Apache Flink Discrepancy Analysis

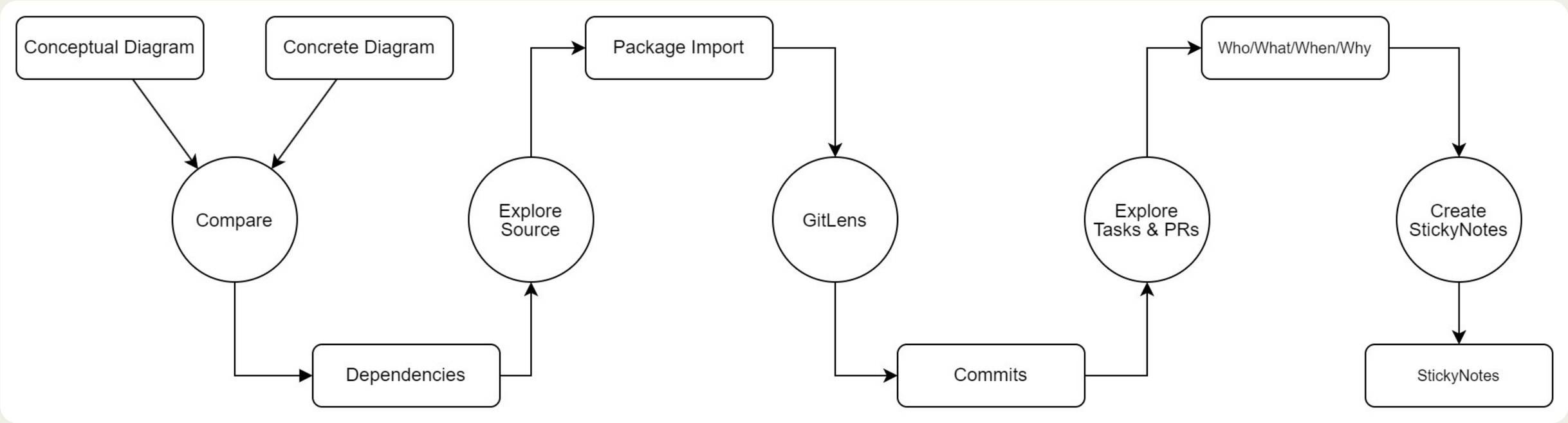
TEAM FLINKFORCE

Checkpoint Subsystem & Reflexion Models

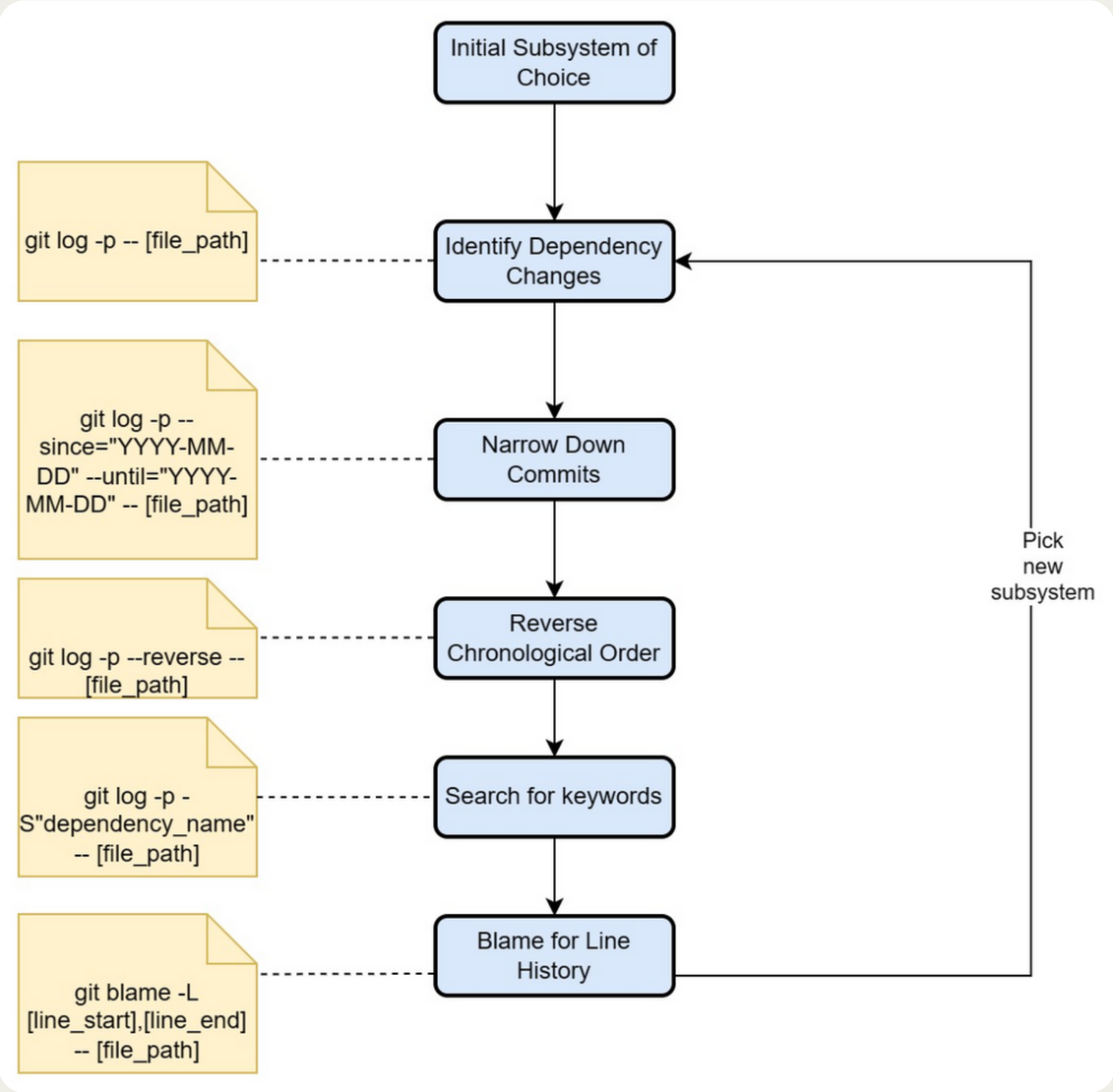
Agenda

1. Methodology & Reflexion Process
2. Discrepancy Analysis
    a. Top Level
    b. Runtime
    c. Checkpoint Subsystem
3. Revised Architecture
4. Concurrency Aspects
5. Team Issues
6. Limitations & Lessons Learned
7. Q&A

