Janus: Fine-grained and efficient provenance querying for Taverna

Paolo Missier

Information Management Group

School of Computer Science, University of Manchester, UK

in collaboration with:

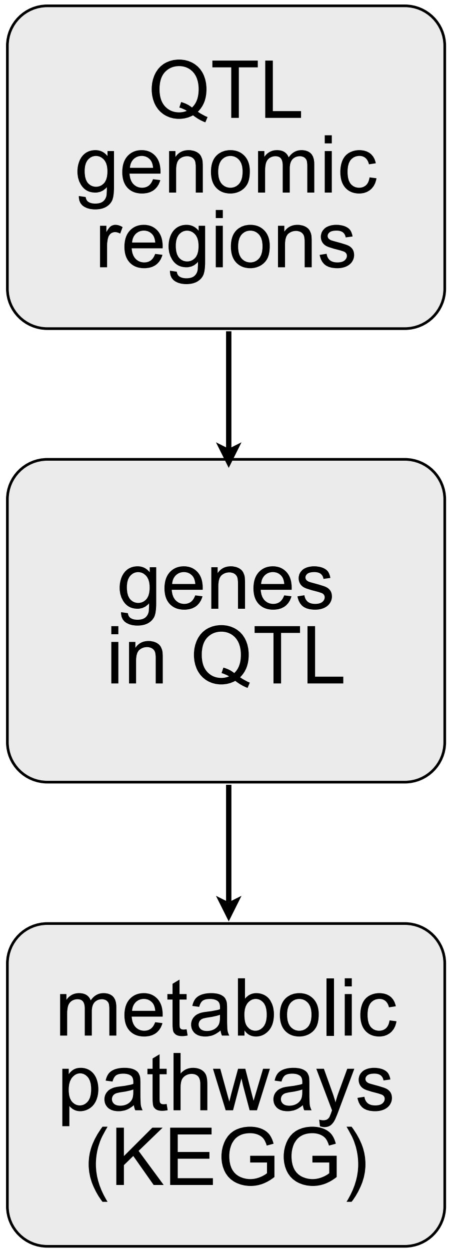
Norman Paton, Khalid Belhajjame

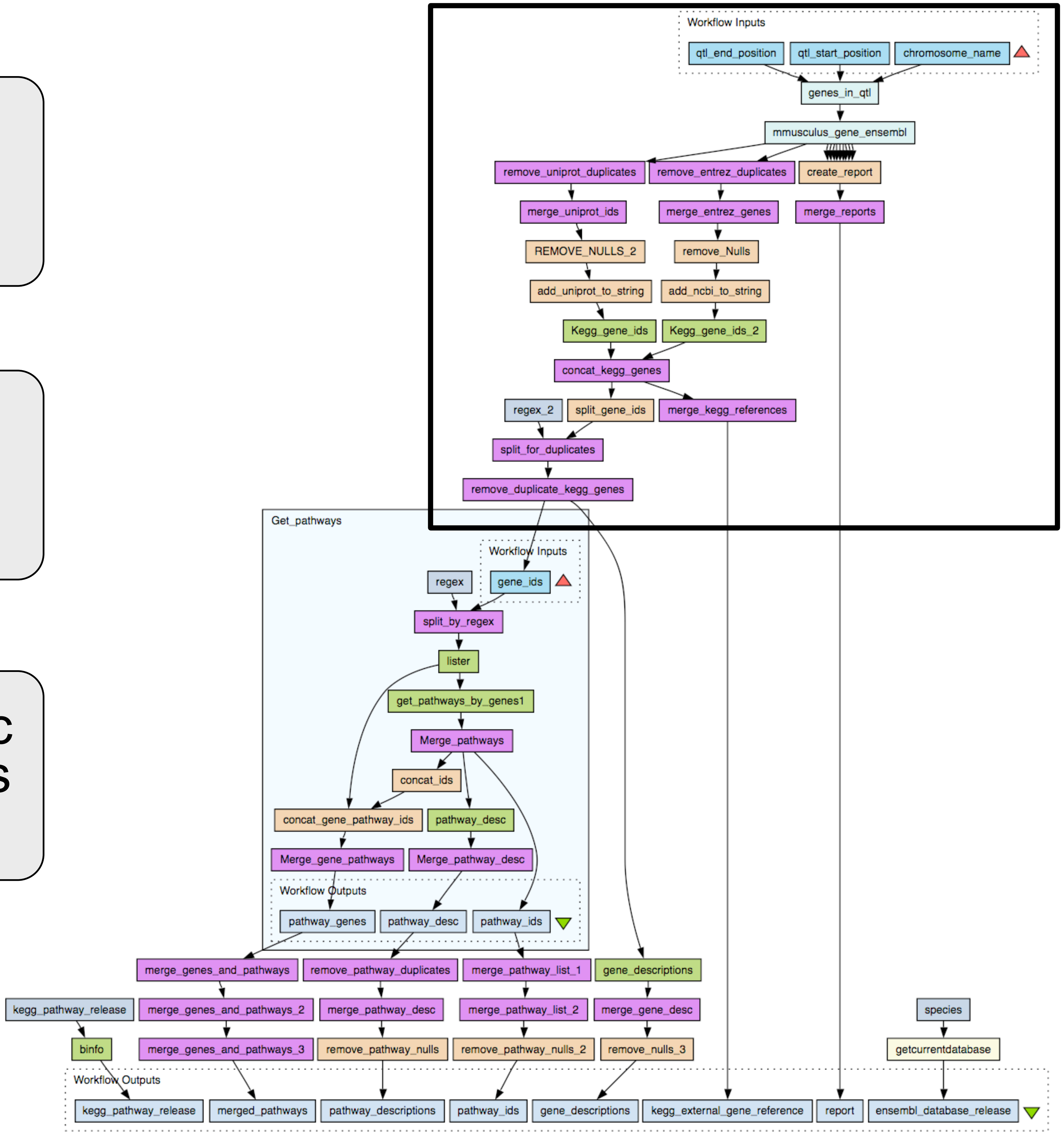
IMG Seminar, June 2009

Outline

- Motivation: Taverna for rapid information integration
- Fine-grained process provenance and its role in Taverna
- Context and scope: forms and uses for process provenance
- Technical challenges in querying provenance traces
- A solution, and experimental results

Workflow as data integrator





Workflow as data integrator

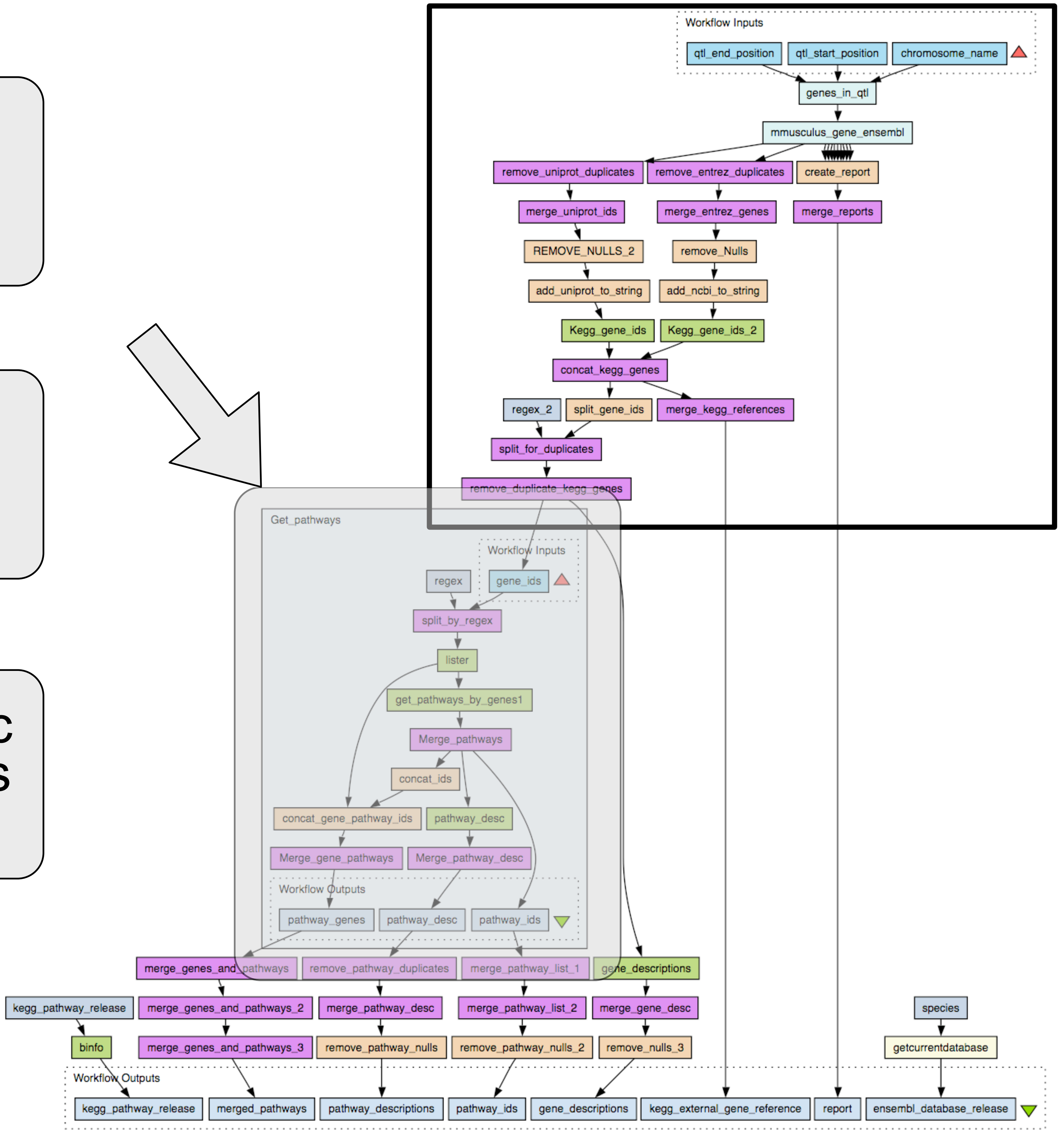
QTL genomic regions

↓

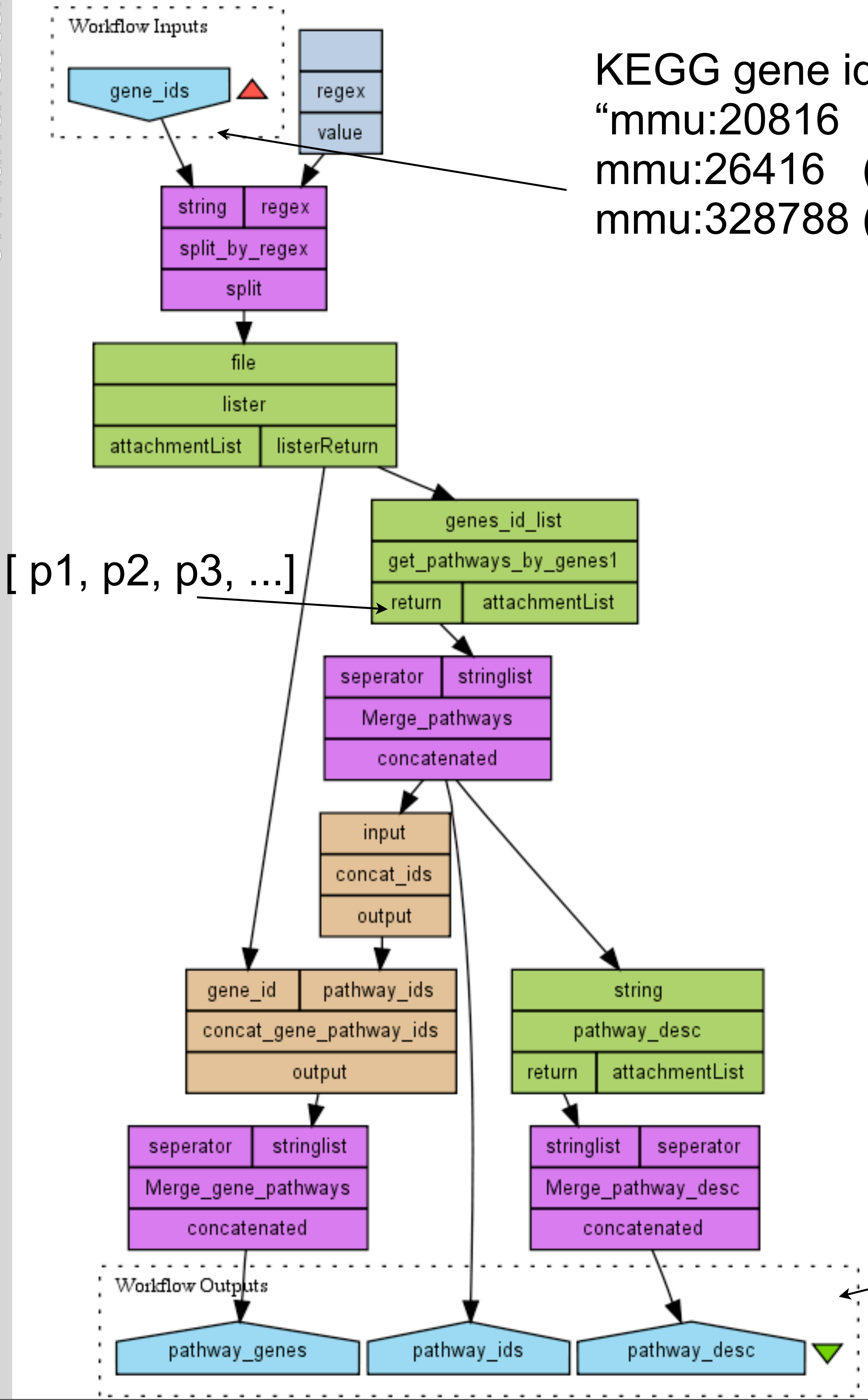
genes in QTL

↓

metabolic pathways (KEGG)



The genes→pathways workflow in action



KEGG gene ids:
"mmu:20816 (g1)
mmu:26416 (g2)
mmu:328788 (g3)"

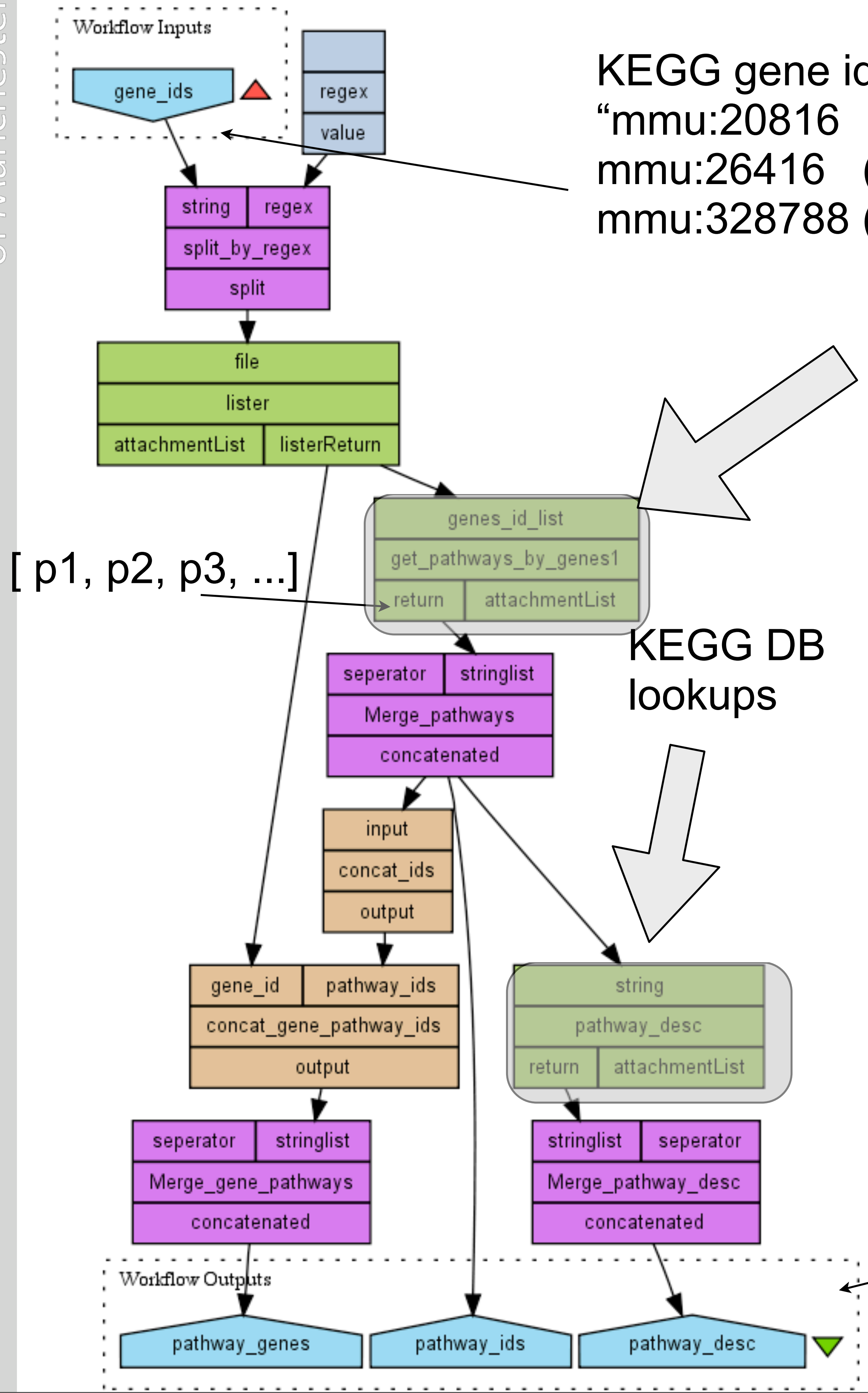
[p1, p2, p3, ...]

"p1 MAPK signaling pathway
p2 VEGF signaling pathway
... "

MANCHESTER 1824

The University of Manchester

The genes→pathways workflow in action

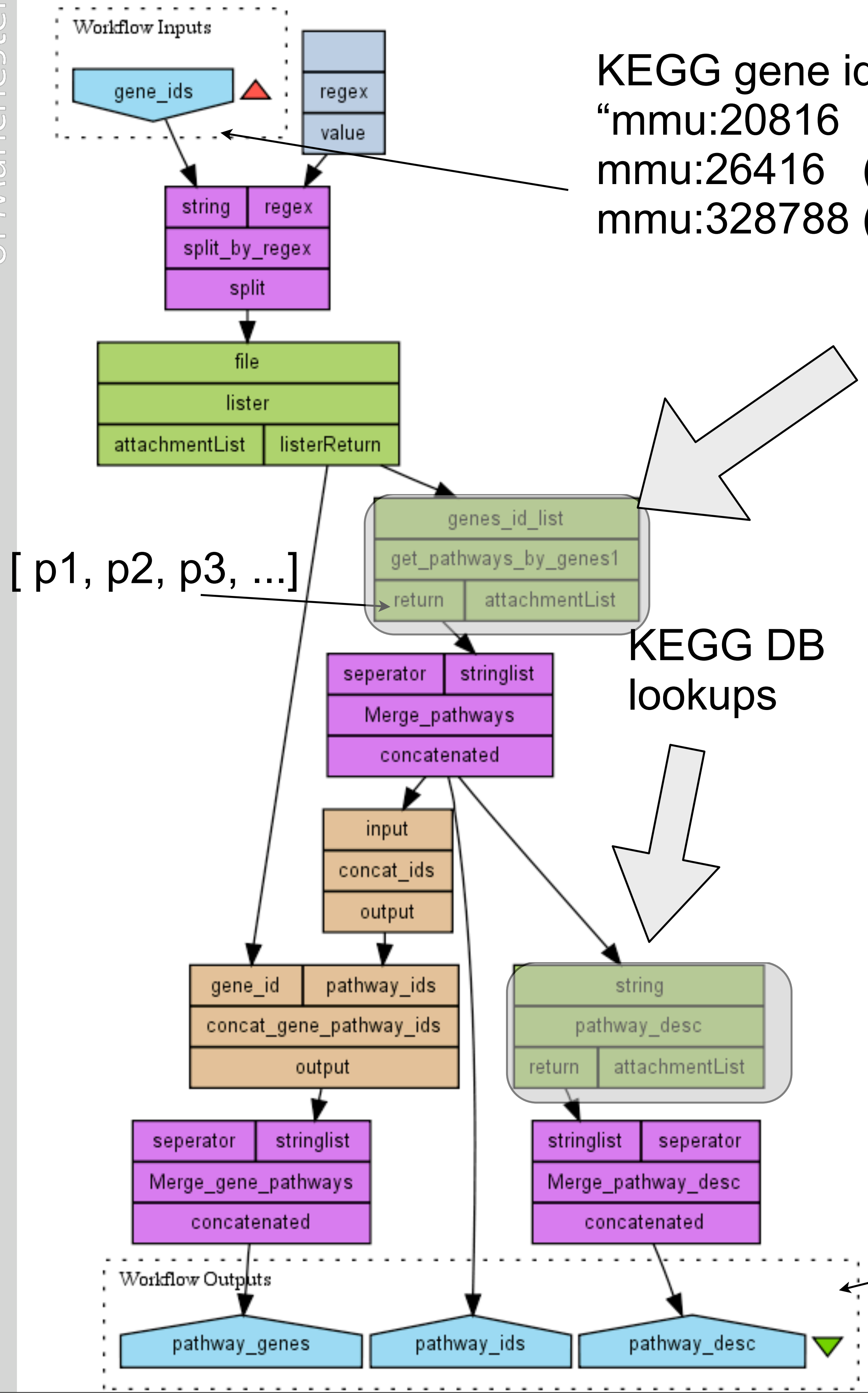


KEGG gene ids:
“mmu:20816 (g1)
mmu:26416 (g2)
mmu:328788 (g3)”

KEGG DB lookups

“p1 MAPK signaling pathway
p2 VEGF signaling pathway
... “

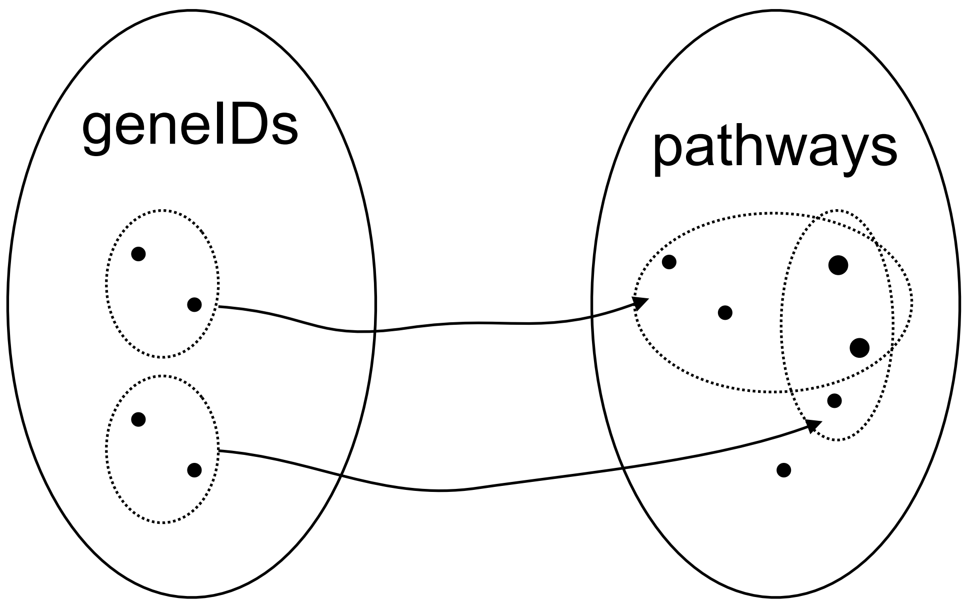
The genes→pathways workflow in action



KEGG gene ids:
"mmu:20816 (g1)
mmu:26416 (g2)
mmu:328788 (g3)"

[p1, p2, p3, ...]

