A Practical Guide to Protocols

Kevin Rockwood

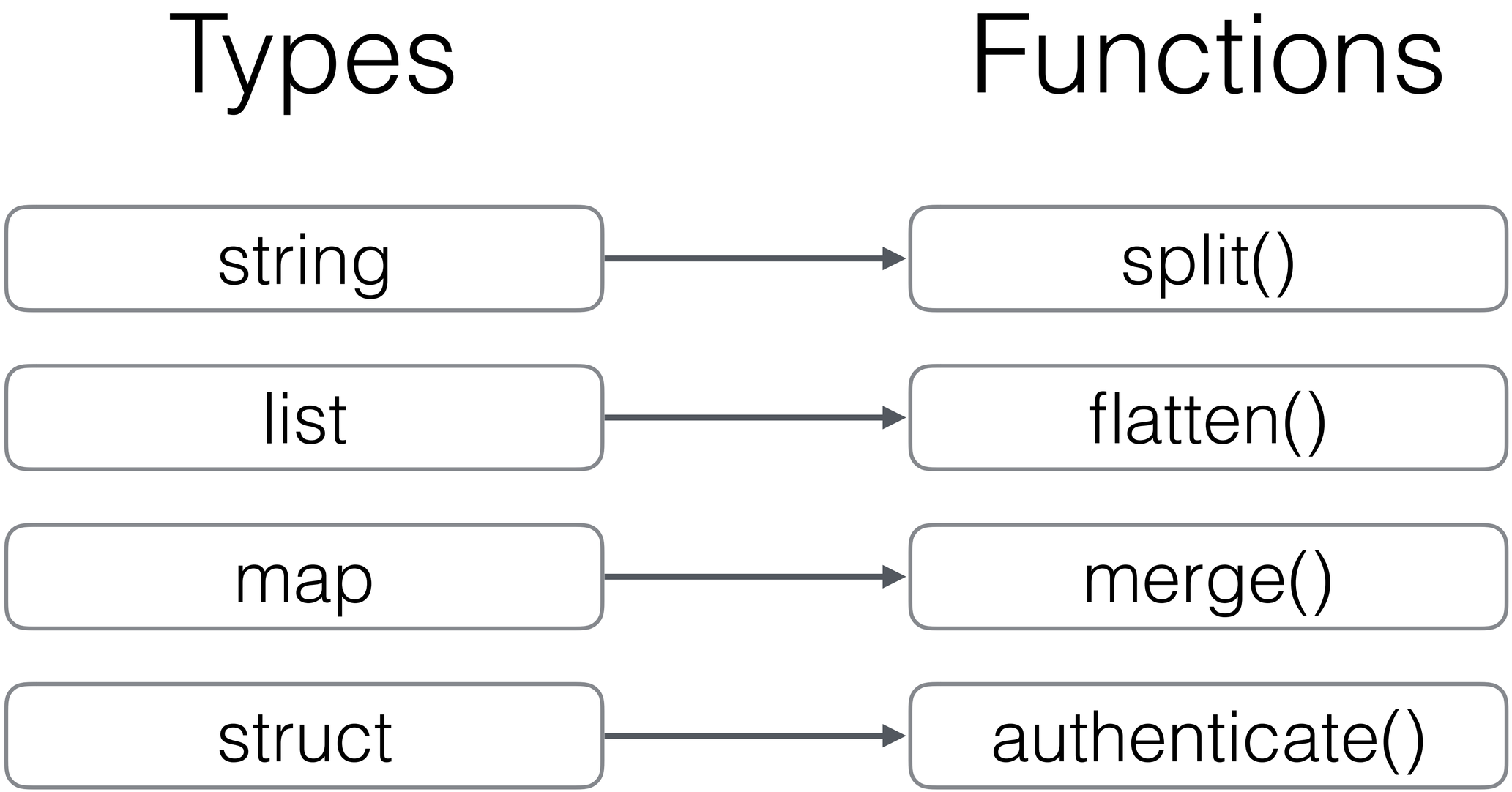
Hi, I'm Kevin


SkelterLabs

Protocols

Programs are made of two elements

Types & Functions


Types
Functions
string
split()
list
flatten()
map
merge()
struct
authenticate()

Types & Functions

Languages organize them in the most optimal way possible

The Expression Problem

How can we add new **types** and **functions** without modifying our existing ones

