Apache Flink Concrete Architecture

Team FlinkForce

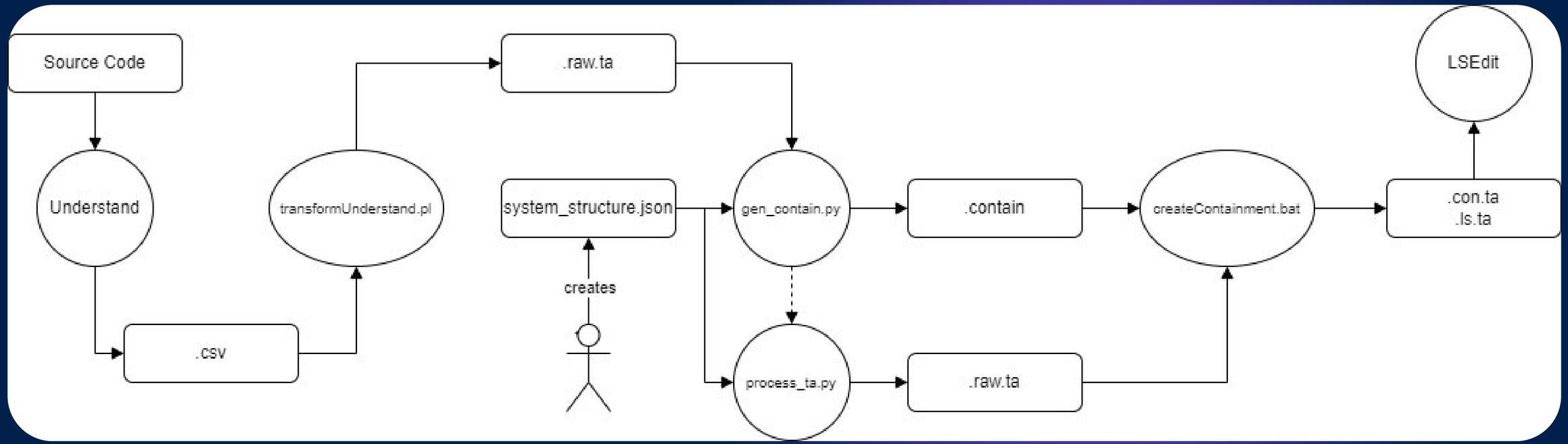
Checkpoint Subsystem

Agenda

1. Concrete Architecture Derivation & Extraction
2. Extracted Concrete Architecture from LSEdit
3. Subsystem Interactions
4. Checkpoint Subsystem
5. Conceptual Architecture Recap
6. Conceptual vs. Concrete Architecture - Reflexion
7. Architectural Styles & Design Patterns
8. Use Cases
9. Limitations & Lessons Learned
10. Extraction Demo

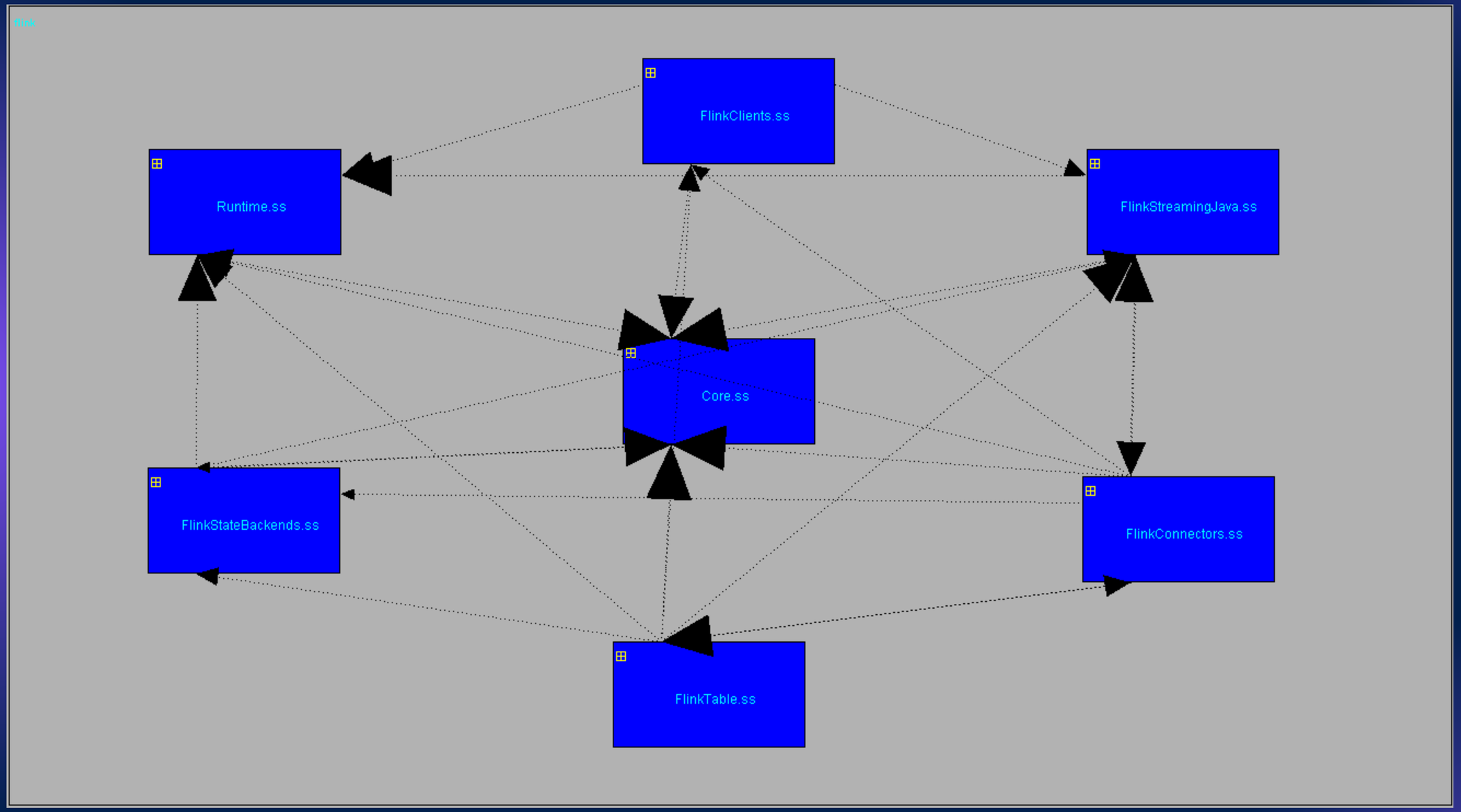
Concrete Architecture – Derivation & Extraction

Extraction Process Flow



Extracted Concrete Architecture from LSEdit

Top Level View of Concrete Architecture



Functionalities & Subsystem Interactions

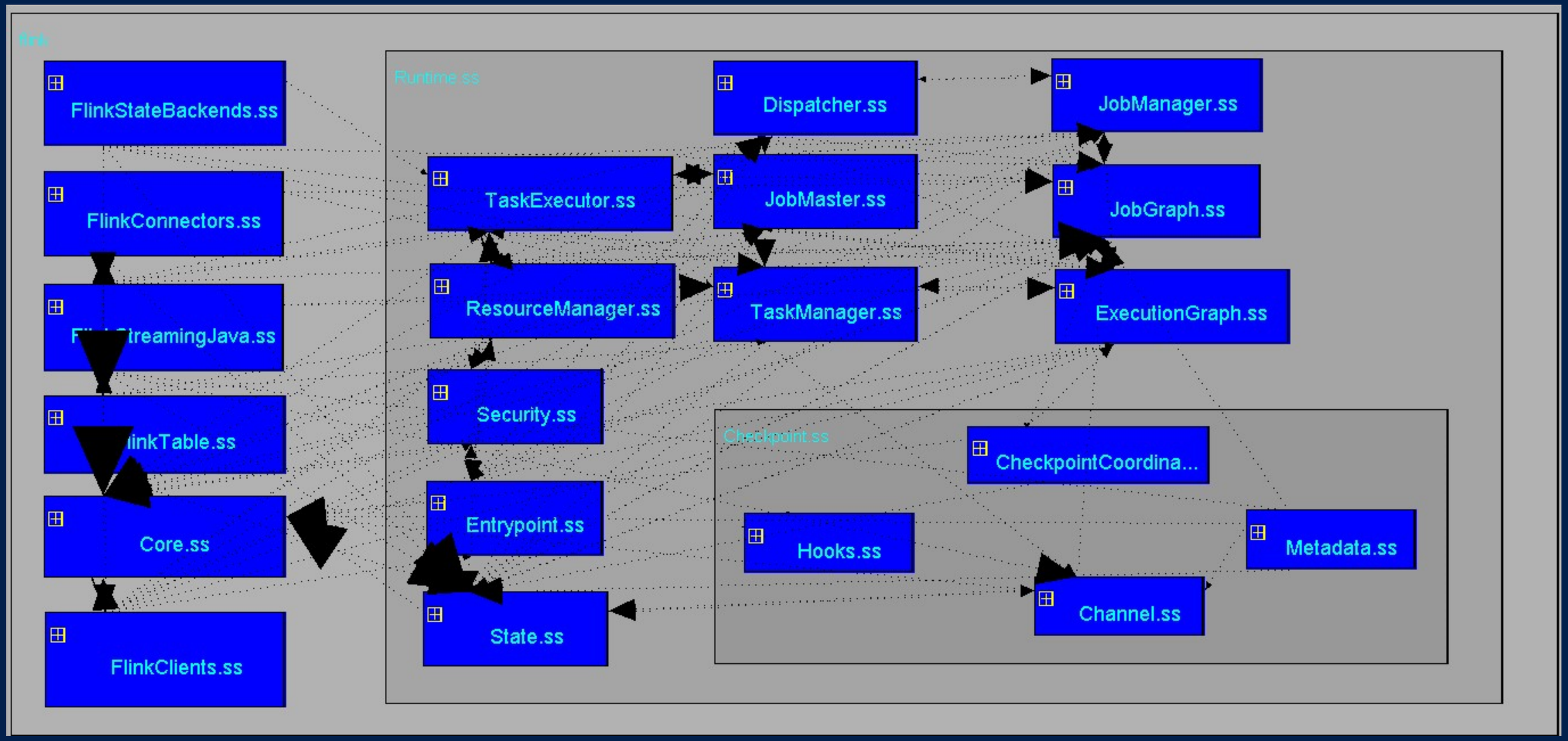
Subsystem	Function	Interactions
RunTime.s	Task Scheduling, source allocation, parallelism management, checkpoint management, and dynamic scaling.	Interacts with Checkpoint to ensure fault tolerance and data integrity. Inserts “Barriers” to prevent data processing beyond a certain point.
FlinkClients.ss	Job submission, job control, cluster configuration, cluster interaction, result retrieval, exception handling, job management, and security features.	FlinkClient uses Checkpoint by allowing users to checkpoint parameters during job submission, monitoring job status, and controlling the job’s lifecycle.
Core.ss	Data processing, Batch processing, filesystem, read and write memory, and security.	Checkpoint subsystem guarantees fault tolerance and data integrity for batch processing and data stream processing.
Flink Connectors	Data ingestion, data output, data forma transformation, support for fault tolerance, exactly-once semantics, and simplify data movements.	Connectors coordinate with checkpoint to ensure the data ingested is consistent and fault-tolerant. Checkpoint is used to prevent data loss as well.
FlinkTable	data transformation, task analysis through SQL-Like queries.	To support exactly-once semantics and allows users to process data with high-level declarative queries while maintaining fault tolerance.

