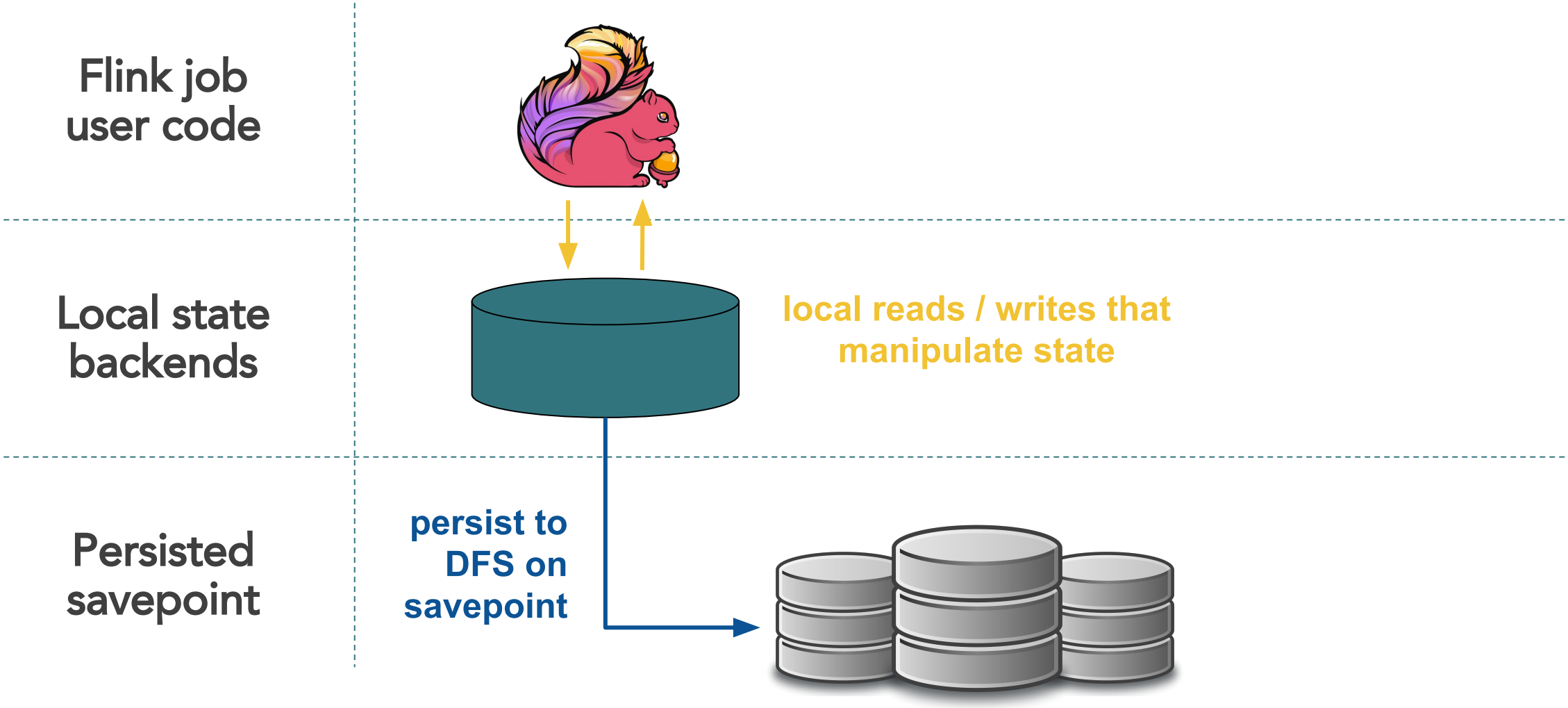
State Migration

Nico Kruber, Solutions Architect

Anatomy of a Flink Stream Job Upgrade



 |

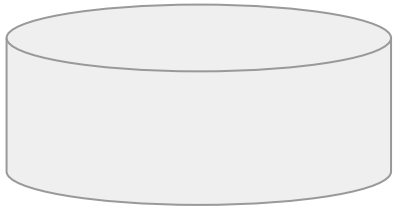
