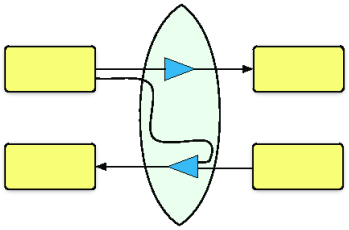
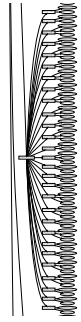
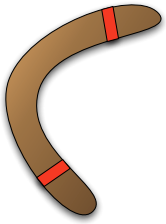
Bidirectional Programming Languages

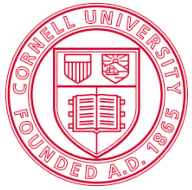


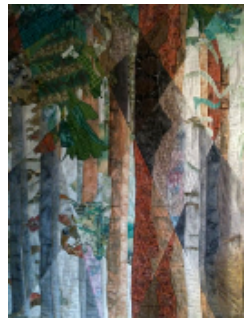
Nate Foster
Cornell University



CS 2110
24 November 2015







Most programming languages, like C...

Java,

Python,

and C++ are general purpose.

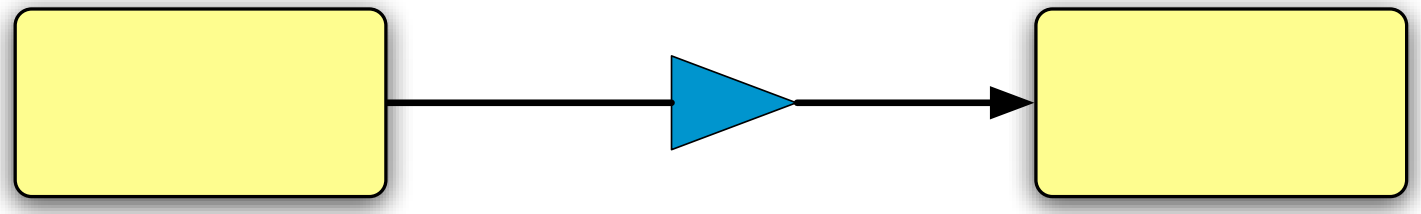
I’m interested in designing languages that are *specifically designed* for particular tasks

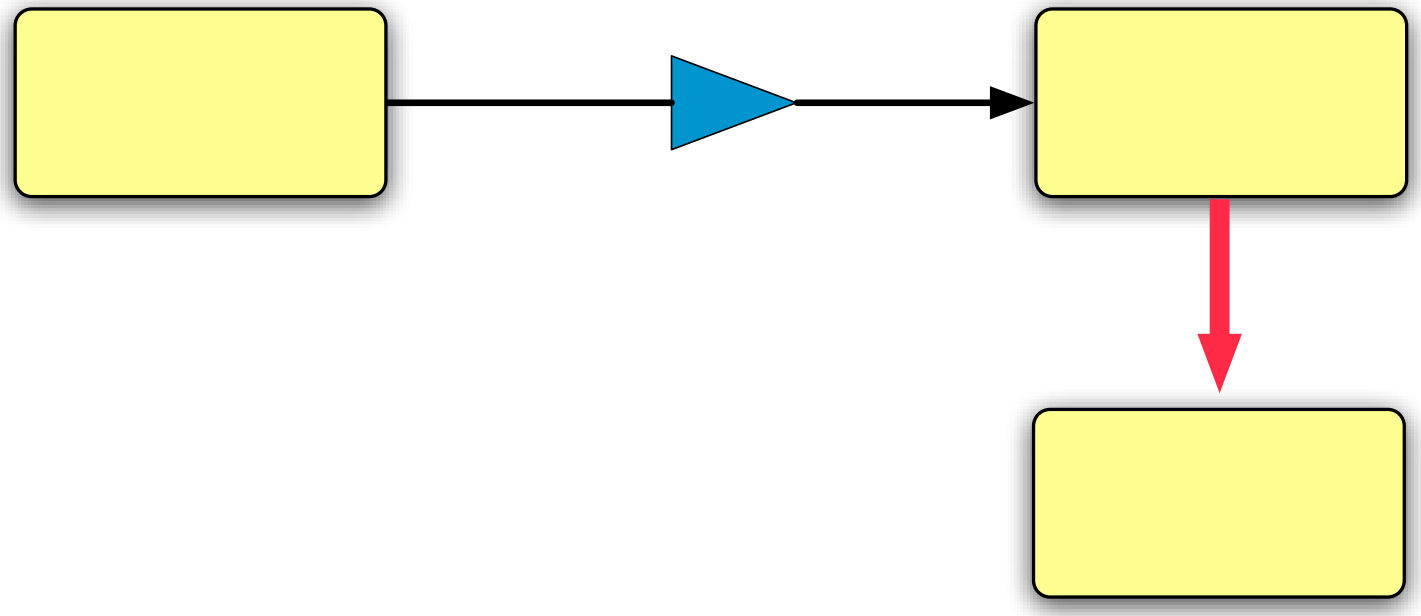
Domain-specific languages

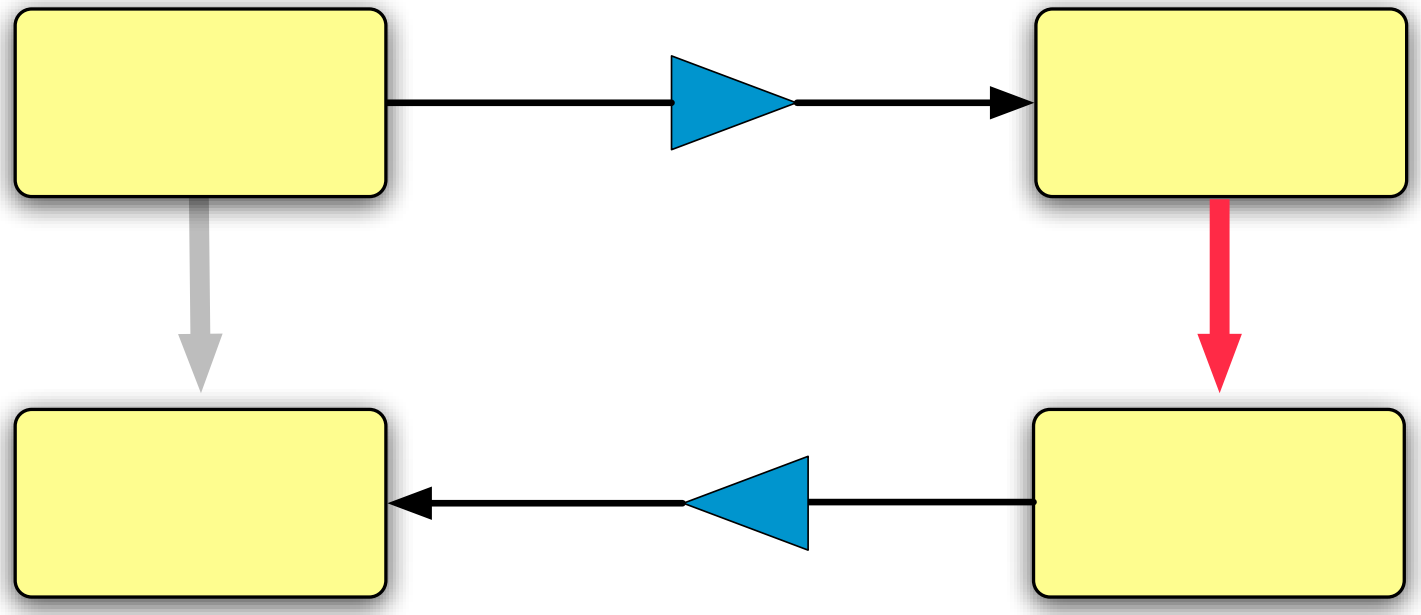
- Clean semantics
- Natural syntax
- Better tools


convert
update
exchange
replicate
redact
integrate
query
analyze
mashup
transform
clean
data
curate
reconcile
modify
hide
maintain
synchronize
extract
summarize
evolve


convert
update
exchange
replicate
redact
integrate
query
analyze
mashup
clean
data
transform
curate
reconcile
maintain
modify
hide
synchronize
extract
summarize
evolve

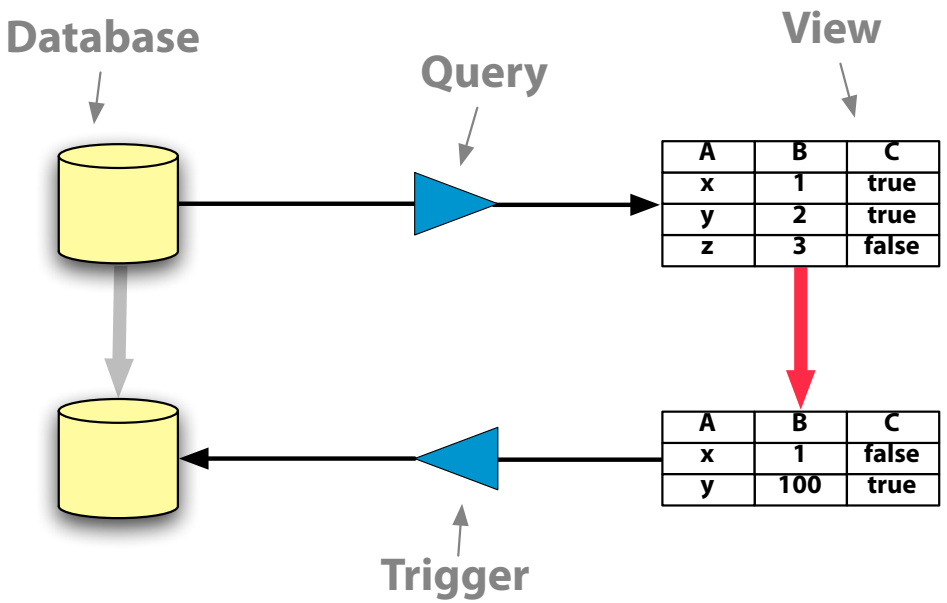






The View Update Problem

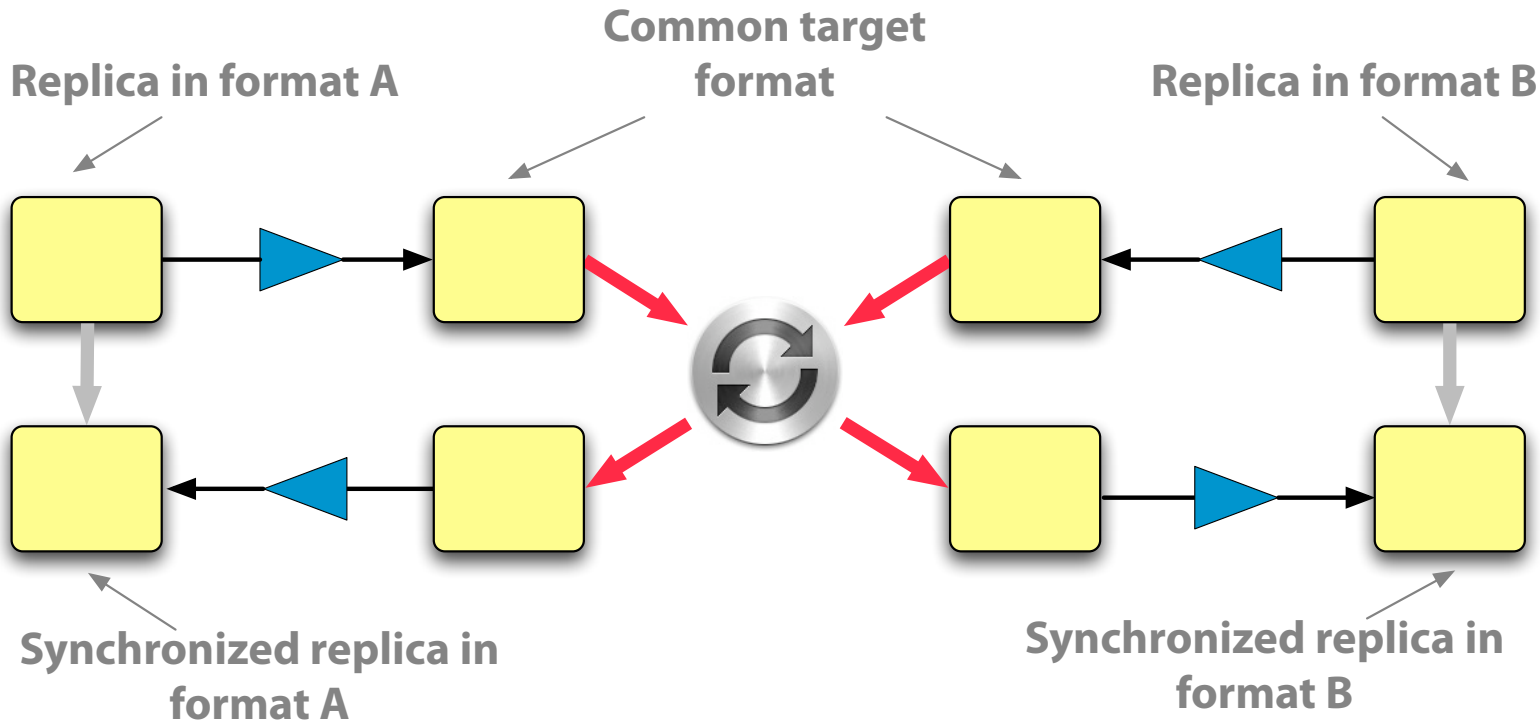
In databases, this is known as the view update problem.



[Bancilhon, Spryatos '81]

The View Update Problem In Practice

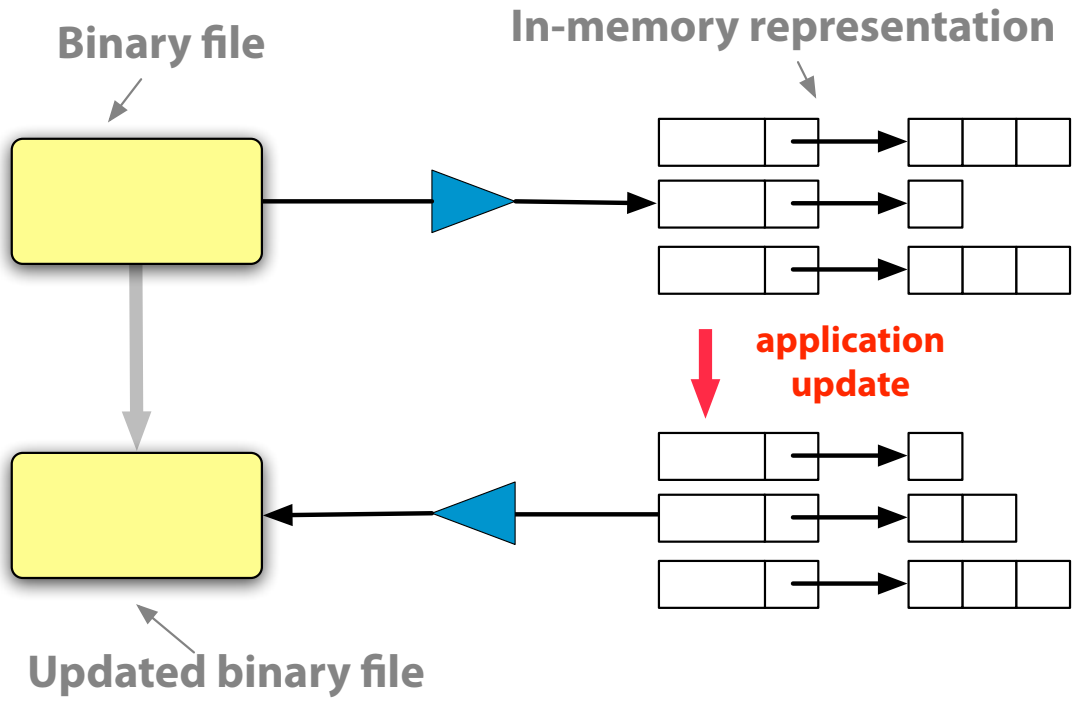
It also arises in data converters and synchronizers...



[Foster, Greenwald, Pierce, Schmitt JCSS '07]— Harmony

The View Update Problem In Practice

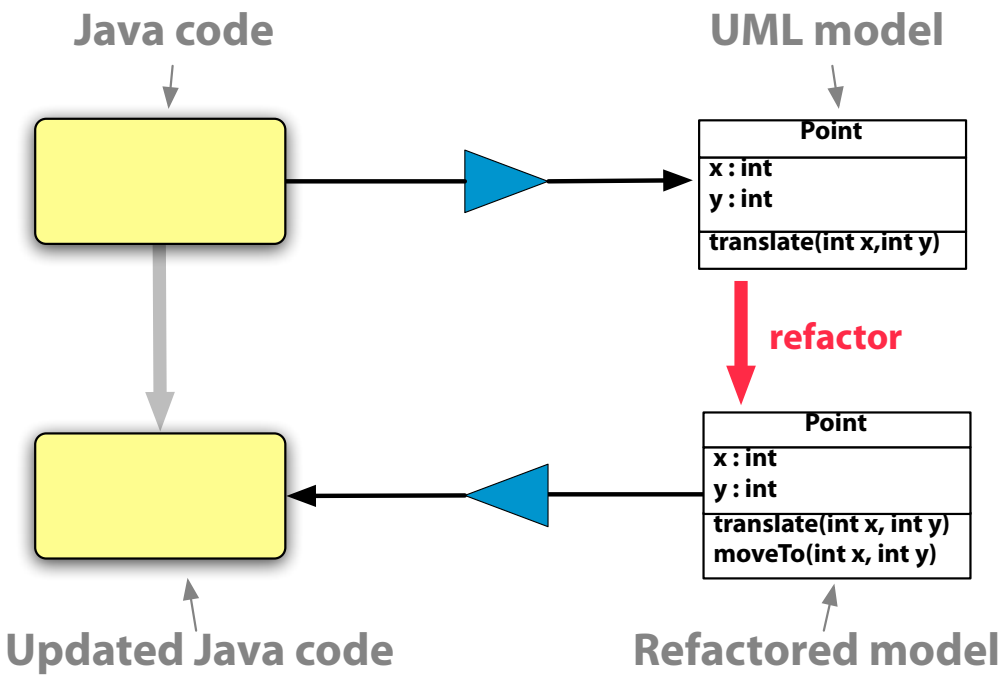
...in picklers and unpicklers...



