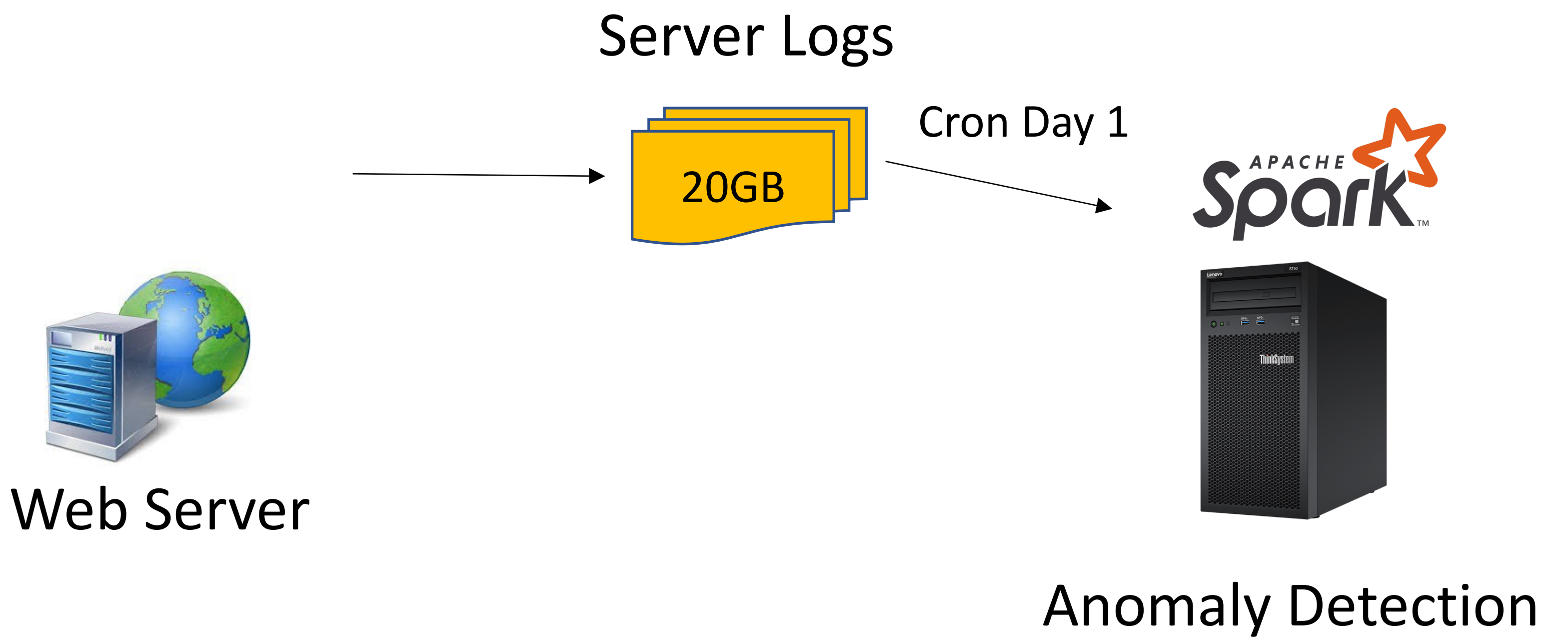
PerfDebug: Performance Debugging of Computation Skew in Dataflow Systems

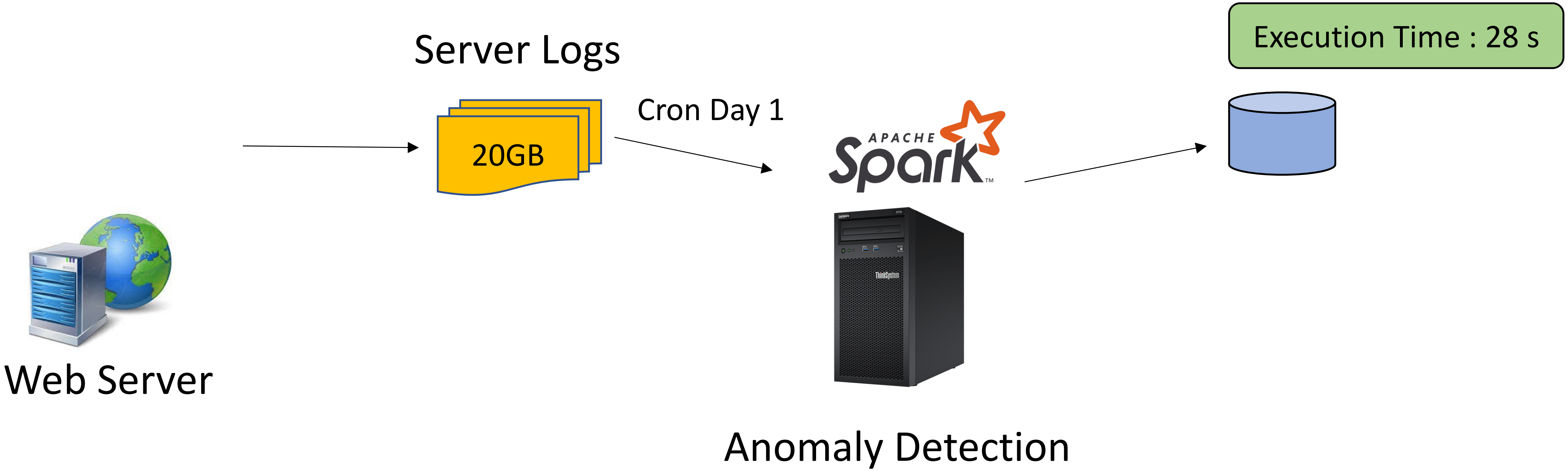
Jason Teoh, Muhammad Ali Gulzar, Harry Xu, Miryung Kim

University of California, Los Angeles

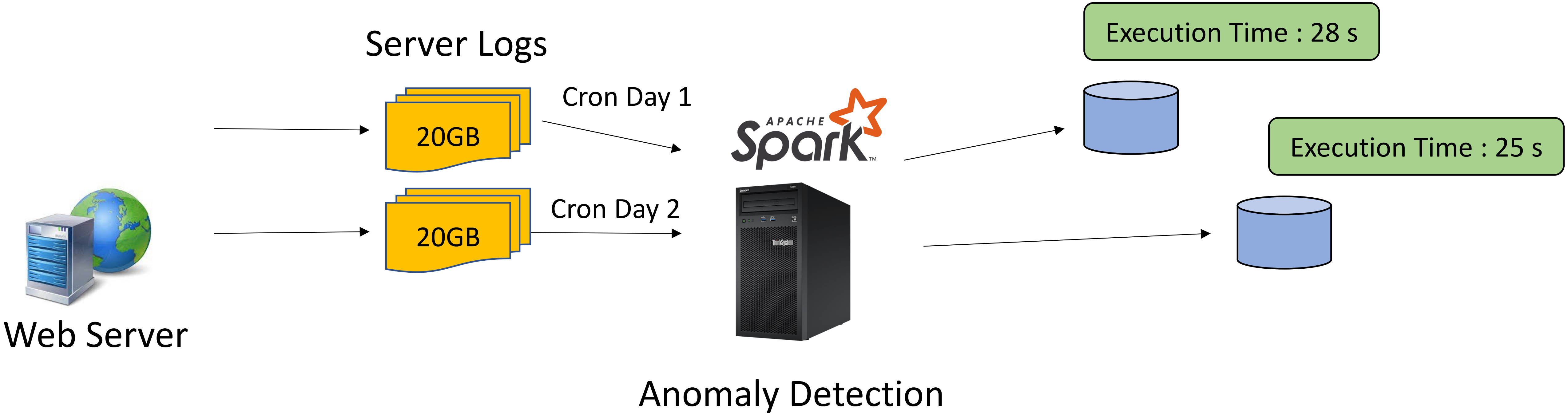
Motivating Example



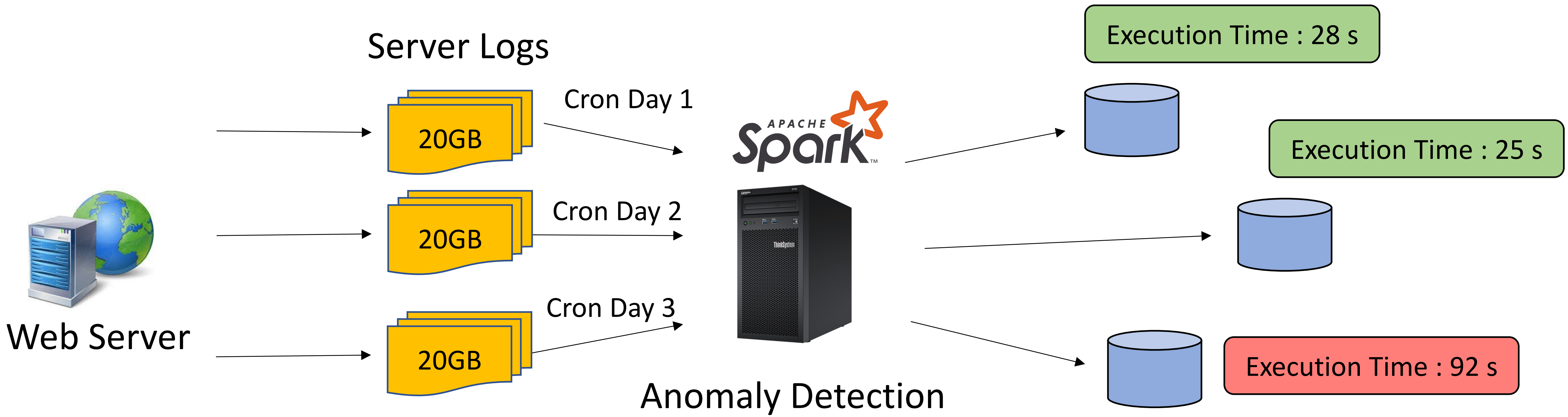
Motivating Example



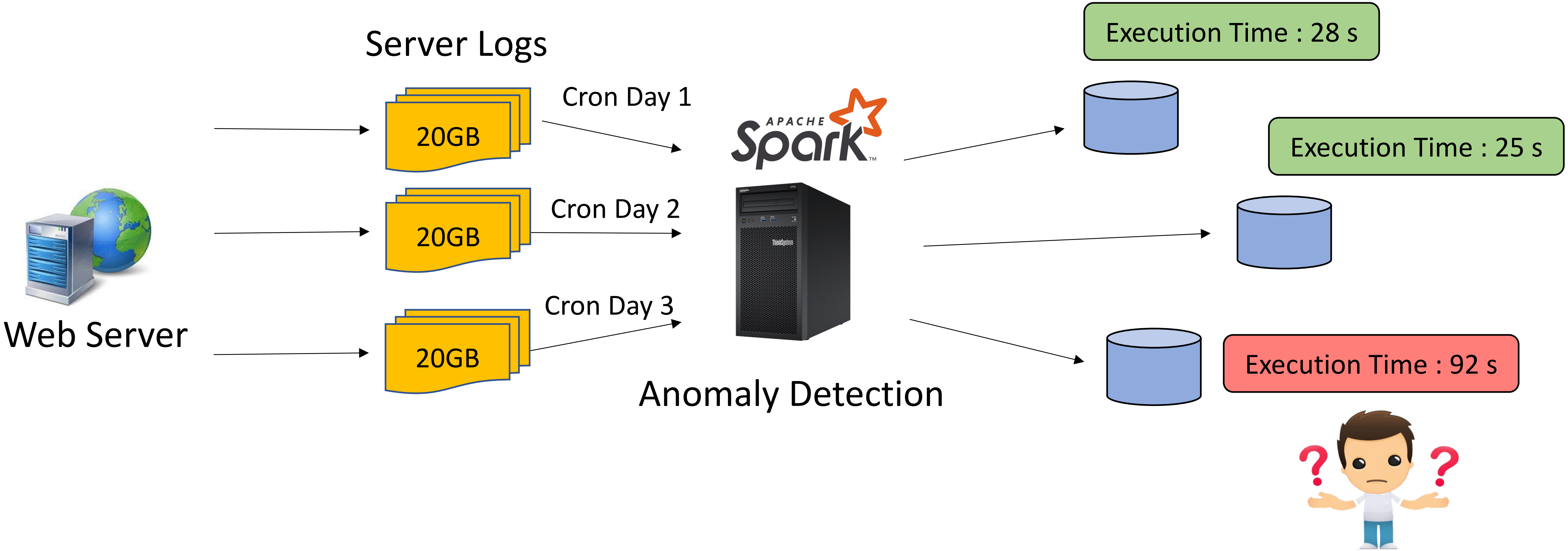
Motivating Example



Motivating Example

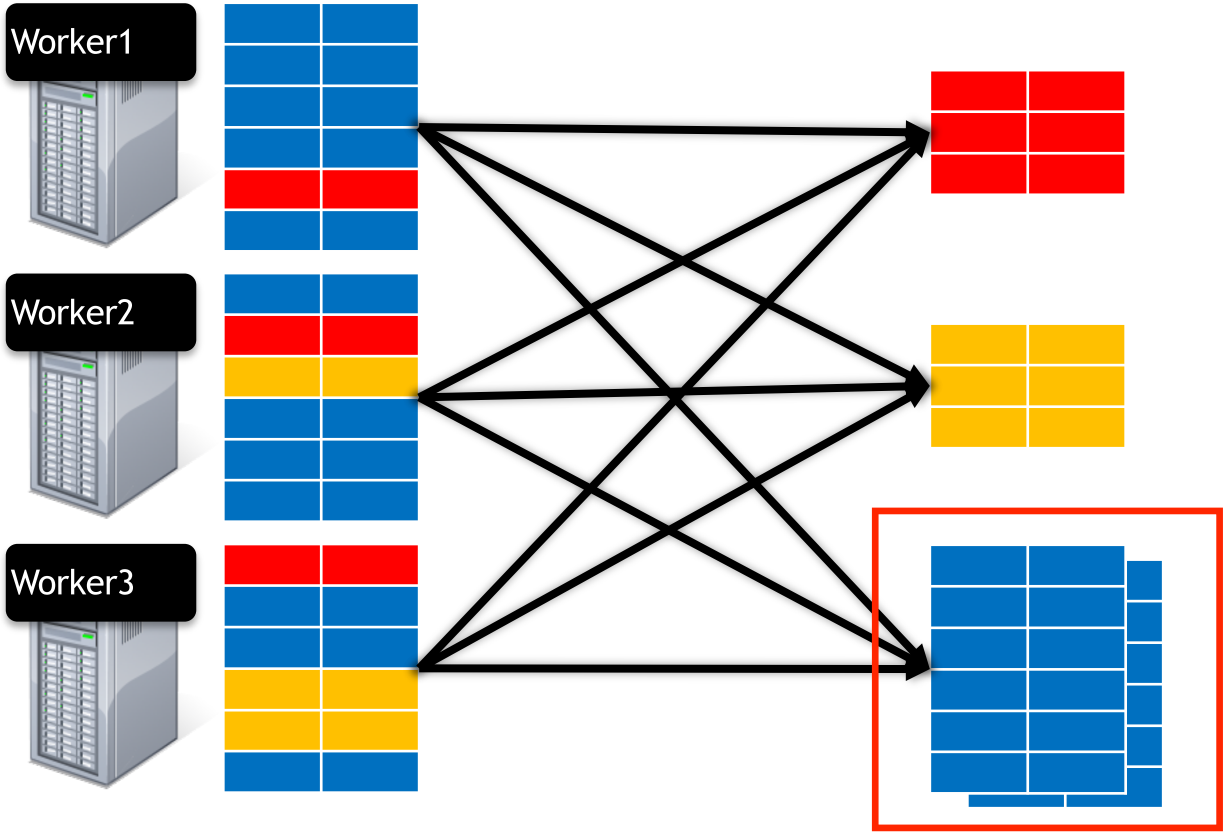


Motivating Example



Motivating Example

Data Skew in Distributed Processing



*Uneven distribution of **data** across partitions, tasks, or workers can lead to performance delays.*

