Unified Batch & Stream Processing in Apache Flink

Apache Flink Meetup Berlin #6
April 29, 2015

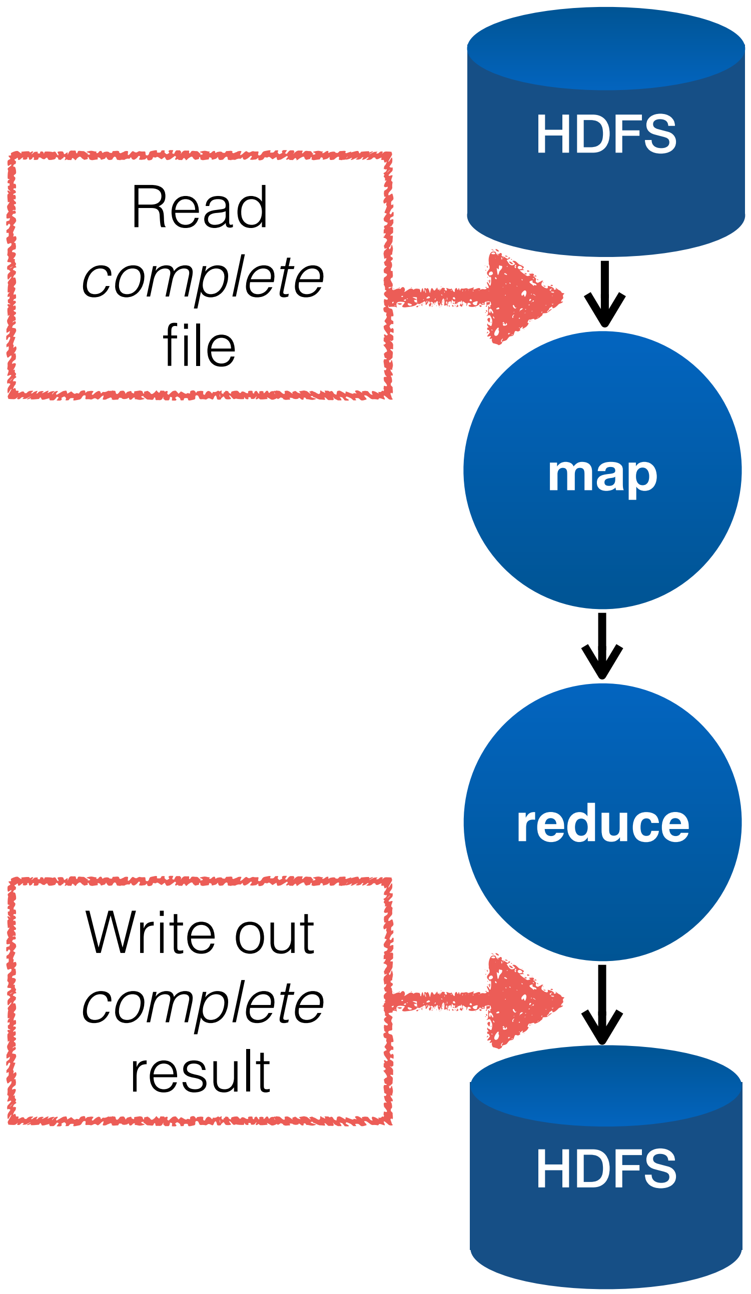
Ufuk Celebi
uce@apache.org

Batch vs. Stream Processing

Batch Processing	**Stream** Processing
High Latency	**Low** Latency
Batch Processors	**Stream Processors**
Static Files	**Event Streams**

What's the *difference* between a **Batch** and **Streaming** *Runtime*?

Batch Processing



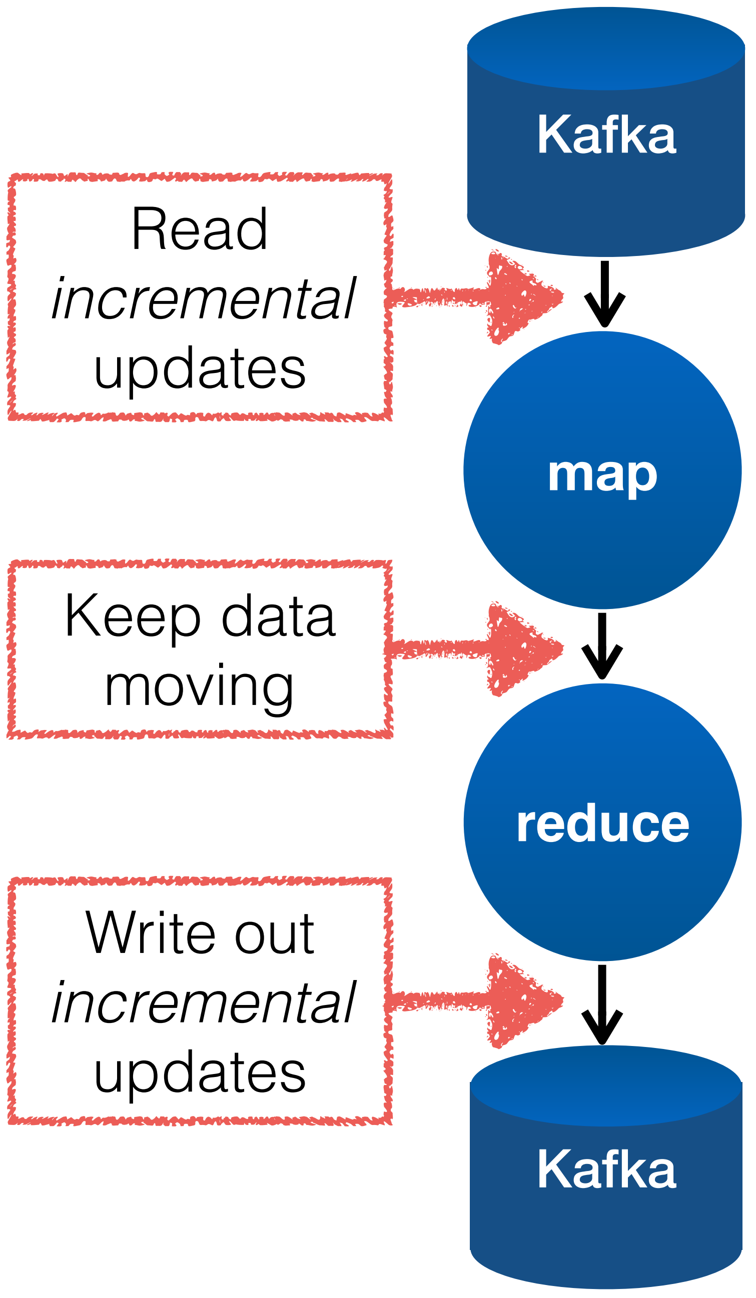
O Romeo, Romeo,
wherefore art thou Romeo?

Romeo, 1
Romeo, 1
wherefore, 1
art, 1
thou, 1
Romeo, 1

Write out intermediate data set

Romeo, 3
wherefore, 1
art, 1
thou, 1

Stream Processing



O	Romeo	Romeo	
wherefore	art	thou	Romeo

Romeo, 1
Romeo, 1
wherefore, 1
art, 1
thou, 1
Romeo, 1

Romeo, 3
wherefore, 1
art, 1
thou, 1

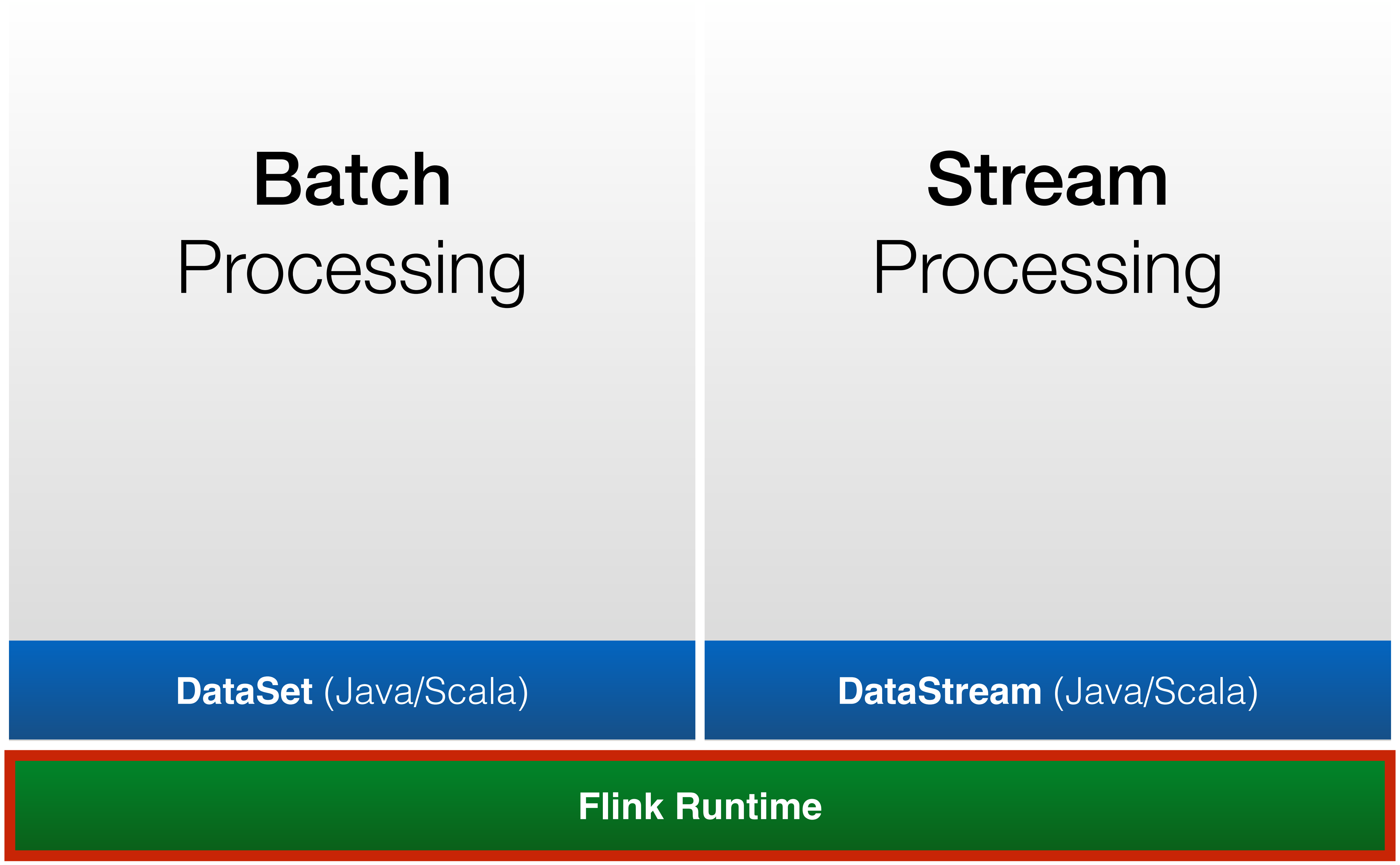
and

Batch ~~vs.~~ Stream

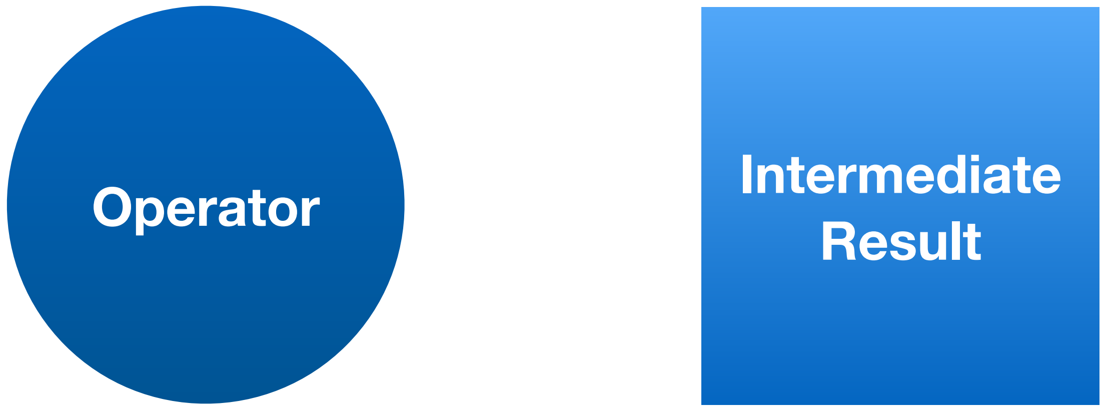
Processing

Do we *really* need two **separate systems** for this?

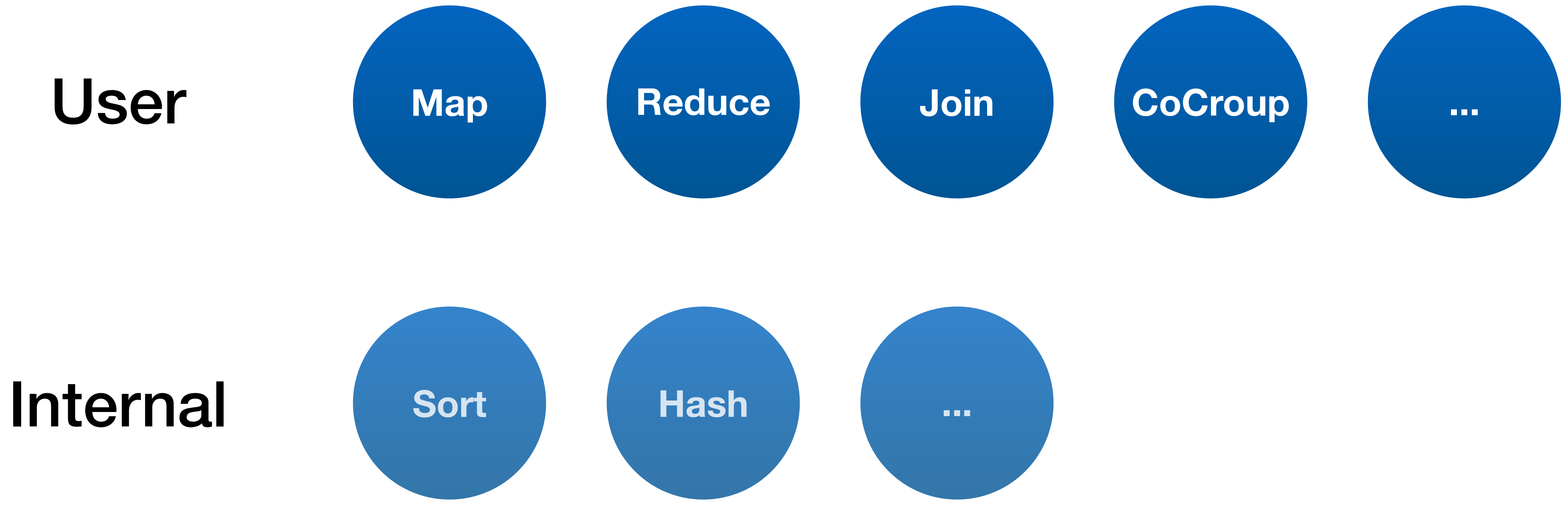
Apache Flink



Flink Runtime

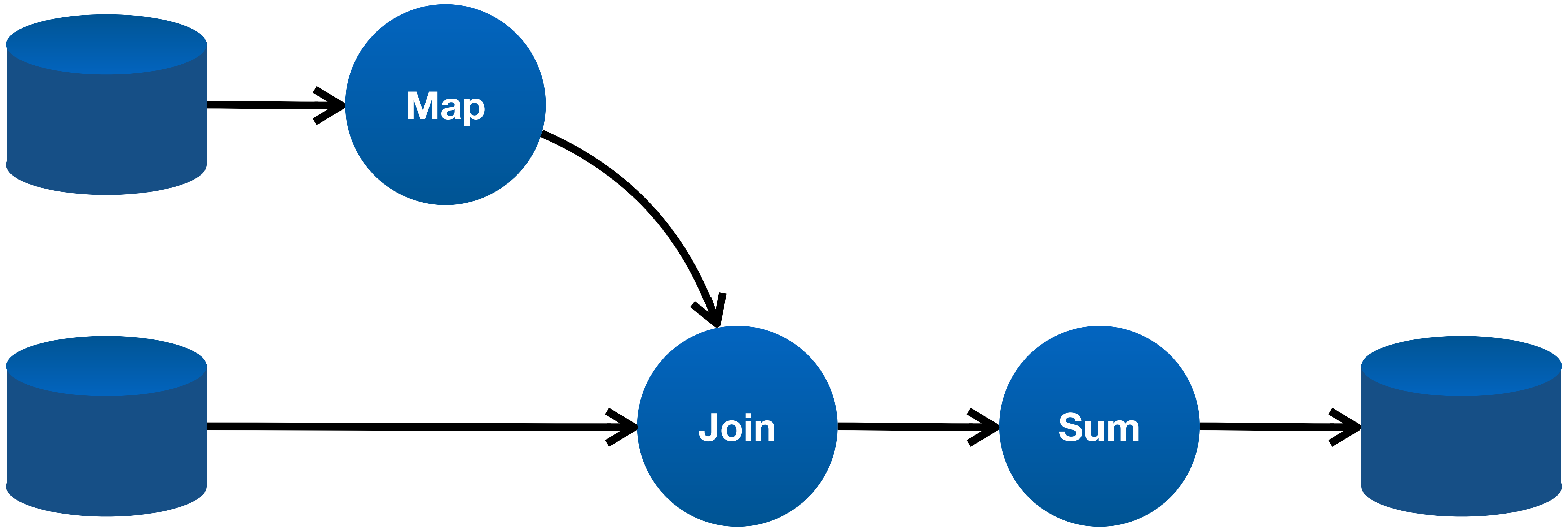


Operators



Operators represent computations over data.

