Validating Method Input

Andrejs Doronins
TEST AUTOMATION ENGINEER

```
if(input != null){  // code     }
```

Class Invariants

Guard Clauses:
Fail Fast
&
Return Early

Best practices and pitfalls when checking null, strings, numbers and dates

Preventing propagation of invalid values

Demo

Project intro:

- Our (fictional) company
- UML
- Classes

Desktop

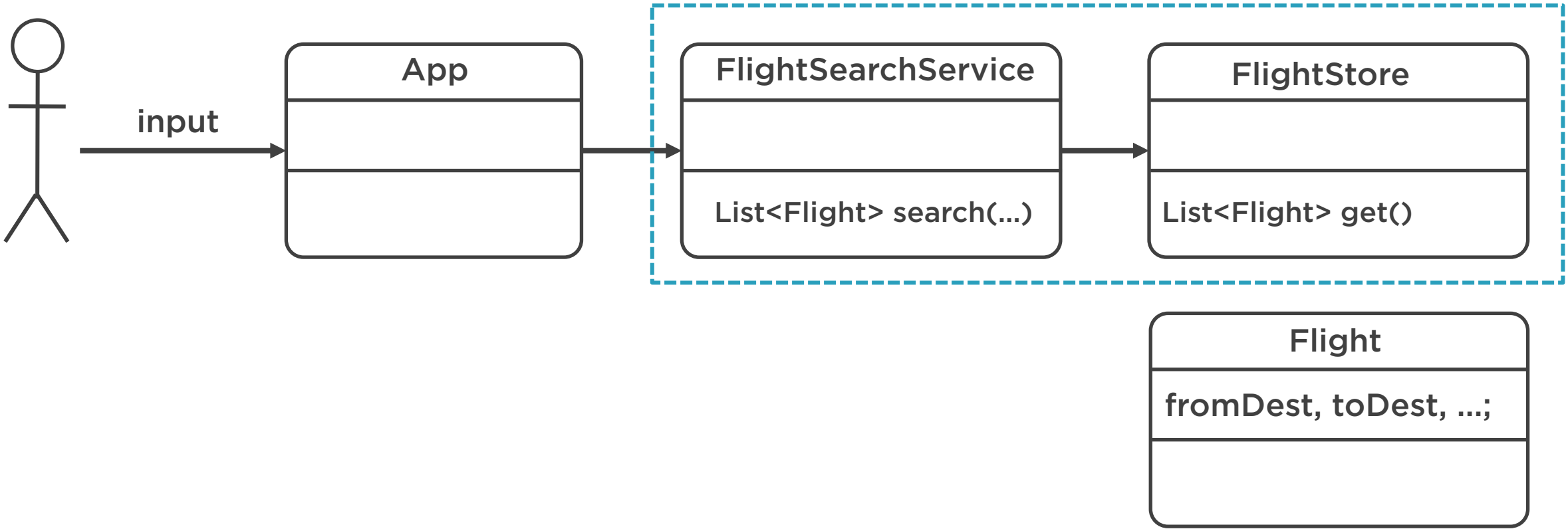
A lot of unrelated code

Web app

Complex Web Server setup

CLI

No dependencies


input
App
FlightSearchService
List<Flight> search(...)
FlightStore
List<Flight> get()
Flight
fromDest, toDest, ...;

Lambdas

Given	Then	Check	Result
x	->	x<4	

Given	Then	Check	Result
s	->	s.equals(<some string>)	

Java 8 Streams and Lambdas

PS Course: What's New in Java 8

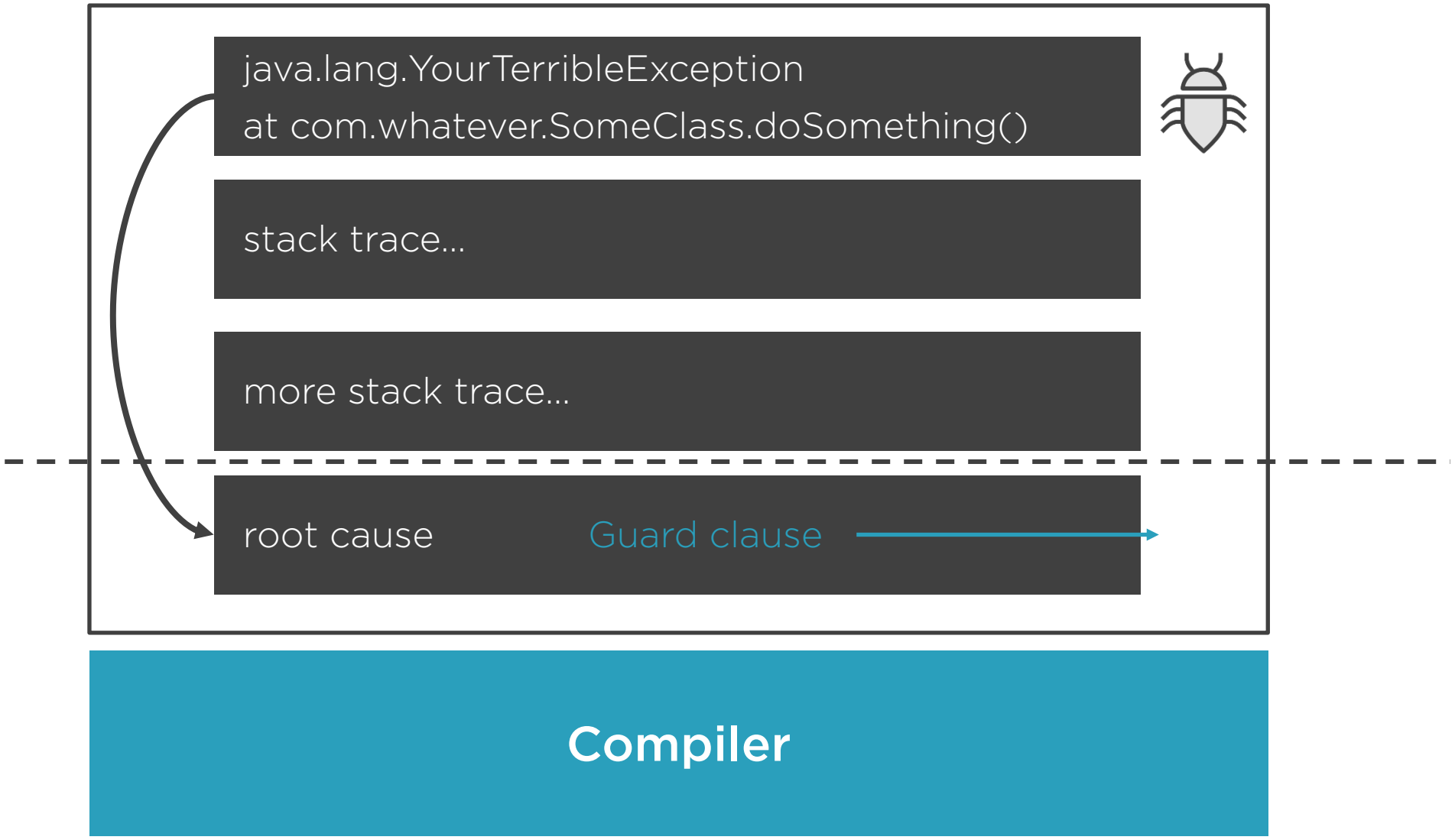
Book: Java 8 in Action

When should we spot an issue?

