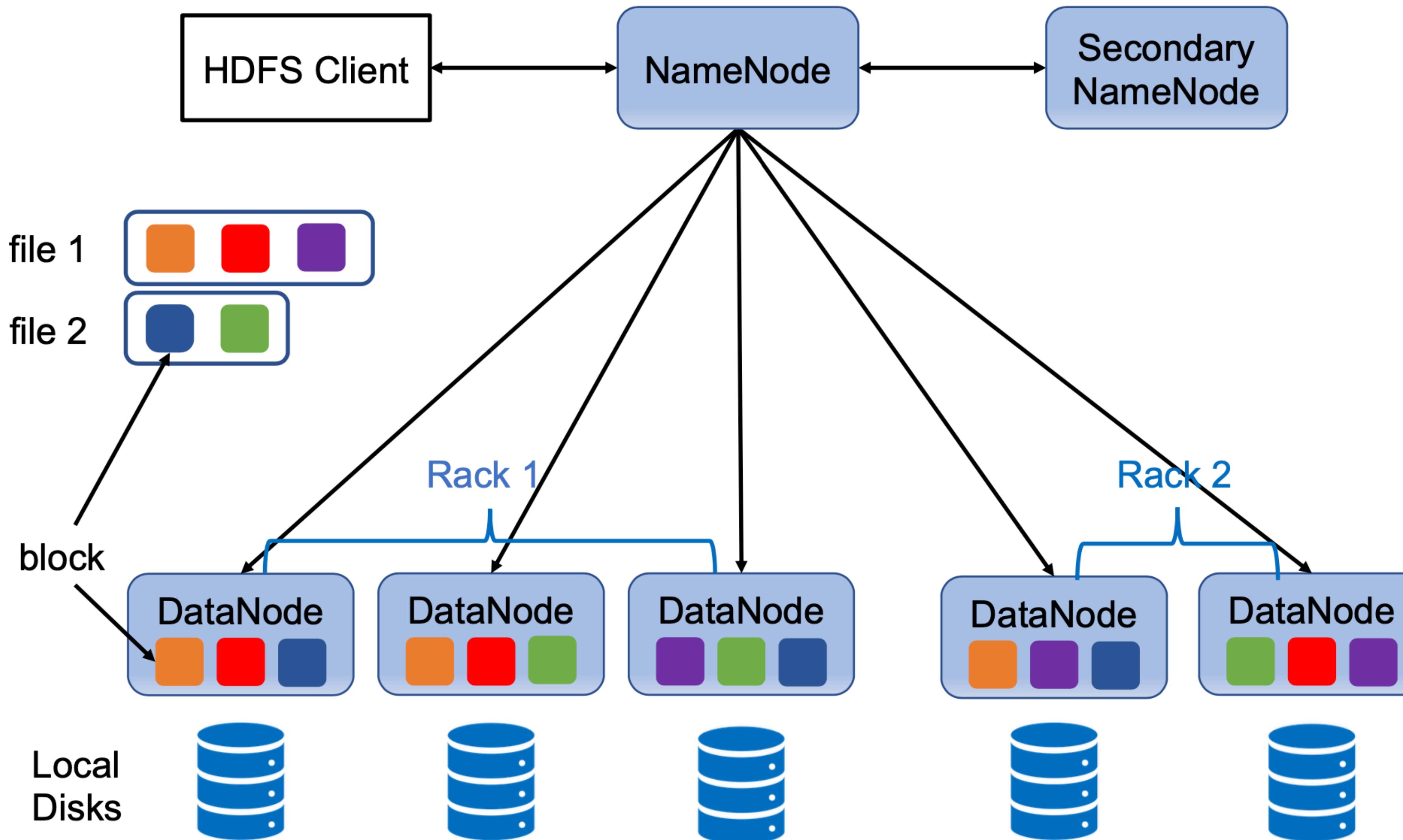
The Master-Slave Architecture of Hadoop



Hadoop Distributed File Systems (HDFS)

- HDFS is a file system that
 - follows master-slave architecture
 - allows us to store data over multiple nodes (machines) ,
 - allows multiple users to access data.
 - just like file systems in your PC
- HDFS supports
 - distributed storage
 - distributed computation
 - horizontal scalability

HDFS Architecture



考点 4：DN 1NN 2NN 的作用/区别/工作机制

DataNode

- A commodity hardware stores the data
 - Slave node
- Functions
 - stores actual data
 - perform the read and write requests
 - report the health to NameNode (heartbeat)

考点 4：DN 1NN 2NN 的作用/区别/工作机制

NameNode

- NameNode maintains and manages the blocks in the DataNodes (slave nodes).
 - Master node
- Functions: 元数据
 - records the metadata of all the files
 - FsImage: file system namespace
 - EditLogs: all the recent modifications
 - records each change to the metadata
 - regularly checks the status of datanodes
 - keeps a record of all the blocks in HDFS
 - if the DataNode failure, handle data recoveray

Secondary NameNode

- Take checkpoints of the file system metadata present on NameNode
 - It is not a backup NameNode!
- Functions:
 - Stores a copy of FsImage file and Editlogs
 - Periodically applies Editlogs to FsImage and refreshes the Editlogs.
 - If NameNode is failed, File System metadata can be recovered from the last saved FsImage on the Secondary NameNode.

理解为什么要引入 Secondary ?

1. 如果所有的 Record 都存内存 -> 一断电就没了
2. 如果所有的 Record 都存磁盘 -> 效率太低
3. 于是引入 fsimage edits 两个文件
4. 如果每次都是 NameNode 自己启动的时候合并 -> 启动速度太慢
5. 于是引入 Secondary Name Node 提醒他，帮他合并
6. NameNode 挂掉的时候，还可以把 Secondary NameNode 里面的文件拷贝过来进行补救

考点 4：DN 1NN 2NN 的作用/区别/工作机制

NameNode vs. DataNode

	NameNode	DataNode
Quantity	One	Multiple
Role	Master	Slave
Stores	Metadata of files	Blocks
Hardware Requirements	High Capacity Memory	High Volume Hard Drive
Failure rate	Lower	Higher
Solution to Failure	Secondary NameNode	Replications

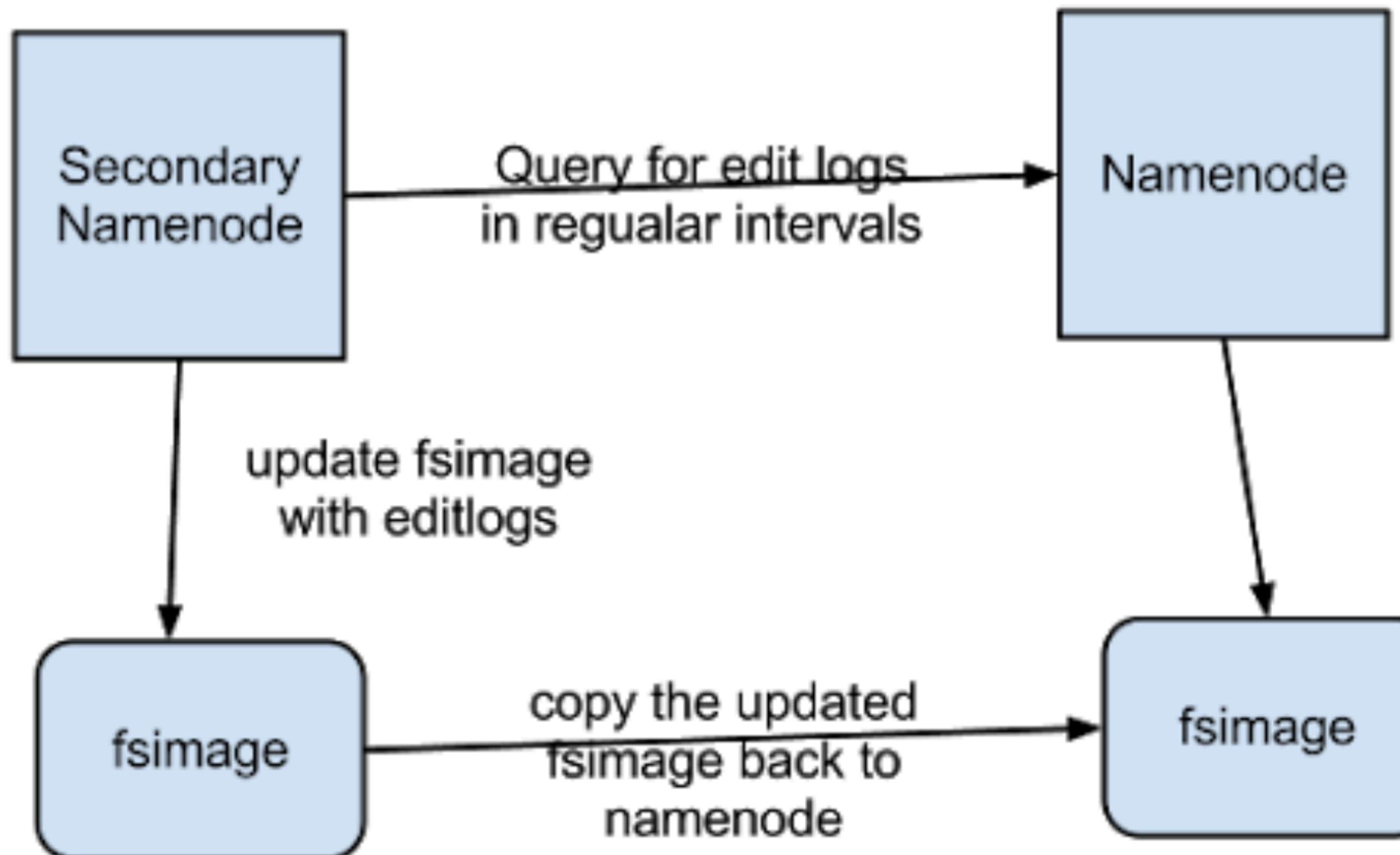
Sample Question 1

Question 1 HDFS

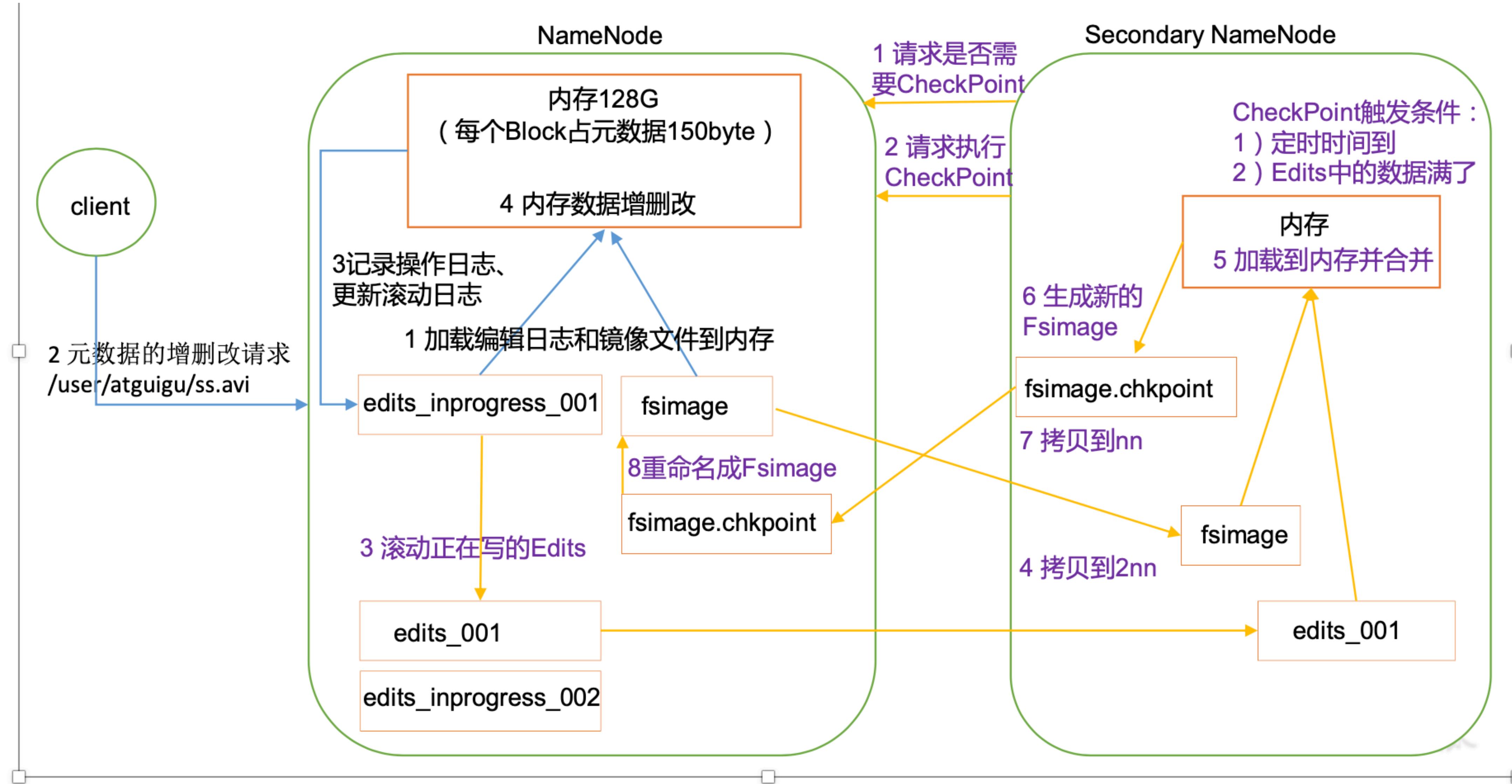
- Explain the difference between NameNode and DataNode.

考点 4：DN 1NN 2NN 的作用/区别/工作机制

NameNode vs. Secondary NameNode



考点 4：DN 1NN 2NN 的作用/区别/工作机制 拓展



NameNode 和 SecondaryNameNode 的工作机制（重点面试题）拓展

- 首先知道 NameNode 是用来存元数据的，元数据是通过两个不同的文件 edits, fdimage 共同存储的
- 其次 SecondaryNameNode 的任务就是提醒 NN 该合并了，并且帮他合并，以免 edits 这个 log 文件过大，拖慢 NN 启动速度

正式的流程：

1. NN 被启动的时候，**第一件事就是合并上次生成的 edits, fdimage 文件然后加载到内存**，并且生成一个**新的空的 edits_inprogress_001** 文件
2. 当 NameNode 中的数据需要修改更新时，这些更新的操作**首先**会被记录到这个 edits_inprogress_001 中，然后才在内存中作修改
3. 整个过程，**每到一定时间（一小时），或者每到一定的 transcation 次数时（这个次数时 2nn 每过一分钟去询问 NN 你到 100万 次了吗，这个也是个参数可以设置）**（取决于配置参数的设置），SecondaryNameNode 会发出 checkPoint 请求
4. NN 收到 checkpoint 请求后，会**另起一个 edits_inprogress_002 文件**记录后续的更新，此时把 edits_inprogress_001 (重命名为 edits_001 -001) 和 fdimage **拷贝给 SecondaryNameNode，供其合并**
5. SecondaryNameNode 合并之后生成新的 **fsimage.chkpoint** 文件传回给 NN
6. NN 把收到的 **fsimage.chkpoint** 重命名成 **fsimage** 如此往复

Blocks

- Block is a sequence of bytes that stores data
 - Data stores as a set of blocks in HDFS
 - Default block size is 128MB (Hadoop 2.x and 3.x)
 - A file is spitted into multiple blocks

File: 330 MB

Block a:
128 MB

Block b:
128 MB

Block c:
74 MB

File, Block and Replica

- A file contains one or more blocks
 - Blocks are different
 - Depends on the file size and block size
 - $\# = \left\lceil \frac{\text{file size}}{\text{block size}} \right\rceil$
- A block has multiple replicas
 - Replicas are the same
 - Depends on the preset replication factor

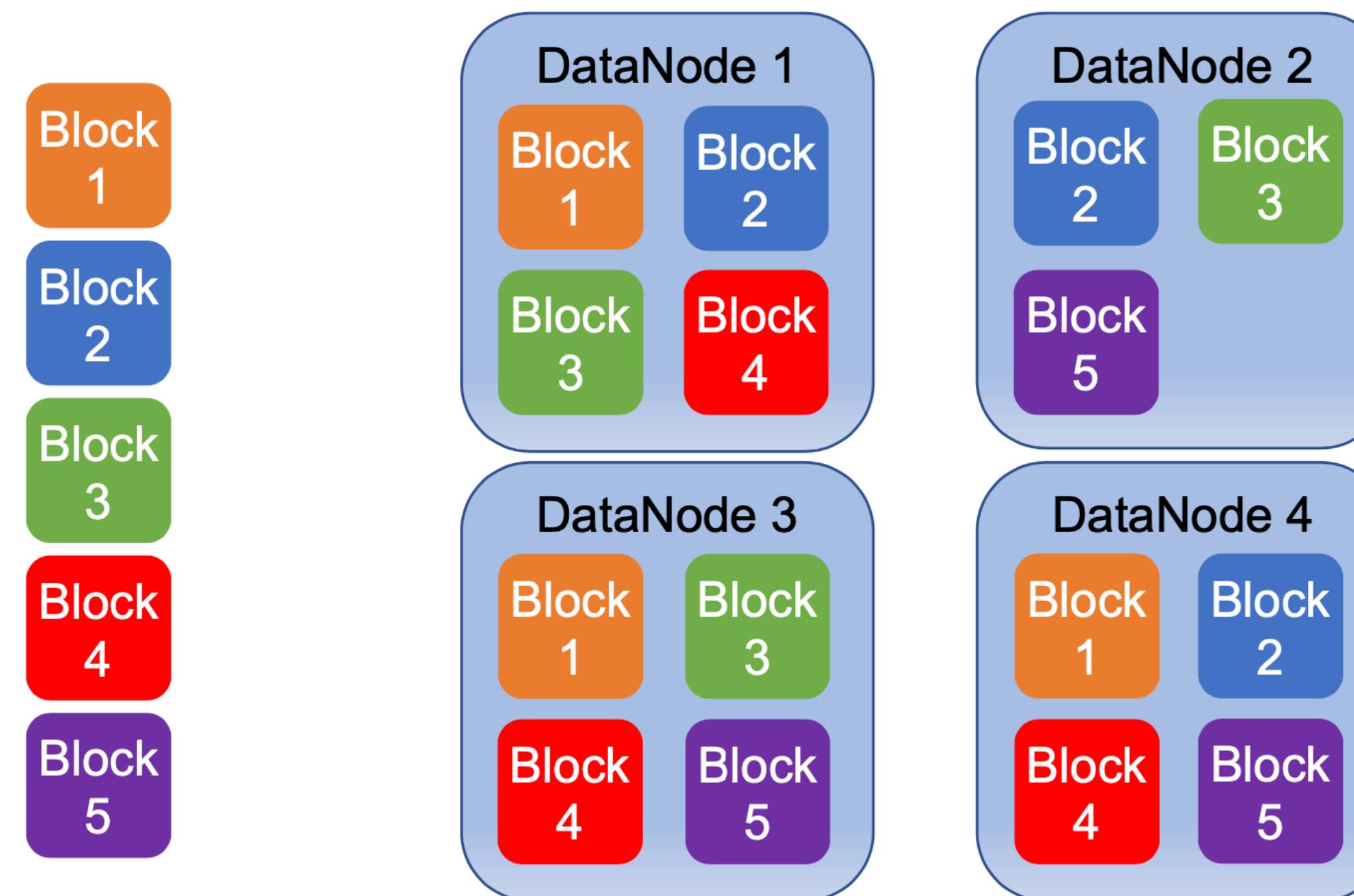
考点 5 : HDFS Block / replica / 机制

注意区分分块和副本的概念，大文件分好块了之后

每个块都会被备份，存在不同的 DataNode 中

Replication Management

- Each block is replicated 3 times and stored on different DataNodes



Sample Question 1

- Given a file of 500MB, let block size be 150MB, and replication factor=3. How much space do we need to store this file in HDFS?
Why?

考点 5.5 : HDFS 分块大小的确定/为什么要分大块? /分小块或者小文件有什么弊端?

Why Large Block Size?

- HDFS stores huge datasets
- If block size is small (e.g., 4KB in Linux), then the number of blocks is large:
 - too much metadata for NameNode
 - too many seeks affect the read speed
 - harm the performance of MapReduce too
- We don't recommend using HDFS for small files due to similar reasons.
 - Even a 4KB file will occupy a whole block.

考点 5.5：拓展 为什么是 128MB?

- 这个 128MB 取决于磁盘的传输速度， 因为普遍计算机硬盘传输速度为 100MB/s, 而对于HDFS而言， 寻址速度是传输速度的1/100是最理想的， 而普遍寻址时间为10ms，则 $10 * 100 = 1s$ ， 所以 $100MB/s * 1s = 100MB$ 是一个近似的比较理想的块大小， 于是取整 128MB
- 如果 server 硬盘速度很快， 可以增加这个大小

What about Simultaneous Failure?

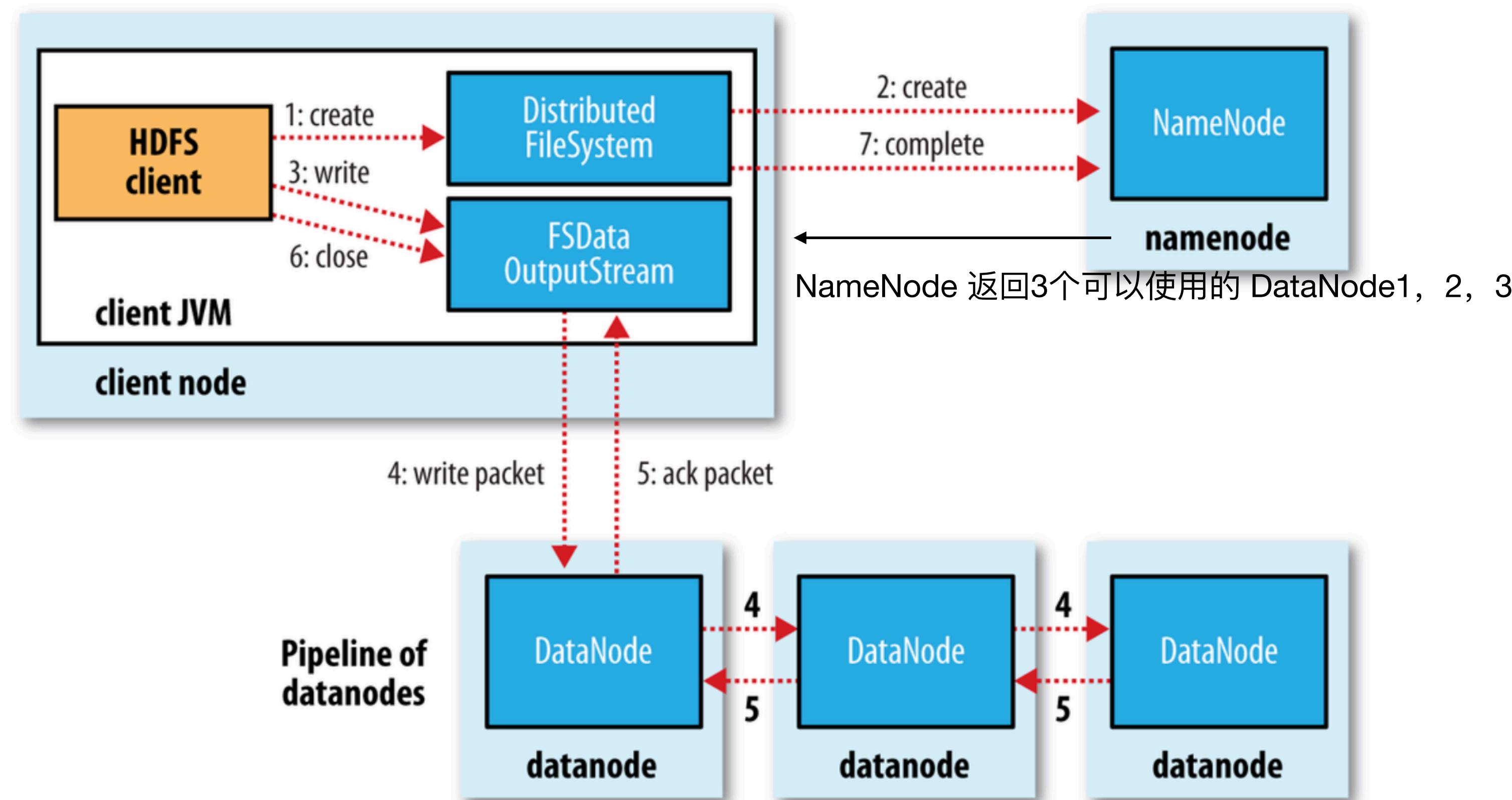
- In general
 - $L1(k,N) = k*B - 2*L2(k,N) - 3*L3(k,N)$
 - $L2(k,N) = 2*L1(k-1,N)/(N-k+1) + L2(k-1,N) - L2(k-1,N)/(N-k+1)$
 - $L3(k,N) = L2(k-1,N)/(N-k+1) + L3(k-1,N)$
- Let $N = 4000$, $B = 750$, we have

Failed Nodes	1 st replicas lost	2 nd replicas lost	3 rd replicas lost
50	36,587	454	2
100	71,332	1,811	15
150	104,272	4,037	52
200	135,441	7,095	123

考点 7：HDFS 读写流程

Write in HDFS

- Create file – Write file – Close file

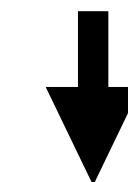


考点 7：HDFS 读写流程

Write in HDFS

- There is only **single** writer allowed at any time
- The blocks are writing **simultaneously**
- For one block, the replications are replicating **sequentially**
- The choose of DataNodes is random, based on replication management policy, rack awareness, ...