[Fisher, Gruber '05]— PADS

The View Update Problem In Practice

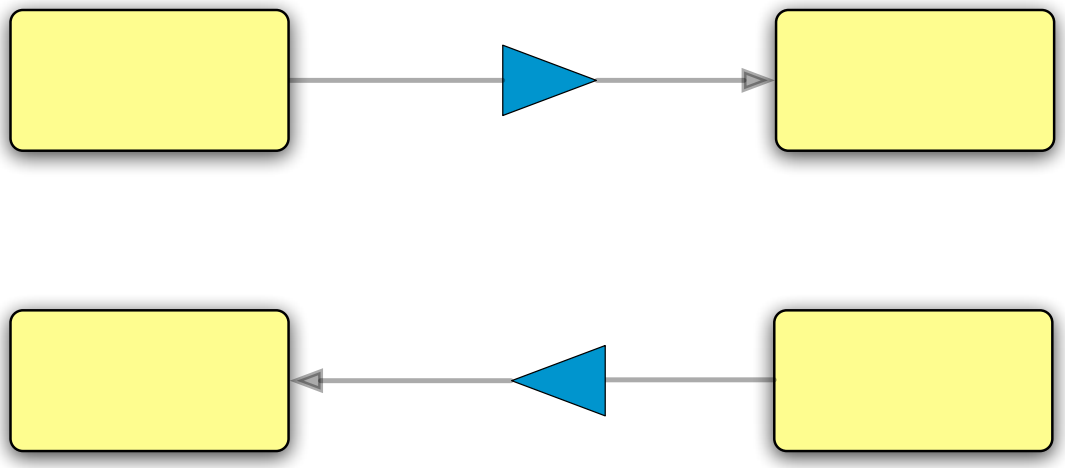
...in model-driven software development...



[Stevens '07]— bidirectional model transformations

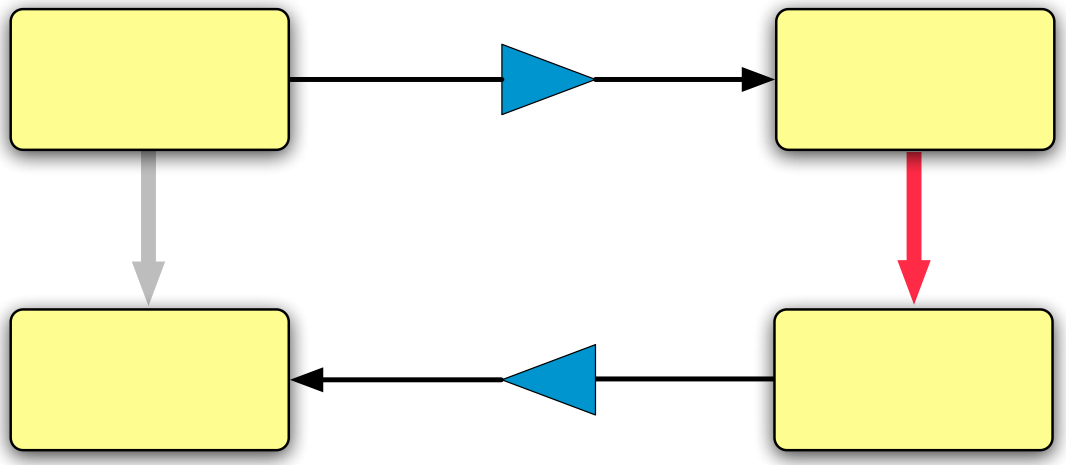
Problem

How do we write these bidirectional transformations?



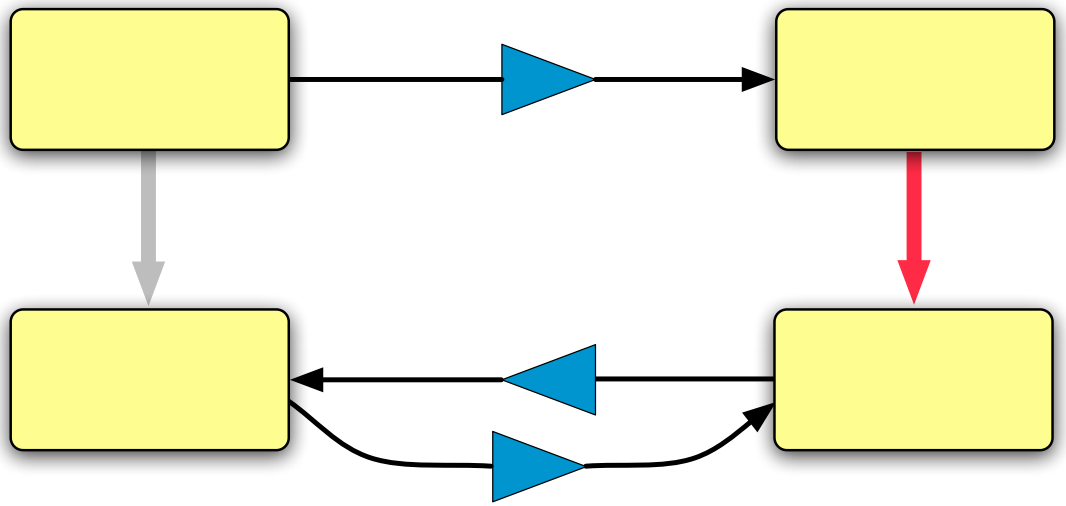
Problem: Why is it hard?

We want updates to the view to be translated “exactly”…



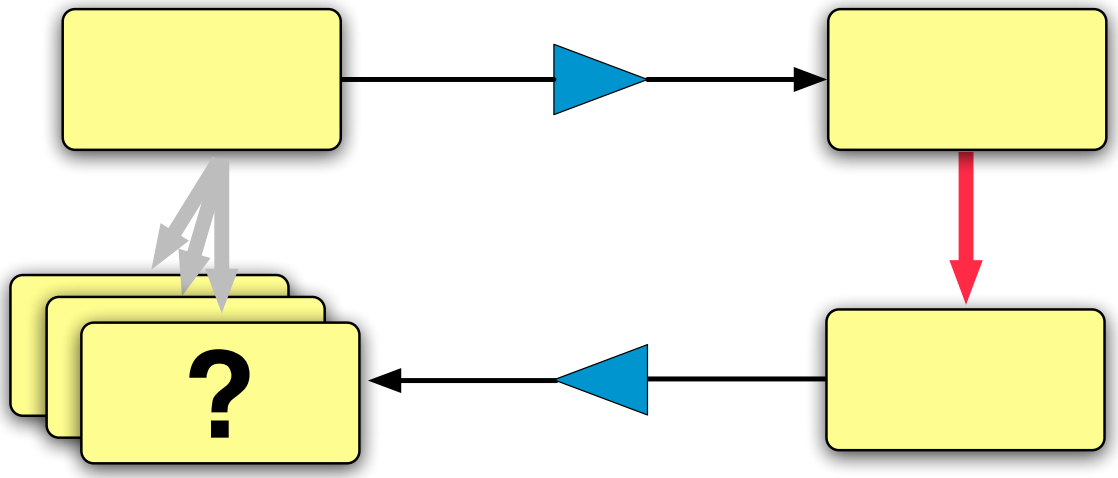
Problem: Why is it hard?

We want updates to the view to be translated “exactly”…



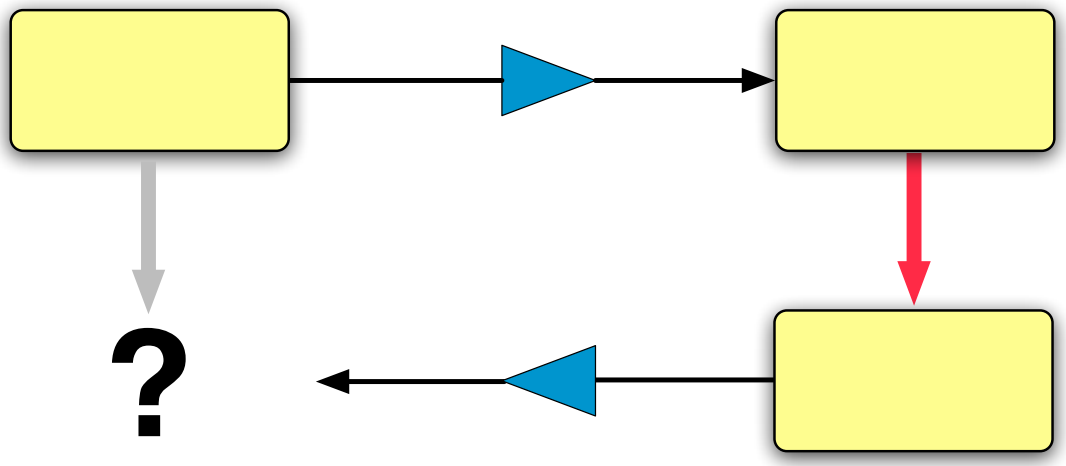
Problem: Why is it hard?

...but some updates have *many* corresponding source updates...

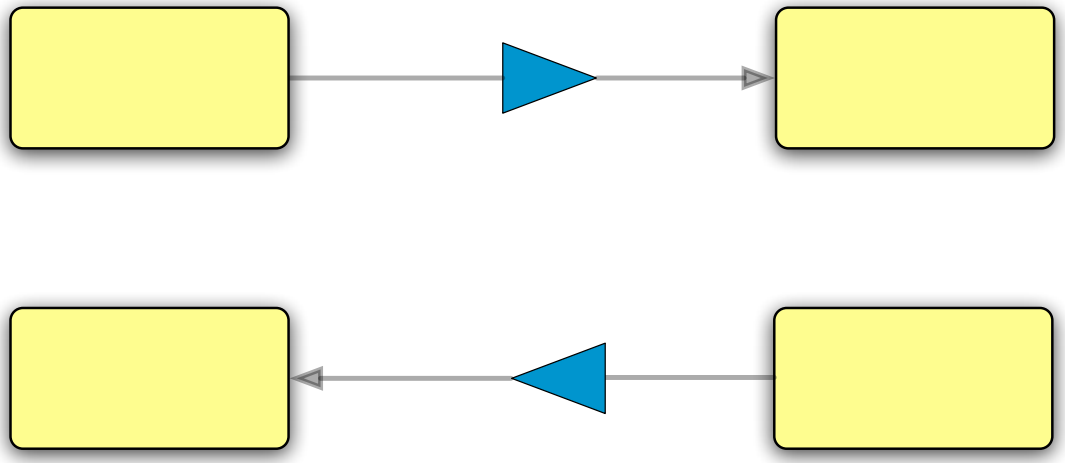


Problem: Why is it hard?

...while others have *none*!



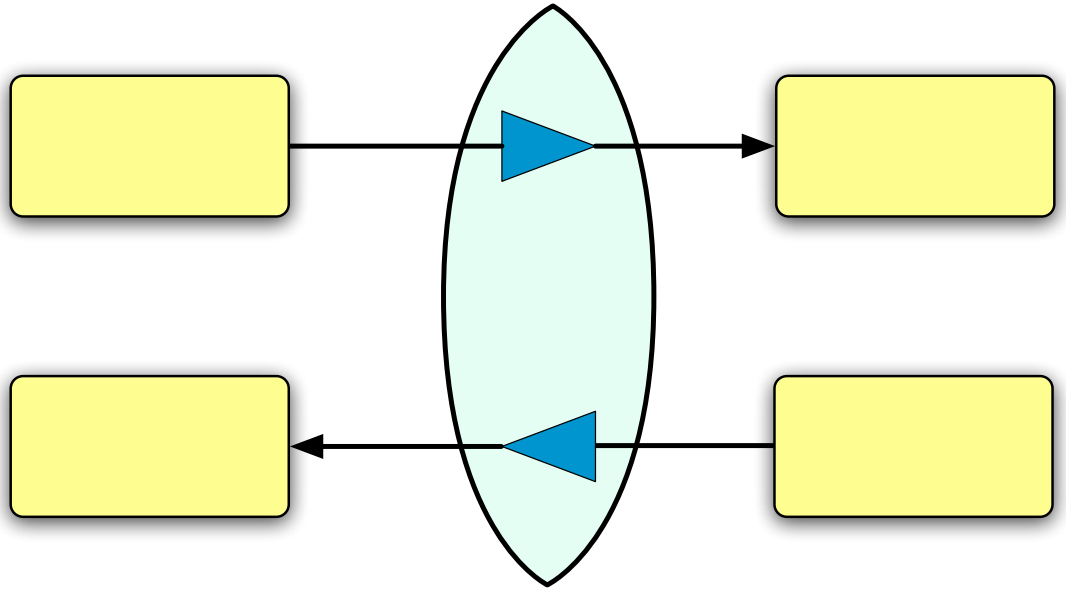
Possible Approaches



Bad: write the two transformations as separate functions.

- tedious to program
- difficult to get right
- a nightmare to maintain

Possible Approaches



Good: derive both transformations from the same program.

- Clean semantics: behavioral laws guide language design
- Natural syntax: parsimonious and compositional
- Better tools: type system guarantees well-behavedness

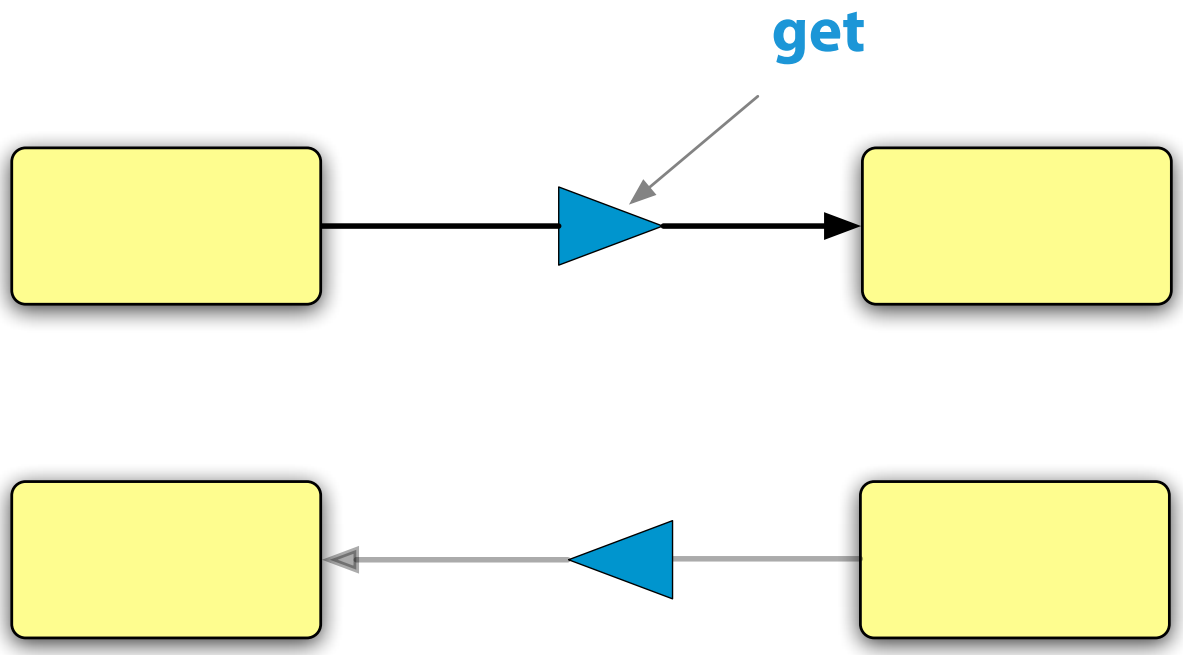
“Bidirectional languages are an effective and elegant means of describing updatable views”

[Foster, Greenwald, Moore, Pierce, Schmitt TOPLAS '07]

