Granular workflow provenance in Taverna

Paolo Missier
Information Management Group
School of Computer Science, University of Manchester, UK

Symposium on Provenance in Scientific Workflows
Salt Lake City, Oct. 2008

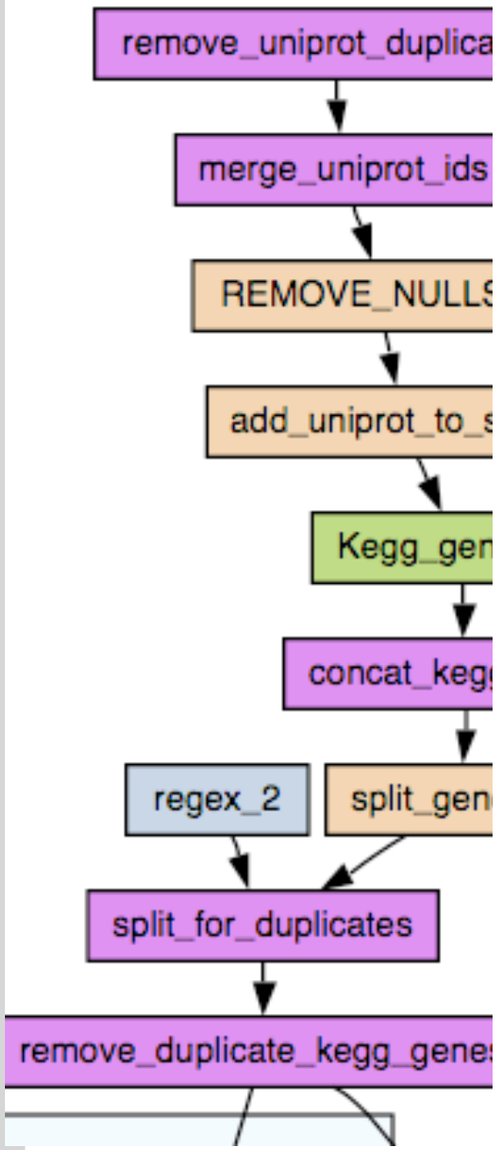
Outline

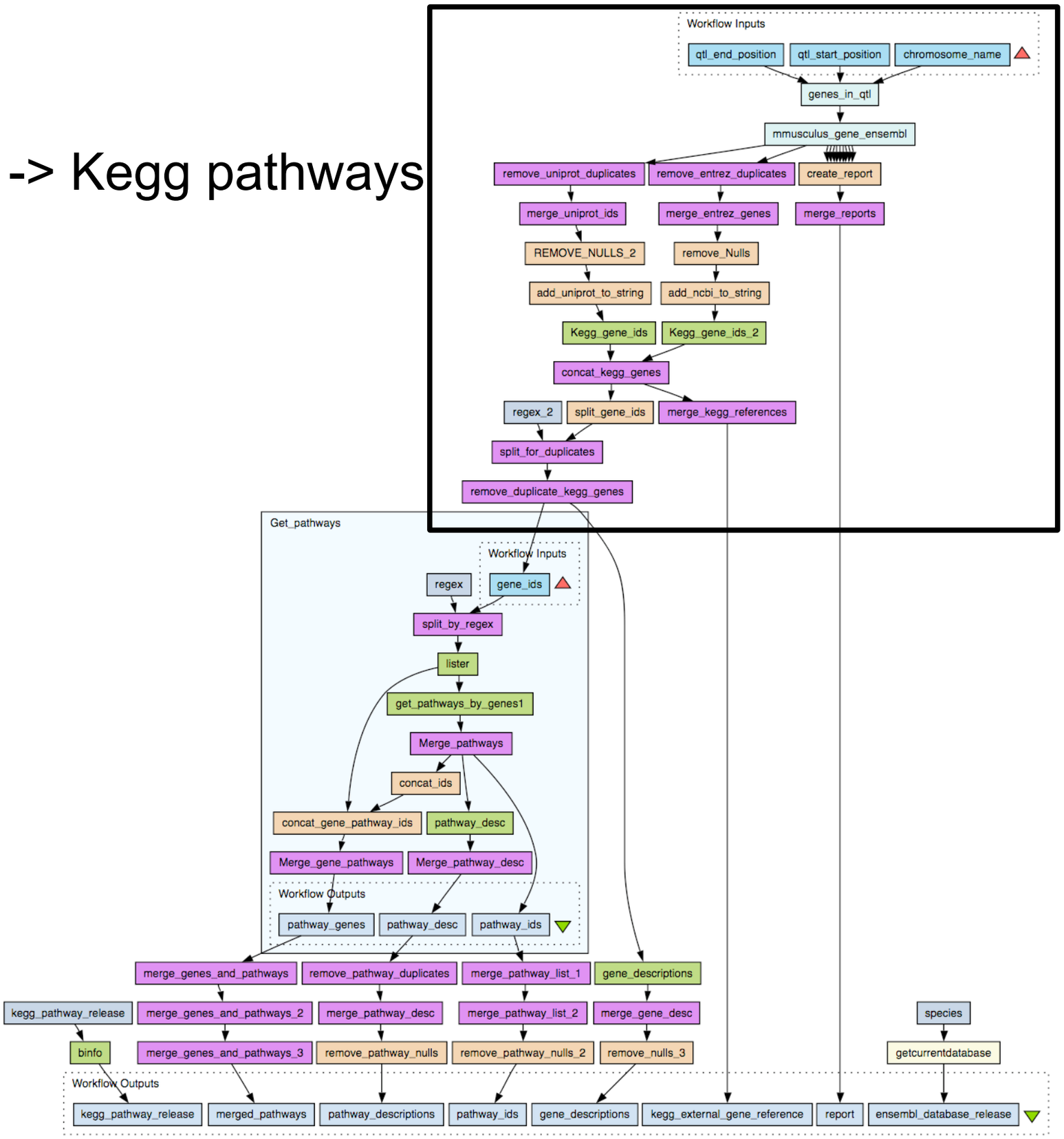
- Collection values in [bioinformatics] workflows are important
- Granular provenance over collections: model and issues
- Measuring “provenance friendliness” of dataflows
- Increasing friendliness of existing dataflows
- Extending the Open Provenance Model graph to describe granular data derivations

- Provenance service architecture - brief description

Example (Taverna) dataflow

QTL -> genes -> Kegg pathways

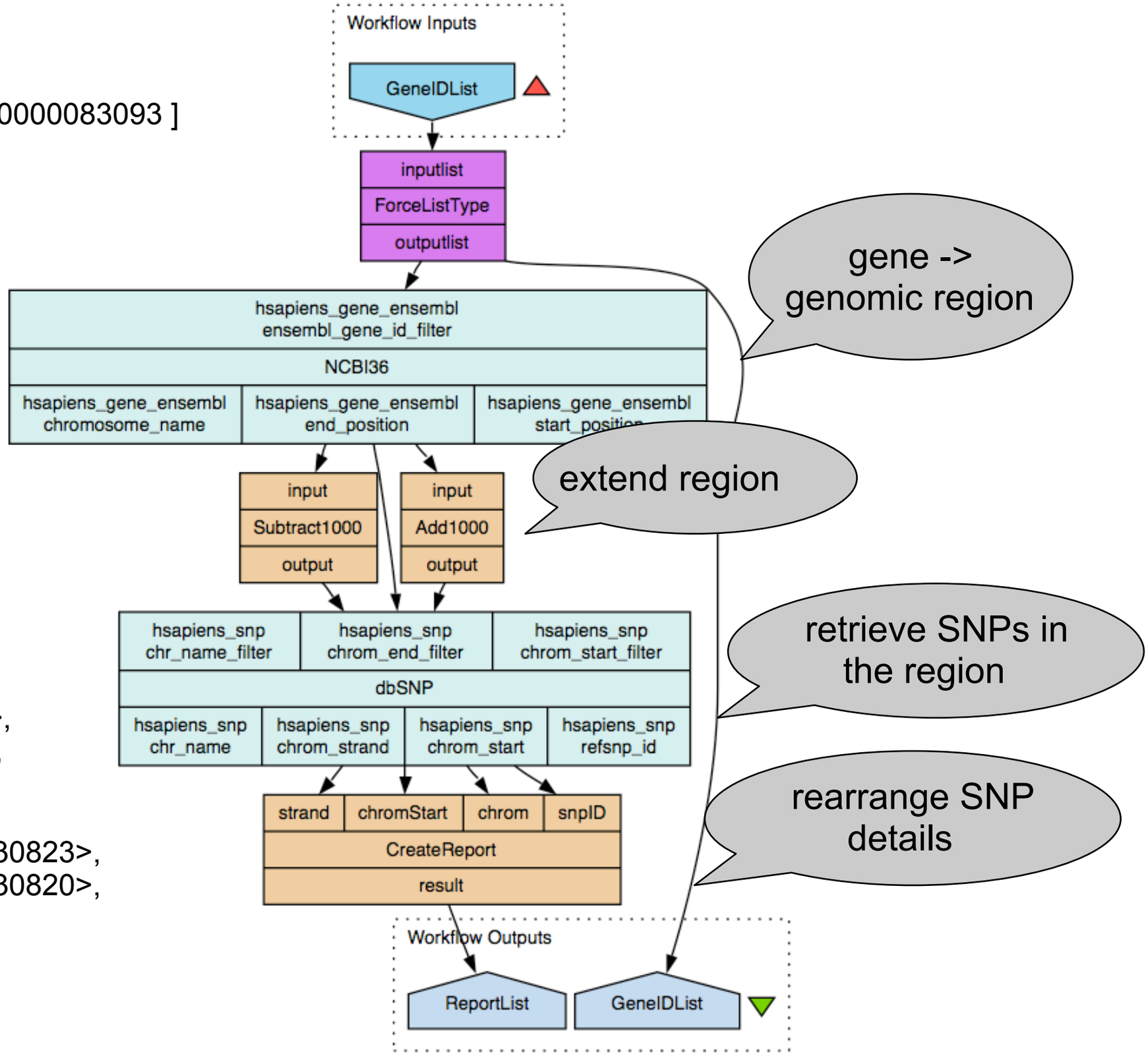




Collections example: from genes to SNPs

- See myexperiment.org: http://www.myexperiment.org/workflows/166

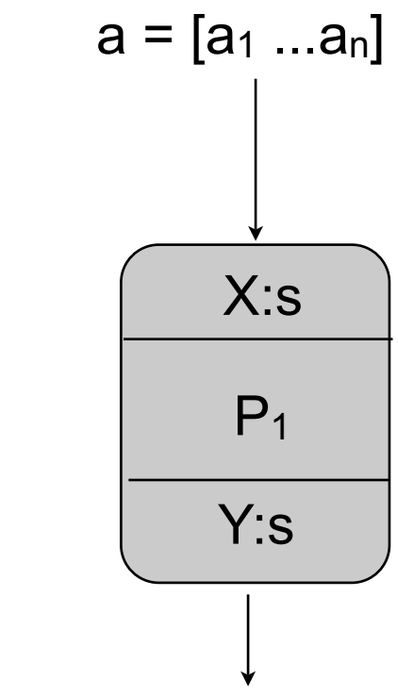
[ENSG00000139618 , ENSG00000083093]



