Impersonators and Chaperones:
Run-time Support for Higher-Order Contracts

T. Stephen Strickland
Northeastern University

Sam Tobin-Hochstadt
Northeastern University

Robert Bruce Findler
Northwestern University

Matthew Flatt
University of Utah

... This Talk ...

- It's about an **untyped** language

  ... in practice, though not in principle

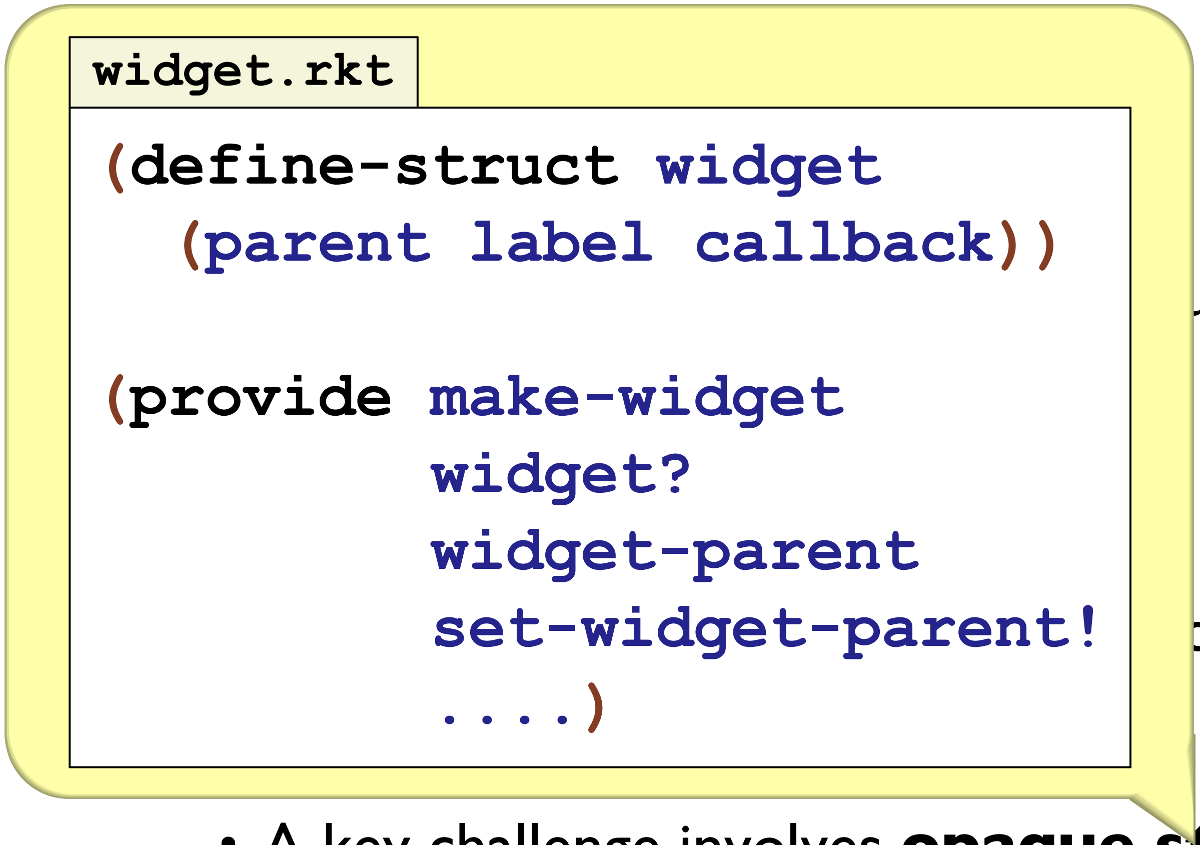
- It's about an **extensibile** language

  ... in the sense of implementing contracts as a layer

- A key challenge involves **opaque structures**

  ... not in Javascript, Ruby, or Python

... This Talk ...



widget.rkt

```
(define-struct widget
  (parent label callback))

(provide make-widget
         widget?
         widget-parent
         set-widget-parent!
         ....)
```

ugh not in principle

contracts as a layer

- A key challenge involves **opaque structures**

... not in Javascript, Ruby, or Python

... This Talk ...

- It’s about an **untyped** language

  ... in practice, though not in principle

- It’s about an **extensibile** language

  ... in the sense of implementing contracts as a layer

- A key challenge involves **opaque structures**

  ... not in Javascript, Ruby, or Python

Why This Talk is Relatively Racket-Specific

- It's about an **untyped** language

  ... in practice, though not in principle

- It's about an **extensibile** language

  ... in the sense of implementing contracts as a layer

- A key challenge involves **opaque structures**

  ... not in Javascript, Ruby, or Python

Contracts

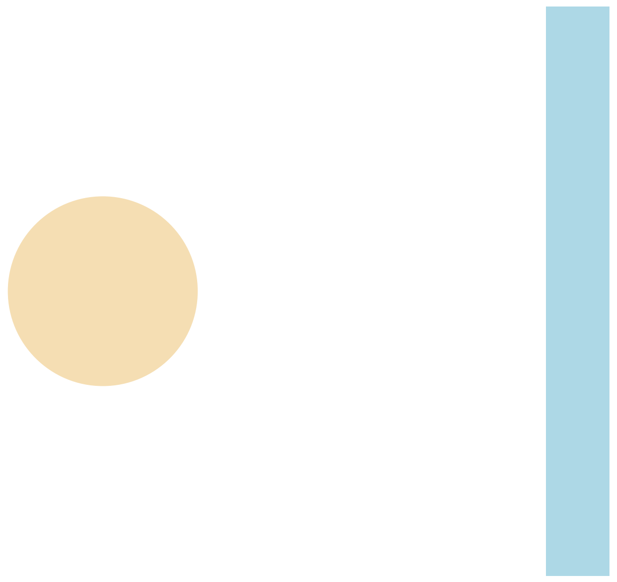
```
make-gauge : nonnegative-integer? -> widget?

make-choice : (listof label-string?) -> widget?

make-button : label-string?
              (button-event? -> status?)
              -> widget?
```

Contracts

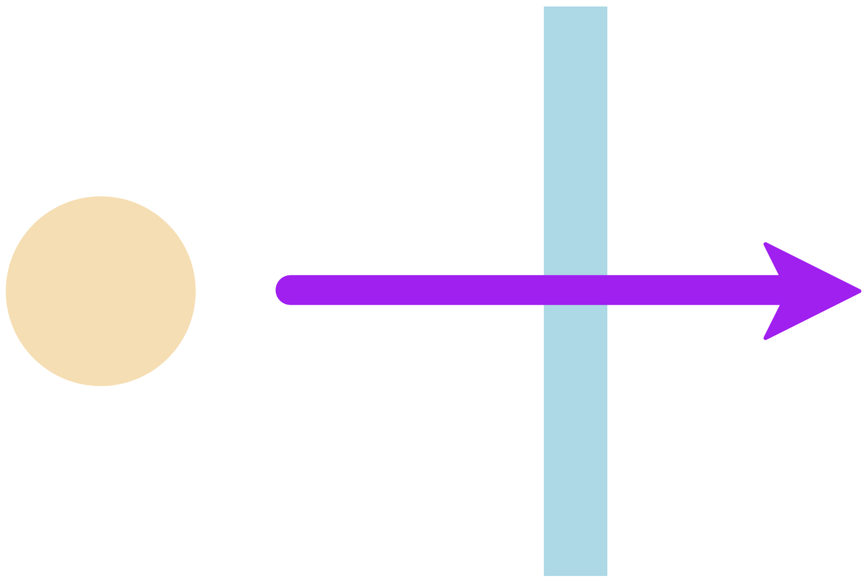
```
make-gauge : nonnegative-integer? -> widget?

make-choice : (listof label-string?) -> widget?

make-button : label-string?
              (button-event? -> status?)
              -> widget?
```

