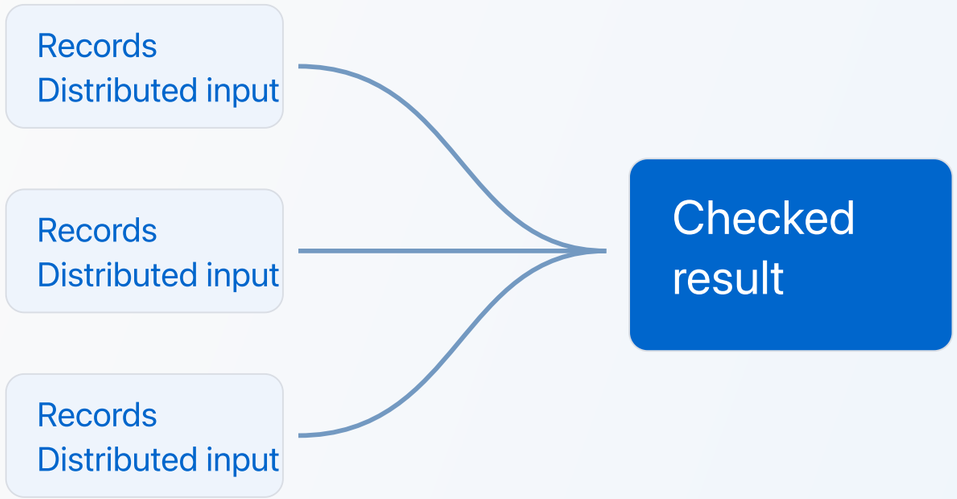
Big Data

Revision · Topics 1–5

Lectures and tutorials



Topics and tutorials

Follow one progression: distribute and store records, express and execute queries, improve the pipeline, build a model, and analyse stored data with SQL.

Topic	Core knowledge	Tutorial connection
1 · Hadoop	HDFS, YARN and MapReduce	1 · Word count on EMR
2 · Spark	Data models and distributed execution	2 · Interactive word count
3 · Spark in depth	Correct pipelines, movement and storage	3 · Adult data analysis
4 · Spark ML	Features, fitting and honest evaluation	4 · Customer churn
5 · Hive	Metadata, SQL and storage layout	5 · MovieLens ratings

Data to a checked result

A useful result depends on every preceding step: the files must be right, the schema must interpret them correctly, and the computation must answer the intended question.

Acquire + store

Keep original records

Record source + scope

Prepare + compute

Validate types + rows

Join and aggregate

Analyse or predict

SQL result or ML output

Check against a baseline

Verify + preserve

Read back saved output

Keep evidence off cluster

Hadoop and distributed data

Before a distributed program can calculate anything, its records need storage and its tasks need resources.

Files

Split into blocks

HDFS stores replicas

Resources

YARN allocates capacity

Workers launch containers

Processing

Map emits keyed values

Reduce combines groups

The five Vs

Big Data is relative to the task and available capacity. Diagnose the actual pressure before choosing a larger cluster or a new tool.

Pressure	Meaning	Consequence
Volume	Amount of data	Storage and scan cost
Velocity	Arrival and processing rate	Backlog grows if arrivals exceed capacity
Variety	Different structures and formats	Parsing and integration work
Veracity	Trustworthiness of records	Check missing, duplicate and invalid values
Value	Useful outcome for a question	A correct calculation still needs a purpose

Scale up and scale out

Scaling up adds capacity to one machine; scaling out divides work among machines. Distributed work also needs coordination, data movement and recovery.

Choice	Potential gain	Limit or cost
Scale up	More CPU, memory or storage per machine	Hardware limits; one machine remains a failure point
Scale out	More workers can process independent data	Network, partitioning and coordination costs
Example	Two workers scan separate log partitions	A grouped total still has to combine their results
Decision	Measure the limiting resource first	More nodes do not repair duplicate or invalid rows

Hadoop component roles

Hadoop Distributed File System (HDFS) stores bytes, YARN allocates resources, and MapReduce processes records. These responsibilities cooperate within one job.

HDFS · storage

NameNode: names + locations

DataNodes: replicated bytes

YARN · resources

ResourceManager: allocation

NodeManagers: containers

MapReduce · work

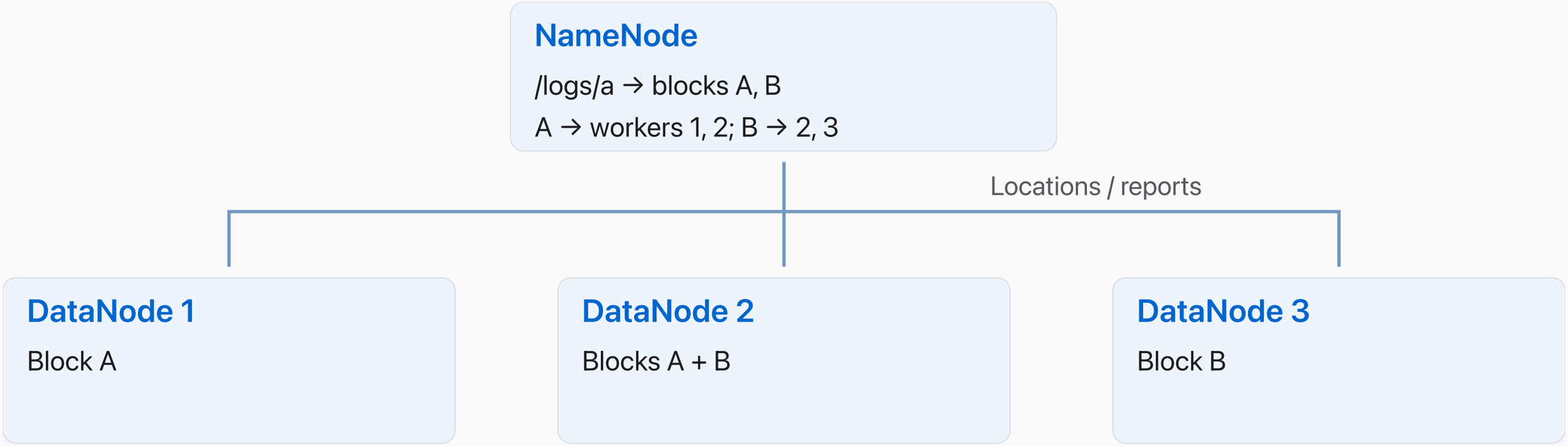
Map: emit key–value pairs

Reduce: combine grouped values

Storage, resource allocation and computation have separate responsibilities.

HDFS blocks and metadata

HDFS splits files into blocks and stores replicated block bytes on DataNodes. The NameNode maintains the file namespace and block-location metadata.



HDFS read and write paths

For a read, the client asks the NameNode for block locations and then reads bytes from DataNodes. A write uses selected DataNodes in a replication pipeline.

Operation	Metadata path	Data path
Read	Client → NameNode: find block locations	Client ← DataNode: read block bytes
Write	Client asks NameNode to create the file	Client → DataNode → replica DataNode
Responsibility	NameNode tracks names and locations	DataNodes hold the actual block bytes
Workload fit	Large sequential scans and writes	Many tiny files add metadata overhead

Replication and recovery

Replicas let a client read another copy after a DataNode fails. The system can replace missing replicas while surviving copies remain available.

