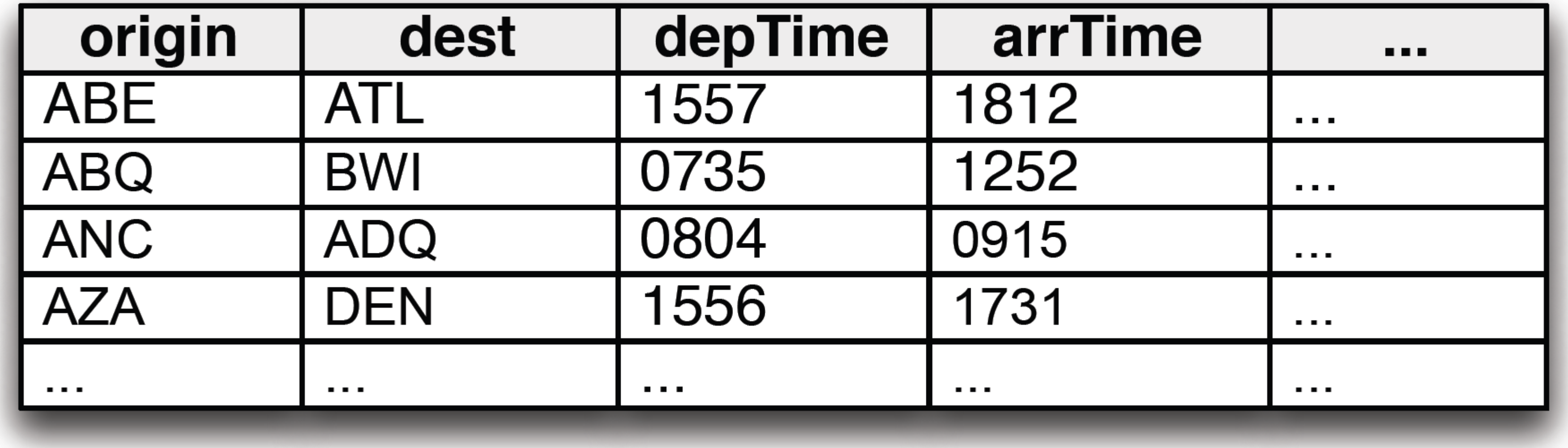
An Approach for Efficient Querying of Large Relational Datasets with OCL-based Languages

Dimitrios S. Kolovos
Ran Wei
Konstantinos Barmpis
{dimitris.kolovos, rw542, kb634}@york.ac.uk

Motivation

- Data used in MDE likely found in non-model artefacts:
  - Spreadsheets
  - Databases
  - XML documents
- Such data needs to be converted for use in model transformations & queries

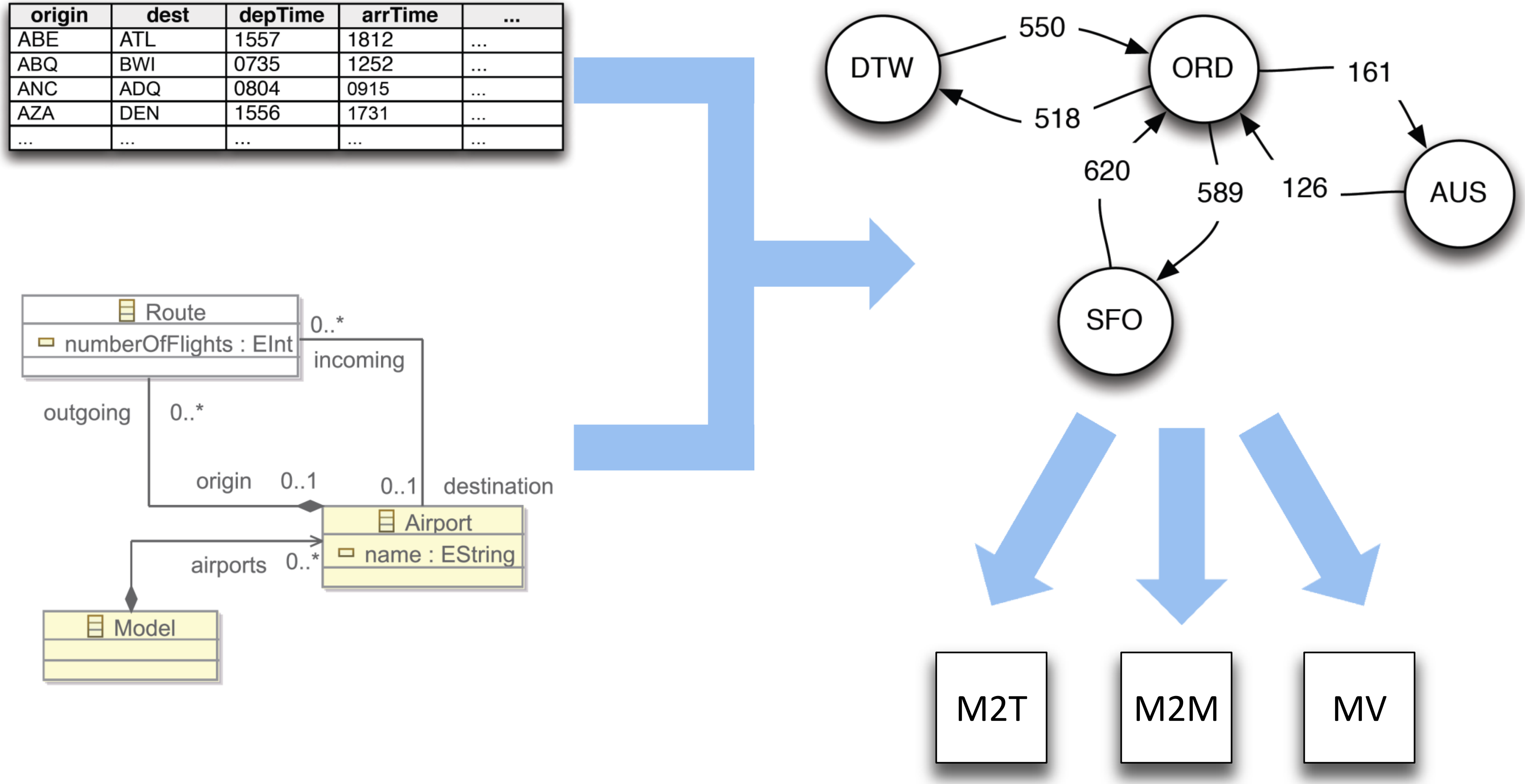
The ATM System



origin	dest	depTime	arrTime	...
ABE	ATL	1557	1812	...
ABQ	BWI	0735	1252	...
ANC	ADQ	0804	0915	...
AZA	DEN	1556	1731	...
...	...	...	...	...

