Distributed data management with the rule-based language: *Webdamlog*

Ph.D. defense

Émilien ANTOINE

Supervisor: Serge ABITEBOUL

Webdam Inria ENS-Cachan Université de Paris Sud

December 5th, 2013

Focus of this thesis

Allow the Web users to manage their personal data in place

Two main aspects:

- personal data are distributed
- Web users want to automate tasks and they are not programmers

Problem overview

We are interested in all kinds of data of a Web user

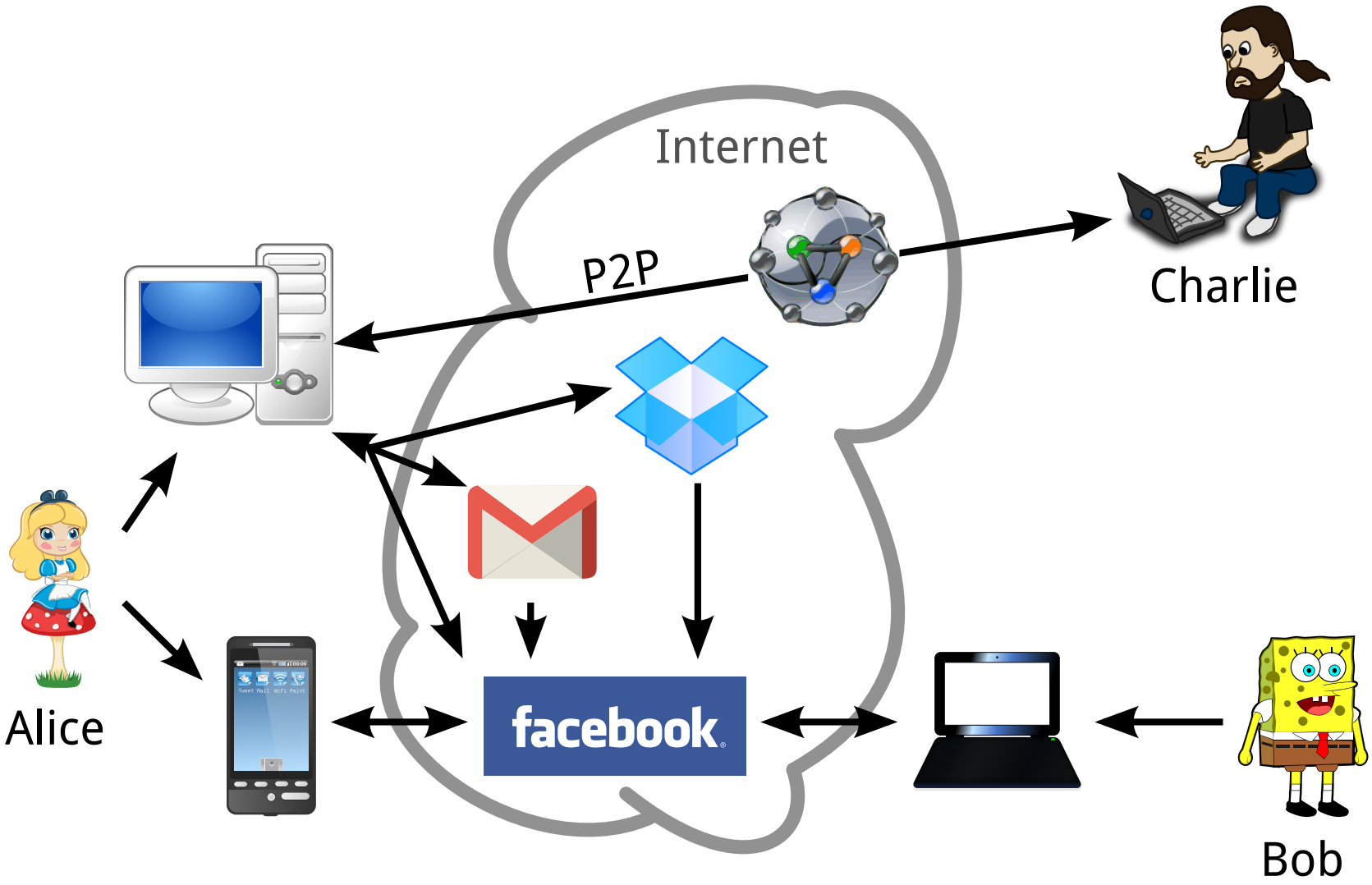
personal data	photos, movies, music, emails
social data	annotations, recommendations, social links
localization	bookmarks, phone numbers
privacy	logins/passwords, ssh keys

Data of users are heterogeneous by nature

Data are distributed on several

devices	laptops, smartphones, Internet boxes, cloud-storage, ...
systems	Facebook, Picasa, Gmail, ...

Typical distribution of knowledge



Example of distributed task on personal data

Alice has

- a blog on Wordpress.com to publish movie reviews
- a Facebook and Gmail account to talk to friends
- a Dropbox account to share files

she wishes to advertise friends about new reviews and share the movie

Cumbersome tasks for humans

- remember each login/password
- sign on each website
- use each GUI

Solutions

- hope that a system, e.g. Facebook, provides an appropriate wrapper
- try to specify it with a system such as `ifttt.com` "if this then that"
- if everything fails, write a script (only for hackers)

Example of distributed task on personal data

Alice has

- a blog on Wordpress.com to publish movie reviews
- a Facebook and Gmail account to talk to friends
- a Dropbox account to share files

she wishes to advertise friends about new reviews and share the movie

Cumbersome tasks for humans

- remember each login/password
- sign on each website
- use each GUI

Solutions

- hope that a system, e.g. Facebook, provides an appropriate wrapper
- try to specify it with a system such as `ifttt.com` “if this then that”
- if everything fails, write a script (only for hackers)

The Web as a distributed knowledge base

Give to Alice a system that

- use the Web service to manage her knowledge
- allow her to specify tasks
- hide the network and protocol issues

Principles underlying the approach

- Knowledge and computation are distributed on several peers
- These peers are autonomous (P2P)
- They are willing to collaborate (delegation)
- Knowledge from heterogeneous systems is integrated using “wrappers” in a mediation style

Representing knowledge in datalog

Facts in relations:

Friend

user	friend with
Alice	Bob
Alice	Charlie
Bob	Charlie

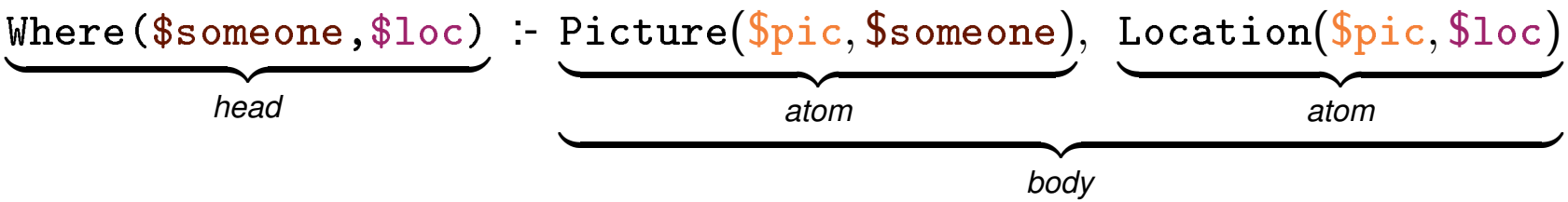
Picture

id	tag
pic1	Alice
pic2	Bob
pic3	Charlie

Location

pic id	location
pic1	Grenoble
pic2	Nantes
pic3	Rennes

Rules:



Representing knowledge in datalog

In datalog relations are either:

- extensional: `Friend`,`Picture`,`Location`
  - a list of facts stored in database
  - only in the body of datalog rules
- intensional: `Where`
  - a list of facts defined by rules ie. a *view*
  - appear at least once in the head of a rule

Datalog supports recursion

Recursion needed in graphs e.g. network path, social link

```
FoF($x,$y) :- Friend($x,$y)
FoF($x,$y) :- Friend($x,$z), FoF($z,$y)
```

Already exists in SQL but ugly

```
1       WITH RECURSIVE FoF(From, To) AS
2       (
3       SELECT From, To, FROM Friend
4       UNION
5       SELECT Friend.From, FoF.End
6       FROM Friend, FoF
7       WHERE Friend.To = FoF.From);
```

Some datalog extensions we use

update extensional relations in the head of rules:

```
Friend($y,$x) :- Friend($x,$y)
```

negation in the body of rules

```
Picture($x,Charlie) :- Picture($x,Alice), ¬Picture($x,Bob)
```

distributed relations are distributed over the network

```
Picture@Bob($x,Bob) :- Picture@Alice($x,Bob)
```

Webdamlog

Schema

(π, E, I, σ) where

- π is a possibly *infinite* set of peer names
- E is a set of extensional relations of the form $\texttt{m@p}$ for $p \in \pi$
- I is a set of intensional relations of the form $\texttt{m@p}$ for $p \in \pi$
- $\sigma(\texttt{m@p})$ typing function, arity and sorts of $\texttt{m@p}$ fields

Webdamlog

Facts

$\texttt{m@p}(a_1, ..., a_n)$, where

- $\texttt{m}$ is a relation name
- $\texttt{p}$ is a peer name in π
- $n = \sigma(\texttt{m@p})$ and $a_1, \ldots, a_n$ are data values
  data values includes the relations and peer names

Extensional

```
Friend@Alice(Alice,Bob)
Friend@Alice(Alice,Charlie)
Friend@Alice(Bob,Charlie)
Picture@Alice(pic1,Alice)
...
```

Intensional

```
Where@Alice(Alice,Grenoble)
Where@Alice(Bob,Nantes)
Where@Alice(Charlie,Rennes)
```

Webdamlog

Rules

$\$R_{n+1}@\$P_{n+1}(\$\overline{U}_{n+1})$:- $(\neg)\$R_1@\$P_1(\$\overline{U}_1), \ldots, (\neg)\$R_n@\$P_n(\$\overline{U}_n)$

- $\$R_i$ are relation terms possibly variables
- $\$P_i$ are peer terms possibly variables
- $\$\overline{U}_i$ are tuples of terms
- read body from left to right

Safety condition

- $\$R_{n+1}$ $\$P_{n+1}$ must appear positively bound in the body
- $\$P_i$ must be previously bound
- each variables must appear in positive atom before being used

Rules reside at peers:

```
[at Alice]
Picture@Alice($x,$y) :- Friend@Alice(Alice, $friend ),
                                    Picture@ $friend ($x,$y)
```

Webdamlog

Rules

$\$R_{n+1}@\$P_{n+1}(\$\overline{U}_{n+1})$:- $(\neg)\$R_1@\$P_1(\$\overline{U}_1), \ldots, (\neg)\$R_n@\$P_n(\$\overline{U}_n)$

- $\$R_i$ are relation terms possibly variables
- $\$P_i$ are peer terms possibly variables
- $\$\overline{U}_i$ are tuples of terms
- read body from left to right

Safety condition

- $\$R_{n+1}$ $\$P_{n+1}$ must appear positively bound in the body
- $\$P_i$ must be previously bound
- each variables must appear in positive atom before being used

Rules reside at peers:

```
[at Alice]
Picture@Alice($x,$y) :- Friend@Alice(Alice, $friend ),
                        Picture@ $friend  ($x,$y)
```

Particularity of Webdamlog rules

[at p]
$r_0@p_0(\overline{x_0})$:- $r_1@p_1(\overline{x_1})$, $r_2@p_2(\overline{x_2})$, ...,$r_n@p_n(\overline{x_n})$

Semantic depending on 3 criteria

- $p_1, \ldots, p_n = p$, the rule is called *local*
- head $r_0@p_0(\overline{x_0})$ is intensional or extensional
- head $r_0@p_0(\overline{x_0})$ is local or not (p_0 = p or not)

Local rules with local intensional head

Extensional: Friend@Alice: (Alice,Bob),(Alice,Charlie),(Bob,Charlie)
Intensional: FoF@Alice

[at Alice]

```
FoF@Alice($x,$y):-Friend@Alice($x,$y)
FoF@Alice($x,$y):- Friend@Alice($x,$z), FoF@Alice($z,$y)
```

FoF will contain the transitive closure of Friend:
(Alice,Bob), (Alice,Charlie), (Bob,Charlie), (Alice,Charlie)

Intuition

This is standard datalog evaluation

Local rules with local extensional head

Extensional: Friend@Alice

Step 0: (Alice,Bob),(Alice,Charlie),(Bob,Charlie)

[at Alice]

```
Friend@Alice($y,$x):-Friend@Alice($x,$y)
```

- Step 1: (Bob,Alice),(Charlie,Alice),(Charlie,Bob)
- Step 2: (Alice,Bob),(Alice,Charlie),(Bob,Charlie)

Intuition

Database updates

Remarks:

- by default extensional facts are consumed
- unless we declare the relation as *persistent*

Local rules with non-local extensional head

Extensional: `Today@Alice(december 5)`
Extensional: `Event@Alice(birthday,december 5,SMS,Bob-phone)`