KEGG DB lookups

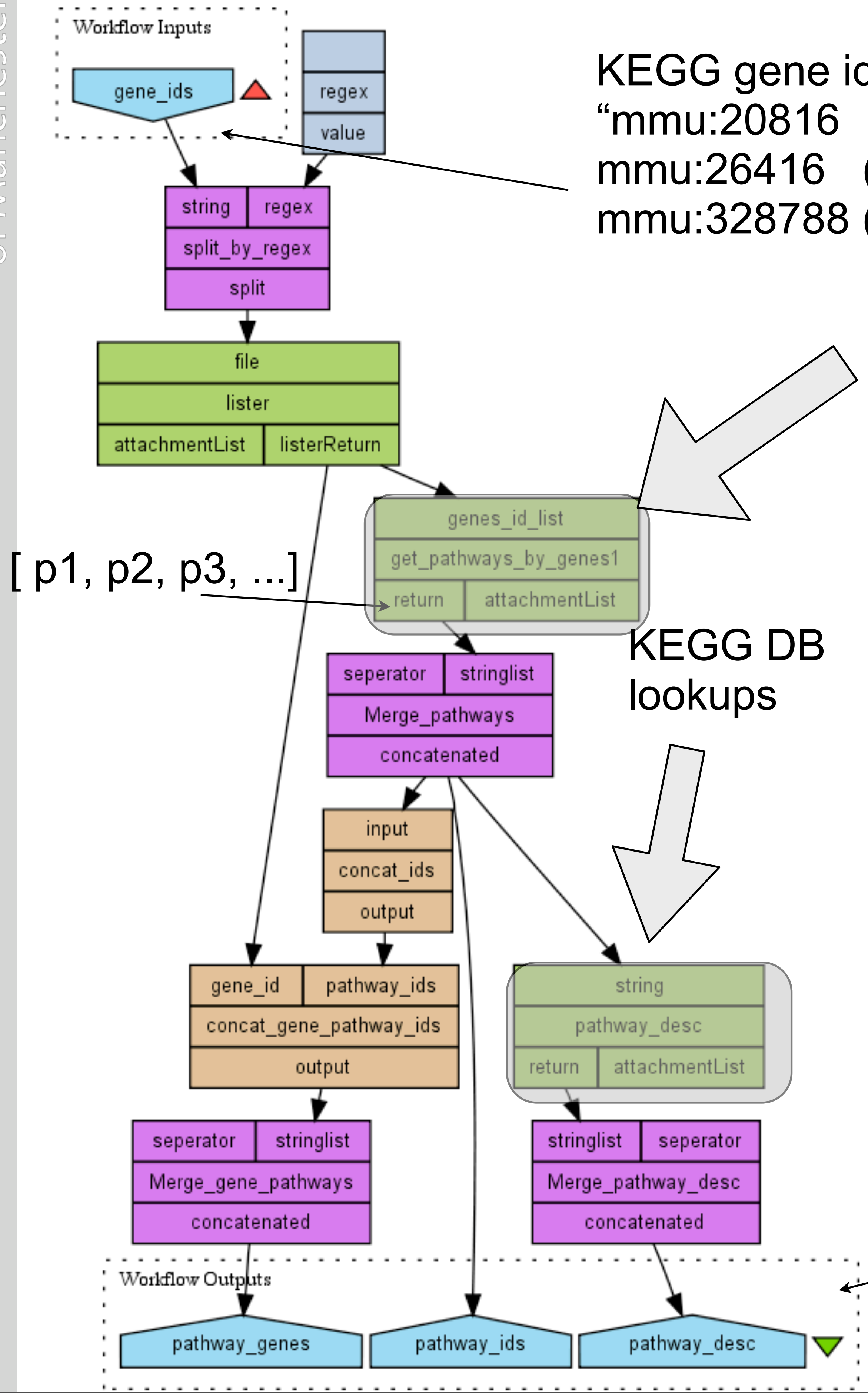


"p1 MAPK signaling pathway
p2 VEGF signaling pathway
... "

MANCHESTER 1824

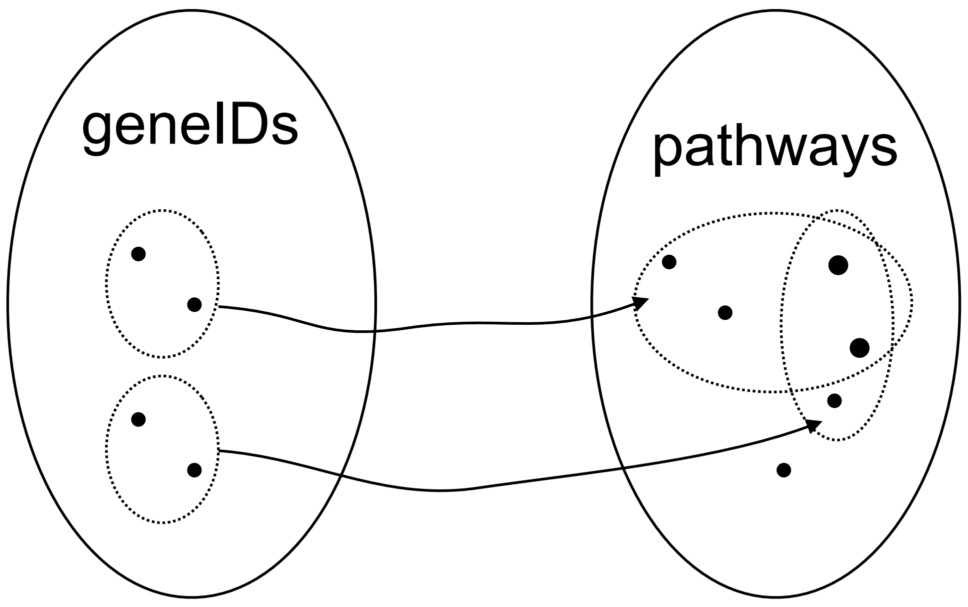
The University of Manchester

The genes→pathways workflow in action



KEGG gene ids:
"mmu:20816 (g1)
mmu:26416 (g2)
mmu:328788 (g3)"

KEGG DB lookups

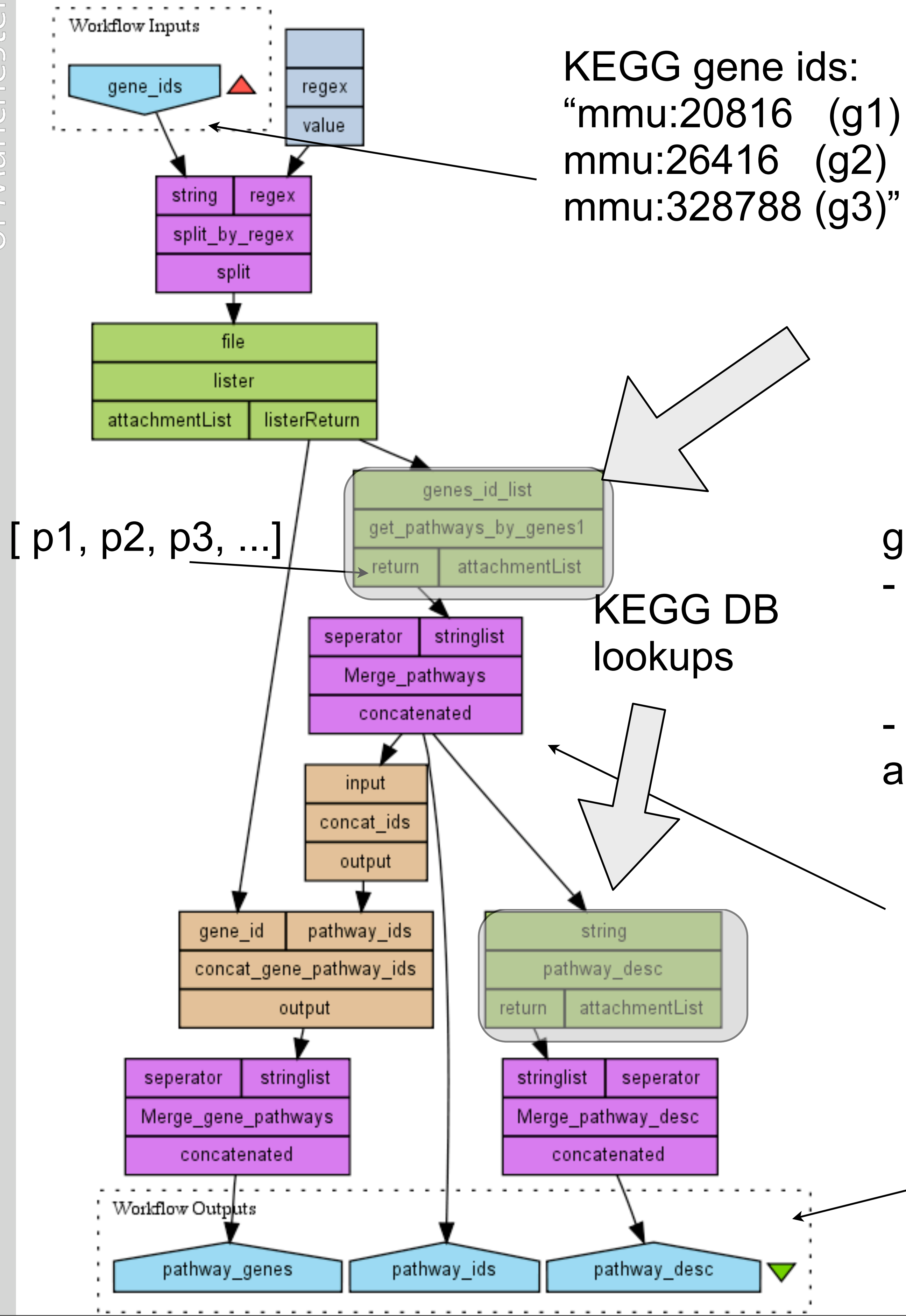


goal:
- list all pathways that are mapped from both sets of genes

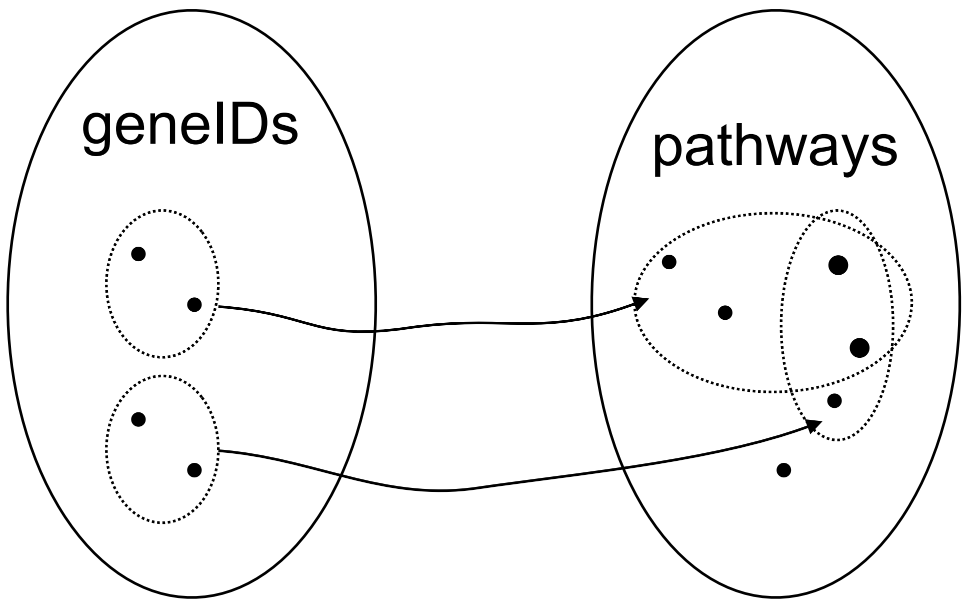
- substantial list manipulation involved in achieving this

"p1 MAPK signaling pathway
p2 VEGF signaling pathway
... "

The genes→pathways workflow in action



KEGG gene ids:
"mmu:20816 (g1)
mmu:26416 (g2)
mmu:328788 (g3)"



goal:
- list all pathways that are mapped from both sets of genes

- substantial list manipulation involved in achieving this

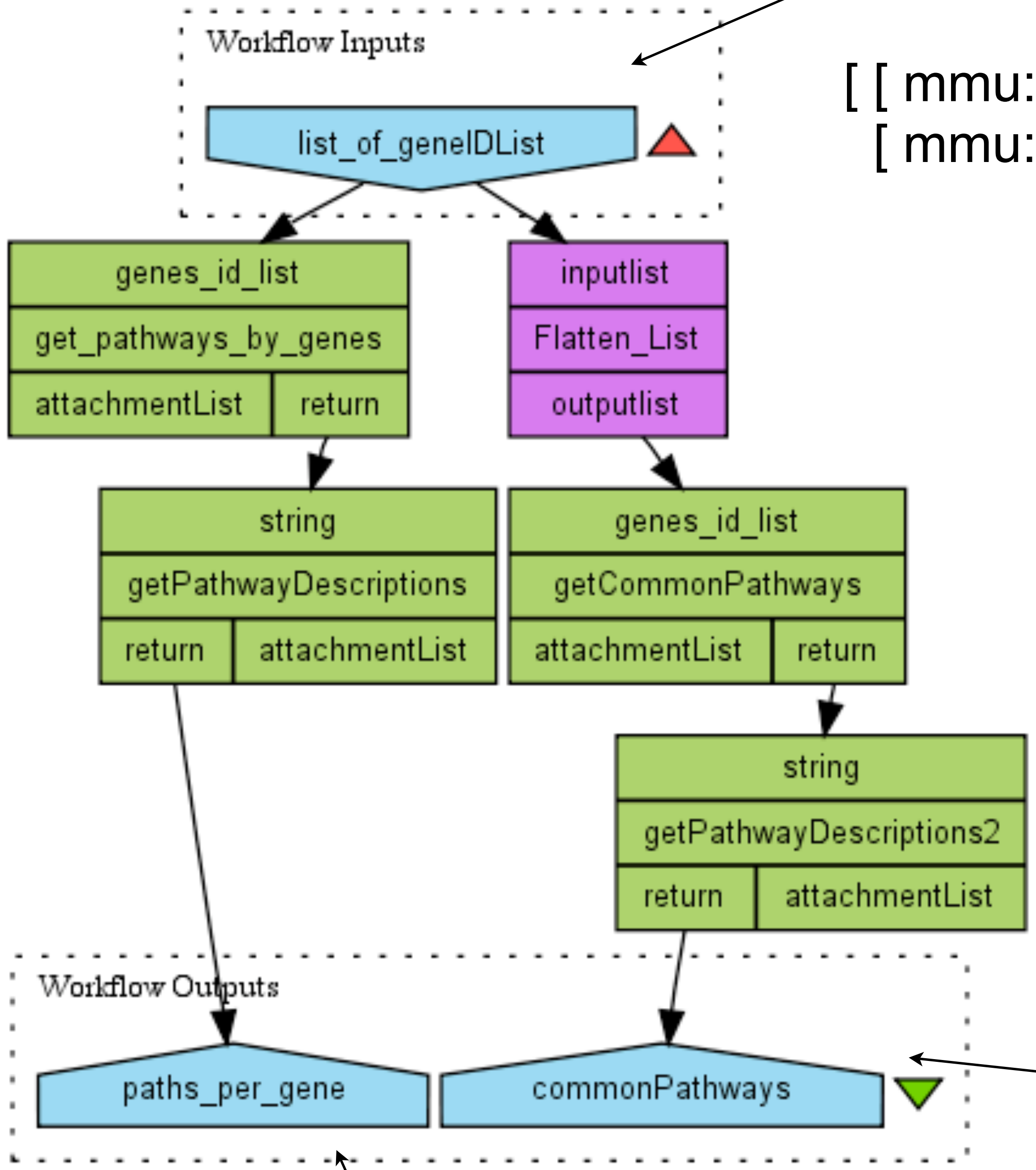
[[g1, g2, g3],
["g1 p1 p2 ...", "g2 p1 p2 ..." ,"g3 p1 p2 ..."]
[g1, g2, g3]]

"p1 MAPK signaling pathway
p2 VEGF signaling pathway
... "

The University of Manchester

MANCHESTER 1824

An alternative design



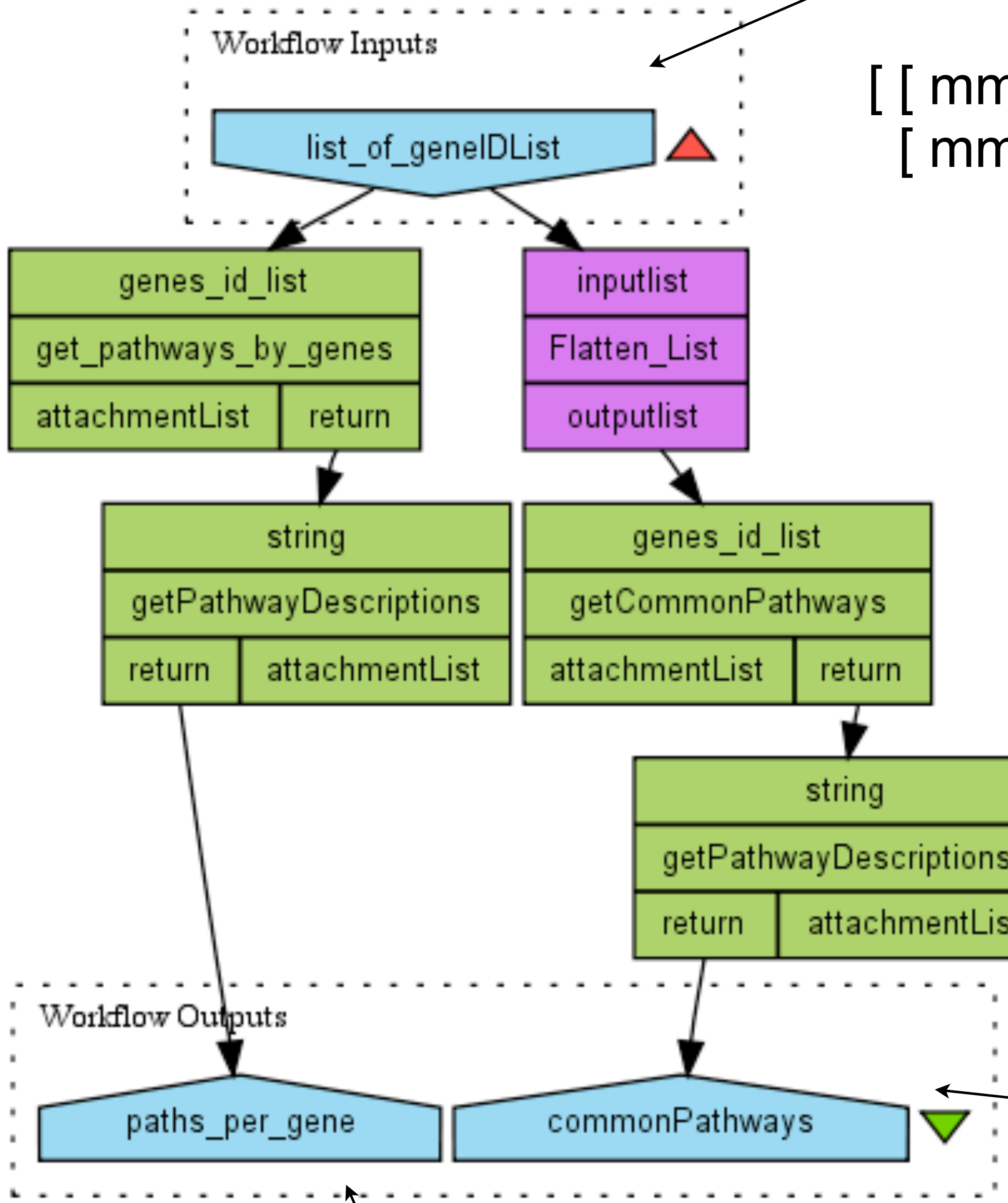
List-structured
KEGG gene ids:

[[mmu:26416],
  [mmu:328788]]

[path:mmu04010 MAPK signaling,
  path:mmu04370 VEGF signaling]

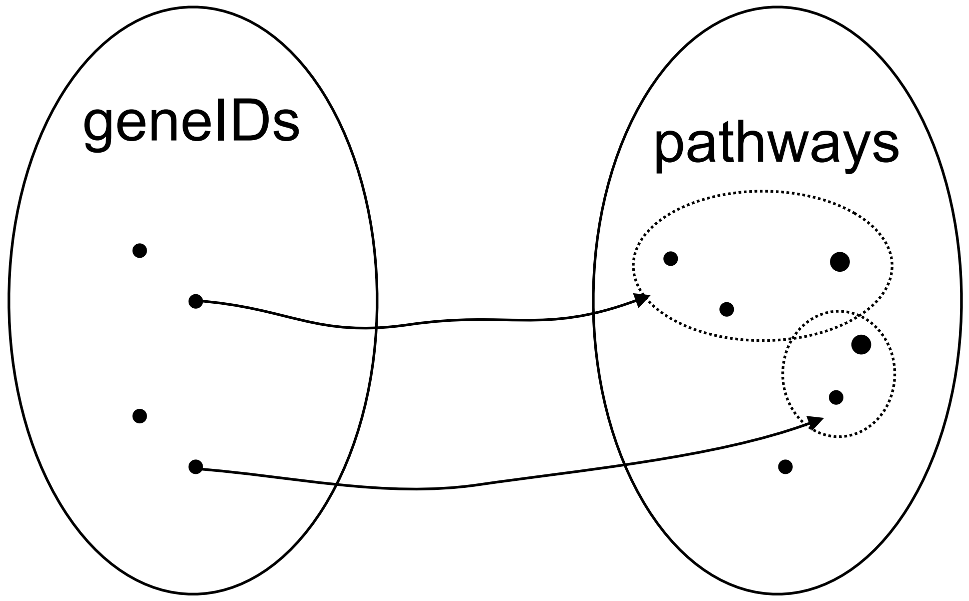
[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
  [path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

An alternative design



List-structured
KEGG gene ids:

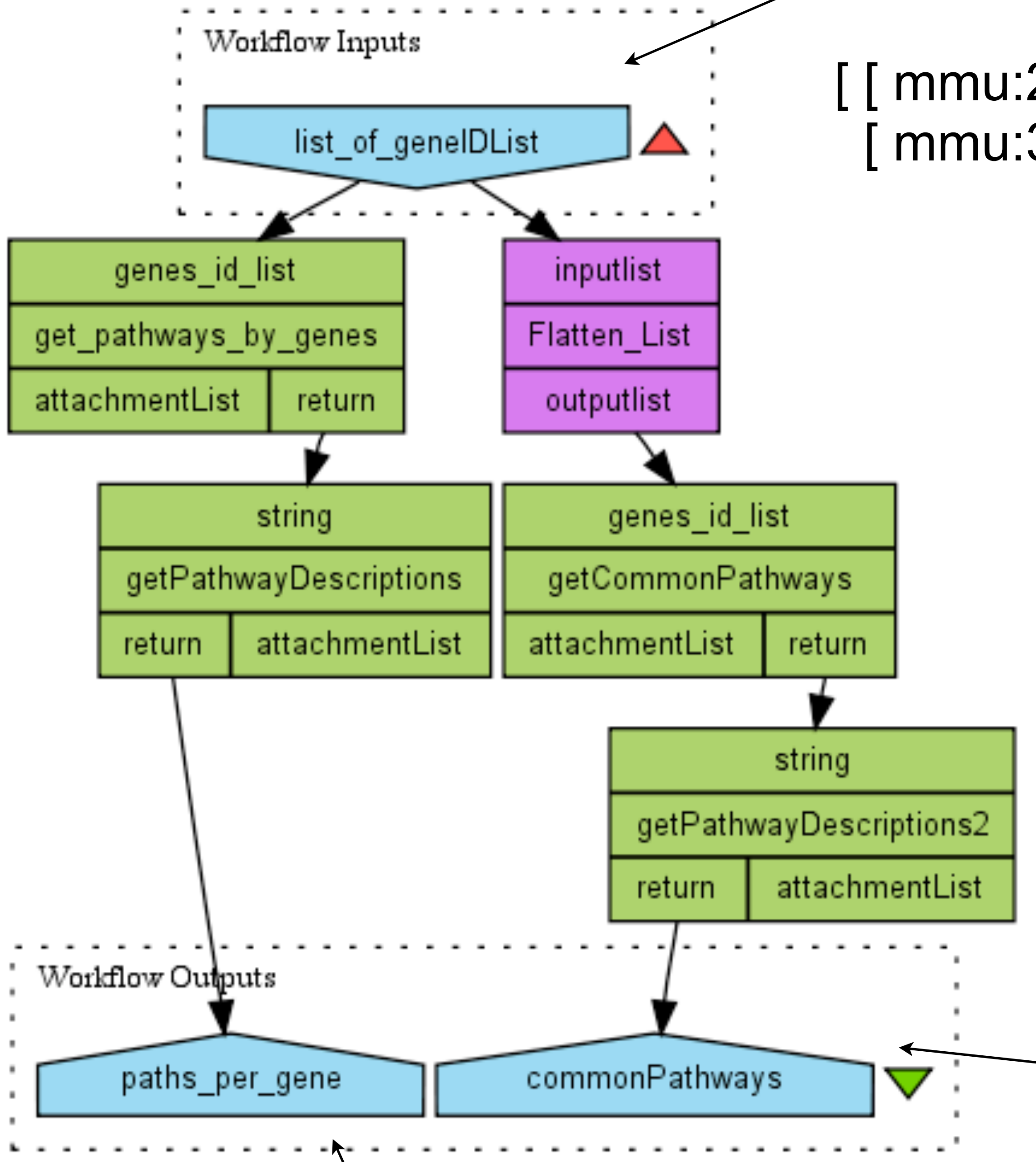
[[mmu:26416],
[mmu:328788]]



[path:mmu04010 MAPK signaling,
path:mmu04370 VEGF signaling]

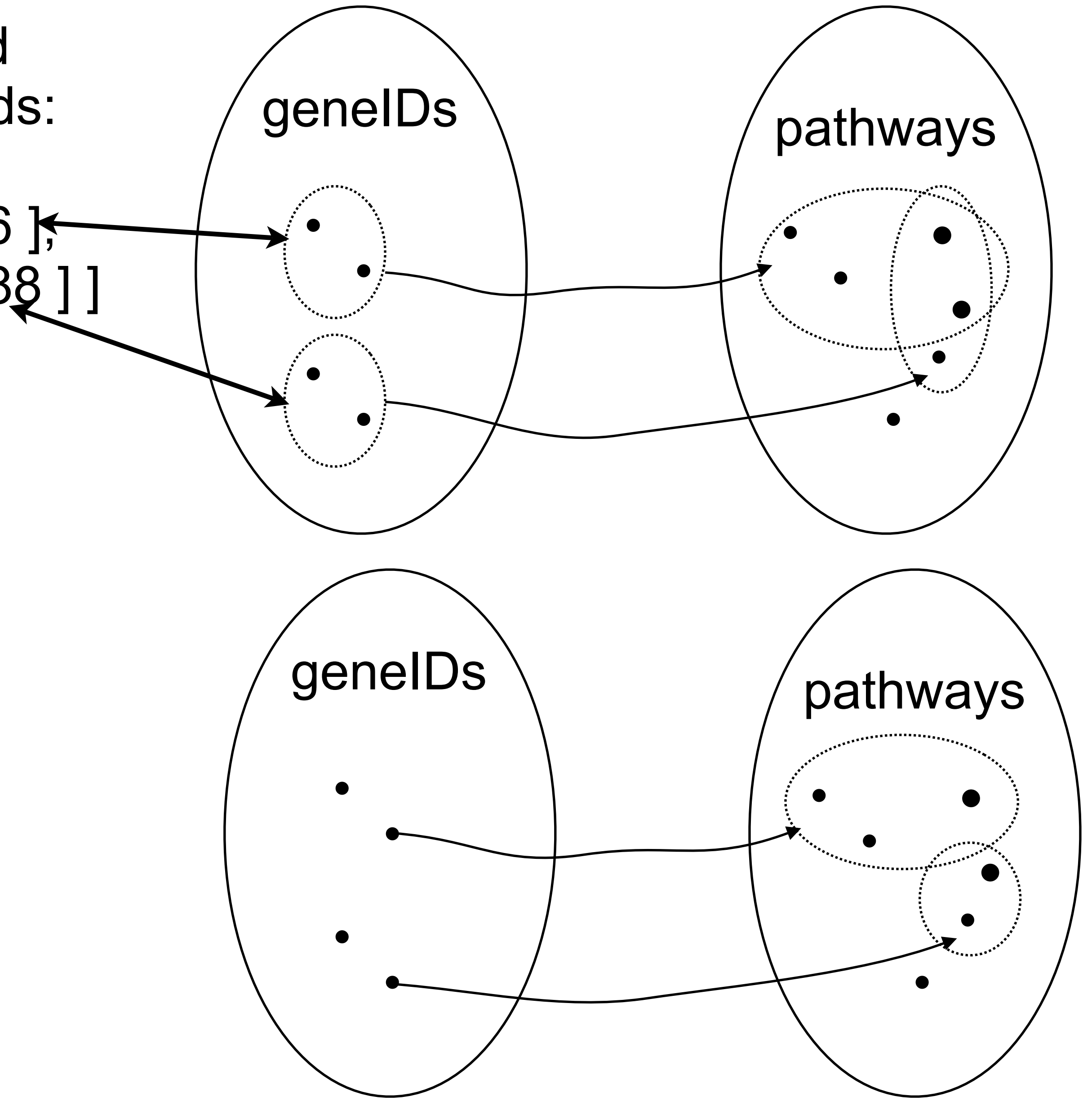
[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
[path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

An alternative design



List-structured KEGG gene ids:

[[mmu:26416],
  [mmu:328788]]



[path:mmu04010 MAPK signaling,
  path:mmu04370 VEGF signaling]

[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
  [path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

MANCHESTER 1824

The University of Manchester

Is either design better?

