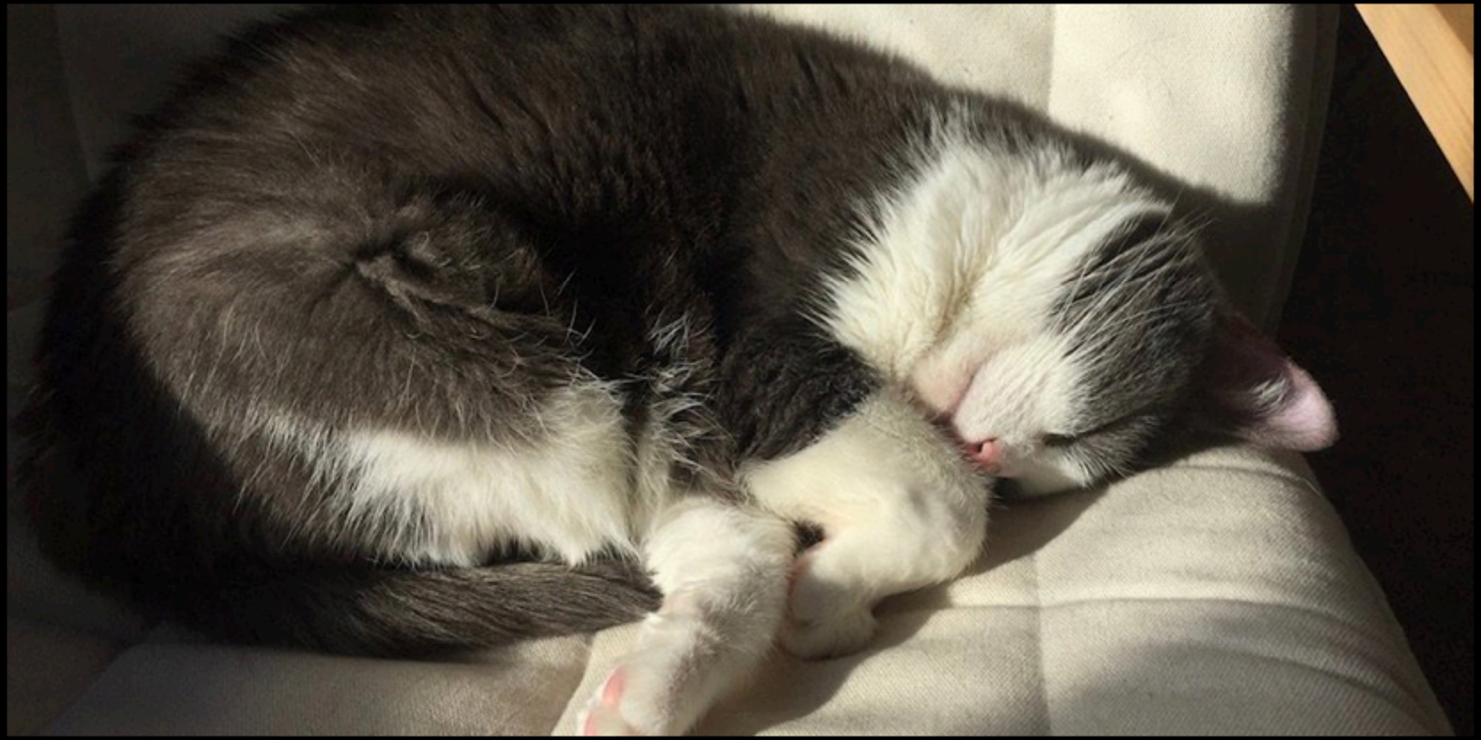
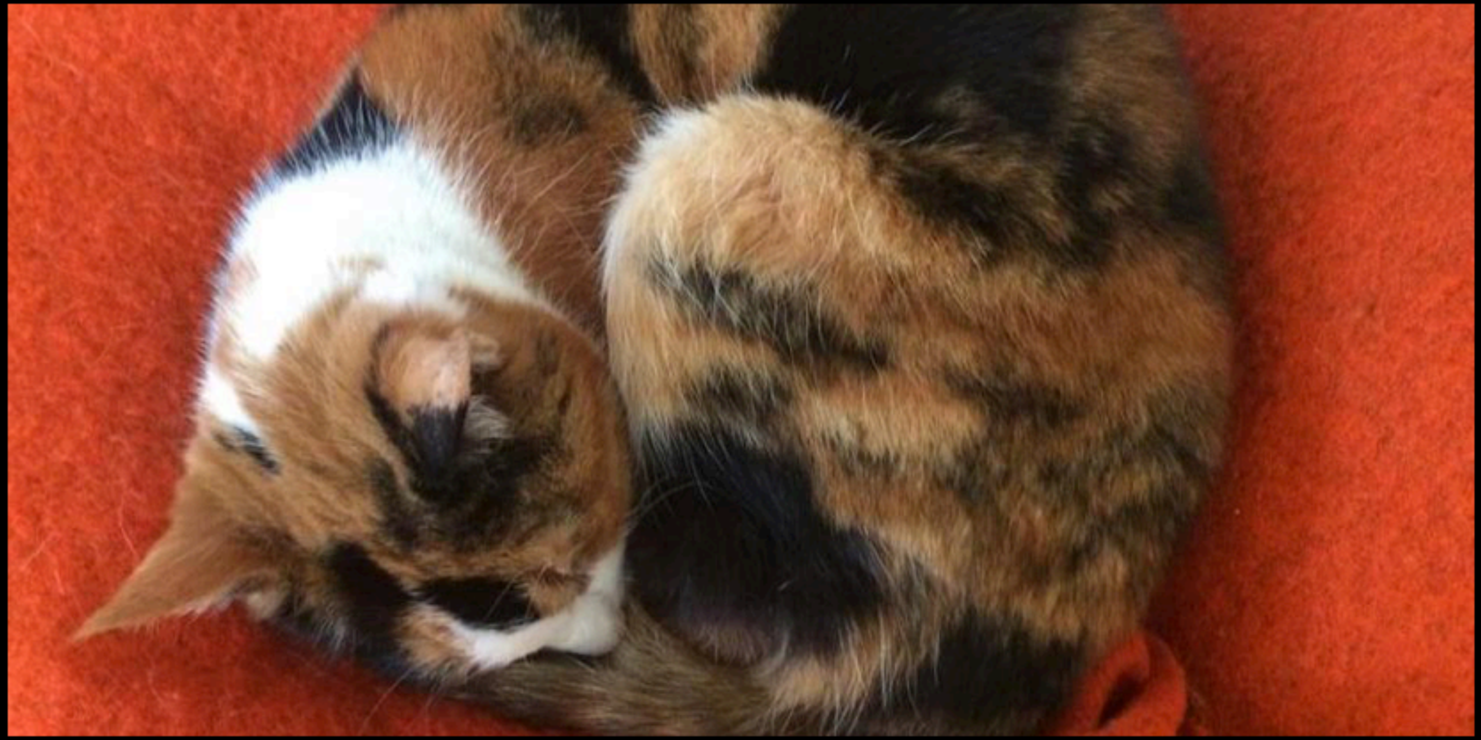
Robustness as a Refinement Type

