Cracking the Code to Secure Software

Daniel Deogun & Daniel Sawano

Platinum Sponsor: CROSSOVER

Daniel Deogun

Daniel Sawano

@DanielDeogun @DanielSawano #DevoxxPL

Secure by Design

Secure by Design is a new approach to software security that lets you create secure software while still focusing on business features.

Secure by Design

“Any activity involving ***active decision making*** *should be considered part of the software design process and can thus be referred to as* ***design****.”*

- Johnsson, Deogun, and Sawano

Key Takeaway

By focusing on good design principles you can create secure software without constantly thinking about security.

What we'll cover today…

- Domain Primitives
- Entity Snapshots
- Dealing with Legacy Code
- Security in your Delivery Pipeline
- Domain DoS Attacks

Design patterns

Security & tests

Upcoming threat

Domain Primitives

A value object so precise in its definition that it, by its mere existence, manifests its validity is called a ***domain primitive****.*

Domain Primitives

- A Domain Primitive is very strict in its definition
- If it's not valid then it cannot exist
- Defined in the current domain
- It's preciseness brings robustness in your code
- It's immutable so it will always be valid

Domain Primitives

```
import static org.apache.commons.lang3.Validate.inclusiveBetween;
import static org.apache.commons.lang3.Validate.notNull;

public final class Quantity {
   private final int value;

   public Quantity(final int value) {
      inclusiveBetween(1, 200, value);
      this.value = value;
   }

   public int value() {
      return value;
   }

   public Quantity add(final Quantity addend) {
      notNull(addend);
      return new Quantity(value + addend.value);
   }

   // ...
}
```

Quantity is not just an int!

- Enforces invariants at creation
- Provides domain operations to
- Encapsulate domain behavior

CIA

- **Confidentiality** - protecting data from being read by unauthorized users
- **Integrity** - ensures data is changed in an authorized way
- **Availability** - concerns having data available when authorized users need it

Not this

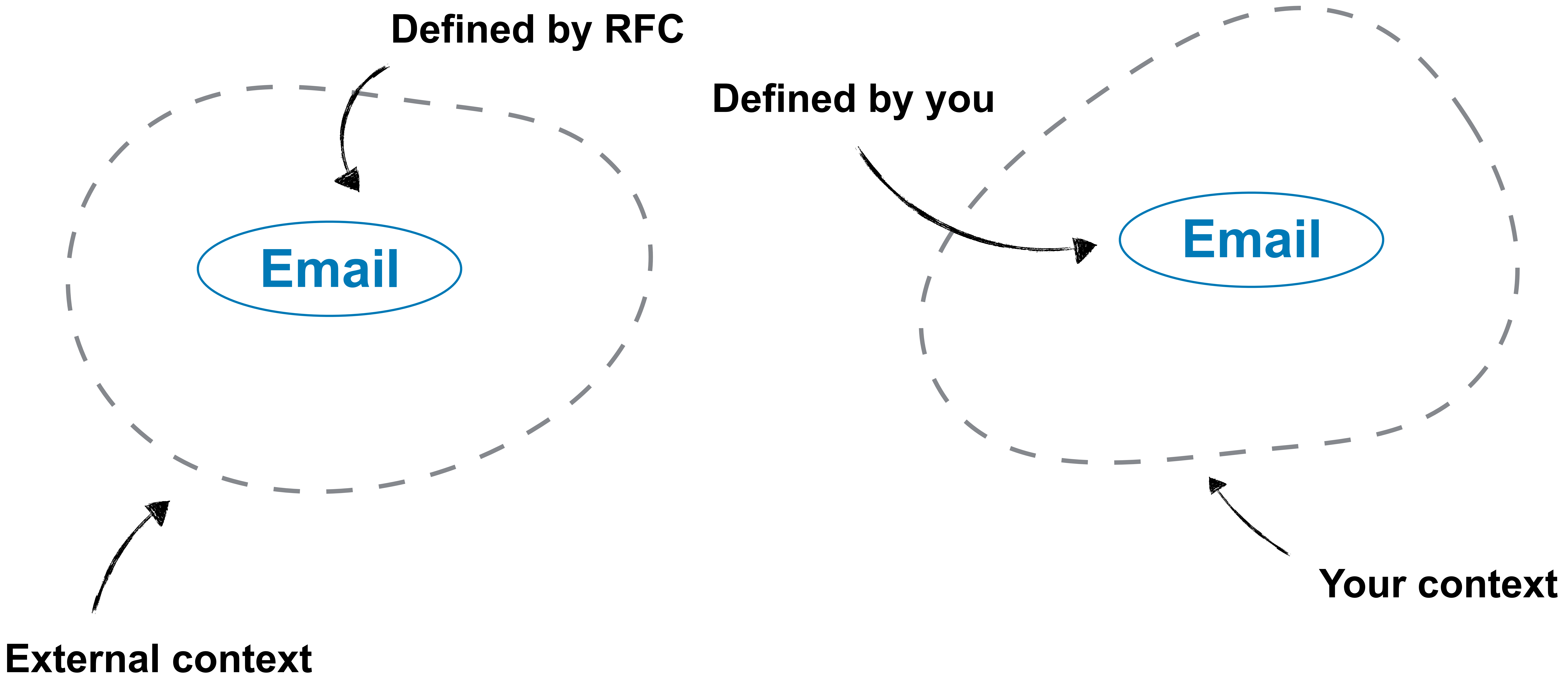
Domain Primitives

```
import static org.apache.commons.lang3.Validate.inclusiveBetween;
import static org.apache.commons.lang3.Validate.notNull;

public final class Quantity {
   private final int value;

   public Quantity(final int value) {
      inclusiveBetween(1, 200, value);
      this.value = value;
   }

   public int value() {
      return value;
   }

   public Quantity add(final Quantity addend) {
      notNull(addend);
      return new Quantity(value + addend.value);
   }

   // ...
}
```

