HADOOP RELIABILITY

BIG DATA PROCESSING

Félix Cuadrado

felix.cuadrado@qmul.ac.uk

Queen Mary University of London

School of Electronic Engineering and Computer Science

Queen Mary
University of London

Contents

- **Distributed systems reliability**
- Hadoop Reliability
- Data joins

High availability

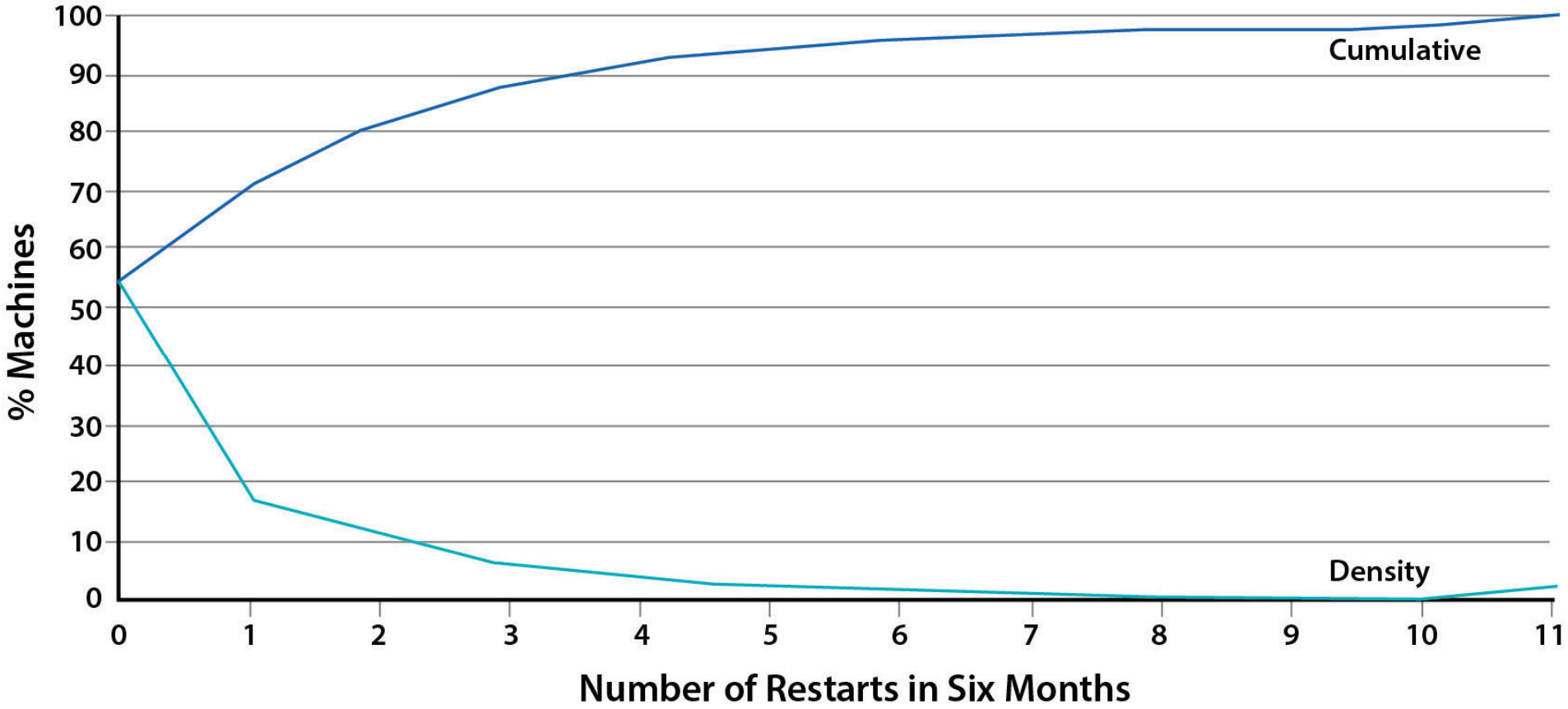
- **Availability** - the percentage of total time that a System is available for use
- **High availability** (HA) is a characteristic of a system, which aims to ensure an agreed level of operational performance for a higher than normal period.
  - Fault tolerance: Property of a system that continues to operate correctly on the event of a failure
  - HA implies there are no single points of failure
  - Graceful degradation: when some components fail, the system temporarily works with worse performance (but still works)

Queen Mary University of London

High Availability measurement: counting nines

Percentage Uptime	Percentage Downtime	Downtime per year	Downtime per week
98%	2%	7.3 days	3h22m
99%	1%	3.65 days	1h41m
99.8%	0.2%	17h30m	20m10s
99.9%	0.1%	8h45m	10m5s
99.99%	0.01%	52.5m	1m
99.999%	0.001%	5.25m	6s
99.9999%	0.00001%	31.5s	0.6s

