BLESSERS

Apache Flink

Discrepancy Analysis Presentation (A3)

By Noor, Abeed, Daoud, Kingsley, Kas And Kamsi

Contents

01
Introduction

02
Architecture Overview

03
Discrepancy Analysis

04
Rationale and Reflections

05
Limitations and Lessons Learned

06
Conclusion

01

Introduction

01 Introduction

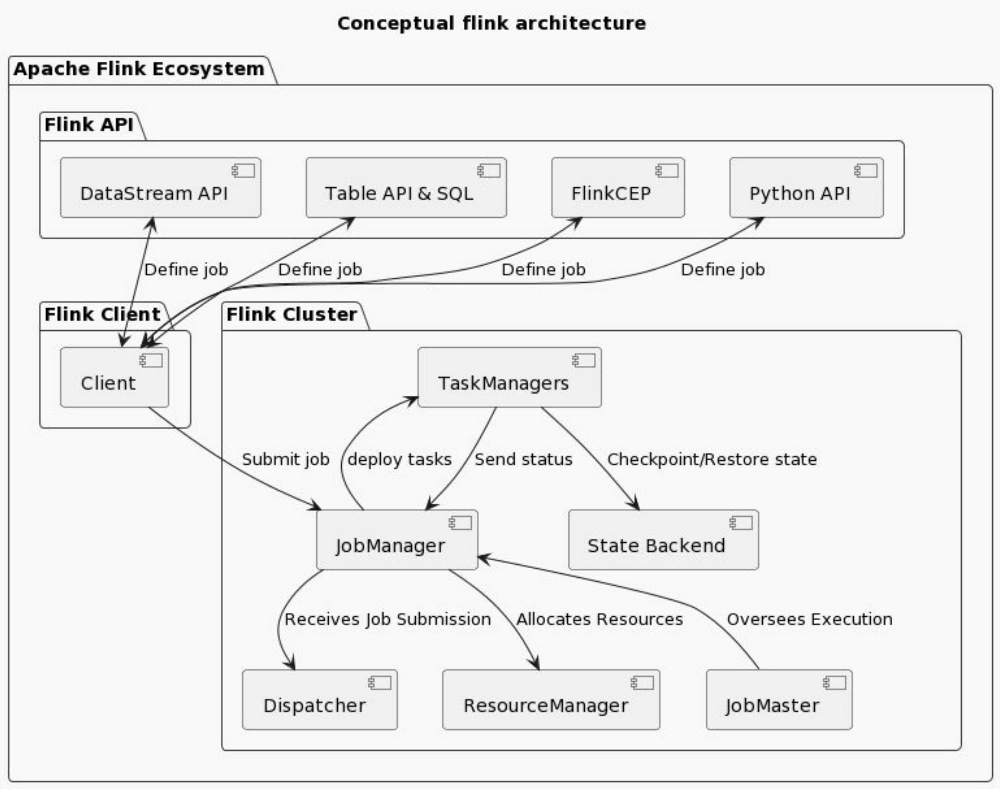
Apache Flink

- Open-source platform for scalable stream and batch data processing
- Designed for high speed, in-memory computations in various cluster environments
- Importance of alignment between design and implementation

