
catch
BOX

BC/NAV TECH DAYS 2023
mibuso.com

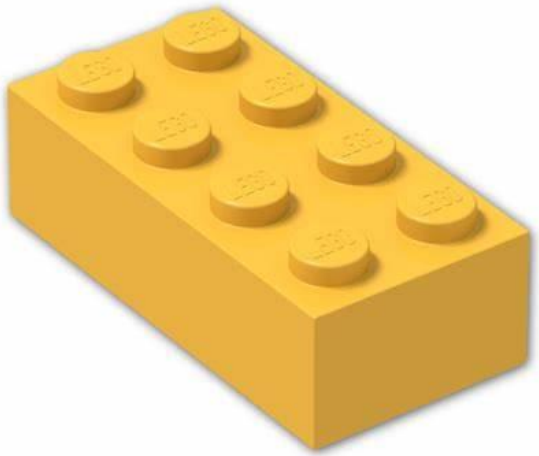
Interfaces – an Intro

BC/NAV TECH DAYS 2023
mibuso.com

Componentization

Componentization

Define a component



Componentization

Describe the purpose

Componentization

Facilitates Clear Architecture and Interactions

Componentization

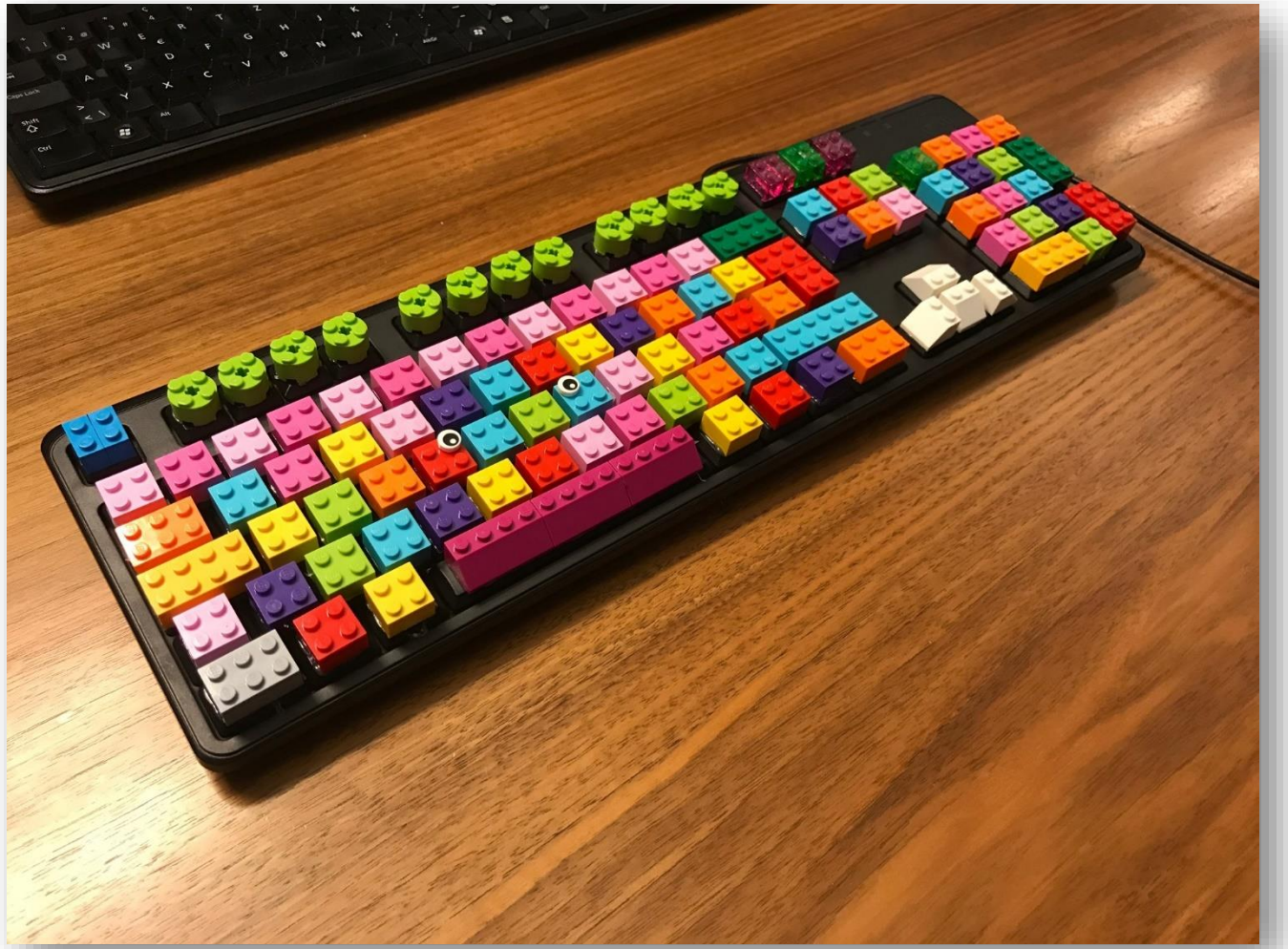
Replace a component
(decoupling)

Componentization

Switch implementation

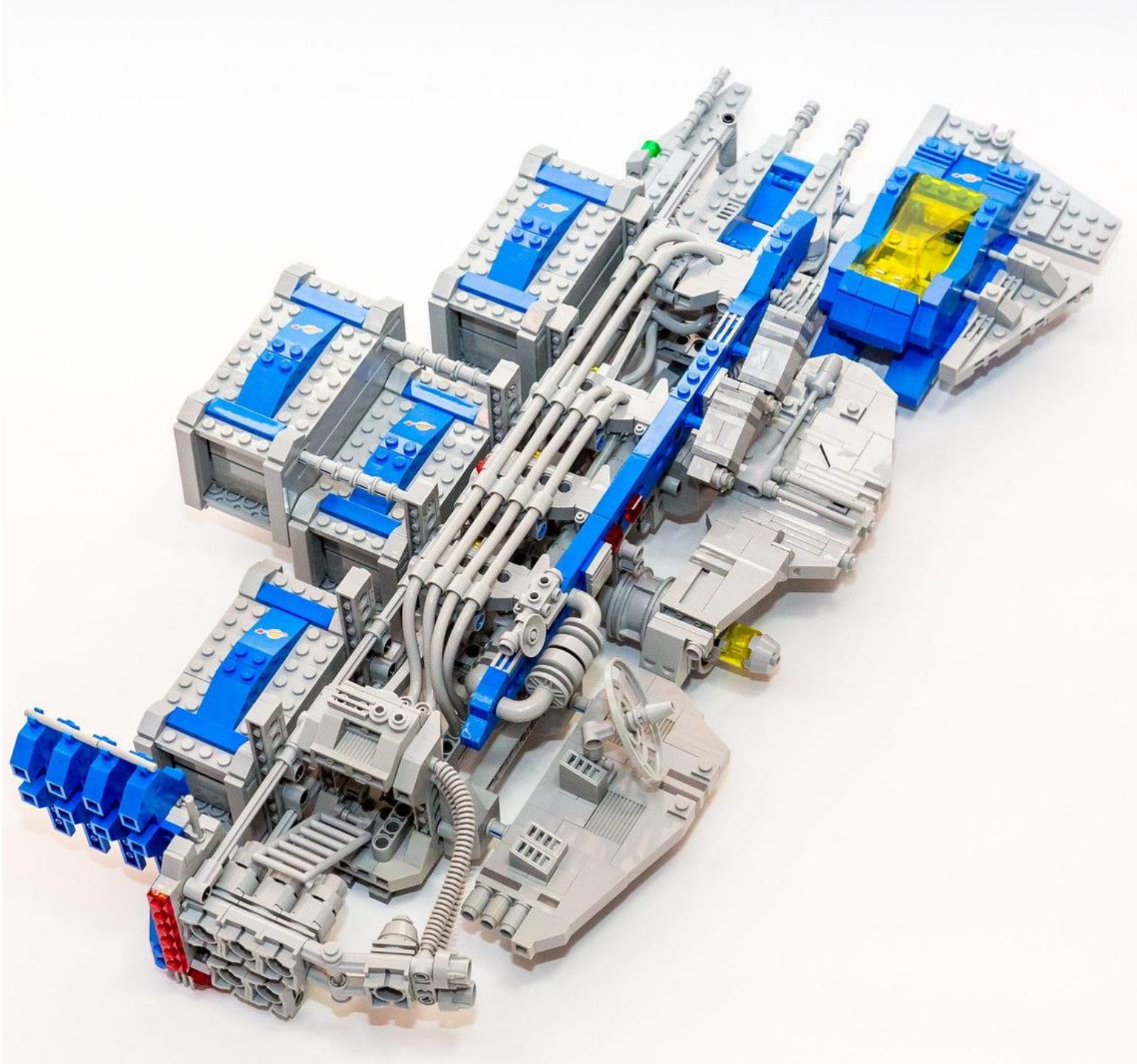
Componentization

Allow Override of Default Code



Componentization

Extend a Component



Componentization

Test a Component

For example...

BC/NAV TECH DAYS 2023
mibuso.com

BC/NAV TECH DAYS 2023
mibuso.com

What is an Interface?

"In programming languages, an interface is a construct that defines a **contract** or a **set of methods** that a class must implement. It specifies the **signature** (name, parameters, return types) of the methods without providing their actual implementation."

```
interface "IScale"
{
    1 reference
    procedure GetWeight(): Decimal;
    0 references
    procedure Tare();
    2 references
    procedure Getinfo(): Text;
}
```

Interface Implementations

```
codeunit 50404 "Scale Foo" implements IScale
{
    0 references
    procedure GetWeight(): Decimal;
    begin
        Randomize();
        Exit(Random(30));
    end;

    0 references
    procedure Tare();
    begin
        //TODO: Implement Tare
    end;

    0 references
    procedure Getinfo(): Text;
    begin
        exit('Info: Scale "Foo"');
    end;

}
```

```
codeunit 50405 "Scale Tefal" implements IScale
{
    0 references
    procedure GetWeight(): Decimal;
    begin
        Randomize();
        Exit(Random(30) + 20);
    end;

    0 references
    procedure Tare();
    begin
        //TODO: Implement Tare
    end;

    0 references
    procedure Getinfo(): Text;
    begin
        exit('Info: Scale "Tefal"');
    end;

}
```

Factory

"an object that the client code can use to obtain instances of these objects, without explicitly knowing the concrete class being instantiated."

```
enum 50400 "Scales" implements IScale
{
    Extensible = true;

    0 references
    value(1; Tefal)
    {
        Caption = 'Tefal';
        Implementation = Iscale = "Scale Tefal";
    }
    0 references
    value(2; Foo)
    {
        Caption = 'Foo';
        Implementation = Iscale = "Scale Foo";
    }
}
```

Factory

"an object that the client code can use to obtain instances of these objects, without explicitly knowing the concrete class being instantiated."

```
enum 50400 "Scales" implements IScale
{
    Extensible = true;

    0 references
    value(1; Tefal)
    {
        Caption = 'Tefal';
        Implementation = Iscale = "Scale Tefal";
    }
    0 references
    value(2; Foo)
    {
        Caption = 'Foo';
        Implementation = Iscale = "Scale Foo";
    }
}
```

```
1 reference
local procedure DoGetWeight(var Item: Record Item; var Handled: Boolean);
var
    ScaleSetup: Record ScaleSetup;
    Scale: Interface IScale;
begin
    ScaleSetup.Get;

    Scale := ScaleSetup.Scale;

    Item.validate(Weight, scale.GetWeight());
    Item.Modify(true);

    if (scale.Getinfo() <> '') then
        message(scale.Getinfo());

end;
```

