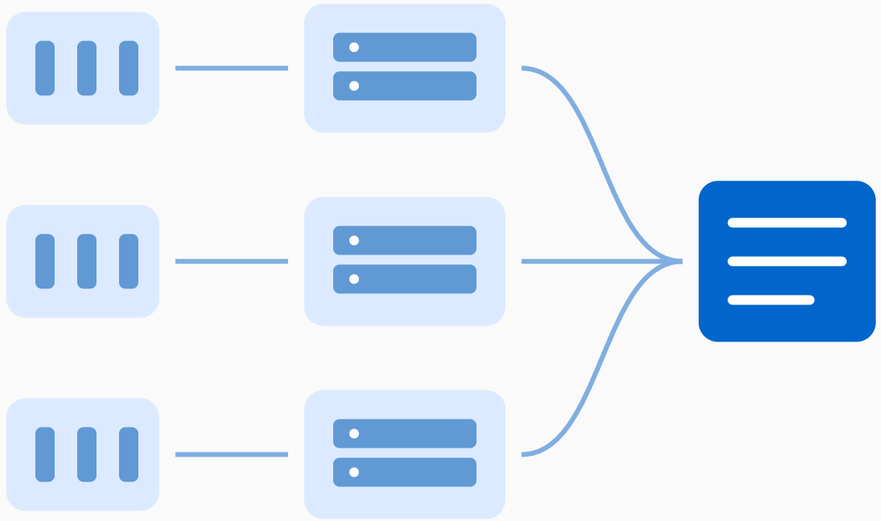
Big Data, Hadoop and MapReduce

Scale, storage and distributed processing



Liu Junhua, PhD

j@forth.ai

Topic 1: what you should be able to do

Use these outcomes to check your understanding of Topic 1.

Skill	Explain or demonstrate
Diagnose	Use the five Vs; distinguish data quality from capacity.
Compare	Scale up vs scale out; explain a benefit and a cost.
Explain	HDFS storage, YARN resources and MapReduce processing.
Trace	Read/write paths, task allocation and keyed aggregation.
Justify	Storage choice, combiner correctness and EMR data survival.

Topic 1 study map

Study a section, answer without looking, then check the worked explanation.

Topic	Revision task	Slides
1 · Big Data + scaling	Diagnose pressures; compare capacity choices.	8–26
2 · Hadoop + HDFS	Separate roles; trace storage and recovery.	27–41
3 · YARN	Match each actor to its scope and actions.	42–48
4 · MapReduce	Trace pairs, groups, tasks and output files.	49–65
5 · EMR + conclusion	Explain node roles; practise and check answers.	66–80

Five lectures: from distributed data to useful results

01

Hadoop

HDFS, YARN and MapReduce

Distribute storage and work.

02

Spark

Architecture and DataFrames

Build a data pipeline.

03

Performance

Plans, shuffle and storage

Explain where resources go.

04

Spark ML

Features, models and evaluation

Evaluate predictions correctly.

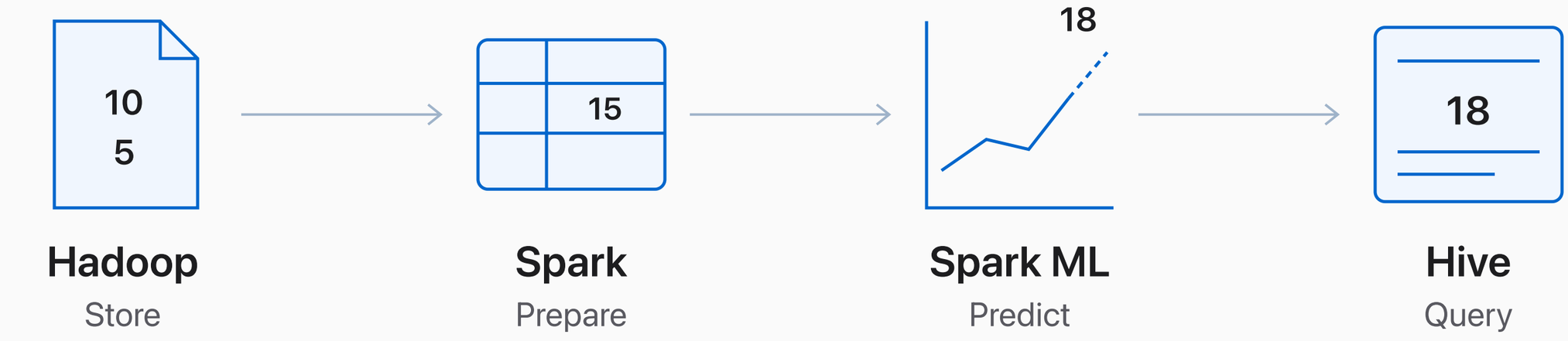
05

Hive

Catalog, tables and SQL

Query shared data.

From sales data to a demand forecast



Practise the concepts in five labs

Save results and evidence. Terminate the cluster when the lab is complete.

01

Lab 06

Hadoop

Run distributed processing.

02

Lab 07

Spark

Use Spark operations.

03

Lab 08

PySpark pipelines

Transform and store data.

04

Lab 09

Machine learning

Build and evaluate a model.

05

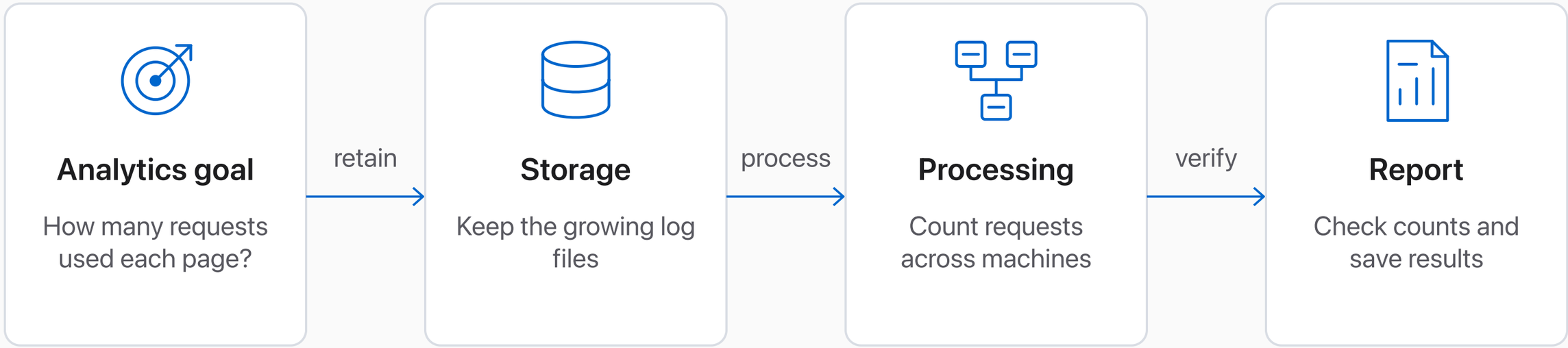
Lab 10

Hive

Define tables and run SQL.

Real-world example: turning web logs into user analytics

Store the records, share the work, and check the answer against the business question.



Big Data

What makes data useful, and when does its size become a problem?

Big Data: size matters relative to the task

Describe the data and the task before choosing tools.

Question	Meaning	Example
What makes it big?	Data strains the available storage or processing.	No universal file-size threshold.
What makes it useful?	It can answer a defined question.	Orders measure sales; clicks measure interest.
What makes it trustworthy?	Identity, missing values and timing are checked.	A Tuesday release cannot inform Monday’s prediction.

The five Vs describe different pressures



Volume

How much must fit?



Velocity

How quickly must it arrive?



Variety

Which formats and sources?



Veracity

How reliable is it?



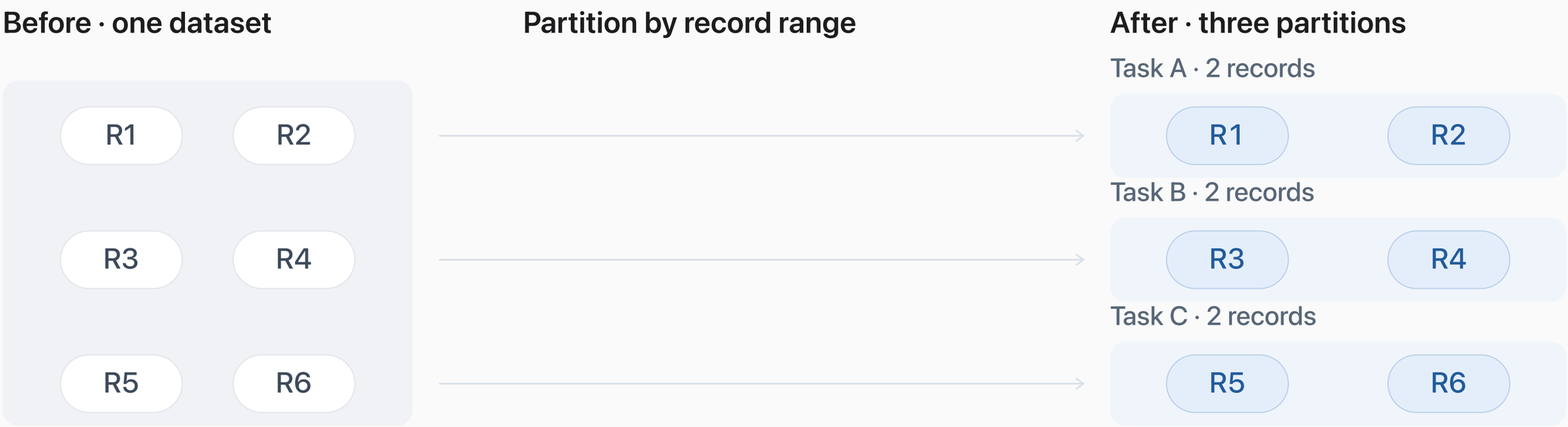
Value

Which decision improves?

