InterMine

Object integration and warehousing software

FlyMine

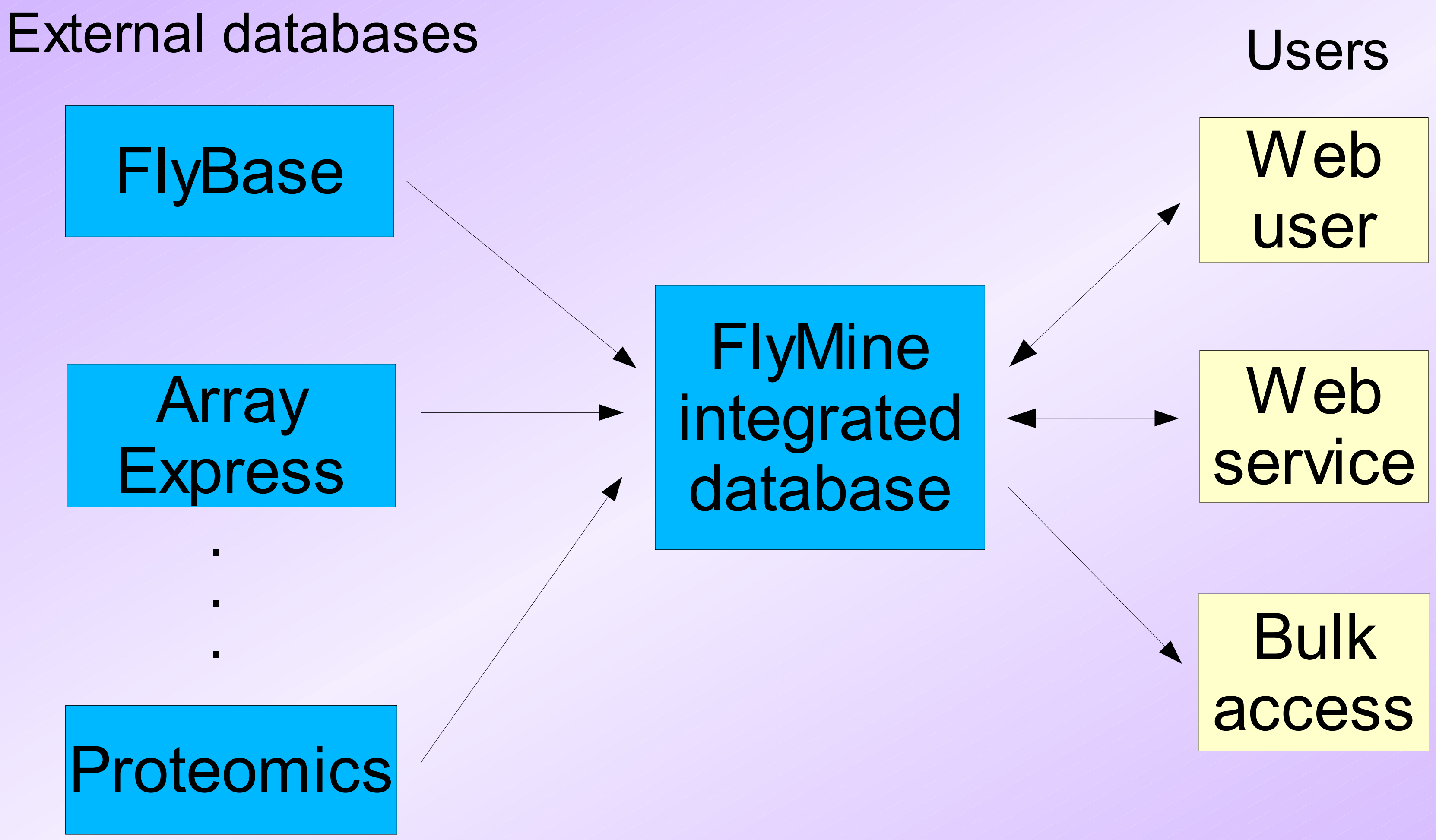
An integrated database for *Drosophila* and *Anopheles* genomics

Andrew Varley,
Genetics Department,
Cambridge University

Why yet another database?

- Currently lots of little databases
  - fine for “browsing”, bad for “querying”
- Hard to query across them
  - lots of “cut and paste” on web pages
- Massive amounts of experimental data (microarray, proteomics) being produced
- Need to tie this information together

Outline


External databases
FlyBase
Array Express
.
.
.
Proteomics
FlyMine integrated database
Users
Web user
Web service
Bulk access

InterMine - aims

- Integrate data from multiple sources
- Allow arbitrary queries from users
- Queries based on objects, not SQL
- Complex models (multiple inheritence)
- Multiple query interfaces (Java, OQL, graphical, HTML, etc)
- Different classes of user (web-based, SOAP)
- Open source!

Query interfaces – OQL

“Show gene expression data for genes which have GO term GO:0000278 applied”

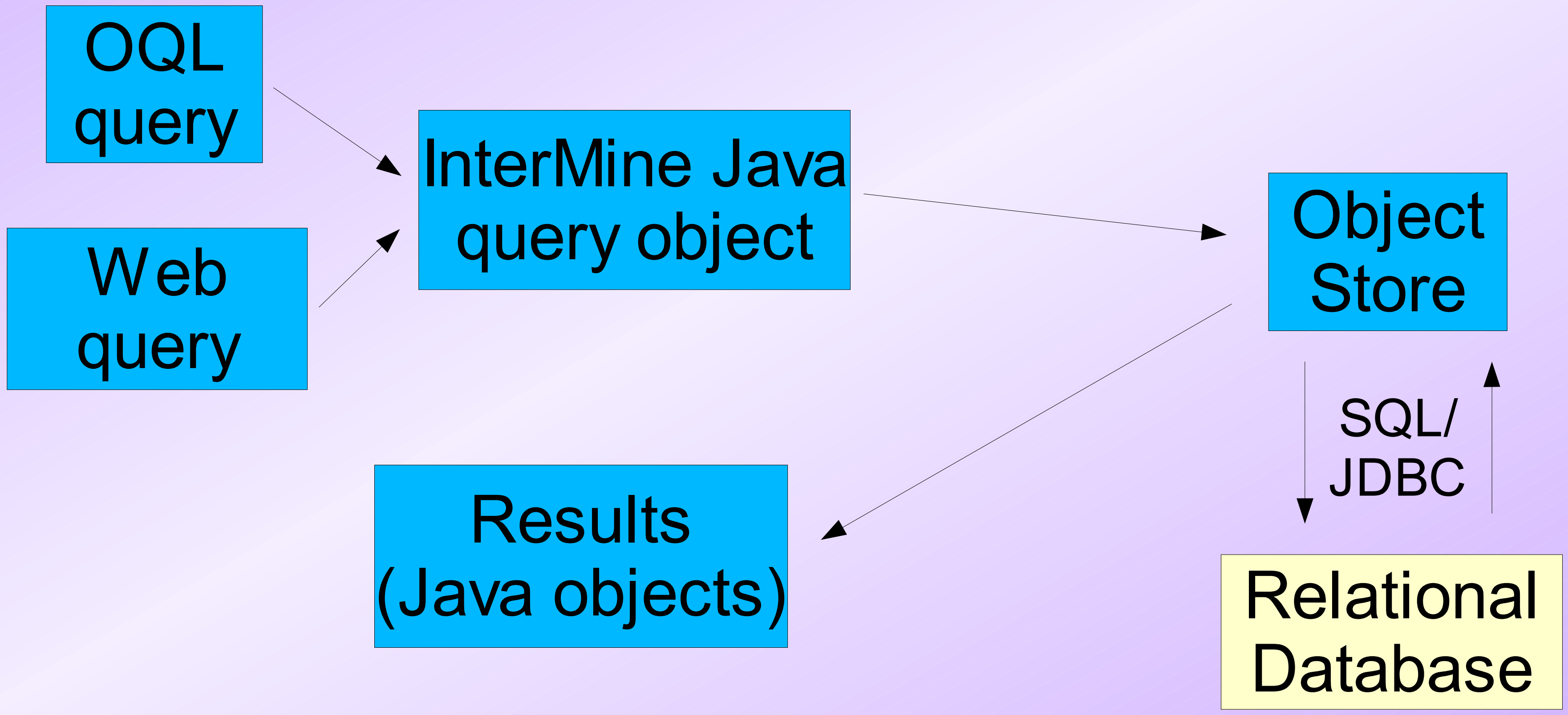
```
SELECT gene, exp
FROM Gene AS gene, GOTerm AS term,
     Experiment AS exp
WHERE gene.GOTerms CONTAINS term
AND term.code = “GO:0000278”
AND gene.experiments CONTAINS exp
AND exp.type = “Gene Expression”
```

Query interfaces – Java

“Show gene expression data for genes which have GO term GO:0000278 applied”

```
Query q = new Query();
QueryClass qcGene = new QueryClass(Gene.class);
QueryClass qcTerm = new QueryClass(GOTerm.class);
QueryField qfGeneTerm = new QueryField(qcGene, “GOTerms”
ConstraintSet cs = new ConstraintSet();
Constraint con1 = new ContainsConstraint(qfGeneTerm,
                  ContainsConstraint.CONTAINS, qcTerm);
cs.add(con1);
.
.
q.setConstraint(cs);
```

InterMine ObjectStore

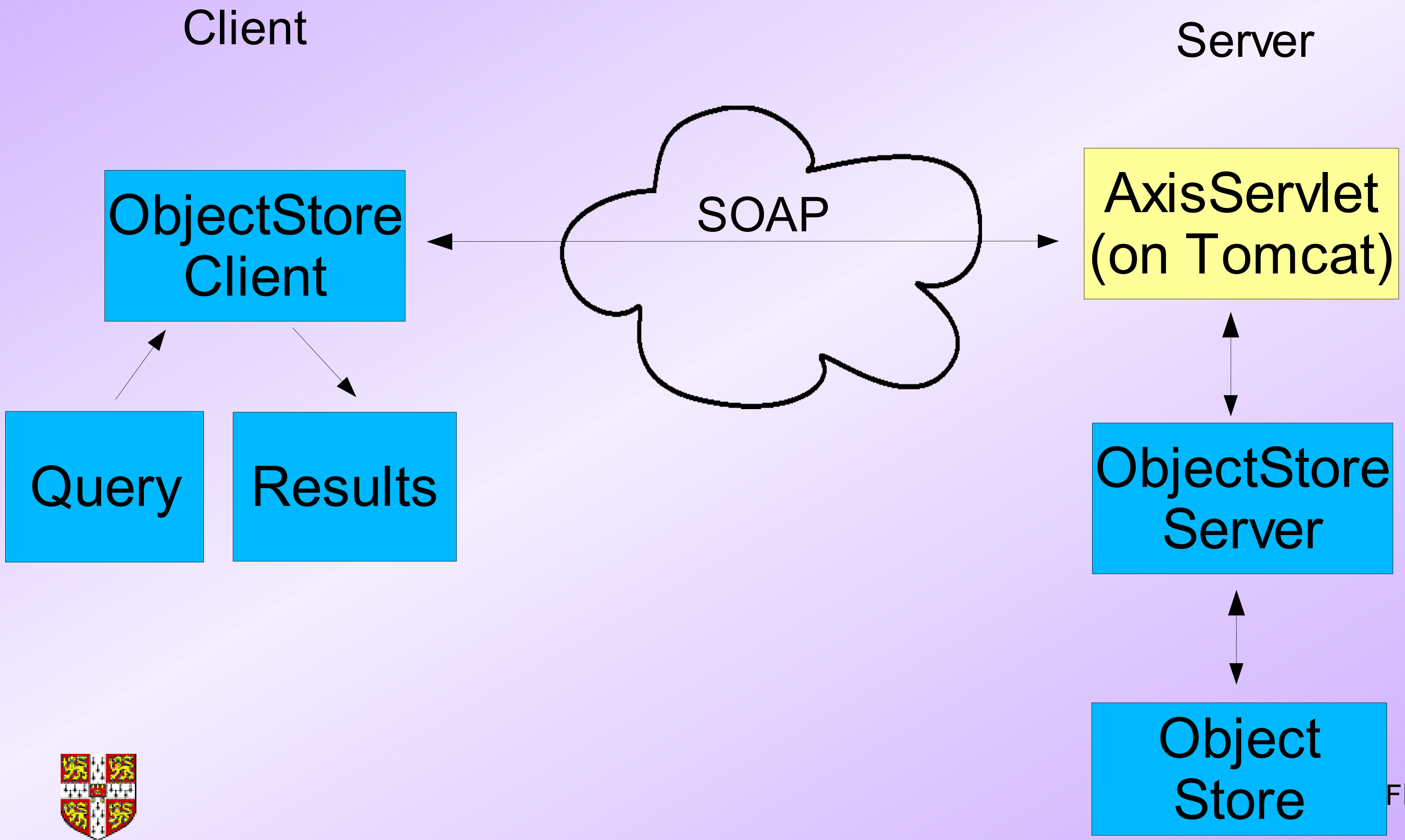


ObjectStore interface

```
public interface ObjectStore {

  Results execute(Query q);
  List execute(Query q, int start, int limit);
  int count(Query q);
  ResultsInfo estimate(Query q);
  Object getObjectById(Integer id);
  Model getModel();
}
```