Anatomy of a Flink Stream Job Upgrade

Flink job user code

Local state backends

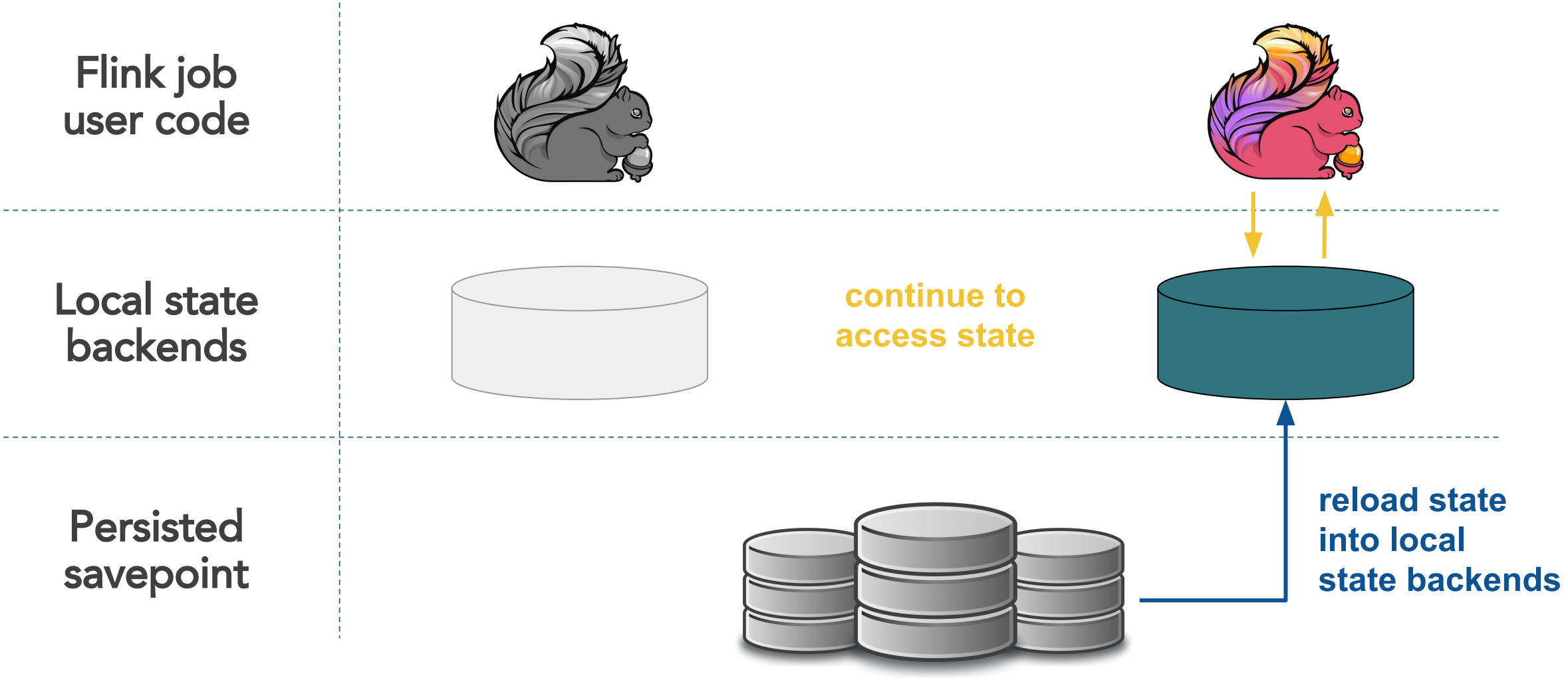
Persisted savepoint

Upgrade Application

