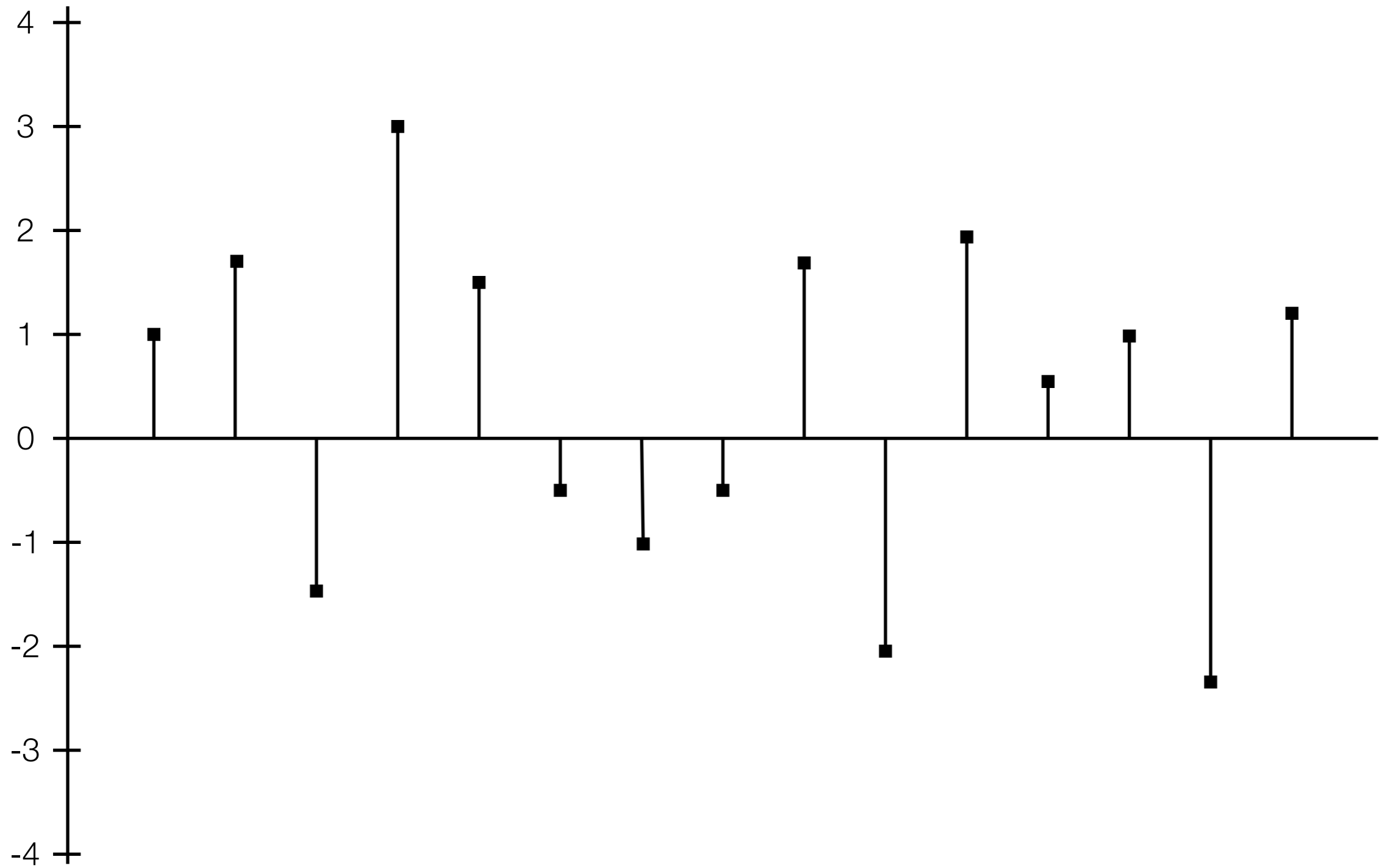


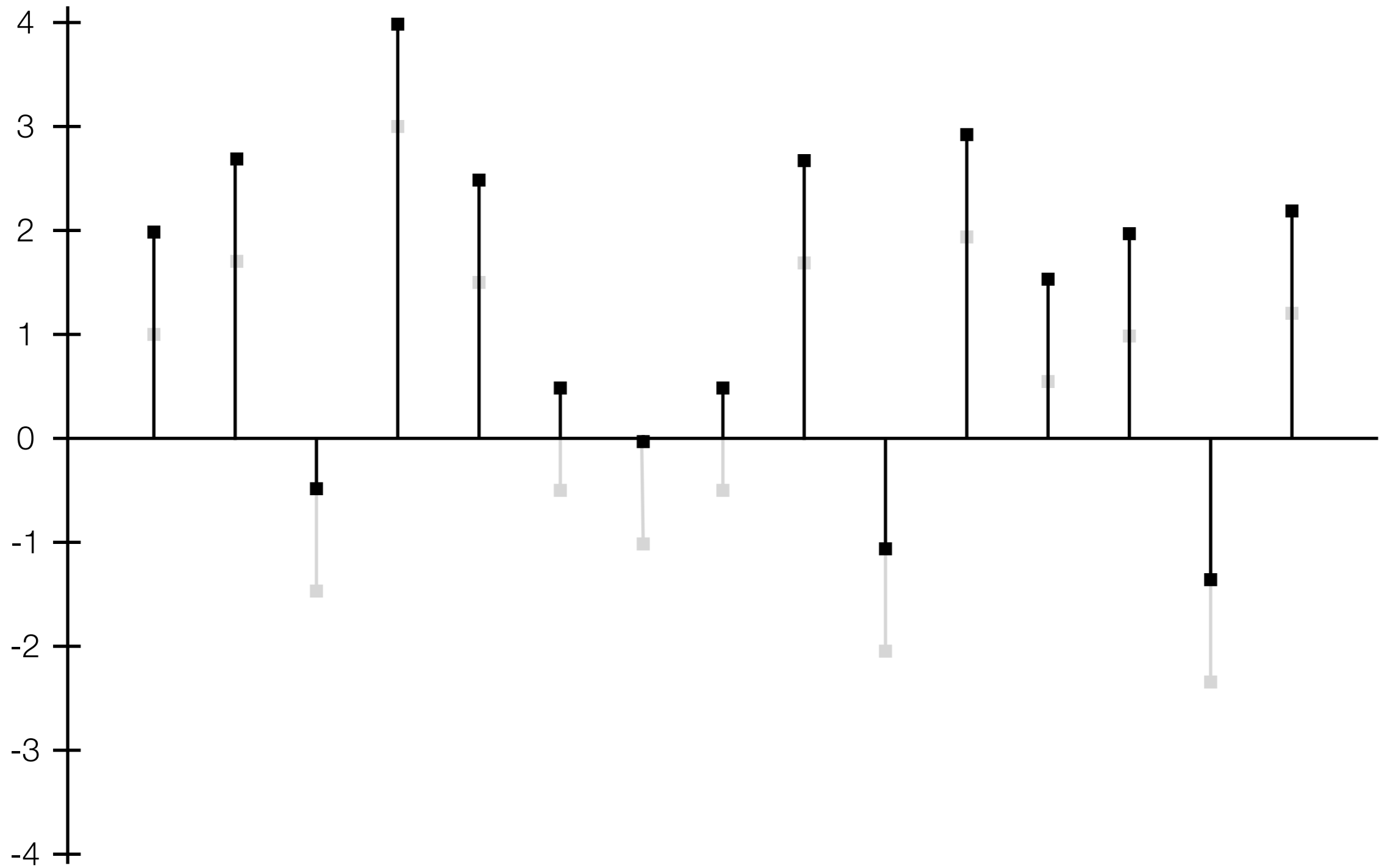
Data Flow Fusion

Ben Lippmeier
3/10/2014

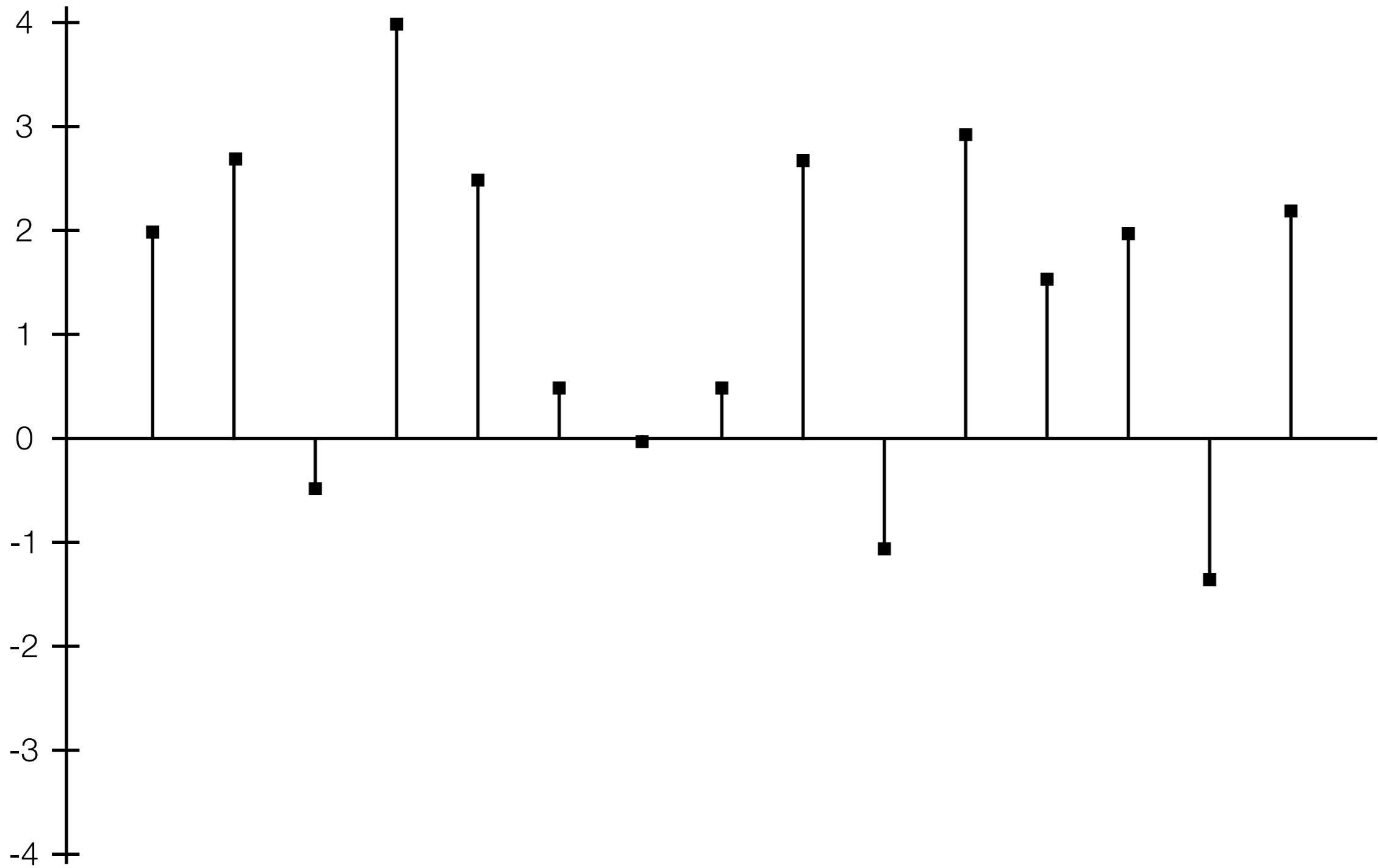
FilterMax



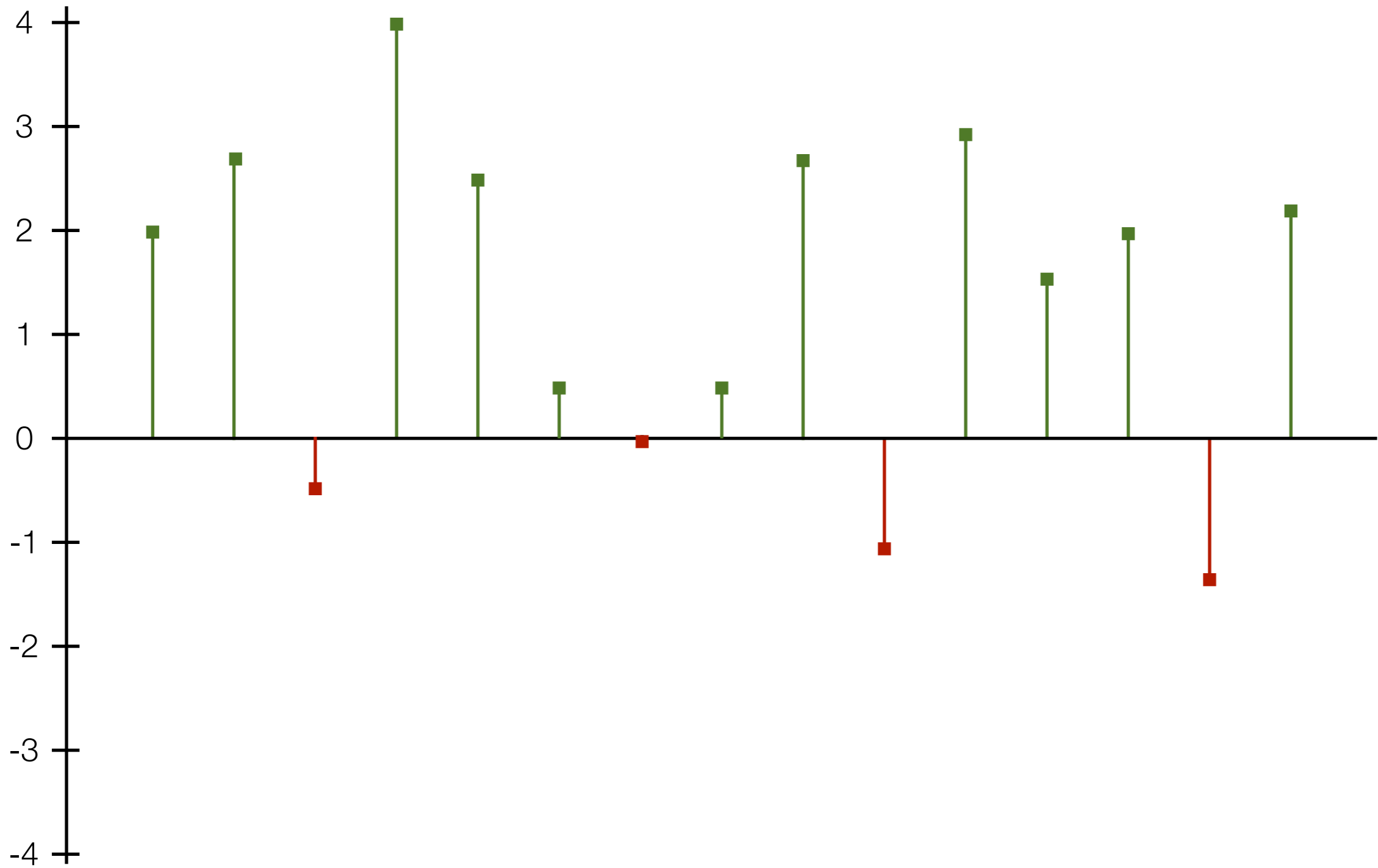
FilterMax



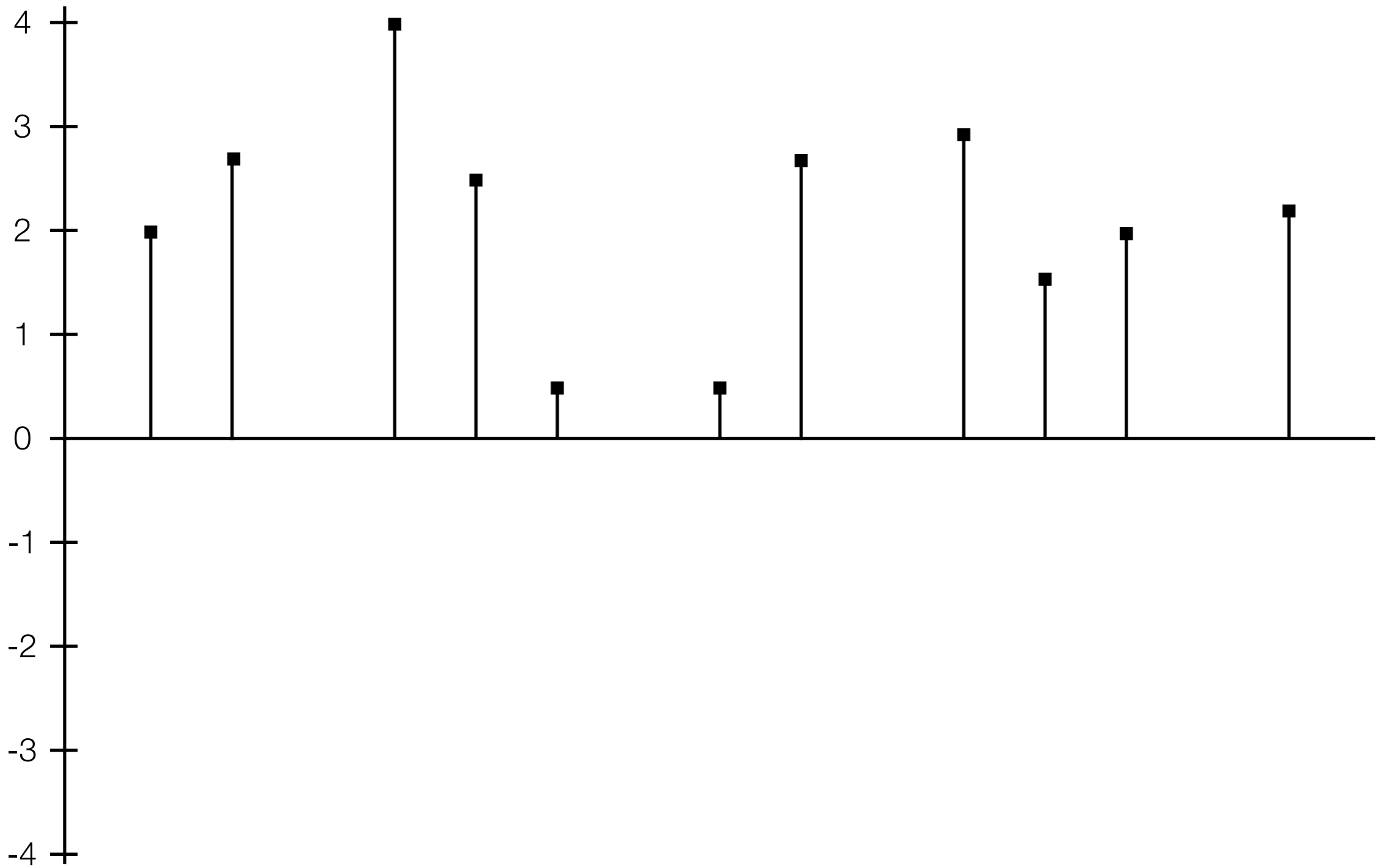
FilterMax



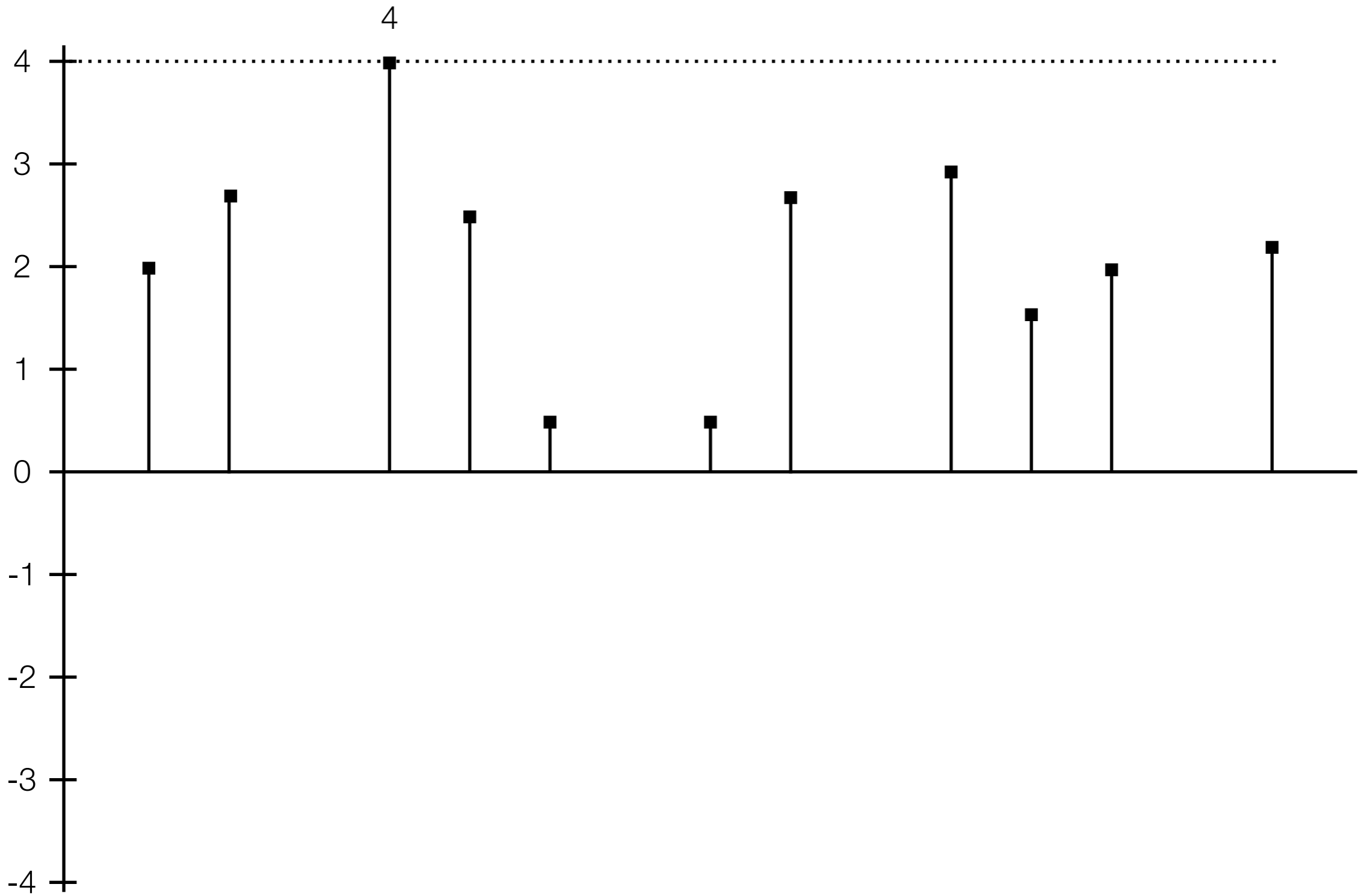
FilterMax



FilterMax

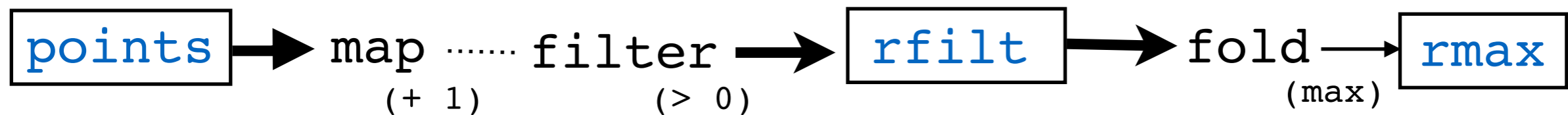


FilterMax



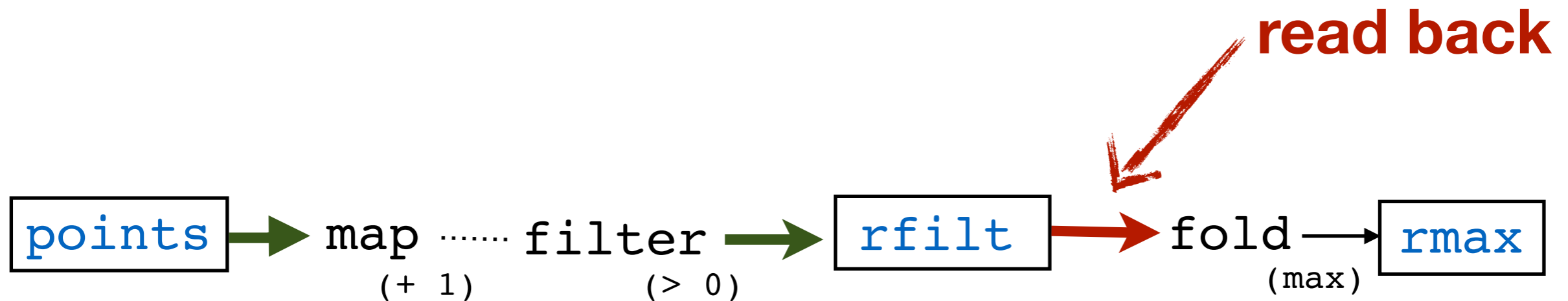
```
create table rfilt as
select y + 1 as h
from points
where h > 0
```

```
create table rmax as
select max(h) from rfilt
```




```
create table rfilt as
select y + 1 as h
from points
where h > 0
```

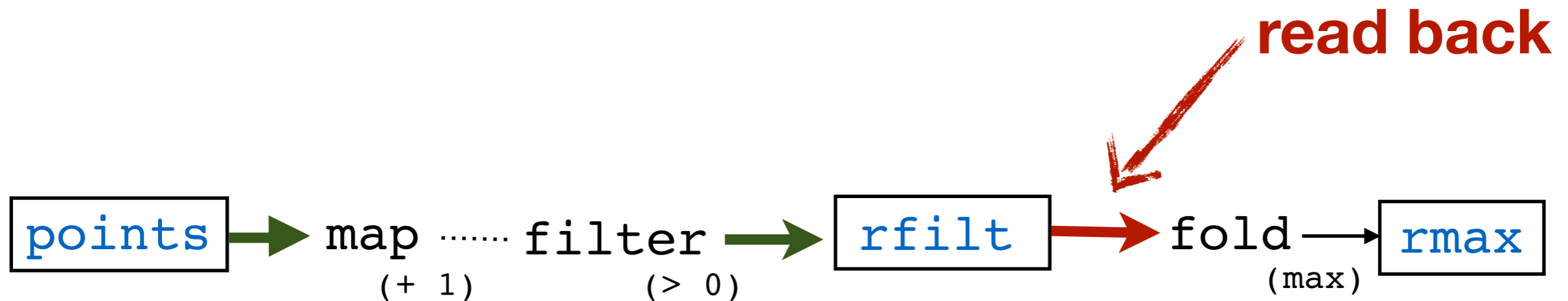
```
create table rmax as
select max(h) from rfilt
```



```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let t2      = map (+ 1) points
      rfilt   = filter (> 0) t2
      rmax    = fold max 0 rfilt
  in (rfilt, rmax)

```



```
filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let t2      = map (+ 1) points
      rfilt   = filter (> 0) t2
      rmax    = fold max 0 rfilt
  in (rfilt, rmax)
```

```
map f      = unstream . mapS f      . stream
filter p   = unstream . filterS p   . stream
fold f z   = foldS f z . stream
```

```
filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let t2      = unstream (mapS (+ 1) (stream points))
      rfilt   = unstream (filterS (> 0) (stream t2))
      rmax    = foldS max 0 (stream rfilt)
  in (rfilt, rmax)
```

```
map f      = unstream . mapS f      . stream
filter p   = unstream . filterS p   . stream
fold f z   = foldS f z . stream
```

```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let
    rfilt = unstream (filters (> 0)
                      (stream (unstream (mapS (+ 1)
                                             (stream points))))))
    rmax  = foldS max 0 (stream rfilt)
in (rfilt, rmax)

```

```

map f      = unstream . mapS f      . stream
filter p   = unstream . filters p   . stream