- Pros:
  - simpler to design and understand (hopefully)
  - (no shims!)
  - accepts multiple gene sets
    - returns list of pathways separately for each gene set
    - in addition to those shared by the union of all sets
- Cons:
  - no genes in output list:

    [[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
    [path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

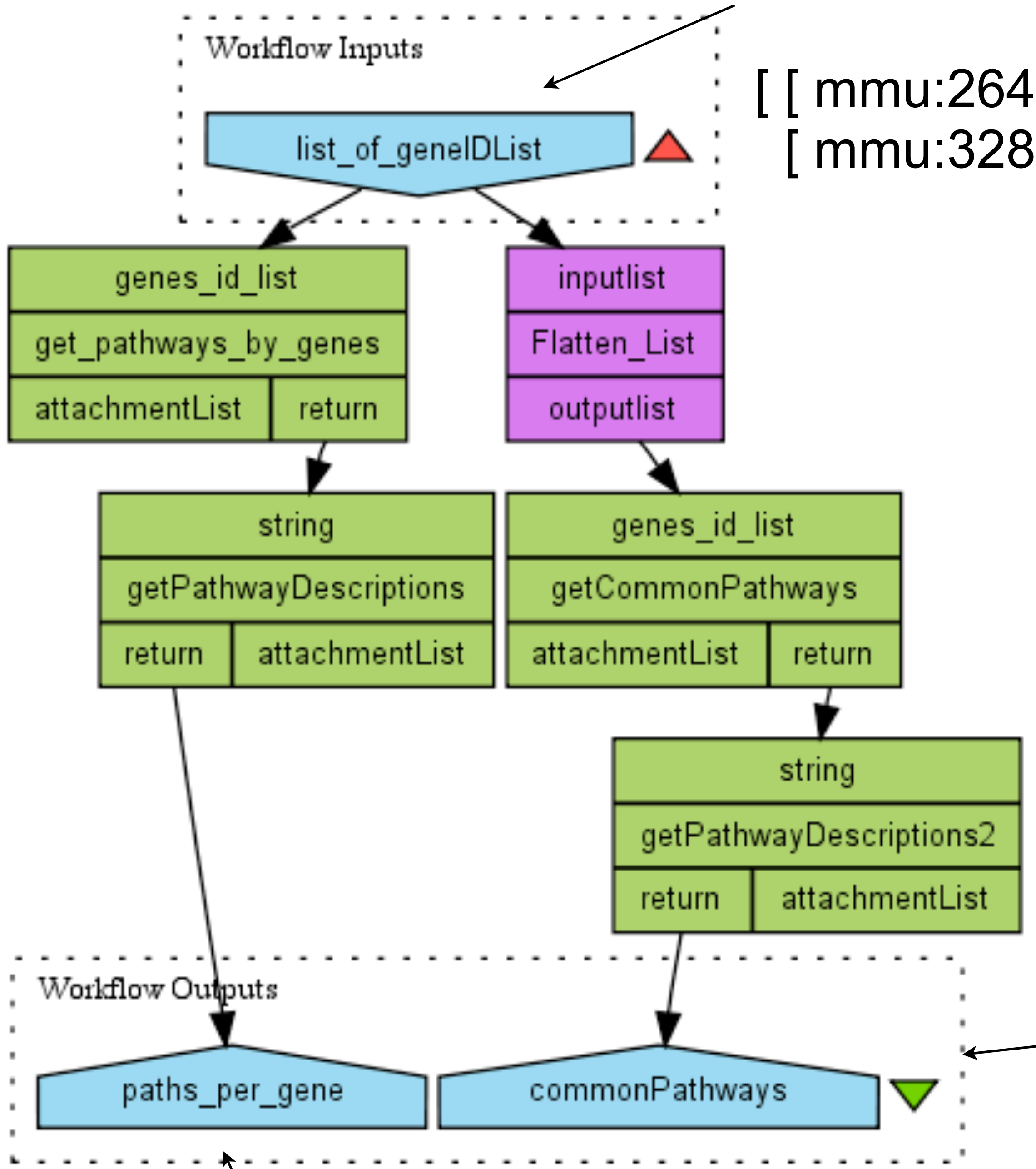
  - so the relationship between the gene set and the pathway set is lost...

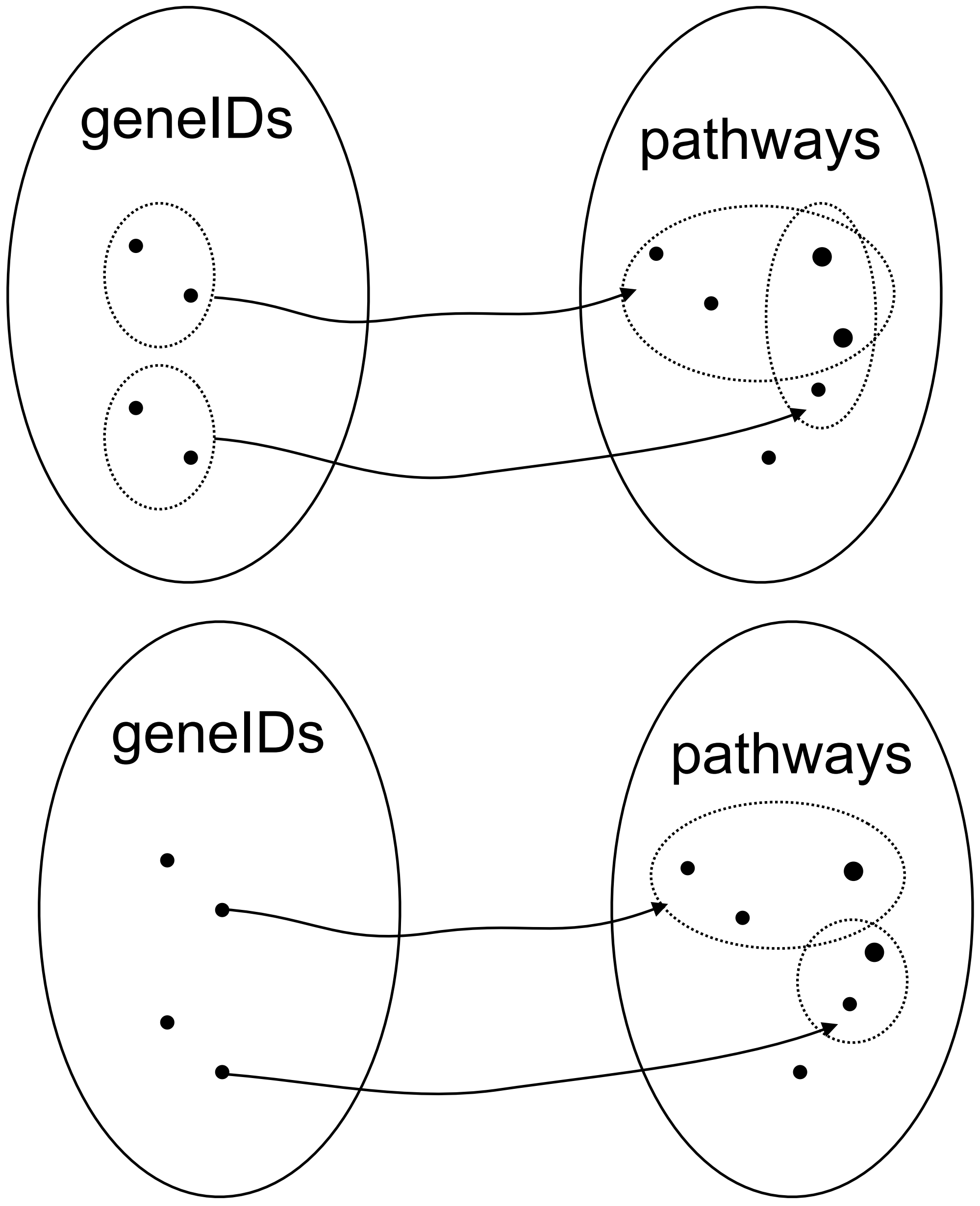
...Or is it?

Provenance trace to the rescue

List-structured KEGG gene ids:

[[mmu:26416],
  [mmu:328788]]





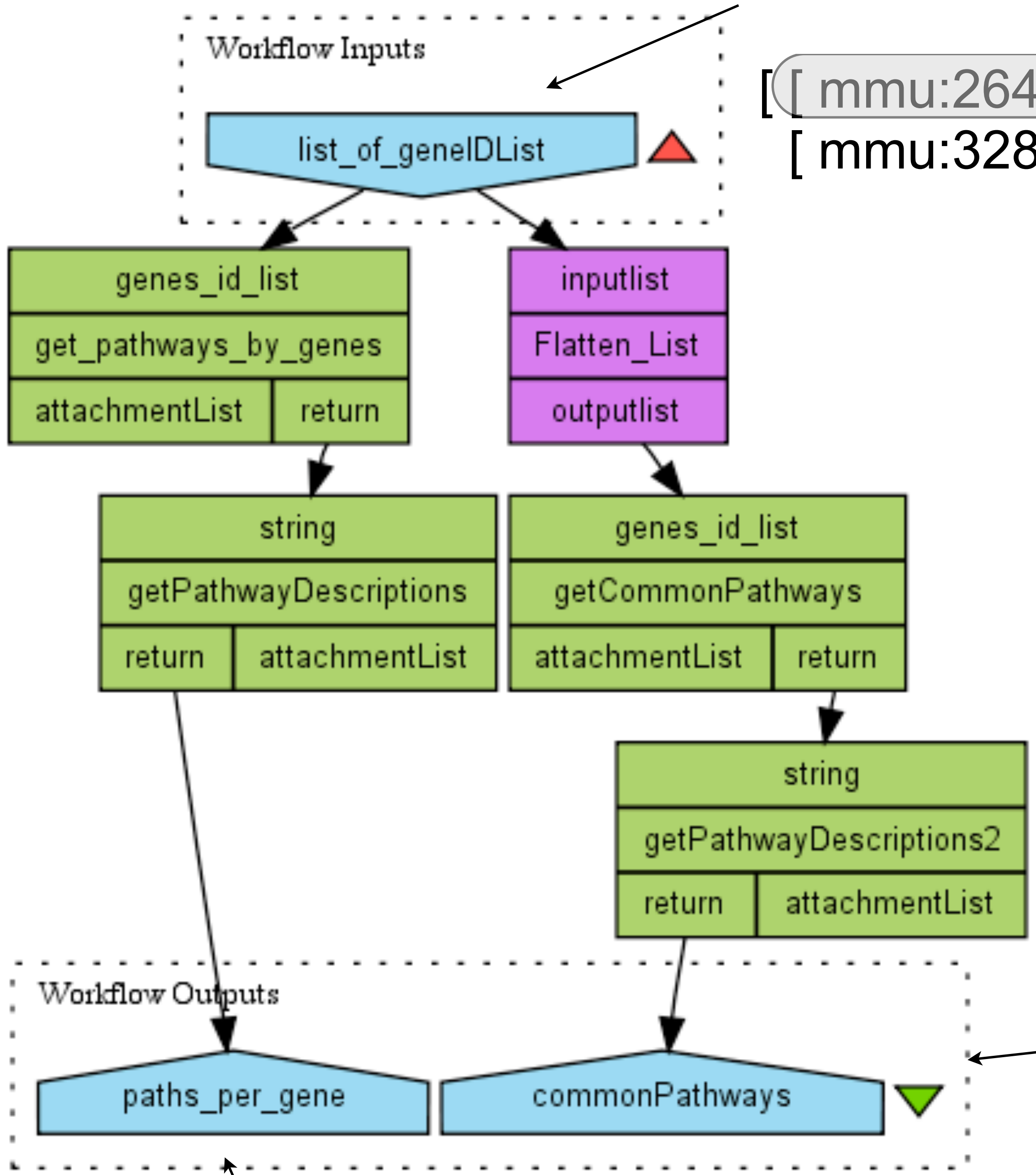
[path:mmu04010 MAPK signaling,
  path:mmu04370 VEGF signaling]

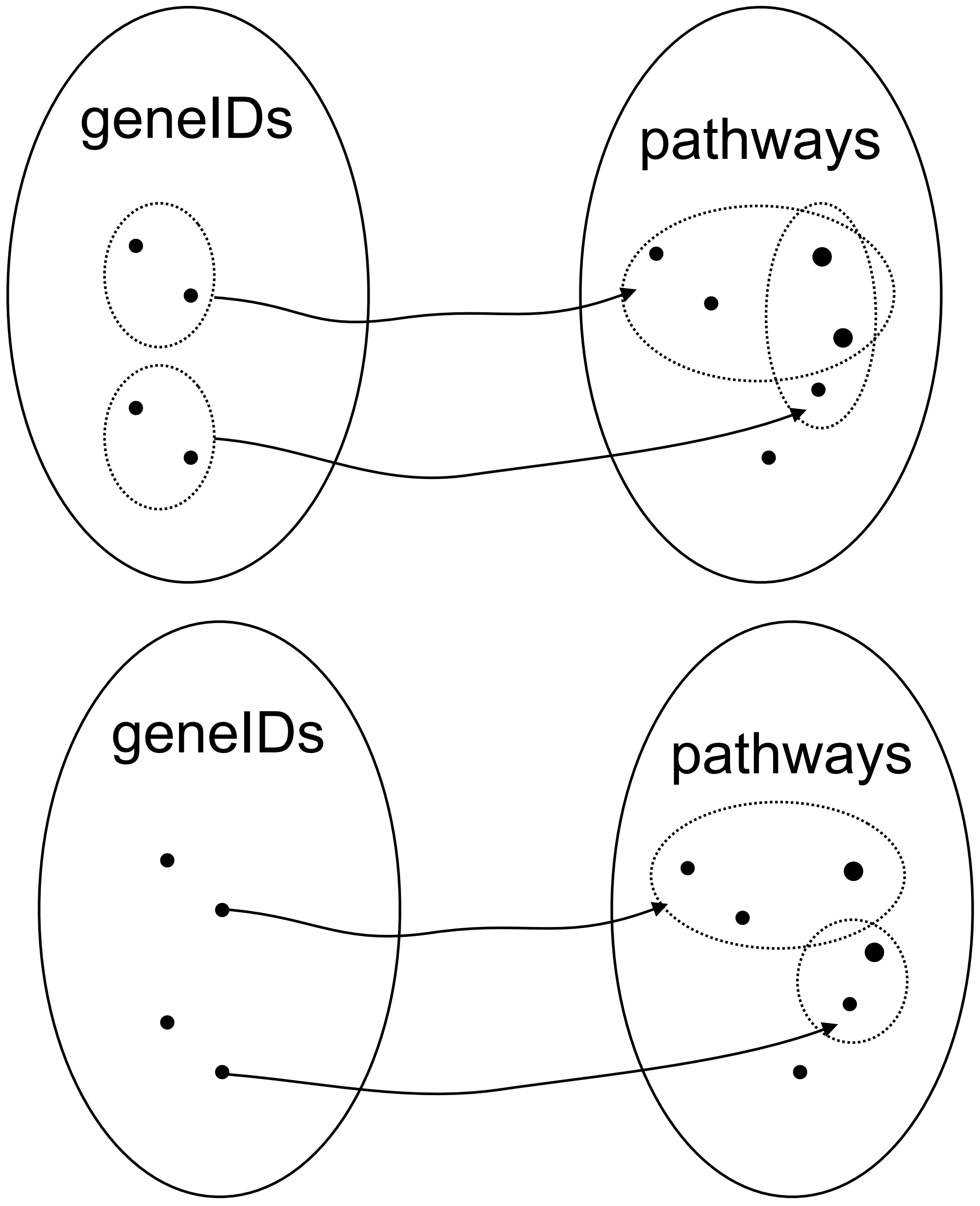
[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
  [path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

Provenance trace to the rescue

List-structured
KEGG gene ids:

[[mmu:26416],
[mmu:328788]]





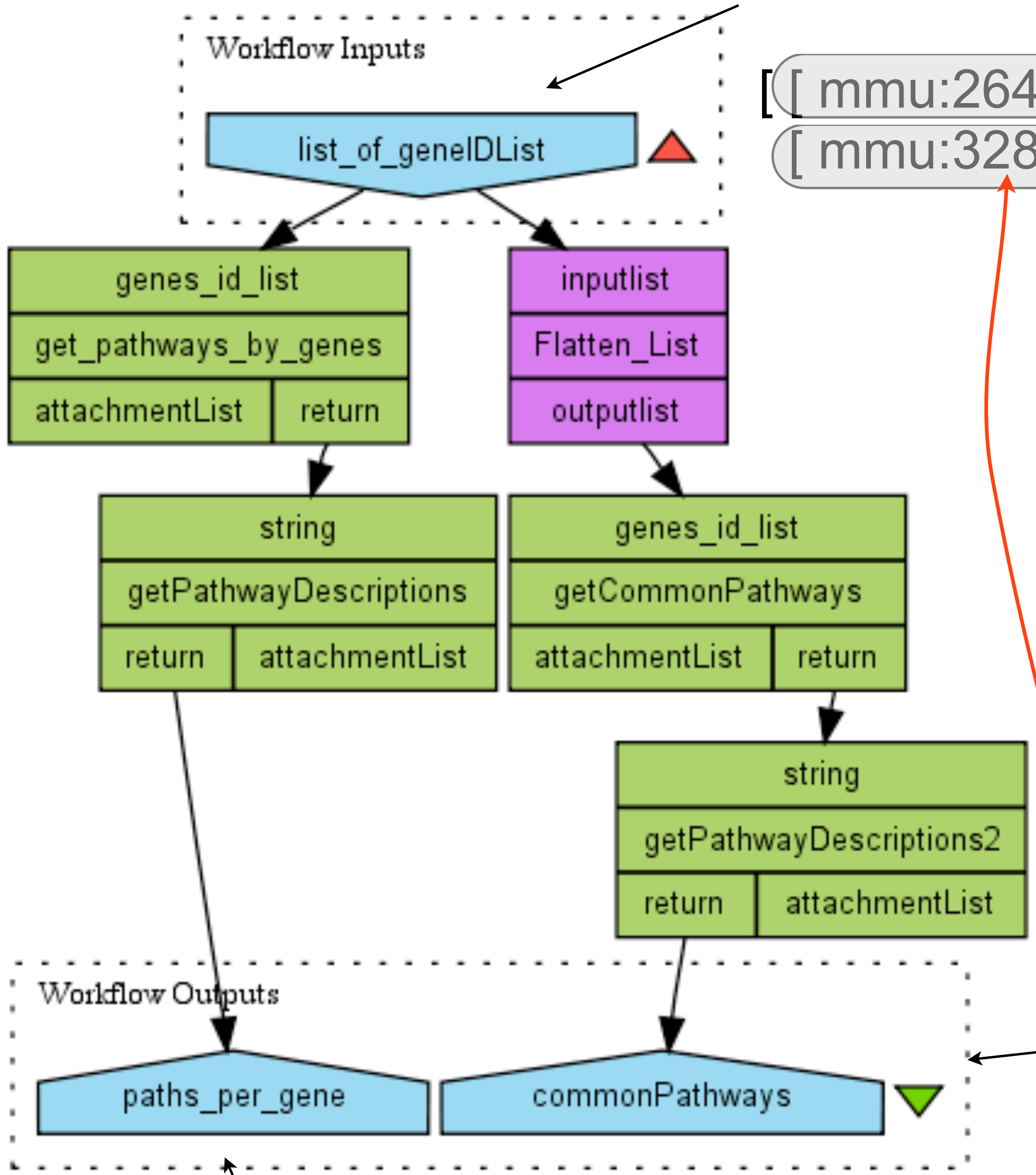
[path:mmu04010 MAPK signaling,
path:mmu04370 VEGF signaling]

[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
[path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

The University of Manchester

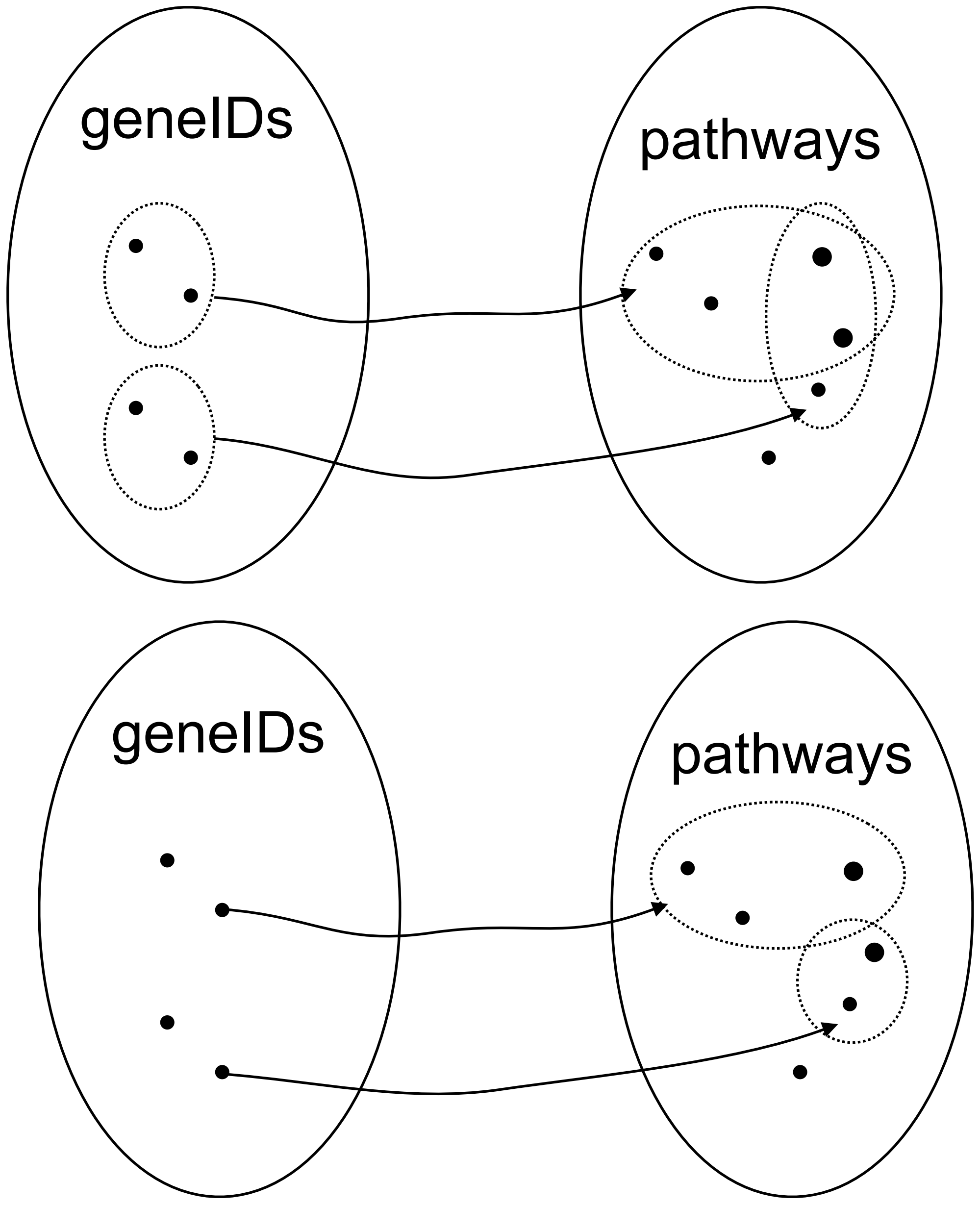
MANCHESTER 1824

Provenance trace to the rescue



List-structured KEGG gene ids:

[[mmu:26416], [mmu:328788]]



[path:mmu04010 MAPK signaling, path:mmu04370 VEGF signaling]

[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...], [path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

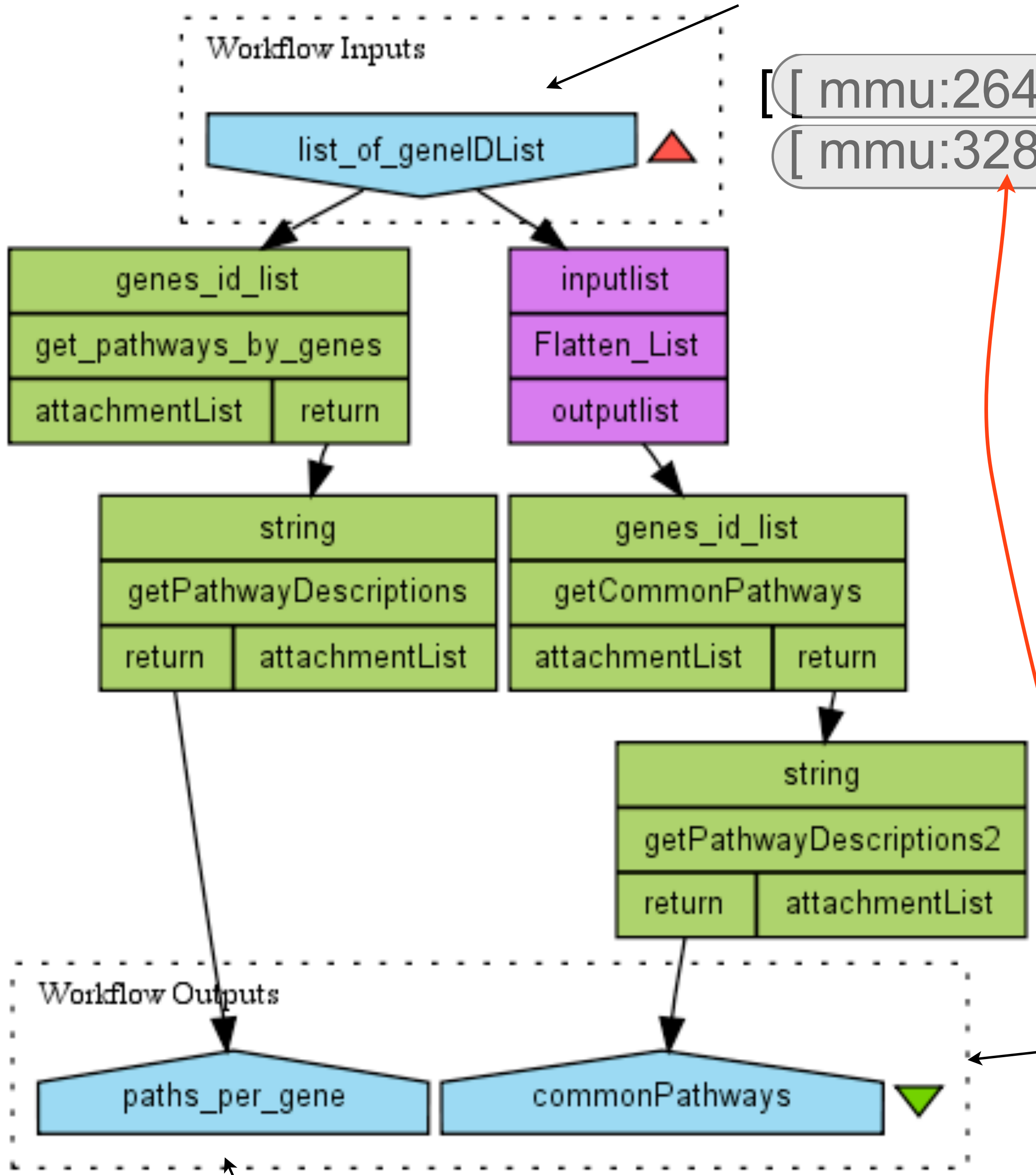
The University of Manchester

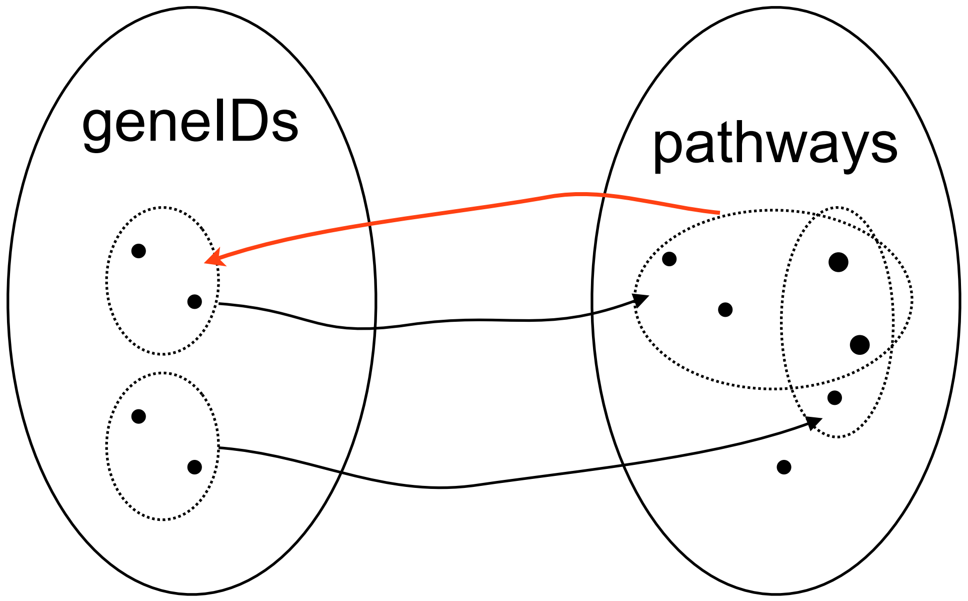
MANCHESTER 1824

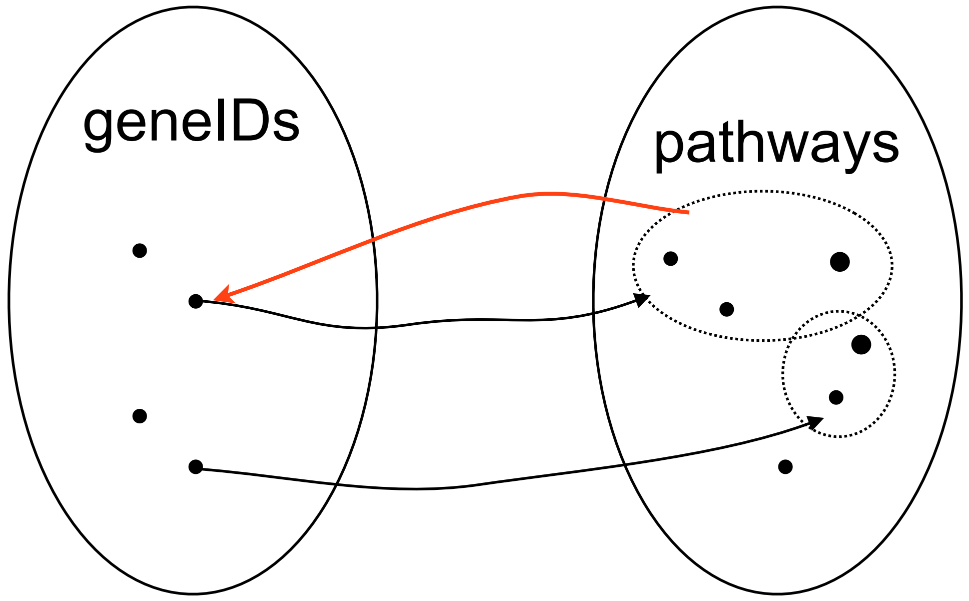
Provenance trace to the rescue

List-structured KEGG gene ids:

[[mmu:26416],
[mmu:328788]]







[path:mmu04010 MAPK signaling,
path:mmu04370 VEGF signaling]

[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
[path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

MANCHESTER 1824

The University of Manchester

Taverna + provenance

- Taverna type system: strings + nested lists
  - “cat”, [“cat”, “dog”], [[“cat”, “dog”], [“large”, “small”]]
- Taverna dataflow model: data-driven execution
  - services activate when input is ready
- Workflow provenance: a detailed trace of workflow execution
  - which services were executed
  - when
  - inputs used, outputs produced

Taverna + provenance

- Taverna type system: strings + nested lists
  - "cat", ["cat", "dog"], [["cat", "dog"], ["large", "small"]]
- Taverna dataflow model: data-driven execution
  - services activate when input is ready
- Workflow provenance: a detailed trace of workflow execution
  - which services were executed
  - when
  - inputs used, outputs produced

Taverna dataflow model + provenance traces
can be a powerful combination

MANCHESTER 1824

The University of Manchester

Some additional user questions

- Causal relations:
  - ✓ which pathway sets come from which gene sets?
  - which processes contributed to producing this image?
  - which process(es) caused this data to be incorrect?
  - which data caused this process to fail?

- Process and data analytics:
  - show me the variations in output in relation to an input parameter sweep (multiple process runs)
  - how often has my favourite service been executed?
    - on what inputs?
  - who produced this data?
  - how often does this pathway turn up when the input genes range over a certain set S?

Forms of provenance ...

Focus is on the data: the observable outcomes from a process

	raw provenance metadata	**provenance metadata + interpretation framework**
design	• process structure (workflow graph) • history of process composition - reuse • process versions	• service annotations: • ex. get_pathways_by_genes • who created /edited: attribution • why: purpose, intent
execution	process events: - service invocation - data production / consumption - causal dependency graphs ex.: - list_of_geneIDList = [a, b, c] - paths_per_gene = [[d,e,f], [g,h,j]] - ... in run #32	- data annotations, results interpretation in terms of conceptual data model: set of pathways → gene sets

...and their uses and associated challenges

	raw provenance metadata	**provenance metadata + interpretation framework**
design	• exploiting semantic properties of the process structure to improve provenance exploitation • exploring process space across versions and structural similarities • graph matching	• semantic-based search of process space
execution	- enabling partial re-runs of resource-intensive workflows - storing very large provenance traces that accumulate over time - efficient query over large traces - presentation of query answers	- semantic-based query answering over annotated traces

...and their uses and associated challenges

	raw provenance metadata	provenance metadata + interpretation framework
design	• exploiting semantic properties of the process structure to improve provenance exploitation • exploring process space across versions and structural similarities • graph matching	• semantic-based search of process space
execution	- enabling partial re-runs of resource-intensive workflows - storing very large provenance traces that accumulate over time - efficient query over large traces - presentation of query answers	- semantic-based query answering over annotated traces

The rest of this talk!

MANCHESTER 1824

The University of Manchester

...and their uses and associated challenges

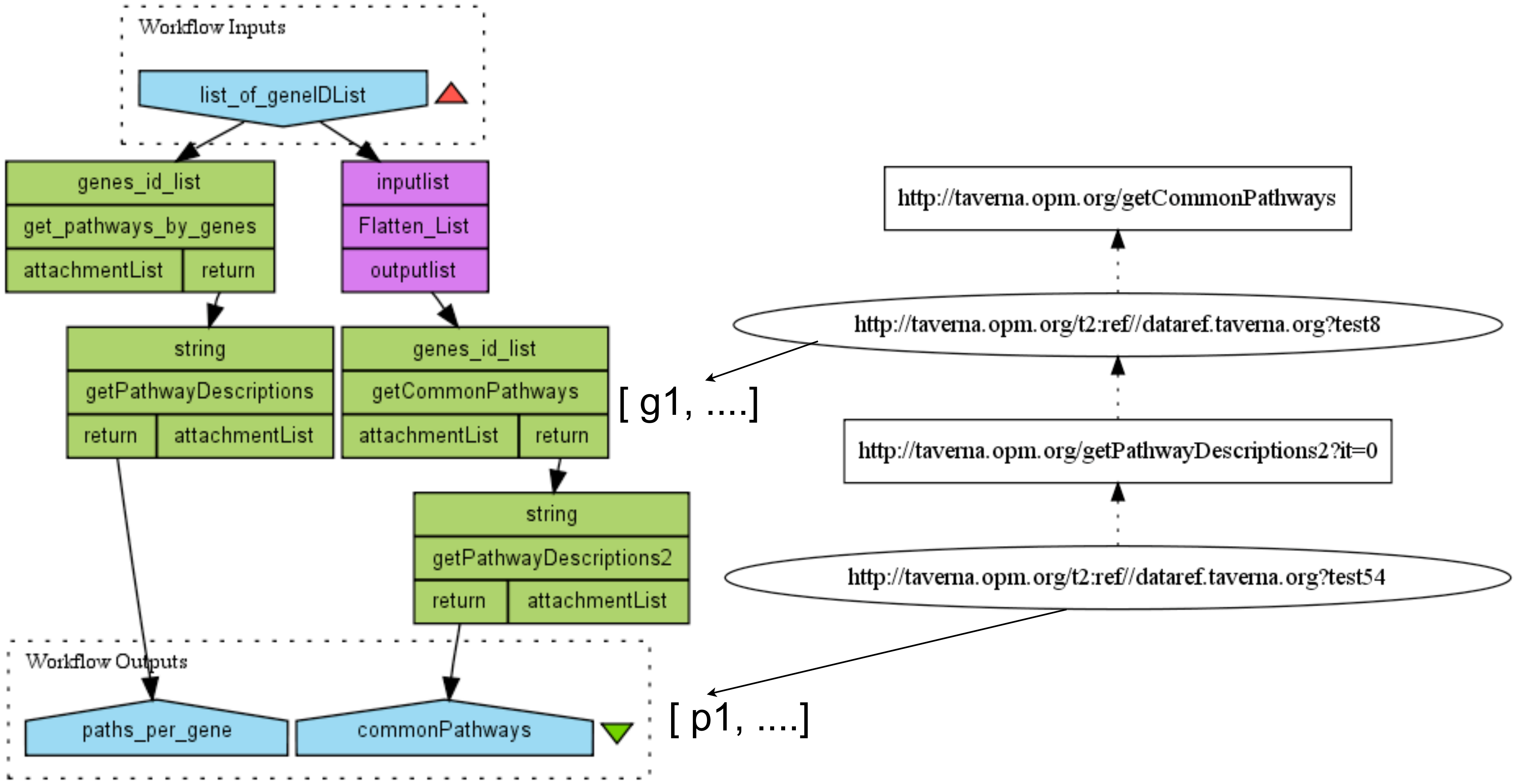
	raw provenance metadata	provenance metadata + interpretation framework
design	• exploiting semantic properties of the process structure to improve provenance exploitation • exploring process space across versions and structural similarities • graph matching	• semantic-based search of process space
execution	- enabling partial re-runs of resource-intensive workflows - storing very large provenance traces that accumulate over time - efficient query over large traces - presentation of query answers	- semantic-based query answering over annotated traces

The rest of this talk!

MANCHESTER 1824

Querying provenance traces

- Lineage queries involve traversing a *provenance graph* from bottom to top



MANCHESTER 1824

The University of Manchester

Naive provenance trace queries

– In most approaches, the originating process are not used for querying
– consequence: query requires provenance graph traversal
  - large traces → computationally complex
  - view materialization used in practice to get around the computational complexity

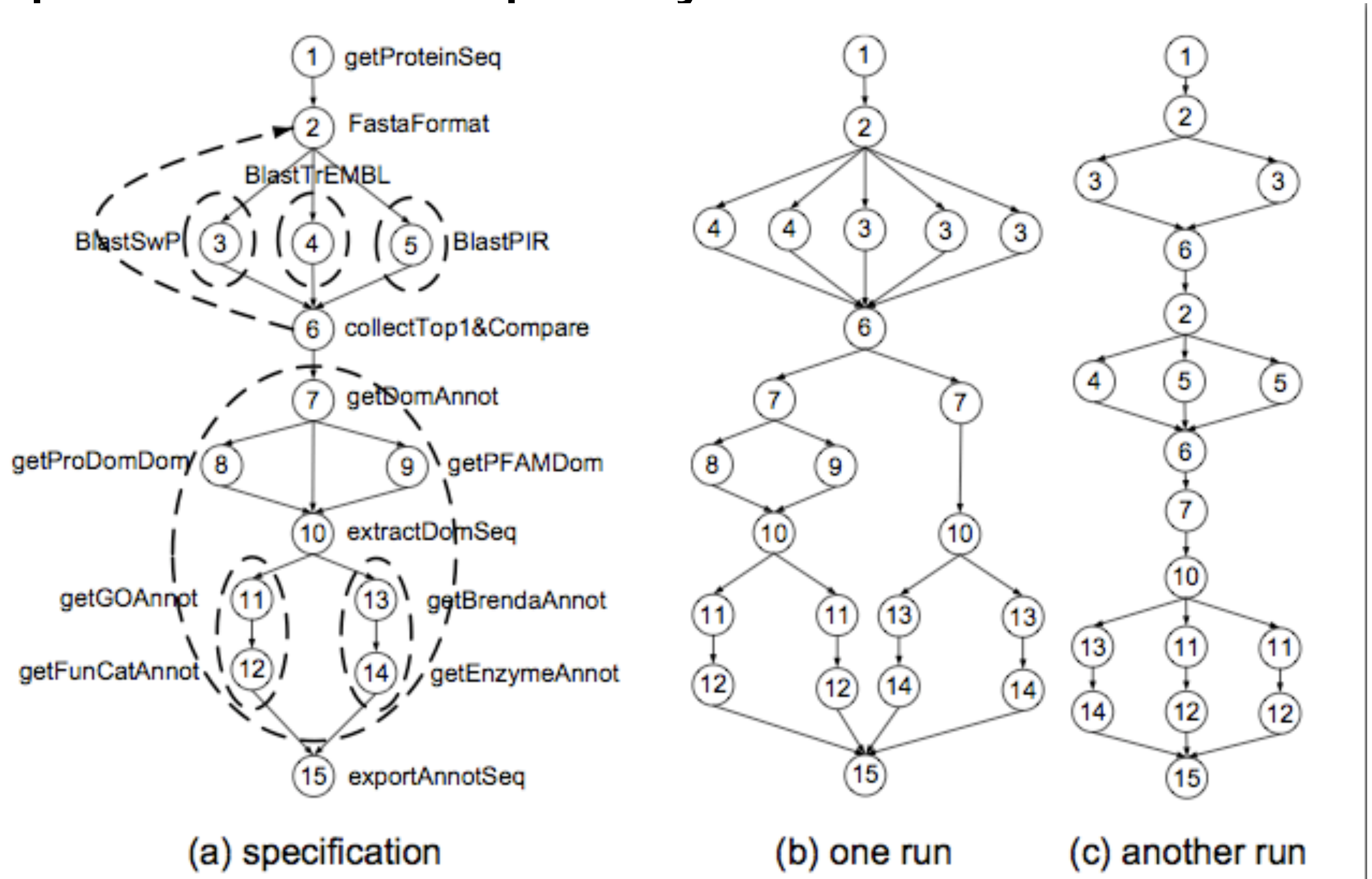
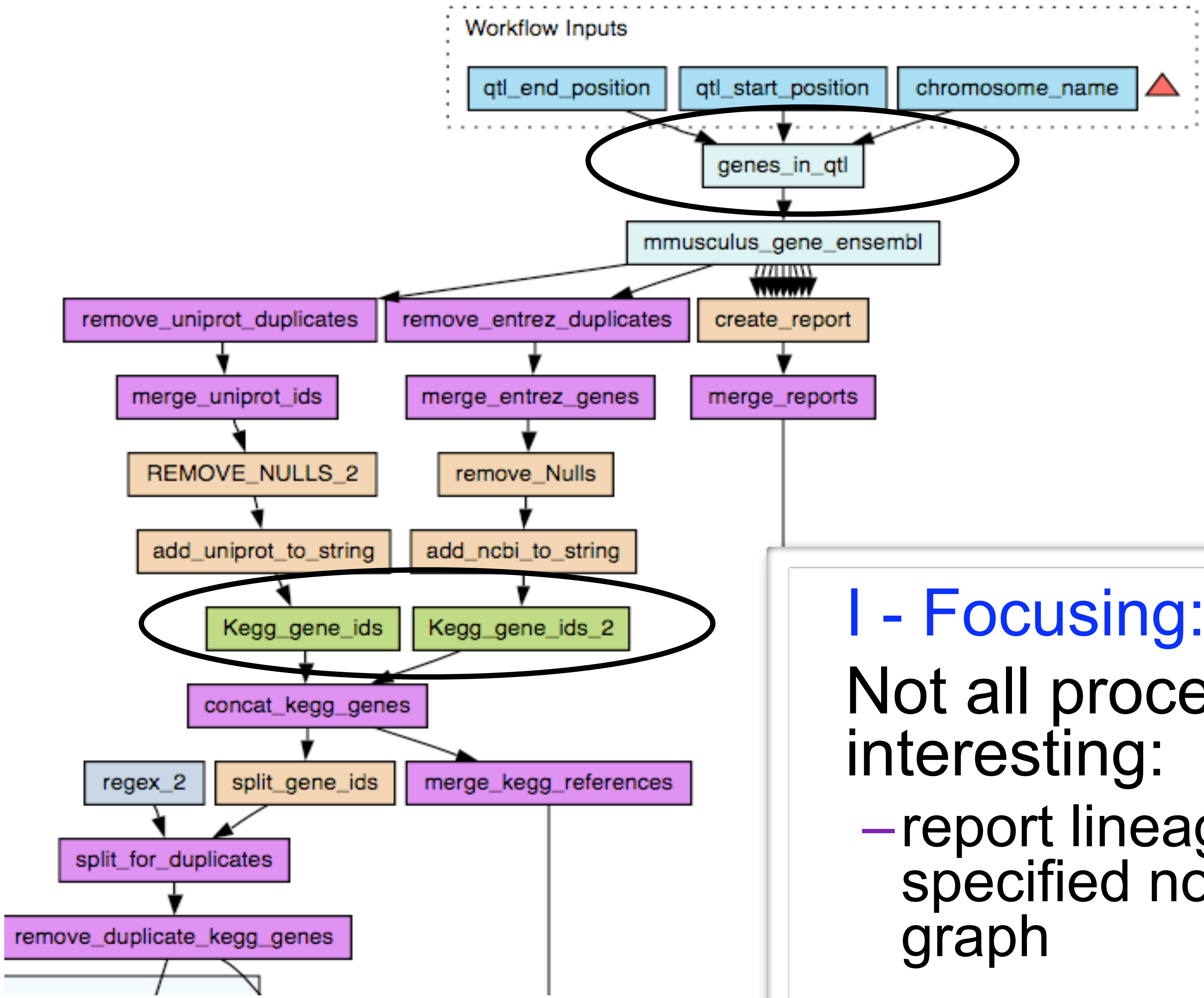


Fig. 1. Protein annotation workflow specification and runs

MANCHESTER 1824

The University of Manchester

Requirements for lineage queries - I



I - Focusing:

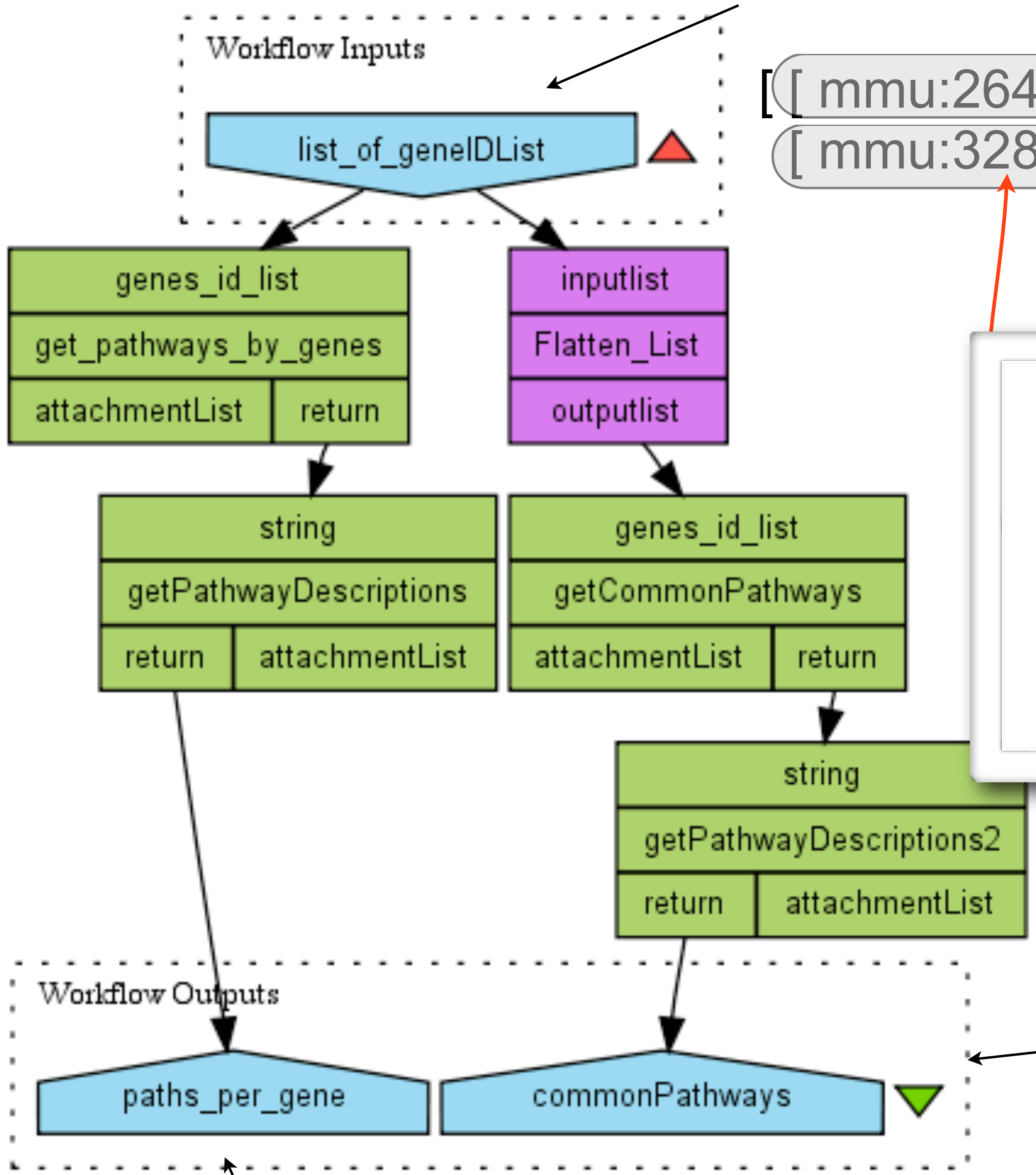
Not all processors are interesting:

– report lineage only at specified nodes in the graph

The University of Manchester

MANCHESTER 1824

Requirements for lineage queries - II



List-structured
KEGG gene ids:

[[mmu:26416],
[mmu:328788]]

II - Granularity:

Trace lineage for individual elements within collections

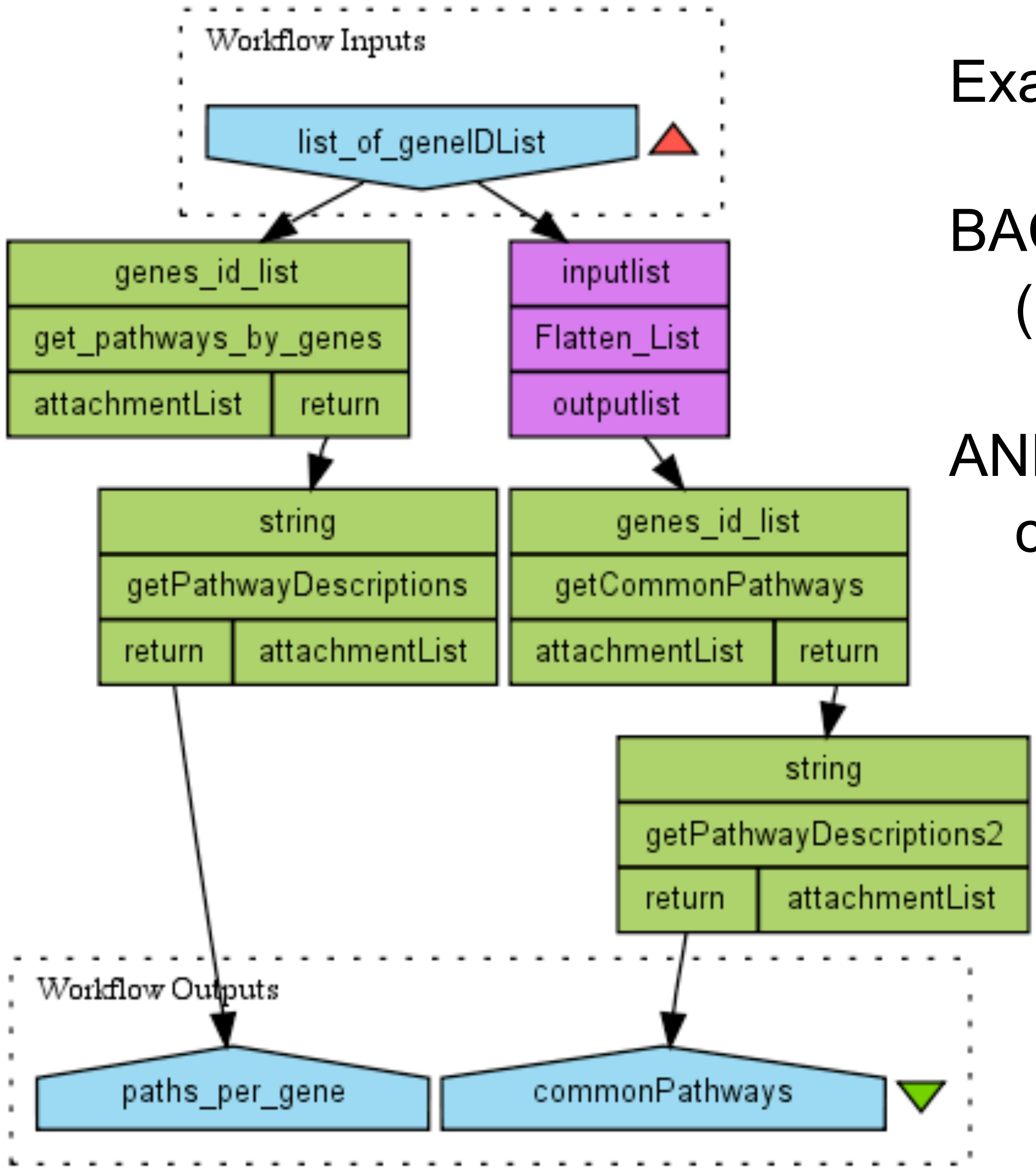
- when possible!

