Seeing Arrows Below the Code

Narek Asadorian
flatMap(Oslo)
May 9, 2019

I’m Narek.

Senior Software Engineer

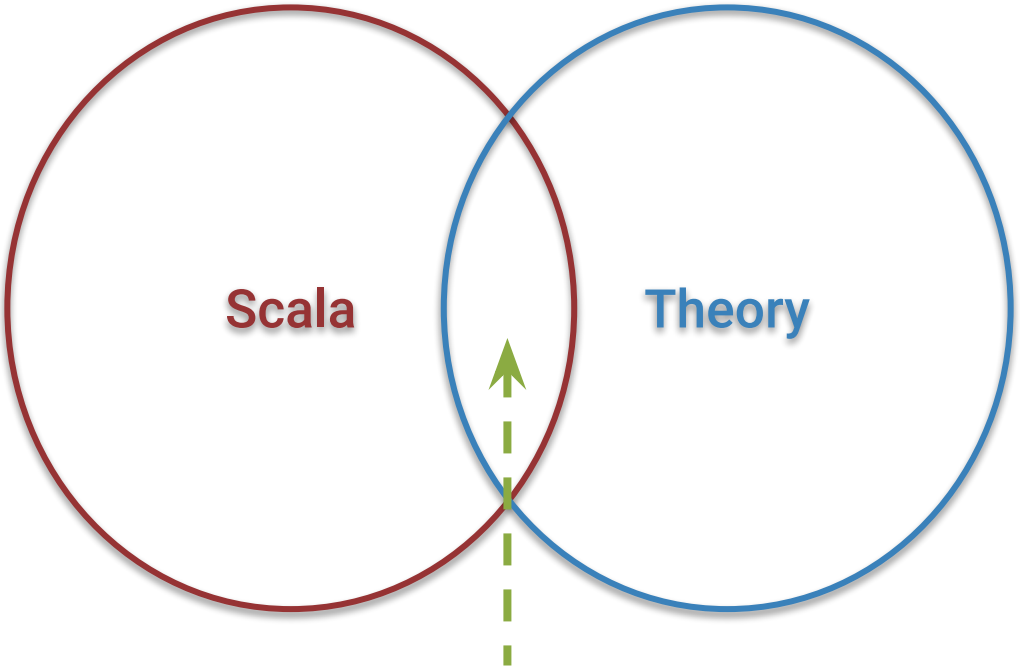
Scala & FP Advocate

@portal_narlish

About Me:

- Writing Scala for 2+ years
- Working on a R&D team improving the sales process with data
- Data engineering & machine learning

Motivation



Pure functional programming lies here…

We want to understand the (dis)connection between ***underlying structures*** *and* ***implementation details****.*

What's this talk about?

- ➔ Art of Abstraction & Composition
  - ◆ Learning to blur out details in software
- ➔ Ditching Objects for Objects
  - ◆ Looking past Scala's implementation
- ➔ Arrows, Arrows Everywhere
  - ◆ Underlying structures in code examples

What to walk away with?

- ➔ Better understanding of categorical terminology used in Cats
- ➔ Importance of composition in software design
  - ◆ And the math behind it
- ➔ Thinking and programming in terms of arrows, not values

Some Caveats

No Math PhD

Informal, elementary category theory **exclusively** as it applies to FP.

Int. Scala

Assuming familiarity with syntax, language features and Cats FP library.

My Own Views

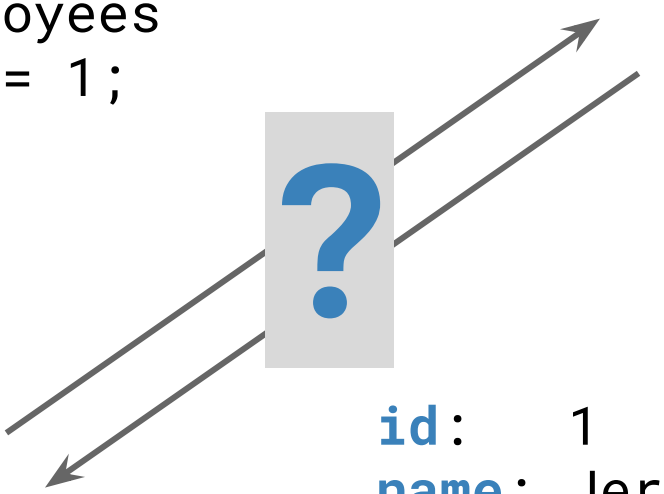
Opinions and statements are mine alone, and not my employer's.

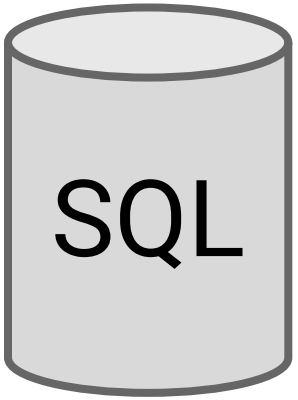
Art of Abstraction & Composition

*Dropping out the **details***

Database Access

```
SELECT *
FROM employees
WHERE id = 1;
```



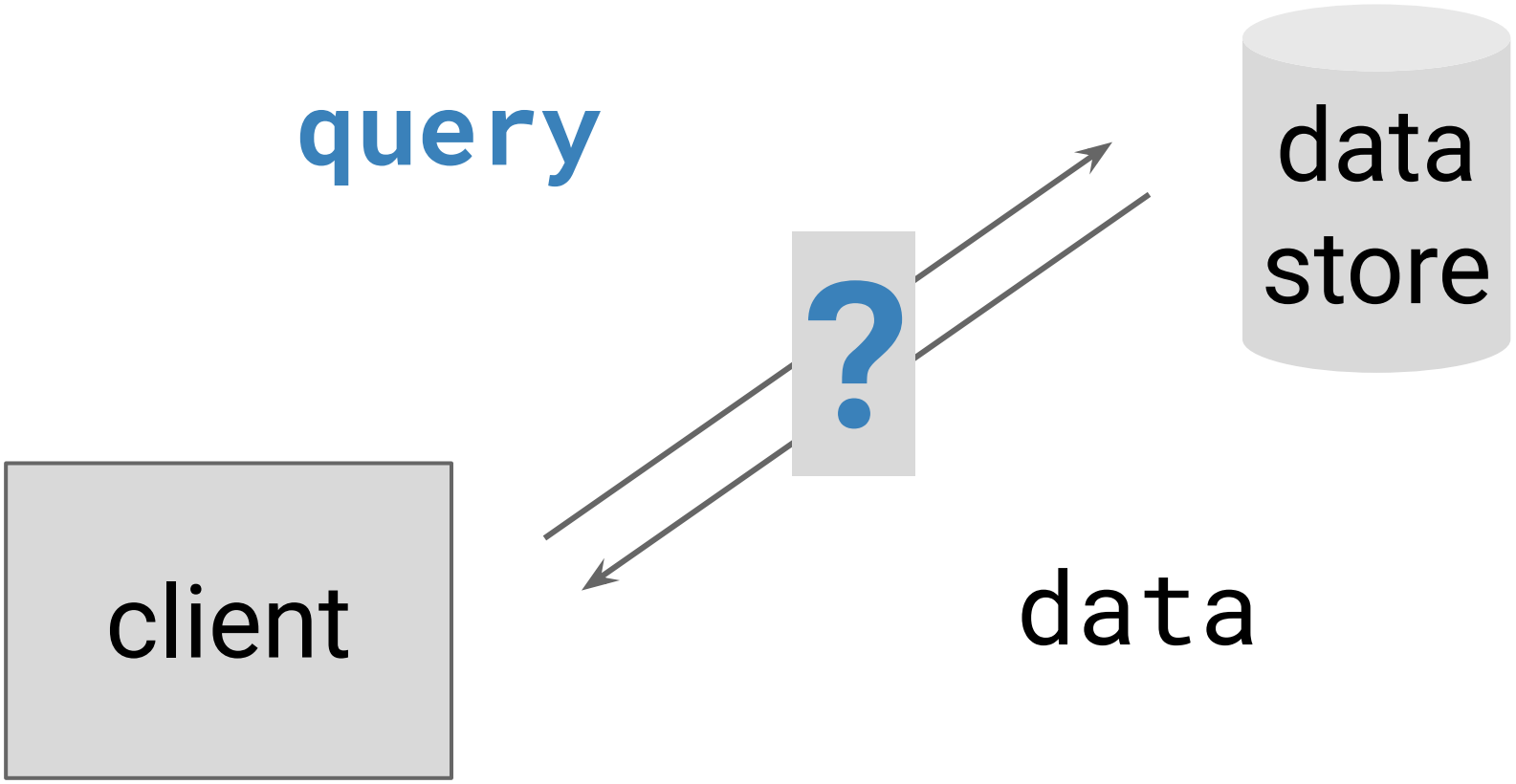



```
id:   1
name: Jerry
age:  53
role: Comedian
```