Volume: the amount of data

Volume is the quantity of data to store and process. It becomes a challenge when the dataset exceeds the capacity of the available system.

Partitioning divides the work; it does not reduce the total data. Here, all six records remain across three partitions.



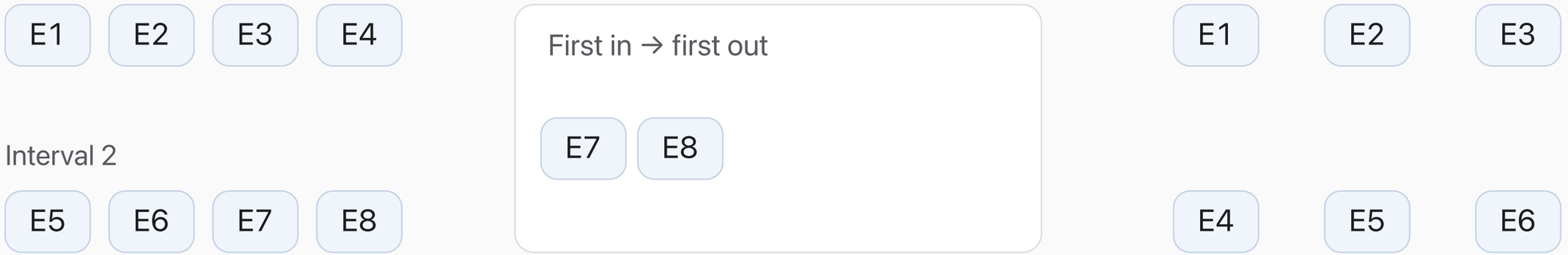
6 input records → 3 partitions × 2 records. No records are lost.

Illustrative data transformation

Velocity: the rate of incoming data

Velocity describes how quickly data arrives and must be processed. A system must keep up with arrivals and meet the required response time.

Arrivals exceed processing by one event per interval, so the backlog grows. Buffering delays the problem; sustained throughput must catch up.



Illustrative data transformation

Variety: different forms of data

Variety means differences in data formats, structures and meaning. Tables, nested records and images require different ways to read and interpret them.

Parsing makes fields usable, but does not make their meanings equivalent. Image dimensions describe its size, not what the image shows.

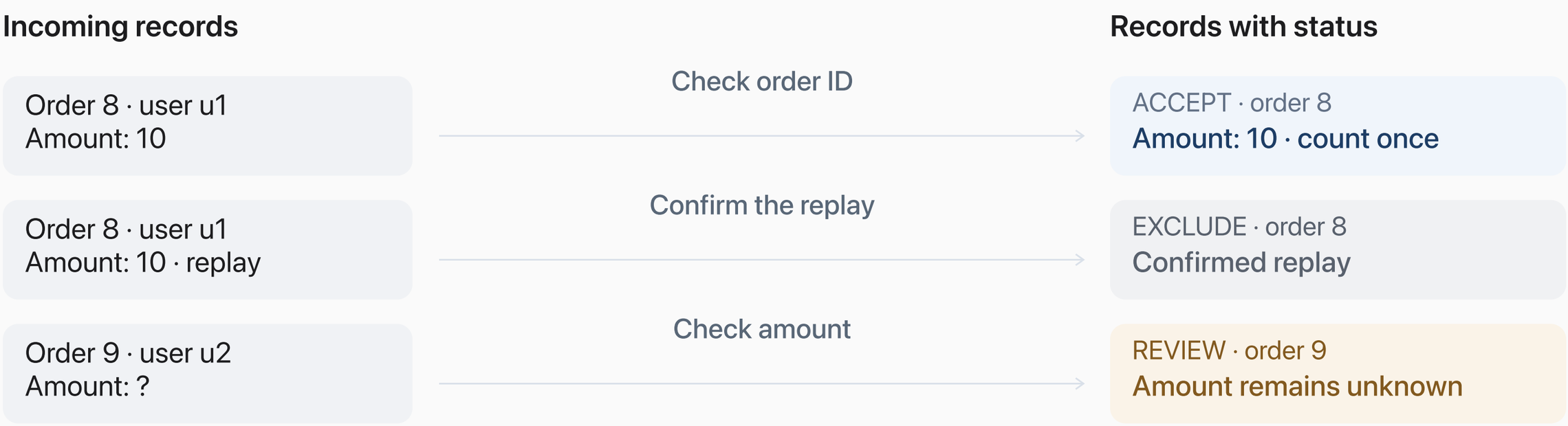
Source data			Usable fields
CSV `id,amount 7,12`	Read CSV	→	Numeric fields id: 7 · amount: 12
JSON `{"id":7,"event": {"type":"view"}}`	Parse JSON	→	Event fields id: 7 · type: view
IMAGE I7 	Measure width and height	→	Image metadata I7 · width: 2 · height: 2

The image produces dimensions, not a semantic label.

Veracity: the trustworthiness of data

Veracity concerns data accuracy, completeness and reliability. Duplicates, missing values and incorrect records can make a technically correct calculation misleading.

Exclude the confirmed replay and keep the missing amount unknown. The accepted amount is 10; it is not a complete total for all orders.



Known accepted amount: 10. Order 9 still needs review.

Value: the usefulness of data

Value is the benefit gained by using data to answer a meaningful question or improve a decision. More data or faster processing does not automatically create value.

Demand of 7 minus stock of 4 suggests a gap of 3. The uncertainty range makes this evidence for a replenishment decision, not a guaranteed order quantity.

1 Read the data

Available stock · 4 units



Expected demand · 7 units



Demand range: 5–9 units

Required arrival: Friday

2 Find the gap

7 – 4 = 3 units



4 covered

3 short

Gap range: 1–5 units

Demand range minus stock

3 Review replenishment

Review: 3 units



Range: 1–5 units

Arrival: Friday

Review before placing an order.

Store N · item A | Each box represents one unit.

Illustrative data transformation

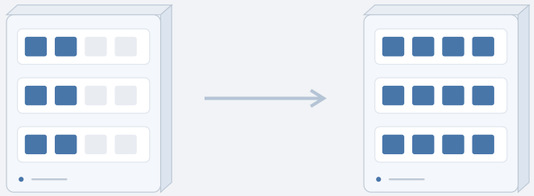
Scaling

How can we add capacity when one machine is no longer enough?

Compare scaling up with scaling out

More machines introduce coordination, networks and partial failures.

Scale up



Add resources to one host.
CPU, memory and storage

Scale out



Add hosts and divide the work.
Tasks run on separate machines.

Benefit and cost



Parallel work

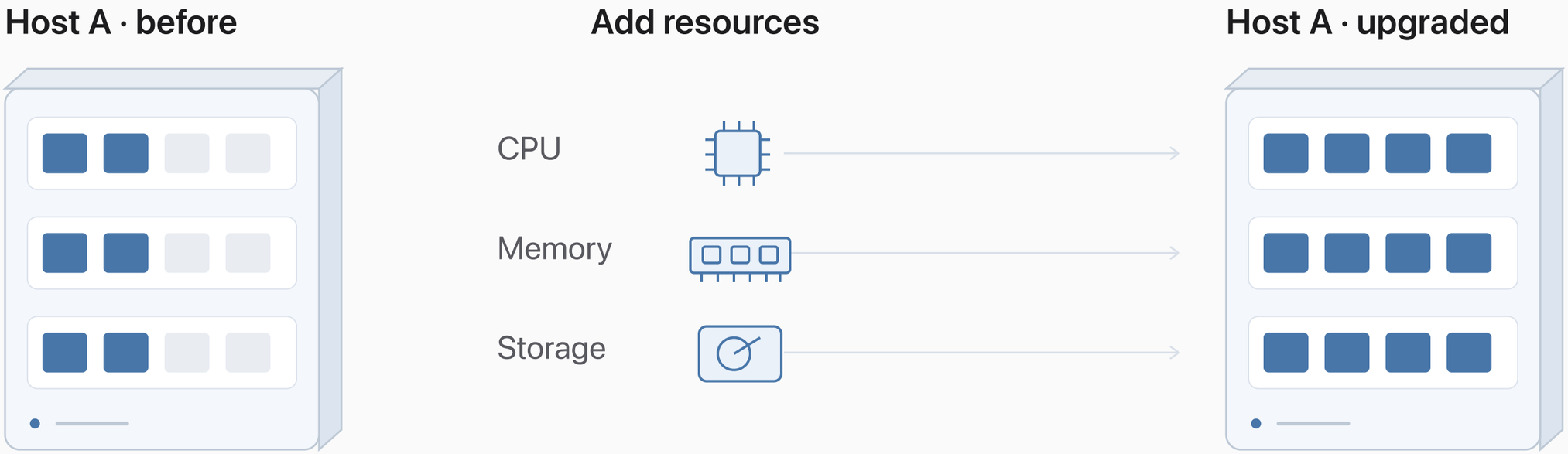


Coordination

More capacity, with time spent moving and combining results.

Scale up: add resources to one machine

A larger server helps only when the added resource is the bottleneck.

