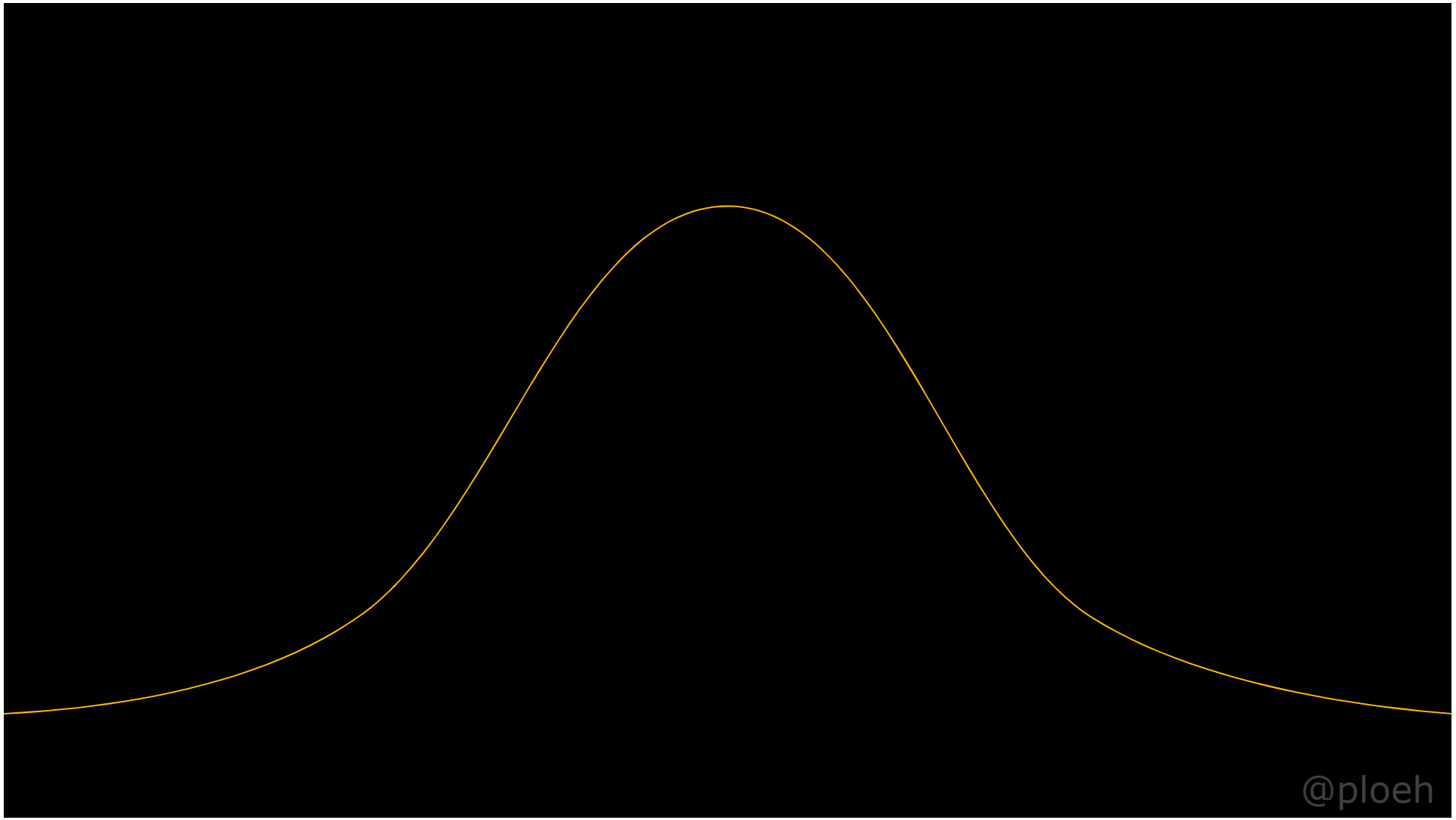
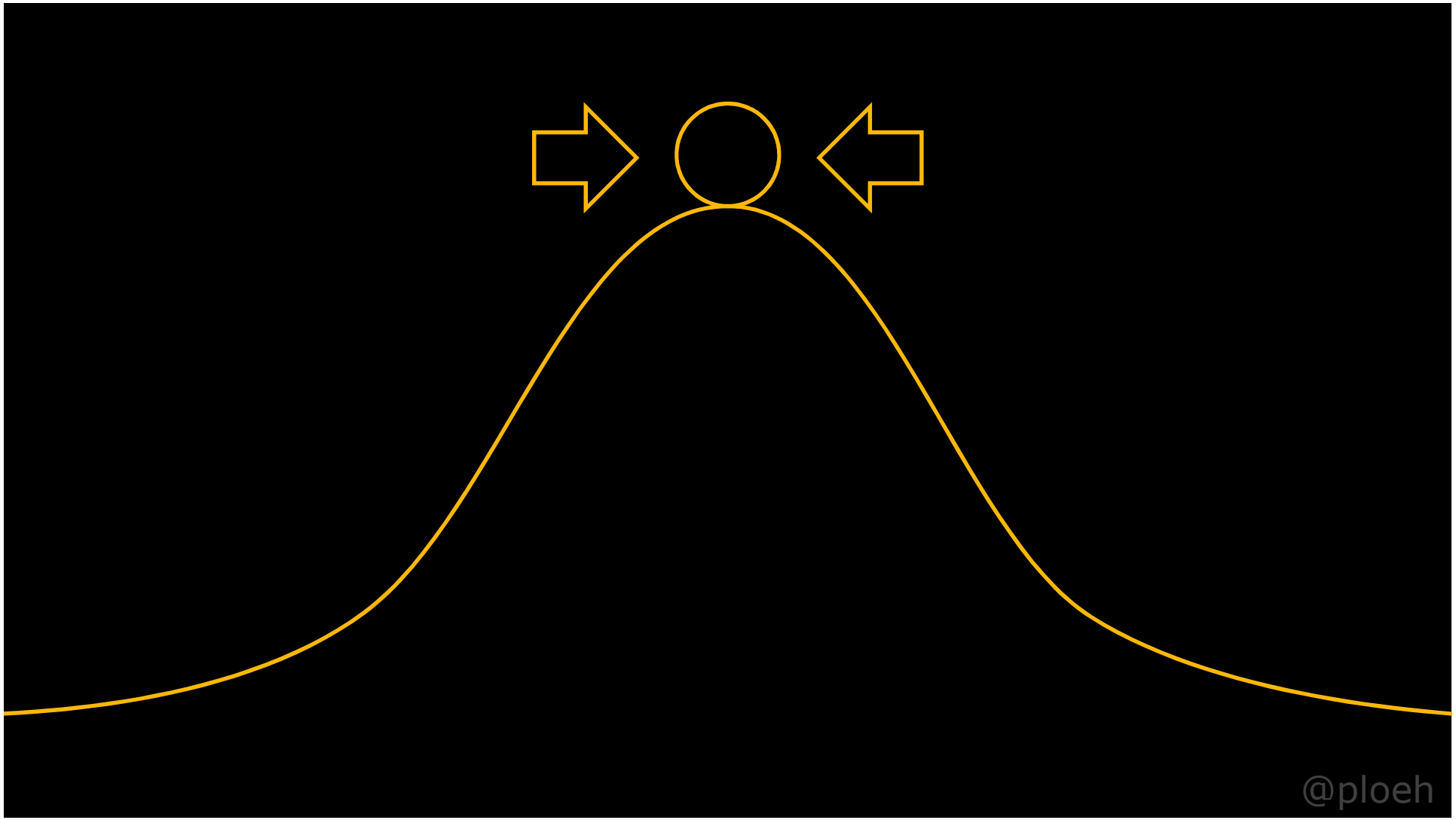
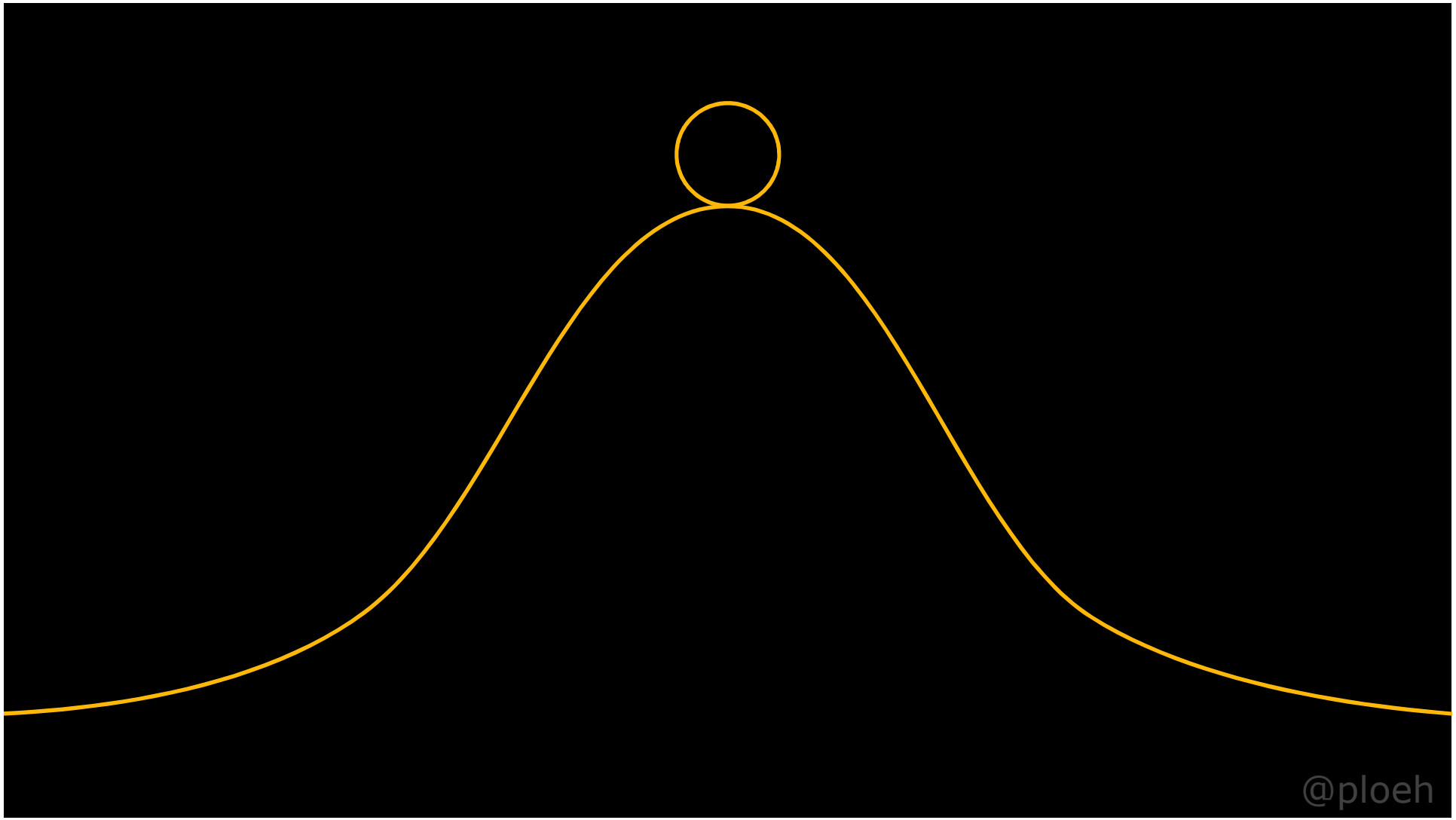
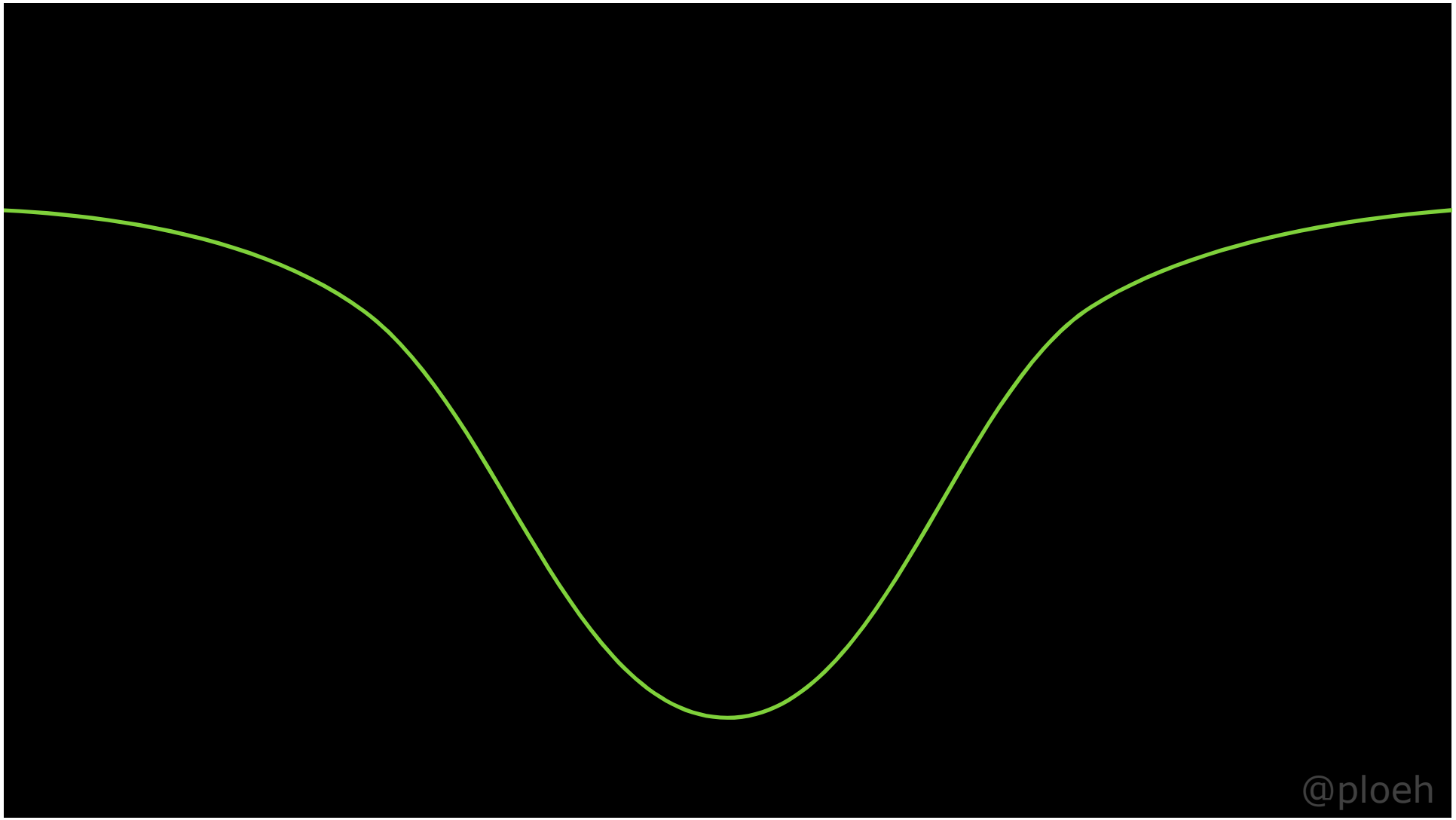
Functional architecture
The pits of success

