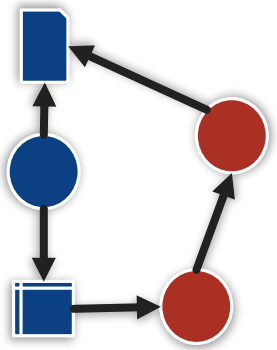


Mappings, Updates, and *Provenance*

Zachary G. Ives  University of Pennsylvania

with Z. Yan, P. Talukdar, T. Green, G. Karvounarakis, N. Taylor, V. Tannen

Funded in part by NSF IIS-1217798, NSF ACI-1547360, NIH 5U24NS063930, NIH 1U01EB020954, and gifts from Google

BX 2019

June 4, 2019

Mapping Data → Bidirectional Updates

File synchronization, across formats and physical values

CSV → Excel or JSON BibTeX → CFF or RIS XML → XSLT → document

Harmony (Foster, Pierce, et al), Boomerang (Bohannon, Pierce, et al), (Hu et al)

Views oriented around a particular perspective

Rollup views in a data warehouse

Access control: secure views of (subsets of) data

Early versions of SalesForce

Data exchange and collaborative data sharing

- IBM Clio project (Haas, Fagin, Tan, et al); Microsoft Model Management (Bernstein)
- Orchestra (Green, Karvounarakis); Youtopia (Kot, Koch); Dejima (Hu et al)

A Running Example: Universities, Insurance, Publications

University

Student

sid	first	las
123	Maya	Sh
456	Alex	Sm

Member

Club

How do we map overlapping data among these sites?

How do we allow users to modify data at any site, and propagate to the rest?

ID Map

sid	id
123	3333
456	4444

Student Insurance

Clients

id	first	last
3333	Maya	Shah
4444	Alex	Smith

Coverages

id	coverage
3333	dental
4444	life
3333	vision

dex

Publications

sid	venue
123	NeurIPS
456	POPL
456	VLDB

OtherNames

venue	aka
VLDB	PVLDB

Challenge I: Attributes in Only One Schema

Student

sid	first	last	age
123	Maya	Shah	22
456	Alex	Smith	33
?	Jun	Li	?′
789	Jill	West	19

ID Map

sid	id
123	3333
456	4444
?	5555
789	?″

Student Insurance

id	first	last	coverage
3333	Maya	Shah	dental
4444	Alex	Smith	life
5555	Jun	Li	vision
?″	Jill	West	?‴

Update translation must create "fresh" values for unmapped attributes!

Challenge II: Shared Sources and Side Effects

Student

sid	first	last	age
123	Maya	Shah	22
456	Alex	Smith	33
789	Jill	West	19

Member

sid	cid
123	1
456	2
789	1

Club

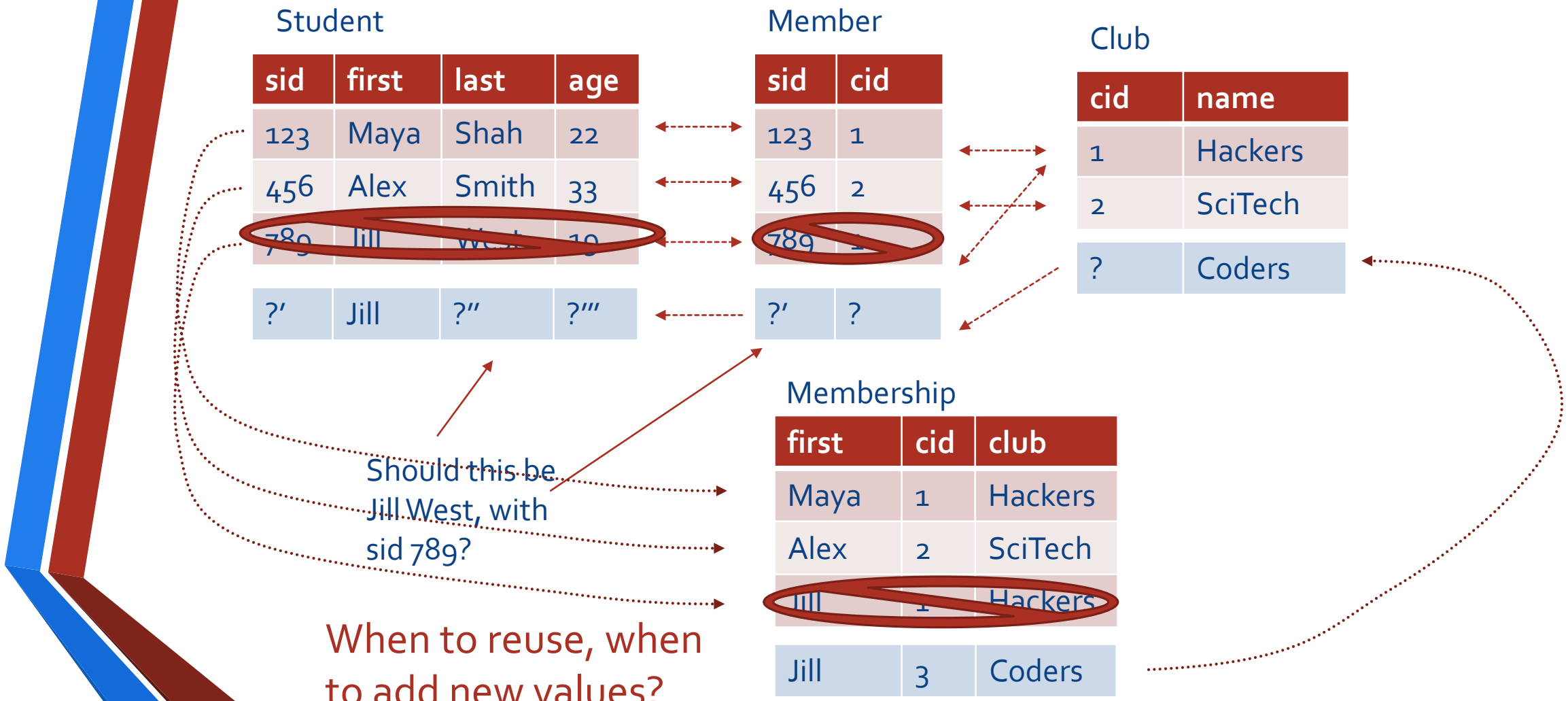
cid	name
1	~~Hackers~~ Coders
2	SciTech

Membership

first	cid	club
Maya	1	~~Hackers~~ Coders
Alex	2	SciTech
Jill	1	~~Hackers~~ Coders

Danger: "Side effects" (on the same view, on other views!)

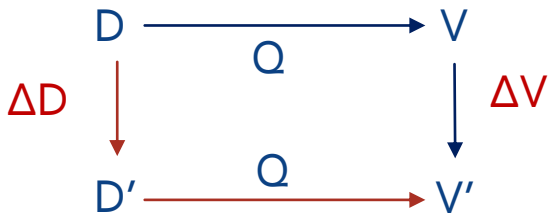
Challenge III: Non-Unique Translations



Road Map

- **Foundational approaches**
  - **Database views**
  - **Bidirectional programming languages**
  - **Data exchange and constraint-based mappings**
- Pushing the envelope: collaborative data sharing
- The challenges of bidirectionality
- The need for provenance
- Looking beyond

Updatable Database Views



Assume DB instance D conforming to schema Σ, query Q over Σ, and view instance V = Q(D)

Update translation: Given an update ΔV to V resulting in V′,

Find an update ΔD to D, such that for the resulting instance D′, Q(D′) = V′

[Keller 85] explores alternatives; generally we assume deletes over V should translate to deletes over D, inserts over V translate to inserts over D

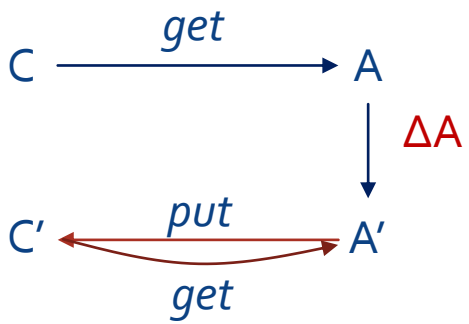
[Dayal & Bernstein 82]: identify FDs under which updates are translatable, and a procedure

Suppose $V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k$, then if $V \rightarrow R_1, R_2, \ldots, R_k$, the update is translatable

[Bancilhon & Spyratos 81] consider *complement* of the view and require the update not change it! When applicable, the update to D is unique and deterministic.