客户端向 NameNode 请求上传文件的时候， NameNode 挑选哪些 DataNode ?

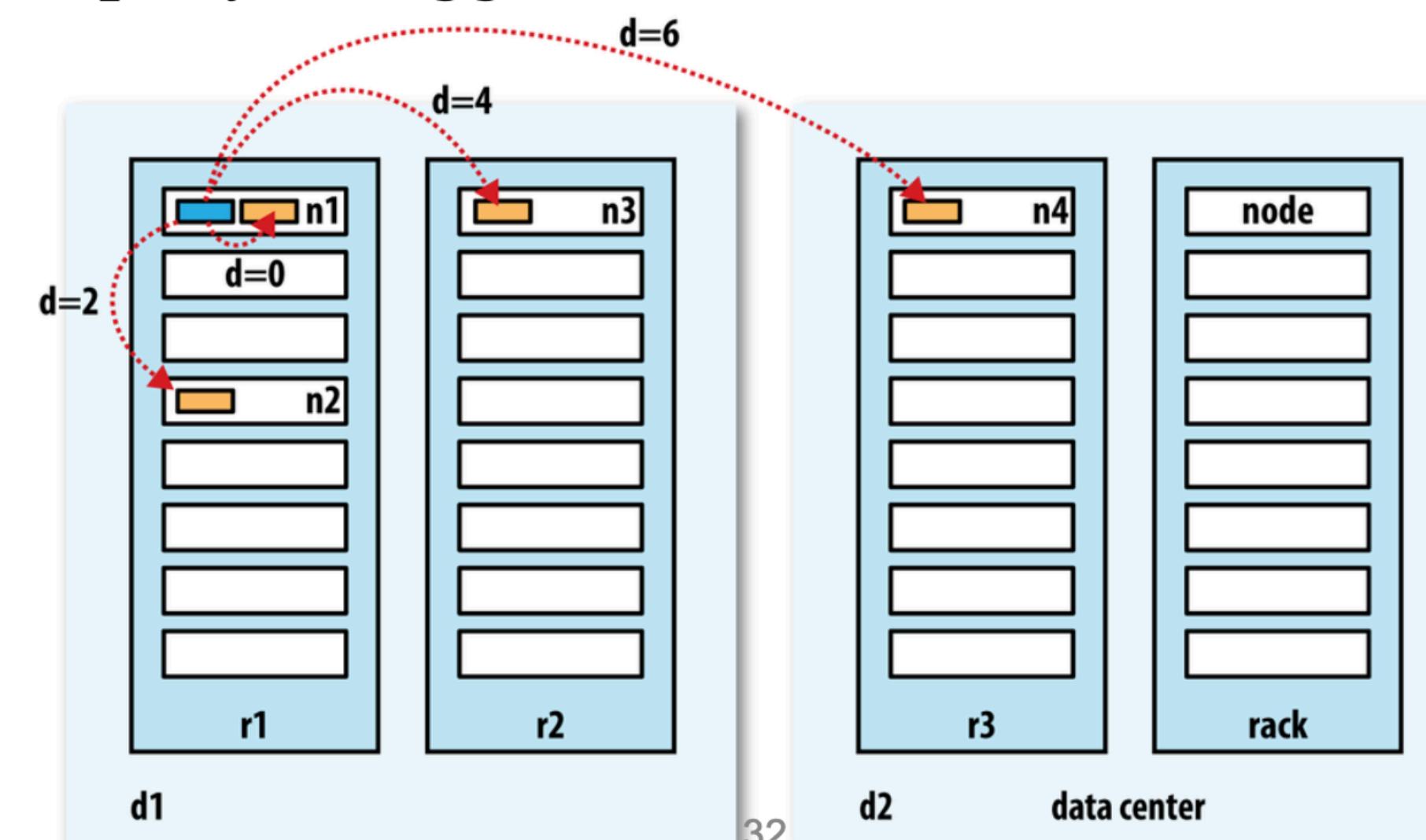


机架感知 rack awareness

考点 8：HDFS 机架感知/网络拓扑

Why Rack Awareness?

- Reduce latency
 - Write: to 2 racks instead of 3 per block
 - Read: blocks from multiple racks
- Fault tolerance
 - Never put your eggs in the same basket

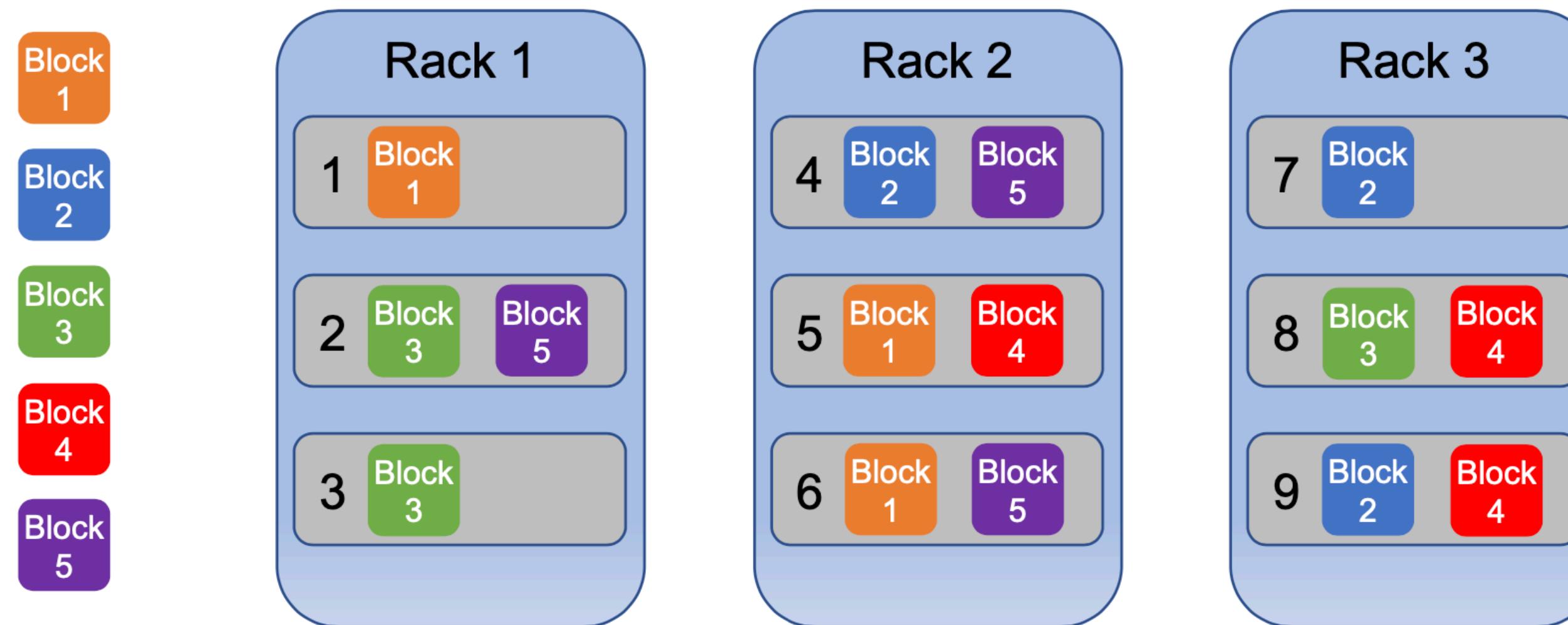


考点 8：HDFS 机架感知/网络拓扑

影响因素：网络 I/O 传输

Rack Awareness Algorithm

- If the replication factor is 3:
 - 1st replica will be stored on the local DataNode
 - 2nd on a different rack from the first.
 - 3rd on the same rack as 2nd, but on a different node.



考点 8：HDFS 机架感知/网络拓扑

Hadoop 3.2.1 官方解释：

https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Data_Replication

For the common case, when the replication factor is three, HDFS's placement policy is to put one replica on the local machine if the writer is on a datanode, otherwise on a random datanode in the same rack as that of the writer, another replica on a node in a different (remote) rack, and the last on a different node in the same remote rack.

考点 8：HDFS 机架感知/网络拓扑

网络拓扑计算距离

如何计算两个节点之间的距离？

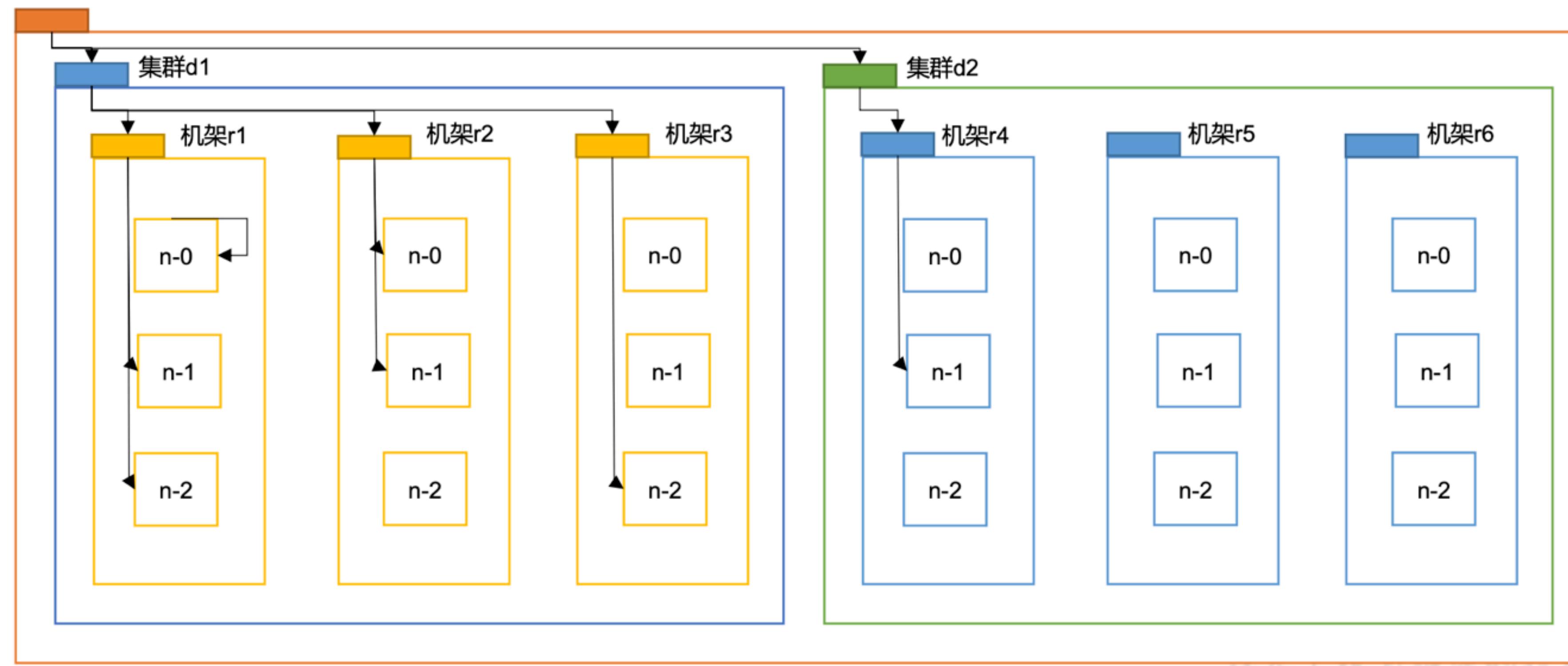
节点之间的距离 = 两个节点到达最近的公共路由/交换机的距离之和

Distance(/d1/r1/n0, /d1/r1/n0)=0 (同一节点上的进程)

Distance(/d1/r1/n1, /d1/r1/n2)=2 (同一机架上的不同节点)

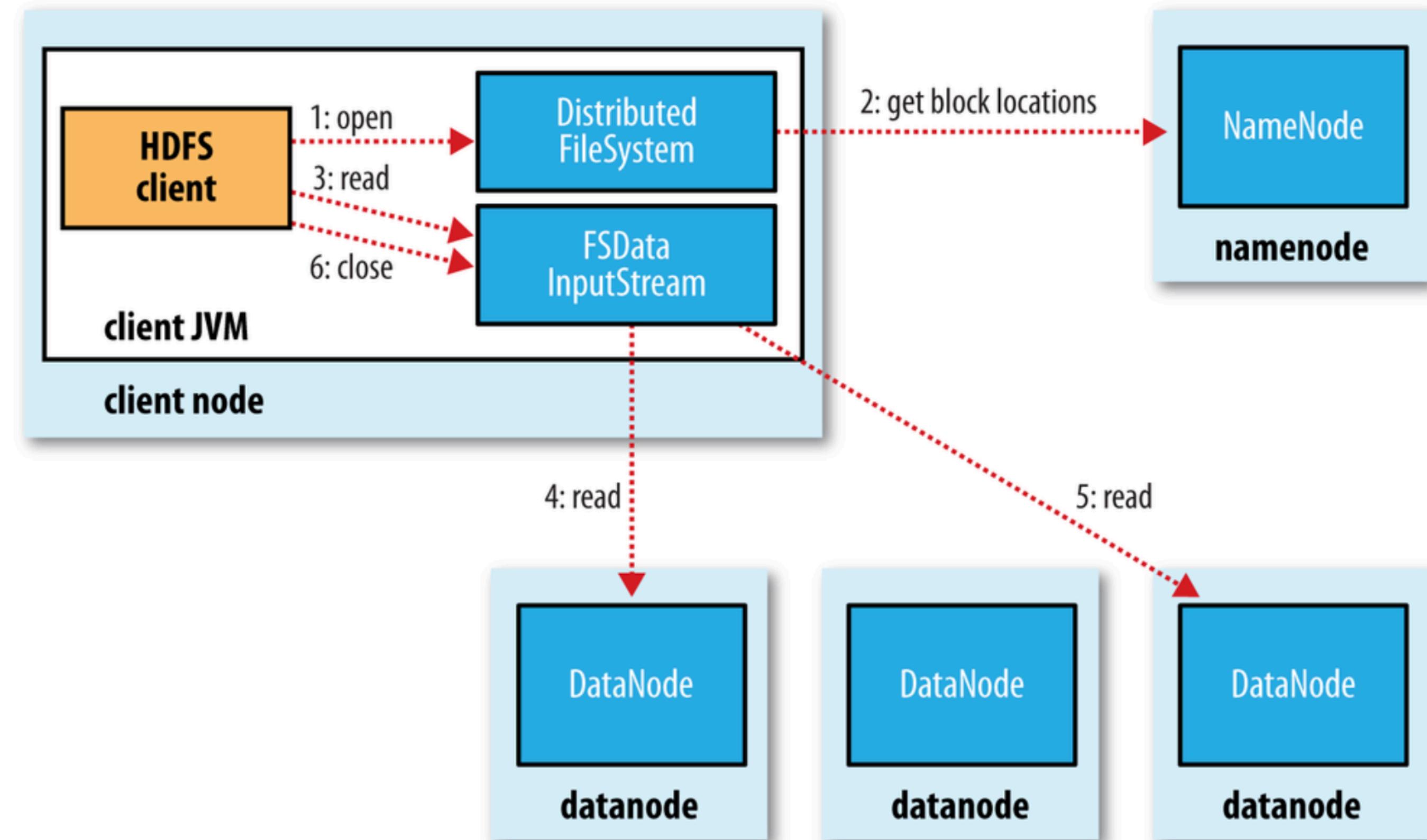
Distance(/d1/r2/n0, /d1/r3/n2)=4 (同一数据中心不同机架上的节点)

Distance(/d1/r2/n1, /d2/r4/n1)=6 (不同数据中心的节点)



考点 7：HDFS 读写流程

Read in HDFS



考点 7：HDFS 读写流程

Read in HDFS

- Multiple readers are allowed to read at the same time
- The blocks are reading **simultaneously**
- Always choose the **closest** DataNodes to the client (based on the network topology)
- Handling errors and corrupted blocks
 - avoid visiting the dataNode again
 - report to NameNode

考点 9 : HDFS Erasure Coding

知识点：什么是 HDFS Erasure Coding?
相对于 Replication 有什么特点？

HDFS Erasure Coding

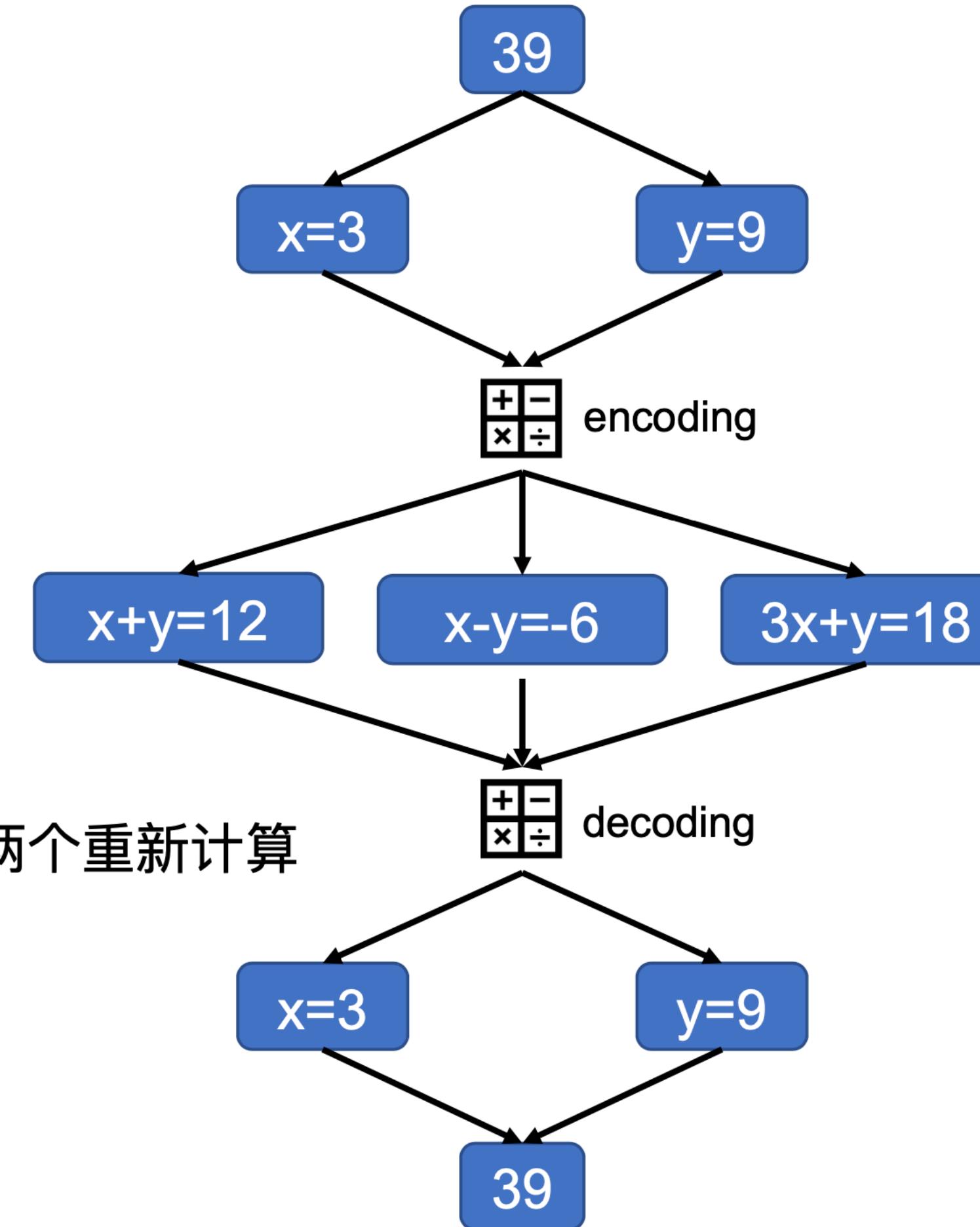
一种替代 3 份复制品的存储方法

- Drawback of replication
 - space overhead (e.g., 200%)
 - rarely accessed replicas
- Erasure coding
 - same or better level of fault-tolerance
 - much less overhead
 - used in RAID

Erasure Coding: Idea

- We can decode **39** using any two of the three equations
 - lose one equation does not matter, and we can recover it easily!