02

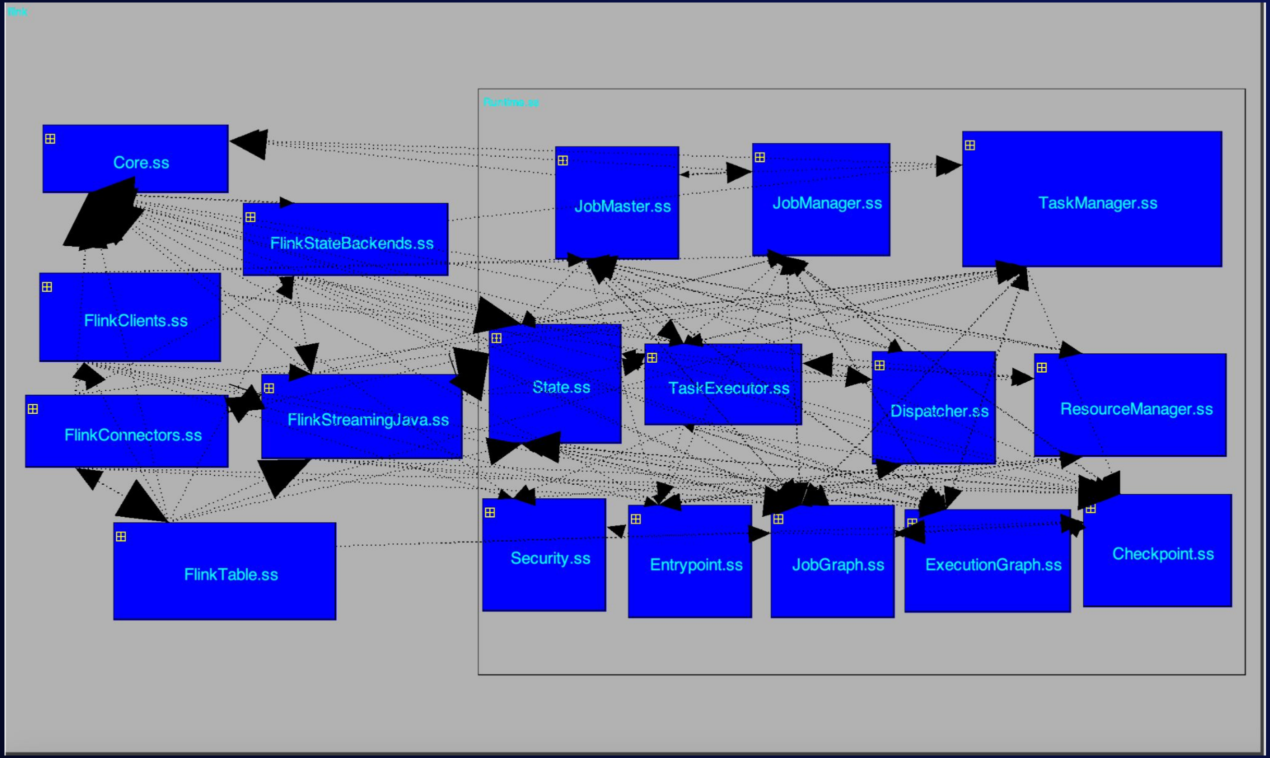
Architecture Overview

02 Top Level Conceptual Architecture - recap



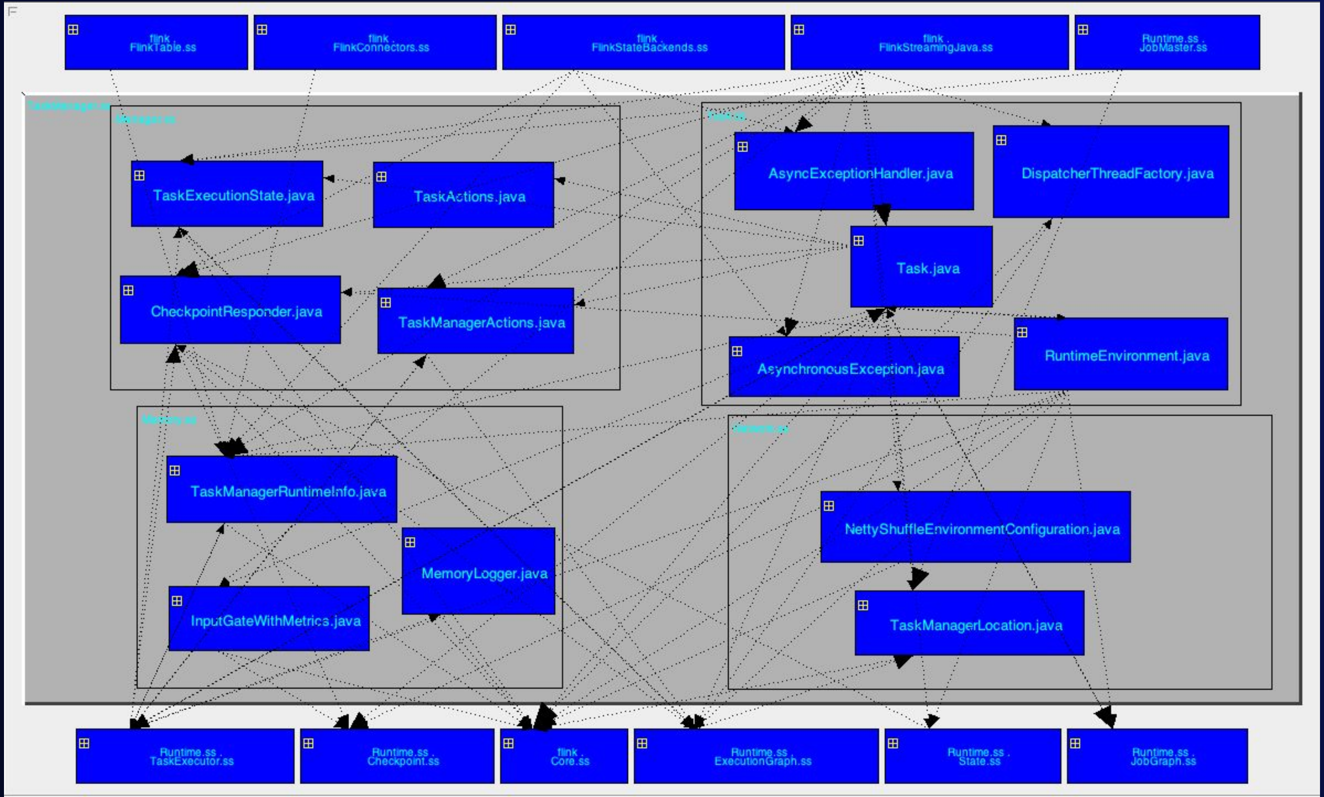
Flink Conceptual Architecture

02 Top Level Concrete Architecture - Recap



Flink Concrete Architecture

02 Task Manager subsystem Concrete Architecture

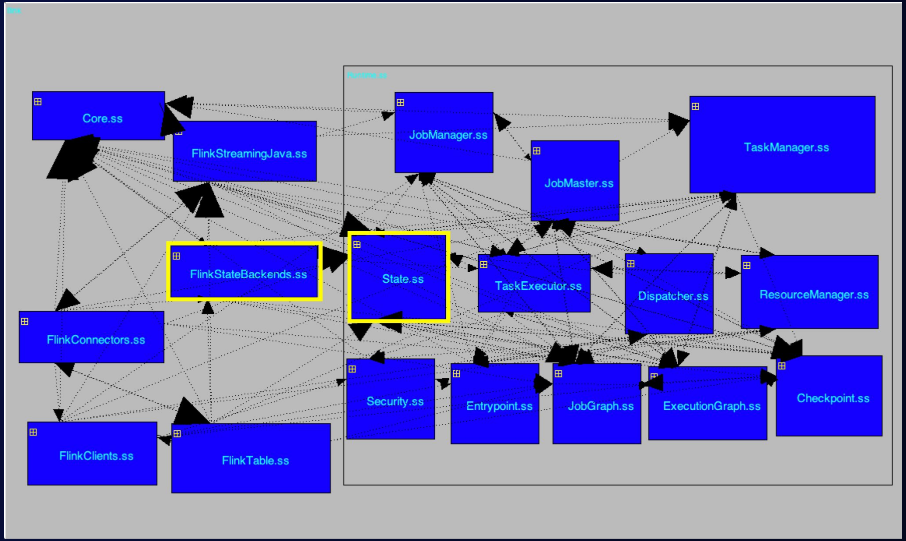

flink . FlinkTable.ss
flink . FlinkConnectors.ss
flink . FlinkStateBackends.ss
flink . FlinkStreamingJava.ss
Runtime.ss . JobMaster.ss
TaskExecutionState.java
TaskActions.java
CheckpointResponder.java
TaskManagerActions.java
AsyncExceptionHandler.java
DispatcherThreadFactory.java
Task.java
AsynchronousException.java
RuntimeEnvironment.java
TaskManagerRuntimeInfo.java
MemoryLogger.java
InputGateWithMetrics.java
NettyShuffleEnvironmentConfiguration.java
TaskManagerLocation.java
Runtime.ss . TaskExecutor.ss
Runtime.ss . Checkpoint.ss
flink . Core.ss
Runtime.ss . ExecutionGraph.ss
Runtime.ss . State.ss
Runtime.ss . JobGraph.ss

Task Manager Concrete Architecture

02 Architectural and Design Styles

Architectural Styles & Design Patterns

- **Repository Style**: State backend stores data.
- **Pipe & Filter Style**: Data flow via network subsystem and processing data using operators in Tasks.
- **Factory Pattern**: Can be found in The thread dispatcher to create a threads for tasks to run.
- **Observer Pattern**: Checkpoint responder observes changes in checkpoint subsystem .