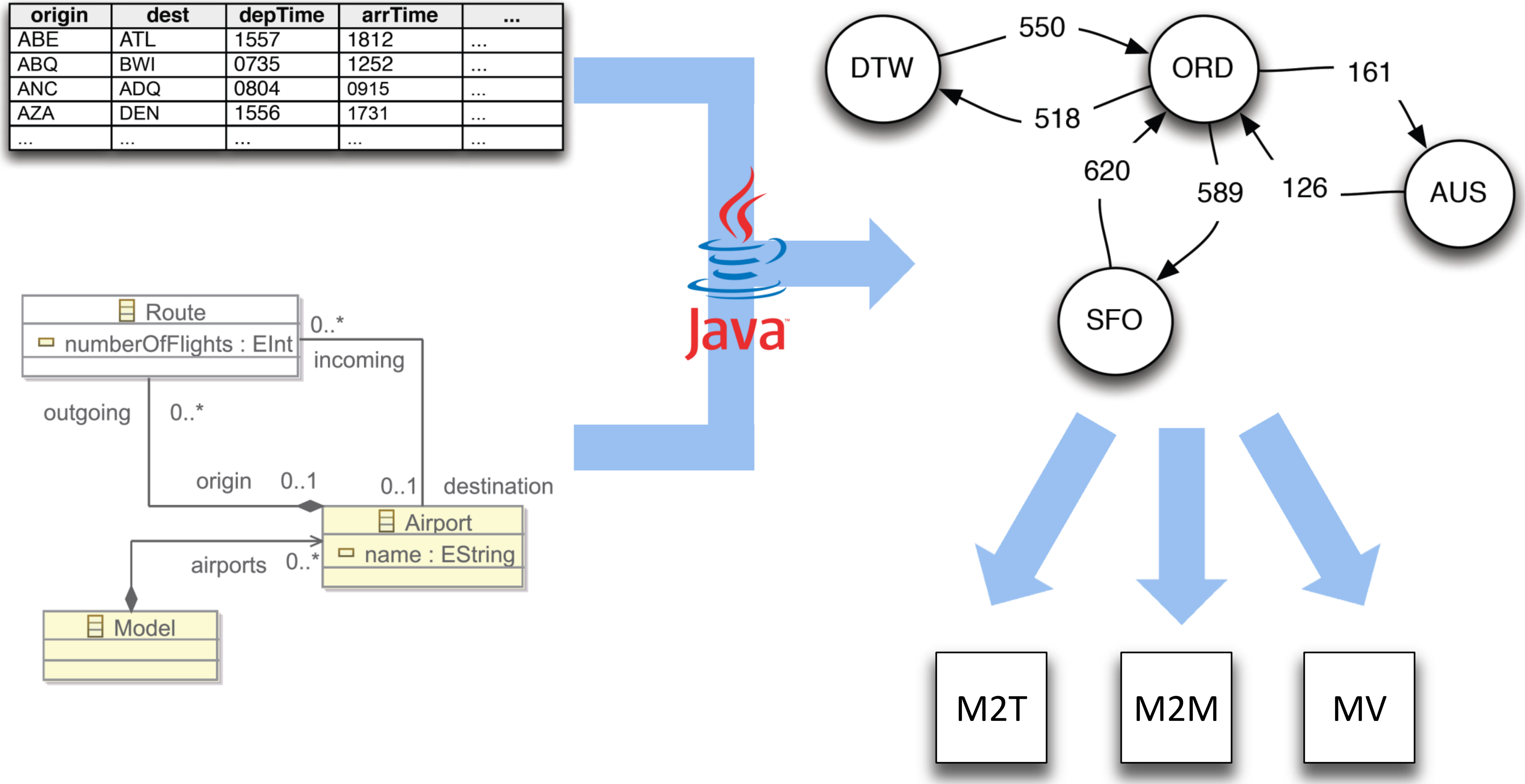
- 1 Table (Flight)
- > 200 Columns
- > 500,000 Rows

