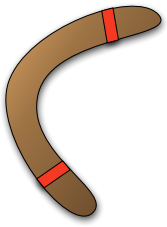
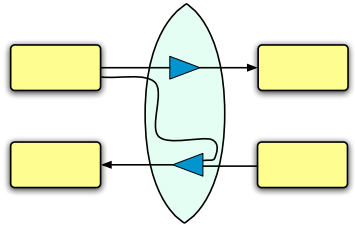
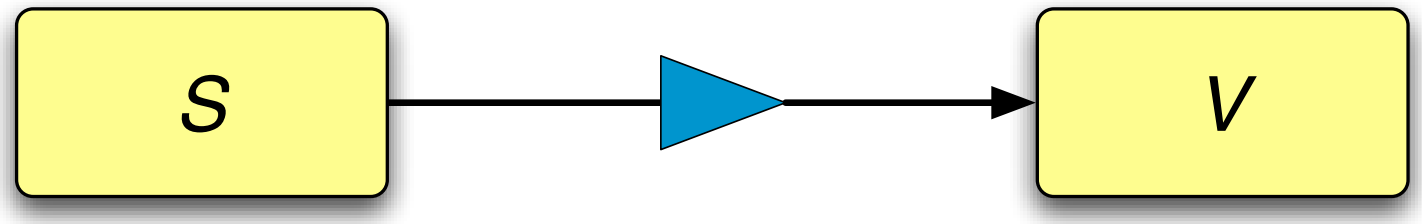
Bidirectional Programming Languages

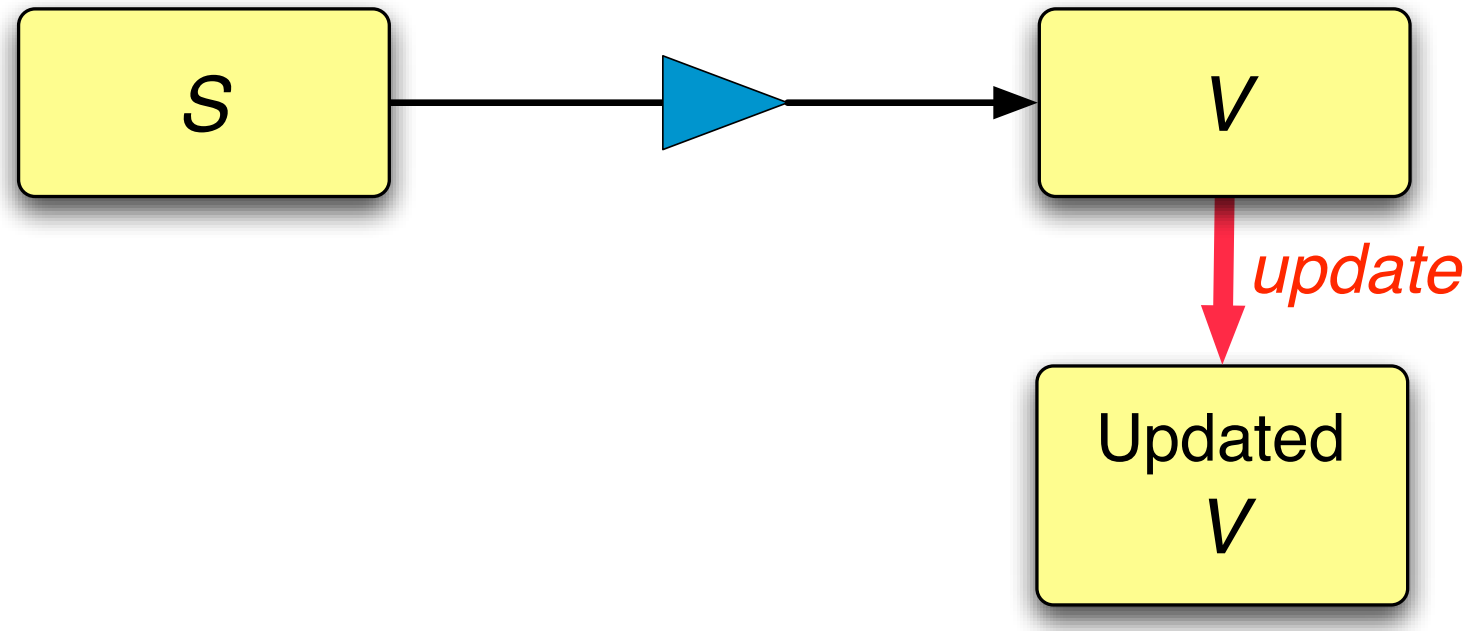
Nate Foster
University of Pennsylvania

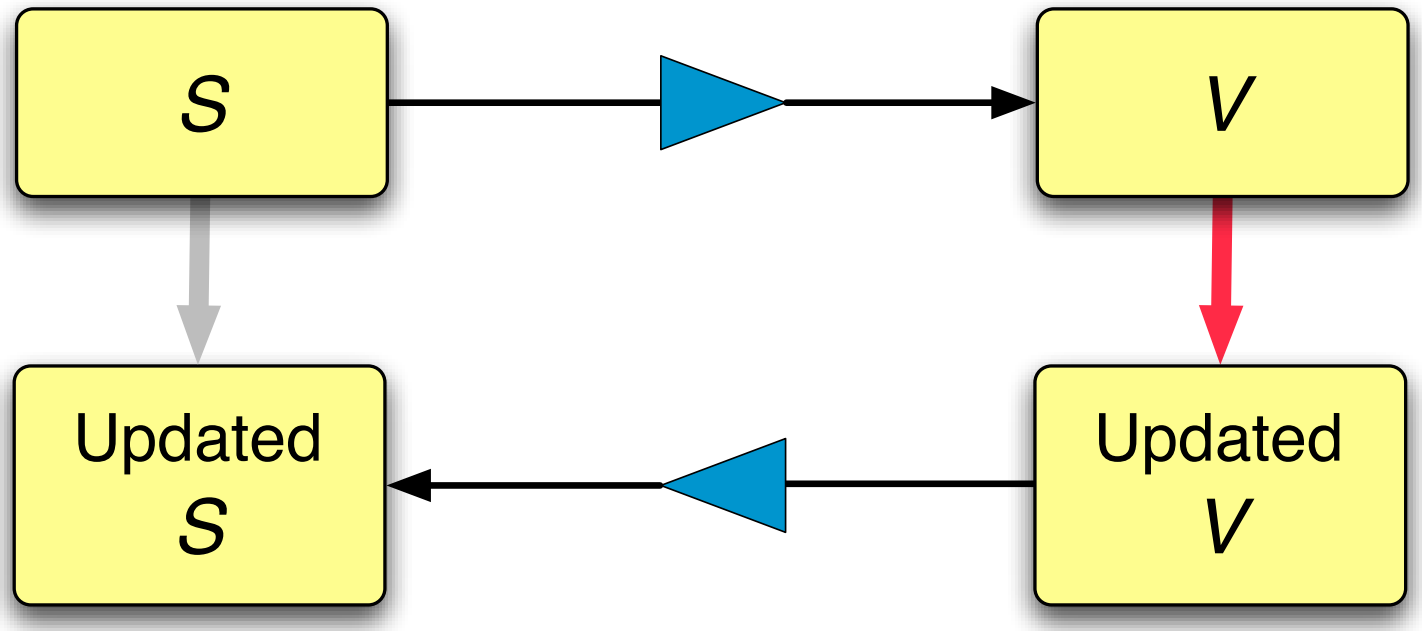
Dissertation Defense
11 September 2009





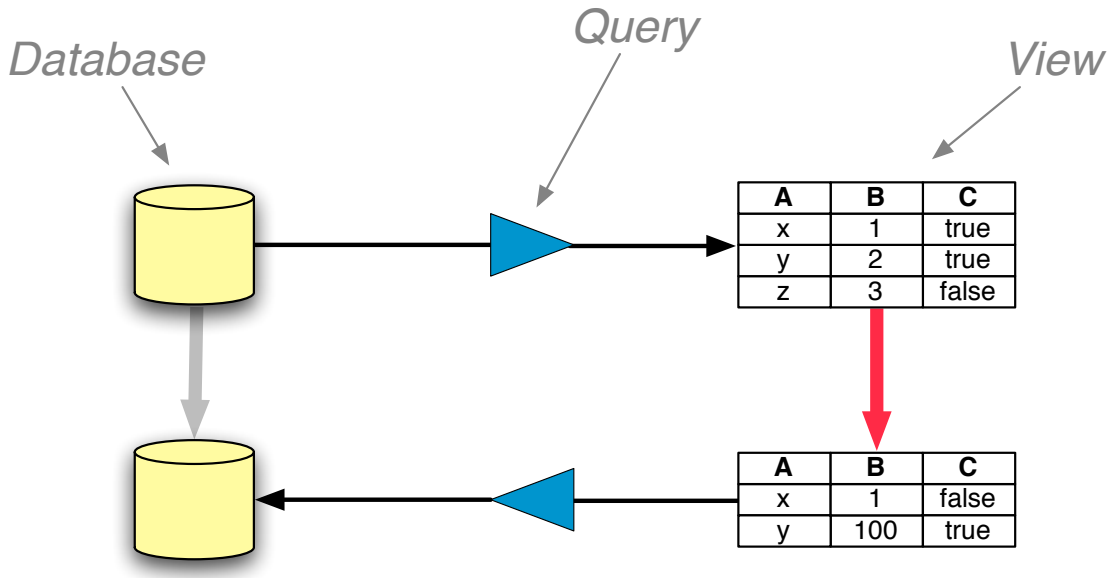

S
V


S
V
update
Updated
V


S
V
Updated
S
Updated
V

The View Update Problem

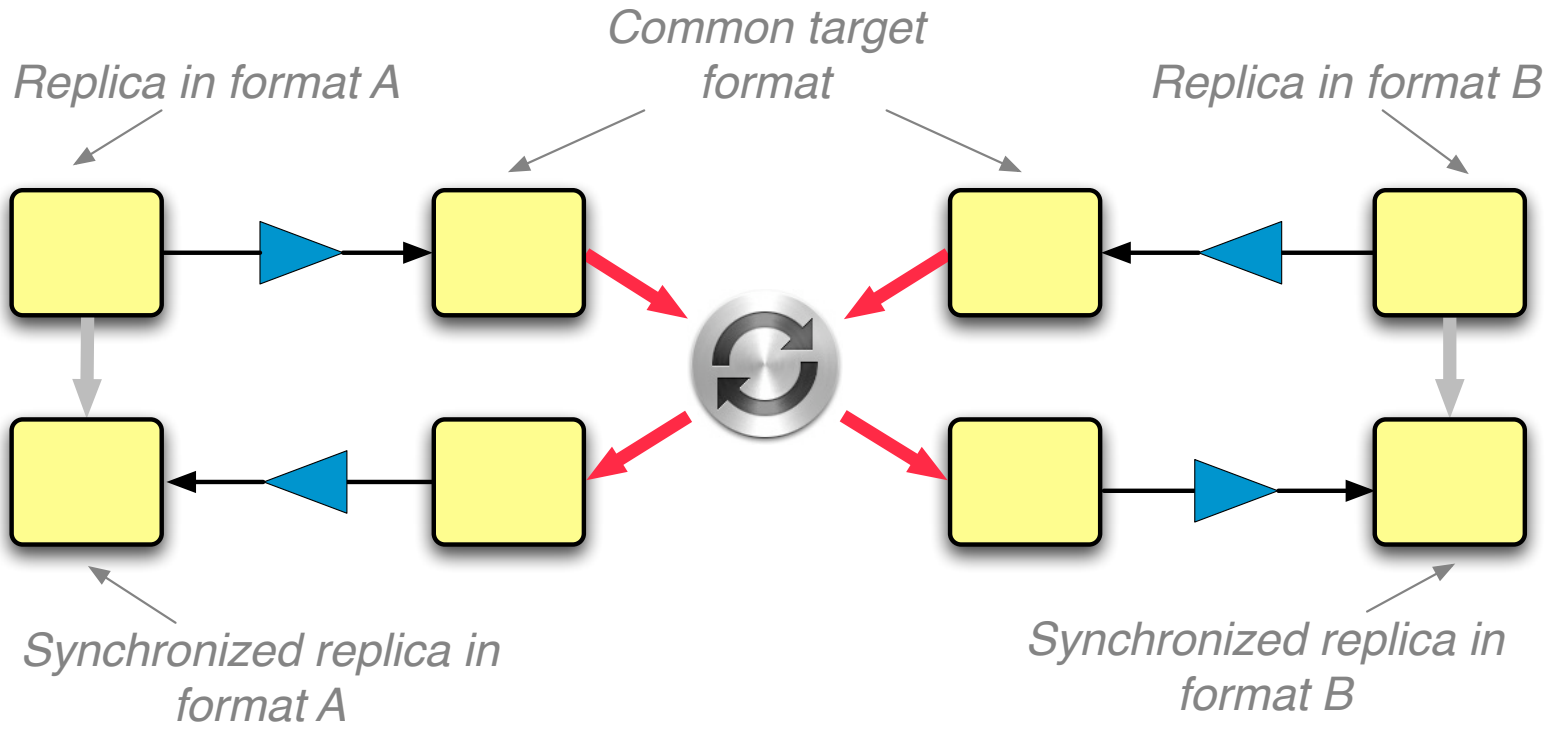
In databases, this is known as the view update problem.



[Bancilhon, Spryatos '81]

The View Update Problem In Practice

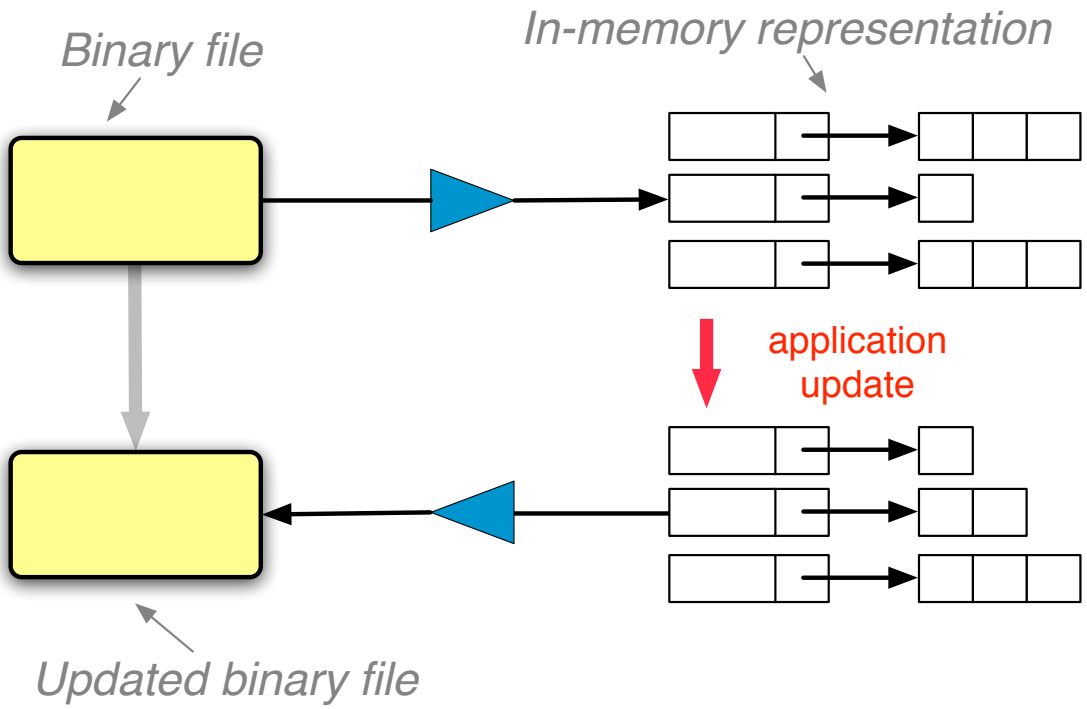
It also arises in data converters and synchronizers...



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

The View Update Problem In Practice

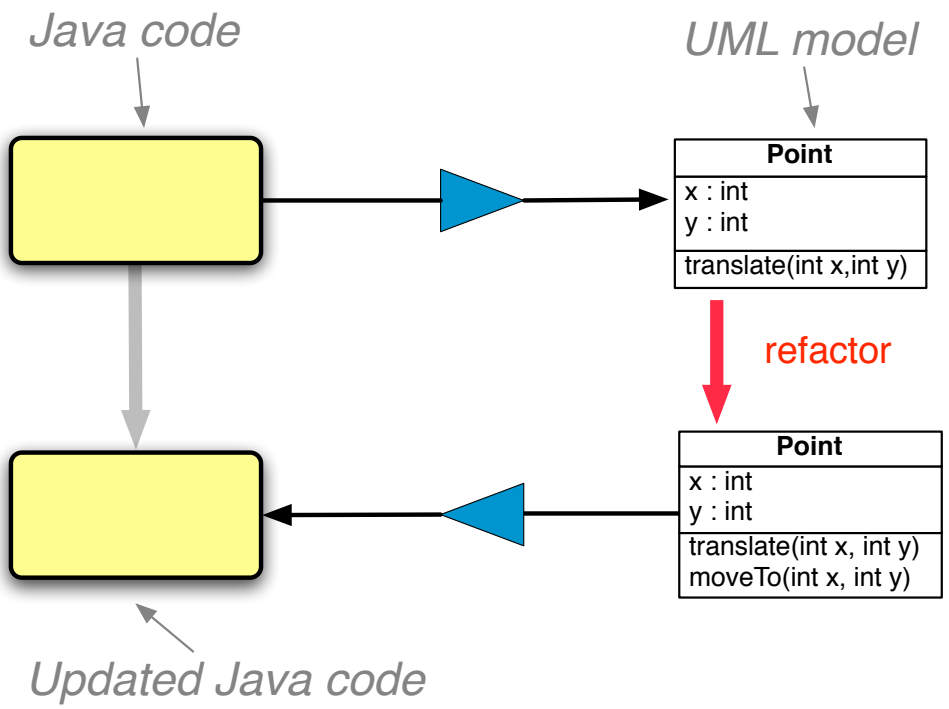
...in picklers and unpicklers...



