Designing Elixir Systems with OTP

Write Highly Scalable, Self-Healing Software with Layers

James Edward Gray, II
Bruce A. Tate
edited by Jacquelyn Carter

Designing Elixir Systems with OTP

Write Highly Scalable, Self-Healing Software with Layers

@redrapids

James Edward Gray, II
Bruce A. Tate
edited by Jacquelyn Carter

grox.io

Learning

Career Rocket Fuel for Curious Coders

bring on the

FREE BEES

20% off PP books:

CodeBEAM_SF_2020

Designing Elixir Systems with OTP

Write Highly Scalable, Self-Healing Software with Layers

"Do Fun Things with Big, Loud Worker-Bees"

by James E Gray, II
and Bruce A. Tate

Designing Elixir Systems with OTP

Write Highly Scalable, Self-Healing Software with Layers

"Do Fun Things with Big, Loud Worker-Bees"

by James E Gray, II and Bruce A. Tate

Do

Fun

Things

Big

Loud

WBees

Data

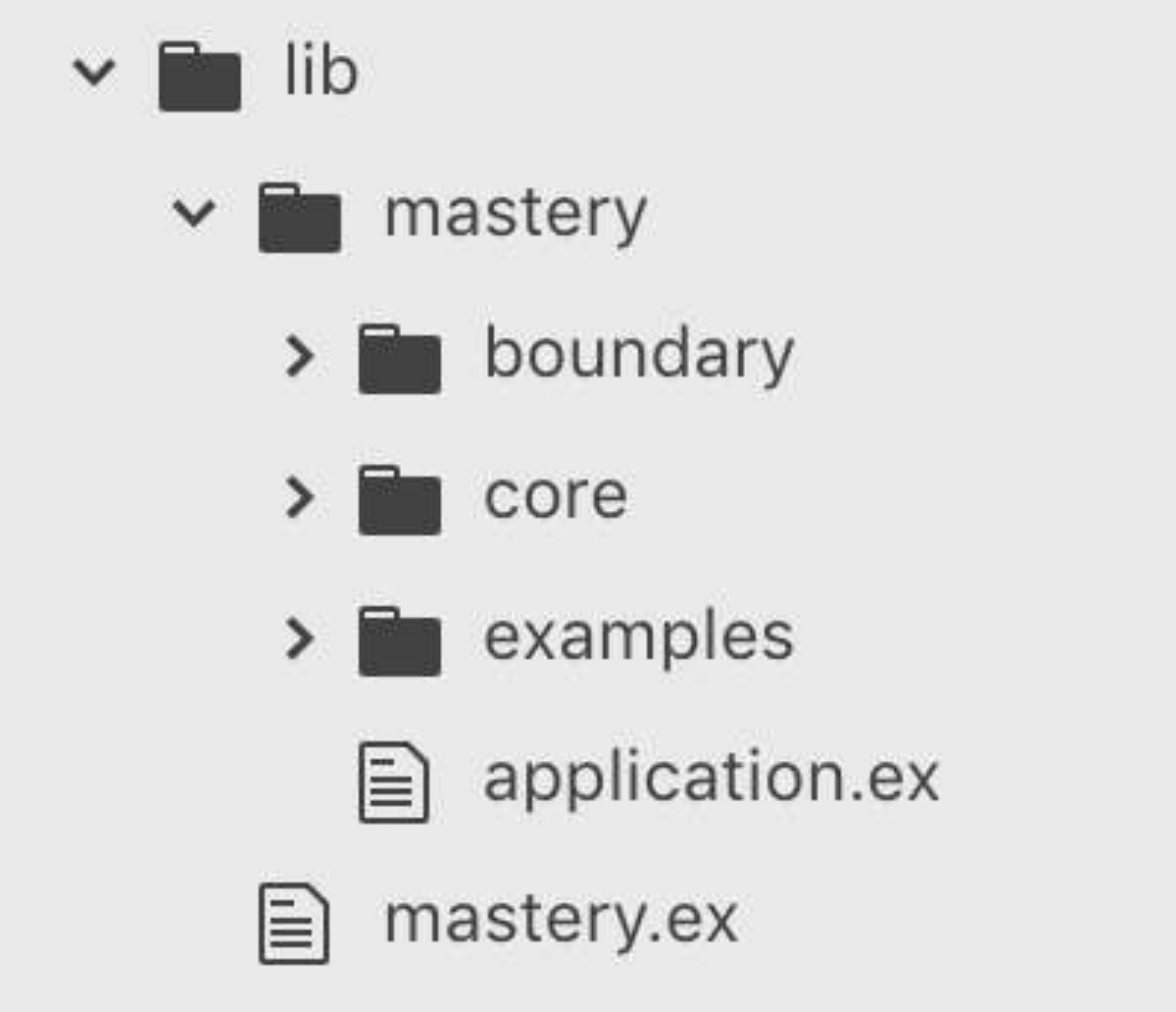
Functions

Tests

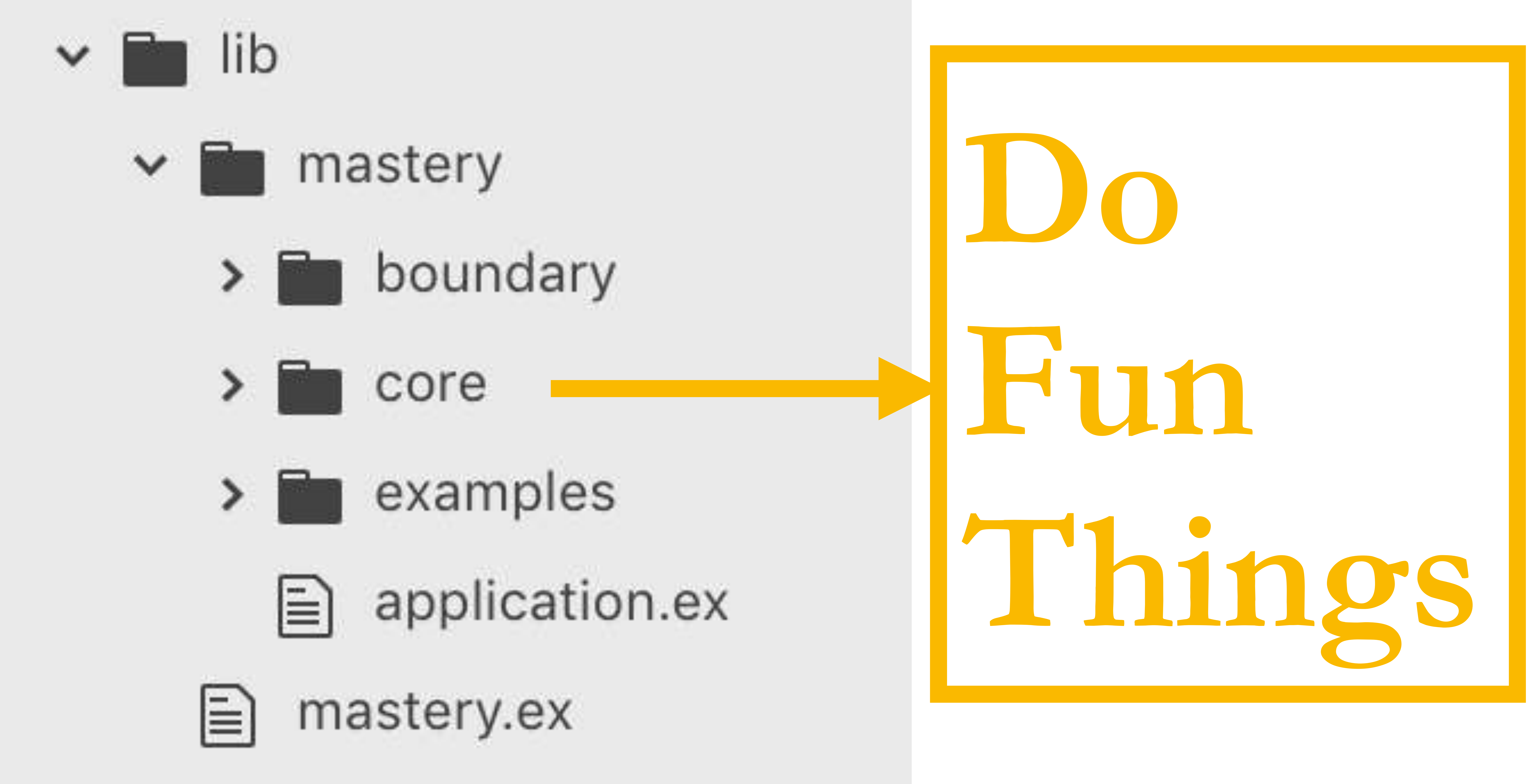
Boundaries

Lifecycles

Workers

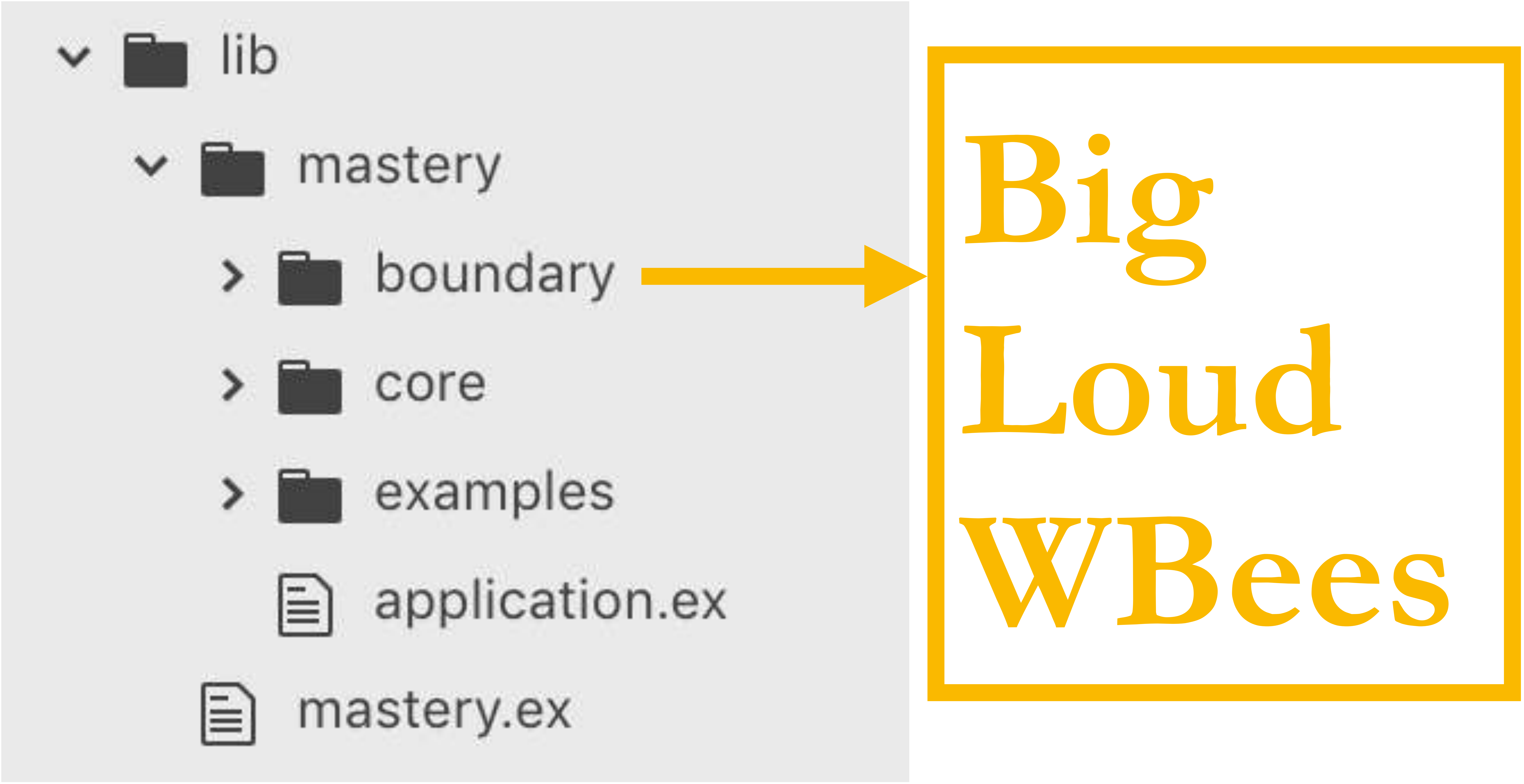


```
lib
  mastery
    boundary
    core
    examples
    application.ex
  mastery.ex
```