The ATM System



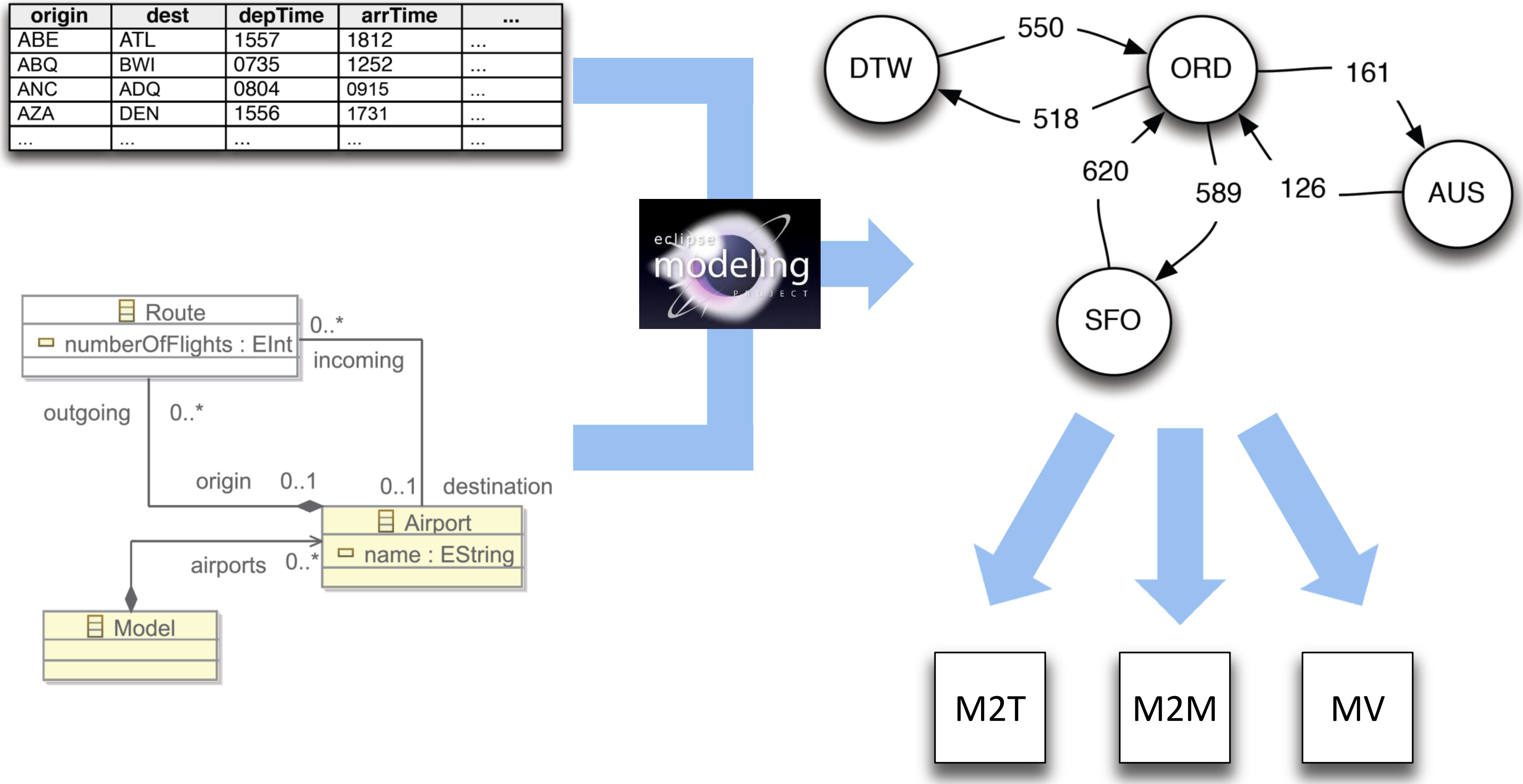
origin	dest	depTime	arrTime	...
ABE	ATL	1557	1812	...
ABQ	BWI	0735	1252	...
ANC	ADQ	0804	0915	...
AZA	DEN	1556	1731	...
...	...	...	...	...

The ATM System



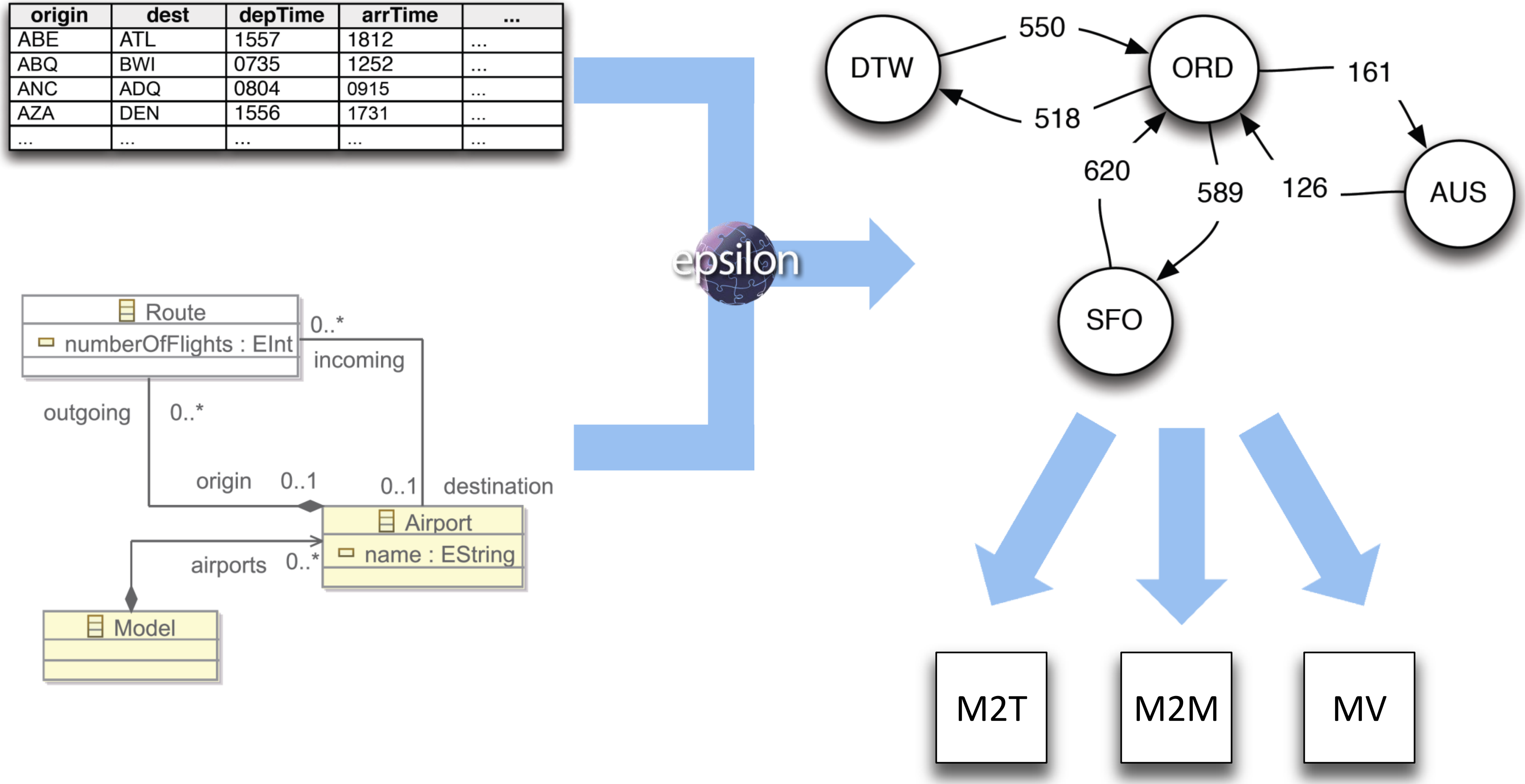
origin	dest	depTime	arrTime	...
ABE	ATL	1557	1812	...
ABQ	BWI	0735	1252	...
ANC	ADQ	0804	0915	...
AZA	DEN	1556	1731	...
...	...	...	...	...