[Gottlob, Paolini, & Zicari 88] consider *consistent views*, generalizing [Bancilhon & Spyratos 81]

Bidirectional Programming Languages



Assume "concrete" model instance C, *get* function supplying "abstract" model instance A

Given an update ΔA to A resulting in A′,

Execute function *put* on C, resulting in C′
Such that applying *get* to C′ yields A′

Lenses [Foster et al 07] with *get* and *put* functions – with *well-behaved* properties guaranteeing updatability

GetPut: put(get(C),C) = C PutGet: get(put(A, C)) = A

- Simple compositional lenses for mapping trees [Foster et al 07]: hoist, fork, map, copy, merge, …
- Extensions for relations [Bohannon et al. 06], and strings, lists, and dictionaries [Foster et al 07b]
- Linguistically enforces the *consistent views* approach [Gottlob et al 88]

"Injective languages" for structured editing with views [Mu et al. 04, Hu et al. 06] – weaker notion of well-behavedness allows content to be replicated within concrete + abstract views

Data Exchange beyond Views

Clio [Fagin et al], Youtopia [Kot & Koch], Orchestra [Ives et al]

So far: the view represents a *subset* of the information in the sources

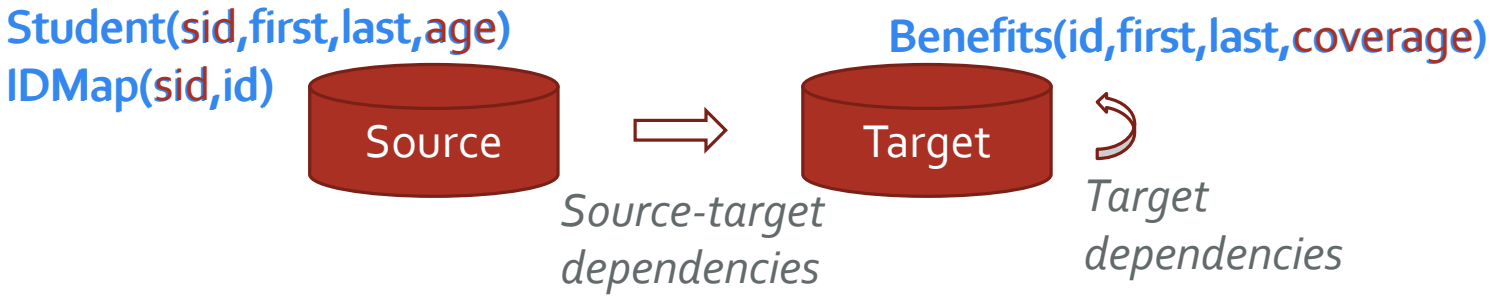
But what happens:

(1) If each site has some unique attributes?

(2) If we allow for sharing across more than one source-target pair?

This requires a generalization of views…

From *Views* to *Schema Mappings*



Data wrangling, fusion, integration:

Define *mappings* as *constraints* or *dependencies* between "source" and "target" schemas

Source and target schemas may include data (tuples, attribs) not present in the other!

Source-target tuple-generating dependencies (tgds) [Beeri & Vardi 84, Popa & Tannen 99]:

$$\forall \bar{x}, \bar{y} (\phi_S(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi_T(\bar{y}, \bar{z}))$$

$$\forall s, f, l, a, i\ (Student(s, f, l, a) \wedge IDMap(s, i) \rightarrow \exists c\ Benefits(i, f, l, c))$$

Chasing Source-Target Mappings: Data Exchange [Fagin et al. 03]

Data exchange instances can incorporate *incomplete information* in the form of *v-tables* [Imielinski & Lipski 84]

For missing values, we use variables, also called *labeled nulls*

$$\forall i, f, l, c\ (Benefits(i, f, l, c) \rightarrow \exists a, s\ \ Student(s, f, l, a) \wedge IDMap(s, i))$$

Benefits

5555	Jun	Li	vision

Student

s_1	Jun	Li	a_1

ID Map

s_1	5555

To propagate: apply the *chase* [Beeri & Vardi 84] using the constraints and the source instance; results in a *canonical universal solution* in the target

Result is homomorphically equivalent to *any other solution* to the chase (hence universal)!

Computing Canonical Universal Solutions in Datalog [Green et al. 07]

Applying the chase with tgds is equivalent to running a *Datalog program with Skolem functions* (which can be expressed in SQL + UDFs)!

$$\forall i, f, l, c\ (Benefits(i, f, l, c) \rightarrow \exists a, s\ \ Student(s, f, l, a) \wedge IDMap(s, i))$$

Student(sid(i),f,l,age(i)) :- Benefits(i,f,l,c)
IDMap(sid(i), i) :- Benefits(i,f,l,c)

Update Propagation with Canonical Universal Solutions [Green et al. 07]

[Gupta & Mumick 95] showed how to incrementally update Datalog views, ie map updates over the source to the target. Eg for insertion:

Given Benefits(id, first, last, coverage) define insert relation +Benefits(id, first, last, coverage)

Give

Student(sid(i

IDMap(sid(i),

its(i,f,l,c)

This is an update in the direction of the mapping. To go *bidirectionally* we need to create a mapping in reverse – which is a simple *network of mappings*.

Let's consider what happens when we have more than one source and target!

(and similarly for deletion relations -Student, -IDMap, -Benefits)

Road Map

- Foundational approaches
- **Pushing the envelope: collaborative data sharing**
- The challenges of bidirectionality
- The need for provenance
- Looking beyond

The Collaborative Data Sharing System (CDSS)

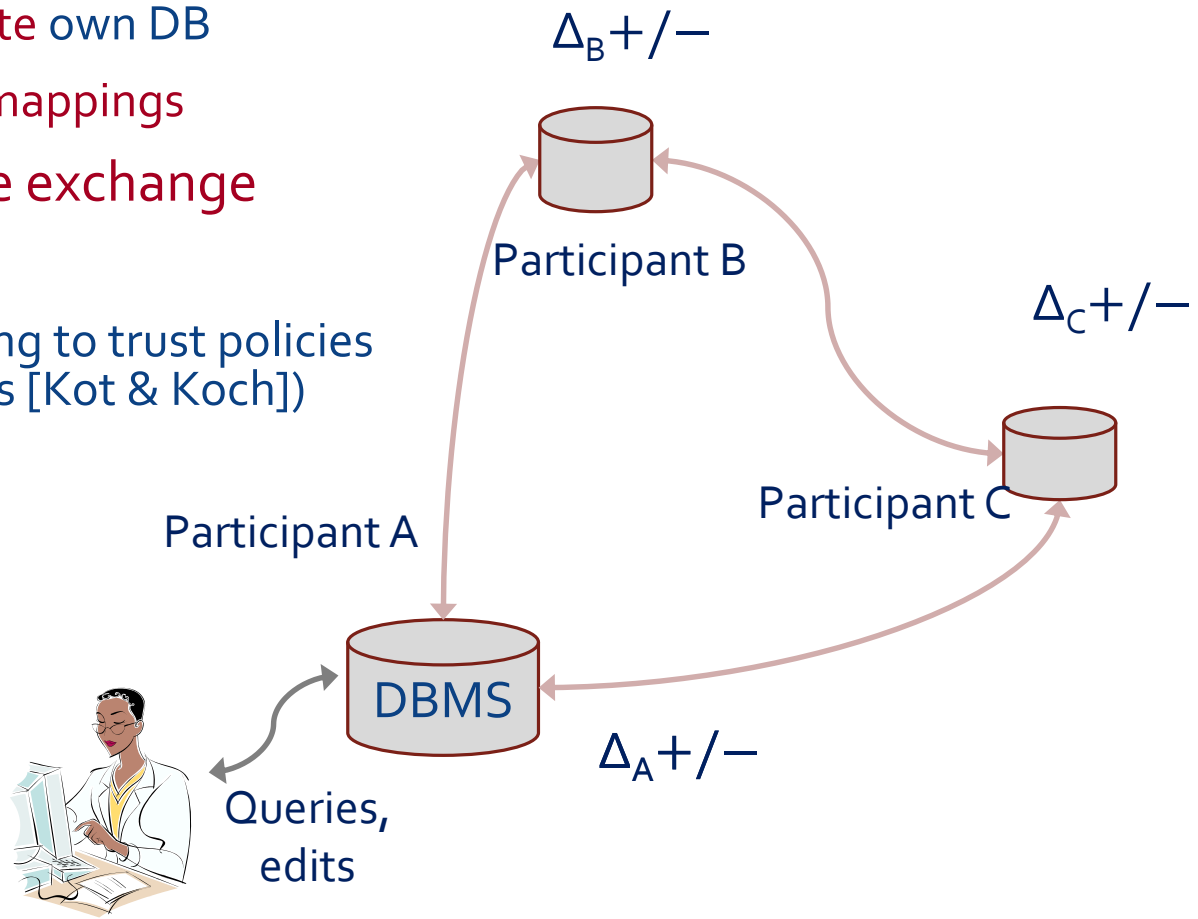
[Ives et al. CIDR05; SIGMOD Rec. 08]