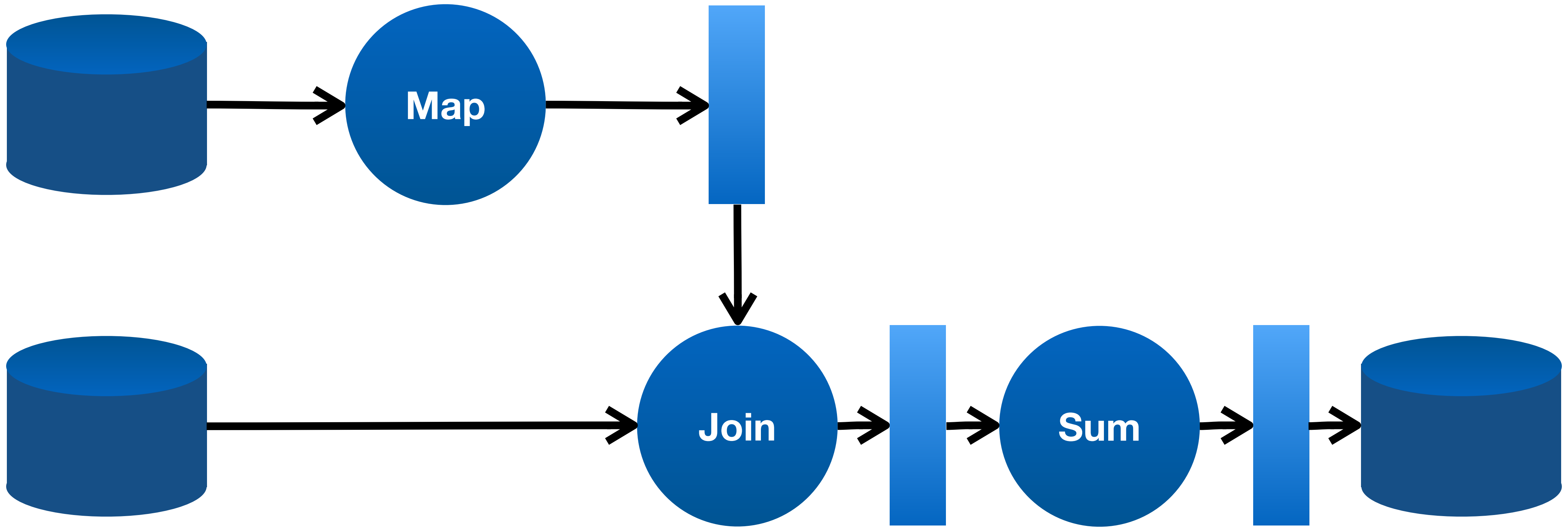
Operators



Intermediate Results

Logical handle to data produced by operators.

Intermediate Results

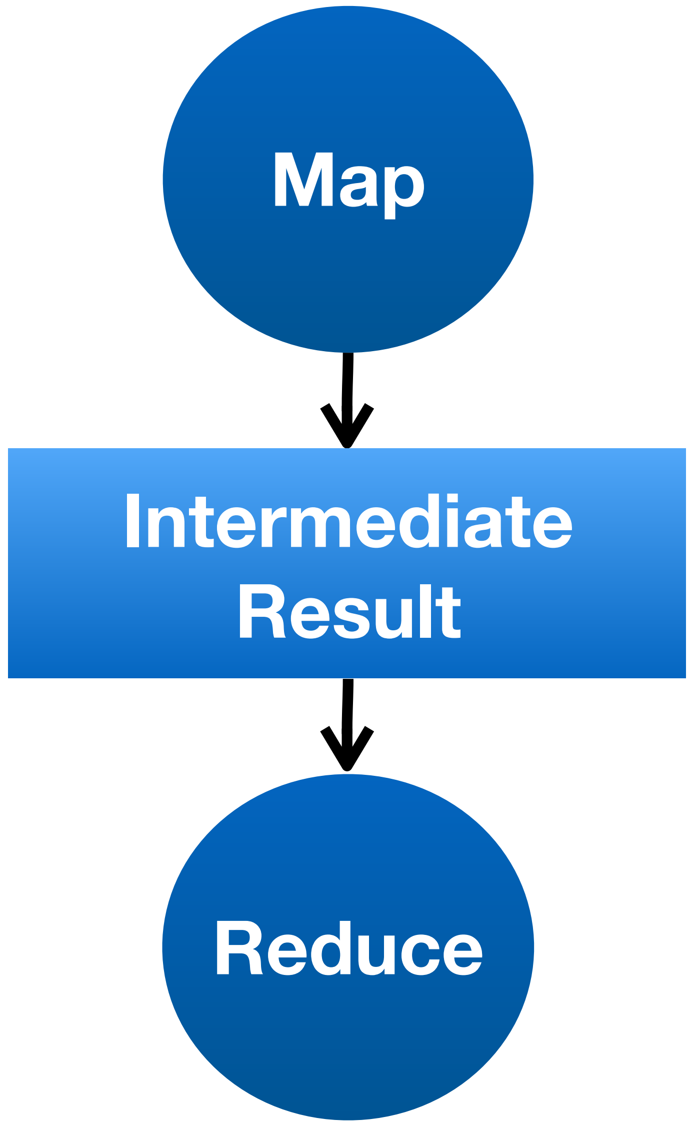


Logical handle to data **produced by operators.**

Map-Reduce Example

```
DataSet<Tuple2<String, Integer>>
  counts;

counts = input
  .flatMap(new LineSplitter())
  .groupBy(0)
  .sum(1);
```



Logical handle to data produced by operators.

Result Characteristics

Pipelined vs. **Blocking**

Ephemeral vs. **Checkpointed**

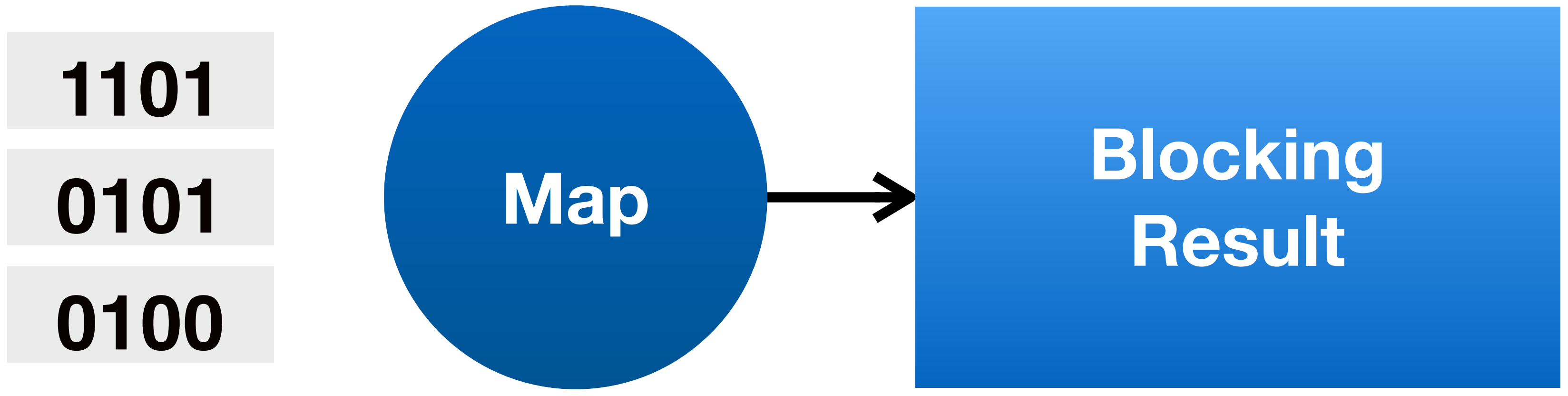
Result Characteristics

Pipelined vs. **Blocking**

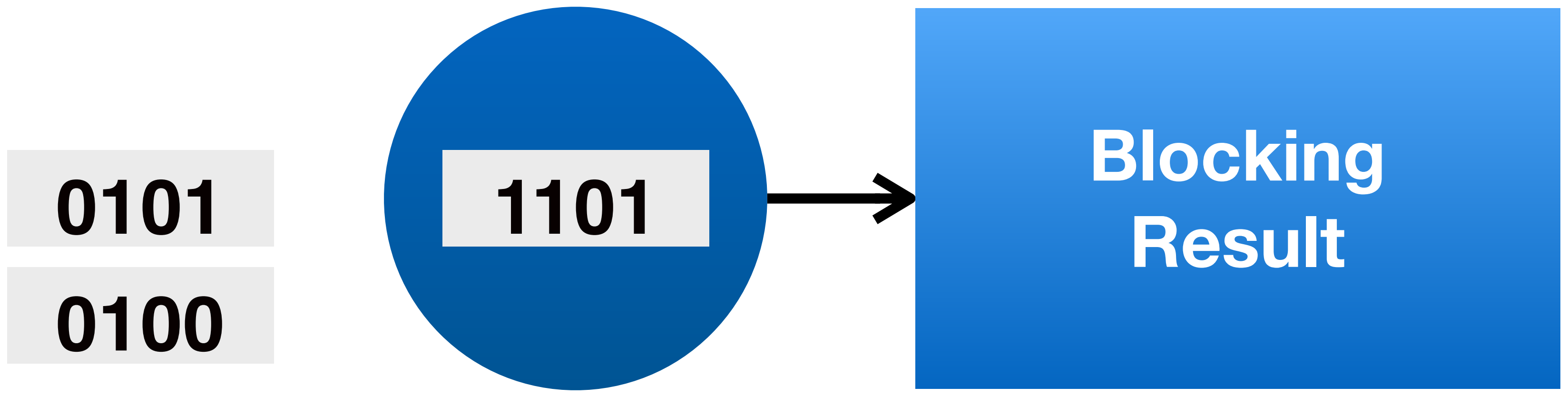
Ephemeral vs. Checkpointed

How and when to do data exchange?