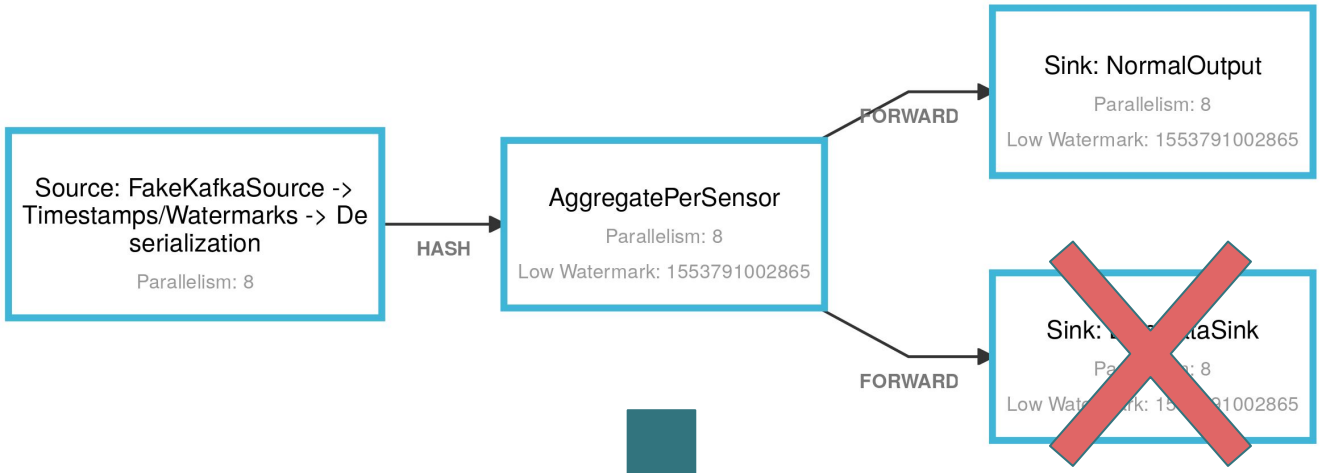


1. *Upgrade Flink cluster*
2. *Fix bugs*
3. *Pipeline topology changes*
4. *Job reconfigurations*
5. *Adapt state schema*
6. *...*

