

COMP9313

Week 5

Lecturer:

Elijah

- 
- Spark Shuffle 过程
 - LSH

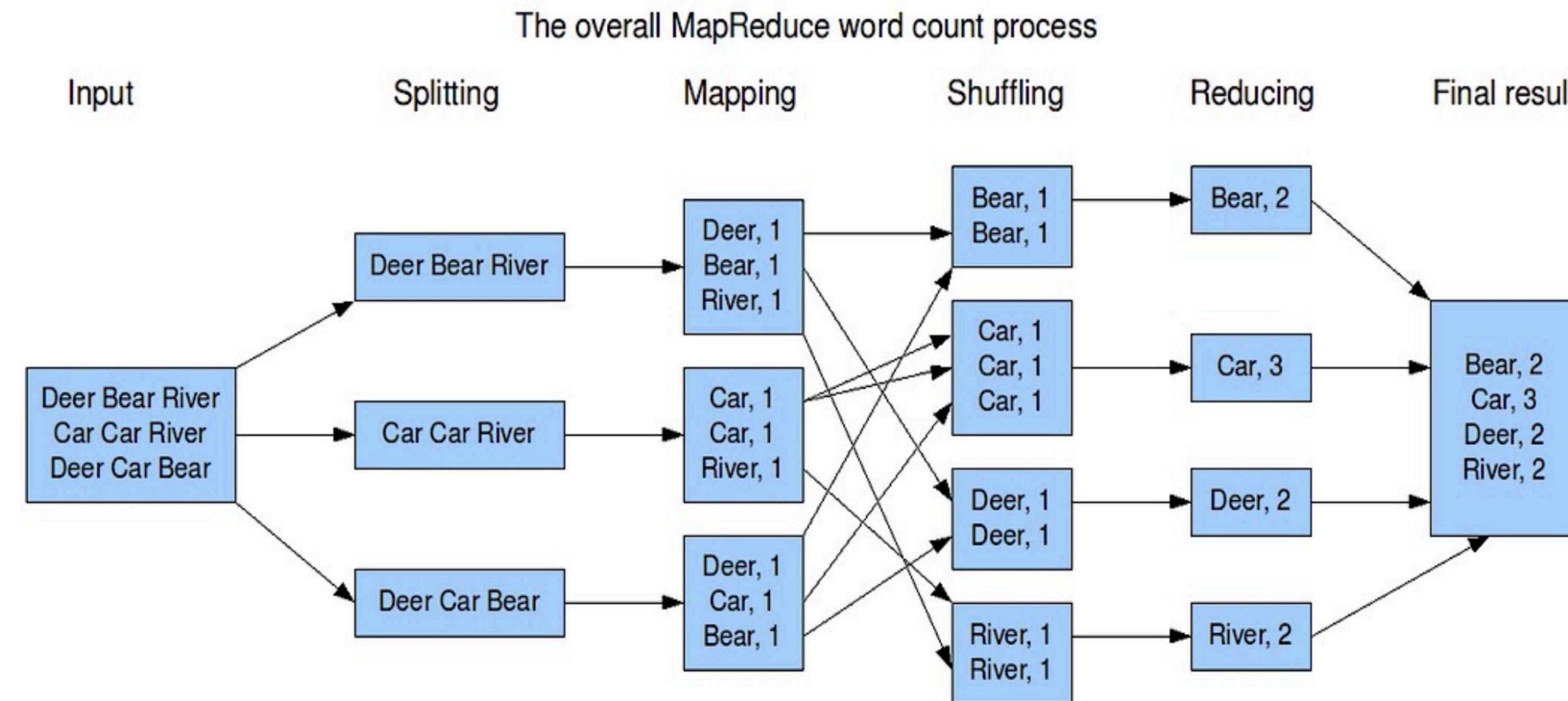


- **Spark Shuffle 过程**

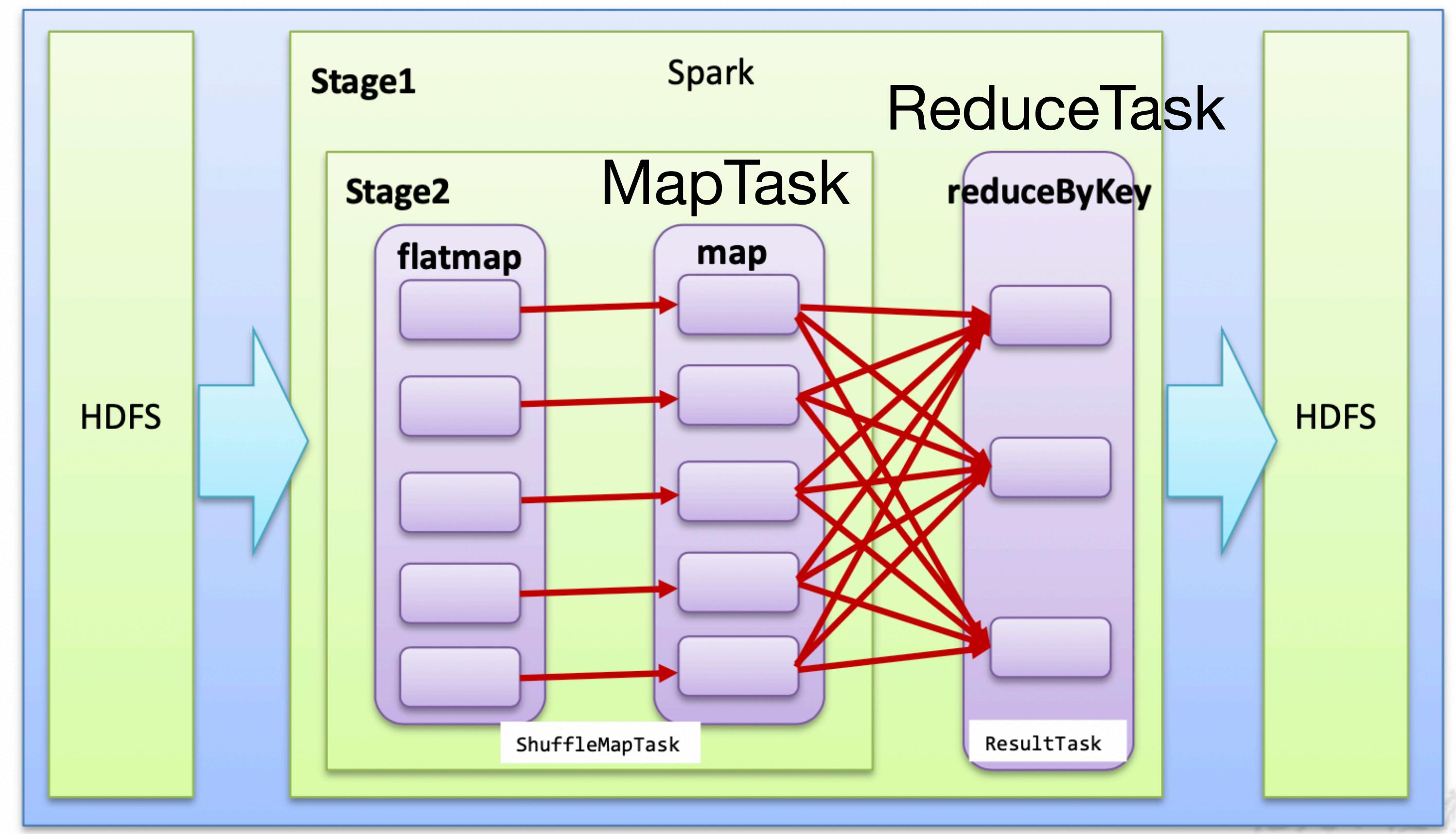
知识点：MR 的 Shuffle

Example of MapReduce in Hadoop

一提到 MR Shuffle 就要想到： 1. 2.



注意：Spark 中的 shuffle 不一定按 Key 排序

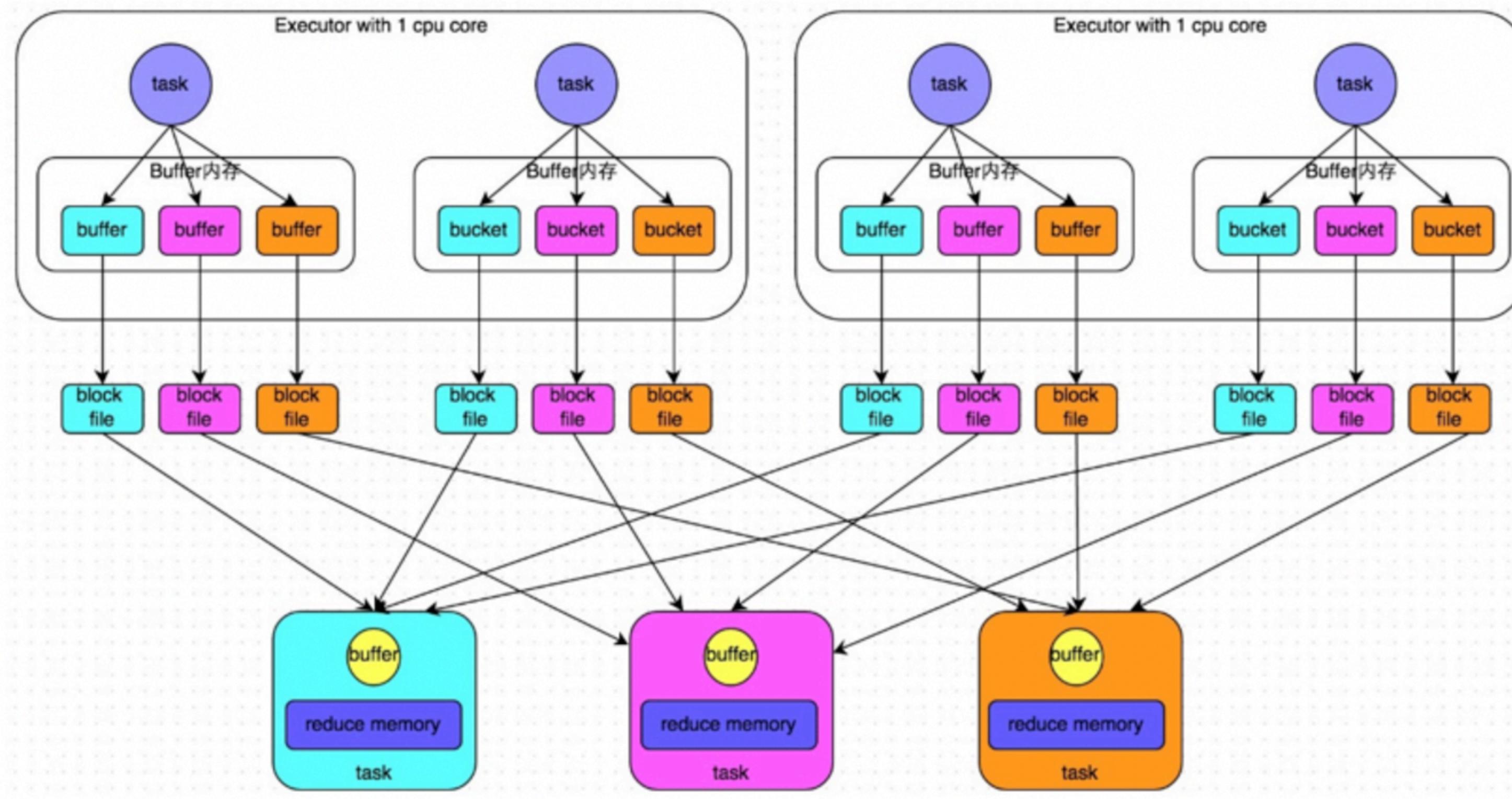


Hash Shuffle

- Data are hash partitioned on the map side
 - Hashing is much faster than sorting
- Files created to store the partitioned data portion
 - # of mappers X # of reducers
- Use consolidateFiles to reduce the # of files
 - From $M * R \Rightarrow E*C/T * R$
- Pros:
 - Fast
 - No memory overhead of sorting
- Cons:
 - Large amount of output files (when # partition is big)