Key Functionalities & Interactions

Checkpoint Subsystem

- Key component for ensuring fault tolerance
- States are “1st class citizens” & checkpoints are “butlers” that guarantee this citizenry
- Failures trigger stream processing to halt, snapshots get taken & then system resumes

Looking Inside Checkpoint



Checkpoints Under the Hood

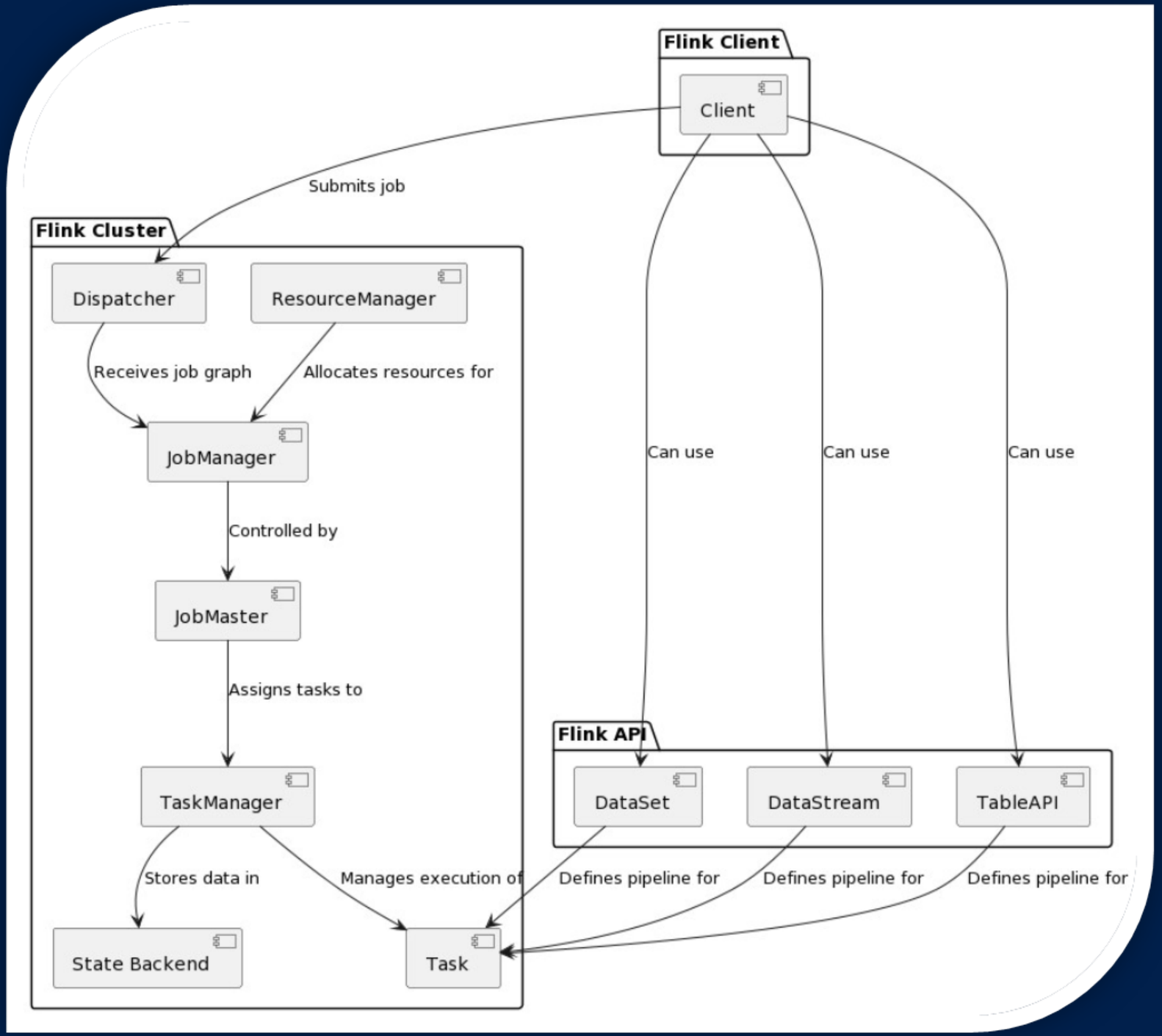
Subsubsytem	Function	Interactions
Checkpoint Coordinator	Distributed snapshotting, fault state → halt processing, snapshot then resume	JobManager, TaskManager, JobMaster, StateBackend
Metadata	Accessory data for storage, location, status & (de)serialization of checkpts.	CheckpointCoordinator, JobManager, TaskManager
Hooks	Allows for custom actions associated with checkpts. begin triggered	CheckpointCoordinator, TaskManager, TaskExecutor
Channel	Snapshot of data in transit & sender/receiver to resume streams	CheckpointCoordinator, State, TaskManager

Conceptual Architecture – Recap

Client has option to utilize APIs or runtime functions

Runtime Cluster works through dispatcher and resource manager flow with Job Manager

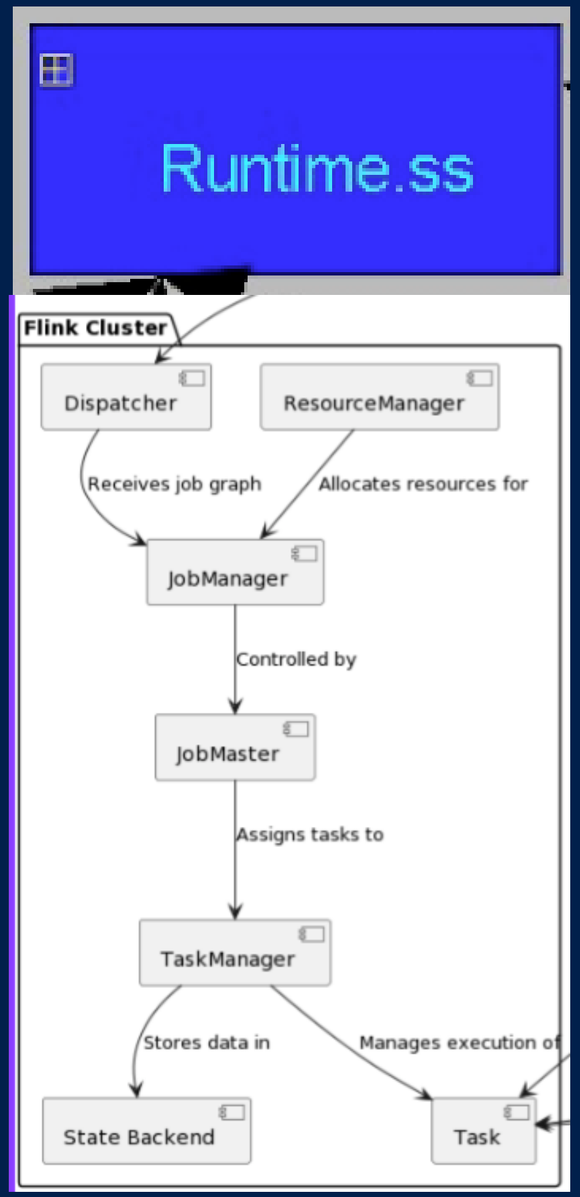
Job Master transfers task flow to Task Manager, which in turn reaches State Backend