fold f z   = foldS f z . stream

```

```
filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let
    rfilt = unstream (filterS (> 0)
        (stream (unstream (mapS (+ 1)
            (stream points)))))
    rmax = foldS max 0 (stream rfilt)
in (rfilt, rmax)
```

```
filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let
    rfilt = unstream (filterS (> 0)
                        (stream (unstream (mapS (+ 1)
                                                (stream points)))))
    rmax  = foldS max 0 (stream rfilt)
in (rfilt, rmax)
```

RULE "stream/unstream"

forall xs. stream (unstream xs) = xs

```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let
    rfilt = unstream (filterS (> 0)
        (stream (unstream (mapS (+ 1)
            (stream points)))))
    rmax  = foldS max 0 (stream rfilt)
in (rfilt, rmax)

```

RULE "stream/unstream"

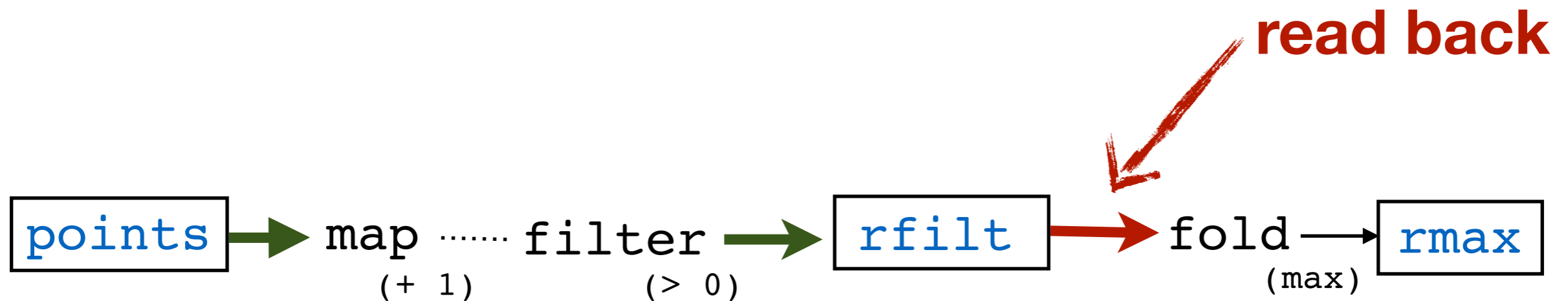
forall xs. stream (unstream xs) = xs


```
filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let
    rfilt = unstream (filterS (> 0) (mapS (+ 1)
                                           (stream points)))
    rmax  = foldS max 0 (stream rfilt)
in (rfilt, rmax)
```

```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let
    rfilt = unstream (filterS (> 0) (mapS (+ 1)
                                     (stream points)))
    rmax   = foldS max 0 (stream rfilt)
in  (rfilt, rmax)

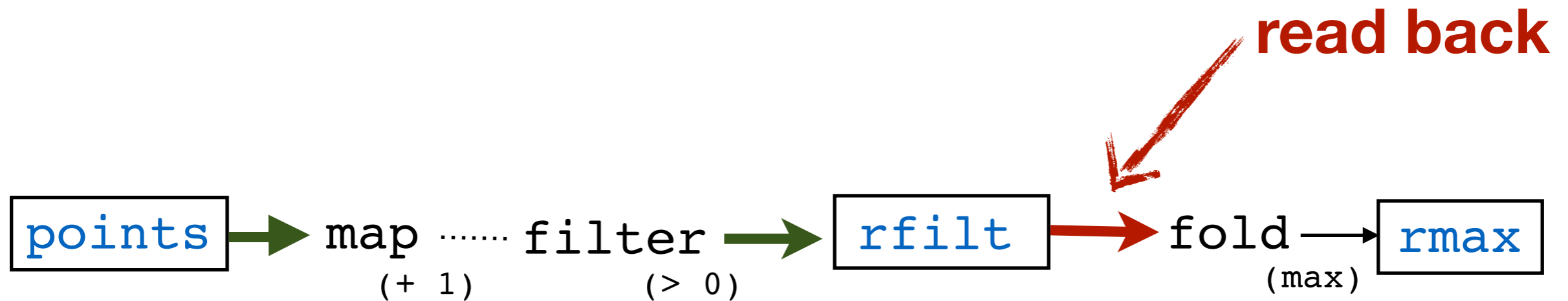
```



Back to SQL

```
create table rfilt as
select y + 1 as h
from points
where h > 0
```

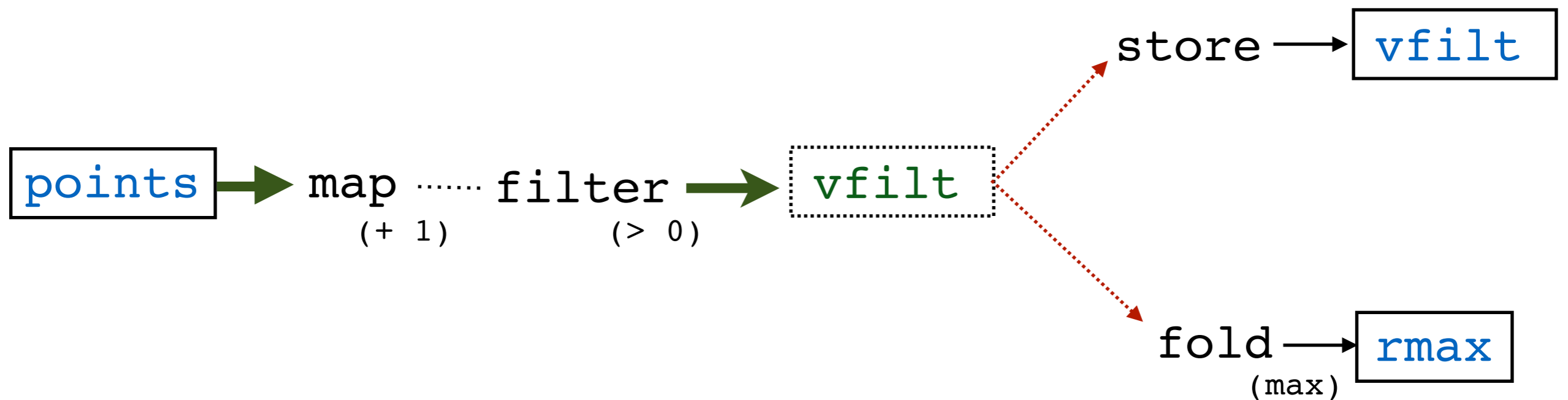
```
create table rmax as
select max(h) from rfilt
```



```
create view vfilt as
select y + 1 as h
from points
where h > 0

create table rfilt as
select * from vfilt

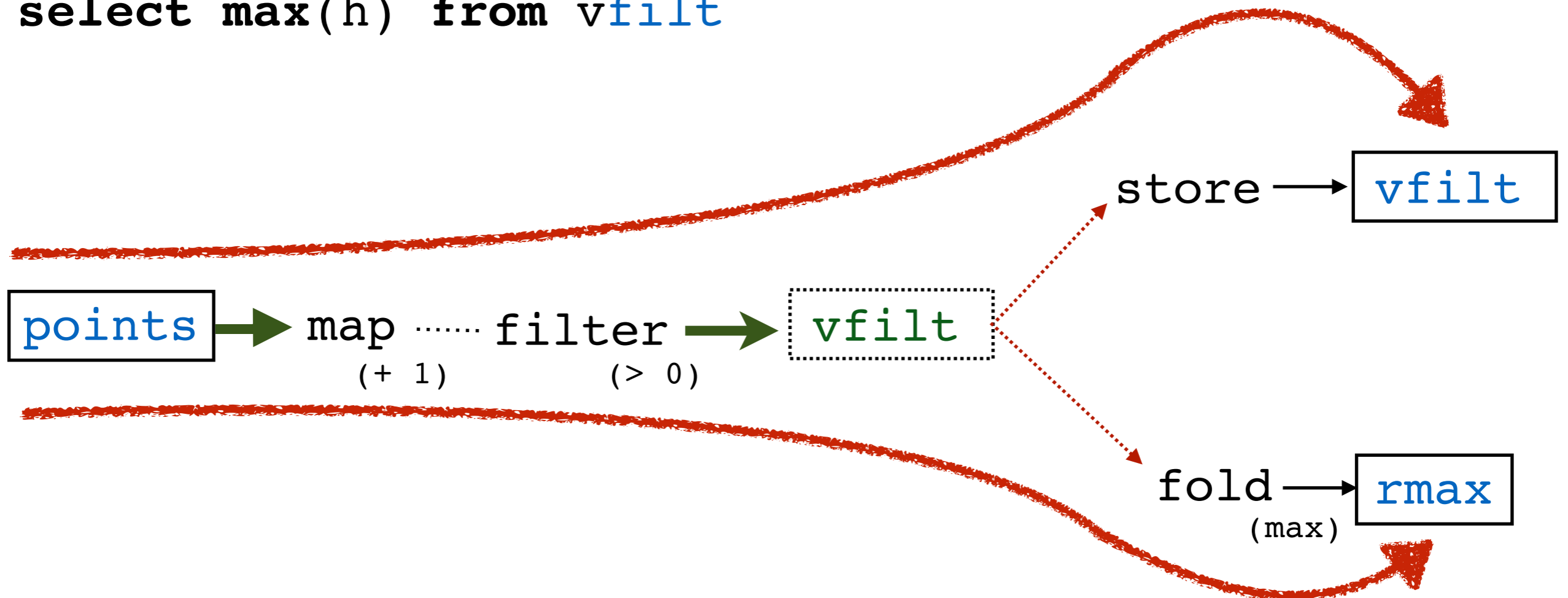
create table rmax as
select max(h) from vfilt
```



```
create view vfilt as
select y + 1 as h
from points
where h > 0

create table rfilt as
select * from vfilt

create table rmax as
select max(h) from vfilt
```

