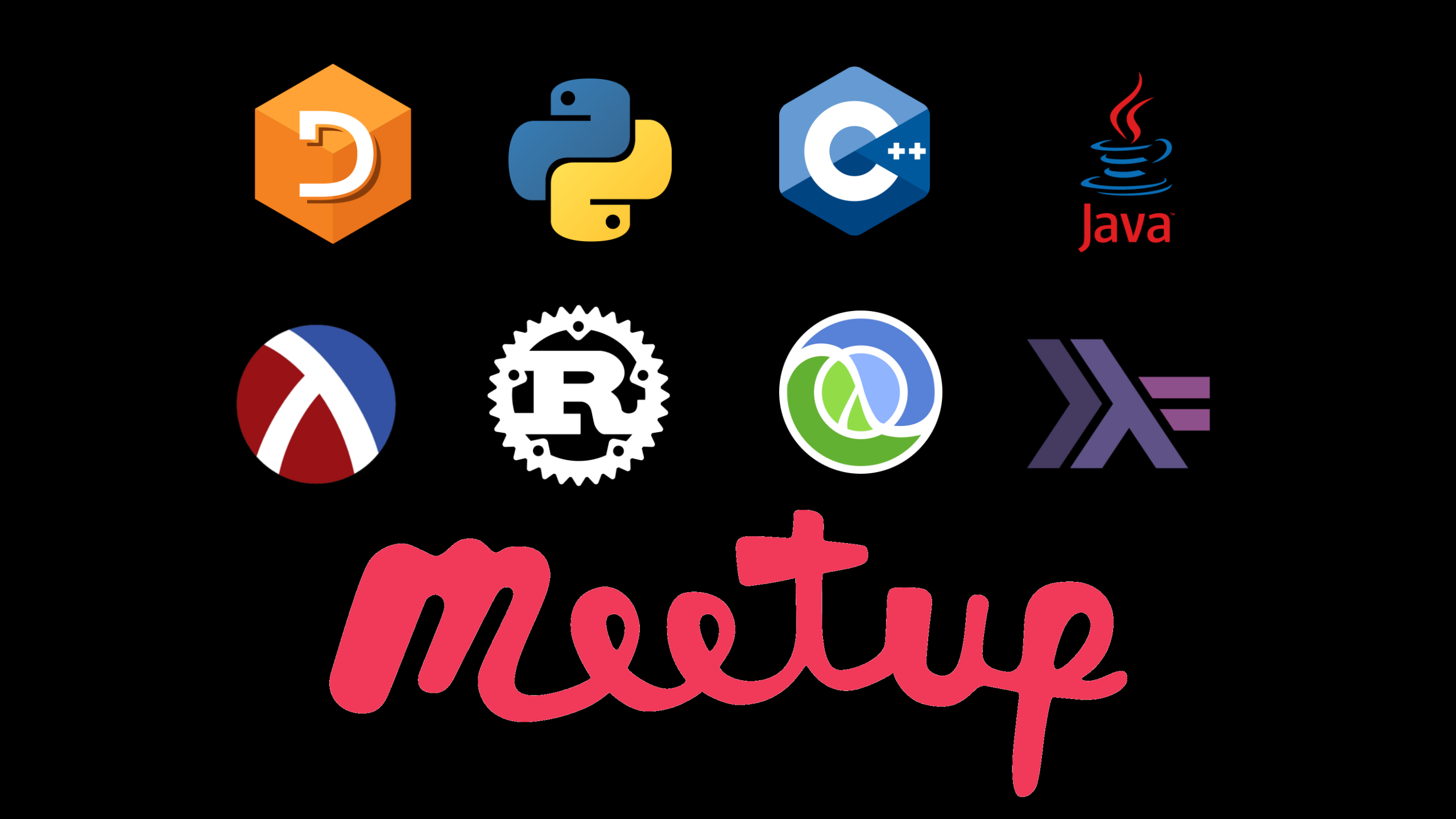
