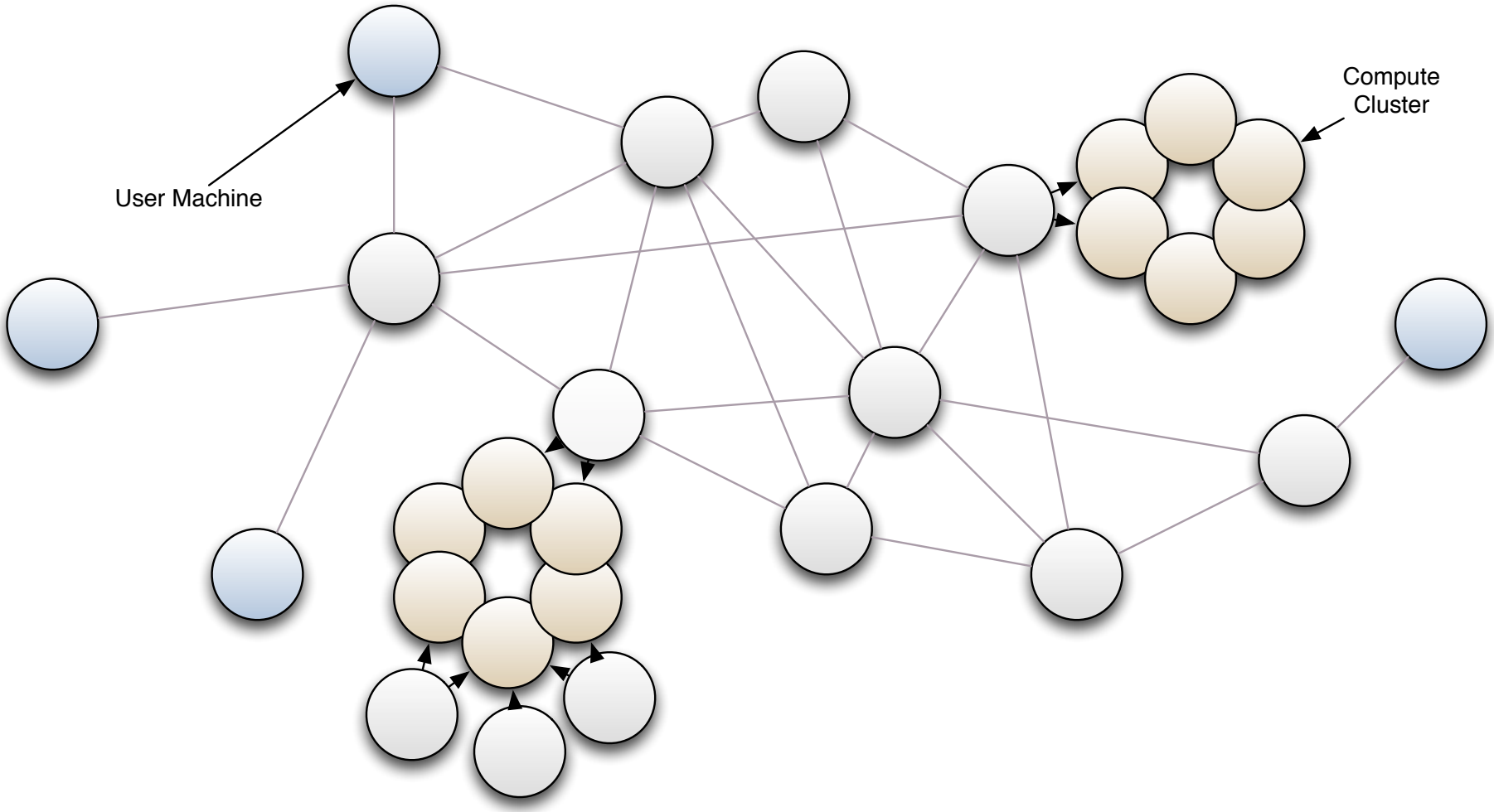
DISPEL Tutorial

Paul Martin

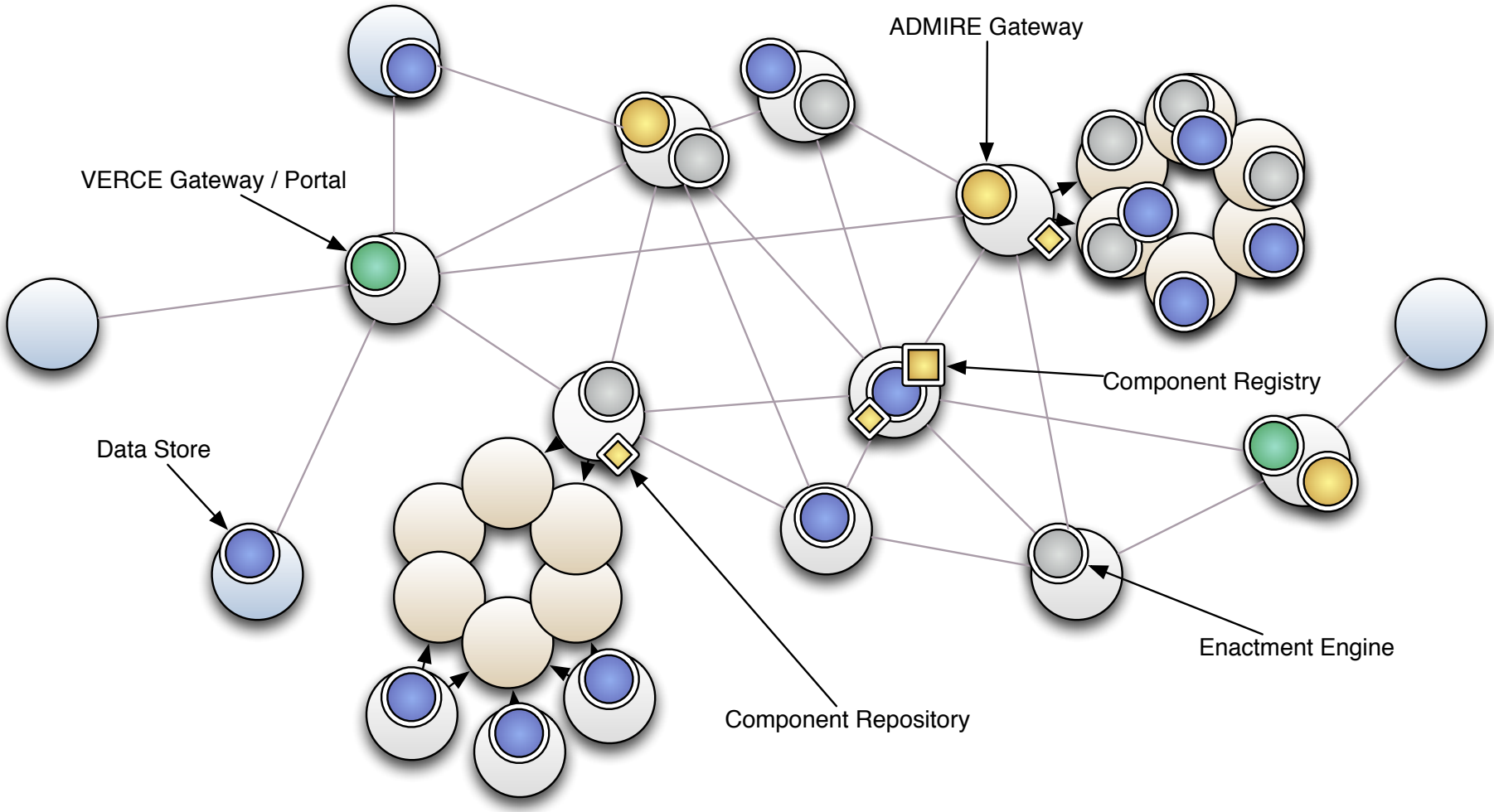
VERCE Training Programme

3rd September 2012

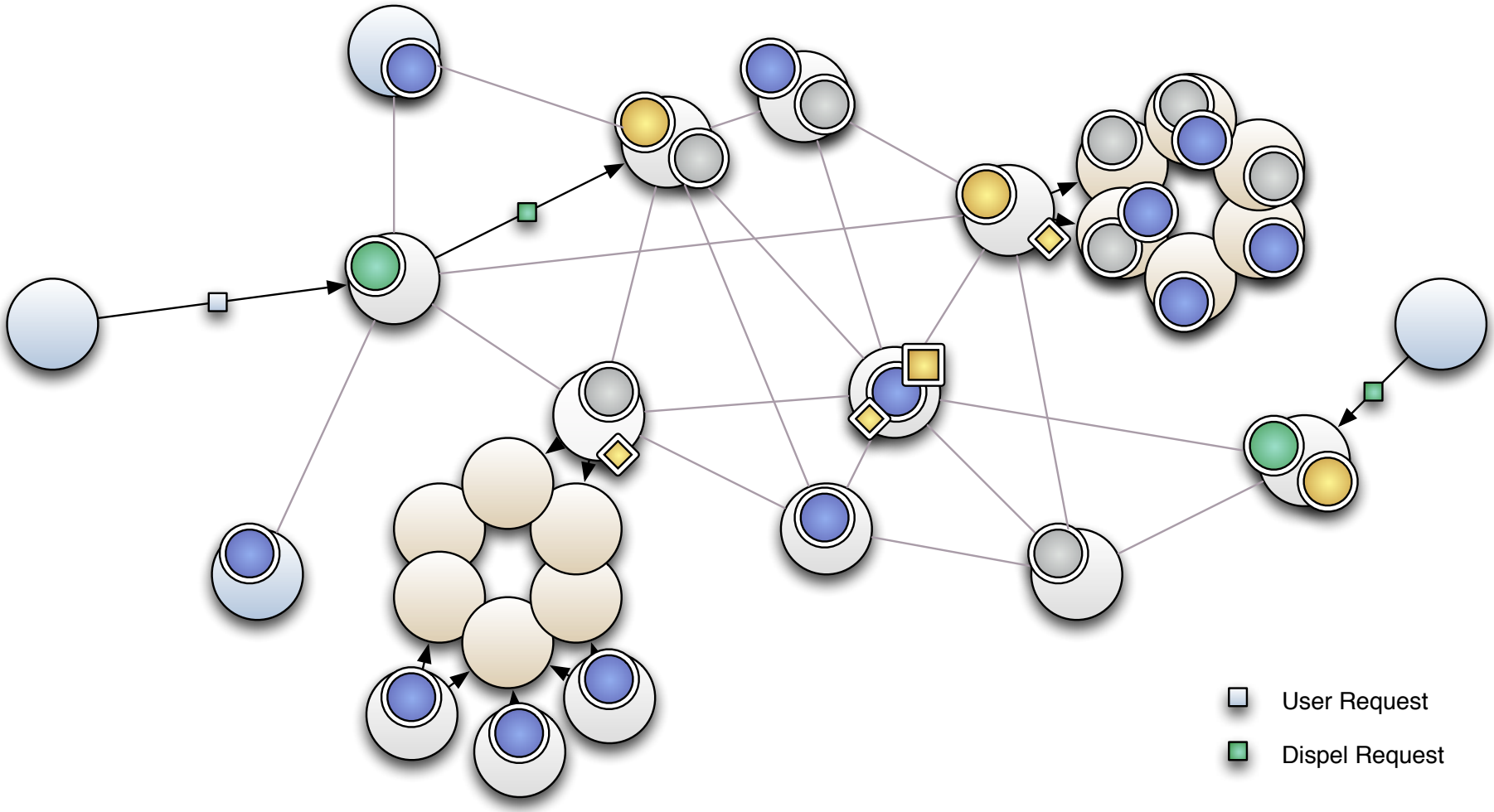
Workflow enactment



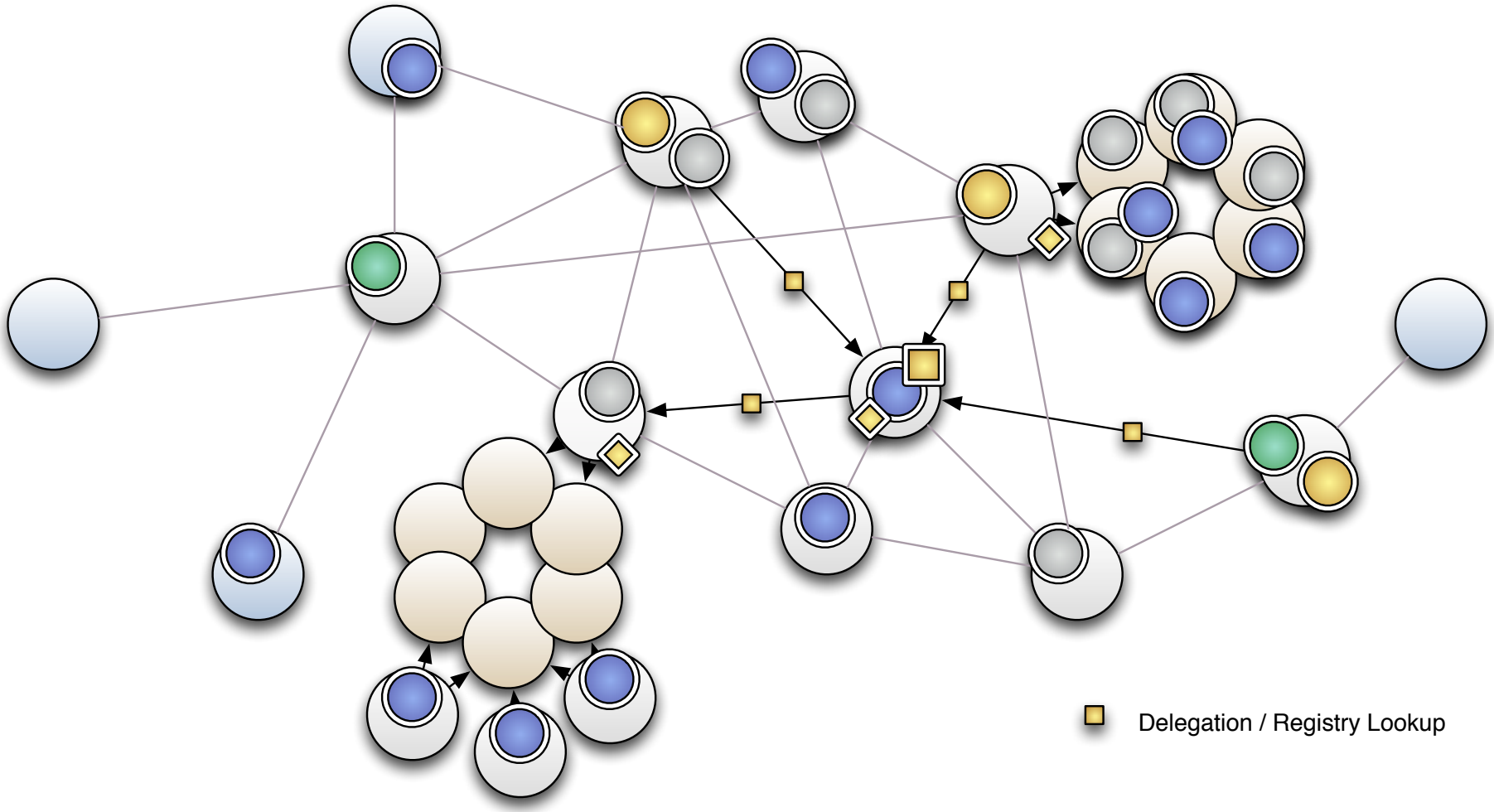
Workflow enactment



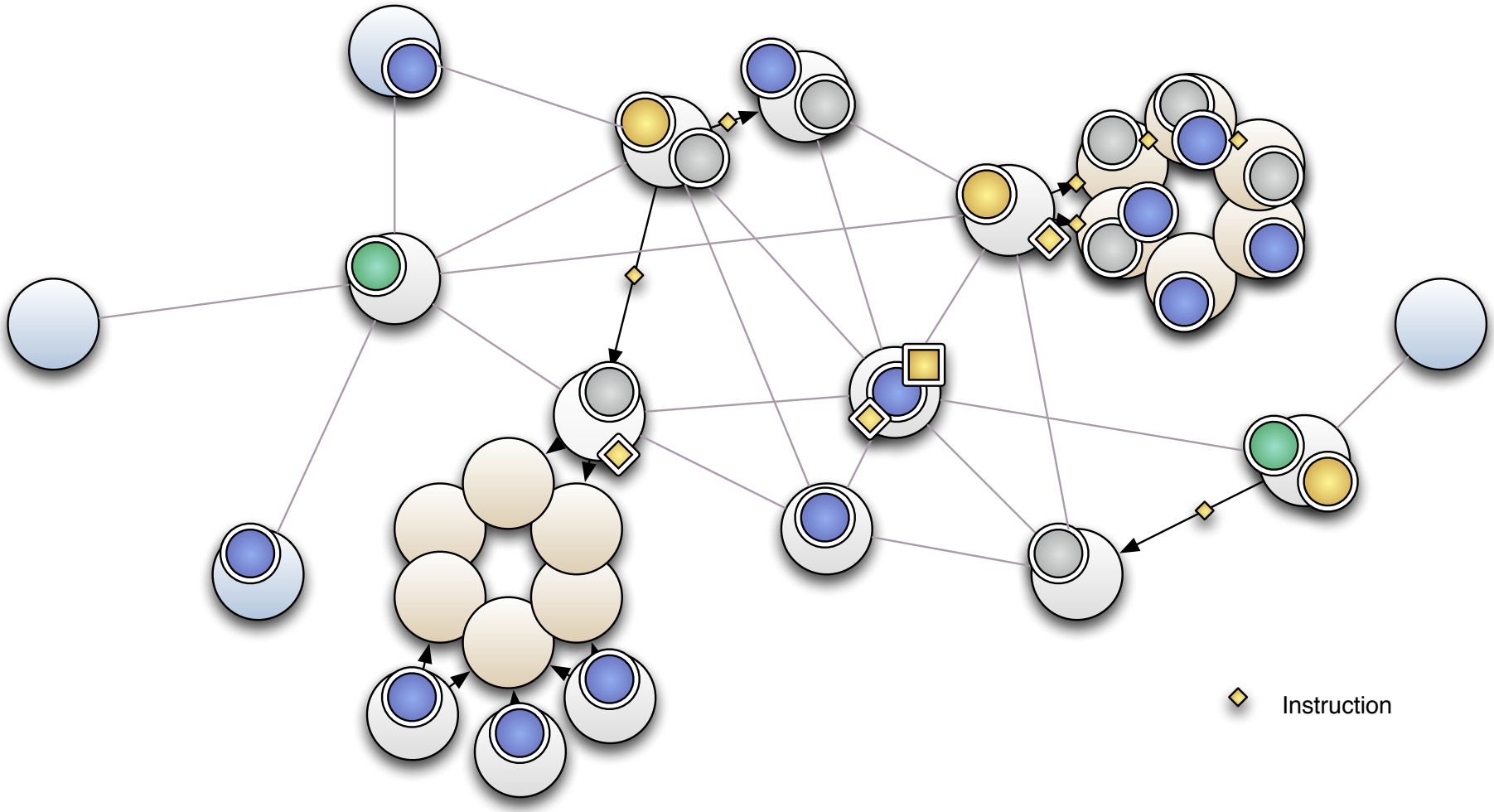
Workflow enactment



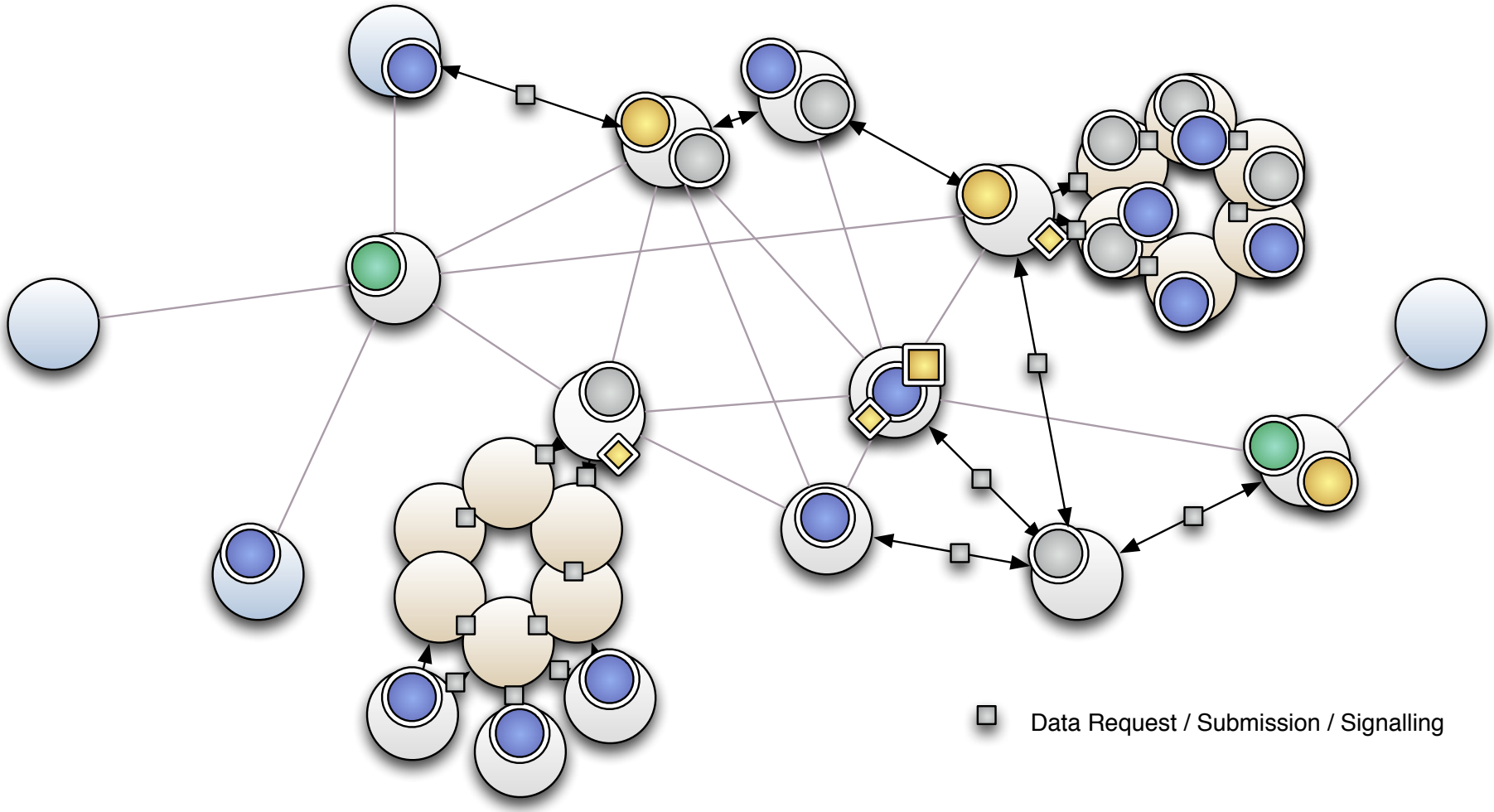