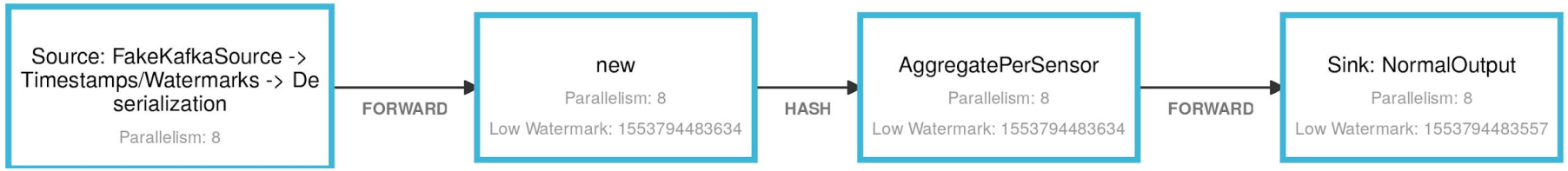
Anatomy of a Flink Stream Job Upgrade



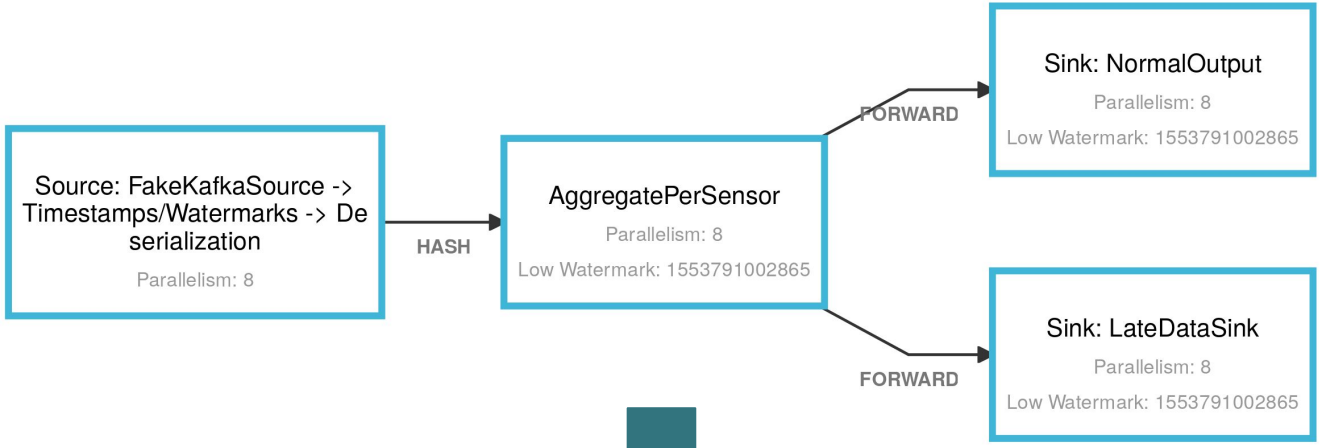
Job Upgrade - Topology Changes




```
flink run --allowNonRestoredState
<jar-file> <arguments>
```



Job Upgrade - Topology Changes




```
streamOperator
    .uid("AggregatePerSensor")
```

new operator starts → empty state

Still the same?

 |

State Registration with Built-in Serialization

```
ValueStateDescriptor<AggregatedSensorStatistics> aggregationStateDesc =
        new ValueStateDescriptor<>(
                "aggregationStats",
                AggregatedSensorStatistics.class
        );
aggregationState = getRuntimeContext().getState(aggregationStateDesc);
```

type information for state

- Flink infers information about the type and creates a serializer for it
  - Primitive types: `IntSerializer`, `DoubleSerializer`, `LongArraySerializer`, etc.
  - Tuples: `TupleSerializer`
  - POJOs / Scala case classes: `PojoSerializer`, `CaseClassSerializer`
  - Apache Avro types: `AvroSerializer`
  - Fallback is Kryo: `KryoSerializer`

Status Quo of Schema Evolution Support

- **Flink 1.7**: **schema evolution for Avro types** (only)
  - Can evolve schema according to Avro specifications*
  - Can swap between `GenericRecord` and code-generated `SpecificRecords`
  - **Cannot change namespace** of generated `SpecificRecord` classes
- **POJO schema evolution for Flink 1.8**
- More planned for 1.9+: Scala case classes, Rows (for Flink Tables)
- Avoid using Kryo if you want evolvable schema for state

*Avro specifications: http://avro.apache.org/docs/1.7.7/spec.html#Schema+Resolution

Exercise

Exercise 5.1: State Migration with Avro

Task package: `com.ververica.training.statemigration`
`entryClass: com.ververica.training.statemigration.avro.StateMigrationJob`

1. Setup a **Stateful** deployment with **LATEST_SAVEPOINT** restore strategy
2. `Extend the job:`

Add an average value of the sensor data to `MeasurementAggregationReport` and extend `avro.SensorAggregationProcessing` accordingly.

   a. **Do not add another state object!** Instead, extend the existing state class via `resources/avro/AggregatedSensorStatistics.avsc`
   b. run `mvn clean package` to re-generate the Java class via Avro
3. Verify the upgrade works in Ververica Platform:
   a. Upload new jar
   b. suspend the running deployment
   c. start again (will pick up new jar if file name hasn't changed)

 |

Walk-Through

Custom Serializers

Why Custom Serializers?

- only way to support schema evolution pre-1.7
- performance tuning
- custom schema evolution
- special needs

State Registration with Custom Serializers

```
ValueStateDescriptor<AggregatedSensorStatistics> aggregationStateDesc =
     new ValueStateDescriptor<>(
           "aggregationStats",
           new AggregatedSensorStatisticsSerializerV1());

aggregationState = getRuntimeContext().getState(aggregationStateDesc);

class AggregatedSensorStatisticsSerializerV1
    extends TypeSerializer<AggregatedSensorStatistics> {...}
```