Web service

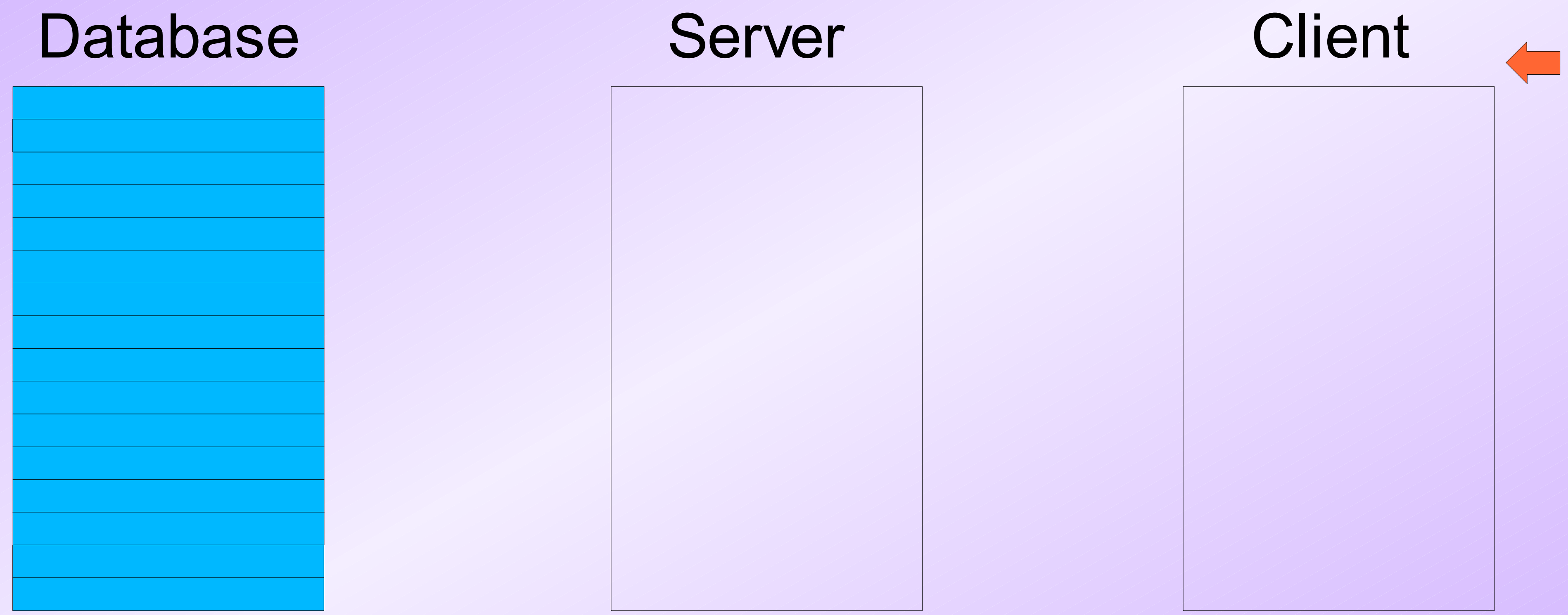


FlyMine

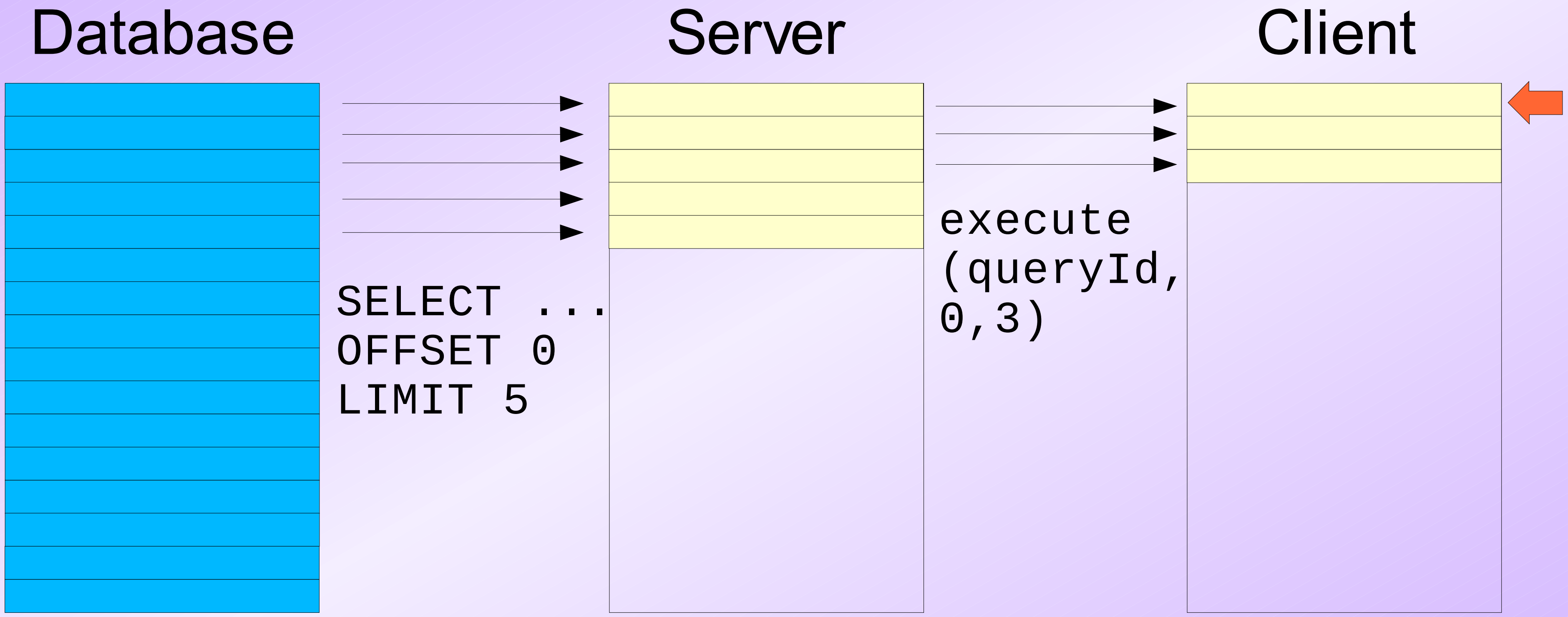
InterMine ObjectStore

- Results are tables of Java objects
  - All collections and references are proxied
- Results rows are fetched in batches from the underlying database server
- Results rows are pre-emptively fetched and cached (per JVM).

Results



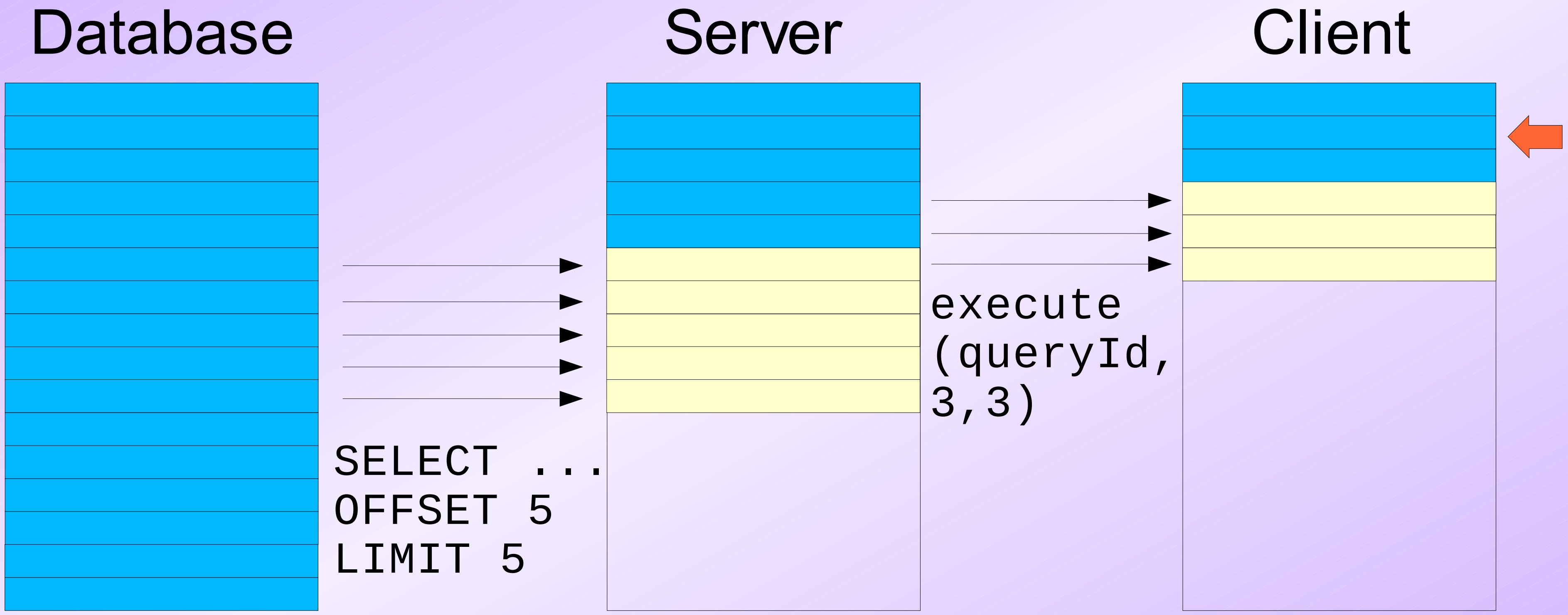
Results



```
SELECT ...
OFFSET 0
LIMIT 5
```

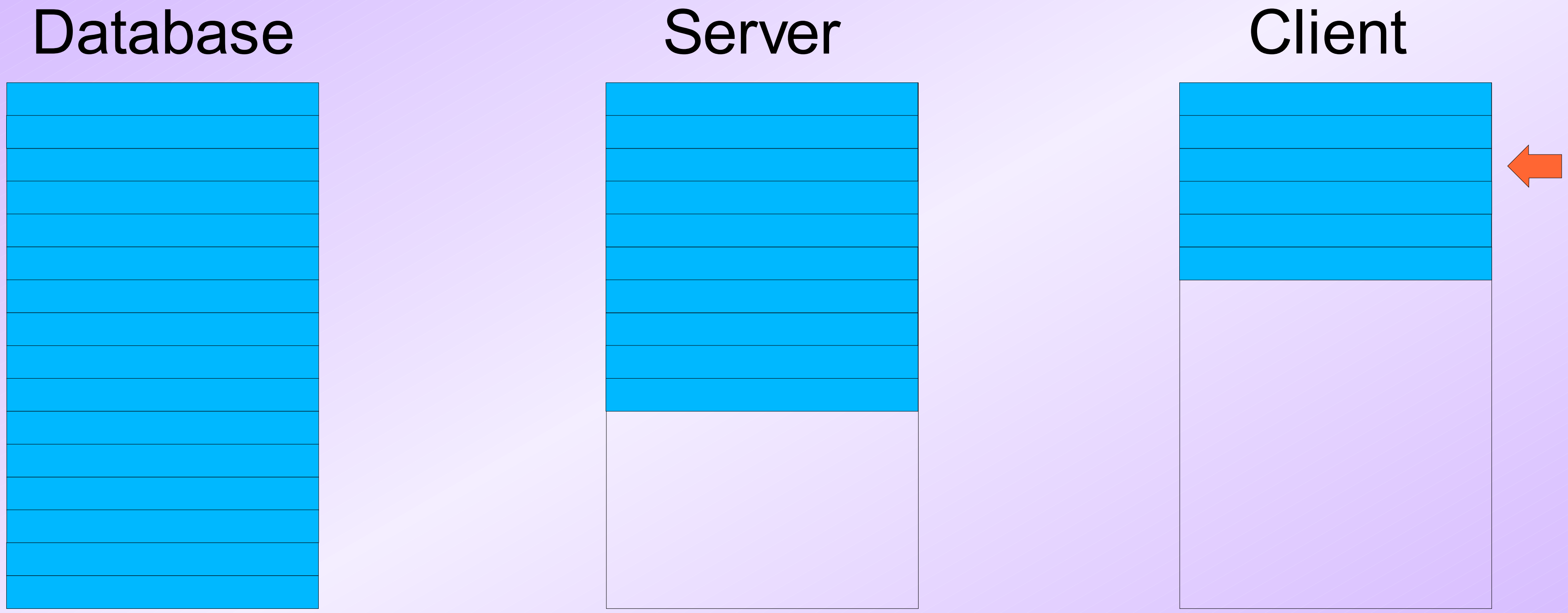
```
execute
(queryId,
0,3)
```

Results

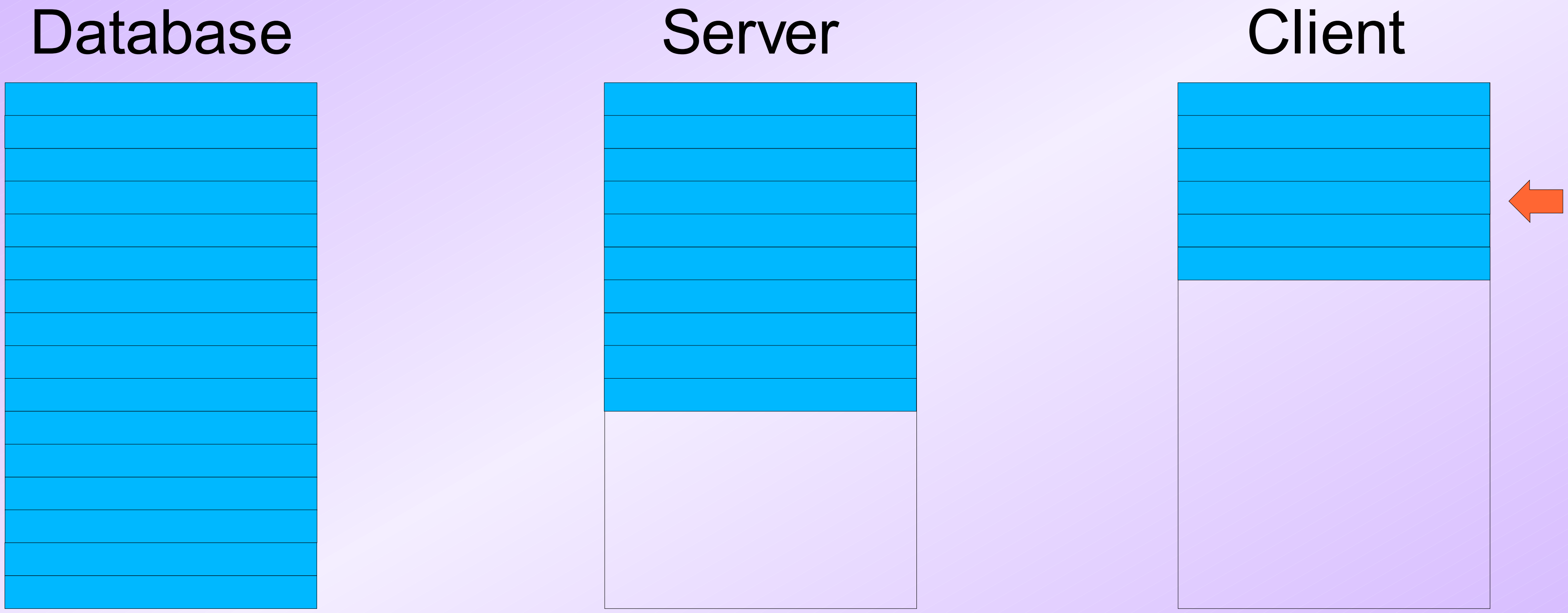


Database
Server
Client
SELECT ...
OFFSET 5
LIMIT 5
execute
(queryId,
3,3)