Before failure

Block A on workers 1 + 2

Metadata records both

Worker 1 fails

Its copy unavailable

Other files may be affected

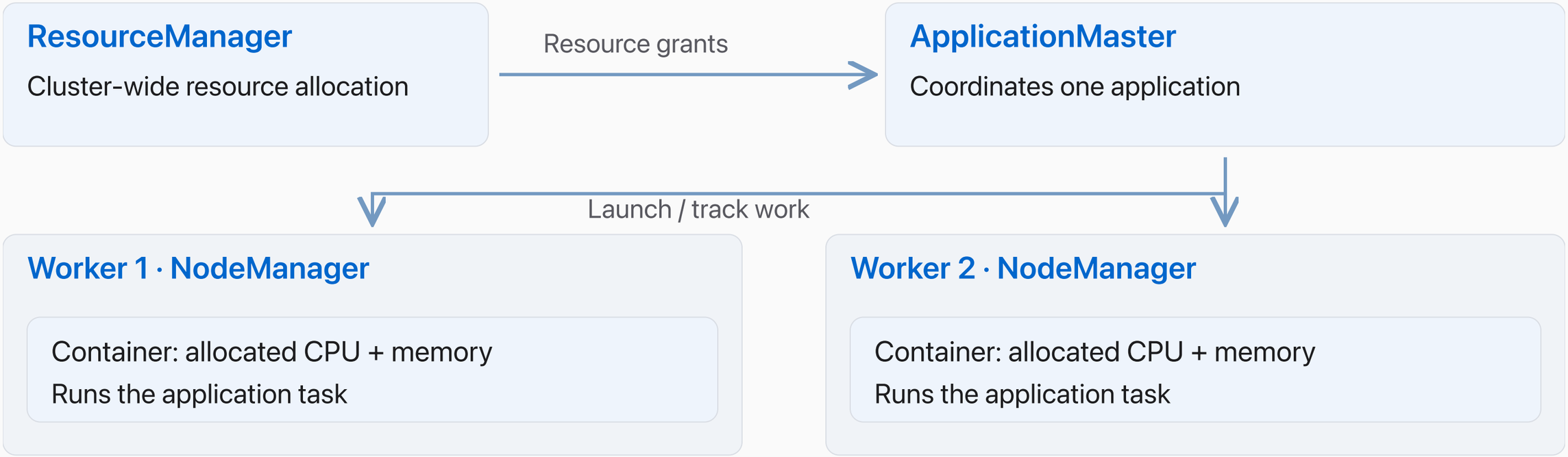
Read surviving copy

Client reads worker 2

Restore desired replication

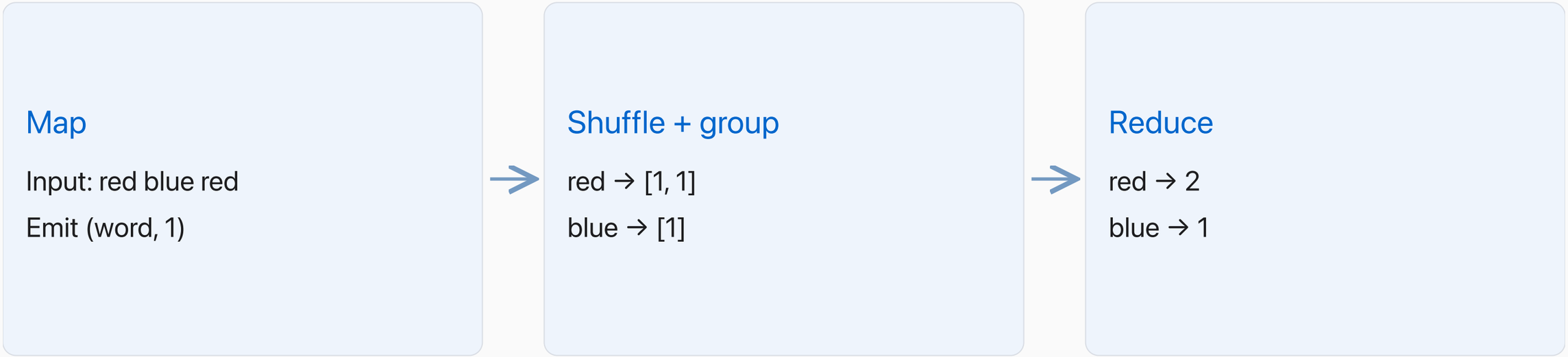
YARN resources and tasks

YARN means Yet Another Resource Negotiator. Its ResourceManager handles cluster resource allocation; each application has an ApplicationMaster, and NodeManagers manage containers on workers.



MapReduce

A mapper emits key–value pairs. The shuffle routes and groups matching keys; reducers combine their values into output records. A key identifies the group, while a value supplies the data to combine.



Keys, reducers and output records

A reducer task can process several distinct keys. Partitioning determines which reducer receives a key; it does not create one reducer per word.

Quantity	Illustrative case	Interpretation
Mapper output	(a,1), (b,1), (a,1), (c,1), (b,1), (a,1)	Six emitted records
Distinct keys	a, b, c	Three groups
Reducer task 0	a → [1,1,1]; c → [1]	Two groups in one task
Reducer task 1	b → [1,1]	One group in another task
Final output	(a,3), (b,2), (c,1)	Three output records; two reducers

Combiner correctness

A combiner optionally aggregates mapper output locally before the shuffle. The final answer must remain correct if combining happens zero, one or several times.

Operation	Safe intermediate state	Why
Sum	Partial sums	Sums can be merged in any grouping
Count	Partial counts	Adding counts preserves total records
Mean	(sum, count), then final sum / count	Unweighted means lose group sizes
Counterexample	[10,20] and [90]	Mean of means = 52.5; true mean = 40
Correct merge	(30,2) + (90,1) → (120,3)	120 / 3 = 40

Blocks and input splits

A block is a storage unit; an input split describes logical work for a mapper. Their boundaries often relate, but the input format determines how records become mapper input.

Term	Role	Avoid confusing it with
HDFS block	Replicated bytes in storage	A record or a reducer group
Input split	Logical portion assigned to a mapper	A permanently stored new file
Record	Input key and value read by a task	An entire storage block
Map task	Processes an input split	One task for each distinct output key

EMR node roles and data lifetime

An EMR primary node coordinates cluster services; core nodes provide processing and HDFS storage; task nodes add processing capacity. Closing a terminal or notebook does not terminate the cluster.

Object	What it does	Lifetime consequence
Primary node	Coordinates services and applications	Its local files belong to that cluster
Core nodes	Run work and hold HDFS blocks	Cluster HDFS is lost on cluster termination
Task nodes	Add processing capacity	Do not provide persistent HDFS storage
Independent storage	Approved S3 or other external location	Can preserve outputs beyond the cluster
Shutdown	Save, verify, then terminate in EMR	Confirm the cluster reaches its final state

Tutorial 1 · Hadoop word count

The handout is Lab 06. It stages Sherlock Holmes text in HDFS, runs the supplied MapReduce word-count JAR, inspects counters and reads reducer output.

```
hadoop fs -mkdir -p /user/wordcount/input
hadoop fs -put sherlock.txt /user/wordcount/input/
hadoop jar /usr/lib/hadoop-mapreduce/\
hadoop-mapreduce-examples.jar wordcount \
  /user/wordcount/input /user/wordcount/output
hadoop fs -cat /user/wordcount/output/part-r-00000 | head
```

What to explain

-put copies a local file into HDFS.

Map emits words; reduce adds counts.

The output path must be fresh for a normal rerun.

Tutorial 1 · Evidence and cleanup

A successful launch is only the start. Connect the input file, completed job and actual output using paths, counters and result inspection.

Stage

List HDFS input

Check file and size

Execute

Job reaches completion

Inspect input/output counters

Inspect

List part files

Read actual word counts

Preserve + stop

Save required evidence

Terminate cluster

Hadoop responsibilities

Which statement correctly assigns the responsibilities of HDFS, YARN and MapReduce?

A HDFS schedules tasks; YARN stores blocks; MapReduce defines schemas.

B HDFS stores blocks; YARN allocates resources; MapReduce processes records.

C HDFS stores blocks; YARN groups keys; MapReduce allocates all cluster resources.

D HDFS defines SQL tables; YARN trains models; MapReduce stores replicas.

Answer: B

HDFS stores file blocks and their replicas.

YARN allocates resources for applications; MapReduce performs map, shuffle and reduce work.

The other choices move these responsibilities to the wrong component.

Combining averages

Mapper 1 reads values 10 and 20; mapper 2 reads 90. Which intermediate output allows a reducer to calculate the correct overall mean?

A Emit 15 and 90, then take their unweighted mean.

B Emit 30 and 90, then divide their sum by two mappers.

C Emit only each mapper’s largest value, then average them.

D Emit (sum, count) pairs (30, 2) and (90, 1), then merge them.

Answer: D

Merge the pairs to obtain (120, 3), so the mean is 40.

A mean alone loses the number of records behind it; the other methods give 52.5, 60 or 55.

Sum and count remain correct when local combining is repeated or skipped.

Read the reducer output

The word-count job has completed. Which command displays the records in its HDFS output file?

A hadoop fs -cat /user/wordcount/output/part-r-00000

B hadoop fs -ls /user/wordcount/output

C cat /user/wordcount/output/part-r-00000

D hadoop fs -mkdir -p /user/wordcount/output

Answer: A

The HDFS -cat command reads file contents and displays the word counts.

The -ls command lists files; plain cat reads the local filesystem.

The -mkdir command creates a directory and does not inspect the result.

Spark execution

The next program answers the same kind of grouped question, but Spark builds a connected plan and executes it across partitions.

Describe

RDD functions or

DataFrame expressions

Request

An action needs output

Driver schedules work

Execute

Tasks process partitions

Shuffle combines keys

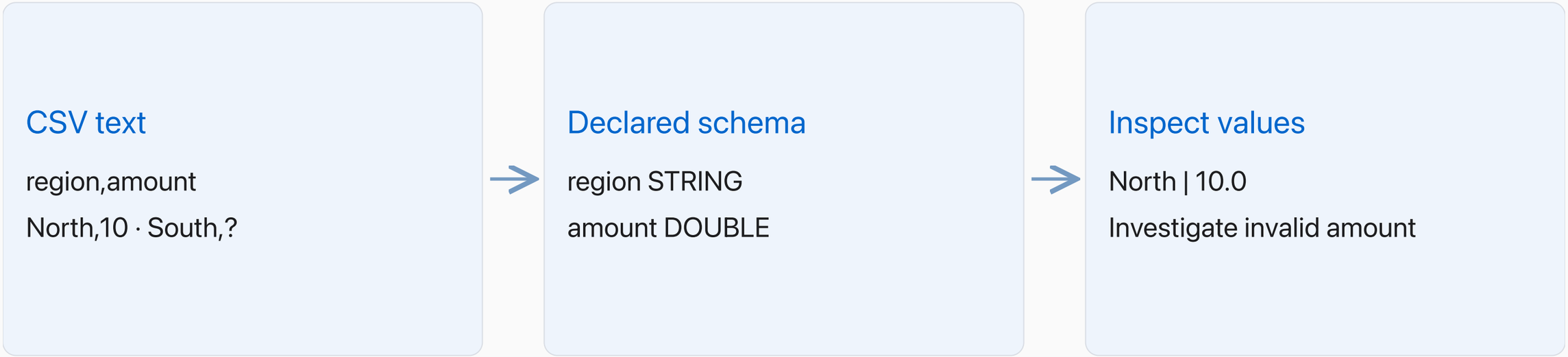
RDDs and DataFrames

RDD means Resilient Distributed Dataset: distributed records transformed with functions. A DataFrame provides distributed rows with named, typed columns that Spark can use to plan structured queries.