未经优化的 HashShuffle

未经优化的 HashShuffle 的特点是：每一个 Map Task 都会生成 Reduce Task 数量那么多个文件，以 reduceByKey 举例，假如 Reduce 阶段一个 Key 占一个 Partition，也就是说一个 Key 对应一个 Reduce Task，那么 Map 阶段的每一个 Map Task 都要生成 Key 的数量个小文件。如下：



到了 reduce 阶段，因为 map task 给下游 stage 的每个 reduce task 都创建了一个磁盘文件，每个 reduce task 只要从上游每一个 map task 所在节点上，**拉取属于自己的那一个磁盘文件即可**。这里也是一边拉取一边聚合或者链接

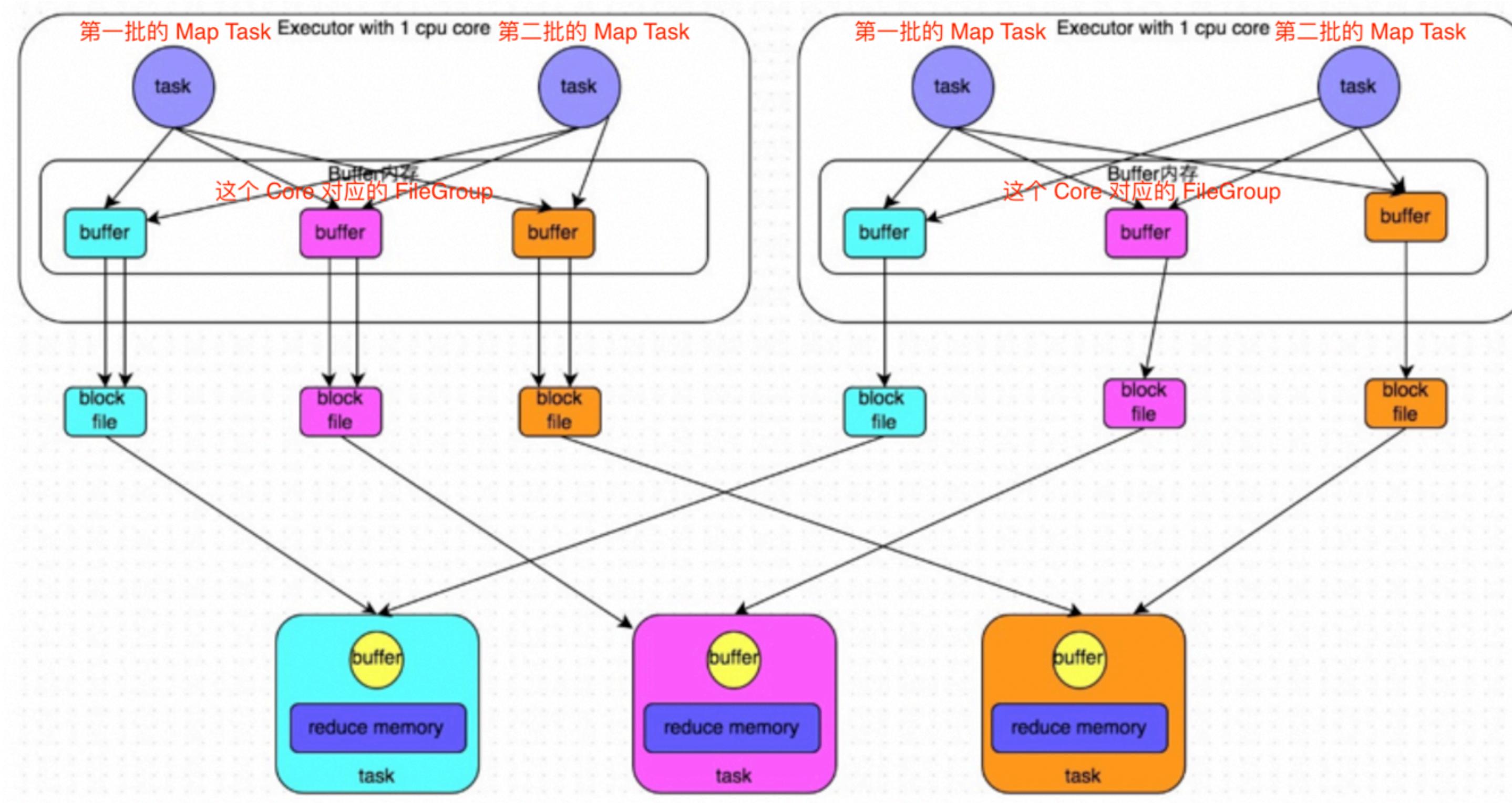
特点：会产生大量的小文件，现率低下

产生文件的数量 : $\text{numMapTask} * \text{numReduceTask}$

优化后的 HashShuffle

`spark.shuffle.consolidateFiles` 该参数默认值为 `false`

引入了，`shuffleFileGroup` 的概念，对于每一个 Executor，一个 Executor 上有多少个 CPU core，就可以并行执行多少个 Map task，每个 Core 同一时间只能运行一个 Map Task 第一批被运行的 MapTask 也会创建下游 Reduce Task 那么多个磁盘文件，并组成一个 shuffleFileGroup，也就是说：每一个 Core 对应一个 `shuffleFileGroup` 从第二批开始，每个 Map Task 都会复用第一批的 `shuffleFileGroup`，而不产生新的文件：