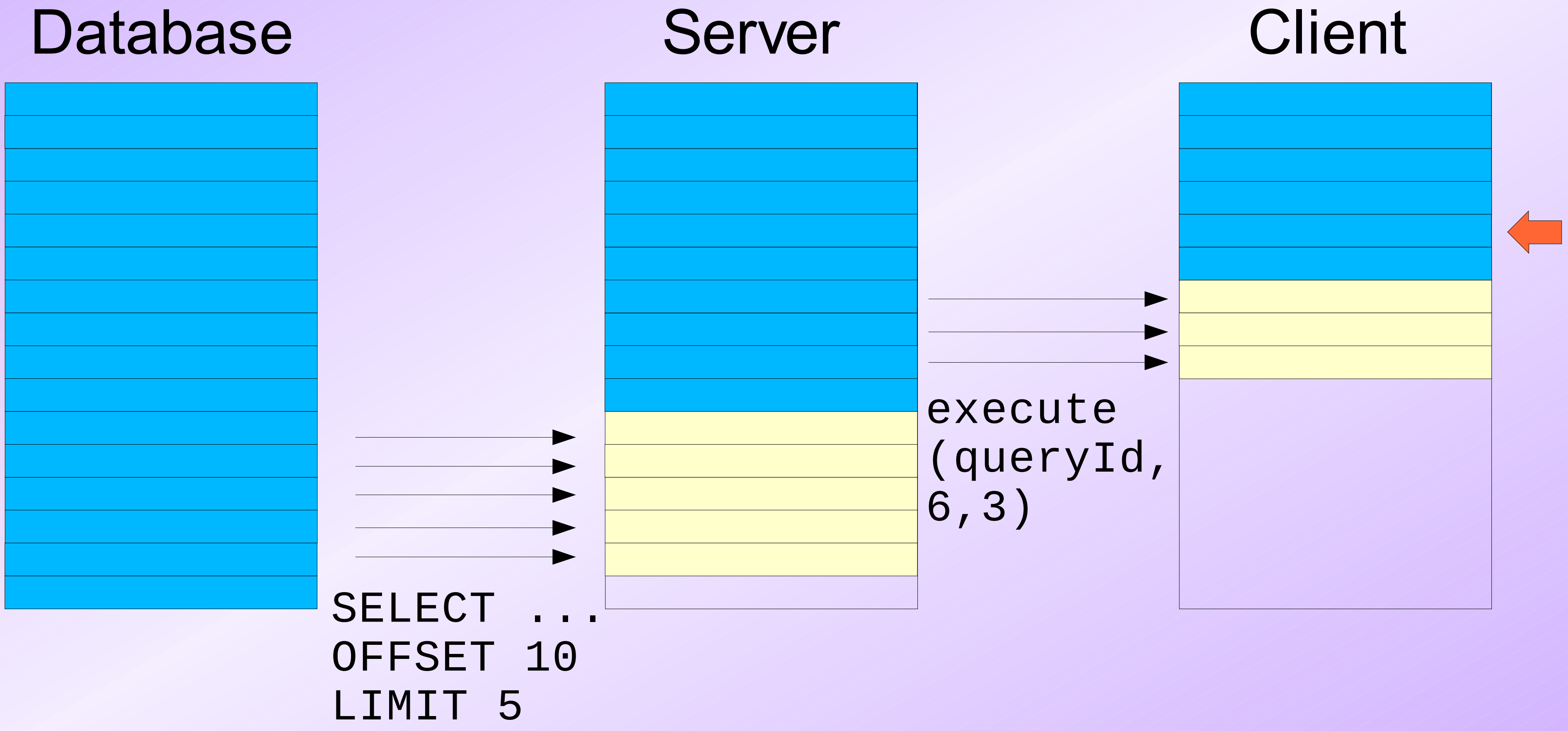
Results

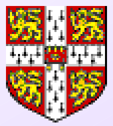


Results



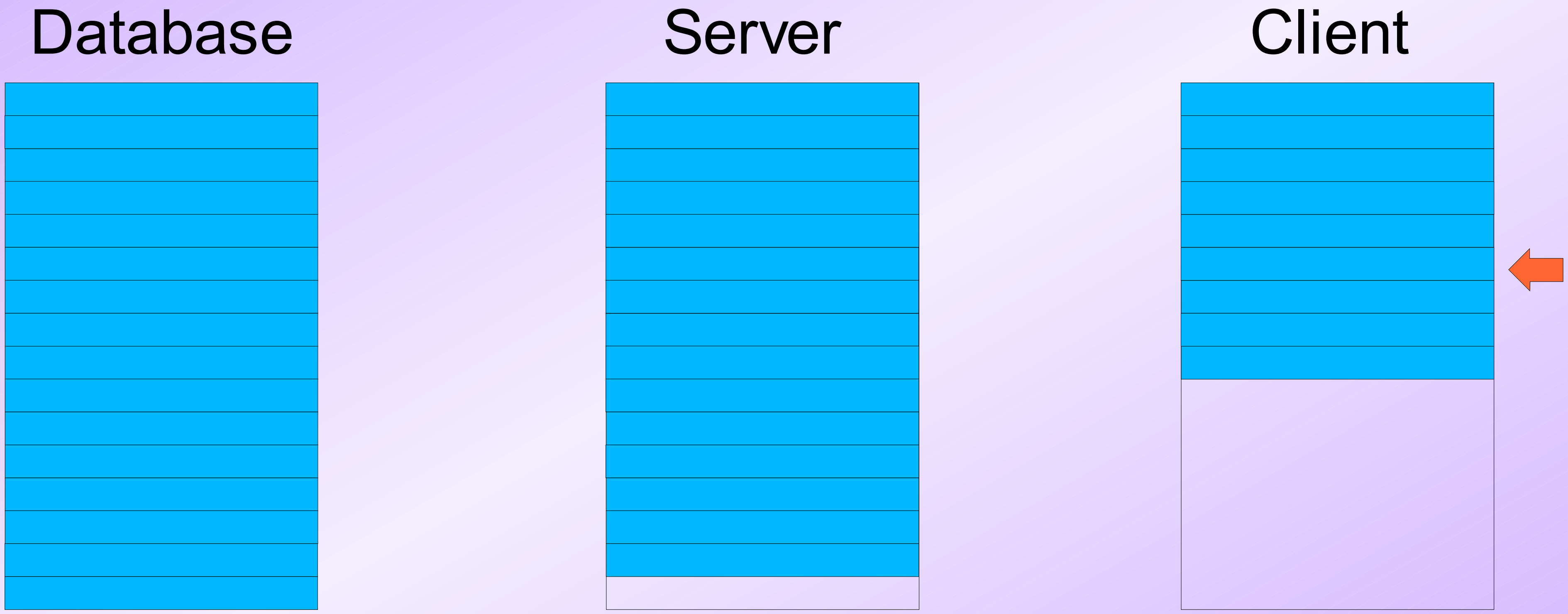
Results



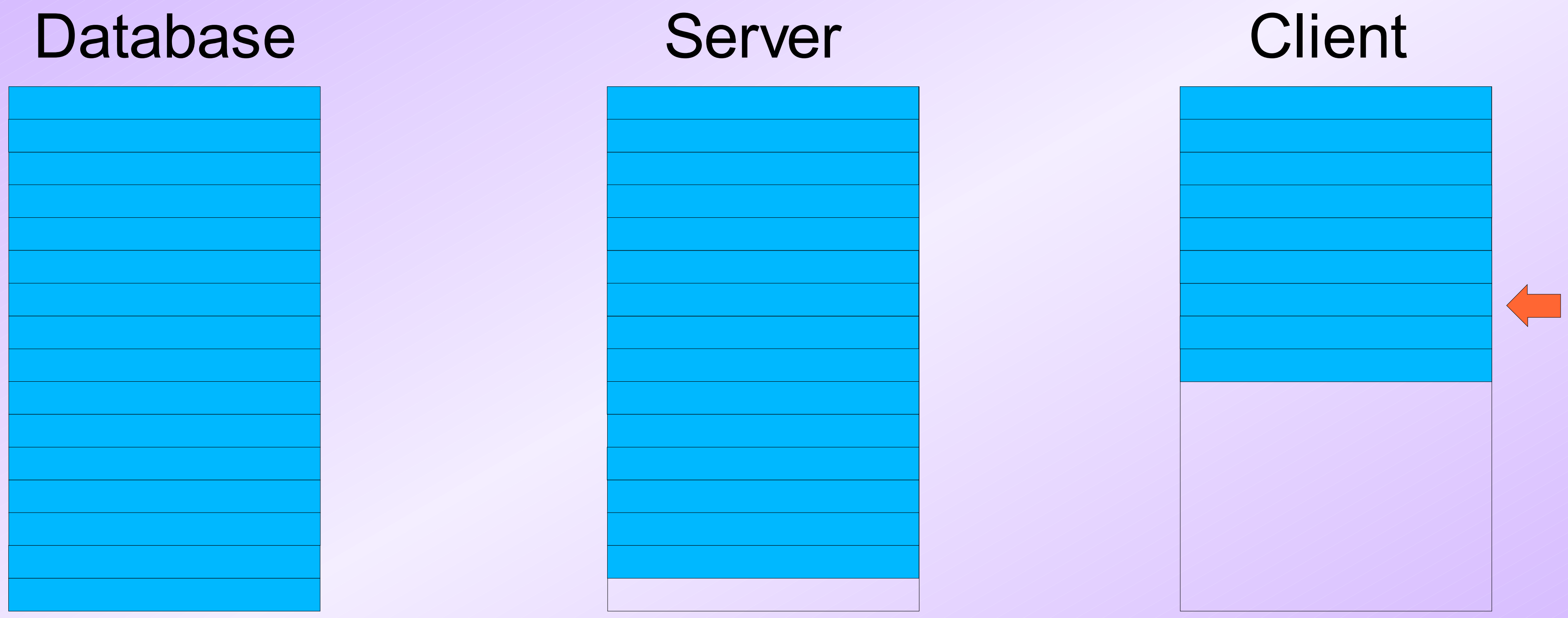


FlyMine

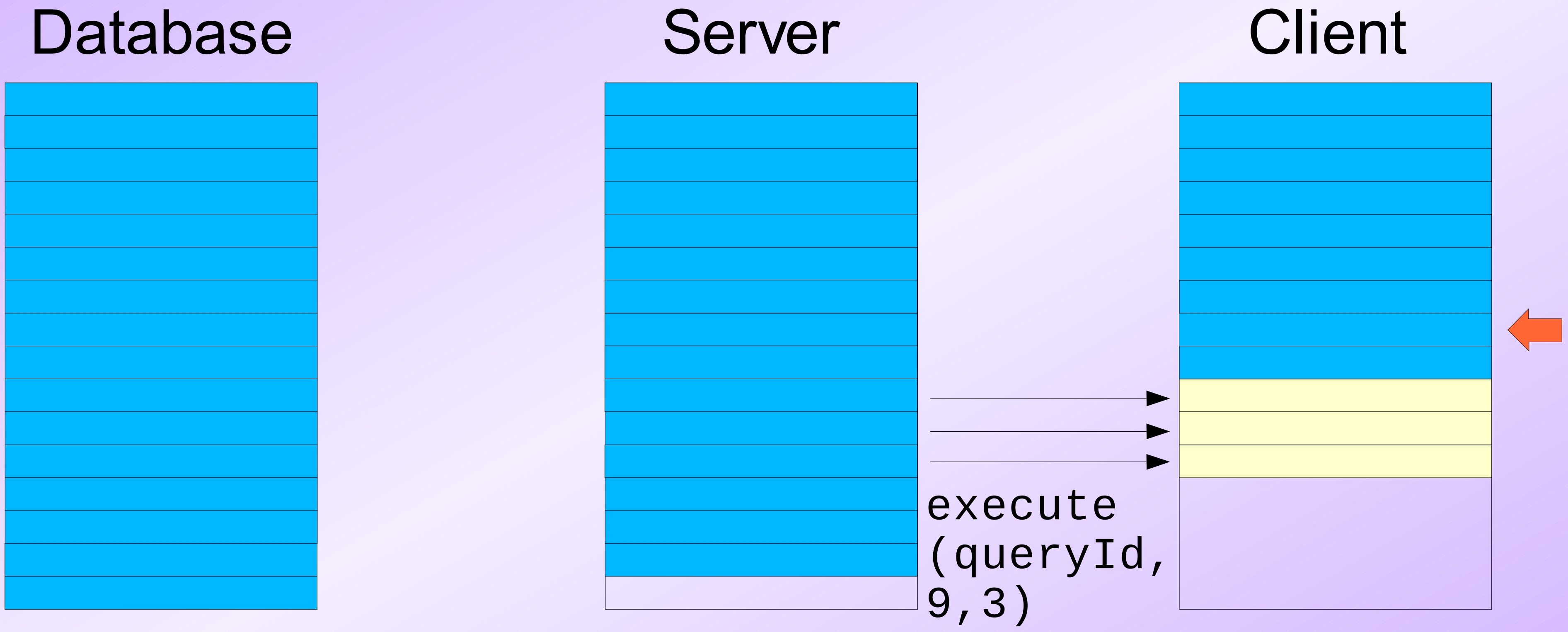
Results