The ATM System



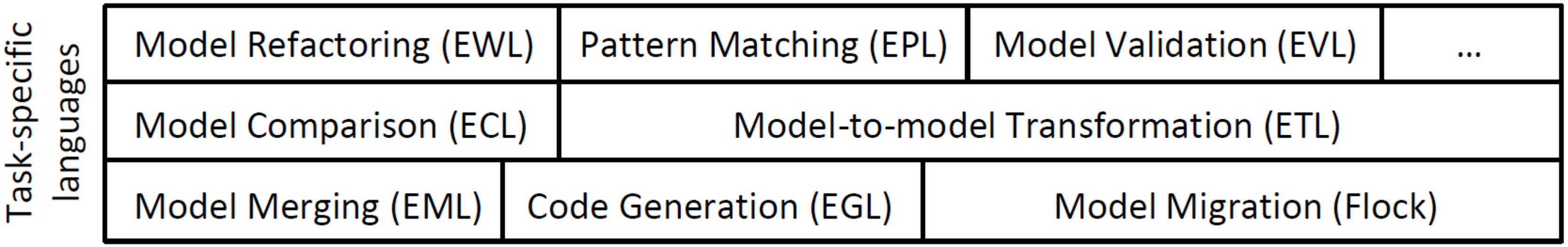
origin	dest	depTime	arrTime	...
ABE	ATL	1557	1812	...
ABQ	BWI	0735	1252	...
ANC	ADQ	0804	0915	...
AZA	DEN	1556	1731	...
...	...	...	...	...

The ATM System



origin	dest	depTime	arrTime	...
ABE	ATL	1557	1812	...
ABQ	BWI	0735	1252	...
ANC	ADQ	0804	0915	...
AZA	DEN	1556	1731	...
...	...	...	...	...

The Epsilon Modeling Suite & EOL



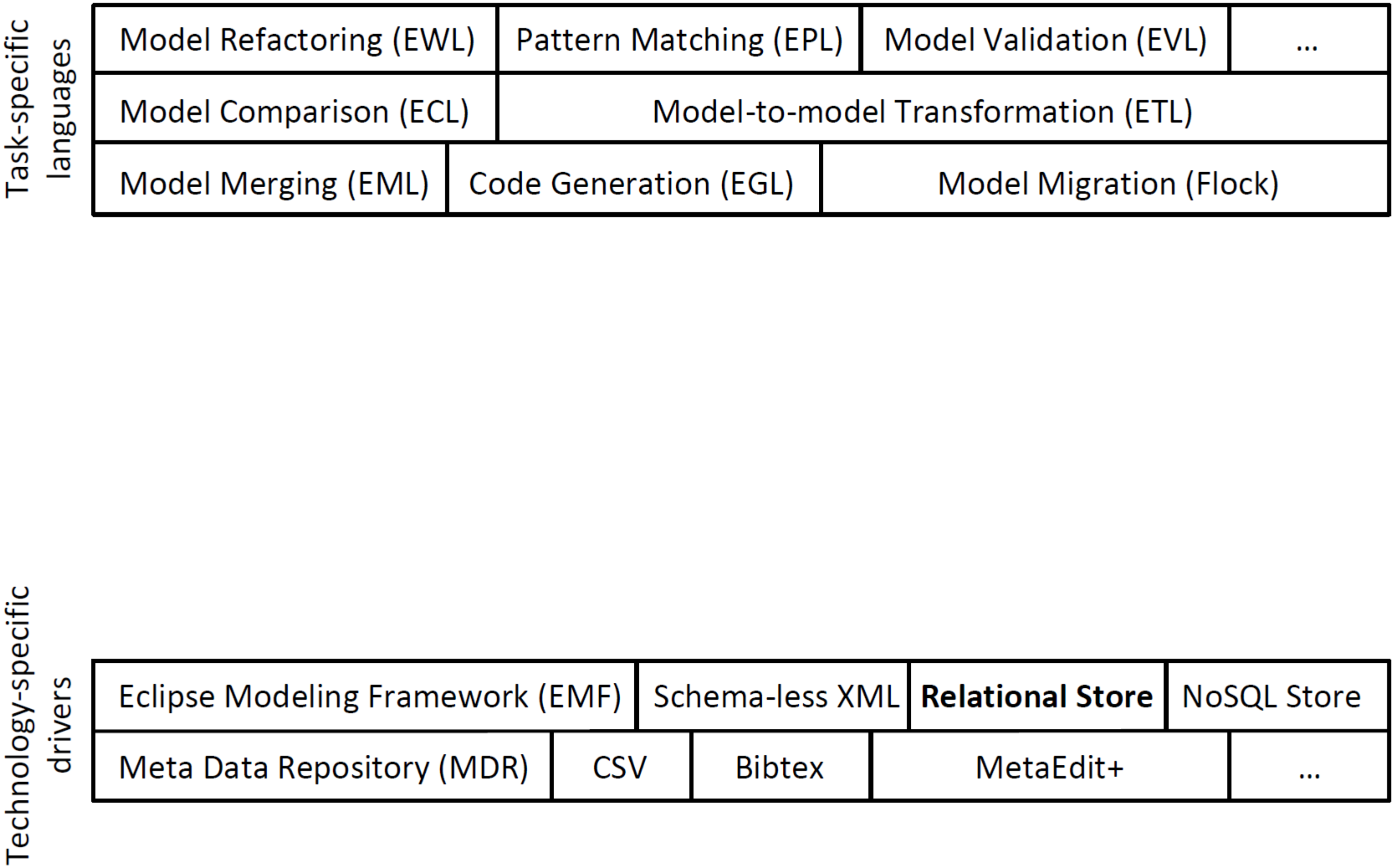
Task-specific languages

Model Refactoring (EWL) | Pattern Matching (EPL) | Model Validation (EVL) | …

Model Comparison (ECL) | Model-to-model Transformation (ETL)

Model Merging (EML) | Code Generation (EGL) | Model Migration (Flock)

The Epsilon Modeling Suite & EOL



Task-specific languages

Model Refactoring (EWL) | Pattern Matching (EPL) | Model Validation (EVL) | ...

