RADICAL 2010

RADICAL 2010

Wiki
Everyone can use to
Store,
Organize,
Manage and
Exchange data

RADICAL 2010

Wiki
Everyone can use to
Store,
Organize,
Manage and
Exchange data

James Cheney
University of Edinburgh

Joint work with
Peter Buneman, Sam Lindley, Heiko Mueller (UoE)
Michael Benedikt (Oxford)

Or:

How to train your database wiki

Or:

~~*How to train your database wiki*~~

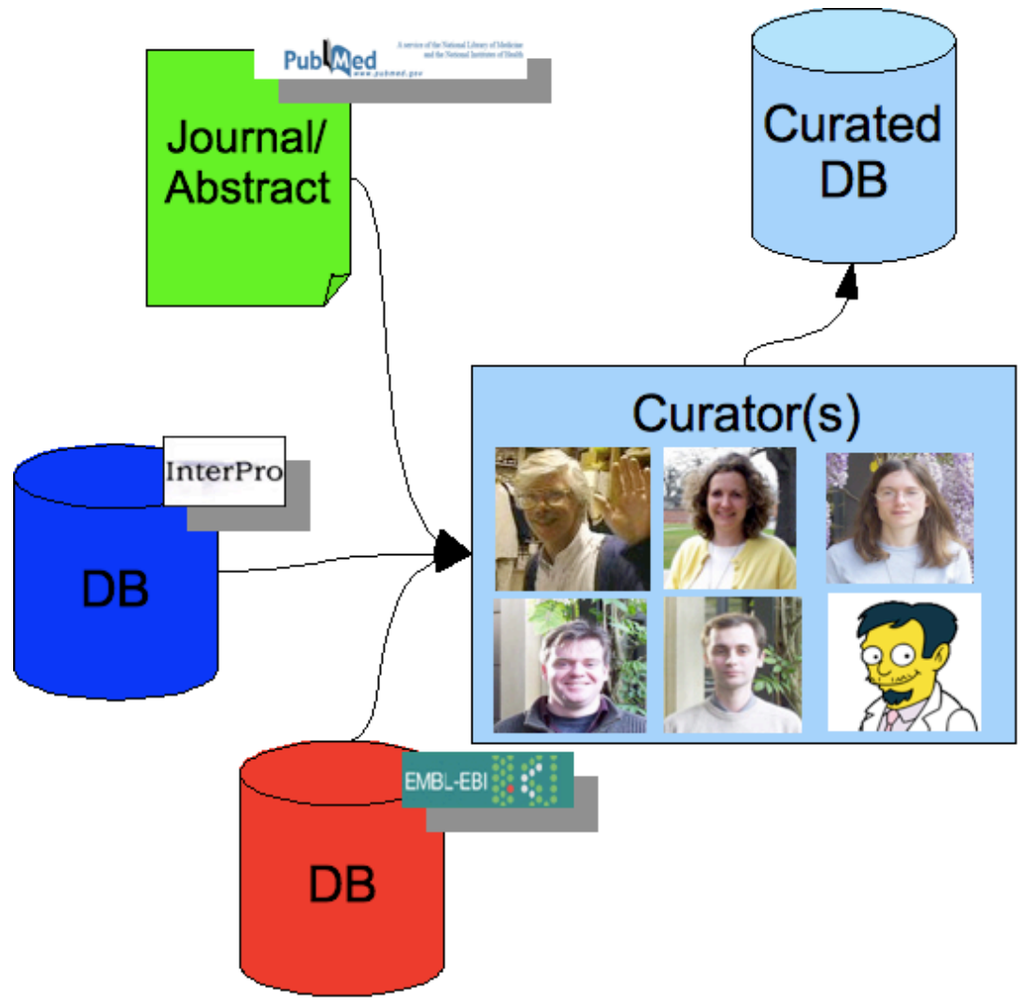
Or:

~~*How to train your database wiki*~~

This is what happens when
James has too much coffee

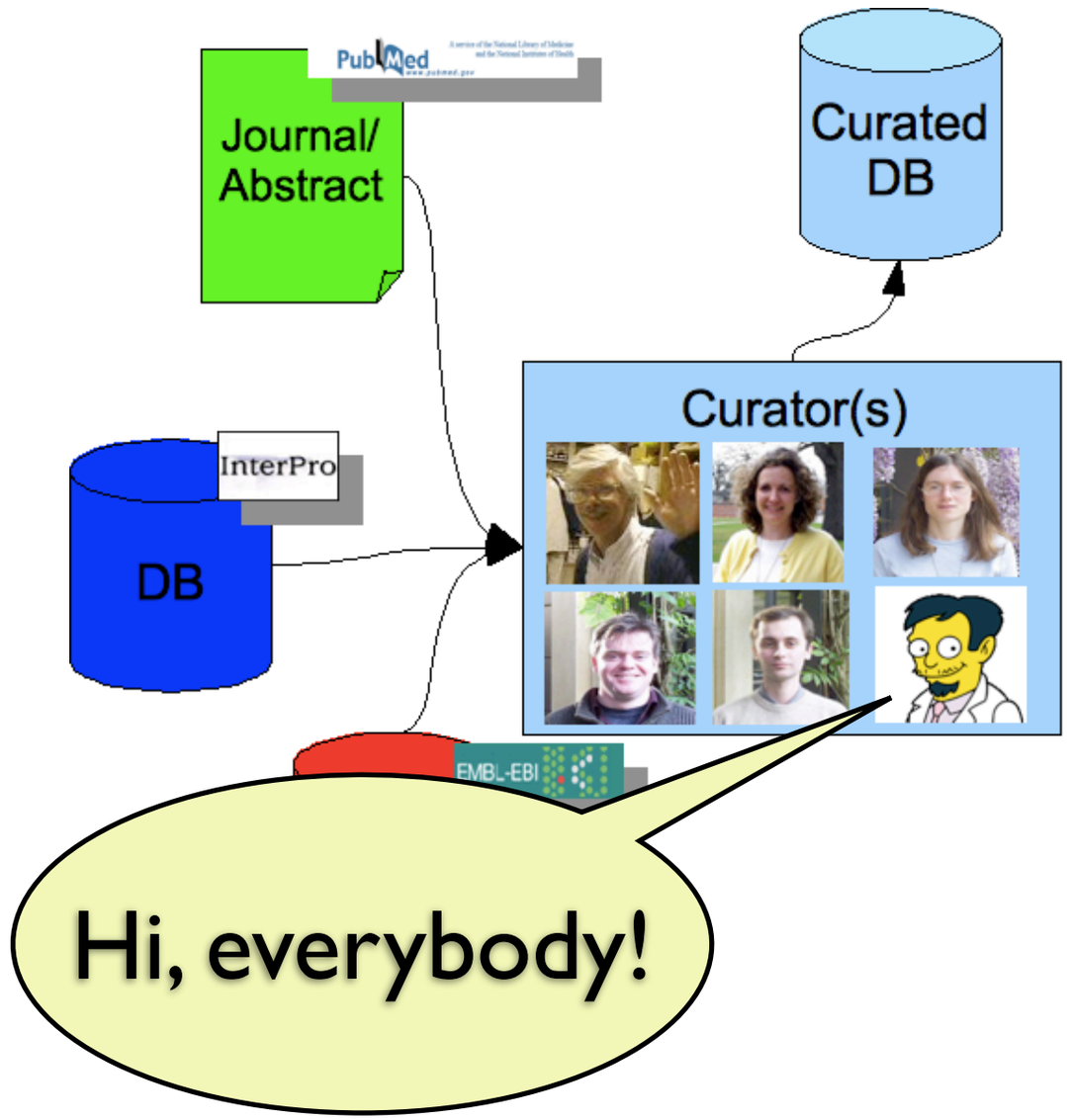
Curated databases

- Created by manual effort
- Curators copy data from papers, other DBs
- Some sources unreliable
  - some curators too



Curated databases

- Created by manual effort
- Curators copy data from papers, other DBs
- Some sources unreliable
  - some curators too



Problem

- We know basically what to do
- "Curated databases" should provide built-in:
  - archiving [Buneman et al. 04, 08, ...]
  - provenance [Buneman et al. 01, 06, 07, 08, ...]
  - annotation [Geerts et al. 06, ...]
- But hard to convince users to do the right thing

Solution:
A database wiki

- Standard wiki stuff
  - WikiLinks, editable pages, brain-dead syntax, page history
- New: editable, (semi)structured data with
  - Transclusion (via queries embedded in pages)
  - Annotation (discuss data, propose changes)
  - Stable citation, copy/paste for wiki data
  - Archiving - record all past versions
  - Provenance - automagical (?)

