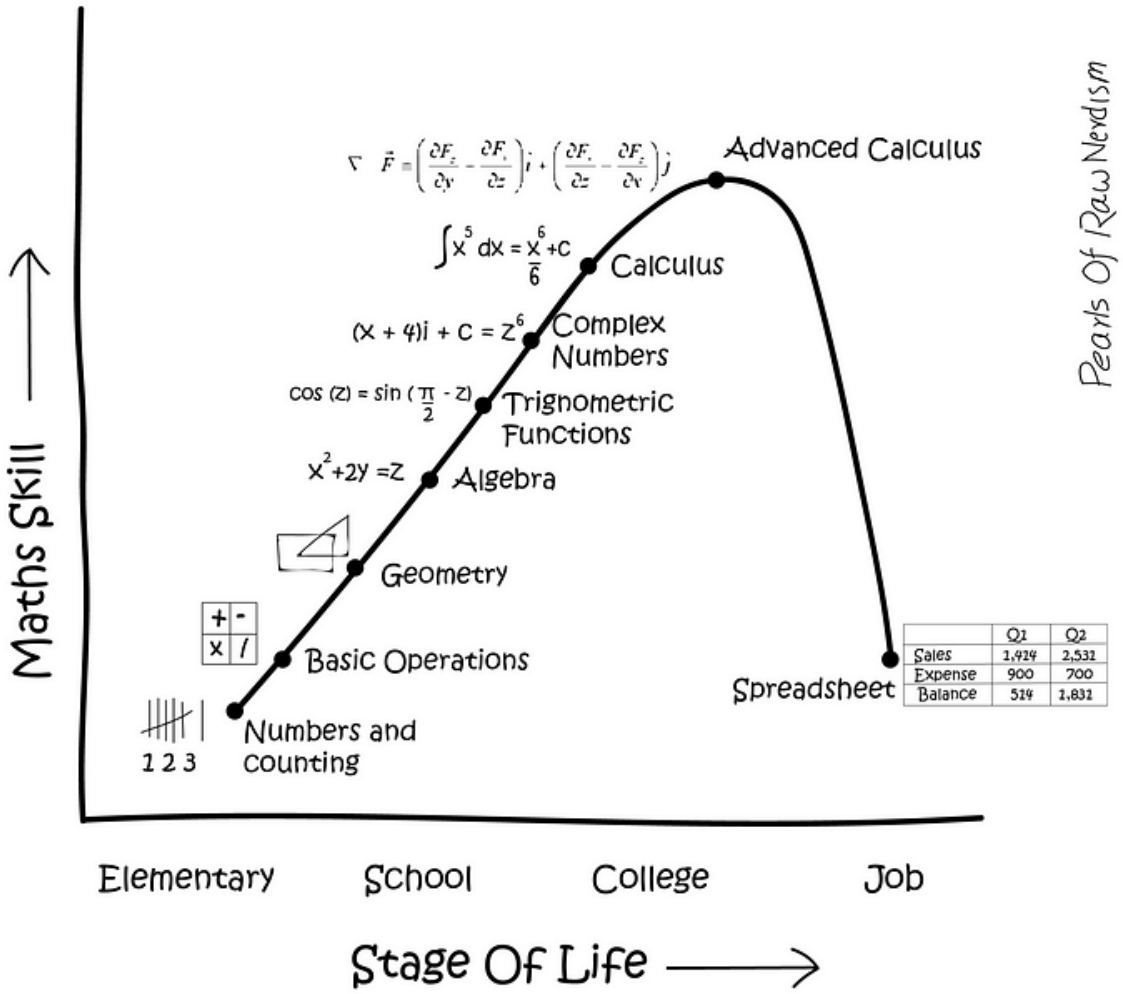
UNIVERSITÀ DI PAVIA