Computation Skew

Term
Hello World
Big Data
Debugging
PerfDebug

User-defined function

```
commonDefs = {
  "Hello World": ...,
  "Big Data": ...,
  "Debugging": ...,
  ...
}

if (commonDefs.contains(term)) {
  return commonDefs.get(term)
} else {
  r = new RedisClient(…)
  return r.get(term)
}
```

Term	Latency
Hello World	2 ms
Big Data	1 ms
Debugging	3 ms
PerfDebug	**442 ms**

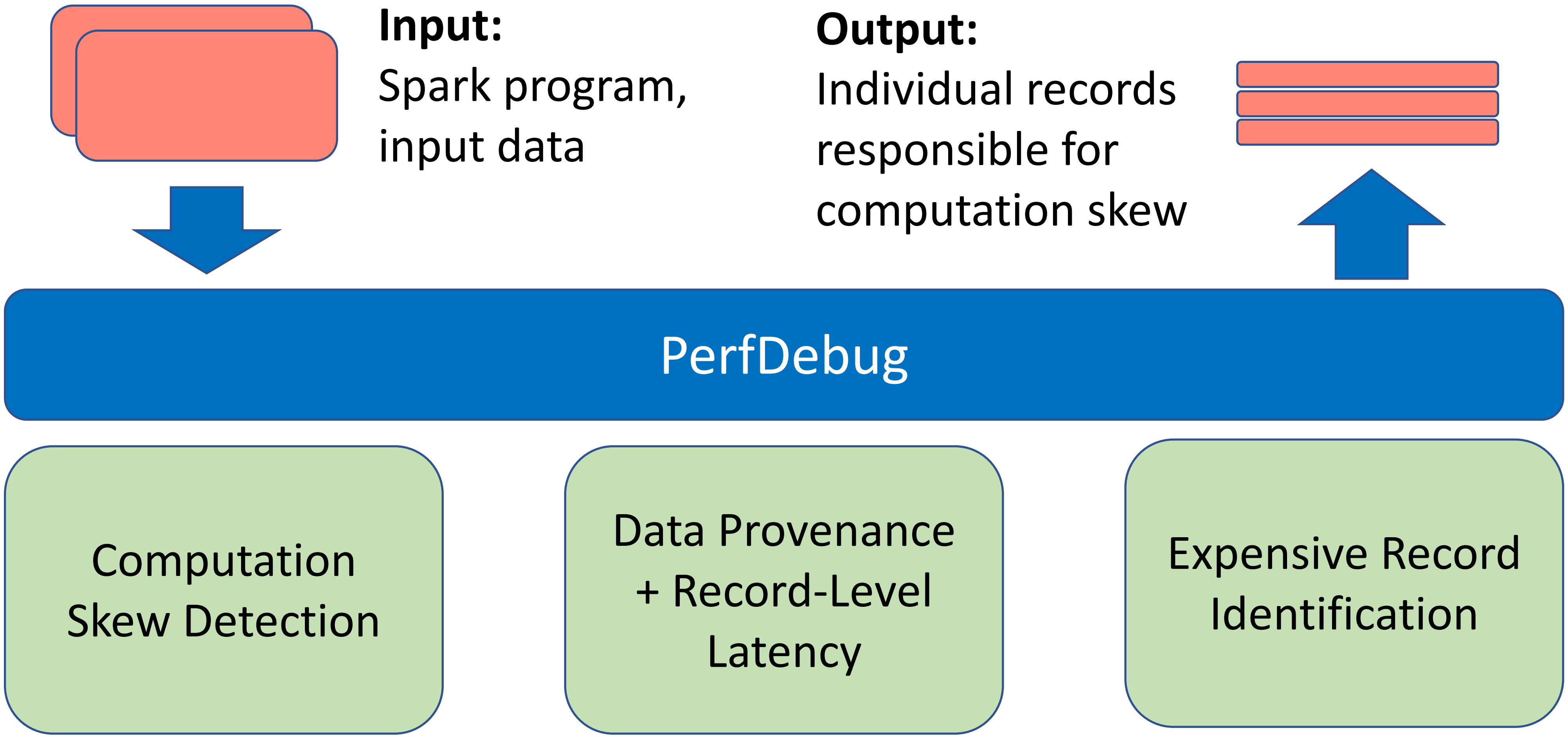
*Uneven distribution of **computation** due to interactions between data and application code.*

Computation Skew

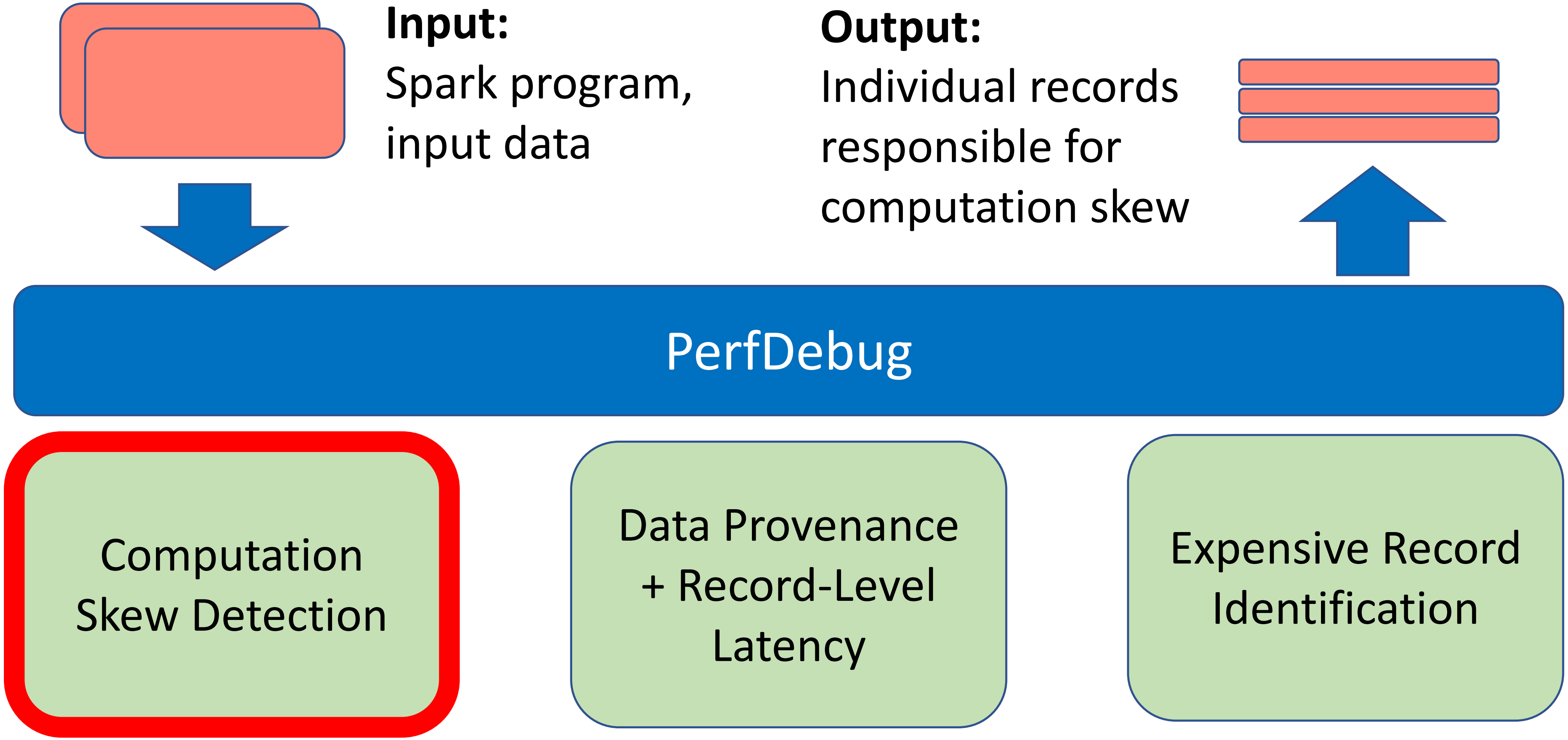
Why is it challenging?

- Requires insight on how application code interacts with data.
- Occurs across multiple stages.
- Affected applications are inherently expensive to run.
- Isolating individual records that impact performance is difficult with existing tools.

Performance Debugging of Computation Skew



PerfDebug Approach



Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

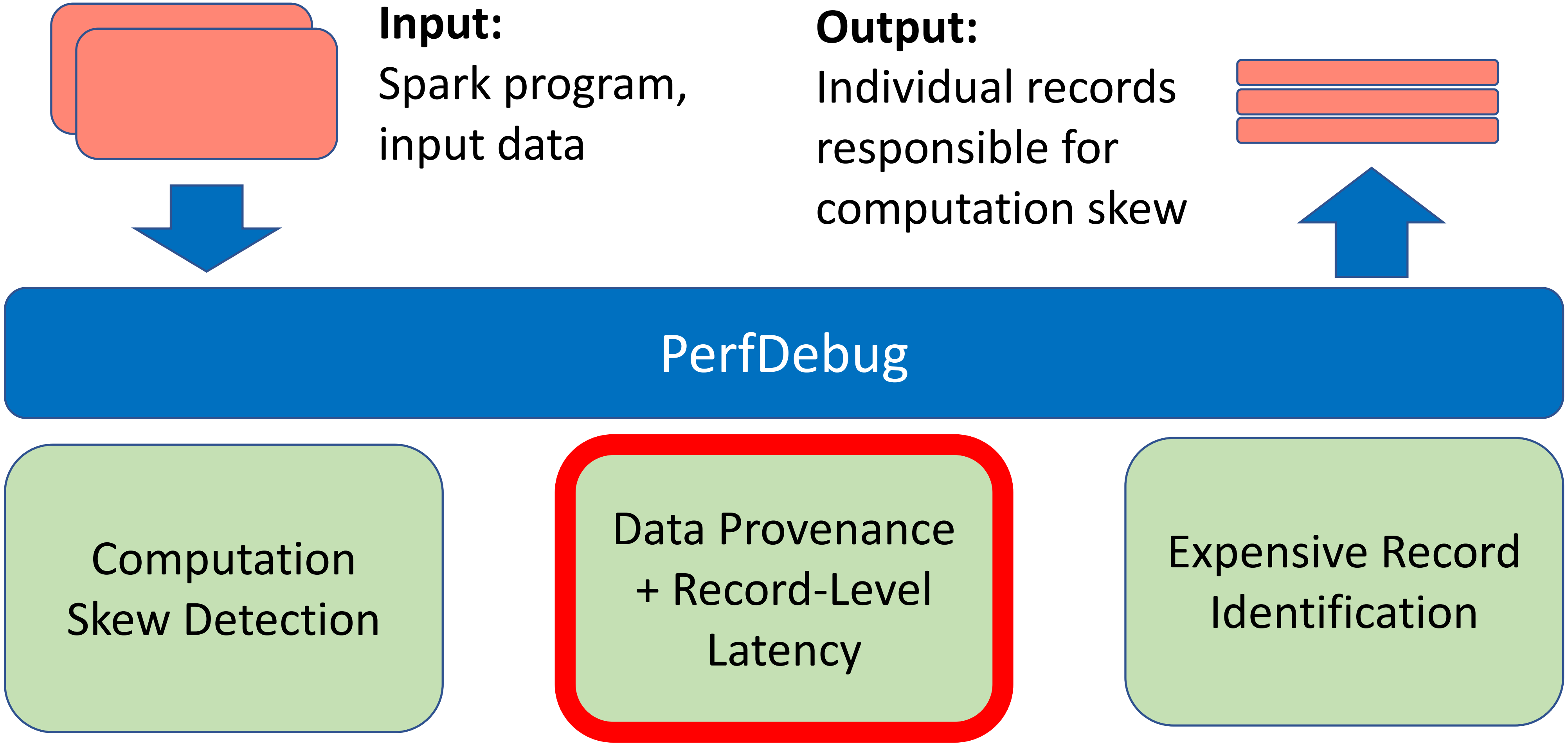
Computation Skew Detection

- PerfDebug monitors task-level metrics such as latency, garbage collection, and serialization using SparkListener API.
- If potential computation skew is found, rerun the user program in debugging mode to collect additional information.