"Errors should be detected as soon as possible [...] ideally at compile time"

Joshua Bloch, Effective Java

Runtime


java.lang.YourTerribleException
at com.whatever.SomeClass.doSomething()
stack trace...
more stack trace...
root cause
Guard clause
Compiler

Guard Clause

```
if(/* validate input */){



}
```

Return Early

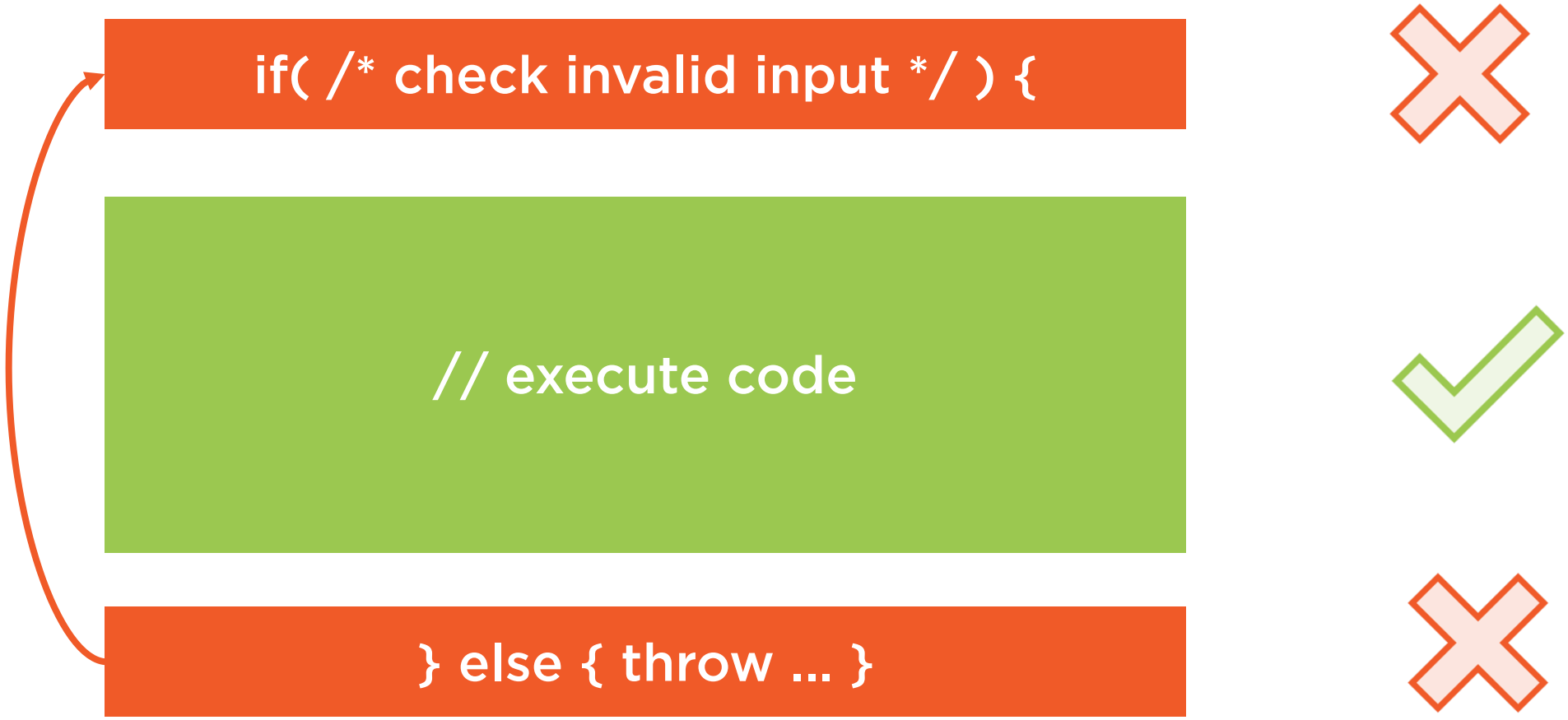
1) return false;
2) return;