[Fisher, Gruber '05]— PADS

The View Update Problem In Practice

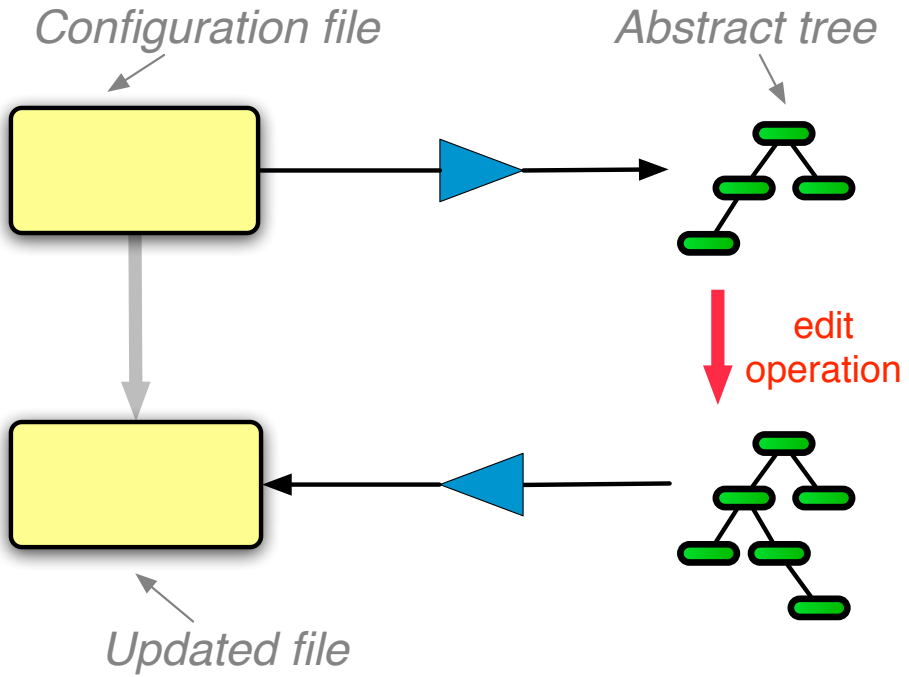
...in model-driven software development...



[Stevens '07]— bidirectional model transformations

The View Update Problem In Practice

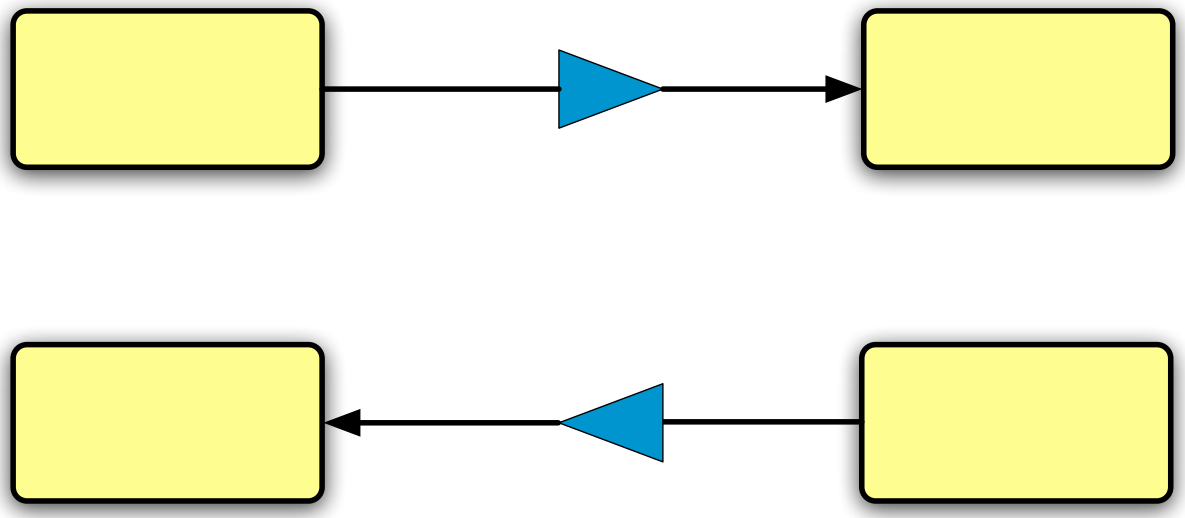
...in tools for managing operating system configurations...



[Lutterkort '08]— Augeas

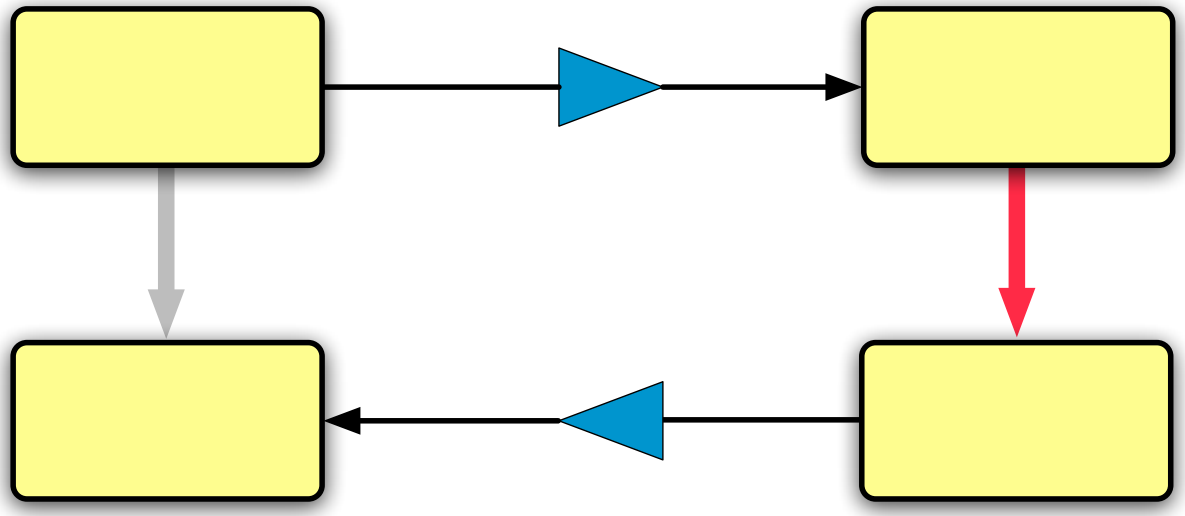
Problem

How do we write these bidirectional transformations?



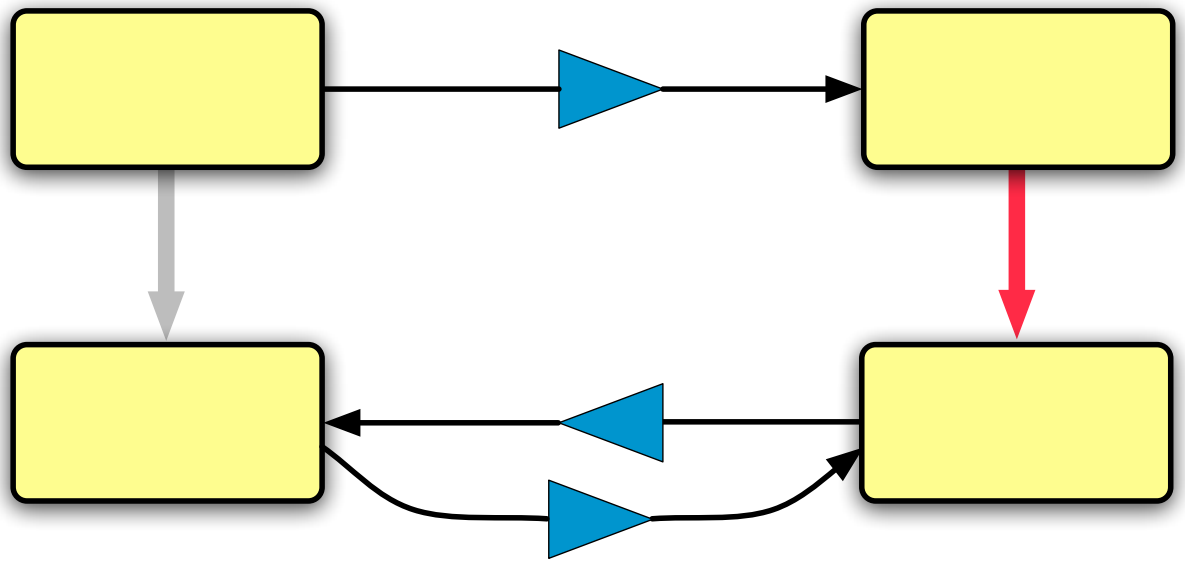
Problem: Why is it hard?

We want updates to the view to be translated "exactly" ...



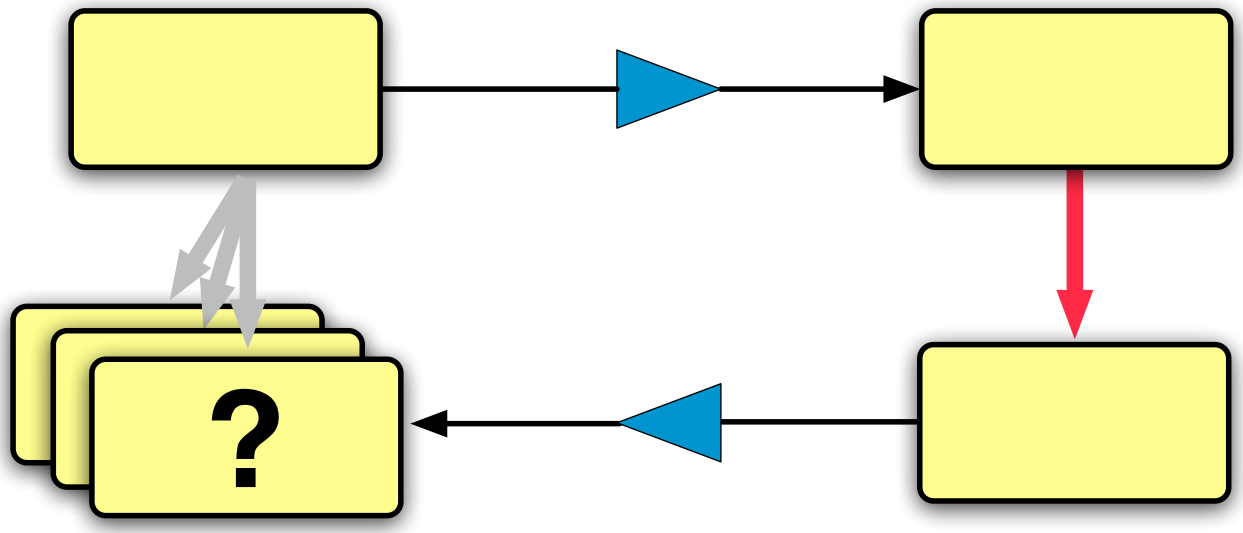
Problem: Why is it hard?

We want updates to the view to be translated "exactly" ...



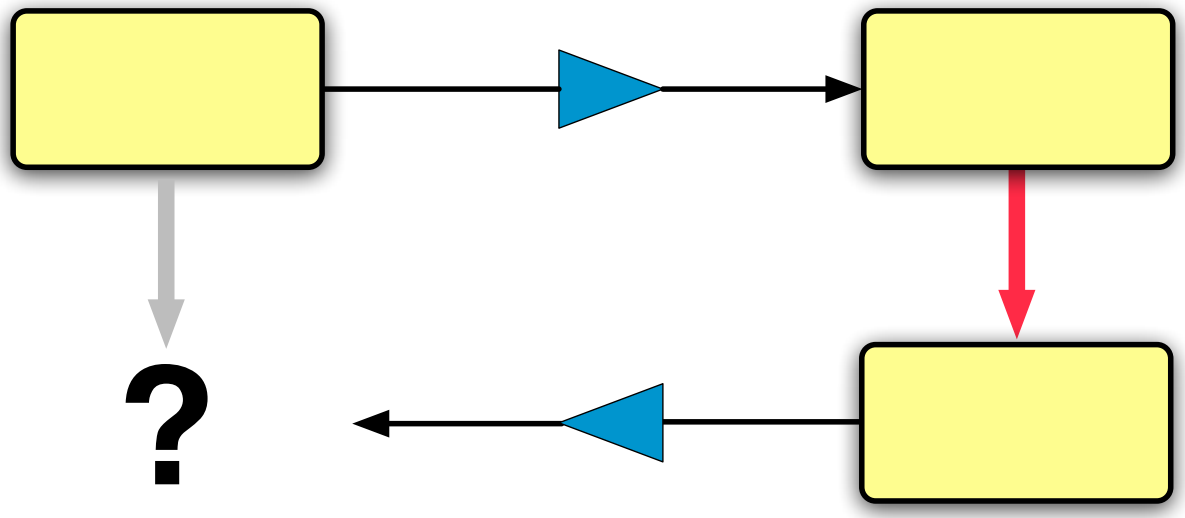
Problem: Why is it hard?

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



Problem: Why is it hard?

...while others have *none*!

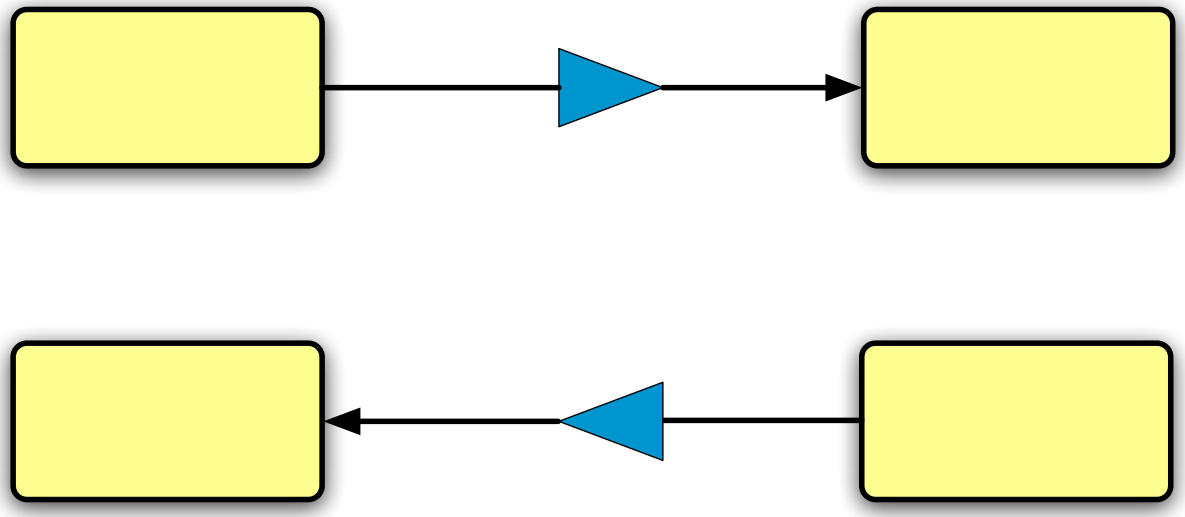


We *can* implement updatable views in C...

or Java...

or C++...

Possible Approaches

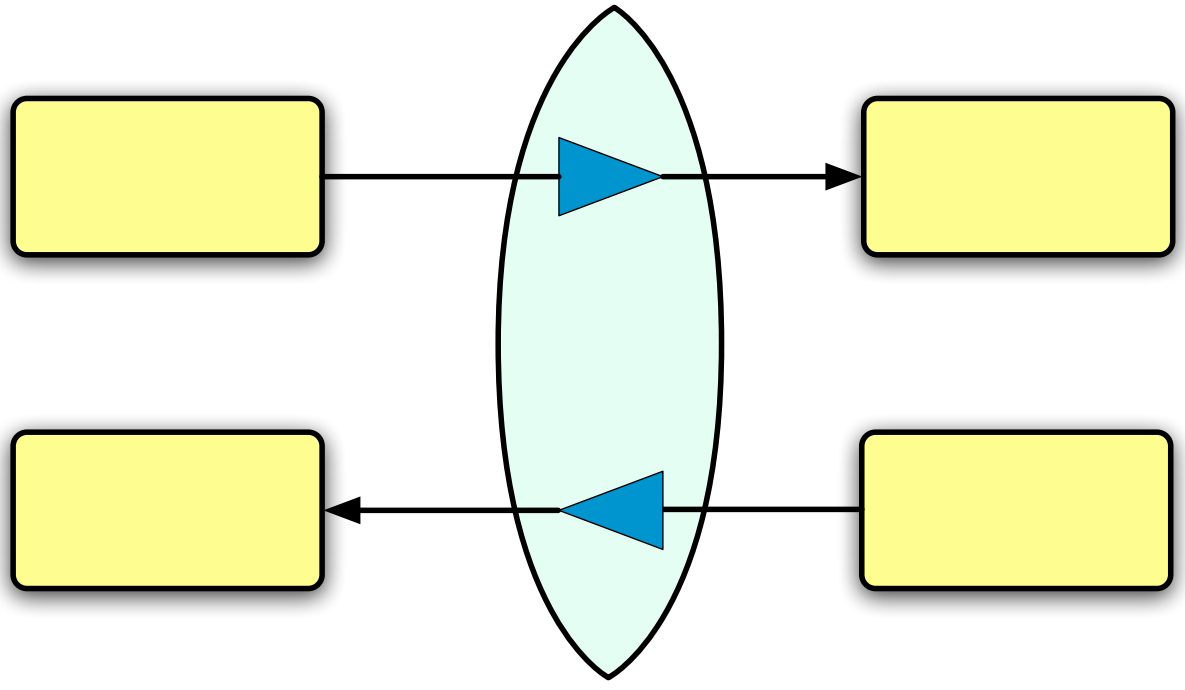


Bad: write the two transformations as separate functions.

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

Or we can use a language designed for the task at hand!