[[<1,23554512,16,rs45585833>,
<1,23554712,16,rs45594034>,
...
],
[<1,31820153,13,ENSSNP10730823>,
<1,31818497,13,ENSSNP10730820>,
...
]]

Computational model for collections

Implicit iteration over a collection



Nesting level mismatch:

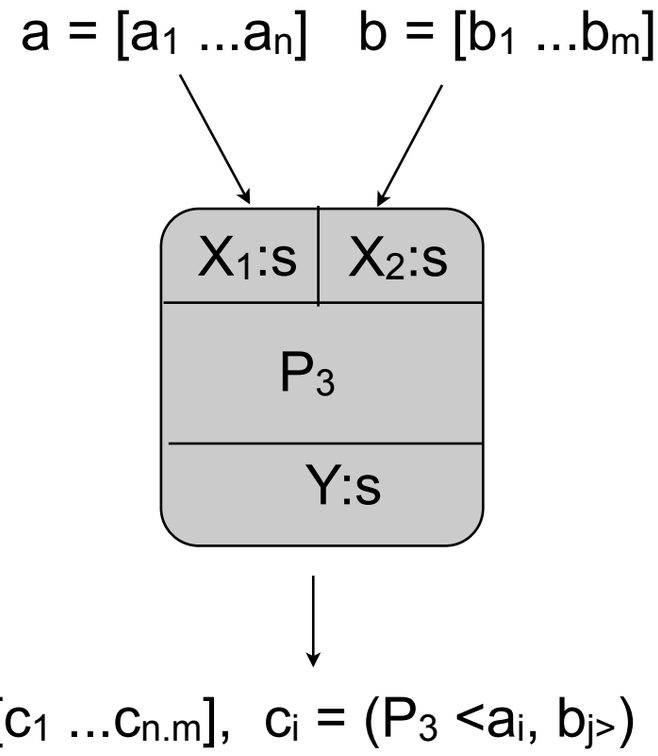
X:s but a:I(s)

Semantics of P_1's execution:

Y = (map P_1 X)

Y = [(P_1 a_1) ... (P_1 a_n)]

Implicit iteration over multiple collections



Nesting level mismatch:

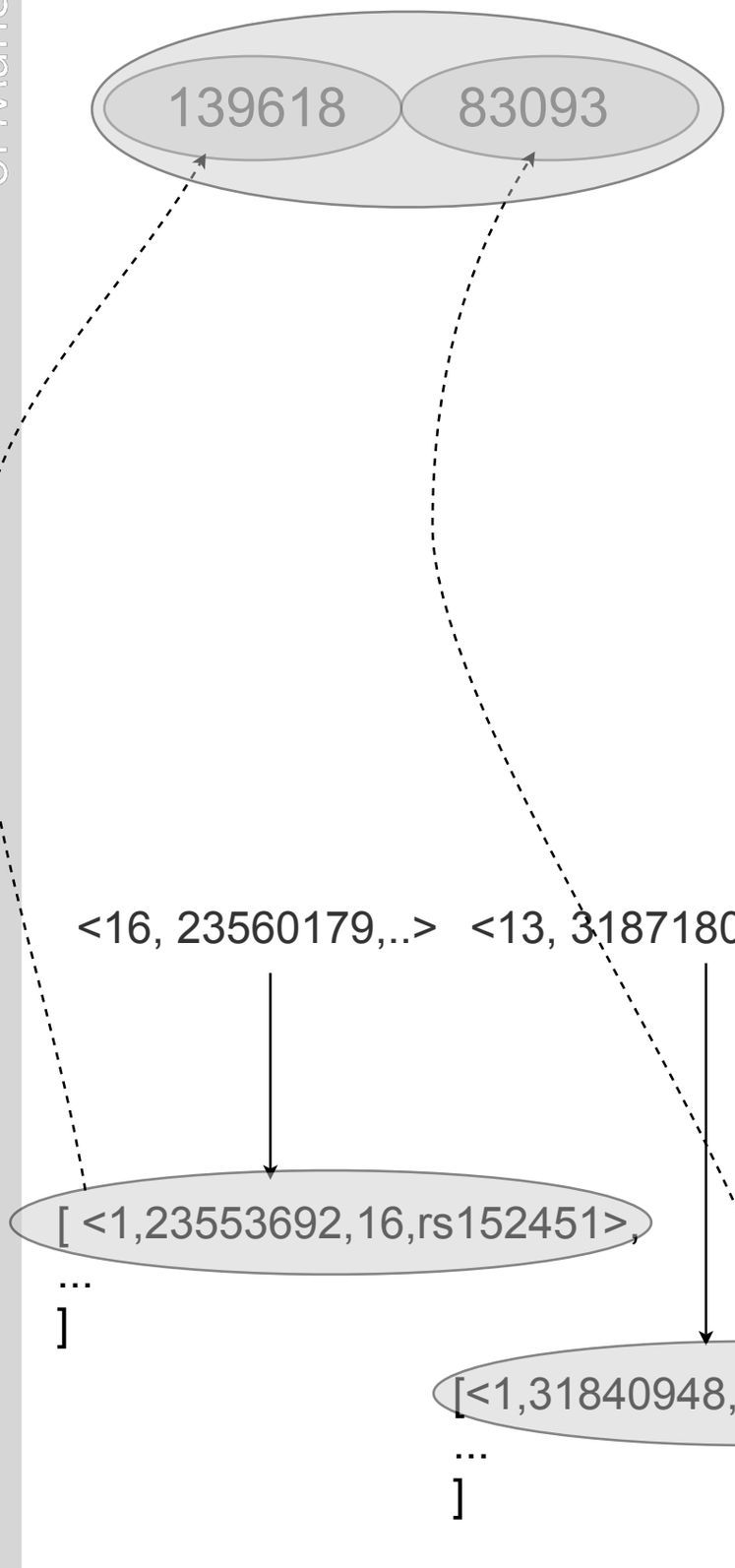
X_1:s but a:I(s), X_2:s but b:I(s)

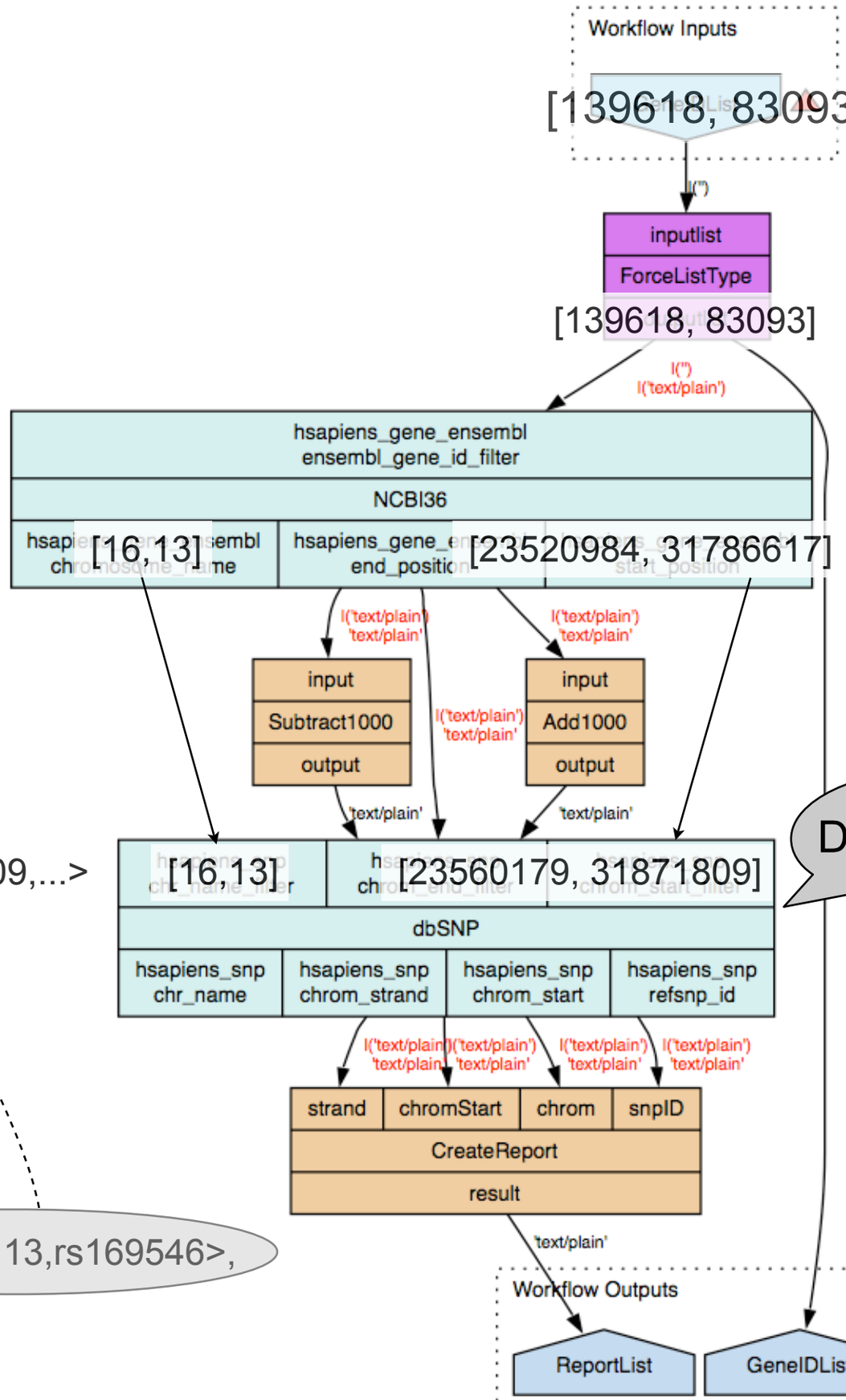
Semantics of P_1's execution:

Y = (map P_1 $X_1 \otimes X_2$) // cross product

Y ← [(P_1 <a_1,b_1>) ... (P_1 <a_n,b_m>)]

Collections and iterations





Processor signatures

l(s) → l(s)

l(s) → l(s)

s → s

Dot product

s → l(s)

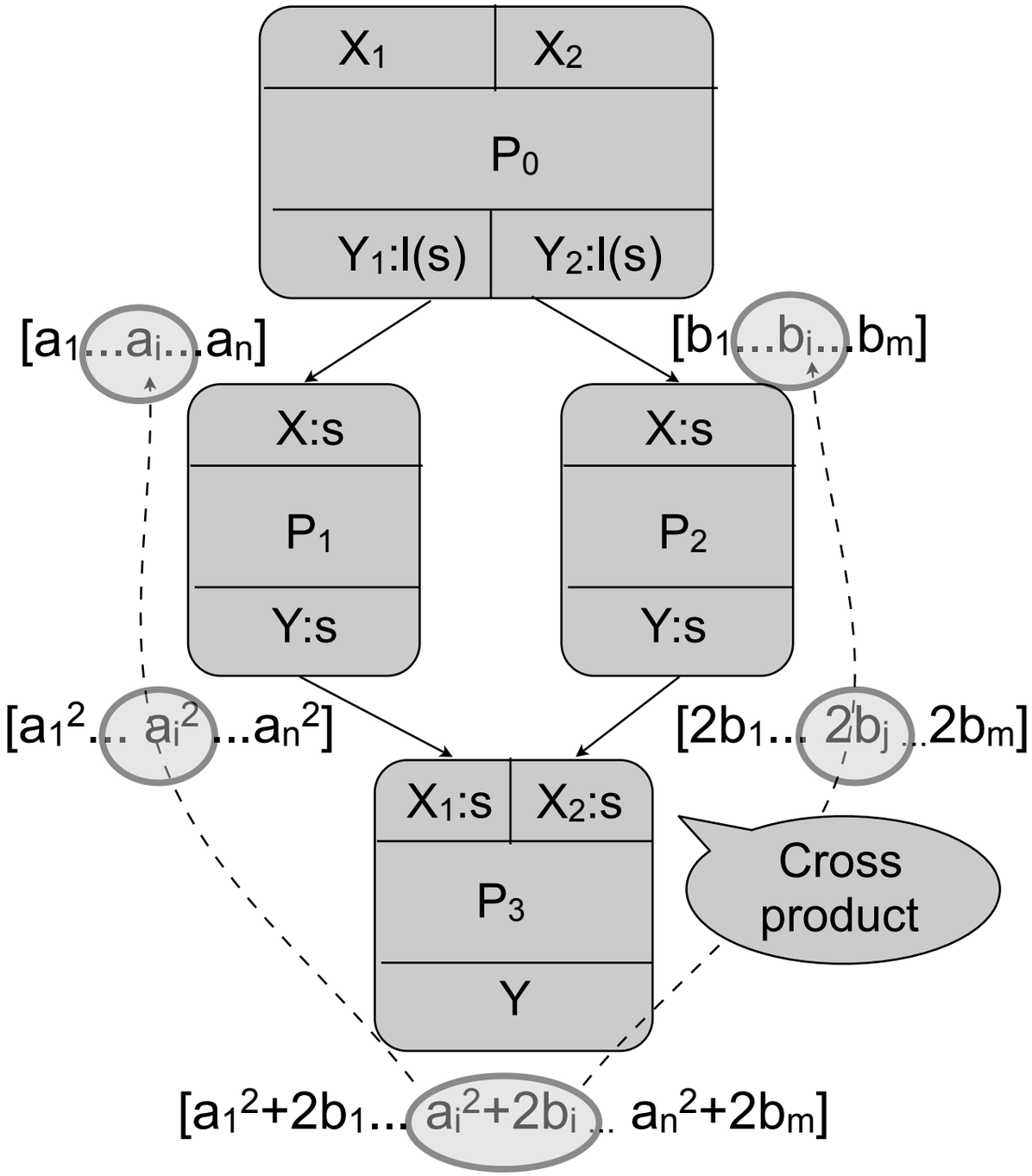
s → s

Tracing granular lineage

- Provenance traces are most useful when they are granular
  - trace individual items in a collection
  - “which geneID is responsible for the presence of SNP rs169546 in the output?”

- Curse of black box processors:
  - M-M (many-many) and M-1 (many-one) processors destroy granularity

