Flink Checkpoint-轻量级分布式快照

唐云 · 阿里巴巴 / 高级开发工程师

Apache Flink Community China

CONTENT

目录 >>

01 /

总览: Checkpoint与state

02 /

什么是state

03 /

如何在Flink中使用state

04 /

Checkpoint的执行机制

01

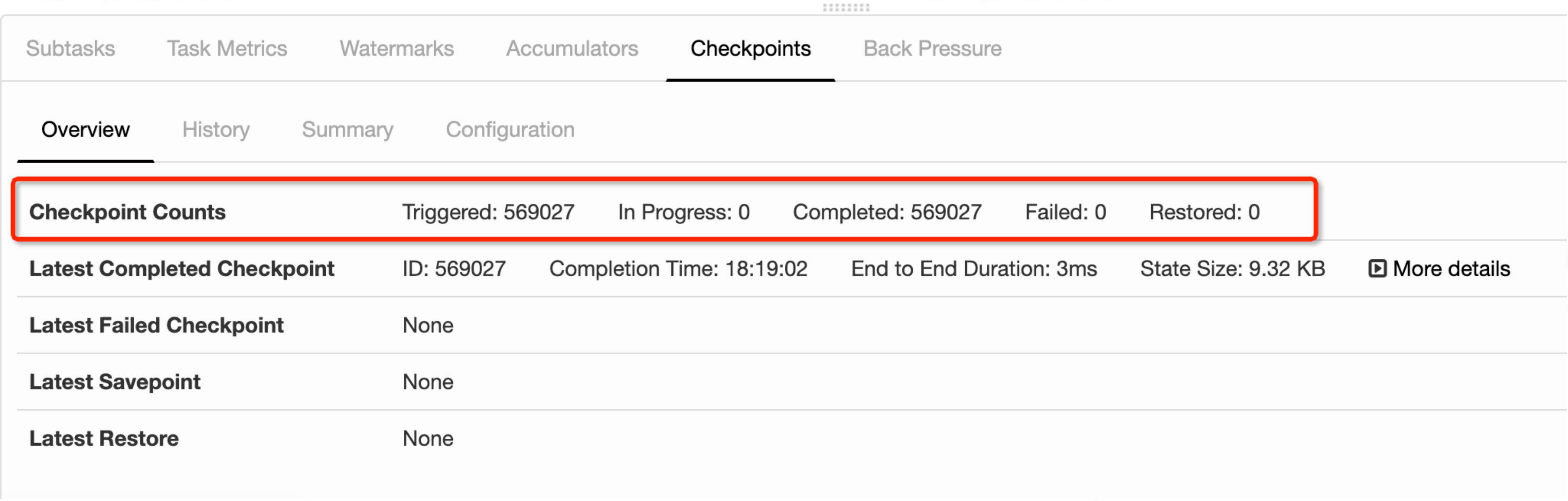
Checkpoint与State

二者的关系

Checkpoint与State

Apache Flink

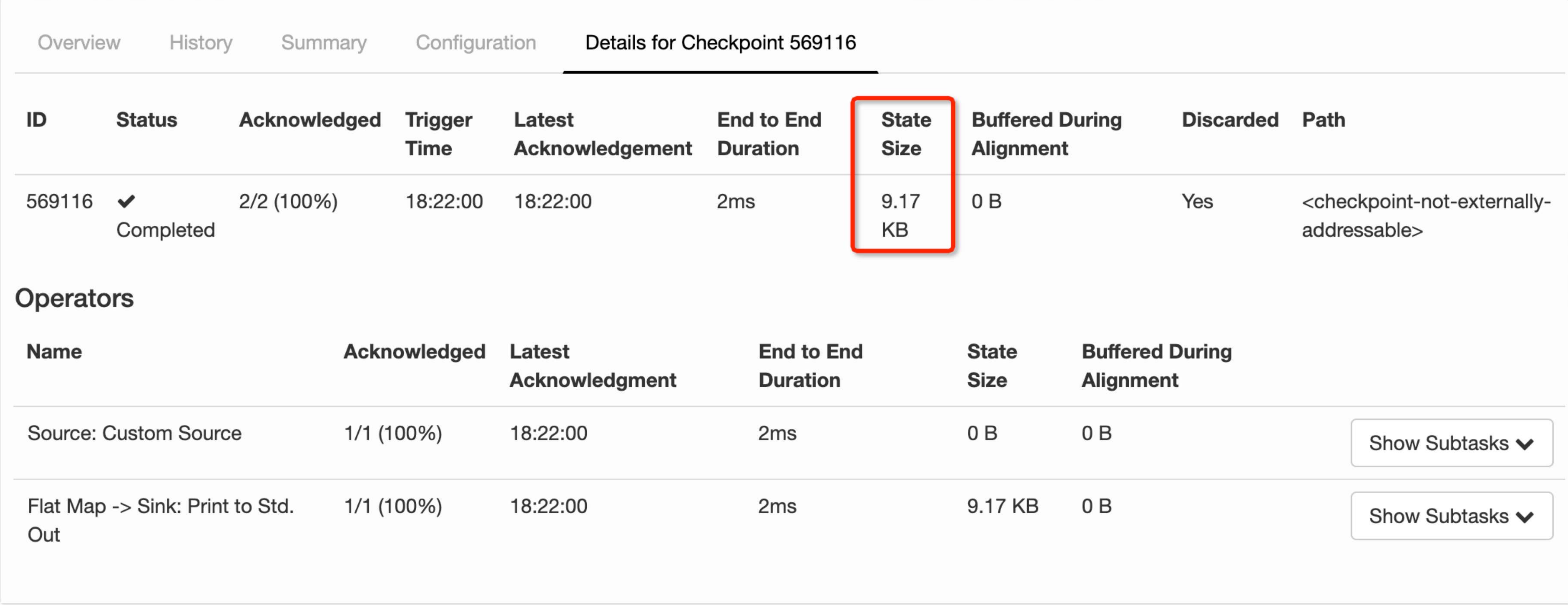
Checkpoint，在Flink中是一个执行操作，最终产生的结果作为分布式快照提供容错机制。



Checkpoint Counts	Triggered: 569027	In Progress: 0	Completed: 569027	Failed: 0	Restored: 0
Latest Completed Checkpoint	ID: 569027	Completion Time: 18:19:02	End to End Duration: 3ms	State Size: 9.32 KB	More details
Latest Failed Checkpoint	None				
Latest Savepoint	None				
Latest Restore	None				

Checkpoint与State

State是构成checkpoint的数据构成



ID	Status	Acknowledged	Trigger Time	Latest Acknowledgement	End to End Duration	State Size	Buffered During Alignment	Discarded	Path
569116	✓ Completed	2/2 (100%)	18:22:00	18:22:00	2ms	9.17 KB	0 B	Yes	<checkpoint-not-externally-addressable>

Operators

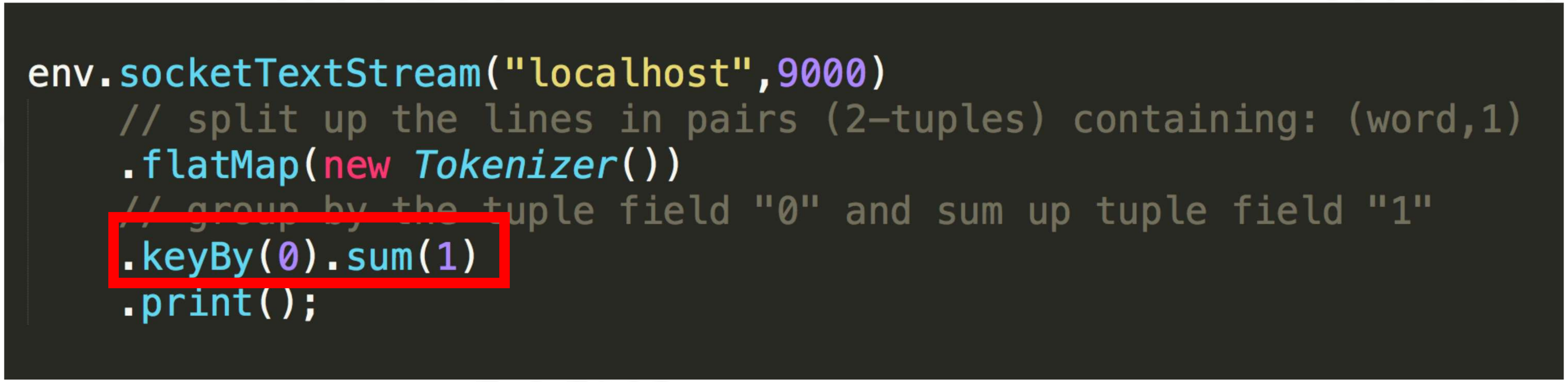
Name	Acknowledged	Latest Acknowledgment	End to End Duration	State Size	Buffered During Alignment	
Source: Custom Source	1/1 (100%)	18:22:00	2ms	0 B	0 B	Show Subtasks ⌄
Flat Map -> Sink: Print to Std. Out	1/1 (100%)	18:22:00	2ms	9.17 KB	0 B	Show Subtasks ⌄

02

什么是state

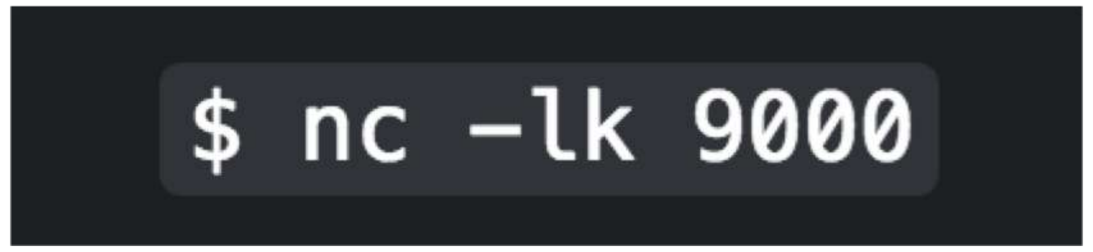
什么是state

Apache Flink



```
env.socketTextStream("localhost",9000)
    // split up the lines in pairs (2-tuples) containing: (word,1)
    .flatMap(new Tokenizer())
    // group by the tuple field "0" and sum up tuple field "1"
    .keyBy(0).sum(1)
    .print();
```

