FP in JAVA 8

JUG Barcelona

sponsored by

Ignasi Marimon-Clos (@ignasi35)

thanks!

@ignasi35

about you

about me

@ignasi35

1) problem solver, Garbage Collector, mostly scala, java 8, agile for tech teams

2) kayaker

3) under construction

4) all things JVM

FP in Java 8

Pure Functions

no side effects

if not used, remove it

fixed in — fixed out

FP in Java 8

```
(T, R) -> Q
```

FP in Java 8

`Supplier<T>`	`() -> T`
`Function<T,R>`	`(T) -> R`
`BiFunction<T,R,Q>`	`(T, R) -> Q`
`Predicate<T>`	`(T) -> boolean`
`Consumer<T>`	`(T) ->`

currying

```
(T, R) -> (Q) -> (S) -> J
```

BiArgumentedPrototipicalFactoryFactoryBean

KEEP
CALM
AND
CURRY
ON

thanks!

@ignasi35

End of presentation


WALLY, I DISCOVERED A DEADLY SAFETY FLAW IN OUR PRODUCT. WHO SHOULD I INFORM?
NO ONE. THE STOCK WOULD PLUNGE AND WE'D HAVE MASSIVE LAYOFFS. YOUR CAREER WOULD BE RUINED.
BUT MY NEGLIGENCE COULD CAUSE THE DEATHS OF A DOZEN CUSTOMERS.
THE FIRST DOZEN IS ALWAYS THE HARDEST.
scottadams@aol.com
www.dilbert.com
© 2004 Scott Adams, Inc./Dist. by UFS, Inc.
1-21-04

@ignasi35

Patient Pill Sizes and Medication

Last Dosage Change/Update: 5/26/2009 5:41:06 PM

Warfarin Type: Coumadin

Patient Pill Sizes Currently Available to Patient

1	2	2.5	3	4	5	6	7.5	10
☑	☑	☐	☐	☐	☐	☑	☐	☐

Update Pill Sizes and Medication >>>

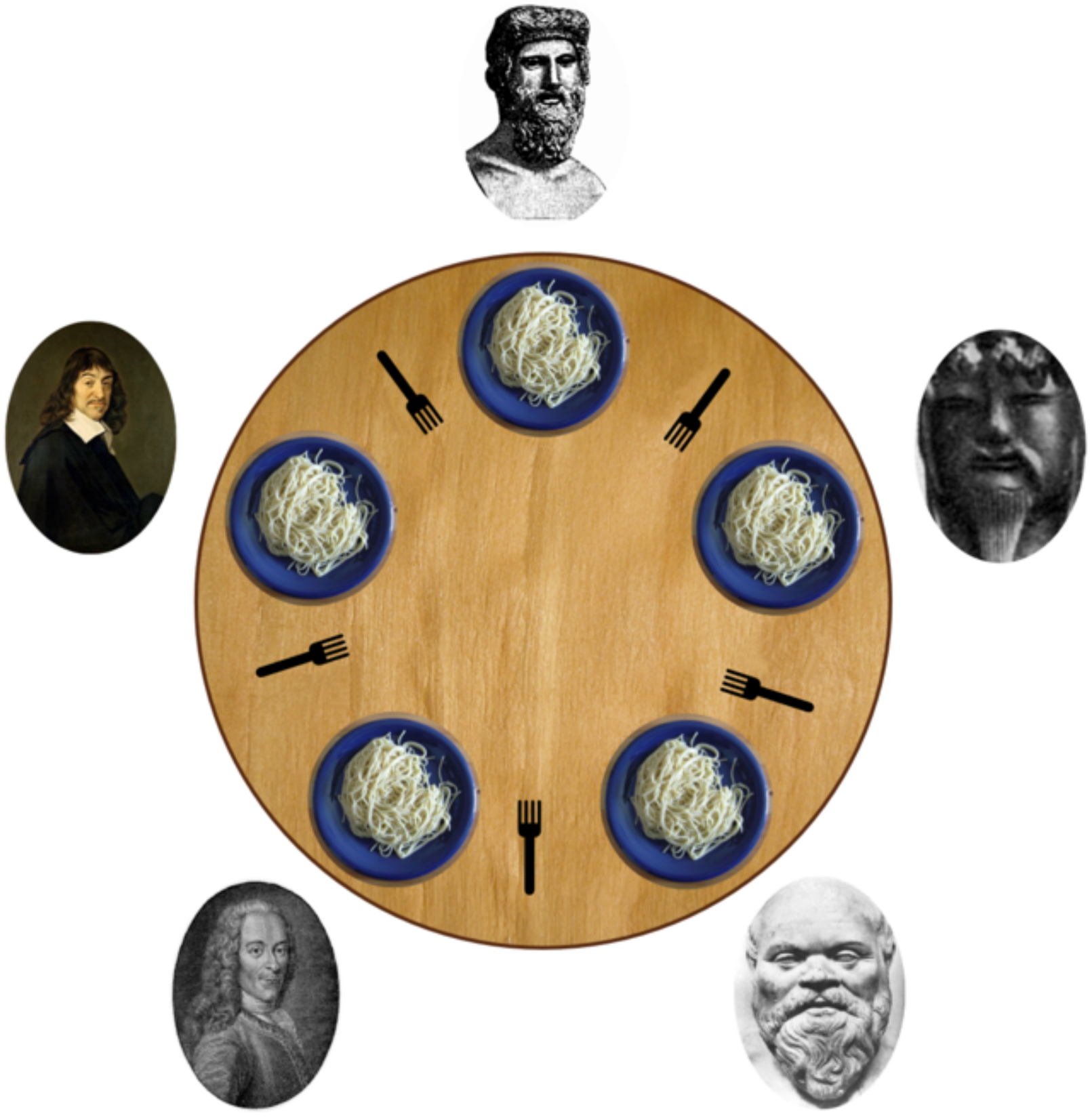
Change Patient Dosage by Day of Week

Day of Week	Current (mg)	New (mg)
Sunday	2	2 mg
Monday	3	3 mg
Tuesday	4	4 mg
Wednesday	2	2 mg
Thursday	4	4 mg
Friday	3	3 mg
Saturday	3	3 mg

Total mg amount per week = **21** mg

@ignasi35

fast FORWARD



@ignasi35

KIDS NATIONS

MINI FIGURINE

THE TRANSFORMERS MORE THAN MEETS THE EYE! BRAND

GRIMLOCK STARSCREAM SOUNDWAVE BUMBLEBEE OPTIMUS PRIME

kidslogic

Licensed by: Hasbro

@ignasi35

KEEP

CALM

AND

USE THE

FOLD


ALICE IN WONDERLAND
LEWIS CARROLL

VINTAGE HOME & GARDEN MARKET

Down the Rabbit Hole

May 4, 10am-4pm

412 West Hazard Road
Spokane WA

Please join us for a beautiful day in a charming garden setting. Featuring a select group of vendors with vintage home and garden goods, antiques and vintage finds. Enjoy a spot o' tea, tasty treats, and fabulous foods to enjoy there, or take home to share.

PRESENTED BY:

JUNEBUG
Furniture & Design

junebugfurnitureanddesign.blogspot.com

VINTAGE HOME & GARDEN MARKET

Down the Rabbit Hole

y 4, 10am-4pm

4 t Hazard Road

e VA

Please join us f
beautiful day in a
charming ga
setting. Featurin
select gro
vendo
den
ntiques and
age finds. Enjoy a
spot o' tea, tasty
treats, and fabulous
foods to enjoy there,
or take home to share.

