PRISM++

Update Rewriting and Integrity Constraint Maintenance

Carlo Curino

Hyun J. Moon, Alin Deutsch, Carlo Zaniolo

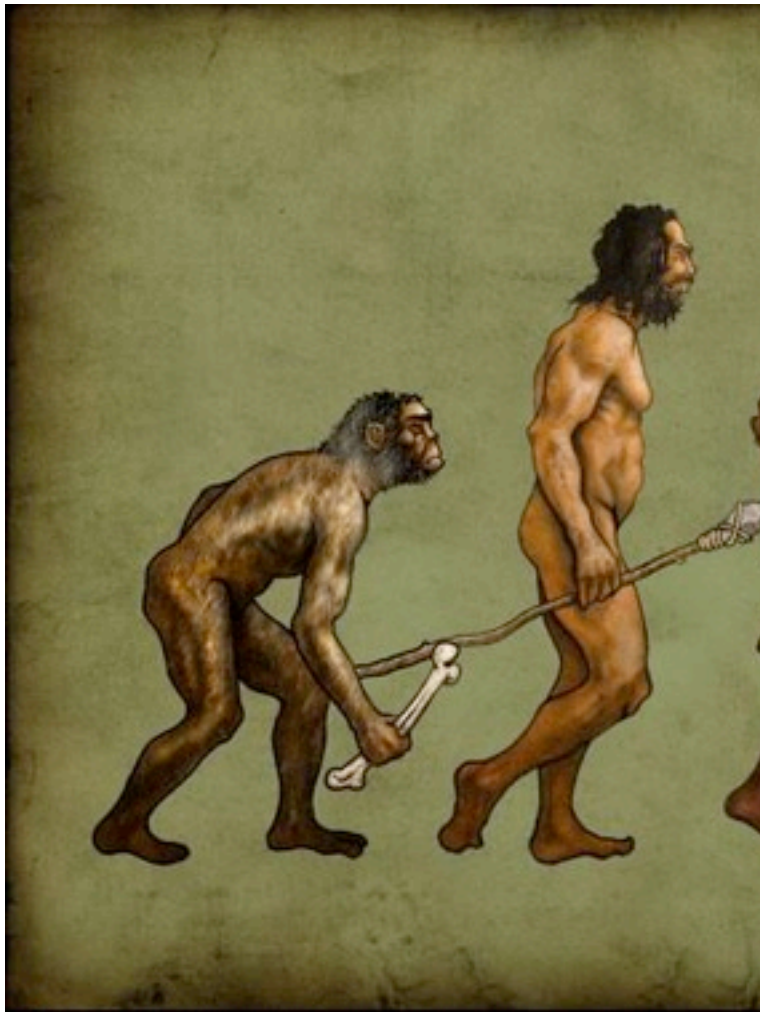
Motivation

- Information Systems Evolution… it's hard!

Motivation

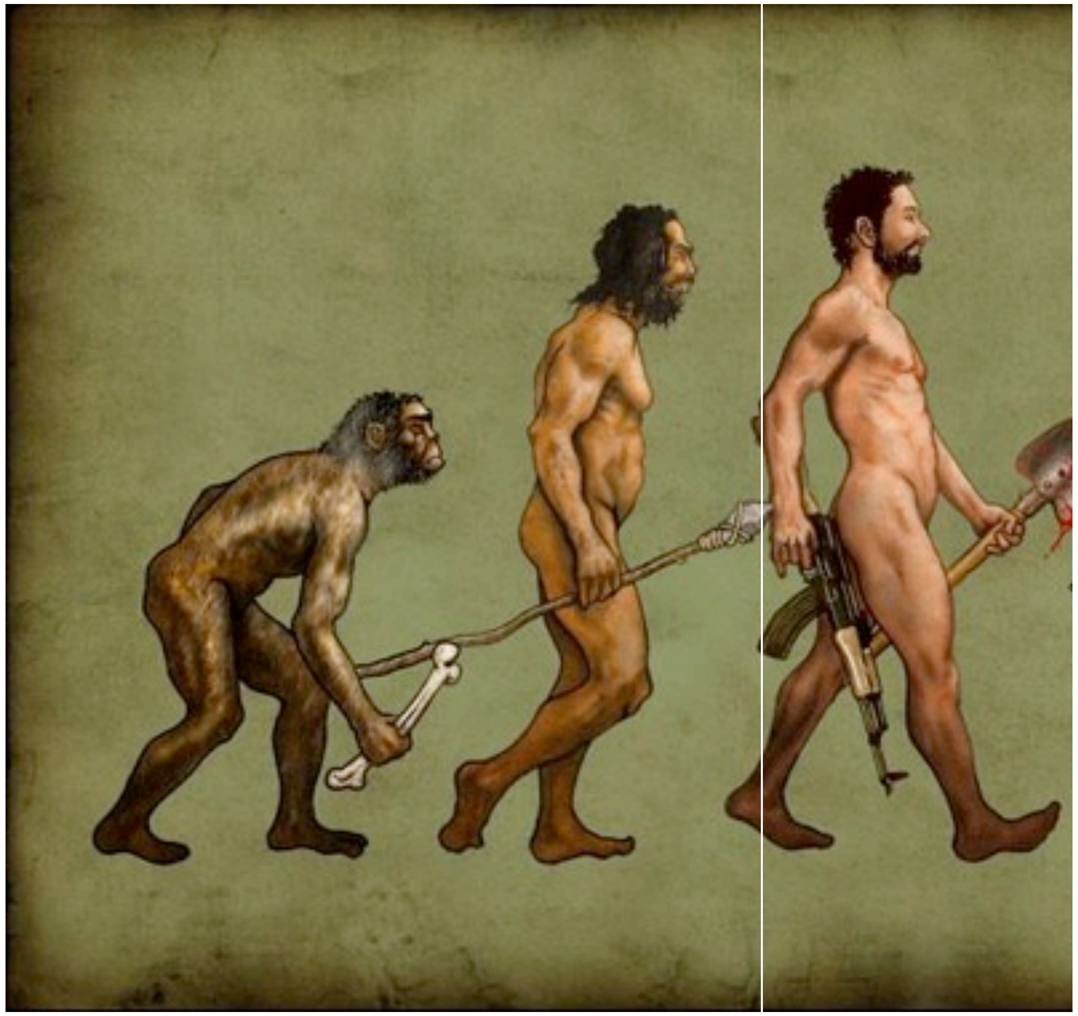
- Information Systems Evolution... it's hard!

Motivation



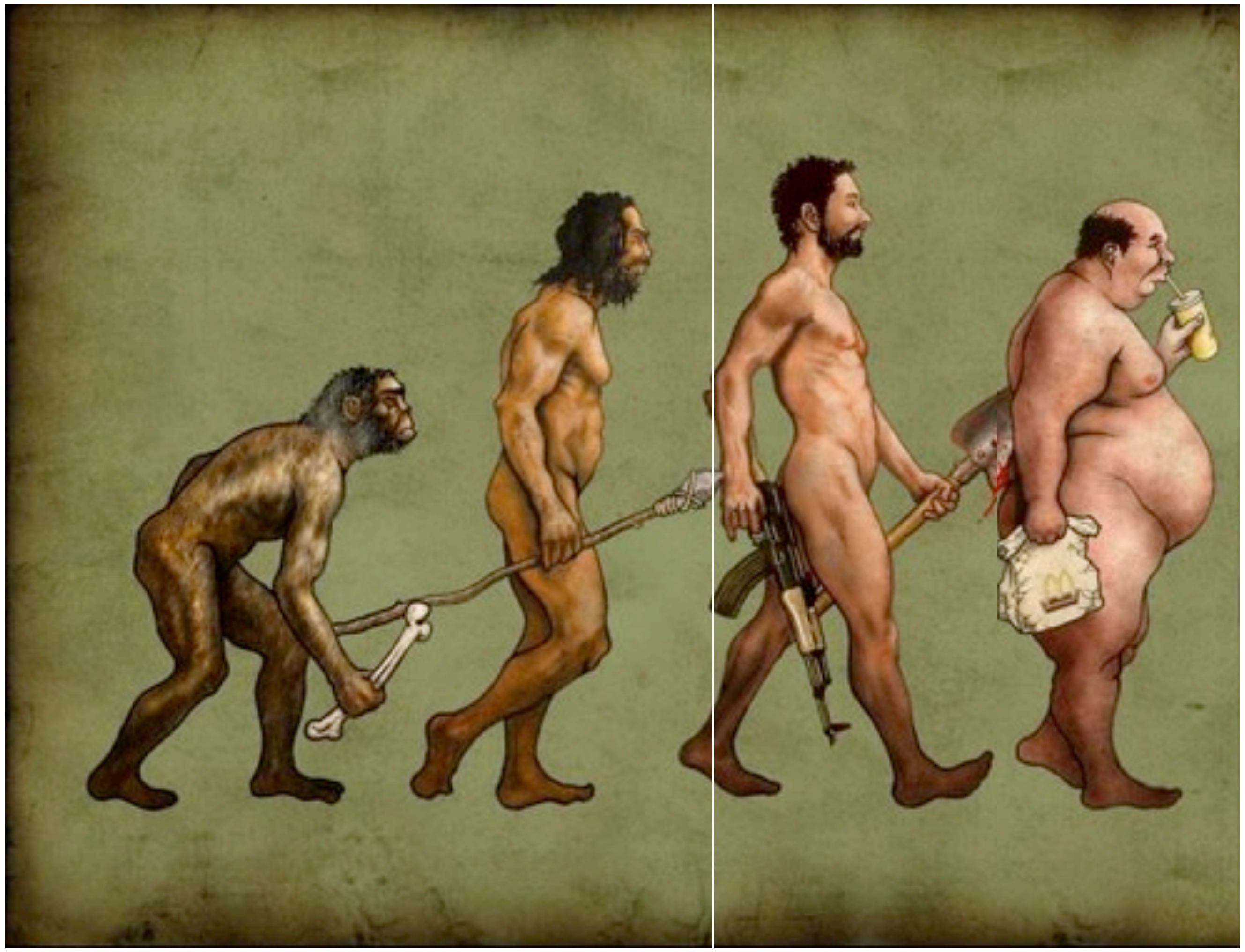
- Information Systems Evolution... it’s hard!