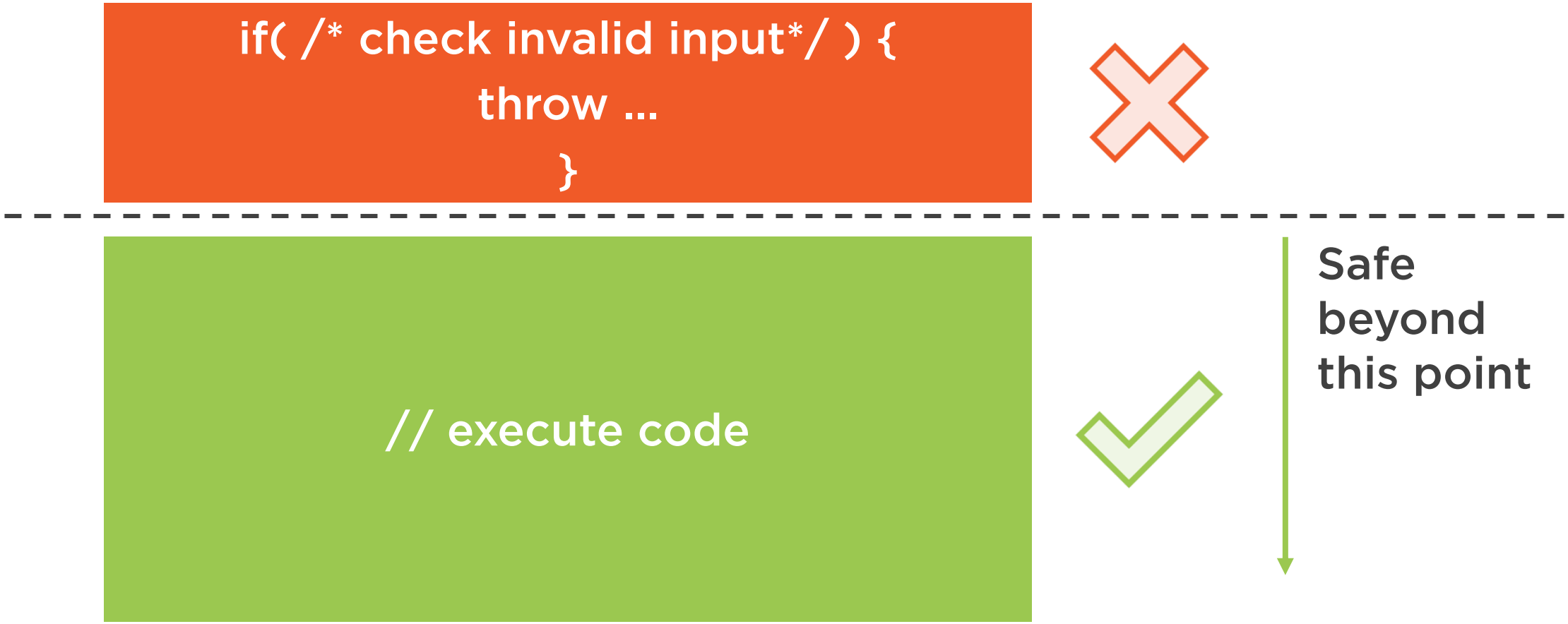
Fail Fast

throw new AppropriateException();

Alternative Execution

e.g. Display a user-friendly message of what went wrong


if(/* check invalid input */) {
// execute code
} else { throw ... }


if(/* check invalid input*/) {
throw ...
}
// execute code
Safe
beyond
this point


Specific but lengthy
Generic but short

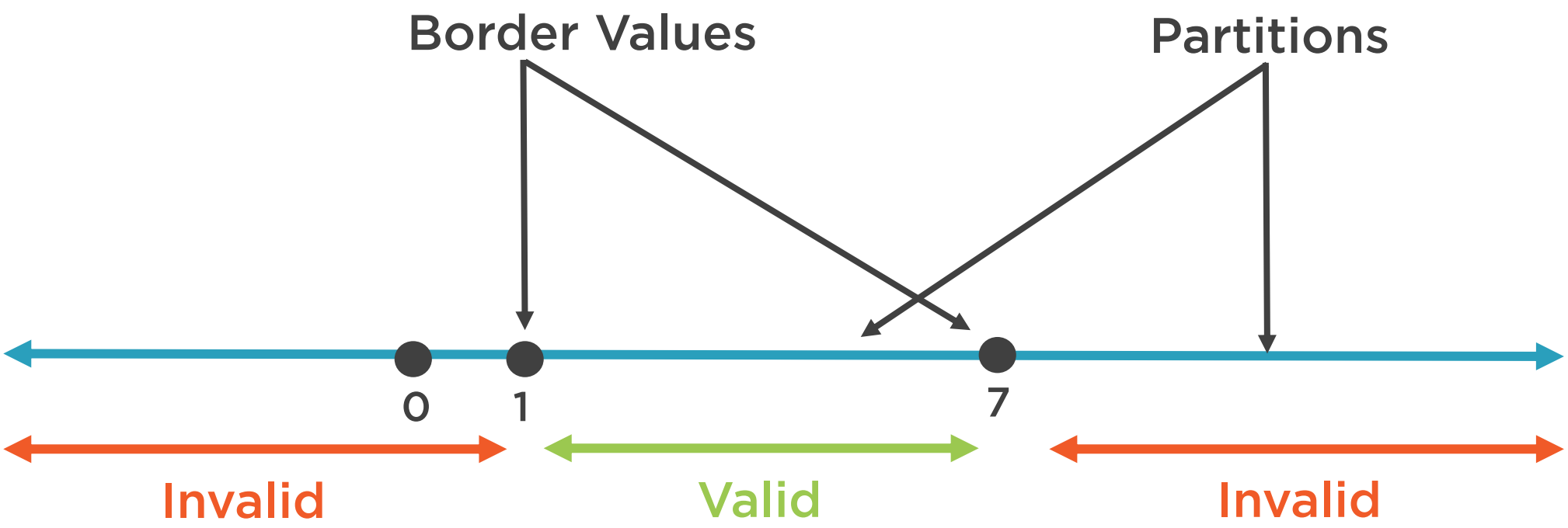
Place your Guard Clauses at the very beginning

NullPointerException
vs.
IllegalArgumentException

```
if() else { if() else {}}
```

```
if( ok ) { code } else { throw }
```

```
if (not ok) { throw } // rest of code
```


Border Values
Partitions
0
1
7
Invalid
Valid
Invalid

```
if(passengerNum < 1 && passengerNum > 7)
```

```
if(passengerNum < 0 && passengerNum > 7)
```

```
if(passengerNum < 0 && passengerNum > 8)
```

```
if(passengerNum < 1  ||  passengerNum > 7)
```

```
if(passengerNum <= 0 ||  passengerNum >= 8)
```

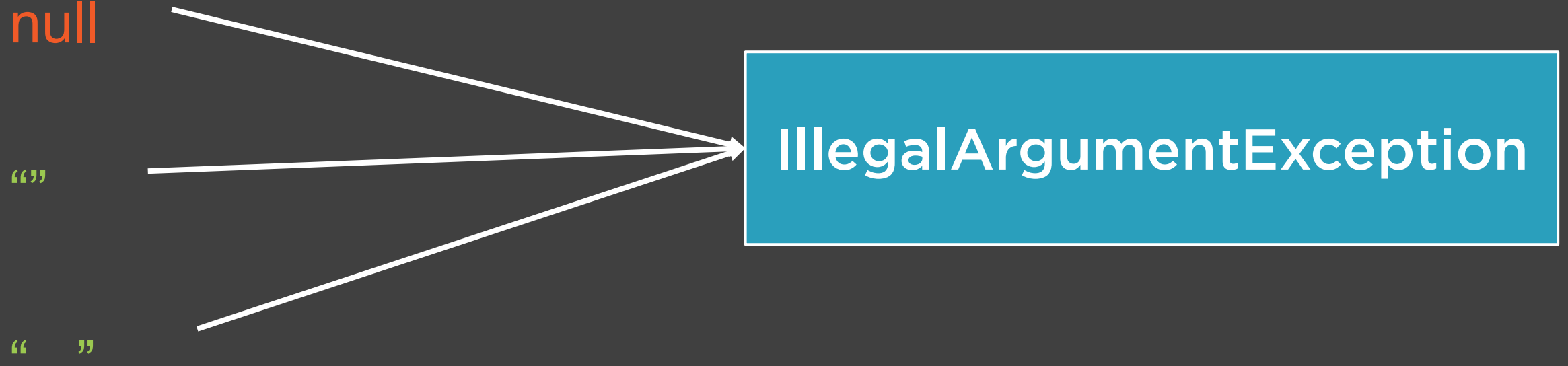
Double-check it!

