CONFLUENT

The Nuts and Bolts of Kafka Streams: An Architectural Deep Dive (’24 update)

Matthias J. Sax
Software Engineer | Apache Kafka Committer and PMC member

Twitter/𝕏 @MatthiasJSax

The DSL Program

(let’s go down the rabbit hole)

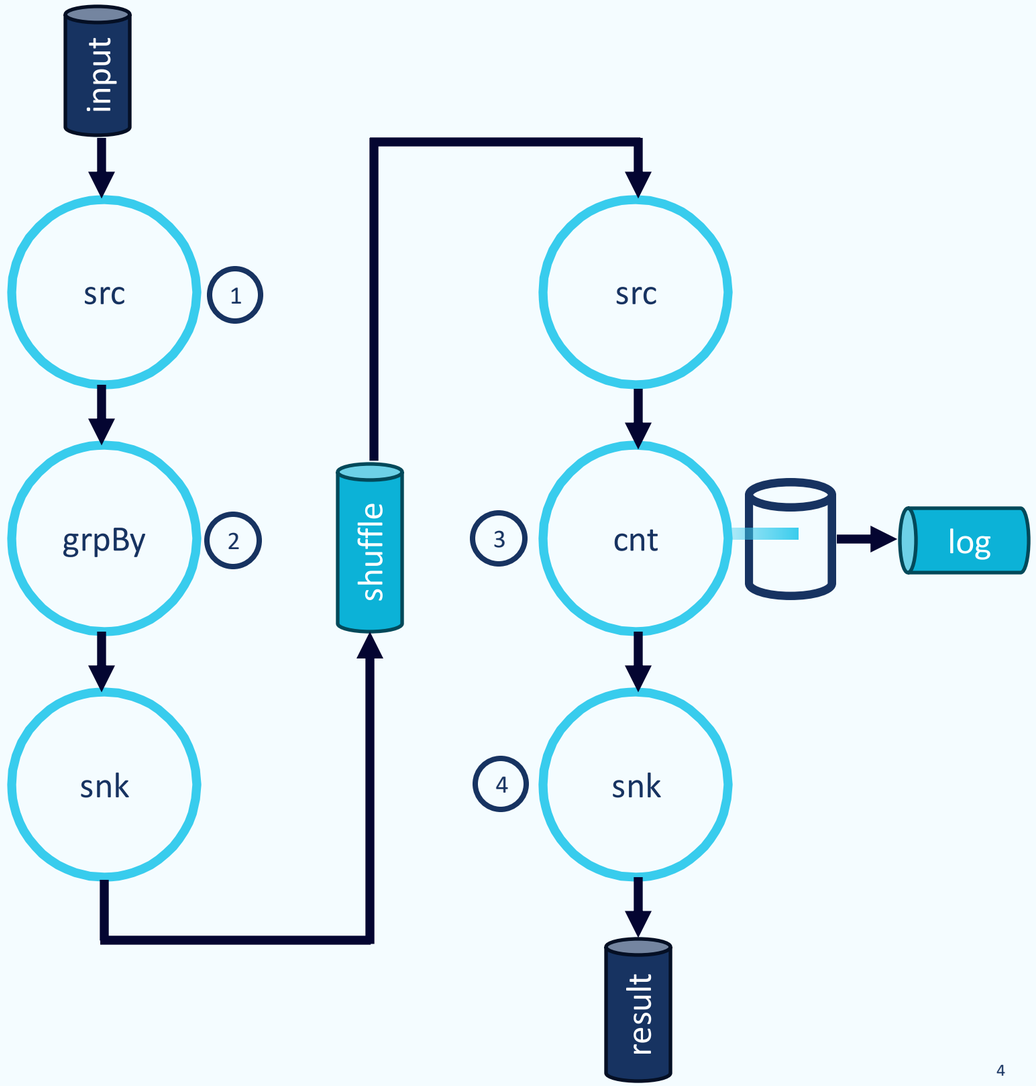
Program

```
StreamsBuilder builder = …

KStream words = builder.stream(…);
KTable result = words.groupBy(…)
                     .count();
result.toStream().to(…);


Topology t = builder.build();
```

Program

```
StreamsBuilder builder = …

KStream words = builder.stream(…);          1
KTable result = words.groupBy(…)            2
                     .count();              3
result.toStream().to(…);                    4


Topology t = builder.build();
```

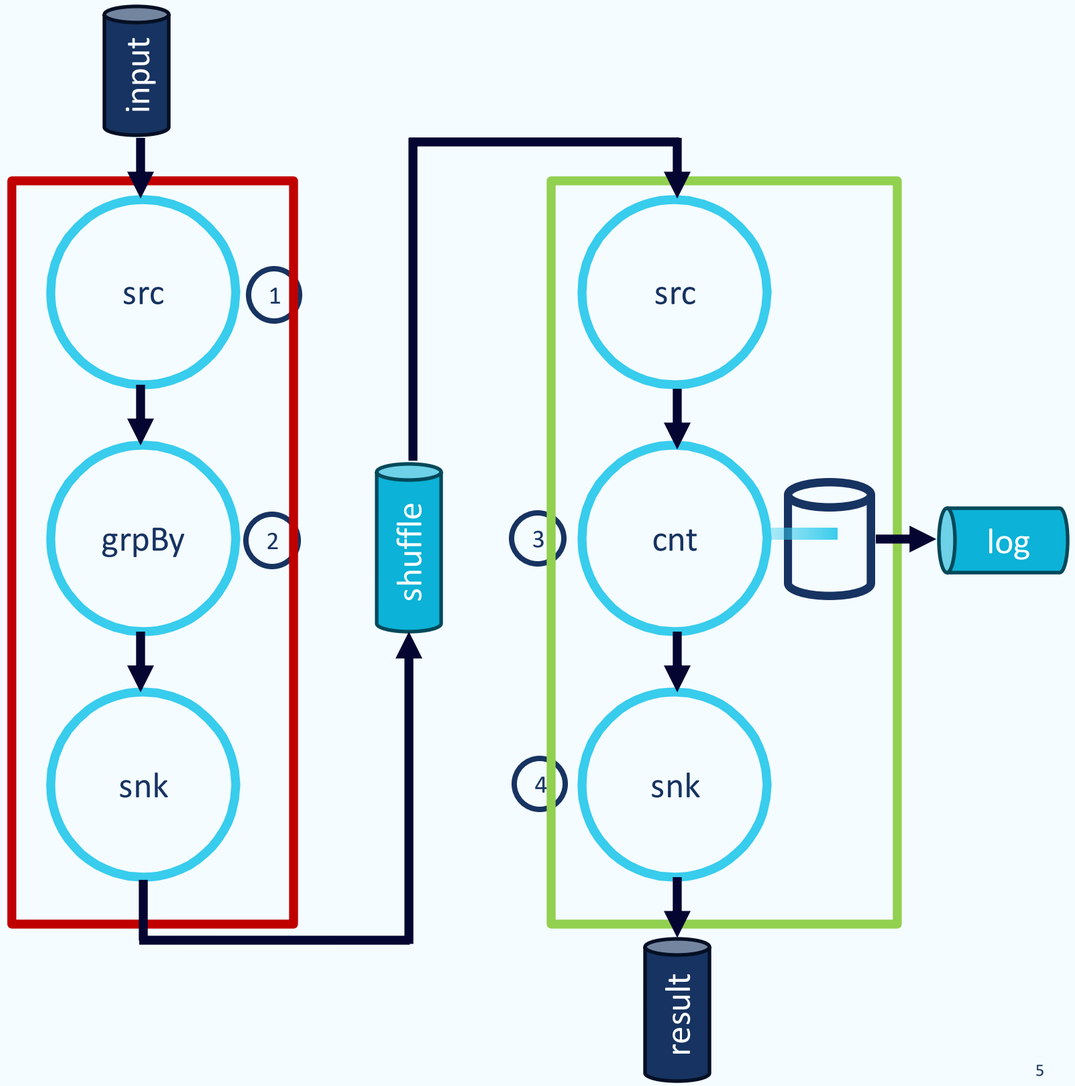
(Sub-)Topology



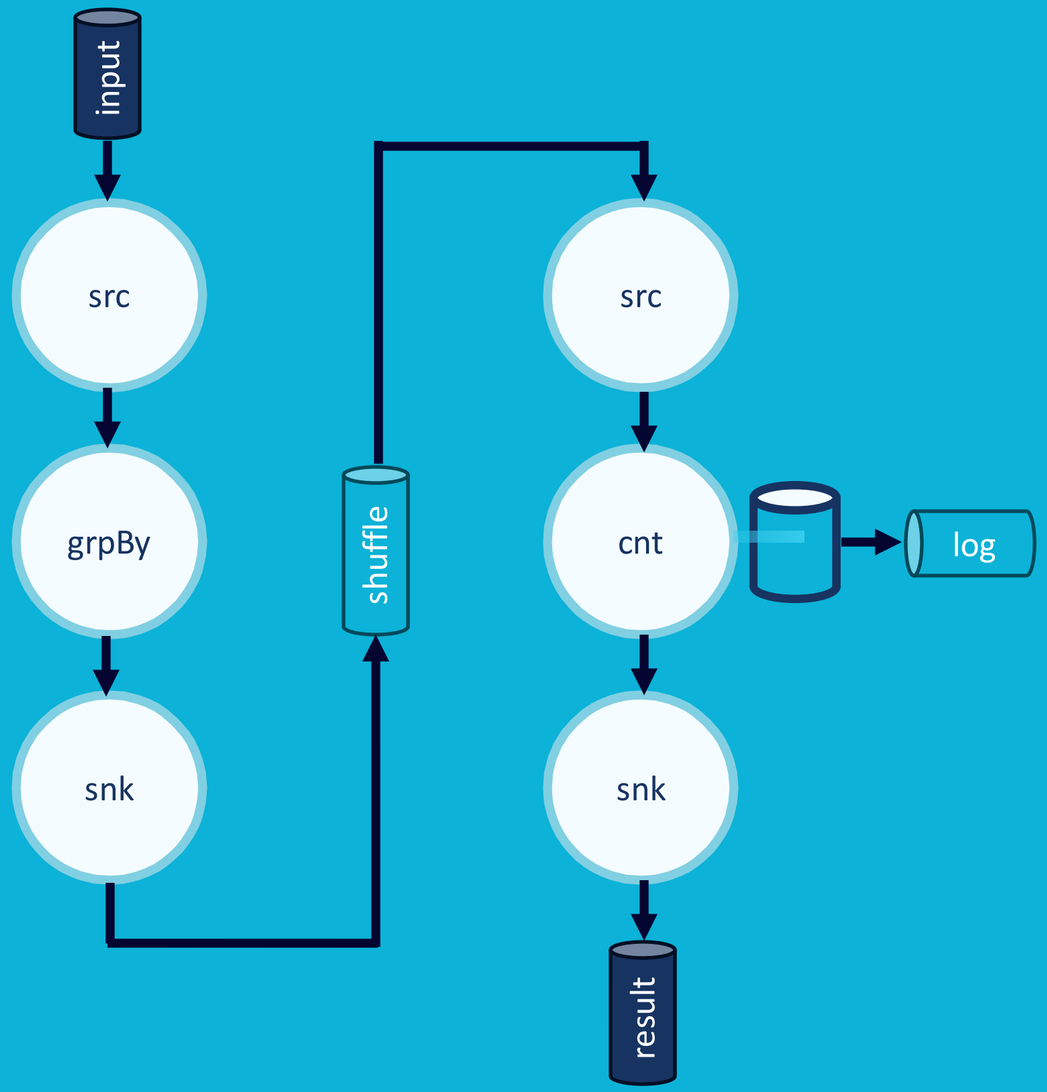
Program

```
StreamsBuilder builder = …

KStream words = builder.stream(…);          1
KTable result = words.groupBy(…)            2
                     .count();              3
result.toStream().to(…);                    4


Topology t = builder.build();
```

