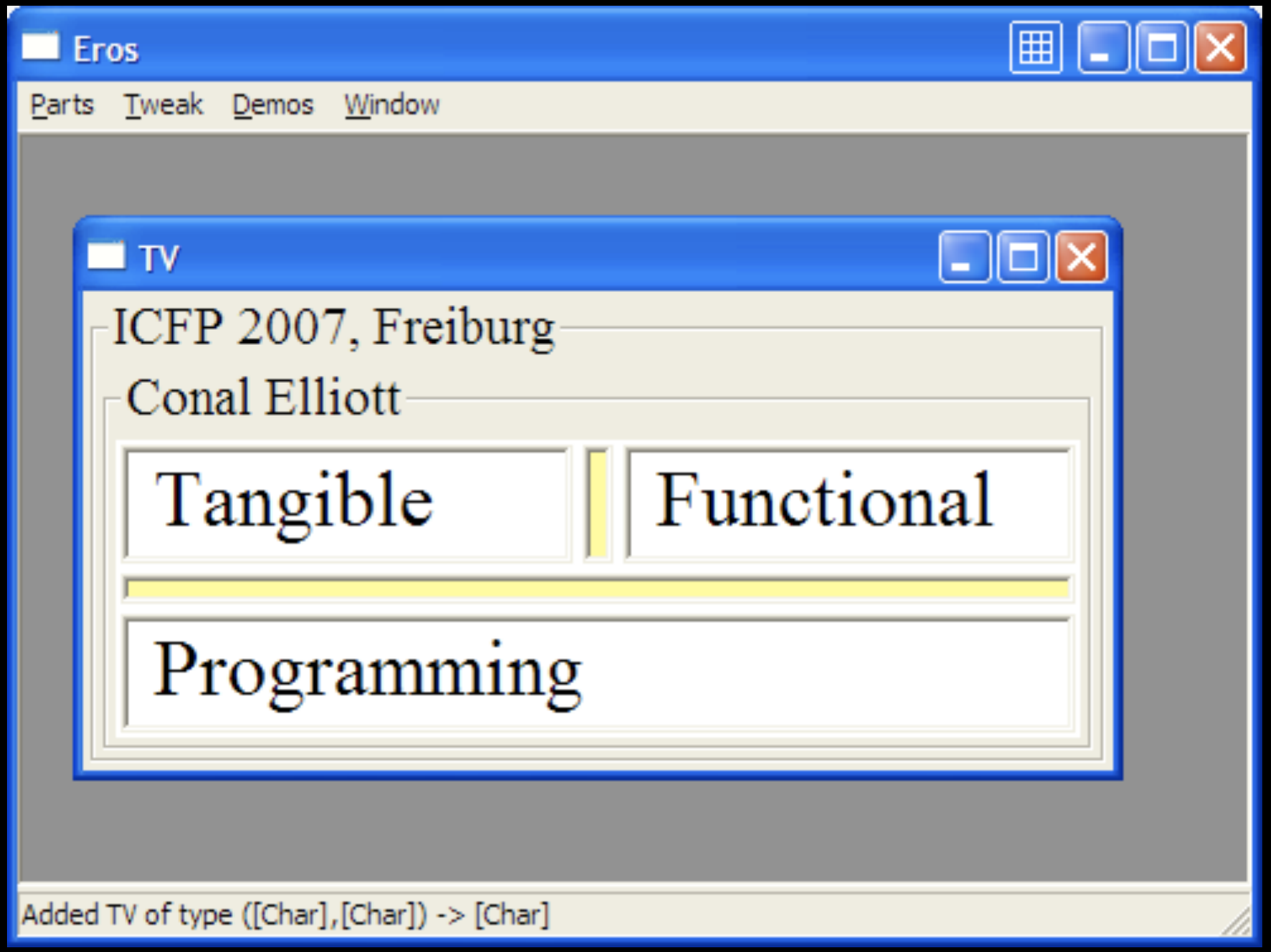
Tangible Value

in Haskell

Conal Elliott

- http://www.youtube.com/watch?v=faJ8N0giqzw
- http://conal.net/papers/Eros/
- http://journal.conal.net/#
  [[separating IO from logic -- example]]
- http://conal-elliott.blogspot.com/search/label/TV