BC/NAV TECH DAYS 2023
mibuso.com

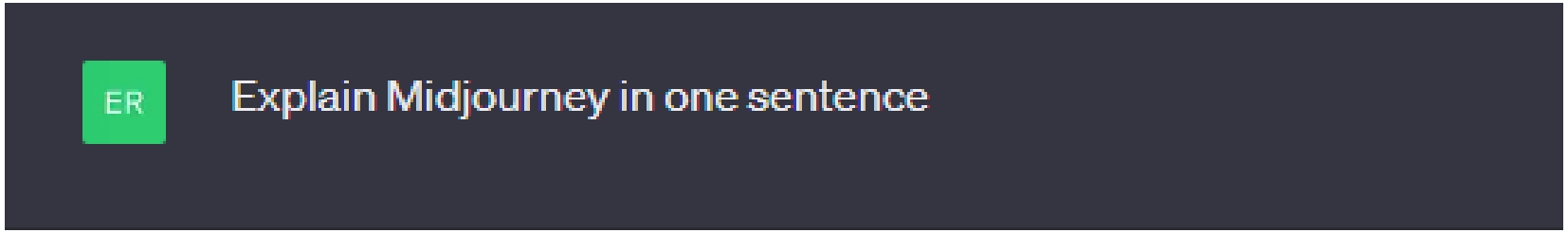
BC/NAV TECH DAYS 2023
mibuso.com

ChatGPT

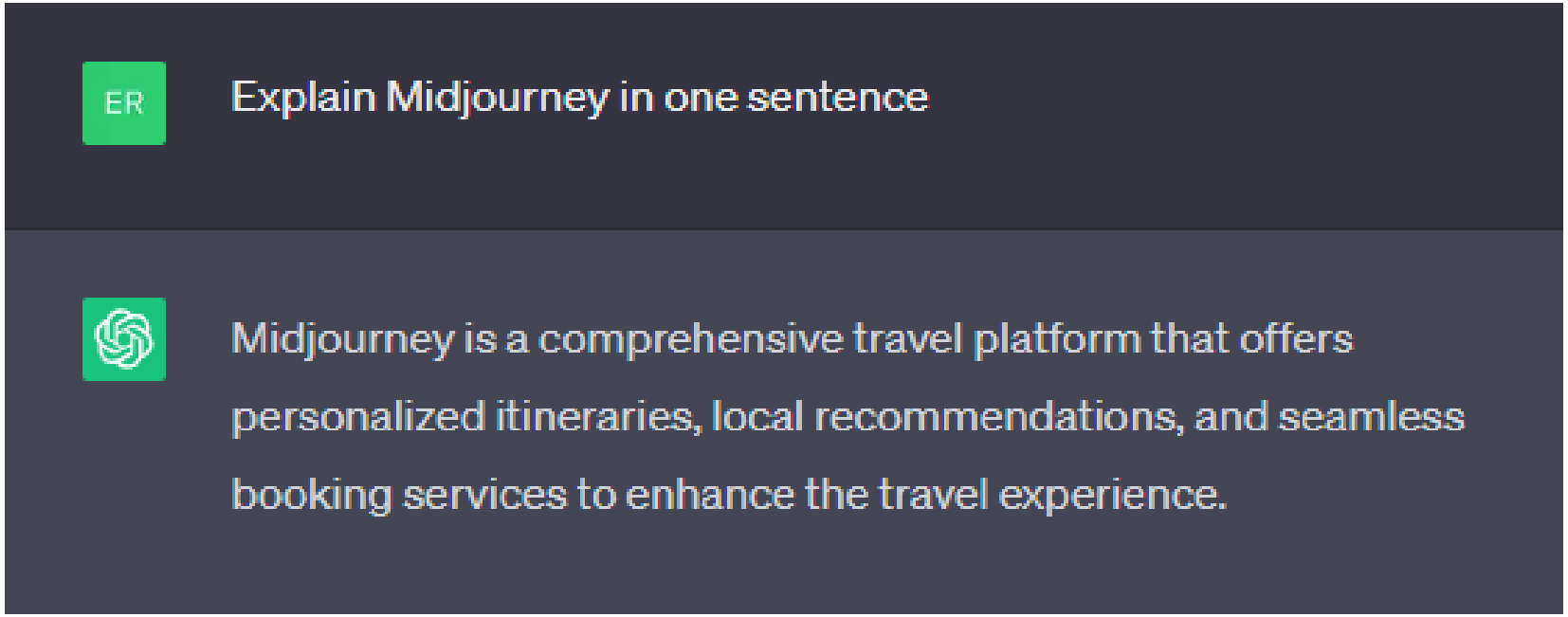
BC/NAV TECH DAYS 2023
mibuso.com

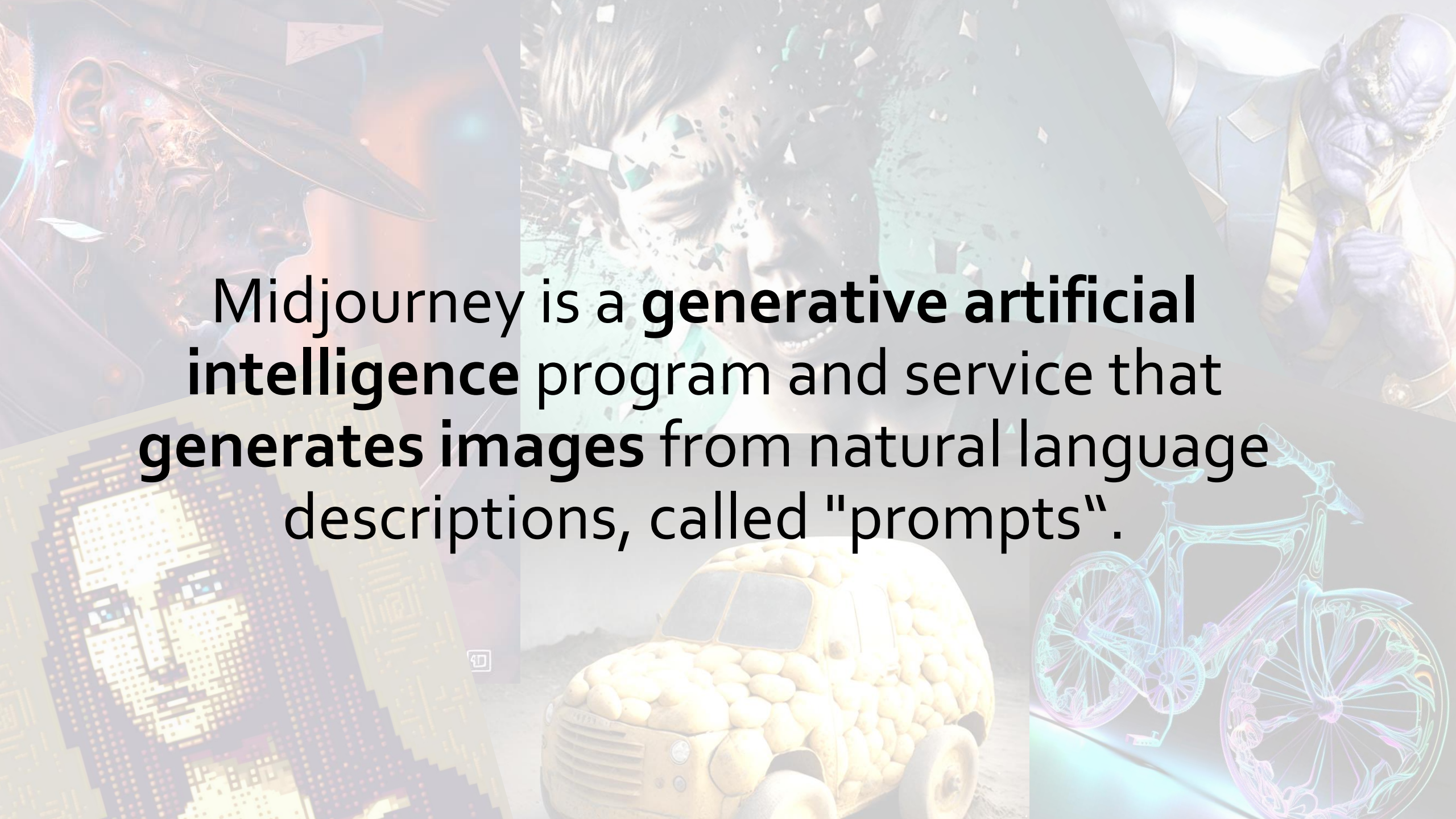
Midjourney

Midjourney



Midjourney


ER
Explain Midjourney in one sentence
Midjourney is a comprehensive travel platform that offers personalized itineraries, local recommendations, and seamless booking services to enhance the travel experience.



Midjourney is a **generative artificial intelligence** program and service that **generates images** from natural language descriptions, called "prompts".

Prompt:
“Imagine: Shocked Potato when seeing Belgian Fries” –ar 1:1

Prompt:
“Imagine: Shocked Potato when seeing Belgian Fries” –ar 1:1

Prompt:
“Imagine: Shocked Potato when seeing Belgian Fries” –ar 1:1

Prompt:
Imagine: <url to picture> hilarious caricature of both persons, drawing --ar 1:1

Prompt:
Imagine: <url to picture> hilarious caricature of both persons, drawing --ar 1:1

So… what if …

Item Card

Saved

1936-S · BERLIN Guest Chair, yellow

Imagine | Home | Request Approval | Item | Prices & Discounts | More options

Copy Item | Create Stockkeeping Unit | Apply Template

Item

Show more

No.	1936-S
Description	BERLIN Guest Chair, yellow
Blocked	
Type	Inventory
Base Unit of Measure	PCS
Item Category Code	CHAIR
Variant Mandatory if E...	Default (No)
Picture URL	https://cdn.discordapp.com/attac...

Inventory

Show more

Shelf No.	D7
Inventory	0
Qty. on Purch. Order	0
Qty. on Prod. Order	0
Qty. on Component Li...	0
Qty. on Sales Order	0
Stockout Warning	Default (Yes)
Unit Volume	0.25
Over-Receipt Code	

Details | Attachments (0)

Picture

Marketing Text

Prompt:
Imagine: <url to picture> on a beach, sunset, realistic --ar 1:1 --iw 1.25

Prompt:
Imagine: <url to picture> on a beach, sunset, realistic --ar 1:1 --iw 1.25

BC/NAV TECH DAYS 2023
mibuso.com

Let’s see this in practice…

Action: Item.Imagine

Imagine With MidJourney Meth (GetImageUrl)

Import Picture to Item table

BC/NAV TECH DAYS 2023
mibuso.com

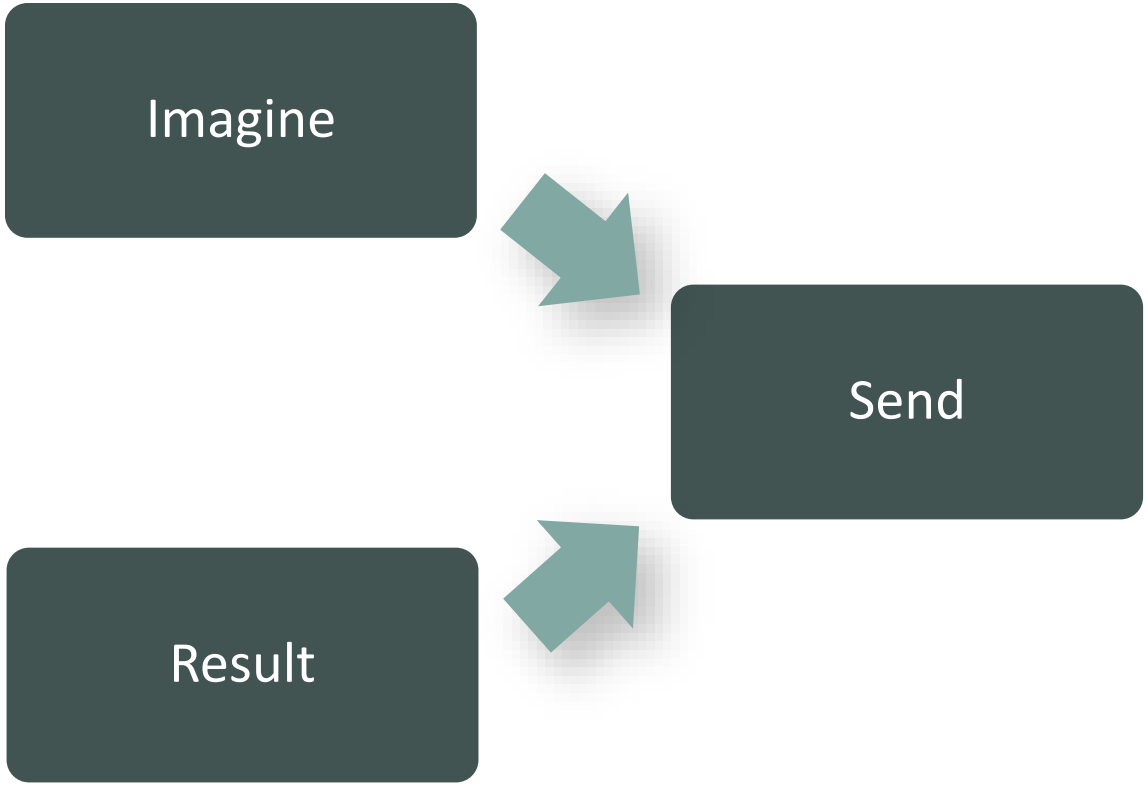
Imagine With MidJourney Meth (GetImageUrl)