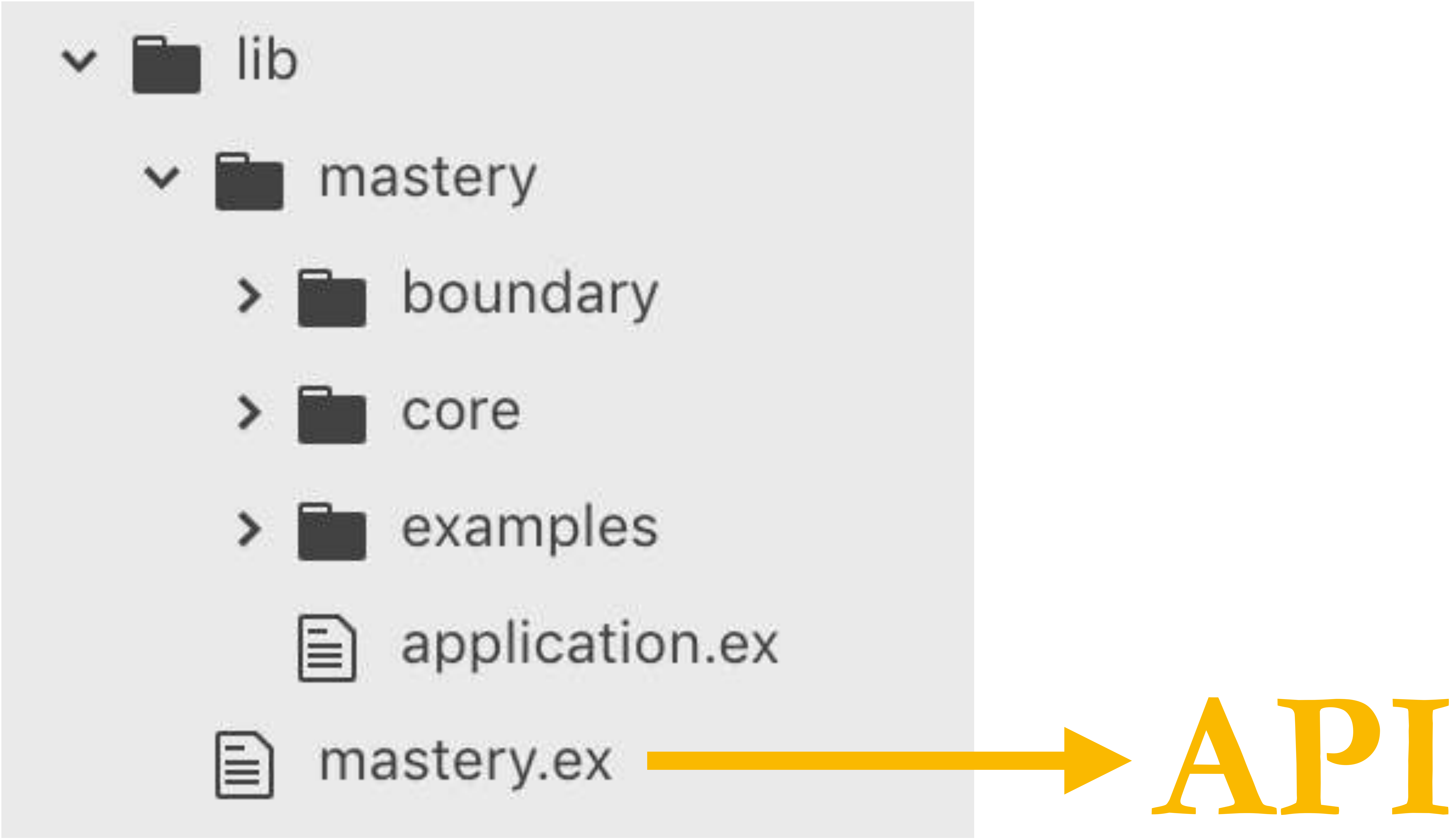
```
lib
  mastery
    boundary
    core
    examples
    application.ex
  mastery.ex
```

Do
Fun
Things



```
lib
  mastery
    boundary
    core
    examples
    application.ex
  mastery.ex
```

Big
Loud
WBees



```
lib
  mastery
    boundary
    core
    examples
    application.ex
  mastery.ex
```

API

Designing Elixir Systems with OTP

Write Highly Scalable, Self-Healing Software with Layers

"Do Fun Things with Big, Loud Worker-Bees"