Model Comparison (ECL) | Model-to-model Transformation (ETL)

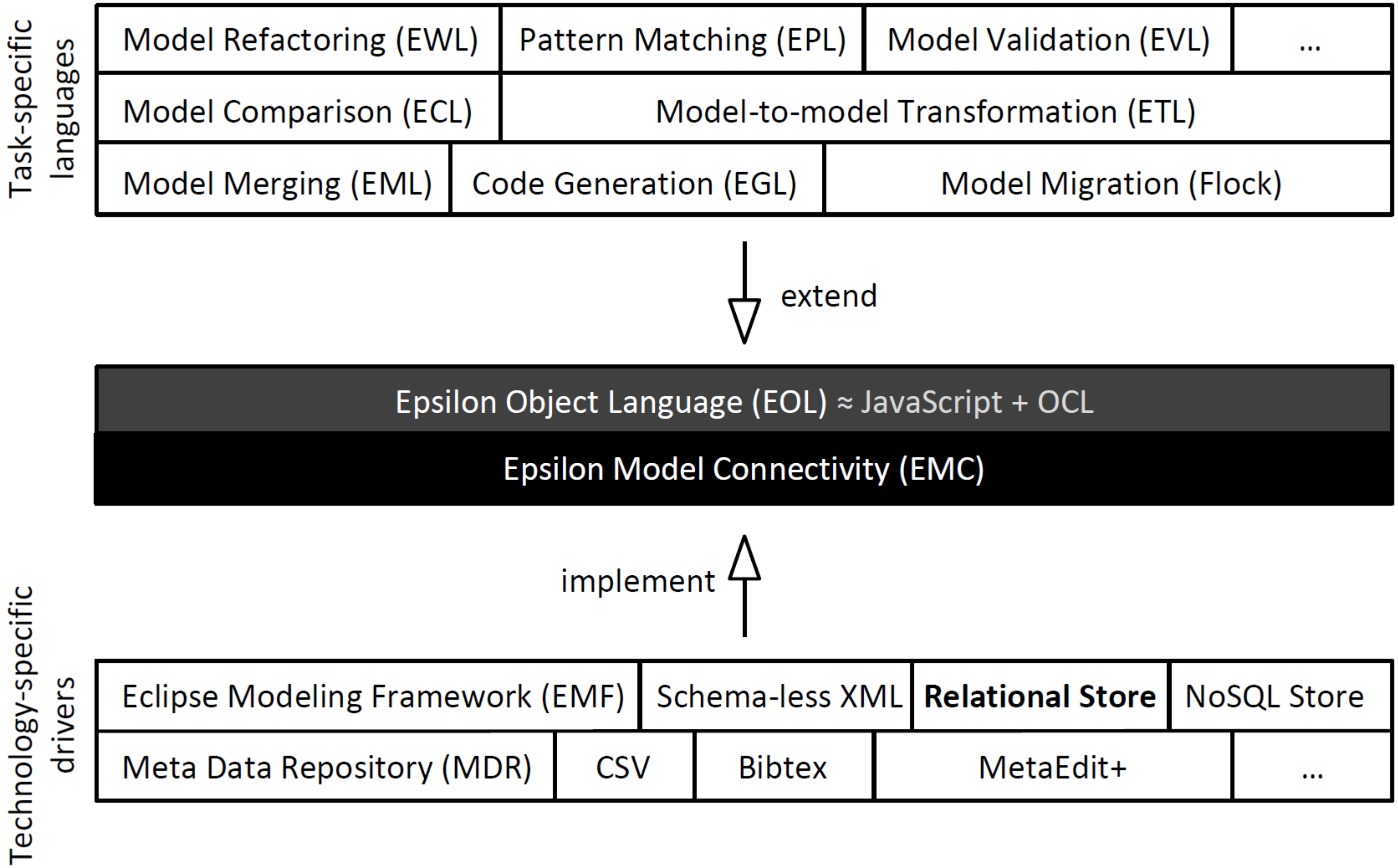
Model Merging (EML) | Code Generation (EGL) | Model Migration (Flock)

Technology-specific drivers

Eclipse Modeling Framework (EMF) | Schema-less XML | **Relational Store** | NoSQL Store

Meta Data Repository (MDR) | CSV | Bibtex | MetaEdit+ | ...

The Epsilon Modeling Suite & EOL



Task-specific languages

Model Refactoring (EWL) | Pattern Matching (EPL) | Model Validation (EVL) | …

Model Comparison (ECL) | Model-to-model Transformation (ETL)

Model Merging (EML) | Code Generation (EGL) | Model Migration (Flock)

extend

Epsilon Object Language (EOL) ≈ JavaScript + OCL

Epsilon Model Connectivity (EMC)

implement

Technology-specific drivers

Eclipse Modeling Framework (EMF) | Schema-less XML | **Relational Store** | NoSQL Store

Meta Data Repository (MDR) | CSV | Bibtex | MetaEdit+ | …

Challenges (1)

Taking the following OCL-like expression to retrieve the number of distinct airports:

Flight.allInstances.origin.asSet().size()

We would need to:

1. Inspect the model and compute a collection of all model elements of type Flight;
2. Iterate through the contents of the collection (from step 1) and collect the values of the property *origin* in a new collection;
3. Remove all duplicates from the collection (from step 2);
4. Compute the size of the collection computed in step 3.

Challenges (2)

The following issues arise if the information is stored in a relational database:

- Computing the *Flight.allInstances* collection requires the engine to perform a:

  *select * from Flight*

  SQL query. For large tables (such as Flight) the returned set needs to be streamed from the database.
- Such streamed sets restrict us to:
  - Forward-only iteration
  - Size can only be calculated after exhaustive iteration
  - Only 1 set can be streamed at a time in a MySQL store.

Challenges (3)

The following issues arise if the information is stored in a relational database:

- The next step would be to iterate through all the rows of the Flight table through the streamed set and collect the values of *origin*.
- This is inefficient as using a:

  select origin from Flight

  SQL statement would be orders of magnitude faster.

Challenges (4)

The following issues arise if the information is stored in a relational database:

- Eliminating duplicates is similarly inefficient and can be easily done using a

  select distinct origin from Flight

  SQL statement.
- Calculating the size of a streamed result-set without invalidating the result-set itself is an issue. By contrast, this could be computed in one step using a:

  select count(distinct origin) from Flight.