Workflow enactment



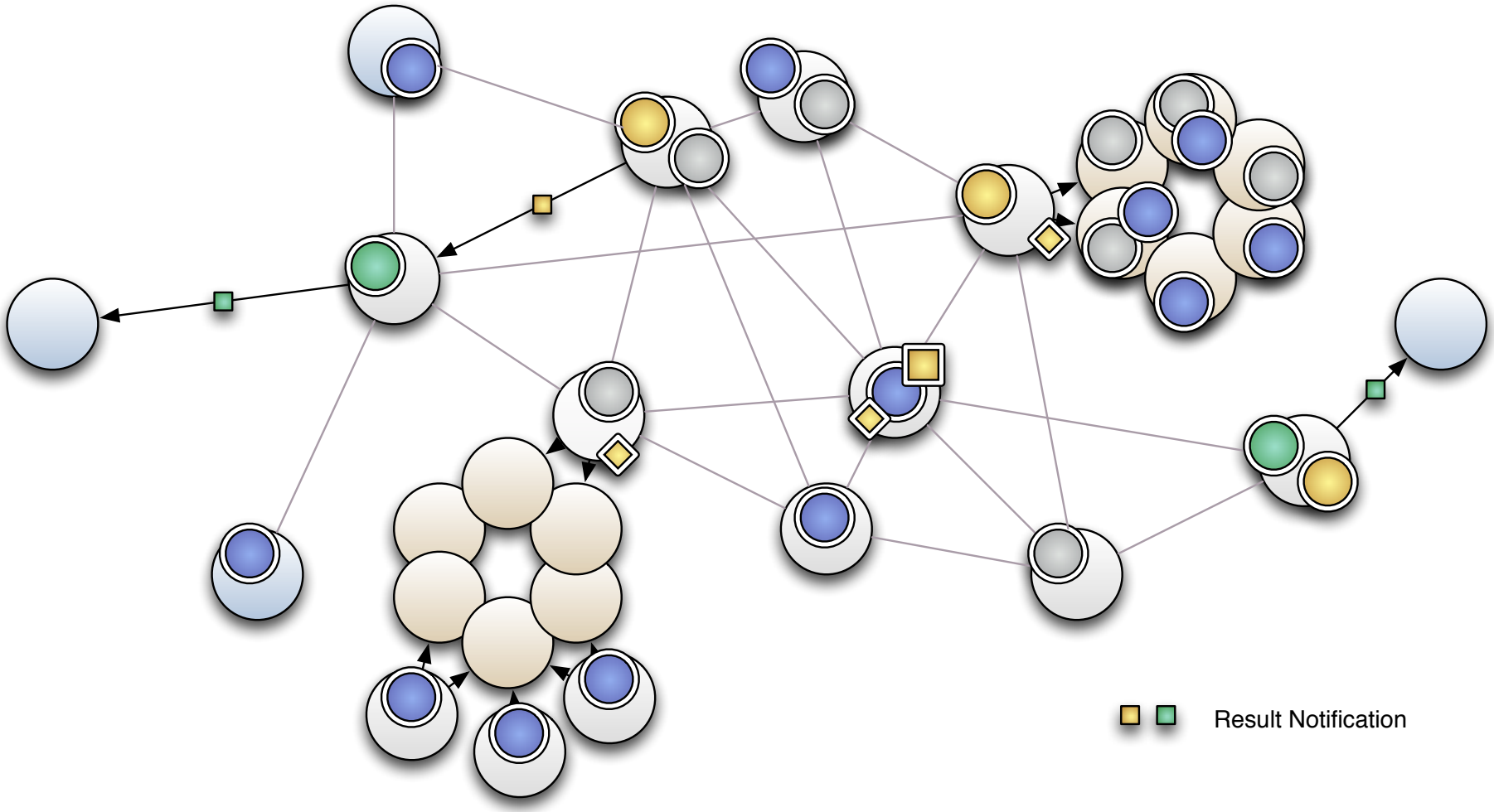
Workflow enactment



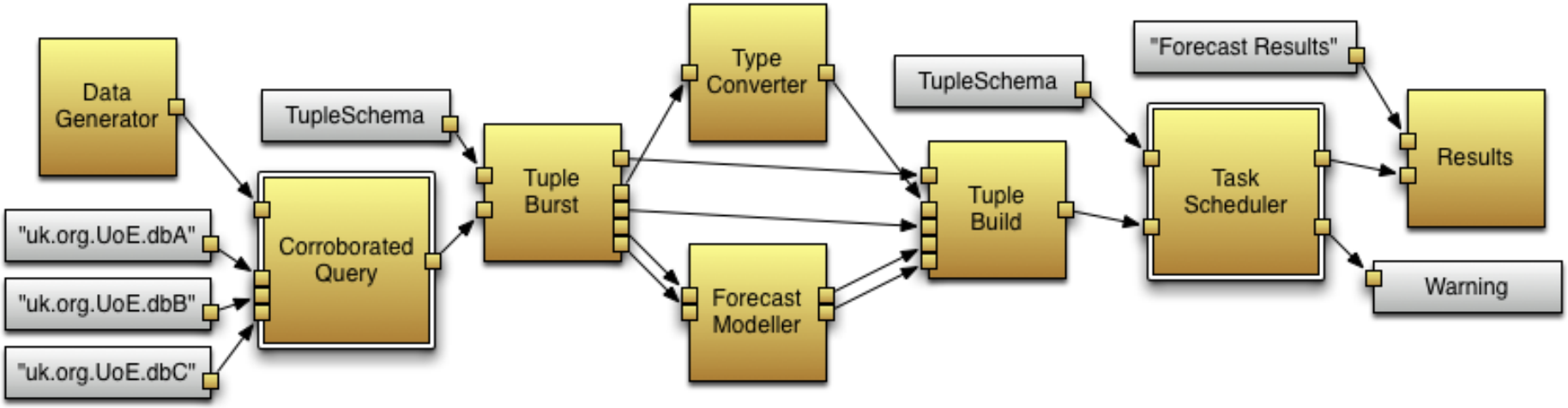
Workflow enactment



Workflow enactment



An example workflow



A simple script

```
1   // Import PEs from the local registry.
2   use dispel.db.SQLQuery;
3   use dispel.lang.Results;
4
5   // Create instances of PEs.
6   SQLQuery query   = new SQLQuery;
7   Results  results = new Results;
8
9   // Construct workflow and feed in data.
10  |-"SELECT * FROM littleblackbook WHERE id < 10"-| => query.expression;
11  |-"uk.org.UoE.dbA"-| => query.resource;
12  |-"10 entries from the little black book"-| => results.name;
13  query.data => results.input;
14
15  // Submit the entire workflow.
16  submit;
```

