Rufous

Automated Data Structure Benchmarking in Haskell

Ben Simner
University of Camsridge

+

Colin Runciman
University of York

Work In Progress

Automated Benchmarking of Haskell Data Structure

Ben Simner + Colin Runciman

University of Cambridge + University of York

Persistent Data Structures

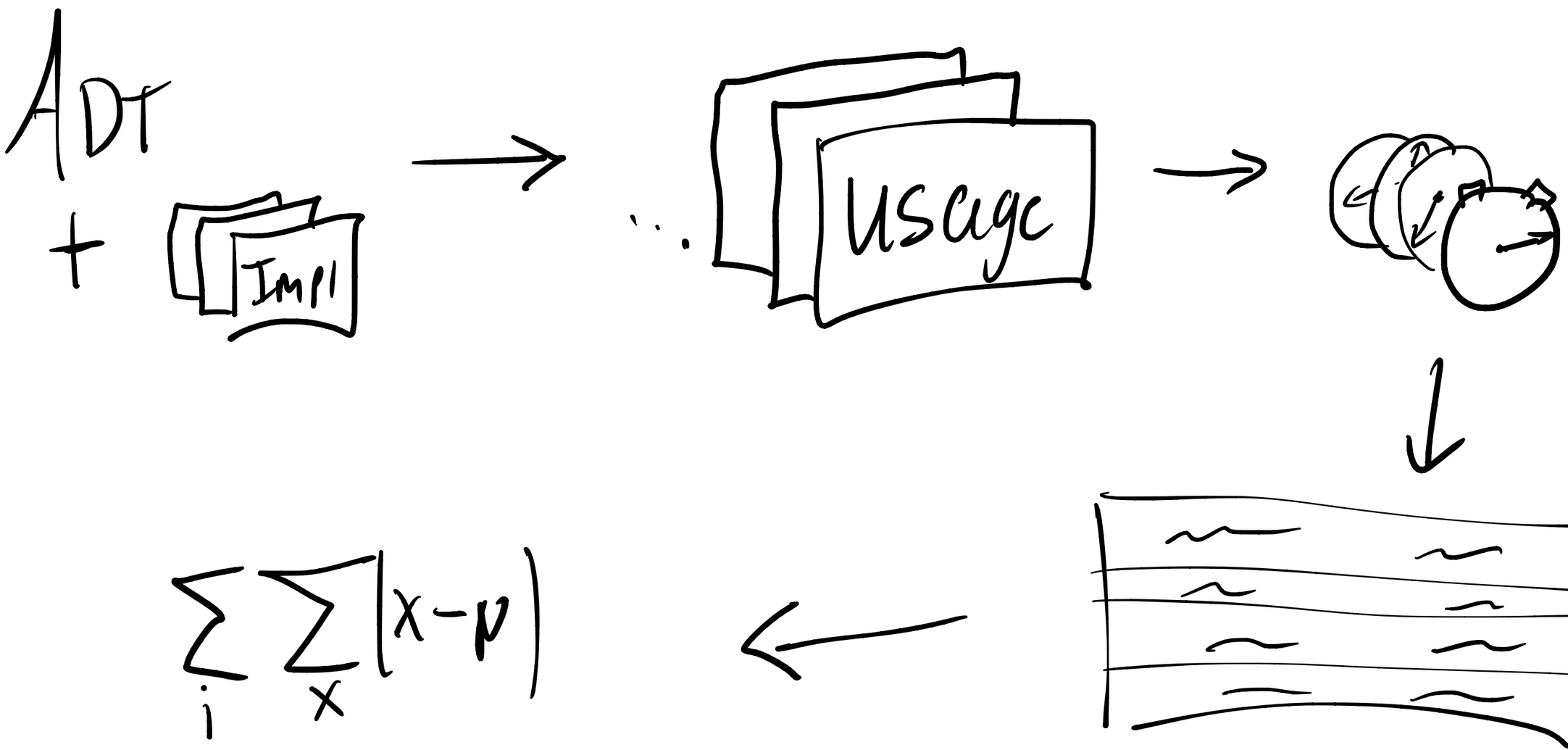
```
main = do
    let xs = [1]
    let ys = xs ++ [2]
    print ys
```

Persistent Data Structures

```
main = do
    let xs = [1]
    let ys = xs ++ [2]
    Print ys

    let zs = xs ++ [3]
    Print zs
```

Rufous



Haskell

```
f x y = y

main = print (f undefined 42)
```

Queues (In Haskell)

```
type Queue a = [a]

empty = []
snoc q v = q ++ [v]
tail (_:qr) = qr
head (v:_) = v
```

Queues (In Haskell)

```
type Queue a = [a]

empty = []
snoc q v = q ++ [v]
tail (_:qr) = qr
head (v:_) = v
```

ty var

CAF

gen/mut/obs

partial

lazy

Every Data type Ever

$$
\begin{aligned}
ADT \equiv (\; & name : String \\
, \; & tyvar : a \\
, \; & generators \\
, \; & mutators \\
, \; & observors \\
, \; & \\
) \; & \\
\end{aligned}
\Bigg\} : 2^{Fn}
$$

Queues (But Better!)

Queues (But Better)

```
type Queue a = ([a], [a])

empty = ([], [])

snoc x (f, b) = (f, x : b)

head (h : f, b) = h

tail (h : [], b) = (rev b, [])
   | (h : f, b)  : (f, b)
```