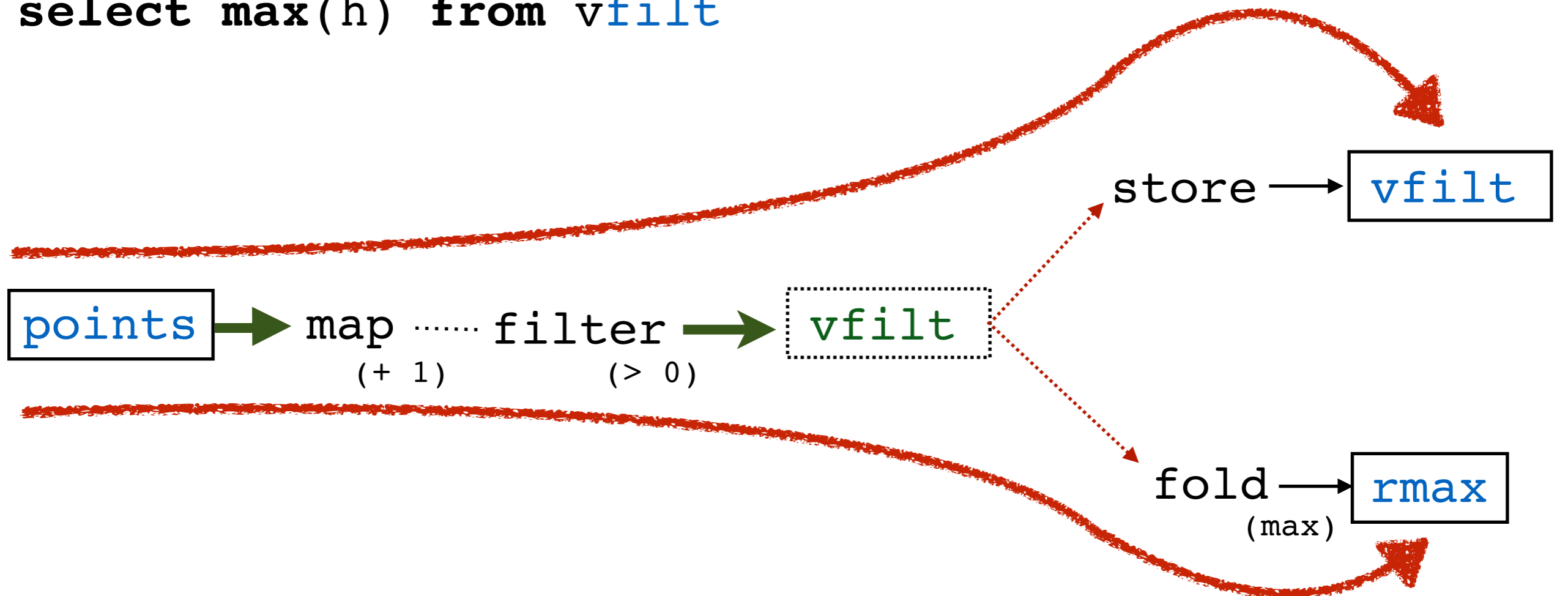


Apache Hive

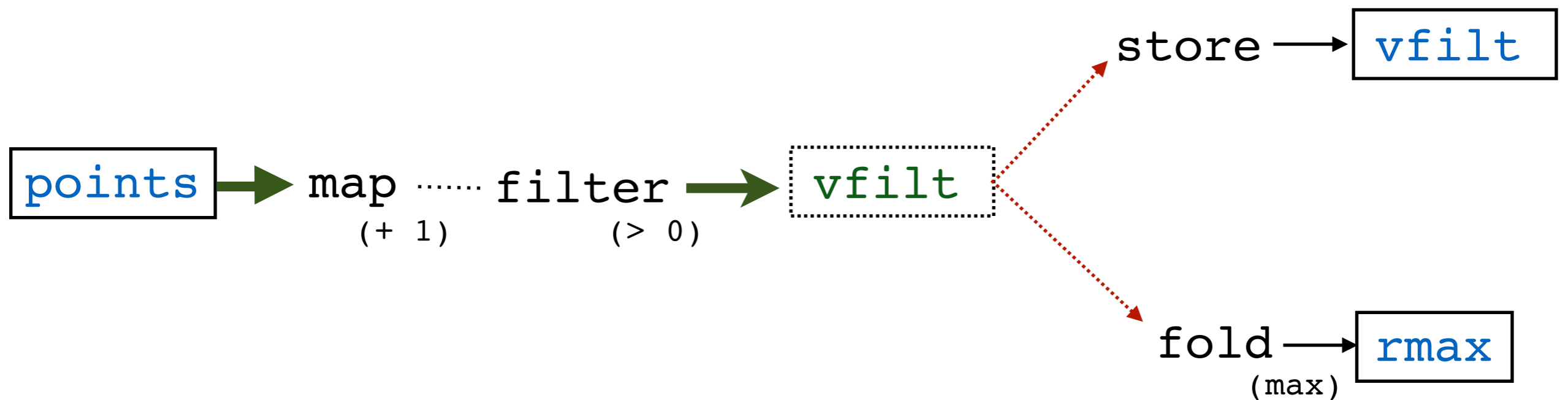
```
create view vfilt as
select y + 1 as h
from points
where h > 0

create table rfilt as
select * from vfilt

create table rmax as
select max(h) from vfilt
```




```
from (select y + 1 as h
      from points
      where h > 0) vfilt
insert overwrite table rfilt
select vfilt.h
insert overwrite table rmax
select max(vfilt.h)
```



Data Flow Fusion

Data Flow Fusion

- Guaranteed fusion for a particular class of queries.
- Can evaluate these in constant space with single imperative loops.
- Straightforward to do native code generation.

Data Flow Fusion

1. **Refactor** to expose desired data flow.
(currently working to make this automatic)
2. **Slurp** out a data flow graph from the source.
3. **Schedule** the graph into an abstract loop nest.
4. **Extract** implementation code from the nest.

```
filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let t2      = map (+ 1) points
      rfilt  = filter (> 0) t2
      rmax   = fold max 0 rfilt
  in (rfilt, rmax)
```

```
filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= run vec1 (\s1 ->
  let s2      = map (+ 1) s1
      flags = map (> 0) s2
  in mkSel flags (\sel ->
    let s3      = pack sel s2
        vec3 = store s3
            n      = fold max 0 s3
    in (vec3, n)))
```

```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= run vec1 (\s1 ->
  let s2      = map (+ 1) s1
      flags  = map (> 0) s2
  in mkSel flags (\sel ->
    let s3      = pack sel s2
        vec3    = store s3
            n    = fold max 0 s3
    in (vec3, n)))

```

```

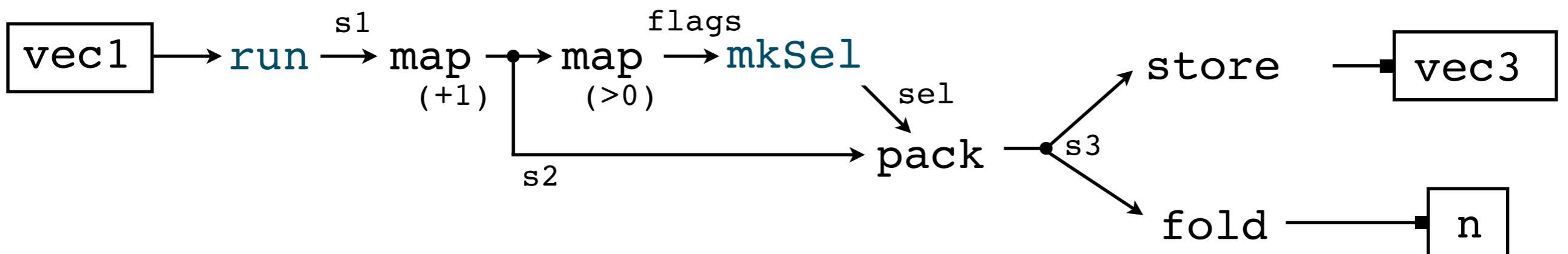
pack (Sel [T F F T F]) [1 2 3 4 5] = [1 4]

```

```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= run vec1 (\s1 ->
  let s2      = map (+ 1) s1
      flags  = map (> 0) s2
  in mkSel flags (\sel ->
    let s3    = pack sel s2
        vec3  = store s3
            n  = fold max 0 s3
    in (vec3, n)))

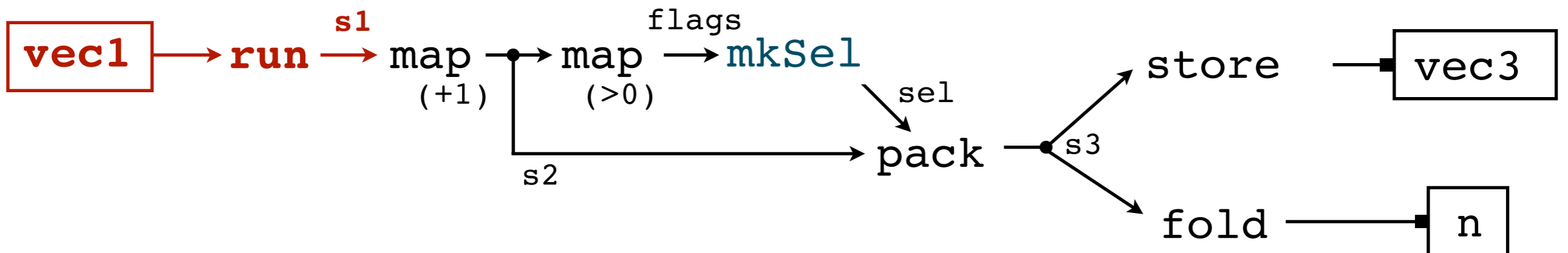
```




```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= run vec1 (\s1 ->
  let s2      = map (+ 1) s1
      flags  = map (> 0) s2
  in mkSel flags (\sel ->
    let s3    = pack sel s2
        vec3  = store s3
            n  = fold max 0 s3
    in (vec3, n)))

```

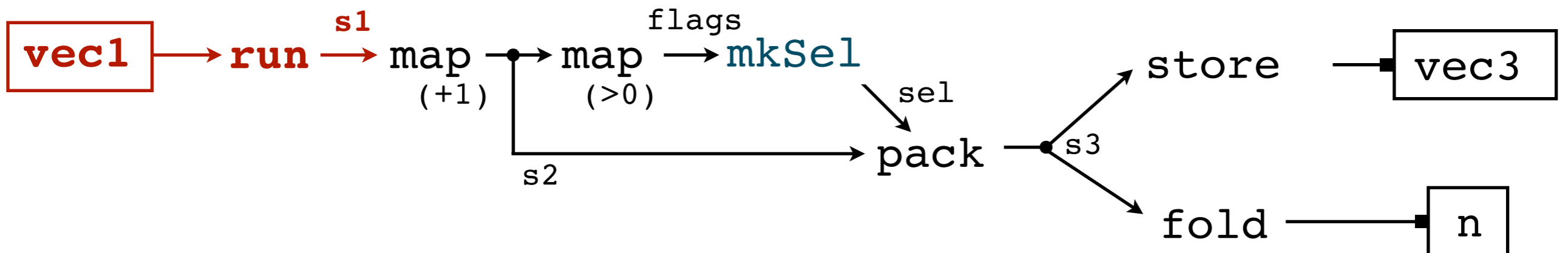


```
filterMax :: Vector Int -> (Vector Int, Int)
```

```
filterMax vec1
```

```
= run vec1 (\s1 ->  
  let s2      = map (+ 1) s1  
      flags   = map (> 0) s2  
  in mkSel flags (\sel ->  
    let s3     = pack sel s2  
        vec3   = store s3  
        n      = fold max 0 s3  
    in (vec3, n)))
```

```
s1 :: Series k1 Int
```



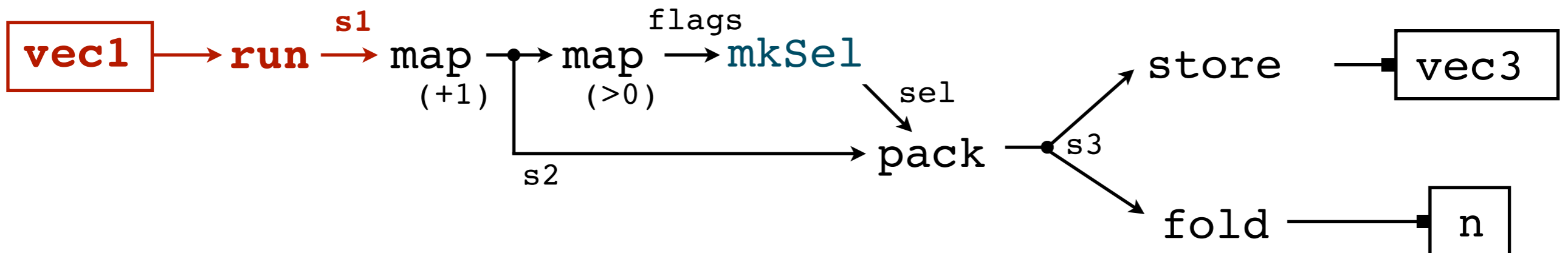
```
filterMax :: Vector Int -> (Vector Int, Int)
```

```
filterMax vec1
```

```
= run vec1 (\s1 ->  
  let s2      = map (+ 1) s1  
      flags  = map (> 0) s2  
  in mkSel flags (\sel ->  
    let s3      = pack sel s2  
        vec3   = store s3  
        n       = fold max 0 s3  
    in (vec3, n)))
```

```
s1 :: Series k1 Int
```

Rate Variable



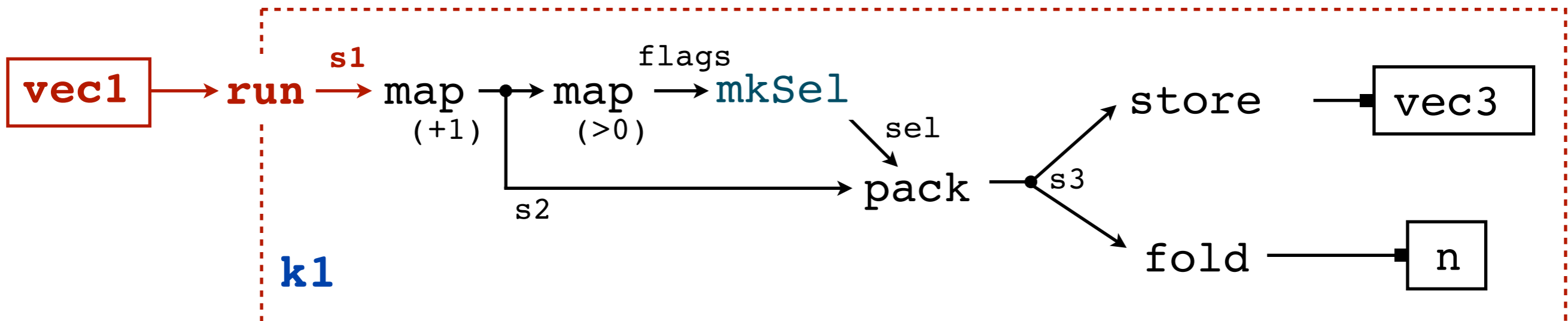
```
filterMax :: Vector Int -> (Vector Int, Int)
```

```
filterMax vec1
```

```
= run vec1 (\s1 ->  
  let s2      = map (+ 1) s1  
      flags   = map (> 0) s2  
  in mkSel flags (\sel ->  
    let s3     = pack sel s2  
        vec3   = store s3  
        n      = fold max 0 s3  
    in (vec3, n)))
```

```
s1 :: Series k1 Int
```

Rate Variable



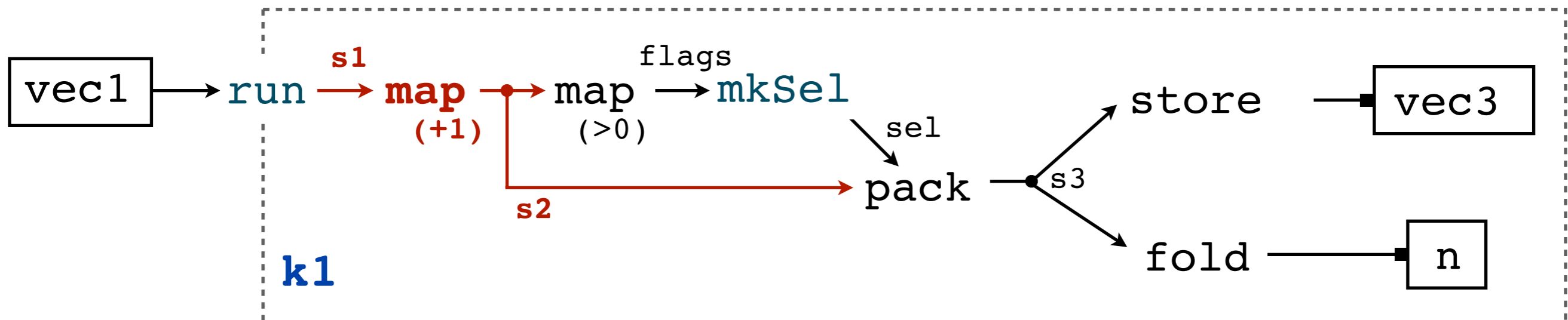
```
filterMax :: Vector Int -> (Vector Int, Int)
```

```
filterMax vec1
```

```
= run vec1 (\s1 ->  
  let s2      = map (+ 1) s1  
      flags = map (> 0) s2  
  in mkSel flags (\sel ->  
    let s3      = pack sel s2  
        vec3 = store s3  
        n      = fold max 0 s3  
    in (vec3, n)))
```

```
s1 :: Series k1 Int
```

```
s2 :: Series k1 Int
```

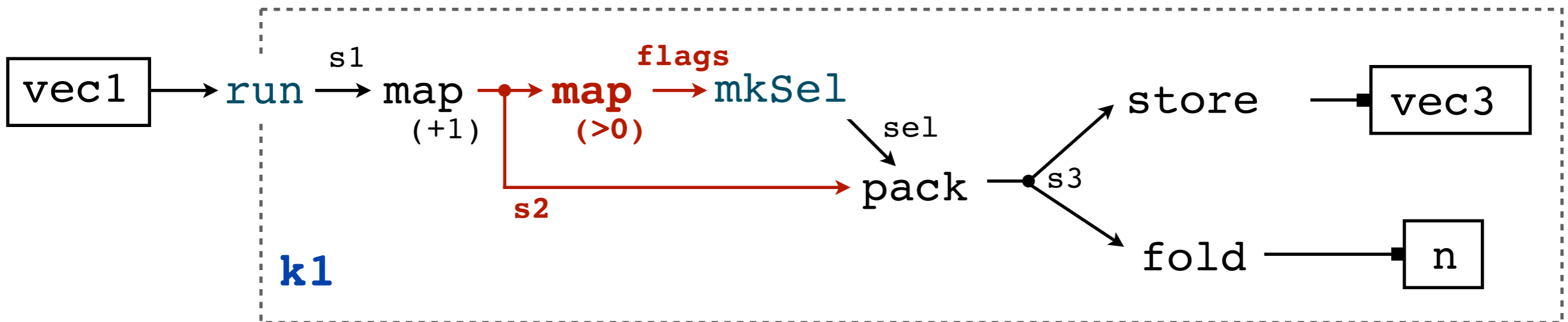


```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
  = run vec1 (\s1 ->
    let s2      = map (+ 1) s1
        flags  = map (> 0) s2
    in mkSel flags (\sel ->
      let s3    = pack sel s2
          vec3  = store s3
              n  = fold max 0 s3
      in (vec3, n)))