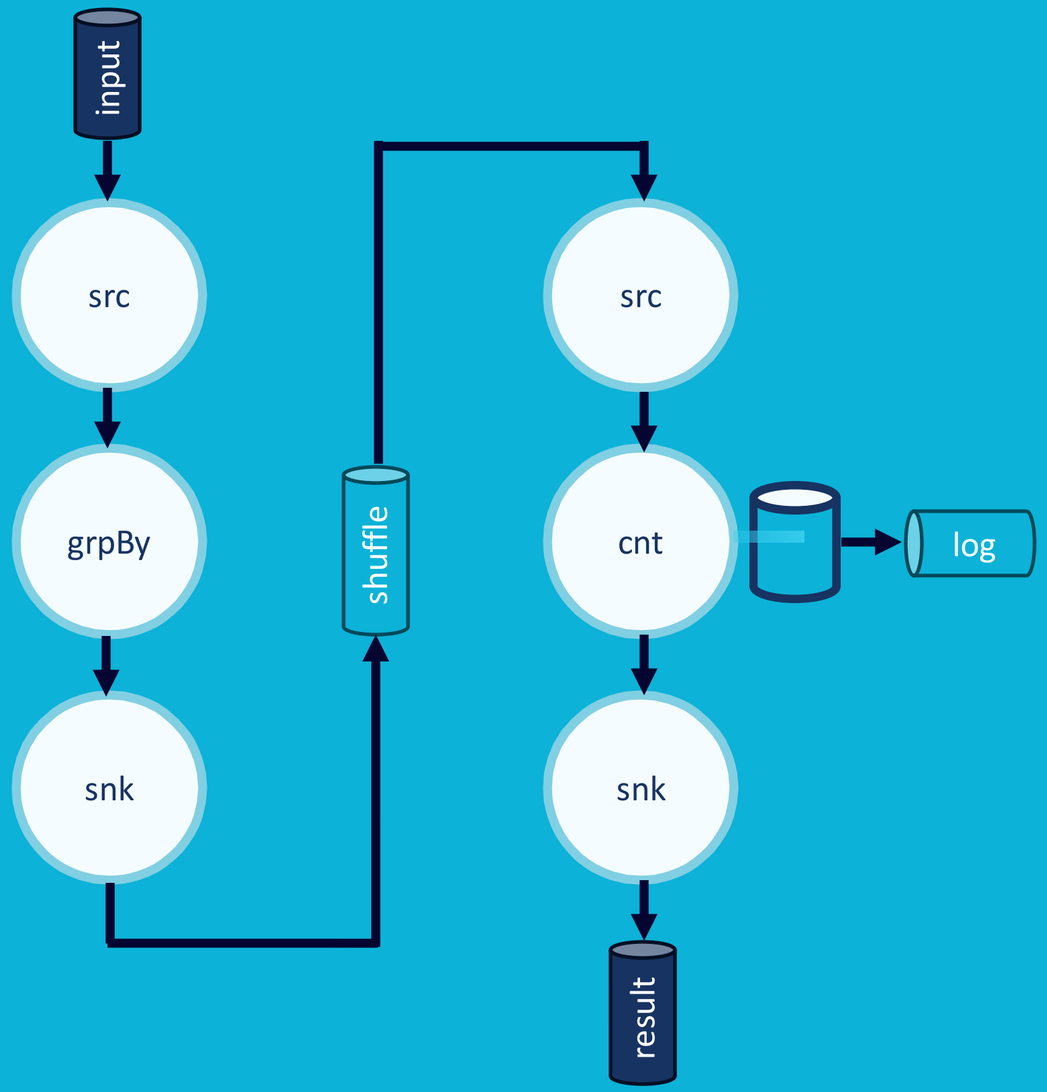
(Sub-)Topology



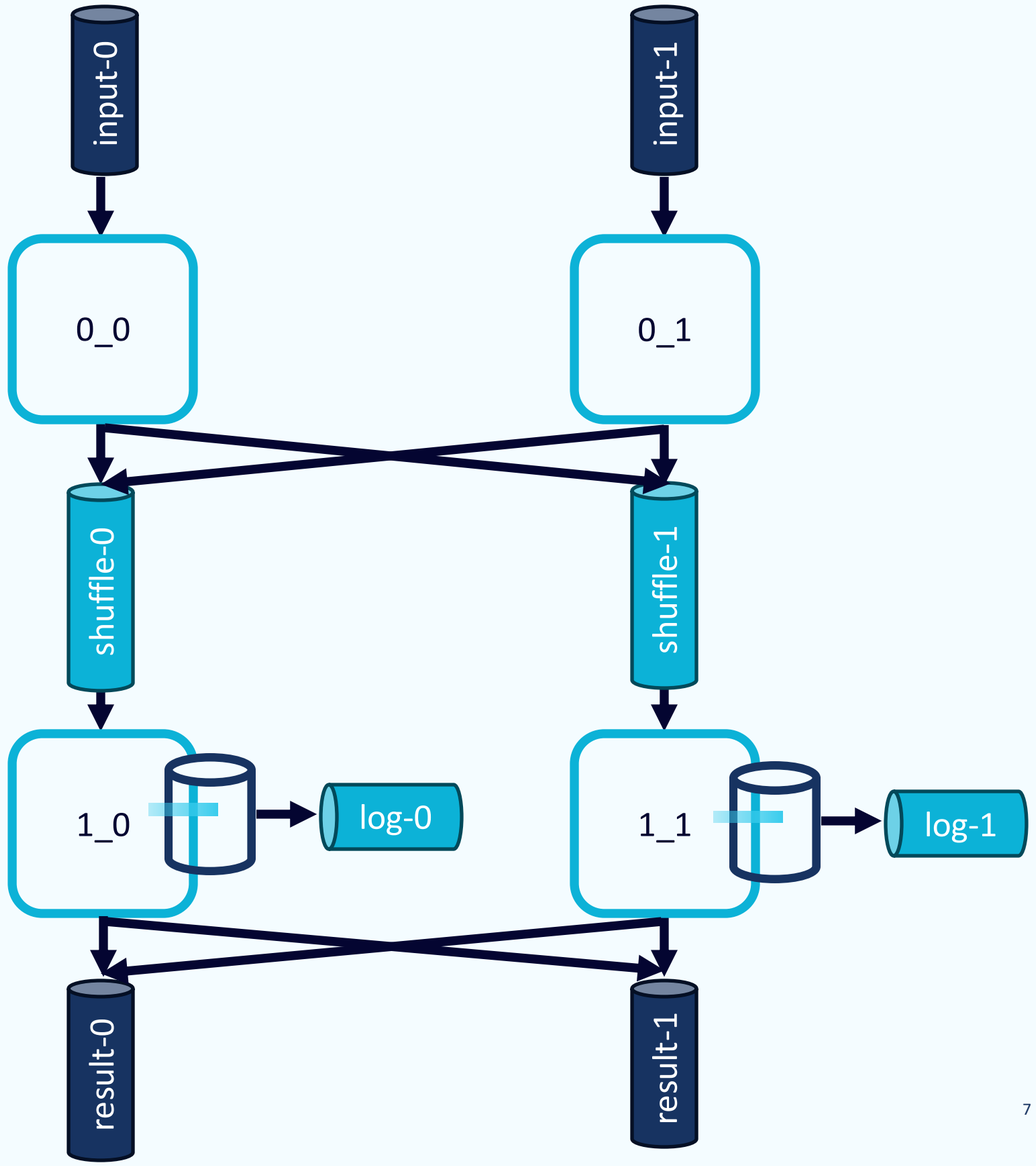
(Sub-)Topology



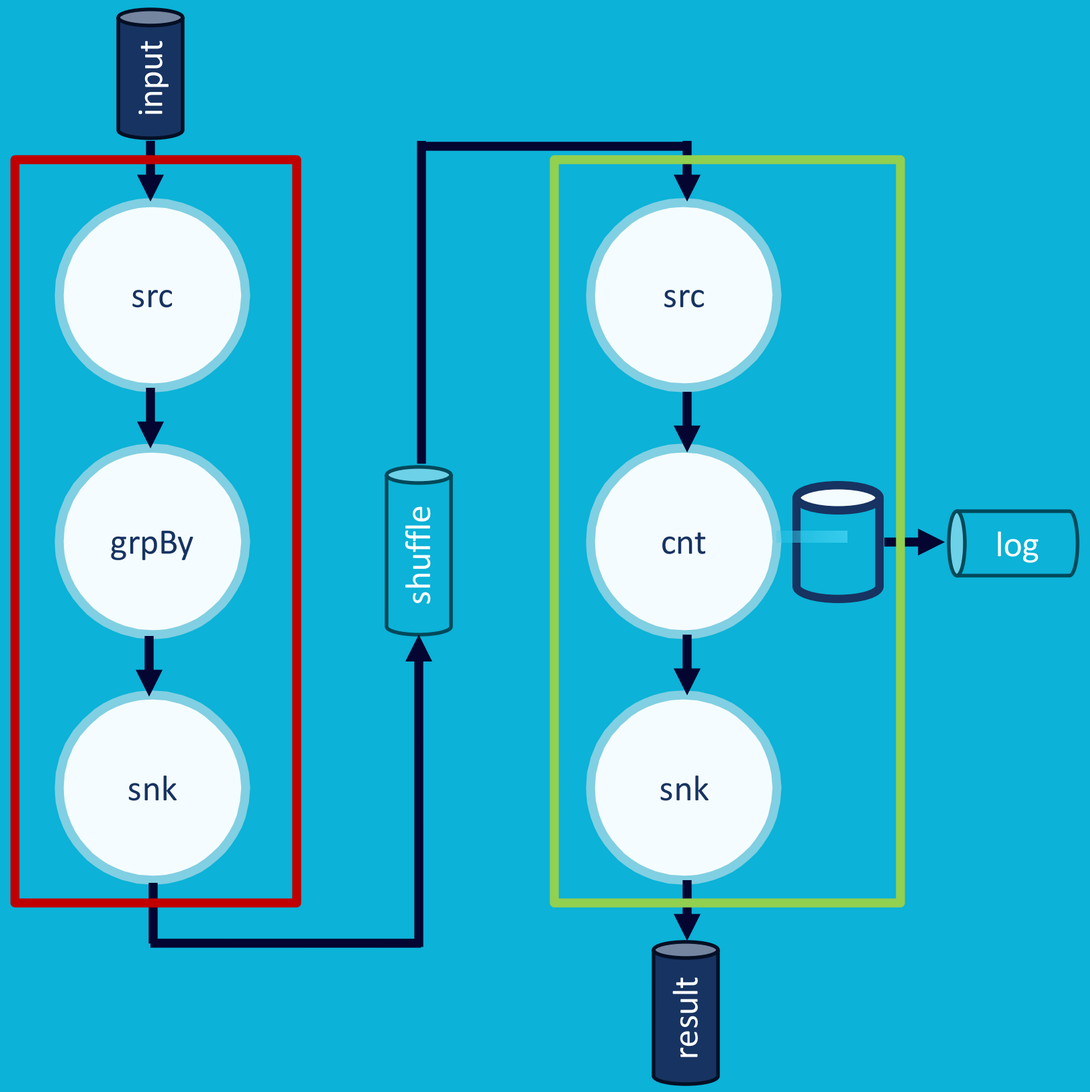
(Sub-)Topology



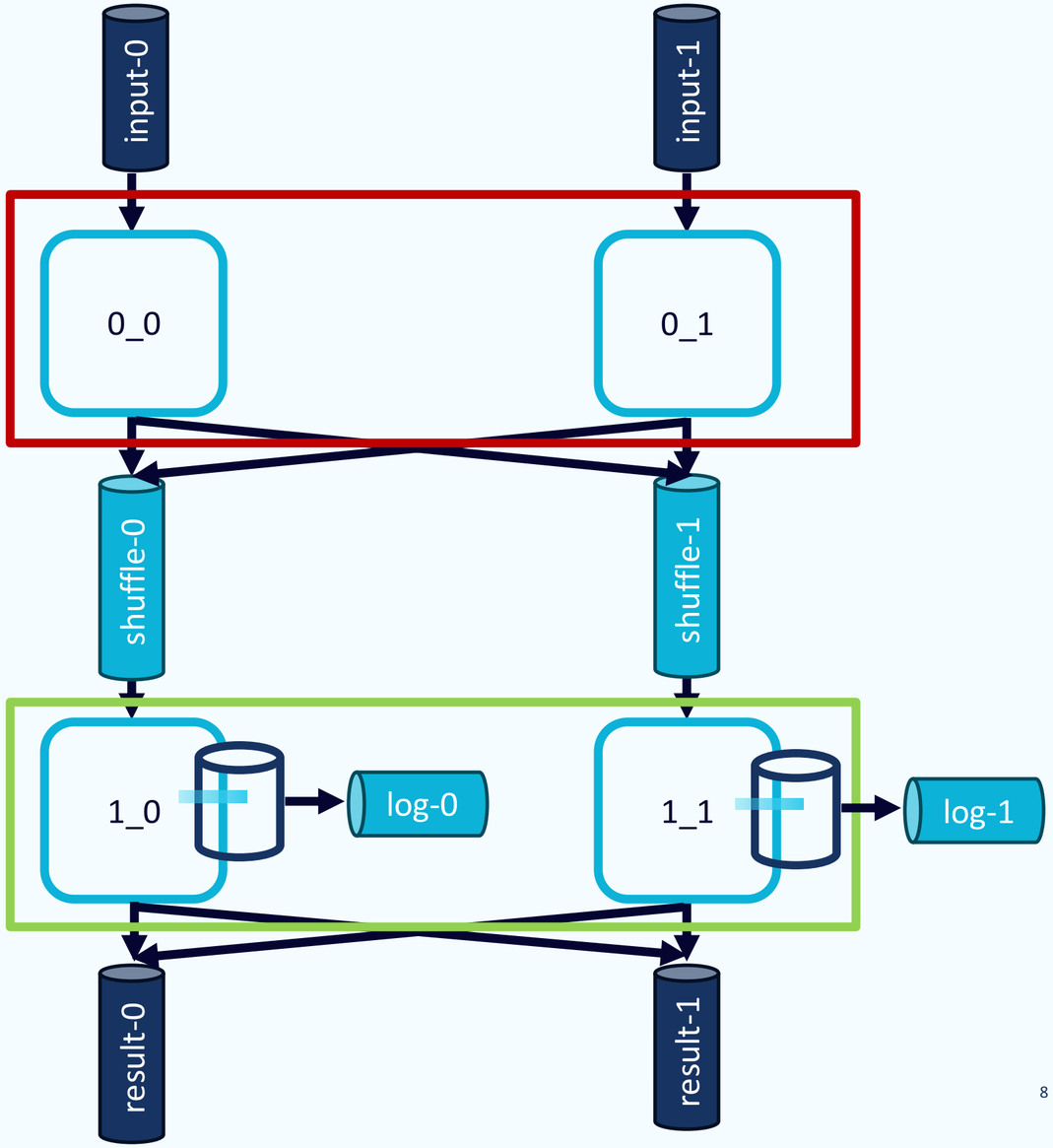
Tasks



(Sub-)Topology

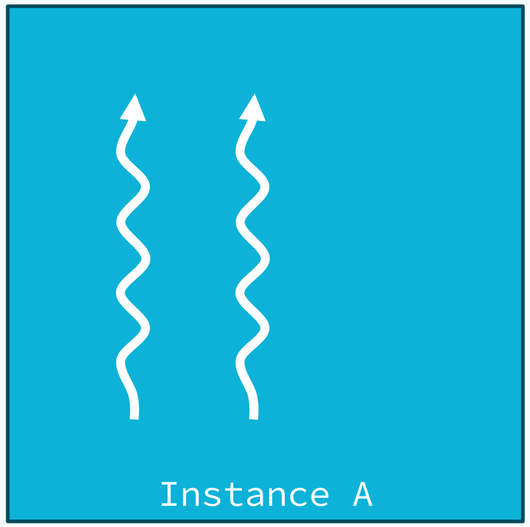


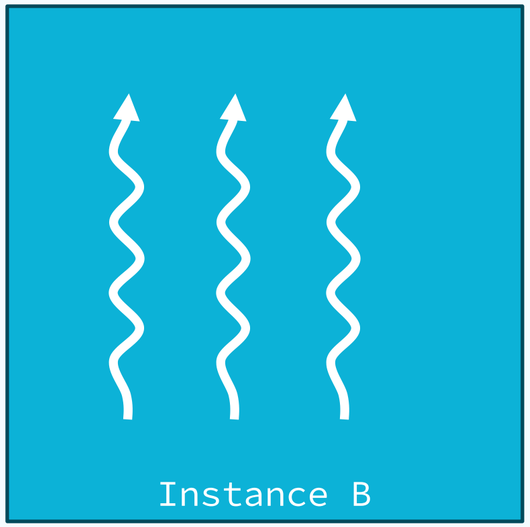
Tasks



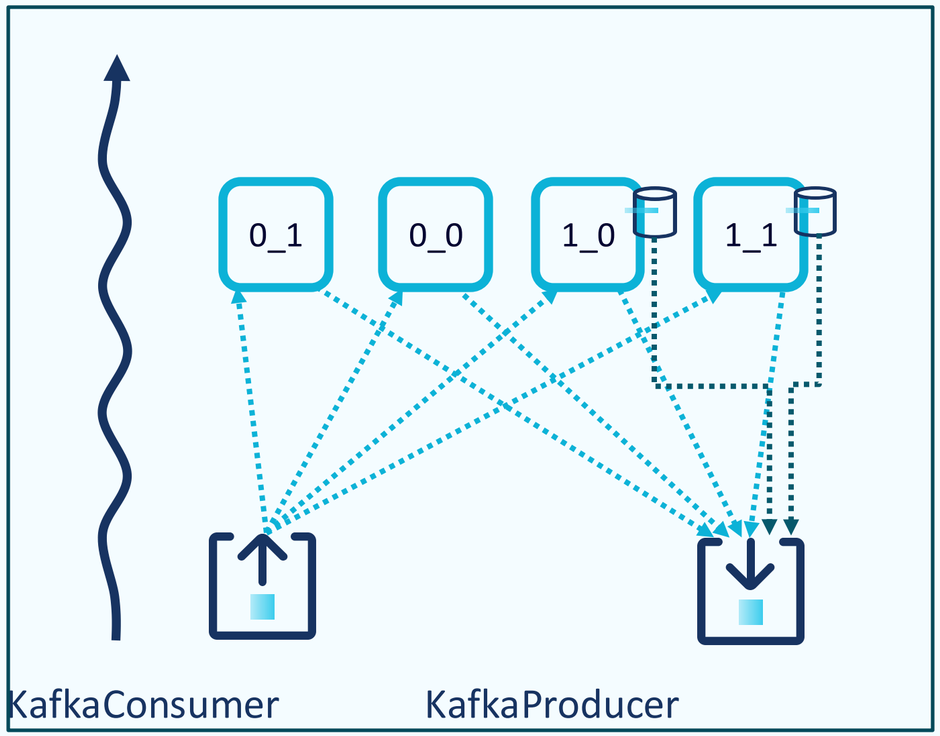
Kafka Streams Instances and StreamThreads

```
KafkaStreams streamsClient = new KafkaStreams(…); // num.stream.threads
streamsClient.start();

streamsClient.addStreamThread();
streamsClient.removeStreamThread();

// error handling: uncaught exception handler
```





StreamThreads and Tasks



Kafka Cluster

Main Processing Loop

```
while (running)
  records = mainConsumer.poll()      // config: poll.ms
                                     //         max.poll.records
                                     //         fetch configs

  addRecordsToTask(records);         // no deserialization
                                     // config: buffered.records.per.partition

  while (keepProcessing)
    for each task:
      if (task.isReady)              // data available for all inputs?
        task.process(nRecords)       // -> ts-extraction
                                     // -> (de)serialization + exception-handling

    maybePunctuate()