Implementation

- Using Links
  - Other web programming languages would probably also work (but we have in-house expertise)
- Currently:
  - basic wiki stuff
  - "data tree" - editable via browser & persistent
  - path transclusions and type-based selection queries

Demo

and now for something completely different:

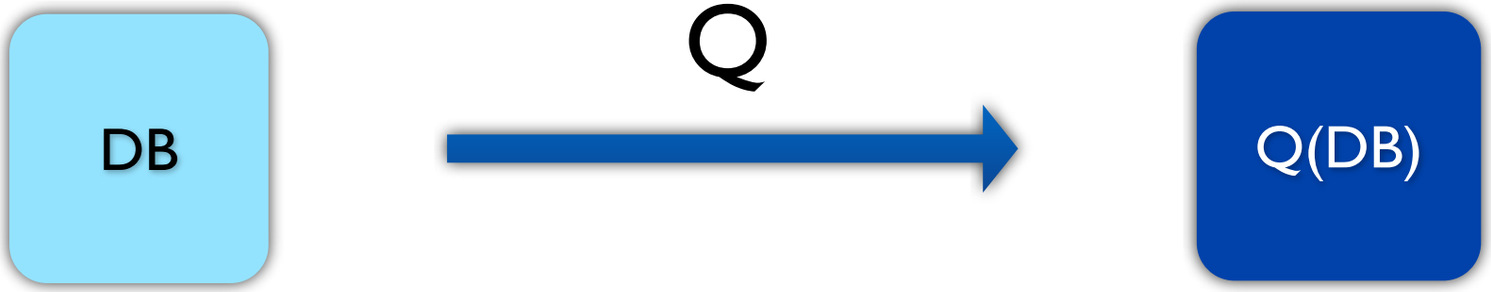
Independence Analysis for semistructured data

Motivation

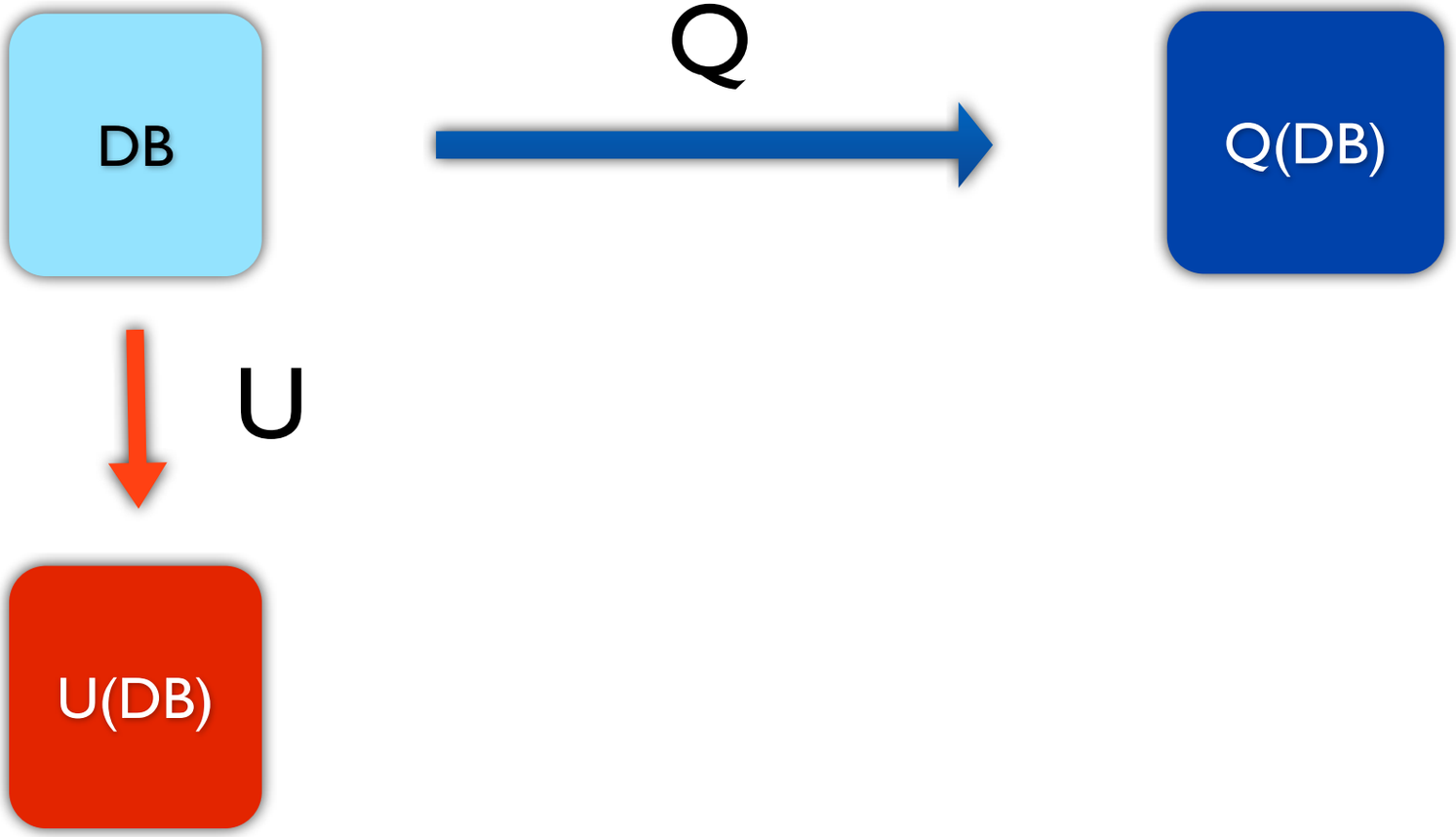
- Suppose we have multiple (cached) pages
  - expressed by (XQuery/XPath) queries Q_1, Q_2, ...
- When the database wiki is updated:
  - Which queries may be affected?
  - If we can determine (quickly) that queries and updates are *independent*
  - then can keep using cached version of unchanged pages