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

Thesis

"Bidirectional programming languages are an effective and elegant means of describing updatable views"

Outline

1. Lenses
   - Design goals
   - Semantics
2. String Lenses
   - Core operators
   - Type system
3. Quotient Lenses
4. Resourceful Lenses
5. Boomerang
   - High-level syntax
   - Implementation
   - Adoption in industry
6. Secure Lenses
7. Conclusion

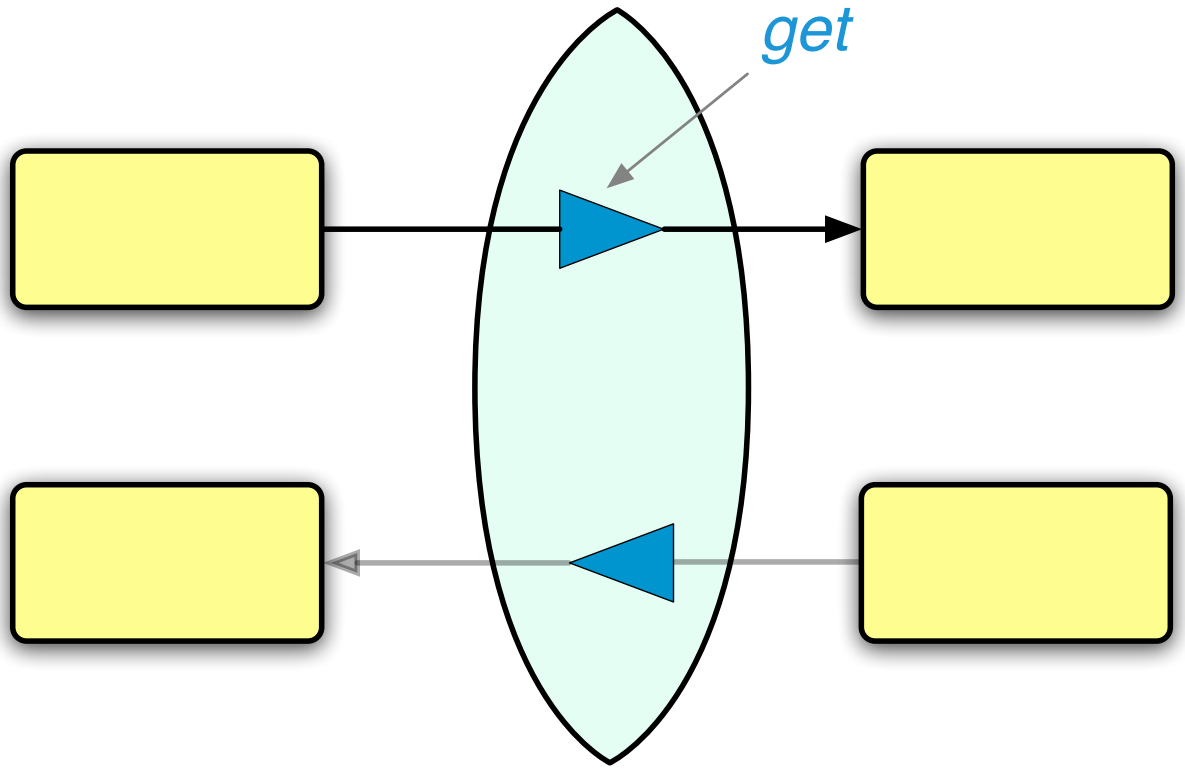
[Foster, Greenwald, Moore, Pierce, Schmitt TOPLAS '07]

Lenses

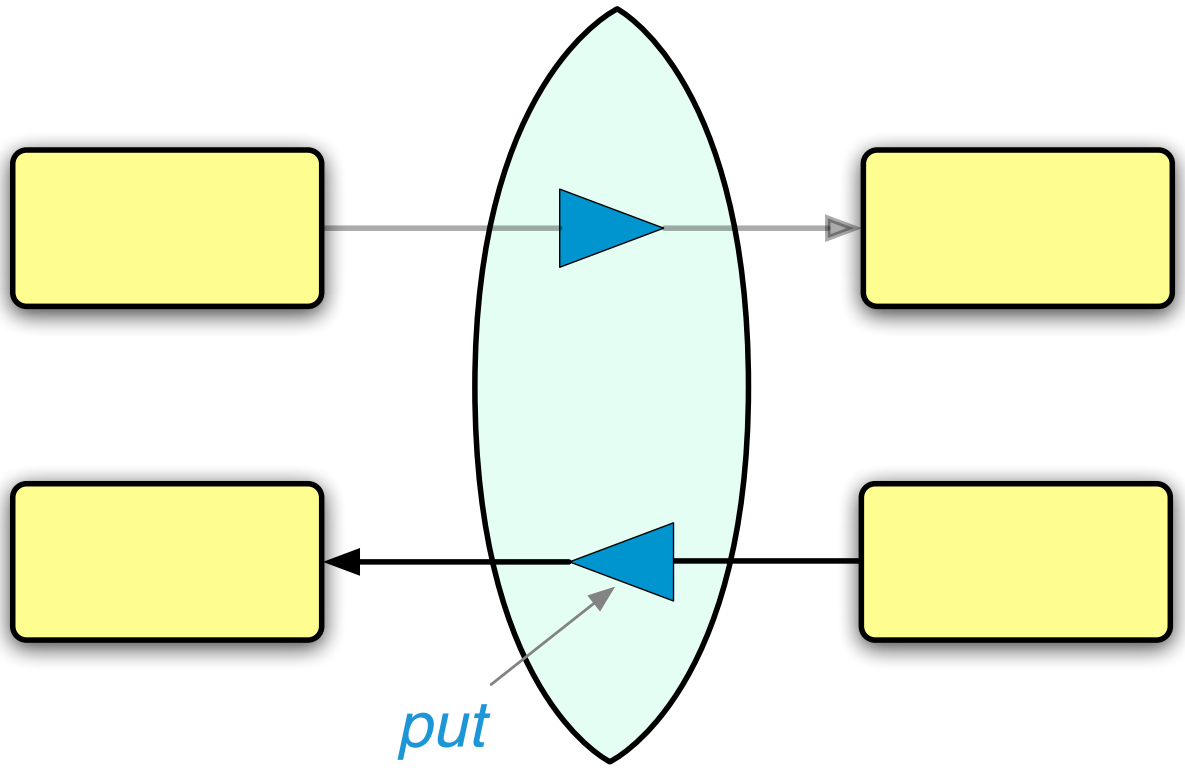
''Never look back unless you are planning to go that way"

—H D Thoreau

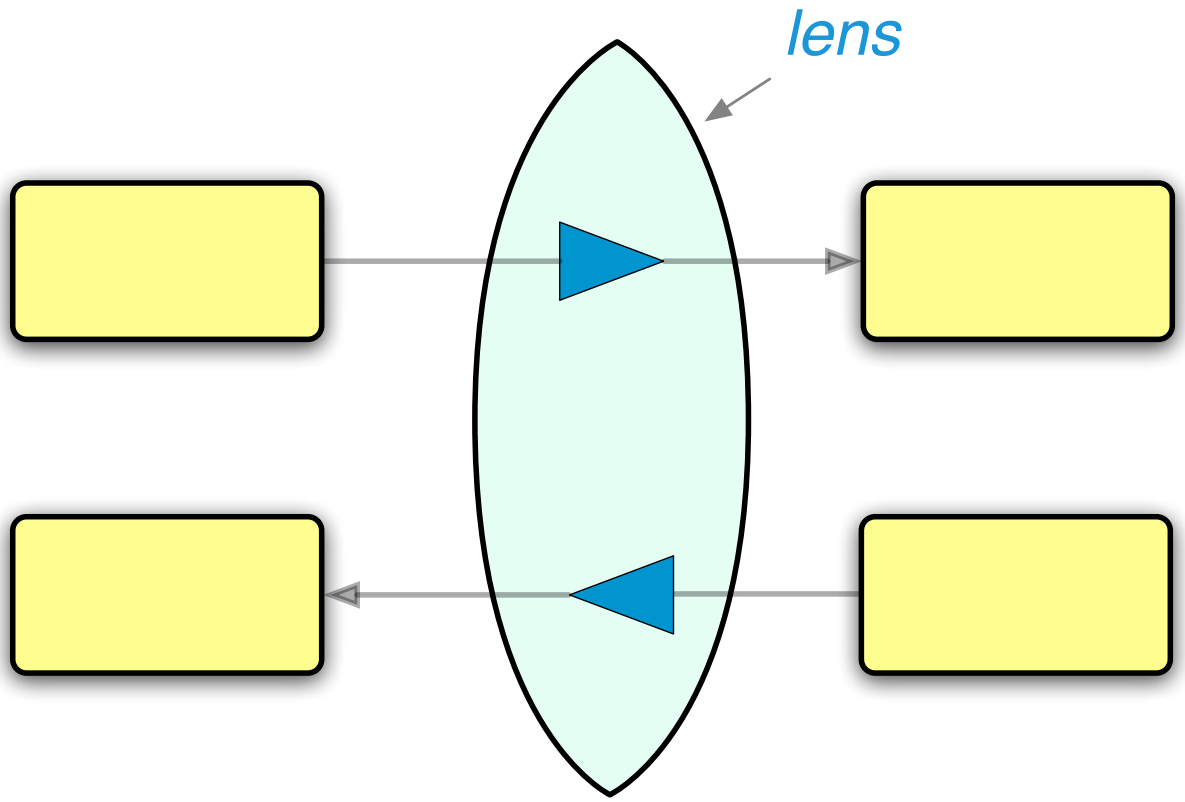
Terminology



Terminology



Terminology



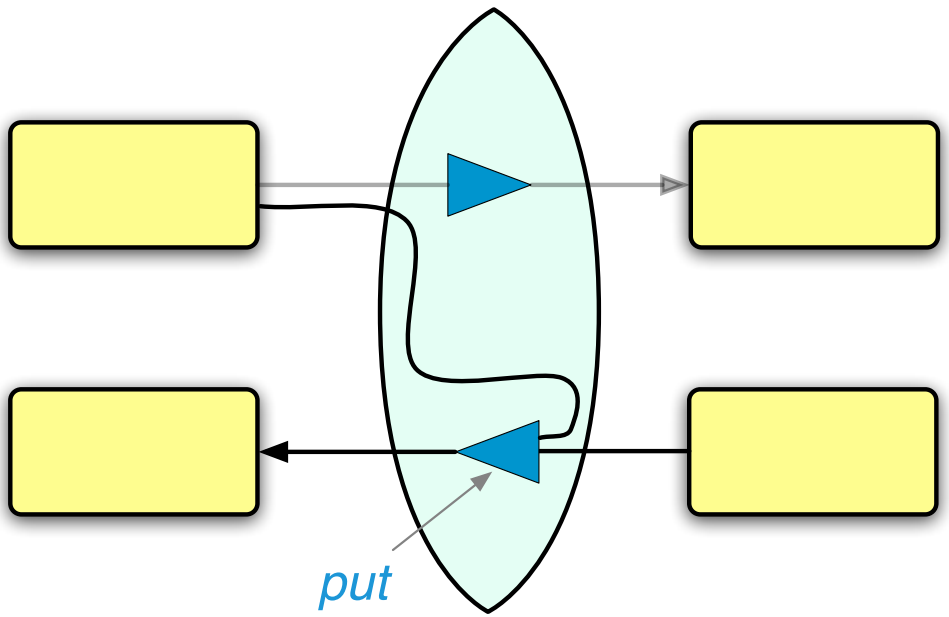
Bidirectional vs. Bijective

Goal #1: lenses should be capable of hiding source data.

Bidirectional vs. Bijective

Goal #1: lenses should be capable of hiding source data.

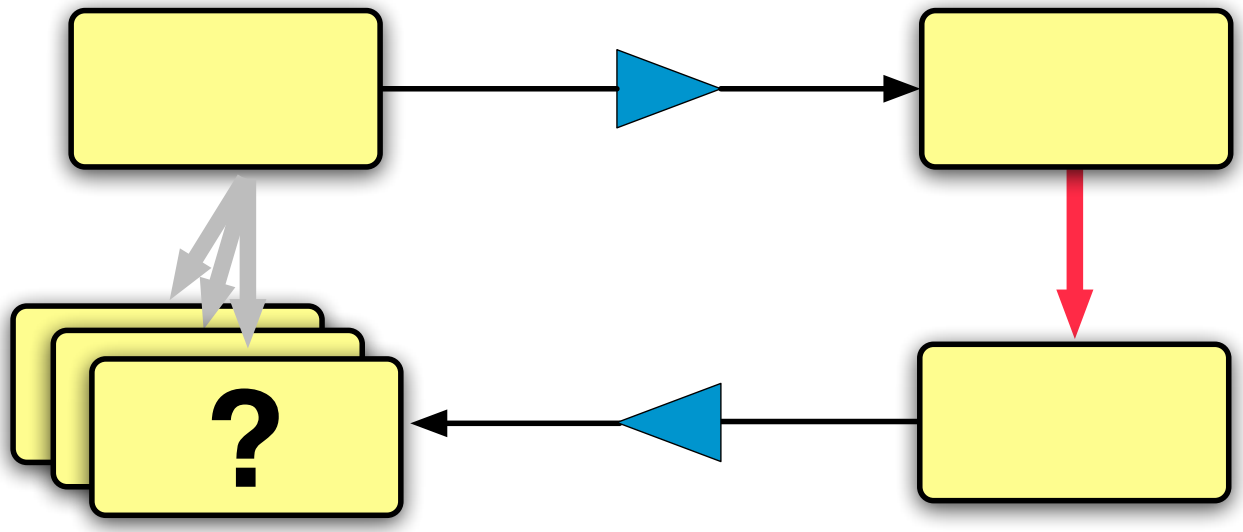
- In general, **get** may be non-injective
- and so **put** needs to take the original source as an argument



(Of course, the purely bijective case is also very interesting.)

Choice of Put Function

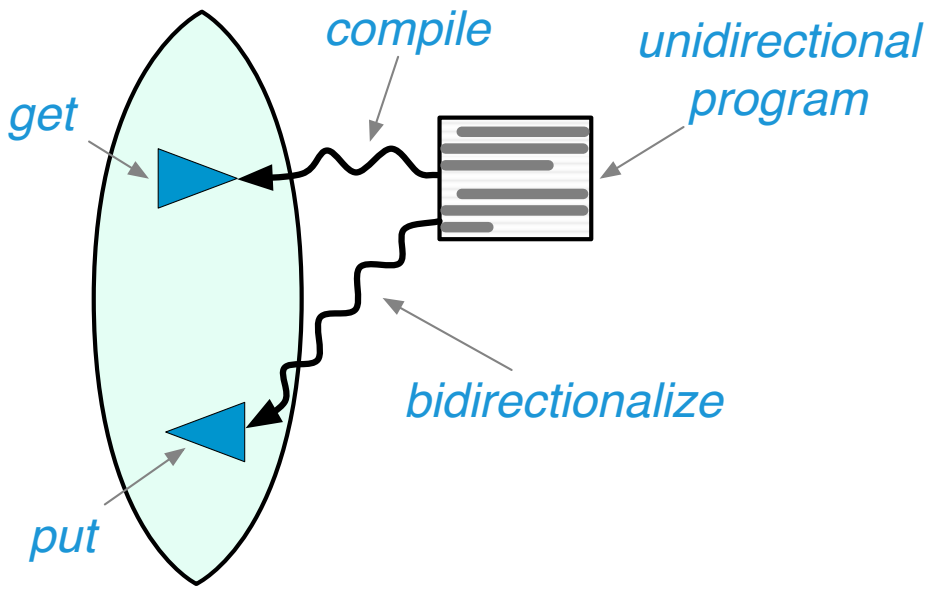
Recall that for some view updates there are *many* corresponding source updates.



Choice of Put Function

Goal #2: programmers should be able to choose a **put** function that embodies an appropriate policy for propagating updates back to sources.

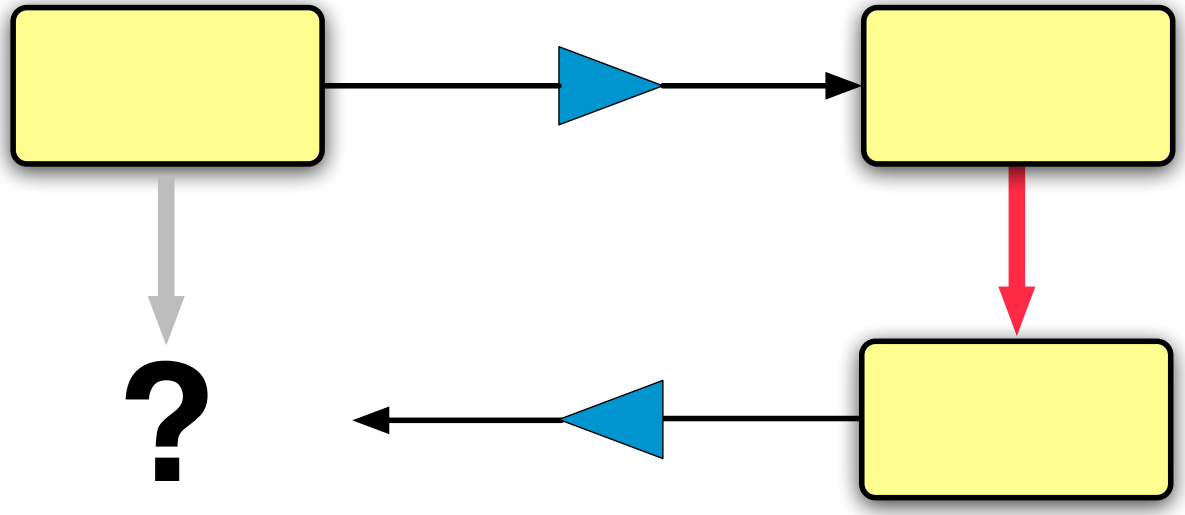
"Bidirectionalization" appears attractive...



...but does not provide a way to make this choice.

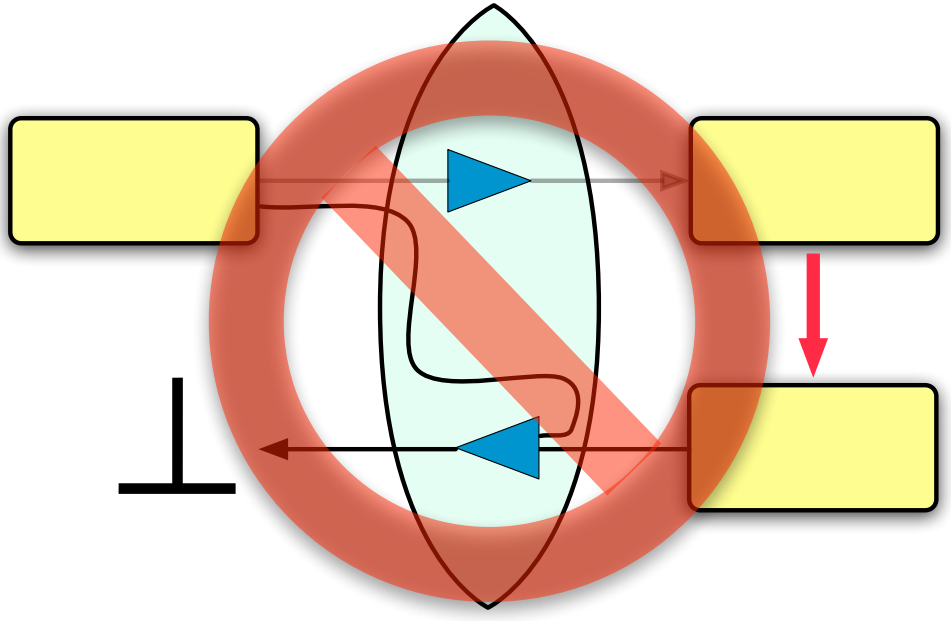
Totality

Recall that some view updates do not have *any* corresponding source updates.



Totality

Goal #3: the **put** function should be a total function, capable of doing *something* reasonable with every view and source.



Totality ensures that the view is a robust abstraction, but forces us to use an extremely precise type system.

Well-Behaved Lenses

A 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

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

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

for every $s \in S$ and $v \in V$.

