Stylish Concurrency
using
Functional programming

Eduardo Morango

@caju

@edvmorango

Agenda

- Concurrency and parallelism
- A simple scraper
- Effects
- Optimizing the scraper
- Massive Parallelism
- Evaluation
- Monoids
- Functors
- Aplicative Functors
- Monads
- FP Intuition
- Conclusion

References

https://zio.dev

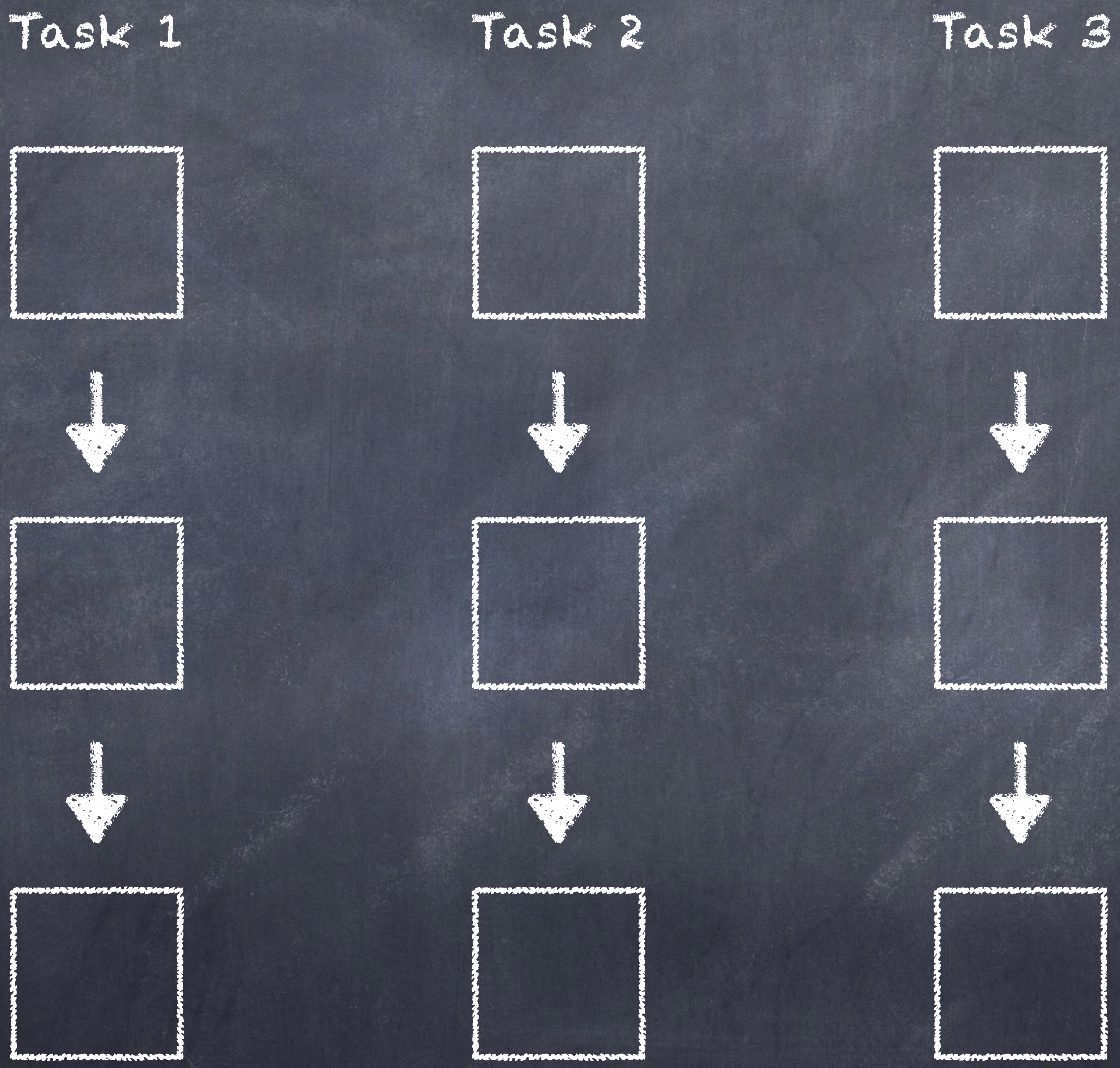
https://functional.works-hub.com/learn/the-science-behind-functional-programming-3b060