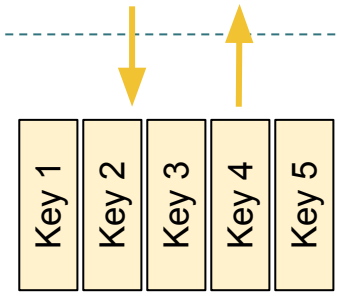
evolves with data schema and/or serialization format

State Migration in Heap-based State Back-ends

Flink job user code

```
ValueStateDescriptor<AggregatedSensorStatistics>
  aggregationStateDesc =
     new ValueStateDescriptor<>(
           "aggregationStats",
           new SerializerV1());
```

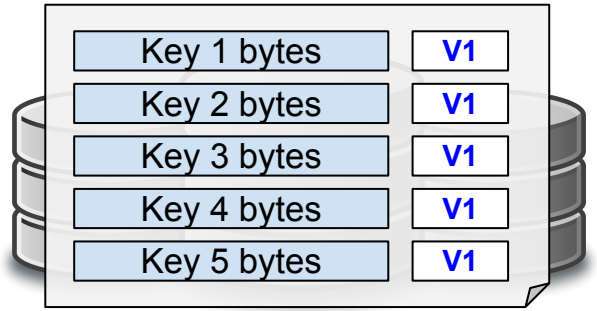
Local state backends



Persisted savepoint

Serialized by SerializerV1

… 010101110



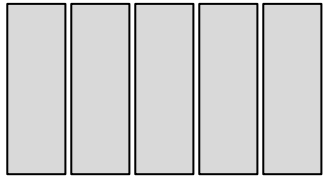
 |

State Migration in Heap-based State Back-ends

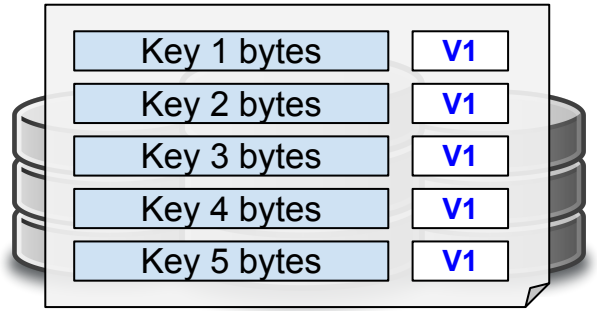
Flink job user code

```
ValueStateDescriptor<AggregatedSensorStatistics>
  aggregationStateDesc =
     new ValueStateDescriptor<>(
           "aggregationStats",
           new SerializerV2());
```

Local state backends



Persisted savepoint

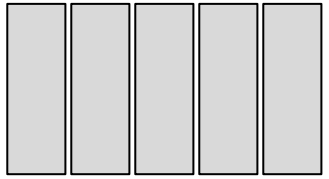


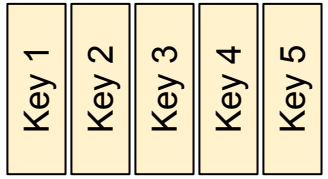
State Migration in Heap-based State Back-ends

Flink job user code

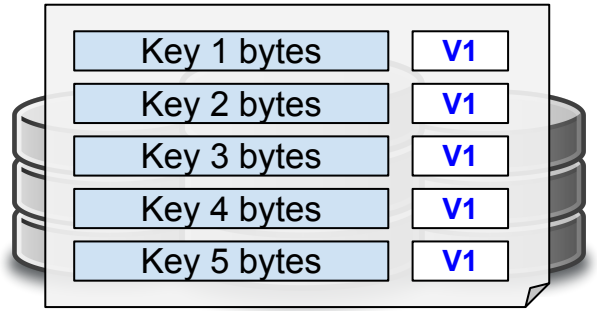
```
ValueStateDescriptor<AggregatedSensorStatistics>
  aggregationStateDesc =
    new ValueStateDescriptor<>(
          "aggregationStats",
          new SerializerV2());
```