Motivation



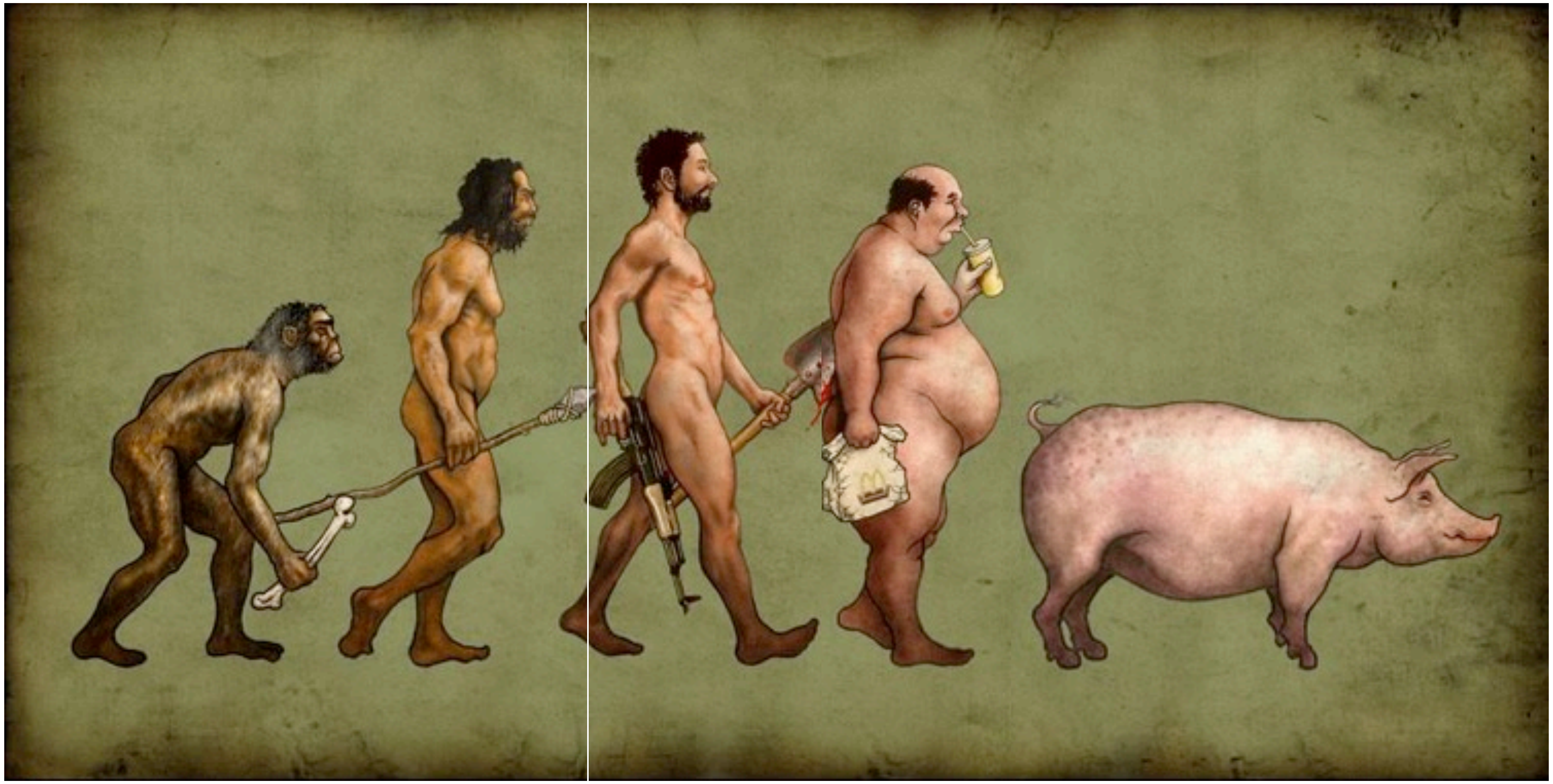
- Information Systems Evolution... it's hard!

Motivation



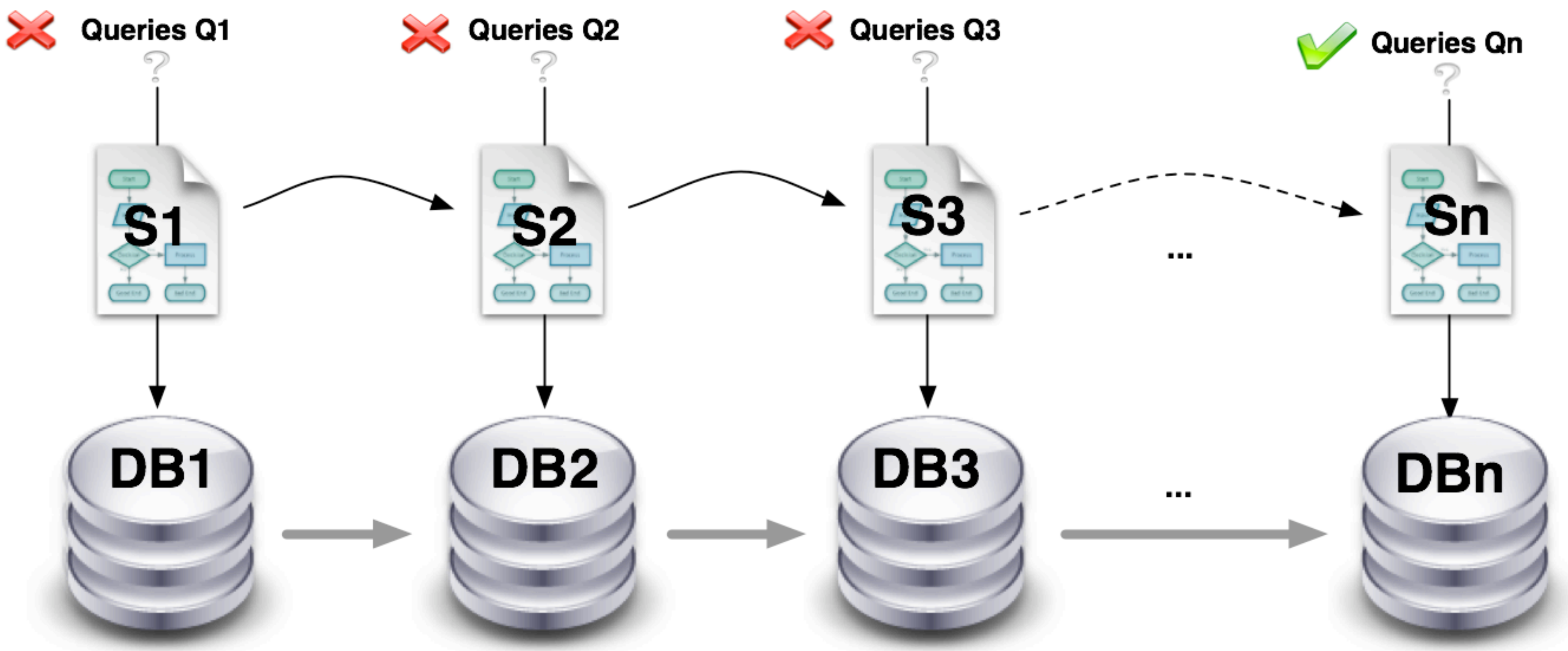
- Information Systems Evolution... it's hard!

Motivation



- Information Systems Evolution... it's hard!