by James E Gray, II
and Bruce A. Tate

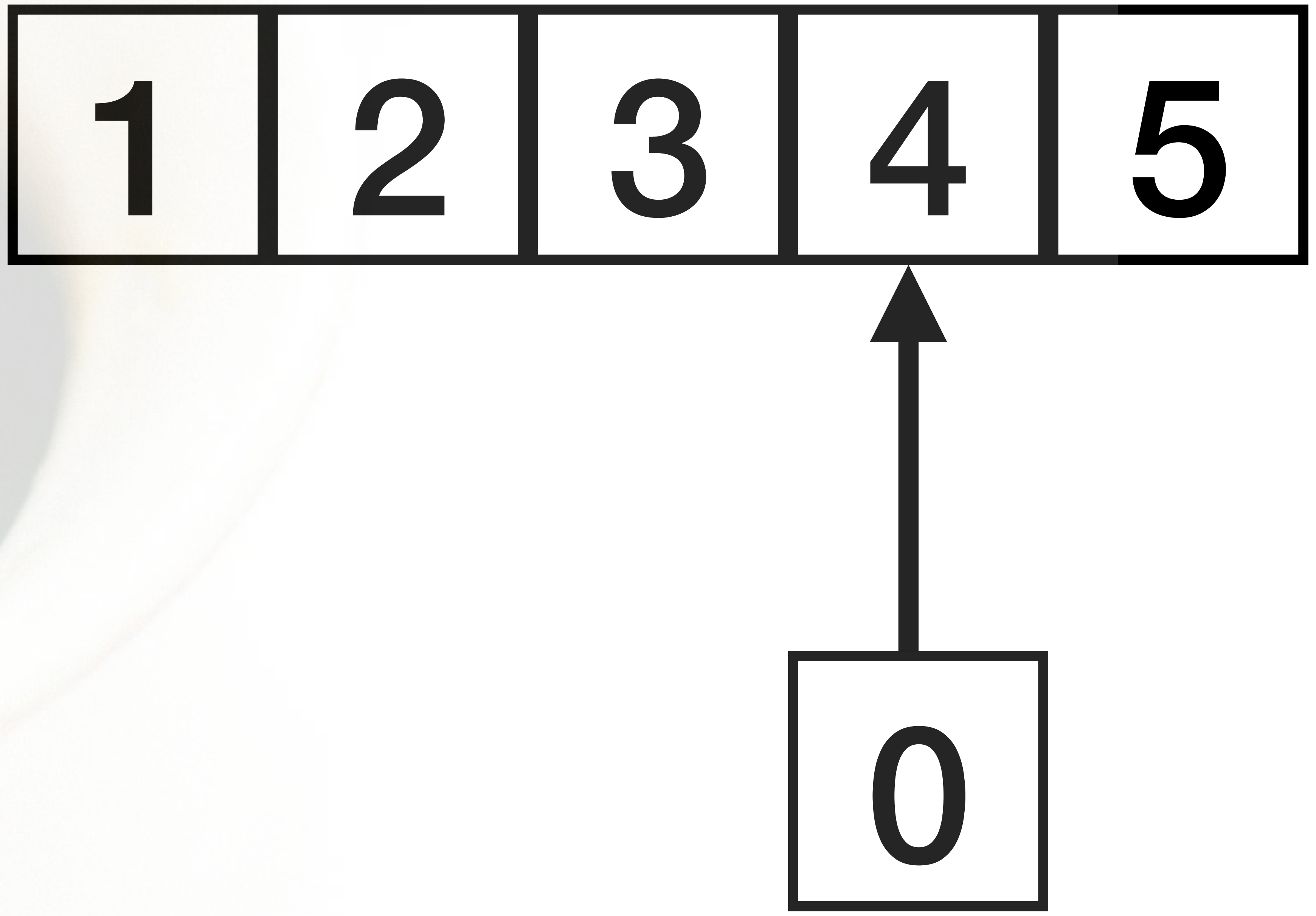
Do

Data

Harmony

Cost of Immutability

OOP



1	2	3	4	5

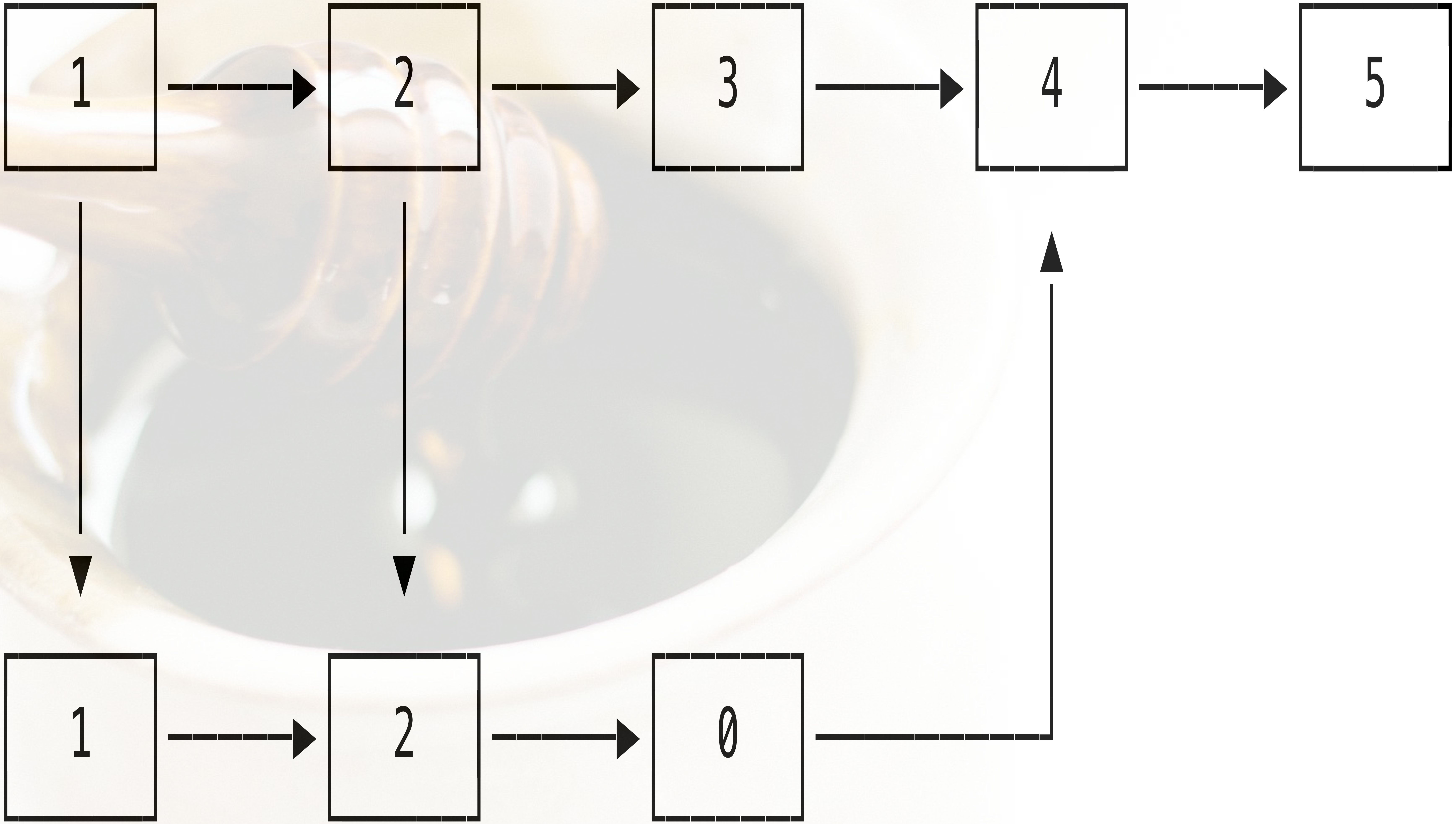
0

a[3] = 0

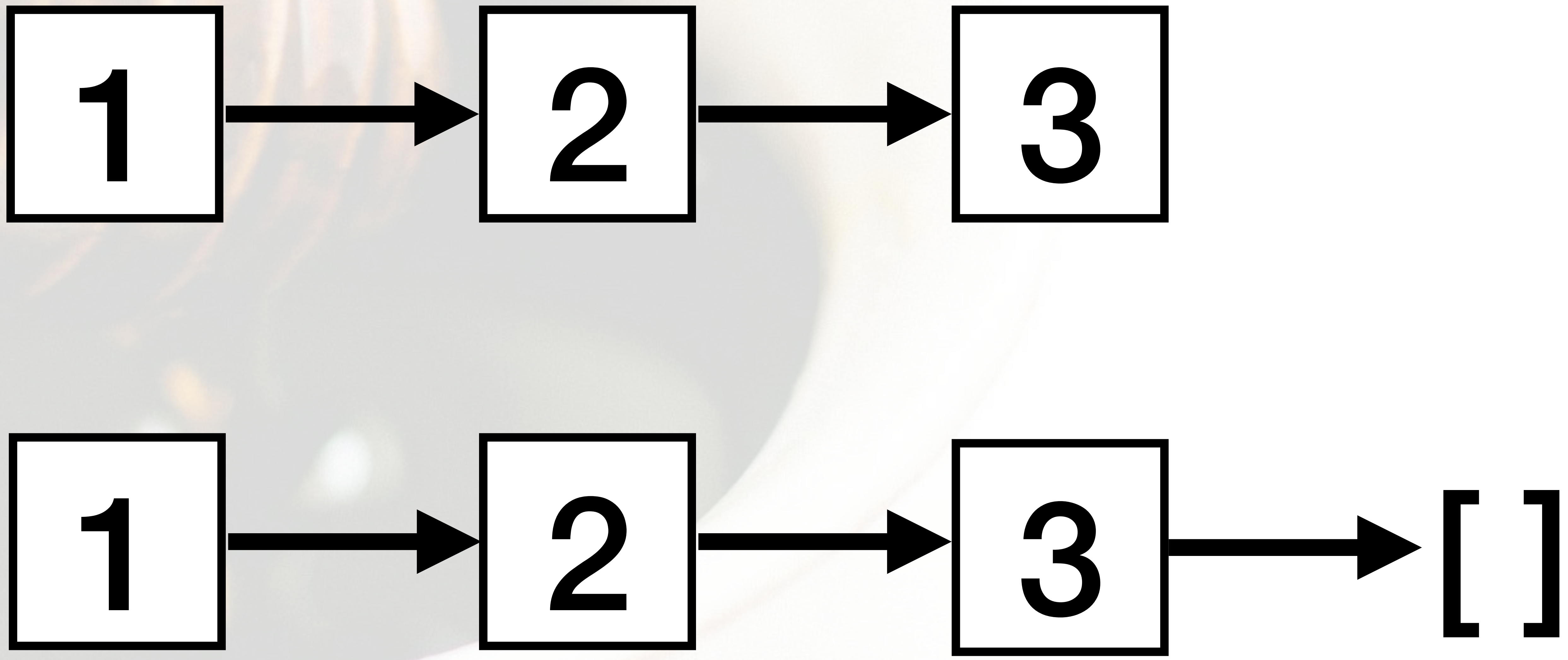
1	2	3	0	5

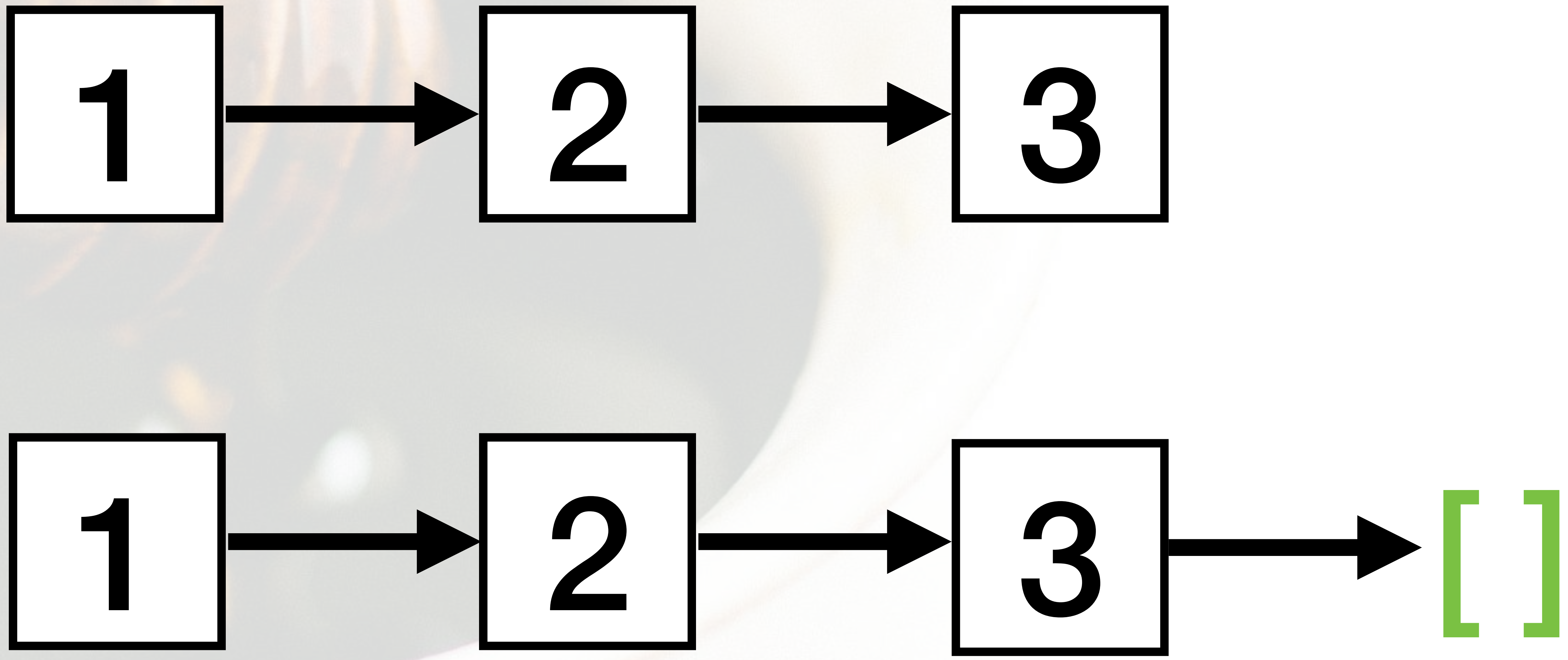
FP

a[3] = 0



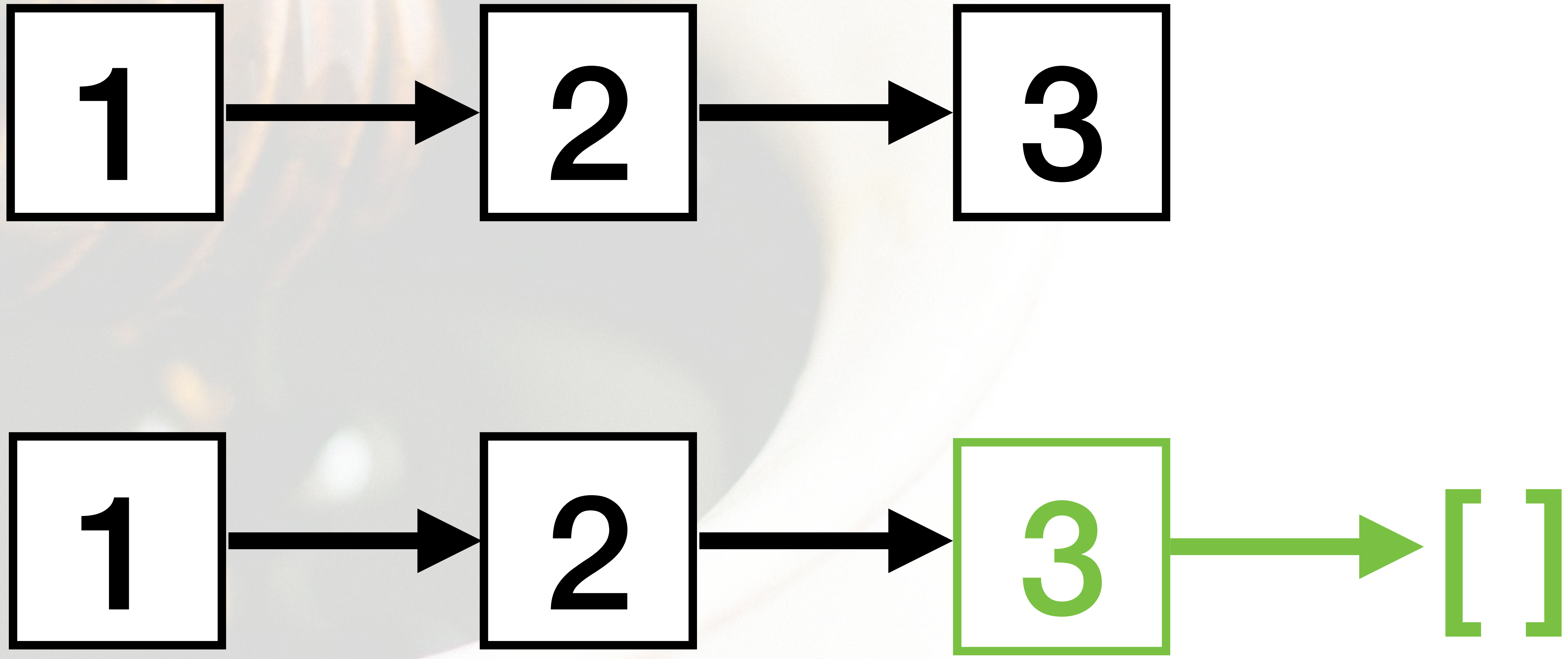
```
List.replace_at([1, 2, 3, 4, 5], 2, 0)
```


1
2
3
1
2
3
[ ]



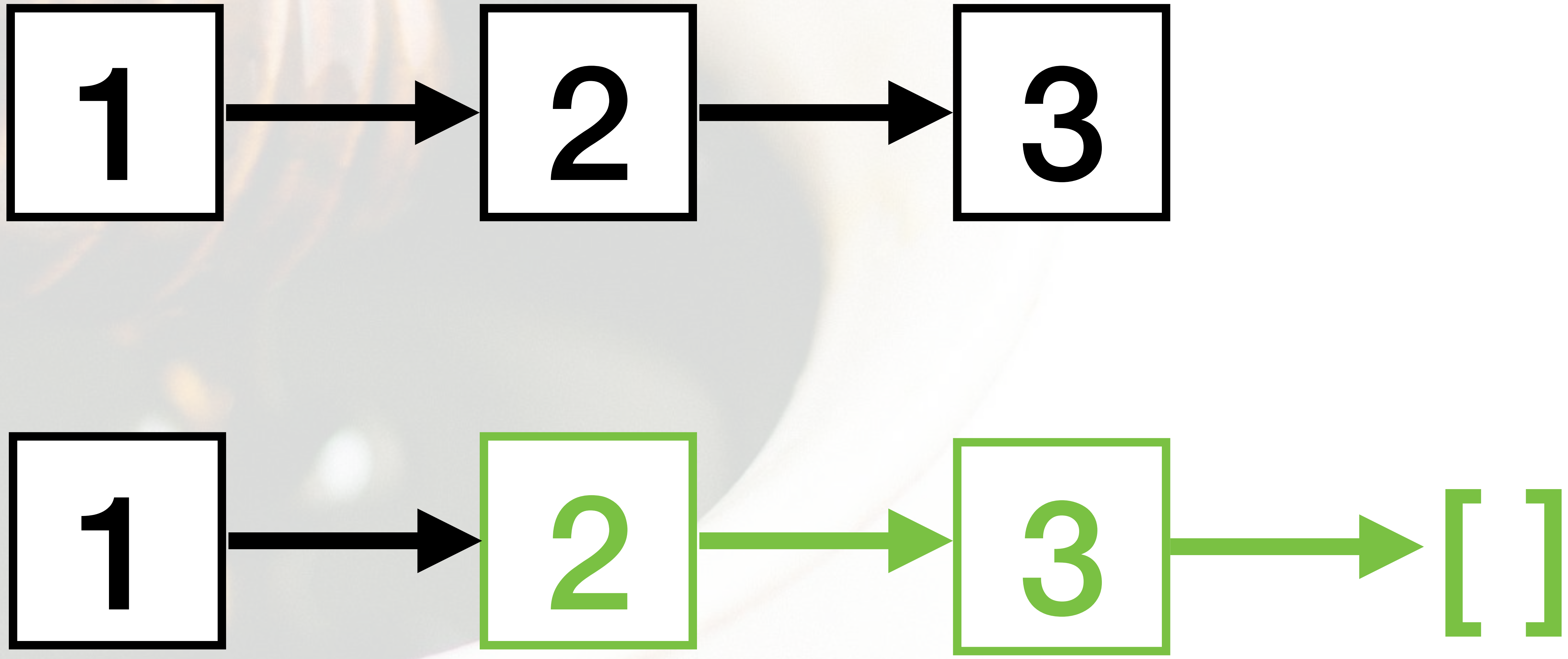
1 2 3

1 2 3 [ ]