1. 执行上述代码
2. 本地启动netcat



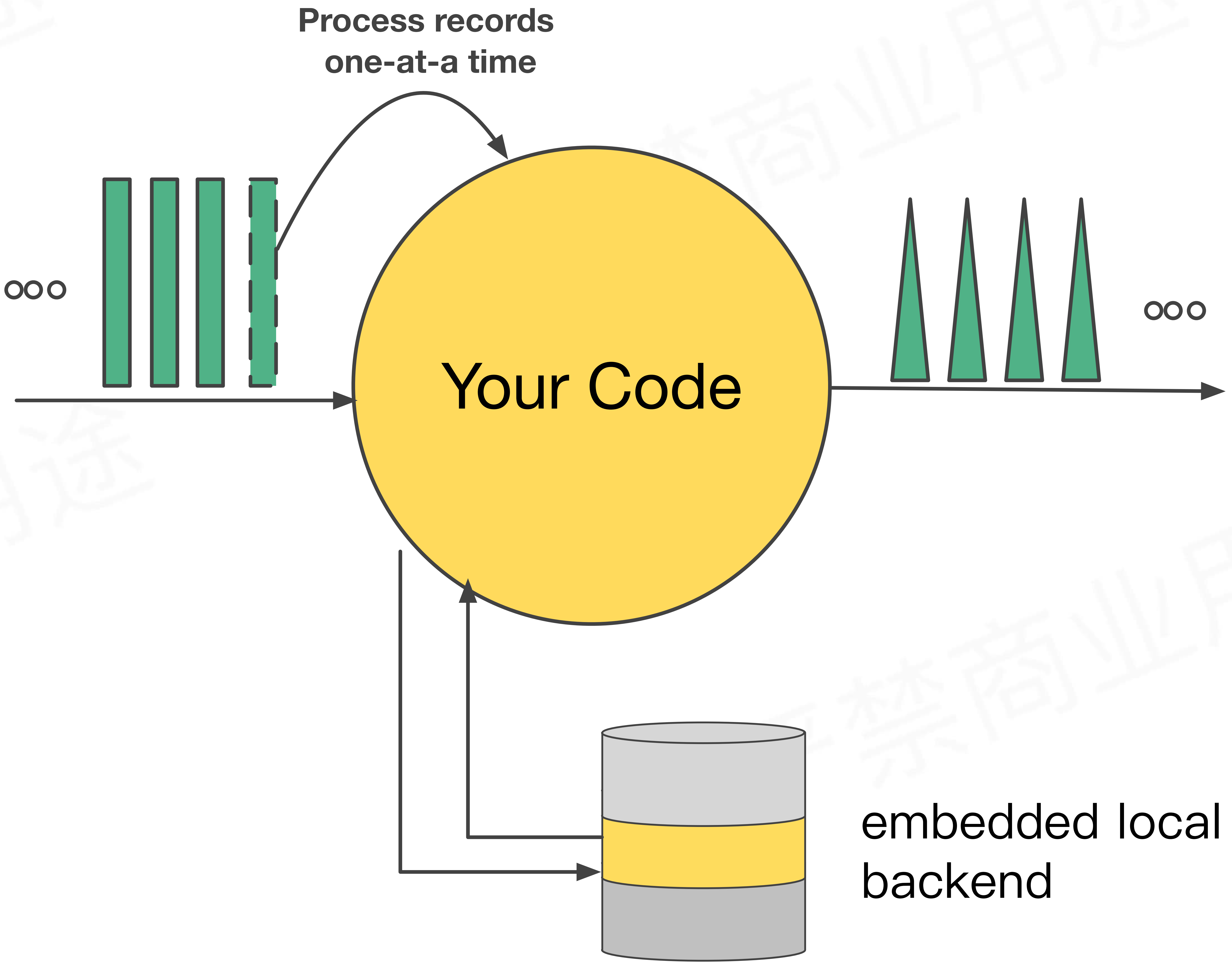
```
$ nc -lk 9000
```

3. 键入 `hello world`，执行程序会输出什么？

   再次键入 `hello world`，执行程序会输出什么？

什么是state

State:流式计算中持久化了的状态



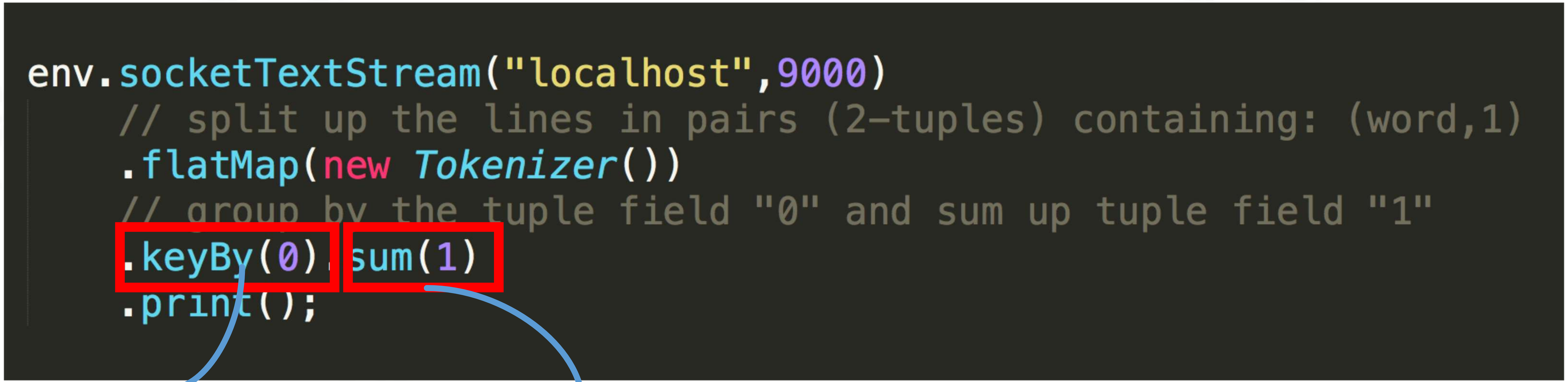
什么是state

Keyed State

- 只能应用于 KeyedStream 的函数与操作中，例如Keyed UDF, window state
- Keyed State是已经分区/划分好的，每一个key只能属于某一个keyed state

Keyed State

再来看这段 word count代码



```
env.socketTextStream("localhost",9000)
    // split up the lines in pairs (2-tuples) containing: (word,1)
    .flatMap(new Tokenizer())
    // group by the tuple field "0" and sum up tuple field "1"
    .keyBy(0).sum(1)
    .print();
```

创建 KeyedStream（对key进行了划分，不同task上不会出现相同的key）

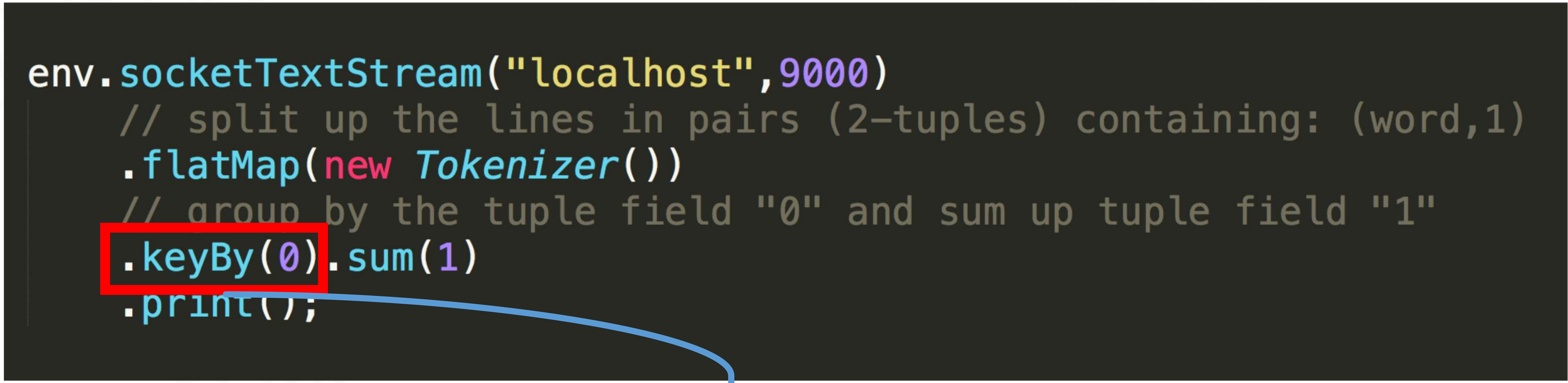
调用内置的 StreamGroupedReduce UDF

什么是state

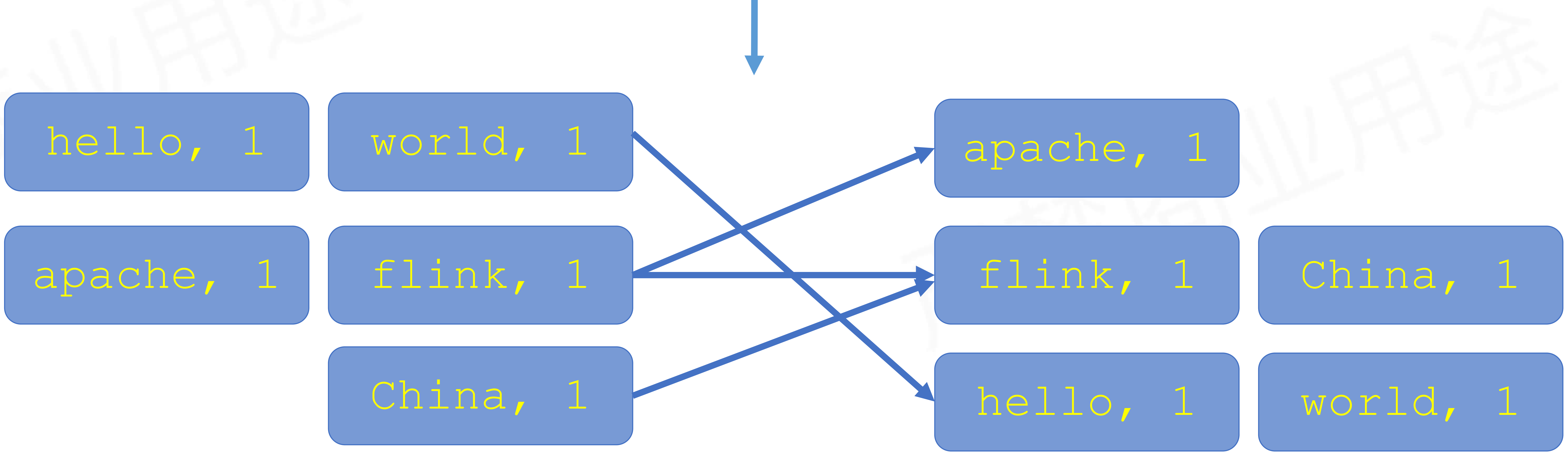
Keyed State

Apache Flink

再来看这段
word count代码



```
env.socketTextStream("localhost",9000)
    // split up the lines in pairs (2-tuples) containing: (word,1)
    .flatMap(new Tokenizer())
    // group by the tuple field "0" and sum up tuple field "1"
    .keyBy(0).sum(1)
    .print();
```