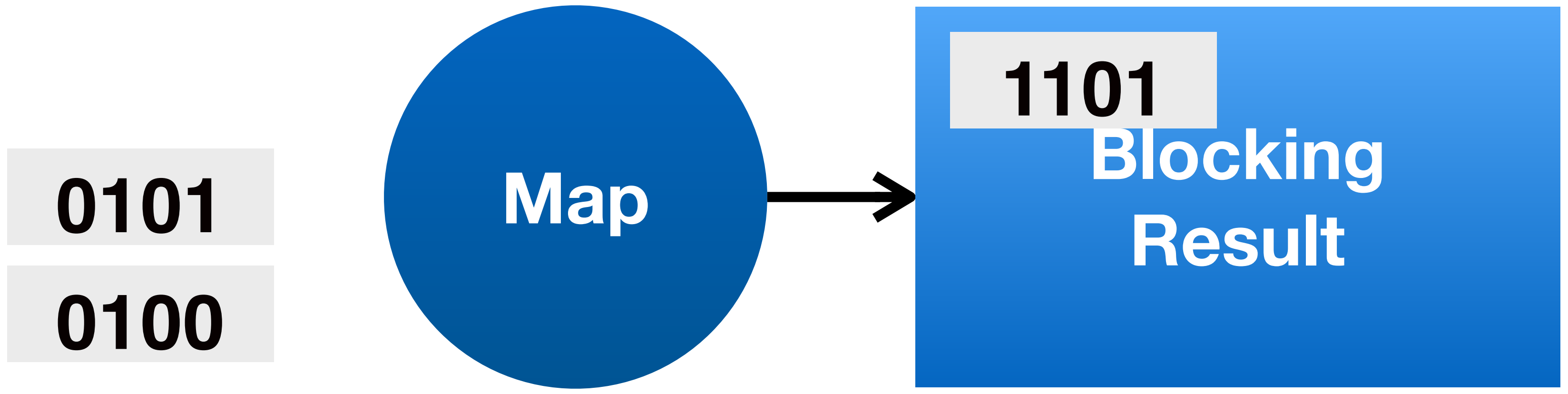
Blocking Results



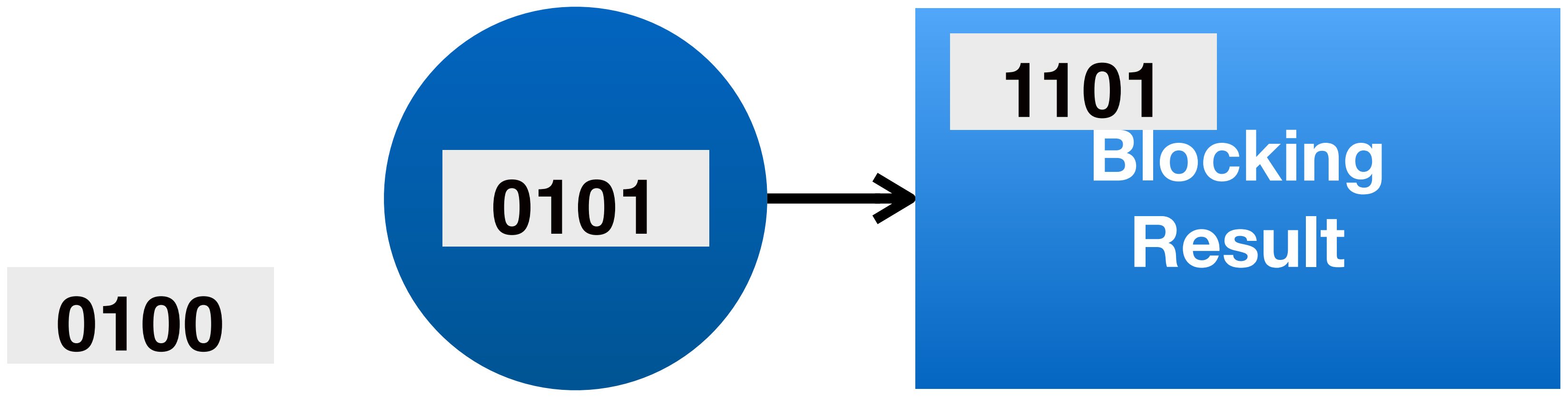
Blocking Results



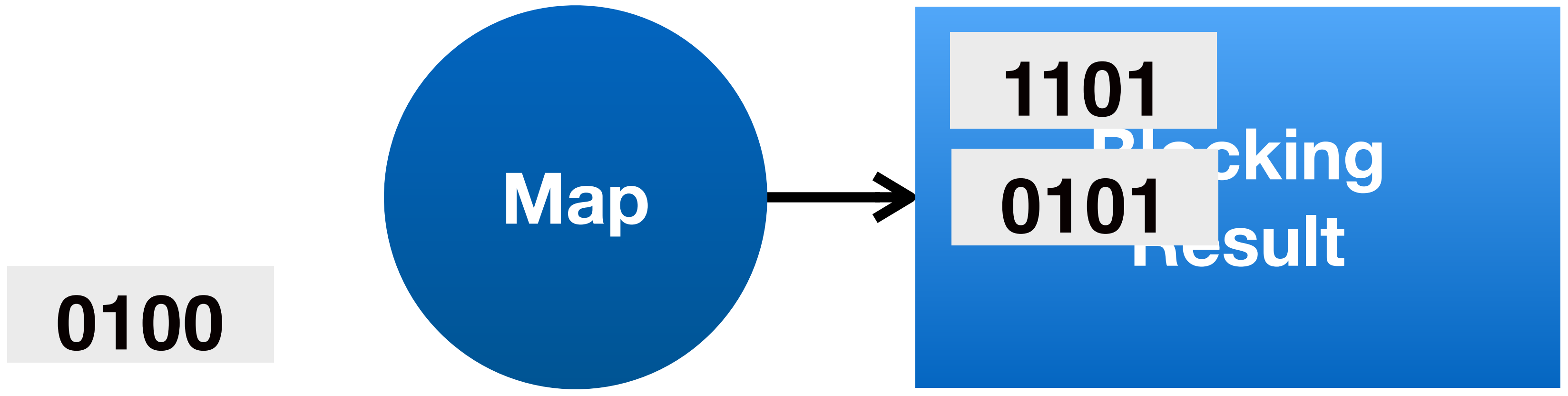
Blocking Results



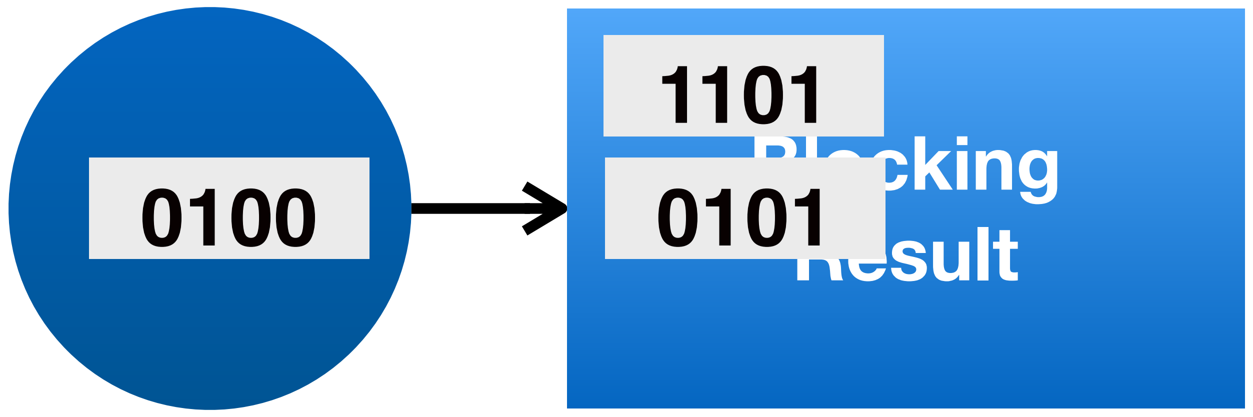
Blocking Results



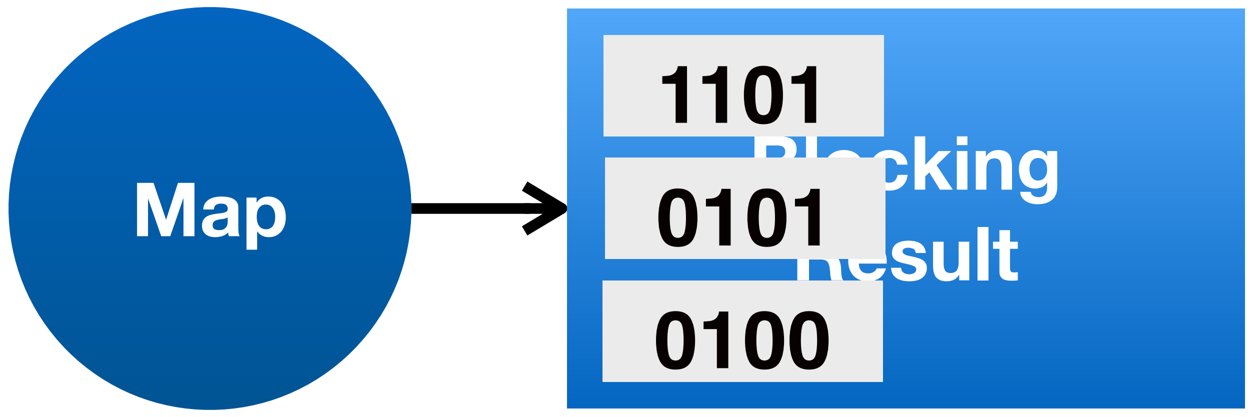
Blocking Results



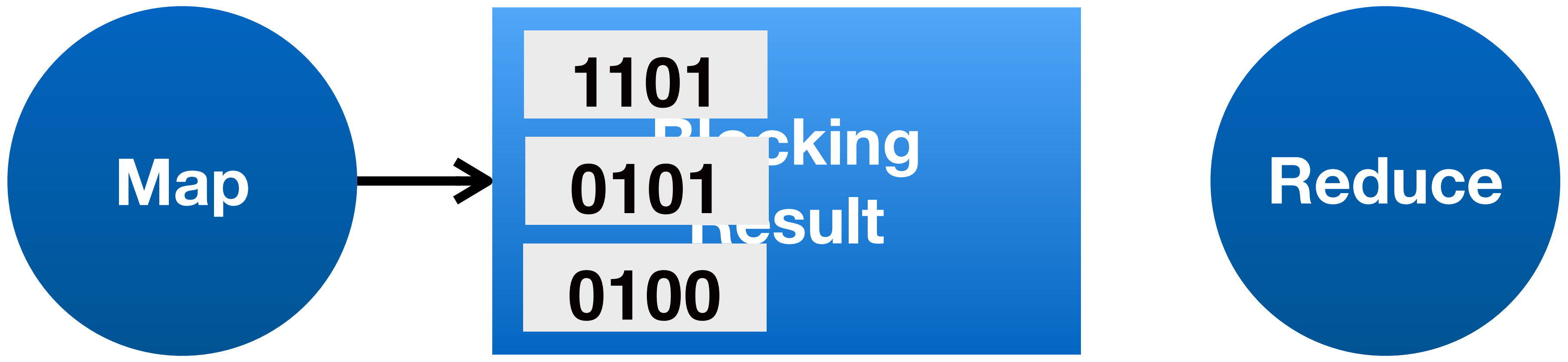
Blocking Results



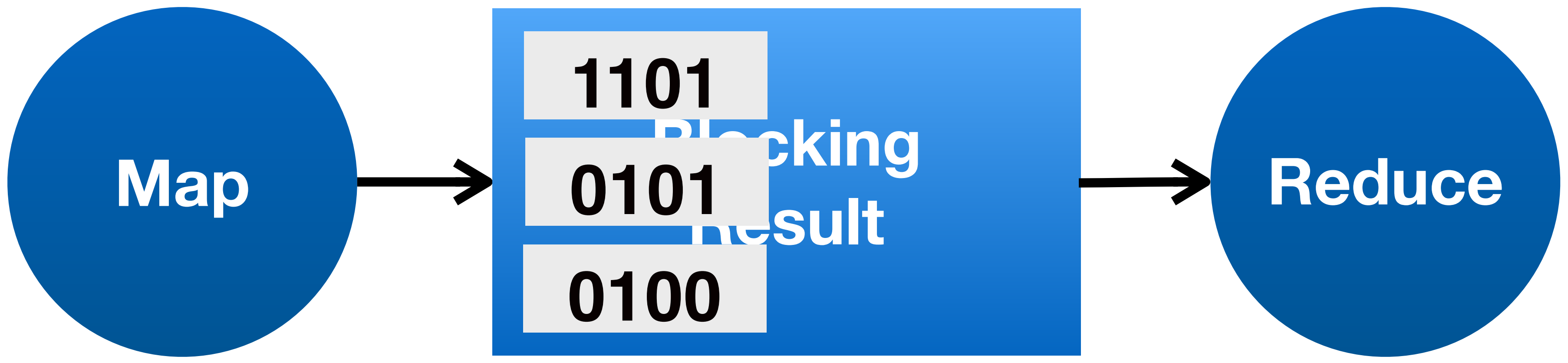
Blocking Results



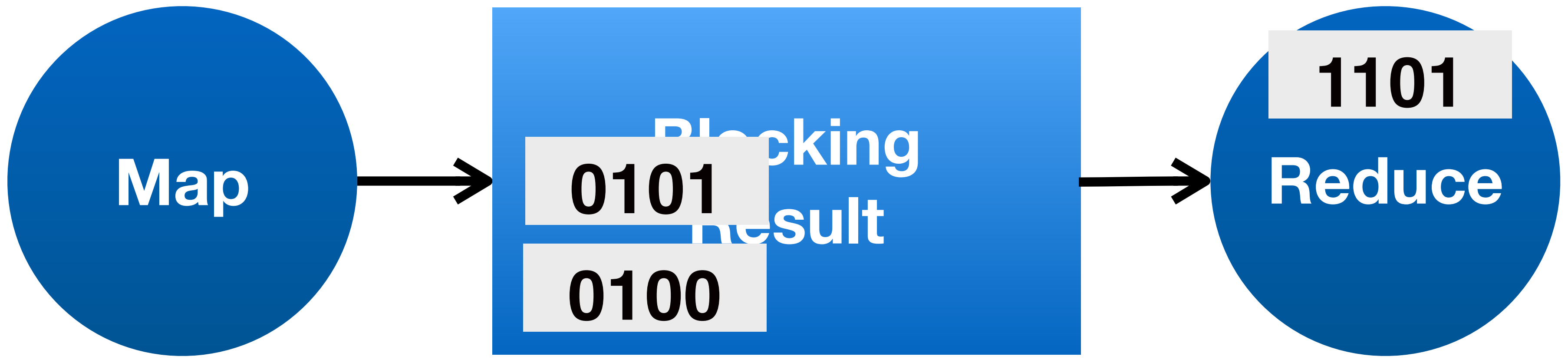
Blocking Results



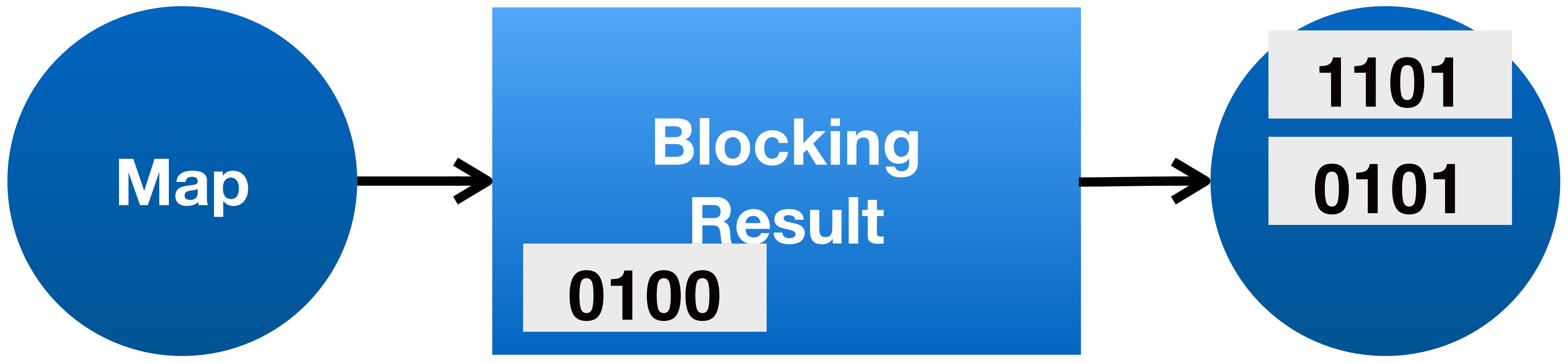
Blocking Results



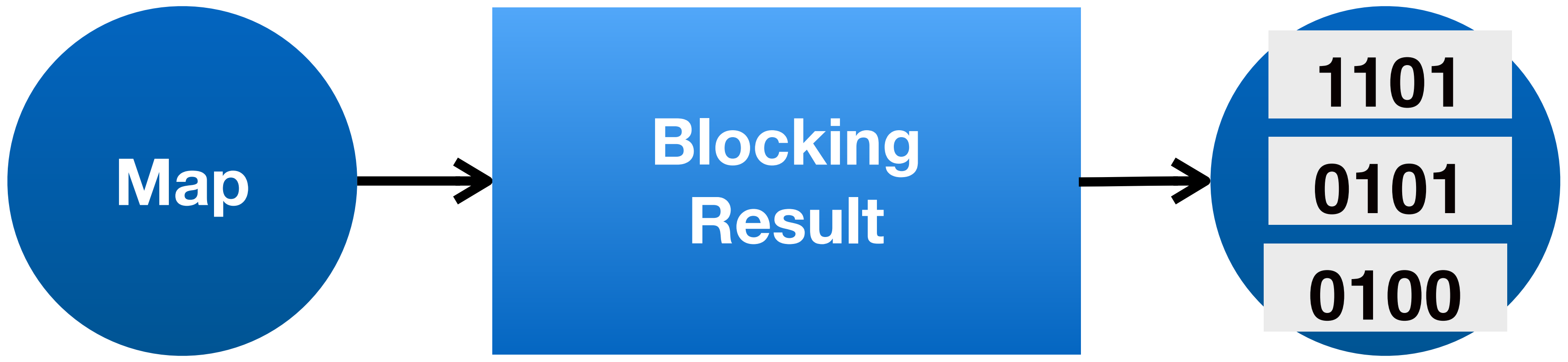
Blocking Results



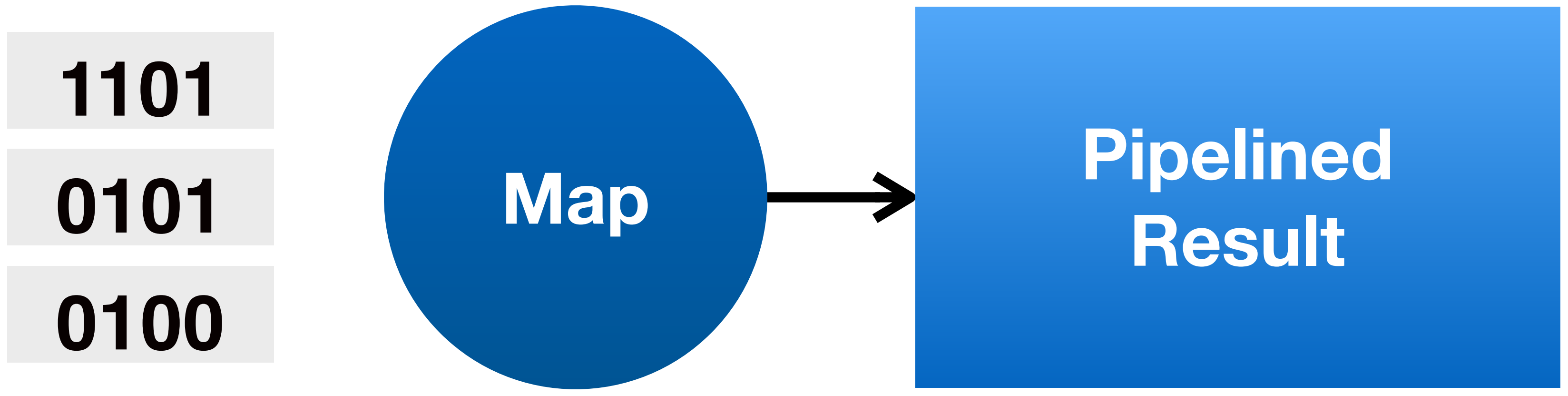
Blocking Results



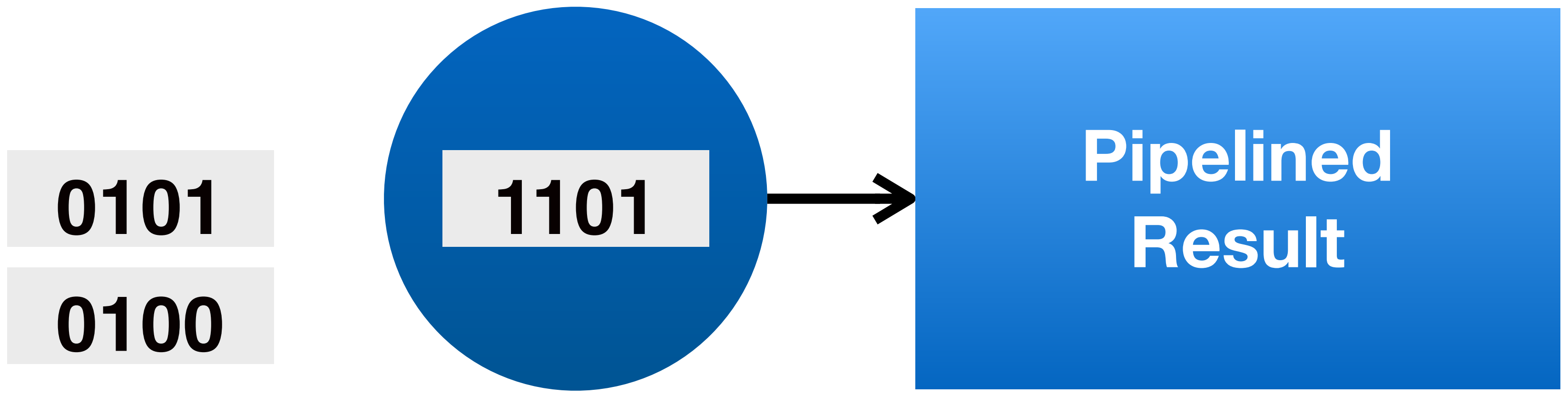
Blocking Results



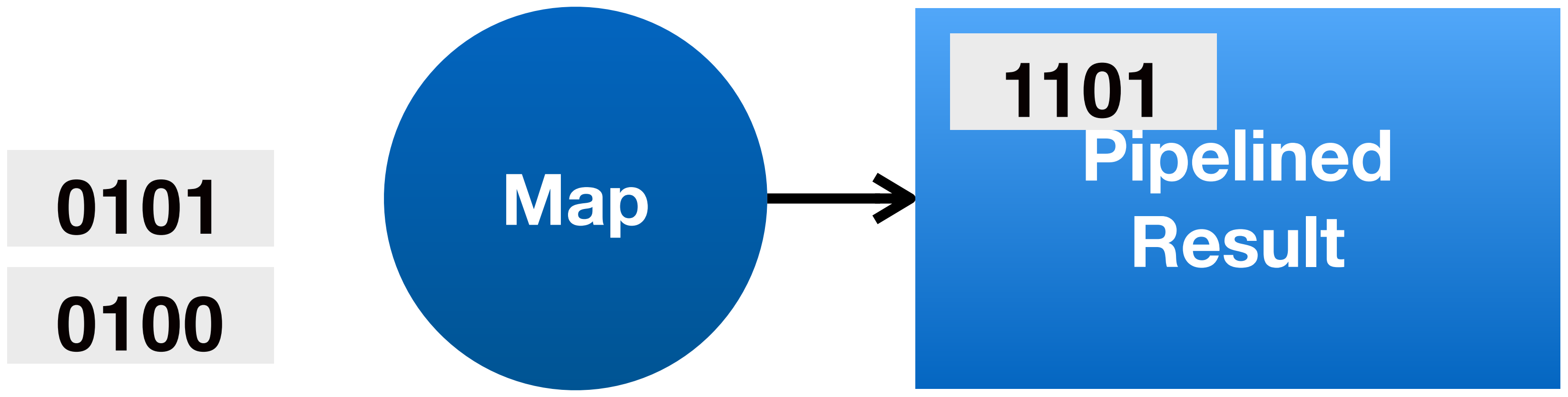
Pipelined Results



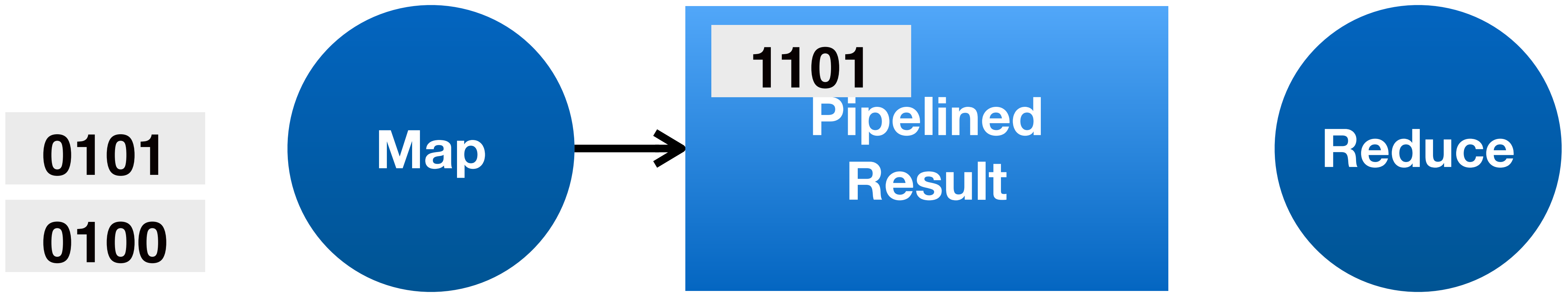
Pipelined Results



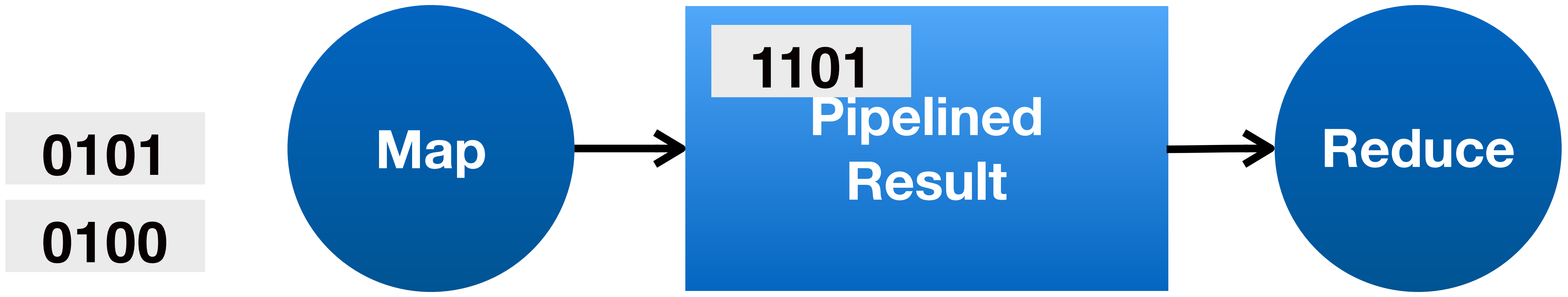