Types

Circle

Rectangle

Functions

area()

perimeter()

Let's try Imperative

(Object Oriented)

```ruby
class Rectangle
  def initialize(width, height)
    @width = width
    @height = height
  end

  def area
    @width * @height
  end
end
```

```ruby
class Circle
  def initialize(radius)
    @radius = radius
  end

  def area
    Math::PI * @radius ** 2
  end
end
```

```ruby
class Printer
  def self.print(shape)
    puts shape.area()
  end
end
```

- New types? 👍

```ruby
class Rectangle
  def initialize(width, height)
    @width = width
    @height = height
  end

  def area
    @width * @height
  end
end
```

```ruby
class Circle
  def initialize(radius)
    @radius = radius
  end

  def area
    Math::PI * @radius ** 2
  end
end
```

```ruby
class Printer
  def self.print(shape)
    puts shape.perimeter()
  end
end
```

- New types? 👍
- New functions? 👎

Let's try Functional

```elixir
defmodule Rectangle do
  defstruct length: 0, width: 0

  def area(%{length: length, width: width}) do
    length * width
  end
end
```

```elixir
defmodule Circle do
  defstruct radius: 0

  def area(%{radius: radius}) do
    :math.pi() * :math.pow(radius, 2)
  end
end
```

```elixir
defmodule Printer do
  def print(%Rectangle{} = shape) do
    IO.puts Rectangle.area(shape)
  end
end
```

- New types? 👎

```elixir
defmodule Rectangle do
  defstruct length: 0, width: 0

  def area(%{length: length, width: width}) do
    length * width
  end
end
```

```elixir
defmodule Circle do
  defstruct radius: 0

  def area(%{radius: radius}) do
    :math.pi() * :math.pow(radius, 2)
  end
end
```

```elixir
defmodule Printer do
  def print(shape) do
    IO.puts perimeter(shape)
  end
end
```

- New types? 👎

```elixir
defmodule Rectangle do
  defstruct length: 0, width: 0

  def area(%{length: length, width: width}) do
    length * width
  end
end
```

```elixir
defmodule Circle do
  defstruct radius: 0

  def area(%{radius: radius}) do
    :math.pi() * :math.pow(radius, 2)
  end
end
```

```elixir
defmodule Printer do
  def print(shape) do
    IO.puts perimeter(shape)
  end

  def perimeter(%Rectangle{length: length, width: width})
    2 * length + 2 * width
  end
  def perimeter(%Circle{radius: radius}) do
    2 * :math.pi() * radius
  end
end
```