  SQL statement.

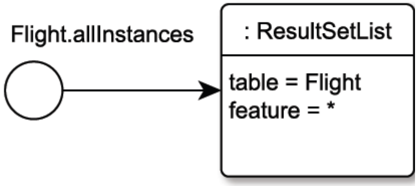
Solutions (1)

Calculate the average delay of flights flying from JFK to LAX on Sundays:

```
Flight.allInstances
        .select(f | f.origin=“LAX”)
        .select(f | f.dest=“JFK”
         and f.dayOfWeek=1)
        .collect(f | f.delay)
        .avg()
```

Solutions (1)

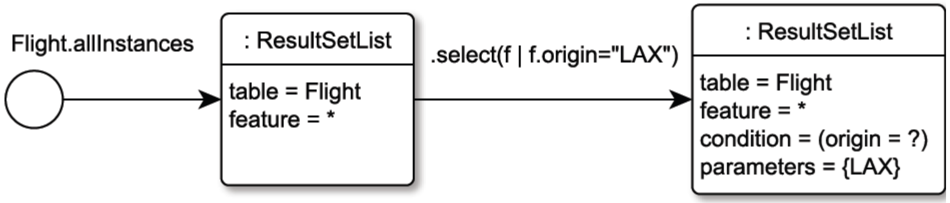
Calculate the average delay of flights flying from JFK to LAX on Sundays:



```
Flight.allInstances
        .select(f | f.origin="LAX")
        .select(f | f.dest="JFK"
         and f.dayOfWeek=1)
        .collect(f | f.delay)
        .avg()
```

Solutions (1)

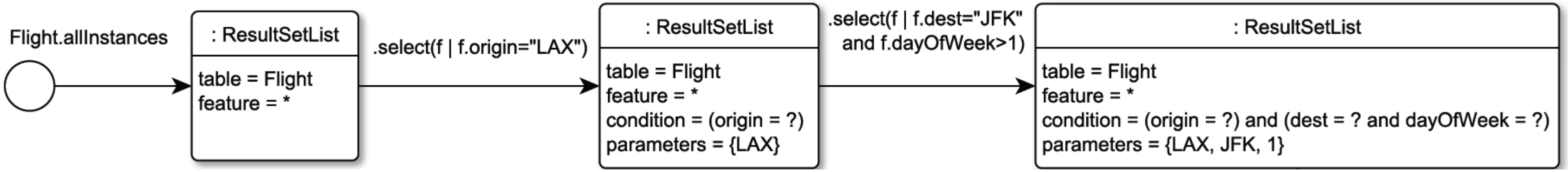
Calculate the average delay of flights flying from JFK to LAX on Sundays:



```
Flight.allInstances
        .select(f | f.origin=“LAX”)
        .select(f | f.dest=“JFK”
         and f.dayOfWeek=1)
        .collect(f | f.delay)
        .avg()
```

Solutions (1)

Calculate the average delay of flights flying from JFK to LAX on Sundays:



Flight.allInstances

: ResultSetList

table = Flight
feature = *

.select(f | f.origin="LAX")

: ResultSetList

table = Flight
feature = *
condition = (origin = ?)
parameters = {LAX}

.select(f | f.dest="JFK" and f.dayOfWeek>1)

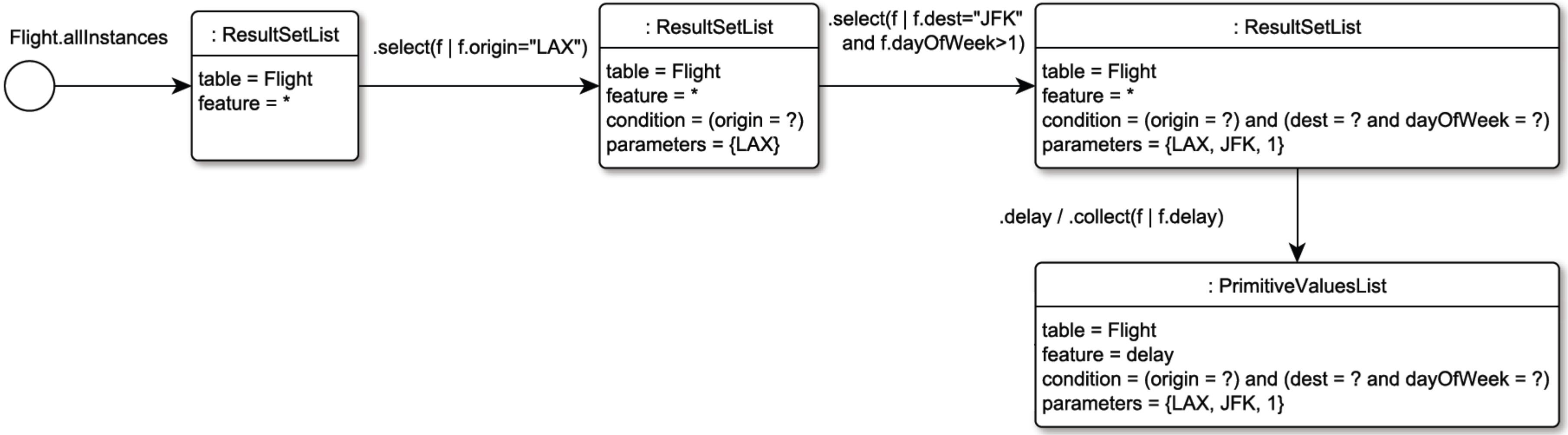
: ResultSetList

table = Flight
feature = *
condition = (origin = ?) and (dest = ? and dayOfWeek = ?)
parameters = {LAX, JFK, 1}

```
Flight.allInstances
        .select(f | f.origin=“LAX”)
        .select(f | f.dest=“JFK”
         and f.dayOfWeek=1)
        .collect(f | f.delay)
        .avg()
```

Solutions (1)