Query-update independence

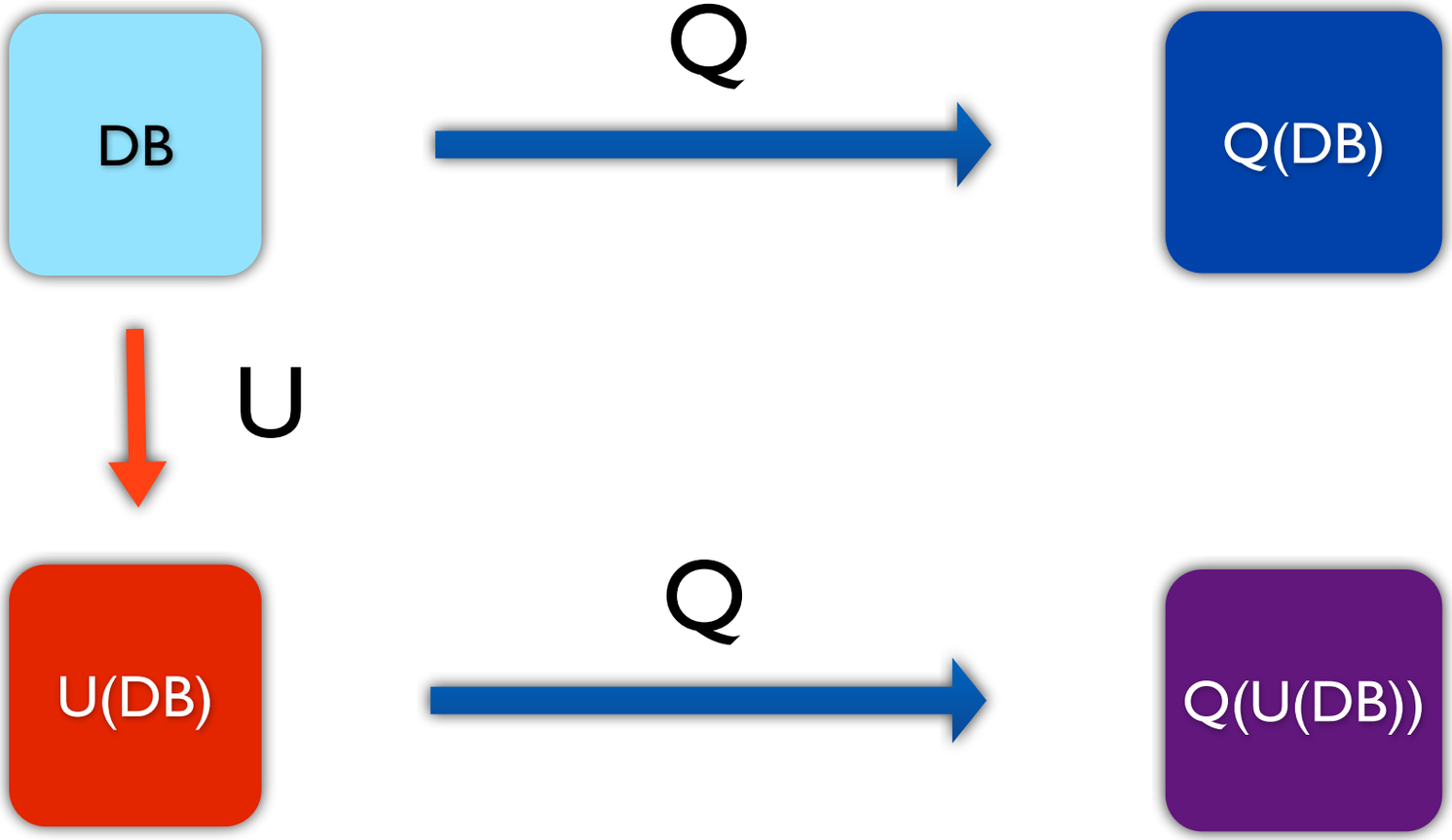
Query-update independence



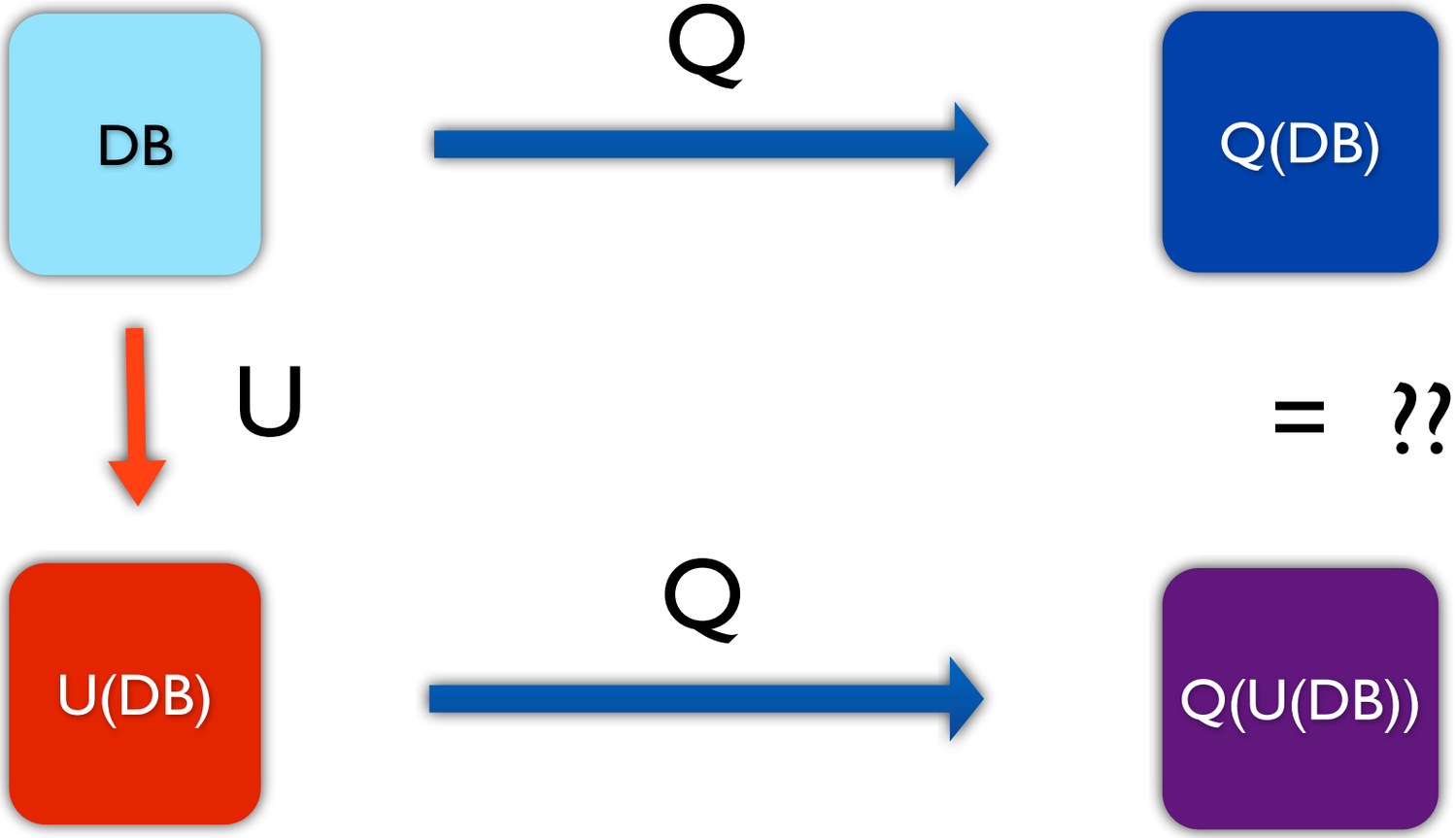
Query-update independence



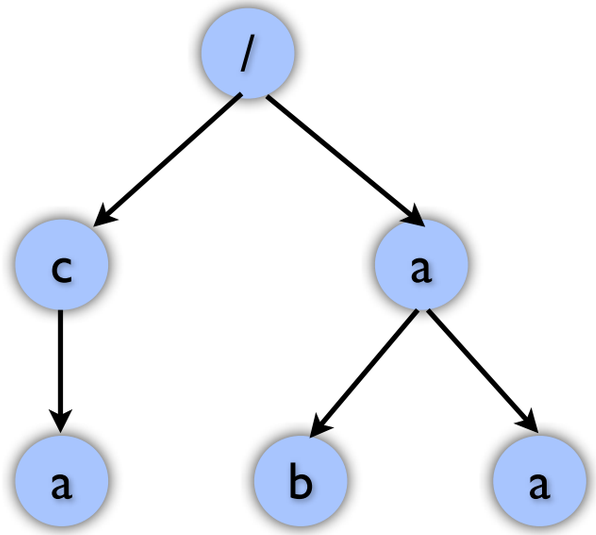