APACHE Spark 2.1.1 Jobs Stages Storage Environment Executors

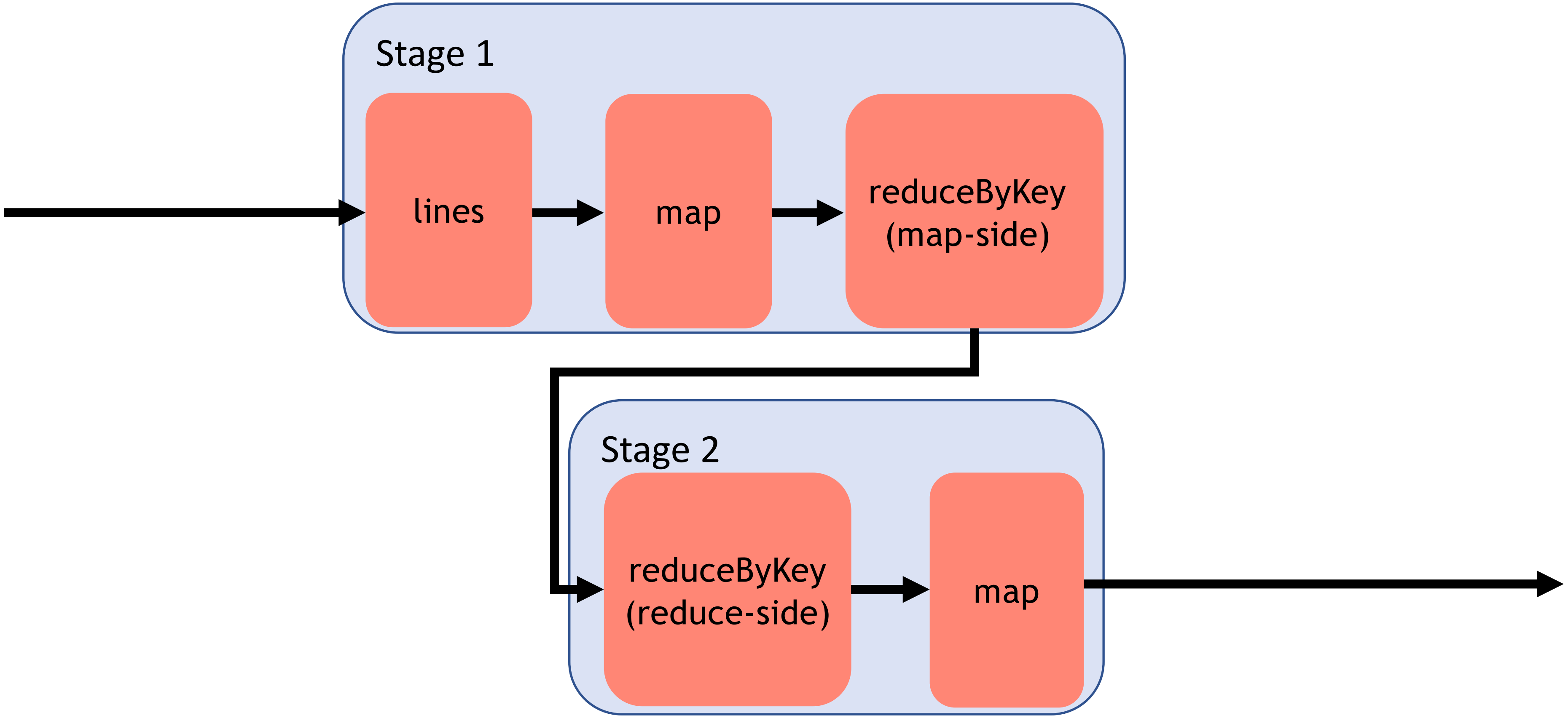
Index	ID	Executor ID / Host	Duration ▾	GC Time	Input Size / Records
33	33	8 / 131.179.96.204	1.2 min	7 s	128.0 MB / 17793
34	34	1 / 131.179.96.211	51 s	11 s	128.0 MB / 1
35	35	5/ 131.179.96.212	44s	3 s	128.0 MB / 1
25	25	5 / 131.179.96.212	38 s	2 s	128.0 MB / 33602
36	36	9 / 131.179.96.206	36 s	4 s	128.0 MB / 1
130	130	1 / 131.179.96.211	36 s	9 s	128.0 MB / 33505
37	37	6 / 131.179.96.203	35s	4 s	128.0 MB / 1
22	22	3 / 131.179.96.209	35 s	2 s	128.0 MB / 33564

PerfDebug Approach



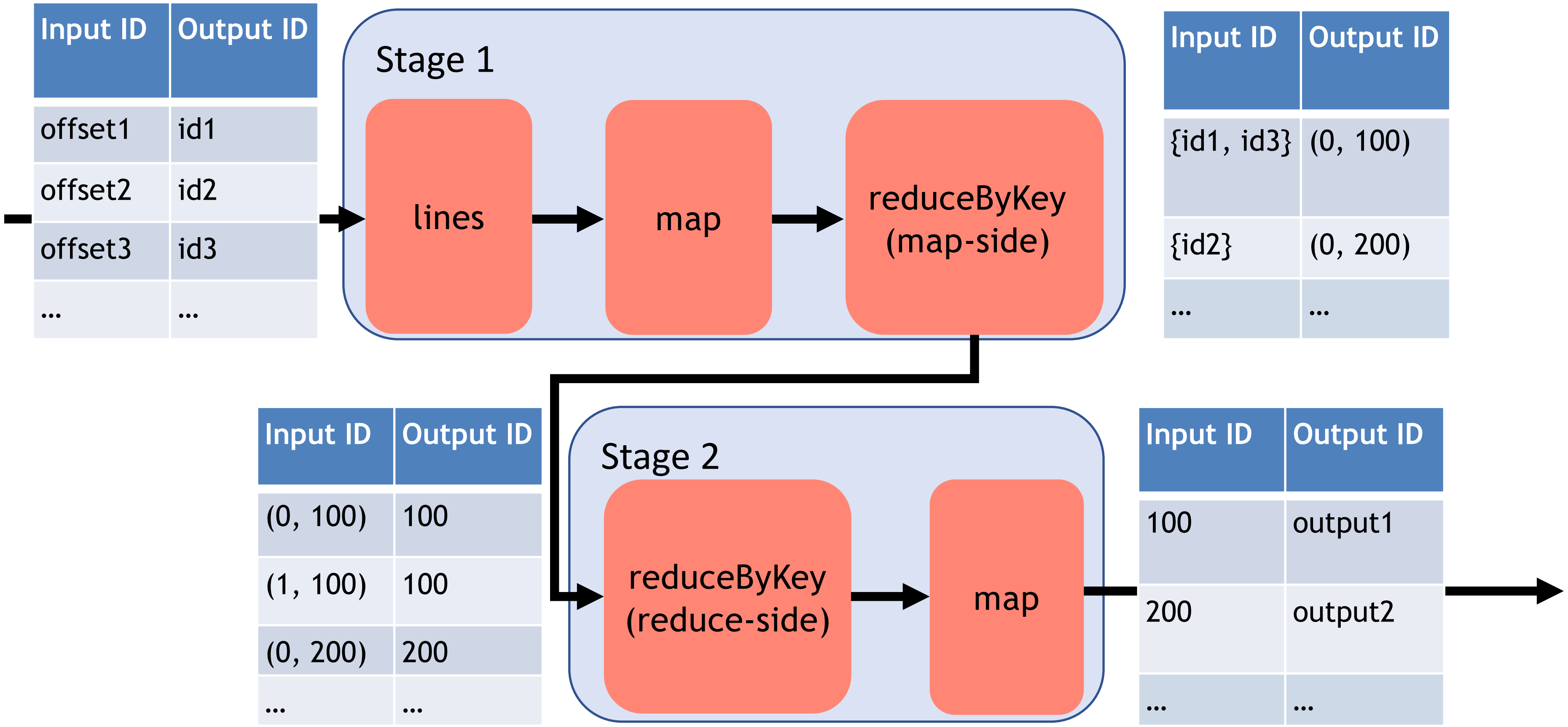
Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

Capture Data Provenance



Titian [VLDB 2016] provides data provenance using provenance tables at the start/end of stages to track input-output record mappings.

Capture Data Provenance



Titian [VLDB 2016] provides data provenance using provenance tables at the start/end of stages to track input-output record mappings.

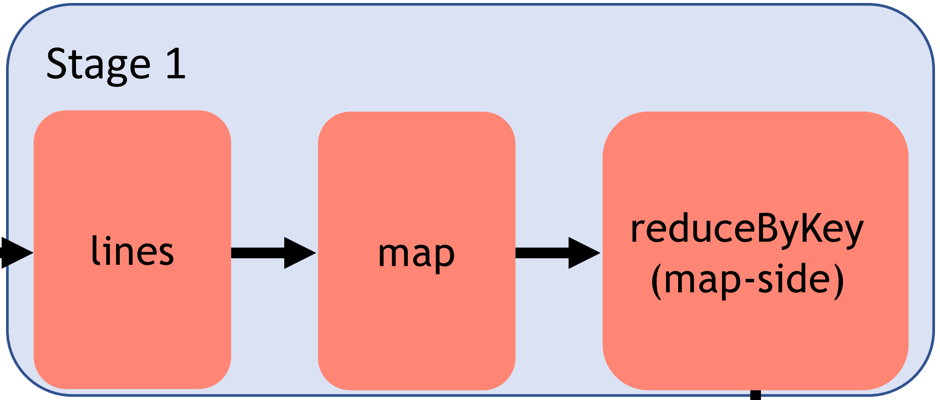
Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

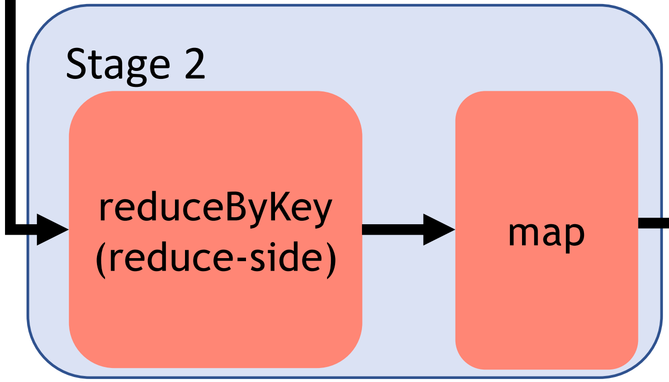
Capture Data Provenance

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…



Input ID	Output ID
{id1, id3}	(0, 100)
{id2}	(0, 200)
…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…



Input ID	Output ID
100	***output1***
200	output2
…	…

Titian [VLDB 2016] provides data provenance using provenance tables at the start/end of stages to track input-output record mappings.

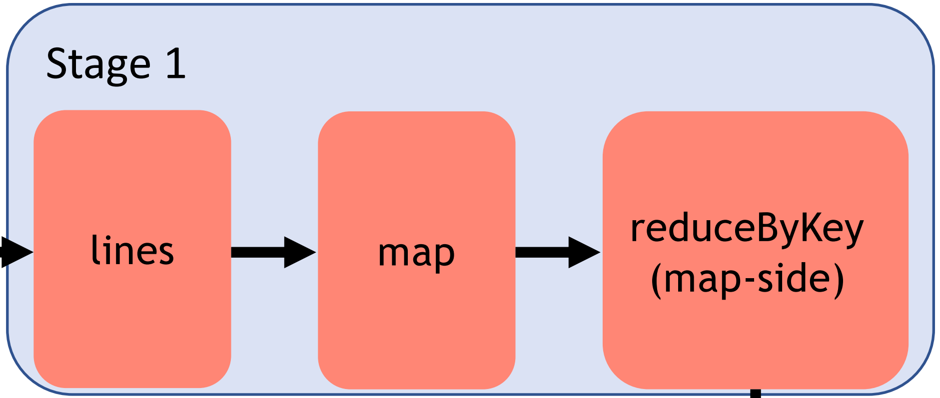
Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

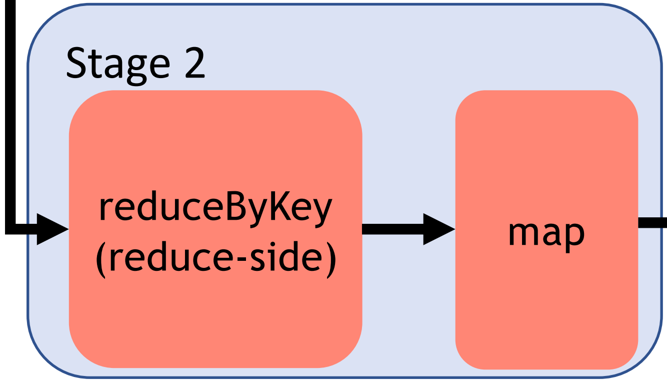
Capture Data Provenance

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…



Input ID	Output ID
{id1, id3}	(0, 100)
{id2}	(0, 200)
…	…

Input ID	Output ID
(0, 100)	***100***
(1, 100)	***100***
(0, 200)	200
…	…



Input ID	Output ID
100	***output1***
200	output2
…	…

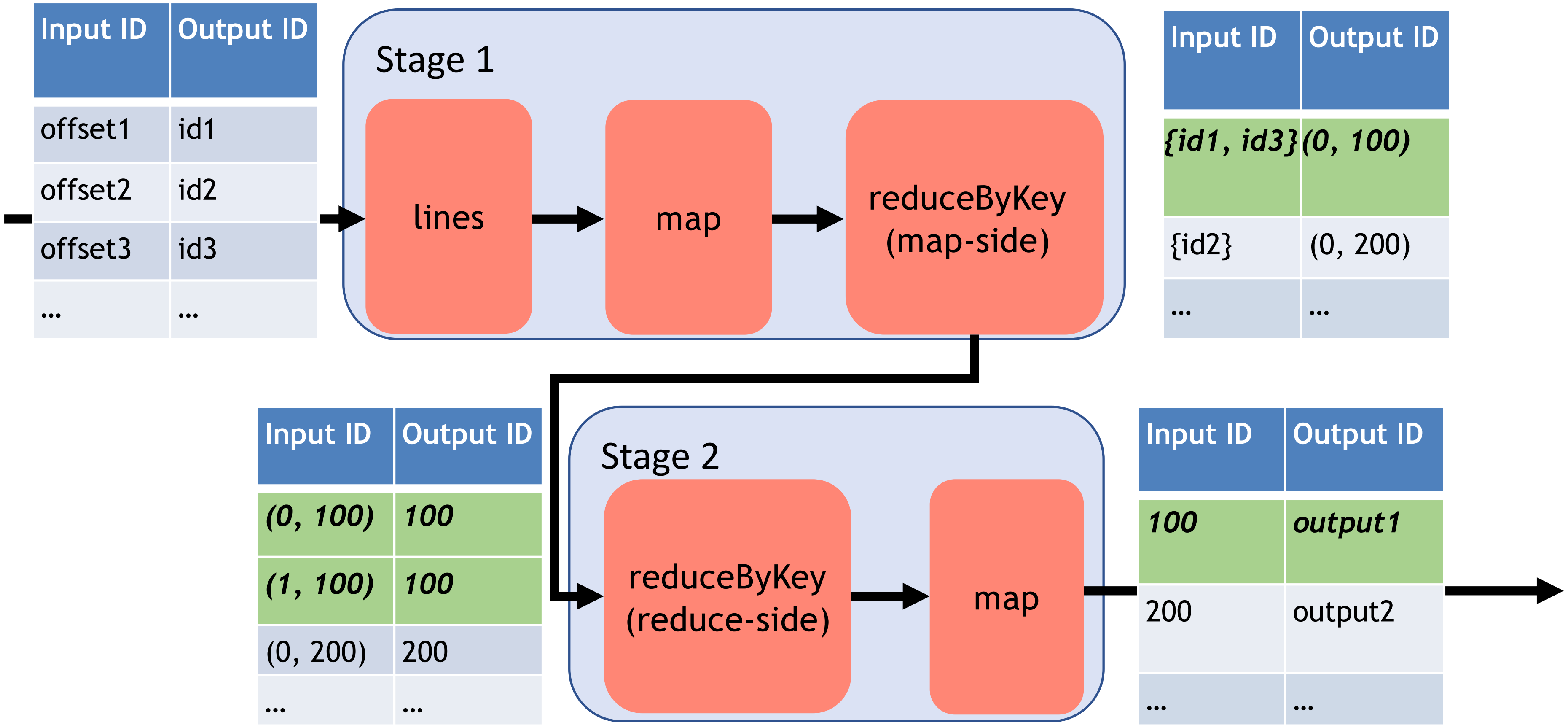
Titian [VLDB 2016] provides data provenance using provenance tables at the start/end of stages to track input-output record mappings.

Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

Capture Data Provenance



Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

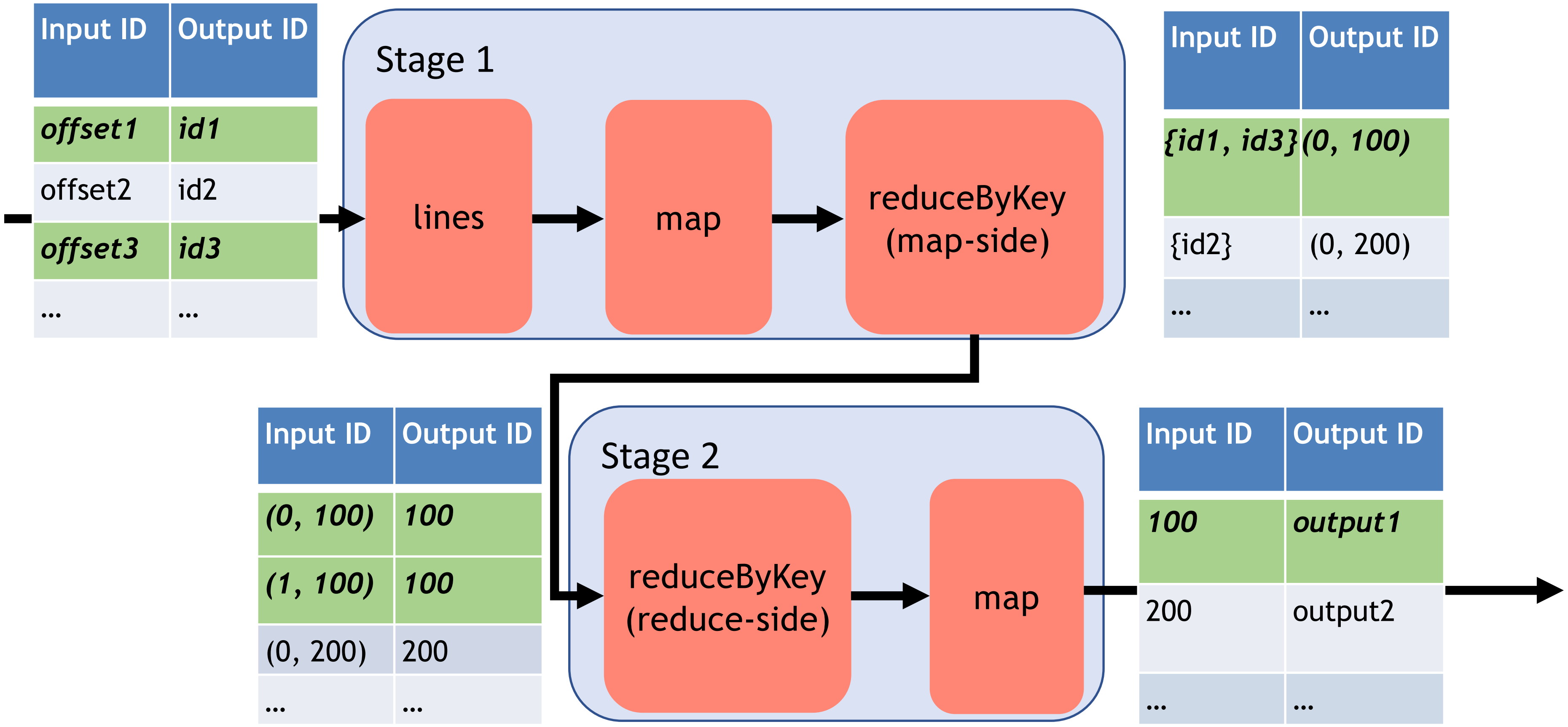
Input ID	Output ID
{id1, id3}	***(0, 100)***
{id2}	(0, 200)
…	…

Input ID	Output ID
(0, 100)	***100***
(1, 100)	***100***
(0, 200)	200
…	…

Input ID	Output ID
100	***output1***
200	output2
…	…

Titian [VLDB 2016] provides data provenance using provenance tables at the start/end of stages to track input-output record mappings.

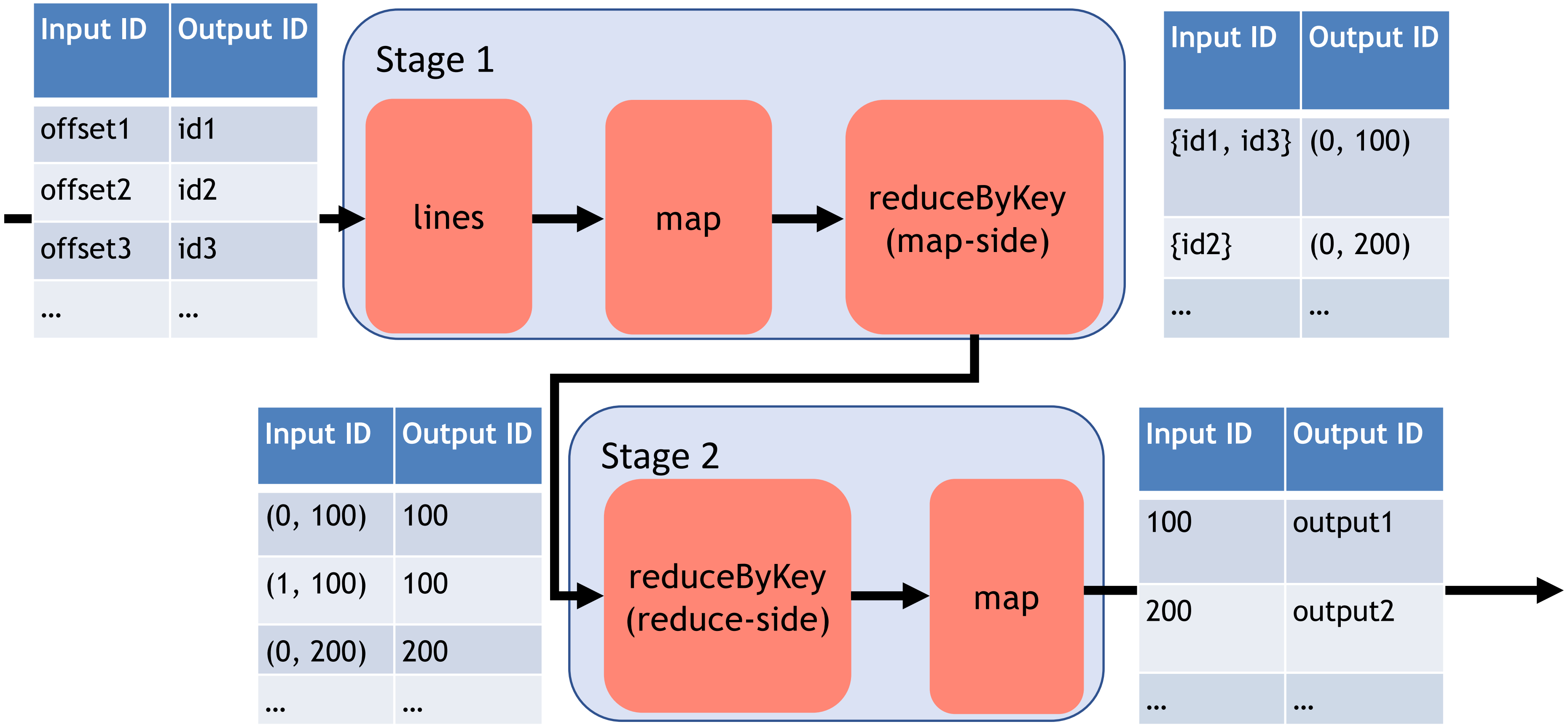
Capture Data Provenance



Titian [VLDB 2016] provides data provenance using provenance tables at the start/end of stages to track input-output record mappings.

Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

Measure UDF Latency



Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

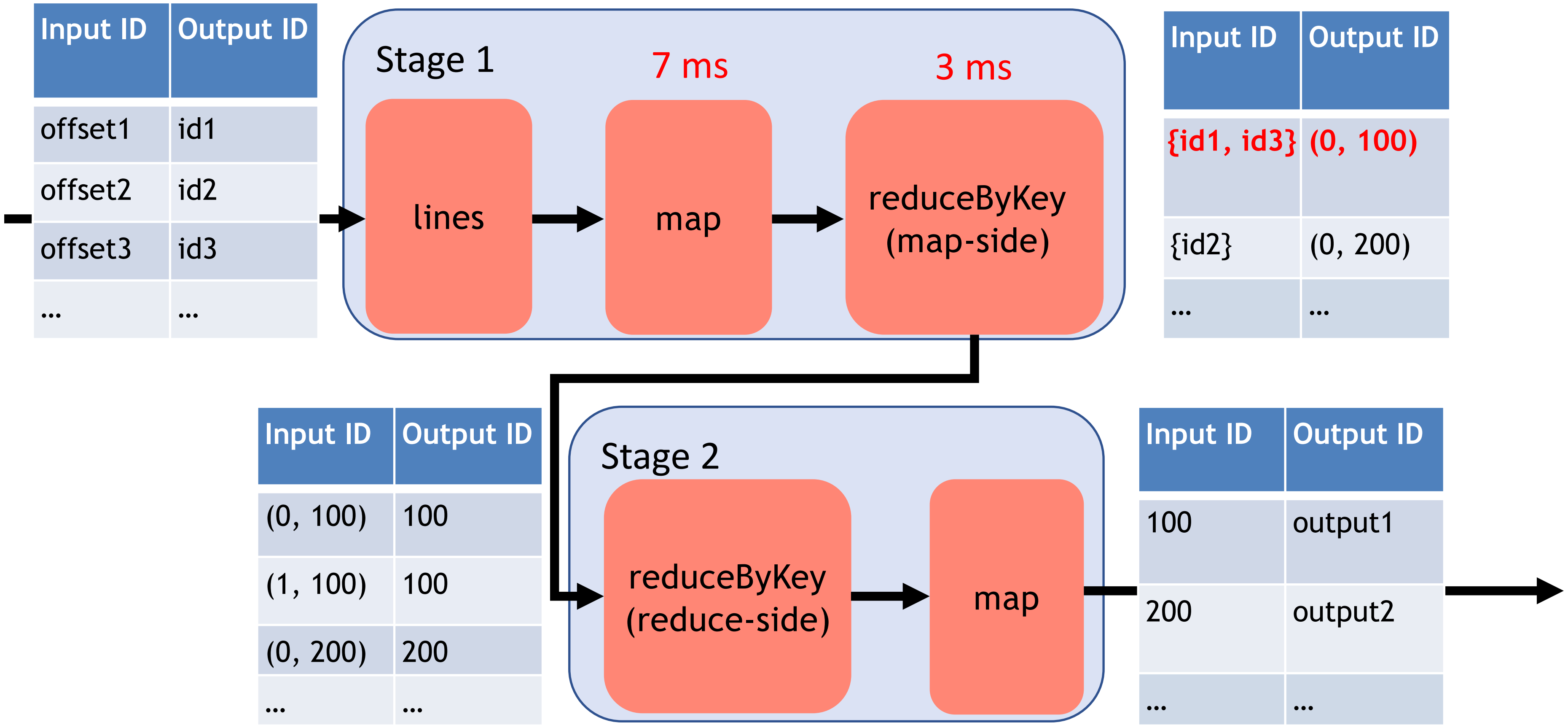
Input ID	Output ID
{id1, id3}	(0, 100)
{id2}	(0, 200)
…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…

Input ID	Output ID
100	output1
200	output2
…	…

PerfDebug extends Titian by capturing summed UDF execution times.

Measure UDF Latency



Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

Input ID	Output ID
{id1, id3}	(0, 100)
{id2}	(0, 200)
…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…

Input ID	Output ID
100	output1
200	output2
…	…

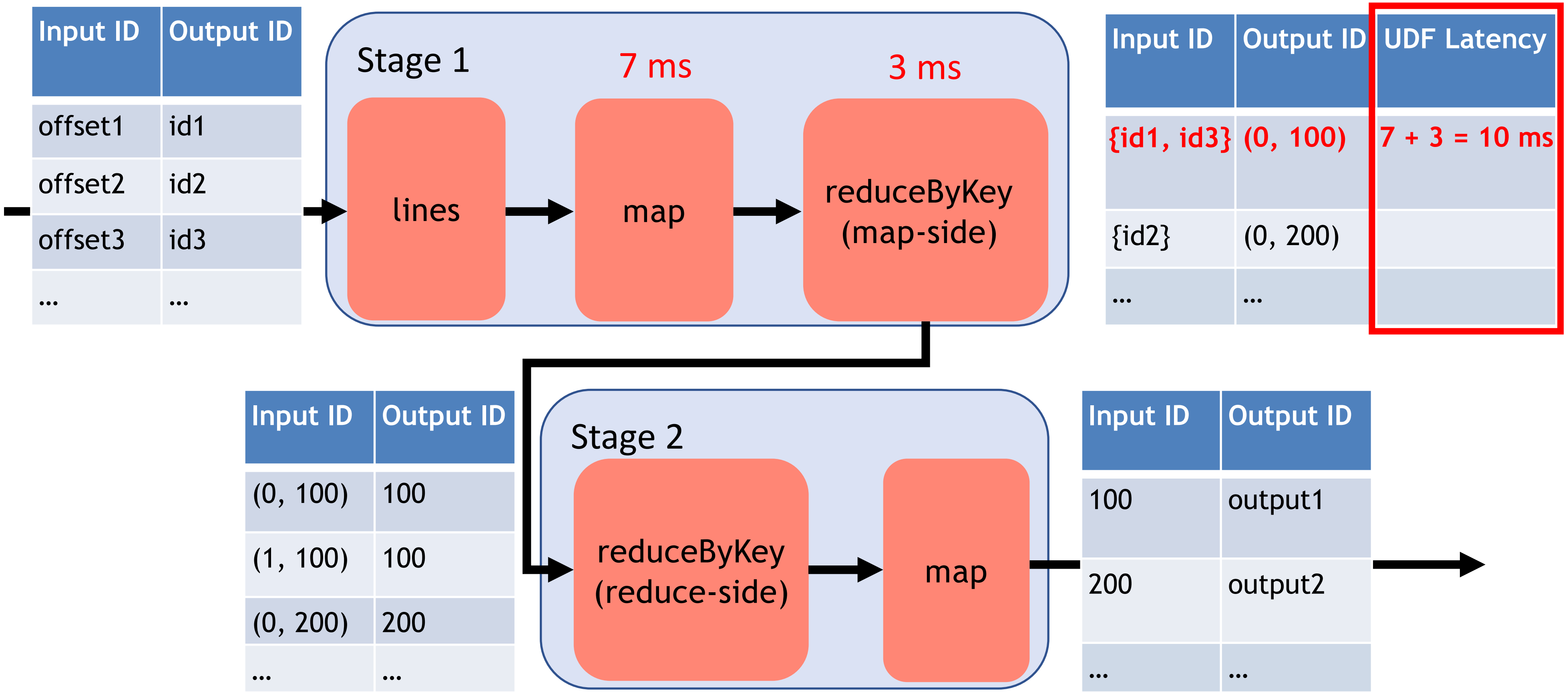
PerfDebug extends Titian by capturing summed UDF execution times.

Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

Measure UDF Latency



Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

Input ID	Output ID	UDF Latency
{id1, id3}	(0, 100)	7 + 3 = 10 ms
{id2}	(0, 200)	
…	…	

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…

Input ID	Output ID
100	output1
200	output2
…	…

PerfDebug extends Titian by capturing summed UDF execution times.

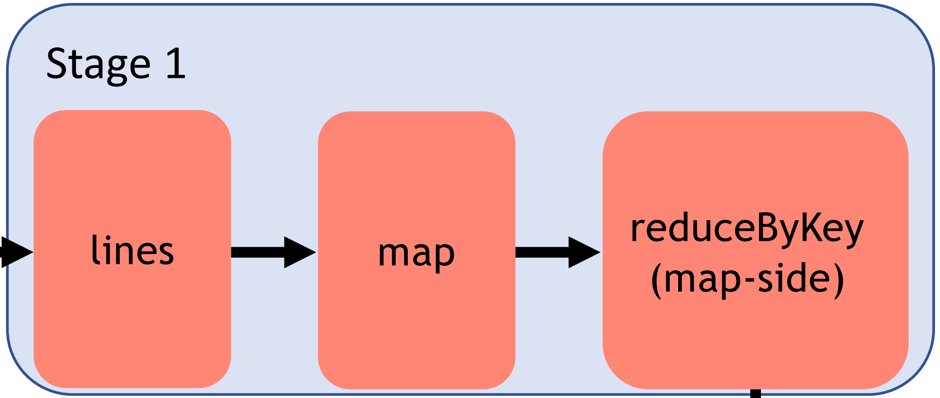
Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

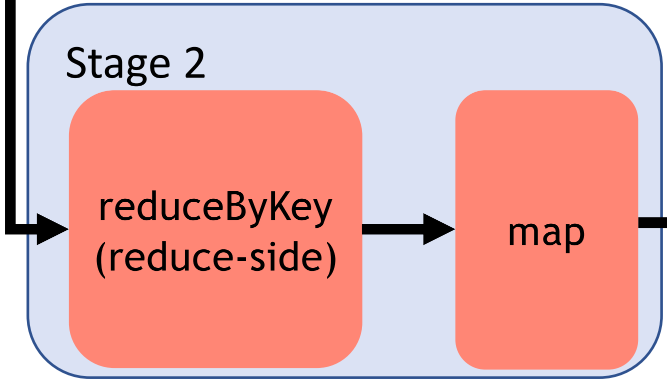
Measure UDF Latency

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…



Input ID	Output ID	UDF Latency
{id1, id3}	(0, 100)	10 ms
{id2}	(0, 200)	20 ms
…	…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…



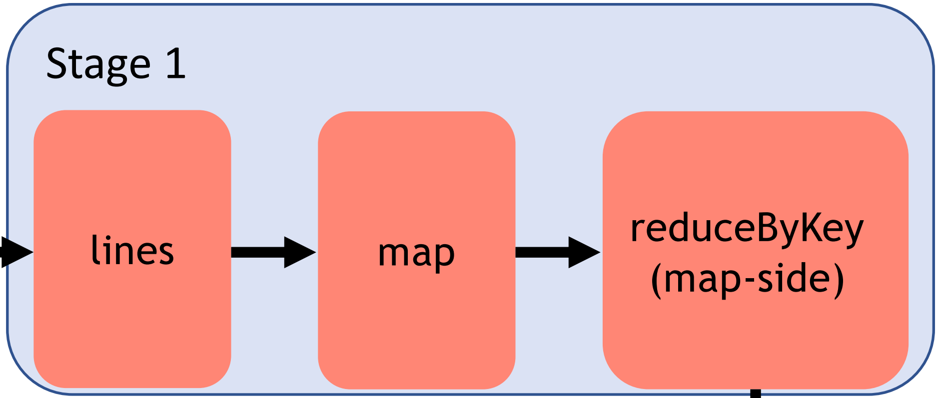
Input ID	Output ID	UDF Latency
100	output1	30 ms
200	output2	40 ms
…	…	…

PerfDebug extends Titian by capturing summed UDF execution times.

Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

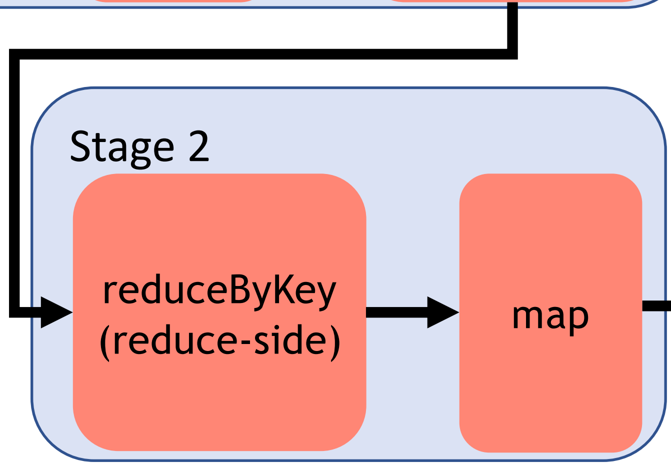
Measure Shuffle Latency

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…



Input ID	Output ID	UDF Latency
{id1, id3}	(0, 100)	10 ms
{id2}	(0, 200)	20 ms
…	…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…



Input ID	Output ID	UDF Latency
100	output1	30 ms
200	output2	40 ms
…	…	…

PerfDebug captures data movement costs through partition-level shuffle latencies.

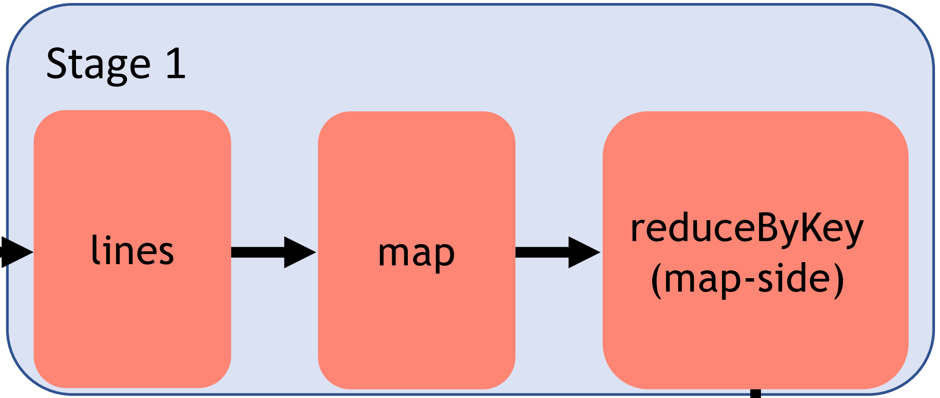
Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

Measure Shuffle Latency

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

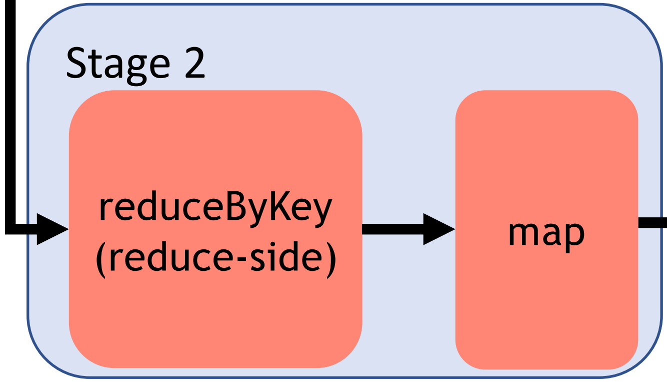


Input ID	Output ID	UDF Latency
{id1, id3}	(0, 100)	10 ms
{id2}	(0, 200)	20 ms
…	…	…

Stage 2

Partition	Shuffle Latency
1	80 ms
2	50 ms
3	100 ms
…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…



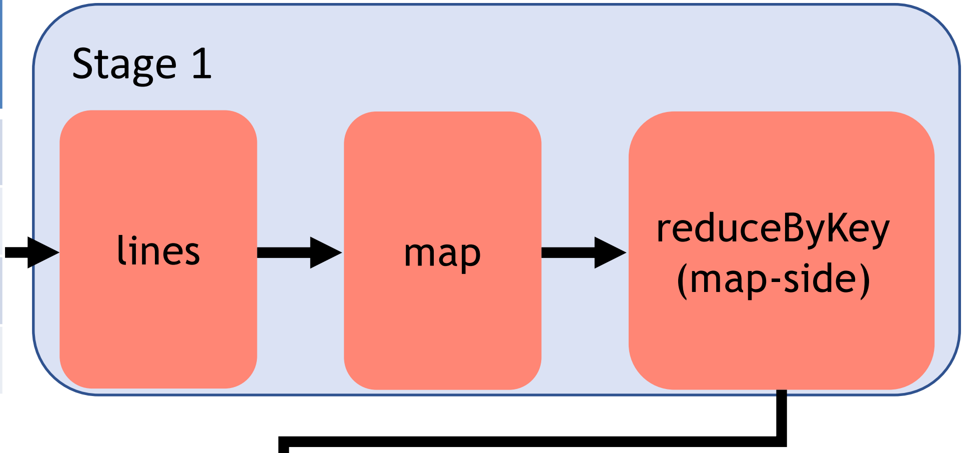
Input ID	Output ID	UDF Latency
100	output1	30 ms
200	output2	40 ms
…	…	…

PerfDebug captures data movement costs through partition-level shuffle latencies.

Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

Calculate Stage Latency

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

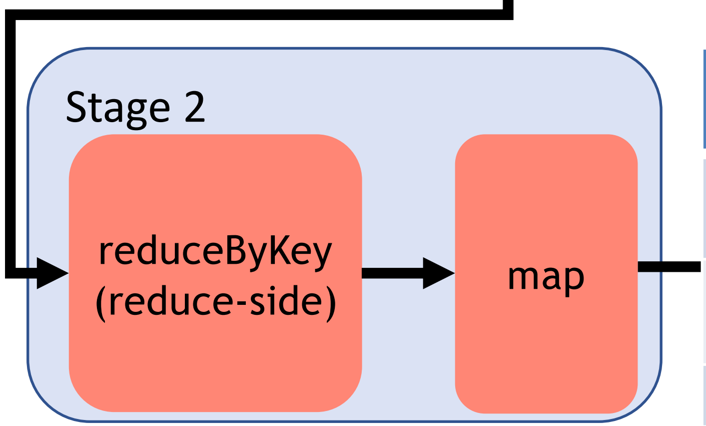


Input ID	Output ID	UDF Latency
{id1, id3}	(0, 100)	10 ms
{id2}	(0, 200)	20 ms
…	…	…

Stage 2

Partition	Shuffle Latency
1	80 ms
2	50 ms
3	100 ms
…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…

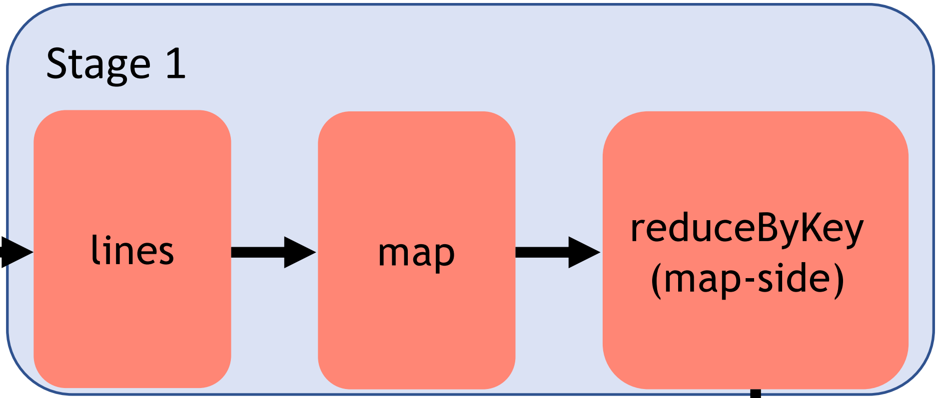


Input ID	Output ID	UDF Latency
100	output1	30 ms
200	output2	40 ms
…	…	…

PerfDebug calculates per-record stage latency by adding UDF latency and shuffle latency proportional to input count.

Calculate Stage Latency

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

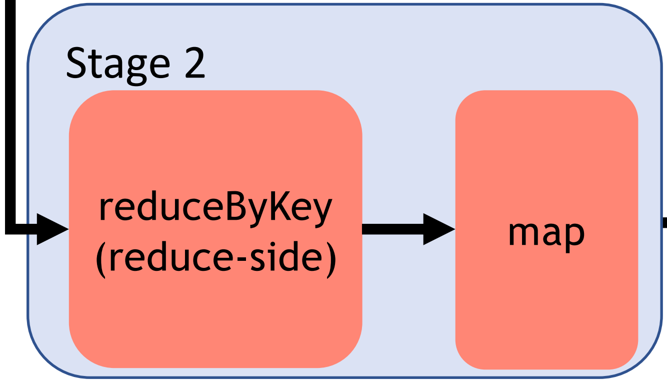


Input ID	Output ID	UDF Latency	Stg Latency
{id1, id3}	(0, 100)	10 ms	10 + 0 ms
{id2}	(0, 200)	20 ms	20 + 0 ms
…	…	…	…

Stage 2

Partition	Shuffle Latency
1	80 ms
2	50 ms
3	100 ms
…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…



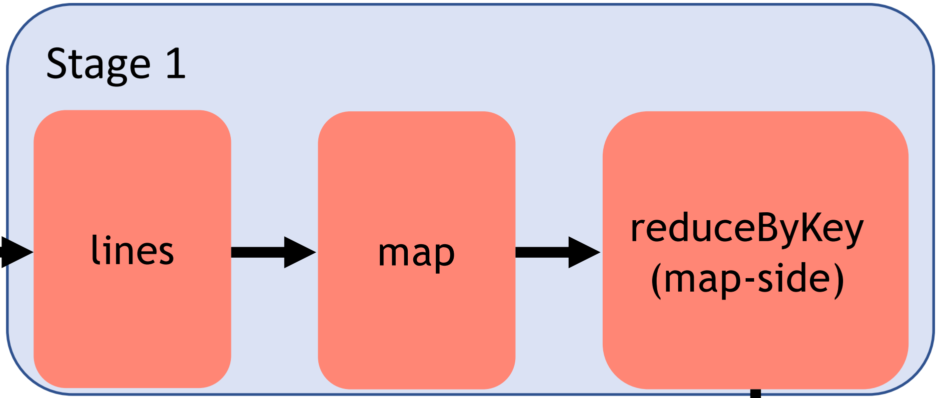
Input ID	Output ID	UDF Latency
100	output1	30 ms
200	output2	40 ms
…	…	…

PerfDebug calculates per-record stage latency by adding UDF latency and shuffle latency proportional to input count.

Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

Calculate Stage Latency

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

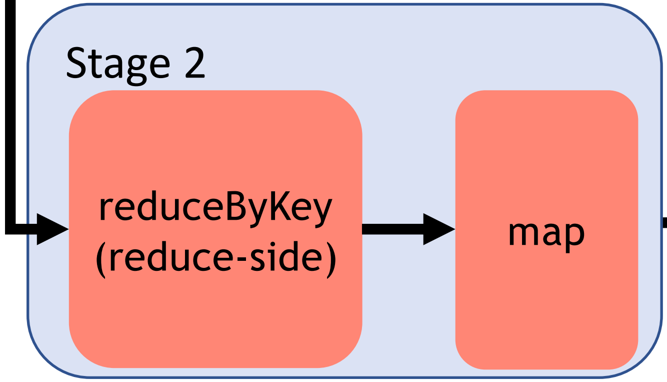


Input ID	Output ID	UDF Latency	Stg Latency
{id1, id3}	(0, 100)	10 ms	10 ms
{id2}	(0, 200)	20 ms	20 ms
…	…	…	…

Stage 2

Partition	Shuffle Latency
1	80 ms
2	50 ms
3	100 ms
…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…



Input ID	Output ID	UDF Latency	Stg Latency
100	**output1**	**30 ms**	
200	output2	40 ms	
…	…	…	…

PerfDebug calculates per-record stage latency by adding UDF latency and shuffle latency proportional to input count.

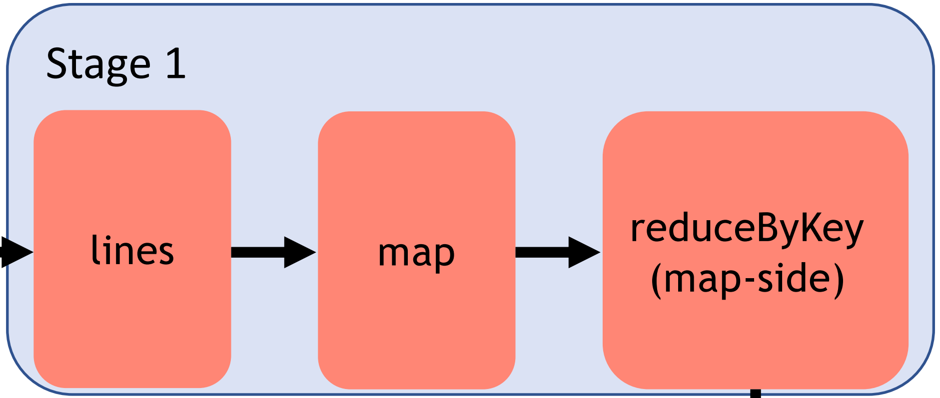
Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

Calculate Stage Latency

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

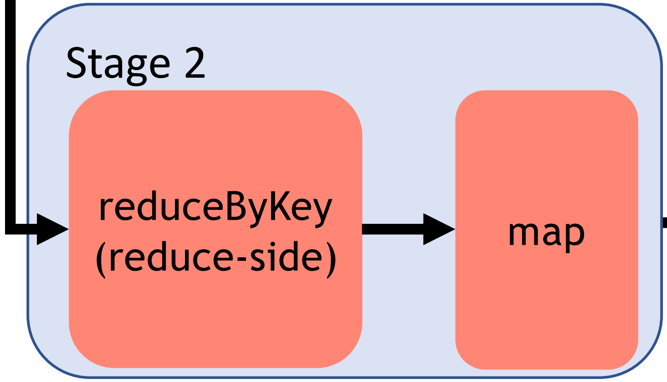


Input ID	Output ID	UDF Latency	Stg Latency
{id1, id3}	(0, 100)	10 ms	10 ms
{id2}	(0, 200)	20 ms	20 ms
…	…	…	…

Stage 2

Partition	Shuffle Latency
1	80 ms
2	50 ms
3	100 ms
…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…



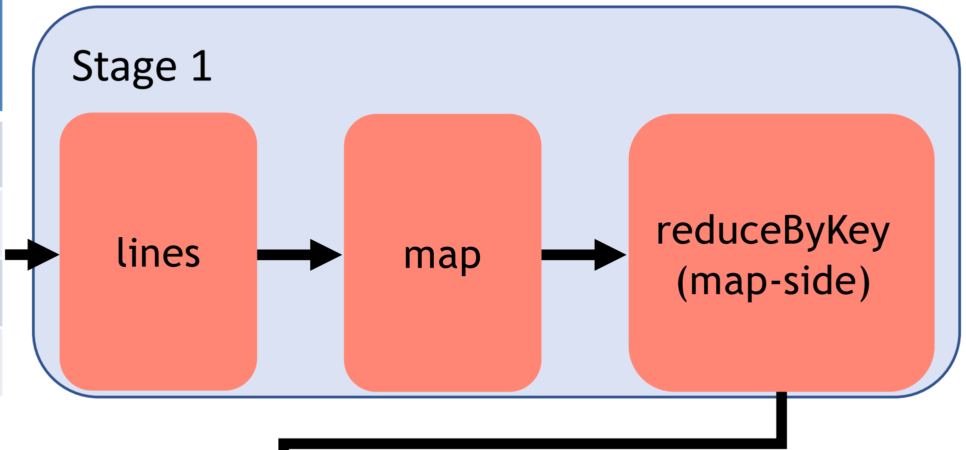
Input ID	Output ID	UDF Latency	Stg Latency
100	output1	30 ms	$30 + \frac{2}{16} * 80$
200	output2	40 ms	
…	…	…	…

PerfDebug calculates per-record stage latency by adding UDF latency and shuffle latency proportional to input count.

Calculate Stage Latency

Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3
…	…

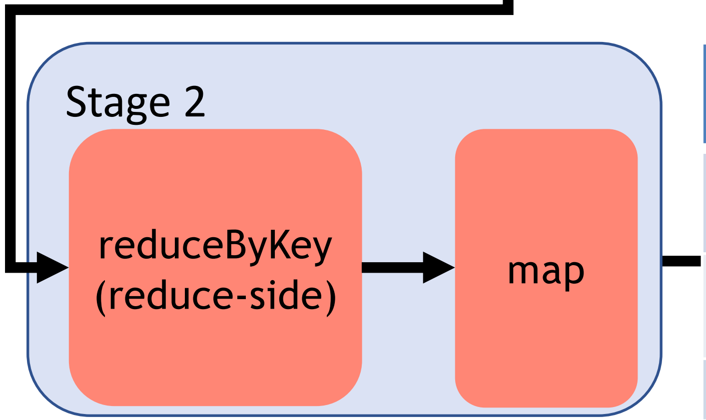


Input ID	Output ID	UDF Latency	Stg Latency
{id1, id3}	(0, 100)	10 ms	10 ms
{id2}	(0, 200)	20 ms	20 ms
…	…	…	…

Stage 2

Partition	Shuffle Latency
1	80 ms
2	50 ms
3	100 ms
…	…

Input ID	Output ID
(0, 100)	100
(1, 100)	100
(0, 200)	200
…	…



Input ID	Output ID	UDF Latency	Stg Latency
100	output1	30 ms	40 ms
200	output2	40 ms	45 ms
…	…	…	…

PerfDebug calculates per-record stage latency by adding UDF latency and shuffle latency proportional to input count.

PerfDebug Approach

Input:
Spark program, input data

Output:
Individual records responsible for computation skew

PerfDebug

Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

Expensive Record Identification

- Stage Latency is *within* a given stage and insufficient for debugging.
- Code and data interact across *multiple* stages.

Expensive Record Identification

- Stage Latency is *within* a given stage and insufficient for debugging.
- Code and data interact across *multiple* stages.

Stage 1

InputID	Output ID	Stg Latency
input1	o1	40 ms
input2	o2	30 ms
input3	o2	25 ms
input4	o3	40 ms
input5	o1	55 ms
input6	o3	60 ms

Stage 2

InputID	Output ID	Stg Latency
o1	output1	65 ms
o2	output2	70 ms
o3	output3	40 ms

Expensive Record Identification

- Stage Latency is *within* a given stage and insufficient for debugging.
- Code and data interact across *multiple* stages.

Stage 1

InputID	Output ID	Stg Latency
input1	o1	40 ms
input2	o2	30 ms
input3	o2	25 ms
input4	o3	40 ms
input5	o1	55 ms
input6	o3	60 ms ?

Stage 2

InputID	Output ID	Stg Latency
o1	output1	65 ms
o2	output2	70 ms ?
o3	output3	40 ms

What about the slowest records in each stage?

Expensive Record Identification

- Stage Latency is *within* a given stage and insufficient for debugging.
- Code and data interact across *multiple* stages.

Stage 1

InputID	Output ID	Stg Latency
input1	**o1**	**40 ms**
input2	o2	30 ms
input3	o2	25 ms
input4	o3	40 ms
input5	**o1**	**55 ms**
input6	o3	60 ms ?

Stage 2

InputID	Output ID	Stg Latency
o1	**output1**	**65 ms**
o2	output2	70 ms ?
o3	output3	40 ms

Computation skew can span multiple stages!

Propagate End-to-End Latency

Stage K

InputID	Output ID	E2E Latency
input1	o1	40 ms
input2	o2	30 ms
input3	o2	25 ms
input4	o3	40 ms
input5	o1	55 ms
input6	o3	60 ms

Stage K+1

InputID	Output ID	Stg Latency
o1	output1	65 ms
o2	output2	70 ms
o3	output3	40 ms

PerfDebug propagates end-to-end latency by adding stage latency to the slowest (max) end-to-end latency of the previous stage.

Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

Propagate End-to-End Latency

Stage K

InputID	Output ID	E2E Latency
input1	**o1**	**40 ms**
input2	o2	30 ms
input3	o2	25 ms
input4	o3	40 ms
input5	**o1**	**55 ms**
input6	o3	60 ms

Stage K+1

InputID	Output ID	Stg Latency	E2E Latency
o1	**output1**	**65 ms**	**65 + max(40,55)**
o2	output2	70 ms	
o3	output3	40 ms	

PerfDebug propagates end-to-end latency by adding stage latency to the slowest (max) end-to-end latency of the previous stage.

Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

Propagate End-to-End Latency

Stage K

InputID	Output ID	E2E Latency
input1	o1	40 ms
input2	o2	30 ms
input3	o2	25 ms
input4	o3	40 ms
input5	o1	55 ms
input6	o3	60 ms

Stage K+1

InputID	Output ID	Stg Latency	E2E Latency
o1	output1	65 ms	65 + max(40,55)
o2	output2	70 ms	70 + max(30,25)
o3	output3	40 ms	40 + max(40,60)

PerfDebug propagates end-to-end latency by adding stage latency to the slowest (max) end-to-end latency of the previous stage.

Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

Propagate End-to-End Latency

Stage K

InputID	Output ID	E2E Latency
input1	o1	40 ms
input2	o2	30 ms
input3	o2	25 ms
input4	o3	40 ms
input5	o1	55 ms
input6	o3	60 ms

Stage K+1

InputID	Output ID	E2E Latency
o1	output1	120 ms
o2	output2	100 ms
o3	output3	100 ms

PerfDebug propagates end-to-end latency by adding stage latency to the slowest (max) end-to-end latency of the previous stage.

Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

Propagate End-to-End Latency

Stage K

InputID	Output ID	E2E Latency
input1	**o1**	**40 ms**
input2	o2	30 ms
input3	o2	25 ms
input4	o3	40 ms
input5	**o1**	**55 ms**
input6	o3	60 ms

Stage K+1

InputID	Output ID	E2E Latency
o1	**output1**	**120 ms**
o2	output2	100 ms
o3	output3	100 ms

PerfDebug propagates end-to-end latency by adding stage latency to the slowest (max) end-to-end latency of the previous stage.

Propagate Expensive Inputs

- Not all inputs contribute equally to application performance.
- Data provenance alone cannot differentiate between these inputs if multiple map to the same record.

Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

Propagate Expensive Inputs

Stage 1

InputID	Output ID	E2E Latency
input1	o1	40 ms
input2	o2	30 ms
input3	o2	25 ms
input4	o3	40 ms
input5	o1	55 ms
input6	o3	60 ms

Stage 2

InputID	Output ID	E2E Latency
o1	output1	120 ms
o2	output2	100 ms
o3	output3	100 ms

For each record within each stage, PerfDebug extends end-to-end latency by tracking the program input for the path of max latency.

Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

Propagate Expensive Inputs

Stage 1

InputID	Output ID	E2E Latency	Exp. Input
input1	o1	40 ms	**input1**
input2	o2	30 ms	**input2**
input3	o2	25 ms	**input3**
input4	o3	40 ms	**input4**
input5	o1	55 ms	**input5**
input6	o3	60 ms	**input6**

Stage 2

InputID	Output ID	E2E Latency
o1	output1	120 ms
o2	output2	100 ms
o3	output3	100 ms

For each record within each stage, PerfDebug extends end-to-end latency by tracking the program input for the path of max latency.

Computation Skew Detection | Data Provenance + Record-Level Latency | Expensive Record Identification

Propagate Expensive Inputs

Stage 1

InputID	Output ID	E2E Latency	Exp. Input
input1	**o1**	**40 ms**	**input1**
input2	o2	30 ms	input2
input3	o2	25 ms	input3
input4	o3	40 ms	input4
input5	**o1**	**55 ms**	**input5**
input6	o3	60 ms	input6

Stage 2

InputID	Output ID	E2E Latency	Exp. Input
o1	**output1**	**120 ms**	**input5**
o2	output2	100 ms	
o3	output3	100 ms	

For each record within each stage, PerfDebug extends end-to-end latency by tracking the program input for the path of max latency.