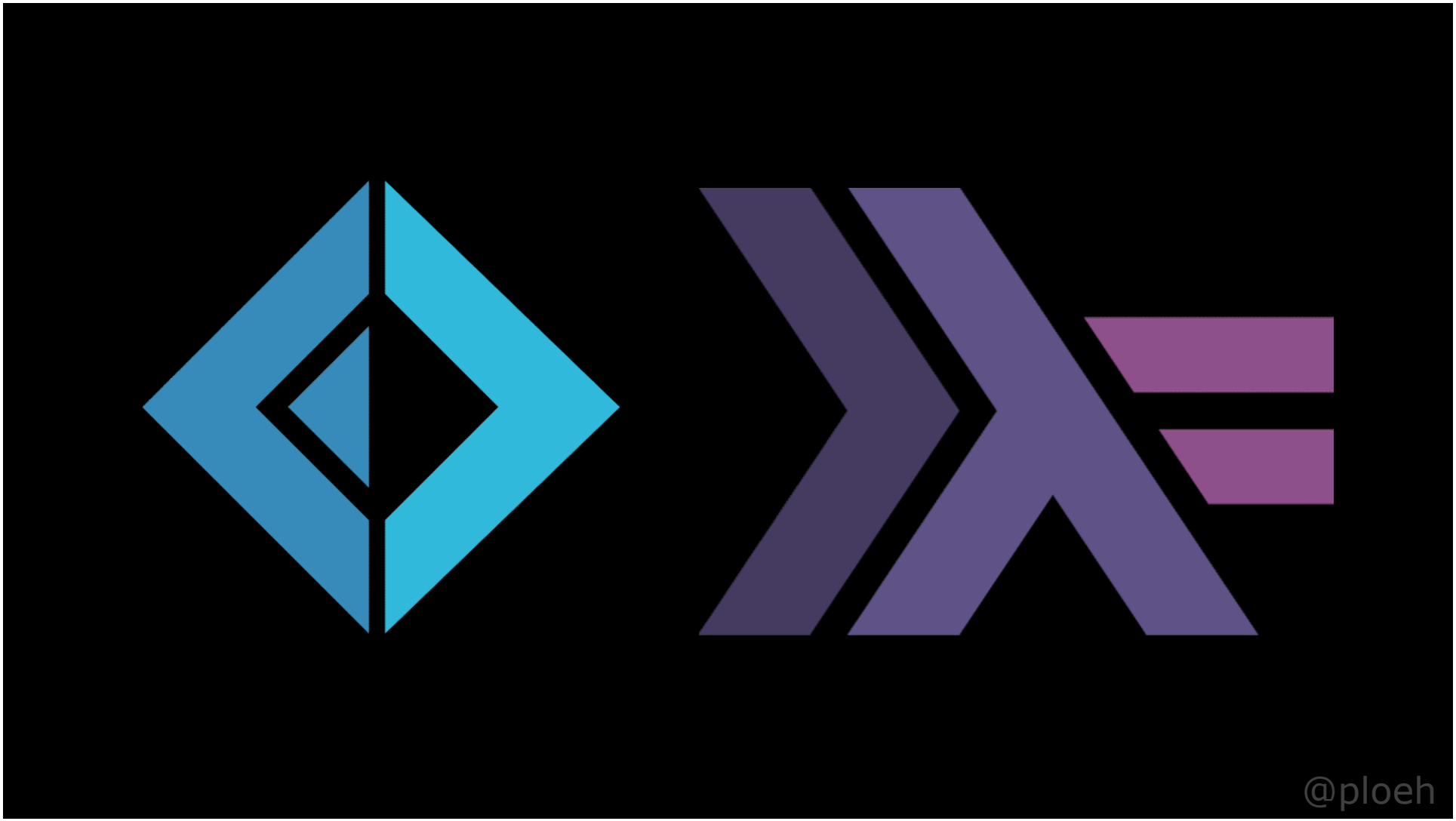
Mark Seemann
http://blog.ploeh.dk
@ploeh