- New types? 👎
- New functions? 👍

The Expression Problem

	Imperative	**Functional**
Add Types	👍	👎
Add Functions	👎	👍

Protocols

Influenced by

Clojure

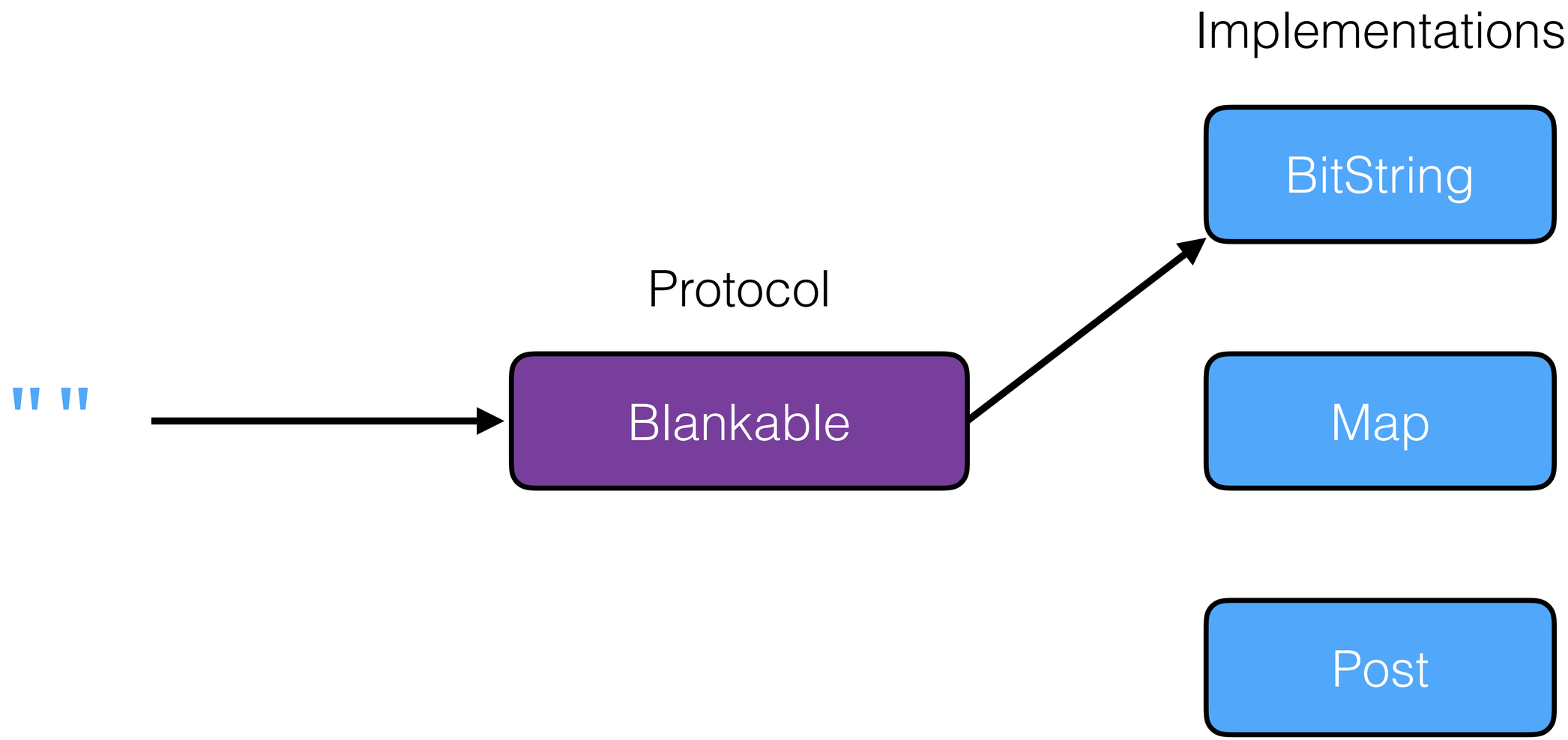
Elixir Only

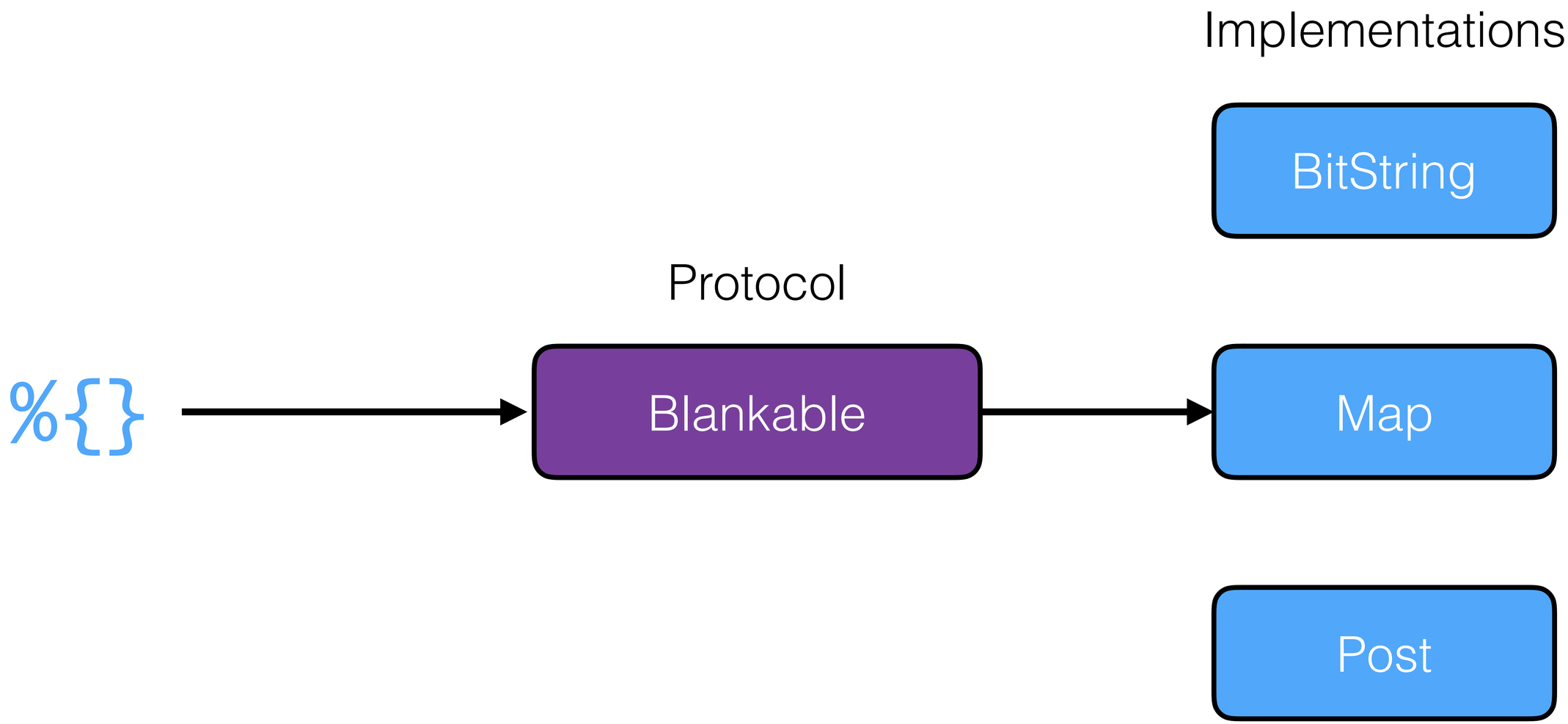
Decouples the definition of a **function** from its implementation for a specific data **type**