PRESENTED BY:

JUNEBUG
FURNITURE & DESIGN

junebugfurnitureanddesign.blogspot.com

@ignasi35

XXIst Century DateTime

@ignasi35

XXIst Century DateTime

- Date is DEAD (my opinion)
- Calendar is DEAD (my opinion)
- DEAD is 57005 (that's a fact)

XXIst Century DateTime

- Stephen Colebourne et al.
- JodaTime
- JSR 310 — 7 years FTW

XXIst Century DateTime

- Clock
- LocalDate
- LocalDateTime
- Duration vs Period
- ZonedDateTime

- Enum.Month
- Enum.DayOfWeek

XXIst Century DateTime

Enum.Month

- Not just JAN, FEB, MAR
- Full arithmetic
  - plus(long months)
  - firstDayOfYear(boolean leapYear)
  - length(boolean leapYear)
  - …

XXIst Century DateTime

- Clock
  - replace your sys.currentTimeMillis
  - allows testing
  - Instant/now
- LocalDate
- LocalDateTime
- Duration vs Period
- ZonedDateTime


BBC one HD

XXIst Century DateTime

- Clock
- LocalDate
  - a date
  - no TimeZone
  - birth date, end of war, man on moon,…
- LocalDateTime
- Duration vs Period
- ZonedDateTime

XXIst Century DateTime

- Clock
- LocalDate
- LocalDateTime
  - an hour of the day
  - noon
  - 9am
- Duration vs Period
- ZonedDateTime

XXIst Century DateTime

- Clock
- LocalDate
- LocalDateTime
- Duration vs Period
  - Duration: 365*24*60*60*1000
  - Period: 1 year (not exactly 365 days)
  - Duration (Time) vs Period (Date)
- ZonedDateTime


Qatar
Foundation
10
21

XXIst Century DateTime

- Clock
- LocalDate
- LocalDateTime
- Duration vs Period
- ZonedDateTime (not an Instant!!)
  - Immutable
  - nanosecond detail
  - Normal, Gap, Overlap