Related Frameworks

Databases: *many* related ideas

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

Quantum Computing: [Bennet '73] "reversible Turing machine"

User Interfaces: [Meertens '98] "constraint maintainers"

Related Languages

Harmony Group @ Penn

- [Foster *et al.* TOPLAS '07] — trees
- [Bohannon, Pierce, Vaughan PODS '06] — relations
- [Foster *et al.* JCSS '07] — data synchronizer

Bijective languages

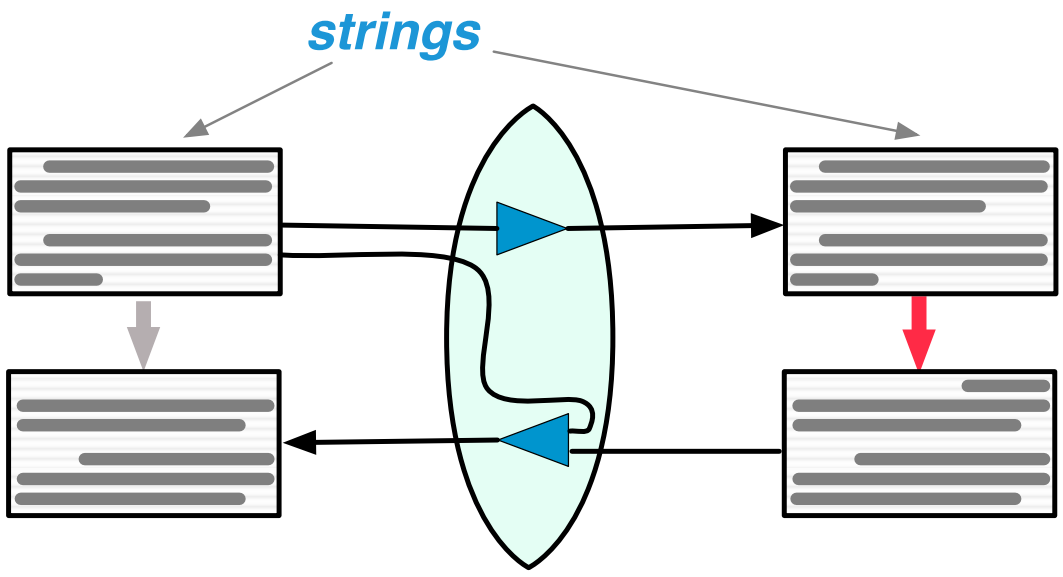
- [PADS Project @ AT&T] — picklers and unpicklers
- [Hosoya, Kawanaka '06] — biXid
- [Braband, Møller, Schwartzbach '05] — XSugar

Bidirectional languages

- [PSD @ Tokyo] — "bidirectionalization", structure editors
- [Gibbons, Wang @ Oxford] — Wadler's views
- [Voïgtlaender '09] — bidirectionalization "for free"
- [Stevens '07] — lenses for model transformations

String Lenses

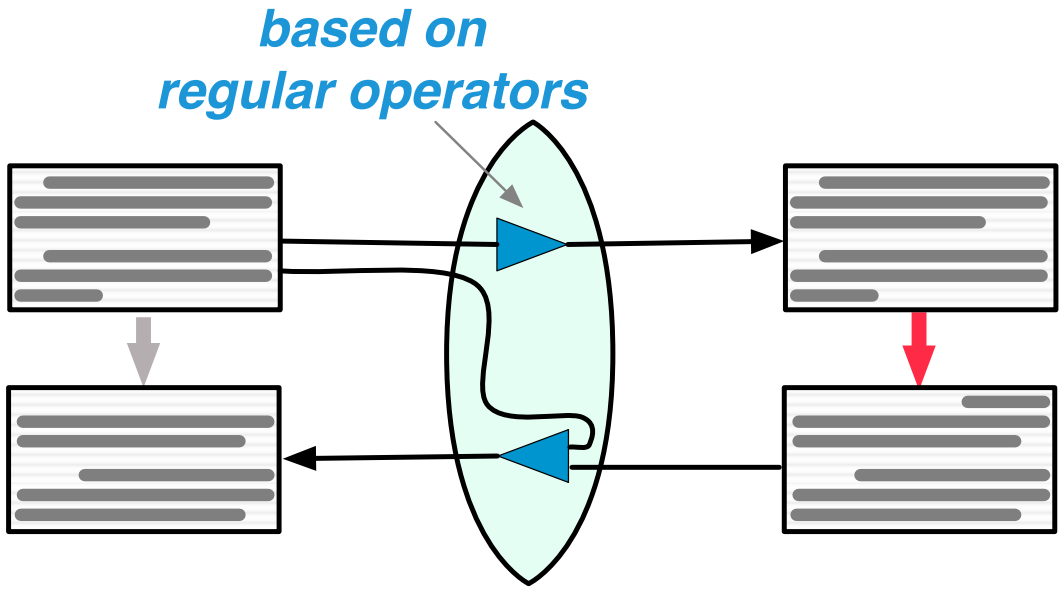
Data Model



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

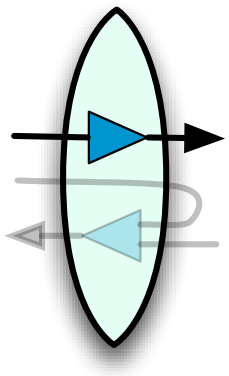


Why strings?

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)

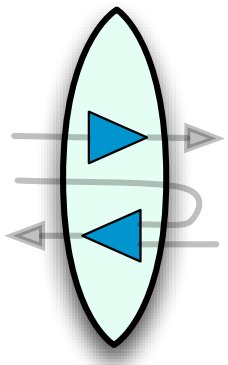
```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 PLClu
15:00 BUSY
```

Example: Redacting Lens (Update)

```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```



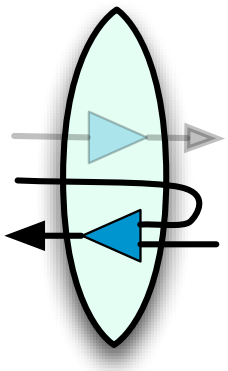
```
08:30 BUSY
12:15 PLClu
15:00 BUSY
```

```
08:30 BUSY
12:15 PLClub
15:00 BUSY
16:00 Meeting
```

Example: Redacting Lens (Put)

```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```

```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClub (Seminar room)
*15:00 Workout (Gym)
 16:00 Meeting (Unknown)
```



```
08:30 BUSY
12:15 PLClu
15:00 BUSY
```



```
08:30 BUSY
12:15 PLClub
15:00 BUSY
16:00 Meeting
```

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*
```

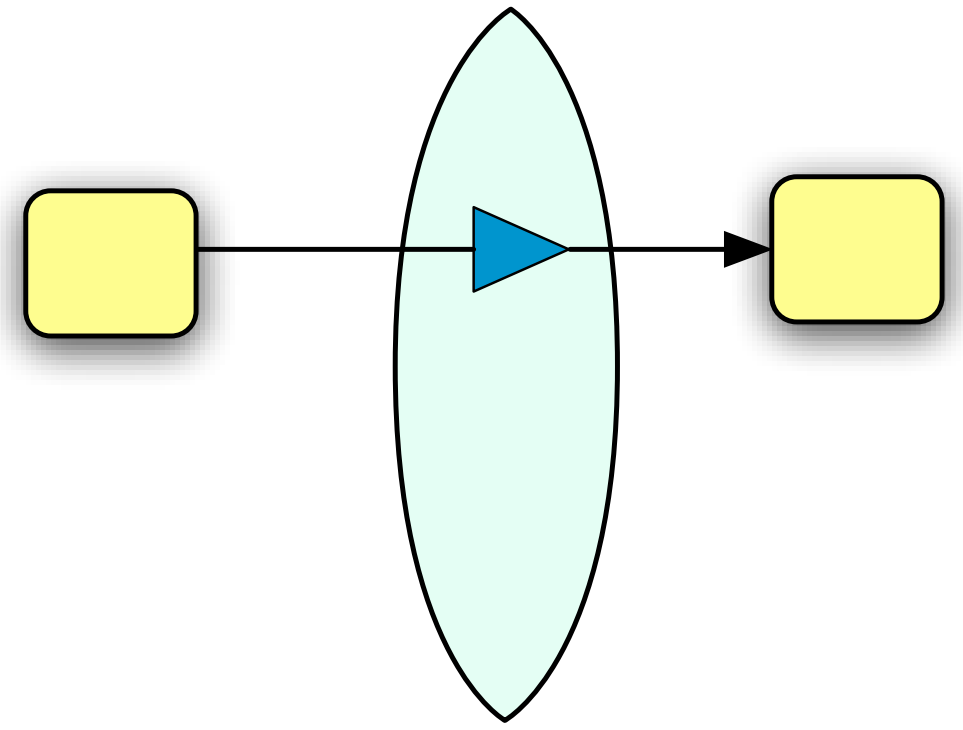
Example: Redacting Lens (Definition)

```
(* 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*
```

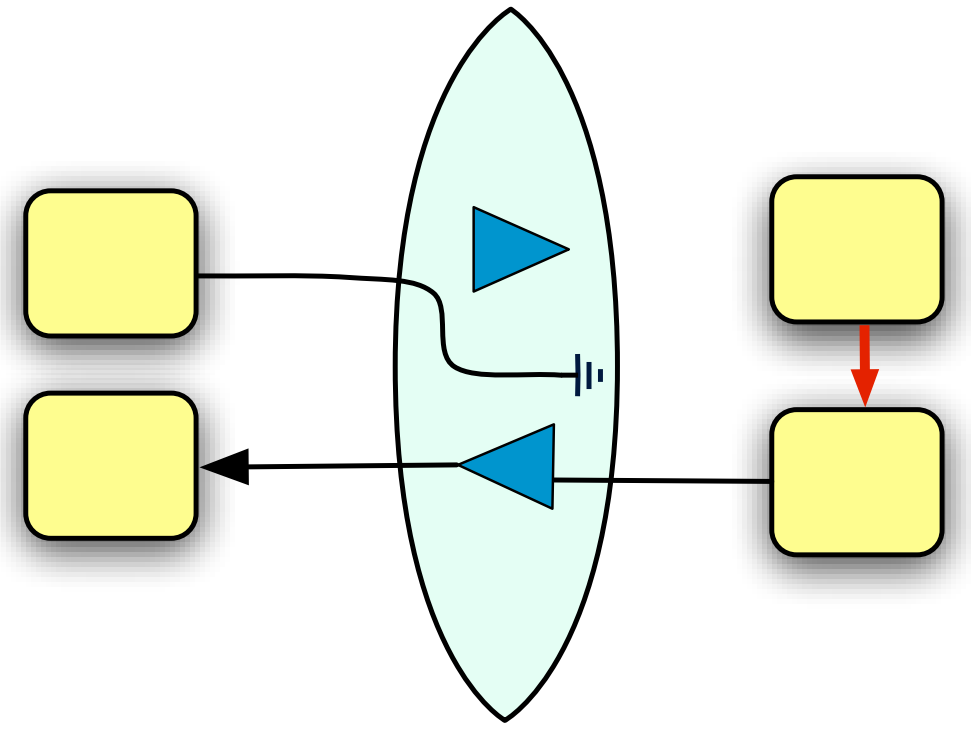
Example: Redacting Lens (Definition)