Eros


Eros
Parts
Tweak
Demos
Window
TV
ICFP 2007, Freiburg
Conal Elliott
Tangible
Functional
Programming
Added TV of type ([Char],[Char]) -> [Char]

applications:	libraries:
1. user-friendly	1. programmer-friendly
2. usable	2. composable
3. concrete	3. abstract
4. visual	4. syntactic

UNIX philosophy

- Write programs that do one thing and do it well
- Write programs to work together
- Write programs to handle text streams, because that is a universal interface

Doug McIlroy

```
godfat ~/p/l/l/l/proc> ls | sort | cat -n
     1 bind.rb
     2 chain.rb
     3 compose.rb
     4 curry.rb
```

TV

原始
程式

```haskell
module Grading where

import Data.List (sort)
import Data.Map (Map,empty,keys,insertWith,findWithDefault)
import Text.Printf

import Interface.TV
import Interface.TV.OFun() -- work around GHC bug.  ticket #1145
```

```haskell
grades = do
    src <- readFile "tasks"
    let pairs  = map (split.words) (lines src)
        grades = foldr insert empty pairs
    mapM_ (draw grades) (sort (keys grades))
  where
    insert (s, g) = insertWith (++) s [g]
    split [name,mark] = (name, read mark)
    draw g s = printf "%s\t%s\tAverage: %f\n" s (show marks) avg
      where
        marks = findWithDefault (error "No such student") s g
        avg   = sum marks / fromIntegral (length marks) :: Double
```

抽出

I/O

```
gradingStr src = concatMap (draw grades) (sort (keys grades))
  where
    pairs  = map (split.words) (lines src)
    grades = foldr insert empty pairs

    insert (s, g) = insertWith (++) s [g]
    split [name,mark] = (name, read mark)
    draw g s = printf "%s\t%s\tAverage: %f\n" s (show marks) avg
      where
        marks = findWithDefault (error "No such student") s g
        avg   = sum marks / fromIntegral (length marks) :: Double
```

```haskell
type GradingStr = String -> String
gradingStr :: GradingStr

grades_2 = readFile "tasks" >>= return . gradingStr >>= putStr
```

in

TV

```haskell
type GradingStr = String -> String
gradingStr :: GradingStr

grades_2 = readFile "tasks" >>= return . gradingStr >>= putStr
```

```haskell
gradingStrOut = oLambda (fileIn "tasks") stringOut
gradingStrT :: TV KIO GradingStr
gradingStrT = tv gradingStrOut gradingStr

grades_3 = runTV gradingStrT
```

Eros

(⊥)