    maybeCommit()                    // config: commit.interval.ms [ctx.commit()]

    adjust nRecords
```

Main Processing Loop

```
while (running)
  records = mainConsumer.poll()

  addRecordsToTask(records);

  while (keepProcessing)
    for each task:
      if (task.isReady)
        task.process(nRecords)

    maybePunctuate()
    maybeCommit()

    adjust nRecords
```

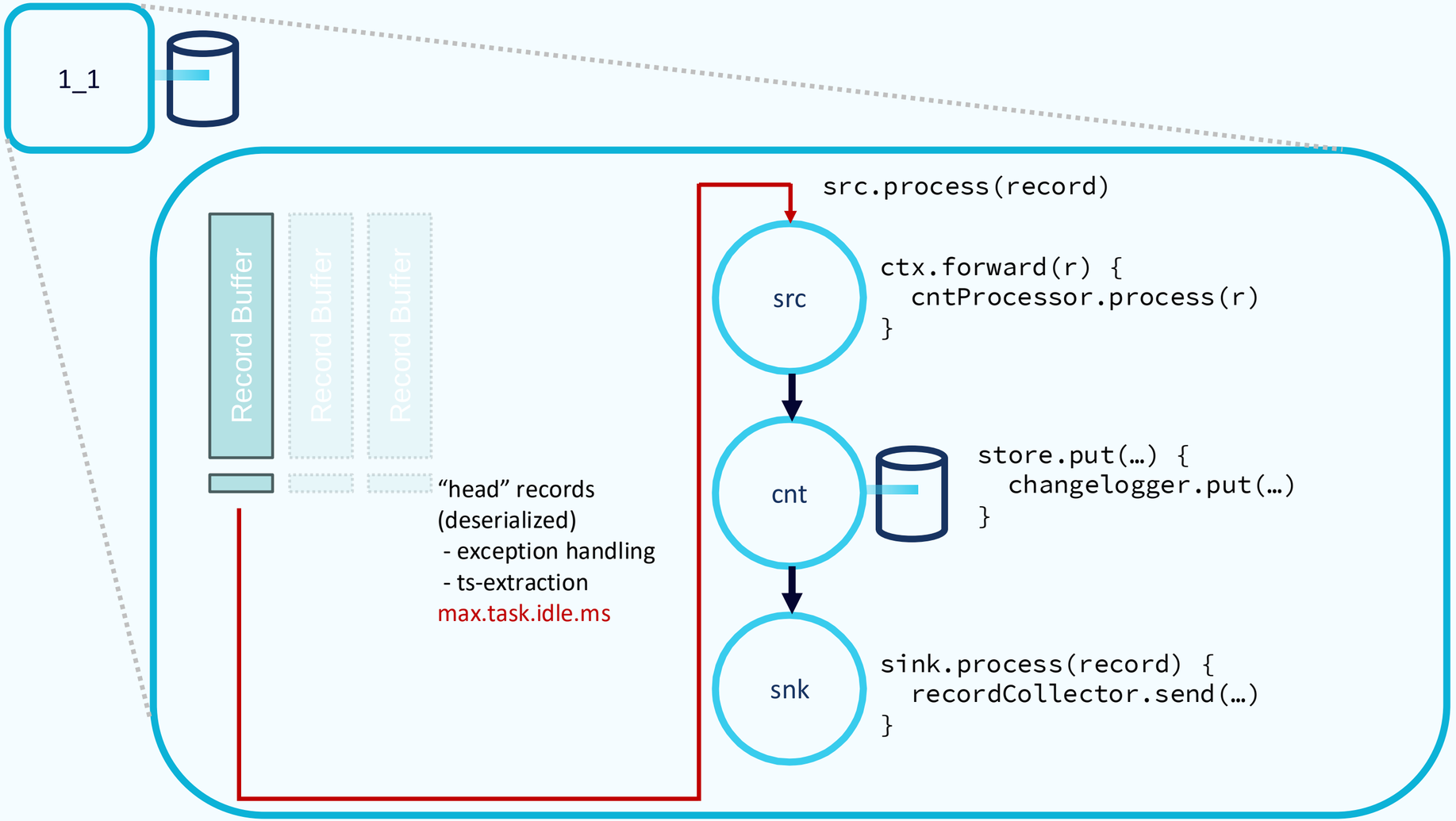
```
// config: poll.ms
//         max.poll.records
//         fetch configs

// no deserialization
// config: buffered.records.per.partition

// data available for all inputs?
// -> ts-extraction
// -> (de)serialization + exception-handling

// config: commit.interval.ms [ctx.commit()]

max.poll.interval.ms
```

Main Processing Loop

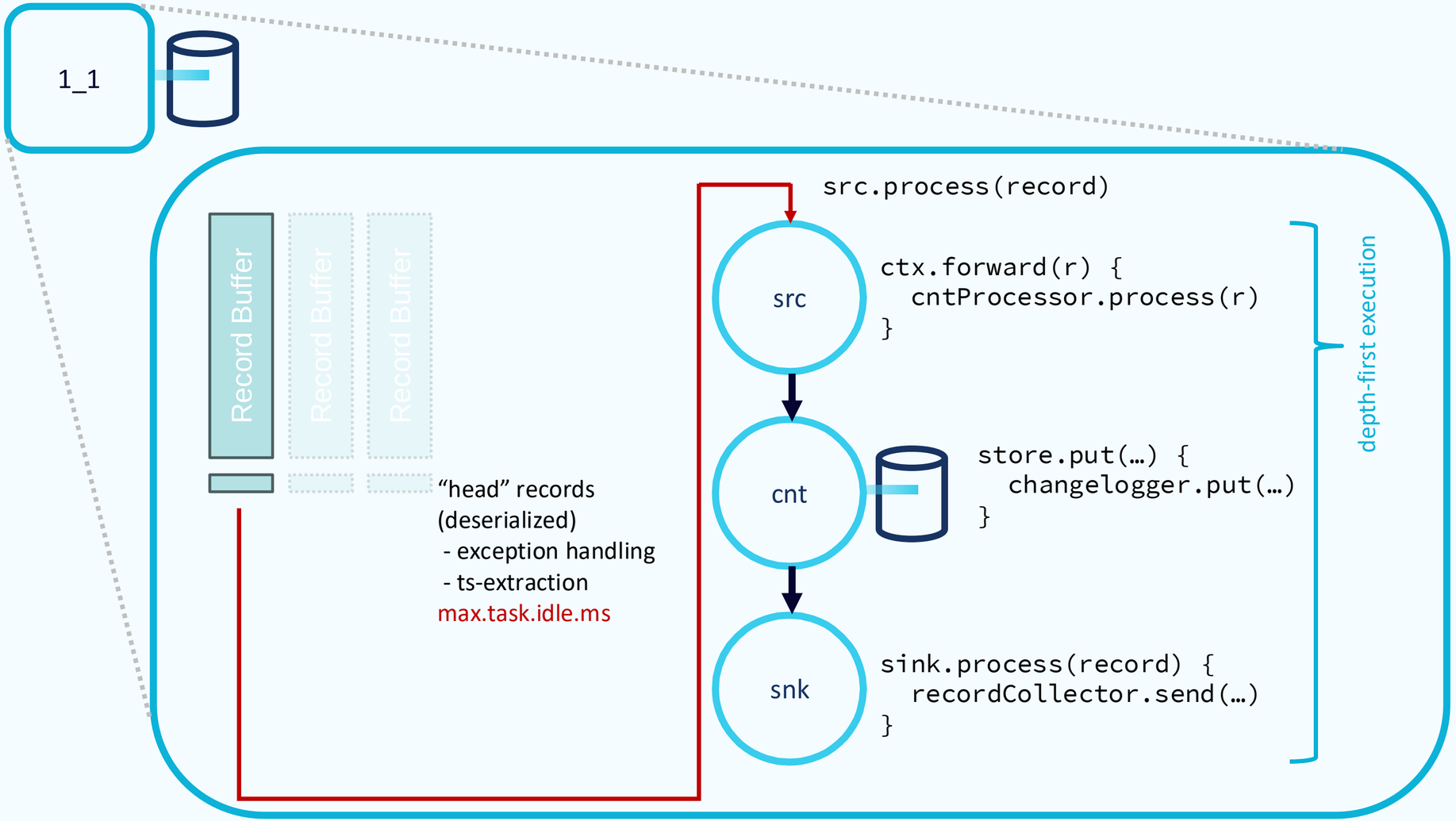
```
while (running)
  records = mainConsumer.poll()

  addRecordsToTask(records);

  while (keepProcessing)
    for each task:
      if (task.isReady)
        task.process(nRecords)

    maybePunctuate()
    maybeCommit()

    adjust nRecords
```

```
// config: poll.ms
//         max.poll.records
//         fetch configs

// no deserialization
// config: buffered.records.per.partition

// data available for all inputs?
// -> ts-extraction
// -> (de)serialization + exception-handling

// config: commit.interval.ms [ctx.commit()]

max.poll.interval.ms
```