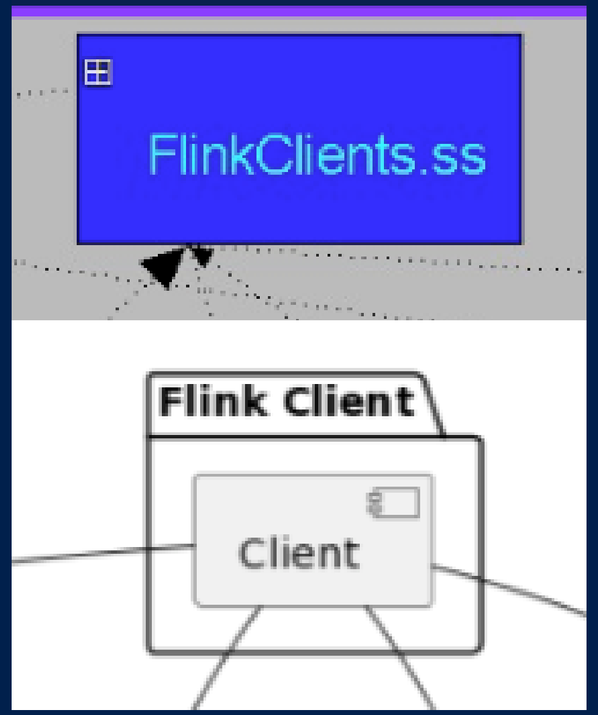


Conceptual Architecture
vs.
Concrete Architecture

Similarities

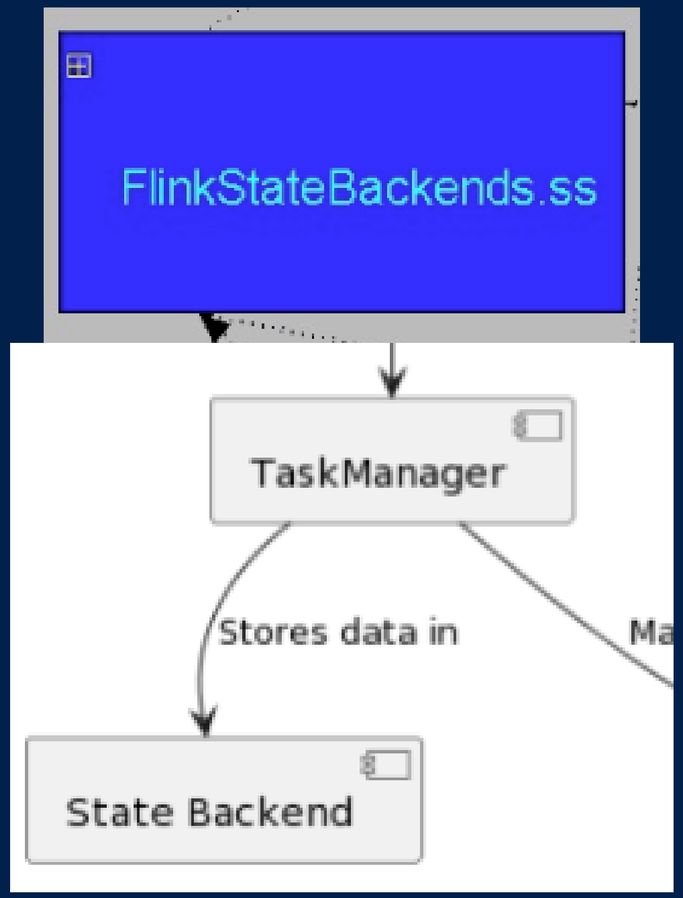
Runtime.ss and Flink Cluster

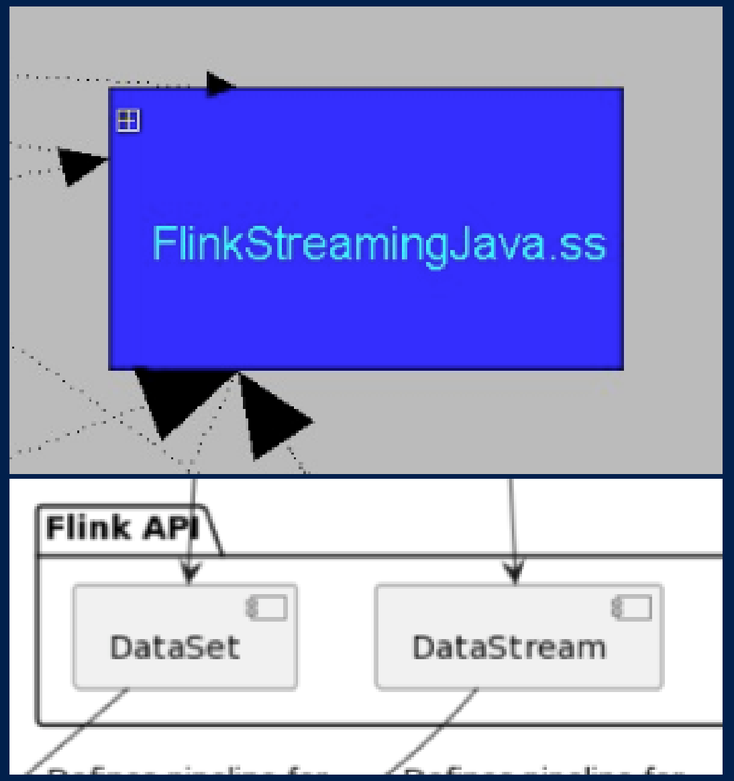




Flink Clients.ss and Flink Client

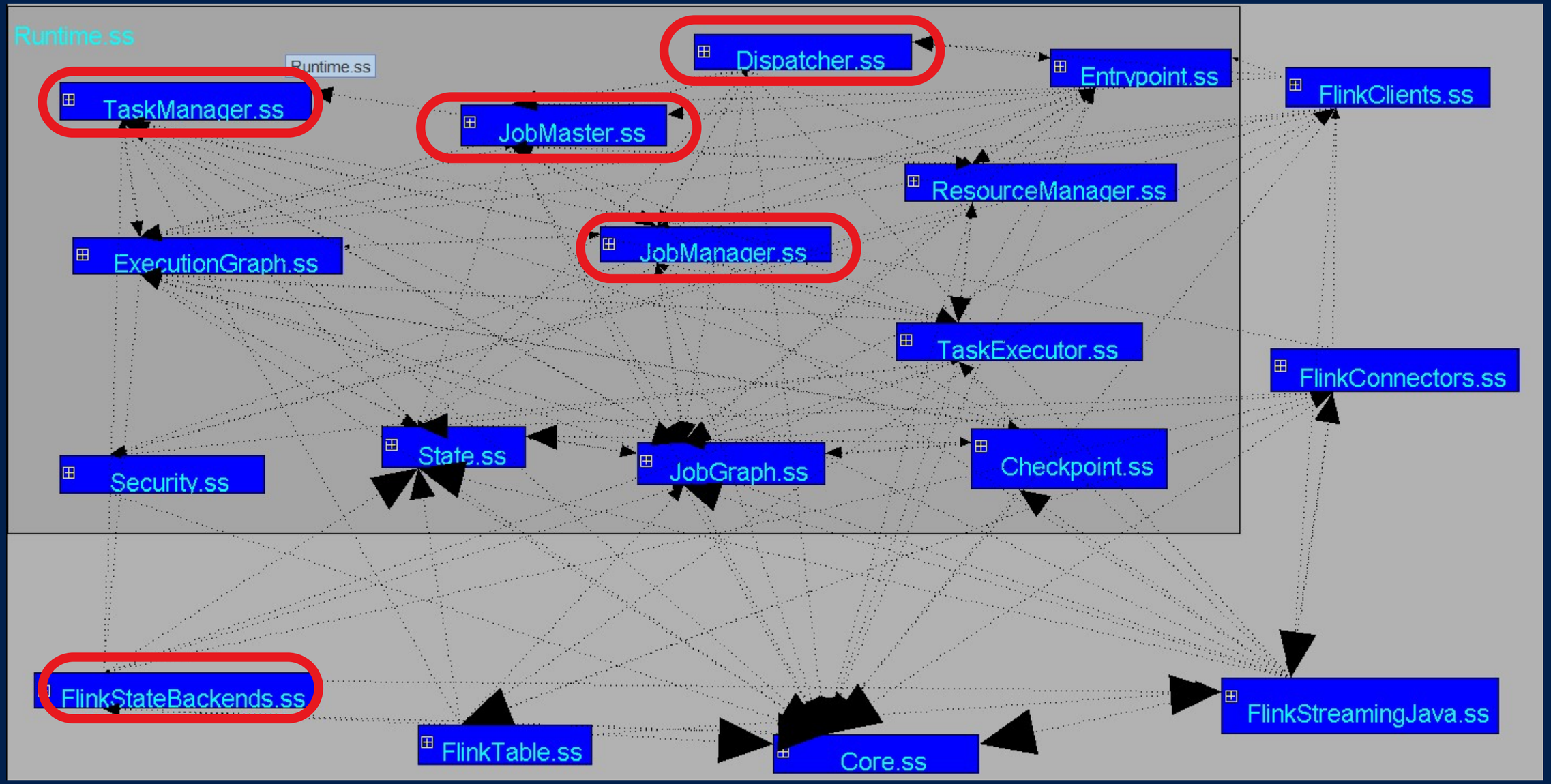
FlinkStateBackends.ss and State Backend





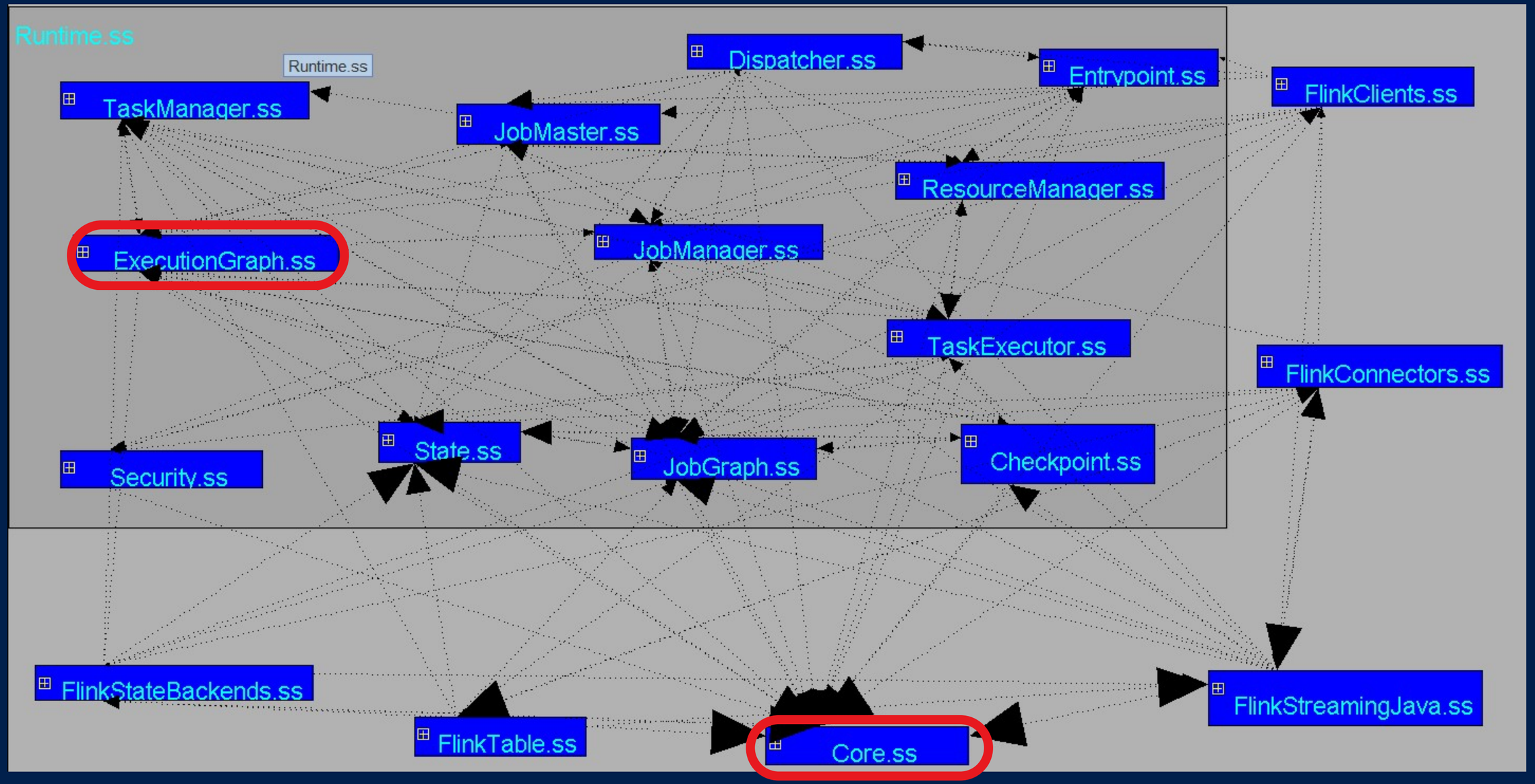
FlinkStreamingJava.ss and Data Stream API