其中一个块丢了，还可以通过其他两个重新计算



考点 9.5 : HDFS Erasure Coding 的计算 (不太可能考)

知识点: 6,3 RS 的计算

Erasure Coding: (6,3)-Reed-Solomon

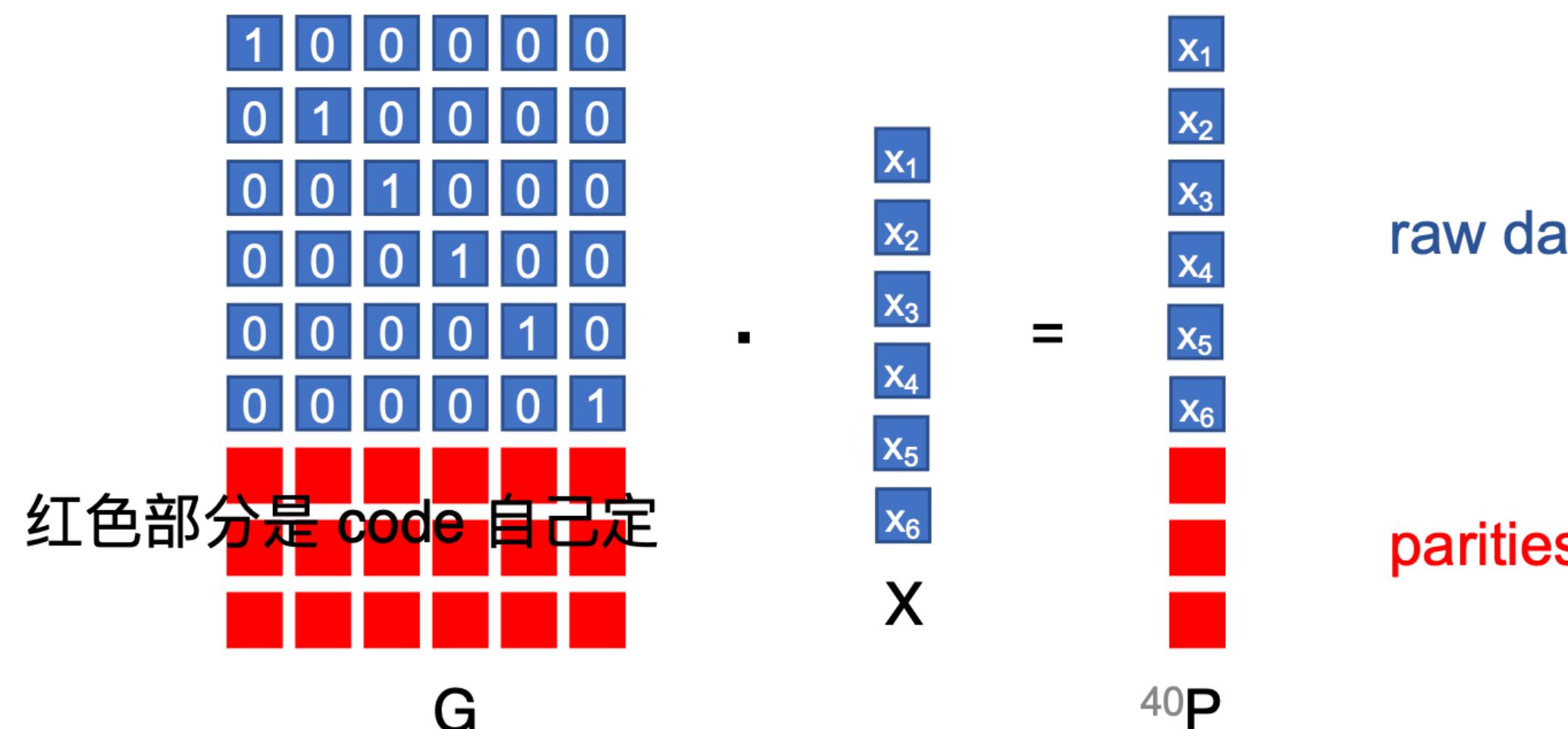
- Now consider $X = [x_1, \dots, x_6]^T$ and $G = \begin{bmatrix} I_6 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}$

- a matrix with any 6 rows from G has full rank.

- Then we can have $P = G \cdot X$

- We can recover X using any 6 rows from G and P

- $X = G'^{-1} \cdot P'$



红色部分是 code 自己定

G 是自己保存的，如果 P 中某些行丢了，我们可以通过对应该行的 G 组成的矩阵的逆矩阵，来还原 X

这里对于数学的要求：

- 矩阵的乘法
- 求一个矩阵的逆矩阵

考点 9.5 : HDFS Erasure Coding 的计算 (不太可能考)

G

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 1 & 2 & 3 \\ 2 & 1 & -1 & 0 \end{pmatrix}$$

X

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

P

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 17 \\ 1 \end{pmatrix}$$

$$G \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \dots & \dots & \dots & \dots \\ -1 & 1 & 2 & 3 \\ 2 & 1 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ \dots \\ 17 \\ 1 \end{pmatrix}.$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \cancel{0} & \cancel{0} & \cancel{1} & \cancel{0} \\ \cancel{0} & \cancel{0} & 0 & 1 \\ -1 & 1 & 2 & 3 \\ 2 & 1 & -1 & 0 \end{pmatrix}$$

G'

$$\longrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & 1 & -1 & 0 \end{pmatrix}$$

G'

求逆

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

G'^{-1}

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

G'^{-1}

$$\cdot \begin{pmatrix} 1 \\ 2 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

P'

$X \checkmark$

考点 9.5 Erasure Coding 的存储过程

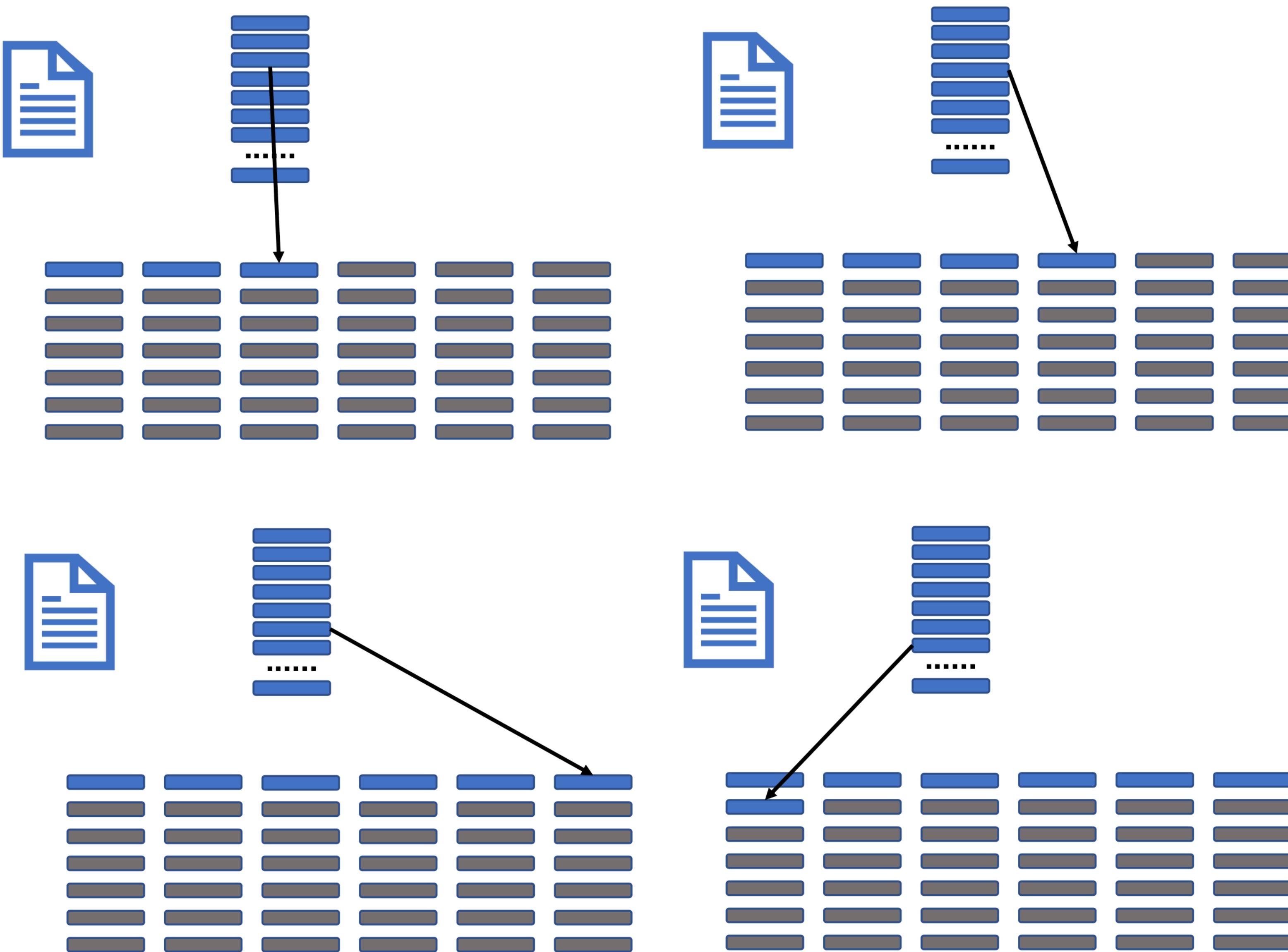
横着看是一个 Stripe
竖着看是一个 Block



每一个 Block 都被竖着分为 200 个 Cell (一个 Cell 64KB, 一个 Block 128 MB)

所有将要存储的文件，一律分成一个一个的 Cell，按照从左到右，从上到下的顺序，存入 Block，每存 6 个 cell，就多存 3 个 cell 作为 parties 以便将来计算恢复丢失的 Data

考点 9.5 Erasure Coding 的存储过程



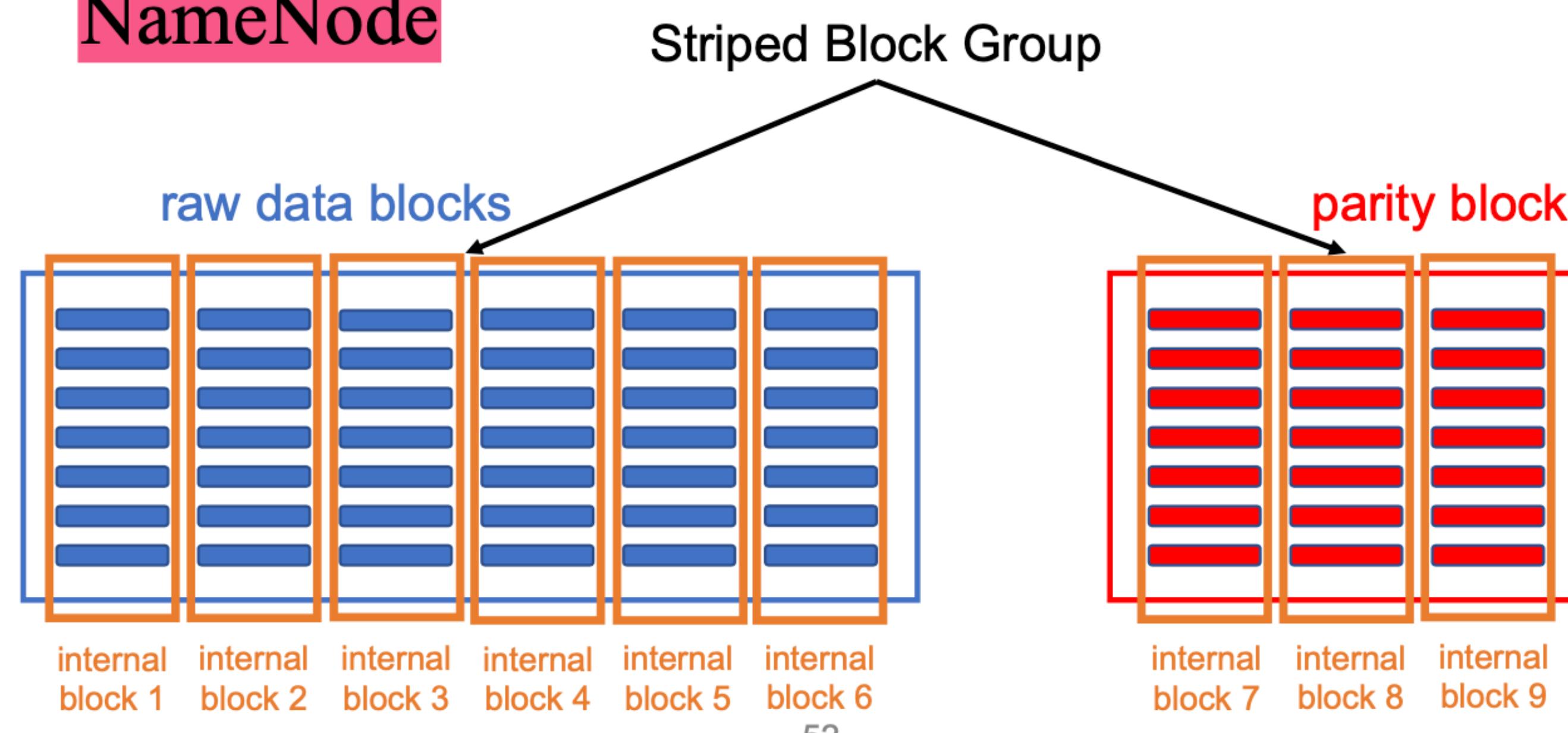
写文件的时候是横着写的，但是实际存储是数着存的

Striped Block Management

考点 9.5 Erasure Coding 的存储过程

- **Block group**

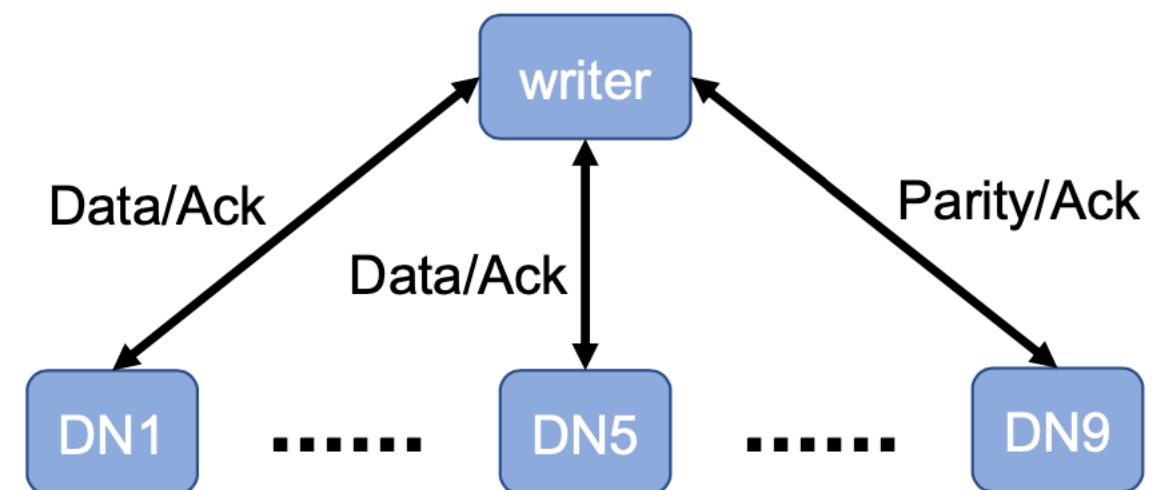
- Contains 6 raw data blocks and 3 parity blocks
- The blocks will be stored in different DataNodes
- Information of the block group will be stored in NameNode



考点 10 Erasure Coding 的读写流程

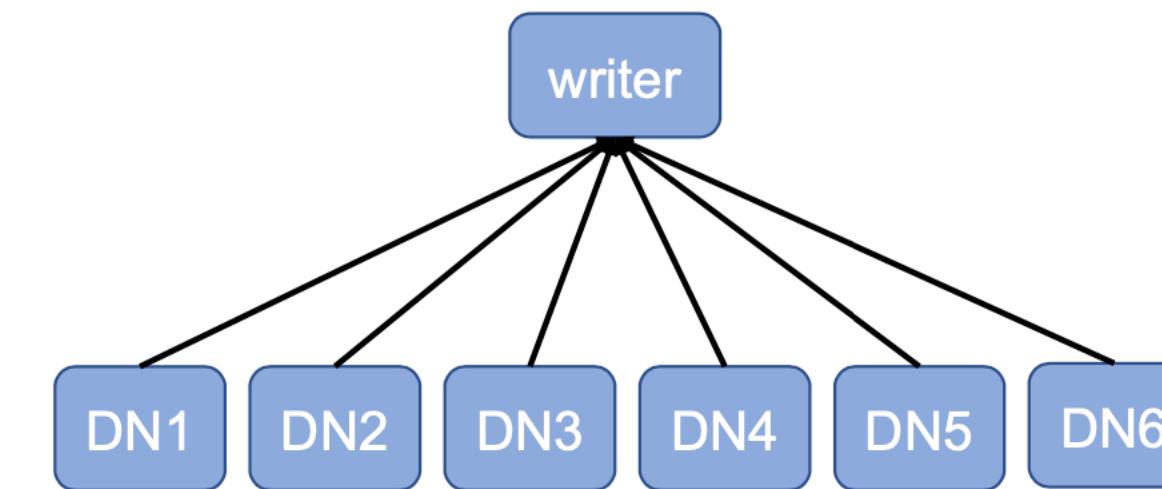
Parallel write

- Client writes a block group of 9 DataNodes simultaneously



Read

- Parallelly read from 6 DataNodes with data blocks

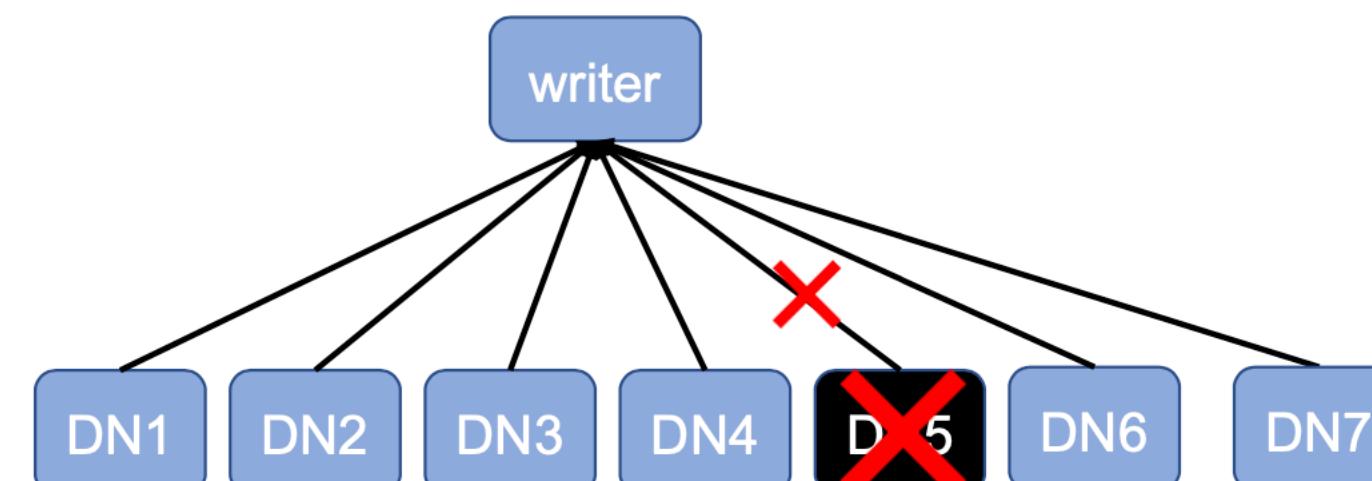


Handle Write Failure

- Client ignores the failed DataNode and continue writing
- Can tolerant up to 3 failures
- Missing blocks will be reconstructed later

Handle Read Failure

- Continue reading from any of the remaining DataNodes
- Reconstruct the failed nodes later



考点 10 Erasure Coding 与 Replication 的对比

Replication vs. Erasure Coding

- EC is better for large and rarely accessed files.
 - HDFS users and admins can turn on and off erasure coding for individual files or directories.

	Replication	Erasure Coding
storage overhead	High	Low
data durability	Yes	Yes (better)
data locality	Yes	No
write performance	Good	Poor
read performance	Good	Poor
recovery cost	Low	High

58

3-Replication vs. (6,3)-RS

	3-Replication	(6,3)-RS
Durability		
Maximum Toleration	2	3
Disk Space Consumption		
Data: n bytes	3n	1.5n
Number of Client-DataNode connections		
Write	1	9
Read	1	6

3-Replication vs. (6,3)-RS

- Number of blocks required to read the data

# of Blocks	3-Replication	(6,3)-RS
1	1	6
2	2	
3	3	
4	4	
5	5	
6	6	

押题：

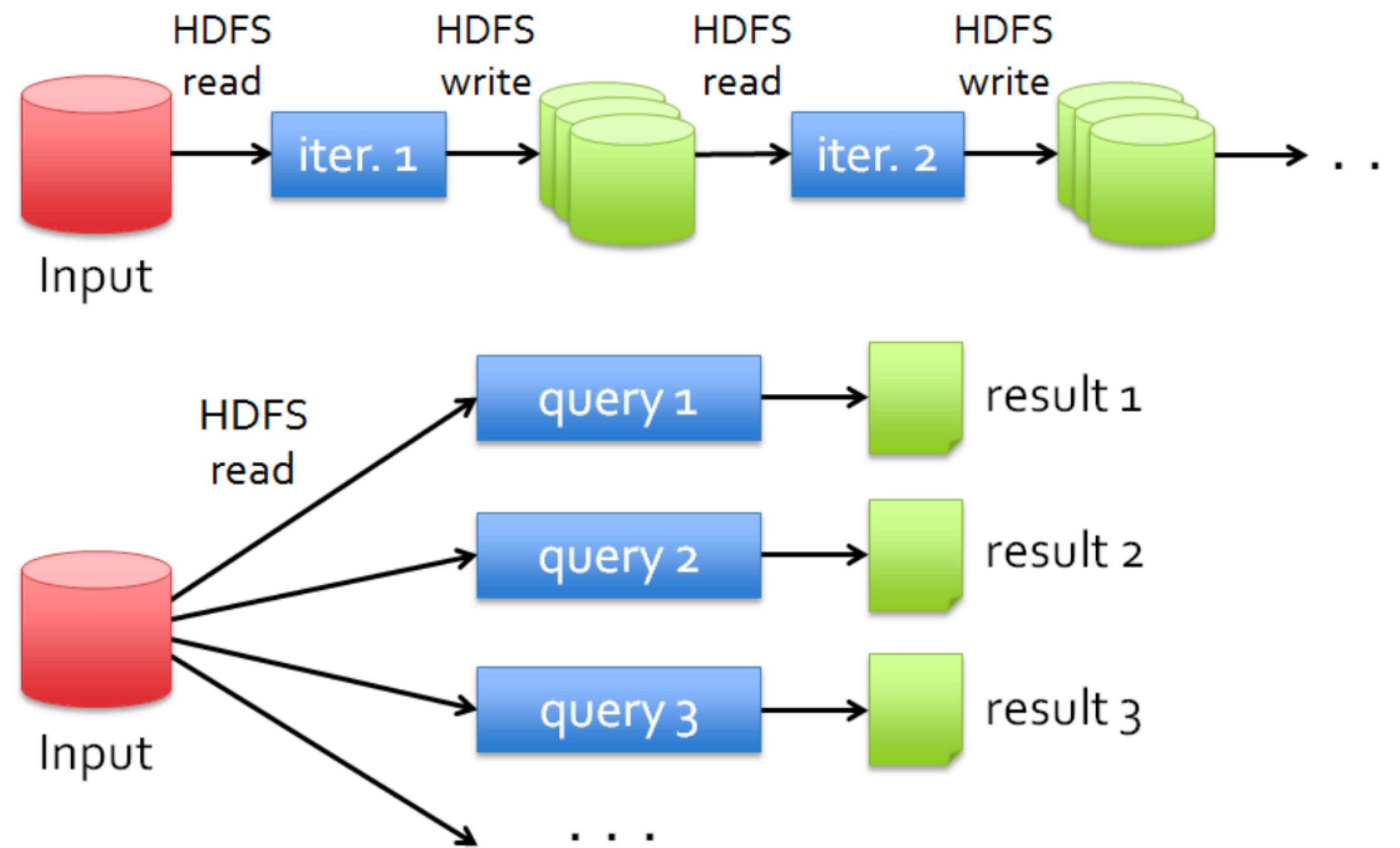
1. Describe the core component of Hadoop and the role they play ?
2. Name three techniques where follows the Master-Slave architecture We learned in this course and describe the mechanism ?
3. Explained the reason why hadoop introduce a 2NN describe how it corporate with 1NN?
4. Discuss the disadvantage of Large Block size and Small Block size in HDFS?
5. Describe the mechanism of uploading a file to HDFS as well as how 1NN choose the DataNode?
6. Describe the pros and cons about replications and EC in HDFS ?