[at Alice]

```
$r@$p(Happy birthday):-Today@Alice($date),
                       Event@Alice(birthday,$date,$r,$p)
```

Produce `SMS@Bob-phone(Happy birthday)`

Intuition

Messaging between peers

Local rules with non-local intensional head

Extensional: Friend@Alice
Intensional: Contact@Bob

[at Alice]

```
Contact@Bob($x,$y):- Friend@Alice($x,$y)
```

Bob gets a view on Alice's friends

Intuition

External view definition

Non-local rules: *delegation*

The main novelty of Webdamlog

Extensional: `Friend@Alice`: (Alice,Bob),(Alice,Charlie),(Bob,Charlie)

[at Alice]

```
Picture@Alice($pic,Alice):-Friend@Alice(Alice,$f),
                           Picture@$f($pic,Alice)
```

This will install two rules:

[at Bob] `Picture@Alice($pic,Alice):-Picture@Bob($pic,Alice)`
[at Charlie] `Picture@Alice($pic,Alice):-Picture@Charlie($pic,Alice)`

Remark

If `Friend@Alice(Alice,Bob)` no longer holds the delegation is uninstalled

Non-local rules: *delegation 2*

The main novelty of Webdamlog

Intuition

- Possible to ask another peer to perform some tasks for you (distributed computation)
- Possible to exchange knowledge between peers

Webdamlog semantic

State

$(I, \Gamma, \widetilde{\Gamma})$

- $I(p)$ the local state of p is a finite set of extensional facts
- $\Gamma(p)$ is the finite set of rules at p
- $\widetilde{\Gamma}(p, q)$ $(p \neq q)$ is the set of rules that p delegated to q

State transition

Choose some peer p randomly – asynchronously
Compute the transition: $(I_0, \Gamma_0, \widetilde{\Gamma}_0) \rightarrow (I_1, \Gamma_1, \widetilde{\Gamma}_1) \rightarrow (I_2, \Gamma_2, \widetilde{\Gamma}_2) \rightarrow \ldots$

- the database updates at p
- the messages sent to the other peers
- the delegations of rules to other peers

Fair sequence: each peer is selected infinitely often

Webdamlog summary

Webdamlog is datalog with novel extensions

- variables in relation and peer names
- delegation

both imply installing rules at run-time

Results:

- formal definition of Webdamlog
- expressivity results:
  - the model with delegation is more general, unless all peers and programs are known in advance
- convergence is very hard to achieve because of asynchronicity

1 Context and problematic

2 Webdamlog — a language for distributed knowledge

3 System — implementation of a Webdamlog engine
- Webdamlog evaluation
- Deletions in Webdamlog

4 Proof of concept — application and feasibility of Webdamlog

5 Conclusion

Implementation on top of *Bud*

Bud [UC Berkeley] is a distributed datalog engine with updates; it supports

- local rules with local extensional head (semi-naive)
- local rules with non-local extensional head (semi-naive)

Bud does not support

- negation
- intensional relation optimizations (query sub-query, magic sets)

Bud neither any other engine implements requirements for *Webdamlog*

- variables in relation and peer names
- delegation
- installing rules at run-time

Semi-naive

Naive evaluation leads to redundant computation

```
Edge:  (1,2),(2,3),(3,4),(4,5)
Path($x,$y):-Edge($x,$y)
Path($x,$y):-Edge($x,$z),Path($z,$y)
```

$$
\begin{aligned}
Path(I)^0 &= \emptyset \\
Path(I)^1 &= Path(I)^0 \cup \{Path(1,2), Path(2,3), Path(3,4), Path(4,5)\} \\
Path(I)^2 &= Path(I)^1 \cup \{Path(1,3), Path(2,4), Path(3,5)\} \\
Path(I)^3 &= Path(I)^2 \cup \{Path(1,4), Path(2,5)\} \\
Path(I)^4 &= Path(I)^3 \cup \{Path(1,5)\} \\
Path(I)^5 &= Path(I)^4
\end{aligned}
$$

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$

Δ new facts since previous step

Semi-naive use delta of previous step to compute the current step

$\texttt{Path}^{\texttt{i}}\texttt{(\$x,\$y):-Edge(\$x,\$z),}\Delta Path^{i-1}\texttt{(\$z,\$y)}$

Webdamlog engine run

Run a Webdamlog stage $(I, \Gamma, \widetilde{\Gamma}) \rightarrow (I', \Gamma', \widetilde{\Gamma}')$ in three steps

1. inputs are collected and a new state is defined
   - insert/delete *Webdamlog* facts and rules
   - update deltas of relations
   - compile the *Webdamlog* program into a *Bud* program
2. semi-naive evaluation: monotonic *Bud* program with a fixed set of local rules run to fixpoint
3. output messages and delegations are collected and sent to other peers for next stage

Management of variables

[at p]
Fact: r_1@p(relname,peername)
Rule: $\underbrace{r_0@p(\overline{x_o})}_{head}$:- $\underbrace{r_1@p(\$x,\$y),}_{static}$ $\underbrace{\$x@p(\overline{x_1}),r_2@\$y(\overline{x_2}),\ldots}_{dynamic}$

Stage i:

step 1 install at p: i@p($x,$y) :- $\underbrace{r_1@p(\$x,\$y)}_{static}$

step 2 semi-naive evaluation evaluate i@p

step3 FOR EACH fact $F(v_x,v_y)$ IN i@p
send new rule: *head* :- *dynamic*
with variables bounded by $F(v_x,v_y)$
$\underbrace{r_0@p(\overline{x_0})}_{head}$:- $\underbrace{v_x@p(\overline{x_1}),r_2@v_y(\overline{x_2}),\ldots}_{dynamic}$

Stage i + 1, at step 1:
receive $r_0@p(\overline{x_o})$:- relname@p($\overline{x_1}$),r_2@peername($\overline{x_2}$),...

Management of variables

[at p]
Fact: r_1@p(relname,peername)
Rule: $\underbrace{r_0@p(\overline{x_o})}_{head}$:- $\underbrace{r_1@p(\$x,\$y),}_{static}$ $\underbrace{\$x@p(\overline{x_1}),r_2@\$y(\overline{x_2}),\ldots}_{dynamic}$

Stage i:

step 1 install at p: i@p($x,$y) :- $\underbrace{r_1@p(\$x,\$y)}_{static}$

step 2 semi-naive evaluation evaluate i@p

step3 FOR EACH fact $F(v_x,v_y)$ IN i@p
send new rule: *head* :- *dynamic*
with variables bounded by $F(v_x,v_y)$
$\underbrace{r_0@p(\overline{x_0})}_{head}$:- $\underbrace{v_x@p(\overline{x_1}),r_2@v_y(\overline{x_2}),\ldots}_{dynamic}$

Stage i + 1, at step 1:
receive $r_0@p(\overline{x_o})$:- relname@p($\overline{x_1}$),r_2@peername($\overline{x_2}$),...