Property	RDD	DataFrame
Example record	("North", 10)	region = North; amount = 10
Operation	Functions inspect each record	Expressions use named columns
Structure	Application controls record interpretation	Schema defines column names and types
Use here	Trace low-level word records	Express sales totals and analytical queries
Common behaviour	Lazy transformations and distributed actions	Lazy transformations and distributed actions

Schema and data validity

A schema states column names and types. Parsing a file successfully does not prove its values satisfy the analytical rules.



Transformations and actions

Transformations describe a new distributed result; actions request computation. Creating a DataFrame variable does not mean Spark has materialised all its rows.

```
a = sales.filter("amount > 0")
b = a.select("region", "amount")
c = b.groupBy("region").sum("amount")
c.show()
```

Trace the four lines

a, b and c describe DataFrame plans.

show() requests the result.

The input sales DataFrame is unchanged.

Regional totals

Use three rows: North 10, South 20 and North 5. The required result is one summed amount per region: North 15 and South 20.

```
from pyspark.sql import functions as F
valid = sales.filter(F.col("amount") > 0)
totals = valid.groupBy("region").agg(
    F.sum("amount").alias("total"))
totals.show()
```

Expected rows · order unspecified

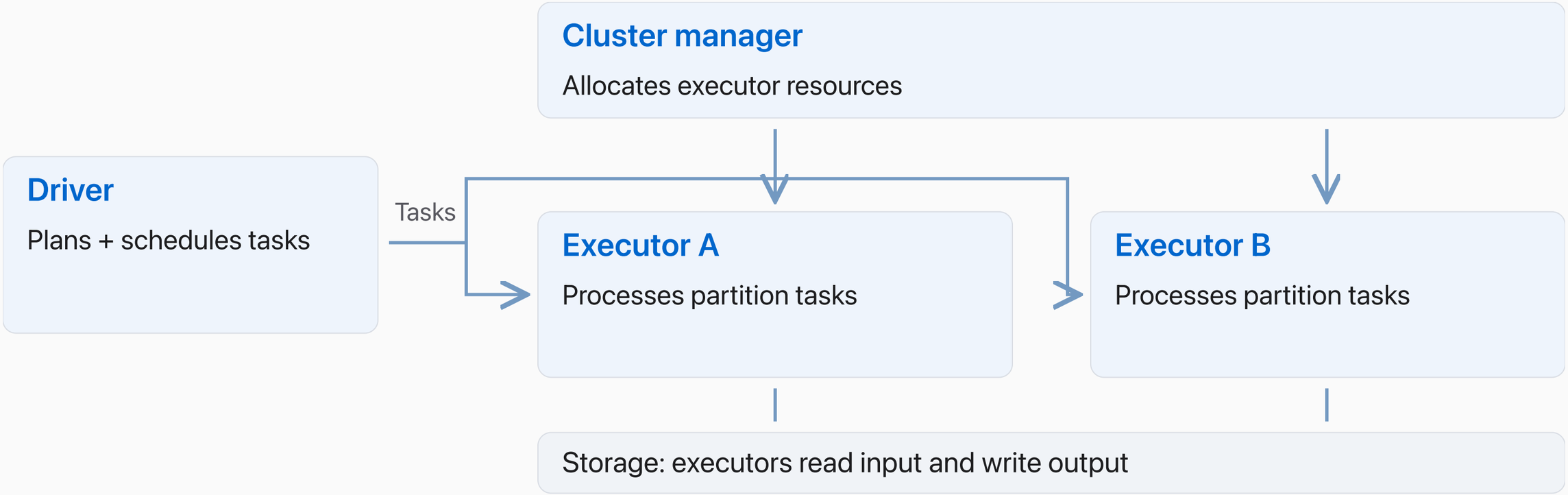
North | 15

South | 20

3 input sales → 2 groups; total = 35.

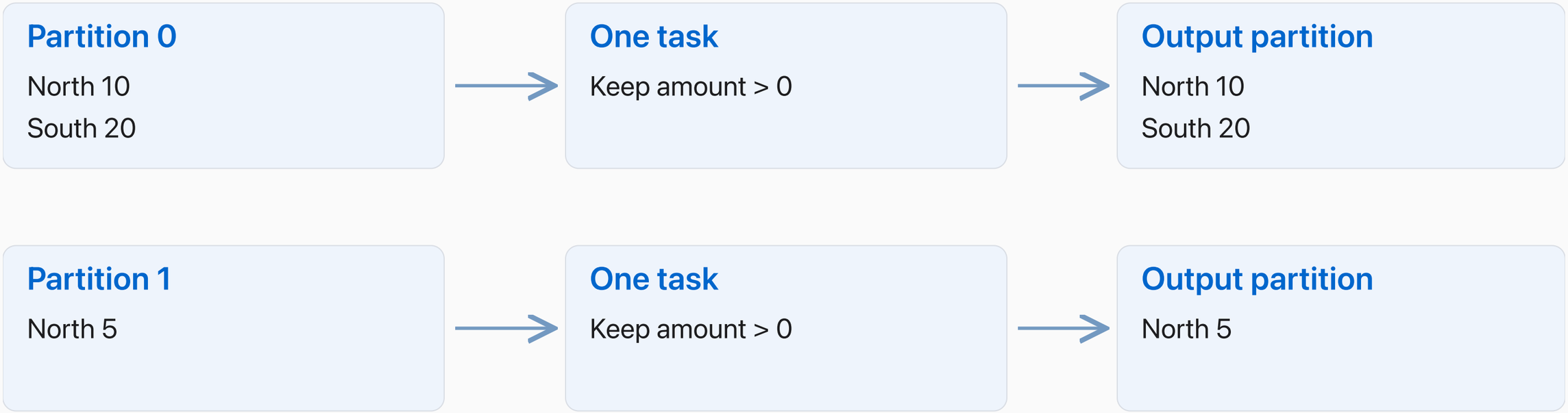
Driver, executors and manager

The driver plans and schedules work. The cluster manager allocates resources for executor processes; executors process partitions and access storage. An executor is a process, not an entire machine.



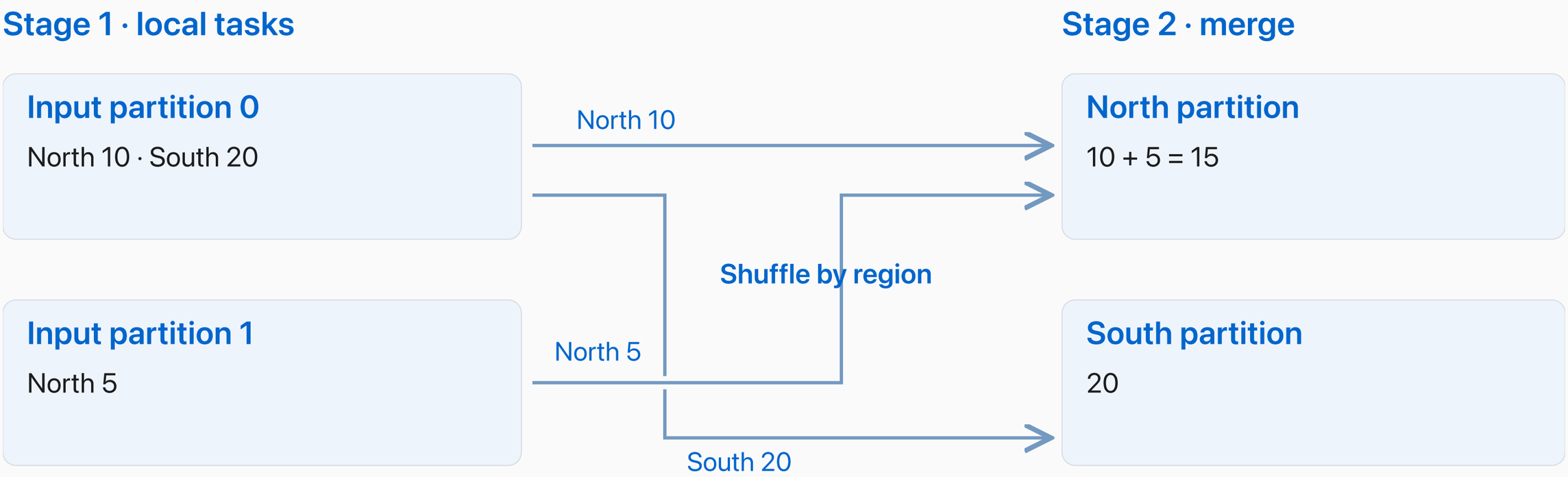
Partitions and tasks

A partition is a slice of distributed data. In a stage, one task processes one partition; rows, partitions, tasks and machines are different counts.