- 
- MapReduce Spark

Limitations of MapReduce

- As a general programming model:
 - more suitable for one-pass computation on a large dataset
 - hard to compose and nest multiple operations
 - no means of expressing iterative operations
- As implemented in Hadoop
 - all datasets are read from disk, then stored back on to disk
 - all data is (usually) triple-replicated for reliability

Data Sharing in Hadoop MapReduce



- Slow due to replication, serialization, and disk IO
- Complex apps, streaming, and interactive queries all need one thing that MapReduce lacks:
 - Efficient primitives for data sharing

关于MapReduce 的缺点一句话：因为总要读写磁盘，所以慢
Spark 怎么解决的？把中间计算过程全存在内存里

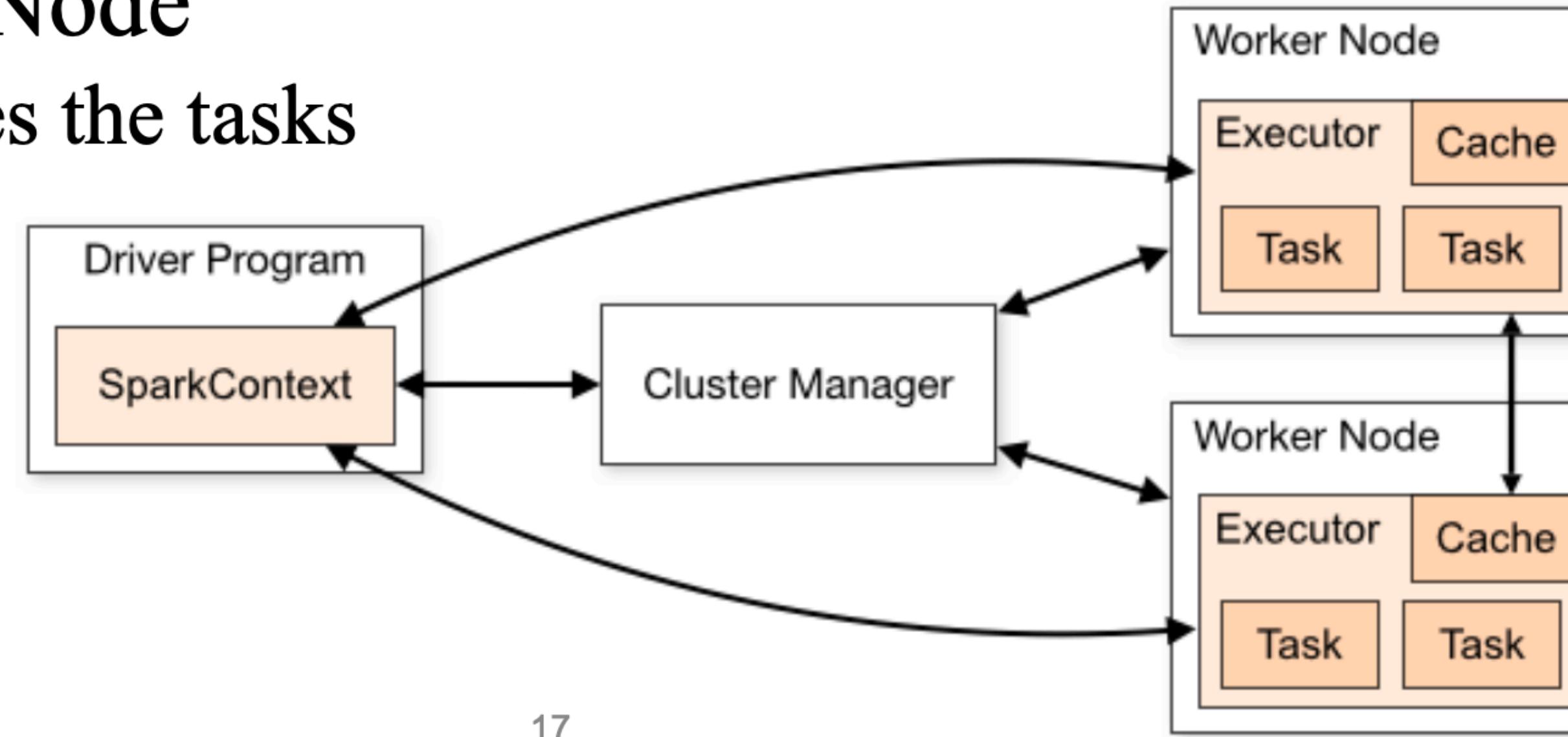
考点 2: Spark 的 Feature

1. In memory
2. Parallel
3. Fault-tolerant
4. Lazy evaluation

- Apache Spark is an open-source cluster computing framework for real-time processing.
- Spark provides an interface for programming entire clusters with
 - implicit data parallelism
 - fault-tolerance
- Built on top of Hadoop MapReduce
 - extends the MapReduce model to efficiently use more types of computations

Spark Architecture

- Master Node
 - takes care of the job execution within the cluster
- Cluster Manager
 - allocates resources across applications
- Worker Node
 - executes the tasks



考点 3: Spark 的架构 / 组成部分 / worker / driver / executor / Manager

1. Driver 程序的入口 (main) 里面声明了 SparkContext 是
2. Cluster Manager 是一台不管计算，只负责任务、资源调度的机器
3. Worker 是负责计算的机器
4. Executor 是真正实实在在的 JVM 线程

考点 4: Spark RDD / trans & action / lazy Evaluation

Resilient Distributed Dataset (RDD)

- RDD is where the data stays
- RDD is the fundamental data structure of Apache Spark
 - is a collection of elements
 - Dataset
 - can be operated on in parallel
 - Distributed
 - fault tolerant
 - Resilient

考点 4: Spark RDD / trans & action / lazy Evaluation

RDD Operation

RDD Transformations

Narrow

Map

Flatmap

Filter

Sample

Wide

ReduceByKey

GroupByKey

Fold

SortByKey

Join

RDD Actions

Collect

Take

Reduce

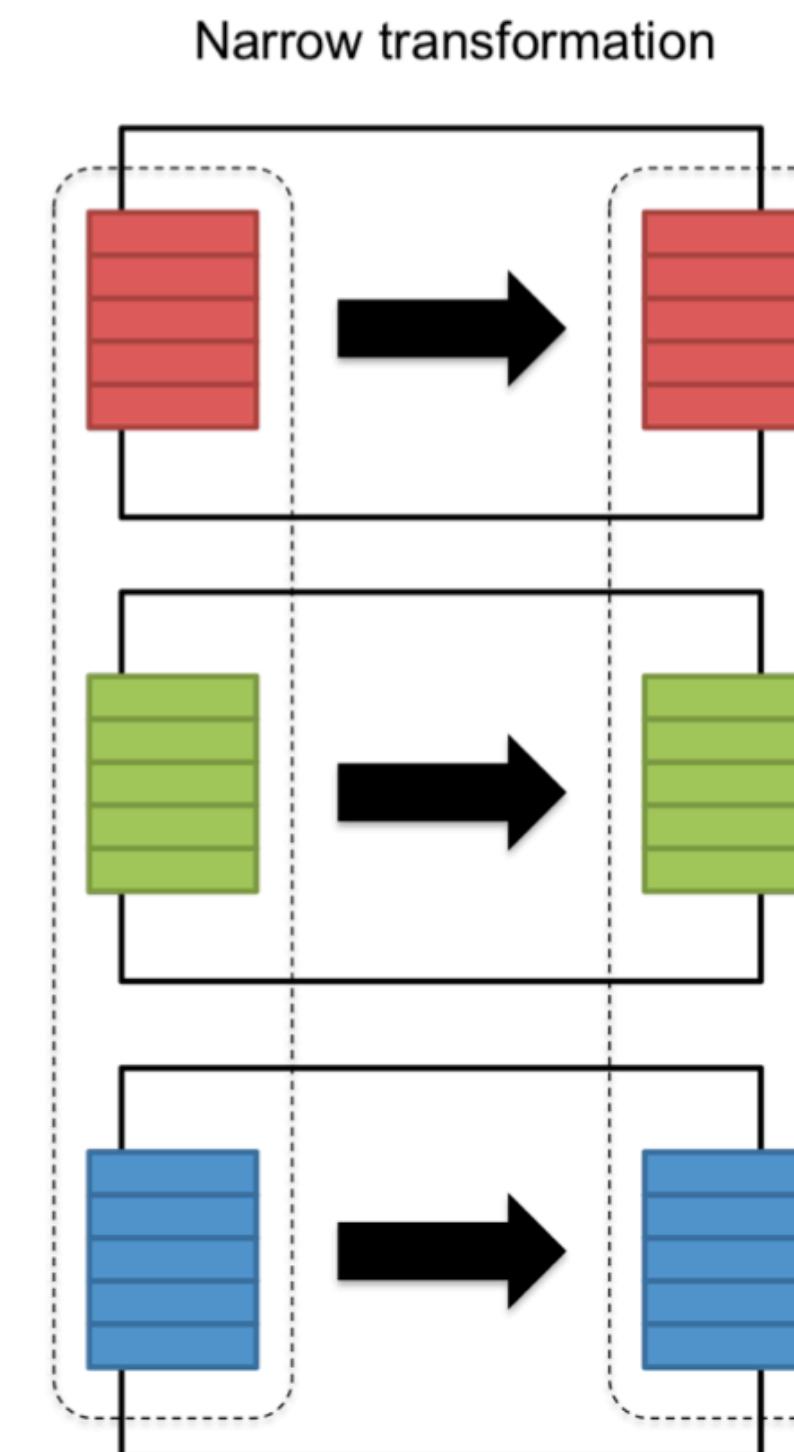
ForEach

SaveAsText

Narrow and Wide Transformations

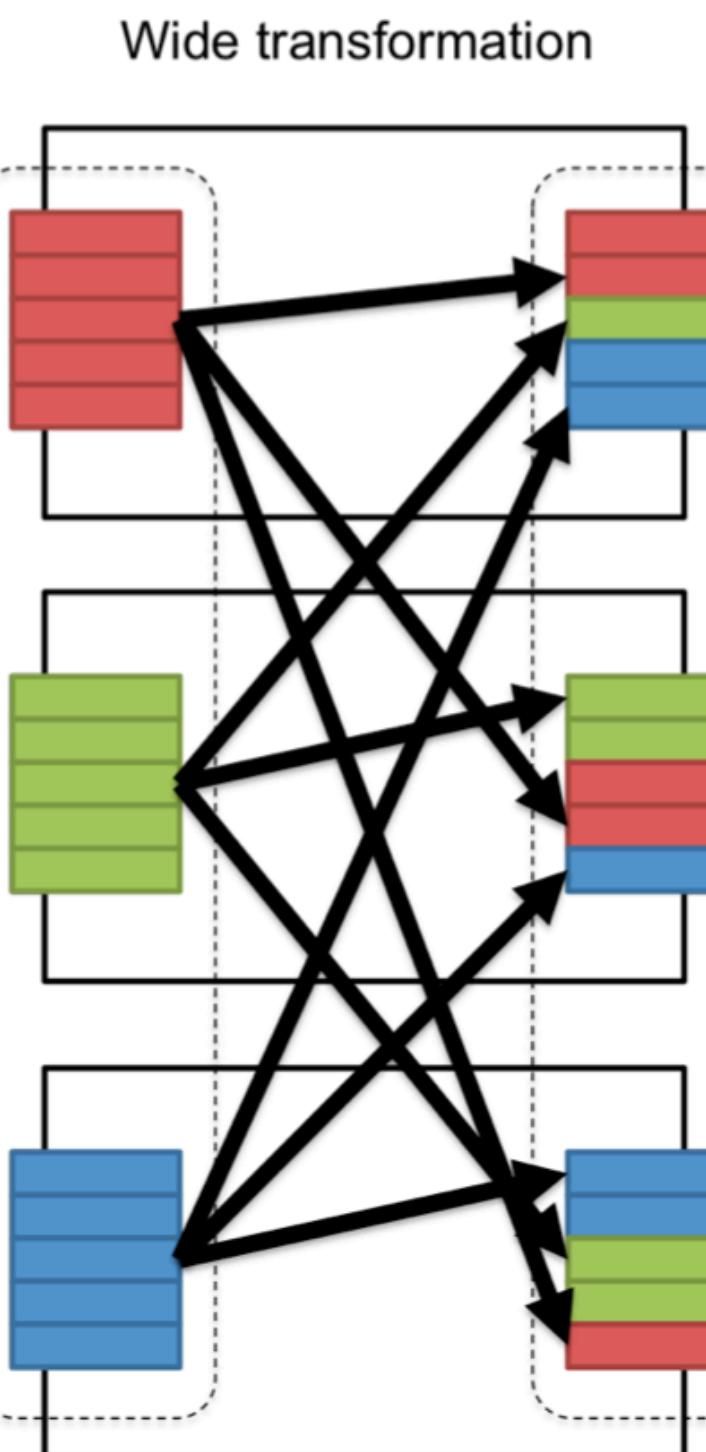
Narrow transformation
involves no data shuffling

- map
- flatMap
- filter
- sample



Wide transformation
involves data shuffling

- sortByKey
- reduceByKey
- groupByKey
- join



也可以从Data分区的角度来理解

RDD Traits

- **In-Memory**, i.e. data inside RDD is stored in memory as much (size) and long (time) as possible
- **Immutable or Read-Only**, i.e. it does not change once created and can only be transformed using transformations to new RDDs
- **Lazy evaluated**, i.e. the data inside RDD is not available or transformed until an action is executed that triggers the execution
- **Cacheable**, i.e. you can hold all the data in a persistent "storage" like memory (default and the most preferred) or disk (the least preferred due to access speed)
- **Parallel**, i.e. process data in parallel
- **Typed**, i.e. values in a RDD have types, e.g. `RDD[Long]` or `RDD[(Int, String)]`
- **Partitioned**, i.e. the data inside an RDD is partitioned (split into partitions) and then distributed across nodes in a cluster (one partition per JVM that may or may not correspond to a single node)

Sample Question 2

Question 2 Spark

- Given a large text file, your task is to find out the top- k most frequent co-occurring term pairs. The co-occurrence of (w, u) is defined as: u and w appear in the same line (this also means that (w, u) and (u, w) are treated equally). Your Spark program should generate a list of k key-value pairs ranked in descending order according to the frequencies, where the keys are the pair of terms and the values are the co-occurring frequencies (**Hint:** you need to define a function which takes an array of terms as input and generate all possible pairs).

Sample Question 2

```
In [1]: from pyspark import SparkConf, SparkContext
conf = SparkConf().setMaster("local").setAppName("hellow_9313")
sc = SparkContext(conf = conf).getOrCreate();
inputFile= ["hello world, there is an apple.",
           "If your program shows.",
           "If your program shows.",
           "this occurs because you may have multiple versions"]
textFile = sc.parallelize(inputFile)
words = textFile.map(lambda x: x.lower().split())
words.collect()
```

```
Out[1]: [['hello', 'world', 'there', 'is', 'an', 'apple.'],
          ['if', 'your', 'program', 'shows.'],
          ['if', 'your', 'program', 'shows.'],
          ['this', 'occurs', 'because', 'you', 'may', 'have', 'multiple', 'versions']]
```

```
In [2]: def getAllPairs(line):
click to scroll output; double click to hide
    sortedLine = sorted(line)
    for i in range(len(sortedLine)):
        for j in range(i + 1, len(sortedLine)):
            res += [((sortedLine[i], sortedLine[j]), 1)]
    return res
getAllPairs(['if', 'your', 'program', 'shows.'])
```

```
Out[2]: [(['if', 'program'), 1),
          ('if', 'shows.'), 1),
          ('if', 'your'), 1),
          ('program', 'shows.'), 1),
          ('program', 'your'), 1),
          ('shows.', 'your'), 1)]
```

Sample Question 2

```
In [3]: pairs = words.flatMap(getAllPairs).reduceByKey(lambda x, y: x + y)
topk = pairs.map(lambda x: (x[1],x[0])).sortByKey(False).map(lambda x: (x[1], x[0]))
topk.collect()
```

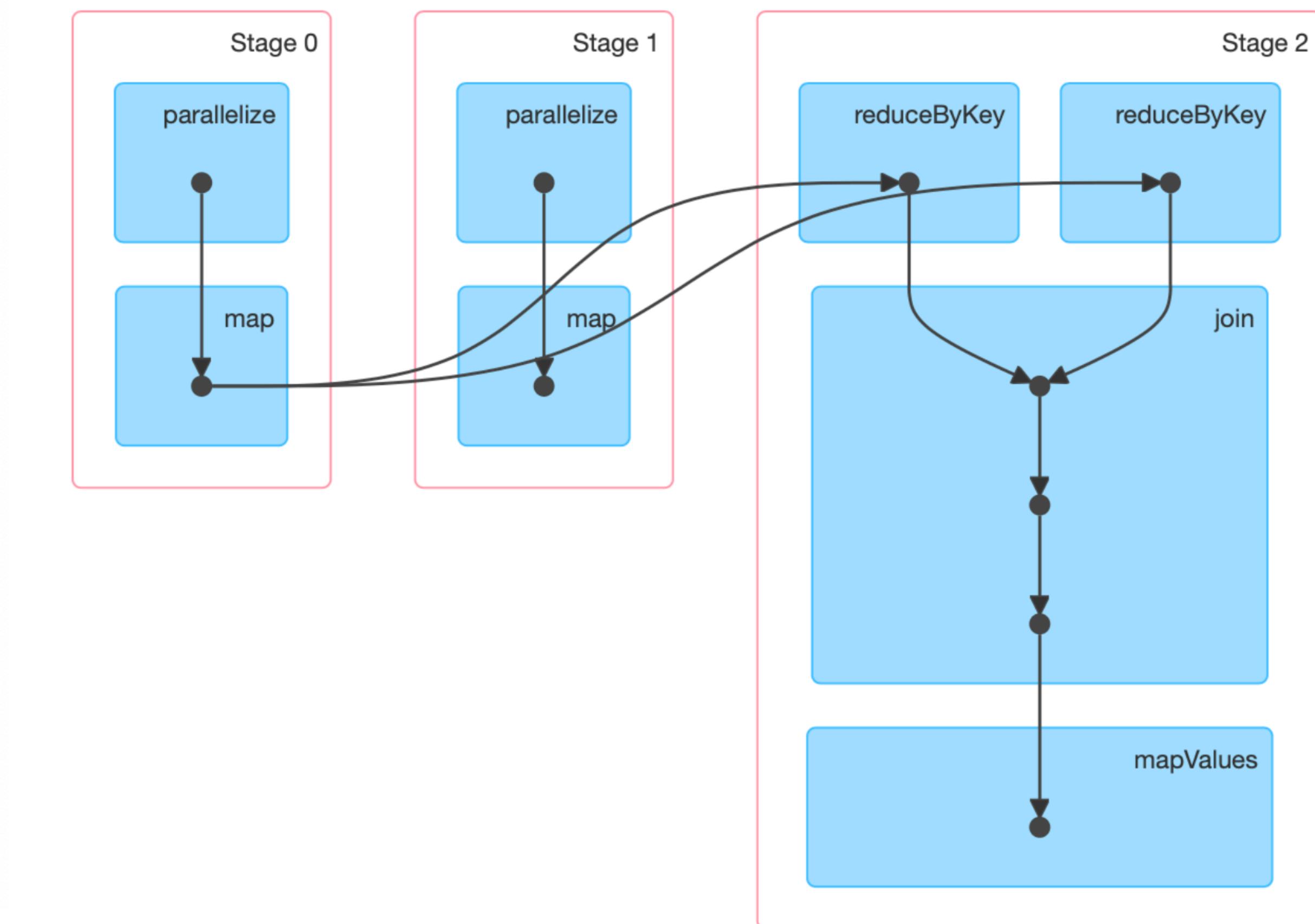
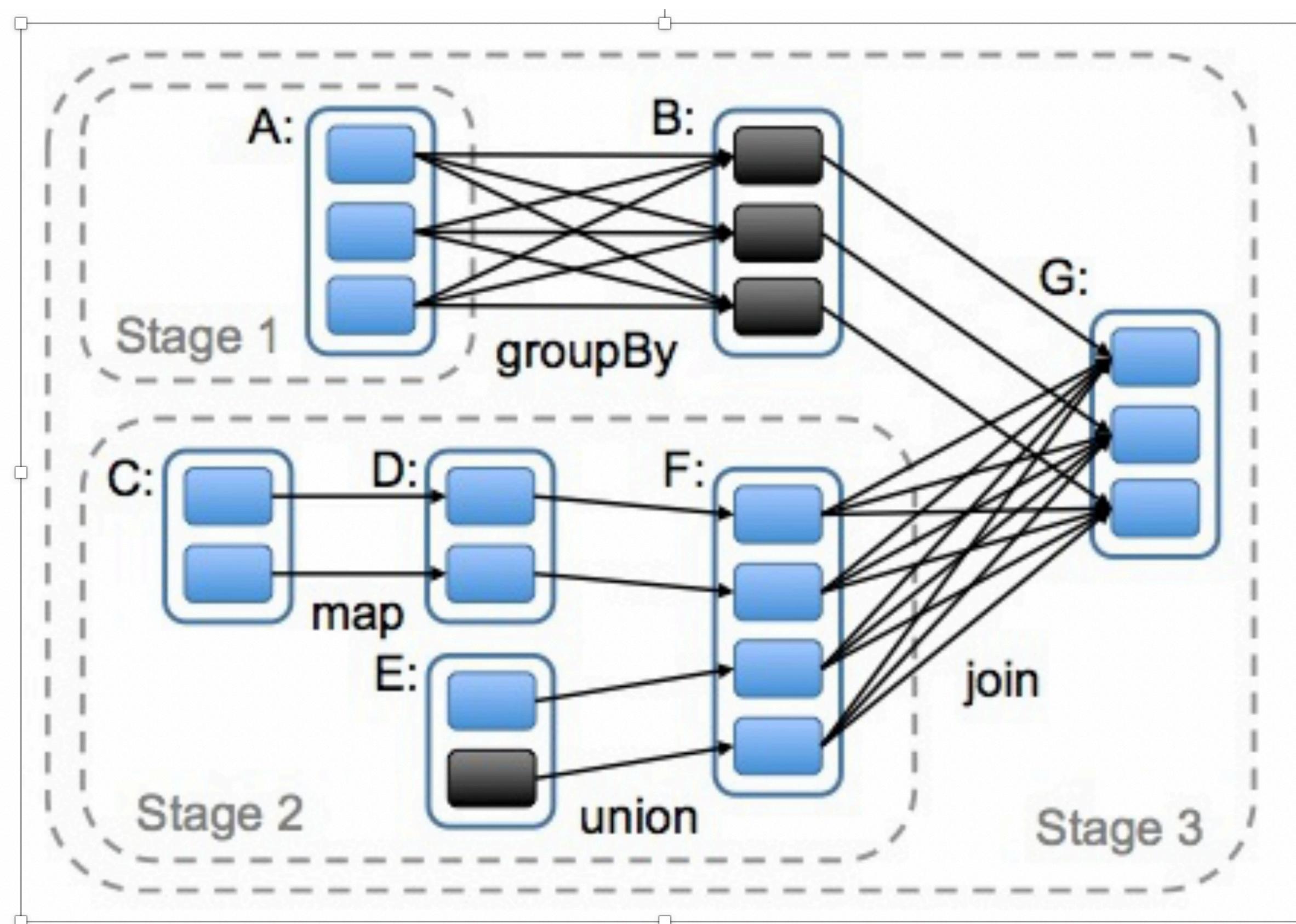
```
Out[3]: [('if', 'program'), 2),
          ('if', 'shows.'), 2),
          ('if', 'your'), 2),
          ('program', 'shows.'), 2),
          ('program', 'your'), 2),
          ('shows.', 'your'), 2),
          ('an', 'apple.'), 1),
          ('an', 'hello'), 1),
          ('an', 'is'), 1),
          ('an', 'there'), 1),
          ('an', 'world,'), 1),
          ('apple.', 'hello'), 1),
          ('apple.', 'is'), 1),
          ('apple.', 'there'), 1),
          ('apple.', 'world,'), 1),
          ('hello', 'is'), 1),
          ('hello', 'there'), 1),
          ('hello', 'world,'), 1),
          ('is', 'there'), 1),
          ('is', 'world,'), 1),
          ('there'. 'world.'), 1).
```