Granular lineage I: no loss of precision



$P_1 \equiv \lambda X . X^2$

$P_2 \equiv \lambda X . 2X$

$P_3 \equiv \lambda X_1 . \lambda X_2 . X_1 + X_2$

Let

$P_0{:}Y_1 = [a_1 ... a_n]$,

$P_0{:}Y_2 = [b_1 ... b_m]$

Then,

$P_1{:}Y = [a_1^2 ... a_n^2]$,

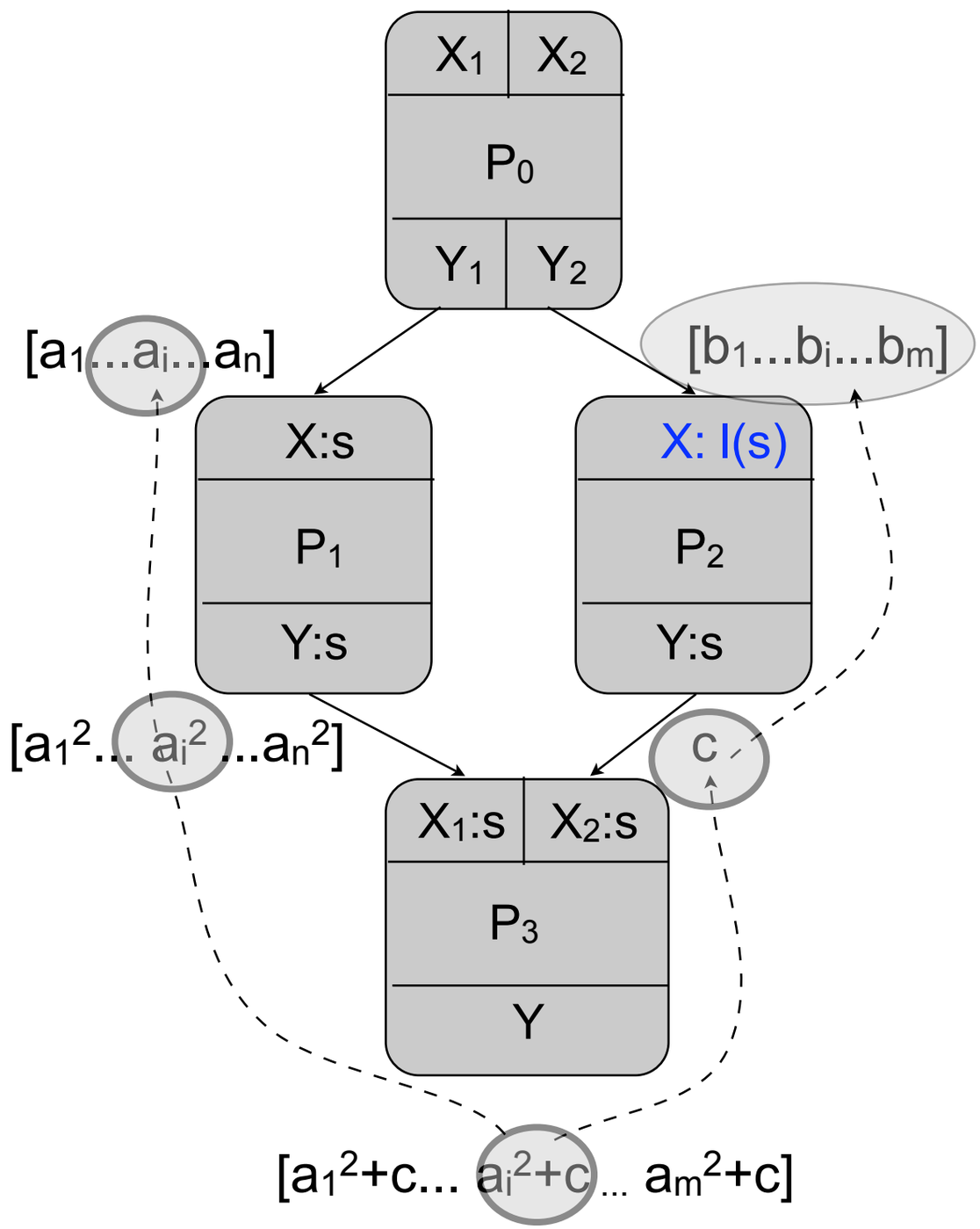
$P_2{:}Y = [2b_1 ... 2b_m]$

$P_3{:}Y = [a_1^2+2b_1 ... a_n^2+2b_m]$

And

lineage($P_3{:}Y[i]$, $\{P_0\}$) = { $P_0{:}Y_1[i]$, $P_0{:}Y_2[j]$ }

Granular lineage II: loss of precision



$P_1 \equiv \lambda X . X^2$

$P_2 \equiv \lambda X . \min X$

$P_3 \equiv \lambda X_1 . \lambda X_2 . X_1 + X_2$

Let

$P_0{:}Y_1 = [a_1 ... a_n]$,
$P_0{:}Y_2 = [b_1 ... b_m]$

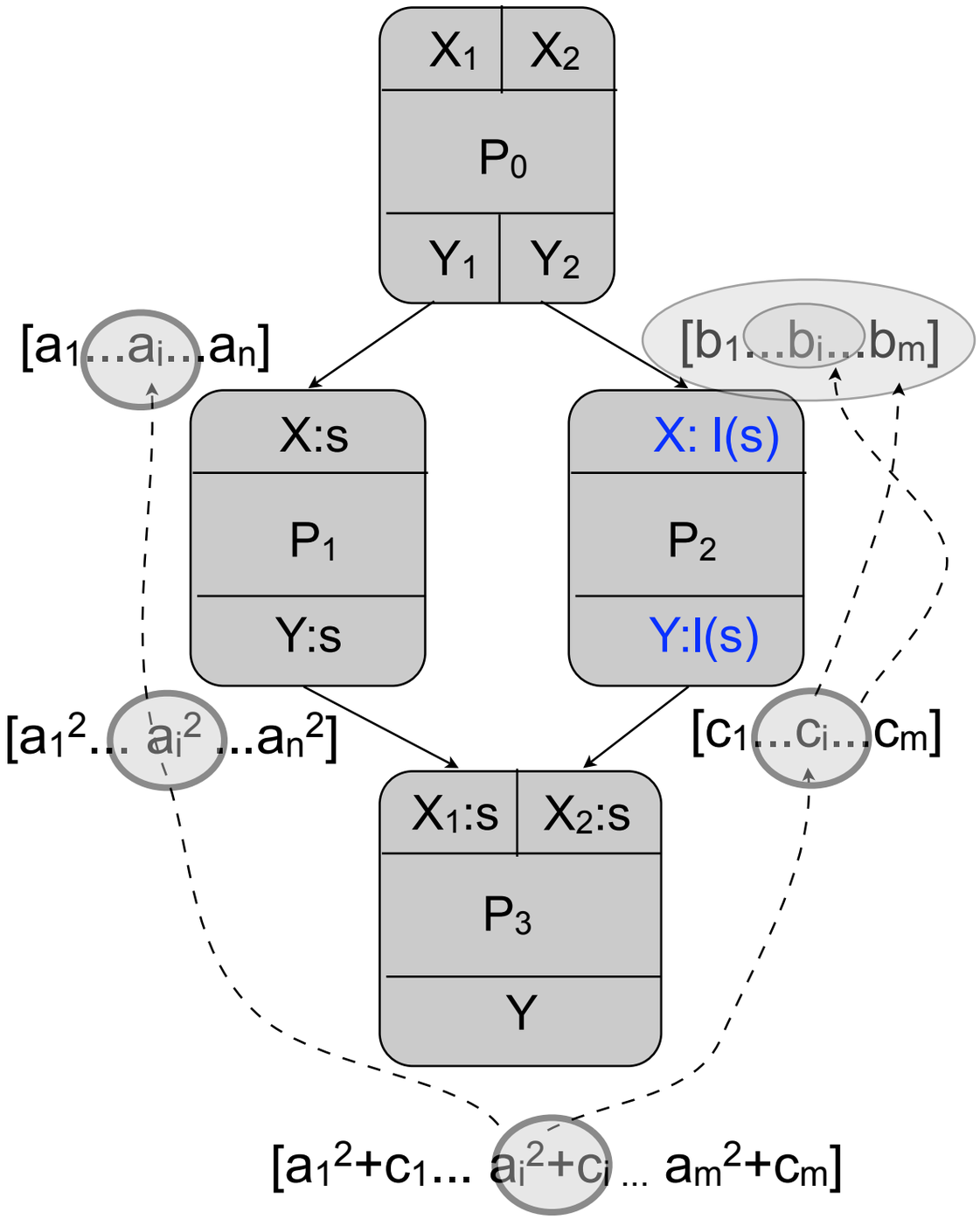
Then,

$P_1{:}Y = [a_1^2 ... a_n^2]$,
$P_2{:}Y = c = \min \{b_1 ... b_m\}$
$P_3{:}Y = [a_1^2 + c ... a_m^2 + c]$

And

lineage$(P_3{:}Y[i]) = \{ P_0{:}Y_1[i], P_0{:}Y_2 \}$

III: recoverable loss of precision



$P_1 \equiv \lambda\, X\, .\, X^2$

$P_2 \equiv \lambda\, X\, .\, f\, X$

"f is index-preserving"

$P_3 \equiv \lambda\, X_1\, .\, \lambda\, X_2\, .\, X_1 + X_2$

Let $P_0{:}Y_1 = [a_1 ... a_n]$, $P_0{:}Y_2 = [b_1 ... b_m]$

Then, $P_1{:}Y = [a_1^2 ... a_n^2]$, $P_2{:}Y = c$

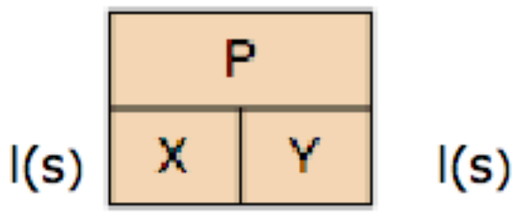
$P_3{:}Y = [a_1^2 + c ... a_m^2 + c]$

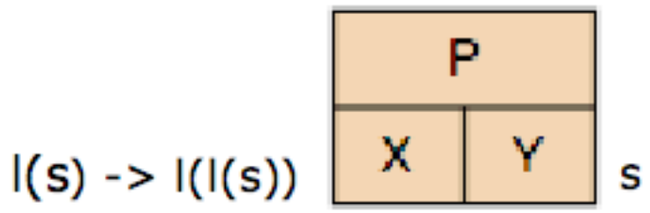
And

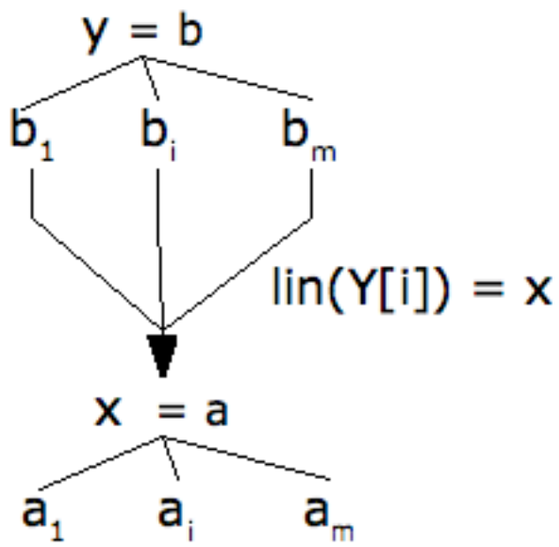
lineage$(P_3{:}Y[i]) = \{ P_0{:}Y_1[i], P_0{:}Y_2 \}$

lineage$(P_3{:}Y[i]) = \{ P_0{:}Y_1[i], P_0{:}Y_2[i] \}$

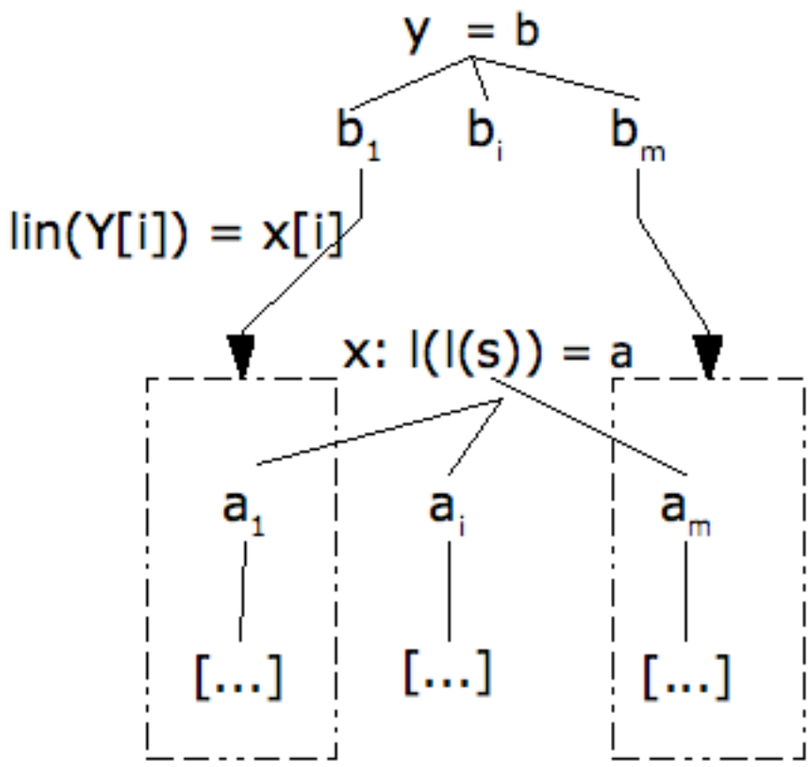
Multi-level nesting and lineage precision







(a)



(b)

Adding annotations to the original workflow

The University of Manchester

geneIdList:

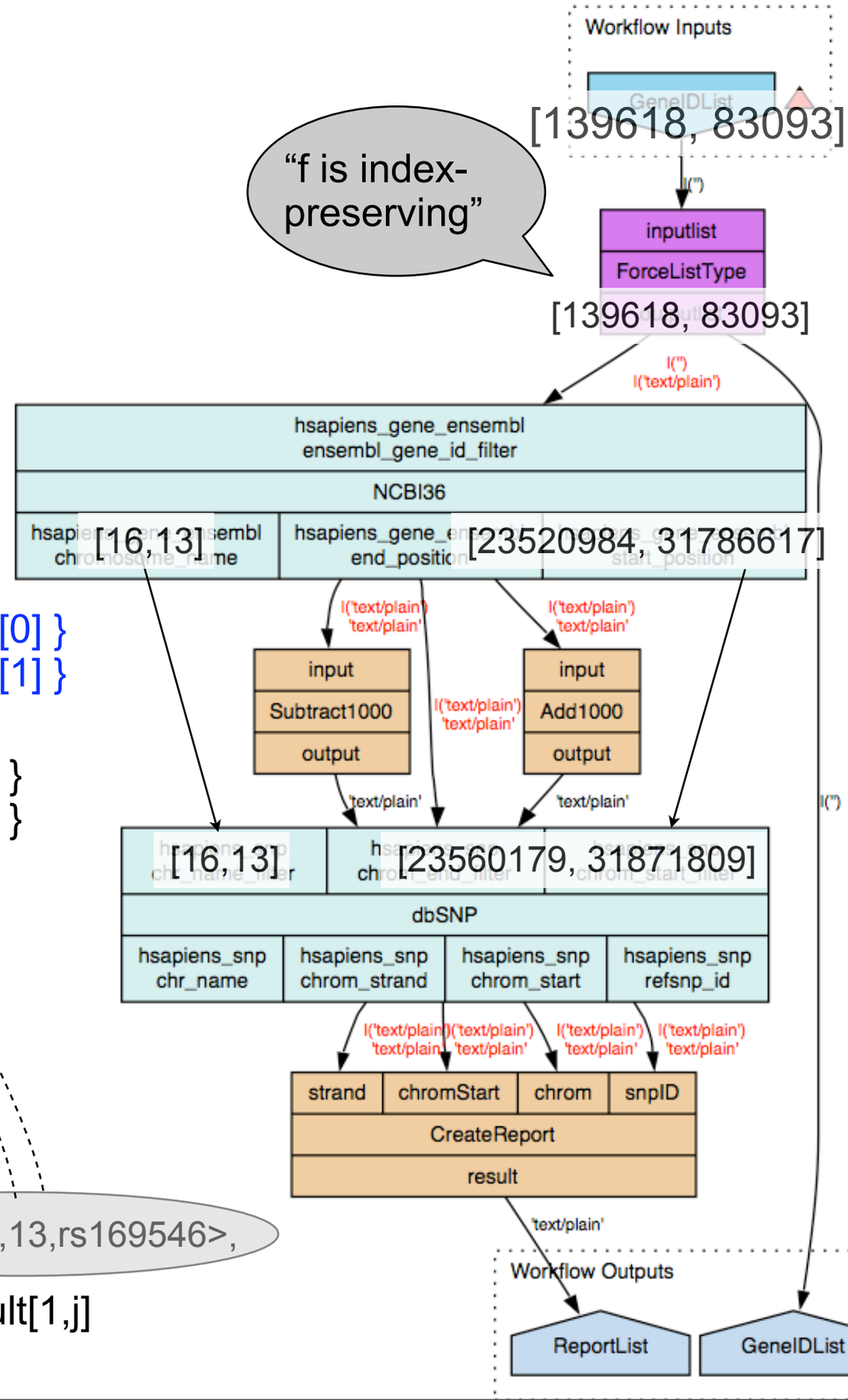
[139618, 83093]

lineage(CR:result[0,i]) = { geneIdList[0] }
lineage(CR:result[1,j]) = { geneIdList[1] }

lineage(CR:result[0,i]) = { geneIdList }
lineage(CR:result[1,j]) = { geneIdList }

[<1,23553692,16,rs152451>,
...
]
CR:result[0,i]

[<1,31840948,13,rs169546>,
...
]
CR:result[1,j]



Processor signatures

l(s) → l(s)

"f is index-preserving"

l(s) → l(s)

s → s

s → l(s)

s → s

Granular lineage: recap

- Lineage query model accounts for granular traces over nested collections
- arbitrary nesting levels:
  - values are trees in general
  - lineage query identifies the correct sub-trees

- Lineage queries are efficient
  - recursion problem “compiled away” by query rewriting
  - (shameless claim - details omitted)

- But:
  - One single M-* processor can destroy granularity
  - in some cases annotations are a remedy

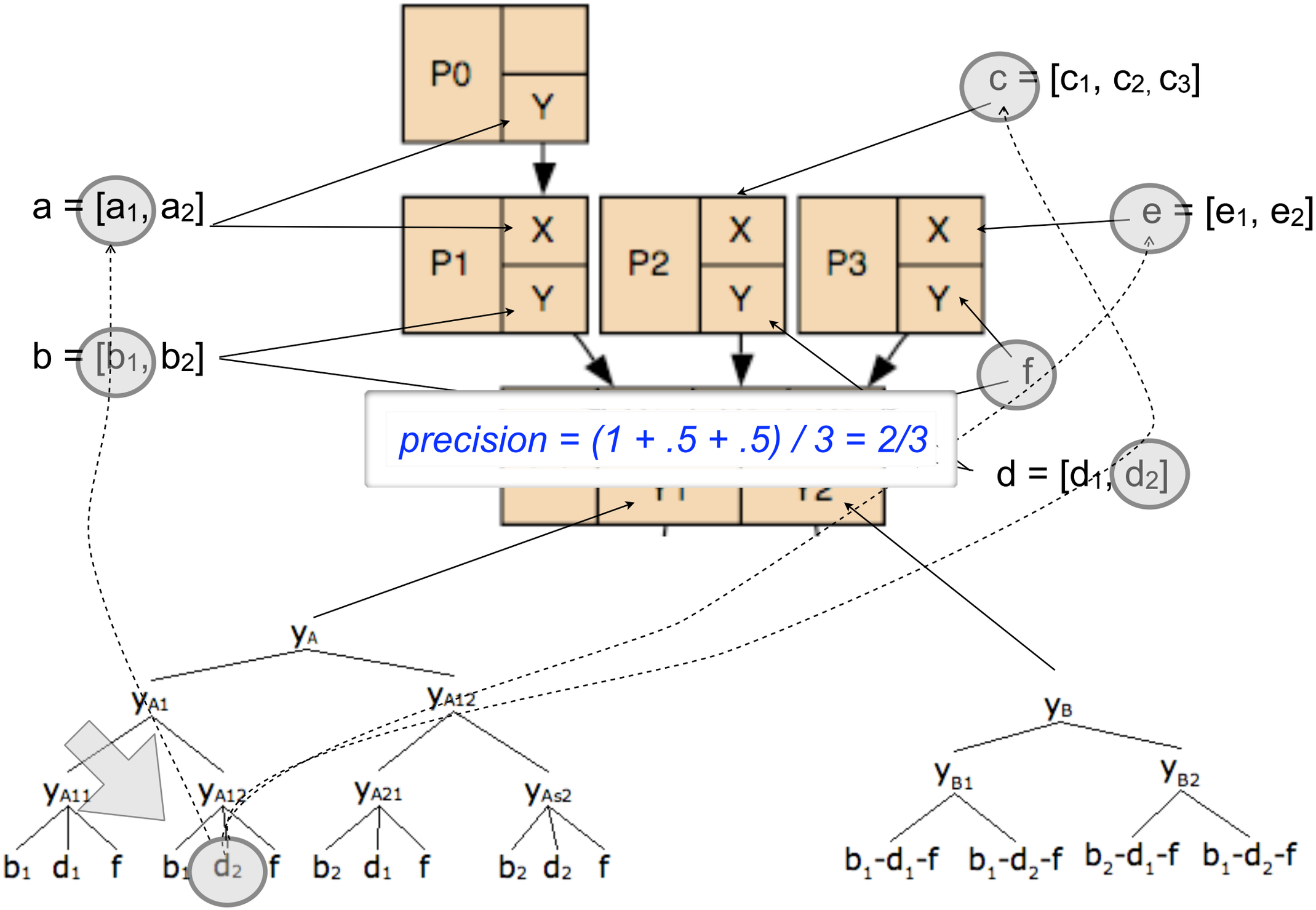
Towards provenance-friendly workflows

1. Define metrics for workflow *provenance precision*
   - how well is granularity preserved over a lineage trace?
   - what is the impact of M-* processors?
   - use to prioritize remedial actions

2. Make workflows more provenance friendly:
   - Add knowledge (static):
     - "lightweight annotations" [MBZ+08] -- see IPAW08
   - Add knowledge (dynamic):
     - provenance-active workflow processors
   - Redesign processors / workflow
     - general guidelines, provenance friendly patterns

[MBZ+08] Missier, Khalid Belhajjame, Jun Zhao, Carole Goble, *Data lineage model for Taverna workflows with lightweight annotation requirement*s, Procs. International Provenance and Annotation Workshop (IPAW 2008)

Lineage precision: example



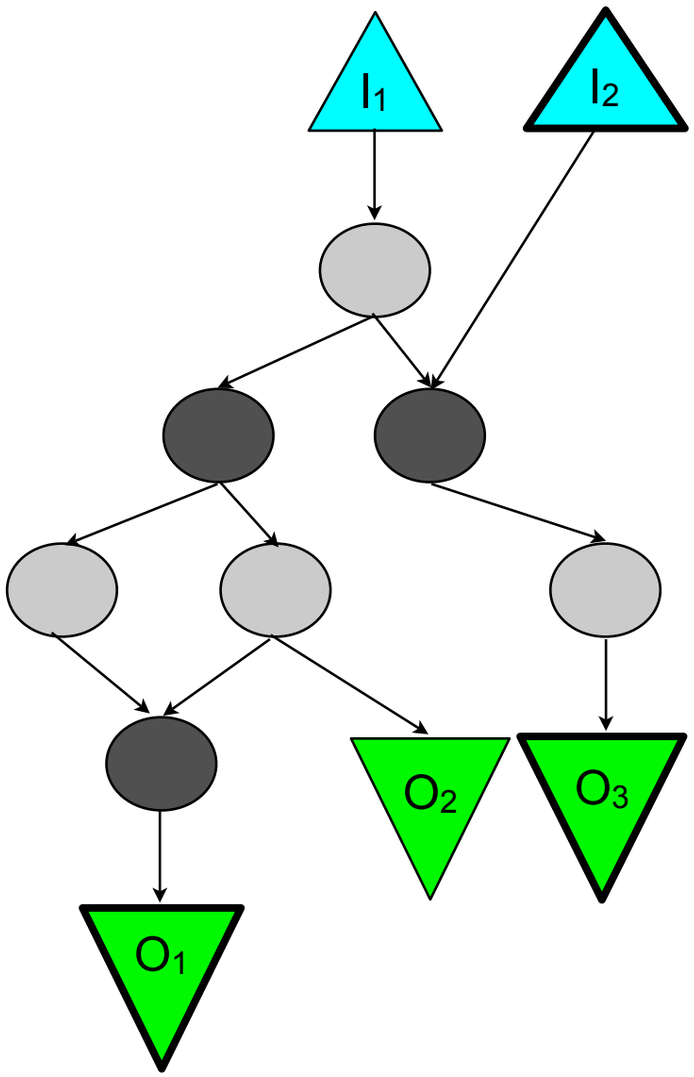
lineage(P_4:Y_1[1.2.2], {P_0, P_2, P_3}) =