Reflexion Methodology



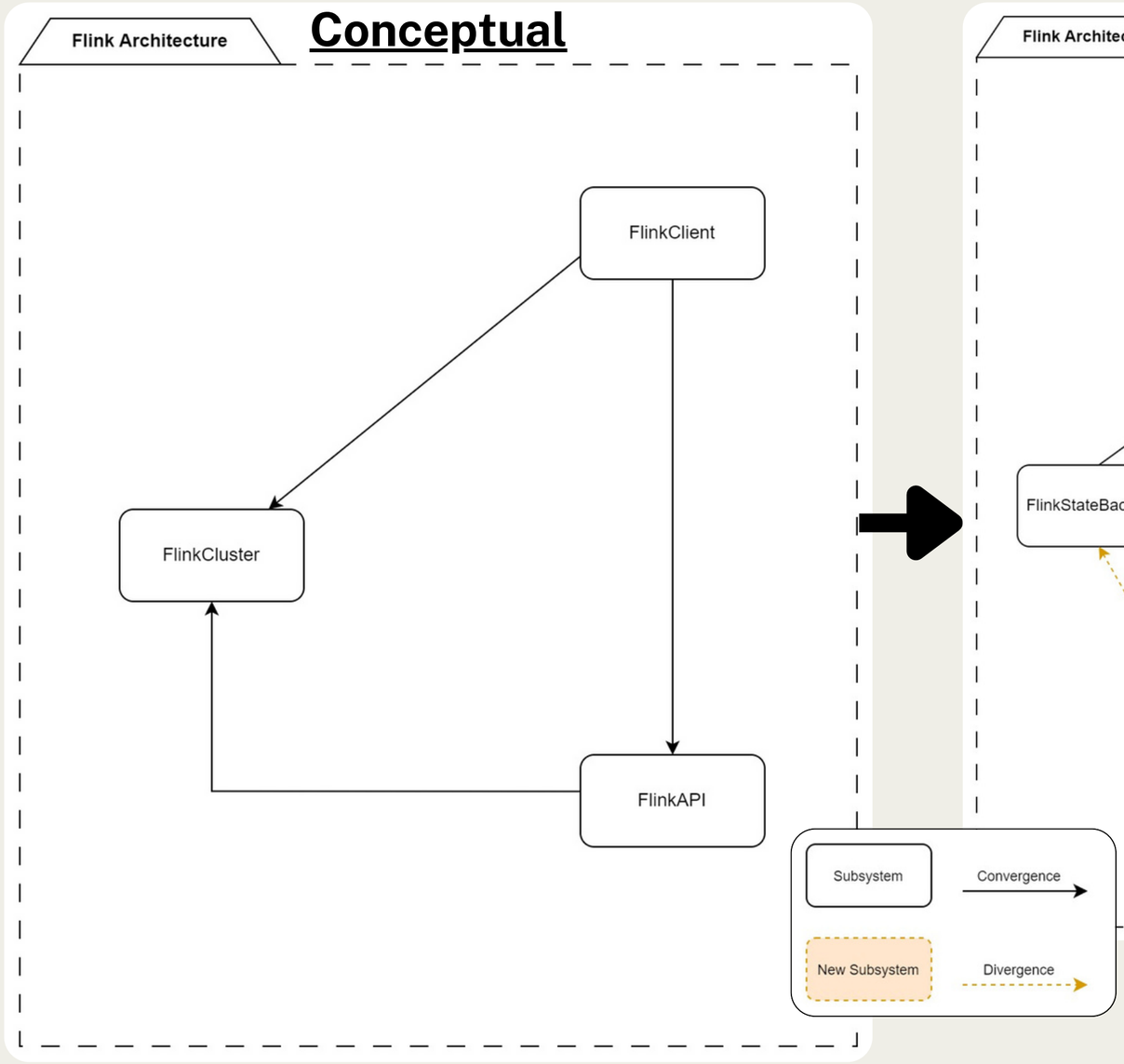
Alternate Methodology



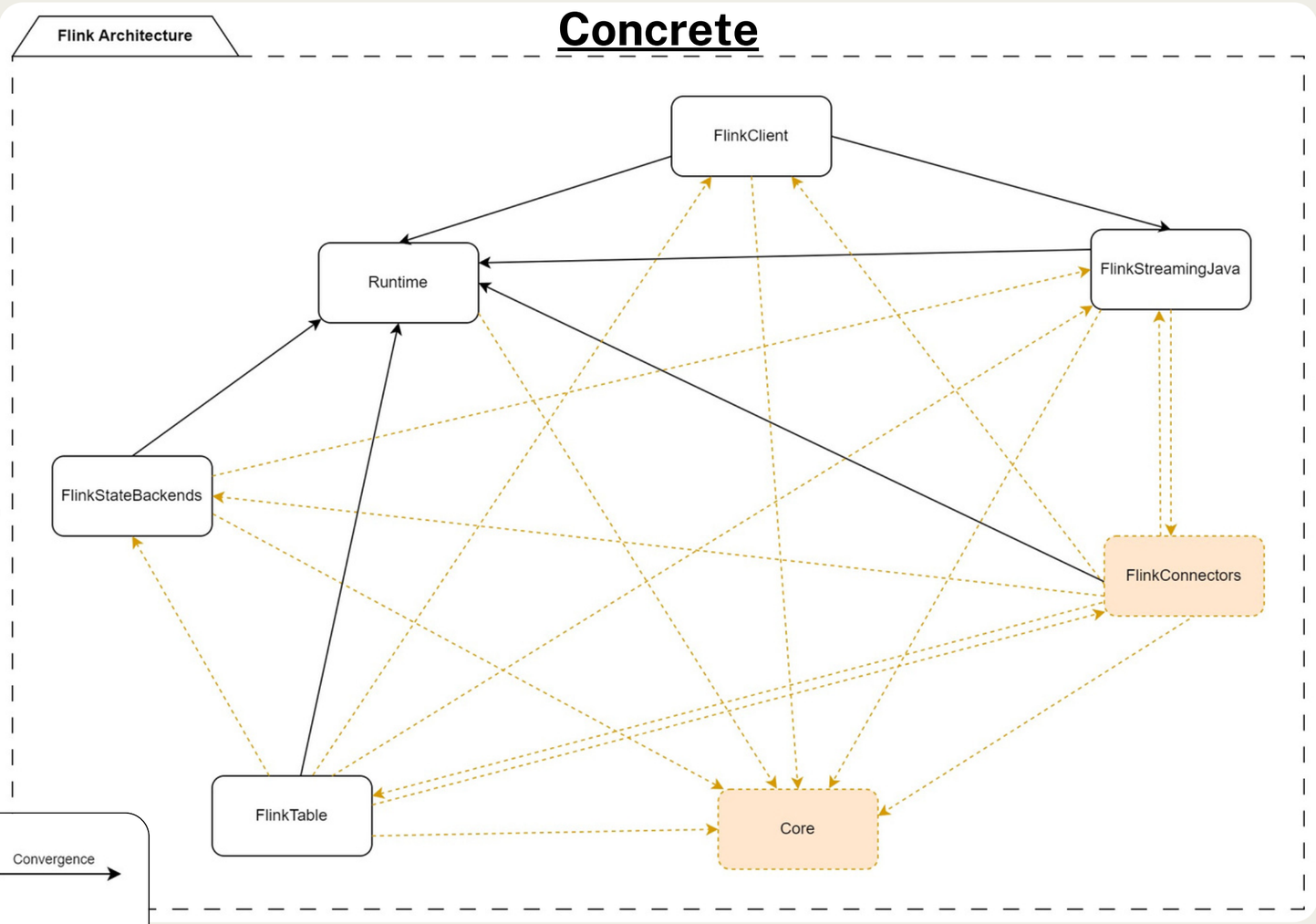
- Too time consuming
- Need a script
- No graphics/visuals

Top Level

Conceptual

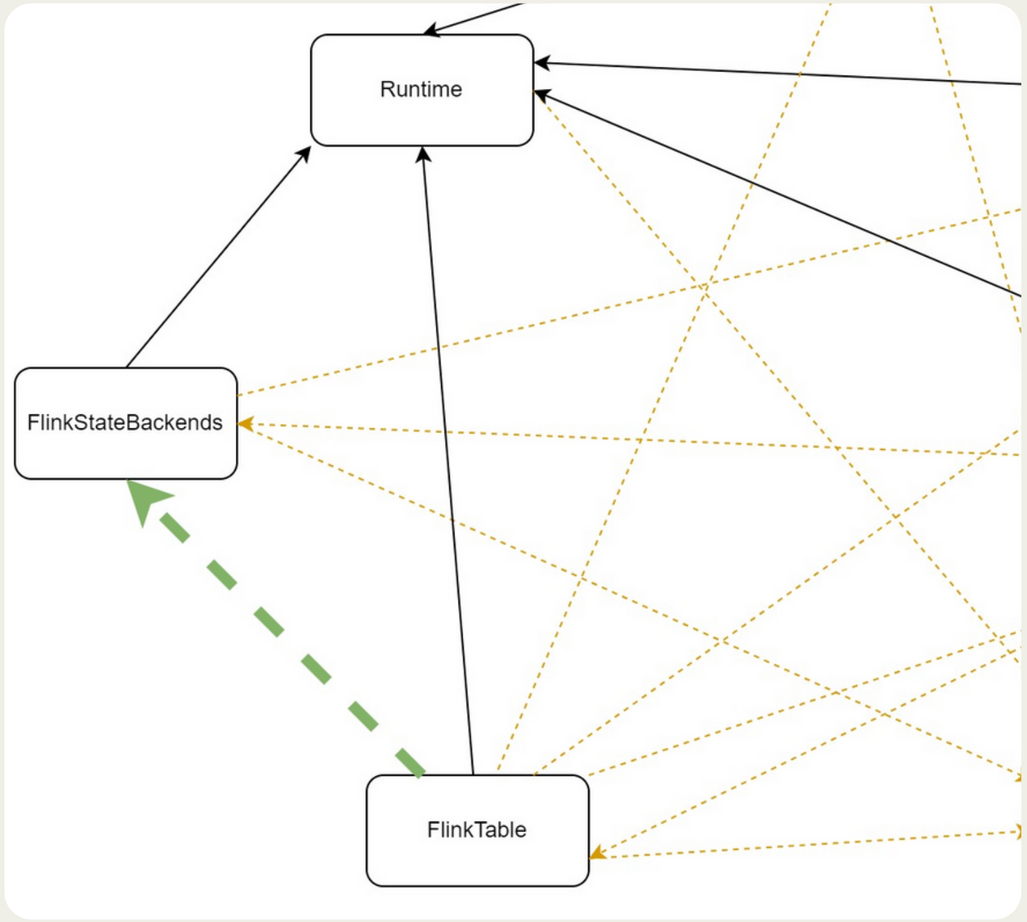


Concrete



Convergence Concerns

Divergence




```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-statebackend-rocksdb</artifactId>
    <version>${project.version}</version>
    <scope>test</scope>
```

Chesnay Schepler, 2 years ago *(October 25th, 2021 6:48 PM)*

[FLINK-24018][build] Remove Scala dependencies from Java APIs

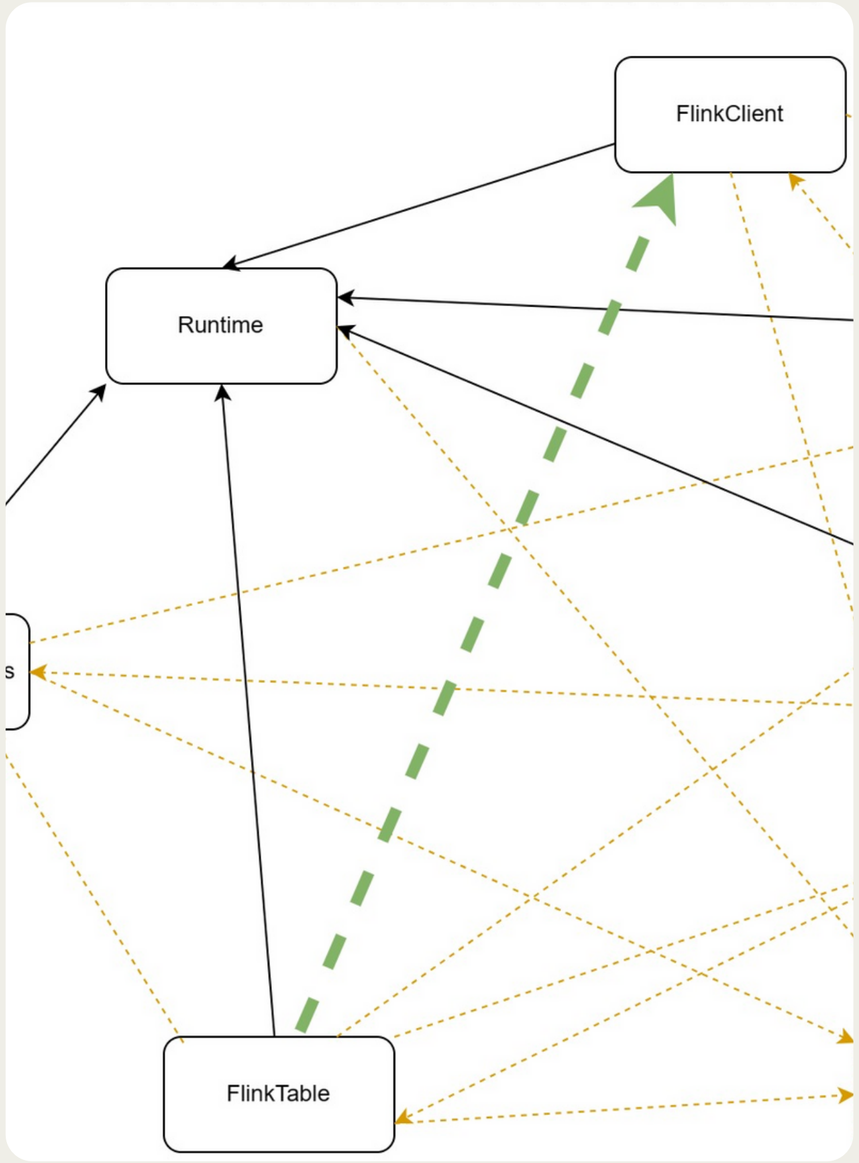
dd48d05 | Connect to GitHub... | Team ... | ···

