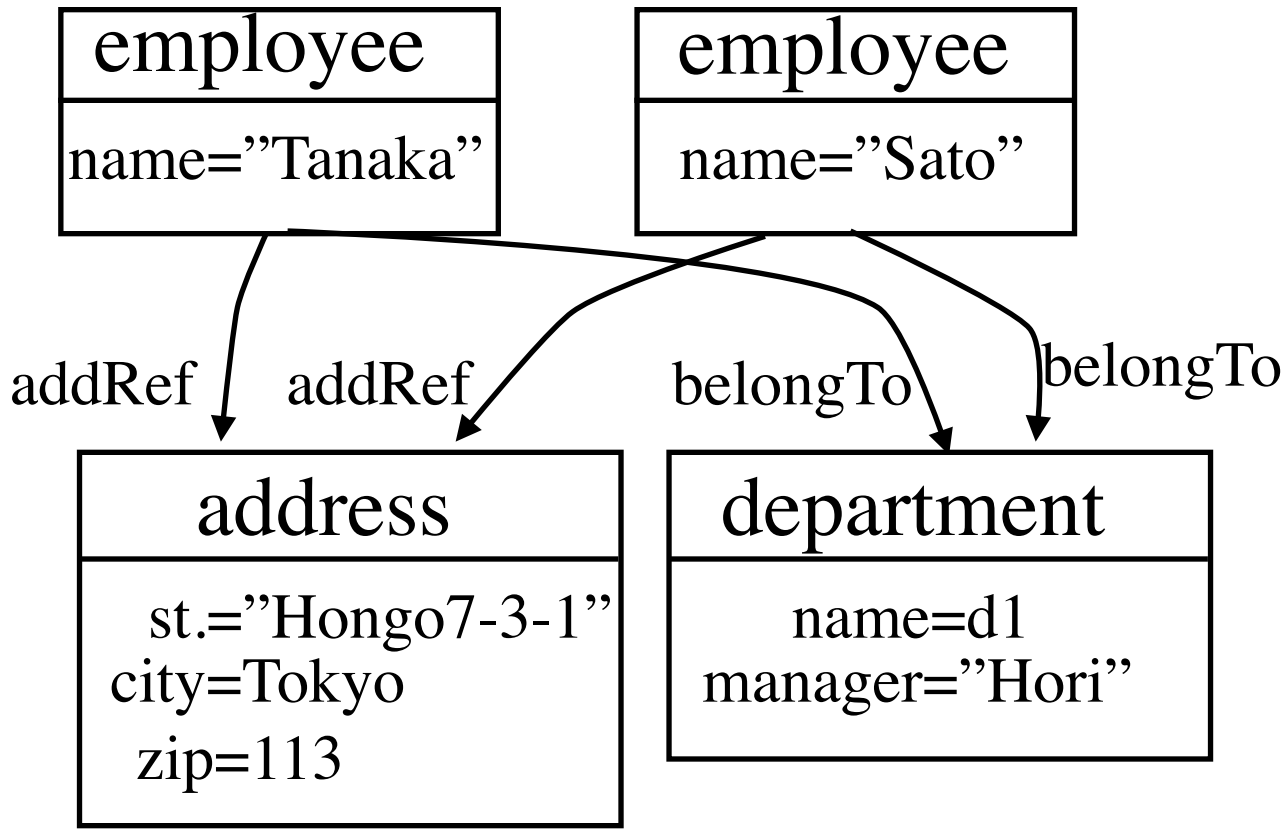
Two Semantics of Updating Graphs

Hiroyuki Kato
National Institute of Informatics, Japan

4th Bi-Trans in ABC

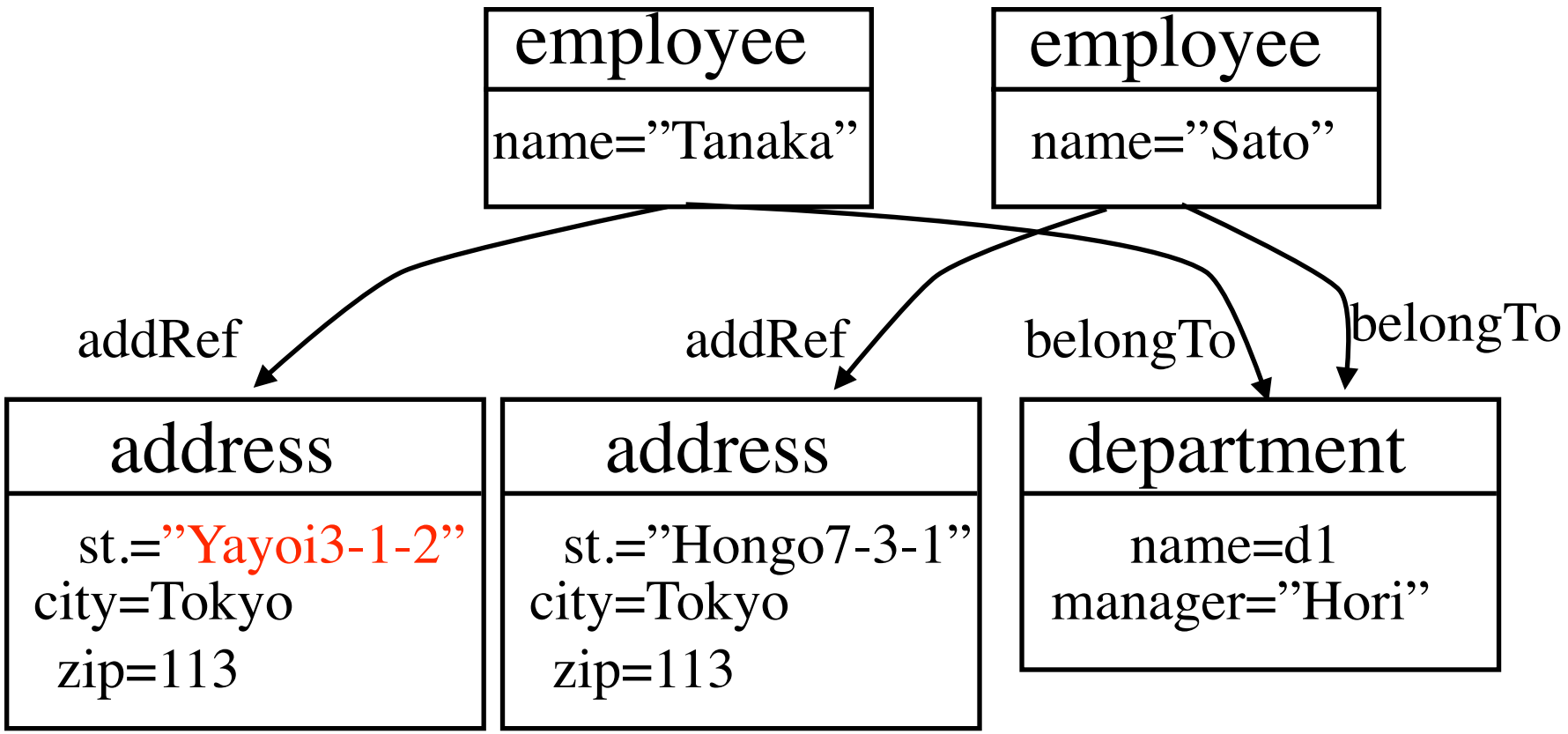
Graphs

- Natural representation of entities in real world