{ P_0:Y[1]= a_1, P_2:X=c, P_3:X=e }

Precision relative to a sub-graph

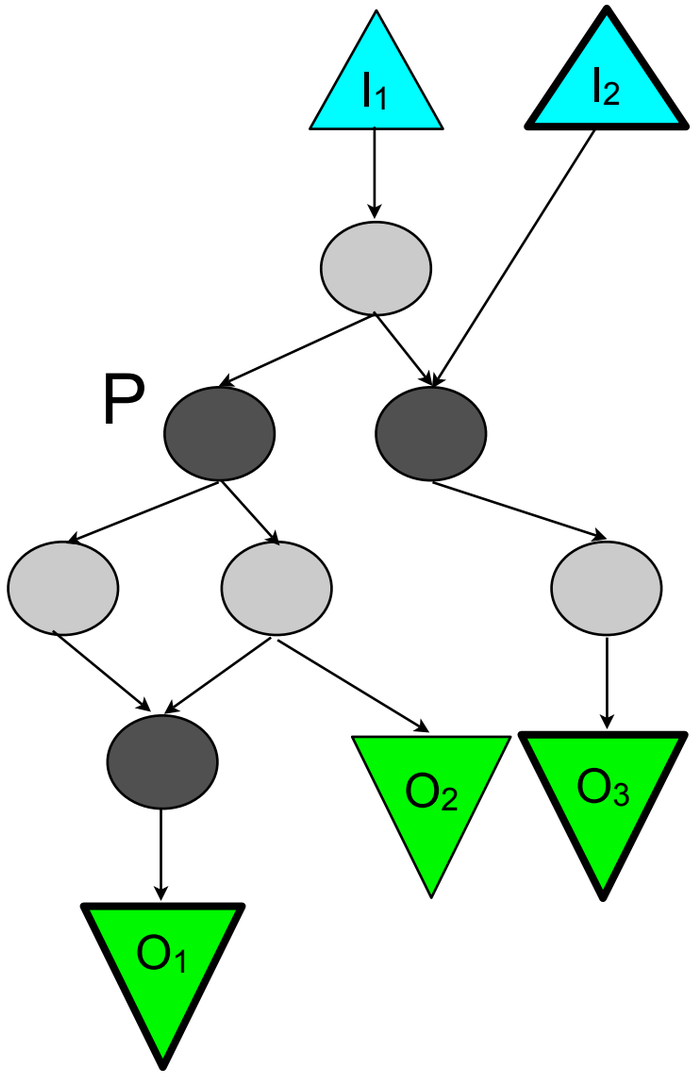
- Refining the previous idea:
  - precision relative to a set O of output variables and a set I of input variables
    - because not all variables are equally interesting...
    - weights W_I, W_O account for relative importance of variables

$$prec(I, W_I, O, W_O) = \sum_{j:1\ldots|O|} \Big[W_O(O_j) \sum_{X_i(p_i) \in lin(O_j, I)} W_I(X_i) \cdot \frac{len(p_i)}{nl(X_i)} \Big]$$



$$\sum_{w_i \in W_I} w_i = \sum_{w_j \in W_O} w_j = 1$$

Impact of M-* processors on precision



Count the number of variables in O that can be reached from P

- weighted sum

$$impact(P, O) = \sum_{o \in O} W(o) \cdot reach(P, o)$$

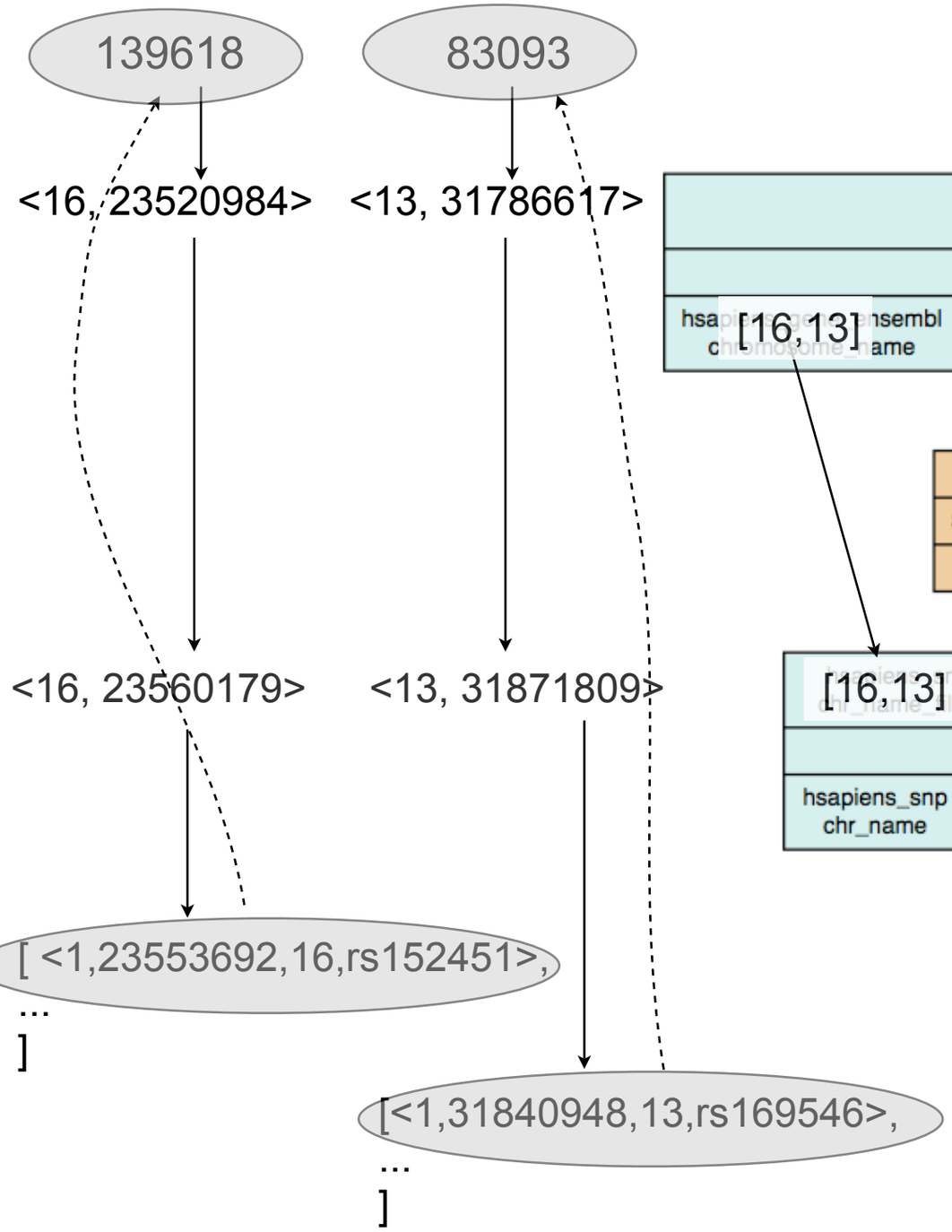
$$reach(P, v) = \begin{cases} 1 & \text{if } v \text{ is reachable from } P \\ 0 & \text{otherwise} \end{cases}$$