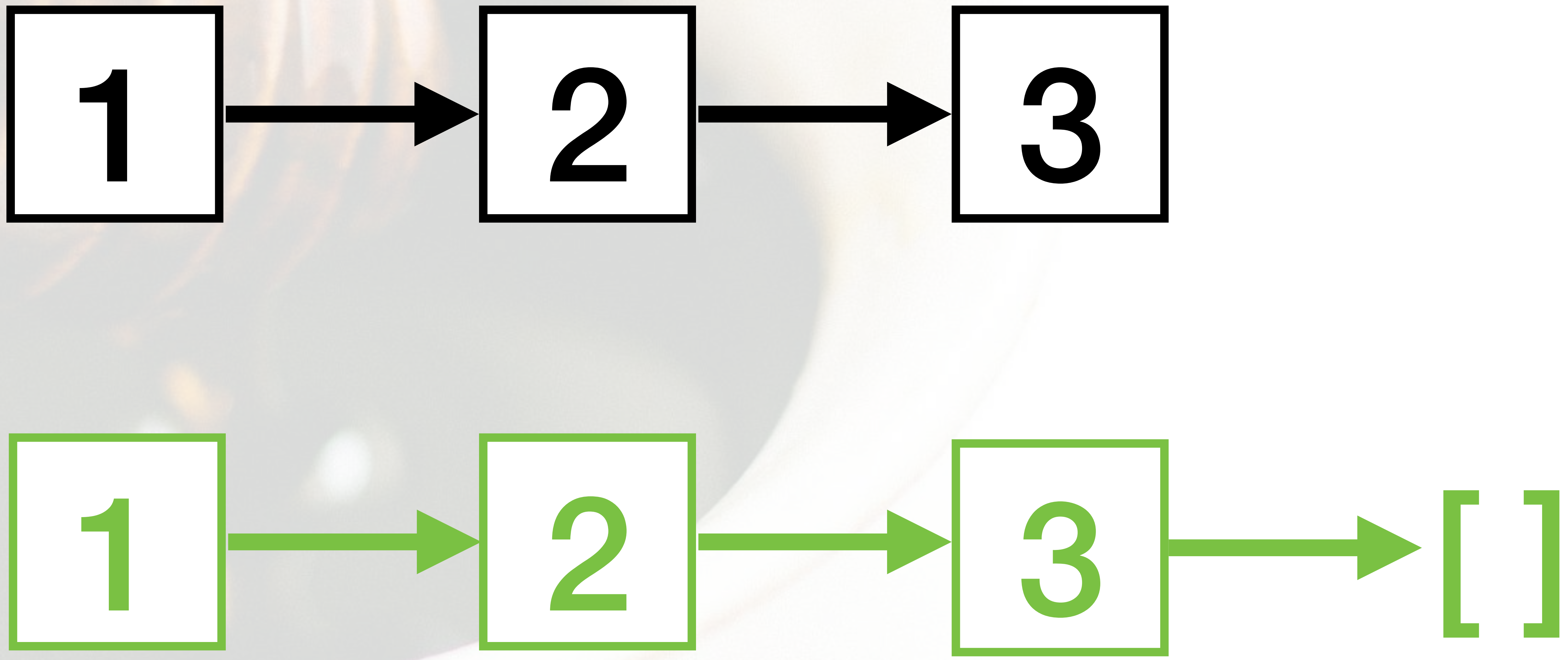
1 2 3

1 2 3 [ ]



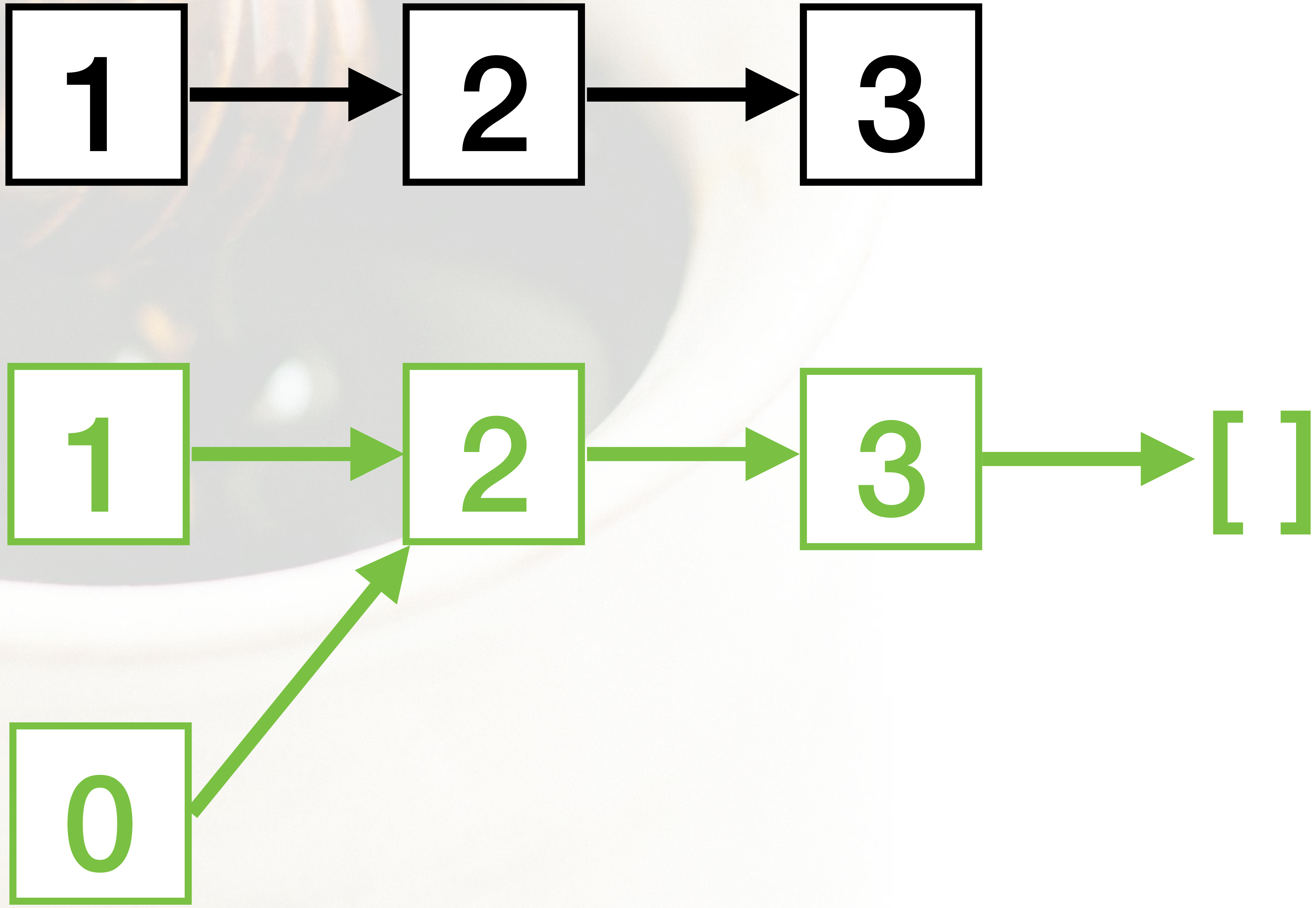
1 → 2 → 3

1 → 2 → 3 → [ ]



1 → 2 → 3

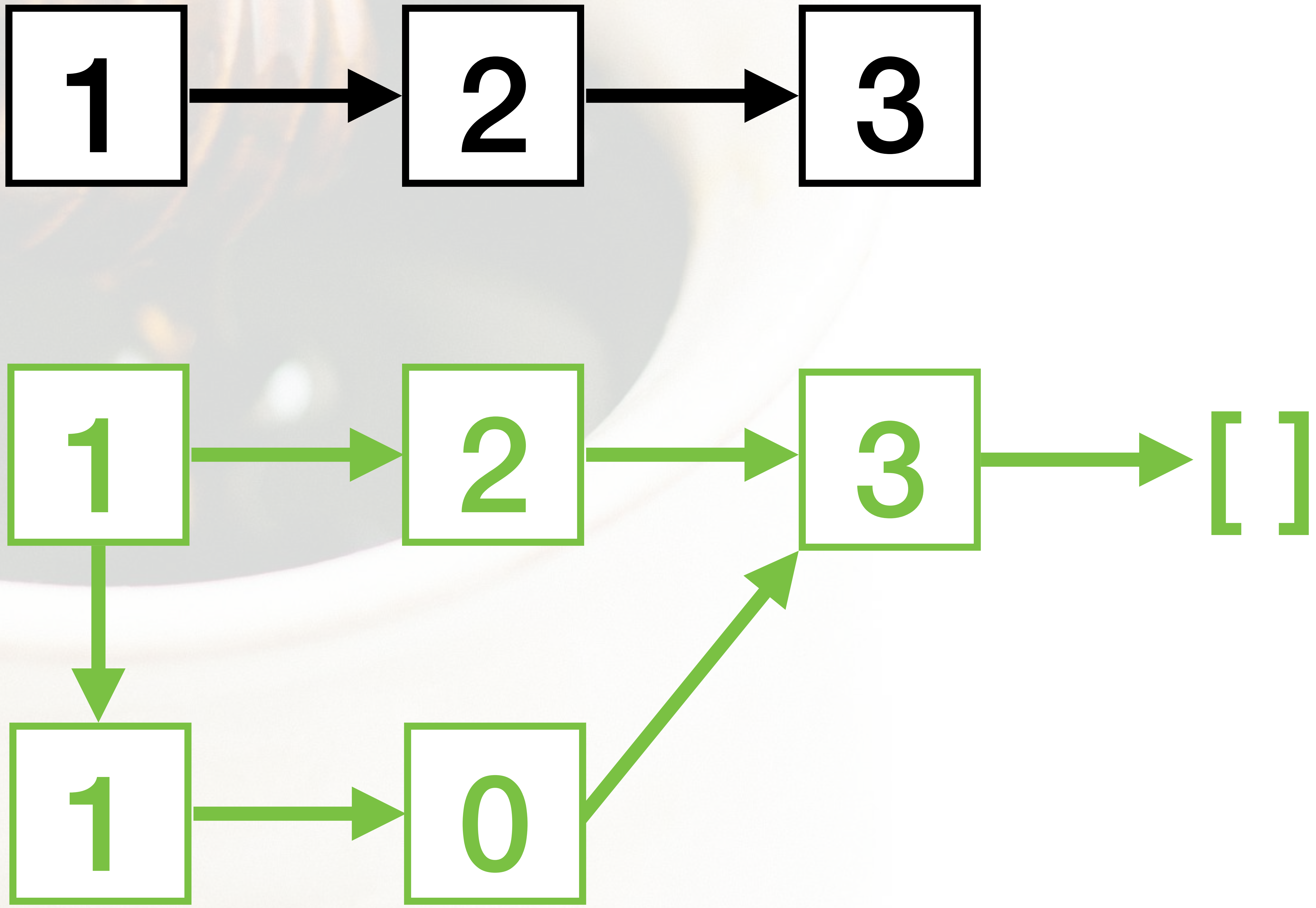
1 → 2 → 3 → [ ]



1 2 3

1 2 3 [ ]