Blur the Lines

25%

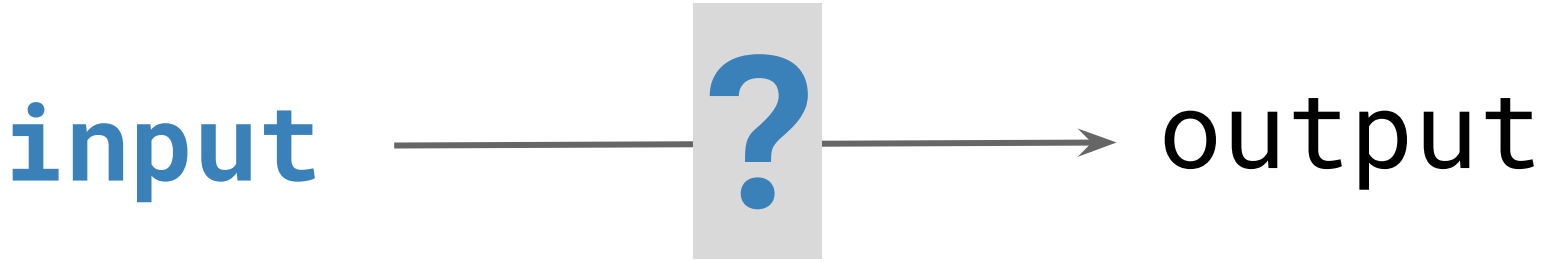
Database Access



Blur the Lines

50%

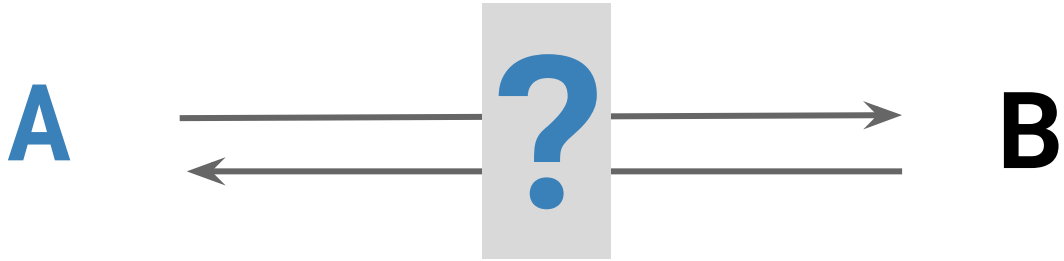
Database Access



Blur the Lines

75%

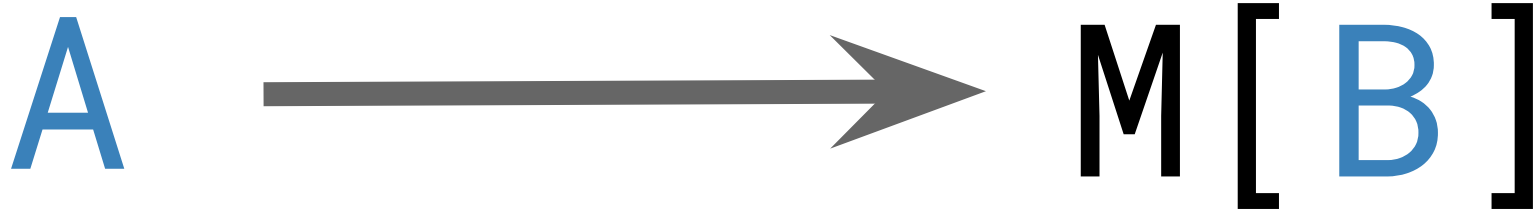
Database Access



Blur the Lines

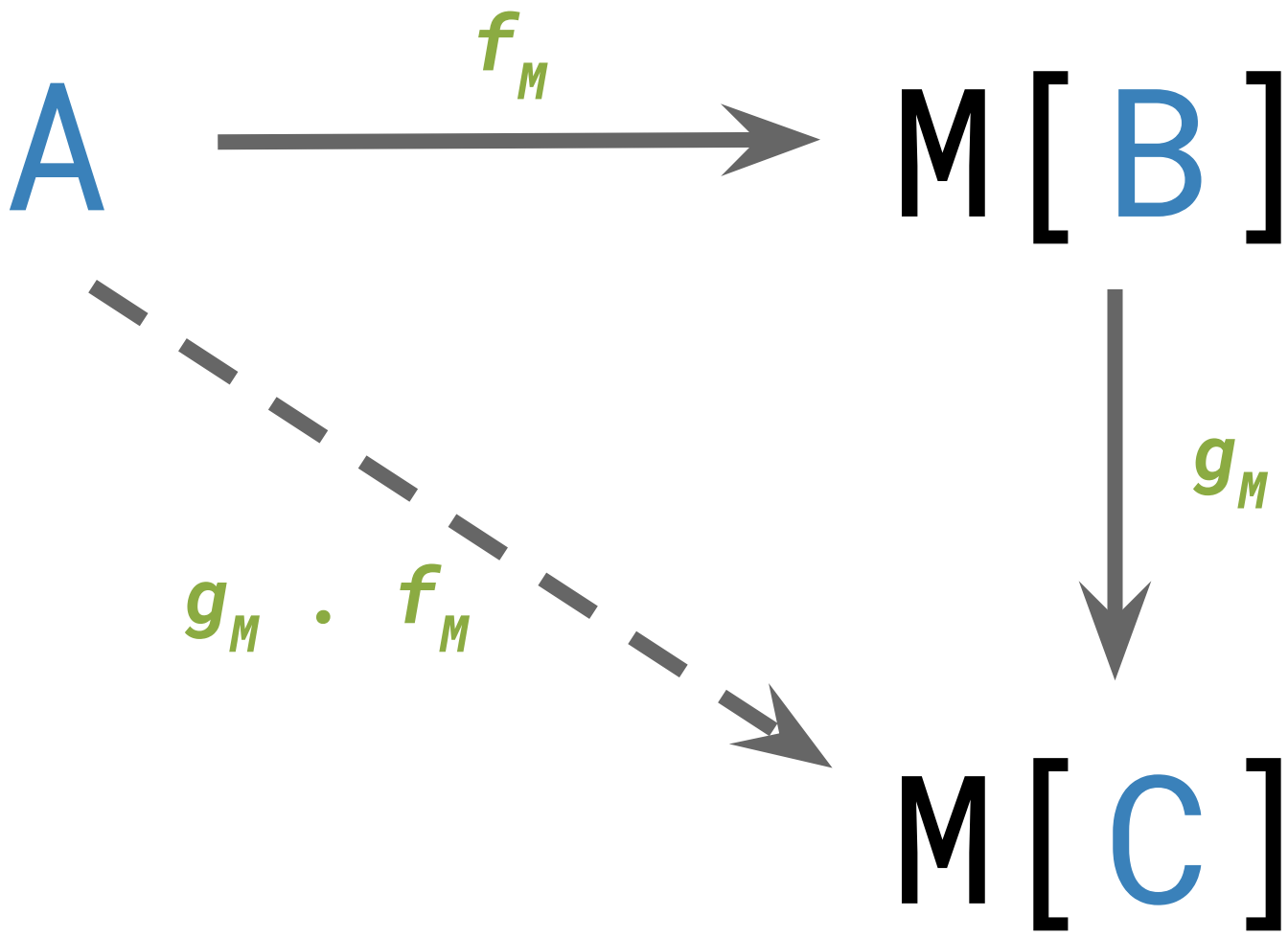
95%

It’s just an arrow...



A Kleisli Arrow.

Kleisli Category



Blur the Lines

100%

Java™

OOP

Abstraction & Composition

Abstraction

Blurring out details from a system exposes essential objects and relationships.

Composition

When individual components of a system must fit together, generic interfaces emerge.

Abstraction leads to **composition.**

Composition leads to **abstraction.**

Ditching Objects for Objects

Scala as a category

*Objects in a category are not the same as objects in programming… you should **object** to that kind of object!*

- *Prof. Philip Wadler*

What is category theory?

- Branch of abstract mathematics concerned with **relationships** and **composition**
- **Abstractive toolkit** for drawing connections between things and building models
- **Foundational** part of modern functional programming

Why should you care?