Quantity is not just an int!

- Enforces invariants at creation
- Provides domain operations to
- Encapsulate domain behavior

Domain Primitives



Domain Primitives

Use Domain Primitives as:

- the smallest building block in your domain model
- to build your Domain Primitive Library
- to harden your code and your APIs

What we'll cover today…

- ✓ Domain Primitives
- **Entity Snapshots**
- Dealing with Legacy Code
- Security in your Delivery Pipeline
- Domain DoS Attacks

Design patterns

Security & tests

Upcoming threat

Entities

- An entity has an identity that doesn't change over time
- The values/data belonging to an entity can change over time
- Typically modeled as mutable objects

Classic Entity

```java
public final class Order {

    private final OrderId id;
    private final List<OrderItem> orderItems = new ArrayList<>();

    public Order(final OrderId id) {
        this.id = notNull(id);
    }

    public void addItem(final OrderItem item) {
        notNull(item);
        orderItems.add(item);
    }

    // ...
}
```

Entity Snapshots

Entity Snapshots are:

- Securing mutable state by making it immutable
- An immutable representation of a mutable entity
- Solves many of the security problems with regular entities

Entity Snapshots

```
public final class Order {

   private final OrderId id;
   private final List<OrderItem> orderItems;

   public Order(final OrderId id, final List<OrderItem> orderItems) {
      noNullElements(orderItems);
      notNull(id);
      this.id = id;
      this.orderItems = unmodifiableList(new ArrayList<>(orderItems));
   }

   public List<OrderItem> orderItems() {
      return orderItems;
   }

   // ...
}
```

Entity Snapshots

```
public final class WritableOrder {

   private final OrderId id;
   private final OrderRepository repository;

   public WritableOrder(final OrderId id, final OrderRepository repository) {
      this.id = notNull(id);
      this.repository = notNull(repository);
   }

   public void addOrderItem(final OrderItem orderItem) {
      notNull(orderItem);
      isOkToAdd(orderItem);
      repository.addItemToOrder(id, orderItem);
   }

   private void isOkToAdd(final OrderItem orderItem) {
      // domain validation logic to ensure it's ok to add order
   }
}
```

What we'll cover today…

- ✓ Domain Primitives
- ✓ Entity Snapshots
- **Dealing with Legacy Code**
- Security in your Delivery Pipeline
- Domain DoS Attacks

Design patterns

Security & tests

Upcoming threat

Dealing with Legacy Code

3 good design patterns

[6]

Draw the Line

[7]

Harden your APIs

[8]

Declutter Entities

Draw the Line

- We need to identify the semantic boundary of a context
- Add a layer that internally translates data to a domain primitive and then back again

```
data -> domain primitive -> data
```

- This way, we have created a validation boundary that protects the inside from bad input
- But, if rejecting data is to harsh, consider logging it for insight

[6]

Harden the API

- Create a library of domain primitives
- Express your APIs with your domain primitives
- Never accept generic input if you have specific requirements

[7]

Generic		Specific
`void buyBook(String, int)`	$\longrightarrow$	`void buyBook(ISBN, Quantity)`