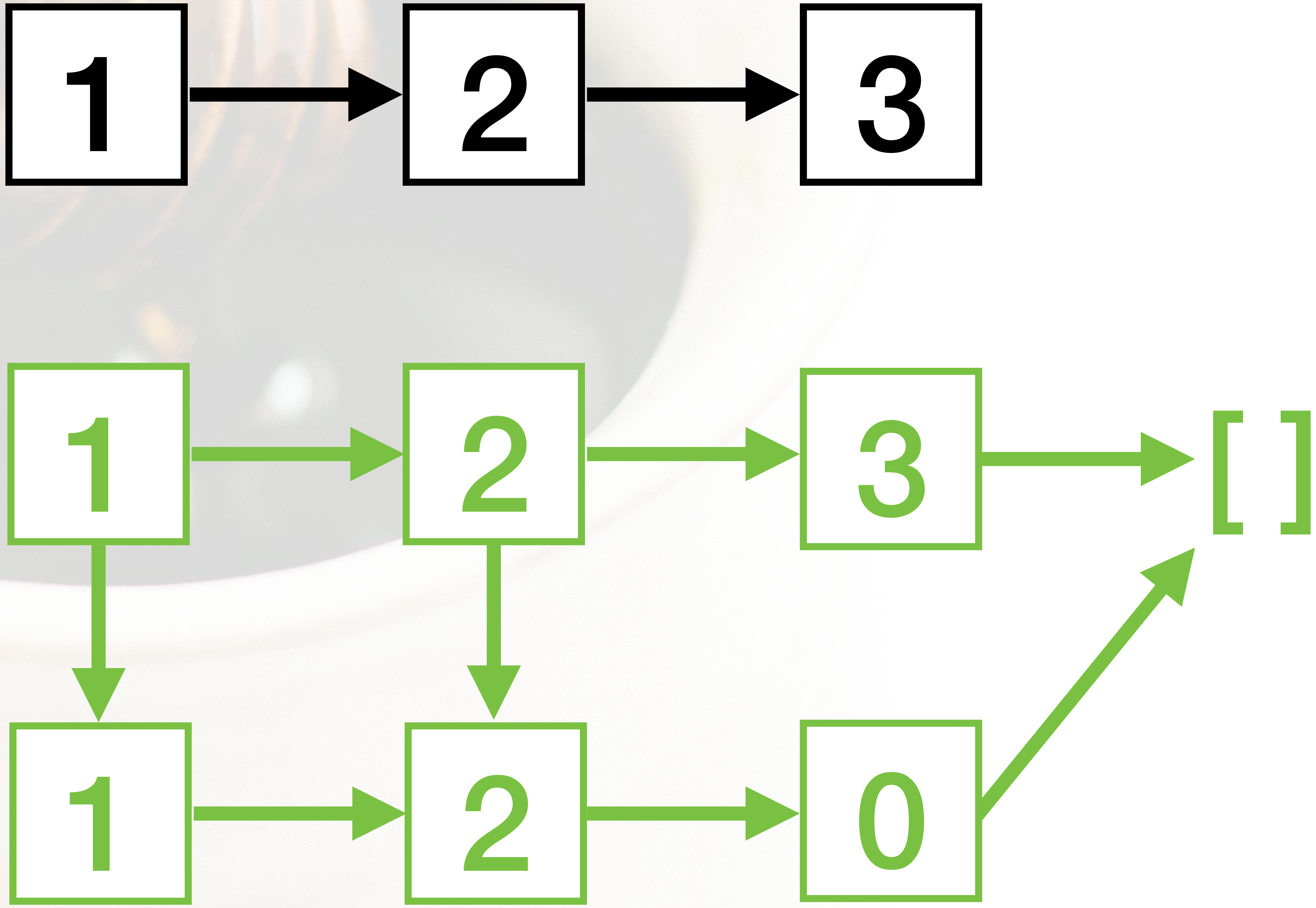
0



1 2 3

1 2 3 [ ]

1 0


1
2
3
1
2
3
[ ]
1
2
0

Embrace Immutability

Bank:

- Balance

Bank:

- Balance

+ Transactions

CRUD

create read update delete

CR~~UD~~

create read ~~update delete~~

Harmony

Fun

Functions

Happy Place

The Core

Pipe-land

The Core

Pipe-land

The Core

Certainty

Pipe-land

Sanitized

The Core

Certainty

Pipe-land

Sanitized

The Core

Certainty

Valid

Pipe-land

Sanitized

The Core

Certainty

Valid

Library

Purity is not the Goal

Purity

:random.uniform

is not the

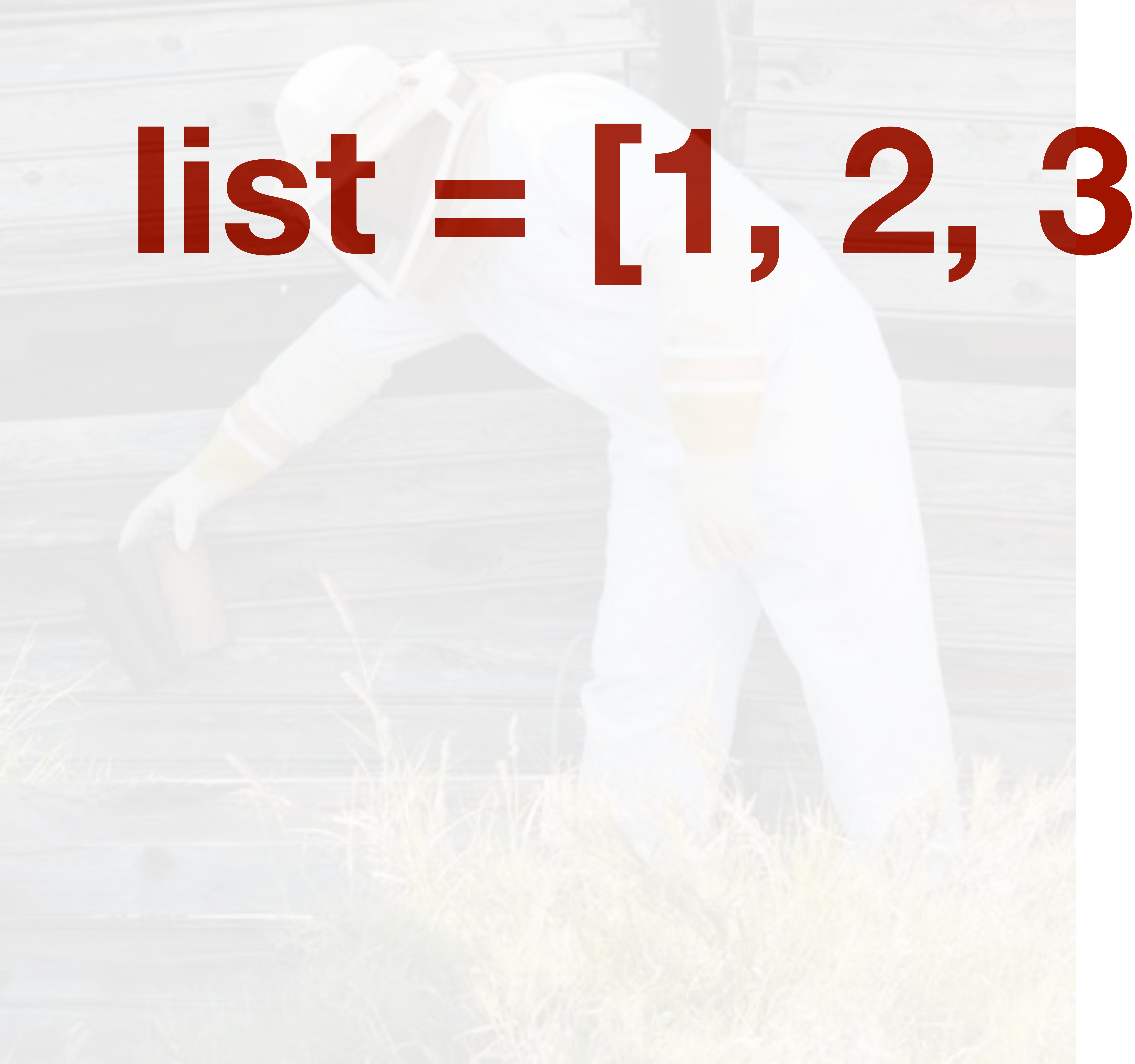
Goal

Purity is not the Goal

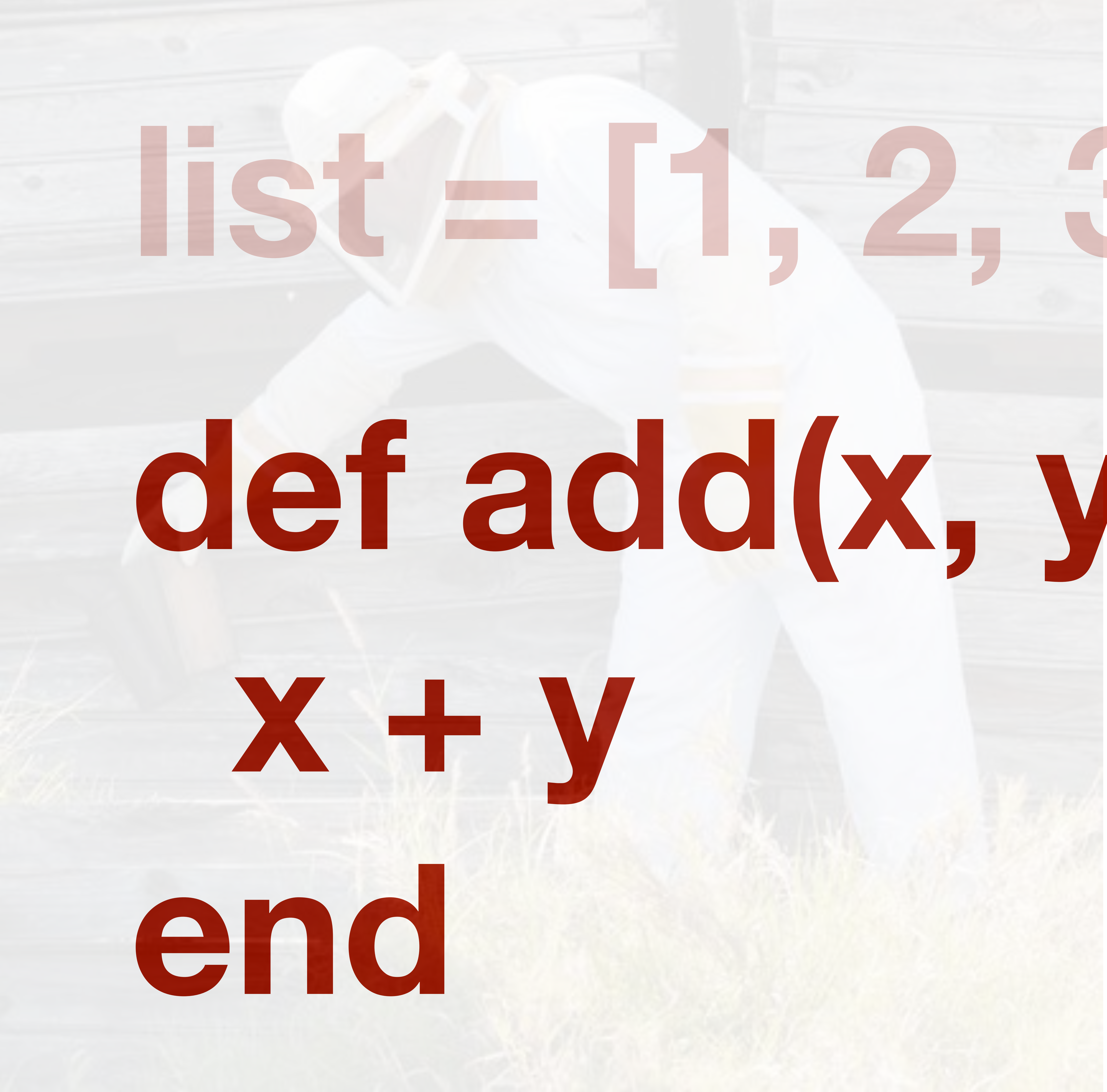
```
:random.uniform
:DateTime.utc_now()
```