Queues (even more better)

```
type Queue a = ([a], [a], [a])
```

Version Graphs

$V_1 = []$

$V_2 = 1 : V_1$

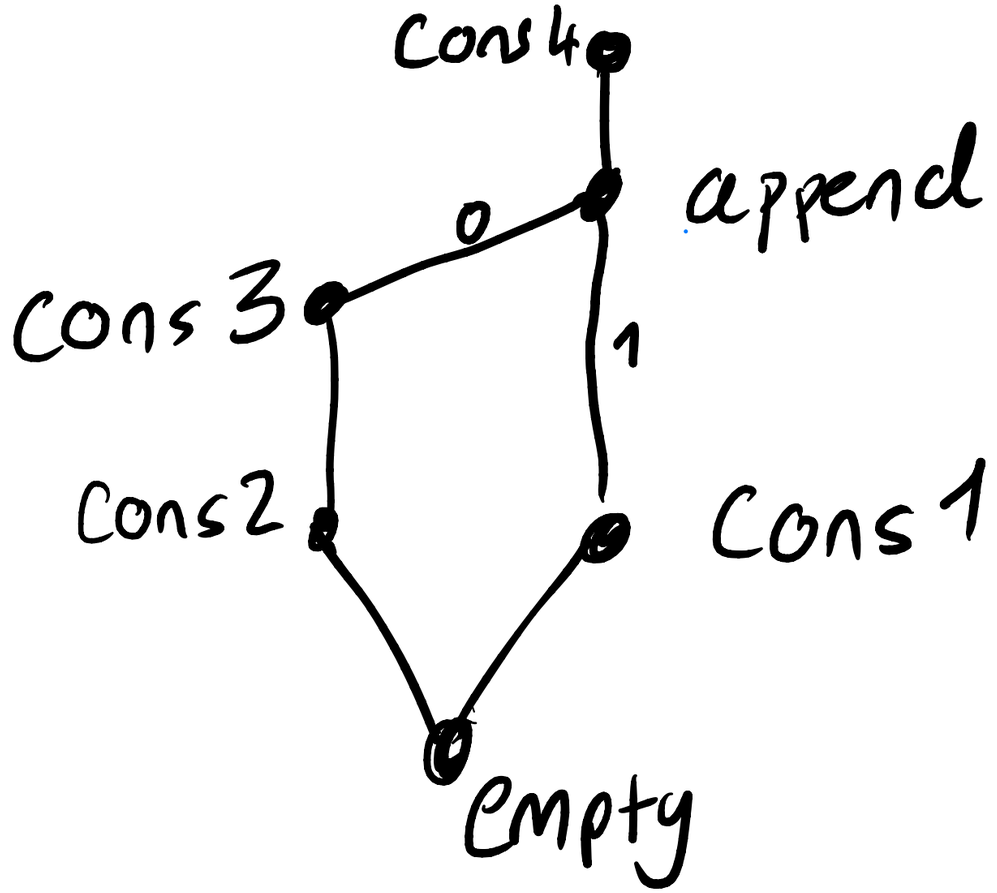
$V_3 = 2 : V_1$

$V_4 = 3 : V_3$

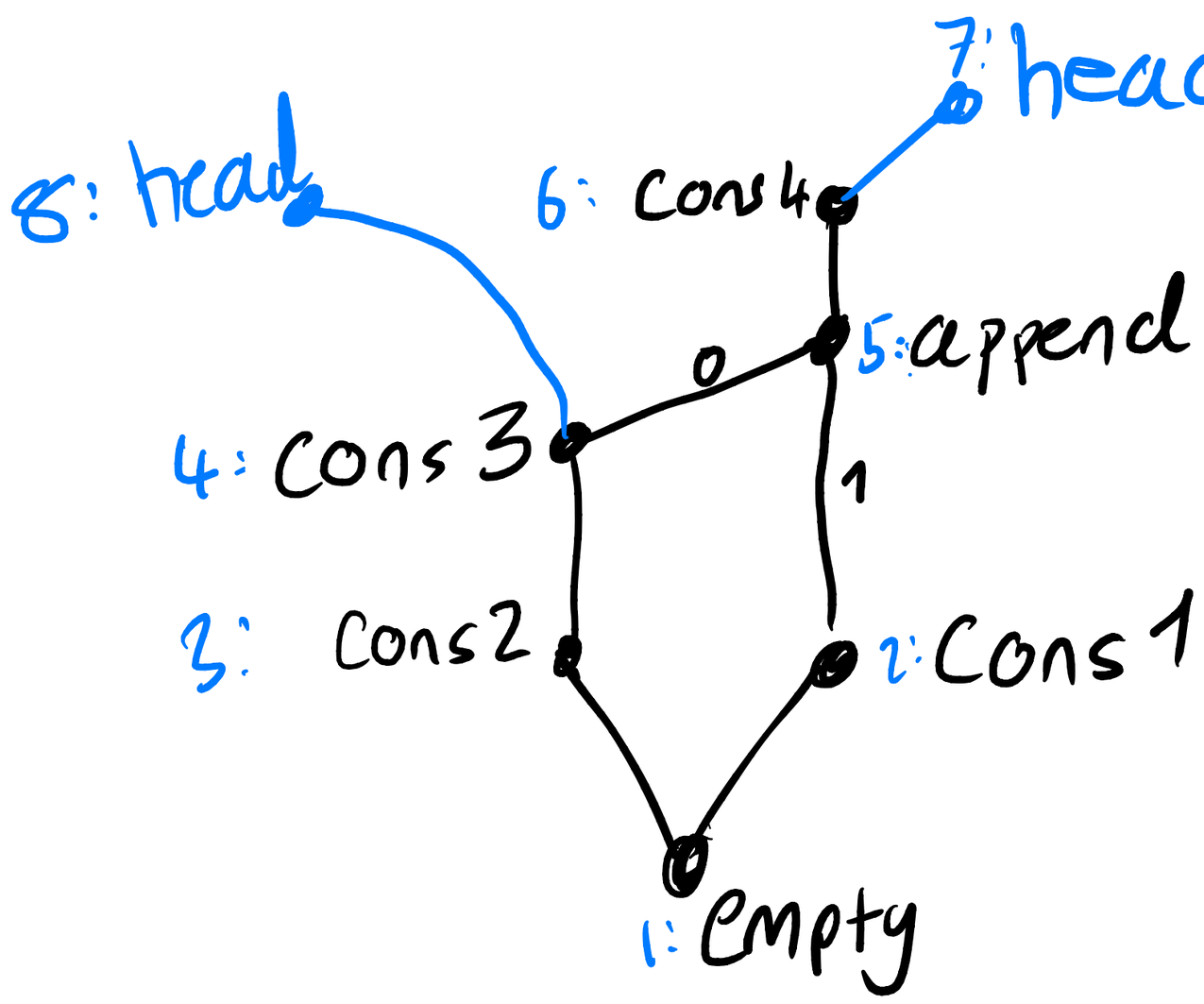
$V_5 = V_4 ++ V_2$

$V_6 = 4 : V_5$

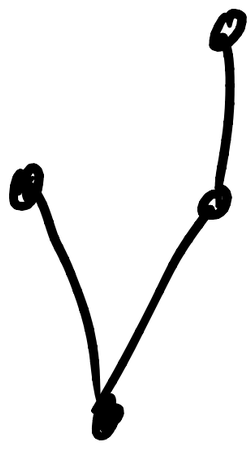
```
main = do
  print (head v6)
  Print (head v4)
```

