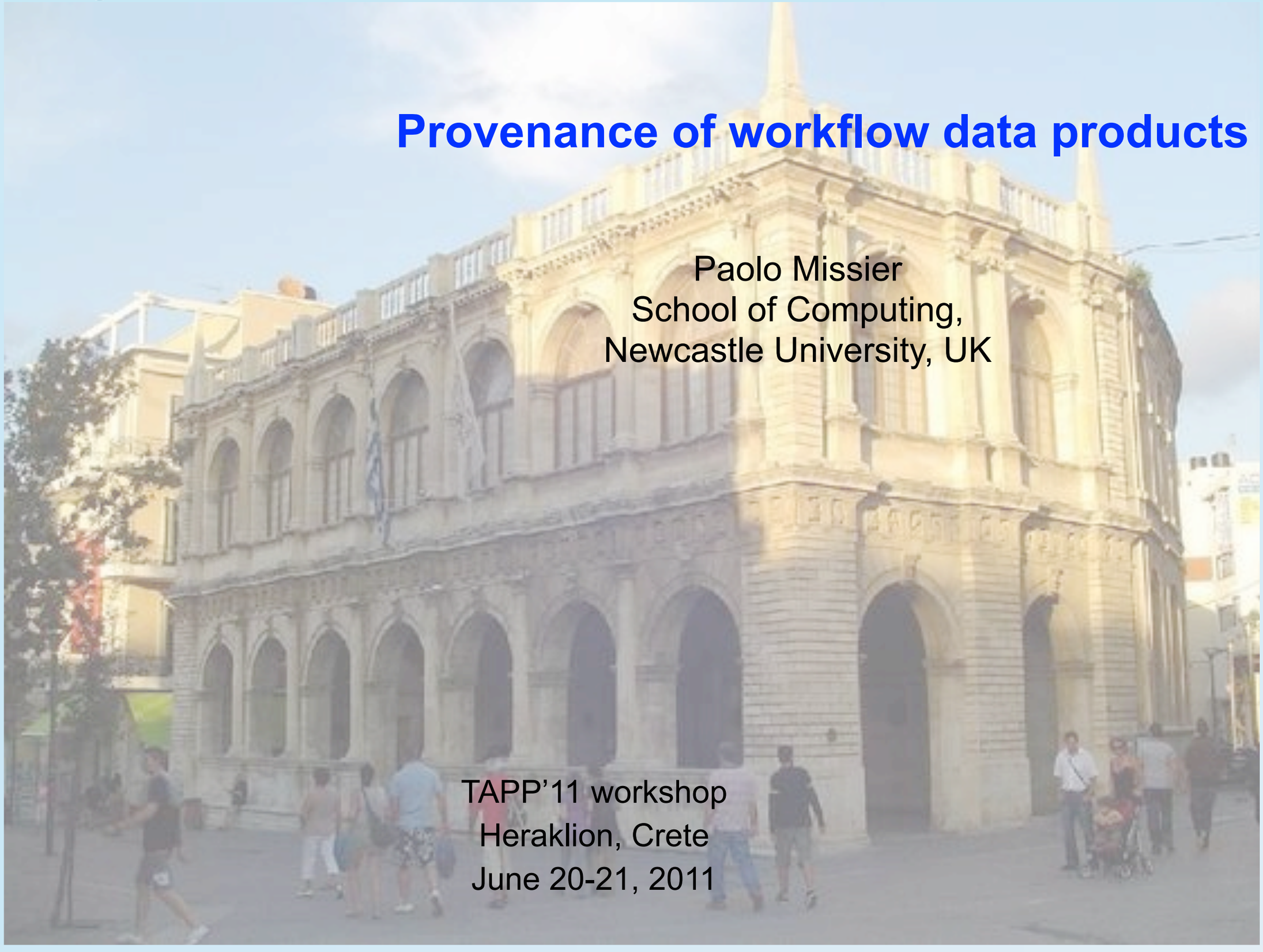
Newcastle University

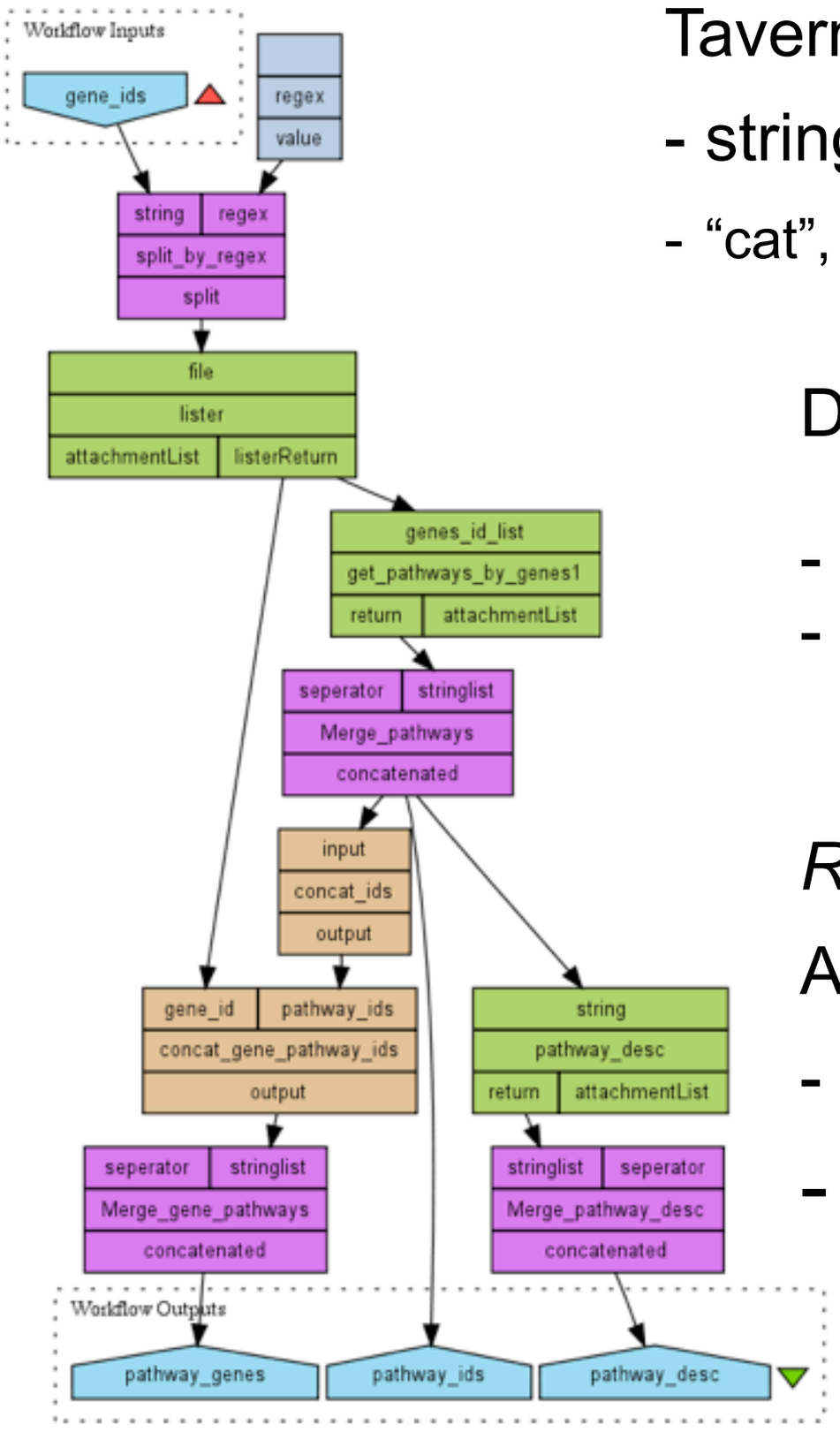
Provenance of workflow data products

Paolo Missier
School of Computing,
Newcastle University, UK



TAPP'11 workshop
Heraklion, Crete
June 20-21, 2011

Workflow provenance



Taverna type system:

- strings + nested lists

- “cat”, [“cat”, “dog”], [[“cat”, “dog”], [“large”, “small”]]

Dataflow model:

- data-driven execution
- services activate when input is ready

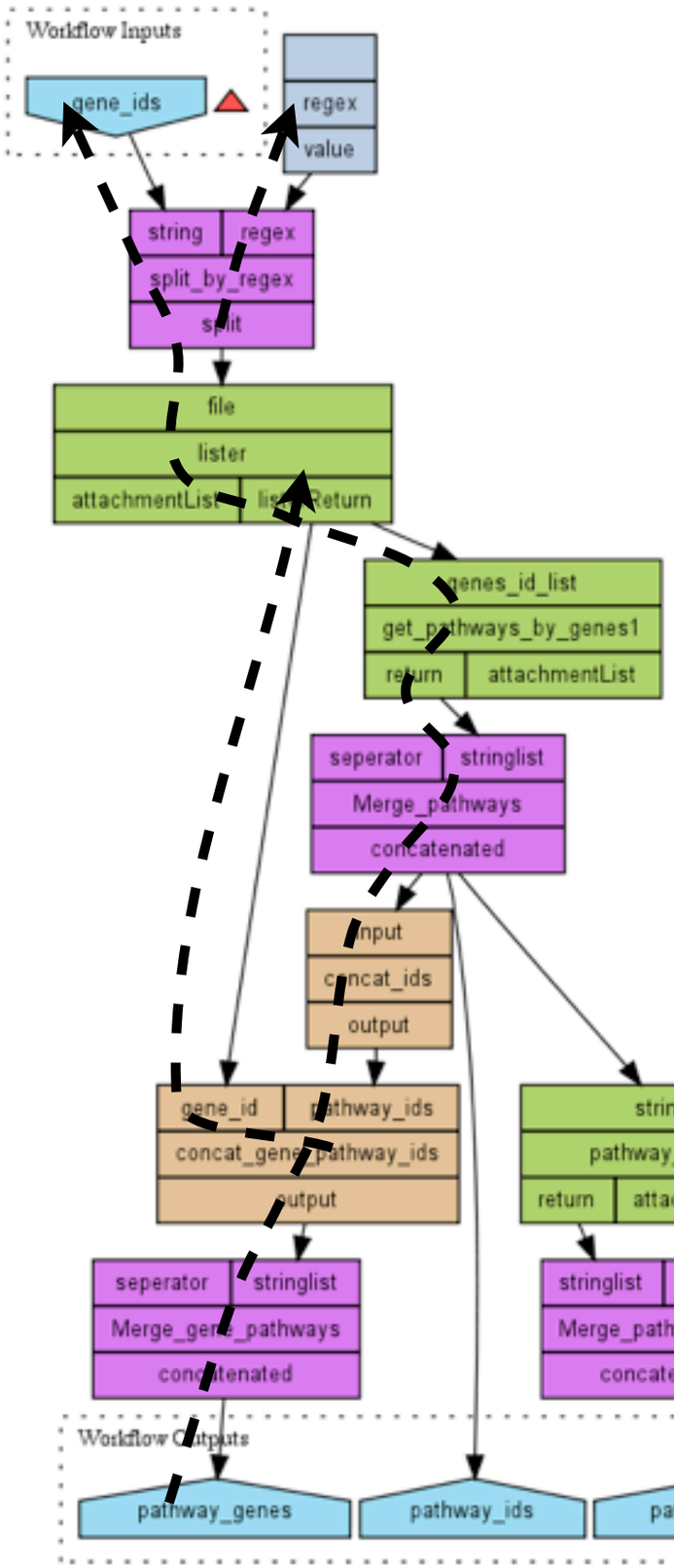
Raw provenance:

A detailed trace of workflow execution

- tasks performed, data transformations

- inputs used, outputs produced

Workflow provenance



Taverna type system:

- strings + nested lists

- “cat”, [“cat”, “dog”], [[“cat”, “dog”], [“large”, “small”]]

Dataflow model:

- data-driven execution
- services activate when input is ready

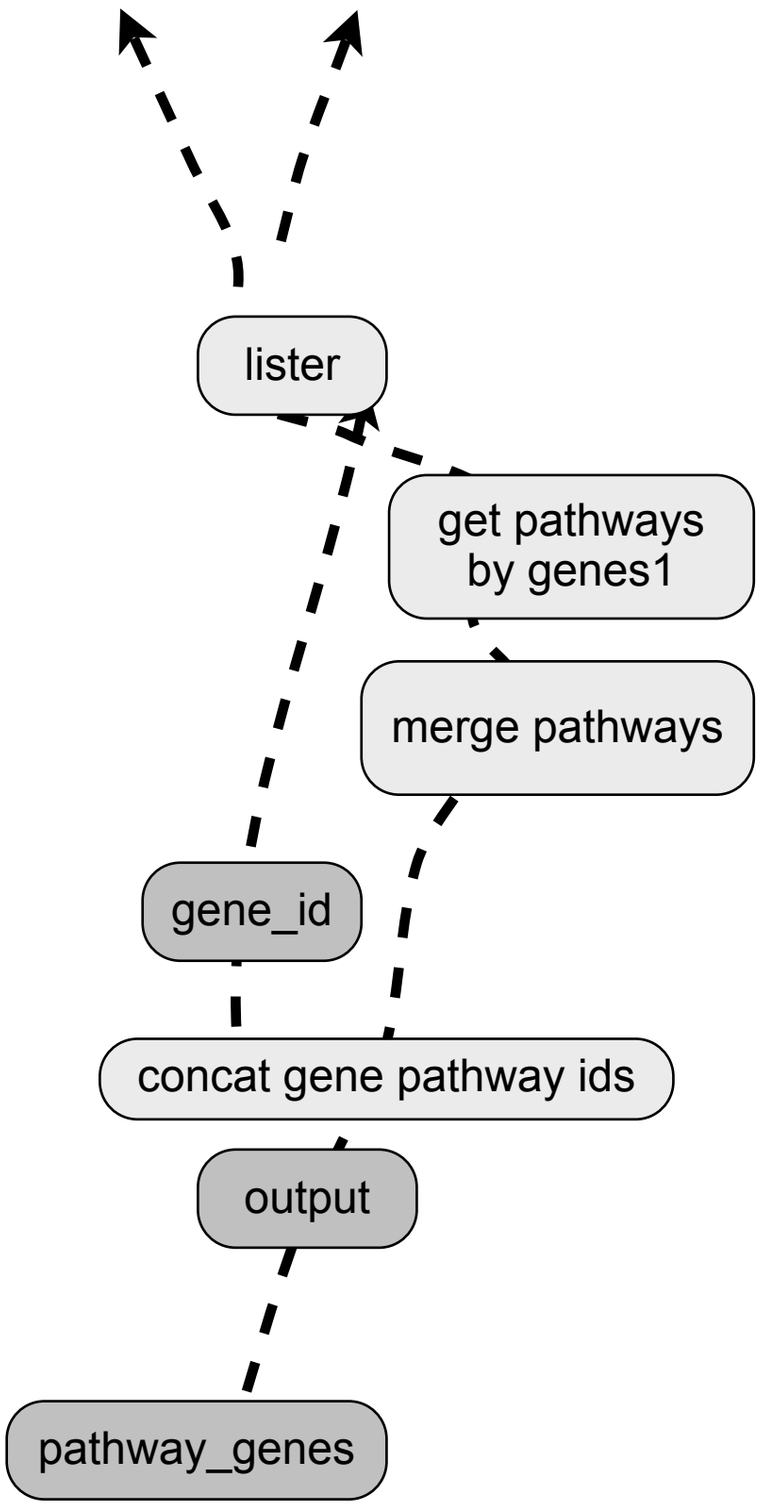
Raw provenance:

A detailed trace of workflow execution

- tasks performed, data transformations

- inputs used, outputs produced

Workflow provenance



Taverna type system:

- strings + nested lists

- “cat”, [“cat”, “dog”], [[“cat”, “dog”], [“large”, “small”]]

Dataflow model:

- data-driven execution
- services activate when input is ready

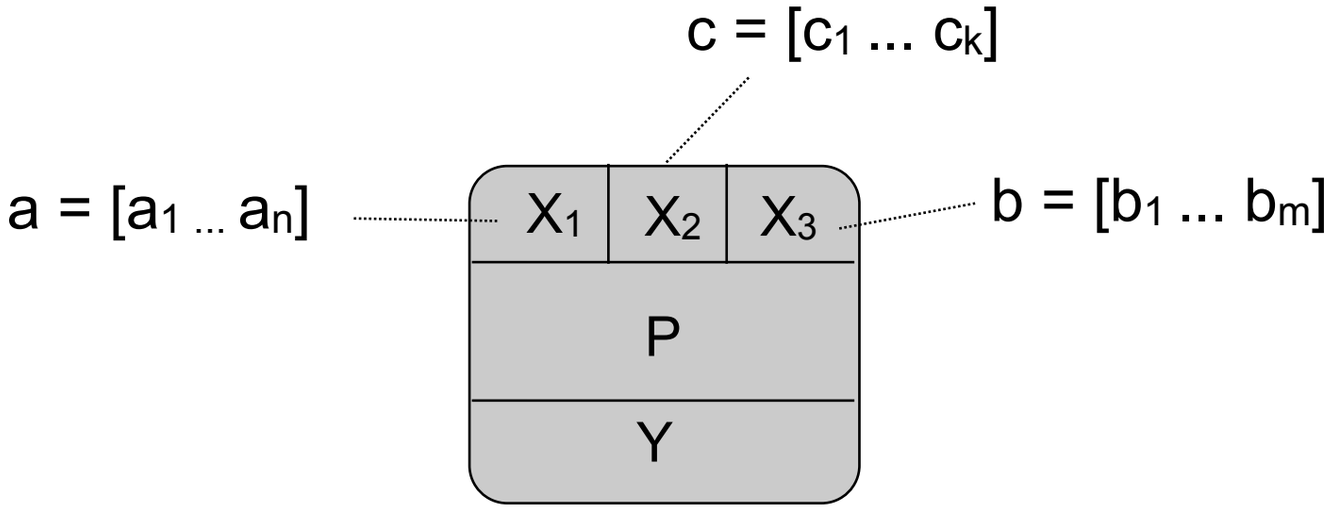
Raw provenance:

A detailed trace of workflow execution

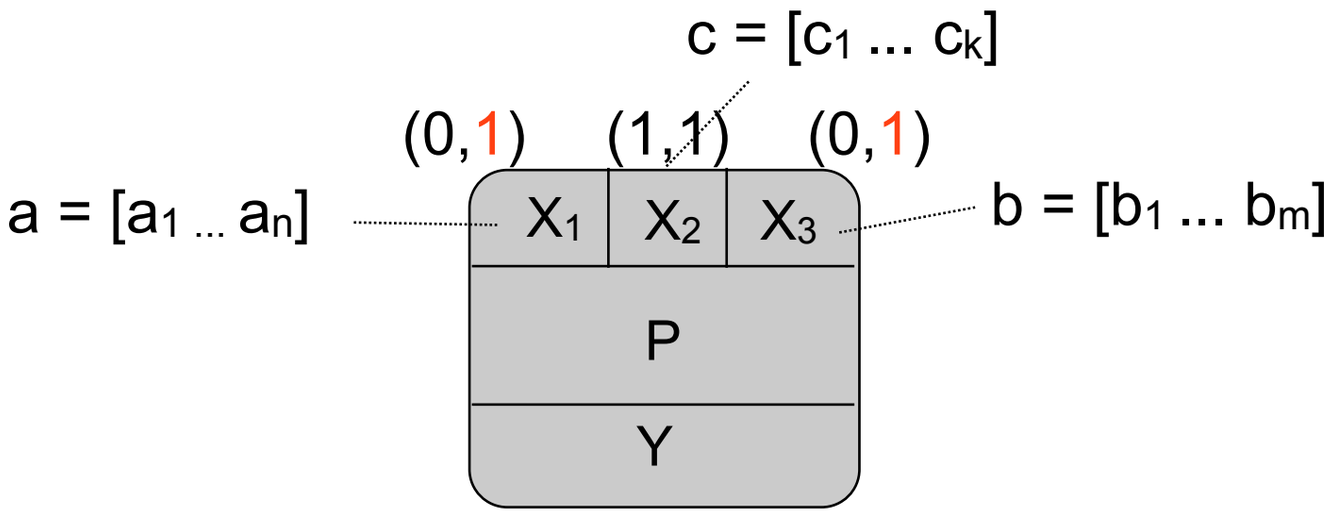
- tasks performed, data transformations

- inputs used, outputs produced

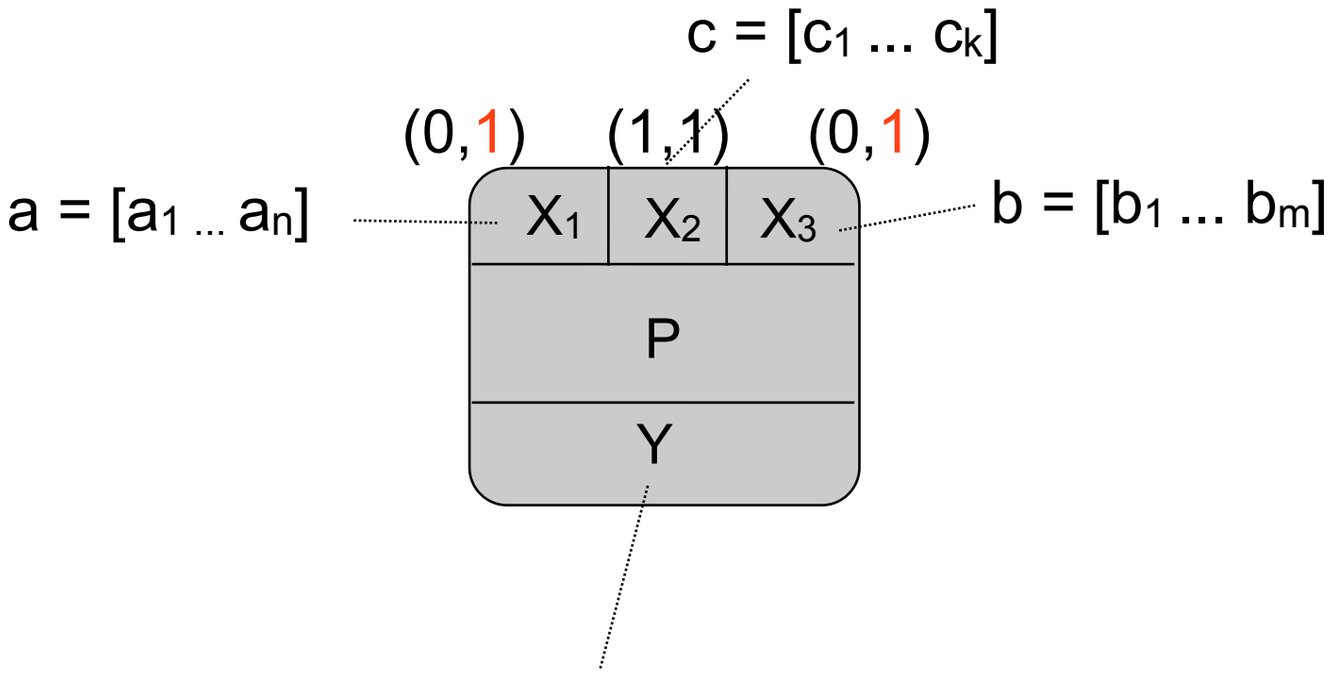
Implicit iteration in Taverna



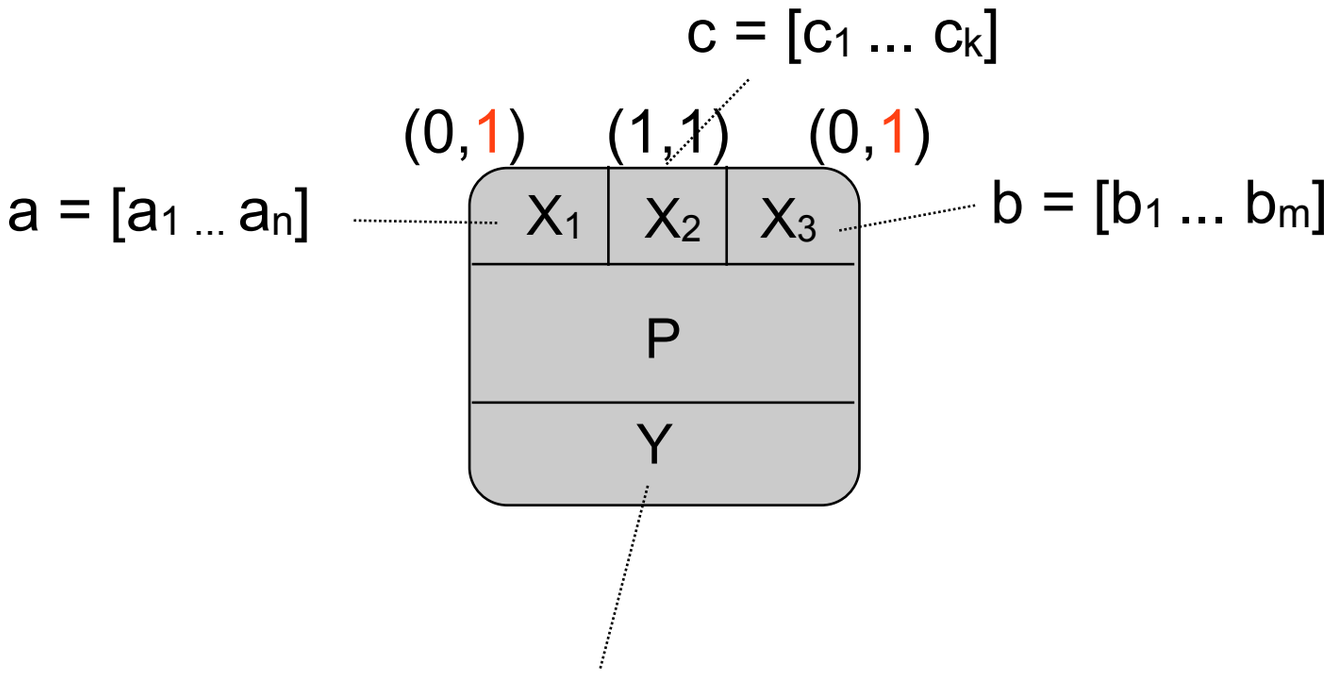
Implicit iteration in Taverna



Implicit iteration in Taverna



Implicit iteration in Taverna



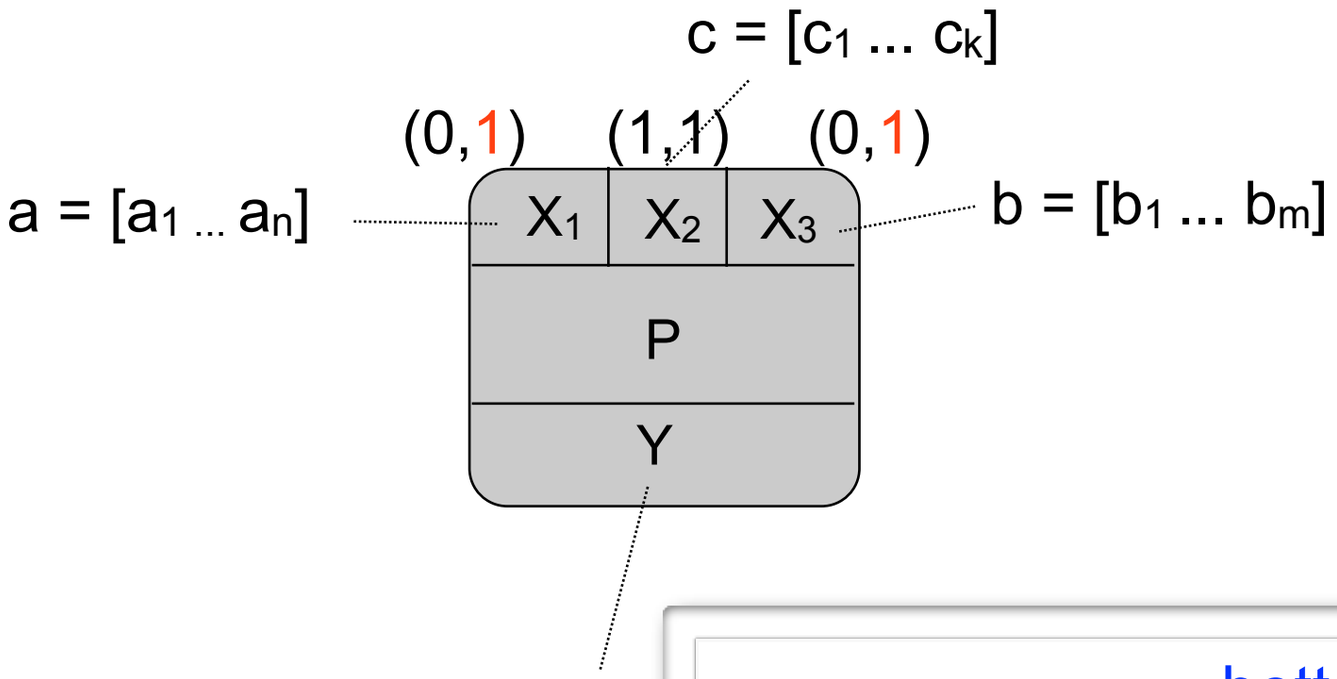
How *y* is computed at P:

let I = a ⊗ b = [[<a_i, b_j> | $b_j \in b$] | $a_i \in a$] // cross product

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

y = (**map** (**map** P) I') = [(map P [<a_1,c, b_1> ... <a_1,c, b_m>]), ...,