Motivation: Schema Evolution



- change schema
- migrate data
- fix queries/updates
- check/modify app code

Motivation: Schema Evolution

System Name	System type	# of schema versions	lifetime (years)
ATutor	Educational CMS	216	5.7
CERN DQ2	Scientific DB	51	1.3
Dekiwiki	CRM, ERP	11	1.11
E107	CMS	16	5.4
Ensembl	Scientific DB	412	9.8
KT-DMS	CMS	105	4
Nucleus CMS	CMS	51	6.7
PHPWiki	Wiki	18	4.11
SlashCode (slashdot.org)	News Website	256	8.10
Tikiwiki	Wiki	99	0.9
Mediawiki (Wikipedia.org)	Wiki	242	6.2
Zabbix	Monitoring solution	196	8.3

- Average of 31 schema version per year

Motivation: Schema Evolution

System Name	System type	# of schema versions	lifetime (years)
ATutor	Educational CMS	216	5.7
CERN DQ2	Scientific DB	51	1.3
Dekiwiki	CRM, ERP	11	1.11
E107	CMS	16	5.4
Ensembl	Scientific DB	412	9.8
KT-DMS	CMS	105	4
Nucleus CMS	CMS	51	6.7
PHPWiki	Wiki	18	4.11
SlashCode (slashdot.org)	News Website	256	8.10
Tikiwiki	Wiki	99	0.9
Mediawiki (Wikipedia.org)	Wiki	242	6.2
Zabbix	Monitoring solution	196	8.3

- Average of 31 schema version per year

Our Previous Work

	Structural Evolution	Integrity Constraints Evolution
Data	✓	✓ ✗
Queries	✓	✗
Updates	✗	✗

- Schema Modification Operators (SMOs)
- Query rewriting engine based on <u>*chase&backchase*</u>

What are we going to do?

- Integrity Constraints Evolution
  - Introduce integrity-constraint mod. operators (ICMOs)
  - Adapt schema modification operators (SMOs)

- Updates (and queries with negation)
  - Novel update representation *(query equivalence)*
  - Extended rewriting engine *(support for negation and ICMOs)*

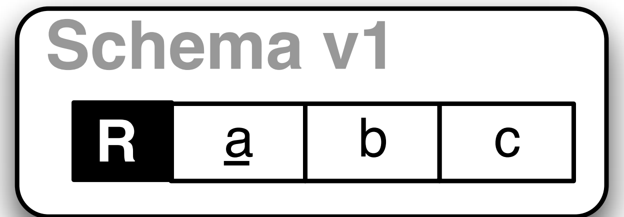
Evolution Operators

- Key idea: *separate structural changes (SMOs) from non-information preserving* ones (ICMOs)*

**information-preserving = invertible mapping = constant information-capacity*

Evolution Operators

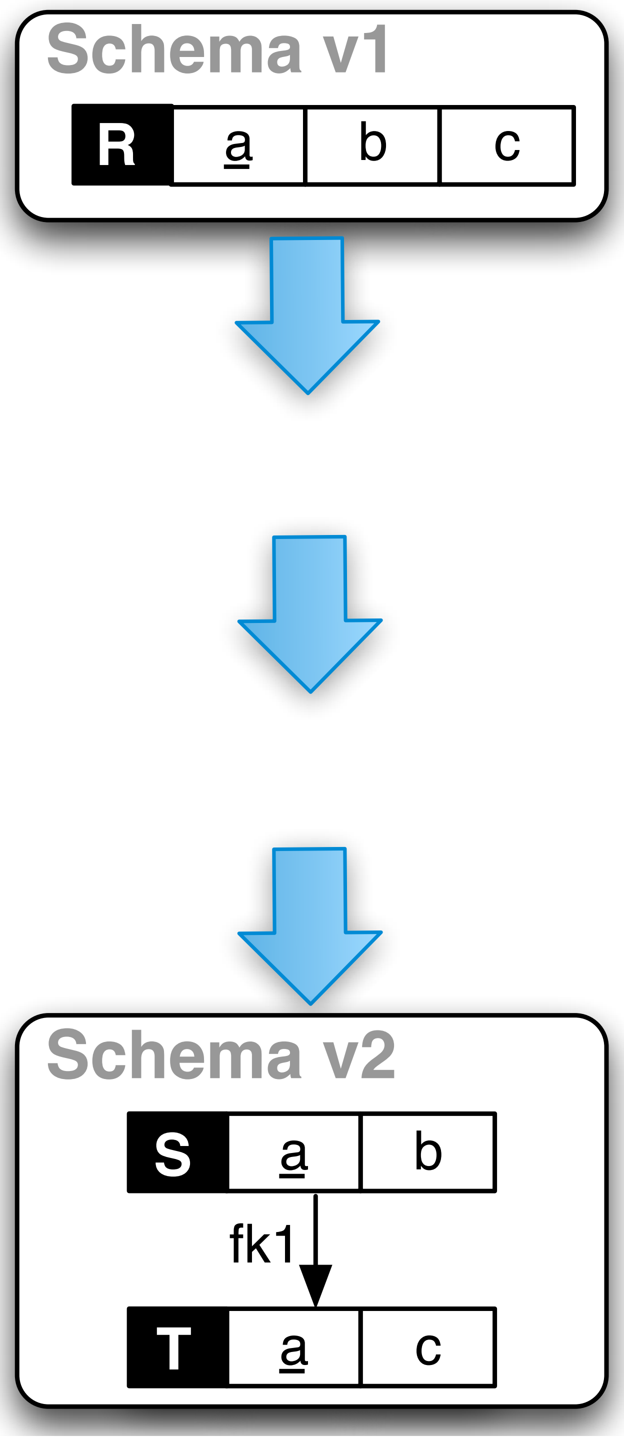
- Key idea: *separate structural changes (SMOs) from non-information preserving* ones (ICMOs)*



**information-preserving = invertible mapping = constant information-capacity*

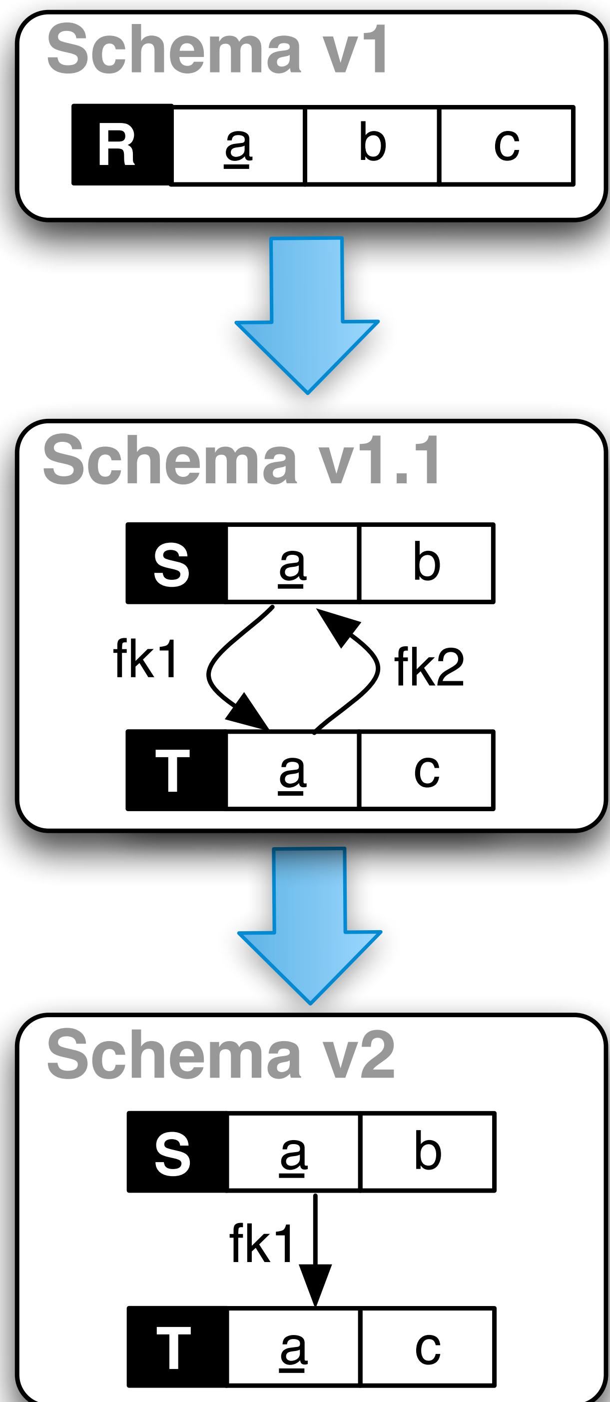
Evolution Operators

- Key idea: *separate structural changes (SMOs) from non-information preserving* ones (ICMOs)*



Evolution Operators

- Key idea: *separate structural changes (SMOs) from non-information preserving* ones (ICMOs)*