- Some contracts are not types
- Some contracts guard a typed–untyped boundary

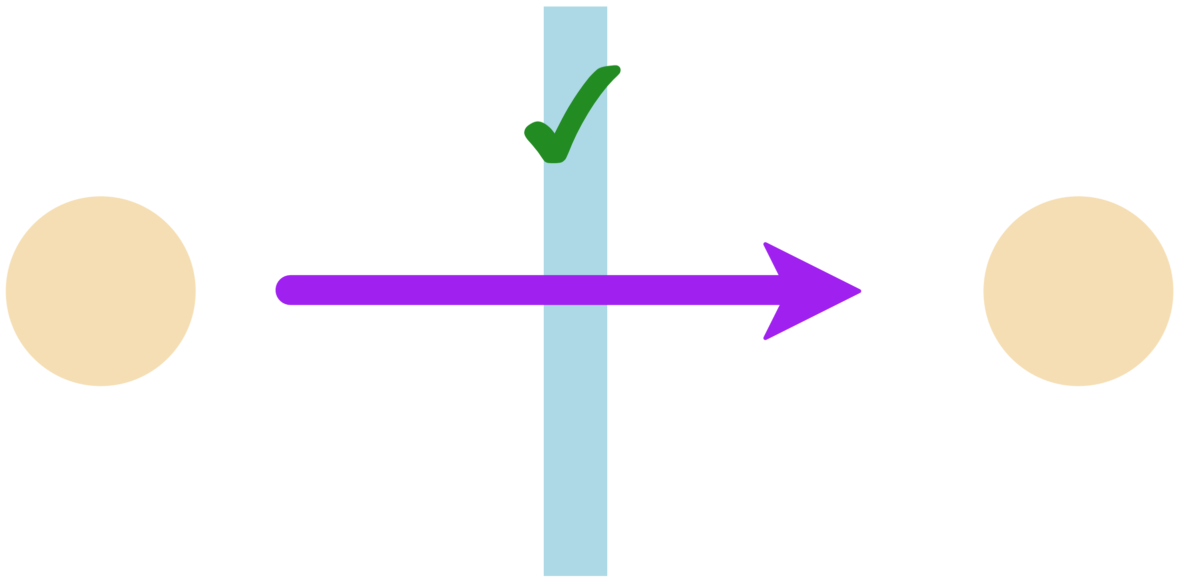
Contracts and Boundaries



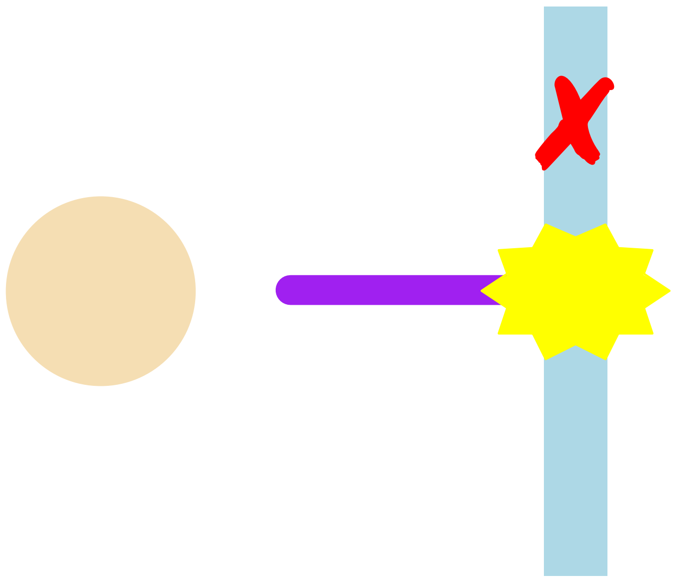
Contracts and Boundaries



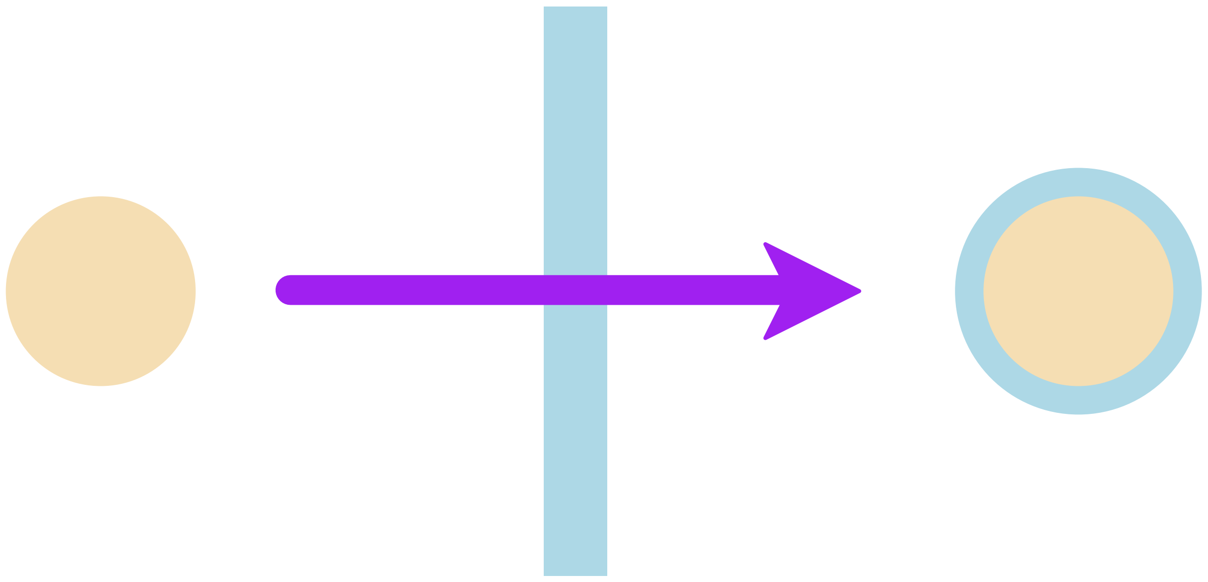
Contracts and Boundaries



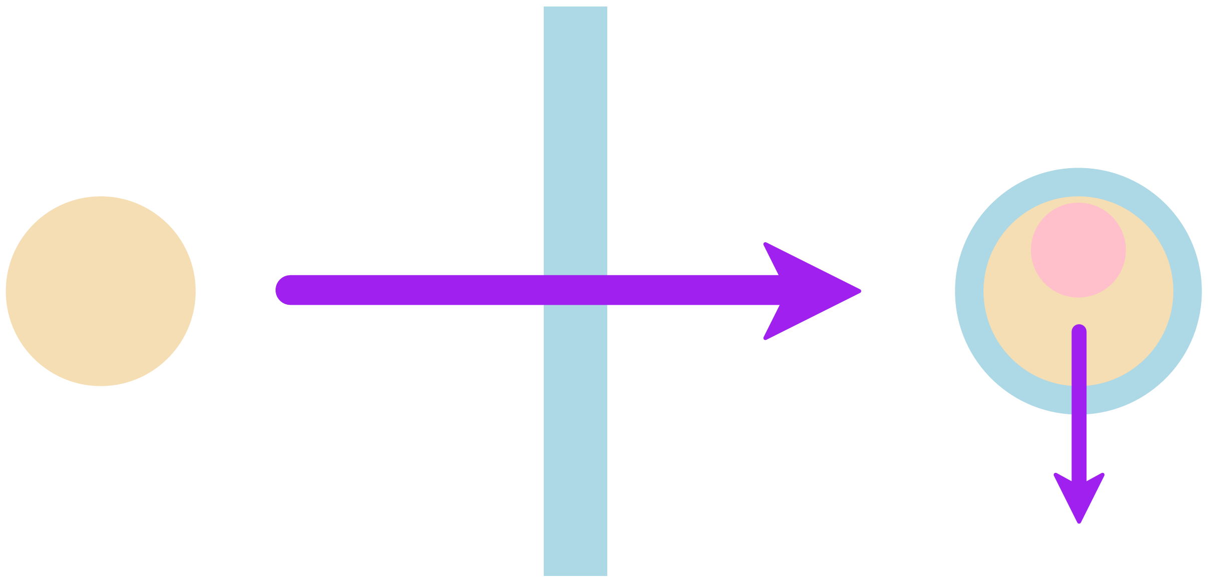
Contracts and Boundaries



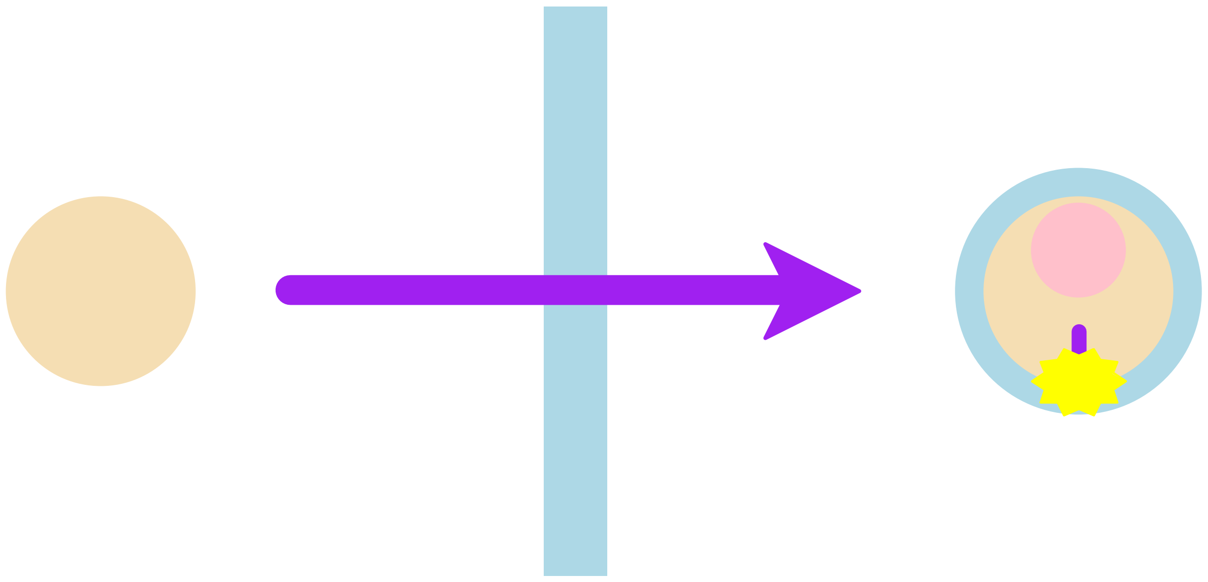
Contracts and Boundaries



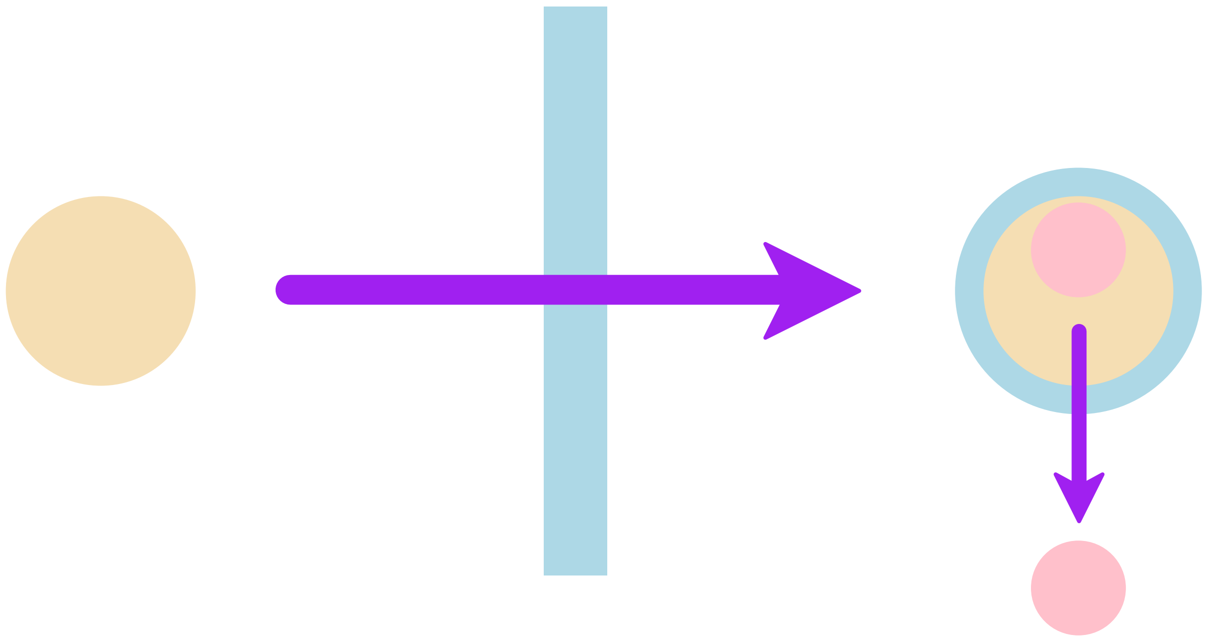
Contracts and Boundaries



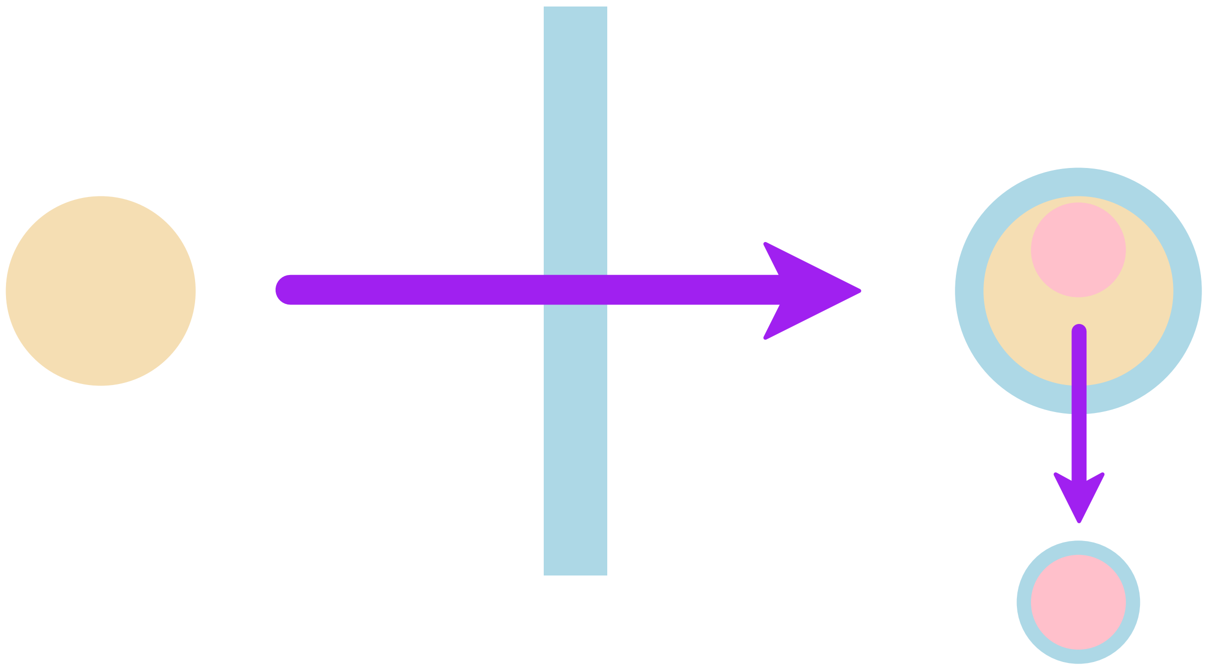
Contracts and Boundaries



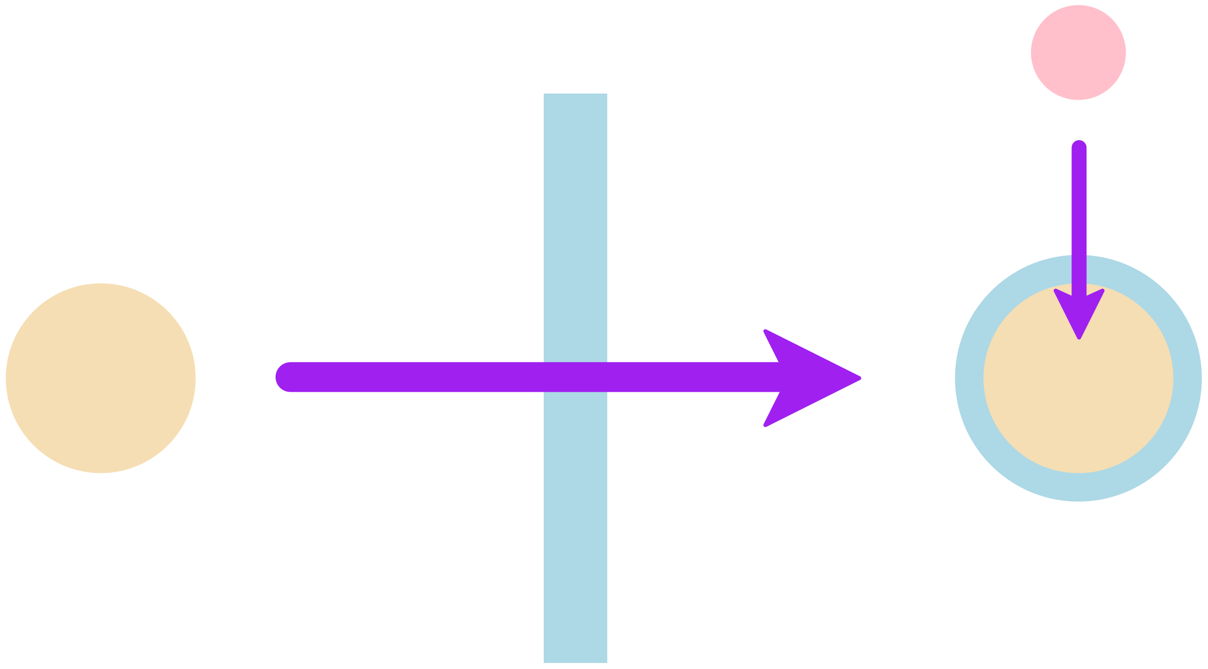
Contracts and Boundaries