Demonstrating stream literals

```
1  // Import PEs from the local registry.
2  use dispel.db.SQLQuery;
3
4  // Create instances of PEs.
5  SQLQuery query   = new SQLQuery;
6  Results  results = new Results;
7
8  // Define a set of queries.
9  String[] queries = new String[2];
10 queries[0] = "SELECT name FROM littleblackbook WHERE id < 10";
11 queries[1] = "SELECT name FROM littleblackbook WHERE id >= 10 AND id < 20";
12 queries[2] = "SELECT address FROM littleblackbook WHERE name = 'David Hume'";
13
14 // Construct workflow and feed in data.
15 |-queries[0], queries[1], queries[2]-| => query.expression;
16 |-"uk.org.UoE.dbA", "uk.org.UoE.dbB", "uk.org.UoE.dbC"-| => query.resource;
17 |-"Results from the little black book"-| => results.name;
18 query.data => results.input;
19
20 // Submit the entire workflow.
21 submit;
```

Demonstrating stream literals

```
1  // Import PEs from the local registry.
2  use dispel.db.SQLQuery;
3
4  // Create instances of PEs.
5  SQLQuery query   = new SQLQuery;
6  Results  results = new Results;
7
8  // Define a set of queries.
9  String[] queries = new String[2];
10 queries[0] = "SELECT name FROM littleblackbook WHERE id < 10";
11 queries[1] = "SELECT name FROM littleblackbook WHERE id >= 10 AND id < 20";
12 queries[2] = "SELECT address FROM littleblackbook WHERE name = 'David Hume'";
13
14 // Construct workflow and feed in data.
15 |-queries[0], queries[1], queries[2]-| => query.expression;
16 |-repeat 3 of "uk.org.UoE.dbA"-| => query.resource;
17 |-"Results from the little black book"-| => results.name;
18 query.data => results.input;
19
20 // Submit the entire workflow.
21 submit;
```

A precocious script

```
1   // Import PEs from the local registry.
2   use dispel.db.SQLQuery;
3   use dispel.tutorial.PrecociousChild;
4
5   // Create instances of PEs.
6   PrecociousChild child   = new PrecociousChild;
7   SQLQuery        query   = new SQLQuery;
8   Results         results = new Results;
9
10  // Construct workflow and feed in data.
11  child.output => query.expression;
12  |-repeat enough of "uk.org.UoE.dbA"-| => query.resource;
13  |-"Adult responses"-| => results.name;
14  query.data => results.input;
15
16  // Submit the entire workflow.
17  submit;
```

PE Specification

```
1  // Within the Dispel database package.
2  package dispel.db {
3      // Link to database ontology.
4      namespace db "http://www.dispel-lang.org/resource/db";
5      // Type signature for the SQLQuery PE.
6      Type SQLQuery is PE( <Connection:String::"db:SQLQuery" terminator expression; Connection:String::"db:URI" locator resource> =>
7                           <Connection:[<rest>]::"db:TupleRowSet" data> );
8      // Register the PE.
9      register SQLQuery;
10 }
11
12 // Within the package of generically useful PEs.
13 package dispel.core {
14     // A generic interpolation PE with few restrictions.
15     Type Interpolate is PE( Stype Element is Any; <Connection[]:Element inputs> => <Connection:Element output> );
16     // This PE evenly interpolates in order of input connections.
17     Type OrderedInterpolate is Interpolate
18         with roundrobin inputs, @description = "Interpolates evenly by order of inputs.";
19
20     register Interpolate, OrderedInterpolate;
21 }
22
23 // Within the Dispel filter package.
24 package dispel.filter {
25     Type AbstractFilter is PE( Stype Element is Any;
26                                <Connection:Element inputs> => <Connection:Element filtered; Connection:Element unfiltered> )
27         with @description = "A filter splits input into filtered and unfiltered streams.";
28
29     Type HeadFilter is AbstractFilter
30         with filtered as head, unfiltered as tail, @description = "Diverts the head of a stream.";
31
32     Type ProgrammableIntegerFilter is PE( <Connection:Integer terminator input; Connection:String initiator expression;
33                                            Connection[]:Integer lockstep parameters> =>
34                                           <Connection:Integer filtered; Connection:Integer unfiltered> )
35         with @description = "Splits the input integer stream into 'filtered' and 'unfiltered' streams in" +
36                             "accordance with the expression and parameters given.";
37
38     register AbstractFilter, HeadFilter, ProgrammableIntegerFilter;
39 }
40
```

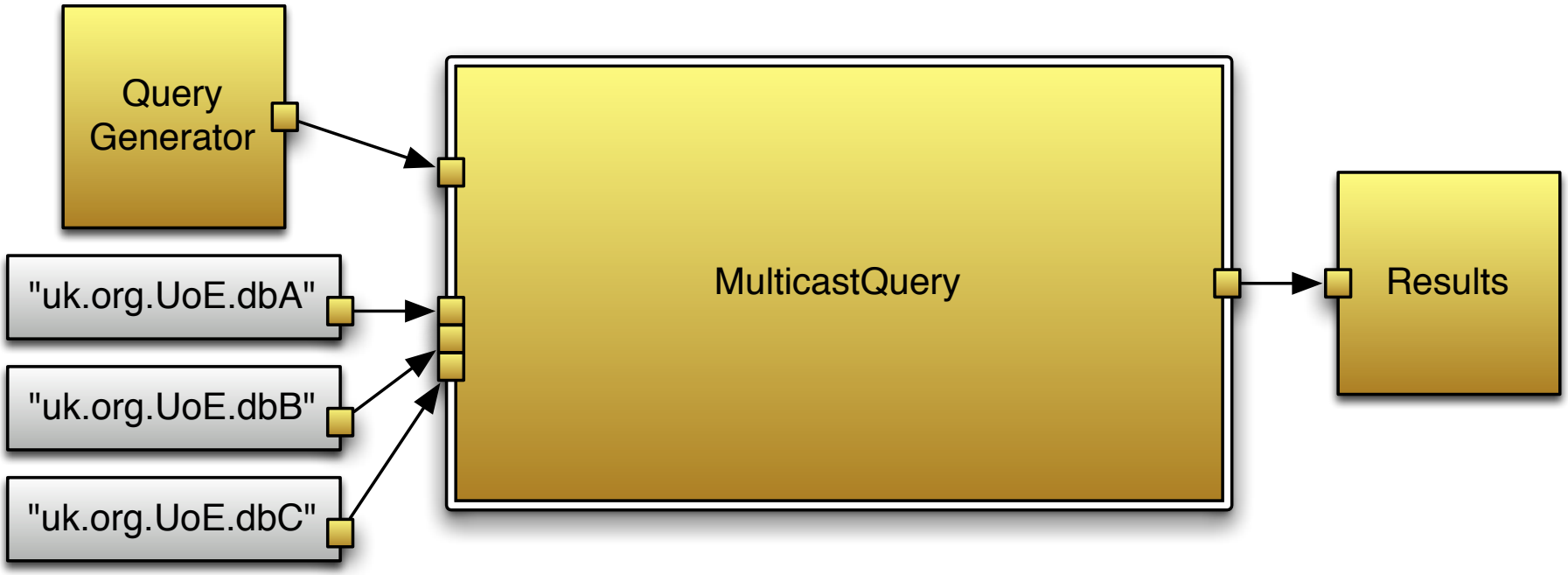
Constructing new PEs

```
1  package tutorial.example {
2      // Import existing PE from the registry.
3      use dispel.db.SQLQuery;
4
5      // Define new PE type.
6      Type SQLToTupleList is PE( <Connection expression> => <Connection data> );
7
8      // Define new PE constructor.
9      PE<SQLToTupleList> lockSQLDataSource(String dataSource) {
10         SQLQuery query = new SQLQuery;
11         |-repeat enough of dataSource-| => query.resource;
12         return PE( <Connection expression = query.expression> =>
13                    <Connection data = query.data> );
14     }
15
16     // Create new PEs.
17     PE<SQLToTupleList> TutorialQuery = lockSQLDataSource("uk.org.UoE.dbA");
18     PE<SQLToTupleList> MirrorQuery   = lockSQLDataSource("uk.org.UoE.dbB");
19
20     // Register new entities.
21     register TutorialQuery, MirrorQuery;
22 }
```