Management of variables

[at p]
Fact: r_1@p(relname,peername)
Rule: $\underbrace{r_0@p(\overline{x_o})}_{head}$:- $\underbrace{r_1@p(\$x,\$y),}_{static}$ $\underbrace{\$x@p(\overline{x_1}),r_2@\$y(\overline{x_2}),\ldots}_{dynamic}$

Stage i:

step 1 install at p: i@p($x,$y) :- $\underbrace{r_1@p(\$x,\$y)}_{static}$

step 2 semi-naive evaluation evaluate i@p

step3 FOR EACH fact $F(v_x,v_y)$ IN i@p
send new rule: *head* :- *dynamic*
with variables bounded by $F(v_x,v_y)$
$\underbrace{r_0@p(\overline{x_0})}_{head}$:- $\underbrace{v_x@p(\overline{x_1}),r_2@v_y(\overline{x_2}),\ldots}_{dynamic}$

Stage i + 1, at step 1:
receive $r_0@p(\overline{x_o})$:- relname@p($\overline{x_1}$),r_2@peername($\overline{x_2}$),...

Management of variables

[at p]
Fact: r_1@p(relname,peername)
Rule: $\underbrace{r_0@p(\overline{x_o})}_{head}$:- $\underbrace{r_1@p(\$x,\$y),}_{static}$ $\underbrace{\$x@p(\overline{x_1}),r_2@\$y(\overline{x_2}),\ldots}_{dynamic}$

Stage i:

step 1 install at p: i@p($x,$y) :- $\underbrace{r_1@p(\$x,\$y)}_{static}$

step 2 semi-naive evaluation evaluate i@p

step3 FOR EACH fact $F(v_x,v_y)$ IN i@p
send new rule: *head* :- *dynamic*
with variables bounded by $F(v_x,v_y)$
$\underbrace{r_0@p(\overline{x_0})}_{head}$:- $\underbrace{v_x@p(\overline{x_1}),r_2@v_y(\overline{x_2}),\ldots}_{dynamic}$

Stage i + 1, at step 1:
receive $r_0@p(\overline{x_o})$:- relname@p($\overline{x_1}$),r_2@peername($\overline{x_2}$),...

Management of variables

[at p]
Fact: r_1@p(relname,peername)
Rule: $\underbrace{r_0@p(\overline{x_o})}_{head}$:- $\underbrace{r_1@p(\$x,\$y),}_{static}$ $\underbrace{\$x@p(\overline{x_1}),r_2@\$y(\overline{x_2}),\ldots}_{dynamic}$

Stage i:

step 1 install at p: i@p($x,$y) :- $\underbrace{r_1@p(\$x,\$y)}_{static}$

step 2 semi-naive evaluation evaluate i@p

step3 FOR EACH fact $F(v_x,v_y)$ IN i@p
send new rule: *head* :- *dynamic*
with variables bounded by $F(v_x,v_y)$
$\underbrace{r_0@p(\overline{x_0})}_{head}$:- $\underbrace{v_x@p(\overline{x_1}),r_2@v_y(\overline{x_2}),\ldots}_{dynamic}$

Stage i + 1, at step 1:
receive $r_0@p(\overline{x_o})$:- relname@p($\overline{x_1}$),r_2@peername($\overline{x_2}$),...

Management of variables

[at p]
Fact: r_1@p(relname,peername)
Rule: $\underbrace{r_0@p(\overline{x_o})}_{head}$:- $\underbrace{r_1@p(\$x,\$y),}_{static}$ $\underbrace{\$x@p(\overline{x_1}),r_2@\$y(\overline{x_2}),\ldots}_{dynamic}$

Stage i:

step 1 install at p: i@p($x,$y) :- $\underbrace{r_1@p(\$x,\$y)}_{static}$

step 2 semi-naive evaluation evaluate i@p

step3 FOR EACH fact $F(v_x,v_y)$ IN i@p
send new rule: *head* :- *dynamic*
with variables bounded by $F(v_x,v_y)$
$\underbrace{r_0@p(\overline{x_0})}_{head}$:- $\underbrace{v_x@p(\overline{x_1}),r_2@v_y(\overline{x_2}),\ldots}_{dynamic}$

Stage i + 1, at step 1:
receive r_0@p($\overline{x_o}$) :- relname@p($\overline{x_1}$),r_2@peername($\overline{x_2}$),...

Evaluation of deletion in datalog

```
Edge@Alice:  (1,2),(2,3),(3,4),(4,5)
Rule1:  Path@Alice($x,$y):-Edge@Alice($x,$y)
Rule2:  Path@Alice($x,$y):-Edge@Alice($x,$z),Path@Alice($z,$y)

Path@Bob:  (1,3),(1,5)
Rule3:  Path@Alice($x,$y):-Path@Bob($x,$y)
```

Path@Alice:

$$\underbrace{(1,2),(2,3),(3,4),(4,5),\overbrace{(1,3)}^{Rule3},(2,4),(3,5),(1,4),(2,5),\overbrace{(1,5)}^{\text{Rule 3}}}_{\text{Rule 1 and 2}}$$

Delete Path@Alice(2,3)

- semi-naive requires full recomputation
- we introduce a novel optimization based on provenance

Example for provenance in Webdamlog evaluation

Alice creates a gallery of pictures where she is tagged from her own pictures and the pictures of her friends

```
Picture@Alice(pic1,Alice)
Friend@Alice(Alice,Bob)
Picture@Bob(pic1,Alice)
```

- [Rule 1 at Alice]

```
Gallery@Alice($pic,Alice):-Picture@Alice($pic,Alice)
```

- [Rule 2 at Alice]

```
Gallery@Alice($pic,Alice):-Friend@Alice(Alice,$f),
                           Picture@$f($pic,Alice)
```

- [Rule 3 at Bob] delegation

```
Gallery@Alice($pic,Alice):-Picture@Bob($pic,Alice)
```

Example for provenance in Webdamlog evaluation

Alice creates a gallery of pictures where she is tagged from her own pictures and the pictures of her friends

```
Picture@Alice(pic1,Alice)
Friend@Alice(Alice,Bob)
Picture@Bob(pic1,Alice)
```

- [Rule 1 at Alice]

```
Gallery@Alice($pic,Alice):-Picture@Alice($pic,Alice)
```