Double-check it!


```
if( check 1  ||  check 2)
```

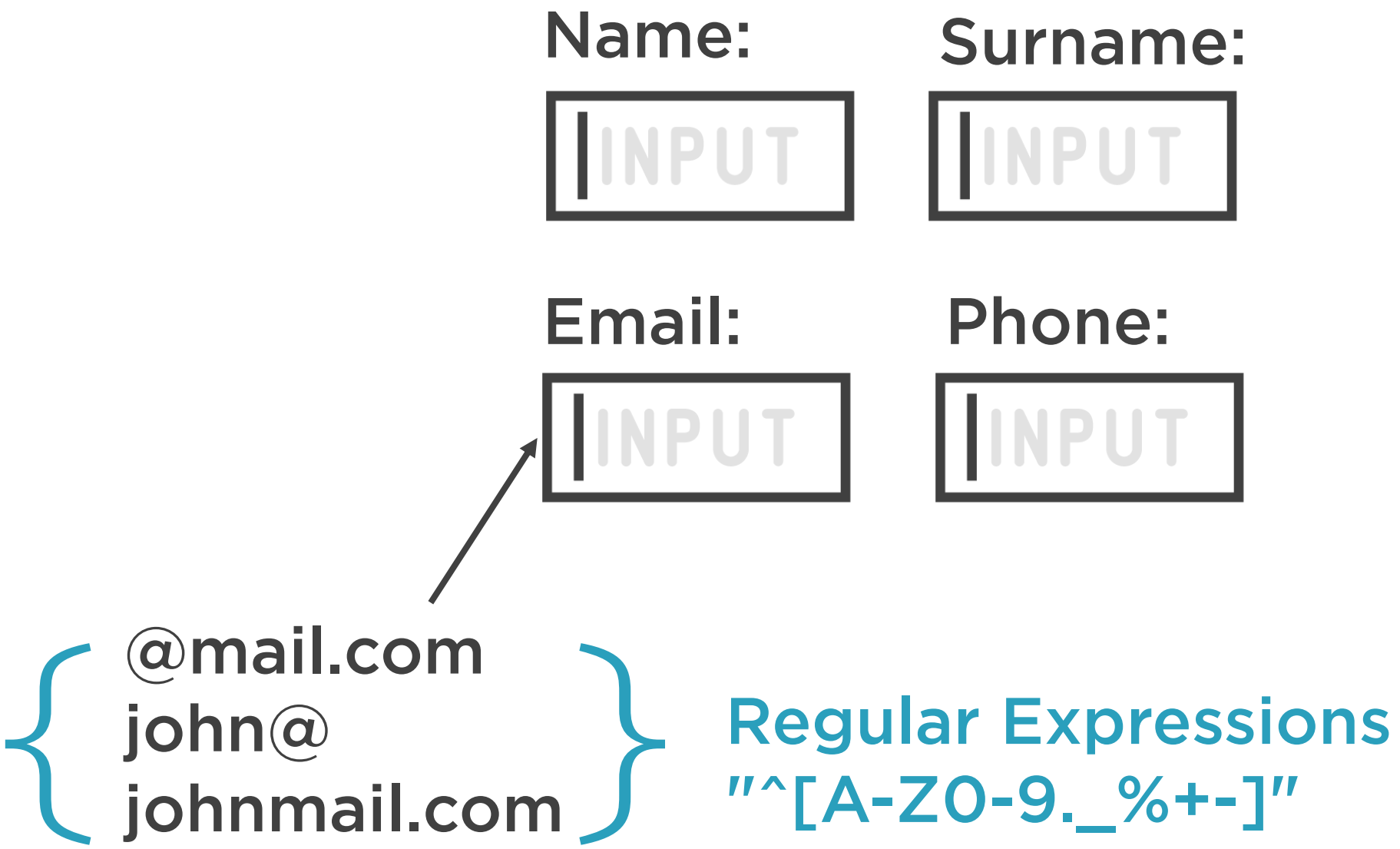
Watch out!

Invalid Strings



Decompose Conditional

(a form of Extract method)


Name:
INPUT
Surname:
INPUT
Email:
INPUT
Phone:
INPUT
@mail.com
john@
johnmail.com
Regular Expressions
"^[A-Z0-9._%+-]"

Regular Expressions are complex

"If you have a problem and the solution is a regex, now you have two problems."