Abstractions are the Goal

Reducers

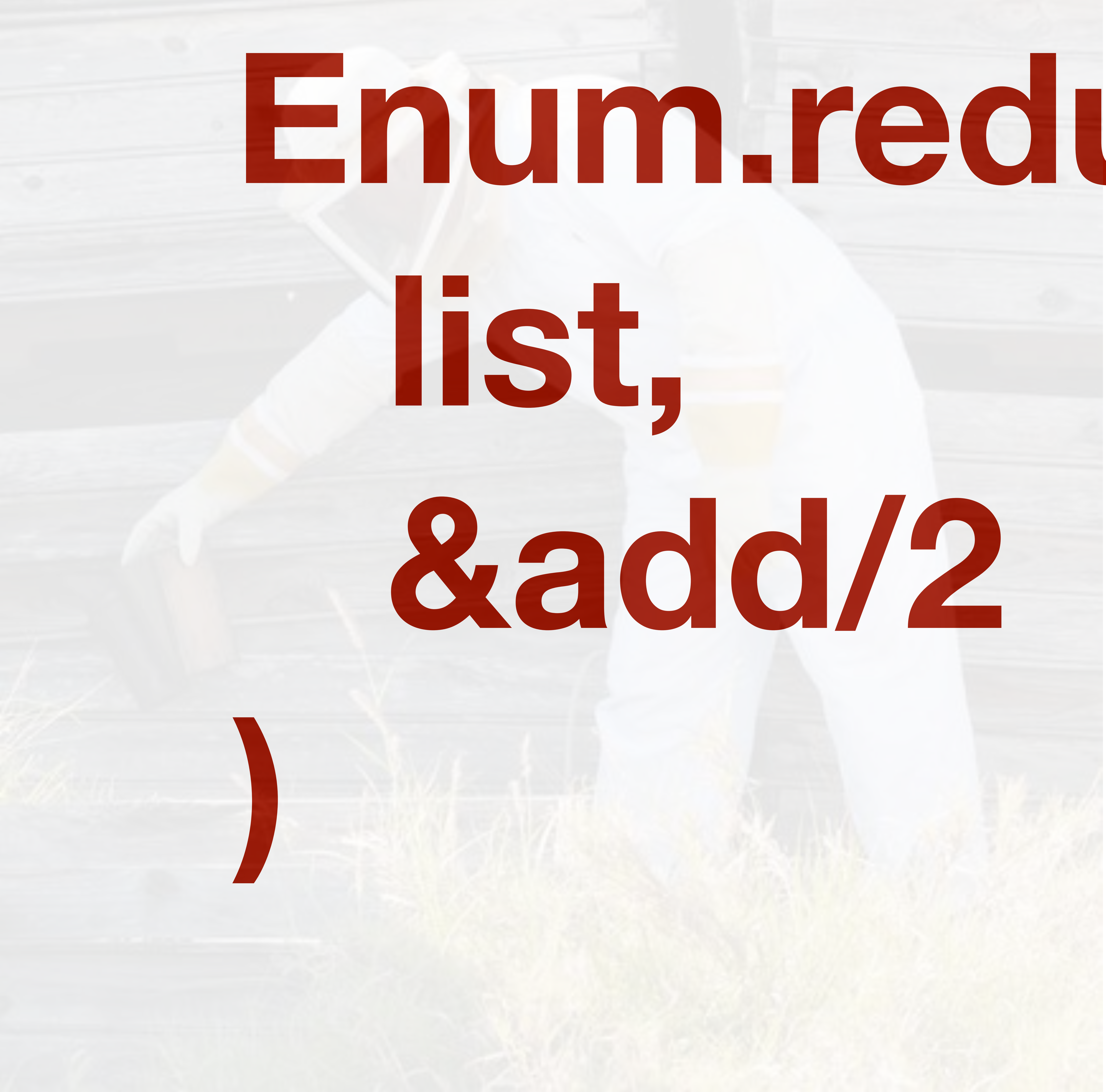


```
list = [1, 2, 3]
```

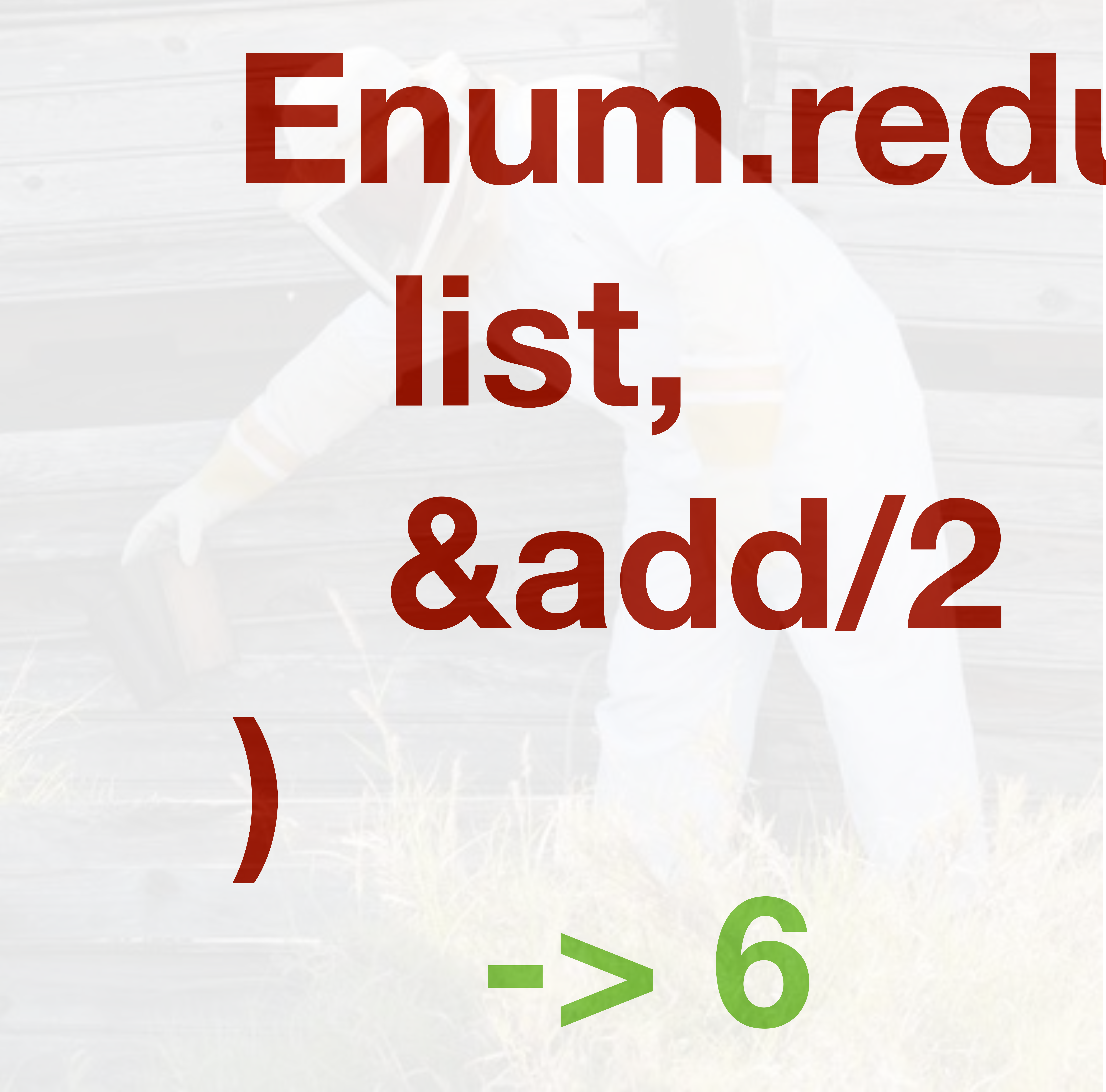


```
list = [1, 2, 3]

def add(x, y) do
  x + y
end
```

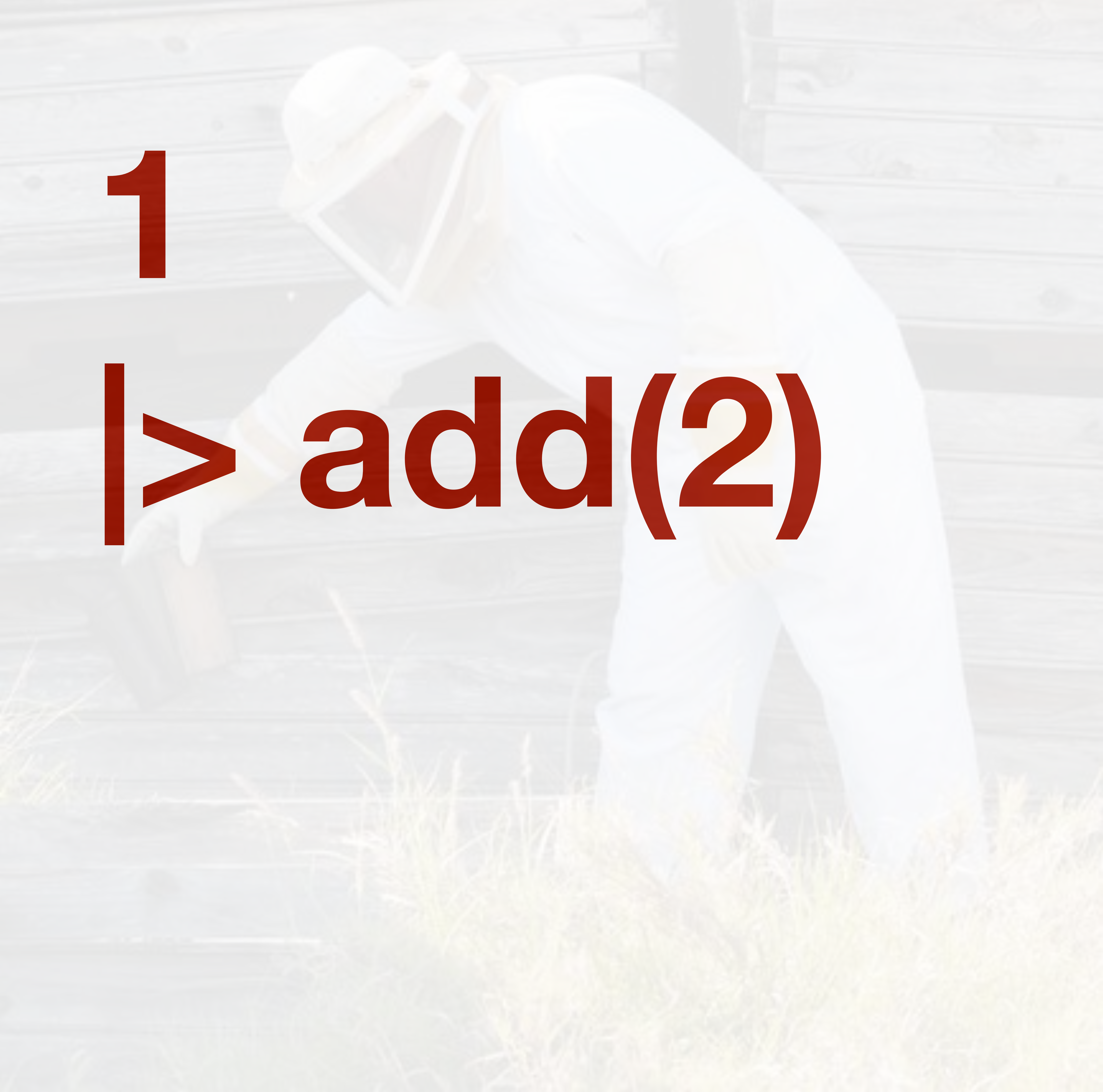


```
Enum.reduce(
  list,
  &add/2
)
```