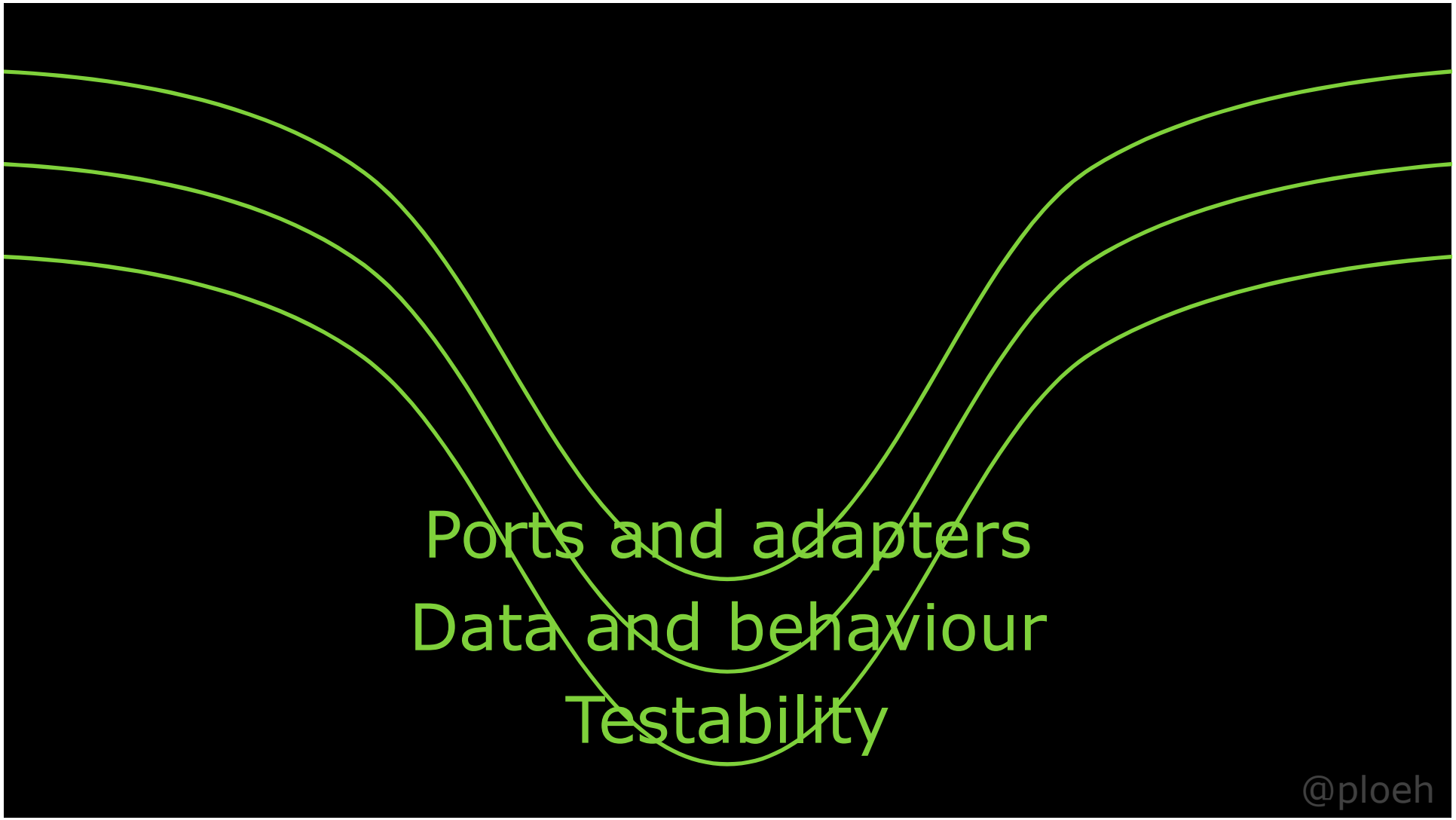

@ploeh


@ploeh

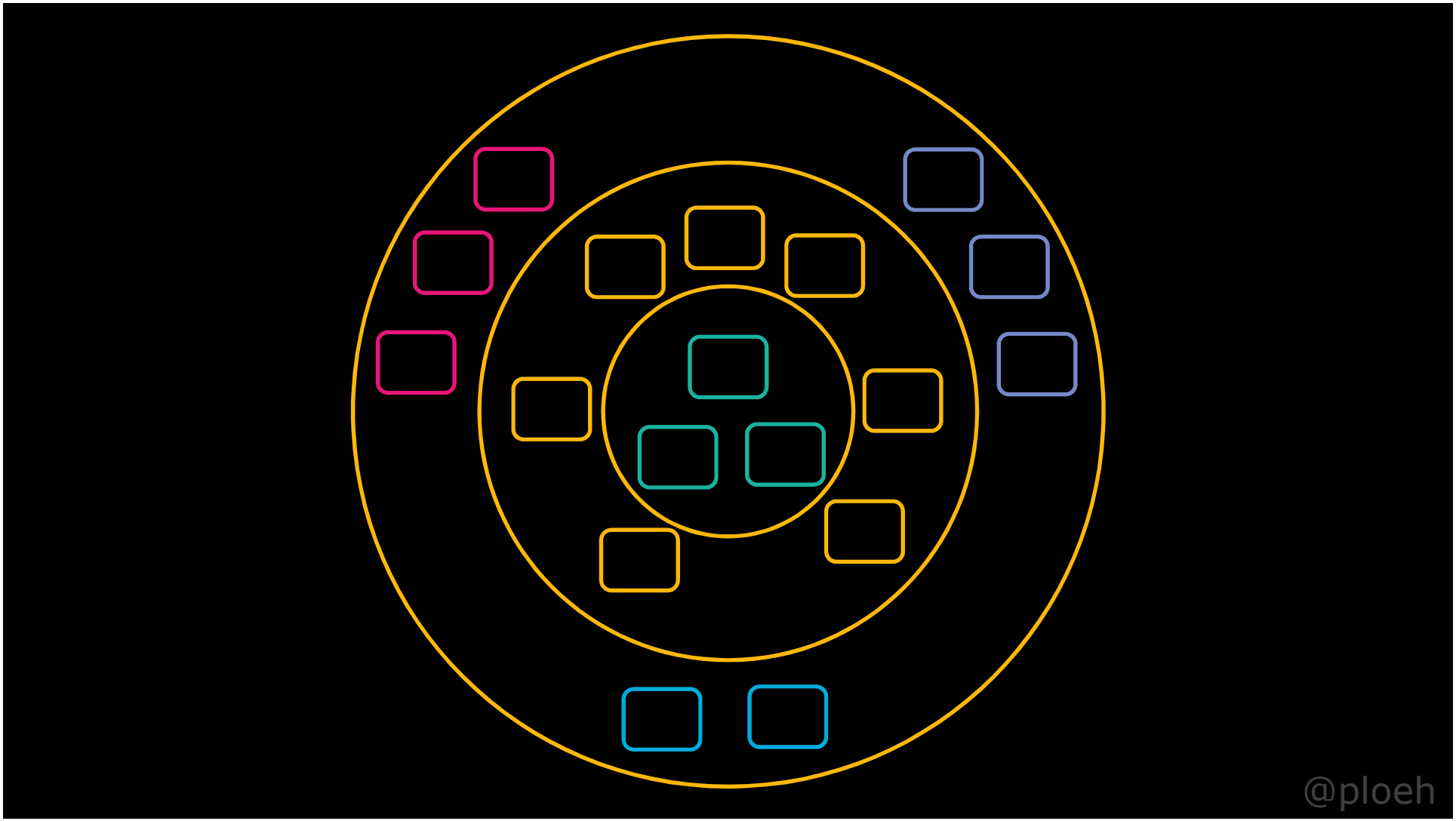

@ploeh

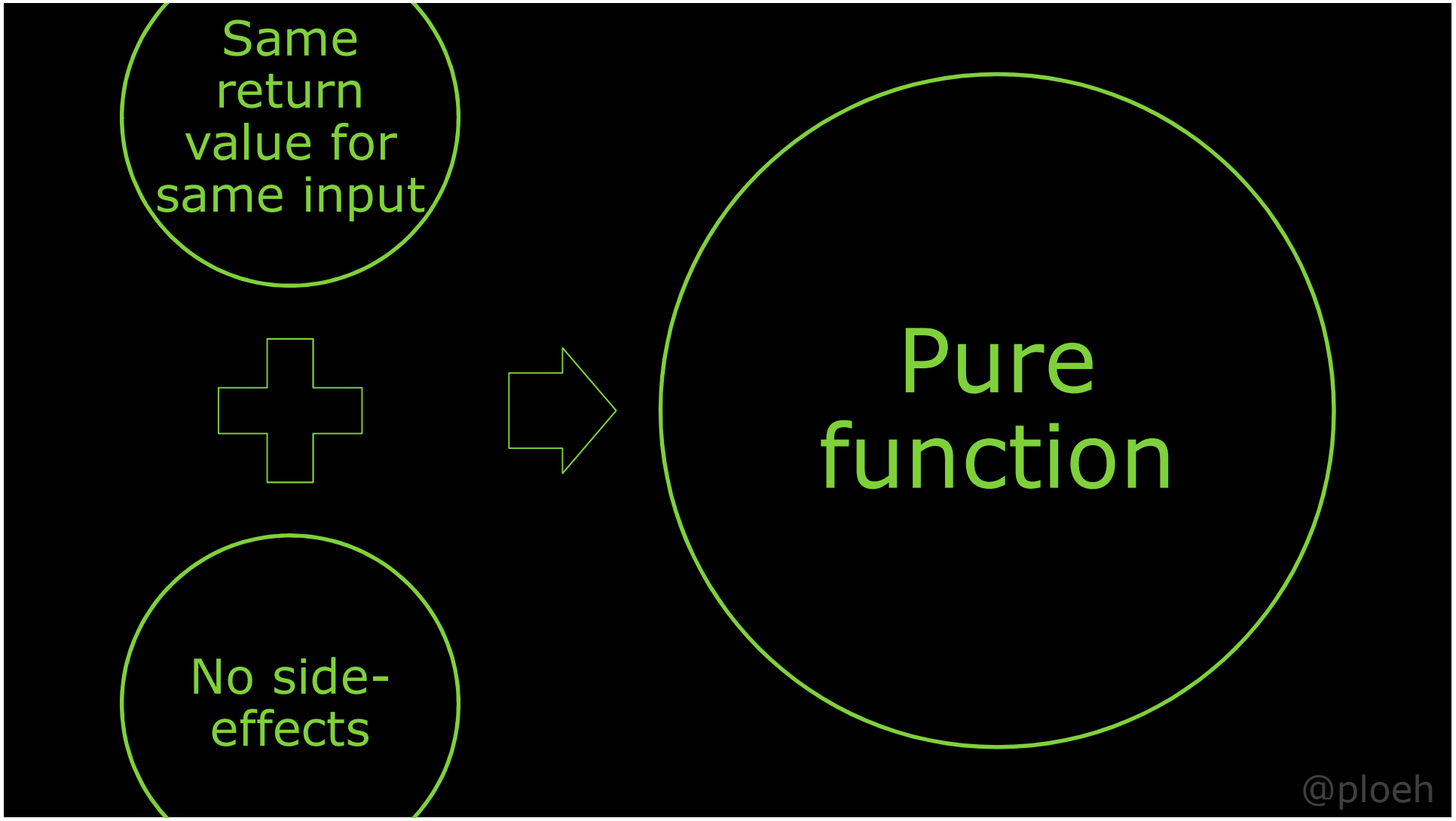

@ploeh

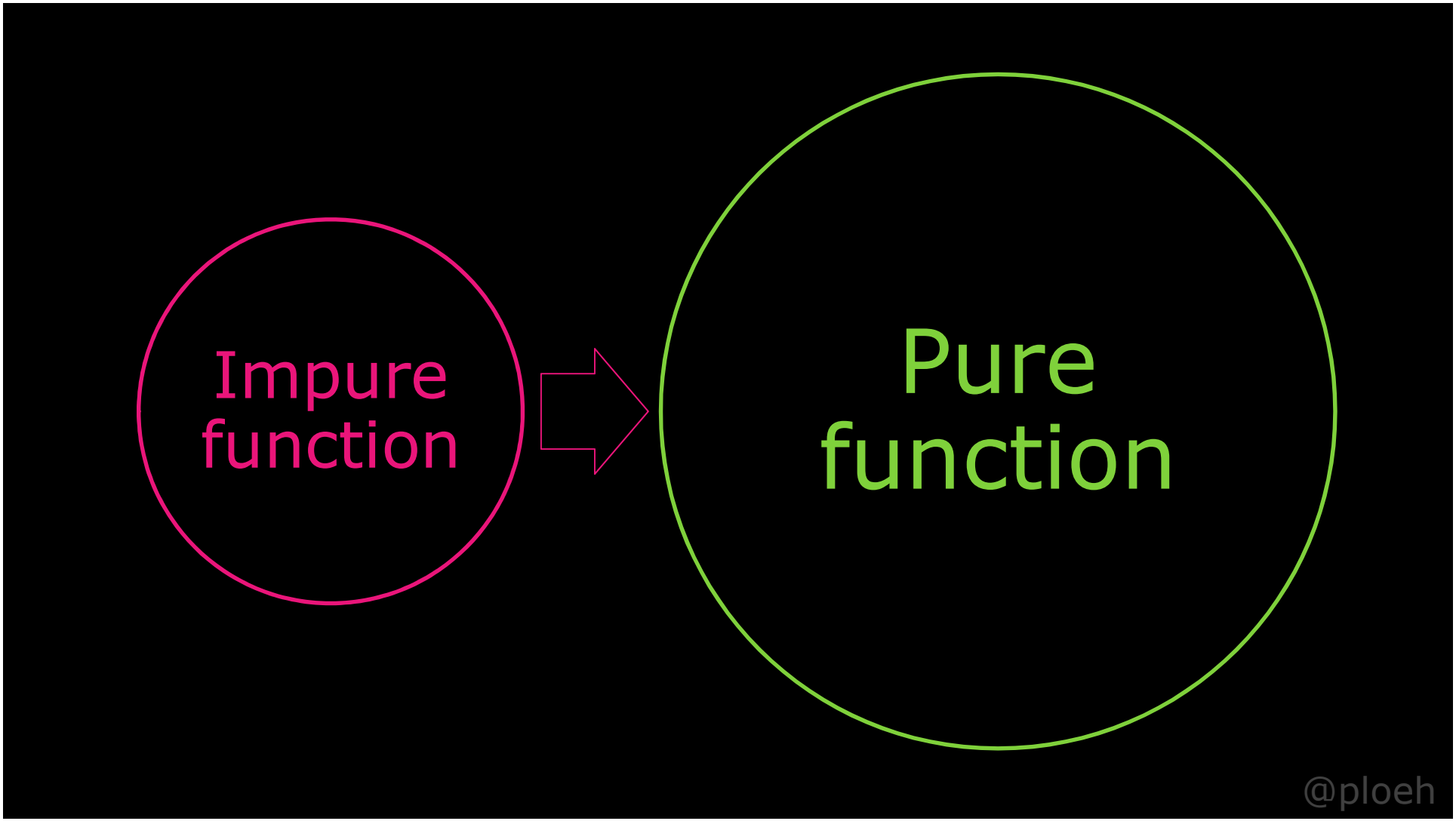

@ploeh


Ports and adapters
Data and behaviour
Testability
@ploeh

PORTS AND ADAPTERS


@ploeh


Same return value for same input
No side-effects
Pure function
@ploeh


Impure
function
Pure
function
@ploeh

Restaurant Reservation

Make a reservation	
Date	
Name	
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2
Name	
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	20
Name	
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	201
Name	
Email	
Quantity	
	Submit