[path:mmu04010 MAPK signaling,
path:mmu04370 VEGF signaling]

[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
[path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

Requirements for lineage queries - III

- III - Answer queries efficiently without special auxiliary data structures
- (and, please provide declarative query specification)



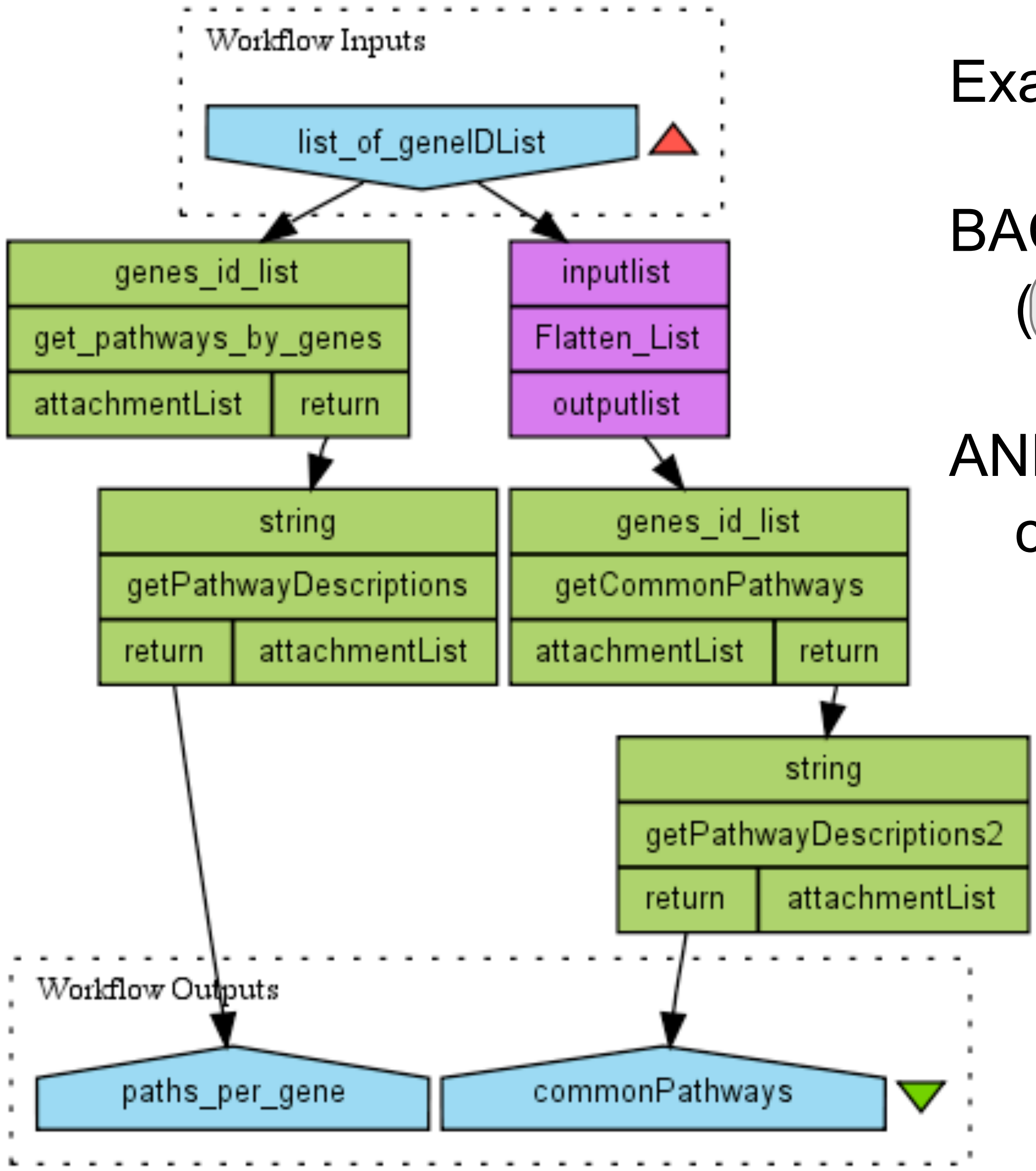
Example:

```
BACKTRACE
    (paths_per_gene[3,4],  paths_per_gene[1,2])
        AT get_pathway_by_genes
AND
    commonPathways[1]
        AT TOP
```

[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
[path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

Requirements for lineage queries - III

- III - Answer queries efficiently without special auxiliary data structures
- (and, please provide declarative query specification)



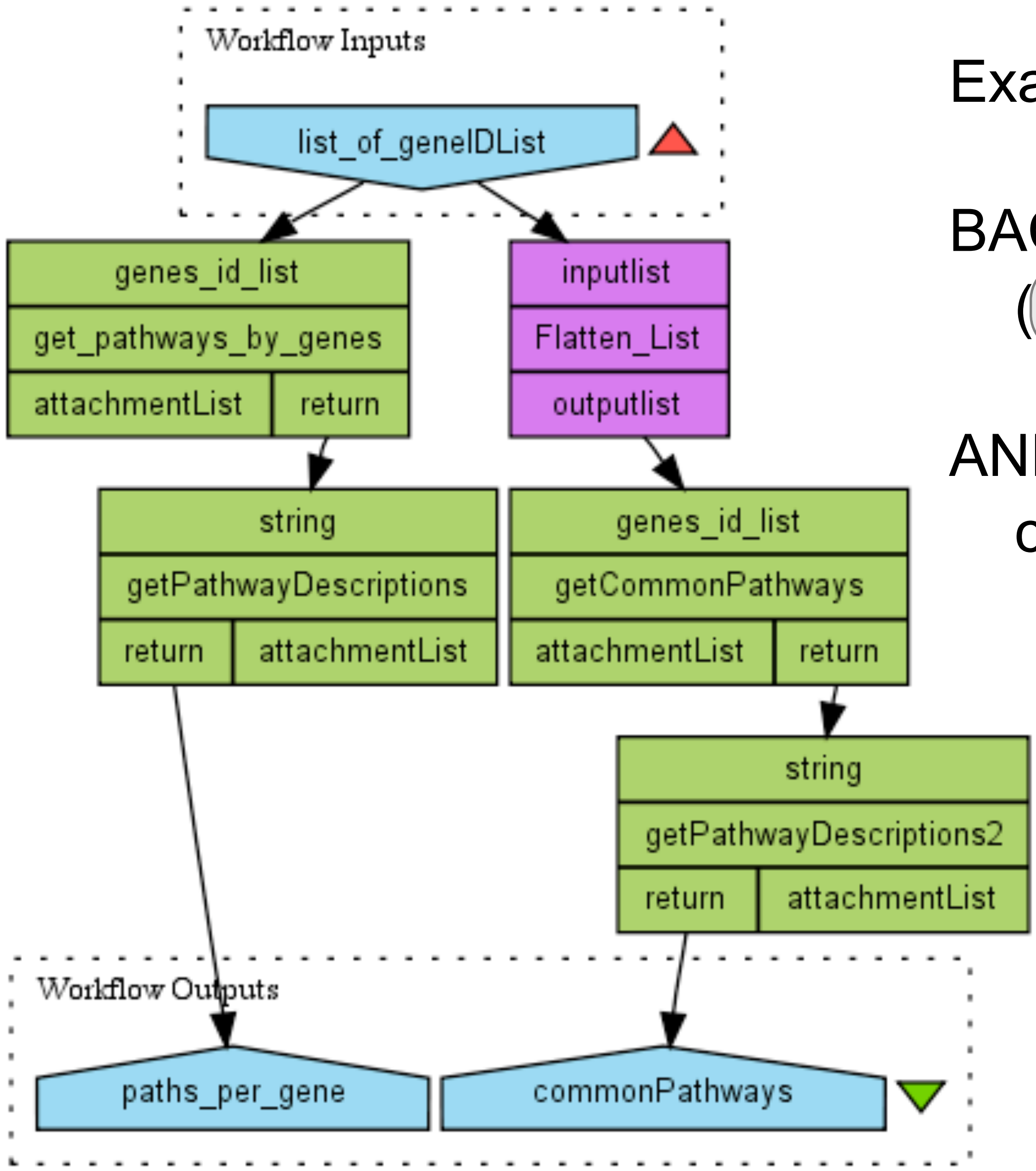
Example:

```
BACKTRACE
    (paths_per_gene[3,4],  paths_per_gene[1,2])
        AT get_pathway_by_genes
AND
    commonPathways[1]
        AT TOP
```

[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
  [path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

Requirements for lineage queries - III

- III - Answer queries efficiently without special auxiliary data structures
- (and, please provide declarative query specification)



Example:

targets

BACKTRACE
(paths_per_gene[3,4], paths_per_gene[1,2])
AT get_pathway_by_genes
AND
commonPathways[1]
AT TOP

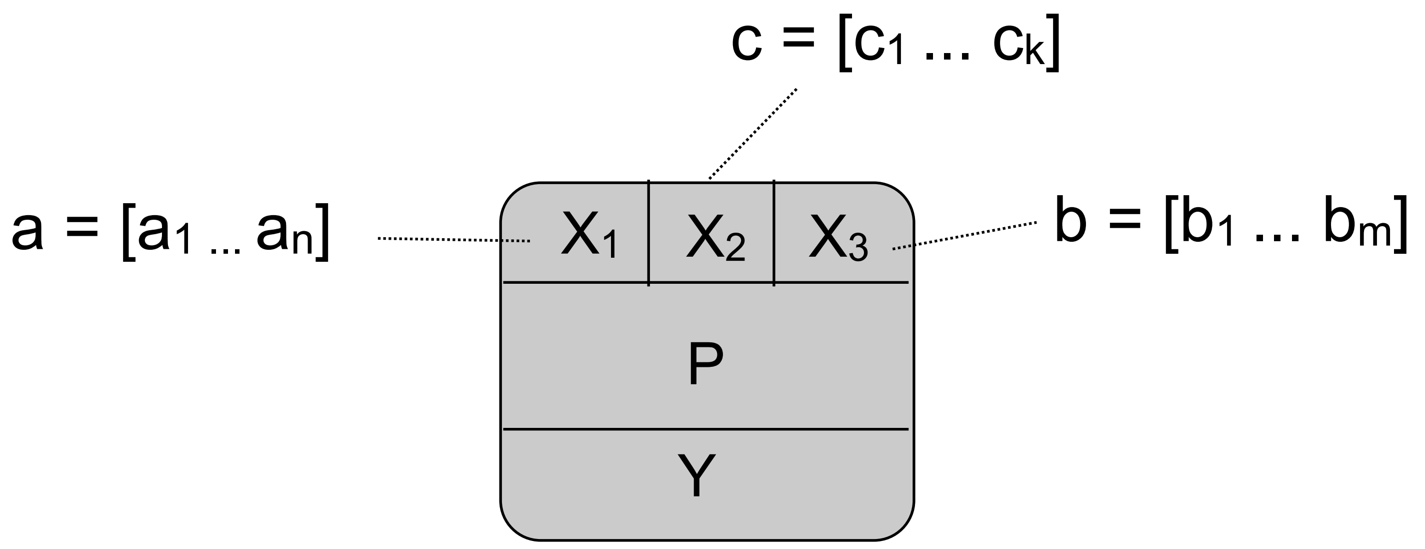
selected processors

[[path:mmu04210 Apoptosis, path:mmu04010 MAPK signaling, ...],
[path:mmu04010 MAPK signaling , path:mmu04620 Toll-like receptor, ...]]

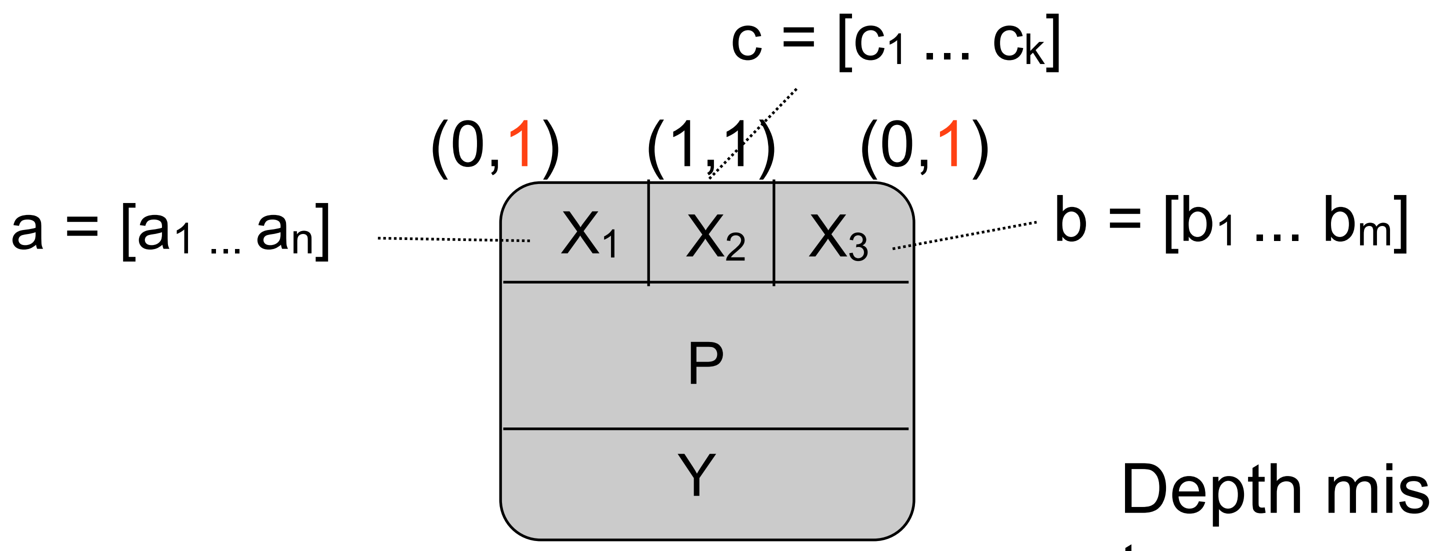
MANCHESTER 1824

The University of Manchester

How: Implicit iteration in Taverna



How: Implicit iteration in Taverna



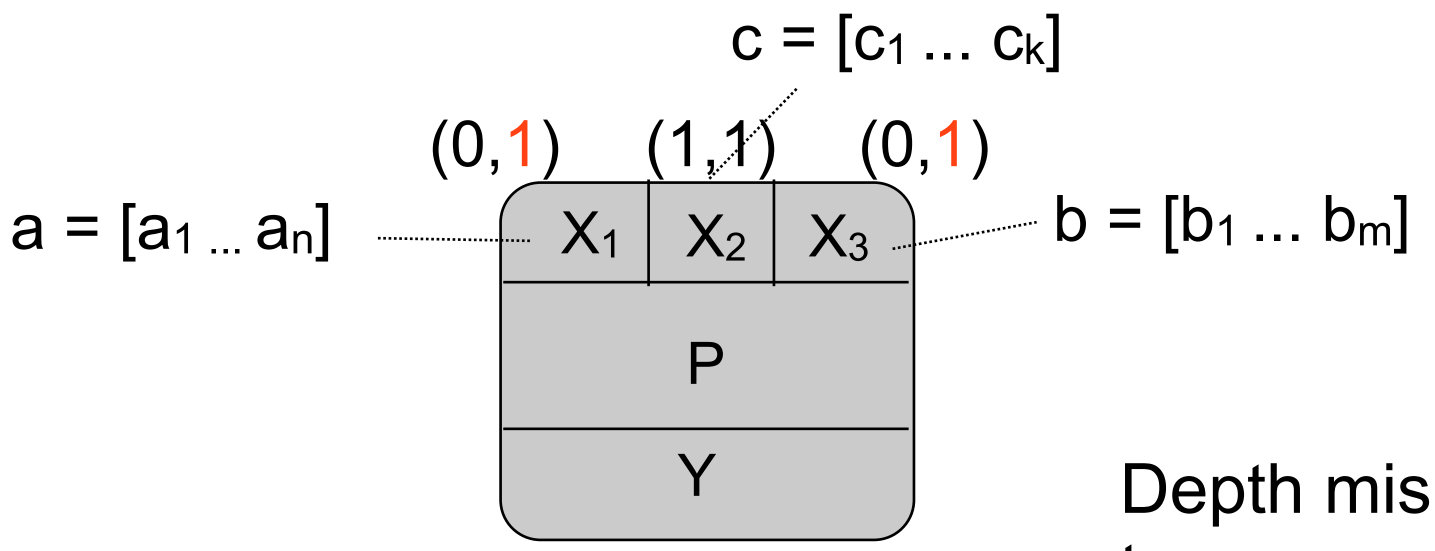
Depth mismatch between declared / offered type:

depth(P:X1) = 0 but depth(a) = 1

depth(P:X2) = depth(c) = 1

depth(P:X3) = 1 but depth(c) = 1

How: Implicit iteration in Taverna



Depth mismatch between declared / offered type:

depth(P:X1) = 0 but depth(a) = 1

depth(P:X2) = depth(c) = 1

depth(P:X3) = 1 but depth(c) = 1

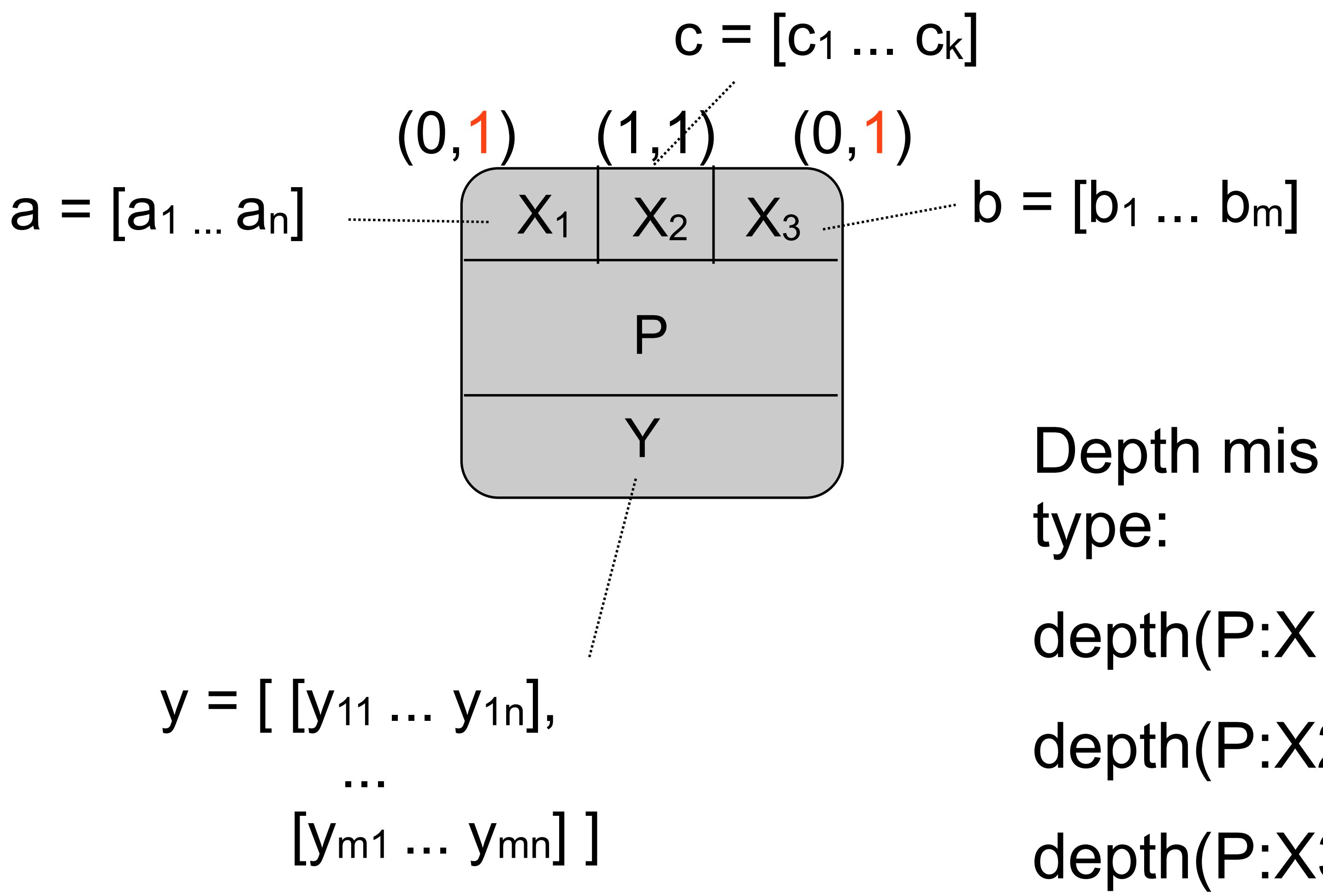
How *y* is computed at P:

let $l = a \otimes b = [\,[\,\langle a_i, b_j\rangle \mid b_j \in b\,] \mid a_i \in a\,]$ // cross product

$l' = [\,[\,\langle a_i, c, b_j\rangle \mid b_j \in b\,] \mid a_i \in a\,]$ // same product but with c interleaved

y = (**map** (**map** P) l') = $[(\text{map } P\, [\,\langle a_1,c, b_1\rangle \ldots \langle a_1,c, b_m\rangle]), \ldots,$
$(\text{map } P\, [\,\langle a_n,c, b_1\rangle \ldots \langle a_n,c, b_m\rangle])\,] =$
$[\,[y_{11} \ldots y_{1n}], \ldots [y_{n1} \ldots y_{nm}]\,]$

How: Implicit iteration in Taverna



Depth mismatch between declared / offered type:

depth(P:X1) = 0 but depth(a) = 1

depth(P:X2) = depth(c) = 1

depth(P:X3) = 1 but depth(c) = 1

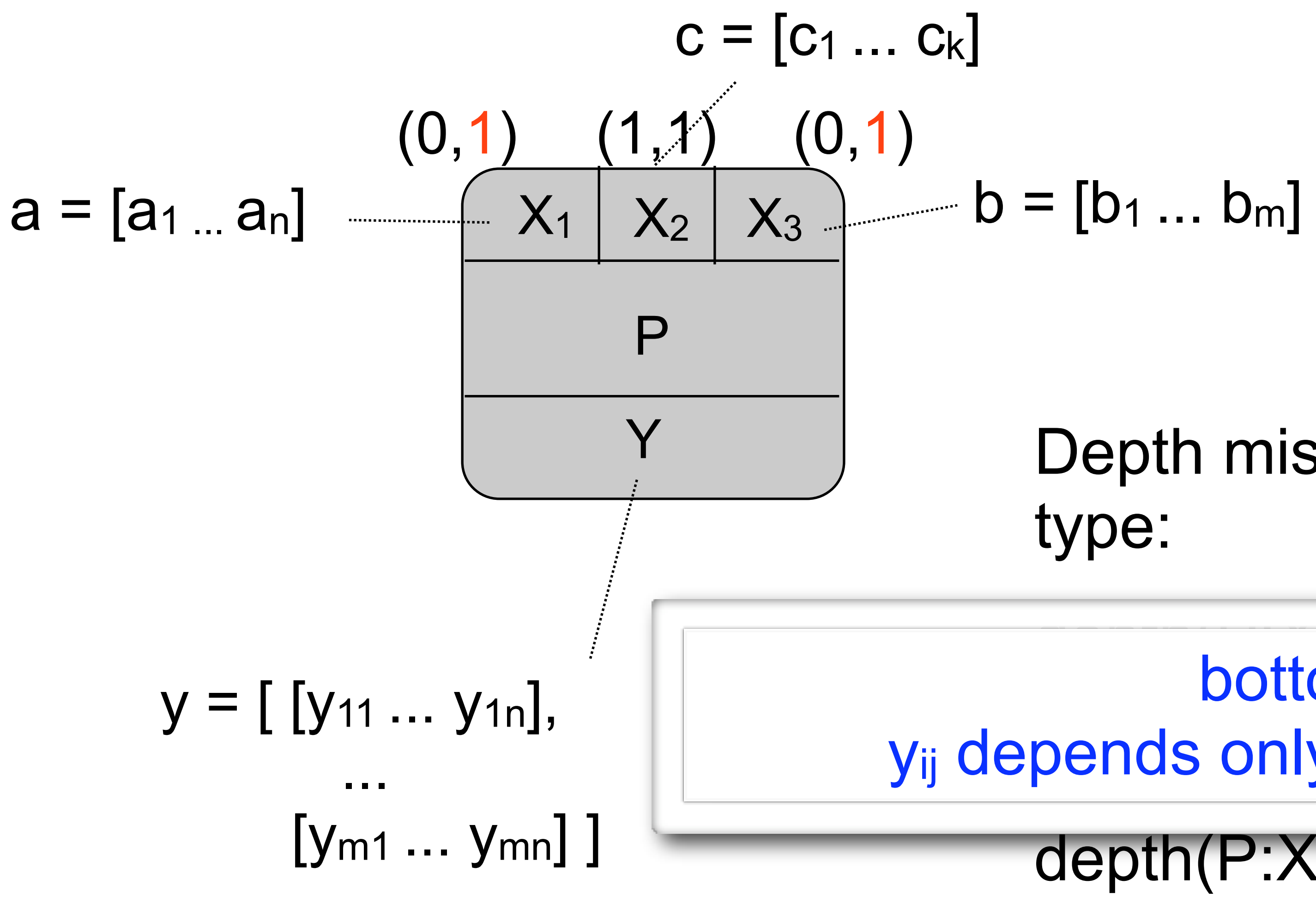
How *y* is computed at P:

let $l = a \otimes b = [\,[\,<a_i, b_j> \mid b_j \in b\,] \mid a_i \in a\,]$ // cross product

$l' = [\,[\,<a_i, c, b_j> \mid b_j \in b\,] \mid a_i \in a\,]$ // same product but with c interleaved

y = (**map** (**map** P) l') = $[(\text{map } P\, [\,<a_1,c, b_1> \dots <a_1,c, b_m>]), \dots,$
$(\text{map } P\, [\,<a_n,c, b_1> \dots <a_n,c, b_m>])\,] =$
$[\,[y_{11} \dots y_{1n}], \dots [y_{n1} \dots y_{nm}]\,]$

How: Implicit iteration in Taverna



Depth mismatch between declared / offered type:

bottom line:
y_{ij} depends only on values a_i, c, b_j

depth(P:X3) = 1 but depth(c) = 1

How *y* is computed at P:

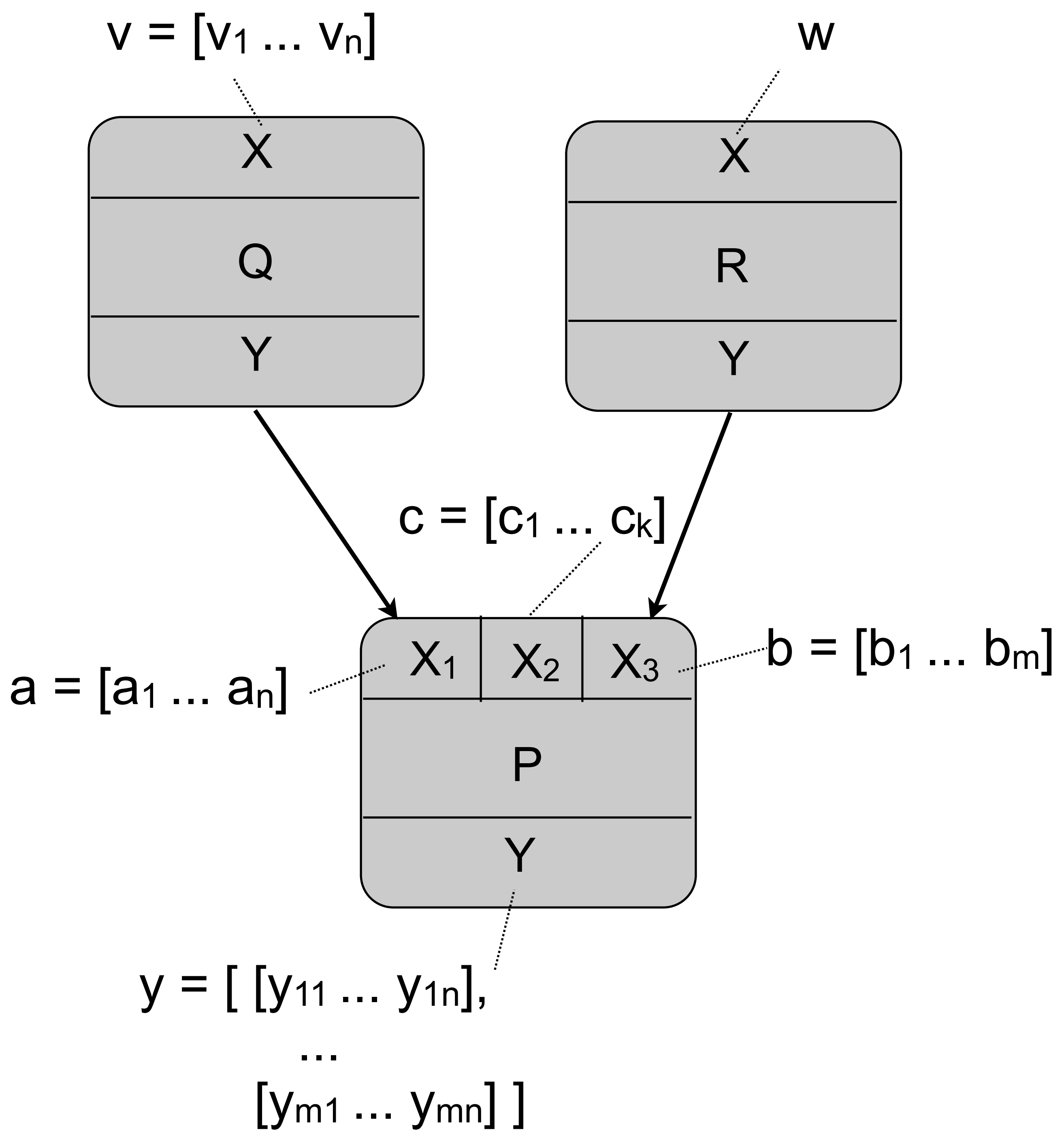
let $l = a \otimes b = [[<a_i, b_j> \mid b_j \in b] \mid a_i \in a]$ // cross product

