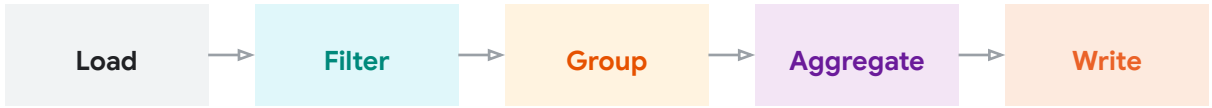
Authoring Your First Pipeline

[Raj Katakam] · BeamCollege 2026

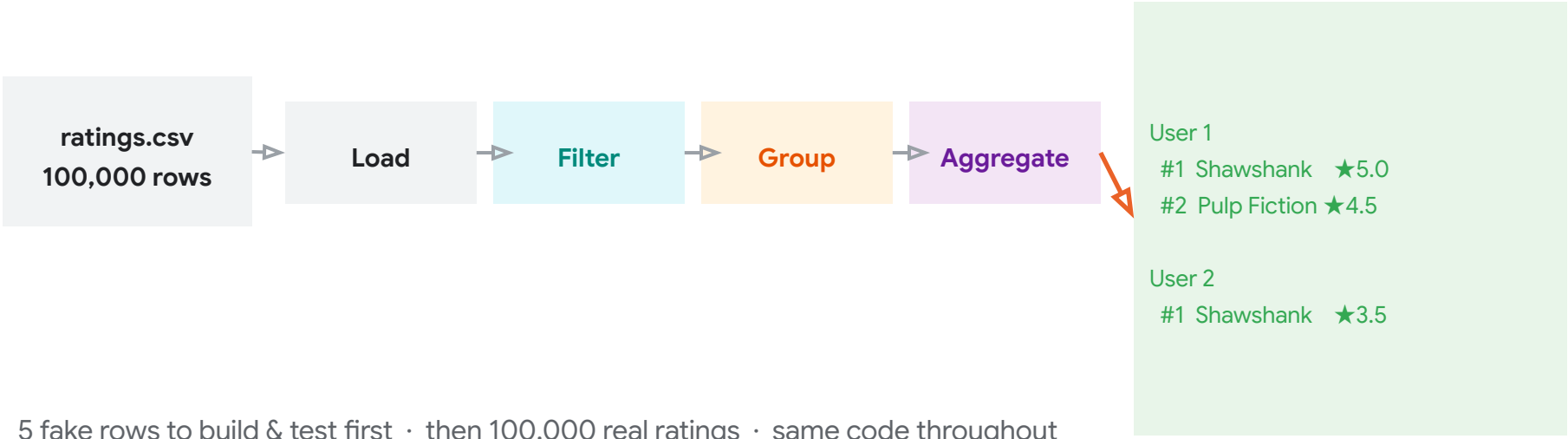
Authoring Your First Pipeline - Goals



1. Understand Apache Beam
what it is · what it unlocks

2. Build a real pipeline
MovieLens ratings · step by step

What We'll Build Today



5 fake rows to build & test first · then 100,000 real ratings · same code throughout

10 rows → a for loop works
group by user, sort, take top 5. Easy.

10 billion → one machine runs out of RAM
That's what Apache Beam solves.

What is Apache Beam?

A unified programming model for defining and executing data processing pipelines

Batch & Streaming

Any Language

Any Scale

One API — write once — batch or streaming — test locally — run at cloud scale

The Core Idea

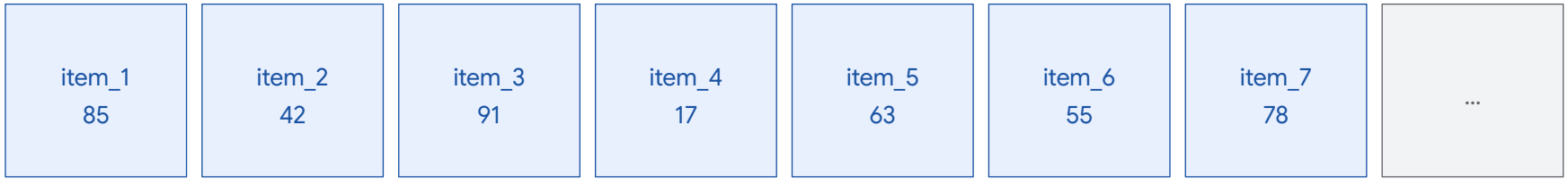
You describe WHAT to compute.