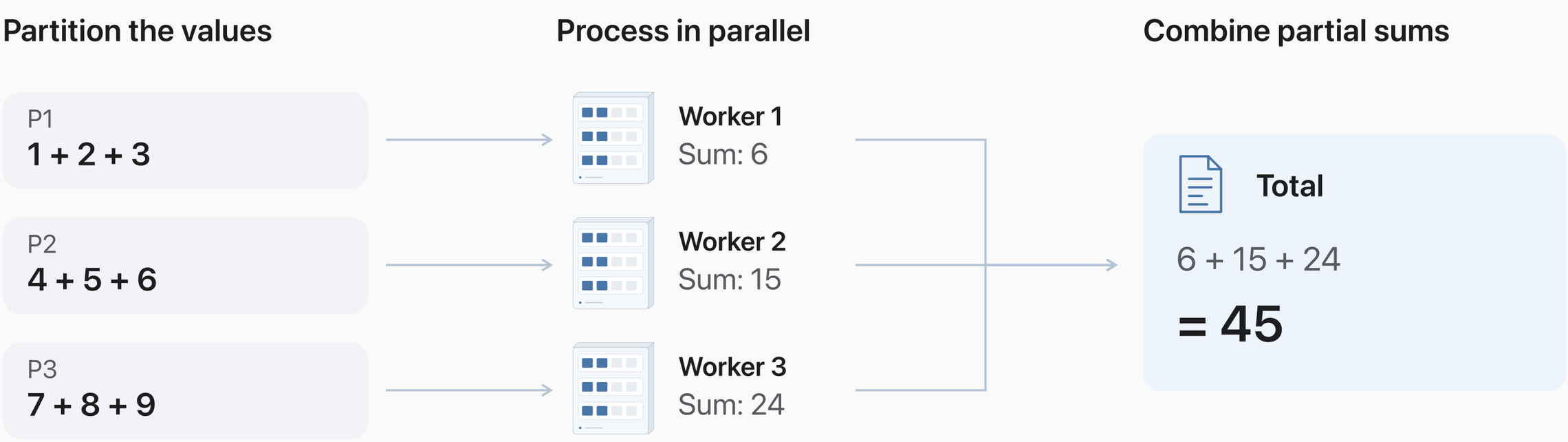


One machine before and after. One failure domain.

Illustrative resource upgrade

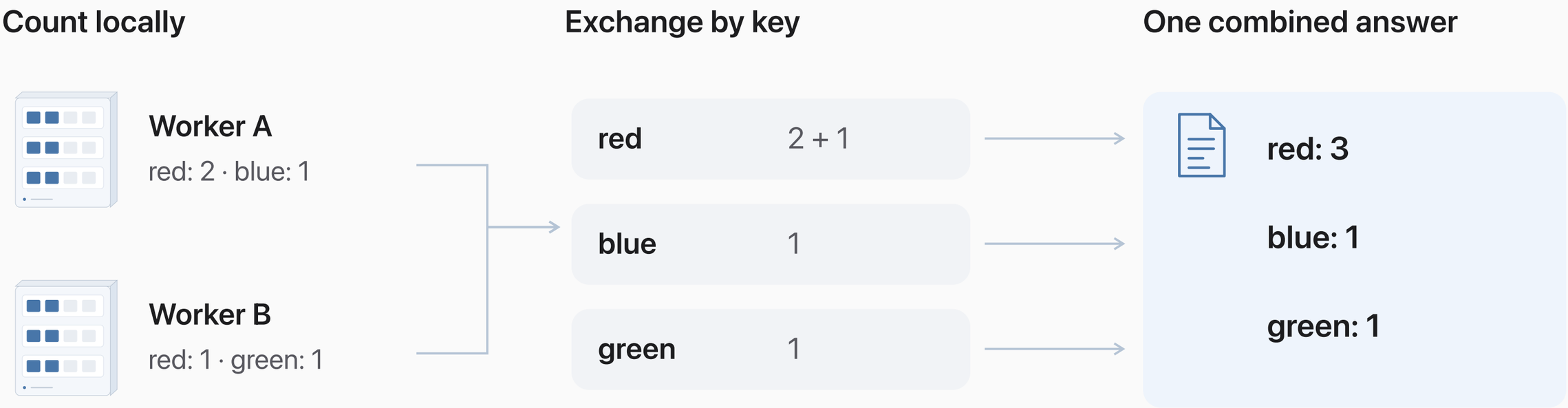
Scale out: divide work across machines

The benefit of parallel work must exceed scheduling and communication costs.



Trade-off: a shared result needs coordination

Workers process data in parallel, but they must exchange partial results to produce one correct answer.



Revision: identify the problem before the remedy

For a scenario answer: identify the pressure → explain the cause → justify the remedy.

Distinction	Rule	Reason
Volume / velocity	Amount of data / rate of arrival.	A growing queue signals insufficient throughput.
Variety / veracity / value	Formats / trust / useful outcome.	Different pressures need different remedies.
Scale up	Add CPU, memory or storage to one host.	Helps only if that resource limits the work.
Scale out	Add hosts and divide work.	Communication and coordination reduce the gain.
Capacity / correctness	Measure speed; validate meaning separately.	More workers can still produce a wrong answer.

Case study

How can we produce a correct hourly report before the next batch arrives?

Case study: an hourly report is late and wrong

Count distinct posts per account within 60 minutes. Diagnose the incorrect count and the missed deadline.

Hourly input

Post events
P1, P2, P1 replay

Attachment
I1 belongs to P1

Account rows
U1, U1 repeat

One batch job

Read, join, count
No duplicate checks

90 minutes per batch

A report with two problems

U1: 6 rows

Incorrect post count

Late by 30 minutes

Deadline: 60 minutes

Remove confirmed duplicates before the join

Remove the replay of P1 and the repeated account row, then join two posts to one account record.

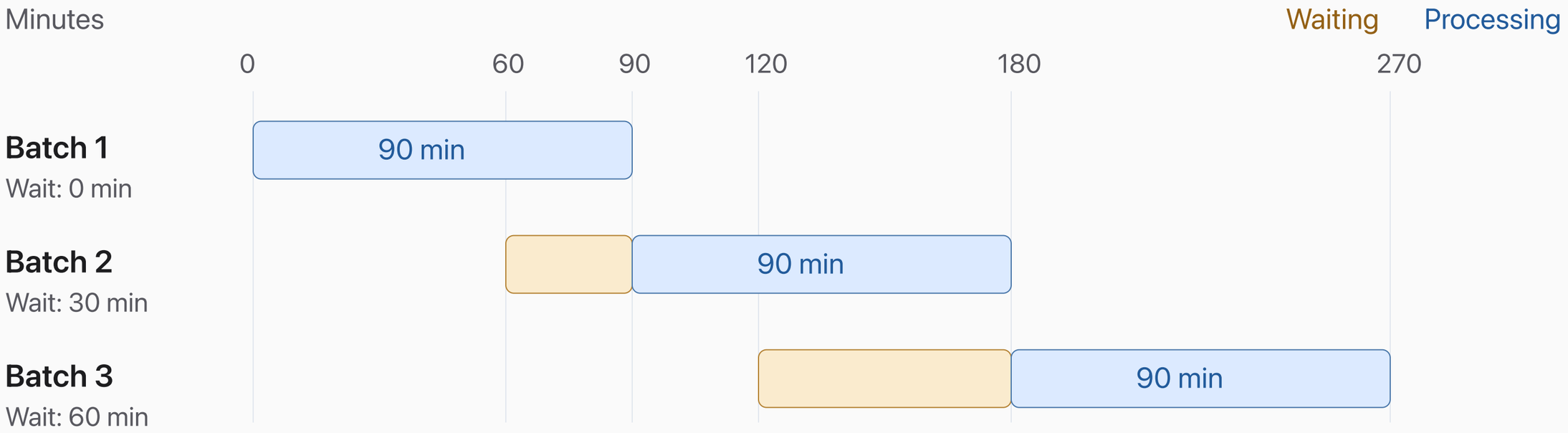
Before validation	Remove confirmed repeats	After validation
P1, P2, P1 replay	Keep one valid record for each source key.	**P1, P2**
U1, U1 repeat	Keep one valid record for each source key.	**U1**

Naive join: 3 × 2 = 6 rows

Validated join: 2 × 1 = 2 rows

The queue grows when a batch takes too long

Batches arrive every 60 minutes, but each takes 90 minutes, so their waiting times grow from 0 to 30 to 60 minutes.



Arrival every 60 min < processing time of 90 min

Accept the design only after these checks pass

The report must show two posts for U1, resist replay, and finish within the 60-minute deadline.

Correct count

Reference: P1 and P2

Required result: U1 = 2

Replay safety

Send P1 again.

Required result: still 2

Deadline

Include all processing steps.

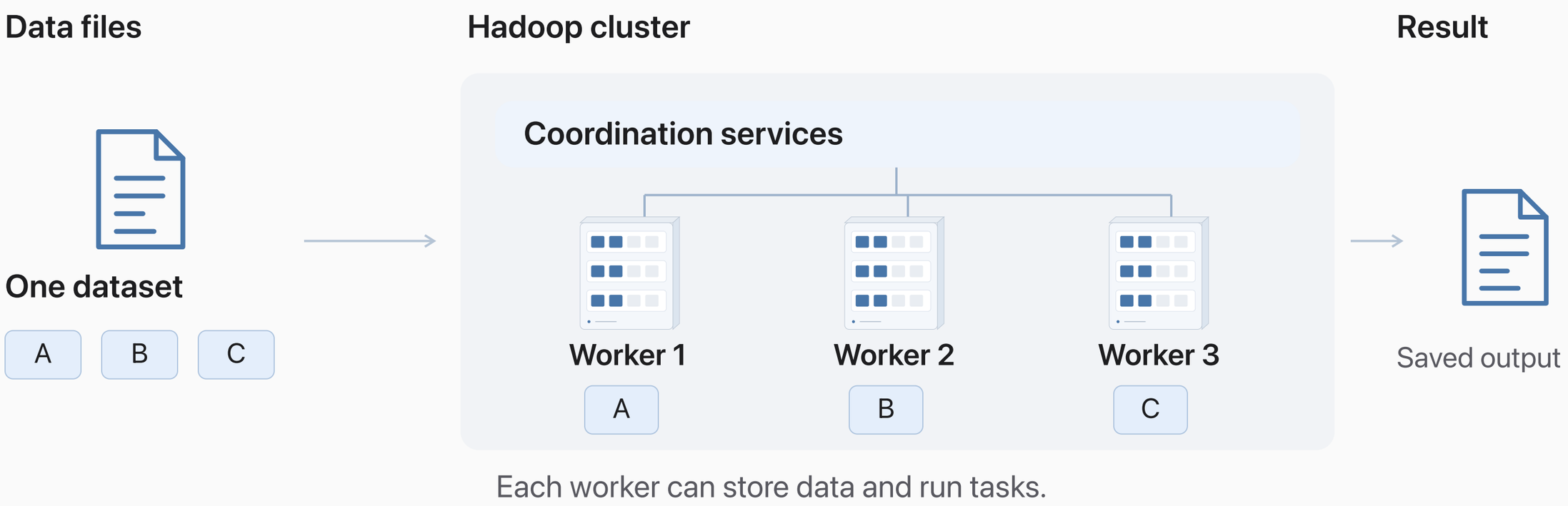
Finish within 60 minutes.