```
(* 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*
```

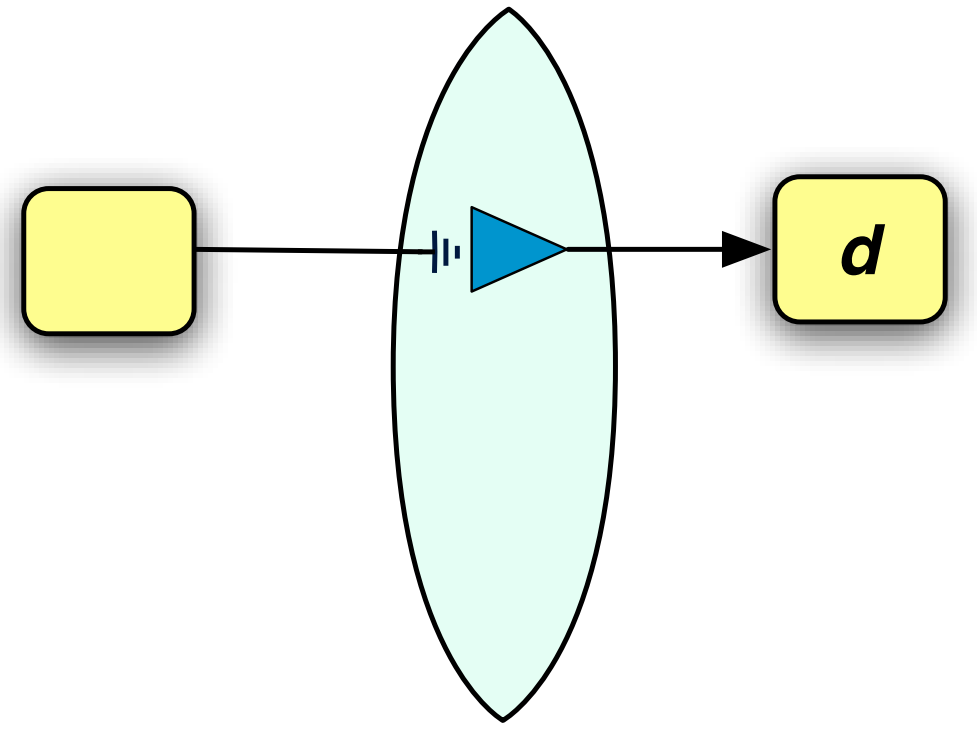
copy E (Get)



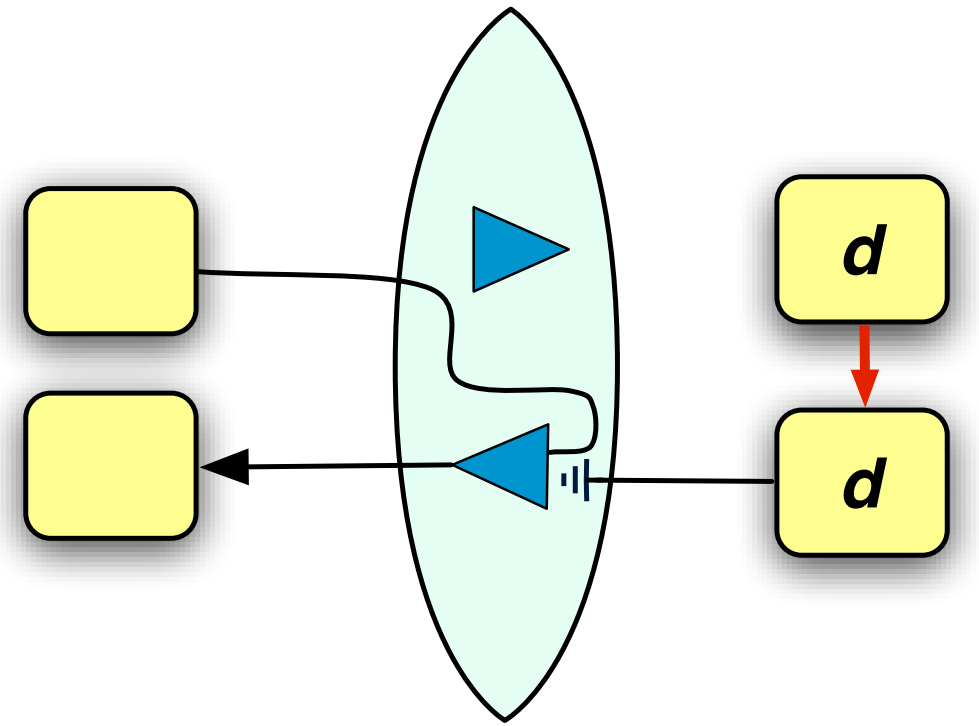
copy E (Put)



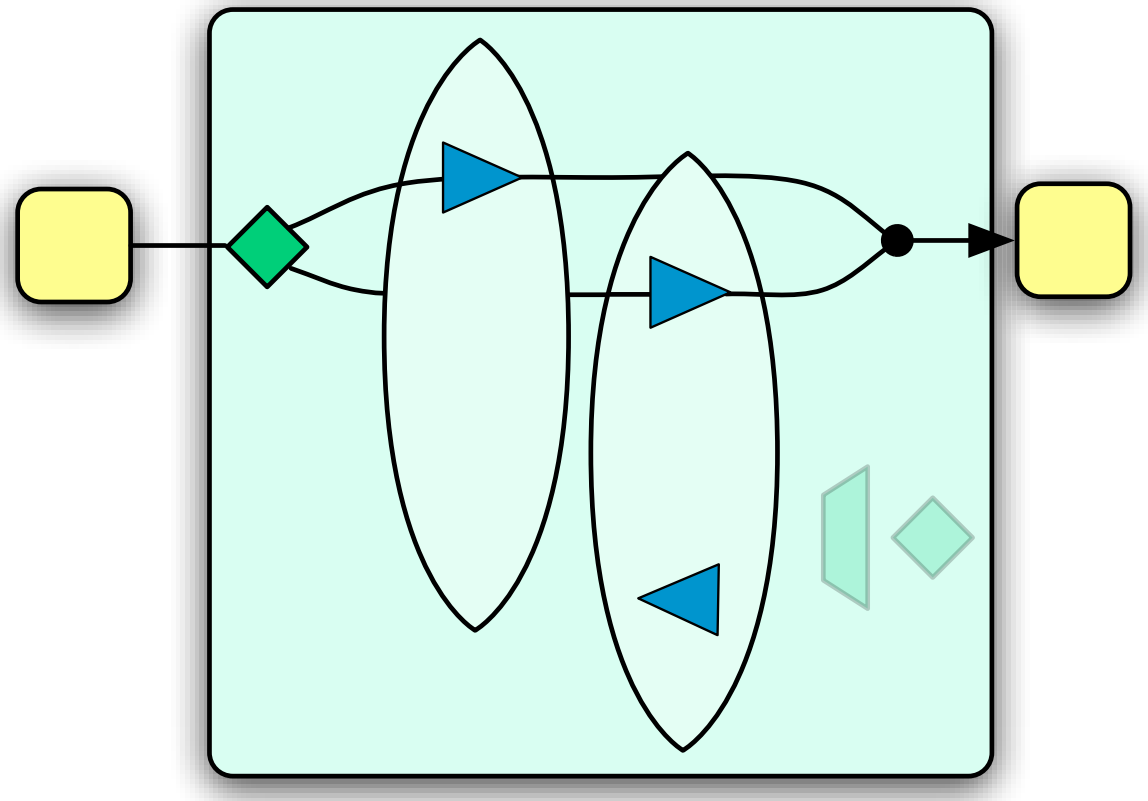
$E \leftrightarrow d$ (Get)



$E \leftrightarrow d$ (Put)

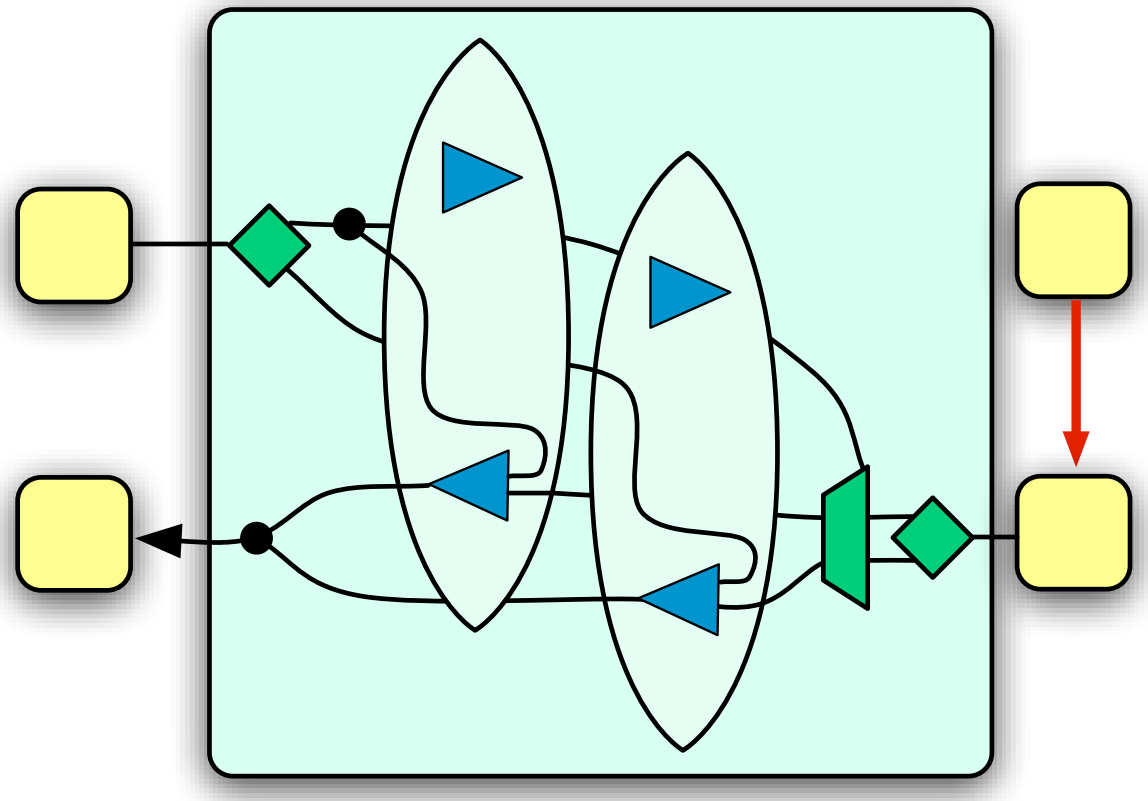


$(l_1 \mid l_2)$ (Get)



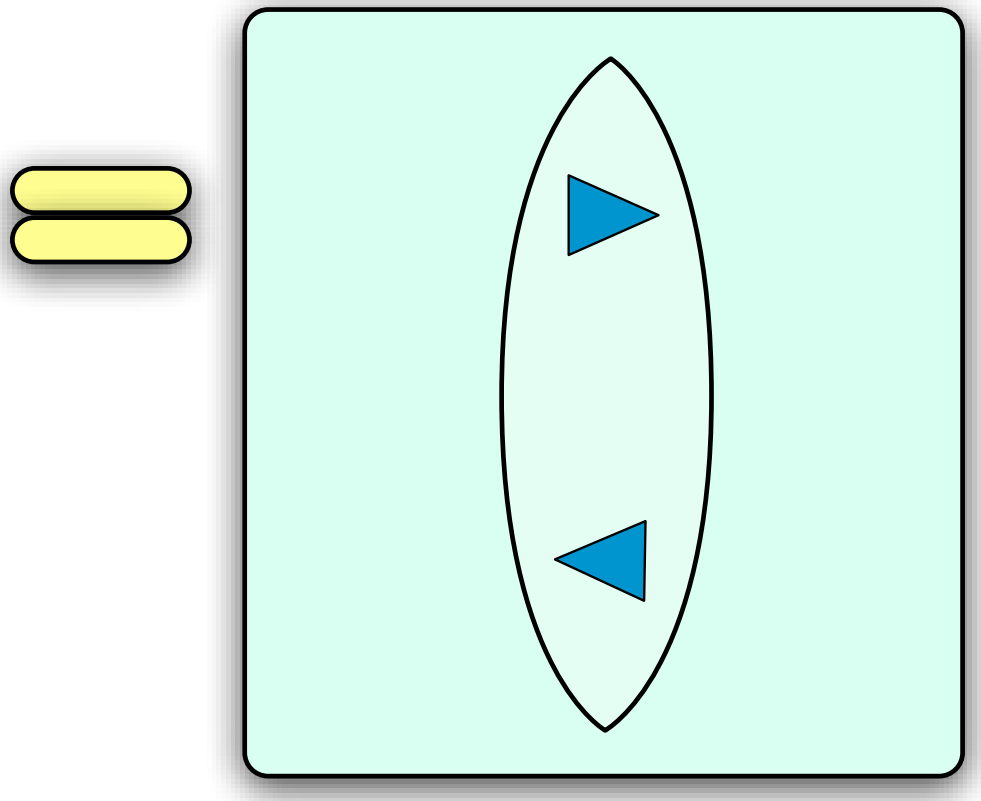
Type system ensures that choice is deterministic.

$(l_1 \mid l_2)$ (Put)

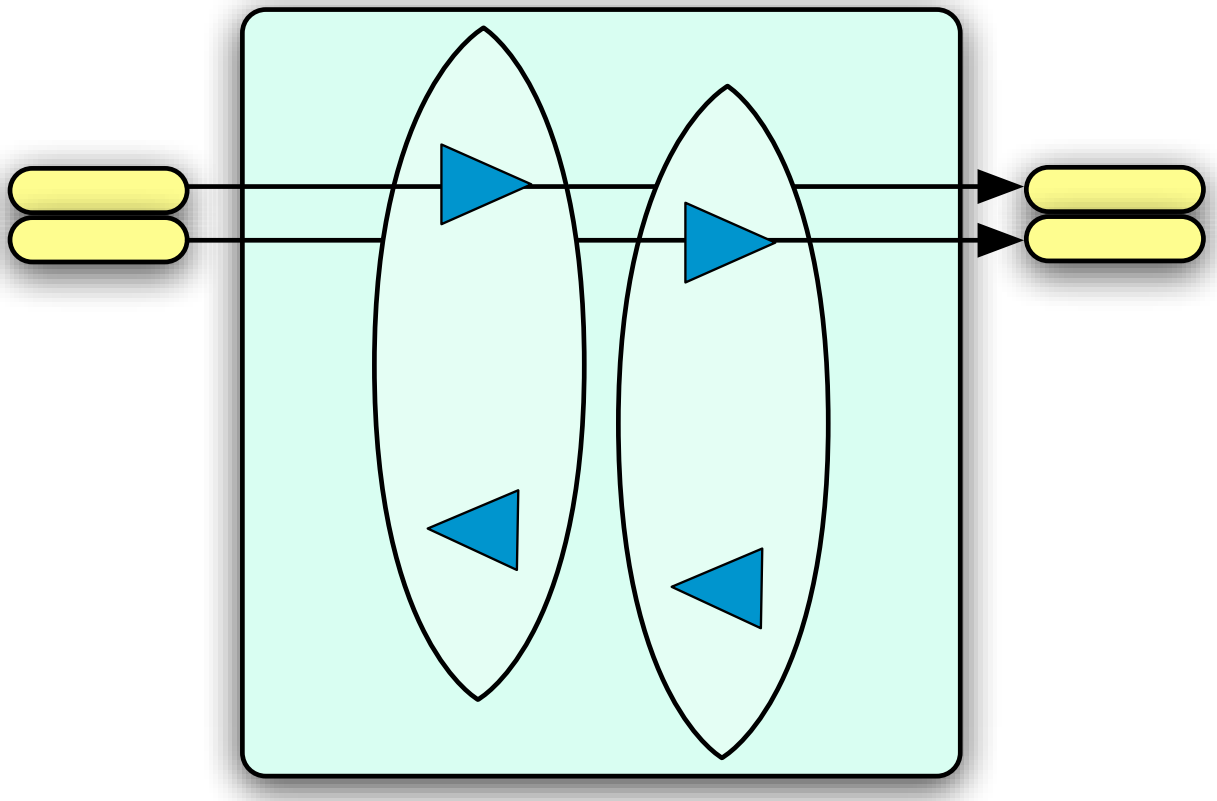


Type system ensures that choice is deterministic.

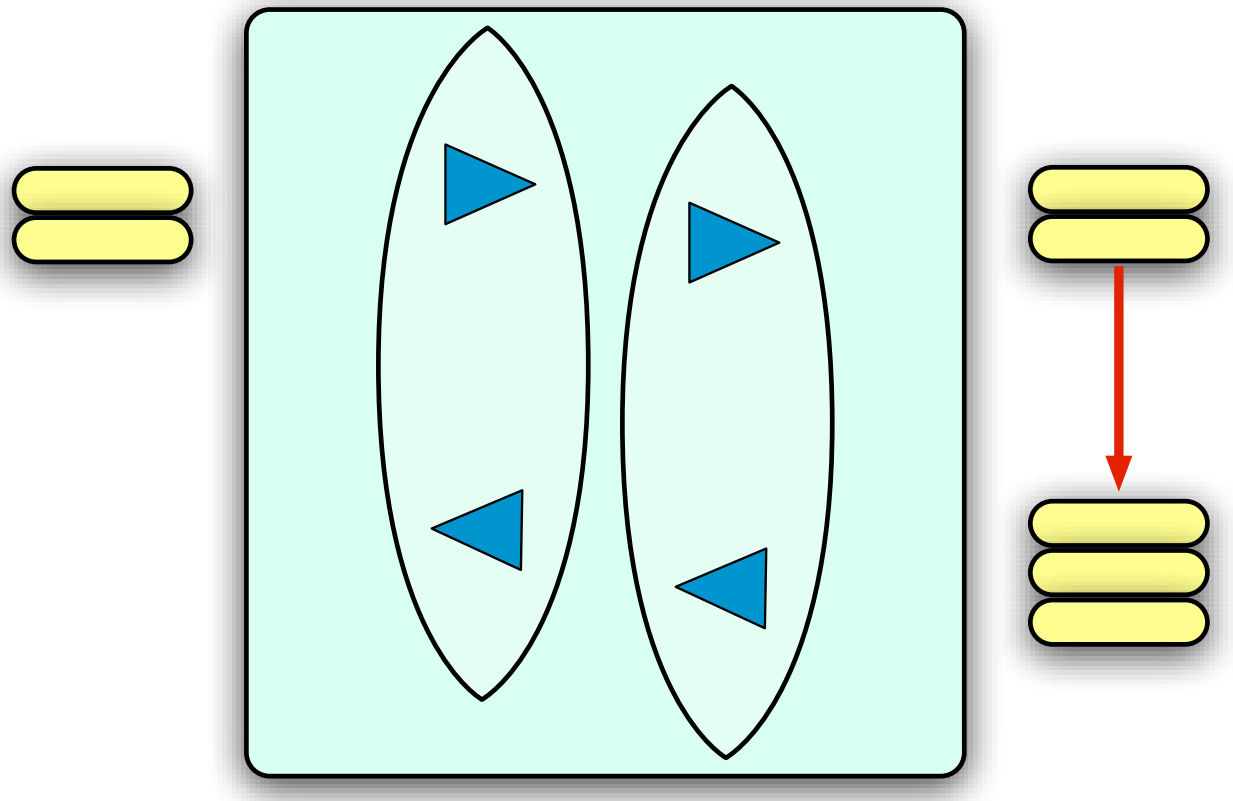
/* (Get)



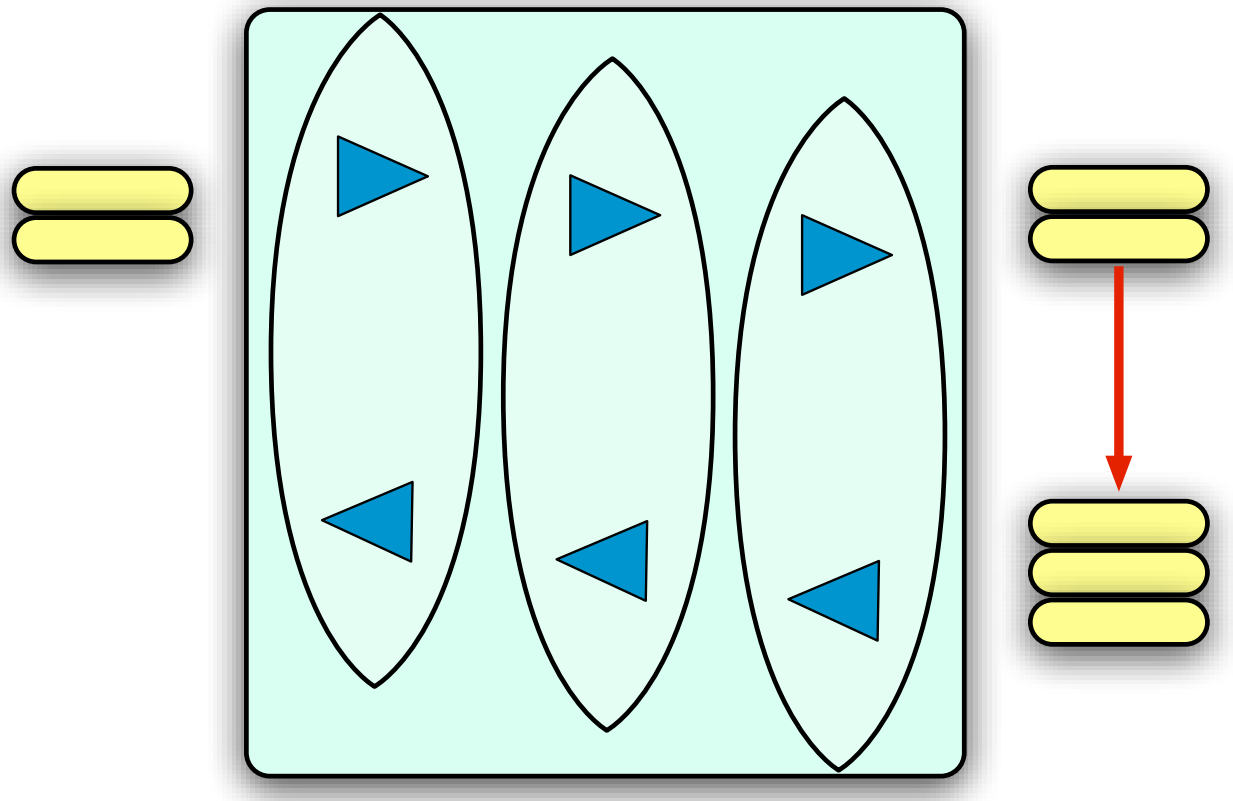
/* (Get)



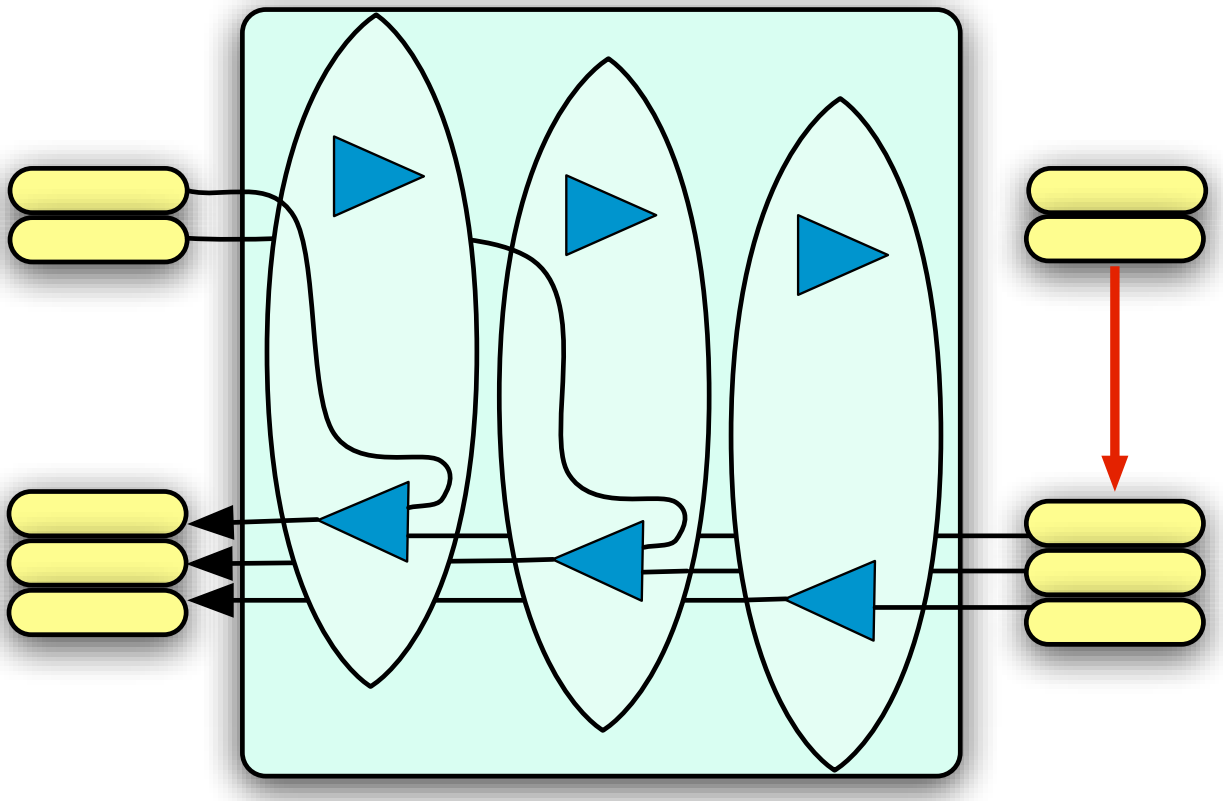
/* (Put)



/* (Put)



/* (Put)



Type system ensures that strings are split the same way.

String Lens Type System

Based on regular expression types...

String Lens Type System

Based on regular expression types...

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

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

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

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

String Lens Type System

Based on regular expression types...

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

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

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

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

String Lens Type System

Based on regular expression types...

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

$$\frac{l \in S \Longleftrightarrow V \qquad d \in [\![S]\!]}{\mathit{default}\ l\ d \in S \Longleftrightarrow V}$$

$$\frac{l_1 \in S_1 \Longleftrightarrow V_1 \qquad S_1 \mathbin{.^!} S_2 \\ l_2 \in S_2 \Longleftrightarrow V_2 \qquad V_1 \mathbin{.^!} V_2}{(l_1 \cdot l_2) \in S_1 \cdot S_2 \Longleftrightarrow V_1 \cdot V_2}$$

$$\frac{l_1 \in S_1 \Longleftrightarrow V_1 \qquad S_1 \cap S_2 = \emptyset \\ l_2 \in S_2 \Longleftrightarrow V_2}{(l_1 \mid l_2) \in S_1 \cup S_2 \Longleftrightarrow V_1 \cup V_2}$$

$$\frac{l \in S \Longleftrightarrow V \qquad S^{!*} \qquad V^{!*}}{l^* \in S^* \Longleftrightarrow V^*}$$

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

Theorem

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

Comparison: Separate Functions

Helpers