考点 5: Spark RDD Lineage / DAG / 阶段划分

DAG(Directed Acyclic Graph)叫做有向无环图，原始的RDD通过一系列的转换就形成了DAG，根据RDD之间的依赖关系的不同将DAG划分成不同的Stage，对于窄依赖，partition的转换处理在Stage中完成计算。对于宽依赖，由于有Shuffle的存在，只能在parent RDD处理完成后，才能开始接下来的计算，因此**宽依赖是划分Stage的依据。**

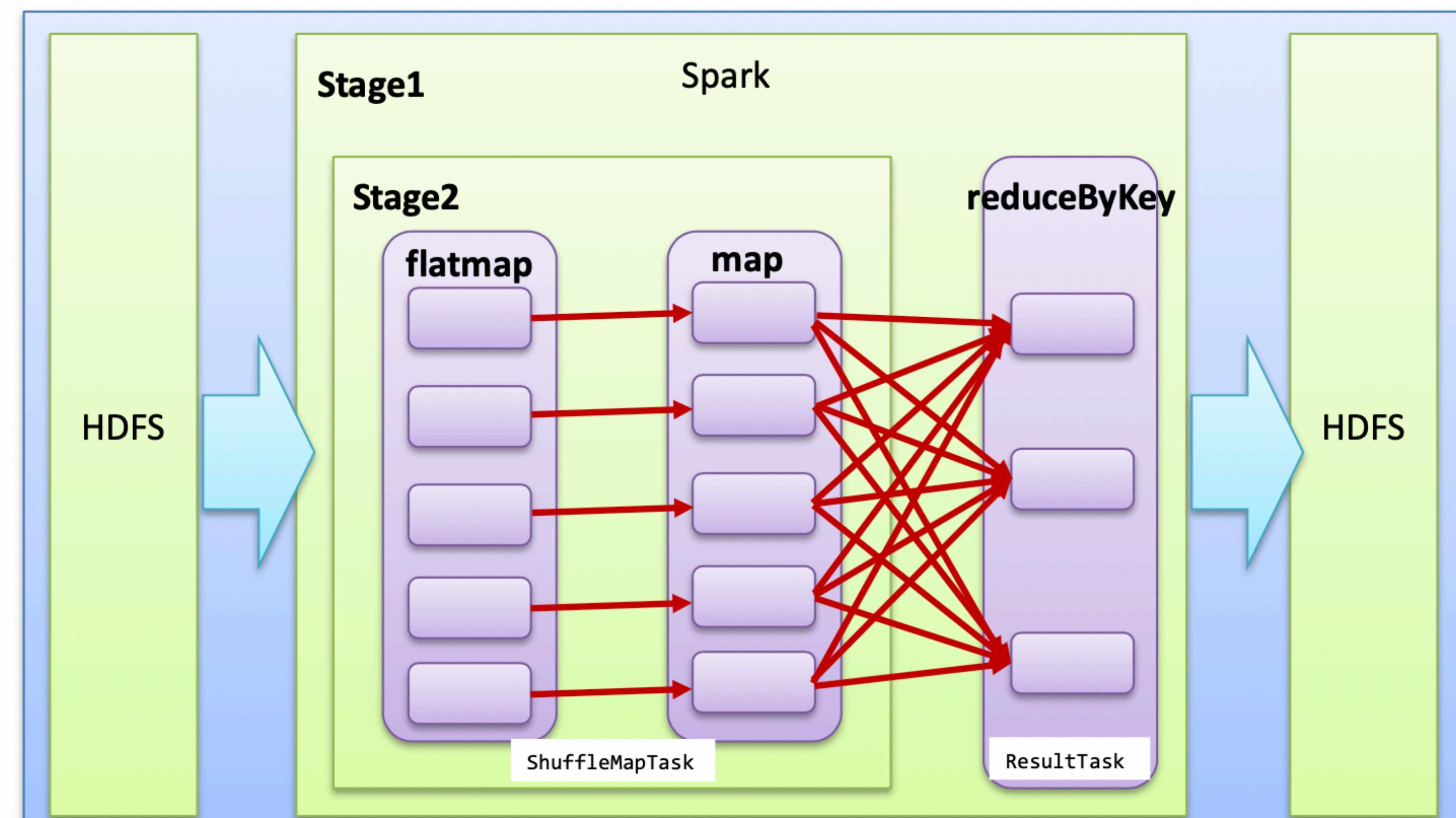
判断阶段，要从后往前推

考点 5: Spark RDD Lineage / DAG 阶段划分



考点 5: Spark RDD Lineage / DAG 阶段划分

- Job: 一个 Action 算子就会生成一个 Job
- Stage: 根据 RDD 之间的依赖关系(DAG)的不同将 Job 划分成不同的 Stage, 遇到一个**宽依赖**则划分一个 Stage。 $1 + \text{num(shuffle)}$
- Task: 一个 Stage 里面会有很多个 Task (一个分区就是一个 Task), 同一个阶段的 task 会并行计算
- 一个 Job 会有多个 Stage
- 一个 Stage 里面会有很多个 Task



考点 5.5 : Spark RDD Lineage DAG 的区别

Lineage vs. DAG in Spark

- They are both DAG (data structure)
- Different end nodes
- Different roles in Spark

1. Lineage 描述的是某一个 RDD 它的所有依赖关系
2. DAG 描述的是一个 Action 算子前所有 RDD 的依赖/阶段关系

考点 6: MapReduce Map/Reduce 过程 shuffle 过程

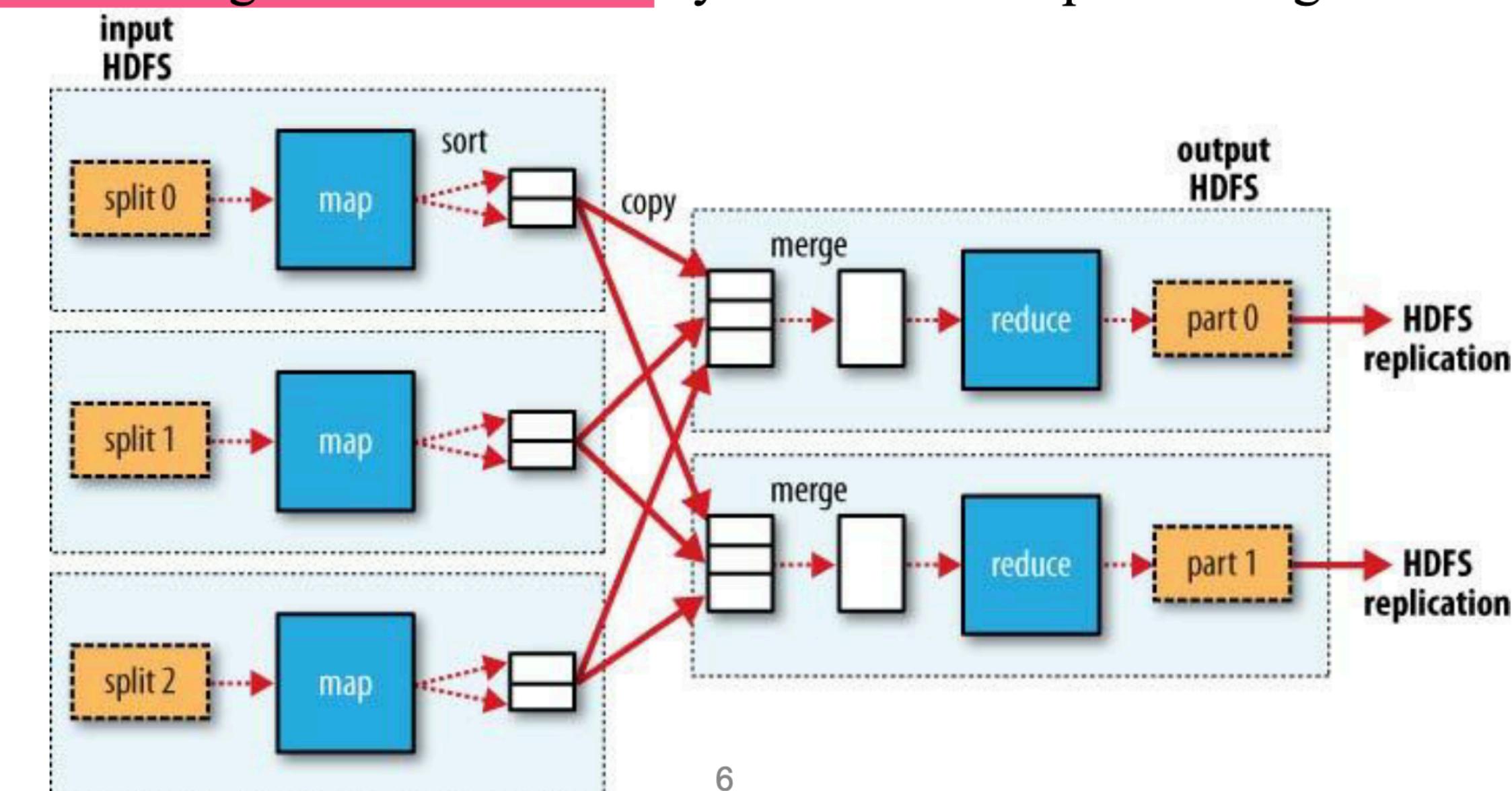
首先知道 MR job 工作的时候分为 MapTask 和 ReduceTask 两个阶段

MapReduce in Hadoop

- Map tasks write their output to local disk (not to HDFS)
 - Map output is intermediate output
 - Once the job is complete the map output can be thrown away
 - Storing it in HDFS with replication, would be overkill
 - If the node of map task fails, Hadoop will automatically rerun the map task on another node
- Reduce tasks don't have the advantage of data locality
 - Input to a single reduce task is normally the output from all mappers
 - Output of the reduce is stored in HDFS for reliability
 - The number of reduce tasks is not governed by the size of the input, but is specified independently

More Detailed MapReduce Dataflow

- When there are multiple reducers, the map tasks partition their output:
 - One partition for each reduce task
 - The records for every key are all in a single partition
 - Partitioning can be controlled by a user-defined partitioning function



考点 6: MapReduce Map/Reduce 过程 shuffle 过程

MR 中 Shuffle 的意义在于： 把相同的 Key 聚合在一起，并对 Key 排序

Shuffle in Hadoop (handled by framework)

- Happens between each Map and Reduce phase
- Use **Shuffle and Sort mechanism**
 - Results of each Mapper are sorted by the key
 - Starts as soon as each mapper finishes
- Use **combiner** to reduce the amount of data shuffled
 - Combiner combines key-value pairs with the same key in each par
 - This is not handled by framework!

考点 7: MapReduce Efficiency 3 点

The Efficiency of MapReduce in Spark

- Number of transformations
 - Each transformation involves a linearly scan of the dataset (RDD)
- Size of transformations
 - Smaller input size => less cost on linearly scan
- Shuffles
 - data transferring between partitions is costly
 - especially in a cluster!
 - Disk I/O
 - Data serialization and deserialization
 - Network I/O

押题：

1. Describe the case where MR is not suitable for using compared with Spark and what is the main difference between these two?
2. Explain what is lazy evaluation in Spark and why its designed by this way?
3. Name three wide transformation and narrow transformation in Spark, explain their difference ?
4. Consider this scenario Draw the Linage Graph of RDD1 :

```
data = ["hello", "world", "hello", "world", "word", "count", "hello"]
rdd1 = sc.parallelize(data)
rdd2 = rdd1.map(lambda word: (word, 1))
rdd3 = rdd2.reduceByKey(lambda a, b: a + b)
rdd4 = rdd3.map(lambda word: (word, 1))
```

5. Exaplain what happened in MR shuffle and why its expensive to do?
6. Exaplain what relations / difference between RDD and DataFrame?

极大可能出计算题，重在理解算法和流程



- **High Dimensional Similarity Search**

考点 1: LSH 概念 / 为什么要引入 LSH ?

Locality Sensitive Hashing

- Index: make the hash functions error tolerant
 - Similar data \Rightarrow same hash key (with high probability)
 - Dissimilar data \Rightarrow different hash keys (with high probability)
- Retrieval:
 - Compute the hash key for the query
 - Obtain all the data has the same key with query (i.e., candidates)
 - Find the nearest one to the query 对 Candidates 就要使用传统的距离计算公式
- Cost:
 - Space: $O(n)$
 - Time: $O(1) + O(|cand|)$

考点 1: LSH 的两个重要部分

1. Hash Function 数学证明不需要掌握
2. $\Pr[h(q) = h(o)]$ 与不同距离公式的关系
3. And OR composition

考点 2: LSH functions- min hash

MinHash - LSH Function for Jaccard Similarity

- Each data object is a set

- $Jaccard(S_1, S_2) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|}$

- Randomly generate a global order for all the elements in $C = \bigcup_1^n S_i$ First Member
- Let $h(S)$ be the minimal member of S with respect to the global order

- For example, $S = \{b, c, e, h, i\}$, we use inversed alphabet order, then re-ordered $S = \{i, h, e, c, b\}$, hence $h(S) = i$.

考点 2: LSH functions- min hash 的 pr 计算

MinHash

- Now we compute $\Pr[h(S_1) = h(S_2)]$
- Every element $e \in S_1 \cup S_2$ has equal chance to be the first element among $S_1 \cup S_2$ after re-ordering
- $e \in S_1 \cap S_2$ if and only if $h(S_1) = h(S_2)$
- $e \notin S_1 \cap S_2$ if and only if $h(S_1) \neq h(S_2)$
- $\Pr[h(S_1) = h(S_2)] = \frac{|\{e_i | h_i(S_1) = h_i(S_2)\}|}{|\{e_i\}|} = \frac{|S_i \cap S_j|}{|S_i \cup S_j|} = Jaccard(S_1, S_2)$

Question 3 Finding Similar Items @696

Suppose we wish to find similar sets, and we apply locality-sensitive hashing with $k=5$ and $l=2$.

If two sets had Jaccard similarity 0.6, what is the probability that they will be identified in the locality-sensitive hashing as candidates (i.e. they hash at least once to the same super-hash)? You may assume that there are no coincidences, where two unequal values hash to the same hash value.

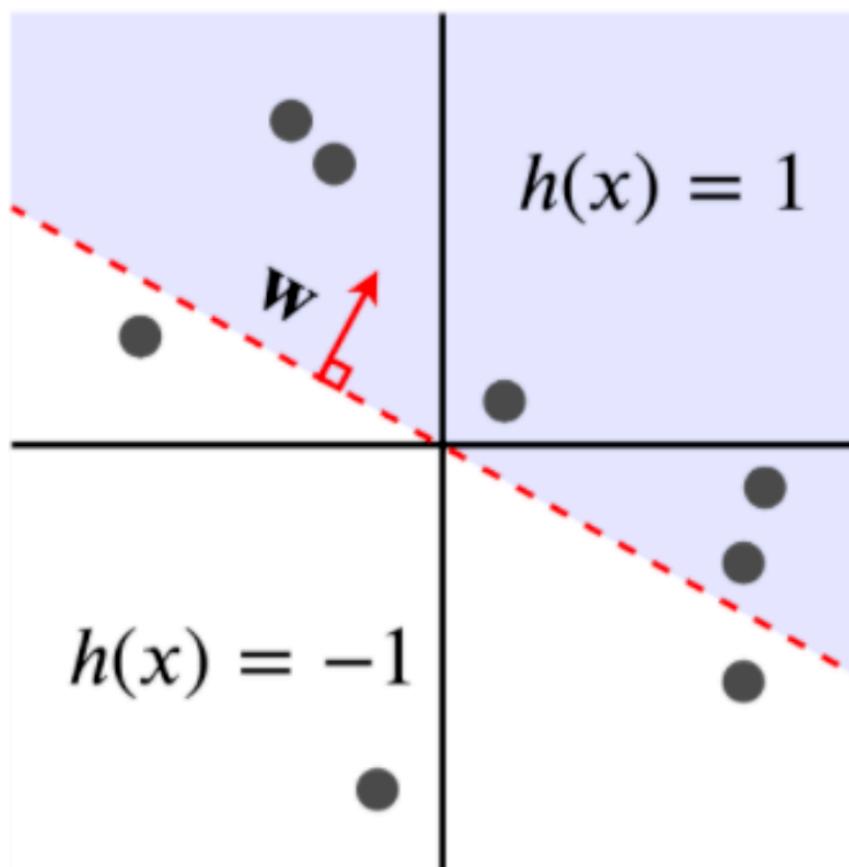
- The probability of o is a nearest neighbor candidate of q is
 - $1 - (1 - p_{q,o}^k)^l$

考点 2: LSH functions for different distance

Ass1 考过了不大可能考了

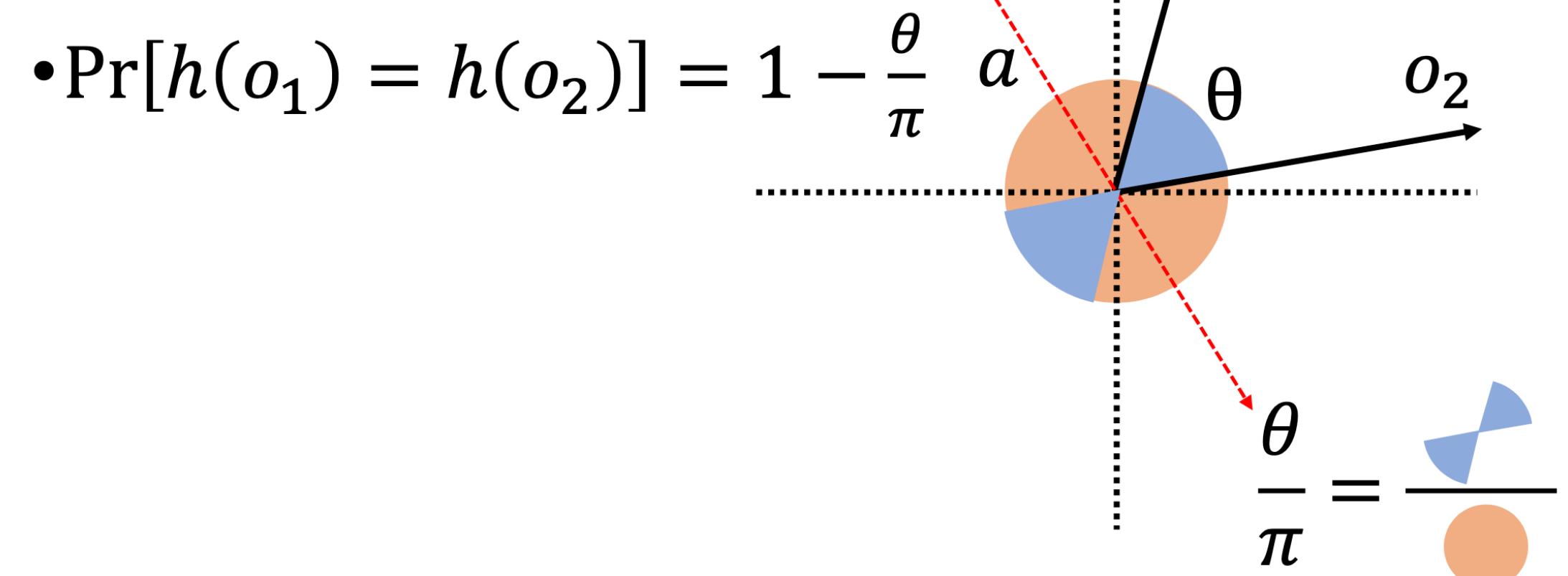
SimHash – LSH Function for Angular Distance

- Each data object is a d dimensional vector
 - $\theta(x, y)$ is the angle between x and y
- Randomly generate a normal vector a , where $a_i \sim N(0,1)$
- Let $h(x; a) = \text{sgn}(a^T x)$
 - $\text{sgn}(o) = \begin{cases} 1; & \text{if } o \geq 0 \\ -1; & \text{if } o < 0 \end{cases}$
 - x lies on which side of a 's corresponding hyperplane



SimHash

- Now we compute $\Pr[h(o_1) = h(o_2)]$
- $h(o_1) \neq h(o_2)$ iff o_1 and o_2 are on different sides of the hyperplane with a as its normal vector



考点 2: LSH functions for different distance

很有可能在这里出题

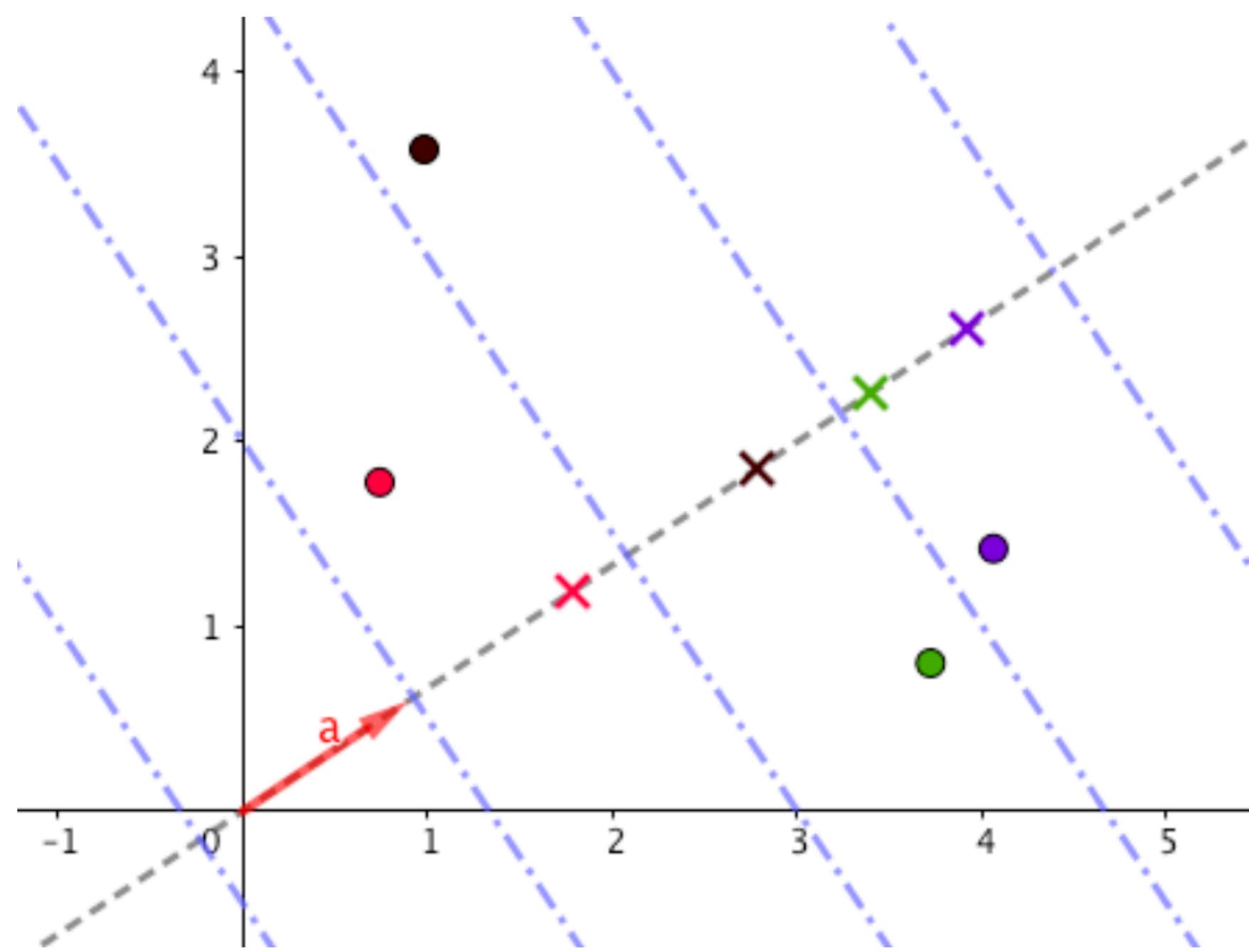
p-stable LSH - LSH function for Euclidean distance

- Each data object is a d dimensional vector
 - $dist(x, y) = \sqrt{\sum_1^d (x_i - y_i)^2}$
- Randomly generate a normal vector a , where $a_i \sim N(0, 1)$
 - Normal distribution is 2-stable, i.e., if $a_i \sim N(0, 1)$, then $\sum_1^d a_i \cdot x_i \sim N(0, \|x\|_2^2)$
- Let $h(x; a, b) = \left\lfloor \frac{a^T x + b}{w} \right\rfloor$, where $b \sim U(0, 1)$ and w is user specified parameter
 - $\Pr[h(o_1; a, b) = h(o_2; a, b)] = \int_0^w \frac{1}{\|o_1, o_2\|} f_p\left(\frac{t}{\|o_1, o_2\|}\right) \left(1 - \frac{t}{w}\right) dt$
 - $f_p(\cdot)$ is the pdf of the absolute value of normal variable

考点 2: LSH functions for different distance

p-stable LSH

- Intuition of p-stable LSH
 - Similar points have higher chance to be hashed together



Consider these data:

$$\begin{aligned} \mathbf{a} &= (2, -1) \\ b &= 0.5 \\ w &= 4 \end{aligned}$$

• Let $h(x; a, b) = \left\lfloor \frac{\mathbf{a}^T \mathbf{x} + b}{w} \right\rfloor$.