Hadoop architecture

How does Hadoop divide storage, resource management and processing?

Hadoop makes a cluster work together

Apache Hadoop is open-source software for distributed storage and processing across a cluster of computers.



Each worker can store data and run tasks.

Follow one job from stored data to a saved result

HDFS stores the input and output. YARN allocates resources. MapReduce tasks count the records.

01 HDFS

Store the input



4 records

R1 · R2 | R3 · R4

Two input blocks

02 YARN

Allocate resources



CPU
Memory

Resources for tasks
on worker machines

03 MapReduce

Process the records



Map task A **2**



Map task B **2**

Reduce: 2 + 2 = 4

04 HDFS

Store the result



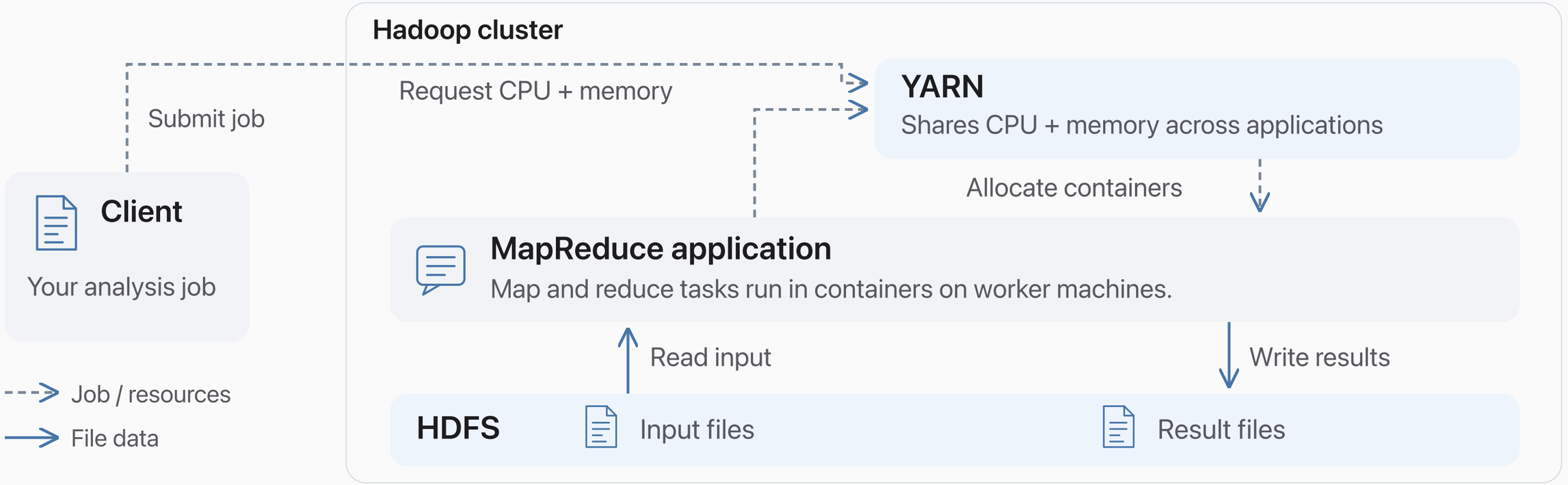
Record count

4

Store input → allocate resources → process records → save output

One job connects storage, resources and processing

YARN allocates resources. MapReduce tasks read data from HDFS, process it, and save results to HDFS.



Hadoop separates storage, resources and processing

These components work together. Each component solves a different problem.

HDFS

Distributed file storage

YARN

Cluster resource allocation

MapReduce

Batch processing framework

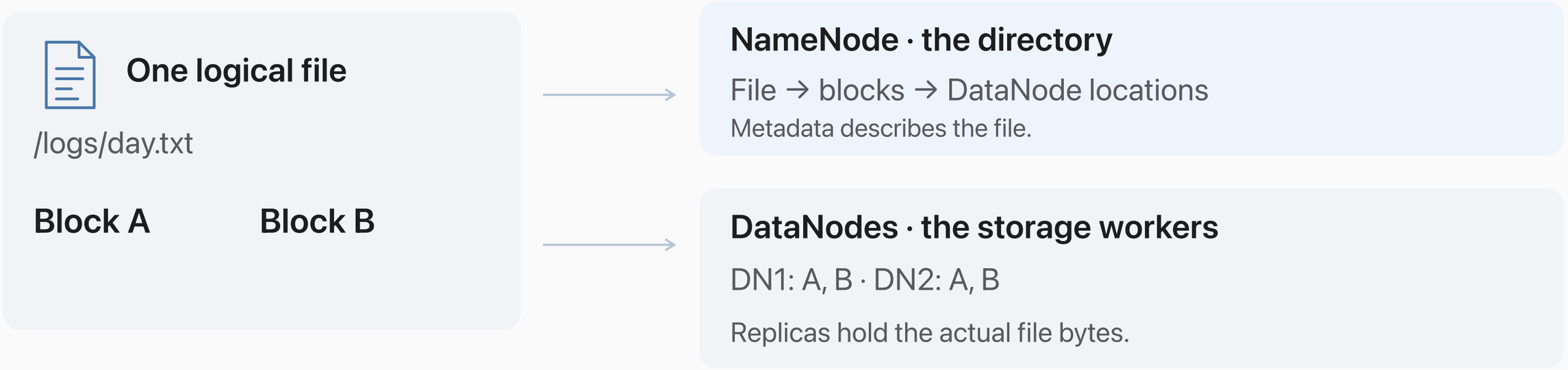
HDFS

Store blocks. Track locations. Recover replicas.

HDFS separates the file directory from its bytes

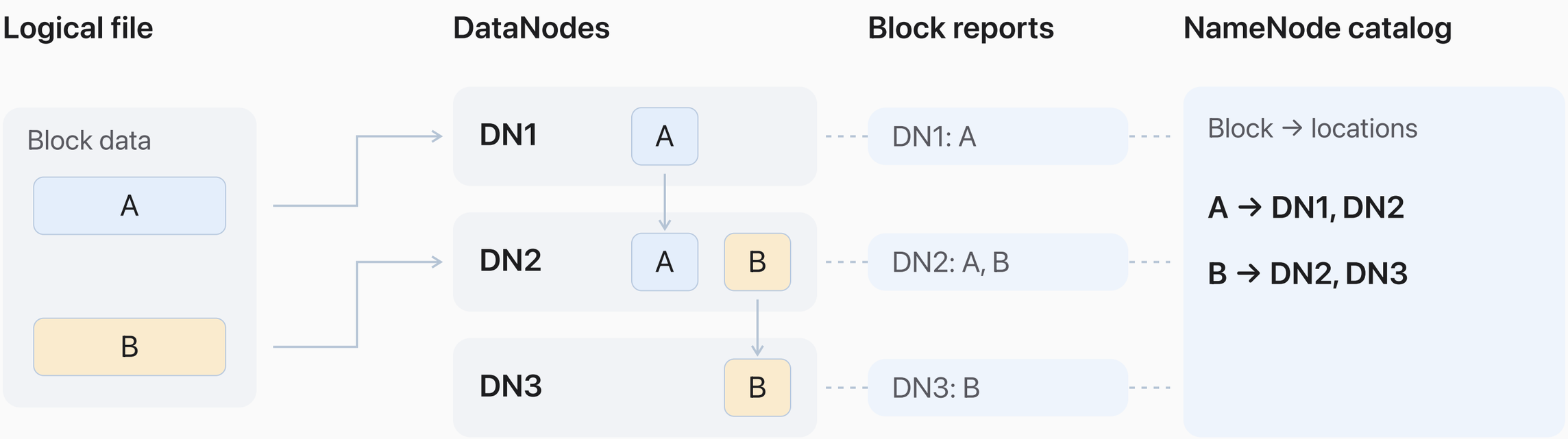
The NameNode tracks files and block locations; DataNodes store the block replicas.

Hadoop Distributed File System (HDFS)



HDFS splits a file into blocks and stores replicas

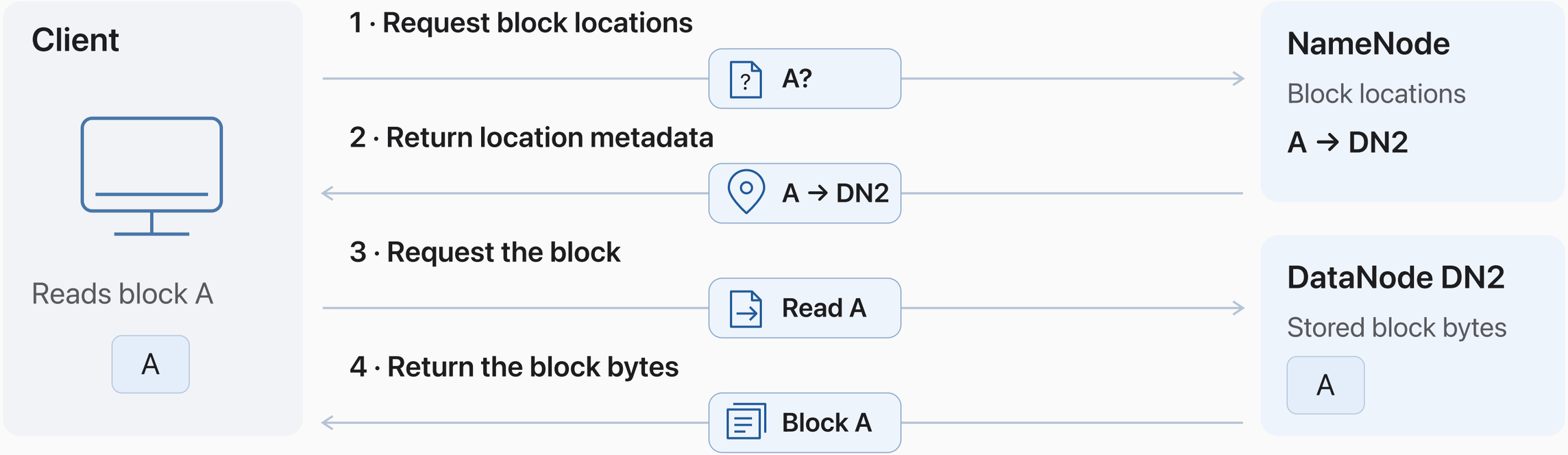
This example stores two copies of each block on DataNodes, while the NameNode records their locations.