http://haskellbook.com/

https://underscore.io/books/scala-with-cats/

https://typelevel.org/cats/

Concurrency and Parallelism

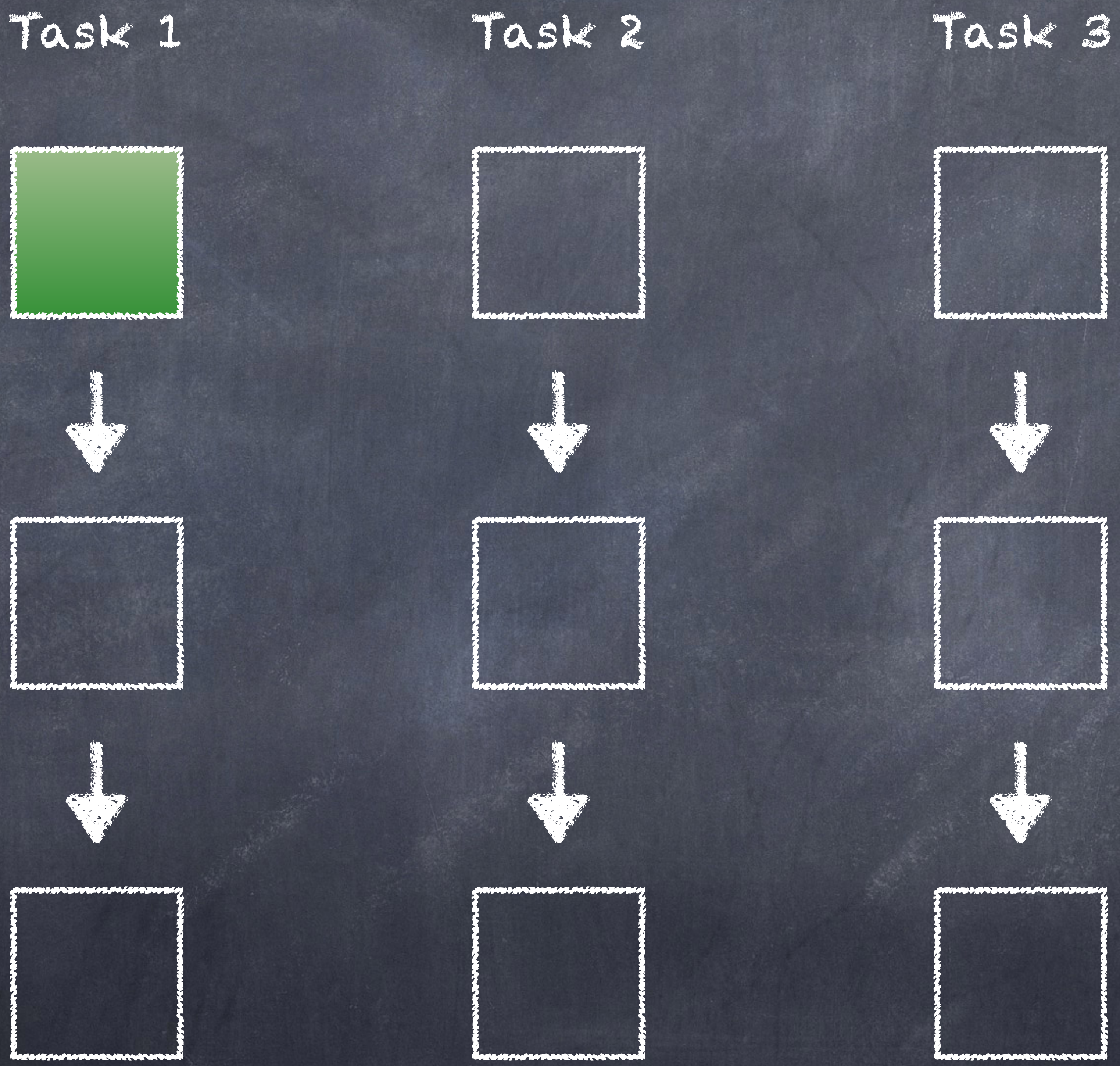


Task 1

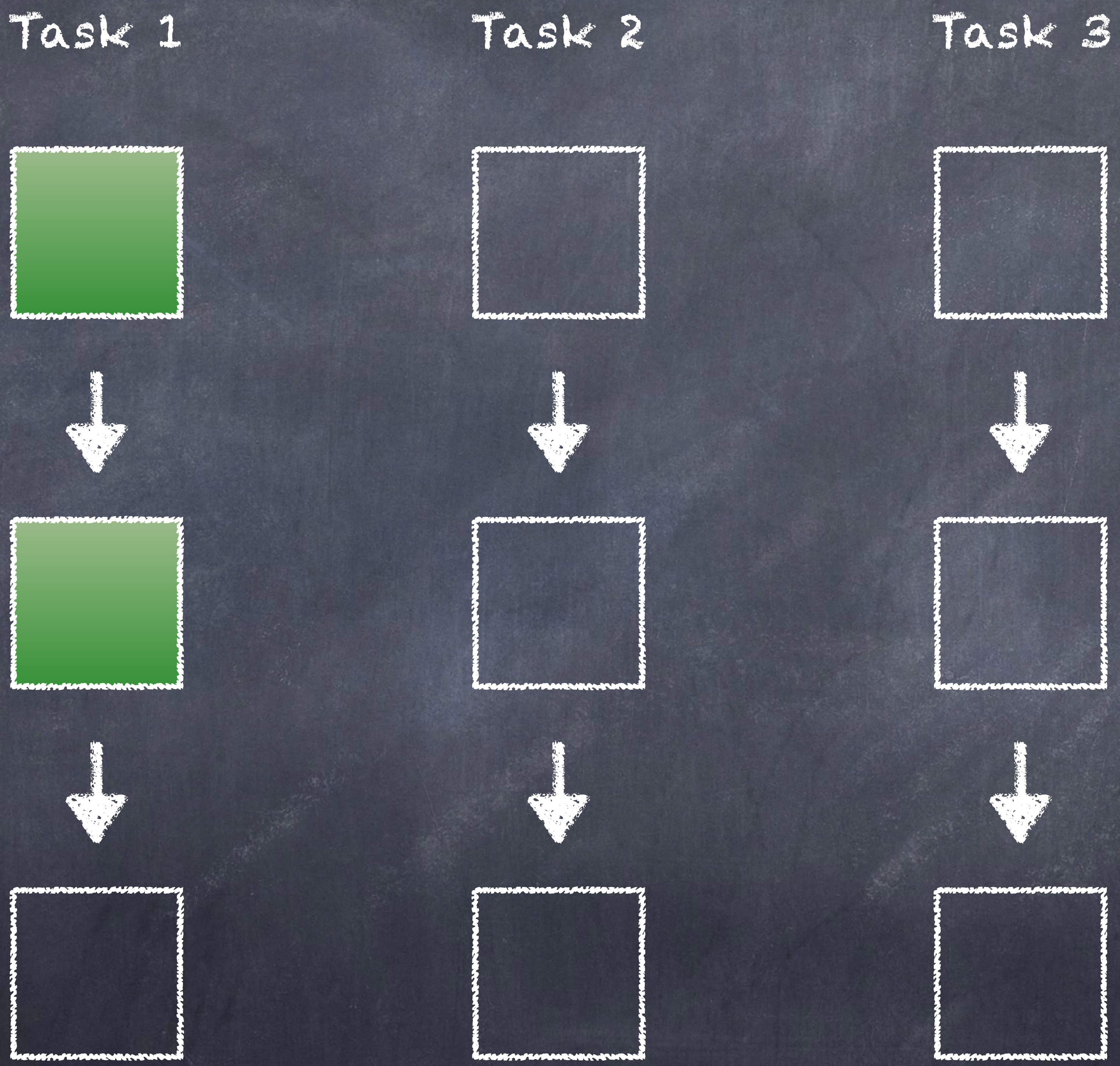
Task 2

Task 3

Concurrency and Parallelism



Concurrency and Parallelism

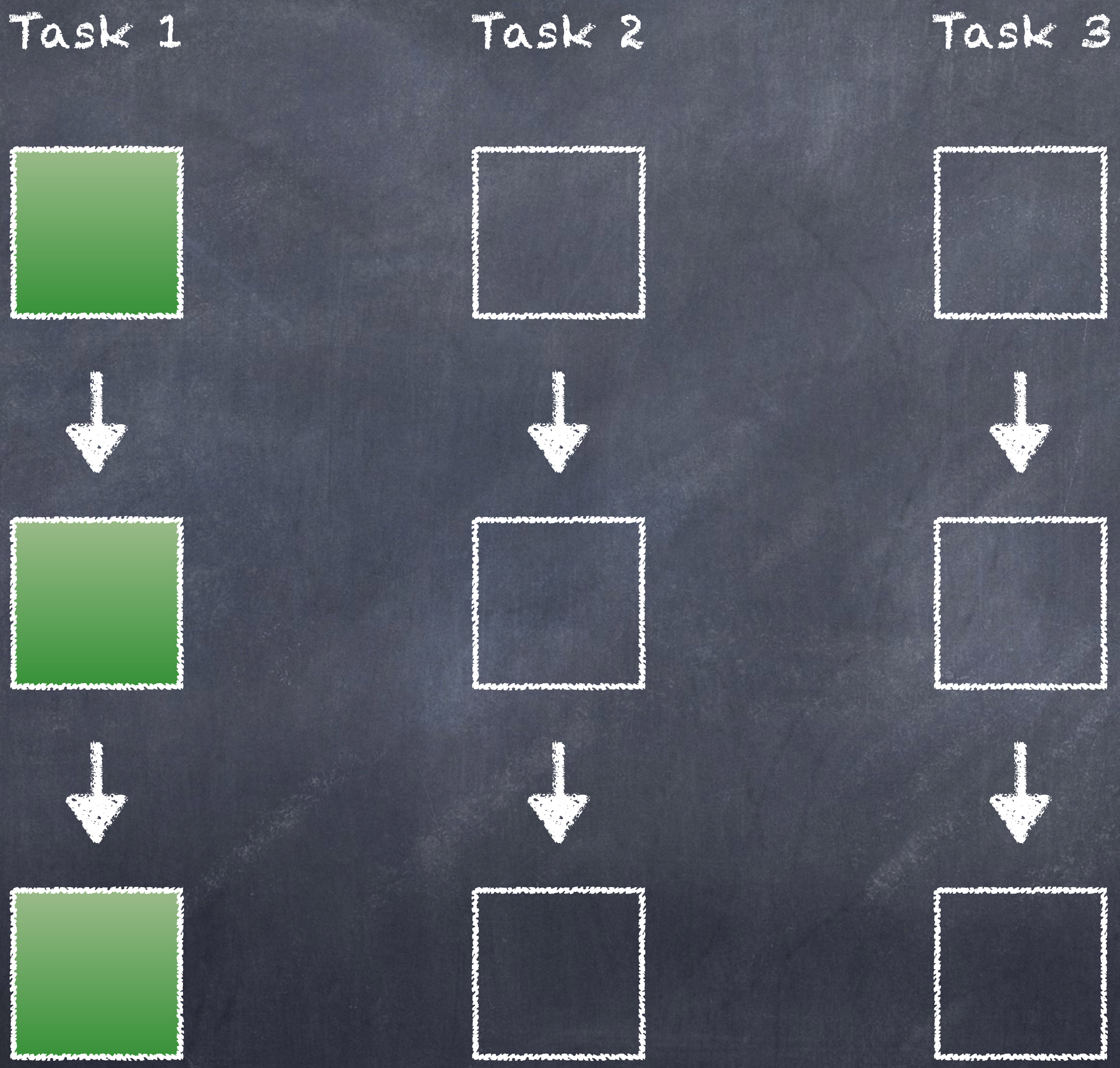


Task 1

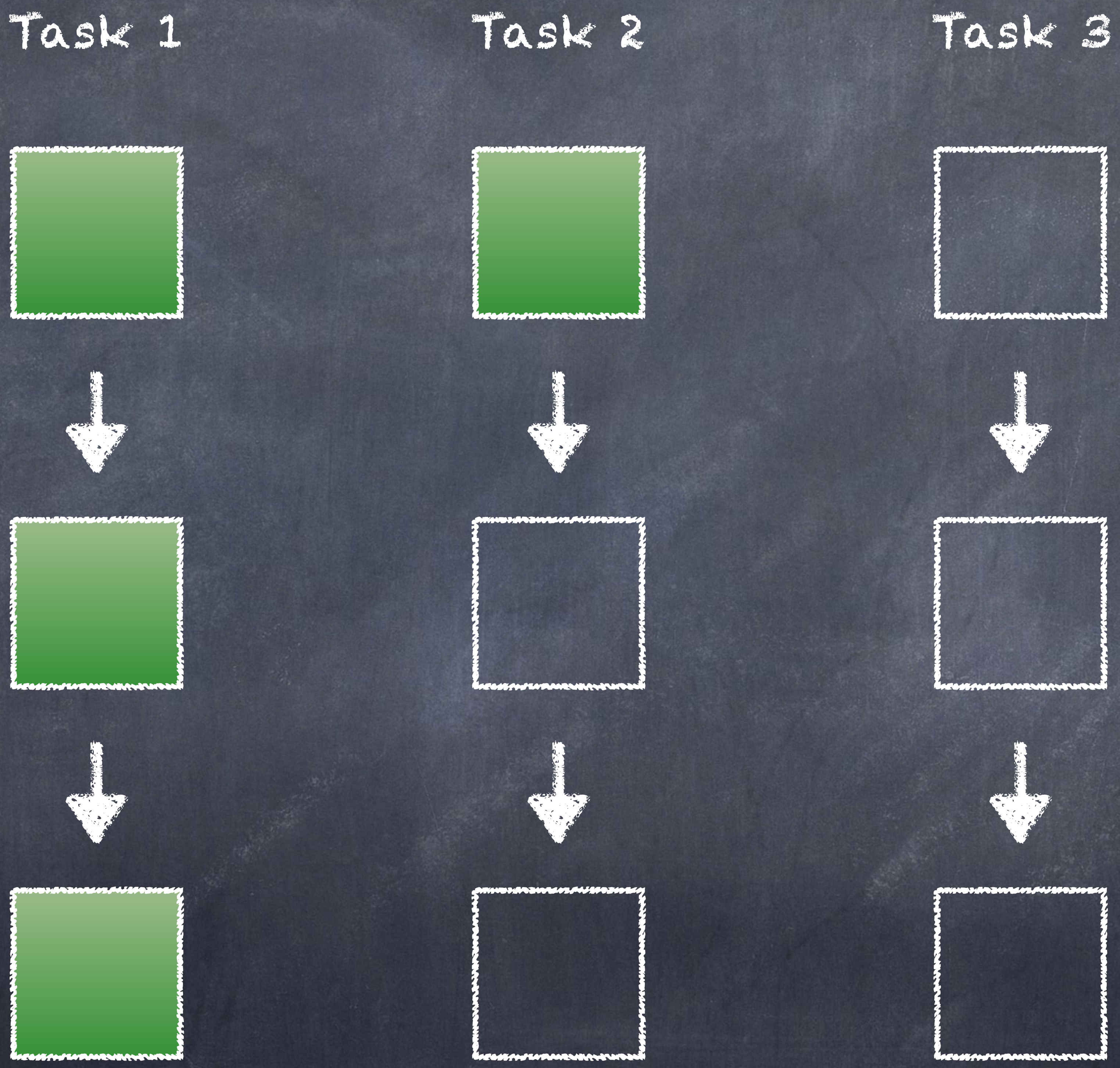
Task 2

Task 3

Concurrency and Parallelism

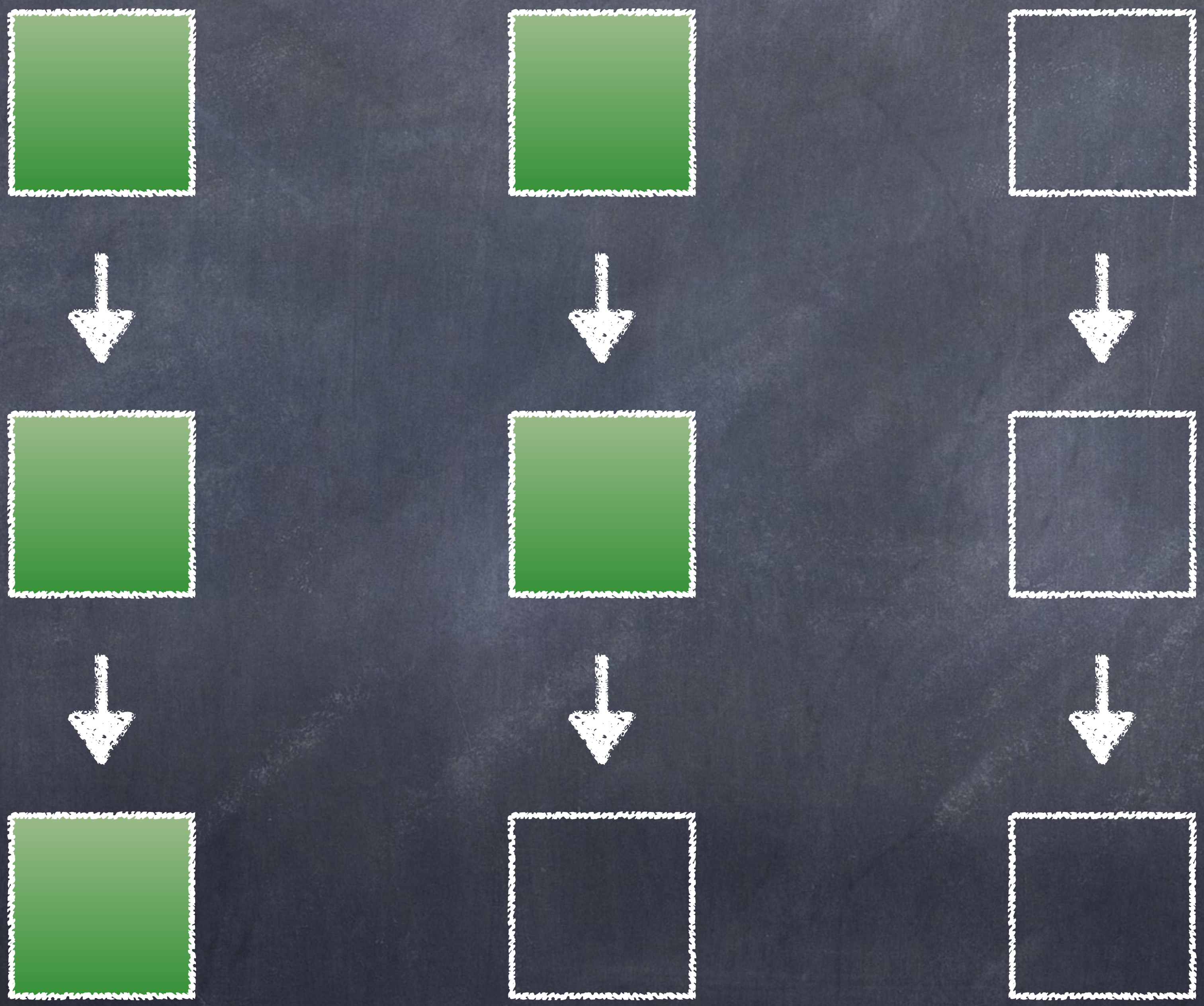


Concurrency and Parallelism

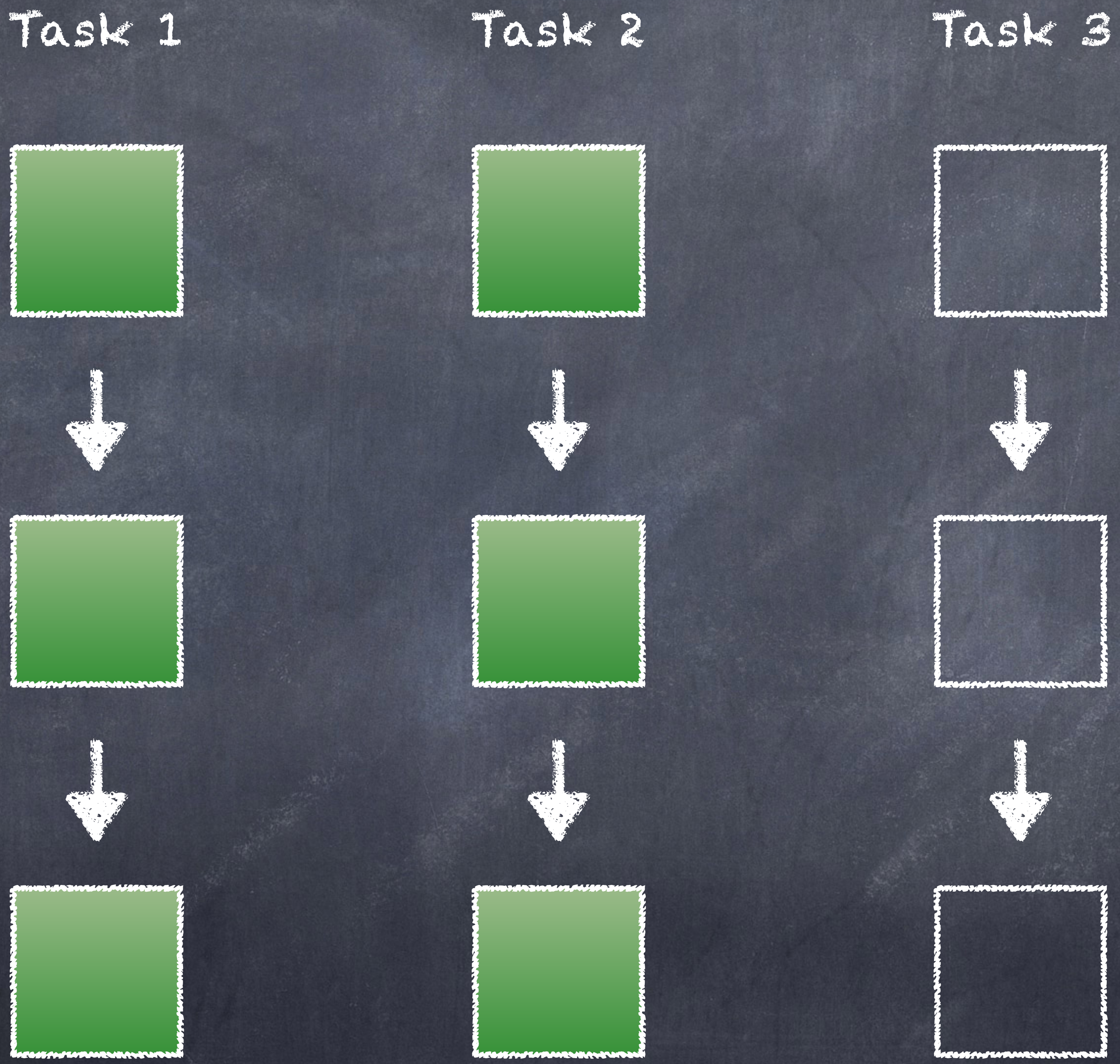


Concurrency and Parallelism

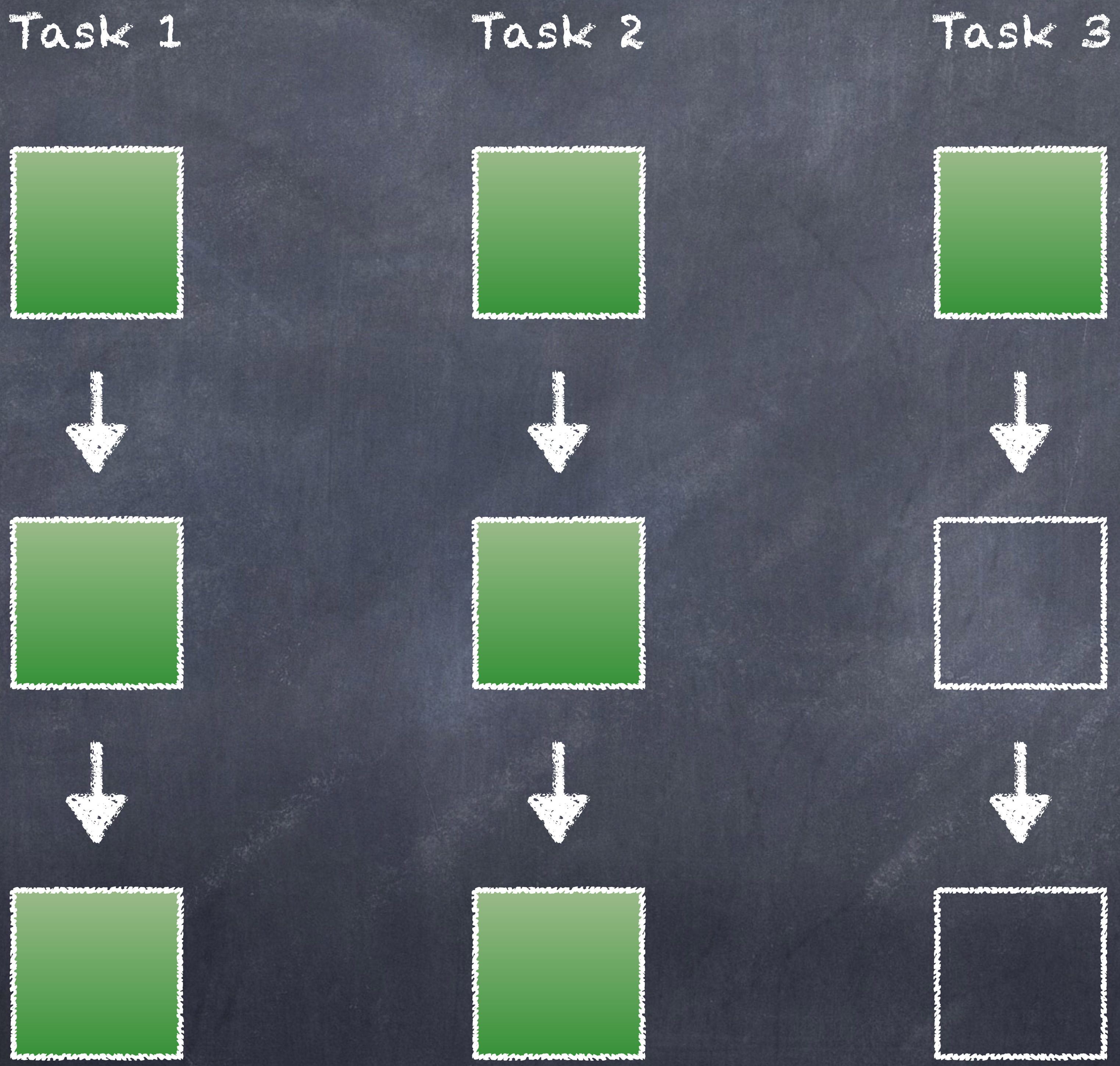
Task 1 Task 2 Task 3



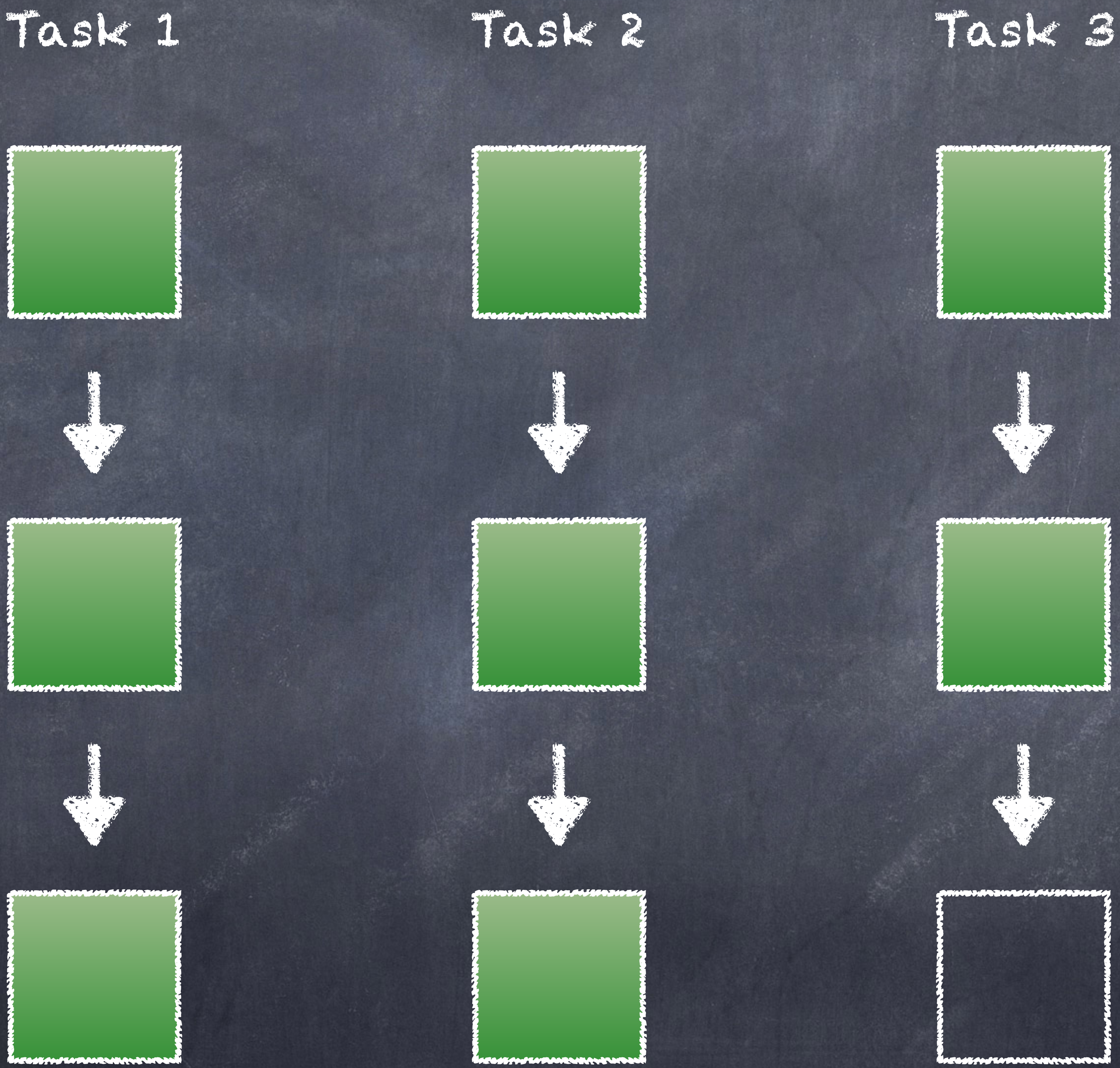
Concurrency and Parallelism



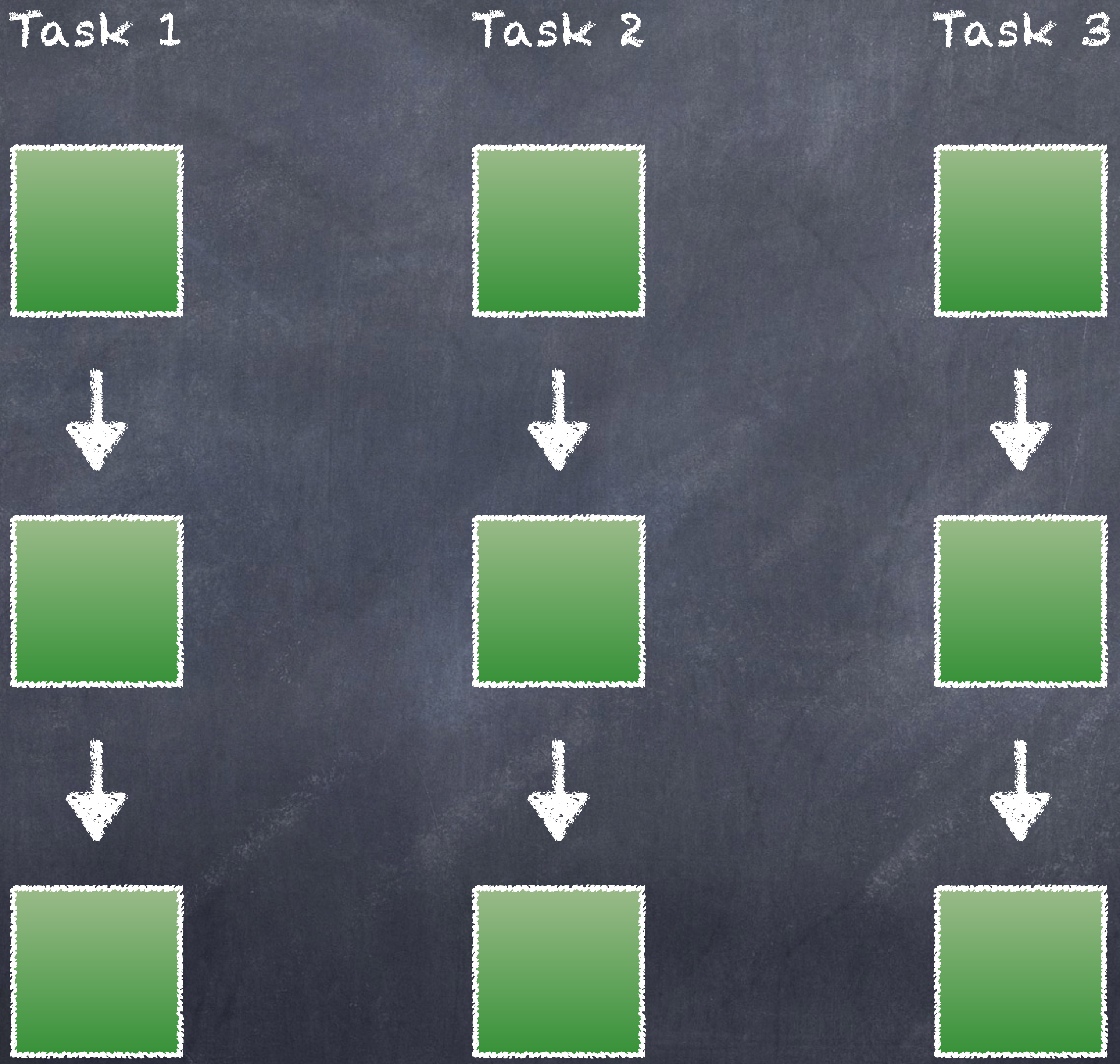
Concurrency and Parallelism



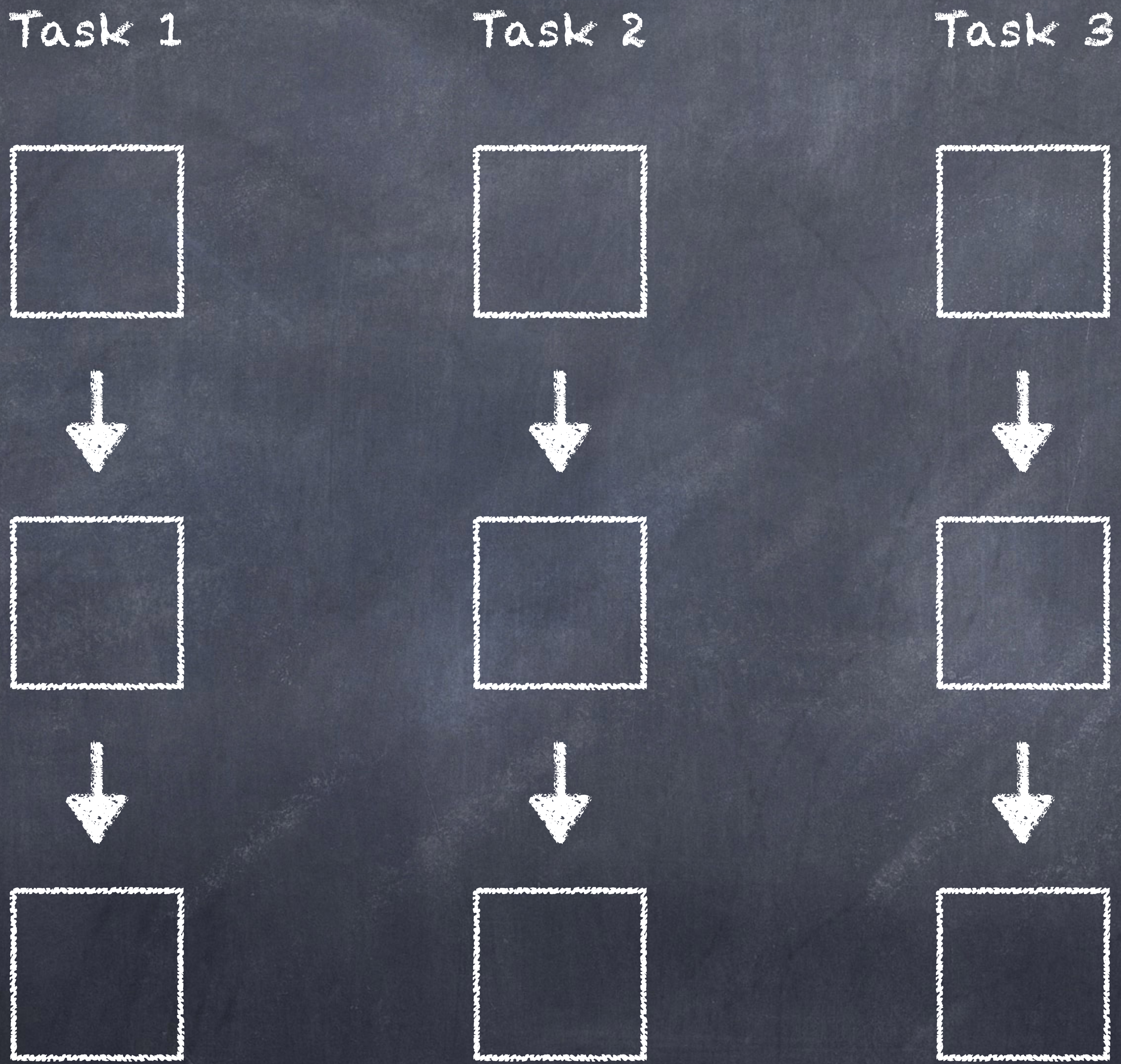
Concurrency and Parallelism



Concurrency and Parallelism



Concurrency and Parallelism

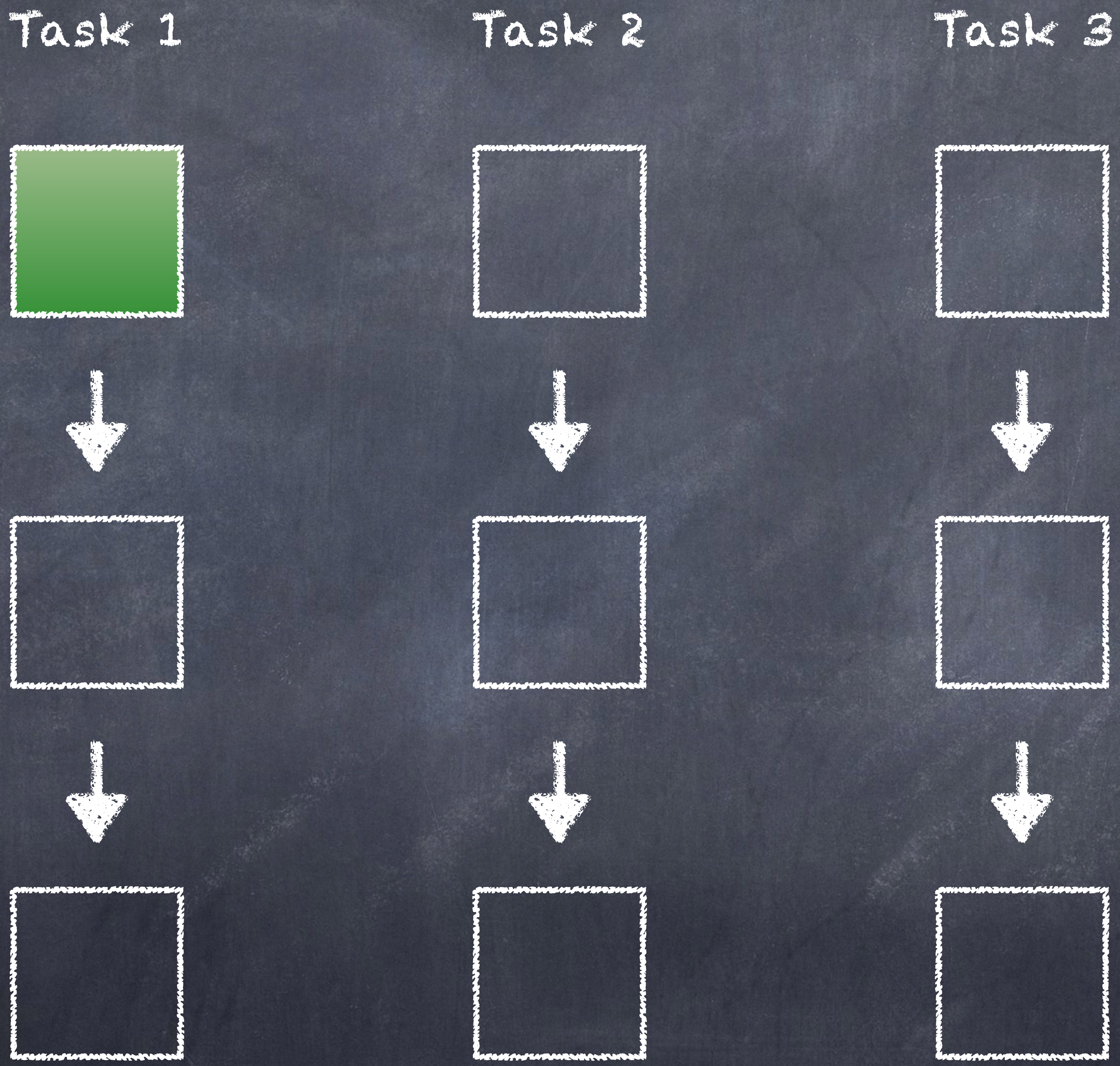


Task 1

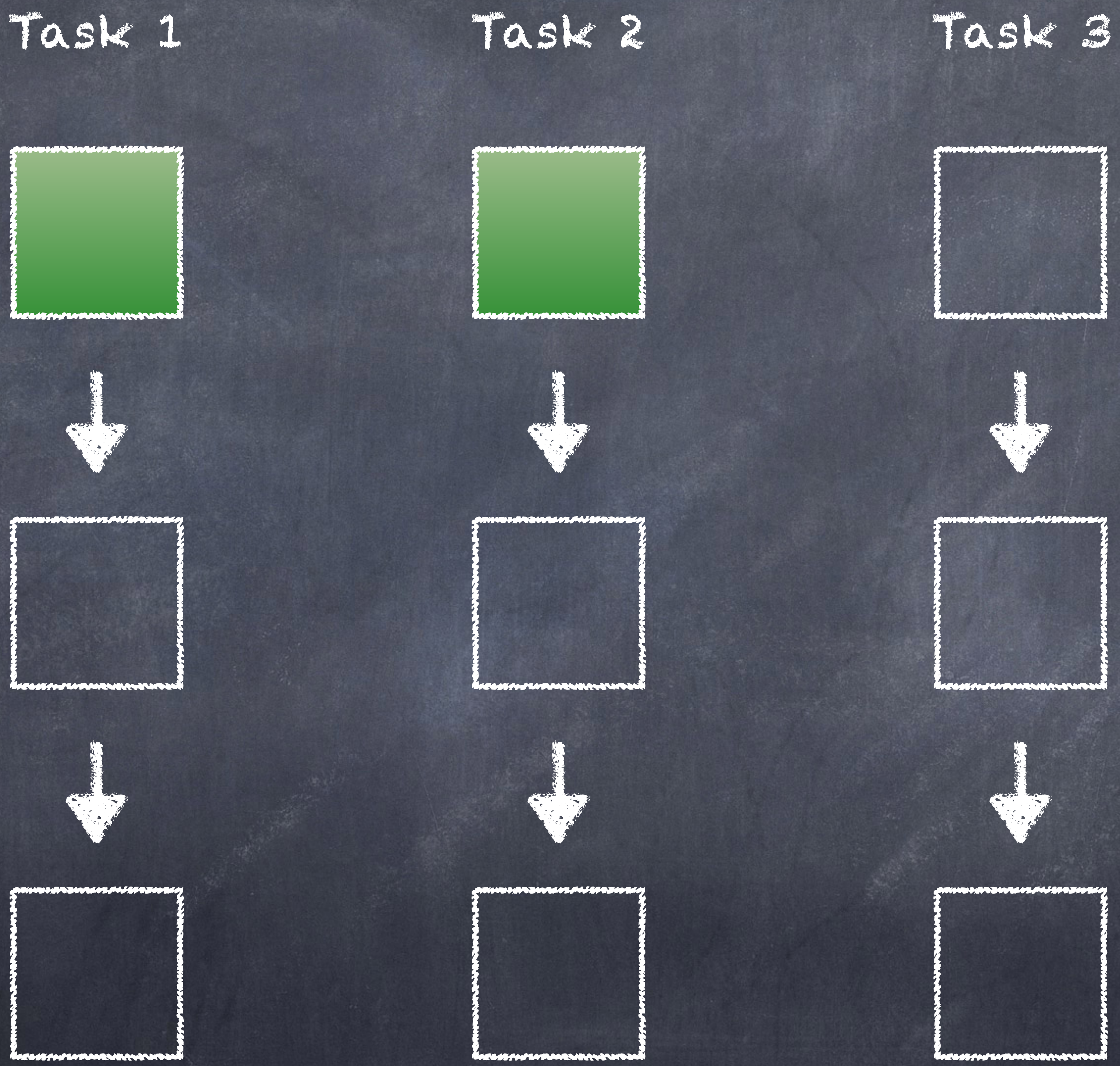
Task 2

Task 3

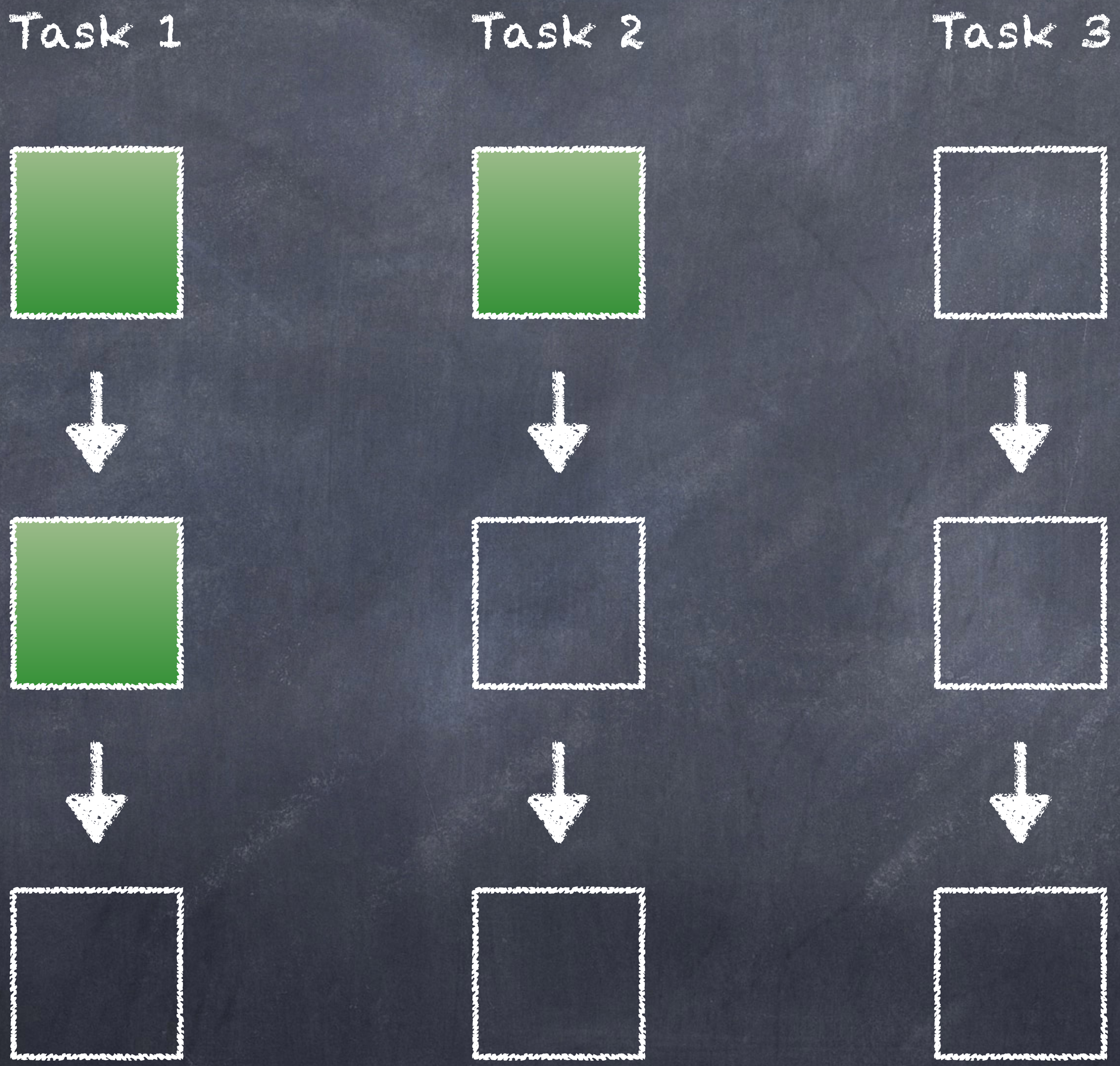
Concurrency and Parallelism



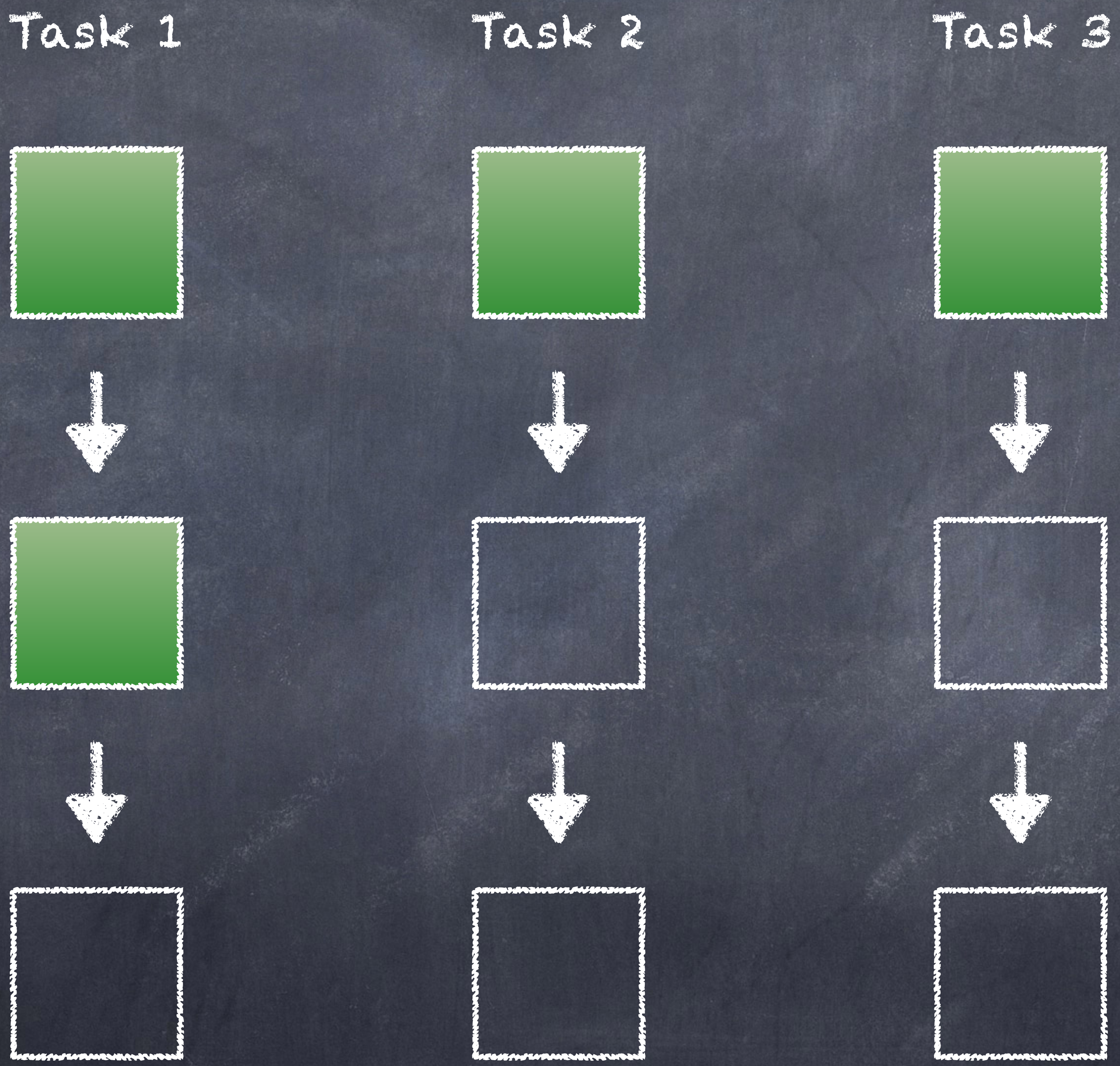
Concurrency and Parallelism



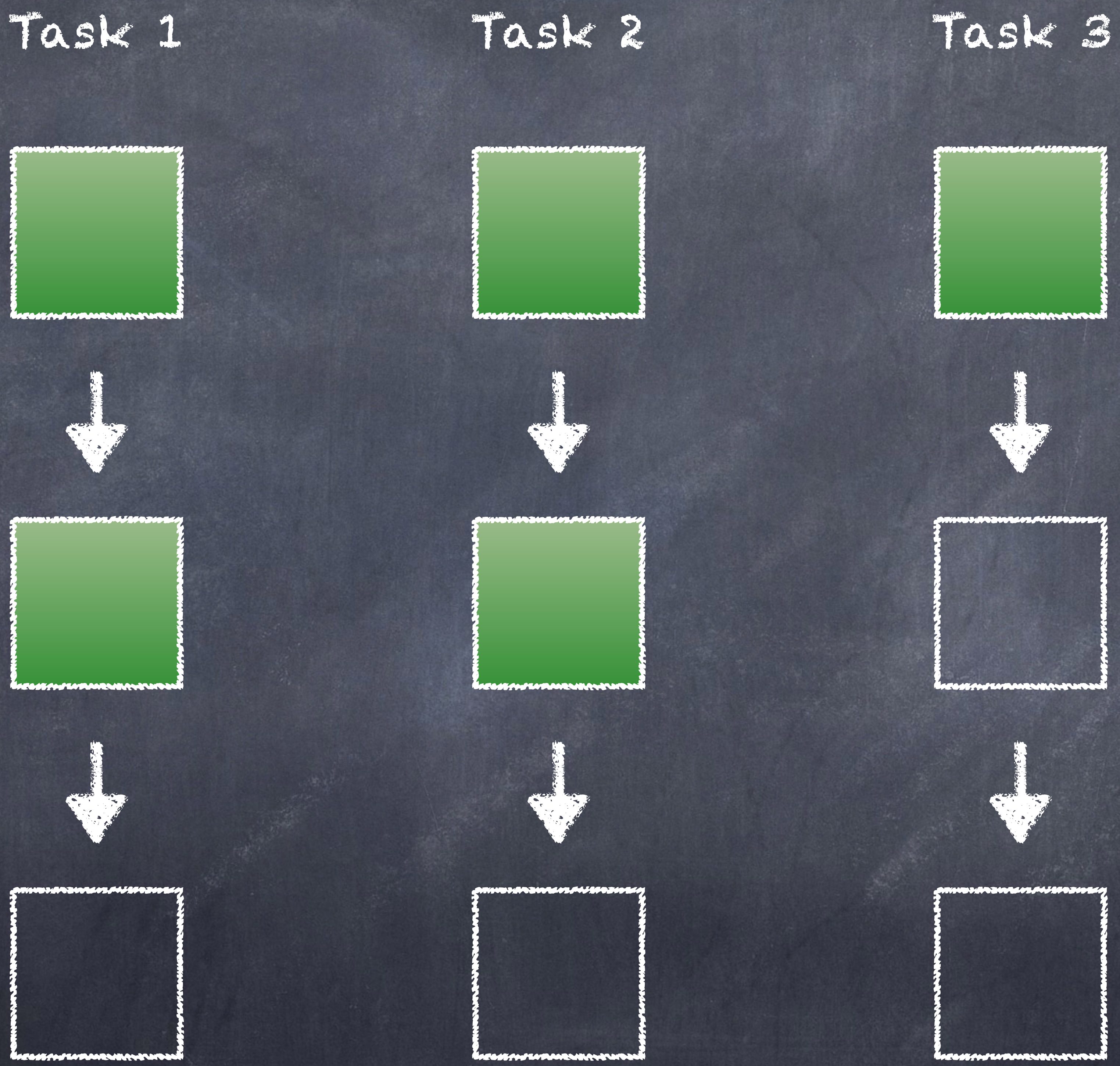
Concurrency and Parallelism



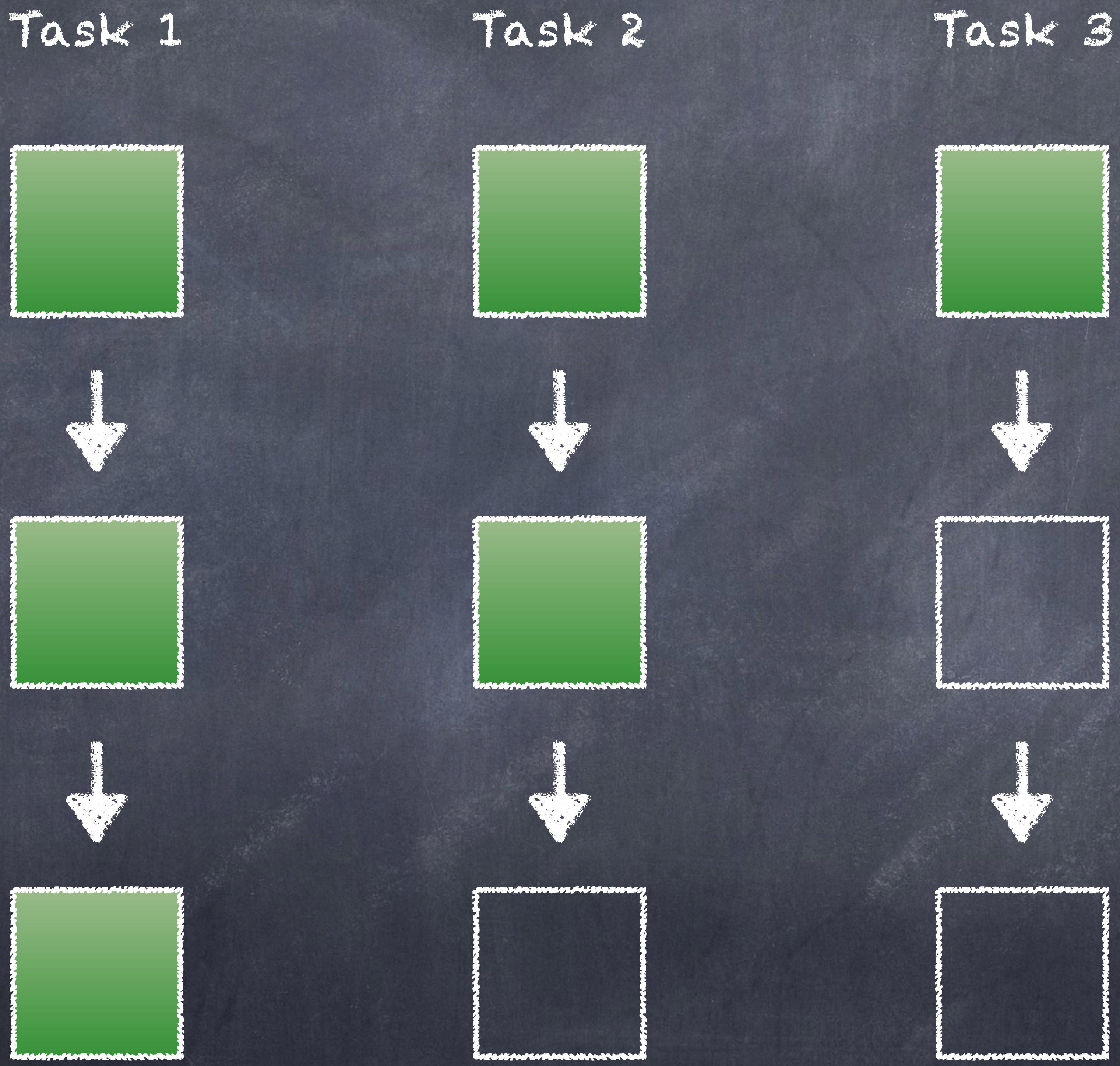
Concurrency and Parallelism



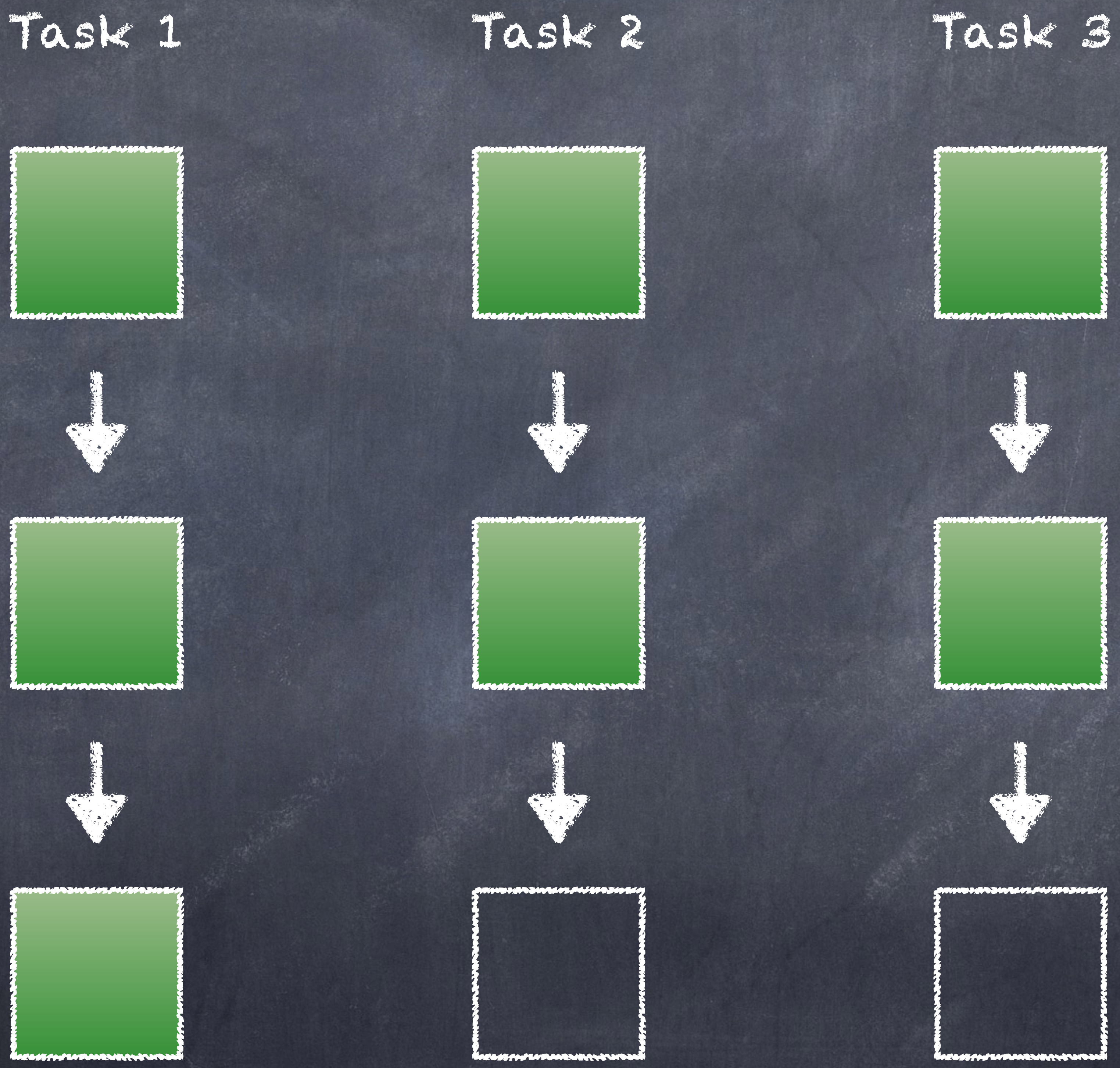
Concurrency and Parallelism



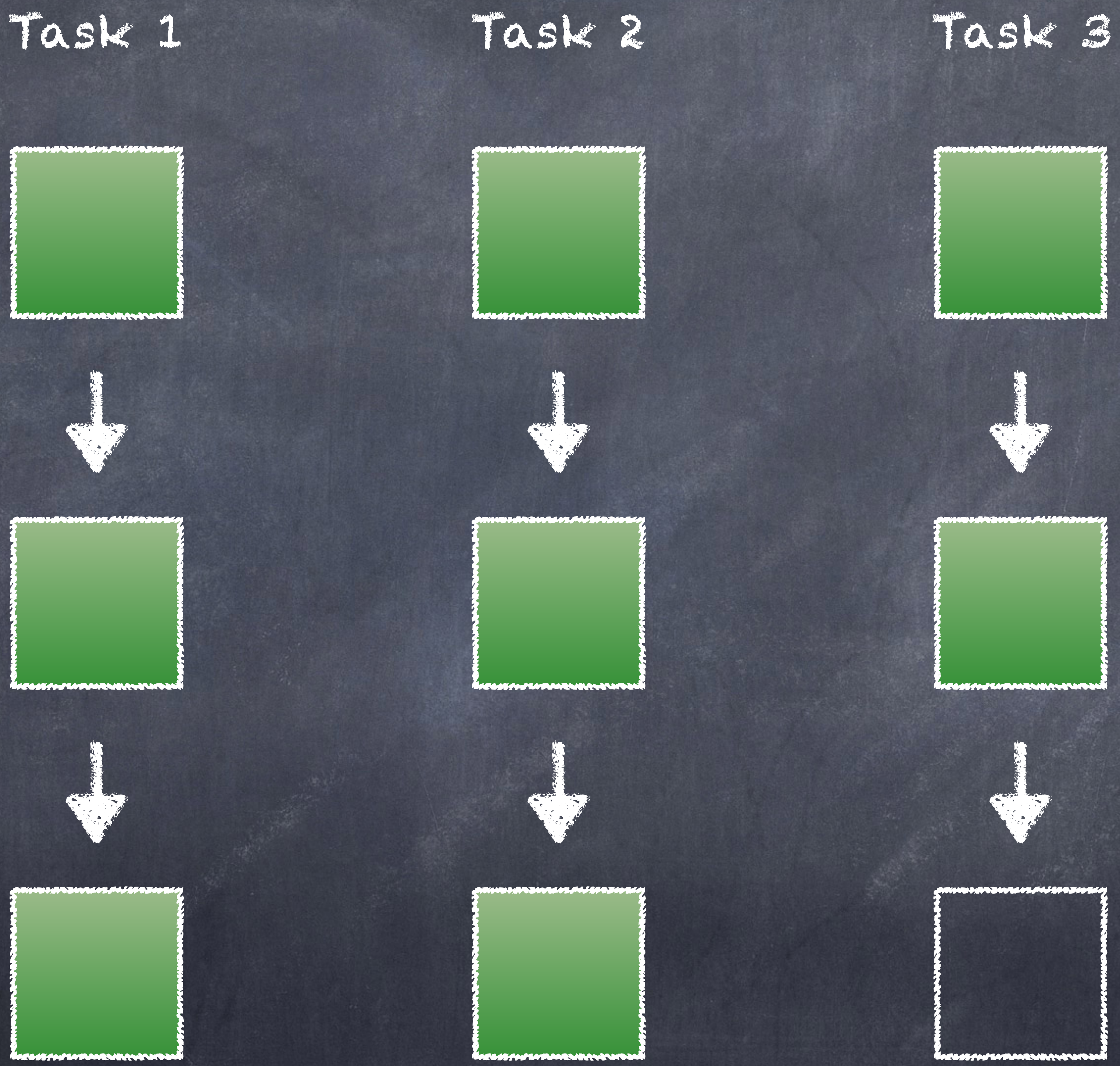
Concurrency and Parallelism



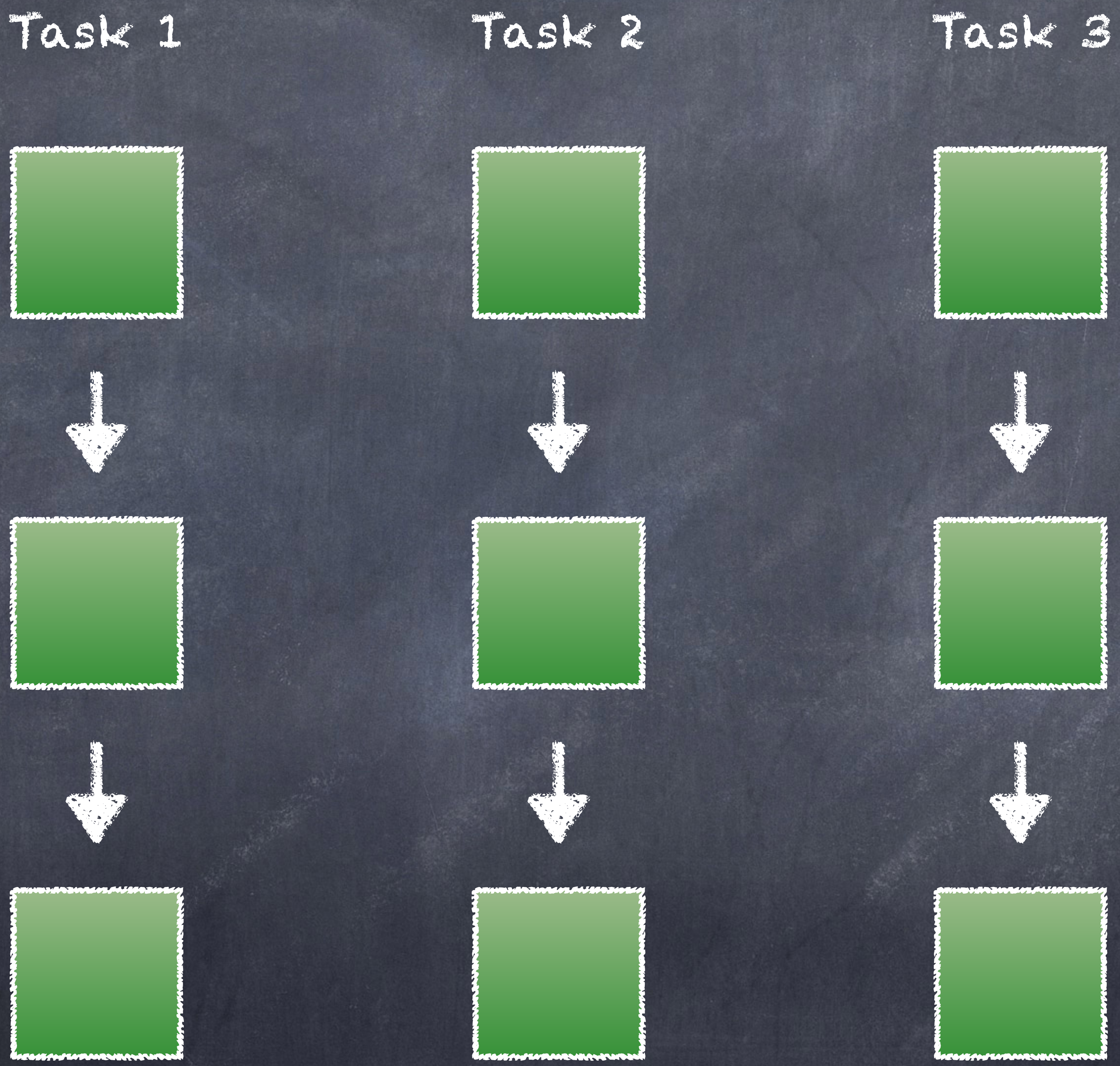
Concurrency and Parallelism



Concurrency and Parallelism



Concurrency and Parallelism

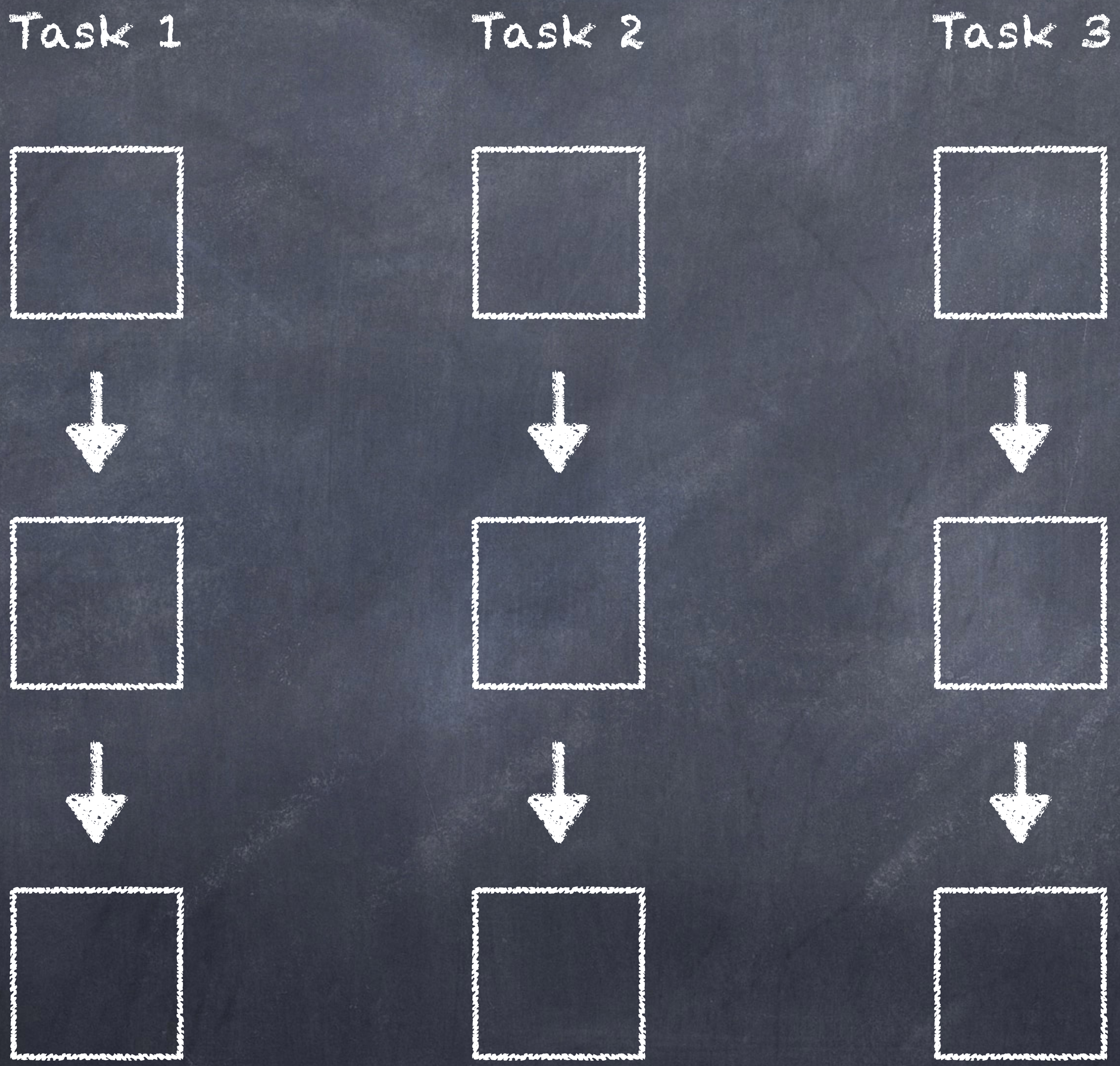


Task 1

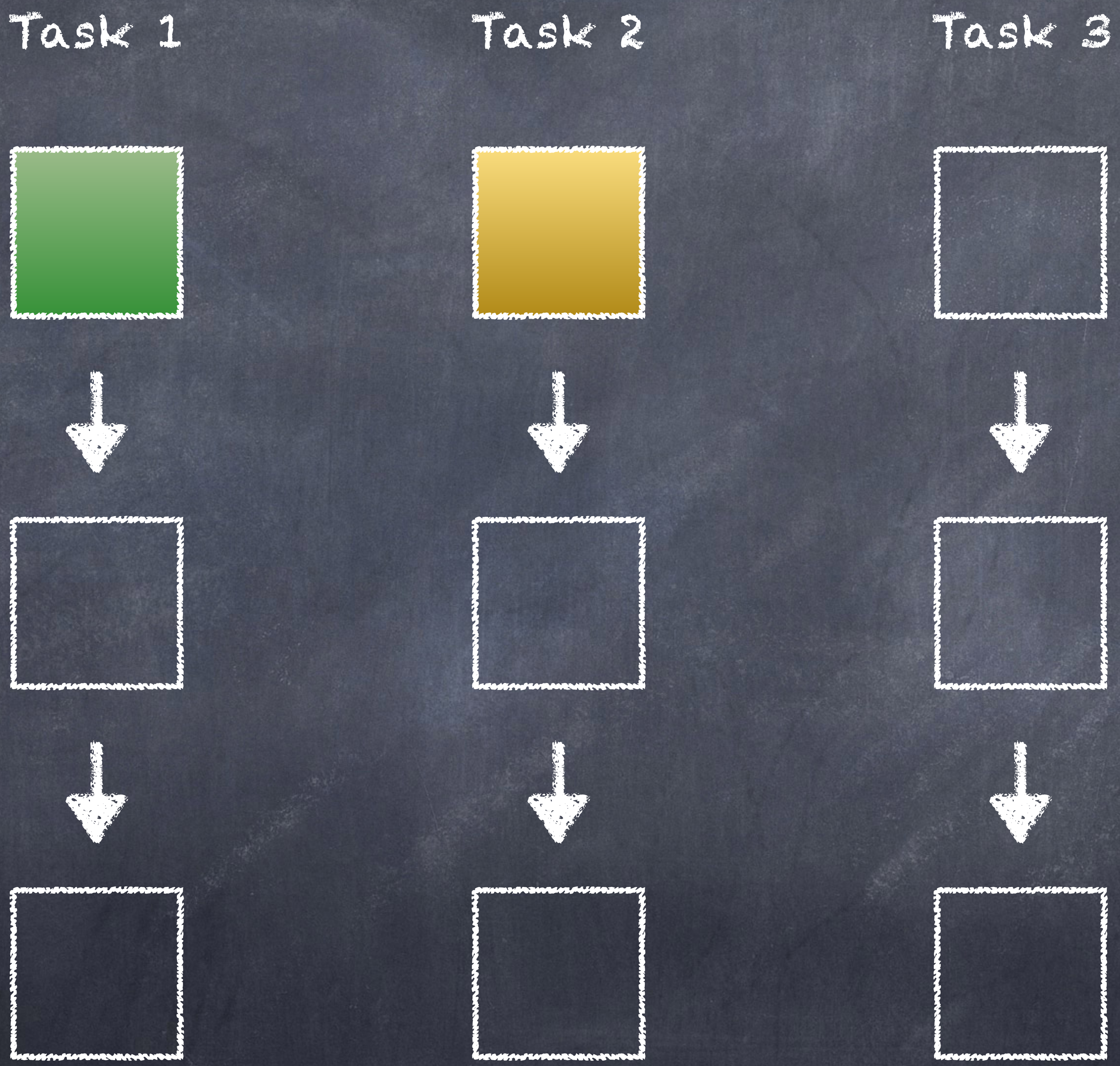
Task 2

Task 3

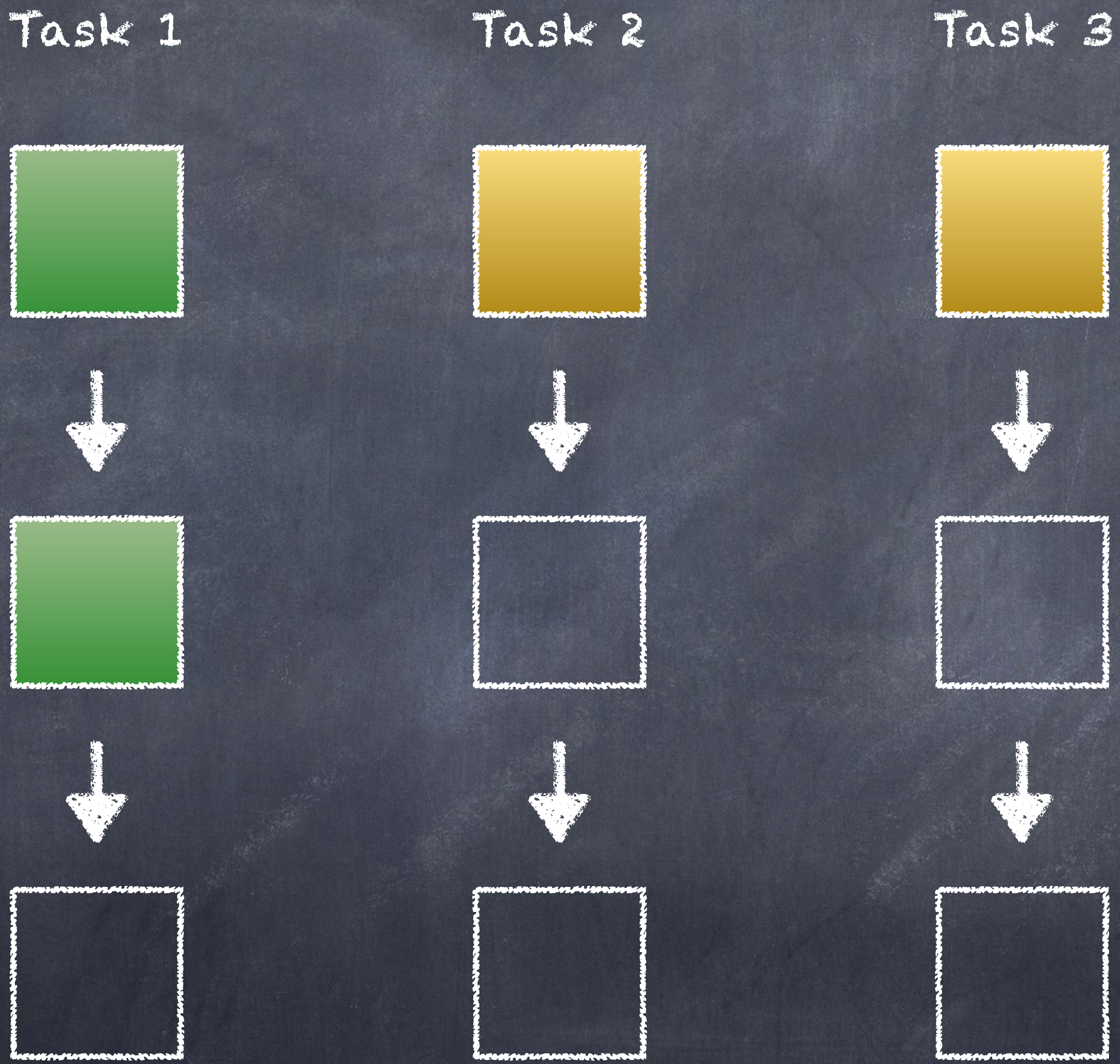
Concurrency and Parallelism



Concurrency and Parallelism



Concurrency and Parallelism

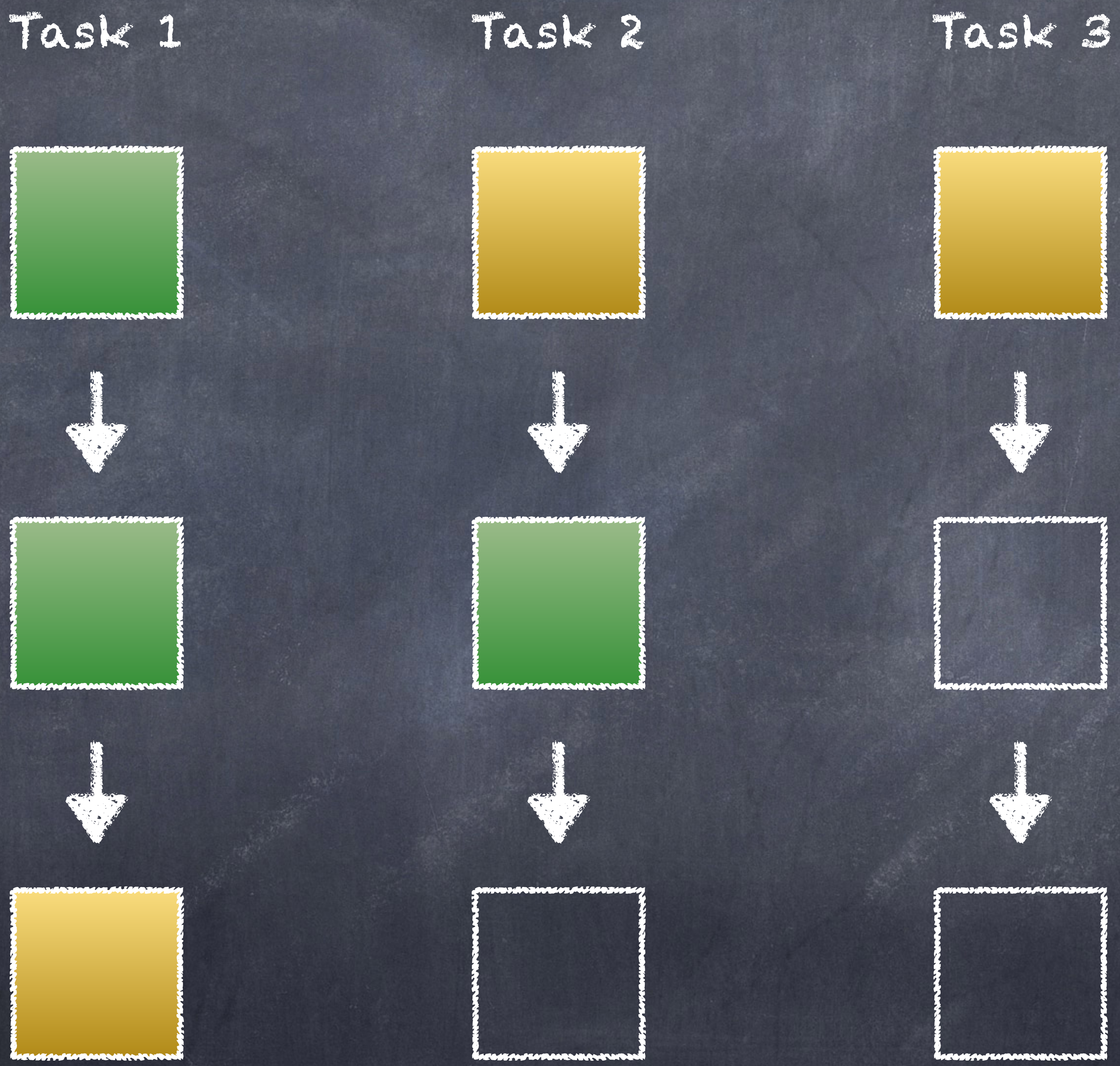


Task 1

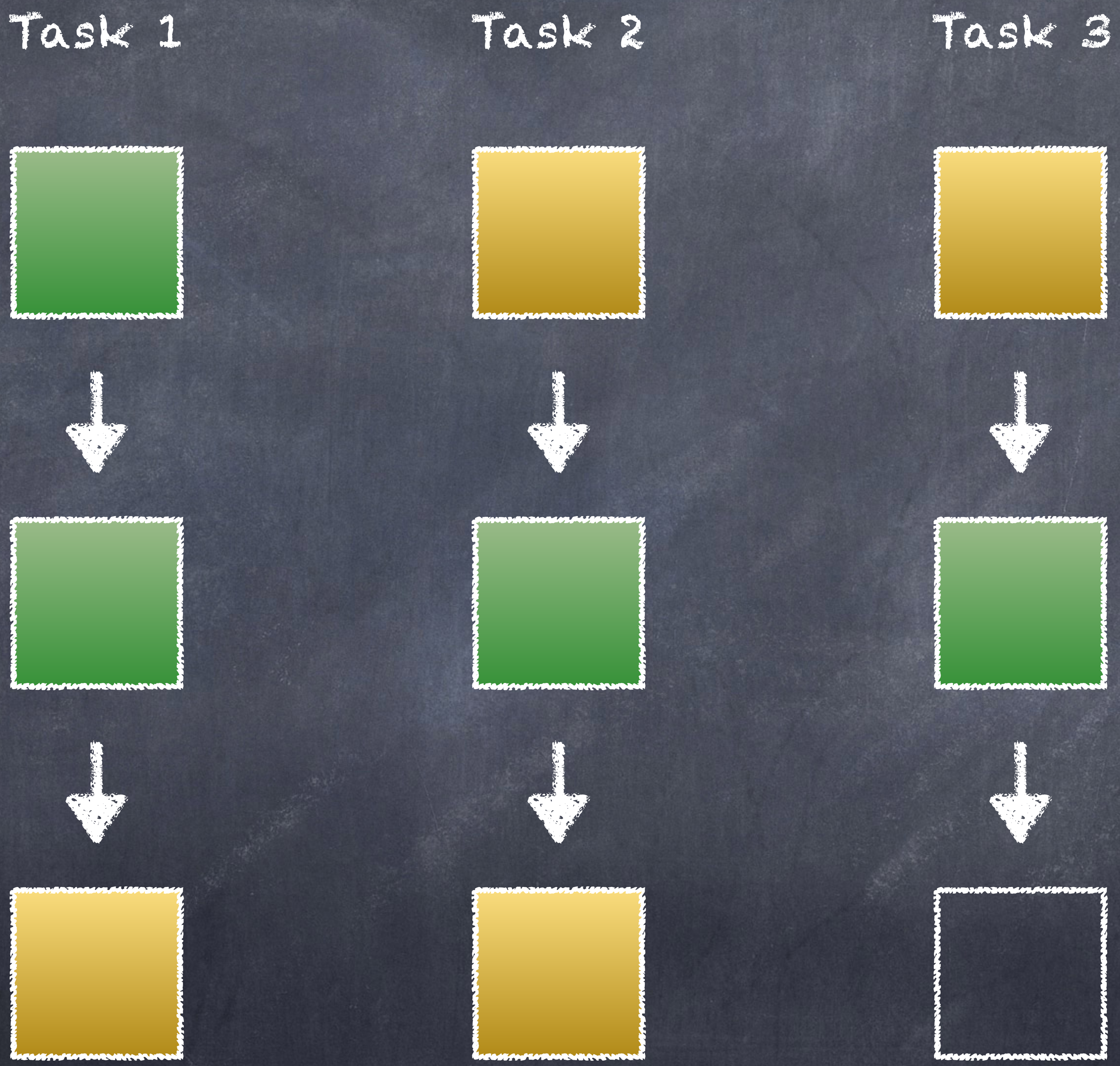
Task 2

Task 3

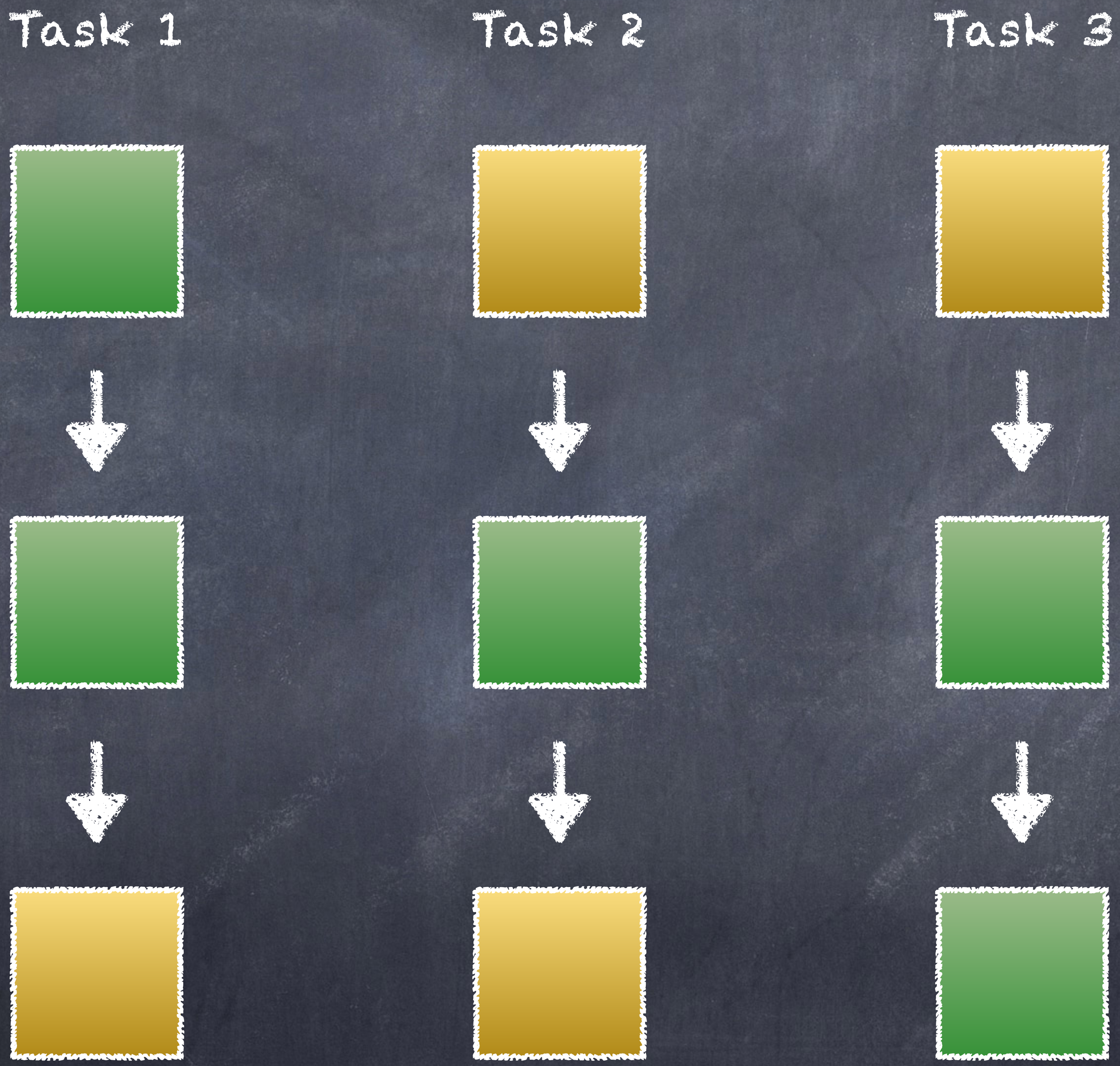
Concurrency and Parallelism



Concurrency and Parallelism



Concurrency and Parallelism



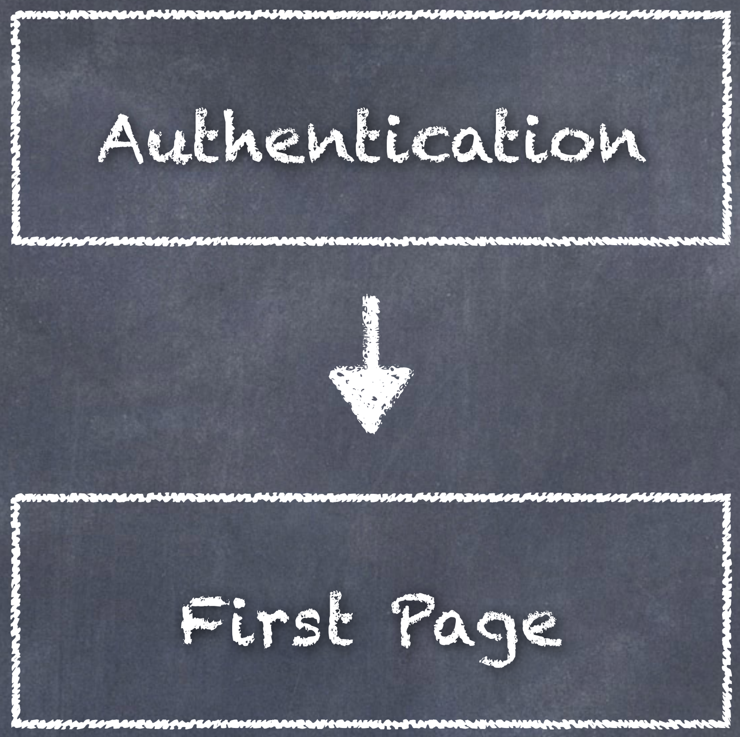
> "Simplicity is prerequisite for reliability."
>
> -Edsger W. Dijkstra

A simple scraper

A simple scraper

Authentication

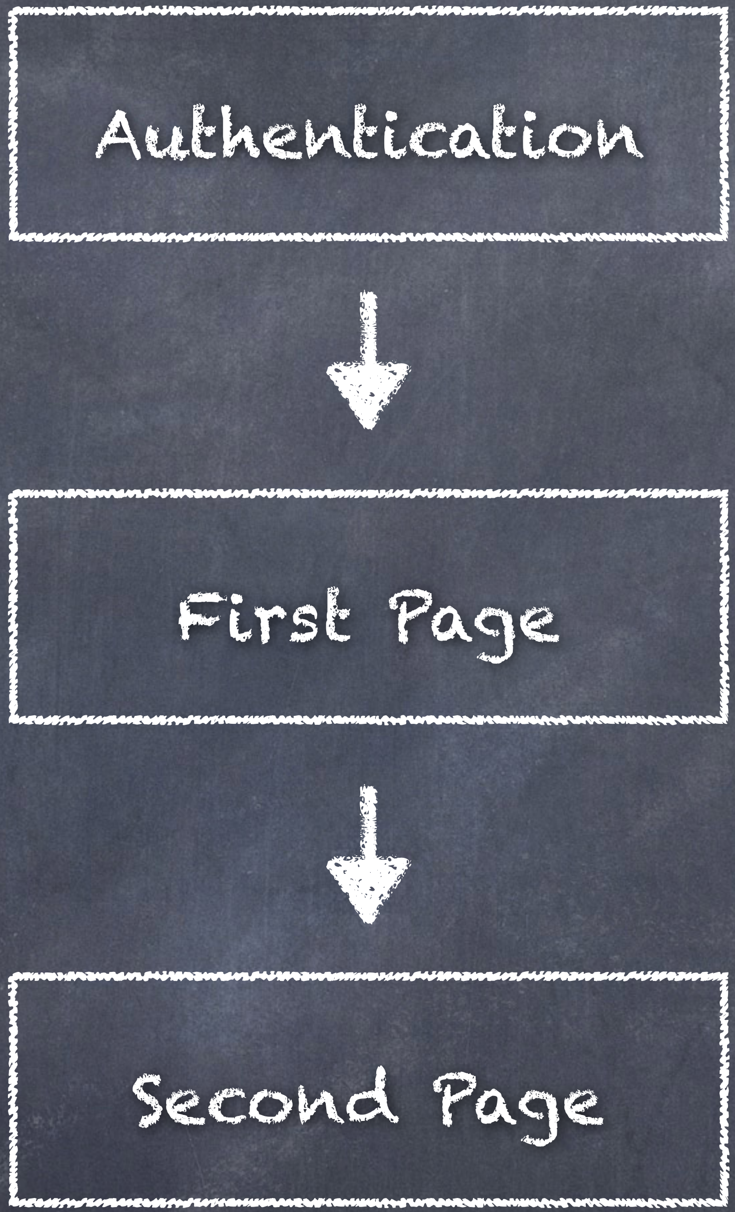