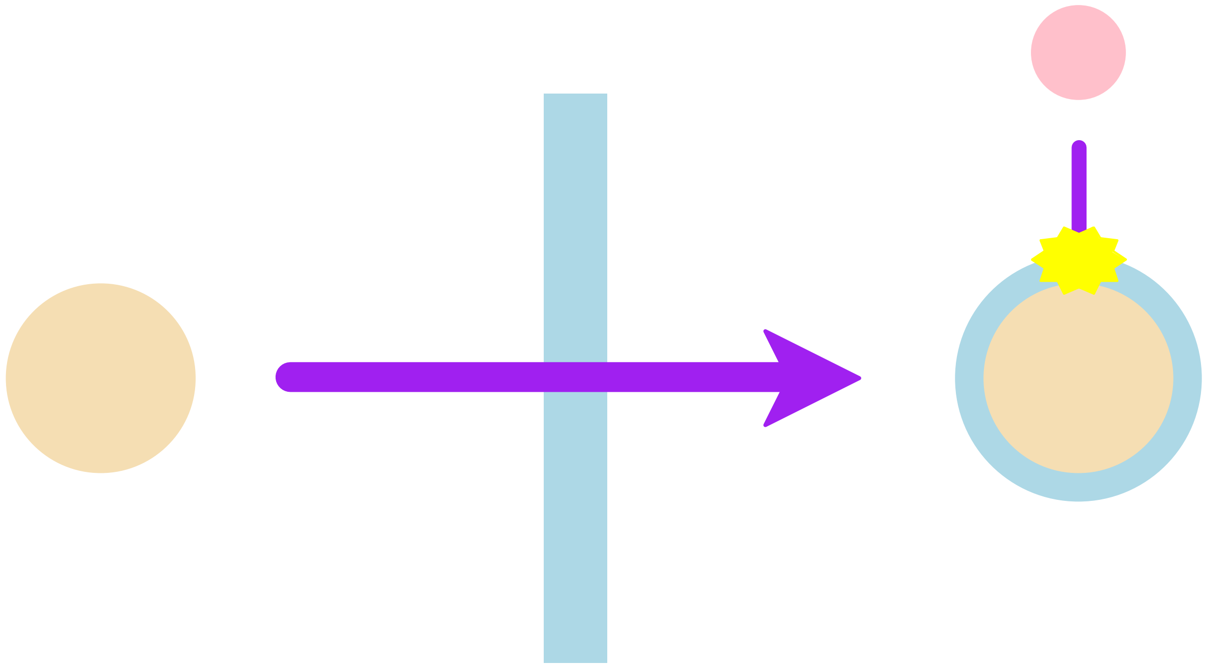
Contracts and Boundaries



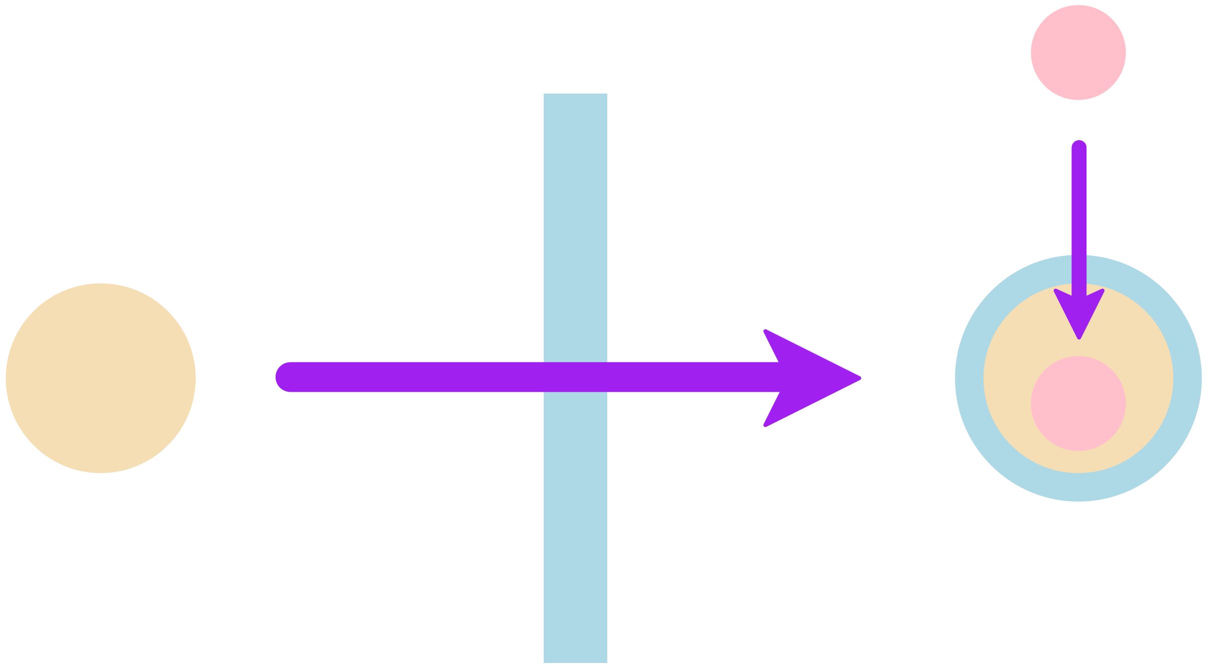
Contracts and Boundaries



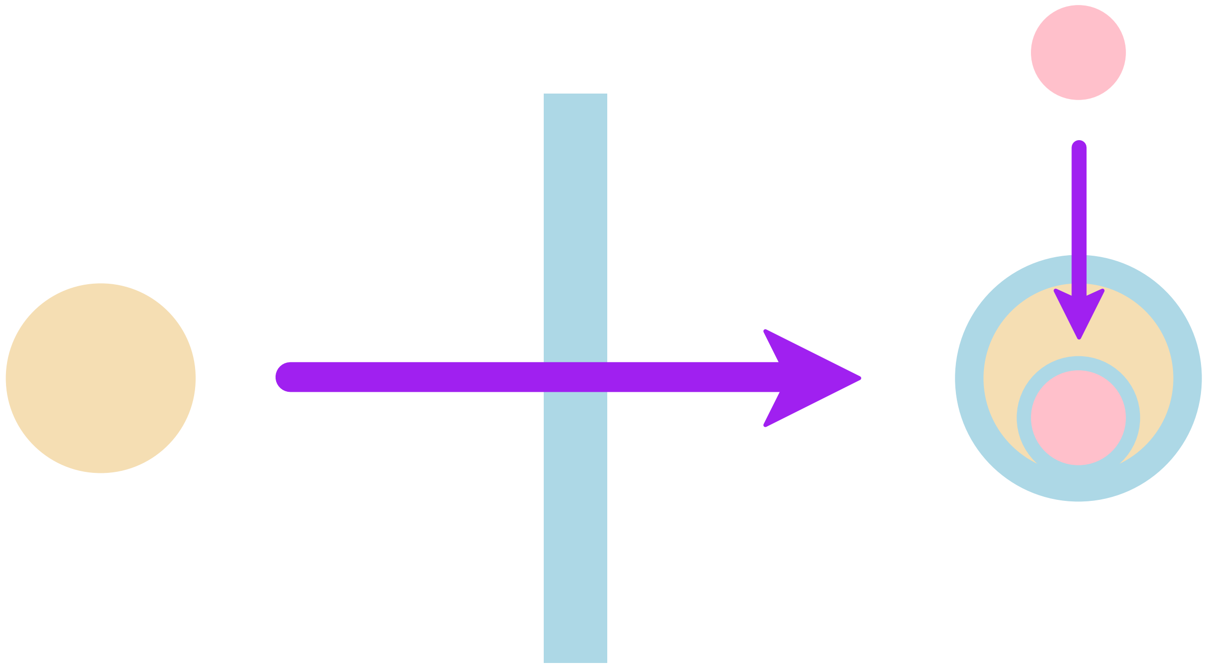
Contracts and Boundaries



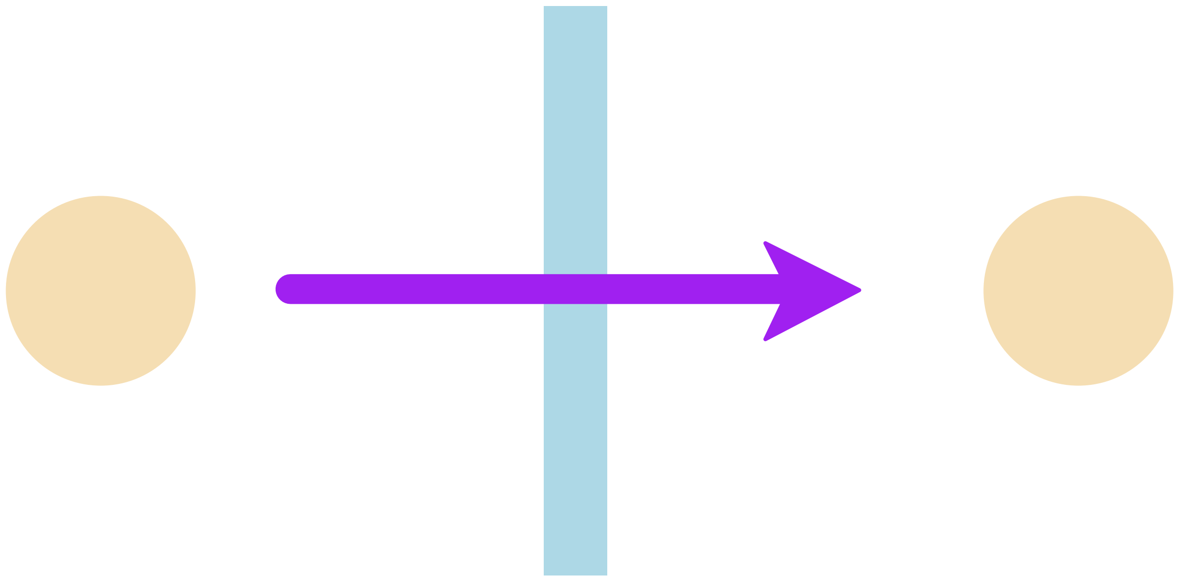
Contracts and Boundaries



Contracts and Boundaries



Contract Examples



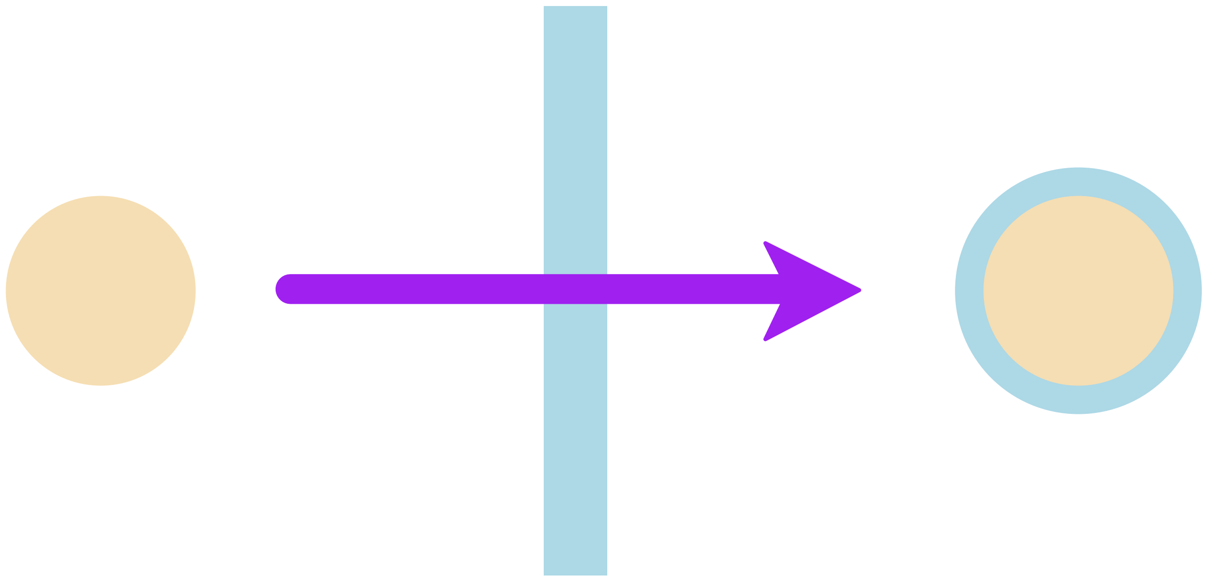
math.rkt

```
(define pi 3.14)

(provide/contract
 [pi real?])
```

circles.rkt

```
(require "math.rkt")

pi
```

Contract Examples



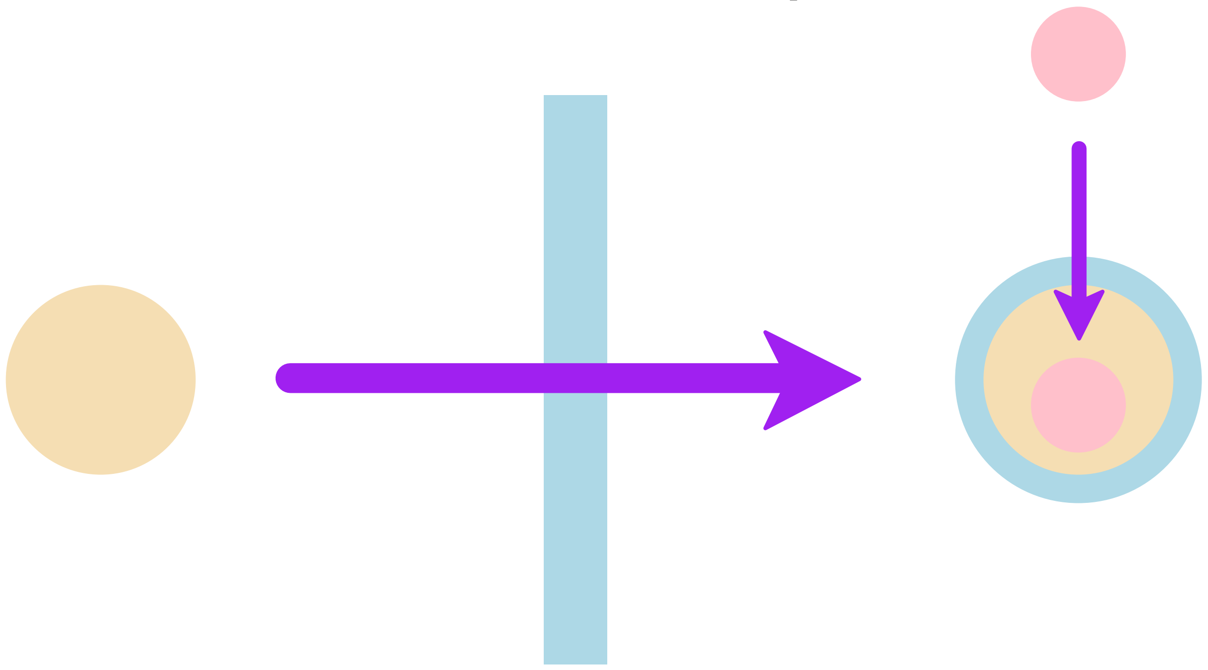
math.rkt

```
(define (sqr x) (* x x))

(provide/contract
 [sqr
  (real? . -> .
           nonnegative-real?)])
```