THE BIG SIZE

KEEP

CALM

AND

USE THE

FOLD

KEEP
CALM
AND USE
THE
FORCE

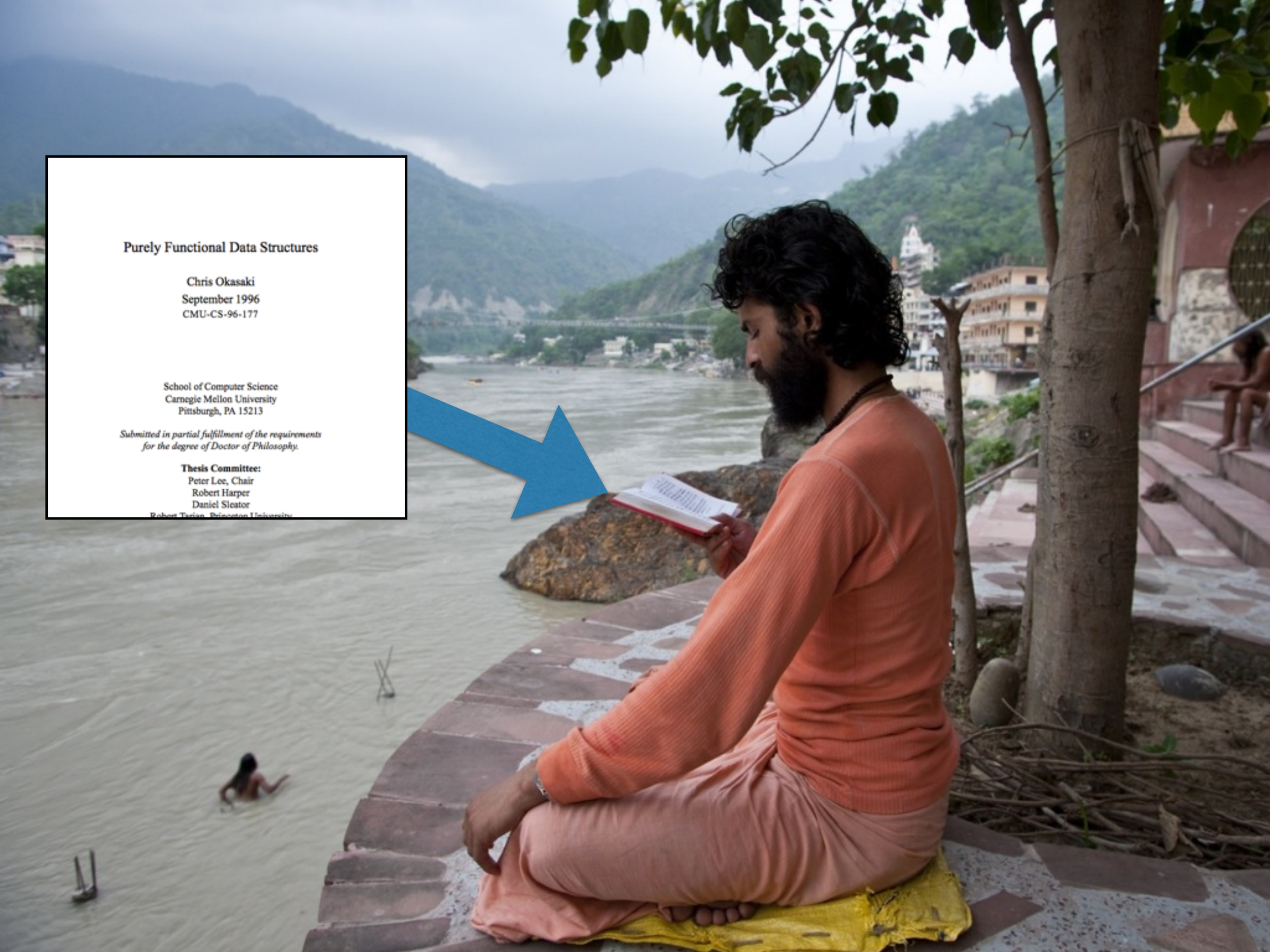
KEEP

CALM

AND

USE THE

FOLD



Purely Functional Data Structures

Chris Okasaki

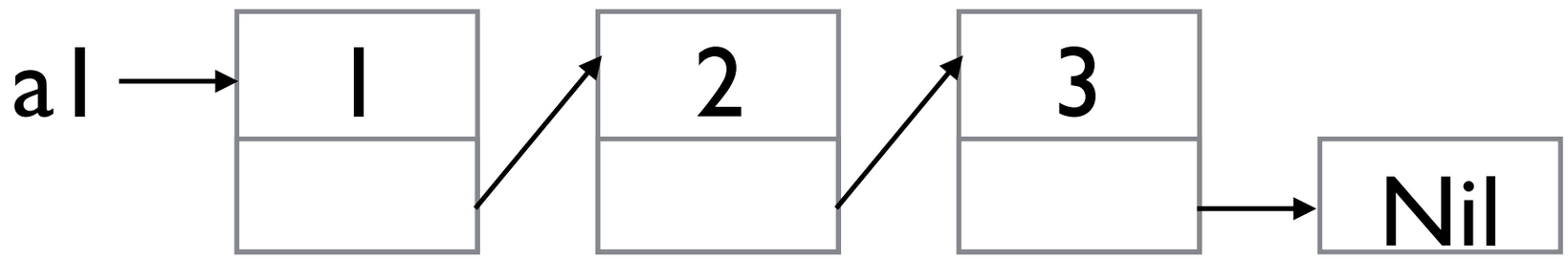
September 1996
CMU-CS-96-177

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

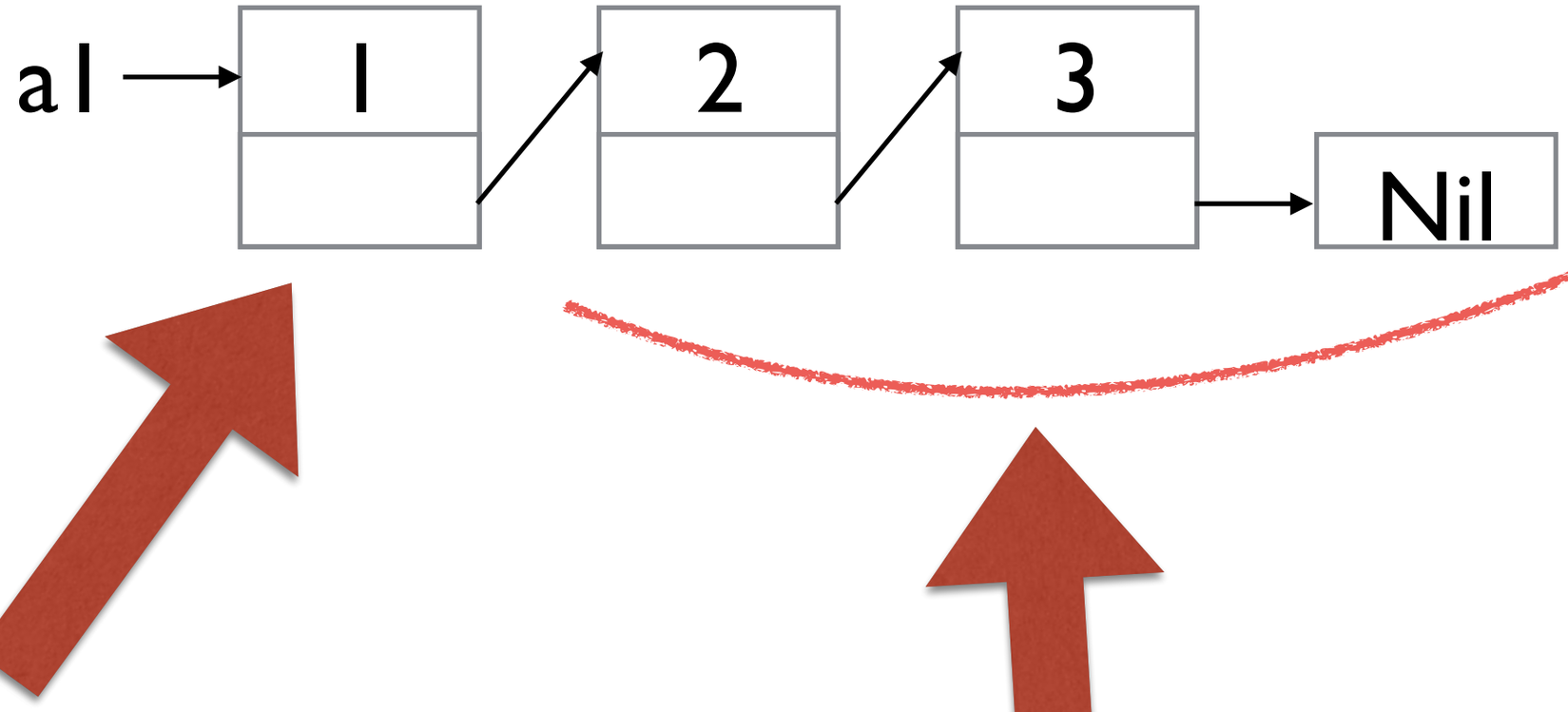
Thesis Committee:
Peter Lee, Chair
Robert Harper
Daniel Sleator
Robert Tarjan, Princeton University

Lists




```
List<Integer> nil = Lists.nil();

List<Integer> a3 = nil.prepend(3);
List<Integer> a2 = a3.prepend(2);
List<Integer> a1 = a2.prepend(1);
```

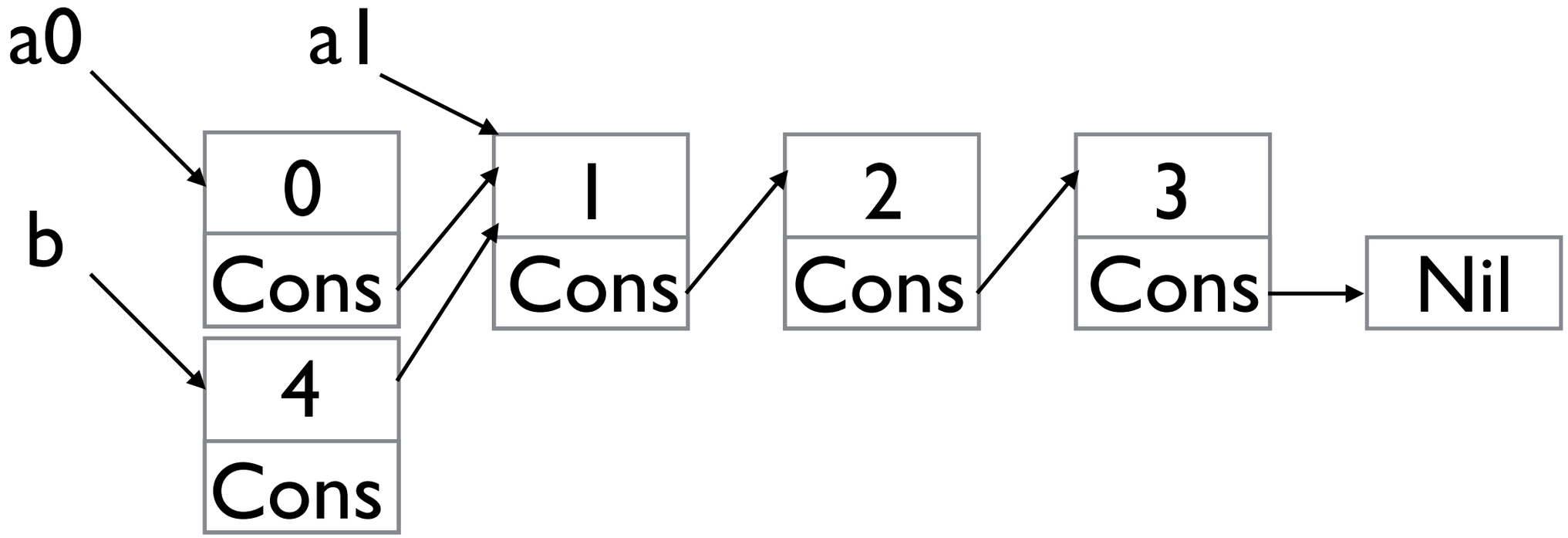
Lists



Lists

```
public interface List<T> {
    T head();
    List<T> tail();
    boolean isEmpty();
    void forEach(Consumer<? super T> f);
    default List<T> prepend(T t) {
        return new Cons<>(t, this);
    }
}
```

Lists




```
List<Integer> a0 = a1.prepend(0);
List<Integer> b = a1.prepend(4);
```

