1. **@dispatch(...)** is a decorator provided by the multipledispatch library.
2. When you define:
3. @dispatch(int, int)
4. def product(a, b): ...

it doesn’t just define a function named product. Instead:

A **Dispatcher object** is created (if not already created) with the name product.

The decorated function (i.e., the implementation for (int, int)) is **registered** inside that Dispatcher.

So, yes — the name **product now refers to a Dispatcher object**, not to a normal Python function.

The Dispatcher class has a method called dispatch(...) which:

* + Accepts argument types
  + Looks into its internal **registry (a dictionary)** with type tuples as keys:
  + {
  + (int, int): function\_obj1,
  + (int, int, int): function\_obj2,
  + (float, float, float): function\_obj3,
  + }
  + Returns the correct function for given argument types.

1. When you call:
2. product(10, 20)

it’s:

* + Delegated to the \_\_call\_\_() of the Dispatcher
  + Which looks up the key (int, int)
  + Finds the matching function
  + And invokes it with your arguments

**Summary in Simple Steps:**

* product → refers to Dispatcher object
* Dispatcher has registry like:
* {
* (int, int): func1,
* (int, int, int): func2,
* ...
* }
* Call like product(10, 20):
  + Internally does: Dispatcher.\_\_call\_\_(10, 20)
  + Finds (int, int) → func1
  + Invokes func1(10, 20)