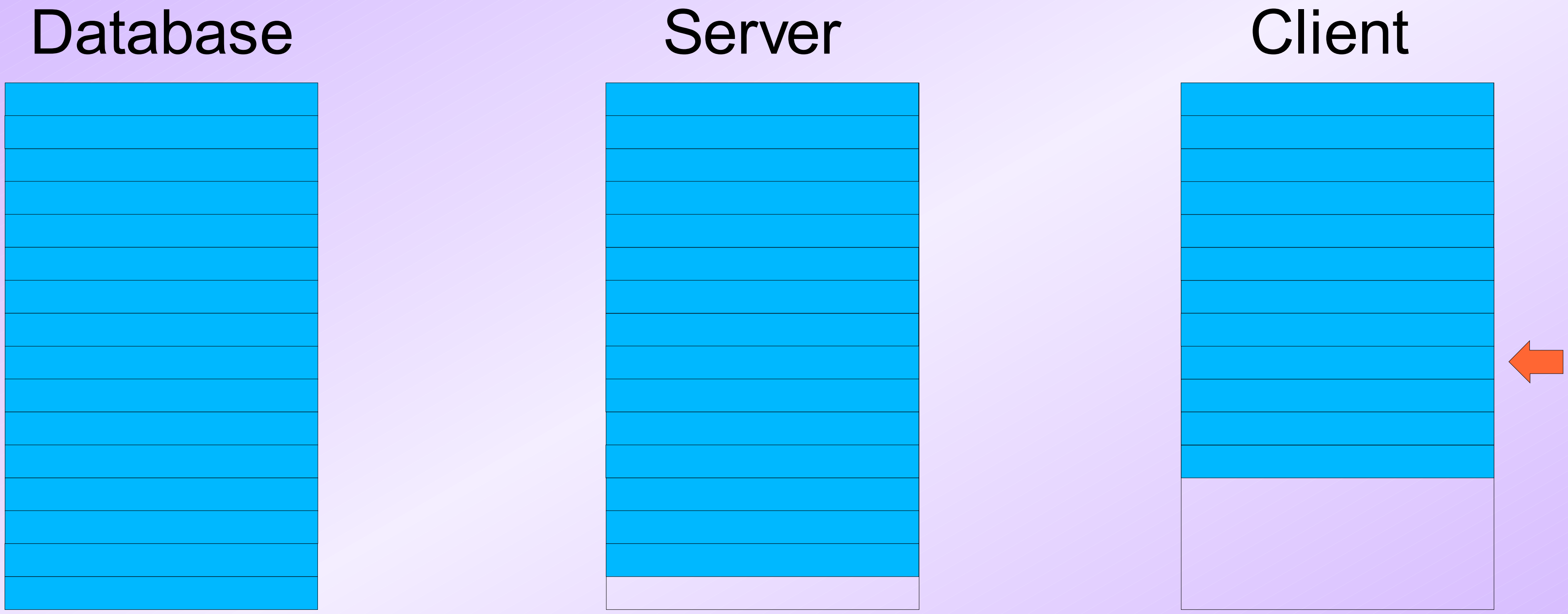
Results



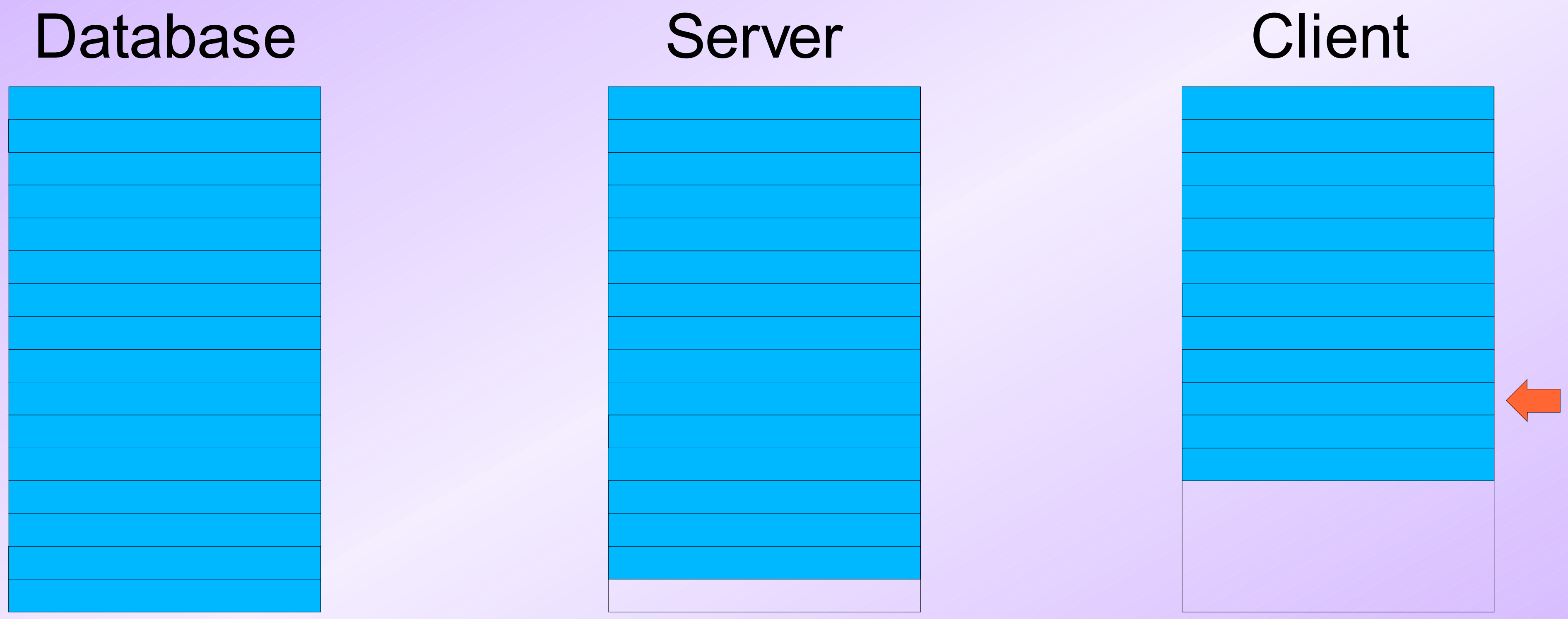
Results



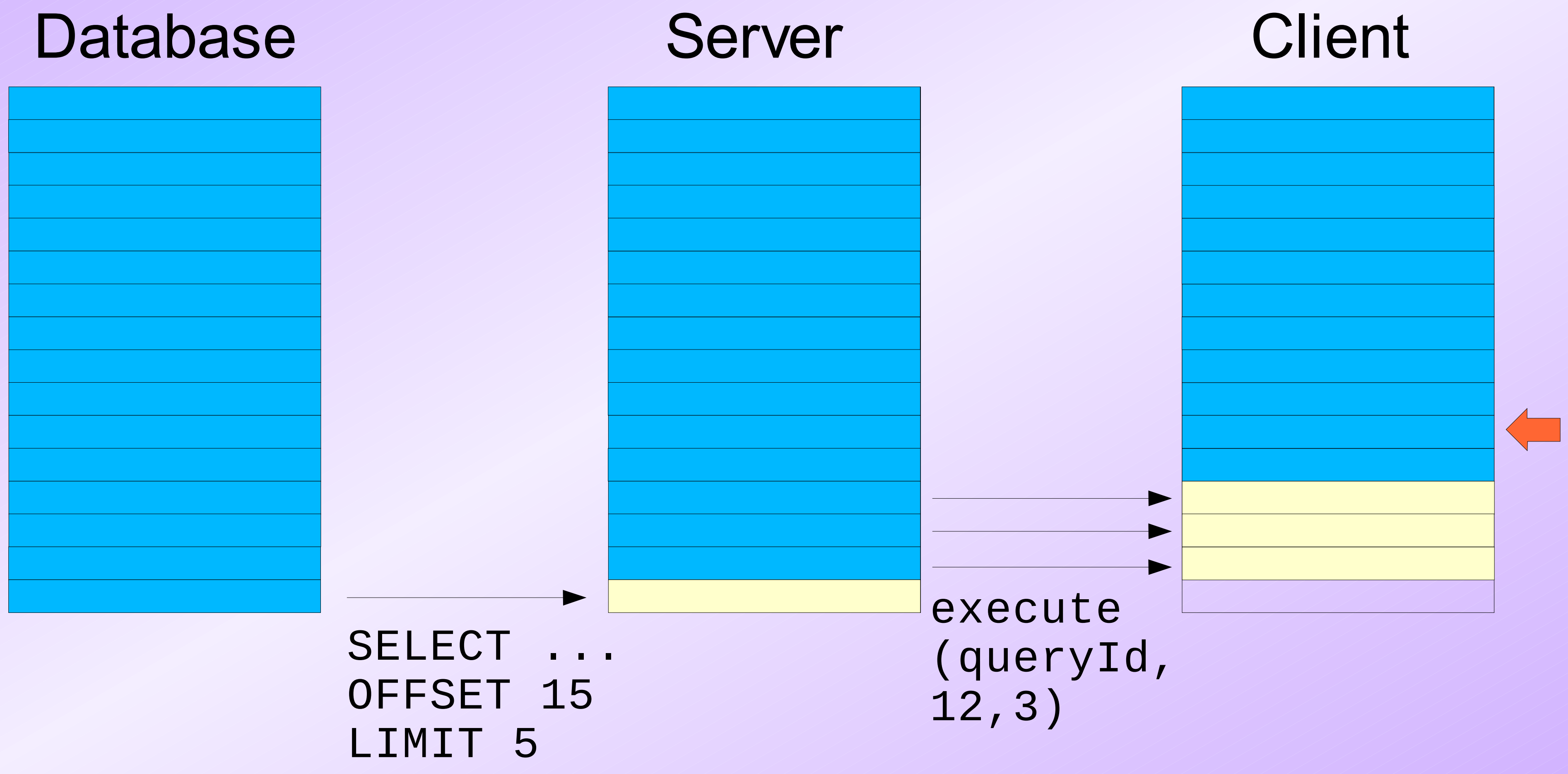
Results



Results



Results



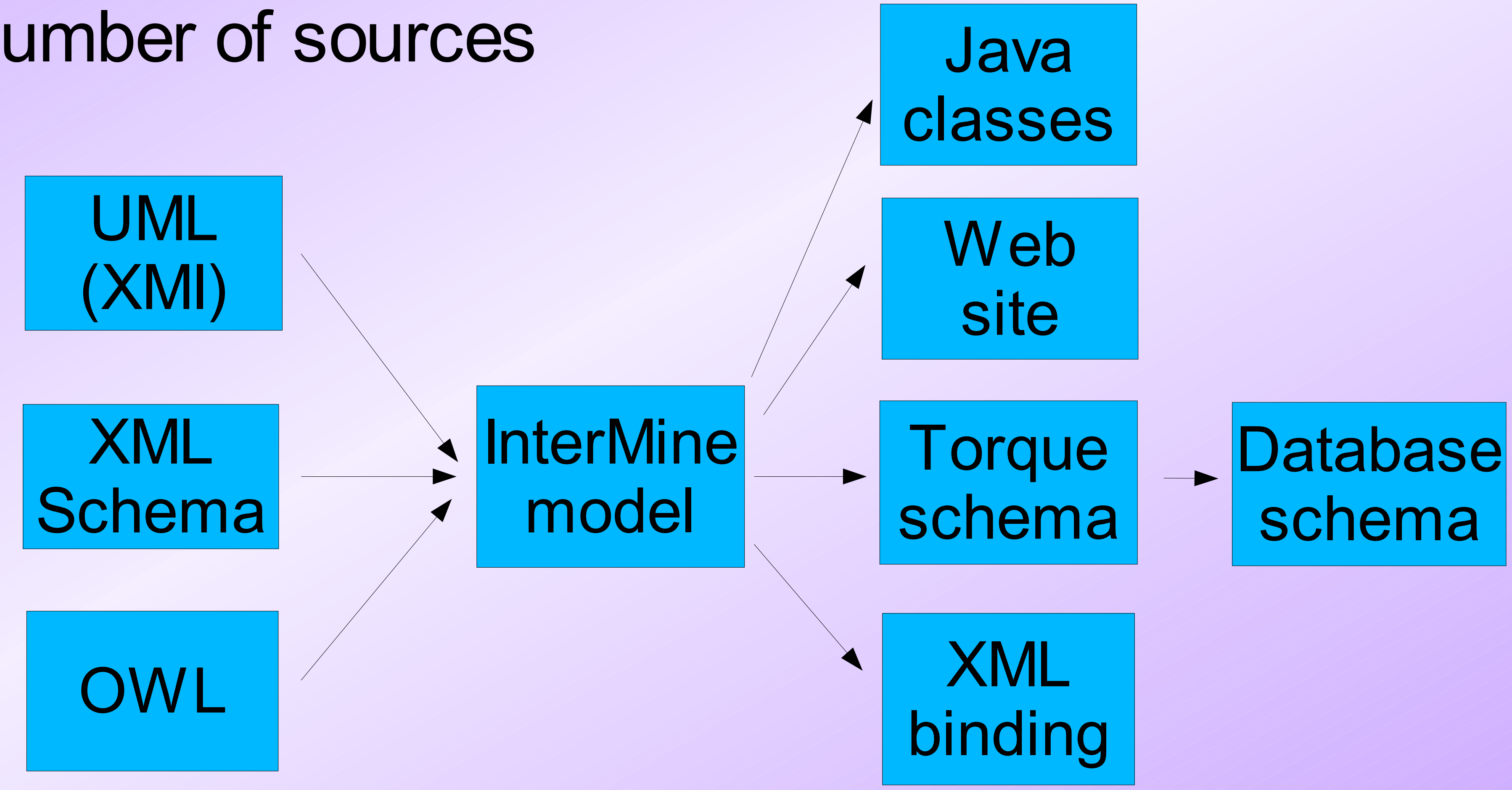
```
SELECT ...
OFFSET 15
LIMIT 5
```

```
execute
(queryId,
12,3)
```