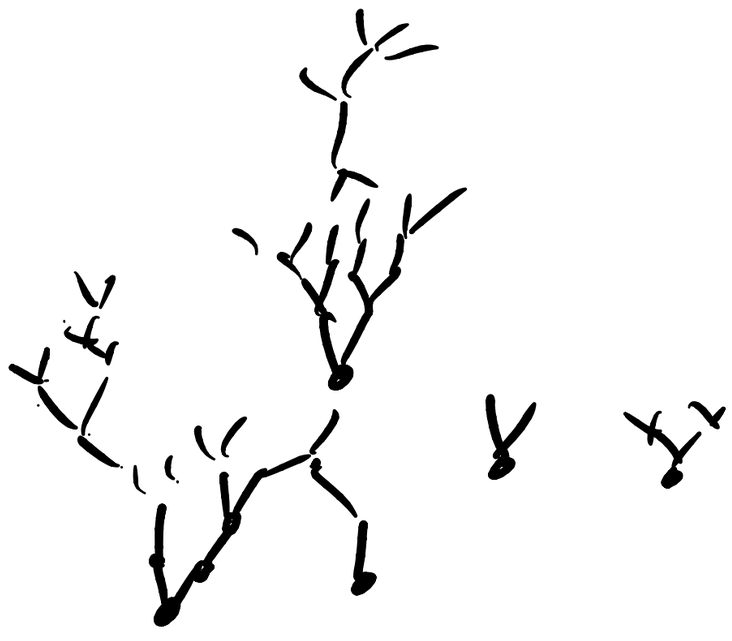


Datatype Usage Graphs

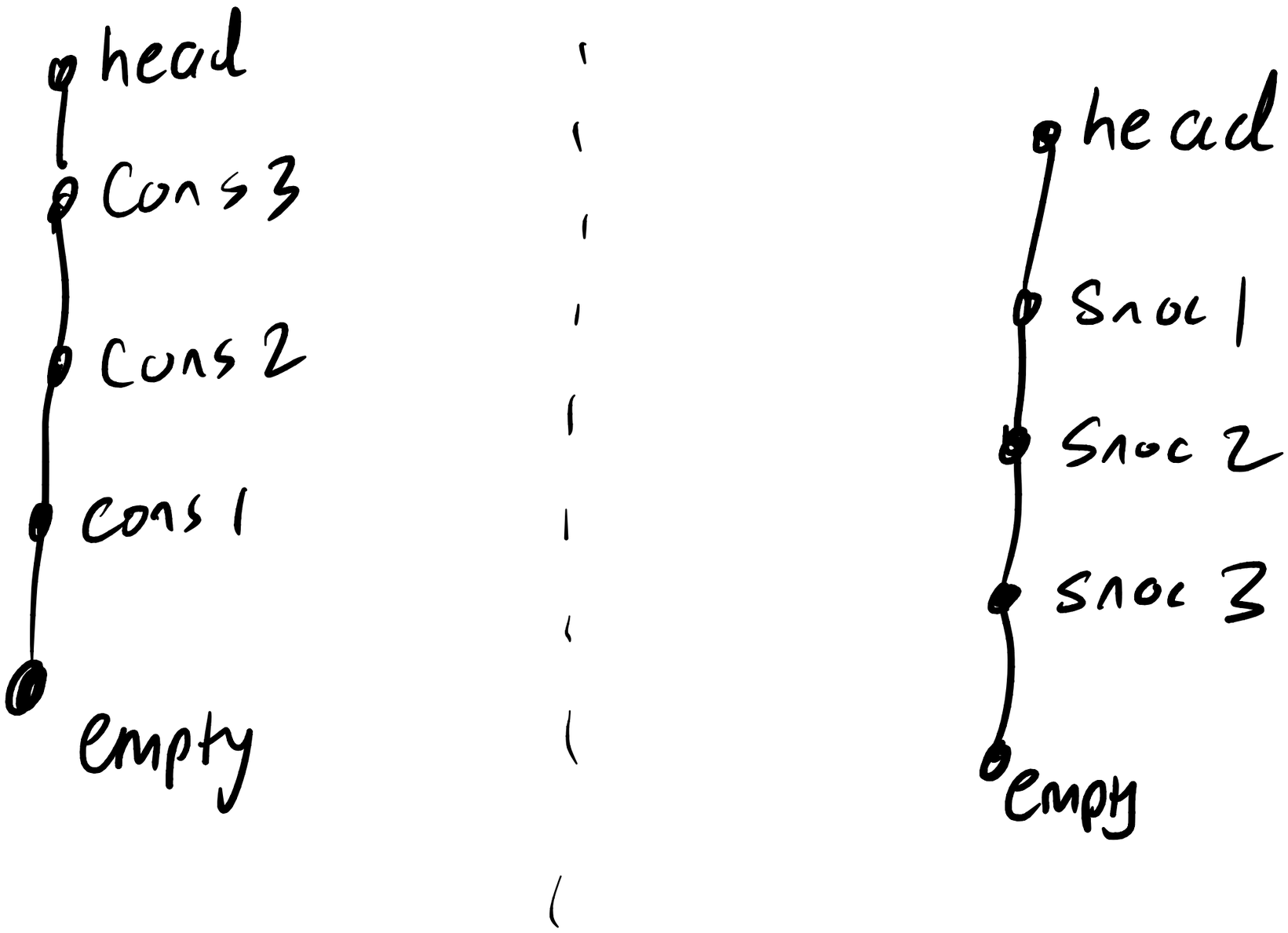


Categorizing Usages: Size

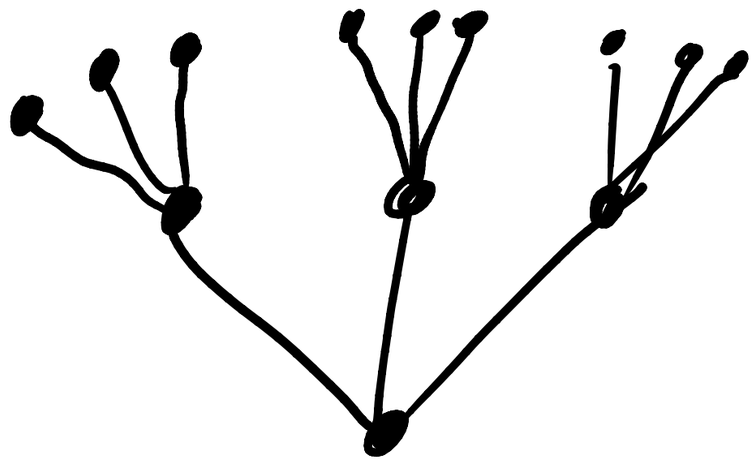




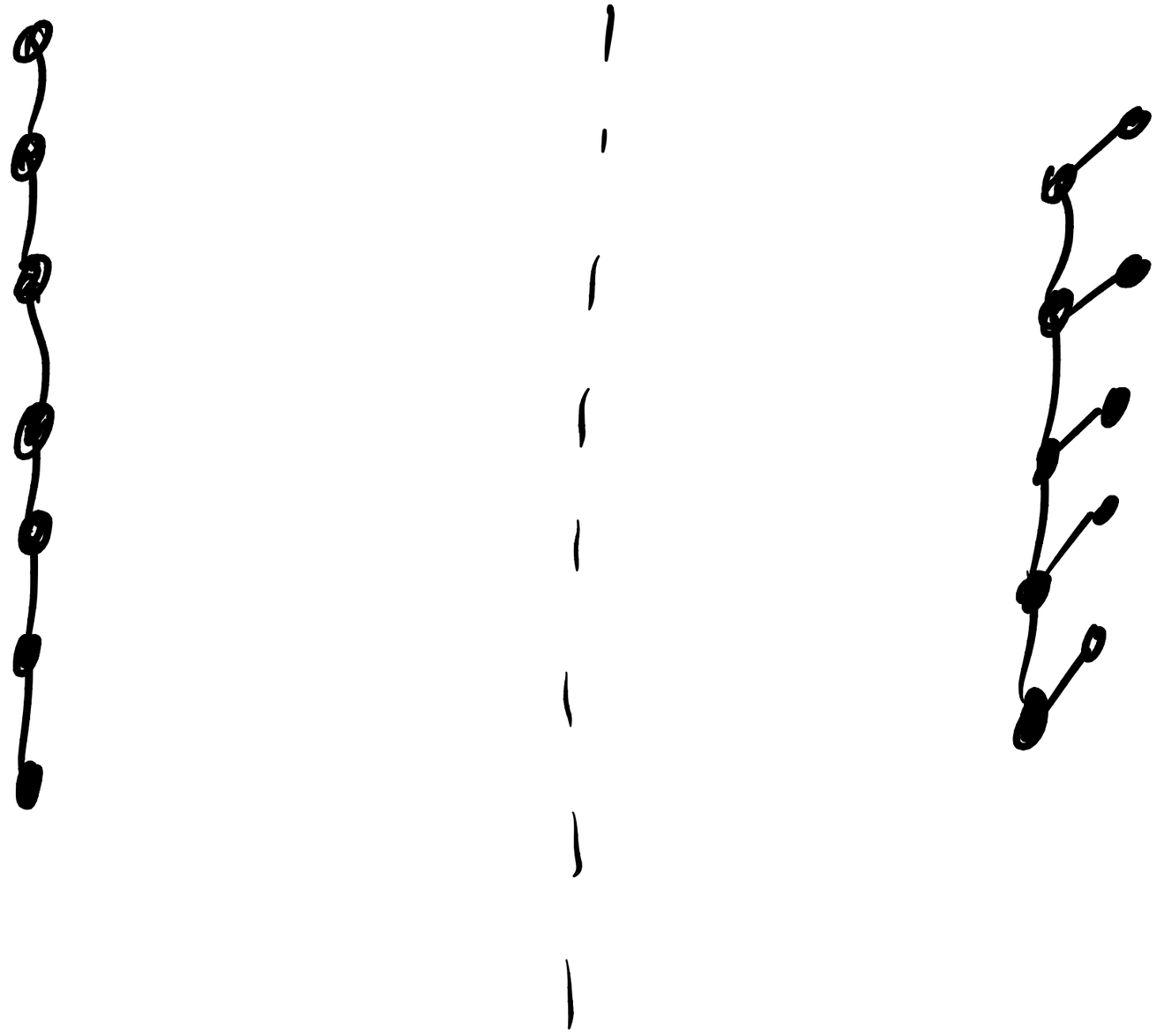
Categorizing Usage: Weights



Categorizing Usage: Persistence



Categorizing Usage: Mortality



Usage Profile

$$\text{Profile} \stackrel{\text{def}}{=} w : Op \to Prob$$

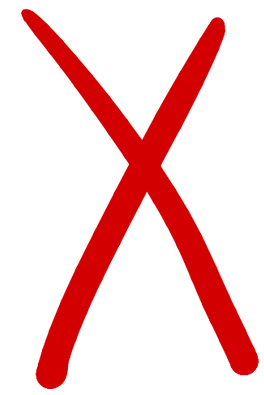