circles.rkt

```
(require "math.rkt")

(map sqr ....)
```

Contract Examples



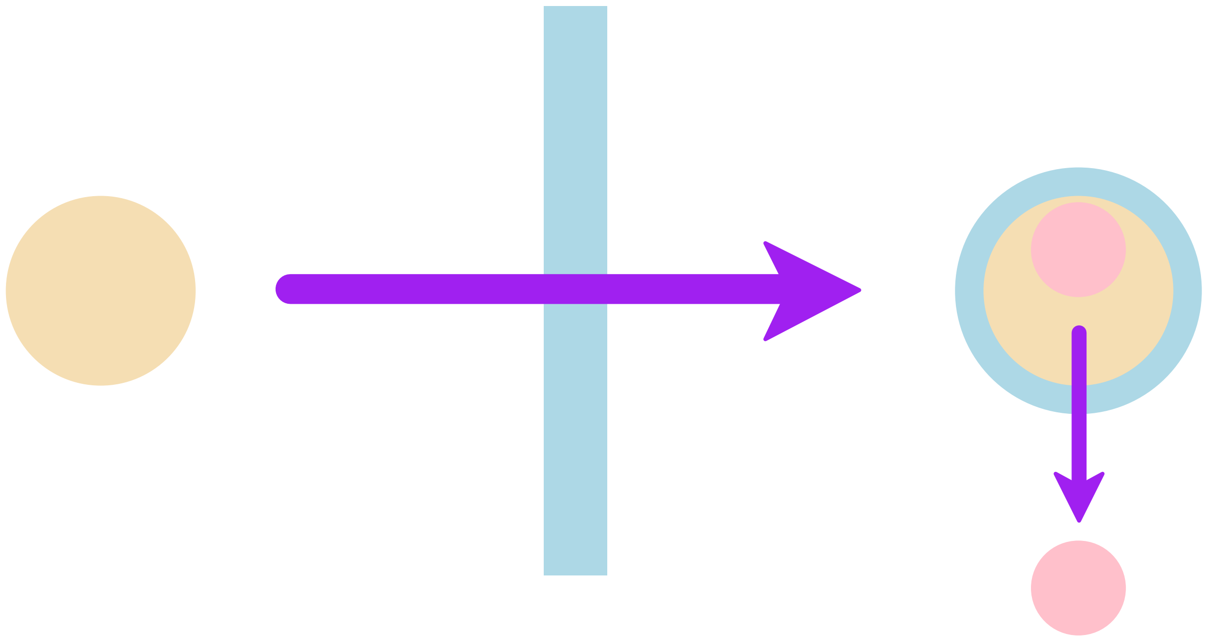
math.rkt

```
(define (sqr x) (* x x))

(provide/contract
 [sqr
  (real? . -> .
             nonnegative-real?)])
```

circles.rkt

```
(require "math.rkt")

(sqr 1.414)
```

Contract Examples



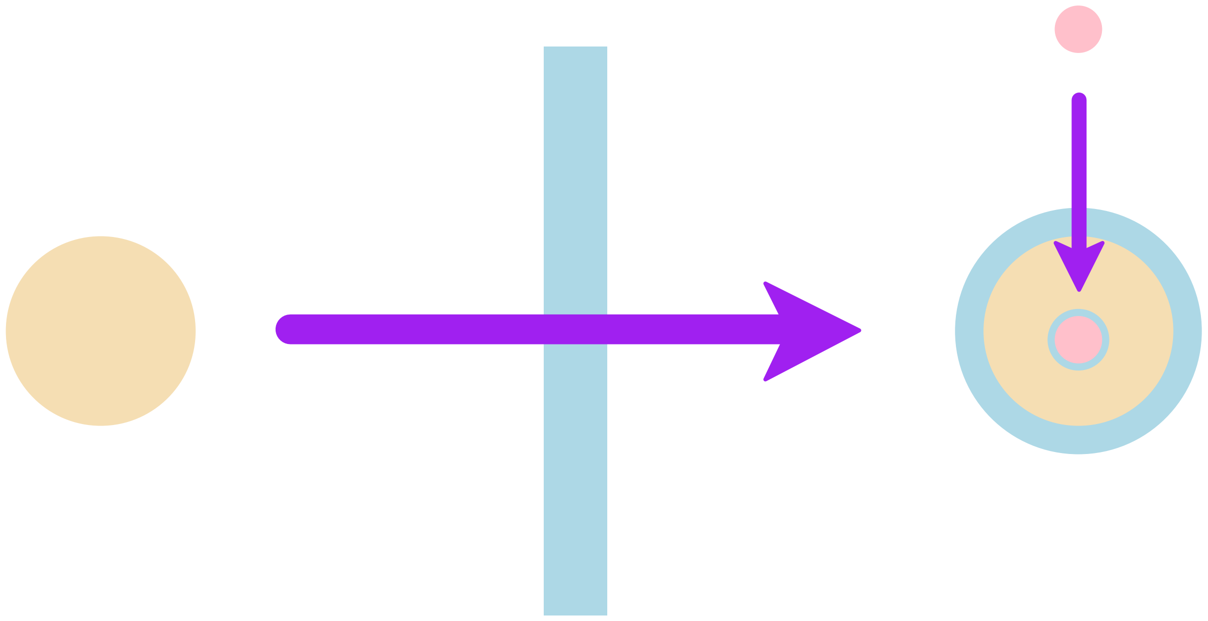
math.rkt

```
(define (sqr x) (* x x))

(provide/contract
 [sqr
  (real? . -> .
           nonnegative-real?)])
```

circles.rkt

```
(require "math.rkt")

(sqr 1.414)
```

Contract Examples



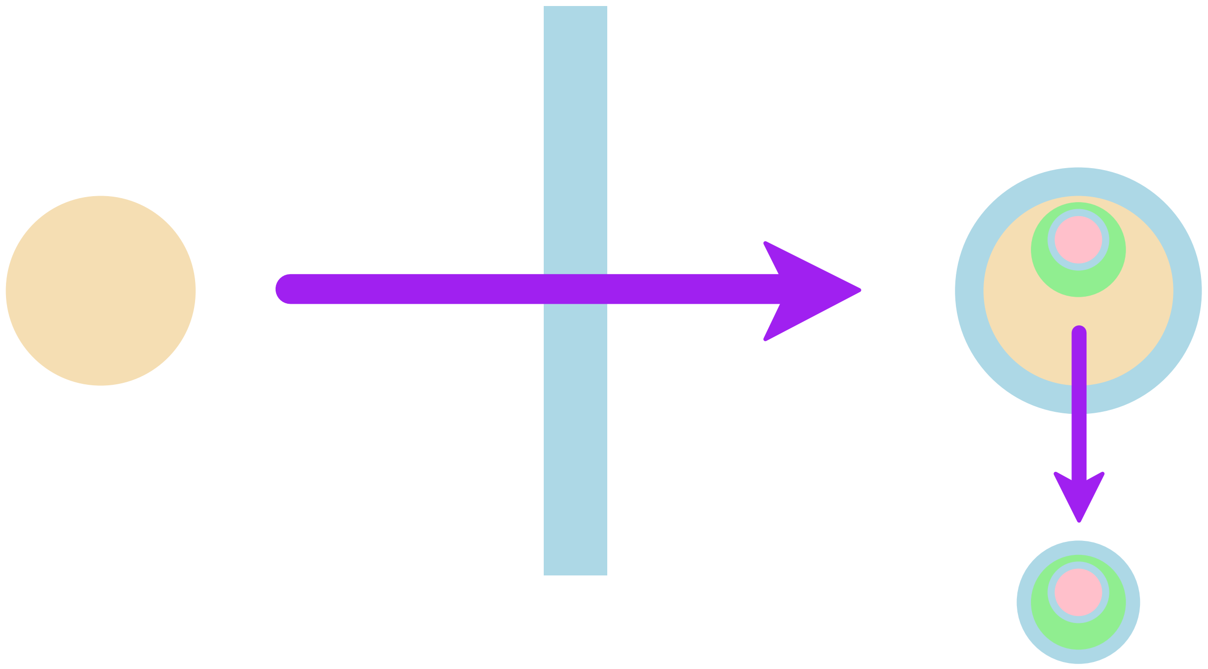
math.rkt

```
(define (derivative f)
  ....)

(provide/contract
 [derivative
  ((real? . -> . real?)
   . -> .
   (real? . -> . real?))])
```

circles.rkt

```
(require "math.rkt")

(derivative cos)
```

Contract Examples



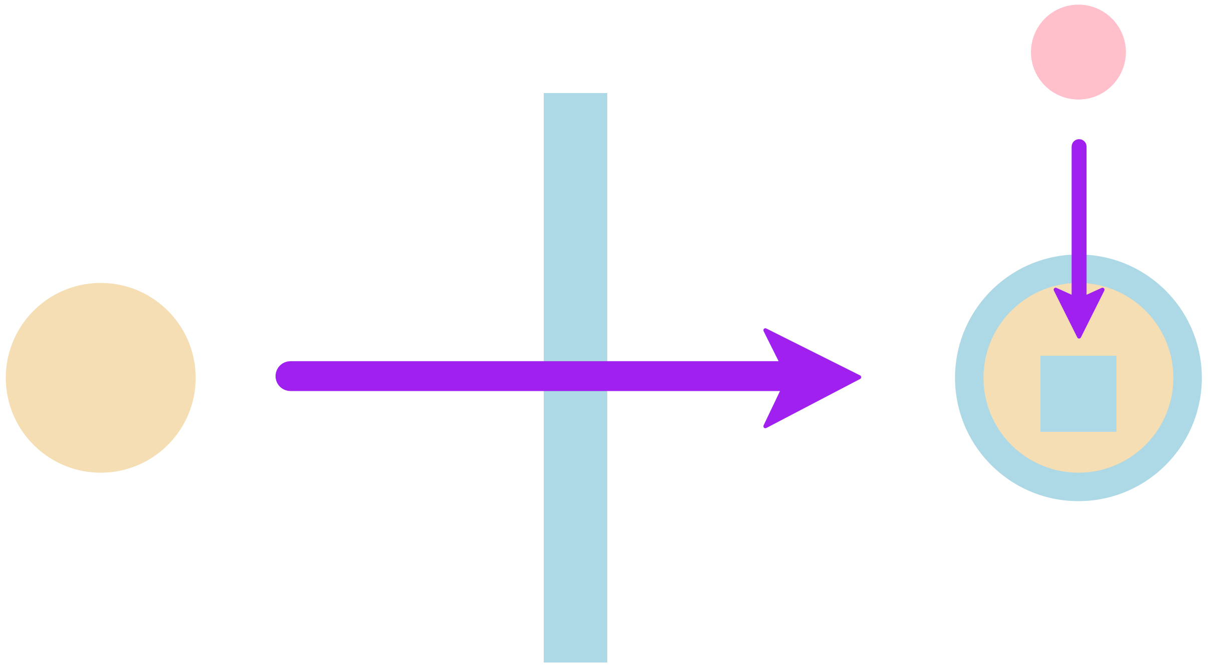
math.rkt

```
(define (derivative f)
  ....)

(provide/contract
 [derivative
  ((real? . -> . real?)
   . -> .
   (real? . -> . real?))])
```

circles.rkt

```
(require "math.rkt")

(derivative cos)
```

Contract Examples



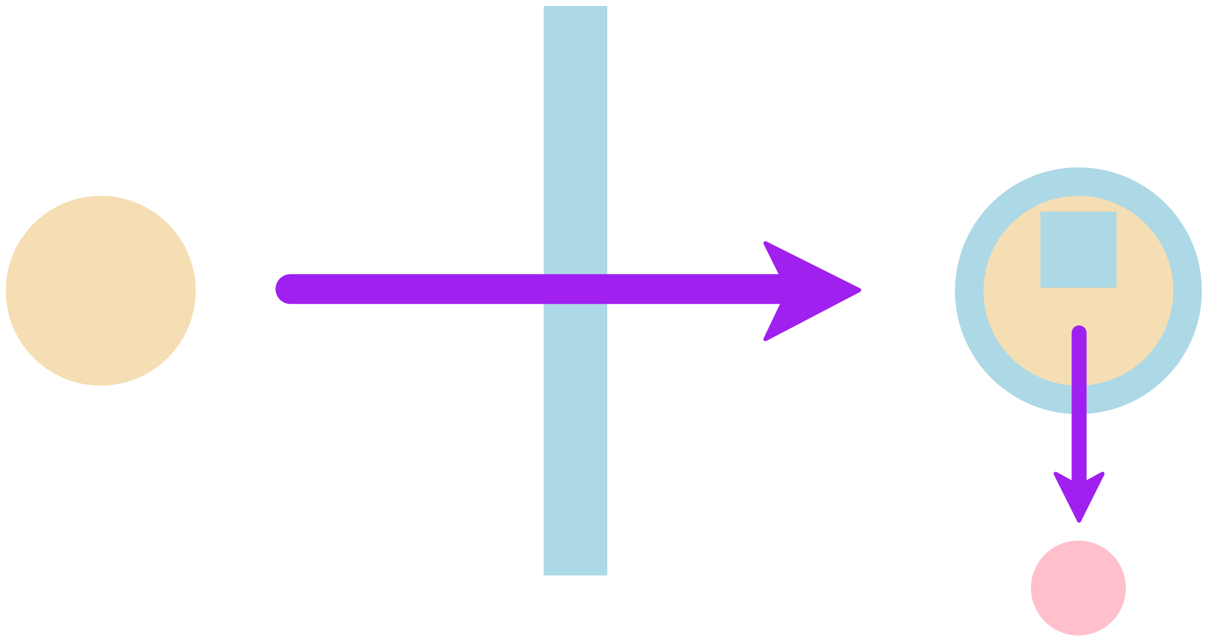
vault.rkt

```
(define α (new-∀/c))
....

(provide/contract
 [put (α . -> . any)]
 [get (-> α)])
```