A simple scraper



Authentication

First Page

A simple scraper

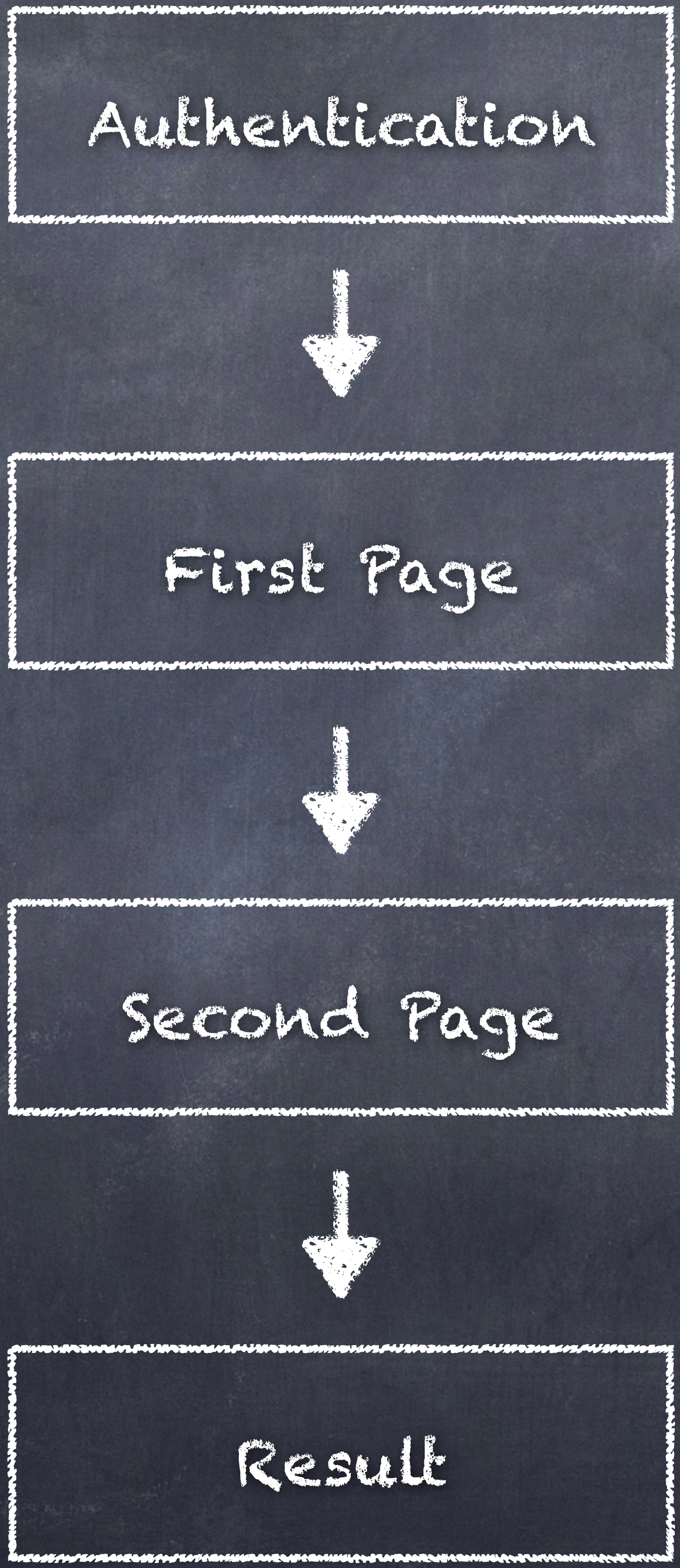


Authentication

First Page

Second Page

A simple scraper



Authentication

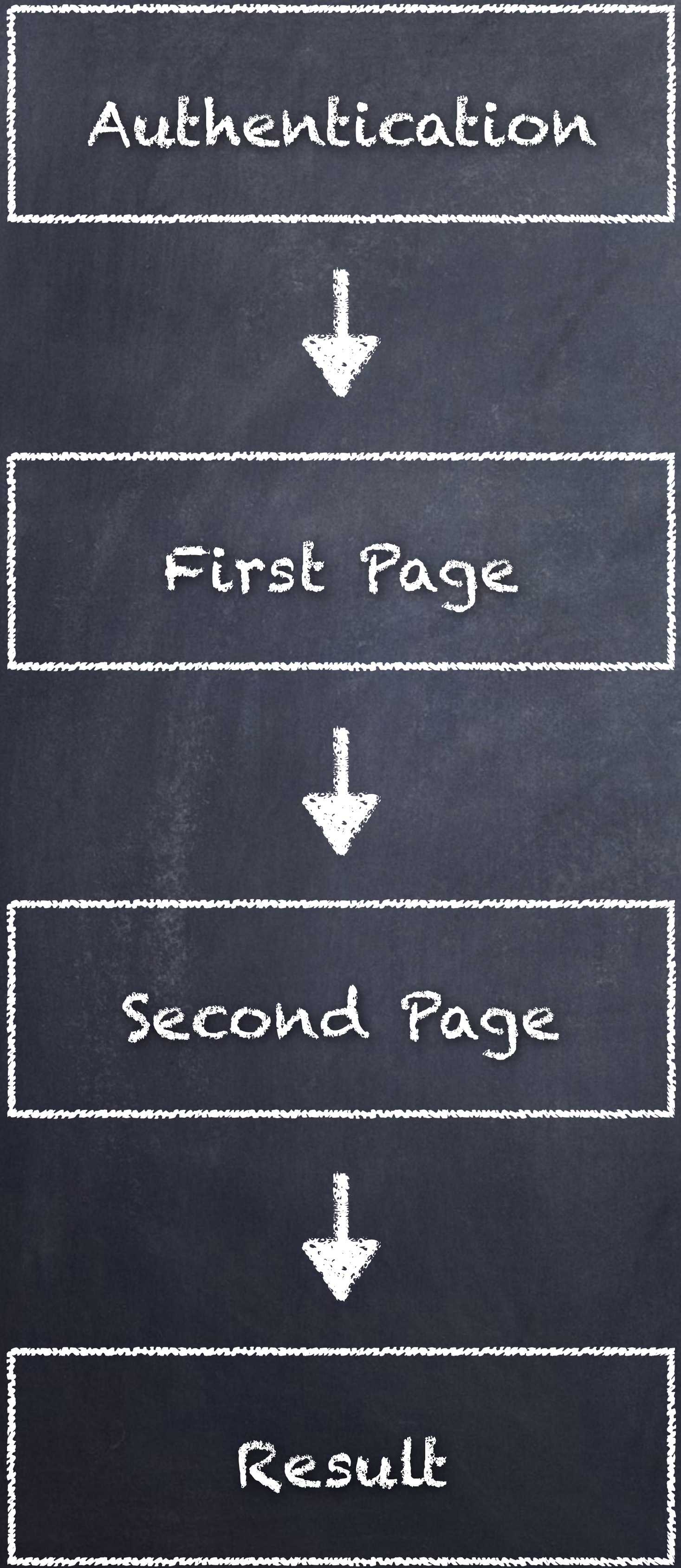
First Page

Second Page

Result

A simple scraper

A simple scraper



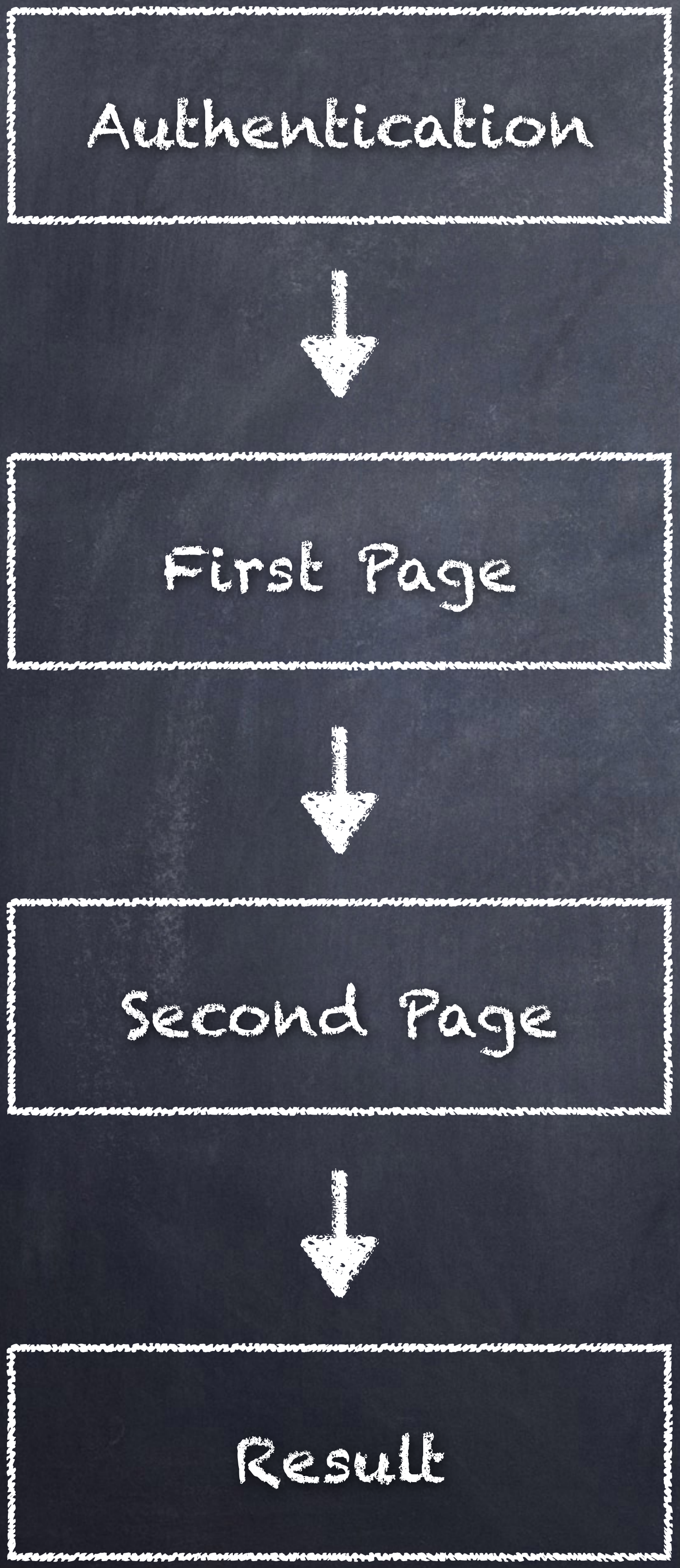
Authentication

First Page

Second Page

Result

A simple scraper



Authentication

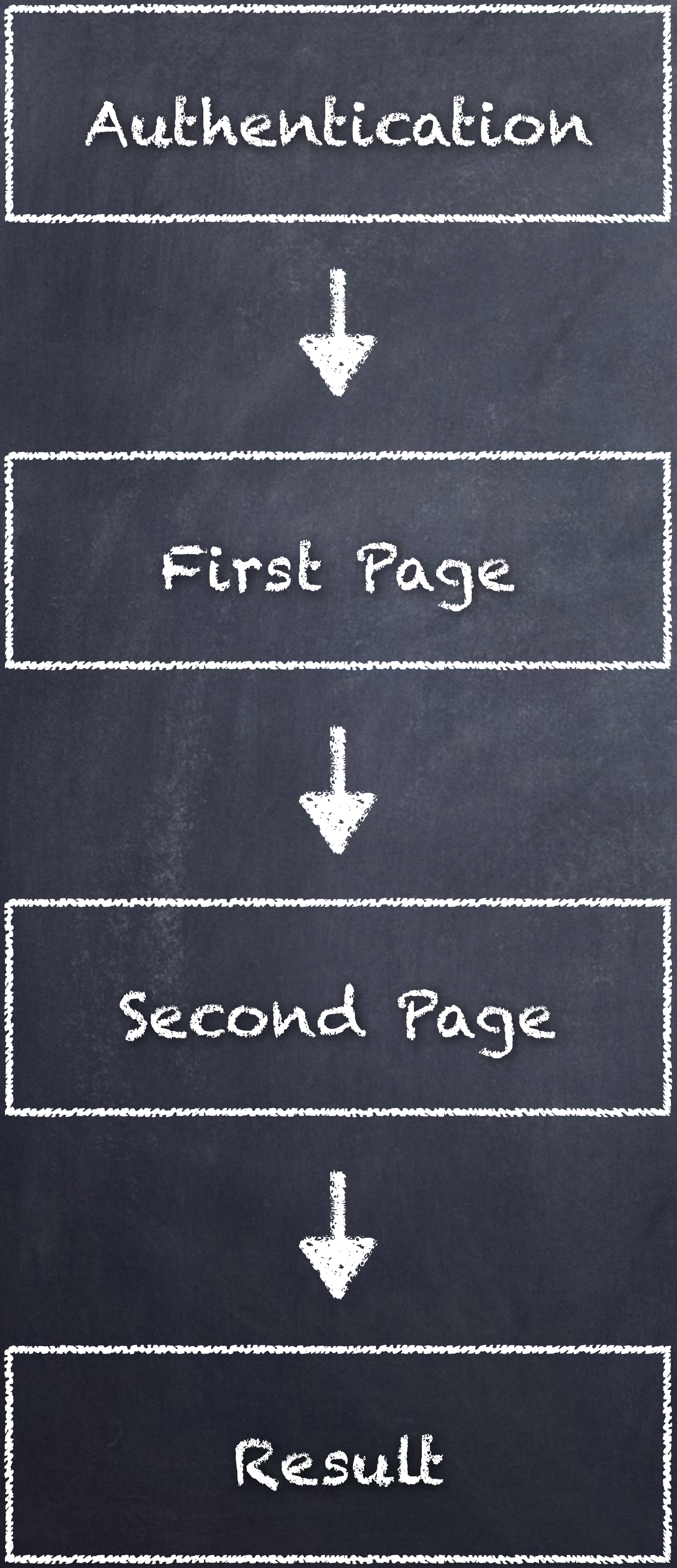
First Page

Second Page

Result

```
def authenticate(username: String, password: String): AsyncEffect[Token] = ...

def scrapePage(url: String, token: Token): AsyncEffect[PageContent] = ...

def mergePages(pages: List[PageContent]): AsyncEffect[Result] = ...
```

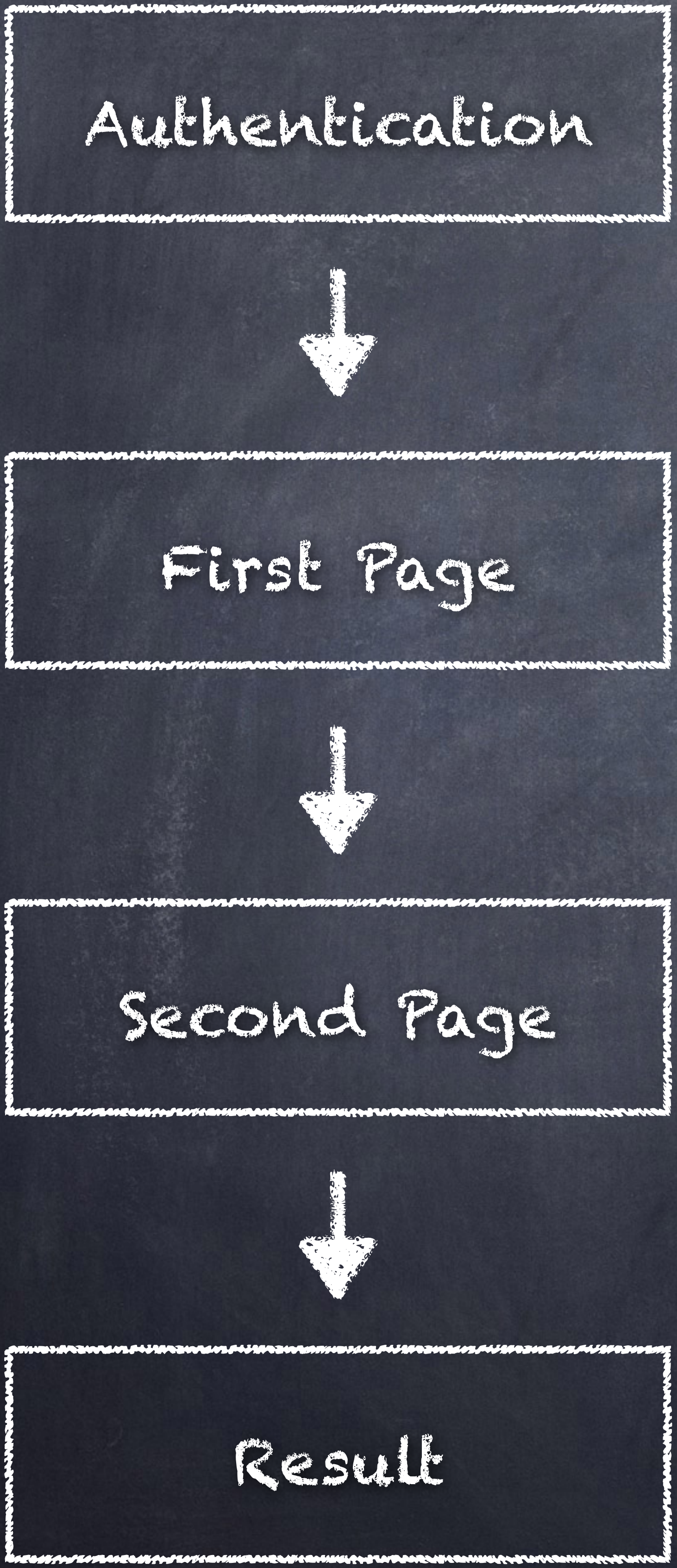
A simple scraper



```
def authenticate(username: String, password: String): AsyncEffect[Token] = ...

def scrapePage(url: String, token: Token): AsyncEffect[PageContent] = ...

def mergePages(pages: List[PageContent]): AsyncEffect[Result] = ...
```

```
authenticate("tdc", “2019").flatMap { token =>
  scrapePage("page1 ", token).flatMap { firstPage =>
    scrapePage("page2", token ).flatMap { secondPage =>
      mergePages(List(firstPage, secondPage))
    }
  }
}
```

A simple scraper



Authentication

First Page

Second Page

Result

```
def authenticate(username: String, password: String): AsyncEffect[Token] = ...

def scrapePage(url: String, token: Token): AsyncEffect[PageContent] = ...

def mergePages(pages: List[PageContent]): AsyncEffect[Result] = ...
```

```
authenticate("tdc", “2019").flatMap { token =>
    scrapePage("page1", token).flatMap { firstPage =>
        scrapePage("page2", token ).flatMap { secondPage =>
            mergePages(List(firstPage, secondPage))
        }
    }
}
```

```
for {
    token <- authenticate("tdc", "2019")
    firstPage <- scrapePage("page1", token)
    secondPage <- scrapePage("page2", token )
    result <- mergePages(List(firstPage, secondPage))
} yield result
```

Effects

Effects

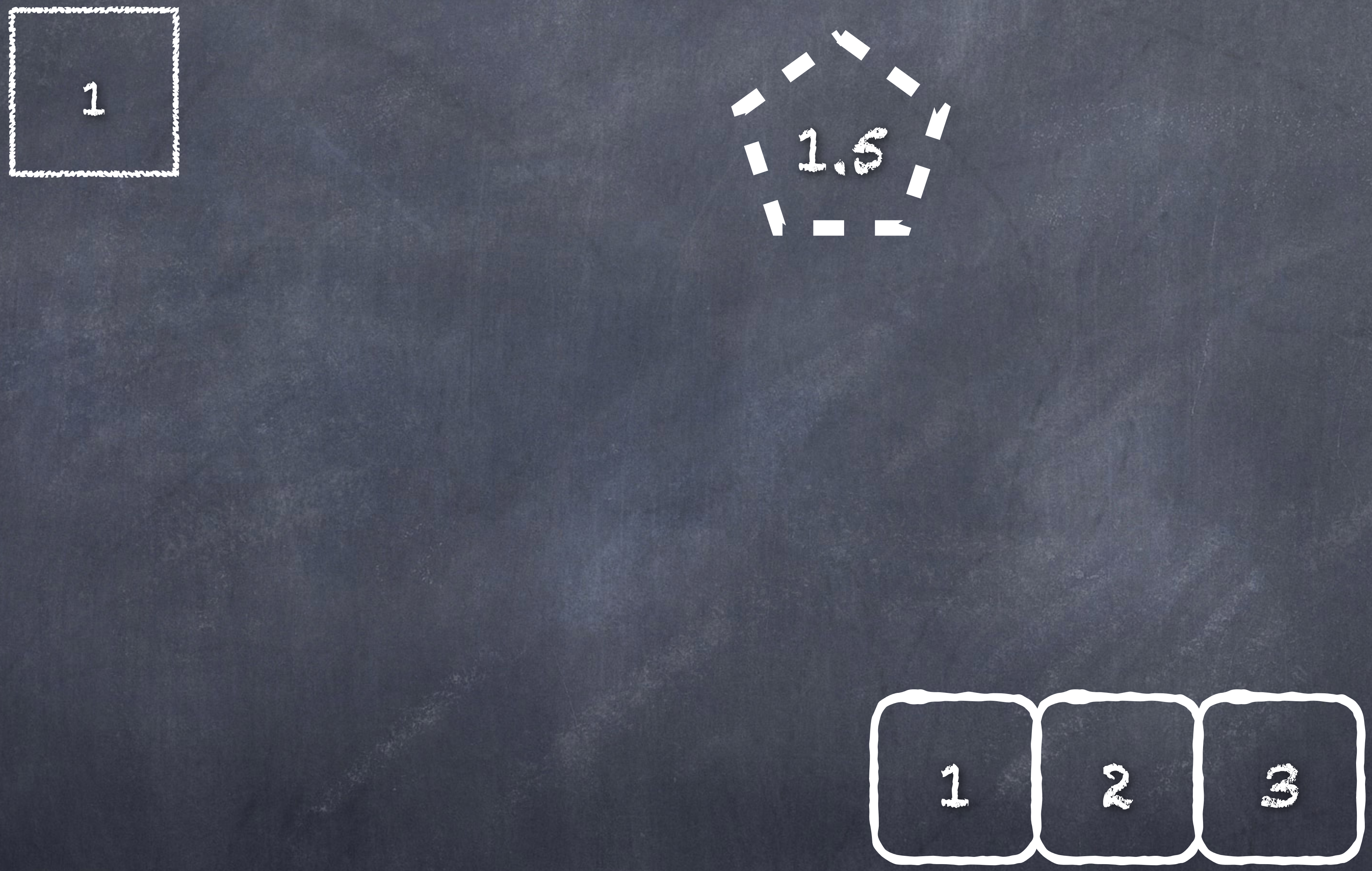
1

Effects

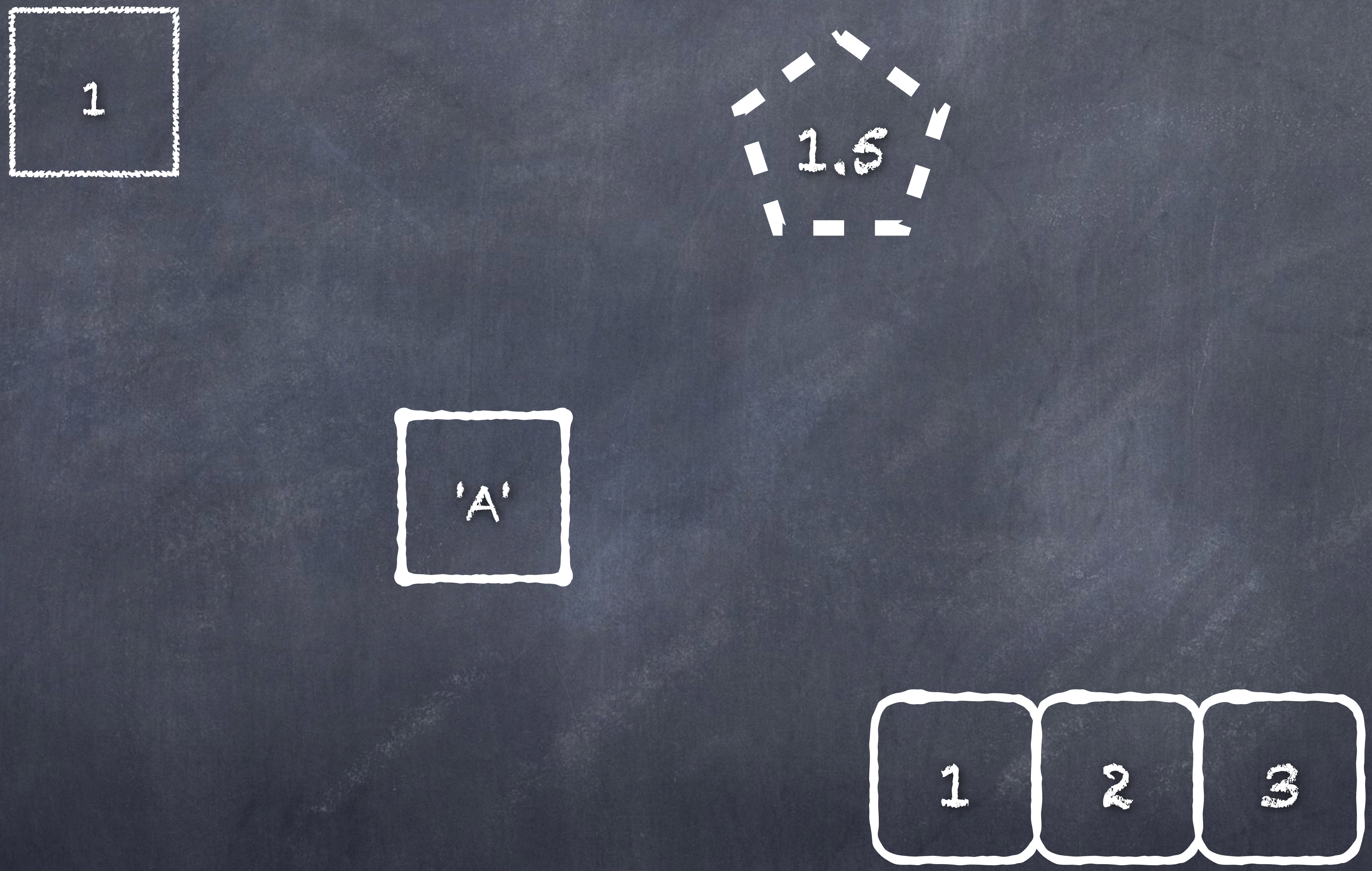
1

1 2 3

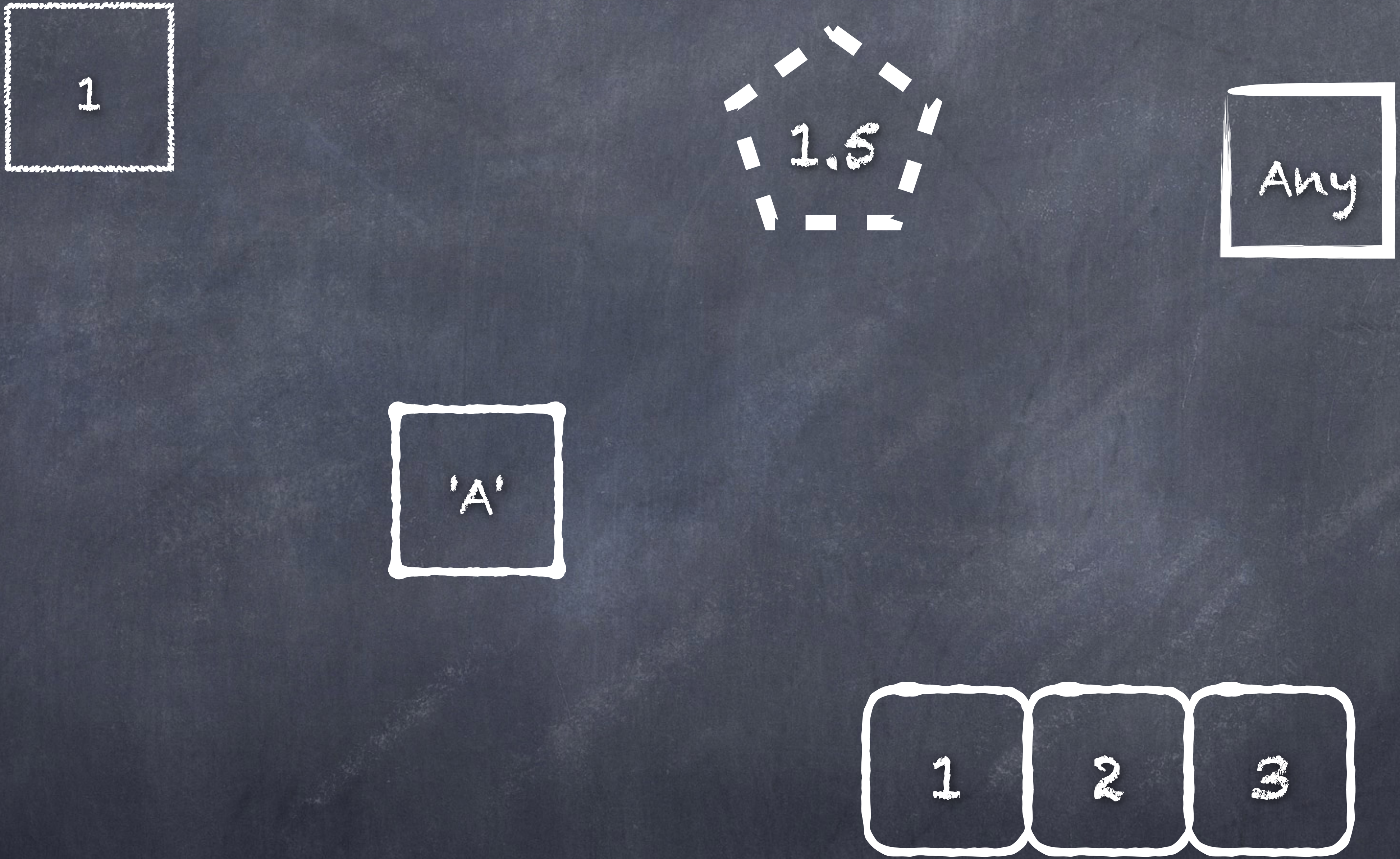
Effects



Effects



Effects



1

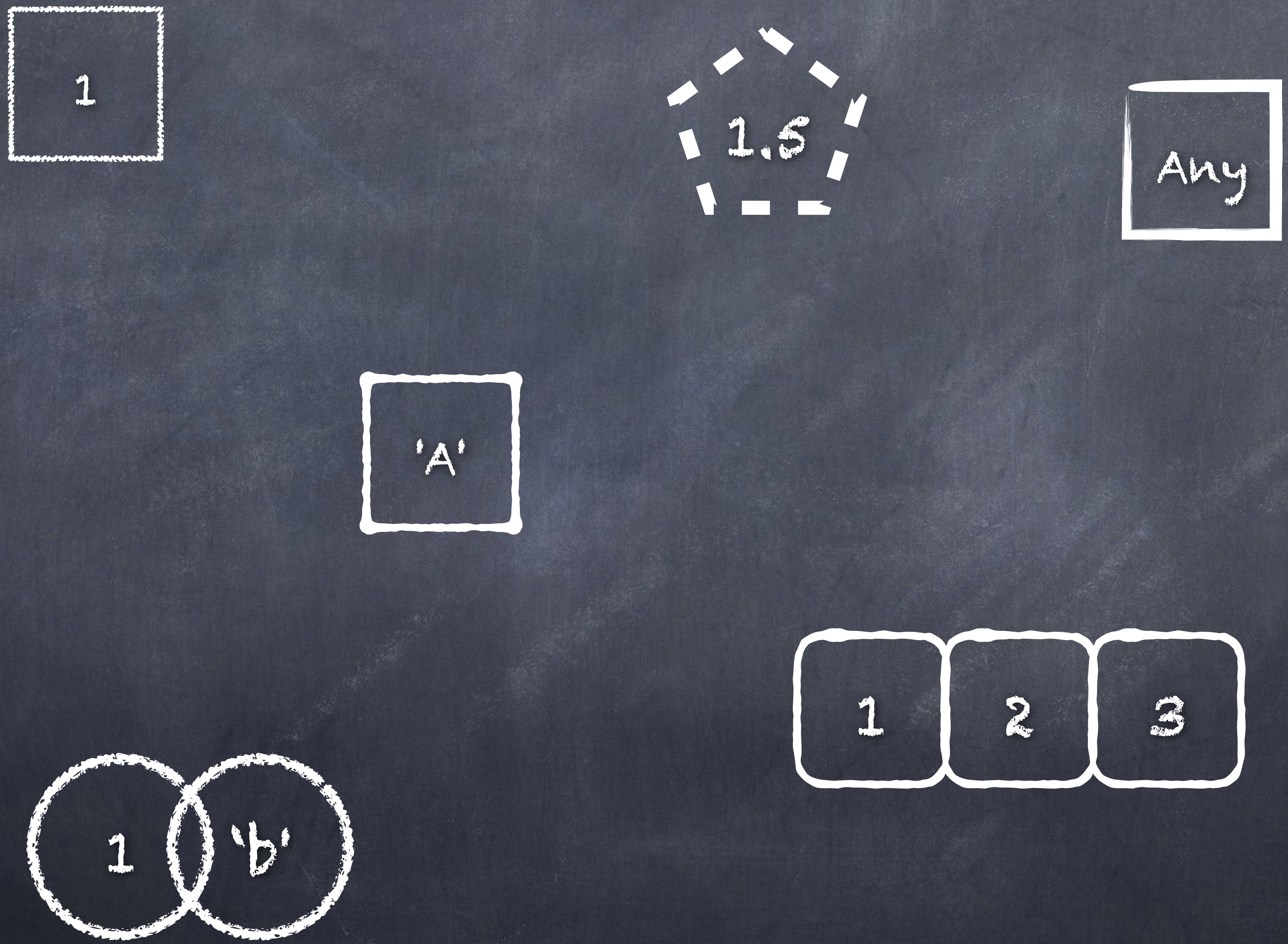
1.5

Any

'A'