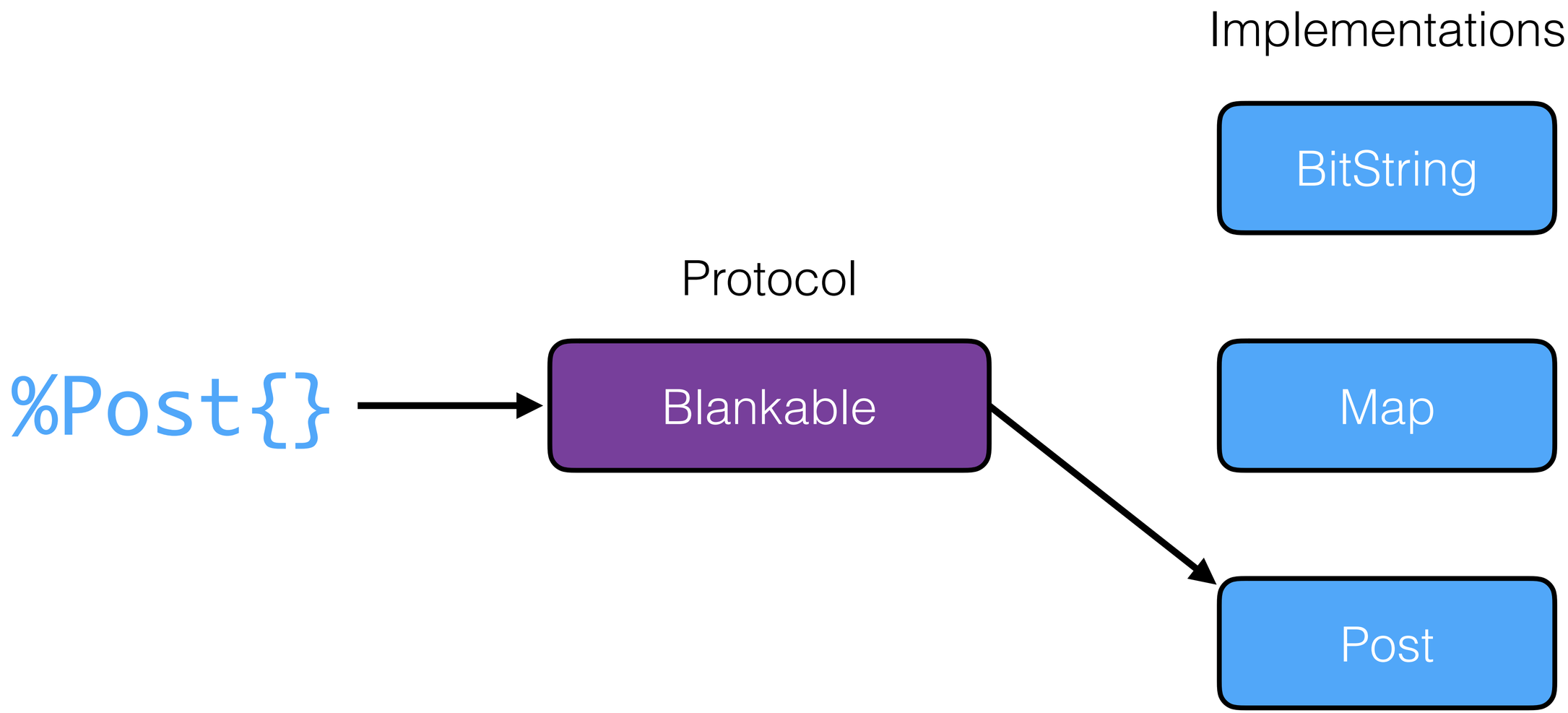

Data type
Protocol
Implementation

```elixir
defprotocol Blankable do
  def blank?(term)
end

defimpl Blankable, for: BitString do
  def blank?(""), do: true
  def blank?(_), do: false
end

defimpl Blankable, for: Map do
  def blank?(map), do: map_size(map) == 0
end

defmodule Post do
  defstruct [:title, :body]

  defimpl Blankable do
    def blank?(%{body: nil}), do: true
    def blank?(%{body: _}), do: false
  end
end
```

```
> Blankable.blank?("foo")
false

> Blankable.blank?(%{body: nil})
false

> Blankable.blank?(%Post{body: nil})
true

> Blankable.blank?(:foo)
** (Protocol.UndefinedError) protocol Blankable
   not implemented for :foo
```


Implementations
BitString
Protocol
" "
Blankable
Map
Post


Implementations
BitString
Protocol
%{}
Blankable
Map
Post


Implementations
BitString
Protocol
%Post{}
Blankable
Map
Post

Any

```elixir
defprotocol Blankable do
  @fallback_to_any true
  def blank?(term)
end

defimpl Blankable, for: Any do
  def blank?(_), do: false
end
```



```
> Blankable.blank?(:foo)
false
```

```elixir
defprotocol Blankable do
  def blank?(term)
end


defimpl Blankable, for: Any do
  def blank?(_), do: false
end


defmodule Post do
  @derive [Blankable]
  defstruct [:title, :body]
end
```

```
> Blankable.blank?(%Post{})
false
```

Back to the Expression Problem

```elixir
defmodule Rectangle do
  defstruct length: 0, width: 0

  defimpl Area do
    def calc(%{length: length, width: width}) do
      length * width
    end
  end
end
```

```elixir
defmodule Circle do
  defstruct radius: 0

  defimpl Area do
    def calc(%{radius: radius}) do
      :math.pi() * :math.pow(radius, 2)
    end
  end
end
```

```elixir
defmodule Printer do
  def print(shape) do
    IO.puts Area.calc(shape)
  end
end
```

```elixir
defprotocol Area do
  def calc(shape)
end
```

- New type? 👍

```elixir
defmodule Rectangle do
  defstruct length: 0, width: 0

  defimpl Area do
    def calc(%{length: length, width: width}) do
      length * width
    end
  end
end
```

```elixir
defmodule Circle do
  defstruct radius: 0

  defimpl Area do
    def calc(%{radius: radius}) do
      :math.pi() * :math.pow(radius, 2)
    end
  end
end
```

```elixir
defmodule Printer do
  def print(shape) do
    IO.puts Perimeter.calc(shape)
  end
end

defimpl Perimeter, for: Rectangle do
  def calc(%{length: length, width: width}) do
    2 * length + 2 * width
  end
end

defimpl Perimeter, for: Circle, do: ...
```

```elixir
defprotocol Area do
  def calc(shape)
end
```

- New type? 👍
- New function? 👍

Core Protocols

String.Chars

Responsible for converting a data type to a string

- `to_string/1`

```
> to_string(:foo)
"foo"

> "Hello #{:foo}"
"Hello foo"
```

Collectable

Used to take values out of a collection

- `into/2`

```
> Enum.into([a: 1, b: 2], %{})
%{a: 1, b: 2}
```

Inspect

Responsible for converting any Elixir data structure into a pretty printed format