bank.rkt

```
(require "vault.rkt")

(put 199.99)
(get)
```

Contract Examples



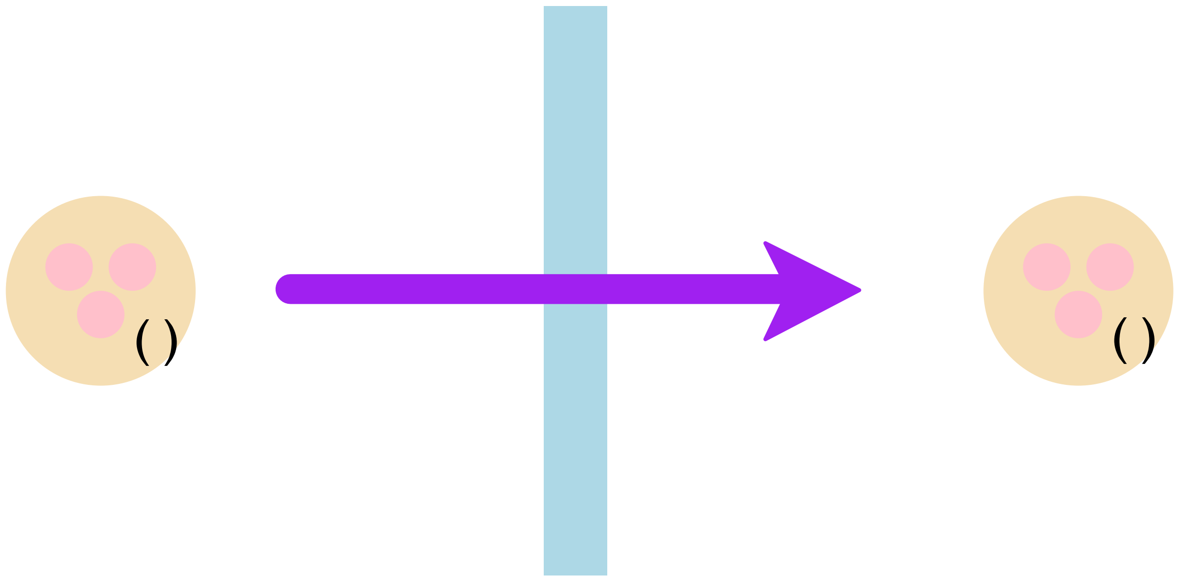
vault.rkt

```
(define α (new-∀/c))
....

(provide/contract
 [put (α . -> . any)]
 [get (-> α)])
```

bank.rkt

```
(require "vault.rkt")

(put 199.99)
(get)
```

Contract Examples



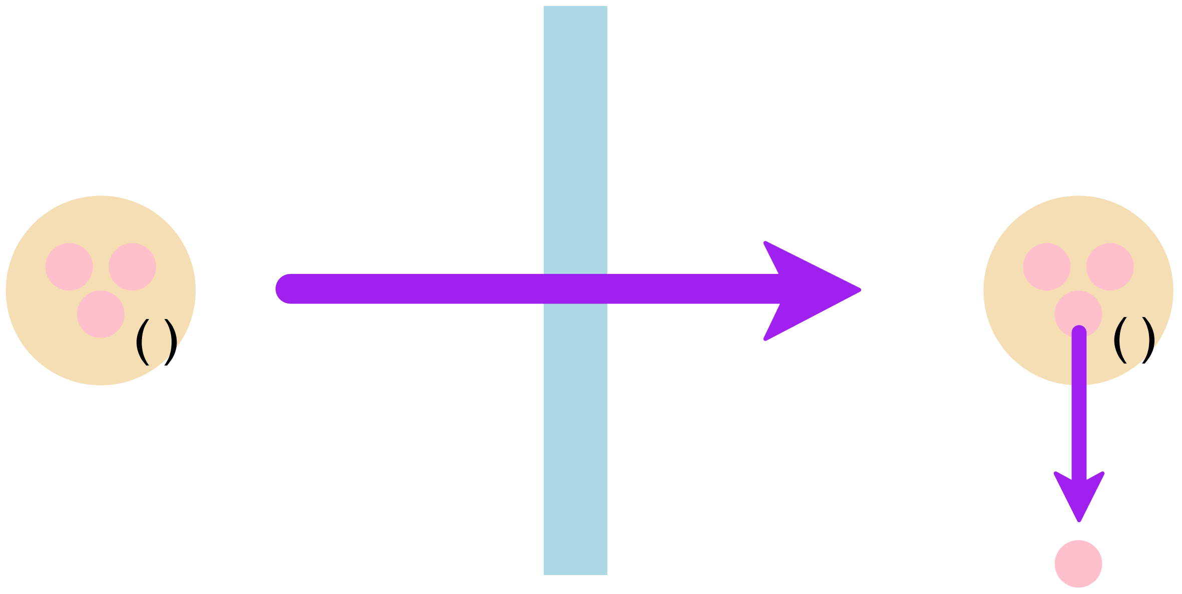
math.rkt

```
(define constants
  (list 299000000.0
        6.67e-11
        6.63e-34))

(provide/contract
 [constants
  (listof nonnegative-real?)])
```

circles.rkt

```
(require "math.rkt")

constants
```

Contract Examples



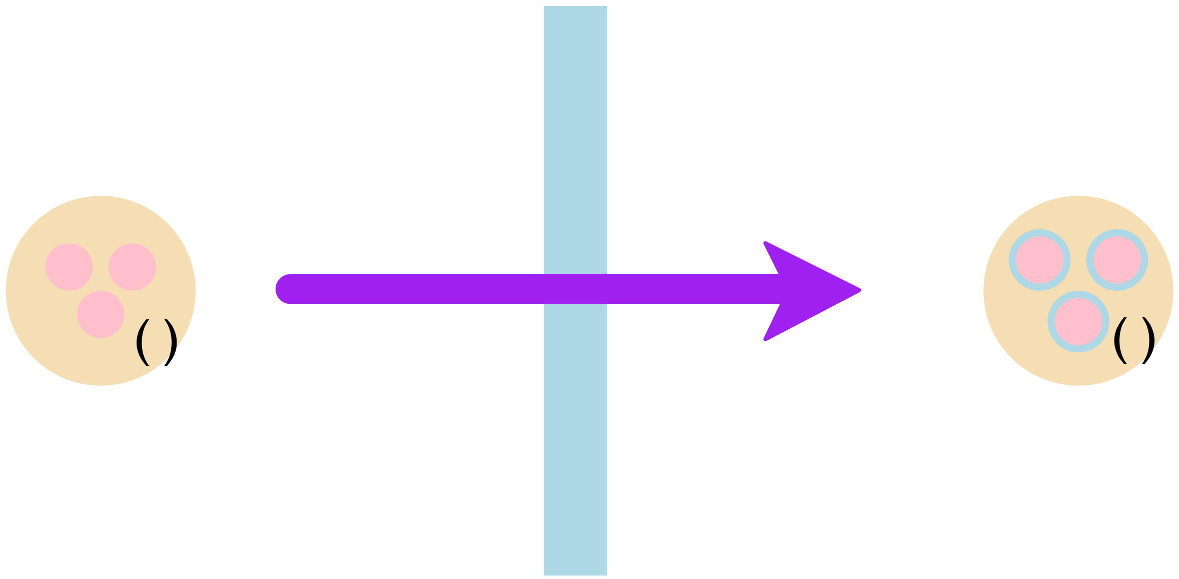
math.rkt

```
(define constants
  (list 299000000.0
        6.67e-11
        6.63e-34))

(provide/contract
 [constants
  (listof nonnegative-real?)])
```

circles.rkt

```
(require "math.rkt")

(first constants)
```

Contract Examples



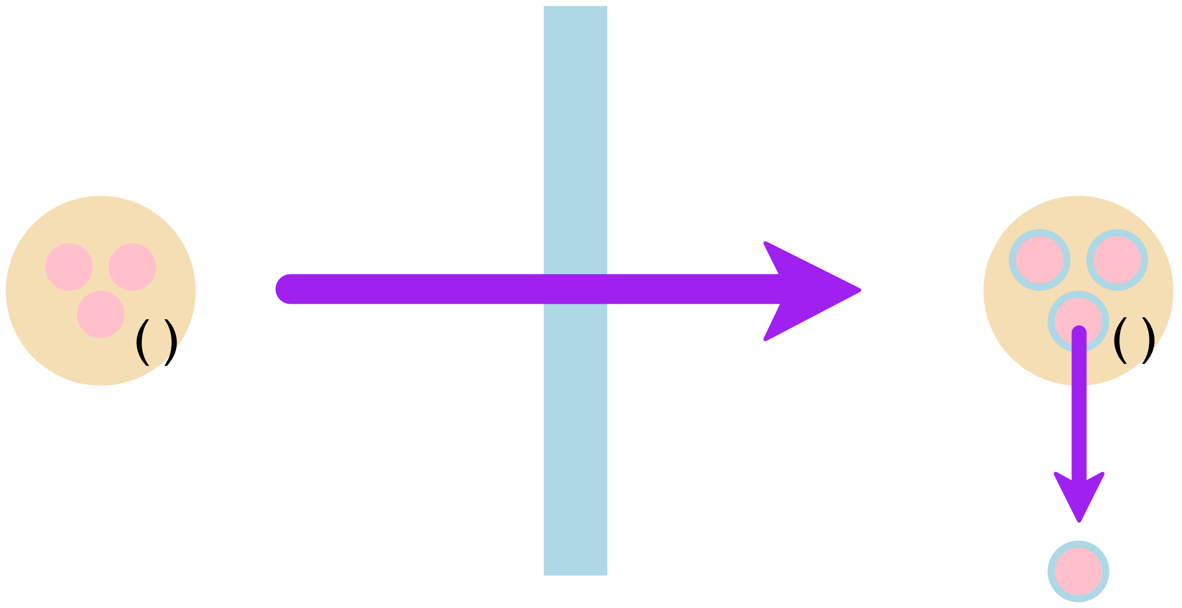
math.rkt

```
(define transforms
  (list identity sqr sqrt))

(provide/contract
 [constants
  (listof
   (nonnegative-real?
    . -> . nonnegative-real?))])
```

circles.rkt

```
(require "math.rkt")

transforms
```

Contract Examples



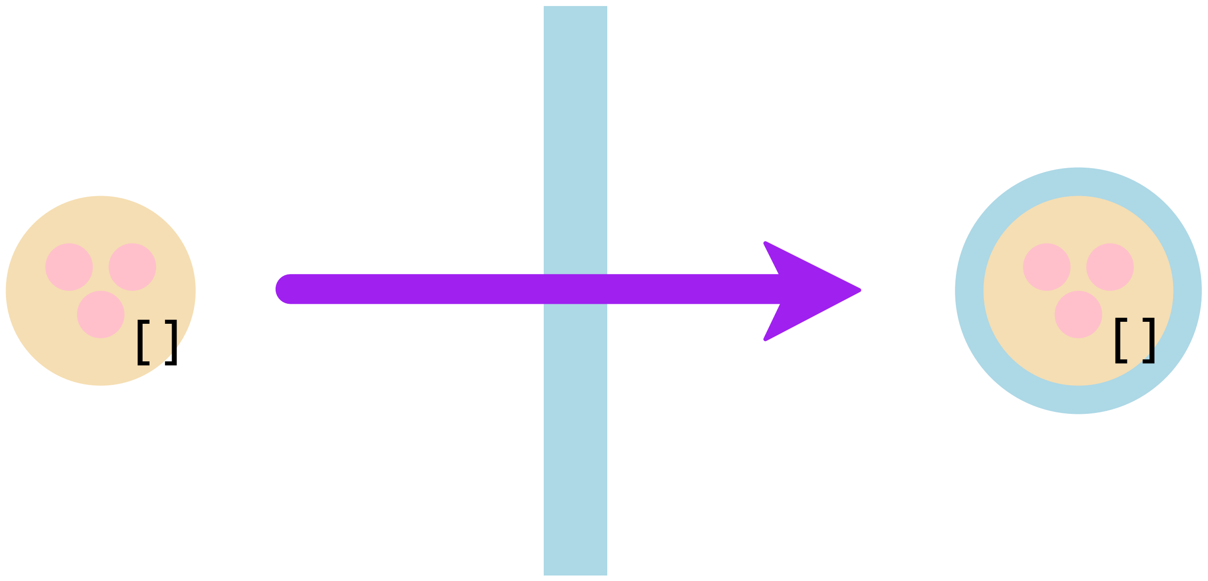
math.rkt

```
(define transforms
  (list identity sqr sqrt))

(provide/contract
 [constants
  (listof
   (nonnegative-real?
    . -> . nonnegative-real?))])
```