- [Rule 2 at Alice]

```
Gallery@Alice($pic,Alice):-Friend@Alice(Alice,$f),
                           Picture@$f($pic,Alice)
```

- [Rule 3 at Bob] delegation

```
Gallery@Alice($pic,Alice):-Picture@Bob($pic,Alice)
```

Example for provenance in Webdamlog evaluation

Alice creates a gallery of pictures where she is tagged from her own pictures and the pictures of her friends

```
Picture@Alice(pic1,Alice)
Friend@Alice(Alice,Bob)
Picture@Bob(pic1,Alice)
```

- [Rule 1 at Alice]

```
Gallery@Alice($pic,Alice):-Picture@Alice($pic,Alice)
```

- [Rule 2 at Alice]

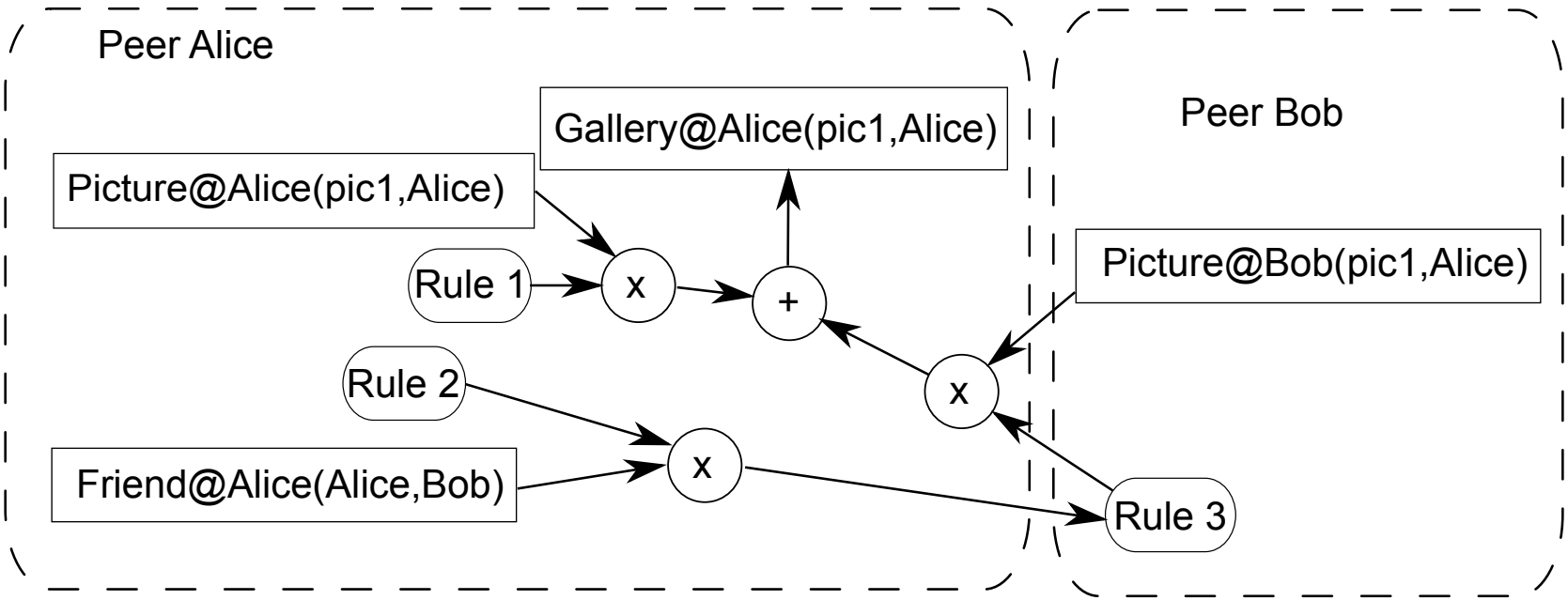
```
Gallery@Alice($pic,Alice):-Friend@Alice(Alice,$f),
                           Picture@$f($pic,Alice)
```

- [Rule 3 at Bob] delegation

```
Gallery@Alice($pic,Alice):-Picture@Bob($pic,Alice)
```

Optimization deletion using provenance

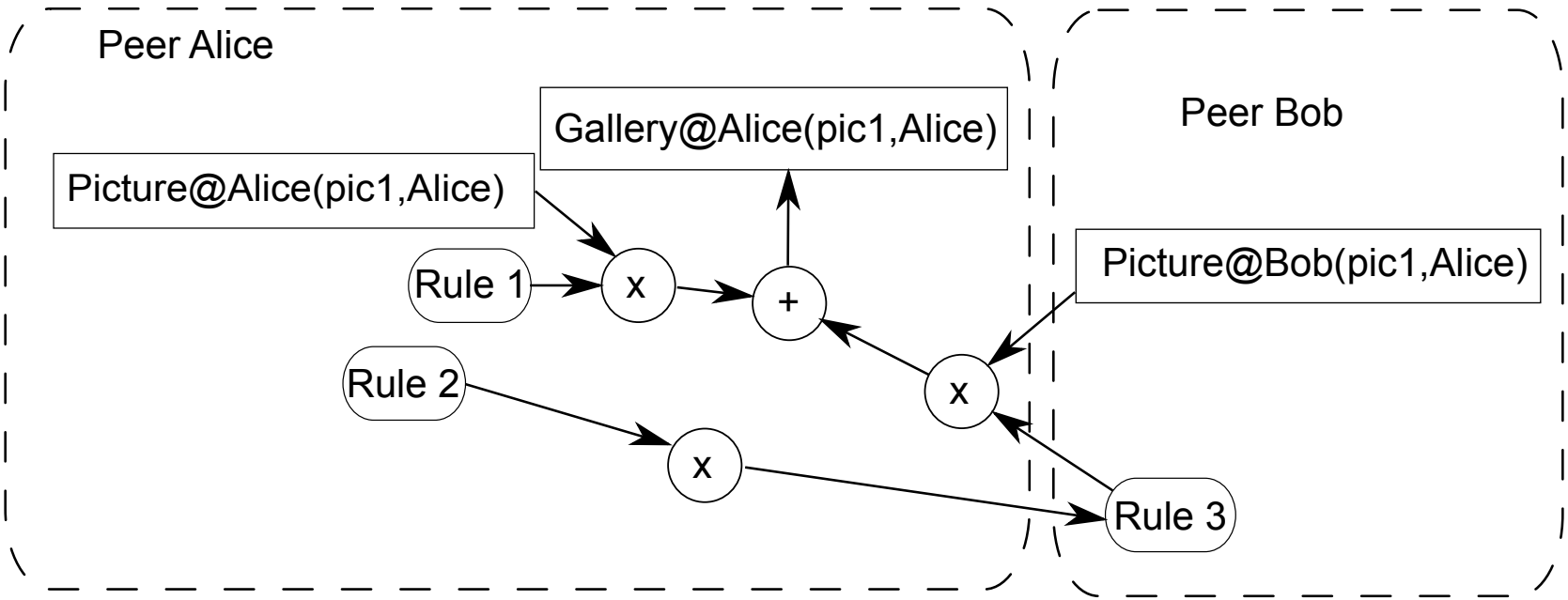
- Time nodes x are conjunction
- Plus nodes + are disjunction



