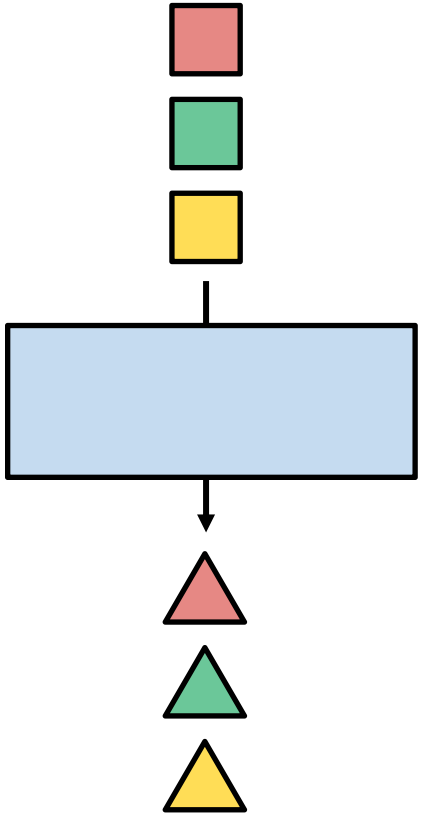
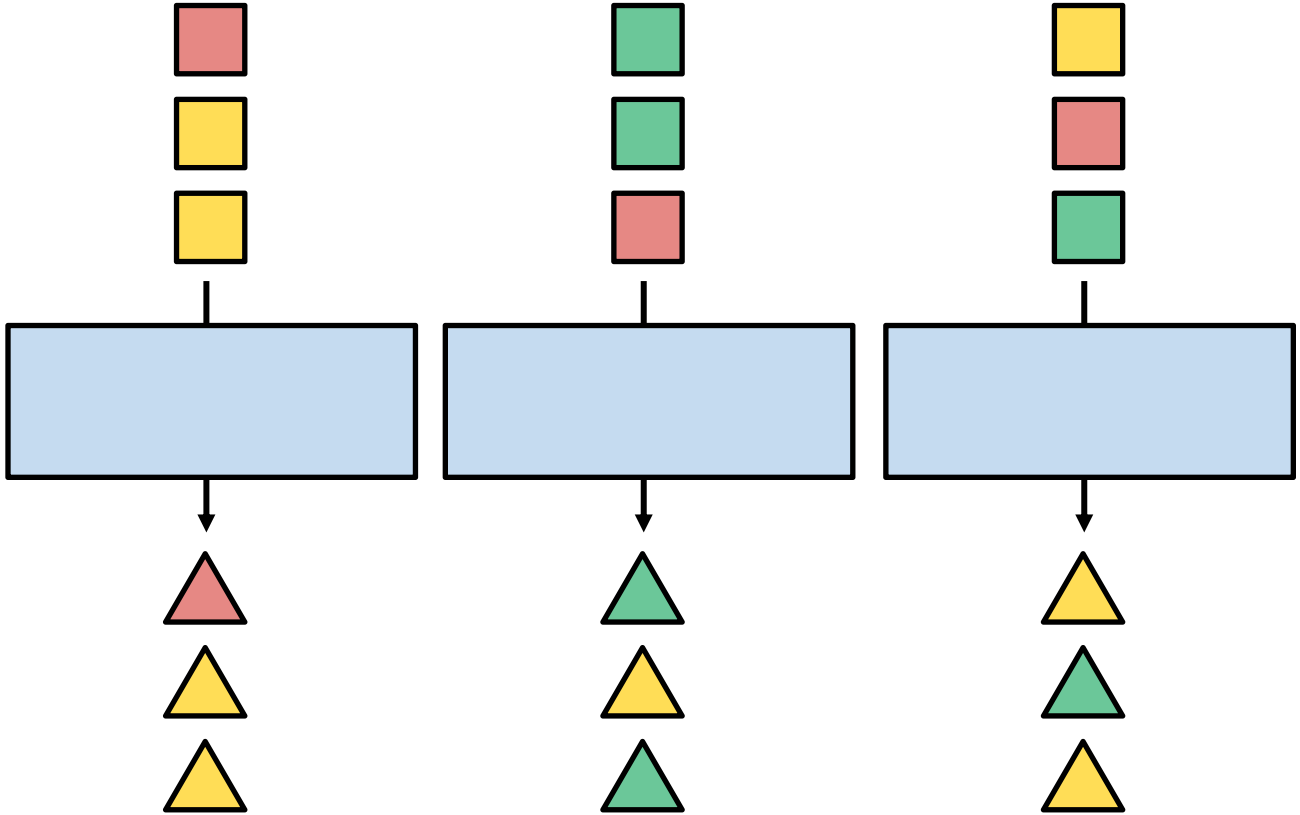
dataArtisans