hello, 1

world, 1

apache, 1

flink, 1

China, 1

apache, 1

flink, 1

China, 1

hello, 1

world, 1

什么是state

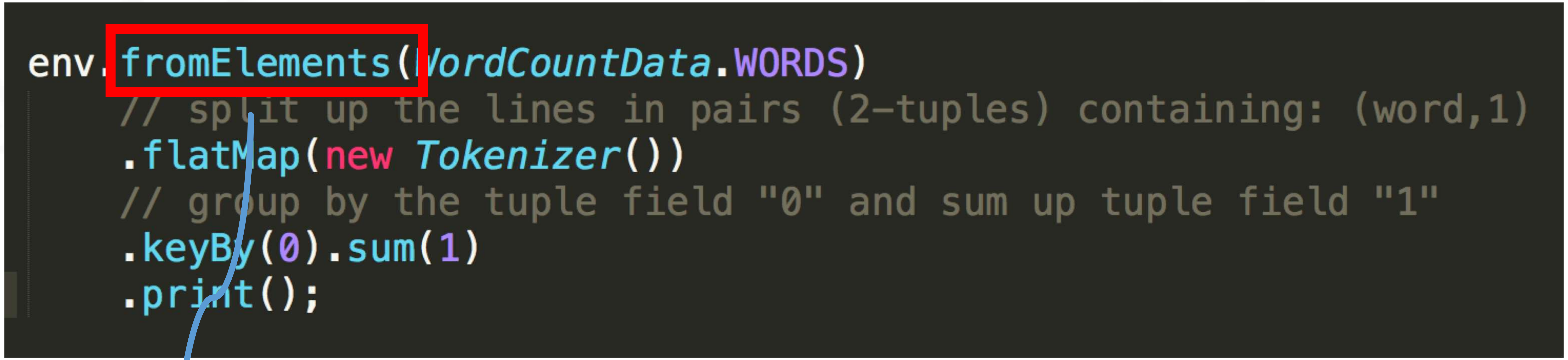
Apache Flink

Operator State

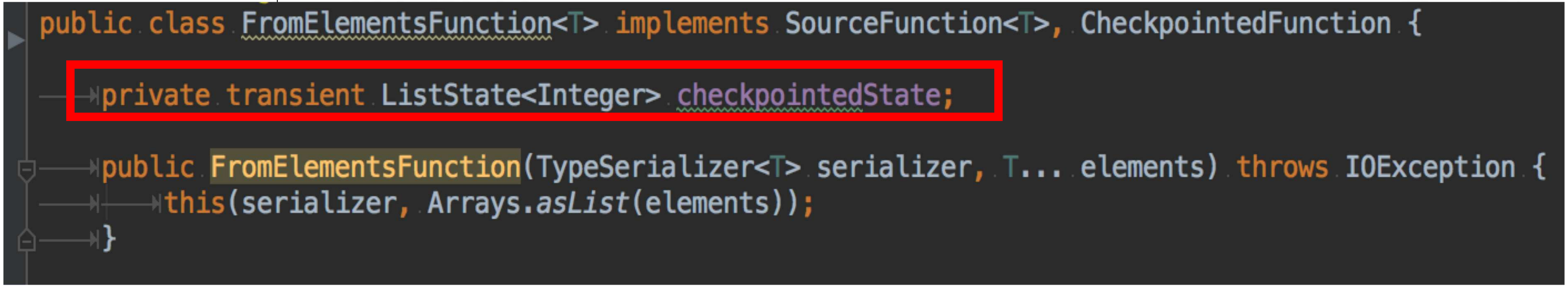
- 又称为non–keyed state，每一个operator state都仅与一个operator的实例绑定。
- 常见的operator state是source state，例如记录当前source的offset

什么是state Operator State

再来看一段
word count代码



```
env.fromElements(WordCountData.WORDS)
    // split up the lines in pairs (2-tuples) containing: (word,1)
    .flatMap(new Tokenizer())
    // group by the tuple field "0" and sum up tuple field "1"
    .keyBy(0).sum(1)
    .print();
```



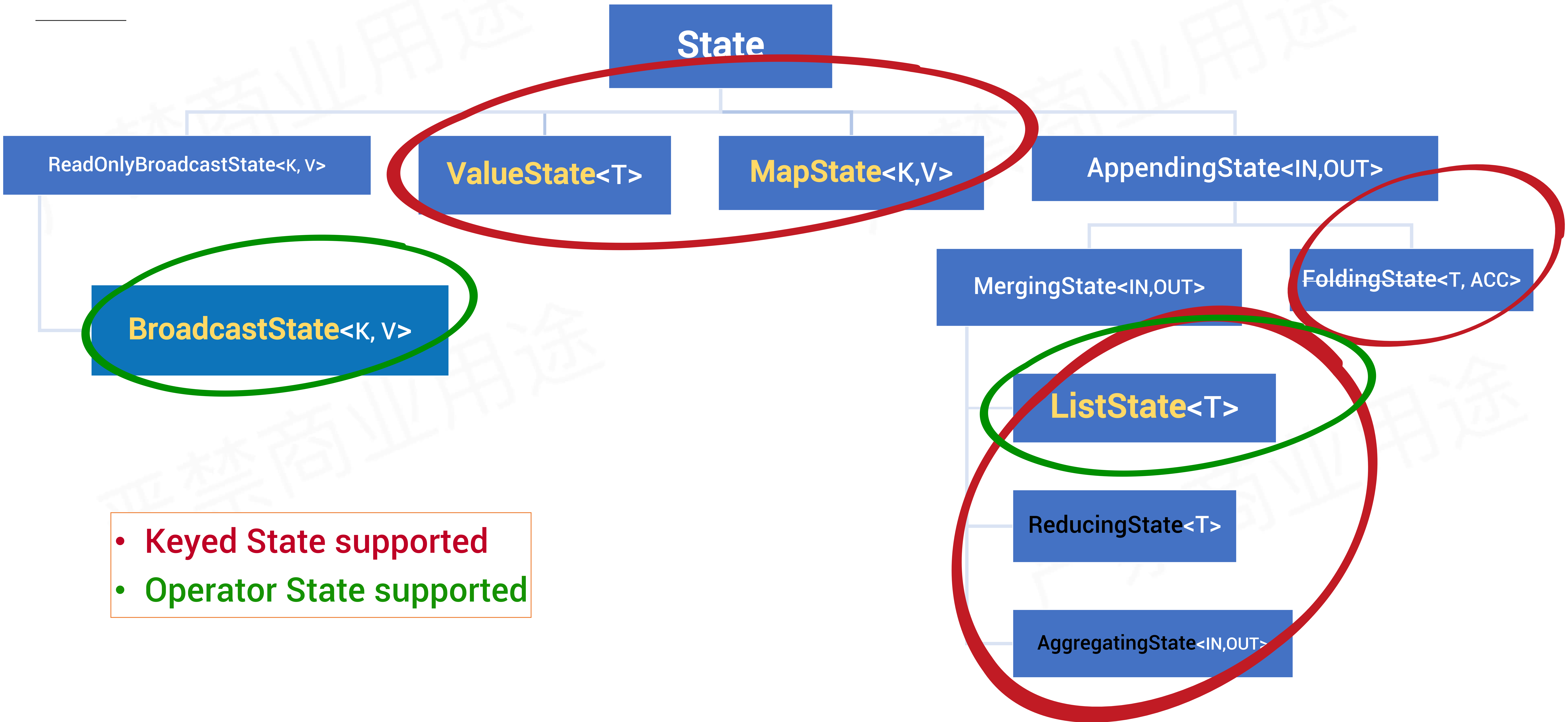
```
public class FromElementsFunction<T> implements SourceFunction<T>, CheckpointedFunction {

    private transient ListState<Integer> checkpointedState;

    public FromElementsFunction(TypeSerializer<T> serializer, T... elements) throws IOException {
        this(serializer, Arrays.asList(elements));
    }
```

什么是state

Apache Flink

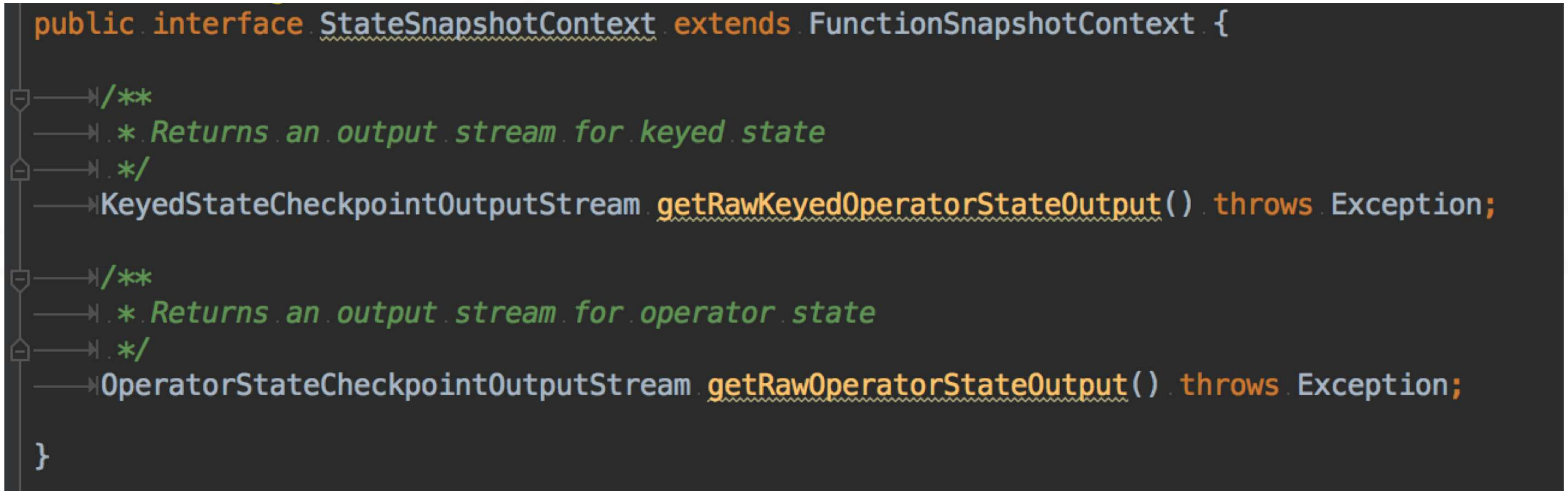


- Keyed State supported
- Operator State supported

什么是state

- Managed State：由Flink管理的state，刚才举例的所有state均是managed
- Raw State：Flink仅提供stream可以进行存储数据，对其而言

  只是一些bytes



```
public interface StateSnapshotContext extends FunctionSnapshotContext {

    /**
     * Returns an output stream for keyed state
     */
    KeyedStateCheckpointOutputStream getRawKeyedOperatorStateOutput() throws Exception;

    /**
     * Returns an output stream for operator state
     */
    OperatorStateCheckpointOutputStream getRawOperatorStateOutput() throws Exception;

}
```