Run Time Sub Components

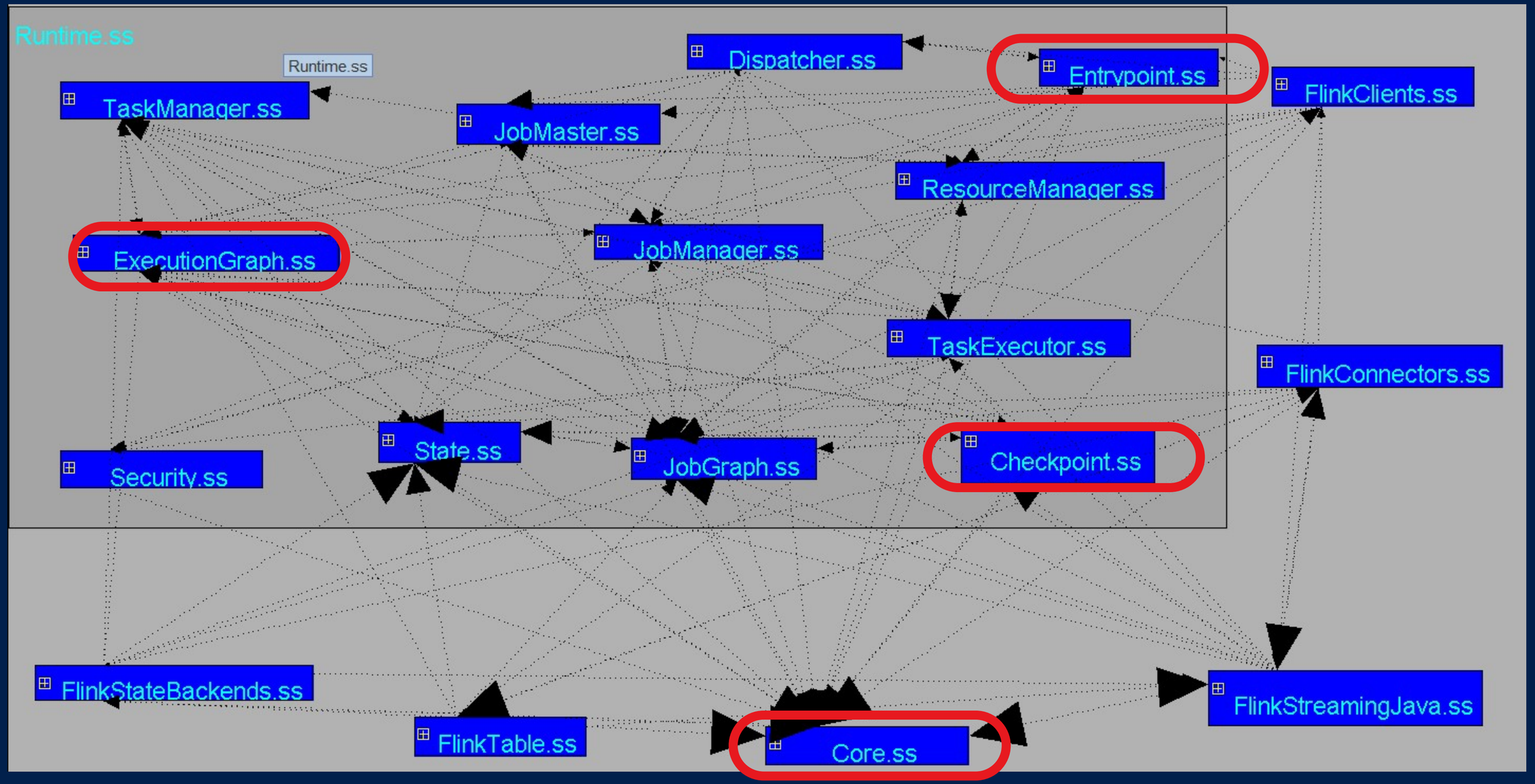


Differences

Absences ExecutionGraph.ss Core.ss

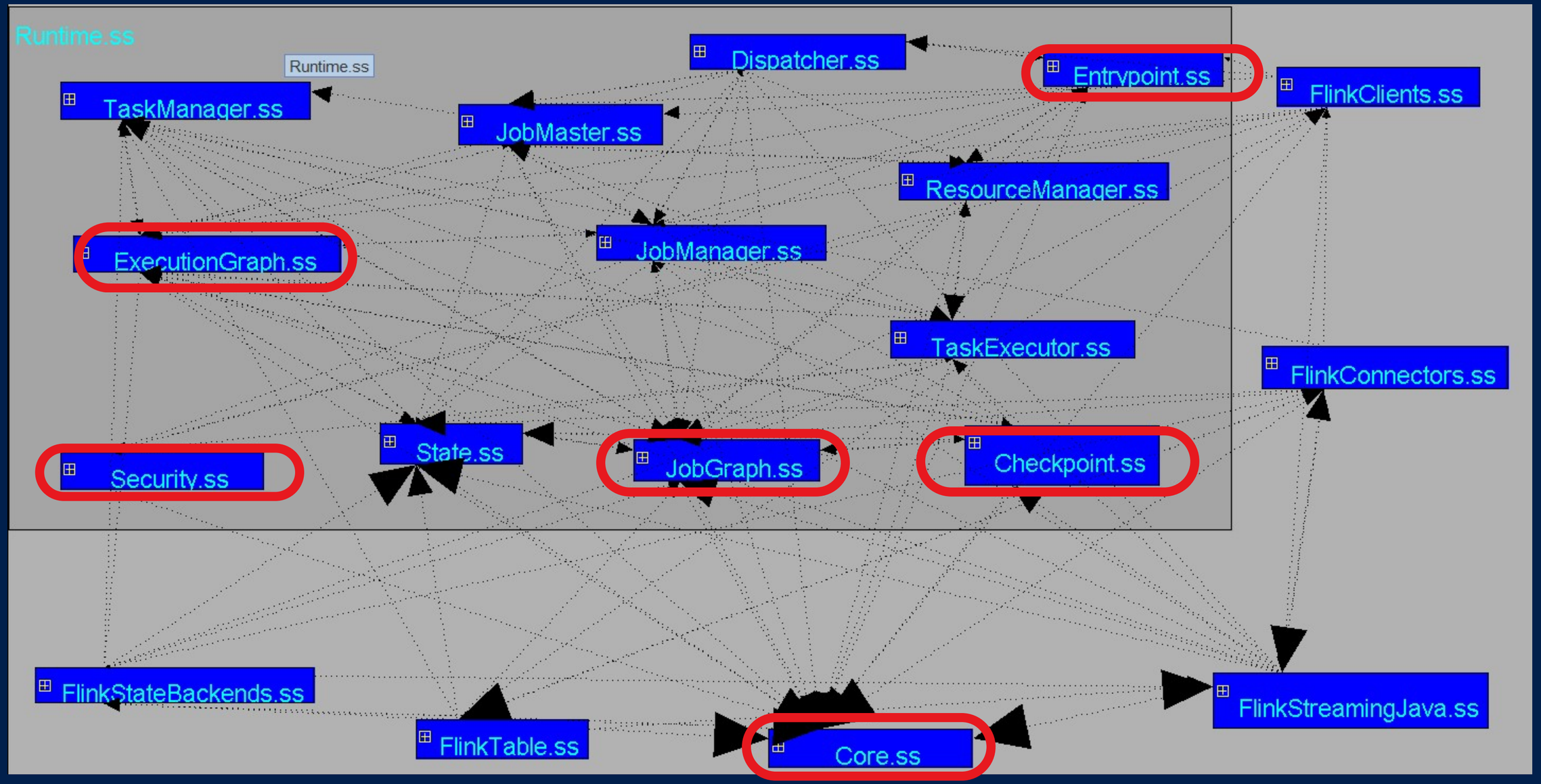


Entrypoint.ss Checkpointing.ss

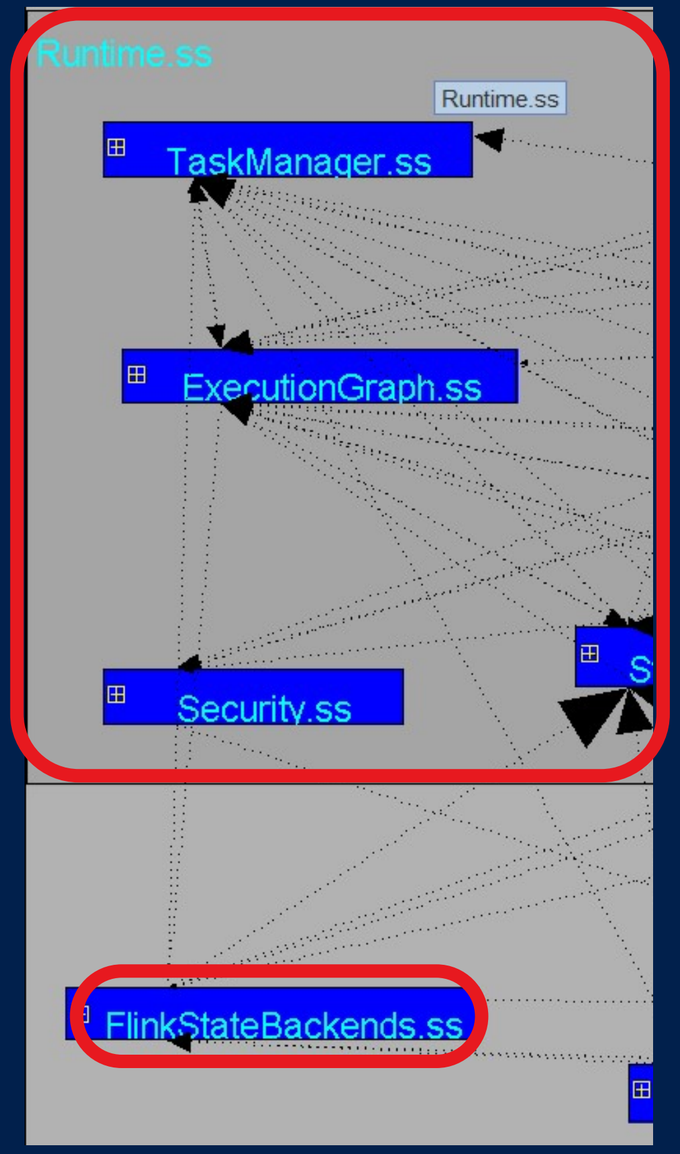


Security.ss

JobGraph.ss

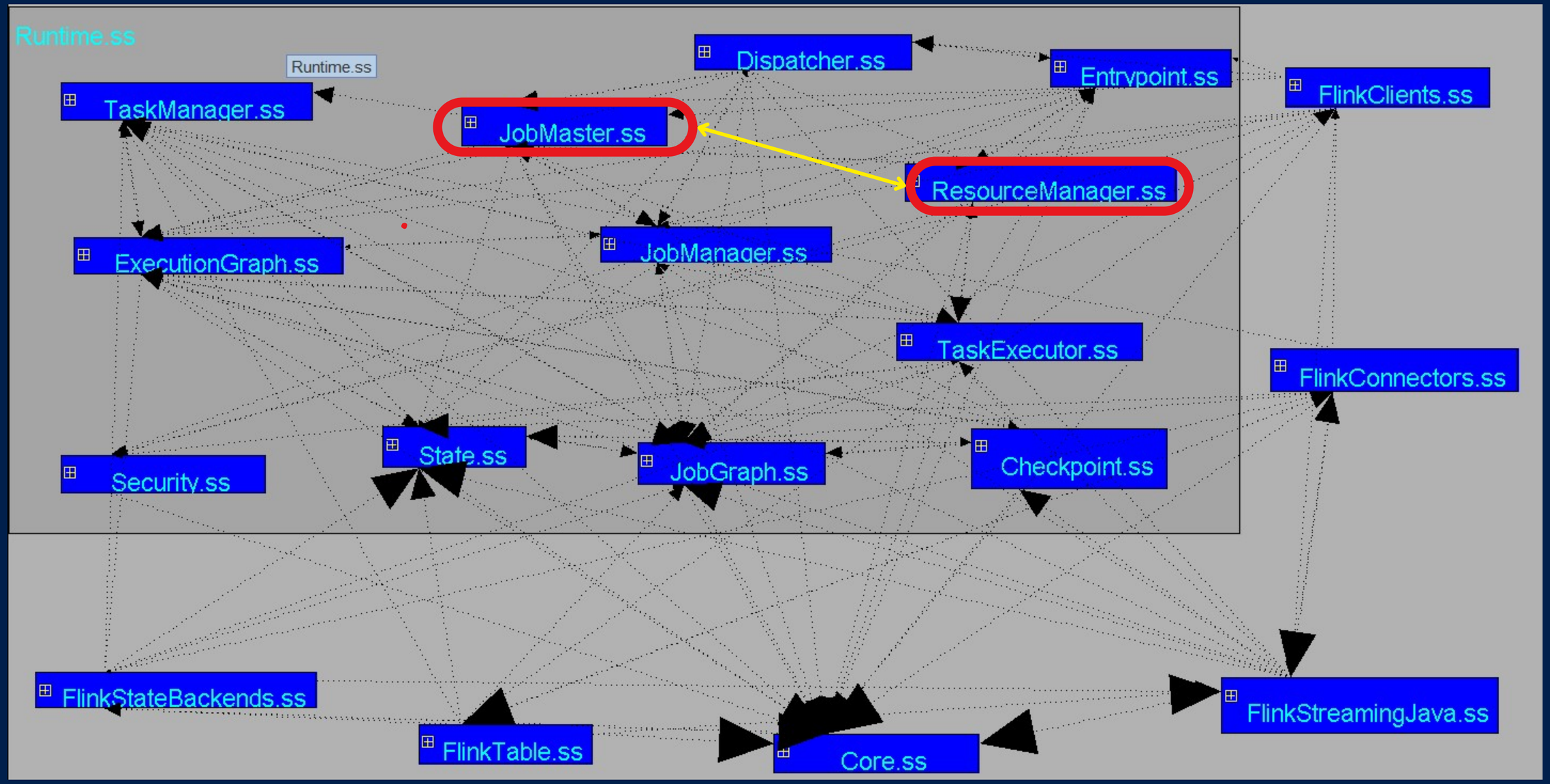


Divergences

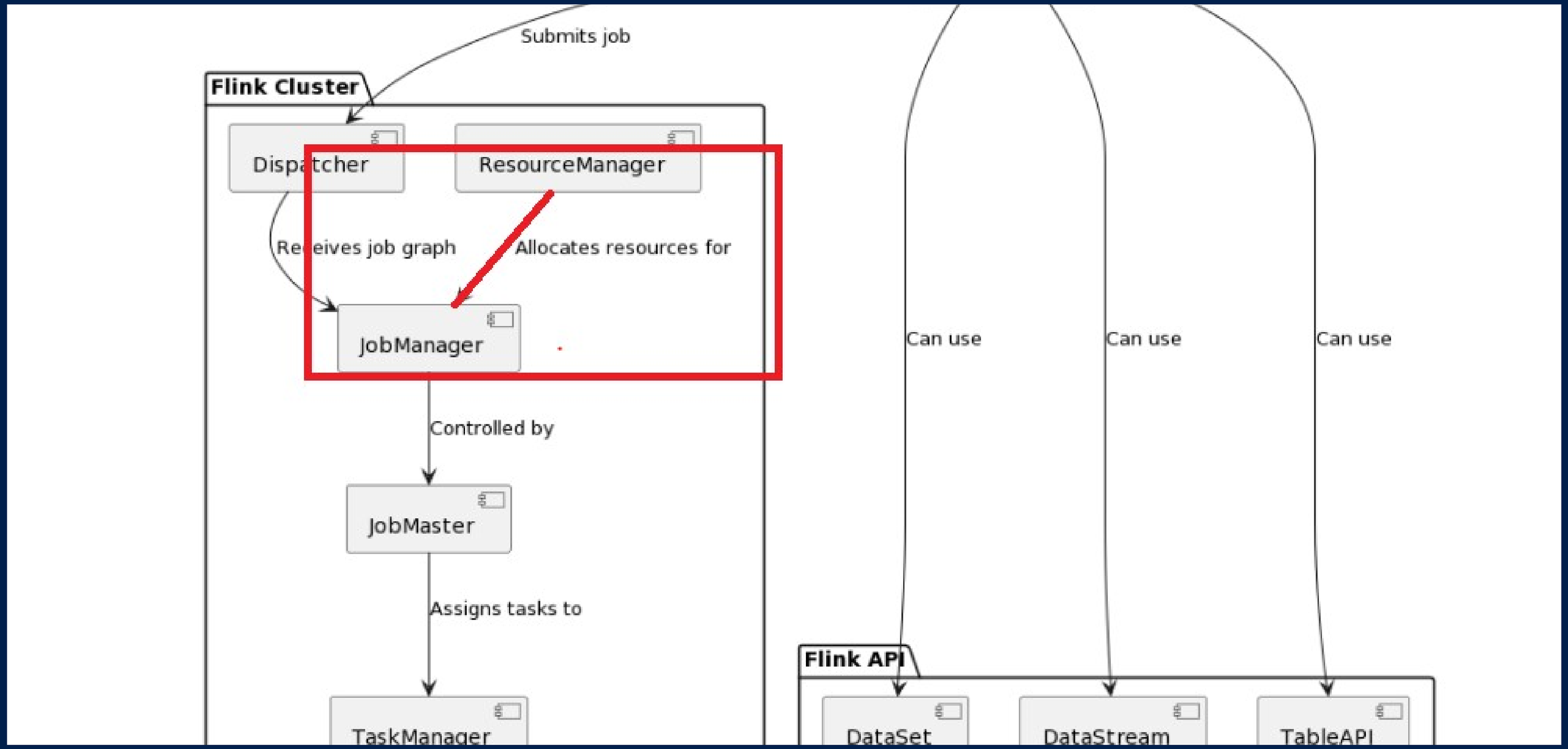