(map P [<a_n,c, b_1> ... <a_n,c, b_m>])] =
[[y_{11} ... y_{1n}], ... [y_{n1} ... y_{nm}]]

Implicit iteration in Taverna



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

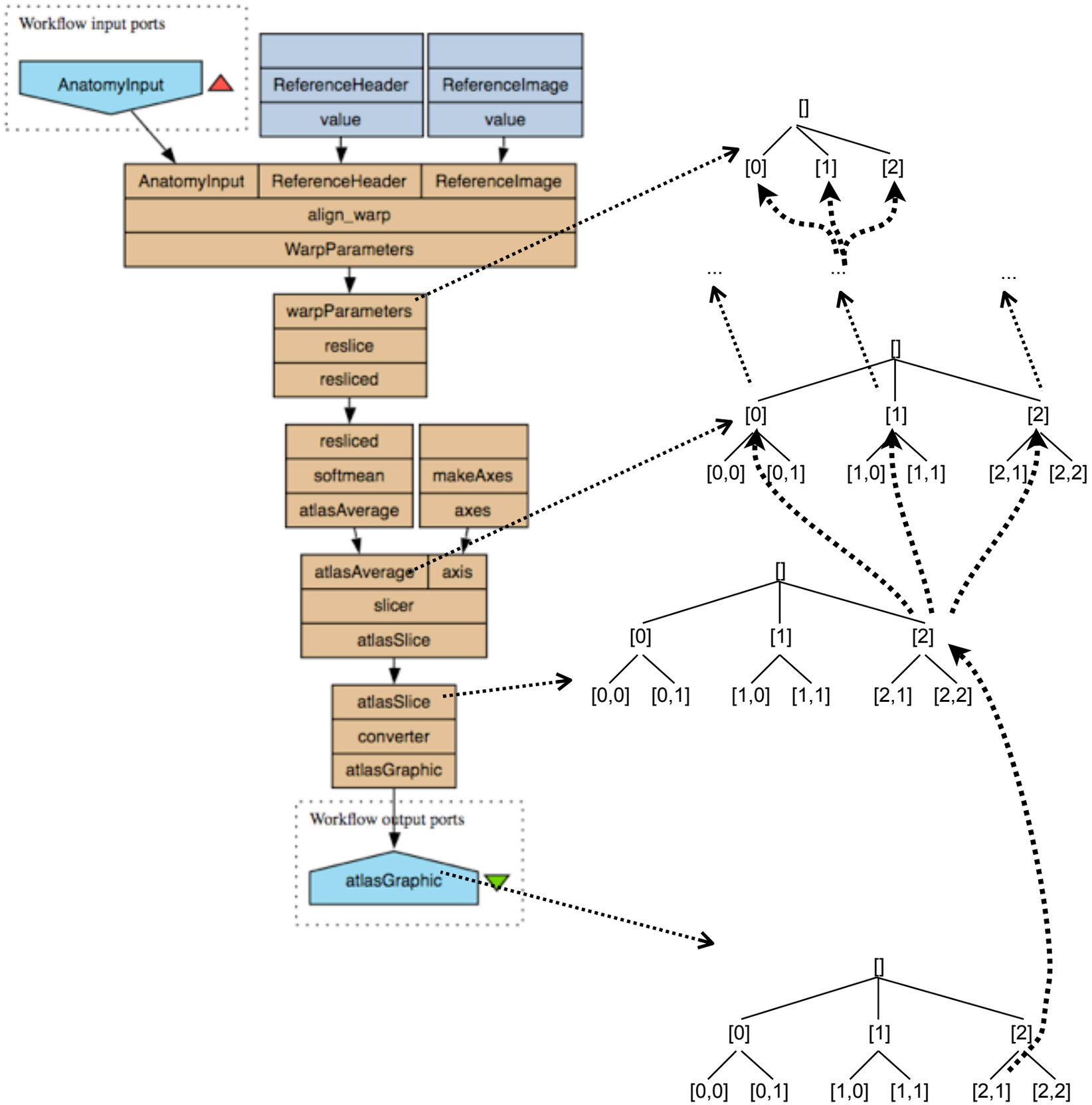
How *y* is computed at P:

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

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

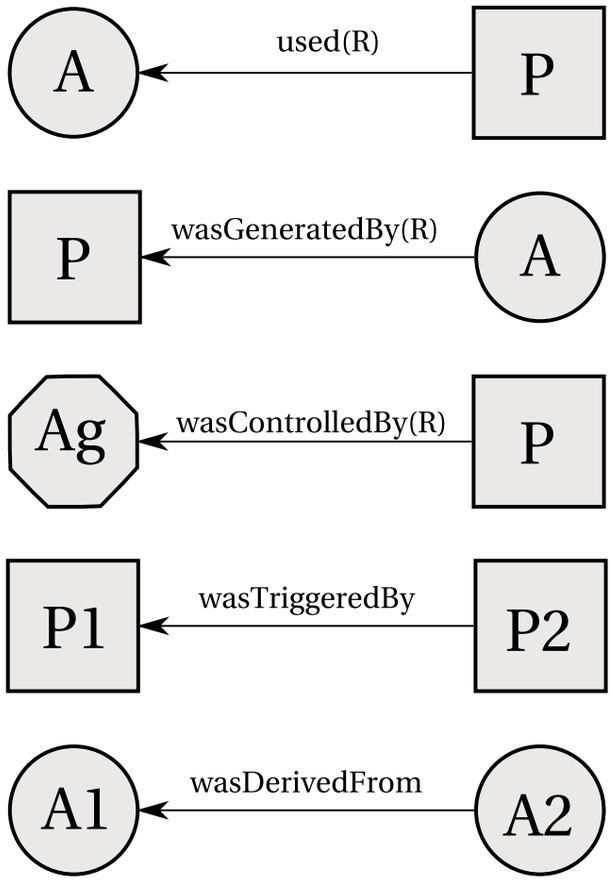
y = (**map** (**map** P) I') = [(map P [$<a_1,c, b_1>$ ... $<a_1,c, b_m>$]), ...,
(map P [$<a_n,c, b_1>$ ... $<a_n,c, b_m>$])] =
[[y_{11} ... y_{1n}], ... [y_{n1} ... y_{nm}]]

Fine-grained (precise?) provenance



Which provenance model/language?