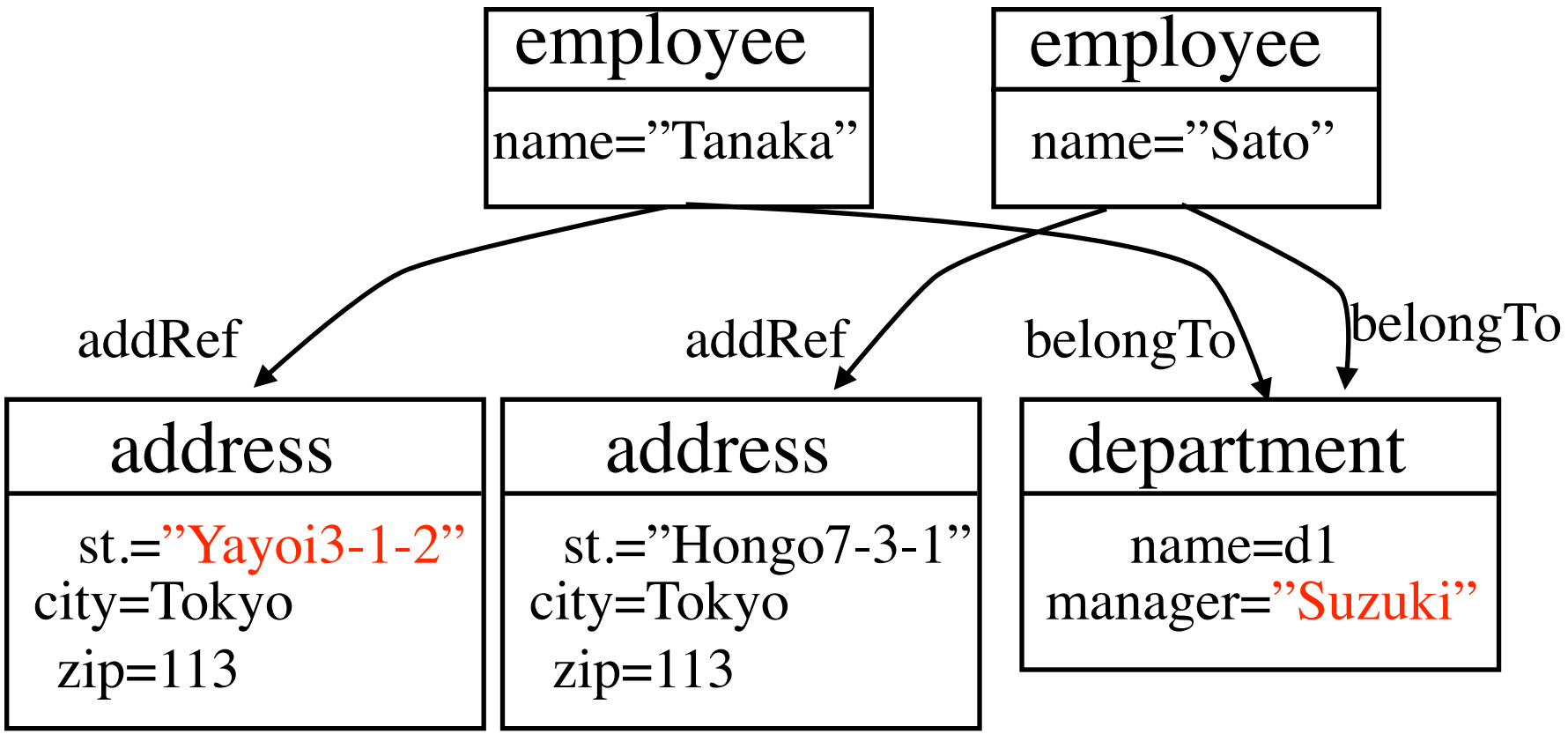
Graphs

- “Tanaka” has moved to new address.



Graphs

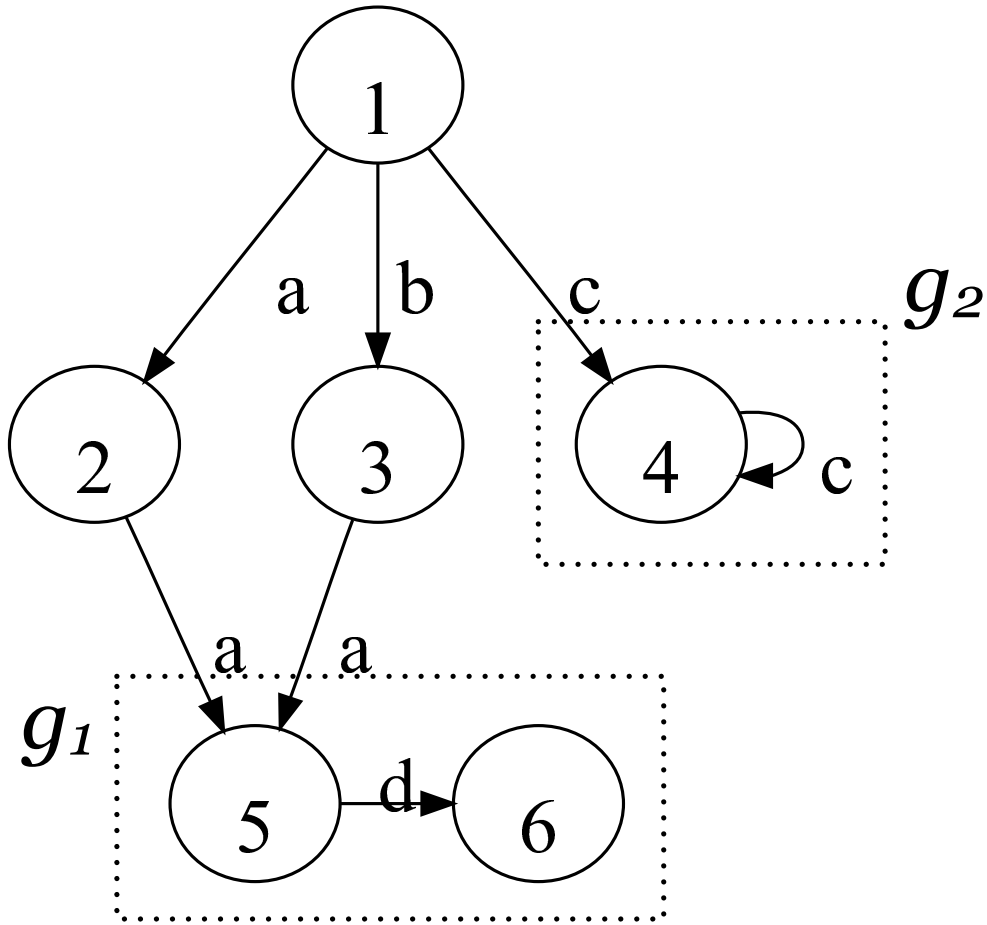
- The manager of d1 has changed.



Outline of this presentation

Towards handling these two kinds of update on shared-nodes in our graph model.