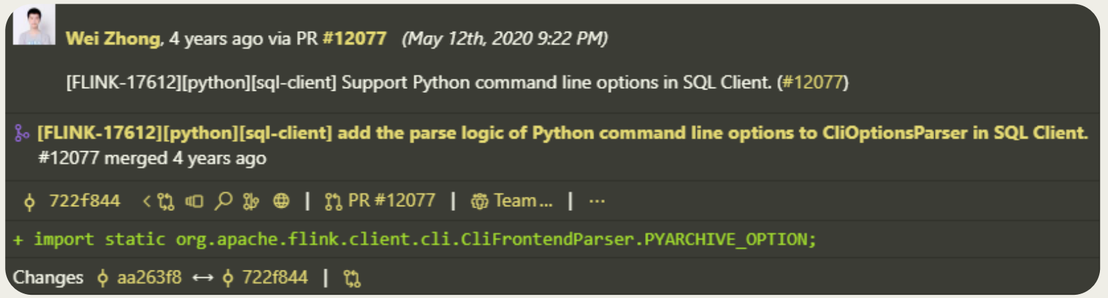
```
+ <artifactId>flink-statebackend-rocksdb</artifactId>
```

Changes 055c8c8 ↔ dd48d05

Which	Package org.apache.flink.**table** **Depends on** Package org.apache.flink.**statebackends**
Who	Committer: **Chesnay Schepler** Reviewer: **Matthias Pol**
When	PR [FLINK-24018] committed on Oct. 25, 2021
Why	The FlinkTable module now uses FlinkStateBackends directly for unit testing. This change was made after Scala dependencies were removed from two other modules, flink-streaming-java and flink-cep. Since FlinkTable needs to handle data reliably, especially when it's constantly changing during streaming, it's important to test that its state management works well. FlinkStateBackends helps with this, and the well-known RocksDB backend is part of it. This new setup helps to check everything works as expected, even without the Scala parts that used to be there. It's a careful step to make sure nothing breaks with the updates.

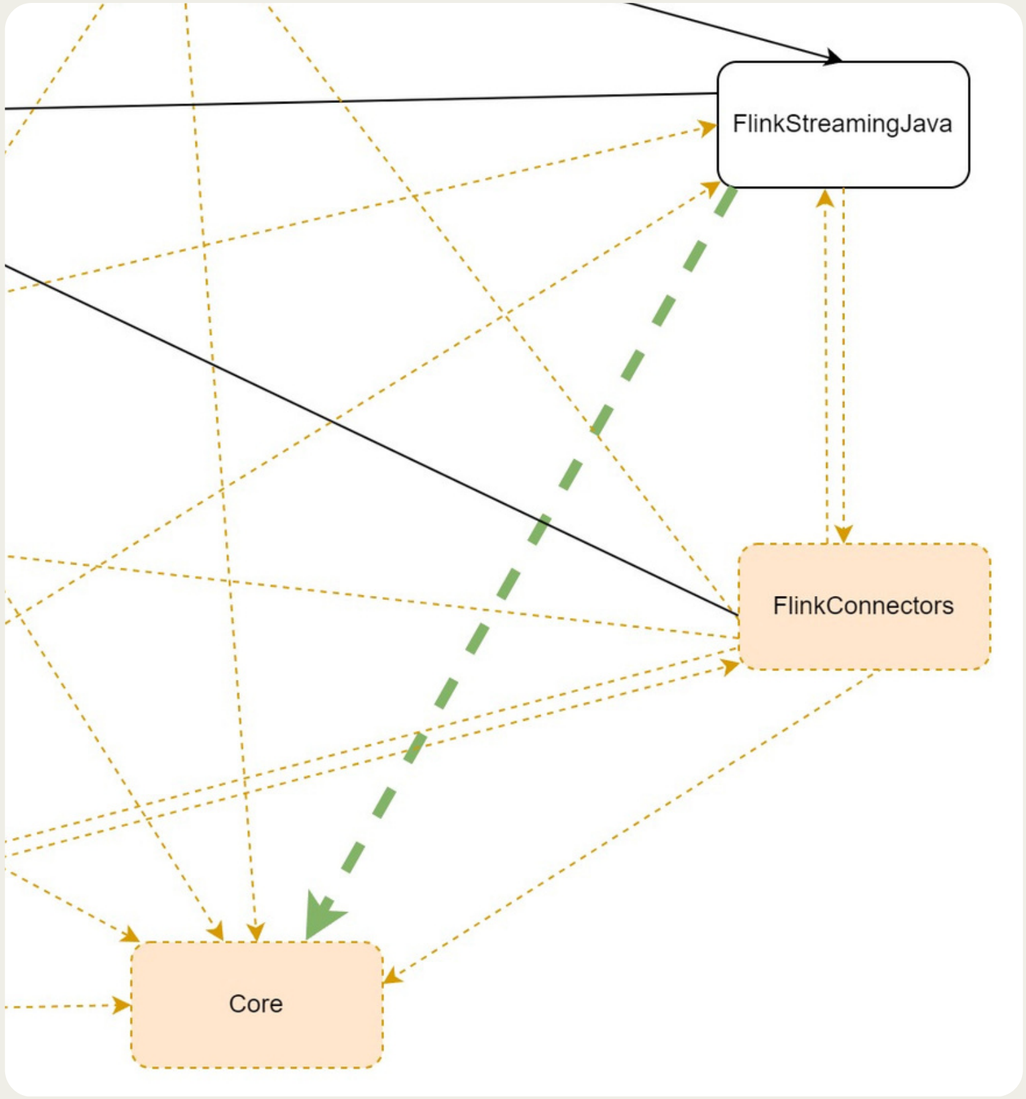
Divergence

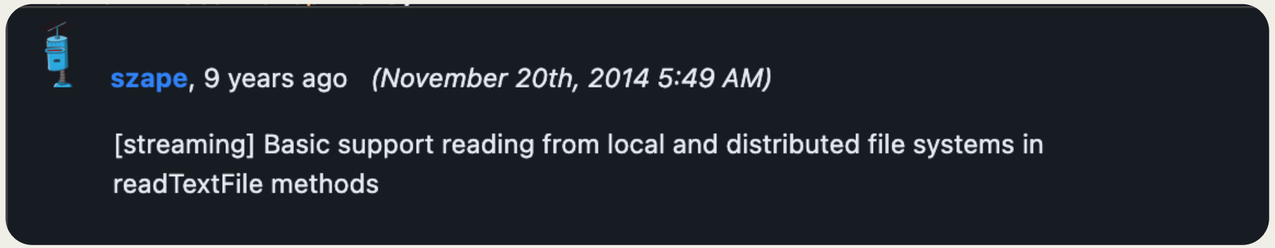




Which	Package org.apache.flink.**table** **Depends on** Package org.apache.flink.**client**
Who	Committer: **Wei Zhong** Reviewer: **Diane Fu**
When	PR [FLINK-17612] committed on May 11, 2020
Why	The dependency between FlinkTable and FlinkClient was established to enhance FlinkClient with the capability to parse Python command-line options. This was necessary because FlinkTable, which provides a unified table API for batch and stream processing, needed to support configurations for Python UDFs (User-Defined Functions) within the SQL Client environment.

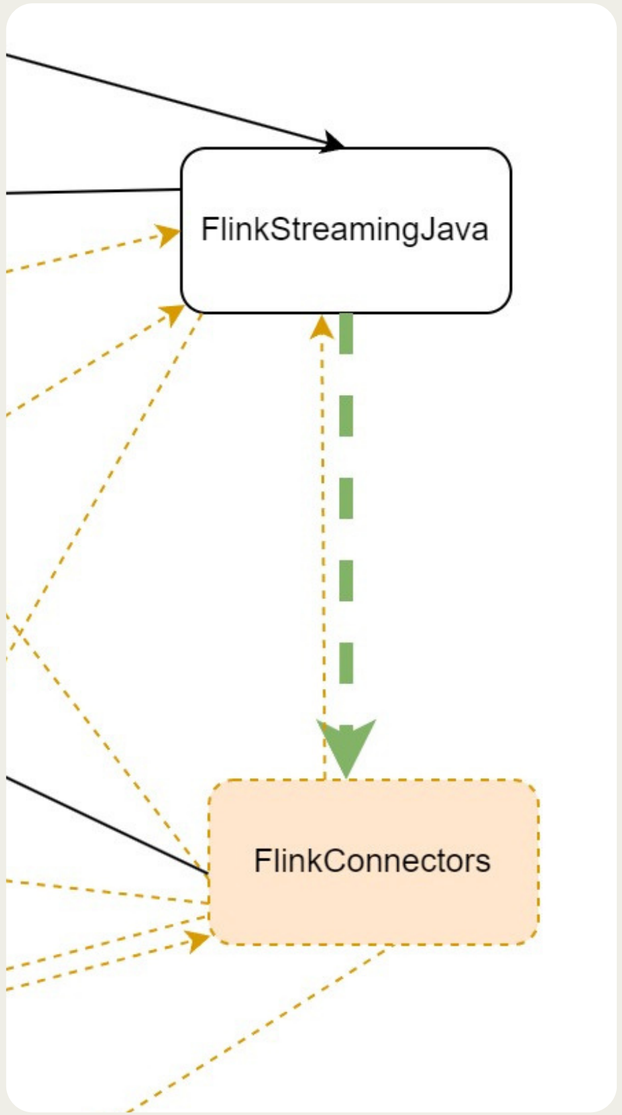
Divergence





Which	Package org.apache.flink.**streaming** **Depends on** Package org.apache.**core**
Who	Committer: **szape**
When	Dec 15, 2014
Why	Streaming imported functionalities from core to help with reading text files in distributed file systems. 2 main classes were imported, Path and InputSplit.