- Let's look at the Open Provenance Model as a starting point



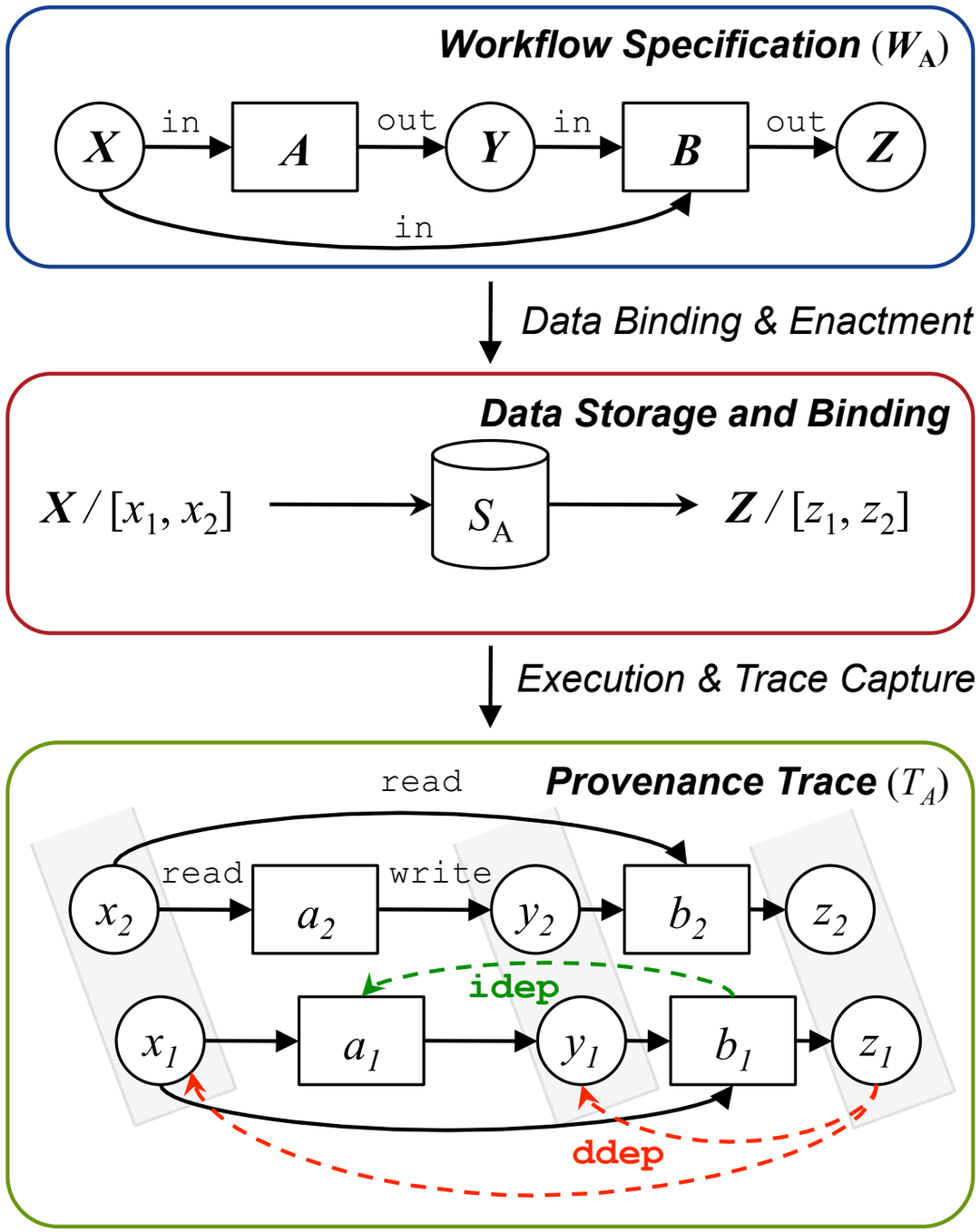
Core OPM

- **agnostic wrt Artifact, Processor types**
- roles: annotations on binary relations
- extensions by subclassing
  - node types,
  - relation types

Formal (temporal) semantics hopefully available soon

OPM relations in the workflow context

Single User's View (Alice)



Process space

Data space

Provenance space

ans(X) :- ddep*(X, z_1). ***Provenance Queries***

Missier, P., Ludascher, B., Bowers, S., Anand, M. K., Altintas, I., Dey, S., et al. (2010). Linking Multiple Workflow Provenance Traces for Interoperable Collaborative Science. *Proc.s 5th Workshop on Workflows in Support of Large-Scale Science (WORKS)*.

Data and invocation dependencies

- **`read`**, **`write`** are natural observables for a workflow run
- possible additional relations (recorded or inferred):

- **invocation dependencies**: $a_1 \overset{\texttt{idep}}{\leftarrow\!-} a_2$

Explicit or via: $\texttt{idep}(a_1, a_2) :- \texttt{write}(a_1, d), \texttt{read}(d, a_2)$

"a_2 depends on a_1" because a_1 has written data d, a_2 has read d

- **data dependencies**: $d_1 \overset{\texttt{ddep}}{\leftarrow\!-} d_2$

Explicit or via: $\texttt{ddep}(d_1, d_2) :- \texttt{read}(d_1, a), \texttt{write}(a, d_2)$

"d_2 depends on d_1"
… because some **actor invocation** *a* read d_1 prior to writing d_2

Provenance queries

- Closure queries:
- operate on the transitive closure *ddep** over *ddep*:

$$\texttt{ddep}^*(d_1, d_2) :- \texttt{ddep}(d_1, d_2)$$

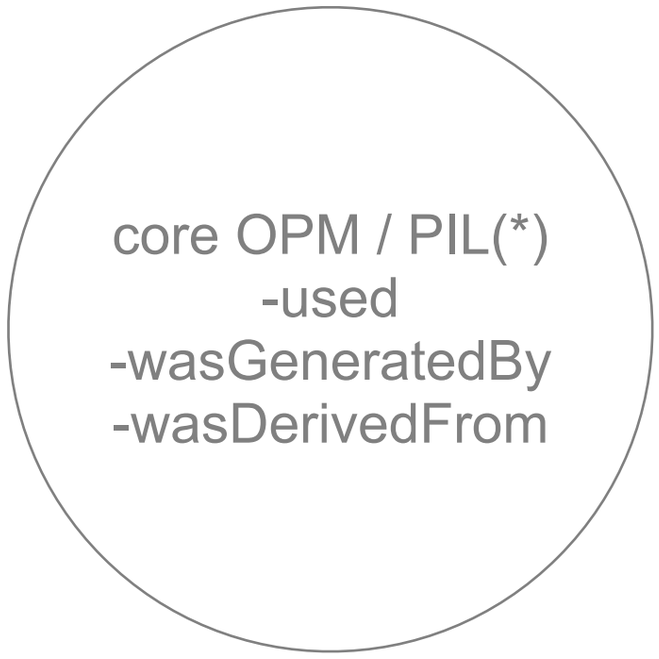
$$\texttt{ddep}^*(d_1, d_2) :- \texttt{ddep}(d_1, d), \texttt{ddep}^*(d, d_2)$$

But also:
- queries on the workflow structure
- queries on the data structures (e.g. collections)

and importantly:
use workflow graphs to justify/explain the provenance graph for one workflow run:

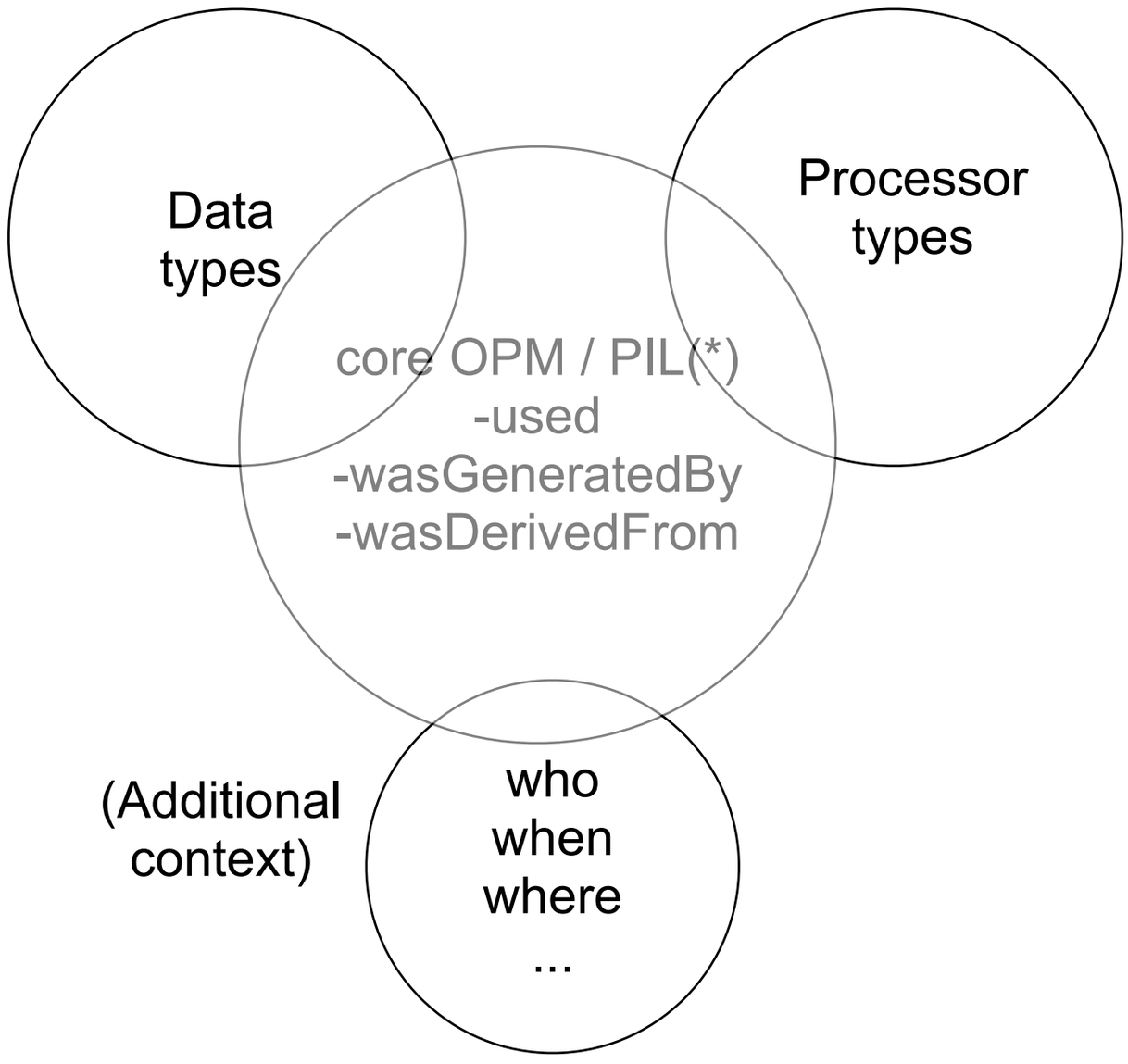
T_A *trace instance* of W_A:
h: $T_A \rightarrow W_A$ *homomorphism*
$h(x_1 \rightarrow a_1) = h(x_2 \rightarrow a_2) = X \rightarrow A$,
$h(a_1 \rightarrow y_1) = h(a_2 \rightarrow y_2) = A \rightarrow Y$
...

OPM extensions, principled



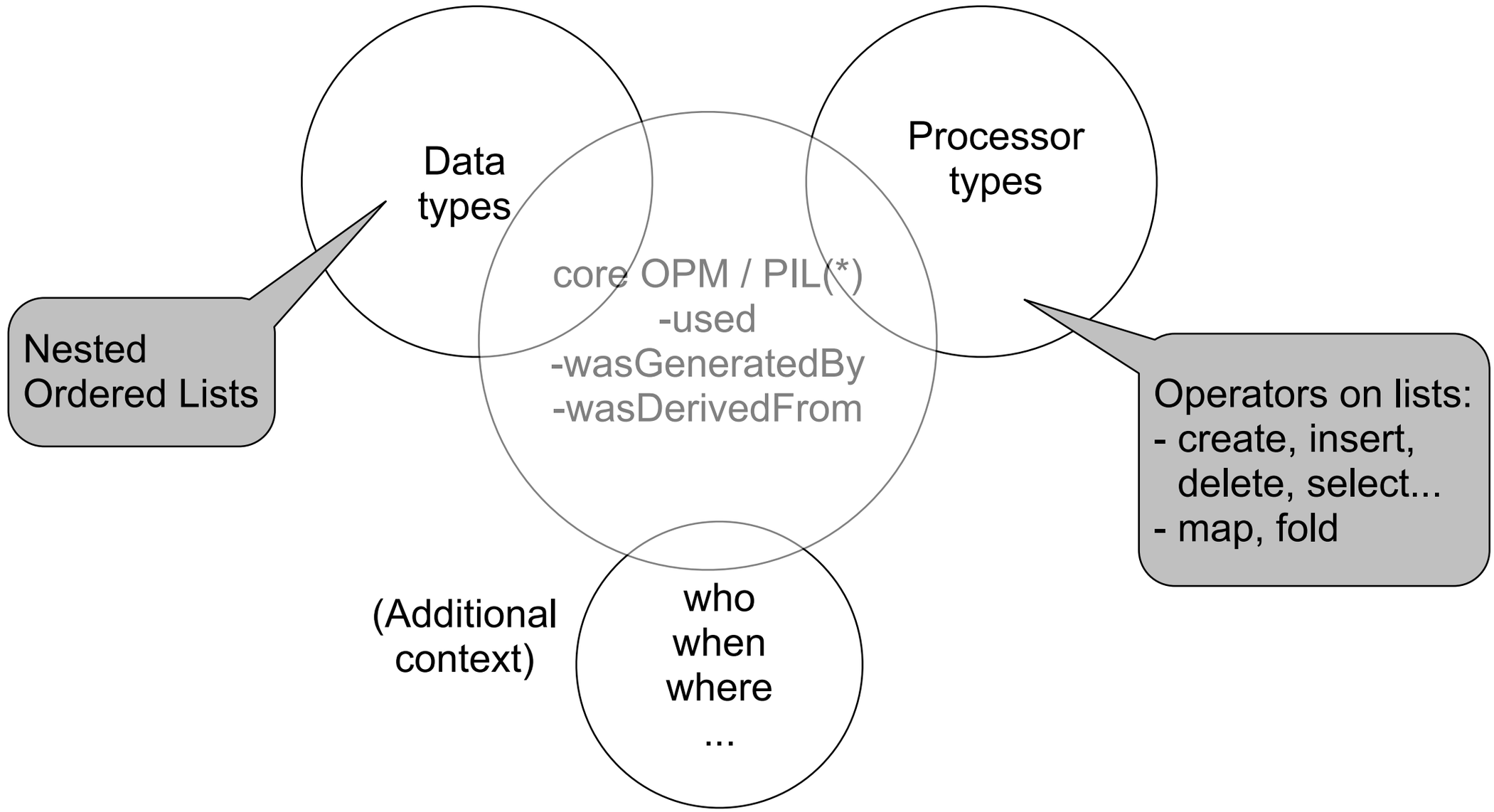
(*) PIL = Provenance Interchange Language, W3C Provenance Working Group