視覚化

Eros

Parts Tweak Demos Window

TV

true

TV

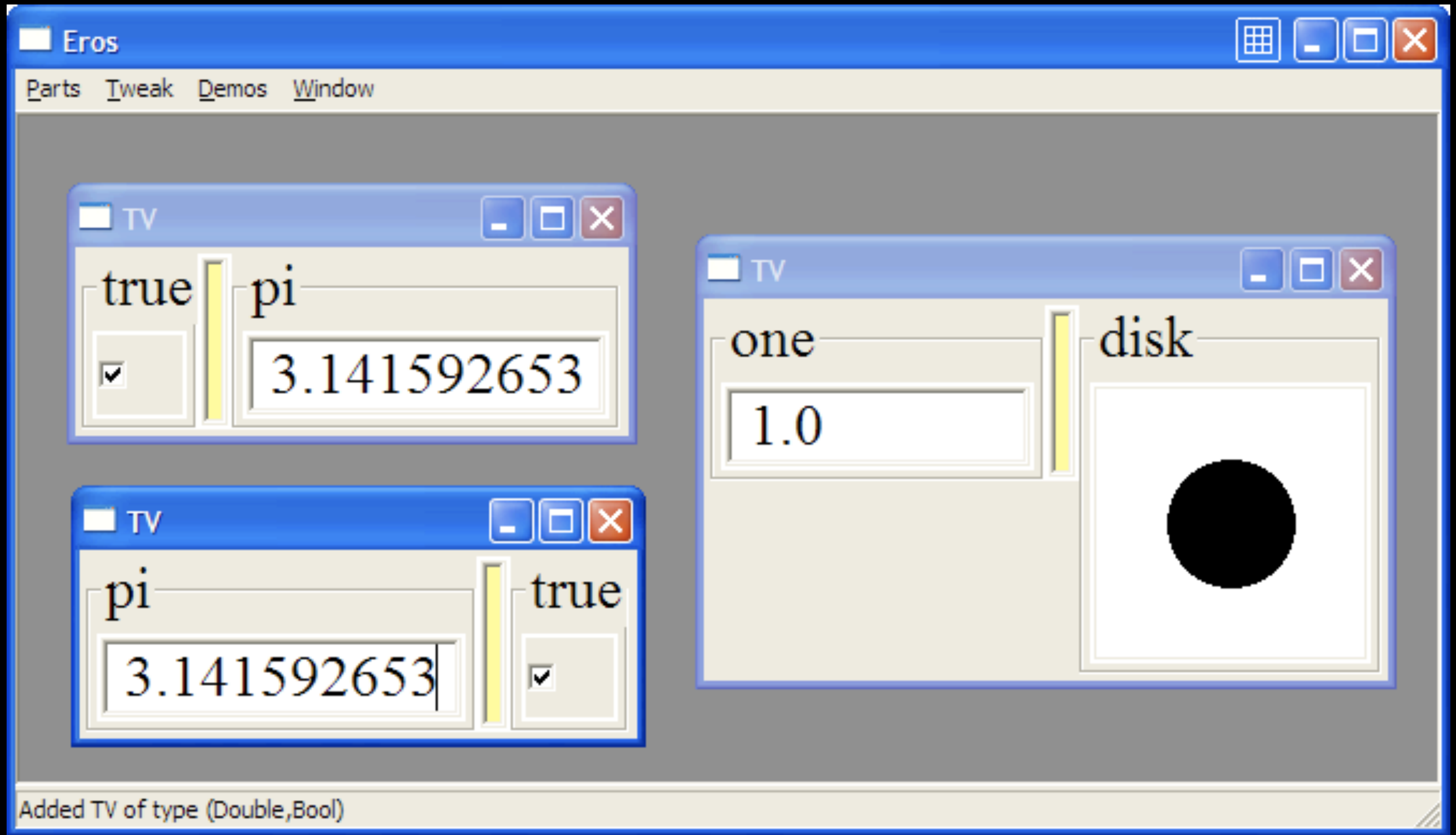
disk

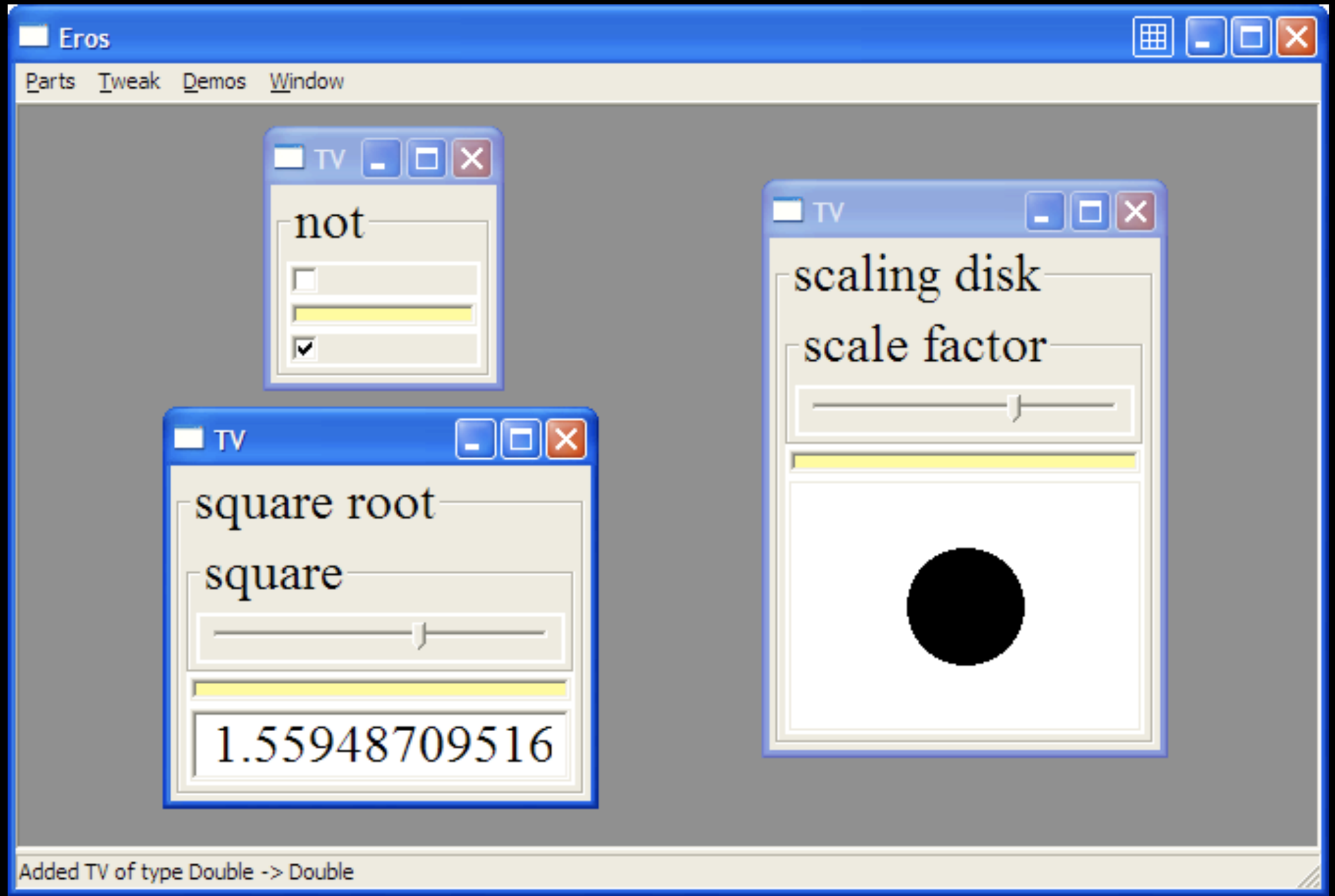
TV

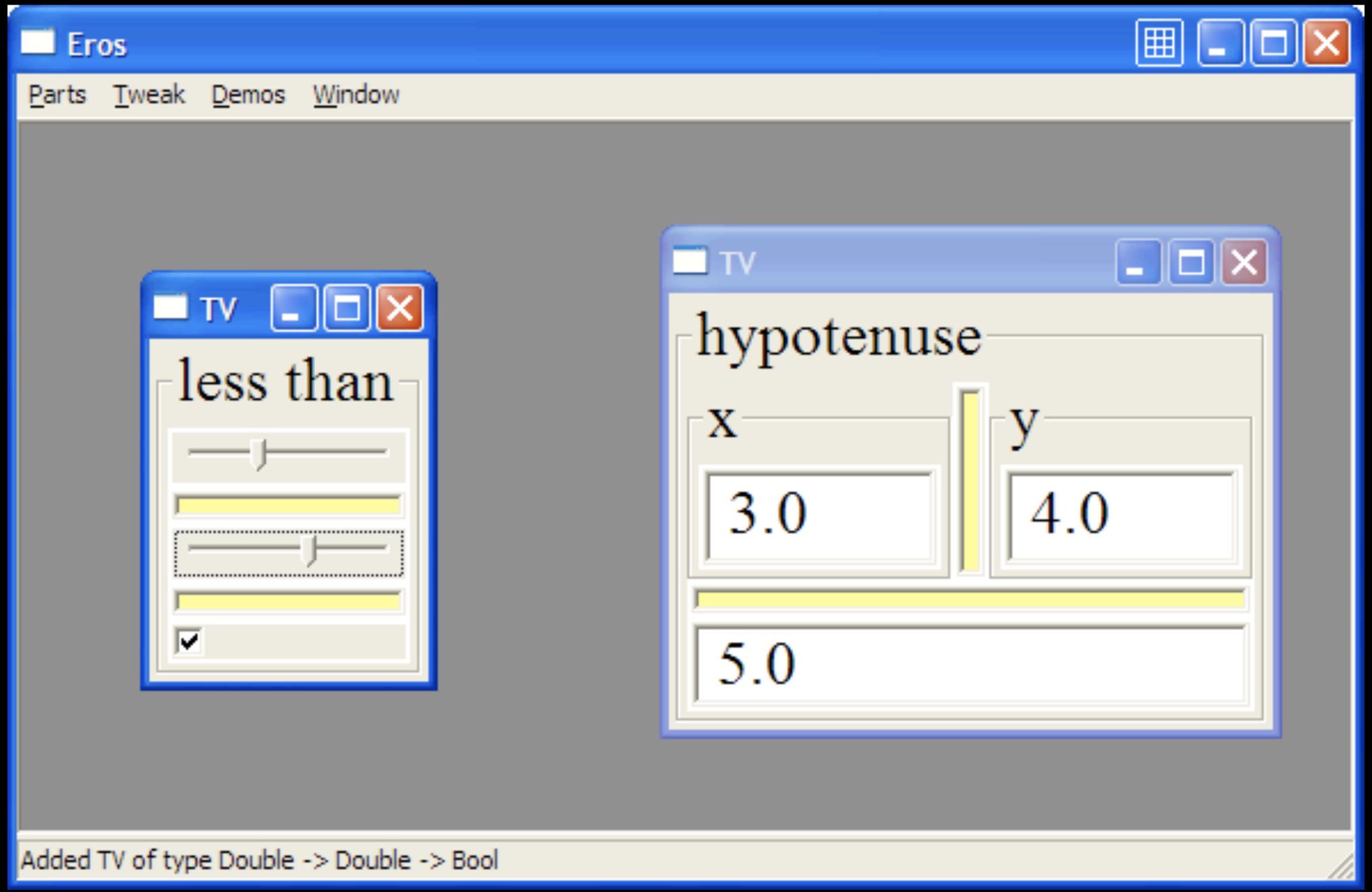
pi

3.14159265358897

Added TV of type Double


Eros
Parts
Tweak
Demos
Window
TV
true
pi
3.141592653
TV
one
1.0
disk
TV
pi
3.141592653
true
Added TV of type (Double,Bool)


Eros
Parts
Tweak
Demos
Window
TV
not
TV
square root
square
1.55948709516
TV
scaling disk
scale factor
Added TV of type Double -> Double


Eros
Parts
Tweak
Demos
Window
TV
less than
TV
hypotenuse
x
y
3.0
4.0
5.0
Added TV of type Double -> Double -> Bool

(2)