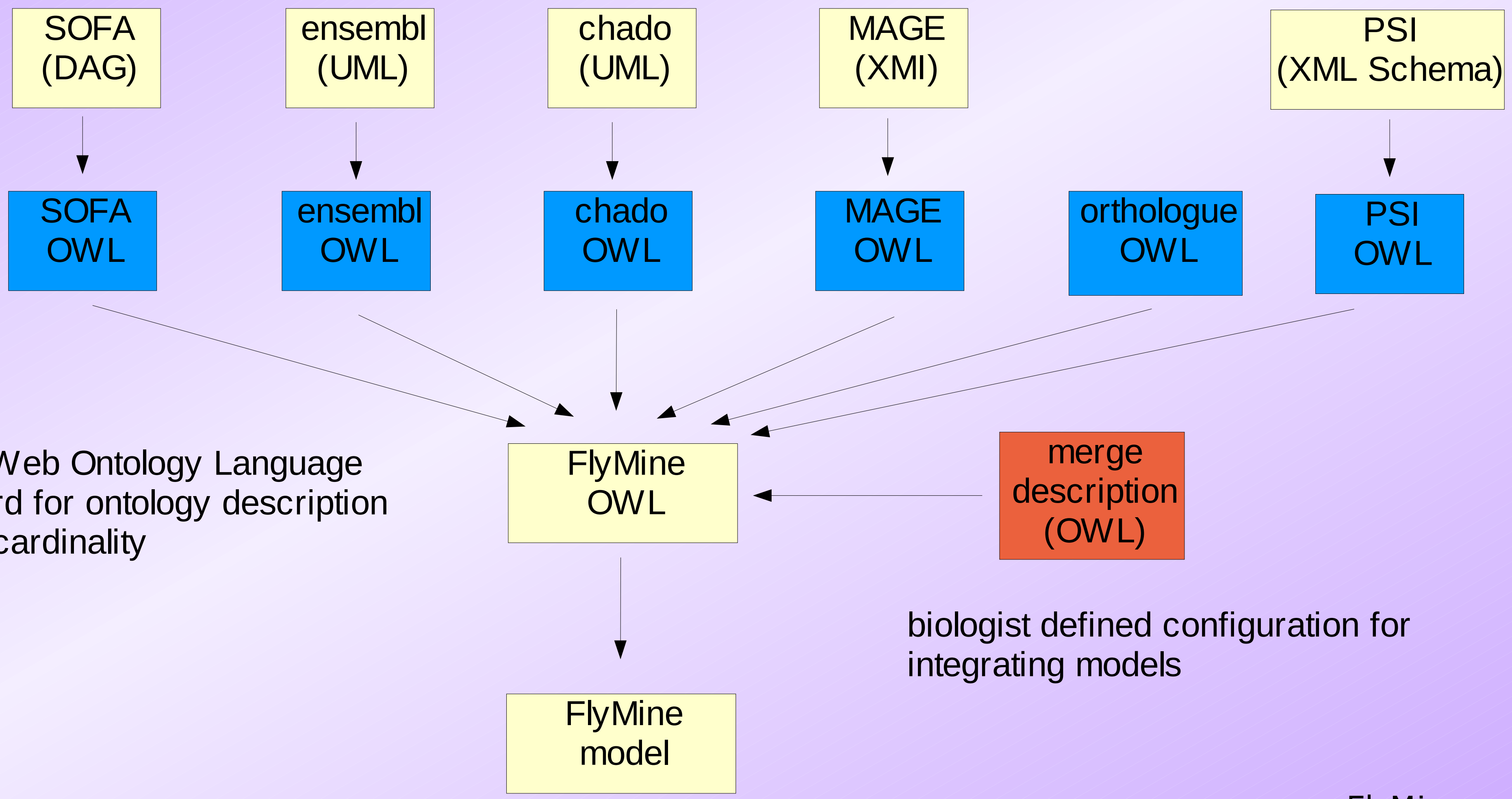
FlyMine

Data model

- Data model can be generated from one of a number of sources



Model Integration



OWL – Web Ontology Language
- standard for ontology description
- allows cardinality

biologist defined configuration for integrating models

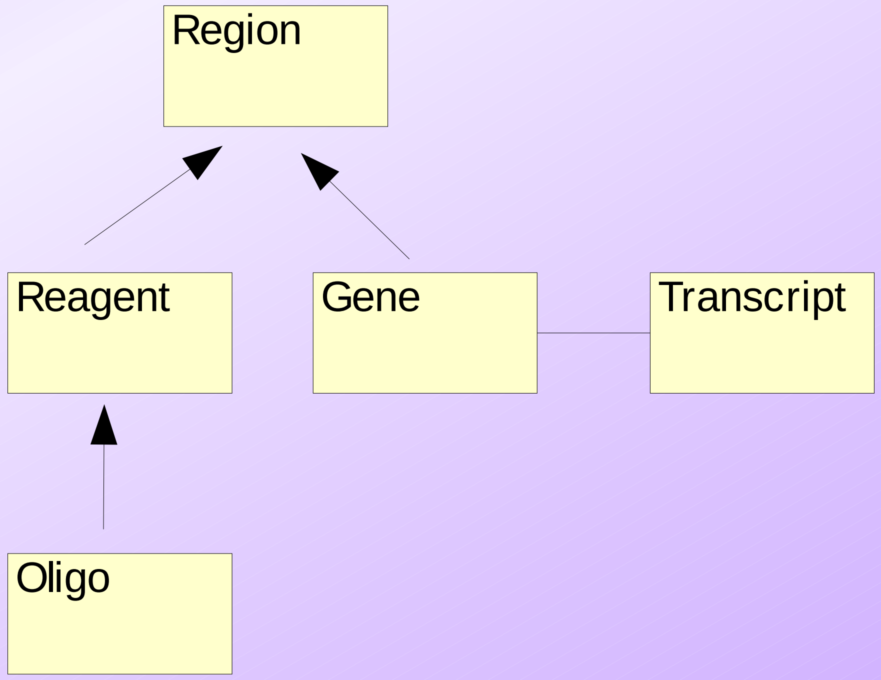
Integrate existing standards

- Link existing and emerging standards to define FlyMine model
  - e.g. SO, MAGE, PSI,GO, other ontologies
- Avoids creating a 'schema of everything'
- Easy to add new types of data
- Evolve as standards change

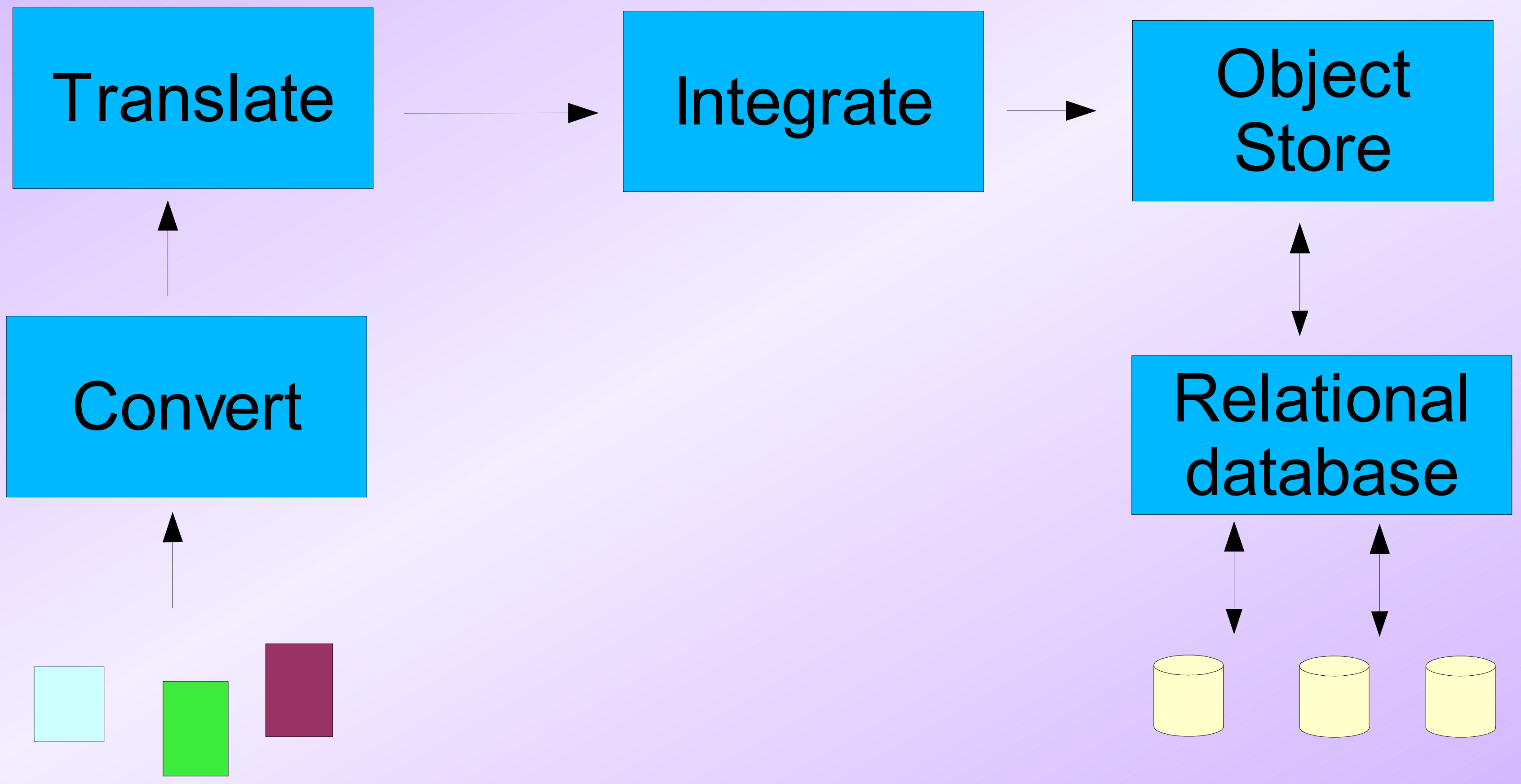
Example – Sequence Ontology

- All terms in ontology become classes in model

```
@is_a@region
   @is_a@gene
      @part_of@transcript
   @is_a@reagent
      @is_a@oligo
```