$l' = [[<a_i, c, b_j> \mid b_j \in b] \mid a_i \in a]$ // same product but with c interleaved

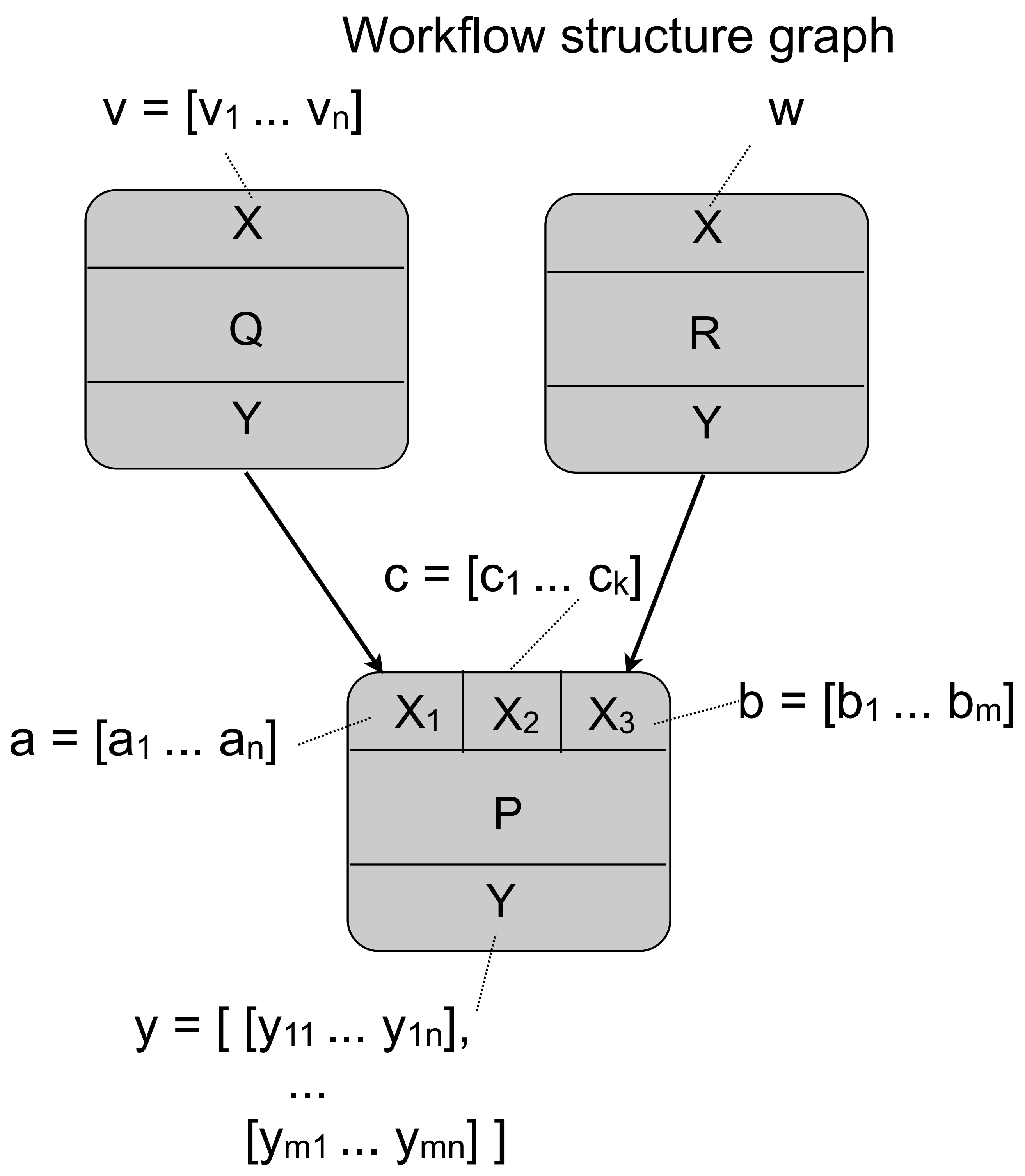
y = (**map** (**map** P) l') = $[(\text{map } P [<a_1,c, b_1> \dots <a_1,c, b_m>]), \dots,$
$(\text{map } P [<a_n,c, b_1> \dots <a_n,c, b_m>])] =$
$[[y_{11} \dots y_{1n}], \dots [y_{n1} \dots y_{nm}]]$

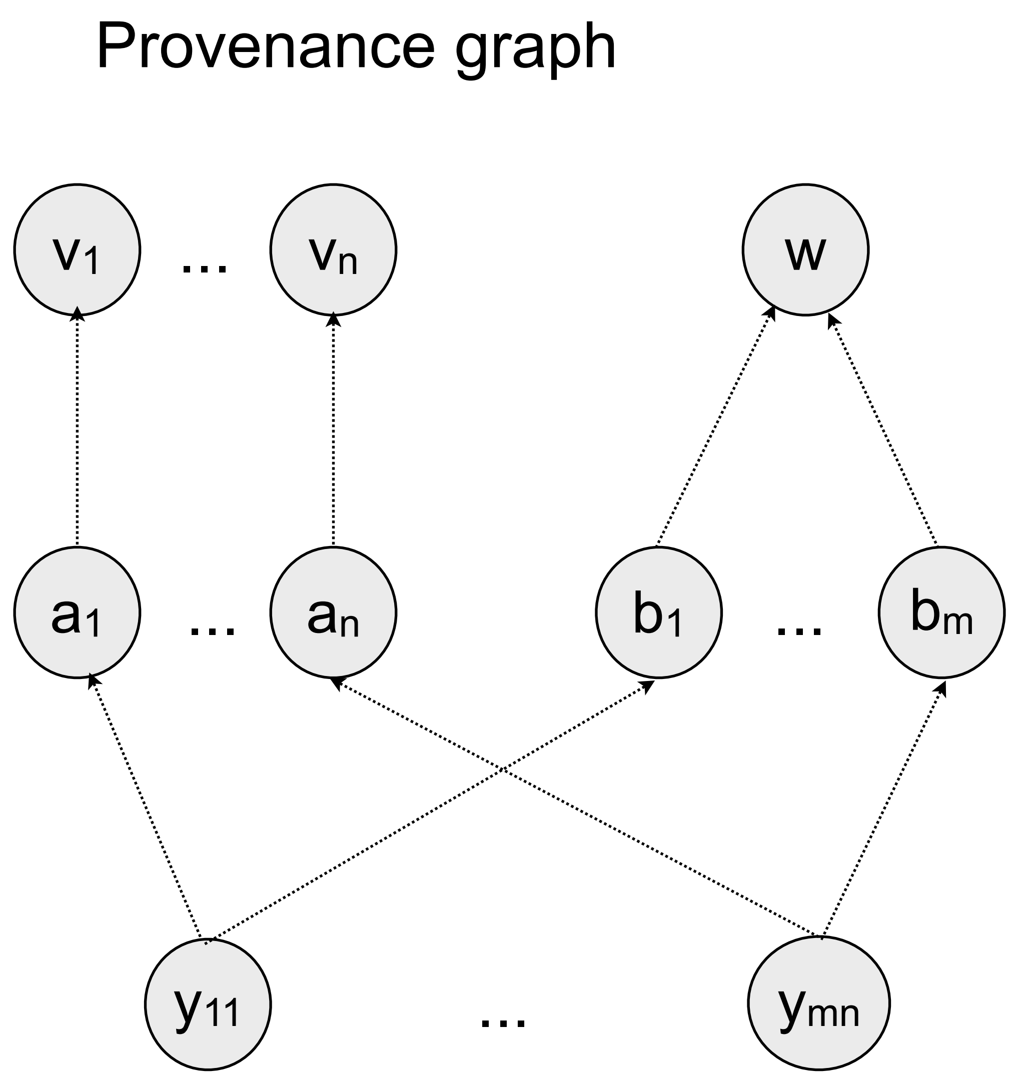
Extensional vs intensional tracing

Workflow structure graph

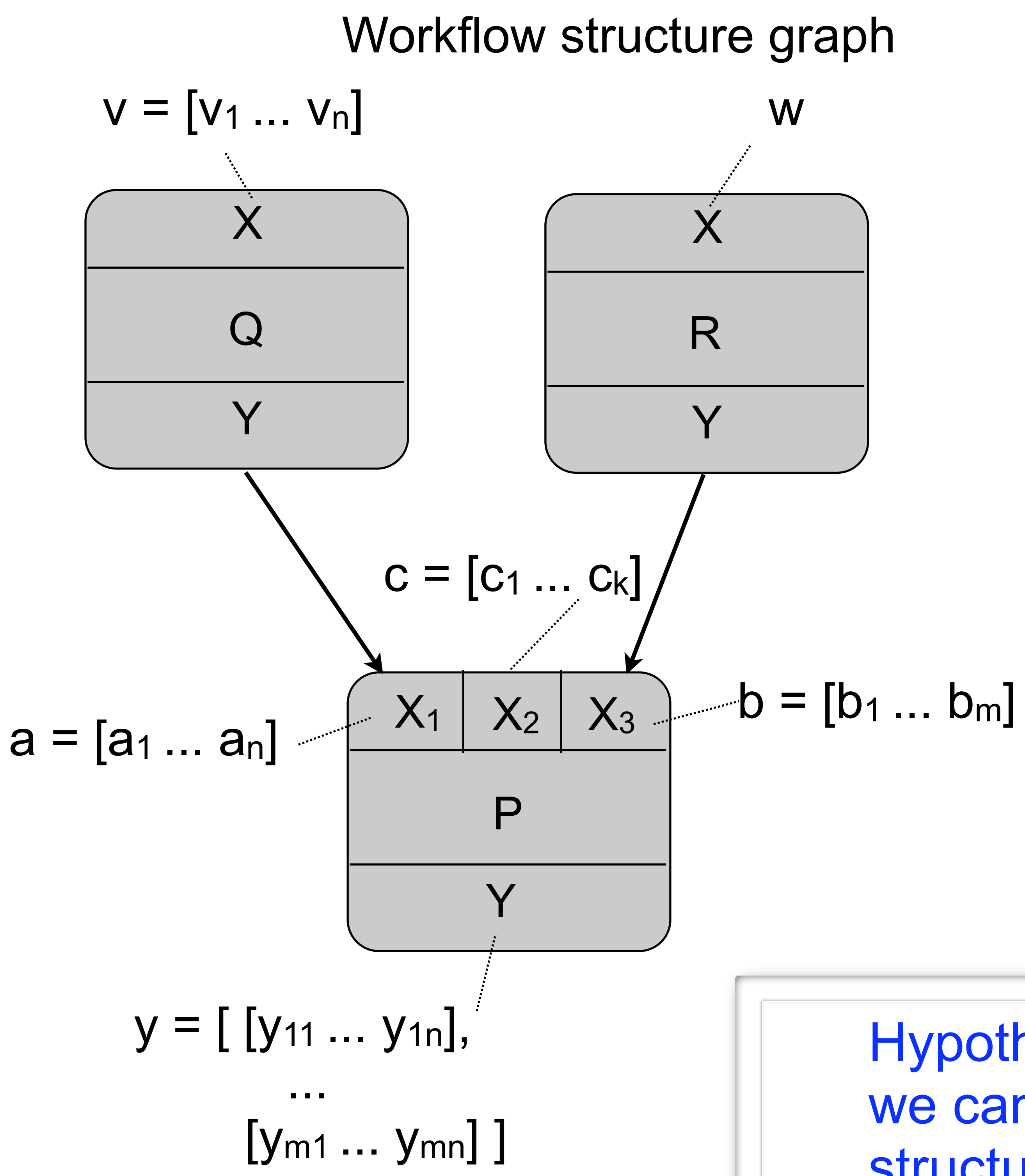


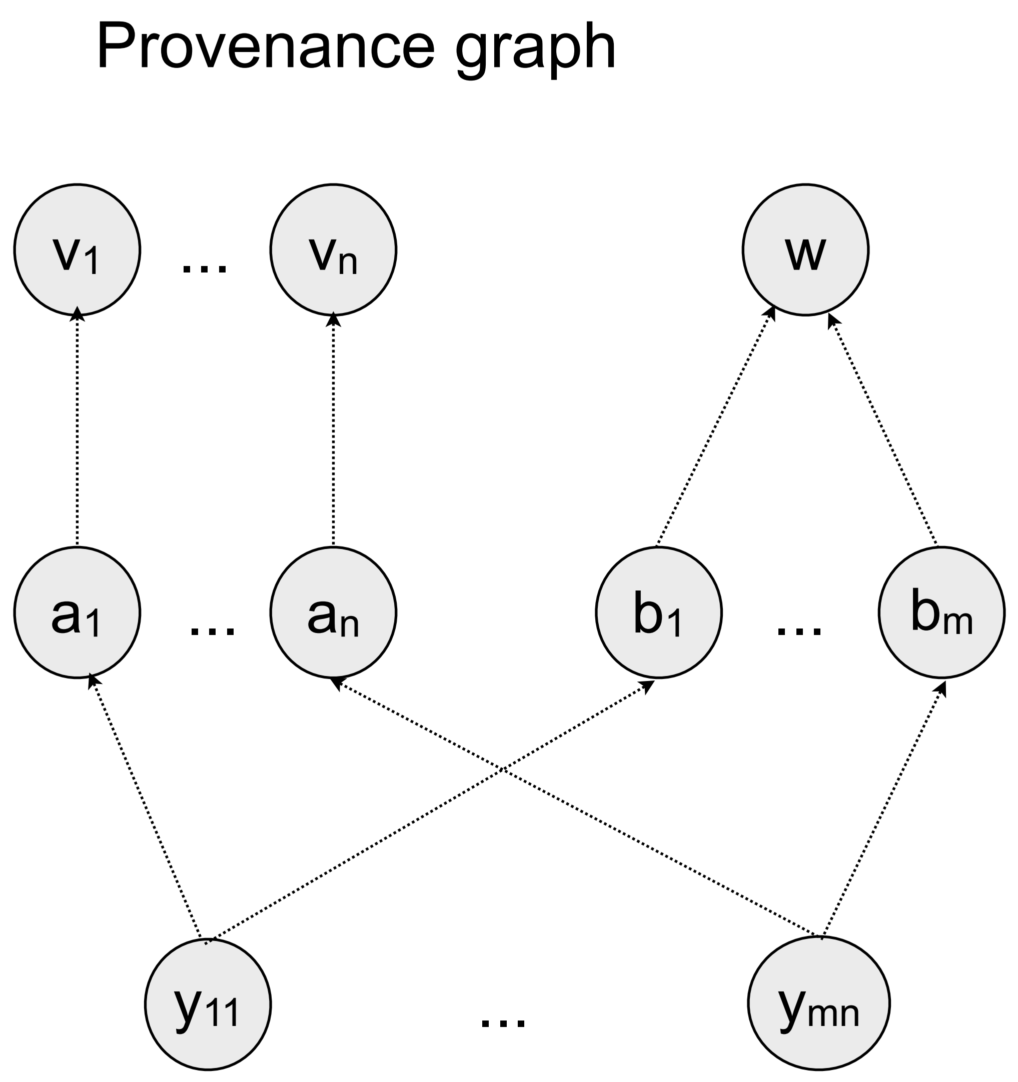
Extensional vs intensional tracing





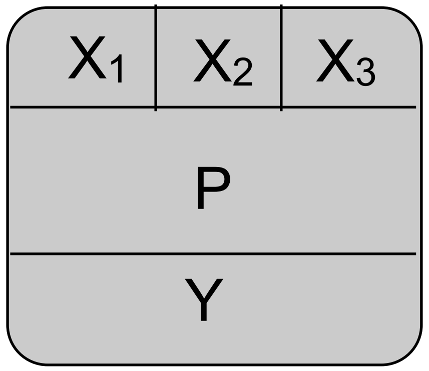
Extensional vs intensional tracing





Hypothesis:
we can exploit the static workflow graph structure to avoid explicitly traversing the entire trace to answer a query

The path projection rule



1) In general the actual depth at the output is:

depth(y) = depth(Y) + Σ $\delta(X_i = x_i)$

where $\delta(X_i = x_i)$ = depth(x_i) - depth(X_i)

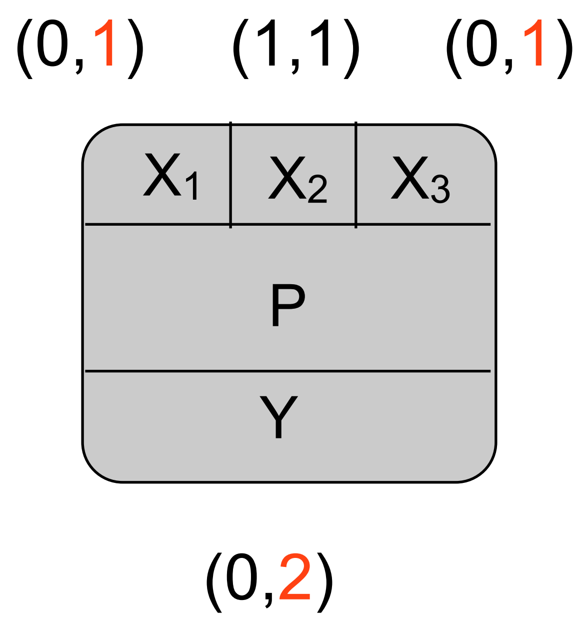
Therefore:
$\delta(X=x)$ can be computed statically on the workflow graph structure,

- given the declared depth(X)
- using a simple propagation algorithm

Y[i.j] $\rightarrow$ X1[i], X2[], X3[j]

$[i_1 \,.\, i_2 \,.\, \ldots \,.\, i_k]$ = ——————————————————

The path projection rule



1) In general the actual depth at the output is:

depth(y) = depth(Y) + Σ $\delta(X_i = x_i)$

where $\delta(X_i = x_i)$ = depth(x_i) - depth(X_i)

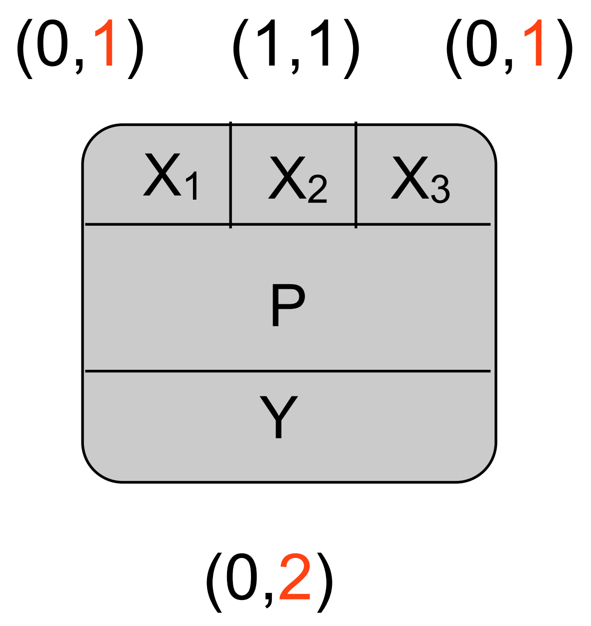
$Y[i.j] \rightarrow X1[i], X2[], X3[j]$

Therefore:
$\delta(X=x)$ can be computed statically on the workflow graph structure,

- given the declared depth(X)
- using a simple propagation algorithm

$[i_1 \,.\, i_2 \,.\, \ldots \,.\, i_k]$ = ————————————————————

The path projection rule



1) In general the actual depth at the output is:

$$\text{depth}(y) = \text{depth}(Y) + \Sigma\, \delta(X_i = x_i)$$

where $\delta(X_i = x_i) = \text{depth}(x_i) - \text{depth}(X_i)$

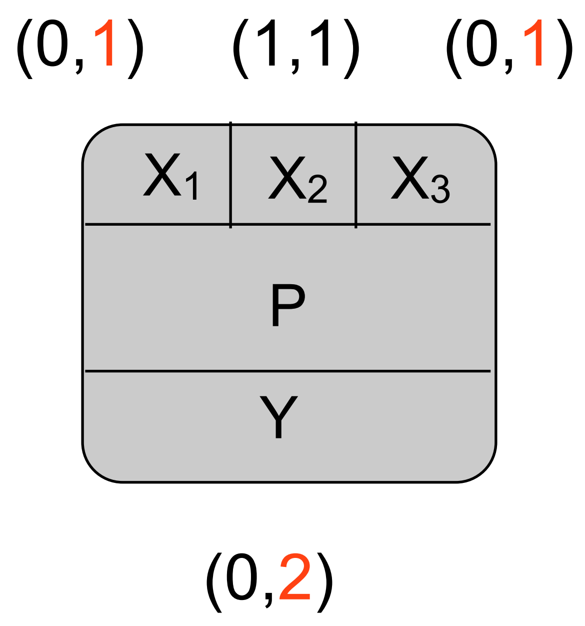
Therefore:
$\delta(X=x)$ can be computed statically on the workflow graph structure,

- given the declared depth(X)
- using a simple propagation algorithm

$Y[i.j] \rightarrow X1[i], X2[], X3[j]$

$$[i_1 \,.\, i_2 \,.\, \ldots \,.\, i_k] = \frac{\delta(X_1 = x_1)}{\qquad\qquad\qquad\qquad\qquad}$$

The path projection rule



1) In general the actual depth at the output is:

$$depth(y) = depth(Y) + \Sigma\, \delta(X_i = x_i)$$

where $\delta(X_i = x_i) = depth(x_i) - depth(X_i)$

Therefore:
$\delta(X=x)$ can be computed statically on the workflow graph structure,

- given the declared depth(X)
- using a simple propagation algorithm

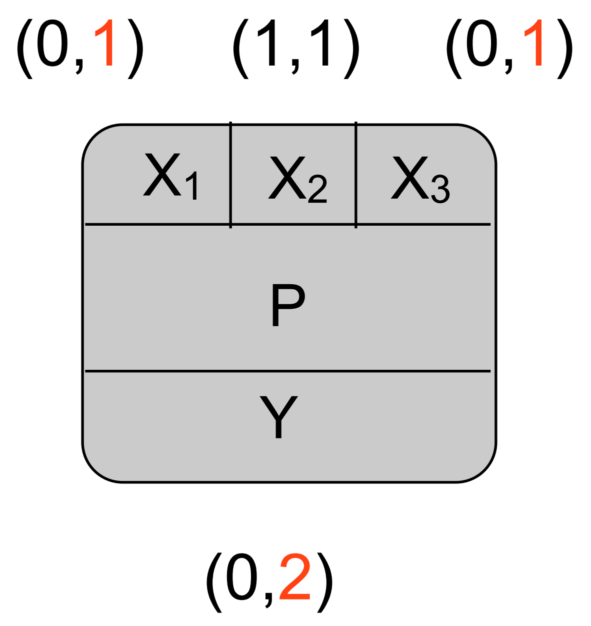
$Y[i.j] \rightarrow X1[i], X2[], X3[j]$

$[i_1 . i_2 . \ldots . i_k] =$

$\overline{}$

X_1

The path projection rule



1) In general the actual depth at the output is:

depth(y) = depth(Y) + Σ $\delta(X_i = x_i)$

where $\delta(X_i = x_i)$ = depth(x_i) - depth(X_i)

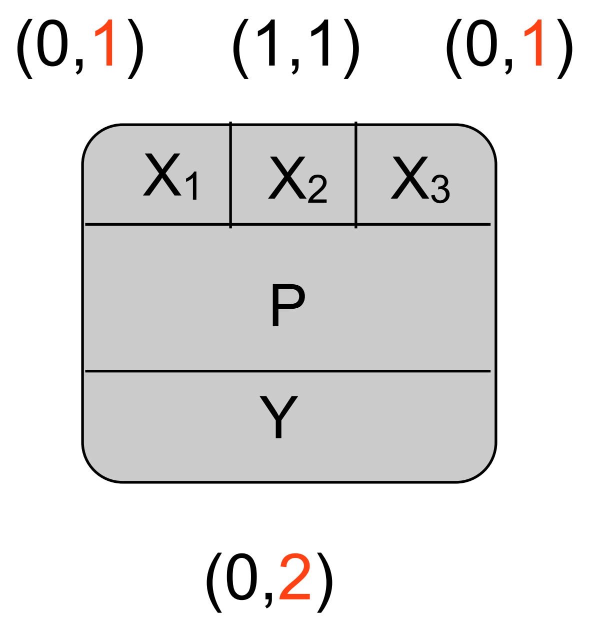
Therefore:
$\delta(X=x)$ can be computed statically on the workflow graph structure,
- given the declared depth(X)
- using a simple propagation algorithm

$Y[i.j] \rightarrow X1[i], X2[], X3[j]$

$[i_1 . i_2 . \dots . i_k] =$ $\underline{\hspace{3em}}$ $\underline{\delta(X_2 = x_2)\hspace{10em}}$

$\overline{\hspace{2em}X_1\hspace{2em}}$

The path projection rule



1) In general the actual depth at the output is:

$$depth(y) = depth(Y) + \Sigma\ \delta(X_i = x_i)$$

where $\delta(X_i = x_i) = depth(x_i) - depth(X_i)$

Therefore:
$\delta(X=x)$ can be computed statically on the workflow graph structure,