1 2 3

Effects



1

1.5

Any

'A'

1 2 3

1 'b'

Effects

```
def authenticate(username: String, password: String): AsyncEffect[Token]

def scrapePage(url: String, token: Token): AsyncEffect[PageContent]

def mergePages(pages: List[PageContent]): AsyncEffect[Result]
```

```
authenticate("tdc", "2019").flatMap { token =>
  scrapePage("page1", token).flatMap { firstPage =>
    scrapePage("page2", token ).flatMap { secondPage =>
      mergePages(List(firstPage, secondPage))
    }
  }
}
```

```
for {
  token <- authenticate("tdc", "2019")
  firstPage <- scrapePage("page1", token)
  secondPage <- scrapePage("page2", token )
  result <- mergePages(List(firstPage, secondPage))
} yield result
```

ES6 Promise

```
def authenticate(username: String, password: String): Promise[Token]

def scrapePage(url: String, token: Token): Promise[PageContent]

def mergePages(pages: List[PageContent]): Promise[Result]
```

```
authenticate("tdc", “2019").then { token =>
    scrapePage("page1", token).then { firstPage =>
        scrapePage("page2", token ).then { secondPage =>
            mergePages(List(firstPage, secondPage))
        }
    }
}
```

```
for {
    token <- authenticate("tdc", "2019")
    firstPage <- scrapePage("page1", token)
    secondPage <- scrapePage("page2", token )
    result <- mergePages(List(firstPage, secondPage))
} yield result
```

Scala Promise (a.k.a Future)

```
def authenticate(username: String, password: String): Future[Token]

def scrapePage(url: String, token: Token): Future[PageContent]

def mergePages(pages: List[PageContent]): Future[Result]
```

```
authenticate("tdc", “2019").flatMap { token =>
    scrapePage("page1", token).flatMap { firstPage =>
        scrapePage("page2", token ).flatMap { secondPage =>
            mergePages(List(firstPage, secondPage))
        }
    }
}
```

```
for {
    token <- authenticate("tdc", "2019")
    firstPage <- scrapePage("page1", token)
    secondPage <- scrapePage("page2", token )
    result <- mergePages(List(firstPage, secondPage))
} yield result
```

Rx Scala

```
def authenticate(username: String, password: String): Observable[Token]

def scrapePage(url: String, token: Token): Observable[PageContent]

def mergePages(pages: List[PageContent]): Observable[Result]
```

```
authenticate("tdc", “2019").flatMap { token =>
    scrapePage("page1", token).flatMap { firstPage =>
        scrapePage("page2", token ).flatMap { secondPage =>
            mergePages(List(firstPage, secondPage))
        }
    }
}
```

```
for {
    token <- authenticate("tdc", "2019")
    firstPage <- scrapePage("page1", token)
    secondPage <- scrapePage("page2", token )
    result <- mergePages(List(firstPage, secondPage))
} yield result
```

Java/Kotlin Reactor Mono

```
def authenticate(username: String, password: String): Mono<Token>

def scrapePage(url: String, token: Token): Mono<PageContent>

def mergePages(pages: List[PageContent]): Mono<Result>
```

```
authenticate("tdc", “2019").flatMap { token =>
    scrapePage("page1", token).flatMap { firstPage =>
        scrapePage("page2", token ).flatMap { secondPage =>
            mergePages(List(firstPage, secondPage))
        }
    }
}
```

```
for {
    token <- authenticate("tdc", "2019")
    firstPage <- scrapePage("page1", token)
    secondPage <- scrapePage("page2", token )
    result <- mergePages(List(firstPage, secondPage))
} yield result
```

Scala cats-effect IO (Fibers)

```
def authenticate(username: String, password: String): IO[Token]

def scrapePage(url: String, token: Token): IO[PageContent]

def mergePages(pages: List[PageContent]): IO[Result]
```

```
authenticate("tdc", “2019").flatMap { token =>
    scrapePage("page1", token).flatMap { firstPage =>
        scrapePage("page2", token ).flatMap { secondPage =>
            mergePages(List(firstPage, secondPage))
        }
    }
}
```

```
for {
    token <- authenticate("tdc", "2019")
    firstPage <- scrapePage("page1", token)
    secondPage <- scrapePage("page2", token )
    result <- mergePages(List(firstPage, secondPage))
} yield result
```

Scala ZIO (Fibers)

```
def authenticate(username: String, password: String): ZIO[R, E, Token]

def scrapePage(url: String, token: Token): ZIO[R, E, PageContent]

def mergePages(pages: List[PageContent]): ZIO[R, E, Result]
```

```
authenticate("tdc", “2019").flatMap { token =>
    scrapePage("page1", token).flatMap { firstPage =>
        scrapePage("page2", token ).flatMap { secondPage =>
            mergePages(List(firstPage, secondPage))
        }
    }
}
```

```
for {
    token <- authenticate("tdc", "2019")
    firstPage <- scrapePage("page1", token)
    secondPage <- scrapePage("page2", token )
    result <- mergePages(List(firstPage, secondPage))
} yield result
```

Haskell IO

Haskell MVar

Haskell STM

Scala ZIO STM

Scala ZIO Ref

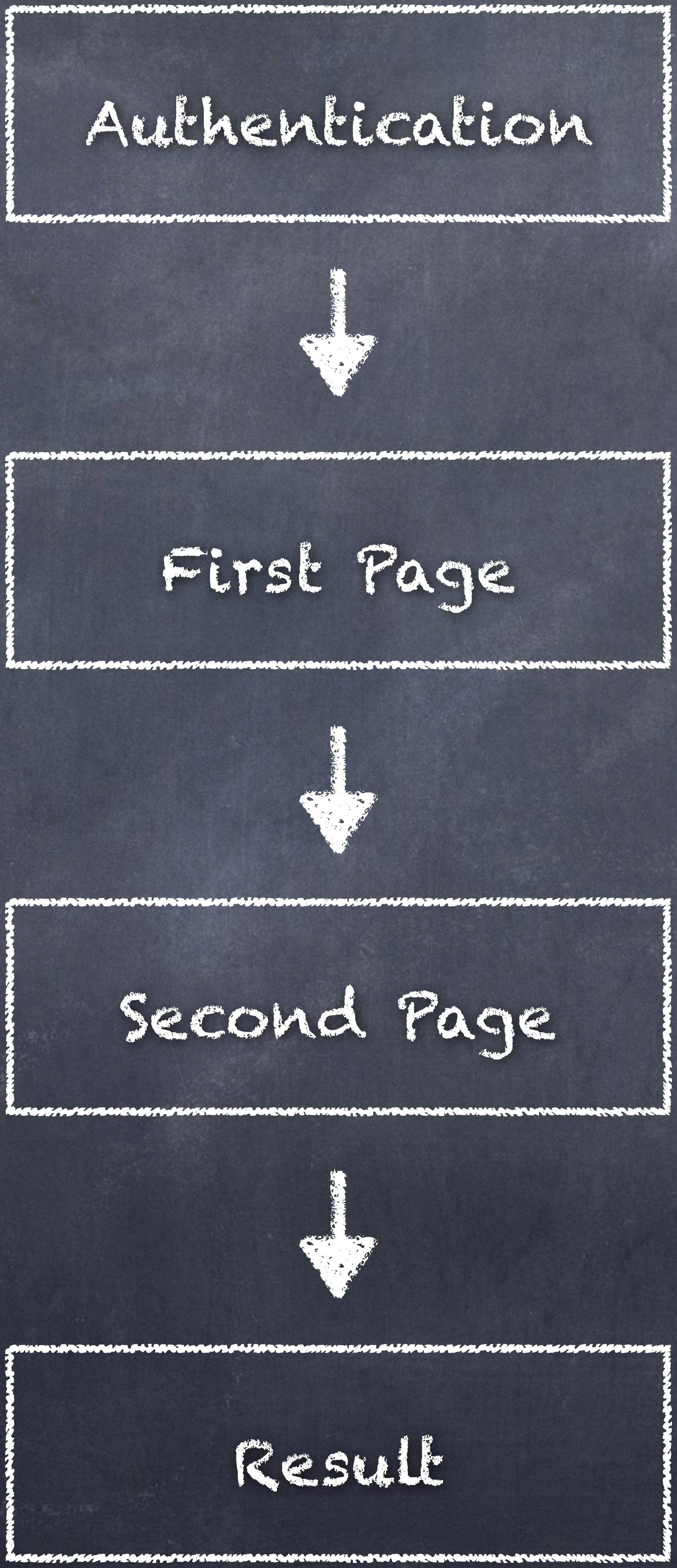
Scala ZIO Stream

Scala Twitter Future

Scala Monix

Optimizing the scraper

Optimizing the scraper



Authentication

First Page

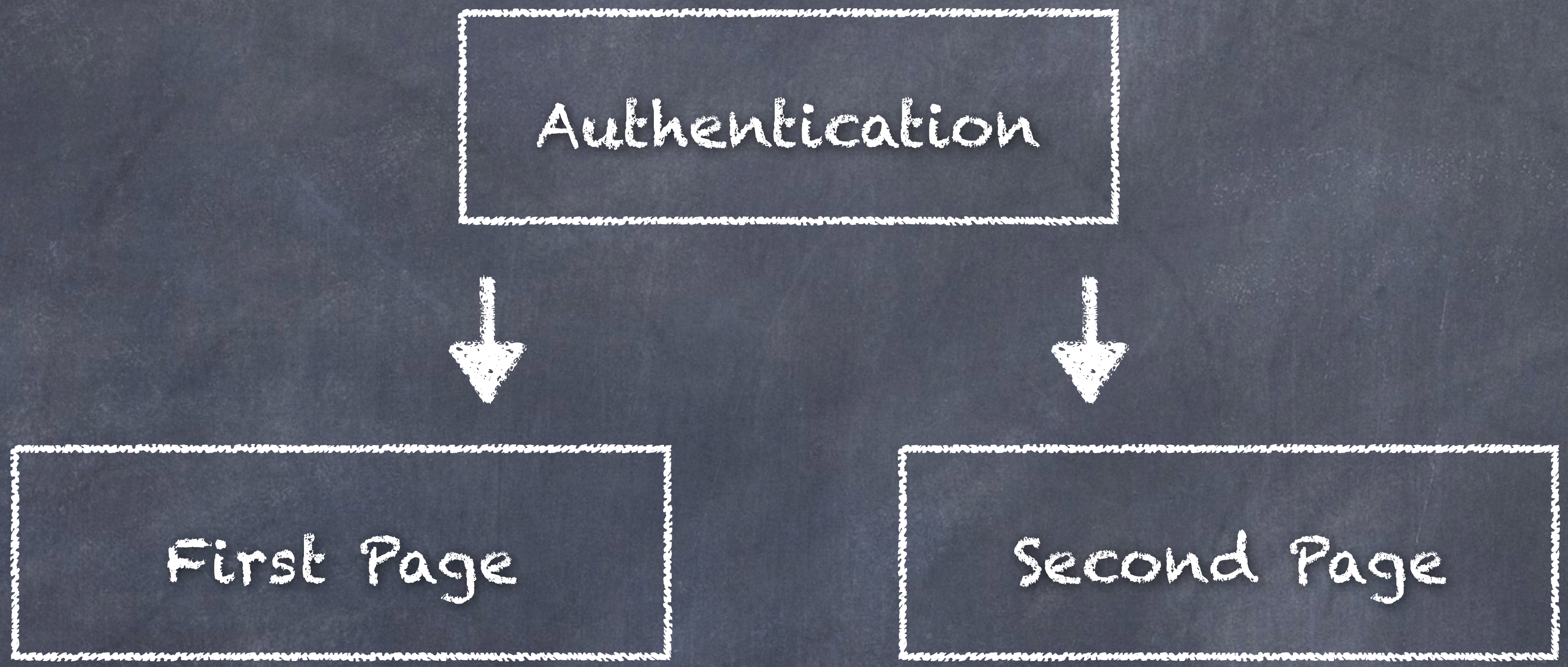
Second Page

Result

Optimizing the scraper

Authentication

Optimizing the scraper

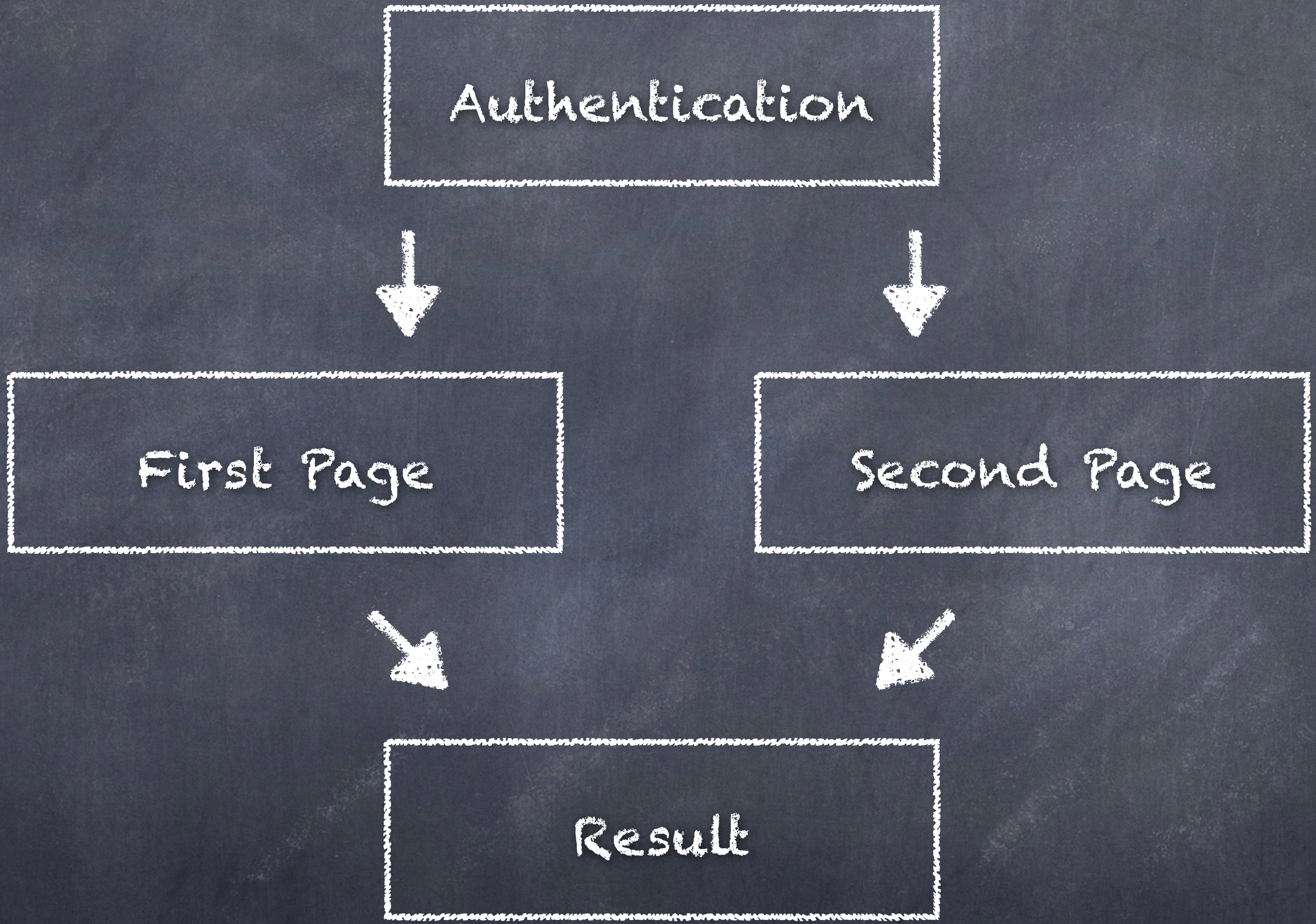


Authentication

First Page

Second Page

Optimizing the scraper



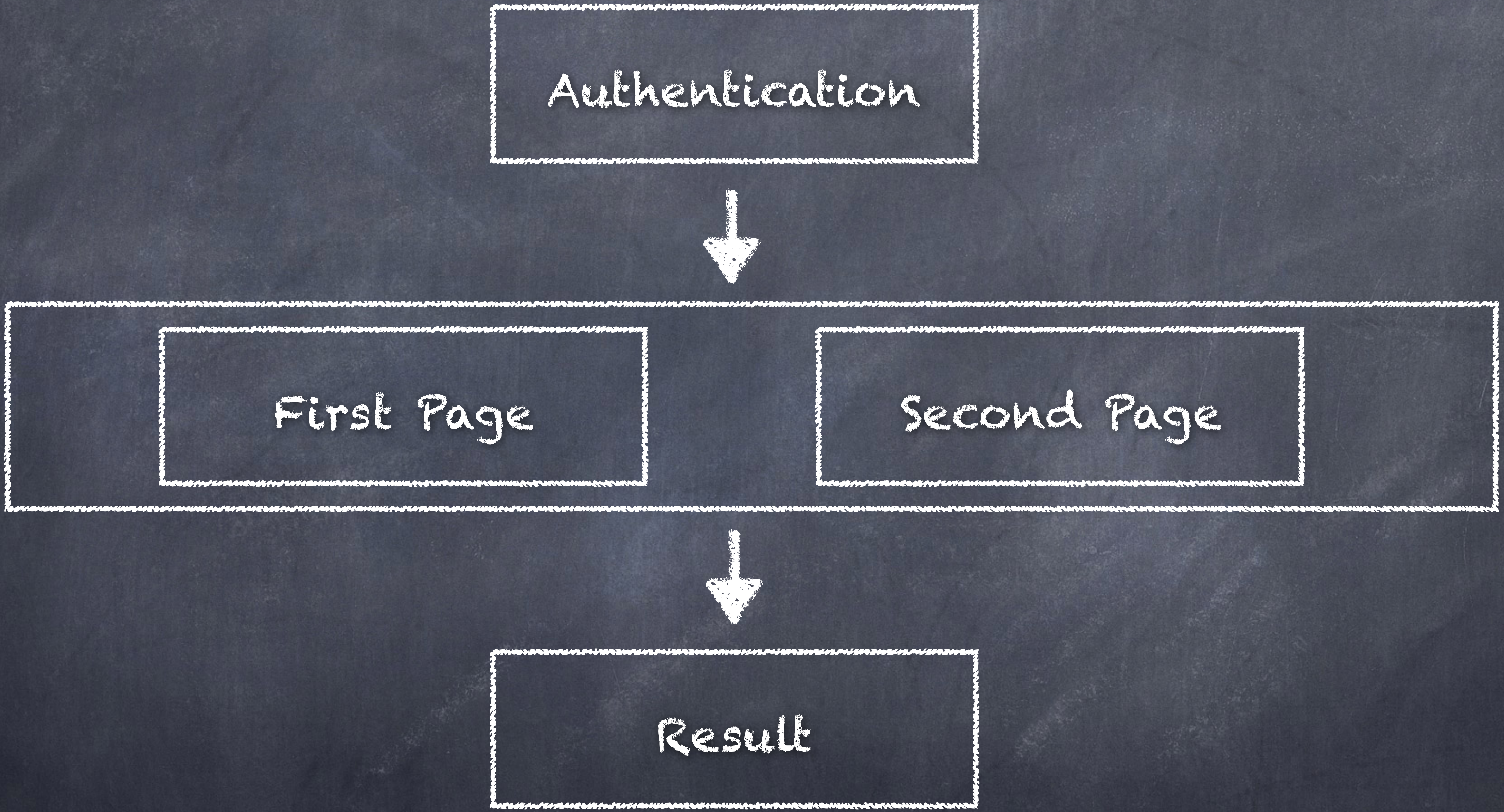
Authentication

First Page

Second Page

Result

Optimized scraper



Parallelizing the scraper

Parallelizing the scraper

```
def authenticate(username: String, password: String): AsyncEffect[Token]

def scrapePage(url: String, token: Token): AsyncEffect[PageContent]

def mergePages(pages: List[PageContent]): AsyncEffect[Result]
```

Parallelizing the scraper

```scala
def authenticate(username: String, password: String): AsyncEffect[Token]

def scrapePage(url: String, token: Token): AsyncEffect[PageContent]

def mergePages(pages: List[PageContent]): AsyncEffect[Result]
```

```scala
authenticate("tdc", "2019").flatMap { token =>
  (scrapePage("page1", token) zip scrapePage("page2", token))
   .flatMap { tuple =>
       mergePages(tuple.productIterator.toList)
    }
  }
```

Parallelizing the scraper

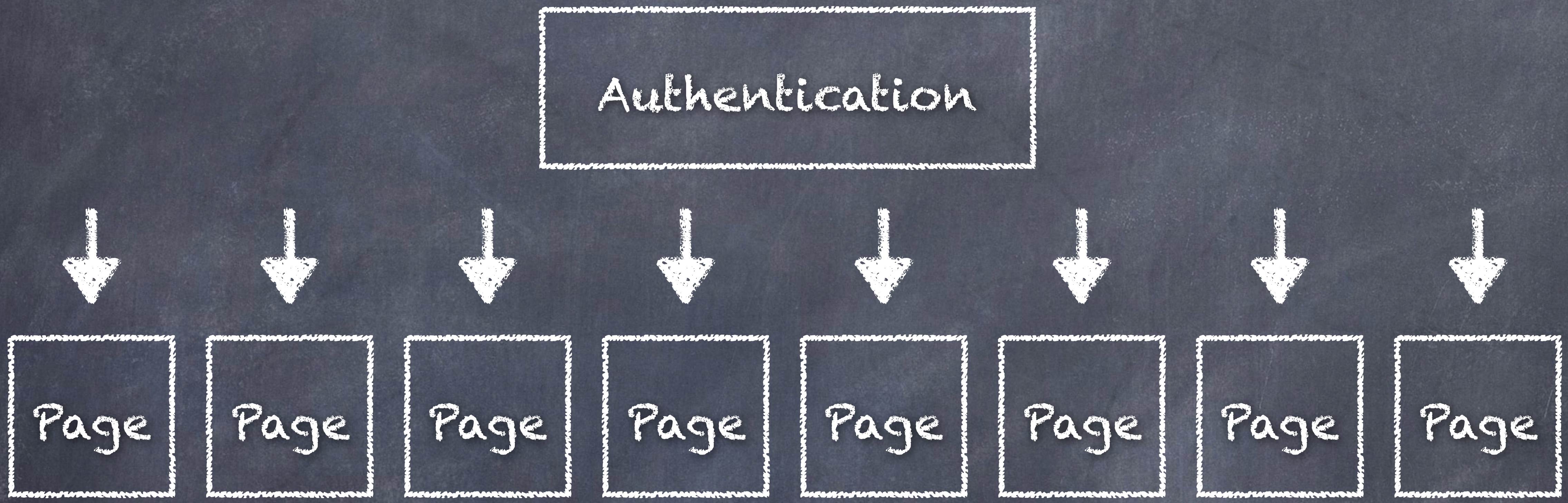
```
def authenticate(username: String, password: String): AsyncEffect[Token]

def scrapePage(url: String, token: Token): AsyncEffect[PageContent]

def mergePages(pages: List[PageContent]): AsyncEffect[Result]


    authenticate("tdc", "2019").flatMap { token =>
      (scrapePage("page1", token) zip scrapePage("page2", token))
       .flatMap { tuple =>
          mergePages(tuple.productIterator.toList)
        }
      }


for {
  token <- authenticate("tdc", "2019")
  tuple <-  scrapePage("page1", token) zip scrapePage("page2", token )
  result <- mergePages(tuple.productIterator.toList)
} yield result
```

Massive Parallelism

Massive Parallelism

Authentication

Massive Parallelism



Authentication

Page

Page

Page

Page

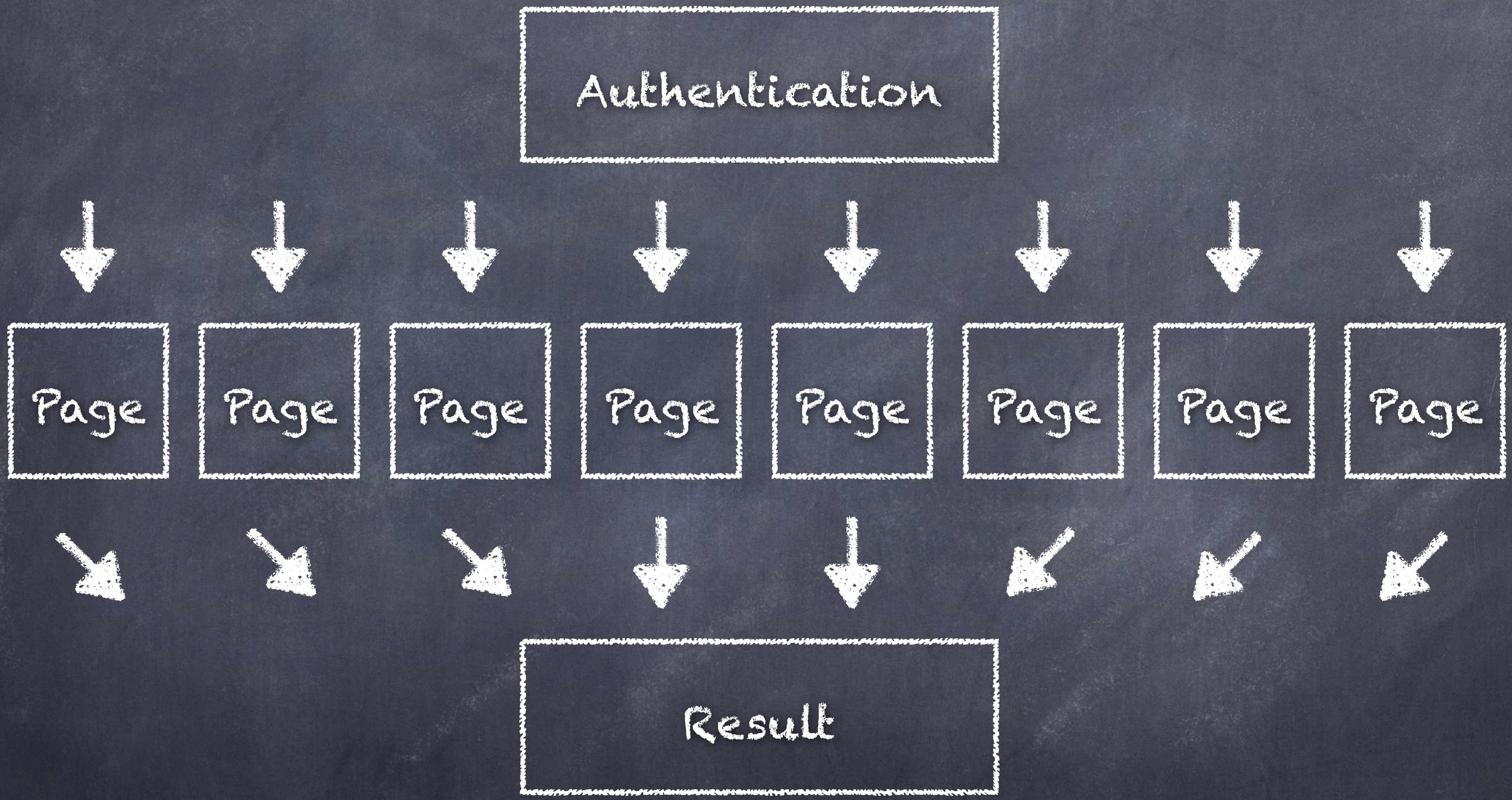
Page

Page

Page

Page

Massive Parallelism



Authentication

Page

Page

Page

Page

Page

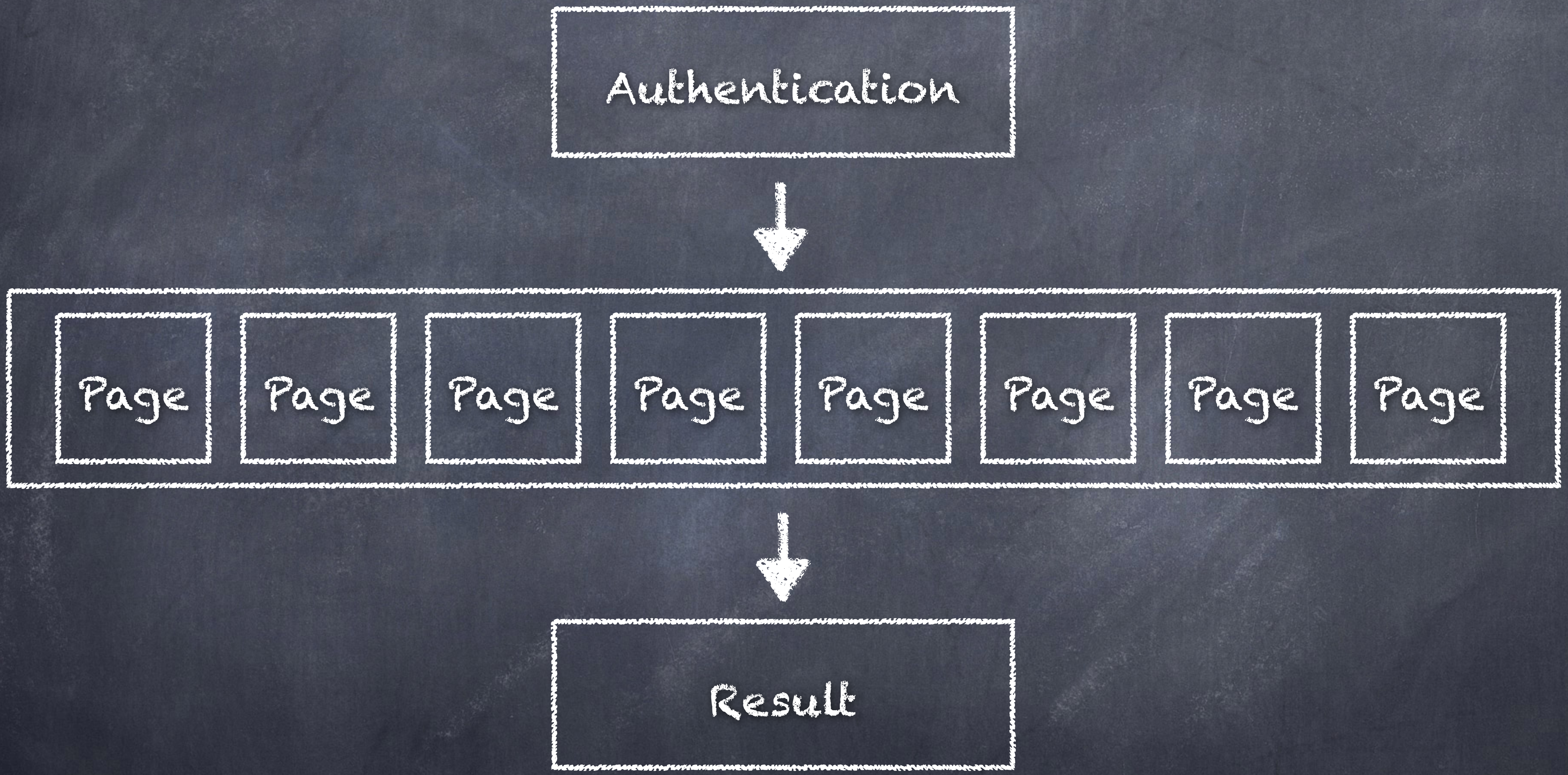
Page

Page

Page

Result

Massive Parallelism Opportunity



High Throughput through Parallelism

High Throughput through Parallelism

```
def authenticate(username: String, password: String): AsyncEffect[Token]

def scrapePage(token: Token)(url: String): AsyncEffect[PageContent]

def mergePages(pages: List[PageContent]): AsyncEffect[Result]
```

High Throughput through Parallelism

```scala
def authenticate(username: String, password: String): AsyncEffect[Token]

def scrapePage(token: Token)(url: String): AsyncEffect[PageContent]

def mergePages(pages: List[PageContent]): AsyncEffect[Result]

val urls: List[String] = ... // 1000 Urls
```

High Throughput through Parallelism

```
def authenticate(username: String, password: String): AsyncEffect[Token]

def scrapePage(token: Token)(url: String): AsyncEffect[PageContent]

def mergePages(pages: List[PageContent]): AsyncEffect[Result]
```

```
val urls: List[String] = ... // 1000 Urls
```

```
authenticate("tdc", "2019").flatMap { token =>
  AsyncEffect
    .traverse(urls)(scrapePage(token))
    .flatMap { list : List[PageContent]  =>
        mergePages(list)
      }
  }
```

High Throughput through Parallelism

```
def authenticate(username: String, password: String): AsyncEffect[Token]

def scrapePage(token: Token)(url: String): AsyncEffect[PageContent]

def mergePages(pages: List[PageContent]): AsyncEffect[Result]
```

```
val urls: List[String] = ... // 1000 Urls
```

```
authenticate("tdc", "2019").flatMap { token =>
  AsyncEffect
   .traverse(urls)(scrapePage(token))
   .flatMap { list : List[PageContent]  =>
        mergePages(list)
     }
  }
```

```
for {
   token <- authenticate("tdc", "2019")
   list <-  AsyncEffect.traverse(urls)(scrapePage(token))
   result <- mergePages(list)
} yield result
```

Expression Evaluation

Expression Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```

Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```

```
expr(10, 20, 30)
```

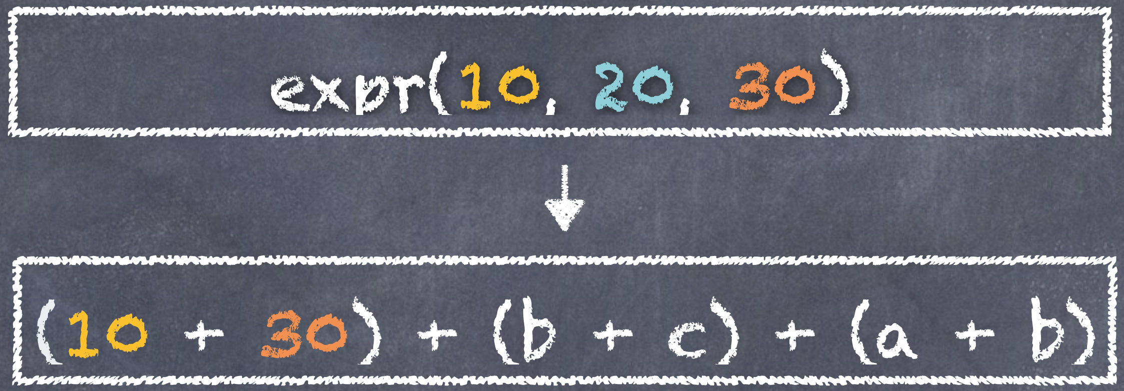
Imperative Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```

```
expr(10, 20, 30)
```

Imperative Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



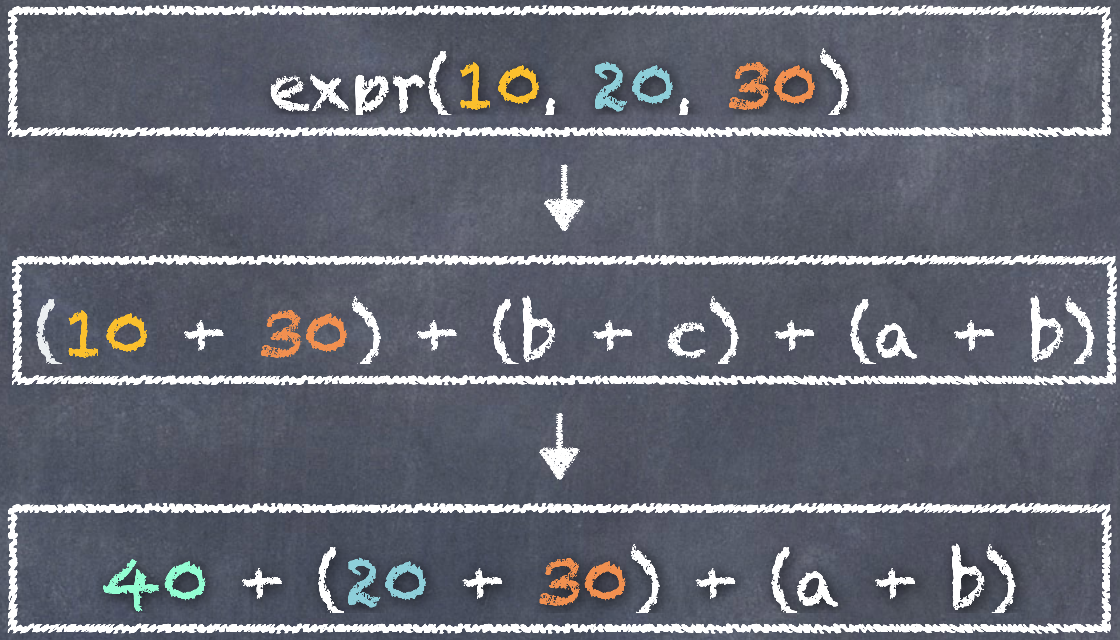
expr(10, 20, 30)

↓

(10 + 30) + (b + c) + (a + b)

Imperative Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



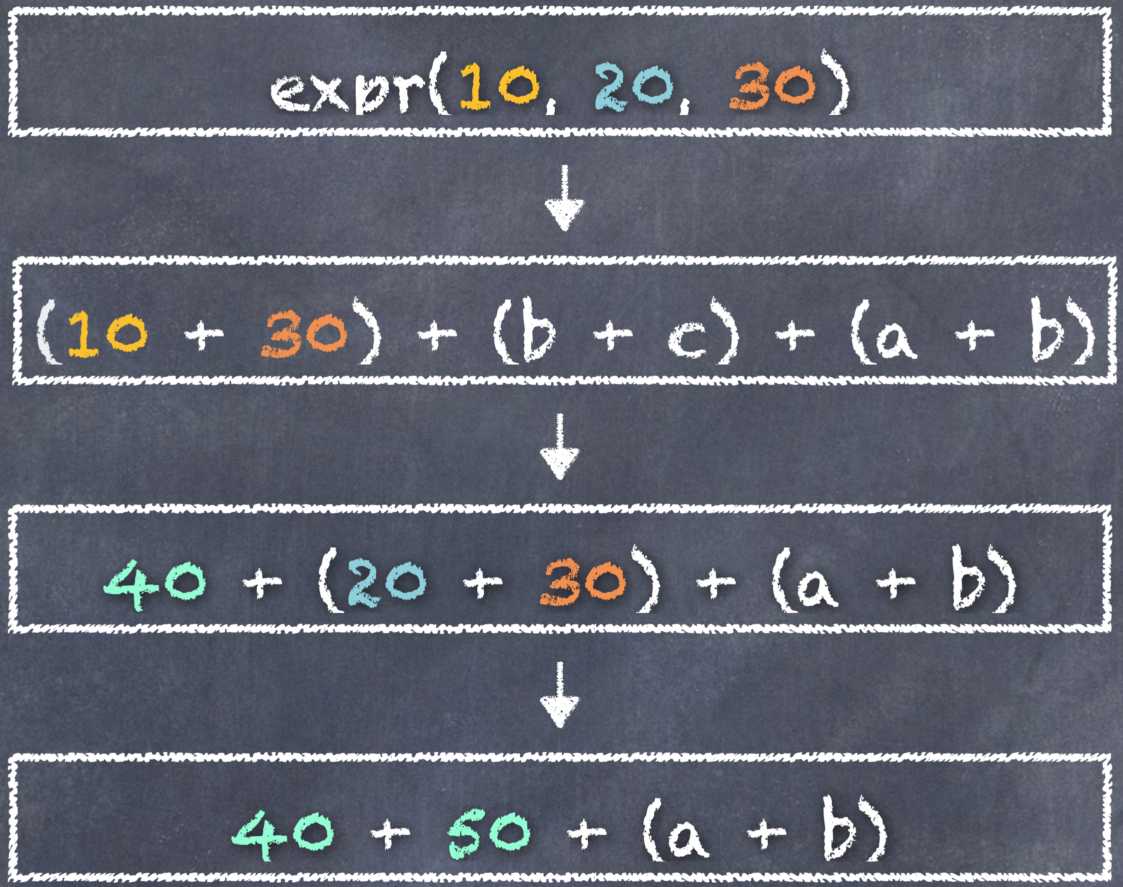
expr(10, 20, 30)