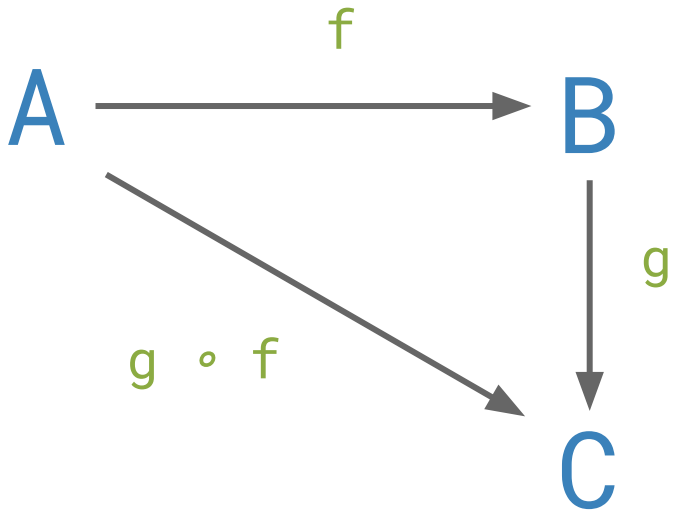
Does a stand-up comedian need to study philosophy? 🤔

Elementary category theory goes a long way in functional programming…

- *Understanding of advanced tools*
  - *cats, monocle, doobie, droste*
- *Leveraging abstractions to write composable code*
- *Riding the Hascalator*

Category

- Objects
- Morphisms
- Identity
- Composition



And everywhere that functions are, there are categories.
Indeed, the subject might better have been called abstract function theory,
or, perhaps even better: ***archery****.*

- ***Prof. Steve Awodey***

The Category **Scal**

- Scala's types are objects
  - Not JVM objects
- Functions are morphisms (arrows)
- Identity and composition of functions

```
def identity[A]: A => A

def compose[A, B, C]: (B => C) => (A => B) => (A => C)
```

Category Typeclass?

```
trait Category[F[_,_]] {

  def identity[A]: F[A,A]

  def compose[A,B,C]:
    F[B,C] => F[A,B] => F[A,C]

}
```

- Difference in encoding, but why?
  - Category theory: diagrams
  - Programming: arrows!

Encoding vs Theory

Types ~ Objects

Functions ~ Arrows

Typeclasses ~ Cat. Structure

Implicit Traits ~ Theorems

Tests ~ Proofs

There is a
non-trivial distance
between the pure mathematical construct and its language encoding...

...and often, noisy "language features" get in the way.

Scala is very guilty of this!

If we focus on the **arrows**, we can cut through some of the "language noise".

Arrows, Arrows Everywhere

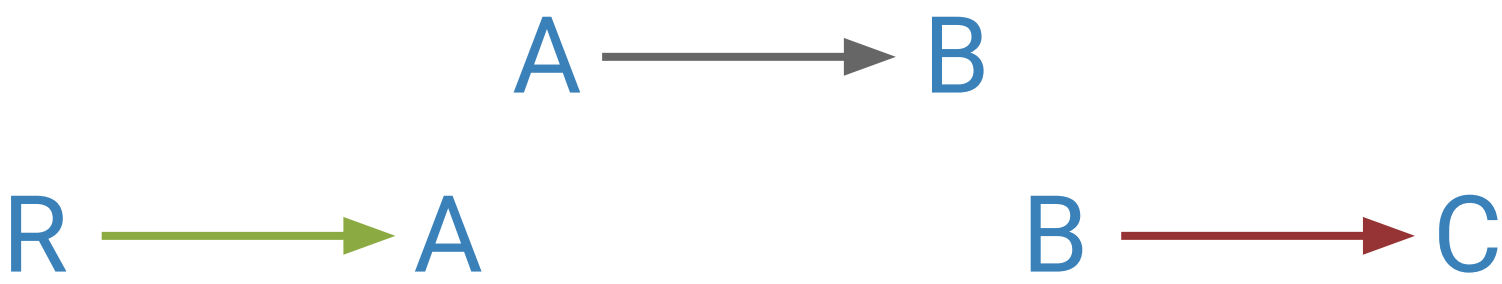
How to see them

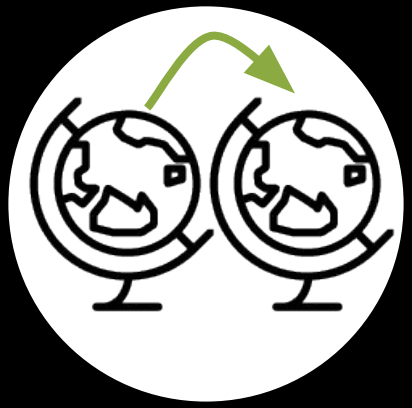
Function

The simplest arrow

Functions Compose

- Functions are arrows
- Scala composes functions
  - `Function1 - compose, andThen`
- More exotic...
  - Cats instances for `Function1`





Functor

Structure preserving
transformation

Functor Example 1

```
def getPerson(s: String): Option[Person]

def personFeature(p: Person): PFeature
```

```
// A simple `map` operation
def nameToFeature(s: String): Option[PFeature] =
  getPerson(s) map personFeature
```

What's the big deal?

- The `map` operation is often taught as "mapping a function ***over*** *a container*"

- In categorical terms, Functor is not a container!
  - **It's really just an arrow**
  - **Composable**

Defining Functor

Cats Docs:

Functor is a type class that abstracts over type constructors that can be map'ed over.

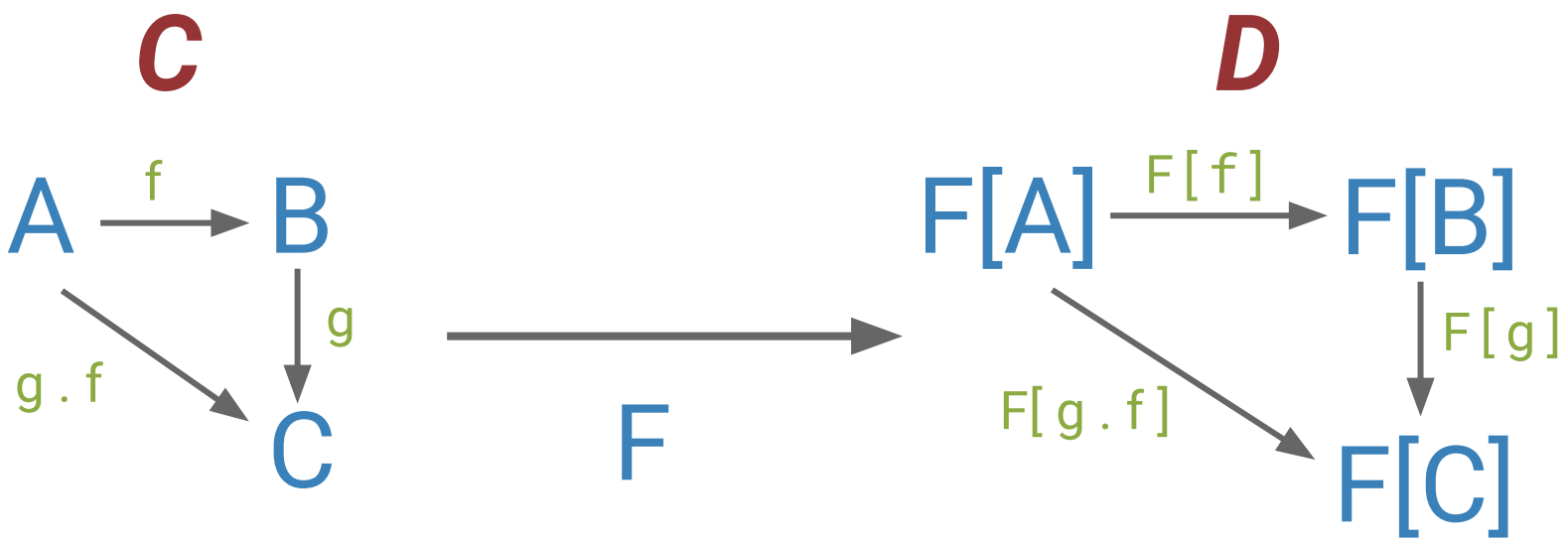
Textbook:

A functor ***F***: $C \rightarrow D$ between categories C and D is a mapping of objects to objects and arrows to arrows.

Functor Typeclass

```
trait Functor[F[_]] {

 def map[A,B](fa: F[A])(f: A => B): F[B]

}
```

Functor



F : ***C*** → ***D*** is a functor preserving domains, codomains, identity arrows and composition.

