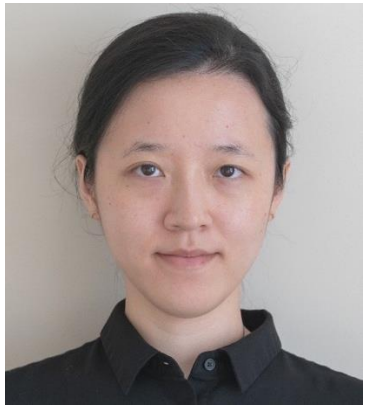
CS460 Projects Overview

Project TAs



Yi (TA)

Hamish (TA)

Logistics

Two projects. Grading is equally weighted

1. Query execution & Optimization with Calcite
2. Recommendation Serving with Spark

Projects are **Individual**

- Do not share code: We will run plagiarism detection tools
- AI code generation is not allowed. No ChatGPT/Claude/Copilot etc

Graded automatically with tests

- Only write code in `src/main/scala`
- Last commit before deadline on `main` branch will be graded

Project IDE: IntelliJ Idea

- Free community edition
- ultimate edition on academic license (epfl.ch email)

Programming Language: Scala

Timeline

Project registration for repos on Moodle 3.03

Project 1: Query execution & Optimization

- Released ~10.03
- Due 17.04

Project 2: Recommendation Serving

- Released ~28.04 (just after easter holidays)
- Due 30.05

Project 1 Learning Goals

- Apache Calcite
  - Extensible framework for query optimization and execution
- Database internals concepts
  - Volcano/iterator (tuple-at-a-time) processing model
  - Late tuple materialization
  - Query plan optimization rules

Project 2 Learning Goals

- Apache Spark
  - Unified engine for large-scale data analytics/science and machine learning

- Data Management / Data Science concepts
  - Loading / Caching/ Pre-processing
  - Data partitioning
  - Predictive analytics / Recommender systems / Machine learning