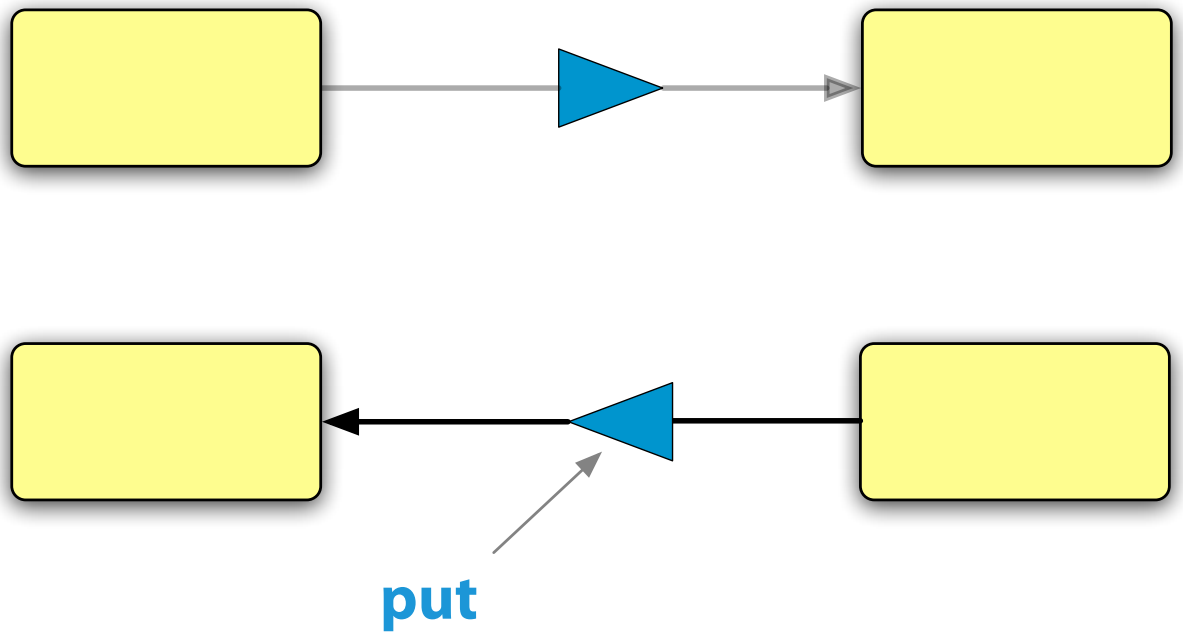
Lenses

''Never look back unless
you are planning to go that way"
—H D Thoreau

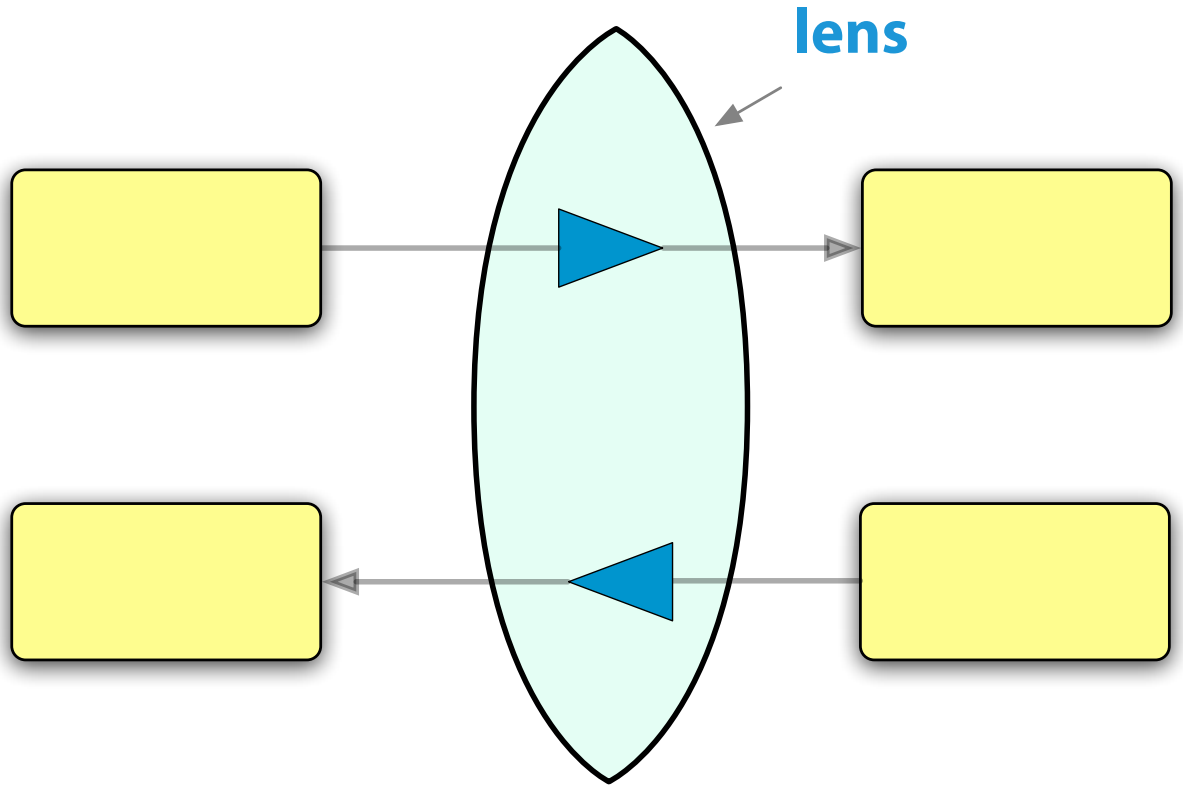
Terminology



Terminology

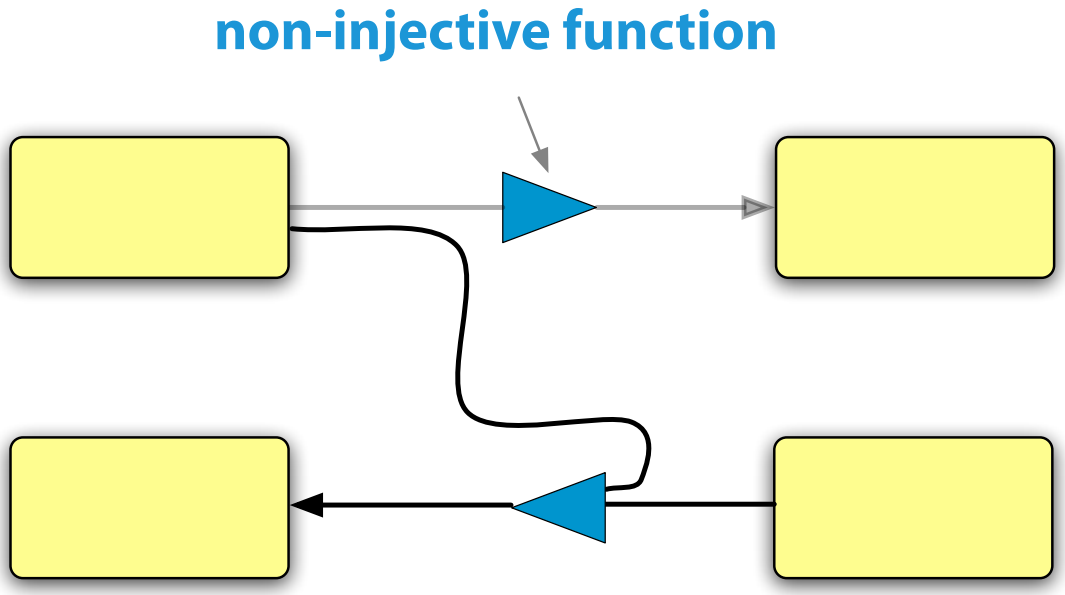


Terminology



Bidirectional vs. Bijective

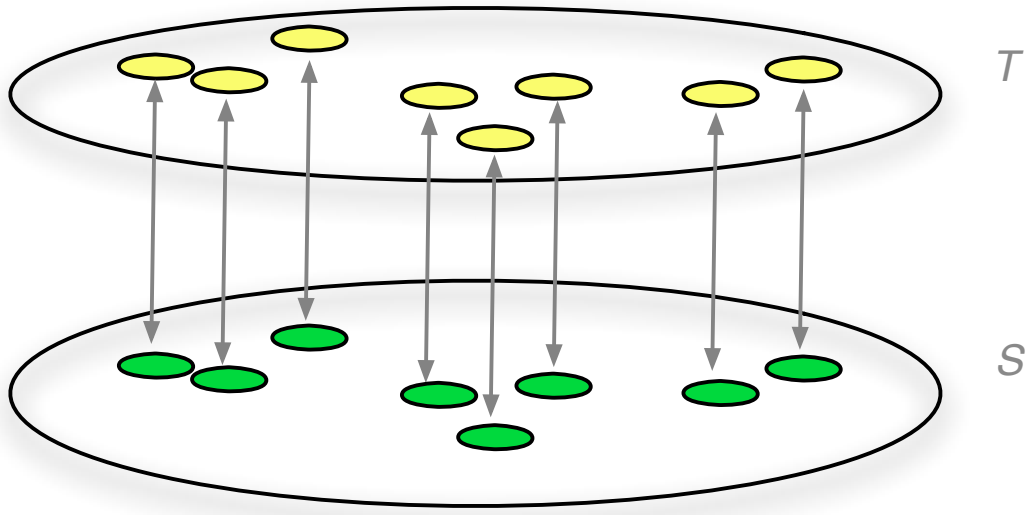
If **get** is non-injective, **put** needs access to the original source.



Of course, the purely bijective case is also interesting.

The Bijective Case

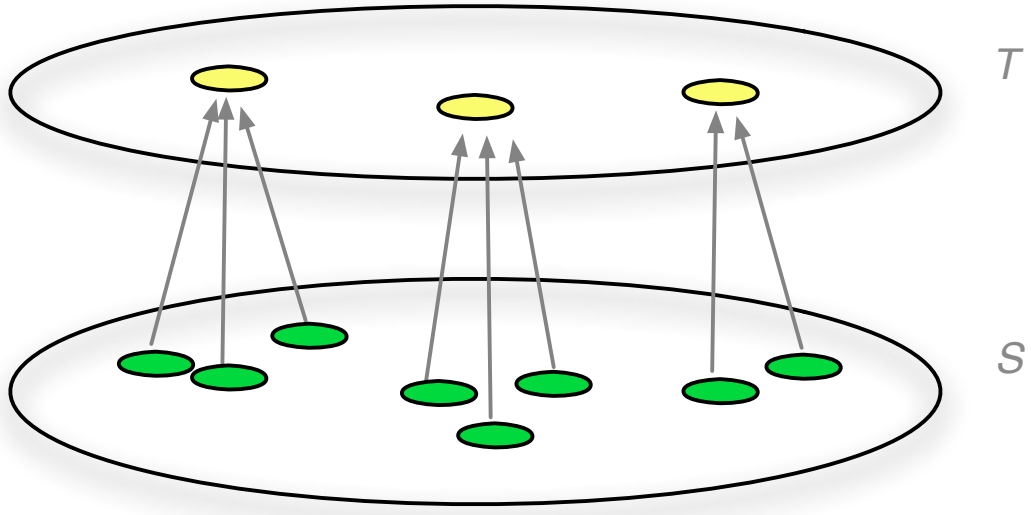
For bijective transformations...



...the desired behavior is obvious.

The General Case

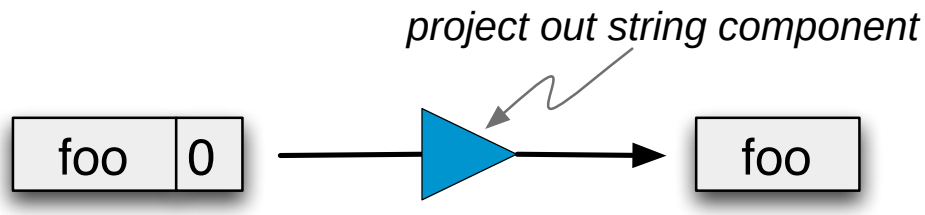
But for bidirectional transformations...



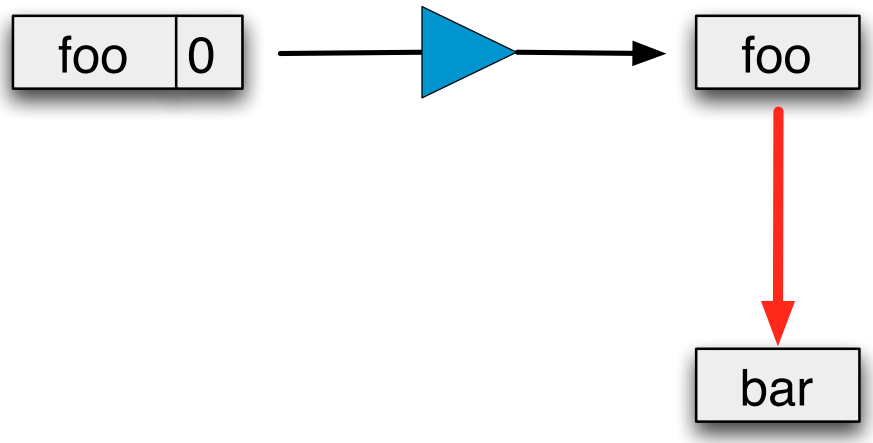
... we need to identify conditions that allow us to

- recognize and reject bad (unreasonable) programs
- understand and predict behavior

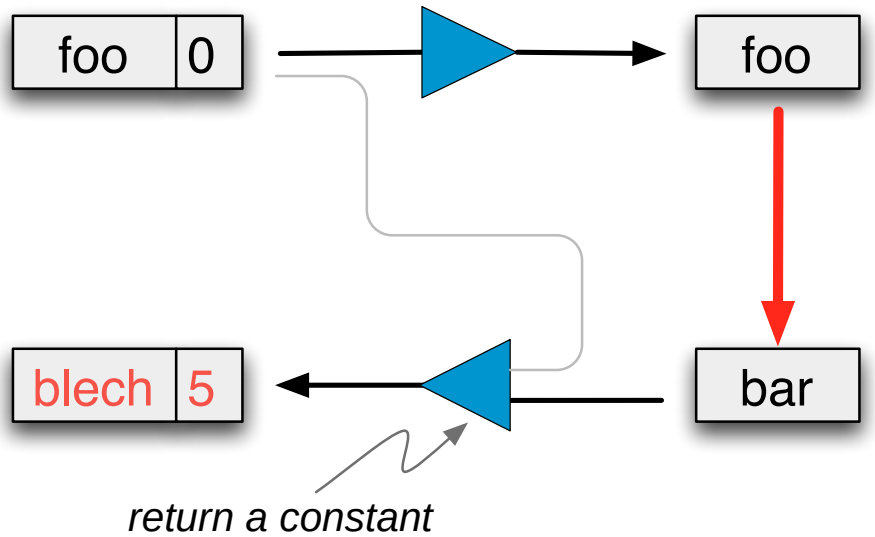
An Unreasonable Example



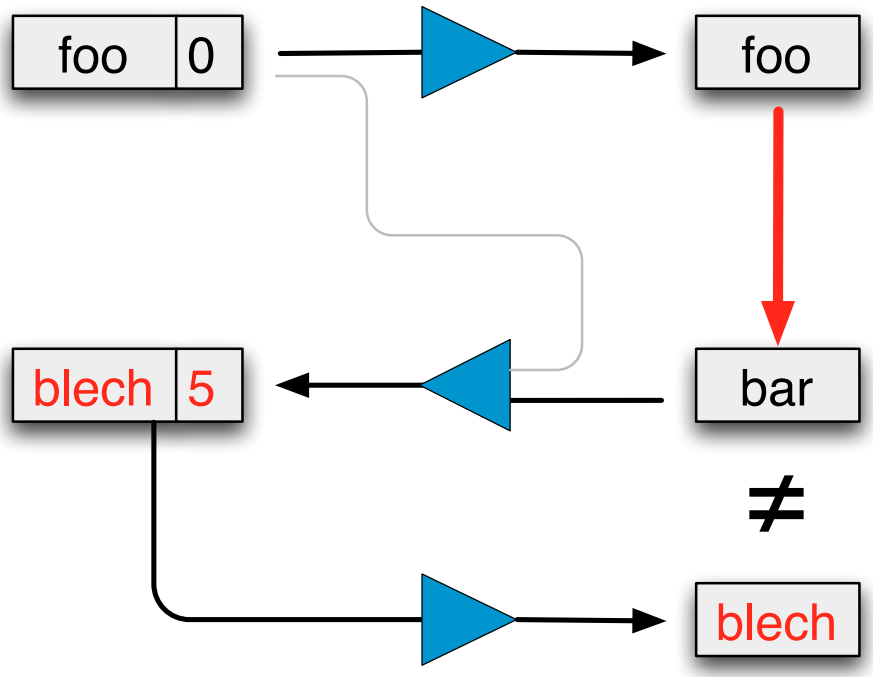
An Unreasonable Example



An Unreasonable Example



An Unreasonable Example



The PutGet law

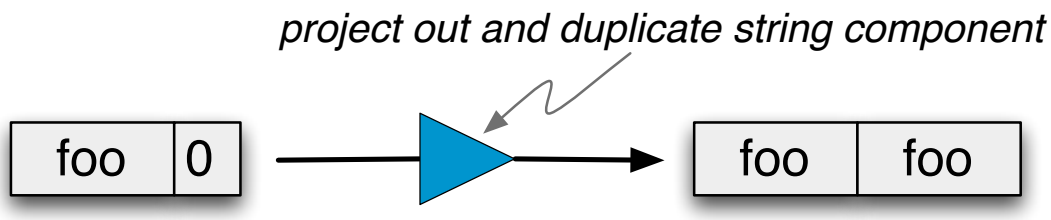
Principle:

> *Updates should be "translated exactly" — i.e., to a source for which* **get** *yields exactly the updated target.*