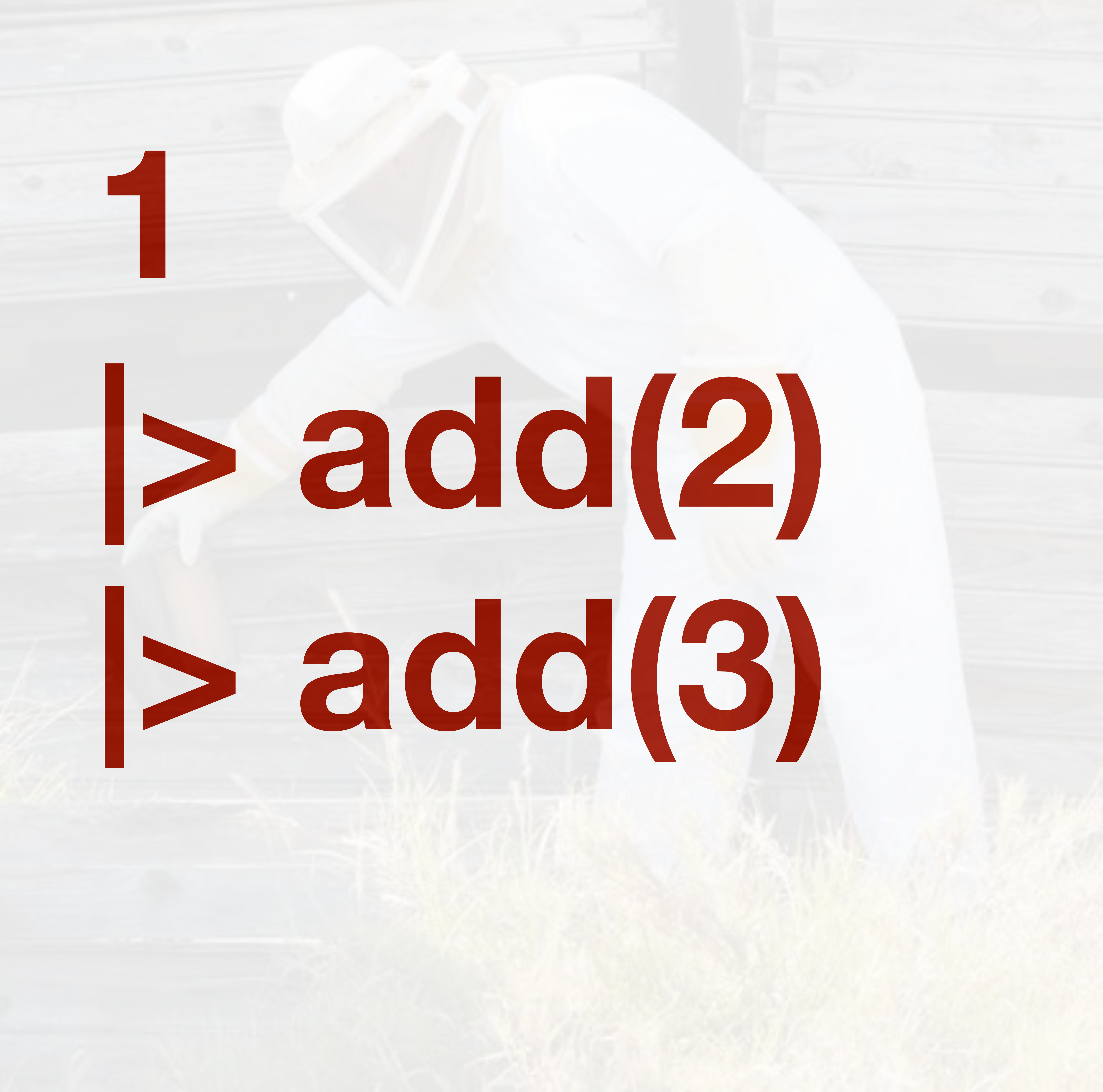


```
Enum.reduce(
  list,
  &add/2
)
    -> 6
```

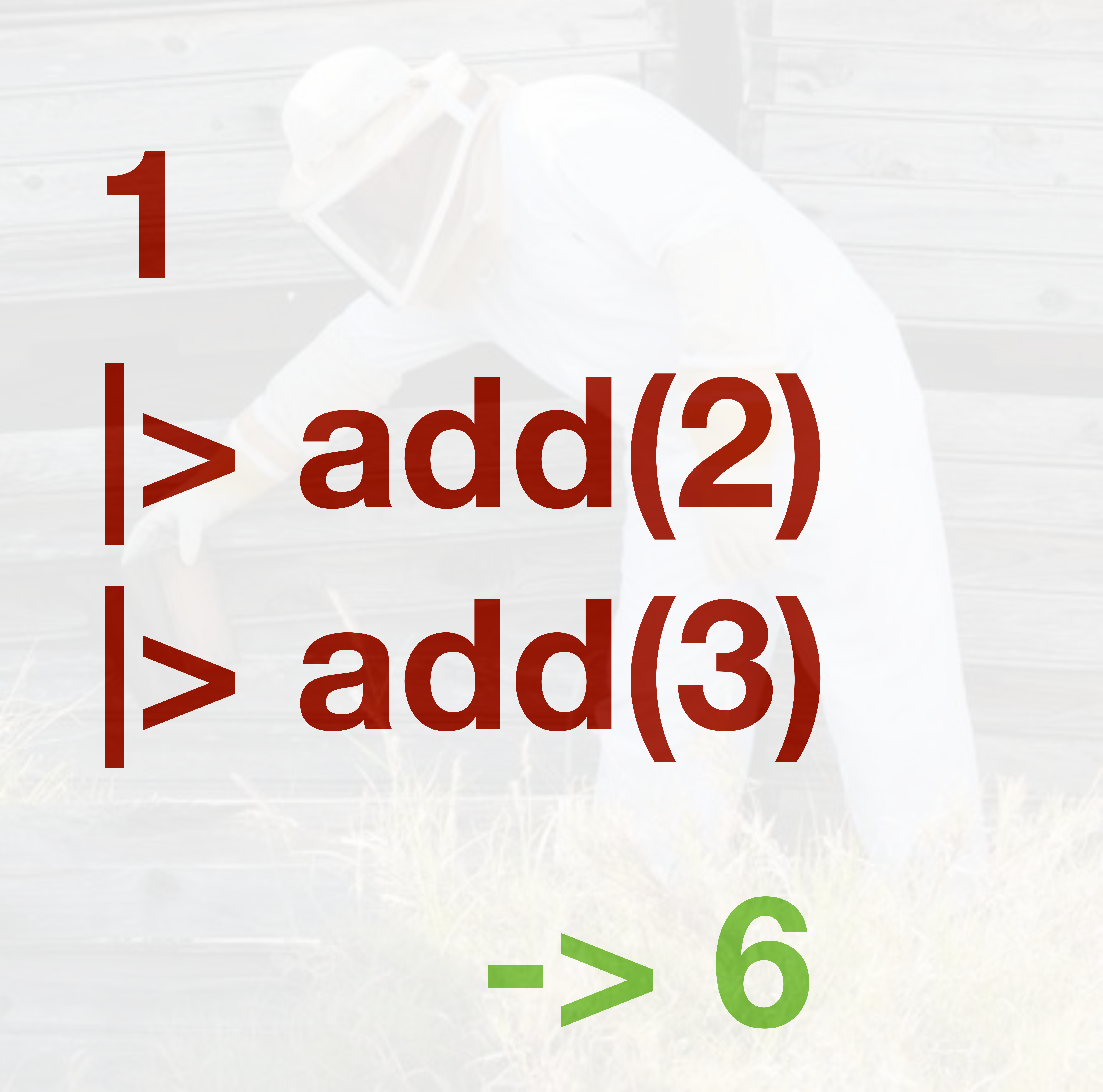
1



```
1
|> add(2)
```



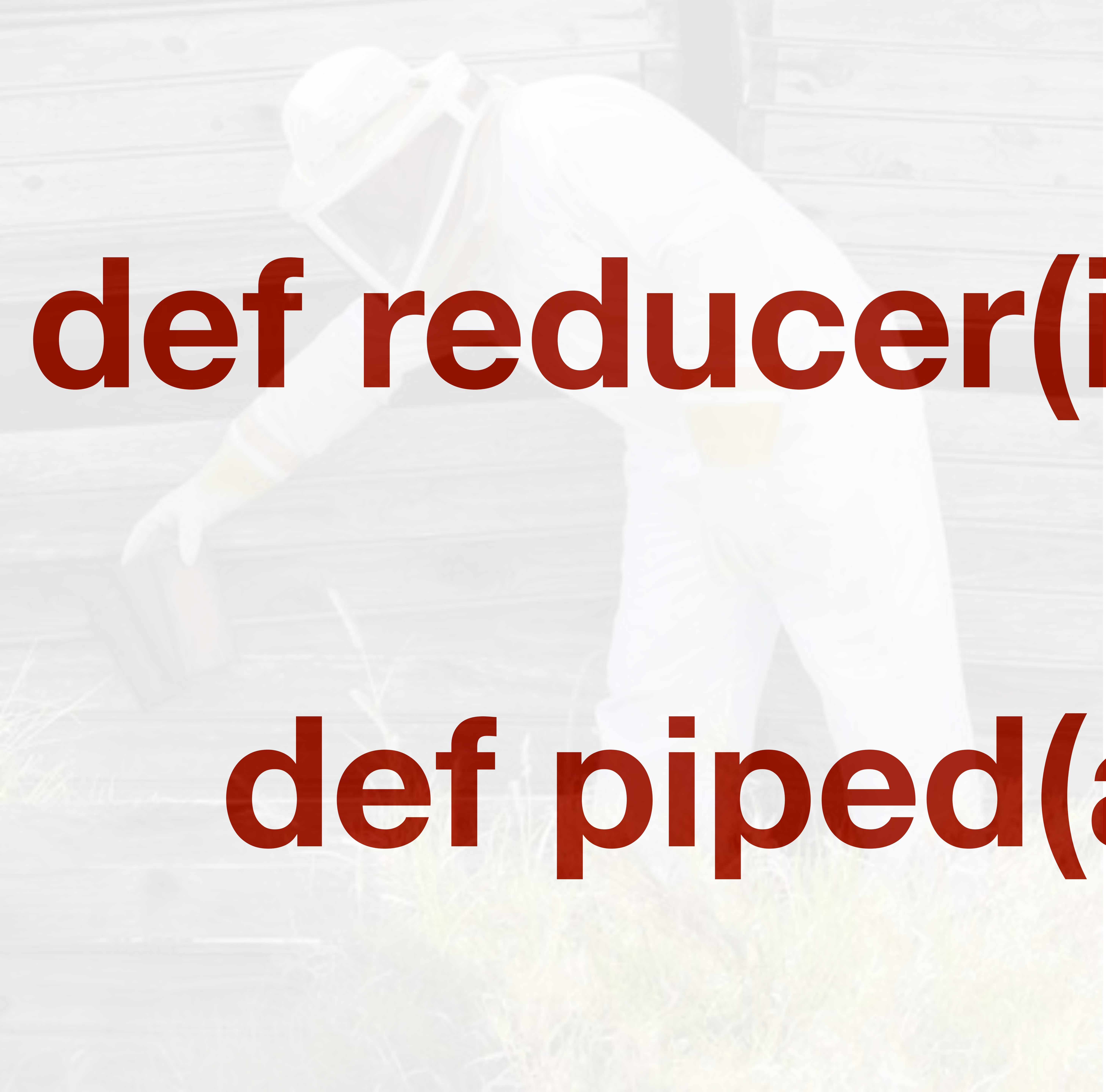
```
1
|> add(2)
|> add(3)
```



```
1
|> add(2)
|> add(3)
```

-> 6

I cheated…



```
def reducer(item, acc)
def piped(acc, item)
```

Happy Place

Things

Tests

Predictable

Test the Core

Property Based Test

```
non_empty( list integer() )
▷ forall( fn list →
  biggest(list) = list ▷ sort ▷ last
end)
```

Generator

```
non_empty( list integer() )
▷ forall( fn list →
  biggest(list) = list ▷ sort ▷ last
end)
```

Property

```
non_empty( list integer() )
▷ forall( fn list →
    biggest(list) = list ▷ sort ▷ last
end)
```

```
non_empty( list integer() )
▷ forall( fn list →
  biggest(list) = list ▷ sort ▷ last
end)
```