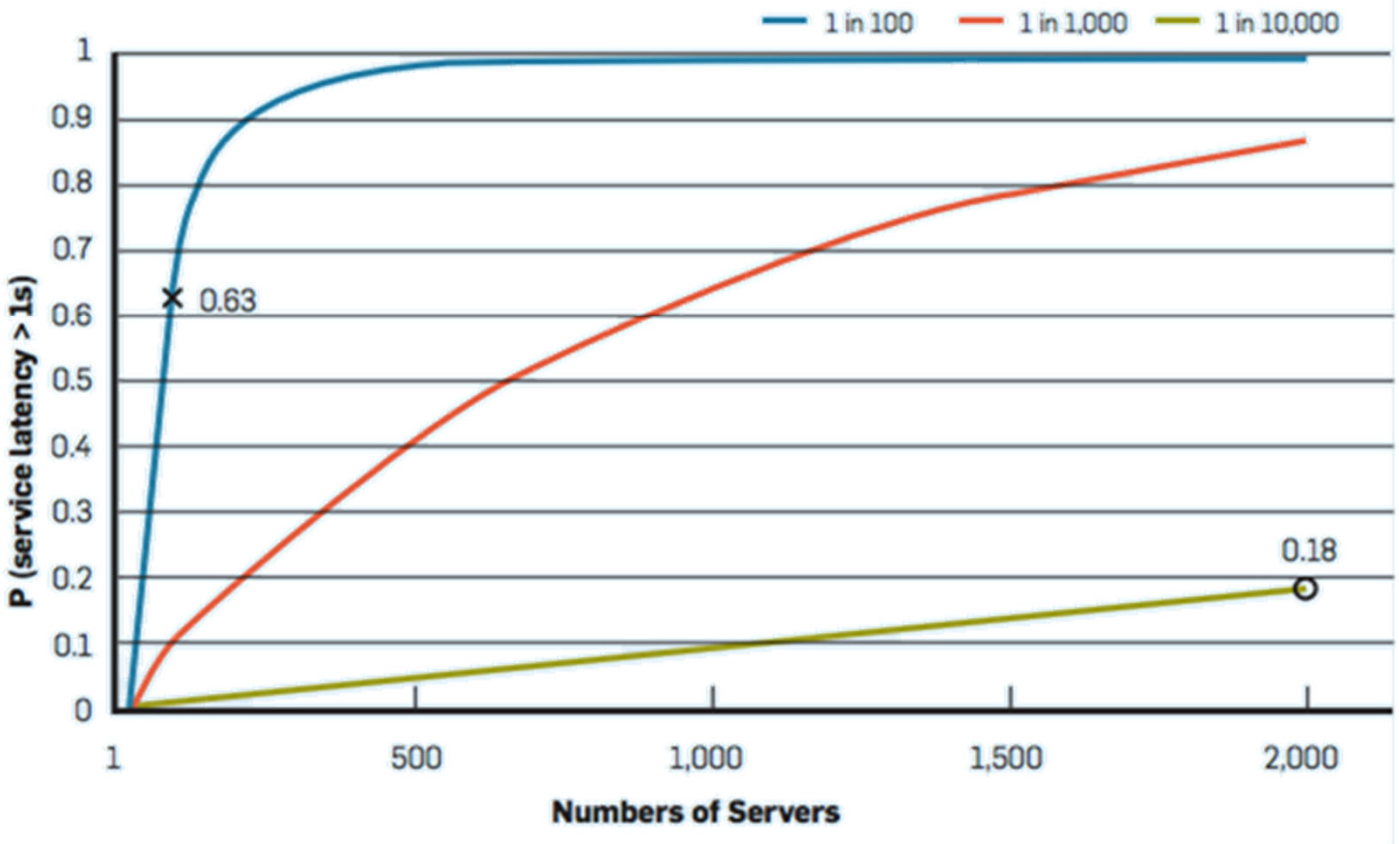
Number of machine restarts in a Google DC



From Barroso et al. Datacenter as a Computer, 2nd Ed, Morgan & Claypool, 2013

Google's Tail of Latency

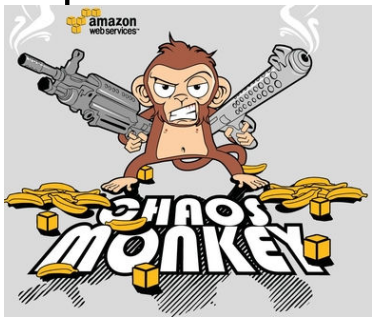
Probability of one-second service-level response time as the system scales and frequency of server-level high-latency outliers varies.



J.Dean, L.A. Barroso, "The Tail at Scale". Communications of the ACM, vol. 56 (2013)

Testing reliability: Netflix Chaos Monkey

- Every weekday between 9am and 5pm, an army of malicious programs, affectionately known as “chaos monkeys,” are unleashed upon Netflix’s information infrastructure.
- “Their sole purpose is to make sure that we’re failing in a consistent and frequent enough way to make sure that we don’t drift into overall failure,” Tseitlin said. The goal is to fail often and uncover potential problems before they become actual problems.
- “The design premise there is that all of the architecture is resilient enough to retry and to begin re-serving the experience in a way that is completely transparent to the customer”