Illustrative data transformation

Clients obtain metadata, then read block bytes

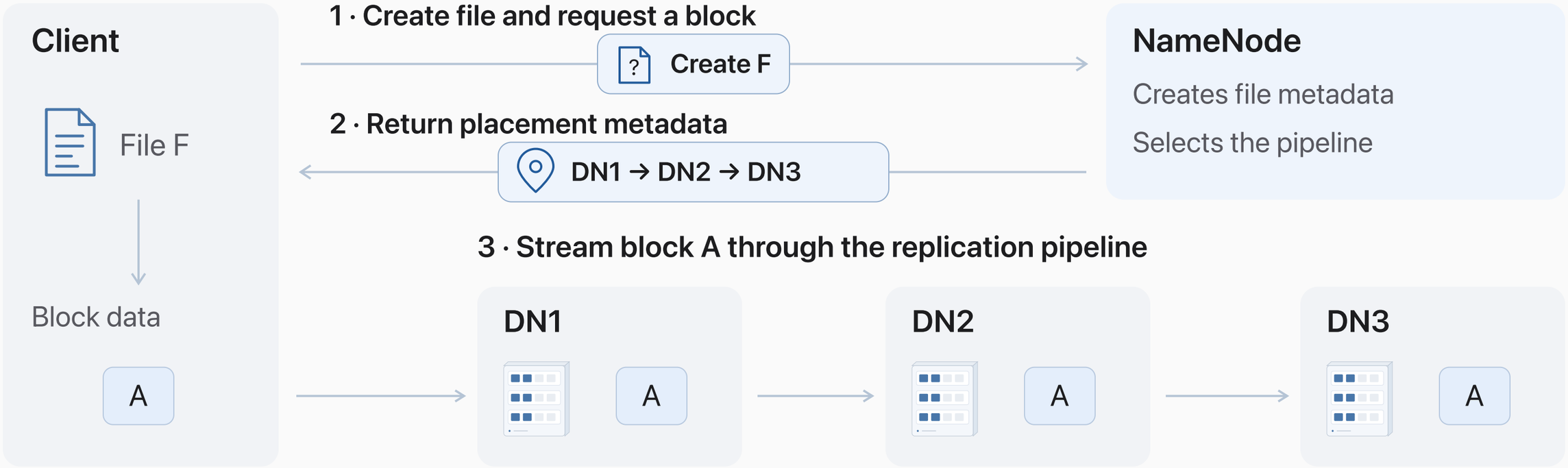
Clients get block locations from the NameNode, then read the data directly from DataNodes.



Illustrative data transformation

A write uses a replication pipeline

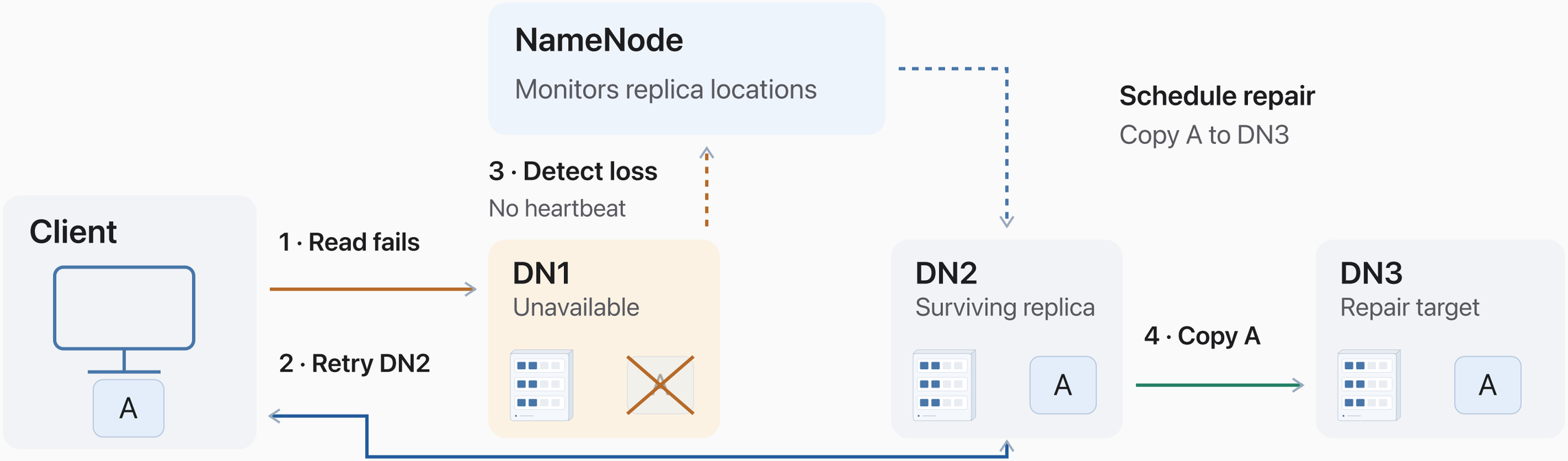
After the NameNode selects DataNodes, the client streams block data through the pipeline to create three replicas.



Illustrative data transformation

A failed replica changes the read path

The client reads a surviving replica while the NameNode arranges a new copy to restore replication.



Illustrative data transformation

HDFS supports large sequential workloads

HDFS suits large sequential reads. Combine small records into larger files to reduce file metadata overhead.

Good fit



Large files

Read many records in sequence.
Use for batch analysis.

Small-file pressure



Many tiny files

Each file needs metadata.
Small reads add overhead.

Design response



Combine records

Write larger analytics files.
Keep records accessible inside.

Case study: choose storage for three workloads

5 min · Your team must store these three workloads. Choose the best HDFS fit, explain the limits of the other two, and propose suitable storage designs.

A Daily log archives



20 files · 5 GB each

Scan all records each night.

Keep closed archives unchanged.

B Product images



10 million files · 50 KB each

Fetch one image per request.

Serve images to an online shop.

C Live user profiles



Small records · frequent edits

Read and update on each click.

Respond quickly to users.

Solution: use HDFS for the log archives

Use HDFS for batch scans, object storage for image retrieval, and a database for frequent profile updates.

A Daily log archives



HDFS is a strong fit.

Large sequential batch reads

Few files limit metadata overhead.

B Product images



Use object storage for serving.

Tiny files add HDFS metadata.

For batch analysis, pack records.

C Live user profiles



Use a database for updates.

HDFS is not a record-update API.

Keep analytics exports separate.

HDFS revision: roles, paths and suitability

Explain both the mechanism and its consequence; a component name alone is incomplete.

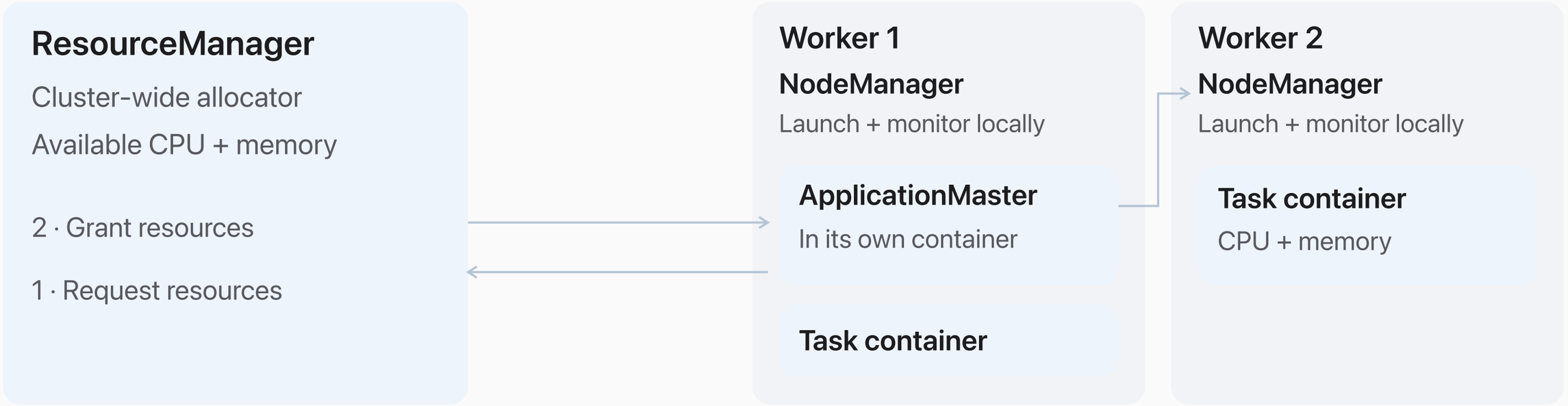
Question	Answer	Why it matters
Who stores what?	NameNode: metadata. DataNodes: block bytes.	Metadata lookup and data transfer are separate.
How does a read work?	Client asks NameNode, then reads a DataNode.	File bytes do not pass through the NameNode.
How does a write work?	NameNode selects nodes; client streams a pipeline.	Replicas store copies of the same block.
What if a replica fails?	Read a survivor; restore the missing replica.	Replication supports availability after node loss.
Which workload fits?	Large sequential scans; fewer larger files.	Tiny files add metadata and file-open overhead.