@ploeh

Restaurant Reservation

Make a reservation	
Date	2016
Name	
Email	
Quantity	
	Submit

@ploeh

Restaurant Reservation

Make a reservation	
Date	2016-
Name	
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-1
Name	
Email	
Quantity	
	Submit

@ploeh

Restaurant Reservation

Make a reservation	
Date	2016-12
Name	
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-
Name	
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-2
Name	
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	M
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Ma
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mar
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark S
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Se
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark See
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seem
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seema
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seeman
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	ma
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mar
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@p
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@pl
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@plo
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@ploe
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@ploeh
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@ploeh.
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@ploeh.d
Quantity	
	Submit

@ploeh

Restaurant Reservation

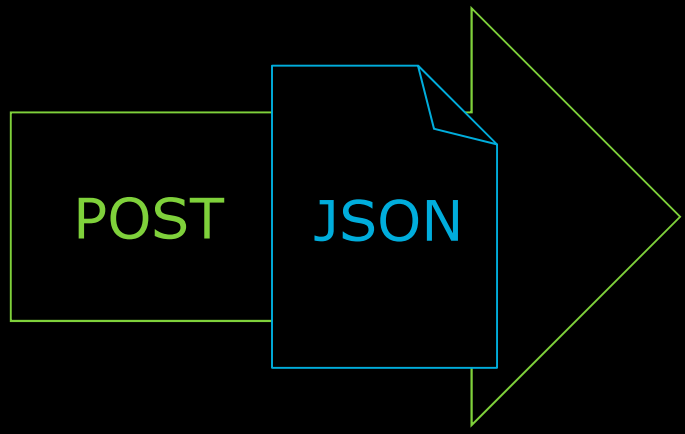
Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@ploeh.dk
Quantity	
	Submit

Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@ploeh.dk
Quantity	4
	Submit

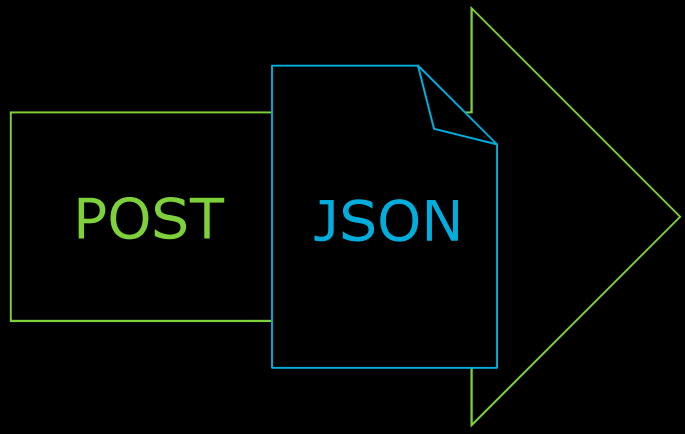
Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@ploeh.dk
Quantity	4
	Submit

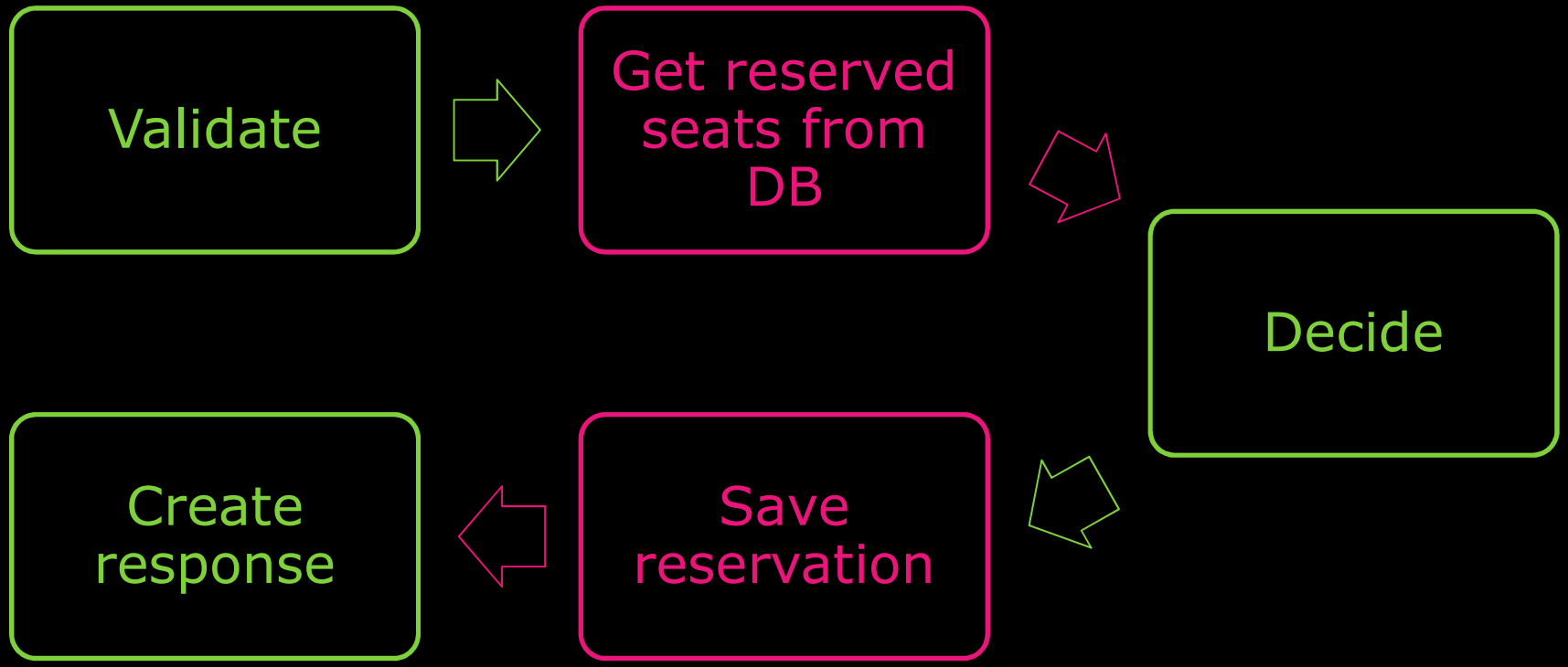


Restaurant Reservation

Make a reservation	
Date	2016-12-29
Name	Mark Seemann
Email	mark@ploeh.dk
Quantity	4
	Submit



Restaurant Reservation



@ploeh

```haskell
validateReservation :: ReservationDto -> Either Error Reservation
validateReservation r =
  case parseDate (rDate r) of
    Just d ->
      Right Reservation
        { date = d
        , name = rName r
        , email = rEmail r
        , quantity = rQuantity r }
    Nothing -> Left (ValidationError "Invalid date.")
```

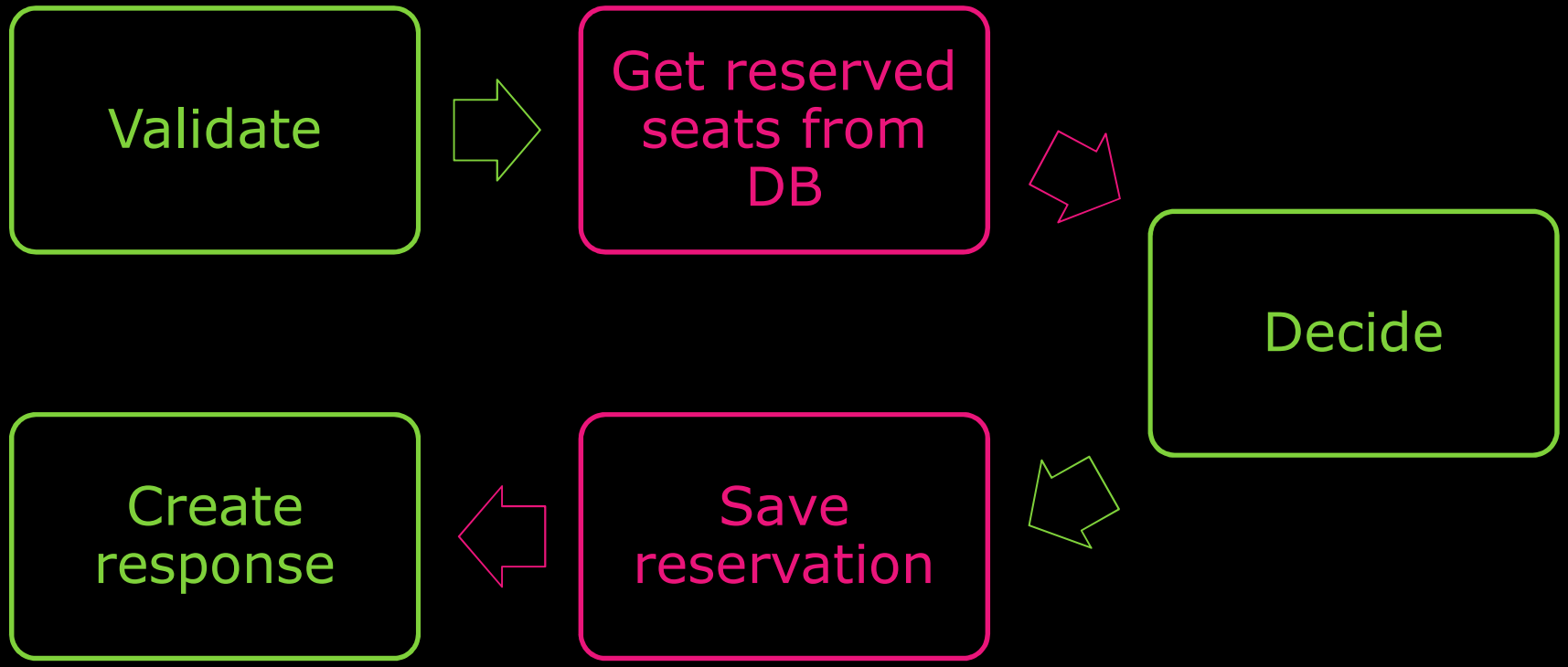
```
getReservedSeatsFromDB :: ConnectionString -> ZonedTime -> IO Int
```

```haskell
checkCapacity :: Int -> Int -> Reservation -> Either Error Reservation
checkCapacity capacity reservedSeats reservation =
    if capacity < quantity reservation + reservedSeats
    then Left CapacityExceeded
    else Right reservation
```

```
saveReservation :: ConnectionString -> Reservation -> IO ()
```