```
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][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
```

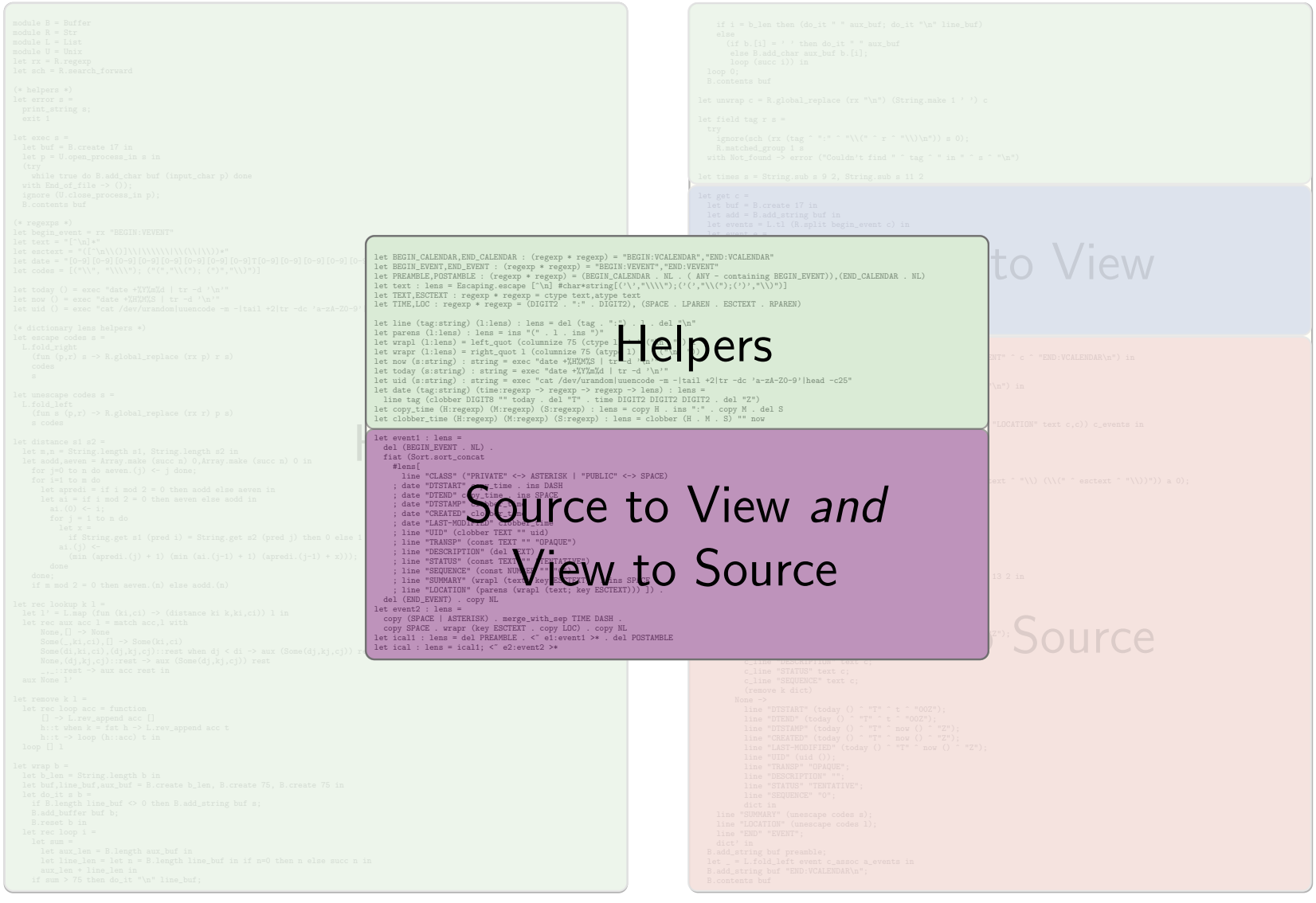
Source to View

```
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
```

View to Source

```
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" (today () ^ "T" ^ 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
```

Comparison: String Lens



[Foster, Pilkiewicz, Pierce ICFP '08]

Quotient Lenses

"Good men must not obey the laws too well"

—R W Emerson

Challenge: Ignorable Data

Many real-world data formats contain inessential data.

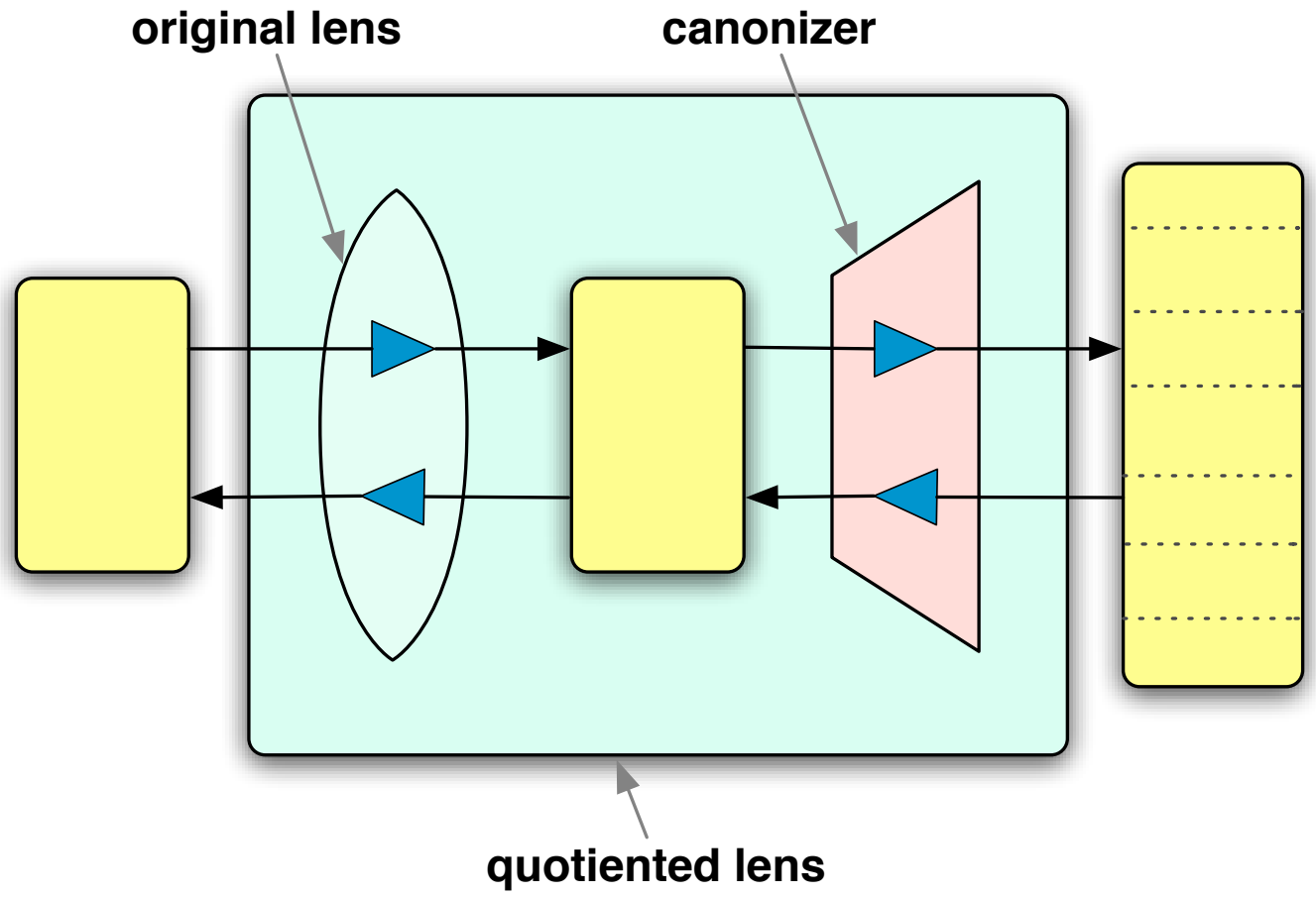
- whitespace, wrapping of long lines of text
- order of fields in record-structured data
- escaping of special characters
- aggregate values, timestamps, etc.

In practice, to handle these details, we need lenses that are well behaved modulo equivalence relations on the source and view.

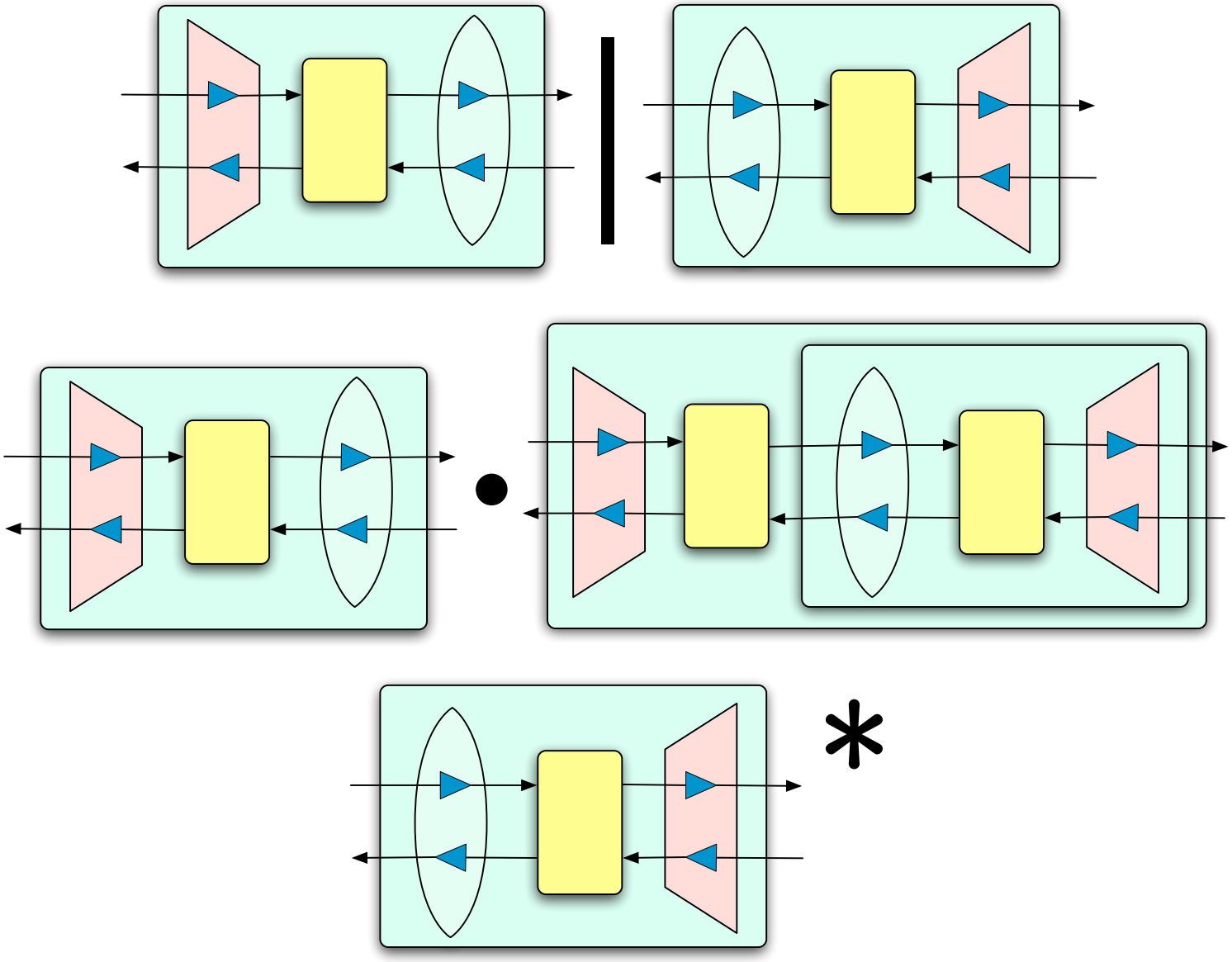
$$l.\mathbf{get}\ (l.\mathbf{put}\ v\ s) \sim_V v \qquad (\textsc{PutGet})$$

$$l.\mathbf{put}\ (l.\mathbf{get}\ s)\ s \sim_S s \qquad (\textsc{GetPut})$$

Quotient Lenses



Quotient Lenses



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

Resourceful Lenses

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

Challenge: Ordered Data

The lenses we have seen so far align data by position.

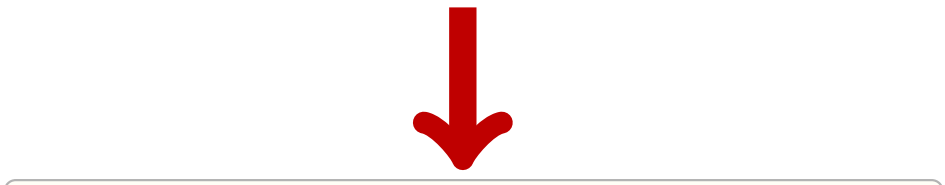
But we often need to align data according to different criteria—e.g., using part of the view as a key.

Challenge: Ordered Data

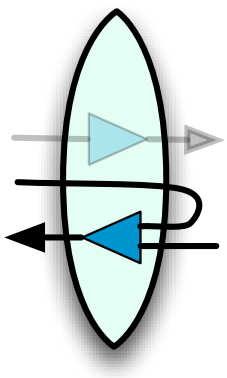
The lenses we have seen so far align data by position.

But we often need to align data according to different criteria—e.g., using part of the view as a key.

```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```



```
*08:30 Coffee with Sara (Starbucks)
 11:45 Meeting (Seminar Room)
 12:15 PLClub (Unknown)
*15:00 Unknown (Unknown)
```



```
08:30 BUSY
12:15 PLClu
15:00 BUSY
```



```
08:30 BUSY
11:45 Meeting
12:15 PLClub
15:00 BUSY
```

A Better Redact Lens

Similar to previous version but with `key` annotations and a new combinator that identifies reorderable "chunks"

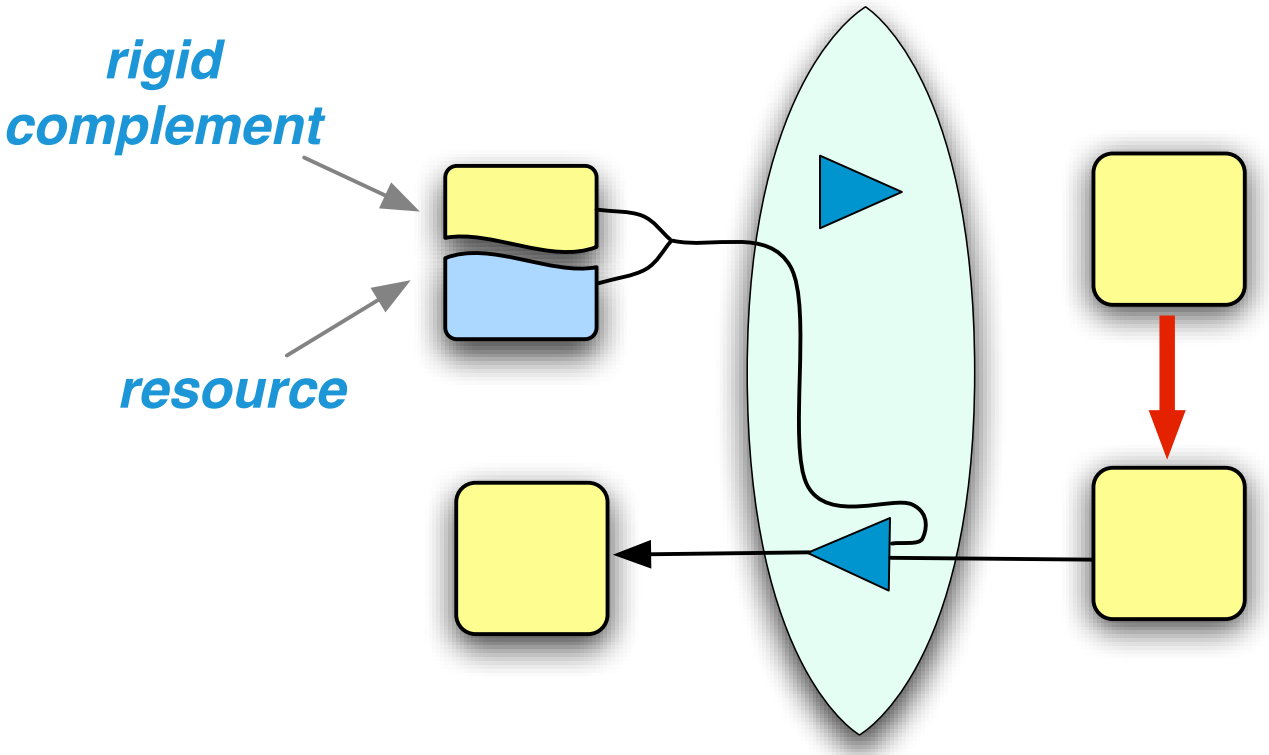
```
(* helper lenses *)
let location : lens = default (del LOCATION) " (Unknown)"

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

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

(* main lens *)
let redact : lens = < sim : event>*
```

A Better Redact Lens

Similar to previous version but with `key` annotations and a new combinator that identifies reorderable "chunks"

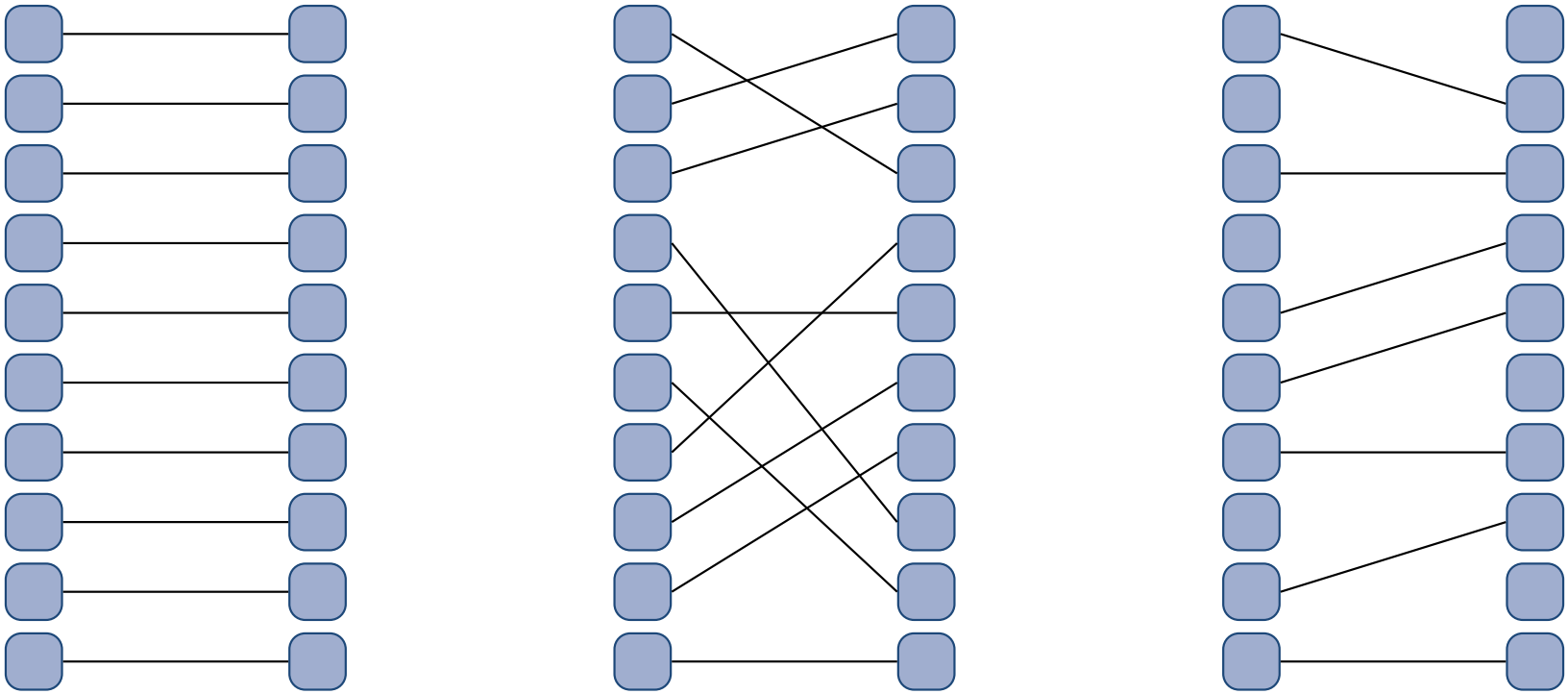
```
(* helper lenses *)
let location : lens = default (del LOCATION) " (Unknown)"

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

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

(* main lens *)
let redact : lens = < sim : event>*
```

Resourceful Lenses



The **put** function takes a rigid complement and a resource instead of the actual source.