circles.rkt

```
(require "math.rkt")

(first transforms)
```

Contract Examples



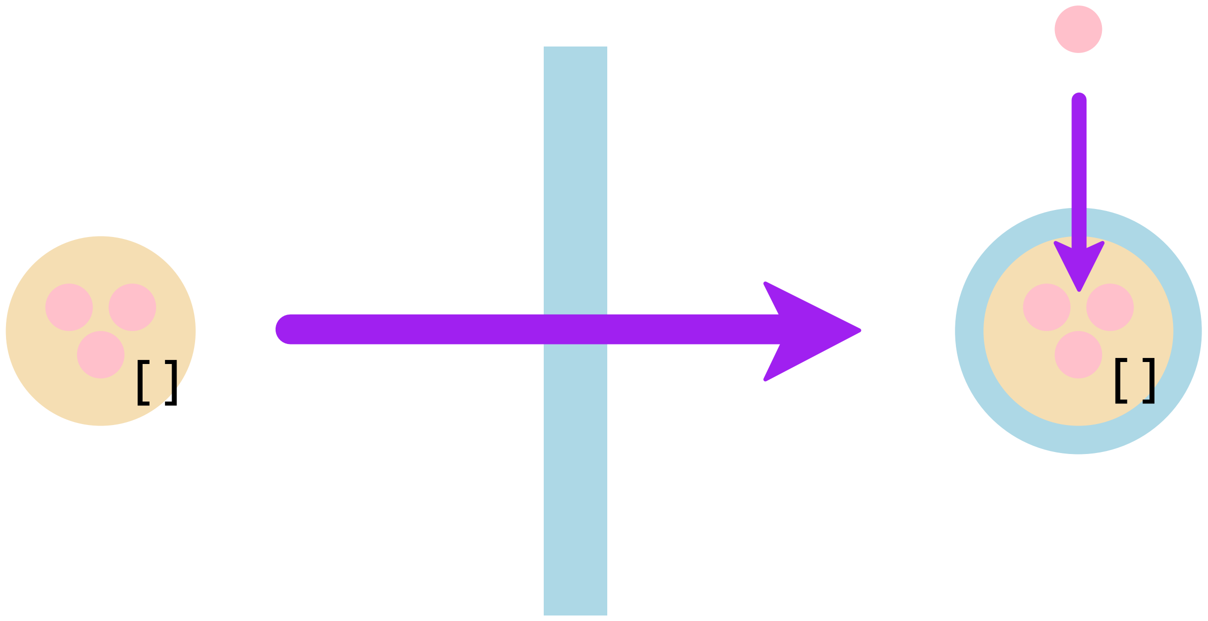
math.rkt

```
(define state
  (vector 0.1
          0.4
          7.9))

(provide/contract
 [state
  nonnegative-real?])
```

circles.rkt

```
(require "math.rkt")

state
```

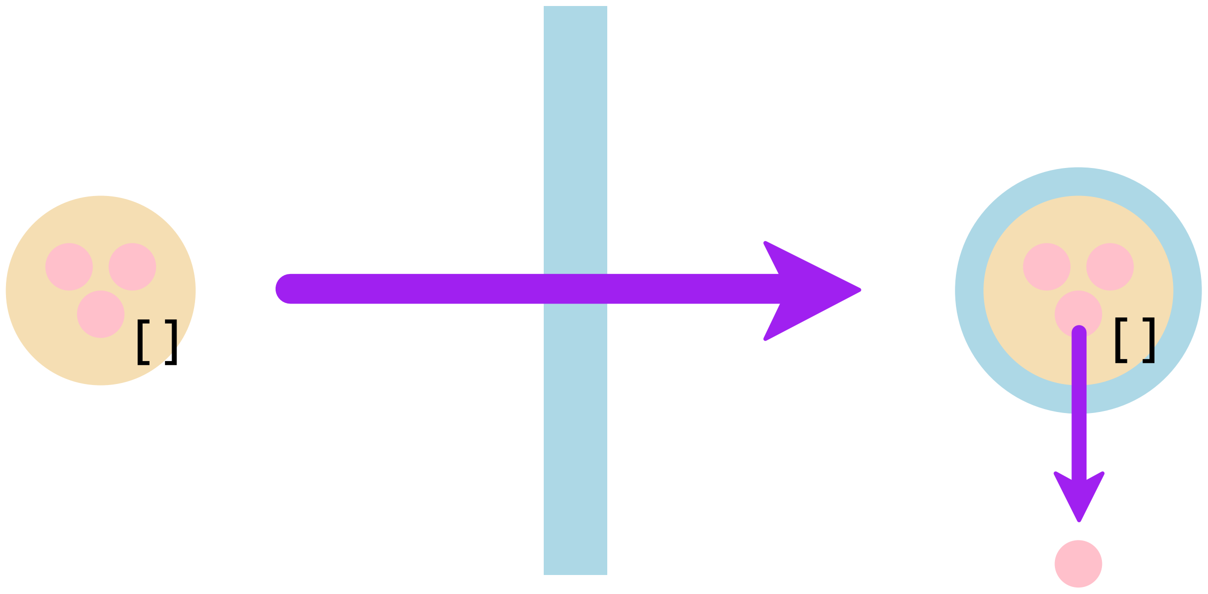
Contract Examples



math.rkt
```
(define state
  (vector 0.1
          0.4
          7.9))

(provide/contract
 [state
  nonnegative-real?])
```

circles.rkt
```
(require "math.rkt")

(vector-set! state 0 0.5)
```

Contract Examples



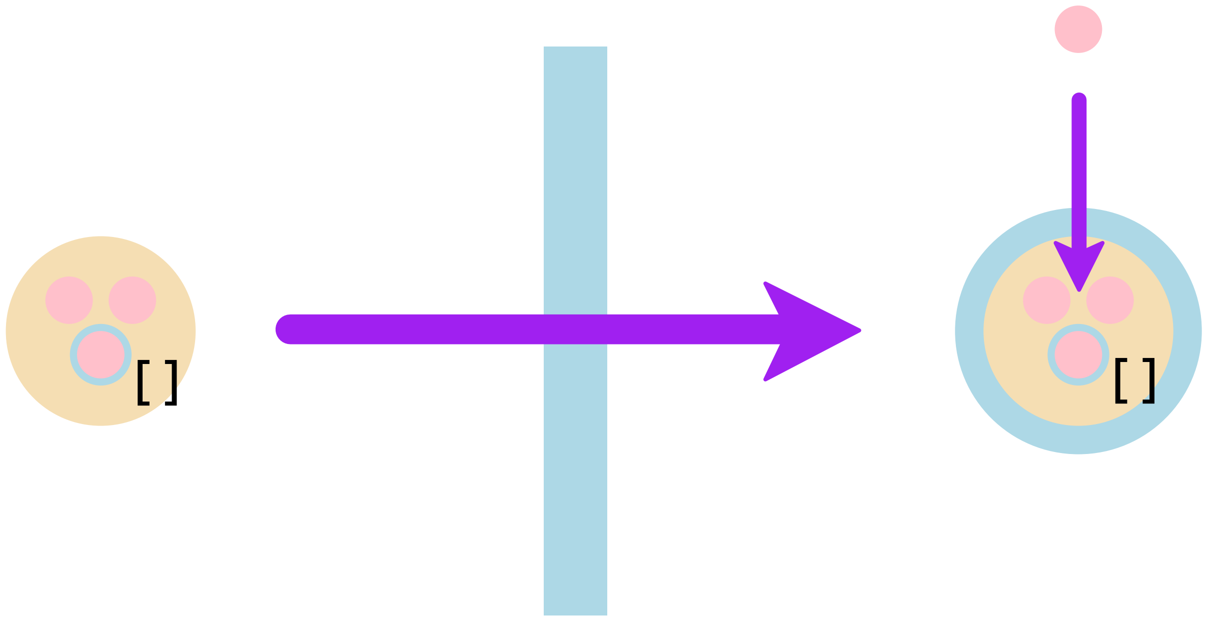
math.rkt

```
(define state
  (vector 0.1
          0.4
          7.9))

(provide/contract
 [state
  nonnegative-real?])
```

circles.rkt

```
(require "math.rkt")

(vector-ref state 0)
```

Contract Examples



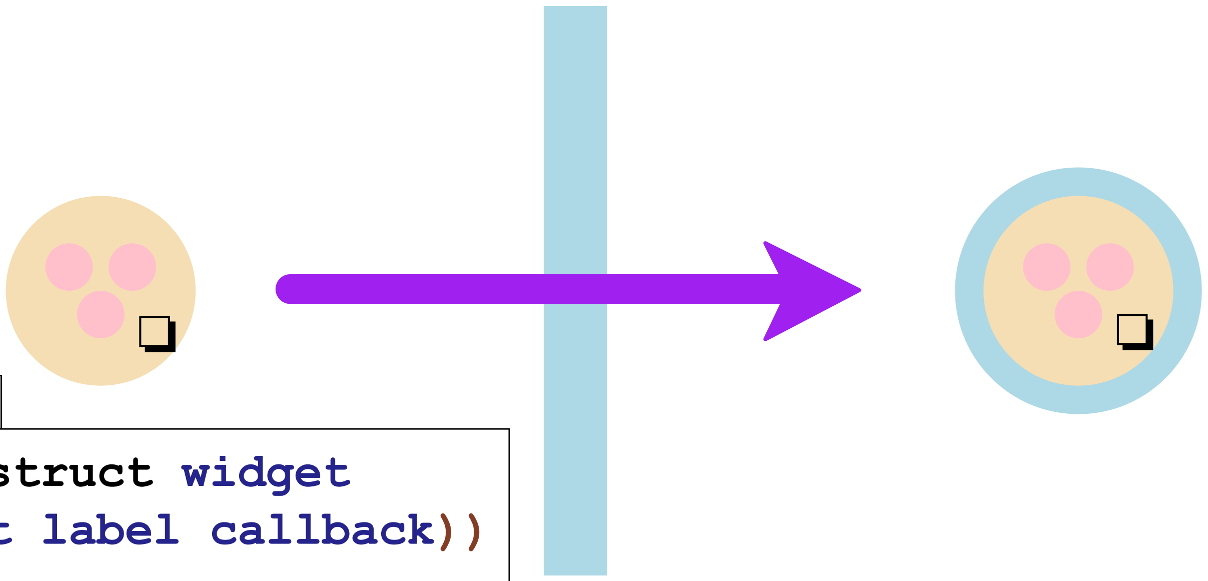
math.rkt

```
(define transforms
  (vector identity sqr sqrt))

(provide/contract
 [state
  (vectorof
   (nonnegative-real?
    . -> . nonnegative-real?))])
```

circles.rkt

```
(require "math.rkt")

(vector-set! transforms
             0
             abs)
```

Contract Examples



widget.rkt

```
(define-struct widget
  (parent label callback))
....
```

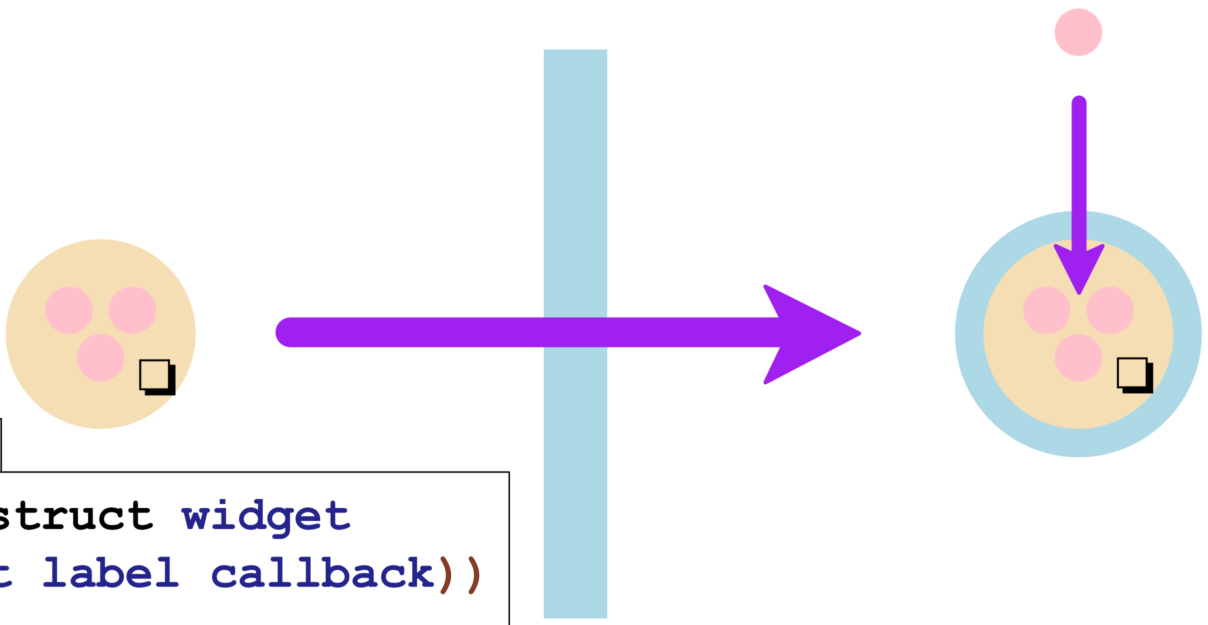
scene.rkt

```
(require "widget.rkt")

(define plot
  (make-button parent "Draw"
               draw-callback))

(provide/contract
 [plot (struct/c widget gl-window?
                 ....)])
```

circles.rkt

```
(require "widget.rkt"
         "scene.rkt")


widget
```

Contract Examples

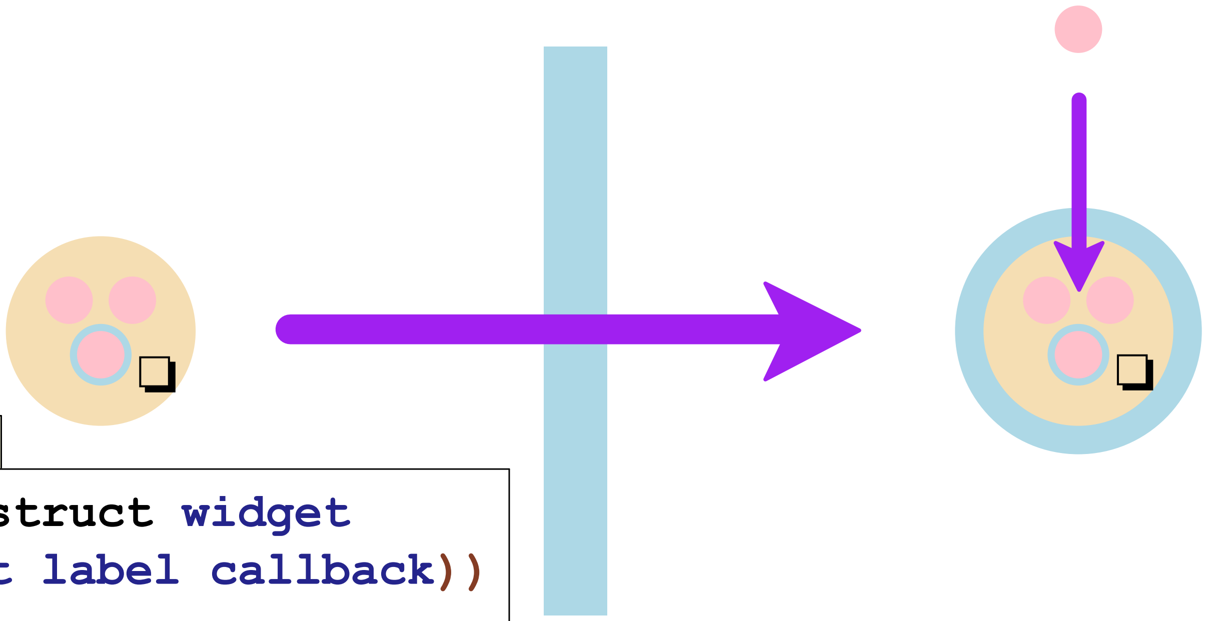


widget.rkt
```
(define-struct widget
  (parent label callback))
....
```

scene.rkt
```
(require "widget.rkt")

(define plot
  (make-button parent "Draw"
                draw-callback))

(provide/contract
 [plot (struct/c widget gl-window?
                      ....)])
```

circles.rkt
```
(require "widget.rkt"
         "scene.rkt")

(set-widget-parent!
 plot
 other-window)
```