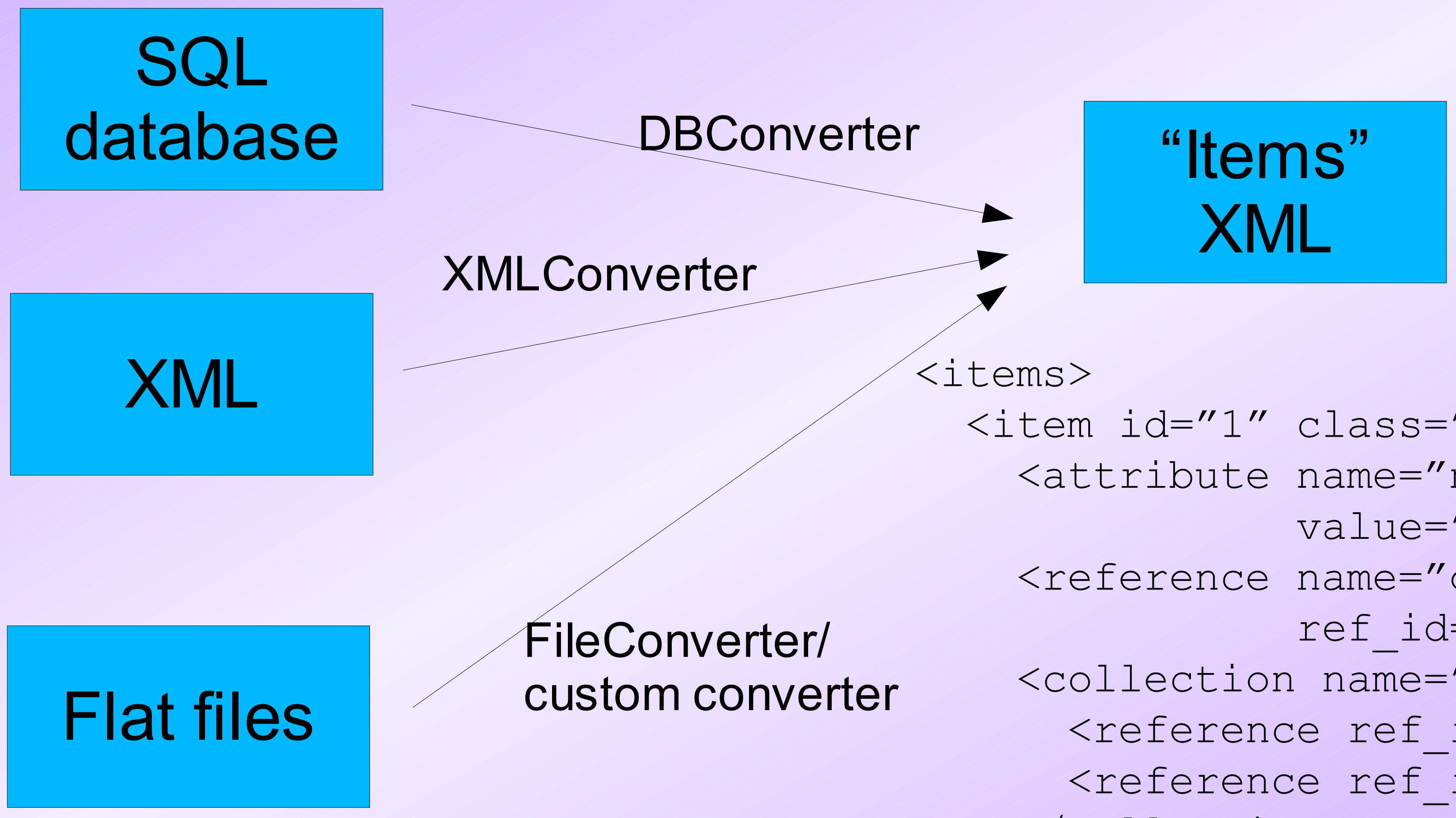


Data loading pipeline



External data sources

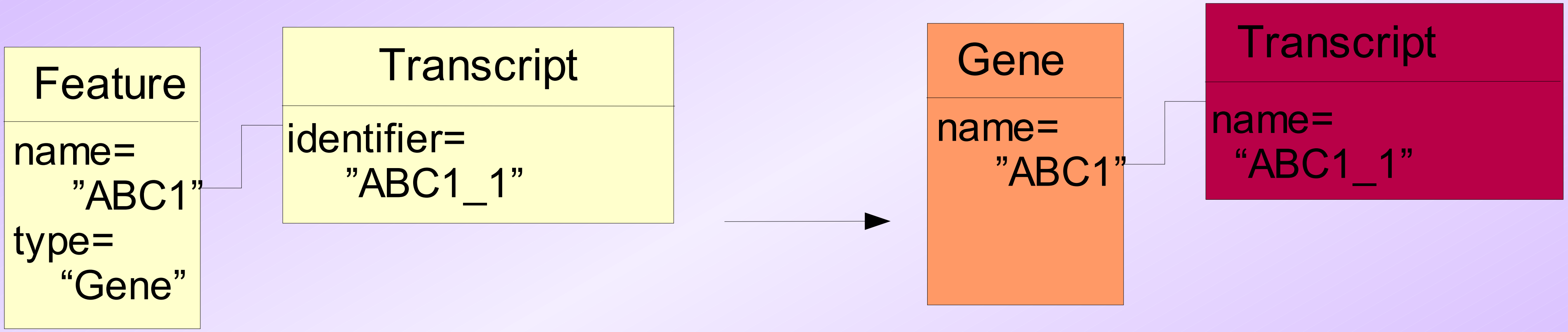
Conversion from external sources



```
<items>
  <item id="1" class="Feature">
    <attribute name="name"
               value="ABC1"/>
    <reference name="organism"
               ref_id="2"/>
    <collection name="synonyms">
      <reference ref_id="3"/>
      <reference ref_id="4"/>
    </collection>
  </item>
        .
        .
</items>
```

FlyMine

Translation



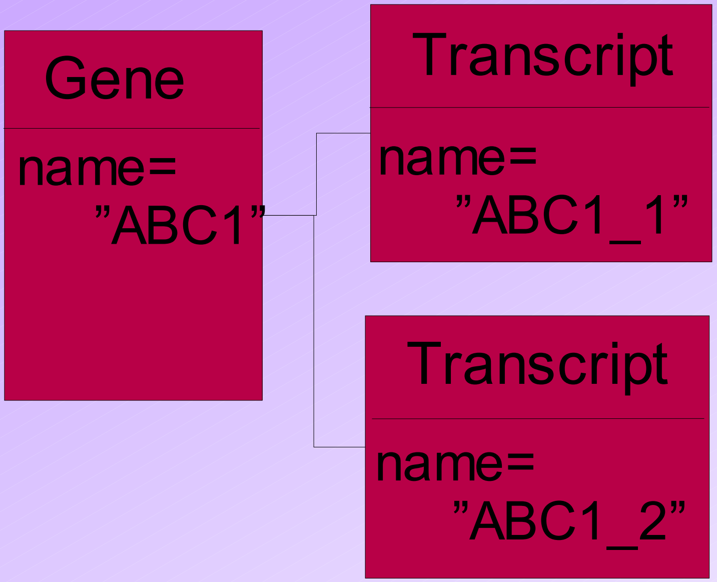
Translation

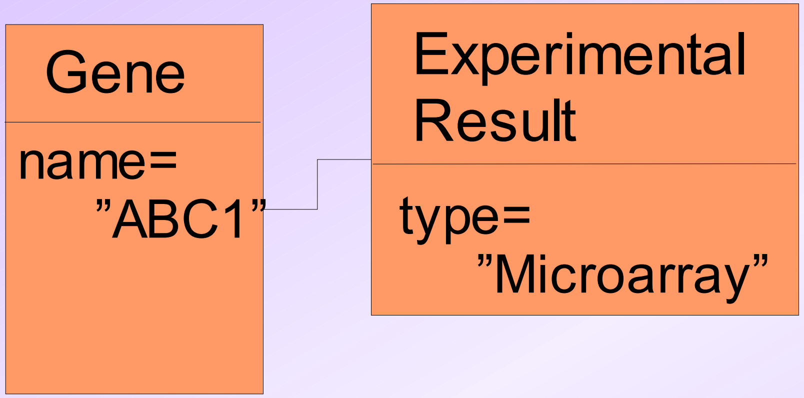
- Controlled by OWL “merge description”

```
:Gene a owl:Class;
  rdfs:subClassOf exampleSrc:Feature ;
  rdfs:subClassOf
    [ a owl:Restriction;
      owl:onProperty exampleSrc:Feature__type
      owl:hasValue “Gene”
     ] .
:Transcript a owl:Class;
  owl:equivalentClass exampleSrc:Transcript .
:Transcript__name a owl:DatatypeProperty;
  owl:equivalentProperty exampleSrc:Transcript__identifier
```

Integration

- Define “primary keys” for each object type
- Define “primary keys” that each source uses
- Define priorities for fields from different sources
- IntegrationWriter keeps track of originating sources for each field of each object



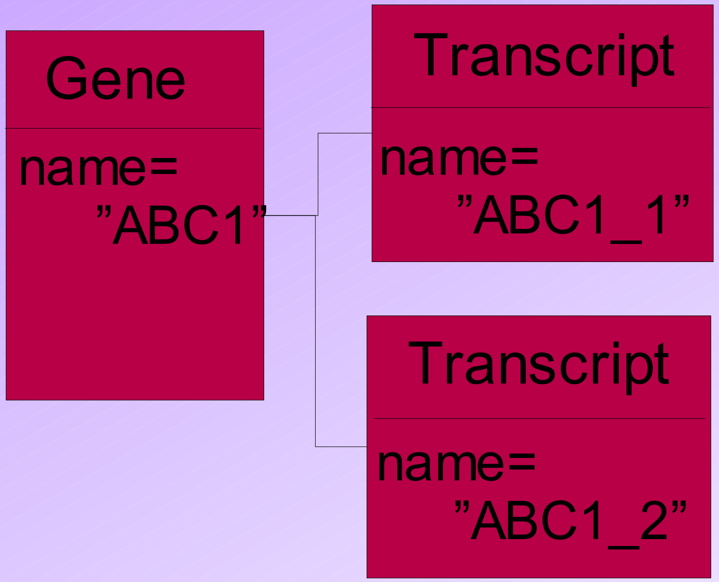


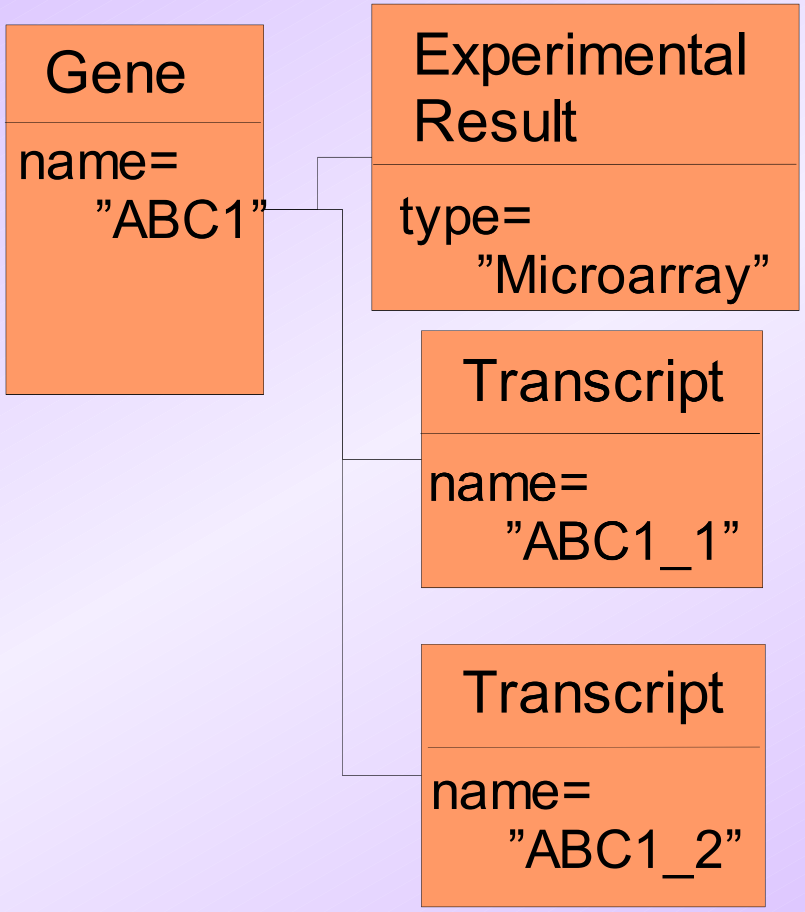
```
Primary keys:
  Gene: name
  Gene: flyBaseName
  Transcript: name,gene
```

New objects

Currently in database

FlyMine

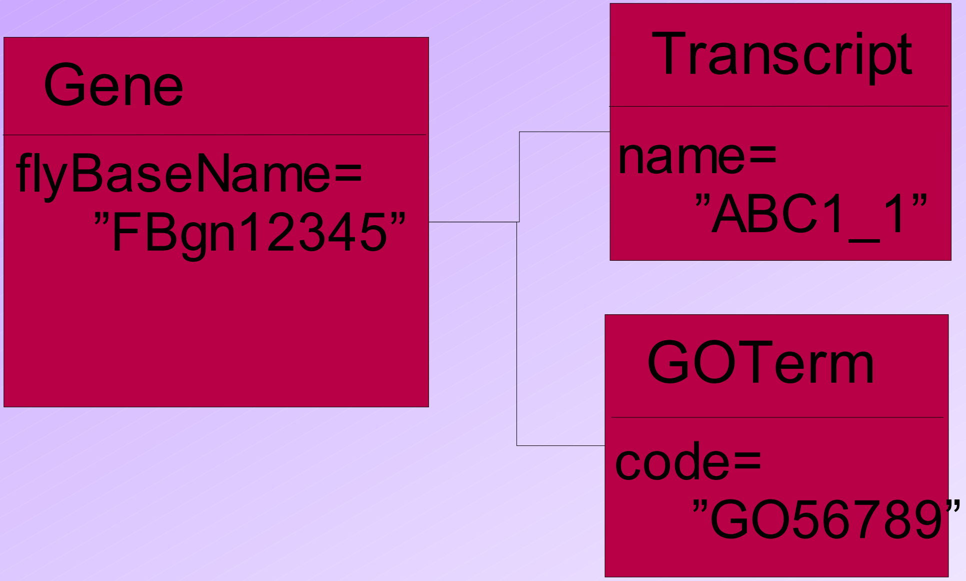


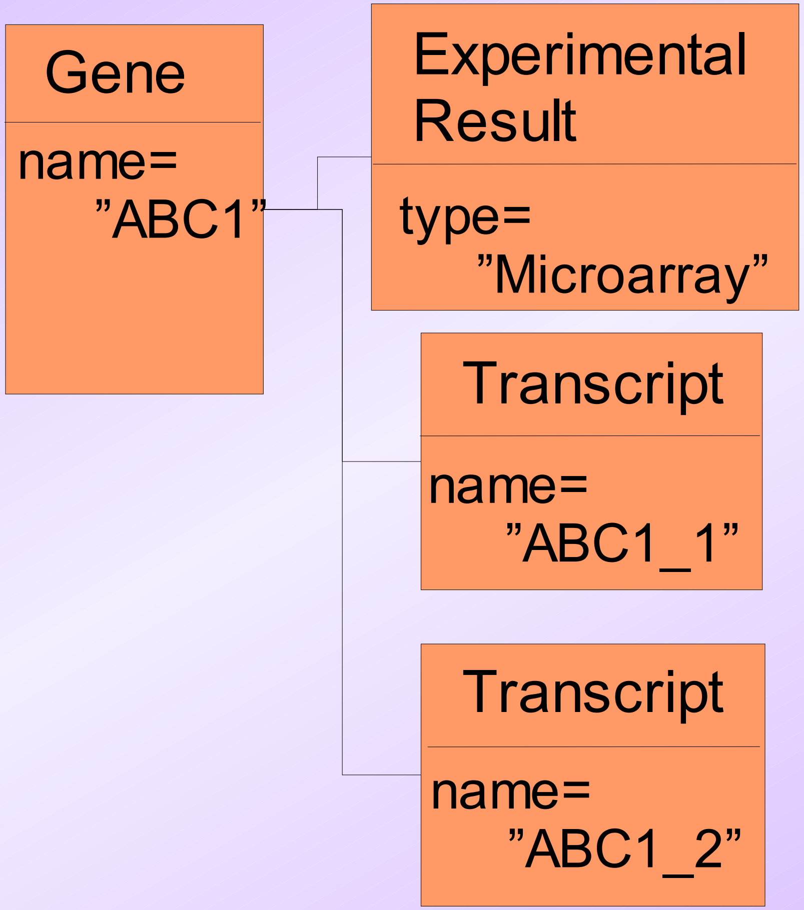


```
Primary keys:
  Gene: name
  Gene: flyBaseName
  Transcript: name,gene
```