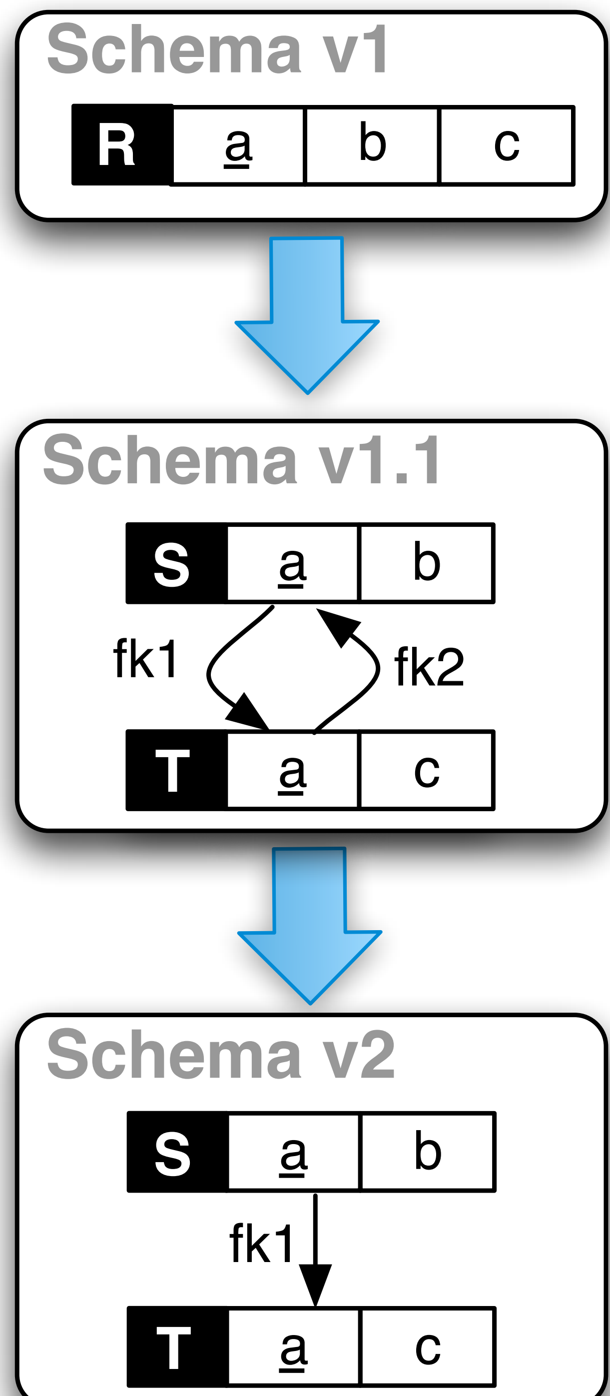



```
DECOMPOSE R INTO S(a,b), T(a,c);
```

```
ALTER TABLE T DROP FOREIGN KEY fk2;
```

Evolution Operators

- Key idea: *separate structural changes (SMOs) from non-information preserving* ones (ICMOs)*



DECOMPOSE R INTO S(a,b), T(a,c);

- changes to schema structure
- information preserving

ALTER TABLE T DROP FOREIGN KEY fk2;

- no changes to schema structure
- not information preserving

Good/Bad News

- We force every *SMO* to be information-preserving (*data migration and query rewriting paradise!*)

- *ICMOs:*
  - risk of data loss
  - rewriting not obvious (*new alg.*)
  - inverse operator (*user input*)

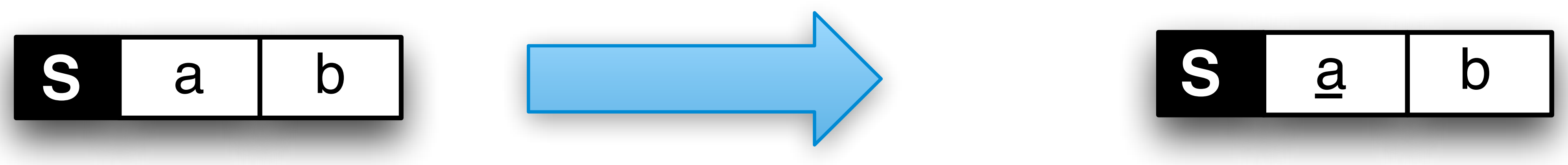
Data Migration

- Challenge: *migrating towards a "tighter" schema (data loss)*

Data Migration

- Challenge: *migrating towards a "tighter" schema (data loss)*

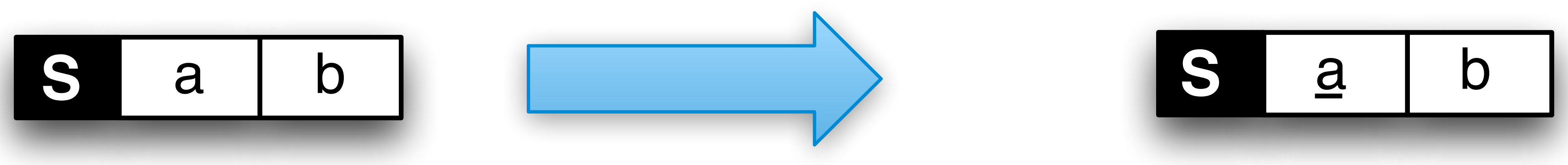
```
ALTER TABLE S
ADD PRIMARY KEY pk1(a)
<policy>;
```



Data Migration

- Challenge: *migrating towards a "tighter" schema (data loss)*

```
ALTER TABLE S
ADD PRIMARY KEY pk1(a)
<policy>;
```



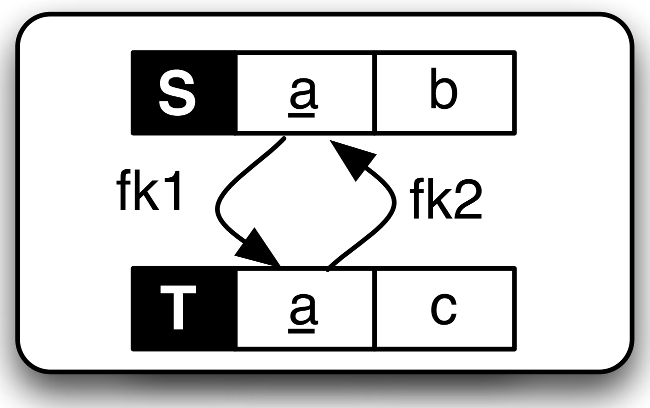
- <policy>:
  - CHECK: *migrates data only if constraint already holds*
  - ENFORCE: *"canonical repair" by moving all violating tuples to special table* S_{viol} a b

Query Rewriting

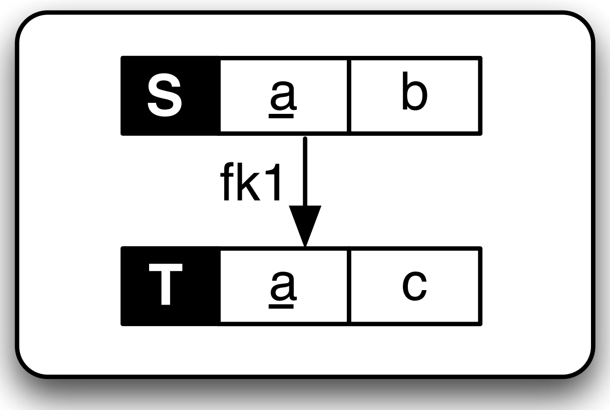
- Challenge: *evolution towards a "looser" schema (inverse is not inf-preserving)*

Query Rewriting

- Challenge: *evolution towards a "looser" schema (inverse is not inf-preserving)*

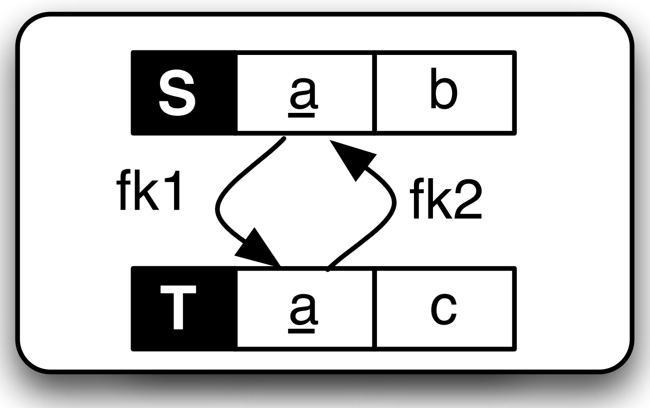



```
ALTER TABLE T
DROP FOREIGN KEY fk2;
```

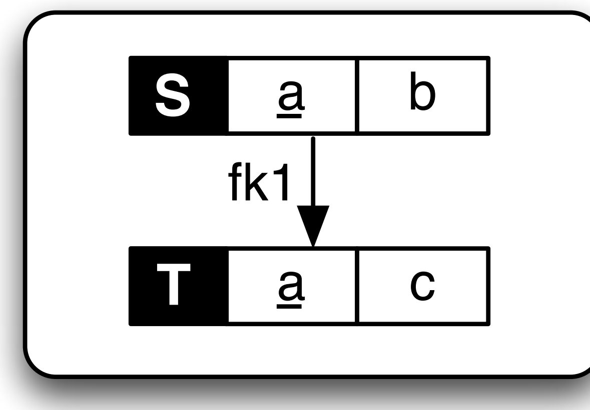


Query Rewriting

- Challenge: *evolution towards a "looser" schema (inverse is not inf-preserving)*