Wen Kokke, Ekaterina Komendantskaya, and Daniel Kienitz

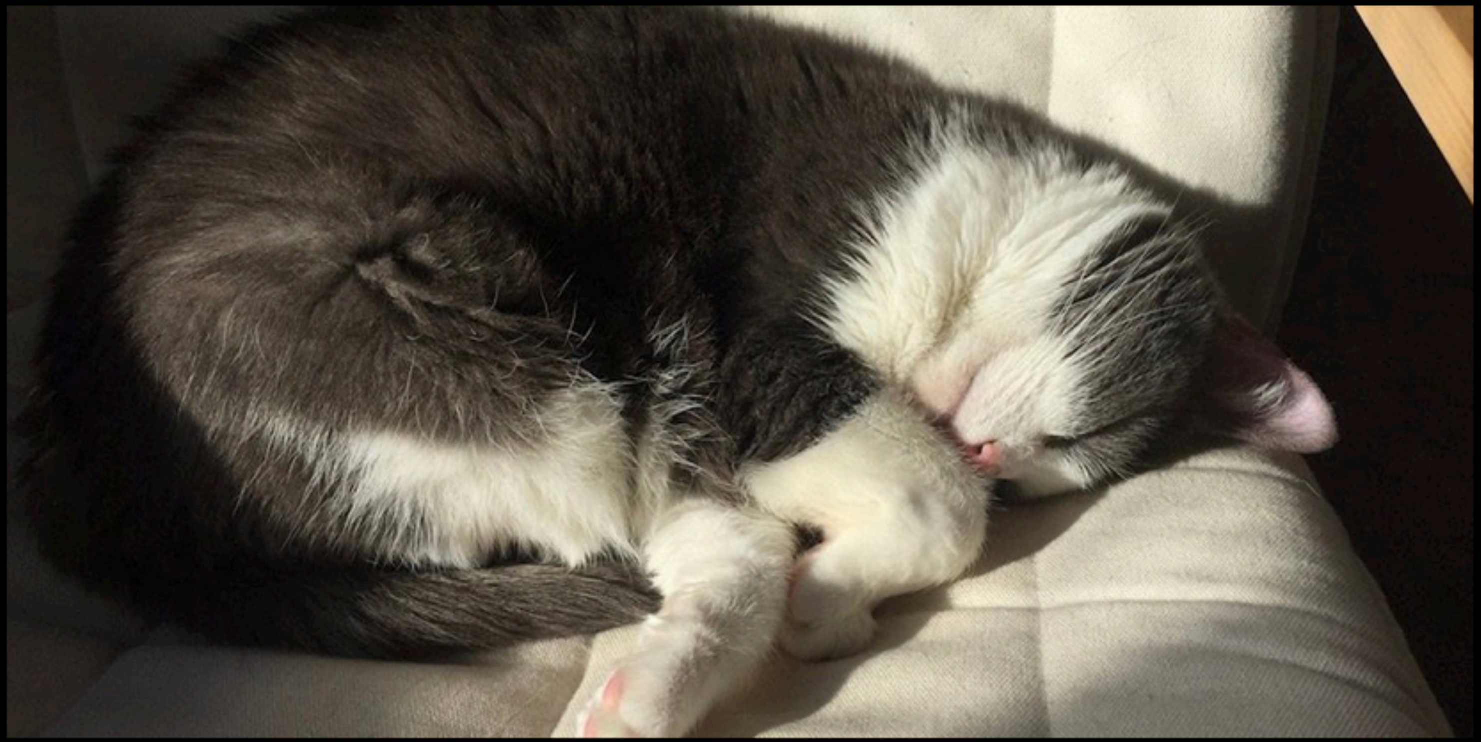
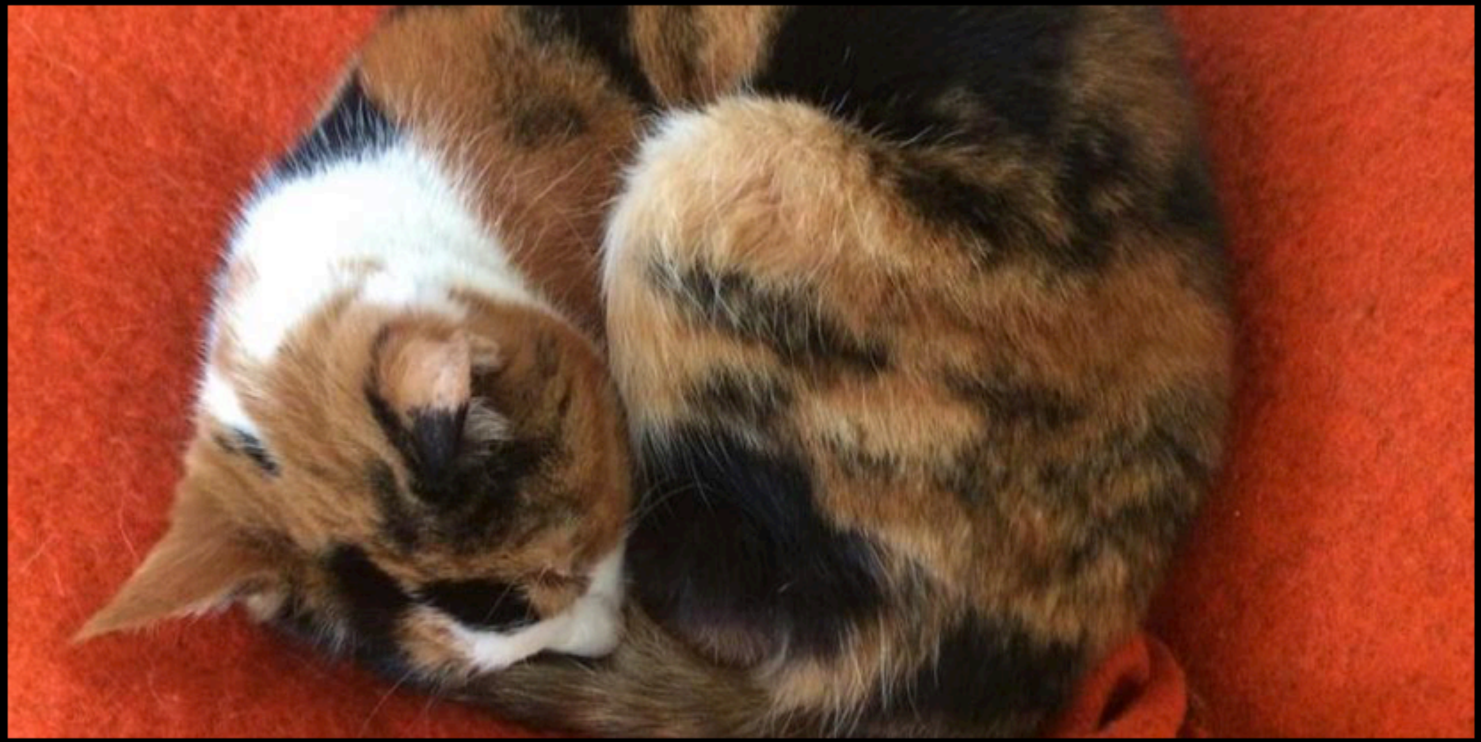
Lab for AI and Verification, Heriot-Watt University



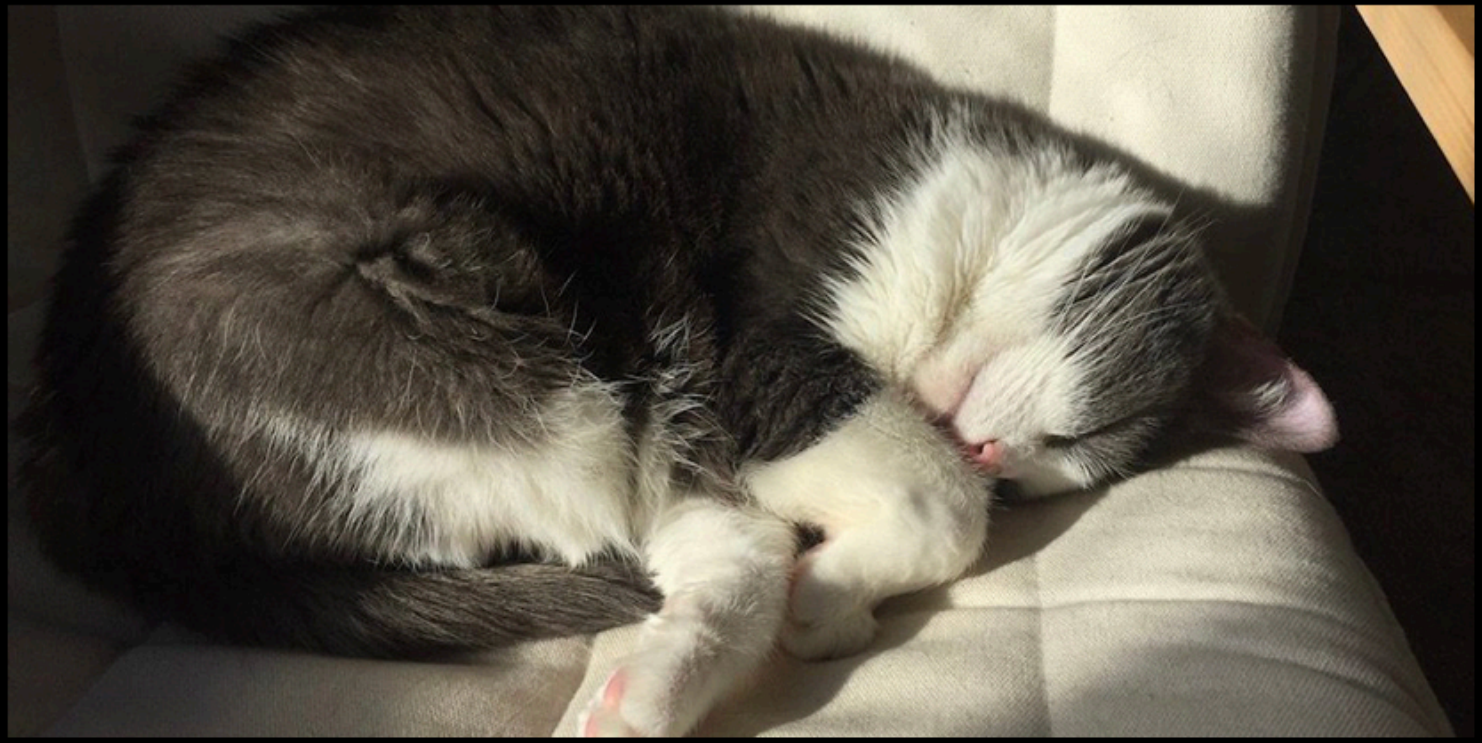
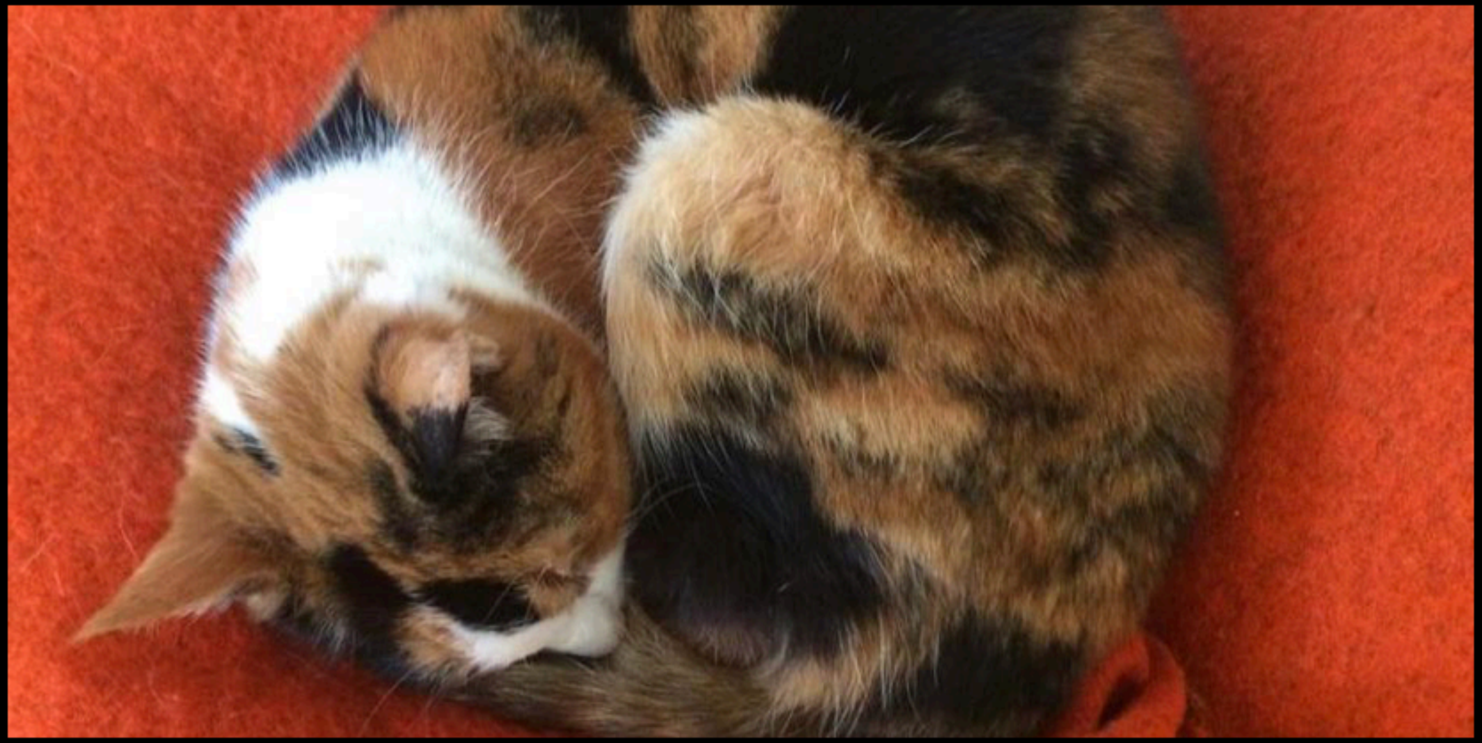