```

s1 :: Series **k1** Int
 s2 :: Series **k1** Int
 flags :: Series **k1** Bool



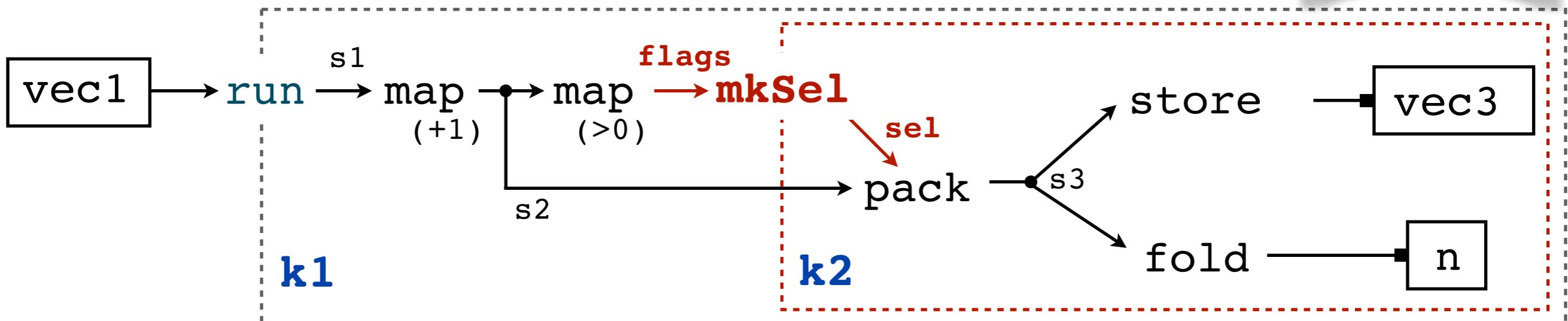
```
filterMax :: Vector Int -> (Vector Int, Int)
```

```
filterMax vec1
```

```
= run vec1 (\s1 ->
  let s2      = map (+ 1) s1
      flags   = map (> 0) s2
  in mkSel flags (\sel ->
    let s3     = pack sel s2
        vec3   = store s3
        n      = fold max 0 s3
    in (vec3, n)))
```

`s1 :: Series k1 Int`
`s2 :: Series k1 Int`
`flags :: Series k1 Bool`
`sel :: Sel k1 k2`

`k1 >= k2`



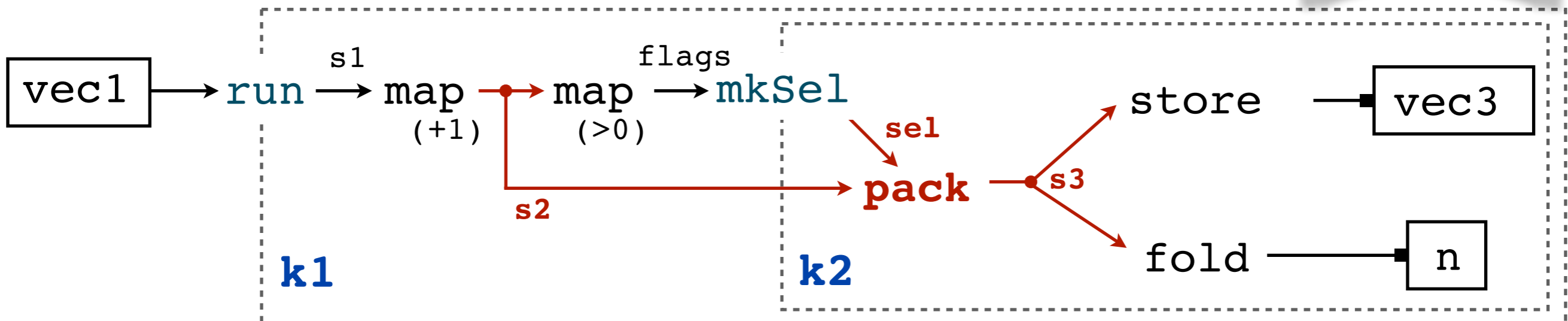
```
filterMax :: Vector Int -> (Vector Int, Int)
```

```
filterMax vec1
```

```
= run vec1 (\s1 ->
  let s2      = map (+ 1) s1
      flags  = map (> 0) s2
  in mkSel flags (\sel ->
    let s3      = pack sel s2
        vec3  = store s3
        n     = fold max 0 s3
    in (vec3, n)))
```

`s1 :: Series k1 Int`
`s2 :: Series k1 Int`
`flags :: Series k1 Bool`
`sel :: Sel k1 k2`
`s3 :: Series k2 Int`

`k1 >= k2`



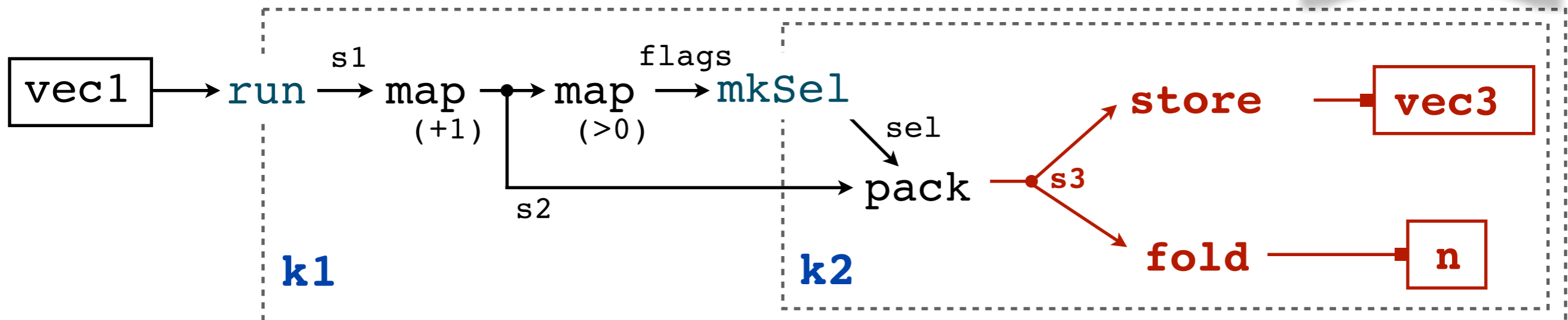

```
filterMax :: Vector Int -> (Vector Int, Int)
```

```
filterMax vec1
```

```
= run vec1 (\s1 ->
  let s2      = map (+ 1) s1
      flags  = map (> 0) s2
  in mkSel flags (\sel ->
    let s3      = pack sel s2
        vec3 = store s3
        n    = fold max 0 s3
    in (vec3, n)))
```

`s1 :: Series k1 Int`
`s2 :: Series k1 Int`
`flags :: Series k1 Bool`
`sel :: Sel k1 k2`
`s3 :: Series k2 Int`
`vec3 :: Vector Int`
`n :: Int`

`k1 >= k2`



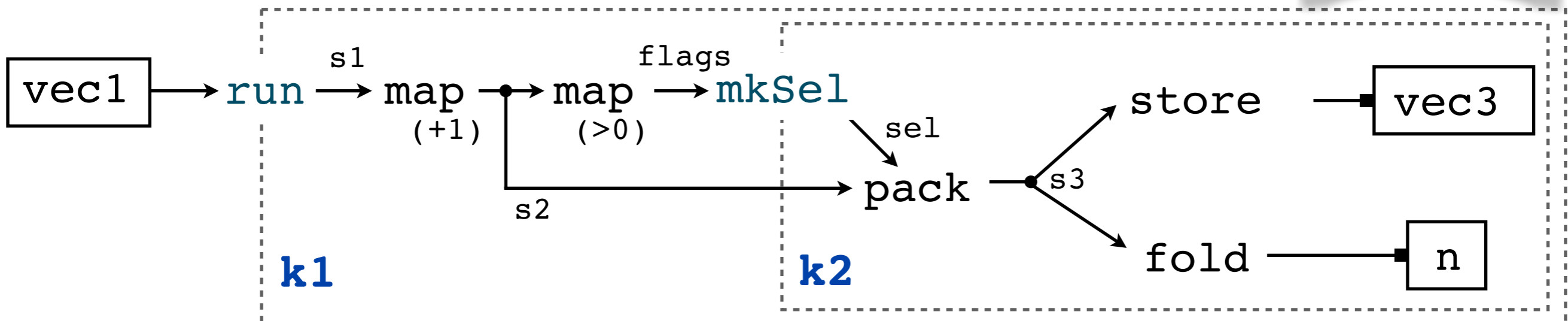
```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
  = run vec1 (\s1 ->
    let s2      = map (+ 1) s1
        flags  = map (> 0) s2
    in mkSel flags (\sel ->
      let s3      = pack sel s2
          vec3    = store s3
              n    = fold max 0 s3
      in (vec3, n)))

```

s1 :: Series **k1** Int
 s2 :: Series **k1** Int
 flags :: Series **k1** Bool
 sel :: Sel **k1** **k2**
 s3 :: Series **k2** Int
 vec3 :: Vector Int
 n :: Int

k1 >= k2

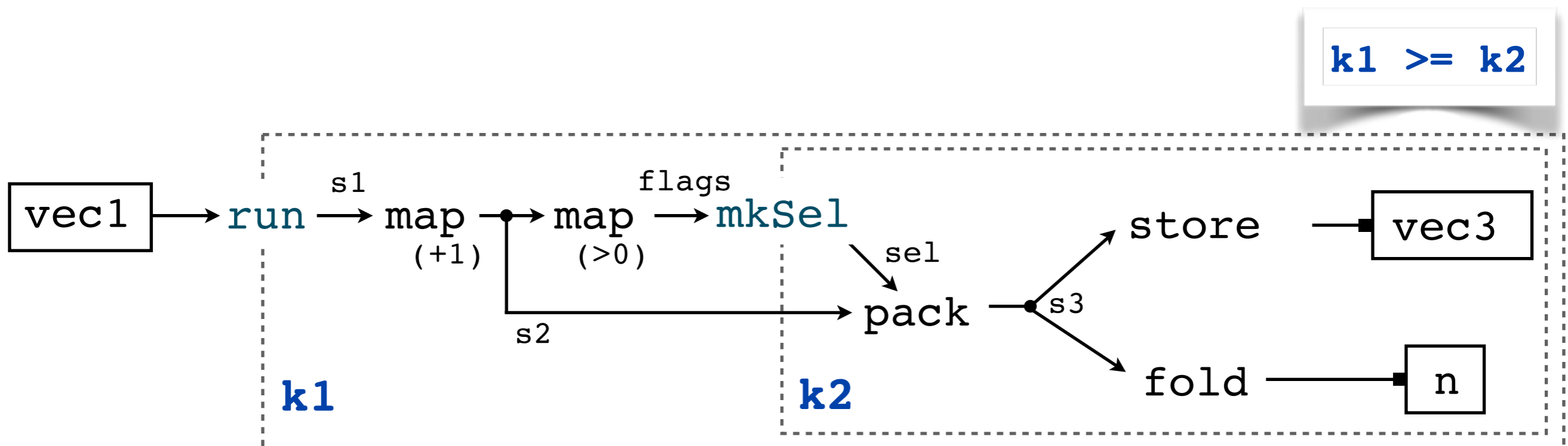


```

run    :: Vector a
      -> (forall k1. Series k1 a -> b) -> b

mkSel  :: Series k1 Bool
      -> (forall k2. Sel k1 k2      -> b) -> b

```



```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
  = loop k1
    { start: ...

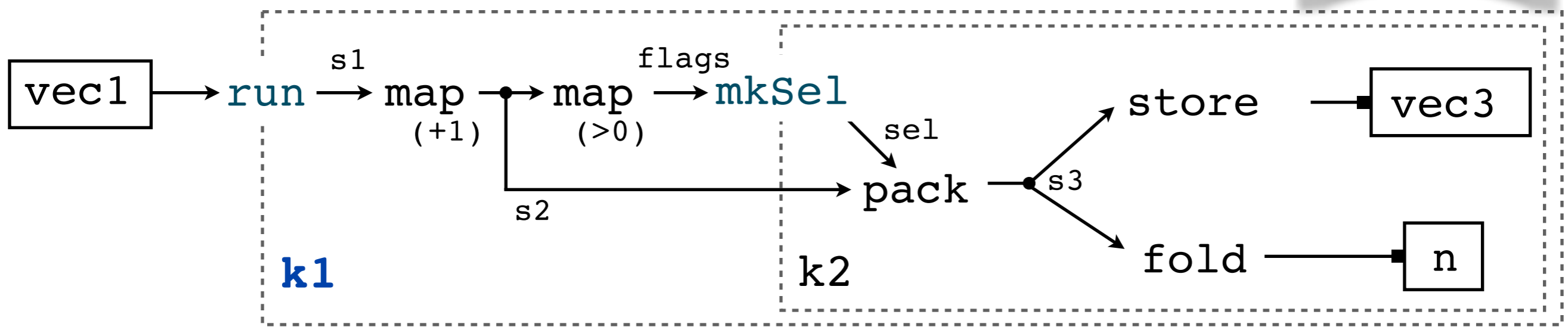
      body: ...

      inner: ...

      end: ...
    } yields ...

```

k1 >= k2



```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: ...

    body:  x1    = next k1 vec1

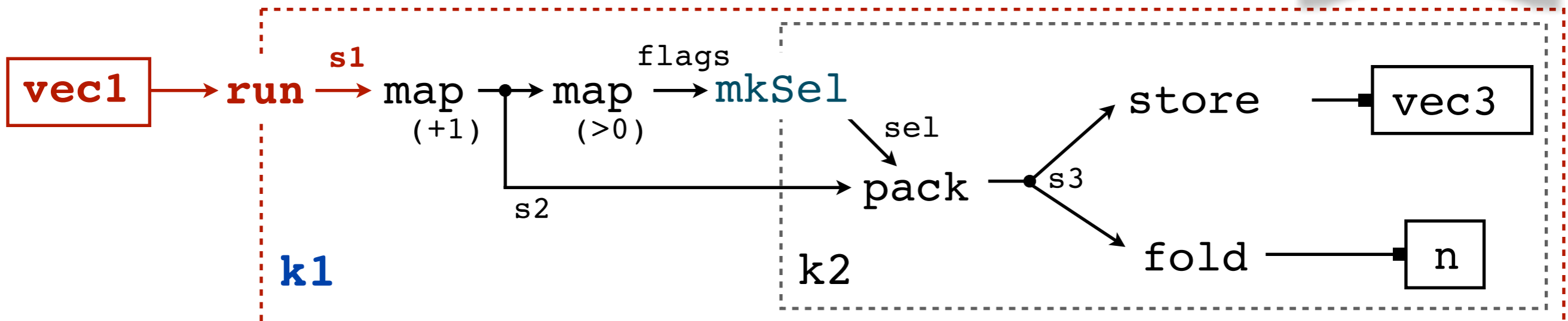
    inner: ...

    end:   ...

  } yields ...

```

k1 >= k2



```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: ...

    body:  x1    = next k1 vec1
           x2    = (+ 1) x1

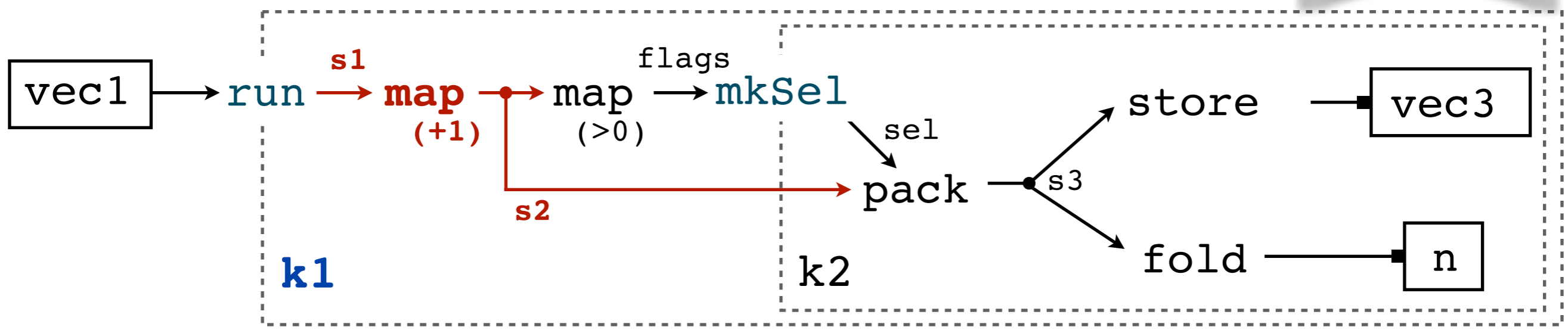
    inner: ...

    end:    ...

  } yields ...

```

k1 >= k2



```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: ...

  body:  x1    = next k1 vec1
         x2    = (+ 1) x1
         xf   = (> 0) x1

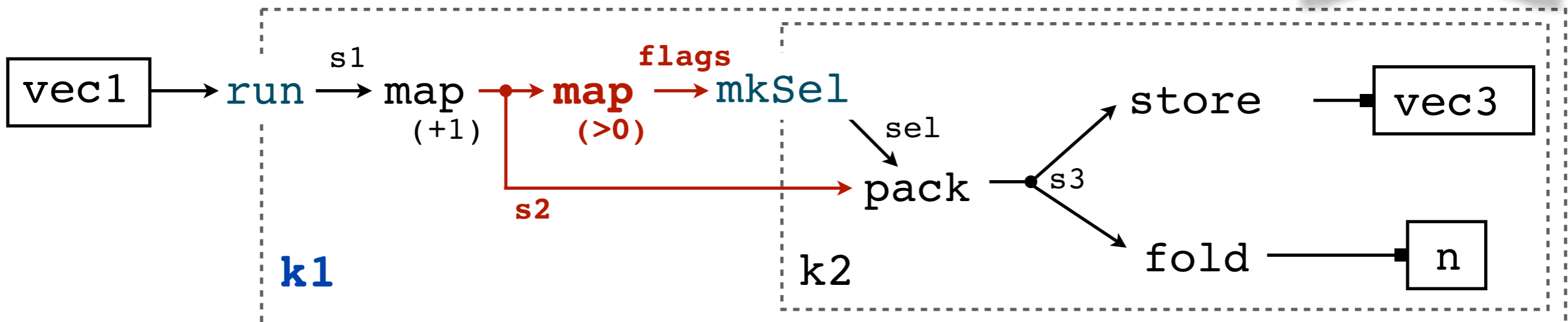
  inner: ...

  end:   ...

} yields ...

```

k1 >= k2



```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: ...

  body:  x1    = next k1 vec1
         x2    = (+ 1) x1
         xf    = (> 0) x1

  inner: guard k2 xf
         { body: ...

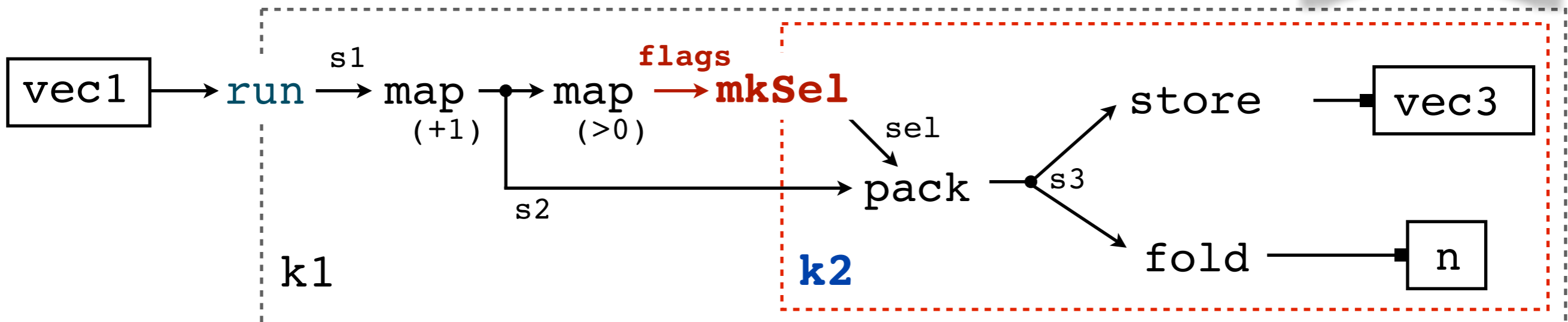
  }

  end:  ...

} yields ...