(10 + 30) + (b + c) + (a + b)

40 + (20 + 30) + (a + b)

Imperative Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



expr(10, 20, 30)

↓

(10 + 30) + (b + c) + (a + b)

↓

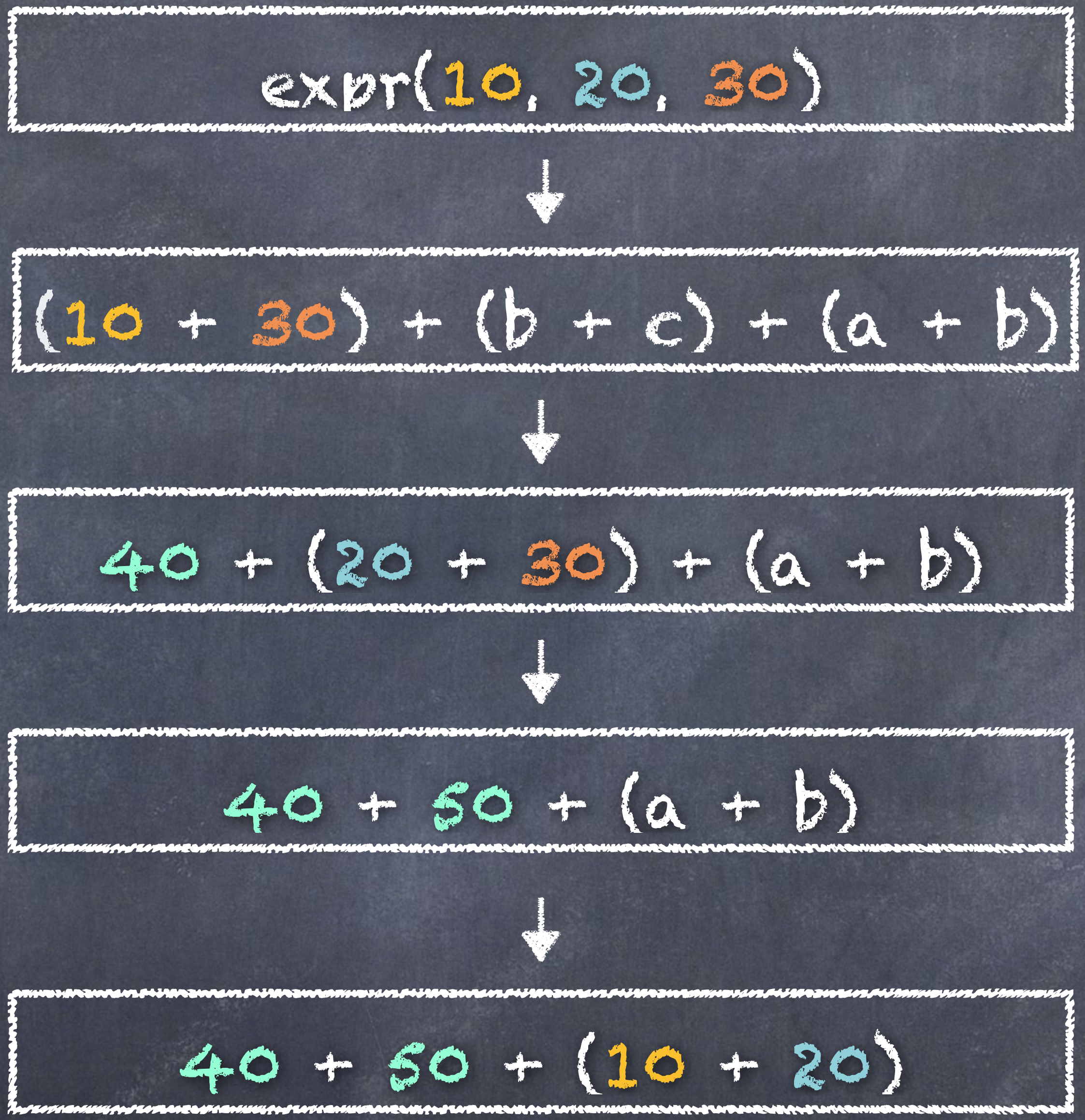
40 + (20 + 30) + (a + b)

↓

40 + 50 + (a + b)

Imperative Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



expr(10, 20, 30)

↓

(10 + 30) + (b + c) + (a + b)

↓

40 + (20 + 30) + (a + b)

↓

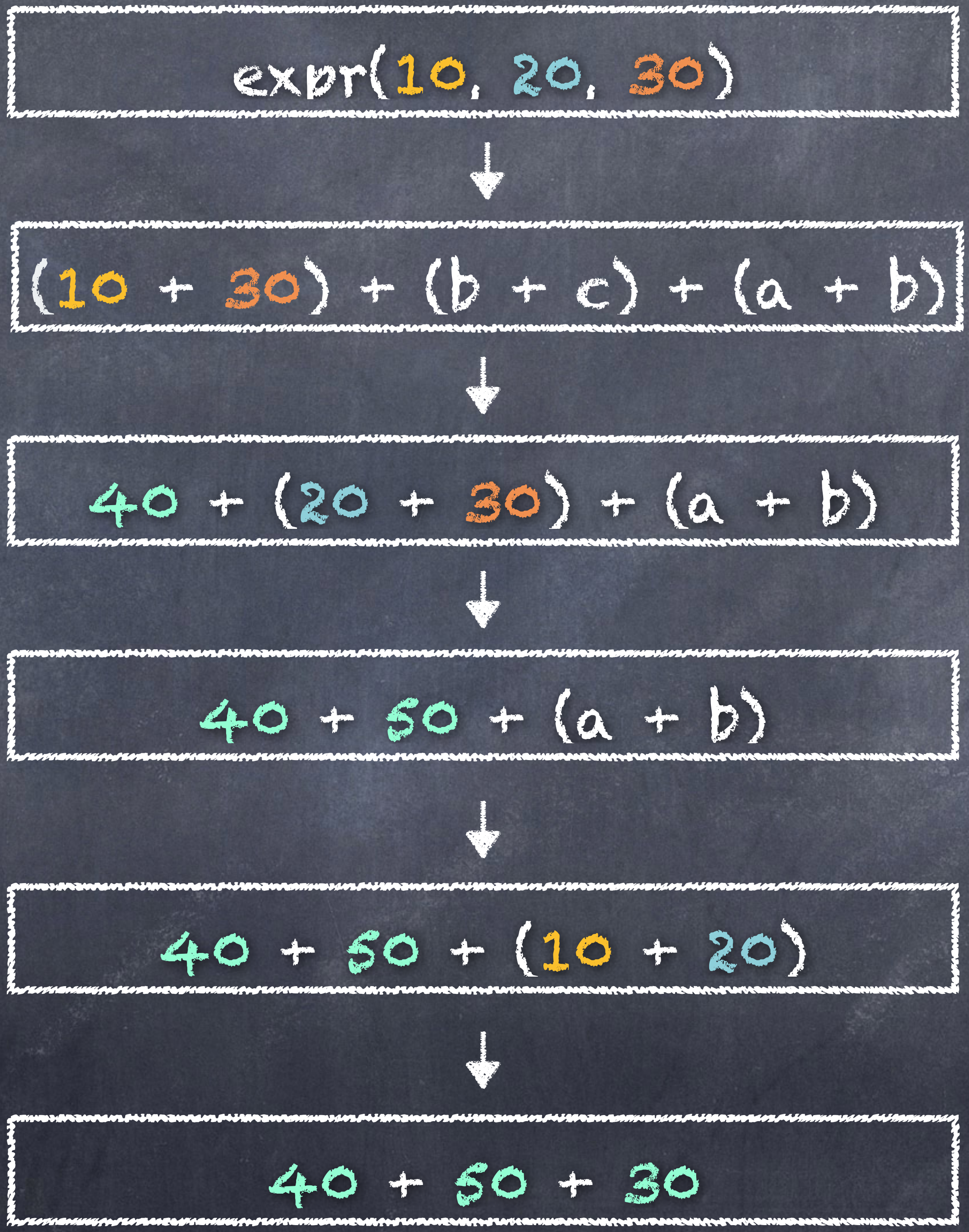
40 + 50 + (a + b)

↓

40 + 50 + (10 + 20)

Imperative Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



expr(10, 20, 30)

(10 + 30) + (b + c) + (a + b)

40 + (20 + 30) + (a + b)

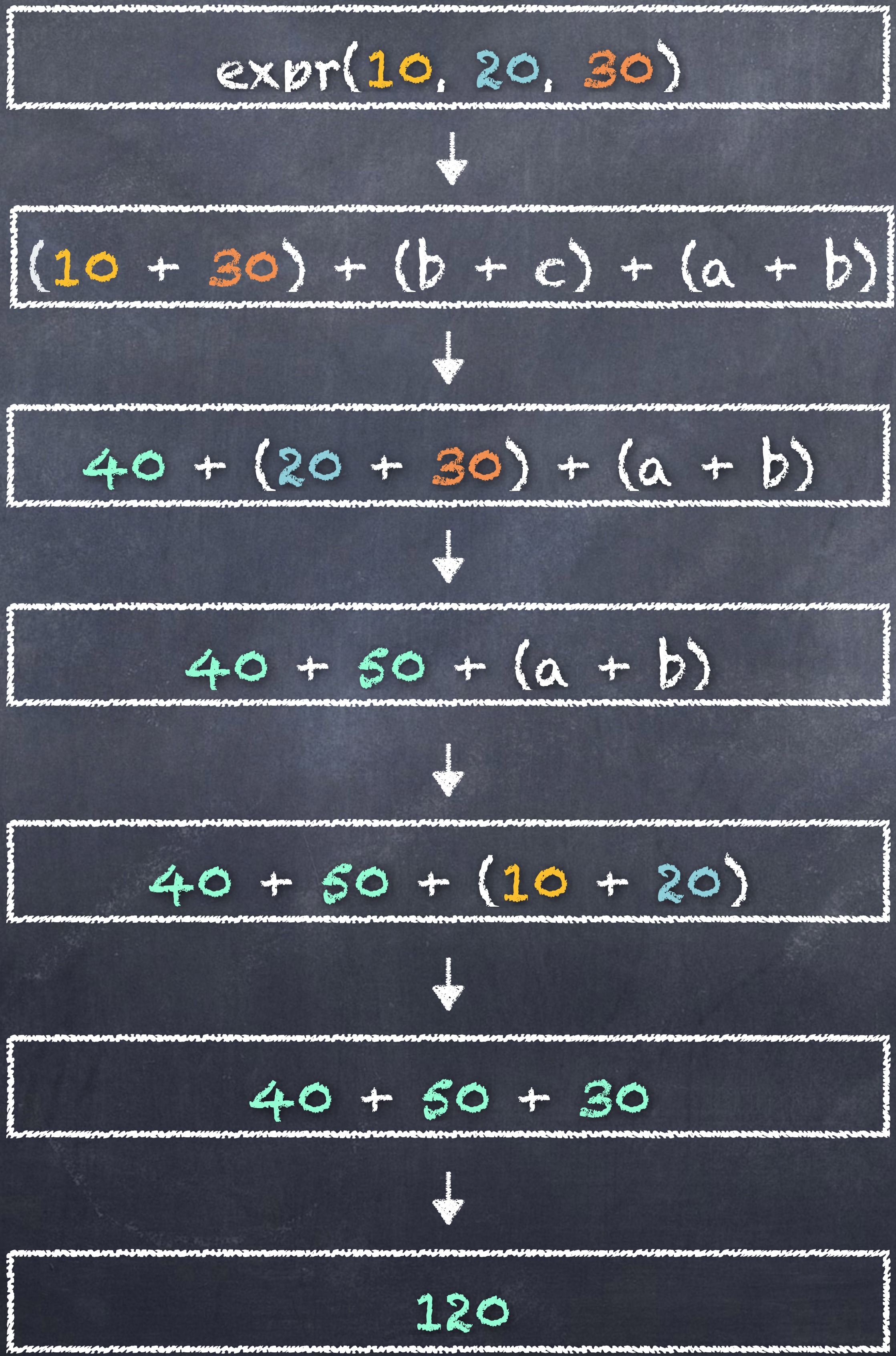
40 + 50 + (a + b)

40 + 50 + (10 + 20)

40 + 50 + 30

Imperative Evaluation

```scala
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



expr(10, 20, 30)

↓

(10 + 30) + (b + c) + (a + b)

↓

40 + (20 + 30) + (a + b)

↓

40 + 50 + (a + b)

↓

40 + 50 + (10 + 20)

↓

40 + 50 + 30

↓

120

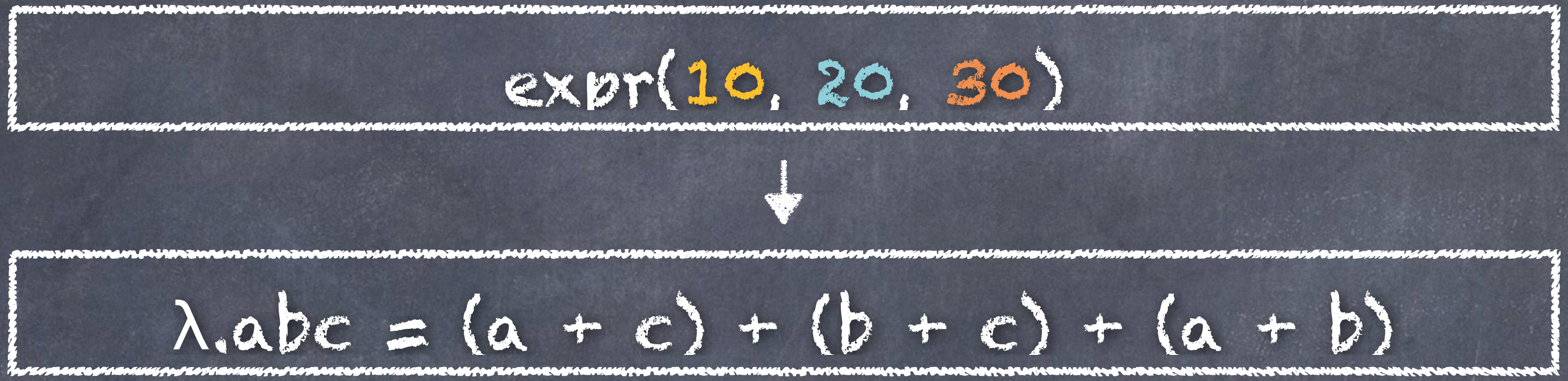
λ-Calculus Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```

```
expr(10, 20, 30)
```

λ-Calculus Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



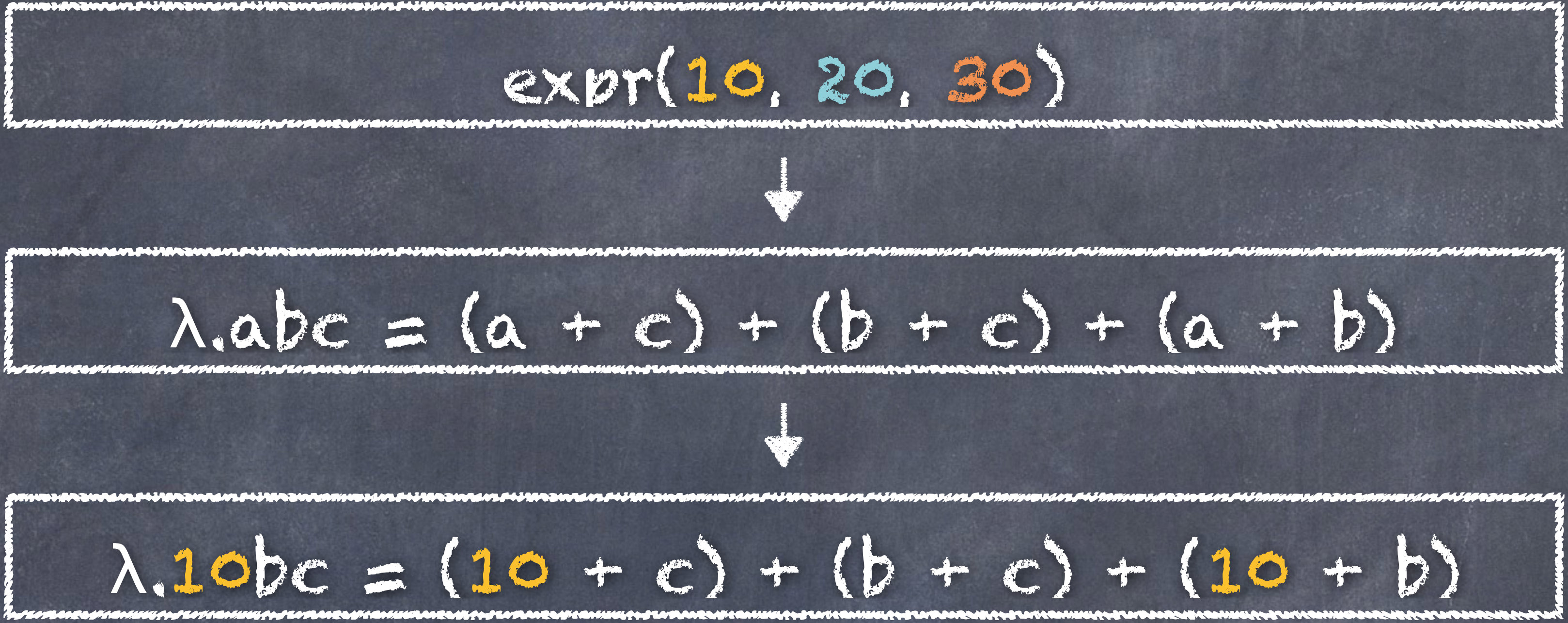
expr(10, 20, 30)

↓

λ.abc = (a + c) + (b + c) + (a + b)

λ-Calculus Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



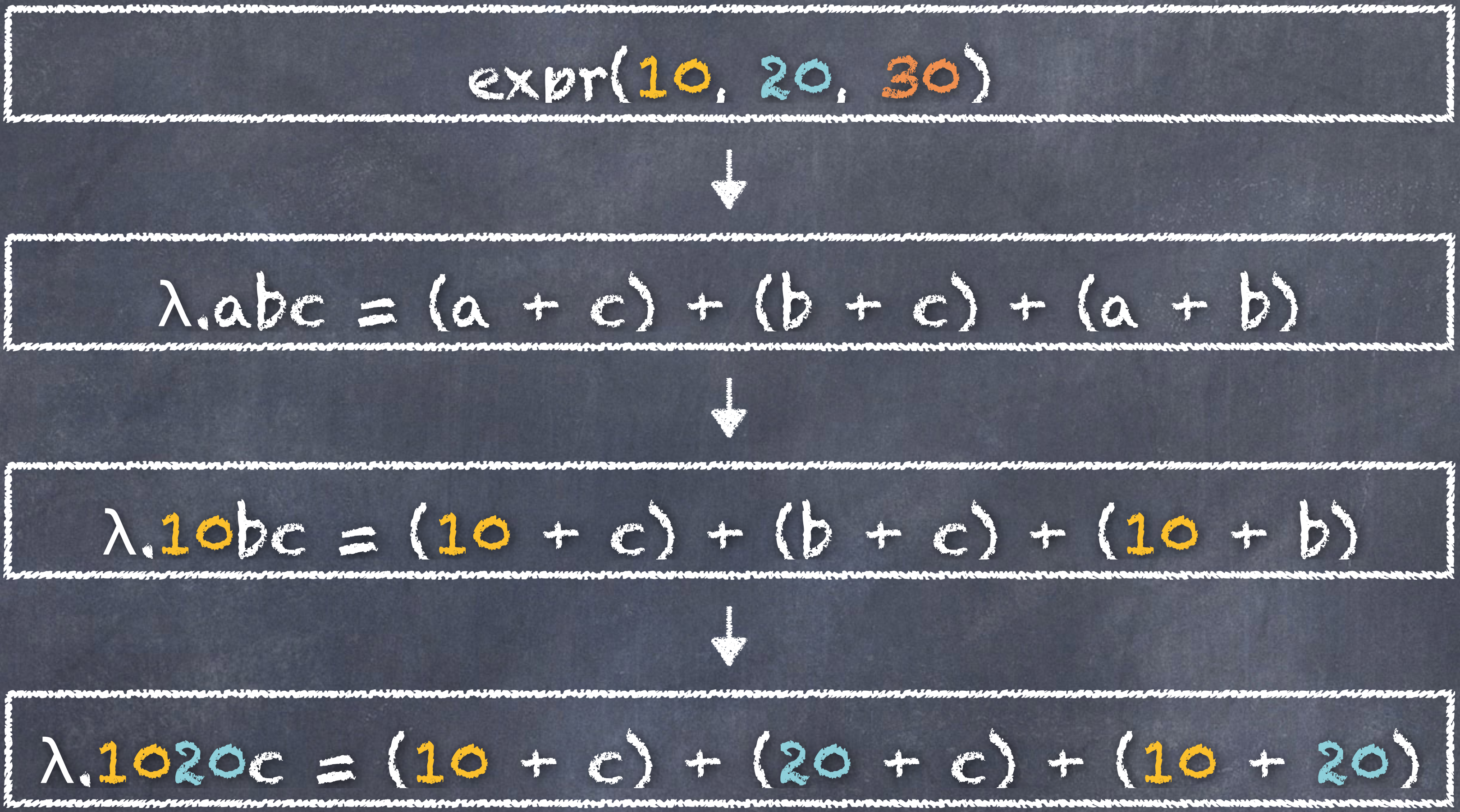
expr(10, 20, 30)

λ.abc = (a + c) + (b + c) + (a + b)

λ.10bc = (10 + c) + (b + c) + (10 + b)

λ-Calculus Evaluation

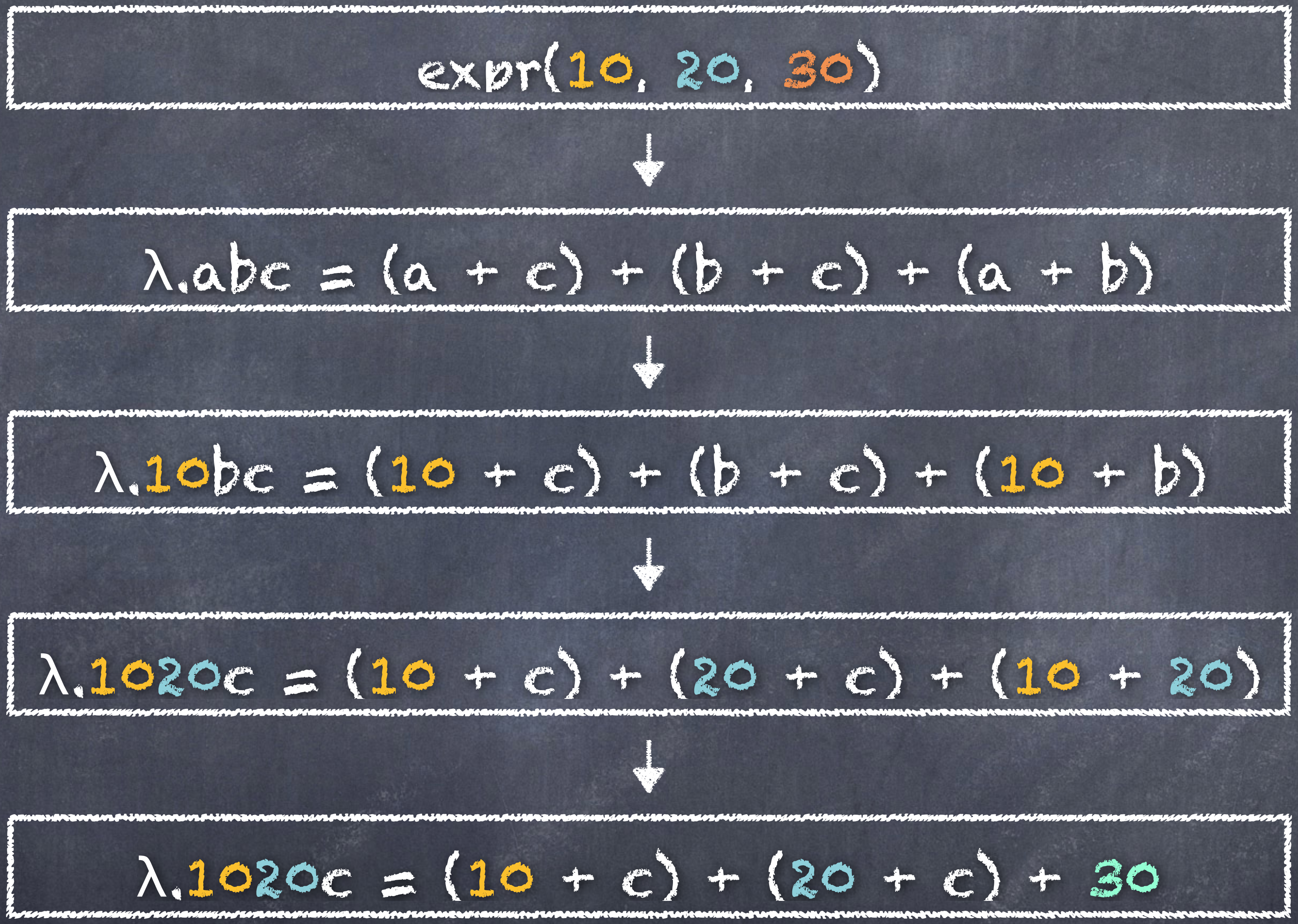
```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



```
expr(10, 20, 30)
↓
λ.abc = (a + c) + (b + c) + (a + b)
↓
λ.10bc = (10 + c) + (b + c) + (10 + b)
↓
λ.1020c = (10 + c) + (20 + c) + (10 + 20)
```

λ-Calculus Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



expr(10, 20, 30)

↓

λ.abc = (a + c) + (b + c) + (a + b)

↓

λ.10bc = (10 + c) + (b + c) + (10 + b)

↓

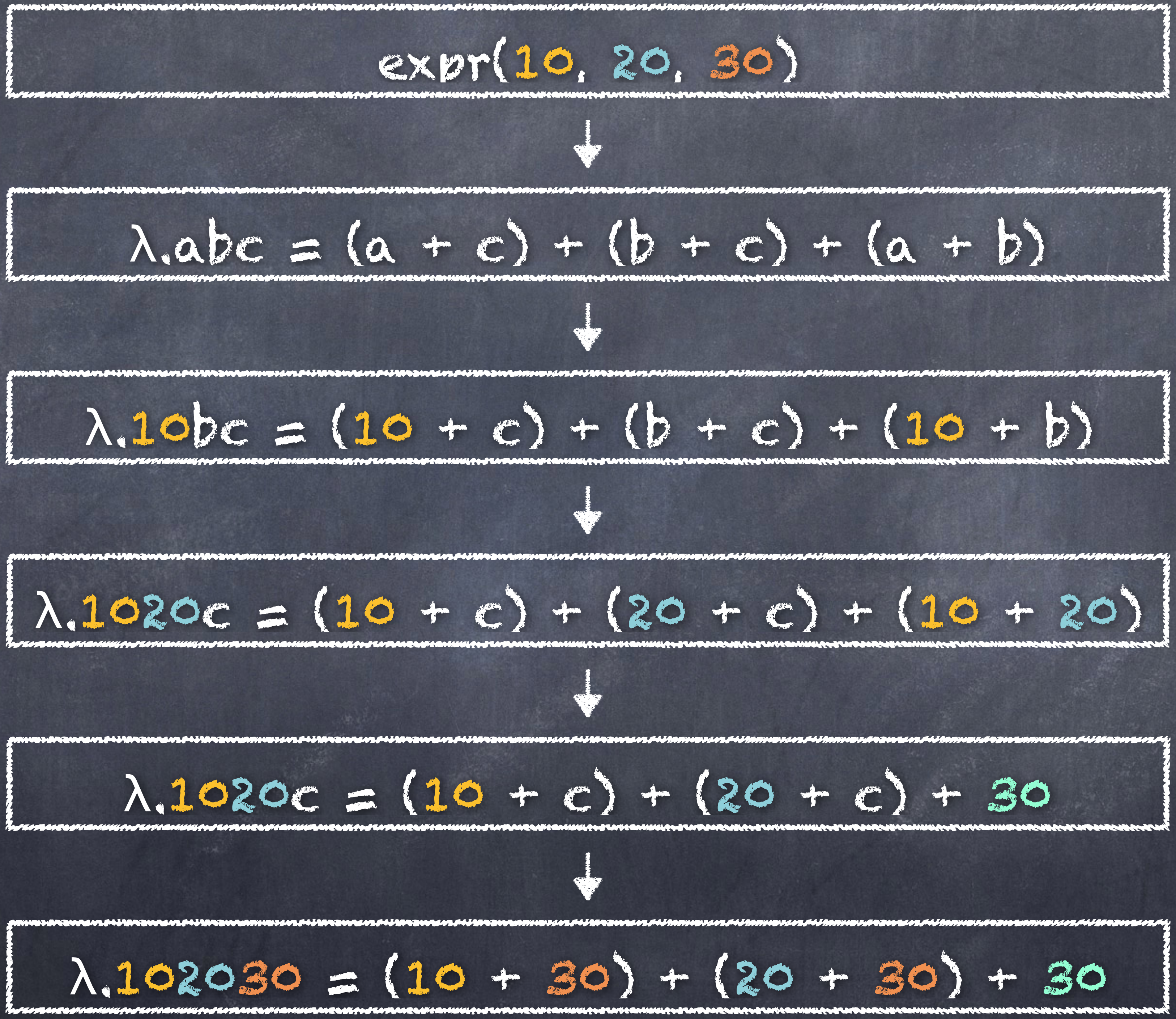
λ.1020c = (10 + c) + (20 + c) + (10 + 20)

↓

λ.1020c = (10 + c) + (20 + c) + 30

λ-Calculus Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



expr(10, 20, 30)

↓

λ.abc = (a + c) + (b + c) + (a + b)

↓

λ.10bc = (10 + c) + (b + c) + (10 + b)

↓

λ.1020c = (10 + c) + (20 + c) + (10 + 20)

↓

λ.1020c = (10 + c) + (20 + c) + 30

↓

λ.102030 = (10 + 30) + (20 + 30) + 30

λ-Calculus Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```

↓

λ.102030 = (10 + 30) + (20 + 30) + 30

λ-Calculus Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```

↓

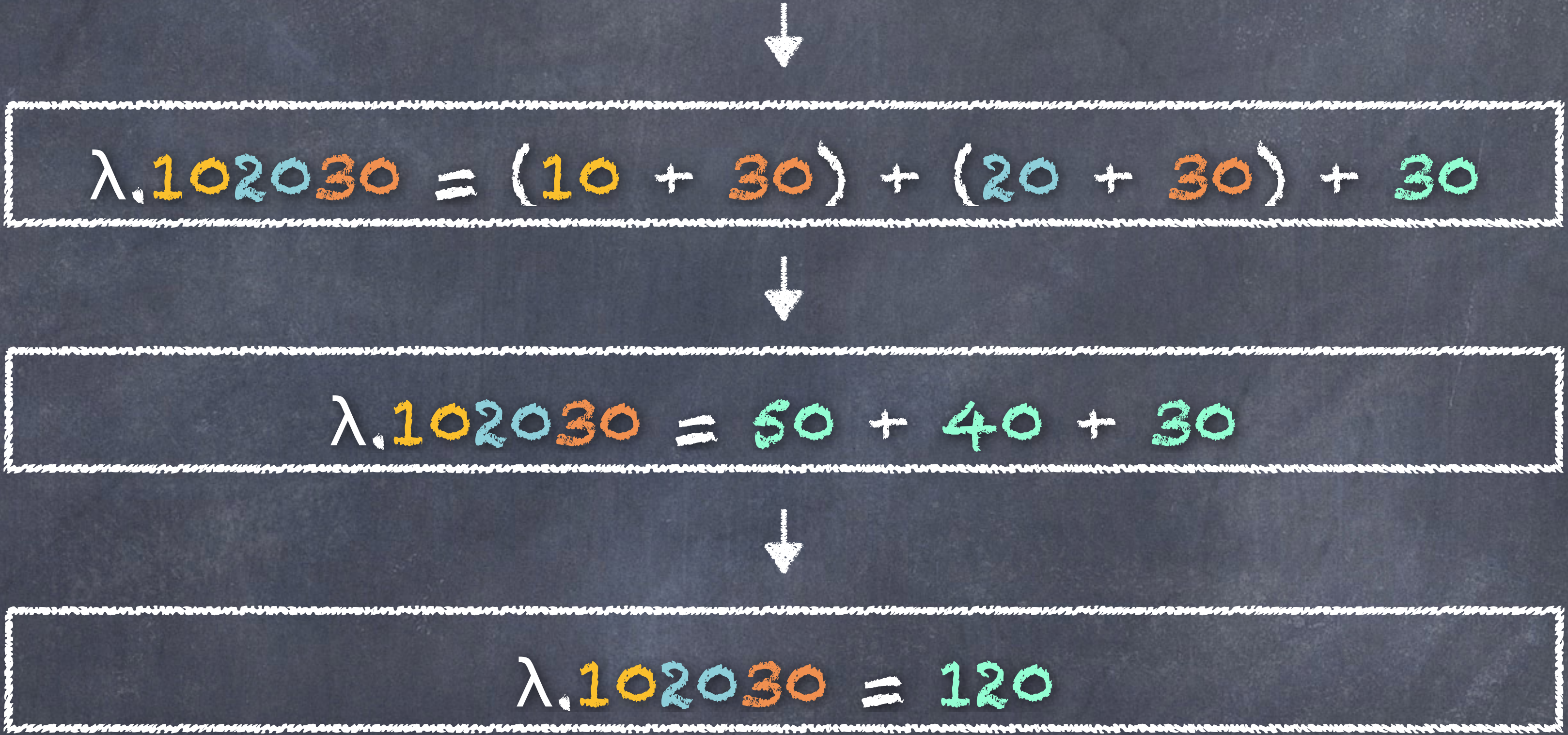
λ.102030 = (10 + 30) + (20 + 30) + 30

↓

λ.102030 = 50 + 40 + 30

λ-Calculus Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



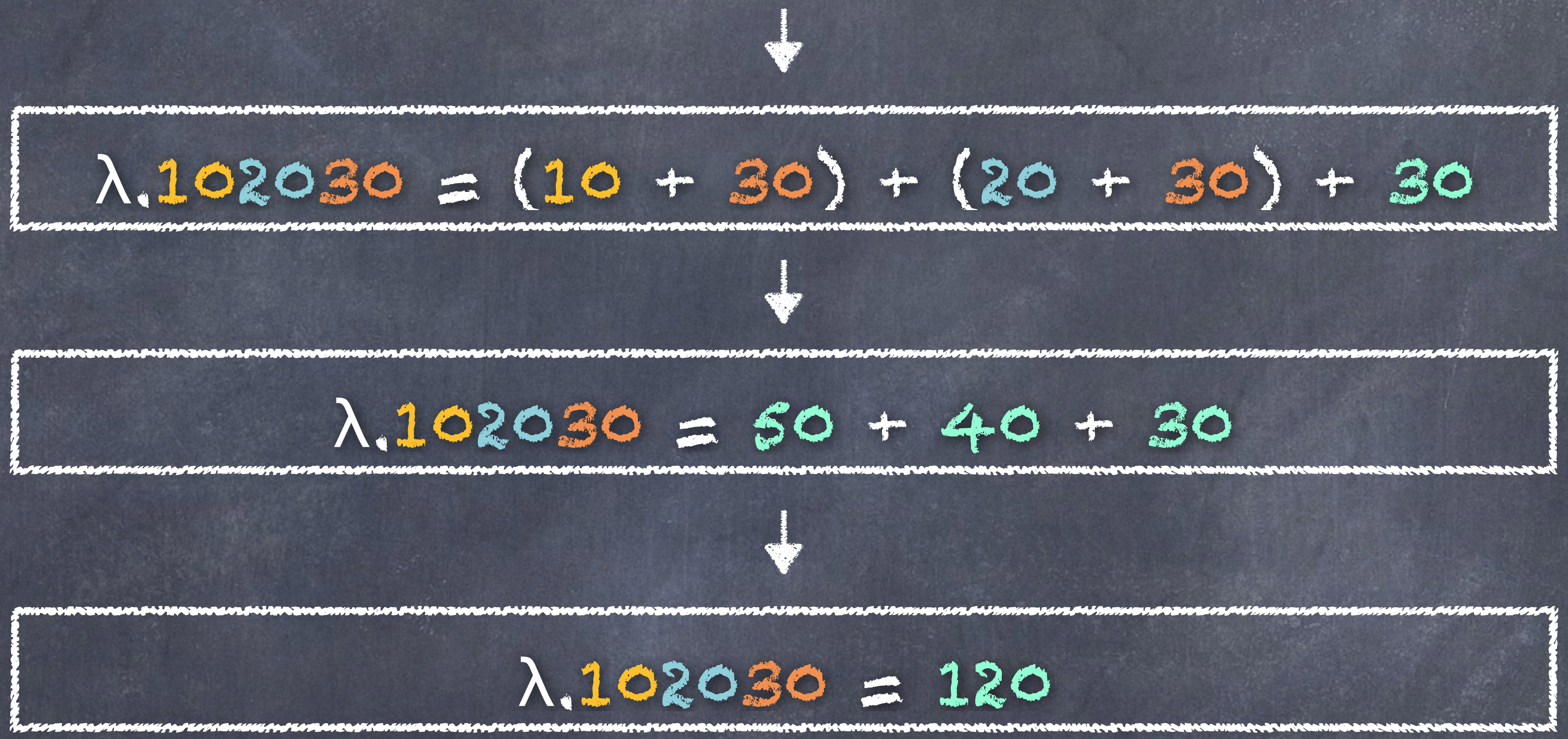
λ.102030 = (10 + 30) + (20 + 30) + 30

λ.102030 = 50 + 40 + 30

λ.102030 = 120

λ-Calculus Evaluation

```
def expr(a: Int, b: Int, c: Int): Int = (a + c) + (b + c) + (a + b)
```



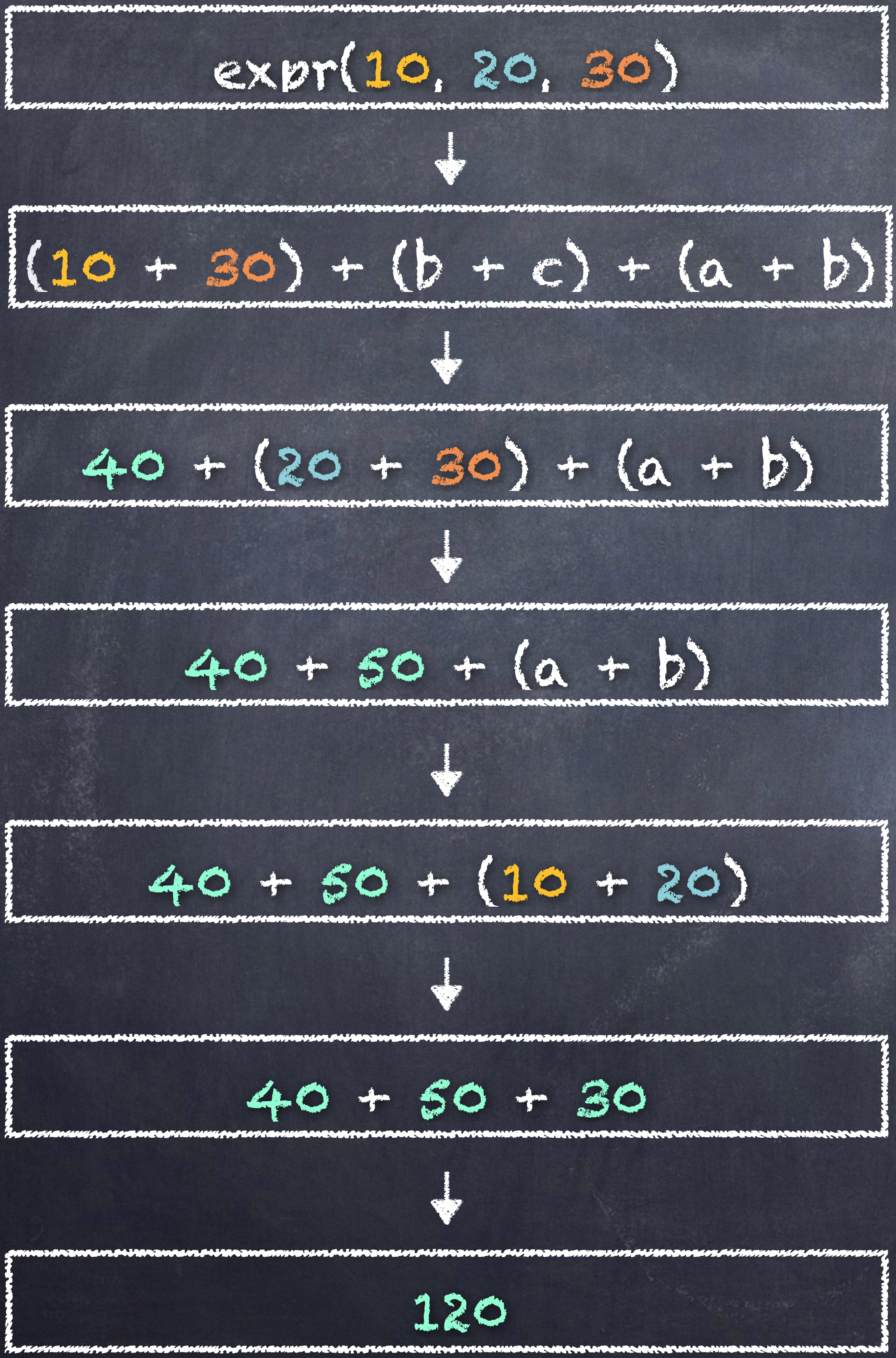
λ.102030 = (10 + 30) + (20 + 30) + 30

λ.102030 = 50 + 40 + 30

λ.102030 = 120

Imperative

Imperative



expr(10, 20, 30)

(10 + 30) + (b + c) + (a + b)

40 + (20 + 30) + (a + b)

40 + 50 + (a + b)

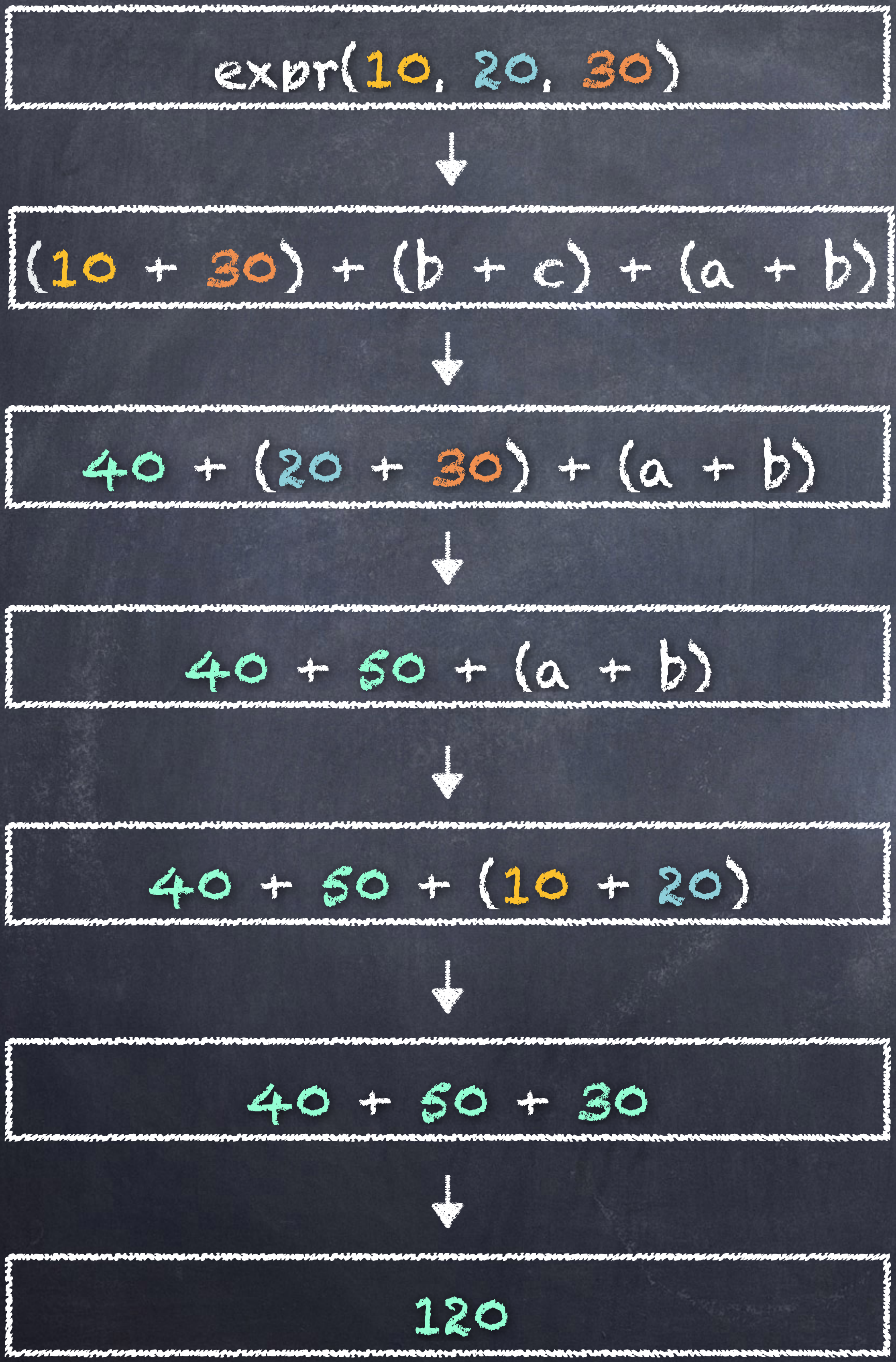
40 + 50 + (10 + 20)