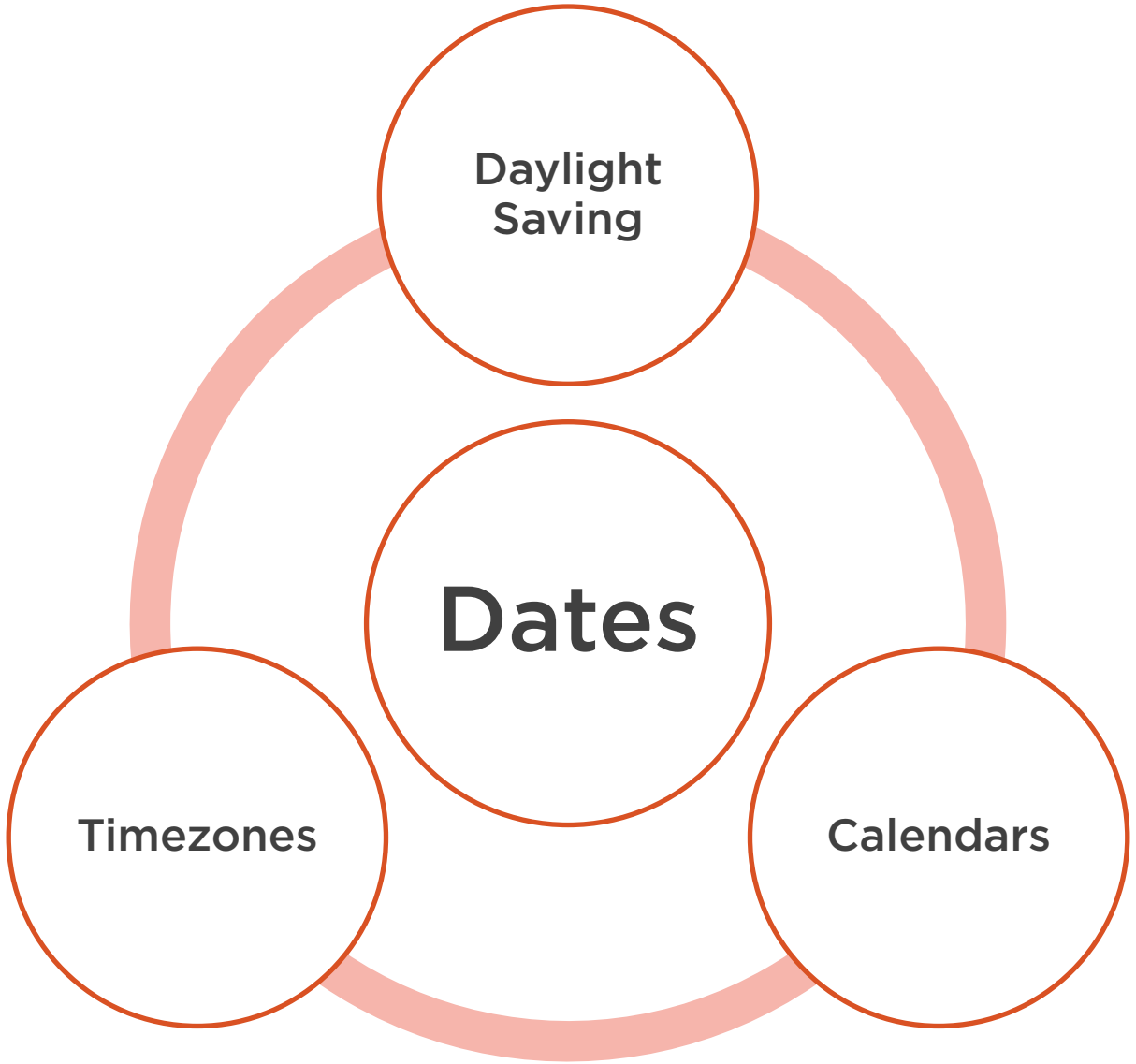
Internet wisdom

String Validation

Basic:

- Null
- Empty “”
- Only spaces or tabs “ ”

- Complex
  - Regular Expressions "^[A-Z0-9._%+-]"


Daylight Saving
Dates
Timezones
Calendars

Defensive Coding with Dates

DONTs	DOs
Don’t use the old java.util.Date API	Use java.time since Java 8: • LocalDate • LocalTime • Instant • etc.
Don’t store dates as Strings	Store dates as (Java 8) date objects
Avoid using Regex to validate String Dates	Use native .parse()

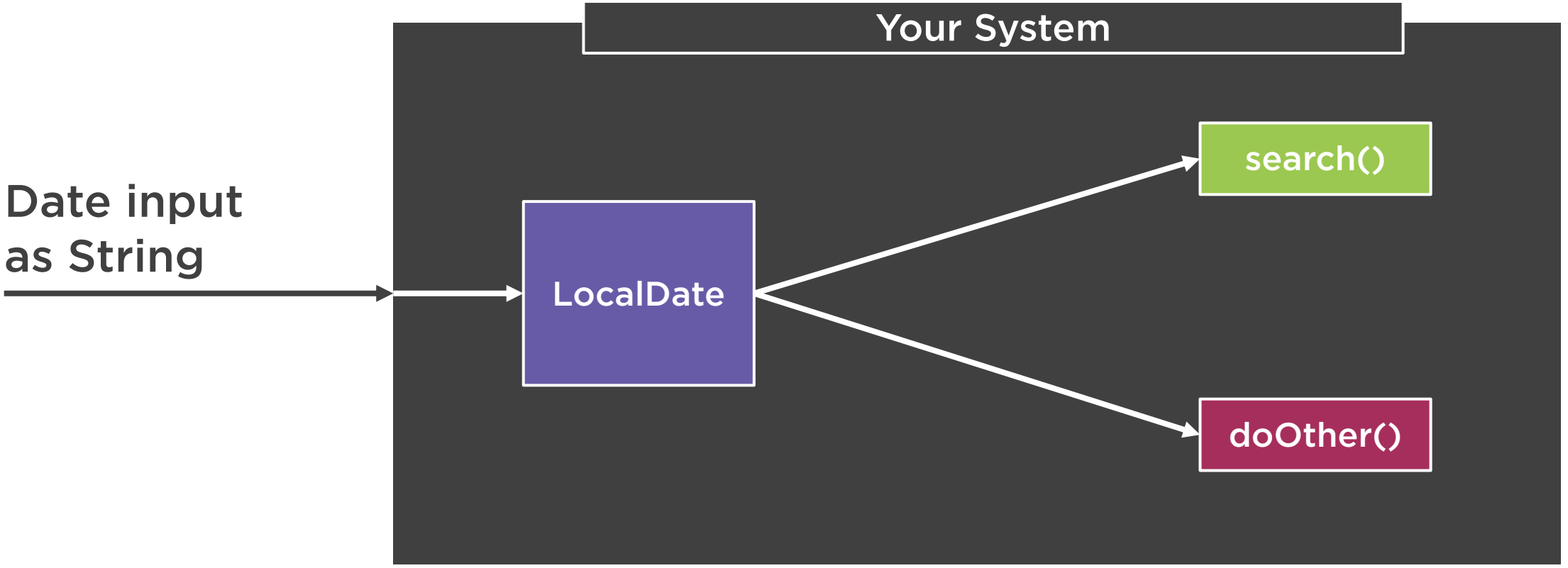
Immutability

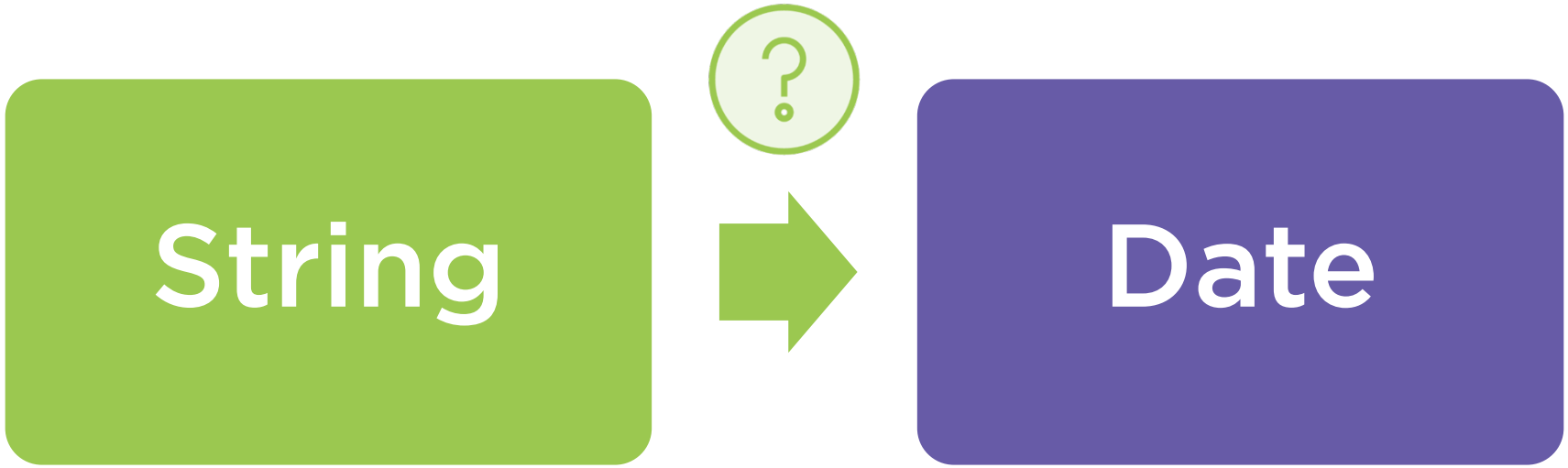
An object is immutable if you can’t modify its state after creation