Props to…

https://jeffkreeftmeijer.com/mix-proper/

Jeff Kreeftmeijer

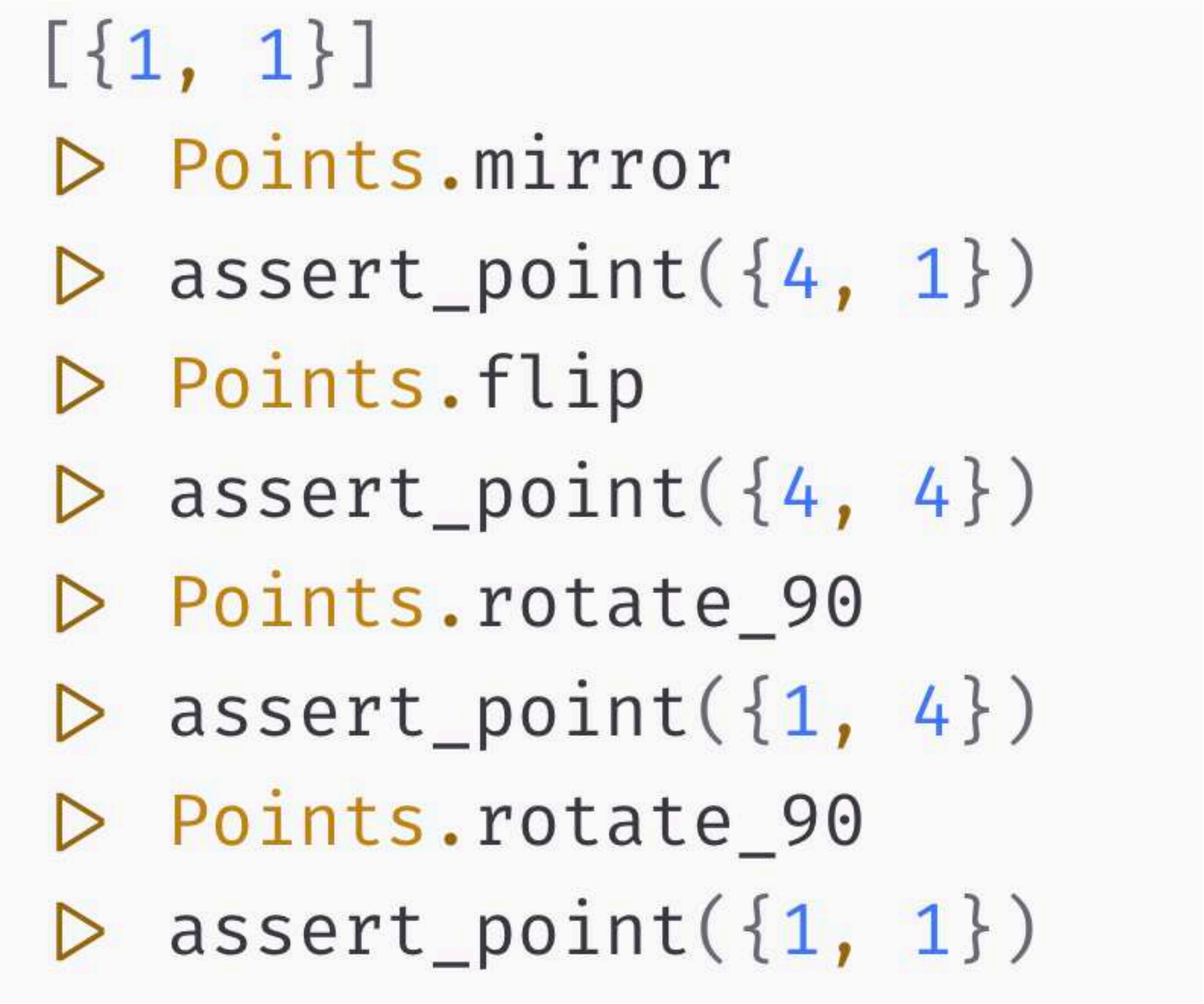
Props to…

John Hughes

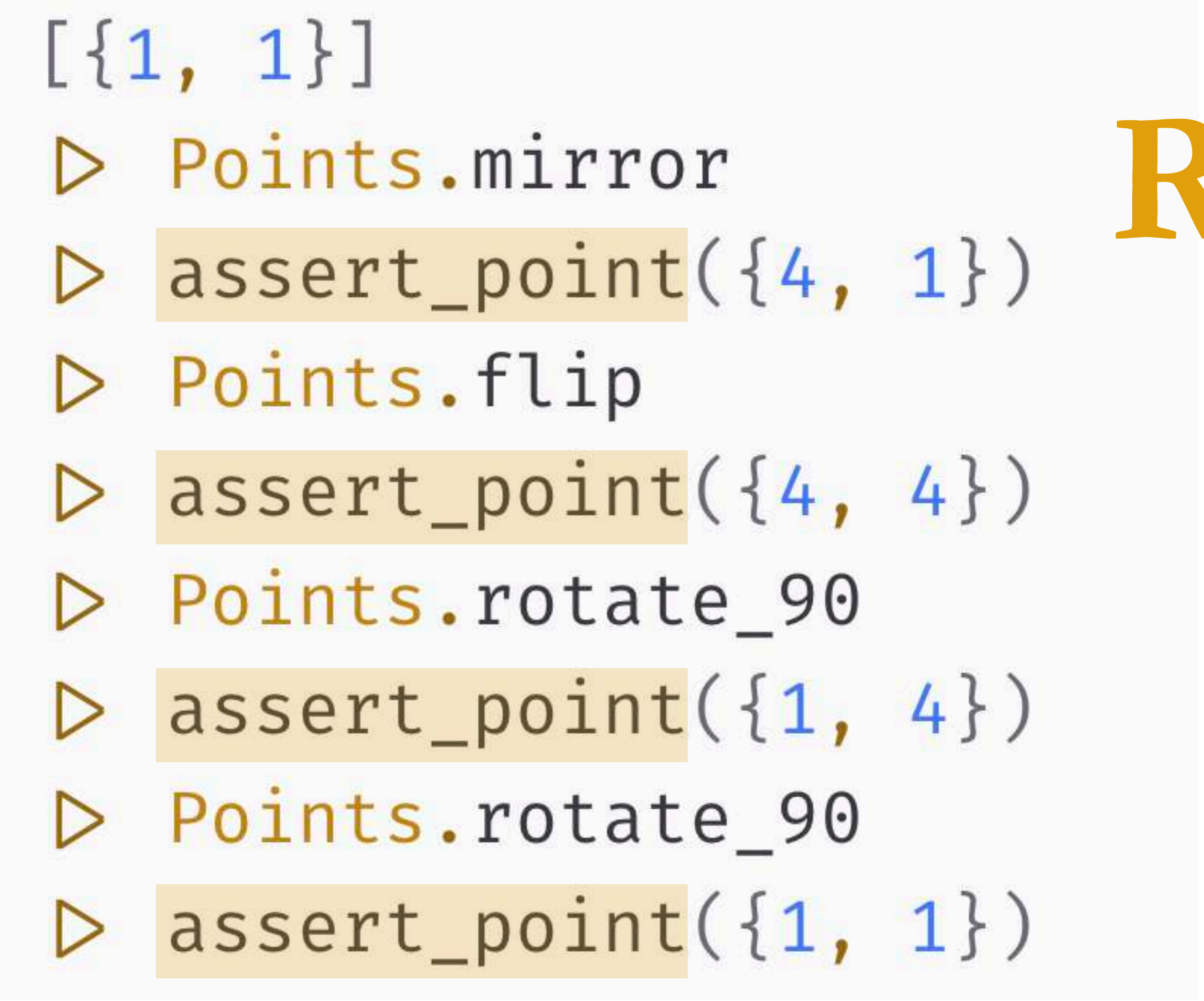
Props to…

Fred Hebert

Pipes in Tests



```
[{1, 1}]
▷ Points.mirror
▷ assert_point({4, 1})
▷ Points.flip
▷ assert_point({4, 4})
▷ Points.rotate_90
▷ assert_point({1, 4})
▷ Points.rotate_90
▷ assert_point({1, 1})
```



```
[{1, 1}]
▷ Points.mirror
▷ assert_point({4, 1})
▷ Points.flip
▷ assert_point({4, 4})
▷ Points.rotate_90
▷ assert_point({1, 4})
▷ Points.rotate_90
▷ assert_point({1, 1})
```

Reducer with side effect

```elixir
def assert_point([actual], expected) do
  assert actual == expected
  [actual]
end
```

```
def assert_point([actual], expected) do
  assert actual == expected
  [actual]
end
```

```
def assert_point([actual], expected) do
  assert actual == expected
  [actual]
end
```

```
[{1, 1}]
▷ Points.mirror
▷ assert_point({4, 1})
▷ Points.flip
▷ assert_point({4, 4})
▷ Points.rotate_90
▷ assert_point({1, 4})
▷ Points.rotate_90
▷ assert_point({1, 1})
```

Streams for Impure

```elixir
def eventually_match(generators, number) do
  Stream.repeatedly(fn ->
    build_question(generators: generators)
    .substitutions
  end)
  |> Enum.find(fn substitution ->
    Keyword.fetch!(substitution, :left) == number
  end)
end
```

```elixir
def eventually_match(generators, number) do
  Stream.repeatedly(fn ->
    build_question(generators: generators)
    .substitutions
  end)
  |> Enum.find(fn substitution ->
    Keyword.fetch!(substitution, :left) == number
  end)
end
```

```elixir
def eventually_match(generators, number) do
  Stream.repeatedly(fn ->
    build_question(generators: generators)
    .substitutions
  end)
  |> Enum.find(fn substitution ->
    Keyword.fetch!(substitution, :left) == number
  end)
end
```

Predictable

Designing Elixir Systems with OTP

Write Highly Scalable,
Self-Healing Software with Layers

"Do Fun Things
with
Big, Loud
Worker-Bees"

by James E Gray, II
and Bruce A. Tate

Big

Boundaries

Maybe

Boundaries

Not Just OTP

Boundaries

Not Just OTP

Boundaries

Complexity

Uncertainty

Not Just OTP

Unsanitized

Process

Boundaries.

Complexity

Uncertainty

With Land

```
def build_quiz(fields) do
  with :ok <- QuizValidator.errors(fields),
    :ok <- GenServer.call(...) do
      :ok
  else
    error ->
      error
  end
end
```

But I want…

More Pipes

More Reducers

More Happiness

?

Skinny Handlers

```
def handle_event("keydown", %{"key" => "Arro
  {:noreply, move(:right, socket)}
end
def handle_event("start", _, socket) do
  {:noreply, new_game(socket)}
end
```

Represent Data as Errors

```elixir
def work(n) do
  if :rand.uniform(10) == 1 do
    raise "Oops!"
  else
    {:result, :rand.uniform(n * 100)}
  end
end
```

```
def make_work_safe(dangerous_work, arg) do
  try do
      apply(dangerous_work, [arg])
  rescue
    error ->
    {:error, error, arg}
  end
end
```

```
def stream_work do
  Stream.iterate(1, &(&1 + 1))
  |> Stream.map(fn i ->
       make_work_safe(&work/1, i)
     end)
end
```

Error Data Composes

Maybe

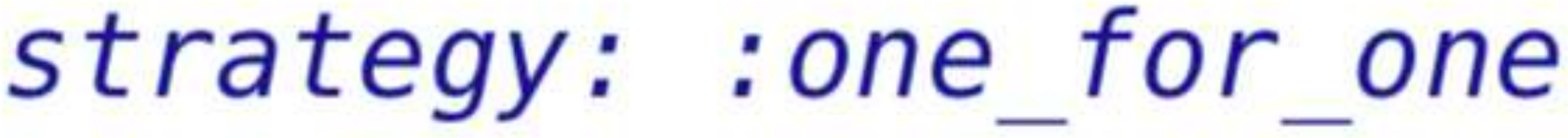
Loud

Lifecycles

Paradigm

```
def start(_type, _args) do
  children = [
QuizManager.lookup_quiz_by_title
```

```
Supervisor.start_link(children, opts)
```



```
strategy: :one_for_one
```

Clean Startup

Clean Startup

+ Clean Shutdown

Clean Startup

+ Clean Shutdown

+ Supervision

OTP goodness

Paradigm

WorkerBees

Workers

Balance

Imbalance

→ boundary

Imba~~l~~ance

→ boundary

Imbalance

core ←

Delay

Persistence

Decision

```
def answer_question(name, answer,
    persistence_fn \\ @persistence_fn)
```

```elixir
def handle_call({:answer_question, answer, fun}
  fun = fun || fn r, f -> f.(r) end
  response = Response.new(quiz, email, answer)
  fun.(response, fn r ->
    quiz
    |> Quiz.answer_question(r)
    |> Quiz.select_question
  end)
  |> maybe_finish(email)
end
```

Balance

Designing Elixir Systems with OTP

Write Highly Scalable, Self-Healing Software with Layers

"Do Fun Things with Big, Loud Worker-Bees"

by James E Gray, II and Bruce A. Tate

Designing Elixir Systems with OTP

Write Highly Scalable, Self-Healing Software with Layers

"Do Fun Things with Big, Loud Worker-Bees"

by James E Gray, II
and Bruce A. Tate

James E.
Gray II

?