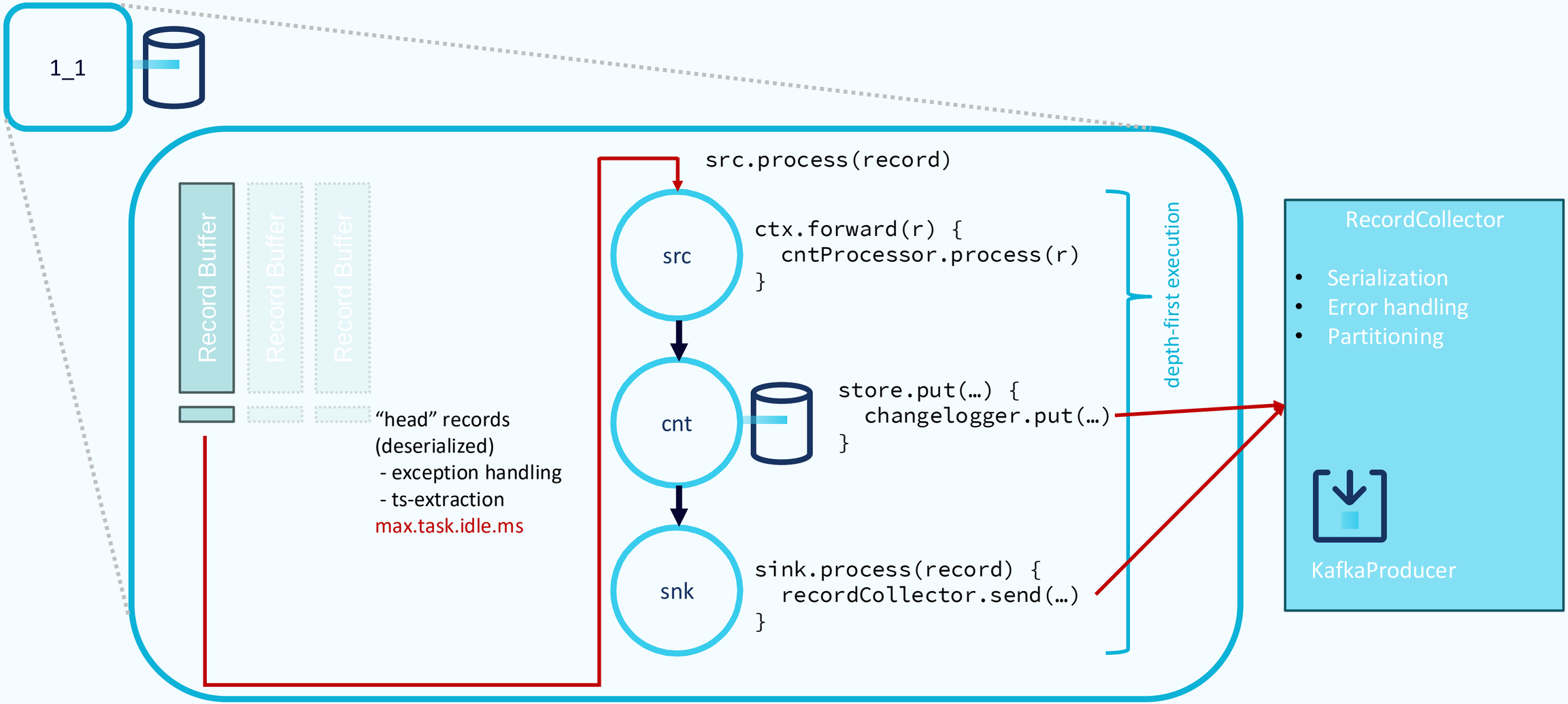
Task Execution



Task Execution



Task Execution



Main Processing Loop

```
while (running)
  records = mainConsumer.poll()

  addRecordsToTask(records);

  while (keepProcessing)
    for each task:
      if (task.isReady)
        task.process(nRecords)

    maybePunctuate()
    maybeCommit()

    adjust nRecords
```

```
// config: poll.ms
//         max.poll.records
//         fetch configs

// no deserialization
// config: buffered.records.per.partition

// data available for all inputs?
// -> ts-extraction
// -> (de)serialization + exception-handling

// config: commit.interval.ms [ctx.commit()]

max.poll.interval.ms
```

Main Processing Loop

```
while (running)
  records = mainConsumer.poll()

  addRecordsToTask(records);

  while (keepProcessing)
    for each task:
      if (task.isReady)
        task.process(nRecords)

    maybePunctuate()
    maybeCommit()

    adjust nRecords
```

```
// config: poll.ms
//         max.poll.records
//         fetch configs

// no deserialization
// config: buffered.records.per.partition

// data available for all inputs?
// -> ts-extraction
// -> (de)serialization + exception-handling

// config: commit.interval.ms [ctx.commit()]

max.poll.interval.ms
```

Committing

At-least-once:

- flush state stores
- flush producer
    - avoid data
- get offset / consumer position
    - record buffers

- `commitSync()`

Exactly-once:

Committing

At-least-once:

- flush state stores
- flush producer
    - avoid data
- get offset / consumer position
    - record buffers

- `commitSync()`

Exactly-once:

- flush state stores
- flush producer
    - avoid data
- get offsets / consumer position
    - record buffers

- `addOffsetsToTransaction()`
- `commitTransaction()`

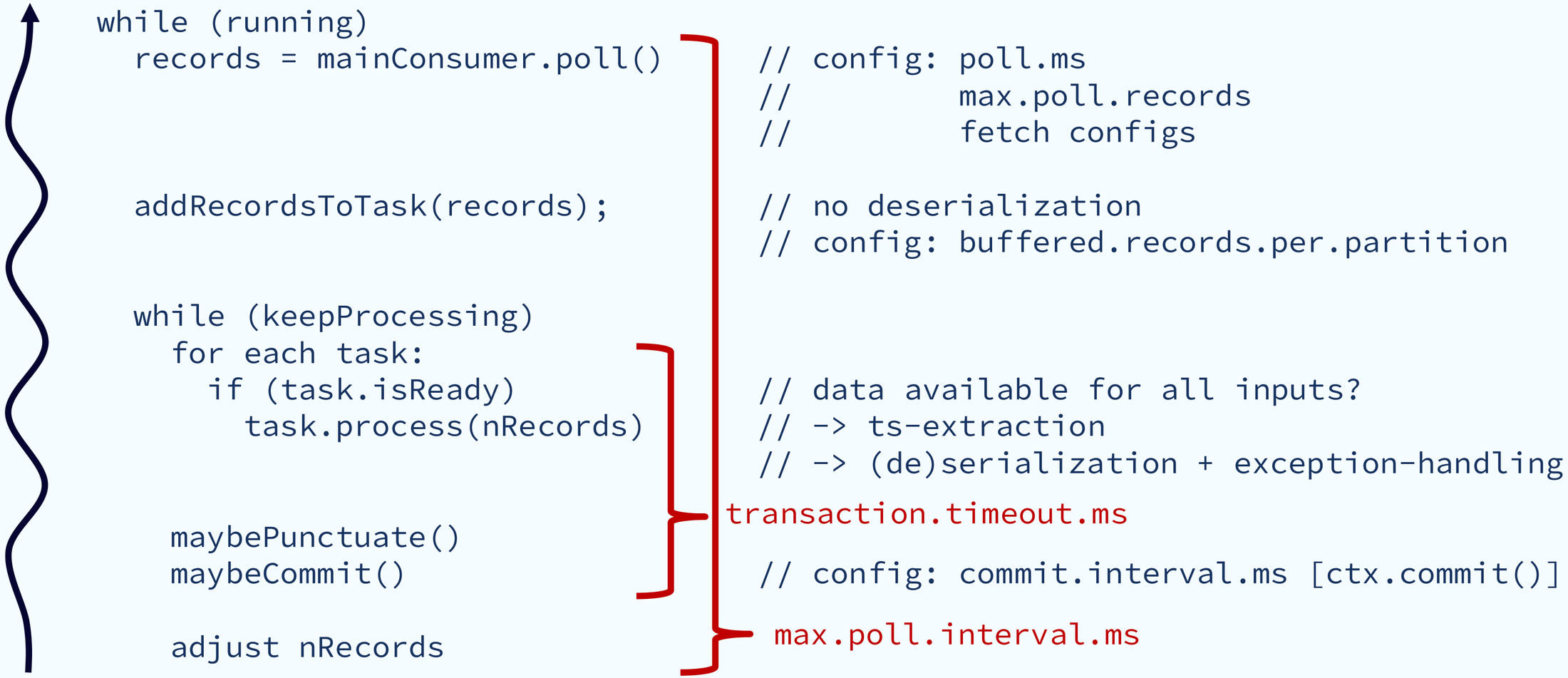
- configs:
    - `transaction.timeout.ms`

Main Processing Loop

```
while (running)
  records = mainConsumer.poll()

  addRecordsToTask(records);

  while (keepProcessing)
    for each task:
      if (task.isReady)
        task.process(nRecords)

    maybePunctuate()
    maybeCommit()

    adjust nRecords
```

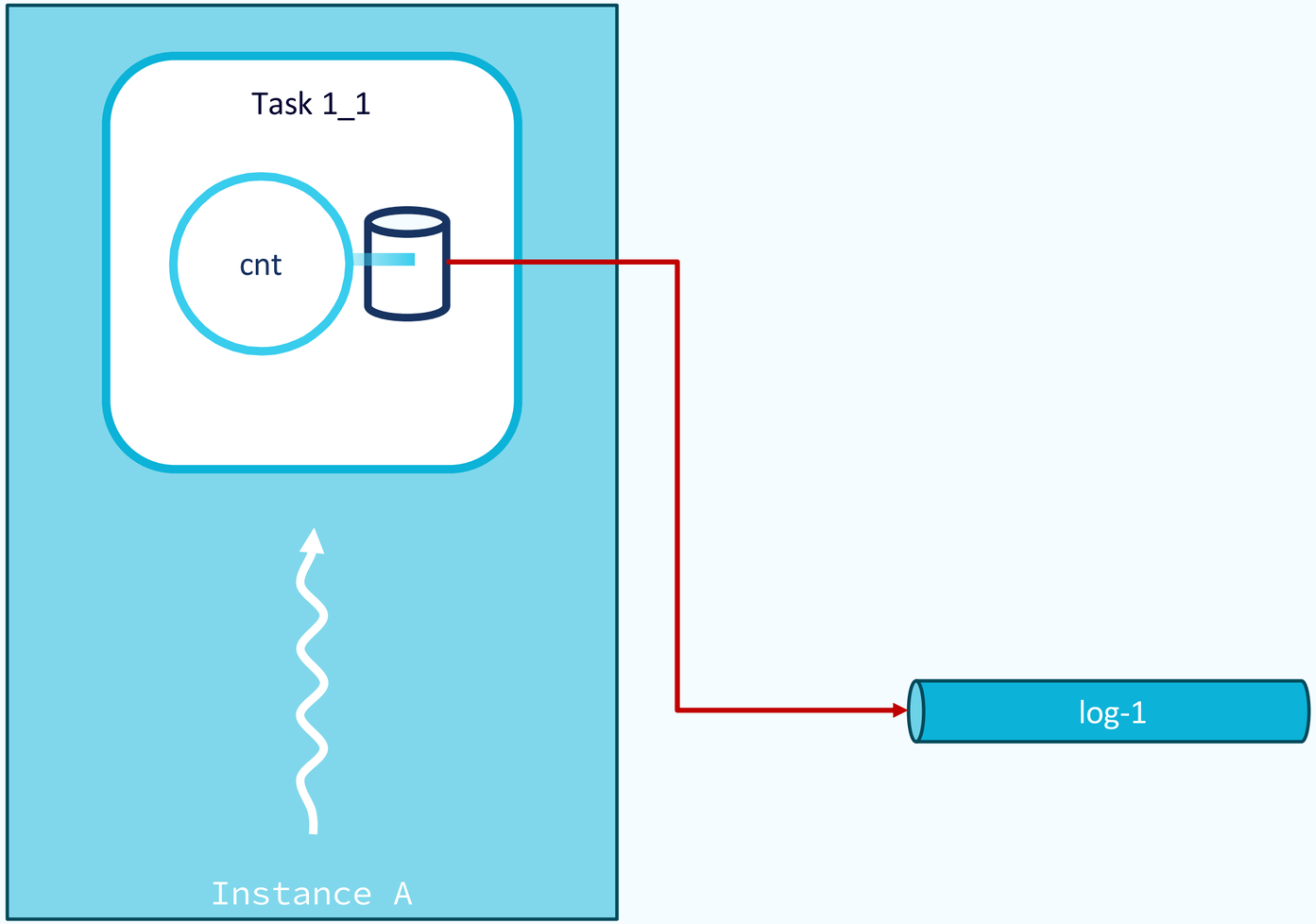
```
// config: poll.ms
//         max.poll.records
//         fetch configs

// no deserialization
// config: buffered.records.per.partition

// data available for all inputs?
// -> ts-extraction
// -> (de)serialization + exception-handling

// config: commit.interval.ms [ctx.commit()]

max.poll.interval.ms
```