Motivated by scientific consortia: logical P2P network of autonomous data portals

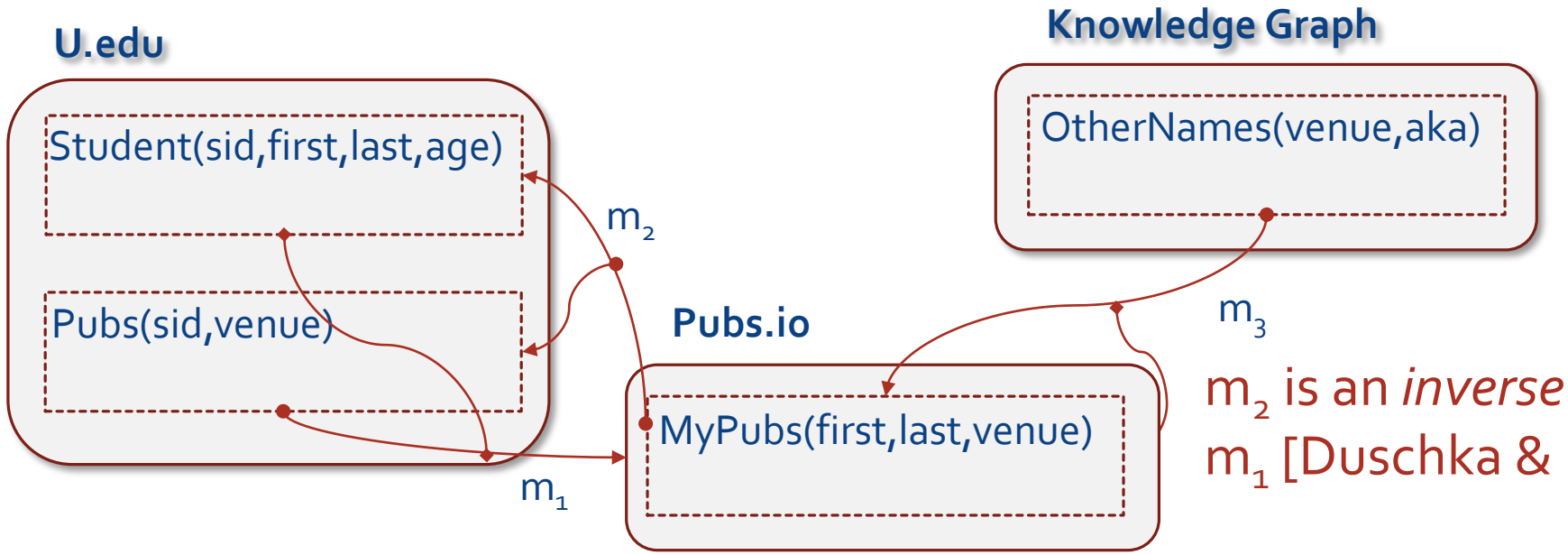
- Participants control & update own DB
- Related by **compositional** mappings

Dataflow: occasional update exchange

- Record data provenance
- (Reconcile conflicts according to trust policies [Green et al] or manual edits [Kot & Koch])



Let's Consider a Simple CDSS *Setting*



m_2 is an *inverse rule* for m_1 [Duschka & Levy 97].

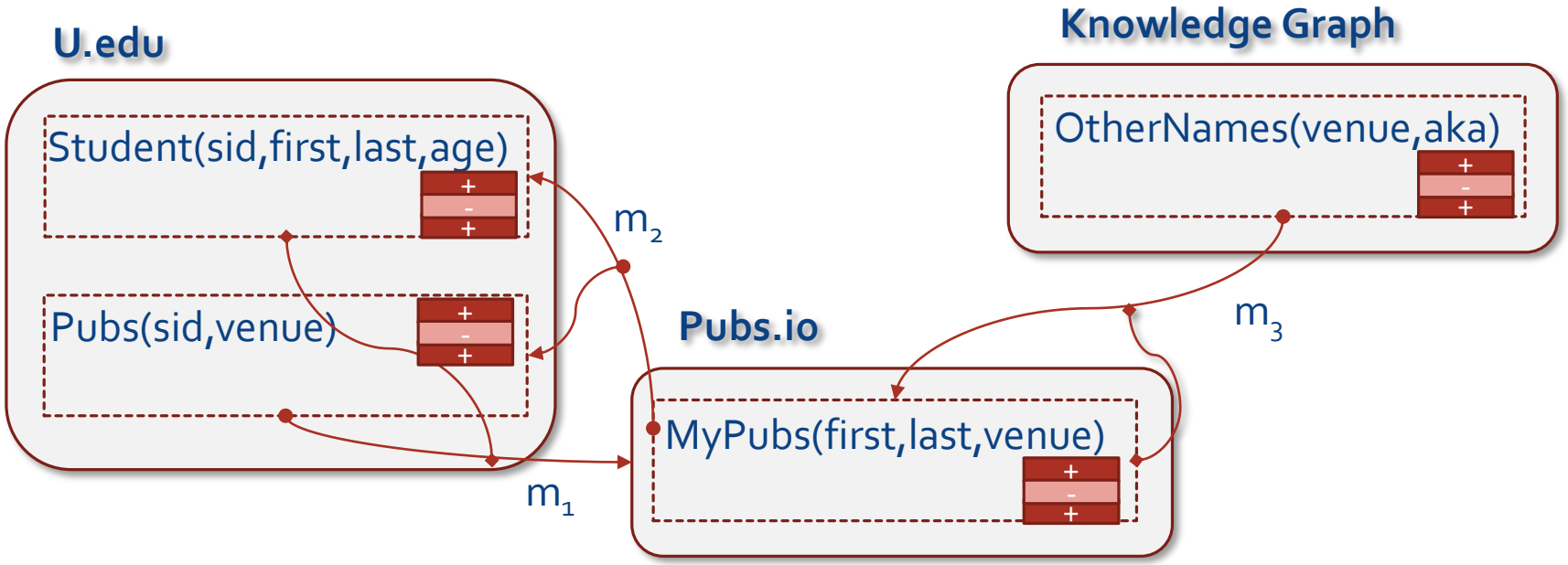
m_1: MyPubs(f,l,v) :- Student(s,f,l,a), Pubs(s,f)

m_{2s}: Student(s(f,l),f,l,a(f,l)) :- MyPubs(f,l,v)
m_{2p}: Pubs(s(f,l),v) :- MyPubs(f,l,v)

m_3: MyPubs(f,l,v) :- MyPubs(f,l,v1), OtherNames(v1,v)