YARN

Allocate resources. Coordinate applications.

YARN shares cluster resources among applications

The ResourceManager grants resources; each ApplicationMaster coordinates its job through worker NodeManagers.



Yet Another Resource Negotiator

3 · Ask NodeManagers to launch tasks in granted containers

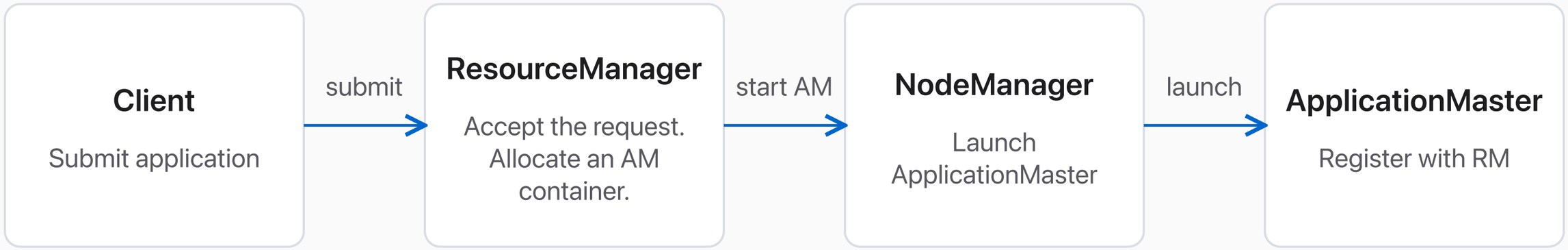
YARN revision: actor, scope and responsibility

YARN allocates resources; the application's tasks perform the calculation.

Actor	Scope	Responsibility
ResourceManager	Whole cluster	Grants resources according to capacity and policy.
ApplicationMaster	One application	Requests resources; coordinates and tracks work.
NodeManager	One worker machine	Launches and monitors local containers.
Container	One resource allocation	CPU and memory for a process; not necessarily Docker.

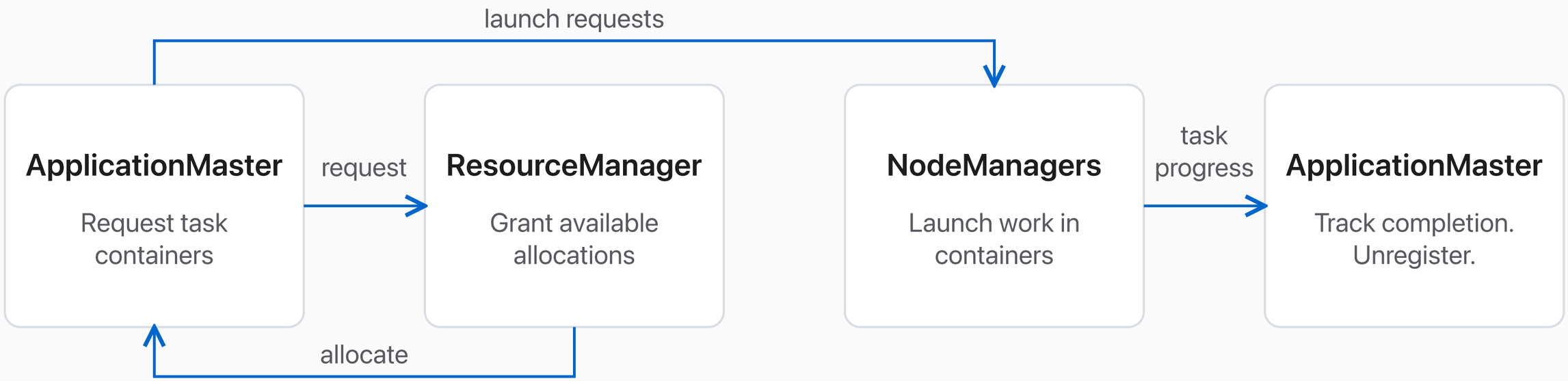
First, YARN starts the application coordinator

The ApplicationMaster needs its own container before requesting task resources.



Then, the coordinator launches and tracks work

Tasks finish. YARN releases their resources. The application unregisters.



Self-check: who allocates, launches and calculates?

Answer before moving to the worked explanation.

Situation	Your task
A word-count application needs two task containers.	Name the actor that requests them and the actor that grants them.
The granted containers are on two worker machines.	Who launches the work? Who tracks application progress?
A reducer produces the final word counts.	Does the ResourceManager calculate those counts? Explain.

Worked answer: separate allocation from computation

An allocation enables computation; it is not the computation itself.

Action	Responsible actor	Reason
Request / grant	ApplicationMaster / ResourceManager	The application requests; the cluster allocator grants.
Launch / monitor locally	NodeManagers	Each manages containers on its own machine.
Track the application	ApplicationMaster	It coordinates the application’s tasks and progress.
Calculate counts	MapReduce task processes	YARN provides resources; tasks execute the logic.

MapReduce

Map records. Group keys. Reduce values.

A key chooses the group; a value supplies the data

For word counts, emit the word as the key and 1 for each occurrence.

Question: count occurrences of each word



One input line

car bus car

Three word occurrences

map

Emit = output a pair

(car , 1)

Key: car

Which group?

Value: 1

What gets added?

Same word → same group. Emit a pair for each occurrence.

MapReduce groups records with matching keys

Local records become intermediate pairs, then grouped results.

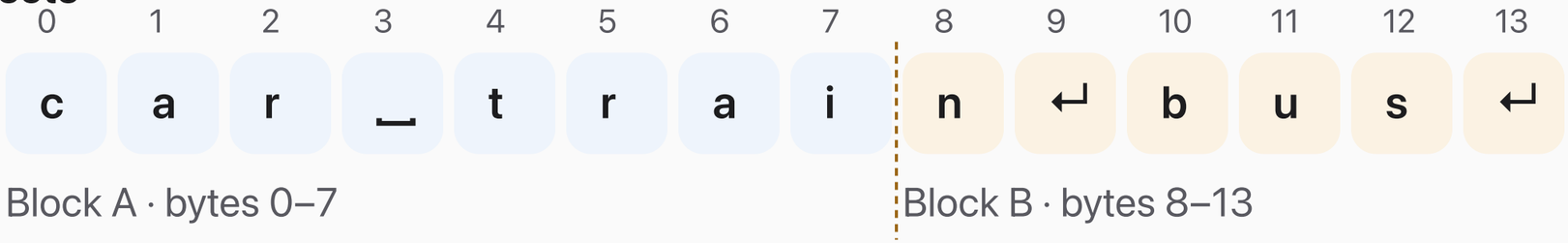
Input words	Map	Shuffle + sort	Reduce → output
car	(car, 1)	bus → [1]	(bus, 1)
bus	(bus, 1)		
car	(car, 1)	car → [1, 1]	(car, 2)

The two car occurrences stay distinct until their counts are added.

An input split is logical work, not a stored block

With text input, each RecordReader delivers complete lines even when a line crosses a split or block boundary.

One text file · byte offsets



Split 1 → mapper 1

Starts at byte 0; completes the first line.

0 → “car train”

Reads across the block boundary.

Split 2 → mapper 2

Starts at byte 8; skips the partial line.

10 → “bus”

Starts its record at the next full line.

Mappers emit what the next stage needs

The mapper splits each line into words and emits one (word, 1) pair for every occurrence.

Mapper code

```
map(offset, line):
    for word in tokenize(line):
        emit(word, 1)
```

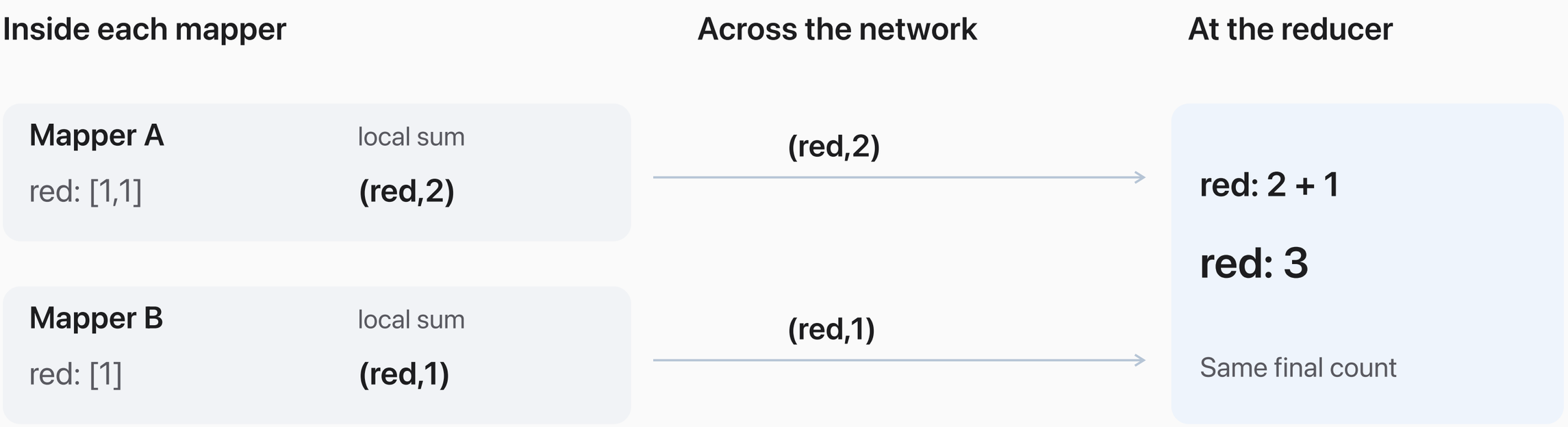
Input: 0 → “car bus car”

	Emitted pairs
car	(car, 1)
bus	(bus, 1)
car	(car, 1)

One input record → three loop iterations → three emitted pairs

A combiner sums locally before records travel

An optional combiner reduces transferred records; the reducer must produce the same answer without it.