dog?

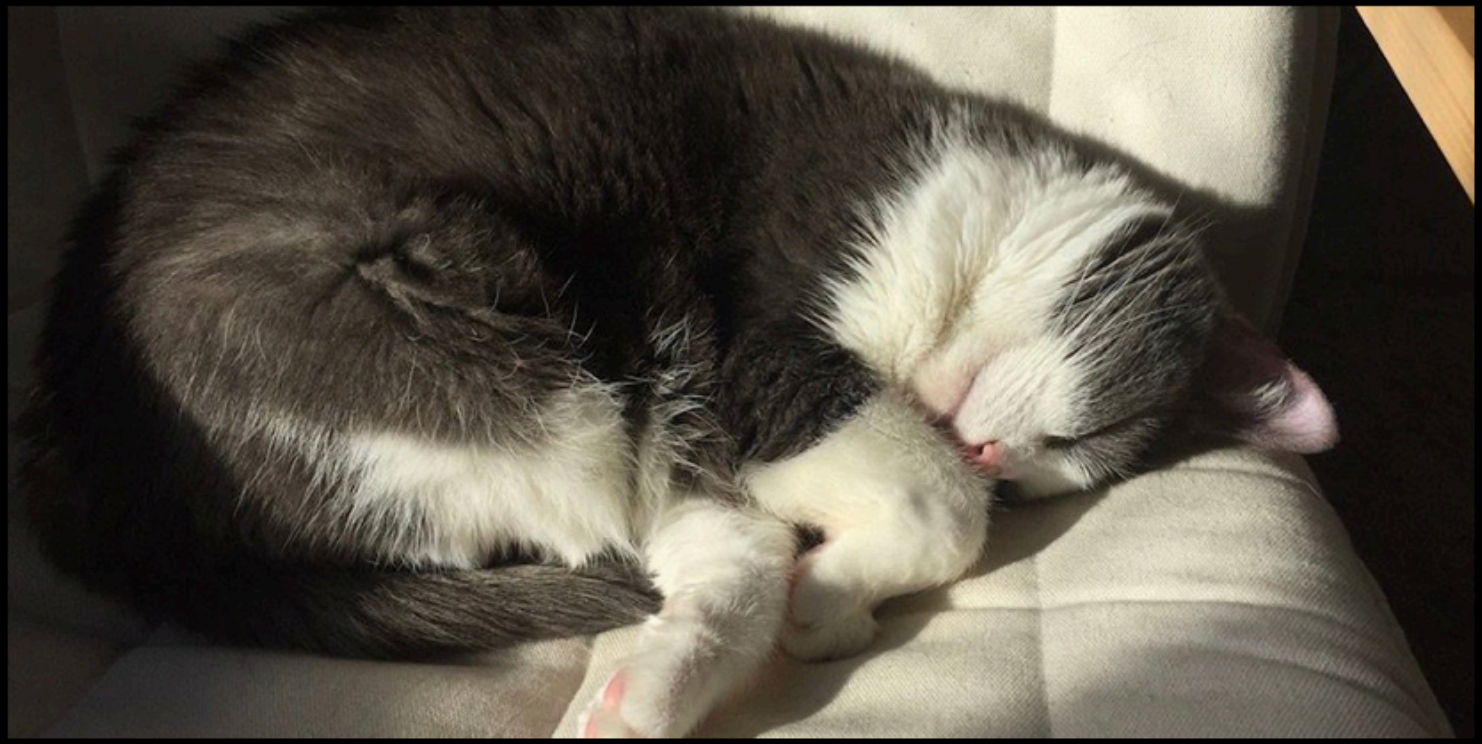
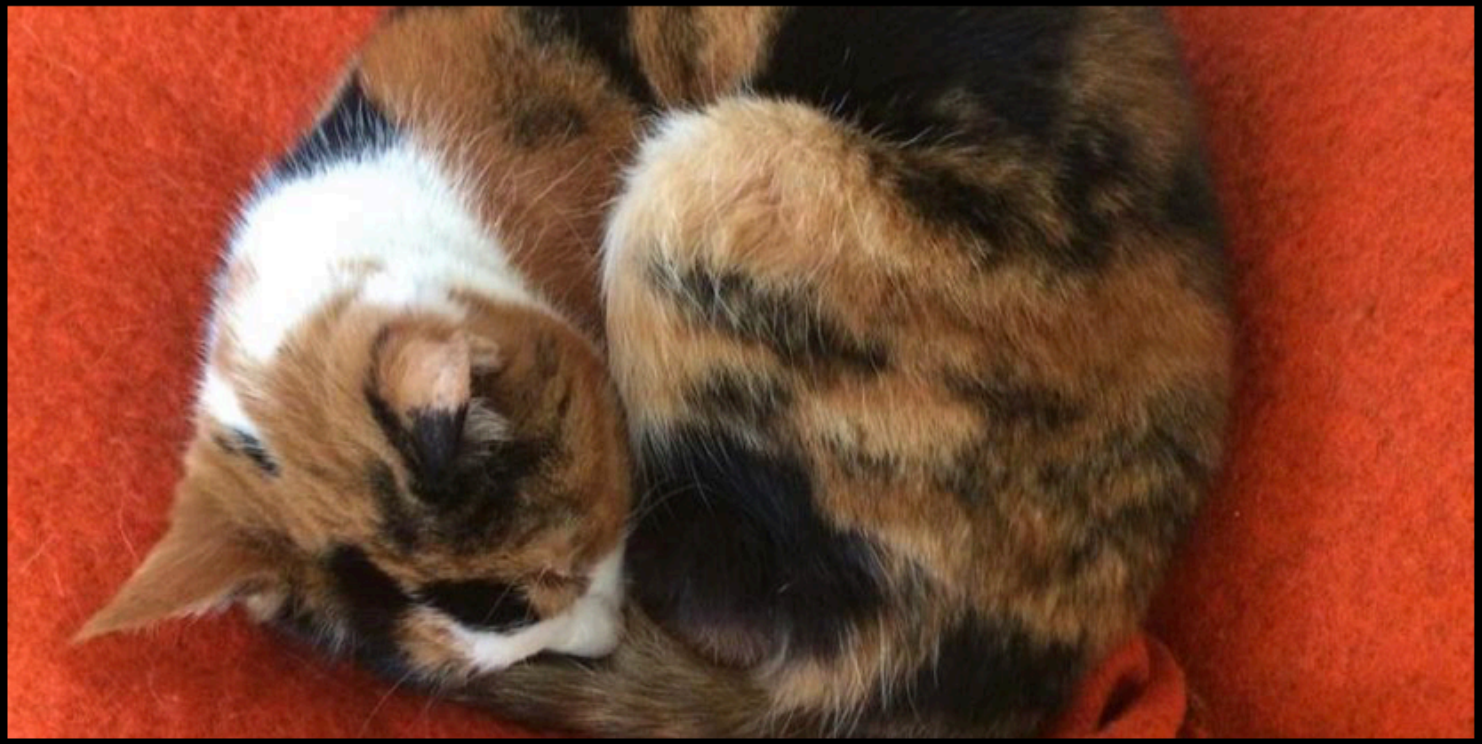