融合

Eros

Parts Tweak Demos Window

TV

scaling disk

scale factor

TV

one

1.0

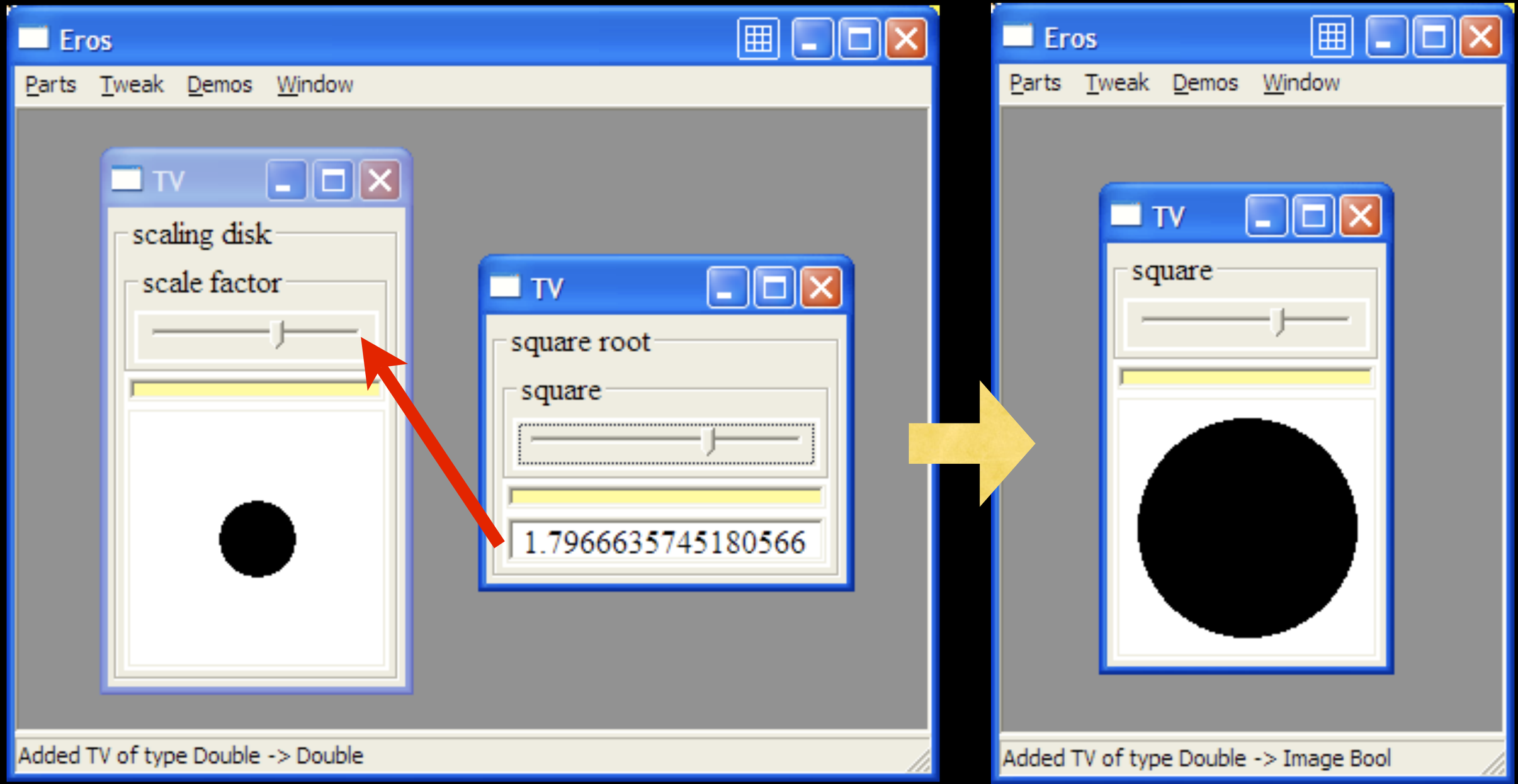
Added TV of type Double -> Double

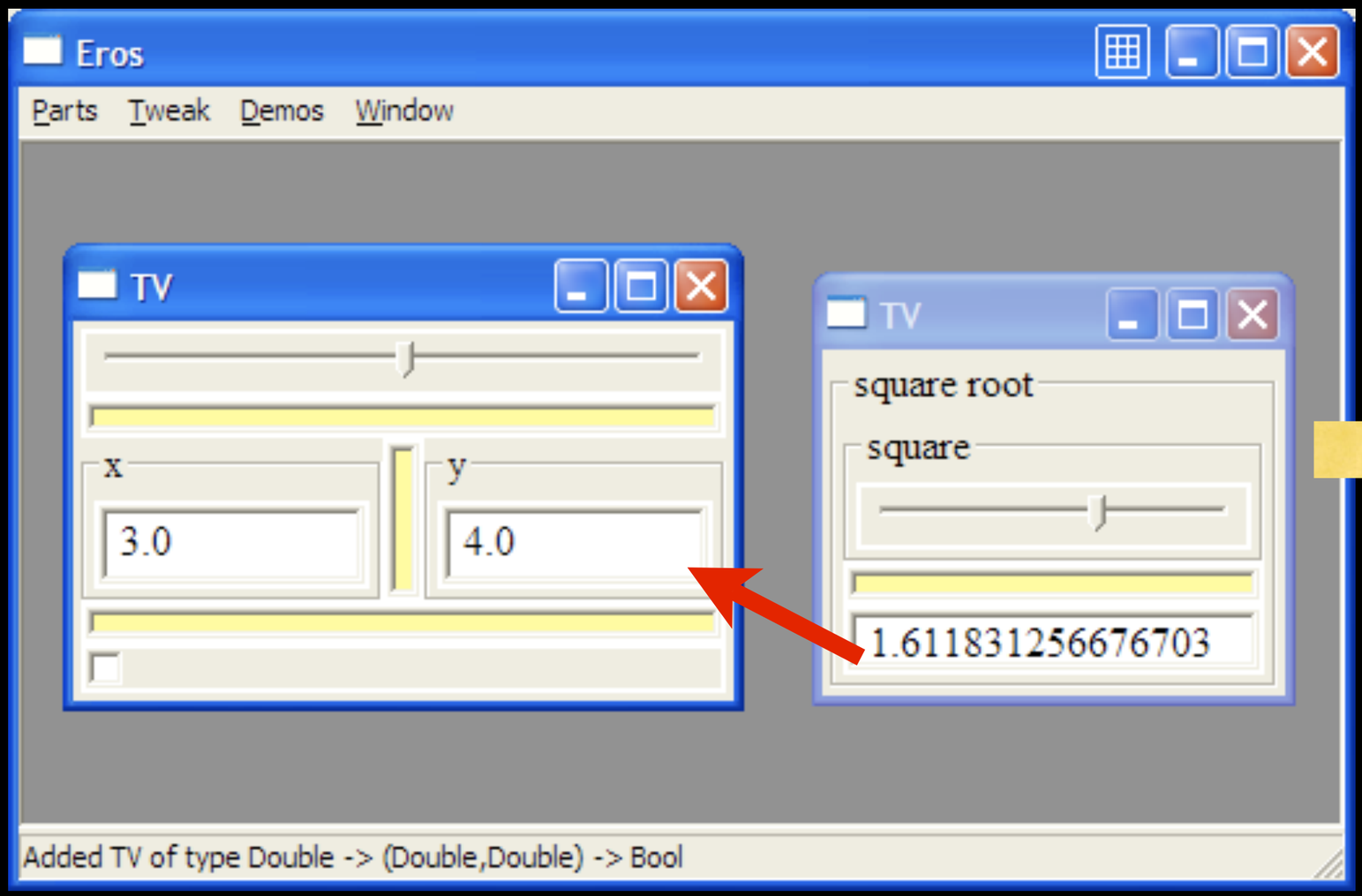
Eros

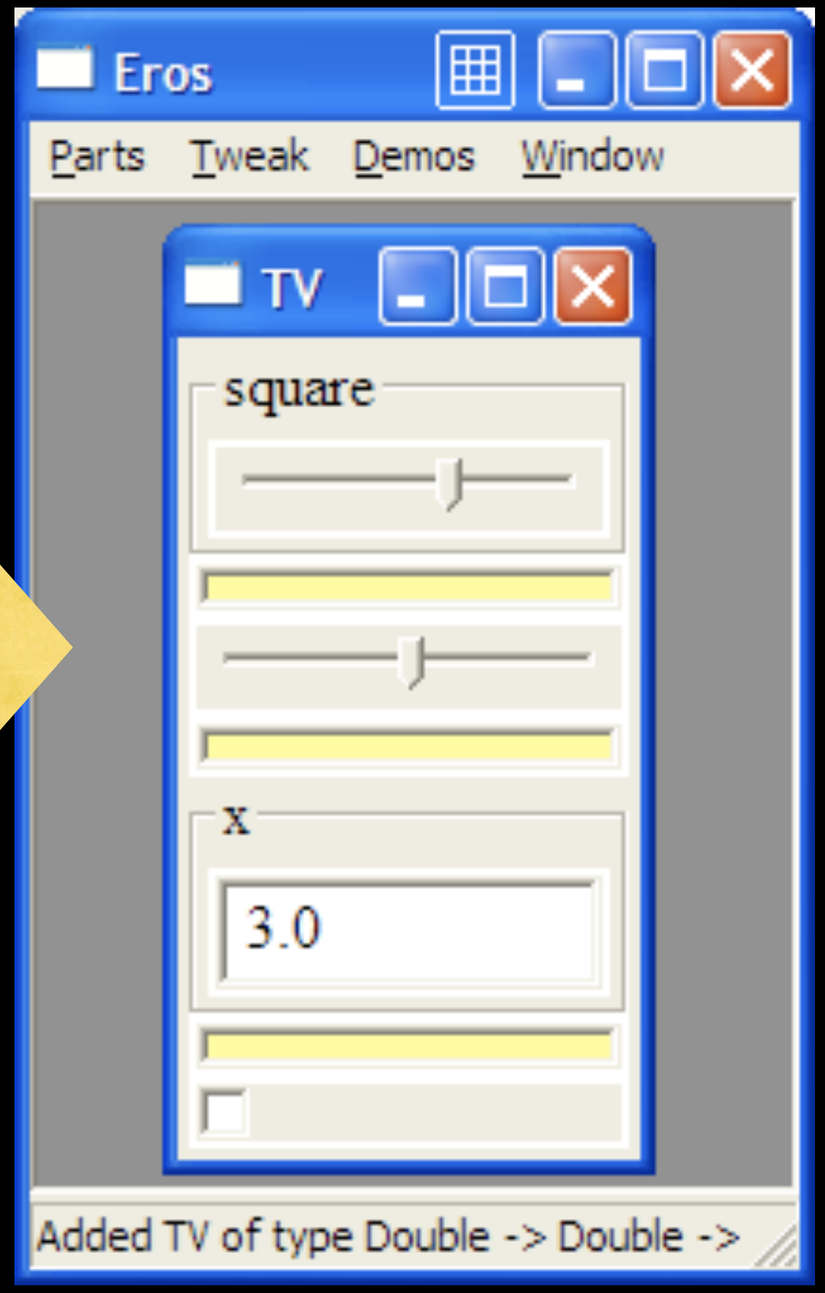
Parts Tweak Demos Window

TV

Added TV of type Image Bool


Eros
Parts Tweak Demos Window
TV
scaling disk
scale factor
TV
square root
square
1.7966635745180566
Added TV of type Double -> Double
Eros
Parts Tweak Demos Window
TV
square
Added TV of type Double -> Image Bool


Eros
Parts Tweak Demos Window
TV
x
3.0
y
4.0
TV
square root
square