Imagine With MidJourney Meth (GetImageUrl)

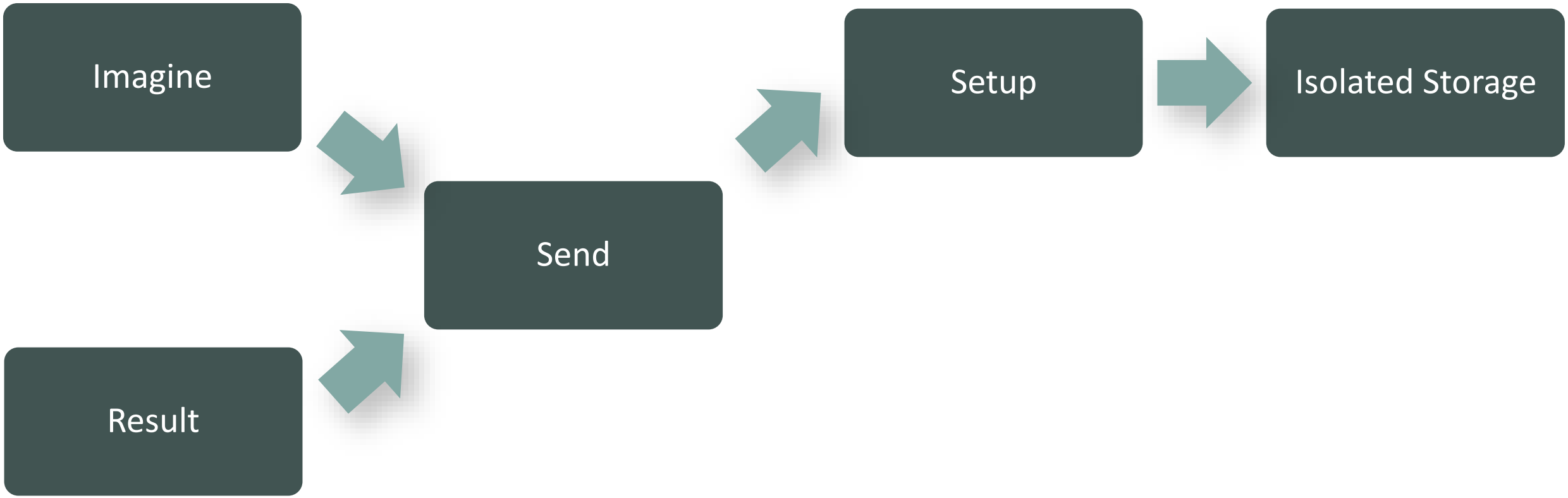
Imagine

Result

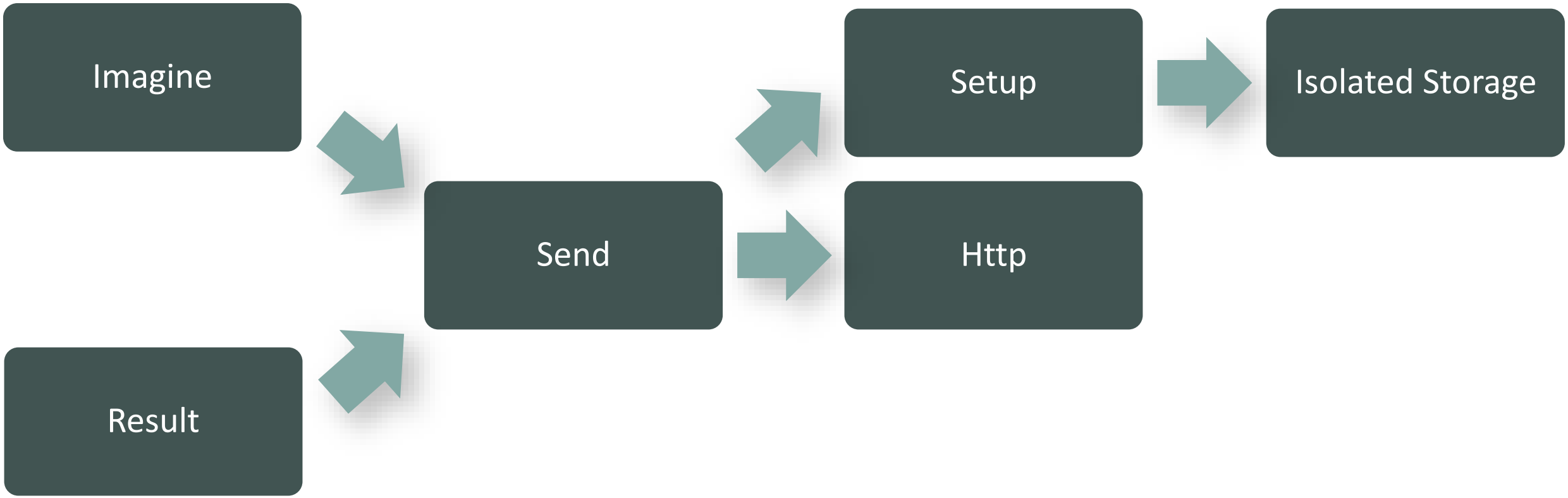
Imagine With MidJourney Meth (GetImageUrl)



Imagine With MidJourney Meth (GetImageUrl)



Imagine With MidJourney Meth (GetImageUrl)



BC/NAV TECH DAYS 2023
mibuso.com

Like it?

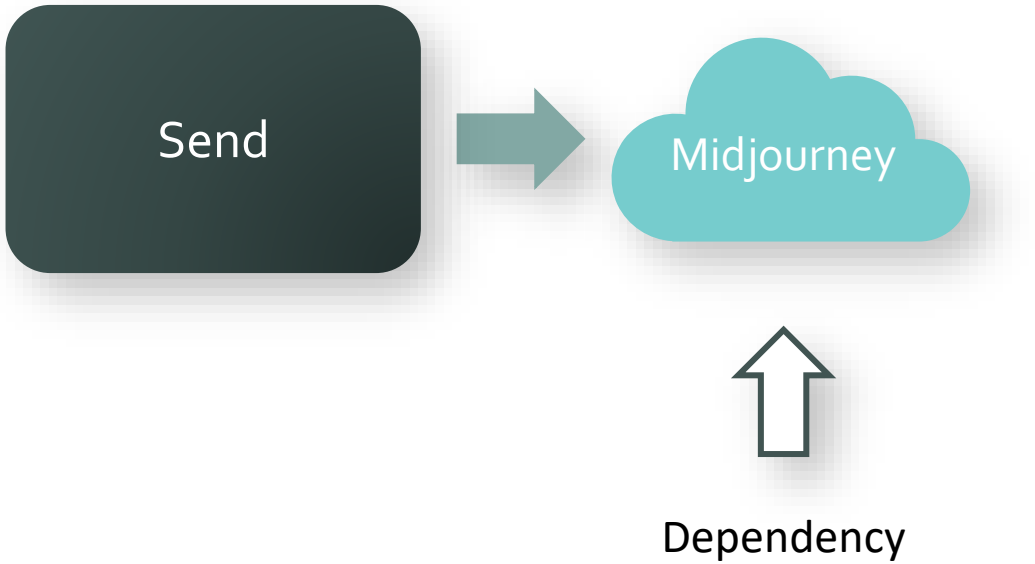
BC/NAV TECH DAYS 2023
mibuso.com

Dependencies

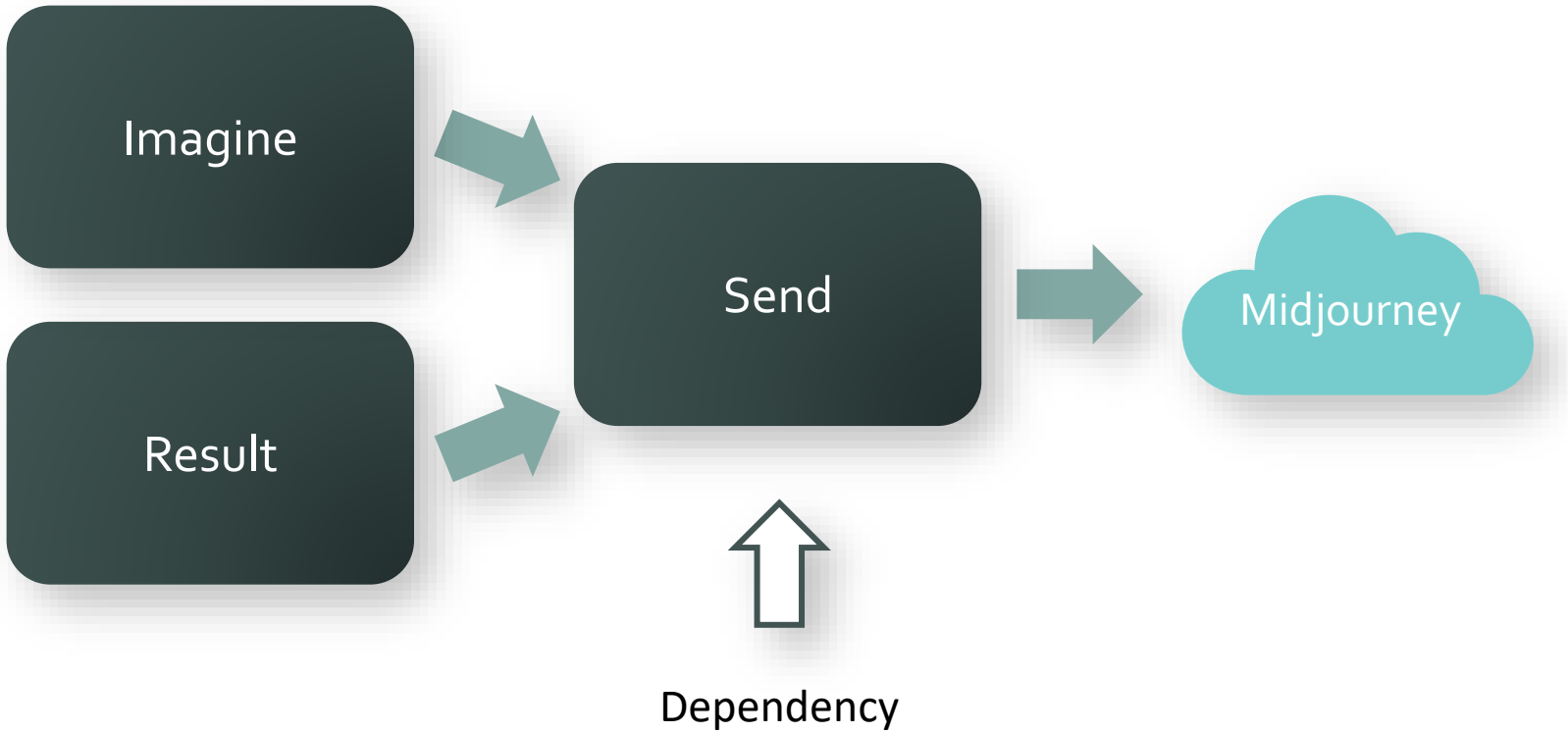
Dependencies

Any component or resource
that any other component
needs to fulfill its functionality