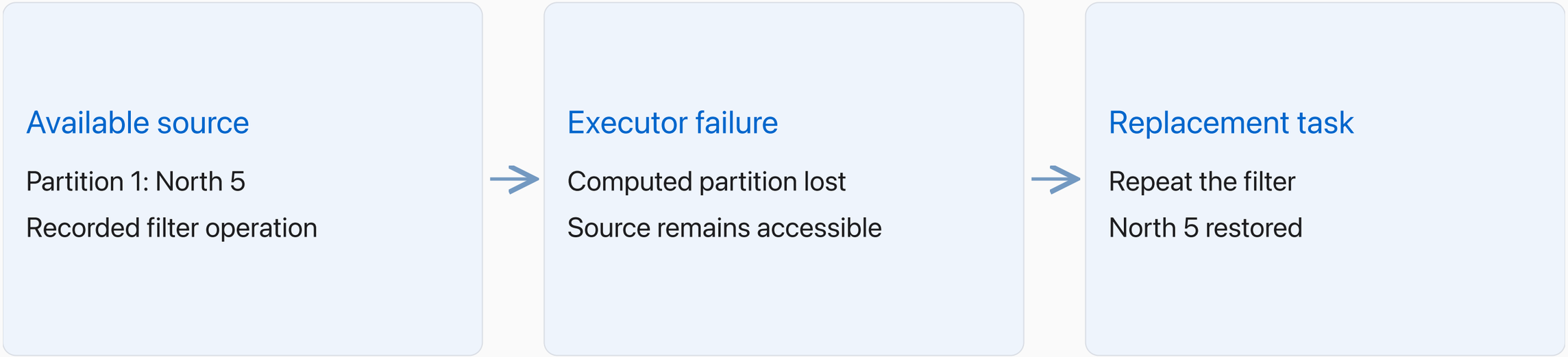
Shuffle and stage boundaries

A narrow operation can work locally within partitions. A shuffle redistributes data across partitions; a dependent stage uses that shuffle output to combine matching keys.



Lineage and recomputation

Lineage records how a computed partition was derived. Spark can recompute lost work from available inputs and resources; caching alone is not a durable backup.



Word count with DataFrames

Tutorial 2 uses the same Sherlock text as Tutorial 1. Lowercase and tokenise lines, expand words into rows, remove empty words, then group and count.

```
from pyspark.sql import functions as F
lines = spark.read.text("/user/hadoop/input/sherlock.txt")
words = lines.select(F.explode(
    F.split(F.lower("value"), r"\W+")).alias("word"))
counts = words.filter("word != ''").groupBy("word").count()
counts.orderBy(F.desc("count"), "word").show(20)
```

Meaning

explode changes one line into many word rows.

groupBy(...).count() builds a lazy aggregation.

show() requests the ordered result.

Word count with RDDs

The RDD version exposes record functions directly. flatMap emits several tokens per input line; reduceByKey merges counts for the same word.

```
import re
lines = spark.sparkContext.textFile(
    "/user/hadoop/input/sherlock.txt")
words = lines.flatMap(lambda s: re.split(r"\W+", s.lower()))
counts = (words.filter(lambda w: w != "")
    .map(lambda w: (w, 1))
    .reduceByKey(lambda a, b: a + b))
counts.take(5)
```

Same operation, different API

map creates one pair per word.

reduceByKey sums values by key.

take requests a sample; it does not rank words.

Tutorial 2 · Notebook and data paths

The handout is Lab 07. A local browser connects through an SSH tunnel to Zeppelin; Spark then reads the staged file from HDFS. These are distinct connections.

Local computer

Browser: localhost:8890

SSH forwards port 8890

EMR primary

Zeppelin notebook

Spark application driver

Cluster storage + workers

HDFS input path

Executor tasks process records

Spark trace exercise

Start with North 10, South 20 and North 5. Predict the type of a, b, c and d, identify the action, and state whether b controls the final answer.

```
a = sales.filter("amount > 0")
b = a.limit(2)
c = a.groupBy("region").count()
d = c.count()
```

Distinctions to apply

A grouped count describes a DataFrame.

A DataFrame count action returns an integer.

Only dependencies of the action need execution.

Spark trace answer

a, b and c are DataFrames. d is the integer 2 because c has one row for North and one for South; c counts sales within each region, while d counts those grouped rows.

Expression	Meaning	Result
a = sales.filter("amount > 0")	Keep positive sales	Plan for 3 rows
b = a.limit(2)	Bound a separate result	Plan for up to 2 unspecified rows
c = a.groupBy("region").count()	Count sales per region	North 2; South 1
d = c.count()	Count grouped rows	2; b is not used

Lazy evaluation

Which statement about Spark DataFrames is correct?

A Assigning a transformation to a variable materialises all output rows.

B Every transformation changes the original DataFrame in place.

C Transformations describe a result; an action requests its computation.

D A transformation automatically saves the result outside the cluster.

Answer: C

Transformations such as filter and select build a new result description.

An action such as show requests the computation needed for that result.

Planning does not mutate the input or create a durable saved copy.

Grouping across partitions

Partition 0 holds North 10 and South 20; partition 1 holds North 5. A query sums sales by region. Why is data movement needed for the North total?

A Each original sale must be collected on the driver before any sum can run.

B North contributions from different input partitions must reach the same final group.

C Every input partition already contains the complete North total.

D Grouping by region creates one executor process for every sale.

Answer: B

Local partial sums can be computed before the shuffle.

Matching North contributions are brought together to produce 10 + 5 = 15.

The driver coordinates tasks; it need not collect all original records.

Tokenise the text

lines is a DataFrame with one text line in column value. Which expression produces one lowercase token per output row?

```
from pyspark.sql import functions as F
```

A lines.select(F.split(F.lower("value"), r"\W+").alias("word"))

B lines.select(F.lower("value").alias("word"))

C lines.groupBy("value").count()

D lines.select(F.explode(F.split(F.lower("value"), r"\W+")).alias("word"))

Answer: D

Assume functions has been imported as F from pyspark.sql.

lower normalises case, split creates an array, and explode emits a row per element.

Choice A leaves an array per line; B leaves whole lines; C counts identical lines.

Correct and efficient pipelines

Once the query meaning is fixed, improve the work Spark performs while preserving the same answer.

Correctness

Types, missing values

Join multiplicity

Execution cost

Scan, shuffle, skew

Task and memory evidence

Saved output

Suitable format + layout

Read back and verify

Cleaning without changing the question

NULL means a value is missing; it does not mean zero. A filter keeps only true conditions, so amount > 0 excludes both NULL and negative amounts.

Input amount	amount > 0	Consequence
10	True	Keep 10
NULL	Unknown / NULL	Excluded by this filter
−4	False	Excluded by this filter
Business meaning	Positive sales vs net sales	Removing returns changes the requested quantity
Validation	Count rejected and missing rows	Do not hide lost records

SQL and DataFrame equivalence

A temporary view lets SQL refer to a DataFrame in the current session. Equivalent APIs must use the same population, grouping, aggregation and ordering.

```
sales.createOrReplaceTempView("sales")
totals_sql = spark.sql("""
  SELECT region, SUM(amount) AS total
  FROM sales
  WHERE amount > 0
  GROUP BY region
""")
totals_sql.show()
```

Check equivalence

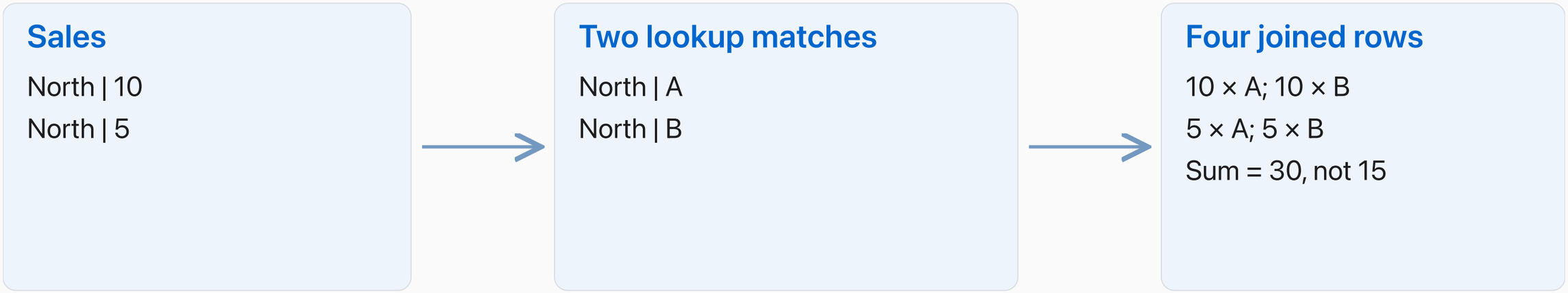
Expected North 15 and South 20.