Local state backends





Persisted savepoint



010101110 …

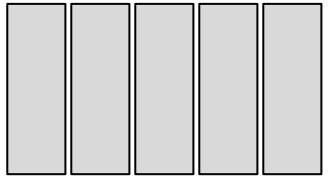
Requires SerializerV1 for restore

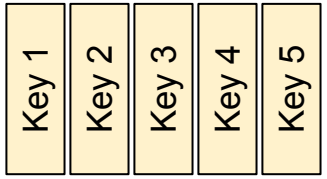
State Migration in Heap-based State Back-ends

Flink job user code

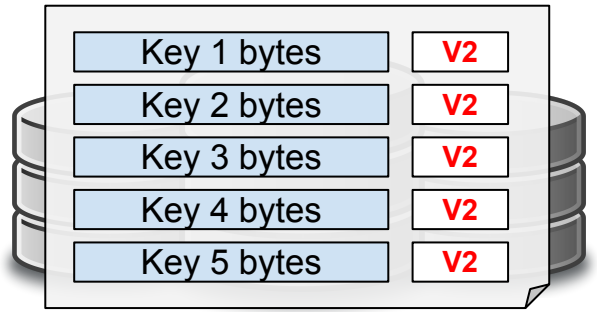
```
ValueStateDescriptor<AggregatedSensorStatistics>
  aggregationStateDesc =
     new ValueStateDescriptor<>(
           "aggregationStats",
           new SerializerV2());
```

Local state backends





Persisted savepoint



010101110 …

Serialized by SerializerV2

 |

State Migration for Out-of-Core State Back-ends

- state de/serialized for each access (vs. only on restore/snapshot)
- savepoint restore is file-copy only
- migration happens on first access if schema has changed

Evolving the Serializer

- changes in the binary format of the objects' representation
- changes in the deserialized class (added/removed fields, type changes, etc)

new serializer is compatible

- binary format only updated when touched
- reads all(!) previous versions

new serializer requires **migration**

- converts old format to new objects
- reads previous version (only)

 |

Main Abstraction: `TypeSerializerSnapshot`

```
public interface TypeSerializerSnapshot<T> {
    int  getCurrentVersion();

    void writeSnapshot(DataOutputView out);
    void readSnapshot(int readVersion, DataInputView in, ClassLoader userCodeClassLoader);

    TypeSerializer<T>                        restoreSerializer();

    TypeSerializerSchemaCompatibility<T> resolveSchemaCompatibility(
        TypeSerializer<T> newSerializer);
}
```

- Represents the written form of a state's serializer, written to snapshots
- Encodes information about the state's written schema + serializer configuration (for generic serializers)
- Serves as a factory for the previous serializer and defines migration type

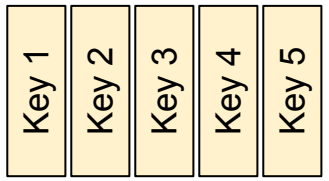
 |

State Migration in Heap-based State Back-ends

Flink job user code

```
ValueStateDescriptor<AggregatedSensorStatistics>
  aggregationStateDesc =
     new ValueStateDescriptor<>(
           "aggregationStats",
           new SerializerV1());
```