Main Processing Loop

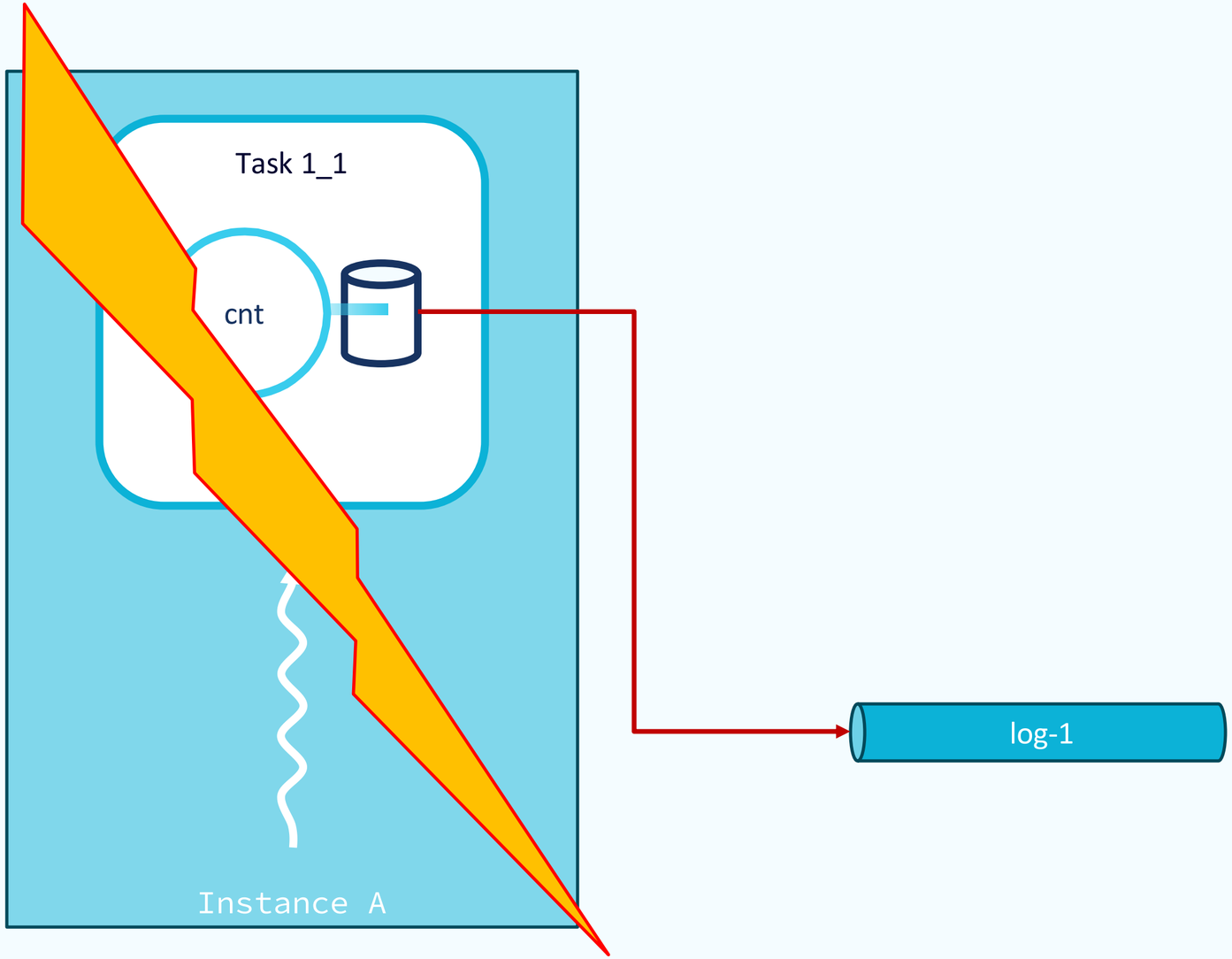


Fault-Tolerance, High-Availability, and Scaling

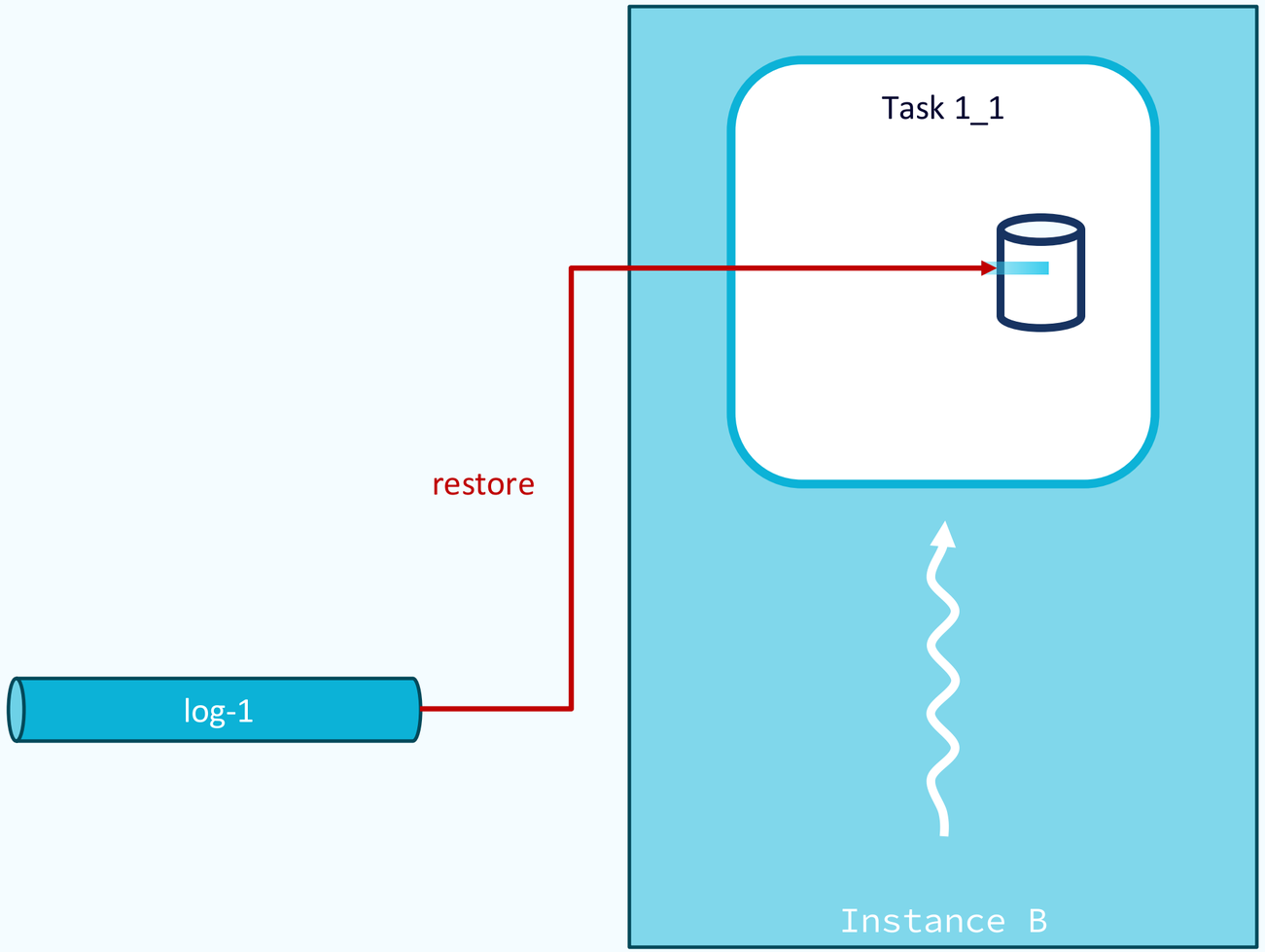
State Stores and Changelog Topics



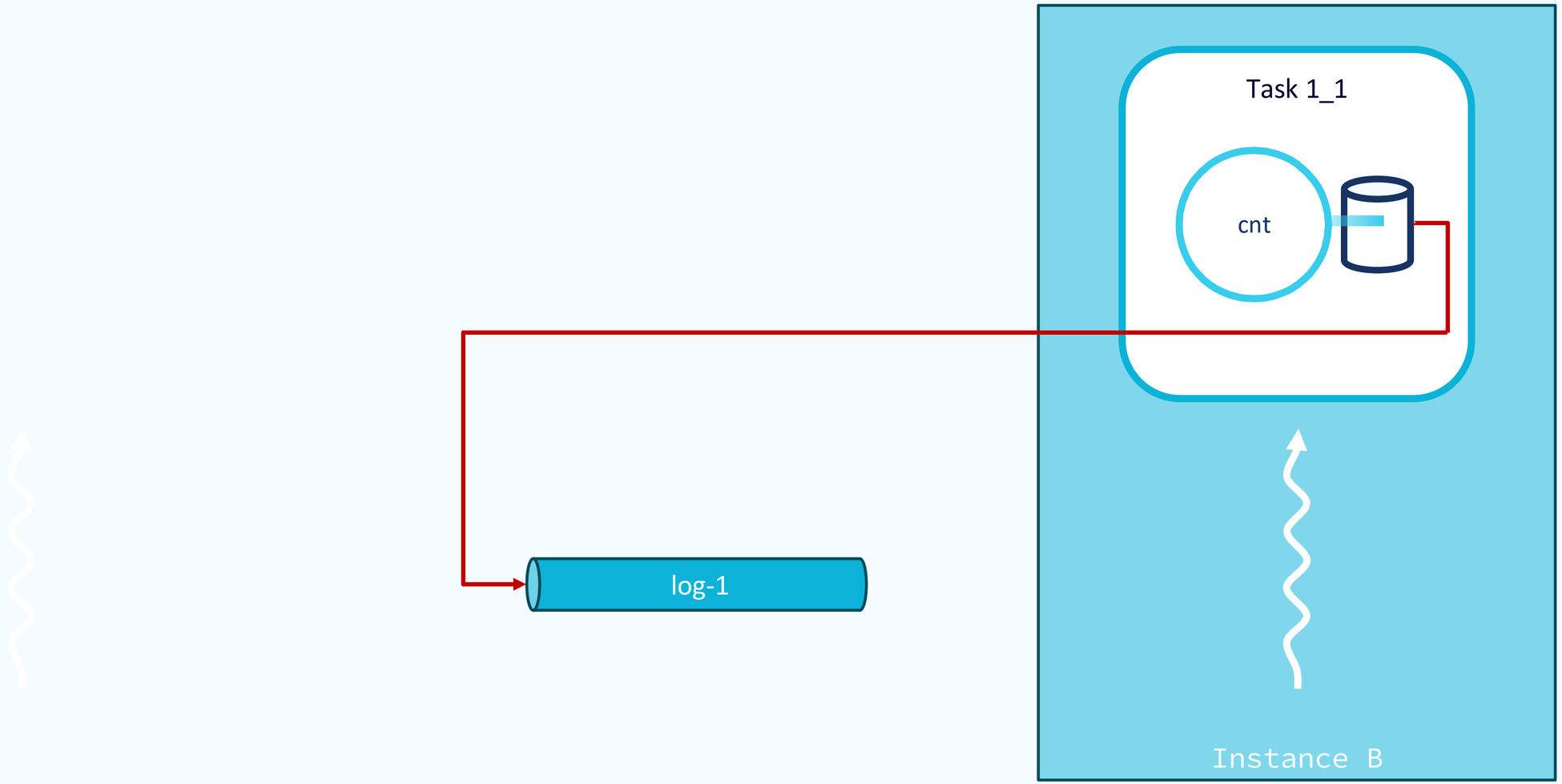
State Stores and Changelog Topics



State Stores and Changelog Topics



State Stores and Changelog Topics



Restore Consumer

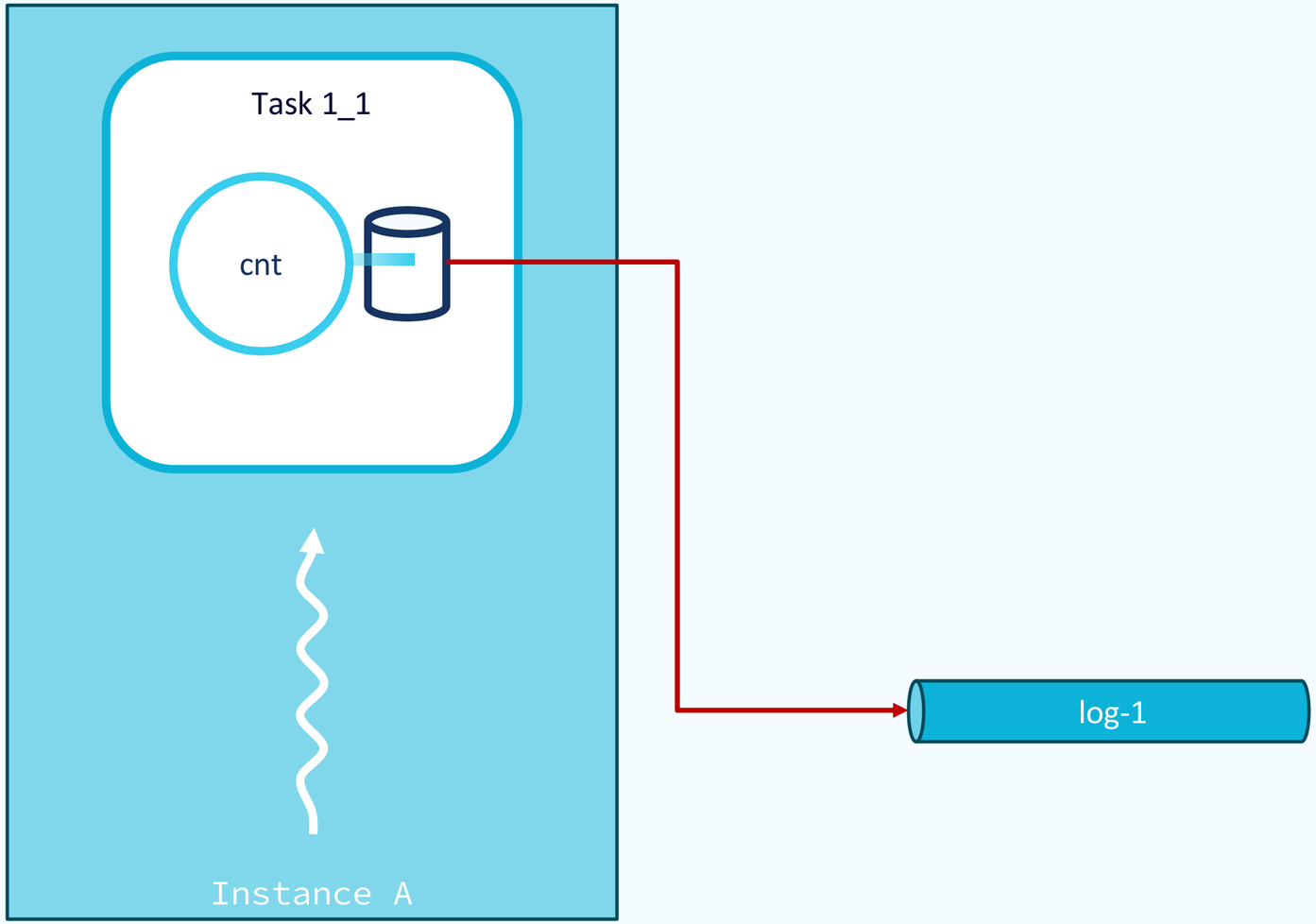
KafkaConsumer (restore)
- no consumer group
- read changelog topics
- used to restore state
- no offset commits

Apache Kafka 3.7 and older
- embedded in main processing loop

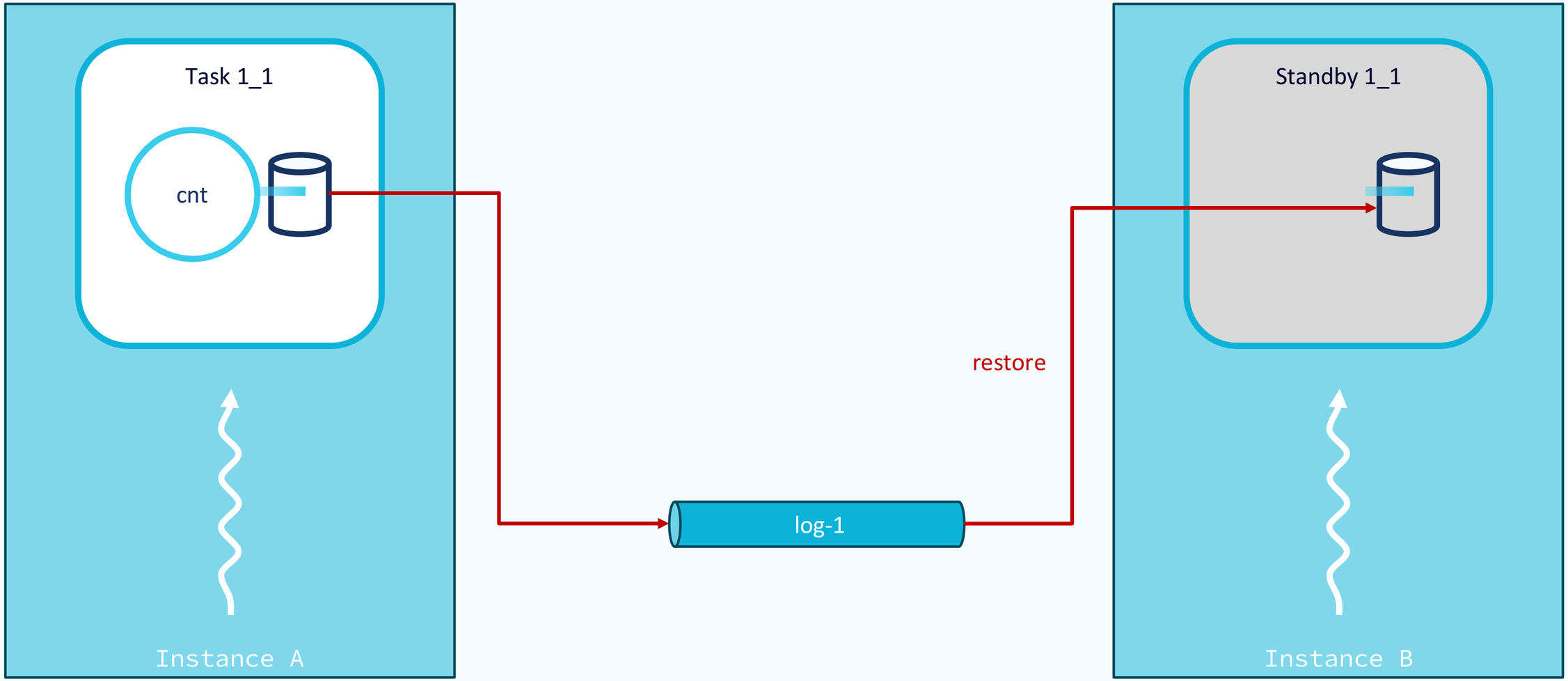