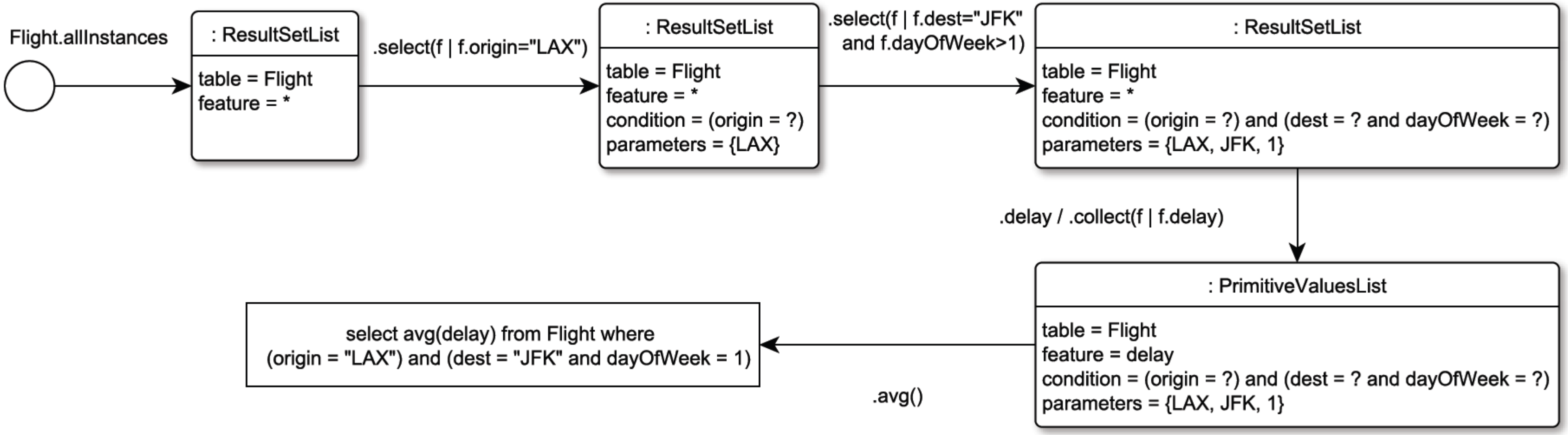
Calculate the average delay of flights flying from JFK to LAX on Sundays:



```
Flight.allInstances
        .select(f | f.origin=“LAX”)
        .select(f | f.dest=“JFK”
         and f.dayOfWeek=1)
        .collect(f | f.delay)
        .avg()
```

Solutions (1)

Calculate the average delay of flights flying from JFK to LAX on Sundays:



Flight.allInstances

: ResultSetList
table = Flight
feature = *

.select(f | f.origin="LAX")

: ResultSetList
table = Flight
feature = *
condition = (origin = ?)
parameters = {LAX}

.select(f | f.dest="JFK" and f.dayOfWeek>1)

: ResultSetList
table = Flight
feature = *
condition = (origin = ?) and (dest = ? and dayOfWeek = ?)
parameters = {LAX, JFK, 1}

.delay / .collect(f | f.delay)

: PrimitiveValuesList
table = Flight
feature = delay
condition = (origin = ?) and (dest = ? and dayOfWeek = ?)
parameters = {LAX, JFK, 1}

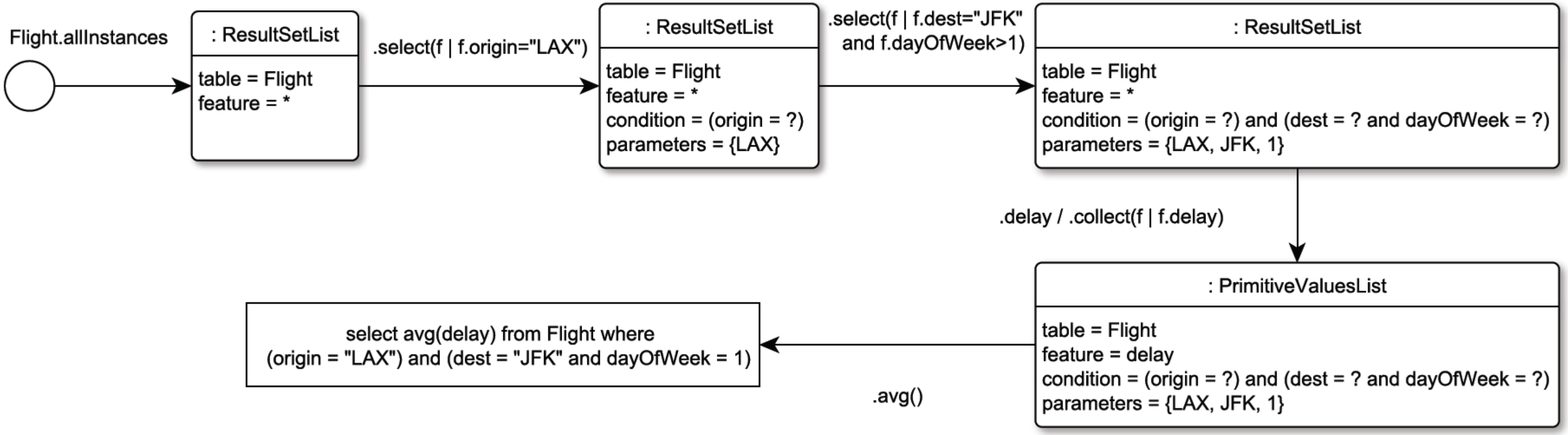
.avg()

select avg(delay) from Flight where
(origin = "LAX") and (dest = "JFK" and dayOfWeek = 1)

```
Flight.allInstances
    .select(f | f.origin=“LAX”)
    .select(f | f.dest=“JFK”
     and f.dayOfWeek=1)
    .collect(f | f.delay)
    .avg()
```

Solutions (1)

Calculate the average delay of flights flying from JFK to LAX on Sundays:



Flight.allInstances

: ResultSetList
table = Flight
feature = *

.select(f | f.origin="LAX")

: ResultSetList
table = Flight
feature = *
condition = (origin = ?)
parameters = {LAX}

.select(f | f.dest="JFK" and f.dayOfWeek>1)

: ResultSetList
table = Flight
feature = *
condition = (origin = ?) and (dest = ? and dayOfWeek = ?)
parameters = {LAX, JFK, 1}

.delay / .collect(f | f.delay)