The runner figures out HOW.

Partitioning · Parallelism · Retries · Fault tolerance · Worker provisioning

PCollection

A distributed, immutable stream of elements



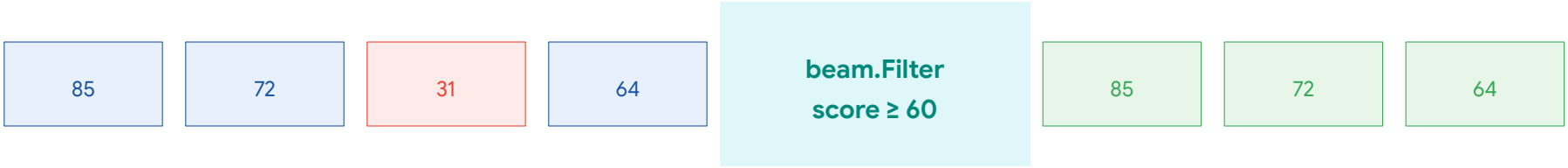
Distributed — the runner partitions it, you don't

Immutable — transforms produce new PCollections

beam.Create · ReadFromText · ReadFromKafka — all produce the same PCollection

PTransforms — Operations on Data

PCollection ——[PTransform]——▶ PCollection



Element-wise (Map · Filter)
Each element processed independently
Perfectly parallel — no coordination needed

Aggregating (GroupByKey)
Brings elements together by key
Requires shuffling data across workers

beam.Map — Transform Every Element

One element in. One element out. Always.


```
parse_csv(line)
CSV line → structured dict
```

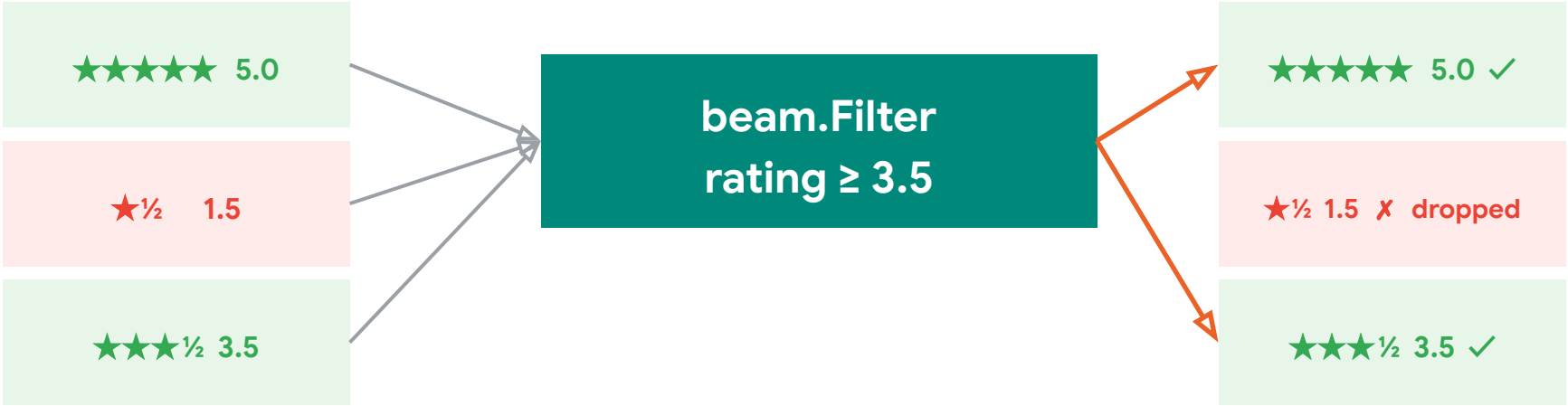
```
key_by_user(row)
row → (userId, row)
```

```
format_output(pair)
pair → output string
```