- http://luckyrobot.com/netflix-chaos-monkey-keeps-movies-streaming/

Queen Mary University of London

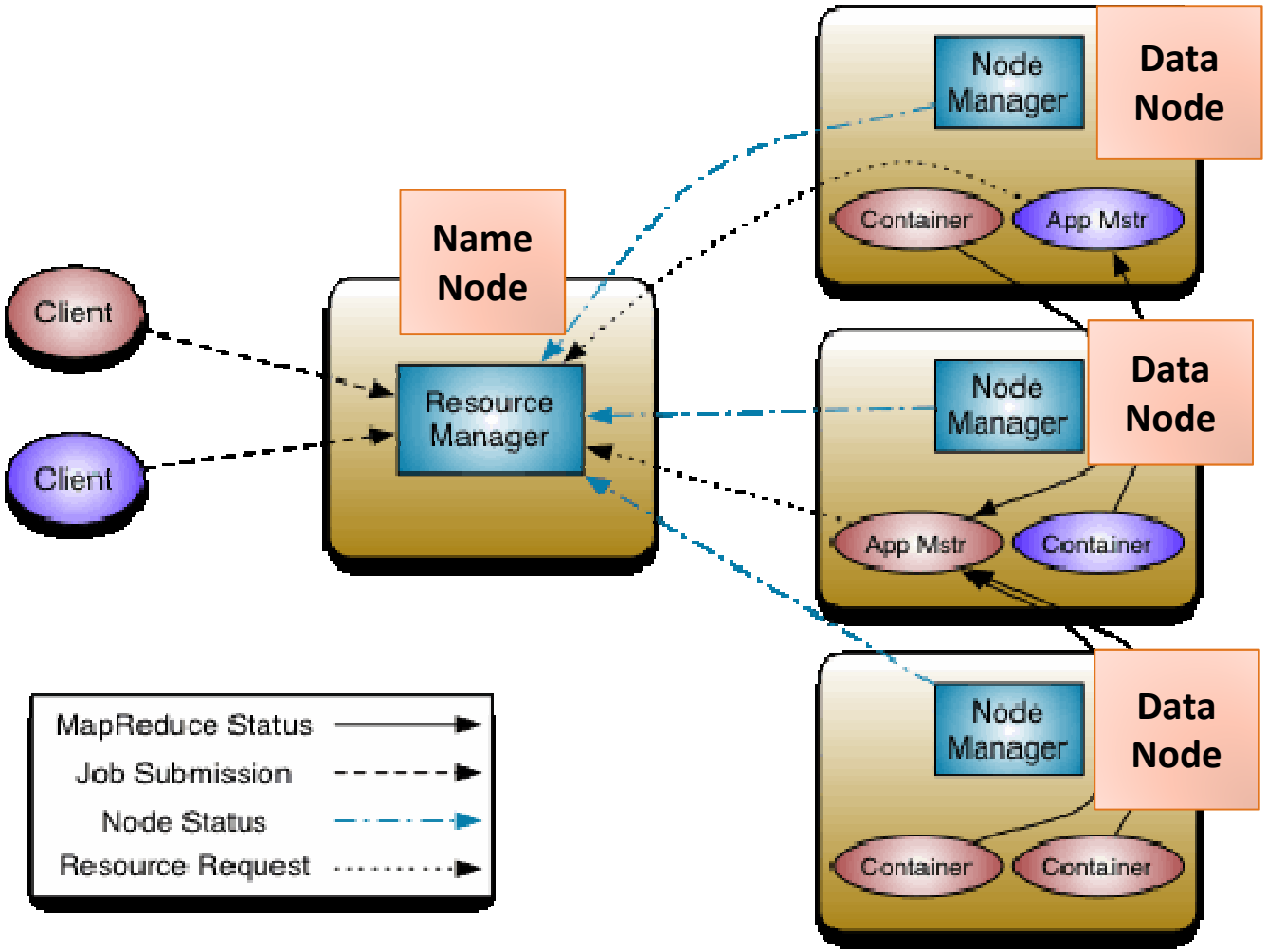
Contents

- Distributed systems reliability
- **Hadoop Reliability**
- Data joins

Error management in Hadoop

- Goal: detect errors and gracefully recover from them while not interrupting job execution (if possible)
- **Any** Hadoop element can fail during a job
  - Data integrity error
  - Task (Map/Reduce) failure
  - NodeManager failure
  - ApplicationMaster failure
  - ResourceManager/ NodeManager failure

Any element can fail



Data integrity errors

- DataNodes verify checksums before writing
- HDFS clients verify checksum of read blocks
- Errors are reported to the NameNode
  1. The block is marked as corrupt
     - No more clients will read that copy of the block
  2. The client is redirected to another copy
  3. A block replica is scheduled to be copied in another node

Task (Map or Reduce) failure

- Caused by software failures
  - e.g., a task requires Java n; node has Java n-1.
  - e.g. the code throws a Java Exception
- Error reported back to NodeManager, task marked as failed
- Hanging tasks are detected by NodeManager and also marked as failed
- The ApplicationMaster/ResourceManager tries to reschedule the task on a different node
- There is a maximum number of retries (4 by default) before declaring job failure

NodeManager failure

- The NodeManager monitors the health of the hardware resources of the node and reports any problem to the ResourceManager. The RM also detects NM failure by no longer receiving heartbeats from it
- The RM marks all hosted Containers as killed, and reports the failure to the ApplicationMaster
- The MapReduceAM will rerun all the hosted Containers in other nodes from the cluster after negotiating with RM
- Completed Map tasks are also rescheduled to other nodes
  - Map results are not stored in the HDFS

ApplicationMaster failure

- The ResourceManager detects the failure and starts a new ApplicationMaster instance in a different container (managed by a ResourceManager).
- The ResourceManager kills all the containers of that Application.
- For MapReduce AMs, it will use the job history to recover the state of the tasks that were already run by the (failed) application so they don't have to be rerun .
- There is a maximum number of attempts (default 2)

ResourceManager/ NameNode failure

- The ResourceManager and the NameNode can be **single points of failure** for Hadoop
  - Loss of data/progress
  - Cluster stops working
- To avoid data loss The Secondary NameNode communicates periodically with NameNode and stores backup copy of index table.
- The ResourceManager can also store the list of ApplicationMasters and the progress of each of them, so it can restart/resume scheduled jobs when restarted.