Divergence

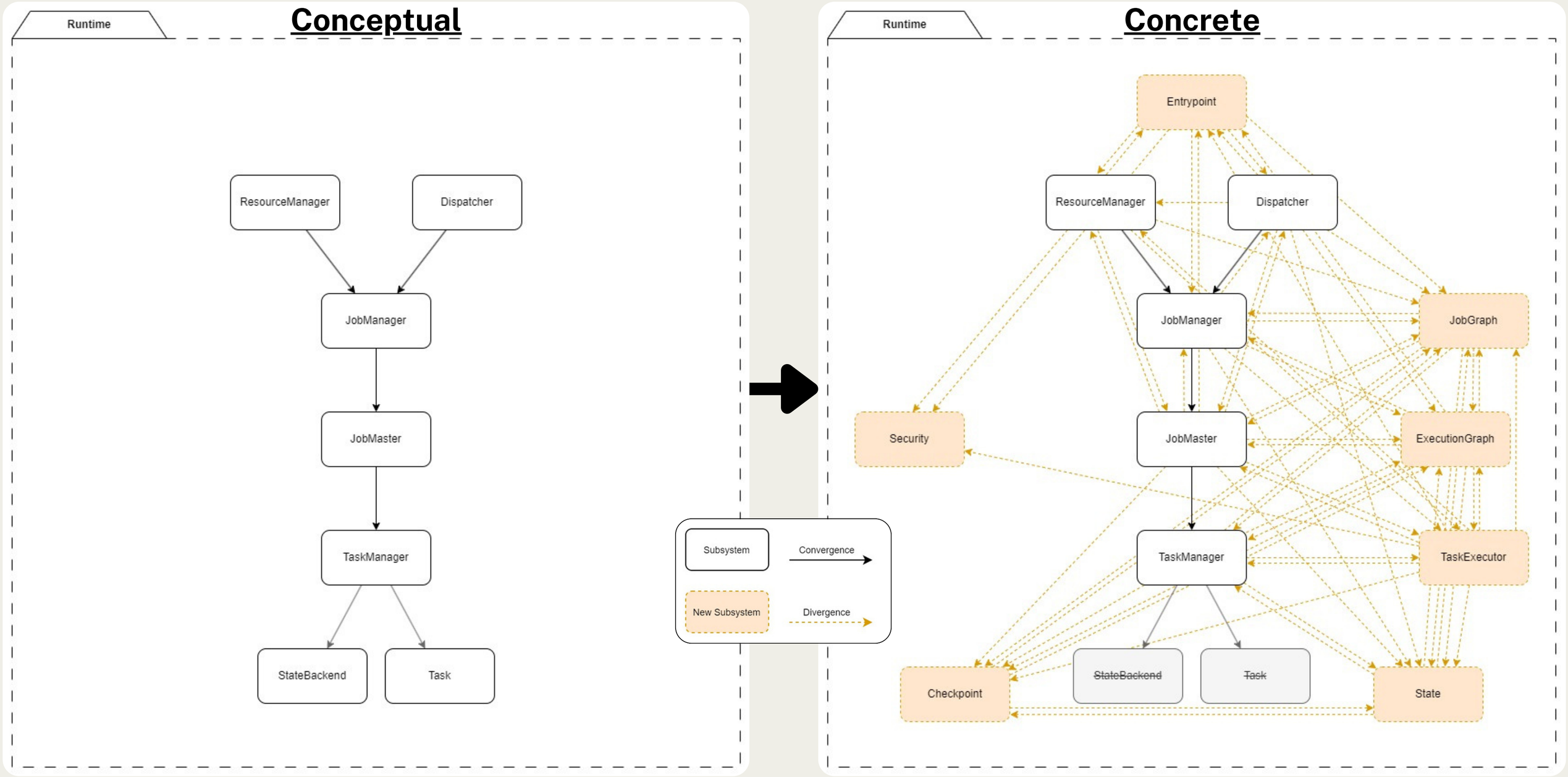


MartijnVisser, 16 months ago *(July 20th, 2022 3:09 AM)*

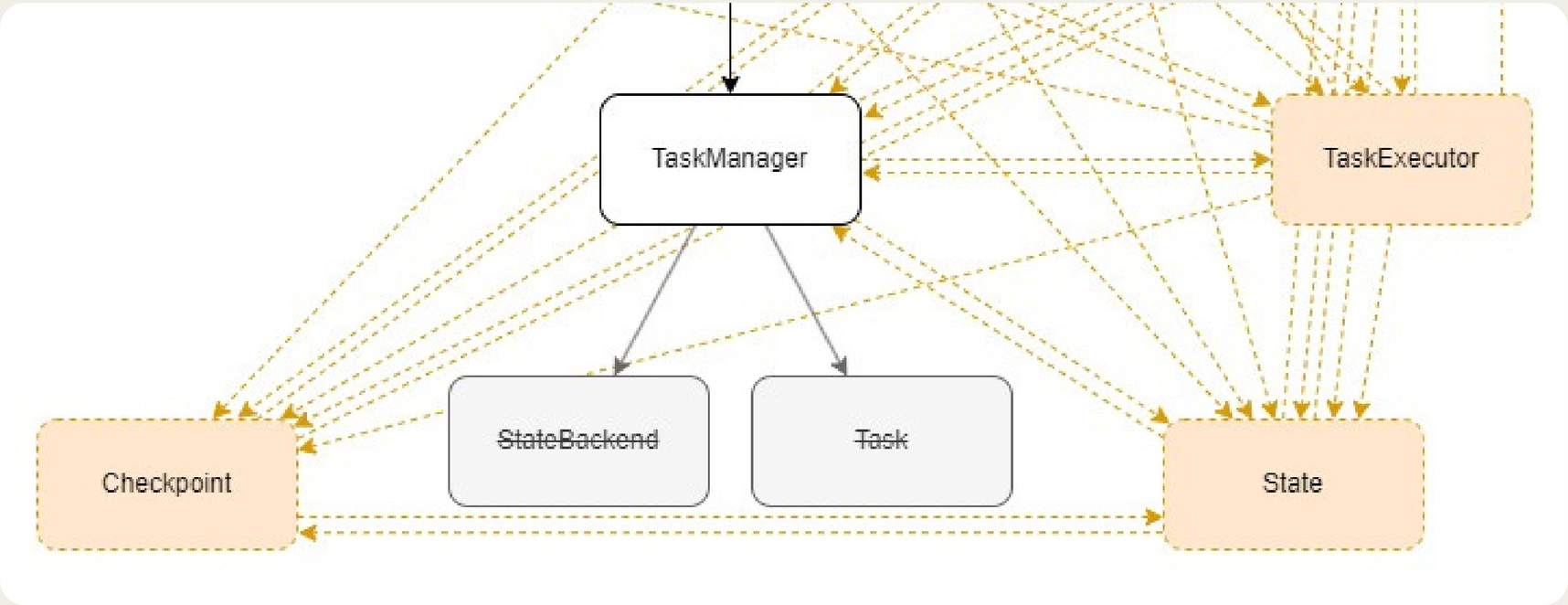
[FLINK-27188][Connector][StreamingFileSink] Mark StreamingFileSink as deprecated since it has been replaced by FileSink. This includes replacing or removing references to StreamingFileSink in the documentation

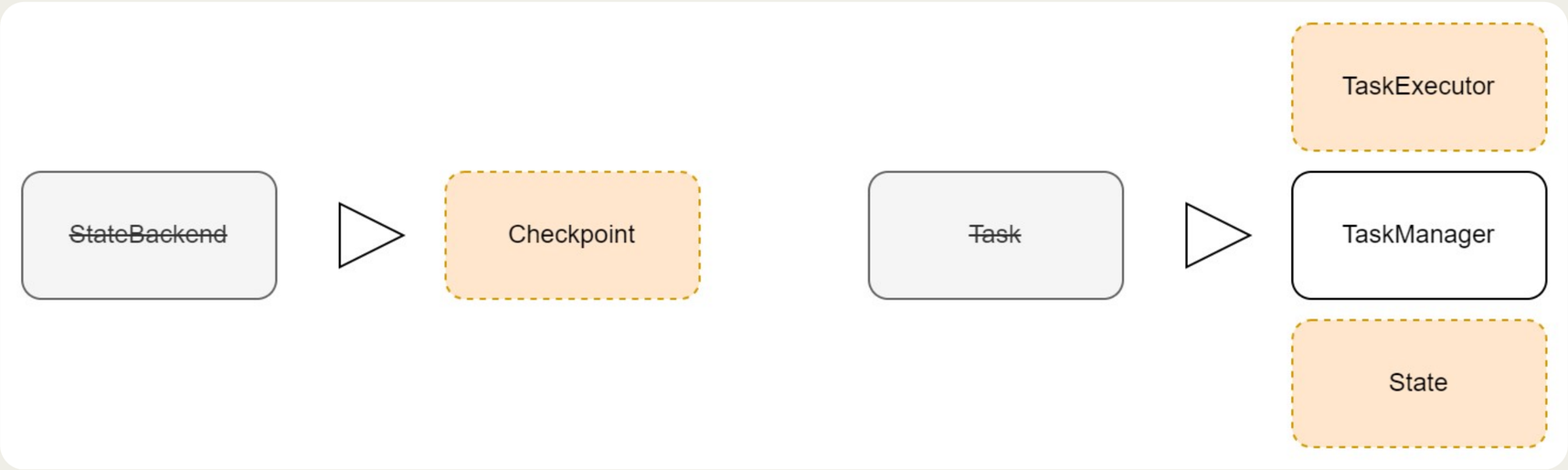
Which	Package org.apache.flink.**streaming** **Depends on** Package org.apache.flink.**connector**
Who	Committer: **Martjin Visser**
When	Committed on Jul 22, 2022
Why	The dependency between streaming and connector exists because streaming utilized a sink existing in connector. StreamingFileSink was a sink that placed input elements into files that were categorized into buckets. That sink however was programmed to work with batch data stream inputs, therefore the commit above aims to deprecate that and replace it with a new sink labelled “FileSink” which aims to support both bounded and unbounded datastreams.