Mappings must be *weakly acyclicy* [Popa & Tannen 99]

Let's Consider a Simple CDSS *Setting*

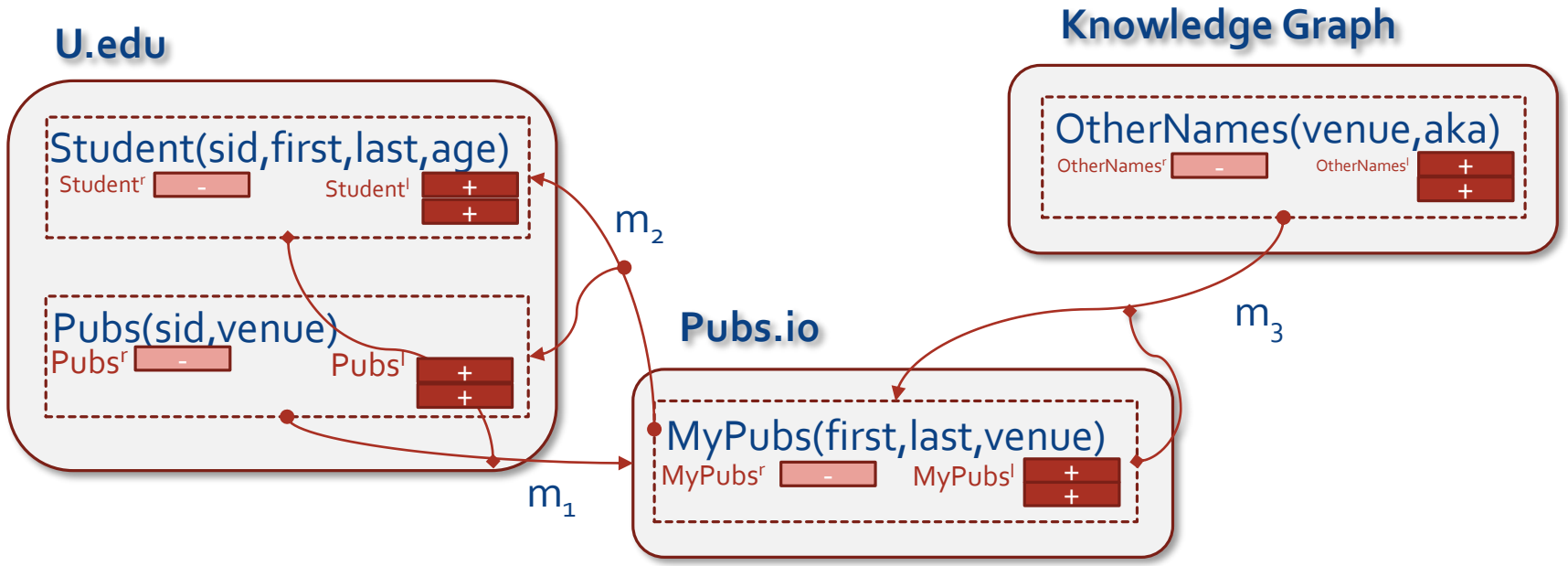


m_1: MyPubs(f,l,v) :- Student(s,f,l,a), Pubs(s,f)

m_{2s}: Student(s(f,l),f,l,a(f,l)) :- MyPubs(f,l,v)
m_{2p}: Pubs(s(f,l),v) :- MyPubs(f,l,v)

m_3: MyPubs(f,l,v) :- MyPubs(f,l,v1), OtherNames(v1,v)

Let's Consider a Simple CDSS *Setting*



m_1: MyPubs(f,l,v) :- Student(s,f,l,a), Pubs(s,f)

m_{2s}: Student(s(f,l),f,l,a(f,l)) :- MyPubs(f,l,v)
m_{2p}: Pubs(s(f,l),v) :- MyPubs(f,l,v)

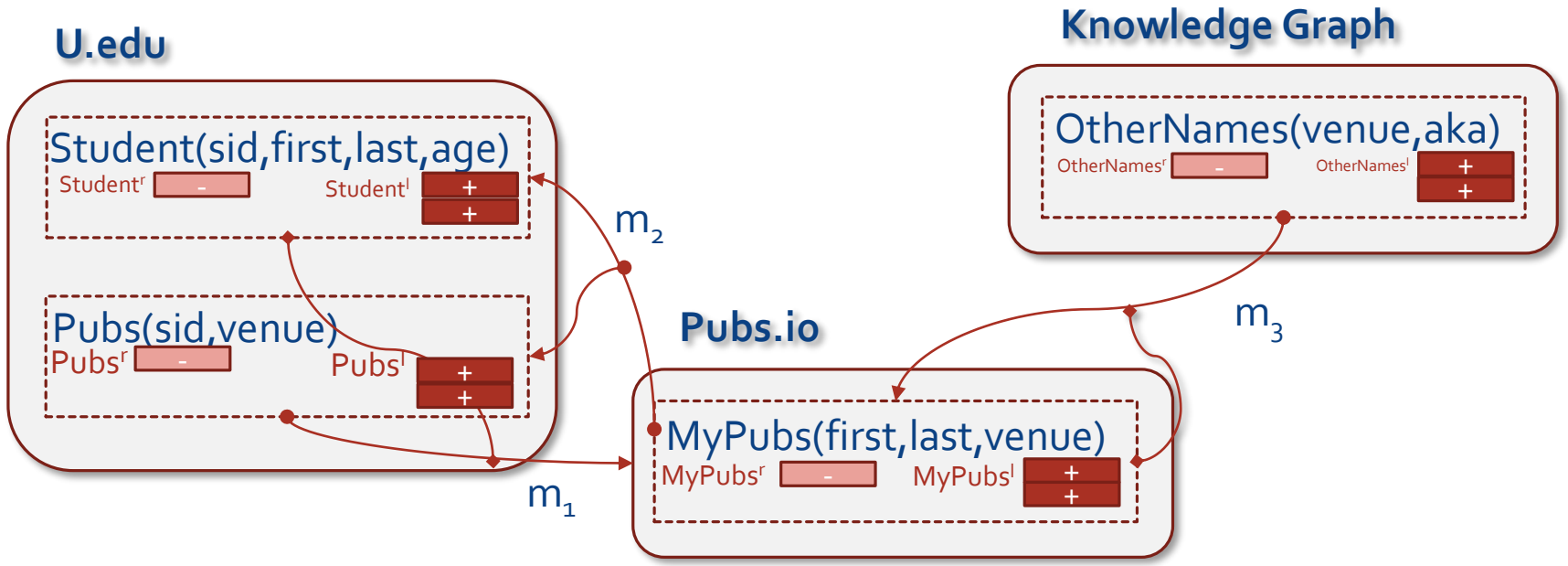
m_3: MyPubs(f,l,v) :- MyPubs(f,l,v1), OtherNames(v1,v)

Encode edit history in relations describing net effects on data

- **Local contributions** (e.g., MyPubs^l)
- **Local rejections** of data imported from elsewhere (e.g., MyPubs^r)

Extend schema mappings to relate these relations

By Default: Updates in the CDSS Affect "Local + Downstream" Data



m_1: MyPubs(f,l,v) :- Student(s,f,l,a), Pubs(s,f), ¬MyPubs^r(f,l,v)

m_{2s}: Student(s(f,l),f,l,a(f,l)) :- MyPubs(f,l,v) , ¬Student^r(s(f,l),l,a(f,l))
m_{2p}: Pubs(s(f,l),v) :- MyPubs(f,l,v) , ¬Pubs^r(s(f,l),l,v)

m_3: MyPubs(f,l,v) :- MyPubs(f,l,v1), OtherNames(v1,v), ¬MyPubs^r(f,l,v)

MyPubs(f,l,v) :- MyPubs^l(f,l,v)

Student(s,f,l,a) :- Student^l(s,f,l,a)
Pubs(s,v) :- Pubs^l(s,v)