Contract Examples



widget.rkt

```
(define-struct widget
  (parent label callback))
....
```

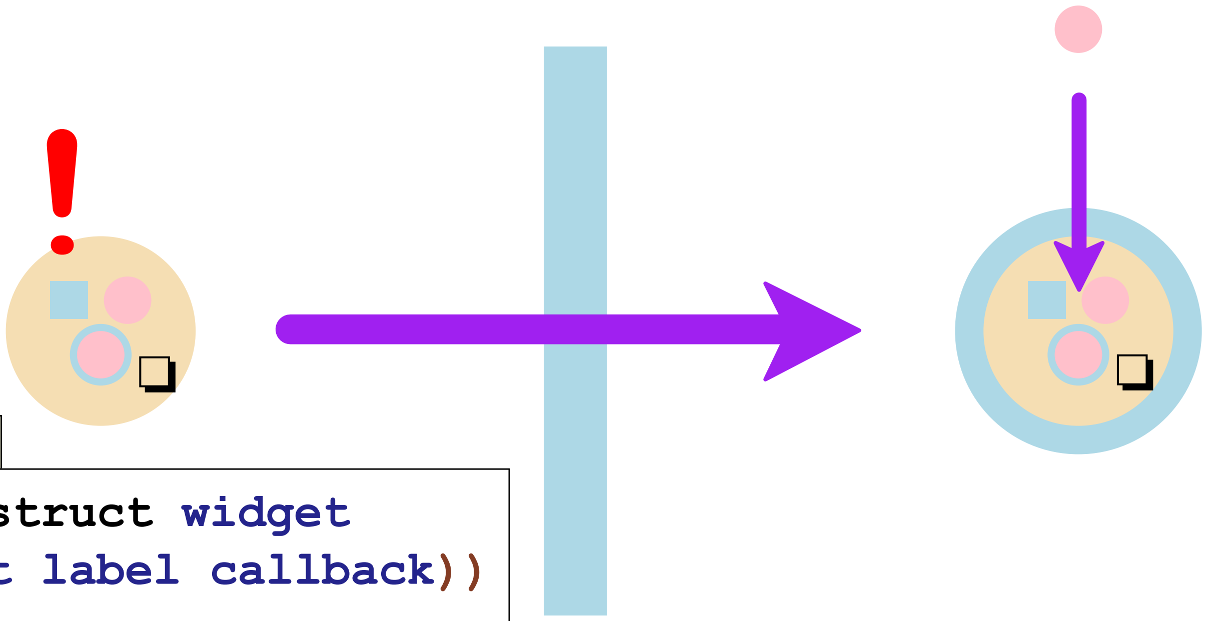
scene.rkt

```
(require "widget.rkt")

(define plot
  (make-button parent "Draw"
               draw-callback))

(provide/contract
 [plot (struct/c widget gl-window?
                 ....)])
```

circles.rkt

```
(require "widget.rkt"
         "scene.rkt")

(set-widget-callback!
 plot
 save-file)
```

Contract Examples



widget.rkt

```
(define-struct widget
   (parent label callback))
....
```

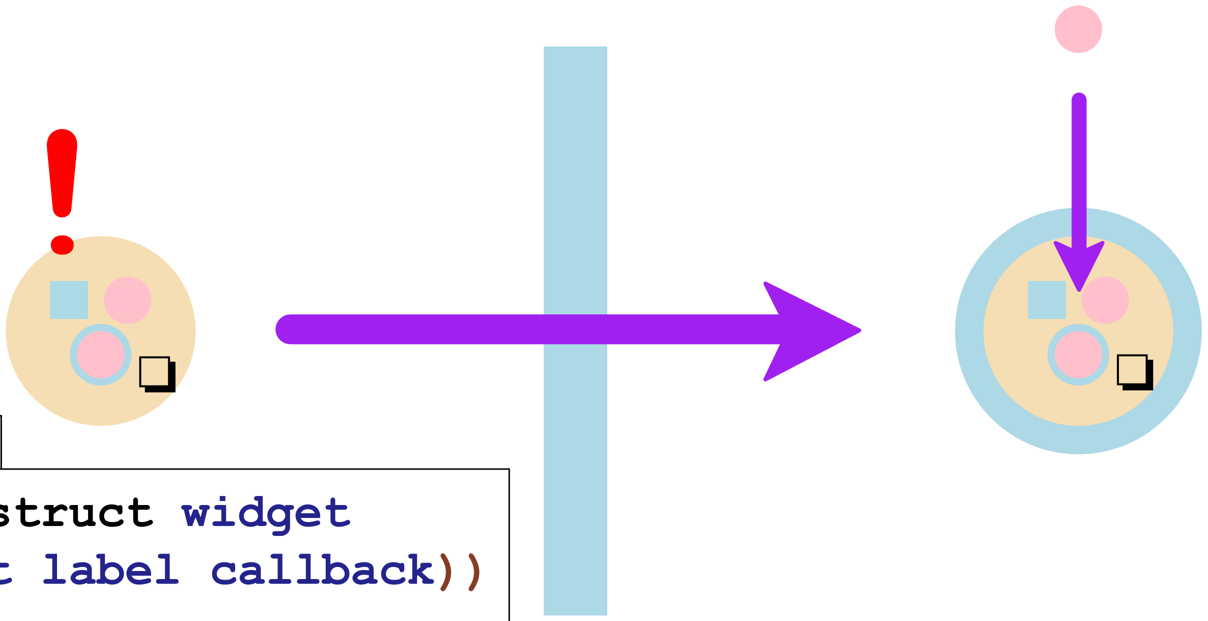
scene.rkt

```
(require "widget.rkt")

(define plot
   (make-button parent "Draw"
                draw-callback))

(provide/contract
 [plot (struct/c widget α
                ....)])
```

circles.rkt

```
(require "widget.rkt"
         "scene.rkt")

(set-widget-parent!
 plot
 other-window)
```

Contract Examples



widget.rkt

```
(define-struct widget
  (parent label callback))
....
```

scene.rkt

```
(require "widget.rkt")

(define plot
  (make-button parent "Draw"
               draw-callback))

(provide/contract
 [plot (struct/c widget α
                 ....)])
```

```
                       .rkt"
                       .rkt")

(set-widget-parent!
 plot
 other-window)
```

originator might assume invariants that are enforced by constructor

Contract Hierarchy

flat contracts $\subset$ chaperone contracts $\subset$ impersonator contracts

Contract Hierarchy

flat: all checks are immediate

$\boxed{\text{flat contracts}} \subset$ chaperone contracts $\subset$ impersonator contracts

```
positive-integer?

  (listof char?)
```

Contract Hierarchy

chaperone: checks may be delayed

$$\text{flat contracts} \subset \boxed{\text{chaperone contracts}} \subset \text{impersonator contracts}$$

```
(positive-integer? . -> . negative-integer?)

(number? (real? . -> . real?) . -> . number?)
```

Contract Hierarchy

impersonator: checks may substitute alternatives

flat contracts $\subset$ chaperone contracts $\subset$ $\boxed{\text{impersonator contracts}}$

```
(α α . -> . α)
```

Contract Hierarchy

chaperone: checks may be delayed

$$\text{flat contracts} \subset \boxed{\text{chaperone contracts}} \subset \text{impersonator contracts}$$

consistent with opacity and immutability

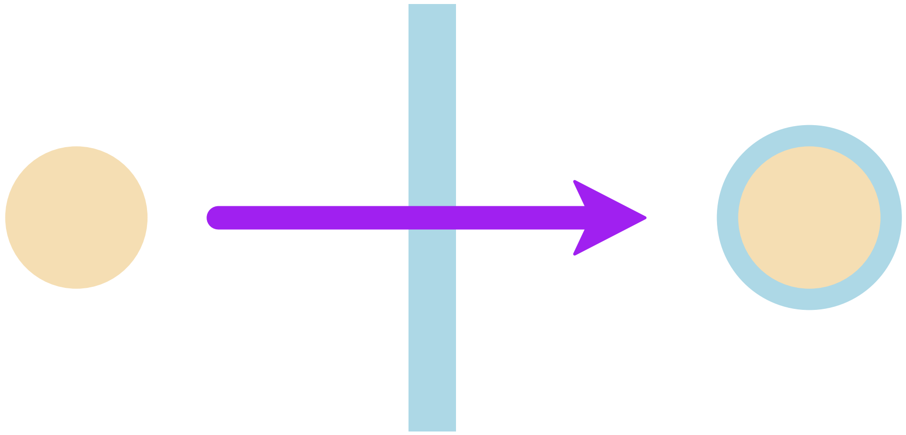
Contract Hierarchy

impersonator: checks may substitute alternatives

flat contracts $\subset$ chaperone contracts $\subset$ impersonator contracts

inconsistent with opacity and immutability

Implementing Contracts

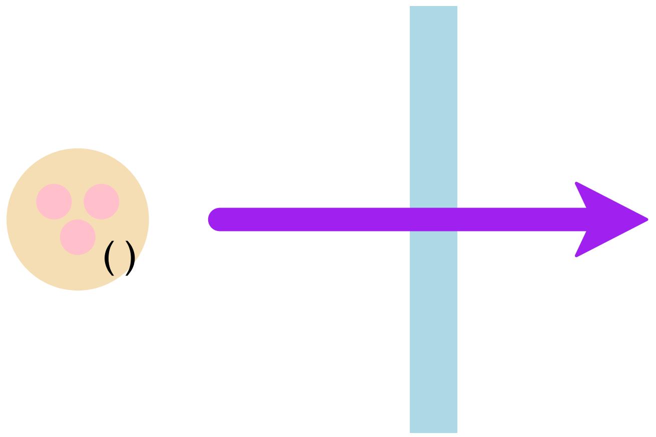


```
(λ (x) (* x x))
```

$\longrightarrow$

```
(lambda (x)
  (unless (real? x)
    (contract-failure))
  (let ([v ((λ (x) (* x x)) x)])
    (unless (nonnegative-real? v)
      (contract-failure))
    v))
```

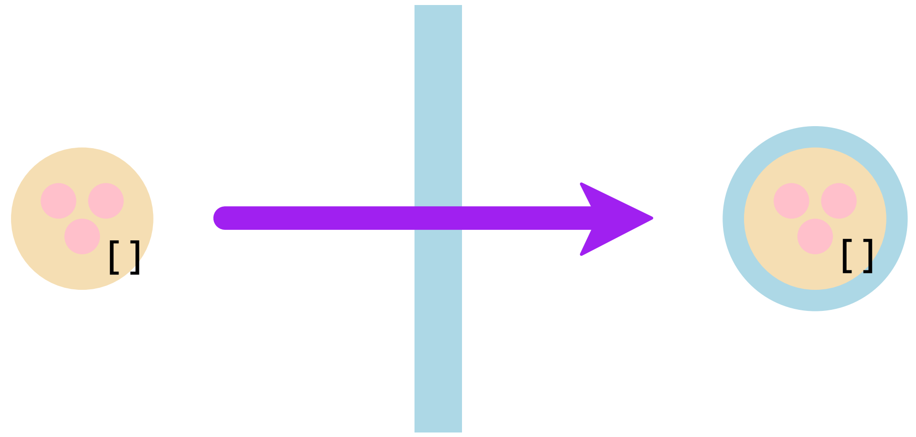
Implementing Contracts



```
(list identity
      sqr
      sqrt)
```

```
(map
 (lambda (transform)
   (lambda (x)
     (unless (nonnegative-real? x)
       (contract-failure))
     (let ([v (transform x)])
       (unless (nonnegative-real? v)
         (contract-failure))
      v)))
 (list identity
       sqr
       sqrt))
```