Runtime Level

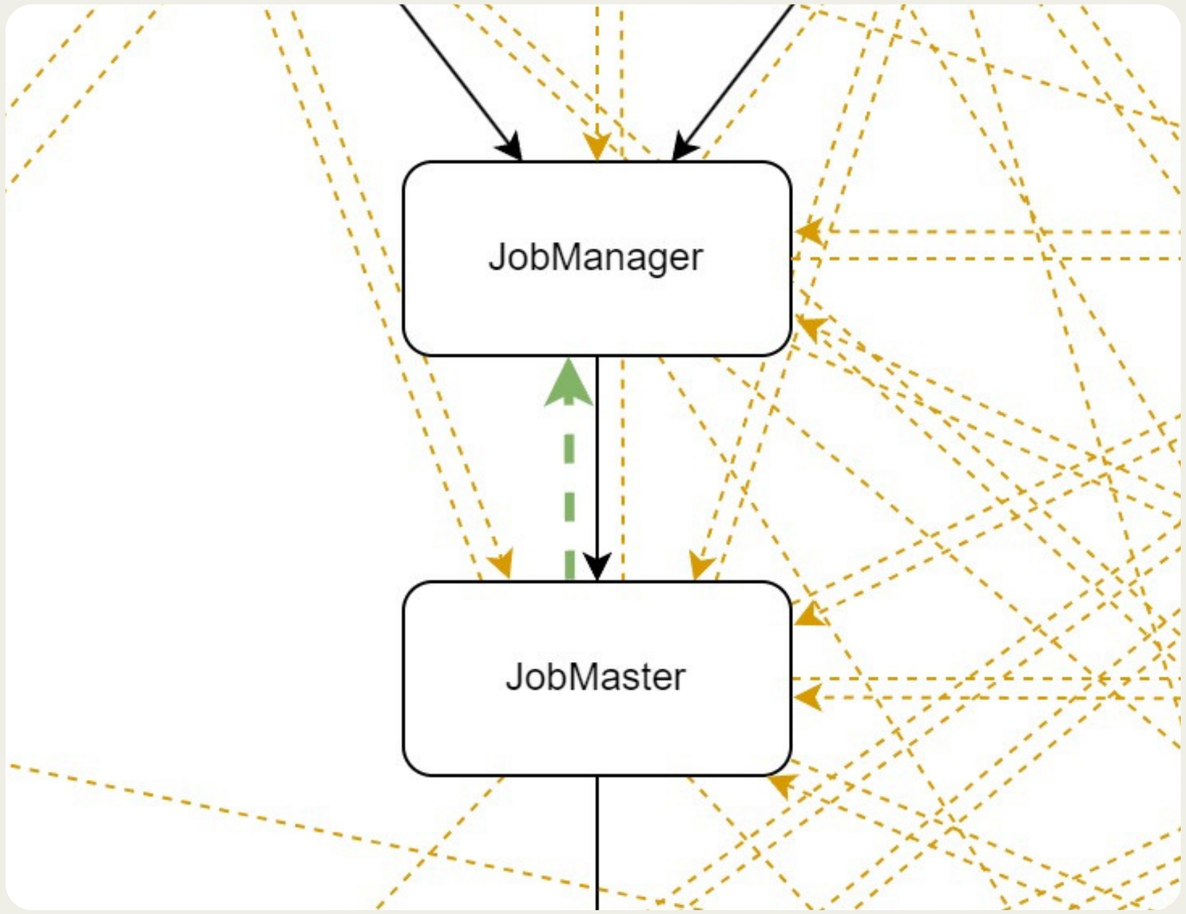


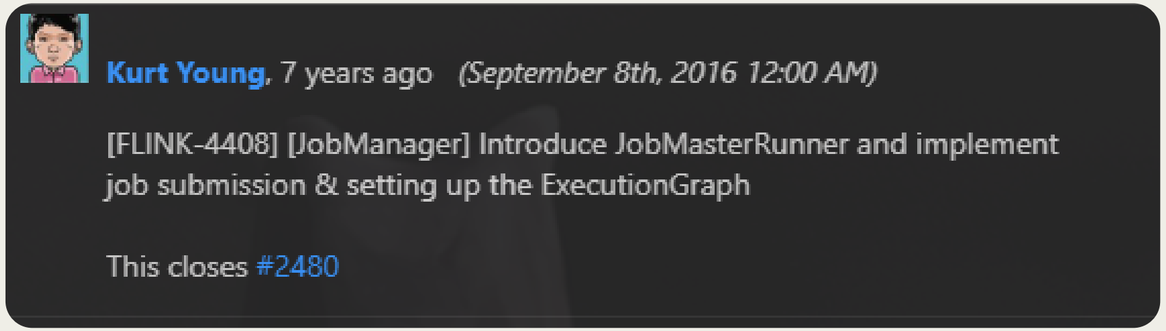
Convergence Concerns





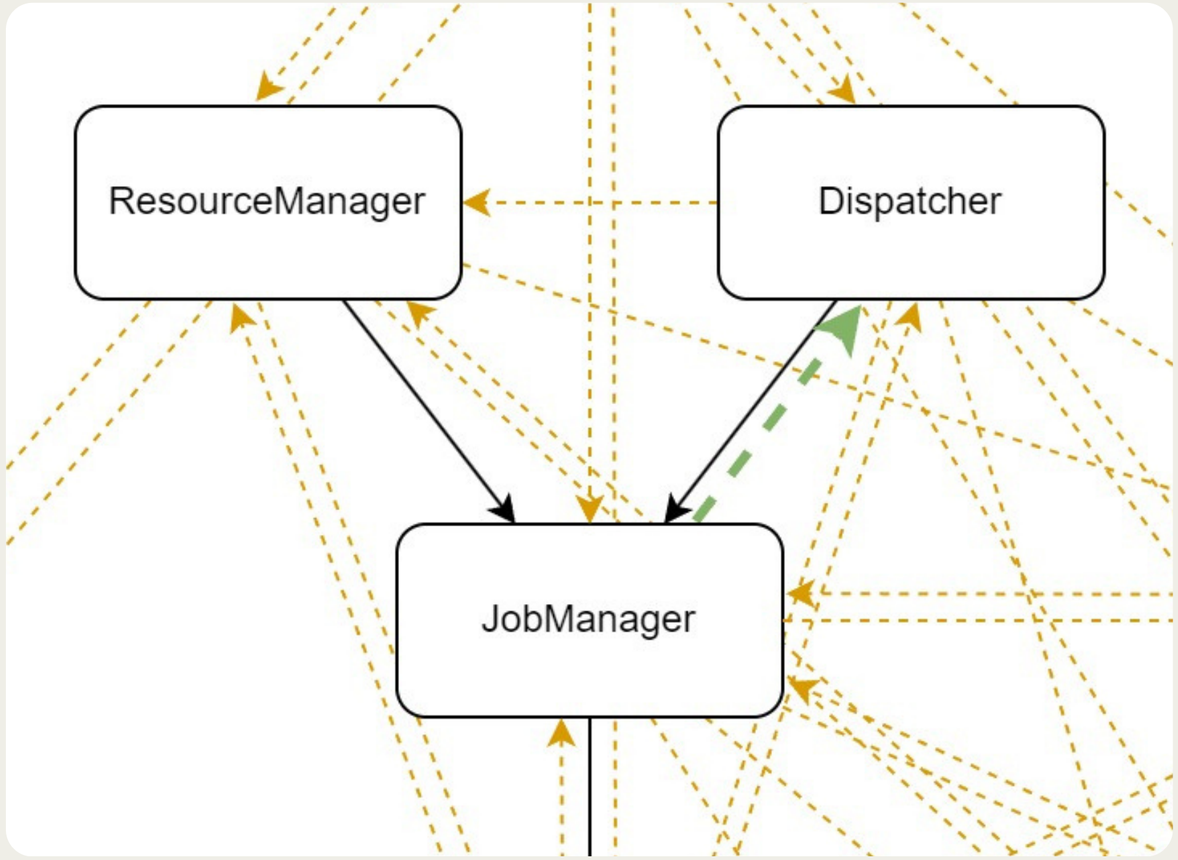
Divergence

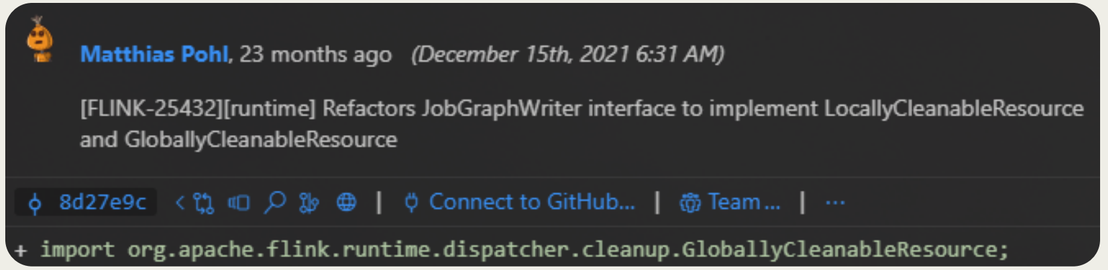




Which	Package org.apache.flink.runtime.**jobmaster** **Depends on** Package org.apache.flink.runtime.**jobmanager**
Who	Committer & Author: **Kurt Young** Reviewer: **Stephan Ewen**
When	PR [FLINK-4408] committed on Sept 8, 2016
Why	To handle job-level leader election and ensure the proper response of the underlying job manager, including determining if job submission should include recovery. Also, it Implements the framework for job submission and configures the ExecutionGraph, with interactions with the client still pending.