Hash Shuffle

- Data are hash partitioned on the map side
 - Hashing is much faster than sorting
- Files created to store the partitioned data portion
 - # of mappers X # of reducers
- Use consolidateFiles to reduce the # of files
 - From $M * R \Rightarrow E*C/T * R$
- Pros:
 - Fast
 - No memory overhead of sorting
- Cons:
 - Large amount of output files (when # partition is big)

Sort Shuffle

- For each mapper 2 files are created
 - Ordered (by key) data
 - Index of beginning and ending of each 'chunk'
- Merged on the fly while being read by reducers
- Default way
 - Fallback to hash shuffle if # partitions is small
- Pros
 - Smaller amount of files created
- Cons
 - Sorting is slower than hashing

Spark SortShuffle 解析

SortShuffleManager 的运行机制主要分成两种，一种是普通运行机制，另一种是 bypass 运行机制。当 shuffle read task 的数量小于等于 spark.shuffle.sort.bypassMergeThreshold 参数的值时（默认为 200），就会启用 bypass 机制。

SortShuffle 普通运行机制

SortShuffle 普通运行机制，跟 MR 的 Shuffle 过程基本一致，都是先写入缓存区，然后排序（快排），然后缓存快满了就溢血文件，最后对多个一些文件进行 Merge 成一个文件，为了为下游的 Reduce 阶段提供方便，还生成了一个 Index 文件，用来为下游 Reduce 拉取属于他们的那部分数据提供便利。

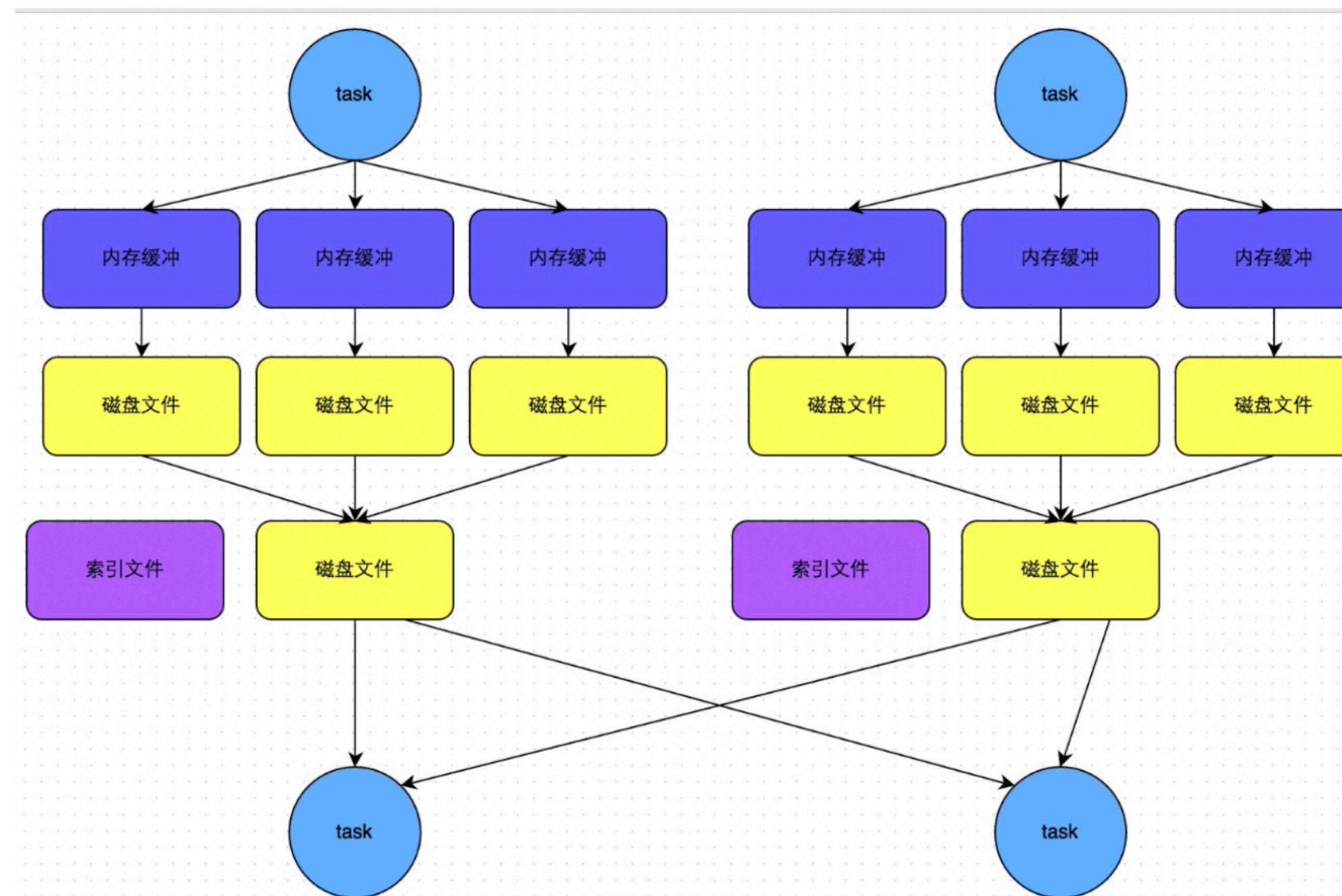
与 MR 的 Shuffle 的区别：

1. 缓存区的类型：MR 是写入到环形缓冲区，但是 Spark 是根据不同的 Shuffle 算子写入到不同的数据结构中，有可能是 HashMap 也有可能是 Array
2. 溢血的规则：MR 是环形缓冲区 80% 开始 Spill，Spark 是每一个 batch 条的数据量一次 spill（默认 batch 是 10000 条）