Rooted, Edge-labelled, Set-based, Graphs

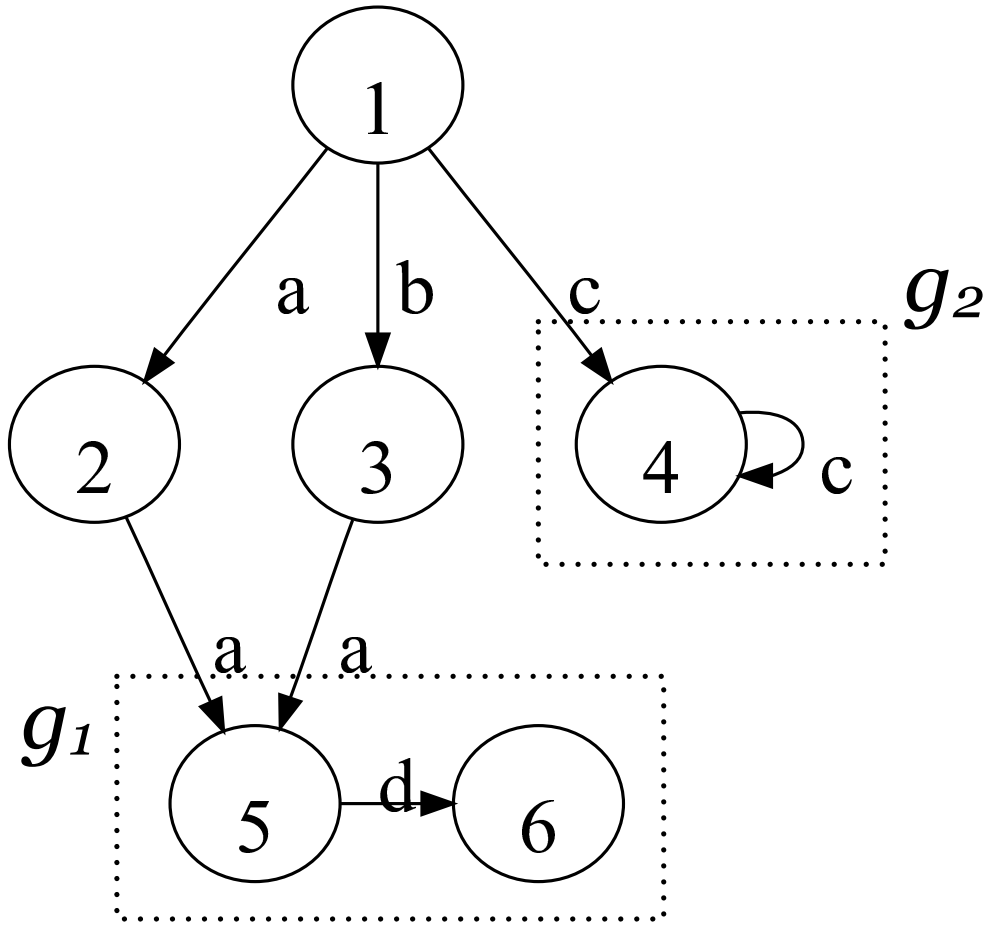


$g = \{a : \{a : g_1\}, b : \{a : g_1\}, c : g_2\}$

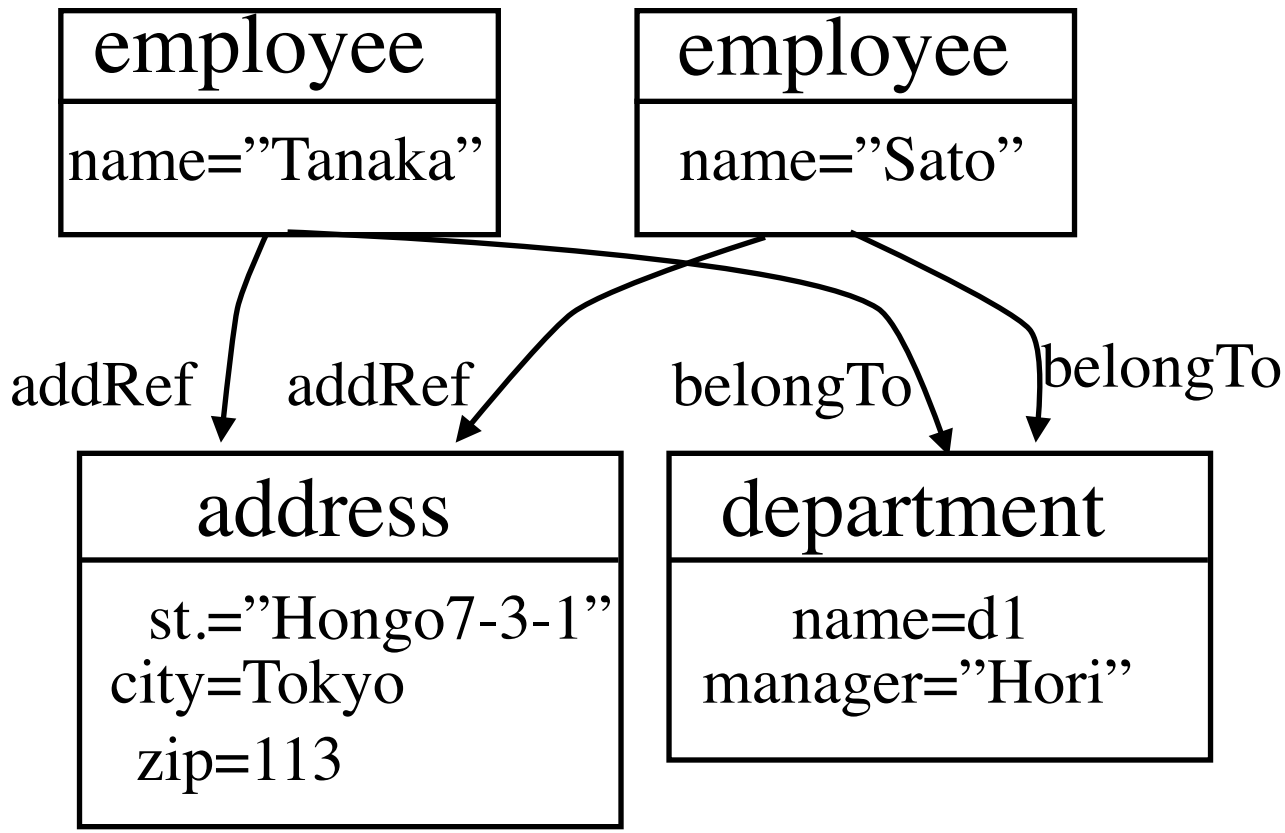
$g_1 = \{d : \{\}\}$

$g_2 = \{c : g_2\}$

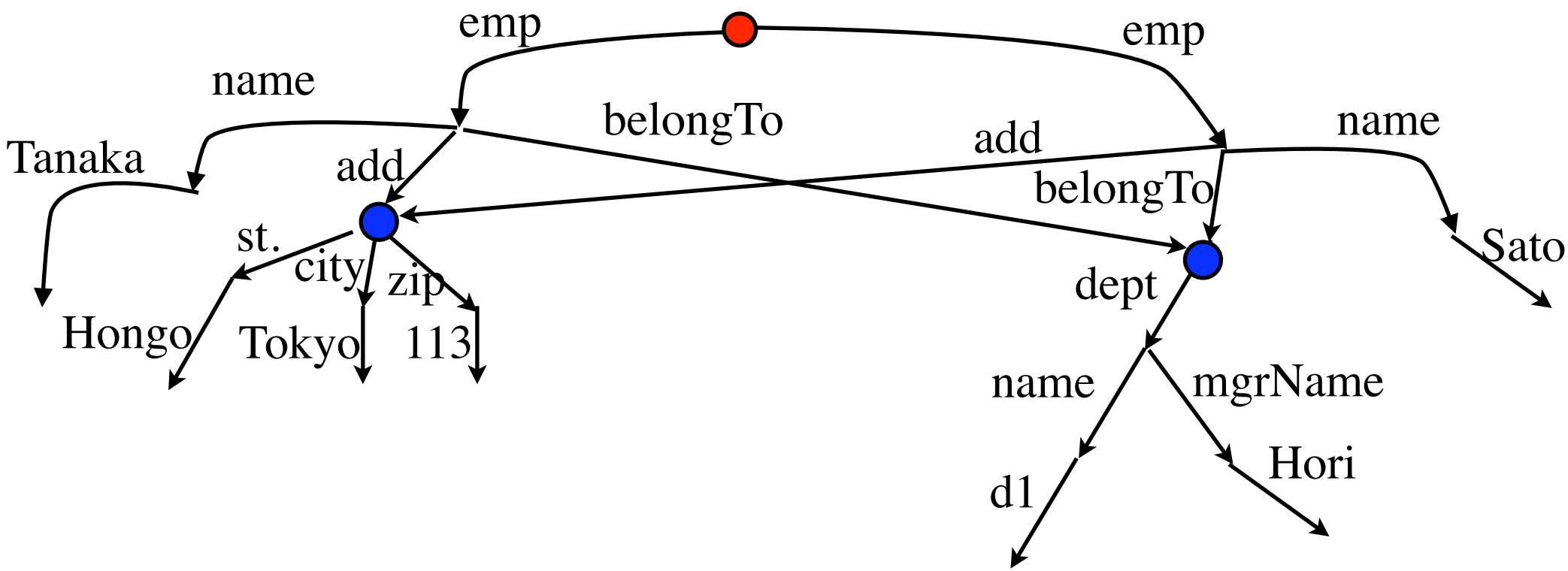
Rooted, Edge-labelled, Set-based, Graphs



$$g = \{a : \{a : g_1\} \cup b : \{a : g_1\} \cup c : g_2\}$$
$$g_1 = \{d : \{\}\}$$
$$g_2 = \{c : g_2\}$$