```

k1 >= k2




```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: ...

  body:  x1    = next k1 vec1
        x2    = (+ 1) x1
        xf    = (> 0) x1

  inner: guard k2 xf
        { body: x3      = x2

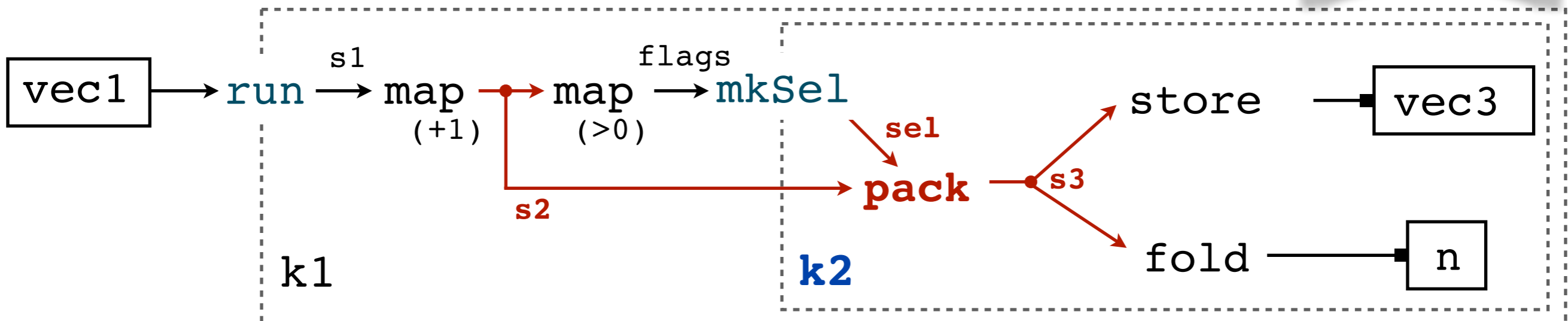
        }

  end:  ...

} yields ...

```

k1 >= k2



```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: vec3 = newVec k2

  body:  x1    = next k1 vec1
         x2    = (+ 1) x1
         xf    = (> 0) x1

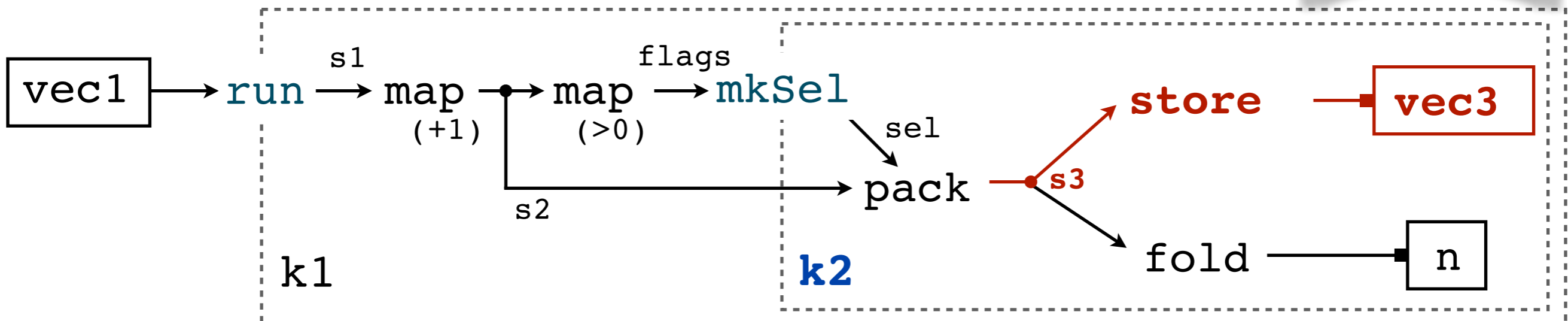
  inner: guard k2 xf
         { body: x3      = x2
           write k2 vec3 x3
         }

  end:  slice k2 vec3

} yields ...

```

k1 >= k2

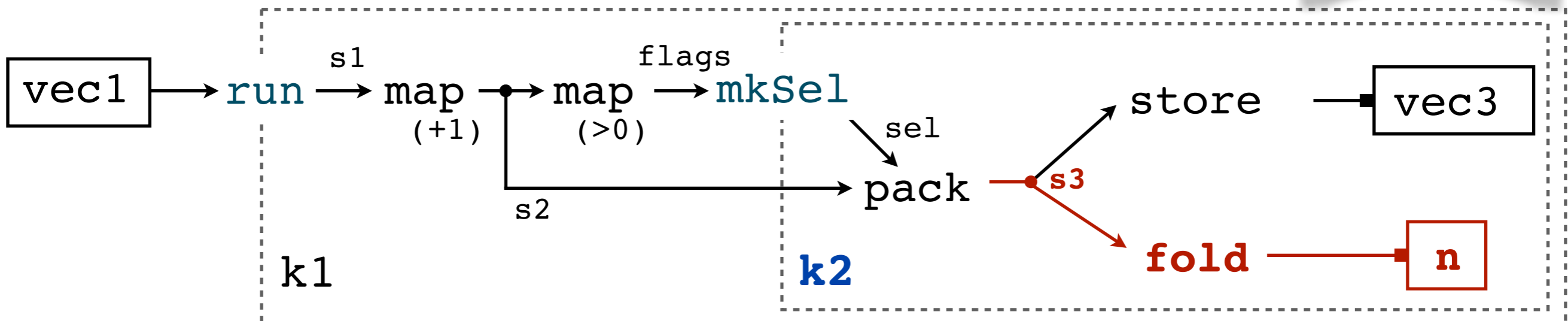


```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: vec3 = newVec k2
    nAcc = newAcc 0
  body: x1 = next k1 vec1
        x2 = (+ 1) x1
        xf = (> 0) x1
  inner: guard k2 xf
        { body: x3 = x2
          write k2 vec3 x3
          nAcc := (+) nAcc x3
        }
  end: slice k2 vec3
      n = readAcc nAcc
} yields ...

```

k1 >= k2

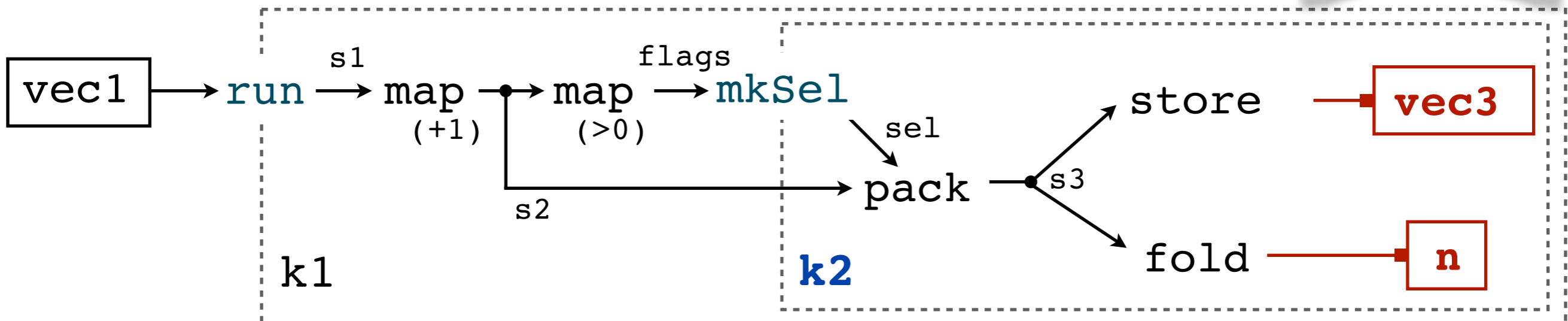


```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: vec3 = newVec k2
    nAcc = newAcc 0
  body: x1 = next k1 vec1
        x2 = (+ 1) x1
        xf = (> 0) x1
  inner: guard k2 xf
        { body: x3 = x2
          write k2 vec3 x3
          nAcc := (+) nAcc x3
        }
  end: slice k2 vec3
       n = readAcc nAcc
} yields (vec3, n)

```

$k1 \geq k2$



```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: vec3 = newVec k2
        nAcc = newAcc 0
    body:  x1   = next k1 vec1
            x2   = (+ 1) x1
            xf   = (> 0) x1
    inner: guard k2 xf
            { body: x3   = x2
                write k2 vec3 x3
                nAcc := (+) nAcc x3
            }
    end:  slice k2 vec3
            n     = readAcc nAcc
  } yields (vec3, n)

```

```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: vec3 = newVec k2
    nAcc = newAcc 0
  body: x1 = next k1 vec1
    x2 = (+ 1) x1
    xf = (> 0) x1
  inner: guard k2 xf
    { body: x3 = x2
      write k2 vec3 x3
      nAcc := (+) nAcc x3
    }
  end: slice k2 vec3
    n = readAcc nAcc
} yields (vec3, n)

```

```

filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= loop k1
  { start: vec3 = newVec k2
    nAcc = newAcc 0
  body: x1 = next k1 vec1
    x2 = (+ 1) x1
    xf = (> 0) x1
  inner: guard k2 xf
    { body: x3 = x2
      write k2 vec3 x3
      nAcc := (+) nAcc x3
    }
  end: slice k2 vec3
    n = readAcc nAcc
} yields (vec3, n)

```

```

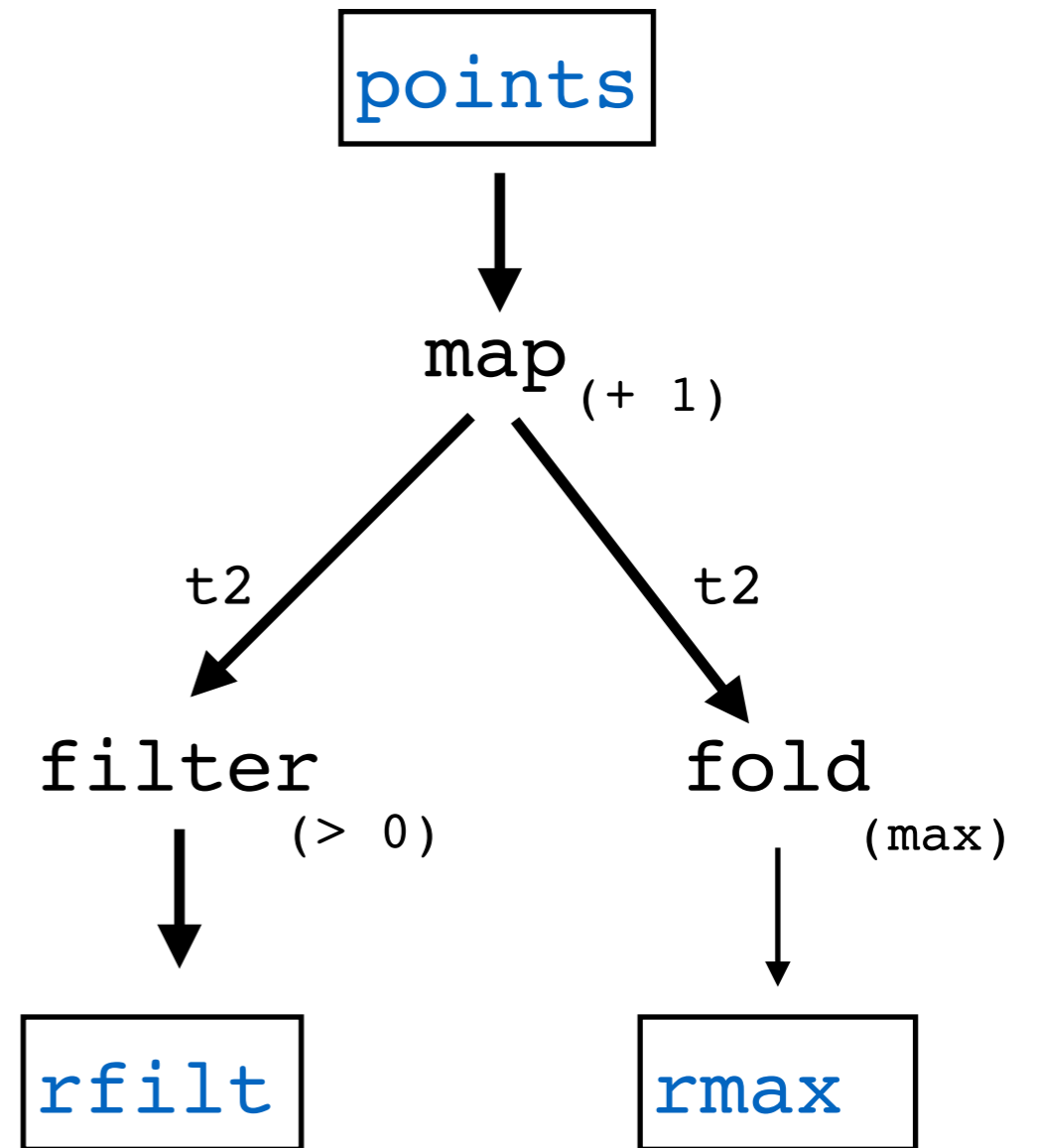
filterMax :: Vector Int -> (Vector Int, Int)
filterMax vec1
= do vec3  <- newVec (length vec1)
     nAcc  <- newAcc 0
     k2Acc <- newAcc 0
     loopM (length vec1)
         (\ix1 ->
            do x1      <- next ix1 vec1
               let x2 = (+ 1) x1
                   xf = (> 0) x1
                   guardM k2Acc xf (\ix2 ->
                      do let x3 = x2
                          write ix2 vec3 x3
                          modifyAcc nAcc (\x -> (+) x x3))
            sliceVec k2Acc vec3
     n      <- readAcc nAcc
     return (vec3, n)

```


Operator Clustering

```
filterMax :: Vector Int -> (Vector Int, Int)
filterMax points
= let t2      = map (+ 1) points
      rfilt   = filter (> 0) t2
      rmax    = fold max 0 rfilt
  in (rfilt, rmax)
```

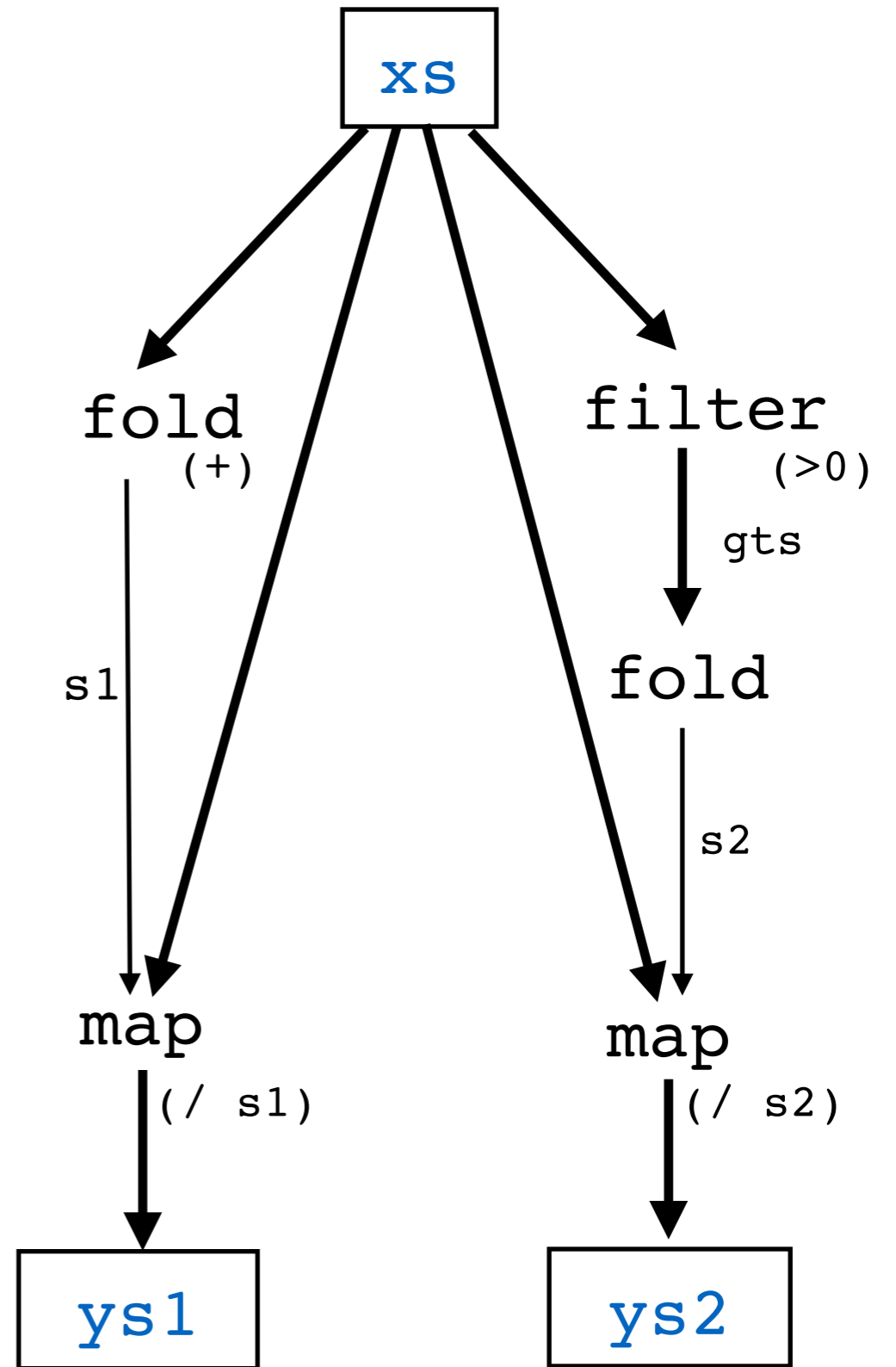
```
t2      = map (+ 1) points
rfilt   = filter (> 0) t2
rmax    = fold max 0 rfilt
```