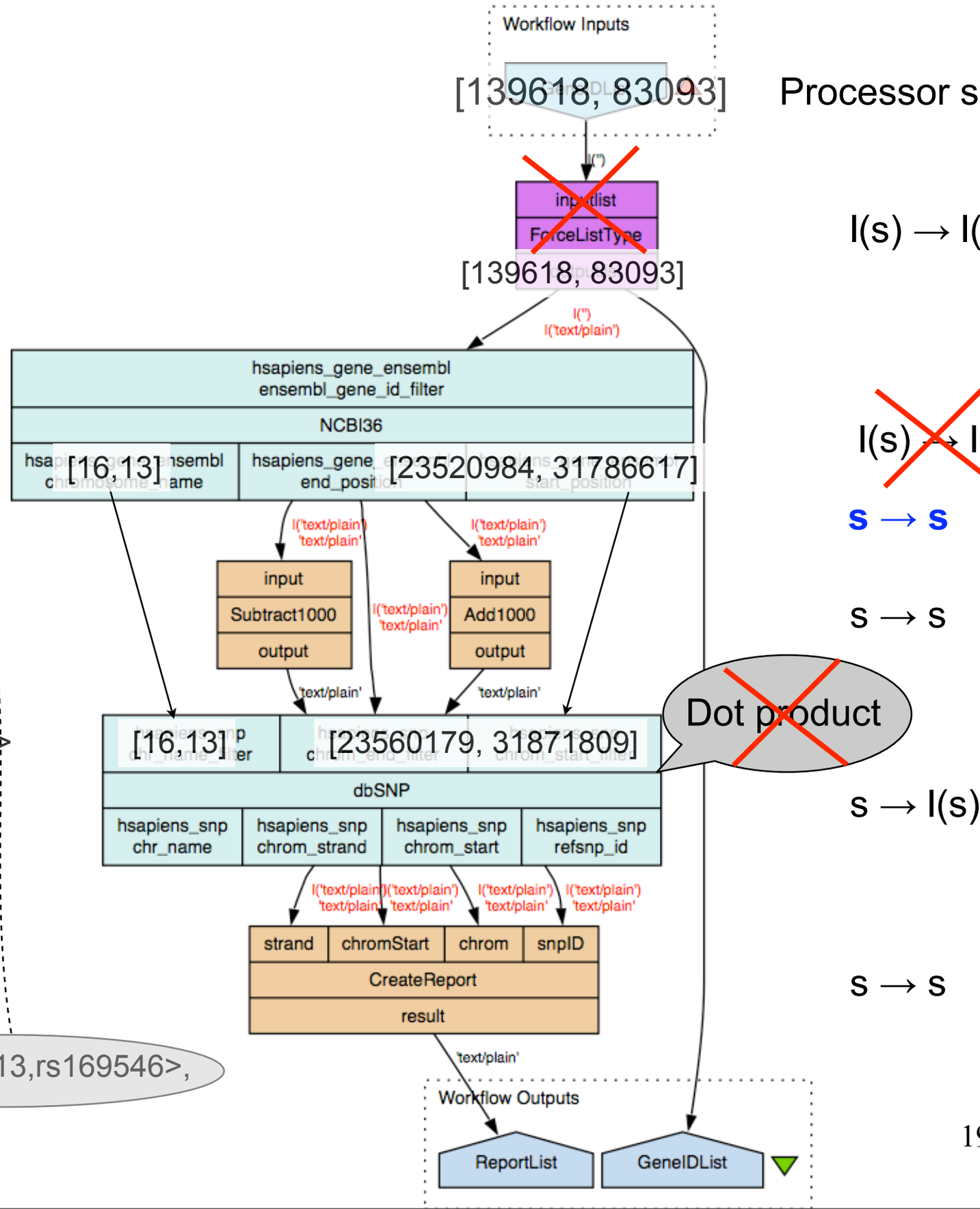
Improving provenance precision

- Impact used to prioritize user actions on processors
- Precision used to assess improvement

- add index-preserving annotations
  - ✓ illustrated earlier
- refactor M-* processors
- make processors provenance-active

Refactoring M-* → 1-1





Processor signatures

l(s) → l(s)

~~l(s) → l(s)~~

s → s

s → s

~~Dot product~~

s → l(s)

s → s

Provenance-active processors

- *Passive* processors do not contribute explicit provenance info
- provenance-active processors actively feed metadata to the lineage service

	X: **I(s)** = [a_1, a_2, a_3] P Y: **s** = b	X: **I(s)** = [a_1, a_2, a_3] P Y: **I(s)** = [b_1, b_2]
Static annotations:	aggregation f()	P is index-preserving
Dynamic annotations:	b = X[i] b = f(X[1]...X[k])	sorting: Y = Π(X)

Open Provenance Model

- A graph notation to represent process provenance
  - independent of the provenance producers
  - suitable for exchanging provenance across different workflow systems
- State: draft 1.01 (July 2008)

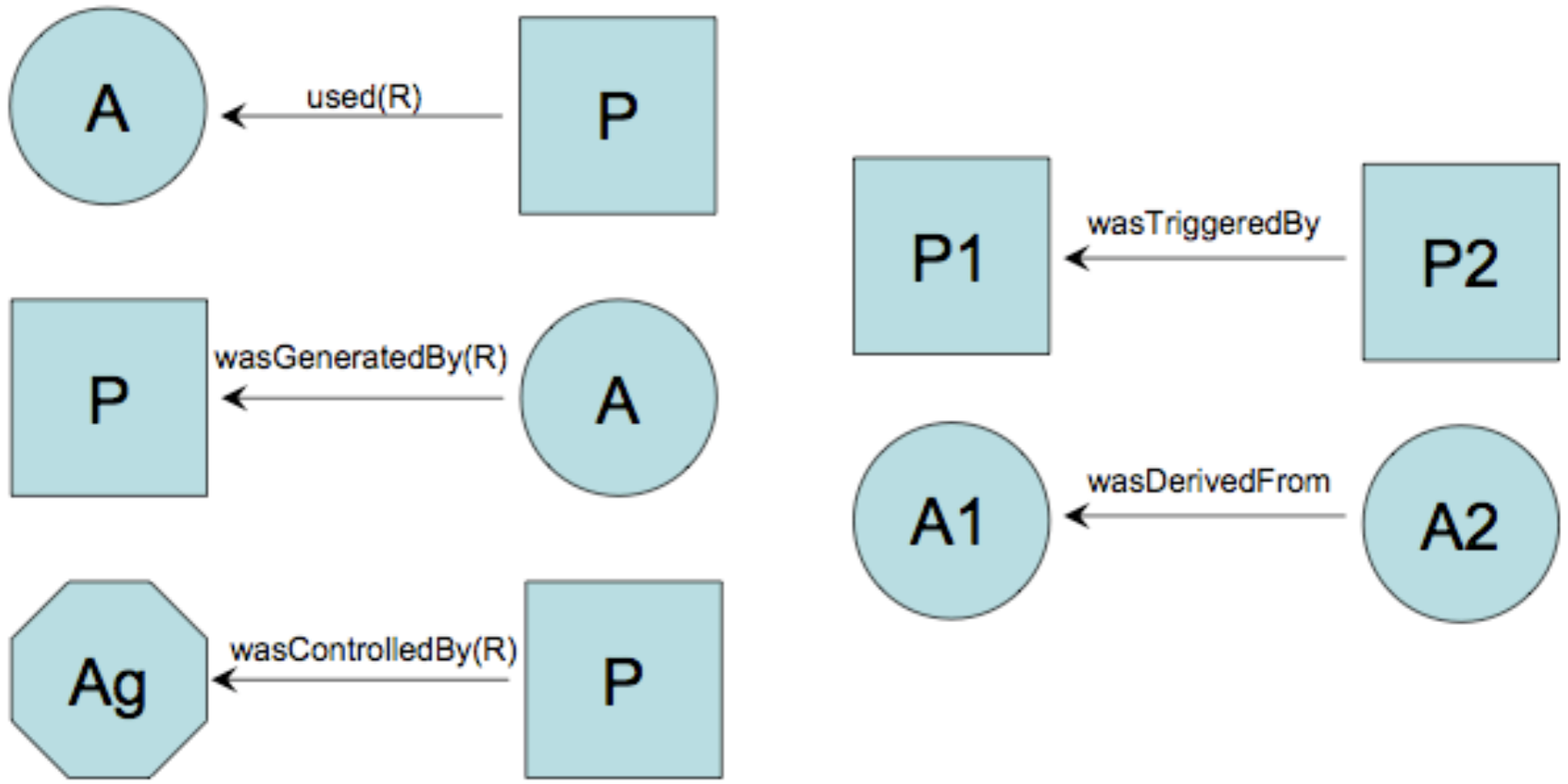
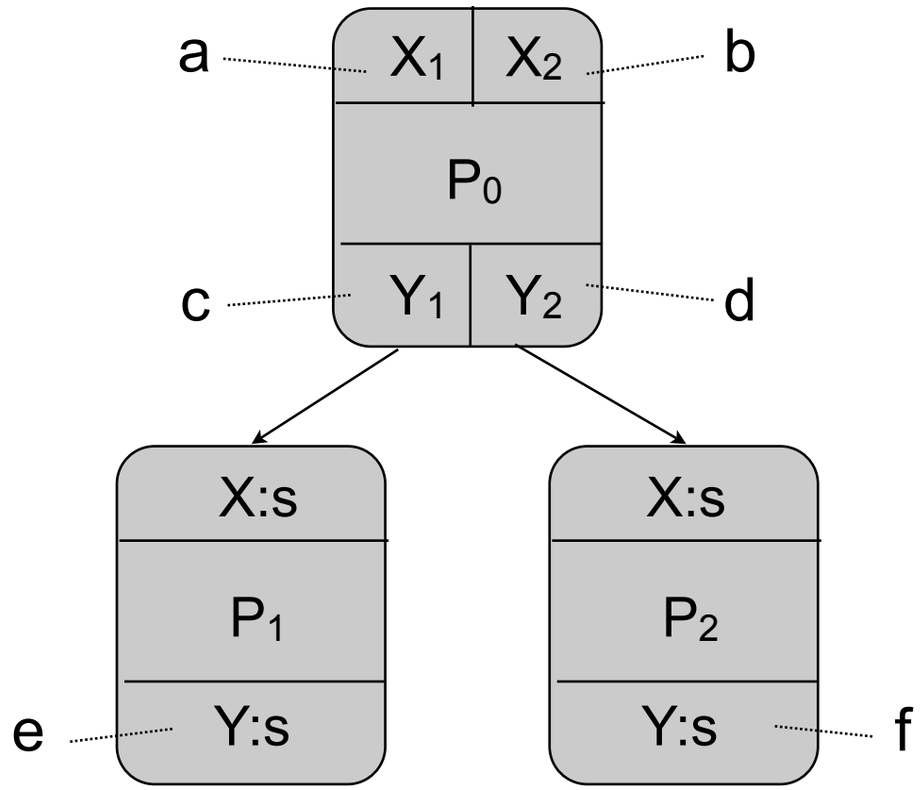
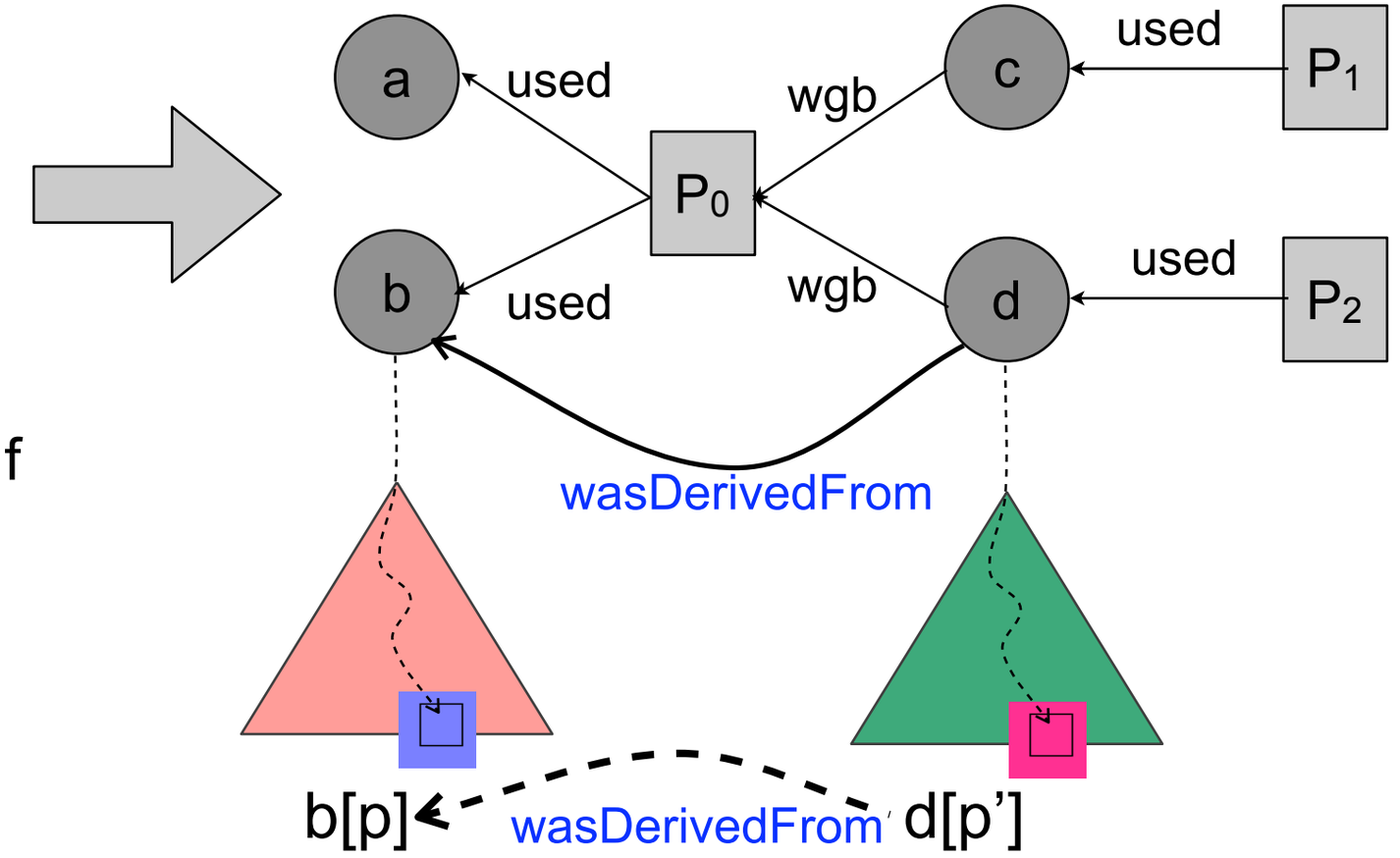