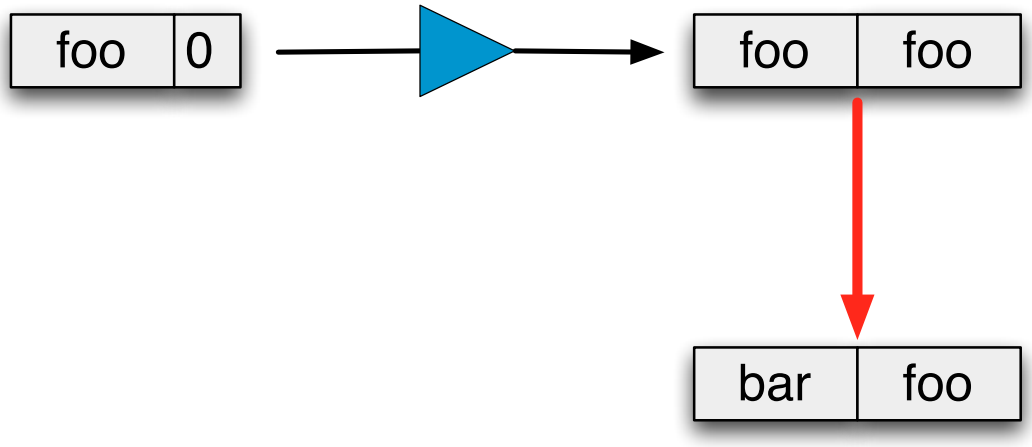
Formally:

$$\textbf{get}\ (\textbf{put}\ v\ s) \quad = \quad v$$

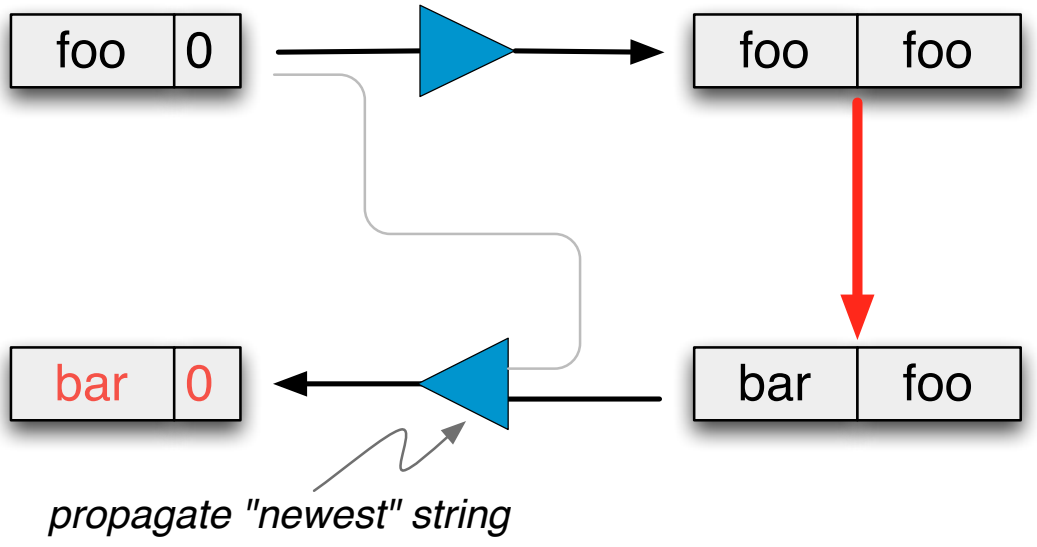
A Debatable Example



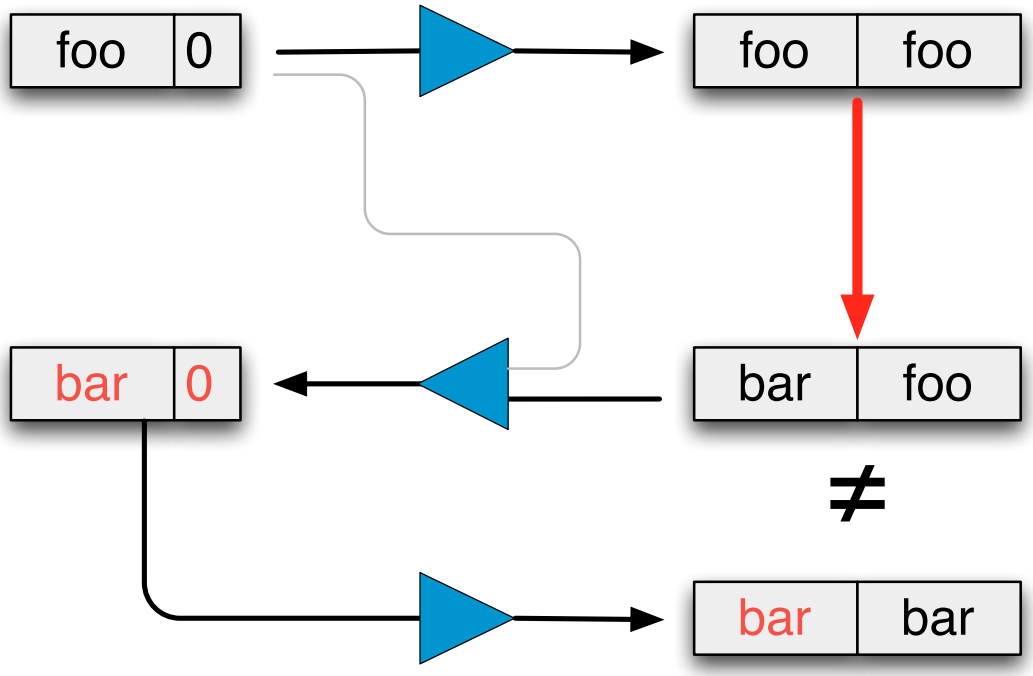
A Debatable Example



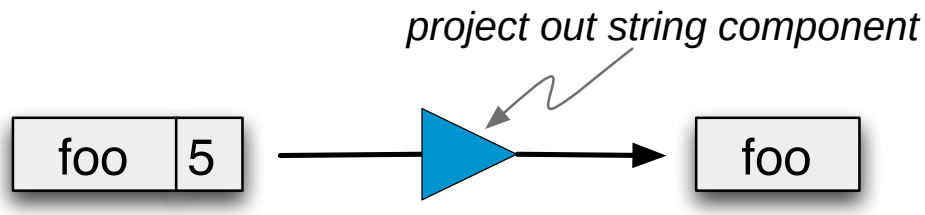
A Debatable Example



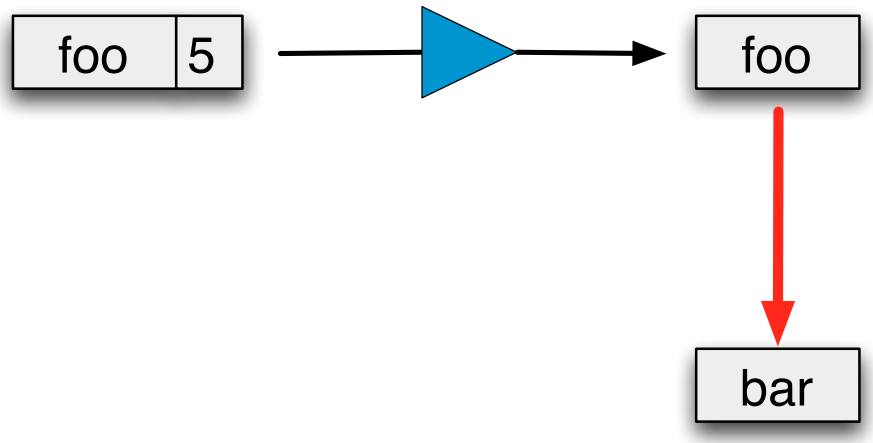
A Debatable Example



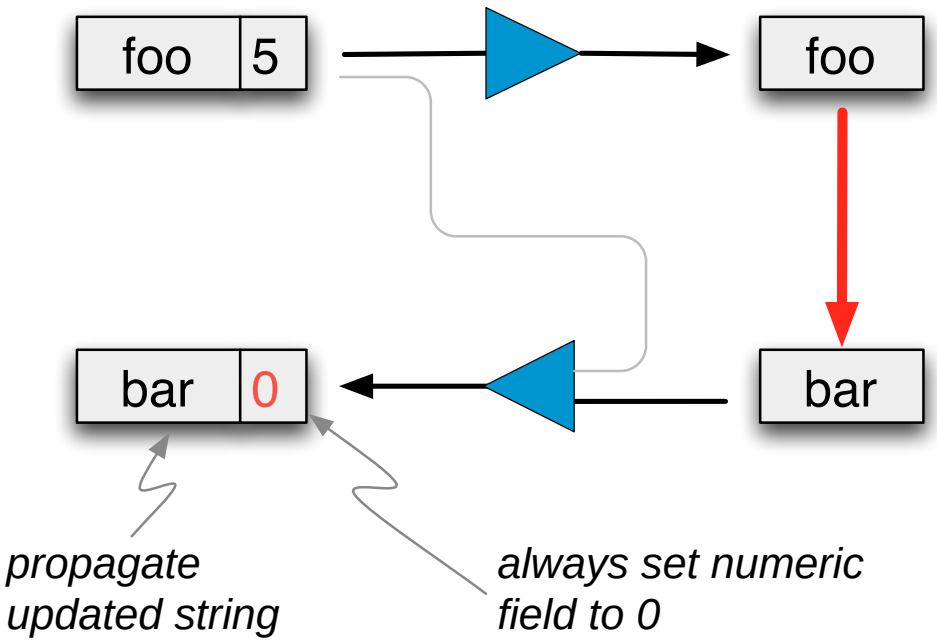
Another Unreasonable Example



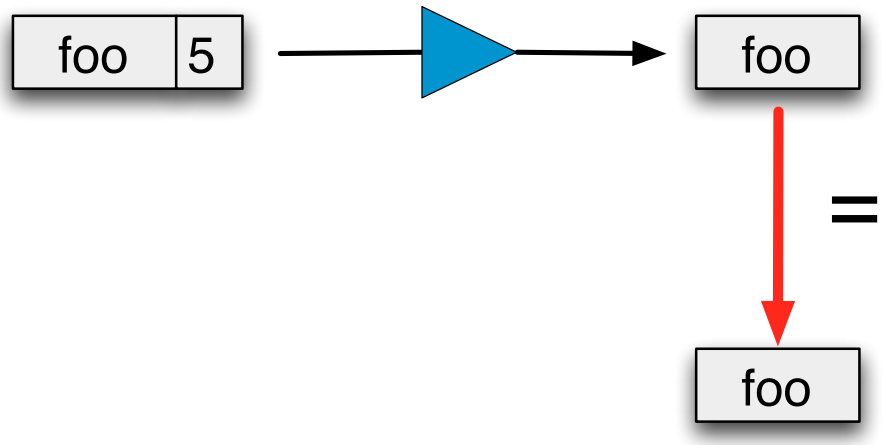
Another Unreasonable Example



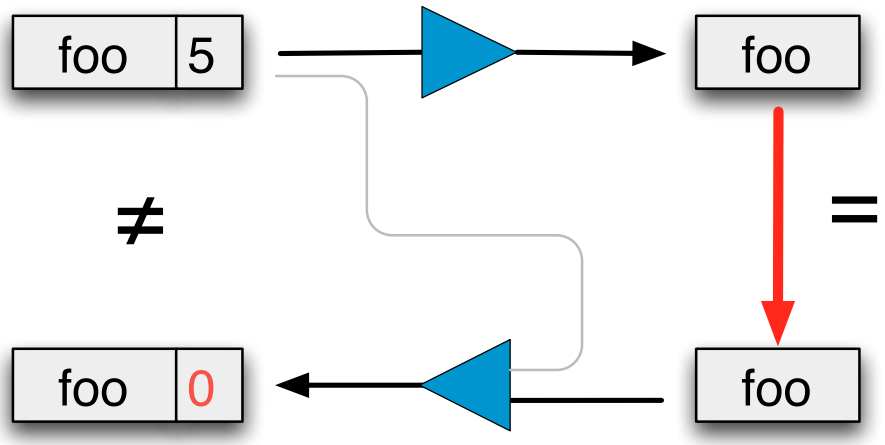
Another Unreasonable Example



Another Unreasonable Example



Another Unreasonable Example



The GetPut law

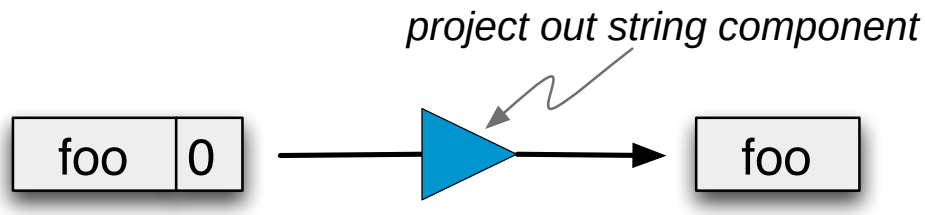
Principle:

If the view does not change, neither should the source.

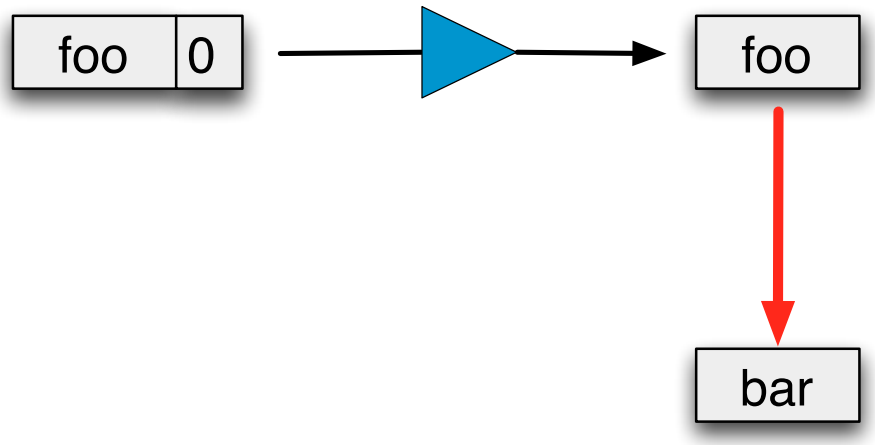
Formally:

$$\textbf{put}\ (\textbf{get}\ s)\ s \quad = \quad s$$

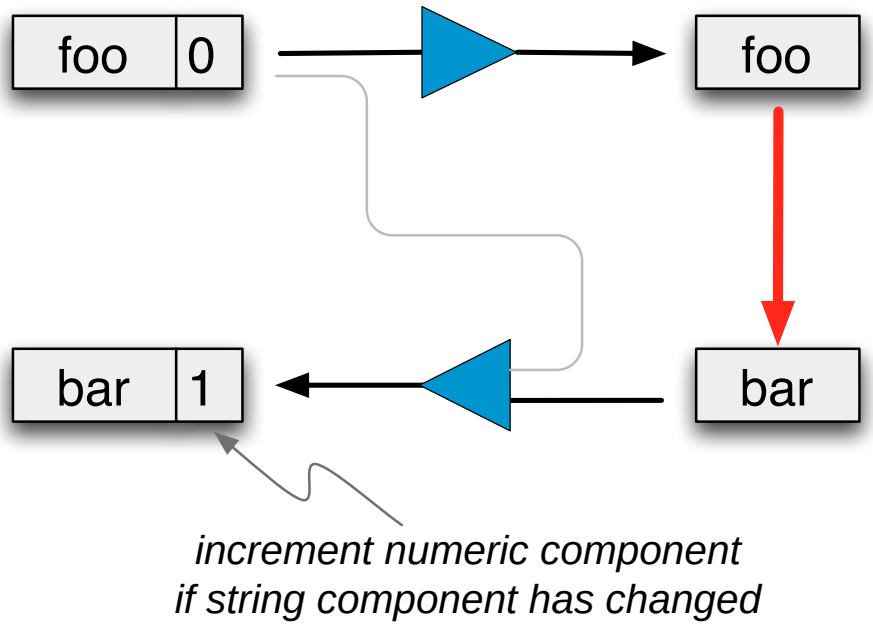
Another Debatable Example



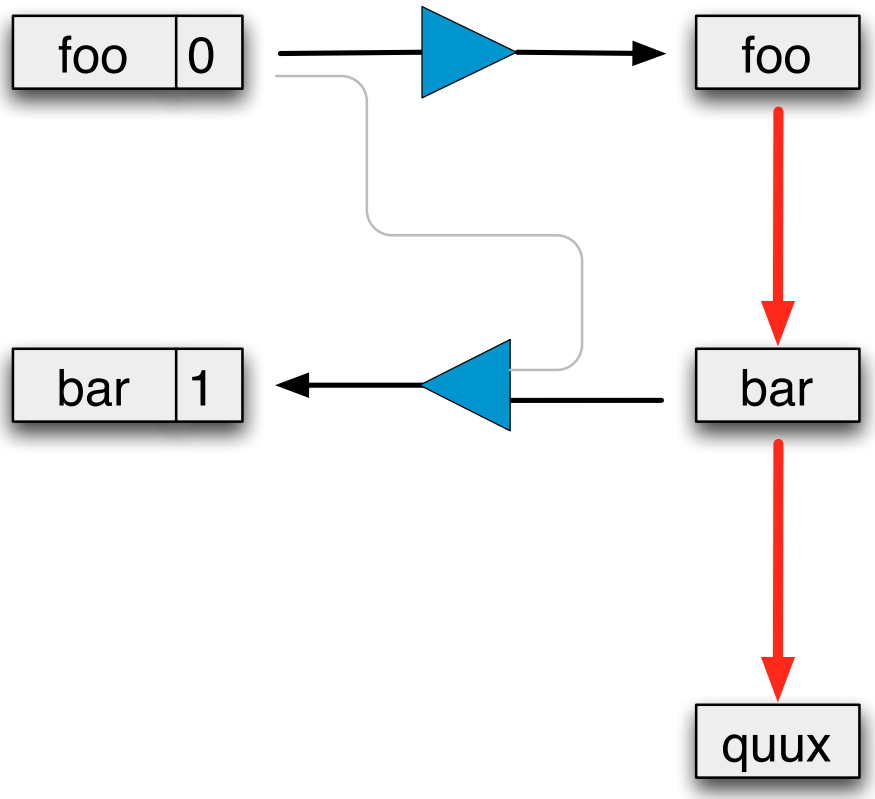
Another Debatable Example



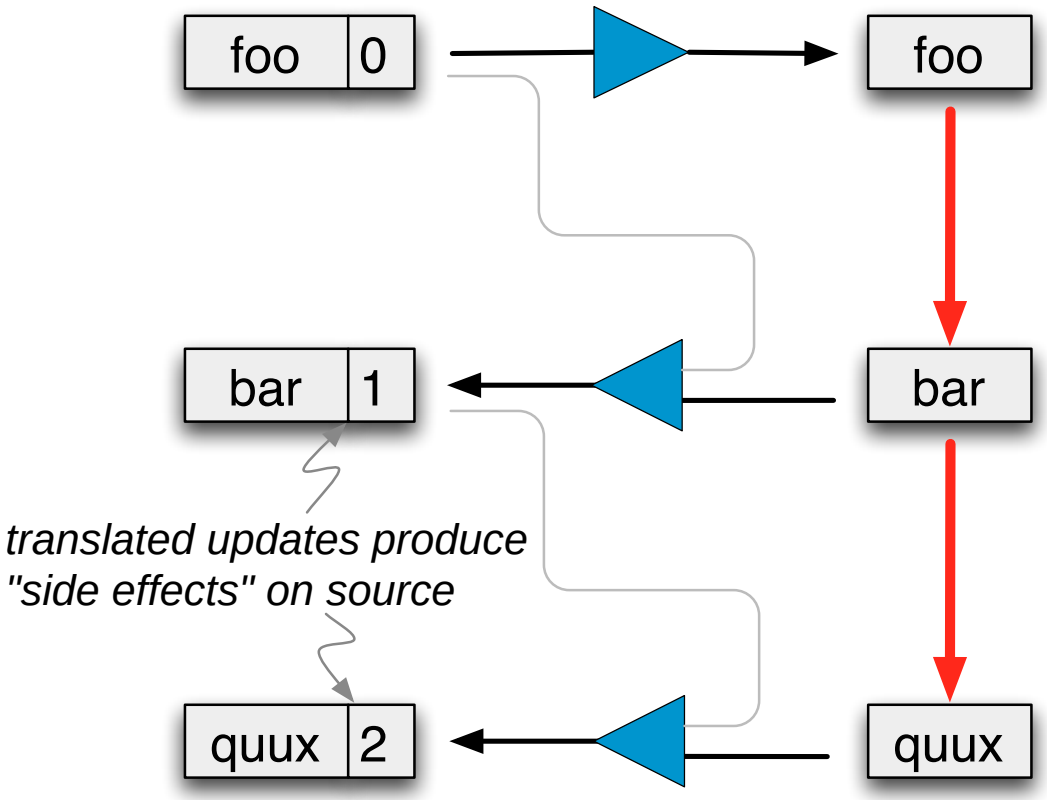
Another Debatable Example



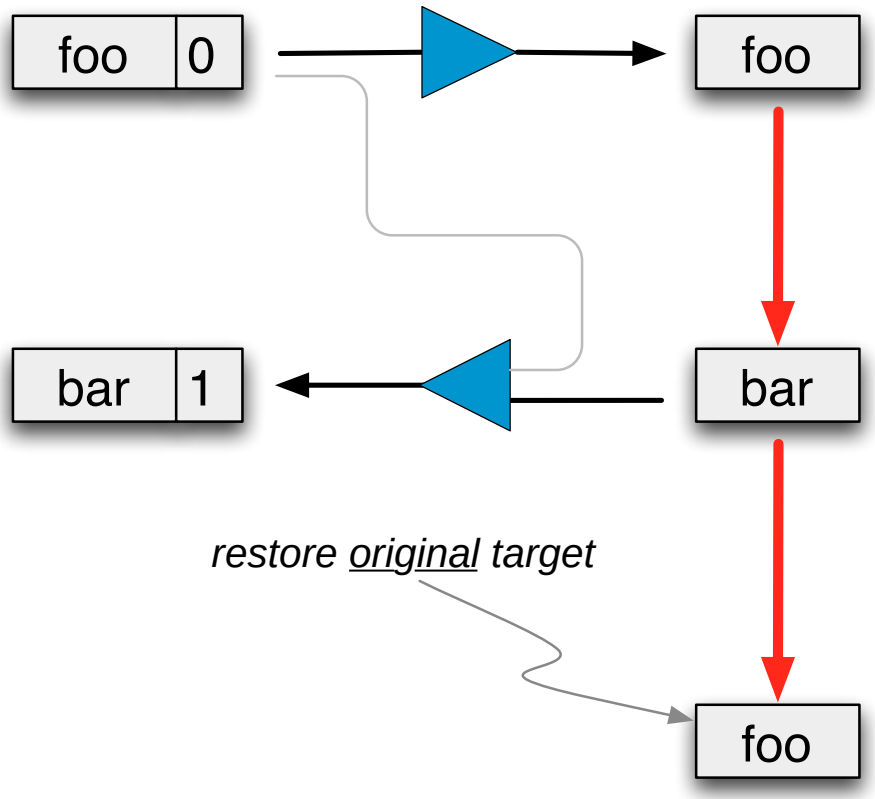
Another Debatable Example



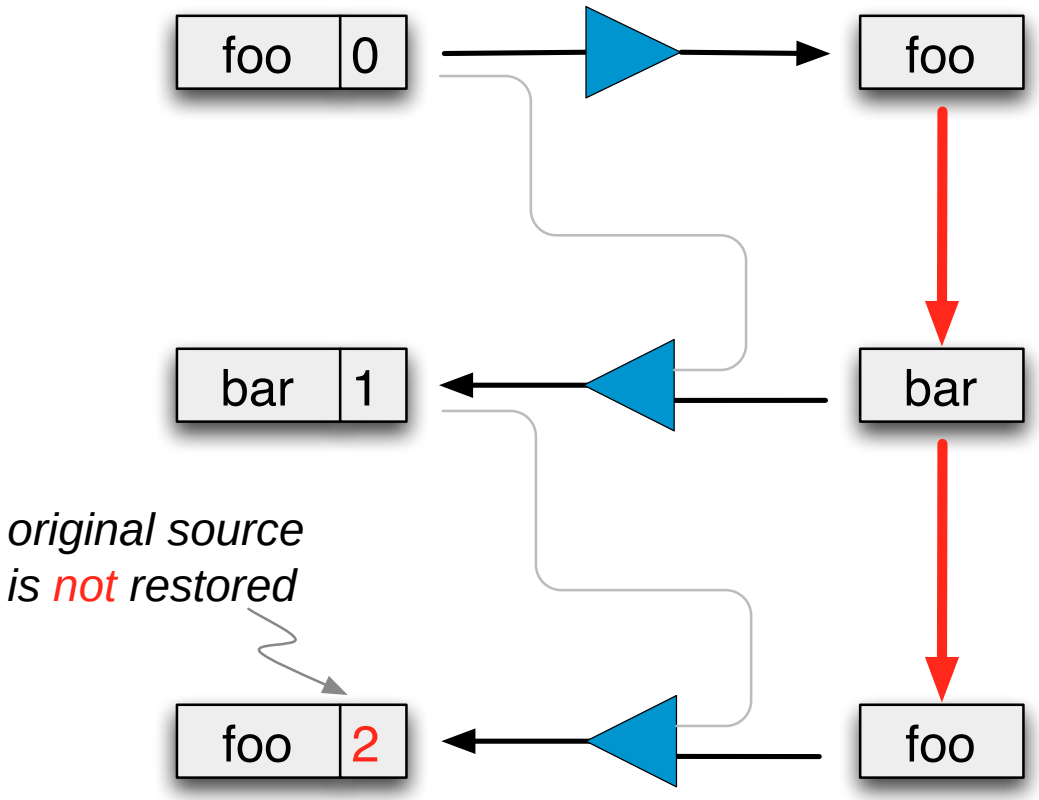
Another Debatable Example



Another Debatable Example



Another Debatable Example



The PutPut law

Principle:

Each update should completely overwrite the effect of the previous one. In particular, the effect of two puts in a row should be the same as just the second.

Formally:

$$\textbf{put}\ v_2\ (\textbf{put}\ v_1\ s) \quad = \quad \textbf{put}\ v_2\ s$$

The PutPut law

Principle:

Each update should completely overwrite the effect of the previous one. In particular, the effect of two puts in a row should be the same as just the second.

Formally:

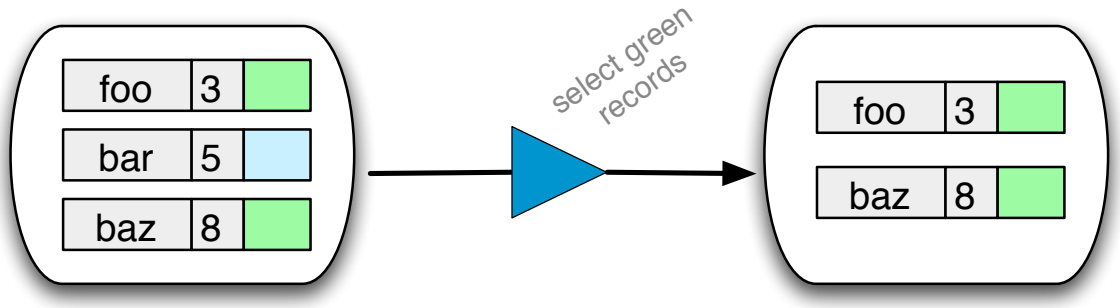
$$\textbf{put}\ v_2\ (\textbf{put}\ v_1\ s) \quad = \quad \textbf{put}\ v_2\ s$$

Nice properties:

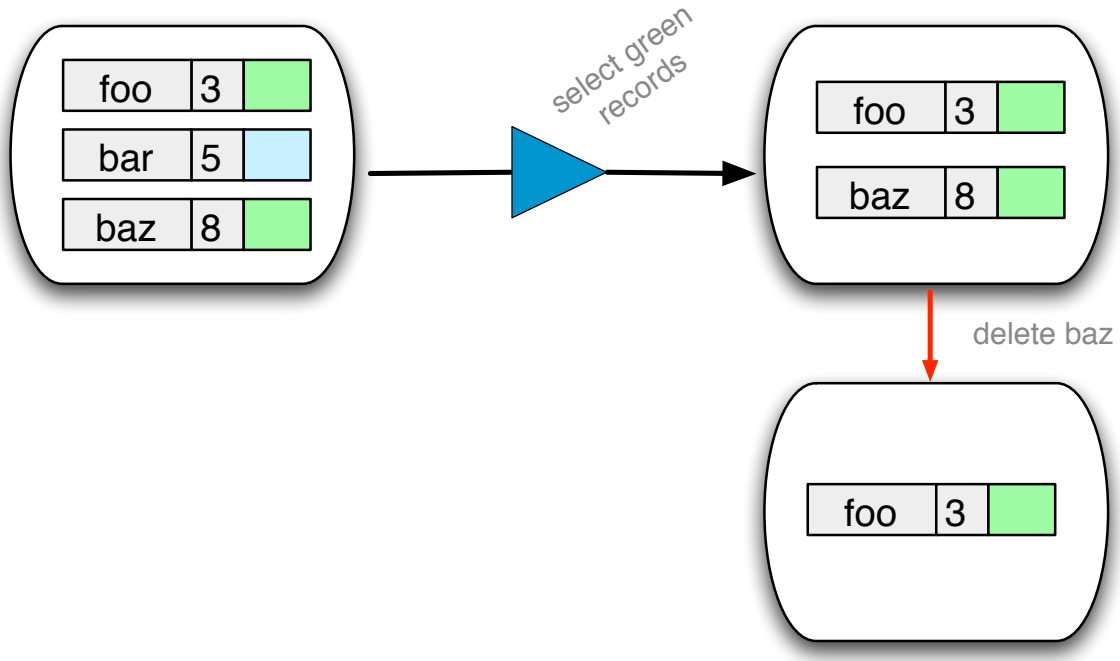
- Ensures that every update can be "rolled back"
- Implies that S is isomorphic to $V \times C$, for some C

Seems sensible. But do we want to always require it?