OtherNames(v,a) :- OtherNames^l(v,a)

Propagation of Updates Downstream

li_1: MyPubs(f,l,v) :- $MyPubs^l$(f,l,v)

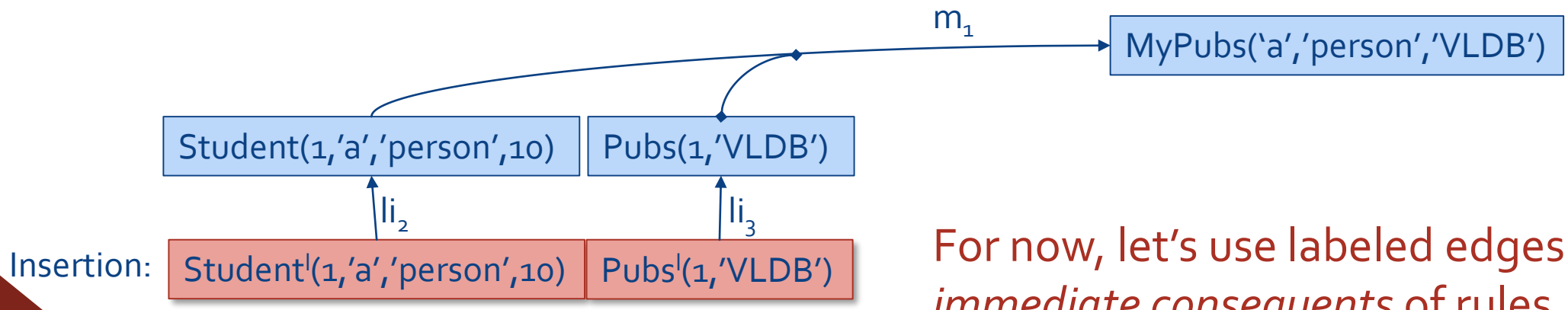
m_1: MyPubs(f,l,v) :- Student(s,f,l,a), Pubs(s,f), ¬$MyPubs^r$(f,l,v)

li_2: Student(s,f,l,a) :- $Student^l$(s,l,f,a)

m_{2s}: Student(s(f,l),f,l,a(f,l)) :- MyPubs(f,l,v) , ¬$Student^r$(s(f,l),l,a(f,l))

li_3: Pubs(s,v) :- $Pubs^l$(s,v)

m_{2p}: Pubs(s(f,l),v) :- MyPubs(f,l,v) , ¬$Pubs^r$(s(f,l),l,v)

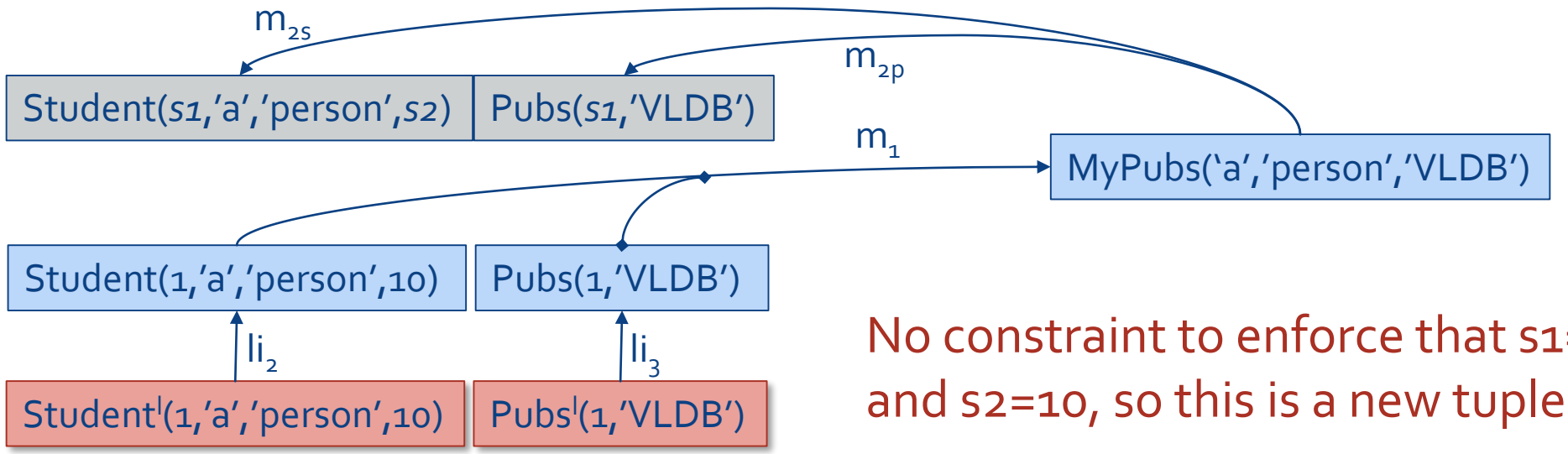


For now, let's use labeled edges to show *immediate consequents* of rules

Propagation of Updates Downstream

li_1: MyPubs(f,l,v) :- $MyPubs^l$(f,l,v)
m_1: MyPubs(f,l,v) :- Student(s,f,l,a), Pubs(s,f), ¬$MyPubs^r$(f,l,v)

li_2: Student(s,f,l,a) :- $Student^l$(s,l,f,a)
m_{2s}: Student(s(f,l),f,l,a(f,l)) :- MyPubs(f,l,v) , ¬$Student^r$(s(f,l),l,a(f,l))
li_3: Pubs(s,v) :- $Pubs^l$(s,v)
m_{2p}: Pubs(s(f,l),v) :- MyPubs(f,l,v) , ¬$Pubs^r$(s(f,l),l,v)



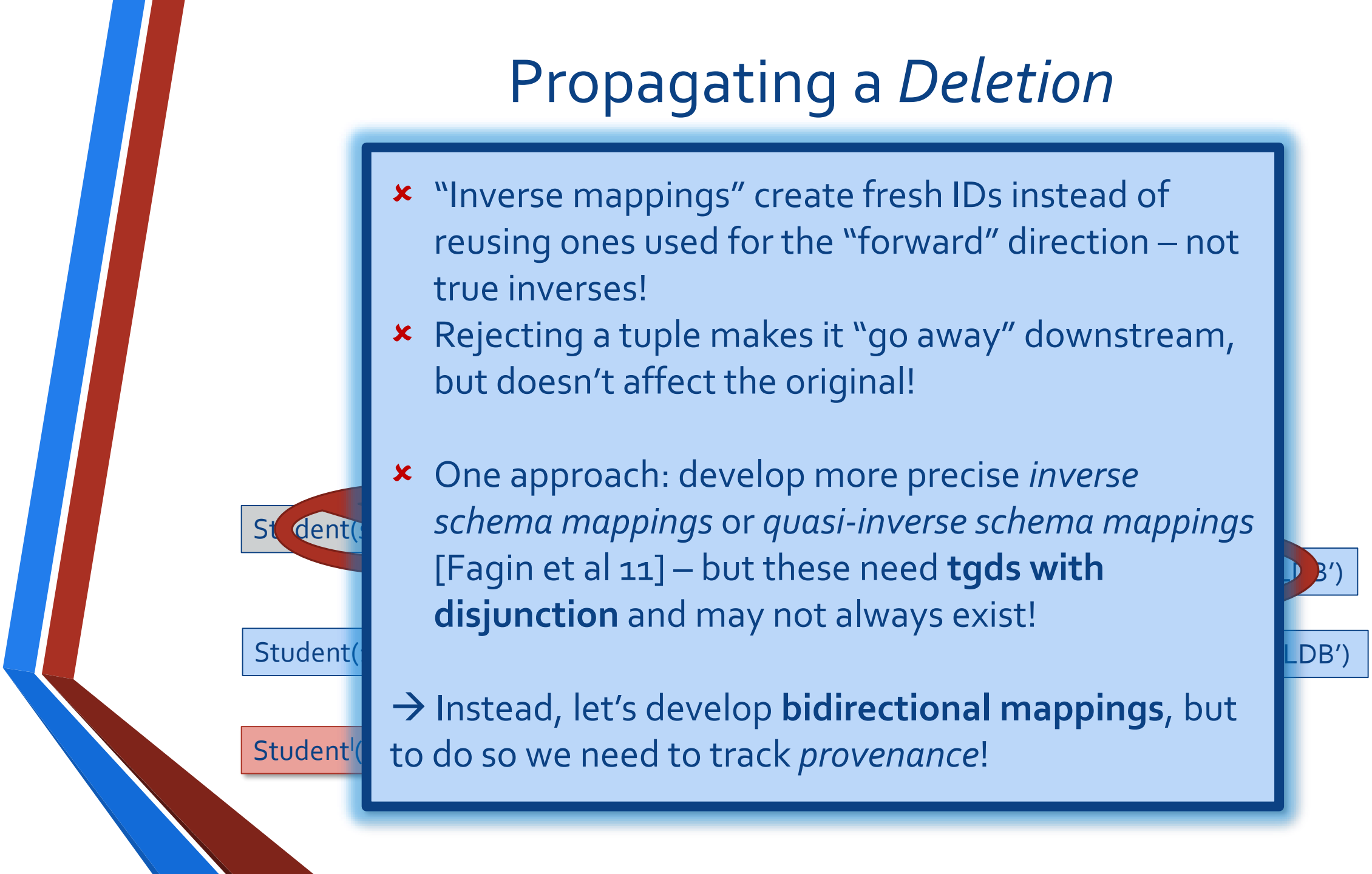
No constraint to enforce that s1=1 and s2=10, so this is a new tuple!

