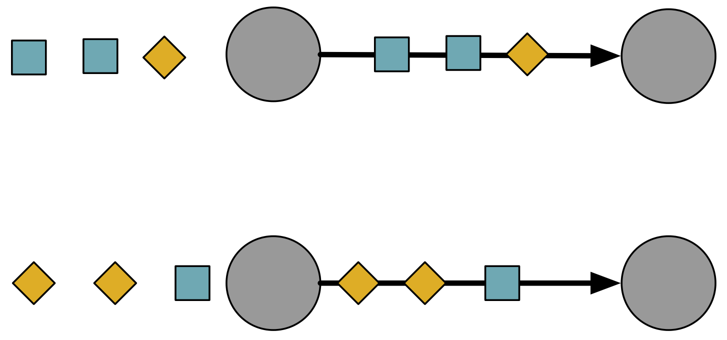
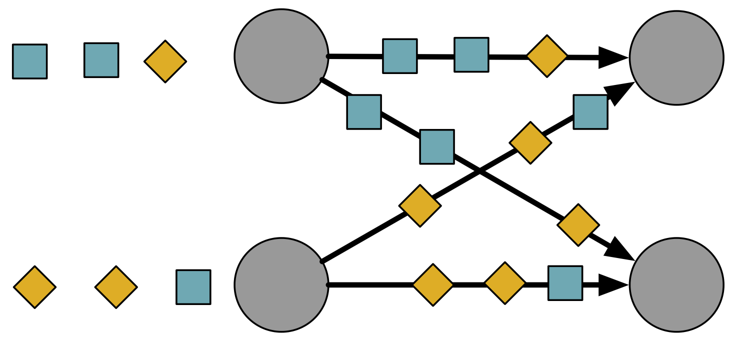
Object Reuse & Serialization (2)

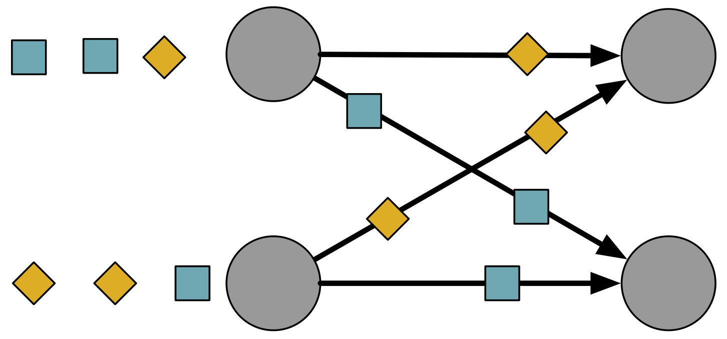
Apache Flink® Tuning & Troubleshooting Training