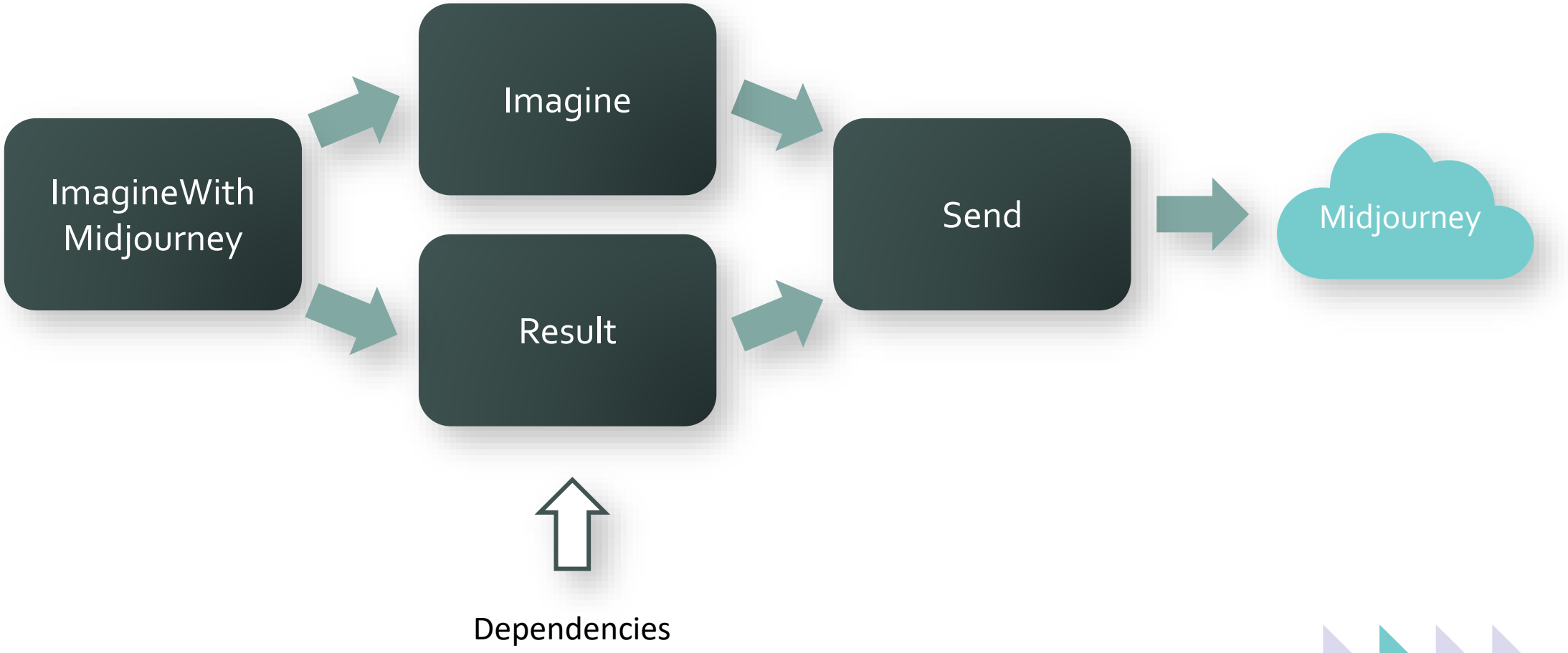
Dependencies



Dependencies



Dependencies



Dependencies

- Coupling vs. Cohesion

Dependencies

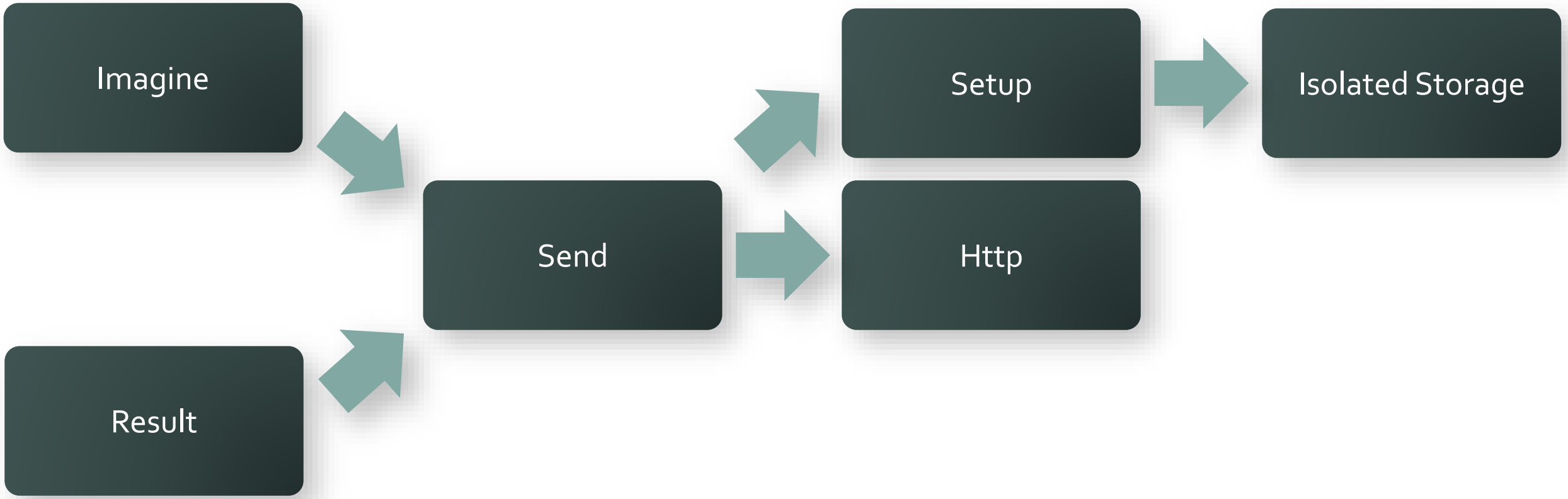
Coupling

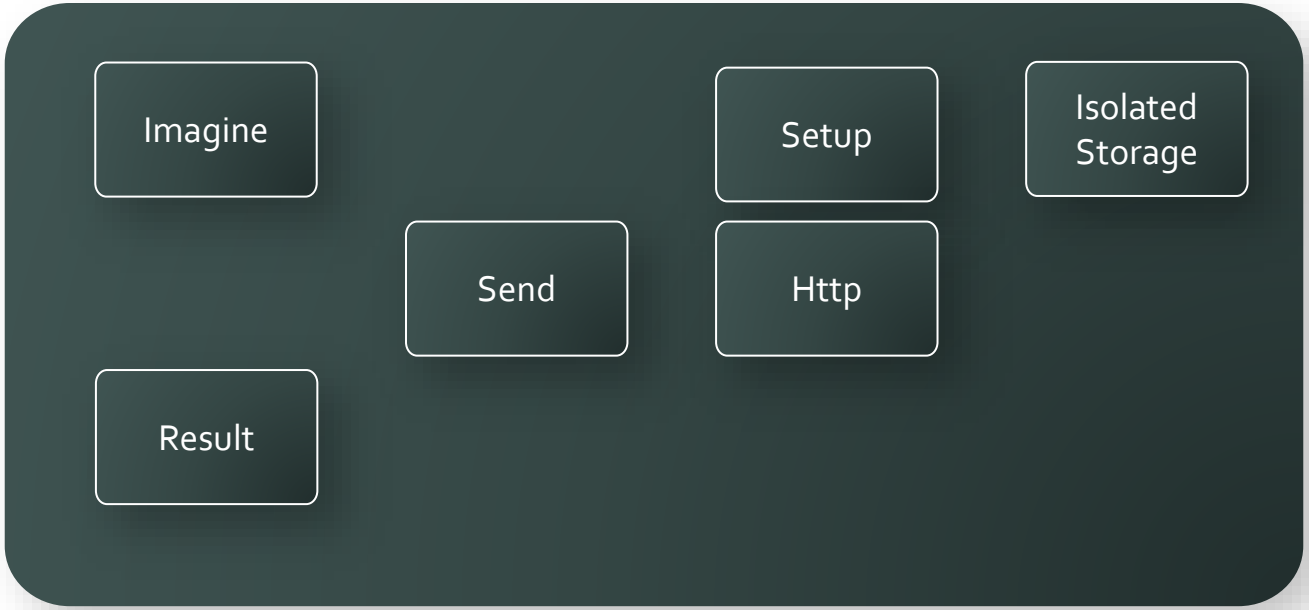
- Degree of interdependence of different components
- *How strongly are these components interdependent?*

Cohesion

- Degree to which components belong together
- *How well do these components belong together?*

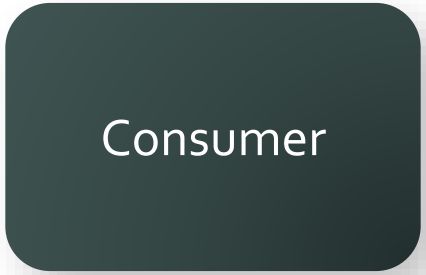
Imagine With MidJourney Meth (GetImageUrl)