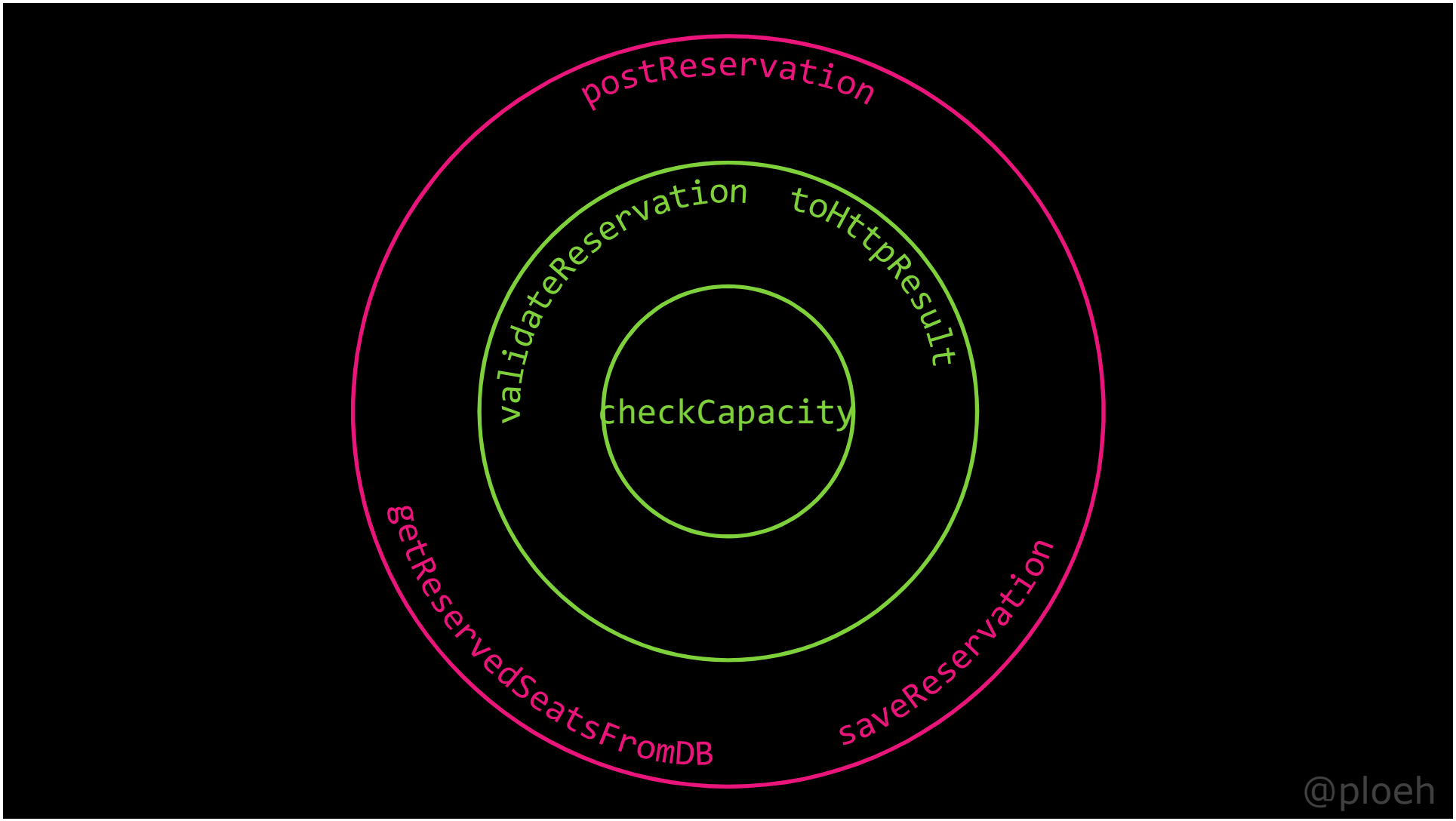
```haskell
toHttpResult :: Either Error () -> HttpResult ()
toHttpResult (Left (ValidationError msg)) = BadRequest msg
toHttpResult (Left CapacityExceeded) = StatusCode Forbidden
toHttpResult (Right ()) = OK ()
```

Restaurant Booking



@ploeh

```haskell
postReservation :: ReservationDto -> IO (HttpResult ())
postReservation candidate = fmap toHttpResult $ runEitherT $ do
  r <- hoistEither $ validateReservation candidate
  i <- liftIO $ getReservedSeatsFromDB connStr $ date r
  hoistEither $ checkCapacity 10 i r
  >>= liftIO . saveReservation connStr
```


postReservation
validateReservation
toHttpResult
checkCapacity
getReservedSeatsFromDB
saveReservation
@ploeh

```
let imp candidate = either {
    let! r = Validate.reservation candidate
    let i = SqlGateway.getReservedSeats connectionString r.Date
    let! r = Capacity.check 10 i r
    return SqlGateway.saveReservation connectionString r }
new ReservationsController(imp) :> _
```

DATA AND BEHAVIOUR

```
public class User
{
    public int Id { get; }

    public string UserName { get; }

    public static User CreateNew(string userName)

    public static User Read(int id)

    public void Update()

    public void Delete()
}
```

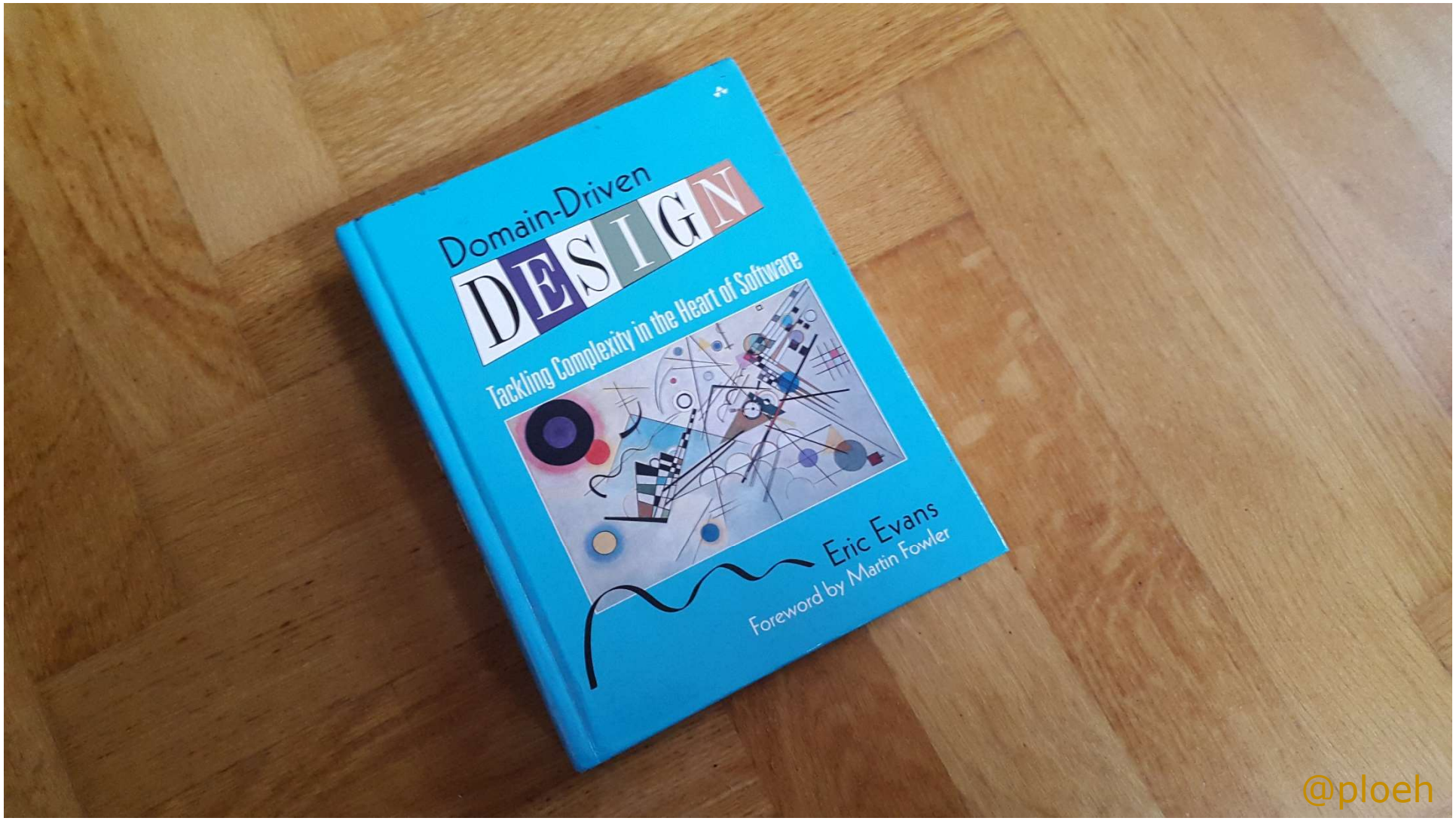
C♯

```
public class User
{
    public int Id { get; }

    public string UserName { get; }

    public static User CreateNew(string userName)

    public static User Read(int id)

    public void Update()

    public void Delete()

}
```

C#

```
public class User
{
    public int Id { get; }

    public string UserName { get; }

    public static User CreateNew(string userName)

    public static User Read(int id)

    public void Update()

    public void Delete()

    public void SendEmail(Email message)
}
```

C♯


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler
@ploeh


Entities
Value Objects
Services
@ploeh


Data
Services
@ploeh

```
public class User
{
    public User(int id, string userName)

    public int Id { get }

    public string UserName { get }
}
```

C♯

@ploeh

```
public class SqlUserRepository : IUserRepository
{
    public User CreateNew(string userName)

    public User Read(int id)

    public void Update(User user)

    public void Delete(User user)
}
```

C♯

```
public class EmailSender : ISendEmail
{
    public void SendEmailTo(User user, Email message)
}
```