Propagating a *Deletion*

- "Inverse mappings" create fresh IDs instead of reusing ones used for the "forward" direction – not true inverses!
- Rejecting a tuple makes it "go away" downstream, but doesn't affect the original!

- One approach: develop more precise *inverse schema mappings* or *quasi-inverse schema mappings* [Fagin et al 11] – but these need **tgds with disjunction** and may not always exist!

→ Instead, let's develop **bidirectional mappings**, but to do so we need to track *provenance*!



Road Map

- Foundational approaches
- **Pushing the envelope: collaborative data sharing**
- **The challenges of bidirectionality**
- The need for provenance
- Looking beyond

Background: Semiring Provenance [Green et al PODS07, Green et al VLDB07]

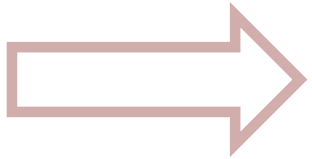
Consumes

org	eats	
cat	mouse	*p*
cat	rat	*q*

Colors

org	color	
mouse	gray	*r*
mouse	red	*s*
rat	gray	*t*

FoodColor(x,z):-
Consumes(x,y),
Color(y,z)



Provenance polynomials
(N[*X*], +, ·, 0, 1) semiring

FoodColor

cat	gray	$p \cdot r + q \cdot t$
cat	red	$p \cdot s$

- · joint use (join, Cartesian product),
- \+ alternative use (union, projection)

\+ is associative, commutative, with 0 identity

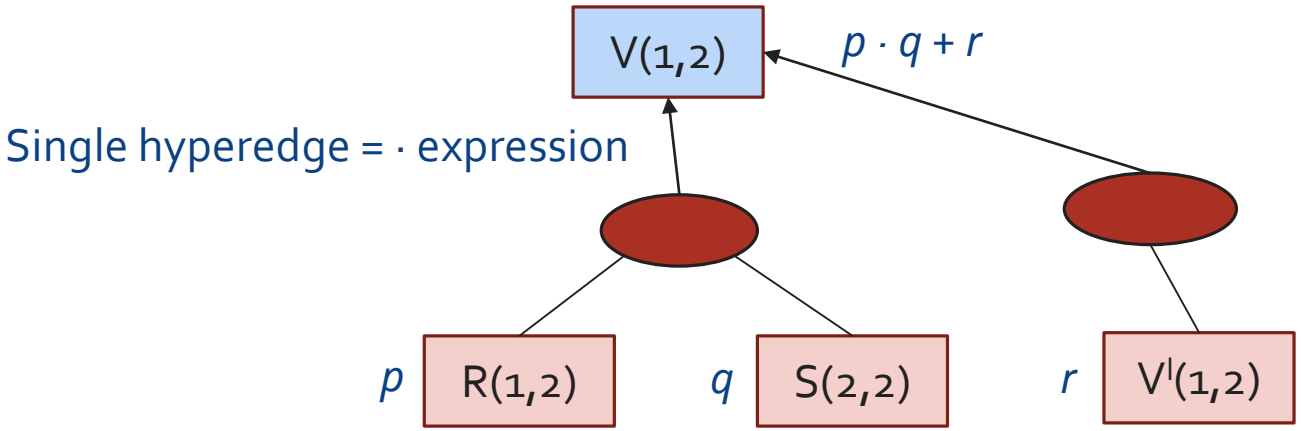
· is associative, commutative with 1 identity

Derived from [Tannen PODS'17 Talk]

Background: Semiring Provenance as a Graph [Green et al VLDB07]

We can also model provenance polynomials expressions in a *hypergraph*

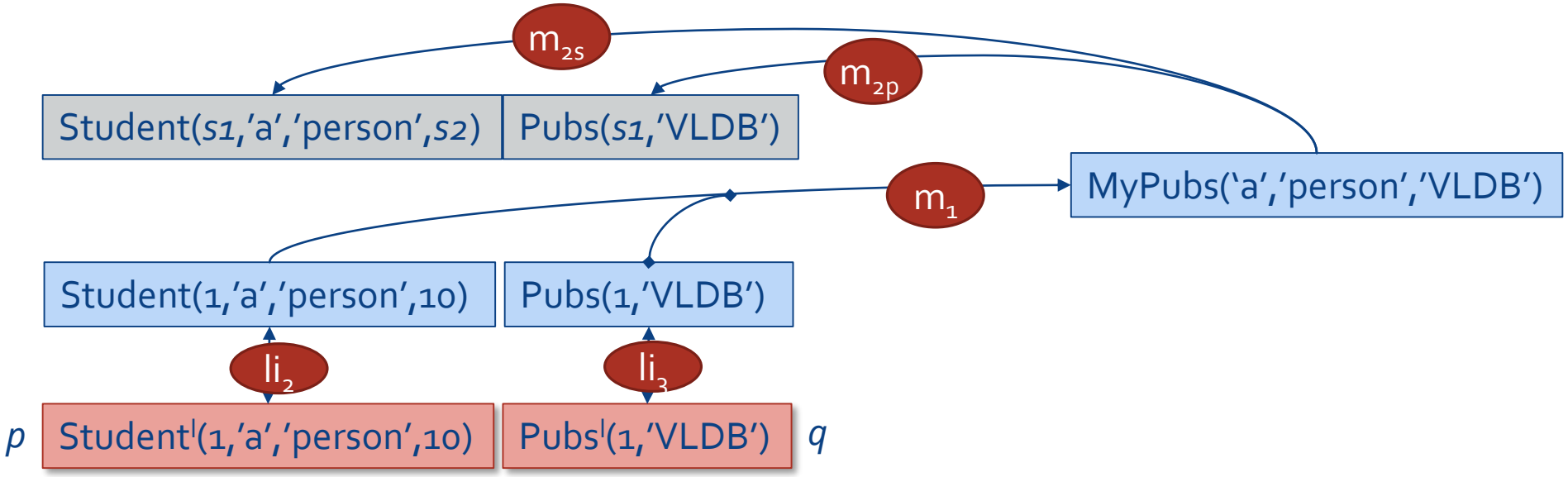
Consider a union of two conjunctive queries
V(a,c) :- R(a,b), S(b,c)
:- V'(a,c)



Provenance in Our Cyclic Mapping

li_1: MyPubs(f,l,v) :- MyPubs^l(f,l,v)
m_1: MyPubs(f,l,v) :- Student(s,f,l,a), Pubs(s,f), ¬MyPubs^r(f,l,v)