- avoid to rederive the full relations
- keep trace of multiple proofs for the same fact

Optimization deletion using provenance

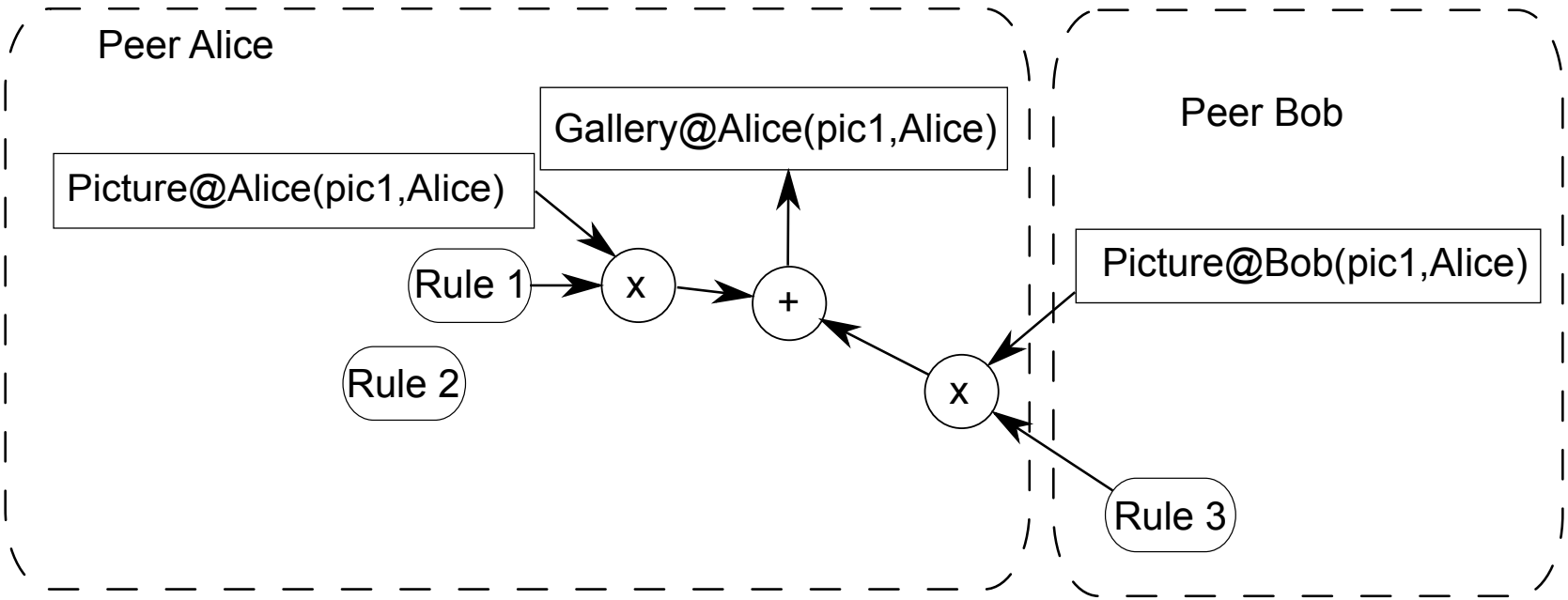
- Time nodes x are conjunction
- Plus nodes + are disjunction



- avoid to rederive the full relations
- keep trace of multiple proofs for the same fact

Optimization deletion using provenance

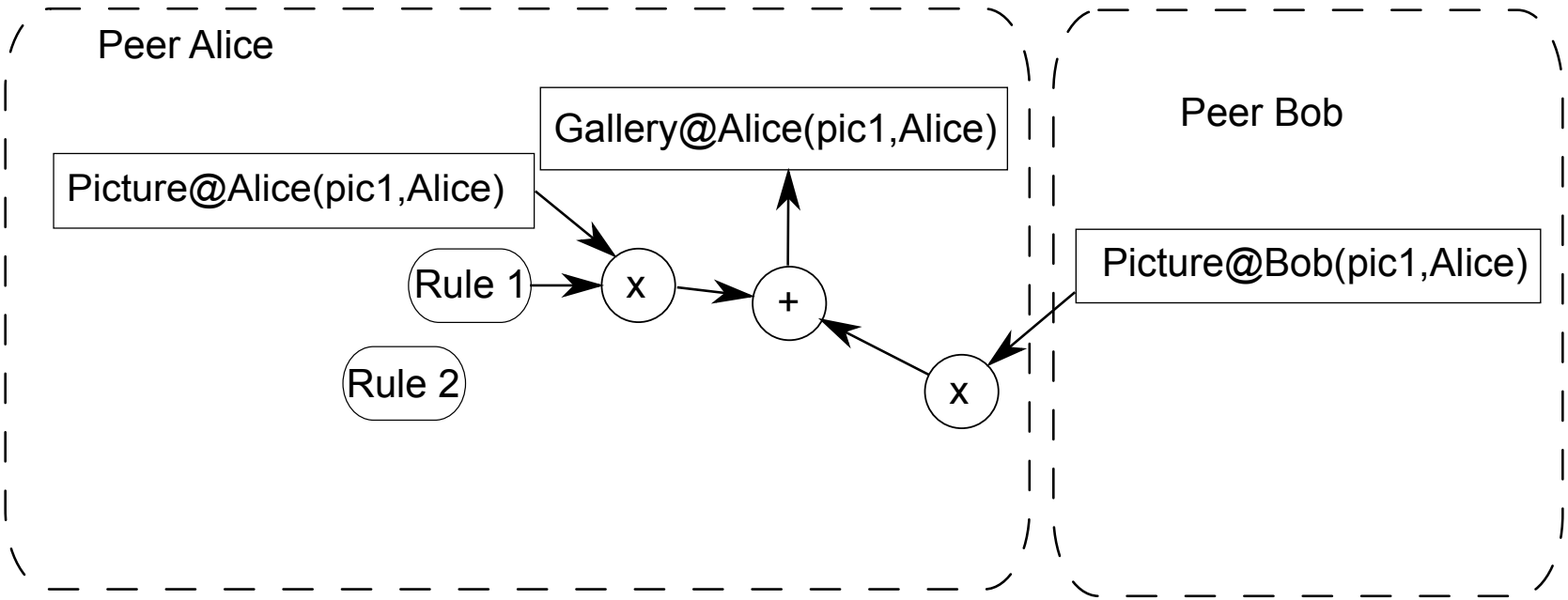
- Time nodes x are conjunction
- Plus nodes + are disjunction