Alignment



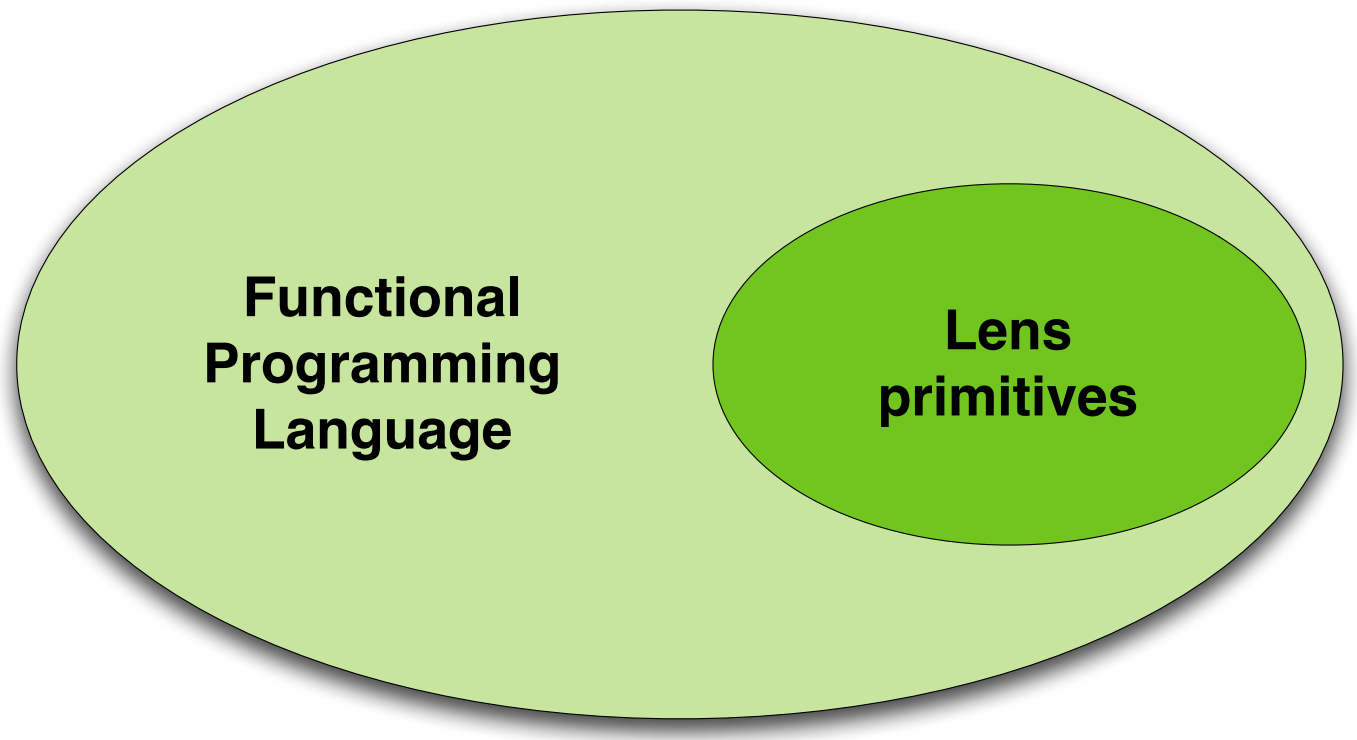
The resource can be reordered, using any heuristic we like to align the chunks of the source and view.

Boomerang

Challenge: Language Design

Writing big programs only using combinators would not be fun!

Boomerang is a full-blown functional language over the base types `string`, `regexp`, `lens`,...



Additional Features

Boomerang has other primitives...

- partition
- filter
- permute
- sort
- duplicate
- merge
- sequential composition
- columnize
- normalize
- clobber
- probe
- etc.

and an extremely rich type system...

- regular expression types
- dependent types
- refinement types
- polymorphism
- user-defined datatypes
- modules

implemented in hybrid style [Flanagan '06][Findler, Wadler '09]

Challenge: Typechecker Engineering

Typechecking uses *many* automata-theoretic operations.

- "Expensive" operations like intersection, difference, and interleaving are used often in practice
- Algorithms for checking ambiguity are computationally expensive rarely implemented

Implementation strategy:

- Compile compact automata [Brzozoswki '64]
- Aggresive memoization [Foster *et al.* PLAN-X '07]

The Boomerang System

Lenses

- Bibliographies (BibTeX, RIS)
- Address Books (vCard, XML, ASCII)
- Calendars (iCal, XML, ASCII)
- Scientific Data (SwissProt, UniProtKB)
- Documents (MediaWiki, literate source code)
- Apple Preference Lists (e.g., iTunes)
- CSV

Libraries

- Escaping
- Sorting
- Lists
- XML

System

- Stable prototype complete
- Available under LGPL

Unison Integration

- Coming...

Boomerang in Industry

"a configuration API."

aliases.aug	exports.aug	logrotate.aug	puppet.aug	sudoers.aug
aptpreferences.aug	fstab.aug	monit.aug	rsyncd.aug	sysctl.aug
aptsources.aug	gdm.aug	ntp.aug	samba.aug	util.aug
bbhosts.aug	group.aug	openvpn.aug	services.aug	vsftpd.aug
crontab.aug	grub.aug	pam.aug	shellvars.aug	webmin.aug
darkice.aug	hosts.aug	passwd.aug	slapd.aug	xinetd.aug
dhclient.aug	inifile.aug	php.aug	soma.aug	xorg.aug
dnsmasq.aug	inittab.aug	phpvars.aug	spacevars.aug	yum.aug
dpkg.aug	interfaces.aug	postfix_main.aug	squid.aug	
dput.aug	limits.aug	postfix_master.aug	sshd.aug	

Also used in

- Puppet – declarative configuration management tool
- Show – SQL-like queries on the filesystem
- Netcf – a network configuration library

Boomerang in Industry

"a configuration API."

```
Date: Thu, 13 Aug 2009 11:33:42
From: Matthew Palmer <matt@anchor.net.au>
To: augeas-devel@redhat.com
Subject: 2009 Lens Fiesta! (inetd.conf edition)

> Who ever said writing lenses was hard ? ;)

Probably me, before I got my head around the syntax.
It really is mind-meltingly weird.  I can't see how
you'd do something this powerful any other way, but
I can't fault people who look at lenses and say "you
know what, I think I've got to go feed my cat".
```

[Foster, Pierce, Zdancewic CSF '09]

Secure Lenses

"Whoever wishes to keep a secret must
hide the fact that he possesses one."
—J W von Goethe


AMERICAN EXPRESS
3746
11002
FECHA DE VALIDEZ / VALID DATE
09/18
12/18
© AMEX
VISA
UEFA CHAMPIONS LEAGUE


STATE OF CALIFORNIA
COUNTY OF SAN MATEO
VOTER REGISTRATION FORM
Jane
Doe
123 Main St.
San Mateo
CA 94032


SEP
94
FEB
00
JUL
03
OCT
99
MAY
03
APR
03
J
O
H
J
O
N
J
O
R
J
U
R
J
U
T
K
A
R
K
K
K
K

The Washington Post

"Pennsylvania yanks voter site after data leak"

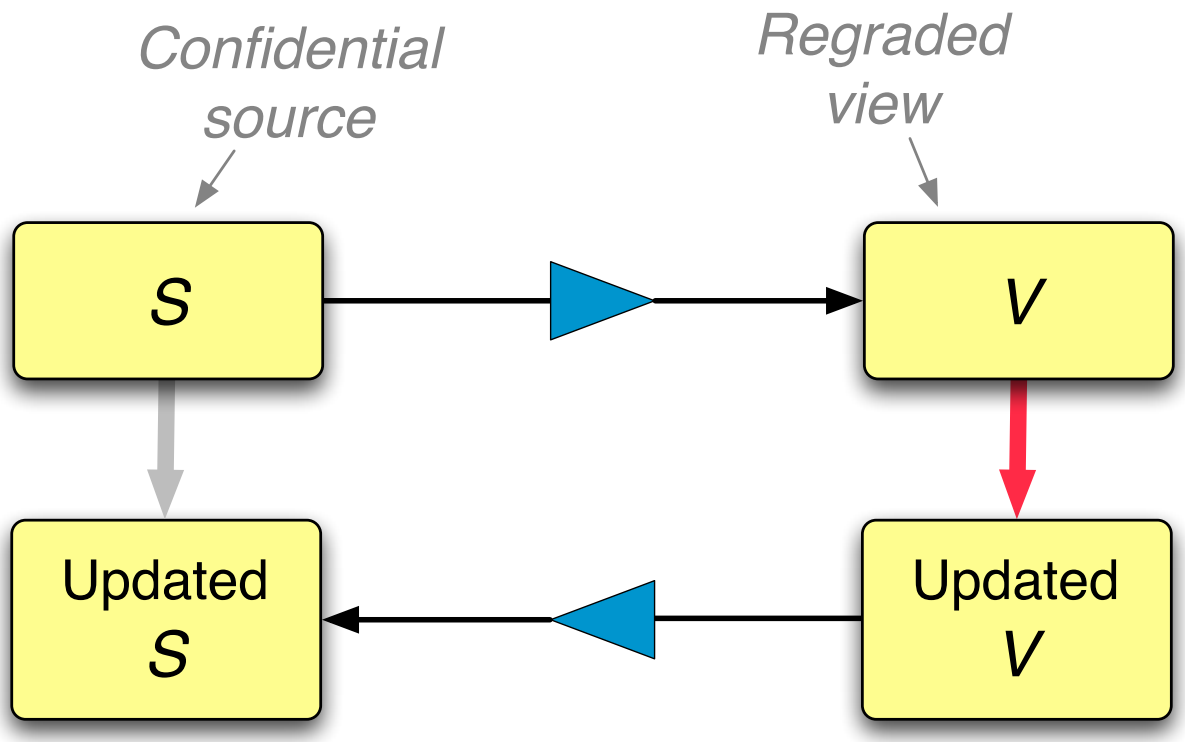
THE GLOBE AND MAIL
CANADA'S NATIONAL NEWSPAPER

"Passport applicant finds massive privacy breach"

The New York Times

"Privacy issue complicates push to link medical data"

Challenge: Updating Security Views

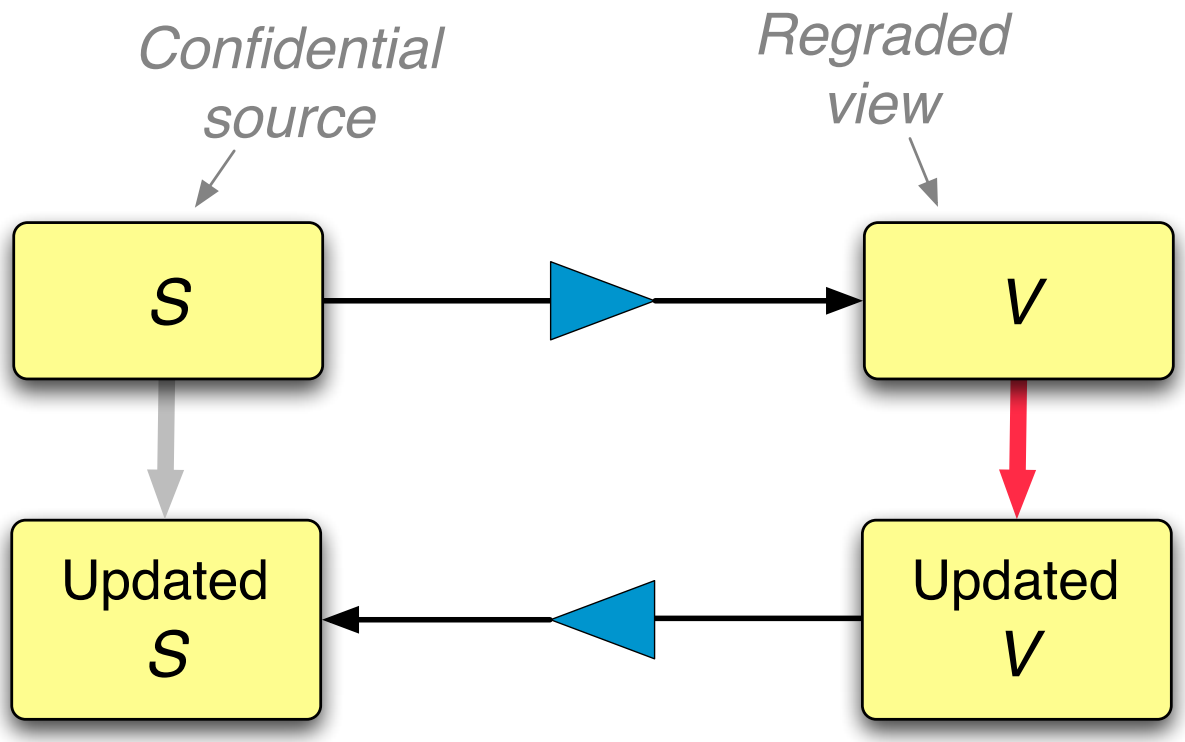


✔ Robust: impossible to leak hidden data

✔ Flexible: enforce fine-grained confidentiality policies

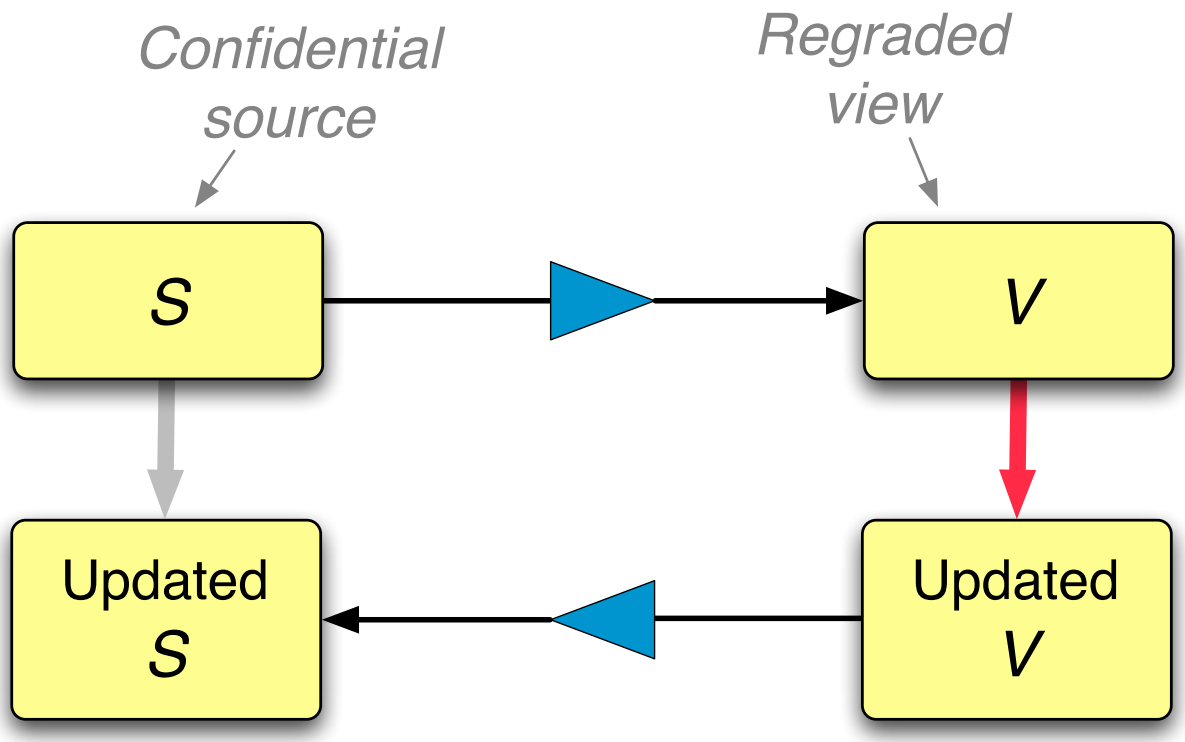
✗ Not usually updatable

Requirements



1. Confidentiality: **get** does not leak secret data
2. Integrity: **put** does not taint trusted data

Requirements

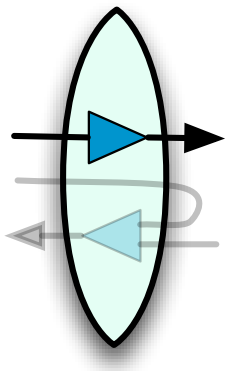


1. Confidentiality: **get** does not leak secret data
2. Integrity: **put** does not taint trusted data

Today

Example: Redacting Calendars (Get)