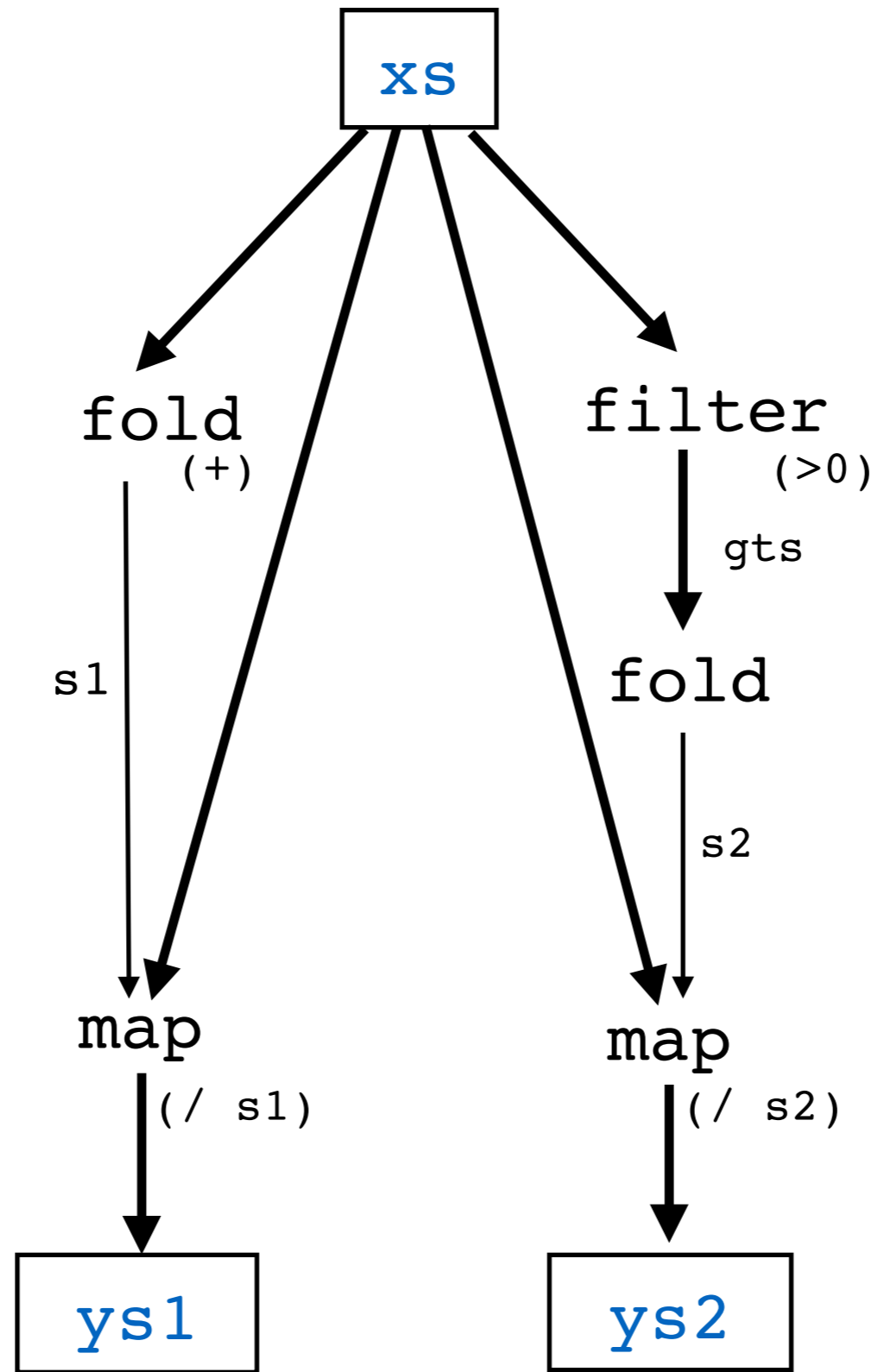


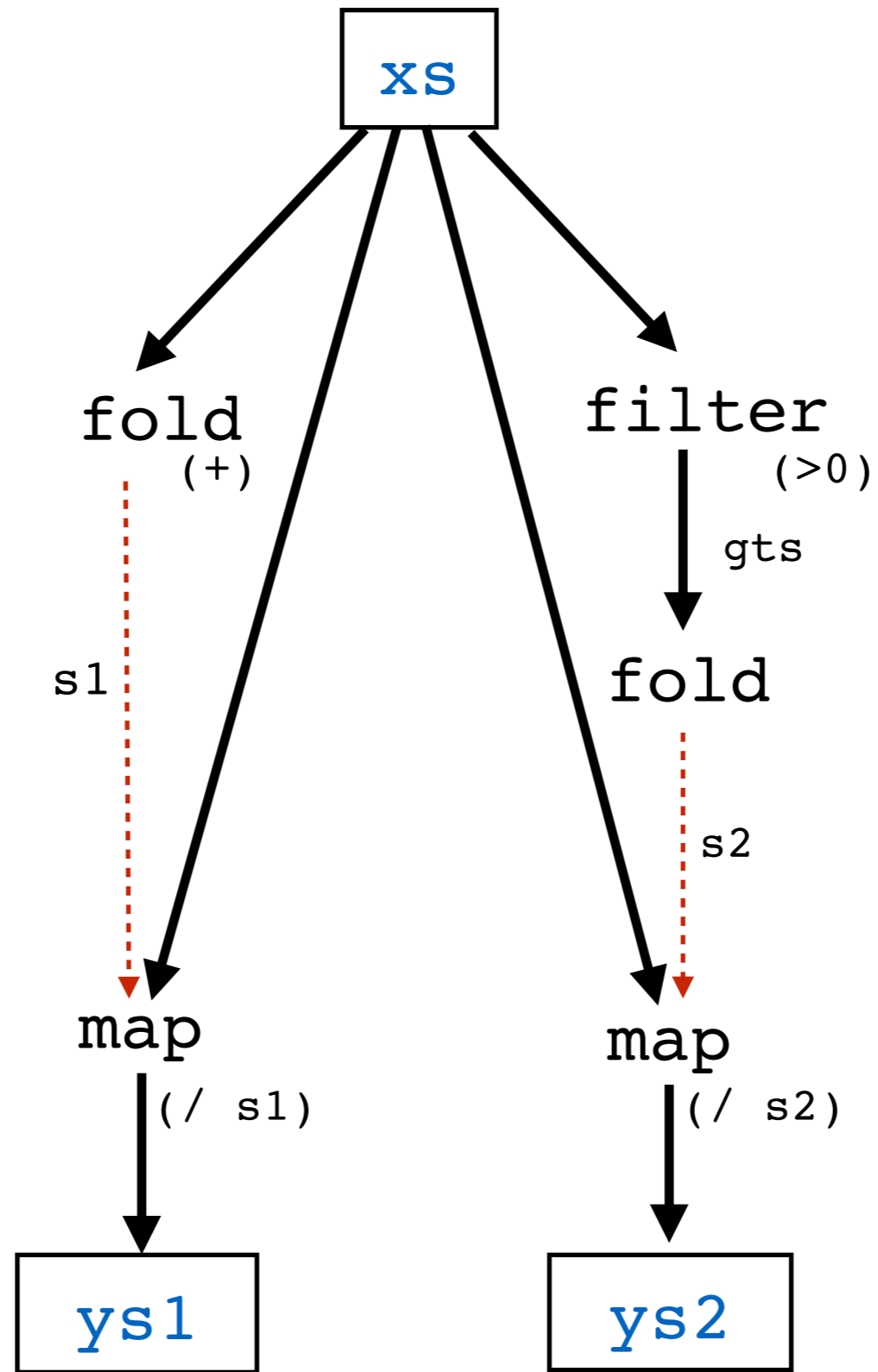
```

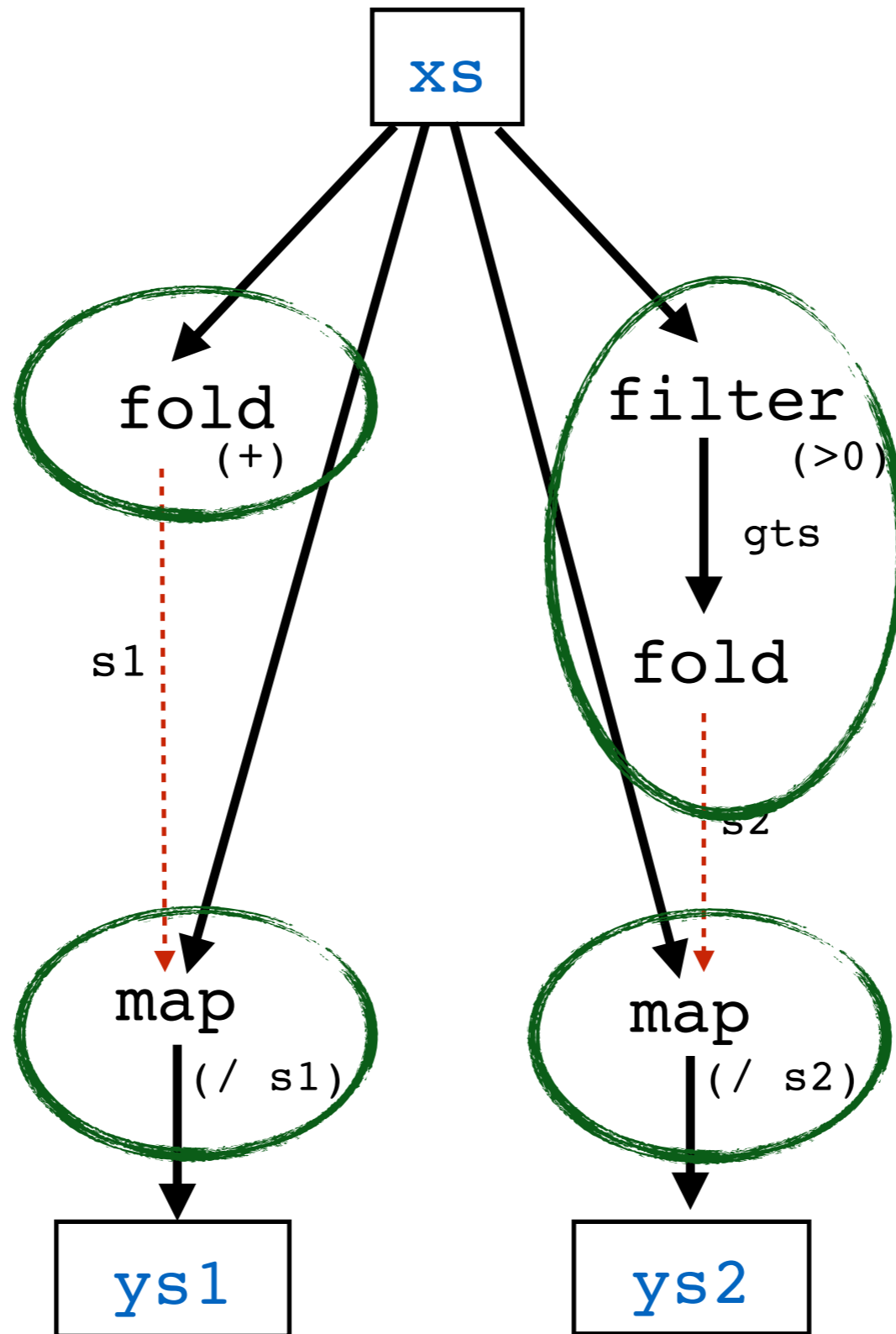
s1    = fold    (+) 0 xs
gts   = filter (> 0) xs
s2    = fold    (+) 0 gts
ys1   = map     (/ sum1) xs
ys2   = map     (/ sum2) xs