As of Apache Kafka 3.8
- new StateUpdater thread

```
while (running) {
   mainConsumer.poll()

   // version <= 3.7
   if (restoring)
      restoreConsumer.poll()
      restoreState()
      continue;

   // version 3.8+
   checkStateUpdater()

   processActiveTasks()
}
```

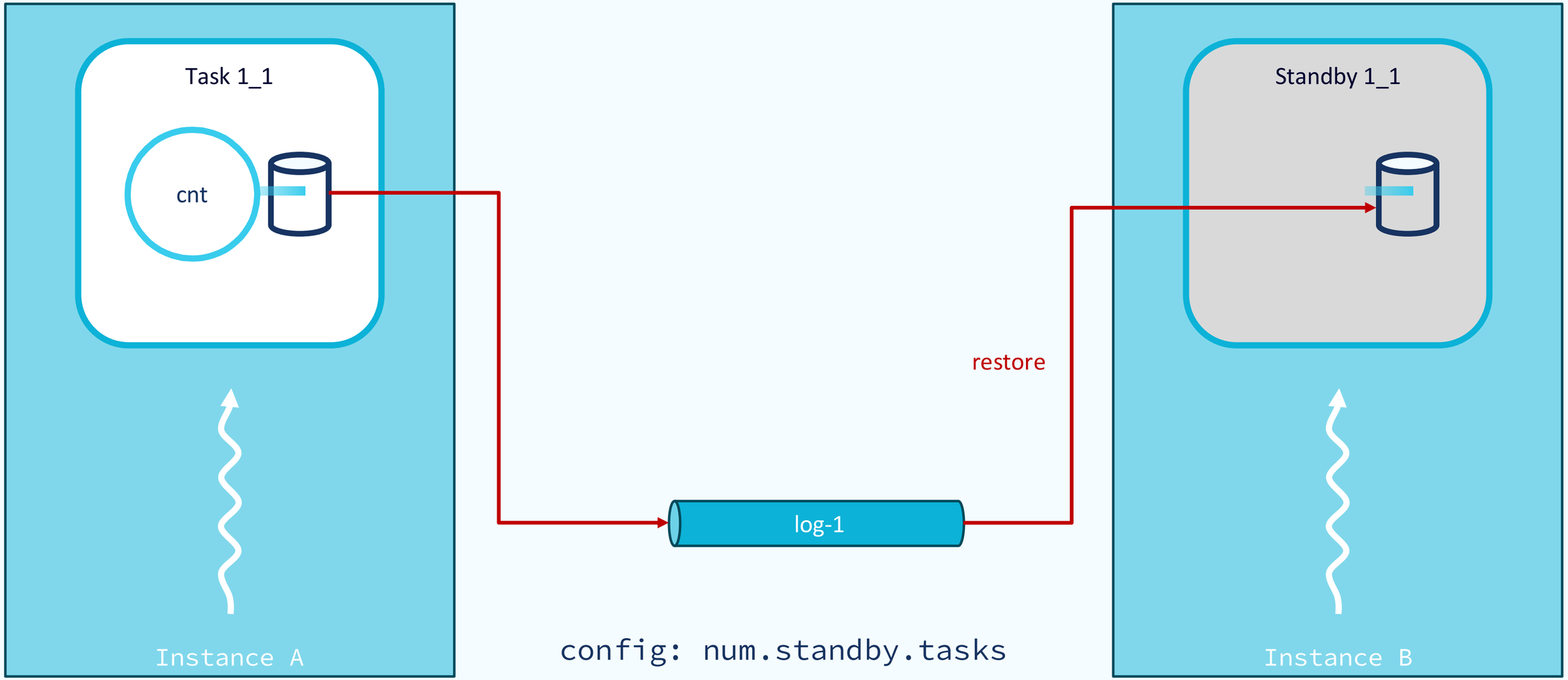
Standby Tasks and High-Availability



Standby Tasks and High-Availability



Standby Tasks and High-Availability



`config: num.standby.tasks`

Restore Consumer

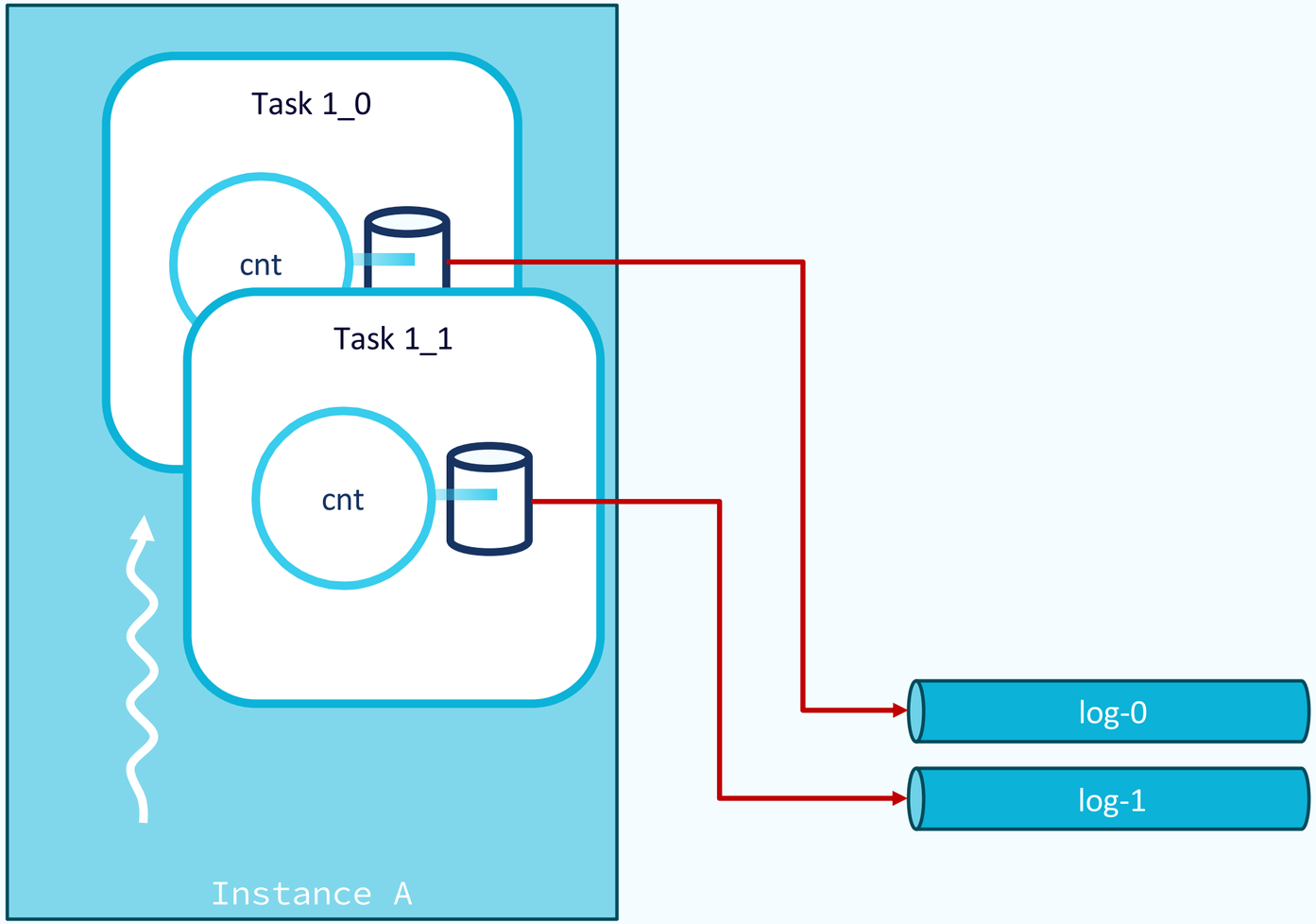
KafkaConsumer (restore)
- no consumer group
- read changelog topics
- used to restore state
- no offset commits