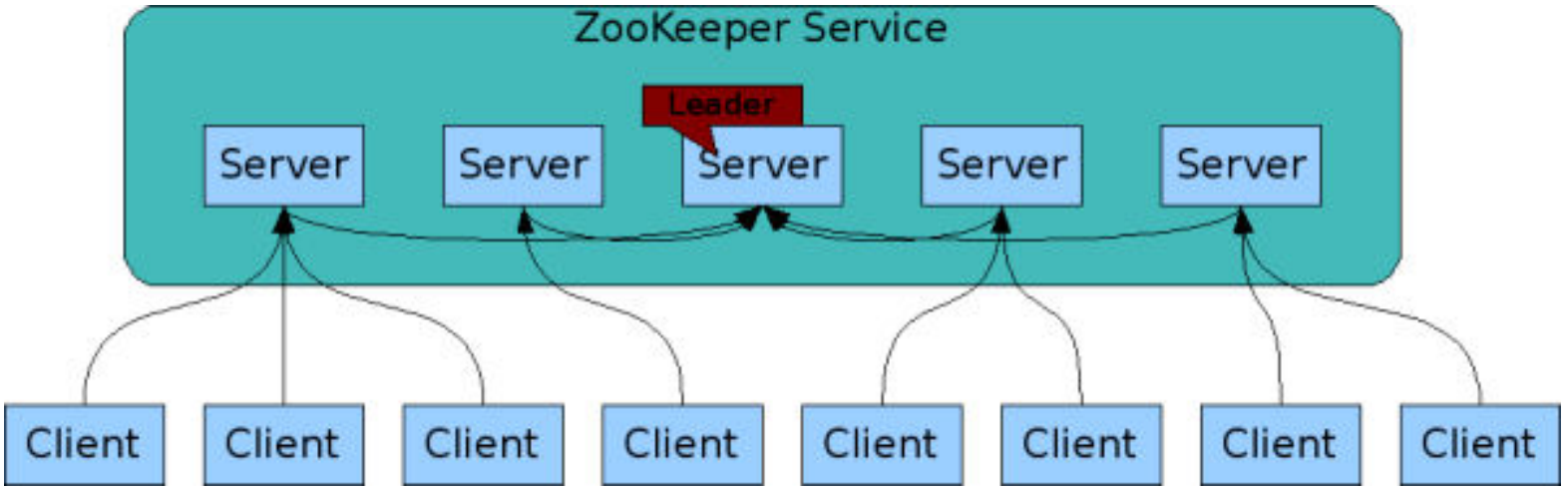
Hadoop 2.0 -> NameNode HA

- Alternative to default setup
- Run 2 redundant **NameNodes** in different machines of the cluster: **Active** and ***Standby***
- New daemon: **JournalNodes** machines (3 default)
  - **Active** NameNode **writes** all changes to ALL journals.
  - Changes must be accepted by majority of journals
  - **Read** by ***Standby*** NameNode to catch up to state
- No **SecondaryNameNode** is needed

https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html

Apache ZooKeeper

- Distributed, open-source coordination service for distributed applications.
- Quorum algorithms for selecting leaders, agreeing on shared state

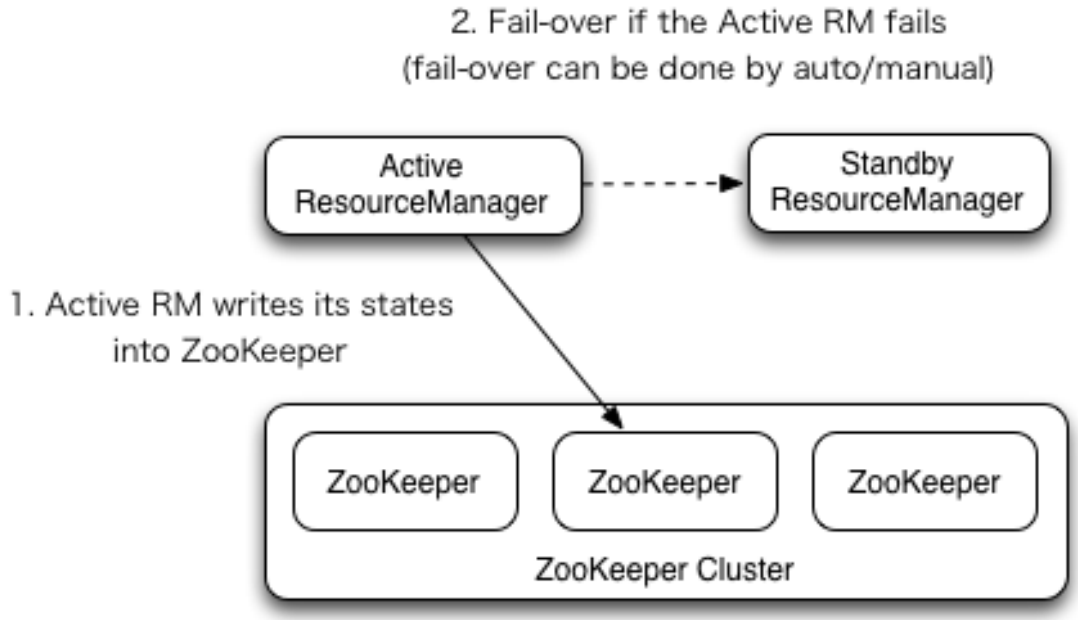


NameNode HA with Automated failover

- ZKFailoverController (ZKFC) is a ZooKeeper client which implements automated failover by:
- **Failure detection** - each NameNode maintains a persistent session in ZooKeeper. If the machine crashes, the ZooKeeper session will expire, notifying the other NameNodes that a failover should be triggered.
- **Active NameNode election** - If the **active** NameNode crashes, another node may take a special exclusive lock in ZooKeeper indicating that it should become the next active.

ResourceManager HA

- Analogous to NameNode HA scheme
  - **Active** // **standby** mode



Queen Mary
University of London

Contents

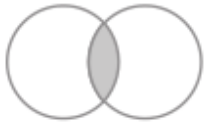
- Distributed systems reliability
- Hadoop Reliability
- **Data joins**

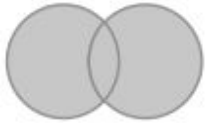
Join Definition

- Operation to combine together related data
- (e.g. find what Amazon users do from the “user database” AND the logs of the web servers.
- What else would you use joins for in this context?
  - Relate purchase habits to demographics
  - Send reminders to inactive users
  - Recommendation systems

Types of Joins

- Inner: compare all tuples in relations L and R, and produce a result if a join predicate is satisfied.