C♯

@ploeh


Anaemic Model
@ploeh

@ploeh

Anaemic D Model

@ploeh

Anaemic Do Model

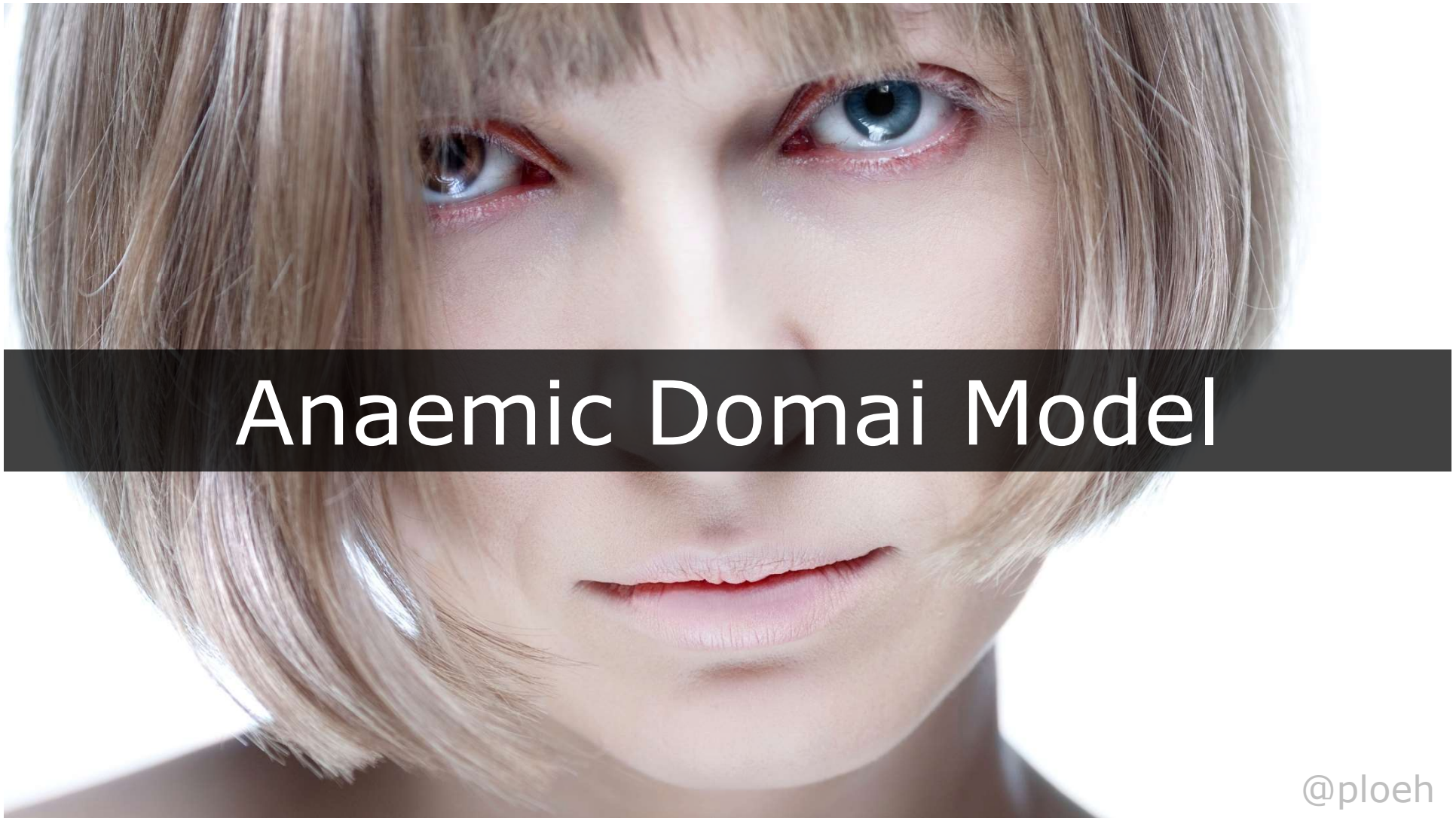
@ploeh

Anaemic Dom Model

@ploeh

Anaemic Doma Model

@ploeh



Anaemic Domai Model

@ploeh


Anaemic Domain Model
@ploeh


Entities
Value Objects
Services
@ploeh

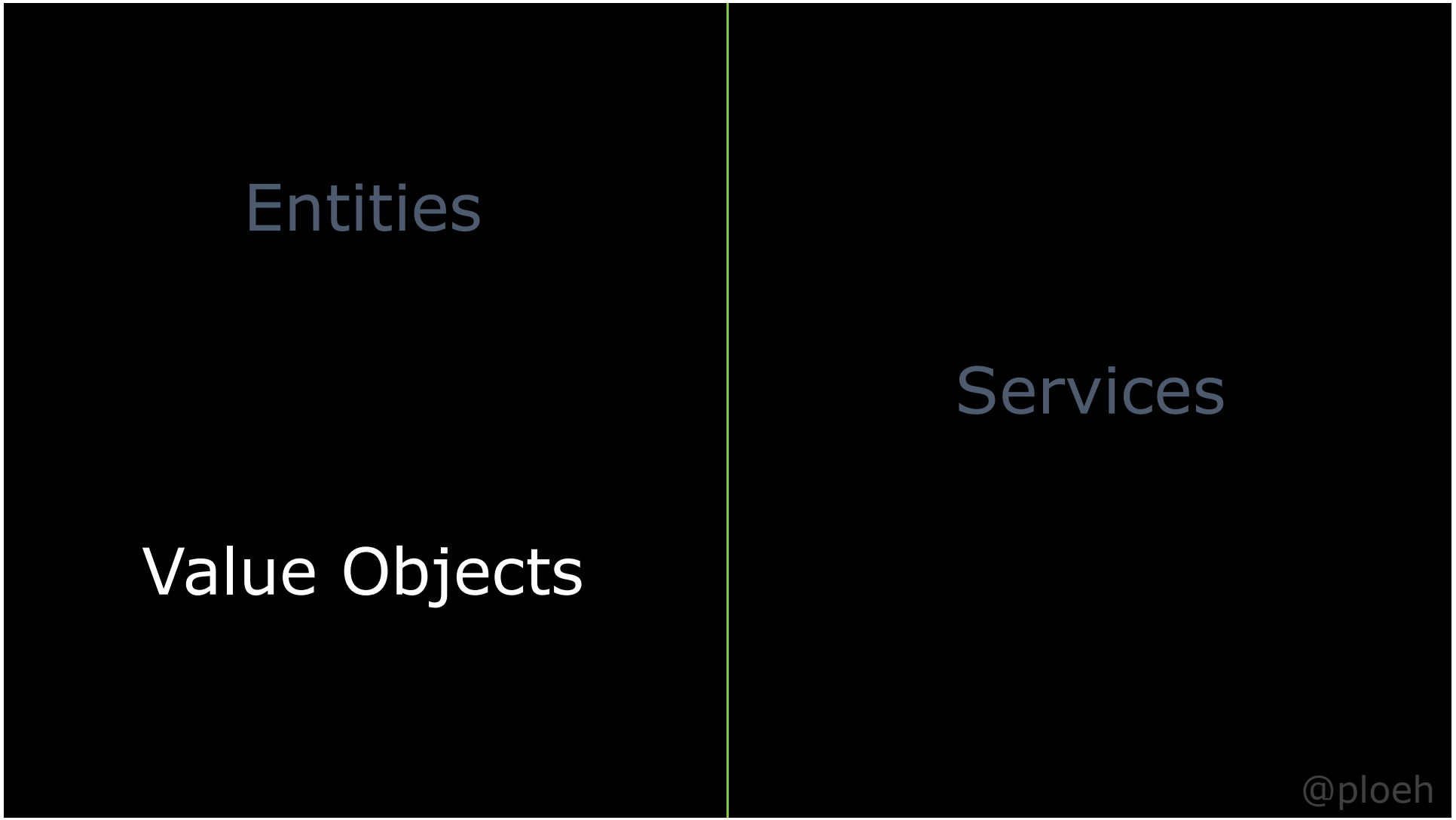

Data
Services
@ploeh


Data
Functions
@ploeh

```
type User = { Id : int; UserName : string }
```

```
module Persistence =
    // string -> User
    let createNew userName = // ..

    // int -> User
    let read id = // ..

    // User -> unit
    let update user = // ..

    // User -> unit
    let delete user = // ..
```

```
module Mail =
    // User -> Email -> unit
    let sendEmailTo user message = // ..

    // ..
```


Entities
Value Objects
Services
@ploeh

```
public sealed class Money
{
    public Money(decimal amount, string currency)

    public decimal Amount { get }

    public string Currency { get }

    public override bool Equals(object obj)

    public override int GetHashCode()
}
```

C♯

```
type Money = { Amount : decimal; Currency : string }
```

```
> { Amount = 1m; Currency = "EUR" } =
  { Amount = 1m; Currency = "EUR" };;
val it : bool = true
> { Amount = 2m; Currency = "EUR" } =
  { Amount = 1m; Currency = "EUR" };;
val it : bool = false
> { Amount = 1m; Currency = "EUR" } =
  { Amount = 1m; Currency = "GBP" };;
val it : bool = false
```

TESTABILITY

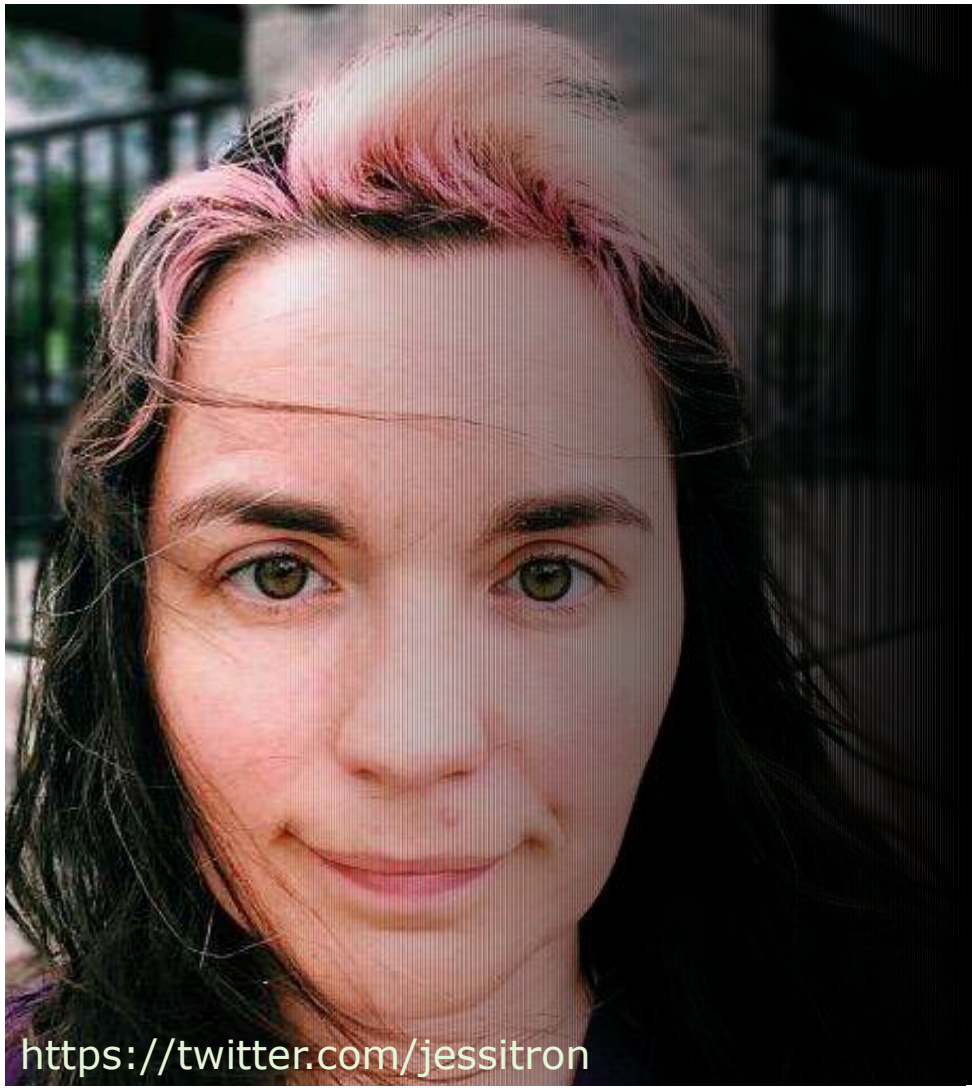

Ascari
Sabelt
Test-induced
damage
https://www.flickr.com/photos/davehamster/8667991007/
@ploeh

```
public class CapacityChecker : ICapacityChecker
{
    private readonly int capacity;
    private readonly IReservationRepository repo;

    public CapacityChecker(int capacity, IReservationRepository repo)
    {
        this.capacity = capacity;
        this.repo = repo;
    }

    public bool HasCapacity(Reservation reservation)
    {
        var reserved = this.repo.GetReservedSeats(reservation.Date);
        return this.capacity < reservation.Quantity + reserved;
    }
}
```

C#

```
let check capacity getReservedSeats reservation =
    let reservedSeats = getReservedSeats reservation.Date
    capacity < reservation.Quantity + reservedSeats
```

```fsharp
[<Fact>]
let ``check returns right result at no prior reservations`` () =
    let capacity = 10
    let getReservedSeats _ = 0
    let reservation = {
        Date =
            DateTimeOffset(DateTime(2014, 8, 10), TimeSpan.FromHours 2.)
        Name = "Mark Seemann"
        Email = "mark@ploeh.dk"
        Quantity = 4 }

    let actual = Capacity.check capacity getReservedSeats reservation

    test <@ actual @>
```