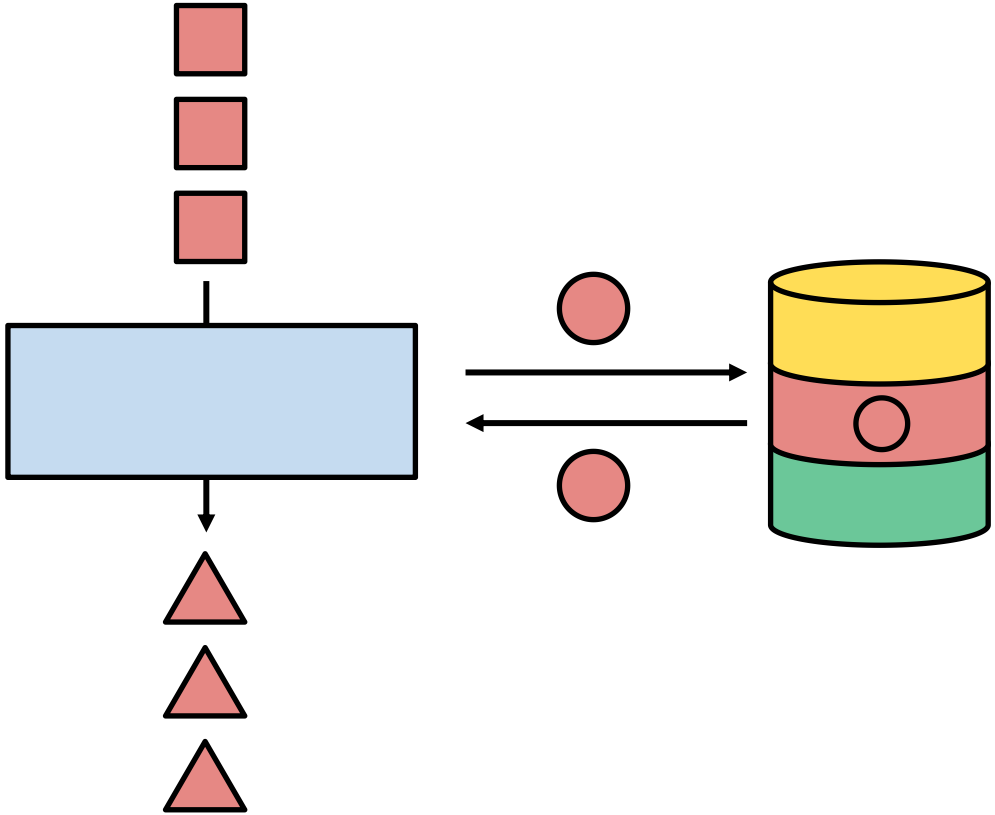
dataArtisans

dA

PLATFORM







-
  -
  -
  -
  -
-
  -

```java
/**
  * Process one element from the input stream.
  */
void processElement(I value, Context ctx, Collector<O> out) throws Exception;


/**
  * Called when a timer set using {@link TimerService} fires.
  */
void onTimer(long timestamp, OnTimerContext ctx, Collector<O> out) throws Exception;
```

```
/**
  * Process one element from the input stream.
  */
void processElement(I value, Context ctx, Collector<O> out) throws Exception;


/**
  * Called when a timer set using {@link TimerService} fires.
  */
void onTimer(long timestamp, OnTimerContext ctx, Collector<O> out) throws Exception;
```

```
/**
  * Process one element from the input stream.
  */
void processElement(I value, Context ctx, Collector<O> out) throws Exception;


/**
  * Called when a timer set using {@link TimerServ
  */
void onTimer(long timestamp, OnTimerContext ctx, Collector<O> out) throws Exception;
```

- ValueState
-
  -
  -
-
  -
  -

```
public class MyProcessFunction extends
        ProcessFunction<Tuple2<String, String>, Tuple2<String, Long>> {

    // define your state descriptors

    @Override
    public void processElement(Tuple2<String, Long> value, Context ctx,
            Collector<Tuple2<String, Long>> out) throws Exception {
        // update our state and register a timer
    }

    @Override
    public void onTimer(long timestamp, OnTimerContext ctx,
                Collector<Tuple2<String, Long>> out) throws Exception {
        // check the state for the key and emit a result if needed
    }
}
```

```
public class MyProcessFunction extends
         ProcessFunction<Tuple2<String, String>, Tuple2<String, Long>> {

     // define your state descriptors
     private final ValueStateDescriptor<CounterWithTS> stateDesc =
          new ValueStateDescriptor<>("myState", CounterWithTS.class);





}
```

```java
public class MyProcessFunction extends
        ProcessFunction<Tuple2<String, String>, Tuple2<String, Long>> {

    @Override
    public void processElement(Tuple2<String, String> value, Context ctx,
            Collector<Tuple2<String, Long>> out) throws Exception {

        ValueState<MyStateClass> state = getRuntimeContext().getState(stateDesc);
        CounterWithTS current = state.value();
        if (current == null) {
            current = new CounterWithTS();
            current.key = value.f0;
        }
        current.count++;
        current.lastModified = ctx.timestamp();
        state.update(current);
        ctx.timerService().registerEventTimeTimer(current.lastModified + 100);
    }
}
```

```
public class MyProcessFunction extends
        ProcessFunction<Tuple2<String, String>, Tuple2<String, Long>> {

    @Override
    public void onTimer(long timestamp, OnTimerContext ctx,
                        Collector<Tuple2<String, Long>> out) throws Exception {

        CounterWithTS result = getRuntimeContext().getState(stateDesc).value();
        if (timestamp == result.lastModified + 100) {
            out.collect(new Tuple2<String, Long>(result.key, result.count));      }

    }
}
```

```
stream.keyBy(”key”)
.process(new MyProcessFunction())
```