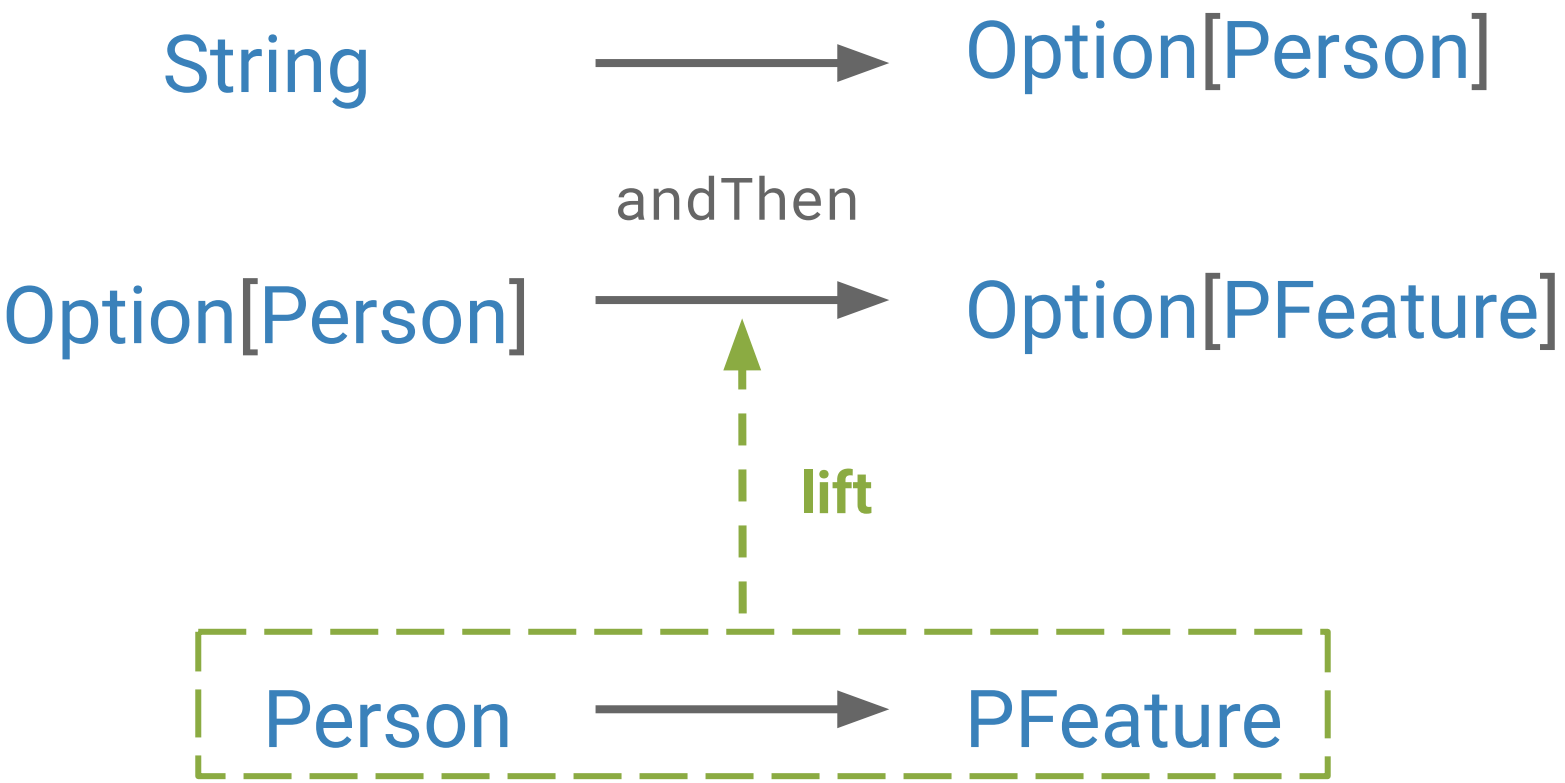
Functor w/ `lift`

```
trait Functor[F[_]] {

 def map[A,B](fa: F[A])(f: A => B): F[B]

 def lift[A,B](f: A => B): F[A] => F[B]=
   fa => map(fa)(f)

}
```

Functor Example 1

```
/**
 * Alternative formulation in “arrow” style
 */
val nameToFeature2: String => Option[PFeature] =
  getPerson andThen
    (Functor[Option] lift personFeature)
```

Lift Brings Composition



Is this necessary?

- No, but it exposes an important idea:
  - We don't always need to think about acting on values, the arrows are usually enough to get the job done.
- **Functional programming really is a job of connecting arrows together...**

Functor Example 2

```
/* Pulls many people from the "database" */
def getPeople(
  names: List[String]
): List[Option[Person]] = names map getPerson
```

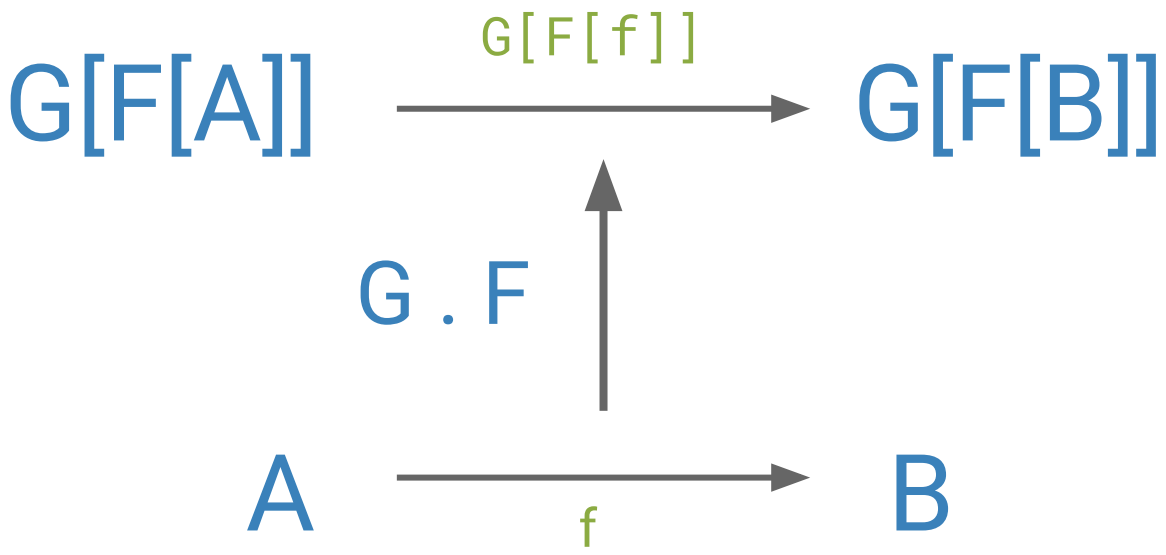
```
/**
 * Can we implement this with our primitives?
 */
val getAndFeaturizePeople:
  List[String] => List[Option[PFeature]] = ???
```

Functor Example 2

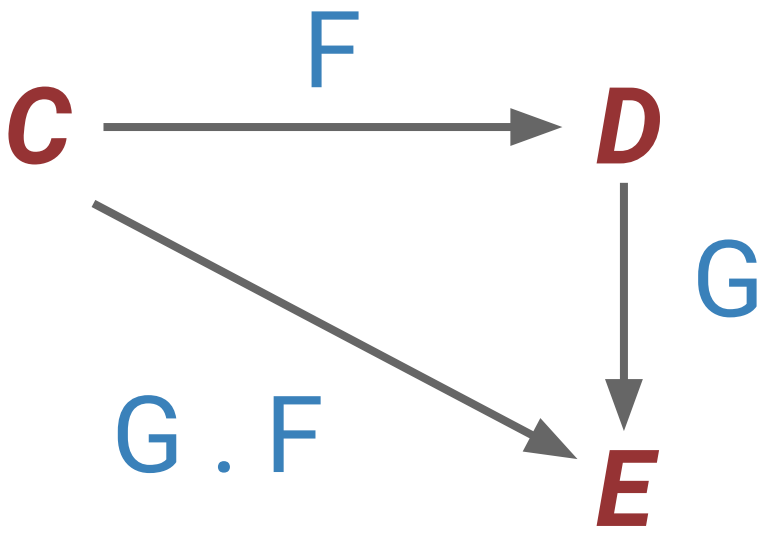
```
/* Extract features for every Person entry */
val peopleFeatures:
 List[Option[Person]] => List[Option[PFeature]]=
  (Functor[List] compose Functor[Option])
    lift personFeature

/**
 * Contramapping aka [[andThen]]
 */
val getAndFeaturizePeople:
 List[String] => List[Option[PFeature]] =
   getPeople andThen peopleFeatures
```

2 Functors

```
Option[?]        is F
List[?]          is G
```



Blur the lines...



In the category of categories, functors compose too… they're just arrows!