COUNT(*) would count sales, not add amounts.

A temporary view is not a saved durable table.

Join multiplicity

A join creates one output row for each matching pair. A non-unique lookup can multiply fact rows and inflate a later sum, even if the join code executes successfully.



Lookup uniqueness controls whether a join preserves or multiplies fact rows.

Physical plans and evidence

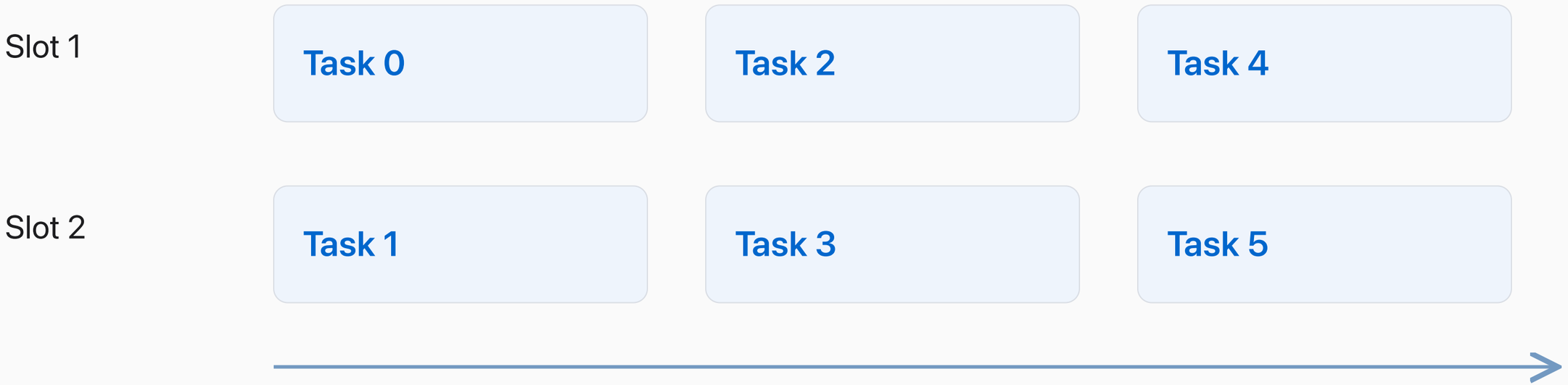
A physical plan shows operators Spark proposes to use. Look for scans, filters, exchanges and join strategies; runtime measurements establish what actually took time.

Evidence	What it reveals	What to do with it
explain("formatted")	Scan, filter, Exchange, join operators	Locate data movement and planned work
Spark UI stage metrics	Task duration, input and shuffle bytes	Identify costly stages
Task duration spread	A few tasks dominate	Check skew or resource differences
Spill metrics	Intermediate work exceeds available memory	Reduce data or adjust layout/resources
Broadcast join	Small side distributed to executors	Verify size and plan; memory is finite
Execution hierarchy	Application → jobs → stages → tasks	Actions request jobs; shuffle boundaries separate dependent stages

Parallelism and partition sizing

More partitions can expose more work, but available task slots limit simultaneous tasks. Very small partitions add overhead; very large partitions can create long tasks and memory pressure.

Six tasks · two available slots



Three waves of work if task durations are similar; times are illustrative.

Skew

Skew means work is unevenly distributed. A hot grouping key can leave one task processing far more data after most other tasks have finished.

Observation	Interpretation	Targeted response
One reducer has most bytes	A frequent key dominates a partition	Inspect key frequencies and query logic
Many tiny tasks	Scheduling overhead may dominate	Use a more suitable partition size
Large shuffle after a join	Data is redistributed widely	Filter/project early; inspect join strategy
Uniformly slow tasks	Could be scan or resource limits	Check input, CPU, I/O and spill evidence

Cache, spill and durability

Caching can reuse a computed dataset across actions; it costs memory and may spill or be evicted. A write to independent storage serves a different purpose: preservation.

```
prepared = sales.filter("amount > 0").cache()
prepared.count()
prepared.groupBy("region").sum("amount").show()
prepared.groupBy("region").count().show()
prepared.unpersist()
```

When reuse helps

The first action can populate the cache.

Later actions may reuse cached partitions.

One use may not repay caching cost.

Parquet and output layout

Parquet is column-oriented: readers can select needed columns and may skip row groups using statistics. Partition directories can exclude unrelated paths; many tiny files add overhead.

Choice	Benefit	Qualification
Column selection	Read fewer fields	Use only needed columns in the query
Directory partition predicate	Skip unrelated partition locations	Filter must match a useful partition key
Row-group statistics	Skip impossible matches	Reader support and data layout matter
repartition(n)	Redistribute data and change partitioning	Normally adds a shuffle
coalesce(n)	Often reduces partitions with less movement	Can reduce parallelism; usually no rebalance

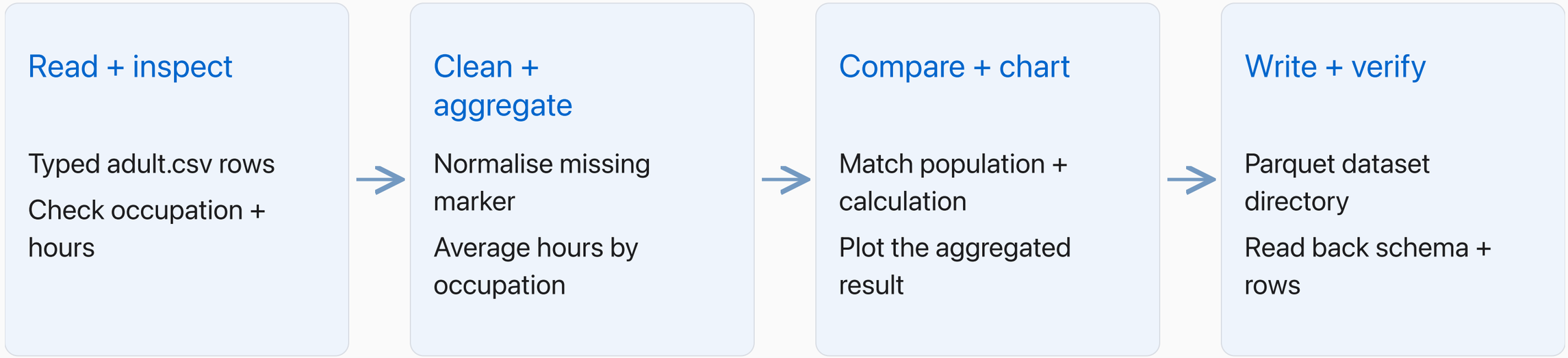
Fair performance comparisons

Compare the same input, output rules and measured work. Changing the answer, using a warm cache in only one run, or skipping output verification invalidates a speed claim.

Record	Reason
Input paths, bytes and row counts	Establish the same workload
Configuration and cache treatment	Explain resource and reuse differences
Action that triggers the full work	Lazy plan construction is not execution time
Repeated times and their spread	One run may be noisy
Stage metrics and checked output	Connect speed changes to work and correctness

Tutorial 3 · Adult data pipeline

The handout is Lab 08. Read Adult records with an explicit schema, register adult_data, compare SQL with DataFrames, chart aggregates, then save Parquet.



Tutorial 3 · Normalise before filtering

The lab reader ignores leading whitespace. Trim occupation and compare with the normalised marker "?"; a literal " ?" test can miss the intended missing-value rule.

```
from pyspark.sql import functions as F
adult = spark.table("adult_data")
clean = adult.withColumn("occupation", F.trim("occupation"))
result = (clean.filter("occupation IS NOT NULL AND occupation != '?'")
    .groupBy("occupation")
    .agg(F.avg("hours_per_week").alias("avg_hours")))
result.write.mode("overwrite").parquet(
    "/user/hadoop/results/occupation_hours")
```

Check the numerical rule too

The lab SQL truncates to two decimal places.

The DataFrame mean keeps full precision.

Compare equal rules, or state the rounding difference.

Tutorial 3 · Verify saved data

A directory listing proves that files exist; reading them back tests whether the saved dataset is interpretable and has the expected structure and values.

```
saved = spark.read.parquet(
    "/user/hadoop/results/occupation_hours")
saved.printSchema()
saved.count()
saved.orderBy("occupation").show()
```

Evidence to retain

Check one row per retained occupation.

Inspect averages and excluded records.

Preserve required output outside cluster HDFS.

Cached and durable data

Which statement correctly distinguishes a cached DataFrame from a durable saved dataset?

A Caching supports reuse, but output must be independently preserved to survive cluster termination.

B Calling cache immediately writes a permanent copy to object storage.

C Cached partitions cannot be lost when an executor fails.

D Any successful write to cluster HDFS survives cluster termination.

Answer: A

Caching reuses computed partitions; lost partitions may need recomputation.