Query-update independence



Query-update independence

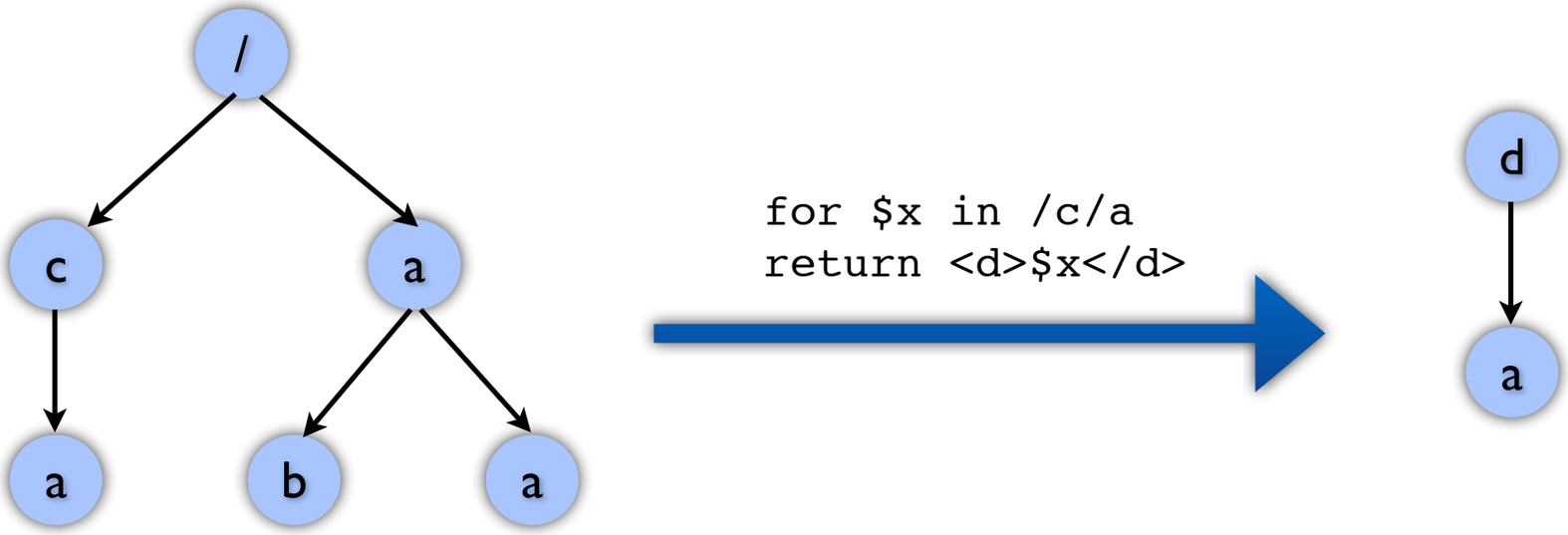


Independence example

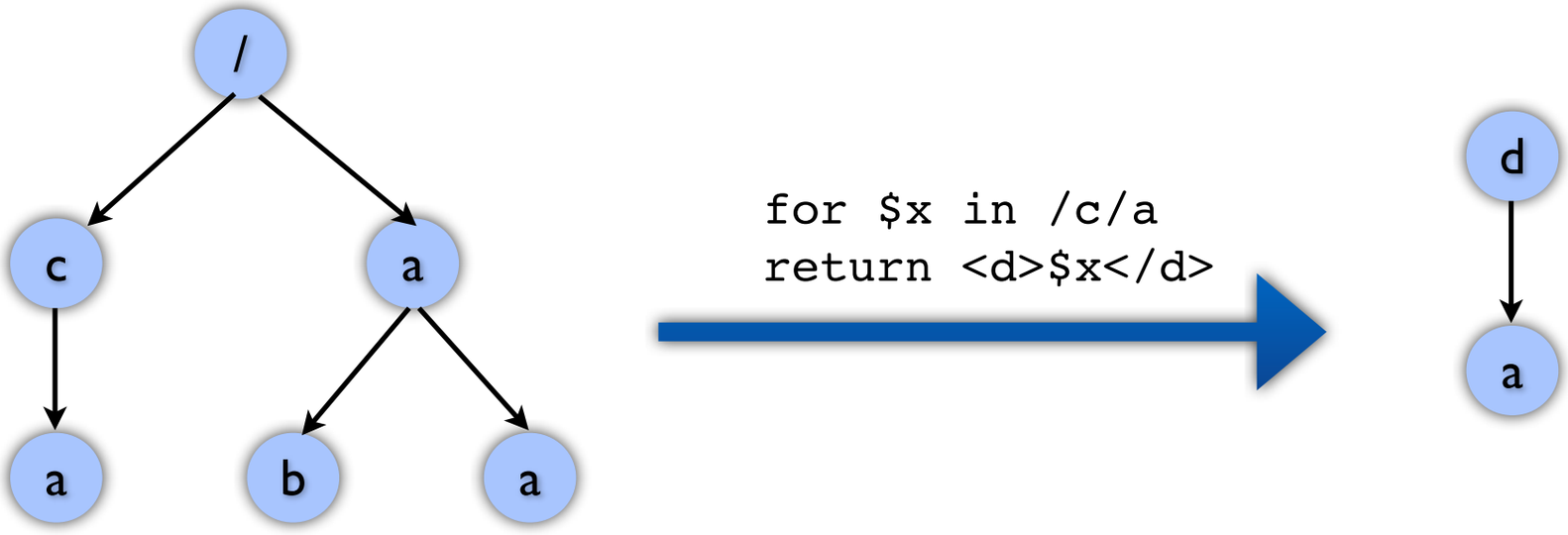



```
for $x in /c/a
return <d>$x</d>
```