```

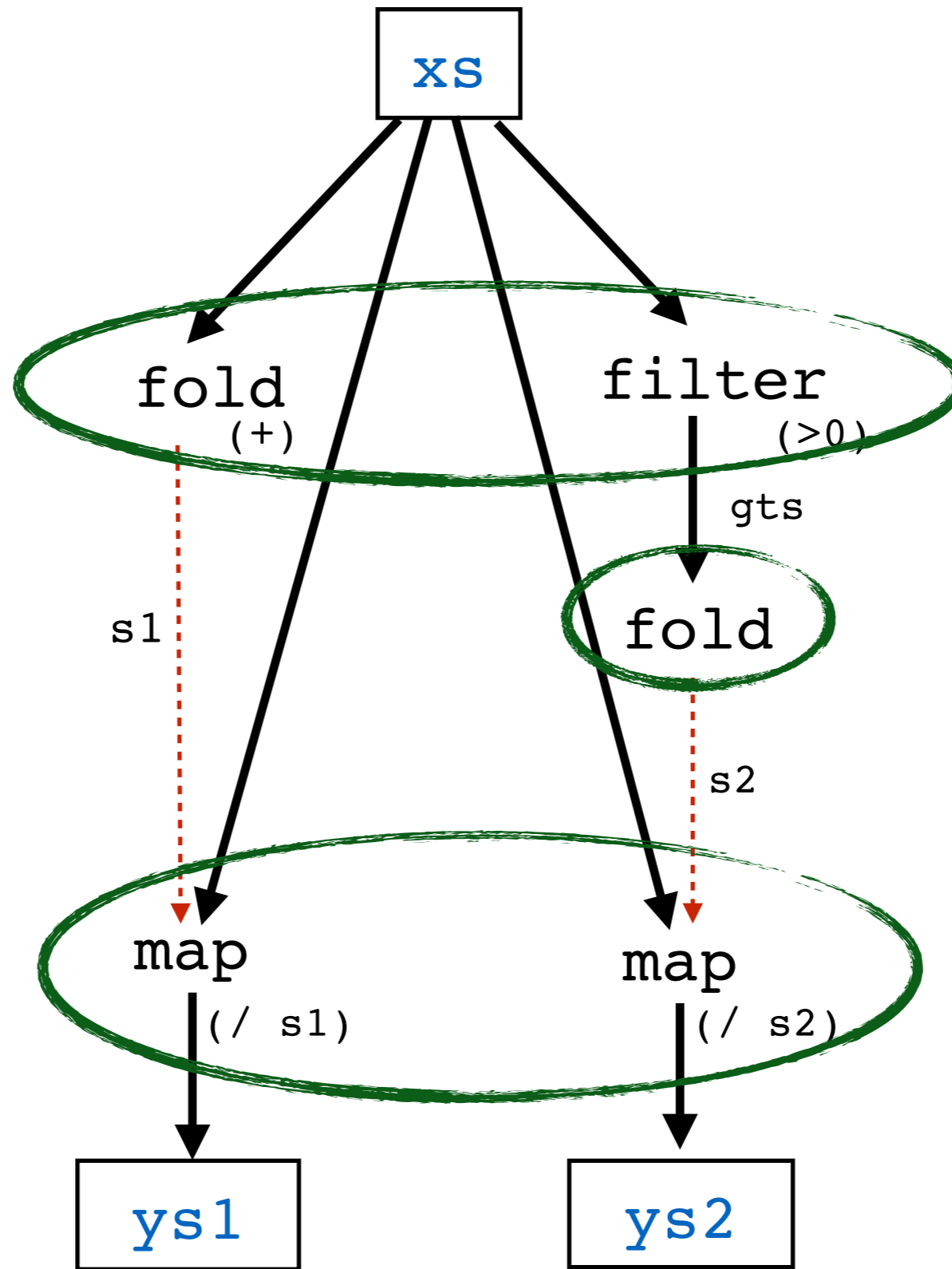


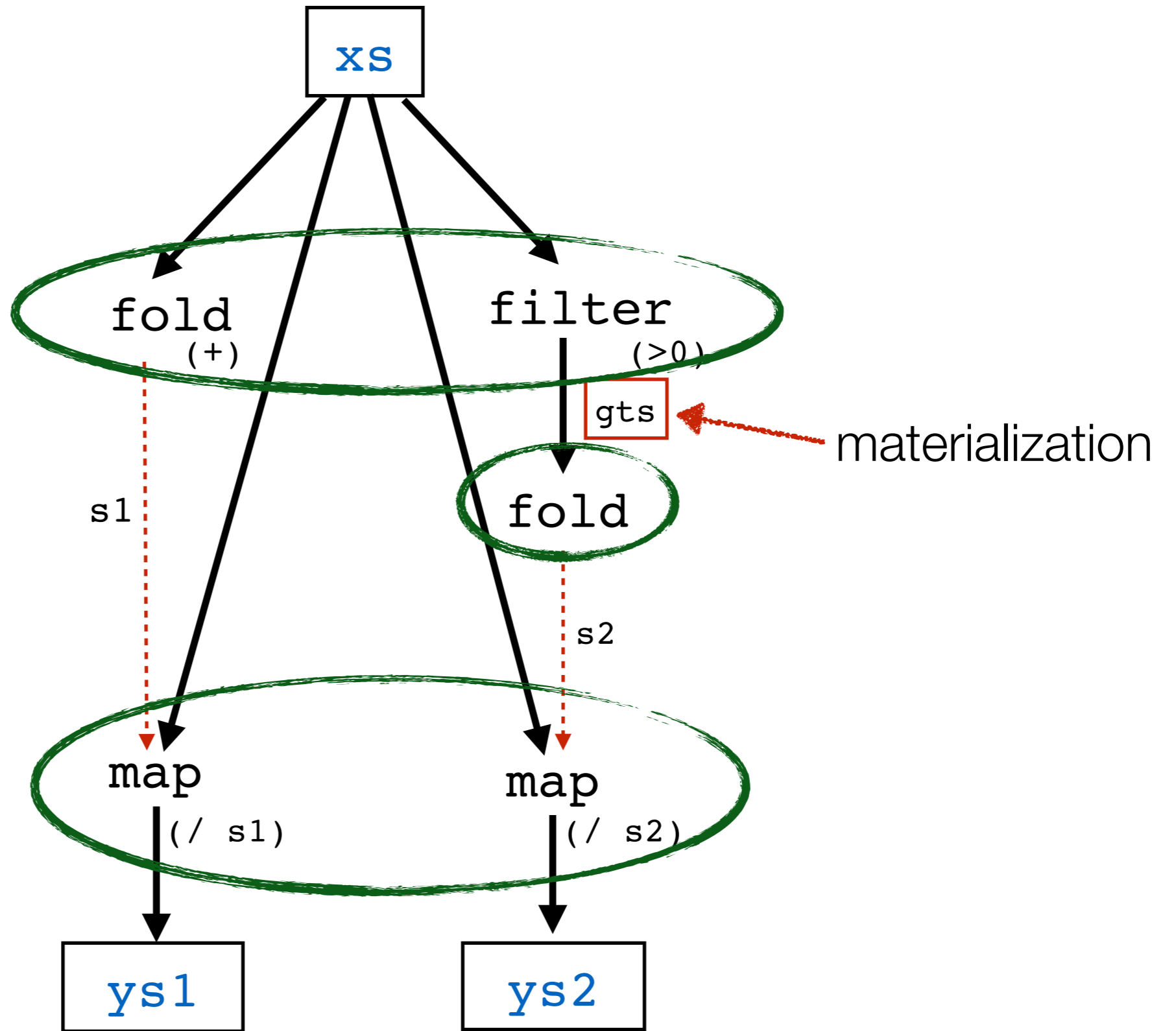


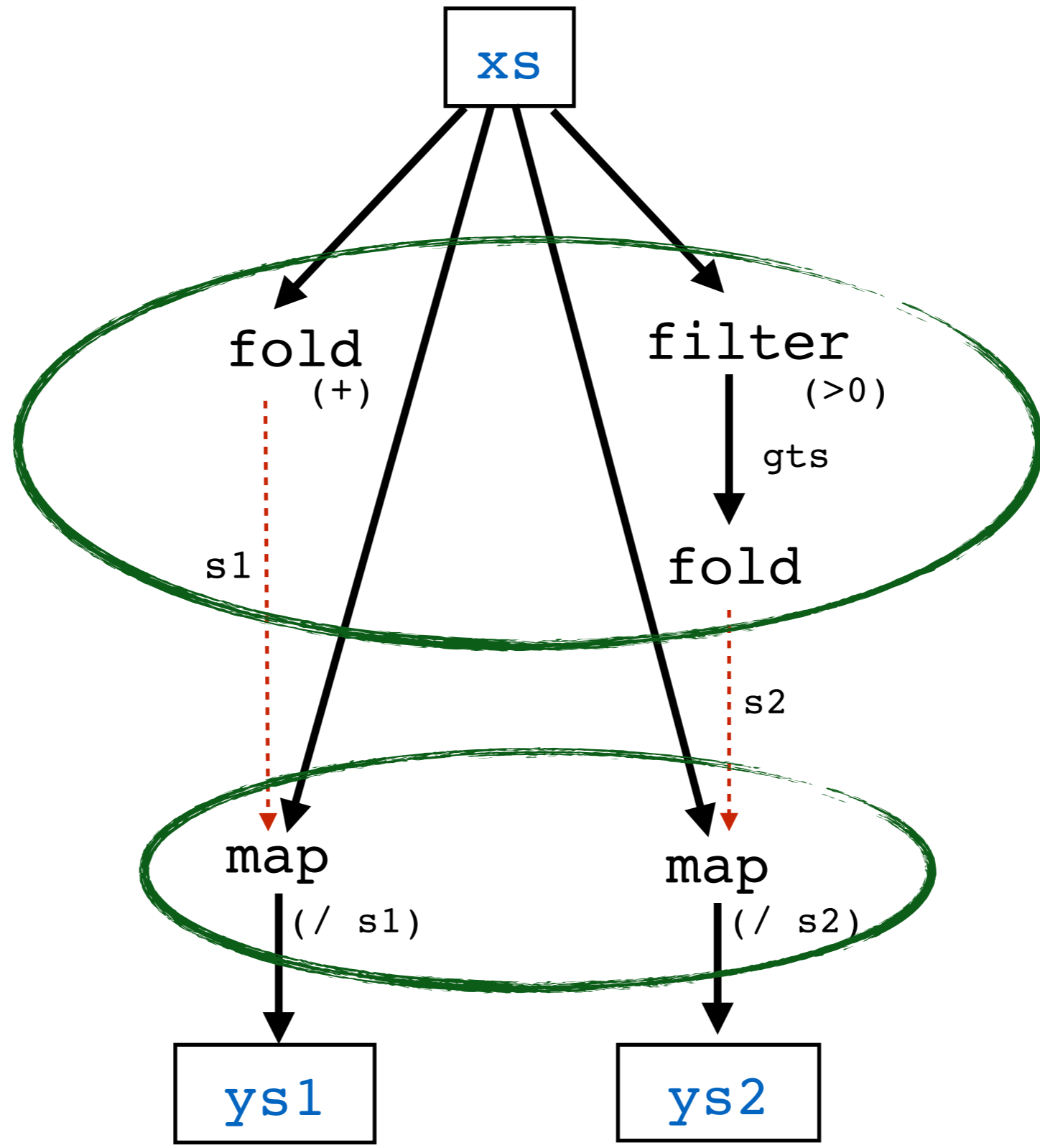




(what you'd get with stream fusion)





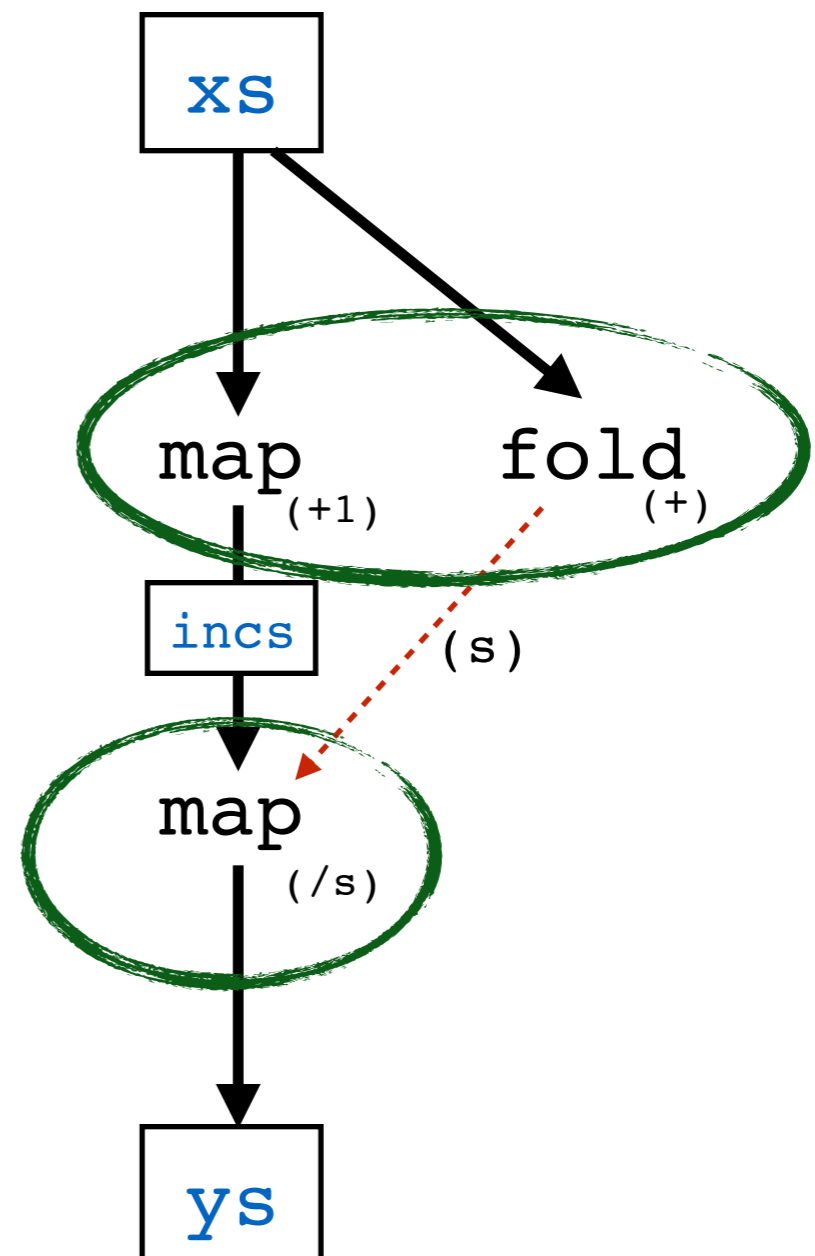
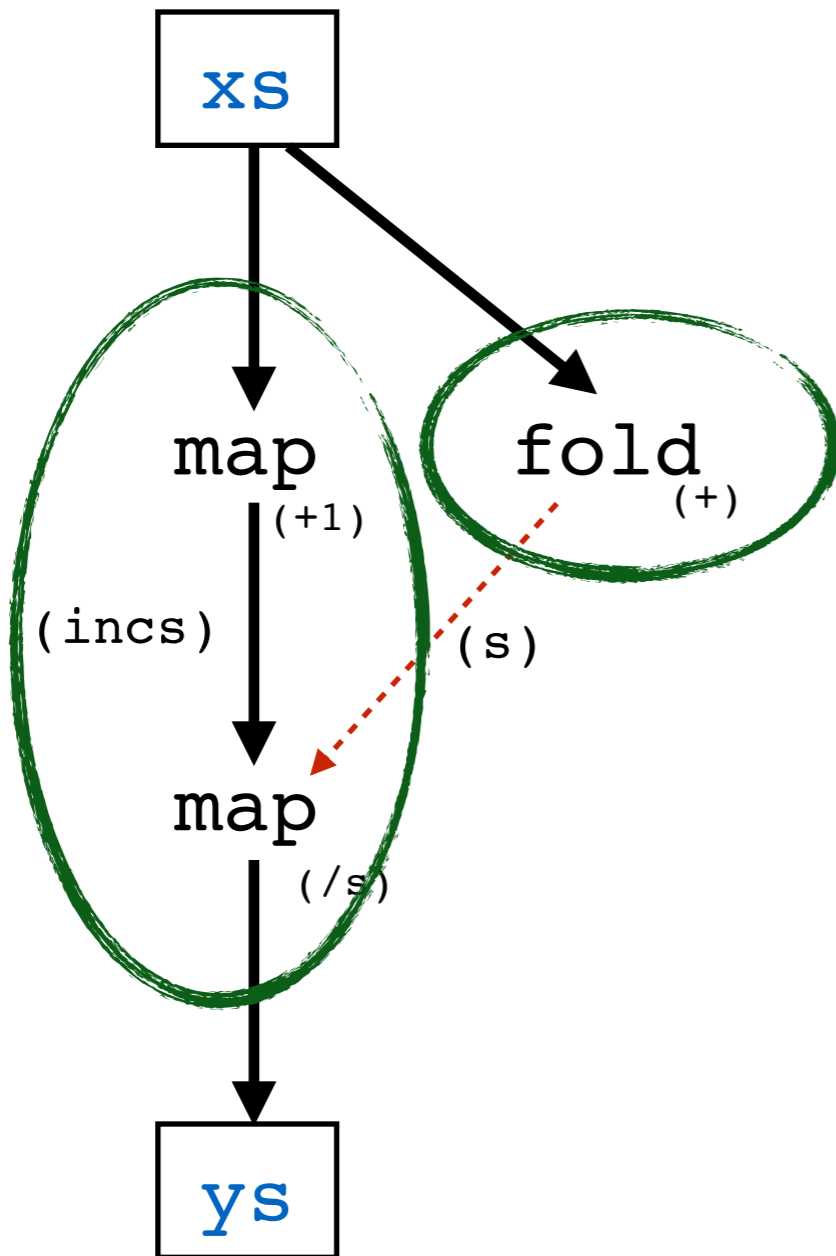


Minimize #access rather than #clusters

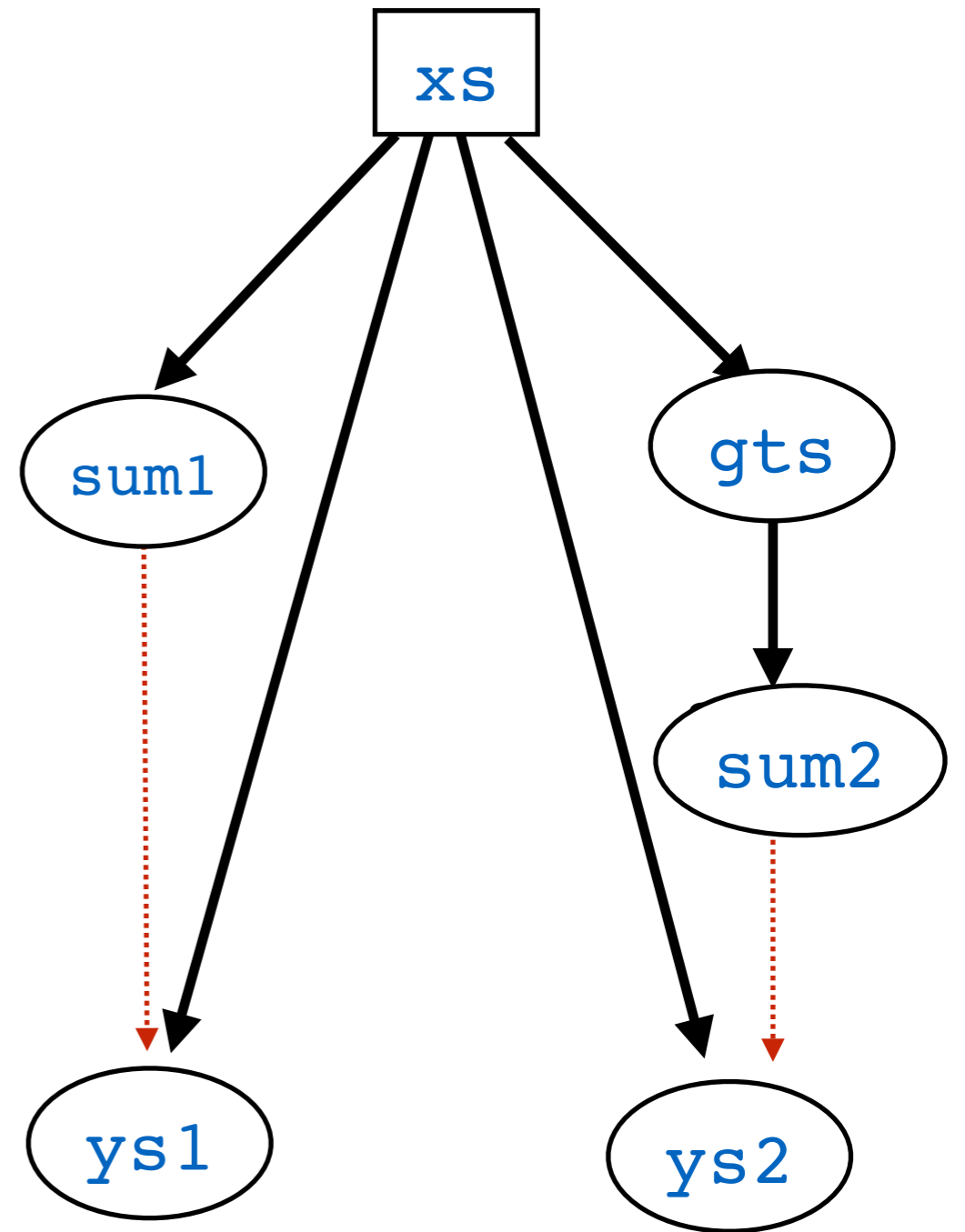
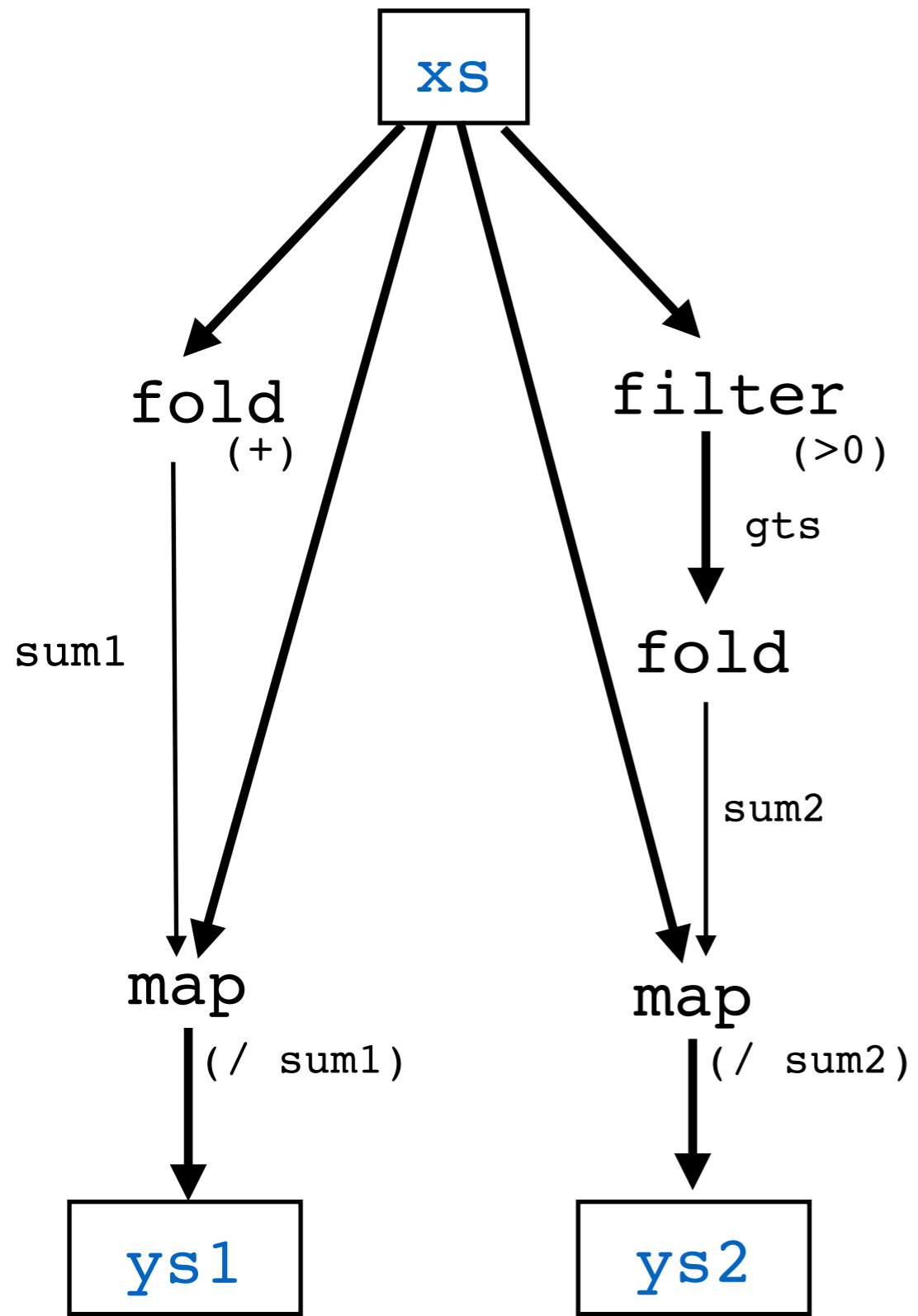
```

normalizeInc :: Vector Int -> Vector Int
normalizeInc xs
  = let incs = map (+1) xs
        s    = fold (+) 0 xs
        ys   = map (/ s) incs
      in ys

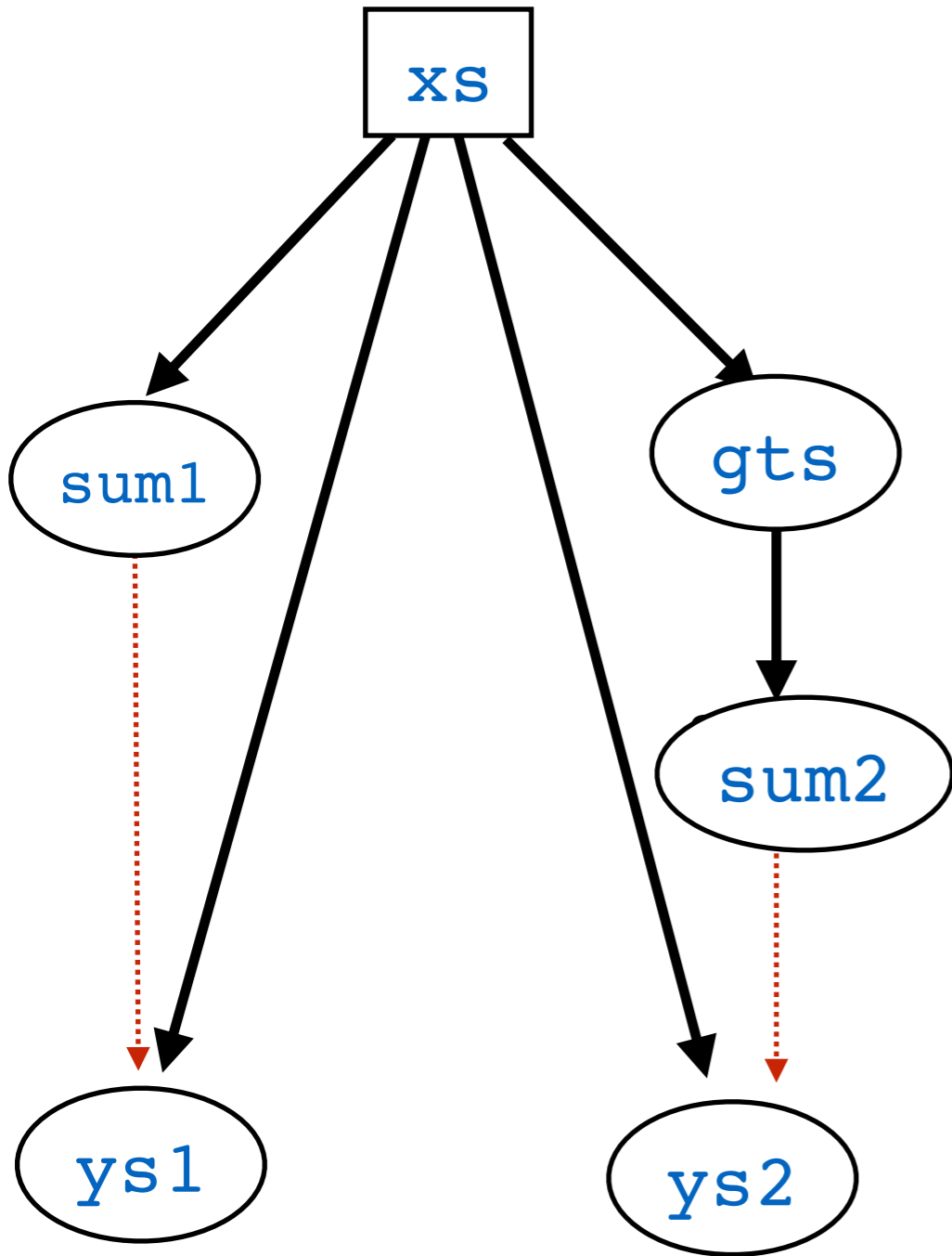
```