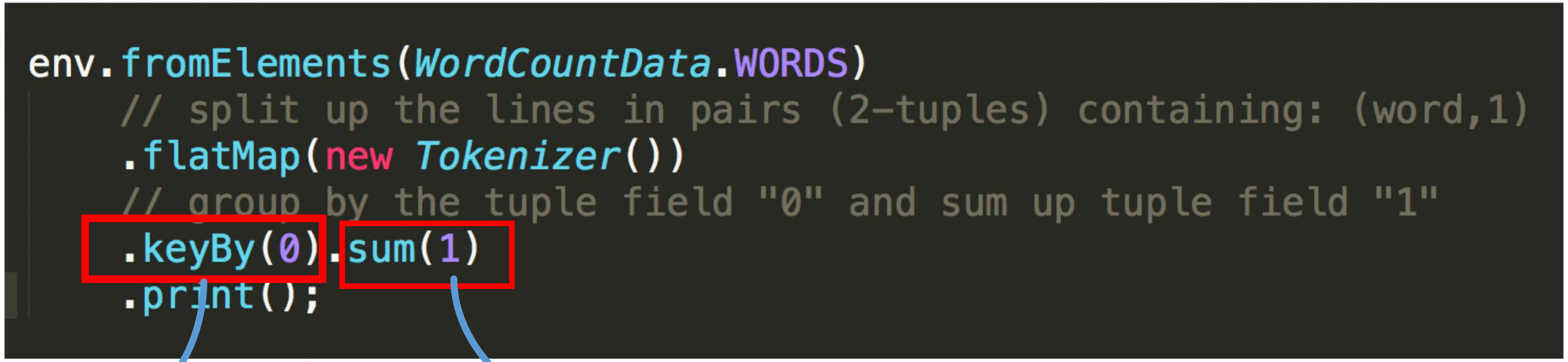
03

如何在Flink中使用state

使用指南

如何在Flink中使用state：Keyed State

再来看这段
word count代码

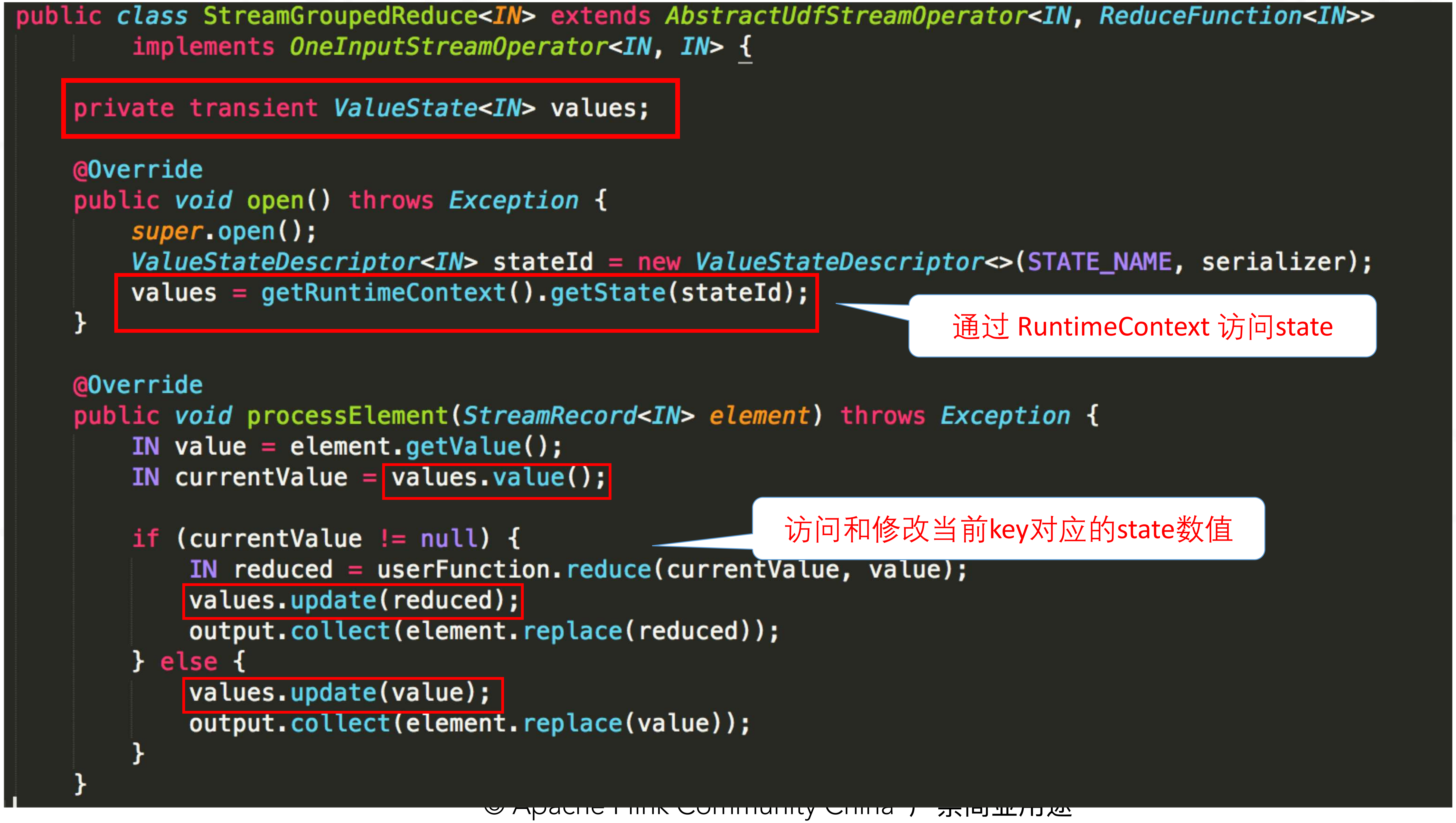


```
env.fromElements(WordCountData.WORDS)
    // split up the lines in pairs (2-tuples) containing: (word,1)
    .flatMap(new Tokenizer())
    // group by the tuple field "0" and sum up tuple field "1"
    .keyBy(0).sum(1)
    .print();
```

创建KeyedStream
（对key进行了划分，
不同task上不会出现相
同的key）

调用内置的StreamGroupedReduce

如何在Flink中使用state：Keyed State



```java
public class StreamGroupedReduce<IN> extends AbstractUdfStreamOperator<IN, ReduceFunction<IN>>
        implements OneInputStreamOperator<IN, IN> {

    private transient ValueState<IN> values;

    @Override
    public void open() throws Exception {
        super.open();
        ValueStateDescriptor<IN> stateId = new ValueStateDescriptor<>(STATE_NAME, serializer);
        values = getRuntimeContext().getState(stateId);
    }

    @Override
    public void processElement(StreamRecord<IN> element) throws Exception {
        IN value = element.getValue();
        IN currentValue = values.value();

        if (currentValue != null) {
            IN reduced = userFunction.reduce(currentValue, value);
            values.update(reduced);
            output.collect(element.replace(reduced));
        } else {
            values.update(value);
            output.collect(element.replace(value));
        }
    }
```

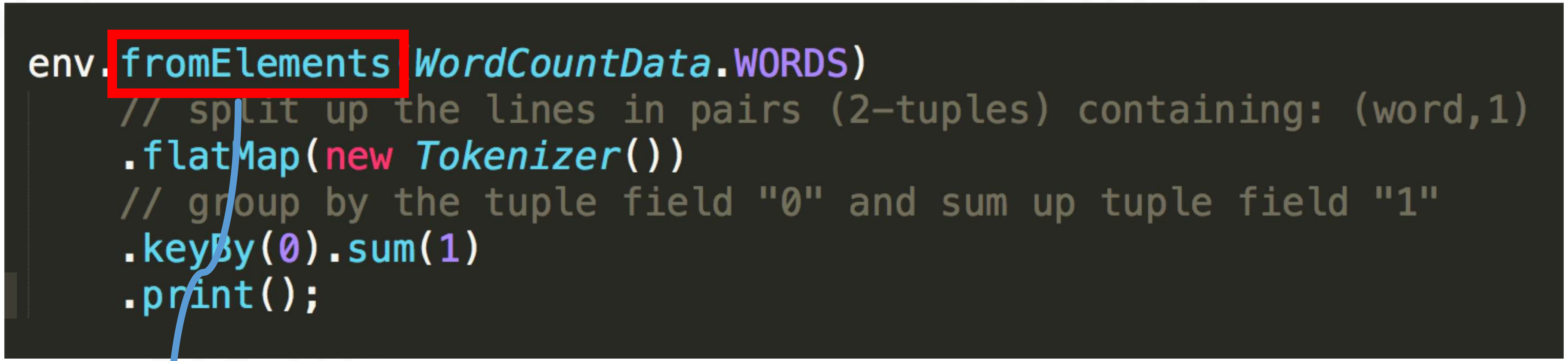
通过 RuntimeContext 访问state

访问和修改当前key对应的state数值

如何在Flink中使用state：Operator State

Apache Flink

再来看这段
word count代码

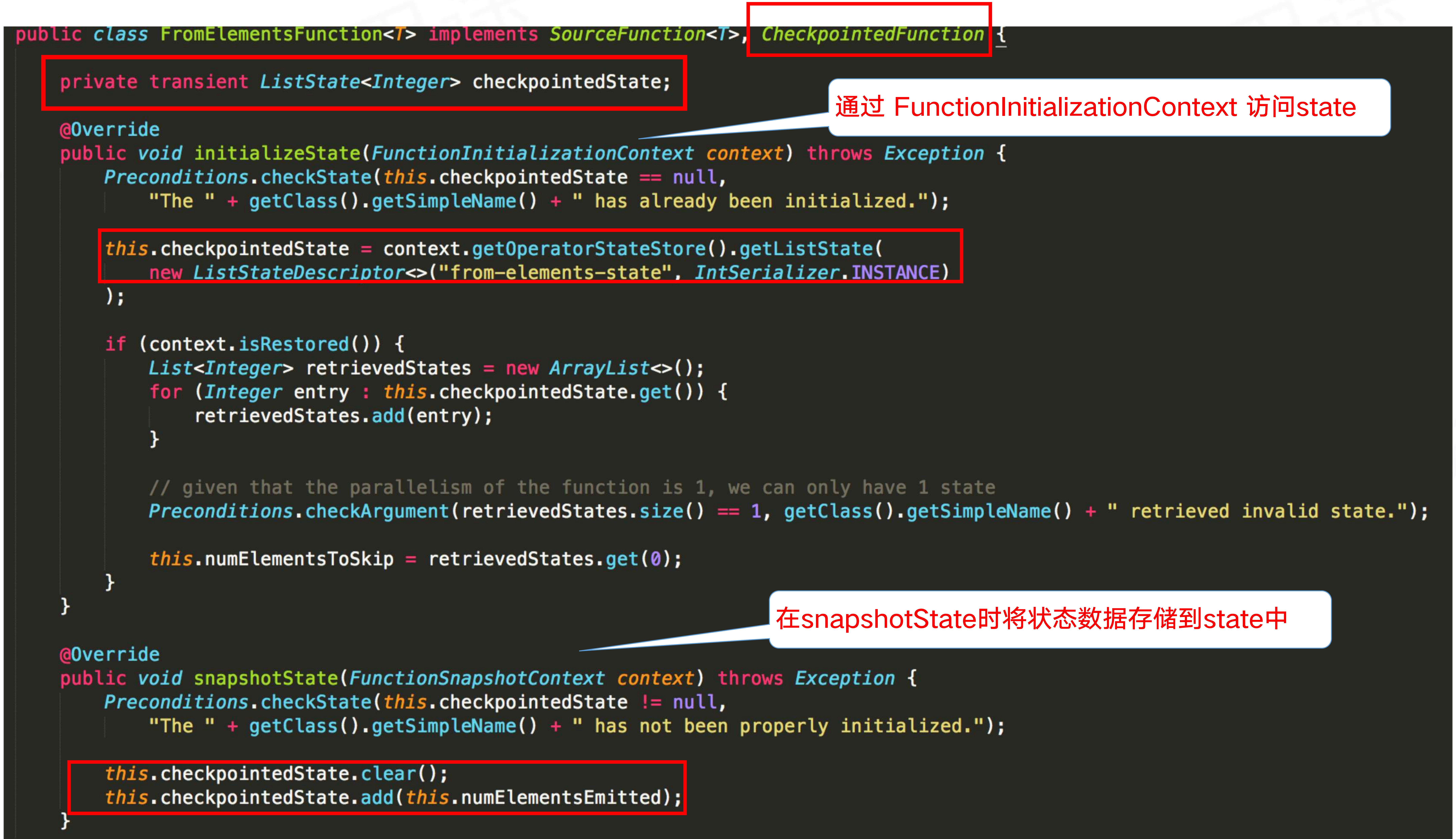


```
env.fromElements(WordCountData.WORDS)
    // split up the lines in pairs (2-tuples) containing: (word,1)
    .flatMap(new Tokenizer())
    // group by the tuple field "0" and sum up tuple field "1"
    .keyBy(0).sum(1)
    .print();
```

调用内置的FromElementsFunction

如何在Flink中使用state：Operator State

Apache Flink



```
public class FromElementsFunction<T> implements SourceFunction<T>, CheckpointedFunction {

    private transient ListState<Integer> checkpointedState;

    @Override
    public void initializeState(FunctionInitializationContext context) throws Exception {
        Preconditions.checkState(this.checkpointedState == null,
            "The " + getClass().getSimpleName() + " has already been initialized.");

        this.checkpointedState = context.getOperatorStateStore().getListState(
            new ListStateDescriptor<>("from-elements-state", IntSerializer.INSTANCE)
        );

        if (context.isRestored()) {
            List<Integer> retrievedStates = new ArrayList<>();
            for (Integer entry : this.checkpointedState.get()) {
                retrievedStates.add(entry);
            }

            // given that the parallelism of the function is 1, we can only have 1 state
            Preconditions.checkArgument(retrievedStates.size() == 1, getClass().getSimpleName() + " retrieved invalid state.");

            this.numElementsToSkip = retrievedStates.get(0);
        }
    }

    @Override
    public void snapshotState(FunctionSnapshotContext context) throws Exception {
        Preconditions.checkState(this.checkpointedState != null,
            "The " + getClass().getSimpleName() + " has not been properly initialized.");

        this.checkpointedState.clear();
        this.checkpointedState.add(this.numElementsEmitted);
    }
```