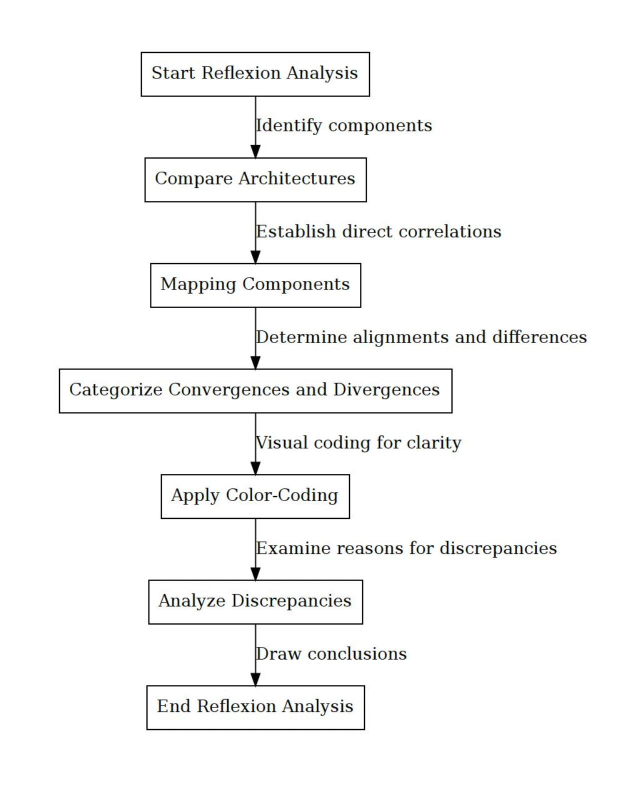


State subsystems indicate a repository architecture style

03

Discrepancy Analysis

03 Discrepancy Analysis



Start Reflexion Analysis

Identify components

Compare Architectures

Establish direct correlations

Mapping Components

Determine alignments and differences

Categorize Convergences and Divergences

Visual coding for clarity

Apply Color-Coding

Examine reasons for discrepancies

Analyze Discrepancies

Draw conclusions

End Reflexion Analysis

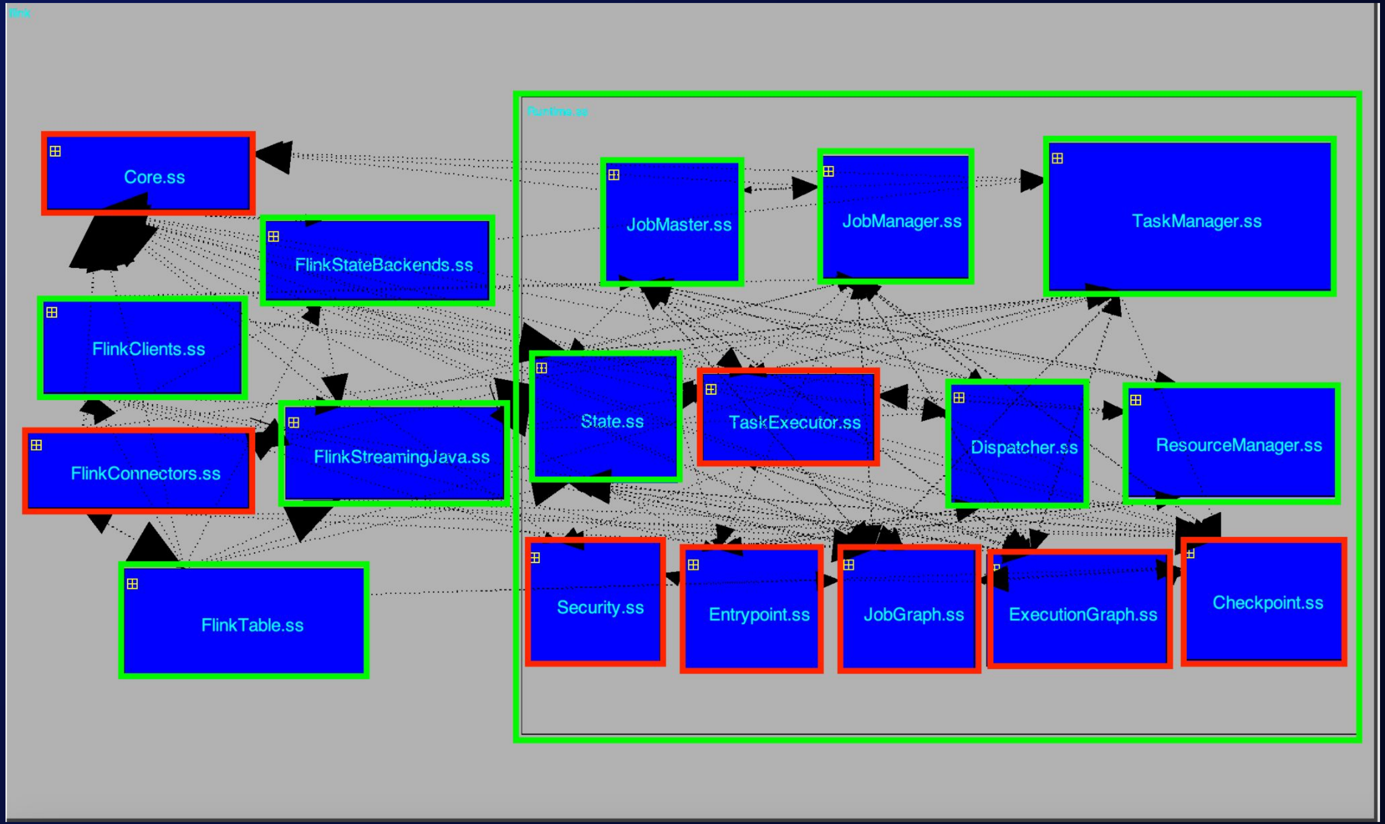
Methodology :