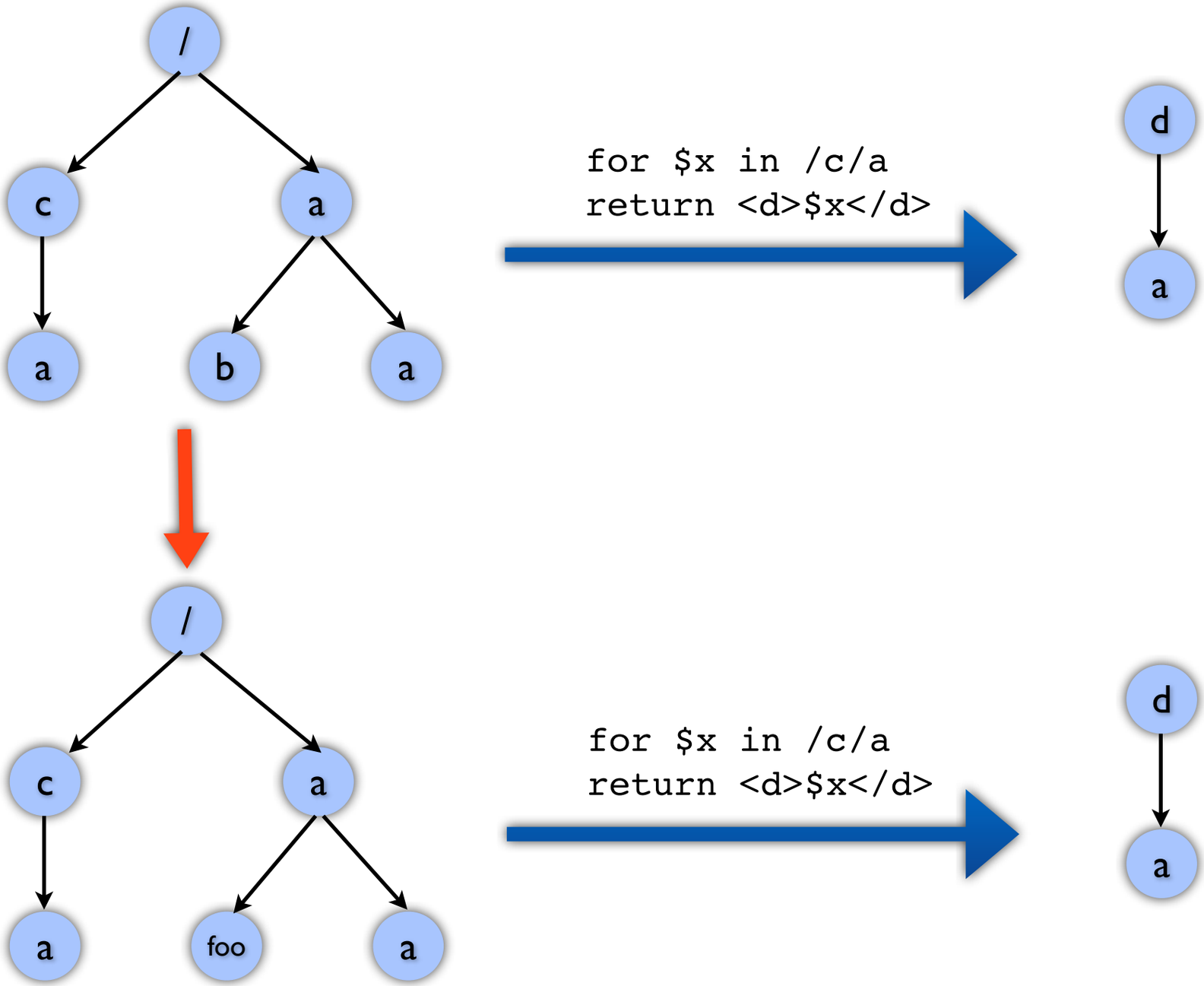
Independence example



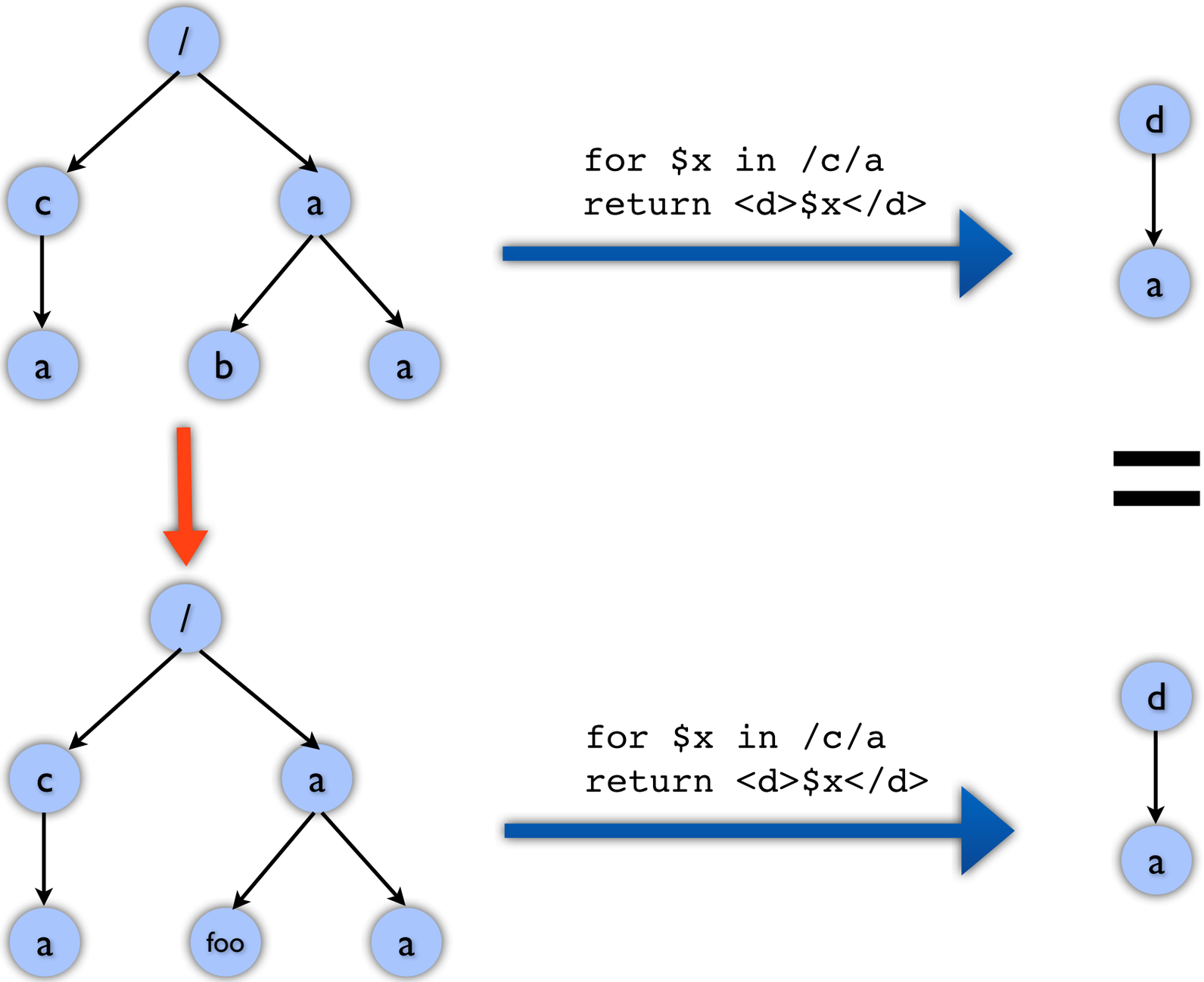
Independence example



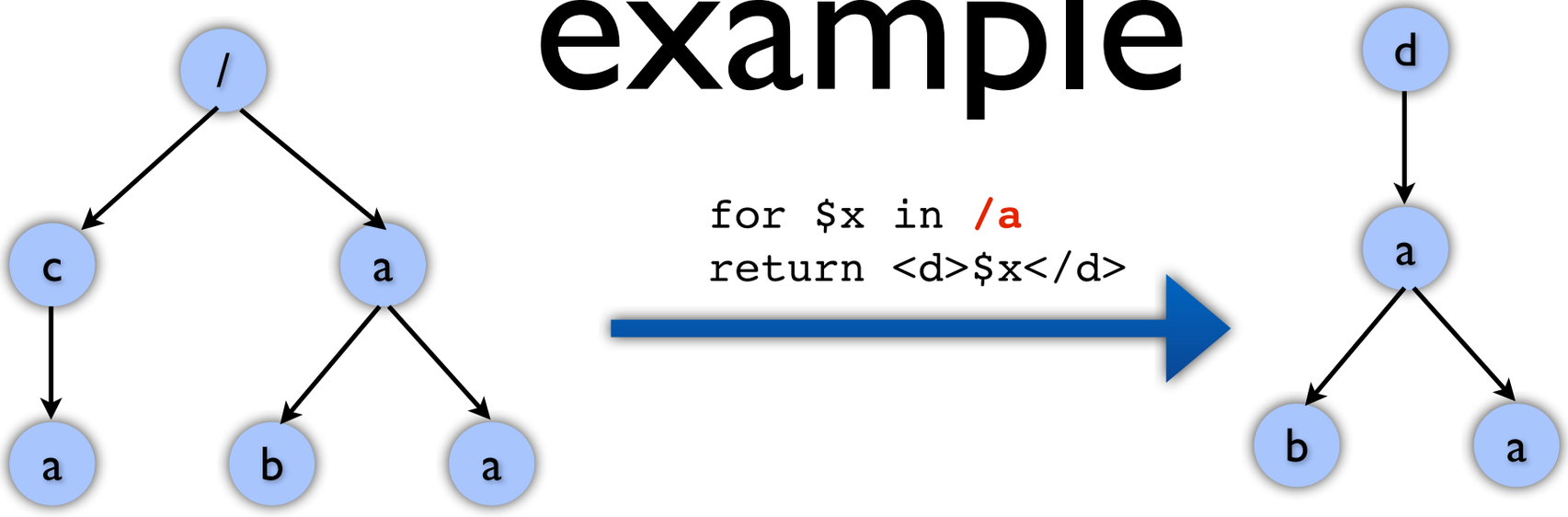
Independence example



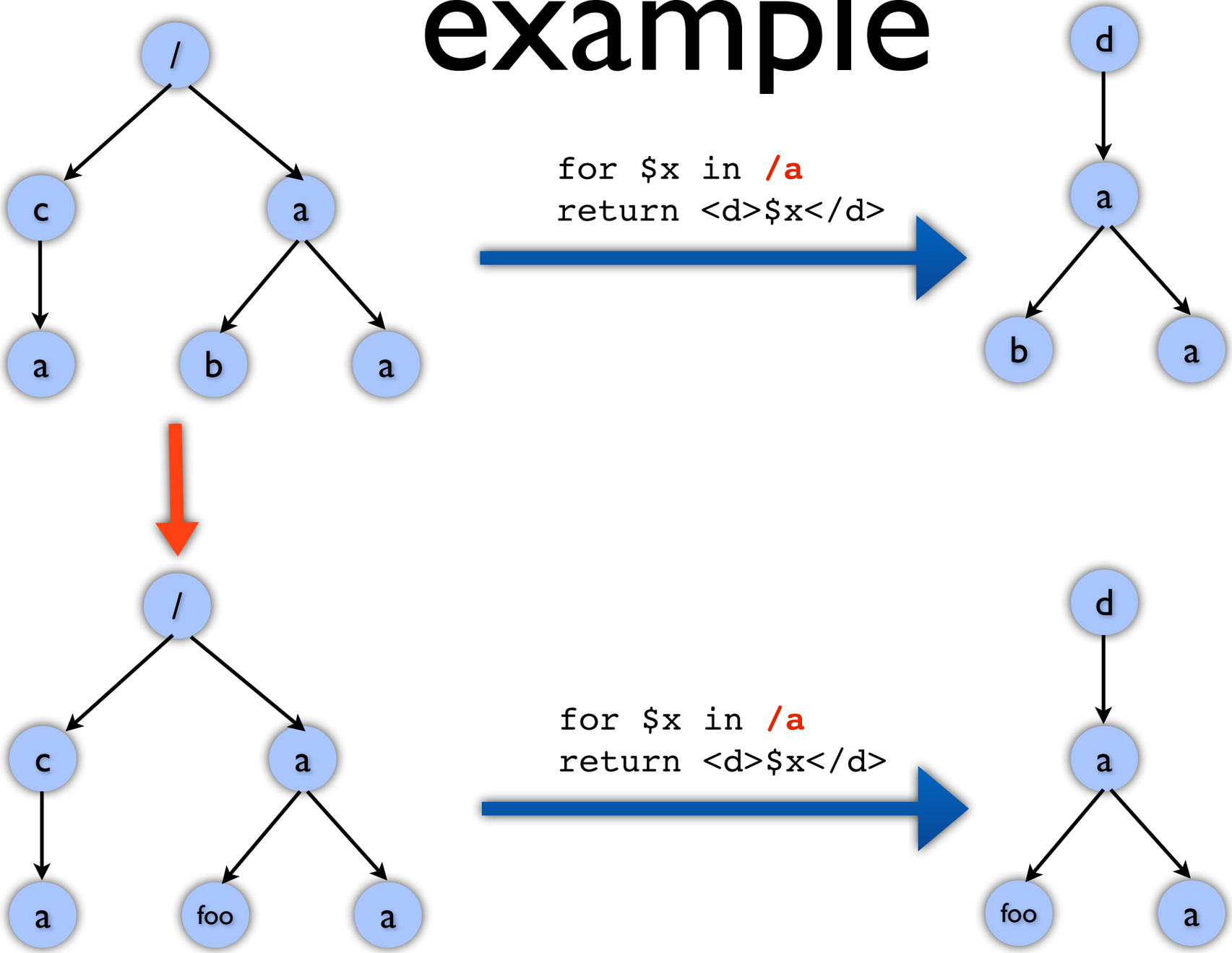
Independence example



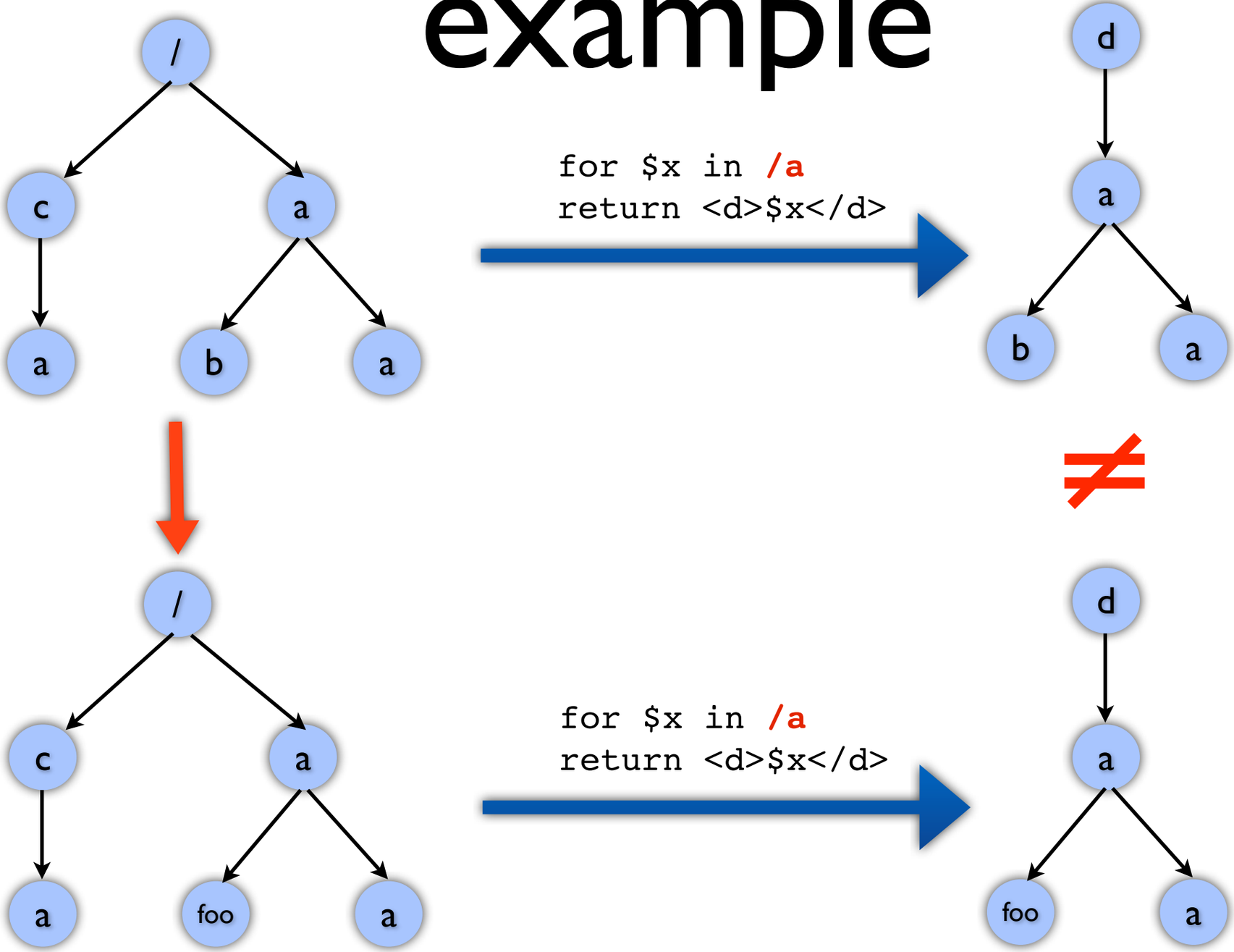
Independence **non-**example



Independence **non-**example



Independence non-example



Prior work

- [Benedikt and C 09]: static independence analysis based on schema (reg. exp. types)
  - Showed effective for avoiding view maintenance
- Problem: Useless if you don't have a schema
  - Some prior work on path-based "XML projection" [Marian & Simeon 03,...], "commutativity analysis" [Ghelli et al. 08]