Computation Skew Detection

Data Provenance + Record-Level Latency

Expensive Record Identification

Propagate Expensive Inputs

Stage 1

InputID	Output ID	E2E Latency	Exp. Input
input1	o1	40 ms	input1
input2	o2	30 ms	input2
input3	o2	25 ms	input3
input4	o3	40 ms	input4
input5	o1	55 ms	input5
input6	o3	60 ms	input6

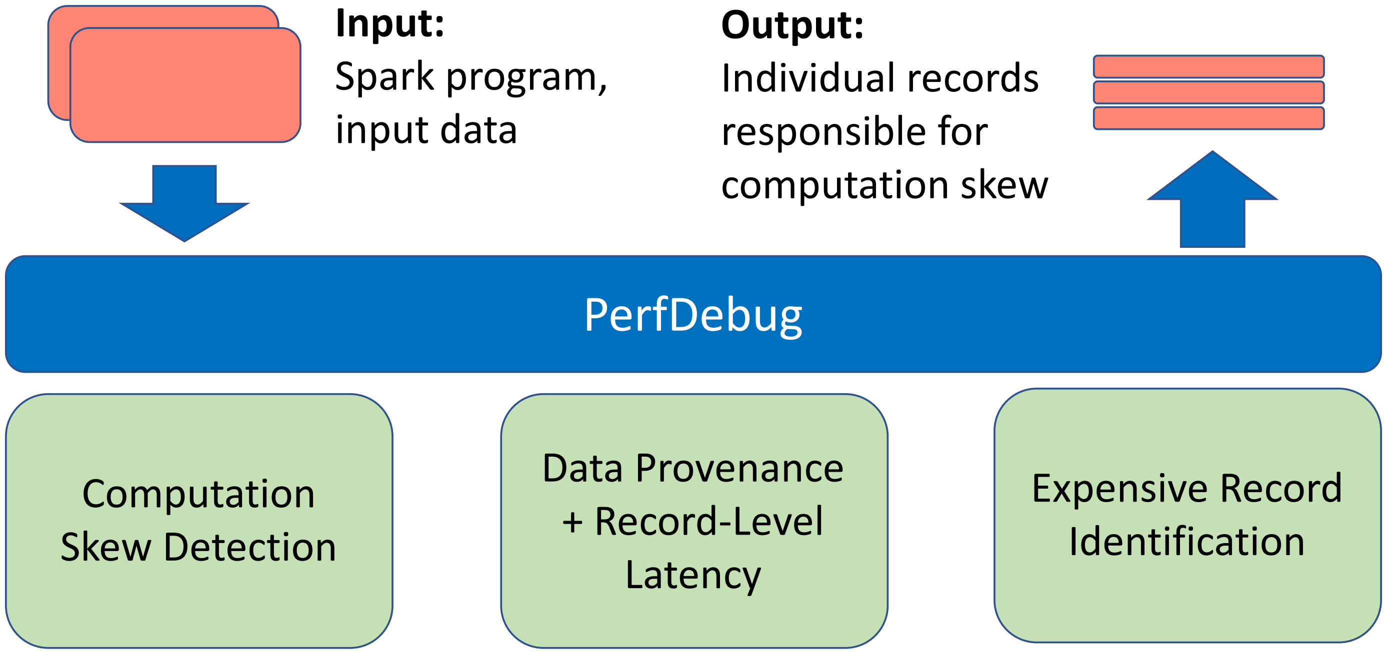
Stage 2

InputID	Output ID	E2E Latency	Exp. Input
o1	output1	120 ms	input5
o2	output2	100 ms	input2
o3	output3	100 ms	input6

For each record within each stage, PerfDebug extends end-to-end latency by tracking the program input for the path of max latency.

PerfDebug Approach Recap

- Monitoring to detect presence of computation skew
- Instrumented execution to collect data provenance and latency
- Propagation algorithm to analyze end-to-end record impact and identify records responsible for computation skew



Evaluation

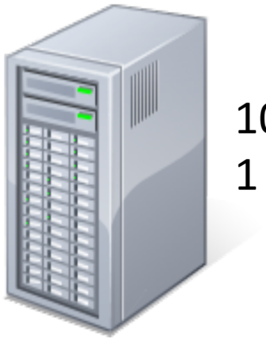
RQ1: What is the impact of applying appropriate remediations?

RQ2: How much overhead does PerfDebug introduce?

RQ3: How accurate is PerfDebug at identifying delay-inducing inputs?

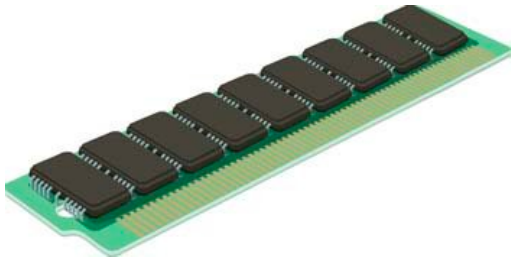
RQ1: Remediation Impact

- Three case studies with varying computation skew causes: *data skew*, *data quality*, and *expensive UDF*.
- **1.5X to 16X performance improvement** with case-specific fixes.



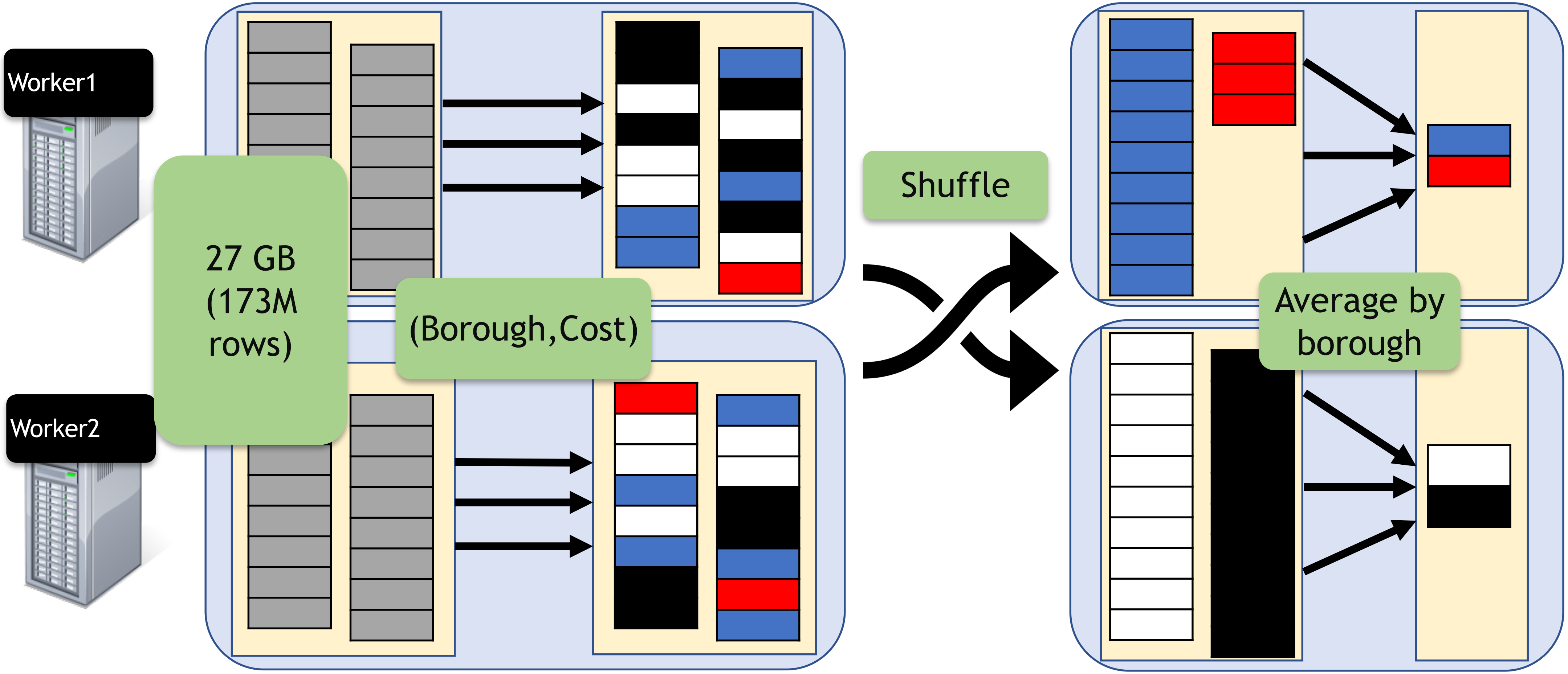
10 Workers
1 Master

15-27 GB Data



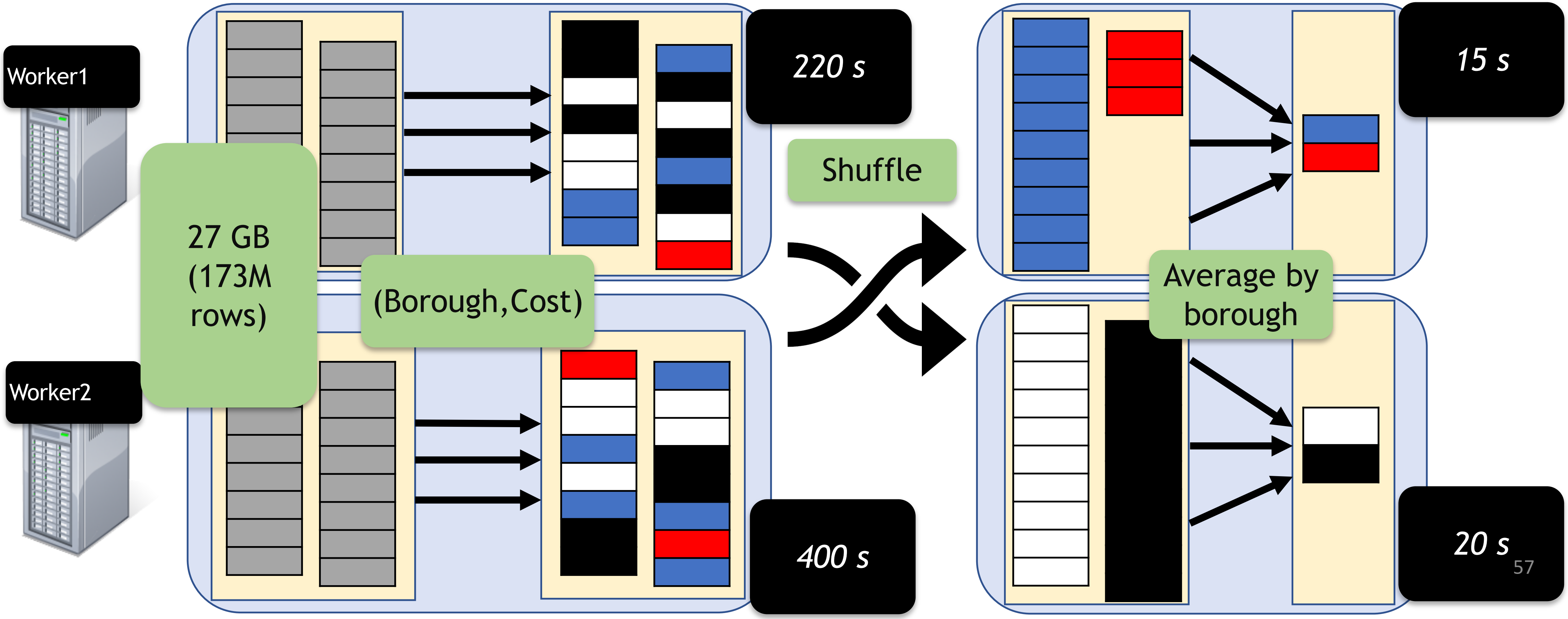
24GB Memory / worker

NYC Taxi Trips Case Study



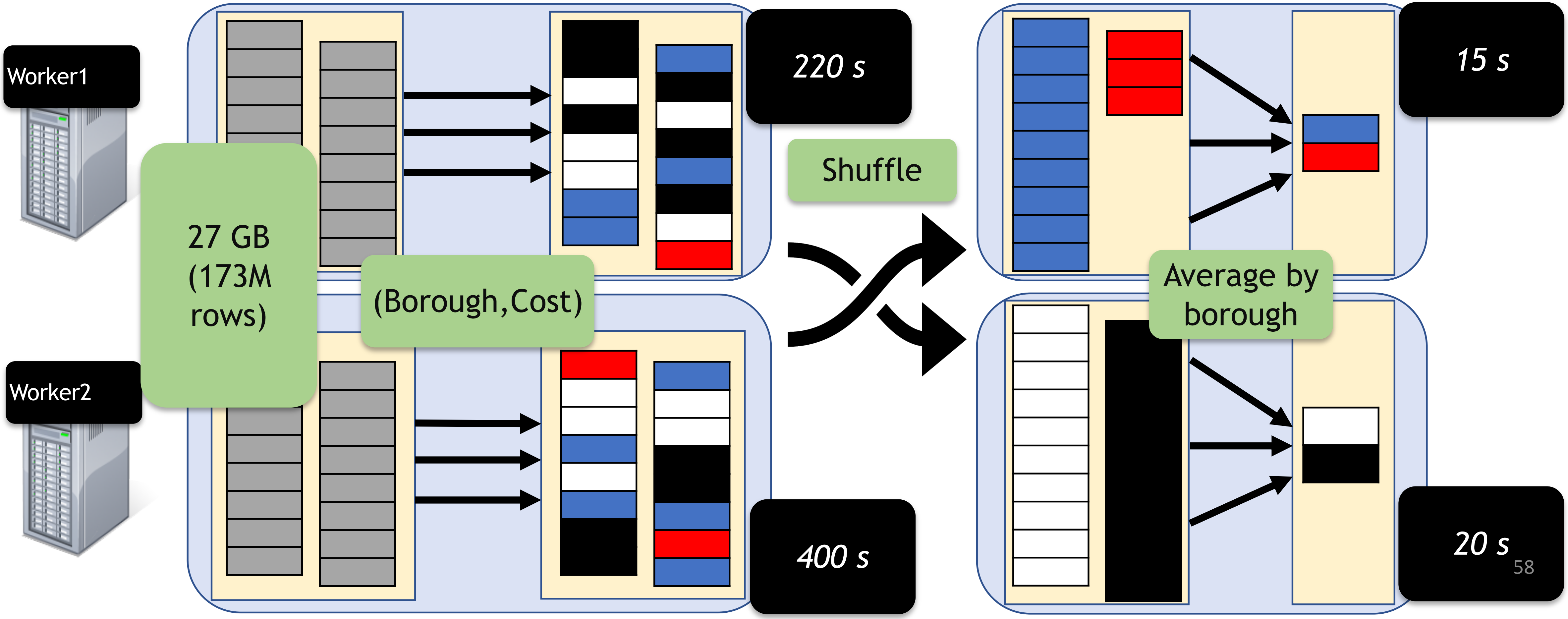
Goal: compute average cost of a taxi ride for each starting borough