$$\times\ mort : Prob$$

$$\times\ pmf, pof : Prob$$

$$\times\ size : \mathbb{Z}$$

Guarding Bad Usages

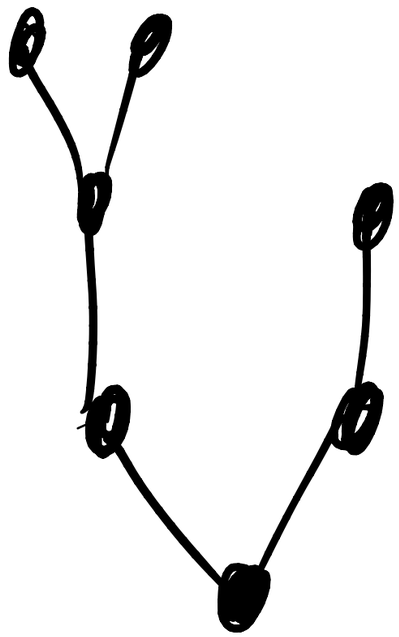
Shadow Implementations



```
type Shadow = Int

empty = 0
cons_s = 1+s
tail 0 = bad
tail n = n-1
head 0 = bad
head n = undef
```

Evaluation



3: head = dyn (head (undyn v_2))

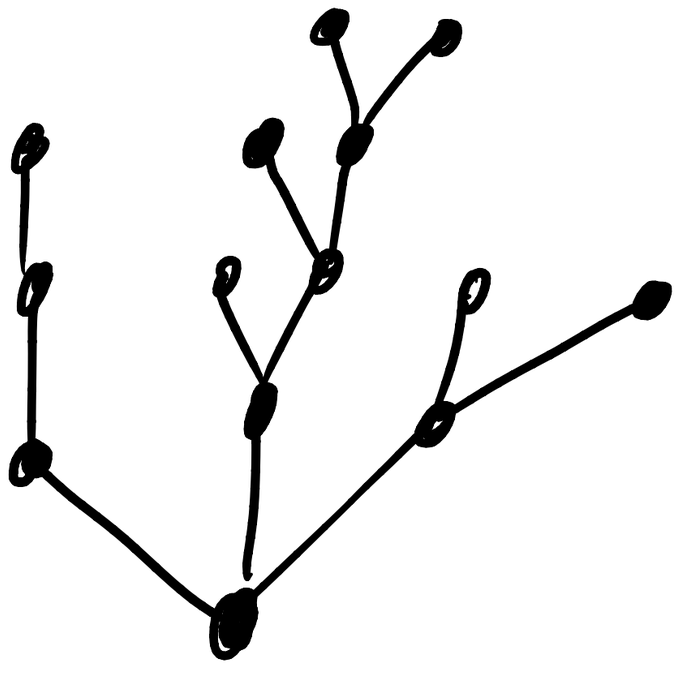
2: cons 1 = dyn (1 : undyn v_1)

1: empty = dyn []