  - But doesn't quite solve independence problem

Current work

- [Benedikt and C 2010]: Use **queries** to statically describe the set of updates that "may destabilize" the query
  - Call this $\Delta(Q)$, the **destabilizer** (or **antiprovenance**) of Q
- *Targets(U)* disjoint from $\Delta(Q)$ implies Q independent of U
- Can be just as effective, without a schema

Key subproblem: XPath intersection analysis

- In the absence of schema, use **paths** to statically describe sets of nodes
  - cf. [Ghelli, Simeon & Rose 2008], others
  - Intersection of **downward** paths is $O(n^2)$
  - But general problem is NP-hard

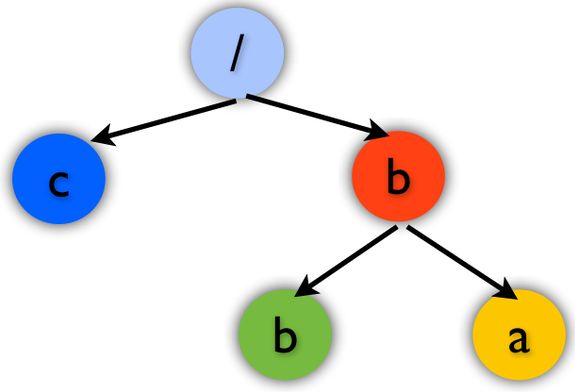