For red: send 2 partial-count records instead of 3 occurrence records.

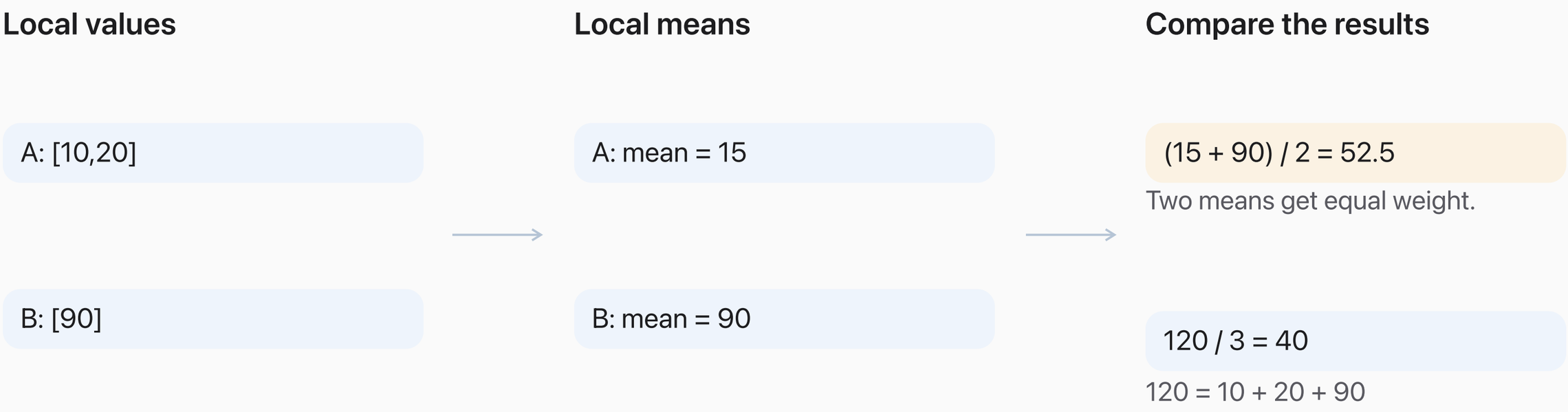
Combiner correctness: the result must stay the same

A combiner reduces transferred data only when local aggregation preserves the final answer.

Check	Requirement	Example
Optional execution	Correct with no combine, one combine or repeated combines.	Counts stay correct when partial counts are added.
Order and grouping	Merge partial results without changing the answer.	For sums: (1 + 1) + 1 = 1 + (1 + 1).
Compatible state	Keep a representation that can be merged again.	For an average, carry (sum, count).

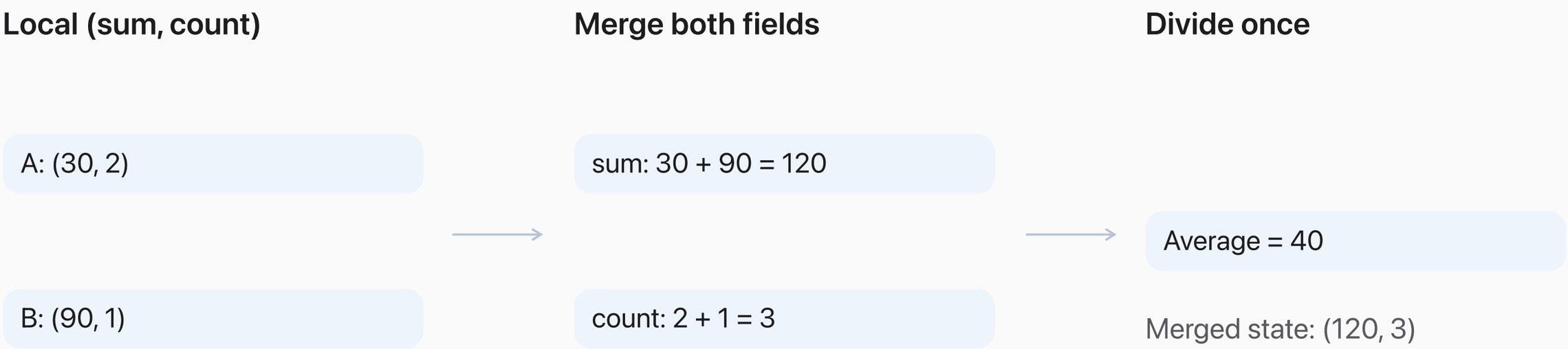
Averages of averages can be wrong

Unequal group sizes break a naive average-of-averages.



Keep the sum and count until the final division

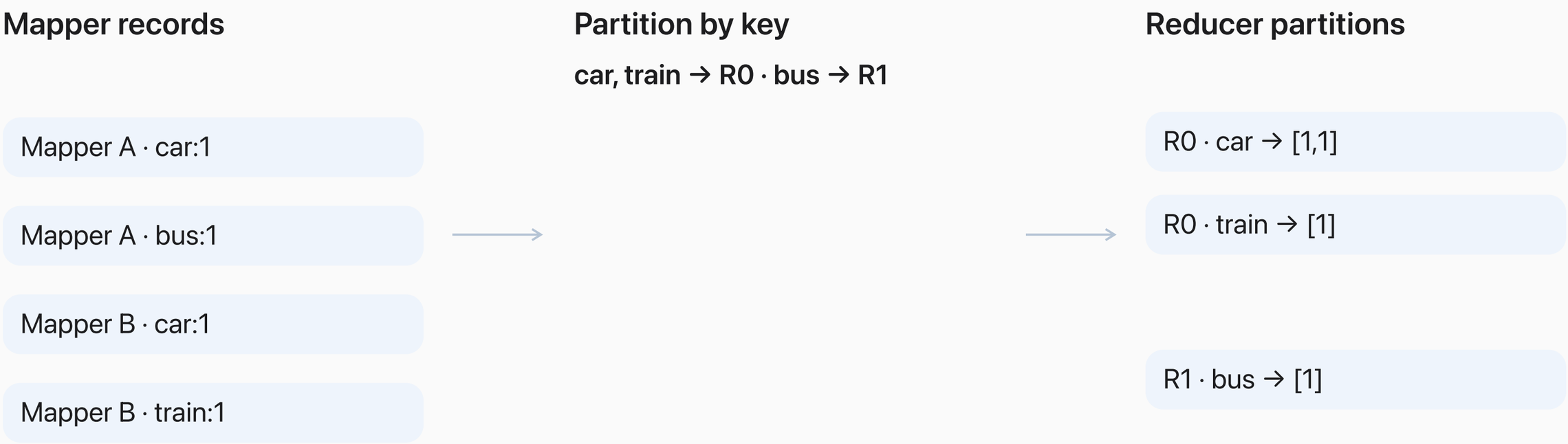
Merge the sums and counts. Calculate the average once at the end.



Conceptual process loop

A partitioner routes many keys to each reducer

Equal keys use the same reducer. Different keys may also use that reducer.



Reducer tasks iterate over grouped keys

One reducer task processes both keys and writes both records to one part-file.

Reducer 0 · grouped input	One reduce call per key	One output part-file
car → [1,1]	reduce(car): 1 + 1 = 2	(car,2)
train → [1]	reduce(train): 1	(train,1)

Conceptual process loop

01 / ACTIVITY

Trace six records through two reducer tasks

Count keys and tasks separately.

5 min · Show grouped input and output for R0 and R1. How many reducer tasks run?

Mapper A

red:1, blue:1, red:1

Mapper B

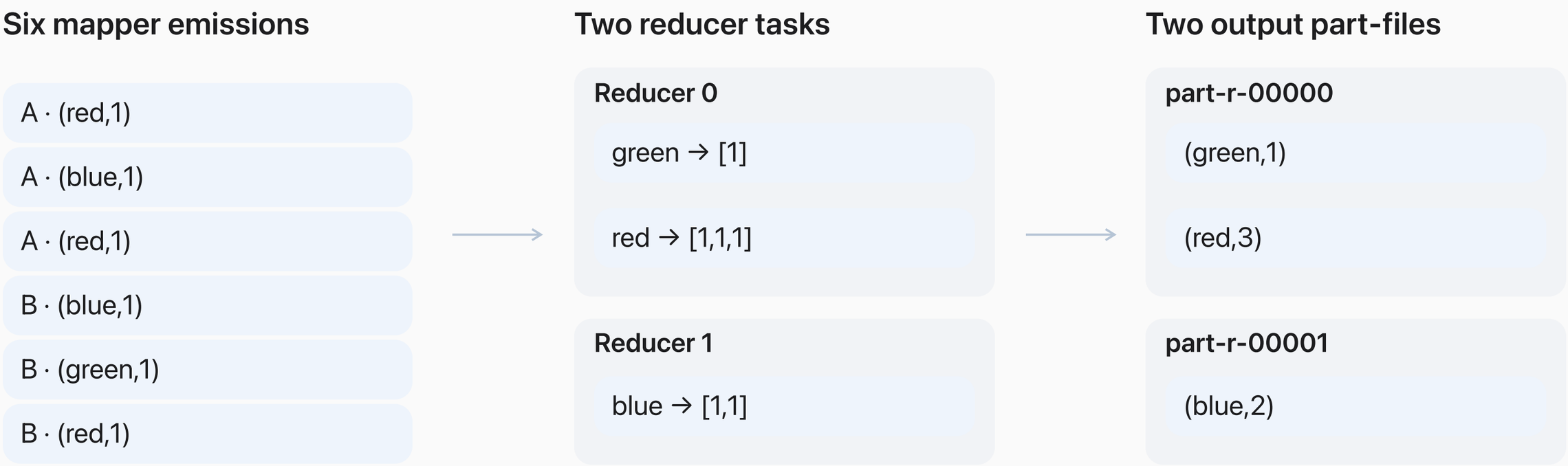
blue:1, green:1, red:1

Partition rule

red/green → R0 · blue → R1

Two reducer tasks handle three distinct keys

Six emitted records form three key groups; two reducer tasks write three totals into two part-files.