Mutable Date

```
Date start = new Date();

Date end = new Date();

TimeWindow booking = new TimeWindow(start, end);

start.setHours(5); // modifies internals of TimeWindow
```

java.util.Date requires defensive copying


Your System
Date input
as String
LocalDate
search()
doOther()



1) Validate
2) Create the object using the String as a parameter

Combinatorial Testing

Combinations

A & B

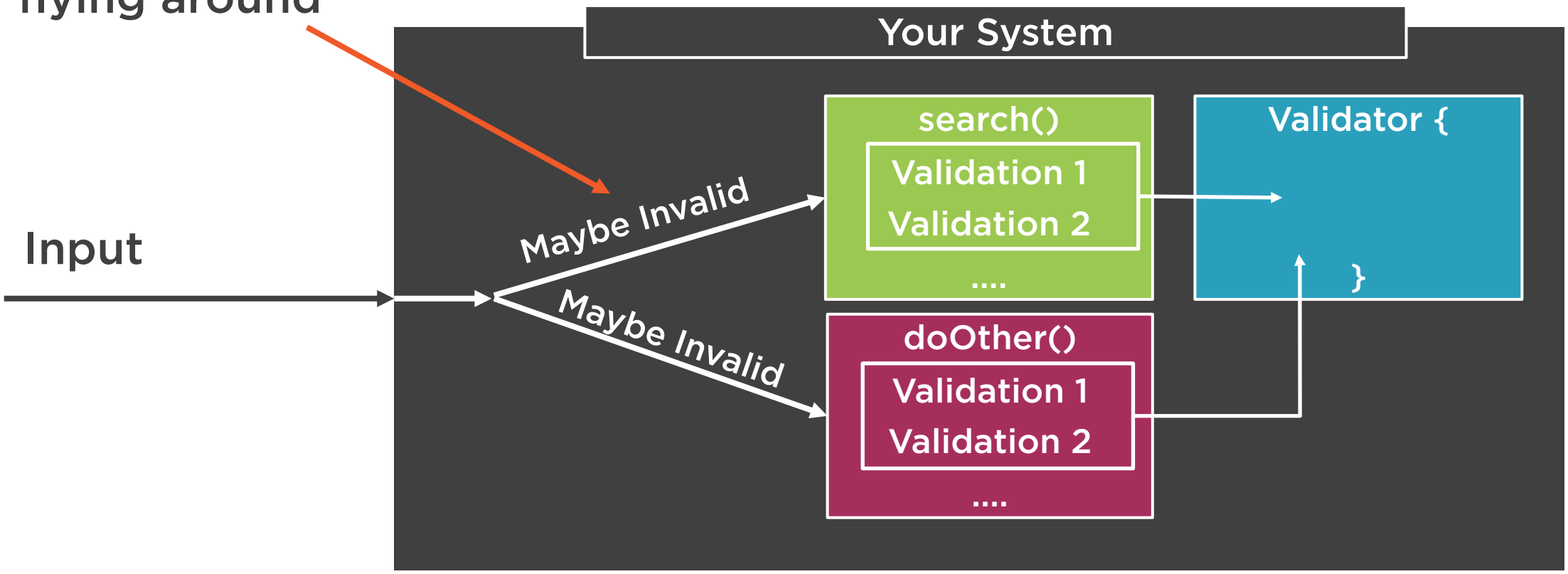
if(A == B)