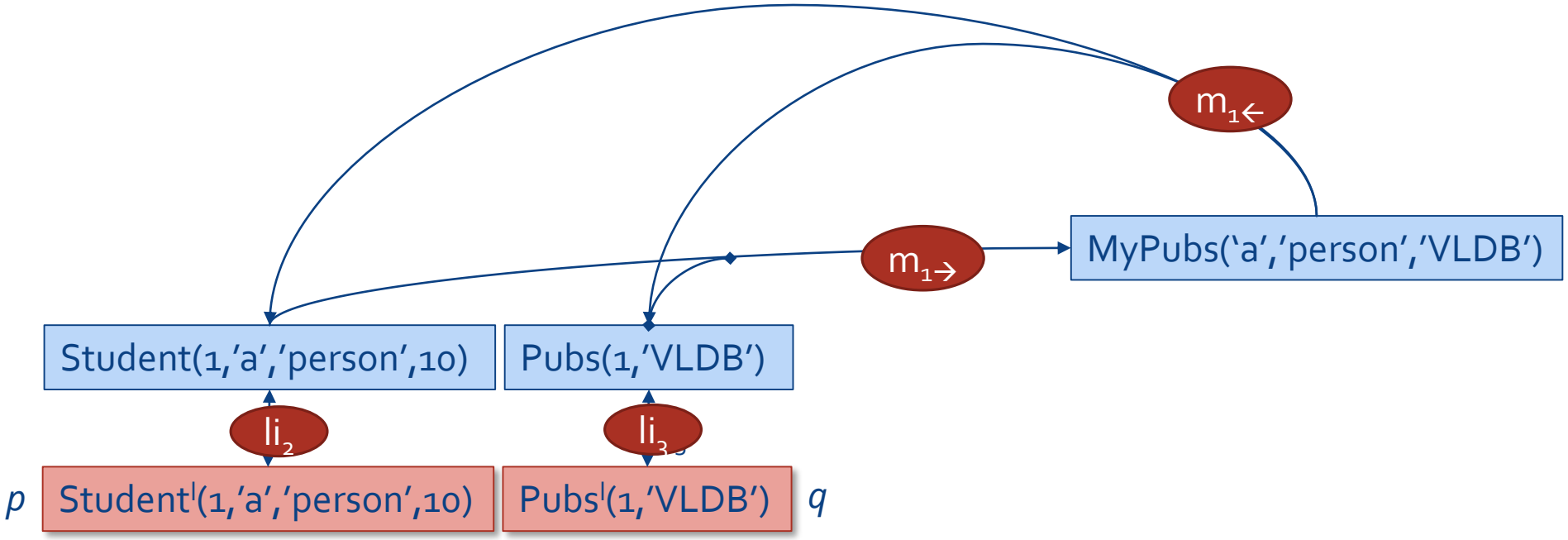
li_2: Student(s,f,l,a) :- Student^l(s,l,f,a)
m_{2s}: Student(s(f,l),f,l,a(f,l)) :- MyPubs(f,l,v) , ¬Student^r(s(f,l),l,a(f,l))
li_3: Pubs(s,v) :- Pubs^l(s,v)
m_{2p}: Pubs(s(f,l),v) :- MyPubs(f,l,v) , ¬Pubs^r(s(f,l),l,v)



Making Our Mapping Bidirectional [Karvounarakis & Ives 08]

li_1: MyPubs(f,l,v) :- MyPubs^l(f,l,v)
m_1: MyPubs(f,l,v) ←→ Student(s,f,l,a), Pubs(s,f)

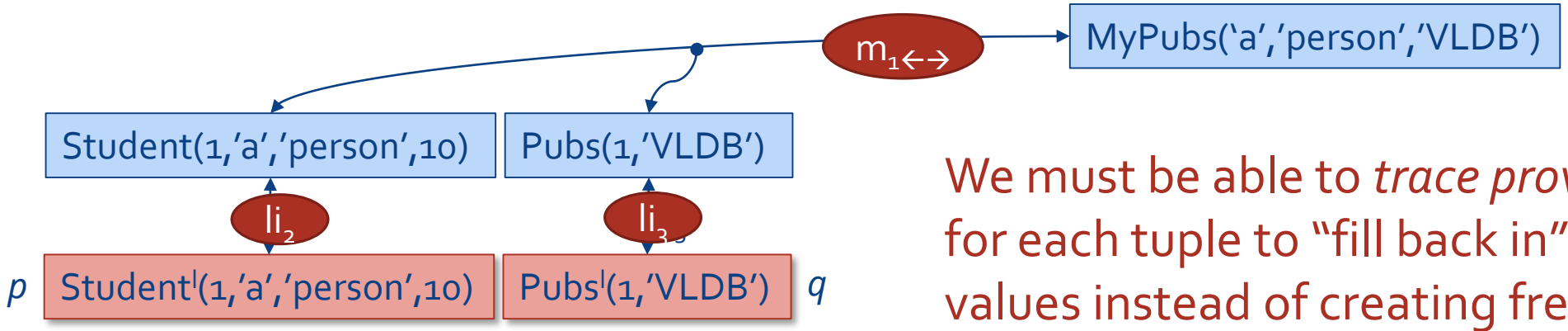
li_2: Student(s,f,l,a) :- Student^l(s,l,f,a)
li_3: Pubs(s,v) :- Pubs^l(s,v)



Making Our Mapping Bidirectional [Karvounarakis & Ives 08]

li_1: MyPubs(f,l,v) :- MyPubs^l(f,l,v)
m_1: MyPubs(f,l,v) ←→ Student(s,f,l,a), Pubs(s,f)

li_2: Student(s,f,l,a) :- Student^l(s,l,f,a)
li_3: Pubs(s,v) :- Pubs^l(s,v)

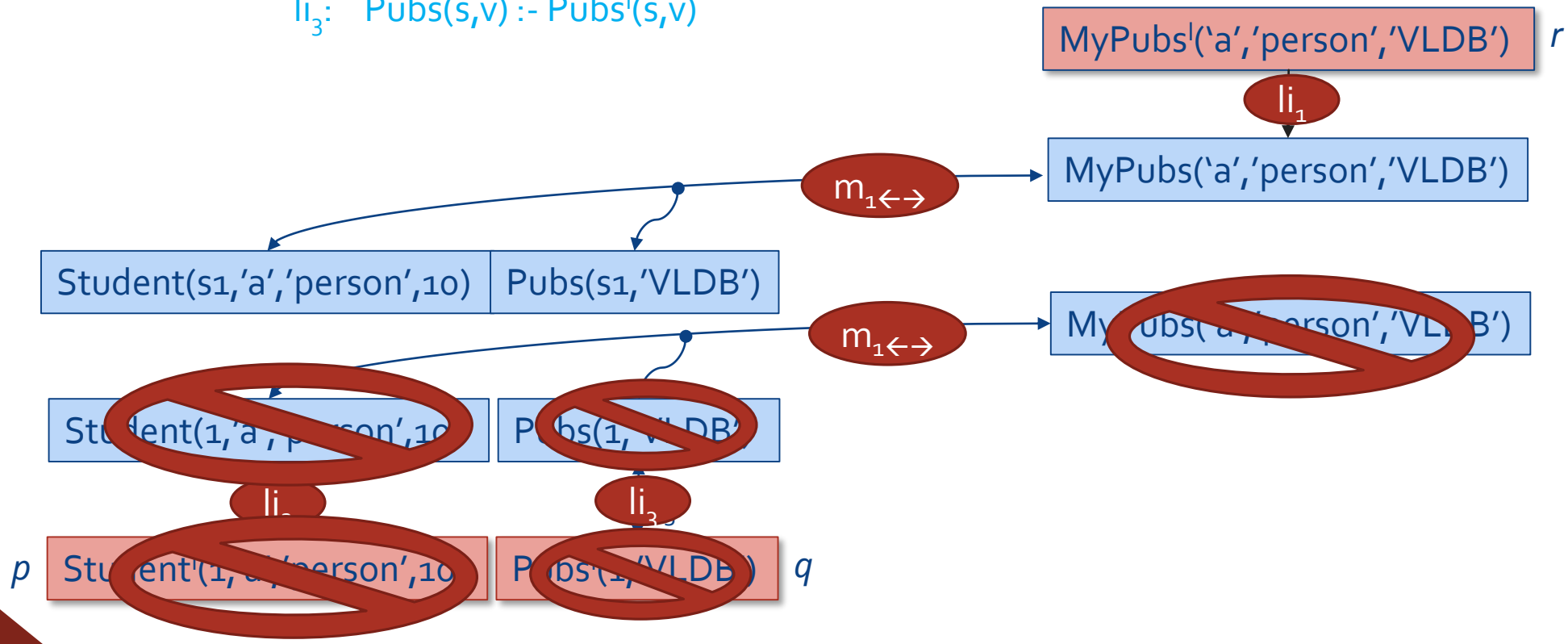


We must be able to *trace provenance* for each tuple to "fill back in" missing values instead of creating fresh ones!