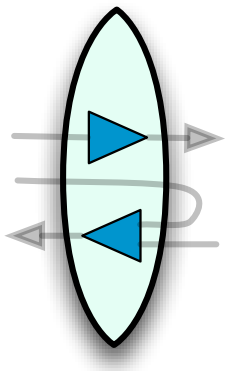
```
*08:30 Coffee with Sara (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

Example: Redacting Calendars (Update)

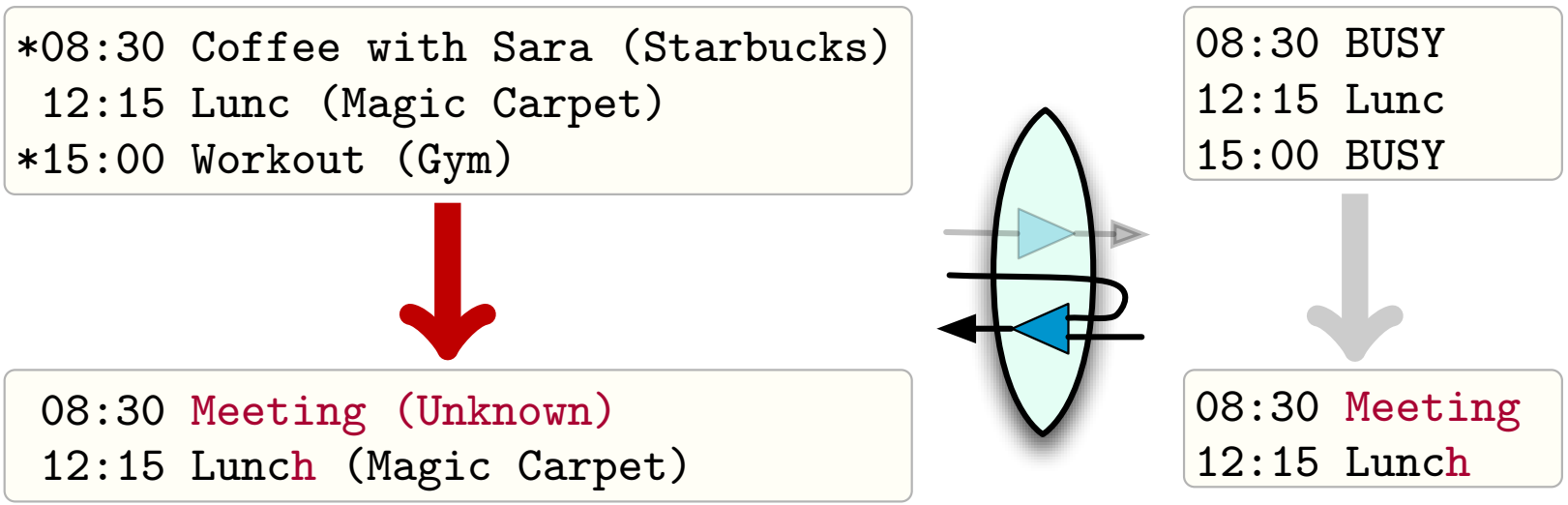
```
*08:30 Coffee with Sara (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

```
08:30 Meeting
12:15 Lunch
```

Example: Redacting Calendars (Put)



Observe that propagating the update to the view back to the source forces **put** to modify some of the hidden source data:

- The entire appointment at 3pm.
- The description and location of the appointment at 8:30am.

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

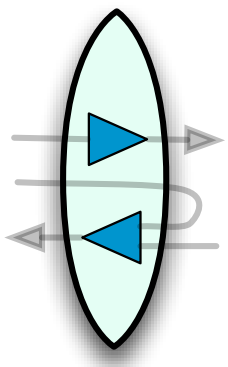
Answer: Maybe! There are *many* alternatives, trading off which information in the source is trusted against which information in the view can be modified.

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

Answer: Maybe! There are *many* alternatives, trading off which information in the source is trusted against which information in the view can be modified.

```
*08:30 Coffee with Sara (Starbucks)
12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

Policy: "Nothing is trusted" (whole source is tainted)

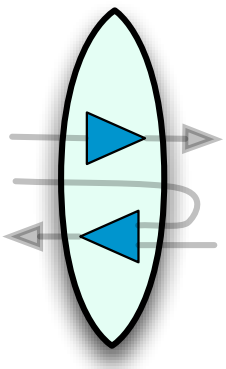
Effect: Arbitrary edits to the view are allowed; any hidden data in the source can be modified by **put**

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

Answer: Maybe! There are *many* alternatives, trading off which information in the source is trusted against which information in the view can be modified.

```
*08:30 Coffee with Sara (Starbucks)
12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



```
08:30 Group meeting
12:15 BUSY
```

Policy: "Nothing is trusted" (whole source is tainted)

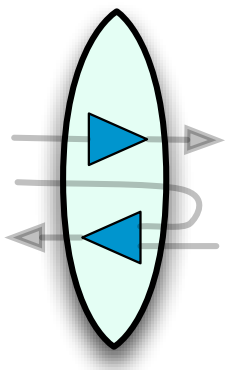
Effect: Arbitrary edits to the view are allowed; any hidden data in the source can be modified by **put**

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

Answer: Maybe! There are *many* alternatives, trading off which information in the source is trusted against which information in the view can be modified.

```
08:30 Group meeting (Unknown)
*12:15 Unknown (Unknown)
```



```
08:30 Group meeting
12:15 BUSY
```

Policy: "Nothing is trusted" (whole source is tainted)

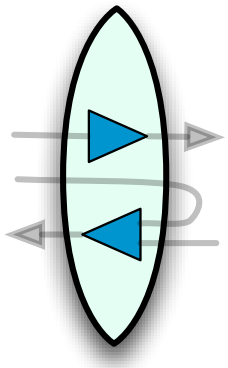
Effect: Arbitrary edits to the view are allowed; any hidden data in the source can be modified by **put**

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

Answer: Maybe! There are *many* alternatives, trading off which information in the source is trusted against which information in the view can be modified.

```
*08:30 Coffee with Sara (Starbucks)
12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

Policy: "Private events are trusted; public ones are tainted"

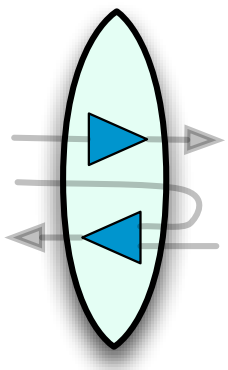
Effect: Okay to edit, add, and delete public events, but not to add or delete private ones, or change between public and private

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

Answer: Maybe! There are *many* alternatives, trading off which information in the source is trusted against which information in the view can be modified.

```
*08:30 Coffee with Sara (Starbucks)
12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 Lunch
15:00 BUSY
17:00 TGIF
```

Policy: "Private events are trusted; public ones are tainted"

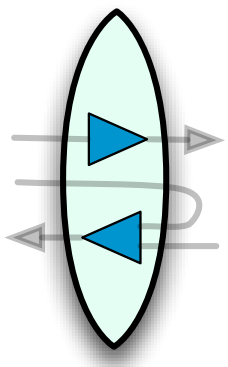
Effect: Okay to edit, add, and delete public events, but not to add or delete private ones, or change between public and private

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

Answer: Maybe! There are *many* alternatives, trading off which information in the source is trusted against which information in the view can be modified.

```
*08:30 Coffee with Sara (Starbucks)
12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
17:00 TGIF (Unknown)
```



```
08:30 BUSY
12:15 Lunch
15:00 BUSY
17:00 TGIF
```

Policy: "Private events are trusted; public ones are tainted"

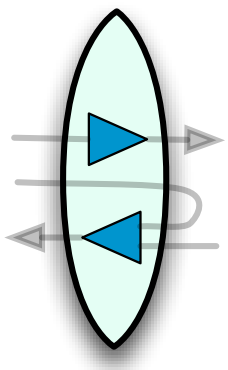
Effect: Okay to edit, add, and delete public events, but not to add or delete private ones, or change between public and private

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

Answer: Maybe! There are *many* alternatives, trading off which information in the source is trusted against which information in the view can be modified.

```
*08:30 Coffee with Sara (Starbucks)
12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



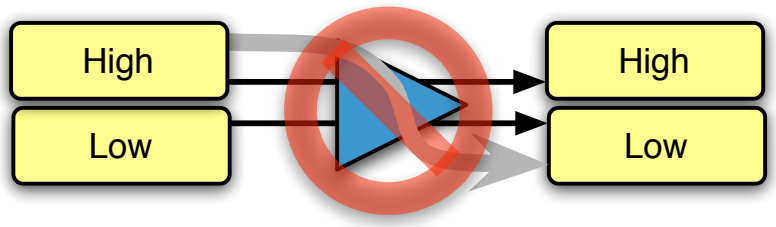
```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

Policy: "Everything is trusted"

Effect: No edits are allowed

Non-interference

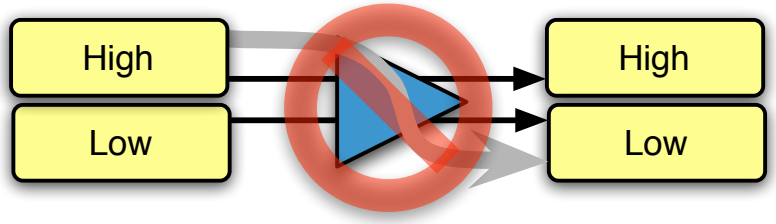
All these policies can be formulated in terms of non-interference.



A transformation is non-interfering if the "low" parts of the output do not depend on the "high" parts of the input.

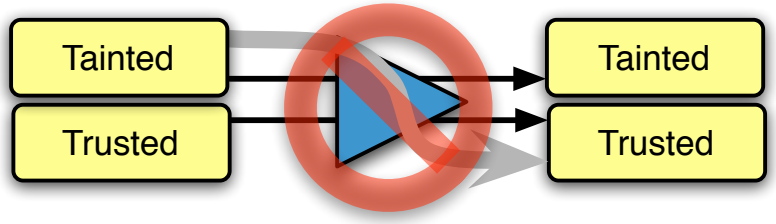
Non-interference — Integrity

All these policies can be formulated in terms of non-interference.



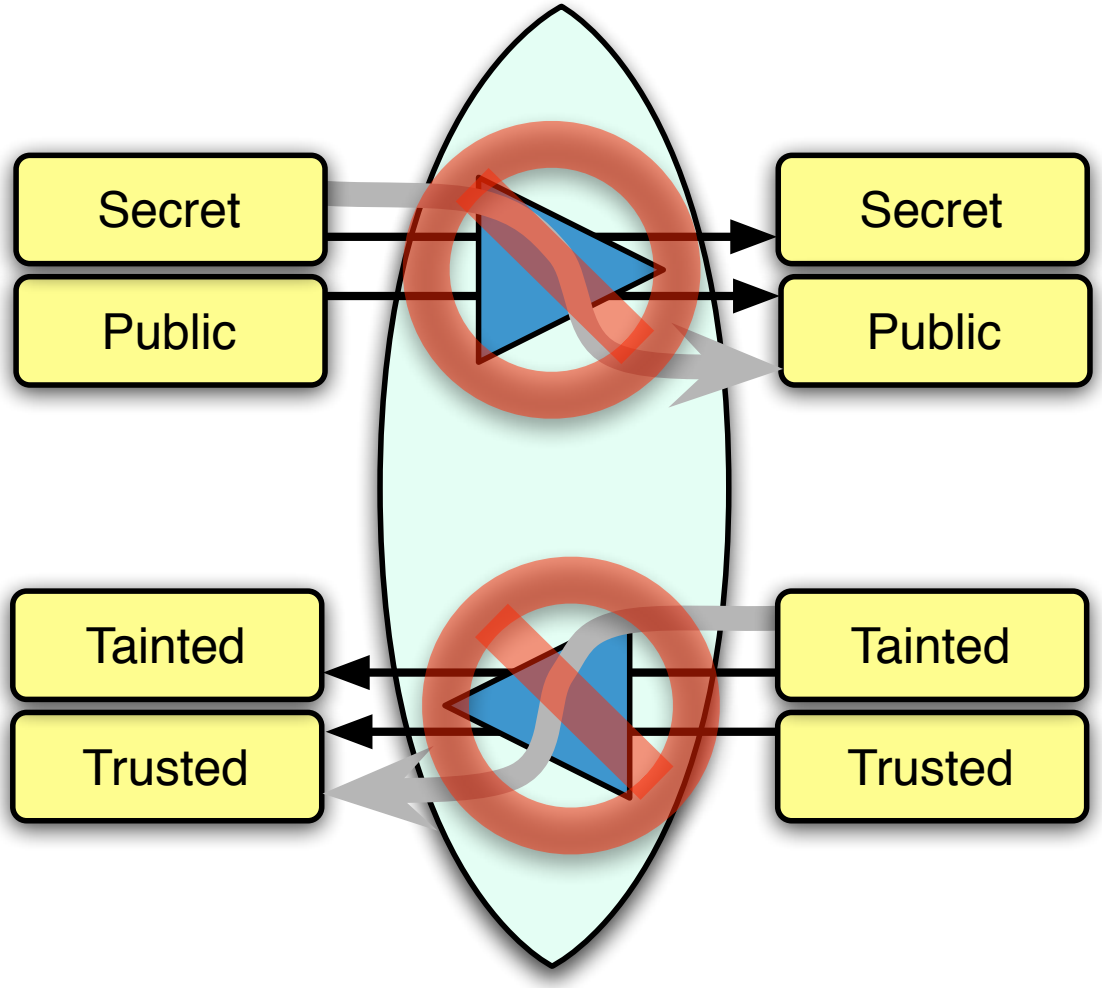
A transformation is non-interfering if the "low" parts of the output do not depend on the "high" parts of the input.

E.g., if the data contains "tainted" and "trusted" portions



then the trusted parts of the output do not depend on the tainted parts of the input.

Secure Lenses



Labels

Fix a lattice $\mathcal{Q}$ of integrity labels, e.g.

Annotated Regular Expressions

Annotate the source and view types with labels to indicate which parts each are *Tainted* and which are *Trusted*.

$$\mathcal{R} ::= \emptyset \mid u \mid \mathcal{R}\cdot\mathcal{R} \mid \mathcal{R}|\mathcal{R} \mid \mathcal{R}^* \mid \mathcal{R} : q$$

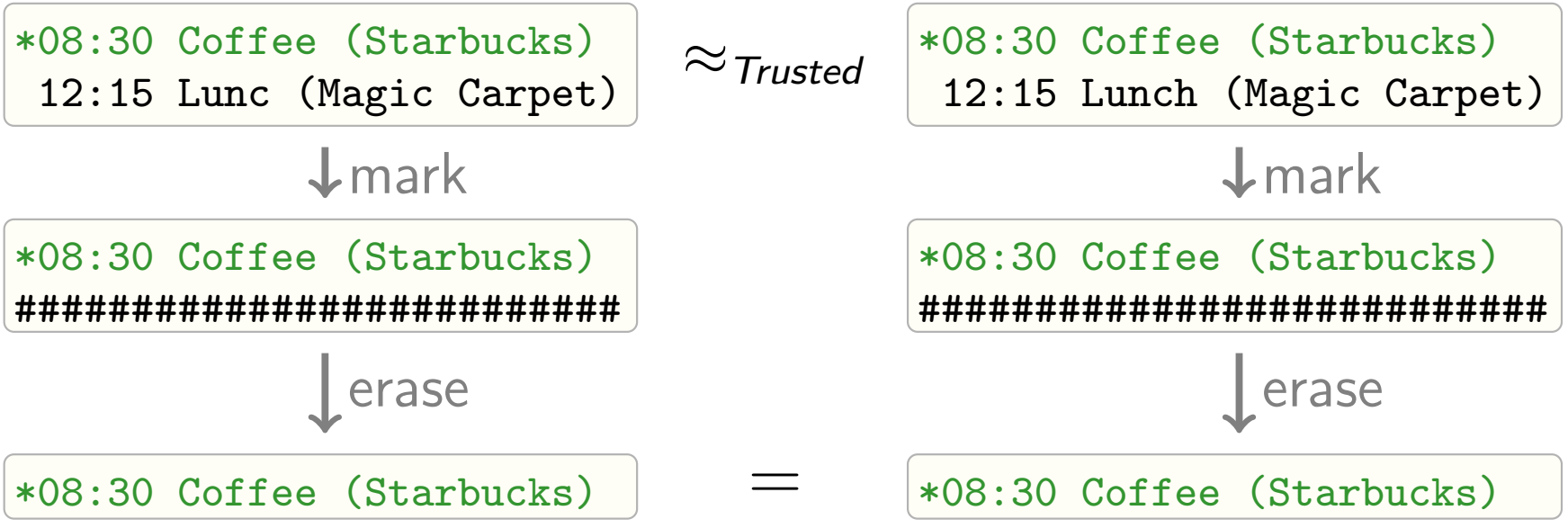
Read off an equivalence relation $\approx_q$ for every $q \in \mathcal{Q}$.

Annotated Regular Expressions

Annotate the source and view types with labels to indicate which parts each are *Tainted* and which are *Trusted*.

$$\mathcal{R} ::= \emptyset \mid u \mid \mathcal{R}\cdot\mathcal{R} \mid \mathcal{R}|\mathcal{R} \mid \mathcal{R}^* \mid \mathcal{R} : q$$

Read off an equivalence relation $\approx_q$ for every $q \in \mathcal{Q}$.



Secure Lenses, Formally

The expectation that "*Tainted* inputs to **put** should not affect *Trusted* outputs" can now be expressed by generalizing the GetPut law...

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

... like this:

$$\frac{v \approx_q (l.\mathbf{get}\ s)}{l.\mathbf{put}\ v\ s \approx_q s} \qquad \text{(GetPutSecure)}$$

To guarantee this law, we refine the typing rules for lenses with an information-flow analysis.

The PutPut Law

The following law can be derived:

$$\frac{v' \approx_q v \approx_q (l.\mathbf{get}\ s)}{l.\mathbf{put}\ v'\ (l.\mathbf{put}\ v\ s) \approx_q l.\mathbf{put}\ v'\ s}$$

It says that doing two **put**s in a row must produce the same result as just the second.

It implies that the **put** function must not have "side-effects" on trusted source data...

...and generalizes the "constant complement" condition, the gold standard for correct view update in databases.

Conclusion

Summary

"Bidirectional programming languages are an effective and elegant means of describing updatable views"

Lenses

- Semantic space of well-behaved bidirectional transformations
- Provides foundation for bidirectional languages

Boomerang

- Language for lenses on strings
- Natural syntax based on regular operators
- Extensions to handle ordered, ignorable, and trusted data
- Type system guarantees well-behavedness and totality

Implementation and Applications

- Lenses for a number of real-world formats
- Adoption in Augeas
- Updatable security views

Thank You!

Adviser: Benjamin Pierce

Informal Mentor: Alan Schmitt

Committee: Zack Ives, Val Tannen, Phil Wadler (external), Steve Zdancewic (chair)

Collaborators: Davi Barbosa, Aaron Bohannon, Ravi Chugh, Julien Cretin, Malo Deniélou, Michael Greenberg, Michael Greenwald, Christian Kirkegaard, Stéphane Lescuyer, Adam Magee, Jon Moore, Alexandre Pilkiewicz, Danny Puller