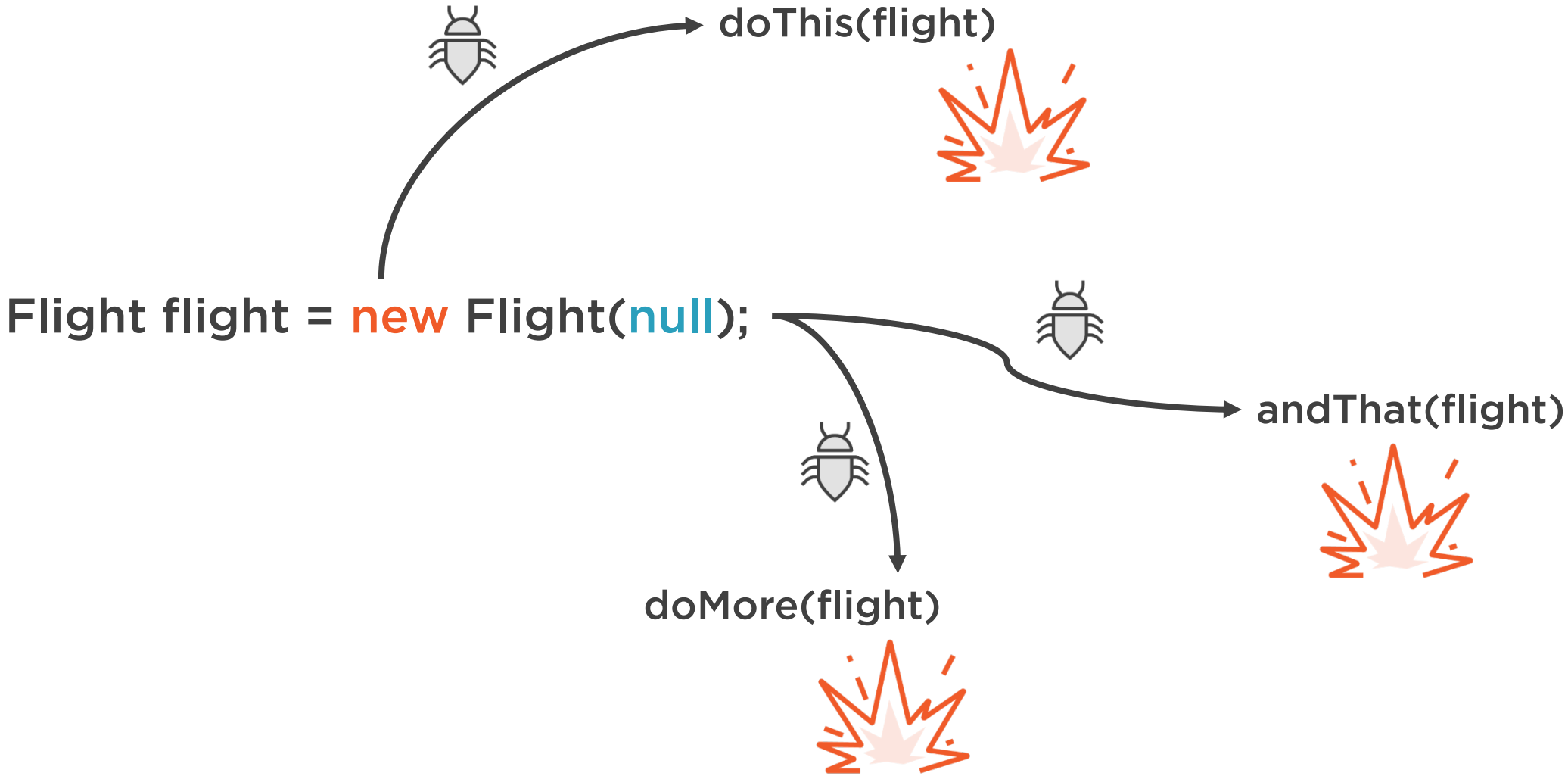
A & B & C & D

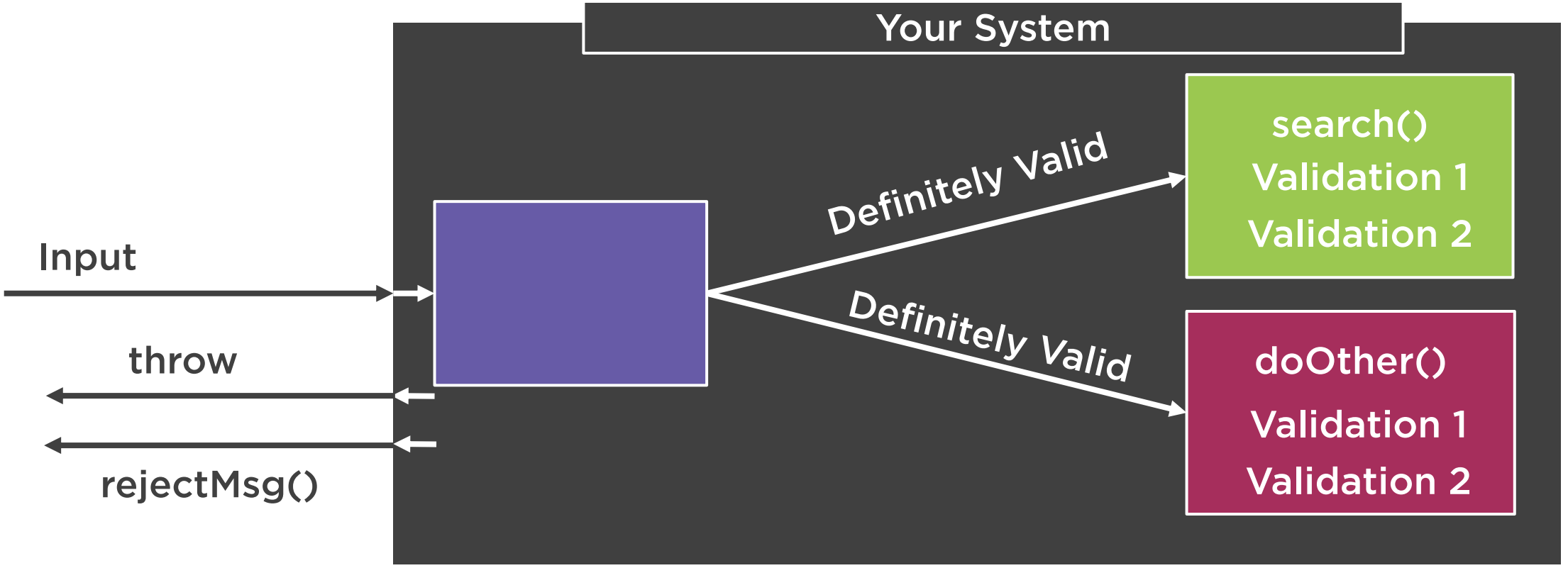
// things get much more

complicated

Invalid data is still flying around

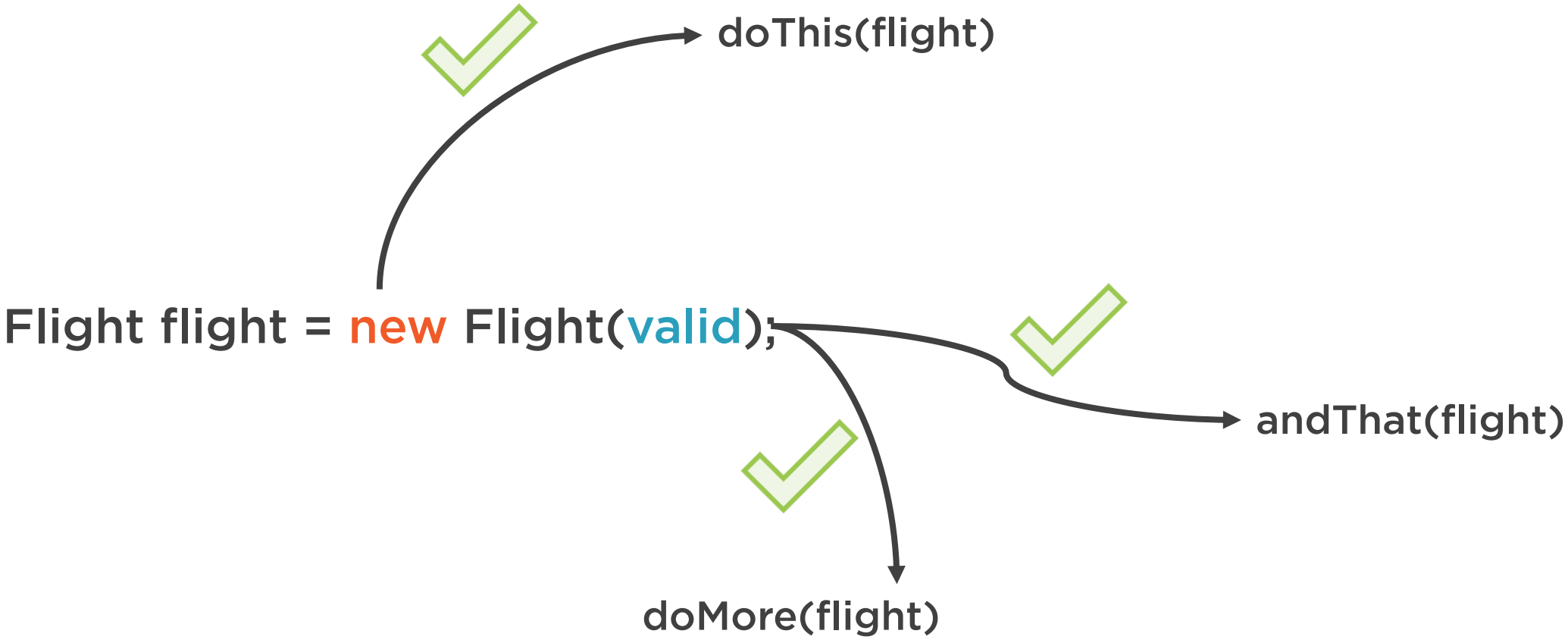


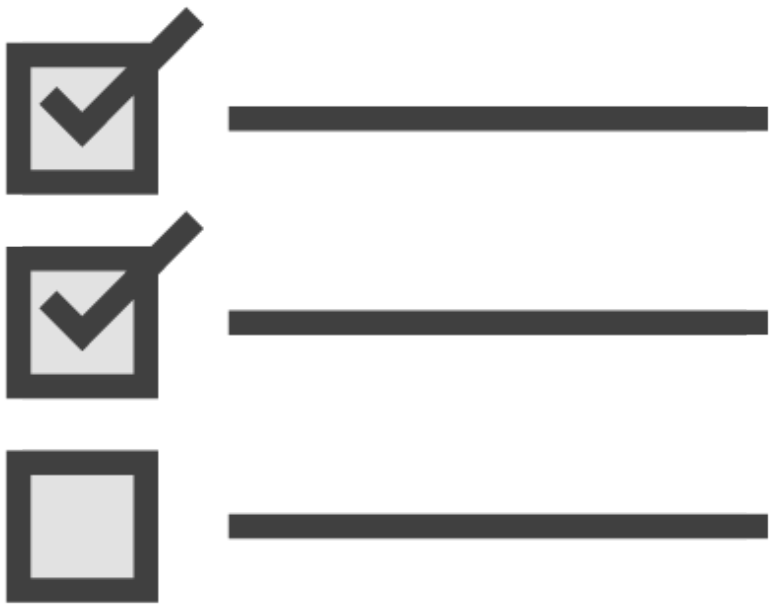

doThis(flight)
Flight flight = new Flight(null);
andThat(flight)
doMore(flight)


Your System
search()
Validation 1
Validation 2
doOther()
Validation 1
Validation 2
Definitely Valid
Definitely Valid
Input
throw
rejectMsg()

Class Invariant

A property that always remains true for all instances of a class no matter what happens


doThis(flight)
Flight flight = new Flight(valid);
andThat(flight)
doMore(flight)



Defend and fail early:

- In methods (good)
- In constructors (if possible)

Protect class invariants

Throwing

DONTs

Throw top-level Error, Exception, RuntimeException or Throwable

DOs

Throw specific exceptions

- IllegalArgumentException
- IllegalStateException
- NullPointerException
- UnsupportedOperationExcetion

NPE

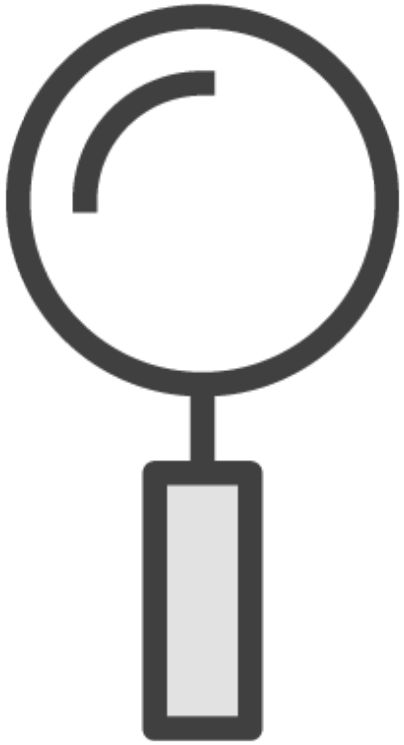
Don’t catch NPEs

OK to throw NPEs when validating parameters