SortShuffle bypass 机制

与普通 SortShuffle 的区别：

1. 创建磁盘文件的机制不同，普通的是放入数据结构（缓存）每 batch 一次溢血，然后最终合并，**bypass** 是每个 Task 生成 Reduce Task 数量那么多个文件（跟未经优化的 HashShuffle 一样）然后最后也是合并称一个我磁盘文件和一个 Index 文件
2. 不排序 该机制的最大好处在于，shuffle write 过程中，不需要进行数据的排序操作，也就节省掉了这部分的性能开销。



中间产生文件的数量 : $\text{numMapTask} * \text{numReduceTask}$

最终文件数量: $\text{numReduceTask} * 2$ (Index file)



- LSH

了解这个算法之前需要掌握的知识：

1. 什么是 Distance (similarity)
2. 对于不同的数据类型，有哪些计算 Distance 的方式？
3. 什么是HashCode？有什么特点？

1. 什么是 Distance (similarity)

计算机如何判断这两个人是否相似？，以及有多相似？

A 身高：180 体重：75 月薪：15k 学历：硕士

B 身高：166 体重：49 月薪：15k 学历：博士

1. 将人的各个属性转化成一串可以计算的数字（向量）
2. 套用某个距离（相似度）计算

对于不同的数据类型，有哪些计算 Distance 的方式？

L_p norm distance 无论 x 是多少维度的数据

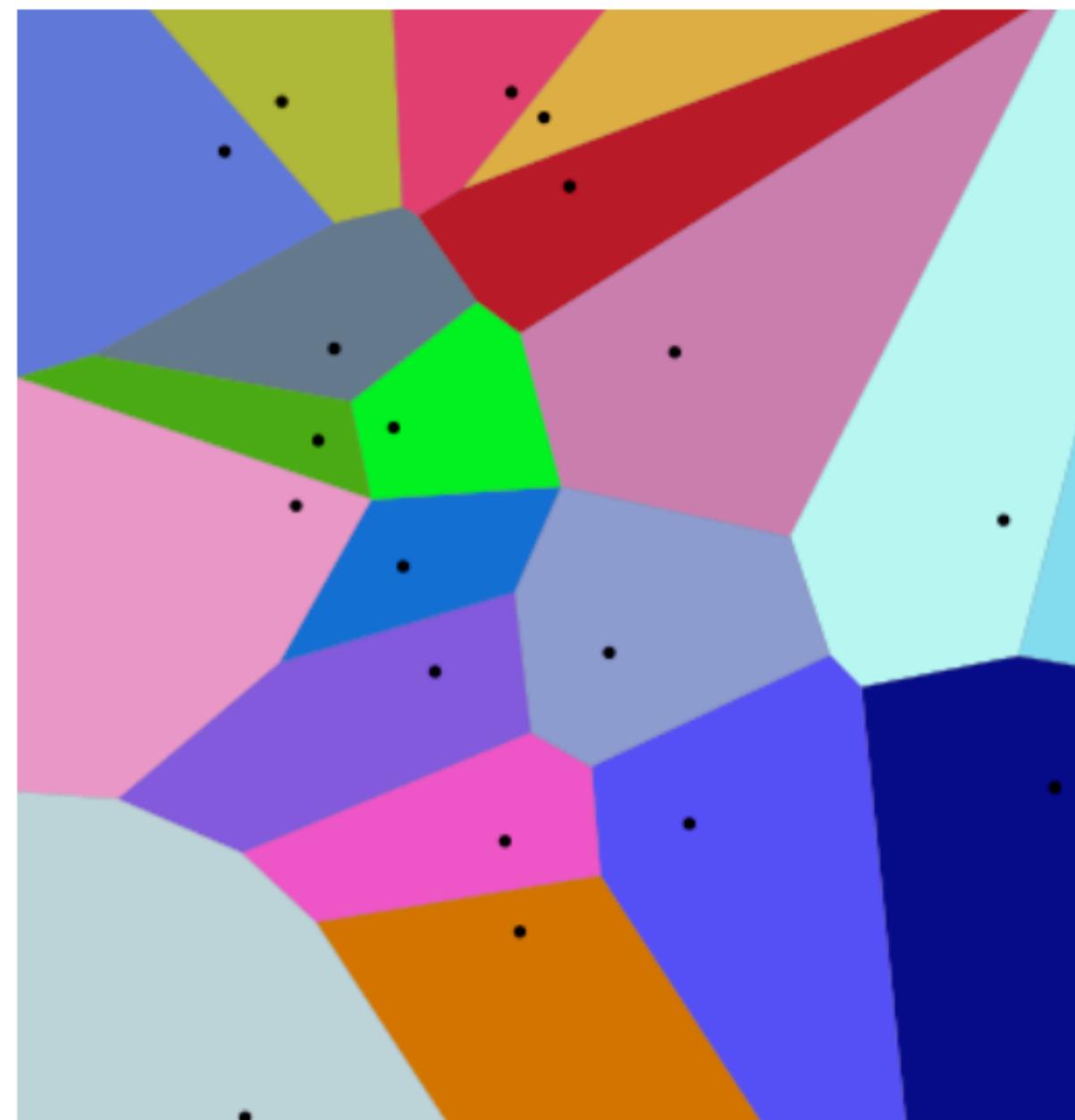
$$D(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