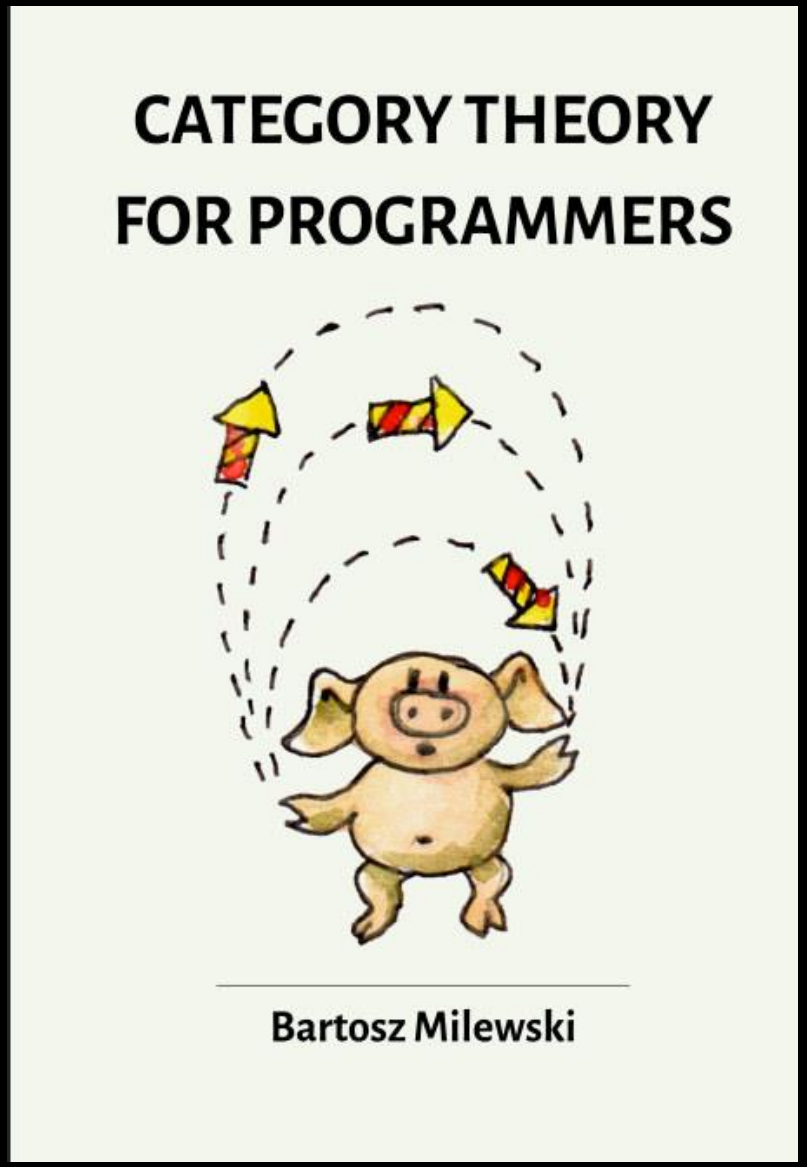