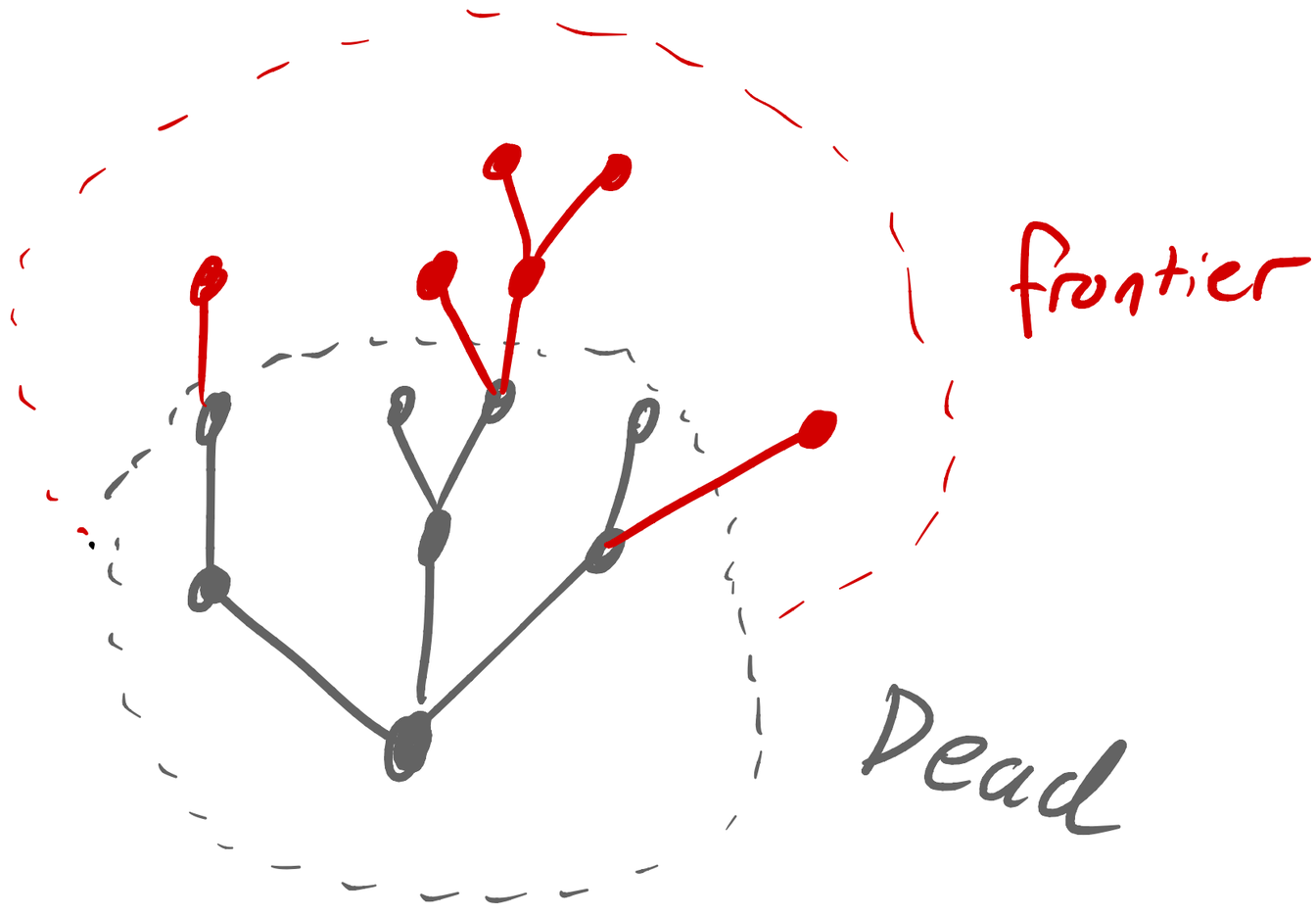
Generating Usages

```
Gen (profile) >>= GenDUG
```