Cluster HDFS is also tied to the cluster's lifetime.

To preserve results after termination, save to approved independent storage and verify them.

Join multiplicity

Sales contains North 10 and North 5. A lookup contains two matching North rows. An inner join uses region, then sums amount. What total does it produce?

A 15, because a join always preserves the original total.

B 20, because only the first North sale is duplicated.

C 30, because both sales match both lookup rows.

D The join must fail because lookup keys are duplicated.

Answer: C

Each sale matches two lookup rows, producing amounts 10, 10, 5 and 5.

Their sum is 30; duplicate lookup keys do not automatically make the join fail.

For a one-row-per-region lookup, validate or resolve key uniqueness before joining.

Normalise missing markers

An occupation value may be "?", " ?" or SQL NULL. Which filter keeps known occupations after leading and trailing spaces are removed?

```
from pyspark.sql import functions as F
clean = adult.withColumn("occupation", F.trim("occupation"))
```

A clean.filter("occupation != ' ?'")

B clean.filter("occupation IS NOT NULL AND occupation != '?'")

C clean.filter("occupation IS NULL OR occupation = '?'")

D clean.filter("occupation IS NOT NULL")

Answer: B

The trim operation makes both marker spellings equal to "?".

Choice B excludes the normalised marker and nulls.

A tests the wrong spelling; C retains missing values; D still retains "?".

Spark ML and evaluation

Prepared rows become a model only after a prediction target, feature contract and evaluation boundary are defined.

Define

One prediction per entity

Features available in time

Learn

Fit preparation + model

Use training data

Evaluate + reuse

Choose with validation

Test once; save fitted workflow

Learning tasks and prediction unit

Classification predicts a category; regression predicts a number; clustering groups similar records without target labels. Define what one row and one prediction represent before selecting features.

Task	Example output	Required interpretation
Classification	Churn: yes / no	Which event and positive class?
Regression	Predicted future spend	Which units and prediction horizon?
Clustering	Similarity-group identifier	No target label used for fitting groups
Prediction unit	One customer at a defined time	Prevent repeated-entity or future-data leakage

Training, validation and test

Training learns parameters, validation chooses settings, and test estimates performance after those choices are fixed. The split must represent deployment: random, chronological or separated by entity.

Boundary	Use	Mistake to avoid
Training	Fit imputation, encoding, scaling and model	Learning preparation from the full dataset
Validation	Choose features, model and threshold	Repeatedly tuning to test outcomes
Test	Evaluate the frozen final workflow	Calling tuned results an untouched test
Time / entity split	Keep future or related records out	Randomly mixing records that share information
Very large data	Choose enough representative holdout cases	Fixed percentages are not mandatory; rare cases still matter

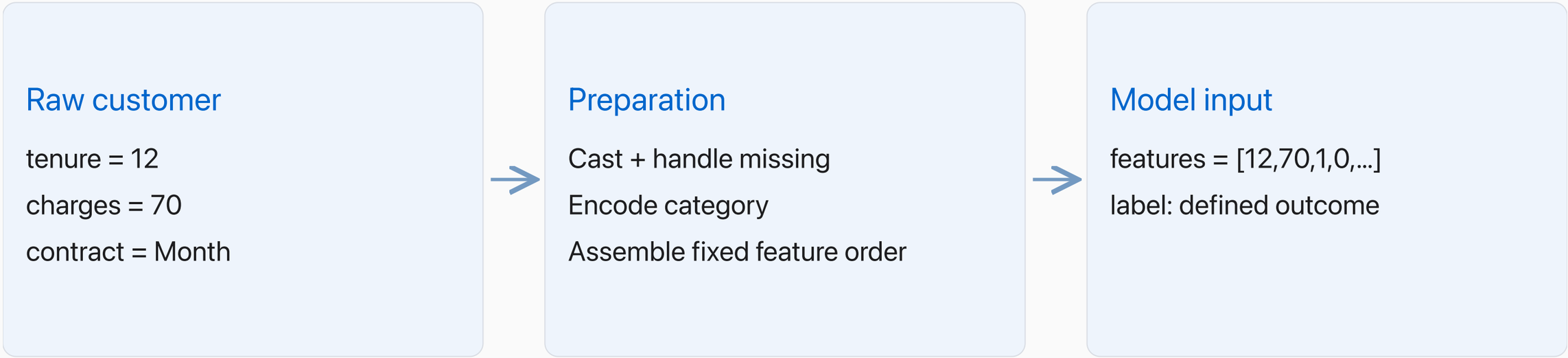
Leakage

Leakage occurs when training or selection uses information unavailable for a real prediction. High scores can result from future outcomes, overlapping customers or preparation fitted across the split.

Leakage route	Example	Correction
Future information	Cancellation reason predicts churn	Use only fields available before prediction
Repeated entity	Same customer appears in train and test	Split by customer when predicting new customers
Fitted preparation	Scaler learns statistics from all rows	Fit on training; transform held-out rows
Model selection	Choose the model with the best test score	Select on validation; reserve the test

Features and labels

Spark ML commonly expects one numeric features vector and a numeric label. Feature order, category mapping and missing-value treatment form a contract shared by training and inference.



Estimator, Transformer and Pipeline

An Estimator learns from data through fit() and returns a fitted Model. A Transformer applies transform() to rows; a Pipeline preserves the ordered sequence of preparation and model stages.

Object	Method	Effect
StringIndexer / StandardScaler	fit(training)	Learn category order or statistics
Fitted indexer / scaler	transform(new_rows)	Apply the learned mapping or statistics
VectorAssembler	transform(rows)	Build a vector in the specified column order
LogisticRegression	fit(training)	Learn weights from features and labels
PipelineModel	transform(test)	Run the complete fitted workflow without refitting

Categorical encoding

StringIndexer assigns category indices; these indices are labels, not meaningful distances. OneHotEncoder gives categories separate vector positions; choose an explicit policy for unseen values.

Input	Intermediate representation	Study point
Contract = Month	Index such as 0	Mapping depends on fitted indexer
Contract = Year	Index such as 1	1 is not twice as much contract as 0
One-hot representation	Separate category positions	Vector size depends on encoder settings
Unknown contract	Configured error or retained unknown category	Use the same fitted policy at inference
Churn label	Explicit No → 0, Yes → 1	Never assume automatic index 1 means Yes

Numeric and text preparation

Fit numeric statistics and text vocabularies on training data, then reuse them for held-out rows. Inverse document frequency (IDF) reduces the weight of terms that occur in many documents.

Input issue	Preparation	Consequence
Missing numeric values	Count and investigate; reject or fit an imputer	Dropping rows changes the evaluated population
Different numeric scales	Fit a StandardScaler on training vectors	Apply those same statistics to held-out rows
Text fields	Tokenise and build term vectors; optionally weight with IDF	Vocabulary and IDF statistics belong inside the training boundary
New prediction rows	Apply the saved fitted preparation	Preserve feature order, types and unknown-category policy

Fit the complete workflow

Split rows before learning preparation. This condensed numeric example assumes clean tenure and MonthlyCharges columns and explicitly maps Yes to positive class 1.

```
from pyspark.sql import functions as F
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
labelled = clean.withColumn("label",
    (F.col("Churn") == "Yes").cast("double"))
train, test = labelled.randomSplit([0.7, 0.3], seed=1234)
assembler = VectorAssembler(inputCols=["tenure", "MonthlyCharges"],
                            outputCol="features")
pipeline = Pipeline(stages=[assembler, LogisticRegression()])
fitted = pipeline.fit(train)
predictions = fitted.transform(test)
```

Pipeline boundary

Learned preparation also belongs inside the pipeline.

Use separate validation data when choosing settings.

A seed helps repeatability; it does not prevent leakage.

Logistic regression outputs

Logistic regression converts a weighted feature score into a class probability. A decision threshold converts a score into a predicted class; changing the threshold changes errors without retraining the weights.

Column or quantity	Meaning	Interpretation
rawPrediction	Model score representation	Used by a suitable evaluator
probability	Estimated class-probability vector	Check which position is the positive class
prediction	Class chosen by the decision rule	Depends on the threshold
Lower positive threshold	More cases predicted positive	Recall can rise while false positives rise
Class probability	A model estimate	Does not establish causation or certainty

Confusion matrix

Use class 1 = churn. True positives are actual churners predicted positive; false negatives are actual churners missed by the model.

Six customers · threshold 0.5 · positive = churn

	Predicted churn	Predicted no churn
Actual churn	**TP = 2** Churners found	**FN = 1** Churners missed
Actual no churn	**FP = 1** False alarms	**TN = 2** Non-churners correct

Accuracy, precision and recall

For TP = 2, FP = 1, FN = 1 and TN = 2, accuracy, precision and recall are all 2/3. Their equal numbers here hide different meanings.

Metric	Formula	What it asks
Accuracy	(TP + TN) / all = 4 / 6	What fraction of all decisions were correct?
Precision	TP / (TP + FP) = 2 / 3	Among predicted churners, how many churned?
Recall	TP / (TP + FN) = 2 / 3	Among actual churners, how many were found?
Imbalance example	95 non-churners out of 100	Always predicting no churn gives 95% accuracy and zero churn recall
Choice	Match costs and positive class	A high overall accuracy can hide poor minority detection

ROC AUC

ROC AUC measures how well scores rank positives above negatives across thresholds. It is not the fraction of correct classifications at a single threshold.