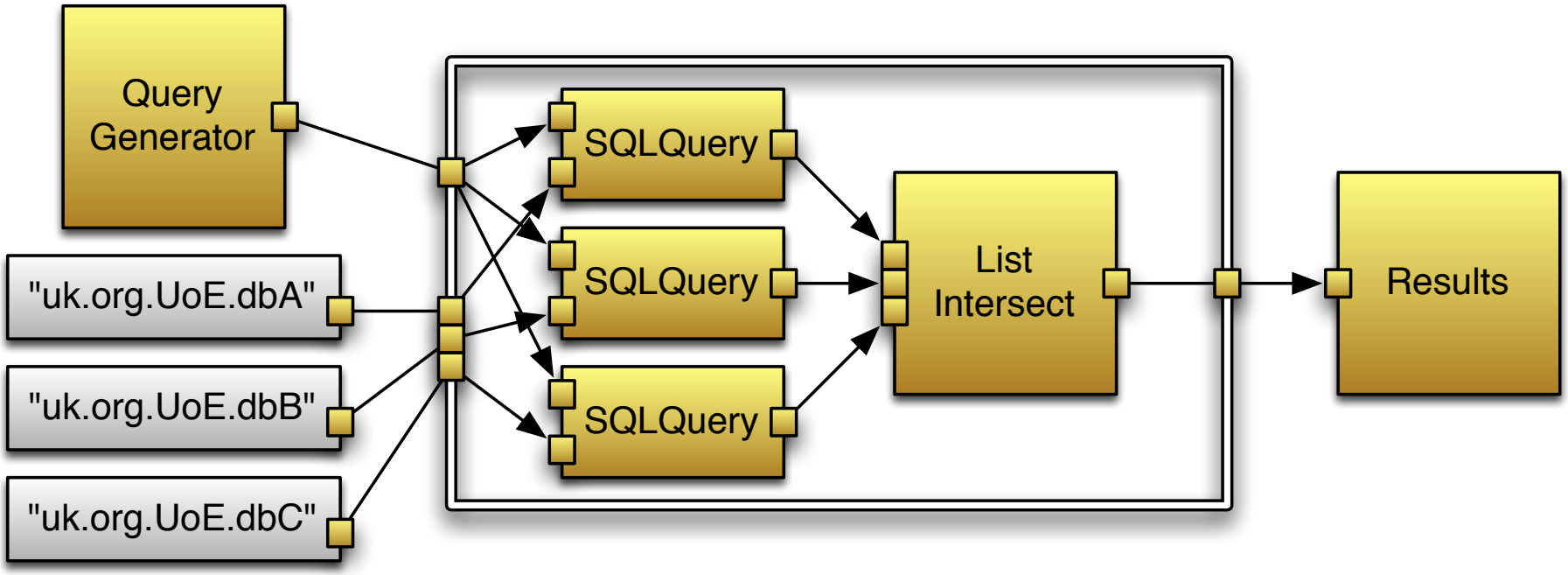
Using the new PE

```
1   // Import PEs from the local registry.
2   use tutorial.example.TutorialQuery;
3
4   // Create instances of PEs (no import necessary).
5   TutorialQuery query   = new TutorialQuery;
6   Results       results = new Results;
7
8   // Connect PEI together to create workflow (no data source needed).
9   |-"SELECT * FROM littleblackbook WHERE id <= 10"-| => query.expression;
10  |-"10 entries from the little black book"-| => results.name;
11  query.data => results.input;
12
13  // Submit workflow.
14  submit results;
```

Multicasting with composite PEs



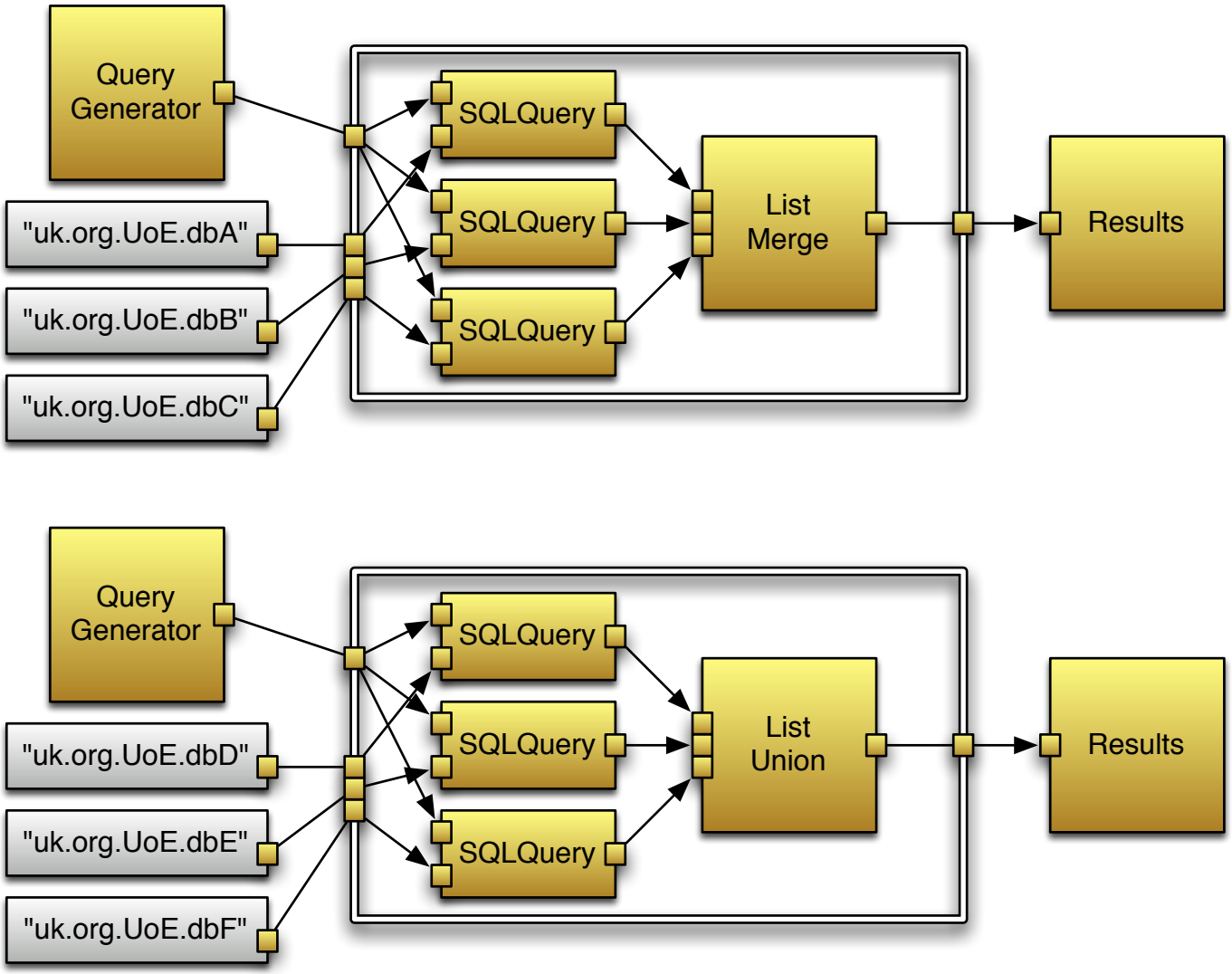
Multicasting with composite PEs



A PE constructor featuring iteration

```
1  package tutorial.example {
2      use dispel.db.SQLQuery;
3      use dispel.list.ListIntersect;
4
5      // A PE type which queries multiple sources.
6      Type MulticastQuery is PE( <Connection expression; Connection[] sources> => <Connection data> );
7
8      // Use parallel SQLQuery instances to corroborate results.
9      PE<MulticastQuery> makeCorroboratedSQLQuery(Integer size) {
10         // Define aliases for workflow inputs in advance.
11         Connection   expr;
12         Connection[] srcs = new Connection[size];
13         // Create instances of internal PEs.
14         SQLQuery[]     queries   = new SQLQuery[size];
15         ListIntersect intersect = new ListIntersect with inputs.length = size;
16
17         // Connect SQLQuery instances in parallel.
18         for (Integer i = 0; i < size; i++) {
19             queries[i] = new SQLQuery;
20             expr => queries[i].expression;
21             srcs[i] => queries[i].resource;
22             queries[i].data => intersect.inputs[i];
23         }
24
25         // Return intersection of query results.
26         return PE( <Connection expression = expr; Connection[] sources = srcs> =>
27                    <Connection data = intersect.output> );
28     }
29
30     register MulticastQuery, makeCorroboratedSQLQuery;
31 }
```