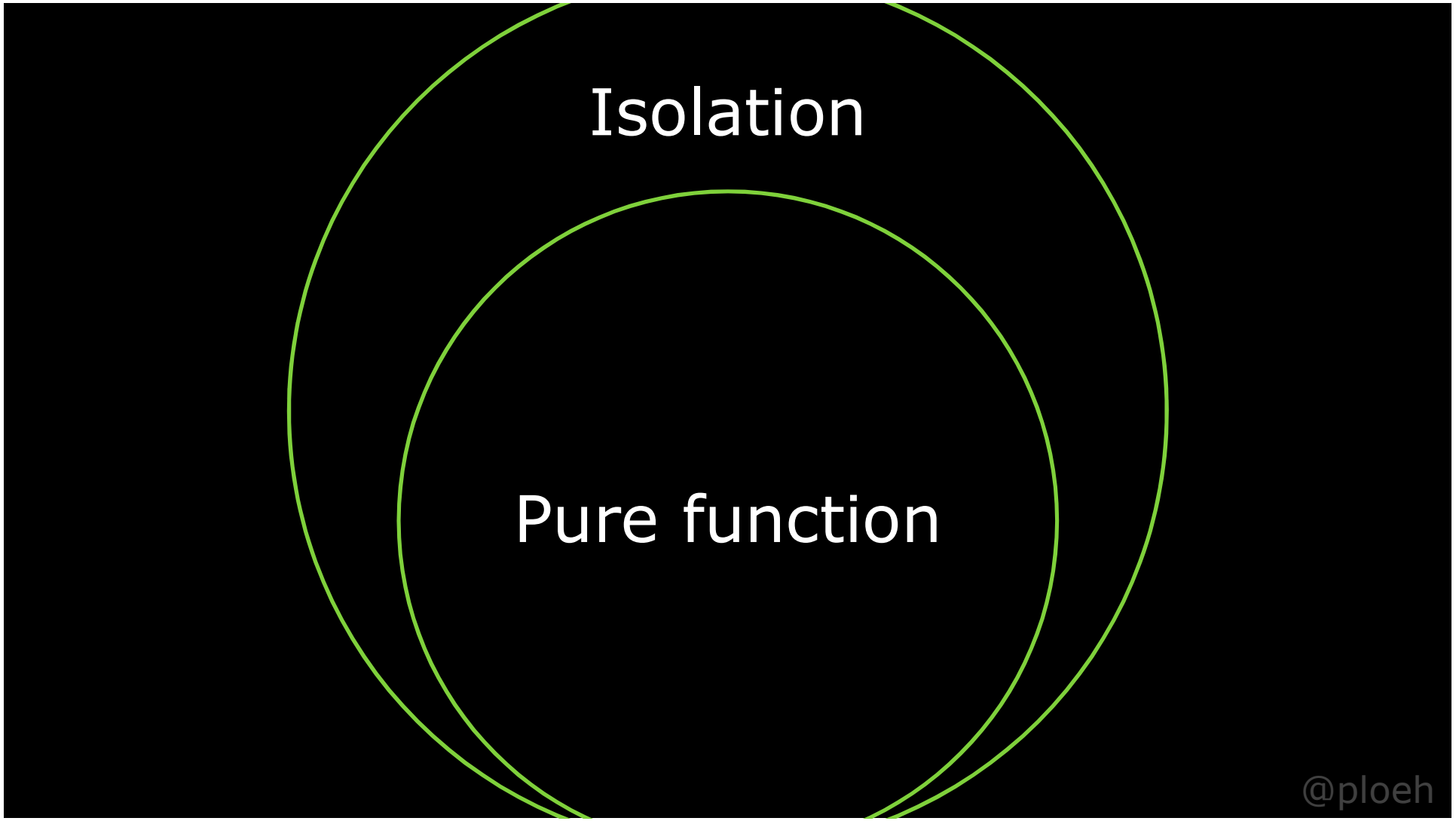


Jessica Kerr

Isolation

When the only information a function has about the external world is passed into it via arguments.

@ploeh

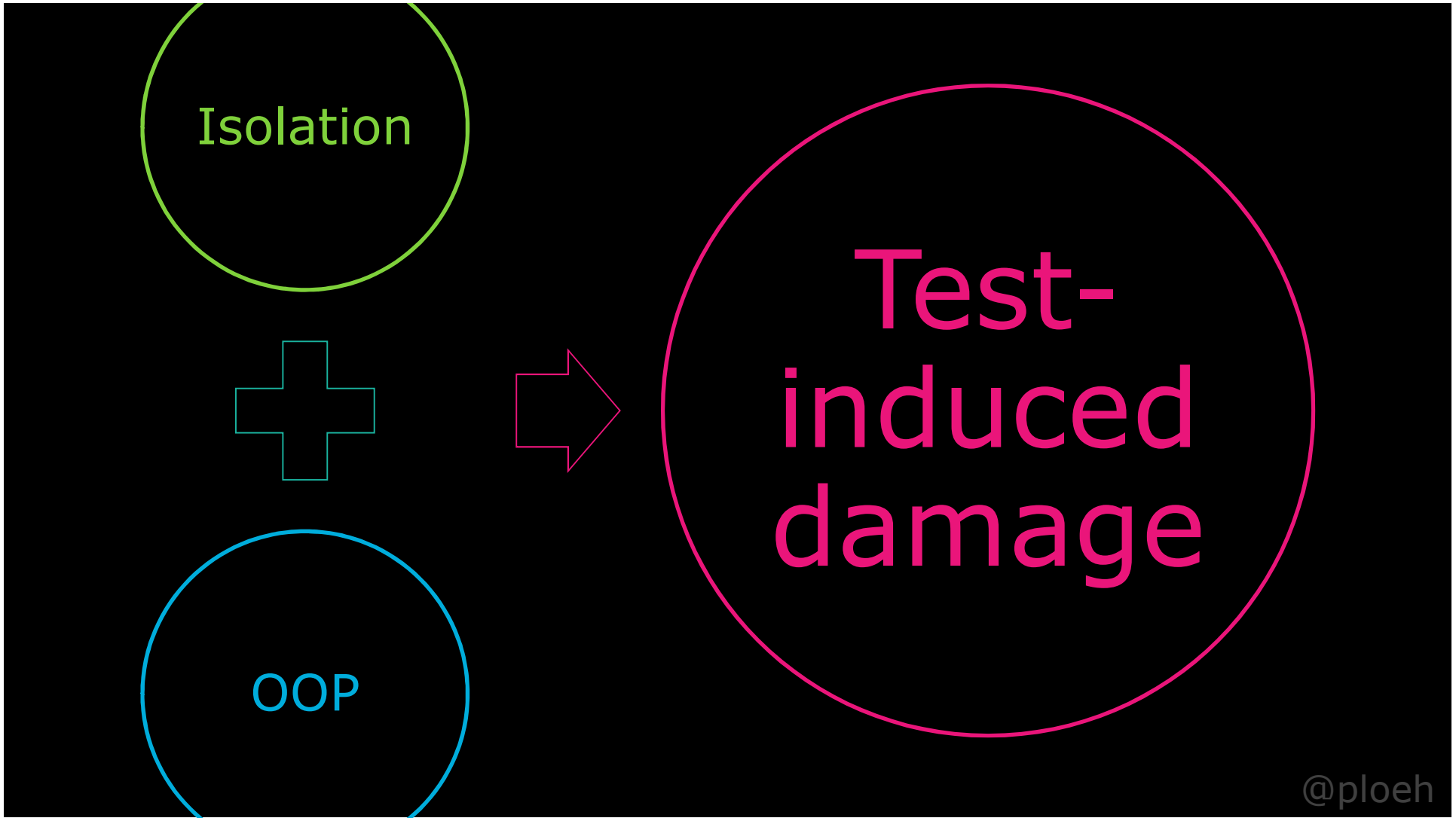

Isolation
Pure function
@ploeh

Unit Test

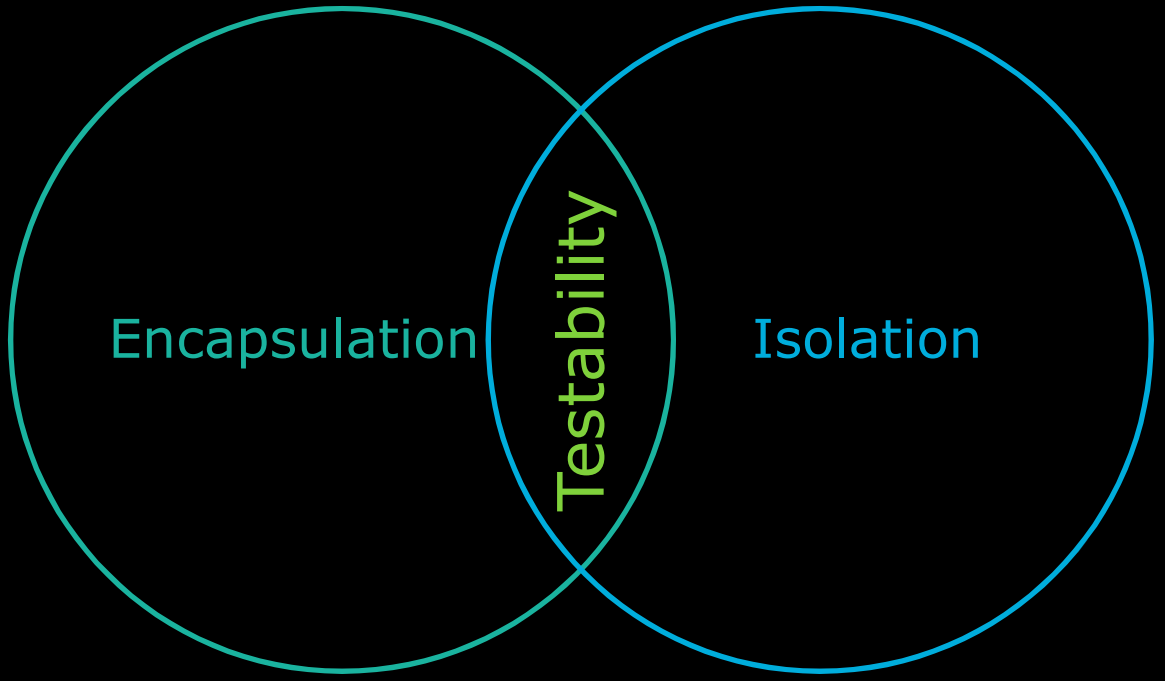
A unit test is an automated test that tests a unit in isolation from its dependencies.

Unit Test

A unit test is an automated test that tests a unit in isolation from its dependencies.