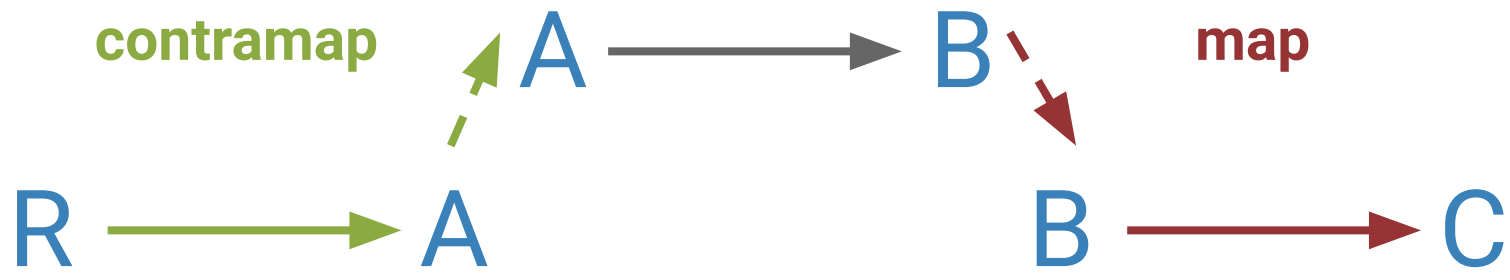
Abstraction leads to **composition.**

Composition leads to **abstraction.**

Instead of writing nested code, we reach for composition to simplify things.

Function is a Functor

- How? Cats provides many instances...
  - `Function[A,?] ~ F[?]`
    - Covariant
      - andThen ~ map
  - `Function[?,B] ~ F[?]`
    - Contravariant
      - compose ~ contramap



Functor Example 3

```
/**
 * Create a data pipeline with chained ops
 */
val featurePipeline:
  List[String] => List[PFeature] =
    peopleFeatures                  // A => B
      .dimap(getPeople)             // R => A
      (_.mapFilter(x => x))         // B => C
```

Kleisli

Sequencing computational effects

Kleisli Example 1

```
case class ServerConf(
  host: String,
  ip: IP,
  serverType: ServerType)

type ServerType = Either[App, MicroService]
```

```
case class App(
  name: String, appConf: Map[String, String])

case class MicroService(
  name: String, locale: String)
```

Kleisli Example 1

- Let's build a config extractor for `region`
  - No guarantees about fields being there
  - Multiple levels of Option
  - Different substructure App/Microservice

locale = "NA-WEST-2-CORE"
locale = "BANK_SERVER_EU"

```
val extractRegion:
  ServerConf => Option[String] = ???
```

Kleisli Example 1

```
/* Get `ServerType` field */
val getServerType:
  ServerConf => Option[ServerType] =
    sc => Option(sc.serverType)


/* Pull the `locale` of a server */
val getLocale: ServerType => Option[String] = {
  case Left(app) => app.appConf.get("locale")
  case Right(ms) => Option(ms.locale)
}
```

Kleisli Example 1

```
/* Extract region from locale string */
val getRegion: String => Option[String] =
 "(NA|EU|AP|AF)".r.findFirstIn(_)
```

```
/* Composing it all together */
val extractRegionKleisli:
  Kleisli[Option, ServerConf, String] =
    Kleisli(getServerType)
    >>> Kleisli(getLocale)
    >>> Kleisli(getRegion)

val extractRegion:
  ServerConf => Option[String] =
    extractRegionKleisli.run
```

Why build this way?

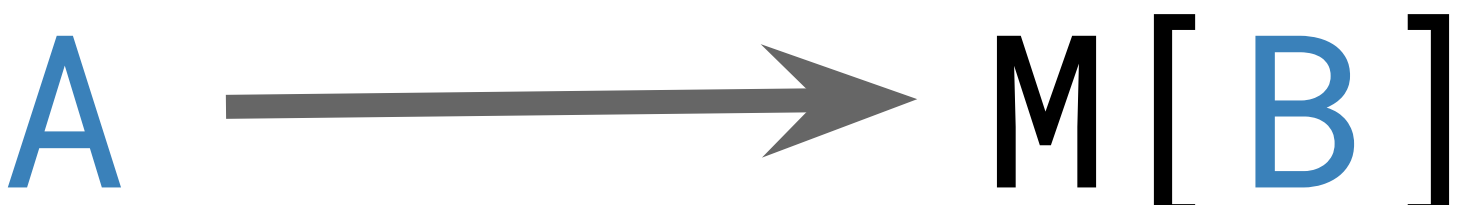
- We could accomplish the same by chaining `flatMap` calls together in a single function

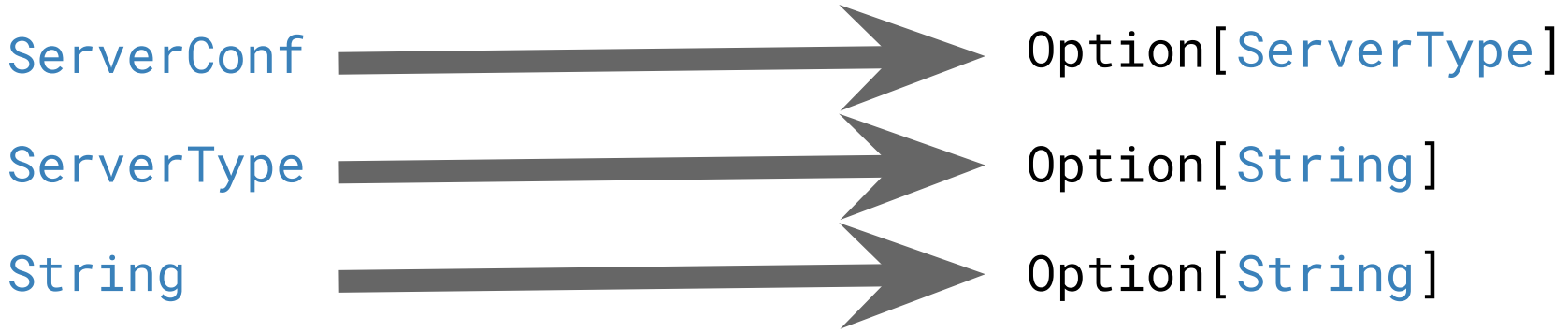
- But what if we wanted to swap out other parts of the pipeline?

```
val getLanguage: String => Option[String] =
 "(EN|NO|DE|FR)".r.findFirstIn(_)
```

- Arrows yield composability and modularity

Looks familiar...





Defining Kleisli

Cats Docs:

Kleisli enables composition of functions that return a monadic value.

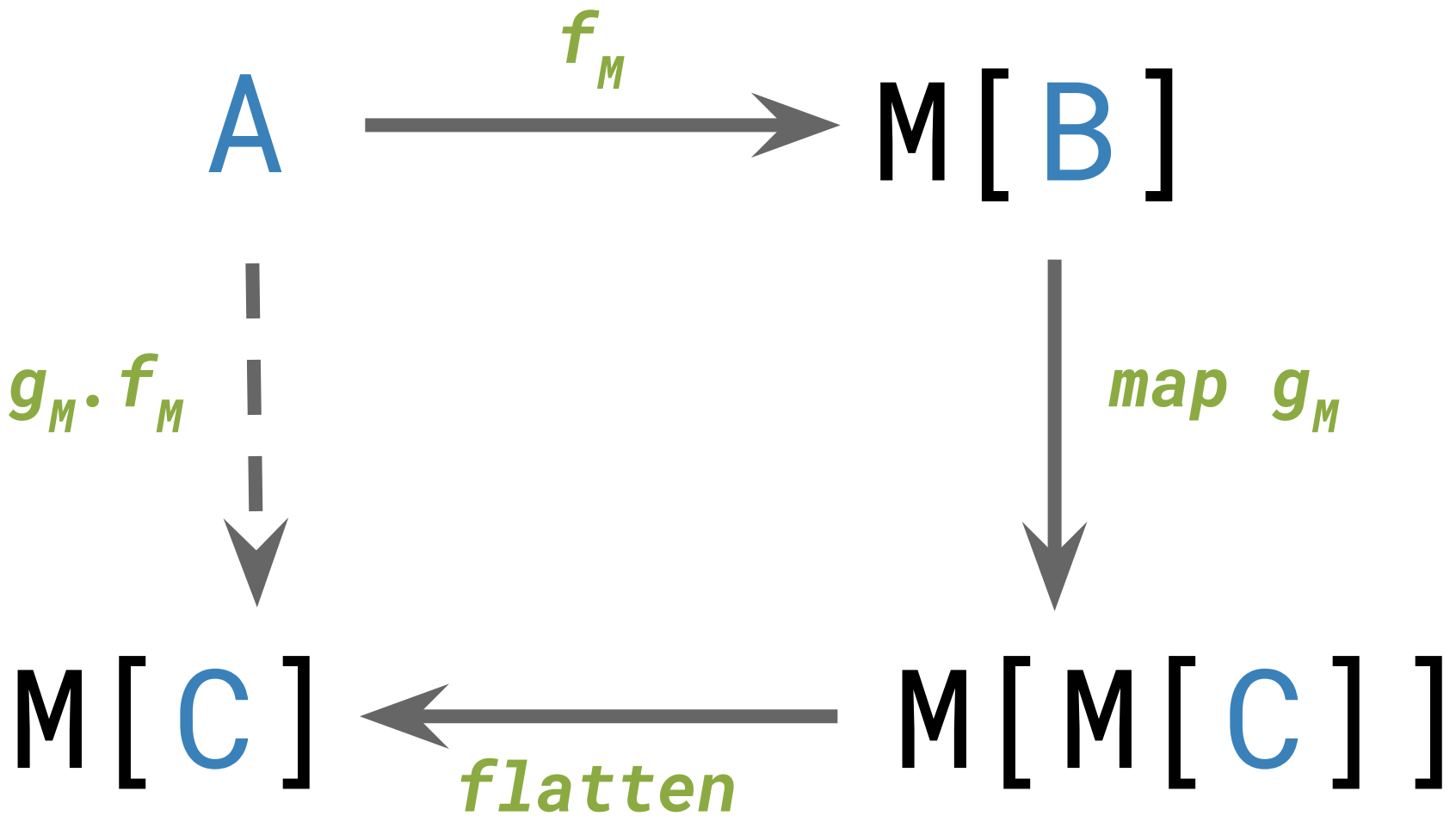
If F[_] has a FlatMap[F] instance, we can compose two Kleislis much like we can two functions.