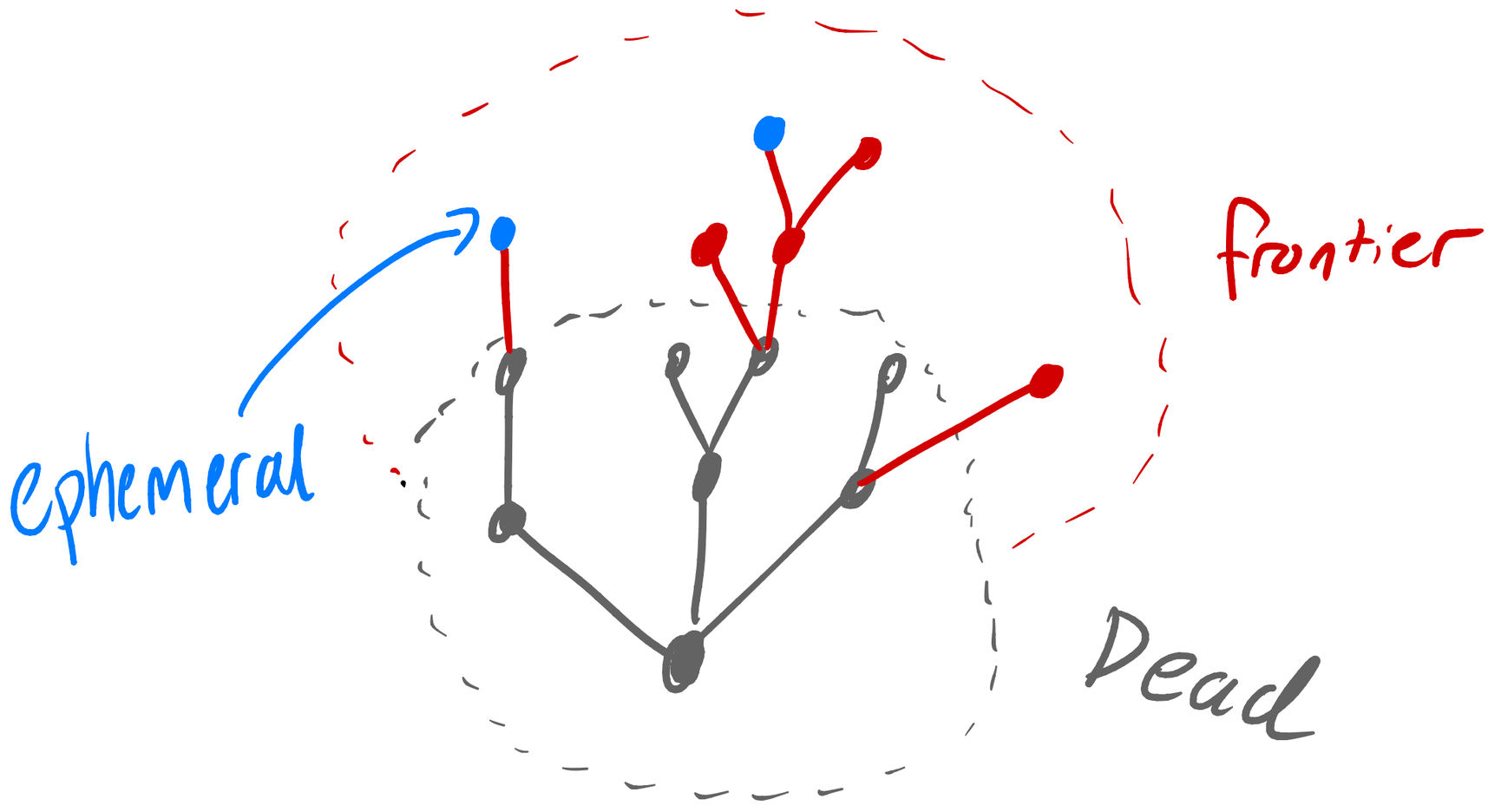
Generating Usages



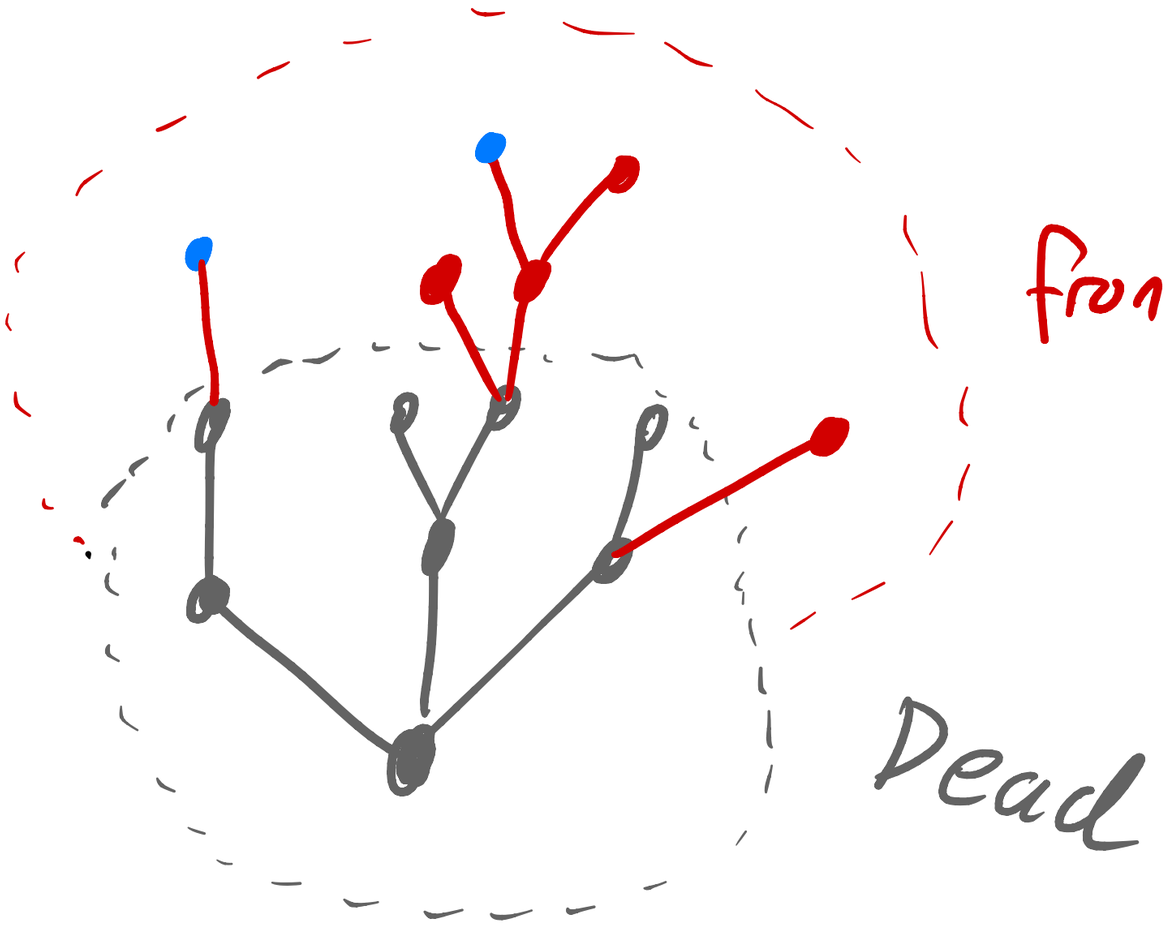
Generating Usages



Generating Usages



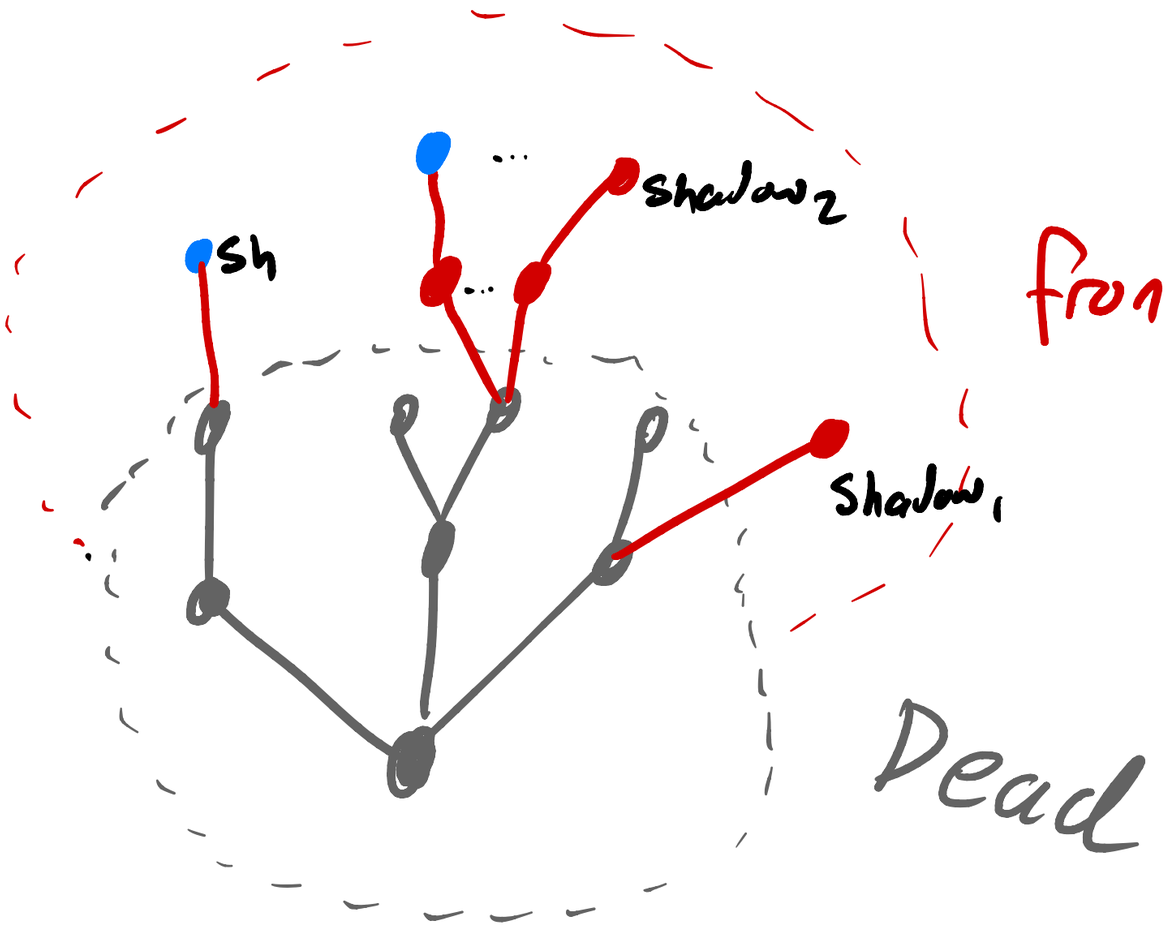
Generating Usages



Insert:

Snoc _ _

Generating Usages

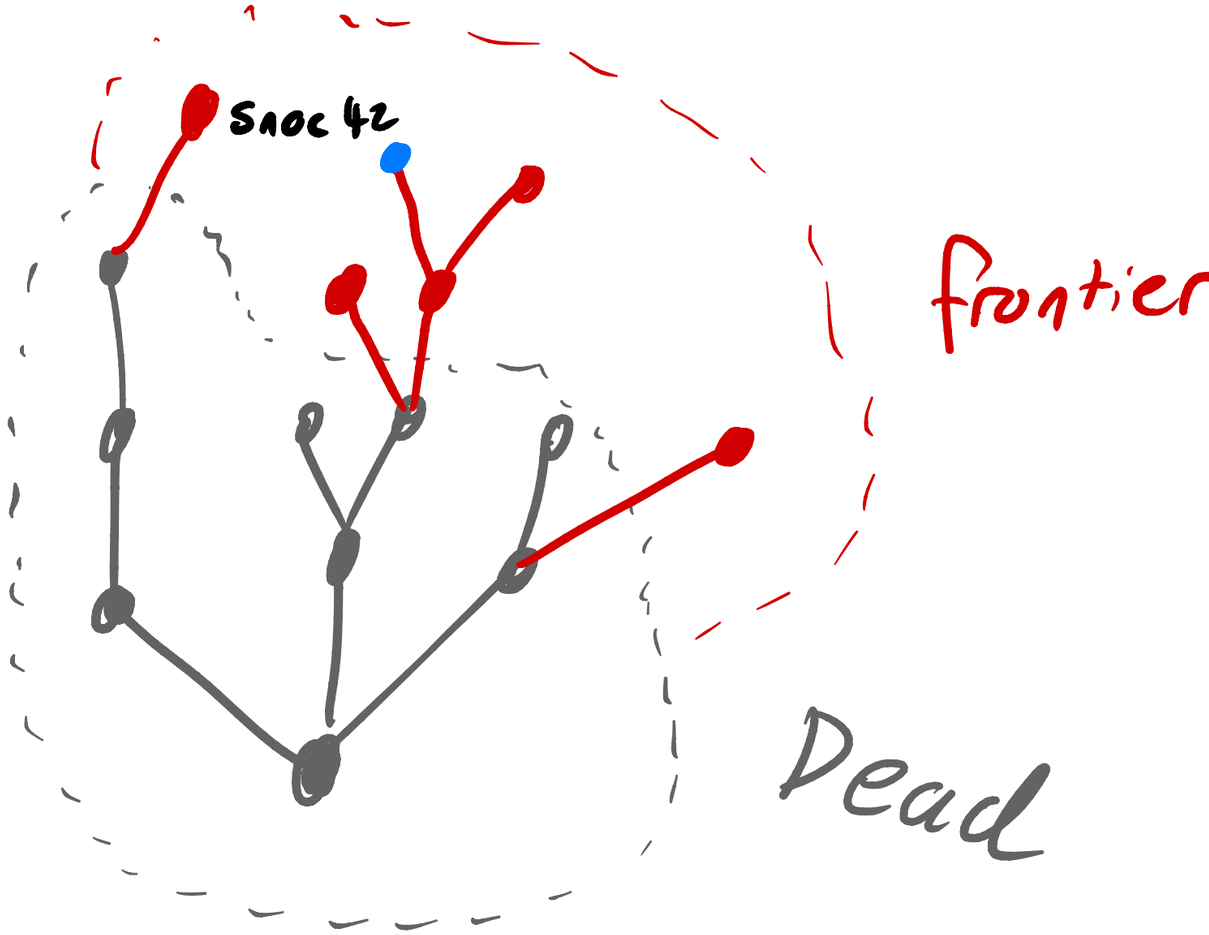


Insert:

Snoc 42 ●?

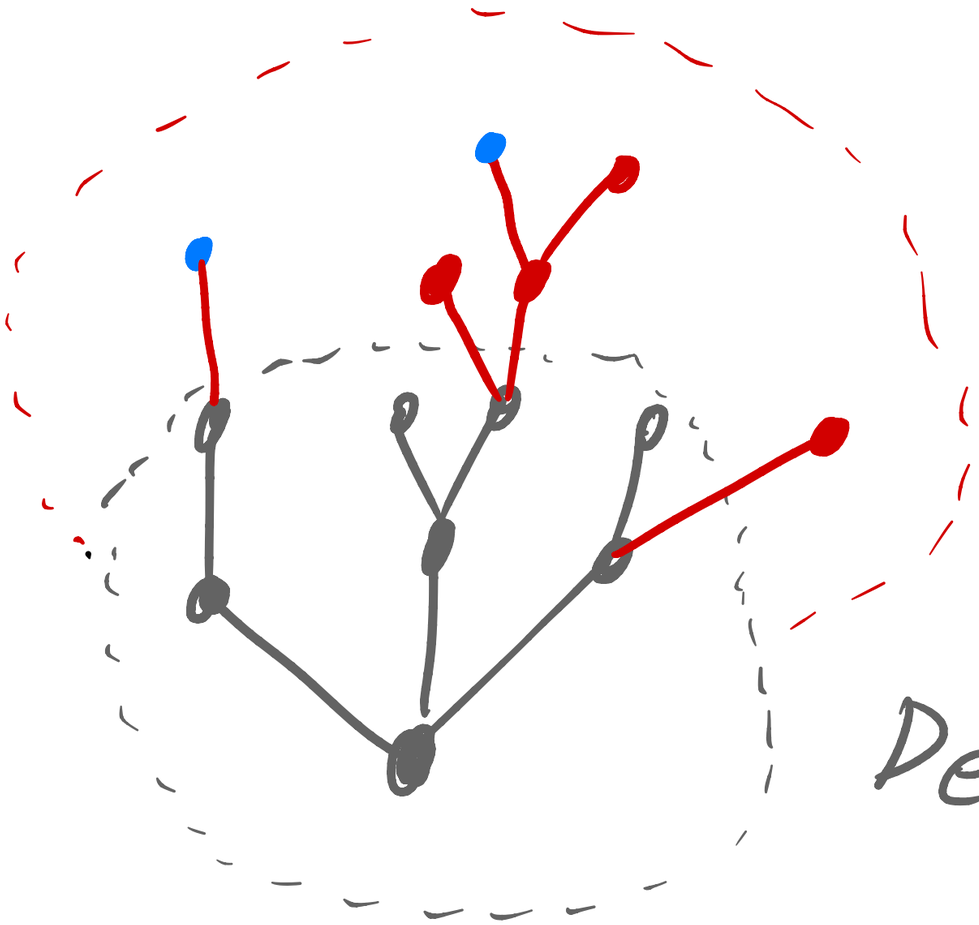
Shadow. snoc 42 Sh?