通过 FunctionInitializationContext 访问state

在snapshotState时将状态数据存储到state中

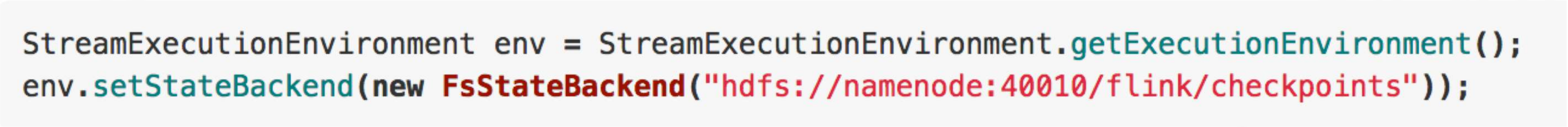
04

Checkpoint的执行机制

Checkpoint internal

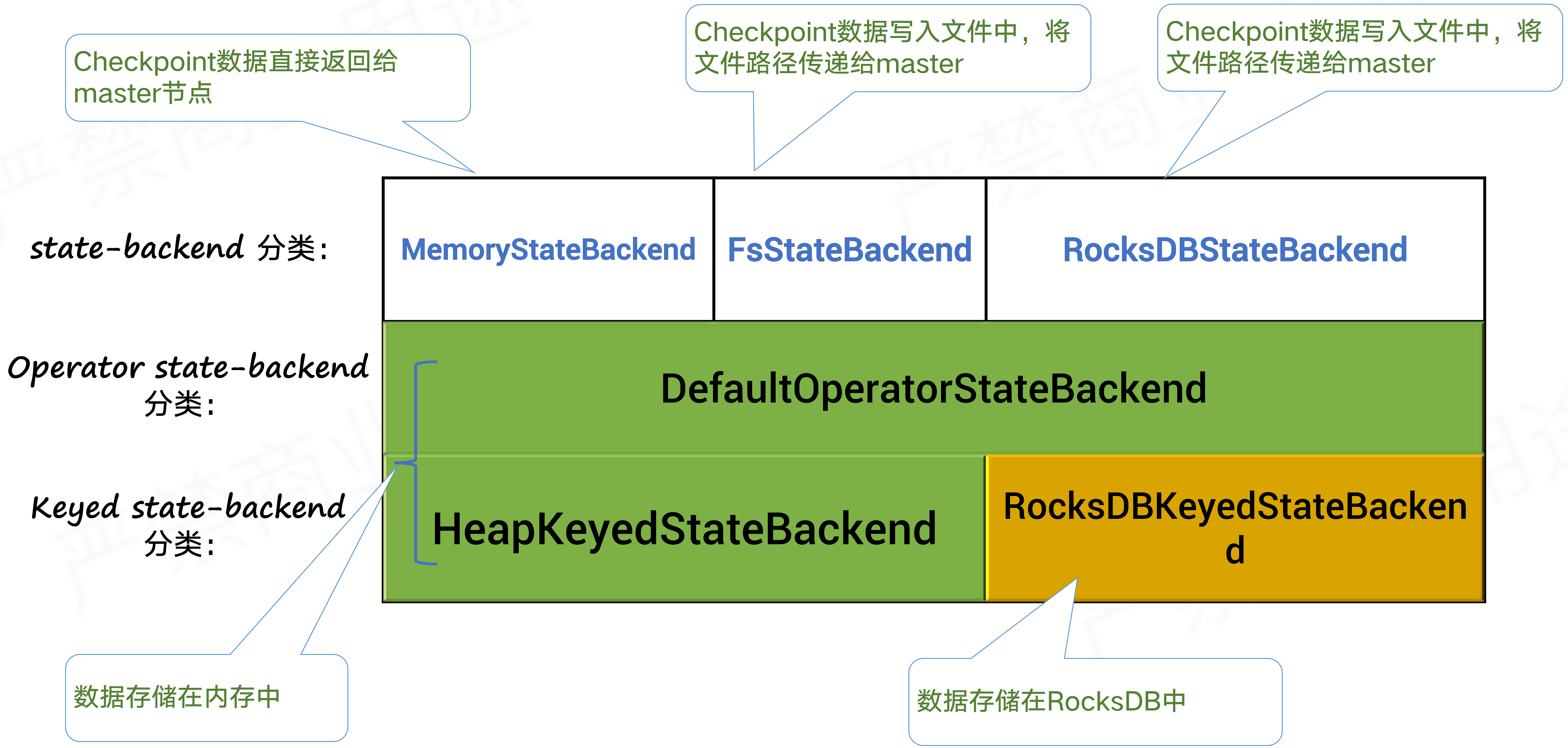
State的存储

<table>
<tr><td><em>state-backend</em> 分类：</td><td>MemoryStateBackend</td><td>FsStateBackend</td><td>RocksDBStateBackend</td></tr>
<tr><td><em>Operator state-backend</em> 分类：</td><td colspan="3">DefaultOperatorStateBackend</td></tr>
<tr><td><em>Keyed state-backend</em> 分类：</td><td colspan="2">HeapKeyedStateBackend</td><td>RocksDBKeyedStateBackend</td></tr>
</table>



```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.setStateBackend(new FsStateBackend("hdfs://namenode:40010/flink/checkpoints"));
```

State的存储



Checkpoint数据直接返回给master节点

Checkpoint数据写入文件中，将文件路径传递给master

Checkpoint数据写入文件中，将文件路径传递给master

state-backend 分类：

MemoryStateBackend

FsStateBackend

RocksDBStateBackend

Operator state-backend 分类：

DefaultOperatorStateBackend

Keyed state-backend 分类：

HeapKeyedStateBackend

RocksDBKeyedStateBackend

数据存储在内存中

数据存储在RocksDB中

State的存储

HeapKeyedStateBackend 存储格式

- 支持异步checkpoint（默认）：CopyOnWriteStateTable<K, N, S>[ ]， 整体相当于一个map
- 仅支持同步checkpoint：Map<N, Map<K, S>> [ ] ， 由嵌套map的数组构成
- 在MemoryStateBackend内使用时，checkpoint序列化数据阶段默认有最大5MB数据的限制

State的存储

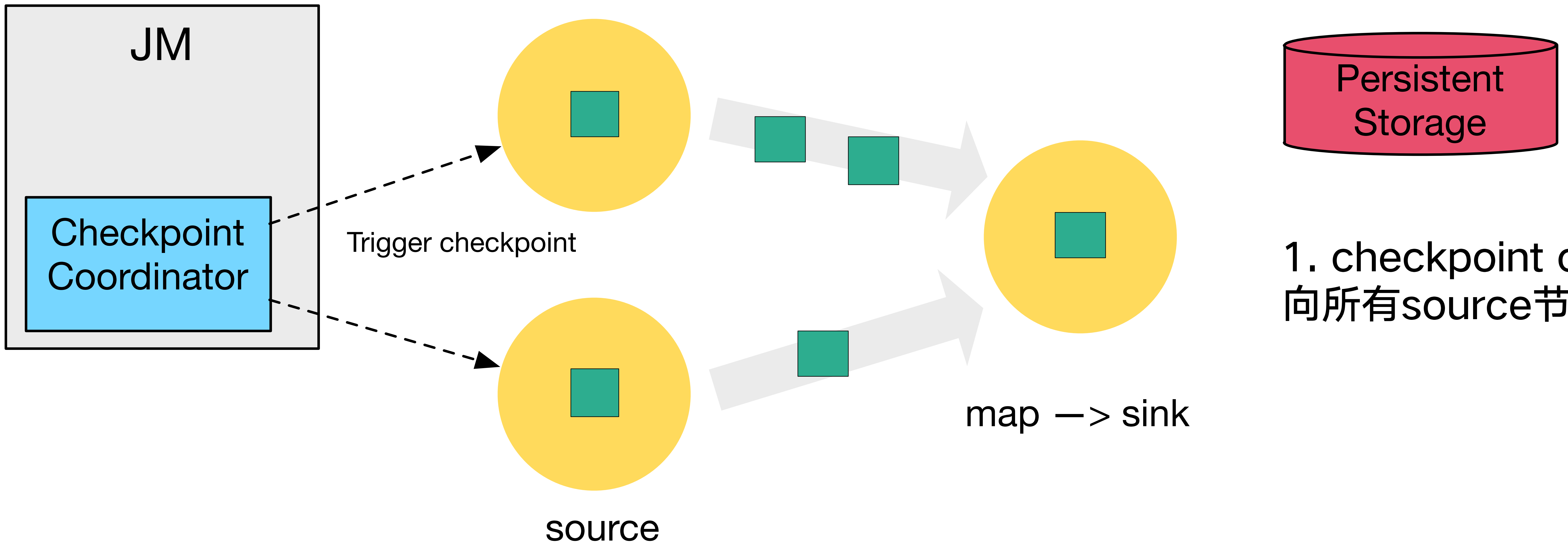
RocksDBKeyedStateBackend 存储格式

每个state都存储在一个单独的column family内

keyGroup，Key和Namespace进行序列化存储在DB作为key

State1		State2	
KeyGroup + Key + Namespace	value	KeyGroup + Key + Namespace	value
(1, K1, Window(10, 20))	v1	(2, K2, Window(10, 20))	v2
(1, K3, Window(10, 20))	v3	(2, K4, Window(10, 25))	v4
...	...	...	...