State backend is included in the runtime when it should be outside

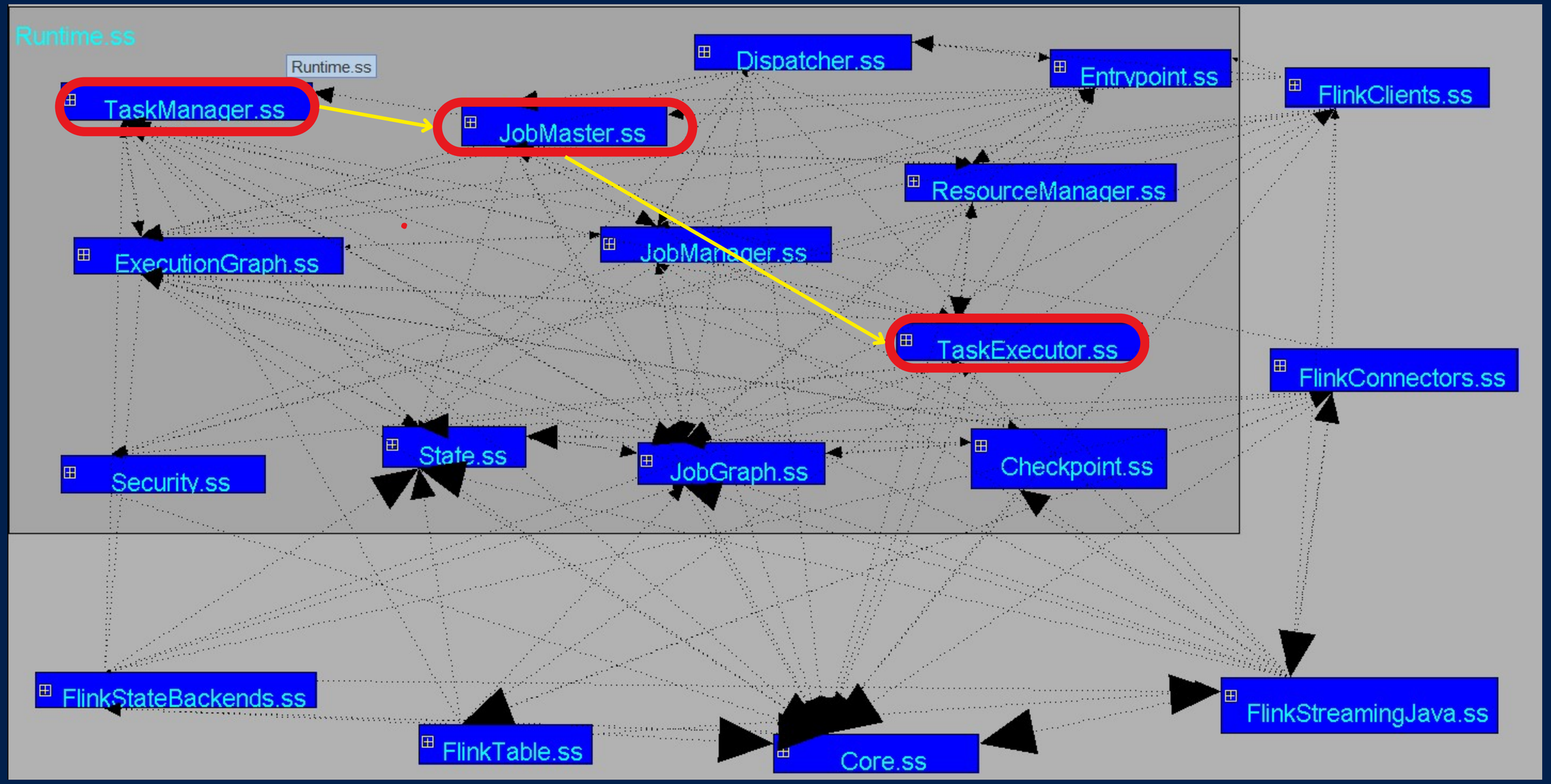
Resource manager interacts with the job master instead of job manager



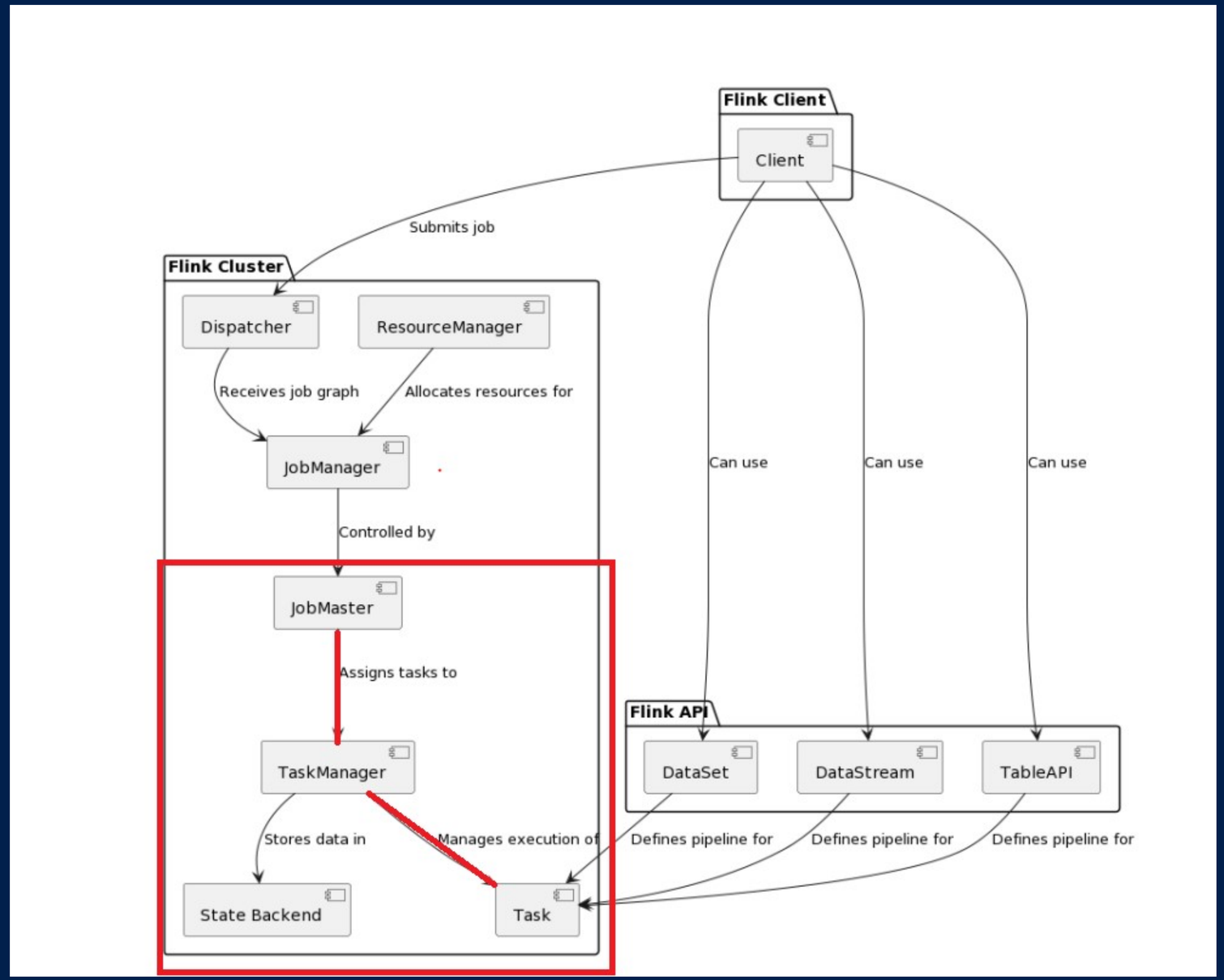
.....In the conceptual



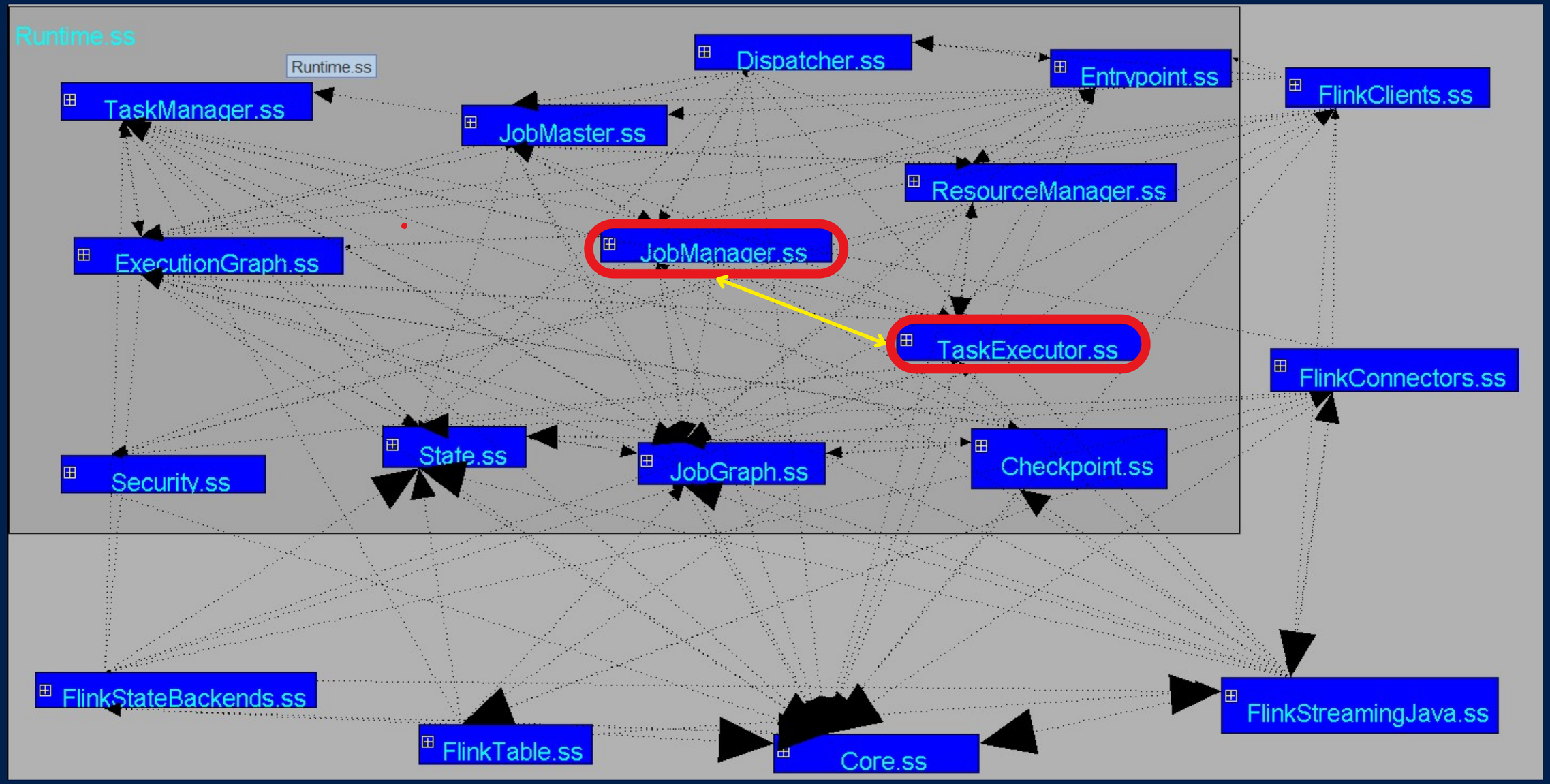
Job master can talk to both task manager and task executor