- Outer: don't require both tuples to match based on a join predicate, and instead can retain a record from L or R even if no match exists.
  - Left outer 
  - Full outer 

Relational Database Joins

user

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

course

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

```
SELECT user.name, course.name FROM `user`
        JOIN `course` on user.course = course.id;
```

Relational Database Joins

user

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

course

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

```
SELECT user.name, course.name FROM `user`
   INNER JOIN `course` on user.course = course.id;
```

Alice	HTML5
Bob	CSS3
Carol	CSS3
David	MySQL

Relational Database Joins

user

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

course

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

```
SELECT user.name, course.name FROM `user`
  RIGHT JOIN `course` on user.course = course.id;
```

Alice	HTML5	NULL	Java
Bob	CSS3	NULL	Python
Carol	CSS3	David	MySQL

Joining datasets in Hadoop

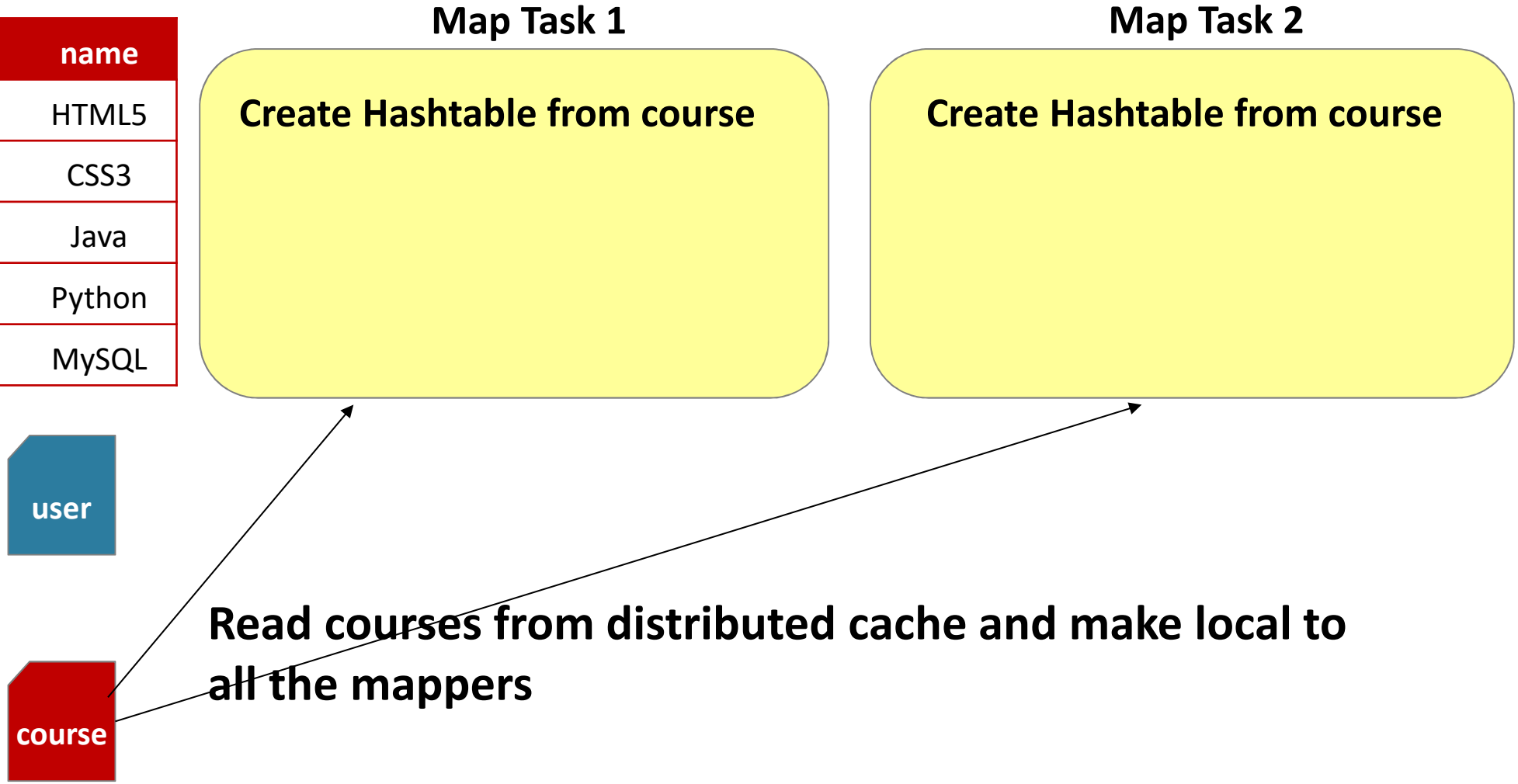
1. **Replication join**—An outer **map-side** join where one of the datasets is *small enough to be kept in memory.*
2. **Repartition join**—A **reduce-side** join for joining two or more *large datasets*.
3. **Semi-join**—A map-side join where one out of several large datasets is filtered so that it fits in memory. (not covered in ECS640/765)

Replication Joins

- Idea: Replicate smallest dataset to all the map hosts using Hadoop's distributed cache.
- Map:
  1. Use the initialization method of each map task to load the small dataset into a hashtable
  2. Use the key from each record of the large dataset to look up the small dataset hashtable
  3. Join between the large dataset record and all of the records from the small dataset that match the join value.
- No Reducer is needed