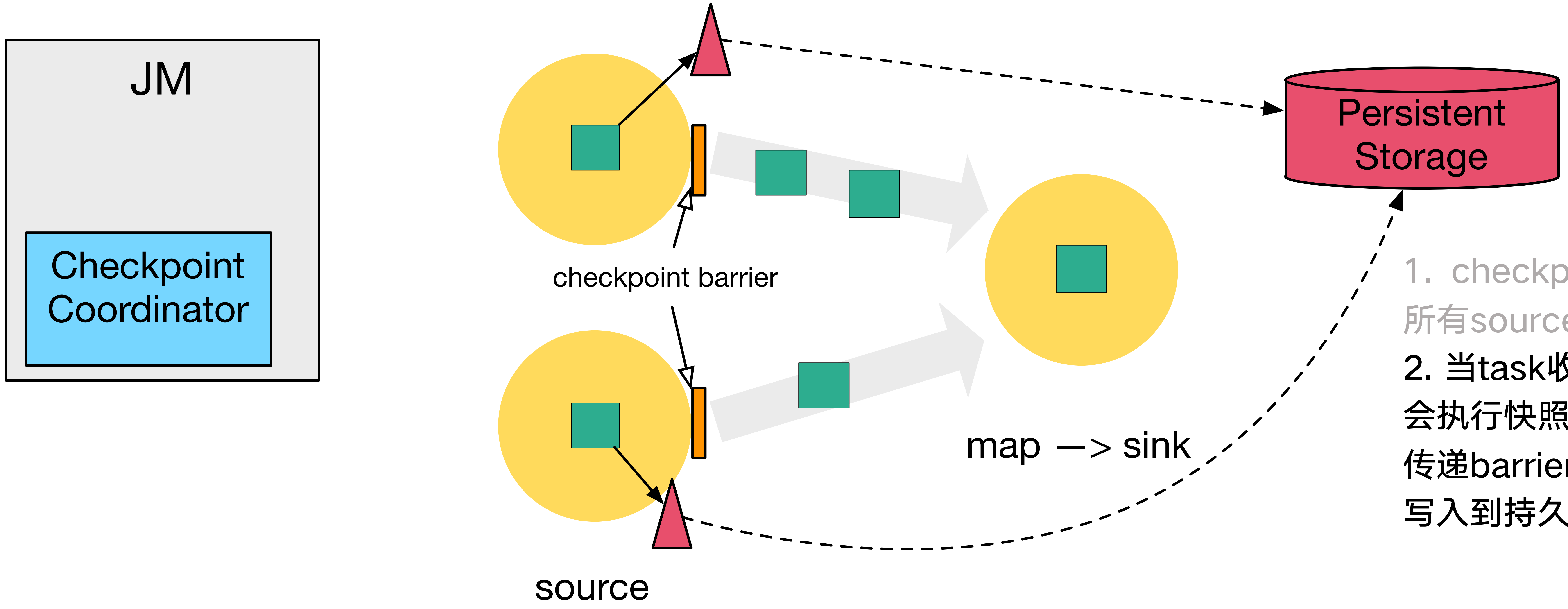
Checkpoint执行流程



1. checkpoint coordinator 向所有source节点trigger checkpoint

Checkpoint执行流程

Apache Flink

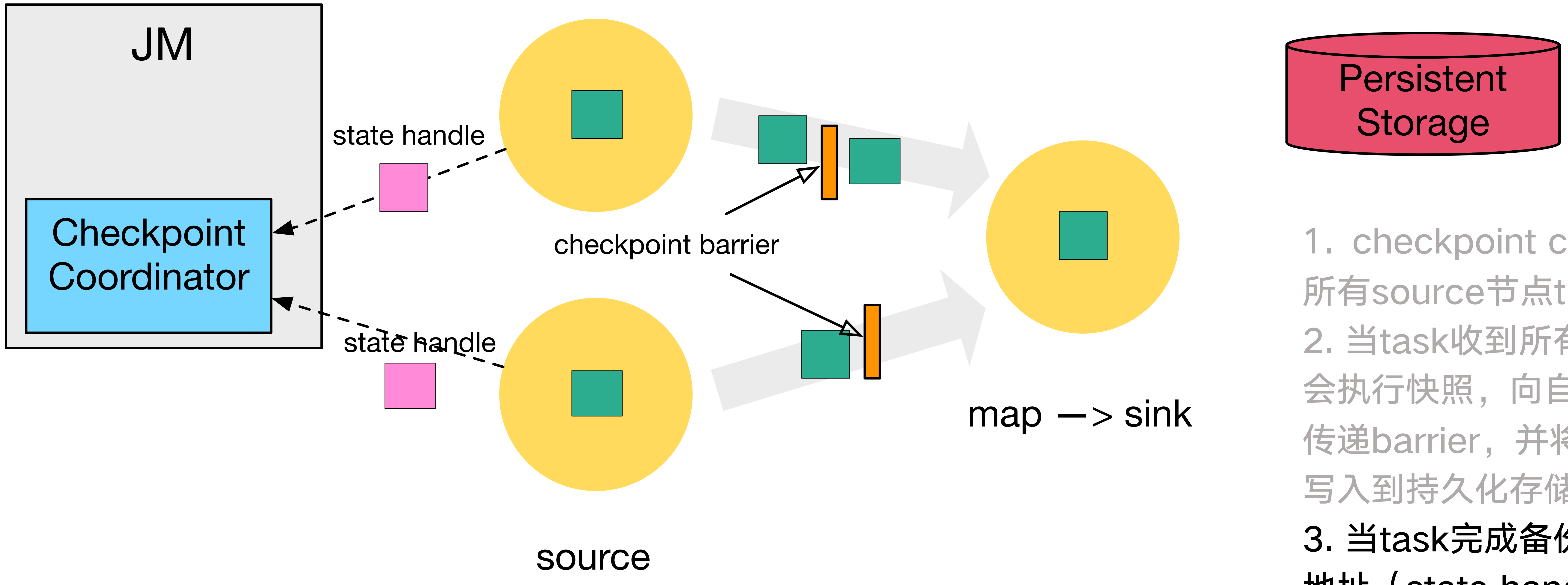


1. checkpoint coordinator向所有source节点trigger checkpoint

2. 当task收到所有barrier之后，会执行快照，向自己的输出继续传递barrier，并将自己的状态(异步)写入到持久化存储中

Checkpoint执行流程

Apache Flink

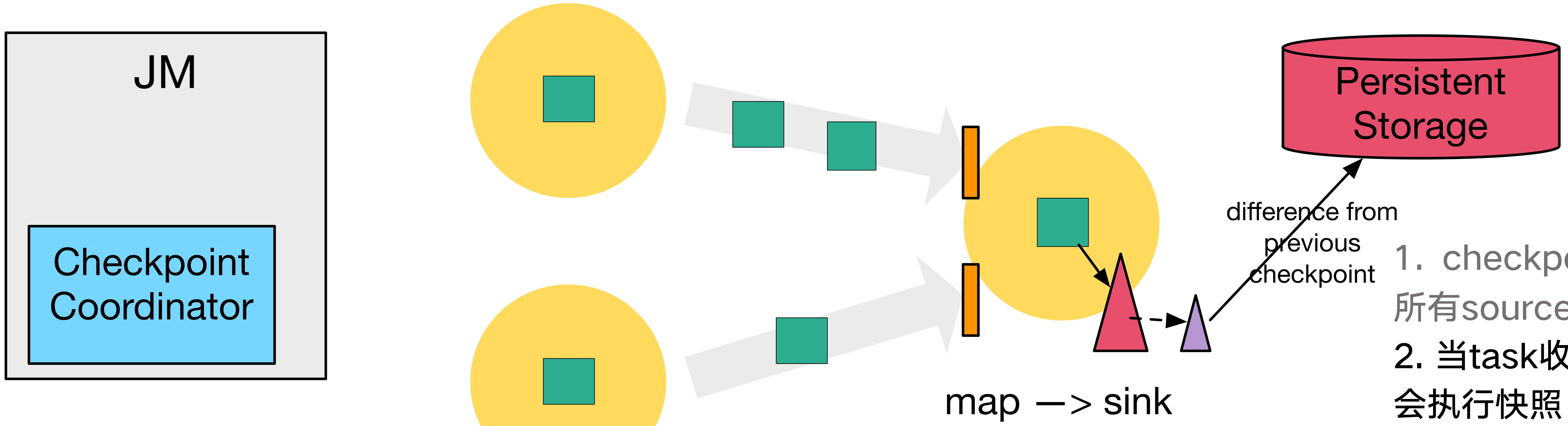


1. checkpoint coordinator向所有source节点trigger checkpoint

2. 当task收到所有barrier之后，会执行快照，向自己的输出继续传递barrier，并将自己的状态(异步)写入到持久化存储中

3. 当task完成备份后，会将备份数据的地址（state handle）通知给checkpoint coordinator

Checkpoint执行流程

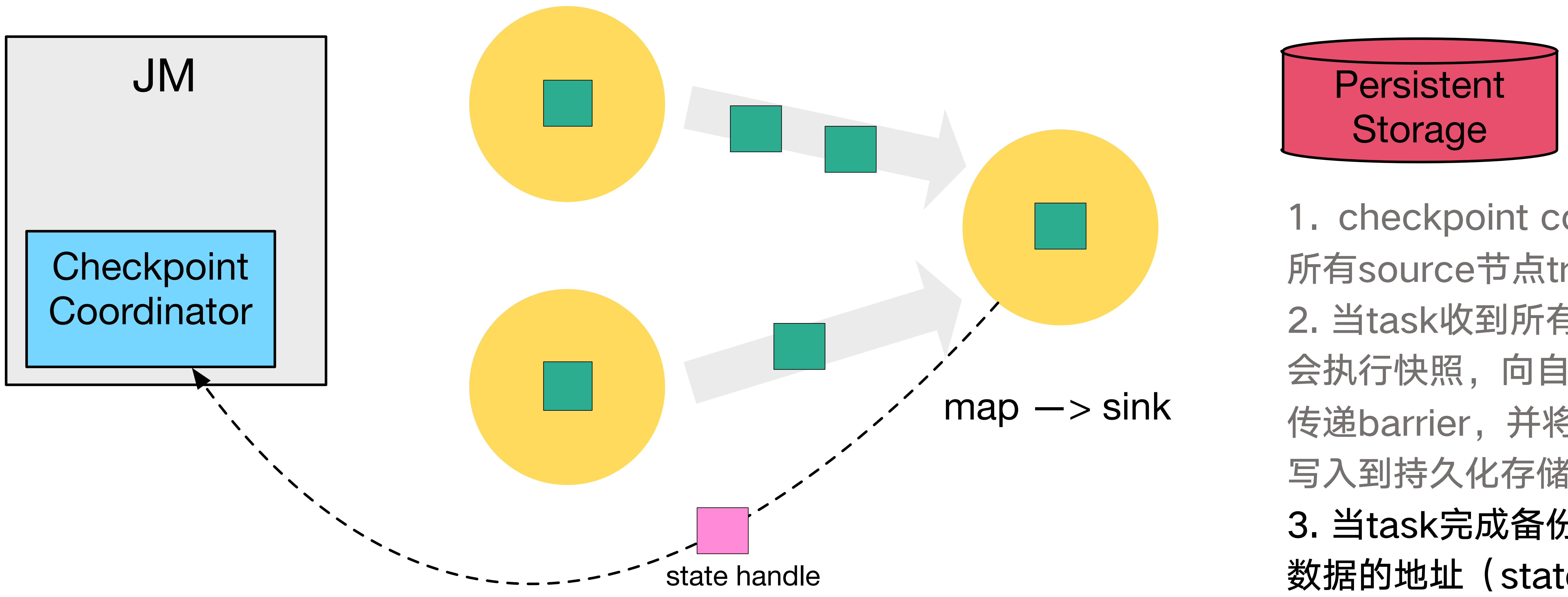


1．checkpoint coordinator向所有source节点trigger checkpoint

2. 当task收到所有barrier之后，会执行快照，向自己的输出继续传递barrier，并将自己的状态(异步)写入到持久化存储中

3. 当task完成备份后，会将备份数据的地址（state handle）通知给checkpoint coordinator

Checkpoint执行流程



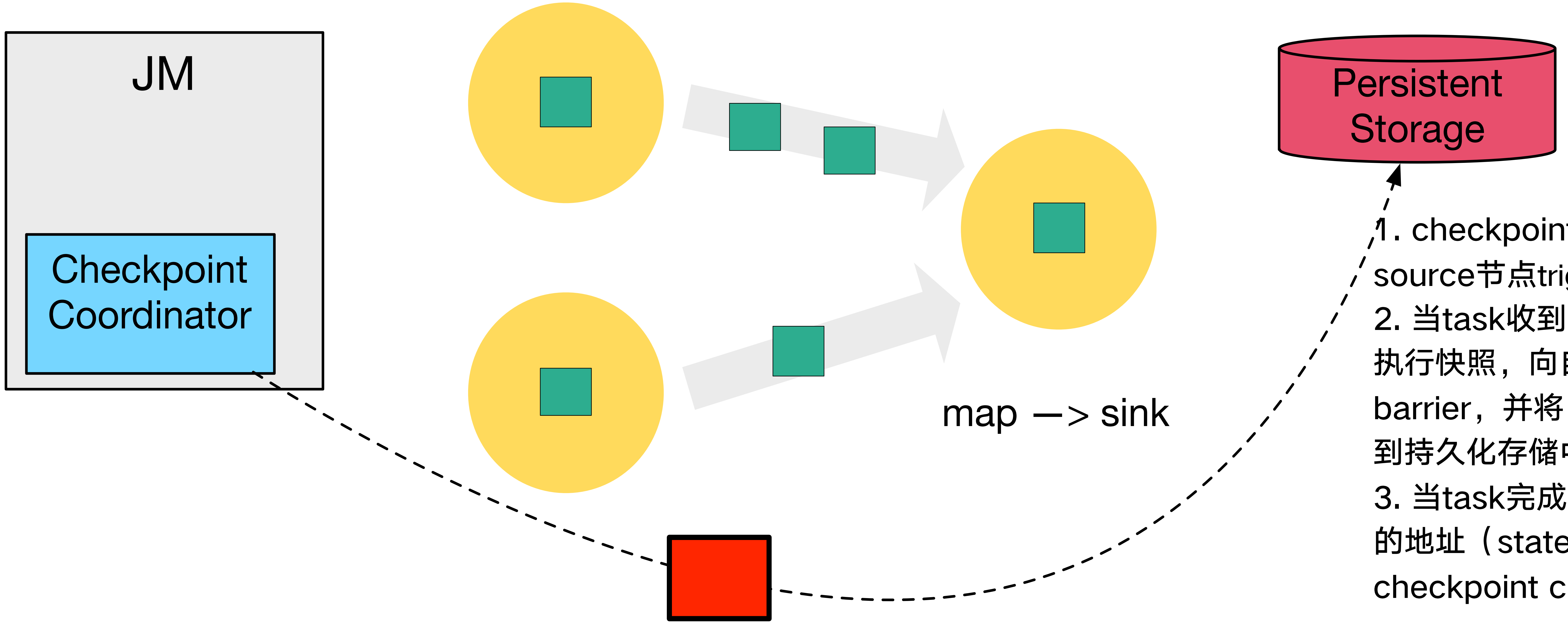