	Data	Hash	IfCandidate
x1	(1, 2)	0	Yes
x2	(-1, 3)	-2	No
x3	(2, 2)	0	Yes
Q	(1, 1)	0	

考点 2: AND-OR composition

- Recall for a single hash function, we have
 - $\Pr[h(o_1) = h(o_2)] = p(\text{dist}(o_1, o_2))$, denoted as p_{o_1, o_2}
- Now we consider two scenarios:
 - Combine k hashes together, using AND operation
 - One must match all the hashes
 - $\Pr[H_{AND}(o_1) = H_{AND}(o_2)] = p_{o_1, o_2}^k$
 - Combine l hashes together, using OR operation
 - One need to match at least one of the hashes
 - $\Pr[H_{OR}(o_1) = H_{OR}(o_2)] = 1 - (1 - p_{o_1, o_2})^l$
 - Not match only when all the hashes don't match

And : 生成 k 个 HashFunction 从而每个 data 对应一个 k 位的 hashcode

Or: 重复 l 次得到 l 个 hashtable, l 个 table 中只要有一个跟 Query 的 Hash 相等即可被选为 candidate

K 是每一个 Data 对应的 HashCode 的 位数

相当于每次 hash 需要用的到 vector a 的个数

L 是 hash 的次数， hash L 次相当于生成了 L 个 HashTable 和 QueryCode

对于每一个 Data 只要这 L 个 HashCode 其中有一个与 QueryCode 匹配

(每一位都匹配)

则这个 Data 就可以成为 Candidate

- The probability of o is a nearest neighbor candidate of q is

- $1 - (1 - p_{q,o}^k)^l$ 简单的二项分布问题

例子：

- Let $h(x; a, b) = \left\lfloor \frac{a^T x + b}{w} \right\rfloor$

$$\begin{aligned} K &= 1 \\ L &= 3 \end{aligned}$$

$$\begin{aligned} a &= (2, -1) \\ b &= 0.5 \\ w &= 4 \end{aligned}$$

table 1

$$\begin{aligned} a &= (1, 0) \\ b &= 0.1 \\ w &= 2 \end{aligned}$$

table 2

$$\begin{aligned} a &= (1, 1) \\ b &= 0.6 \\ w &= 100 \end{aligned}$$

table 3

	Data	Hash	IfCandidate
X1	(1, 2)	0	Yes
X2	(-1, 3)	-2	No
X3	(2, 2)	0	Yes
Q	(1, 1)	0	

	Data	Hash	IfCandidate
X1			
X2	(-1, 3)	-1	No
X3			
Q	(1, 1)	0	

	Data	Hash	IfCandidate
X1			
X2	(-1, 3)	0	Yes
X3			
Q	(1, 1)	0	

The Drawback of LSH

- Concatenate k hashes is too “strong”
 - $h_{i,j}(o_1) \neq h_{i,j}(o_2) \Rightarrow H_i(q) \neq H_i(o)$ for any j
- Not adaptive to the distribution of the distances
 - What if not enough candidates?
 - Need to tune w (or build indexes different w’s) to handle different cases

考点 3: False Positives and False Negatives

False Positives and False Negatives

- False Positive:
 - returned data with $\text{dist}(o, q) > r_2$
- False Negative
 - not returned data with $\text{dist}(o, q) < r_1$
- They can be controlled by carefully chosen k and l
- It's a trade-off between space/time and accuracy

False positive : 某个 Data 本来不应该成为 Candidate, 却最终被选为 Candidate 了

False negative : 某个 Data 本来应该成为 Candidate, 却最终没被选为 Candidate

The Framework of NNS using C2LSH

- Pre-processing
 - Generate LSH functions
 - Random normal vectors and random uniform values
- Index
 - Compute and store $h_i(o)$ for each data object o ,
 $i \in \{1, \dots, m\}$
- Query
 - Compute $h_i(q)$ for query q , $i \in \{1, \dots, m\}$
 - Take those o that shares at least αm hashes with q as candidates
 - Relax the collision condition (e.g., virtual rehashing) and repeat the above step, until we got enough candidates

考点4:C2 LSH

Pseudo code of Candidate Generation in C2LSH

```
candGen(data_hashes, query_hashes,  $\alpha m$ ,  $\beta n$ ):
```

```
    offset  $\leftarrow 0$ 
```

```
    cand  $\leftarrow \emptyset$ 
```

```
    while true:
```

```
        for each (id, hashes) in data_hashes:
```

```
            if count(hashes, query_hashes, offset)  $\geq \alpha m$ : If match
```

```
                cand  $\leftarrow$  cand  $\cup \{id\}$ 
```

```
            if  $|cand| < \beta n$ :
```

```
                offset  $\leftarrow$  offset + 1
```

```
            else:
```

```
                break
```

```
    return cand
```

考点4:C2 LSH

```
count(hashes_1, hashes_2, offset):
    counter ← 0
    for each  $hash_1, hash_2$  in hashes_1, hashes_2:
        if  $|hash_1 - hash_2| \leq offset$ :
            counter ← counter + 1
    return counter
```

假设 $\alpha m = 8$

考点4:C2 LSH

- At first consider $h(o) = h(q)$

Collision

HashCode(query)

q	1	1	1	1	1	1	1	1	1	1
o_1	1	0	2	-1	1	2	4	-3	0	-1

- Consider $h(o) = h(q) \pm 1$ if not enough candidates

q	1	1	1	1	1	1	1	1	1	1
o_1	1	0	2	-1	1	2	4	-3	0	-1

- Then $h(o) = h(q) \pm 2$ and so on...

q	1	1	1	1	1	1	1	1	1	1
o_1	1	0	2	-1	1	2	4	-3	0	-1

考点5 : PQ FrameWork / Distance Estimation

Framework of PQ

- Pre-processing:
 - Step 1: partition data vectors
 - Step 2: generate codebooks (e.g., k-means)
 - Step 3: encode data

例子：

- Step 1: partition data vectors



$d = \text{dimension 数} = 8$

O1	2	4	6	5	-2	6	4	1
O2	1	2	1	4	9	-1	2	0
O3
O4
...

$n = \text{Data 总个数}$

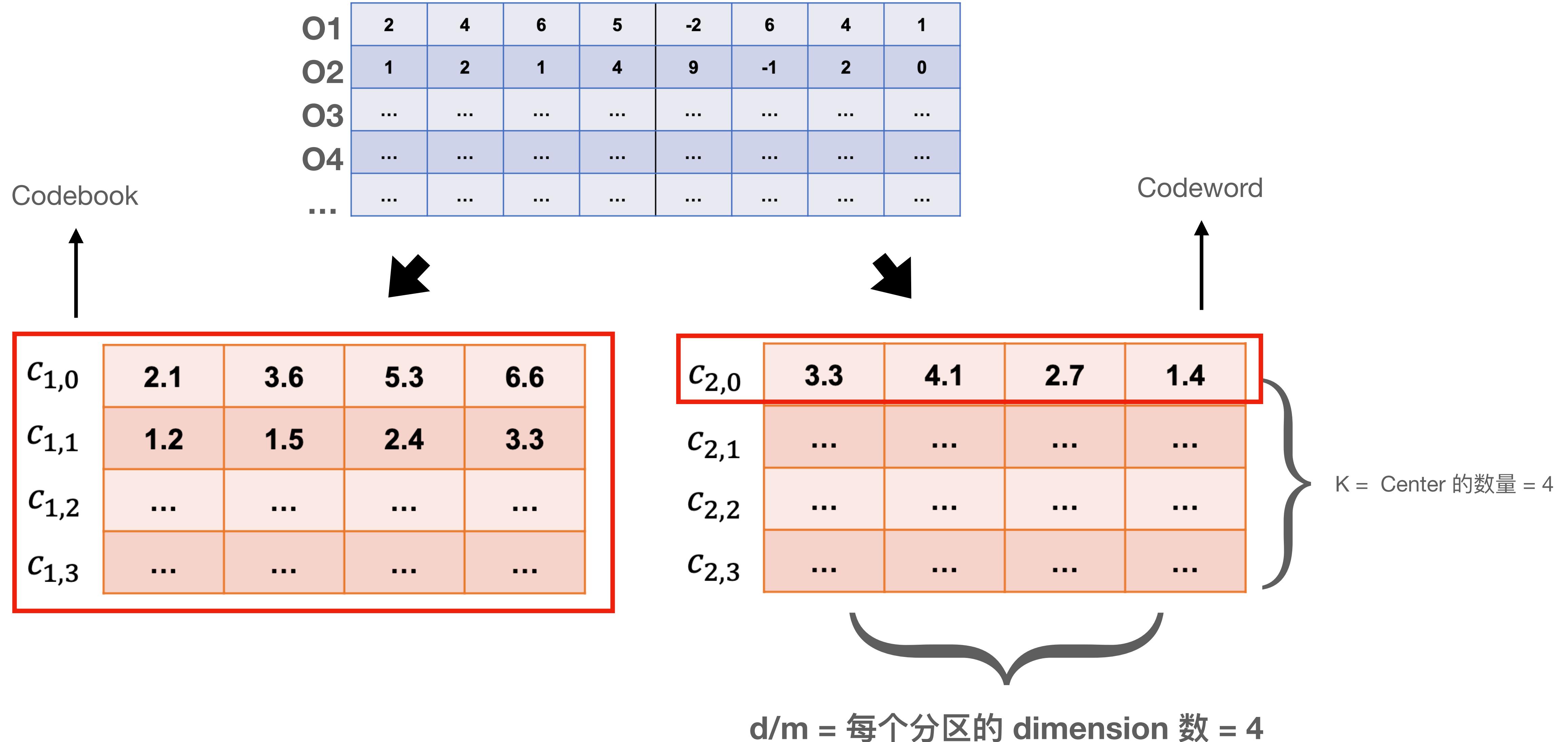
$m = \text{partition 的个数} = 2$

$d/m = \text{每个分区的 dimension 数} = 4$

例子：

• Step 2: generate codebooks (e.g., k-means) 

使用 K-mean Cluster 的方法生成两个 codebook



例子：

• Step 3: encode data



O1	2	4	6	5	-2	6	4	1
O2	1	2	1	4	9	-1	2	0
O3
O4
...



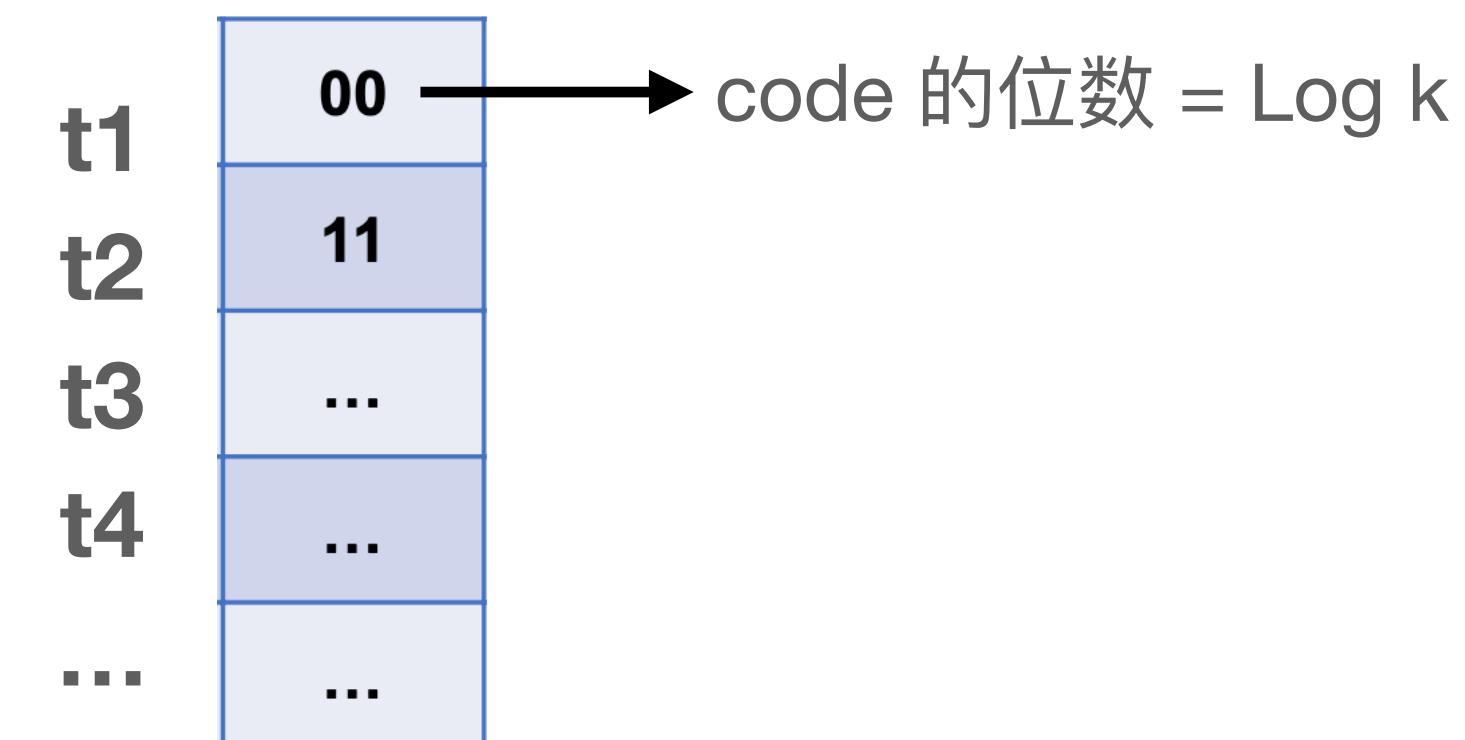
00	$c_{1,0}$	2.1	3.6	5.3	6.6
01	$c_{1,1}$	1.2	1.5	2.4	3.3
10	$c_{1,2}$
11	$c_{1,3}$

$c_{2,0}$	3.3	4.1	2.7	1.4	00
$c_{2,1}$	01
$c_{2,2}$	10
$c_{2,3}$	11



t1	00
t2	01
t3	...
t4	...
...	...

每一个 Data 的每一个 Partition Vector
都对应 code book 里面的一个 center (codeword)
我们这个对应关系 encode 成 bit 的形式 (节省存储空间)

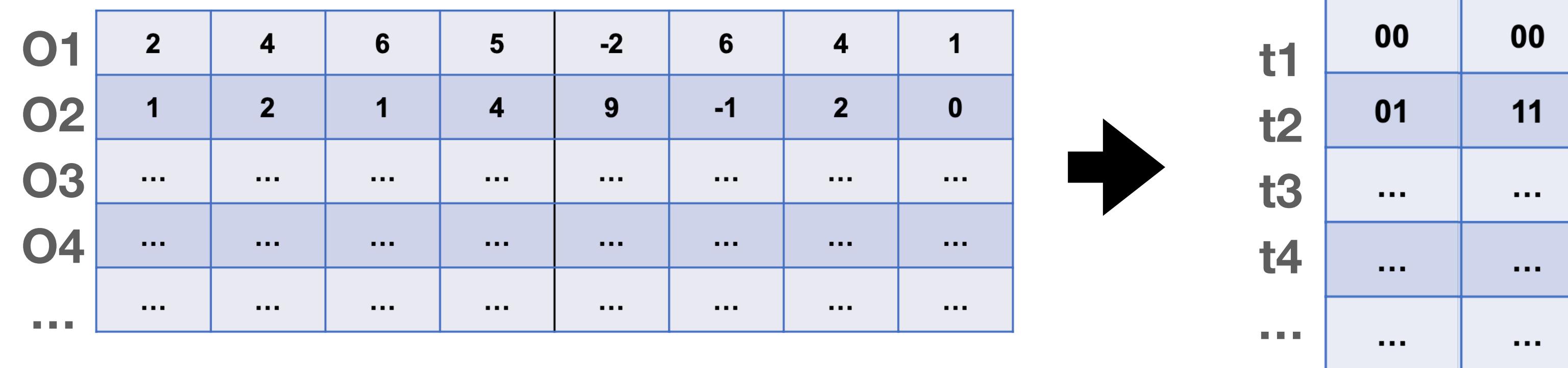


例子：

- Step 1: compute distance between q and codewords



q	1	3	5	4	-3	7	3	2
---	---	---	---	---	----	---	---	---



00	$c_{1,0}$	2.1	3.6	5.3	6.6	$c_{2,0}$	3.3	4.1	2.7	1.4	00
01	$c_{1,1}$	1.2	1.5	2.4	3.3	$c_{2,1}$	01
10	$c_{1,2}$	$c_{2,2}$	10
11	$c_{1,3}$	$c_{2,3}$	11

遍历 data 的 encode table 对于每一个 计算 $d^2(q, t) = \sum_{i=1}^d (q_i - p_i)^2$

以 t1 为例:

2.1	3.6	5.3	6.6	3.3	4.1	2.7	1.4
-----	-----	-----	-----	-----	-----	-----	-----

 每一位跟 q 相减, 平方, 加和

Query Processing

- Compute ADC for every point in the database
 - How? 这里其实可以提前计算，并保存，避免重复计算
- Candidate = those with the l smallest AD
- [Optional] Reranking (if $l > 1$):
 - Load the data vectors and compute the actual Euclidean distance
 - Return the one with the smallest distance

Product Quantization

- Idea
 - Partition the dimension into m partitions
 - Accordingly a vector \Rightarrow subvectors
 - Use separate VQ with k codewords for each chunk
- Example:
 - 8-dim vector decomposed into $m = 2$ subvectors
 - Each codebook has $k = 4$ codewords, (i.e., $c_{i,j}$)
 - Total space in bits:
 - Data: $n \cdot m \cdot \log(k)$
 - Codebook: $m \cdot \frac{d}{m} \cdot k \cdot 32$

极大可能出大题，重在理解算法和流程



- **Data Streams**

考点1 : Data Stream 和 传统 RDBMS 的对比

DBMS vs. DSMS #2

- Traditional DBMS:
 - stored sets of relatively **static** records with no pre-defined notion of time
 - good for applications that require **persistent data storage** and **complex querying**
- DSMS
 - support on-line analysis of rapidly **changing** data streams
 - data stream: **real-time**, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items, too large to store entirely, no ending
 - **continuous queries**

考点1：Data Stream 和 传统 RDBMS 的对比

DBMS vs. DSMS #3

DBMS

- Persistent relations
(relatively static, stored)
- One-time queries
- Random access
- “Unbounded” disk store
- Only current state matters
- No real-time services
- Relatively low update rate
- Data at any granularity
- Assume precise data
- Access plan determined by query processor,
physical DB design

DSMS

- Transient streams
(on-line analysis)
- Continuous queries (CQs)
- Sequential access
- Bounded main memory
- Historical data is important
- Real-time requirements
- Possibly multi-GB arrival rate
- Data at fine granularity
- Data stale/imprecise
- Unpredictable/variable data arrival and
characteristics

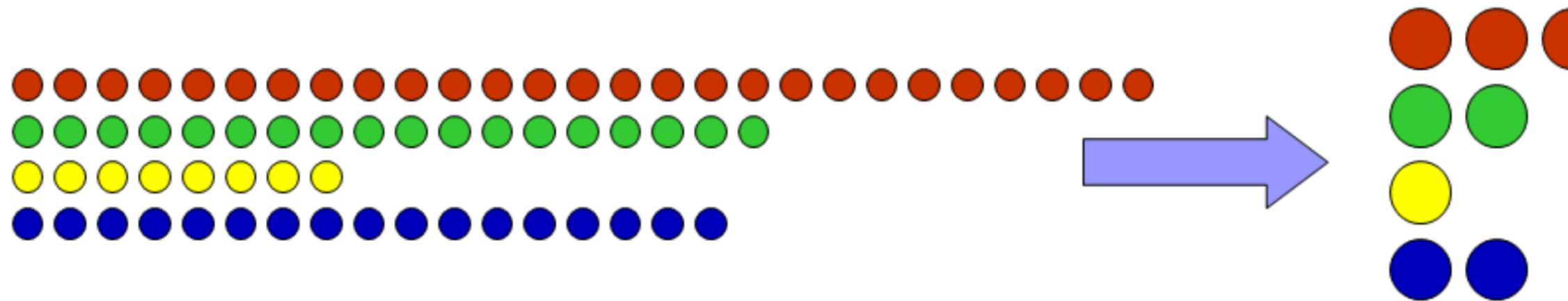
考点1.5 : Data Stream 的三种查询类型

- Sampling
- Sliding window query
- Filtering

考点2 : Sampling on Data Stream

Sampling from a Data Stream

- Since we can not store the entire stream, one obvious approach is to store a **sample**

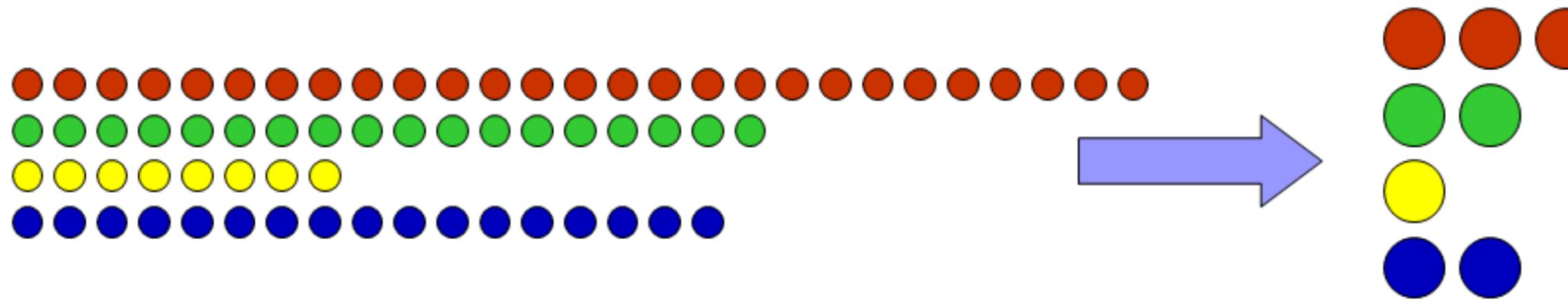


- Two different problems:
 - (1) Sample a **fixed proportion** of elements in the stream (say 1 in 10)
 - As the stream grows the sample also gets bigger
 - (2) Maintain a **random sample of fixed size** over a potentially infinite stream
 - As the stream grows, the sample is of fixed size
 - At any “time” t we would like a random sample of s elements
- **What is the property of the sample we want to maintain?**
 - For all time steps t , each of t elements seen so far has equal probability of being sampled