1.611831256676703
Added TV of type Double -> (Double,Double) -> Bool


Eros
Parts Tweak Demos Window
TV
square
x
3.0
Added TV of type Double -> Double ->

```haskell
first  :: (a -> a') -> ((a, b) -> (a',b ))
second :: (b -> b') -> ((a, b) -> (a ,b'))
result :: (b -> b') -> ((a->b) -> (a->b'))
```

```haskell
first  f = \ (a, b) -> (f a,   b)
second g = \ (a, b) -> (  a, g b)
result g = \ f -> g . f
```

```
sf :: (b->b') -> (a,(b ,c))
               -> (a,(b',c))

sf = second.first

frsrf :: (c->c') -> (a->(f,b->(c ,g)),e)
                  -> (a->(f,b->(c',g)),e)

frsrf = first.result.second.result.first

funFirst ::
  (d -> (c->a)) -> ((d,b) -> (c->(a,b)))
```

(3)

MV

分離

```haskell
type TV a = (Out a, a)

type Out a = ...
put      :: Put a -> Out a
opair    :: Out a -> Out b -> Out (a, b)
olambda :: In  a -> Out b -> Out (a->b)

type In a = ...
get    :: Get a -> In a
ipair :: In  a -> In b -> In (a,b)
```

Eros

- TypeCompose
- DeepArrow
- DataDriven
- Phooey
- TV
- GuiTV
- wxHaskell
- wxWidgets

To explore

- Tangible polymorphism?
- Direct structural tweaks
- Symmetric In/Out (ilambda)
- “GUIs are types” as GUI design guide
- TVs as composable MVC