Replication Joins (I)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL



Replication Joins (II)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

course

Map Task 1

Create Hashtable from course

Get User data

1	Alice	1

course

Map Task 2

Create Hashtable from course

Get User data

2	Bob	2

user

id	name	course
1	Alice	1
3	Carol	2
5	Emma	NULL

id	name	course
2	Bob	2
4	David	5
5	Emma	NULL

course

Replication Joins (III)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

course

Map Task 1

Create Hashtable from course
Get User data
Lookup in Hashtable

1	Alice	1

course

Map Task 2

Create Hashtable from course
Get User data
Lookup in Hashtable

2	Bob	2

user

id	name	course
3	Carol	2
5	Emma	NULL

id	name	course
4	David	5
5	Emma	NULL

course

Replication Joins (IV)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

user

course

Map Task 1

course

Create Hashtable from course
Get User data
Lookup in Hashtable
Join and Emit

Alice	HTML5

id	name	course
3	Carol	2
5	Emma	NULL

Map Task 2

course

Create Hashtable from course
Get User data
Lookup in Hashtable
Join and Emit

Bob	CSS3

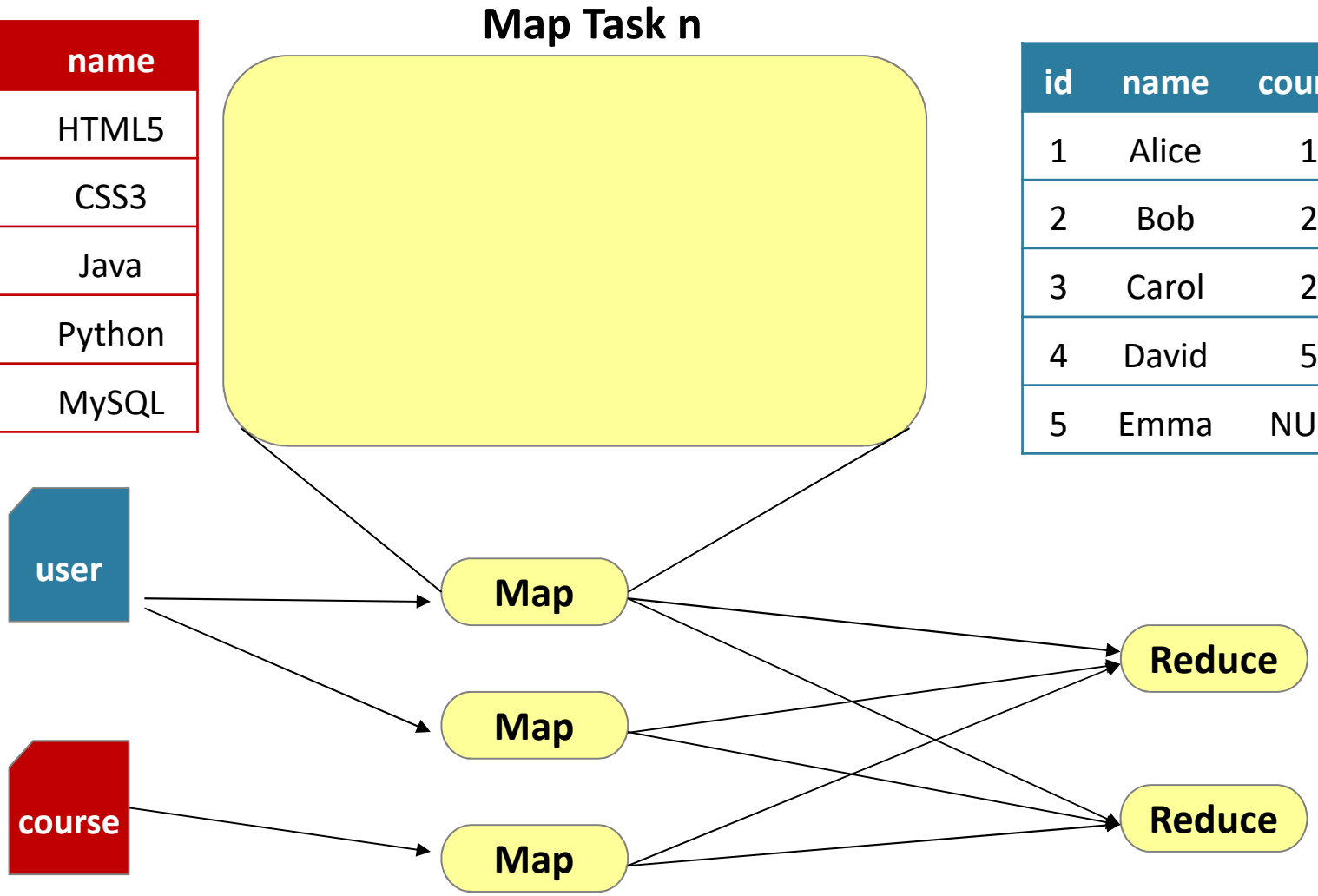
id	name	course
4	David	5
5	Emma	NULL

Repartition join

- Idea: Process both datasets in Mappers, emit the join key as the Map out key
- Perform the join at the reducer among all the elements
- A job can have more than one input path
  - Define a different Mapper for each input path
  - Emitted key value Writables must be of the same type