"If a caller passes null in some parameter for which null values are prohibited, convention dictates that NullPointerException be thrown rather than IllegalArgumentException"

Joshua Bloch, Effective Java

What We Didn't Cover



Numbers:

- Careful when dividing

Strings:

- Use Enums where appropriate
- Comparison order

Avoid using assert

- More freedom with Exceptions
- Asserts can be (accidentally) switched with –da flag

```
double calculate(double input){
        return 100 / input; // ArithmeticException if 0
}

double calculate(double input){
        return input / 100; // 0, probably not what you wanted
}
```

null

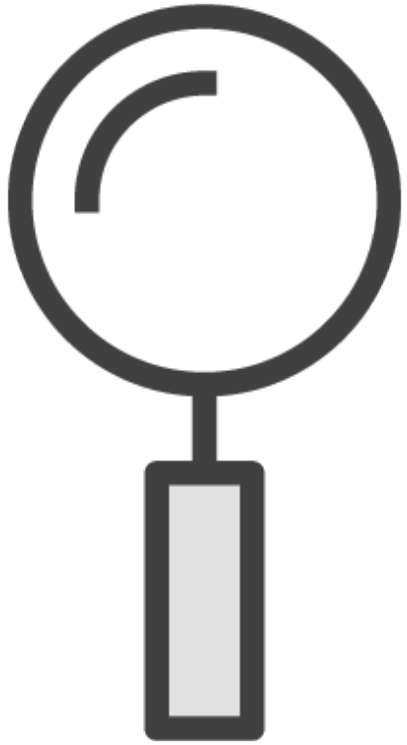
```
inputString.equals("MyConstant");
```

null

```
"MyConstant".equals(inputString);
```

false

What We Didn't Cover (Continued)



UI specific:

- Use dropdowns and widgets to limit input options (same as Enums limit input options compared to plain Strings)

Summary

- Null checking DOs and DONTs
- Concepts: Guard Clause & “Fail Early”
- String verification techniques and pitfalls
- Dealing with numbers
- Handling dates
- Considering value combinations
- Constructing safe(r) Objects

React

Fail early with a guard clause
in methods

Prevent

Fail early with a guard clauses
in constructors