Scores in a small example	Interpretation
Positive scores: 0.9, 0.4	Two actual positives
Negative scores: 0.8, 0.2	Two actual negatives
Correctly ordered pairs	0.9 beats both; 0.4 beats 0.2 → 3 of 4
ROC AUC = 0.75	Ranking quality; ties contribute half credit
At threshold 0.5	One positive missed and one negative flagged
Precision–recall view	Often informative when positives are rare; check deployment prevalence

Model and threshold selection

Choose settings with validation data, then evaluate the fixed choice on test data. Cross-validation repeats fitting across folds; cost grows with parameter choices, folds and pipeline stages.

Baseline

Simple reference model

Same rows + target

Validation

Compare settings

Choose metric + threshold

Test

Freeze all decisions

Report honest held-out result

Reuse

Save fitted PipelineModel

Check reloaded predictions

Tutorial 4 · Customer churn

The handout is Lab 09. Cast numeric fields, investigate missing TotalCharges, build features, train logistic regression, and evaluate predictions on held-out customers.

Lab step	Meaning	Revision clarification
Numeric casting	Convert tenure and charge fields	Count invalid / excluded rows; assess resulting population
Feature preparation	features vector + label	Define positive label and preserve feature order
Split and fit	Learn using training customers	Fit learned preparation after splitting
Evaluation	BinaryClassificationEvaluator	areaUnderROC is ROC AUC, despite the lab variable accuracy
Reuse	Apply fitted preparation and model	Save the whole workflow, not only classifier weights

ML boundary exercise

A team fits an imputer and category indexer on every customer, selects a threshold using test scores, and reports areaUnderROC = 0.91 as 91% accuracy. Identify the three problems.

Preparation

Full dataset used

Before the split

Selection

Threshold chosen

Using test outcomes

Reporting

ROC AUC = 0.91

Called 91% accuracy

ML boundary answer

Preparation leaked held-out information, test data was used for selection, and a ranking metric was mislabelled as classification accuracy. Each flaw changes how the result can be trusted.

Problem	Repair
Preparation fitted on all rows	Split first; fit learned stages only on training
Threshold chosen on test	Choose on validation; evaluate the frozen choice on untouched test
ROC AUC called accuracy	Report ROC AUC as ranking quality; compute accuracy separately if needed
Saved classifier only	Persist the fitted preparation and classifier together

Learning and applying parameters

Which statement correctly distinguishes an Estimator from a fitted Transformer in Spark ML?

A An Estimator only changes column names; a Transformer always learns new parameters.

B Both must be fitted again whenever a held-out dataset is scored.

C An Estimator is a metric; a Transformer is a train/test split.

D An Estimator learns through fit; its fitted model applies learned parameters through transform.

Answer: D

For example, LogisticRegression is fitted on training data to produce a model.

That model transforms held-out rows into predictions without refitting on them.

Learned preparation must follow the same training boundary.

Selecting a threshold

A churn team compares ten thresholds on the test set and reports the best test F1 score. Which correction best protects the final evaluation?

A Choose the threshold on validation data, freeze it, then evaluate once on an untouched test set.

B Keep choosing on test data, but rename the score ROC AUC.

C Choose the threshold with the highest training F1 and report that as the test score.

D Combine training and test rows so that the final score uses more data.

Answer: A

Threshold selection uses outcome information and therefore belongs in validation.

After test results guide selection, that test set is no longer an untouched final check.

A fresh held-out set is needed for an independent final estimate.

Name the evaluation metric

In the churn tutorial, evaluator.evaluate(predictions) returns 0.91 with the configuration below. What does 0.91 represent?

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")
```

A Exactly 91% of customers have the correct predicted class.

B Precision is 91% at the current classification threshold.

C ROC AUC, which measures ranking performance across thresholds.

D The model predicts that every customer has a 91% churn probability.

Answer: C

metricName explicitly selects areaUnderROC, so the result is ROC AUC.

It is not accuracy, threshold-specific precision or an individual customer probability.

Naming a variable accuracy does not change the evaluator’s metric.

Hive and analytical SQL

The final topic exposes stored records as tables. Correct metadata and query semantics determine whether the SQL result represents the intended data.

Files

Typed fields need a format

Storage location matters

Catalog

Table schema + locations

Partitions are metadata too

SQL + execution

Filter rows; group; join

Check and preserve results

Hive, Spark SQL and storage

Hive and Spark SQL can query distributed data through SQL, but have separate planning and execution paths. The Hive execution engine depends on release and configuration; a SQL query does not imply Hive runs on Spark.

Component	Role	Verification
Metastore / catalog	Names, schemas, formats and locations	DESCRIBE FORMATTED; inspect table properties
Hive planner + driver	Resolve and plan HiveQL	Inspect the query plan
Configured Hive engine	Execute distributed operators	SET hive.execution.engine; record release
Spark SQL	Spark planning and execution	Inspect its own plan and catalog configuration
Storage	Contains the record bytes	Check files and interpreted rows

External table definition

Schema-on-read interprets existing records using a declared schema and format. The MovieLens file has four tab-separated columns; CREATE TABLE registers how to read them.

```
CREATE EXTERNAL TABLE ratings (
  user_id INT,
  movie_id INT,
  rating INT,
  rating_ts INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION '/user/hive/data/movielens/';
```

Four fields in order

user_id, movie_id, rating, rating_ts.

LOCATION is a dataset directory.

Creating metadata does not copy or validate every record.

Metadata, validity and lifecycle

A table can exist while its delimiter, column order or location is wrong. Managed and external table types have different lifecycle defaults, but release and purge properties can change deletion behaviour.

Check	Reason
DESCRIBE FORMATTED ratings	Confirm schema, format, type and location
SELECT * FROM ratings LIMIT 5	Inspect interpreted values, not only metadata
COUNT(*), null counts, rating range	Detect missing fields, duplicates and invalid values
Directory contents	Unrelated or duplicated files can change counts
Before DROP TABLE	Inspect type, purge settings and release; external is not a backup

Rows, groups and ordering

WHERE filters input rows, GROUP BY forms groups, aggregate functions calculate group values, and HAVING filters those groups. ORDER BY defines ranking; LIMIT alone does not mean top.

Rows

FROM + JOIN

WHERE keeps true rows

Groups

GROUP BY movie_id

COUNT(*) + AVG(rating)

Group selection

HAVING tests aggregates

Minimum count / average

Rank + return

ORDER BY count DESC

LIMIT requested number

Movie ratings example

Use six ratings: movie 10 has 5 and 3; movie 20 has 4, 5 and 3; movie 30 has 5. These values distinguish popularity, average score and sample size.

Movie	Ratings	Count	Average
10	5, 3	2	4.0
20	4, 5, 3	3	4.0
30	5	1	5.0
Interpretation	Most rated: movie 20	Highest mean: movie 30	Different questions

WHERE and HAVING

Removing low ratings before averaging changes the quantity. To keep movies whose average is at least 4, aggregate all intended rating rows first and test the group with HAVING.