40 + 50 + 30

120

Imperative

λ-Calculus



expr(10, 20, 30)

↓

(10 + 30) + (b + c) + (a + b)

↓

40 + (20 + 30) + (a + b)

↓

40 + 50 + (a + b)

↓

40 + 50 + (10 + 20)

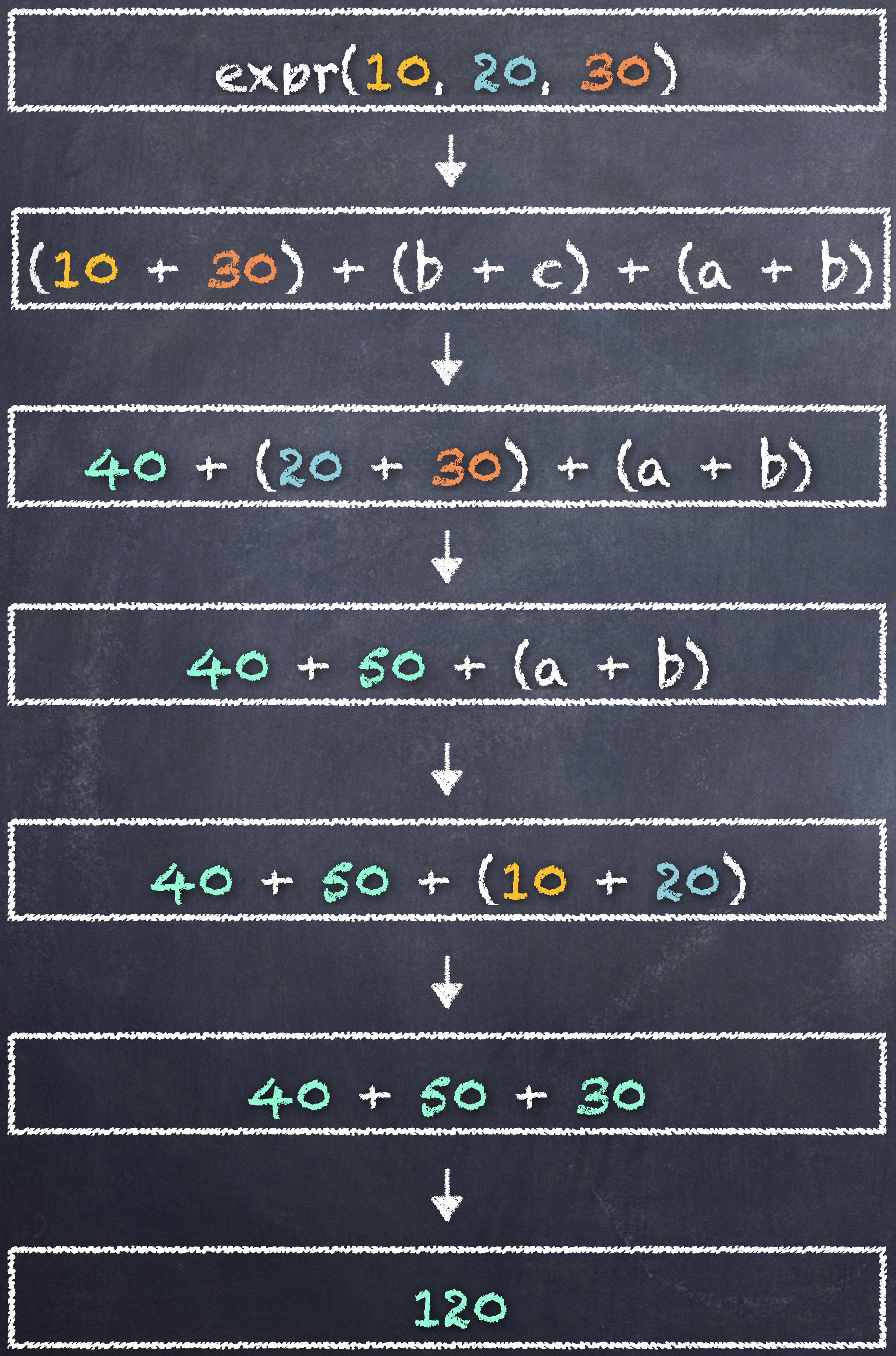
↓

40 + 50 + 30

↓

120

Imperative



expr(10, 20, 30)

(10 + 30) + (b + c) + (a + b)

40 + (20 + 30) + (a + b)

40 + 50 + (a + b)

40 + 50 + (10 + 20)

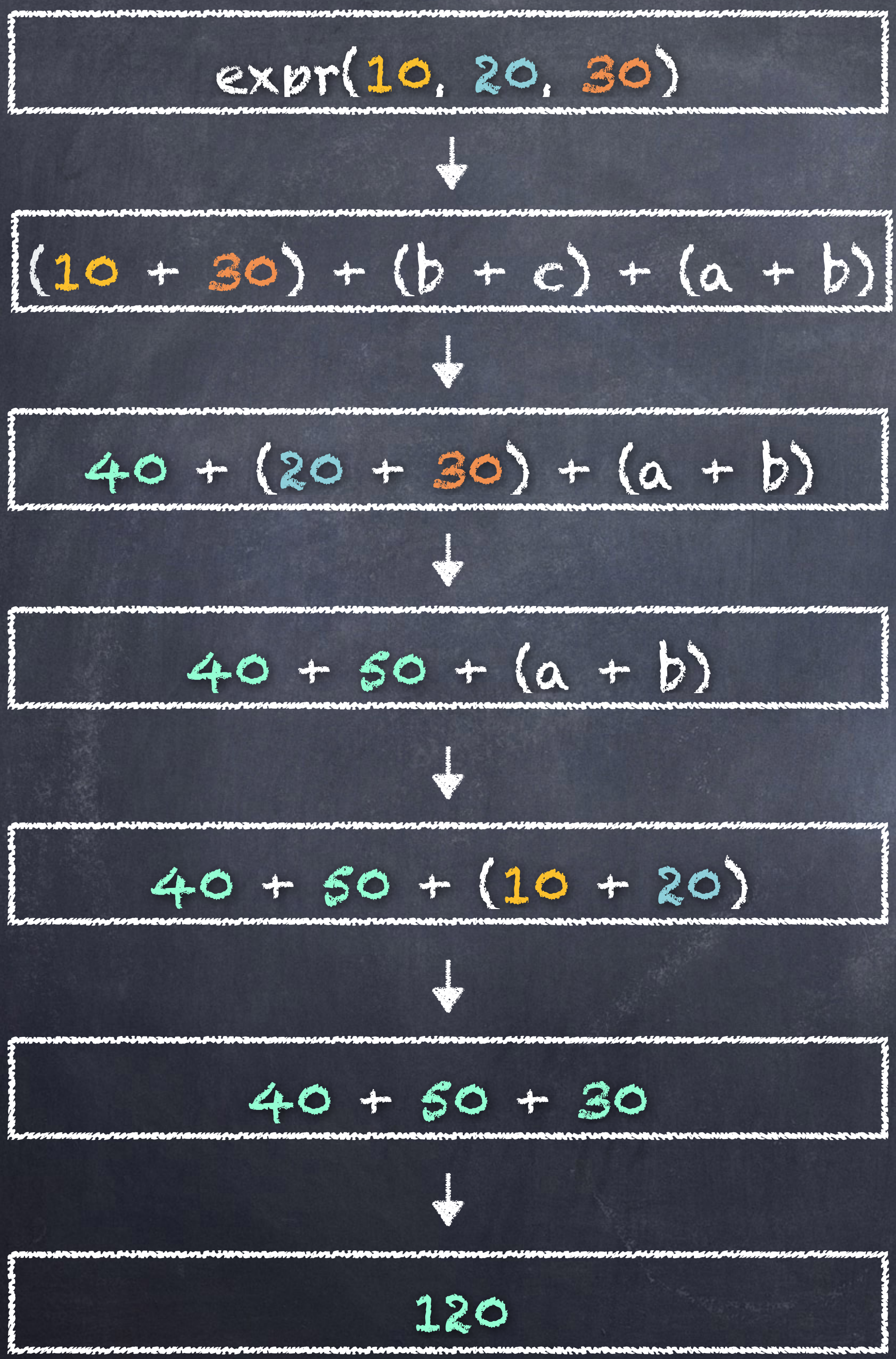
40 + 50 + 30

120

λ-Calculus

expr(10, 20, 30)

Imperative



expr(10, 20, 30)

(10 + 30) + (b + c) + (a + b)

40 + (20 + 30) + (a + b)

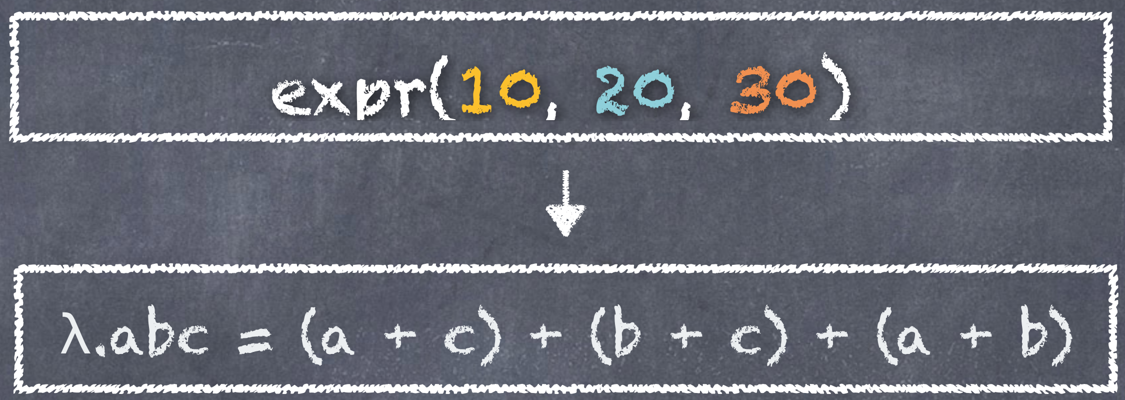
40 + 50 + (a + b)

40 + 50 + (10 + 20)

40 + 50 + 30

120

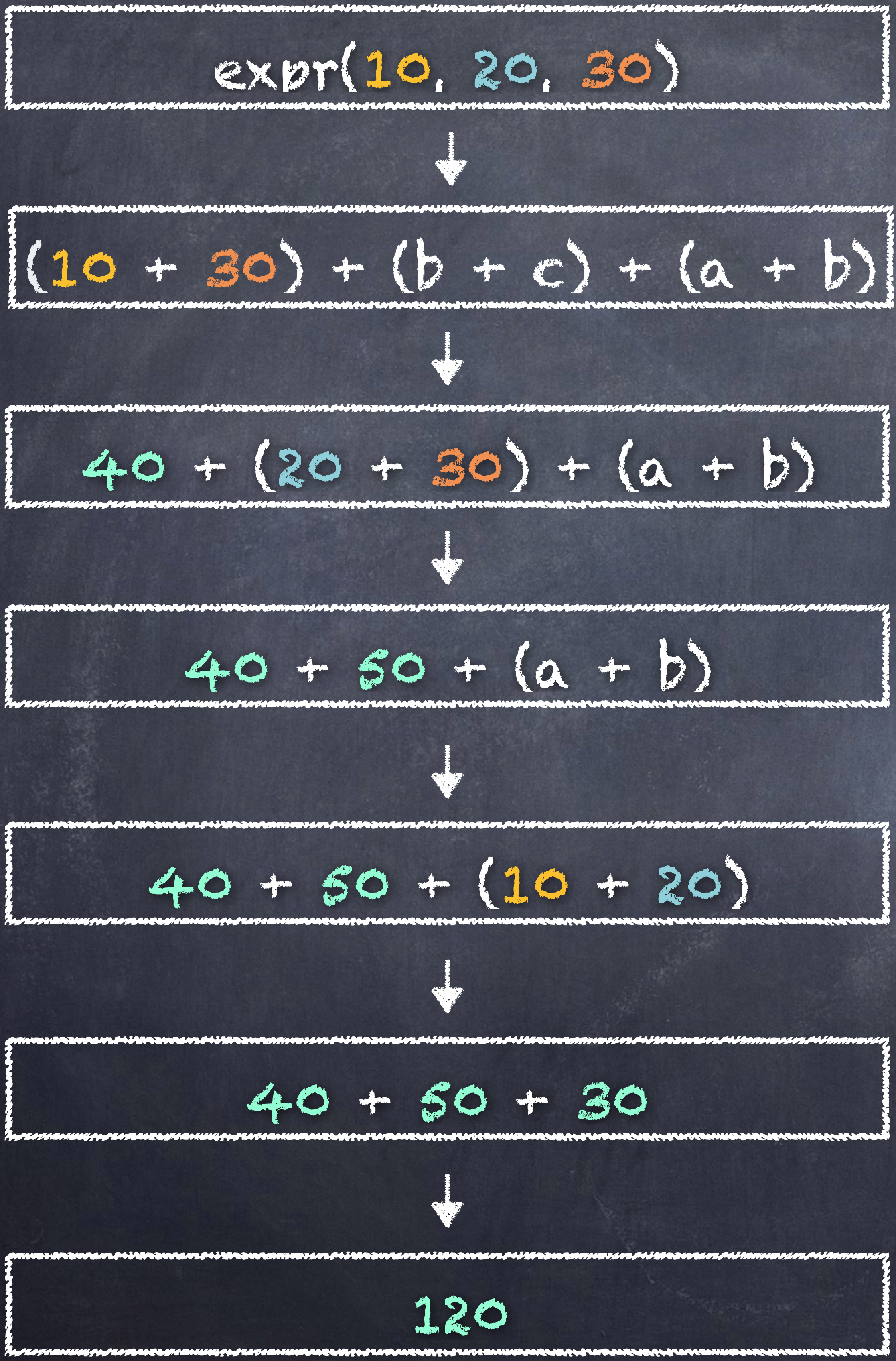
λ-Calculus



expr(10, 20, 30)

λ.abc = (a + c) + (b + c) + (a + b)

Imperative



expr(10, 20, 30)

↓

(10 + 30) + (b + c) + (a + b)

↓

40 + (20 + 30) + (a + b)

↓

40 + 50 + (a + b)

↓

40 + 50 + (10 + 20)

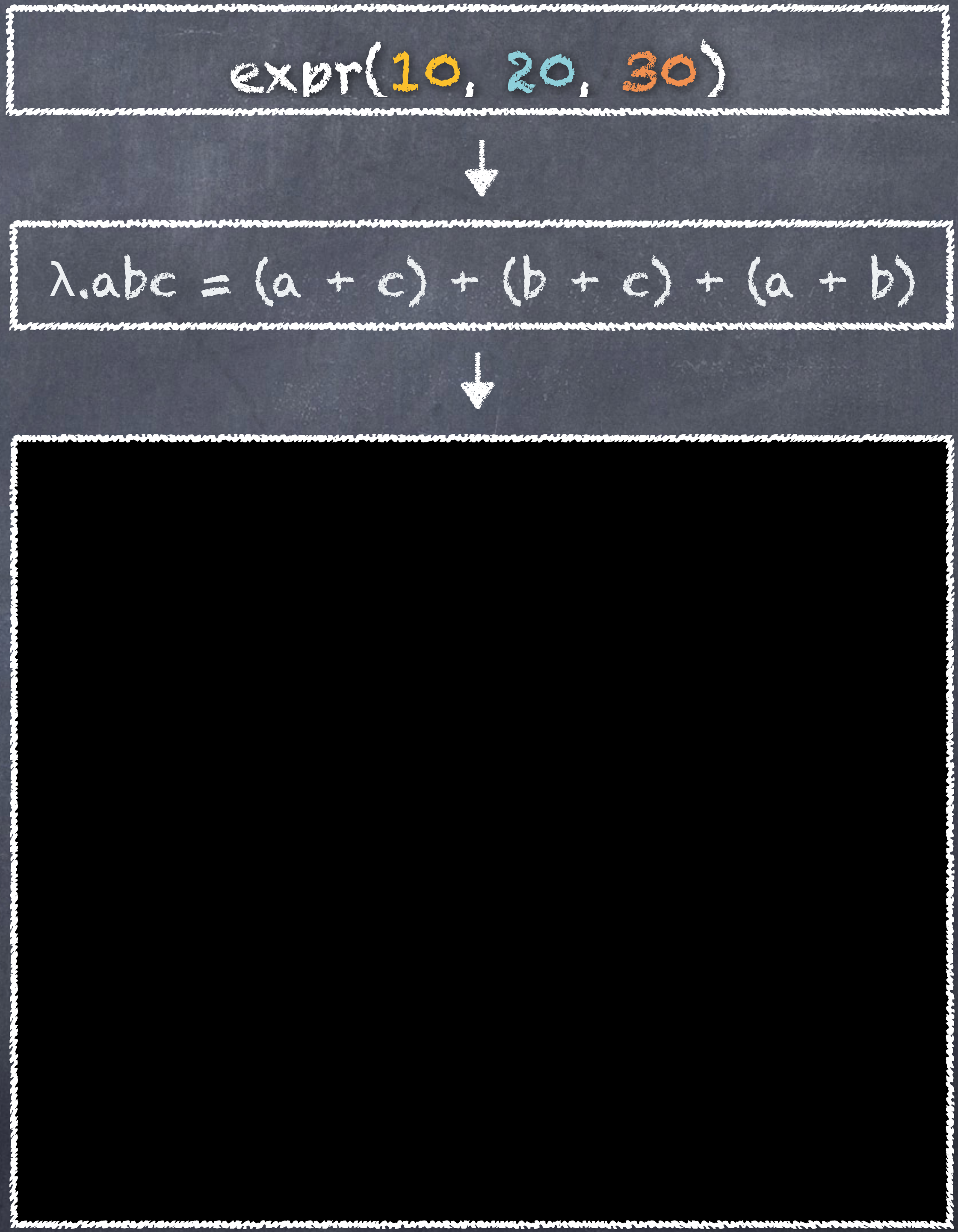
↓

40 + 50 + 30

↓

120

λ-Calculus



expr(10, 20, 30)

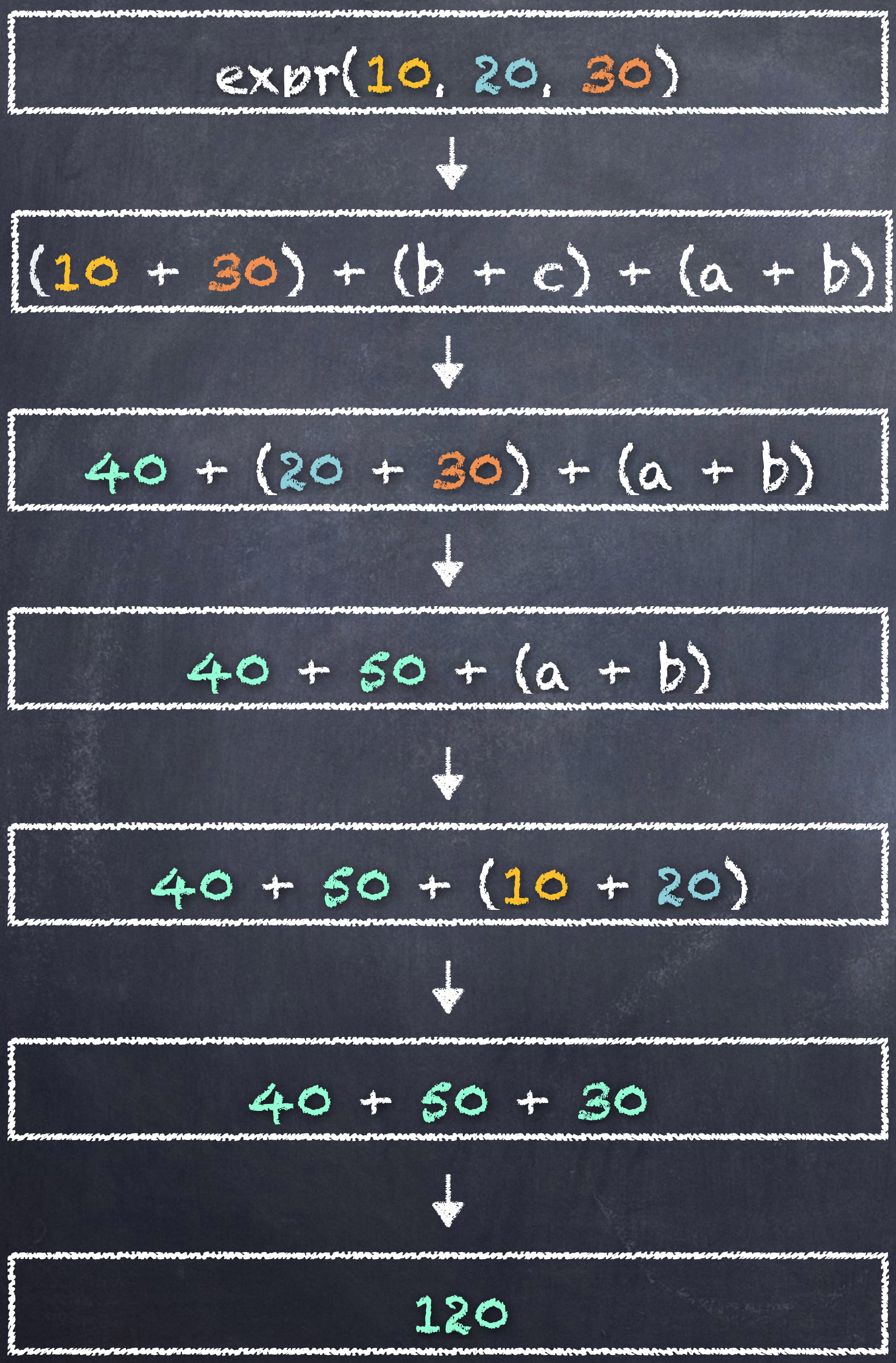
↓

λ.abc = (a + c) + (b + c) + (a + b)

↓

[redacted]

Imperative



expr(10, 20, 30)

↓

(10 + 30) + (b + c) + (a + b)

↓

40 + (20 + 30) + (a + b)

↓

40 + 50 + (a + b)

↓

40 + 50 + (10 + 20)

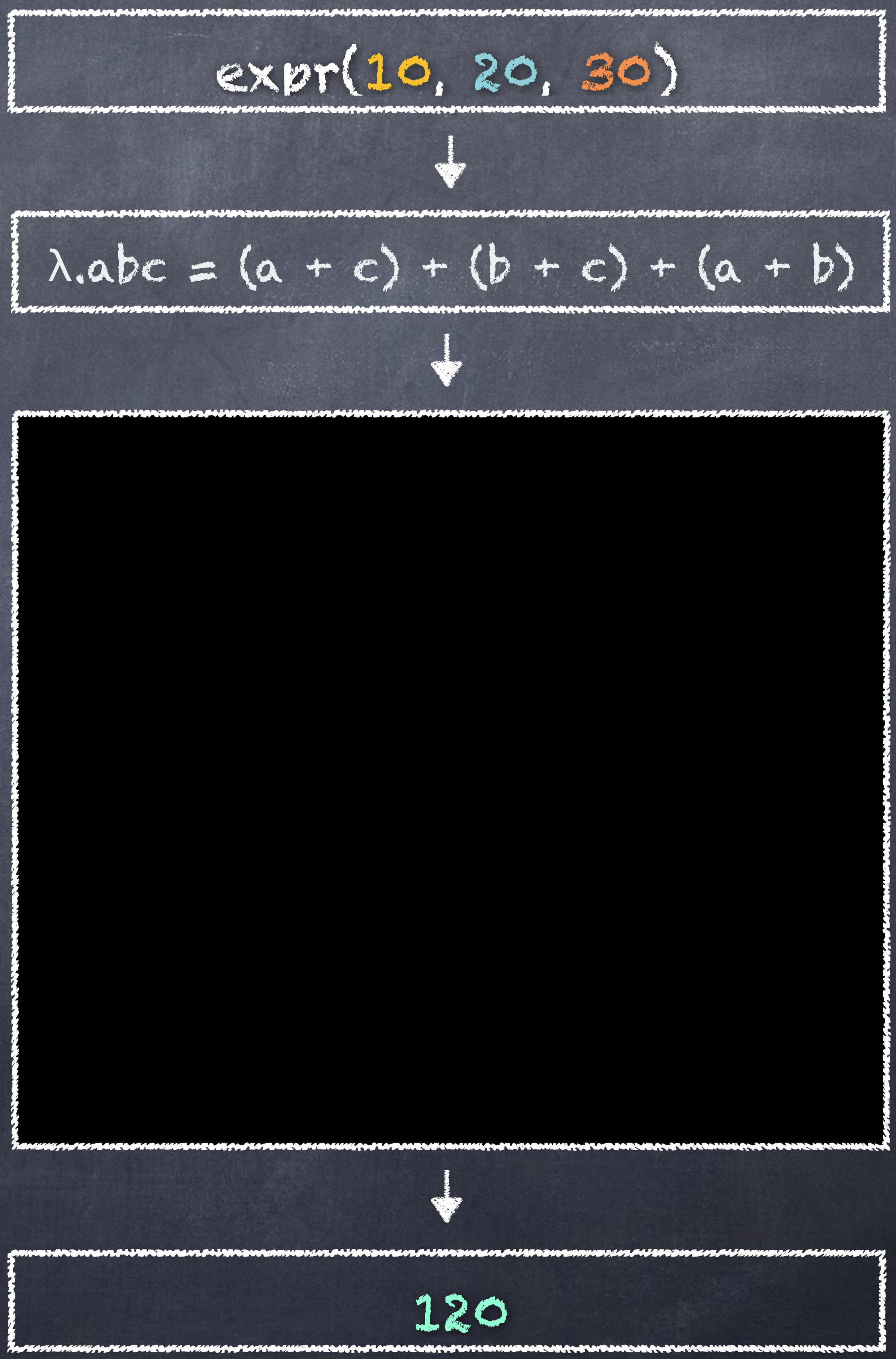
↓

40 + 50 + 30

↓

120

λ-Calculus



expr(10, 20, 30)

↓

λ.abc = (a + c) + (b + c) + (a + b)

↓

[redacted]

↓

120

In functional programming, EVERYTHING is an EXPRESSION.

Monoids

Monoids

```
trait Monoid[A] {
  def mappend(x: A, y: A): A
  def mempty: A
}
```

Monoids

```
trait Monoid[A] {
  def mappend(x: A, y: A): A
  def mempty: A
}
```

$$1 + (2 + 3) = (1 + 2) + 3$$

$$1 + 0 = 1 + 0$$

Monoids

```
trait Monoid[A] {
  def mappend(x: A, y: A): A
  def mempty: A
}
```

$$1 + (2 + 3) = (1 + 2) + 3$$

$$1 + 0 = 1 + 0$$

$$3 * (2 * 3) = (3 * 2) * 3$$

$$3 * 1 = 3 * 1$$

Monoids

```
trait Monoid[Expr] {
  def <>(x: Expr, y: Expr): Expr
  def mempty: id
}
```

Monoids

```
trait Monoid[Expr] {
  def <>(x: Expr, y: Expr): Expr
  def mempty: id
}

  Expr <> Expr <> Expr <> Expr <> Expr
```

Monoids

```
trait Monoid[Expr] {
  def <>(x: Expr, y: Expr): Expr
  def mempty: id
}
```

```
  Expr <> Expr <> Expr <> Expr <> Expr

      Expr <> R <> Expr <> R <> Expr
```

Monoids

```
trait Monoid[Expr] {
  def <>(x: Expr, y: Expr): Expr
  def mempty: id
}
```

```
Expr <> Expr <> Expr <> Expr <> Expr

Expr <> R <> Expr <> R <> Expr

R <> R <> Expr <> R <> R
```

Monoids

```
trait Monoid[Expr] {
  def <>(x: Expr, y: Expr): Expr
  def mempty: id
}
```

Expr <> Expr <> Expr <> Expr <> Expr

Expr <> R <> Expr <> R <> Expr

R <> R <> Expr <> R <> R

RR <> R <> RR

Monoids

```
trait Monoid[Expr] {
  def <>(x: Expr, y: Expr): Expr
  def mempty: id
}
```

Expr <> Expr <> Expr <> Expr <> Expr

Expr <> R <> Expr <> R <> Expr

R <> R <> Expr <> R <> R

RR <> R <> RR

RRR <> RR

Monoids

```
trait Monoid[Expr] {
  def <>(x: Expr, y: Expr): Expr
  def mempty: id
}
```

Expr <> Expr <> Expr <> Expr <> Expr

Expr <> R <> Expr <> R <> Expr

R <> R <> Expr <> R <> R

RR <> R <> RR

RRR <> RR

RRRRR

Monoids

Monoids

Expr <> Expr <> Error <> Expr <> Expr = Error

Monoids

Expr <> Expr <> Error <> Expr <> Expr = Error

Error <> R <> Expr <> R <> Expr = Error

Monoids

Expr <> Expr <> Error <> Expr <> Expr = Error

Error <> R <> Expr <> R <> Expr = Error

R <> R <> R <> Error <> R = Error

Functors

Functors

```
trait Functor[F[_]] {
  def map[A, B](fa: F[A])(f: A => B): F[B]
}
```

Functors

```
trait Functor[F[_]] {
  def map[A, B](fa: F[A])(f: A => B): F[B]
}
```

```
def toB(a: A): B = ...
```

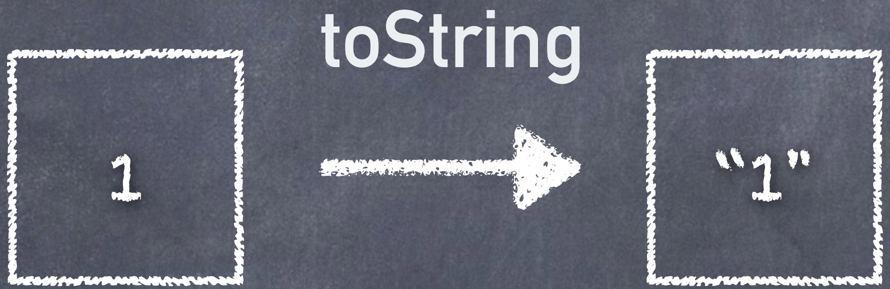
Functors

```
trait Functor[F[_]] {
  def map[A, B](fa: F[A])(f: A => B): F[B]
}
```

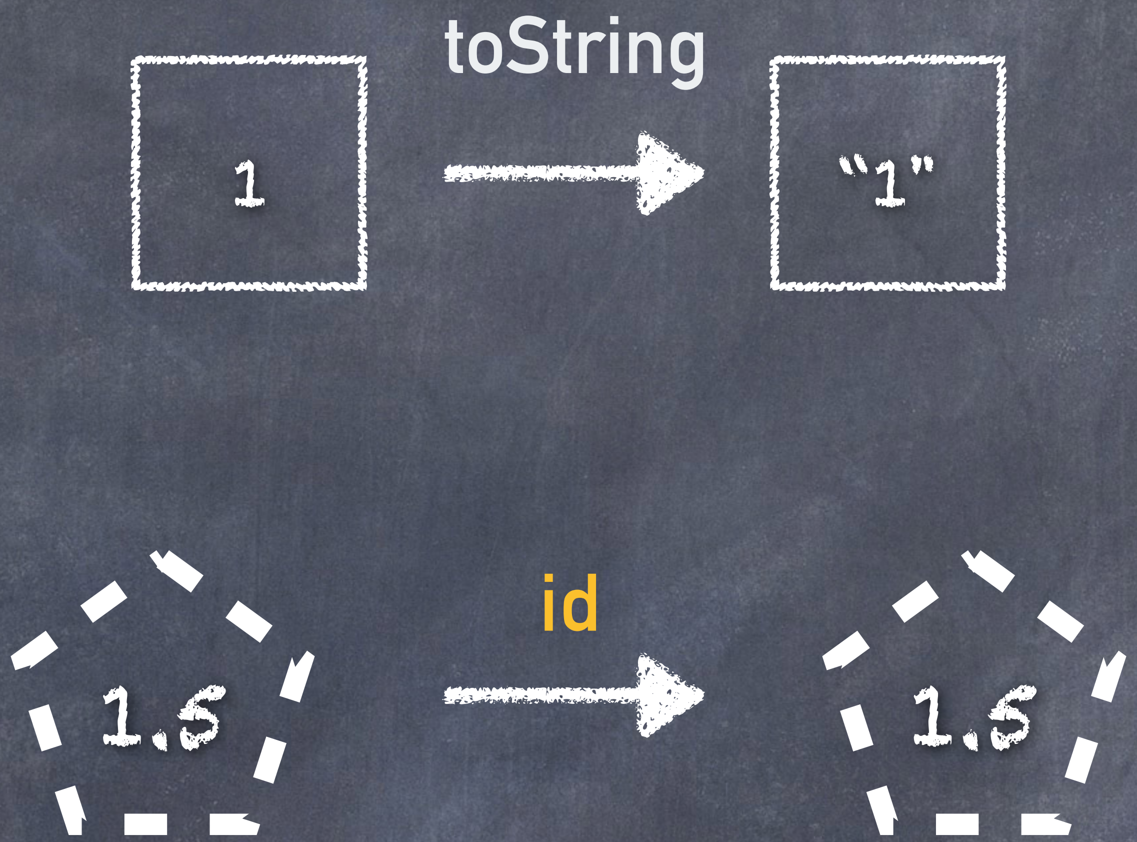
```
def toB(a: A): B = ...

val fb: F[B] = F[A].map(toB)
```

Functors

Functors



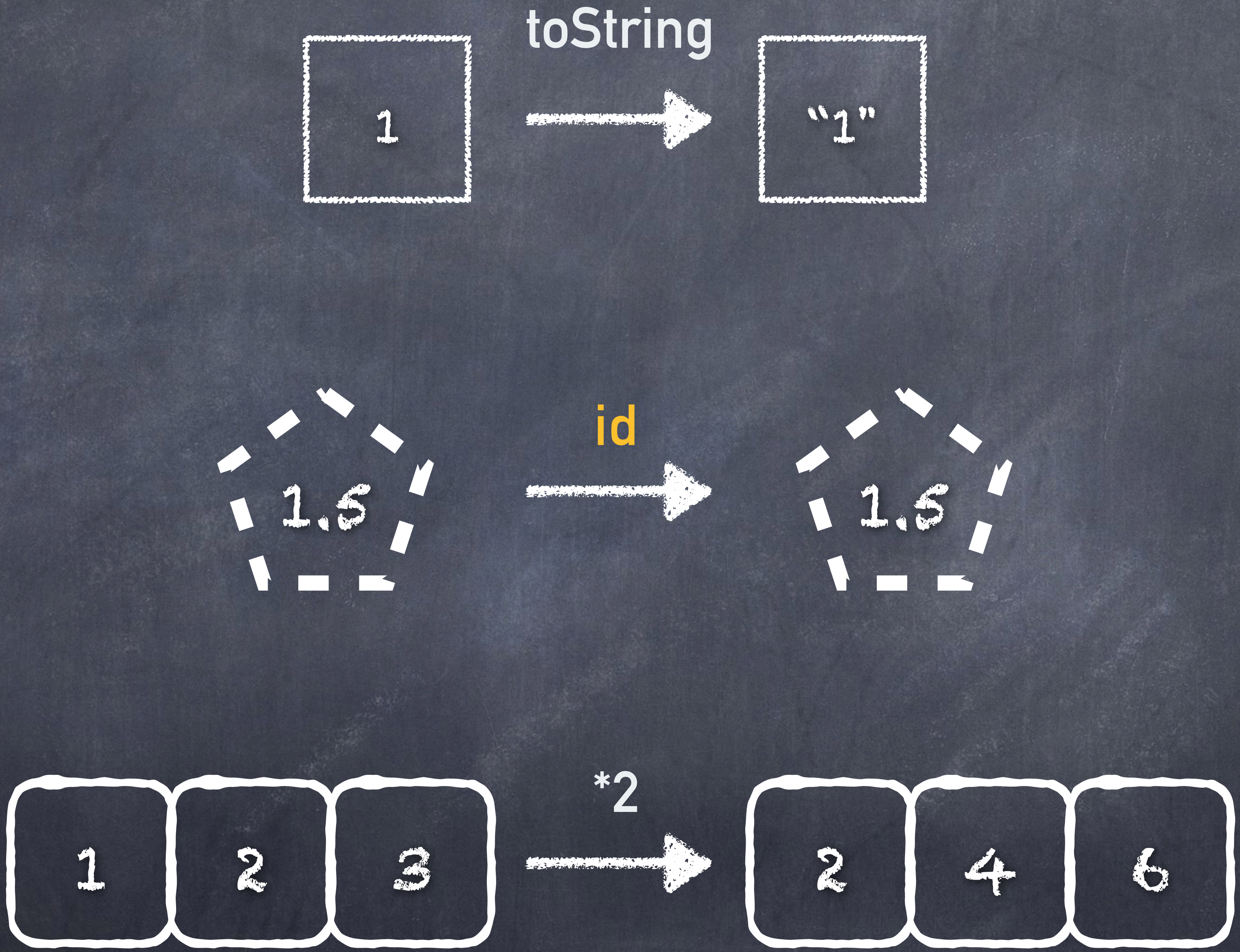
Functors



toString

id

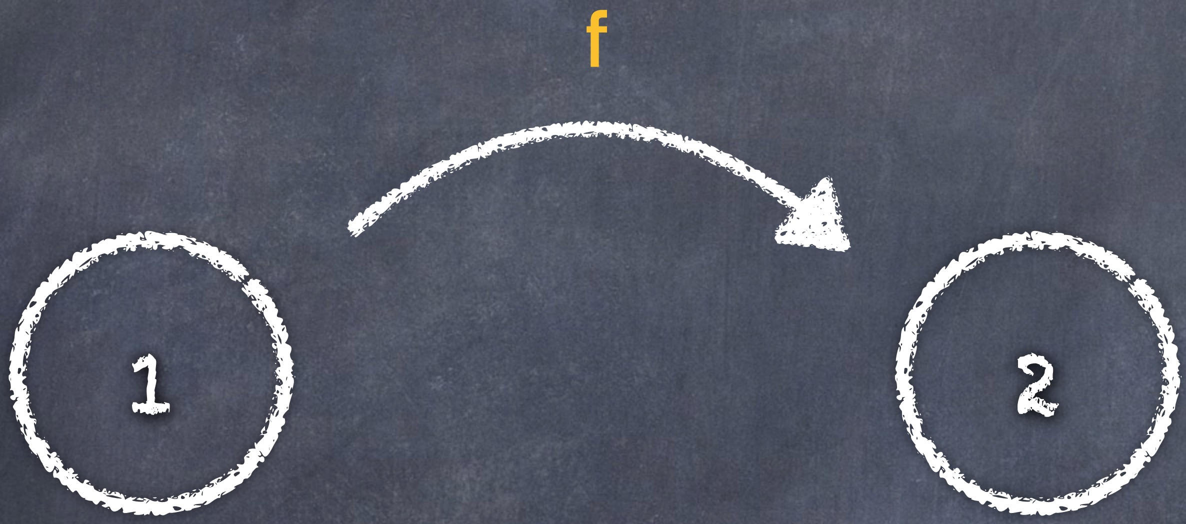
Functors



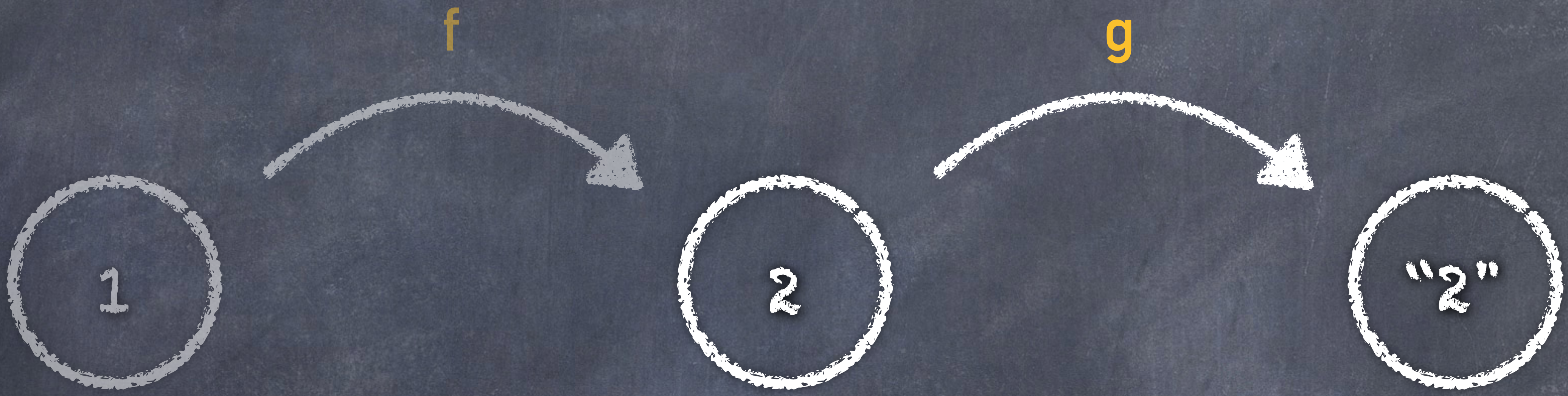
Functors

Functors

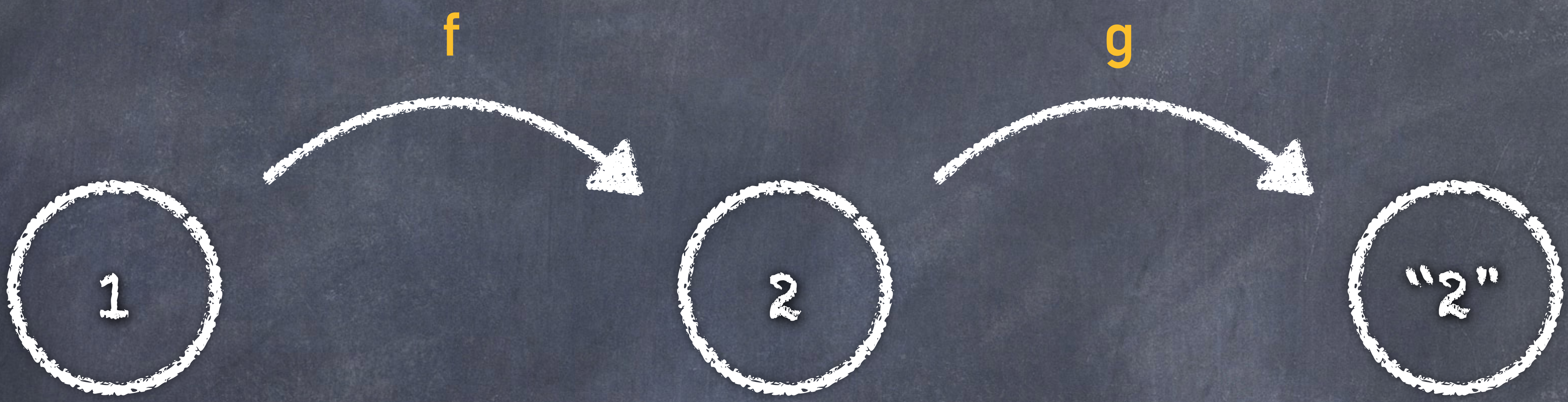
Functors



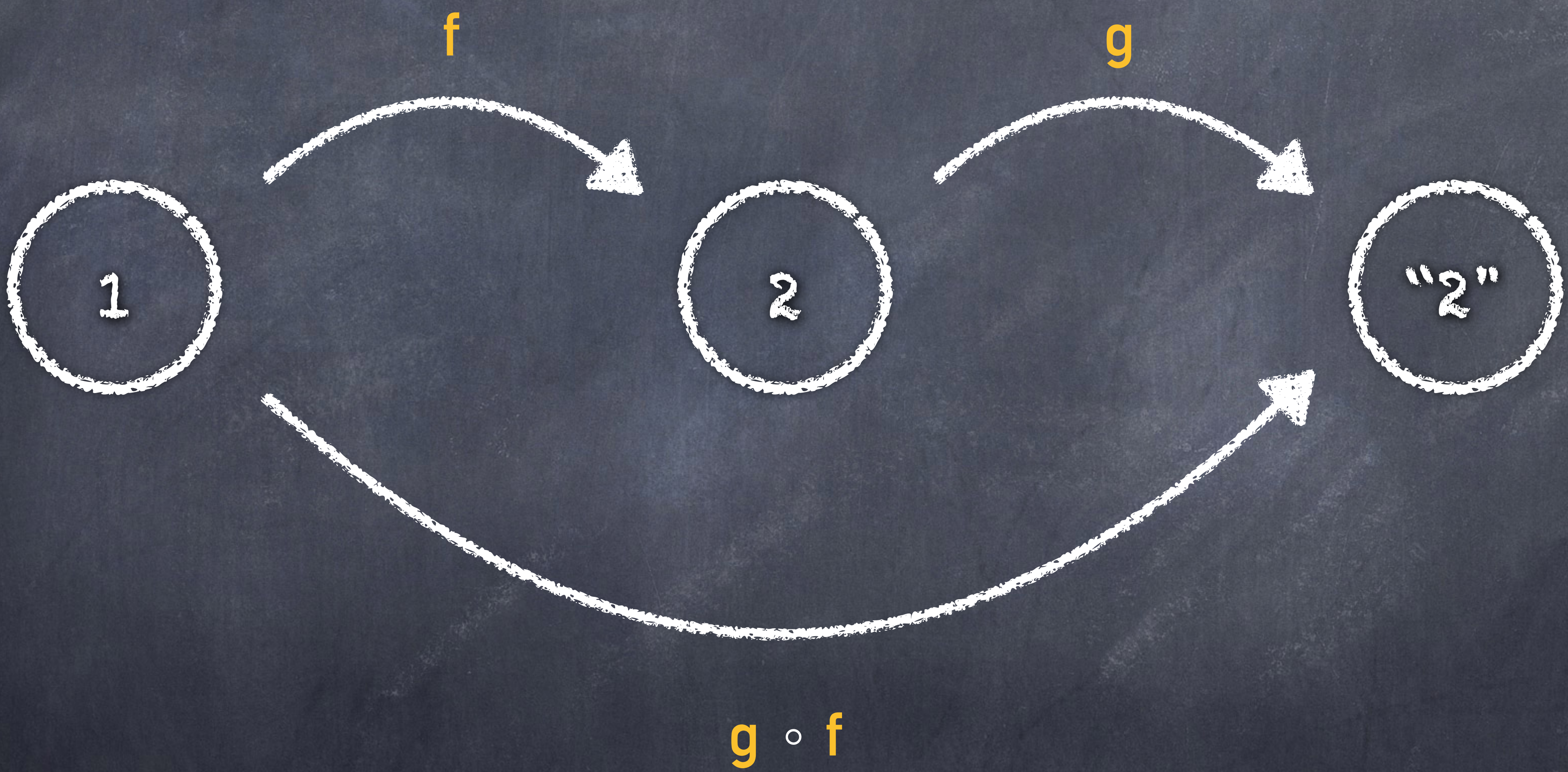
Functors



Functors



Functors



Applicative Functors

Applicative Functors

```
trait Applicative[F[_]] extends Functor[F] {
  def product[A, B](fa: F[A], fb: F[B]): F[(A, B)]
  def pure[A](a: A): F[A]
}
```