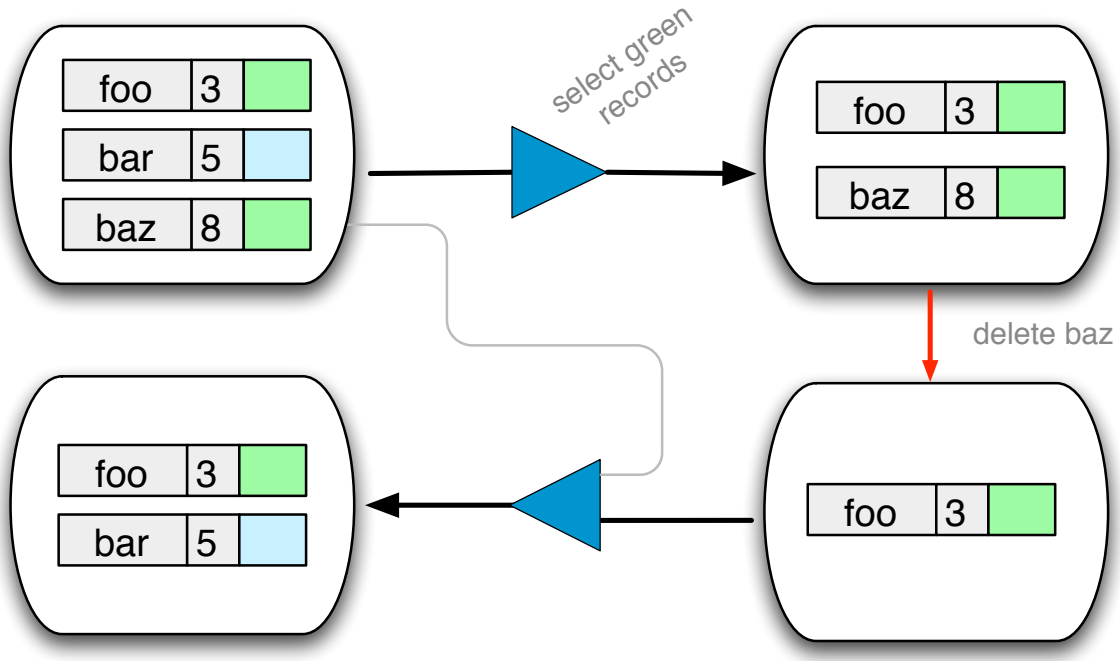
Another Example



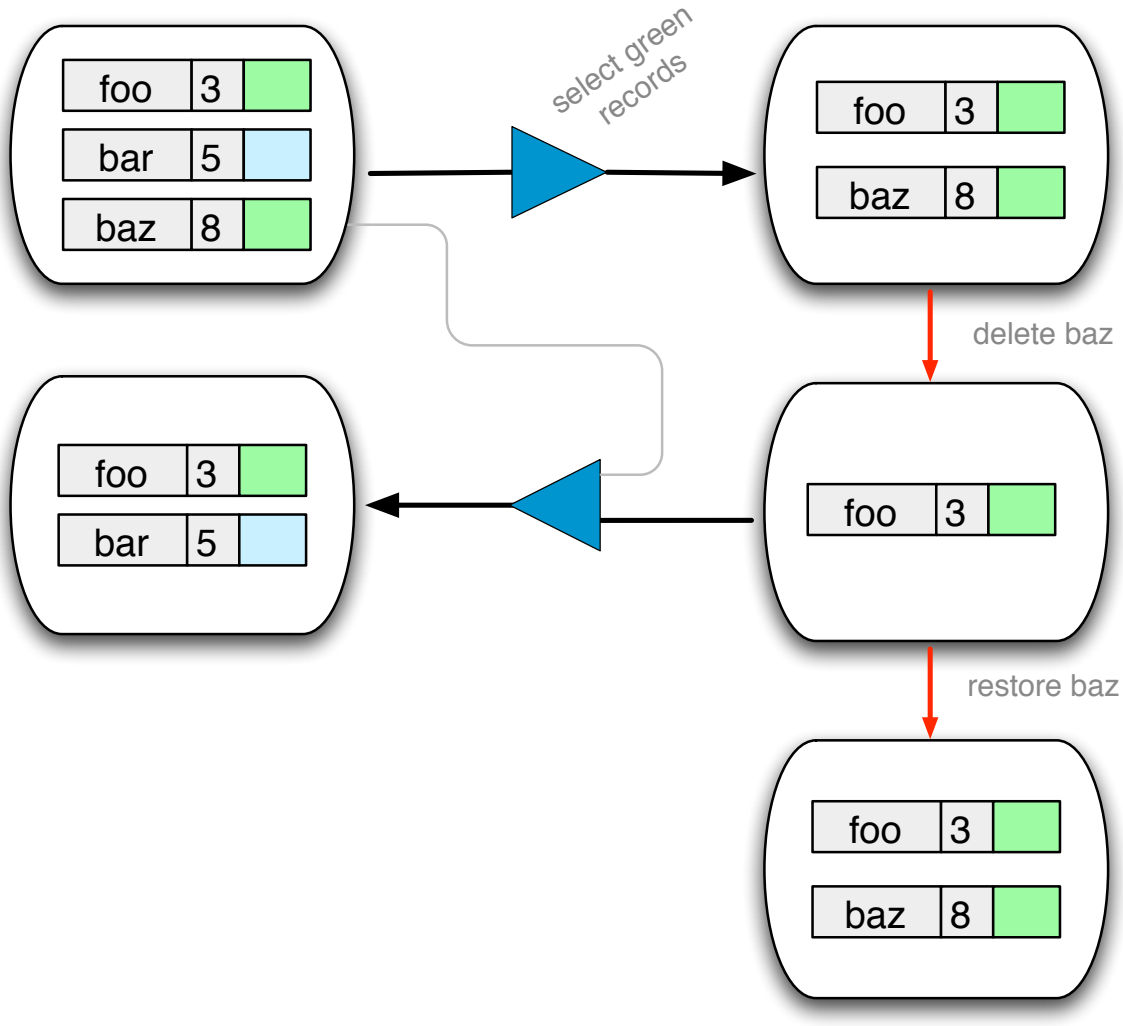
Another Example



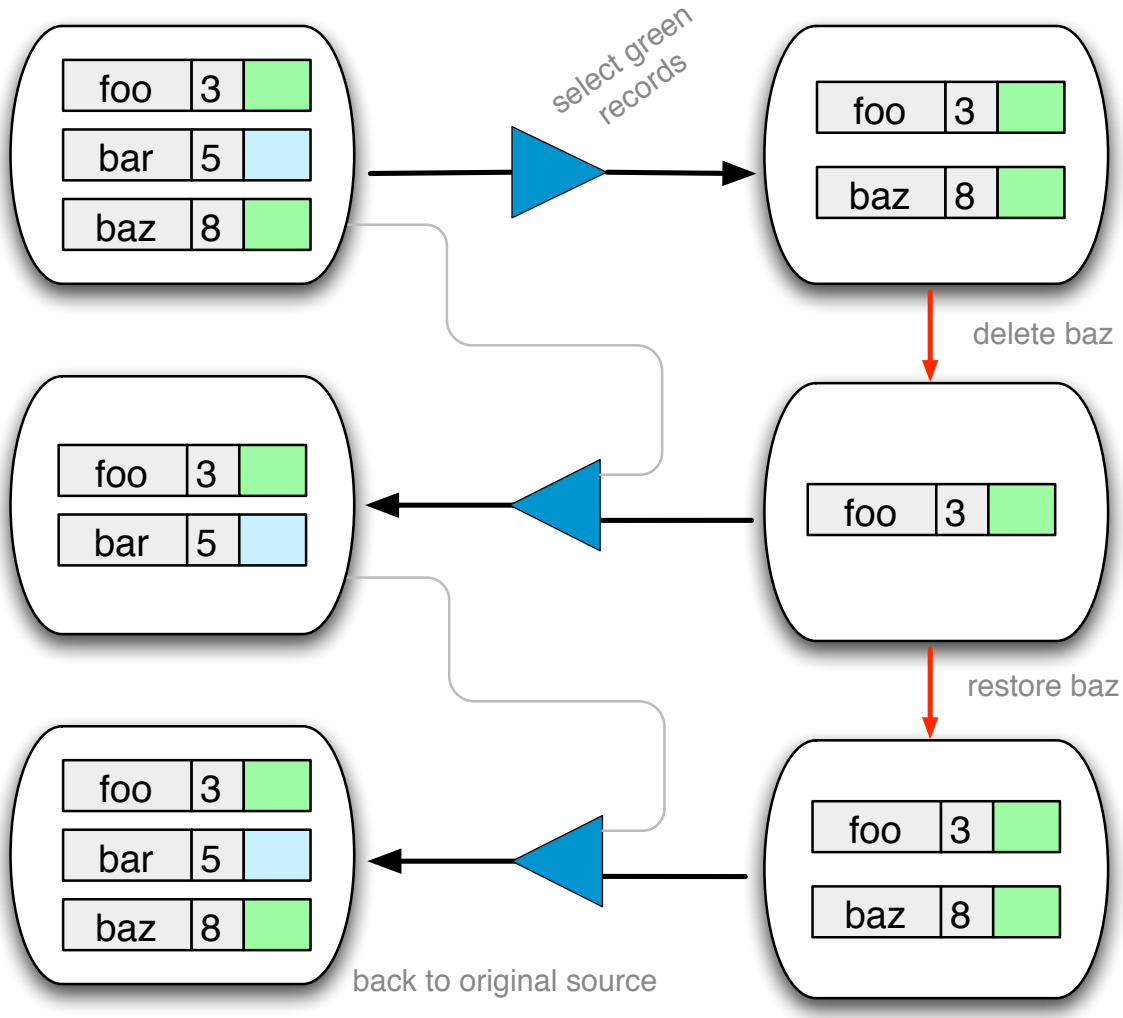
Another Example



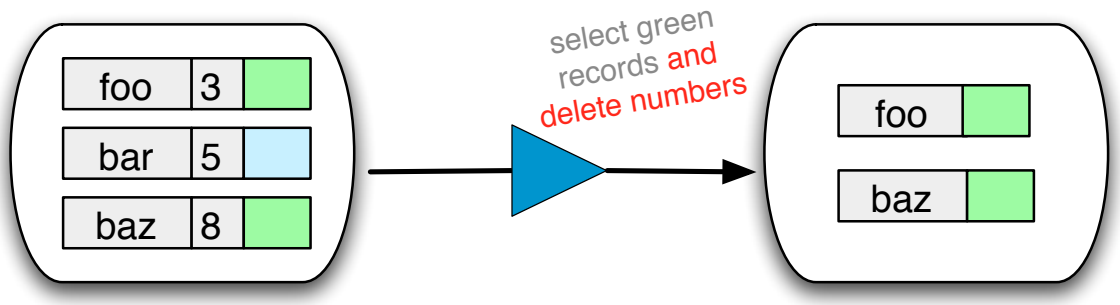
Another Example



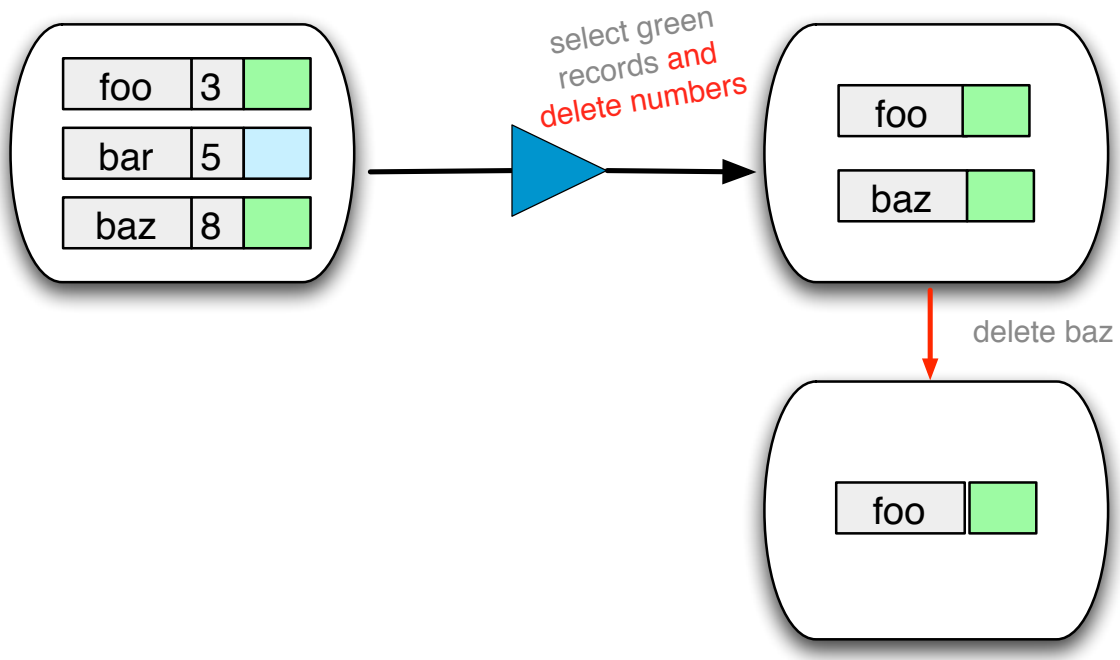
Another Example



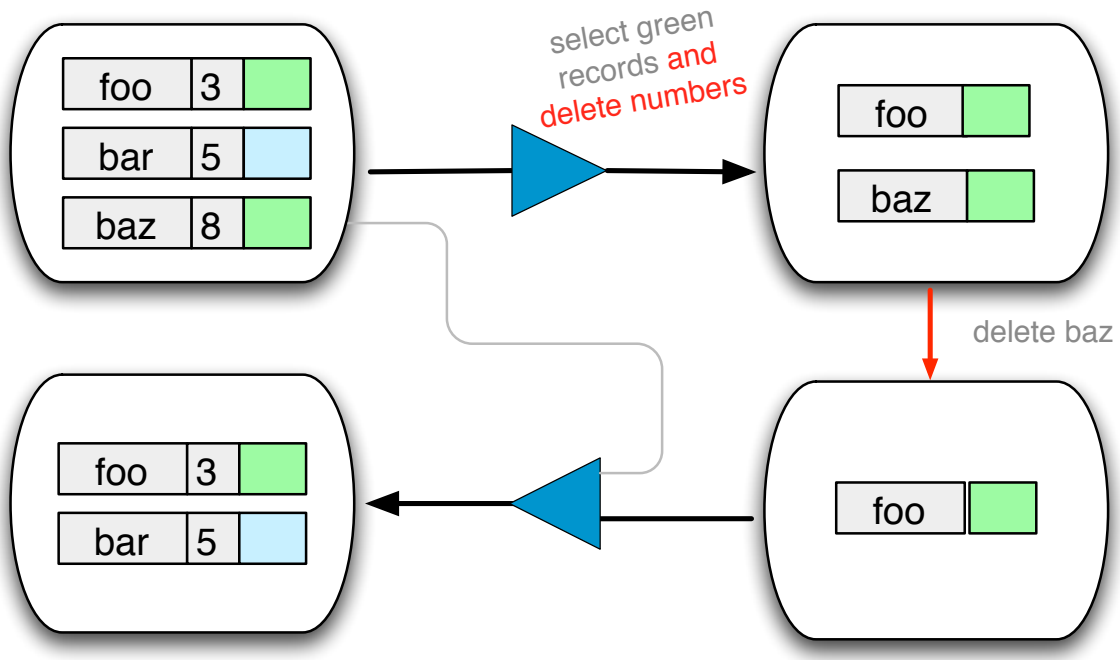
Yet Another Example



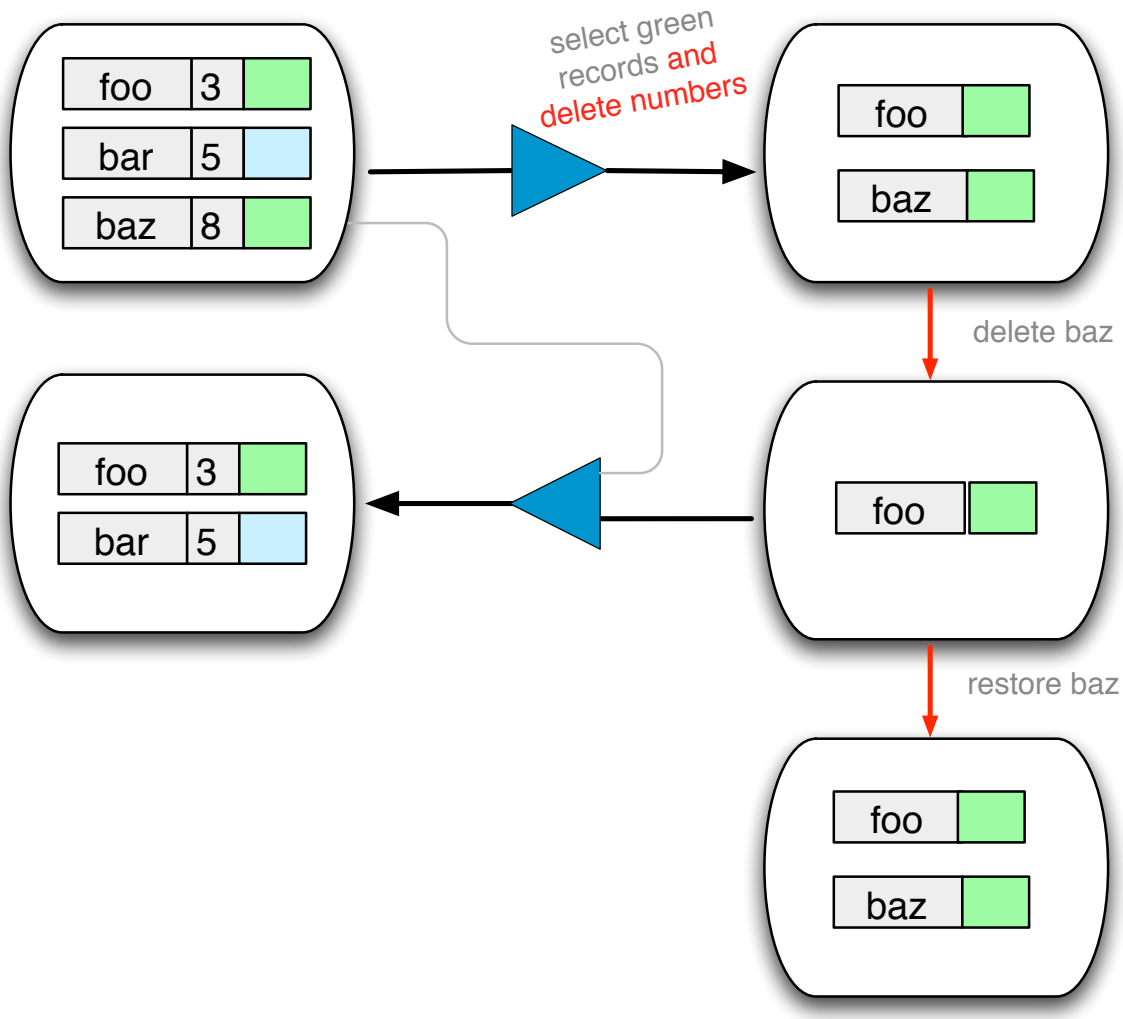
Yet Another Example



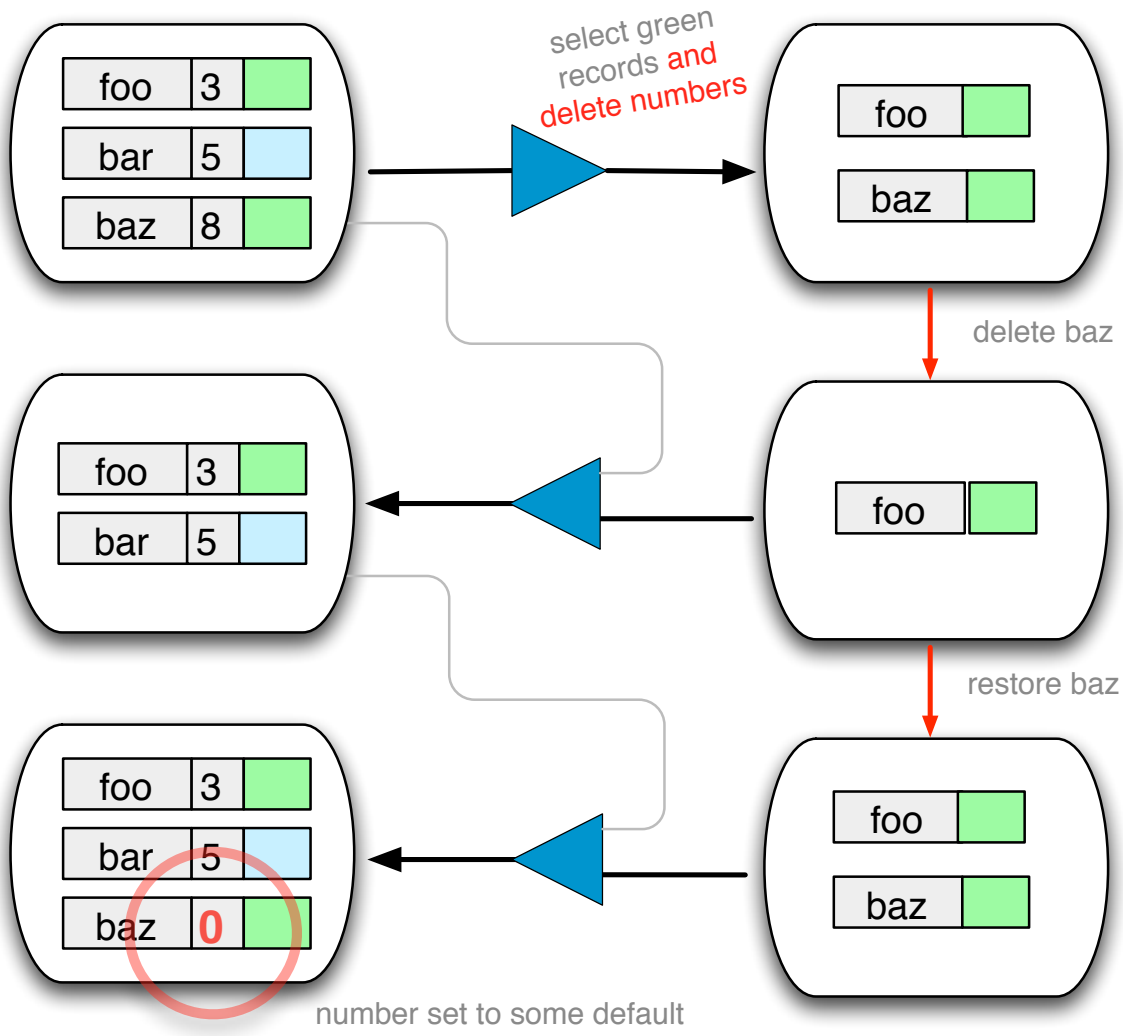
Yet Another Example



Yet Another Example



Yet Another Example



Well-Behaved Lenses

A well-behaved lens l mapping between a set S of sources and V of view is a pair of total functions

$$\begin{aligned} l.\mathbf{get} &\in S \to V \\ l.\mathbf{put} &\in V \to S \to S \end{aligned}$$

obeying "round-tripping" laws

$$l.\mathbf{get}\ (l.\mathbf{put}\ v\ s) = v \qquad \text{(PutGet)}$$

$$l.\mathbf{put}\ (l.\mathbf{get}\ s)\ s = s \qquad \text{(GetPut)}$$

Well-Behaved Lenses

A well-behaved lens l mapping between a set S of sources and V of view is a pair of total functions

$$\begin{aligned} l.\mathbf{get} &\in S \to V \\ l.\mathbf{put} &\in V \to S \to S \end{aligned}$$

obeying "round-tripping" laws

$$l.\mathbf{get}\ (l.\mathbf{put}\ v\ s) = v \qquad \text{(PutGet)}$$

$$l.\mathbf{put}\ (l.\mathbf{get}\ s)\ s = s \qquad \text{(GetPut)}$$

A very well-behaved lens l also obeys

$$\mathbf{put}\ v_2\ (\mathbf{put}\ v_1\ s) \quad = \quad \mathbf{put}\ v_2\ s \qquad \text{(PutPut)}$$

Related Frameworks

Databases: *many* related ideas

- [Dayal, Bernstein '82] "exact translation"
- [Bancilhon, Spryatos '81] "constant complement"
- [Gottlob, Paolini, Zicari '88] "dynamic views"

User Interfaces:

- [Meertens '98] "constraint maintainers"
- [Greenberg '07] DOM trees

Category Theory:

- [O'Connor '10] "co-algebras to monads"
- [Johnson '11] "algebras to co-monads"

See [Foster *et. al* TOPLAS '07] for a survey...