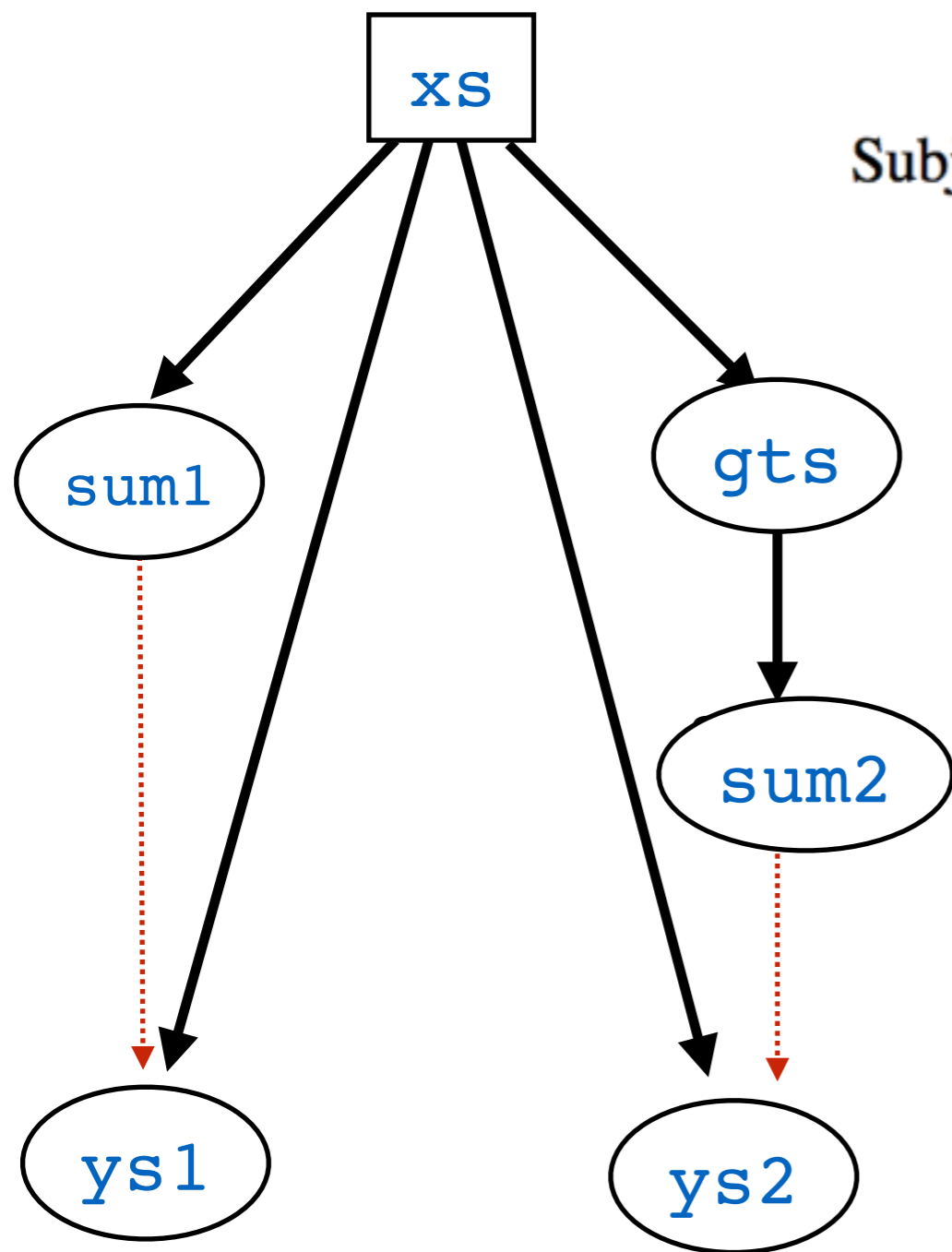


Integer Linear Programming (ILP) Formulation



Minimise $25 \cdot x_{sum1,gts} + 1 \cdot x_{sum1,sum2} + 25 \cdot x_{sum1,ys2} +$
 $25 \cdot x_{gts,sum2} + 25 \cdot x_{gts,ys1} + 1 \cdot x_{sum2,ys1} +$
 $25 \cdot x_{ys1,ys2} + 5 \cdot c_{gts} + 5 \cdot c_{ys1} + 5 \cdot c_{ys2}$

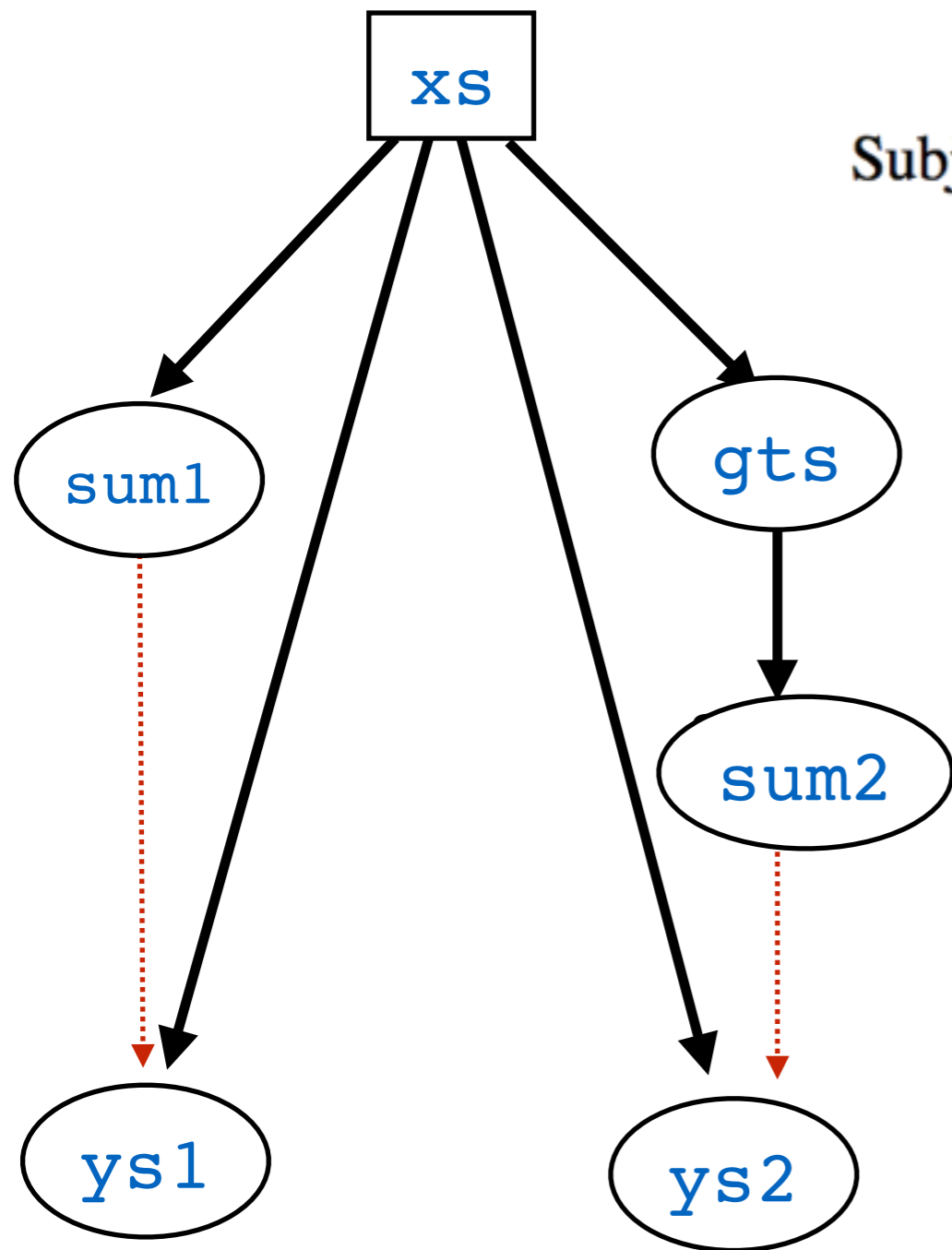




Minimise $25 \cdot x_{sum1,gts} + 1 \cdot x_{sum1,sum2} + 25 \cdot x_{sum1,ys2} +$
 $25 \cdot x_{gts,sum2} + 25 \cdot x_{gts,ys1} + 1 \cdot x_{sum2,ys1} +$
 $25 \cdot x_{ys1,ys2} + 5 \cdot c_{gts} + 5 \cdot c_{ys1} + 5 \cdot c_{ys2}$

Subject to

$-5 \cdot x_{sum1,gts}$	$\leq \pi_{gts} - \pi_{sum1}$	$\leq 5 \cdot x_{sum1,gts}$
$-5 \cdot x_{sum1,sum2}$	$\leq \pi_{sum2} - \pi_{sum1}$	$\leq 5 \cdot x_{sum1,sum2}$
$-5 \cdot x_{sum1,ys2}$	$\leq \pi_{ys2} - \pi_{sum1}$	$\leq 5 \cdot x_{sum1,ys2}$
$-5 \cdot x_{gts,ys1}$	$\leq \pi_{ys1} - \pi_{gts}$	$\leq 5 \cdot x_{gts,ys1}$
$-5 \cdot x_{sum2,ys1}$	$\leq \pi_{ys1} - \pi_{sum2}$	$\leq 5 \cdot x_{sum2,ys1}$
$-5 \cdot x_{ys1,ys2}$	$\leq \pi_{ys2} - \pi_{ys1}$	$\leq 5 \cdot x_{ys1,ys2}$



Minimise $25 \cdot x_{sum1,gts} + 1 \cdot x_{sum1,sum2} + 25 \cdot x_{sum1,ys2} +$
 $25 \cdot x_{gts,sum2} + 25 \cdot x_{gts,ys1} + 1 \cdot x_{sum2,ys1} +$
 $25 \cdot x_{ys1,ys2} + 5 \cdot c_{gts} + 5 \cdot c_{ys1} + 5 \cdot c_{ys2}$

Subject to

$$-5 \cdot x_{sum1,gts} \leq \pi_{gts} - \pi_{sum1} \leq 5 \cdot x_{sum1,gts}$$

$$-5 \cdot x_{sum1,sum2} \leq \pi_{sum2} - \pi_{sum1} \leq 5 \cdot x_{sum1,sum2}$$

$$-5 \cdot x_{sum1,ys2} \leq \pi_{ys2} - \pi_{sum1} \leq 5 \cdot x_{sum1,ys2}$$

$$-5 \cdot x_{gts,ys1} \leq \pi_{ys1} - \pi_{gts} \leq 5 \cdot x_{gts,ys1}$$

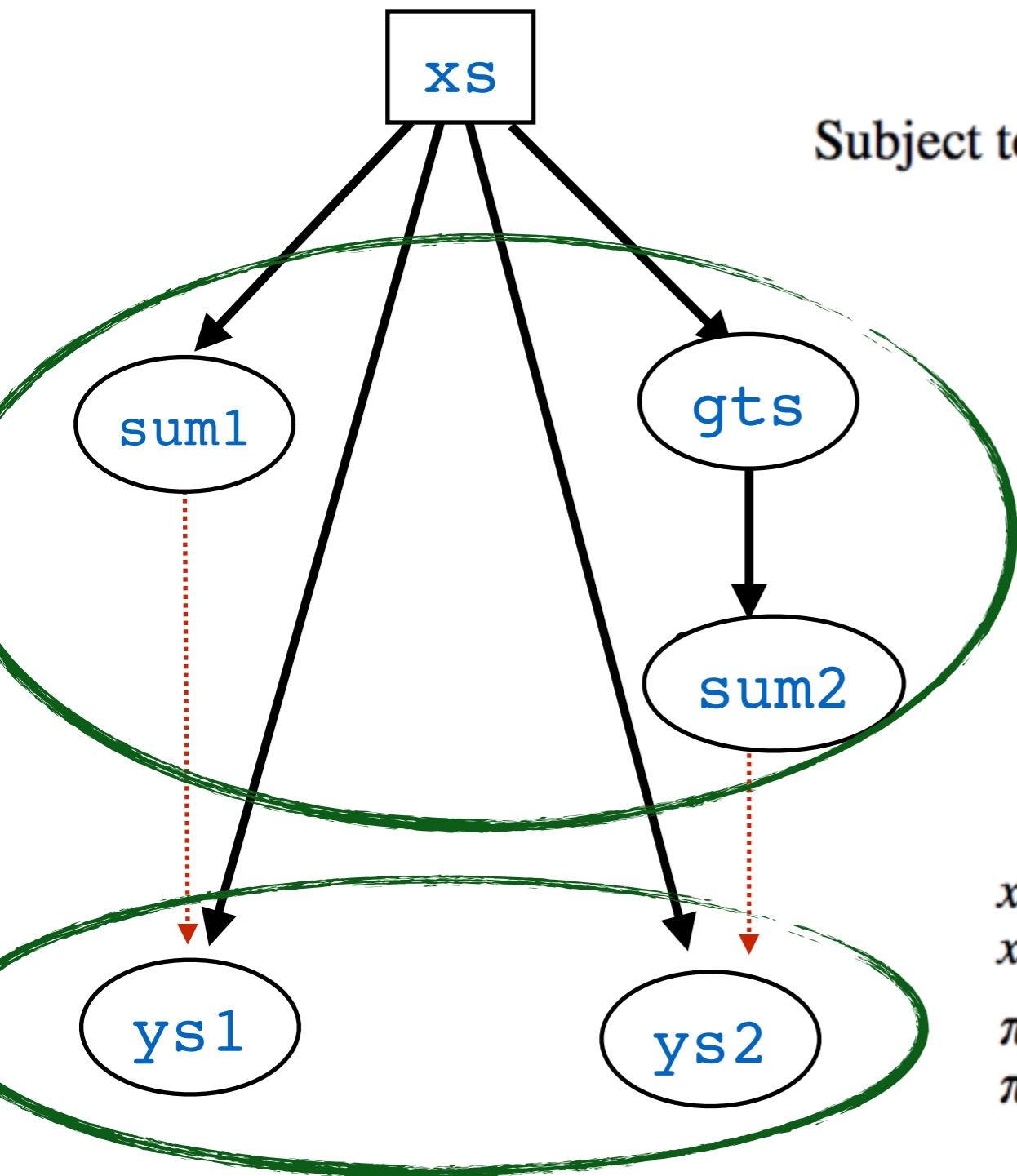
$$-5 \cdot x_{sum2,ys1} \leq \pi_{ys1} - \pi_{sum2} \leq 5 \cdot x_{sum2,ys1}$$

$$-5 \cdot x_{ys1,ys2} \leq \pi_{ys2} - \pi_{ys1} \leq 5 \cdot x_{ys1,ys2}$$

$$x_{gts,sum2} \leq \pi_{sum2} - \pi_{gts} \leq 5 \cdot x_{gts,sum2}$$

$$\pi_{sum1} < \pi_{ys1}$$

$$\pi_{sum2} < \pi_{ys2}$$



Minimise $25 \cdot x_{sum1,gts} + 1 \cdot x_{sum1,sum2} + 25 \cdot x_{sum1,ys2} +$
 $25 \cdot x_{gts,sum2} + 25 \cdot x_{gts,ys1} + 1 \cdot x_{sum2,ys1} +$
 $25 \cdot x_{ys1,ys2} + 5 \cdot c_{gts} + 5 \cdot c_{ys1} + 5 \cdot c_{ys2}$

Subject to

$$\begin{aligned} -5 \cdot x_{sum1,gts} &\leq \pi_{gts} - \pi_{sum1} &\leq 5 \cdot x_{sum1,gts} \\ -5 \cdot x_{sum1,sum2} &\leq \pi_{sum2} - \pi_{sum1} &\leq 5 \cdot x_{sum1,sum2} \\ -5 \cdot x_{sum1,ys2} &\leq \pi_{ys2} - \pi_{sum1} &\leq 5 \cdot x_{sum1,ys2} \\ -5 \cdot x_{gts,ys1} &\leq \pi_{ys1} - \pi_{gts} &\leq 5 \cdot x_{gts,ys1} \\ -5 \cdot x_{sum2,ys1} &\leq \pi_{ys1} - \pi_{sum2} &\leq 5 \cdot x_{sum2,ys1} \\ -5 \cdot x_{ys1,ys2} &\leq \pi_{ys2} - \pi_{ys1} &\leq 5 \cdot x_{ys1,ys2} \\ x_{gts,sum2} &\leq \pi_{sum2} - \pi_{gts} &\leq 5 \cdot x_{gts,sum2} \\ \pi_{sum1} &< \pi_{ys1} \\ \pi_{sum2} &< \pi_{ys2} \end{aligned}$$

$$\begin{aligned} x_{sum1,gts}, x_{sum1,sum1}, x_{sum1,sum2}, x_{gts,sum2}, x_{ys1,ys2} &= 0 \\ x_{sum1,ys2}, x_{gts,ys1}, x_{sum2,ys1} &= 1 \\ \pi_{sum1}, \pi_{gts}, \pi_{sum2} &= 0 \\ \pi_{ys1}, \pi_{ys2} &= 1 \\ &= 0 \end{aligned}$$

Future Work / Challenges

Key Challenges

- Proliferation of distinct systems that want to own the data.
- No overarching framework for comparing the systems.
 - Relational algebra is logical-plan only, need to compare physical plans as well.
- Little separation of foundational issues vs implementation.
 - *Don't know how to do it vs too expensive with TeraData.*
- Lack of space complexity guarantees for intermediate results.
- Query plans transformed with non-confluent rewrite systems.
- Performance correctness?
- The “cult of optimisation”