Pipelined Results



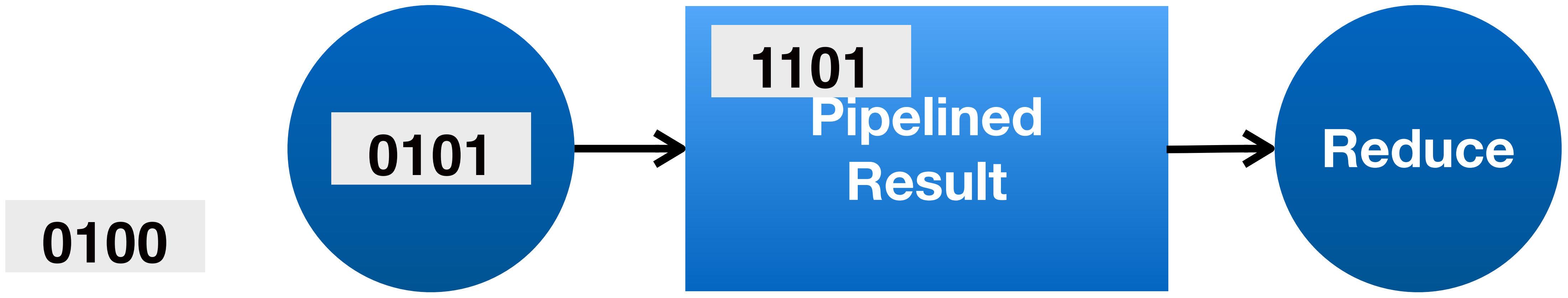
Pipelined Results



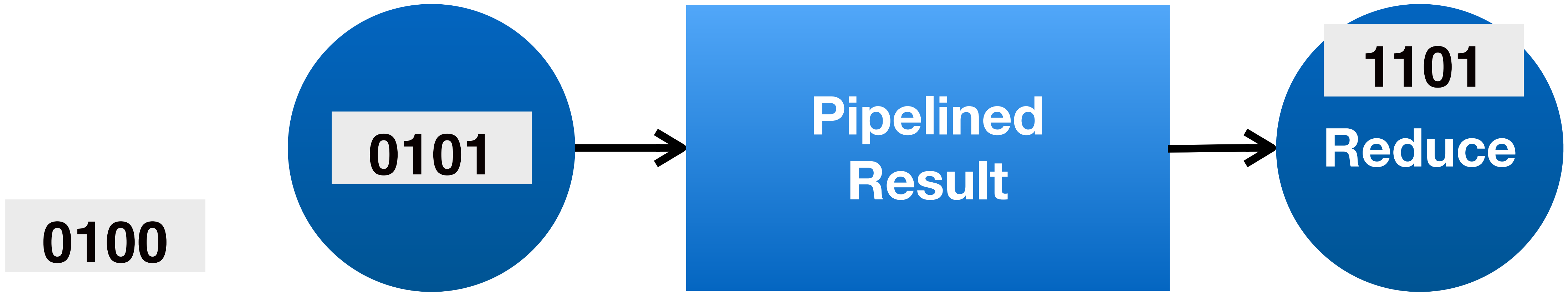
Pipelined Results



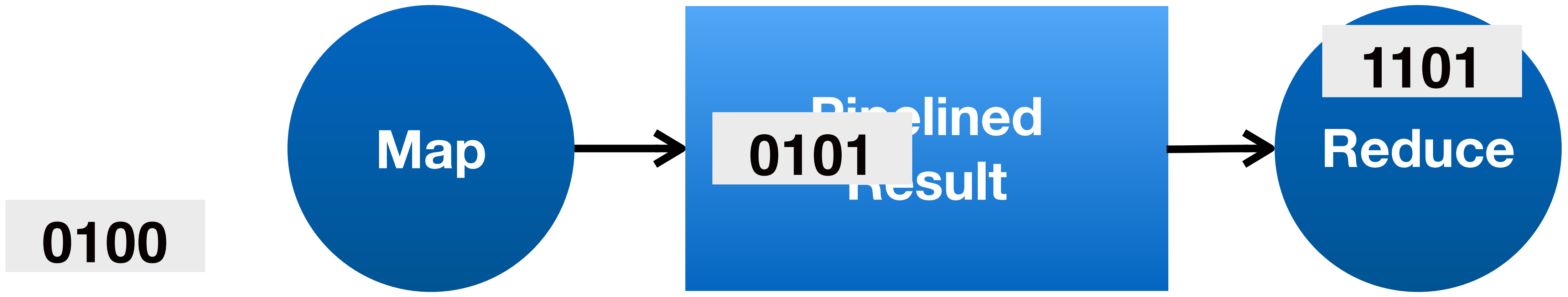
Pipelined Results



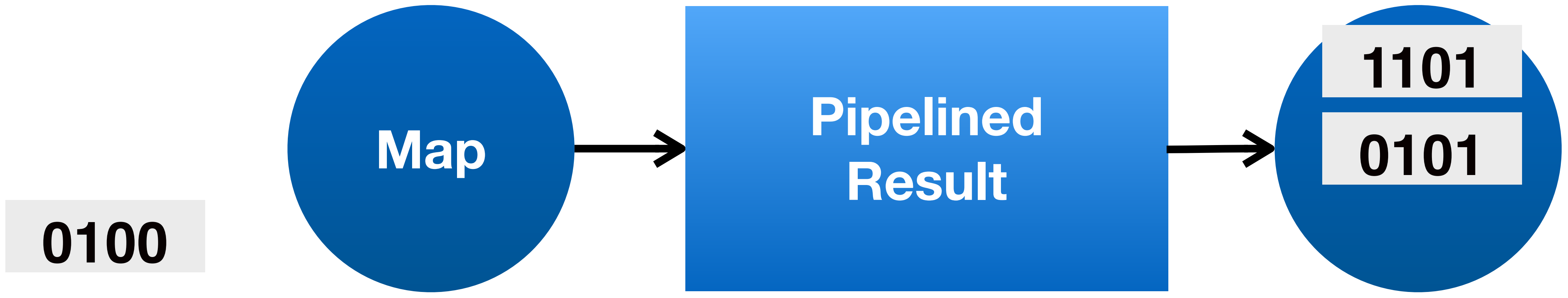
Pipelined Results



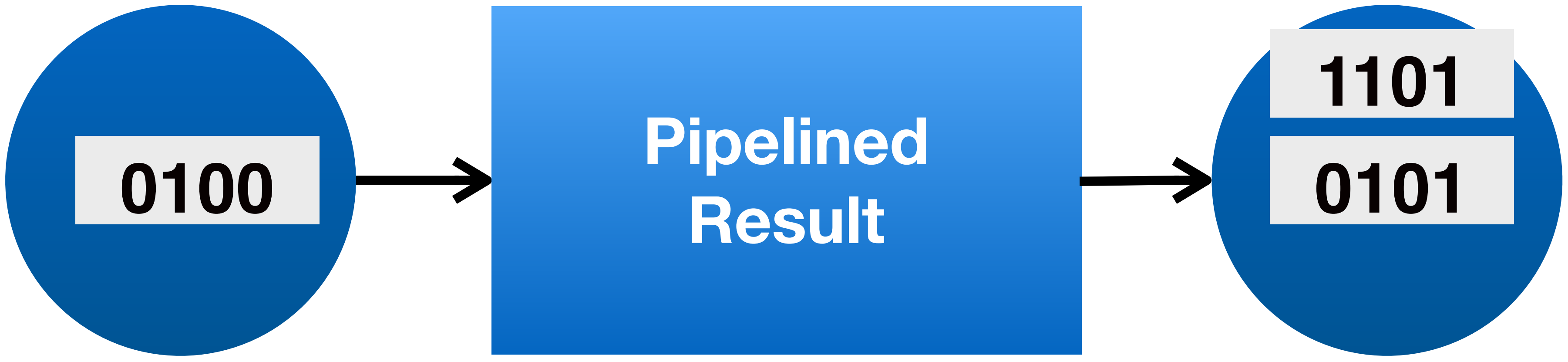
Pipelined Results



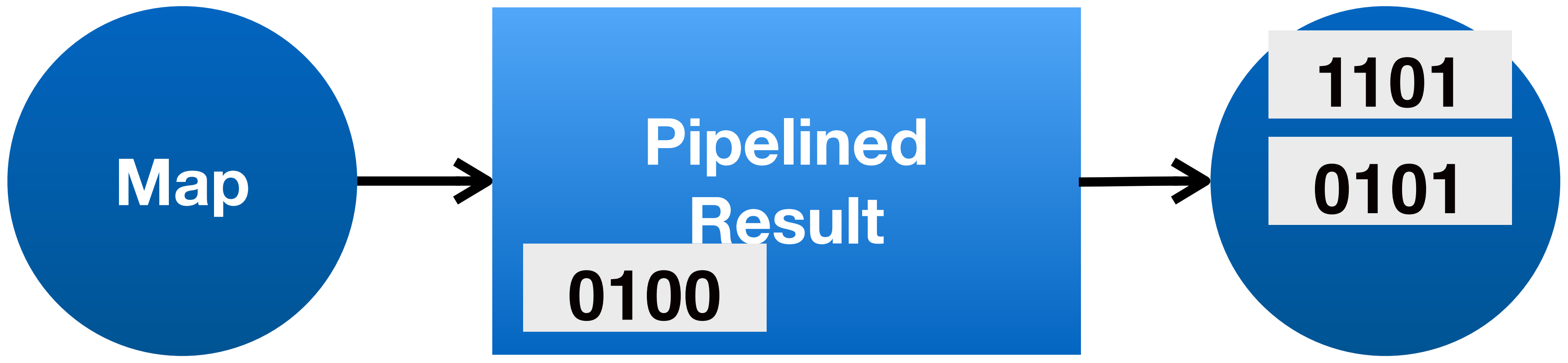
Pipelined Results



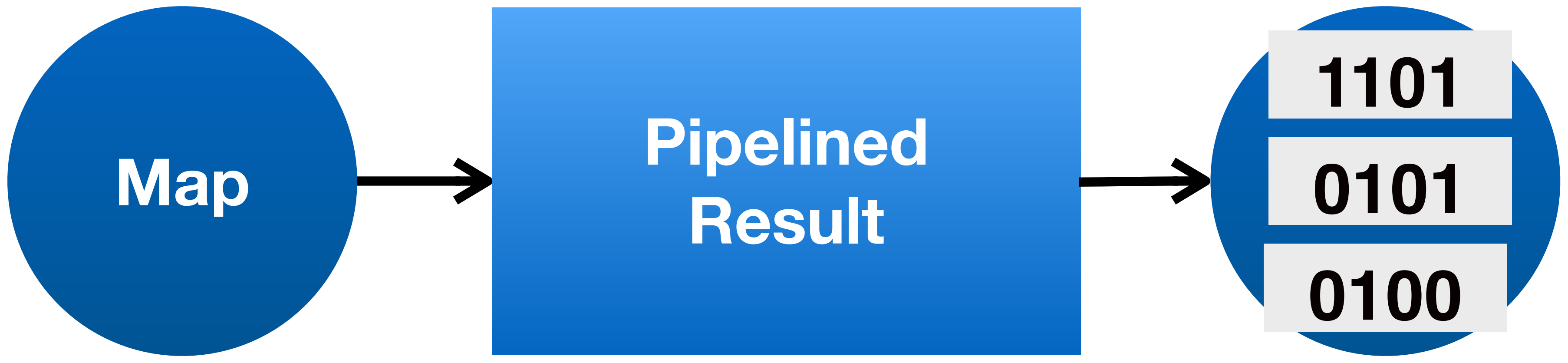
Pipelined Results



Pipelined Results



Pipelined Results



Result Characteristics

	Ephemeral	Checkpointed
Pipelined	Low-latency	Low-latency
Blocking	Easy to reason about resource consumption	Easy to reason about resource consumption

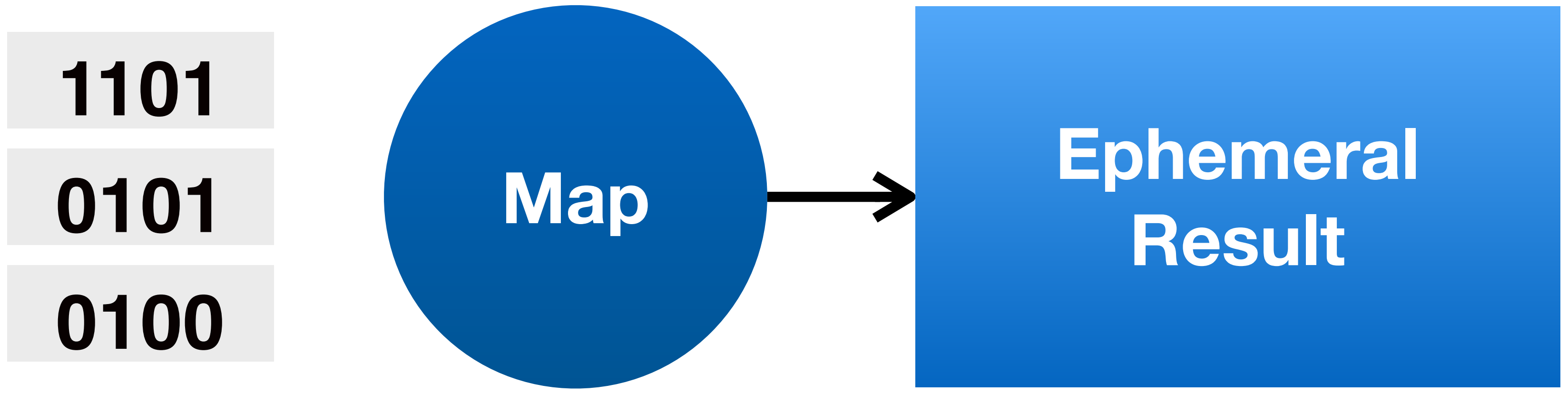
Result Characteristics

Pipelined vs. Blocking

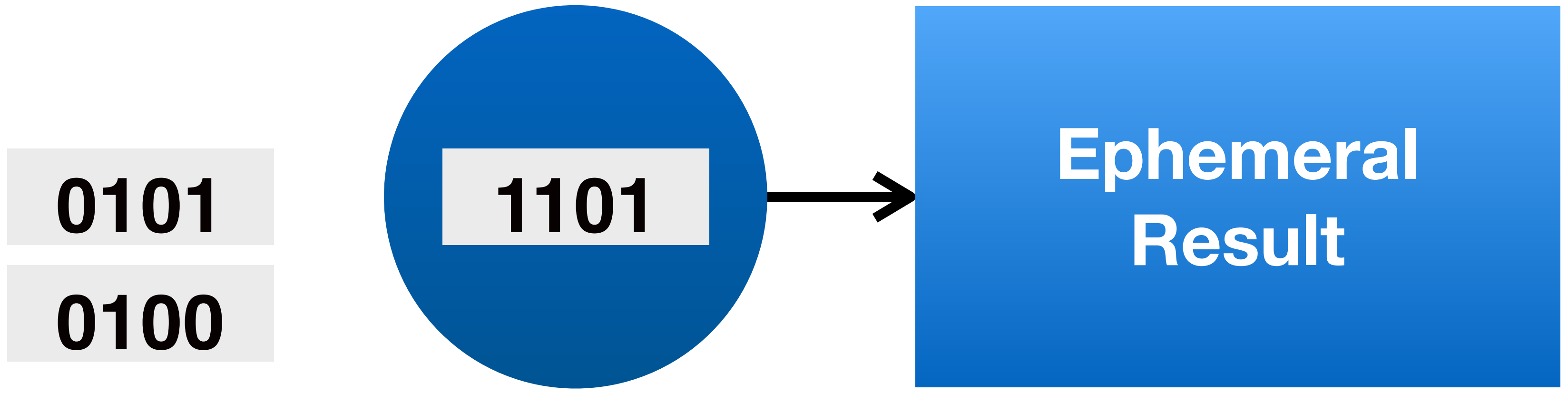
Ephemeral vs. **Checkpointed**

How long to keep results around?