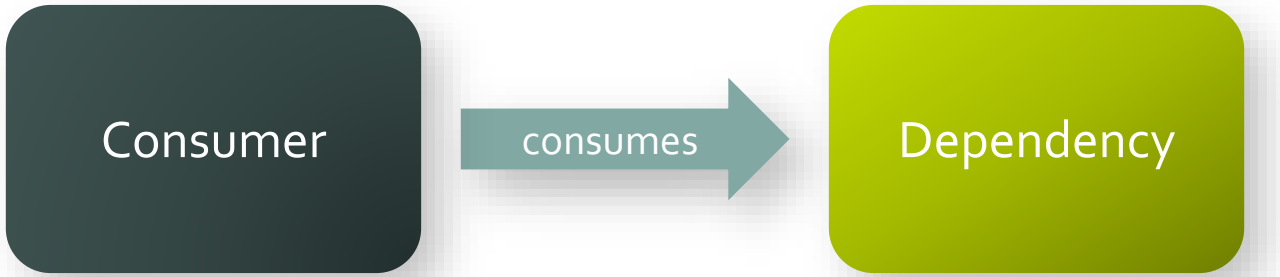

Imagine
Setup
Isolated Storage
Send
Http
Result

Inversion of control

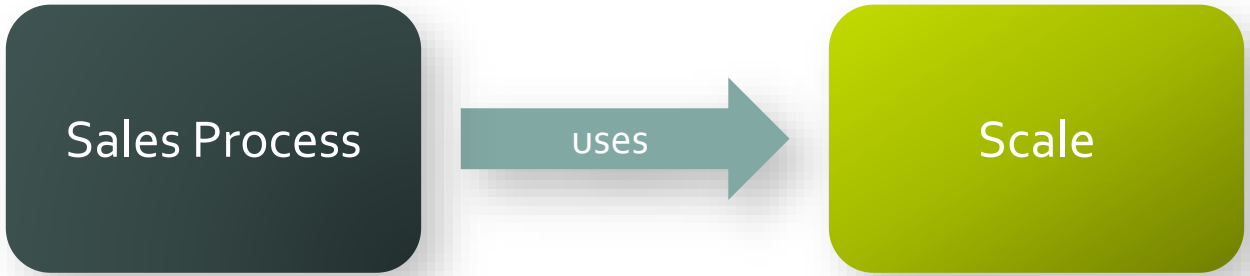
Inversion of control



Inversion of control

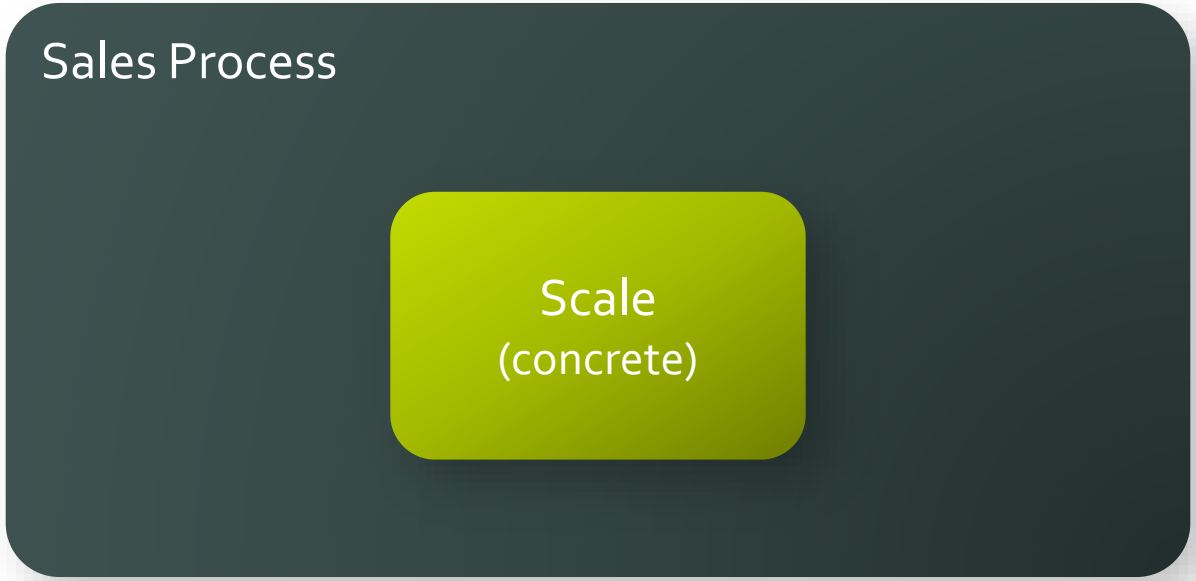


Inversion of control




```
procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Codeunit "Scale - Mettler Toledo";
    Weight: Decimal;
begin
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

Inversion of control

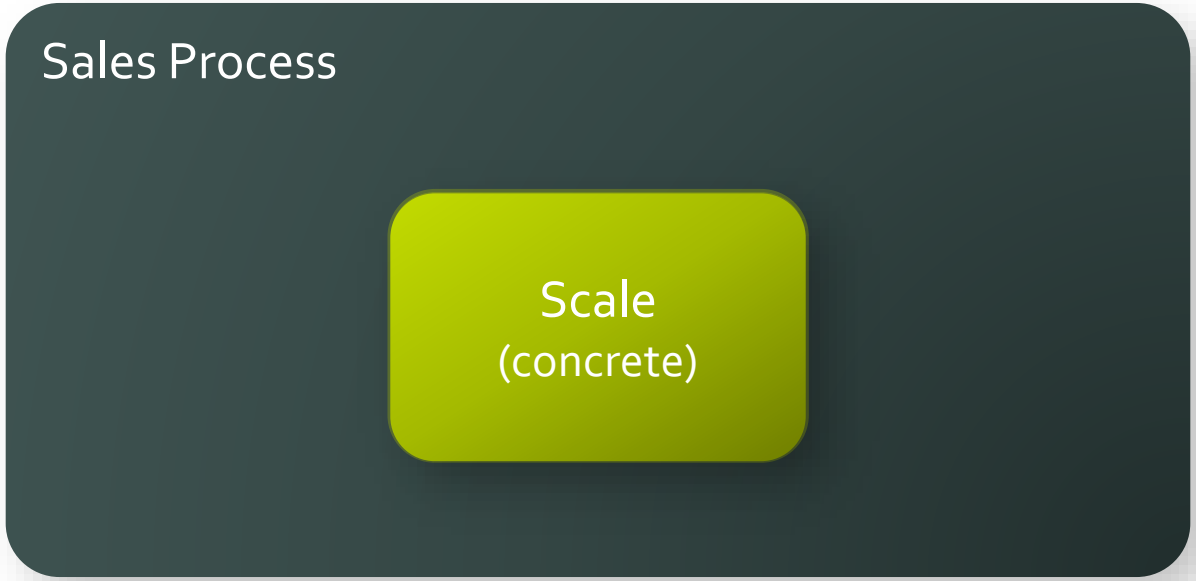


- Sales process declares ***concrete*** scale at compile time
- Sales process ***controls*** this dependency

Inversion of control

- Sales process declares ***abstract*** scale at compile time
- Sales process ***no longer*** controls this dependency

Inversion of control



- Sales process receives/obtains ***concrete*** scale at run time
- Another process ***controls*** this dependency

Dependency Injection

```
procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Codeunit "Scale - Mettler Toledo";
    Weight: Decimal;
begin
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

Concrete
tightly coupled
dependency

Dependency Injection

```
procedure GetWeight(var SalesLine: Record "Sales Line"; Scale: Interface IScale)
var
    Weight: Decimal;
begin
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

Abstract
loosely coupled
dependency

Factory

Abstract
loosely coupled
dependency

```
procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Interface IScale;
    Weight: Decimal;
begin
    Scale := GetScale();
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

Factory

Consumer ***asks*** for concrete implementation at run time

```
procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Interface IScale;
    Weight: Decimal;
begin
    Scale := GetScale();
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

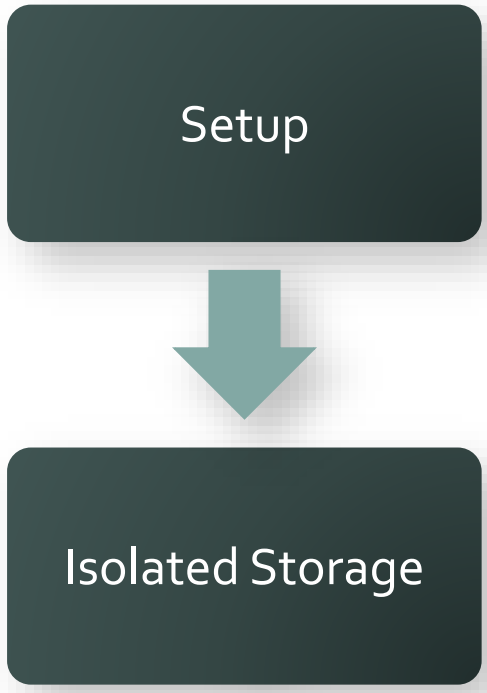
Factory

There are various techniques for providing concrete dependencies

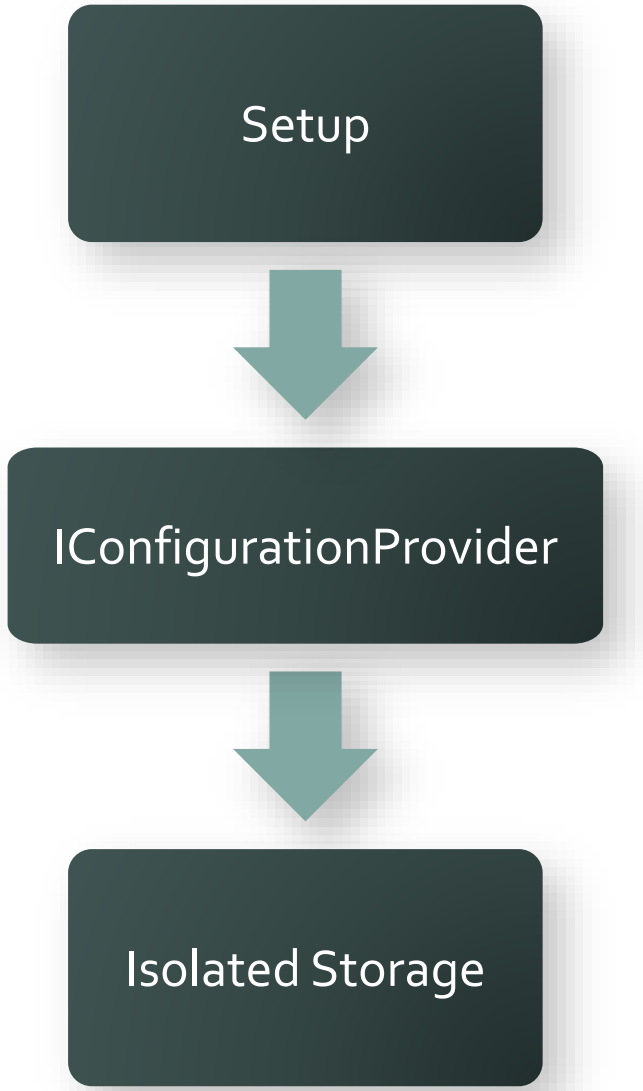
```
local procedure GetScale(): Interface IScale
begin
    // Here be dragons
end;

procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Interface IScale;
    Weight: Decimal;
begin
    Scale := GetScale();
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

What about Isolated Storage?