Repartition Joins (I)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

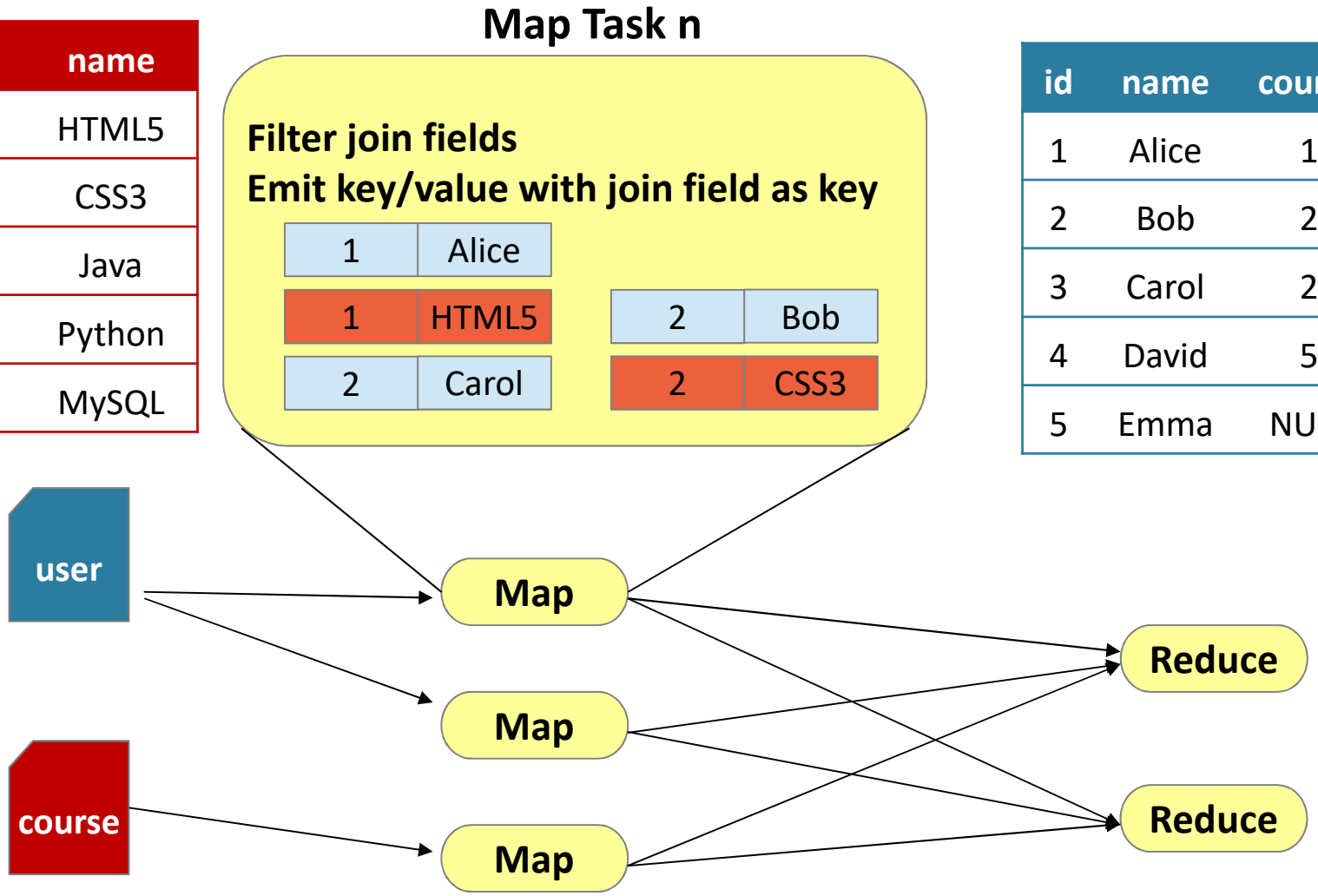


id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

Based on Fig 4.2 From "Hadoop in Practice" by Alex Holmes

Repartition Joins (II)

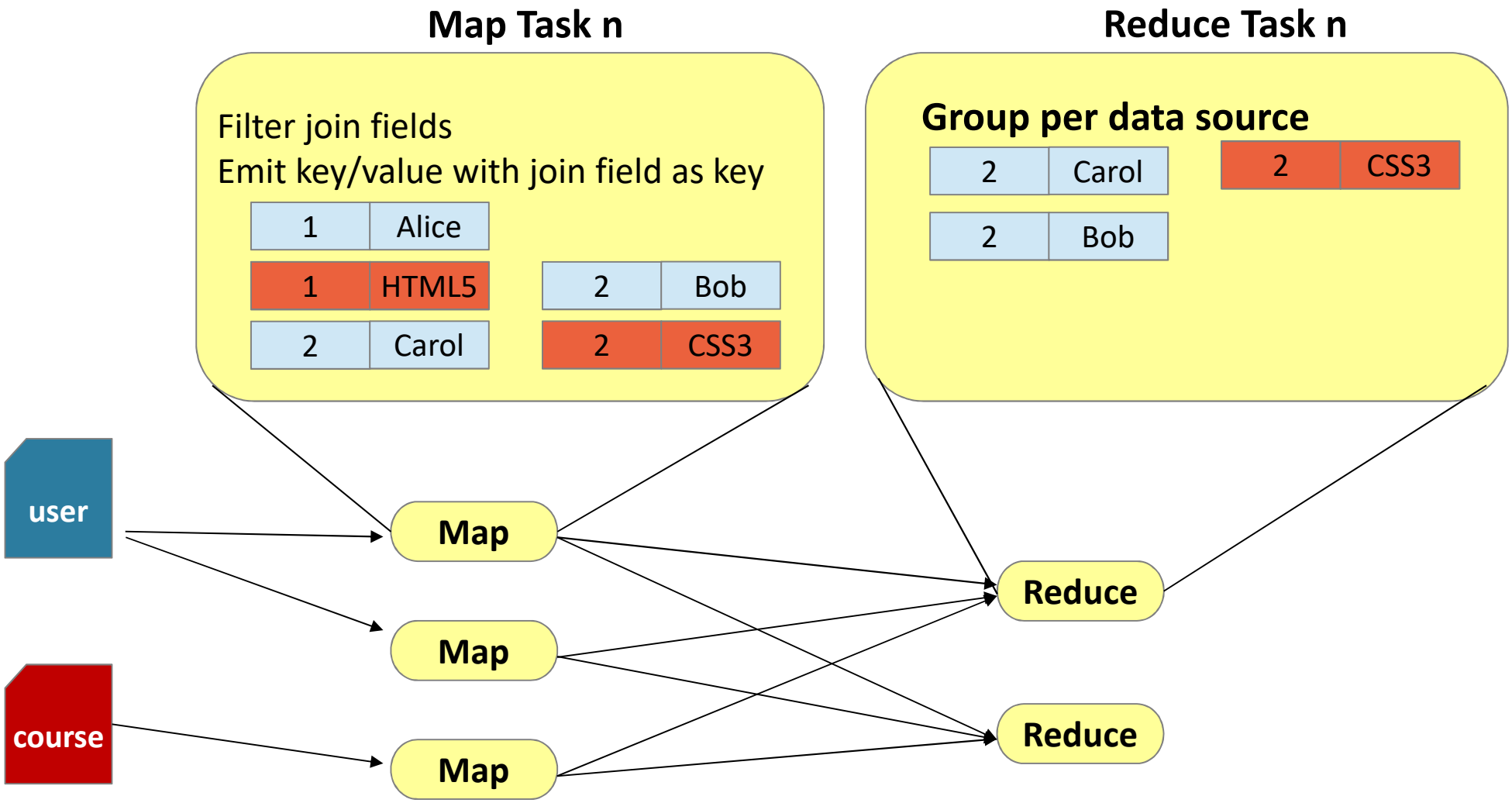
id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL



id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

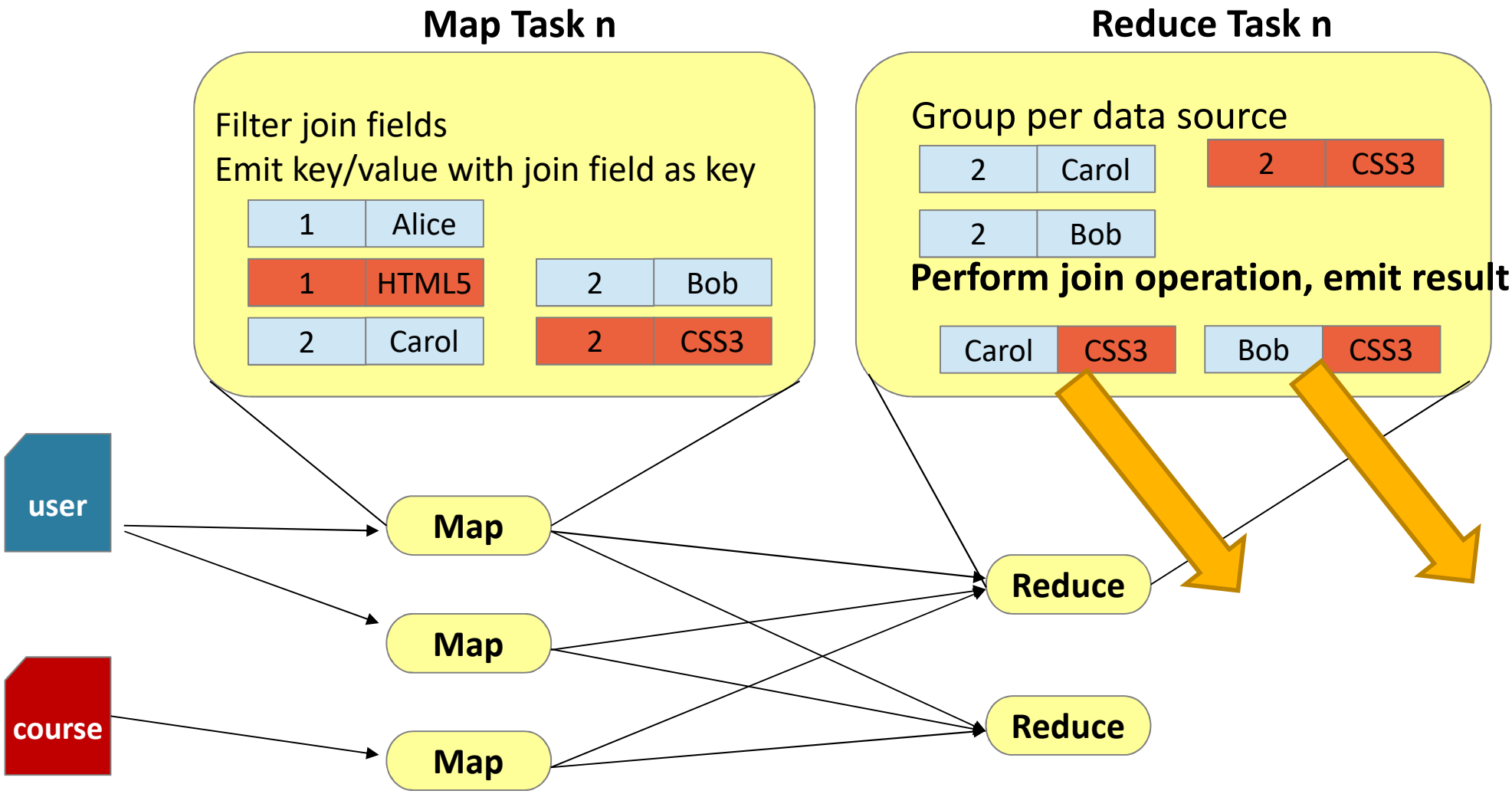
Based on Fig 4.2 From "Hadoop in Practice" by Alex Holmes

Repartition Joins (III)



Based on Fig 4.2 From "Hadoop in Practice" by Alex Holmes

Repartition Joins (IV)



Based on Fig 4.2 From "Hadoop in Practice" by Alex Holmes