Apache Kafka 3.7 and older
- embedded in main processing loop

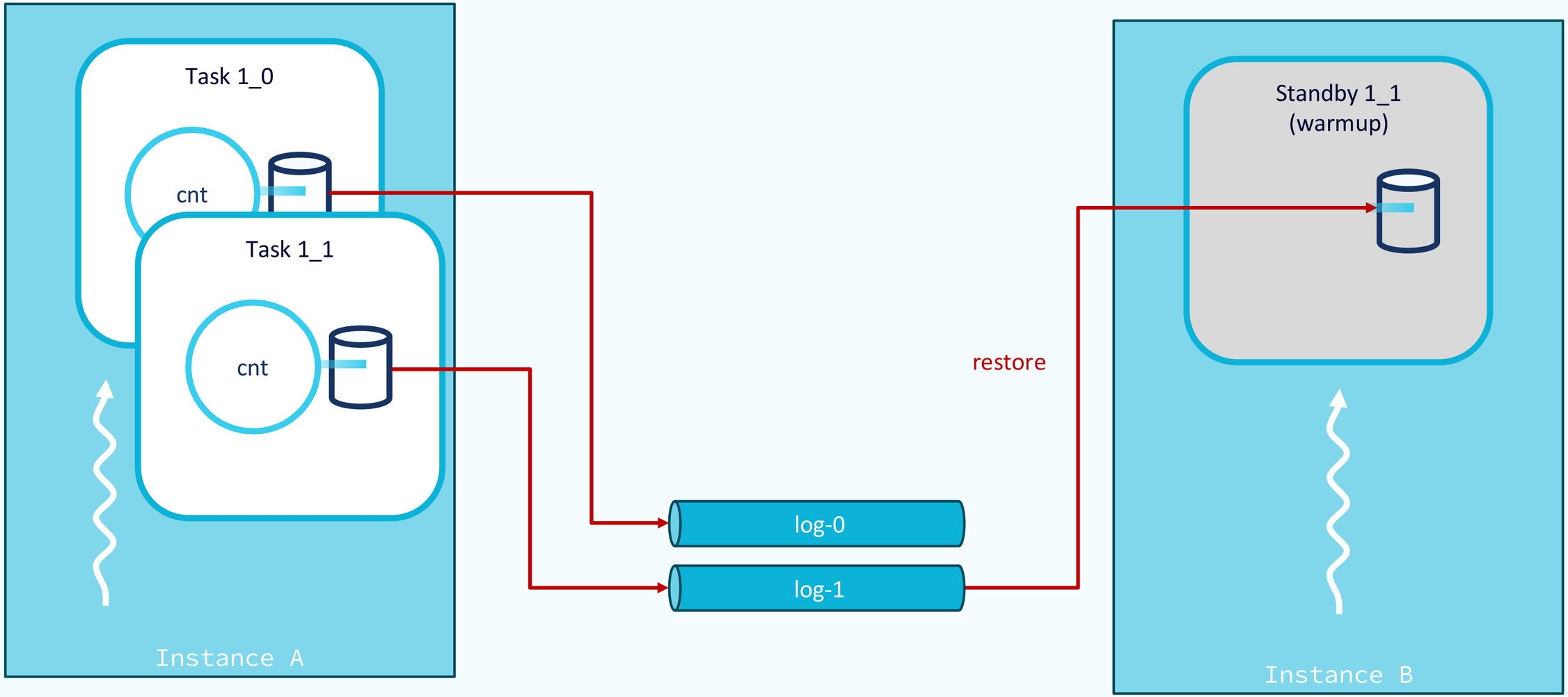
As of Apache Kafka 3.8
- new StateUpdater thread

```
while (running) {
   mainConsumer.poll()

   // version <= 3.7
   if (restoring)
      restoreConsumer.poll()
      restoreState()
      continue;
   else
      restoreConsumer.poll()
      updateStandbys()

   // version 3.8+
   checkStateUpdater()

   processActiveTasks()
}
```