Applicative Functors

```
trait Applicative[F[_]] extends Functor[F] {
  def product[A, B](fa: F[A], fb: F[B]): F[(A, B)]
  def pure[A](a: A): F[A]
}
```

```
val liftedInt: F[Int] = F.pure(1)
```

Applicative Functors

```
trait Applicative[F[_]] extends Functor[F] {
  def product[A, B](fa: F[A], fb: F[B]): F[(A, B)]
  def pure[A](a: A): F[A]
}
```

```
val liftedInt: F[Int] = F.pure(1)

val liftedString: F[String] = F.pure(“a")
```

Applicative Functors

```
trait Applicative[F[_]] extends Functor[F] {
  def product[A, B](fa: F[A], fb: F[B]): F[(A, B)]
  def pure[A](a: A): F[A]
}
```

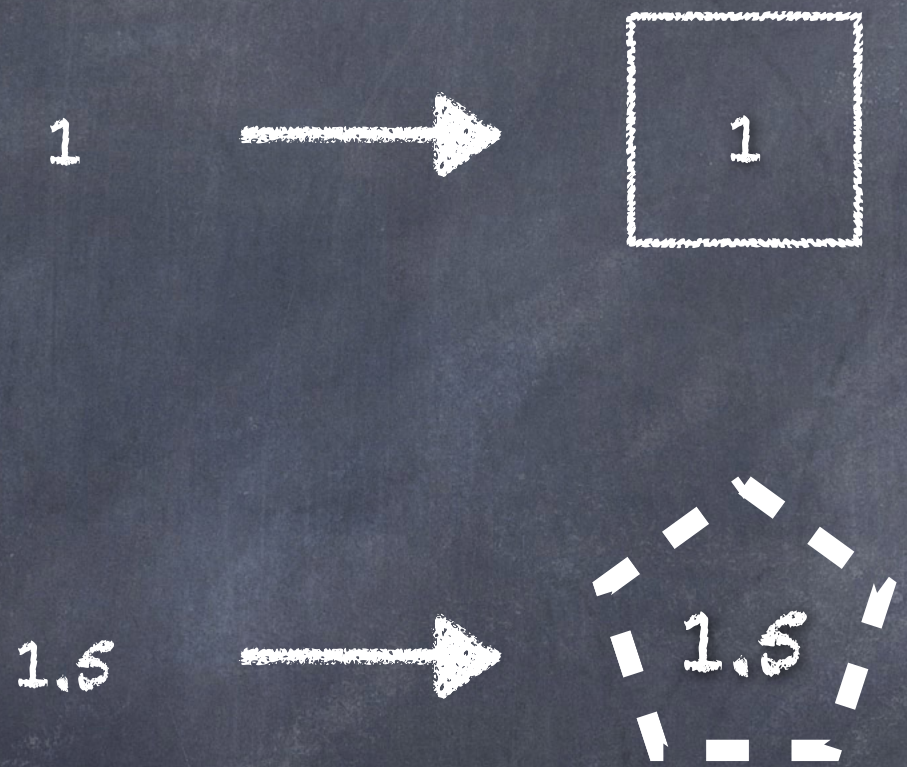
```
val liftedInt: F[Int] = F.pure(1)

val liftedString: F[String] = F.pure(“a")

val res: F[(Int,String)] = F.product(liftedInt, liftedString)
```

Applicative Functors

Applicative Functors

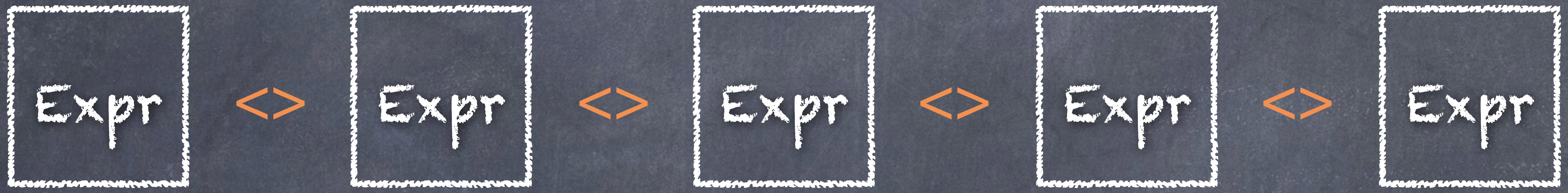
$$1 \longrightarrow \boxed{1}$$

Applicative Functors



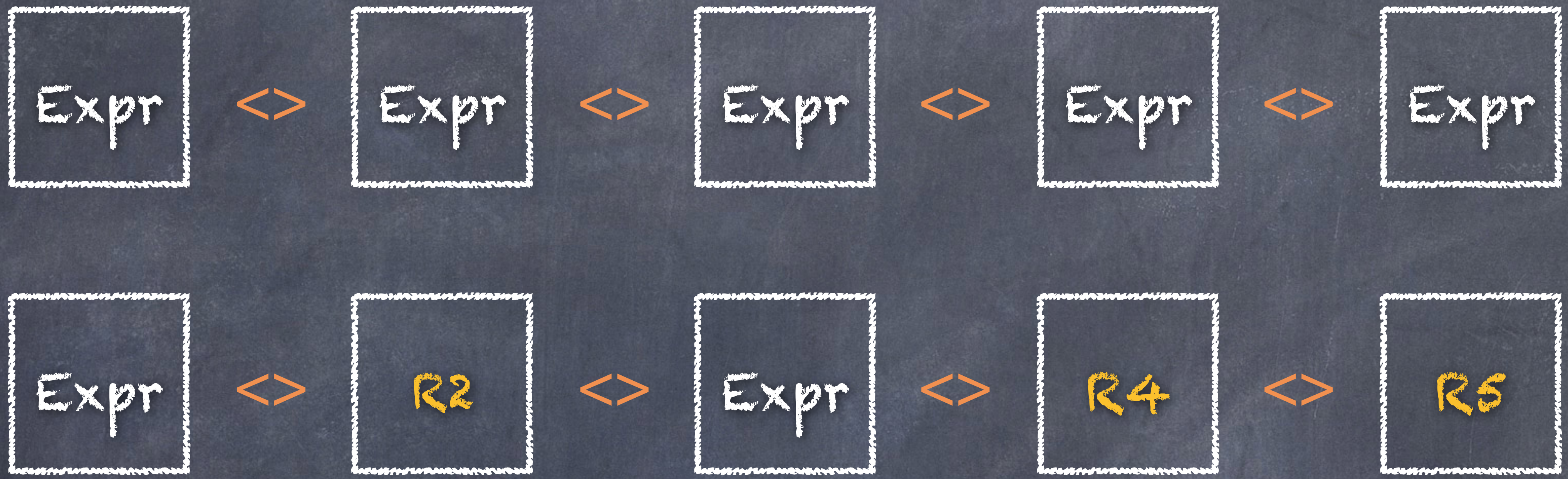
Applicative Functors

Applicative Functors



Expr <> Expr <> Expr <> Expr <> Expr

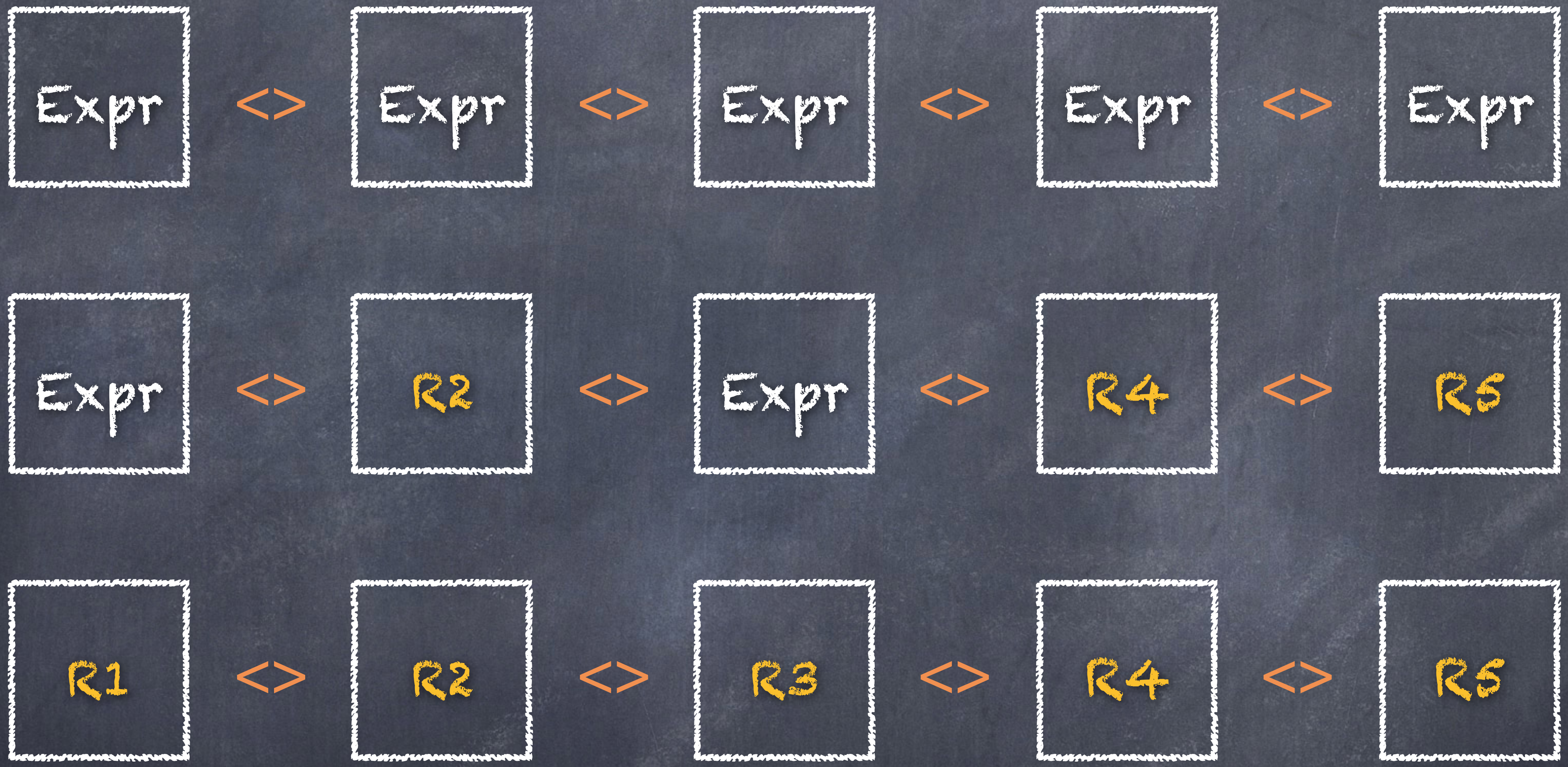
Applicative Functors



Expr <> Expr <> Expr <> Expr <> Expr

Expr <> R2 <> Expr <> R4 <> R5

Applicative Functors

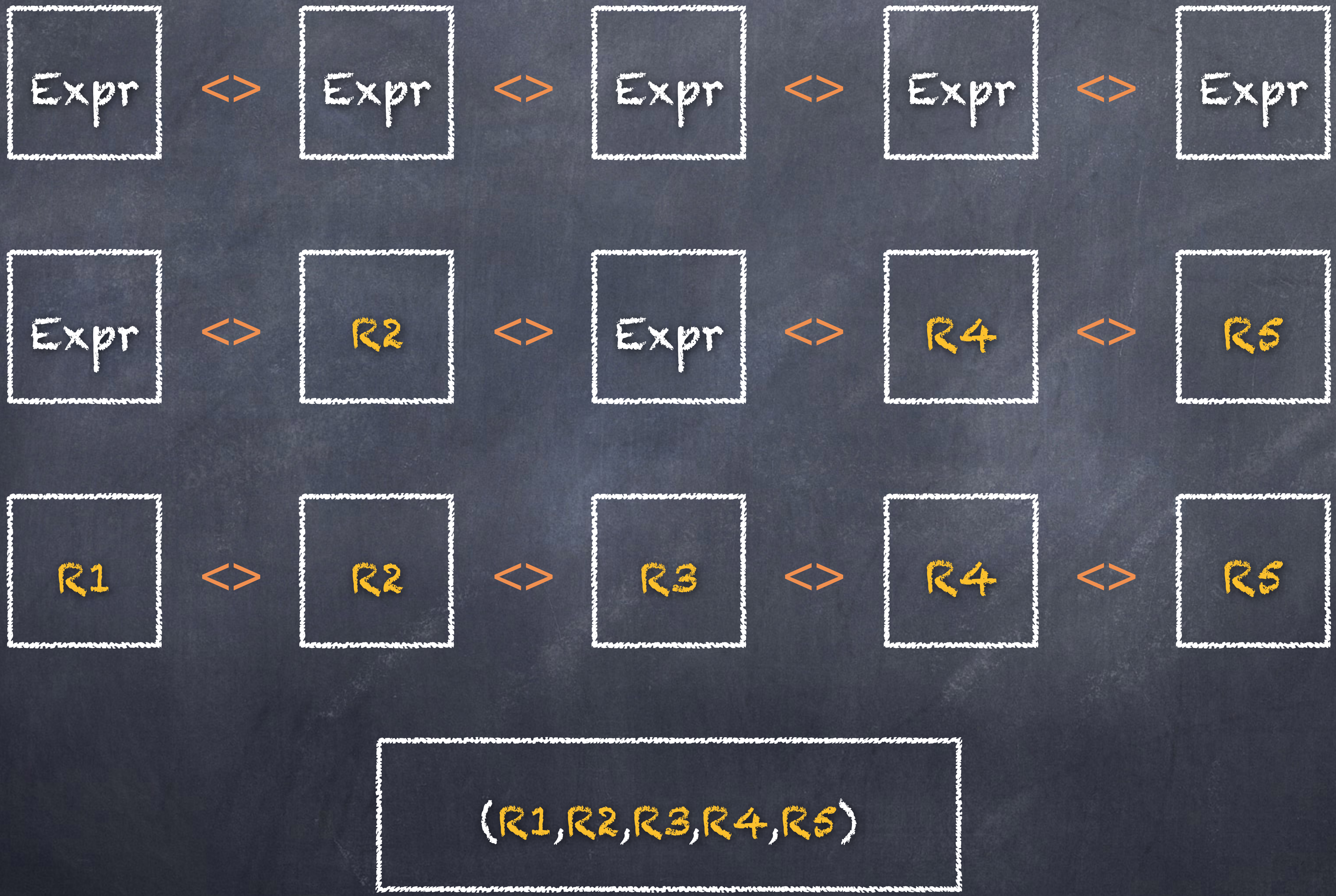


Expr <> Expr <> Expr <> Expr <> Expr

Expr <> R2 <> Expr <> R4 <> R5

R1 <> R2 <> R3 <> R4 <> R5

Applicative Functors



Expr <> Expr <> Expr <> Expr <> Expr

Expr <> R2 <> Expr <> R4 <> R5

R1 <> R2 <> R3 <> R4 <> R5

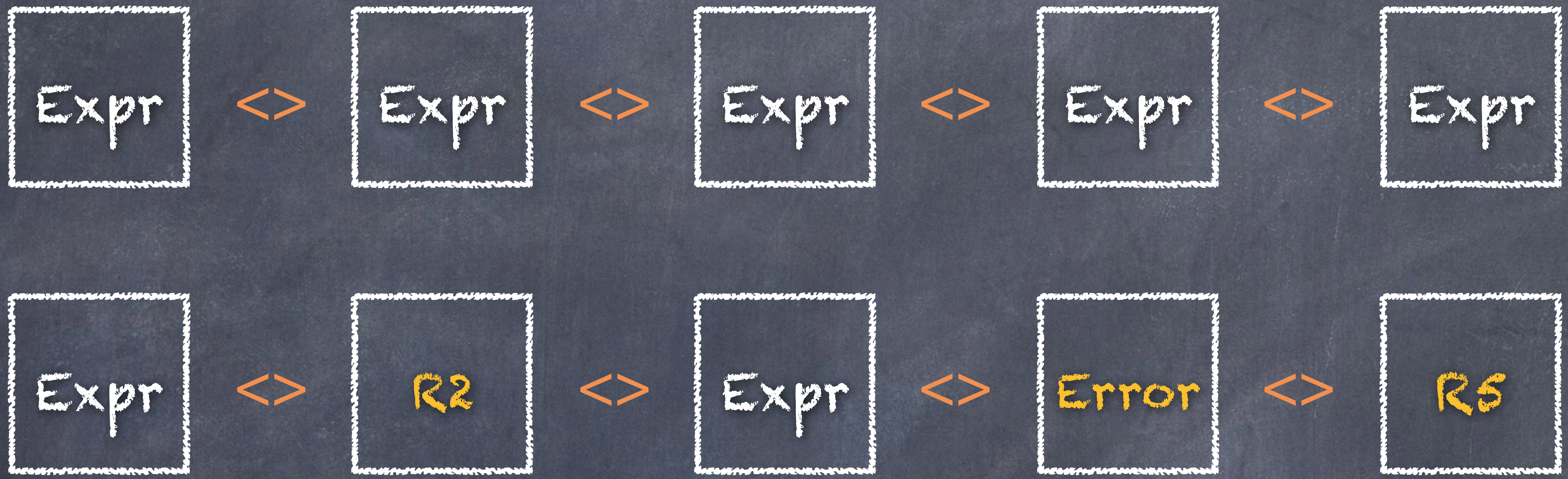
(R1,R2,R3,R4,R5)

Applicative Functors

Applicative Functors

Expr <> Expr <> Expr <> Expr <> Expr

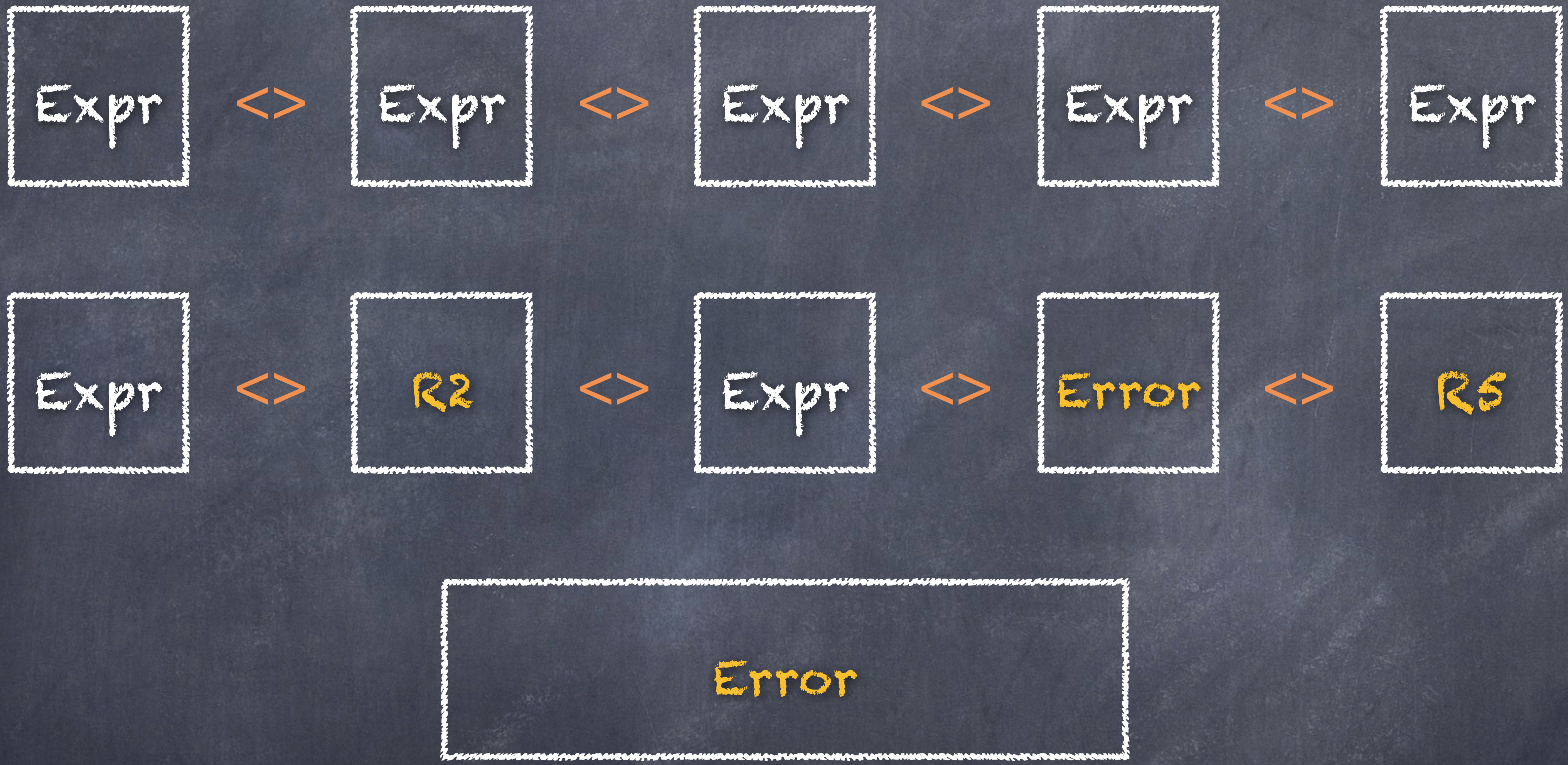
Applicative Functors



Expr <> Expr <> Expr <> Expr <> Expr

Expr <> R2 <> Expr <> Error <> R5

Applicative Functors



Expr <> Expr <> Expr <> Expr <> Expr

Expr <> R2 <> Expr <> Error <> R5

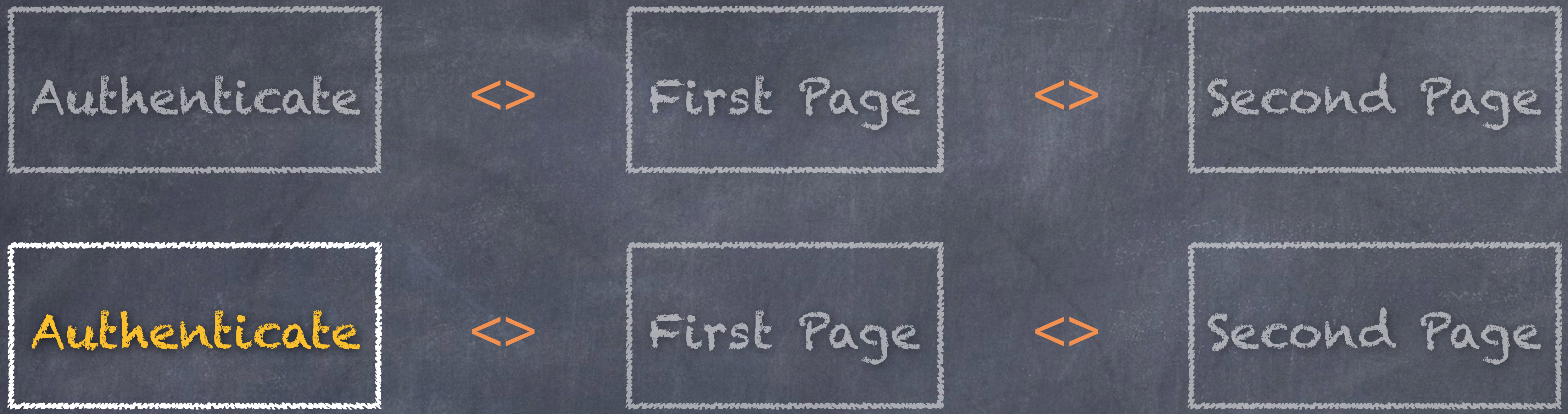
Error

Applicative Functors

Applicative Functors

Authenticate <> First Page <> Second Page

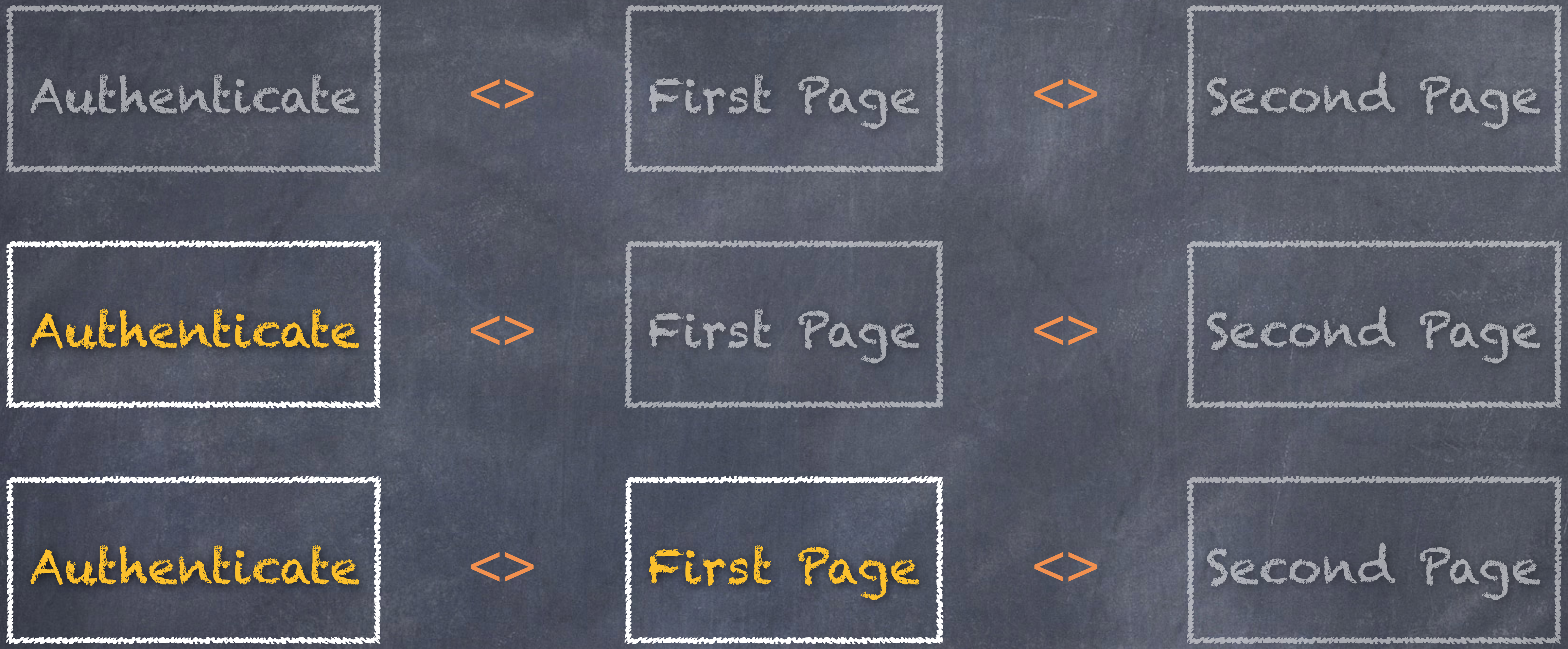
Applicative Functors



Authenticate <> First Page <> Second Page

Authenticate <> First Page <> Second Page

Applicative Functors

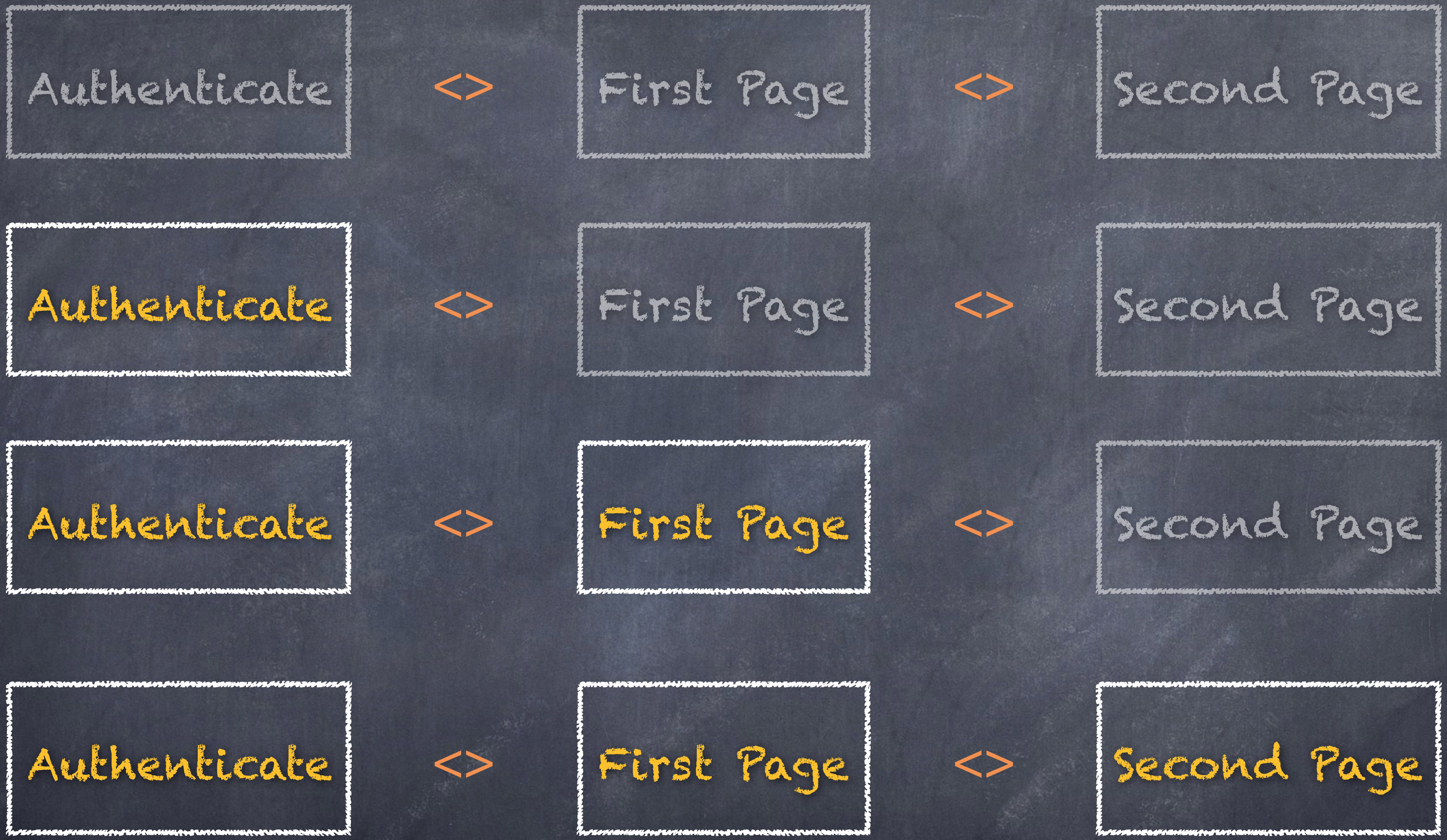


Authenticate <> First Page <> Second Page

Authenticate <> First Page <> Second Page

Authenticate <> **First Page** <> Second Page

Applicative Functors



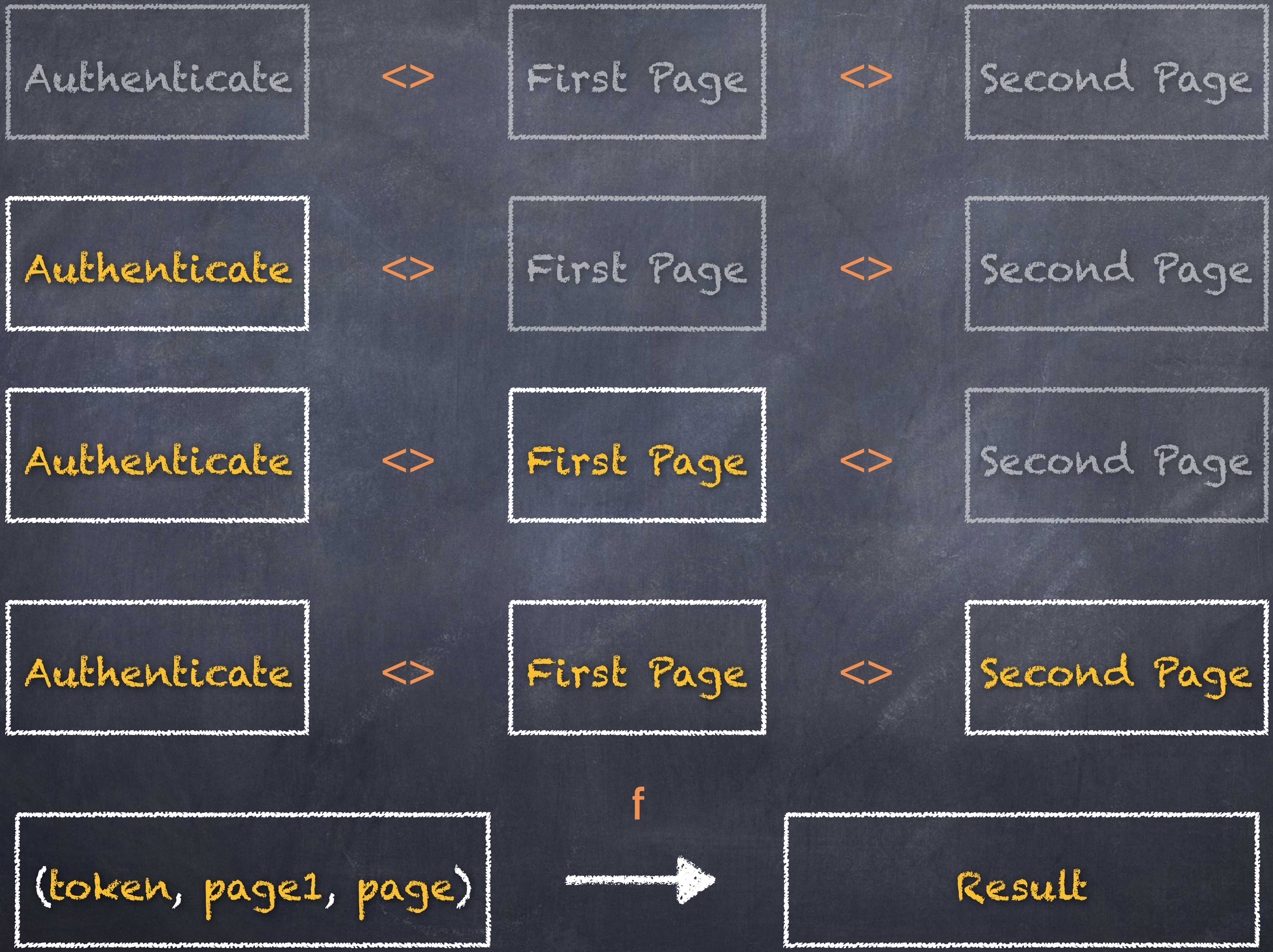
Authenticate <> First Page <> Second Page

Authenticate <> First Page <> Second Page

Authenticate <> First Page <> Second Page

Authenticate <> First Page <> Second Page

Applicative Functors



Authenticate <> First Page <> Second Page

Authenticate <> First Page <> Second Page

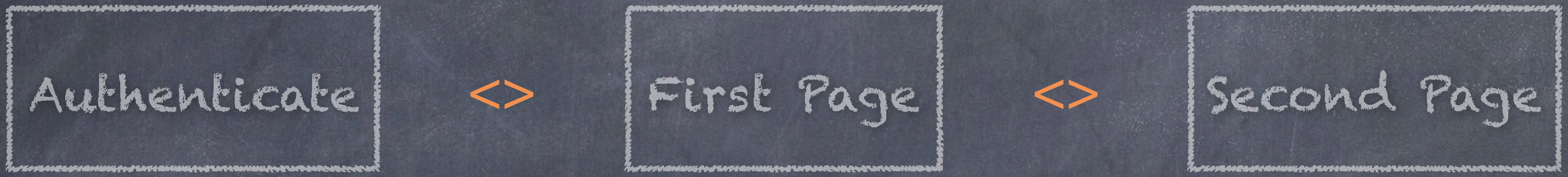
Authenticate <> First Page <> Second Page

Authenticate <> First Page <> Second Page

(token, page1, page) $\xrightarrow{f}$ Result

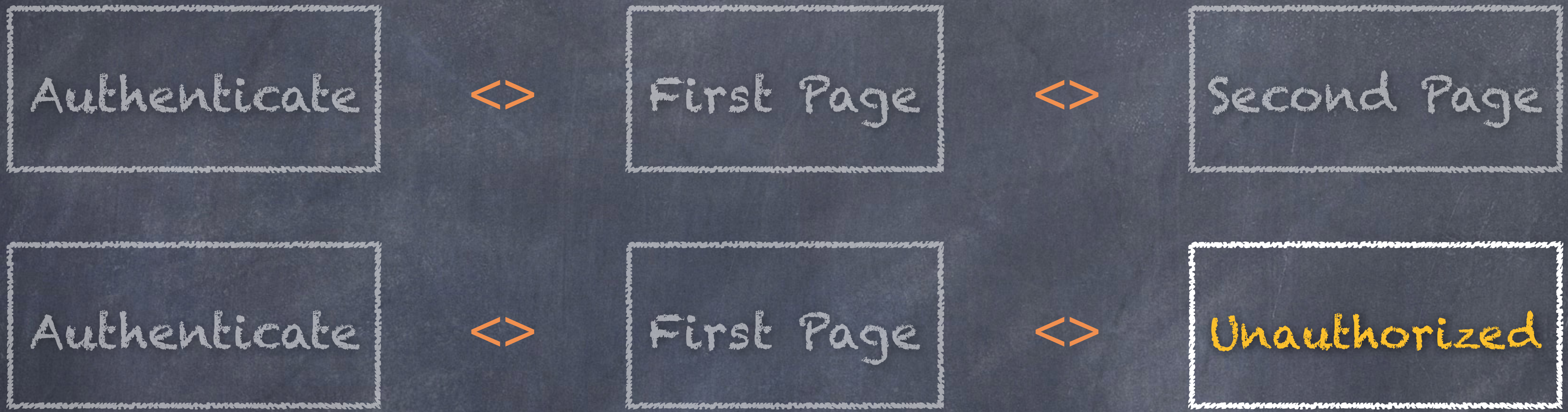
Applicative Functors

Applicative Functors



Authenticate <> First Page <> Second Page

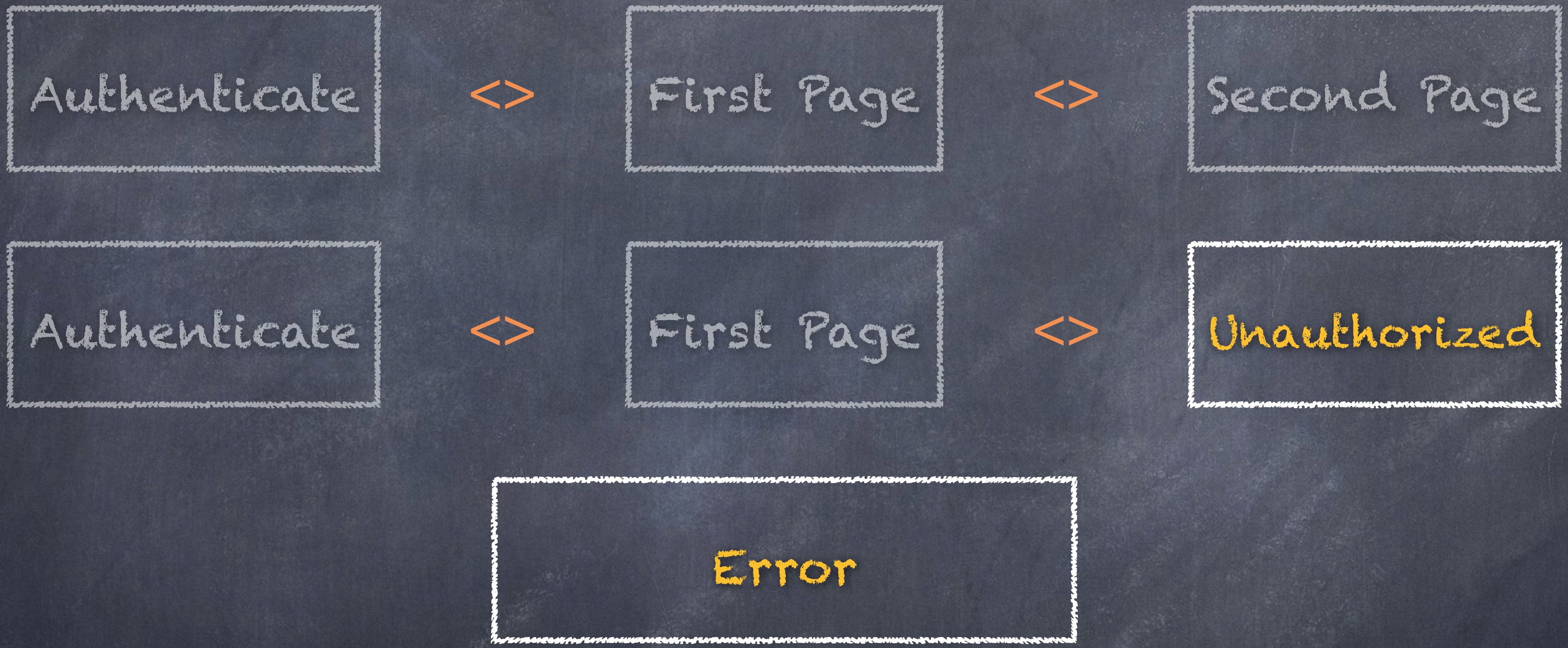
Applicative Functors



Authenticate <> First Page <> Second Page

Authenticate <> First Page <> Unauthorized

Applicative Functors



Authenticate <> First Page <> Second Page

Authenticate <> First Page <> Unauthorized

Error

Monads

Monads

```
trait Monad[F[_]] {
  def flatMap[A, B](a: A): F[B]
  def return[A](a: A): F[A]
}
```

Monads

```
trait Monad[F[_]] {
  def flatMap[A, B](a: A): F[B]
  def return[A](a: A): F[A]
}
```

```
val liftedInt: F[Int] = F.return(1)
```

Monads

```
trait Monad[F[_]] {
  def flatMap[A, B](a: A): F[B]
  def return[A](a: A): F[A]
}
```

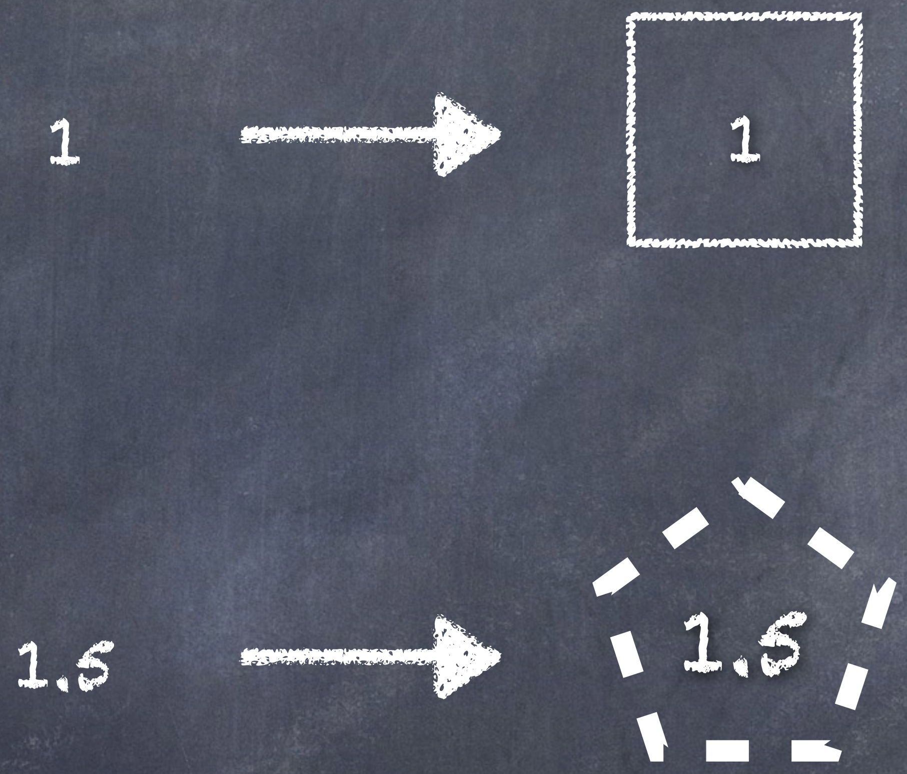
```
val liftedInt: F[Int] = F.return(1)

val liftedString: F[String] = liftedInt.flatMap(a => F.return(a.toString))
```

Monads

Monads

$$1 \longrightarrow \boxed{1}$$

Monads



Monads

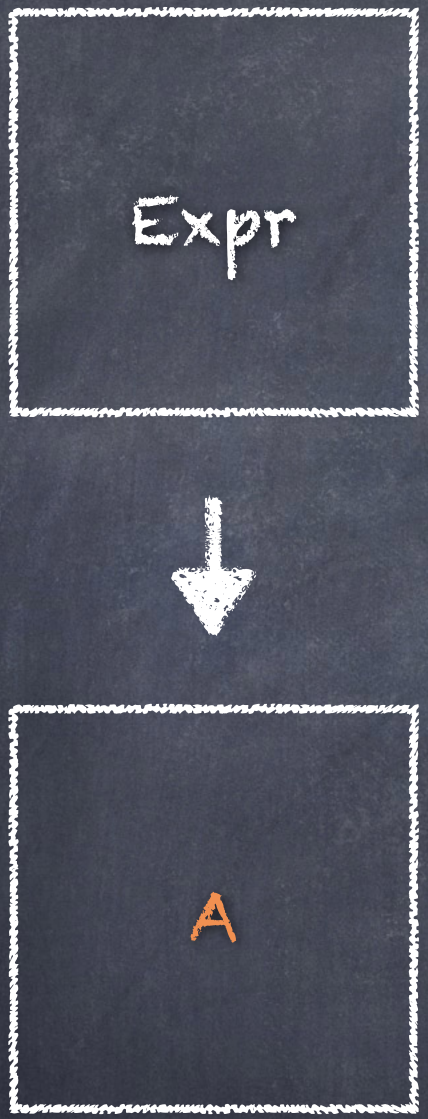
return and pure are the same function!