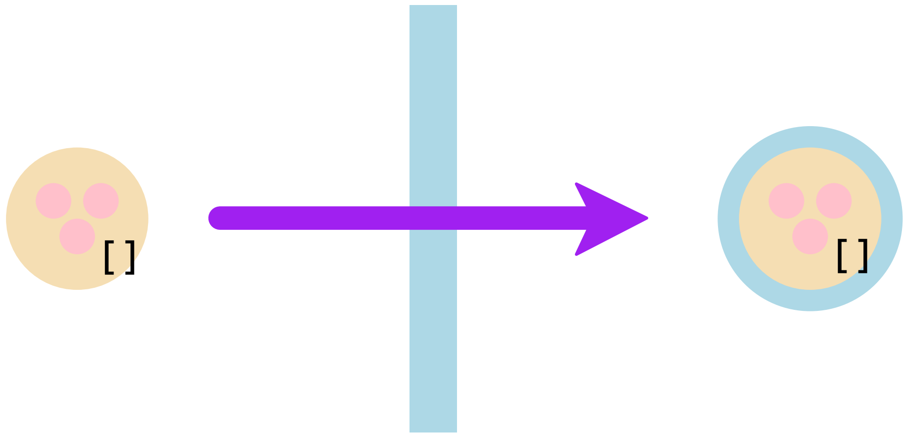
Implementing Contracts



```
(vector 0.1
        0.4
        7.9)
```

```
??? (vector 0.1
            0.4
            7.9)
```

Implementing Contracts



```
(vector 0.1
        0.4
        7.9)
```

$\longrightarrow$

```
??? (vector 0.1
            0.4
            7.9)
```

need a `???` that works with all vector operations

Implementing Contracts

Before adding run-time support:

	flat contracts	higher-order contracts
procedures	✓	✓
immutable data	✓	✓
mutable data	~	✗
opaque structures	~	✗
objects	✓*	✓*

* all objects incur cost

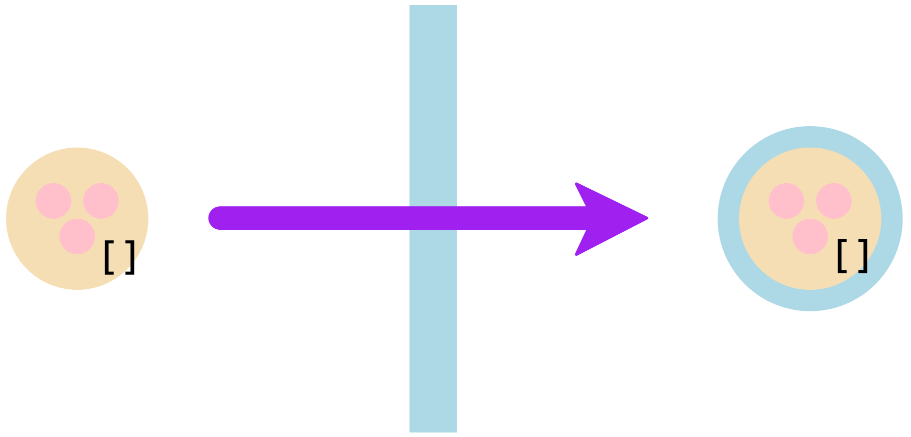
Implementing Contracts

After adding run-time support:

	flat contracts	chaperone contracts	impersonator contracts
procedures	✓	✓	✓
immutable data	✓	✓	✗*
mutable data	✓	✓	✓
opaque structures	✓	✓	✗*
objects	✓	✓	✓

* not sensible

Implementing Contracts



```
(vector 0.1
        0.4
        7.9)
```

```
??? (vector 0.1
            0.4
            7.9)
```

need a ??? that works with all vector operations

Implementing Contracts

- **Proxies**
    - *Proxies: Design Principles for Robust Object-oriented Intercession APIs*, Van Cutsem and Miller, DLS'10
- **Aspects**
    - *Harmless Advice*, Dantas and Walker, POPL'06

Implementing Contracts

- ▶ **Proxies**
  - *Proxies: Design Principles for Robust Object-oriented Intercession APIs*, Van Cutsem and Miller, DLS'10
- **Aspects**
  - *Harmless Advice*, Dantas and Walker, POPL'06

Chaperones and Impersonators

Chaperone constructors:

```
(chaperone-procedure proc alt-apply)

(chaperone-vector vec alt-vec-ref alt-vec-set!)

(chaperone-struct struct
                  accessor alt-accessor ...
                  mutator alt-mutator ...)
```

Impersonator constructors:

```
(impersonate-procedure proc alt-apply)

(impersonate-vector vec alt-vec-ref alt-vec-set!)
```

mutable vectors only

```
(impersonate-struct struct
                    accessor alt-accessor ...
                    mutator alt-mutator ...)
```

mutable fields only

Chaperones and Im

Alternate arguments/results must chaperone originals

Chaperone constructors:

```
(chaperone-procedure proc alt-apply)

(chaperone-vector vec alt-vec-ref alt-vec-set!)

(chaperone-struct struct
                  accessor alt-accessor ...
                  mutator alt-mutator ...)
```

Impersonator constructors:

```
(impersonate-procedure proc alt-apply)

(impersonate-vector vec alt-vec-ref alt-vec-set!)
```

mutable vectors only

```
(impersonate-struct struct
                    accessor alt-accessor ...
                    mutator alt-mutator ...)
```

mutable fields only

Chaperones and Im

Alternate arguments/results must chaperone originals

Chaperone constructors:

```
(chaperone-procedure proc alt-apply)

(chaperone-vector vec alt-vec-ref alt-vec-set!)

(chaperone-struct struct
                  accessor alt-accessor ...
                  mutator alt-m
```

Alternate arguments/results are unconstrained

Impersonator constructors:

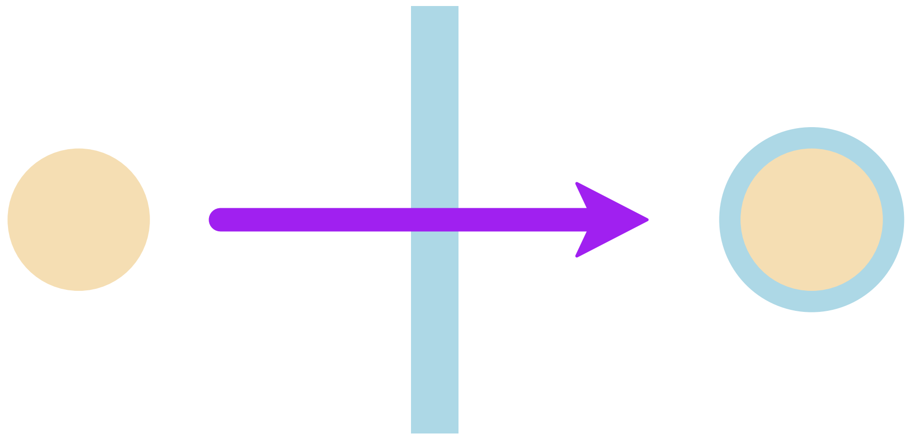
```
(impersonate-procedure proc alt-apply)

(impersonate-vector vec alt-vec-ref alt-vec-set!)
```

mutable vectors only

```
(impersonate-struct struct
                    accessor alt-accessor ...
                    mutator alt-mutator ...)
```

mutable fields only

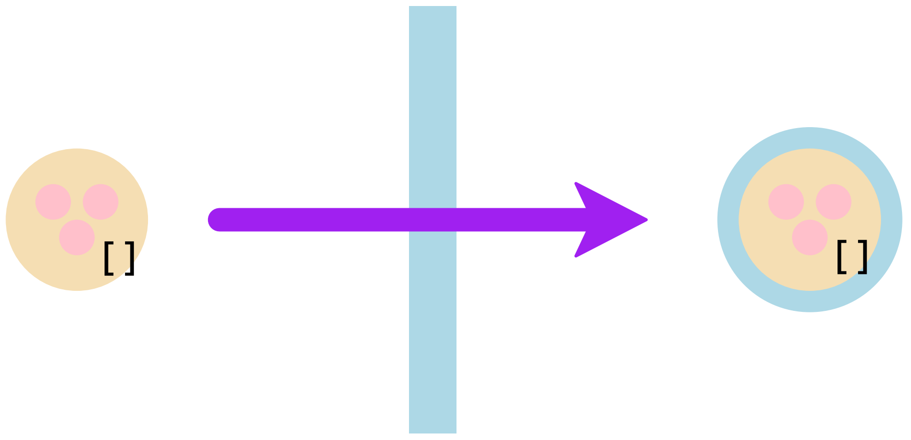
Chaperones and Impersonators



```
(λ (x) (* x x))
```

```
(chaperone-procedure
 (λ (x) (* x x))
 (lambda (x)
   (unless (real? x)
     (contract-failure))
   (values
    x
    (lambda (v)
      (unless (nonnegative-real? v)
        (contract-failure))
     v))))
```

Chaperones and Impersonators

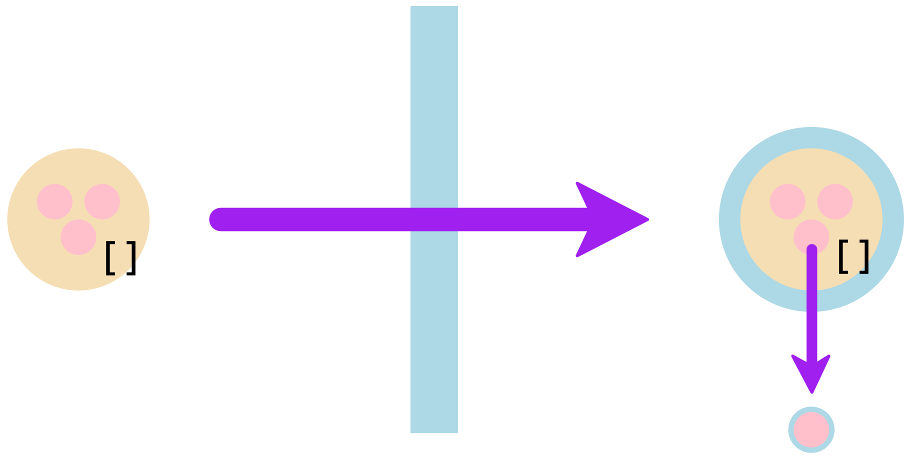


```
(vector 0.1
        0.4
        7.9)
```

→

```
(chaperone-vector
 (vector 0.1
         0.4
         7.9)
 (lambda (vec i val) ; ref
   (unless (nonnegative-real? val)
     (contract-failure))
   val)
 (lambda (vec i val) ; set
   (unless (nonnegative-real? val)
     (contract-failure))
   val))
```

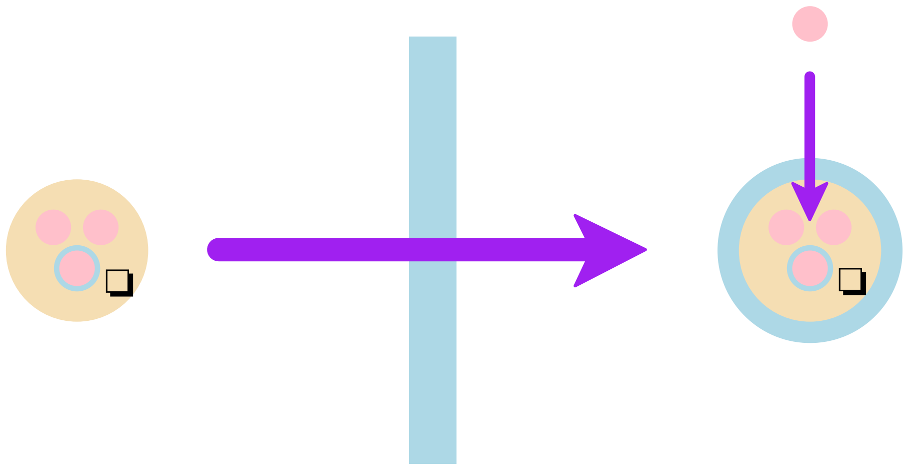
Chaperones and Impersonators



```
(vector identity
        sqr
        sqrt)
```

```
(chaperone-vector
 (vector identity sqr sqrt)
 (lambda (vec i val)
   (unless (procedure? val)
     (contract-failure))
   (chaperone-procedure
    val
    (lambda (x)
      (unless (nonnegative-real? x)
        (contract-failure))
      ....)))
 ....)
```

Chaperones and Impersonators



```
(make-widget parent label
             callback)
```

```
(chaperone-struct
 (make-widget parent label
              callback)
 widget-parent
 (lambda (w val)
   (unless (gl-window? val)
     (contract-failure))
   val)
 set-widget-callback!
 (lambda (w val)
   (chaperone-procedure
    val ....)))
```

Results

	flat contracts	chaperone contracts	impersonator contracts
procedures	✓	✓	✓
immutable data	✓	✓	✗*
mutable data	✓	✓	✓
opaque structures	✓	✓	✗*
objects	✓	✓	✓

* not sensible

Results

	flat contracts	chaperone contracts	impersonator contracts
procedures	✓	✓	✓
immutabl			✗*
mutable data	✓	✓	✓
opaque structures	✓	✓	✗*
objects	✓	✓	✓

1. Closes known holes in Typed Racket

* not sensible

Results

	flat contracts	chaperone contracts	impersonator contracts
procedures	✓	✓	✓
immutable data	✓	✓	✗*
mutable data	✓	✓	✓
opaque structu			
objects	✓	✓	✓

2. Object contracts supported without penalty

* not sensible

Results

	flat contracts	chaperone con acts	impersonator contracts
proce			
immutable			
mutable data	✓	✓	✓
opaque structures	✓	✓	✗*
objects	✓	✓	✓

3. Claim:

No failures in chaperones ⇒

erasing chaperones produces the same result

* not sensible

Results

	flat contracts	chaperone contracts	impersonator contracts
procedures	✓	✓	✓
immutable data	✓	✓	✗*
mutable data	✓	✓	✓
opaque structures	✓	✓	✗*
objects	✓	✓	✓

* not sensible