: PrimitiveValuesList
table = Flight
feature = delay
condition = (origin = ?) and (dest = ? and dayOfWeek = ?)
parameters = {LAX, JFK, 1}

.avg()

select avg(delay) from Flight where
(origin = "LAX") and (dest = "JFK" and dayOfWeek = 1)

```
Flight.allInstances
        .select(f | f.origin=“LAX”)
        .select(f | f.dest=“JFK”
         and f.dayOfWeek=1)
        .collect(f | f.delay)
        .avg()
```

```
select avg(delay) from Flight where
(origin=“LAX”)
and
(dest=“JFK” and dayOfWeek=1)
```

Solutions (2)

EOL Engine Extension for SQL:

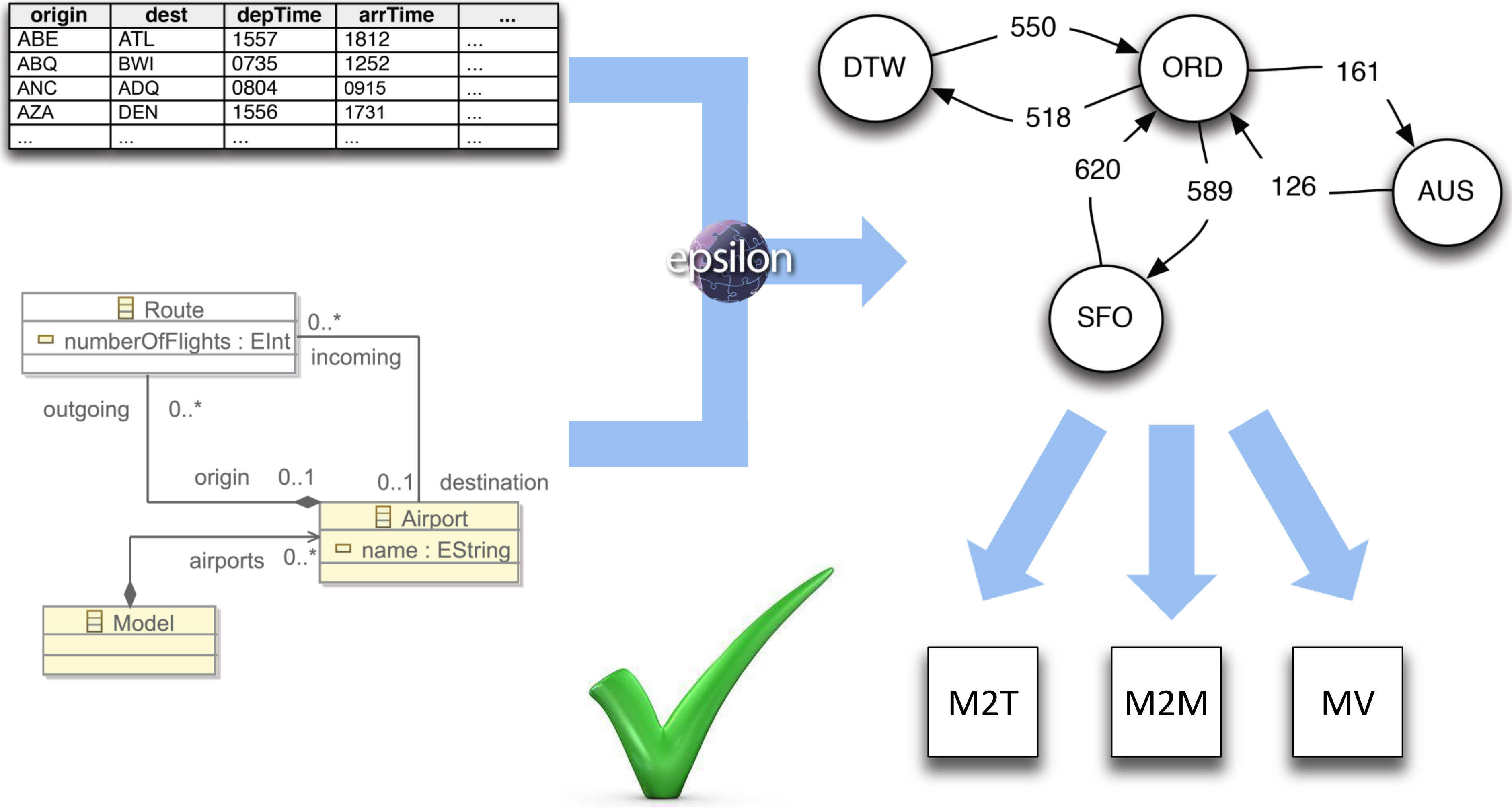
.allInstances Returns a streamed lazy collection (*ResultSetList*) backed by a *select * from <table>* SQL expression.

.select(<iterator>|<condition>) Translates the EOL condition to an SQL expression, and returns a new *ResultSetList*. Similarly for exists(), forAll() and reject() OCL operations.

.collect(<iterator>|<expression>) *R*eturns a streamed lazy collection of primitive values (*PrimitiveValuesList*). Calls to the size() method are interpreted as count SQL queries.

asSet() Returns a new *PrimitiveValuesList* backed by a distinct SQL query.

The ATM System



origin	dest	depTime	arrTime	...
ABE	ATL	1557	1812	...
ABQ	BWI	0735	1252	...
ANC	ADQ	0804	0915	...
AZA	DEN	1556	1731	...
...	...	...	...	...

Extracted Facts

Analysis of this dataset reveals:

- Of the 306 airports, 68 (>20%) are connected directly to only 1 other airport;
- The most distant pair of airports are ABE and BRW. A passenger needs to change 4 flights (ABE-DTW-SEA-FAI-BRW);
- The Atlanta International Airport (ATL) is the busiest airport (# of flights going through it - 67,717), followed by ORD and DFW;
- ATL is the best-connected airport with direct flights to 148 other airports;
- >50% of all the flights go through the 18 busiest airports & >90% of all flights go through the 91 busiest airports.

Conclusion & Further Work

- MDE can greatly benefit from using technologies outside MOF and EMF
- If integrated correctly, relational datasets can be used to contain model data
- The challenges lay in identifying and optimising the way such stores are queried

- We aim at investigating the impact of compile-time static analysis on performance
- We aim at supporting multi-table querying (and hence transformations) by use of foreign keys

Questions?