New objects

Currently in database

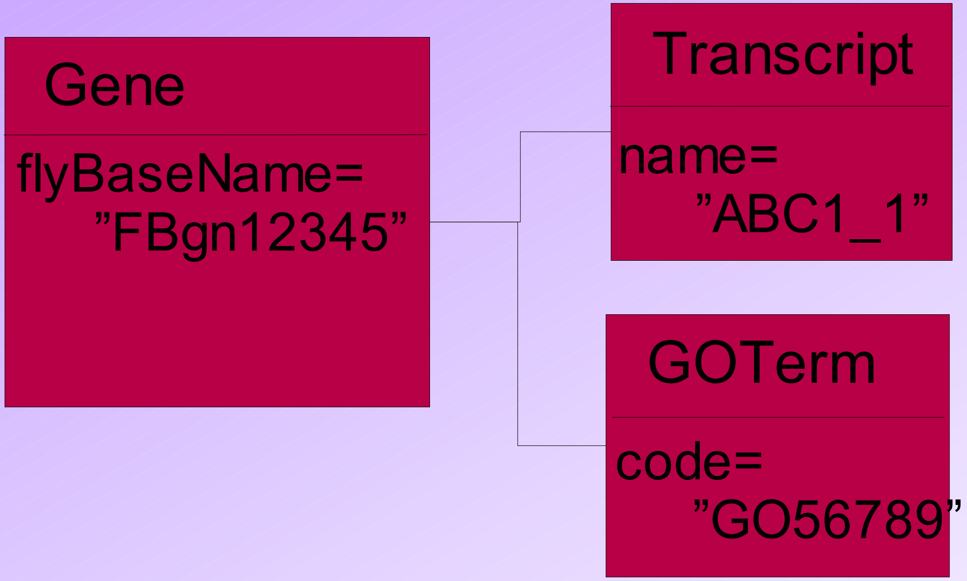


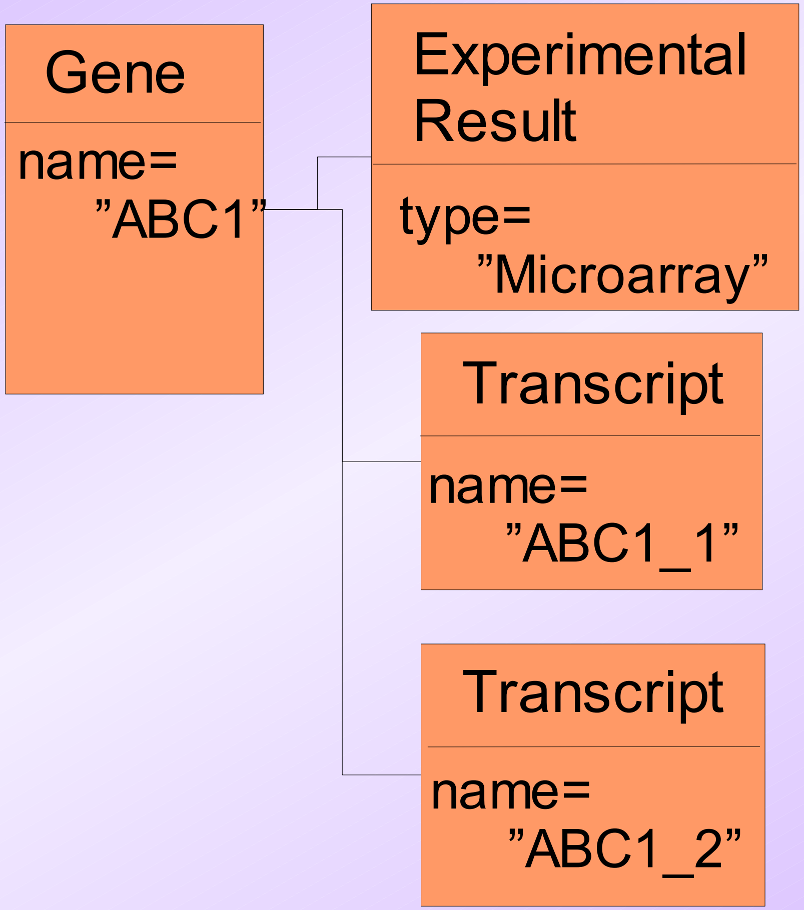


```
Primary keys:
  Gene: name
  Gene: flyBaseName
  Transcript: name,gene
```

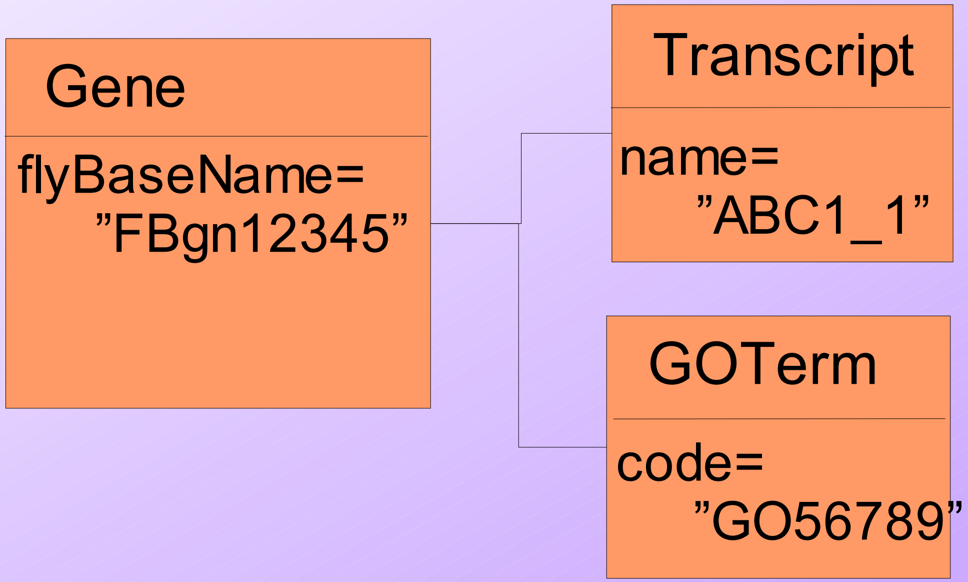
New objects

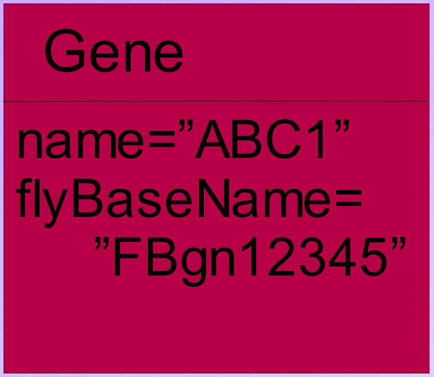
Currently in database

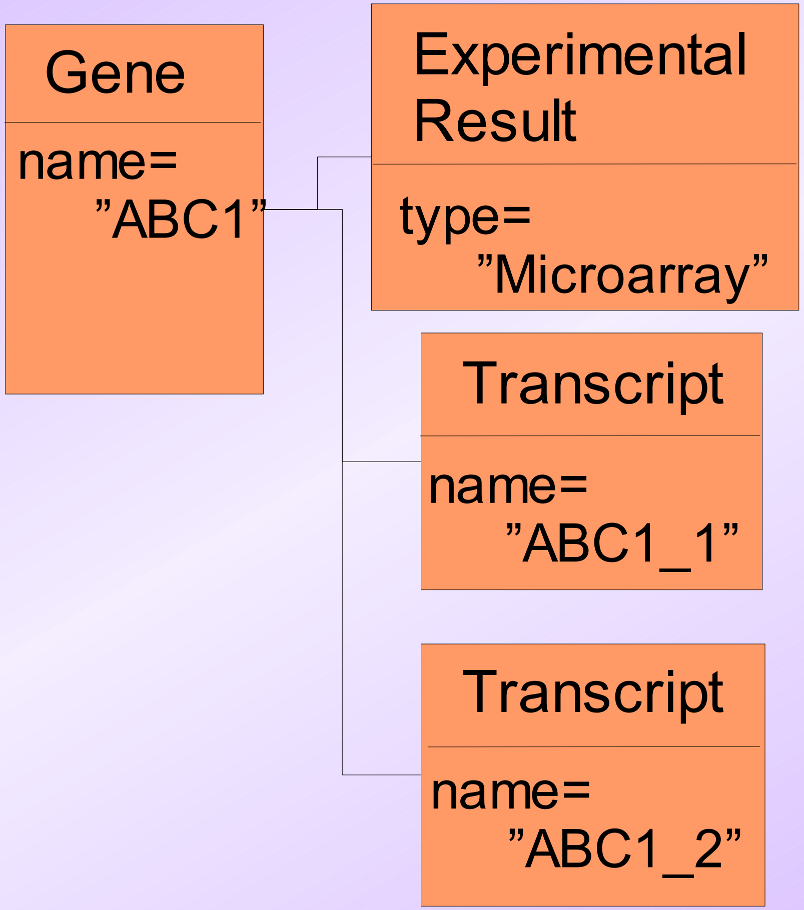




```
Primary keys:
  Gene: name
  Gene: flyBaseName
  Transcript: name,gene
```

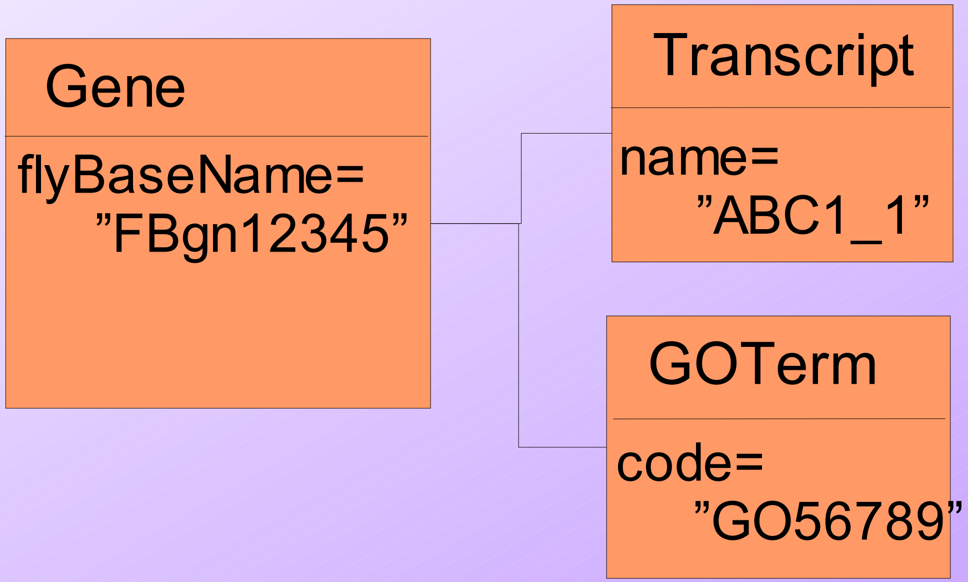


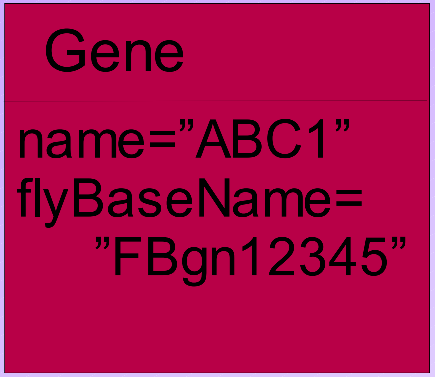


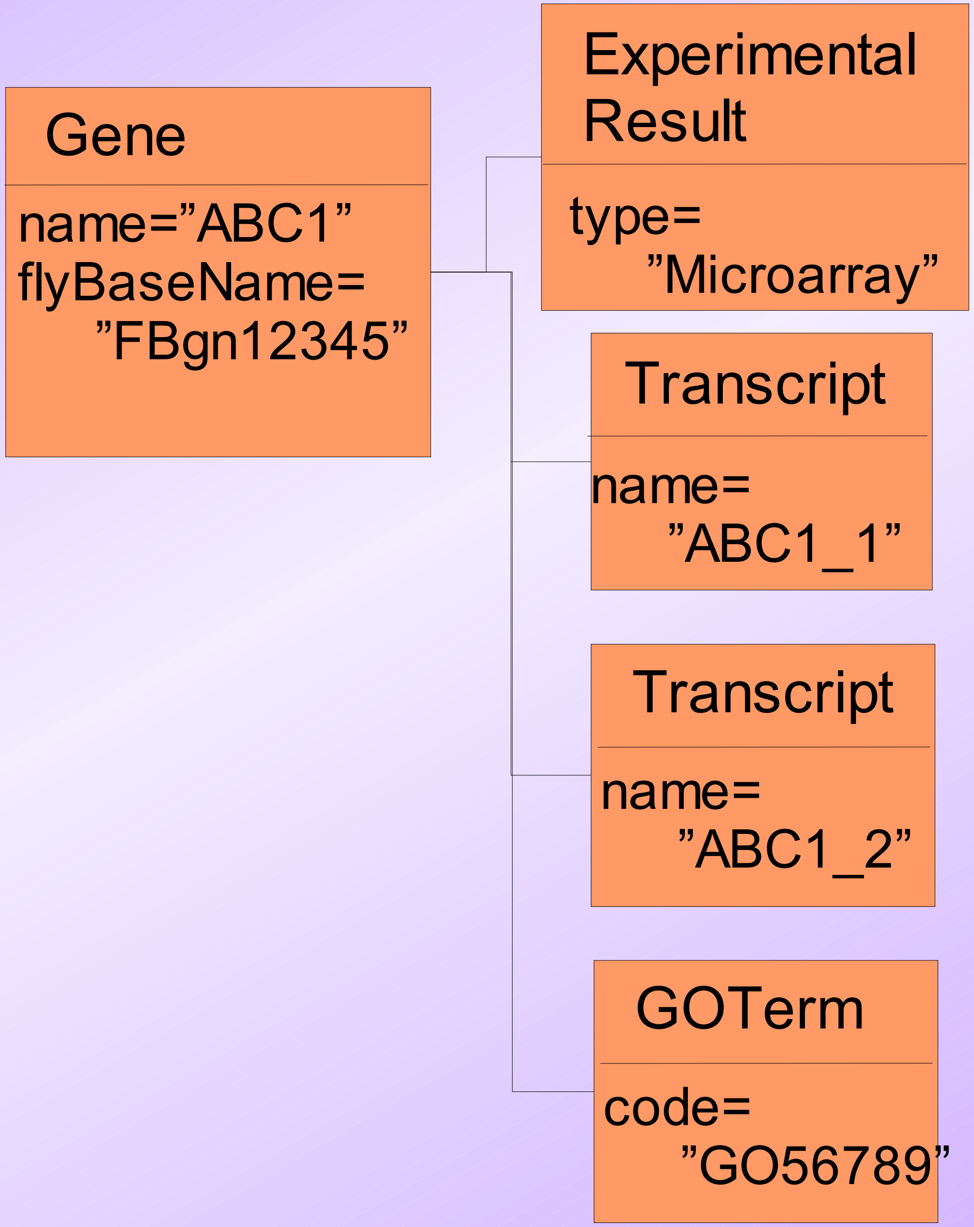


Primary keys:

```
Primary keys:
  Gene: name
  Gene: flyBaseName
  Transcript: name,gene
```







```
Primary keys:
  Gene: name
  Gene: flyBaseName
  Transcript: name,gene
```