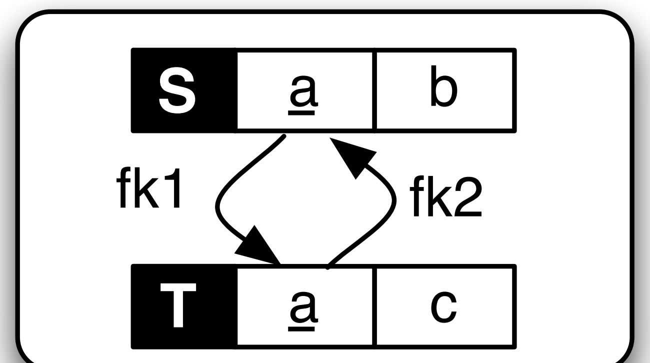

```
ALTER TABLE T
DROP FOREIGN KEY fk2;
```



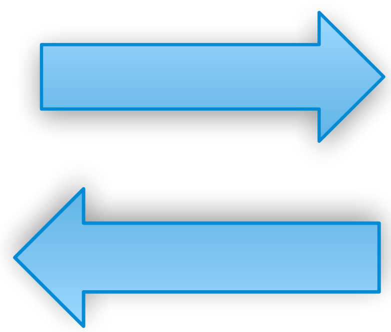

```
ALTER TABLE T
ADD FOREIGN KEY fk2(a)
REFERENCES S(a) <policy>;
```

Query Rewriting

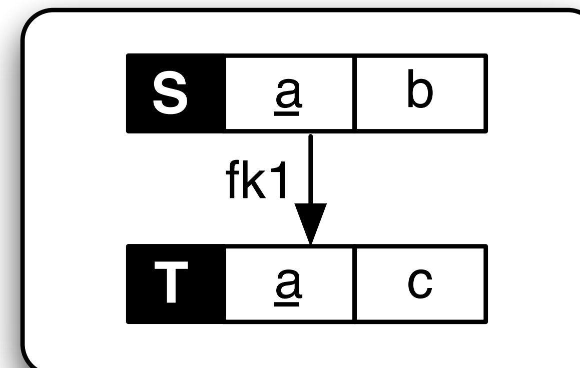
- Challenge: *evolution towards a "looser" schema (inverse is not inf-preserving)*




```
ALTER TABLE T
DROP FOREIGN KEY fk2;
```



```
ALTER TABLE T
ADD FOREIGN KEY fk2(a)
REFERENCES S(a) <policy>;
```

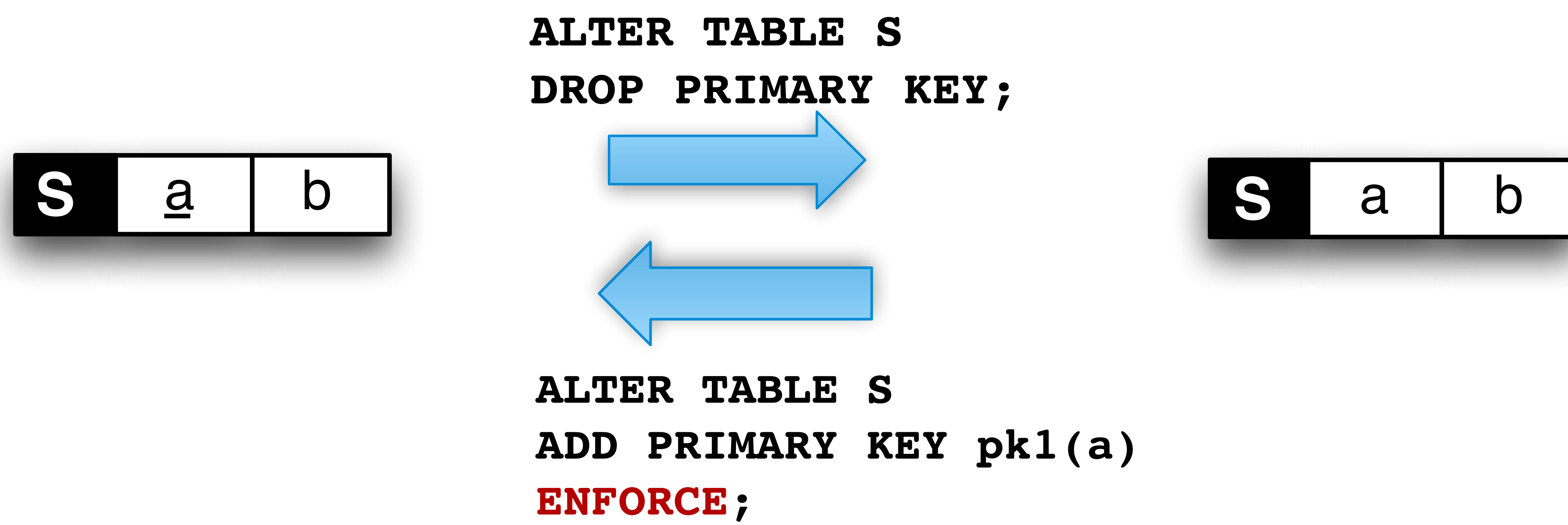


- <policy>:
  - CHECK: *checks constraint before running query*
  - ENFORCE: *limits query scope to non-violating tuples*
  - IGNORE: *runs query as-is*

Query Rewriting Example

- ENFORCE: *limits query scope to non-violating tuples*

Query Rewriting Example

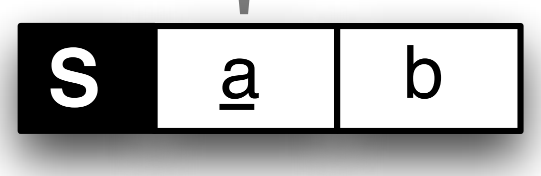


- ENFORCE: *limits query scope to non-violating tuples*

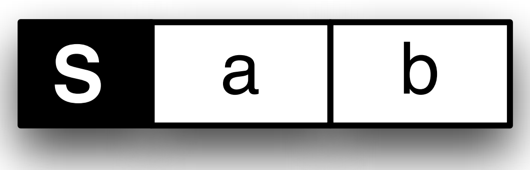
Query Rewriting Example

Q

```
SELECT a,b FROM S
WHERE a=1;
```




```
ALTER TABLE S
DROP PRIMARY KEY;
```




```
ALTER TABLE S
ADD PRIMARY KEY pk1(a)
ENFORCE;
```

- ENFORCE: *limits query scope to non-violating tuples*

Query Rewriting Example

Q

```
SELECT a,b FROM S
WHERE a=1;
```

S	<u>a</u>	b

```
ALTER TABLE S
DROP PRIMARY KEY;
```

```
ALTER TABLE S
ADD PRIMARY KEY pk1(a)
ENFORCE;
```

Q′

```
SELECT a,b FROM S
WHERE a=1 and NOT EXISTS
 (SELECT * FROM S as s2
  WHERE S.a=s2.a AND
  S.b!=s2.b);
```

S	a	b

- ENFORCE: *limits query scope to non-violating tuples*

Negation...

- Intuition: *rewrite independently negative and positive part of the query*

Negation...

- Intuition: *rewrite independently negative and positive part of the query*

...

Q'

...

```
SELECT a,b FROM S
WHERE a=1 and NOT EXISTS
 (SELECT * FROM S as s2
  WHERE S.a=s2.a AND
  S.b!=s2.b);
```

```
RENAME COLUMN b
IN S TO x;
```

Negation...

- Intuition: *rewrite independently negative and positive part of the query*

...

Q'

...

```
SELECT a,b FROM S
WHERE a=1 and NOT EXISTS
 (SELECT * FROM S as s2
  WHERE S.a=s2.a AND
  S.b!=s2.b);
```

S	a	b

```
RENAME COLUMN b
IN S TO x;
```

Q''

```
SELECT a,x FROM S
WHERE a=1 and NOT EXISTS
 (SELECT * FROM S as s2
  WHERE S.a=s2.a AND
  S.x!=s2.x);
```

S	a	x

Negation...

- Intuition: *rewrite independently negative and positive part of the query*

...

...

Q'

```
SELECT a,b FROM S
WHERE a=1 and NOT EXISTS
 (SELECT * FROM S as s2
  WHERE S.a=s2.a AND
  S.b!=s2.b);
```

S	a	b

Q''

```
SELECT a,x FROM S
WHERE a=1 and NOT EXISTS
 (SELECT * FROM S as s2
  WHERE S.a=s2.a AND
  S.x!=s2.x);
```

S	a	x

```
RENAME COLUMN b
IN S TO x;
```

This application of Chase & Back-Chase is sound but not complete

So far...