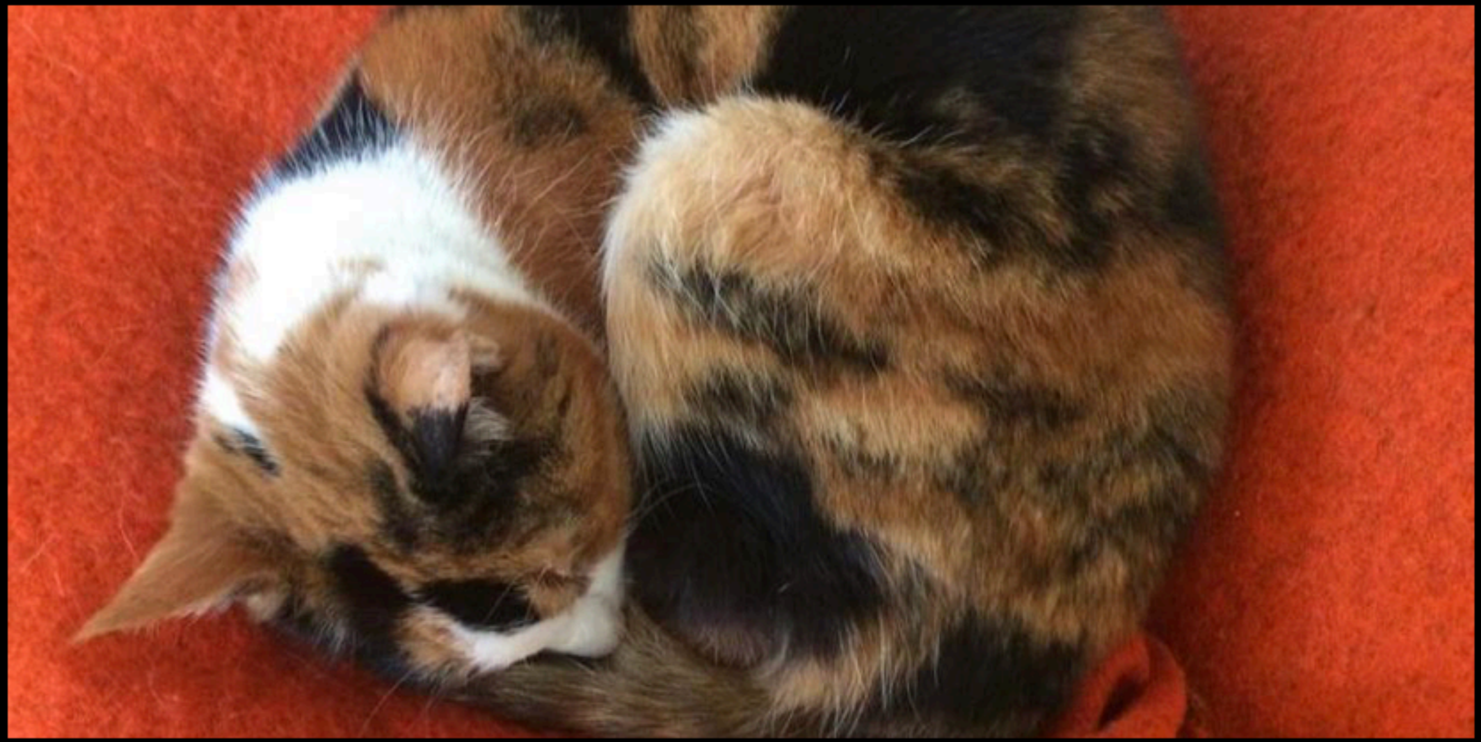


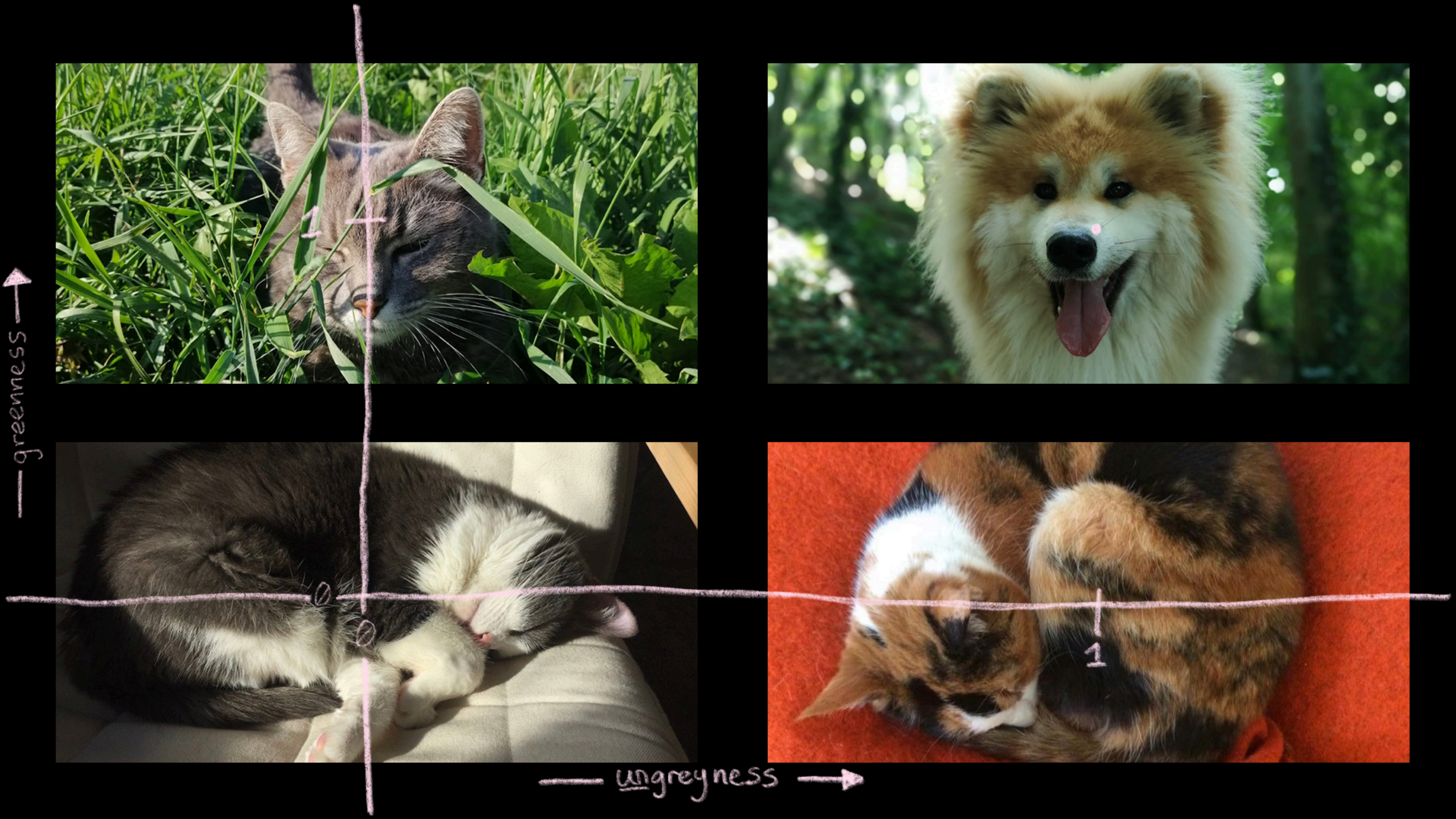


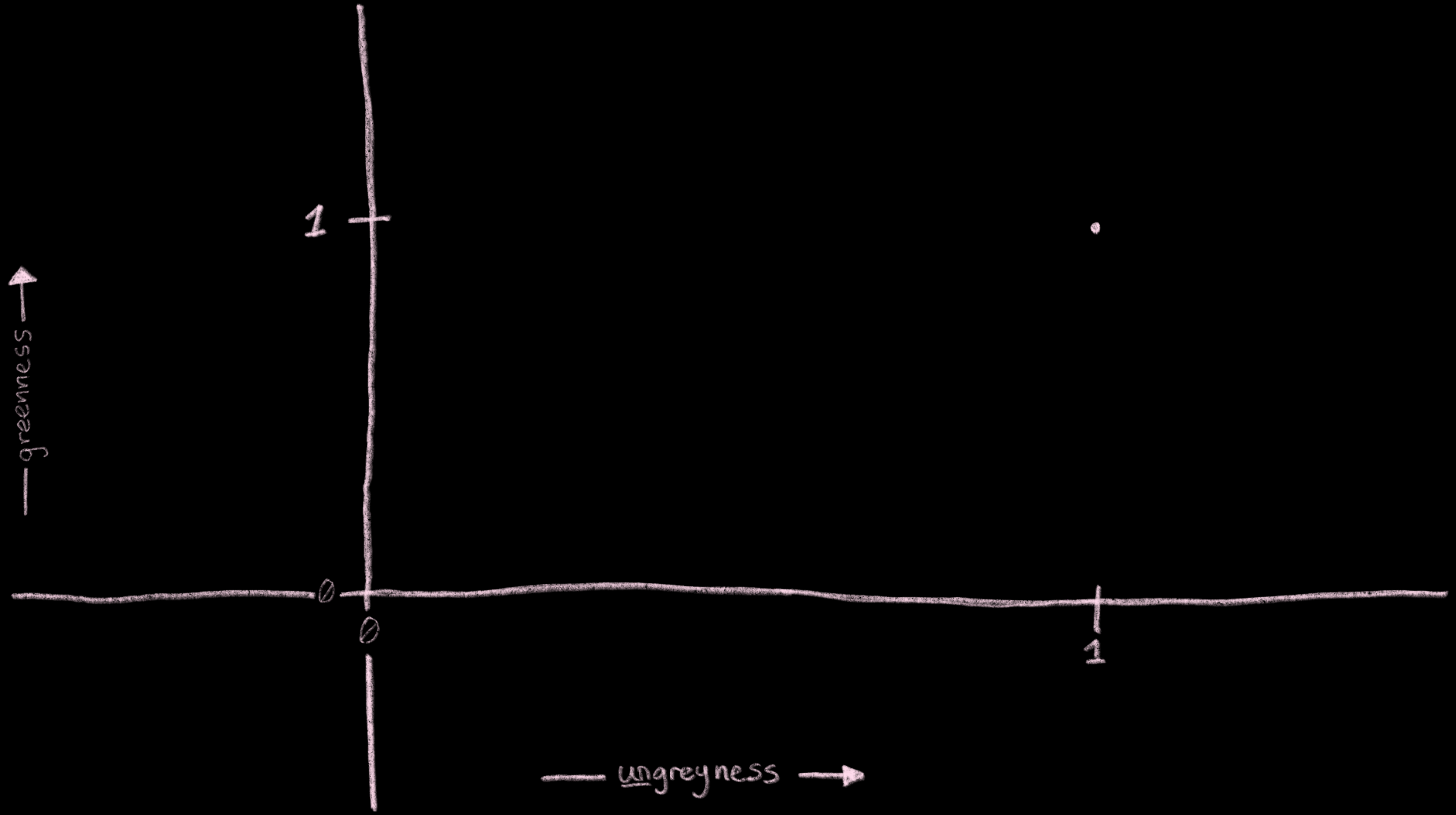


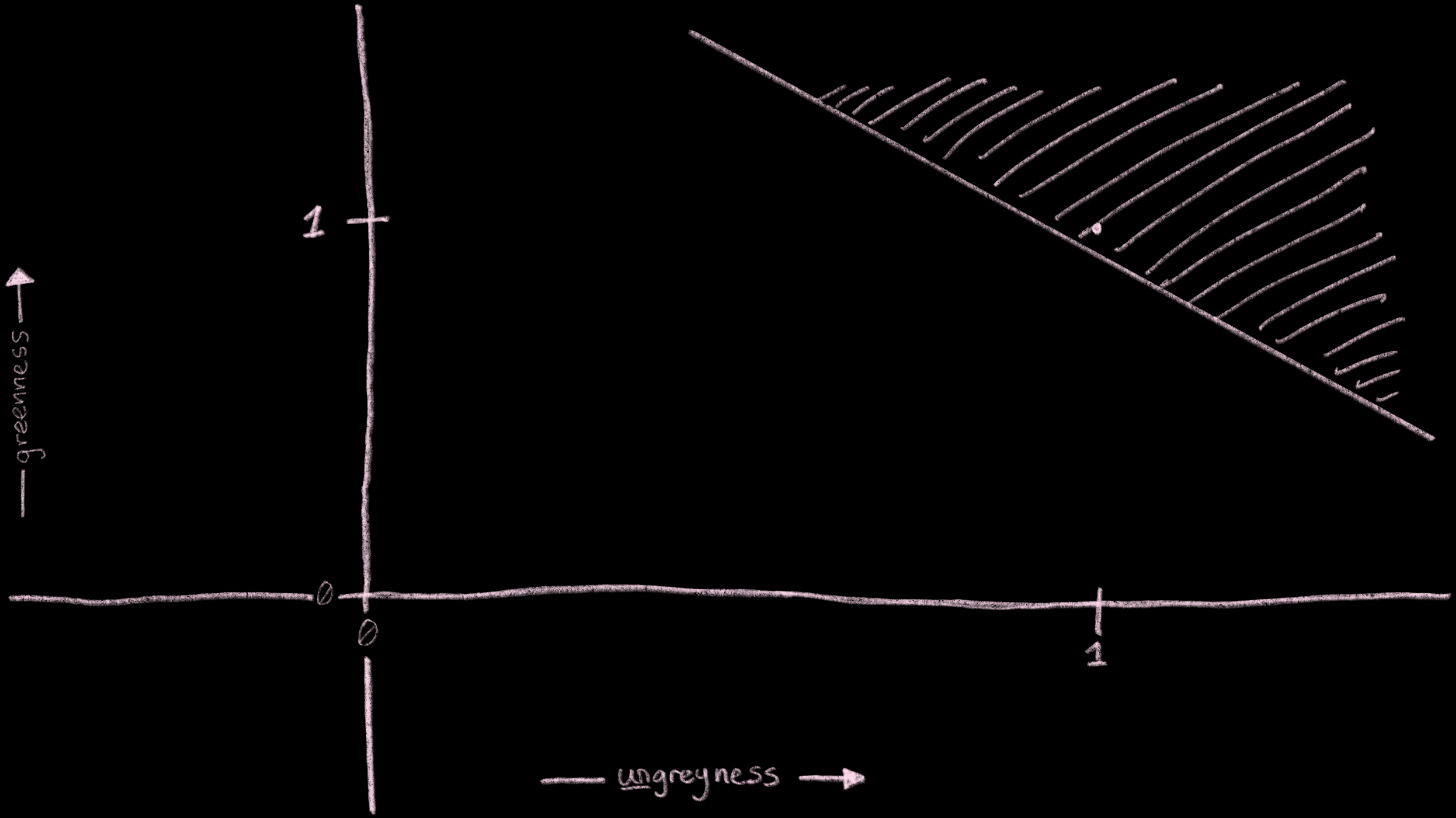



$$\begin{aligned}\mathrm{dog}(\bar{x}) \quad &:= \quad \mathrm{lots\_of\_green}(\bar{x}) \\ &\wedge \quad \mathrm{very\_little\_grey}(\bar{x})\end{aligned}$$