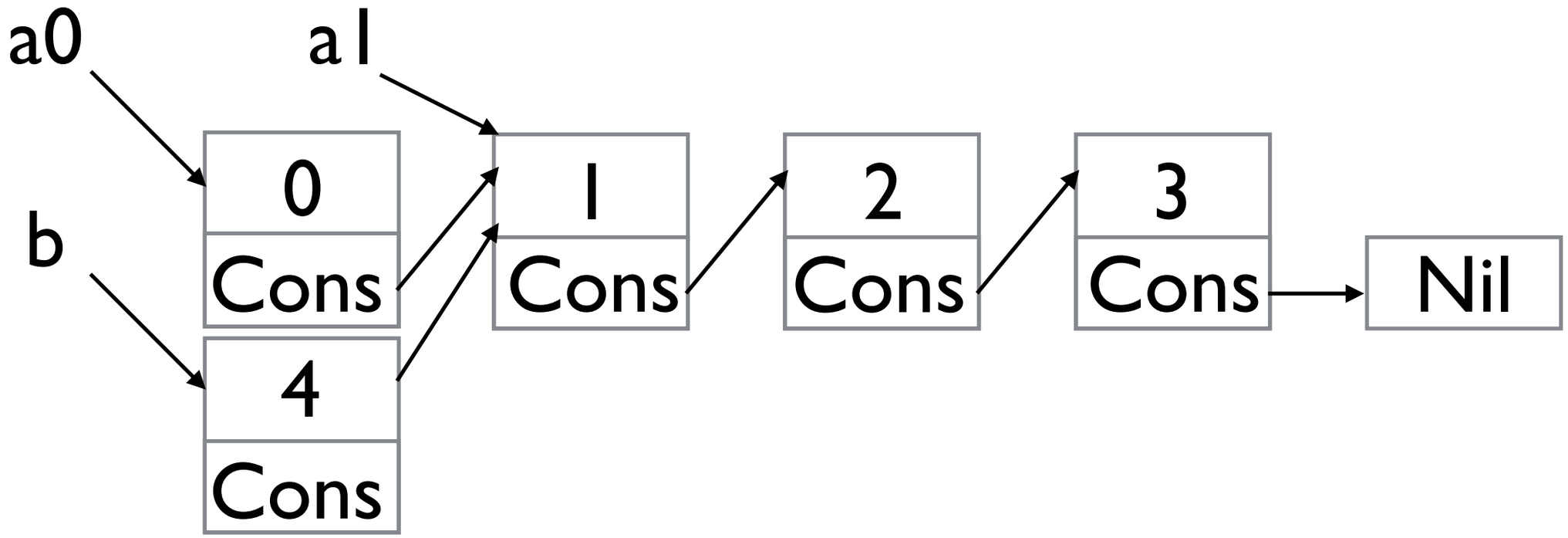
Lists

```
class Cons<T> implements List<T> {
    private T head;
    private List<T> tail;
    Const(T head, List<T> tail) {
        this.head = head;
        this.tail = tail;
    }

    T head(){return this.head;}
    List<T> tail(){return this.tail;}
    boolean isEmpty(){return false;}

    void forEach(Consumer<? super T> f){
        f.accept(head);
        tail.forEach(f);
    }
}
```

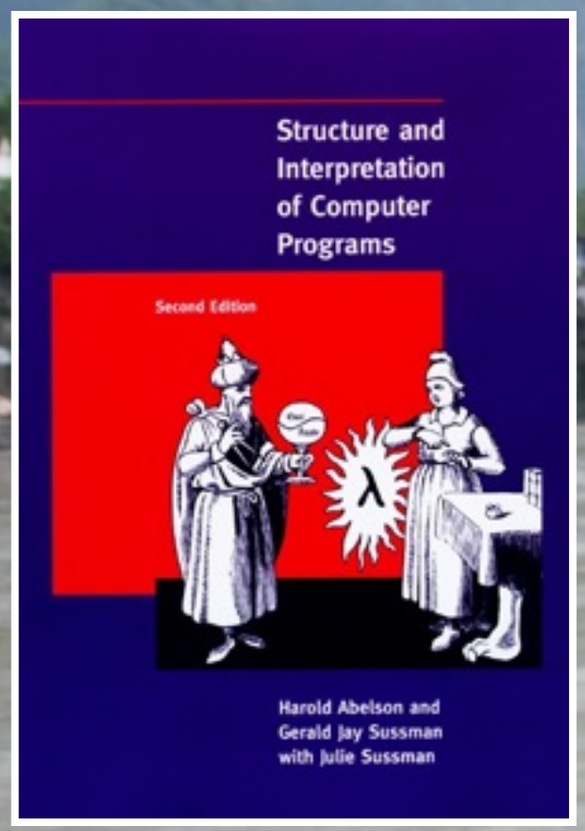
Lists

```
class Nil<T> implements List<T> {

    T head() {
        throw new  NoSuchElementException();
    }
    List<T> tail() {
        throw new NoSuchElementException();
    }
    boolean isEmpty() {
        return true;
    }
    void forEach(Consumer<? super T> f){
    }

}
```

Lists



Persistent Datastructures (not ephemeral, versioning)
Immutable
As efficient (consider amortised cost)

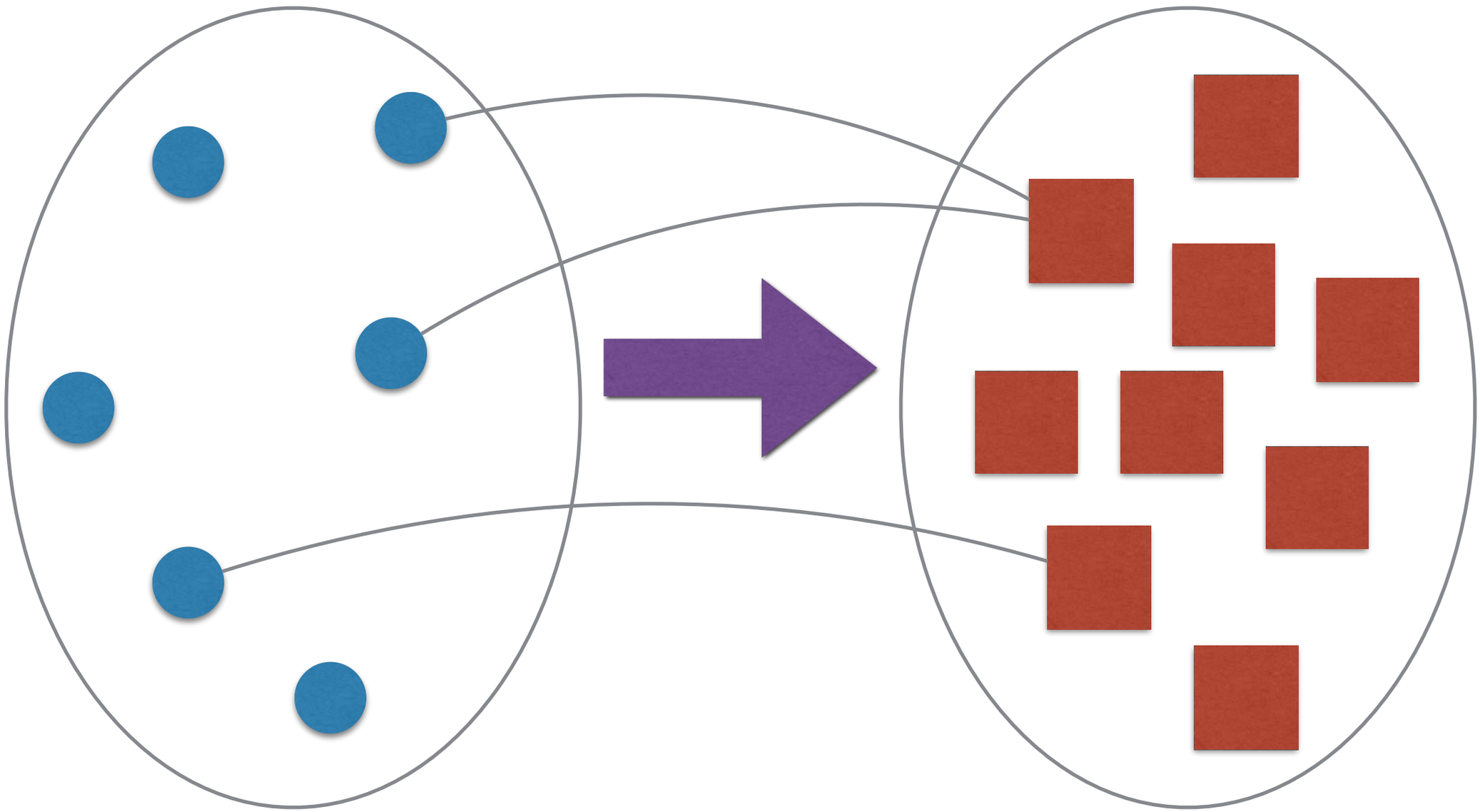

Structure and
Interpretation
of Computer
Programs
Second Edition
λ
Harold Abelson and
Gerald Jay Sussman
with Julie Sussman