- Architectures were compared using a reflexion model to find the alignments and the discrepancy.

- The subsytems were categorized as either convergences or divergences, we didn’t find any absences.

- The use of color-coding was implemented on the concrete architecture to better illustrate this.

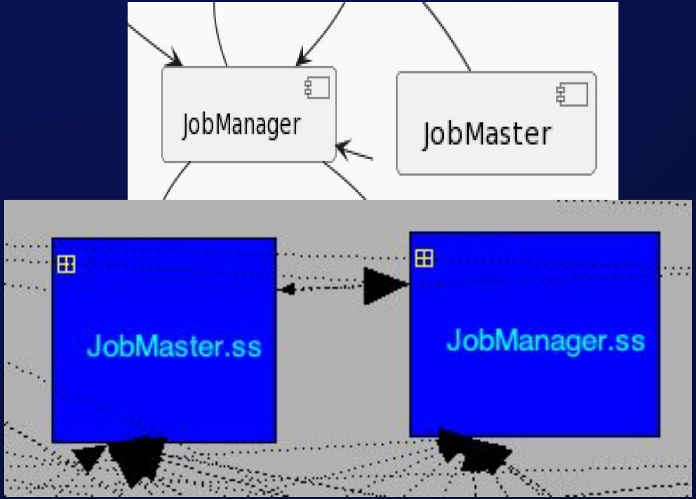
03 Top Level Comparative Analysis



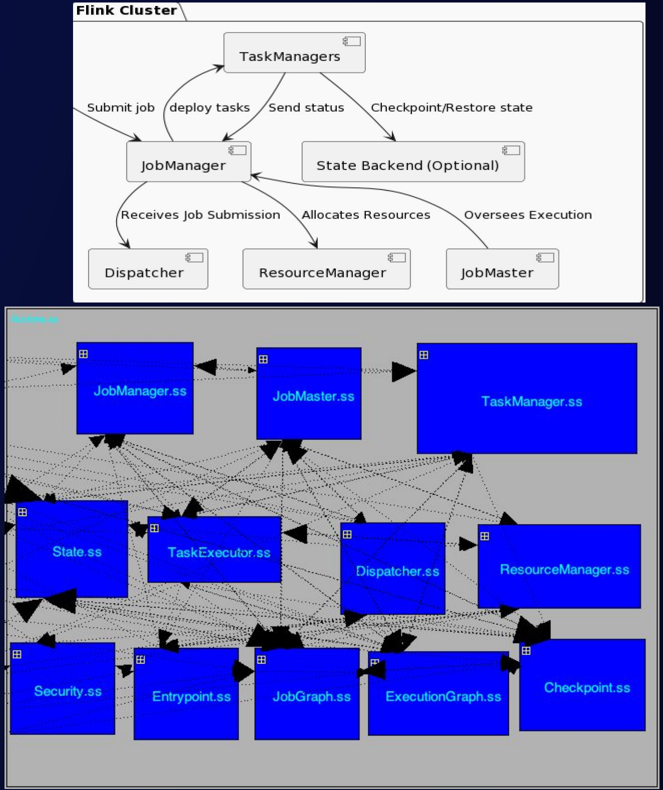