特别的当 P = 1 时： Manhattan Distance

特别的当 P = 2 时： Euclidean Distance

Similarity Search in Two Dimensional Space

- Why binary search no longer works?
 - No order!
- Voronoi diagram



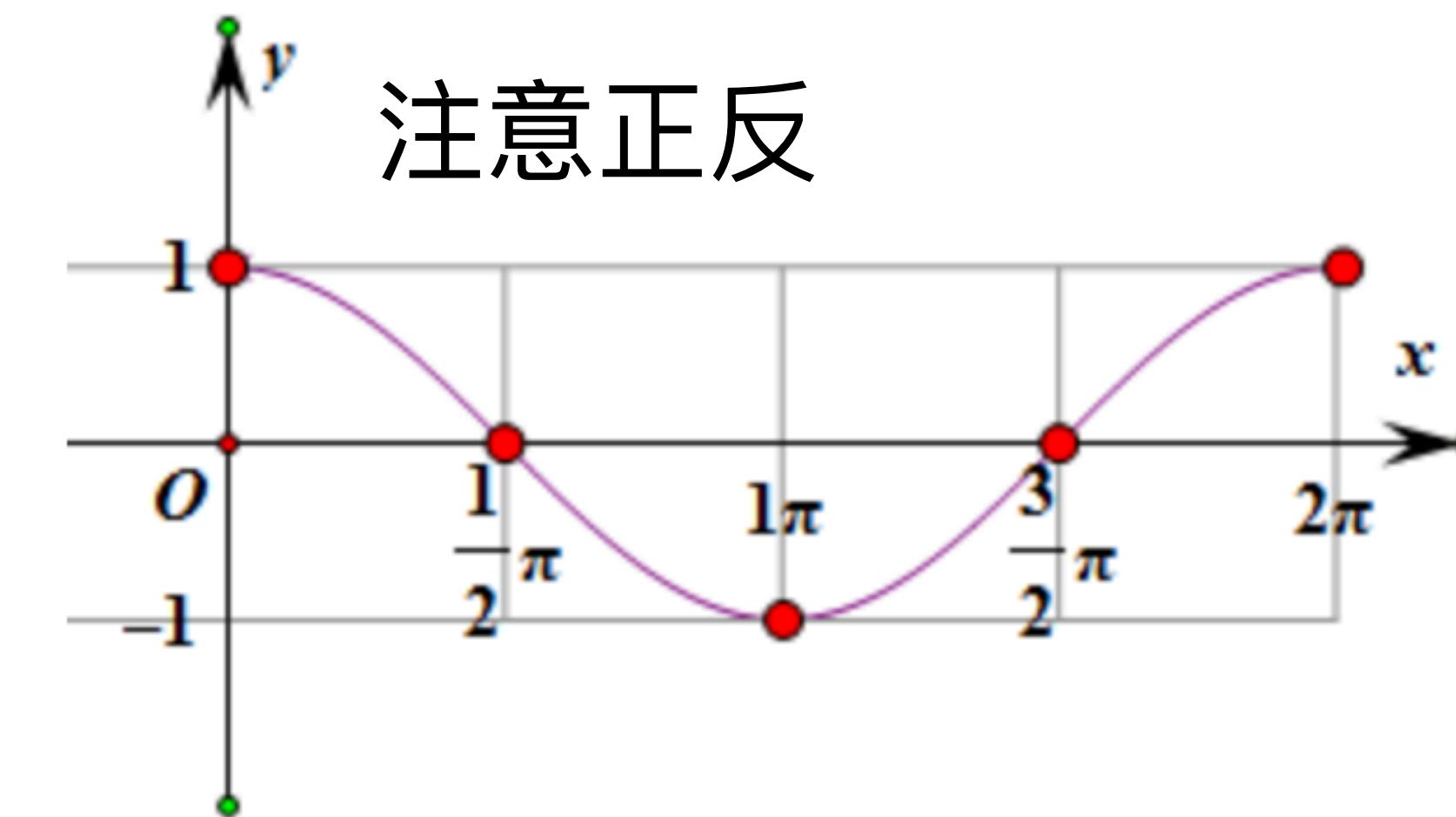
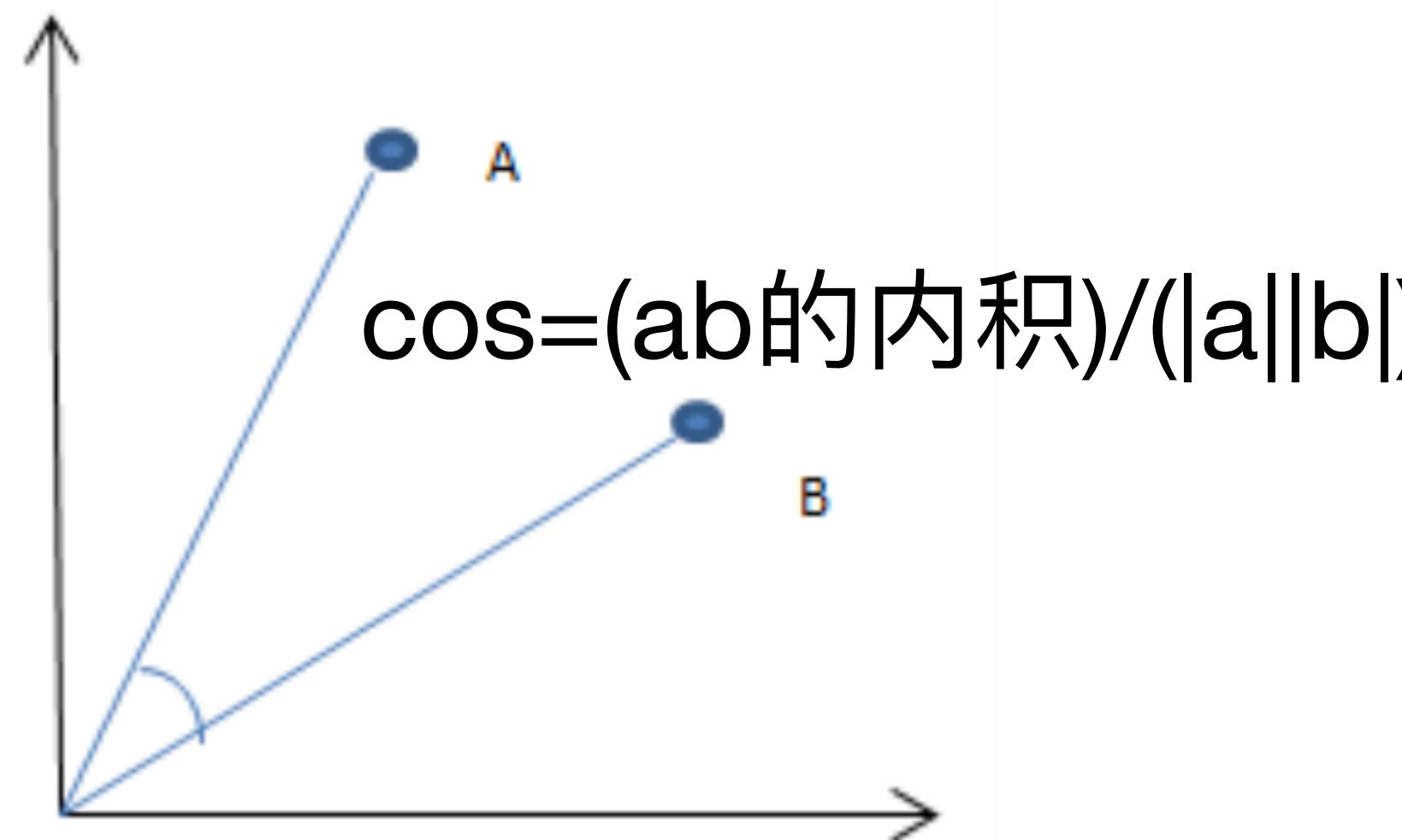
Euclidean distance