Related Languages

Harmony Group @ Penn

- [Hoffman *et al.* POPL '10] — symmetric version
- [Foster *et al.* TOPLAS '07] — trees
- [Bohannon *et al.* PODS '06] — relations
- [Foster *et al.* JCSS '07] — data synchronization

Bidirectional languages

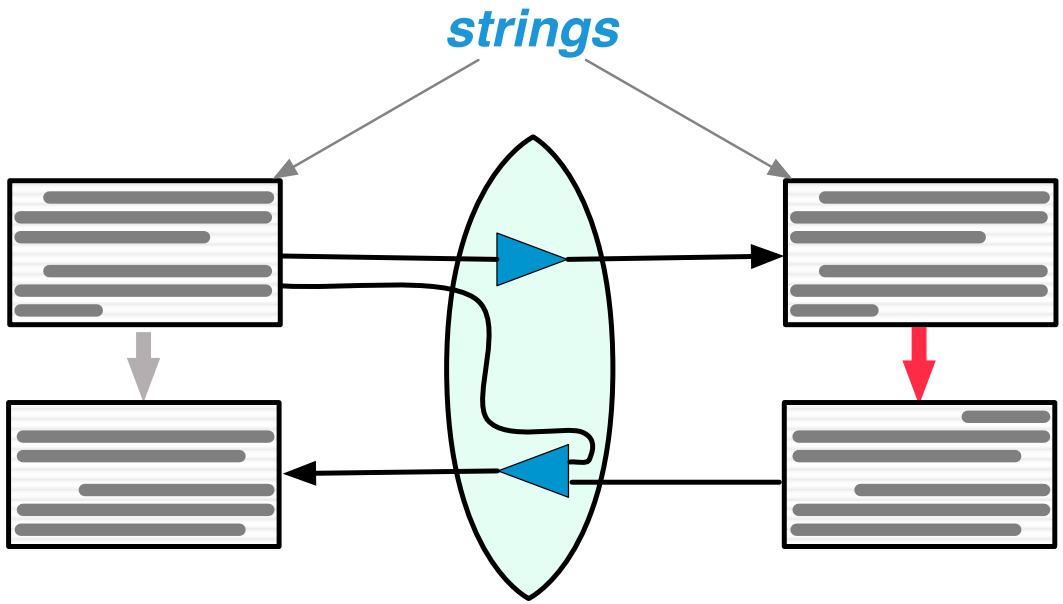
- [PSD @ Tokyo] — "bidirectionalization", structure editors
- [Gibbons, Wang @ Oxford] — Wadler's views
- [Voïgtlaender '09] — bidirectionalization "for free"
- [Stevens '07] — lenses for model transformations
- [GSD @ Waterloo] — synchronizing software models
- [PADS Project @ AT&T] — picklers and unpicklers
- [Braband, Møller, Schwartzbach '05] — XSugar

[Bohannon, Foster, Pierce, Pilkiewicz, Schmitt POPL '08]

String Lenses

"The art of progress is
to preserve order amid change
and to preserve change amid order."
—A N Whitehead

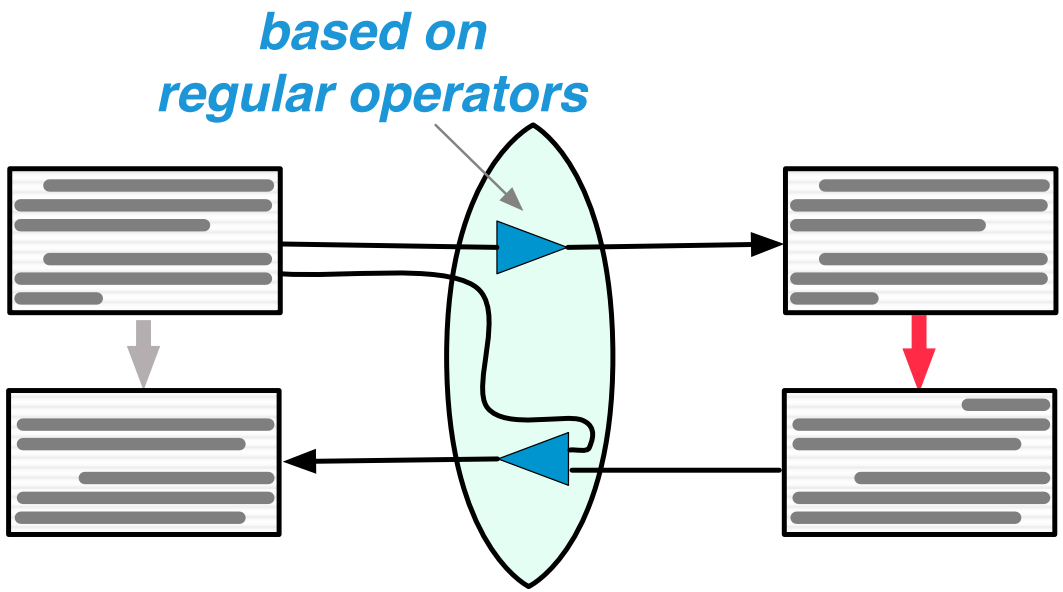
Data Model



Why strings?

1. Simple setting $\longrightarrow$ exposes fundamental issues
2. There's a lot of string data in the world
3. Programmers are already comfortable with regular operators (union, concatenation, and Kleene star)

Computation Model

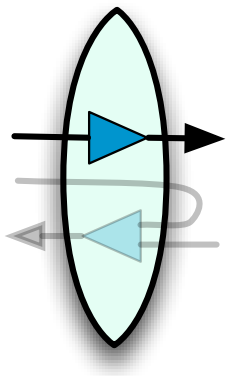


Why strings?

1. Simple setting $\longrightarrow$ exposes fundamental issues
2. There's a lot of string data in the world
3. Programmers are already comfortable with regular operators (union, concatenation, and Kleene star)

Example: Redacting Lens (Get)

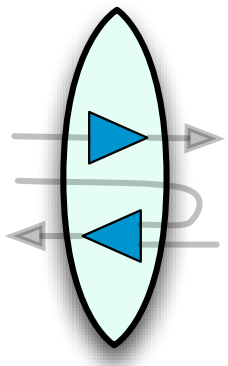
```
*08:30 Coffee with Sara (Gimme!)
 15:30 PLD (Upson 5126)
*19:00 Workout (Noyes)
```



```
08:30 BUSY
15:30 PLD
19:00 BUSY
```

Example: Redacting Lens (Update)

```
*08:30 Coffee with Sara (Gimme!)
 15:30 PLD (Upson 5126)
*19:00 Workout (Noyes)
```



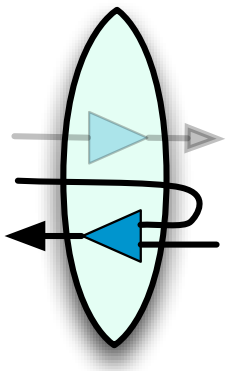
```
08:30 BUSY
15:30 PLClu
19:00 BUSY
```

```
08:30 BUSY
15:30 PLDG
19:00 BUSY
21:00 Dinner
```

Example: Redacting Lens (Put)

```
*08:30 Coffee with Sara (Gimme!)
 15:30 PLD (Upson 5126)
*19:00 Workout (Noyes)
```

```
*08:30 Coffee with Sara (Gimme!)
 12:15 PLDG (Upson 5126)
*19:00 Workout (Noyes)
 21:00 Dinner (Unknown)
```



```
08:30 BUSY
15:30 PLD
19:00 BUSY
```



```
08:30 BUSY
15:30 PLDG
19:00 BUSY
21:00 Dinner
```

Example: Redacting Lens (Definition)

```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN

(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = event*
```

Example: Redacting Lens (Definition)

```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN

(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = event*
```

Example: Redacting Lens (Definition)

```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN

(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = event*
```

Example: Redacting Lens (Definition)

```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN

(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = event*
```

Example: Redacting Lens (Definition)

```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN

(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = event*
```

String Lens Type System

Based on regular expression types...

$$\frac{}{\mathit{copy}\ E : [\![E]\!] \Longleftrightarrow [\![E]\!]} \qquad \frac{}{E \leftrightarrow d : [\![E]\!] \Longleftrightarrow \{d\}}$$

$$\frac{l : S \Longleftrightarrow V \qquad d \in [\![S]\!]}{\mathit{default}\ l\ d : S \Longleftrightarrow V} \qquad \frac{\begin{array}{ll} l_1 : S_1 \Longleftrightarrow V_1 & S_1 \mathbin{.^!} S_2 \\ l_2 : S_2 \Longleftrightarrow V_2 & V_1 \mathbin{.^!} V_2 \end{array}}{(l_1 \cdot l_2) : S_1 \cdot S_2 \Longleftrightarrow V_1 \cdot V_2}$$

$$\frac{\begin{array}{ll} l_1 : S_1 \Longleftrightarrow V_1 & S_1 \cap S_2 = \emptyset \\ l_2 : S_2 \Longleftrightarrow V_2 & \end{array}}{(l_1 \mid l_2) : S_1 \cup S_2 \Longleftrightarrow V_1 \cup V_2} \qquad \frac{l : S \Longleftrightarrow V \qquad S^{!*} \qquad V^{!*}}{l^* : S^* \Longleftrightarrow V^*}$$

$S_1 \mathbin{.^!} S_2$ (or $S^{!*}$) means that the concatenation (or iteration) is unambiguous.

String Lens Type System

Based on regular expression types...

$$\frac{}{\mathit{copy}\ E : [\![E]\!] \Longleftrightarrow [\![E]\!]} \qquad \frac{}{E \leftrightarrow d : [\![E]\!] \Longleftrightarrow \{d\}}$$

$$\frac{l : S \Longleftrightarrow V \qquad d \in [\![S]\!]}{\mathit{default}\ l\ d : S \Longleftrightarrow V} \qquad \frac{\begin{array}{cc} l_1 : S_1 \Longleftrightarrow V_1 & S_1 \mathbin{.^!} S_2 \\ l_2 : S_2 \Longleftrightarrow V_2 & V_1 \mathbin{.^!} V_2 \end{array}}{(l_1 \cdot l_2) : S_1 \cdot S_2 \Longleftrightarrow V_1 \cdot V_2}$$

$$\frac{\begin{array}{cc} l_1 : S_1 \Longleftrightarrow V_1 & S_1 \cap S_2 = \emptyset \\ l_2 : S_2 \Longleftrightarrow V_2 & \end{array}}{(l_1 \mid l_2) : S_1 \cup S_2 \Longleftrightarrow V_1 \cup V_2} \qquad \frac{l : S \Longleftrightarrow V \qquad S^{!*} \qquad V^{!*}}{l^* : S^* \Longleftrightarrow V^*}$$

$S_1 \mathbin{.^!} S_2$ (or $S^{!*}$) means that the concatenation (or iteration) is unambiguous.

Theorem

If $l : S \Longleftrightarrow V$ then l is a well-behaved lens.

Comparison: Separate Functions

```
module B = Buffer
module R = Str
module L = List
module U = Unix
let rx = R.regexp
let sch = R.search_forward

(* helpers *)
let error s =
  print_string s;
  exit 1

let exec s =
  let buf = B.create 17 in
  let p = U.open_process_in s in
  (try
     while true do B.add_char buf (input_char p) done
  with End_of_file -> ());
  ignore (U.close_process_in p);
  B.contents buf

(* regexps *)
let begin_event = rx "BEGIN:VEVENT"
let text = "[^\n]*"
let esctext = "\([^\n\\)]\|\\\\\|\\(\|\\)\)*"
let date = "[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]T[0-9][0-9][0-9][0-9][0-9][0-9]Z"
let codes = [("\\", "\\\\"); ("(","\\("); (")","\\)")]

let today () = exec "date +%Y%m%d | tr -d '\n'"
let now () = exec "date +%H%M%S | tr -d '\n'"
let uid () = exec "cat /dev/urandom|uuencode -m -|tail +2|tr -dc 'a-zA-Z0-9'|head -c25"

(* dictionary lens helpers *)
let escape codes s =
  L.fold_right
    (fun (p,r) s -> R.global_replace (rx p) r s)
    codes
    s

let unescape codes s =
  L.fold_left
    (fun s (p,r) -> R.global_replace (rx r) p s)
    s codes

let distance s1 s2 =
  let m,n = String.length s1, String.length s2 in
  let aodd,aeven = Array.make (succ n) 0,Array.make (succ n) 0 in
    for j=0 to n do aeven.(j) <- j done;
    for i=1 to m do
      let apredi = if i mod 2 = 0 then aodd else aeven in
      let ai = if i mod 2 = 0 then aeven else aodd in
        ai.(0) <- i;
        for j = 1 to n do
          let x =
            if String.get s1 (pred i) = String.get s2 (pred j) then 0 else 1 in
          ai.(j) <-
            (min (apredi.(j) + 1) (min (ai.(j-1) + 1) (apredi.(j-1) + x)));
        done
    done;
    if m mod 2 = 0 then aeven.(n) else aodd.(n)

let rec lookup k l =
  let l' = L.map (fun (ki,ci) -> (distance ki k,ki,ci)) l in
  let rec aux acc l = match acc,l with
      None,[] -> None
      Some(_,ki,ci),[] -> Some(ki,ci)
      Some(di,ki,ci),(dj,kj,cj)::rest when dj < di -> aux (Some(dj,kj,cj)) rest
      None,(dj,kj,cj)::rest -> aux (Some(dj,kj,cj)) rest
      _,_::rest -> aux acc rest in
  aux None l'

let remove k l =
  let rec loop acc = function
      [] -> L.rev_append acc []
      h::t when k = fst h -> L.rev_append acc t
      h::t -> loop (h::acc) t in
  loop [] l

let wrap b =
  let b_len = String.length b in
  let buf,line_buf,aux_buf = B.create b_len, B.create 75, B.create 75 in
  let do_it s b =
    if B.length line_buf <> 0 then B.add_string buf s;
    B.add_buffer buf b;
    B.reset b in
  let rec loop i =
    let sum =
      let aux_len = B.length aux_buf in
      let line_len = let n = B.length line_buf in if n=0 then n else succ n in
      aux_len + line_len in
    if sum > 75 then do_it "\n" line_buf;
    if i = b_len then (do_it " " aux_buf; do_it "\n" line_buf)
    else
      (if b.[i] = ' ' then do_it " " aux_buf
       else B.add_char aux_buf b.[i];
       loop (succ i)) in
  loop 0;
  B.contents buf

let unwrap c = R.global_replace (rx "\n") (String.make 1 ' ') c

let field tag r s =
  try
    ignore(sch (rx (tag ^ ":" ^ "\\(" ^ r ^ "\\)\n")) s 0);
    R.matched_group 1 s
  with Not_found -> error ("Couldn't find " ^ tag ^ " in " ^ s ^ "\n")

let times s = String.sub s 9 2, String.sub s 11 2
```

Helpers

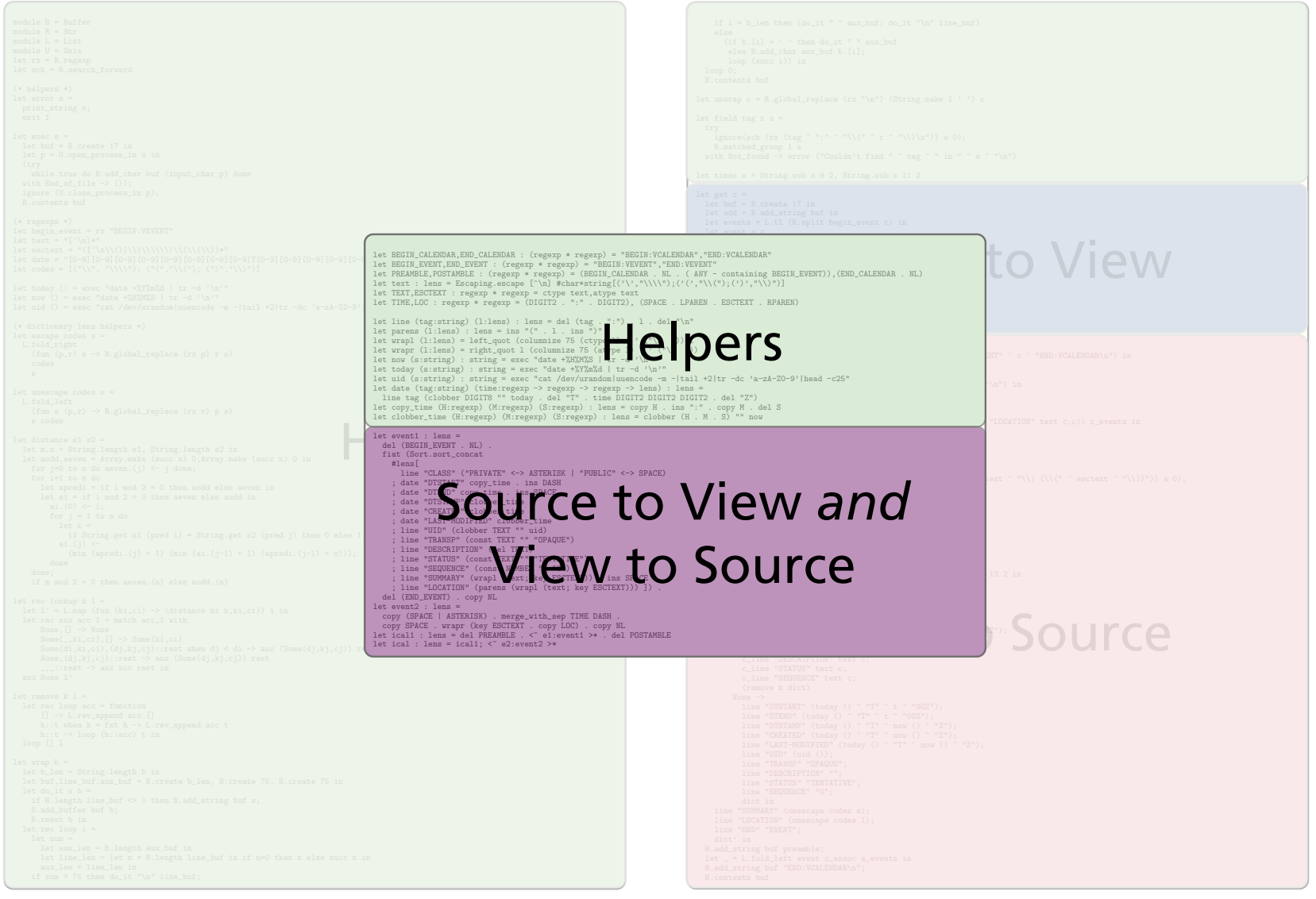
```
let get c =
  let buf = B.create 17 in
  let add = B.add_string buf in
  let events = L.tl (R.split begin_event c) in
  let event e =
    let p = field "CLASS" "PUBLIC\|PRIVATE" e in
    let t1,t2 = times (field "DTSTART" date e) in
    let s = escape codes (field "SUMMARY" text e) in
    let l = escape codes (field "LOCATION" text e) in
    if p = "PUBLIC" then add " " else add "*";
    add (t1 ^ ":" ^ t2 ^ " ");
    add (wrap s ^ " (" ^ l ^ ")");
    add "\n" in
  L.iter event events;
  B.contents buf
```