🟩 Convergences

🟥 Divergences

03 Top Level Architecture - Convergence

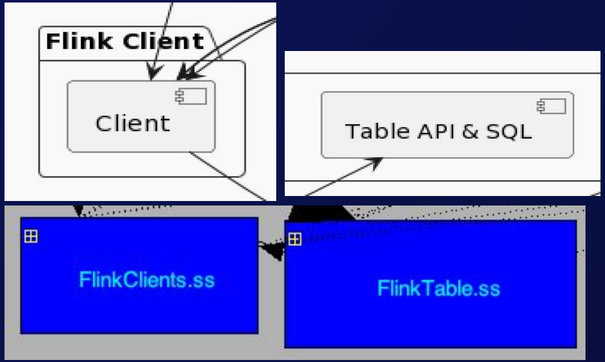


JobMaster/JobManager &
JobMaster.ss/JobManager.ss

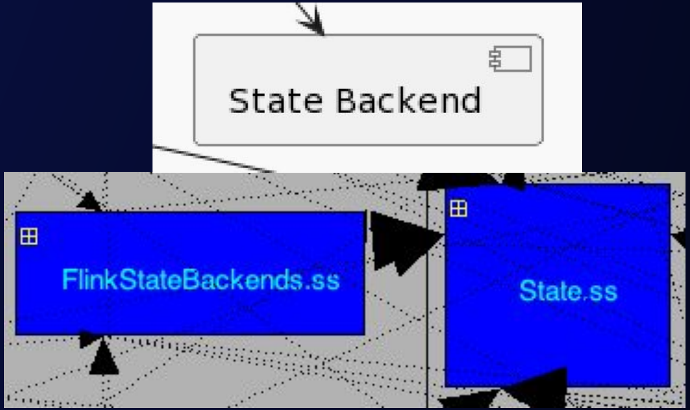


Flink Cluster & Runtime.ss

03 Top Level Architecture - Convergence

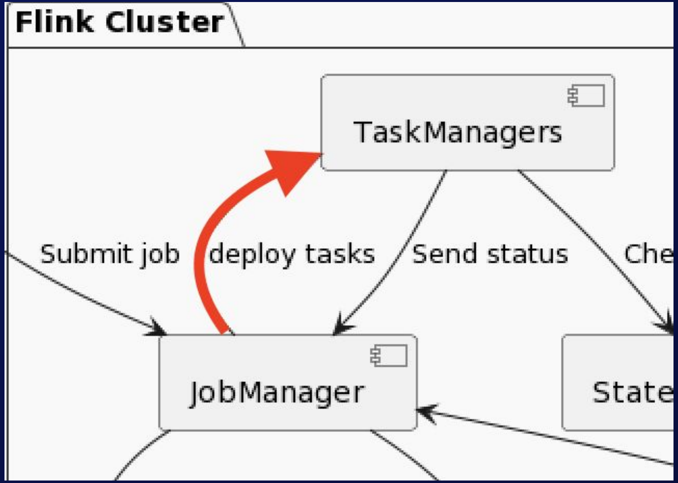


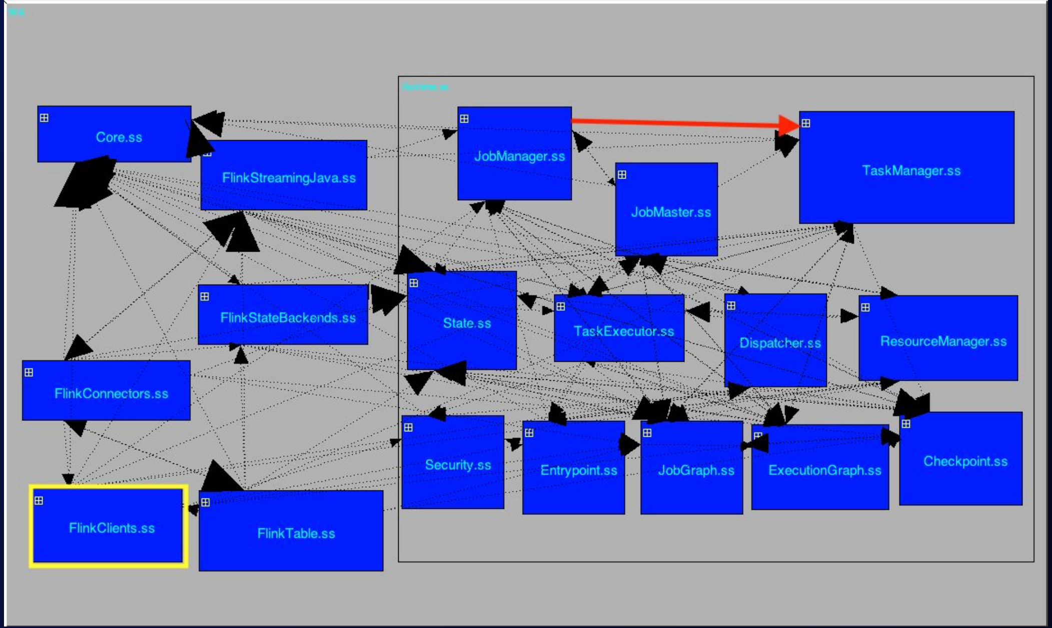
Flink Client/Table API &
FlinkClients.ss/FlinkTable.ss