MapReduce revision: keep these quantities separate

Trace records first, then count key groups, reducer tasks and output files separately.

Concept	Meaning	Consequence
Block / input split	Stored bytes / logical input work.	A RecordReader handles complete records across boundaries.
Map / combine	Emit pairs / optionally aggregate locally.	A combiner must not change the final result.
Partition / group	Choose reducer / gather equal keys.	Equal keys go together; one reducer can handle many keys.
Task / reduce call	One running reducer / one grouped key.	Three keys need not mean three reducer tasks.
Output record / part-file	One emitted result / one task’s output file.	In our standard text-output examples, two reducers write two part-files.

Execution trade-offs

What does MapReduce gain and lose by writing intermediate results to disk?

Separate shuffle storage from output between jobs

Map output uses local storage for shuffle; a later job reads the previous job's saved output.

Within one job

Map output

Buffered, then spilled locally

→

Shuffle

Reducers fetch partitions

→

Reduce

Read, merge and calculate

Between jobs

Job 1 result

Write output to HDFS

→

Stored dataset

Survives task completion

→

Job 2 input

Read the dataset again

Disk-based execution: benefits and costs

Distinguish temporary shuffle storage from saved job output when explaining the trade-off.

Storage	Benefit	Cost or limitation
Local map output	Buffers and spills support data larger than memory.	Disk I/O and network shuffle add work.
Saved output between jobs	A later job can reuse the stored dataset.	Writing and rereading adds I/O and latency.
Failure distinction	Lost map output can be recomputed by rerunning its task.	Local shuffle data is not an HDFS-replicated final result.

Hadoop in the cloud

How do Hadoop responsibilities map to a managed cloud cluster?

EMR revision: three node roles

Choose a node type by the capacity it adds and the responsibilities it holds.

Node type	Role	Consequence
Primary	Coordinates cluster services.	Coordination differs from worker computation.
Core	Runs tasks and HDFS DataNodes.	Removal affects both compute and stored replicas.
Task	Adds task-processing capacity without HDFS DataNodes.	More compute does not add HDFS storage.

Cluster lifetime and data lifetime are different

Save and verify needed results outside the EMR cluster before terminating its compute and HDFS storage.

EMR cluster

HDFS input + output

Usable while the cluster exists

Termination removes this copy.

Save

Independent storage

Approved durable location

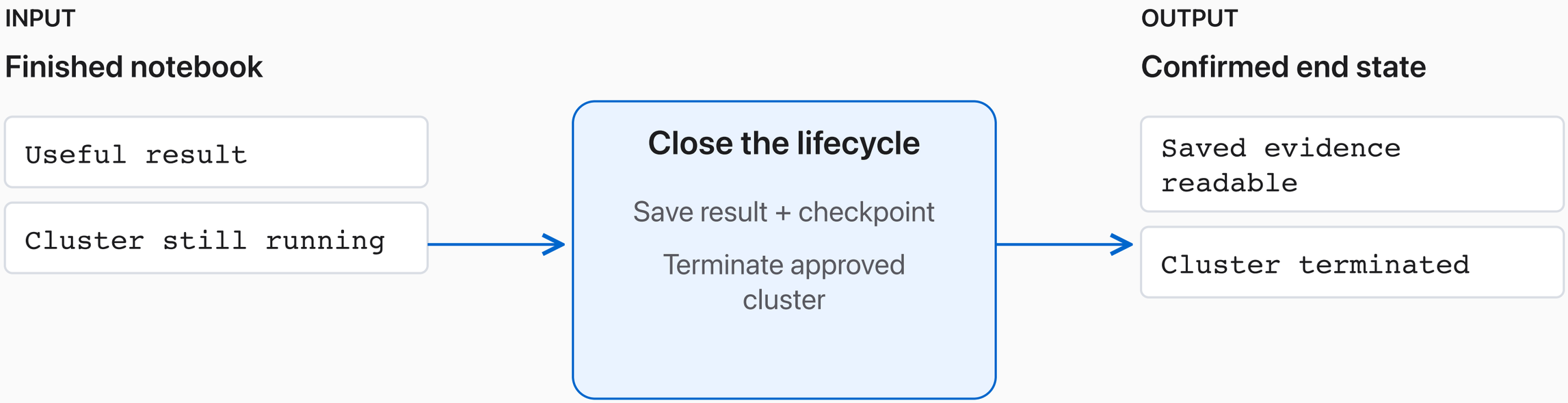
Example: Amazon Simple Storage Service (S3)

Read back the saved output.

Verify the independent copy and evidence before terminating the cluster.

Check saved evidence. Terminate the cluster.

Closing a notebook does not terminate its cluster.



Explain storage, scheduling and keyed aggregation

5 min · Input: “car bus car”; one reducer. Explain storage, resource allocation, emitted pairs, groups and final file contents. What must you verify before and after shutdown?

Storage

Client metadata lookup → block read

Scheduling

YARN resources → application work

Aggregation

Map → partition → group → reduce

Evidence

Checked output. Terminated cluster.

Model answer: trace the file to a checked result

Storage supplies bytes, YARN supplies resources, and MapReduce groups and totals the emitted pairs.

Input file: “car bus car”

1 Read

NameNode → locations
DataNodes → file bytes

2 Run

YARN → task resources
Map → (car,1), (bus,1), (car,1)

3 Group + sum

car → [1,1] → 2
bus → [1] → 1



One reducer → one part-file: bus 1, car 2

Read the actual output. Save it independently. Confirm cluster termination.

Practice: apply MapReduce to sales totals

Write your answer before viewing the solution. Values are illustrative.

Given	Question
Sales: (North,10), (South,7), (North,5).	What key and value should each mapper emit?
Partition rule: North → R0; South → R1.	Show grouped input and totals for each reducer.
Optional local aggregation.	Can a sum combiner be used? Explain why.
Standard text output; both reducers succeed.	How many key groups, reducer tasks and part-files?

Worked answer: sales totals by region

The key changes with the business question; the map–group–reduce reasoning stays the same.

Step	Result	Reason
Map	(North,10), (South,7), (North,5)	Region defines the group; amount is the value.
Group and reduce	R0: North [10,5] → 15; R1: South [7] → 7	Add values within each region.
Combine	Sum partial amounts, if the combiner runs.	Regrouping sums preserves each region’s total.
Count	2 key groups; 2 reducer tasks; 2 part-files	The supplied partition rule assigns one group to each task.

Conclusion: Hadoop essentials

Review the whole lecture: the problem, the system, the processing and the result.

1 · Big Data and scaling

Choose a remedy that addresses the actual cause of the problem.

Key point	What you should explain
Five Vs	Volume: amount. Velocity: arrival rate. Variety: formats. Veracity: trust. Value: useful outcome.
Big Data	The data and task strain available capacity; there is no universal size threshold.
Scale up / scale out	Add resources to one host / add hosts and divide the work.
Cost of distribution	Parallel work adds communication, coordination and partial-failure handling.
Correctness and capacity	Validate records to fix wrong results; measure bottlenecks to address slow results.

2 · Hadoop roles and HDFS storage

A replica is another copy of a block, not another unique input record.

Key point	What you should explain
Three responsibilities	HDFS stores files; YARN allocates resources; MapReduce processes records.
Metadata / bytes	NameNode tracks files, blocks and locations; DataNodes store block replicas.
Read / write	Read: get locations, then fetch bytes. Write: get placement, then stream a replica pipeline.
Replica failure	Read a surviving copy; restore replication. Copies improve availability after node loss.
Workload suitability	Large sequential scans fit HDFS. Tiny files add metadata overhead; frequent record updates need a suitable database.

3 · YARN resource management

Keep cluster-wide allocation, application coordination and local task execution separate.

Actor or step	What you should explain
ResourceManager	Allocates cluster resources; it does not execute the application’s calculation.
ApplicationMaster	Coordinates one application, requests containers and tracks task progress.
NodeManager	Launches and monitors containers on one worker machine.
Container	An allocation of resources such as CPU and memory; not necessarily Docker.
Application lifecycle	Submit → start AM → request task resources → launch and track work → release resources.

4 · MapReduce: input to output

Example: car, bus, car → (car,1), (bus,1), (car,1) → car:[1,1], bus:[1] → car:2, bus:1.

Stage or distinction	What you should explain
Block / split / record	Block: stored bytes. Split: logical mapper input. RecordReader delivers records.
Map	Emit key–value pairs. For word count: each occurrence emits (word, 1).
Partition and shuffle	Route equal keys to the same reducer; transfer, sort and group intermediate pairs.
Reduce	Process each grouped key and its values to emit final results.
Keys / tasks / files	One reducer can process several keys. In our standard text-output examples, each reducer writes one part-file.

5 · Combiners and execution trade-offs

A performance optimisation is valid only if it preserves the required result.

Key point	What you should explain
Optional combiner	Locally aggregate to reduce transfer; the answer must survive zero, one or repeated combines.
Safe partial results	Merging partial state must preserve the answer under regrouping and reordered inputs.
Average pitfall	[10,20] and [90]: mean of means = 52.5, but the true mean is 120 / 3 = 40.
Correct average state	Carry (sum, count): (30,2) + (90,1) = (120,3). Divide once at the end.
Disk distinction	Shuffle uses local map output; saved output can feed later jobs. Disk and network I/O add cost.

6 · EMR and a complete Hadoop answer

You should be able to explain the system and work through a new example without copying a diagram.

Key point	What you should explain
Primary / core / task	Primary coordinates. Core provides compute and HDFS storage. Task adds compute without HDFS DataNodes.
Cluster lifetime	Cluster HDFS is tied to the cluster; preserve required results outside it before termination.
Verified result	Read the saved output, check its meaning and verify the independent copy.
Finish the lab	Save required evidence and terminate the cluster. Closing a notebook does not stop the cluster.
Answer clearly	State the rule → trace the mechanism → show the result → justify the choice or trade-off.