Use Map for: parsing · extracting · converting · reshaping — any 1-to-1 transformation

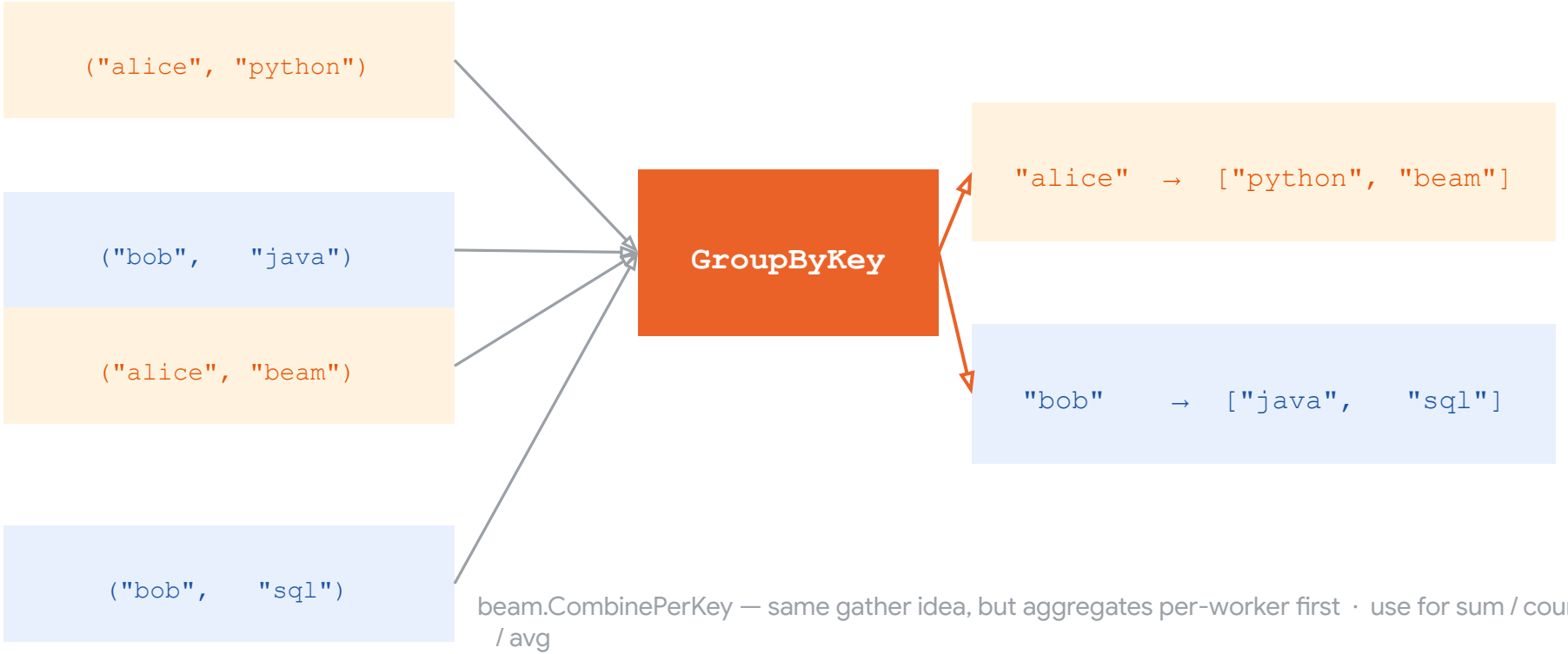
beam.Filter — Keep or Drop

One element in. Zero or one element out.