OPM extensions, principled



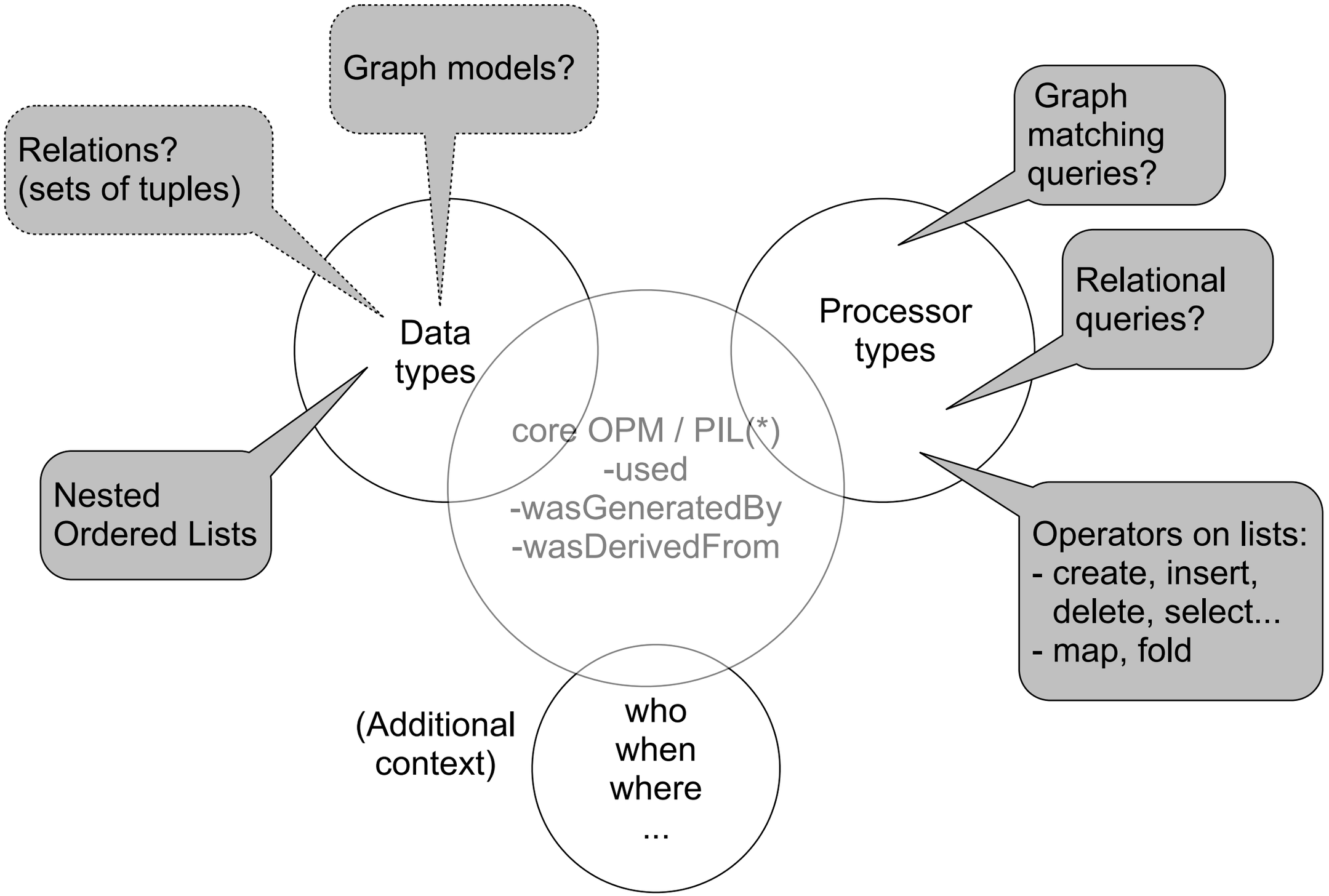
(*) PIL = Provenance Interchange Language, W3C Provenance Working Group

OPM extensions, principled



(*) PIL = Provenance Interchange Language, W3C Provenance Working Group

OPM extensions, principled



(*) PIL = Provenance Interchange Language, W3C Provenance Working Group

Proposed extensions

Role	**used in relation:**	**Context of use**
`element`	*Contained*	L *Contained*(`element`) x
`list`	*Used*	P *Used*(`list`) L
`position`	*Used*	P *Used*(`position`) p

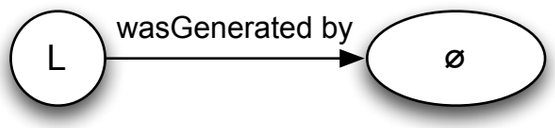
`term`
`generator`
`filter`
	Used	List comprehensions, see Sec. 4.2

`function` `operand`	*Used*	`map`, see Sec. 4.3

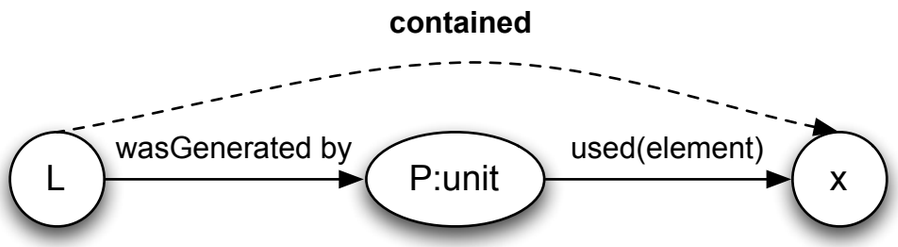
Causal relation	**Example**
$Contained(R) \subseteq [\tau] \times A \times [\texttt{Int}]$	L' *Contained*$([i_1 \ldots i_n])$ x x was inserted into L at position $[i_1 \ldots i_n]$
$wasSelectedFrom(R) \subseteq [\tau] \times [\tau] \times [\texttt{Int}]$	L' *wasSelectedFrom*$([i_1 \ldots i_n])$ L L' at position $[i_1 \ldots i_n]$ was selected from L
$wasRemovedFrom(R) \subseteq [\tau] \times [\tau] \times [\texttt{Int}]$	L' *wasRemovedFrom*$([i_1 \ldots i_n])$ L L' at position $[i_1 \ldots i_n]$ was deleted from L
$wasSameAs \subseteq A \times A$	L *wasSameAs* x inferred (various contexts)