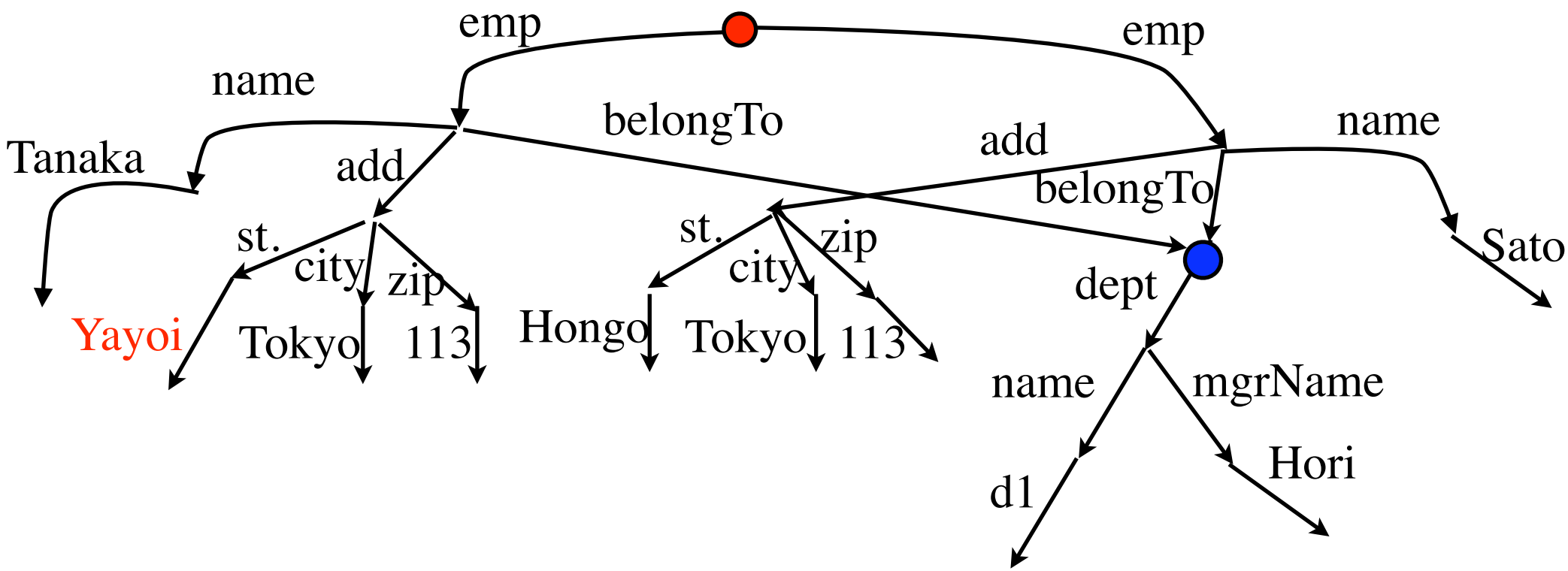

employee
name=”Tanaka”
employee
name=”Sato”
addRef
addRef
belongTo
belongTo
address
st.=”Hongo7-3-1”
city=Tokyo
zip=113
department
name=d1
manager=”Hori”

Our Graph Model



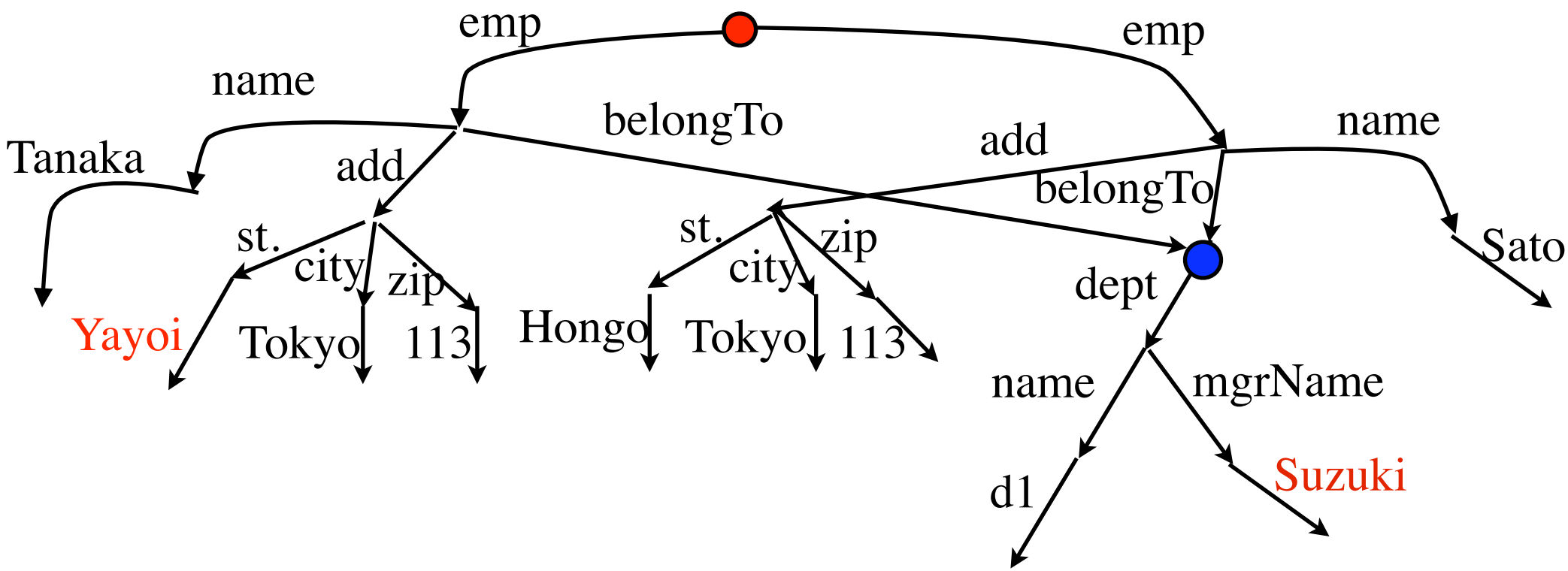
Our Graph Model

- "Tanaka" has moved to new address.



Our Graph Model

- The manager of d1 has changed.



Structural Recursion: Manipulating Graphs

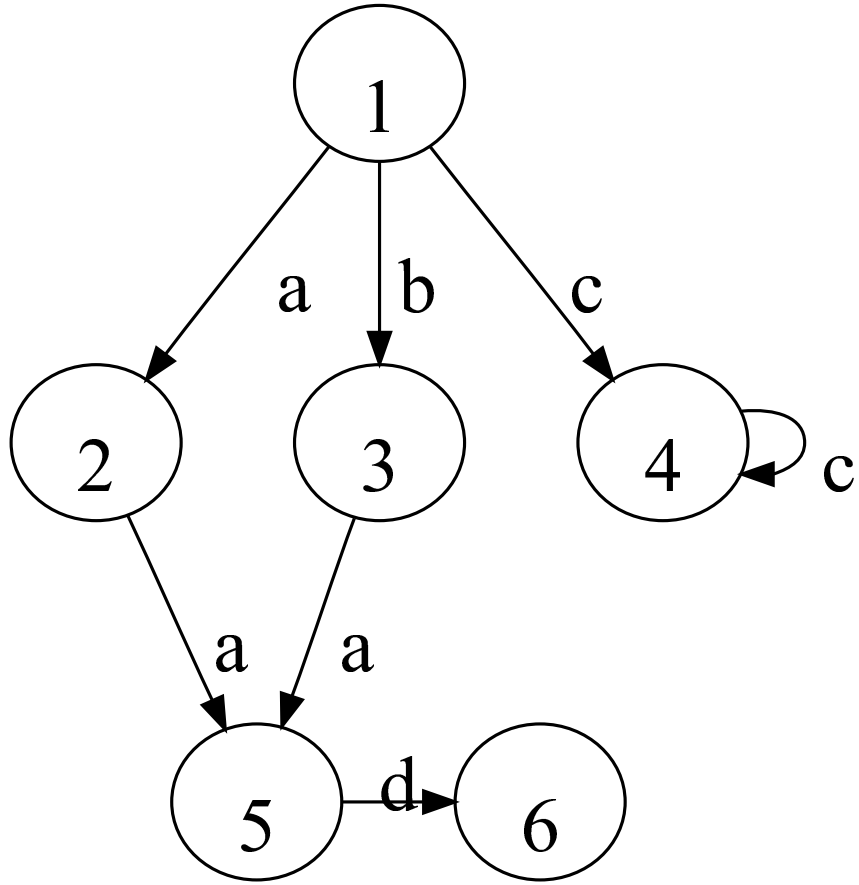
Structural Recursion:

$$
\begin{aligned}
f(\{\}) &= \{\} \\
f(\{\$l : \$g\}) &= e \odot f(\$g) \\
f(g_1 \cup g_2) &= f(g_1) \cup f(g_2)
\end{aligned}
$$

Or written as:

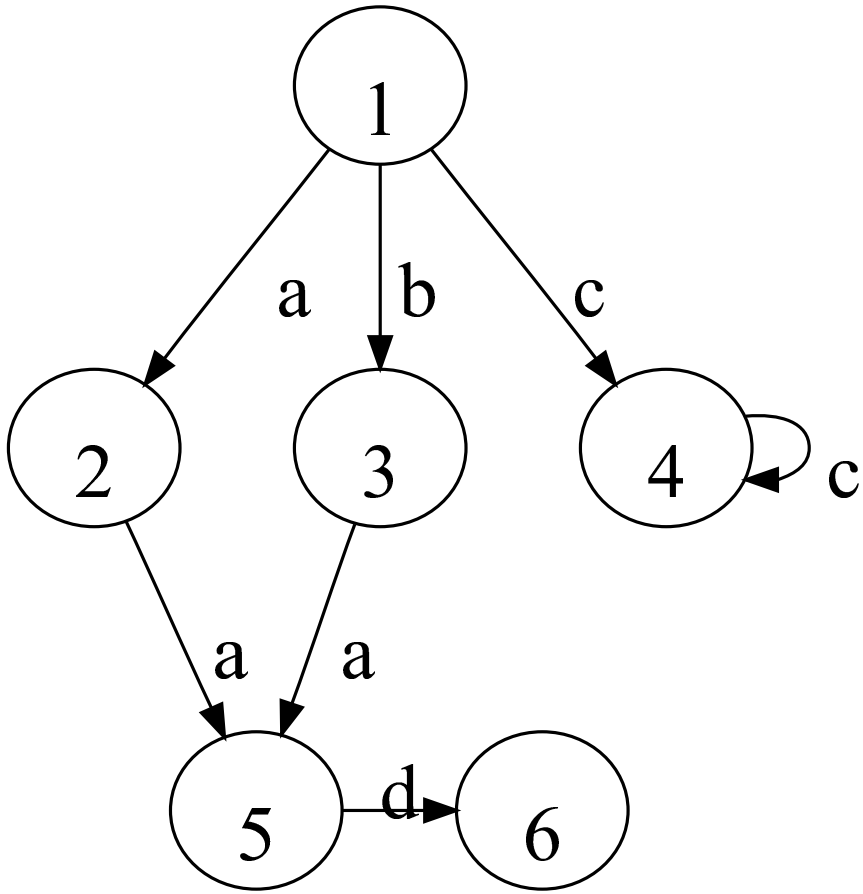
$$\text{sfun } f(\{\$l : \$g\}) = e \odot f(\$g)$$

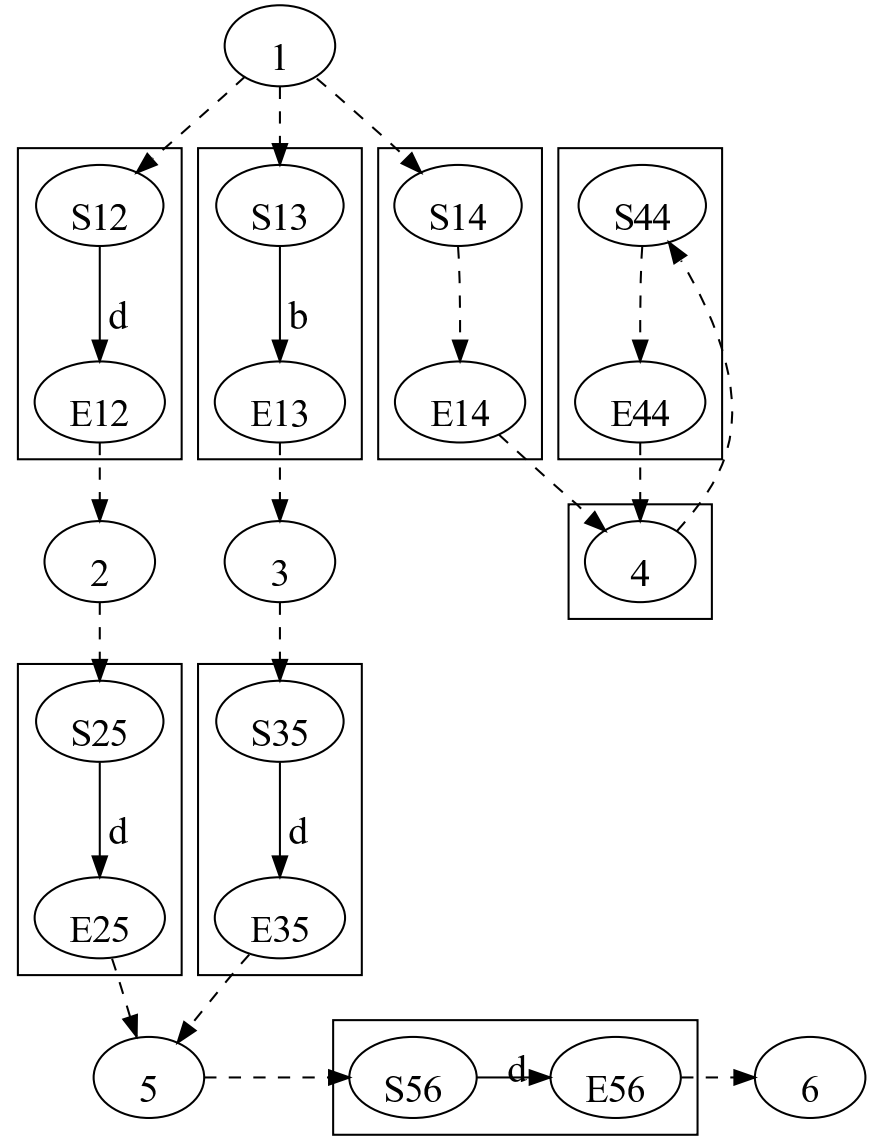
Structural Recursion: An Example



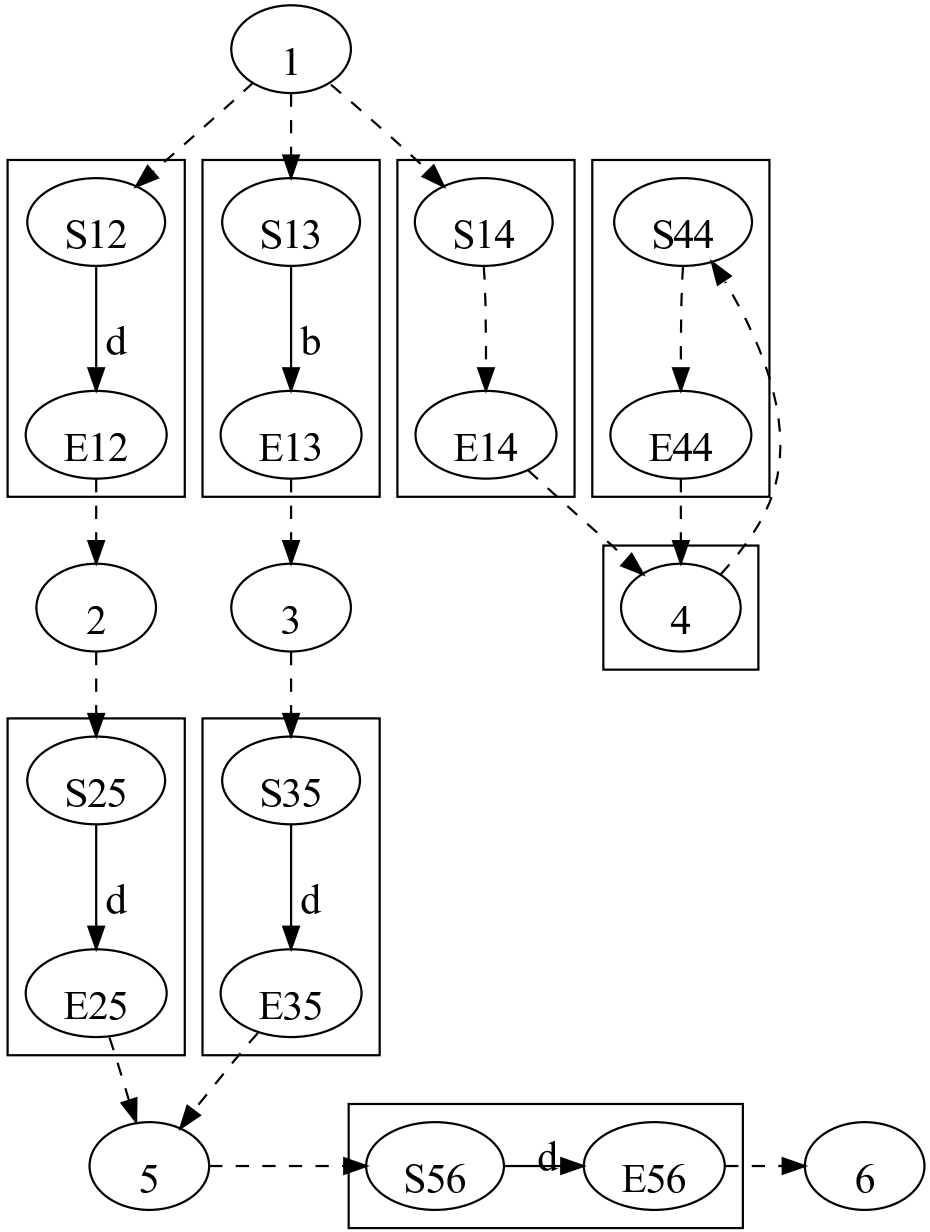
sfun *a2d_xc* ({$*l* : $*g*}) = if $*l* = a then {d : *a2d_xc*($*g*)}
else if $*l* = c then *a2d_xc*($*g*)
else {$*l* : *a2d_xc*($*g*)}