State Backend &
FlinkStateBackends.ss & State.ss

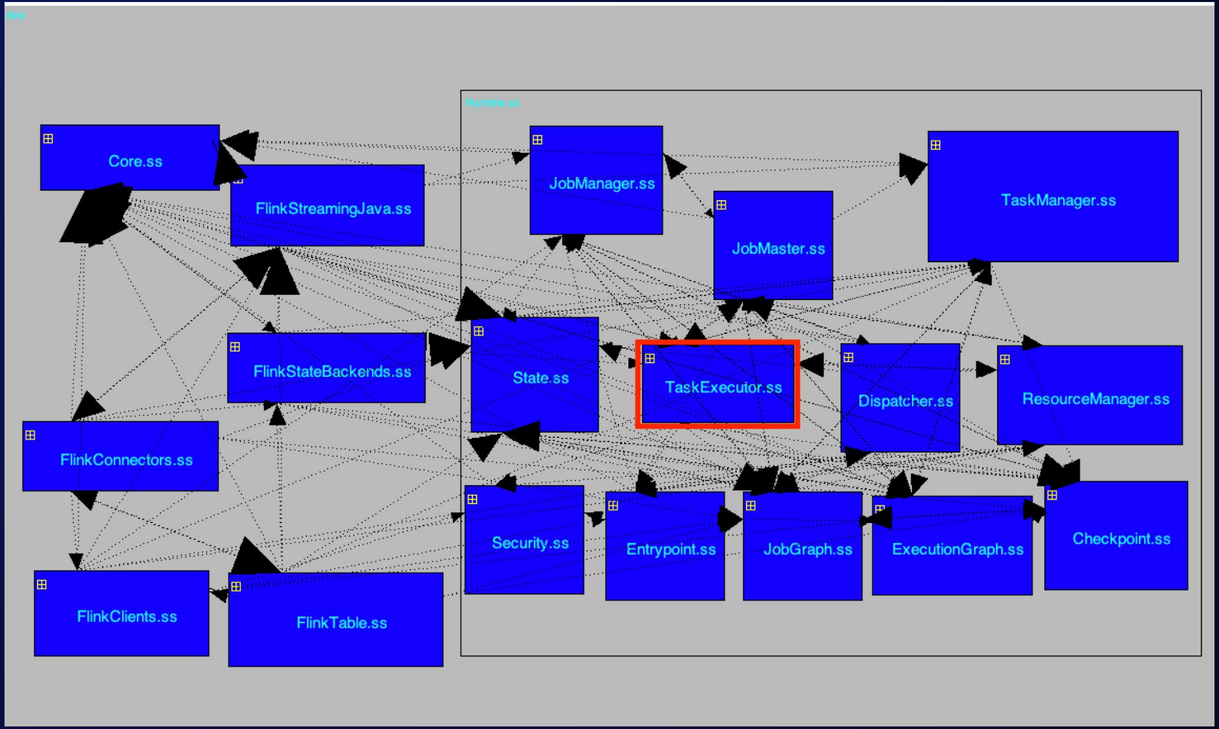
03 Top Level Comparative Analysis - Divergences





The interaction between JobManager & TaskManager is facilitated through the JobMaster

03 Top Level Comparative Analysis - Divergences



The Task Manager executes tasks and manages task execution through the TaskExecutor.ss

03 Enhancing Conceptual and Concrete Architectures

- Improving documentation: Consistency across documents.
- Detailing interaction in conceptual architecture.