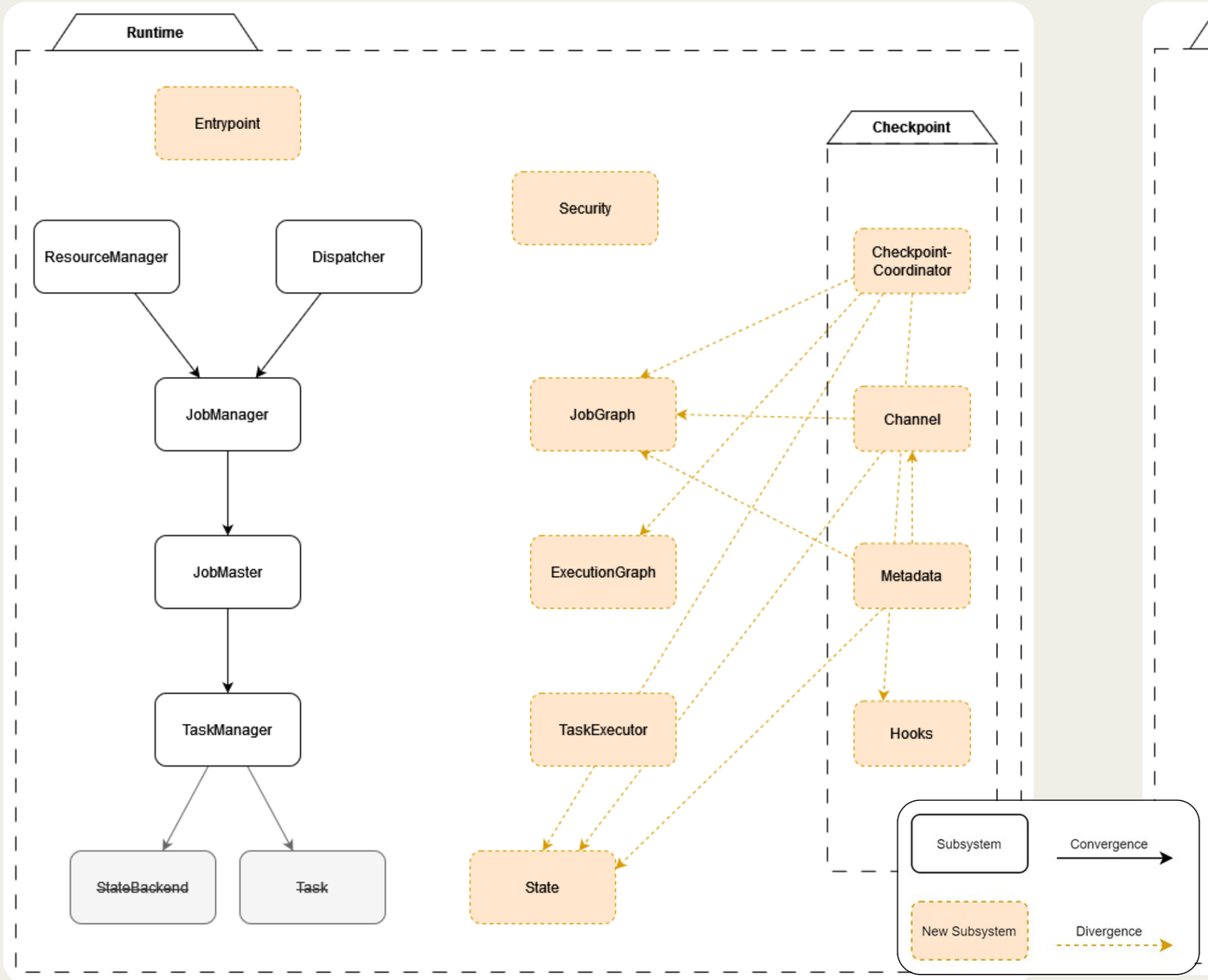
Divergence

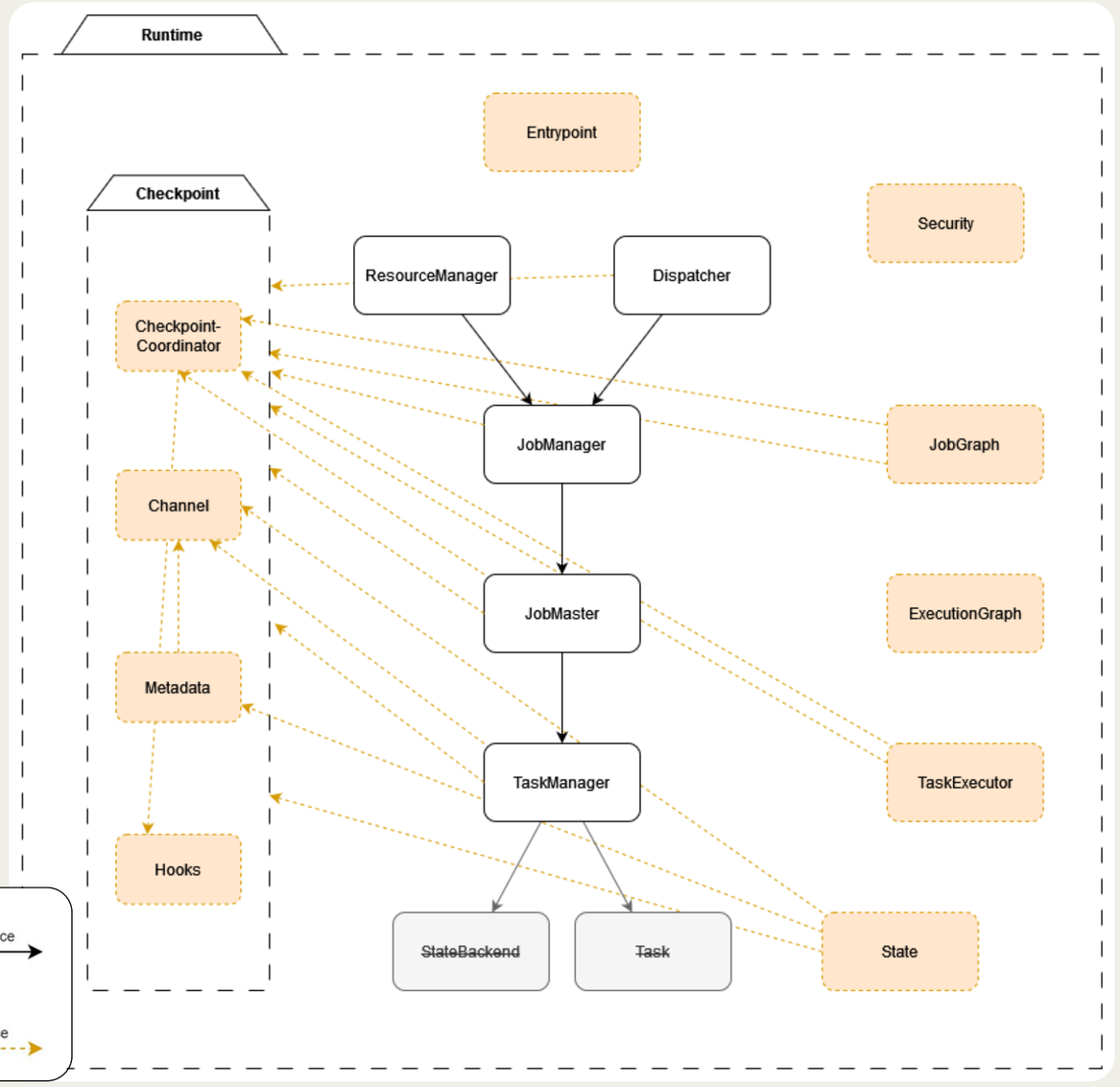




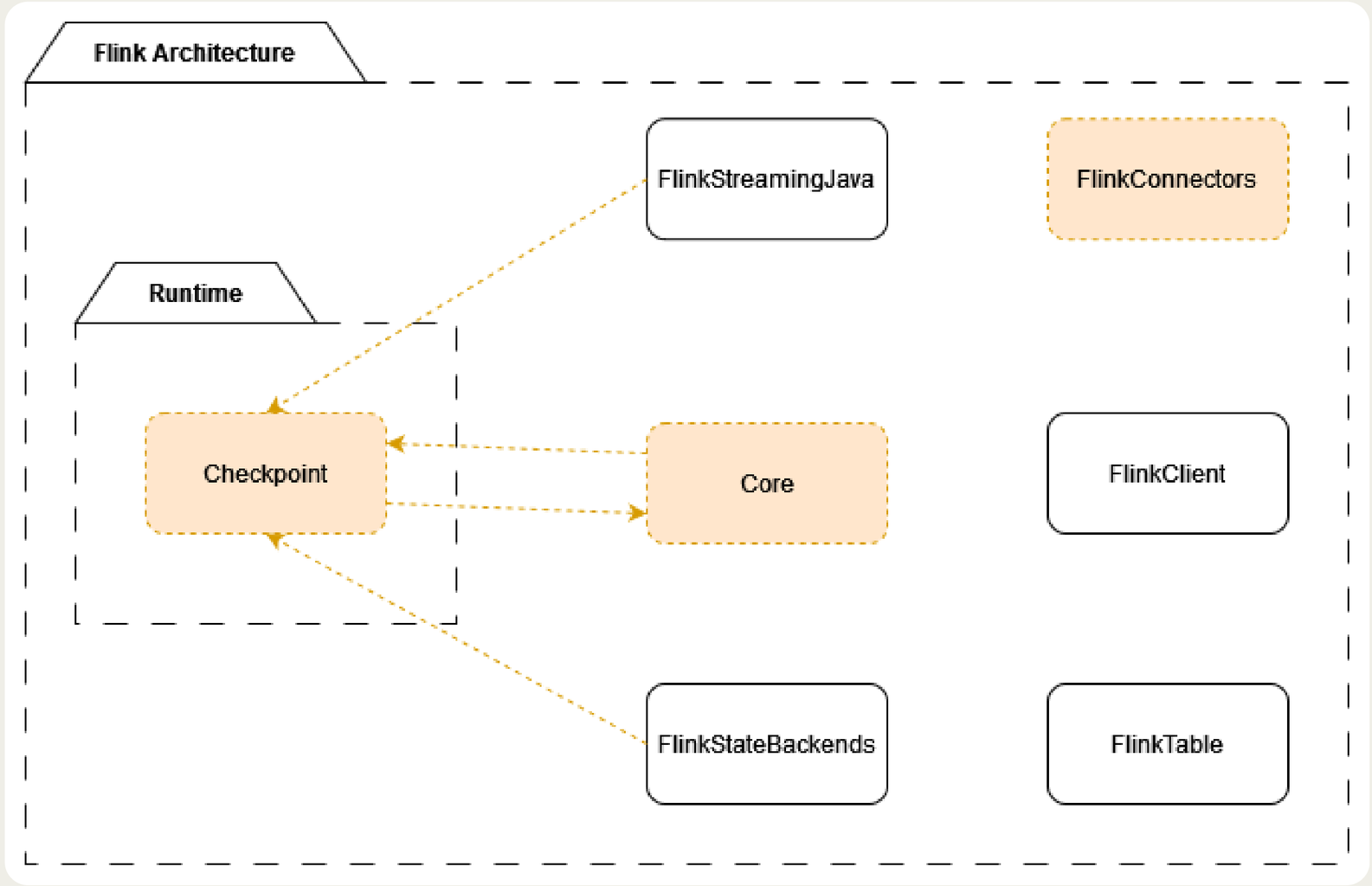
Which	Package org.apache.flink.runtime.**jobmanager** **Depends on** Package org.apache.flink.runtime.**dispatcher**
Who	Committer: **Matthias Pohl** Reviewer: **David Moravek**
When	PR [FLINK-25432] committed on Dec 15th 2021
Why	JobGraphWriter.java was refactored to extend and implement LocallyCleanableResource.java and GloballyCleanableResource.java. “We want to combine the job-specific cleanup of the different resources and provide a common ResourceCleaner taking care of the actual cleanup of all resources.”