	Structural Evolution	Integrity Constraints Evolution
Data	✔	✔
Queries	✔	✔
Updates	✘	✘

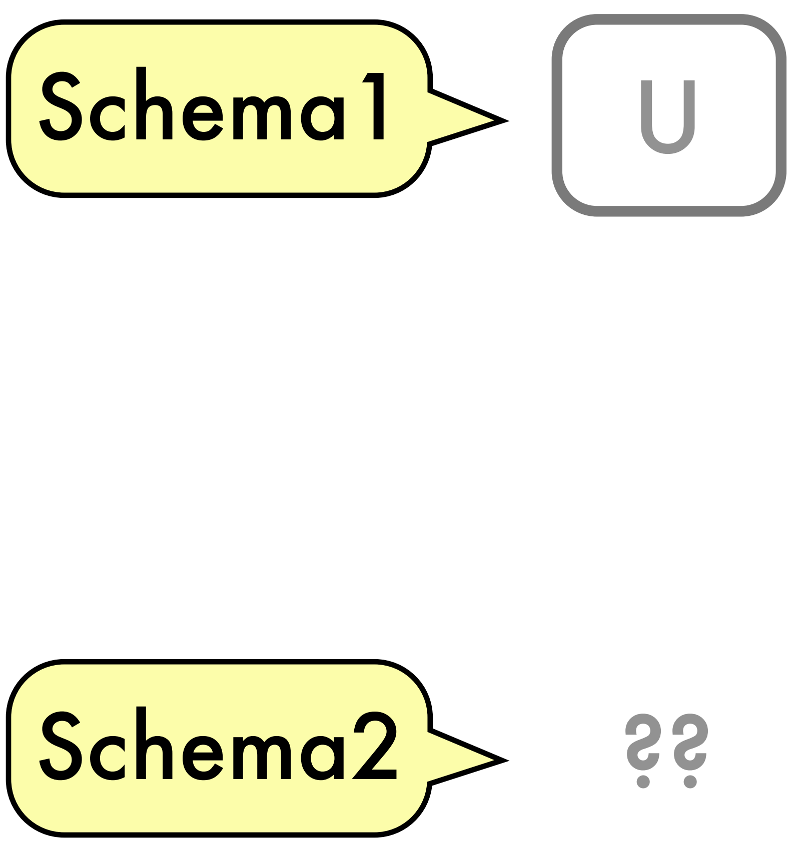
- Introduce ICMOs, Adapted SMOs
- Extended Query Rewriting Engine (ICMOs + neg.)

Update Rewriting (through SMOs)

- Intuition: *reuse query rewriting engine to tackle update rewriting*

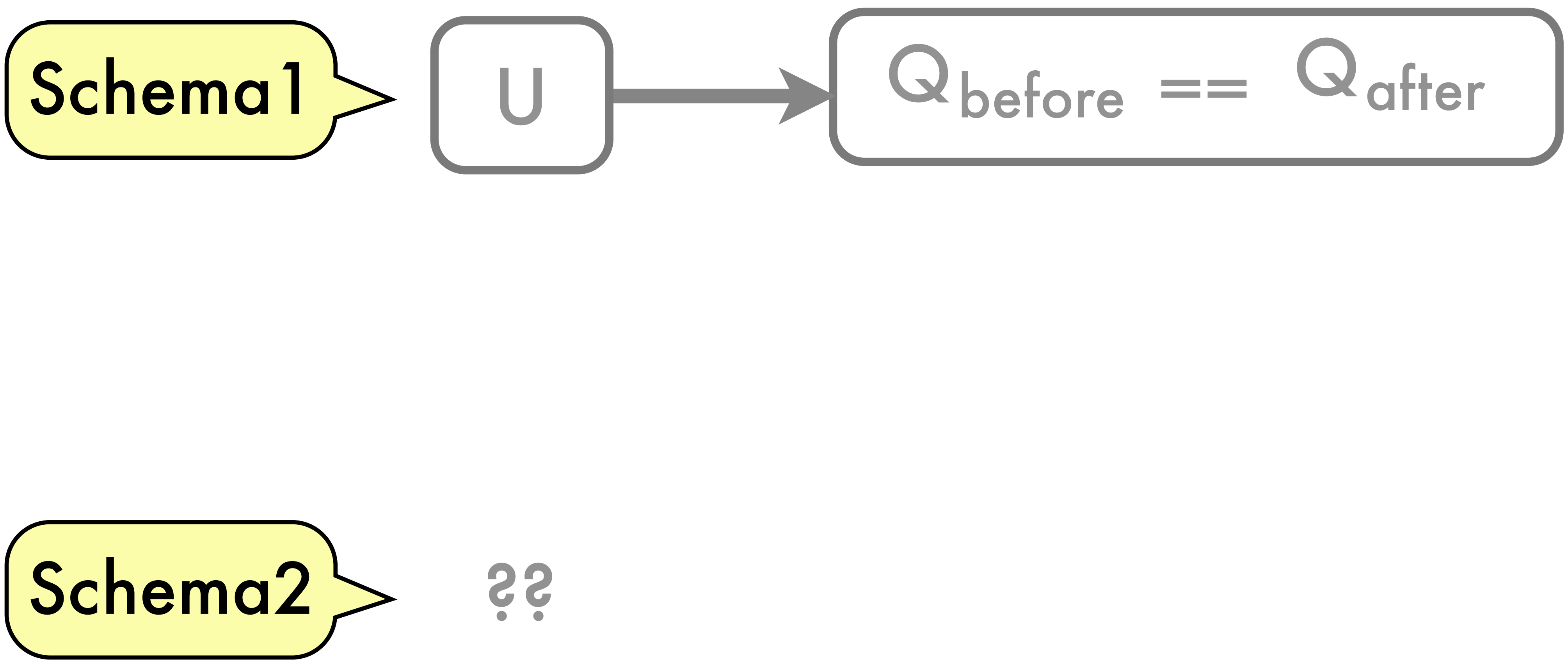
Update Rewriting (through SMOs)

- Intuition: *reuse query rewriting engine to tackle update rewriting*



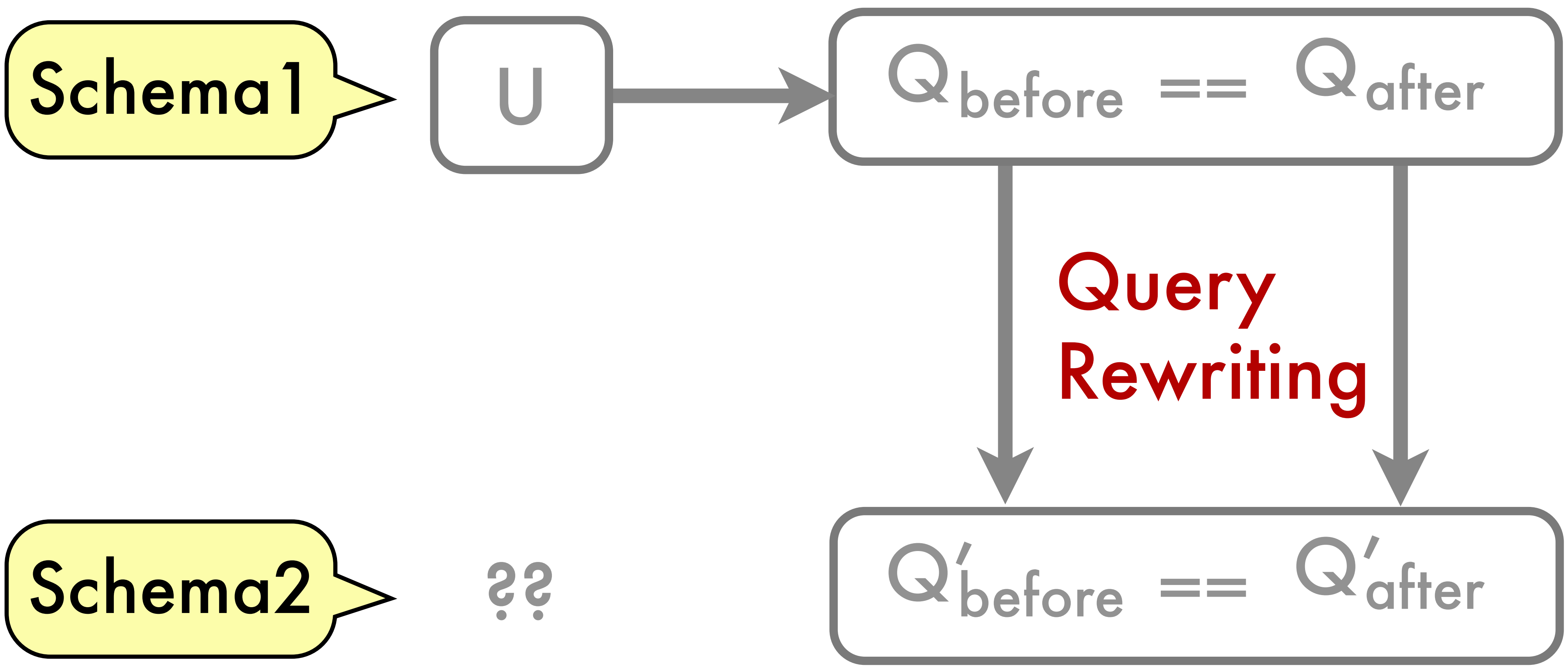
Update Rewriting (through SMOs)