- `inspect/1`

```
> inspect(%Post{title: "My post"})
"%Post{title: \"My post\"}"
```

IEx.Info

Prints helpful info inside an IEx session

- `info/1`

```
> i :foo
Term
  :foo
Data type
  Atom
Reference modules
  Atom
Implemented protocols
  IEx.Info, Inspect, List.Chars, String.Chars
```

Enumerable

Used by `Enum` and `Stream` modules to interact with list-like structures

- `count/1`
- `member?/2`
- `reduce/3`

```
> Enum.map([1, 2, 3], &(&1 * 2))
[2, 4, 6]
```

Protocols in the Wild

poison 3.1.0

An incredibly fast, pure Elixir JSON library

Maintainers

Devin Torres

Links

GitHub
Online documentation (download)

License

CC0-1.0

82 750 downloads this version
4 607 downloads yesterday
34 964 downloads last 7 days
2 799 861 downloads all time

Config

mix.exs

```
{:poison, "~> 3.1"}
```

Build Tools

mix

Owners


devinus

```
> Poison.encode(%{my_int: 1, my_atom: :two})
> {:ok, "{\"my_int\":1,\"my_atom\":\"two\"}"}


defprotocol Poison.Encoder do
  @fallback_to_any true

  def encode(value, options)
end

defimpl Poison.Encoder, for: Atom do
  def encode(nil, _),   do: "null"
  def encode(true, _),  do: "true"
  def encode(false, _), do: "false"

  def encode(atom, _options) do
    Atom.to_string(atom)
  end
end

defimpl Poison.Encoder, for: MyPerson do
  def encode(%{name: name, age: age}, _options) do
    "#{name} (#{age})"
  end
end
```

scrivener 2.3.0

Pagination for the Elixir ecosystem

Maintainers

Drew Olson

Links

github
Online documentation (download)

License

MIT

3 351 downloads this version
578 downloads yesterday
4 025 downloads last 7 days
208 627 downloads all time

Config

mix.exs

```
{:scrivener, "~> 2.3"}
```

Build Tools

mix

Owners

drewolson

```
> MyApp.Repo.paginate(User, %{page_size: 10})
%Scrivener.Page{entries: [...], page_number: 1, page_size: 10, total_entries: 100, total_pages: 10}

defprotocol Scrivener.Paginater do
  def paginate(pageable, config)
end

defimpl Scrivener.Paginater, for: Ecto.Query do
  import Ecto.Query

  def paginate(query, %{page_size: page_size, page_number: page_number, module: repo}) do
    query = from q in query,
      limit: page_size,
      offset: page_number * page_size

    %Scrivener.Page{entries: repo.all(query) page_size: page_size, ...}
  end
end

defimpl Scrivener.Paginater, for: MyThing do
  def paginate(my_thing, %{page_size: page_size, page_number: page_number}) do
    ...
  end
end
```

API Integration

```
> %FetchUsersMessage{role: "admin"} |> Api.send_message()

defprotocol Api::Message do
  def build_request(message)
  def parse_response(message, response)
end

defmodule Api do
  def send_message(message) do
    request = Api::Message.build_request(message)
    response = Http.perform(request)
    Api::Message.parse_response(message, response)
  end
end

defmodule FetchUsersMessage do
  defstruct role: ""

  defimpl Api::Message do
    def build_request(%{role: role}) do: # build request body

    def parse_response(%{role: role}, response), do: # parse response body
  end
end
```

Protocol or Behaviour

Can each implementation of my **function** take the same data **type**?

```
> message = %Message{to: "george@example.com", subject: "Hi George!" body: "..."}
> Mailer.deliver_now(message)
```

```
defmodule Mailer do
  def deliver_now(message) do
    get_current_adapter().deliver(message)
  end
end

defmodule SMTPAdapter do
  def deliver(message), do: ...
end

defmodule SendgridAdapter do
  def deliver(message), do: ...
end

defmodule TestAdapter do
  def deliver(message), do: ...
end
```

Don't get carried away

Another tool in your toolbox

고맙습니다

Thank you

github.com/rockwood