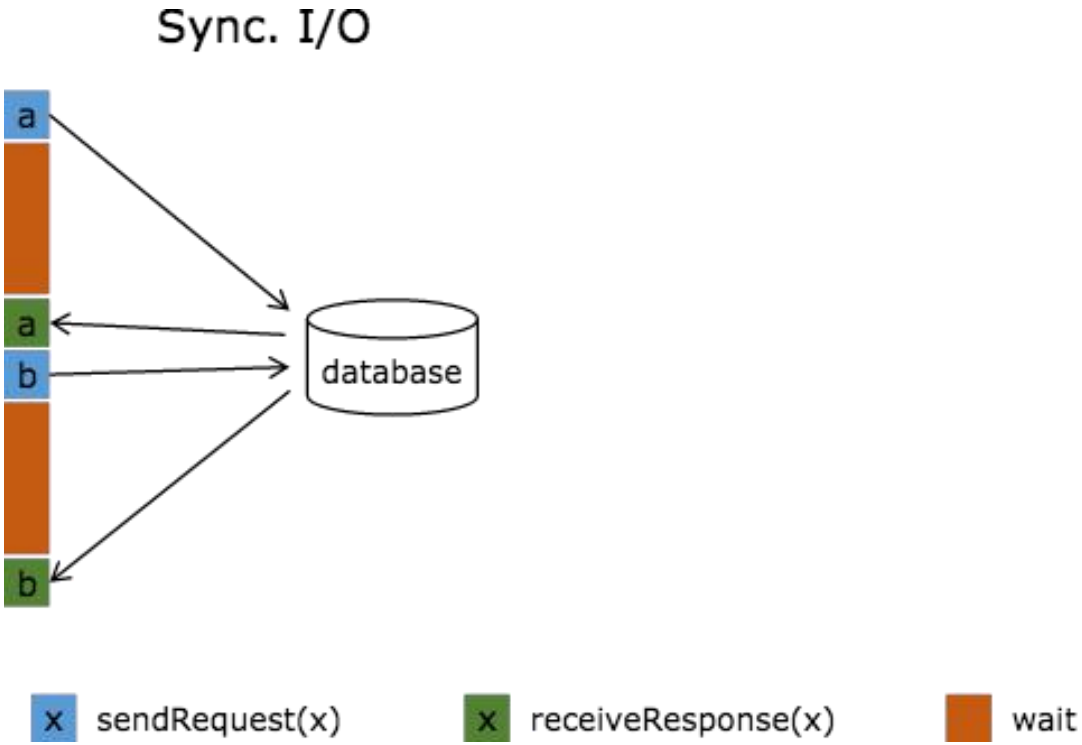
```
final OutputTag<String> outputTag = new OutputTag<String>(”gt10"){};

SingleOutputStreamOperator<Tuple2<String, Long>> mainStream = input.process(
     new ProcessFunction<Tuple2<String, String>, Tuple2<String, Long>>() {

     @Override
     public void onTimer(long timestamp, OnTimerContext ctx,
                    Collector<Tuple2<String, Long>> out) throws Exception {
        CounterWithTS result = getRuntimeContext().getState(adStateDesc).value();
        if (timestamp == result.lastModified + 100) {
             out.collect(new Tuple2<String, Long>(result.key, result.count));
        } else if (result.count > 10) {
             ctx.output(outputTag, result.key);
        }
}

DataStream<String> sideOutputStream = mainStream.getSideOutput(outputTag);
```