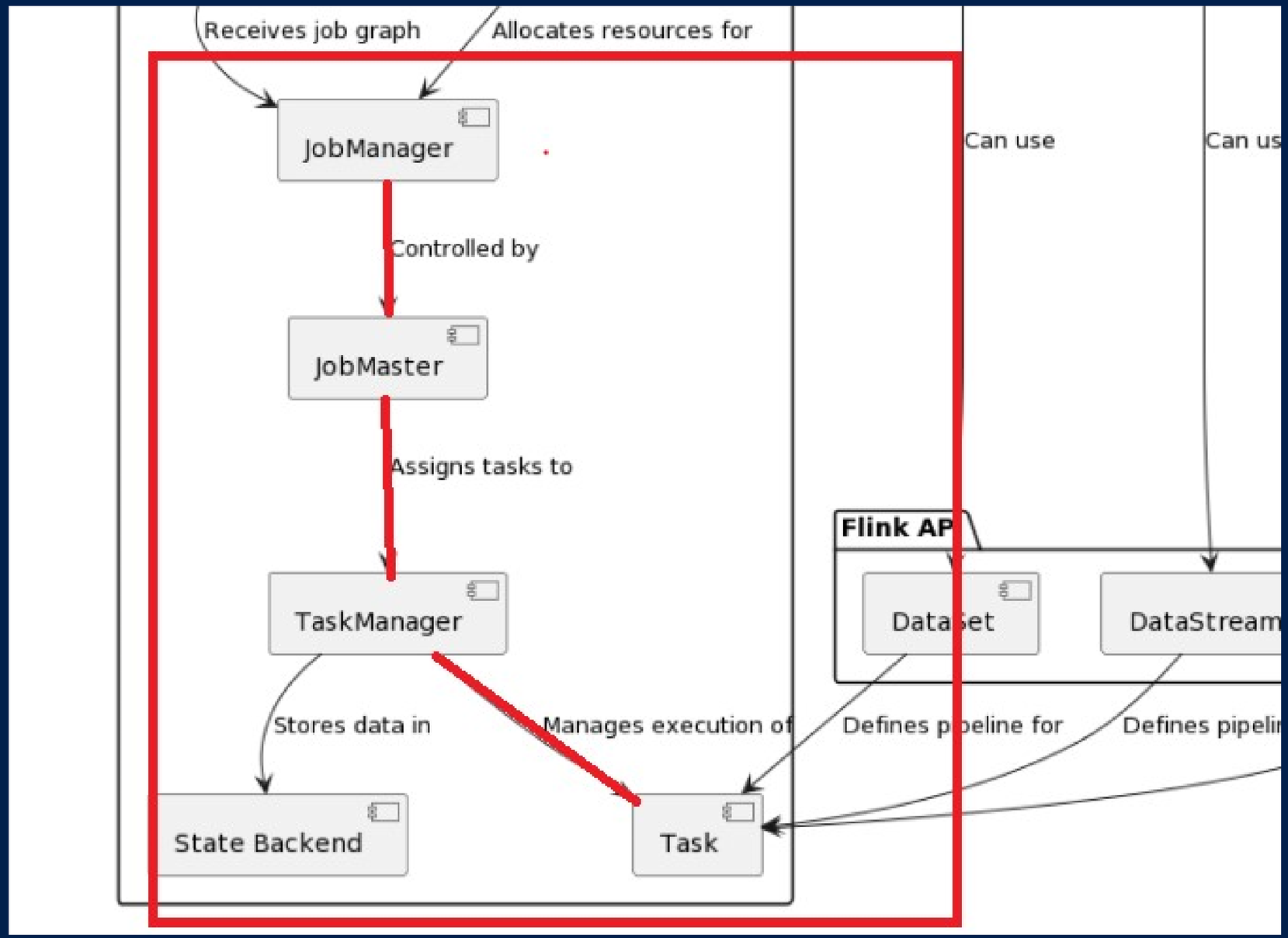
.....In the conceptual



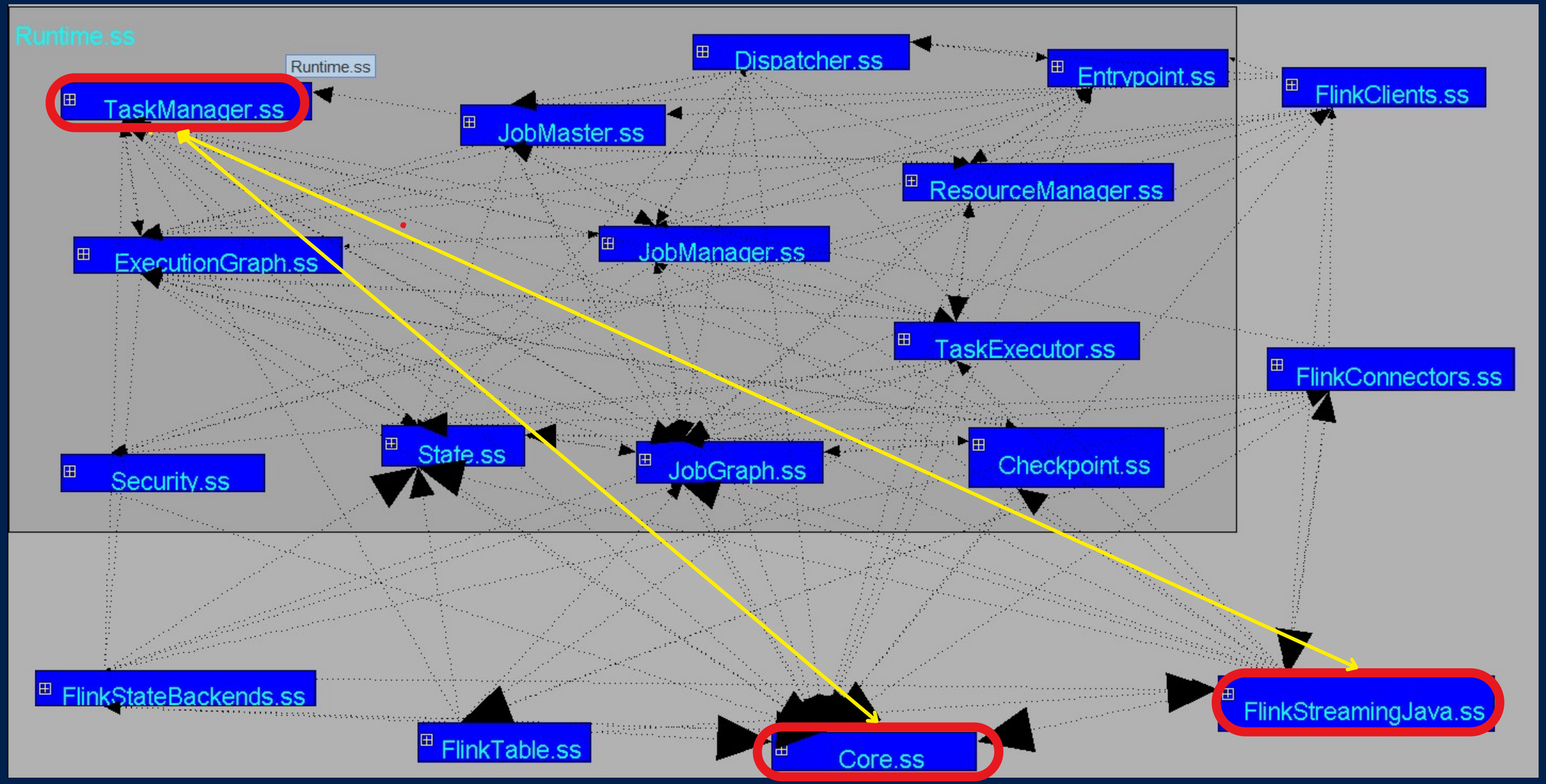
Job manager can talk directly to the task executor



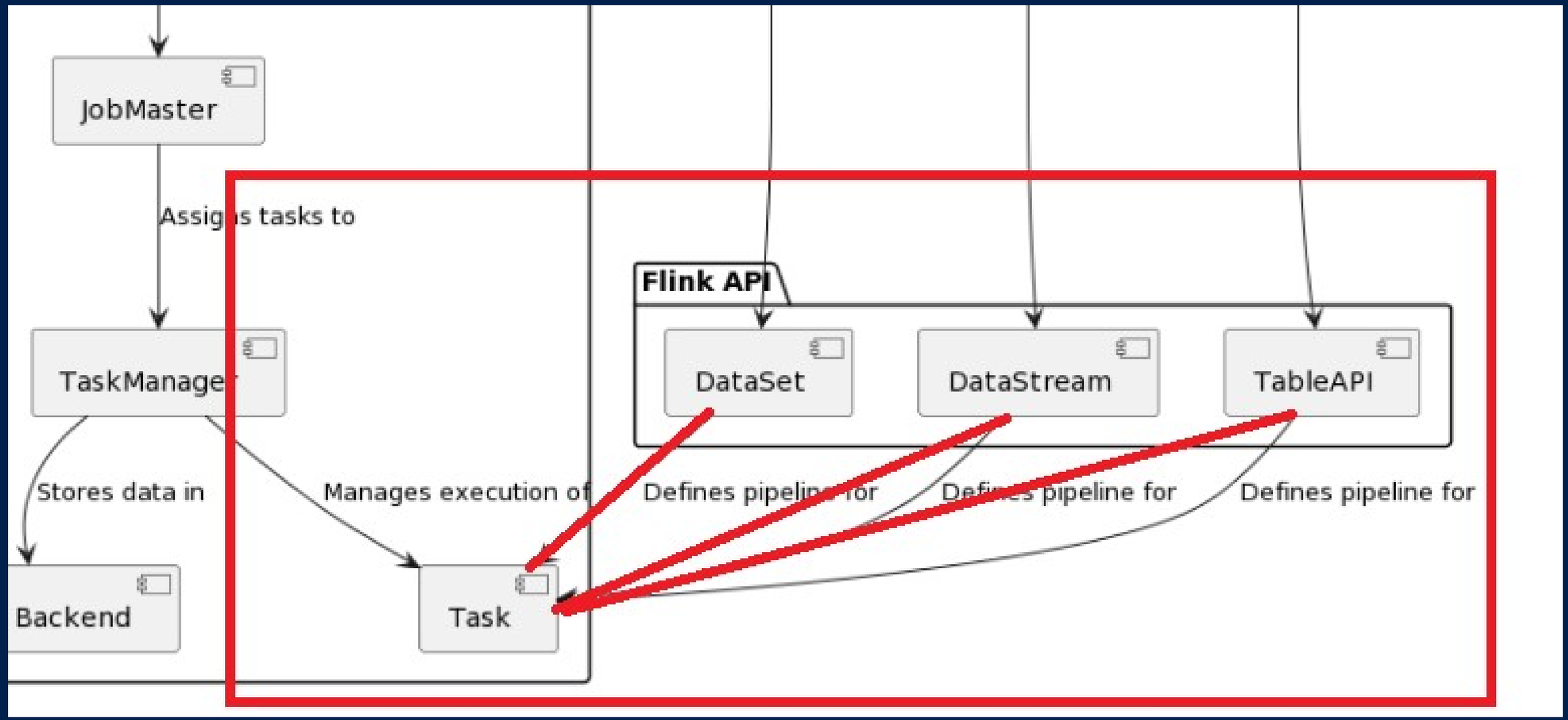
.....In the conceptual



"Flink APIs" do not talk directly with the the task executor, they talk to the task manager



.....In the conceptual



Concurrency

- Concurrency is managed through components like TaskManager.ss and TaskExecutor.ss, they allow parallel task execution on separate threads.