Manhattan distance

对于不同的数据类型，有哪些计算 Distance 的方式？

Angular Distance / Cosine Distance / Cosine Similarity



对于不同的数据类型，有哪些计算 Distance 的方式？

- Each data object is a set

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

什么是HashCode? 有什么特点?

A 身高: 180 体重 : 75

B 身高: 166 体重 : 49

我把每个人（每个数据）看成一个 Object

这个 Object 会有很多属性（身高，体重）

我们希望对每个 Object （基于他们自己的属性值） 对应唯一的一个数（或者是 Code）

(Java Demo)

HashCode 和 Similarity 的关系

HashCode 特点：

属性相同，HashCode 一定相同

属性不同，HashCode 不一定不同（我们希望它不同）

属性稍有不同，HashCode 可能会极大不同（CheckSun MD5）

Similarity 特点：

属性相同，两个 Object 相似度最高

属性稍有不同，两个 Object 相似度依然很高

当 Object 的维度越来越高... 传统的 similarity 计算方法失效

High Dimensional Similarity Search

- Applications and relationship to Big Data
 - Almost every object can be and has been represented by a high dimensional vector
 - Words, documents
 - Image, audio, video
 - ...
 - Similarity search is a fundamental process in information retrieval
 - E.g., Google search engine, face recognition system, ...
- High Dimension makes a huge difference!
 - Traditional solutions are no longer feasible
 - This lecture is about why and how
 - We focus on high dimensional vectors in Euclidean space

- Solution: Locality Sensitive Hashing (LSH)
- 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
 - Cost:
 - Space: $O(n)$
 - Time: $O(1) + O(|cand|)$

假设我们使用了某种 Hash Function 得到下面的 Table

HashCode	Person
101111	A, B
101110	C
101101	D, G, F
...	...

如果想找出跟 D 最相似的人：

1. 先计算 D 的HashCode 发现是 101101
2. 取出 101101 里面对应的所有人 (Candidate)
3. 使用传统 Distance 计算方法逐个计算，得到最相似的人

还有两个问题：

1. 如何设计 Hash Function?
2. 没法保证真正相似人的HashCode 100% 都是相同，也就是说没法保证取出来的 Object 都是该被取出来的

HashCode	Person
101111	A, B
101110	C
101101	D, G, F (c)
...	...

如何设计 Hash Function?