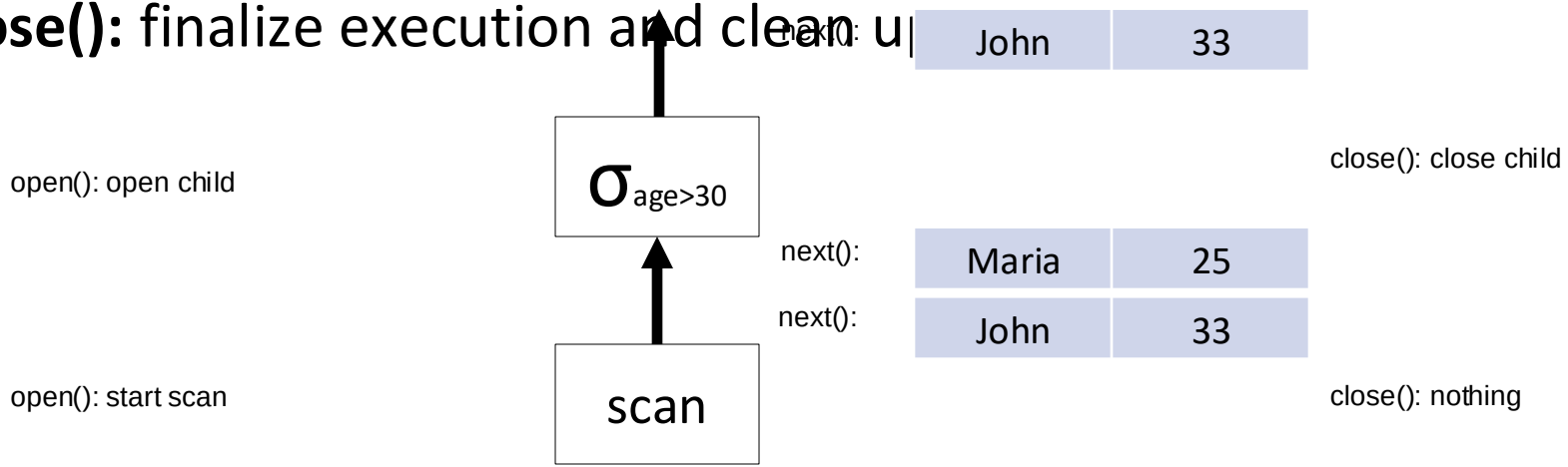
Final Remarks

- You will not have to implement a full system from scratch
  - just the functionality in isolation
- You can run tests locally with Intellij (`src/main/test`)
- IMPORTANT: Do not edit build files/interfaces
  - Auto-grader will fail if you change any interface definition in the skeleton
- Only the latest commit in the main branch will be graded
- **Register for the projects on Moodle before 3.03**

Project 1

Task 1: Volcano engine

- **open():** initialize operator state
- **next():** process and return next tuple (or EOF)
- **close():** finalize execution and clean up



Maria	25
John	33

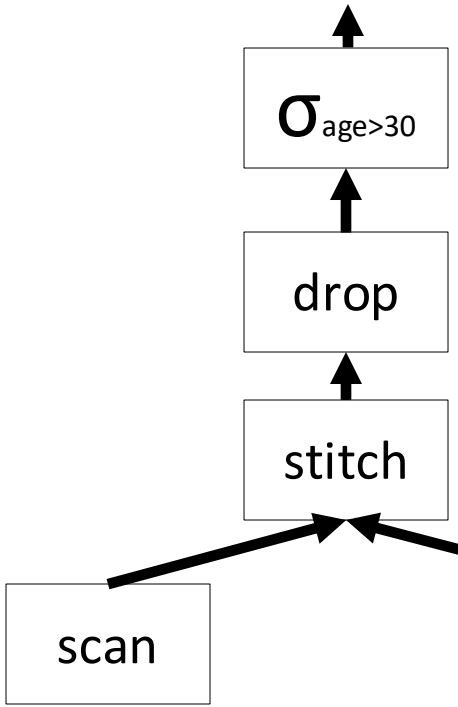
Task 1: Volcano engine (cont)

- Implement Scan, Select, Project, Join, Aggregate, Sort
- Please read documentation from parent classes
- First implement correctly, then optimize

Task 2: Late materialization (naïve)

- No need to reconstruct full tuple on scan
  - As long as we keep virtual IDs, we can process columns individually and stitch them later
- The task involves:
  - stitching
  - interfacing with operators that do not support late materialization
  - late materialization-aware operators
- Operators use volcano model in this task as well