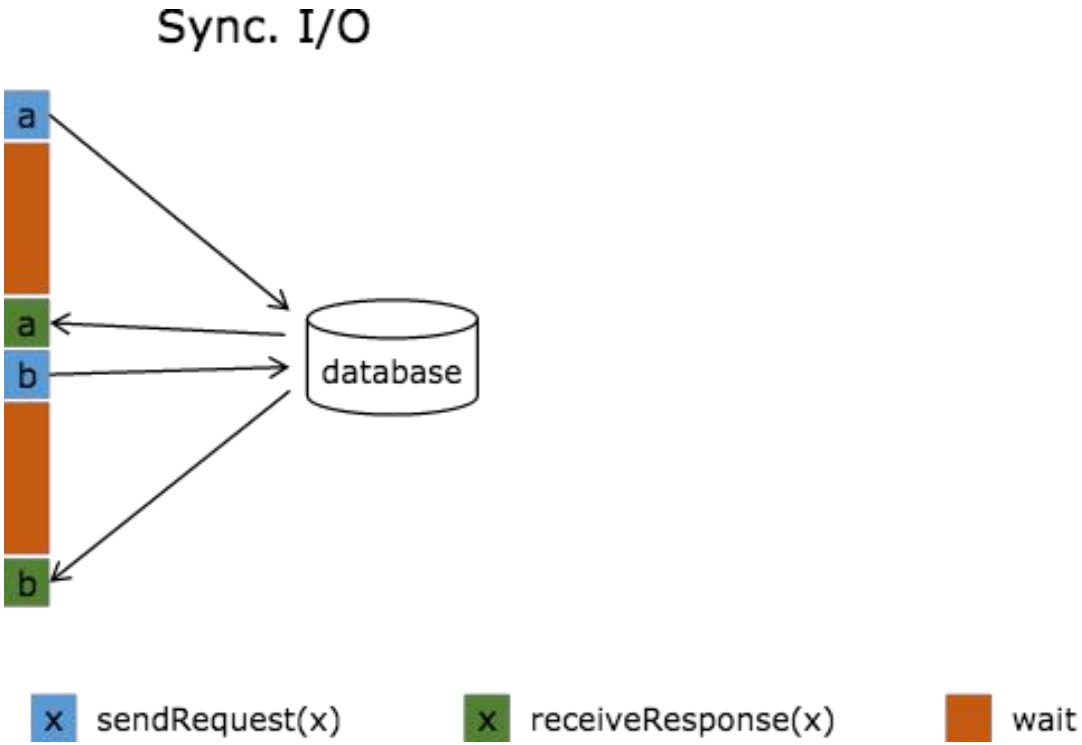
```
final OutputTag<String> outputTag = new OutputTag<String>(”gt10"){};

SingleOutputStreamOperator<Tuple2<String, Long>> mainStream = input.process(
    new ProcessFunction<Tuple2<String, String>, Tuple2<String, Long>>() {

    @Override
    public void onTimer(long timestamp, OnTimerContext ctx,
                        Collector<Tuple2<String, Long>> out) throws Exception {
      CounterWithTS result = getRuntimeContext().getState(adStateDesc).value();
      if (timestamp == result.lastModified + 100) {
          out.collect(new Tuple2<String, Long>(result.key, result.count));
      } else if (result.count > 10) {
          ctx.output(outputTag, result.key);
      }
}

DataStream<String> sideOutputStream = mainStream.getSideOutput(outputTag);
```