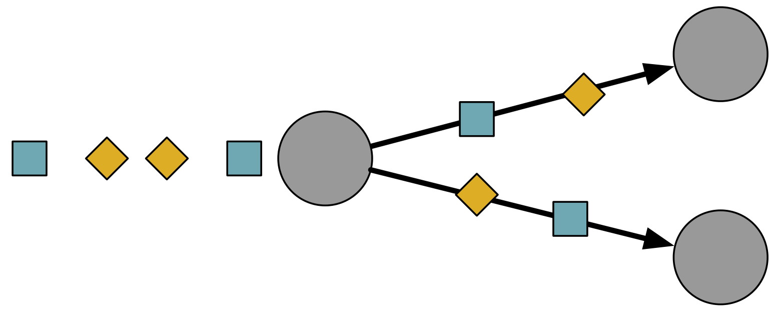
ververicα

Recall: Data exchange strategies


forward

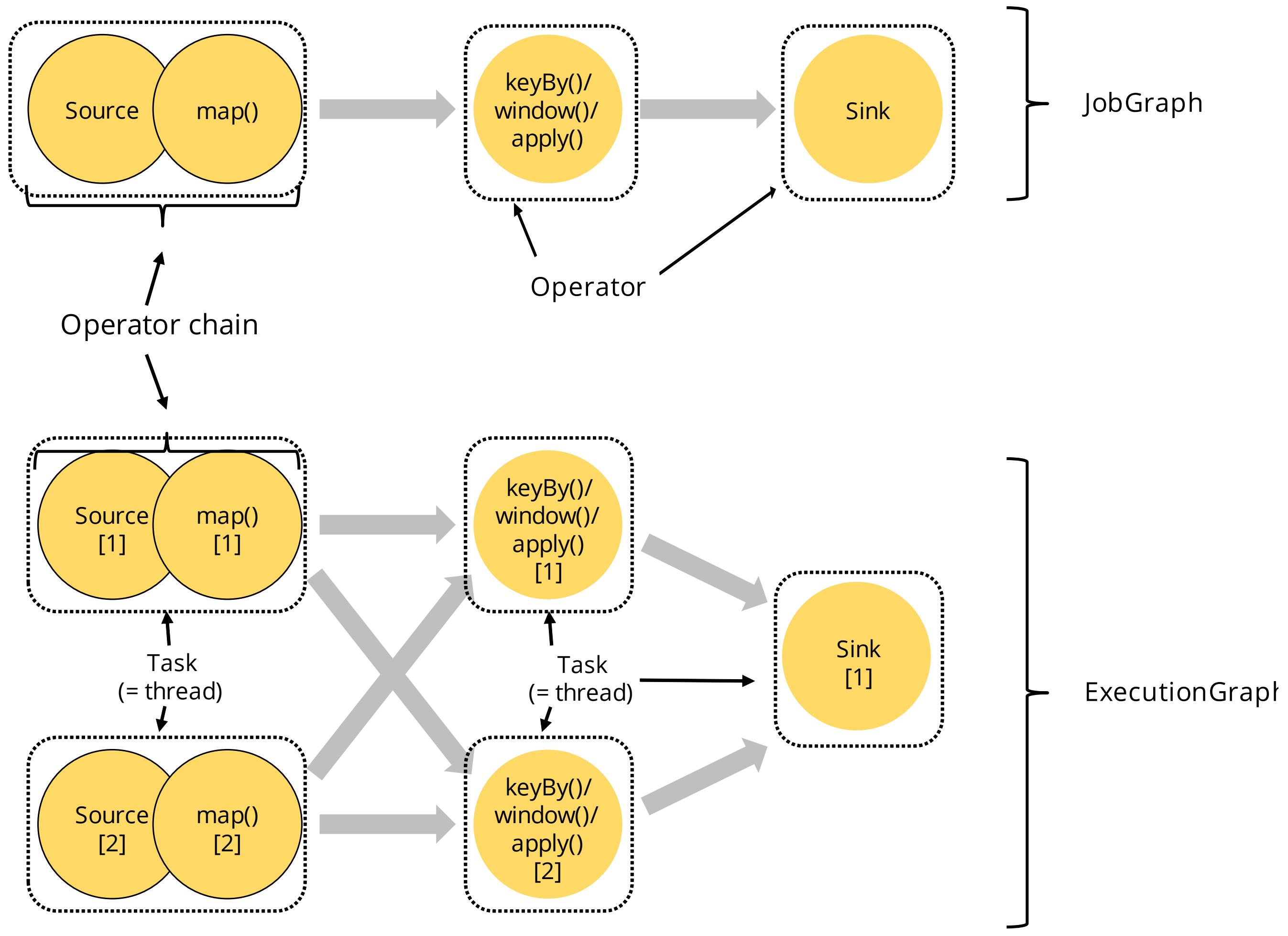

broadcast


key-based


random

Recall: Operator chaining

- Chaining:
  - Sender's `out.collect()` directly calls into receiver
  - Records are passed along with a defensive copy
    - but without any of the lower layers of Flink (de/ser*, network stack)
  - Only possible during **forward** data exchange



Object Reuse

- disabled by default
  - `ExecutionConfig#enableObjectReuse()`
- when enabled
  - streaming: same object is used throughout operator chains
  - batch: more complicated re-use patterns possible https://ci.apache.org/projects/flink/flink-docs-stable/dev/batch/index.html#object-reuse-enabled

Object Reuse in Streaming

Effects

Without a defensive between two operators

- Second operator could change a value the first one is holding
  - stored in field in first operator code
  - stored in (heap-based!) state back-end
  - first operator accesses value after `out.collect()`
- Value storage can be implicit
  - Window state
  - Context key
- Applies to method parameters, Iterables,...

Object Reuse in Streaming

Restrictions

- It is **not safe** to remember input objects across function calls.
- You **must not** modify input objects.
- You *may* modify an output object and emit it again (following the rules above).

Immutable Types (Streaming)

- Serialization via `PojoSerializer` requires getters **and setters** (or public fields).
  - Otherwise falls back to Kryo
- Non-guarded access vs. slow serialization
- Can work around by providing custom serializers.
- Can distinguish between
  - Wire serialization
  - State serialization

Custom Serialization

```java
@TypeInfo(MyTupleTypeInfoFactory.class)
public class MyTuple<T0, T1> {
  public T0 myfield0;
  public T1 myfield1;
}

public class MyTupleTypeInfoFactory extends TypeInfoFactory<MyTuple> {

  @Override
  public TypeInformation<MyTuple> createTypeInfo(
            Type t, Map<String, TypeInformation<?>> genericParameters) {
    return new MyTupleTypeInfo(genericParameters.get("T0"), genericParameters.get("T1"));
  }
}
```

Exercises

Exercise 5

- Run `ObjectReuseJob` and observe the provided graphs in Grafana.
- Improve throughput by enabling object reuse
  - adapt Job arguments in deployment: `mainArgs: '--objectReuse true'`.
- What are the effects? **Try to fix incorrect results.**

Bonus Task 5.1

Make sure that data types are immutable. What are the effects?

Bonus Task 5.2

Use a custom serializer to avoid serialization via Kryo. Compare object-reuse `enabled` and `disabled`.

ververica