Task 2.A: Stitch and Drop



John	33

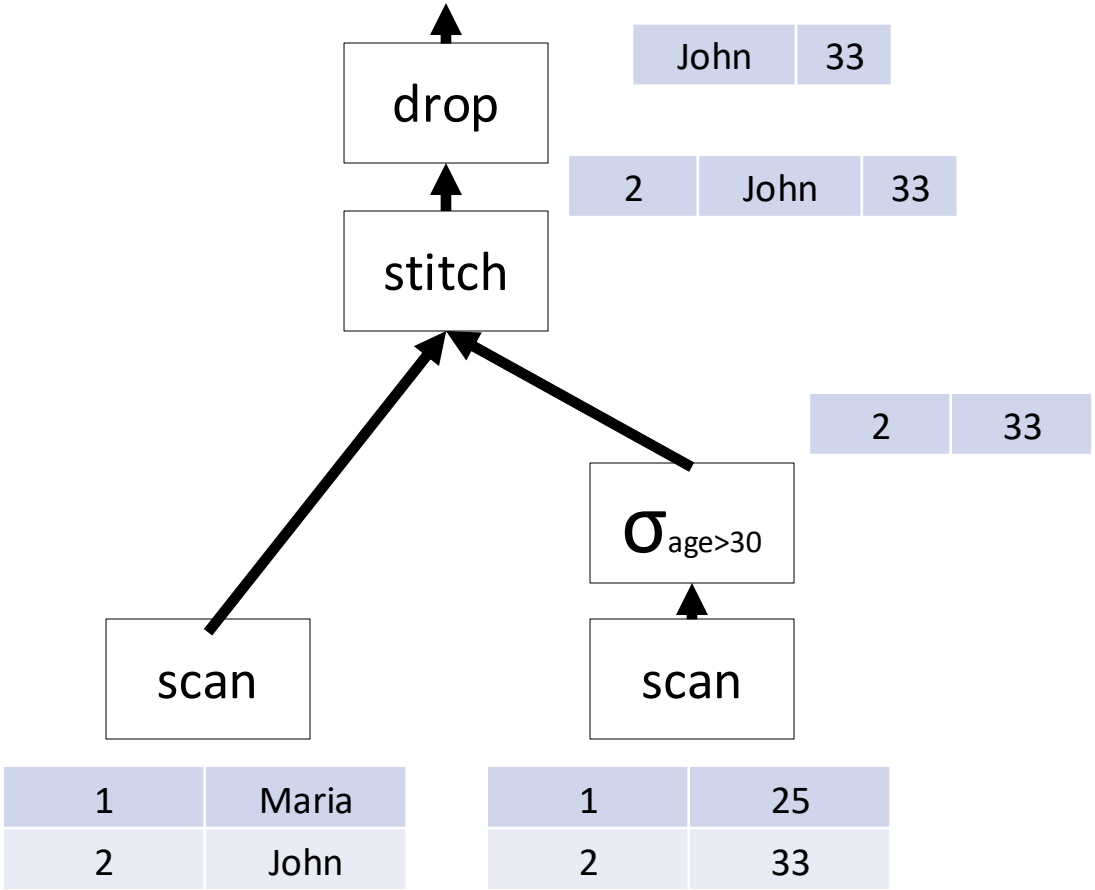
Maria	25
John	33

1	Maria	25
2	John	33

1	Maria
2	John

1	25
2	33

Task 2.B: Late materialization operators

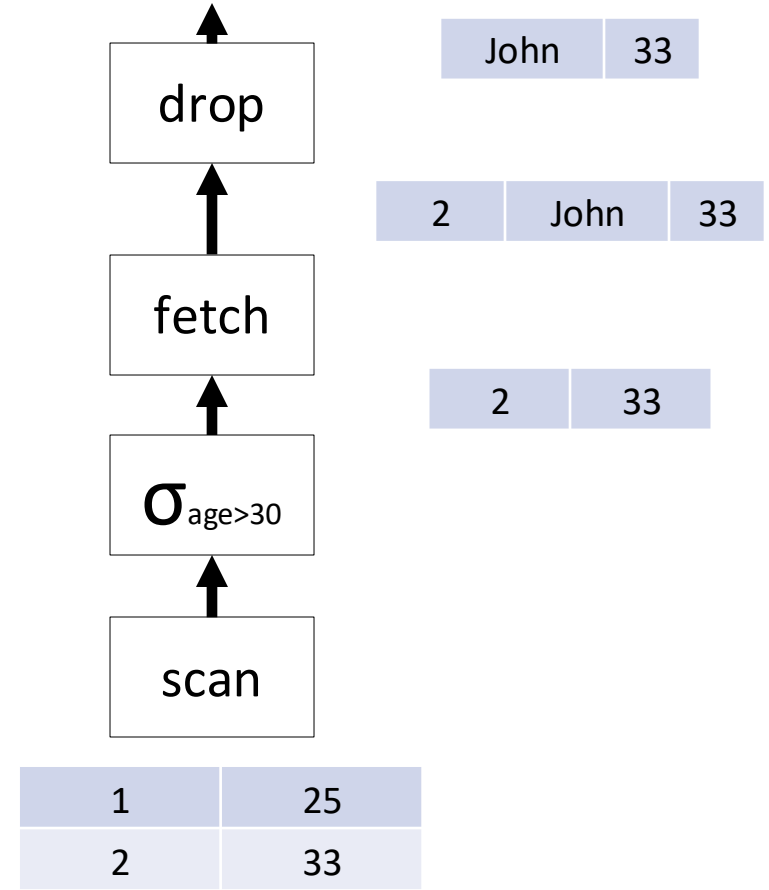


Task 3: Optimization for Late materialization

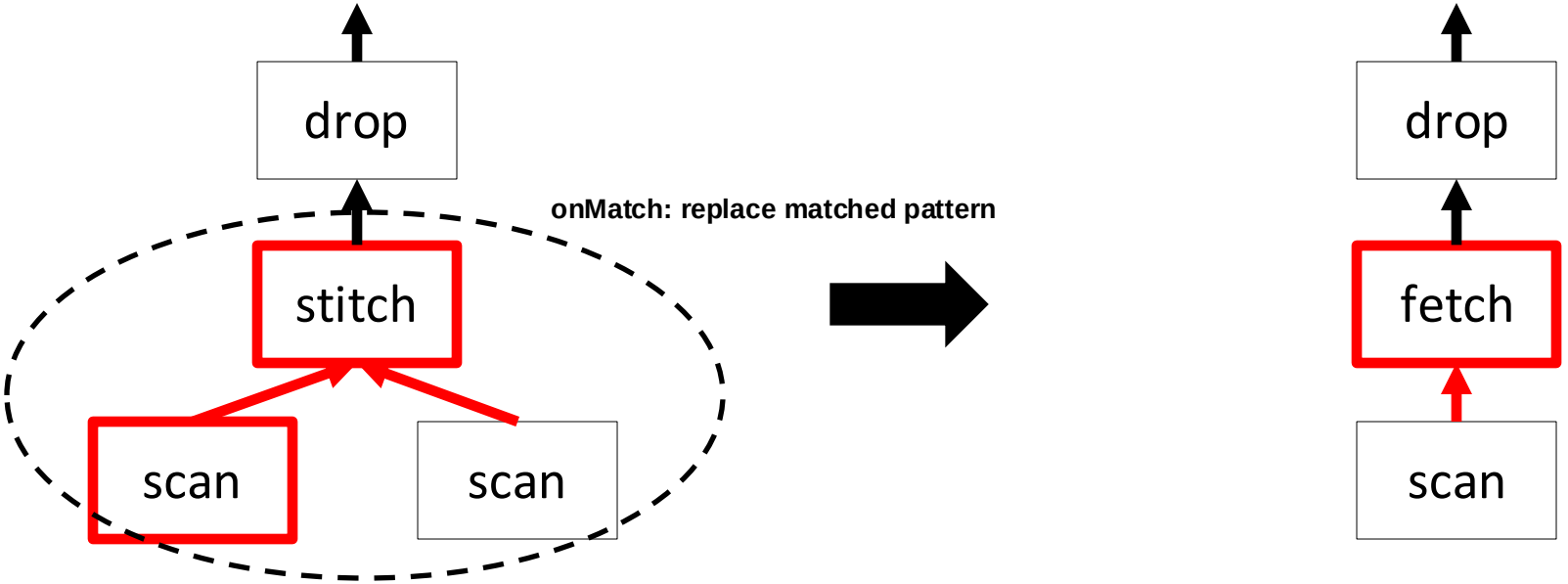
- If virtual ID defines tuples position, stitching is unnecessary. We can fetch missing attributes on-demand.
  - This way, we can avoid scanning the full column
- The task has two goals:
  - Implement operator that fetches missing attributes
  - Implement optimization rules in order to inject the operator to plans

Task 3.A: Fetch Operator

No need to scan full “name” column
Fetch values for virtual IDs that satisfy filter