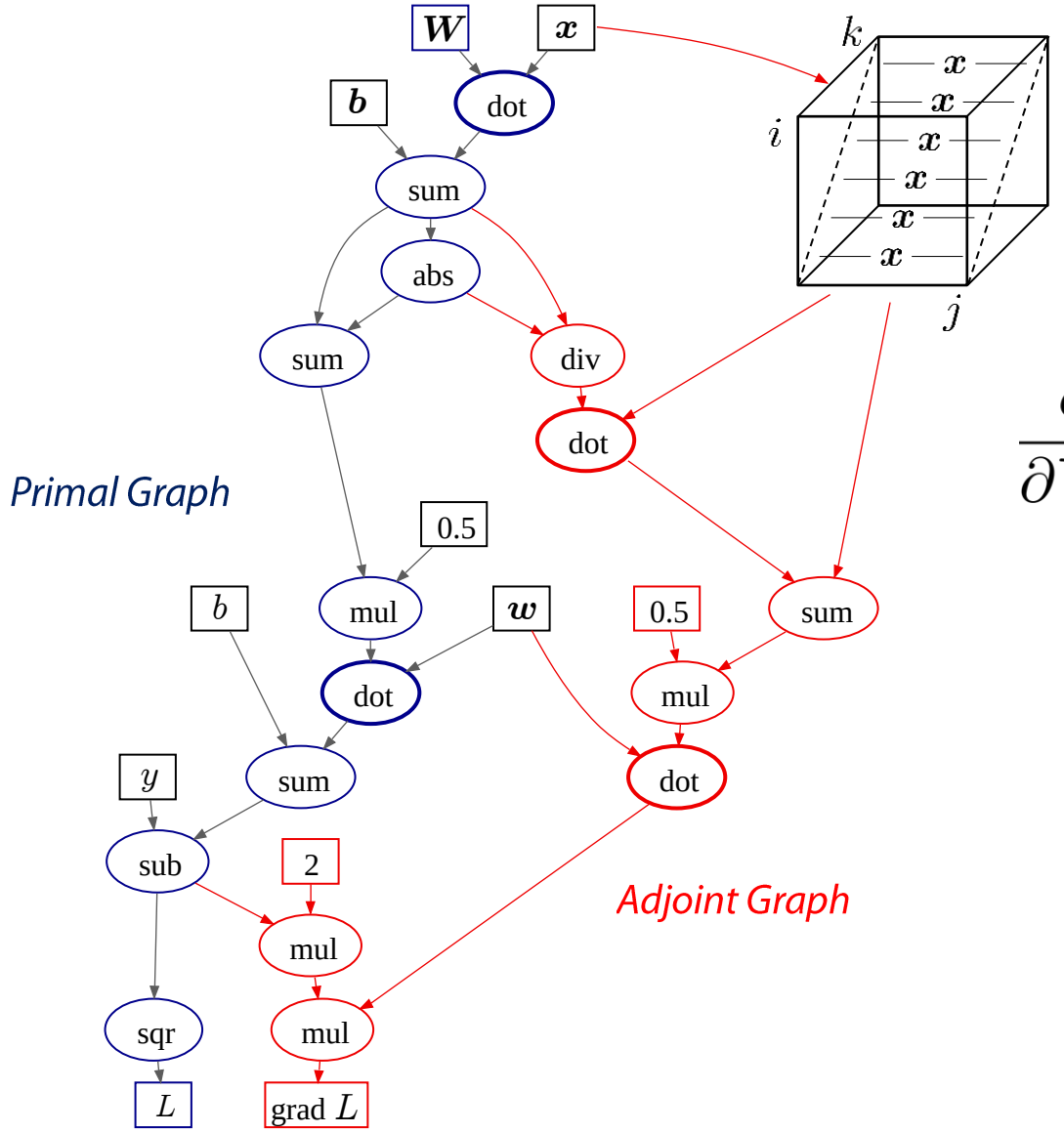
Aside 4: Differentiating Algorithms?