- given the declared depth(X)
- using a simple propagation algorithm

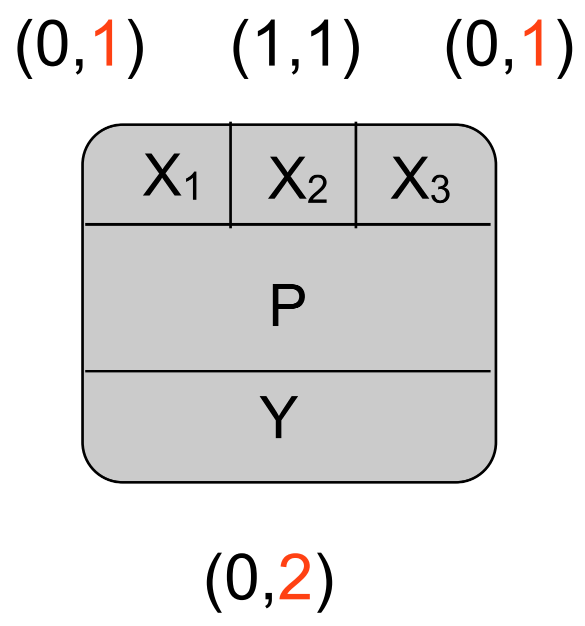
$Y[i.j] \rightarrow X1[i], X2[], X3[j]$

$[i_1 . i_2 . \ldots . i_k] =$

$$\underline{\qquad\qquad} \quad \underline{\qquad\qquad}$$

X_1 X_2

The path projection rule



$Y[i.j] \rightarrow X1[i], X2[], X3[j]$

1) In general the actual depth at the output is:

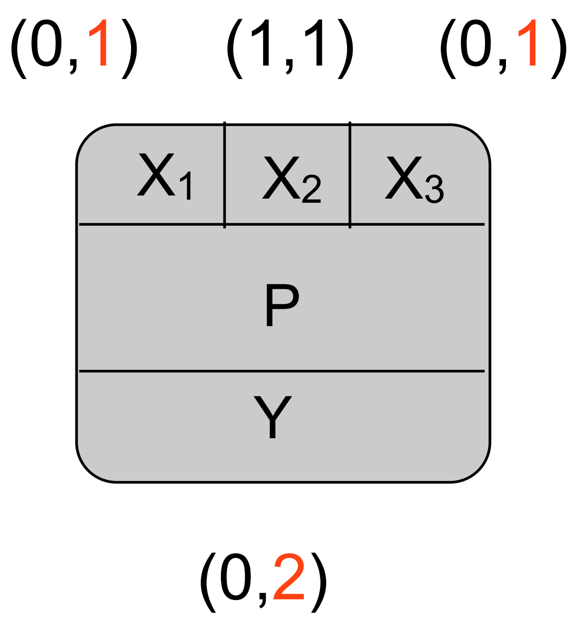
depth(y) = depth(Y) + $\Sigma\ \delta(X_i = x_i)$

where $\delta(X_i = x_i)$ = depth(x_i) - depth(X_i)

Therefore:
$\delta(X=x)$ can be computed statically on the workflow graph structure,
- given the declared depth(X)
- using a simple propagation algorithm

$[i_1 . i_2 . \dots . i_k] = \underbrace{\qquad}_{X_1} \quad \underbrace{\qquad}_{X_2} \quad \underbrace{\qquad}_{\delta(X_k = x_k)}$

The path projection rule



1) In general the actual depth at the output is:

$$\text{depth}(y) = \text{depth}(Y) + \Sigma\, \delta(X_i = x_i)$$

where $\delta(X_i = x_i) = \text{depth}(x_i) - \text{depth}(X_i)$

Therefore:
$\delta(X=x)$ can be computed statically on the workflow graph structure,

- given the declared depth(X)
- using a simple propagation algorithm

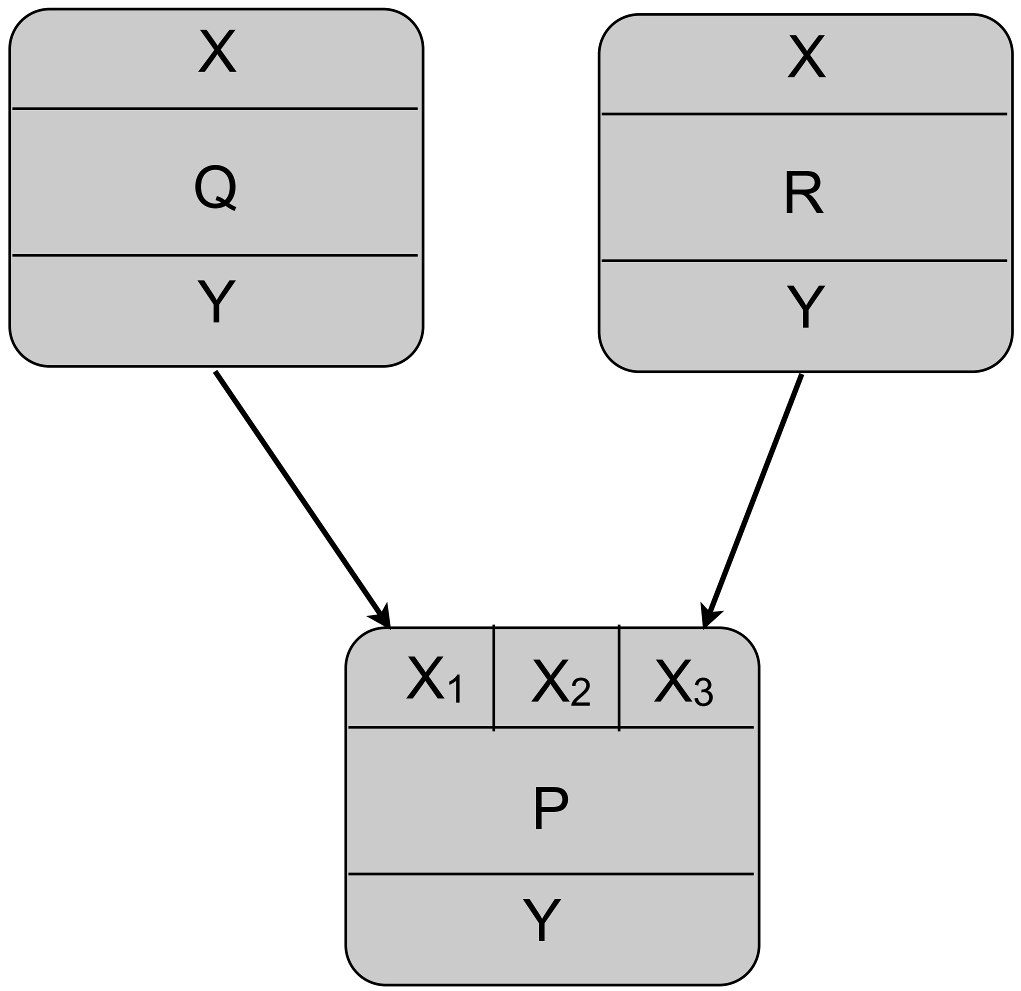
$Y[i.j] \rightarrow X1[i], X2[], X3[j]$

$[i_1 \,.\, i_2 \,.\, \ldots \,.\, i_k] =$

$\underline{\hspace{2cm}}$ $\underline{\hspace{2cm}}$ $\underline{\hspace{2cm}}$

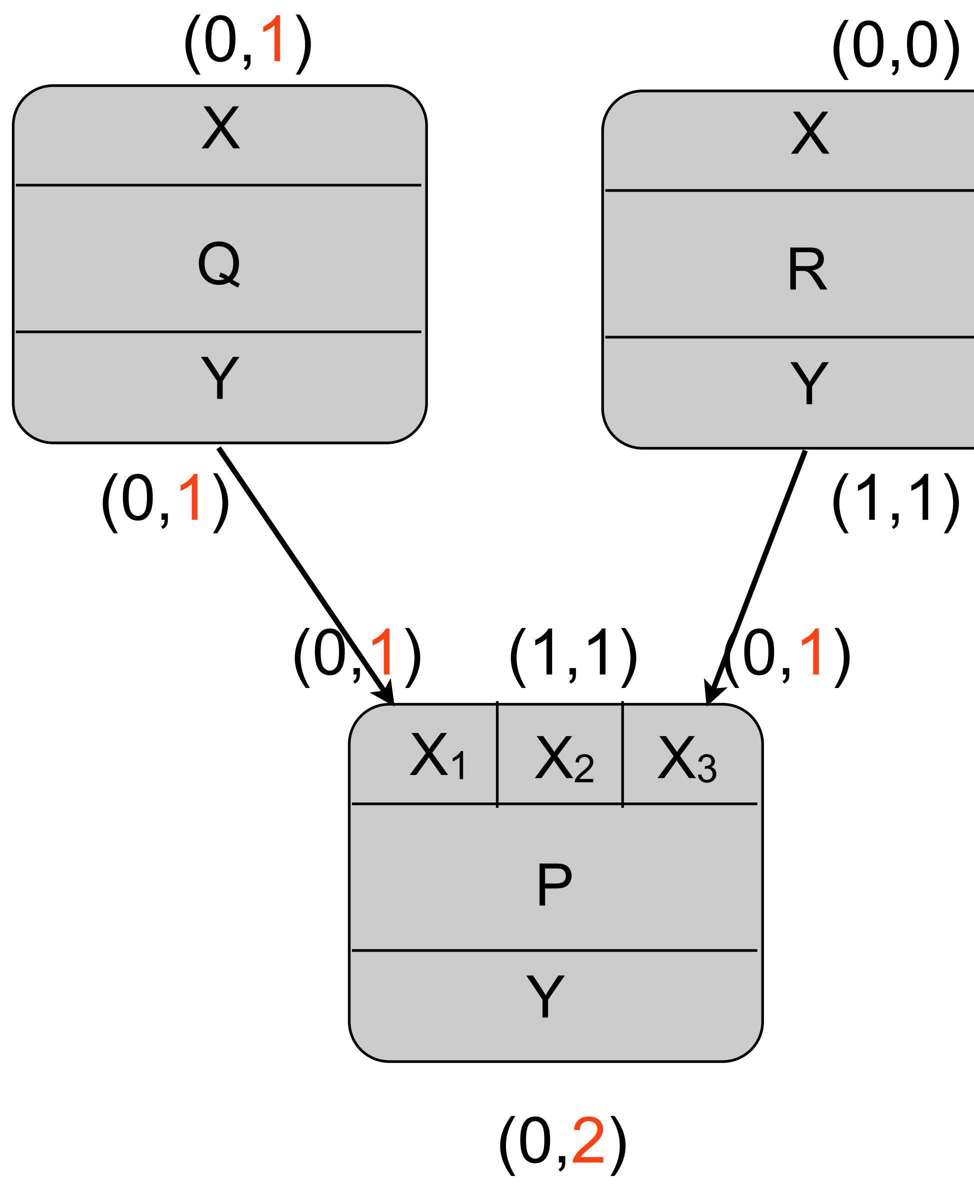
X_1 X_2 X_k

Extension to the entire workflow graph



The University of Manchester

Extension to the entire workflow graph

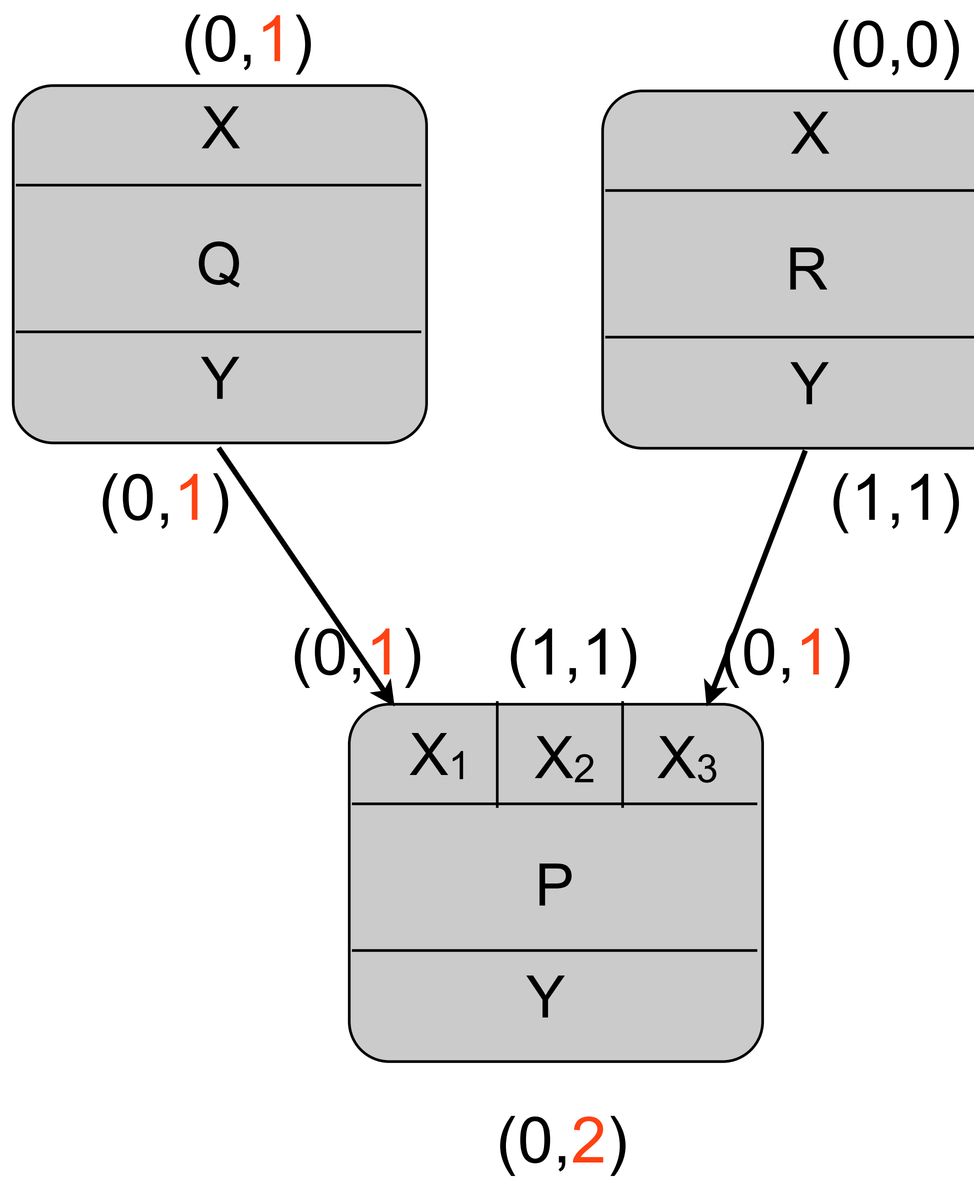


Y = [[...], ... [...]]

MANCHESTER 1824

The University of Manchester

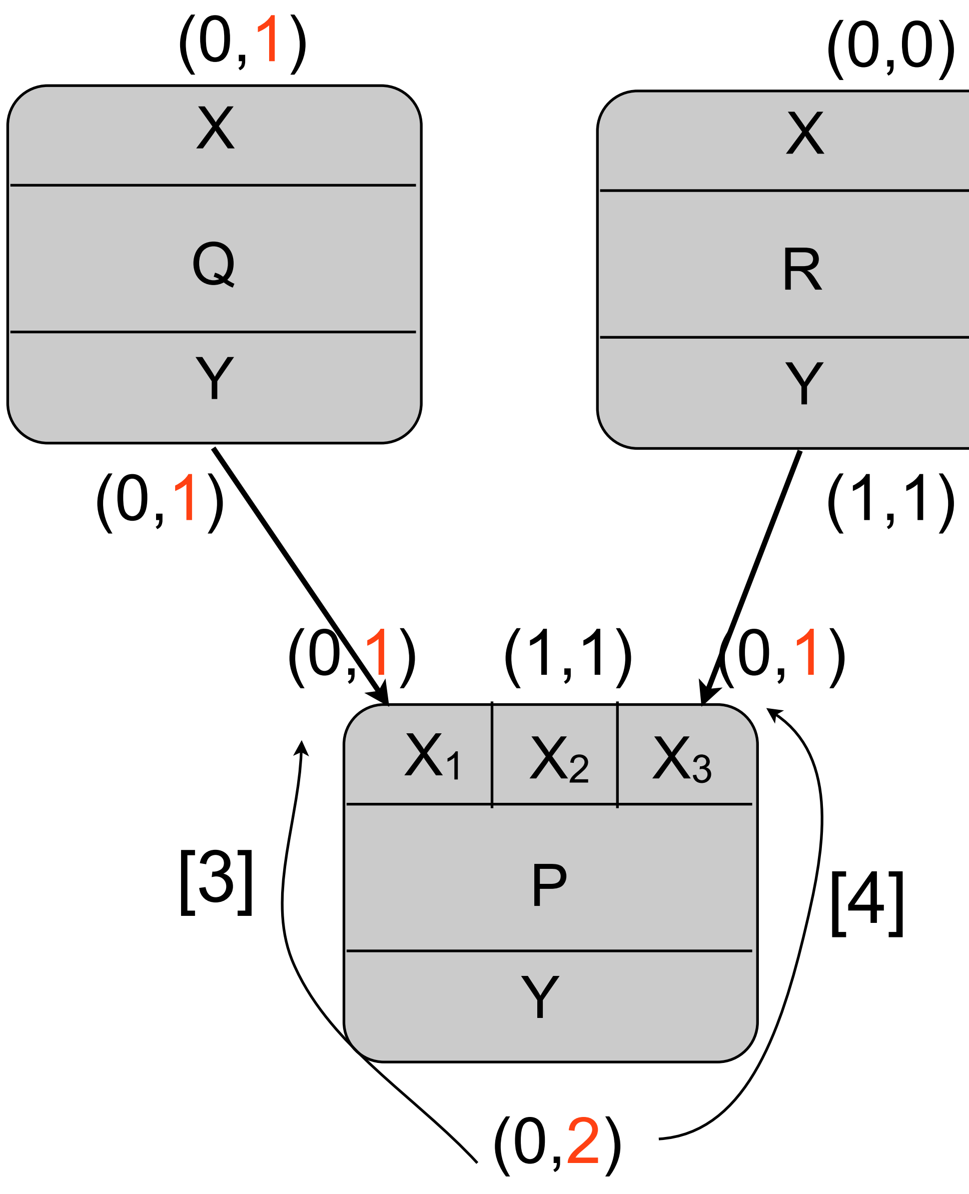
Extension to the entire workflow graph



Y = [[...], ... [...]]

lineage(P:Y[3,4]) → lineage(P:X1[3]),
lineage(P:X2[]),
lineage(P:X2[4])

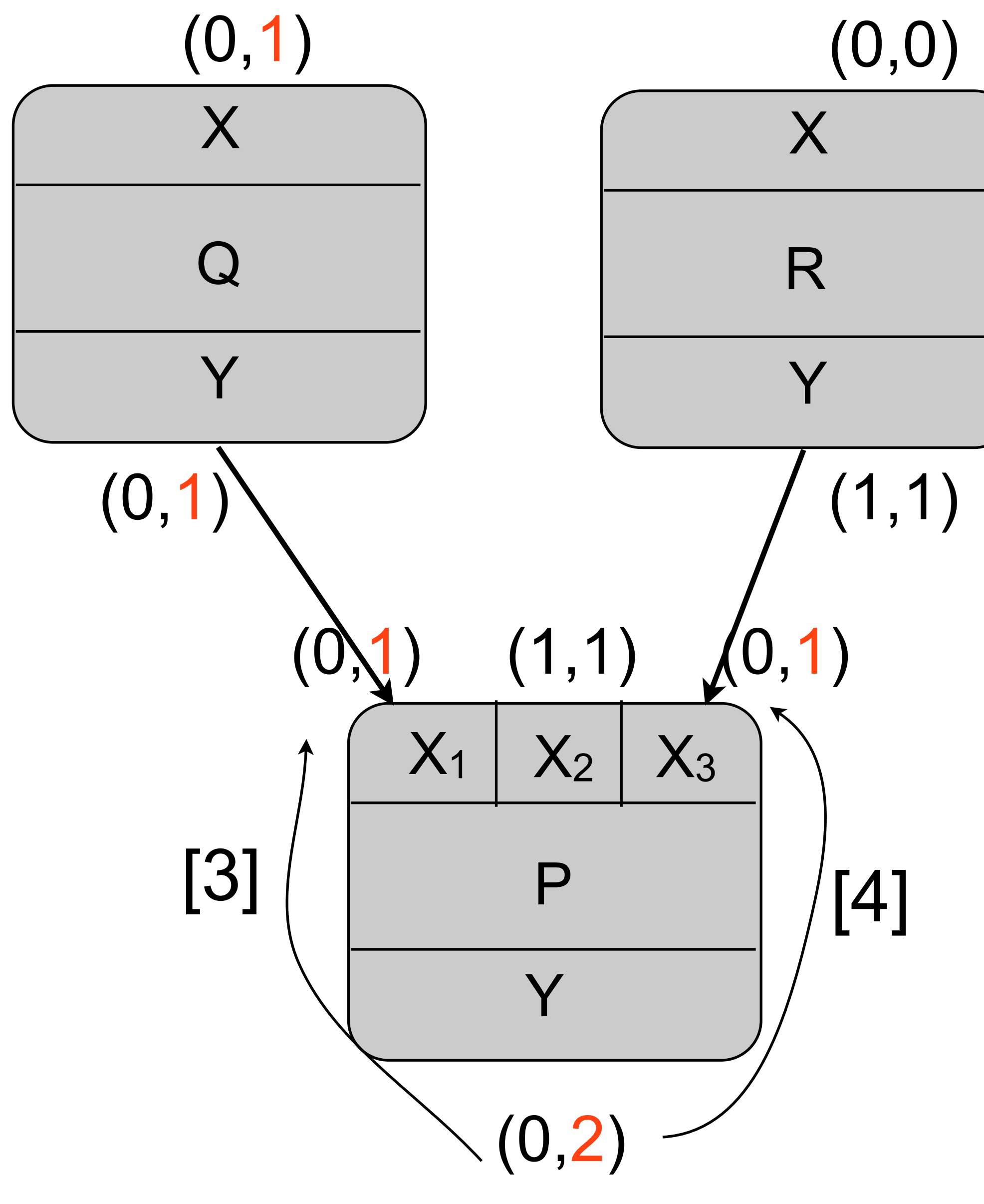
Extension to the entire workflow graph



Y = [[...], ... [...]]

lineage(P:Y[3,4]) → lineage(P:X1[3]),
lineage(P:X2[]),
lineage(P:X2[4])

Extension to the entire workflow graph

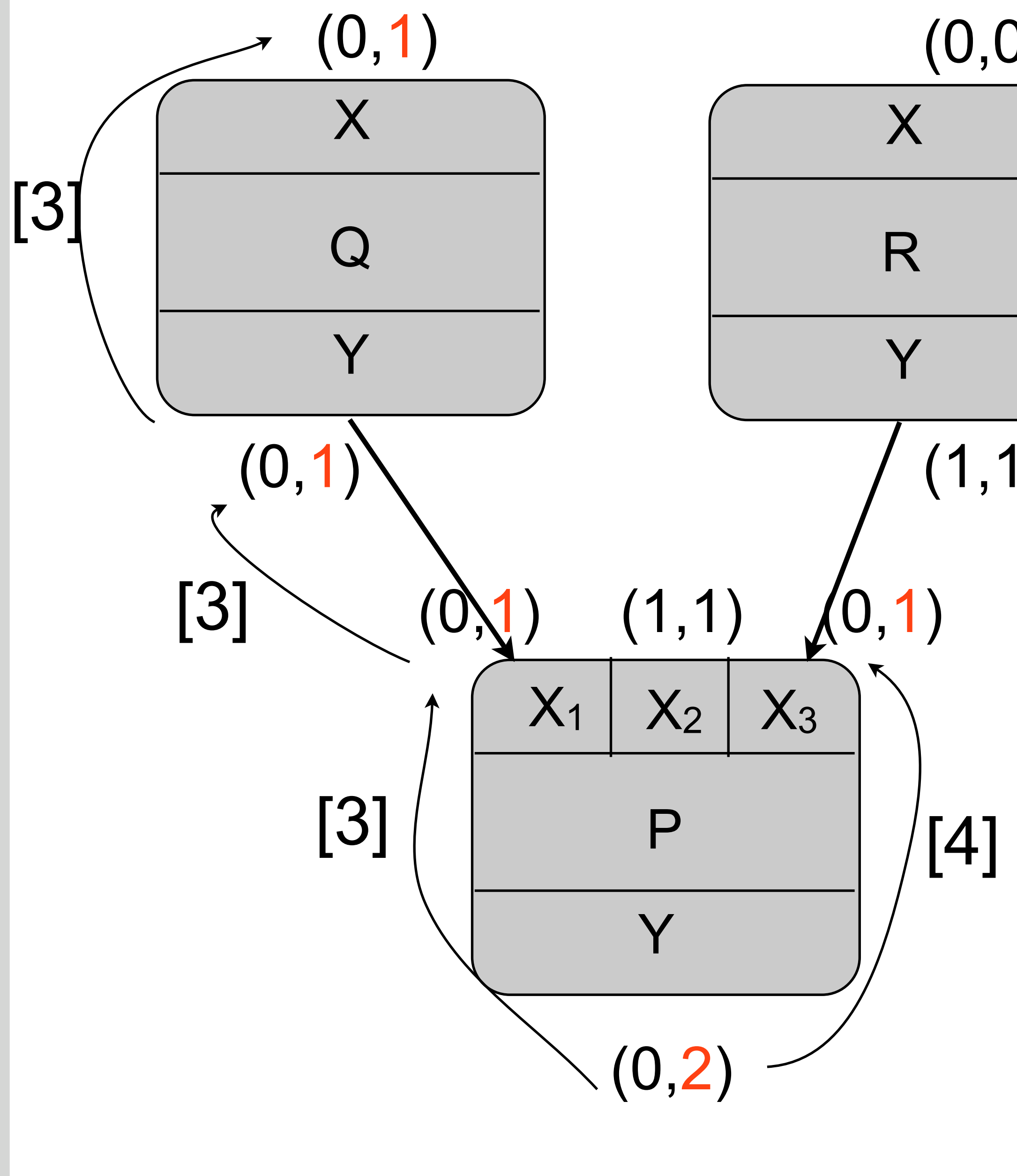


Y = [[...], ... [...]]

lineage(P:Y[3,4]) → lineage(P:X1[3]),
lineage(P:X2[]),
lineage(P:X2[4])

lineage(P:X1[3]) = lineage(Q:Y[3]) →
lineage(Q:X[3])