filter

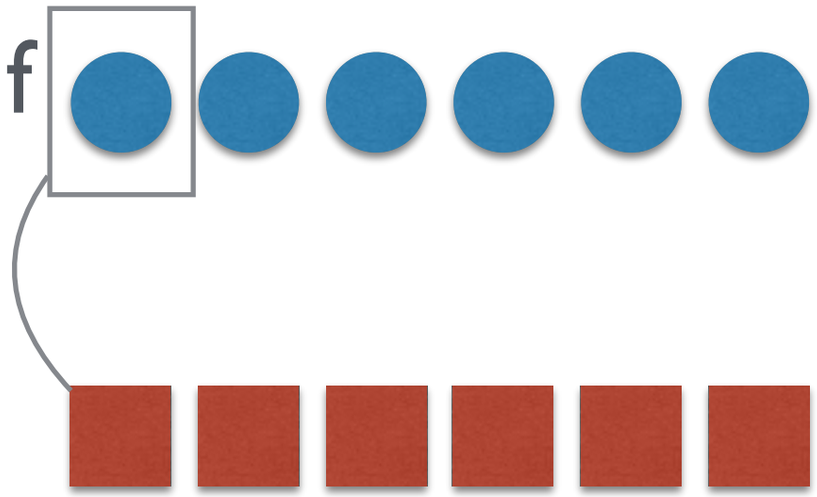
```
class Nil<T> implements List<T> {
  //…
  List<T> filter(Predicate<? super T> p) {
    return this;
  }
}
```

```
class Cons<T> implements List<T> {
  //…
  List<T> filter(Predicate<? super T> p) {
   if (p.test(head))
     return new Const<>(head, tail.filter(p));
   else
     return tail.filter(p);
  }
}
```

map



map



map

```
class Nil<T> implements List<T> {
  //…
  <R> List<R> map(Function<T, R> f) {
    return (List<R>) this;
  }
}

class Cons<T> implements List<T> {
  //…
  <R> List<R> map(Function<T, R> f) {
   return new Const<>(f.apply(head), tail.map(f));
  }
}
```

KEEP

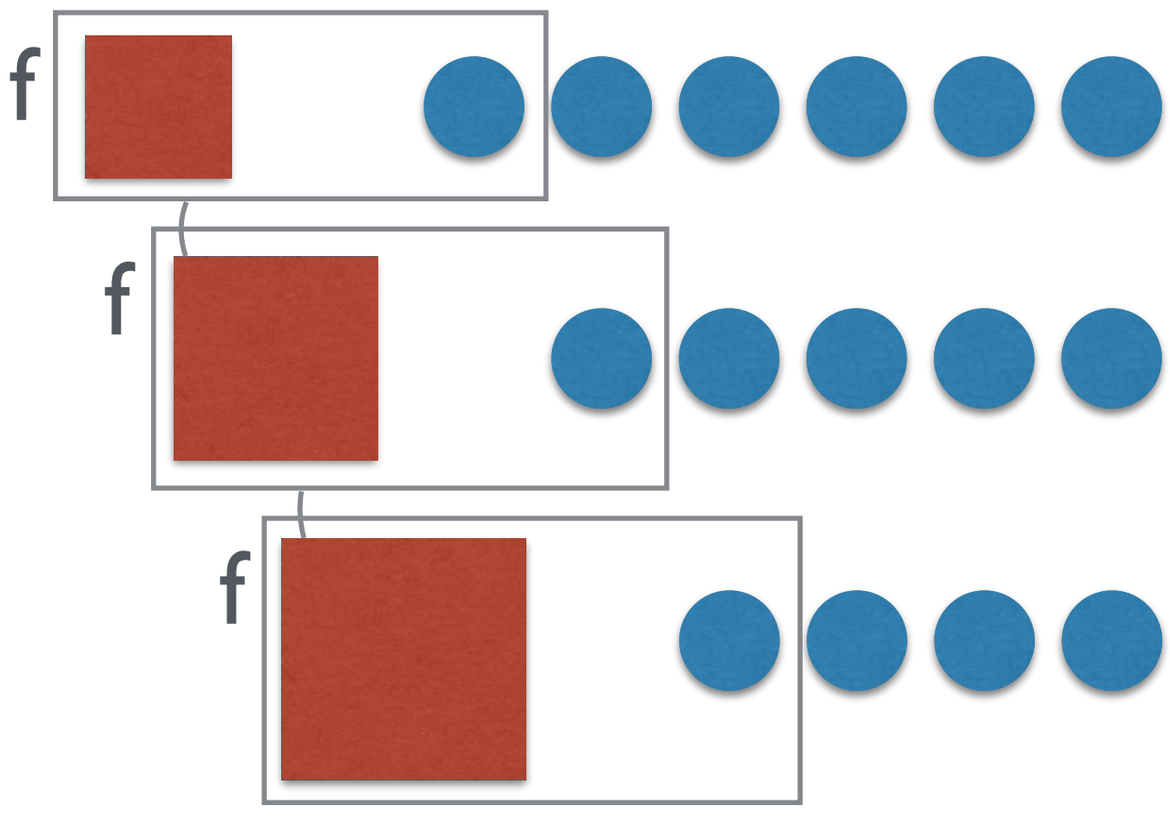
CALM

AND

USE THE

FOLD

fold



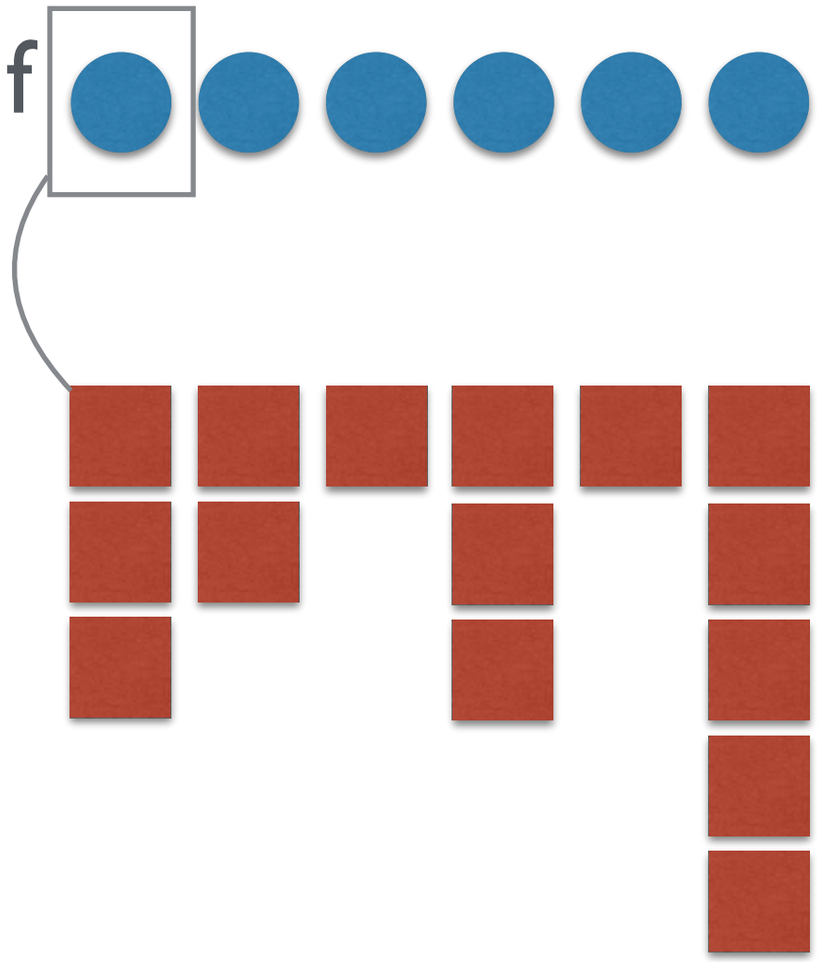
fold

```
class Nil<T> implements List<T> {
 <Z> Z reduce(Z z, BiFunction<Z, T, Z> f) {
     return z;
   }
}

class Cons<T> implements List<T> {
  <Z> Z reduce(Z z, BiFunction<Z, T, Z> f) {
    return tail.reduce(f.apply(z,head), f);
  }
}
```