考点2 : Sampling on Data Stream

Sampling from a Data Stream

- Since we can not store the entire stream, one obvious approach is to store a **sample**



- Two different problems:
 - (1) Sample a **fixed proportion** of elements in the stream (say 1 in 10)
 - As the stream grows the sample also gets bigger
 - (2) Maintain a **random sample of fixed size** over a potentially infinite stream
 - As the stream grows, the sample is of fixed size
 - At any “time” t we would like a random sample of s elements
- **What is the property of the sample we want to maintain?**
 - For all time steps t , each of t elements seen so far has equal probability of being sampled

考点2 : Sampling on Data Stream (fix proportion)

Generalized Problem and Solution

- Problem: Give a data stream, take a sample of fraction a/b .
- Stream of tuples with keys:
 - Key is some subset of each tuple's components
 - e.g., tuple is (user, search, time); key is **user**
 - Choice of key depends on application
- To get a sample of a/b fraction of the stream:
 - Hash each tuple's key uniformly into **b** buckets
 - Pick the tuple if its hash value is at most **a**



How to generate a **30% sample?**

Hash into $b=10$ buckets, take the tuple if it hashes to one of the **first 3 buckets**

考点2 : Sampling on Data Stream (fix size) 水塘抽样

Solution: Fixed Size Sample

水塘抽样

- **Algorithm (a.k.a. Reservoir Sampling)**

- Store all the first s elements of the stream to S
- Suppose we have seen $n-1$ elements, and now the n^{th} element arrives ($n > s$)
 - With probability s/n , keep the n^{th} element, else discard it
 - If we picked the n^{th} element, then it replaces one of the s elements in the sample S , picked uniformly at random

- **Claim:** This algorithm maintains a sample S with the desired property:
 - After n elements, the sample contains each element seen so far with probability s/n

考点2 : Sampling on Data Stream (fix size) 水塘抽样

假设 Data Stream 流有1亿个， 我们的内存只够存 3 个
通过水塘抽样， 保证1亿条数据过去后， 随机抽取 3 个 Data 存下来：

一开始从空开始接受数据， 前三条数据100%留下来
从第四条开始：

4th 有 $3/4$ 的概率留下来

5th 有 $3/5$ 的概率留下来

6th 有 $3/6$ 的概率留下来

...

n th 有 $3/n$ 的概率留下来

一旦确定某条数据留下来， 原来存下来的3条， 有相同的概率被新的替换

这样看来好像每一次的概率并不相等， 其实并不是这样

我们要看的是最终进入 S 中的概率

虽然第4个数进入set的概率比较大,但是到最后他被替换的概率也很大

每一个数留下来的概率都是 S/N (数学证明不大可能考)

Sliding Windows

- A useful model of stream processing is that queries are about a **window** of length N – the N most recent elements received
- Interesting case: N is so large that the data cannot be stored in memory, or even on disk
 - Or, there are so many streams that windows for all cannot be stored

考点3 :Sliding Window

- A useful model of stream processing is that queries are about a **window** of length N – the N most recent elements received
- Interesting case: N is **so large** that the data cannot be stored in memory, or even on disk
 - Or, there are so many streams that windows for all cannot be stored

N = 7

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past

Future →

考点4 : Sliding Window - DGIM 算法流程

- Problem:

- Given a stream of 0s and 1s
- Be prepared to answer queries of the form:
How many 1s are in the last k bits? where $k \leq N$

+

What if we cannot afford to store N bits?

- E.g., we're processing 1 billion streams and $N = 1 \text{ billion}$

=

DGIM Algorithm

Representing a Stream by Buckets

- The right end of a bucket is always a position with a 1
- Every position with a 1 is in some bucket 也就是说有些0是不在 bucket 里面的
- Either **one** or **two** buckets with the same **power-of-2 number** of 1s
- Buckets do not overlap in timestamps
- **Buckets are sorted by size**
 - Earlier buckets are not smaller than later buckets
- Buckets disappear when their end-time is $> N$ time units in the past

考点4 : Sliding Window - DGIM 算法流程-更新

Updating Buckets

- When a new bit comes in, drop the last (oldest) bucket if its end-time is prior to N time units before the current time
- 2 cases: Current bit is 0 or 1
 - If the current bit is 0: no other changes are needed
 - If the current bit is 1:
 - (1) Create a new bucket of size 1, for just this bit
 - End timestamp = current time
 - (2) If there are now three buckets of size 1, combine the oldest two into a bucket of size 2
 - (3) If there are now three buckets of size 2, combine the oldest two into a bucket of size 4
 - (4) And so on ...

考点4 : Sliding Window - DGIM 算法流程-更新

Demo

Example: Updating Buckets

Current state of the stream:

10010101100010110 101010101010110 10101010101110 1010101110101000 10110010

Bit of value 1 arrives

0010101100010110 101010101010110 10101010101110 1010101110101000 101100101

Two white buckets get merged into a yellow bucket

0010101100010110 101010101010110 10101010101110 1010101110101000 10110010101

Next bit 1 arrives, new orange white is created, then 0 comes, then 1:

0101100010110 101010101010110 10101010101110 1010101110101000 101100101101

Buckets get merged...

0101100010110 101010101010110 10101010101110 1010101110101000 101100101101

State of the buckets after merging

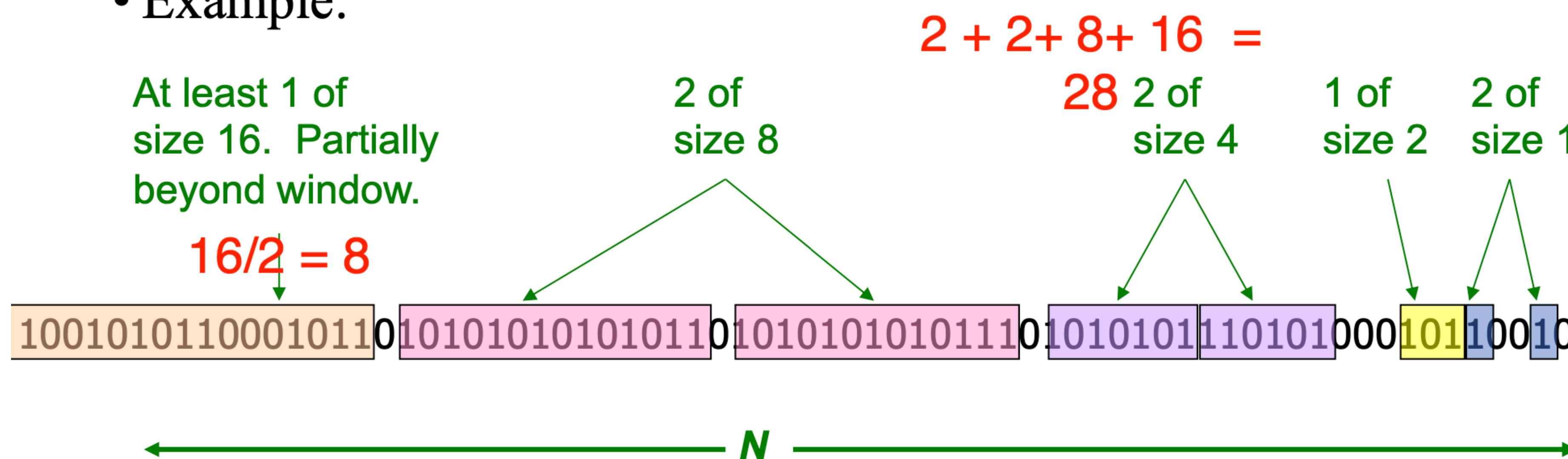
0101100010110 1010101010110101010101110 1010101110101000 101100101101

考点5 : Sliding Window - DGIM 算法流程-query

How to Query?

- Solution gives approximate answer, never off by more than 50%

- To estimate the number of 1s in the most recent N bits:
 - Sum the sizes of all buckets but the last
 - (note “size” means the number of 1s in the bucket)
 - Add half the size of the last bucket
- Remember: We do not know how many 1s of the last bucket are still within the wanted window
- Example:



考点6 : Sliding Window - DGIM Bucket 的时间戳

DGIM: Timestamps

- Each bit in the stream has a timestamp, starting from 1, 2, ...
- Record timestamps modulo N (**the window size**), so we can represent any **relevant** timestamp in $O(\log_2 N)$ bits $0 \leq \text{timestamp mod } N \leq N - 1$
 - E.g., given the windows size 40 (N), timestamp 123 will be recorded as 3, and thus the encoding is on 3 rather than 123

考点7 : Sliding Window - DGIM Bucket 的表示 / 单个bucket的空间复杂度

DGIM: Buckets

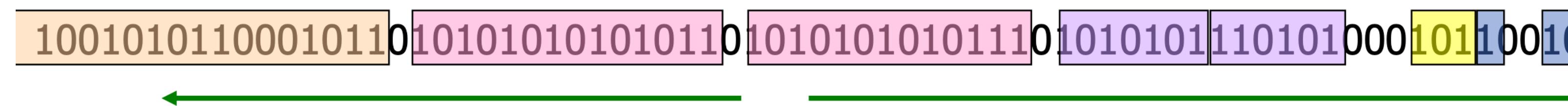
只需要把最右一位 bit 的timestamp 作为这个 bucket 的 timestamp

- A bucket in the DGIM method is a record consisting of:

- (A) The timestamp of its end [$O(\log N)$ bits] $0 \leq$ 时间戳 mod $N \leq N - 1$
- (B) The number of 1s between its beginning and end [$O(\log\log N)$ bits]

- Constraint on buckets:

- Number of 1s must be a power of 2
- That explains the $O(\log\log N)$ in (B) above 可以用 $\log(r)$ bit 来表示 r



$$N = 2^{**r}$$

$$r = \log(N)$$

Question 4 Mining Data Streams

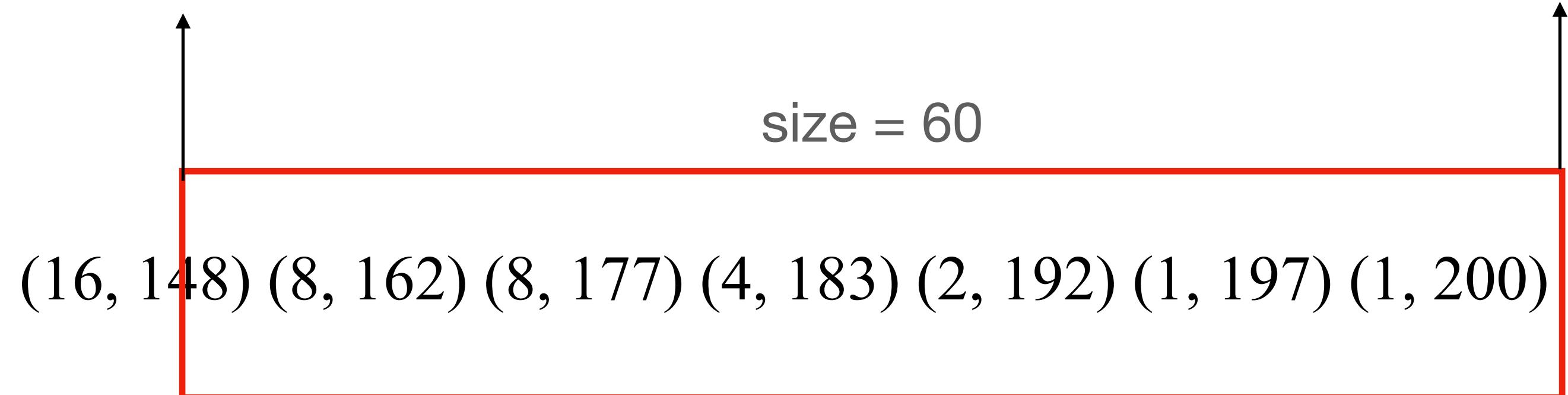
Suppose we are maintaining a count of 1s using the DGIM method. We represent a bucket by (i, t) , where i is the number of 1s in the bucket and t is the bucket timestamp (time of the most recent 1).

Consider that the current time is 200, window size is 60, and the current list of buckets is: $(16, 148)$ $(8, 162)$ $(8, 177)$ $(4, 183)$ $(2, 192)$ $(1, 197)$ $(1, 200)$. At the next ten clocks, 201 through 210, the stream has 0101010101. What will the sequence of buckets be at the end of these ten inputs?

答：

last st in window = 141

first st in window = 200



201	202	203	204	205	206	207	208	209	210
0	1	0	1	0	1	0	1	0	1

Answer

There are 5 1s in the stream. Each one will update to windows to be: [each step 2 marks]

- (1) (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(1, 197)(1, 200), (1, 202)
=> (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (1, 202)
- (2) (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (1, 202), (1, 204)
- (3) (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (1, 202), (1, 204),
(1, 206)
=> (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (2, 204), (1, 206)
=> (16, 148)(8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206)
- (4) Windows Size is 60, so (16,148) should be dropped.
(16, 148)(8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206), (1, 208) => (8,
162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206), (1, 208)
- (5) (8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206), (1, 208), (1, 210)
=> (8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (2, 208), (1, 210)

考点 8 : Bloom Filtering 布隆过滤器-算法流程

Filtering Data Streams

- Each element of data stream is a tuple
- Given a list of keys S
- **Determine which tuples of stream are in S**
- Obvious solution: Hash table
 - But suppose we **do not have enough memory** to store all of S in a hash table
 - E.g., we might be processing millions of filters on the same stream

考点 8 : Bloom Filtering 布隆过滤器-算法流程

Bloom Filter Example

Size of bucket n = 10

size of bucket = 2

- Consider a ~~Bloom filter of size m=10~~ and number of hash functions k=3.
Let H(x) denote the result of the three hash functions.
- The 10-bit array is initialized as below

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

- Insert x_0 with $H(x_0) = \{1, 4, 9\}$

0	1	2	3	4	5	6	7	8	9
0	1	0	0	1	0	0	0	0	1

- Insert x_1 with $H(x_1) = \{4, 5, 8\}$

0	1	2	3	4	5	6	7	8	9
0	1	0	0	1	1	0	0	1	1

- Query y_0 with $H(y_0) = \{0, 4, 8\} \Rightarrow ???$
- Query y_1 with $H(y_1) = \{1, 5, 8\} \Rightarrow ???$ **False positive!**
- Another Example: <https://llimllib.github.io/bloomfilter-tutorial/>

考点 8 : Bloom Filtering 布隆过滤器-算法流程

Bloom Filter

- Consider: $|S| = m$, $|B| = n$
- Use k independent hash functions h_1, \dots, h_k
- **Initialization:**
 - Set B to all 0s
 - Hash each element $s \in S$ using each hash function h_i , set $B[h_i(s)] = 1$ (for each $i = 1, \dots, k$)
- **Run-time:**
 - When a stream element with key x arrives
 - If $B[h_i(x)] = 1$ for all $i = 1, \dots, k$ then declare that x is in S
 - That is, x hashes to a bucket set to 1 for every hash function $h_i(x)$
 - Otherwise discard the element x

考点 8 : Bloom Filtering 布隆的特点

- **Creates false positives but no false negatives**

- If the item is in S we surely output it, if not we may still output it

False positive : 某个 Data 本来不应该成为 Candidate, 却最终被选为 Candidate 了

False negative : 某个 Data 本来应该成为 Candidate, 却最终没被选为 Candidate

推荐阅读: <https://blog.csdn.net/jiaomeng/article/details/1495500>

考点 8 : Bloom Filtering 布隆的错误率估计 / 最优的哈希函数个数

Bloom Filter – Analysis

- What fraction of the bit vector B are 1s?
 - Throwing $k \cdot m$ darts at n targets
 - So fraction of 1s is $(1 - e^{-km/n})$
- But we have k independent hash functions and we only let the element x through if all k hash element x to a bucket of value 1
- So, false positive probability = $(1 - e^{-km/n})^k$

考点 8 : Bloom Filtering 布隆的错误率估计 / 最优的哈希函数个数

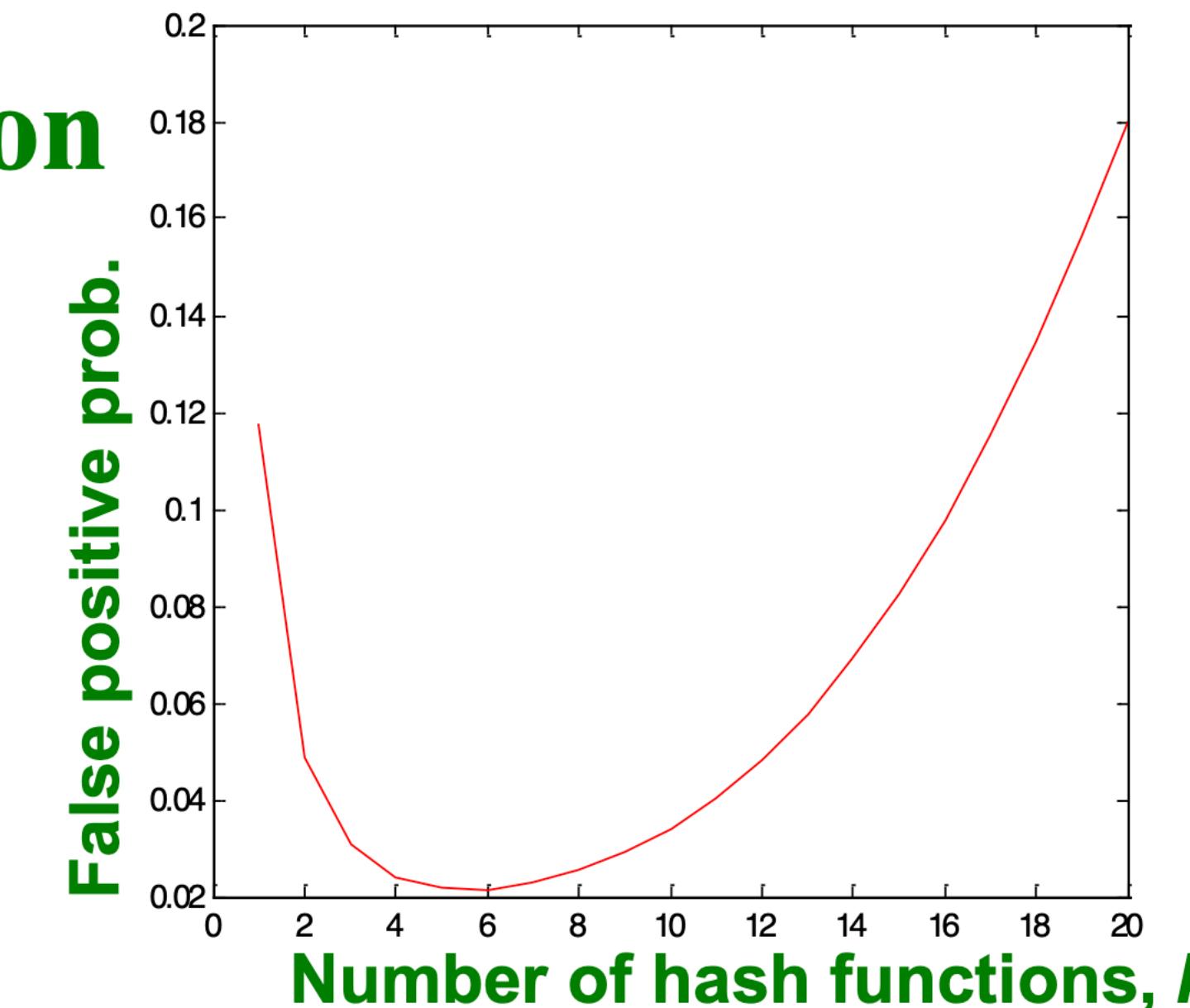
Bloom Filter – Analysis (2)

- **$m = 1 \text{ billion}, n = 8 \text{ billion}$**

- $k = 1: (1 - e^{-1/8}) = 0.1175$
- $k = 2: (1 - e^{-1/4})^2 = 0.0493$

- **What happens as we keep increasing k ?**

- “Optimal” value of k : $n/m \ln(2)$
 - **In our case:** Optimal $k = 8 \ln(2) = 5.54 \approx 6$
 - Error at $k = 6: (1 - e^{-1/6})^2 = 0.0235$





- **Recommander System**

考点1 : RS Model

Formal Model

- $X = \text{set of Customers}$
- $S = \text{set of Items}$
- Utility function $u: X \times S \rightarrow R$
 - $R = \text{set of ratings}$
 - R is a totally ordered set
 - e.g., 0-5 stars, real number in $[0,1]$
- Utility Matrix 事实上就是想一种方法把空缺的rating 预测出来

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

Key Problems

- Gathering “known” ratings for matrix
 - How to collect the data in the utility matrix
- Extrapolate unknown ratings from the known ones
 - Mainly interested in high unknown ratings
 - We are not interested in knowing what you don’t like but what you like
- Evaluating extrapolation methods
 - How to measure success/performance of recommendation methods

Gathering Ratings

- Explicit

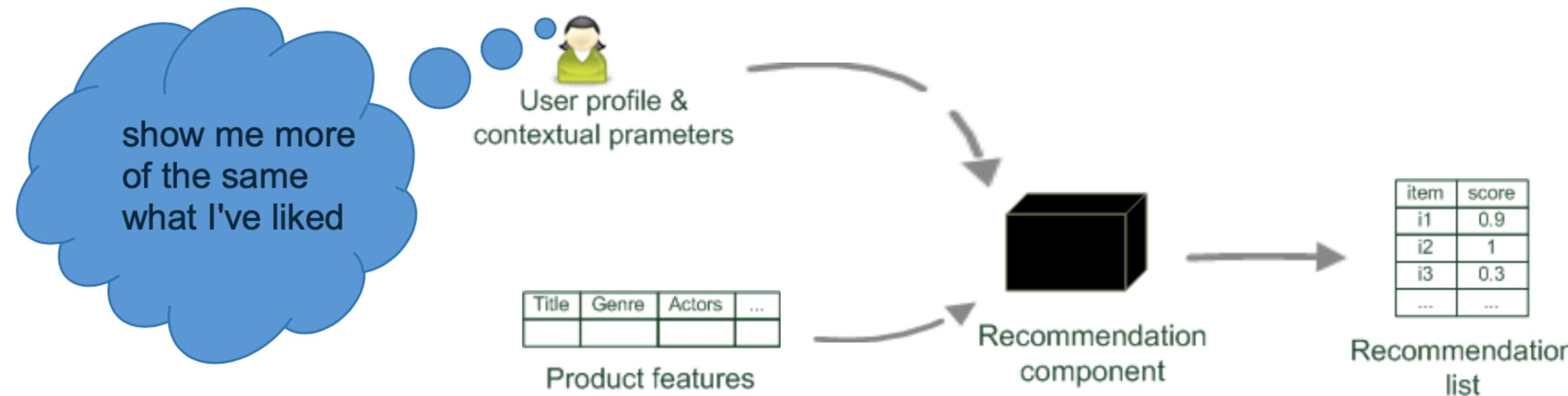
- Ask people to rate items 5 星好评
- Doesn't work well in practice – people can't be bothered

- Implicit

- Learn ratings from user actions 点赞/评论/转发/收藏

- E.g., purchase implies high rating

Content-based Recommendation



- Step 1. 想办法表示所有 Item (Item Profile)
- Step 2. 想办法表示 Given User (User Profile)
- Step 3. 计算 Given User 和每一个 item 的 Cosin Similarity 选最大的那个推荐给 ta

Step 1. 想办法表示所有 Item (Item Profile)

Item Profiles

- For each item, create an **item profile**
- Profile is a set (vector) of features
 - **Movies:** author, title, actor, director,...
 - **Text:** Set of “important” words in document
- How to pick important features?
 - Usual heuristic from text mining is **TF-IDF** (Term frequency * Inverse Doc Frequency)
 - **Term ... Feature**
 - **Document ... Item**

Step 1. 想办法表示所有 Item (Item Profile)

	类别	价格	重量
Item1	0	30	1
Item2	1	15	4
Item3	1	999	3
Item4	0	56	1

$$\text{Item1} = (0, 30, 1)$$

$$\text{Item2} = (1, 15, 4)$$

$$\text{Item3} = (1, 999, 3)$$

$$\text{Item4} = (0, 56, 1)$$

Step 2. 想办法表示 Given User (User Profile)

User Profiles and Prediction

- User profile possibilities:
 - Weighted average of rated item profiles
 - Variation: weight by difference from average rating for item 2种不同的方法生成 User profile
 - ...