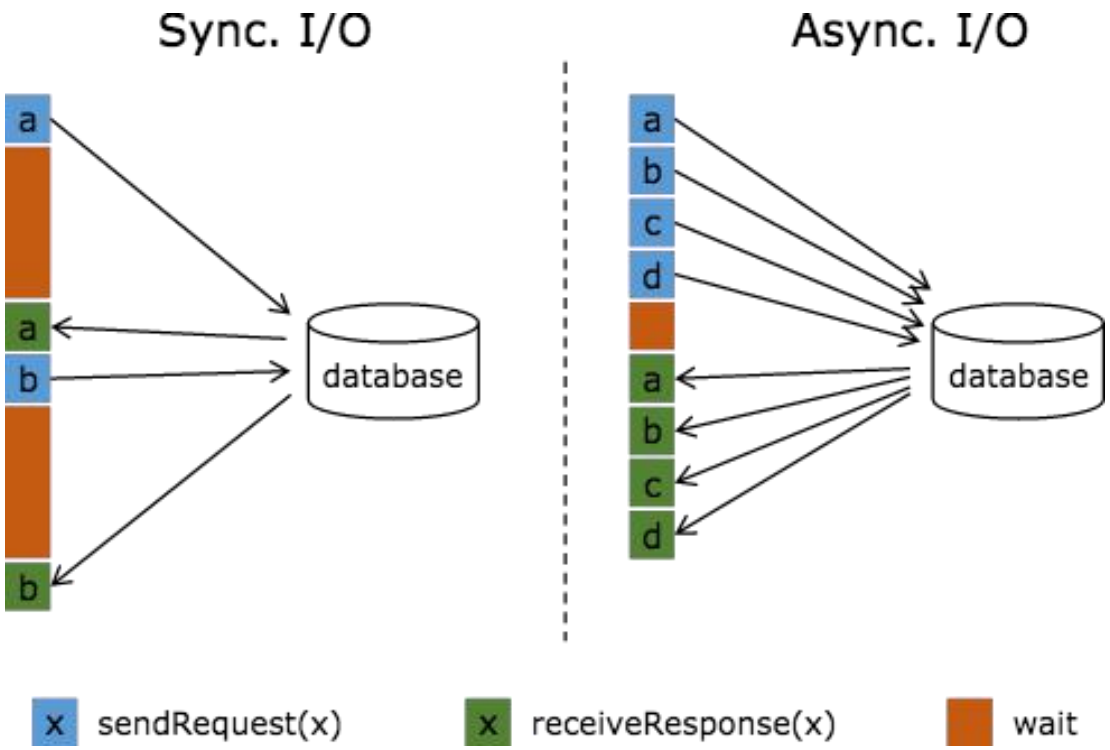
- CoProcessFunction

MapFunction

MapFunction

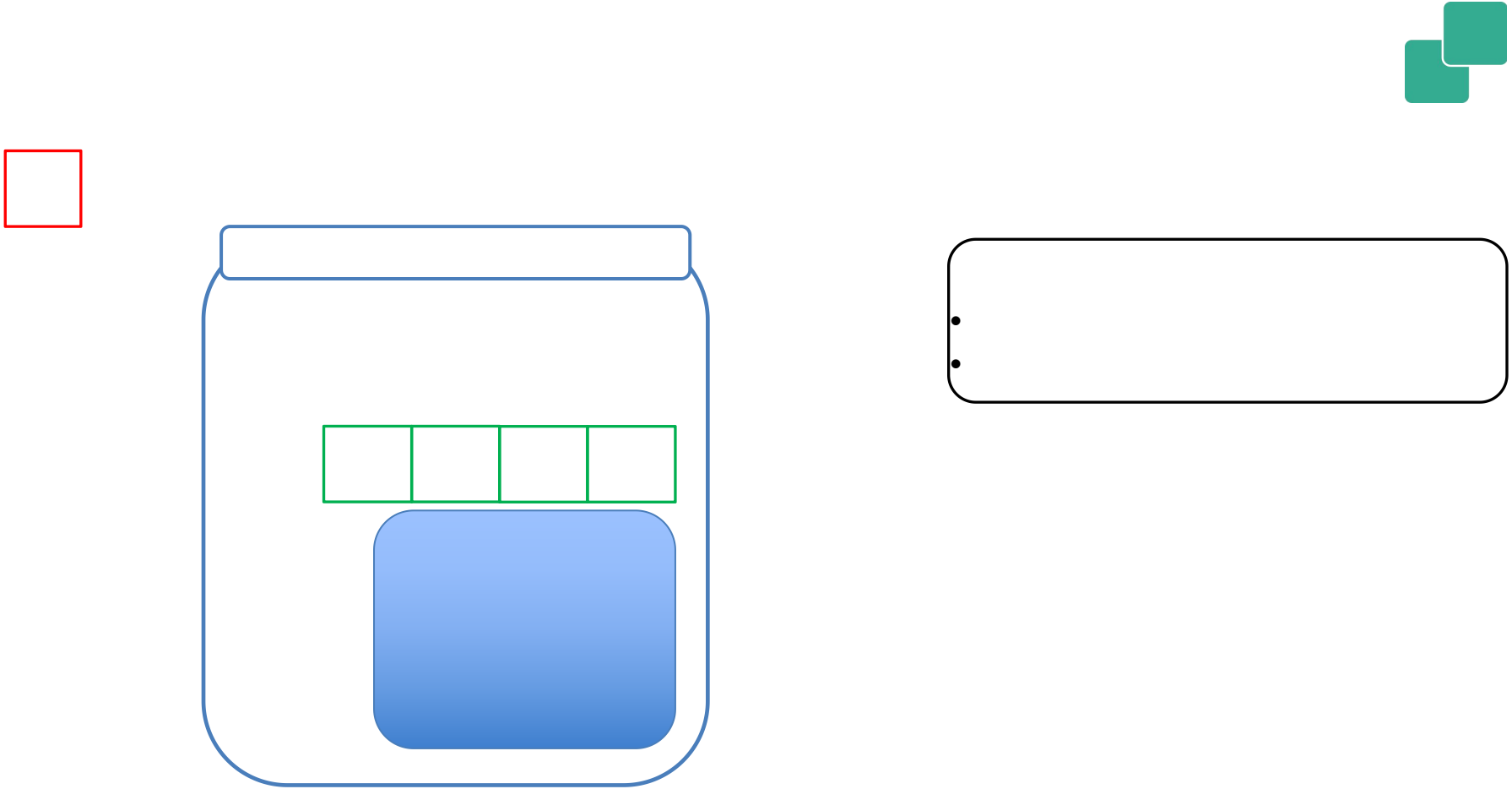

Sync. I/O
a
a
b
b
database
x sendRequest(x)
x receiveResponse(x)
wait

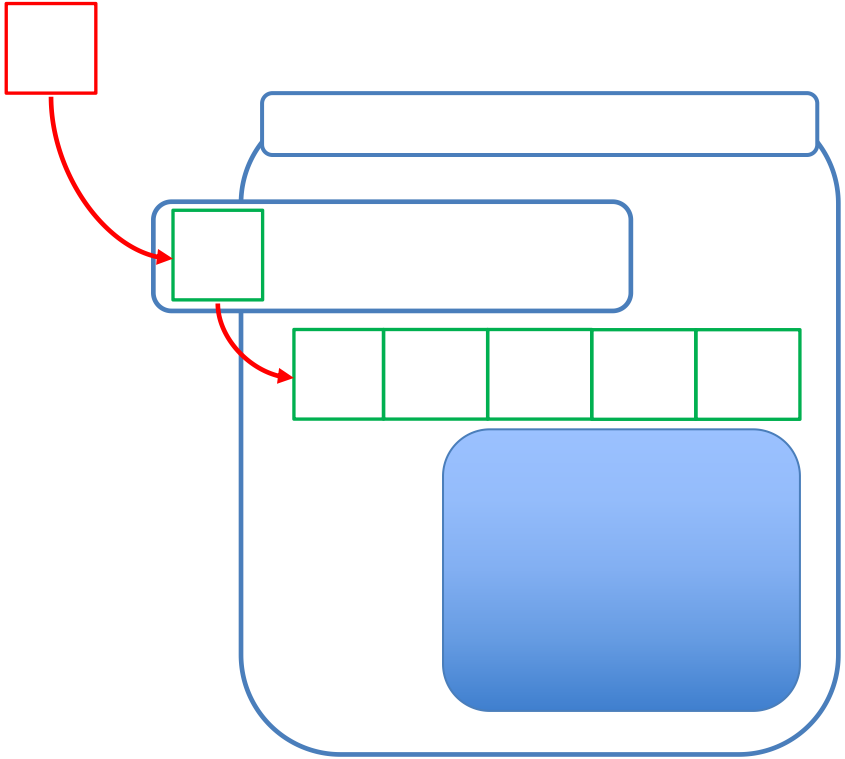

Sync. I/O
a
a
b
b
database
x sendRequest(x)
x receiveResponse(x)
wait


Sync. I/O
Async. I/O
a
a
b
b
database
a
b
c
d
a
b
c
d
database
x sendRequest(x)
x receiveResponse(x)
wait