Java
meetup

Friendly Environment Policy

Berlin Code of Conduct



Category Theory for Programmers

Chapter 23: Comonads

```haskell
class Monad m where
    (>=>) :: (a -> m b) -> (b -> m c) -> (a -> m c)
    return :: a -> m a
```

```
class Monad m where
    (>=>) :: (a -> m b) -> (b -> m c) -> (a -> m c)
    return :: a -> m a
```

```
class Functor w => Comonad w where
    (=>=) :: (w a -> b) -> (w b -> c) -> (w a -> c)
    extract :: w a -> a
```

```haskell
class Monad m where
    (>=>) :: (a -> m b) -> (b -> m c) -> (a -> m c)
    return :: a -> m a
```

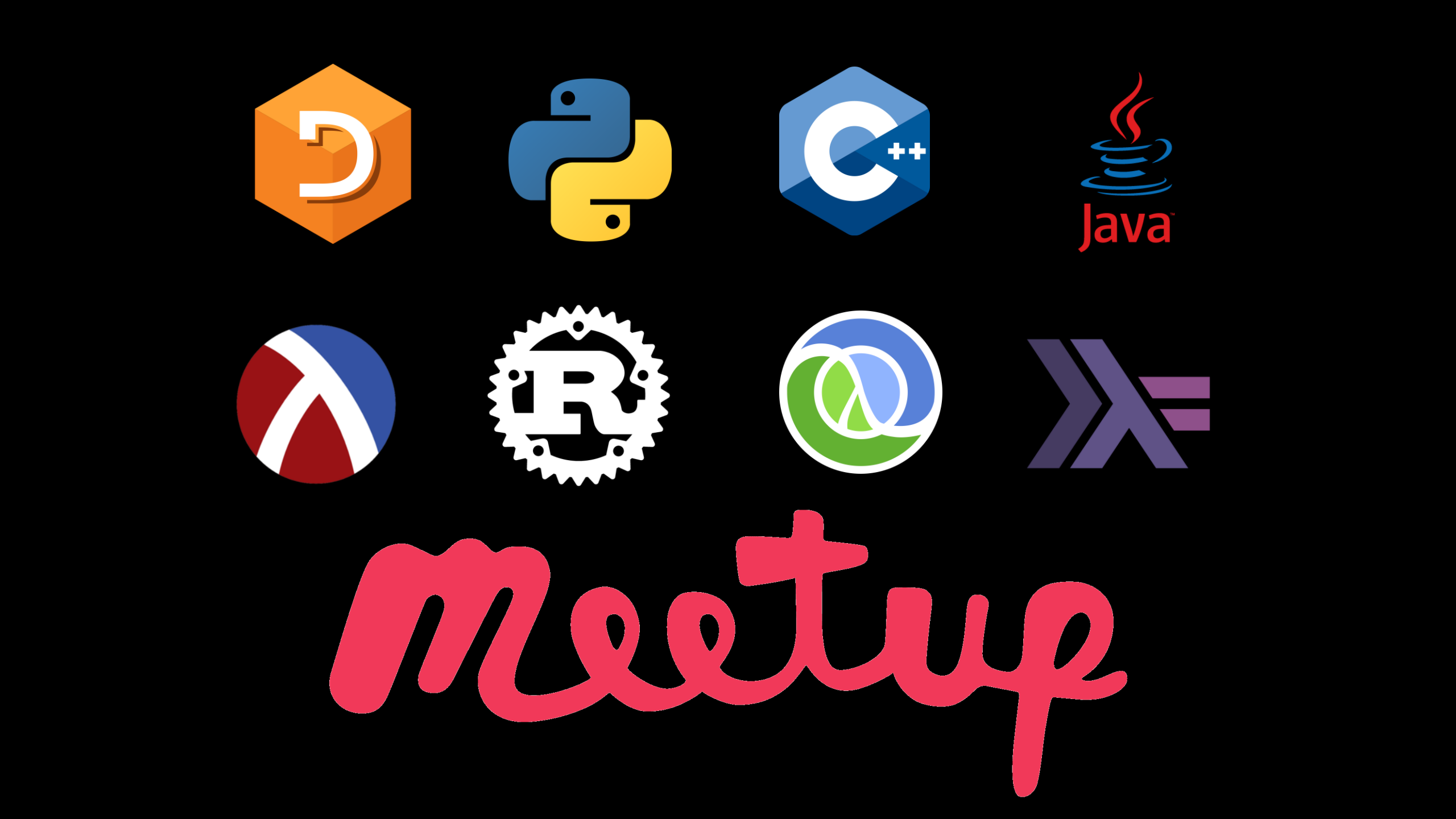
For every monad, instead of defining the fish operator, we may instead define bind. In fact the standard Haskell definition of a monad uses bind:

```haskell
class Monad m where
    (>>=) :: m a -> (a -> m b) -> m b
    return :: a -> m a
```

That leads us to the third option for defining a monad:

```haskell
class Functor m => Monad m where
    join :: m (m a) -> m a
    return :: a -> m a
```

```haskell
class Functor w => Comonad w where
    extract :: w a -> a
    duplicate :: w a -> w (w a)
    duplicate = extend id
    extend :: (w a -> b) -> w a -> w b
    extend f = fmap f . duplicate
```


Java
meetup