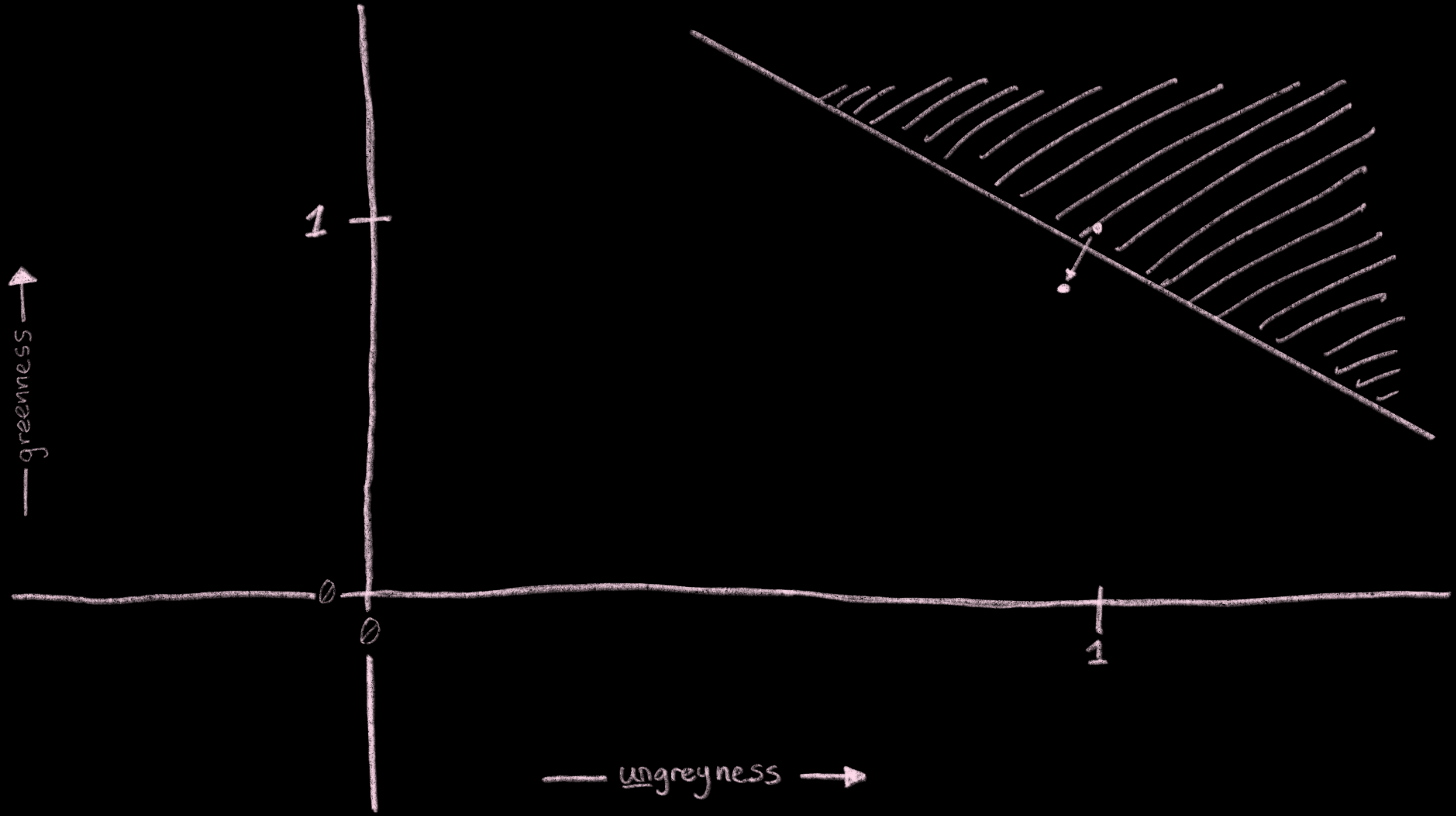


greenness
ungreyness
1
0
0
1

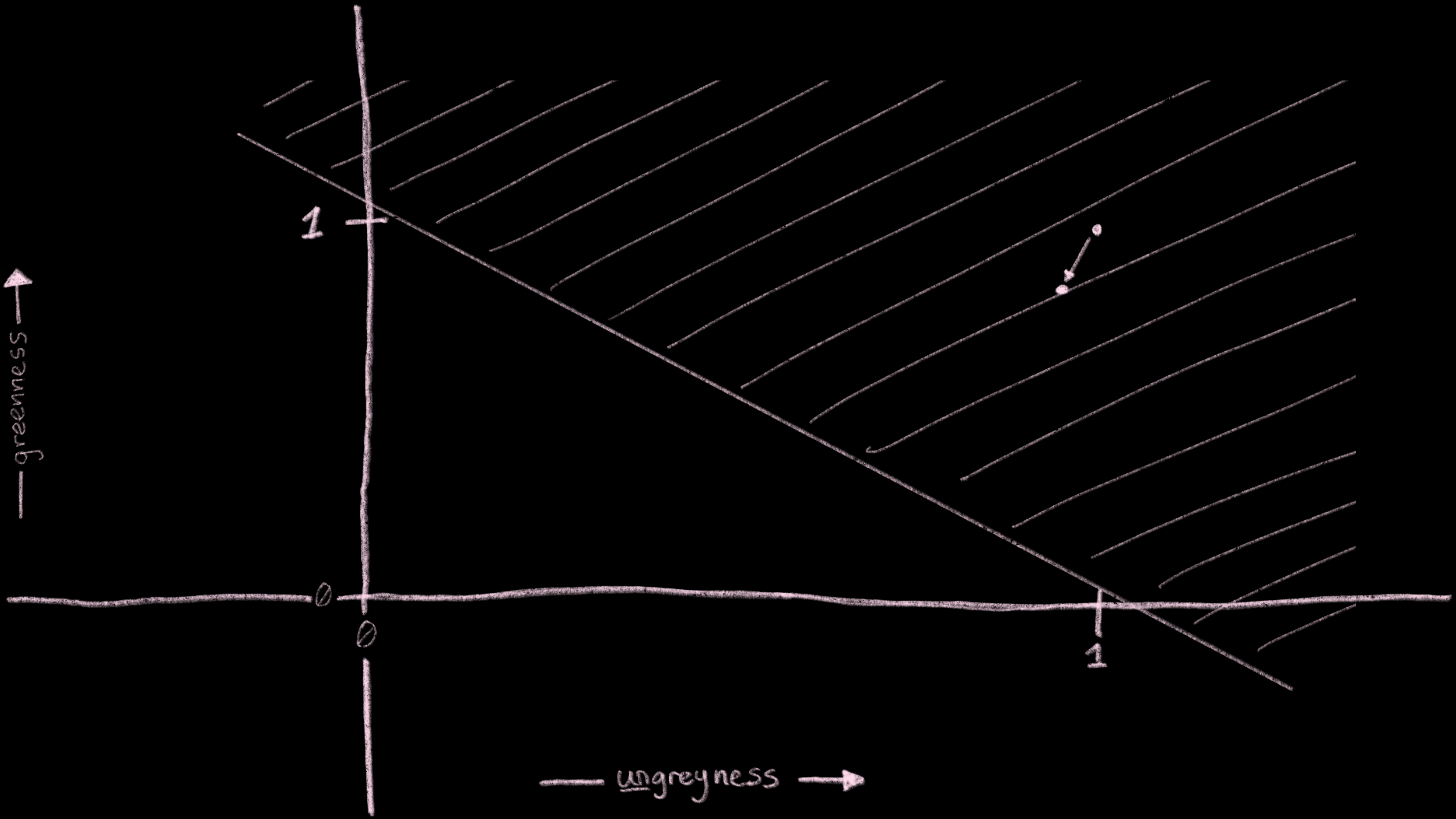

1
greenness
0
0
1
ungreyness

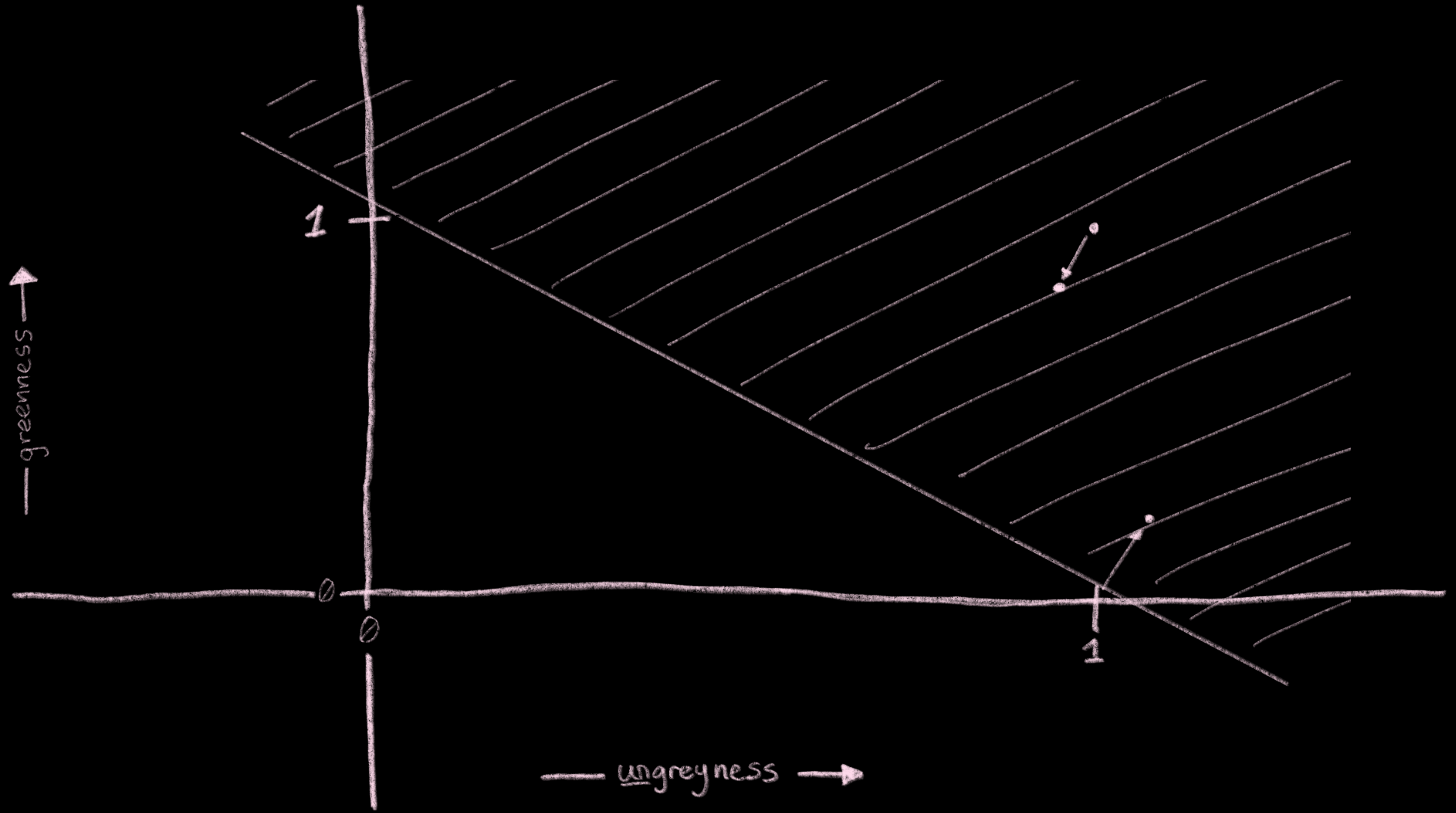


1
greenness
0
0
1
ungreyness

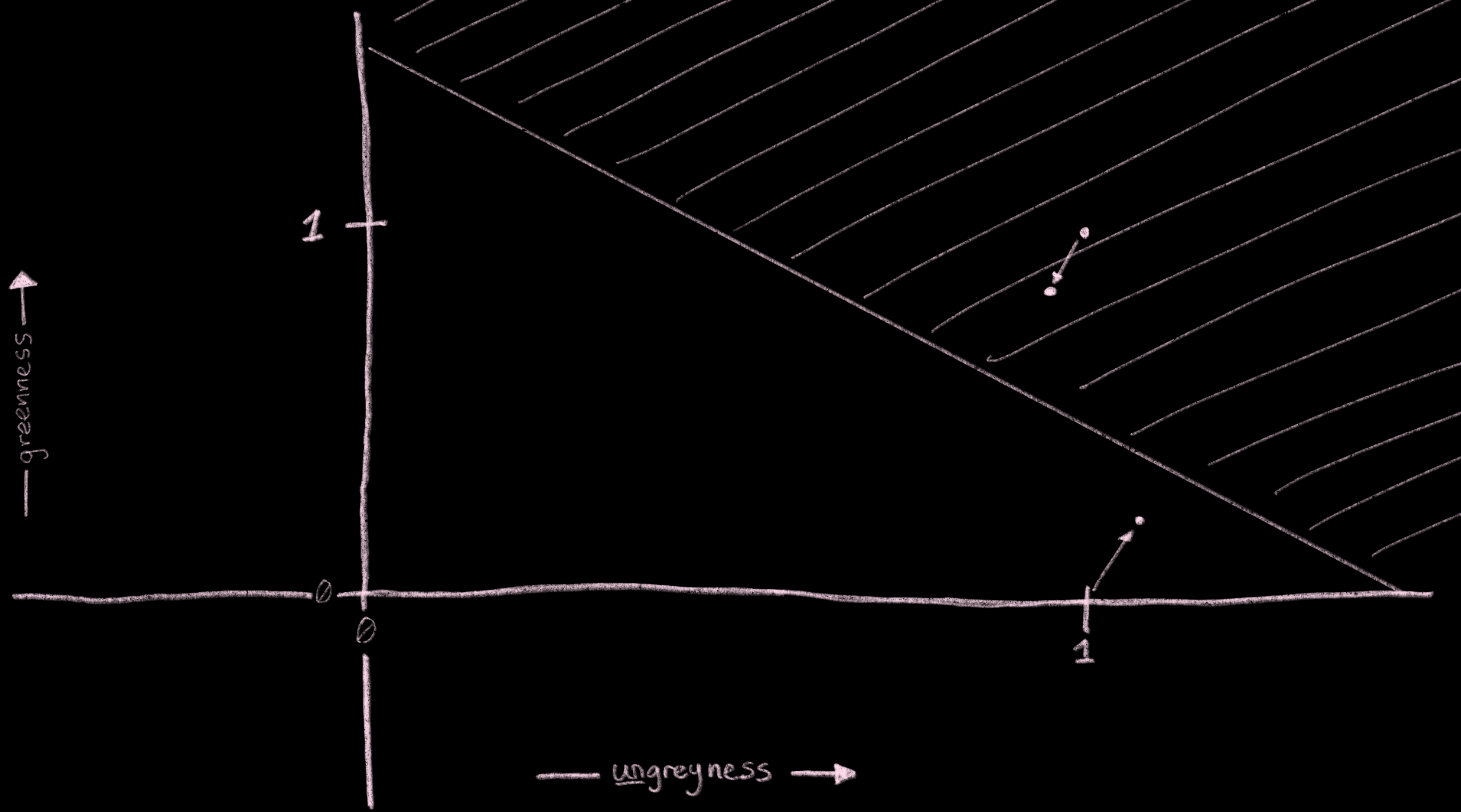


1
0
0
1
greenness
ungreyness


1
0
0
1
greenness
ungreyness


1
0
0
1
greenness
ungreyness

- DOG?



1
0
0
1
greenness
ungreyness

WHAT HAVE WE SEEN SO FAR?

WHAT'S STILL TO COME?

Is our data good?

Did we find the right features?

Are we robust around these features?

Is our data good?

Did we find the right features?

Are we robust around these features?

Robustness as a Refinement Type

WHAT'S A REFINEMENT TYPE?

A type refined with an SMT-checkable predicate.

```
let ℝ⁺ = (x:ℝ {0.0R ≤ x}) // positive reals

let _ = 4.0R : ℝ⁺ // 0.0R ≤ 4.0R

let vector a n = (xs:list a {length xs = n}) // lists of length n

let _ = [0.5R; 1.0R] : vector ℝ⁺ 2 // length [0.5R; 1.0R] = 2
```

$$\text{classify} : (x_1 \to \mathbb{R}) \to (x_2 \to \mathbb{R}) \to (y : \mathbb{R})$$
$$\text{classify}\ x_1\ x_2 = f\ (w_1 x_1 + w_2 x_2 - b)$$