Task 3.B: Optimization Rules



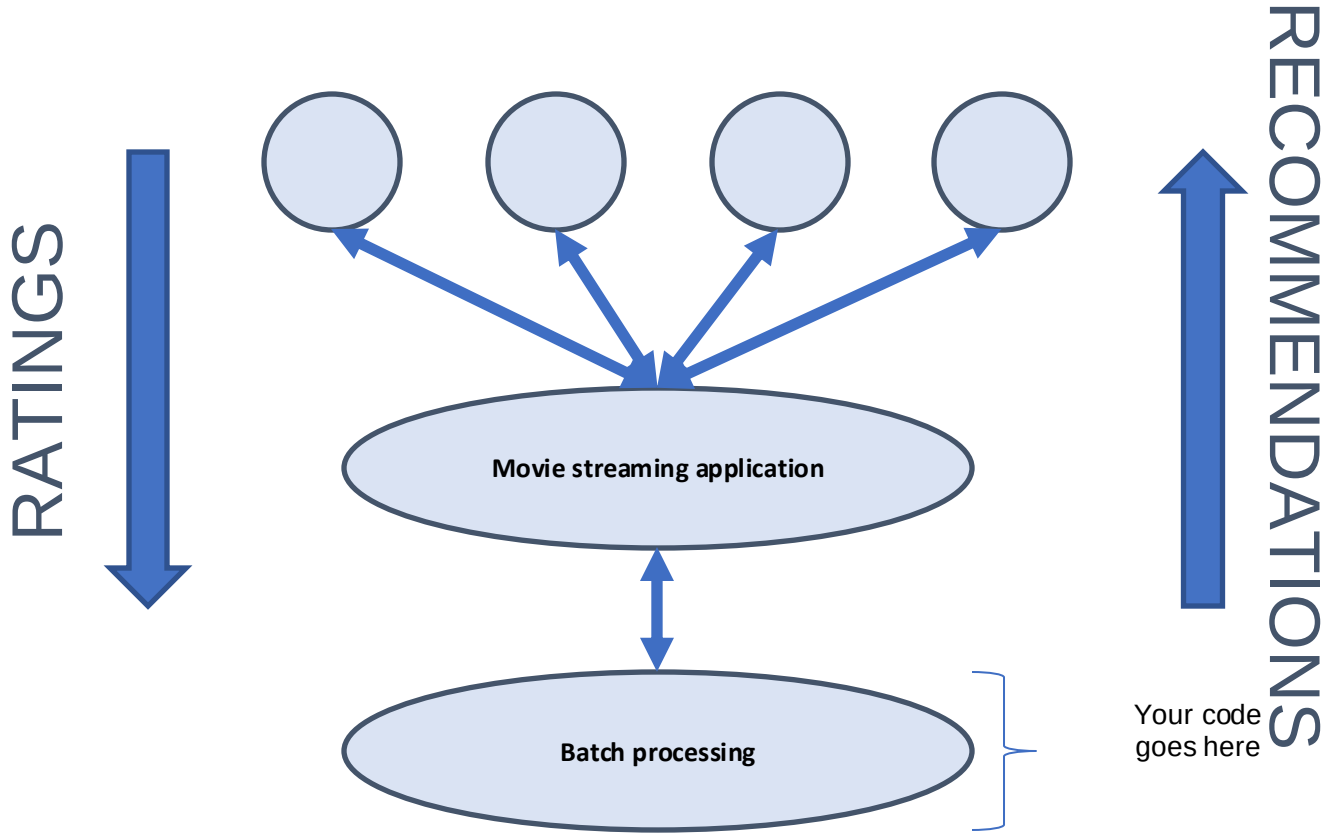
Pattern matching: stitch with scan as left child

Project 2

Project 2 Highlights

- Three milestones (single-deadline for all)
  1. Data loading & Simple data analysis
  2. Movie-ratings pipeline
     - Aggregations & Incremental maintenance
  3. Prediction serving (recommender system)
     - Similarity based recommender: Locality-Sensitive Hashing & Collaborative Filtering
- Dataset: `Movielens`
  - Three sizes:
    - Small for development/debugging
    - Medium for testing/ automatic-testing on Gitlab
    - Large for hands-on experience with cluster

The Usecase


RATINGS
Movie streaming application
Batch processing
RECOMMENDATIONS
Your code goes here

The Data Processing Pipelines

- MovieLens data + simulated ratings
- Loading data with Spark (milestone 1)
- From user ratings to average ratings (milestone 2)
  - Average ratings from log
  - Updates on log propagated to average ratings
- From movie keywords to recommendations (milestone 3)
  - LSH: Similarity-search based on keywords
  - Collaborative filtering through spark mllib

You will not have to implement a full system, just the functionality in isolation