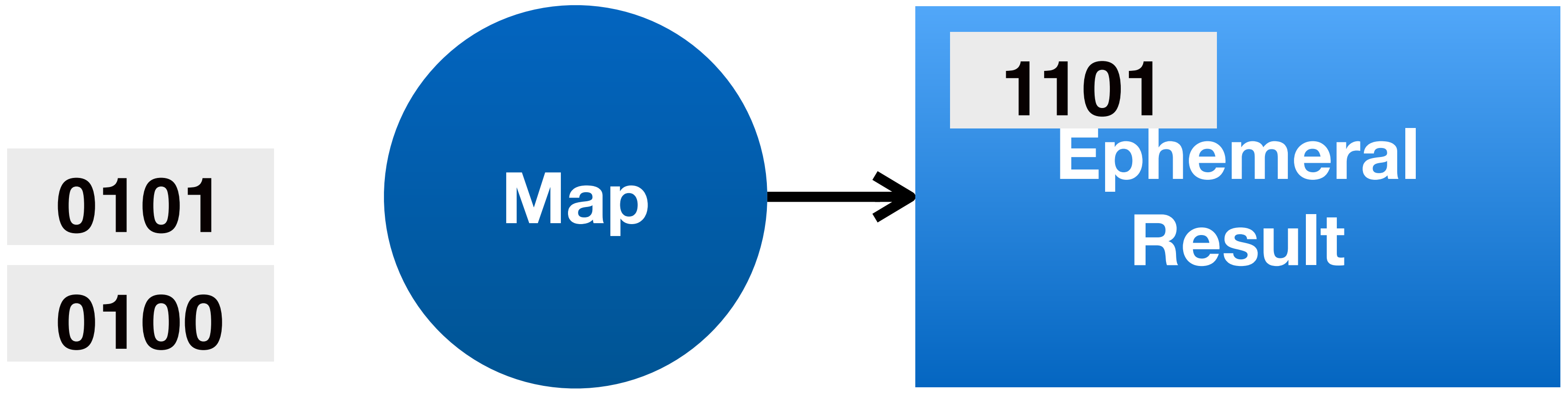
Ephemeral Results



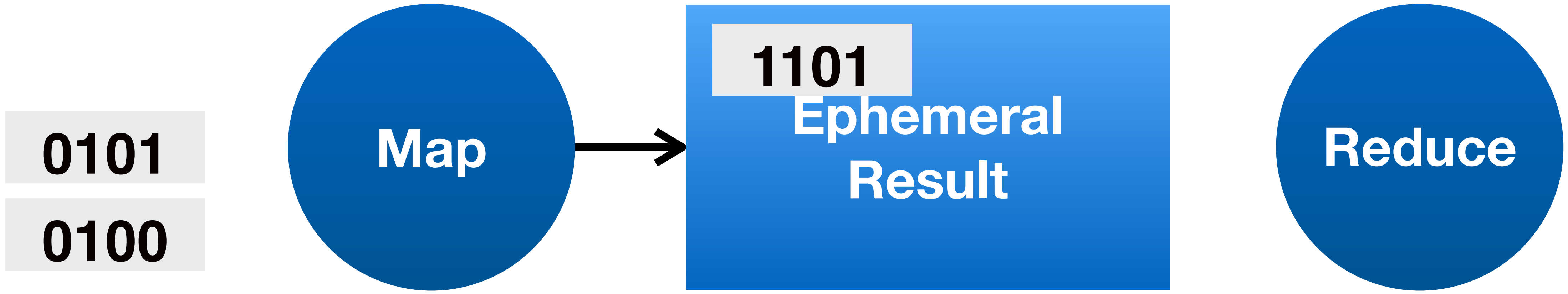
Ephemeral Results



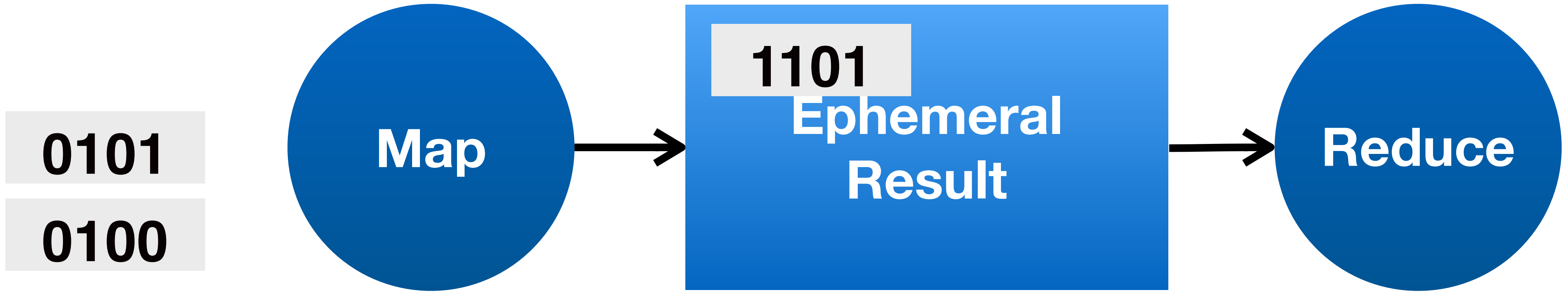
Ephemeral Results



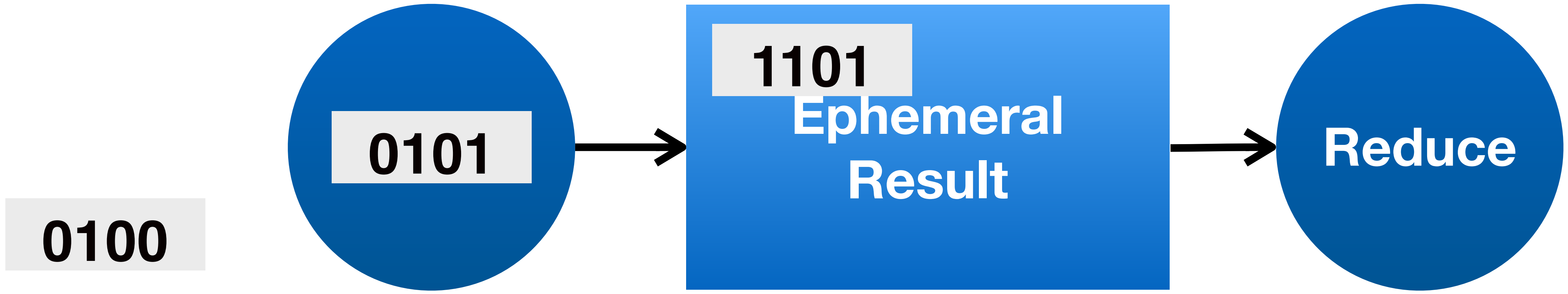
Ephemeral Results



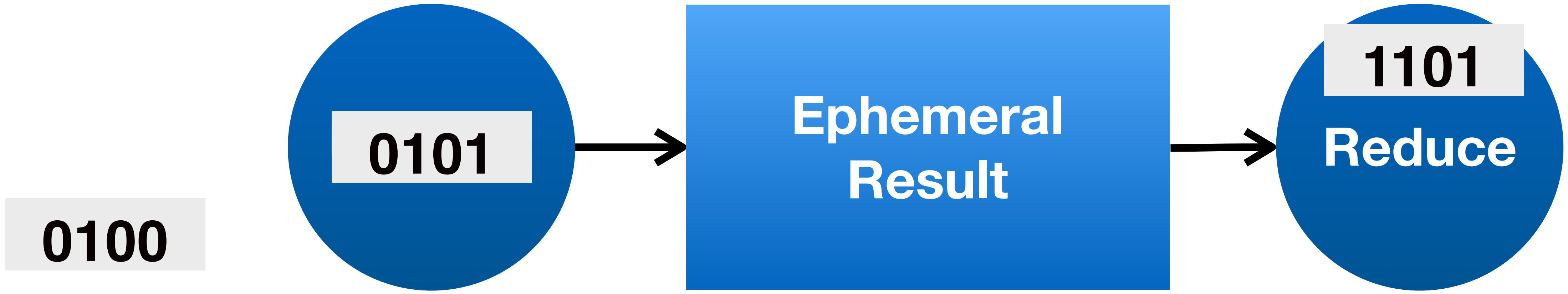
Ephemeral Results



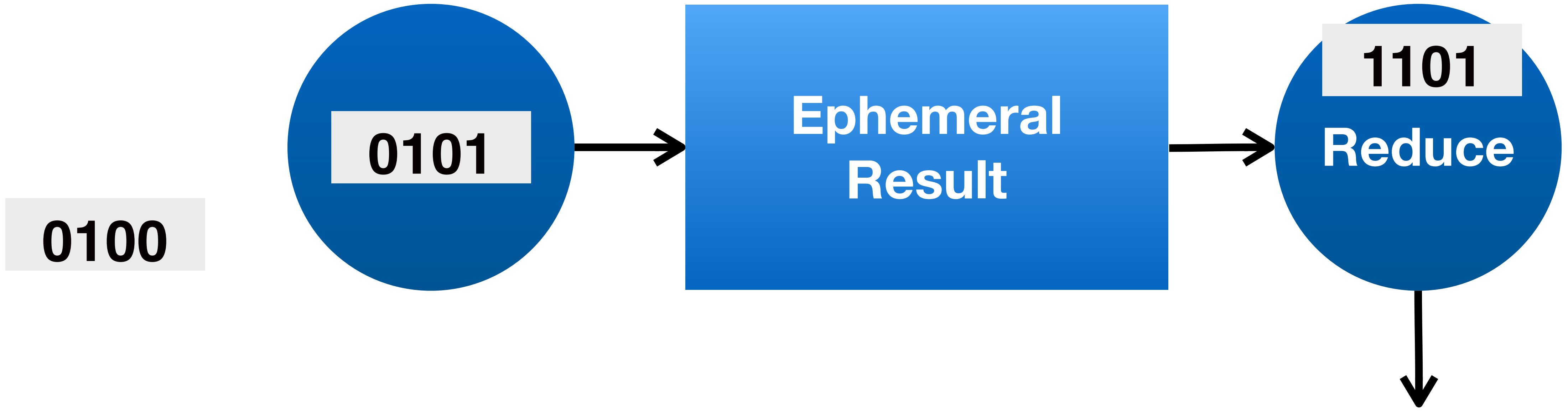
Ephemeral Results



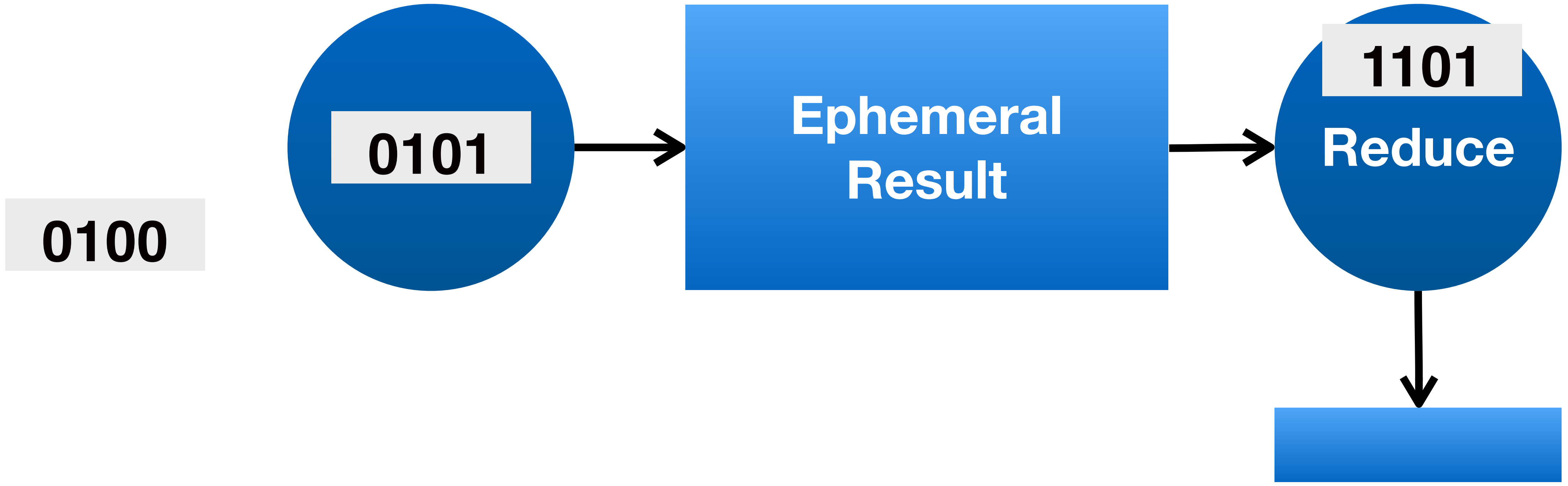
Ephemeral Results



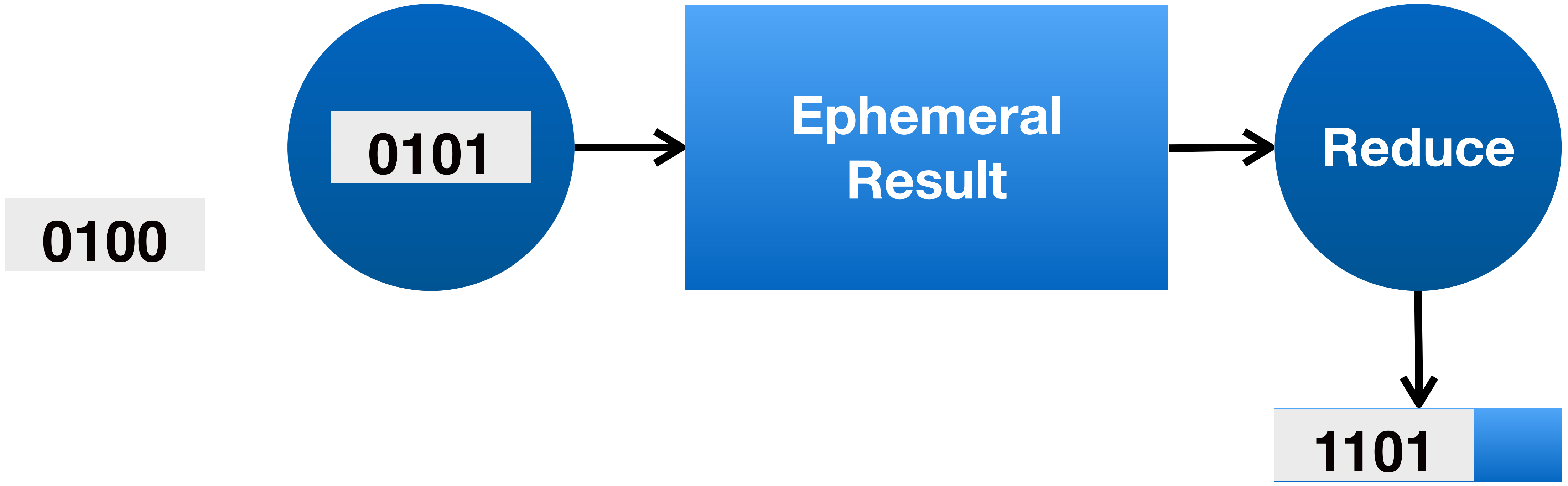
Ephemeral Results



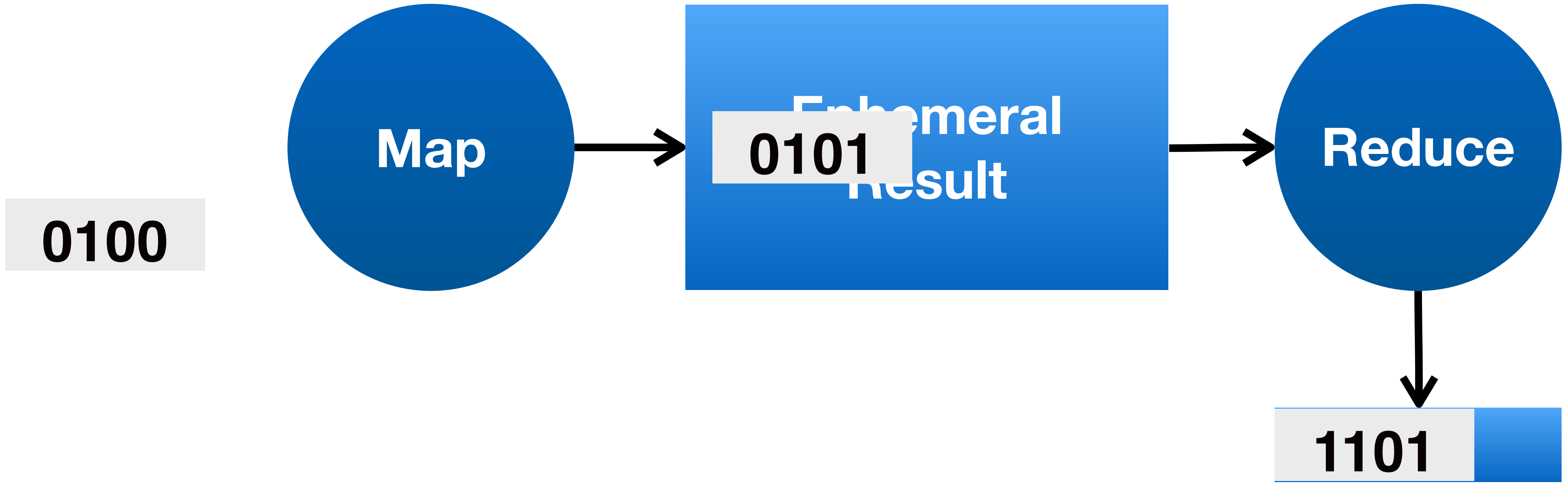
Ephemeral Results



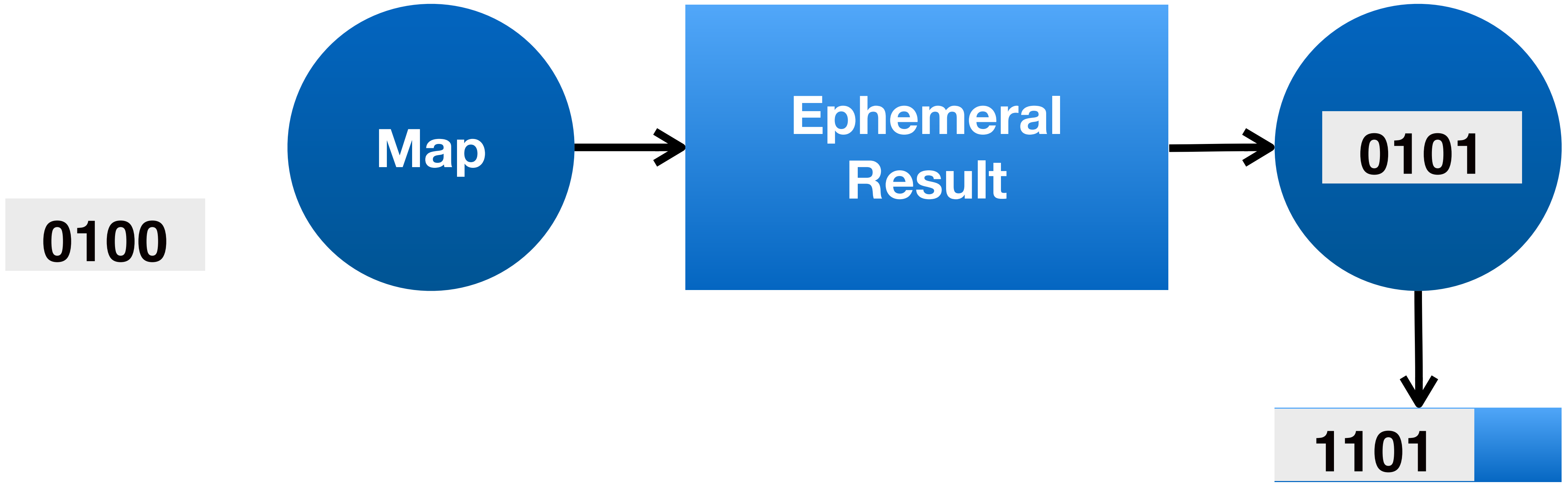
Ephemeral Results



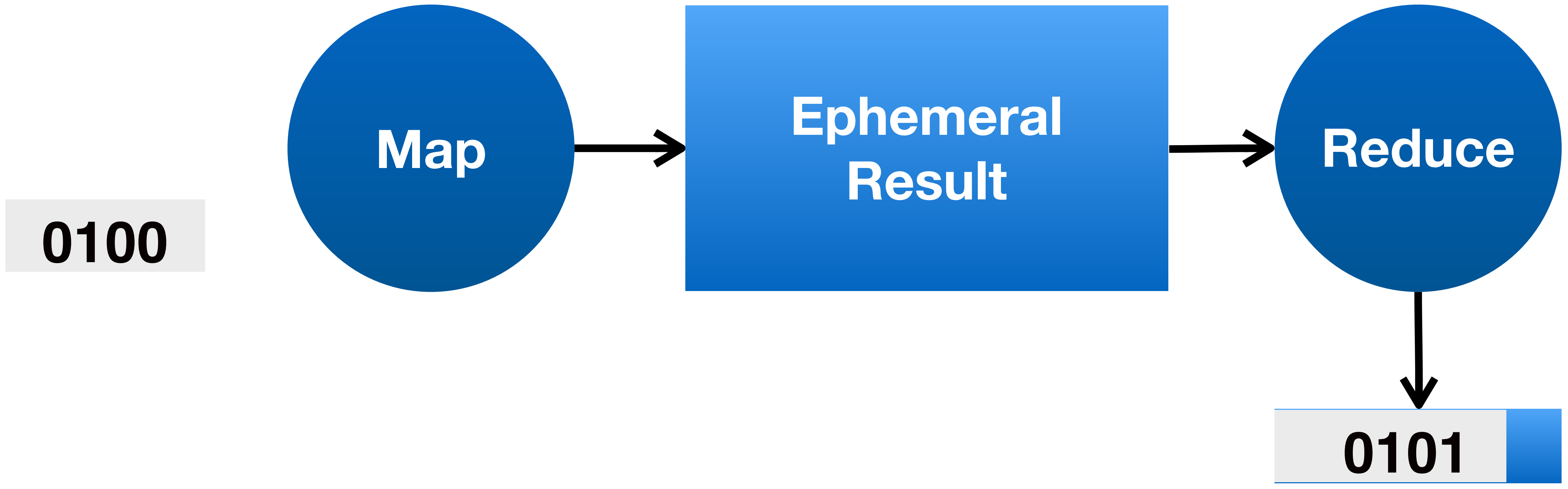
Ephemeral Results



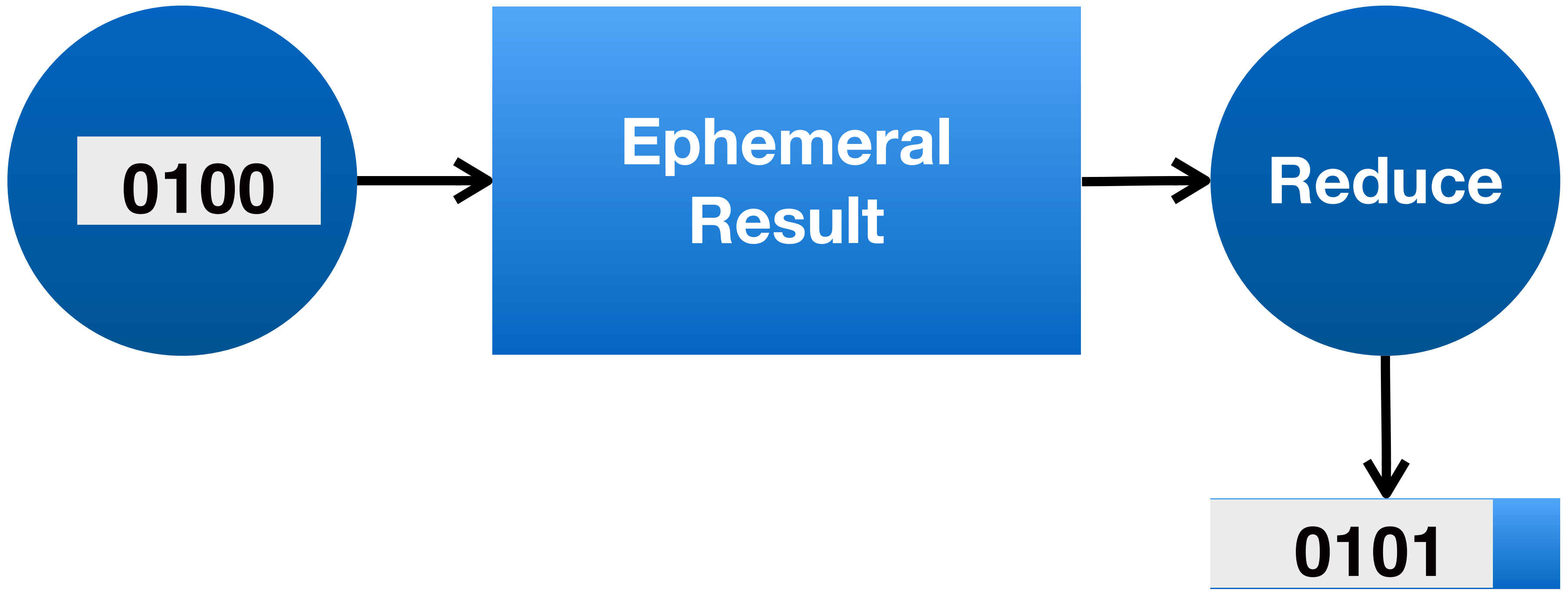
Ephemeral Results



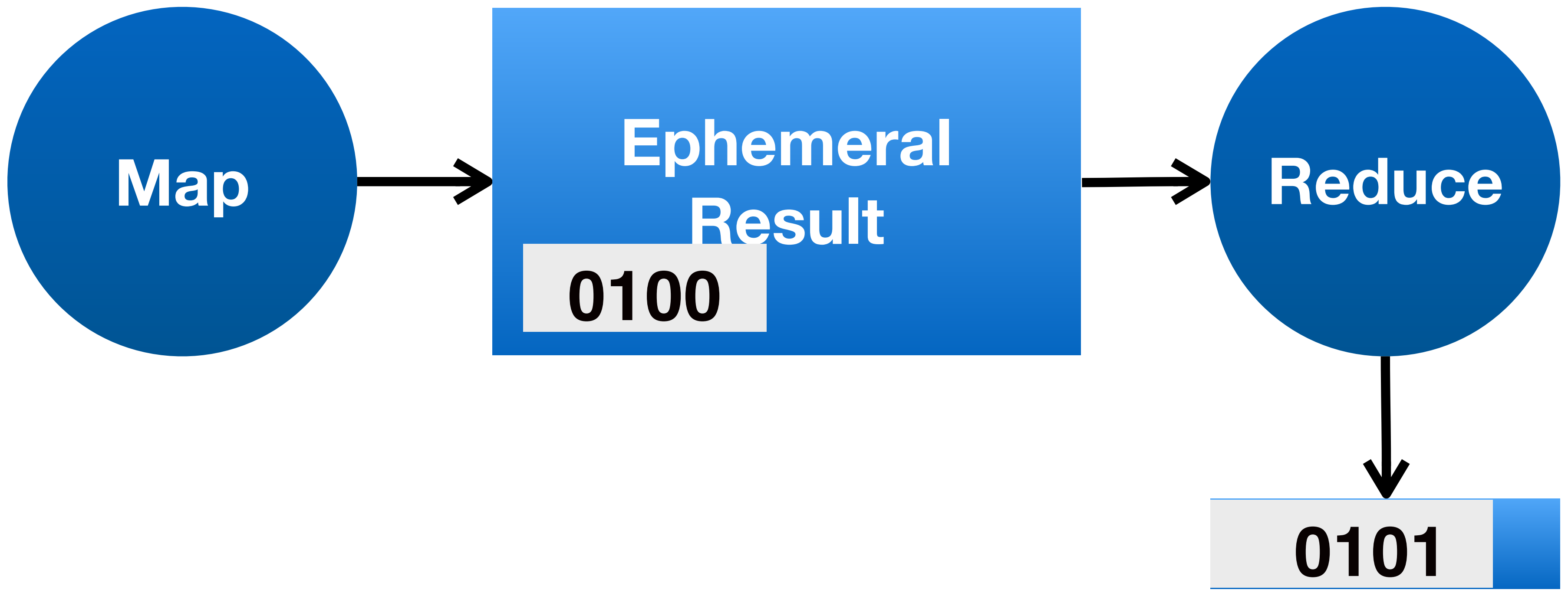
Ephemeral Results



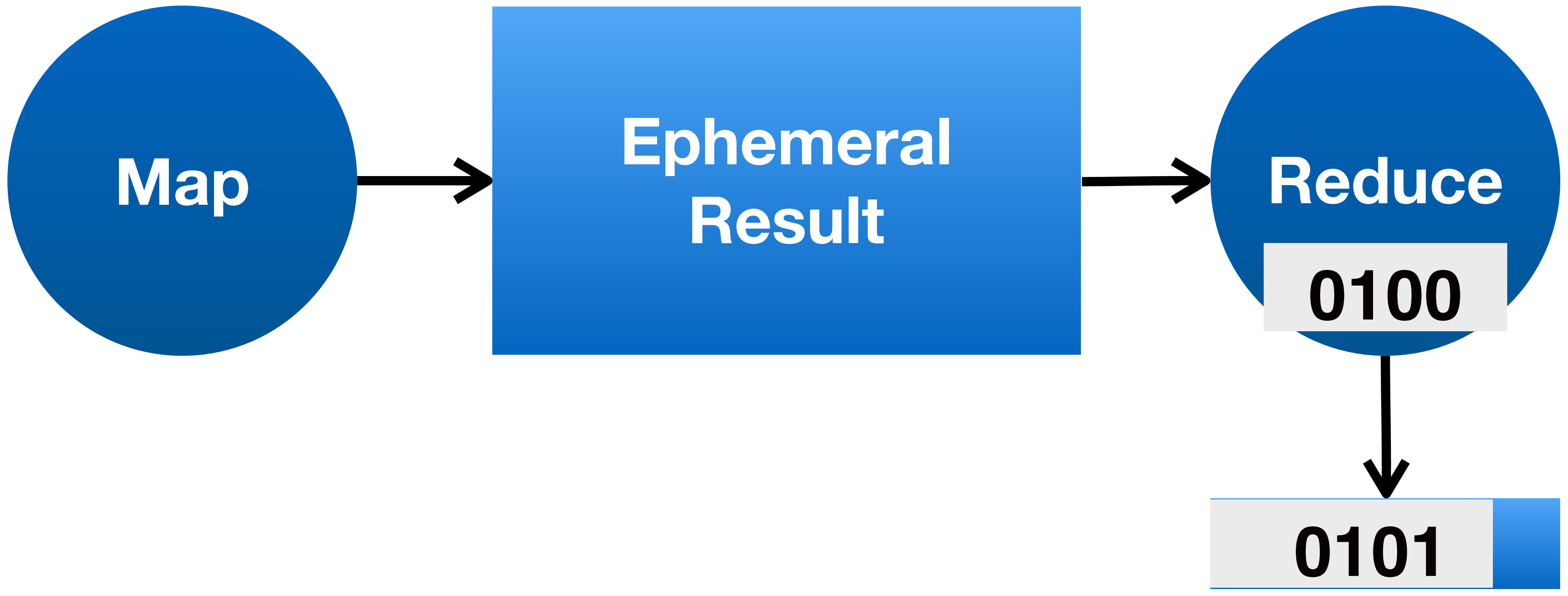
Ephemeral Results



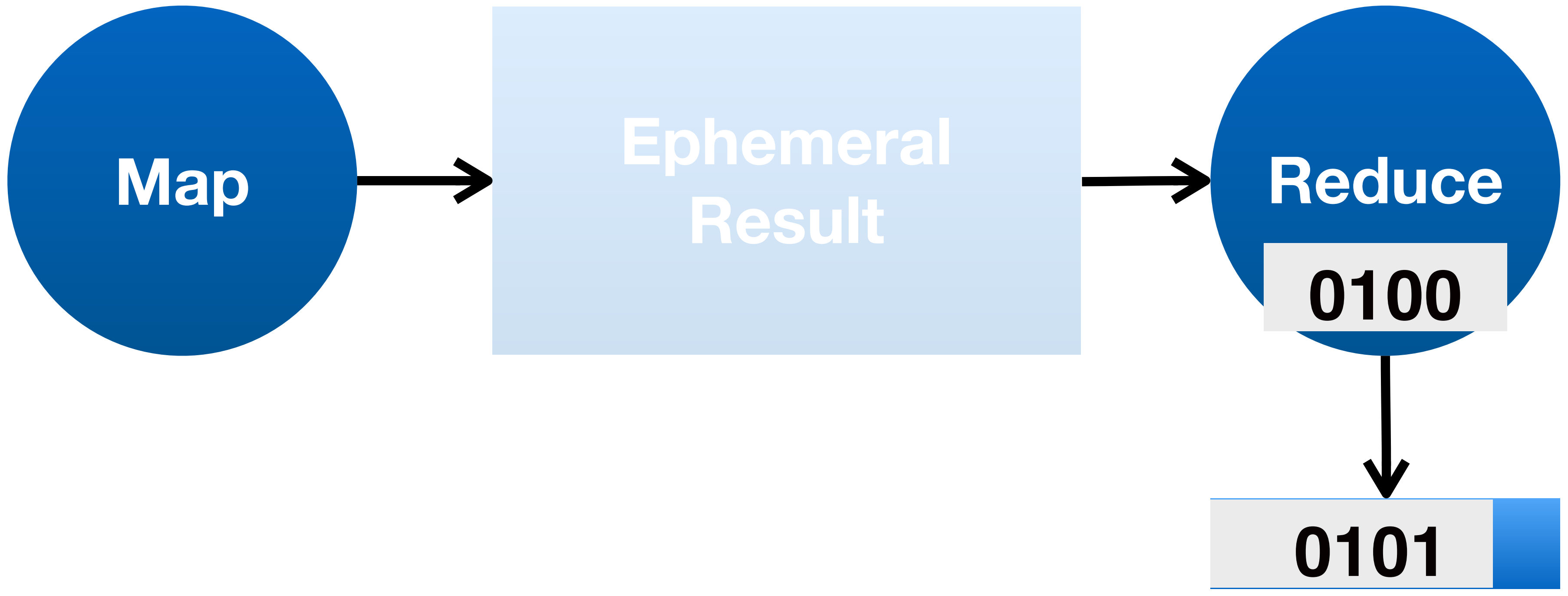
Ephemeral Results



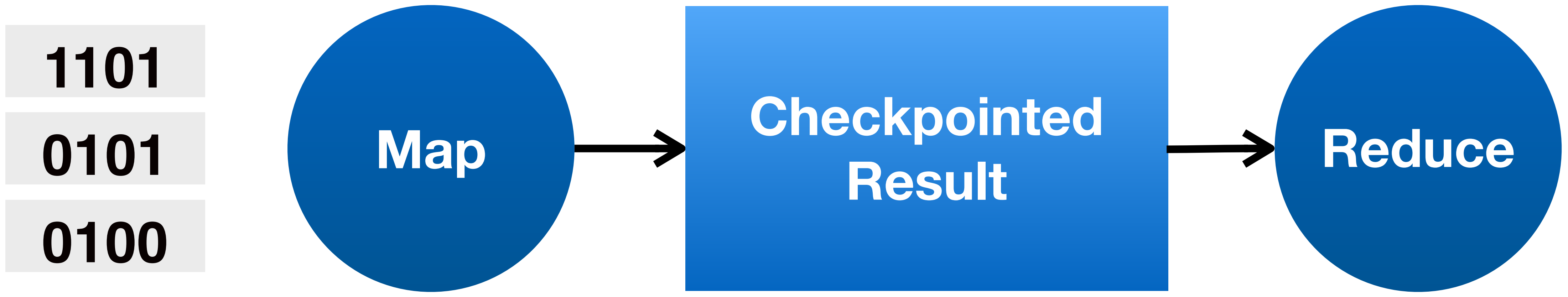
Ephemeral Results