Textbook:

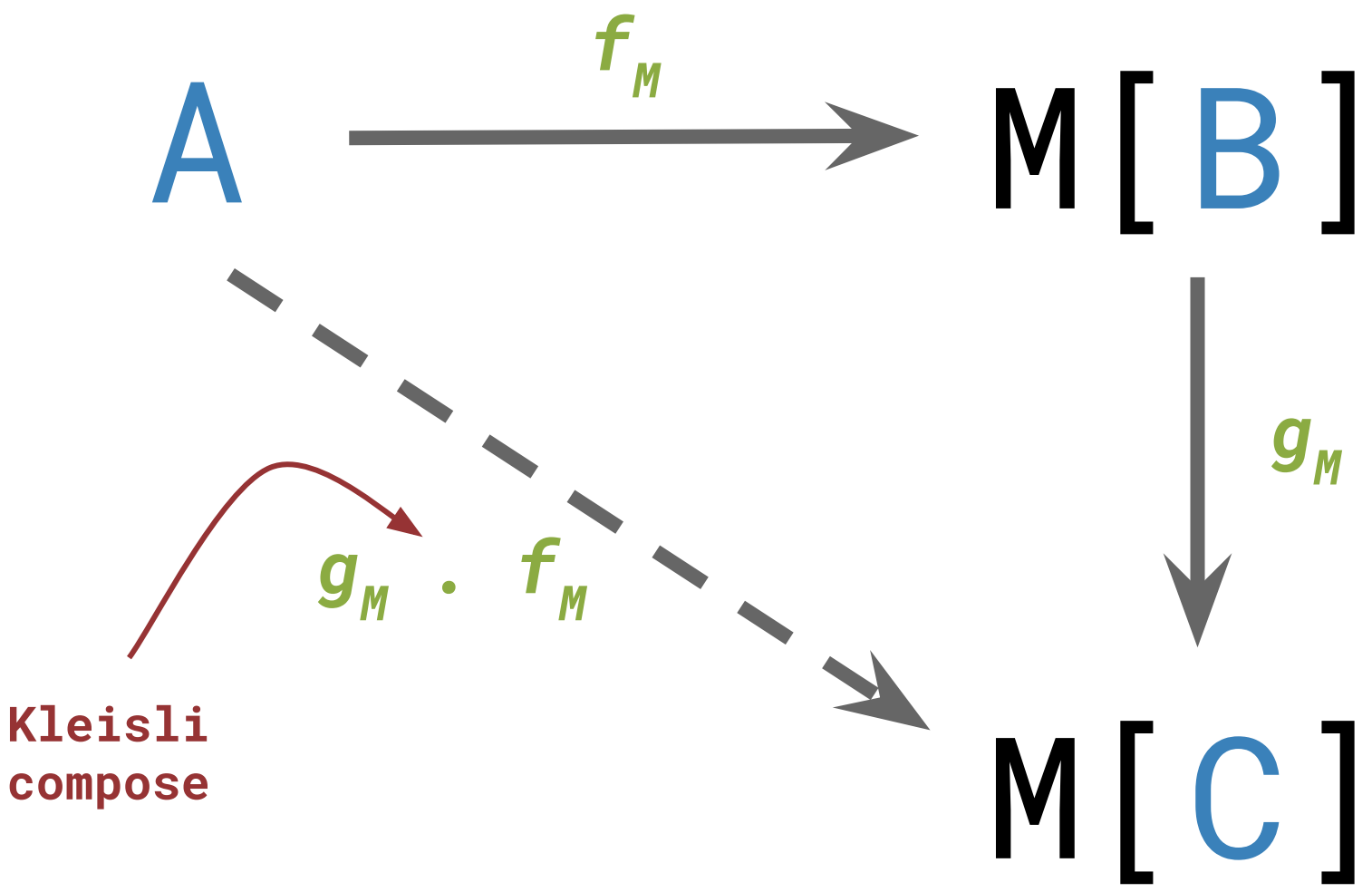
Given a monad (T, η, μ) on a category **C**, we can construct the Kleisli category $\mathbf{C}_T$ where:

- Objects are written A_T, B_T
- An arrow $f_T: A_T \rightarrow B_T$ in $\mathbf{C}_T$ is the arrow $f: A \rightarrow TB$ in **C**
- The identity arrow $1_{AT}: A_T \rightarrow A_T$ is $\boldsymbol{\eta}_A$: $A \rightarrow TA$ in **C**
- For composition, given $\mathbf{f}_T: A_T \rightarrow B_T$ and $\mathbf{g}_T: B_T \rightarrow C_T$, the composite $\mathbf{g}_T \circ \mathbf{f}_T: A_T \rightarrow C_T$ is defined as $\mu_C \circ Tg_T \circ f_T$

Kleisli Category



Kleisli Category



Kleisli Example 2

```
val getIP: Kleisli[Option, ServerConf, IP]


/* Simple health checks, but in IO[_] */
val ping: IP => IO[Double]
val health: IP => IO[Boolean]


/* Combine results of the arrows as tuple */
val pingHealth: Kleisli[IO, IP, (Double,Bool)] =
  Kleisli(ping) merge Kleisli(health)
```

Option != IO… can we still compose?

Kleisli Example 2

```
val getIP: Kleisli[Option, ServerConf, IP]
val pingHealth: Kleisli[IO, IP, (Double, Bool)]

val default = (0.0,false).pure[IO]
```

```
/* Transform Option -> IO using `mapF` */
val extractPingHealth:
  Kleisli[IO, ServerConf, (Double, Bool)] =
    getIP.mapF(_.fold(default)(pingHealth.run))
```

Recap

Closing remarks

What We Haven't Seen

- Burritos, boxes, pipes, etc.
- Value-centric programming or "data container" style
- Hiding from math terminology

What We **Have** Seen

```
// Plain old function composition
(A => B) => (B => C) => (A => C)


// Functor arrow composition; `lift`
(A => F[B]) => (B => C) => (A => F[C])


// Monadic composition with `Kleisli`
(A => M[B]) => (B => M[C]) => (A => M[C])
```

What We **Have** Seen

- Focusing on relationships between objects
  - We can compose functions from functions
  - Highly modular approach to programming
- A subset of Scala allows us to write functional code closer to category diagrams and definitions
  - Arrows are key to this
- Arrow programming can be fun and useful
  - Point free style is not pointless 😁

Acknowledgements

- Cody Allen (@ceedubs) for technical input and help shaping the presentation
- My team at **Salesforce** for their support
- “FP Chat” & gitter/cats for enduring my questions

Sources

- *Category Theory*
  - *Steve Awodey*
- *Category Theory for Programmers*
  - *Bartosz Milewski*
- *A Categorical View of Computational Effects*
  - *Emily Riehl*
- *Cats Library & Documentation*
  - *Typelevel*


Thank You!

TEMPLATE BELOW

About this template

What's this?

This is a free presentation template for Google Slides designed by SlidesCarnival.

We believe that good design serves to better communicate ideas, so we create free quality presentation templates for you to focus on the content.

Enjoy them at will and share with us your results at:

twitter.com/SlidesCarnival

facebook.com/slidescarnival

How can I use it?