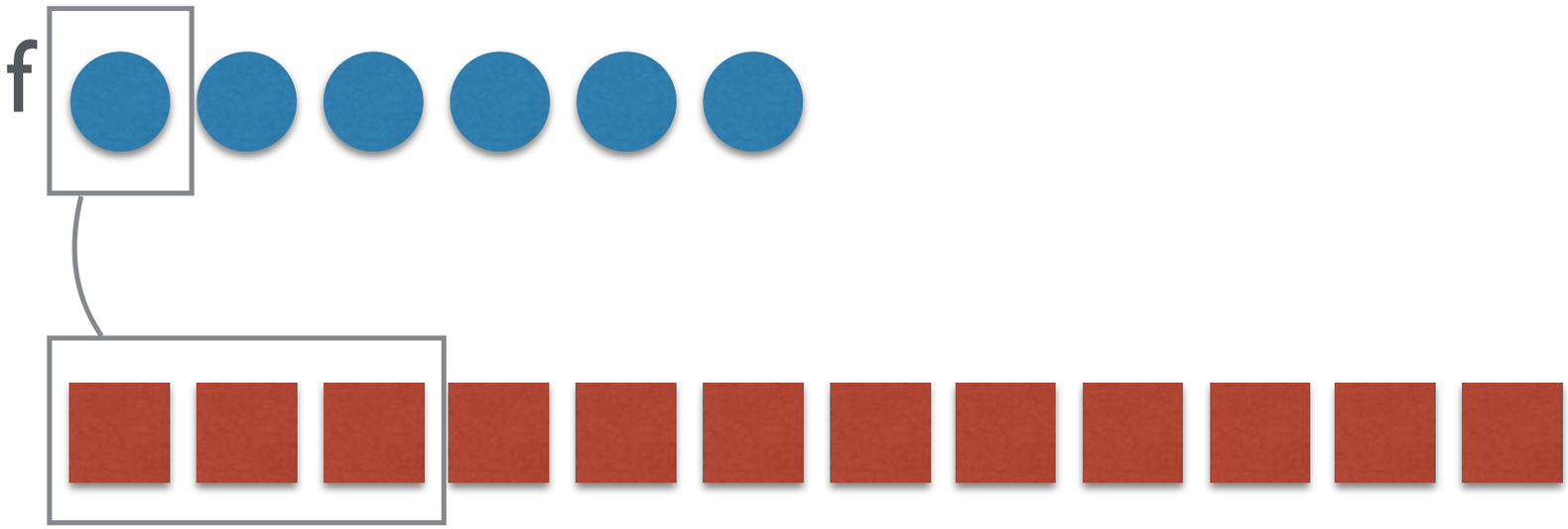
fold

aka reduce

map

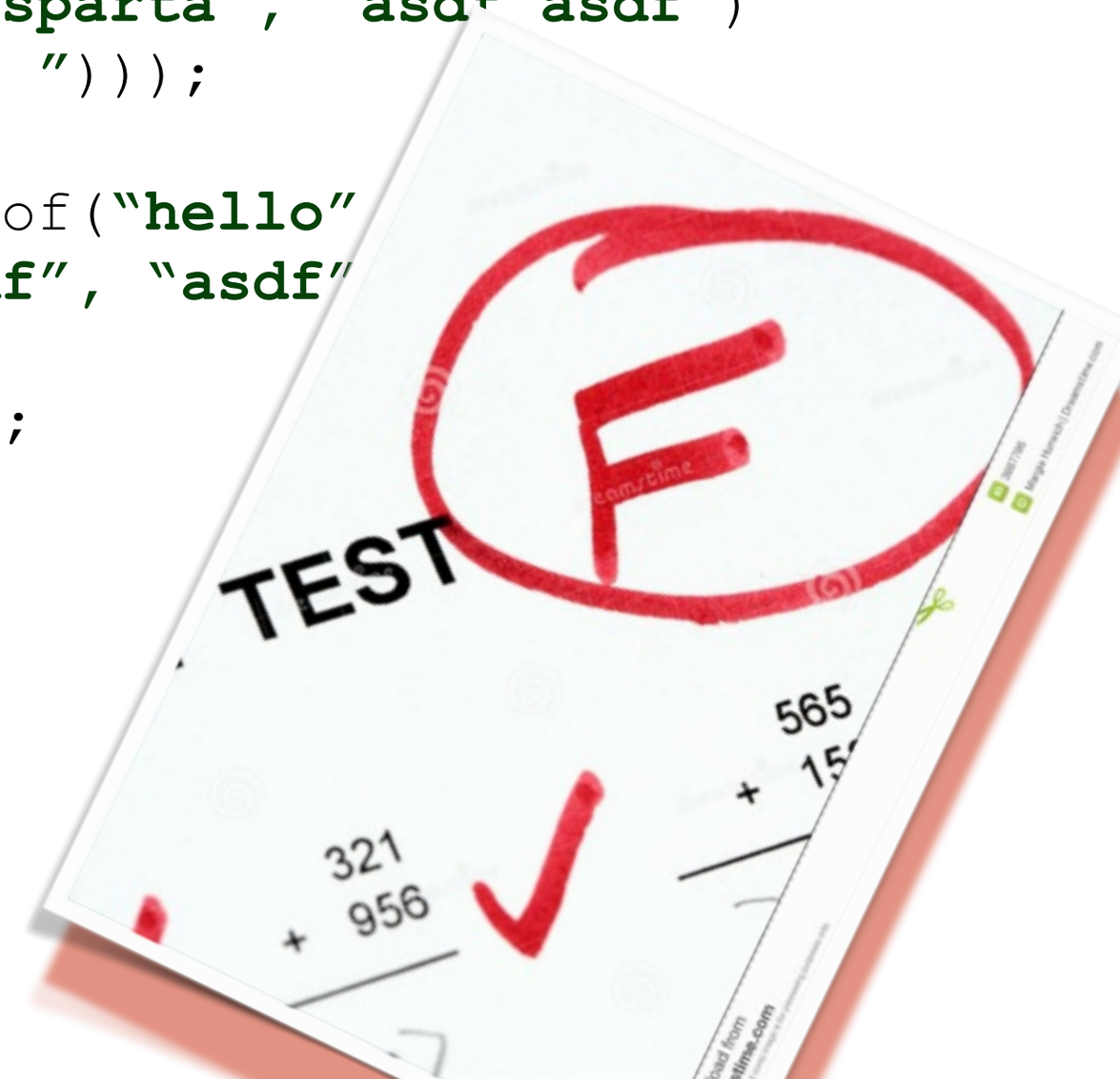


map

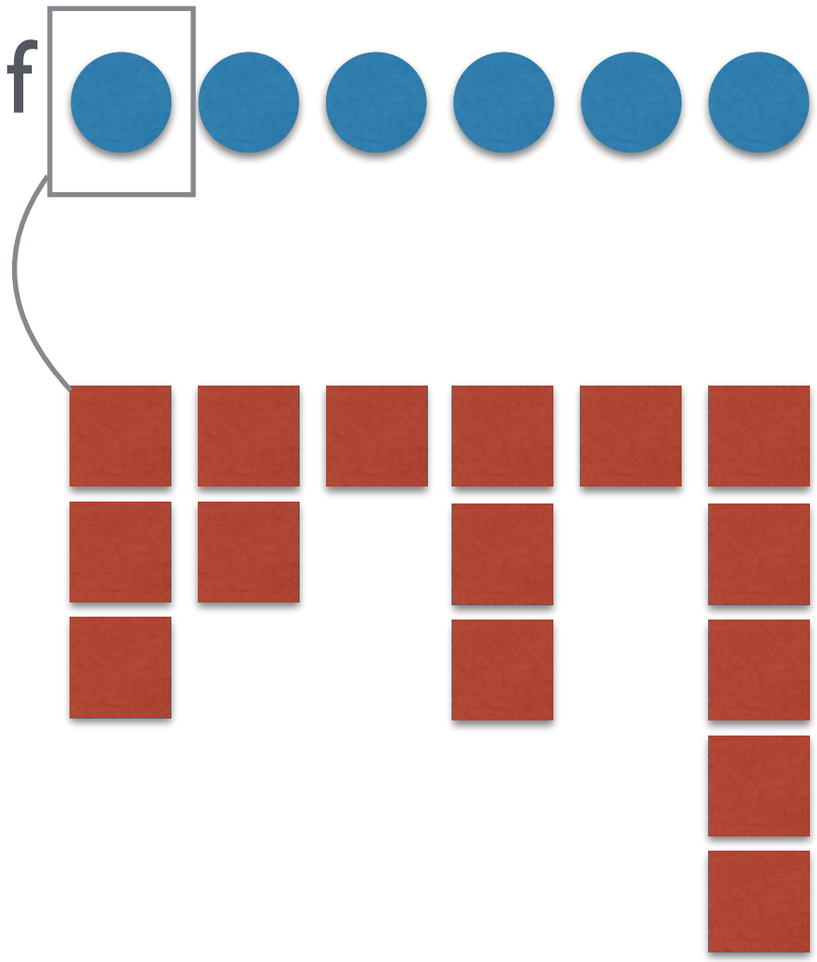


map

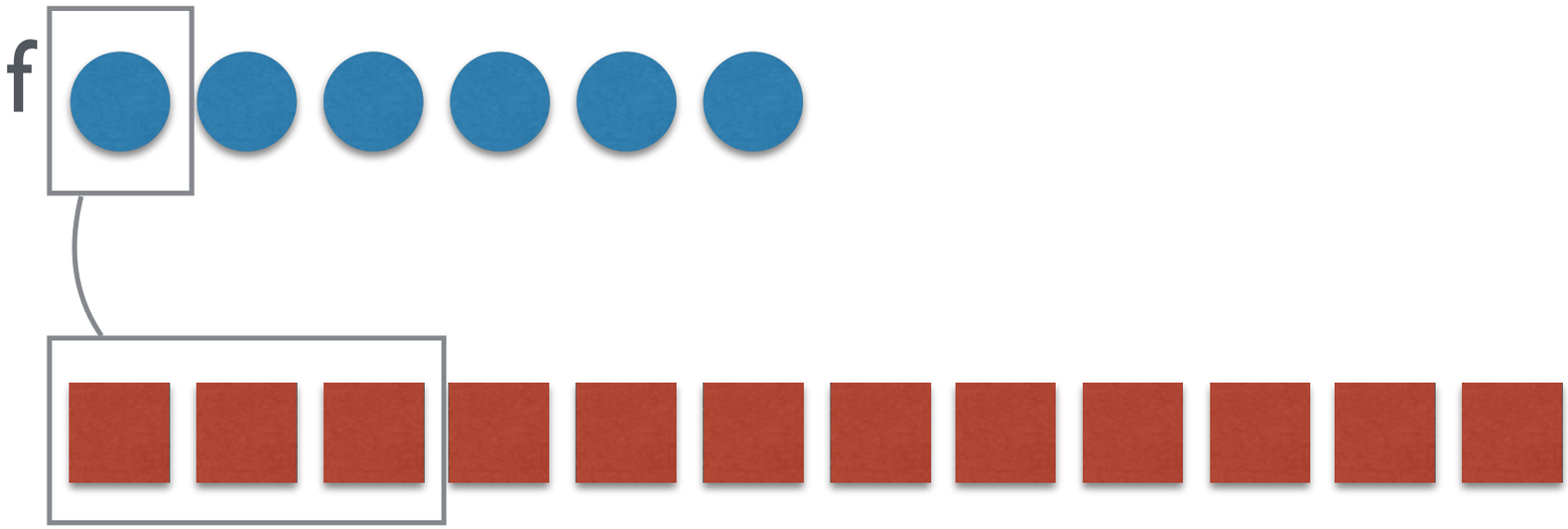
```
@Test
void testMapList() {
  List<List<String>> actual = Lists
    .of("hello world", "This is sparta", "asdf asdf")
    .map(s -> Lists.of(s.split(" ")));

  List<String> expected = Lists.of("hello"
    "This", "is", "sparta", "asdf", "asdf"

  assertEquals(expected, actual);
}
```



map



map




When your life is in danger...
... flatMap that shit.

flatMap

```
class Cons<T> implements List<T> {
  //…
  <R> List<R> map(Function<T, R> f) {
   return new Const<>(f.apply(head), tail.map(f));
  }

  <R> List<R> map(Function<T, List<R> f) {
   return f.apply(head).append(tail.flatMap(f));
  }

}
```

recap