Figure 1: Edges in the Provenance Model

Mapping to OPM - granularity issue





How can this granular dependency be described for all arbitrary paths **p**?

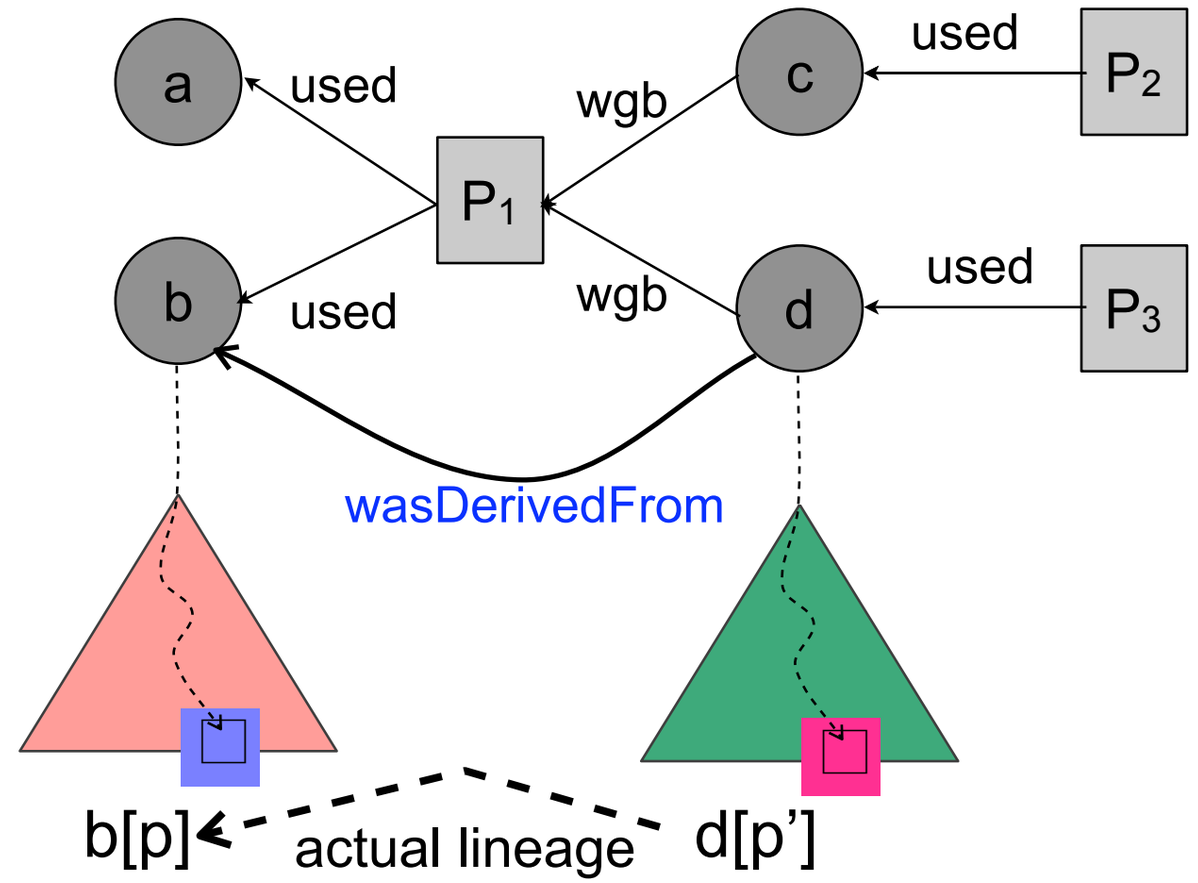
Currently cannot be expressed using OPM

Path mapping rules

Static graph structure sufficient to provide this (in Taverna)

But this is only known at query time

(extensional enumeration not an option)

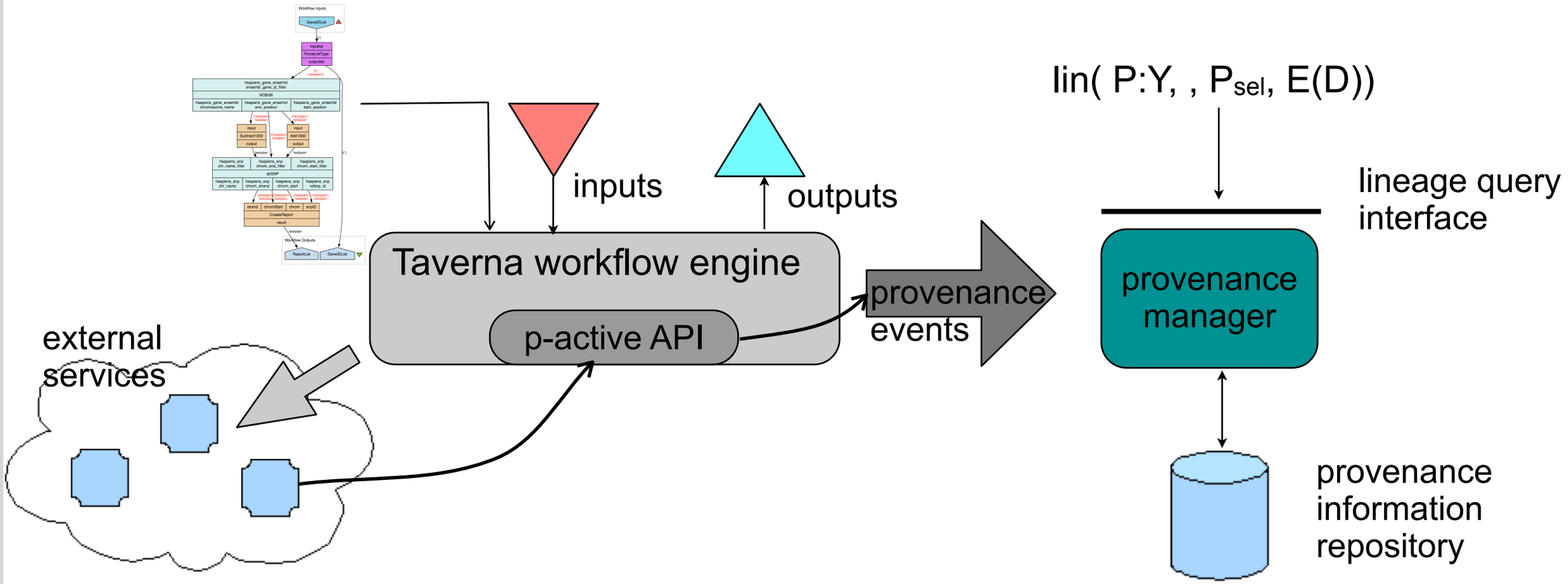


Observation:

- only need to consider individual processor transformations
- exploit local processor rules for propagating granular lineage

Hint:
granularity is only determined by *depth* of the path
At query time, the Taverna lineage query algorithm encodes a path mapping rule to compute **p'** given **p**

Architecture provenance-active processors



1. Common content:

- processor execution details
- binding of input/output variables to values
- completion status

2. Optional content for provenance-active processors:

- explicit output → input dependency assertions:

  let I, O be the input, resp. output variables set

  depends(Y, X[p], <depType>), $X \in I$, $Y \in O$

Ongoing work

- Experimental evaluation:
  - to what extent is granularity a real practical problem?
  - Quantify provenance friendliness by analysing a large collection of workflows from myExperiment
  - Quantify available improvements (i.e. by refactoring)

- Compare collection management in Taverna with other workflow models
  - can we sucessfully exchange provenance graphs?

- Integration of the provenance service with the new version of Taverna
  - to be released before end of year