Solvers to the rescue?

- There are solvers for decidable tree logics
- `MONA` (decides MSO(Tree))
  - Somewhat unpredictable, needs tuning
- [Geneves et al. 07]: Modal mu-calc solver
  - Source not available
  - Optimized version not yet available

A special case

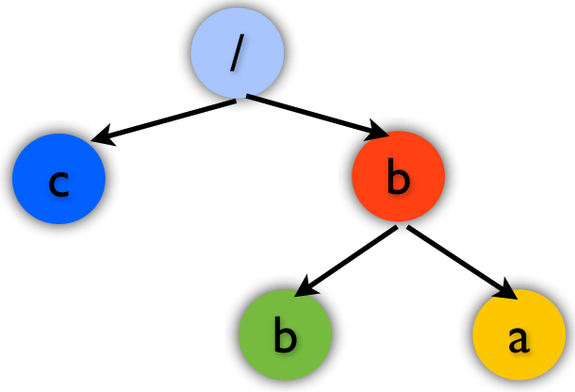
- Our approach: novel (apparently) reduction from EFO(Tree) to EFO(N, <).
- Most SMT solvers are **very** good at EFO(N,<)-SAT (typically complete)
- Typically faster than `MONA` or Geneves solver on our path intersection/ independence benchmark
  - but handles a much weaker theory

Idea



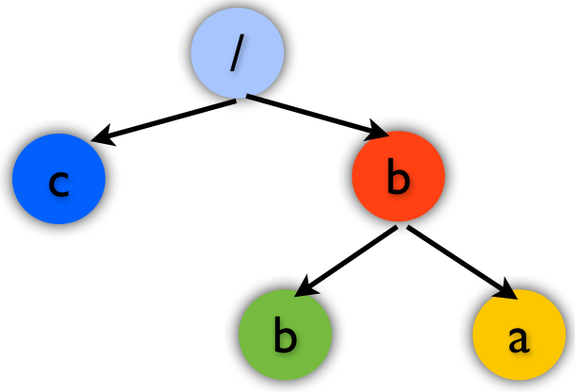
- `Sibling(x,y)`
- `=>`
- `x.post < y.pre`