Rating	Item1	Item2	Item3	Item3
User1	0.5	0.7	0.2	-
User2	-	1	0	-
Avg	0.3	0.8	1	0.9

Weighted average

$$\text{User1} = (0.5 * \text{item1} + 0.7 * \text{item2} + 0.2 * \text{item3}) / 3$$

$$\text{User2} = (1 * \text{item2} + 0 * \text{item3}) / 2$$

Variation

$$\text{User1} = ((0.5 - 0.3) * \text{item1} + (0.7 - 0.8) * \text{item2} + (0.2 - 1) * \text{item3}) / 3$$

$$\text{User2} = ((1 - 0.8) * \text{item2} + (0 - 1) * \text{item3}) / 2$$

Step 3. 计算 Given User 和每一个 item 的 Cosin Similarity 选最大的那个推荐给 ta

- Given user profile x and item profile i , estimate

$$u(x, i) = \cos(x, i) = \frac{x \cdot i}{\|x\| \cdot \|i\|}$$

例如想给 User 2 推荐 ta 没看过的：就推荐 $\text{Max}(\cos(\text{User2}, \text{Item1}), \cos(\text{User2}, \text{Item4}))$

考点 3 : Content-base RS 优缺点

Pros: Content-based Approach

- +: No need for data on other users
- +: Able to recommend to users with unique tastes
- +: Able to recommend new & unpopular items
 - No first-rater problem
- +: Able to provide explanations
 - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

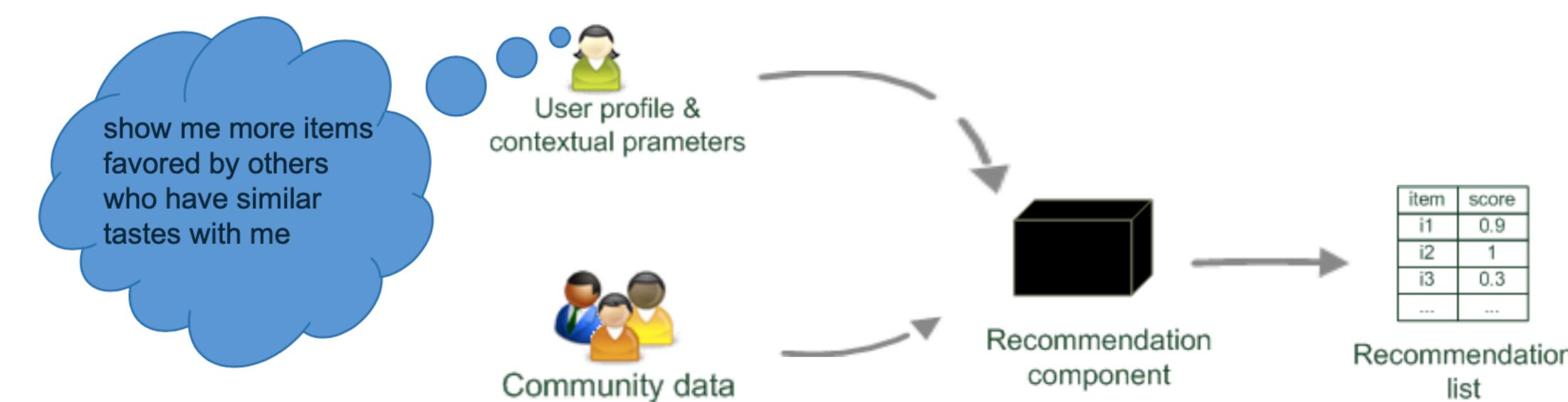
Cons: Content-based Approach

- -: **Finding the appropriate features is hard**
 - E.g., images, movies, music
- -: **Recommendations for new users**
 - **How to build a user profile?** Cold Start 冷启动问题
- -: **Overspecialization** 信息茧房
 - Never recommends items outside user's content profile
 - People might have multiple interests
 - **Unable to exploit quality judgments of other users**

考点 4 : 协同过滤 Collaborative Filtering

Collaborative Filtering { User-User / User-based / Memory-based
Item-item / Item-based

Collaborative Filtering



考点 5 协同过滤：User-User / User-based / Memory-based

User-based Nearest-Neighbor Collaborative Filtering

- The basic technique

- Given an "active user" (Alice) and an item i not yet seen by Alice
 - find a set of users (peers/nearest neighbors) who liked the same items as Alice in the past and who have rated item i
 - use, e.g. the average of their ratings to predict, if Alice will like item i
 - do this for all items Alice has not seen and recommend the best-rated

- Basic assumption and idea

- If users had similar tastes in the past they will have similar tastes in the future
- User preferences remain stable and consistent over time

考点 5 协同过滤：User-User / User-based / Memory-based

- Step 0. User 怎么表示? -> 用 Rating 表示
- Step 1. 想办法找到你的同类 (Similar Users)
- Step 2. 想办法用你同类的 Rating 来推测你的 Rating
- Step 3. 推荐最大的 Rating 的 Item 给你

考点 5 协同过滤：User-User / User-based / Memory-based

Step 0. User 怎么表示? -> 用 Rating 表示

Step 1. 想办法找到你的同类 (Similar Users) 如何定义 Similar ?

Rating	Item1	Item2	Item3	Item4	Item5
User1	1	-	-	1	3
User2	1	-	2	2	-

- **Jaccard similarity measure** $\frac{||r_x \cap r_y||}{||r_x \cup r_y||}$

$$\begin{aligned} \text{User1} &= (1, 4, 5) \\ \text{User2} &= (1, 3, 4) \end{aligned} \quad \text{sim}(\text{User1}, \text{User2}) = 1 / 5$$

- **Cosine similarity measure**

$$\cdot \text{sim}(x, y) = \cos(r_x, r_y) = \frac{r_x \cdot r_y}{\|r_x\| \cdot \|r_y\|}$$

$$\begin{aligned} \text{User1} &= (1, 0, 0, 1, 3) \\ \text{User2} &= (1, 0, 2, 2, 0) \end{aligned} \quad \text{sim}(\text{User1}, \text{User2}) = \cos(\text{User1}, \text{User2})$$

- **Pearson correlation coefficient**

$$\cdot S_{xy} = \text{items rated by both users } x \text{ and } y$$

$$\text{sim}(x, y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)(r_{ys} - \bar{r}_y)}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \bar{r}_y)^2}}$$

$$S_{xy} = (\text{item1}, \text{item4})$$

$$R_x = (1, 1) \quad \bar{r}_x = (1 + 1) / 2 = 1$$

$$R_y = (1, 2) \quad \bar{r}_y = (1 + 2) / 2 = 1.5$$

考点 5 协同过滤：User-User / User-based / Memory-based

Finding “Similar” Users

- Let r_x be the vector of user x 's ratings

• **Jaccard similarity measure** $\frac{||r_x \cap r_y||}{||r_x \cup r_y||}$

- Problem: Ignores the value of the rating

• **Cosine similarity measure**

$$\text{sim}(x, y) = \cos(r_x, r_y) = \frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$$

- Problem: Treats missing ratings as “negative”

• **Pearson correlation coefficient**

- S_{xy} = items rated by both users x and y

$$\text{sim}(x, y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)(r_{ys} - \bar{r}_y)}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \bar{r}_y)^2}}$$

$\bar{r}_x, \bar{r}_y \dots$ avg.
rating of x, y

$$r_x = [* , _, _, *, **]$$

$$r_y = [* , _, **, **, _]$$

r_x, r_y as sets:

$$r_x = \{1, 4, 5\}$$

$$r_y = \{1, 3, 4\}$$

r_x, r_y as points:

$$r_x = \{1, 0, 0, 1, 3\}$$

$$r_y = \{1, 0, 2, 2, 0\}$$

Similarity Metric (Cont')

- A popular similarity measure in user-based CF: **Pearson correlation**

$$sim(x, y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)(r_{ys} - \bar{r}_y)}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \bar{r}_y)^2}}$$

- Possible similarity values between -1 and 1;

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

sim = 0,85
sim = 0,70
sim = 0,00
sim = -0,79

Step 2. 想办法用你同类的 Rating 来推测你的 Rating

Rating Predictions

From similarity metric to recommendations:

- Let r_x be the vector of user x 's ratings
- Let N be the set of k users most similar to x who have rated item i
- Prediction for item s of user x :
 - $r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$
 - $r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$
 - Other options?
- Many other tricks possible...

Shorthand:

$$s_{xy} = sim(x, y)$$

Step 3. 推荐最大的 Rating 的 Item 给你

• Another view: Item-item

- Basic idea:
 - Use the similarity between items (and not users) to make predictions
 - For item i , find other similar items
 - Estimate rating for item i based on ratings for similar items
 - Can use same similarity metrics and prediction functions as in user-user model

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

s_{ij} ... similarity of items i and j

r_{xj} ... rating of user x on item j

$N(i;x)$... set items rated by x similar to i

考点 6 协同过滤：Item-item

- Step 0. item 怎么表示? -> 用所有 User 的 Rating 表示
- Step 1. 想办法找到你想去预测的那个 Item 的同类 (Similar Item)
- Step 2. 想办法用 ta 同类的 Rating 来推测这个 Item 的 Rating

考点 6 协同过滤：Item-item

Item-Item CF ($|N|=2$)

	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2				2	5	
6	1		3		3			2			4	

Legend:

- White square - unknown rating
- Yellow square - rating between 1 to 5

Item-Item CF ($|N|=2$)

	users											
	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	



- estimate rating of movie 1 by user 5

Step 0. item 怎么表示? -> 用所有 User 的 Rating 表示

Step 1. 想办法找到你想去预测的那个 Item 的同类 (Similar Item)

Item-Item CF ($|N|=2$)

adjust cosine similarity

	users												
	1	2	3	4	5	6	7	8	9	10	11	12	
1	1		3		?	5			5		4		
2			5	4			4			2	1	3	
3	3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2		
5			4	3	4	2					2	5	
6	6	1		3		3			2		4		

sim(1,m)

1.00

-0.18

0.41

-0.10

-0.31

0.59

Neighbor selection:

Identify movies similar to
movie 1, rated by user 5

Here we use adjust cosine similarity:

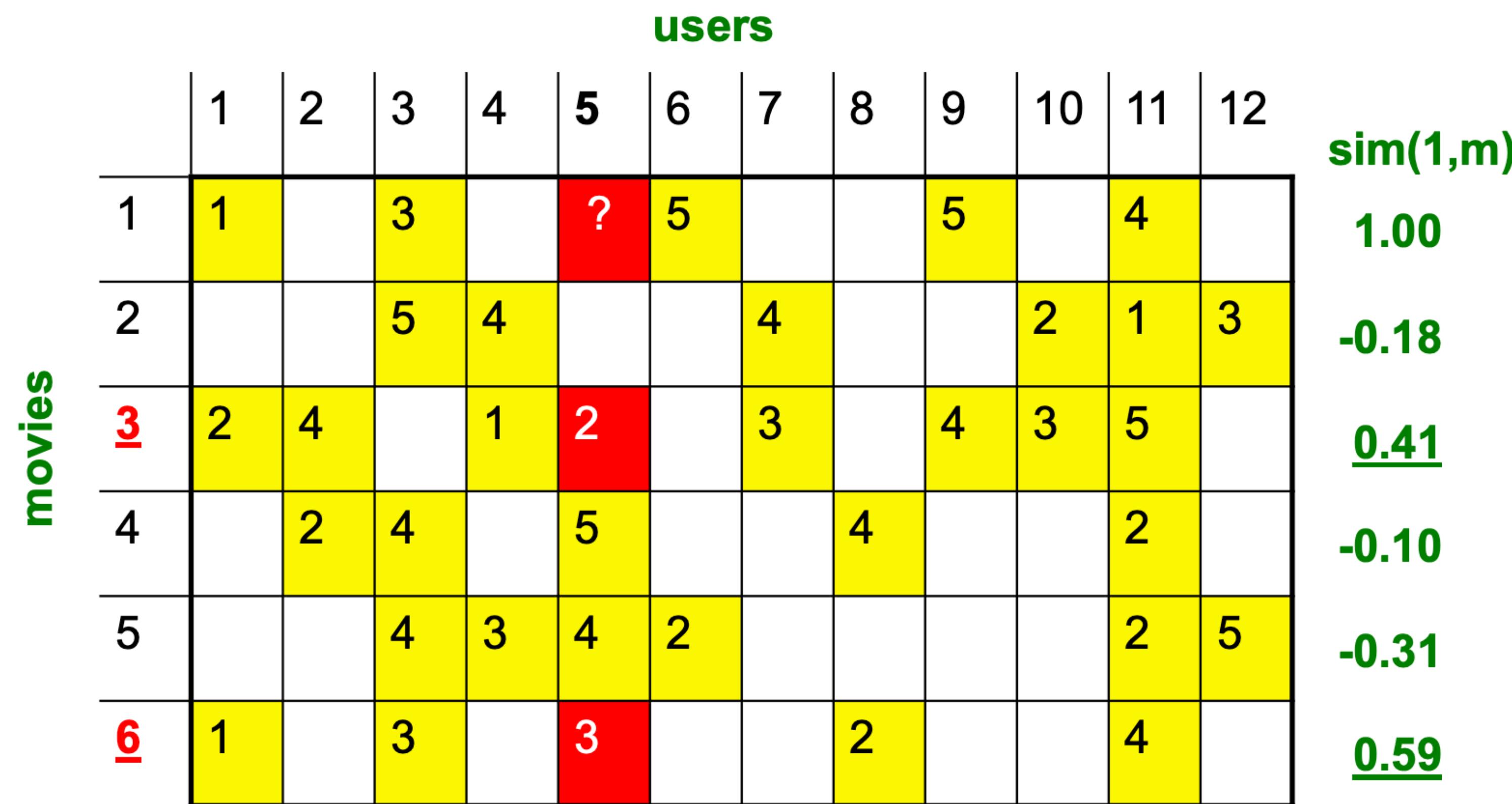
1) Subtract mean rating m_i from each movie i

$$m_1 = (1+3+5+4)/5 = 3.6$$

row 1: [-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]

2) Compute cosine similarities between rows

Item-Item CF ($|N|=2$)



Neighbor selection:

Identify movies similar to
movie 1, rated by user 5

40

Here we use adjust cosine similarity:

1) Subtract mean rating m_i from each movie i

$$m_1 = (1+3+5+4)/5 = 3.6$$

row 1: [-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]

2) Compute cosine similarities between rows

Item-Item CF ($|N|=2$)

	users											
	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		2.6	5			5		4	
2			5	4			4			2	1	3
3	3	2	4		1	2		3		4	3	5
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	6	1		3		3			2		4	

Predict by taking weighted average:

$$r_{1,5} = (0.41 \cdot 2 + 0.59 \cdot 3) / (0.41 + 0.59) = 2.6$$

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}}$$

Evaluating Predictions

- **Compare predictions with known ratings**
 - **Root-mean-square error (RMSE)**
 - $\sqrt{\sum_{xi} (r_{xi} - r_{xi}^*)^2}$ where r_{xi} is predicted, r_{xi}^* is the true rating of x on i
 - **Precision at top 10:**
 - % of those in top 10
 - **Rank Correlation:**
 - Spearman's *correlation* between system's and user's complete rankings
- **Another approach: 0/1 model**
 - **Coverage:**
 - Number of items/users for which system can make predictions
 - **Precision:**
 - Accuracy of predictions
 - **Receiver operating characteristic (ROC)**
 - Tradeoff curve between false positives and false negatives

Modeling Local & Global Effects

- **Global:**

- Mean movie rating: **3.7 stars**
- *The Sixth Sense* is **0.5 stars** above avg.
- Joe rates **0.2 stars** below avg.

⇒ **Baseline estimation:**

Joe will rate *The Sixth Sense* 4 stars

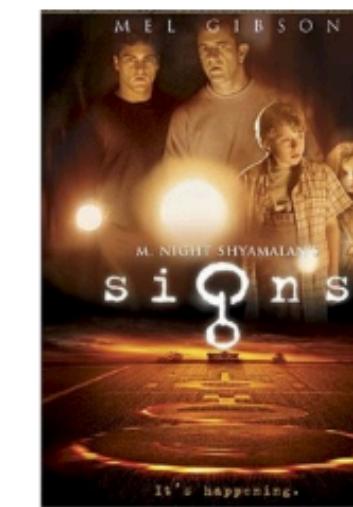


- **Local neighborhood (CF/NN):**

- Joe didn't like related movie *Signs*

⇒ **Final estimate:**

Joe will rate *The Sixth Sense* 3.8 stars



Modeling Local & Global Effects

- In practice we get better estimates if we model deviations:

$$\hat{r}_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$b_{xi} = \mu + b_x + b_i$$

μ = overall mean rating

b_x = rating deviation of user x
 $= (\text{avg. rating of user } x) - \mu$

b_i = (avg. rating of movie i) – μ

Problems/Issues:

- 1) Similarity measures are “arbitrary”
- 2) Pairwise similarities neglect interdependencies among users
- 3) Taking a weighted average can be restricting

Solution: Instead of s_{ij} use w_{ij} that we estimate directly from data

Idea: Interpolation Weights w_{ij}

- Use a **weighted sum** rather than **weighted avg.**:

$$\widehat{r}_{xi} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj})$$

- **A few notes:**

- $N(i; x)$... set of movies rated by user x that are similar to movie i
- w_{ij} is the interpolation weight (some real number)
 - We allow: $\sum_{j \in N(i,x)} w_{ij} \neq 1$
- w_{ij} models interaction between pairs of movies (it does not depend on user x)

Recommendations via Optimization

- Idea: Let's set values w such that they work well on known (user, item) ratings
- How to find such values w ?
- Idea: Define an objective function and solve the optimization problem
- Find w_{ij} that minimize SSE on training data!

$$J(w) = \sum_{x,i} \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj}) \right] - r_{xi} \right)^2$$

Predicted rating
True rating

- Think of w as a vector of numbers

考点 9 Matrix Factorization

Ratings as Products of Factors

- How to estimate the missing rating of user x for item i ?

$$\begin{aligned}
 & \hat{r}_{xi} = q_i \cdot p_x^T \\
 &= \sum_f q_{if} \cdot p_{xf} \\
 & q_i = \text{row } i \text{ of } Q \\
 & p_x = \text{column } x \text{ of } P^T
 \end{aligned}$$

\approx

users		
1	3	5
5	4	2.4
2	4	1
2	4	5
4	3	4
1	3	3
3	5	4
4	2	1
2	1	3
5	3	5
3	2	2
2	5	4
4	2	1
1	3	2

items		
.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3
.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

● factors

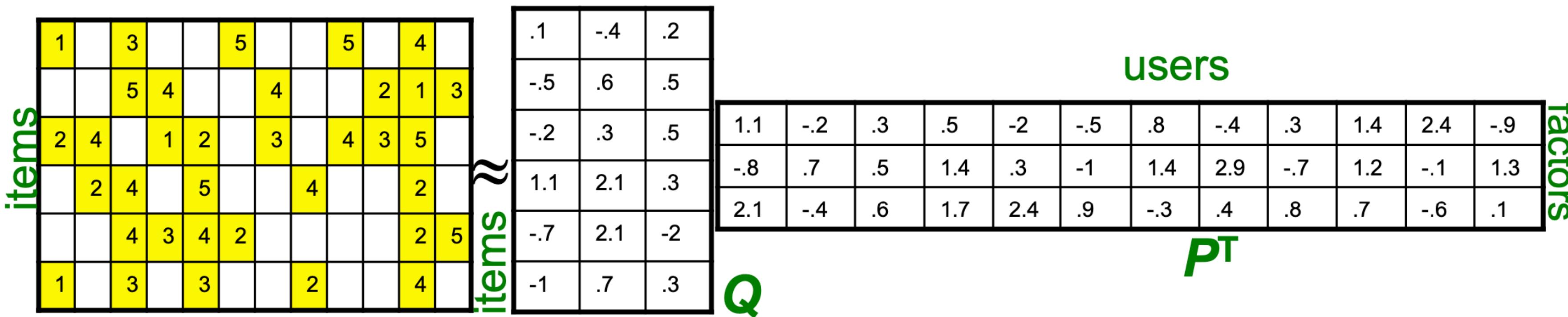
Q

users											
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

P^T

考点 9 Matrix Factorization

Matrix Factorization



- **SVD isn't defined when entries are missing!**
- **Use specialized methods to find P, Q**
 - $\min_{P,Q} \sum_{(i,x) \in R} (r_{xi} - q_i \cdot p_x^T)^2$
 - **Note:**
 - We don't require cols of P, Q to be orthogonal/unit length
 - P, Q map users/movies to a latent space

Pros/Cons of Collaborative Filtering

- + **Works for any kind of item**
 - No feature selection needed
- - **Cold Start:**
 - Need enough users in the system to find a match
- - **Sparsity:**
 - The user/ratings matrix is sparse
 - Hard to find users that have rated the same items
- - **First rater:**
 - Cannot recommend an item that has not been previously rated
 - New items, Esoteric items
- - **Popularity bias:**
 - Cannot recommend items to someone with unique taste
 - Tends to recommend popular items



- **Spark ML**

PySpark MLlib

- Classification/machine learning
 - Definition, training, test, evaluation, ...
 - Two-Step process
 - Cross-validation
- MLlib
 - Pipeline
- Ensemble learning
 - Stacking
- Not needed:
 - Naïve bayes Classification



- **Spark SQL**

DataFrame

- A DataFrame is a distributed collection of data organized into named columns.
- The DataFrames API is:
 - intended to enable wider audiences beyond “Big Data” engineers to leverage the power of distributed processing
 - inspired by data frames in R and Python (Pandas)
 - designed from the ground-up to support modern big data and data science applications
 - an extension to the existing RDD API

DataFrame

- DataFrames have the following features:
 - Ability to scale from kilobytes of data on a single laptop to petabytes on a large cluster
 - Support for a wide array of data formats and storage systems
 - State-of-the-art optimization and code generation through the Spark SQL Catalyst optimizer
 - Seamless integration with all big data tooling and infrastructure via Spark
 - APIs for Python, Java, Scala, and R

DataFrame 与 RDD 相比较：
结构上最大的不同就是有了 Column Name

但是注意！没有 Row Name，每一行为 Row() 的类型

More on DataFrames – Columnar Storage

- DataFrames use columnar storage
 - Transpose of row-based storage
 - Data in each column (with the same type) are packed together

Row Format

1	john	4.1
2	mike	3.5
3	sally	6.4

Column Format

1	2	3
john	mike	sally
4.1	3.5	6.4

考点 3 Operation

Good Luck !