NYC Taxi Trips Case Study



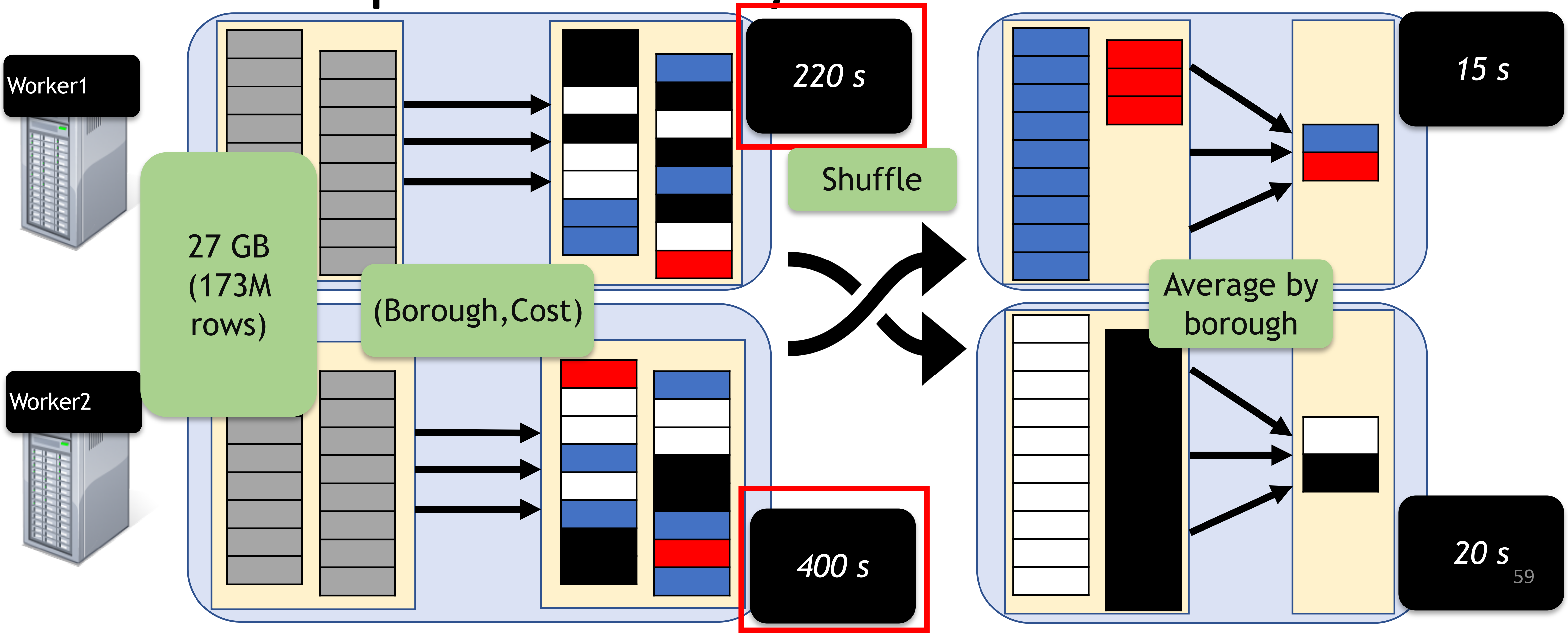
Total runtime: ~7 minutes

NYC Taxi Trips Case Study



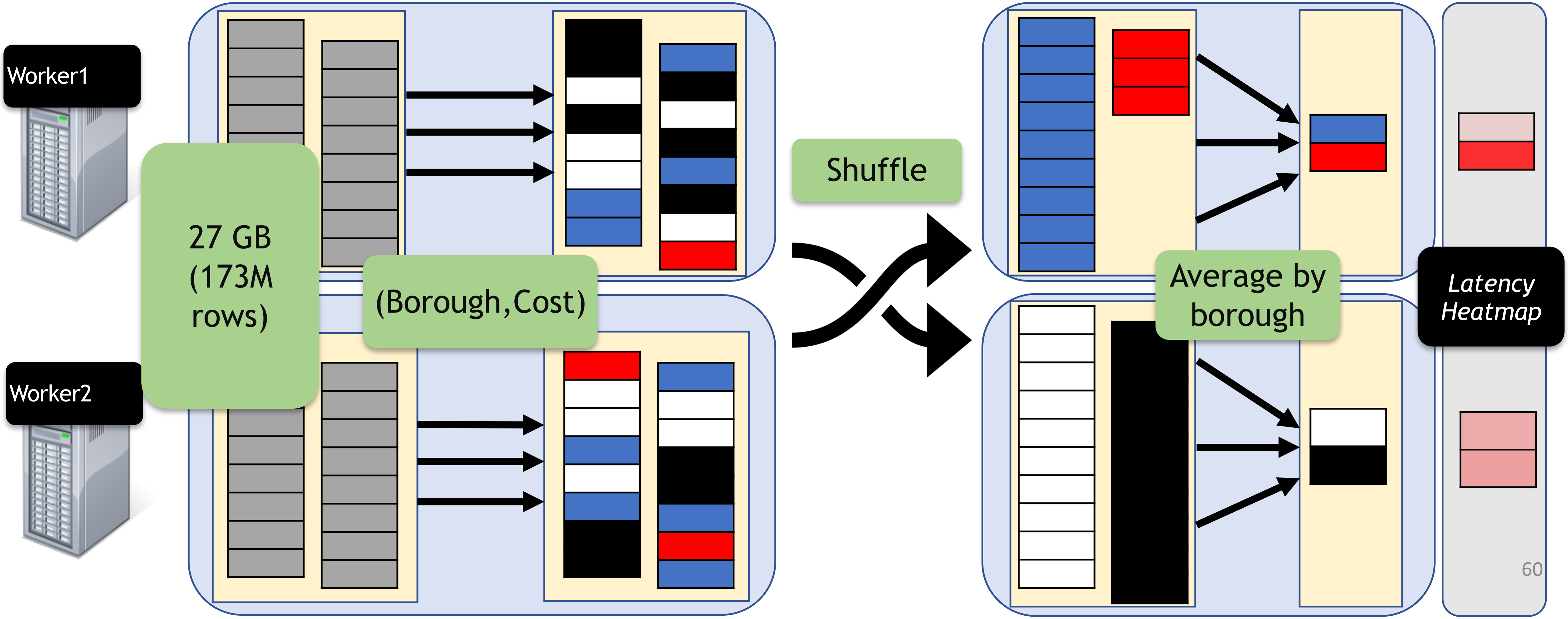
Task times show that data skew is a minor performance factor.

NYC Taxi Trips Case Study



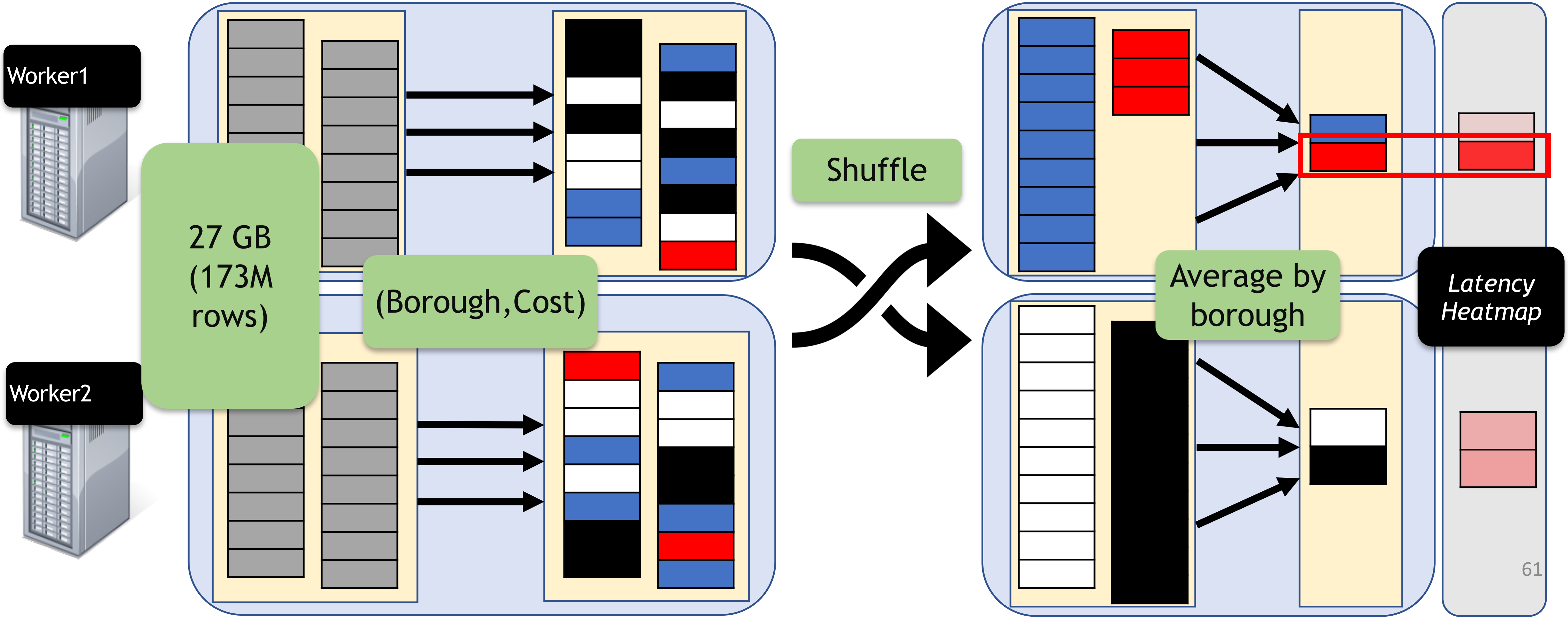
PerfDebug detects potential computation skew in the first stage.

NYC Taxi Trips Case Study



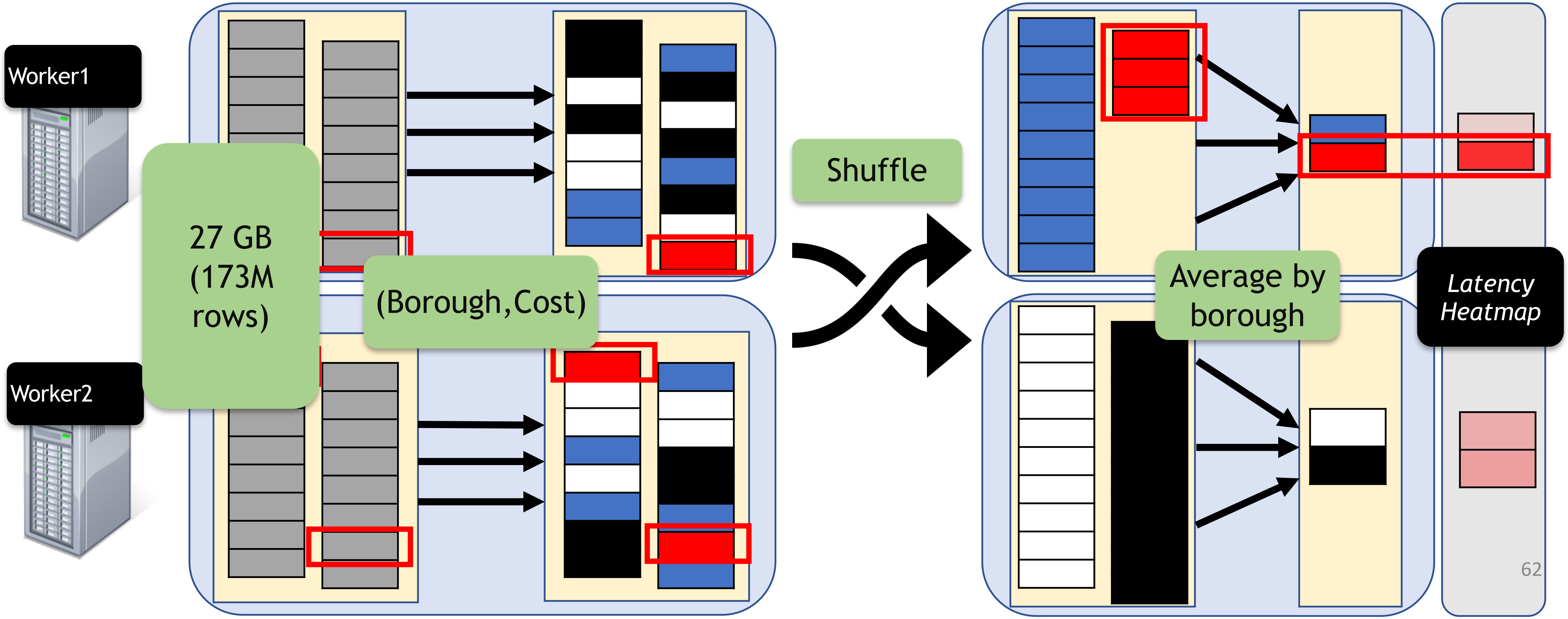
PerfDebug identifies the outputs with the highest latency and uses provenance to trace the corresponding inputs.

NYC Taxi Trips Case Study



PerfDebug identifies the outputs with the highest latency and uses provenance to trace the corresponding inputs.

NYC Taxi Trips Case Study



PerfDebug identifies the outputs with the highest latency and uses provenance to trace the corresponding inputs.

NYC Taxi Trips Case Study Results

- PerfDebug isolates the source of computation skew to a small subset of inputs: 0.0006%
- Inspection reveals that a *getBorough* UDF consumes majority of task time.

Removal of these records results in **~16X performance improvement**.

RQ2: Instrumentation Overhead

- Three benchmarks, ten trials each.
- Titian adds ~30% runtime overhead versus Spark [VLDB 2016].
- PerfDebug adds ~**30% runtime overhead** compared to Titian.
- Majority of additional overhead due to using persistent storage for post-mortem debugging, which was not required in Titian.

RQ3: Precision and Recall

- Three benchmarks, ten trials each.
- Use *mutation testing* to randomly inject an input record with delays.
- PerfDebug consistently identified target: **100% precision and recall**.
- **2-6 orders of magnitude better precision** compared to provenance-only input tracing of outputs using Titian.

Benchmark	Accuracy	Precision Improvement	Overhead
Movie Ratings	100%	10^3 X	1.04X
College Students	100%	10^6 X	1.39X
Weather Analysis	100%	10^2 X	1.48X
Average	100%	10^5 X	1.30X

Conclusion

- PerfDebug is a post-mortem performance debugging tool that combines *data provenance* and *record-level latency* instrumentation to precisely pinpoint records which cause computation skew.
- Case-specific fixes can yield up to 16X performance improvement.

Related Work

- Ernest [NSDI 2016], ARIA [ICAC 2011], Jockey [Eurosys 2012], Starfish [CIDR 2011]: performance modeling for prediction, but not debugging of computation skew
- PerfXplain [VLDB 2012]: job and task comparison for debugging and explanation with respect to collected metrics.
- Titian [VLDB 2016]: data provenance within Apache Spark, used as foundation for PerfDebug implementation.
- Additional works mentioned in paper.