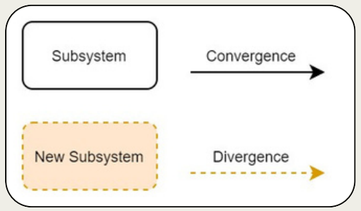
Checkpointing



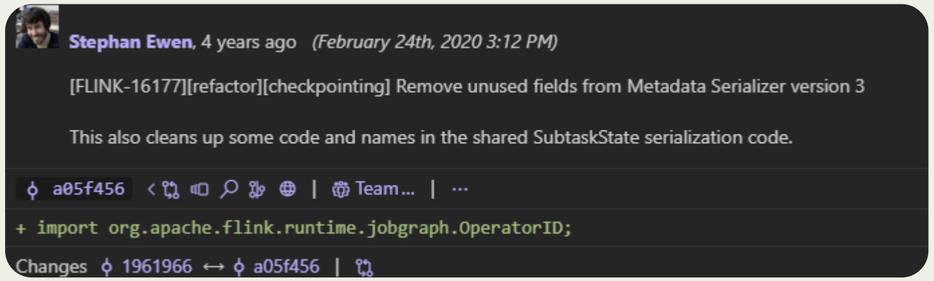


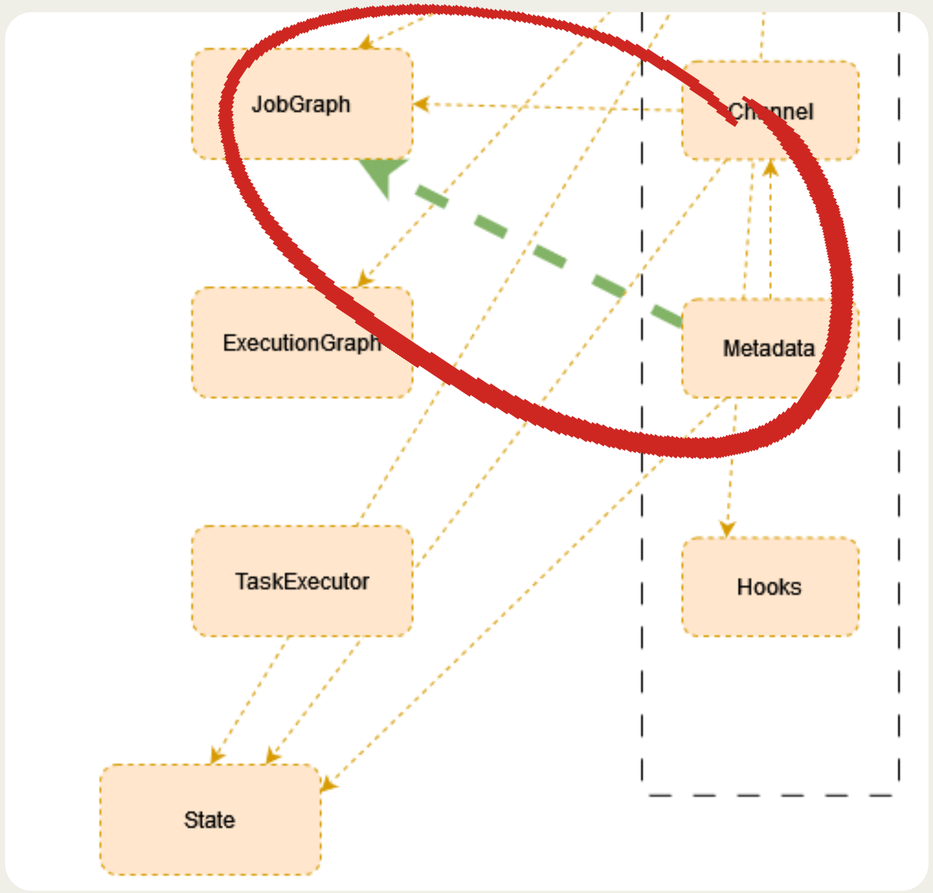
Checkpointing





Divergence





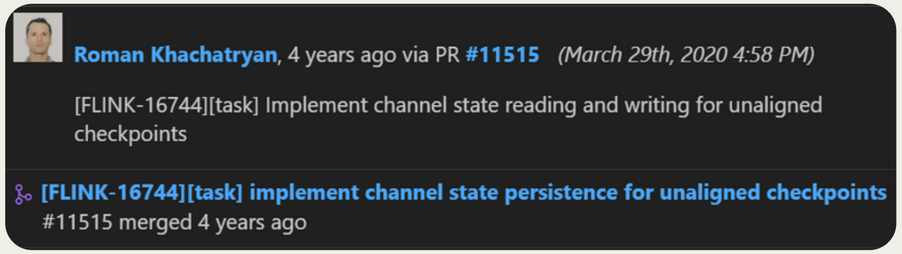

```
// ------------------------------------------------------------------------
//  version-specific serialization
// ------------------------------------------------------------------------

@Override
protected void serializeOperatorState(OperatorState operatorState, DataOutputStream dos) ···

@Override
protected OperatorState deserializeOperatorState( ···
```

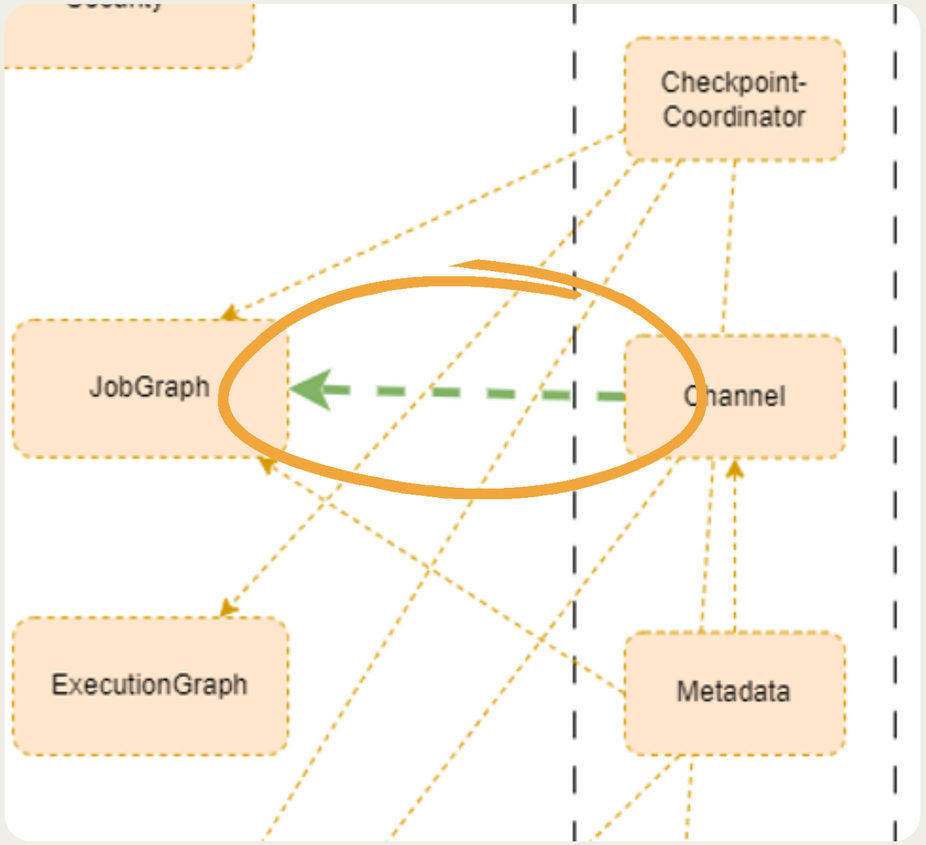
Which	Package org.apache.flink.runtime.**checkpoint.metadata** **Depends on** Package org.apache.flink.runtime.**jobgraph.OperatorID**
Who	Committer/Merger: **Stephen Ewan** (regular contributor) Reviewer: **Jiangjie (Becket) Qin** (regular contributor)
When	PR [FLINK-16177] committed on March 5, 2020 https://github.com/apache/flink/pull/11274
Why	Integration of Operator with checkpointing triggering & committing as operators are also stateful & need to be snapshotted in the checkpoints

```
Stephan Ewen, 4 years ago
package org.apache.flink.runtime.checkpoint.metadata;

import org.apache.flink.runtime.jobgraph.OperatorID; ···

You, 32 seconds ago | 5 authors (Rufus Refactor and others)
/** ···
@Internal
public class MetadataV2Serializer extends MetadataV2V3SerializerBase implements MetadataSerializer {
    //.....................................................................................
    @Override
    protected void serializeOperatorState(OperatorState operatorState, DataOutputStream dos)
            throws IOException {
        checkState(
                !operatorState.isFullyFinished(),
                errorMessage:"Could not support finished Operator state in state serializers.");

        // Operator ID
        dos.writeLong(operatorState.getOperatorID().getLowerPart());
        dos.writeLong(operatorState.getOperatorID().getUpperPart());
        }
    //.....................................................................................

    @Override
    protected OperatorState deserializeOperatorState(
            DataInputStream dis, @Nullable DeserializationContext context) throws IOException {
        final OperatorID jobVertexId = new OperatorID(dis.readLong(), dis.readLong());
        final int parallelism = dis.readInt();
        final int maxParallelism = dis.readInt();
    //.....................................................................................
    }
```

Loading...