Extension to the entire workflow graph

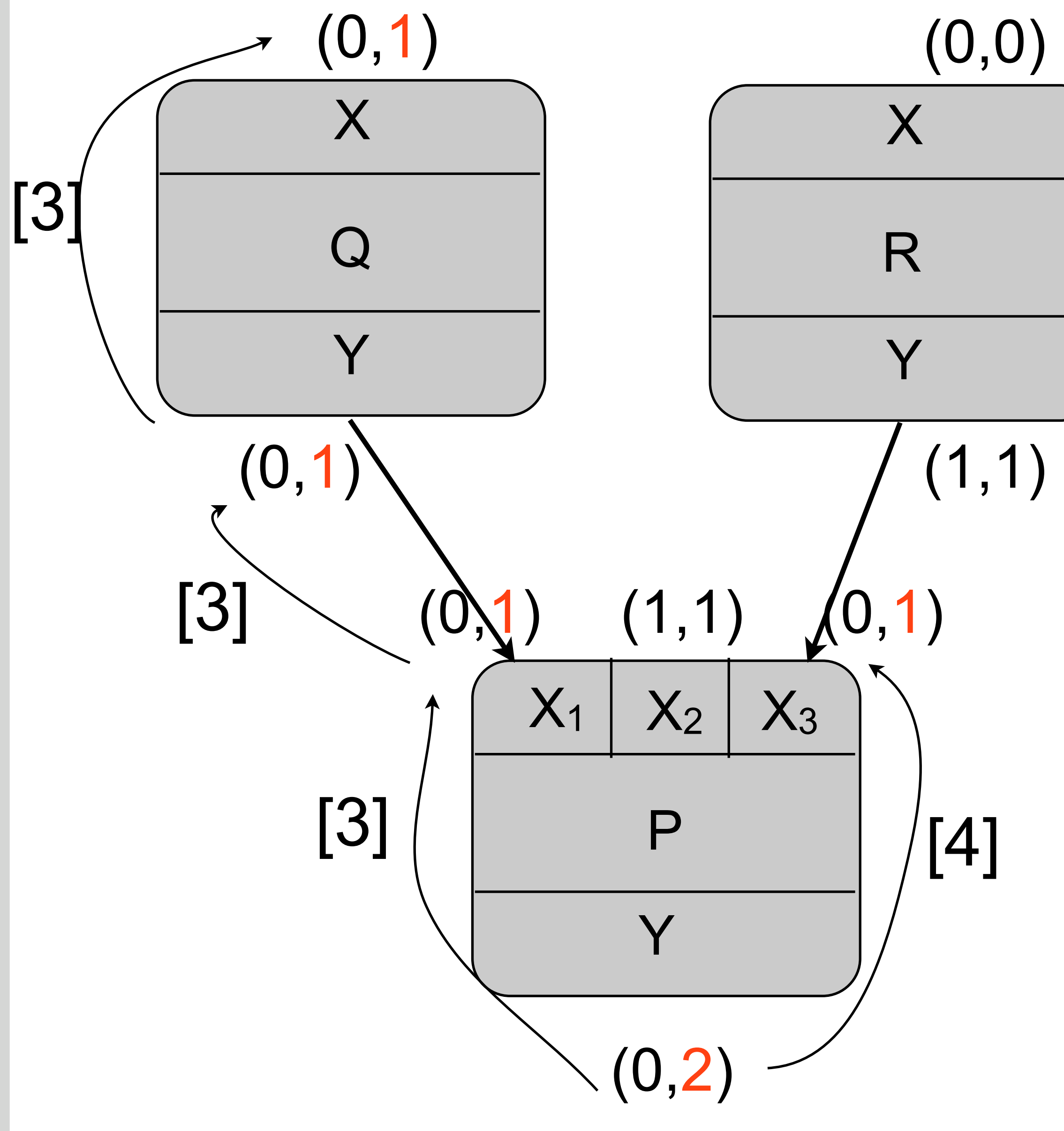


Y = [[...], ... [...]]

lineage(P:Y[3,4]) → lineage(P:X1[3]),
lineage(P:X2[]),
lineage(P:X2[4])

lineage(P:X1[3]) = lineage(Q:Y[3]) →
lineage(Q:X[3])

Extension to the entire workflow graph



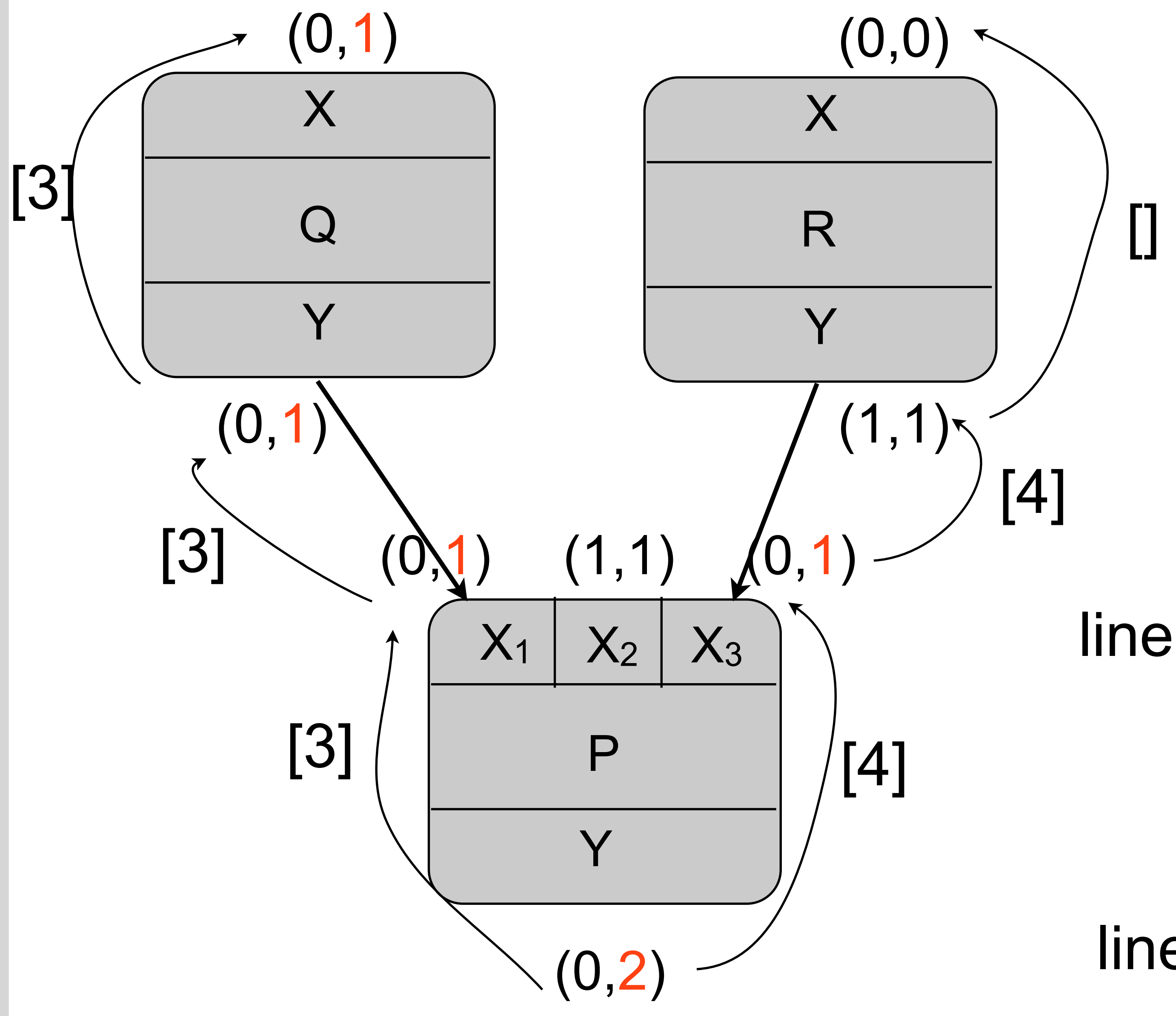
Y = [[...], ... [...]]

lineage(P:Y[3,4]) → lineage(P:X1[3]),
lineage(P:X2[]),
lineage(P:X2[4])

lineage(P:X1[3]) = lineage(Q:Y[3]) →
lineage(Q:X[3])

lineage(P:X3[4]) = lineage(R:Y[4]) →
lineage(R:X[])

Extension to the entire workflow graph



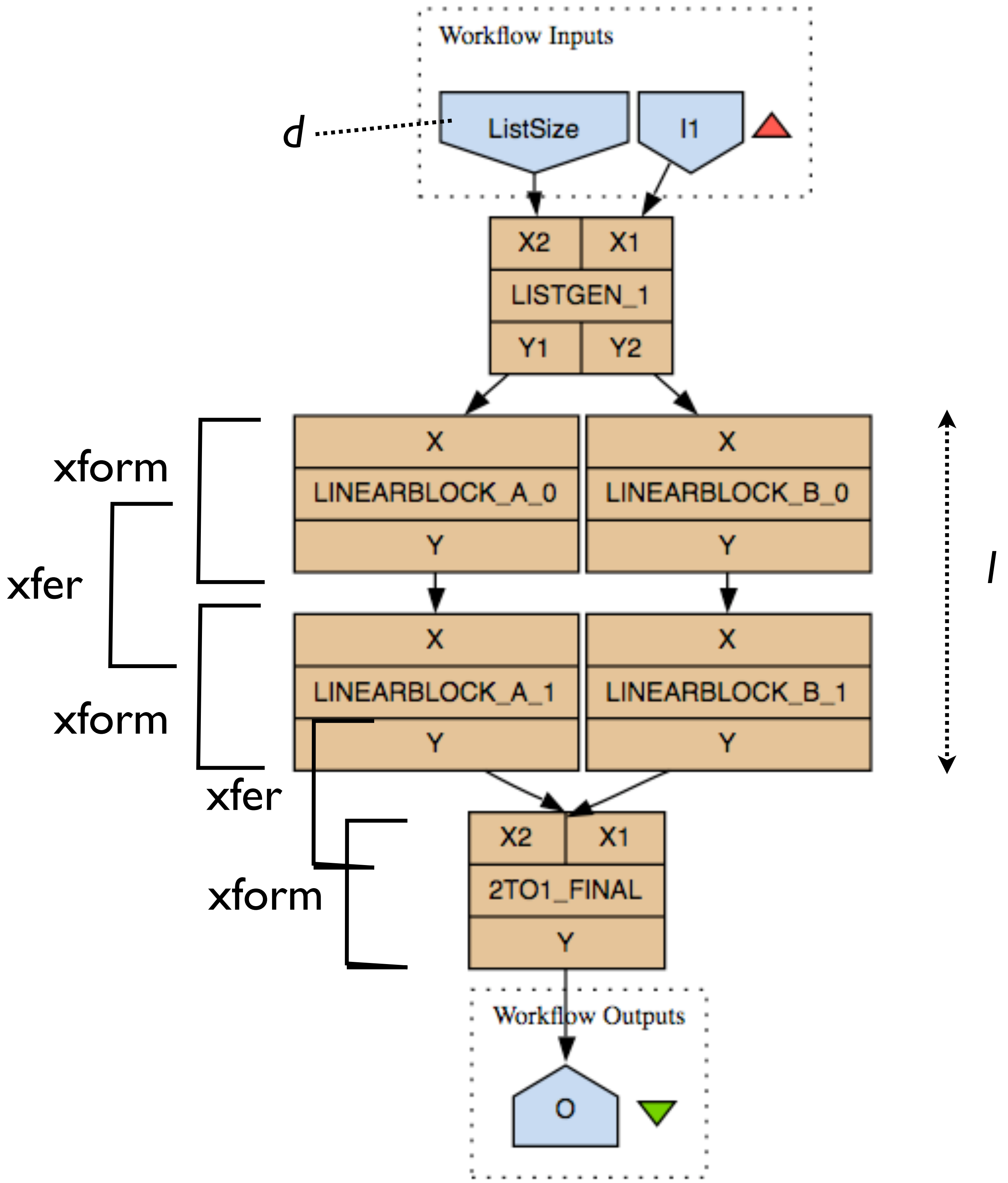
Y = [[...], ... [...]]

lineage(P:Y[3,4]) → lineage(P:X1[3]),
lineage(P:X2[]),
lineage(P:X2[4])

lineage(P:X1[3]) = lineage(Q:Y[3]) →
lineage(Q:X[3])

lineage(P:X3[4]) = lineage(R:Y[4]) →
lineage(R:X[])

Query processing



- Query processing:
  - alternating sequence of *xform* and *xfer* steps
- apply path projection at each *xform* step
- A complete granular and focused query can be answered by traversing the workflow graph alone
  - starting from the target vars
  - one simple query for each selected processor input port

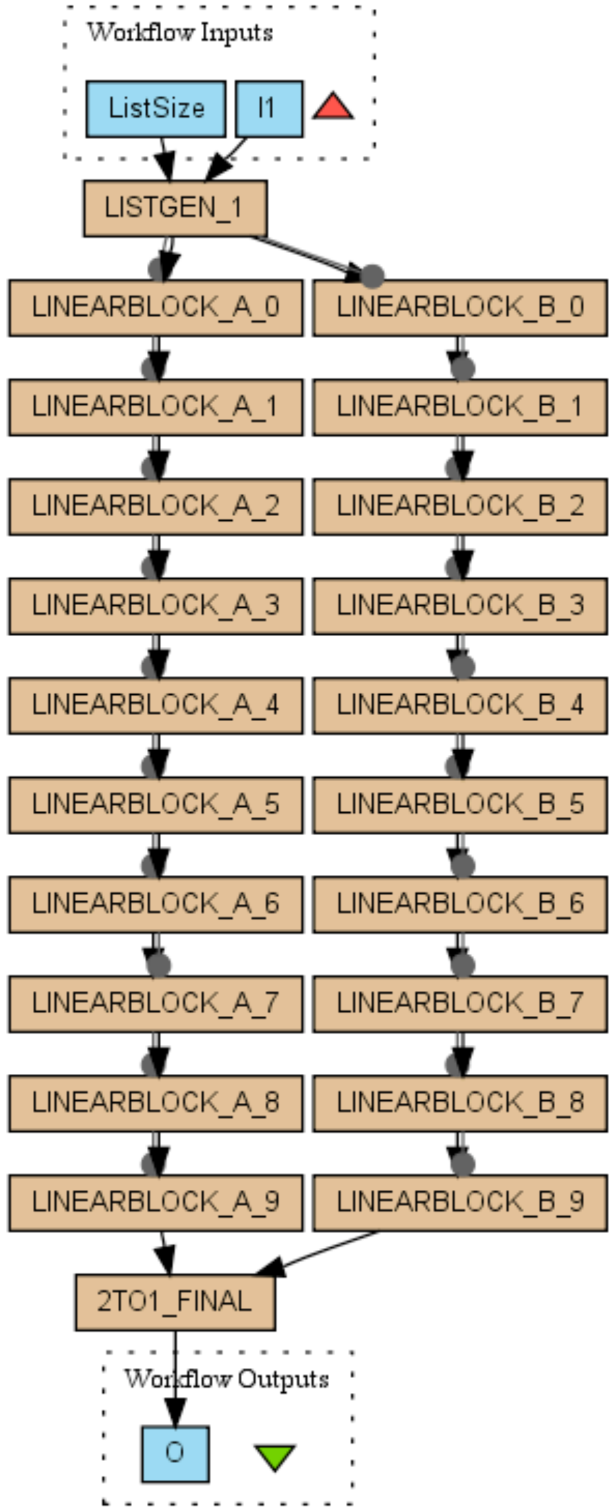
Advantages

- Scalability:
  - query time depends on size of workflow graph, not size of provenance graph
  - workflow graphs are small, fit in memory, can be indexed easily, etc.
  - search over a graph at least as large as the workflow graph is inevitable -- this is the baseline cost!
- Graceful degradation:
  - worst case is a completely unfocused query
  - one query to trace at each *xform* step
  - no worse than other approaches
- Fine-grain answers provided at no additional cost

Experimental setup - I

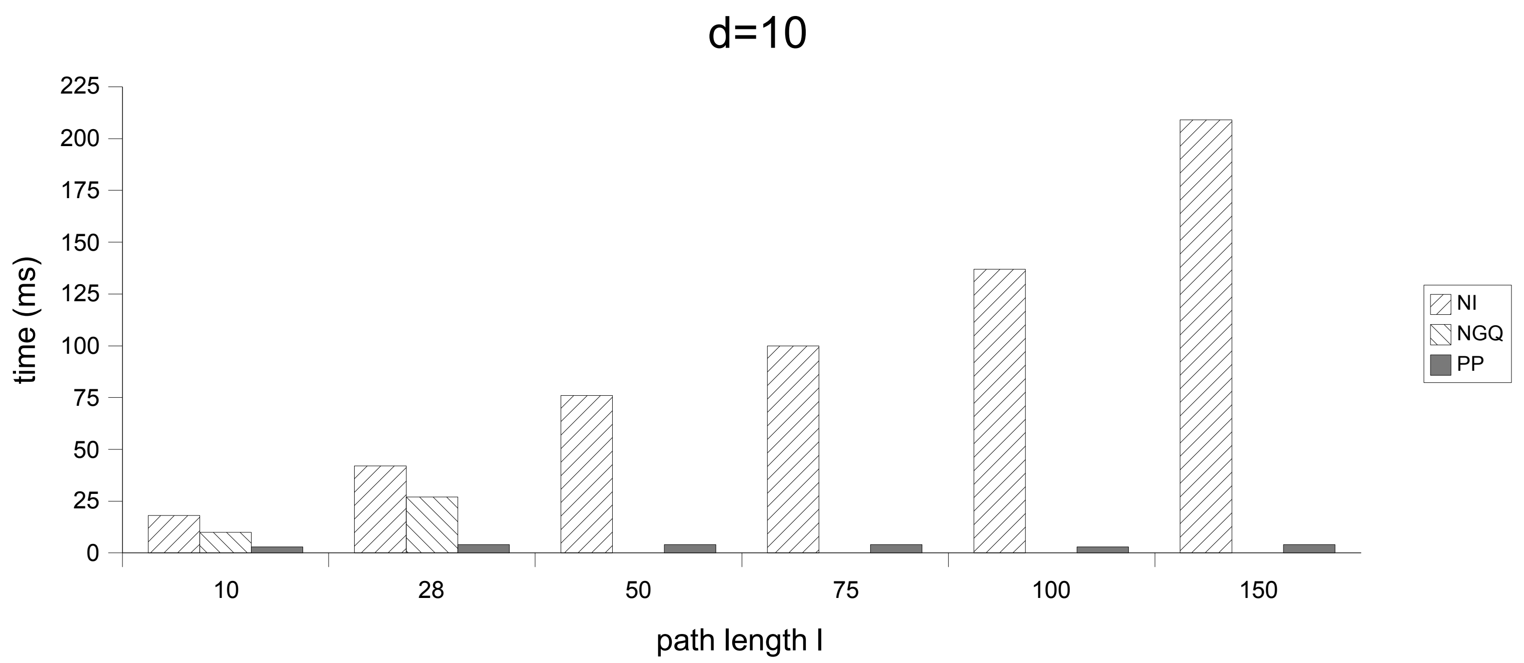
- Performance evaluation performed on programmatically generated dataflows
  - the "T-towers"

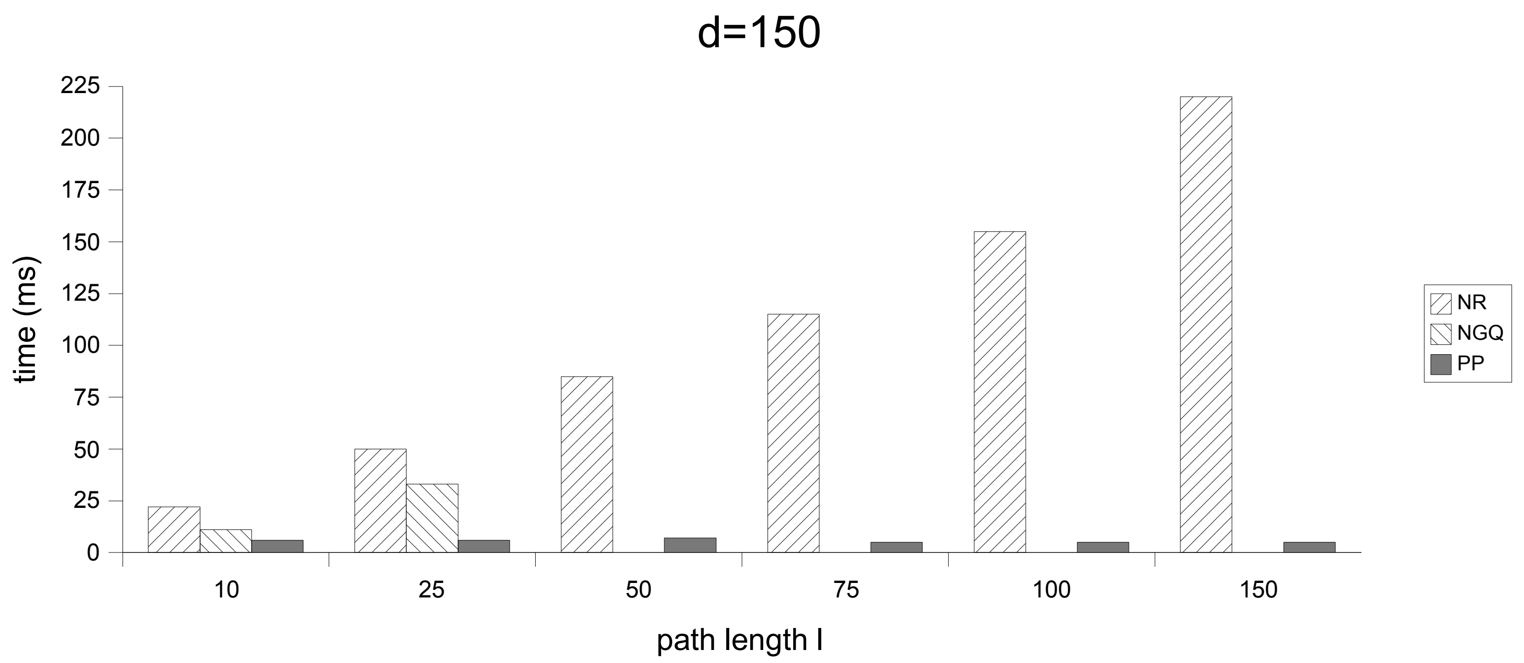
control:
- size of the lists involved
- length of the paths
- includes one cross product



Experimental results - I

- query response time: naive vs. "path projection" approaches



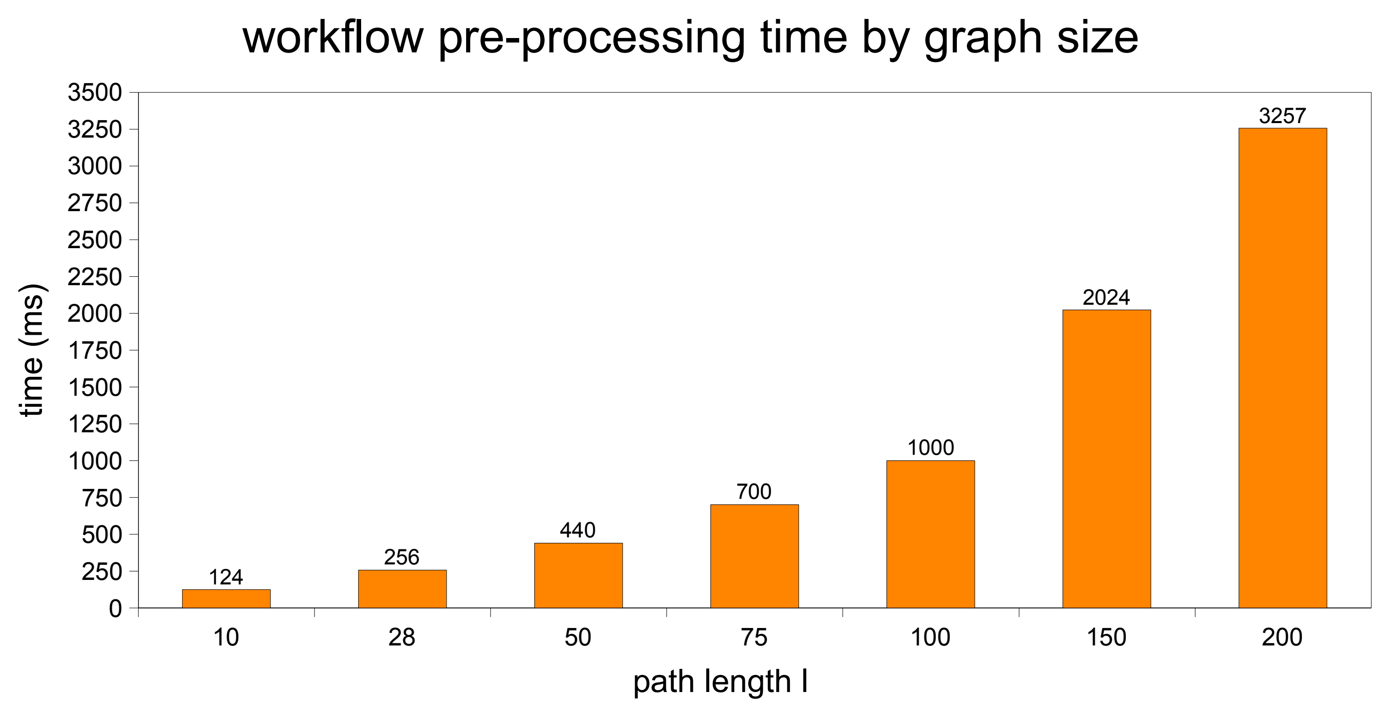


MANCHESTER 1824

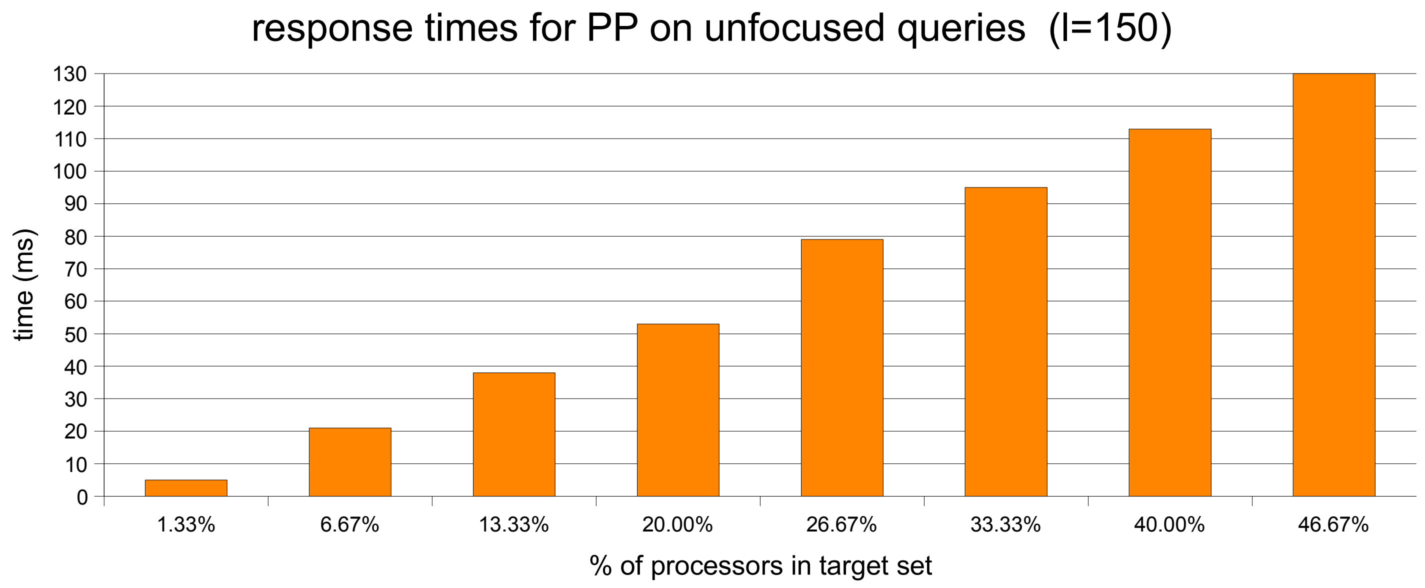
The University of Manchester

Experimental results - II

- workflow search time by path length (“tower height”)
  - common to all strategies!



- performance degradation on fully unfocused queries



Summary

- An original approach to lineage queries for Taverna that combines
  - efficiency and fine-granularity
- Relies on semantic properties of the Taverna dataflow model
- Further work:
  - visual specification of user query
  - visual presentation of query answer
  - space compression
  - semantic overlays, annotations
- To be bundled with some future version of Taverna...