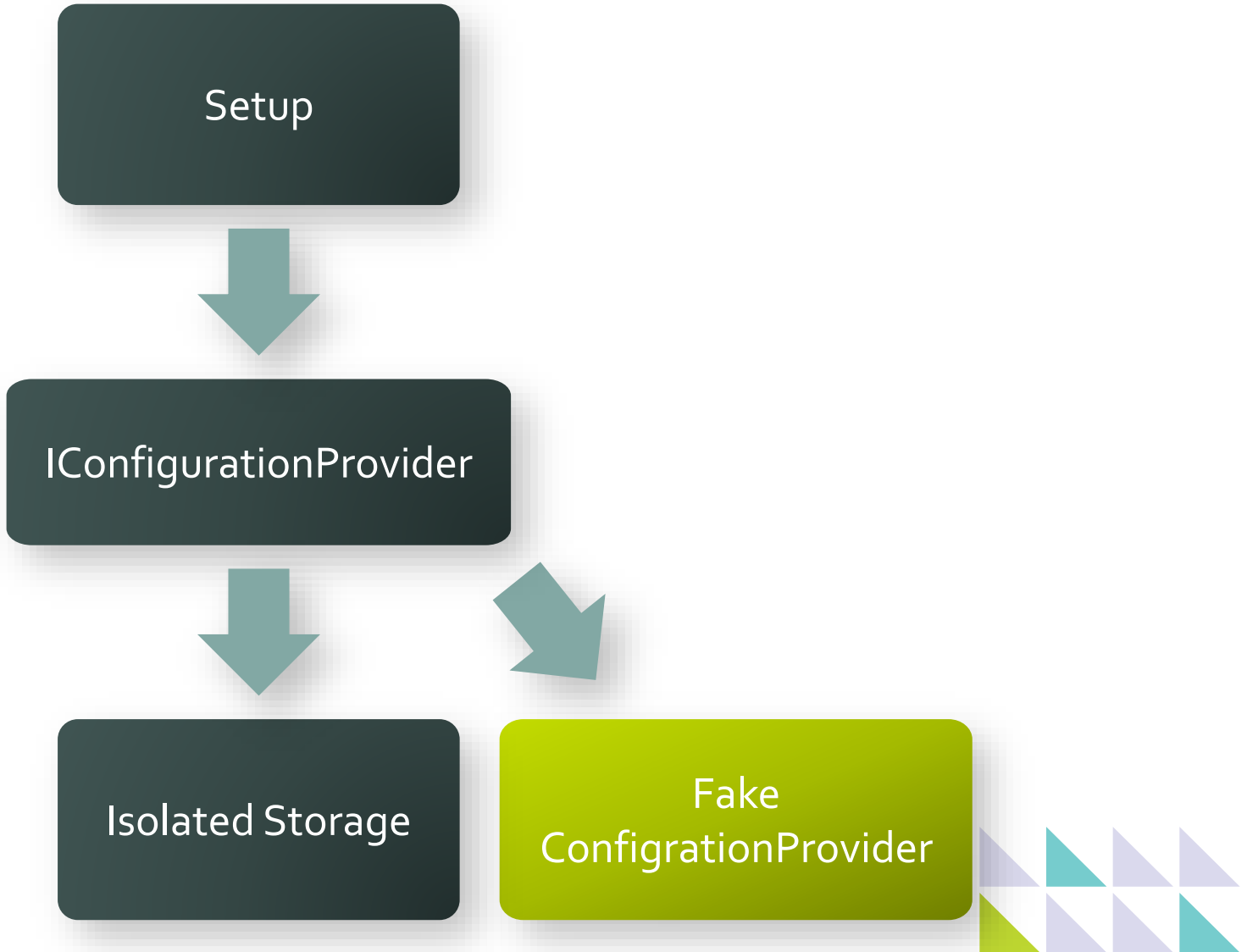
Decouple dependencies



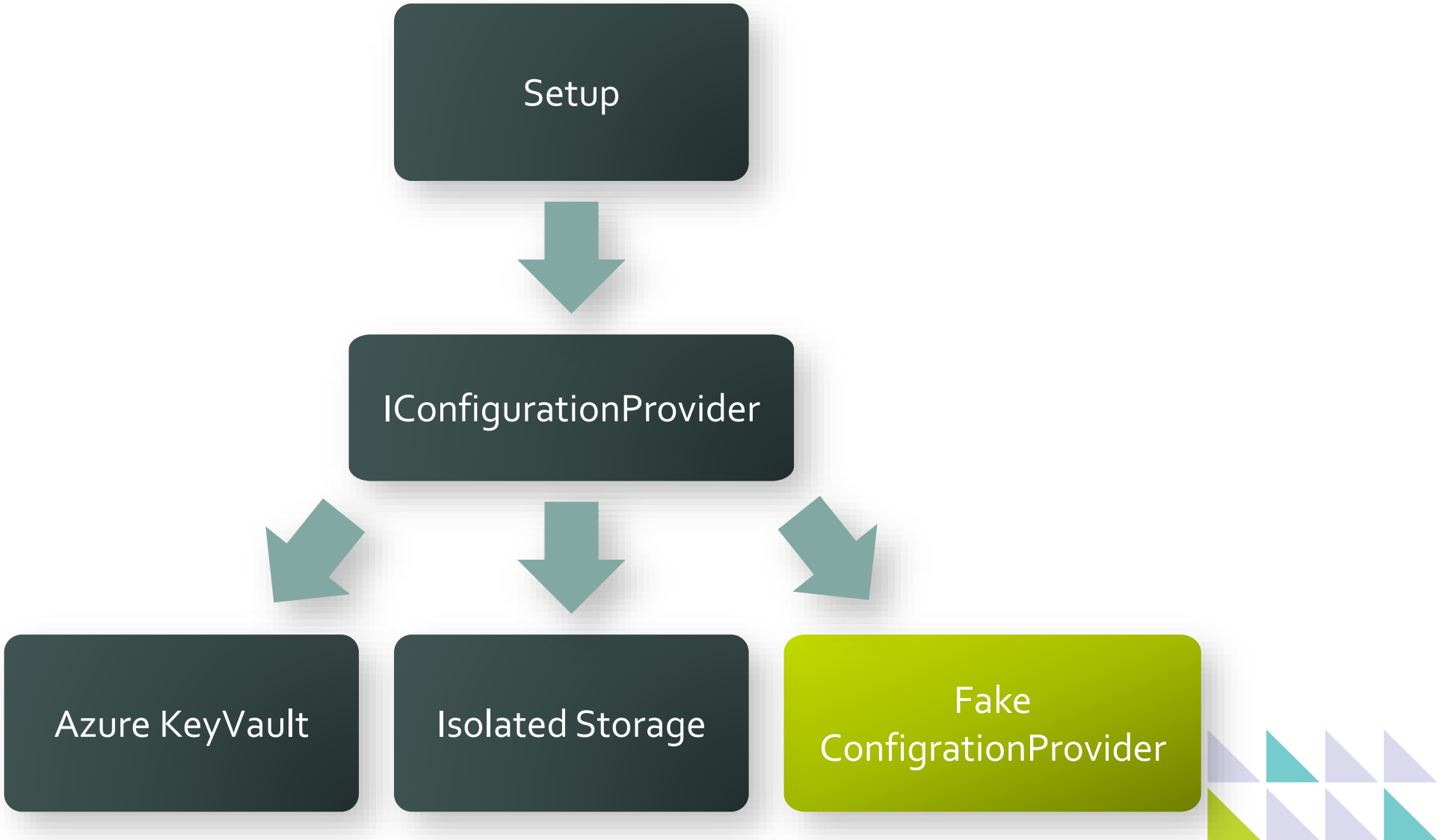
Decouple dependencies



Decouple dependencies



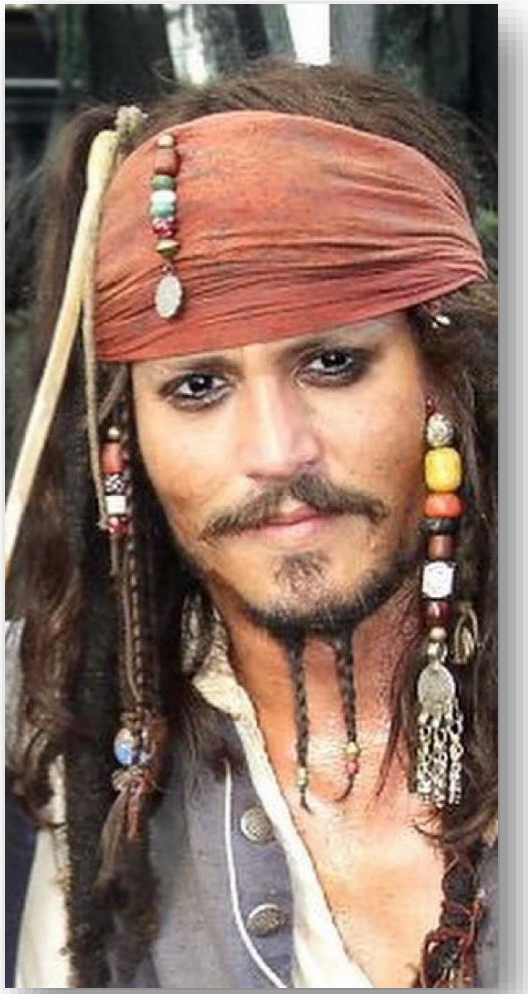
Decouple dependencies



BC/NAV TECH DAYS 2023
mibuso.com

Test doubles

Test doubles



Test doubles

Test doubles

A test double is an object that
***stands in for a real object in a test**,*
just like a stunt double
stands in for an actor in a movie.

How do test doubles work?

```
procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Interface IScale;
    Weight: Decimal;
begin
    Scale := GetScale();
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

Do we really
communicate
with a scale here?

How do test doubles work?

```
procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Interface IScale;
    Weight: Decimal;
begin
    Scale := GetScale();
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

Does this function
really care about ***how exactly***
the weight was taken?

How do test doubles work?

```
procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Interface IScale;
    Weight: Decimal;
begin
    Scale := GetScale();
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

```
interface IScale
{
    1 reference
    procedure TakeWeight(): Decimal;
}
```

The process is
abstracted
behind an interface

How do test doubles work?

```
procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Interface IScale;
    Weight: Decimal;
begin
    Scale := GetScale();
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

Your process does not care which one of these interface implementations was invoked

```
interface IScale
{
    1 reference
    procedure TakeWeight(): Decimal;
}
```

```
codeunit 60219 "Scale - B-TEK" implements IScale
{
    procedure TakeWeight(): Decimal
    begin
        // Some complex hardware communication code
    end;
}
```

```
codeunit 60217 "Scale - Tefal" implements IScale
{
    procedure TakeWeight(): Decimal
    begin
        // Some complex hardware communication code
    end;
}
```

```
codeunit 60218 "Scale - Mettler Toledo" implements IScale
{
    procedure TakeWeight(): Decimal
    begin
        // Some complex hardware communication code
    end;
}
```

How do test doubles work?

```
procedure GetWeight(var SalesLine: Record "Sales Line")
var
    Scale: Interface IScale;
    Weight: Decimal;
begin
    Scale := GetScale();
    Weight := Scale.GetWeight();
    SalesLine.Validate("Gross Weight", Weight);
end;
```

… or if a mock implementation was invoked

```
interface IScale
{
    1 reference
    procedure TakeWeight(): Decimal;
}
```

```
codeunit 60219 "Scale - B-TEK" implements IScale
{
    procedure TakeWeight(): Decimal
    begin
        // Some complex hardware communication code
    end;
}
```

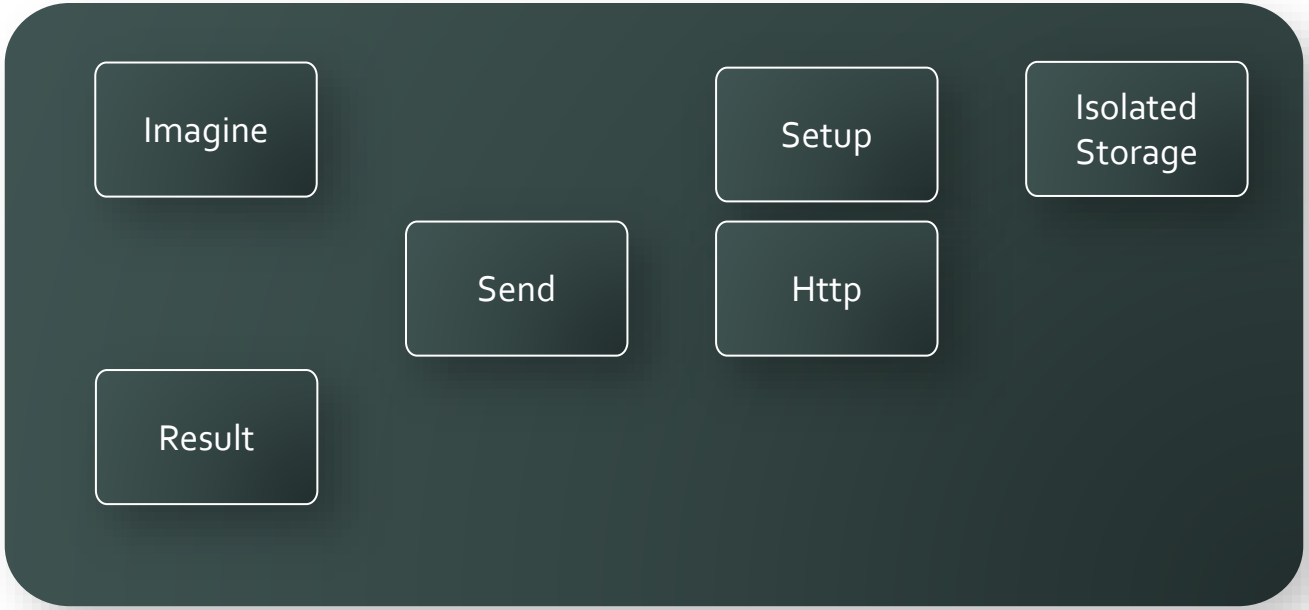
```
codeunit 60217 "Scale - Tefal" implements IScale
{
    procedure TakeWeight(): Decimal
    begin
        // Some complex hardware communication code
    end;
}
```