Open this document in Google Slides (if you are at slidescarnival.com use the button below this presentation)

You have to be signed in to your Google account

- **Edit in Google Slides**
  Go to the **File** menu and select **Make a copy**. You will get a copy of this document on your Google Drive and will be able to edit, add or delete slides.
- **Edit in Microsoft PowerPoint®**
  Go to the **File** menu and select **Download as Microsoft PowerPoint**. You will get a .pptx file that you can edit in PowerPoint. Remember to download and install the fonts used in this presentation (you'll find the links to the font files needed in the Presentation design slide)

hello!

I am Jayden Smith

I am here because I love to give presentations.

You can find me at @username

1.
Transition headline

Let's start with the first set of slides

Quotations are commonly printed as a means of inspiration and to invoke philosophical thoughts from the reader.

This is a slide title

- Here you have a list of items
- And some text
- But remember not to overload your slides with content

You audience will listen to you or read the content, but won’t do both.

big concept

Bring the attention of your audience over a key concept using icons or illustrations

You can also split your content

White

Is the color of milk and fresh snow, the color produced by the combination of all the colors of the visible spectrum.

Black

Is the color of coal, ebony, and of outer space. It is the darkest color, the result of the absence of or complete absorption of light.

In two or three columns

Yellow

Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.

Blue

Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.

Red

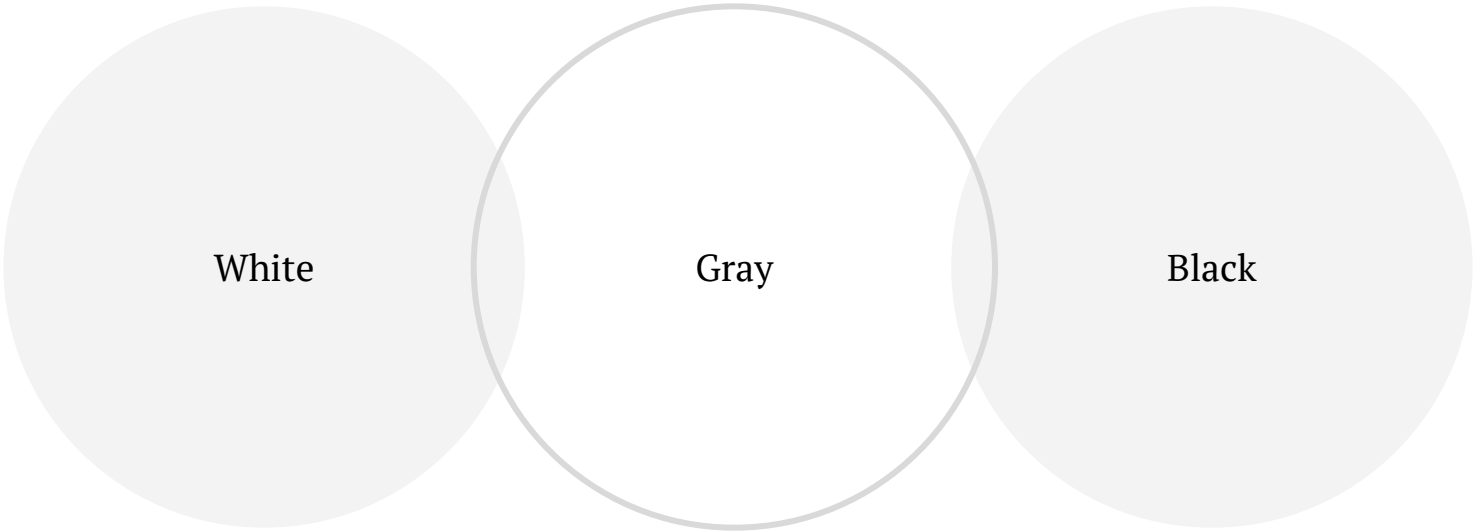
Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.

A picture is worth a thousand words

A complex idea can be conveyed with just a single still image, namely making it possible to absorb large amounts of data quickly.


want big impact?
Use big image

Use charts to explain your ideas



And tables to compare data

	A	B	C
Yellow	**10**	**20**	**7**
Blue	**30**	**15**	**10**
Orange	**5**	**24**	**16**

Maps

89,526,124

Whoa! That’s a big number, aren’t you proud?

89,526,124$

That’s a lot of money

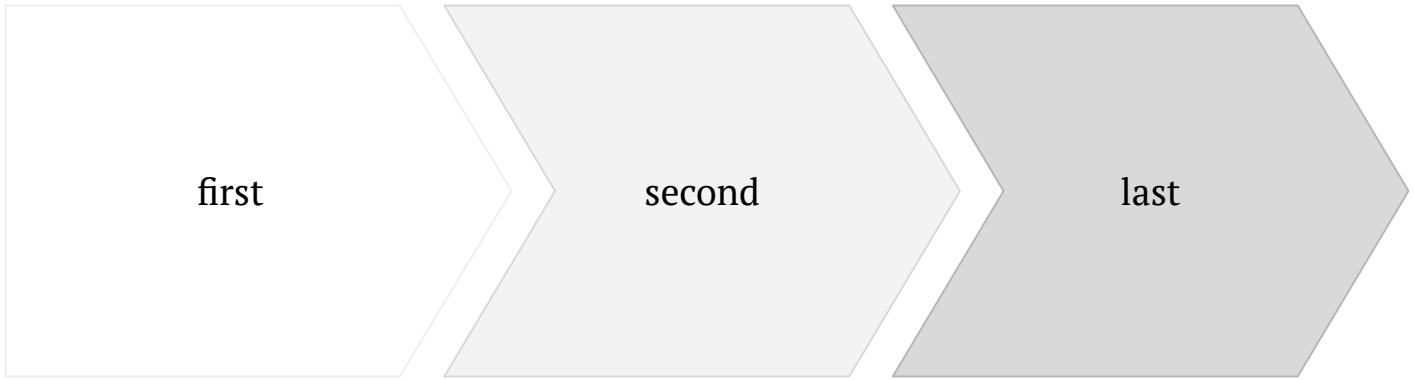
185,244 users

And a lot of users

100%

Total success!

Our process is easy



Let’s review some concepts

Yellow

Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.

Blue

Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.

Red

Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.

Yellow

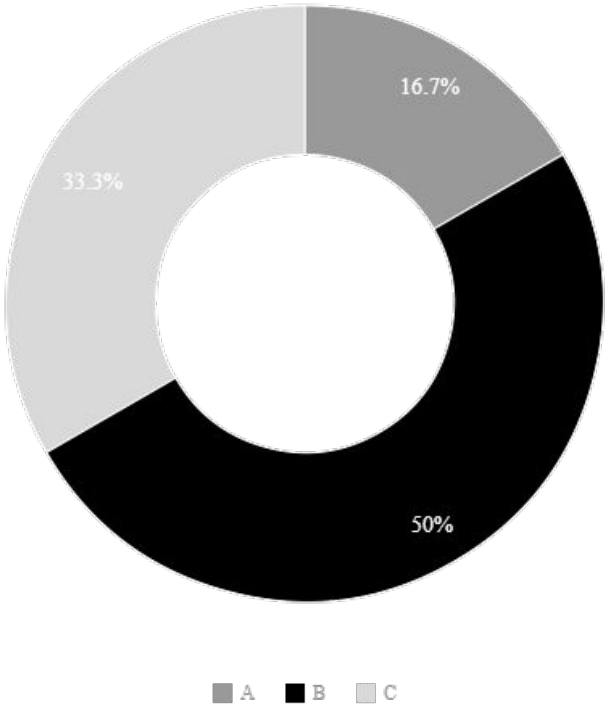
Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.

Blue

Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.

Red

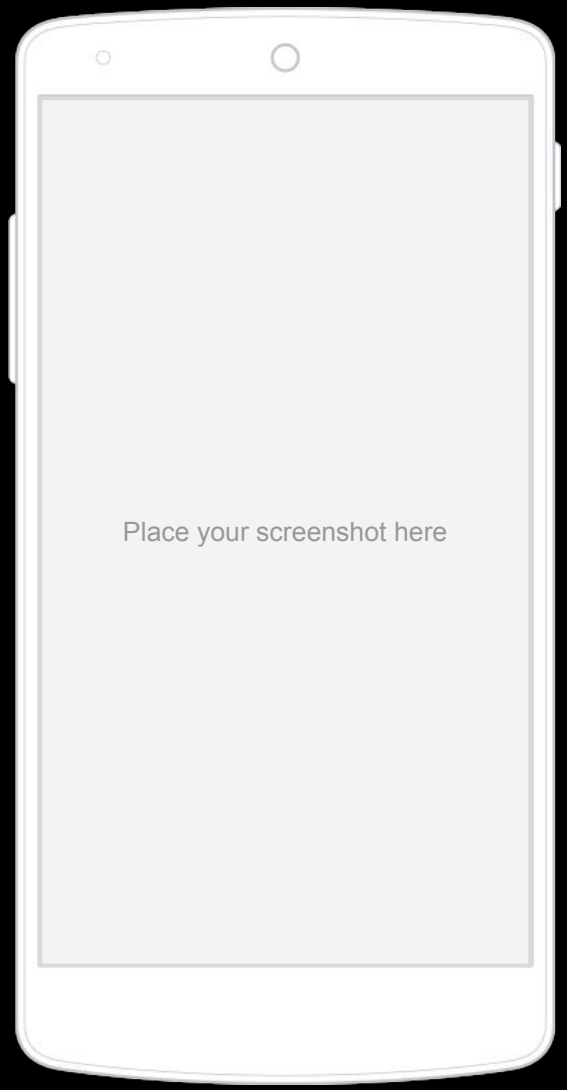
Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.