Use Filter for: removing nulls · enforcing valid ranges · applying quality thresholds · dropping incomplete records

beam.GroupByKey — Scatter → Gather



beam.CombinePerKey — same gather idea, but aggregates per-worker first · use for sum / count / avg

The Transform Zoo — There's a Lot More

We used three today. Beam ships many more — here's a quick map.

Element-Wise (each element independent)	
Map	1 element in → 1 element out
Filter	1 element in → 0 or 1 out
FlatMap	1 element in → 0, 1, or many out
ParDo	Full control: 0, 1, or many out
WithKeys	value → (key, value) pair
Keys / Values	Extract key or value from a KV pair

Aggregating (data shuffles across workers)	
GroupByKey	Group all values sharing the same key
CombinePerKey	Apply CombineFn per key (e.g. sum, max)
Count	Count all elements in the collection
Sum	Sum all elements
Max / Min	Find maximum or minimum element
Mean	Compute mean of all elements

Full reference: beam.apache.org/documentation/transforms/python/overview/

Our Dataset — MovieLens

100,000 ratings	600 users	9,000 movies

What a row looks like:

userId	Movie	Rating	Signal	Meaning
1	Shawshank Redemption	5.0	★★★★★	Genuinely loved it
1	Pulp Fiction	4.5	★★★★½	Really liked it
1	Usual Suspects	1.5	★½	Disliked
2	Shawshank Redemption	3.5	★★★½	Liked it
2	Toy Story	2.0	★★	Watched, meh

The question: for each user, what are their top-5 liked movies?

How We'll Build This

beam.Create
5 fake rows — instant, no setup

▽

TestPipeline
runs locally in milliseconds

▽

assert_that
checks actual output vs expected

▽

GREEN ✓
step verified — move to next

Why Beam makes this natural

beam.Create instant in-memory test data

TestPipeline no infra, runs in pytest

DirectRunner millisecond feedback loop

Same API test data = real data shape

Build with 5 fake rows ⟶ instant green. Swap beam.Create ⟶ ReadFromText when done. Same tests still pass.

The Pipeline — Built & Tested

All steps verified in TestPipeline · DirectRunner · < 2 seconds

5 rows loaded ✓

3 rows survive filter ✓

2 user groups ✓

Ranked output correct ✓

Scale Up — One Line Change

beam.Create → ReadFromText · same pipeline · real data

BEFORE

beam.Create [5 fake rows]

User	Movie	★
1	Shawshank	5.0
1	Pulp Fiction	4.5
1	Usual Susp.	1.5
2	Shawshank	3.5
2	Toy Story	2.0

Tests: PASSED ✓

AFTER

ReadFromText [ratings.csv]

ratings.csv

100,000 rows
600 users · 9,000 movies

parse_csv(line) → same dict shape as beam.Create

parse_csv produces the same dict shape as beam.Create → downstream transforms see no difference
The pipeline logic is identical. The TestPipeline tests stay as unit tests. Real data runs without them.

One Flag → Any Runner

LOCAL DEV

DirectRunner

laptop · milliseconds · no setup
Development & testing

→

CLOUD SCALE

DataflowRunner · Google Cloud · fully managed

SparkRunner · Apache Spark · your cluster

FlinkRunner · Apache Flink · streaming-first

```
--runner=DataflowRunner  ·  --runner=SparkRunner  ·  --runner=FlinkRunner
```

What any cloud runner handles for you:

- Worker provisioning
- Autoscaling
- Fault tolerance
- Monitoring UI

Test Pipeline

Full Pipeline

What You Now Know

Beam = WHAT, runner = HOW	describe logic, let the runner exec
PCollection	distributed, immutable, runner-ag
PTransform	composable operations on PCollections
Build locally first	beam.Create + TestPipeline — verified in 2 s
One line ⟶ real data	swap beam.Create for ReadFromText
One flag ⟶ cloud scale	--runner=<RunnerName> · same pipeline, any cloud

Natural next steps:

beam.RunInference	add a model
ReadFromKafka	go streaming
Beam YAML	config-driven

Thank you!

Questions?