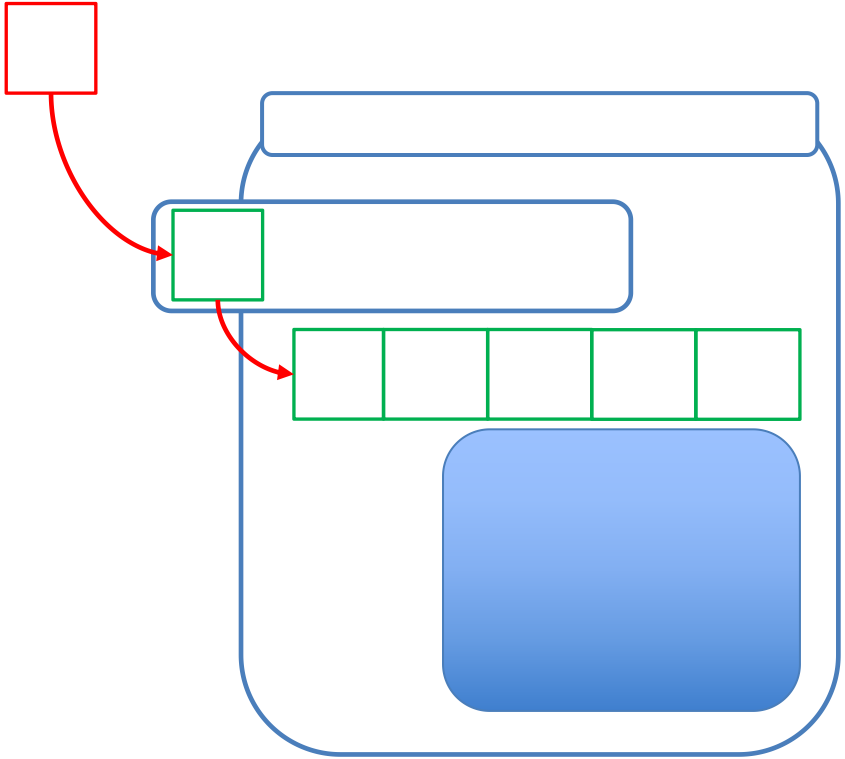
```
/**
  * Trigger async operation for each stream input.
  */
void asyncInvoke(IN input, AsyncCollector<OUT> collector) throws Exception;
```

```
/**
  * Example async function call.
  */
DataStream<...> result = AsyncDataStream.(un)orderedWait(stream,
                  new MyAsyncFunction(), 1000, TimeUnit.MILLISECONDS, 100);
```