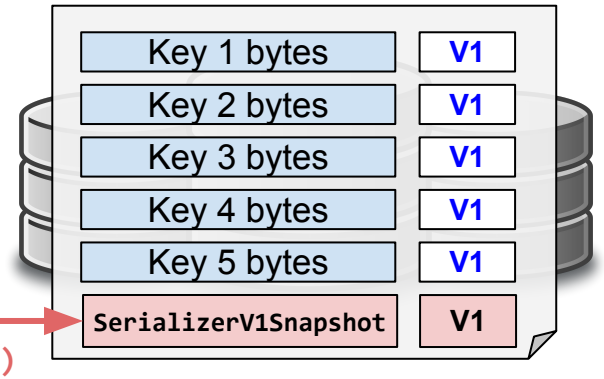
Local state backends



Persisted savepoint

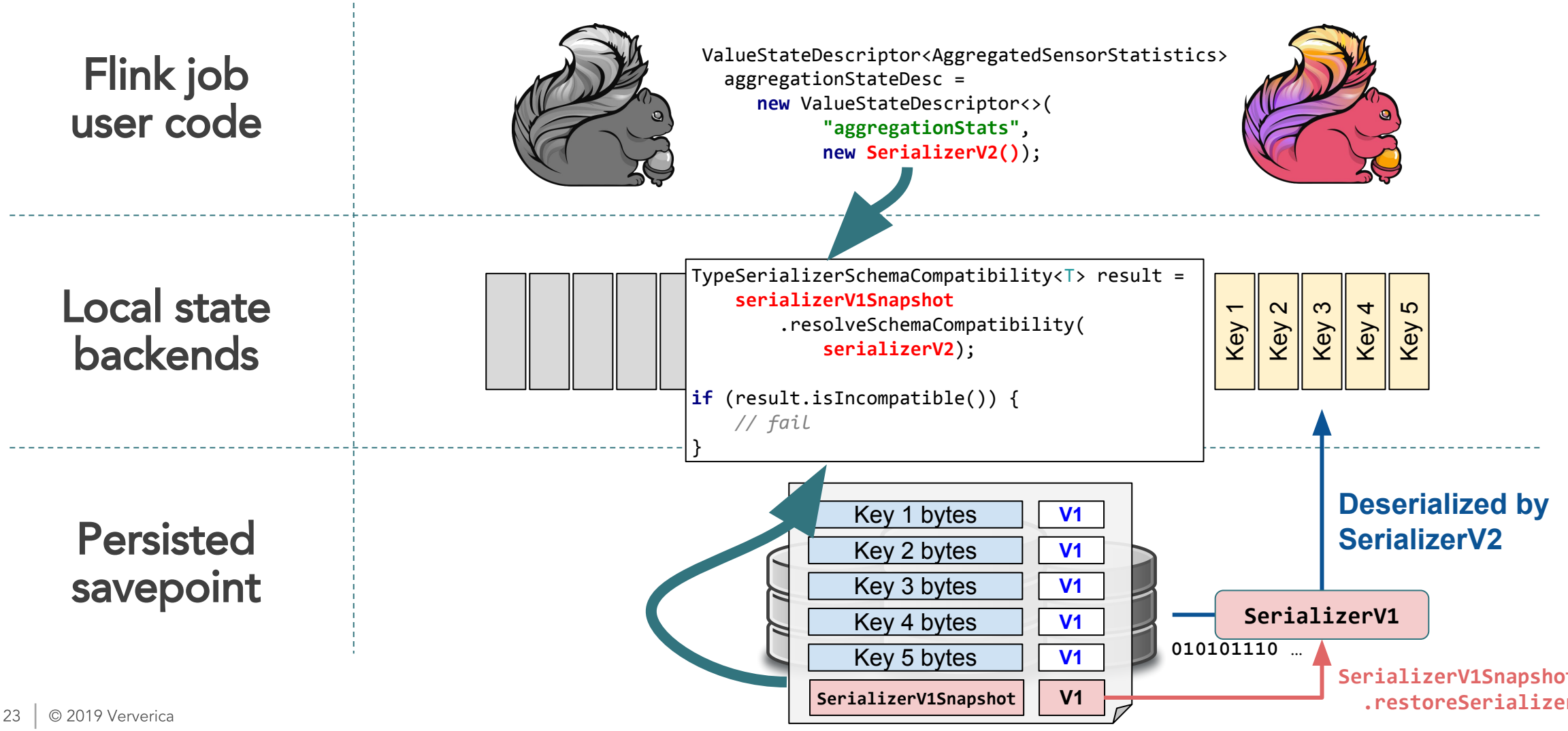
Serialized by SerializerV1

… 010101110




```
SerializerV1
  .snapshotConfiguration.write(...)
```

State Migration in Heap-based State Back-ends



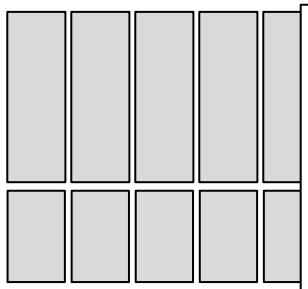
 |

State Migration in Out-of-Core State Back-ends

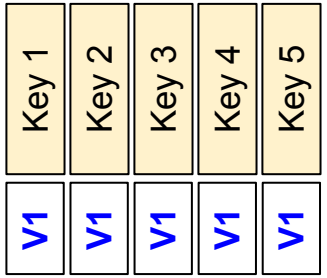
Flink job user code

```
ValueStateDescriptor<AggregatedSensorStatistics>
  aggregationStateDesc =
    new ValueStateDescriptor<>(
          "aggregationStats",
          new SerializerV2());
```

Local state backends

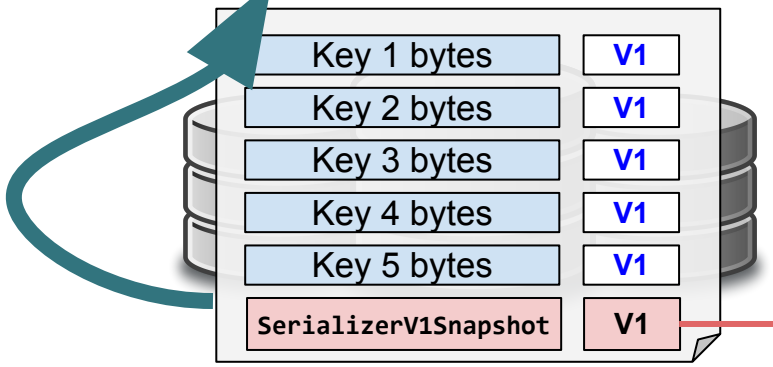


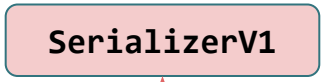
```
TypeSerializerSchemaCompatibility<T> result =
    serializerV1Snapshot
        .resolveSchemaCompatibility(
            serializerV2);
if (result.isCompatibleAfterMigration()) {
    // migrate all(!) state entries
} else if (result.isIncompatible()) {
    // fail
}
```



Migrate!

Persisted savepoint






```
SerializerV1Snapshot
  .restoreSerializer();
```

Best Practices

- Avoid classname changes to the serializer snapshot class
  - Classname is the entrypoint for reading the snapshot (instantiated via classname)
    - snapshot class needs public default constructor!
  - Do not use anonymous or nested classes as the snapshot class
- Use `CompositeSerializerSnapshot` to handle nested `TypeSerializers`

Exercise

Exercise 5.2: State Migration - Custom Serializer