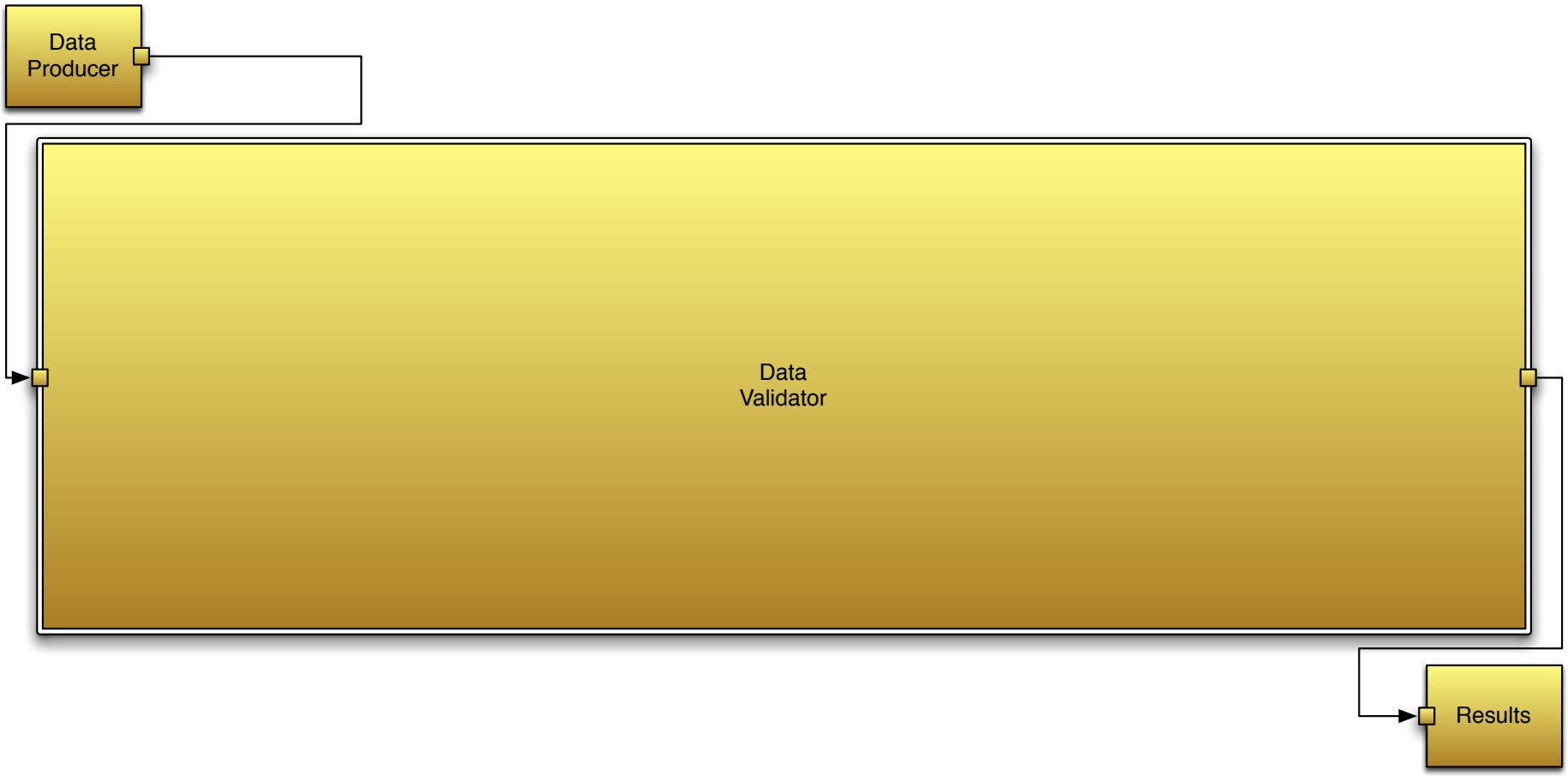
Multicasting with composite PEs



A PE constructor featuring selection

```
1  package tutorial.example {
2      use dispel.db.SQLQuery;
3      use dispel.list.Combiner;
4      use dispel.list.ListUnion;
5      use dispel.list.ListMerge;
6      // Import abstract type defined earlier.
7      use tutorial.example.MulticastQuery;
8
9      // Use parallel SQLQuery instances to collect results.
10     PE<MulticastQuery> makeMassSQLQuery(Integer size, Boolean unique) {
11         // Prepare components for connection.
12         Connection   expr;
13         Connection[] srcs    = new Connection[size];
14         SQLQuery[]   queries = new SQLQuery[size];
15         Combiner     merge;
16         // Select combiner based on uniqueness condition.
17         if (unique) merge = new ListUnion with inputs.length = size;
18         else        merge = new ListMerge with inputs.length = size;
19
20         // Connect SQLQuery instances in parallel.
21         for (Integer i = 0; i < size; i++) {
22             queries[i] = new SQLQuery;
23             expr => queries[i].expression;
24             srcs[i] => queries[i].resource;
25             queries[i].data => merge.inputs[i];
26         }
27
28         // Return merger of query results.
29         return PE( <Connection expression = expr; Connection[] sources = srcs> =>
30                    <Connection data = merge.output> );
31     }
32
33     register makeMassSQLQuery;
34 }
35
```

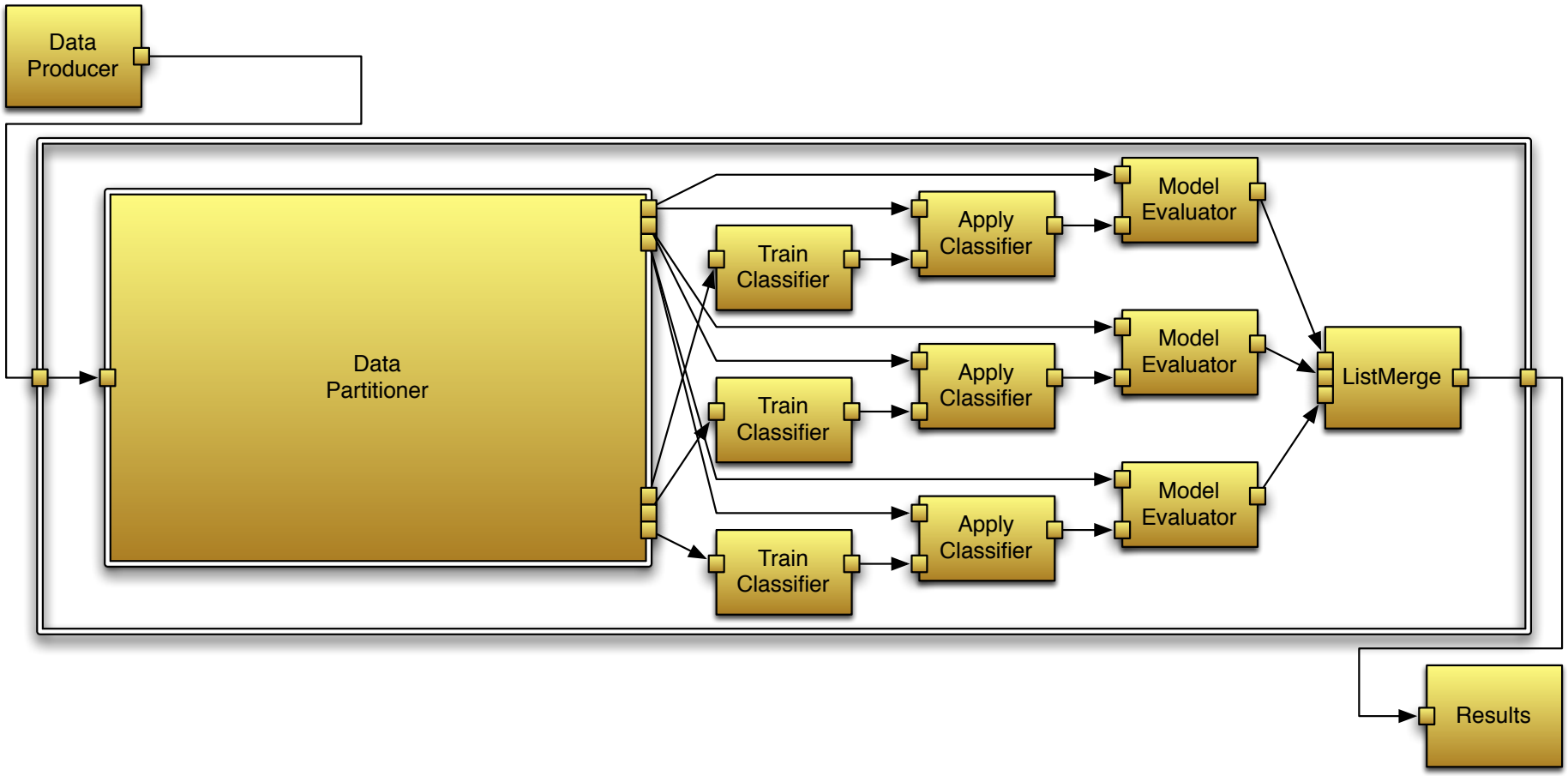
k-fold cross validation



k-fold cross validation

```
1  package dispel.datamine {
2      use dispel.core.DataPartition;
3      use dispel.list.ListMerge;
4
5      // Produces a k-fold cross validation workflow pattern.
6      PE<Validator> makeCrossValidator(Integer k, PE<TrainClassifier> Train,
7                                       PE<ApplyClassifier> Classify, PE<ModelEvaluate> Evaluate) {
8          Connection input;
9          // Data must be partitioned and re-combined for each fold.
10         PE<DataPartitioner> FoldData = makeDataFold(k);
11         FoldData  fold  = new FoldData;
12         ListMerge union = new ListMerge with inputs.length = k;
13
14         // For each fold, train a classifier then evaluate it.
15         input => fold.data;
16         for (Integer i = 0; i < k; i++) {
17             Train    train    = new Train;
18             Classify classify = new Classify;
19             Evaluate evaluate = new Evaluate;
20
21             fold.training[i] => train.data;
22             train.classifier => classify.classifier;
23             fold.test[i] => classify.data;
24             classify.result => evaluate.predicted;
25             fold.test[i] => evaluate.expected;
26             evaluate.score => union.inputs[i];
27         }
28
29         // Return cross validation pattern.
30         return PE( <Connection data = input> => <Connection results = union.output> );
31     }
32
33     register makeCrossValidator;
34 }
```