Monads

Monads

Expr

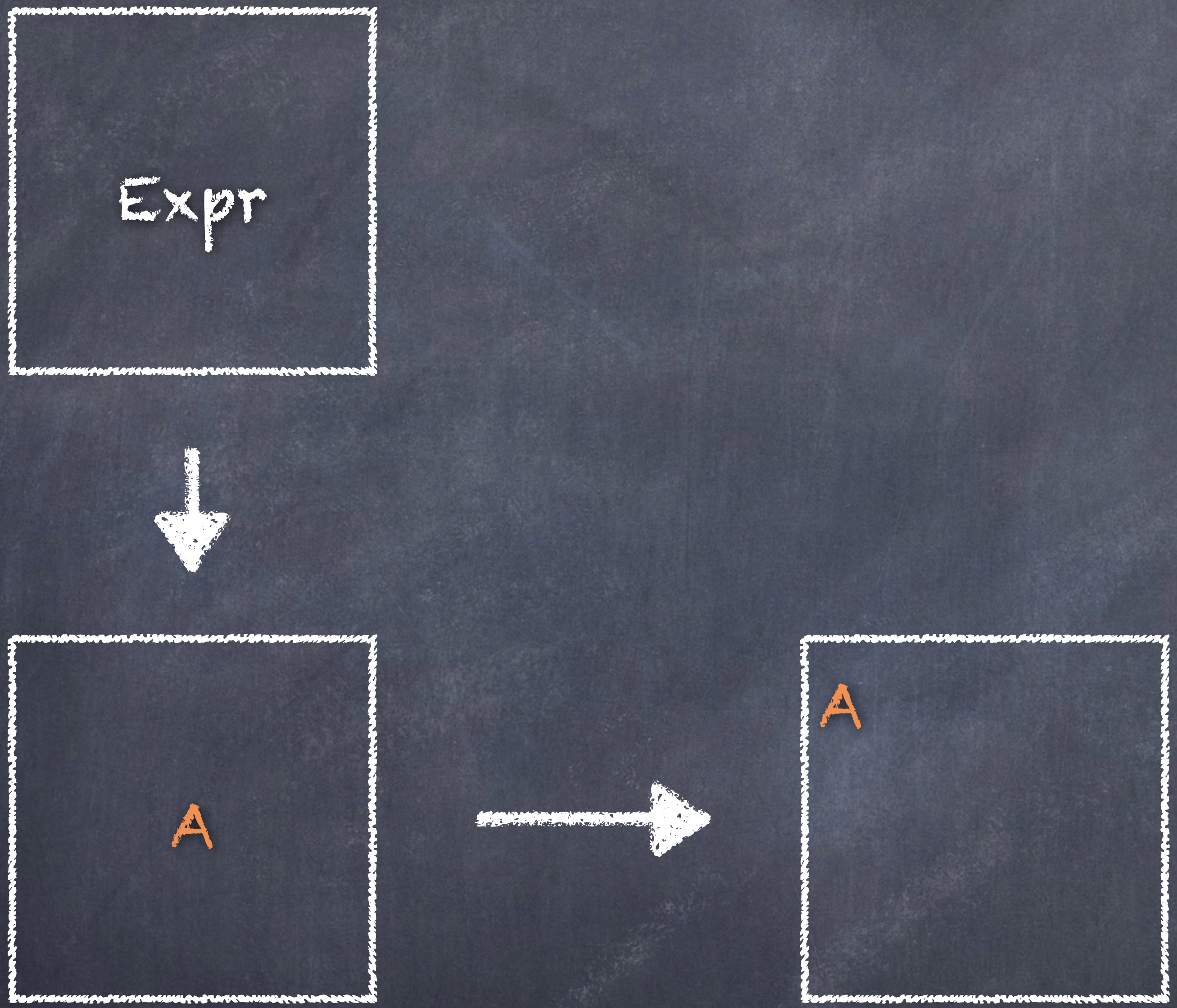
Monads



Expr

A

Monads

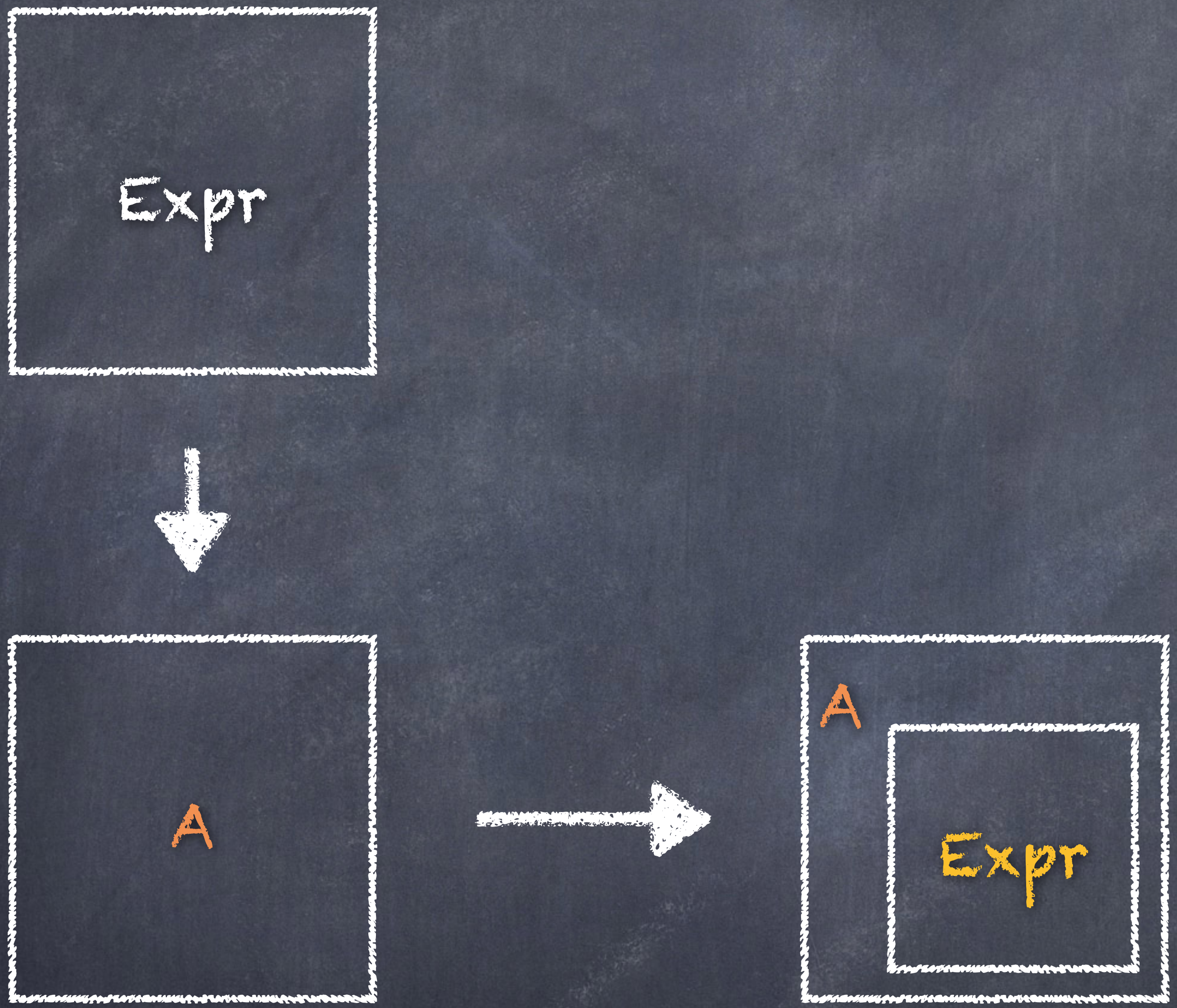


Expr

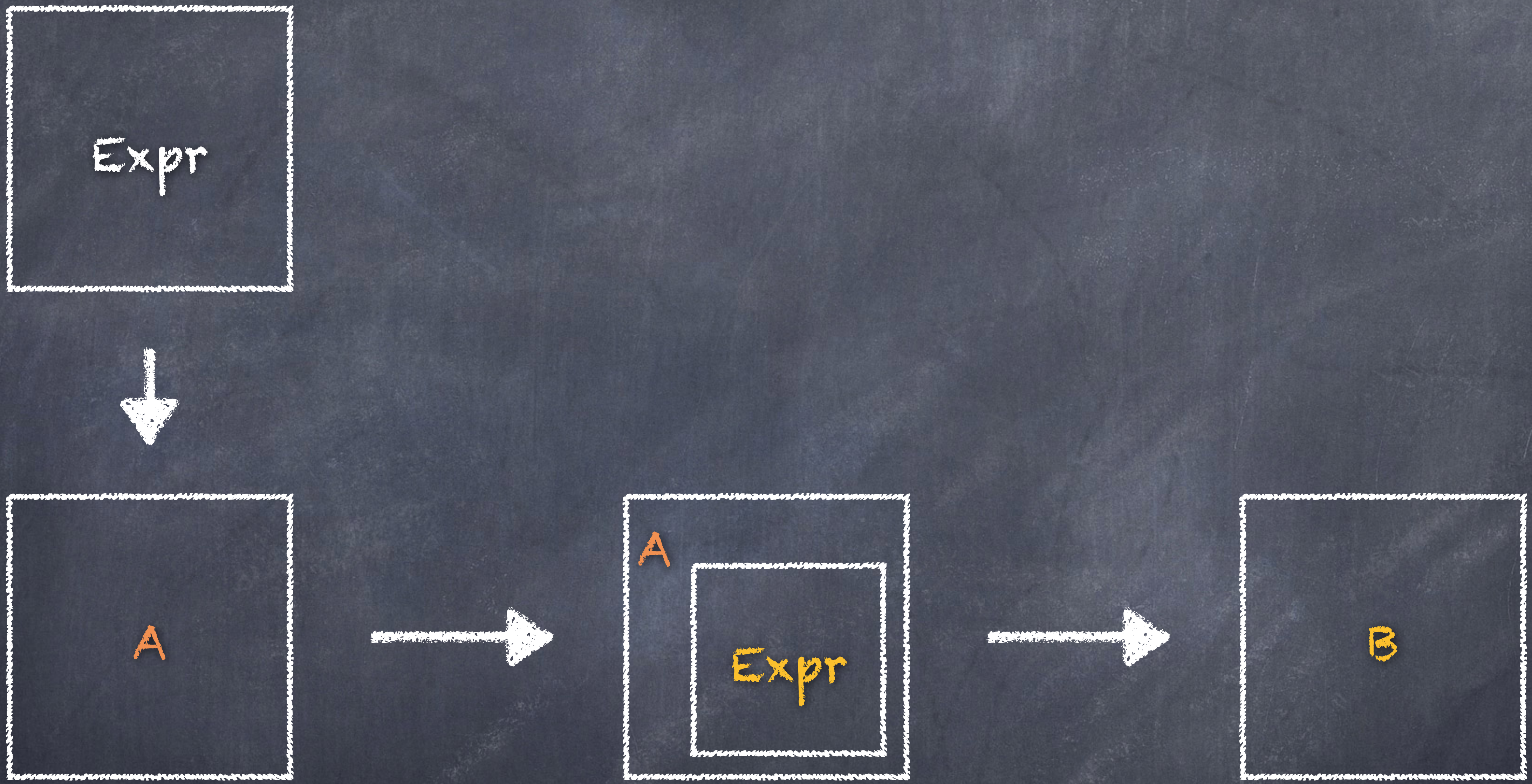
A

A

Monads



Monads

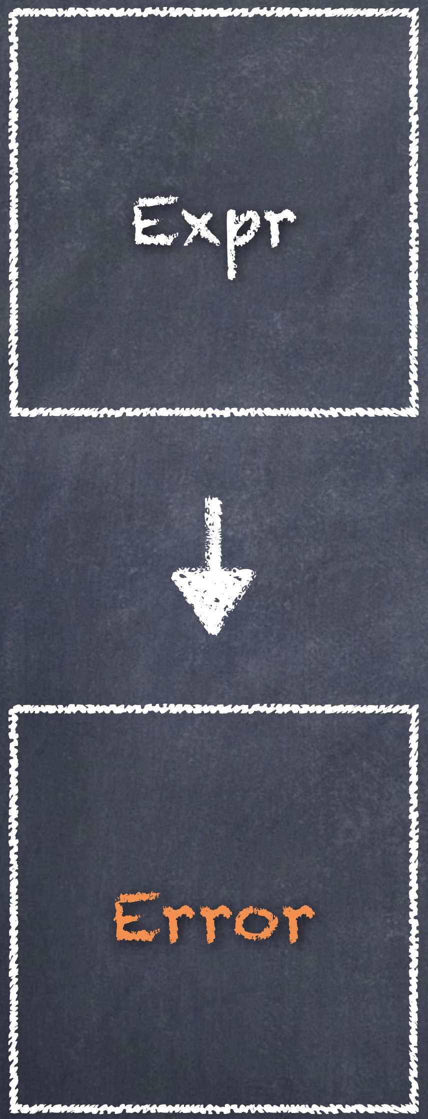


Monads

Monads

Expr

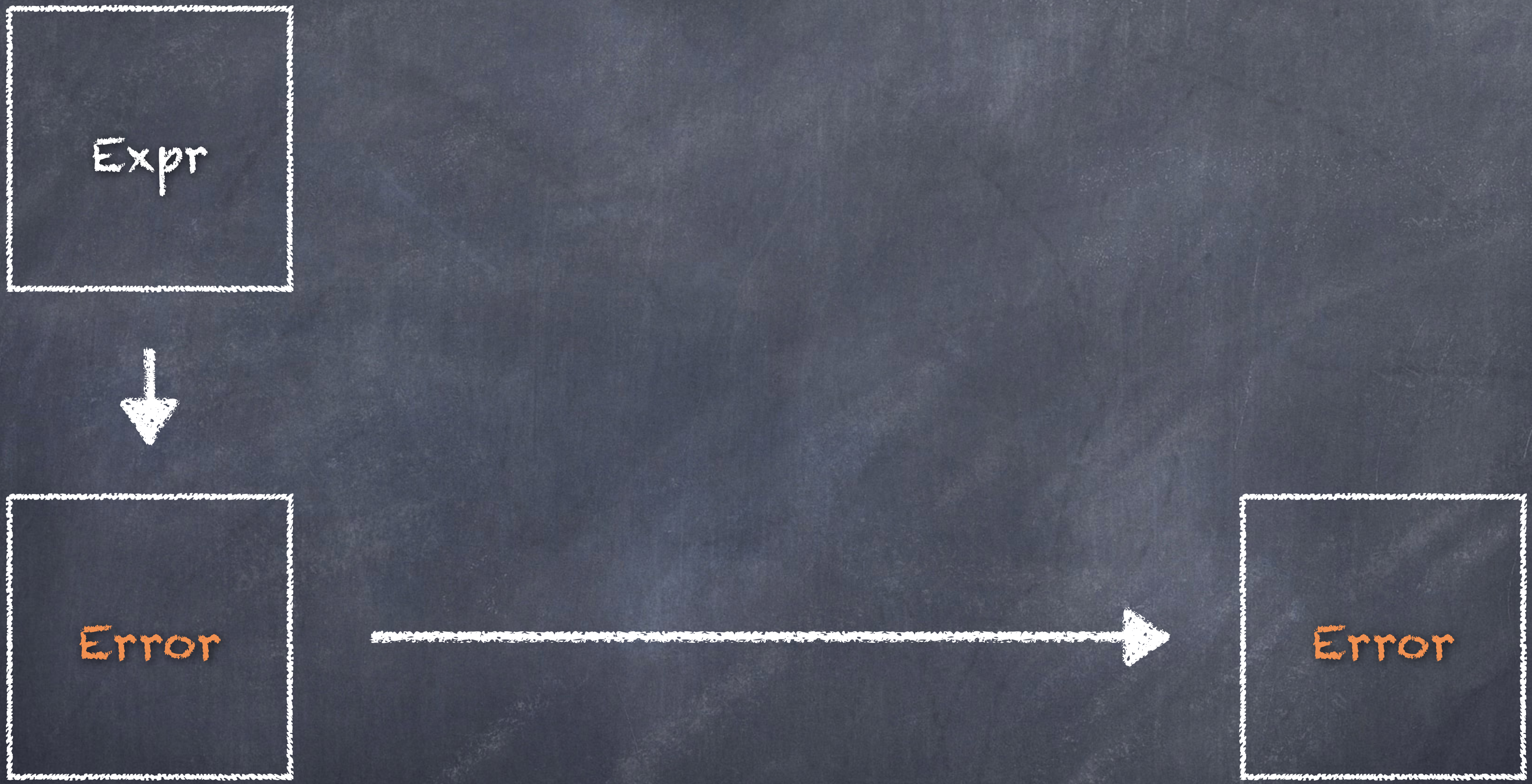
Monads



Expr

Error

Monads



Expr

Error

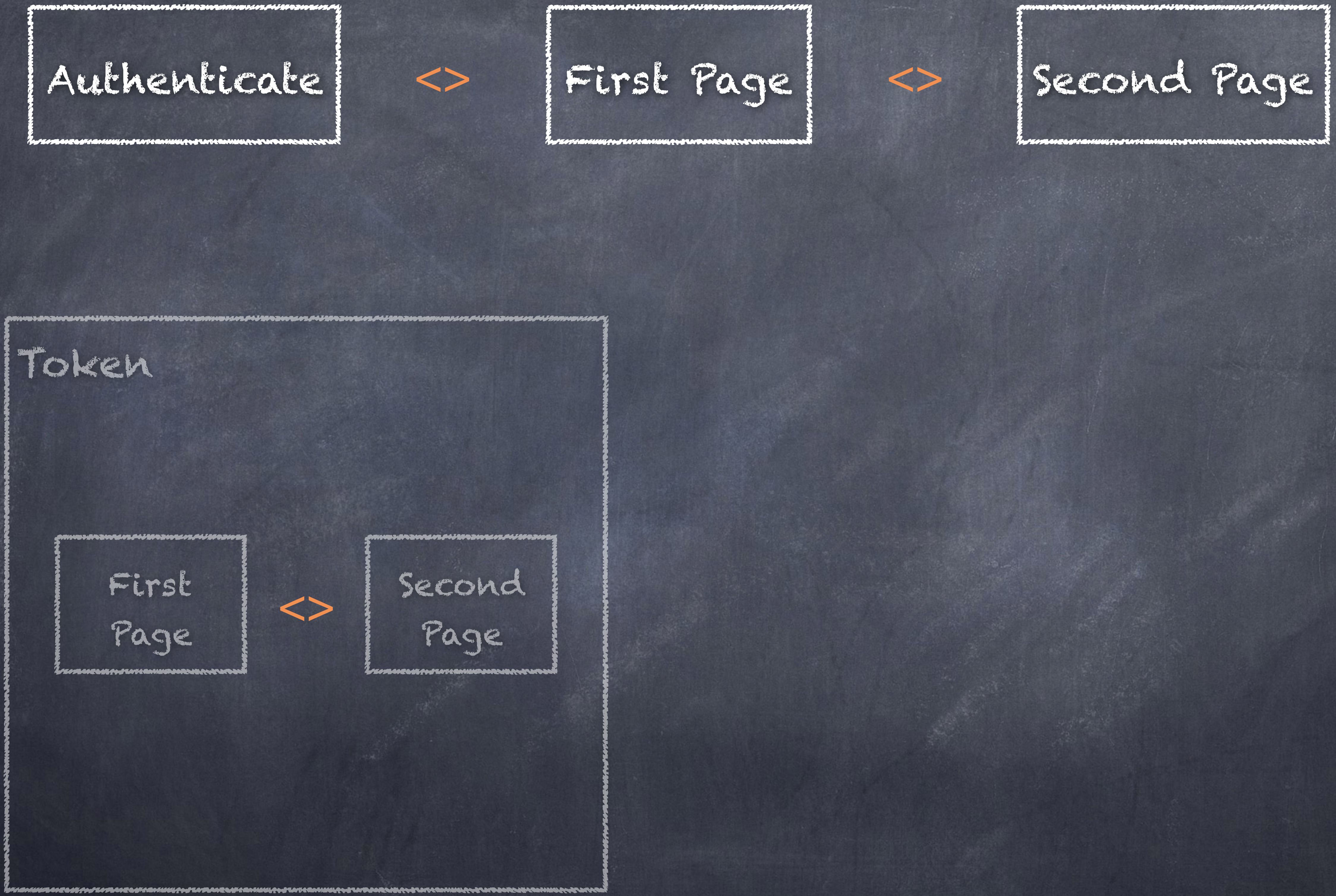
Error

Monads

Monads

Authenticate <> First Page <> Second Page

Monads

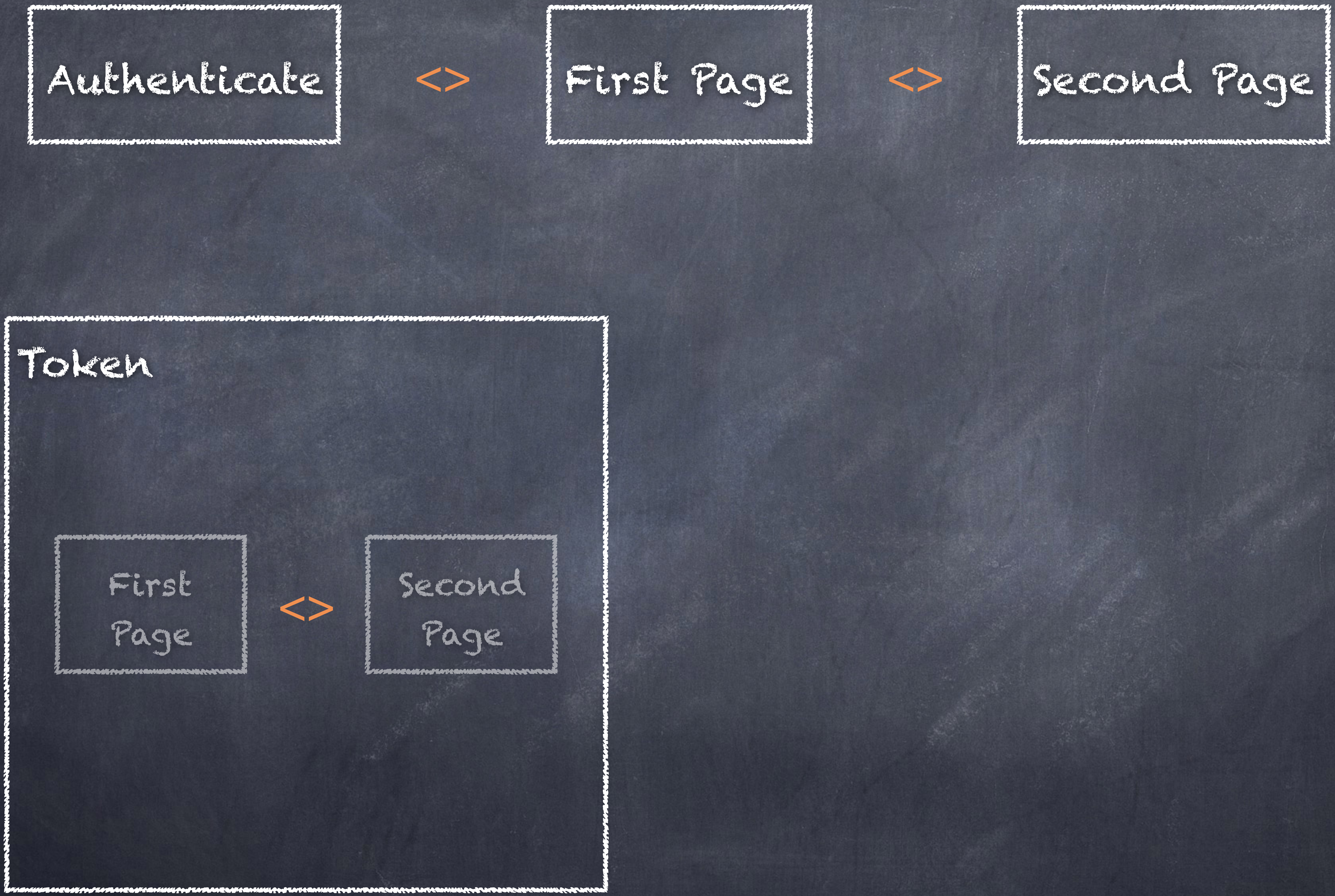


Authenticate <> First Page <> Second Page

Token

First Page <> Second Page

Monads

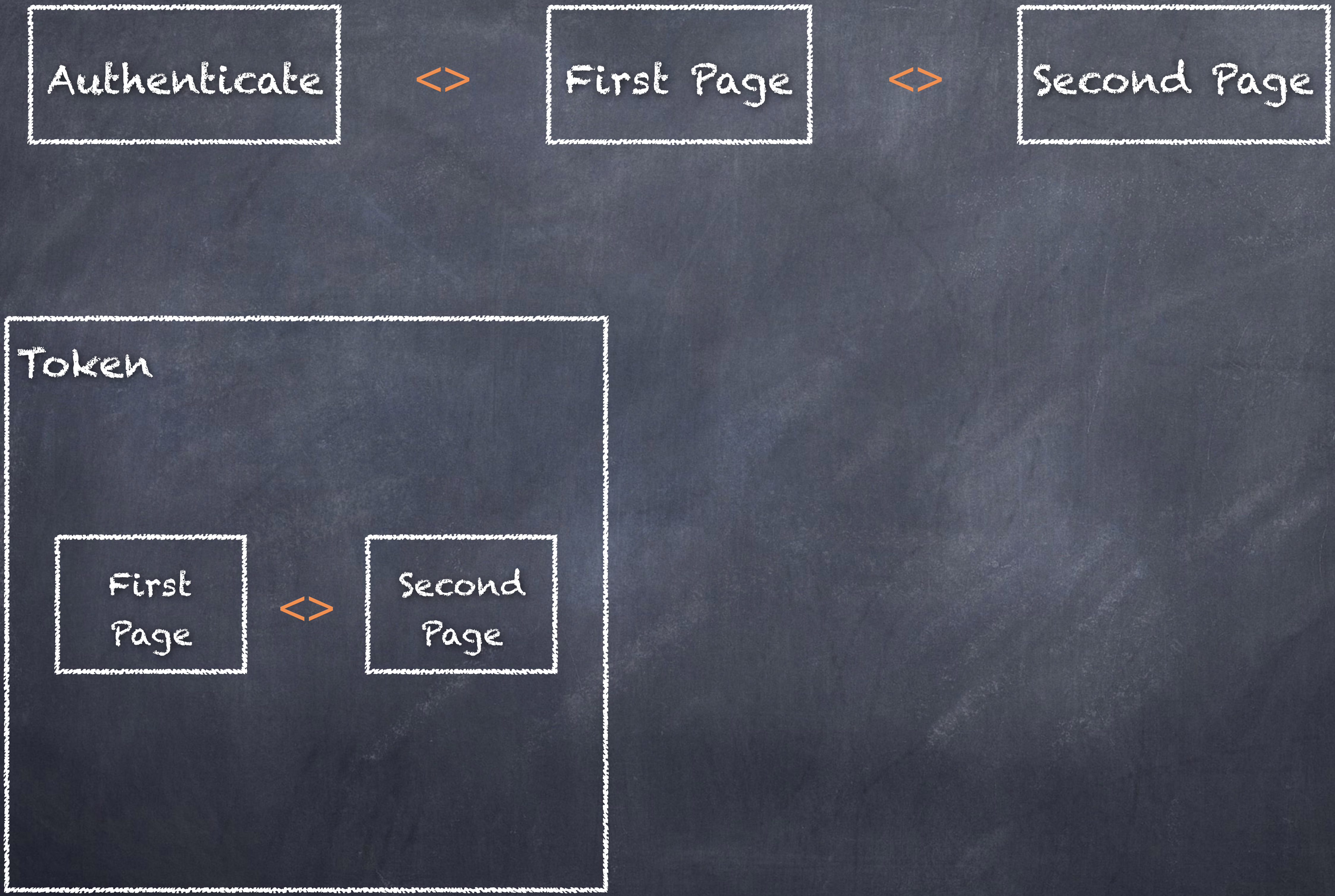


Authenticate <> First Page <> Second Page

Token

First Page <> Second Page

Monads

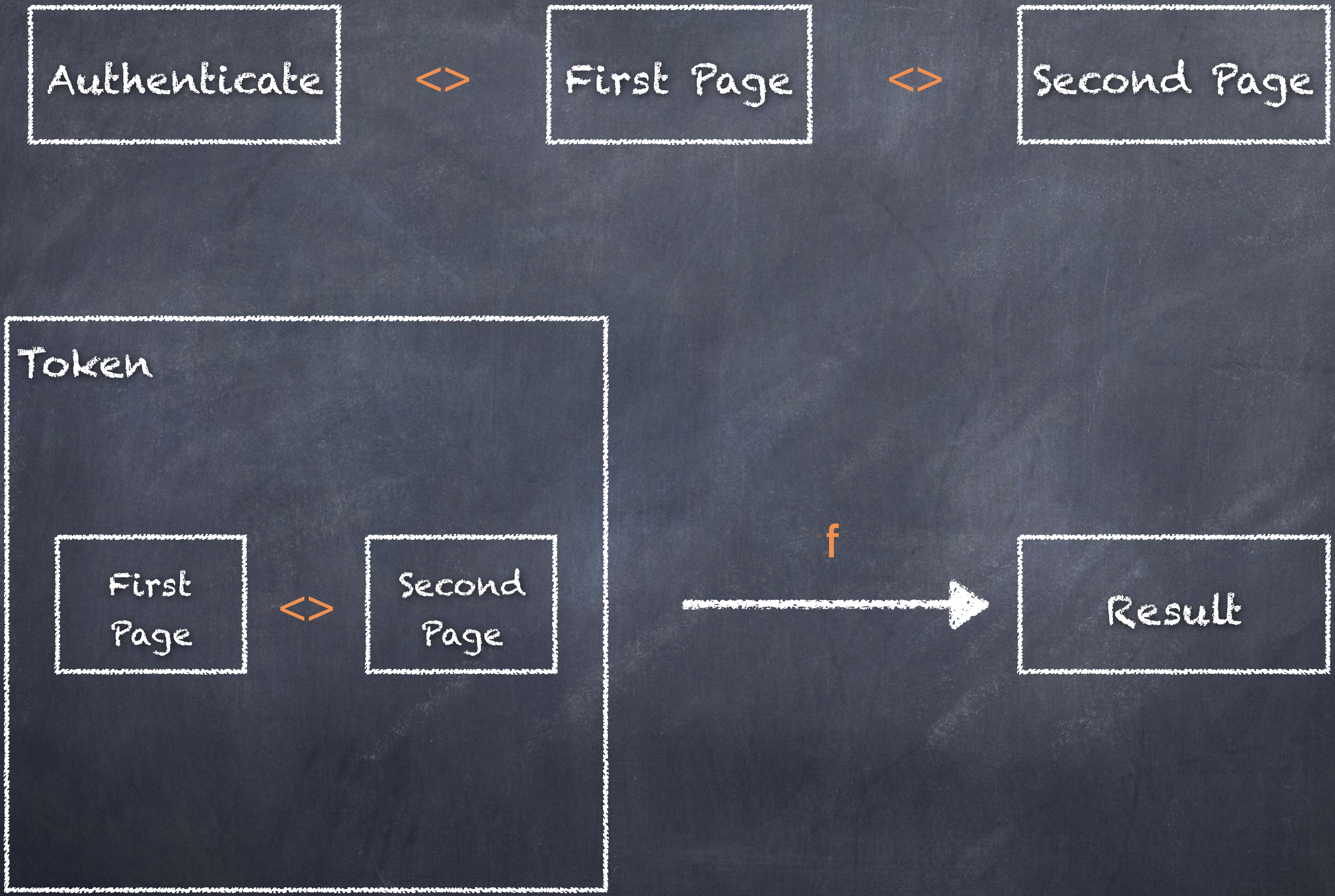


Authenticate <> First Page <> Second Page

Token

First Page <> Second Page

Monads



Authenticate <> First Page <> Second Page

Token

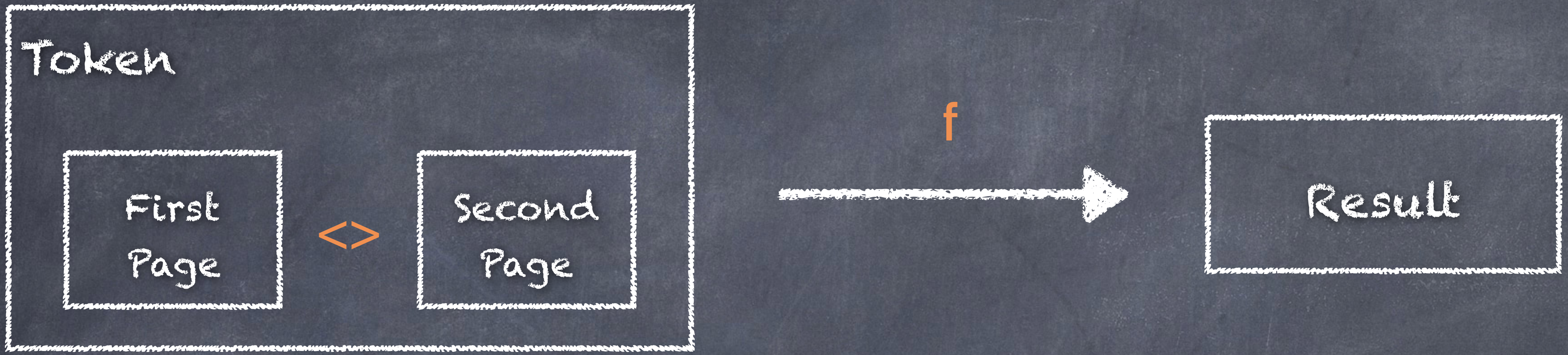
First Page <> Second Page

f

Result

Monads

Monads



Token

First Page

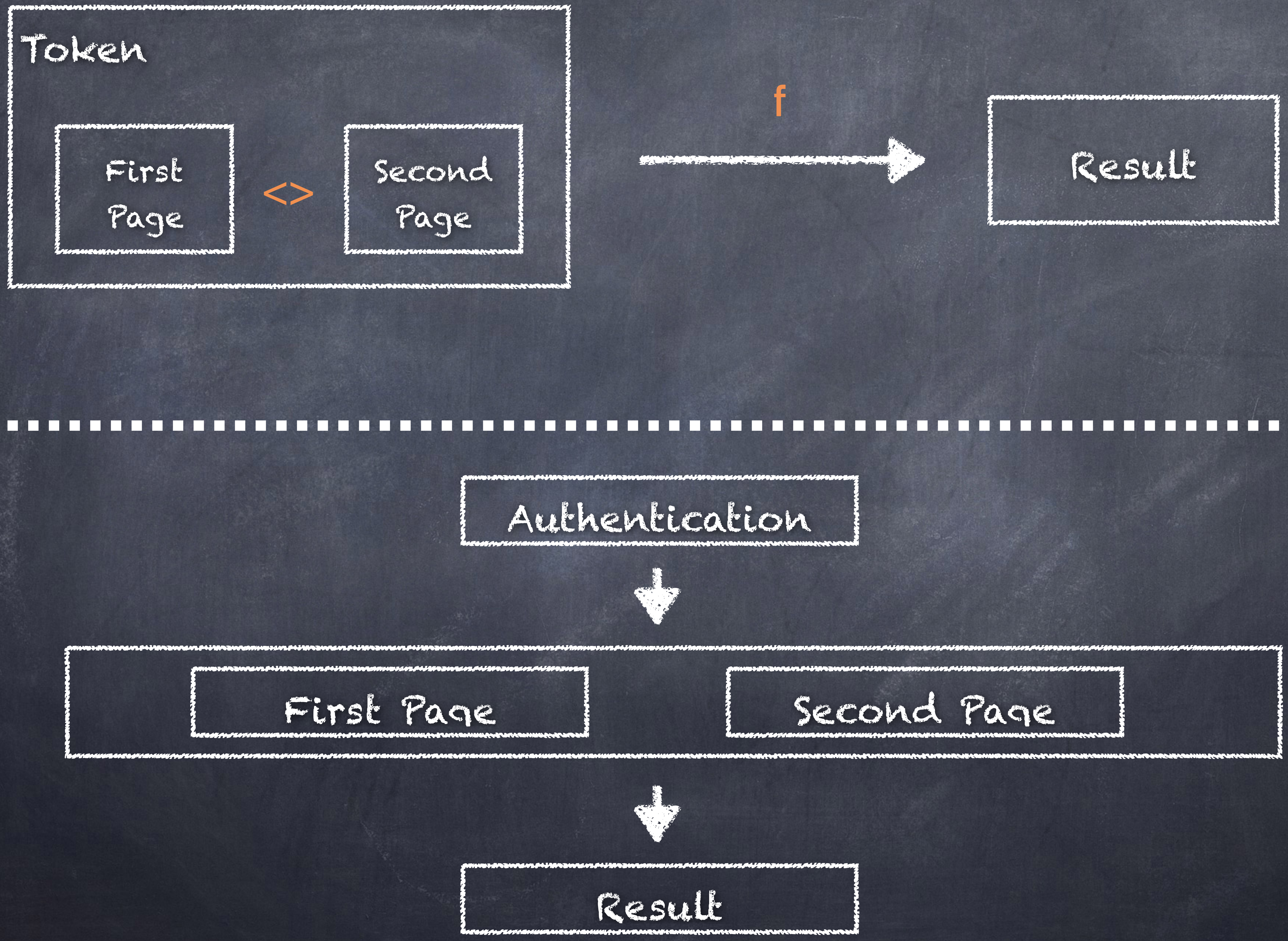
<>

Second Page

f

Result

Monads



Token

First Page

<>

Second Page

f

Result

Authentication

First Page

Second Page

Result

FP Intuition

Dependency	flatMap	Monads
Parallelism	zip/traverse	Applicatives

Conclusion

While FP is about composition,

Parallelism is about evaluation.

Through the composition of

Monads with Applicatives

it's possible to achieve: Simple, reliable and STYLISH

Concurrency and Parallelism.

Thank you!!!!

Questions?