Idea



- `Desc(x,y)`
- `=>`
- `x.pre < y.pre & x.post > y.post`

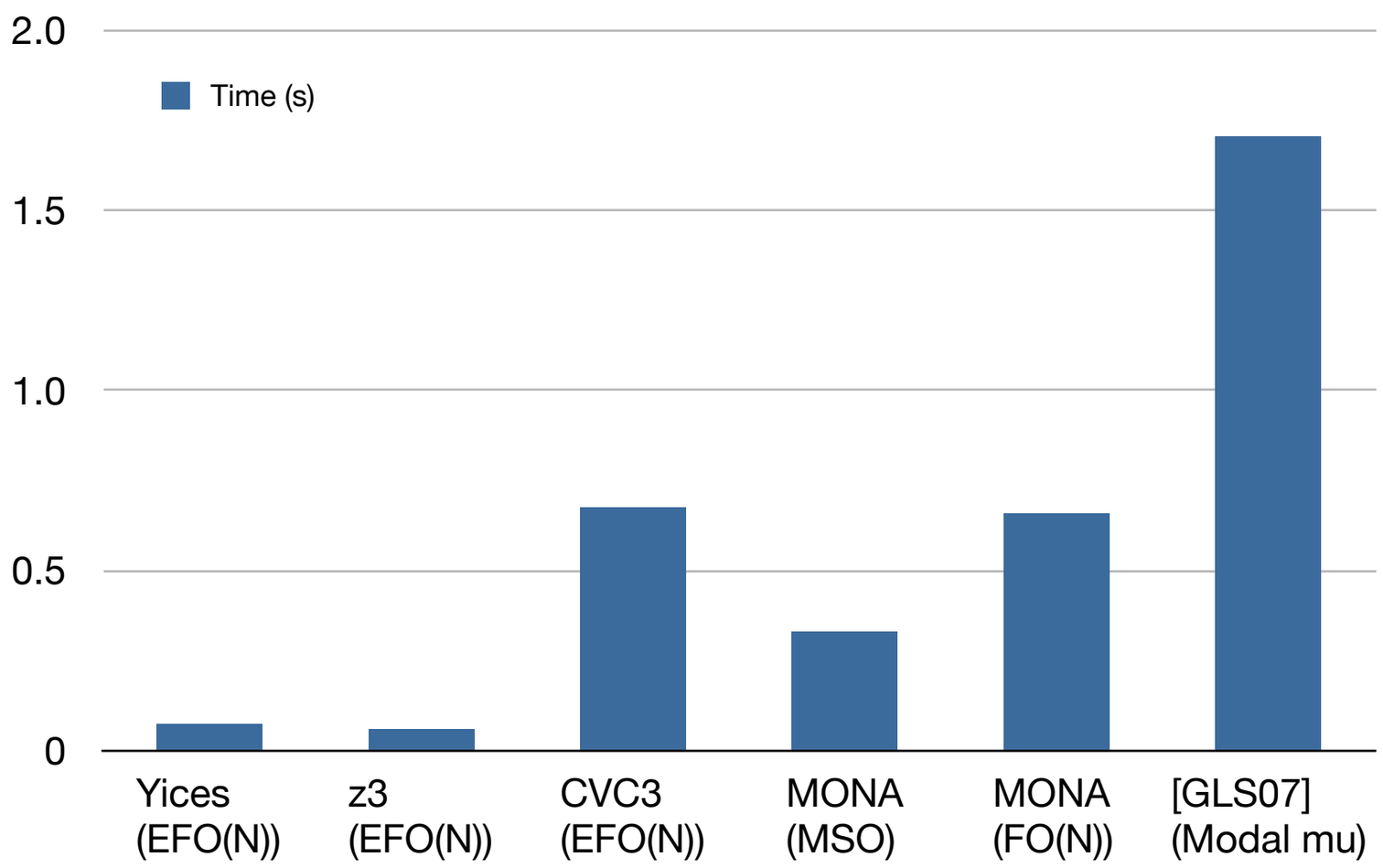
Idea



1 2 3 4 5 6 7 8 9 10

```
Child(x,y) =>
Desc(x,y) & not (Desc(x,x1) & Desc(x1,y)
          & ...
          & not (Desc(x,xn) & Desc(xn,y)
```

Comaprison for one typical problem



(not a comprehensive experiment!)

Experimental results

- Destabilizer-based independence analysis is just as effective as schema-based
  - succeeds/fails on different queries
  - fast enough (using yices) to yield savings
- Close to exact
  - < 1% false positives on benchmark of over 500 problems
  - (hand classified)

Respectable charts and figures

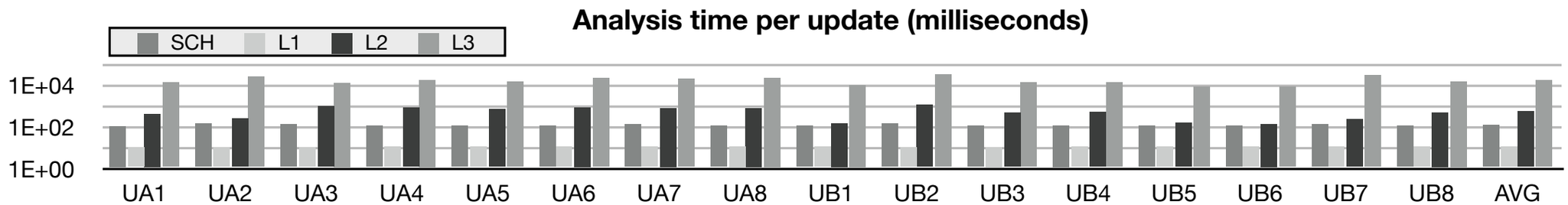


Figure 5: Running times for the generic analysis, in milliseconds (logarithmic scale), broken down by update and analysis level.

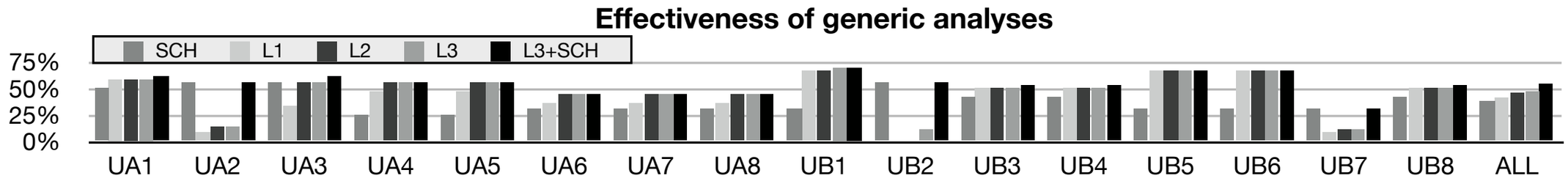


Figure 6: Effectiveness of the generic analysis, expressed as a percentage of query-update pairs determined independent, broken down by update and by analysis level.

	SCH	L_1	L_2	$L_2 + SCH$
1.1MB	8.3%	10.5%	8.3%	10.9%
2.3MB	11.0%	14.5%	14.9%	20.1%

Table 1: Maintenance time savings across whole benchmark

Respectable charts and figures

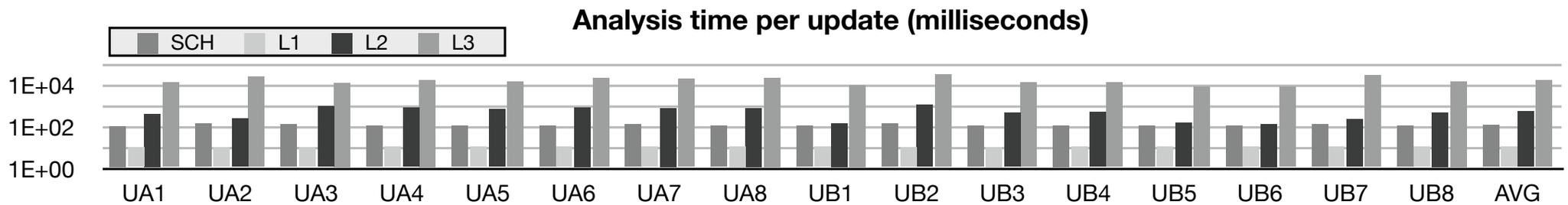


Figure 5: Running times for the generic analysis, in milliseconds (logarithmic scale), broken down by update and analysis level.

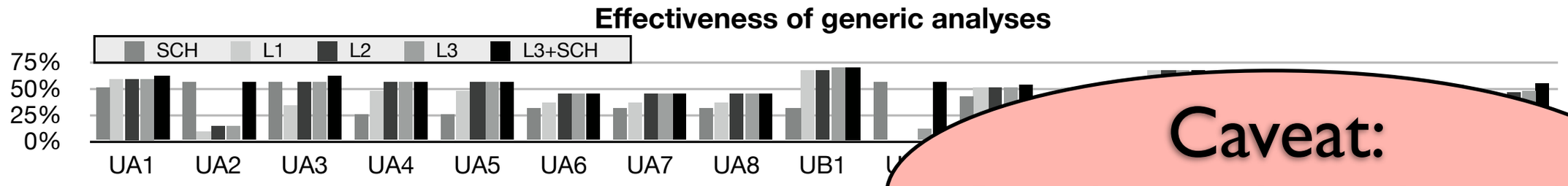


Figure 6: Effectiveness of the generic analysis, expressed as a percentage o… down by update and by analysis level.

Caveat:
Synthetic benchmark

	SCH	L_1	L_2	$L_2 + SCH$
1.1MB	8.3%	10.5%	8.3%	10.9%
2.3MB	11.0%	14.5%	14.9%	20.1%

Table 1: Maintenance time savings across whole benchmark

Limitations/future work

- What if you do have a schema?
  - Naive joint path and schema analysis works OK
  - Smarter: destabilizer intersection modulo schema
- Schemas not expressible exactly in EFO(N)
  - Can SMT approach be extended to handle schemas?
- What about incremental maintenance?
  - ideally, want to combine static and dynamic approaches

Conclusions

- Database wikis will be A.W.E.S.O.M.E.
- and will need good high-level web, XML and constraint programming tools
- Need to solve tree constraints quickly using SMT solvers (or other decision procedures?)
- in order to make R.A.D.I.C.A.L. advances in techniques for database curation