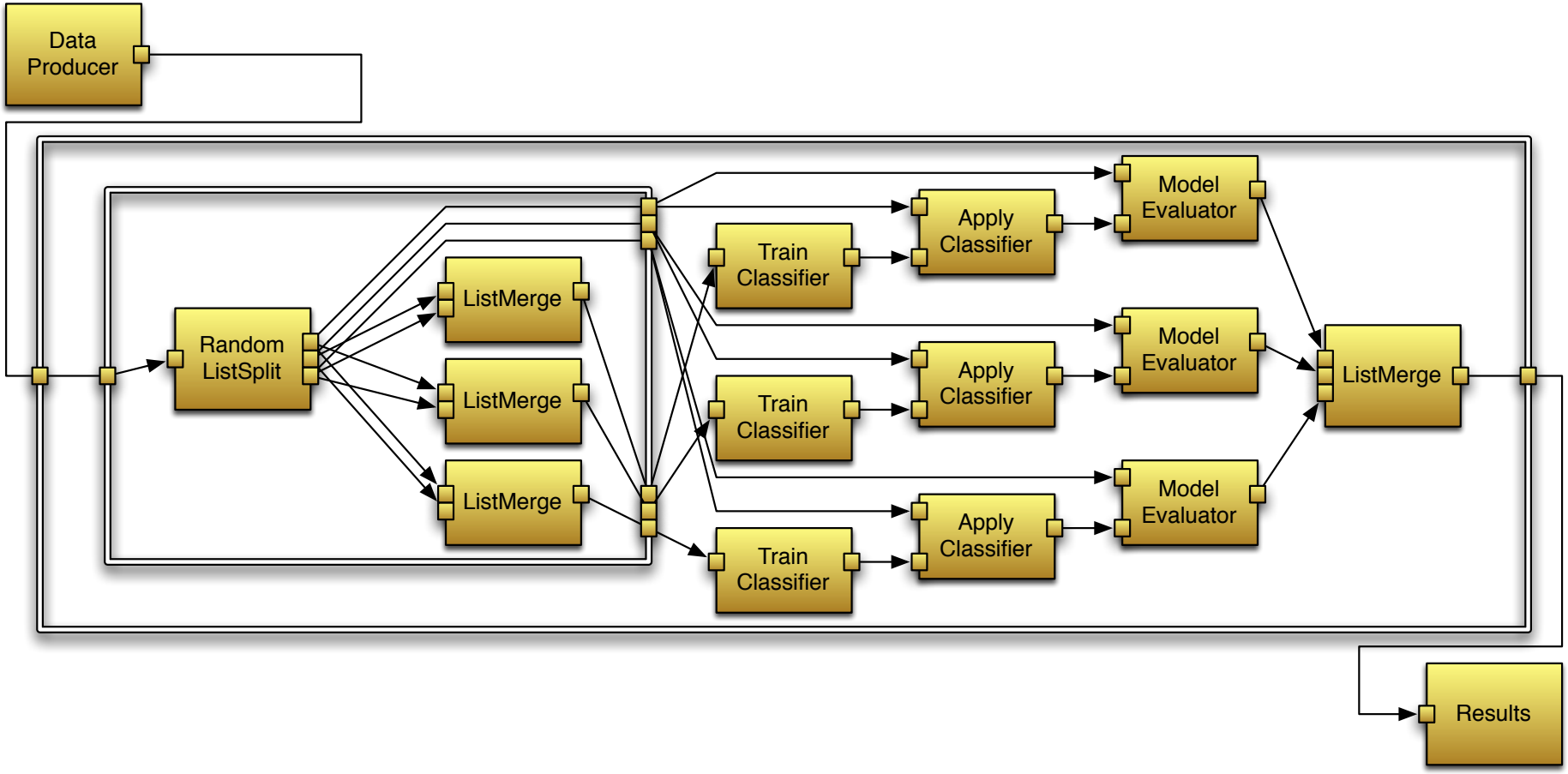
k-fold cross validation



k-fold cross validation

```
1  package dispel.datamine {
2      use dispel.core.RandomListSplit;
3      use dispel.list.ListMerge;
4
5      // Produces a PE capable of splitting data for k-fold cross validation.
6      PE<DataPartitioner> makeDataFold(Integer k) {
7          Connection   input;
8          Connection[] trainingData = new Connection[k];
9          Connection[] testData     = new Connection[k];
10         // Create instance of PEs for randomly splitting and recombining data.
11         RandomListSplit sample = new RandomListSplit with results.length = k;
12         ListMerge[]     union  = new ListMerge[k];
13
14         // After partitioning data, form training and test sets.
15         input => sample.input;
16         for (Integer i = 0; i < k; i++) {
17             union[i] = new TupleUnionAll with inputs.length = k - 1;
18             for (Integer j = 0; j < i; j++) sample.outputs[j] => union[i].inputs[j];
19             sample.outputs[i] => testData[i];
20             for (Integer j = i + 1; j < k; j++) sample.outputs[j] => union[i].inputs[j - 1];
21             union[i].output => trainingData[i];
22         }
23
24         // Return data folding pattern.
25         return PE( <Connection data = input> =>
26                    <Connection[] training = trainingData; Connection[] test = testData> );
27     }
28
29     register makeDataFold;
30 }
```

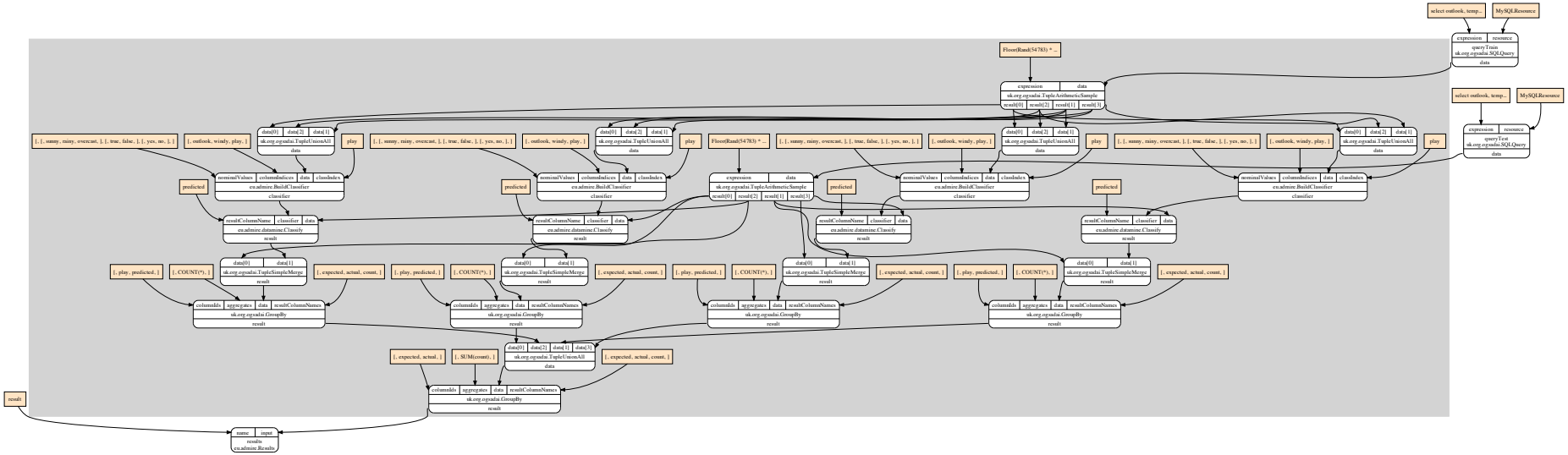
k-fold cross validation



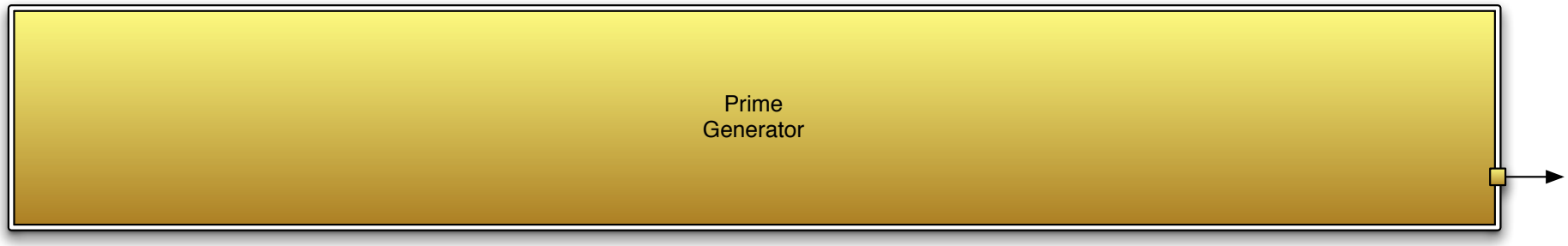
k-fold cross validation

```
1  package dispel.datamine {
2      // Import test PEs.
3      use dispel.tutorial.DataProducer;
4      use dispel.tutorial.TrainingAlgorithmA;
5      use dispel.tutorial.BasicClassifier;
6      use dispel.tutorial.MeanEvaluator;
7
8      // Create a cross validator PE.
9      PE<Validator> CrossValidator
10         = makeCrossValidator(12, TrainingAlgorithmA, BasicClassifier, MeanEvaluator);
11     // Make instances of PEs for workflows.
12     DataProducer   producer  = new DataProducer;
13     CrossValidator validator = new CrossValidator;
14     Results        results   = new Results;
15
16     // Connect workflow.
17     |- "uk.org.UoE.data.corpus11" -| => producer.source;
18     producer.data => validator.data;
19     validator.results => results.input;
20     |- "Classifier Scores" -| => results.name;
21
22     submit;
23 }
```

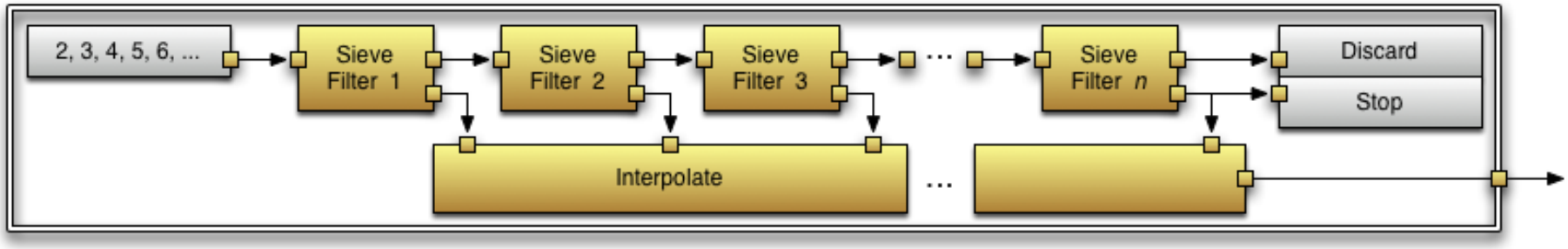
k-fold cross validation



Sieve of Eratosthenes



Sieve of Eratosthenes



Sieve of Eratosthenes

```
1  package tutorial.example {
2      use dispel.filter.AbstractFilter;
3      use dispel.filter.HeadFilter;
4      use dispel.filter.ProgrammableIntegerFilter;
5
6      // Define sieve element constructor.
7      PE<AbstractFilter> makeSieveFilter() {
8          // Create reference to input connection.
9          Connection:Integer input;
10         // Instantiate internal components.
11         HeadFilter split = new HeadFilter;
12         ProgrammableIntegerFilter divide = new ProgrammableIntegerFilter with parameters.length = 1;
13
14         // Construct internal workflow.
15         |-"x if (x % $0) == 0"-| => divide.expression;
16         input => split.input;
17         split.head => divide.parameter[0];
18         split.tail => divide.input;
19         divide.unfiltered => discard;
20
21         // Output first integer received and all indivisible integers.
22         return PE( <Connection input = input> =>
23                    <Connection filtered = split.head; Connection unfiltered = divide.filtered> );
24     }
25
26     // Create the sieve element PE.
27     PE<AbstractFilter> SieveFilter = makeSieveFilter();
28
29     register SieveFilter;
30 }
31
```