- avoid to rederive the full relations
- keep trace of multiple proofs for the same fact

Optimization deletion using provenance

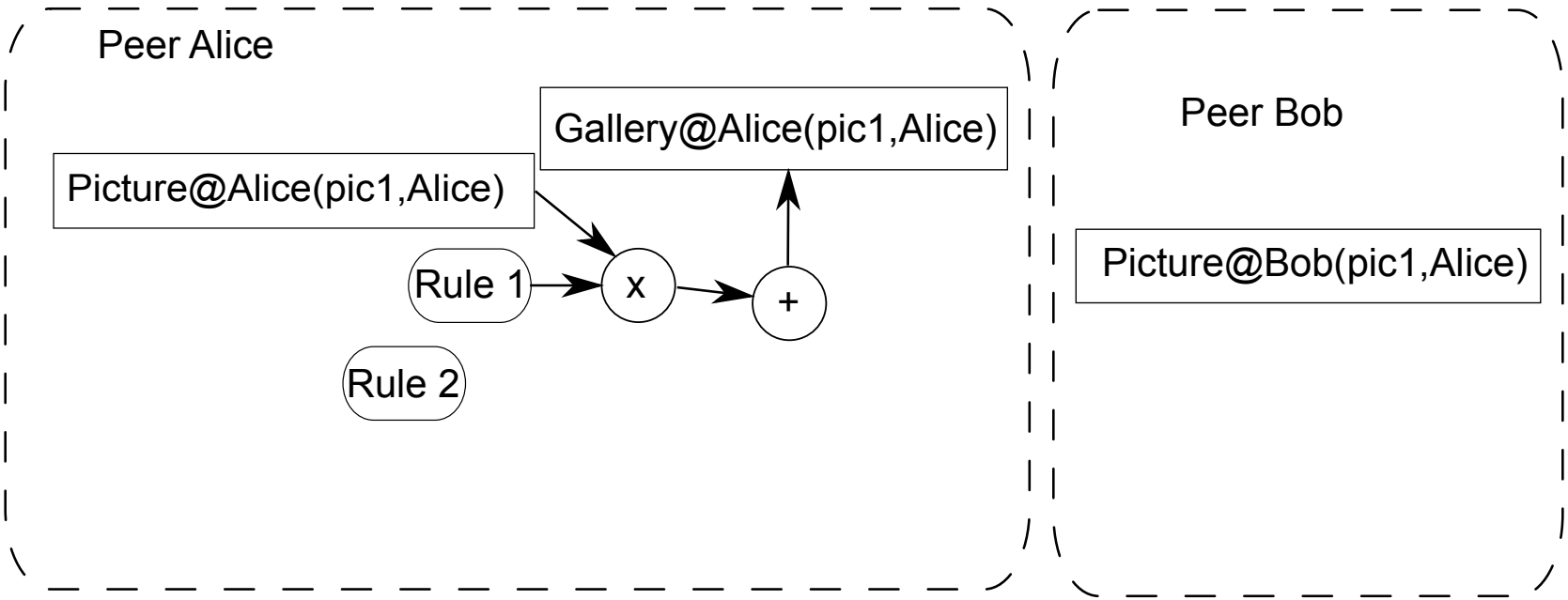
- Time nodes x are conjunction
- Plus nodes + are disjunction



- avoid to rederive the full relations
- keep trace of multiple proofs for the same fact

Optimization deletion using provenance

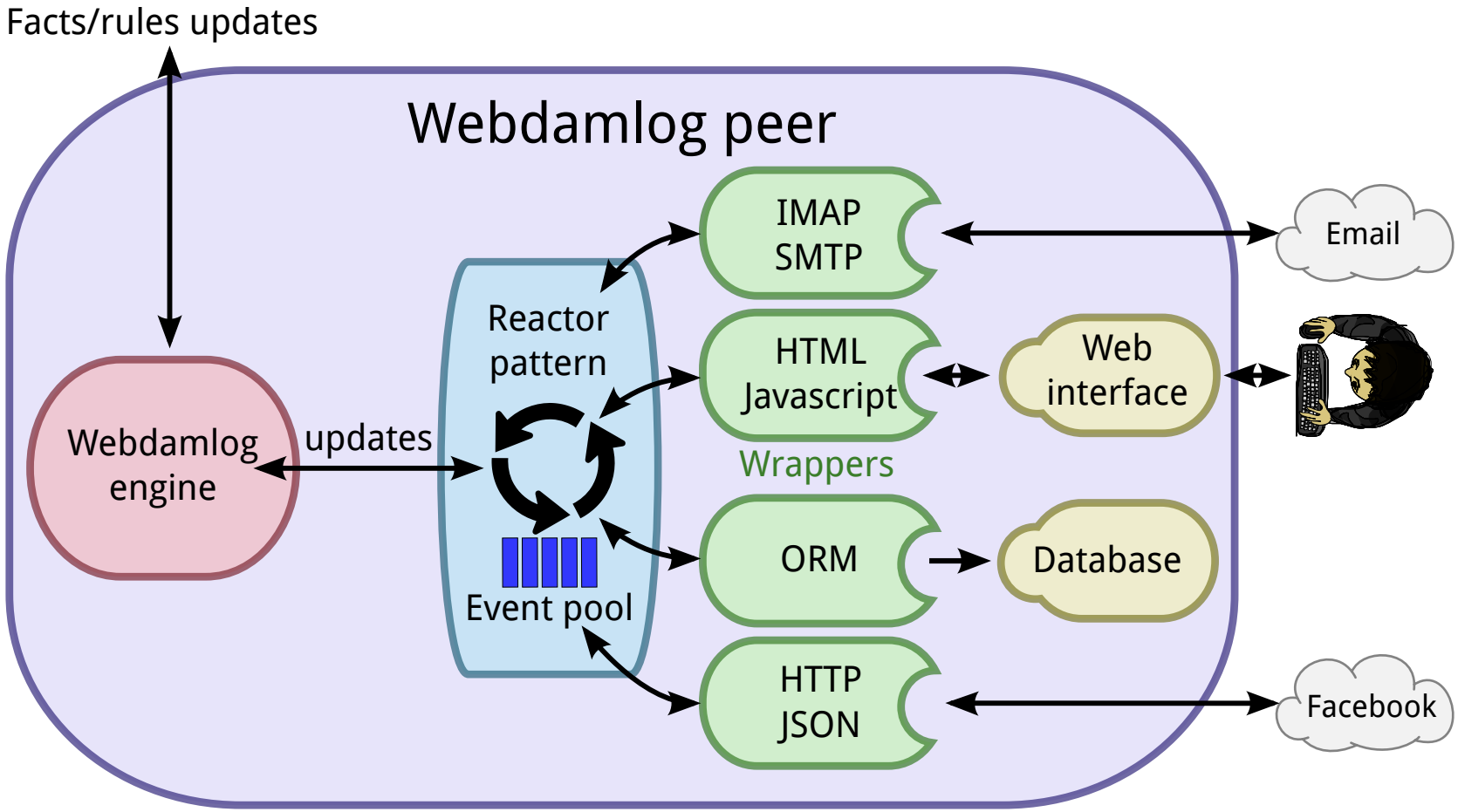
- Time nodes x are conjunction
- Plus nodes + are disjunction