Generating Usages



Success

Generating Usages



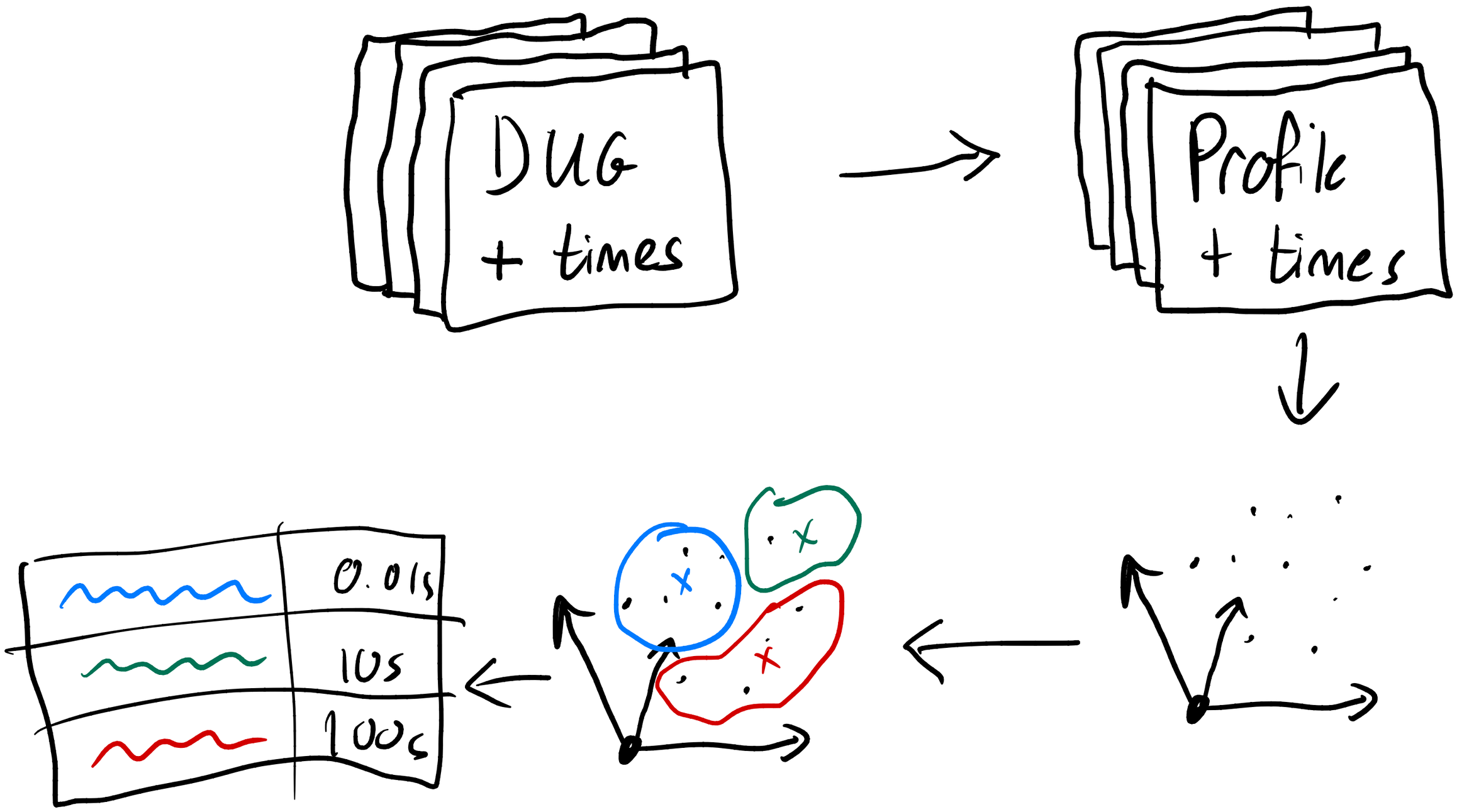
Fail

buffer :=

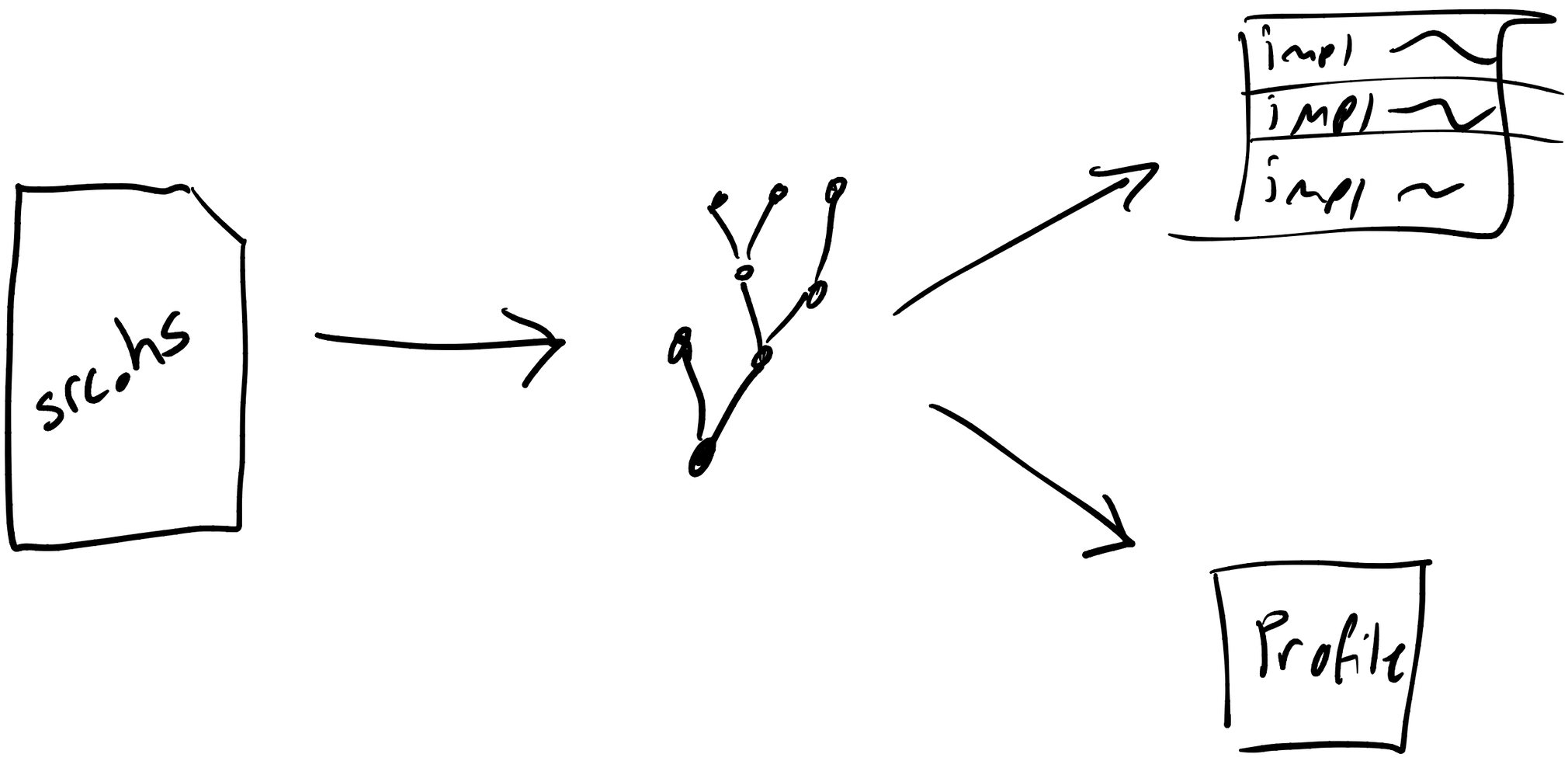
Snoc 42 •?

```
genProfile =
    fix $ λd.
      if (sized == sizeProfile)
      then d
      else drain (inflateProfile d)
```

Aggregation



Extraction



Extraction

```
type Queue' a = Wrapper (Queue a) a

empty' = let vid = VID() in wrap vid []

snoc' x xs = let vid = VID() in
  wrap vid (snoc x (unwrap vid xs))

  .
  .
  .
```

Demo

Recap, Caveats, and Conclusions

+ DUGs define usage
+ Profiles: metric over usage
+ Rufous Generates DUGs
+ Time evaluation
+ Cluster similar profiles & generate reports

- Limited ADT Signature (No conc., No higher order)
- Only time, not space.
- Profile space unmotivated.
- Real examples lacking
- Lack of community ADT interfaces.