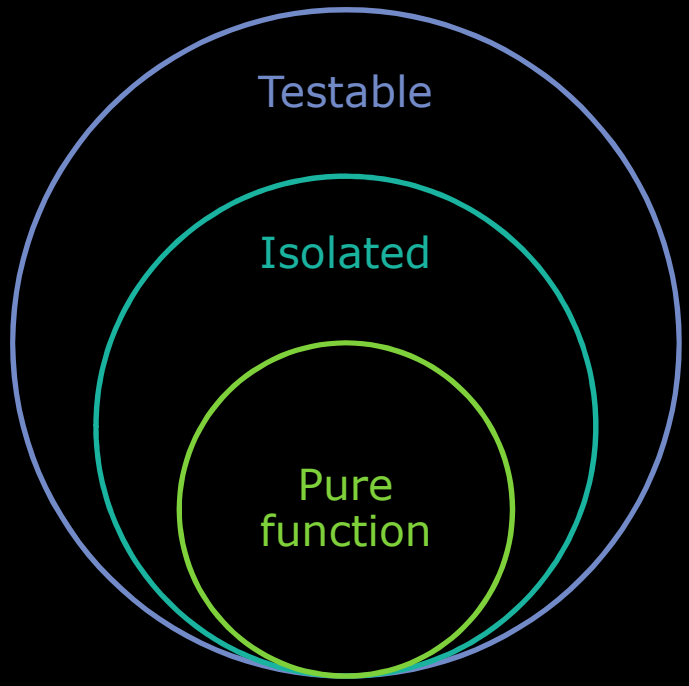

Isolation
OOP
Test-
induced
damage
@ploeh

Object-Oriented Programming

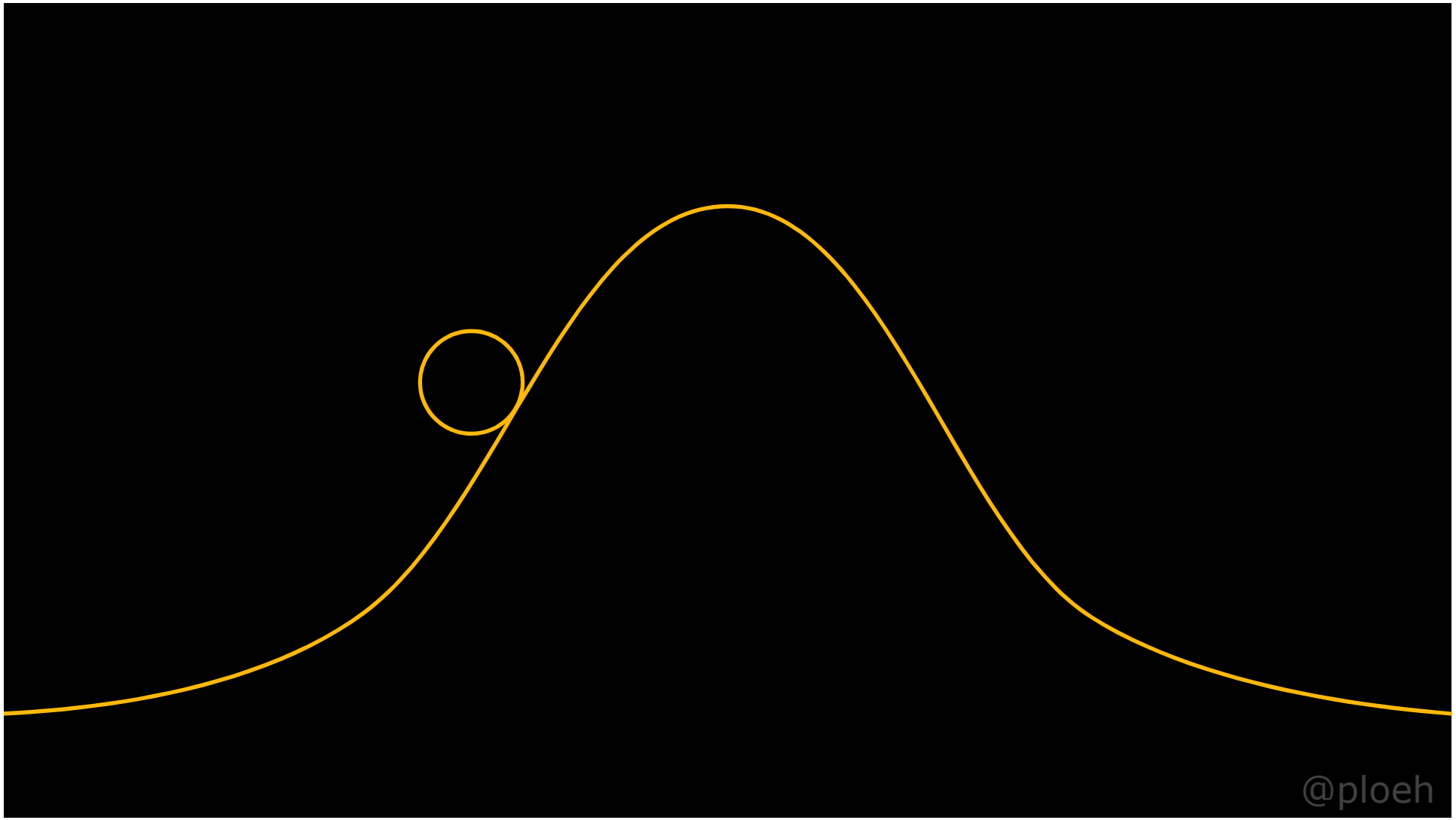


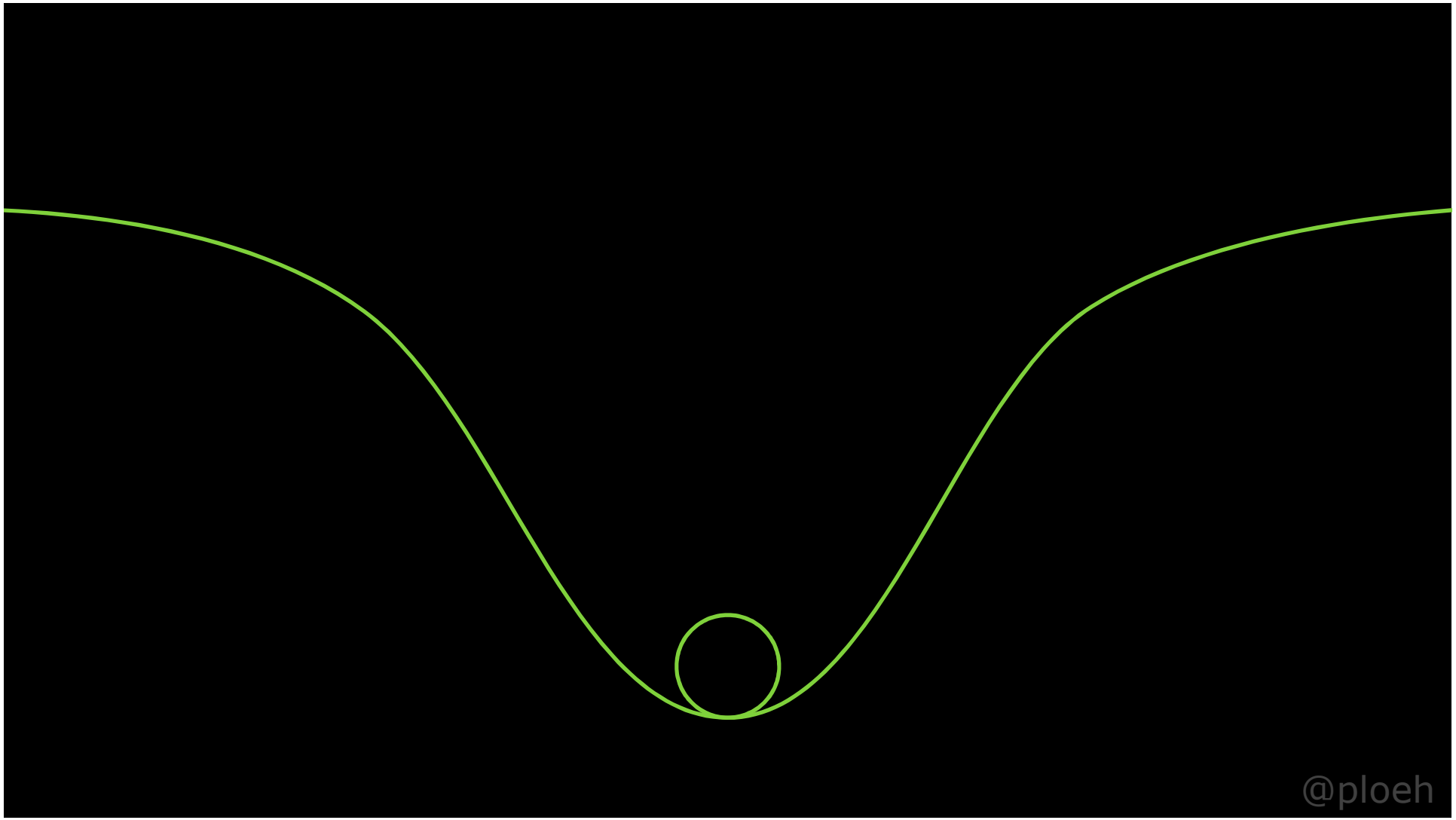
@ploeh

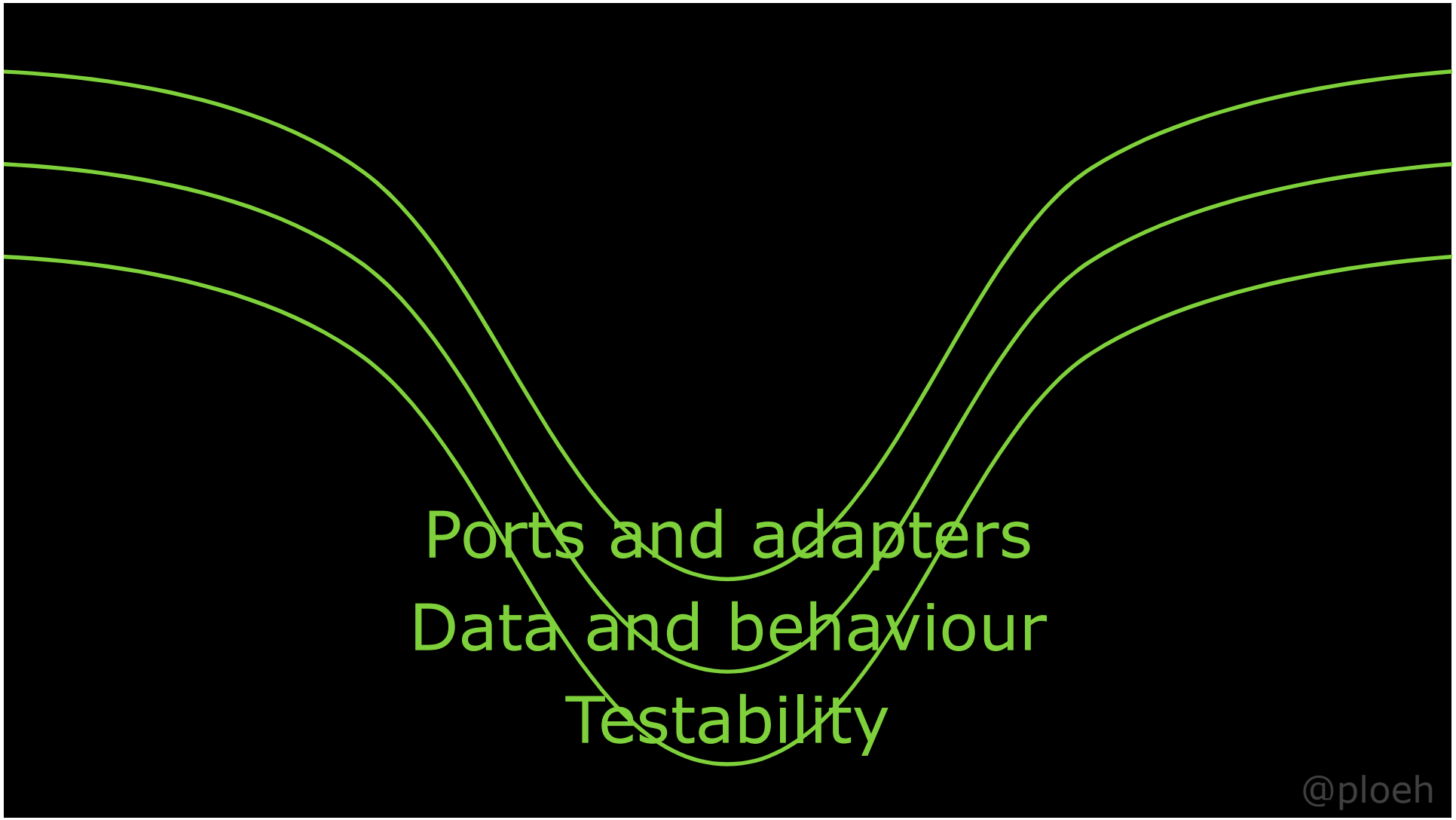
Functional Programming



@ploeh


@ploeh


@ploeh


Ports and adapters
Data and behaviour
Testability
@ploeh

Mark Seemann

http://blog.ploeh.dk

http://bit.ly/ploehralsight

@ploeh