```
SELECT movie_id, COUNT(*) AS n, AVG(rating) AS avg_r
FROM ratings
GROUP BY movie_id
HAVING COUNT(*) >= 2 AND AVG(rating) >= 4
ORDER BY movie_id;
```

Six-row example result

10 | 2 | 4.0

20 | 3 | 4.0

Movie 30 fails the minimum count.

WHERE rating >= 4 would remove movie 10's rating 3.

COUNT and NULL

COUNT(*) counts rows; COUNT(column) counts only rows with a non-null value in that column. AVG also ignores null inputs, so report the denominator behind an average.

Rating input	COUNT(*)	COUNT(rating)	AVG contribution
5	1	1	5
NULL	1	0	Ignored
3	1	1	3
Result	3 rows	2 values	(5 + 3) / 2 = 4

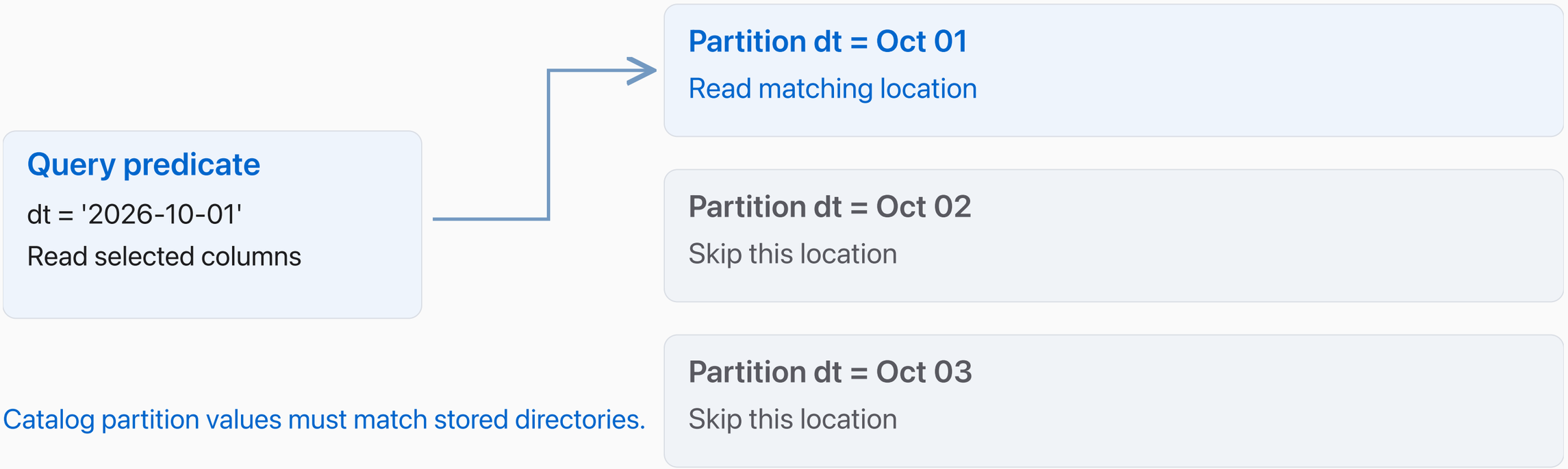
Left joins and right-side filters

An inner join removes unmatched left rows; a left join keeps them with null right fields. A right-side WHERE condition can then remove those retained rows.

Six ratings; lookup has 10 and 20	Output consequence
Inner join on movie_id	5 rows; movie 30 disappears
Left join with unique lookup	6 rows; movie 30 has NULL title
Left join with two lookup rows for 10	8 rows; movie 10 ratings are duplicated
WHERE m.title = 'Alpha' after LEFT JOIN	Keep only matching Alpha rows; null titles do not pass
AND m.title = 'Alpha' inside ON	All left ratings remain; other right titles become NULL

Storage partitions and pruning

A Hive partition maps a partition-key value to a location. Register the locations correctly; a predicate on that key can exclude other directories before data-level filtering.



Catalog partition values must match stored directories.

Nested data and shared catalogs

A STRUCT contains named fields; an ARRAY contains elements. A compatible SerDe, or serializer/deserializer, decodes stored records so nested fields can be queried.

```
SELECT text, author.name, author.followers
FROM posts
WHERE author.followers > 1000000;
```

Interpretation

author is a STRUCT with name and followers.

A matching JSON SerDe and record layout are required.

Shared catalogs need compatible types, formats and permissions.

Tutorial 5 · MovieLens analysis

The handout is Lab 10. Stage one intact MovieLens 100K ratings file, register the external table, verify COUNT(*) = 100000, and answer the specified grouped questions.

```
SELECT movie_id, COUNT(*) AS n, AVG(rating) AS avg_r
FROM ratings
GROUP BY movie_id
HAVING COUNT(*) >= 100 AND AVG(rating) > 4.0
ORDER BY avg_r DESC, movie_id
LIMIT 5;
```

Match the stated task goal

At least 100 ratings, despite the “few reviews” heading.

Average strictly greater than 4.0.

For most-rated movies, order by count instead.

Tutorial 5 · Checks and interpretation

One query counts rating events; another counts events within each rating value. Keep rows, movie groups and result limits distinct, and preserve query evidence before shutdown.

```
SELECT COUNT(*) FROM ratings;

SELECT rating, COUNT(*) AS n
FROM ratings
GROUP BY rating
ORDER BY rating;
```

Checks

100000 is expected for one intact MovieLens 100K file.

Sum of per-rating counts should equal total rows.

Unexpected counts may signal duplicate files or parsing errors.

Schema-on-read

What does creating an external Hive table over existing files establish?

A Proof that every stored record is valid and unique.

B Metadata describing how to interpret records at the declared location.

C A mandatory copy of every record into a new managed directory.

D Proof that the query will execute using Spark.

Answer: B

The table definition supplies column names, types, format and a location.

Creating metadata does not validate every value, copy all bytes or fix the execution engine.

Inspect interpreted rows and the actual engine configuration separately.

Filtering rows or groups

Movie 10 has ratings 5 and 3. The task is to keep movies with at least two ratings and an average of at least 4. Which approach gives the intended decision?

A Apply WHERE rating >= 4, then require COUNT(*) >= 2.

B Apply LIMIT 2 before grouping, then keep every resulting movie.

C Keep the movie because MAX(rating) >= 4.

D Group all intended ratings, then apply HAVING COUNT(*) >= 2 AND AVG(rating) >= 4.

Answer: D

Both original ratings belong in the calculation: count is 2 and average is 4.

Choice D keeps movie 10 using the required group-level conditions.

Choice A removes rating 3 first and changes the population; B and C test different rules.

Select qualifying movies

For the MovieLens task, which clause completes this query to retain movies with at least 100 ratings and an average strictly greater than 4.0?

```
SELECT movie_id, COUNT(*) AS n, AVG(rating) AS avg_r
FROM ratings
GROUP BY movie_id
```

A HAVING COUNT(*) >= 100 AND AVG(rating) > 4.0

B HAVING COUNT(*) > 100 AND AVG(rating) >= 4.0

C HAVING COUNT(*) >= 100 OR AVG(rating) > 4.0

D HAVING COUNT(*) >= 100 AND MAX(rating) > 4.0

Answer: A

At least 100 requires >= 100; strictly greater than 4.0 requires > 4.0.

AND requires both group conditions to hold.

B changes the boundaries, C accepts either condition, and D tests the maximum instead of the mean.

Connections across the five topics

The same evidence chain applies to every tool: define the data, interpret it correctly, execute the intended work, then verify and preserve the result.

Storage + metadata

HDFS blocks / table schema

Find and interpret records

→

Execution

YARN / Spark / Hive engine

Resources and data movement

→

Correct result

SQL rules / ML boundary

Counts, metrics and saved evidence

Keep the quantities separate

Many errors come from using the same word for different levels. State exactly what is being counted, stored, learned or returned.

Distinction	Meaning
Row / group / partition / task	Record; matching-key set; data slice; unit of stage work
Block / split / output file	Storage unit; logical mapper input; written dataset part
Schema / validation	How values are interpreted; whether they satisfy rules
Transformation / action	Describe a result; request its computation
Cache / durable save	Reusable computed partitions; independently preserved output
fit / transform / evaluate	Learn parameters; apply parameters; measure outcomes

Pipeline exercise

Sales are North 10, South 20, North 5 and South −4. The required answer is net revenue per region, including returns. A lookup has two rows for North. The team filters amount > 0, joins the lookup, caches the result and reports success after writing to HDFS. Diagnose the result.

Requested quantity

Net revenue

Returns remain in scope

Implemented pipeline

Positive filter

Duplicate-key join

Reported evidence

Cache + HDFS write

No read-back or independent copy

Pipeline exercise answer

Correct net totals are North 15 and South 16. The positive filter incorrectly removes the return, and the duplicate North lookup doubles its contributions.

Issue	Consequence	Required correction
amount > 0	South becomes 20 instead of 16	Keep returns under the net-revenue rule
Two North lookup matches	North becomes 30 after the join	Resolve lookup multiplicity or define intended one-to-many logic
Cache	Reuse only; not independent preservation	Write to approved independent storage
Successful HDFS write	Files exist, but values may be wrong	Read back; verify 2 rows and total 31
Cluster termination	Cluster HDFS disappears	Preserve and verify required results first

What each tutorial demonstrates

The tutorial result should establish both execution and meaning. A screenshot of a ready cluster or a directory alone does not prove the calculation is correct.

Tutorial	Practical result	Knowledge it demonstrates
1 · Hadoop	MapReduce word counts + counters	HDFS paths, keyed grouping, reducer output
2 · Spark	Interactive ordered word counts	Tunnel, DataFrame plan, action, shuffle
3 · Pipeline	Adult aggregates saved as Parquet	Schema, cleaning, API equivalence, read-back
4 · ML	Churn predictions + named metric	Features, fit/transform, split and evaluation
5 · Hive	Movie counts, averages and HAVING	Schema-on-read, SQL groups, query meaning

A reliable analytical result

Preserve the chain from source records to the final claim. Correctness, execution evidence and interpretation need to agree.

Define

One row / target / scope

Types and business rules

Compute

Suitable engine + resources

Control joins and movement

Check

Expected counts and values

Honest held-out metrics

Preserve

Code + data + fitted workflow

Verified independent output

Lecture and tutorial references

This revision consolidates the current lecture decks and the five supplied lab handouts. Tutorial numbering 1–5 on the contents page corresponds to original handouts Lab 06–10.

Reference	Original resource
Topic 1	Big Data, Hadoop and MapReduce · 80 slides
Topics 2 + 3	Spark · 60 slides
Topic 4	Spark ML · 62 slides
Topic 5	Hive · 61 slides
Tutorials 1–5	06-Lab-Hadoop; 07-Lab-Spark; 08-Lab-Pyspark; 09-Lab-ML; 10-Lab-HIVE