New objects

Currently in database

Lessons learned

- Need to batch database reads and writes to avoid round trip time
  - without: 6 months to load
  - with: 6 hours to load
- Cache objects to avoid database lookups
- Use multi-thread model
- GB ethernet link between computational and database machines useful

Arbitrary queries – problems

- Badly formed queries may overload database server
- Difficult to optimise database for all queries
  - Which indexes to use?
  - Slow response to complex queries involving multi-table joins

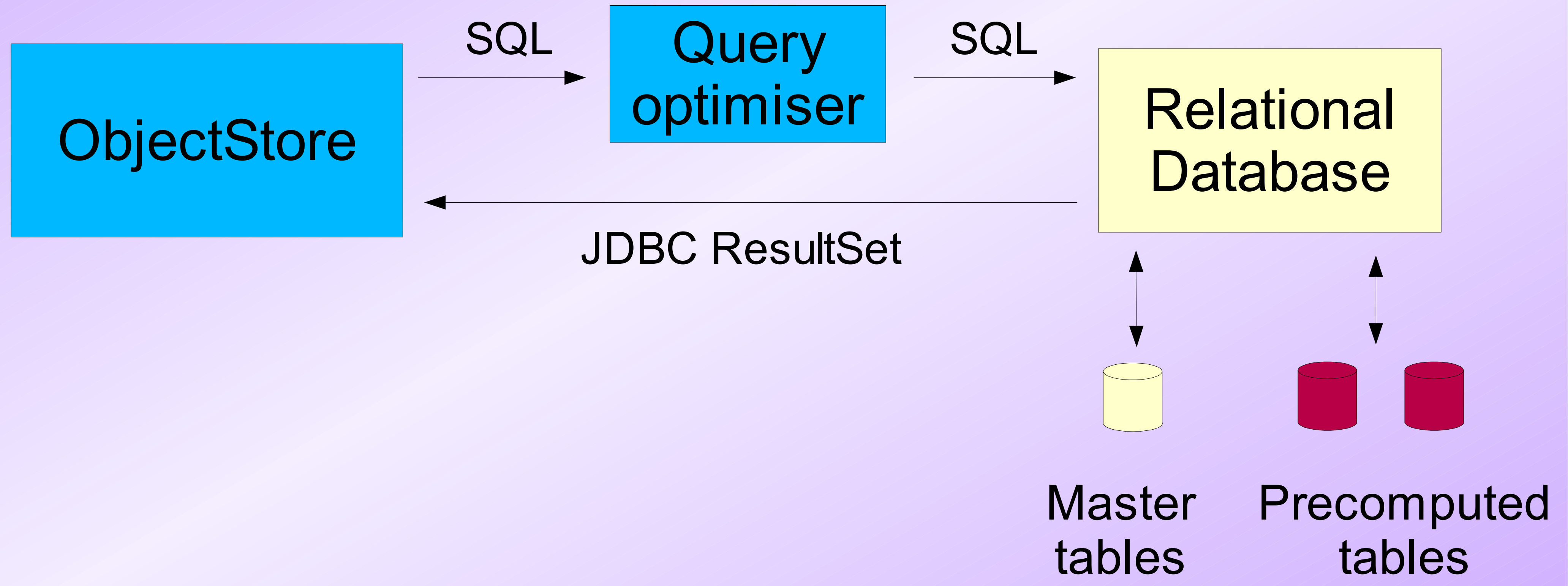
Arbitrary queries – solutions

- Close relationship with database server query planner
  - ask how long a query will take before attempting to run it (~3ms)
  - Disallow queries that will take longer than a certain threshold
- Store data massively redundantly in “precomputed tables” and rewrite incoming queries on-the-fly.

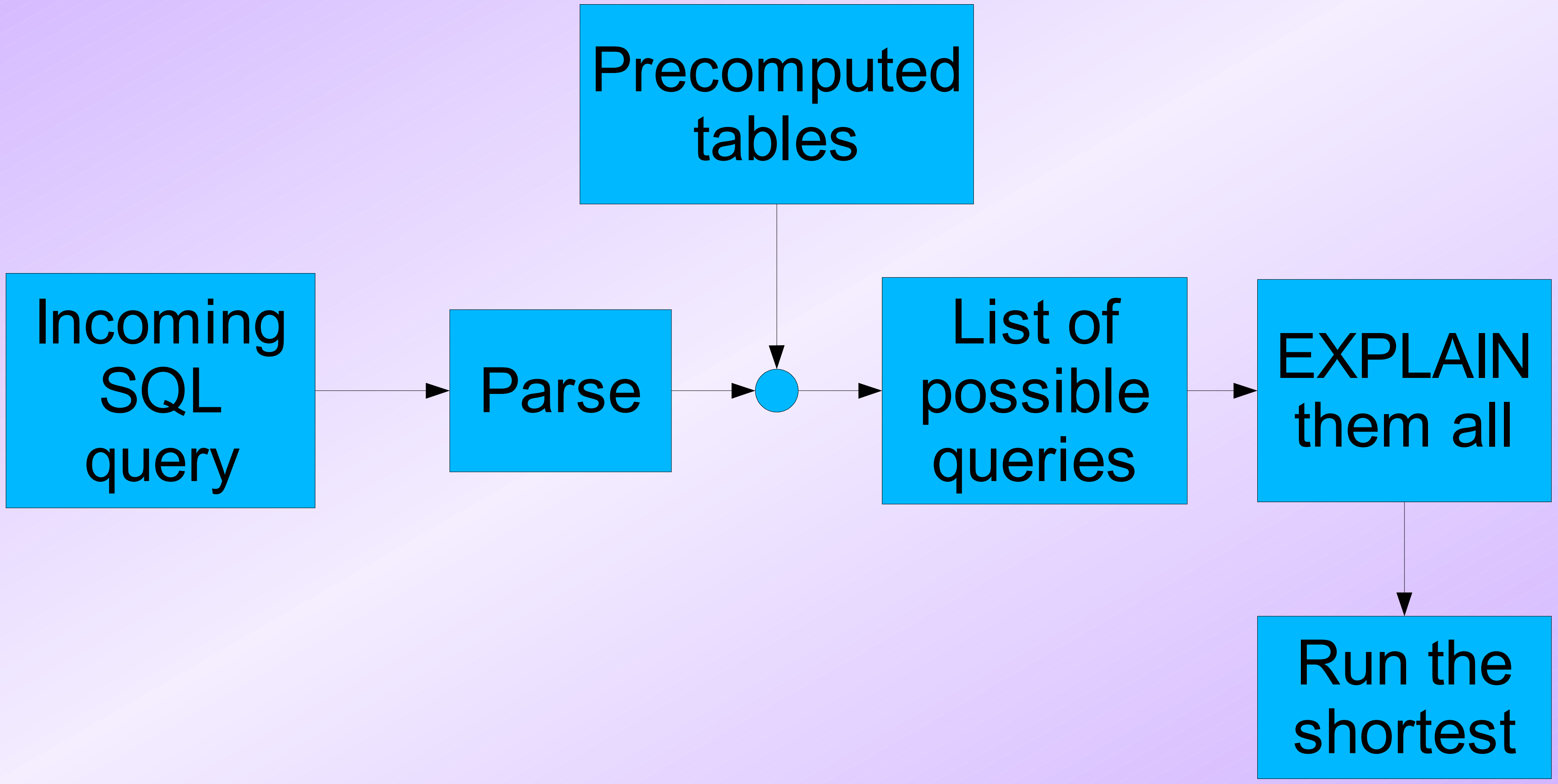
Query optimisation – aims

- Provide a generic way of speeding up complex queries to any read-only SQL database
- Minimal parsing/computational overhead
- Transparent to users/applications
  - no new schema/model to learn
- Make available as standalone module

Query optimisation architecture



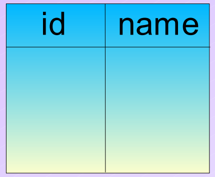
Query optimisation



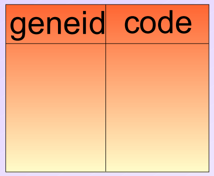
Query optimisation – example

Master tables

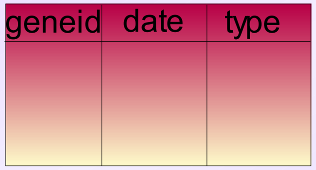
Genes



GOTerms

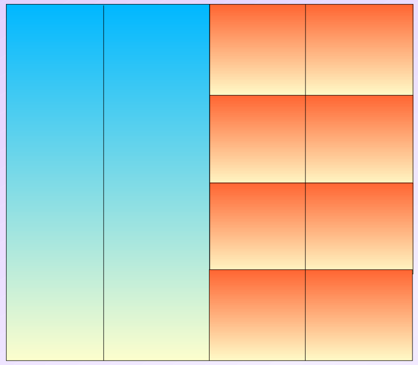


Experiments



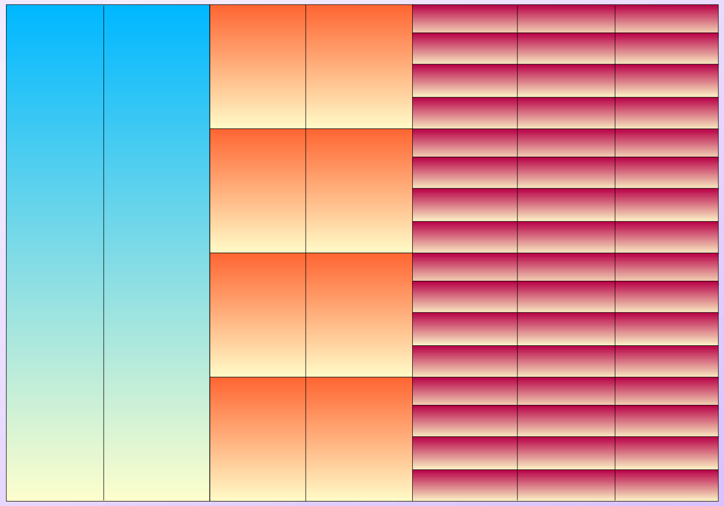
Precomputed tables

GenesGOTerms



```
CREATE TABLE genesgoterms AS
SELECT genes.id AS genes_id,
       genes.name AS genes_name,
       goterms.geneid AS goterms_geneid,
       goterms.code AS goterms_code
FROM genes, goterms
WHERE genes.id = goterms.geneid
```

GenesGOTermsExperiments



```
CREATE TABLE ....
SELECT ....
FROM genes, goterms, experiments
WHERE genes.id = goterms.geneid
AND genes.id = experiments.geneid
```

FlyMine

Query optimisation – example

“Show dates that gene expression experiments were performed on genes which have GO term GO:0000278 applied”

```
SELECT genes.name, experiments.date
FROM genes, goterms, experiments
WHERE genes.id = goterms.geneid
AND genes.id = experiments.geneid
AND goterms.code = “GO:0000278”
AND experiments.type = “Gene Expression”
```

$\Downarrow$ EXPLAIN

5 seconds

Query optimisation – example

```
SELECT genes_name AS name,
       experiments.date
FROM genesgoterms  experiments
WHERE genes_id = experiments.geneid
AND goterms_code = “GO:0000278”
AND experiments.type = “Gene Expression”
```

EXPLAIN

1 second

FlyMine

Query optimisation – example

```
SELECT genes_name AS name,
       experiments_date AS date
FROM genesgotermsexperiments
WHERE goterms_code = “GO:0000278”
AND experiments_type = “Gene Expression”
```

EXPLAIN

200ms

Query optimisation – example

- Without optimiser

  Time = 3ms + 5 seconds

  (EXPLAIN) (EXECUTE)

- With optimiser

  Time = 9ms + 100ms + 200ms

  (3 EXPLAINs) (PARSE) (EXECUTE)

Query optimisation – summary

- For complex queries, the SQL optimisation module can produce large speed increases
- Optimisation is transparent to the user or application generating the SQL.
- Choosing which precomputed tables to store is important – may need to analyse incoming queries.

Query optimiser – summary

- Optimiser trades off disk space for improved query performance.
- Schema independent.
- Can be used in conjunction with P6Spy to intercept JDBC calls from existing software to a database.
- Available from www.intermine.org !!

Summary

- Current status
  - generic SQL query optimiser
  - powerful object data warehouse
  - 2 query interfaces (OQL + Java)
  - web front end
  - web service
  - framework for data loading/integration
  - no model-specific code
  - FlyMine database alpha release

Summary

- Coming up
  - graphical query interface
  - tools for auto-generation of best set of precomputed tables
  - manual curation tools

Acknowledgements

The FlyMine team:

Andrew Varley	François Guillier
Richard Smith	Rachel Lyne
Matthew Wakeling	Kim Rutherford
Mark Woodbridge	Gos Micklem

More information and download at www.intermine.org

FlyMine is funded by the Wellcome Trust (grant no. 067205), awarded to M. Ashburner, G. Micklem, S. Russell, K. Lilley and K. Mizuguchi.