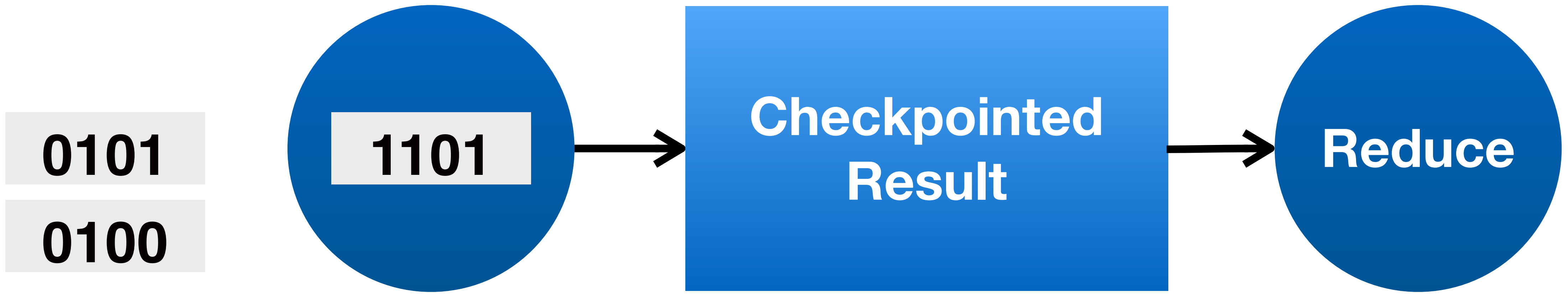
Ephemeral Results



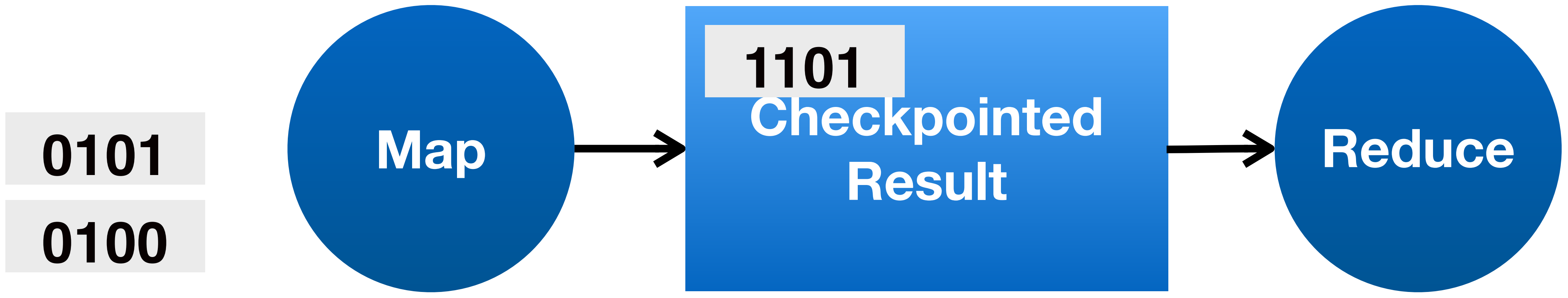
Checkpointed Results



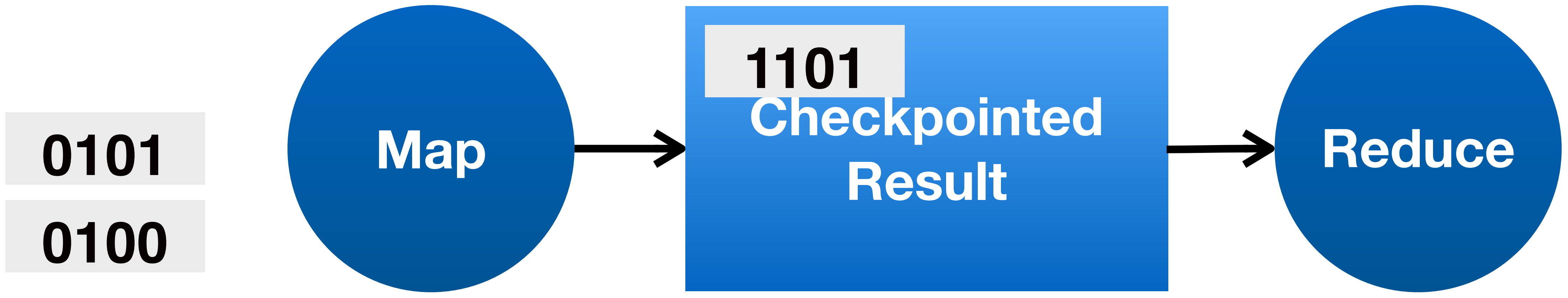
Checkpointed Results



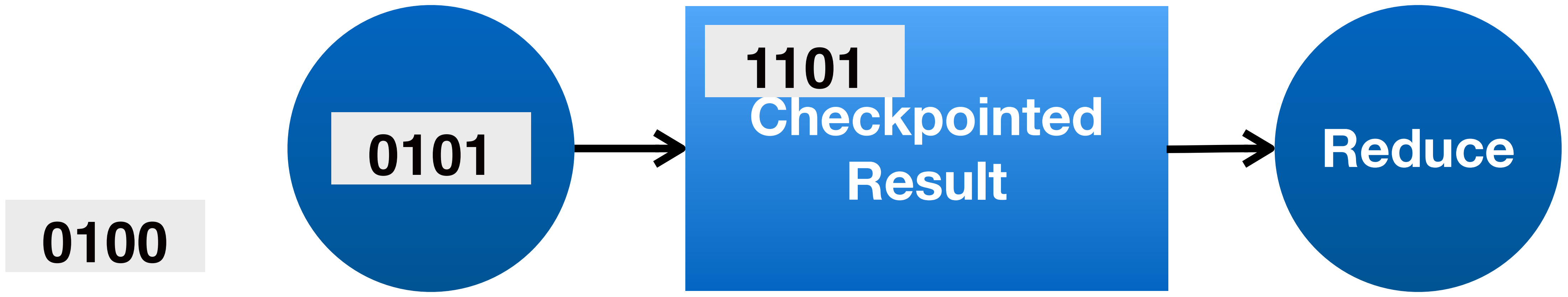
Checkpointed Results



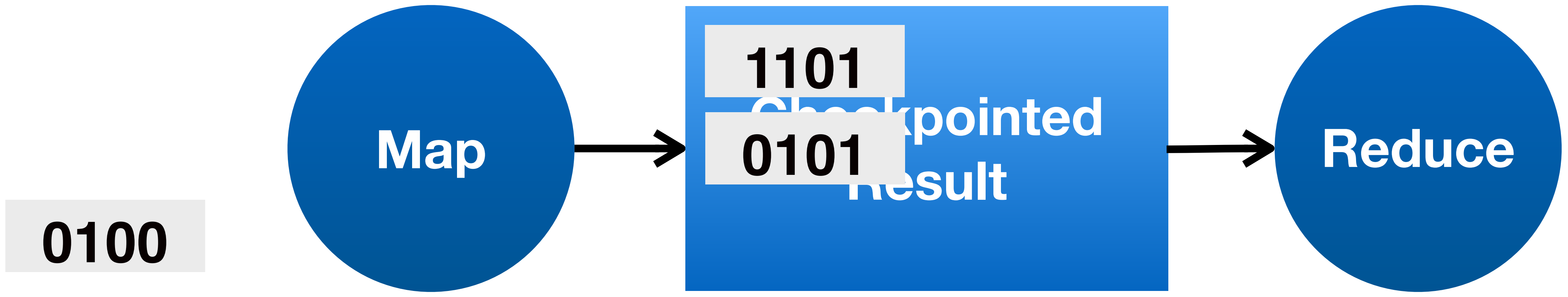
Checkpointed Results



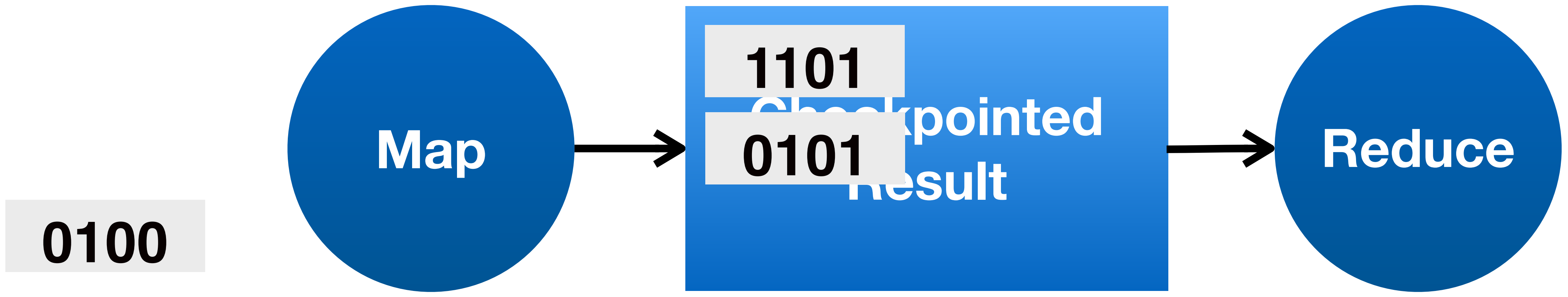
Checkpointed Results



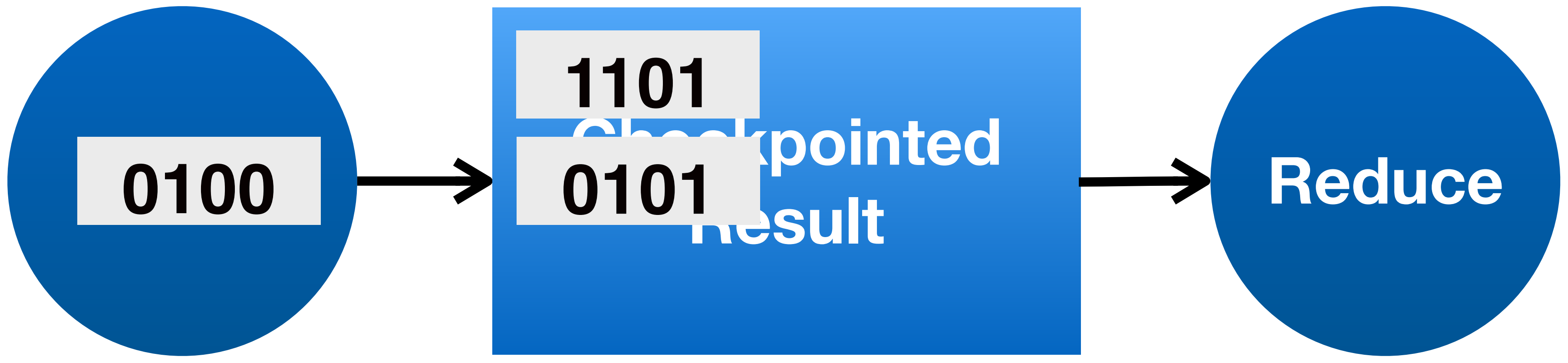
Checkpointed Results



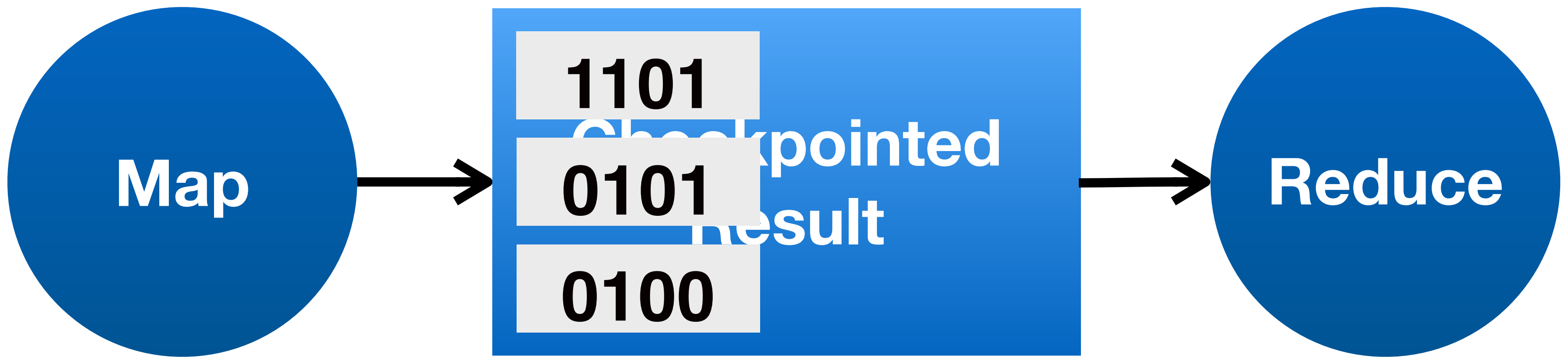
Checkpointed Results



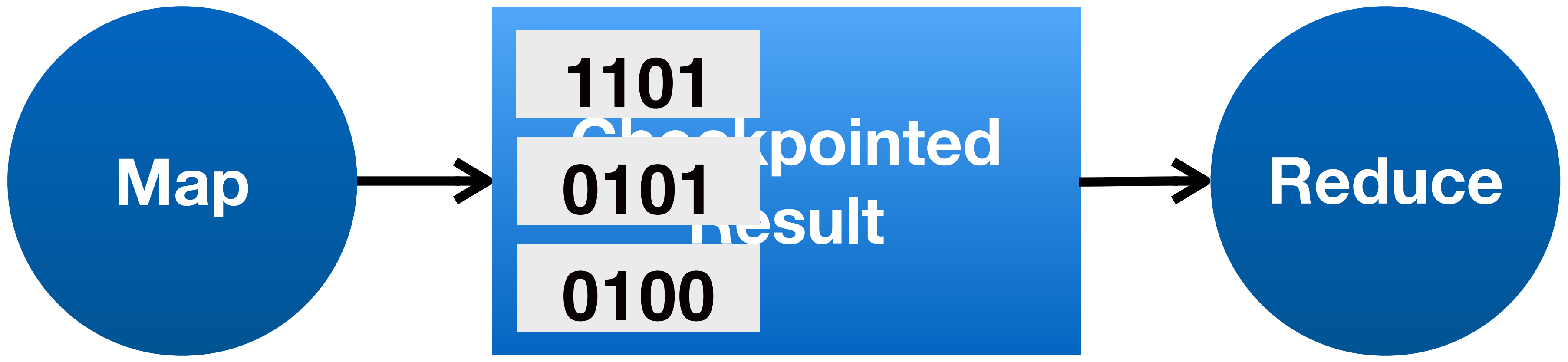
Checkpointed Results



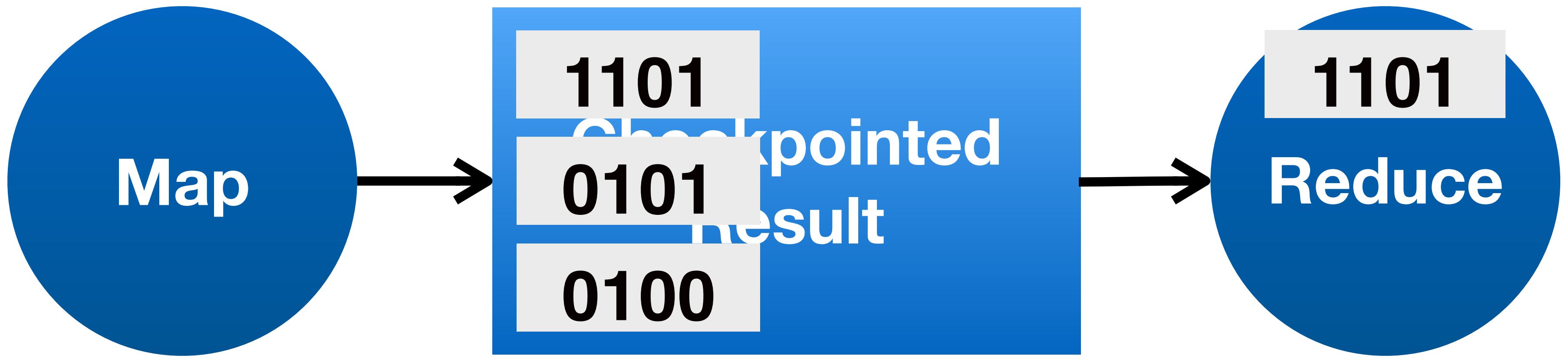
Checkpointed Results



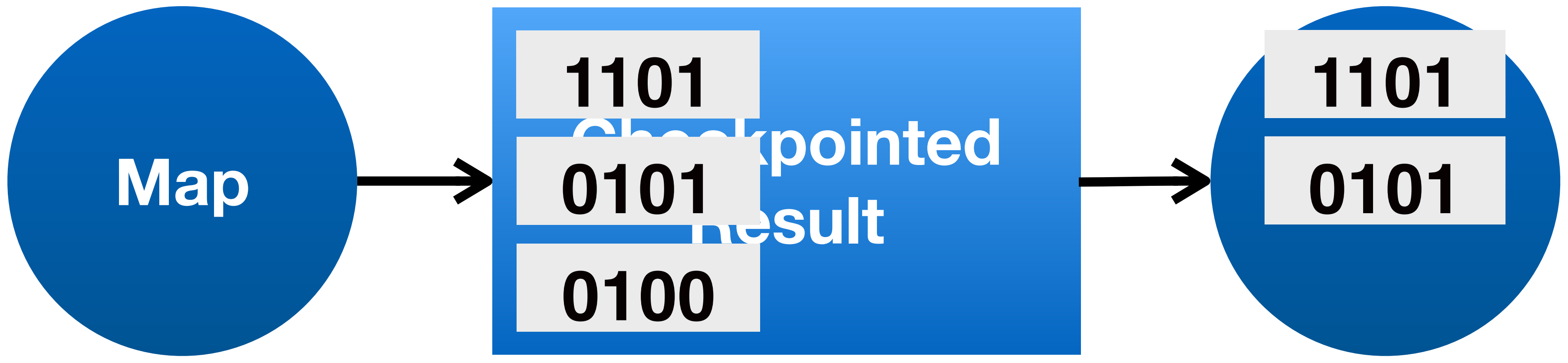
Checkpointed Results



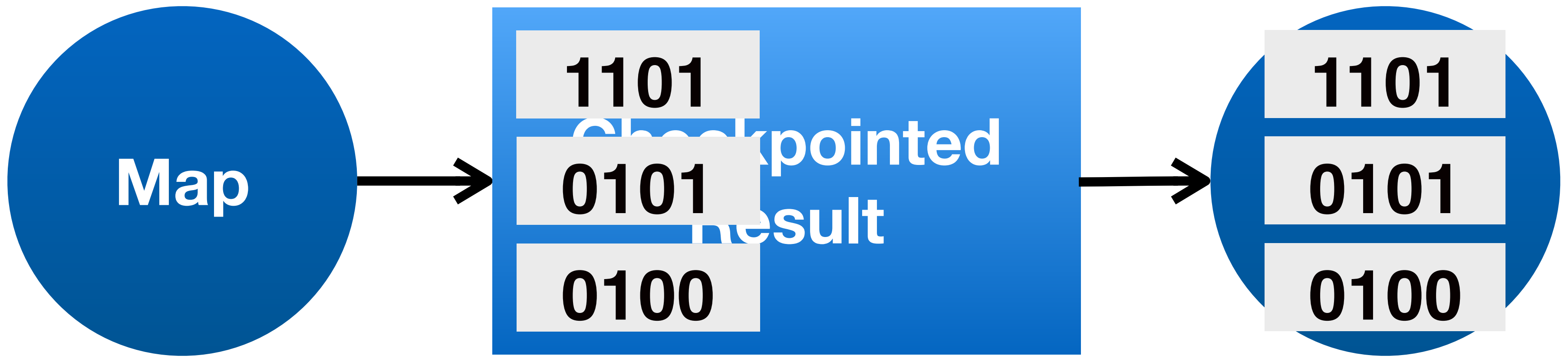
Checkpointed Results



Checkpointed Results



Checkpointed Results



Result Characteristics

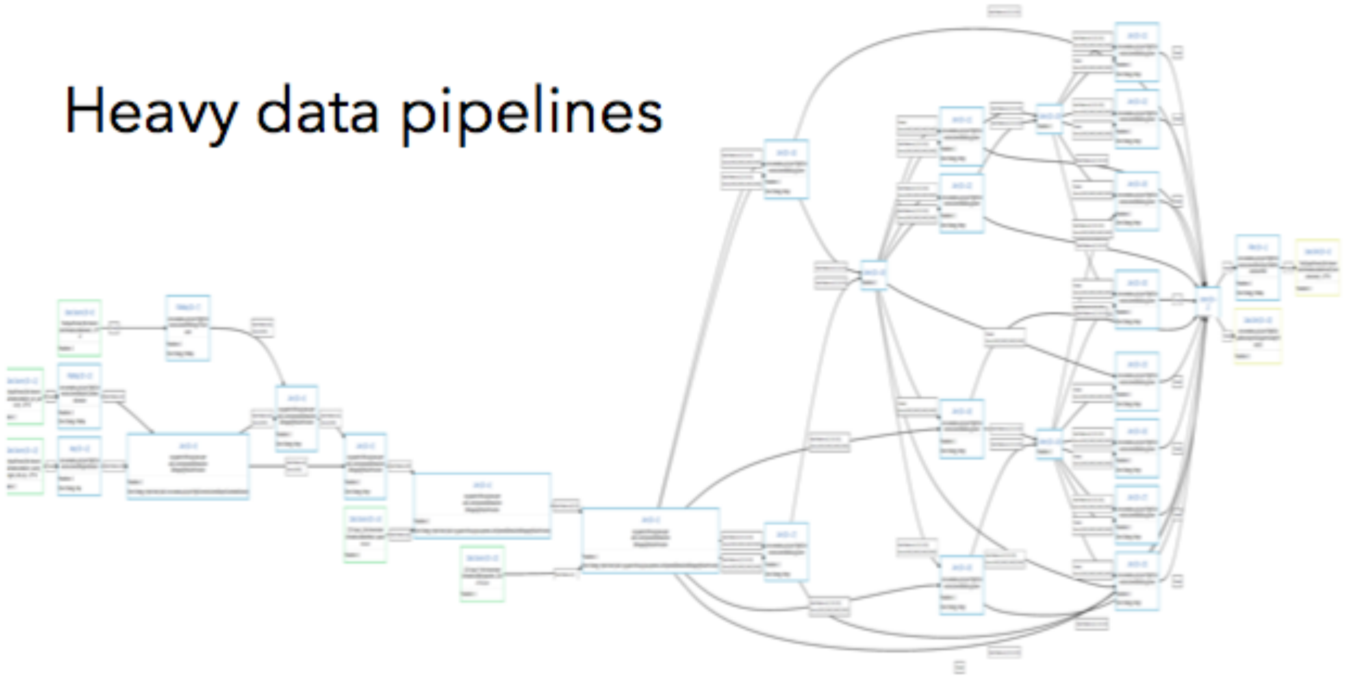
	Ephemeral	Checkpointed
Pipelined	Low-latency	Low-latency **Fine-grained fault tolerance**