04 Rationale and Reflection

Reasons for most discrepancies:

- *performance optimization*
- *scalability changes*
- *iterative development*

04 Performance Optimization in Concrete Architecture

- Focus on task manager and memory manager subsystems
- Adaptations for high throughput and low latency
- Implemented techniques: memory segmentation, dynamic task slot allocation

04 Addressing Scaling Challenges

- Scalability needs: From large businesses to small-scale clusters
- Concrete architecture adjustments: Meeting diverse scalability requirements
- Job Manager Subsystem Modifications: Enhancing resource management and job allocation

04 Iterative Development through Feedback Loops

- Role of feedback Loops: Continuous improvement of subsystems
- Response to user feedback and real world challenges
- Incremental improvements: Enhanced system usefulness and robustness

04 Examples of Iterative developments

WHAT	This change involved moving the CheckpointListener from the flink-runtime module to the flink-core module in Apache Flink.
WHO	Becket Qin
WHEN	November 5, 2020
WHY	This was done most likely to improve the modularity and architectural coherence of Apache Flink. They decided to not immediately delete the CheckpointListener from flink-runtime to maintain backward compatibility.

04 Examples of Iterative developments

WHAT	The change involved moving Executors and ExecutorThreadFactory from their previous location(org.apache.flink.runtime) to flink-core(org.apache.flink.util).
WHO	Chesnay Schepler
WHEN	June 23, 2021
WHY	This centralized the Executors and ThreadFactory functionalities within the core module of Apache Flink. This also potentially improved access, maintainability, and logical organization.

05

Limitations & Lessons Learned

05 Limitations in the Analysis Process

- Complexity and Constant Change
  - Challenge in keeping up with Apache Flink's evolving nature.

- Theory vs. Reality Gap
  - Discrepancies between design plans and practical implementation.

- Subjectivity and Resource Constraints
  - Influence of analysts' perspectives and limitations in available resources.

05 Lessons Learned

- Architectural Flexibility is Key
    - The need to adapt to evolving technologies and requirements.
- Balancing Ideals and Practicalities
    - Harmonizing visionary design with real-world implementations.
- The Importance of Clear Communication
    - Ensuring consistent understanding through effective documentation and terminology.

06

Conclusion

06 Conclusion

- Key Insights for Apache Flink Architecture
  - Alignment of conceptual and concrete architectures is crucial for optimal performance and scalability.
  - Adaptive architecture is essential to accommodate technological changes and community feedback.

- Recommendations for Enhancements
  - Harmonize architectures through standardization and functional mapping.
  - Implement subsystem modifications like Unified Configuration and Dynamic Memory Allocation for improved efficiency.

- Closing Remarks
  - This analysis sets the direction for Apache Flink’s future development, maintaining its status as a leading data processing platform.

Thanks!

Do you have any questions?

References

1. Data Streaming: Benefits, Examples, and Use Cases. (n.d.). Confluent. Retrieved October 15, 2023, from https://www.confluent.io/learn/data-streaming/
2. What is Apache Flink? (n.d.). Apache Flink. Retrieved October 15, 2023, from https://flink.apache.org/what-is-flink/
3. Flink Architecture I Apache Flink. (n.d.). Apache Nightlies Distribution Directory. Retrieved October 17, 2023, from https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/concepts/flink-architecture/
4. Gębiś, W. (2022, November 29). What is Apache Flink? Architecture, Use Cases, and Benefits. nexocode. Retrieved October 17, 2023, from https://nexocode.com/blog/posts/what-is-apache-flink/
5. Bitrock. (Sep. 2023). “Apache Flink and Kafka Stream: A Comparative Analysis.” Medium. Retrieved from https://medium.com/@BitrockIT/apache-flink-and-kafka-stream-a-comparative-analysis-f8cb5b946ec3
6. Apache Flink. (n.d.). Commit 6bf7d77: Move the CheckpointListener from flink-runtime to flink-core. GitHub. Retrieved November 14, 2023, from https://github.com/apache/flink/commit/6bf7d77a76afa1c705c86adf46fb8732841b9dd8
7. Apache Flink. (n.d.). Commit 52609d3: Move Executors/-ThreadFactory to flink-core. GitHub. Retrieved November 14, 2023, from https://github.com/apache/flink/commit/52609d3f018d6402990460d34131efd7815bc61a