lecture 上老师介绍了很多种方法， Project1 我们使用这种

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

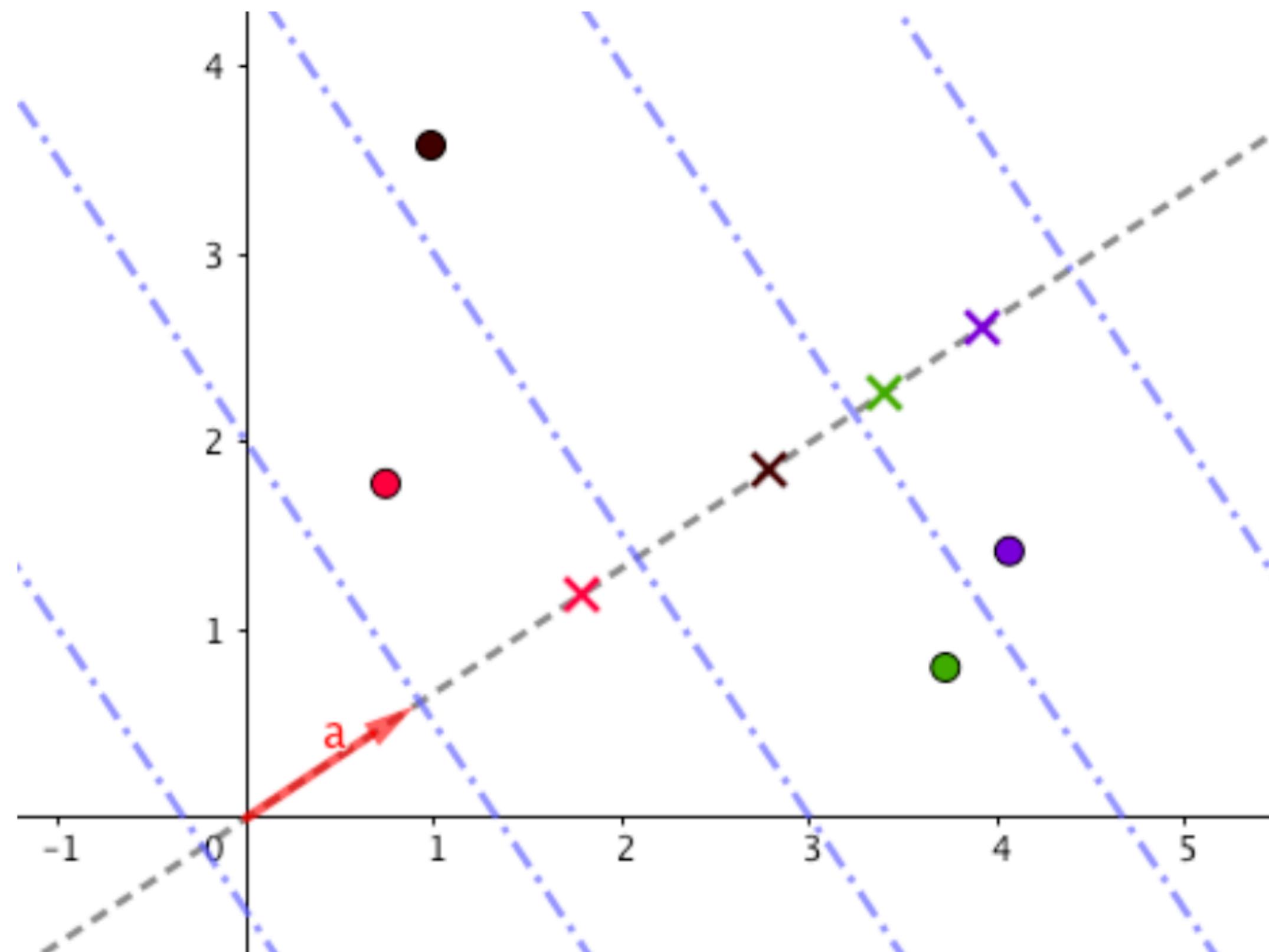
- $\Pr[h(o_1; a, b) = h(o_2; a, b)] = \int_0^w \frac{1}{\|o_1, o_2\|} f_p \left(\frac{t}{\|o_1, o_2\|} \right) \left(1 - \frac{t}{w} \right) dt$

- $f_p(\cdot)$ is the pdf of the absolute value of normal variable

每次对 Object hash 的结果是一个整数（不局限于 0,1）

在 LSH 中，一般会生成 m 个 $h(x; a, b)$

对 Object hash m 次，最后一个 Object 生成一个 m 位的 superHash



没法保证真正相似人的HashCode 100% 都是相同的？

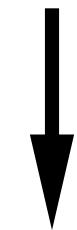
假设我们使用了某种 Hash Function 得到下面的 Table

HashCode	Person
101111	A, B
101110	C
101101	D, G, F
...	...

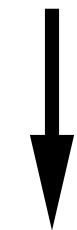
其实是找出HashCode 和 C (query) 相同的那些 Candidate
如果想找出跟 D 最相似的人：

1. 先计算 D 的HashCode 发现是 101101
2. 取出 101101 里面对应的所有人 (Candidate)
3. 使用传统 Distance 计算方法逐个计算，得到最相似的人

其实是找出 `HashCode` 和 `C` (`query`) 相同的那些 `Candidate`



这些 `Candidate` 不够? 或者不全?



放宽条件



如果某些 `Object` 的 `HashCode` 跟 `C` 的 `HashCode` 很相近 (+-1) 也可以



还不够或者不全?



继续放宽条件 (+-2)

问题：如何定义？

某些 Object 的HashCode 跟 C 的HashCode 很相近 (+-1)

Solution：C2LSH

注意：按照之前的算法假设HashCode 有 m 位，当且仅当 $\text{HashCode}(\text{query})$ 和 某个 HashCode

所有 m 位上的数字都一样的时候 (SuperHash) , 我们才把那组 Object 选为 Candidate
这一规则，在 C2LSH 里面有所变化

Counting the Collisions

- Collision: match on a single hash function
 - Use number of collisions to determine the candidates
 - Match one of the super hash with $q \rightarrow$ collides at least αm hash values with q

閾值

假设 $\alpha m = 8$

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

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

Collision

- 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

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

问题：什么时候 got enough Candidate

Solution : Candidate 的数是人为定的

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)  $\geq \alpha m$ :  
                cand  $\leftarrow$  cand  $\cup \{id\}$   
            if  $|cand| < \beta n$ :  
                offset  $\leftarrow$  offset + 1  
            else:  
                break  
    return cand
```

count(hashes_1, hashes_2, offset):

counter \leftarrow 0

for each $hash_1, hash_2$ **in** hashes_1, hashes_2:

if $|hash_1 - hash_2| \leq offset$:

counter \leftarrow counter + 1

return counter

关于 Project 1

18 Jul, 2020

- Late Penalty: 10% on day 1 and 30% on each subsequent day.

In this project, you may need to CREATE YOUR OWN TEST CASES

Task 2: Report (10 points)

You are also required to submit your project report, named: `report.pdf`. Specifically, in the `rep` questions:

1. Implementation details of your `c2lsh()`. Explain how your major transform function works.
2. Show the evaluation result of your implementation using **your own test cases**.
3. What did you do to improve the efficiency of your implementation?

- Must use PySpark, some python modules and PySpark functions are banned.
- E.g., numpy, pandas, collect(), take(), ... • Use transformations!

implementation, hence will be assigned **ZERO** score. Specifically, you are not allowed to use the following PySpark functions:

- aggregate , treeAggregate , aggregateByKey
- collect , collectAsMap
- countByKey , countByValue
- foreach
- reduce , treeReduce
- saveAs* (e.g. saveAsTextFile)
- take* (e.g. take , takeOrdered)
- top
- fold

You don't need to implement hash functions and generate hashed data, we will provide the data hashes for you.

Notice: The order of the elements in the list does not matter (e.g., we will collect the elements and evaluate them as a set).