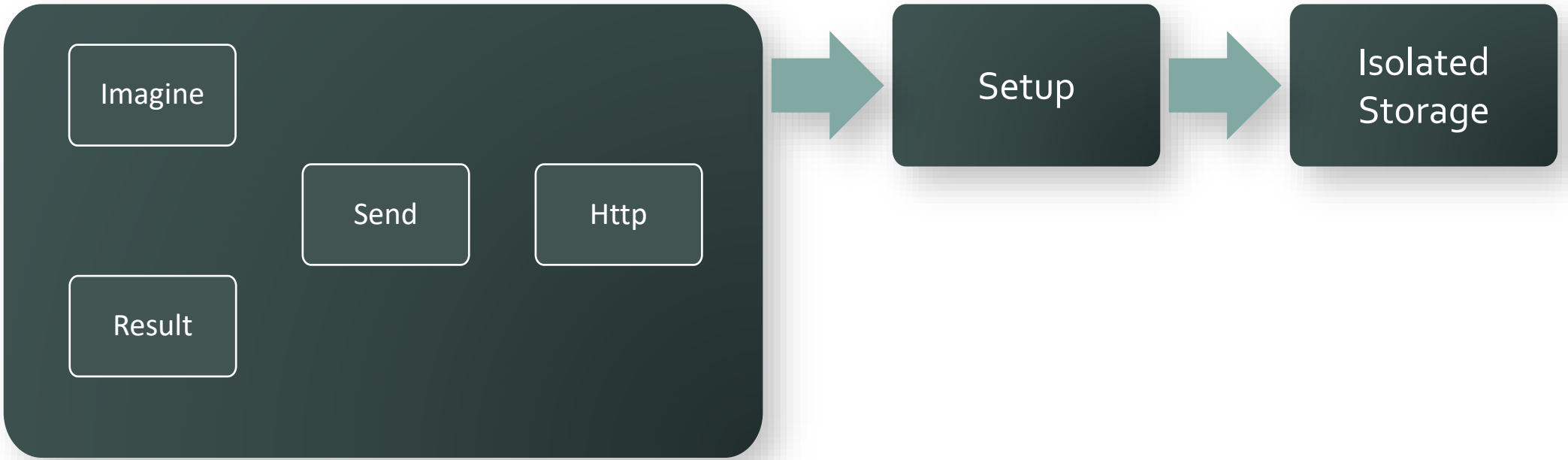
```
codeunit 60218 "Scale - Mettler Toledo" implements IScale
{
    procedure TakeWeight(): Decimal
    begin
        // Some complex hardware communication code
    end;
}
```

```
codeunit 60220 "Scale - Mock" implements IScale
{
    procedure TakeWeight(): Decimal
    begin
        exit(Random() * 100)
    end;
}
```

What have we done so far?

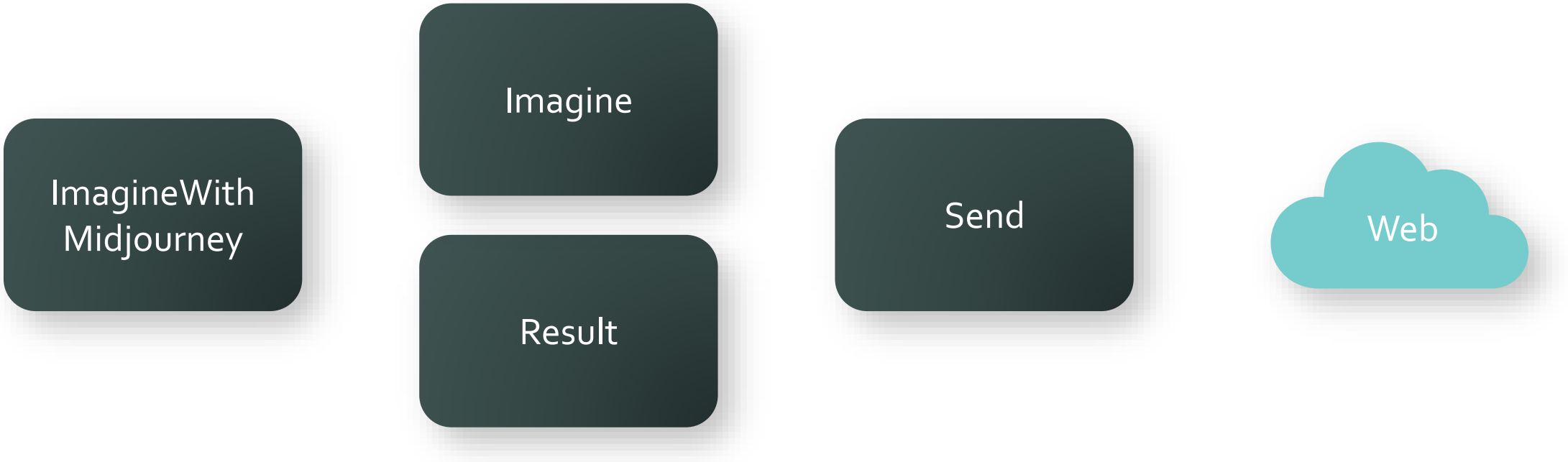
- Decoupled setup
- Decoupled Isolated Storage
- We can test isolated from setup or database
- We can exchange our components (eg, Azure Key Vault)


Imagine
Setup
Isolated Storage
Send
Http
Result

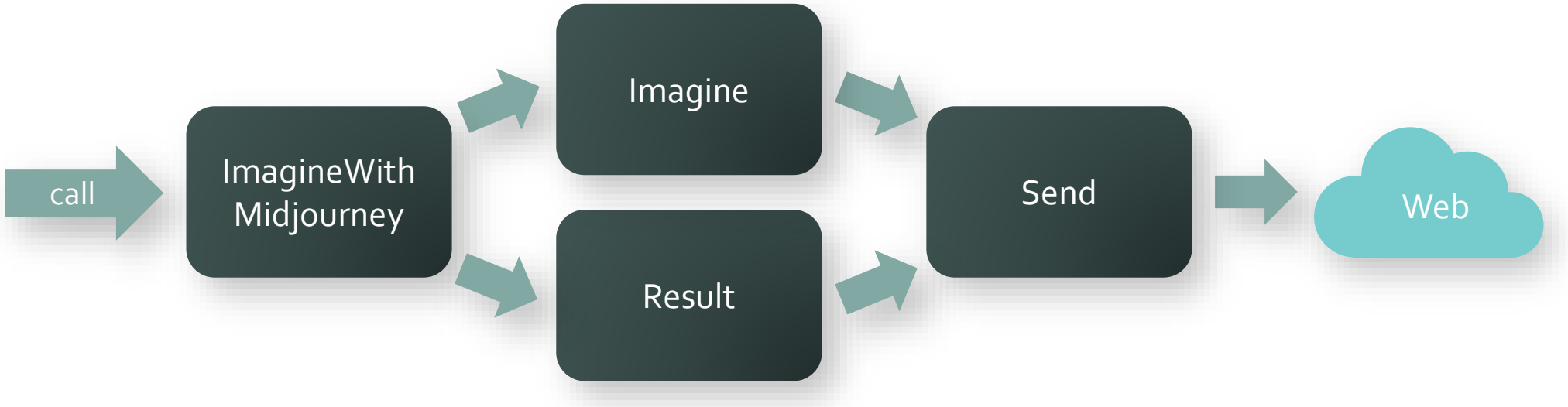

Imagine
Send
Http
Result
Setup
Isolated Storage

Testing in Isolation

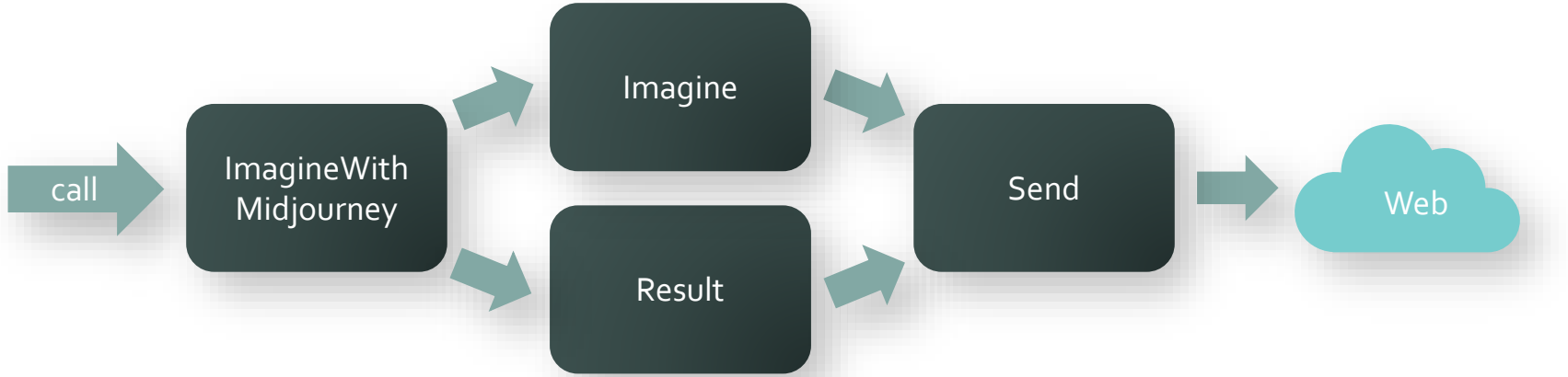
Testing in isolation



Testing in isolation



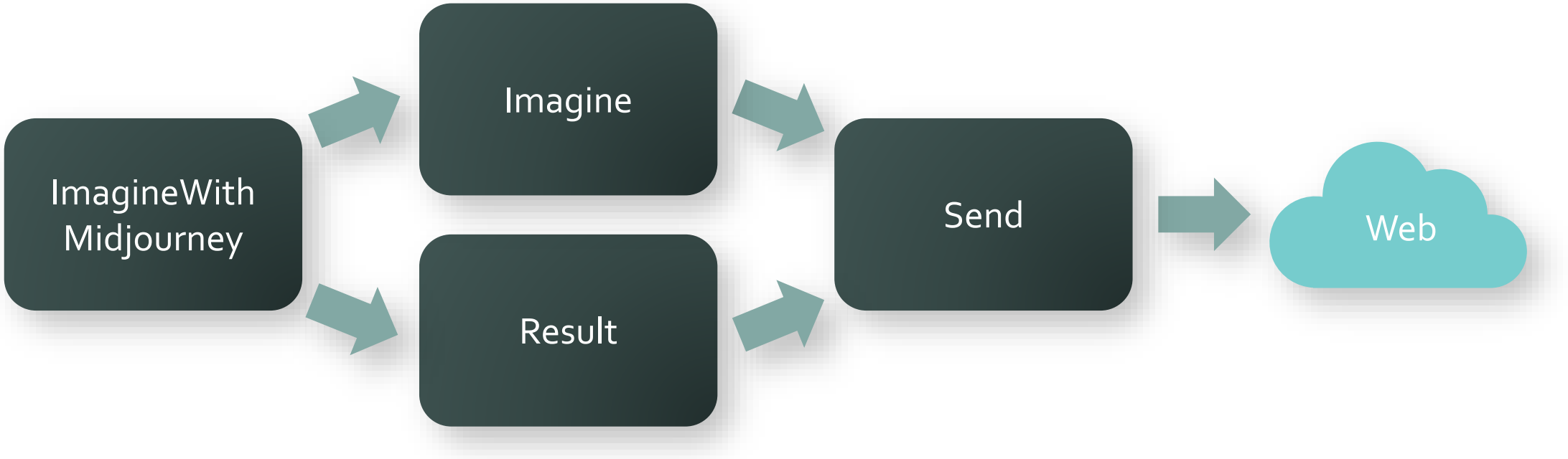
Testing in isolation



Testing directly is problematic:

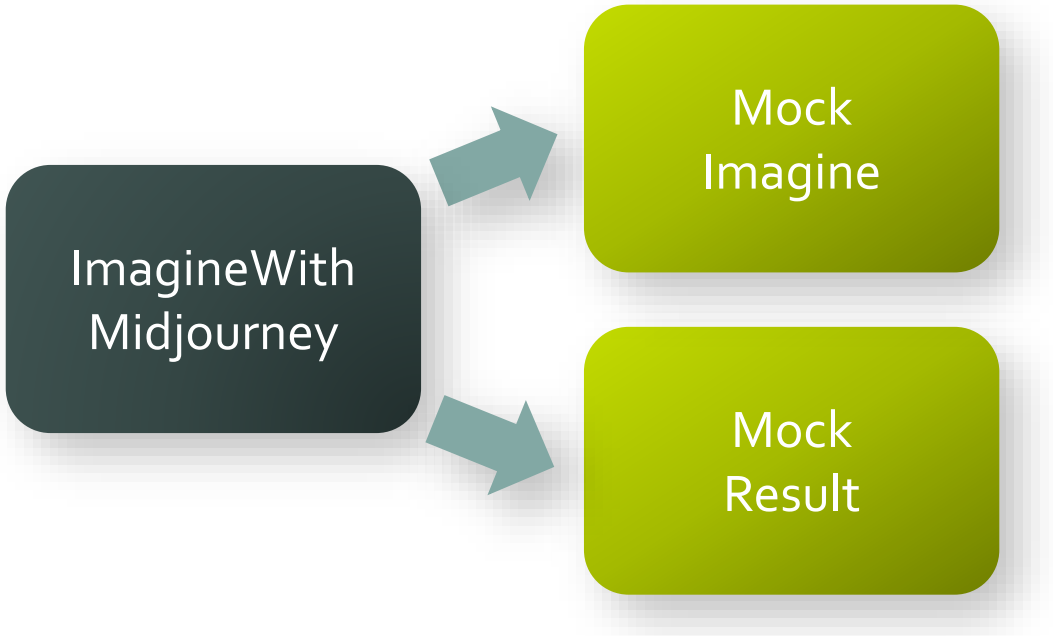
- Requires a lot of "givens"
- Takes a lot of time to properly set up
- Executes slowly
- ***Does not enable testing all flows***

Testing in isolation



We must test each component in isolation from its dependencies

Testing in isolation



At each level:

- We test the component directly
- We mock its direct dependencies

Testing in isolation

At each level:

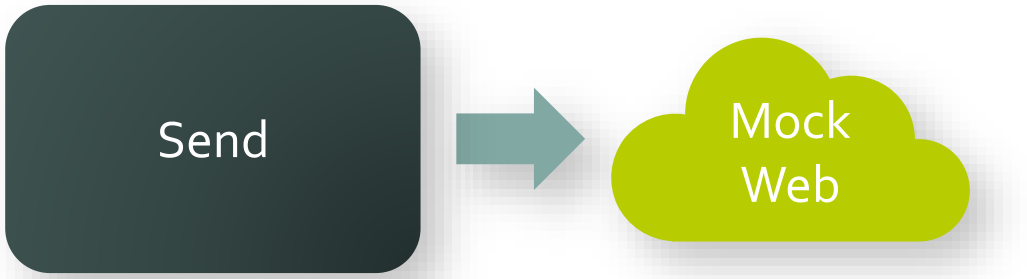
- We test the component directly
- We mock its direct dependencies

Testing in isolation

At each level:

- We test the component directly
- We mock its direct dependencies

Testing in isolation



At each level:

- We test the component directly
- We mock its direct dependencies

Testing in isolation

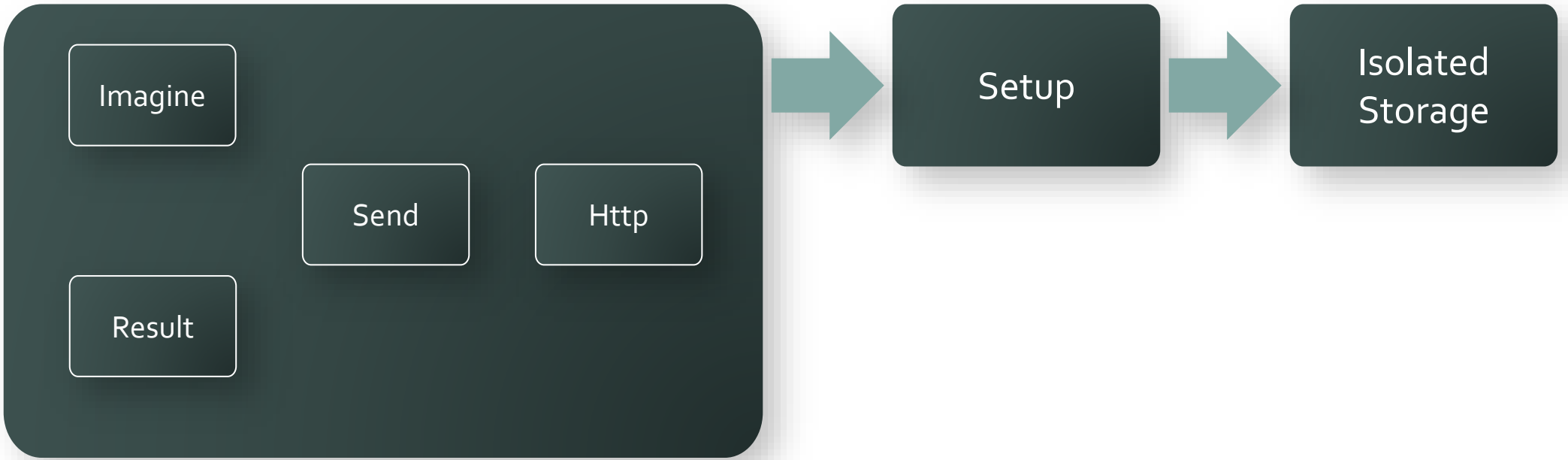
- We tested ***all components***
- We tested ***all code paths***
- We simulated ***all conditions***

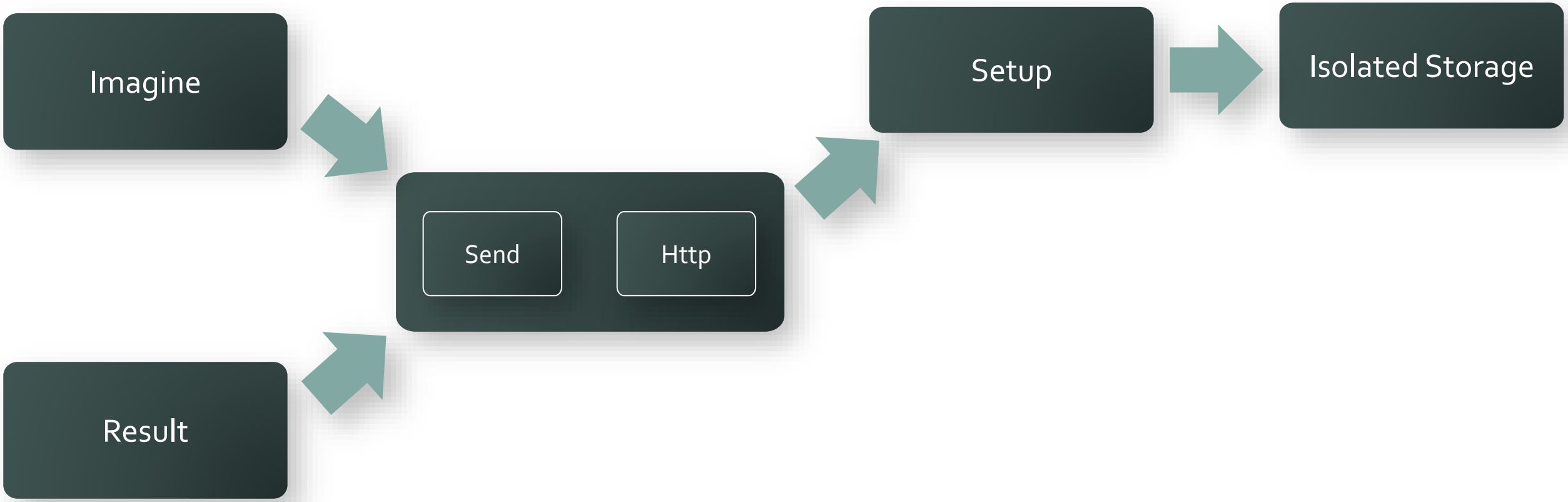
- Tests were ***simple to write***
- Tests ***performed fast***

Tests pinpoint design issues

When you can't test a component in isolation,
you know there is a problem with your design.

-- a wise man once said --


Imagine
Send
Http
Result
Setup
Isolated Storage


Imagine
Result
Send
Http
Setup
Isolated Storage

Where are we …

- Decoupling helps us to test
- But it also gives opportunities
  - Dall-E
  - Azure Key Vault
- Dependency Injection makes us able to control the implementations wherever we want..

But …

```
internal procedure GetImageUrl(Prompt: Text; Imagine: Interface IMidjourneyImagine; Result
var
    IsHandled: Boolean;
begin
```

```
t: Interface IMidjourneyResult; MidjourneySend: Interface IMidjourneySend) MidjourneyUrl: Text
```

BC/NAV TECH DAYS 2023
mibuso.com

Factory

Factory

"An object that the client code can use to obtain instances of its dependencies, without explicitly knowing the concrete class being instantiated."

An object

```
codeunit 50074 "Midjourney Factory"
{
    SingleInstance = true;
```

“ that the client code can use to obtain instances of its dependencies, without explicitly knowing the concrete class being instantiated.”

An object
that the client code can use

```
codeunit 50074 "Midjourney Factory"
{
    SingleInstance = true;
```

```
codeunit 50061 "ImagineWithMidjourney Meth"
{
    var
        Factory: Codeunit "Midjourney Factory";
```

“ to obtain instances of its dependencies, without explicitly knowing the concrete class being instantiated.”

An object
that the client code can use
to obtain instances of its dependencies

```
codeunit 50074 "Midjourney Factory"
{
    SingleInstance = true;
```

```
codeunit 50061 "ImagineWithMidjourney Meth"
{
    var
        Factory: Codeunit "Midjourney Factory";
```

```
local procedure DoGetImage(Prompt: Text; var MidjourneyUrl: Text;
var
    Imagine: Interface IMidjourneyImagine;
    TaskId: Text;
begin
    if IsHandled then
        exit;

    if not _retryDelaySet then
        _retryDelay := 5000;

    Imagine := Factory.GetMidjourneyImagine();
    TaskId := Imagine.Imagine(Prompt);
    MidjourneyUrl := WaitForUrl(TaskId);
end;
```

```
internal procedure GetImageUrl(Prompt: Text; Imagine: Interface IMidjourneyImagine; Result
var
    IsHandled: Boolean;
begin
```

```
internal procedure GetImageUrl(Prompt: Text
var
    IsHandled: Boolean;
begin
```

```
internal procedure GetImageUrl(Prompt: Text) MidjourneyUrl: Text
var
    IsHandled: Boolean;
begin
```

BC/NAV TECH DAYS 2023
mibuso.com

Demo:
Single Instance Factories

Single Instance Factories

Pros:

- Much easier to add/remove dependencies
- Easy to create/read tests

Cons:

- Dependency chains are now obscured
- Compiler doesn't help you
- Can be a liability, because developer might forget

Factory Injection

BC/NAV TECH DAYS 2023
mibuso.com

Demo:
Factory Injection

BC/NAV TECH DAYS 2023
mibuso.com

Hidden Dependencies

Key Takeaways

- **Interfaces** promote loose coupling and easier maintenance of the system.
- **Decoupling** enhances flexibility, testability, and maintainability.
- **Inversion of Control** inverts the control flow, improving modularity and extensibility.

Repo: https://github.com/vjekob/bctechdays2023

BC/NAV TECH DAYS 2023
mibuso.com

TDD

TDD is not about tests

BC/NAV TECH DAYS 2023
mibuso.com

TDD is not about tests
it's about design

BC/NAV TECH DAYS 2023
mibuso.com

Q&A

Any Questions?

BC/NAV TECH DAYS 2023
mibuso.com

Thank You!