$$\text{classify} : (x_1 \to \mathbb{R}) \to (x_2 \to \mathbb{R}) \to (y : \mathbb{R})$$

$$\text{classify}\ x_1\ x_2 = S\ (0.5x_1 + 0.5x_2 - 0.9)$$

$$S\ x = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

```
val model : network (*with*) 2 (*inputs*) 1 (*output*) 1 (*layer*)
let model = NLast // makes single-layer network
  { weights    = [[0.5R]; [0.5R]]
  ; biases     = [-0.9R]
  ; activation = Threshold
  }

val classify : (x1 : ℝ) → (x2 : ℝ) → (y : ℝ)
let classify x1 x2 = run model [x1; x2]
```

```
let ε = 0.1R // how big are tiny steps?

val doggy : (x : ℝ) → bool
let doggy x = 1.0R - ε ≤ x && x ≤ 1.0R + ε

val _ = (x1 : ℝ{doggy x1})
      → (x2 : ℝ{doggy x2})
      → (y  : ℝ{y = 1.0R})
val _ = classify
```

```
(define-fun classify ((x1 Real) (x2 Real)) Real
  (ite (>= (- (+ (* x1 0.5) (* x2 0.5)) 0.9) 0.0) 1.0 0.0))
(define-fun doggy ((x Real)) Bool (and (<= 0.9 x) (<= x 1.1)))
(assert (forall ((x1 Real) (x2 Real))
  (=> (and (doggy x1) (doggy x2)) (= (classify x1 x2) 1.0))))
(check-sat)

> sat ;; it works! your network is totally robust! gj!
```

So it works.
But does it WORK?