You can copy&paste graphs from Google Sheets

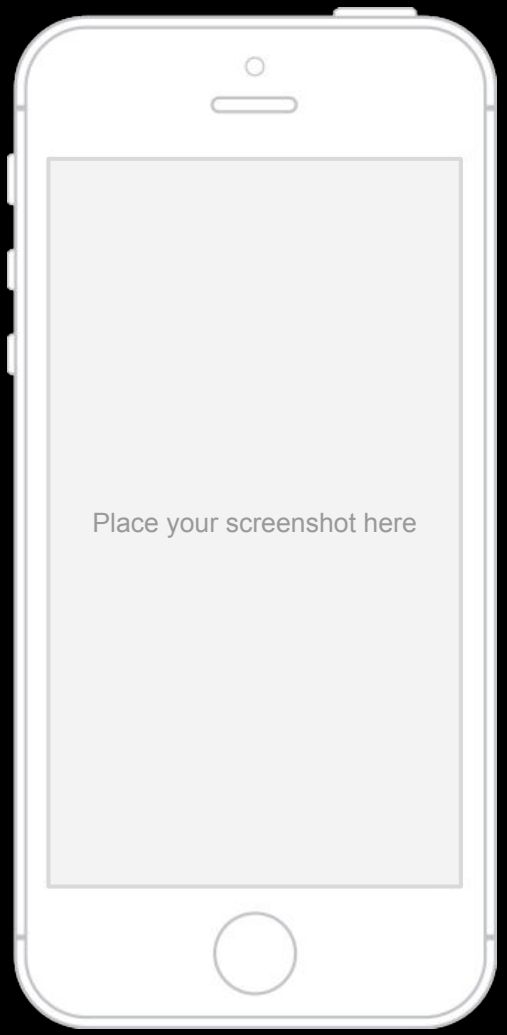
Android project

Show and explain your web, app or software projects using these gadget templates.



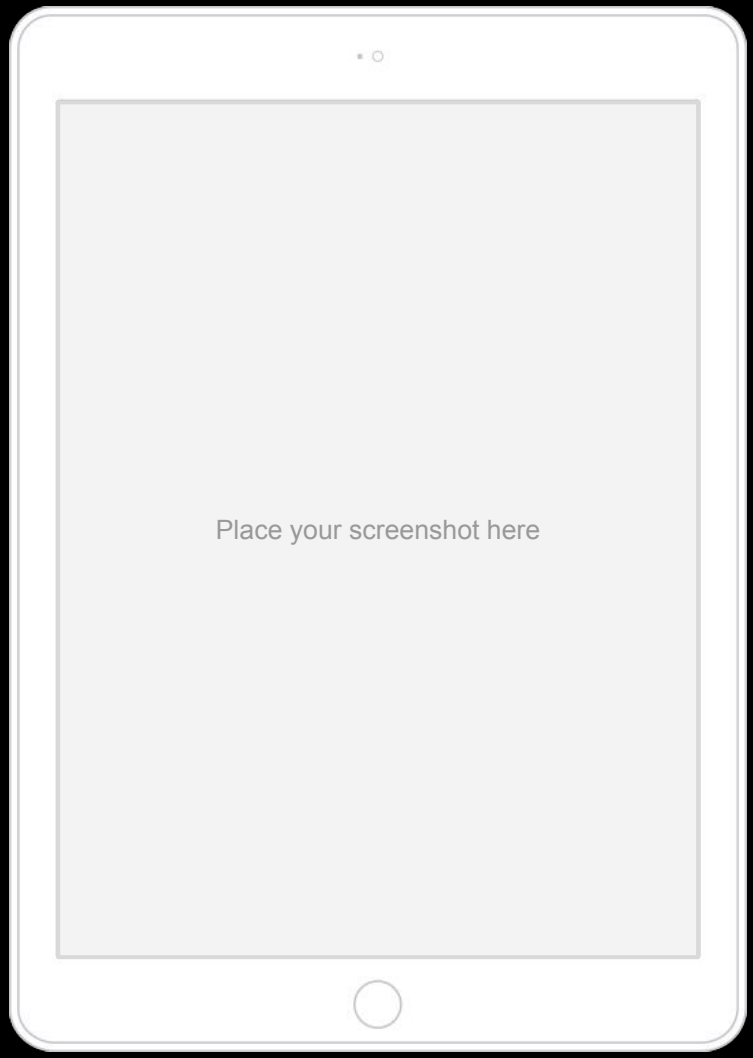
iPhone project

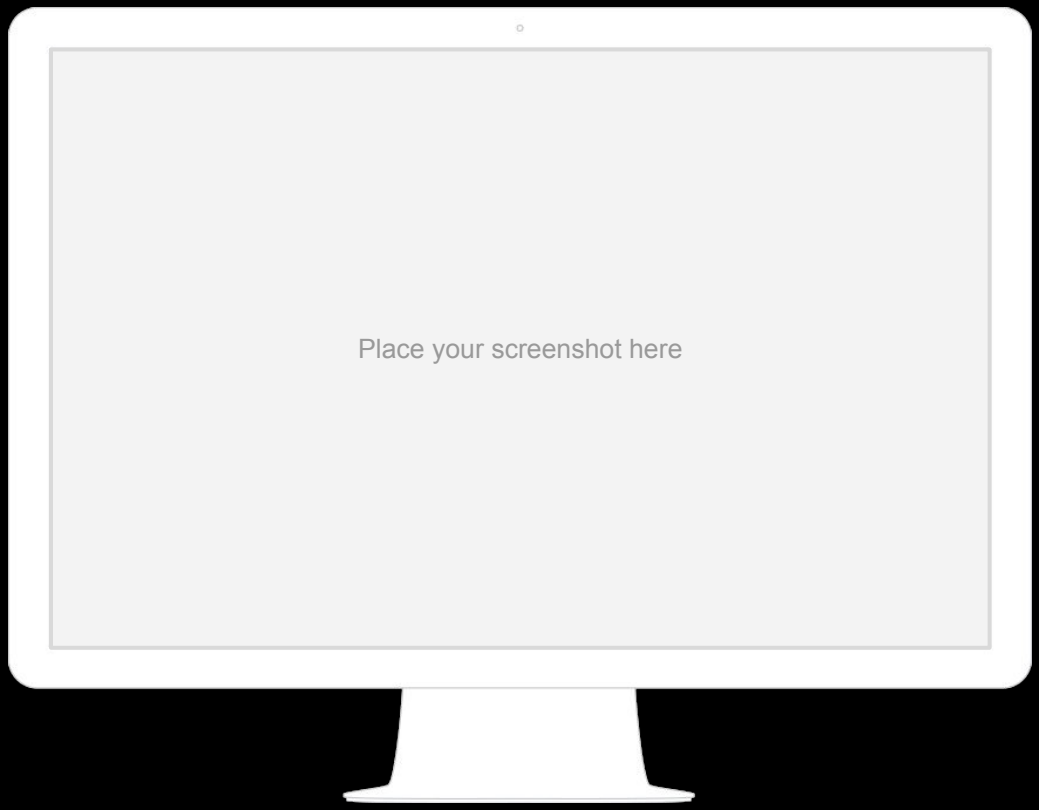
Show and explain your web, app or software projects using these gadget templates.



Tablet project

Show and explain your web, app or software projects using these gadget templates.





Desktop project

Show and explain your web, app or software projects using these gadget templates.

thanks!

Any questions?

You can find me at

@username

user@mail.me

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Simple line icons by Mirko Monti
- E-commerce icons by Virgil Pana
- Streamline iconset by Webalys
- Presentation template by SlidesCarnival
- Photographs by Unsplash

Presentation design

This presentations uses the following typographies:

- Titles: **Playfair Display**
- Body copy: **PT Serif**

You can download the fonts on this page:

https://www.google.com/fonts#UsePlace:use/Collection:Playfair+Display|PT+Serif

Click on the “arrow button” that appears on the top right 

You don’t need to keep this slide in your presentation. It’s only here to serve you as a design guide if you need to create new slides or download the fonts to edit the presentation in PowerPoint®