Decluttering Entities

```
import static org.apache.commons.lang3.Validate.notNull;
import static org.apache.commons.lang3.Validate.isTrue;
public class Order {

   private final List<Object> items;
   private boolean paid;

   public void addItem(String isbn, int qty) {
        if (this.paid == false) {
            notNull(isbn);
            isTrue(isbn.length() == 10);
            isTrue(isbn.matches("[0-9X]*"));
            isTrue(isbn.matches("[0-9]{9}[0-9X]"));
            if (inventory.avaliableBooks(isbn, qty)) {
                Book book = bookcatalogue.findBy(isbn);
                items.add(new OrderLine(book, qty));
            }
        }
    }
```

[8]

Decluttering Entities

```
import static org.apache.commons.lang3.Validate.notNull;
import static org.apache.commons.lang3.Validate.isTrue;
public class Order {

   private final List<Object> items;
   private boolean paid;

   public void addItem(final **ISBN** isbn,
                       final **Quantity** quantity) {
        notNull(isbn);
        notNull(quantity);
        isTrue(notPaid());

        if (inventory.avaliableBooks(isbn, quantity)) {
            Book book = bookcatalogue.findBy(isbn);
            items.add(new OrderLine(book, quantity));
        }
    }
```

[8]

What we'll cover today…

- ✓ Domain Primitives
- ✓ Entity Snapshots
- ✓ Dealing with Legacy Code
- **Security in your Delivery Pipeline**
- Domain DoS Attacks