1. checkpoint coordinator向所有source节点trigger checkpoint

2. 当task收到所有barrier之后，会执行快照，向自己的输出继续传递barrier，并将自己的状态(异步)写入到持久化存储中

3. 当task完成备份后，会将备份数据的地址（state handle）通知给checkpoint coordinator

Checkpoint执行流程

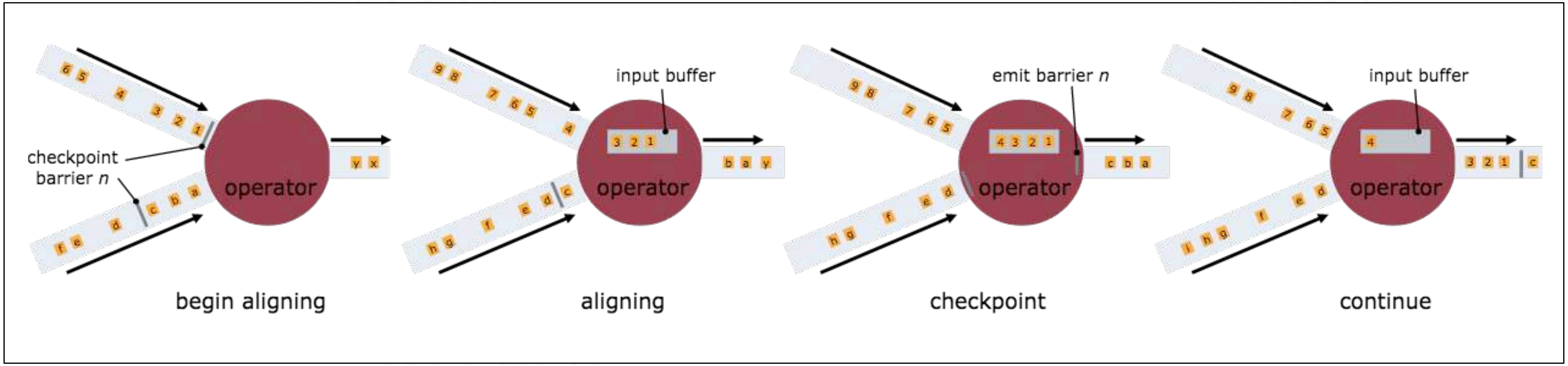
Apache Flink



completed checkpoint meta

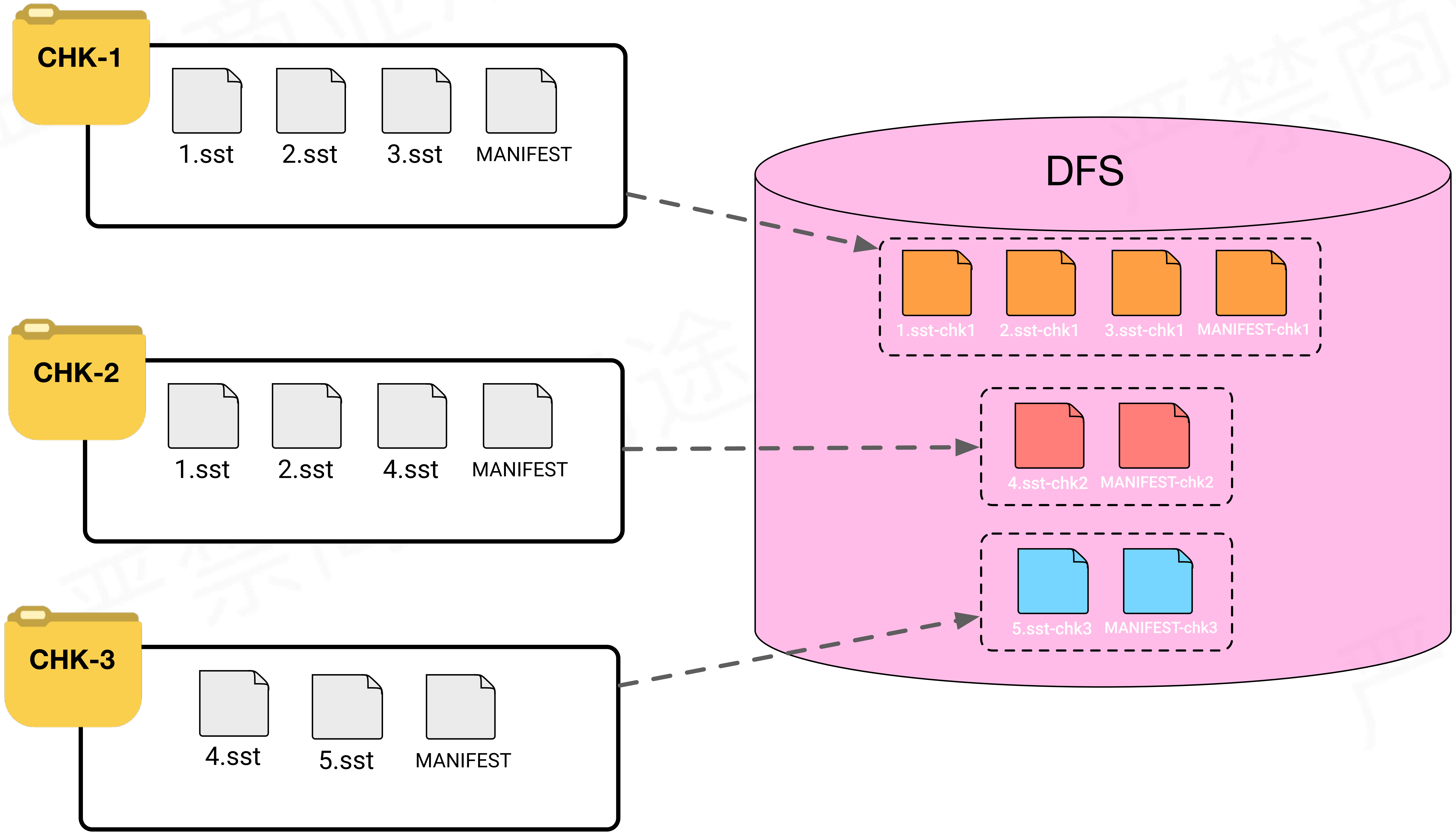
1. checkpoint coordinator向所有source节点trigger checkpoint
2. 当task收到所有barrier之后，会执行快照，向自己的输出继续传递barrier，并将自己的状态(异步)写入到持久化存储中
3. 当task完成备份后，会将备份数据的地址（state handle）通知给checkpoint coordinator

Checkpoint执行流程



- 为了实现EXACTLY ONCE语义，Flink通过一个input buffer将在对齐阶段收到的数据缓存起来，等对齐完成之后再进行处理。
- 对于AT LEAST ONCE语义，无需缓存收集到的数据，会对后续直接处理，所以导致restore时，数据可能会被多次处理。
- Flink的checkpoint机制只能保证Flink的计算过程可以做到EXACTLY ONCE，end-to-end的EXACTLY ONCE需要source和sink支持

基于RocksDB的增量checkpoint



1. 本地snapshot目录创建当前DB内容的备份
2. 与上一次成功的checkpoint本地sst文件列表比对，将不在其中的文件上传到DFS中
3. 所有文件都会重命名防止冲突
4. 包含了所有新旧文件的handle返回给checkpoint coordinator