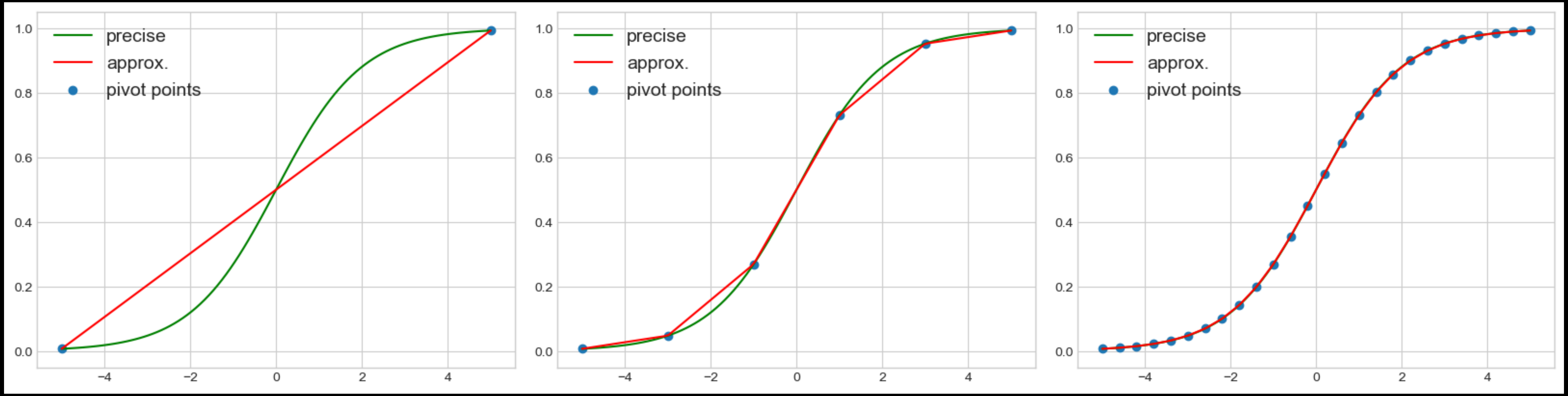
① Solvers don't do non-linear arithmetic

② Integration with F* introduces a significant slowdown

③ Solvers don't scale to realistic sizes

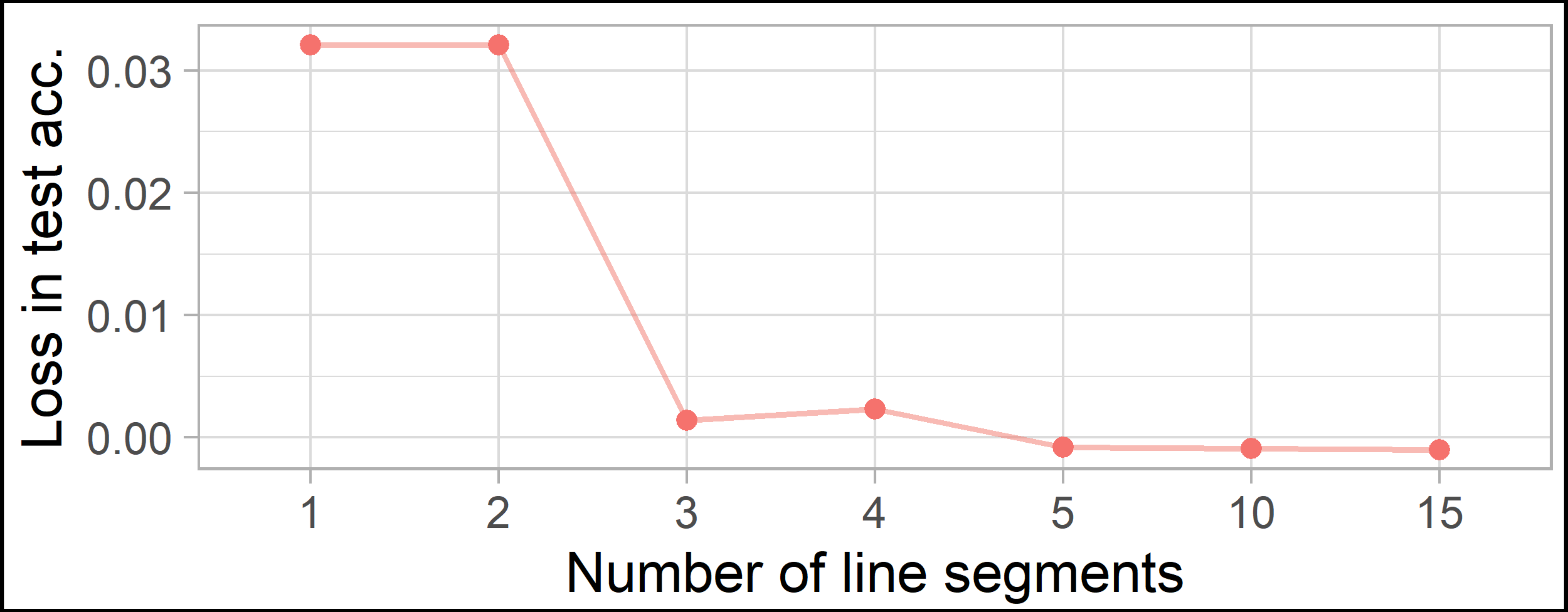
① Solvers don't do non-linear arithmetic

Let's make our activation functions linear!

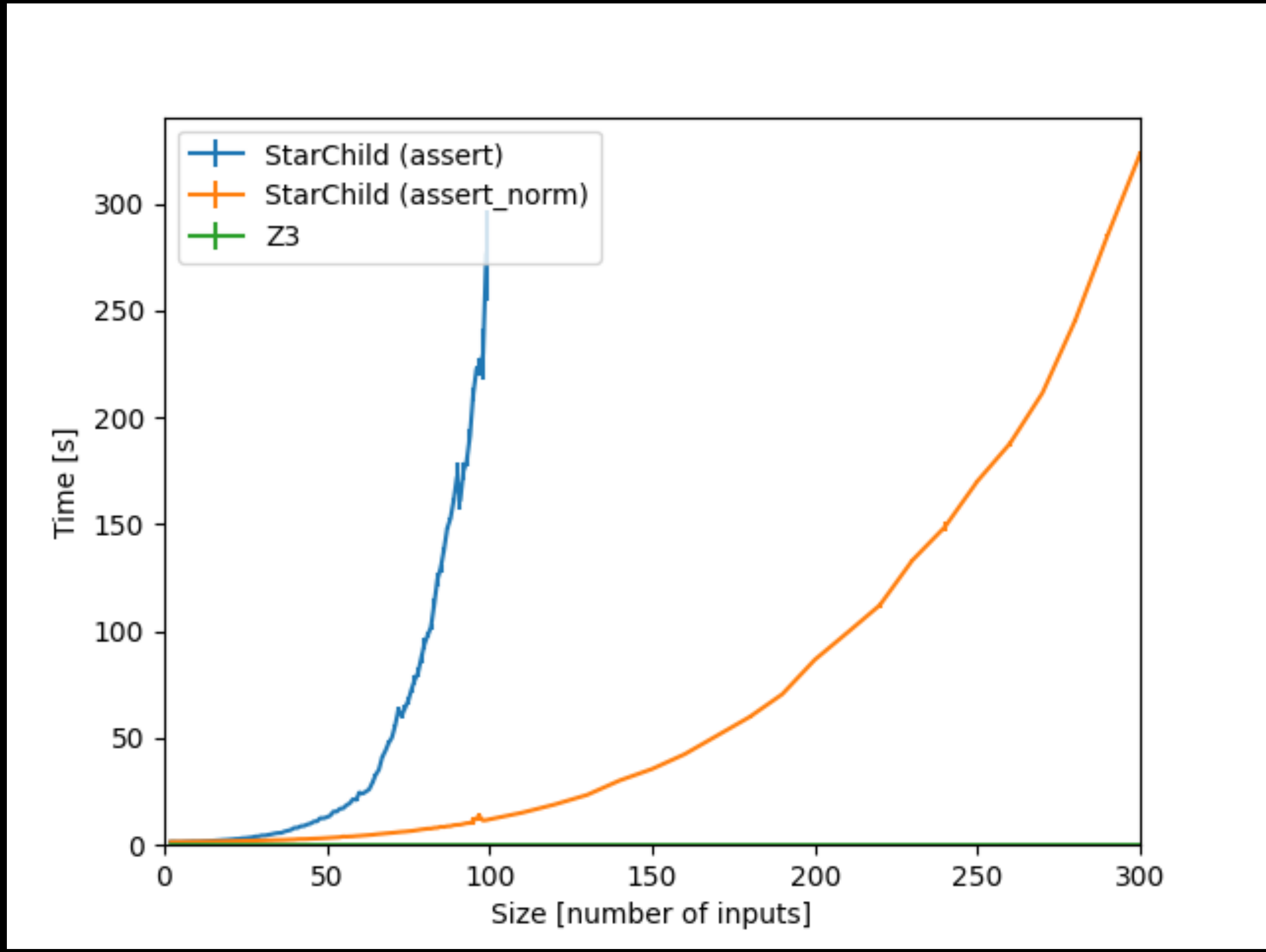


① Solvers don't do non-linear arithmetic

Train with tanh, run with linear approximation!



② Integration with F* introduces a significant slowdown



Ahh! An exponential!

Don’t make Z3 do reduction!

Don’t tell Z3 about data-types.

(Unless you have to.)

③ Solvers don't scale to realistic sizes

Z3 ignores tons of structure!

MetiTarski solves <u>exponentials</u>!

nnenum solves <u>ReLUs</u>!

Marabou solves <u>piecewise-linear functions</u>!

Robustness as a Refinement Type

- encode robustness as a refinement type
- leverage existing integration with solvers
- lightweight verification of robustness

but

- need to improve integration with solvers
- need more flexibility in choosing solvers