OPM fragments for elementary operations

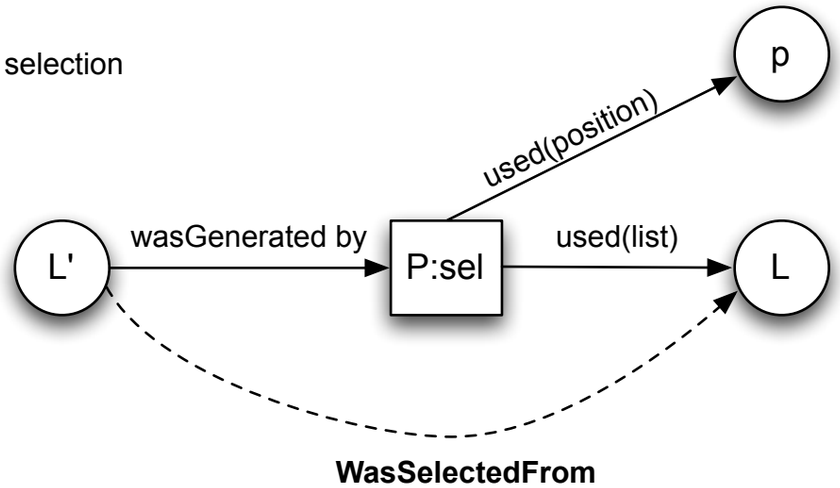
empty list



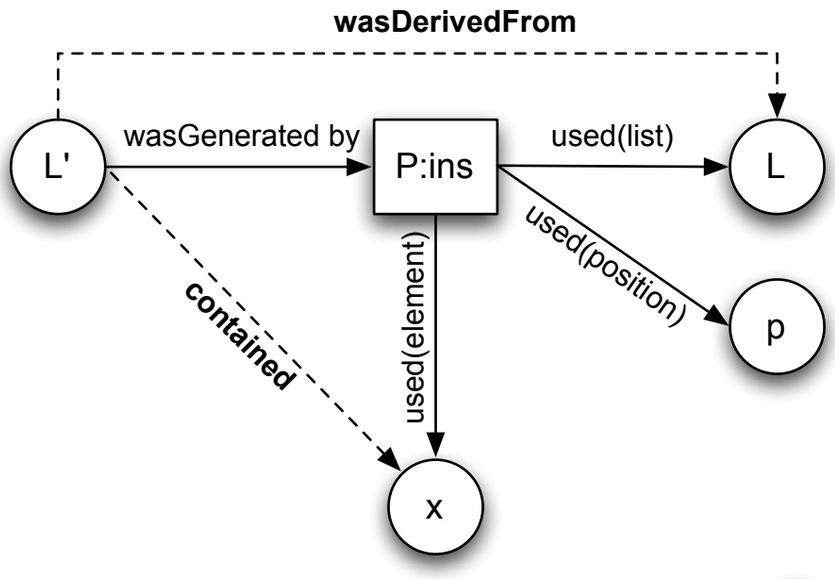
unit



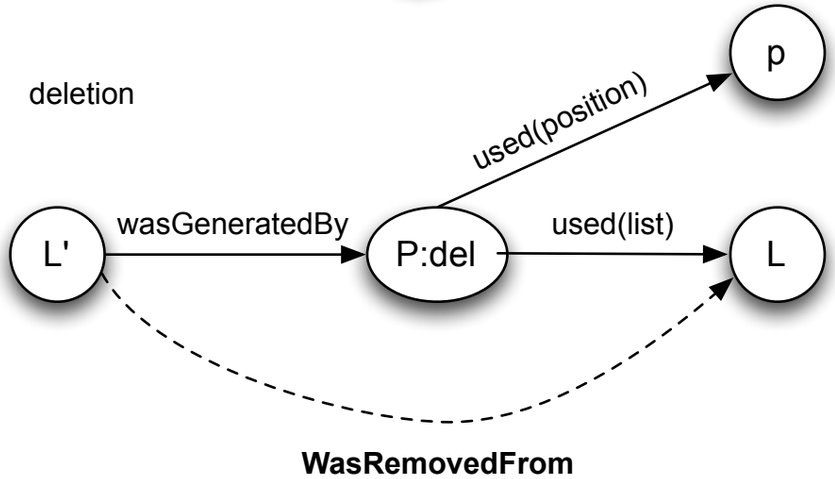
selection



insertion



deletion



Operator composition → graph composition

- Equalities on operators may translate into inferences on the graphs

$$sel(ins\ x\ L\ p)\ p = x$$

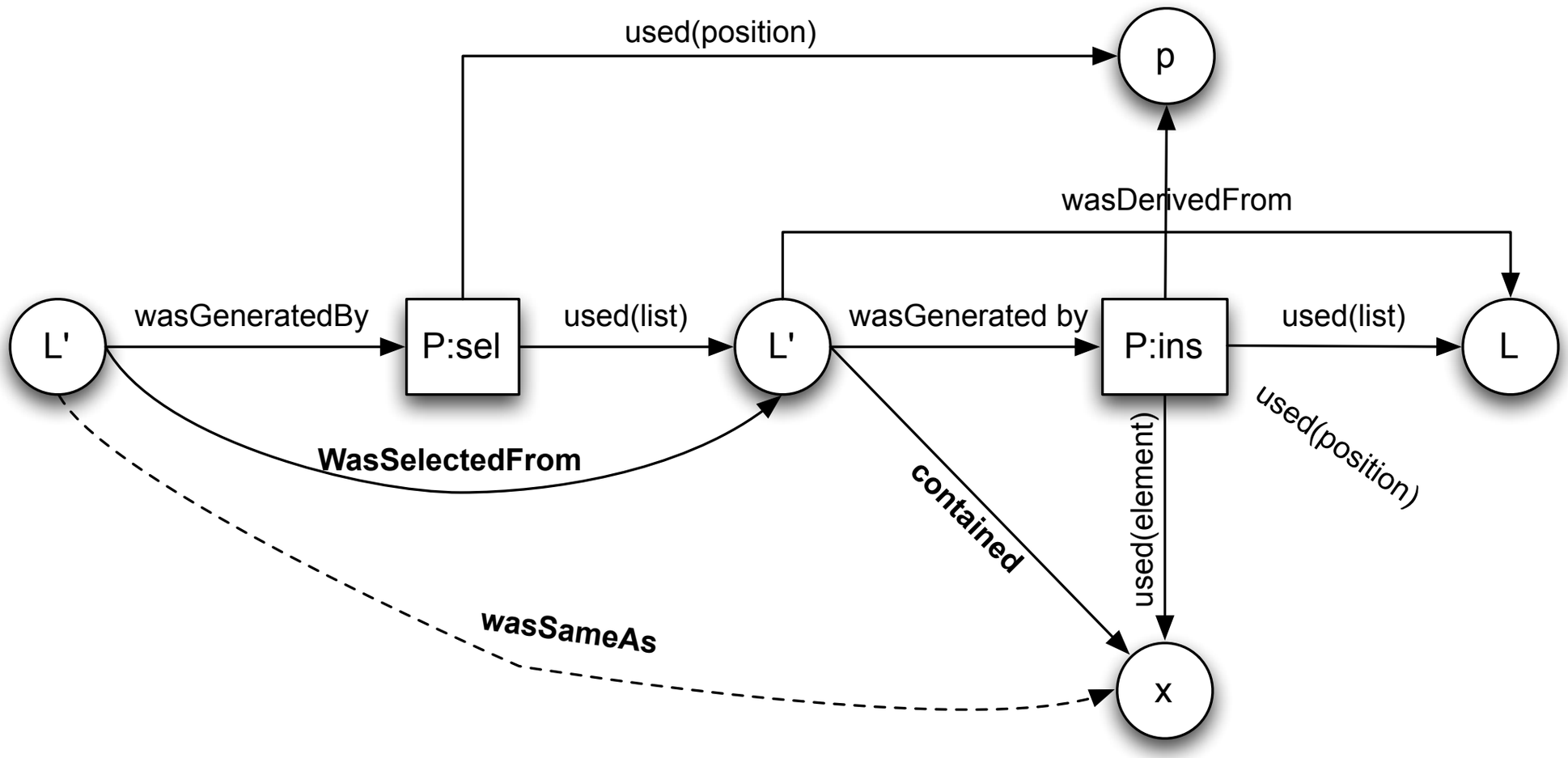
sameAs(L1,L) :-
pType(P1, ins), *used*(P1, X, element), *used*(P1, Pos, position), *wgby*(L,P1),
pType(P2, sel), *used*(P2, L, list), *used*(P2, Pos, position), *wgby*(L1,P2).

Operator composition → graph composition

- Equalities on operators may translate into inferences on the graphs

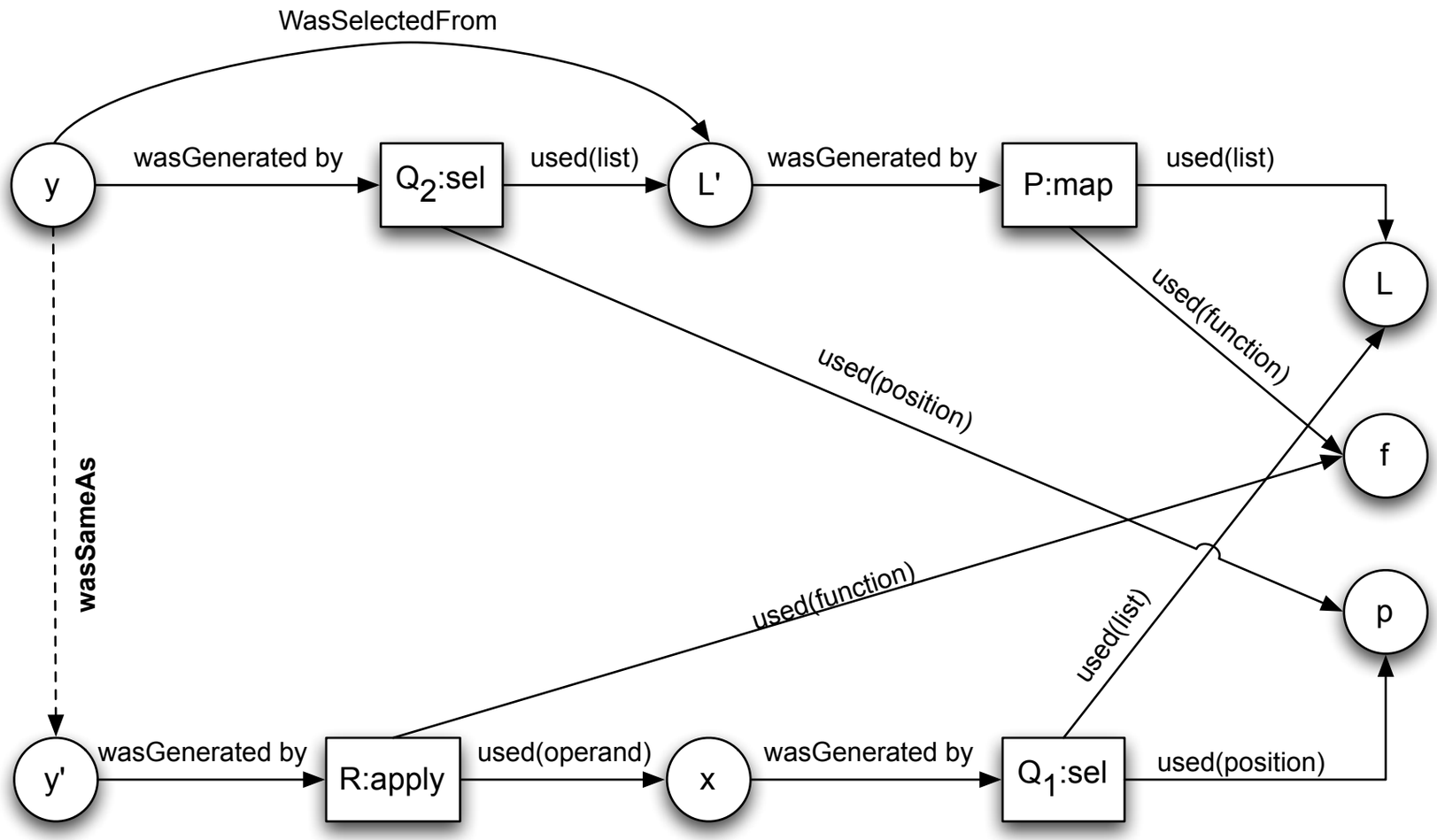
$$sel(ins\ x\ L\ p)\ p = x$$

sameAs(L1,L) :-
pType(P1, ins), *used*(P1, X, element), *used*(P1, Pos, position), *wgby*(L,P1),
pType(P2, sel), *used*(P2, L, list), *used*(P2, Pos, position), *wgby*(L1,P2).



Approach applies to map, fold (reduce)...

$$sel\ (map\ f\ x)\ p = f\ (sel\ x\ p)$$



Take-home message

- OPM (PIL) a candidate starting point for workflow-based provenance
- extension mechanisms are provided, but they must be used sensibly
  - data types
  - processor types
- Provenance of nested ordered lists used as a prototypical example
  - semantics of provenance graphs and graph composition grounded in the semantics of lists
- Can this approach be useful for other interesting data types?
  - sets of tuples / relational algebra