- Intuition: *reuse query rewriting engine to tackle update rewriting*



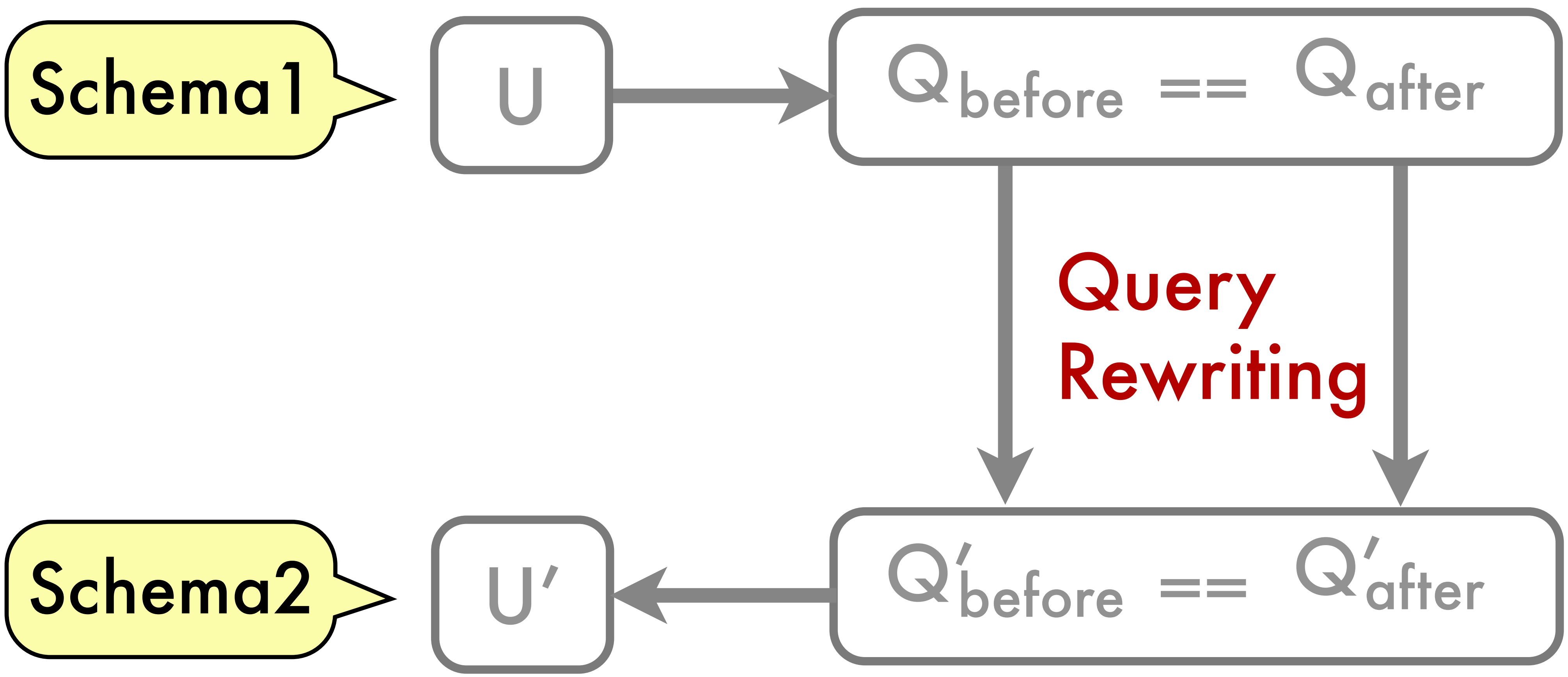
Update Rewriting (through SMOs)

- Intuition: *reuse query rewriting engine to tackle update rewriting*



Update Rewriting (through SMOs)

- Intuition: *reuse query rewriting engine to tackle update rewriting*



A New Update Representation

- Intuition: *represent updates as (equivalence between) queries, exploit query rewriting*

$$\boxed{U} \longrightarrow \boxed{Q_{before} == Q_{after}}$$

A New Update Representation

- Intuition: *represent updates as (equivalence between) queries, exploit query rewriting*

U → Q_{before} == Q_{after}

```
UPDATE S
SET b=7
WHERE a=1;
```

```
SELECT a,7
FROM S
WHERE a=1;
 UNION
SELECT a,b
FROM S
WHERE a!=1;
```

==

```
SELECT a,b
FROM S;
```

Update Rewriting (through ICMOs)

- Intuition: *the policies specify popular special-cases of view-update problem*

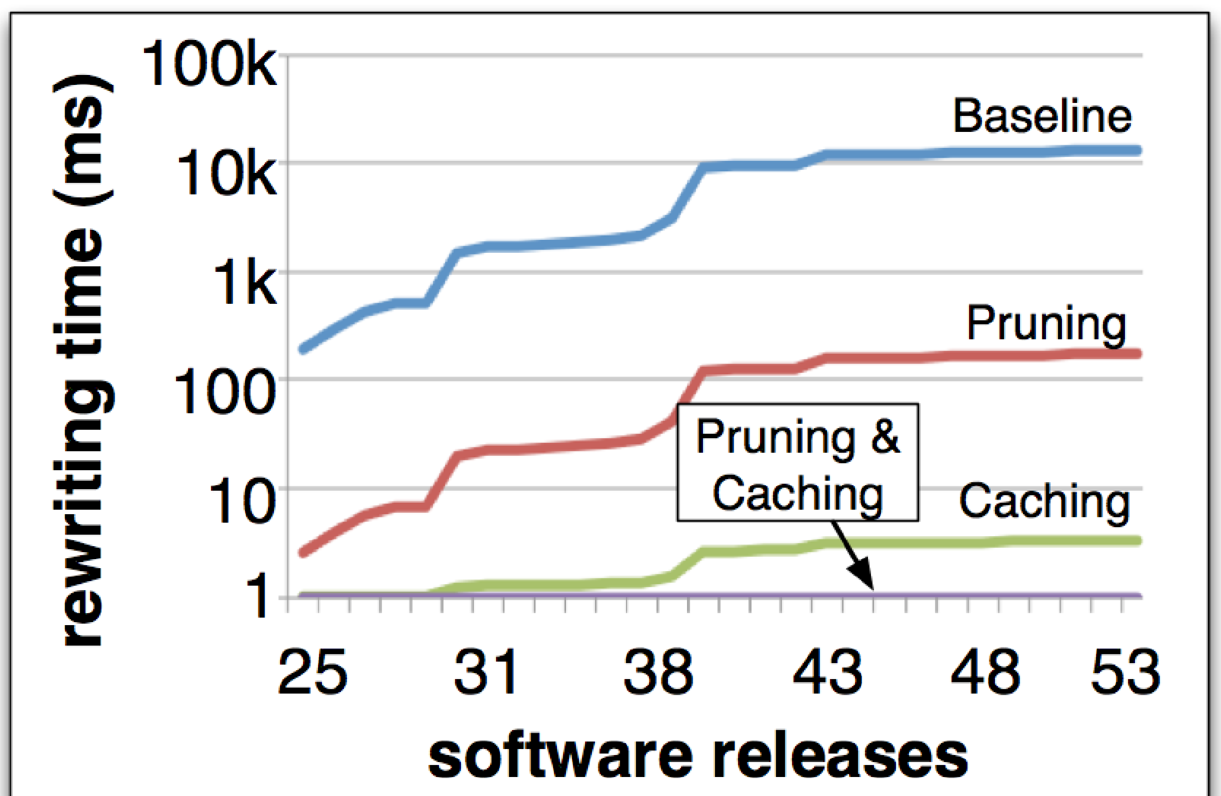
Update Rewriting (through ICMOs)