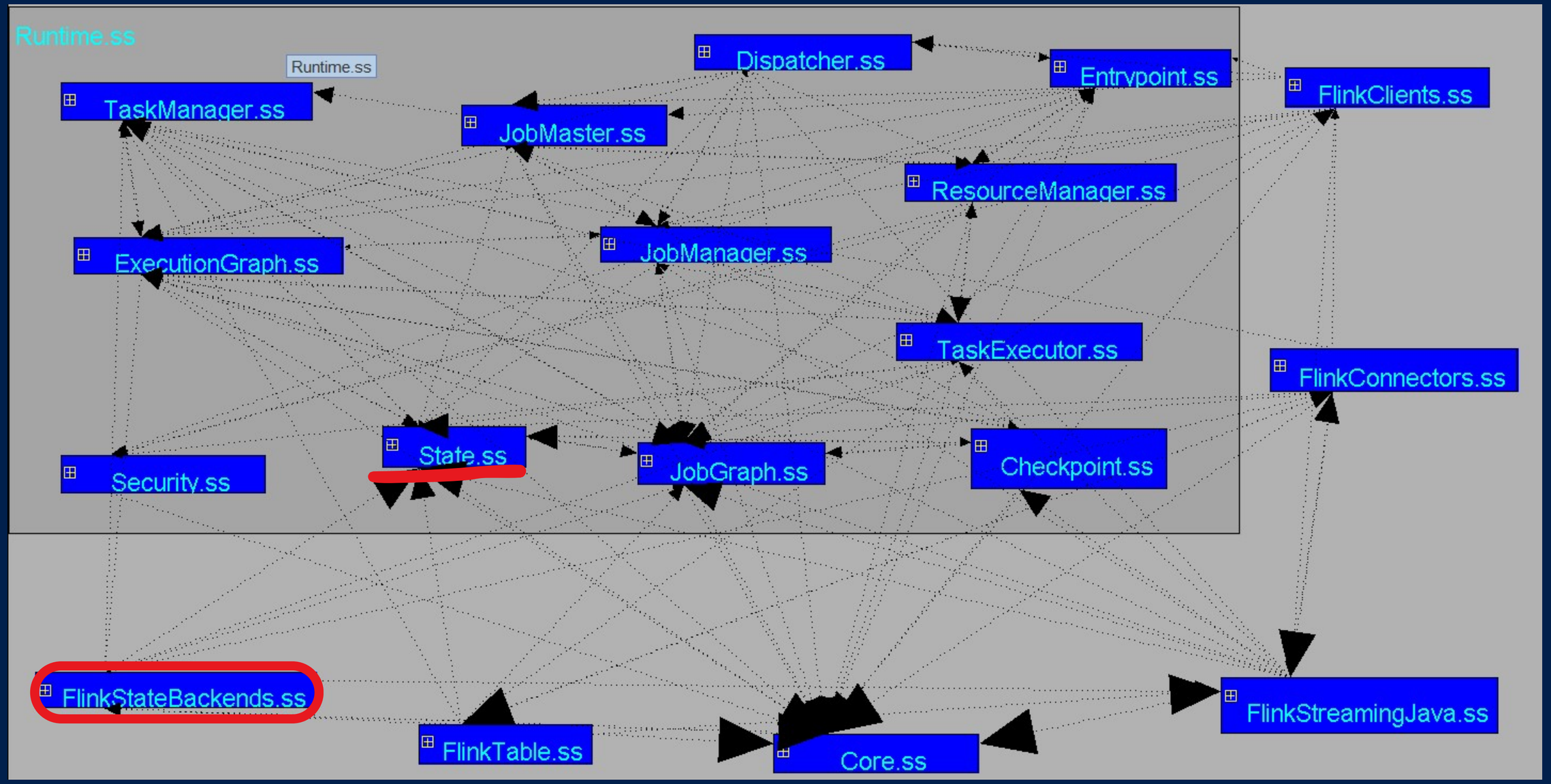
- The system's resilience and fault-tolerance are enhanced by Checkpoint.ss, which periodically saves states from all running tasks, ensuring consistent processing.

- The ExecutionGraph.ss visualizes these concurrent execution pathways, derived from the JobGraph.ss.

- State.ss supports the entire process

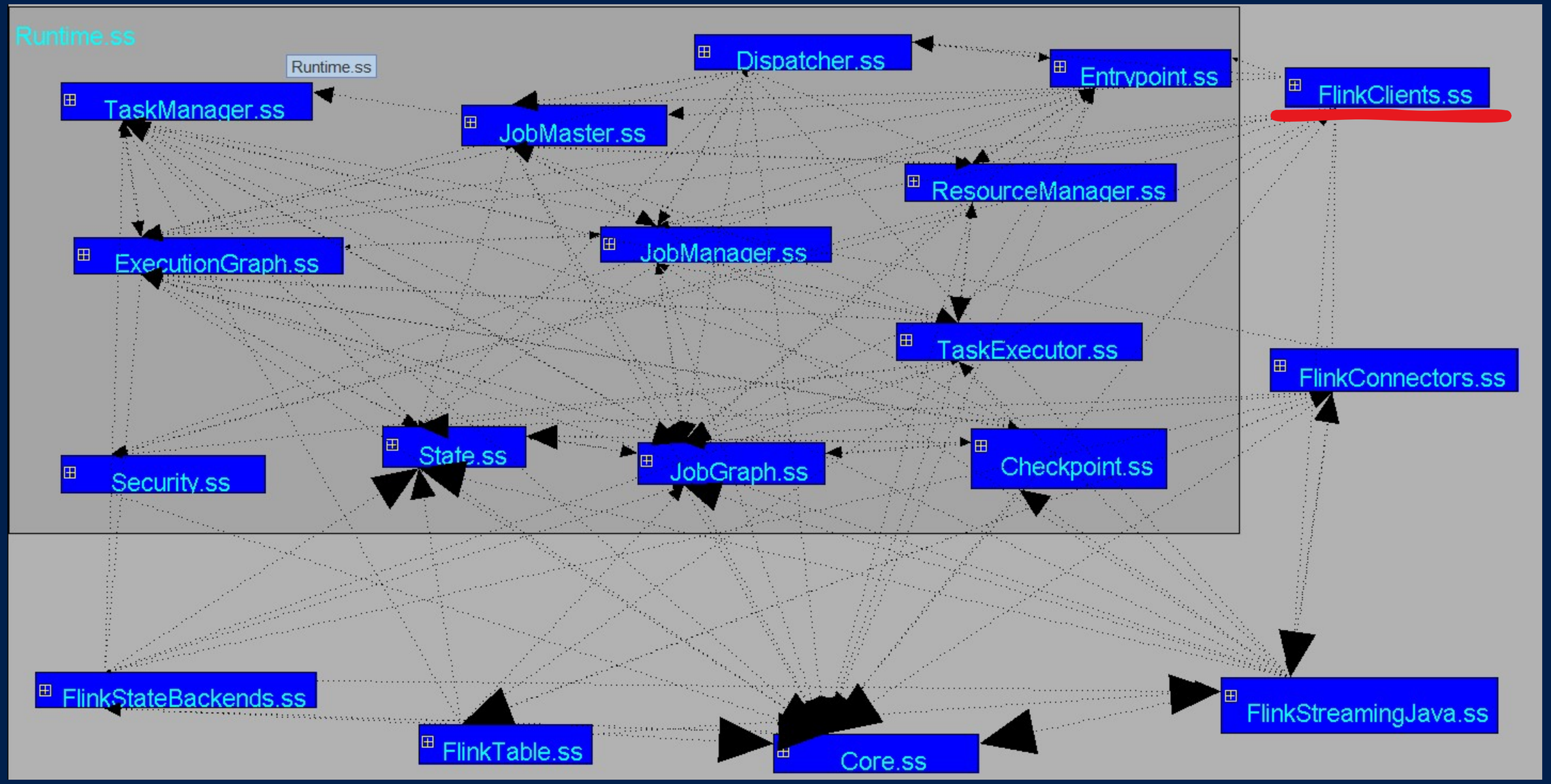
Architectural Styles

Respository



The FlinkStateBackends.ss can be seen as a repository component, where data or states are stored and accessed by other components. It serves as the centralized location for state management.

Pipe & Filter + Layered + Client-Server

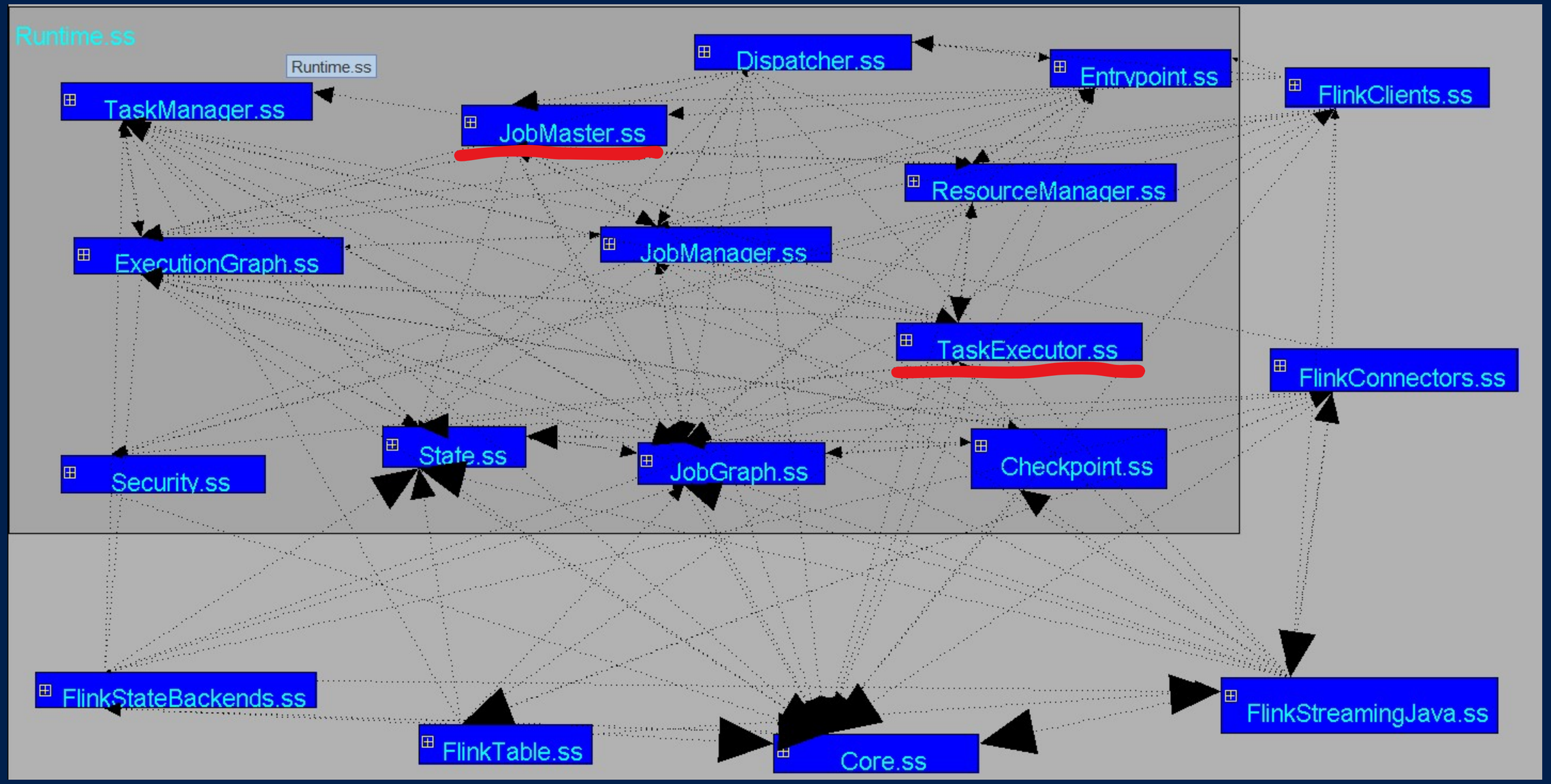


Client components like FlinkClients.ss indicates that there are client components communicating with server components within the system. The server components process requests and serve the clients.

Client components like FlinkClients.ss indicates that there are client components communicating with server components within the system. The server components process requests and serve the clients.

There are distinct layers in the architecture. Components like Entrypoint.ss, FlinkClients.ss, and FlinkStreamingJava.ss can be seen as the top-most layers (or presentation layers) that interface with the outside world or clients. They communicate with internal system components, making the layered style still valid.

Implicit Invocation



This is a style where components publish events without the knowledge of which components, if any, will react. Looking at the diagram, the numerous interconnections and the potential for components to trigger actions in others (especially given the nature of streaming and processing in Flink) suggest an implicit invocation style. Components like JobMaster.ss will trigger certain tasks in TaskExecutor.ss based on events or changes in state.