Divergence

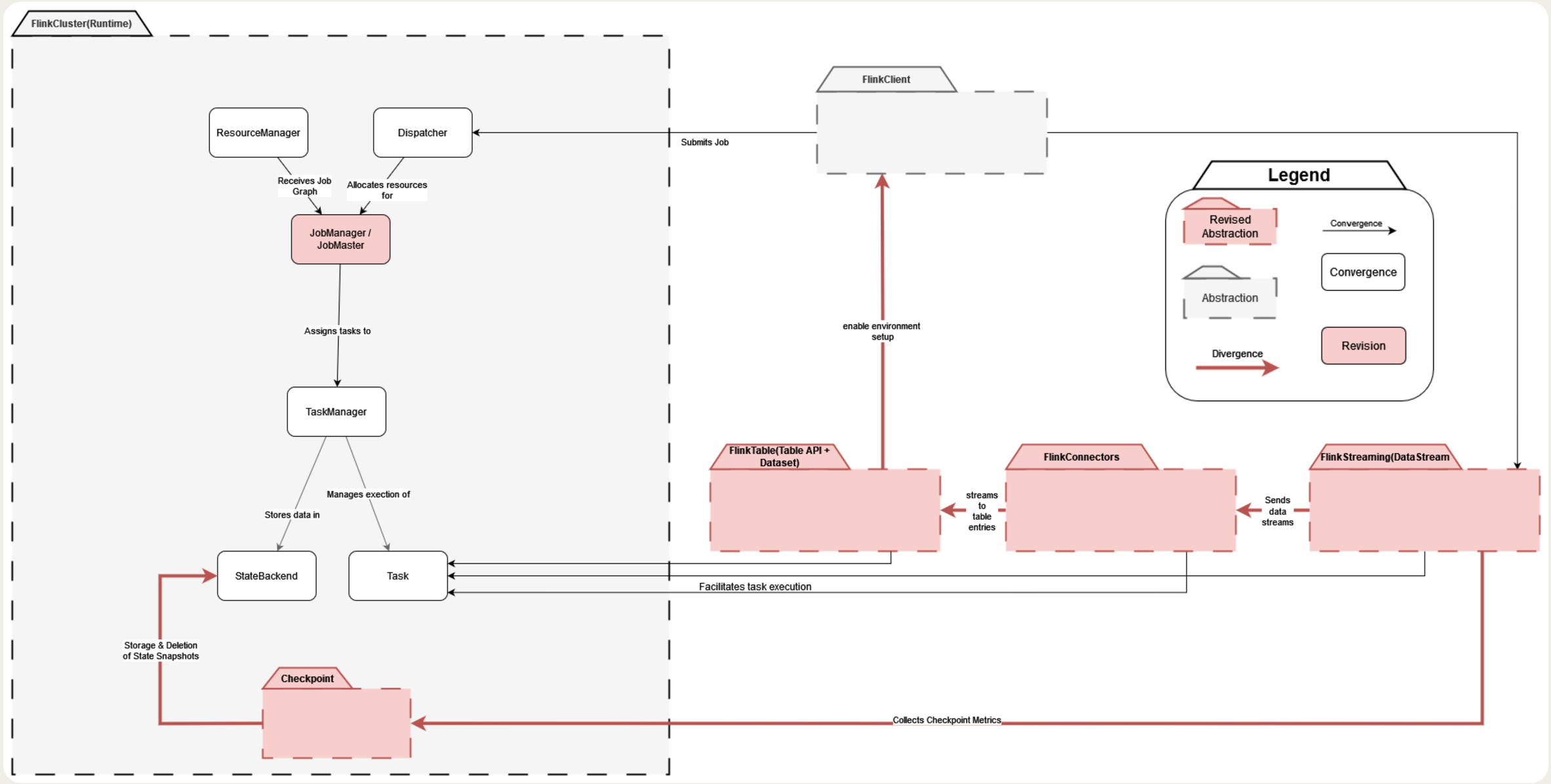




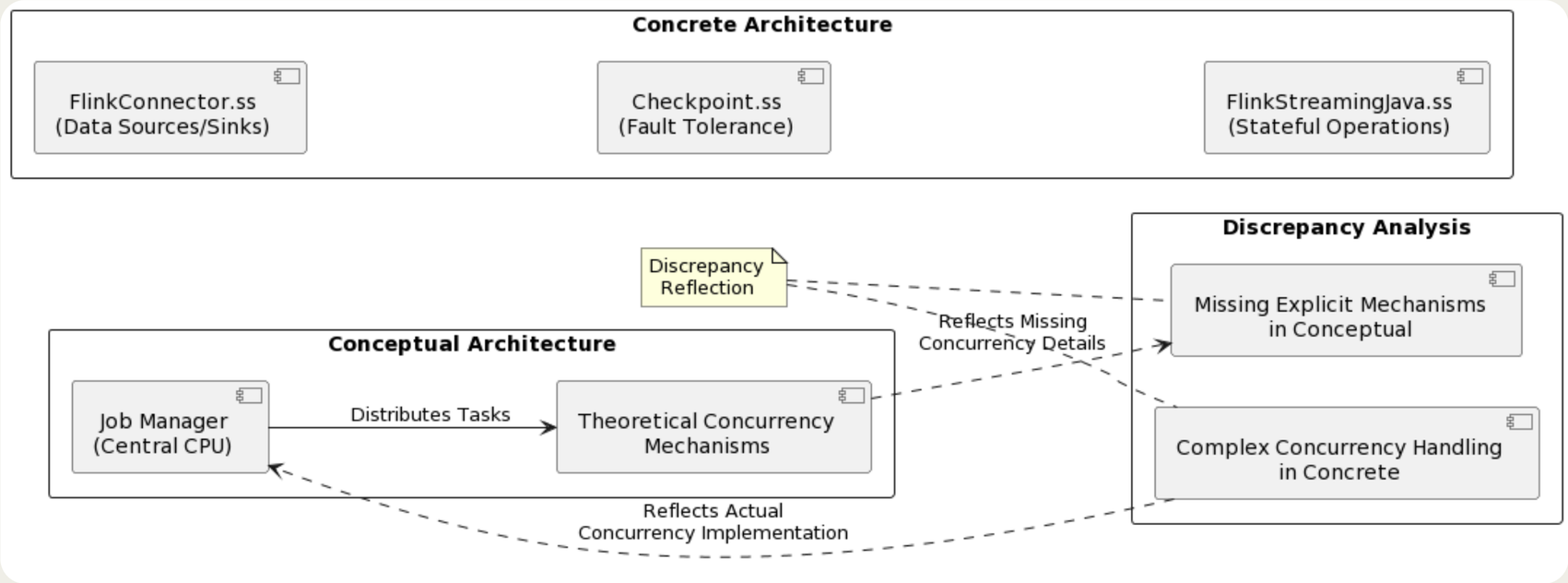

```java
private final JobVertexID jobVertexID;

public ChannelStateWriteRequest(
        JobVertexID jobVertexID, int subtaskIndex, long checkpointId, String name) {
    this.jobVertexID = jobVertexID;
    this.subtaskIndex = subtaskIndex;
    this.checkpointId = checkpointId;
    this.name = name;
}
```

Which	Package org.apache.flink.runtime.**checkpoint.channel** **Depends on** Package org.apache.flink.runtime.**jobgraph.JobVertexID**
Who	Committer: **Roman Khachatryan** Reviewer: **zhijiang**
When	PR [FLINK-16744] commited on March 29, 2020
Why	Implement channel state persistence for unaligned checkpoints. Unaligned checkpoints are a feature in Flink that allows tasks to continue processing while a checkpoint is ongoing.

Revised Conceptual Architecture



Concurrency Aspects



In conceptual architecture:

- Anticipated with mechanisms like Watermarks for event time management.
- Job Manager for task assignment, mimicking CPU-core relationships.

In concrete architecture:

- Implements concurrency via subsystems like FlinkStreamingJava.ss for stateful operations.
- Checkpoint.ss for fault tolerance and state management during failures.

Discrepancies identified:

- Conceptual abstracts concurrency; concrete has explicit mechanisms like checkpointing.
- Complex concurrency handling with subsystems like TaskManager.ss and TaskExecutor.ss.

Team Issues

Checkpoint Contributors:

- Roman Khachatryan
- Rui Fan
- Anton Kalashnikov
- Stephen Ewan
- Till Rohrmann
- Eron Wright
- Chesnay Schepler

Checkpoint Reviewers:

- Robert Metzger
- Jiangjie (Becket) Qin
- Till Rohrmann
- Stephen Ewan
- Yanfei Lei
- Stefan Richter
- Aljoscha Krettek

Team Issues

- Integration testing for pull/merge requests
- Tracking dependencies introduced in commits
- Tracking authorship of code
- Coordinating proper review and revision of contributions
- Documenting thousands of commits & pull requests; Confluence is used for this
- Many are Confluent employees & have other work/projects with each other

Limitations

1. Pull requests documented incorrectly → i.e., introduced a dependency but did not mention, confusing pull request conversation between contributors and reviewers
2. Some subsystems have very complicated directory structures → difficult to parse/read
3. It is possible that not all dependencies are mapped as there are thousands of files & even more dependency

Lessons Learned

1. Runtime concrete architecture diverged a lot from the conceptual
2. Divergences were much more common than absences
3. GitKraken and GitLens are great tools for efficiently analyzing authorship/blame
4. Reading the conversations in commit logs and Jira issues as well as tracing the associated identifier codes helps us understand design decisions

Q&A