Task package: `com.ververica.training.statemigration`
`entryClass: com.ververica.training.statemigration.custom.StateMigrationJob`

1. Setup a **Stateful** deployment with **LATEST_SAVEPOINT** restore strategy
2. `Extend the job:`

Add an average value of the sensor data to `MeasurementAggregationReport` and extend `custom.SensorAggregationProcessing` accordingly.

   a. **Do not add another state object!**
      Instead, extend the existing state class `custom.AggregatedSensorStatistics`
   b. adapt / create new custom serializers accordingly
3. Verify the upgrade works in Ververica Platform:
   a. Upload new jar
   b. suspend the running deployment
   c. start again (will pick up new jar if file name hasn't changed)

 |

Walk-Through

Bonus Exercise 5.3: Custom Serializer → Avro

Task package: `com.ververica.training.statemigration`
`entryClass: com.ververica.training.statemigration.custom.StateMigrationJob`

Migrate from the custom serializer to `AvroSerializer` so that future schema evolutions are covered by Avro and do not need a custom serializer anymore.

1. Change the (new) custom serializer from Exercise 5.2 so that it returns an instance of the Avro-generated class from Exercise 5.1.
2. Verify the upgrade works in Ververica Platform:
   a. Upload new jar
   b. suspend the running deployment
   c. start again (will pick up new jar if file name hasn't changed)

Walk-Through

ververica

nico@ververica.com www.ververica.com @VervericaData