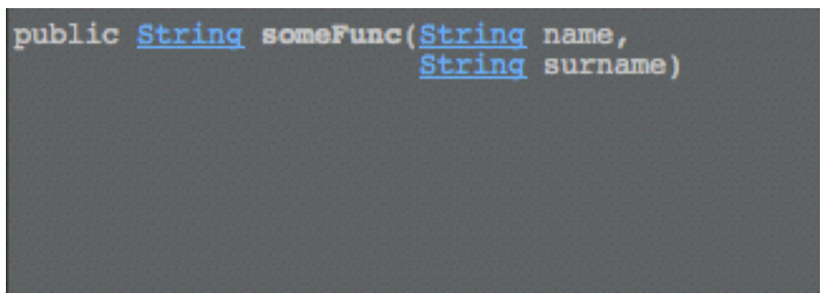
filter

map

fold

flatMap

```
class MyFoo {

 //@param surname may be null
 String someFunc(String name, String surname) {
     …
 }

}
```



Maybe (aka Optional)

replaces null completely

Maybe (aka Optional)

replaces null completely

forever

Maybe (aka Optional)

replaces null completely

forever

and ever

Maybe (aka Optional)

replaces null completely

forever

and ever

and ever

Maybe (aka Optional)

replaces null completely

forever

and ever

and ever

and ever

Maybe (aka Optional)

replaces null completely

forever

and ever

and ever

and ever

and ever

Maybe (aka Optional)

replaces null completely

forever

and ever

and ever

and ever

and ever

and ever

Maybe (aka Optional)

```
class MyFoo {

 String someFunc(String name, Optional<String> surname) {
        ...
    }

    ...

}
```