Aside the Aside



[Image from: https://medium.com/passivelogic/intro-to-differentiable-swift-part-0-why-automatic-differentiation-is-awesome-a522128ca9e3]

Automatic Differentiation (AD): Graph-Based



$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

AD of Flow Control Structures

- Differentiating Any Functions

```
def pow(x, n):
    r = 1
    while n > 0:
        n -= 1
        r *= x
    return r
```

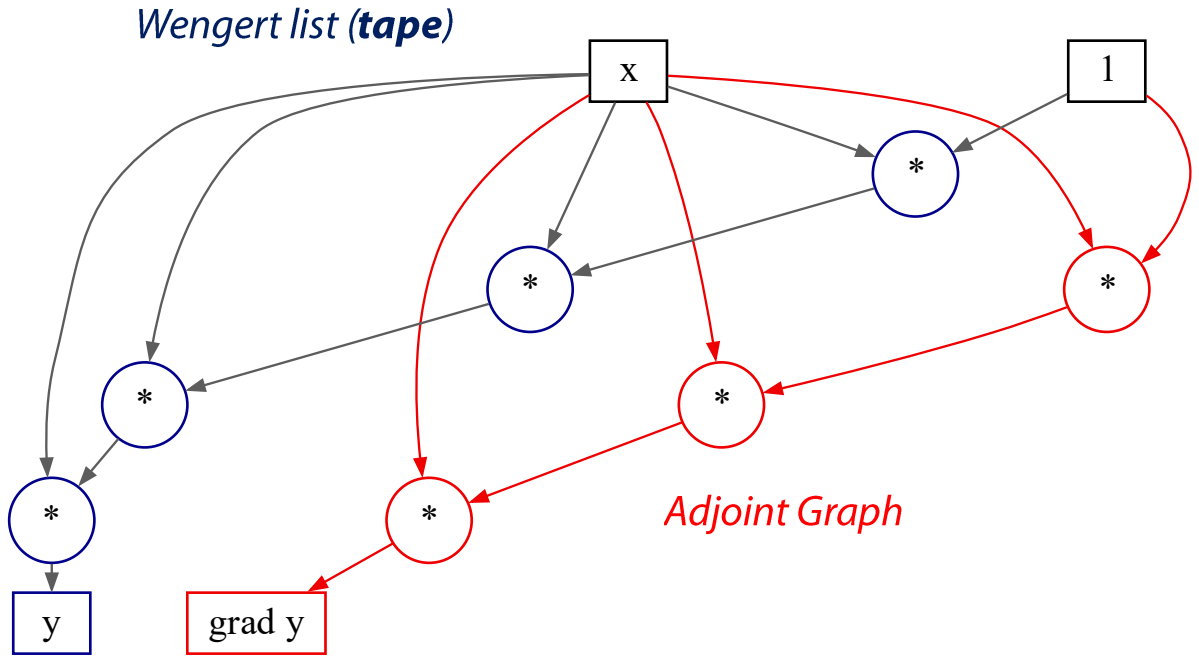
How can a `while` *structure be differentiated?*

Consider the runtime trace of a particular execution:

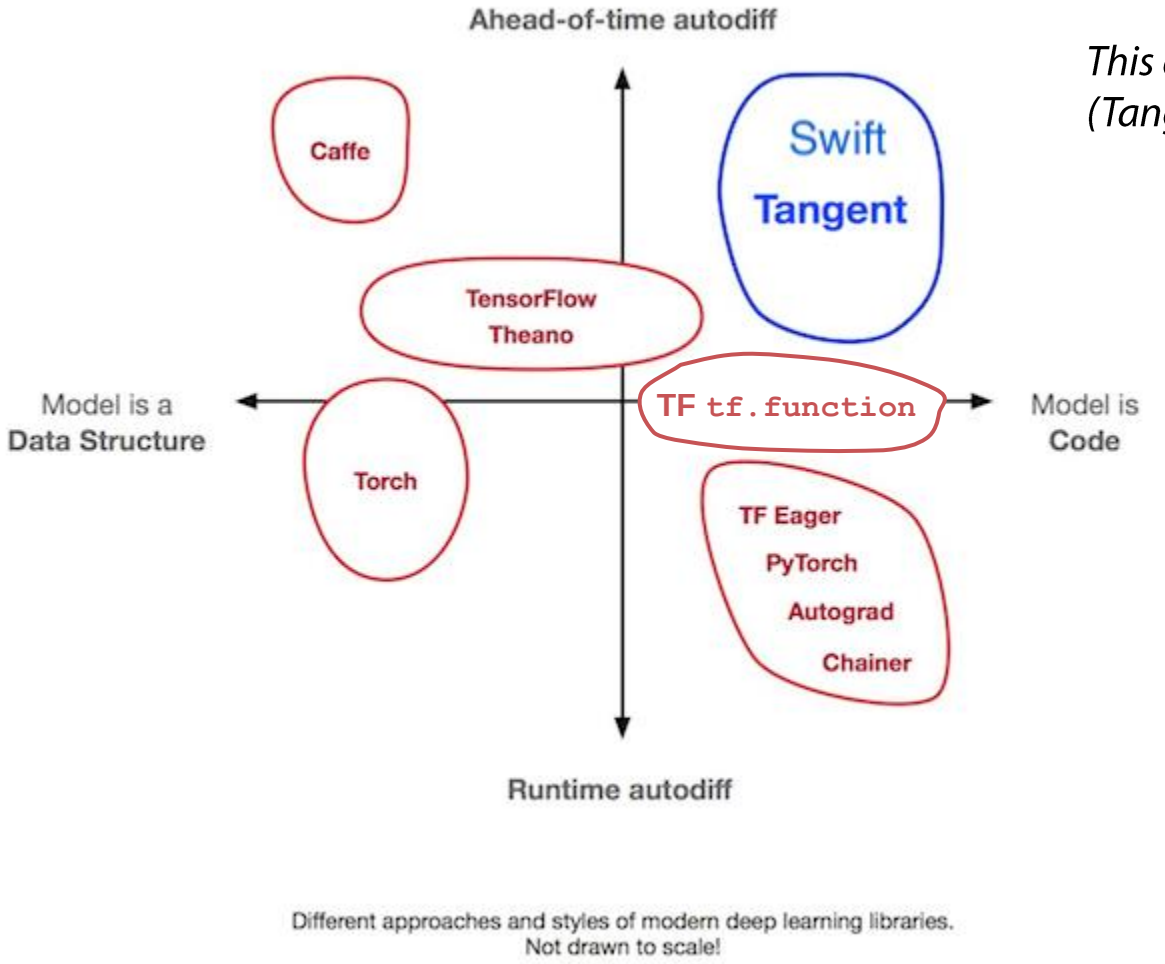
```
y = pow(x, 4)

grad(y) = ?
```

```
r = 1
r = r * x
r = r * x
r = r * x
r = r * x
y = r
```

It is also called *Wengert list*, or ***tape***



AD Frameworks



Different approaches and styles of modern deep learning libraries.
Not drawn to scale!

This diagram is a bit obsolete now
(Tangent was archived in 2021)

[Image edited from: https://github.com/tensorflow/swift/blob/main/docs/AutomaticDifferentiation.md]

AD strategies

- ***Graph-based***
  - It must be constructed explicitly, by the programmer
  - The primal graph and the adjoint graph can be both constructed once and for all
  - The combination of both graphs can be optimized as much as needed
  - Memory blocks need only be allocated at runtime and reclaimed once not used

  Programming is cumbersome and counter-intuitive (with control structures, in particular)

- ***Wengert List ('trace', 'Tape-based')***
  - It can be constructed automatically, at runtime
  - The primal graph must be collected each time, the adjoint graph needs to be computed each time and 'on the fly'
  - Optimization introduces a runtime overhead: apply with care
  - Memory in the primal graph needs be kept allocated until the gradients are computed

  Programming is only slightly different from normal; control structures can be used as usual

Different Frameworks: Engineering Trade-Offs

- **TensorFlow 1.x**

  Construction of static graphs, using a separate language (define-*and*-run)

- **PyTorch 1.x**

  Overloading of Python operators, trace operation on tensors (define-*by*-run)

- **TensorFlow 2.x**

  Eager mode (no `@tf.function` *decorator)*

  Overloading of Python operators, trace operation via tape (define-*by*-run)

  Graph-based, using `@tf.function`

  Decorated function are translated once, on their first execution, into a graph-builder (define-*and*-run)
  Tracing via tape becomes easier (define-*by*-run)

Likely, this is not the end of the story: more solutions are being proposed (e.g., JAX, Julia)