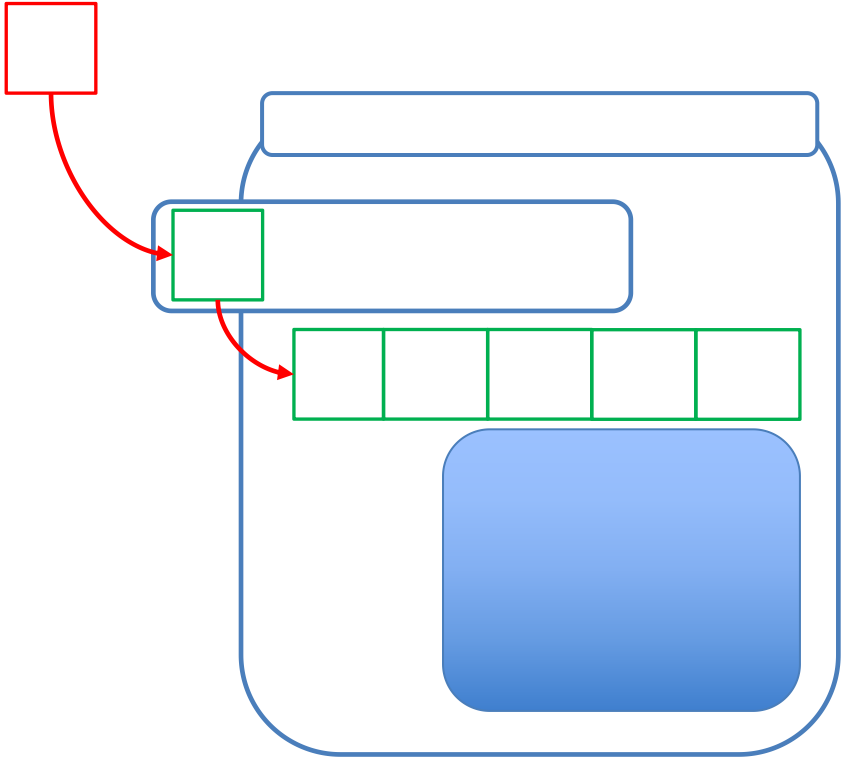




-
-
-

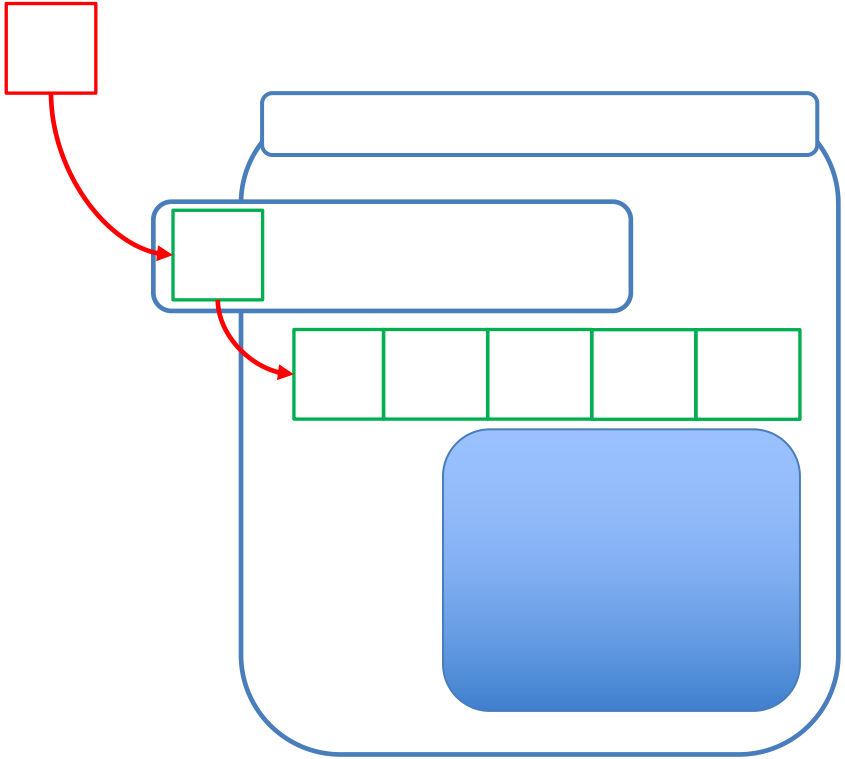


-
-
-

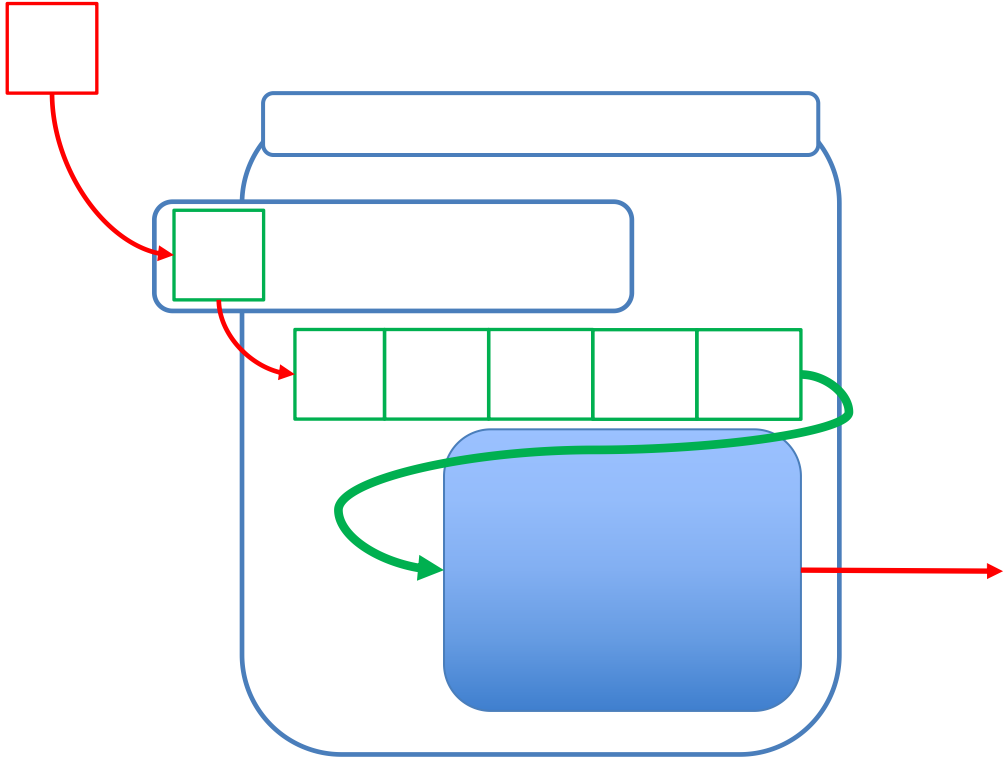


-
-
-

```
Future<String> future = client.query(  );

future.thenAccept((String result) -> {
       .collect(
           Collections.singleton(
                new Tuple2<>(  , result)));
});
```



```
Future<String> future = client.query(  );

future.thenAccept((String result) -> {
       .collect(
           Collections.singleton(
                new Tuple2<>(  , result)));
});
```