Blocking	Easy to reason about resource consumption	**Fine-grained fault tolerance** Easy to reason about resource consumption

Benefits

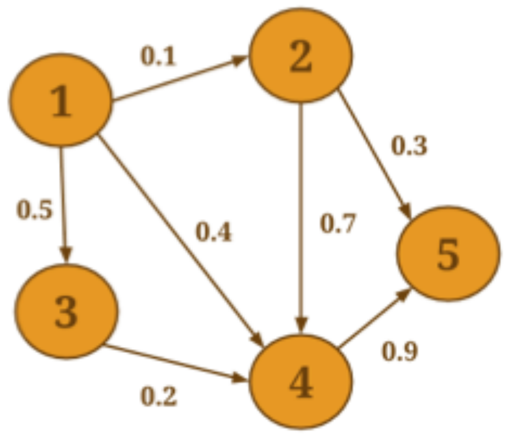
- Very flexible design
- Decouples **high-level requirements** from runtime
  - Fault tolerance for batch vs. streaming
  - Different program optimization paths
  - Iterative programs
  - Interactive queries

Use cases

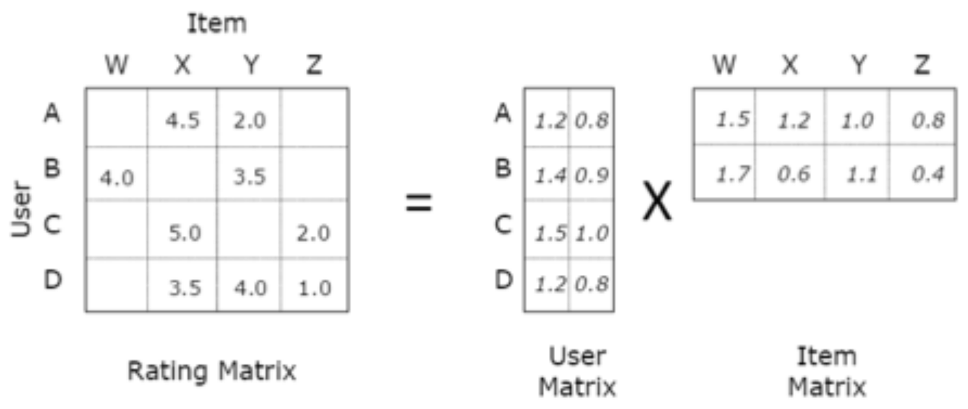
Heavy data pipelines



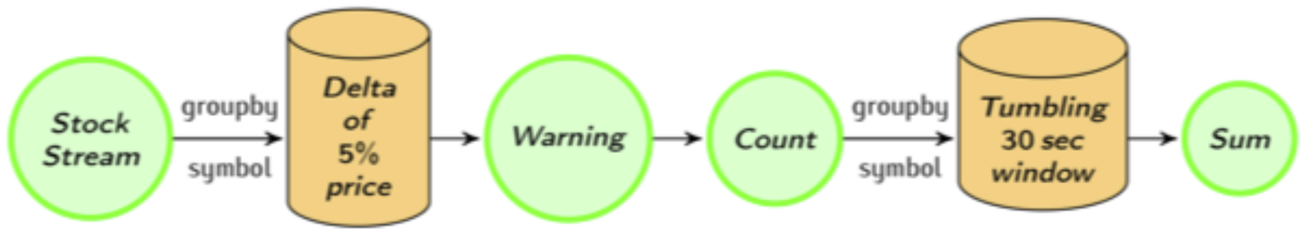
Graph analytics



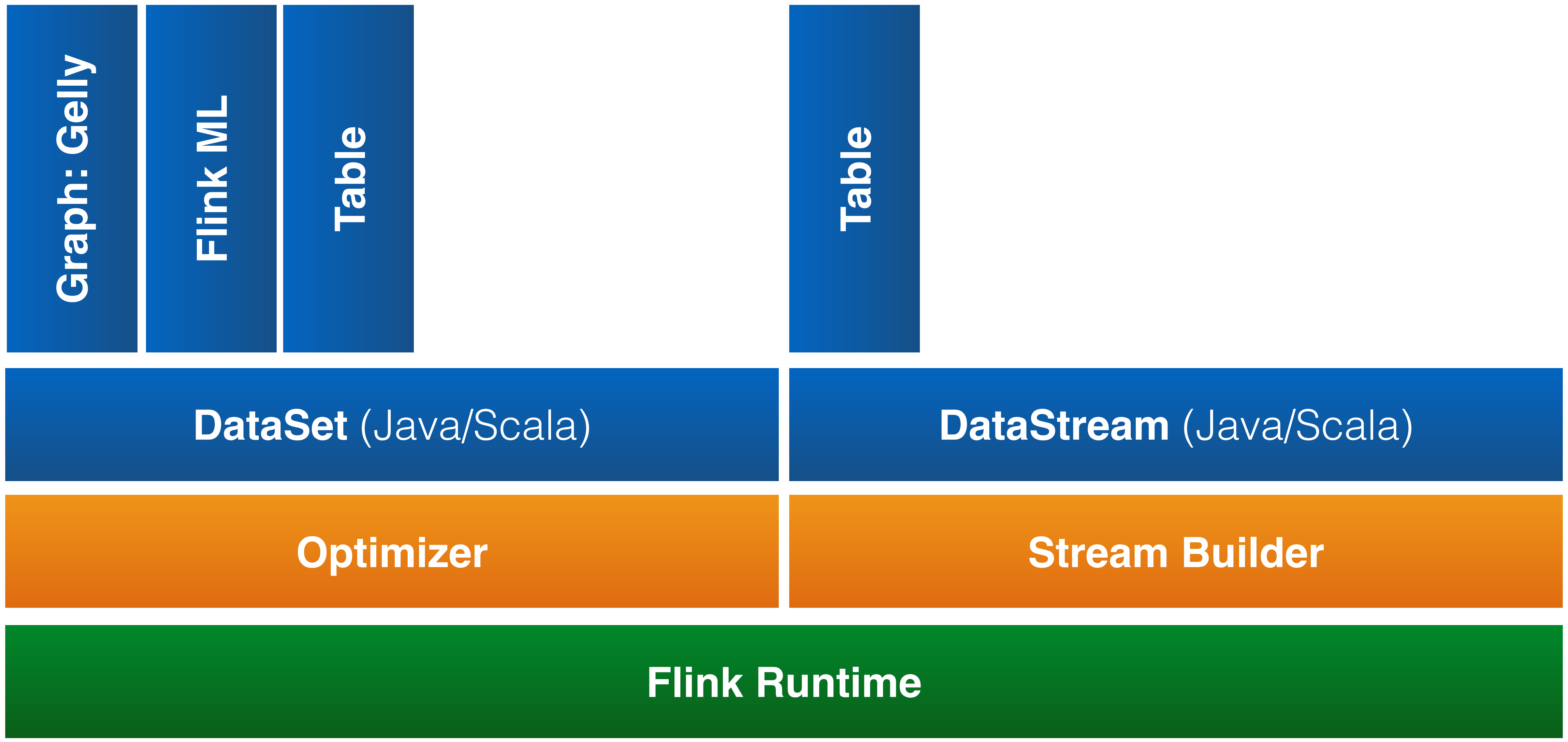
Large-scale Machine Learning



Real-time stream processing



Flink Stack


Graph: Gelly
Flink ML
Table
Table
DataSet (Java/Scala)
DataStream (Java/Scala)
Optimizer
Stream Builder
Flink Runtime

Current Implementations

Pipelined vs. Blocking

Ephemeral vs. Checkpointed

Backpressure vs. No Backpressure

Current Implementations

Pipelined vs. **Blocking**

Ephemeral vs. **Checkpointed**

Backpressure vs. **No Backpressure**

Pipelined vs. Blocking in Flink

- Default type for both batch and streaming programs: **Pipelined**
- In batch mode only: use blocking exchange if necessary (e.g. to avoid deadlocks or break up long pipelines)
- More details: https://cwiki.apache.org/confluence/display/FLINK/Data+exchange+between+tasks

ExecutionConfig

```
// Set up the execution environment
ExecutionEnvironment env = ExecutionEnvironment
  .getExecutionEnvironment();

ExecutionConfig conf = env.getConfig();

// Remote data exchange is blocking (local is pipelined)
conf.setExecutionMode(ExecutionMode.BATCH);

// Remote and local data exchange is blocking
conf.setExecutionMode(ExecutionMode.BATCH_FORCED);

// Remote and local data exchange is pipelined except
// when necessary to avoid deadlocks etc. [DEFAULT]
conf.setExecutionMode(ExecutionMode.PIPELINED);

// Remote and local data exchange is always pipelined
conf.setExecutionMode(ExecutionMode.PIPELINED_FORCED);
```

flink.apache.org

@ApacheFlink