Making Our Mapping Bidirectional [Karvounarakis & Ives 08]

li_1: MyPubs(f,l,v) :- MyPubs^l(f,l,v)
m_1: MyPubs(f,l,v) ←→ Student(s,f,l,a), Pubs(s,f)

li_2: Student(s,f,l,a) :- Student^l(s,l,f,a)
li_3: Pubs(s,v) :- Pubs^l(s,v)

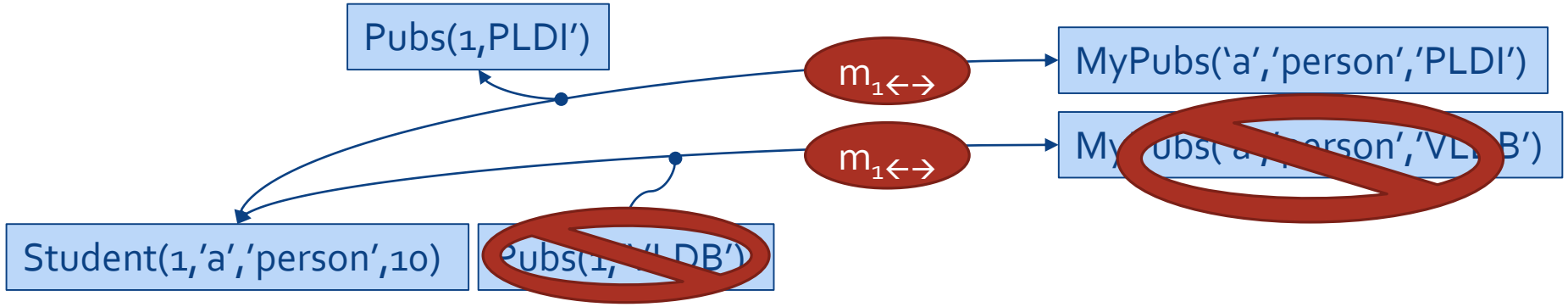


Other Important Details [Karvounarakis & Ives 08]

- Deletion with join can be ambiguous, so we *mark* the source(s) to be deleted

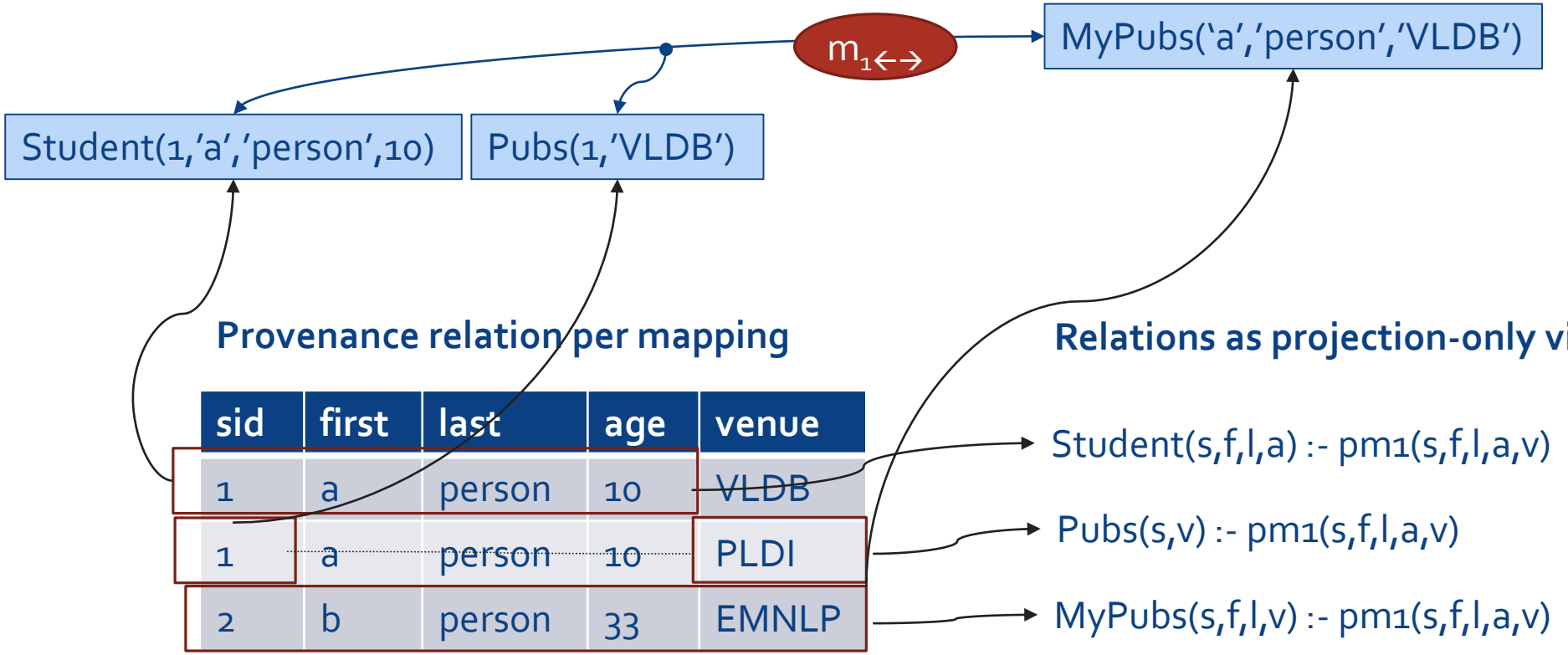
  MyPubs(f,l,v) ←→ Student(s,f,l,a), *Pubs(s,f)

- What to do about updates when a source tuple is shared?
  Too strict to look at it in an *instance independentway*!



Provenance lets us detect these cases *specific to the instance*

Key to Implementation: Provenance Relations [Karvounarakis et al. 10]



Scaling to Many Mappings, Schemas & Updates

- With a commercial RDBMS as the query engine, can scale to 100s of peers with periodic updates

  [Green+07],[Karvounarakis dissert.]

- Moving to a cluster-based, sharded query engine [Taylor & Ives 10] scales up further by a factor of 10+ (10 machines)
  - Could also use Apache Spark, Flink with a cloud DBMS

- Can also incorporate *user intervention* when updates aren't compatible [Kot & Koch]

Conclusions and Open Problems

The update propagation problem has been studied in many contexts

- Desired outcomes are the same but enforcement mechanisms (and instance-independence) vary
- The more general problem involves mappings instead of views – here we need provenance!

Potential avenues of future exploration

- Incremental changes to the transformations? (see [Green et al 12] for a first attempt)
- Provenance & mappings with unknowns from a PL perspective? [Anjorin & Cheney TaPP]
- Updates with aggregate functions and ML – update the **most responsible** [Meliou et al. 10] or **influential** [Koh & Liang 17] inputs?
- Synthesis of bidirectional mappings / lenses, vs schema mapping inference (e.g., [Miller et al 2000]