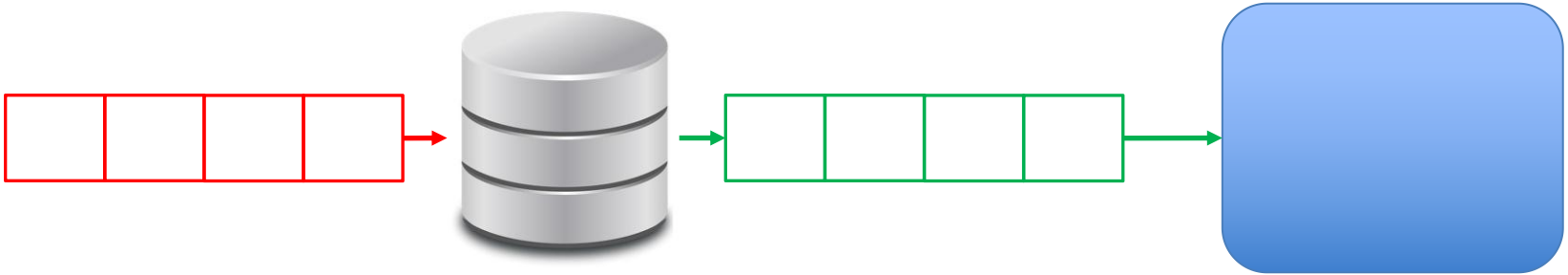
-
-
-

```
DataStream<Tuple2<String, String>> result =
      AsyncDataStream.(un)orderedWait(stream,
                    new MyAsyncFunction(),
                    1000, TimeUnit.MILLISECONDS,
                    100);
```

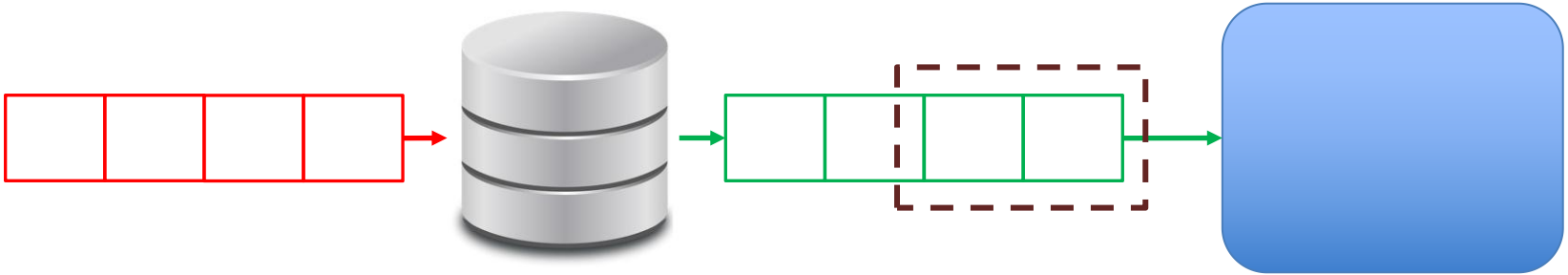
- asyncFunction

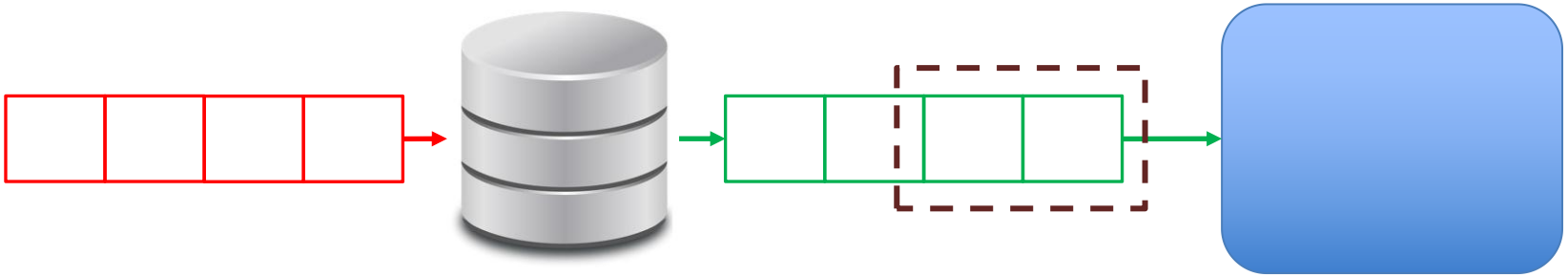
```
DataStream<Tuple2<String, String>> result =
      AsyncDataStream.(un)orderedWait(stream,
                      new MyAsyncFunction(),
                      1000, TimeUnit.MILLISECONDS,
                      100);
```

```
DataStream<Tuple2<String, String>> result =
      AsyncDataStream.(un)orderedWait(stream,
                      new MyAsyncFunction(),
                      1000, TimeUnit.MILLISECONDS,
                      100);
```



```
DataStream<Tuple2<String, String>> result =
      AsyncDataStream.unorderedWait(stream,
                    new MyAsyncFunction(),
                    1000, TimeUnit.MILLISECONDS,
                    100);
```





- unorderedWait
- orderedWait

FLINK FORWARD

FLINK FORWARD IS COMING BACK TO BERLIN

SEPTEMBER 11-13, 2017

BERLIN.FLINK-FORWARD.ORG

dataArtisans