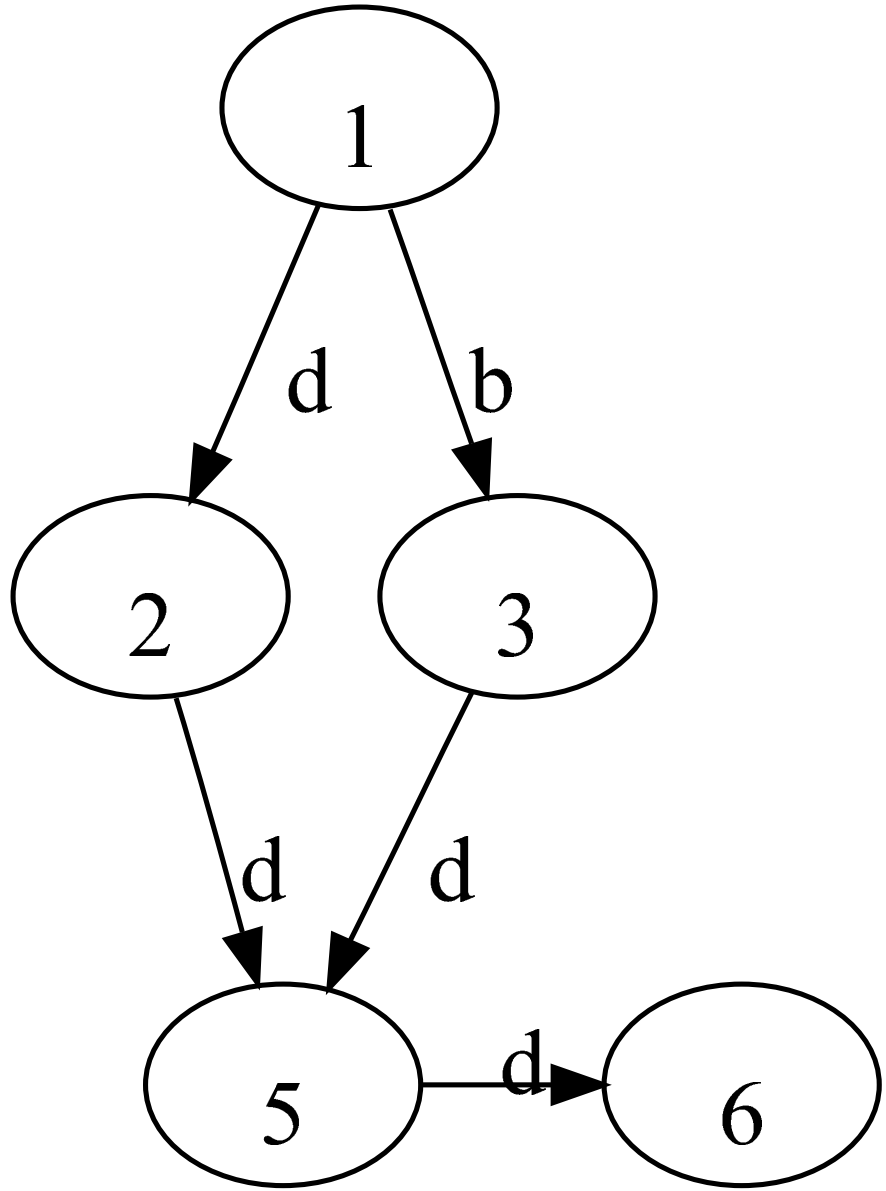
The Bulk Semantics of Structural Recursion





sfun *a2d_xc* ({$*l* : $*g*}) = if $*l* = a then {d : *a2d_xc*($*g*)}
else if $*l* = c then *a2d_xc*($*g*)
else {$*l* : *a2d_xc*($*g*)}

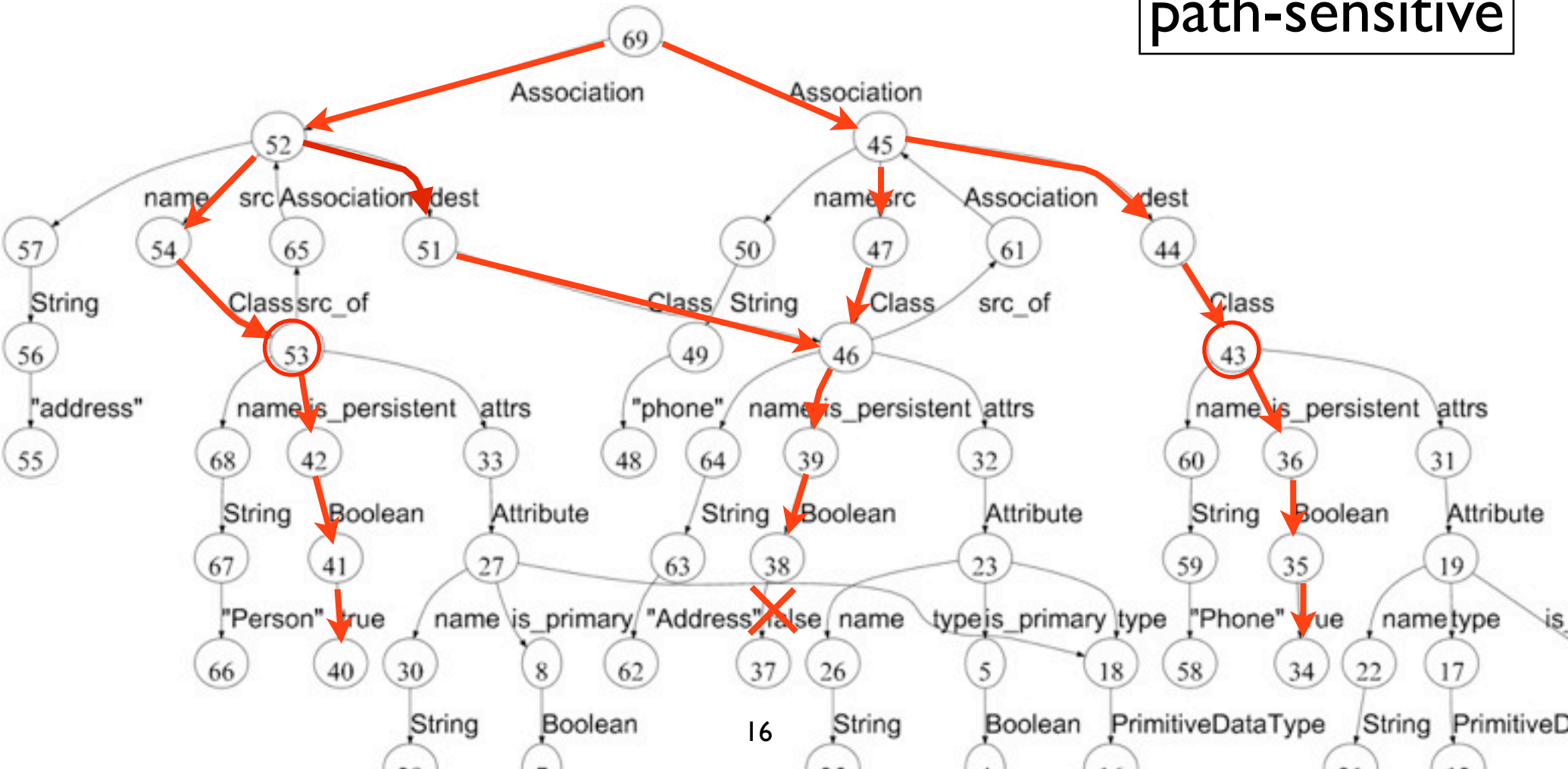

1
S12
S13
S14
S44
d
b
E12
E13
E14
E44
2
3
4
S25
S35
d
d
E25
E35
5
S56
d
E56
6


1
d
b
2
3
d
d
5
d
6

Graph Querying in UnQL

select $*class* where
{*Association*.(*src*|*dest*).*Class*: $*class*} in $*db*,
{*is_persistent*: {*Boolean* : *true*}} in $*class*