从已停止的作业进行状态恢复

Savepoint

用户通过命令触发，由用户管理其创建与删除

标准化格式存储，允许作业升级或者配置变更

用户在恢复时需要提供用于恢复作业状态的savepoint路径

Externalized Checkpoint

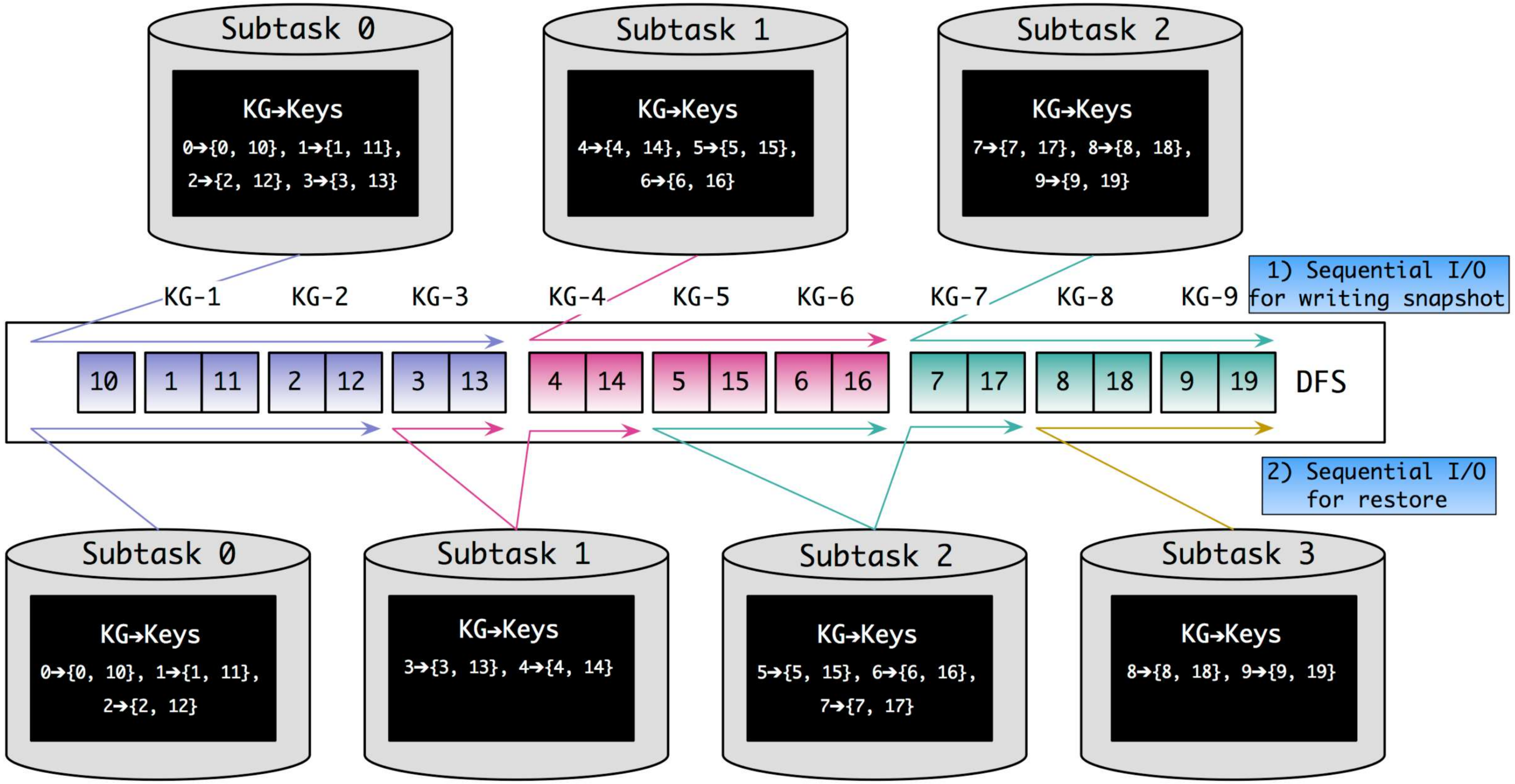
Checkpoint完成时，在用户给定的外部持久化存储保存

当作业FAILED（或者CANCELED）时，外部存储的checkpoint会保留下来

用户在恢复时需要提供用于恢复的作业状态的checkpoint路径

从已停止的作业进行状态恢复 Keyed State的改并发

Apache Flink



Subtask 0

KG→Keys
0→{0, 10}, 1→{1, 11}, 2→{2, 12}, 3→{3, 13}

Subtask 1

KG→Keys
4→{4, 14}, 5→{5, 15}, 6→{6, 16}

Subtask 2

KG→Keys
7→{7, 17}, 8→{8, 18}, 9→{9, 19}

1) Sequential I/O for writing snapshot

2) Sequential I/O for restore

Subtask 0

KG→Keys
0→{0, 10}, 1→{1, 11}, 2→{2, 12}

Subtask 1

KG→Keys
3→{3, 13}, 4→{4, 14}

Subtask 2

KG→Keys
5→{5, 15}, 6→{6, 16}, 7→{7, 17}

Subtask 3

KG→Keys
8→{8, 18}, 9→{9, 19}

```
key ∈ [0, 19], number_of_key_groups = 10
1. hash(key) = key (identity)
2. key_group(key) = hash(key) % number_of_key_groups
3. subtask(key) = key_group(key) * parallelism / number_of_key_groups
```

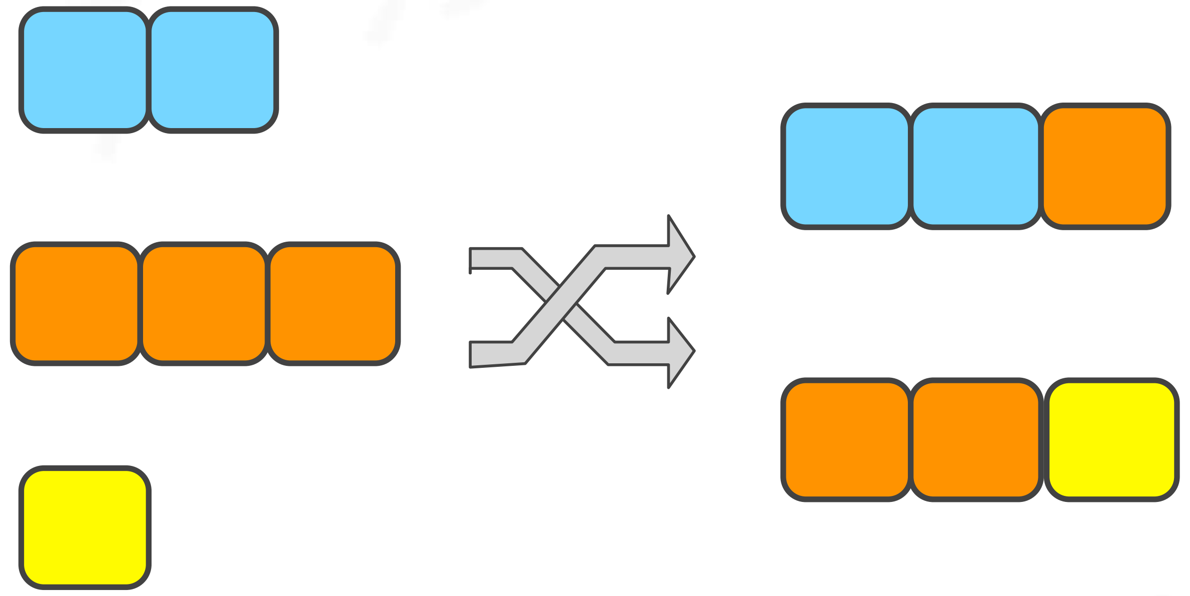
并发度 3 → 4

利用KeyGroup改并发

从已停止的作业进行状态恢复 Operator State的改并发

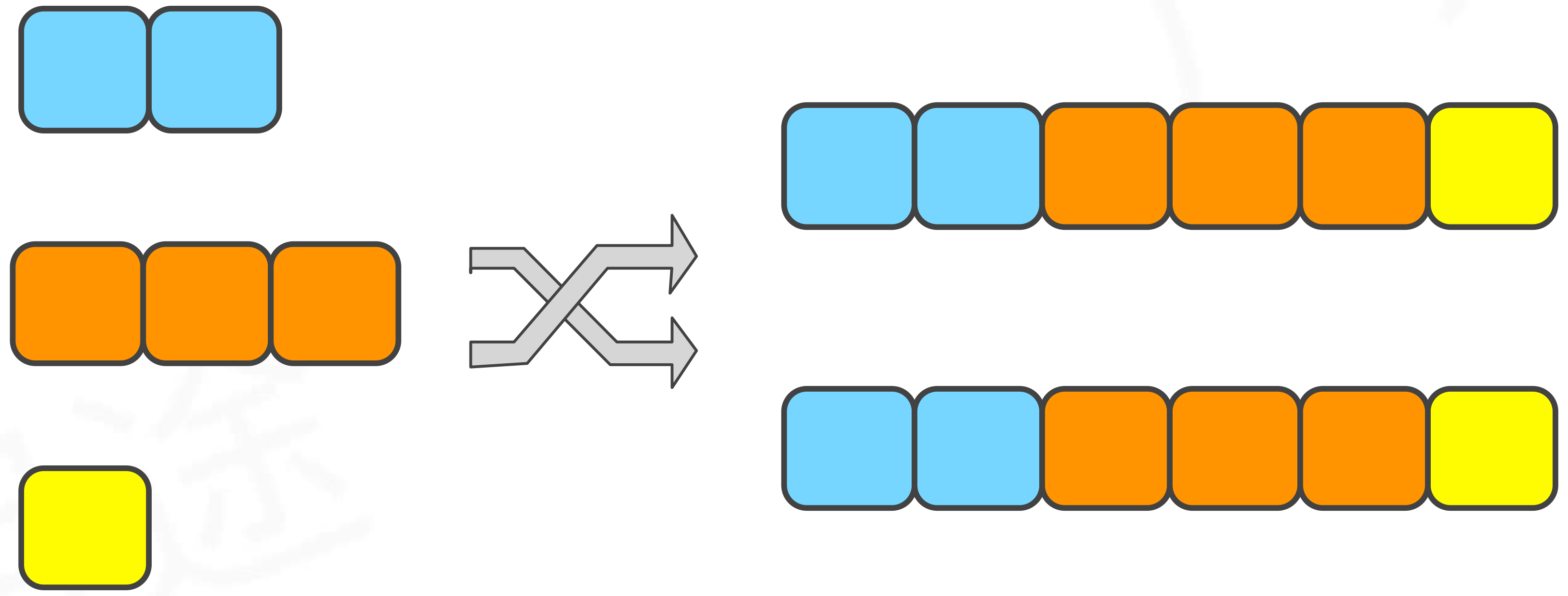
Apache Flink

ListState



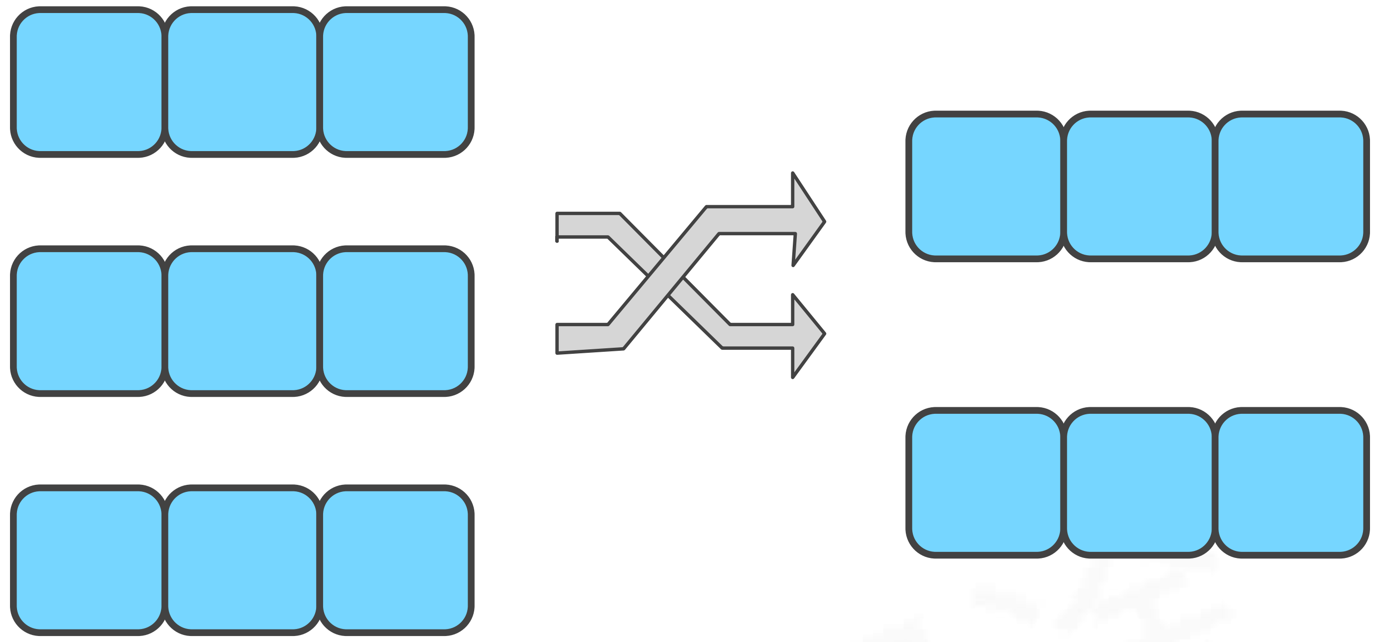
所有task的元素均匀划分给新的task
(Even-split redistribution)

UnionListState



所有task的元素全部划分给新的task
(Union redistribution)

BroadcastState <K, V>



所有task的state相同，改并发时，新的task获得state的一个备份

THANKS

Flink China社区大群

扫一扫群二维码，立刻加入该群。