- avoid to rederive the full relations
- keep trace of multiple proofs for the same fact

1. Context and problematic
2. Webdamlog — a language for distributed knowledge
3. System — implementation of a Webdamlog engine
4. **Proof of concept** — application and feasibility of Webdamlog
   - Architecture
   - Experiments
   - User study
5. Conclusion

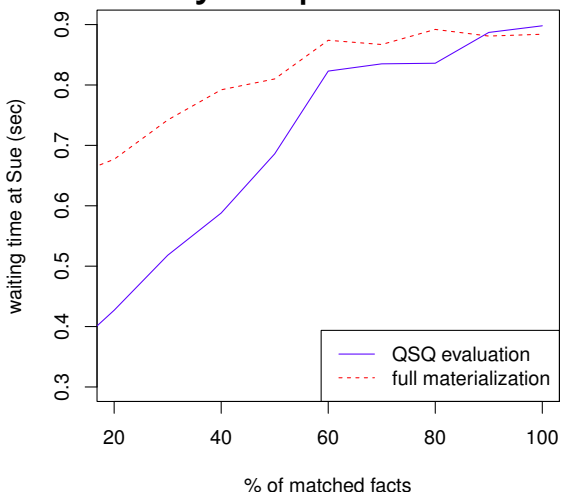
Architecture of a Webdamlog peer



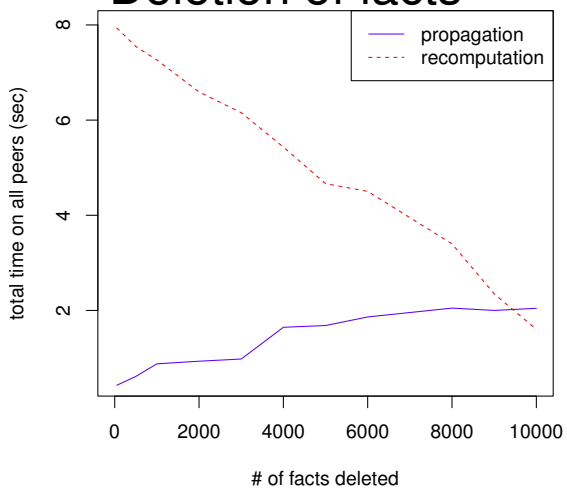
- Wrappers translate external commands into facts/rules updates
- Reactor pattern activates the Webdamlog engine if needed

Experiments

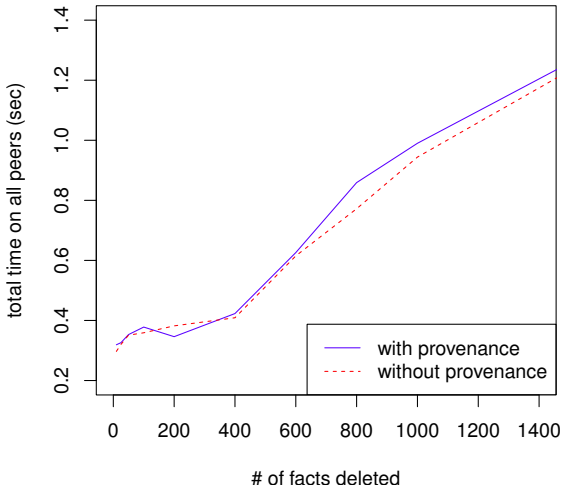
QSQ style optimization



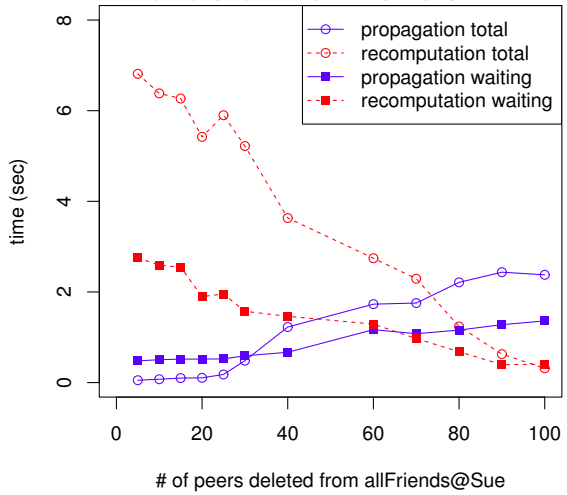
Deletion of facts



Cost of provenance graph maintenance



Deletion of rules



User study

To show that Webdamlog is declarative and user-friendly

Settings:

- pool of 27 participants with and without IT training
- a 20 minute lesson about the language
- an exam to test
  - understanding of Webdamlog programs
  - ability to write Webdamlog programs

Results:

- everyone but 2 participants perfectly understood the language
- a large majority wrote correct rules
- non technical participants took longer to answer

Demonstration

- A demonstration showed at SIGMOD 2013
- A social network to share pictures among the attendees of the conference
- This application “Wepic” run thanks to a Webdamlog engine and wrappers (web interface, database, . . .)
- Scenario
  - the peer “Sigmod” starts with an empty program
  - each attendee runs on his own peer with a basic program
  - each peer discovers other peers via “Sigmod”
  - each peer can customize/add his own rules

Play the video

Conclusion

We propose

- a formal language for data management
- an implementation of an engine for this language (with optimizations)
- a system for application development (with wrappers)

Future work

- on access control (on-going work)
- a better graphical interface
- a more in-depth user study
- an API for the development of applications and wrappers
- the enhancement Webdamlog with ontology technology
- optimization techniques, e.g. dQSQ