path-sensitive



UnQL$^+$: An Extension of UnQL

delete ... where

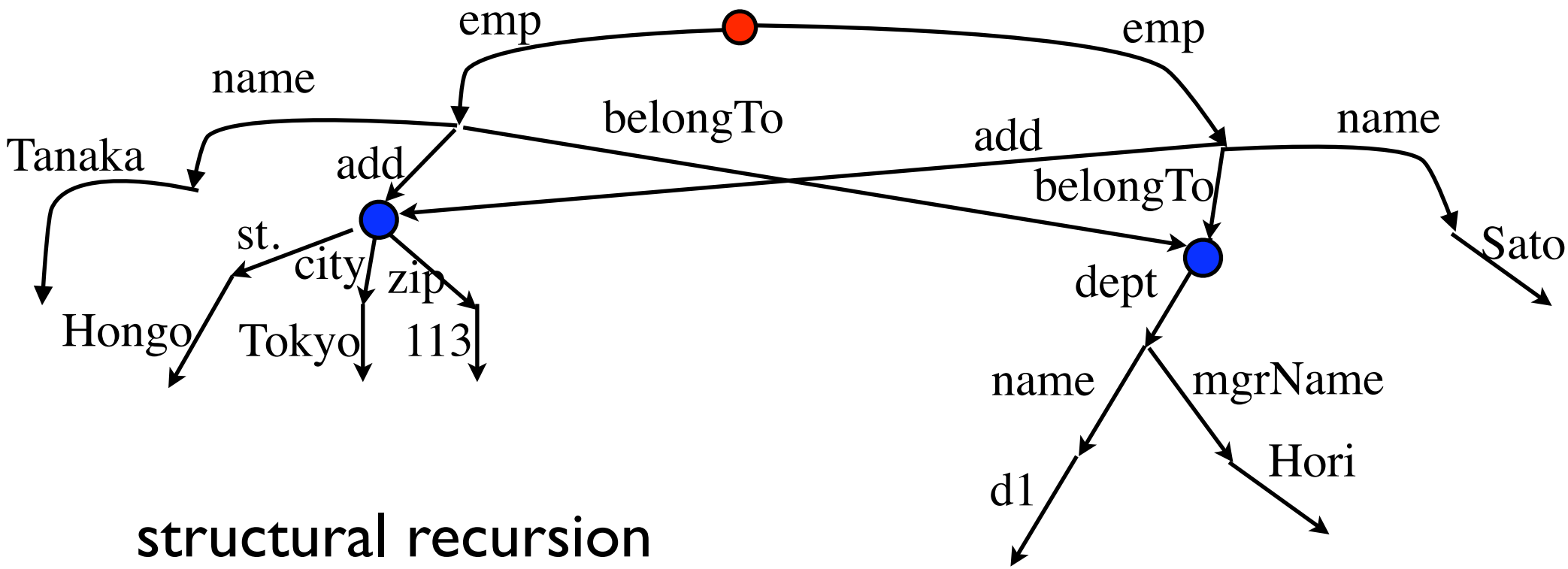
extend ... with ... where

replace ... by ... where

These can be translated into structural recursion.

- "Tanaka" has moved to new address.

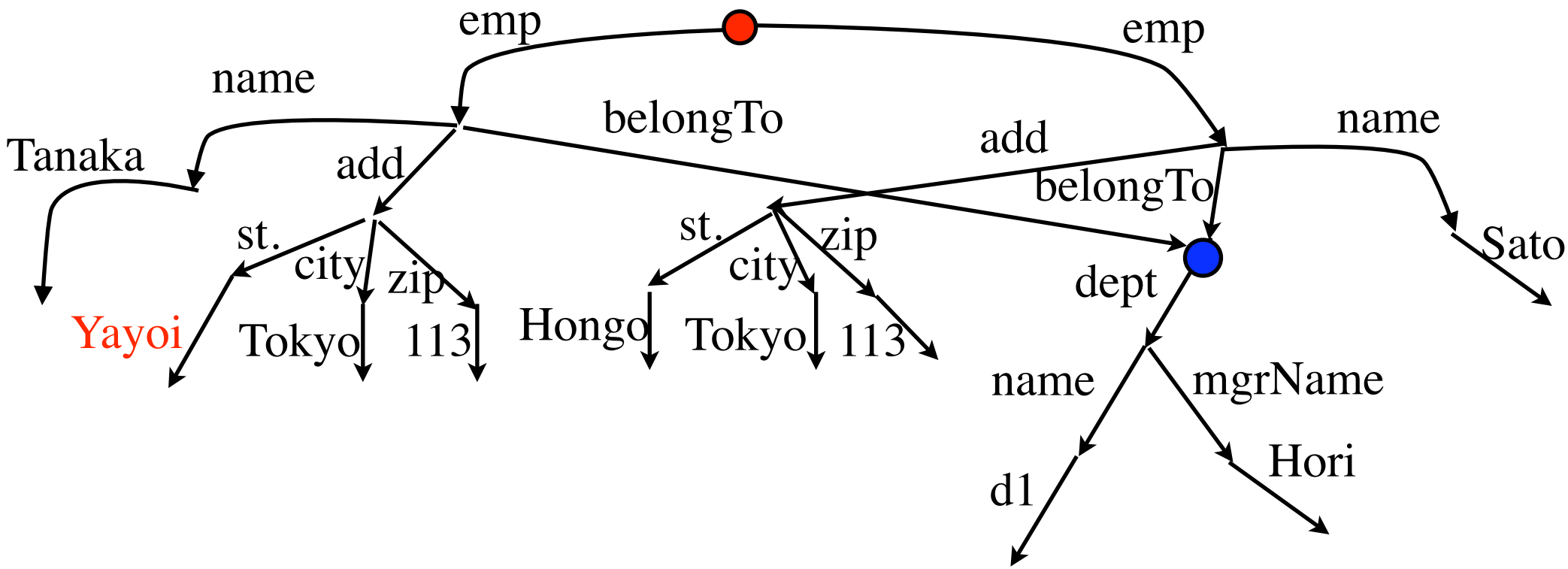
replace *$tgt* **by** {st: {Yayoi: {}}, city: {Tokyo:{}}, zip: {113: {}}}
where {emp:$e} **in** *$db*,
{name: {Tanaka: {}}} **in** $e,
{add: *$tgt*} **in** $e



structural recursion

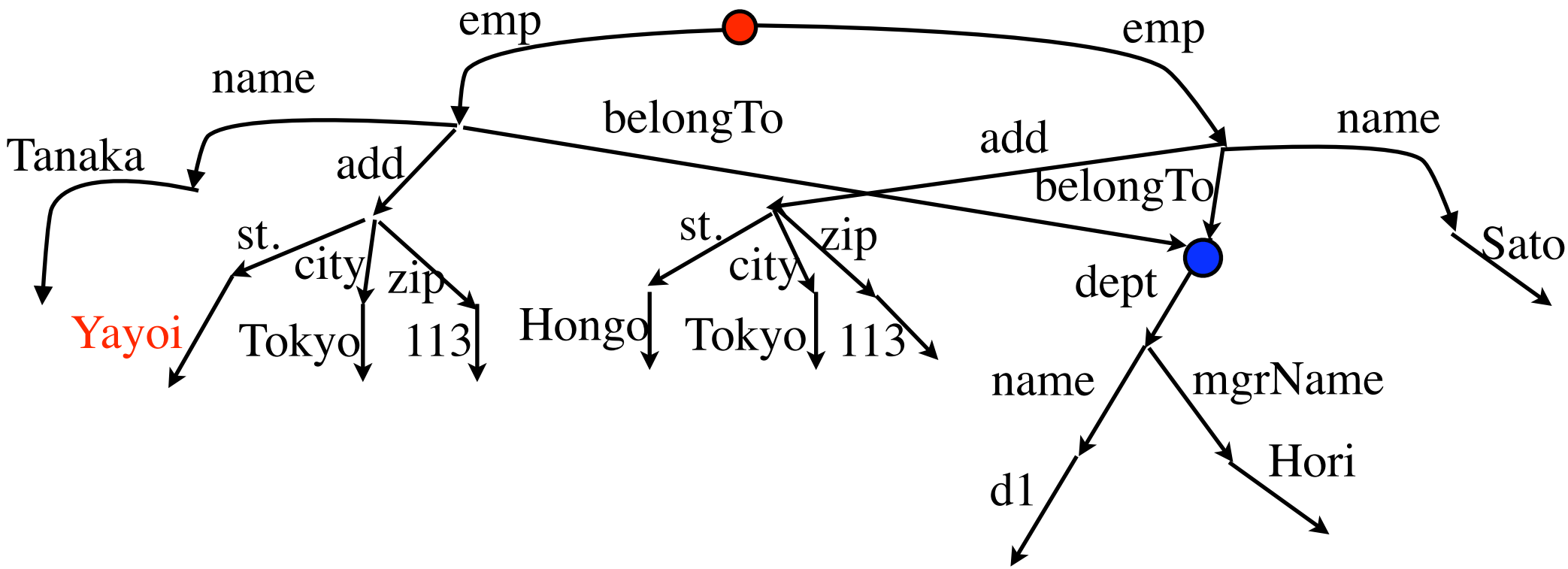
- "Tanaka" has moved to new address.