- Intuition: *the policies specify popular special-cases of view-update problem*

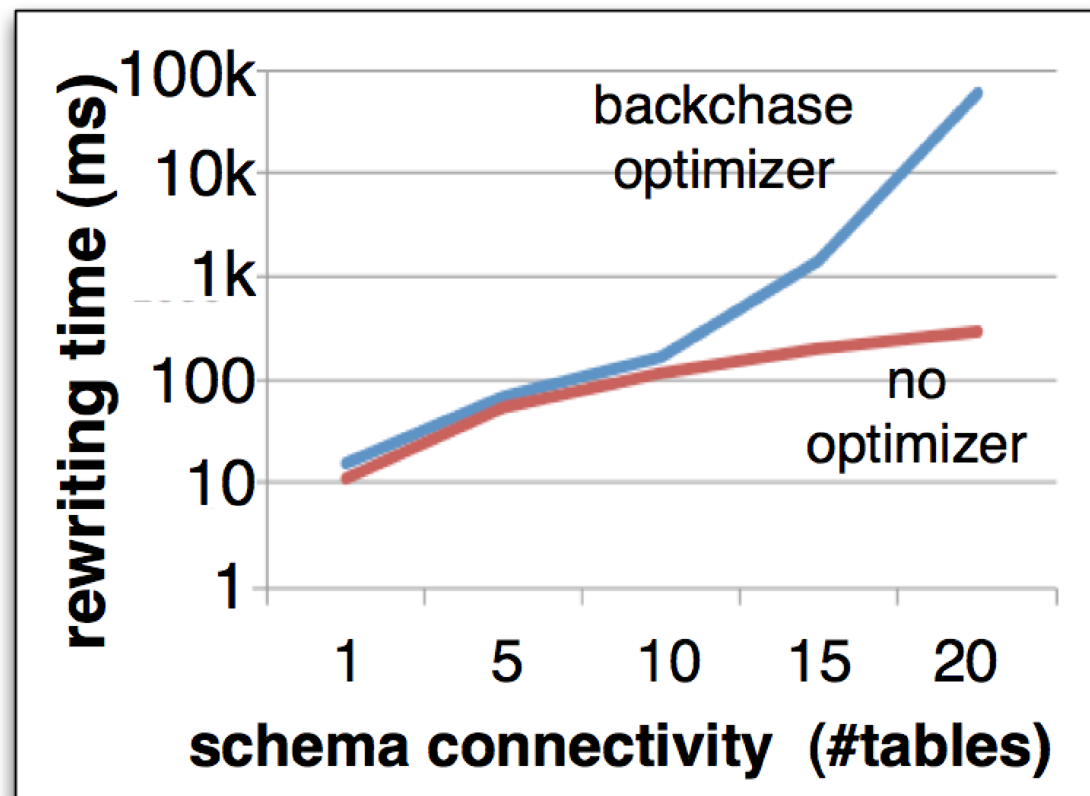
  - <policy>:
    - CHECK: *checks constraint before* ***and after*** *running update*
    - ENFORCE: *limits update scope to non-violating tuples,* ***checks violation-set is not changed***
    - IGNORE: *runs update as-is* ***(allows side effects)***

Optimization

- Challenge: *rewriting complexity depends on mapping size (foreign keys and ICMOs make things harder)*
- *Solution: extract templates, cache rewritings*



Ensembl genetic DB



Synthetic Dataset

Wikipedia hit/miss ratio: up to 88M

Conclusion

- Prism++ is a **high-performance** practical system supporting DB schema-evolution:

	Structural Evolution	Integrity Constraints Evolution
Data	✓	✓
Queries	✓	✓
Updates	✓	✓

For More info contact me:
krl@yahoo-inc.com

Wikipedia

- Solution: *effectiveness of template caching*

Statement type	number of templates	avg hit/miss ratio	max hit/miss ratio
update	142	5,661.21	80,870
select	1294	248,005.41	88,740,689
select*	610	526,096.72	88,740,689

*with improved template extraction factorizing DB names.

Statements	execution time	rewriting time	overhead
S1	77.37 ms	1 ms	1.29%
S2	21.674 ms	1 ms	4.6%
S3	48.2 ms	1 ms	2.07%

Chase & BackChase

- Intuition: *behind the scene Disjunctive Embedded Dependencies and chase-based rewritings*

```
JOIN TABLE gene,g_descr
INTO gene
WHERE gene.id=g_descr.id
```

inverse

```
DECOMPOSE TABLE gene
INTO g_descr(id,description),
     gene(id,type,region,start,end);
```

derive mapping

mapping (S2-S3)

$$gene_3(x,y,z,w,k,l) \implies gene_2(x,y,z,w,k), g\_descr(x,l)$$
$$gene_2(x,y,z,w,k,l), g\_descr(x,l) \implies gene_3(x,y,z,w,k,l)$$

```
SELECT description
FROM gene g, g_descr gd
WHERE g.id=gd.id AND
      g.region=1;
```

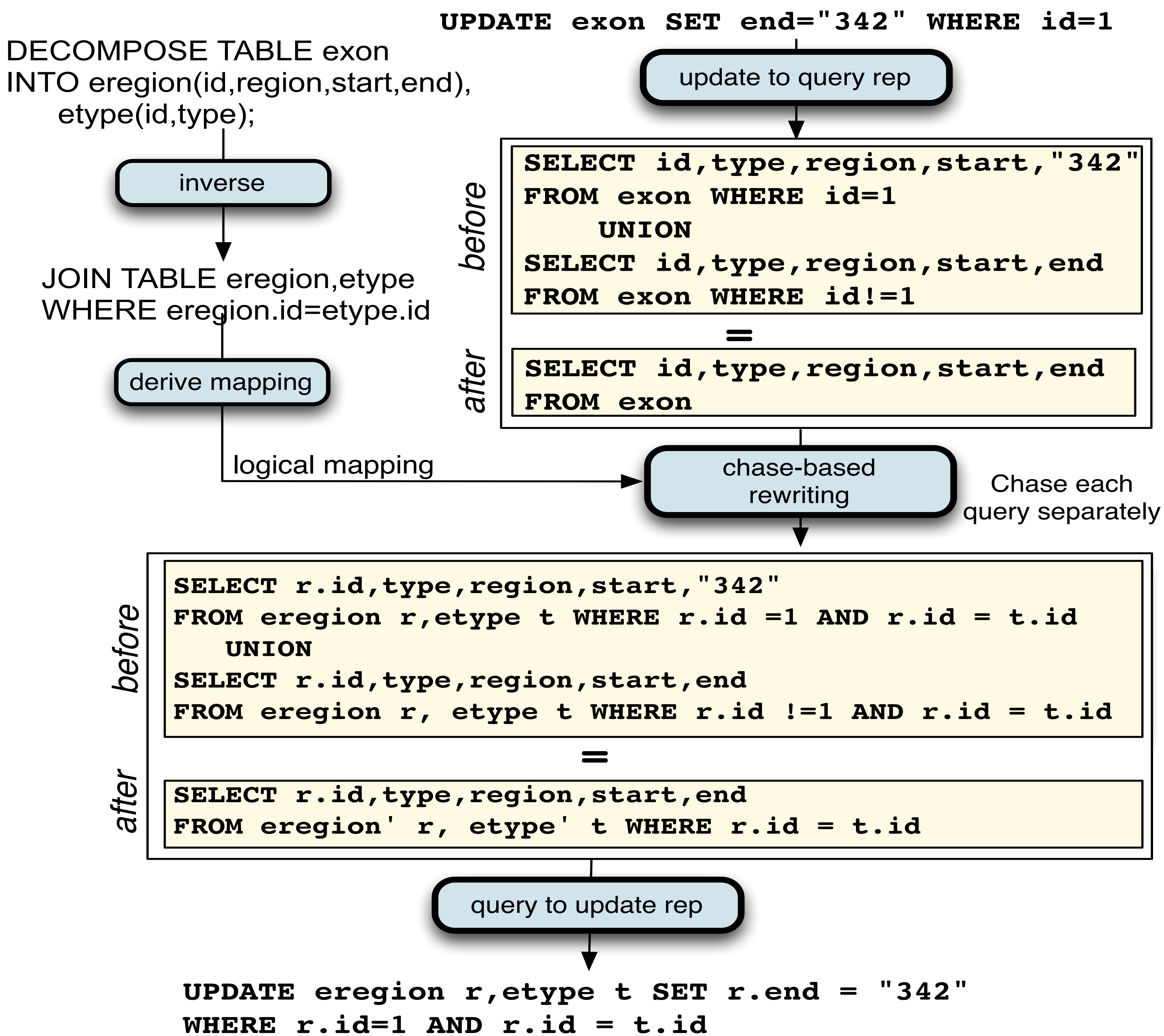
input query (on S2)

chase-based rewriting

rewritten query (on S3)

```
SELECT description
FROM gene g
WHERE g.region=1;
```

Update Rewriting through SMO: Example



Update Rewriting ICMOs: Example

ALTER TABLE exon
DROP PRIMARY KEY pk1

inverse

ALTER TABLE exon
ADD PRIMARY KEY pk1(id)
CHECK

```
INSERT INTO exon VALUES (1,2,3,4,5)
```

ICMO
rewriting

```
@pre = SELECT * FROM exon e,exon e2
       WHERE e.id=e2.id AND e.rank=e2.rank AND
            (e.type!=e2.type OR e.start!=e2.start OR e.end!=e2.end);
@post = SELECT * FROM exon e WHERE e.id=1;

IF(isempty(@pre)&& isempty(@post)) INSERT INTO exon VALUES(1,2,3,4,5)
ELSE RETURN ERROR;
```