Design patterns

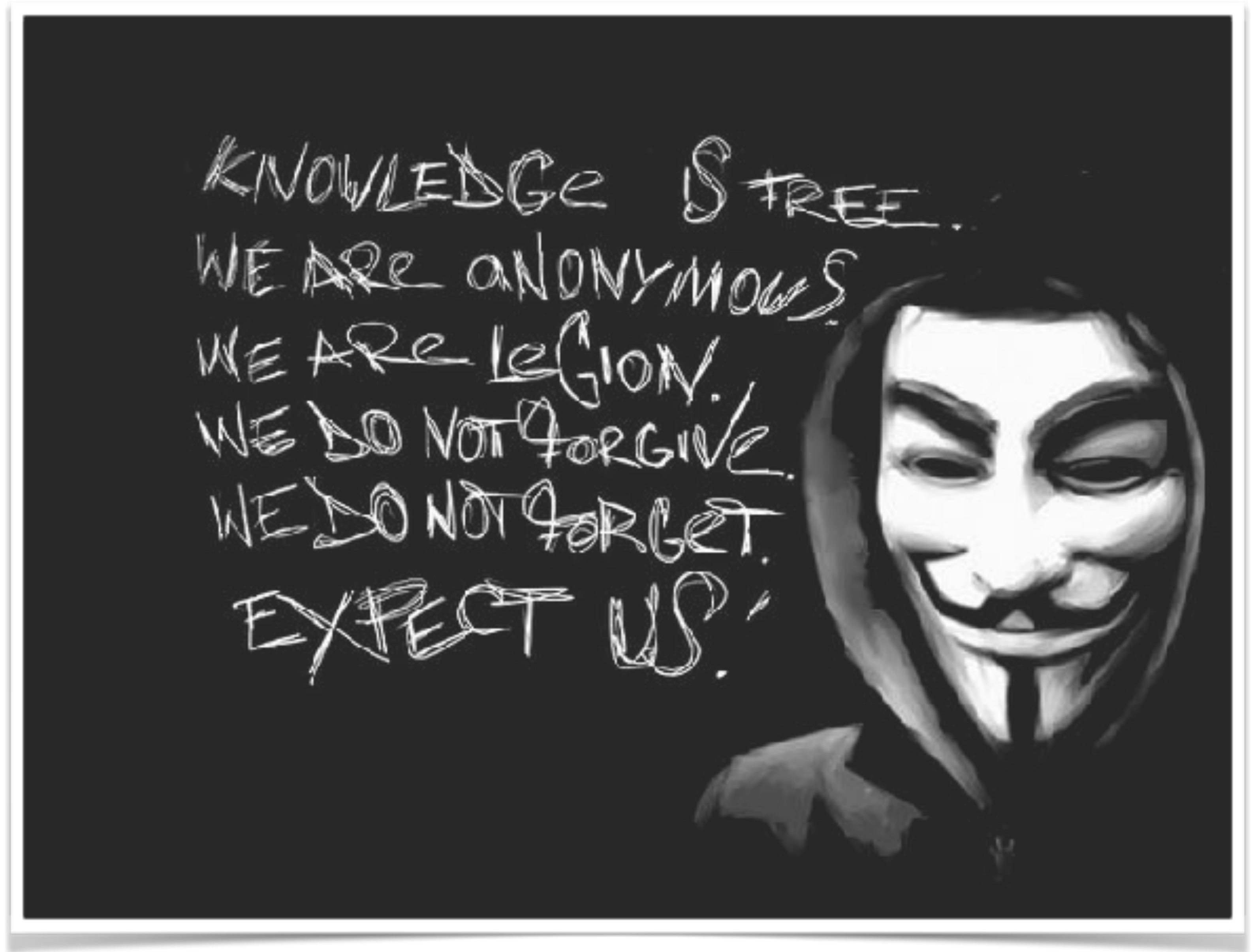
Security & tests

Upcoming threat

Security in your Delivery Pipeline

- Unit Testing

[12]

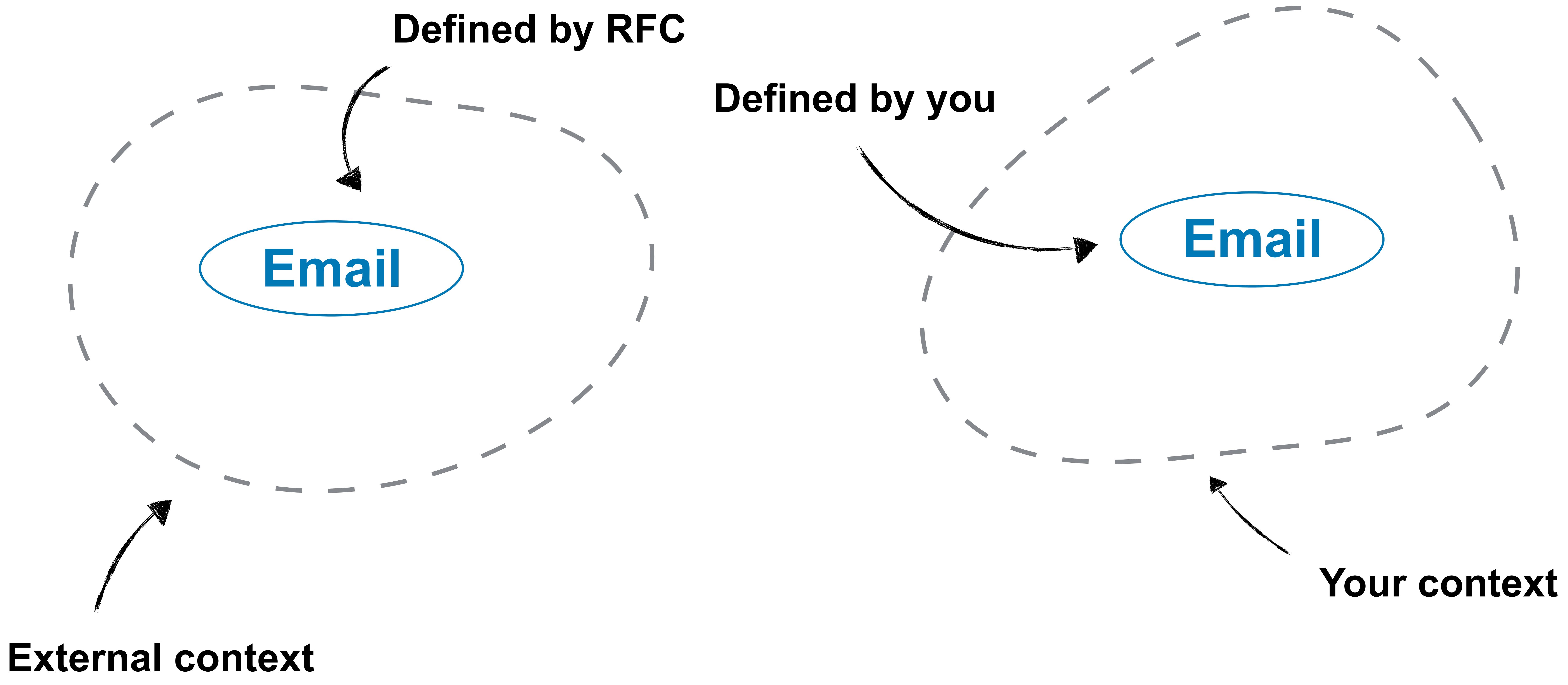


[10]