replace *$tgt* **by** {st: {Yayoi: {}}, city: {Tokyo:{}}, zip: {113: {}}}
where {emp:$e} **in** *$db*,
{name: {Tanaka: {}}} **in** $e,
{add: *$tgt*} **in** $e



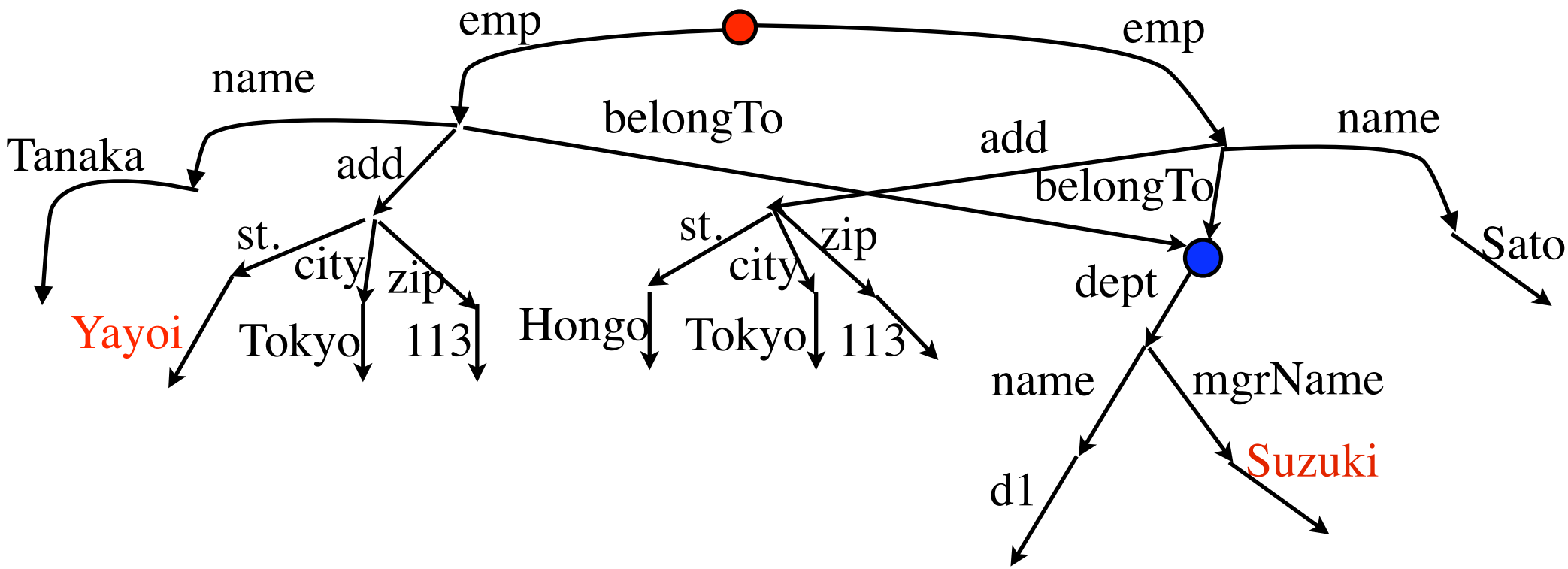
- The manager of d1 has changed.

```
replace $tgt by {Suzuki: {}}
where   {emp.belongTo.dept:$d} in $db,
        {name: {d1: {}}} in $d,
        {mgrName: $tgt} in $d
```



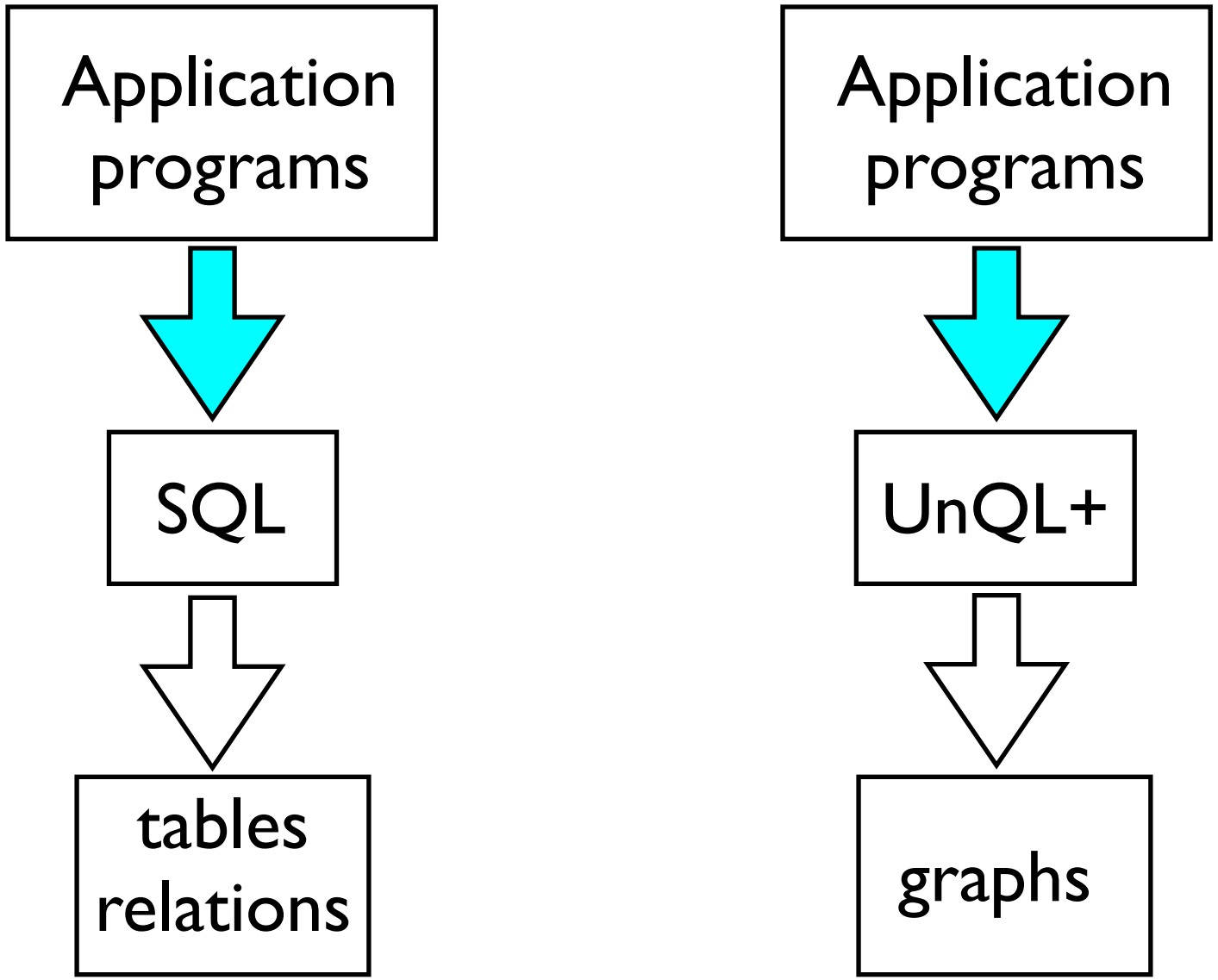
- The manager of d1 has changed.

```
replace $tgt by {Suzuki: {}}
where   {emp.belongTo.dept:$d} in $db,
        {name: {d1: {}}} in $d,
        {mgrName: $tgt} in $d
```



Conclusion

- There are two semantics in updating graphs.
- UnQL+(**replace** expression) can handle these two.


Application programs
SQL
tables relations
Application programs
UnQL+
graphs

Thank you