Source to View

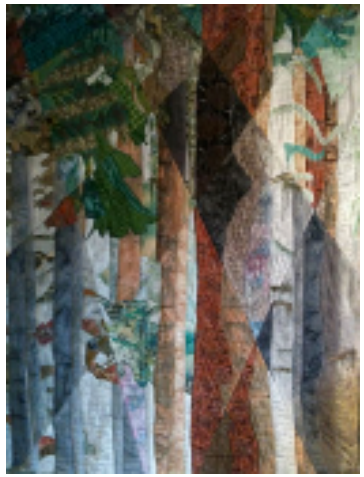
```
let put a c =
  let is_noop a c = a ^ "\n" = get ("BEGIN:VCALENDAR\nBEGIN:VEVENT" ^ c ^ "END:VCALENDAR\n") in
  let buf = B.create 17 in
  let line s v = B.add_string buf (s ^ ":" ^ wrap v ^ "\n") in
  let c_line s v c = B.add_string buf (s ^ ":" ^ field s v c ^ "\n") in
  let preamble,c_events = match R.split begin_event c with
      [] -> error "ill-formed"
    | h::t -> (h,t) in
  let c_assoc = L.map (fun c -> (field "SUMMARY" text c ^ field "LOCATION" text c,c)) c_events in
  let a_events =
    L.map
      (fun a ->
         let p = a.[0] = ' ' in
         let t = (String.sub a 1 2) ^ (String.sub a 4 2) in
         ignore (sch (rx ("[ *][0-9][0-9]:[0-9][0-9] \\(" ^ esctext ^ "\\) (\\(" ^ esctext ^ "\\))")) a 0);
         (p,t,R.matched_group 1 a,R.matched_group 2 a,a))
      (R.split (rx "\n") (unwrap a)) in
  let event dict (p,t,s,l,a) =
    line "BEGIN" "EVENT";
    if p then line "CLASS" "PUBLIC"
    else line "CLASS" "PRIVATE";
    let dict' = match lookup (s ^ l) dict with
        Some (k,c) ->
          let dtstart = field "DTSTART" date c in
          let ymd,s = String.sub dtstart 0 8,String.sub dtstart 13 2 in
          line "DTSTART" (ymd ^ "T" ^ t ^ s ^ "Z");
          c_line "DTEND" date c;
          c_line "DTSTAMP" date c;
          c_line "CREATED" date c;
          if is_noop a c then c_line "LAST-MODIFIED" date c
          else line "LAST-MODIFIED" (now () ^ "Z");
          c_line "UID" text c;
          c_line "TRANSP" text c;
          c_line "DESCRIPTION" text c;
          c_line "STATUS" text c;
          c_line "SEQUENCE" text c;
          (remove k dict)
      | None ->
          line "DTSTART" (today () ^ "T" ^ t ^ "00Z");
          line "DTEND" (today () ^ "T" ^ t ^ "00Z");
          line "DTSTAMP" (today () ^ "T" ^ now () ^ "Z");
          line "CREATED" (today () ^ "T" ^ now () ^ "Z");
          line "LAST-MODIFIED" (today () ^ "T" ^ now () ^ "Z");
          line "UID" (uid ());
          line "TRANSP" "OPAQUE";
          line "DESCRIPTION" "";
          line "STATUS" "TENTATIVE";
          line "SEQUENCE" "0";
          dict in
    line "SUMMARY" (unescape codes s);
    line "LOCATION" (unescape codes l);
    line "END" "EVENT";
    dict' in
  B.add_string buf preamble;
  let _ = L.fold_left event c_assoc a_events in
  B.add_string buf "END:VCALENDAR\n";
  B.contents buf
```

View to Source

Comparison: String Lens



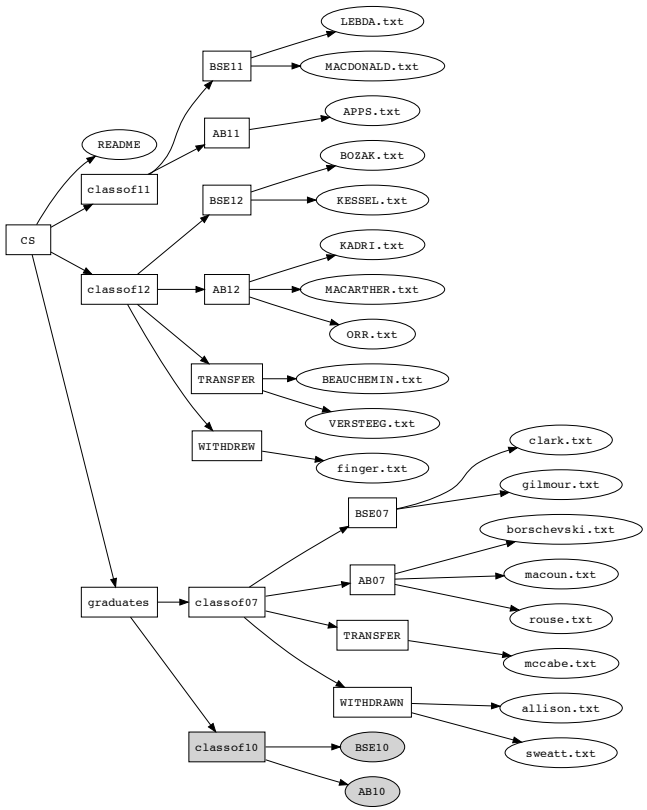
[Fisher, Foster, Walker, Zhu ICFP ’11]



Forest

Lenses for Filestores

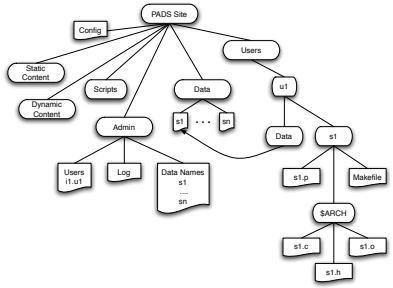
A filestore is a collection of directories, files, and symbolic links organized into a coherent data set.



Filestores are Pervasive

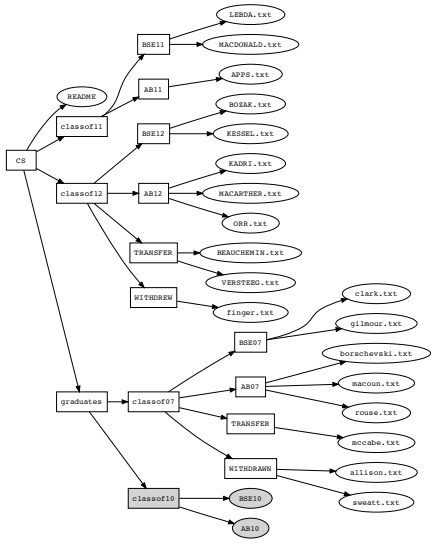
Monitoring:

- CoralCDN
- *Many* applications at AT&T



Science:

- Physics simulations
- Environmental data



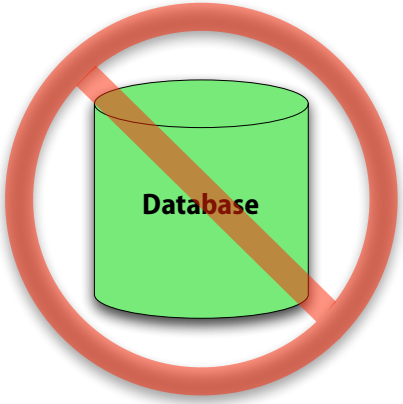
Ad hoc data:

- Princeton student records
- Linux standard base
- Websites, repositories, etc.

Filestores vs. Databases

Database Challenges

- Up-front costs
- Loading overhead
- Loss of control

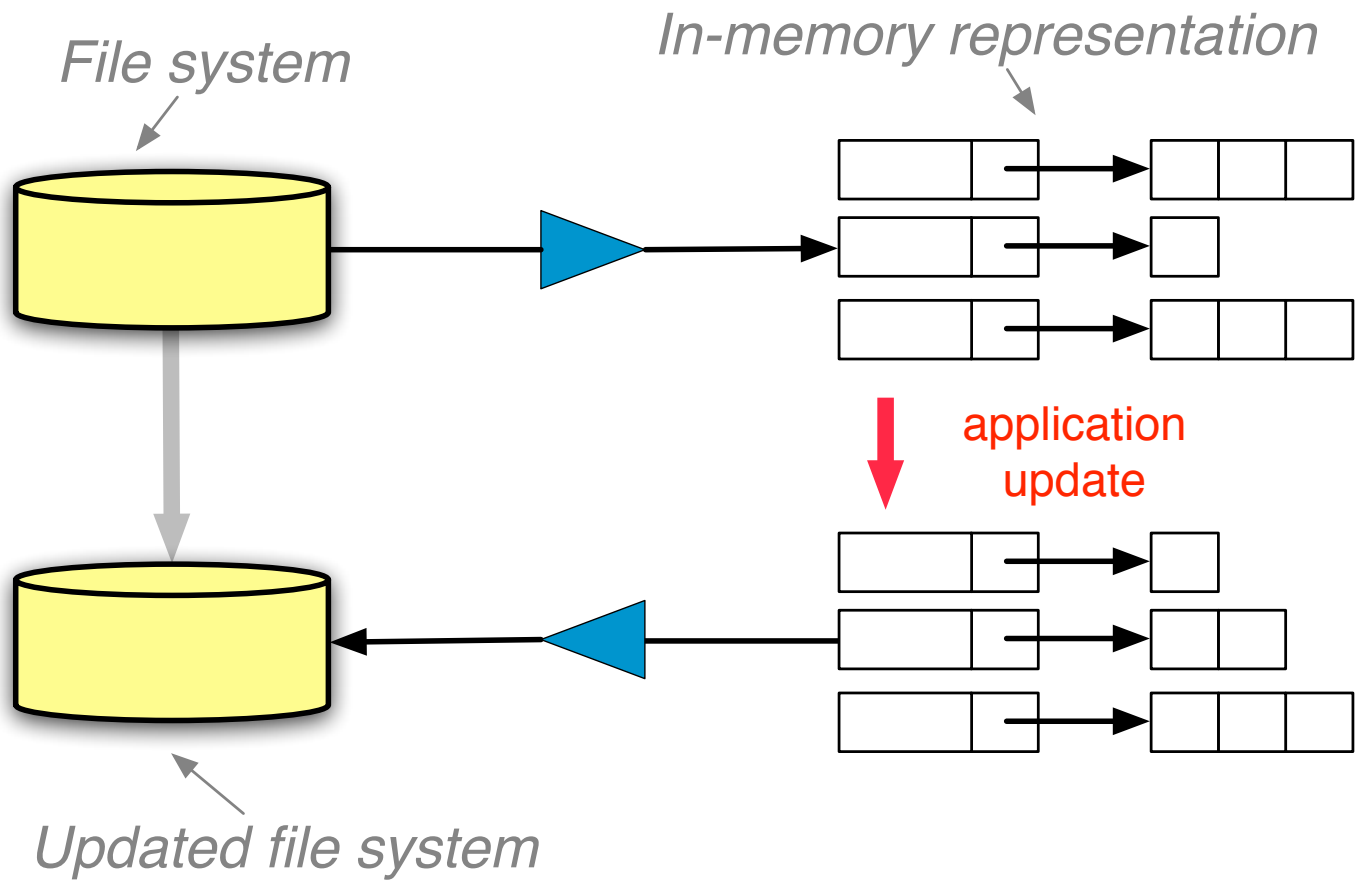


Filestore Challenges

- No documentation
- Must “roll own” tools
- No way to check for errors
- Difficult to maintain
- Large scale

Forest Solution

Idea: lenses for filestores!

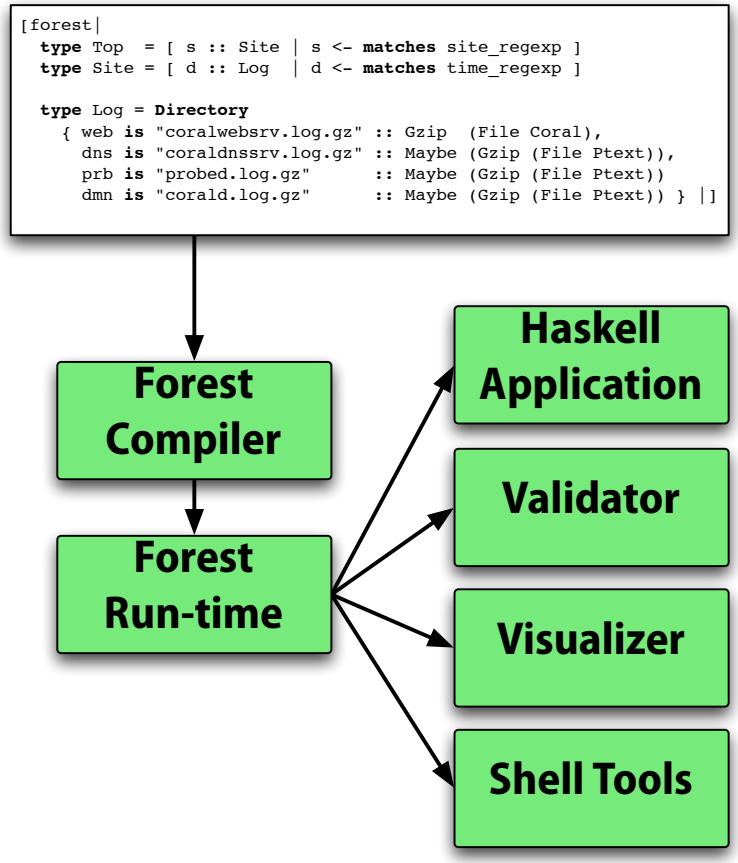


Forest Solution

Idea: lenses for filestores!

Generated Artifacts

- Representation type
- Metadata type
- Load function
- Store function
- Class instances for generic programming



Example: Universal Description

```
[forest|
  type Universal_d = Directory
   { ascii_files  is [ f :: Text
                       f <- matches (GL "*"),
                     <| get_kind  f_att == AsciiK       |> ],
     binary_files is [ b :: Binary
                       b <- matches (GL "*"),
                     <| get_kind  b_att == Binary       |> ]
     directories is [ d :: Universal_d
                       d <- matches (GL "*"),
                     <| get_kind  d_att == DirectoryK   |> ]
     symLinks    is [ s :: SymLink
                       s <- matches (GL "*"),
                     <| get_isSym s_att == True         |> ] }
 |]
```

Forest Properties

Theorem 1 (LoadStore)

$$\left[\begin{array}{l} \mathcal{E}; r; s \vdash \mathsf{load}\, F \rhd (v, d) \\ \wedge\ \mathcal{E}; r; s \vdash \mathsf{store}\,(F, v, d) \rhd (F', \phi) \end{array}\right] \Rightarrow (F = F') \ \wedge\ \phi(F')$$

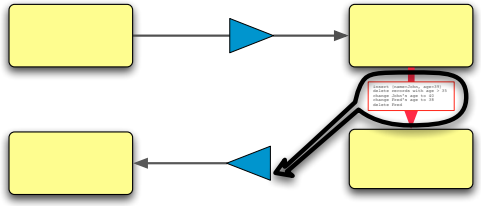
Theorem 2 (StoreLoad)

$$\left[\begin{array}{l} \mathcal{E}; r; s \vdash \mathsf{store}\,(F, v, d) \rhd (F', \phi') \\ \wedge\ \phi'(F') \\ \wedge\ \mathcal{E}; r; s \vdash \mathsf{load}\, F' \rhd (v', d') \end{array}\right] \Rightarrow (v', d') = (v, d).$$

Ongoing Work

Lenses: state-based → operation-based

- More efficient
- Increased precision
- Applications to audit
- Support for concurrent viewers



Frenetic: network programming language

- Automatic code partitioning
- Consistency abstractions
- Isolation/virtualization
- End-host integration

frenetic ≫

Thank You!

Collaborators: Aaron Bohannon, Kathleen Fisher, Michael Greenberg, Benjamin Pierce, Alexandre Pilkiewicz, Raghu Rajkumar, Alan Schmitt, David Walker, Norris Xu, and Kenny Zhu.

Want to play? Boomerang and Forest are available:

- Source code (under an open source license)
- Examples
- Research papers

`http://www.cs.cornell.edu/~jnfoster/`

Weakening the PutGet law

If we want to allow such behavior, we need to weaken PutGet. Here is one possibility:

$$\frac{\textbf{put}\ v\ s = s' \qquad \textbf{get}\ s' = v'}{\textbf{put}\ v'\ s = s'}$$

Intuition:

Propagating an update may have "side-effects", but only on the initial round-trip.

Similar idea in databases:

Propagating an update must have "minimal side-effects".