Maybe (aka Optional)

```
class MyFoo {

 Optional<Person> someFunc(Name x, Optional<Surname> y) {
        ...
    }

    ...

}
```

Maybe (aka Optional)

Some/Just/Algo

ADT

None/Nothing/Nada

KEEP

CALM

AND

USE THE

FOLD

Maybe (aka Optional)

filter: applies predicate and Returns input or None
map: converts content
fold: returns Some(content) or Some(default)
flatMap: see list

get: returns content or throws Exception
getOrElse: returns content or defaultValue

recap

filter

map

fold

flatMap

ADT

Functor

Future (aka CompletableFuture)

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class **Next Class** Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

compact1, compact2, compact3

java.util.concurrent

Class CompletableFuture<T>

java.lang.Object
    java.util.concurrent.CompletableFuture<T>

All Implemented Interfaces:

Future<T>

```
public class CompletableFuture<T>
extends Object
implements Future<T>
```

A `Future` that may be explicitly completed (setting its value and status), and may include dependent functions and actions that trigge

When two or more threads attempt to `complete`, `completeExceptionally`, or `cancel` a CompletableFuture, only one of them su

Methods are available for adding dependents based on user-provided Functions, Consumers, or Runnables. The appropriate form to
dependent action will trigger the completion of another CompletableFuture. Actions may also be triggered after either or both the curr

Future (aka CF, aka CompletableFuture)

[FAIL] Does not use map, flatMap, filter.

[PASS] CF implemented ADT

[FAIL] Because Supplier, Consumer, Function, Bifuction, … CF's API sucks.

recap

filter

map

fold

flatMap

ADT

Functor

recap

Maybe simulates nullable
Future will eventually happen

Exceptions still fuck up your day

KEEP

CALM

AND

USE THE

FOLD

KEEP
CALM
AND USE
THE
FORCE

Try

Simulates a computation that:

succeeded
or
threw exception

compact1, compact2, compact3

java.util

Class OptionalInt

java.lang.Object
    java.util.OptionalInt

```
public final class OptionalInt
extends Object
```

A container object which may or may not contain a `int` value. If a value is present, `isPresent()` will return `true` and `getAsInt()` will retu

Additional methods that depend on the presence or absence of a contained value are provided, such as `orElse()` (return a default value i
(execute a block of code if the value is present).

This is a value-based class; use of identity-sensitive operations (including reference equality (==), identity hash code, or synchronization)
unpredictable results and should be avoided.

Since:

1.8

Method Summary

All Methods	Static Methods	Instance Methods	Concrete Methods

Modifier and Type	Method and Description
static OptionalInt	empty() Returns an empty OptionalInt instance.

Java™ Platform, Standard Edition 8
API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an a communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and with applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphic Interface systems that provides a mechanism to transfer information b

compact1, compact2, compact3

java.nio.file

Class Files

java.lang.Object
    java.nio.file.Files

```
public final class Files
extends Object
```

This class consists exclusively of static methods that operate on files, directories, or other types of files.

In most cases, the methods defined here will delegate to the associated file system provider to perform the file operations.

Since:

1.7

Method Summary

All Methods	Static Methods	Concrete Methods

Modifier and Type	Method and Description
static long	copy(InputStream in, Path target, CopyOption... options) Copies all bytes from an input stream to a file.

compact1, compact2, compact3

java.util.stream

Interface Stream<T>

Type Parameters:

`T` - the type of the stream elements

All Superinterfaces:

`AutoCloseable`, `BaseStream<T,Stream<T>>`

```
public interface Stream<T>
extends BaseStream<T,Stream<T>>
```

A sequence of elements supporting sequential and parallel aggregate operations. The following example illustrates an aggregate operation using `Stream` and `IntStream`:

```
int sum = widgets.stream()
                 .filter(w -> w.getColor() == RED)
                 .mapToInt(w -> w.getWeight())
                 .sum();
```

In this example, `widgets` is a `Collection<Widget>`. We create a stream of `Widget` objects via `Collection.stream()`, filter it to produce a stream containing only the red widgets, and then transform it into a stream of `int` values representing the weight of each red widget. Then this stream is summed to produce a total weight.

In addition to `Stream`, which is a stream of object references, there are primitive specializations for `IntStream`, `LongStream`, and `DoubleStream`, all of which are referred to as "streams" and conform to the characteristics and restrictions described here.

To perform a computation, stream operations are composed into a *stream pipeline*. A stream pipeline consists of a source (which might be an array, a collection, a generator function, an I/O channel, etc), zero or more *intermediate operations* (which transform a stream into another stream, such as `filter(Predicate)`), and a *terminal operation* (which produces a result or side-effect, such as `count()` or `forEach(Consumer)`). Streams are lazy; computation on the source data is only performed when the terminal

DO
OR
DO NOT
THERE IS
NO
TRY

JEDI MASTER YODA

Try in action

```
for (String s : stringList) {
  System.out.println("    --->    " + ((50 / Integer.valueOf(s)) - 1));
}
```

Try in action

```
public static void withoutTryB(java.util.List<String> stringList) {
  for (String s : stringList) {
    try {
      int parsed = Integer.valueOf(s);
      try {
        int div = 50 / parsed;
        int x = div - 1;
        System.out.println("    --->    " + x);
      } catch (ArithmeticException ae) {
        // What do we do ?!
      }
    } catch (NumberFormatException nfe) {
      // What do we do ?!
    }
  }
}
```

Try in action

```
public static void withTry(List<String> stringList) {
  final Function<String, Try<Integer>> toInt =
    (input) -> Tries.to(() -> Integer.valueOf(input));
  final Function<Integer, Try<Integer>> divide50 =
    (input) -> Tries.to(() -> 50 / input);

  // unused
  final Function<Integer, Integer> minusOne = x -> x - 1;

  // pipe
  final Function<String, Try<Integer>> pipe =
    (input) -> toInt.apply(input).flatMap(divide50).map(x -> x - 1);

  // execution
  final List<Try<Integer>> tryList = stringList.map(pipe);

  tryList.forEach(x -> System.out.println("    ---> " + x));
}
```

```java
class Failure<T> implements Try<T> {

  private RuntimeException t;
  public Failure(Throwable t) { this.t = new RuntimeException(t); }

  public <R> Try<R> map(Function<T, R> f) { return (Try<R>) this; }
  public <R> Try<R> flatMap(Function<T, Try<R>> f) { return (Try<R>) this; }
  public T get() { throw t; }
  public boolean isSuccess() { return false; }
  public String toString() { return "Failure{" + "t=" + t + '}'; }
}

class Success<T> implements Try<T> {

  private T s;
  Success(T s) { this.s = s; }

  public <R> Try<R> map(Function<T, R> f) { return new Success<>(f.apply(s)); }
  public <R> Try<R> flatMap(Function<T, Try<R>> f) { return f.apply(s); }
  public T get() { return s; }
  public boolean isSuccess() { return true; }
  public String toString() { return "Success{" + "s=" + s + '}'; }
}
```

Conclusions

```
class Repository {

    Try<Person> loadBy(Name x, Optional<Surname> y) {
        …
    }

    …

}
```

Moar resources

https://github.com/rocketscience-projects/javaslang

by https://twitter.com/danieldietrich

https://github.com/functionaljava/functionaljava

by http://www.functionaljava.org/ (runs in Java7)

KEEP

CALM

AND

USE THE

FOLD


When your life is in danger...
... flatMap that shit.

Namaste

@ignasi35

Questions

@ignasi35

End of presentation