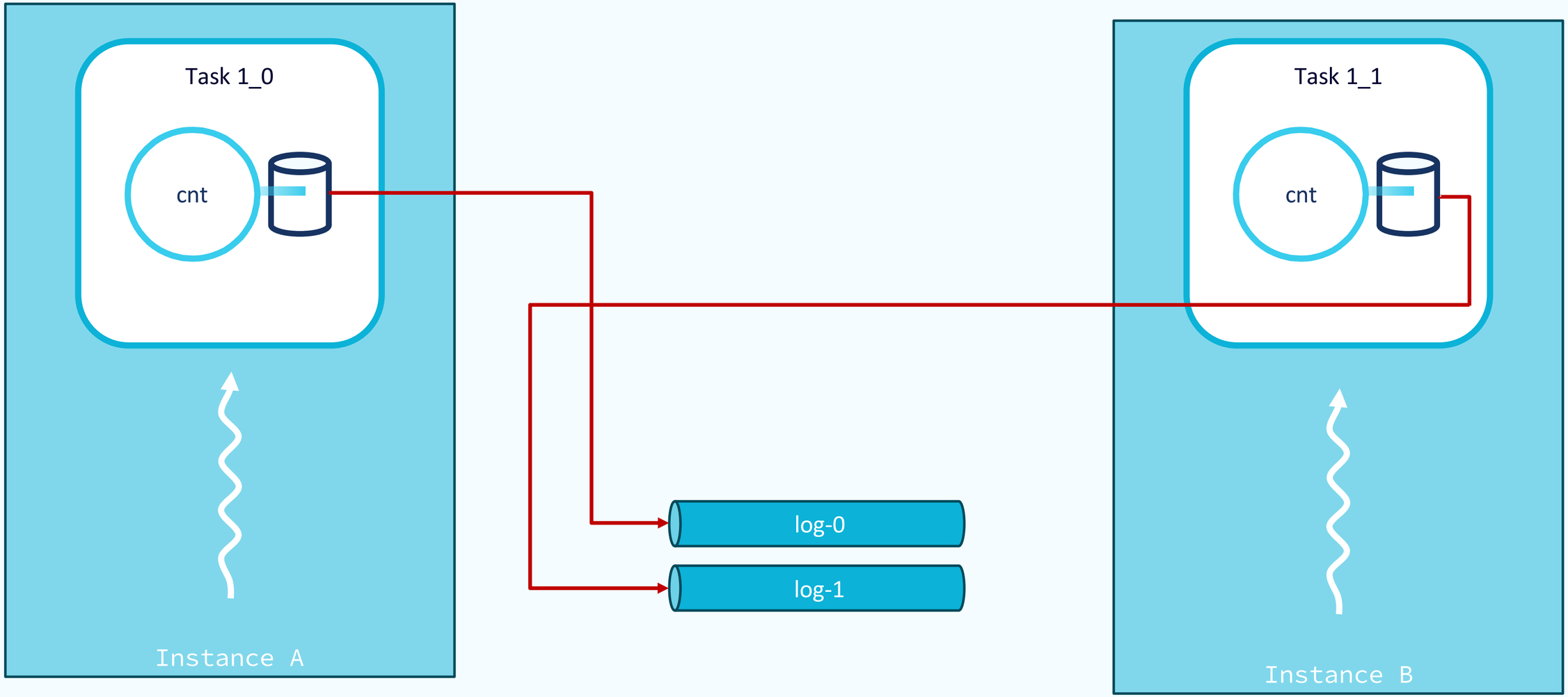
Smooth Scaling with ”Warmup” Tasks



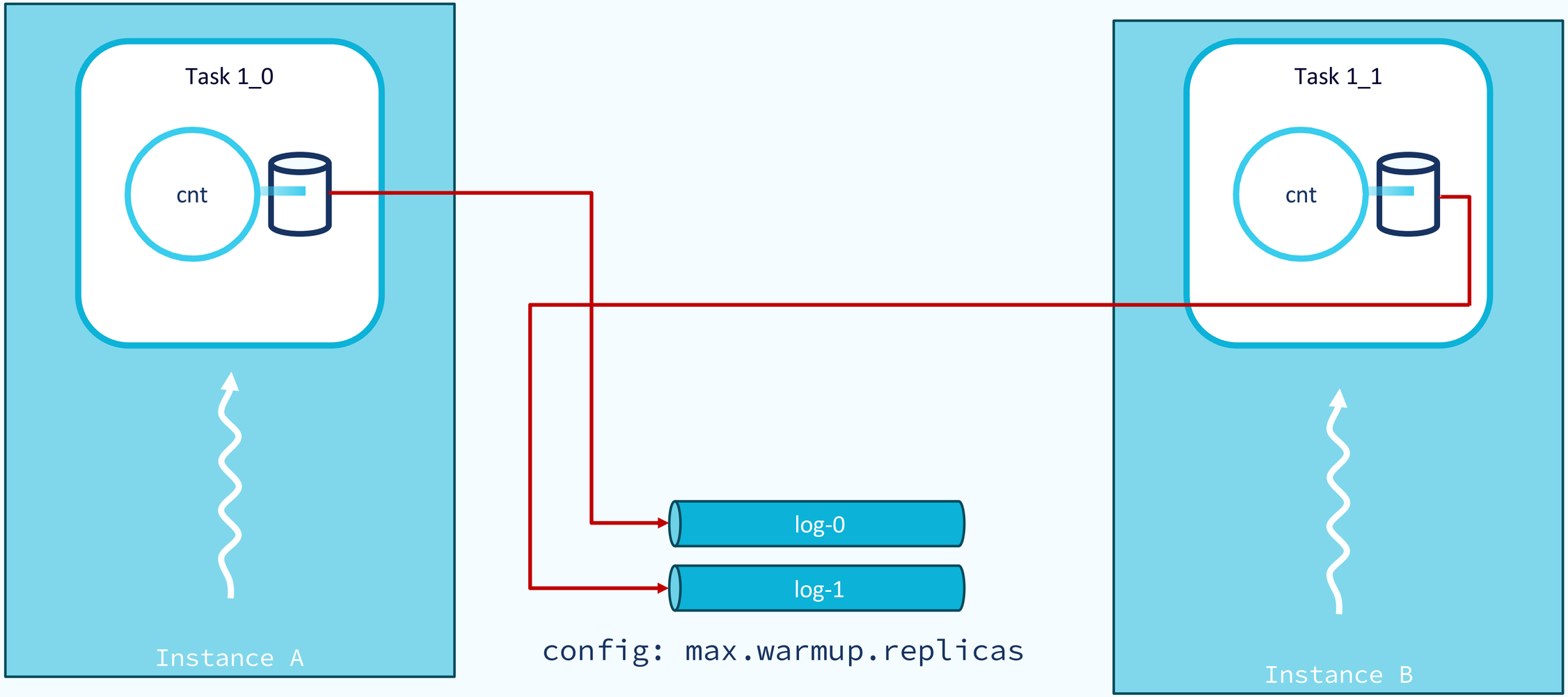
Smooth Scaling with ”Warmup” Tasks



Smooth Scaling with ”Warmup” Tasks

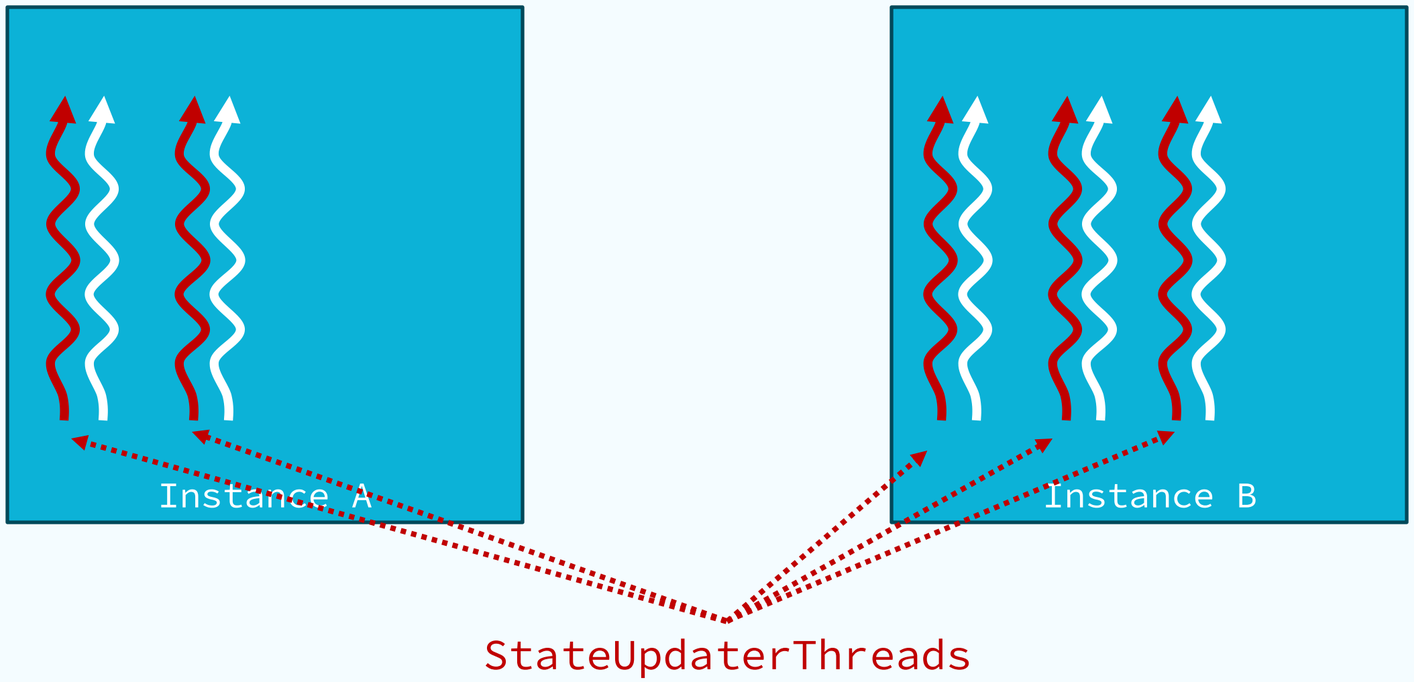


Smooth Scaling with ”Warmup” Tasks

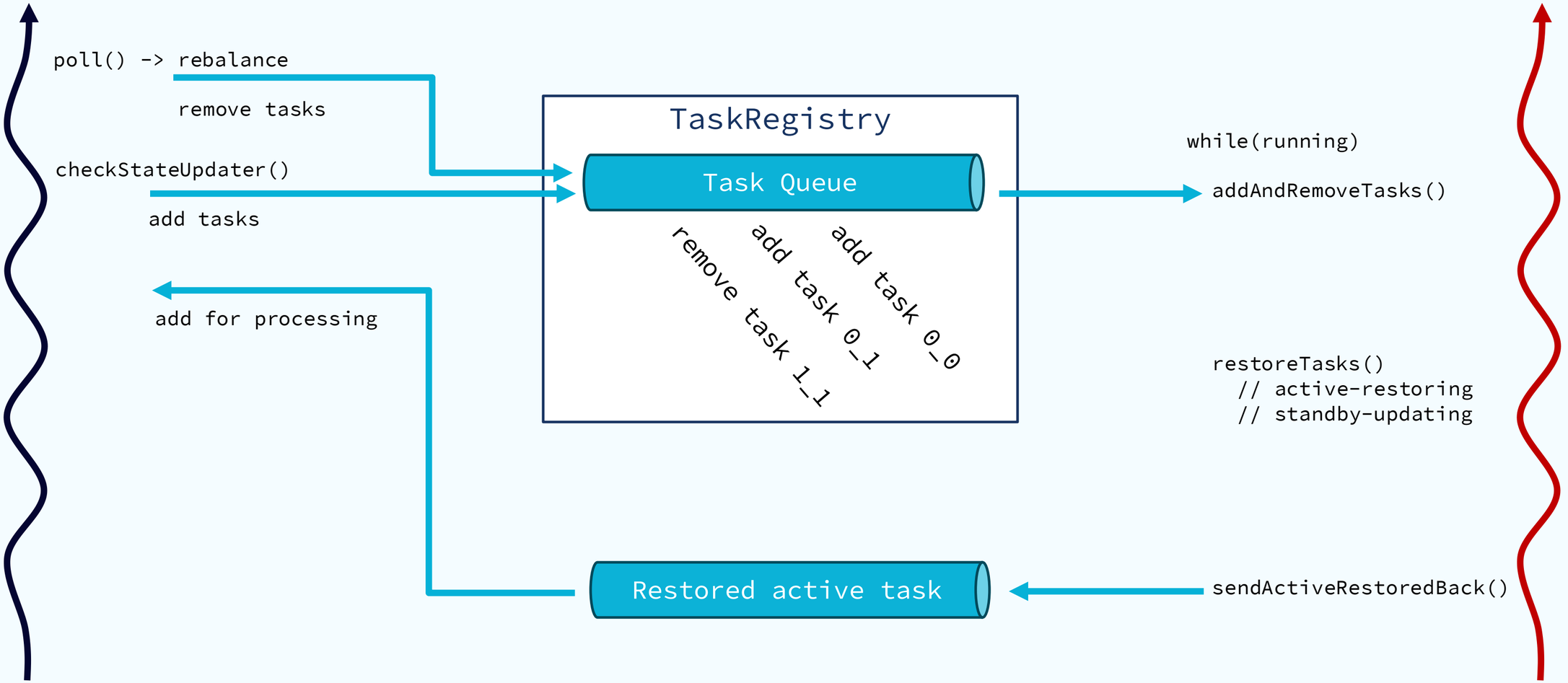


StateUpdater Thread (Apache Kafka 3.8+)

StateUpdaterThread (Apache Kafka 3.8+)

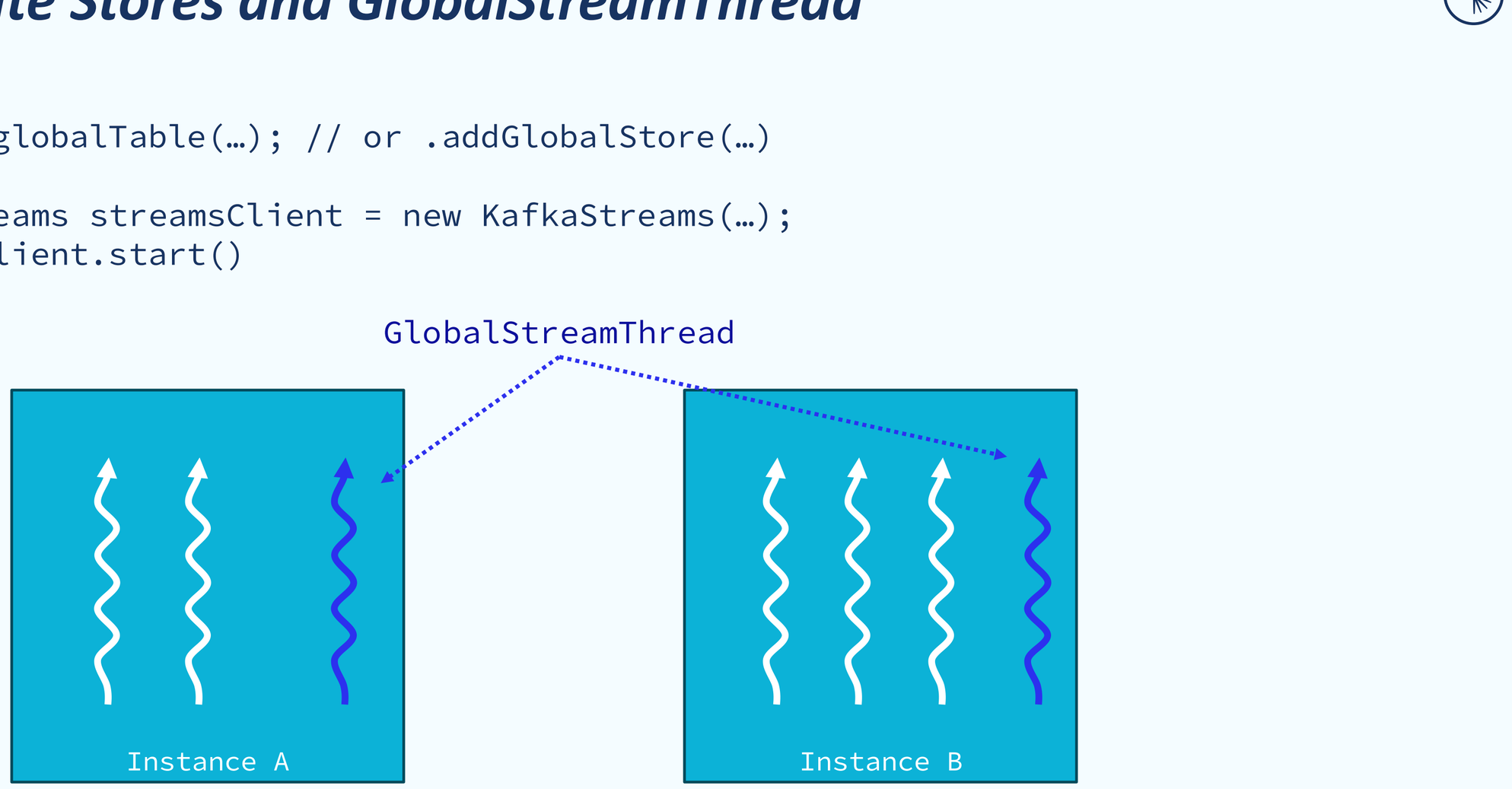


StateUpdaterThread (AK 3.8+)



GlobalKTables/Stores and Threading

Global State Stores and GlobalStreamThread



```
builder.globalTable(…); // or .addGlobalStore(…)

KafkaStreams streamsClient = new KafkaStreams(…);
streamsClient.start()
```

GlobalStreamThread

KafkaConsumer (global)

- no consumer group
- reads *all* global topic partitions
- no offset commits

Global StateStore updating

GlobalStreamThread

KafkaConsumer (global)

- no consumer group
- reads *all* global topic partitions
- no offset commits

Global StateStore updating

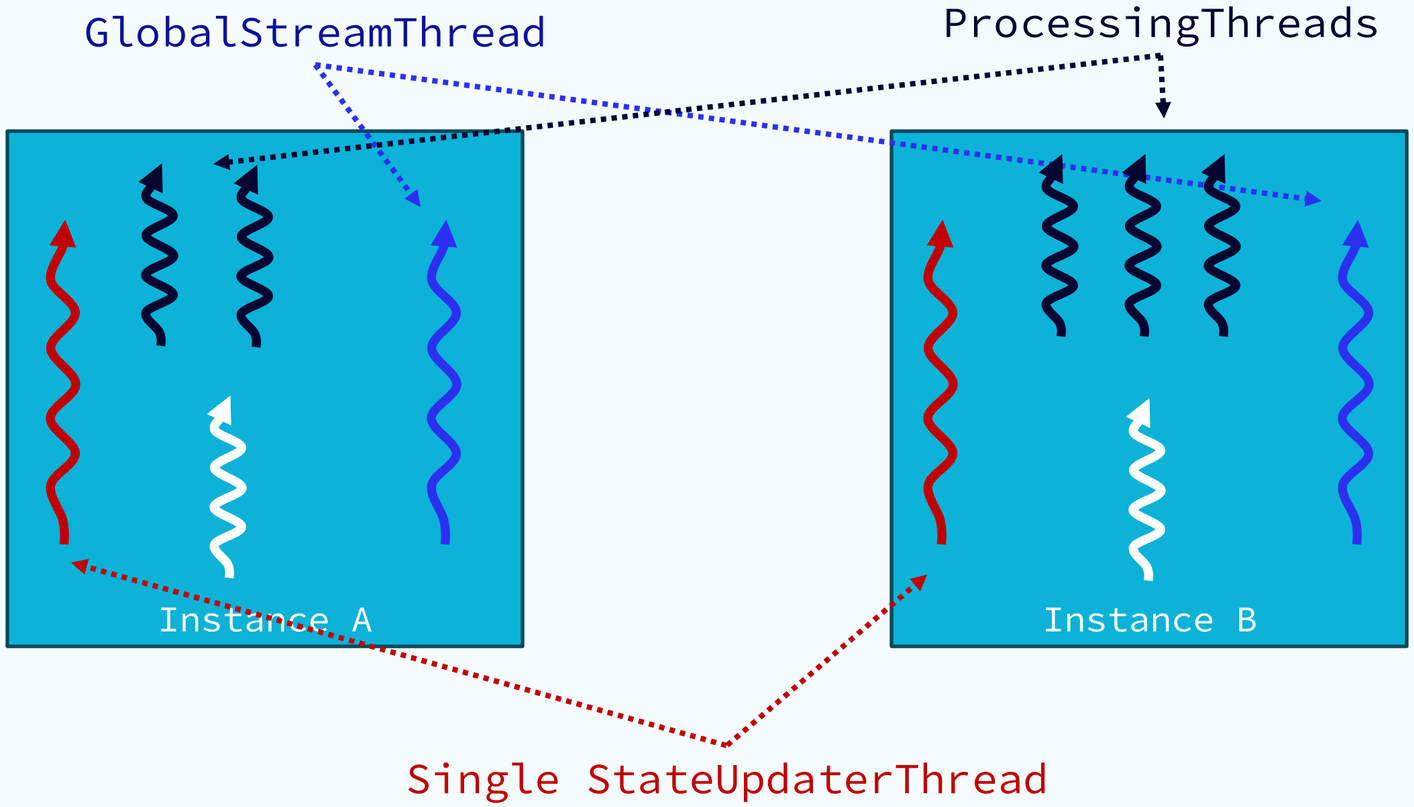
```
// bootstrap global state
start() {
   while(!endOffsetsReached)
      globalConsumer.poll()
      updateGlobalState()
}

// start StreamThreads


// regular processing
while(running) {
   globalConsumer.poll()
   updateGlobalState()
}
```

Thread Refactoring Project

(https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=263429334)

Thread Refactoring – Phase 2 (on hold)

Decouple consumer and processing threads



CONFLUENT

The Nuts and Bolts of Kafka Streams: An Architectural Deep Dive (’24 update)

Matthias J. Sax
Software Engineer | Apache Kafka Committer and PMC member

Twitter/𝕏 @MatthiasJSax