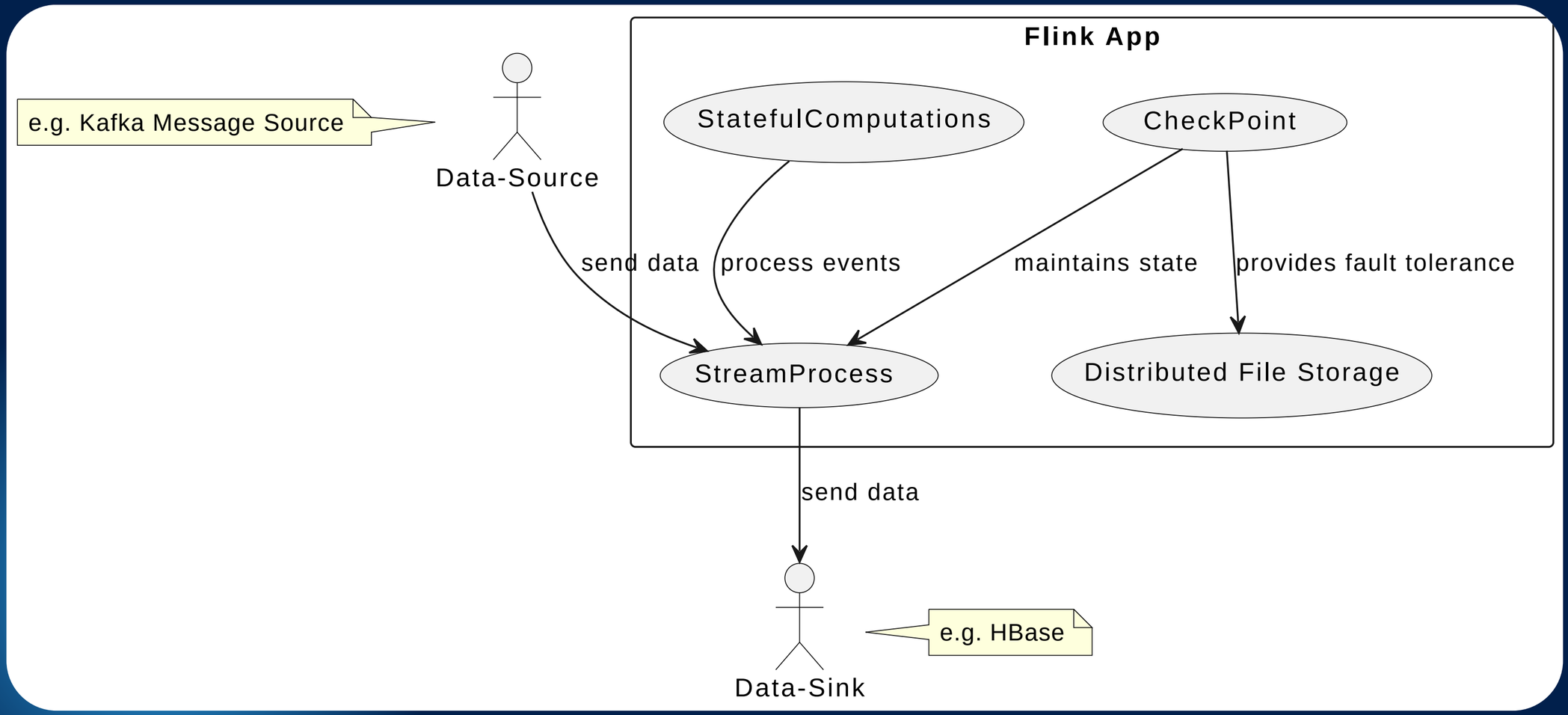
Design Patterns

Factory: used in several classes, such as StreamExecutionEnvironment and Resource Manager instantiation for flexible task management

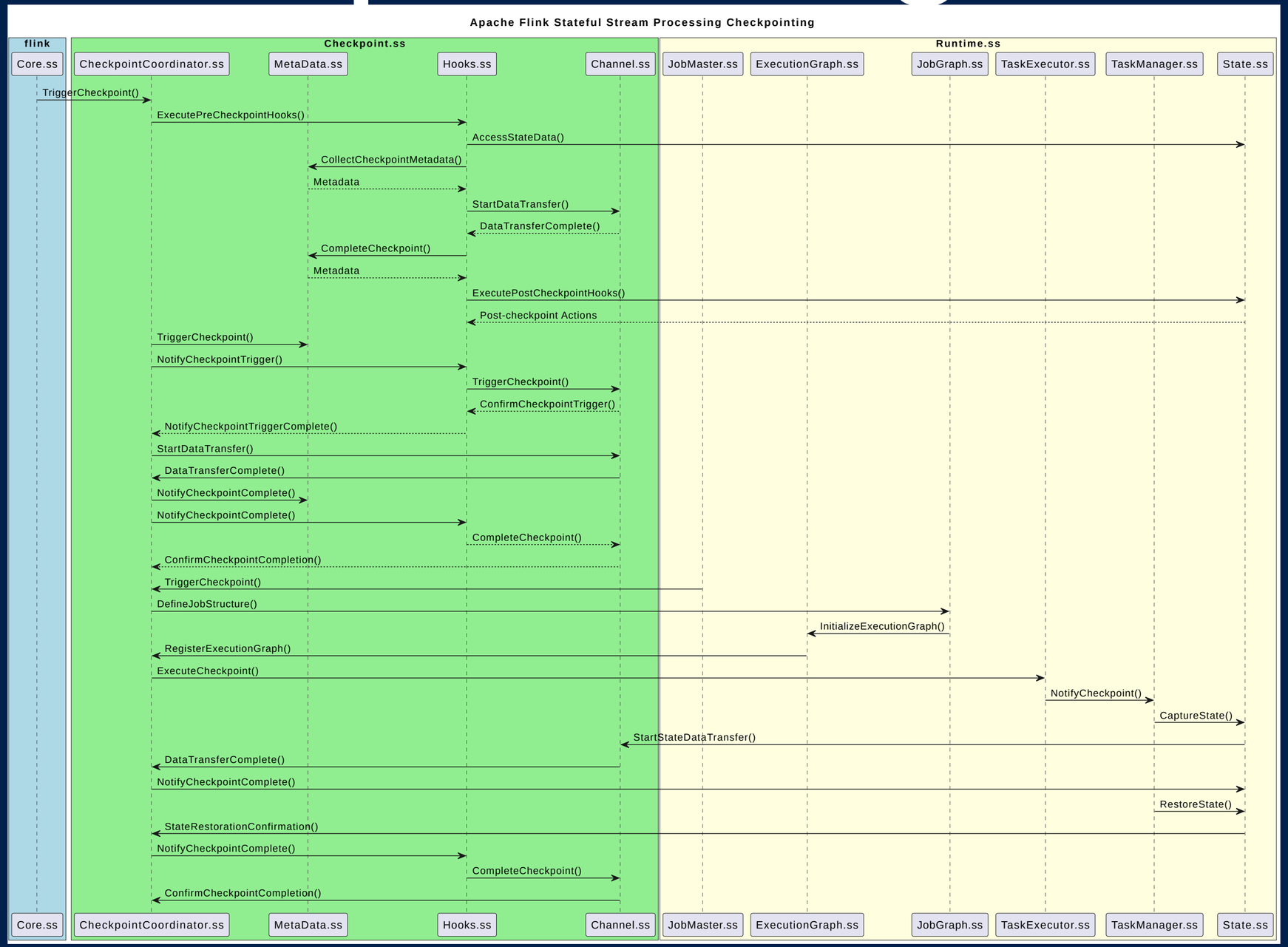
Adapter: found in the Connectors subsystem, facilitates interfacing with external systems like Hadoop

Use Cases

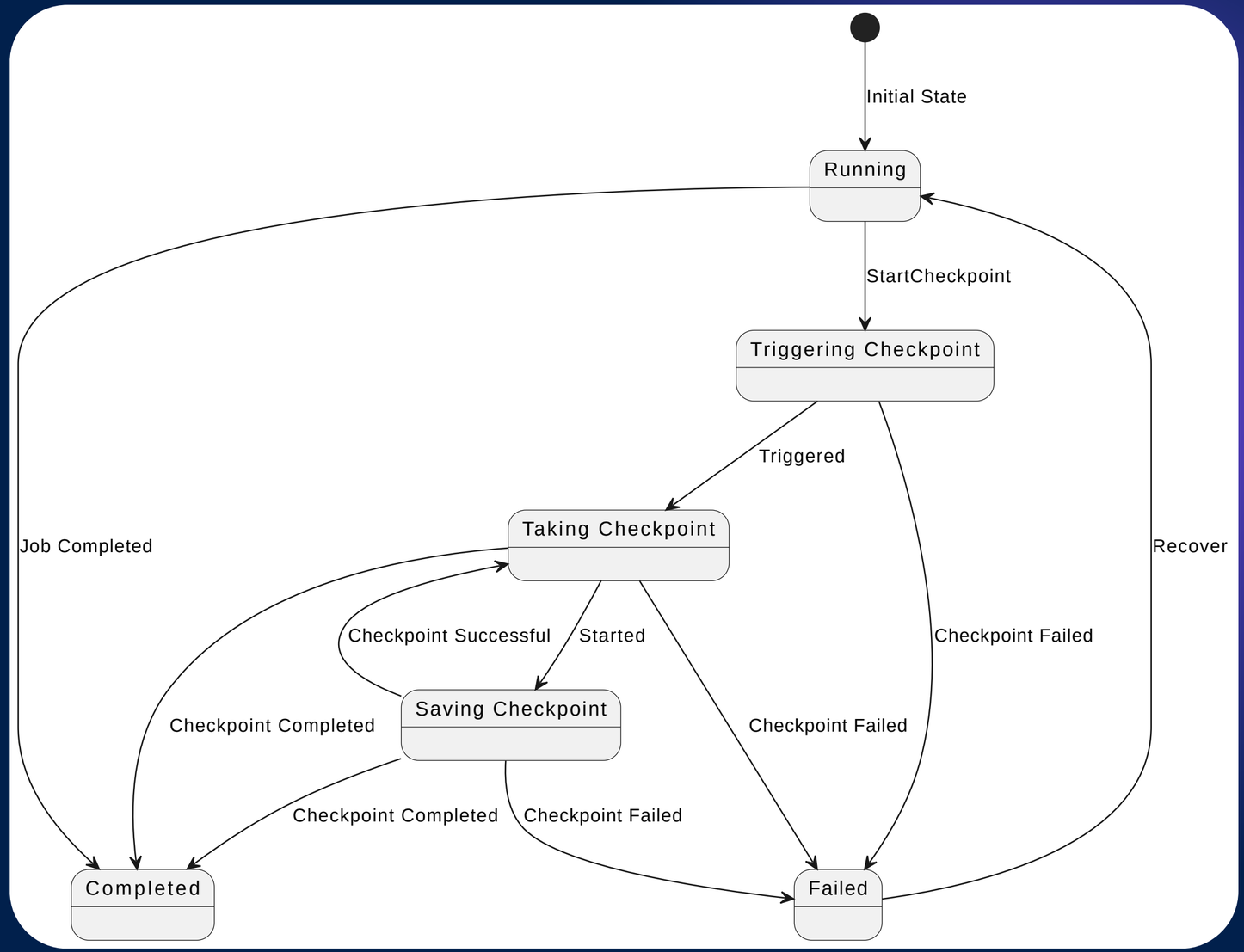
Use Case Diagram



Sequence Diagram



State Diagram



Development Teams

Creator: Volker Markle

Main Contributors: Project Management Committee:
M´arton Balassi, Paris Carbone, Ufuk Celebi, Stephan Ewen, Gyula F´ora, Alan Gates, Greg Hogan,
Fabian Hueske, Vasia Kalavri, Aljoscha Krettek, ChengXiang Li, Andra Lungu, Robert Metzger, Maximilian
Michels, Chiwan Park, Till Rohrmann, Henry Saputra, Matthias J. Sax, Sebastian Schelter, Kostas Tzoumas,
Timo Walther and Daniel Warneke

+ 180 unknown contributors

P. Carbone et al., “Apache FlinkTM: Stream and Batch Processing in a Single Engine,” IEEE Data Engineering Bulletin, vol. 38, Jan. 2015.

Limitations

1. This architecture extraction is not comprehensive as it is done with a focus on the checkpoint subsystem
2. There may be alternative ways to group the subsystems based on which build of Flink is being described
3. lsEdit’s UI has poor usability and crashes often

Lessons Learned

1. Conceptual architecture & concrete architecture have similarities but the details diverge
2. While performing extraction it is important to understand the project directory structure
3. Knowing what to define as subsystems was difficult
4. Checkpointing is a crucial feature in Apache Flink’s value proposition for data integrity

Extraction Demo