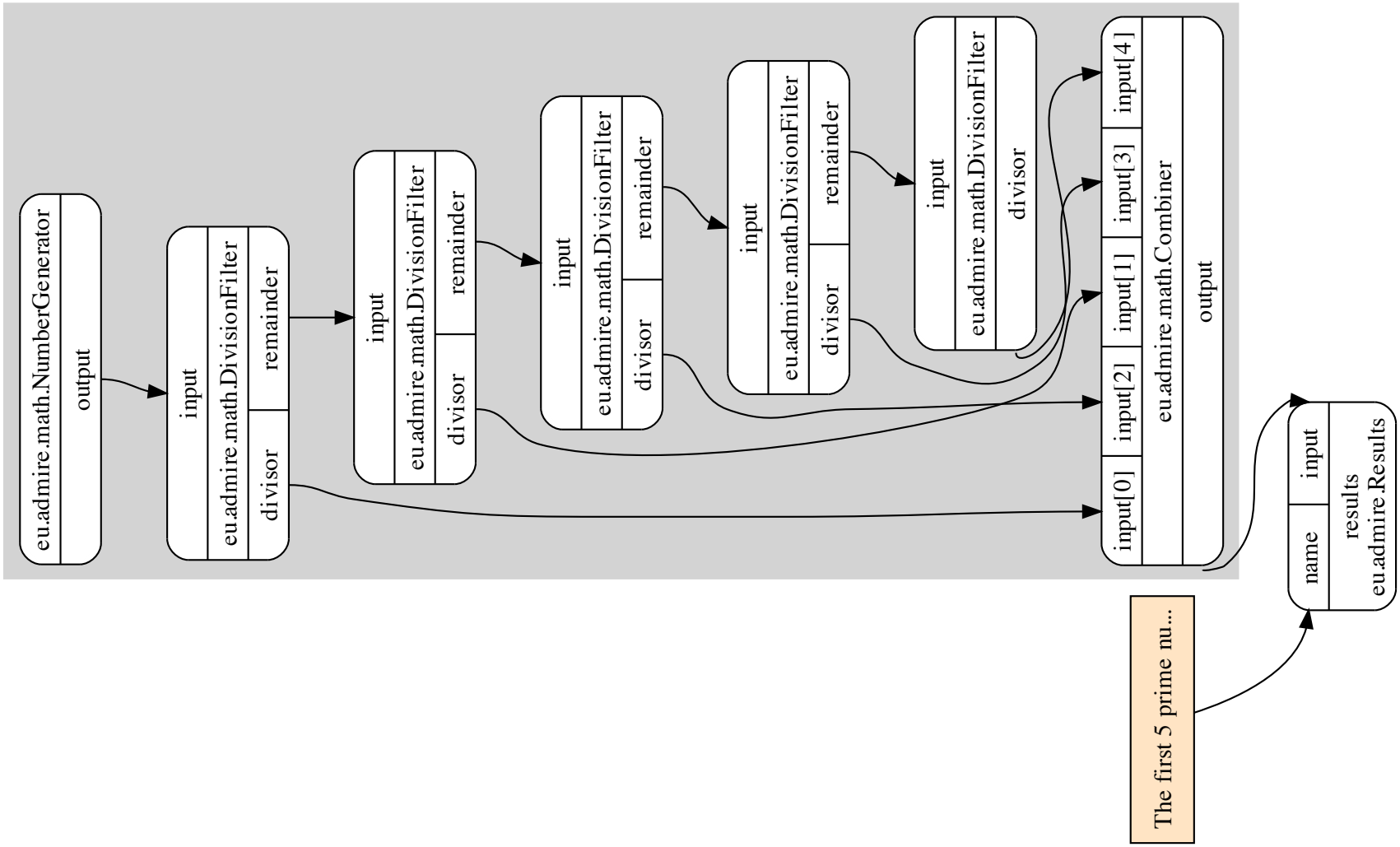
Sieve of Eratosthenes

```
1  package tutorial.example {
2      use dispel.core.Interpolate;
3      use tutorial.example.SieveFilter;
4      use dispel.math.PrimeGenerator;
5
6      // Define sieve constructor.
7      PE<PrimeGenerator> makeSieveOfEratosthenes(Integer count) {
8          // Instantiate internal components.
9          SieveFilter filter      = new SieveFilter[count];
10         Interpolate interpolate = new Interpolate with roundrobin inputs, input.length = count;
11
12         // Initialise sieve elements.
13         for (Integer i = 0; i < count - 1; i++) filter[i] = new SieveFilter with terminator output;
14         filter[count - 1] = new SieveFilter with terminator prime;
15
16         // Construct internal workflow.
17         |-x for x in 2..-| => filter[0].input;
18         for (Integer i = 0; i < count - 1; i++) {
19             filter[i].unfiltered => filter[i + 1].input;
20             filter[i].filtered   => interpolate.inputs[i];
21         }
22         filter[count - 1].unfiltered => discard;
23         filter[count - 1].filtered   => interpolate.input[count - 1];
24         filter[count - 1].filtered   => stop;
25
26         // Return all primes generated.
27         return PE( <> => <Connection primes = interpolate.output> );
28     }
29
30     register makeSieveOfEratosthenes;
31 }
```

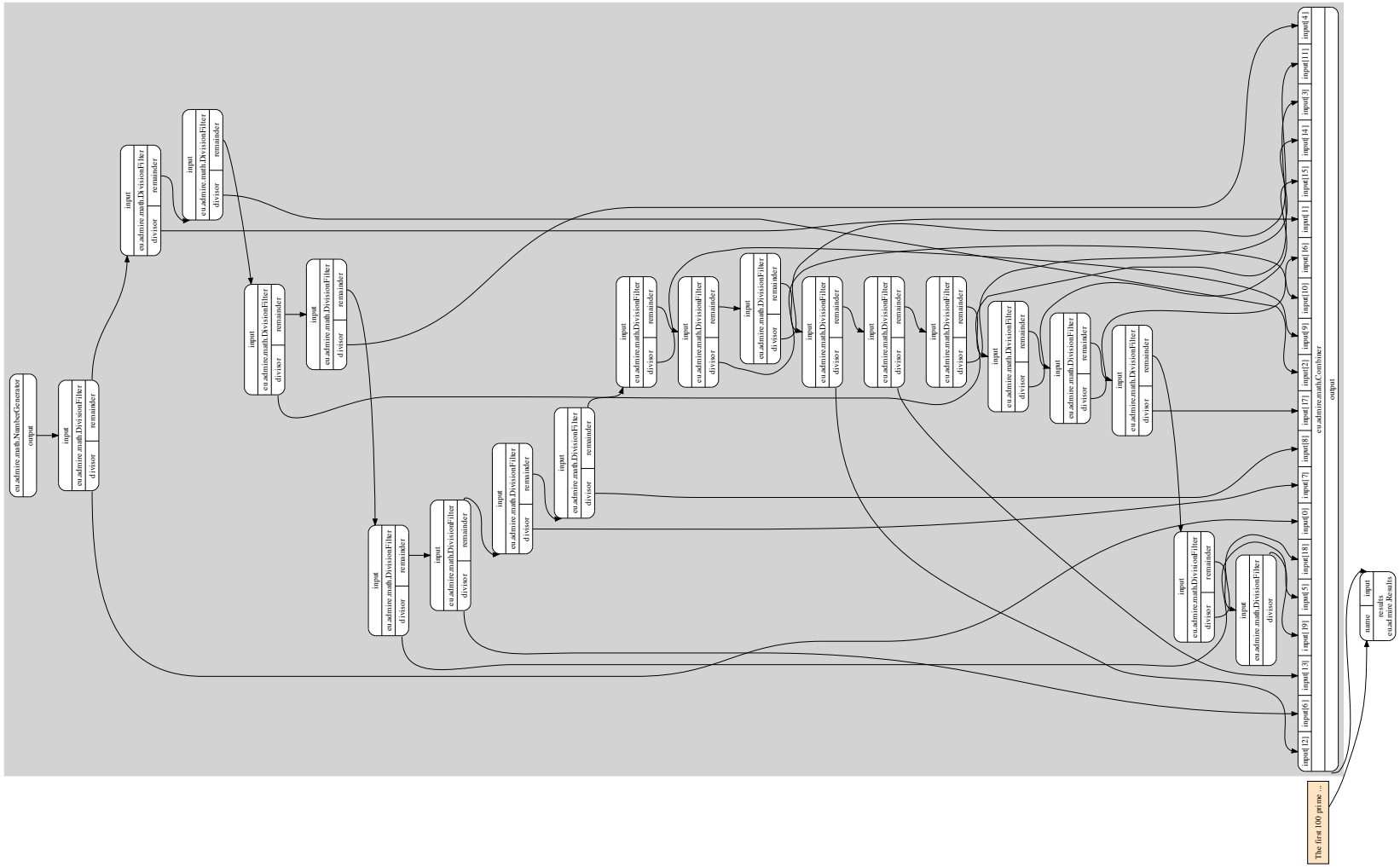
Sieve of Eratosthenes

```
1   package tutorial.example {
2       // Import abstract PE type and constructor.
3       use dispel.math.PrimeGenerator;
4
5       // Construct the sieve.
6       PE <PrimeGenerator> SoE100 = makeSieveOfEratosthenes(100);
7       SoE100  sieve100 = new SoE100;
8       Results results  = new Results;
9
10      // Construct the top-level workflow.
11      |-"100 prime numbers"-| => results.name;
12      sieve100.primes         => results.input;
13
14      // Submit the workflow.
15      submit;
16  }
```

Sieve of Eratosthenes



Sieve of Eratosthenes



Sieve of Eratosthenes