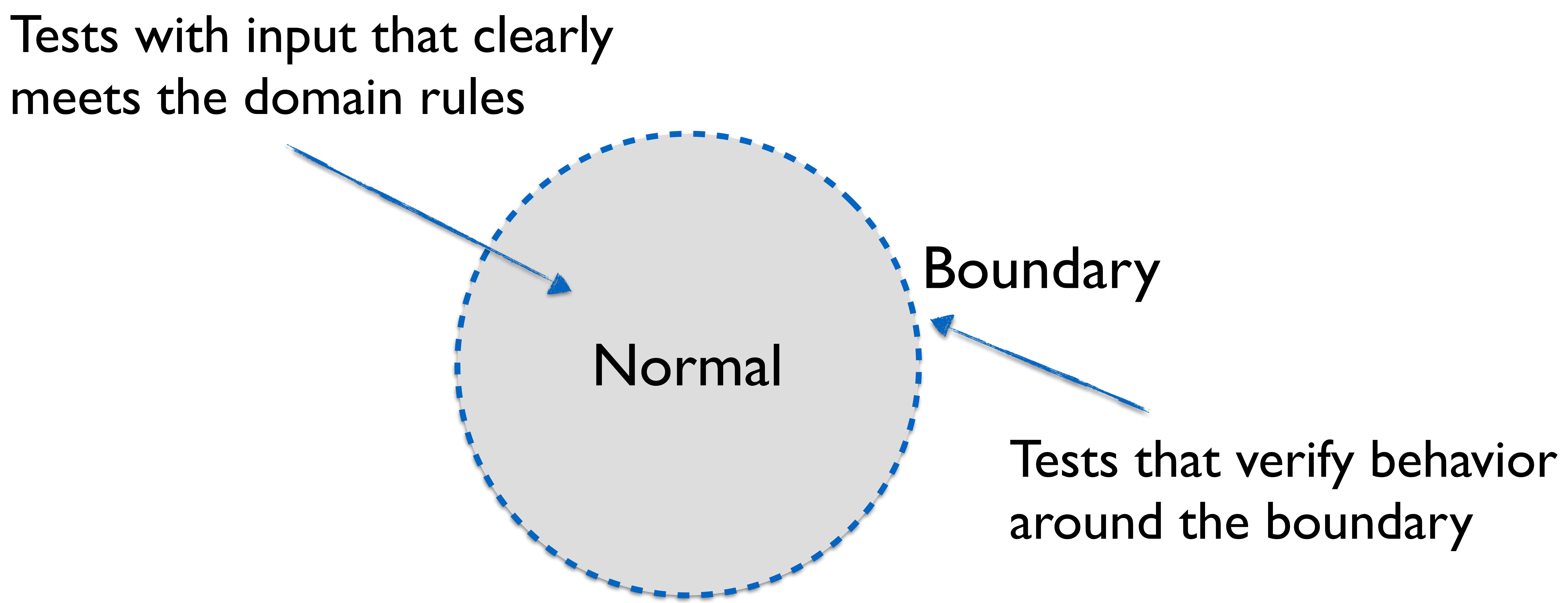
The Hospital Case

[9]

Email Domain Primitive



Normal & Boundary Testing


Tests with input that clearly meets the domain rules
Normal
Boundary
Tests that verify behavior around the boundary

Email Address v1.0

```
public final class EmailAddress {

   public final String value;

   public EmailAddress(final String value) {
      matchesPattern(value.toLowerCase(),
        "^(?=[a-z0-9.@]{15,77}$)[a-z0-9]+\\.?[a-z0-9]+@\\bhospital.com$");

      this.value = value.toLowerCase();
   }
   ...
}
```

Invalid Input Testing

- Any input that doesn't satisfy the domain rules is considered invalid
- For some reason, `null`, empty strings, or "strange" characters tend to result in unexpected behavior

Testing with invalid input

```
@TestFactory
Stream<DynamicTest> should_reject_invalid_input() {
   return Stream.of(
         null,
         "null",
         "nil",
         "0",
        "",
         " ",
         "\t",
         "\n",
         "john.doe\n@hospital.com",
         "   @hospital.com",
         "%20@hospital.com",
         "john.d%20e@hospital.com",
         "john.doe.jane@hospital.com",
         "--",
         "e x a m p l e @ hospital . c o m",
         "=0@$*^%;<!->.:\\()&#\"")
         .map(input -> dynamicTest("Rejected: " + input, assertInvalidEmail(input)));
}
```

Email Address v2.0

```
public final class EmailAddress {

   public final String value;

   public EmailAddress(final String value) {
       notNull(value, "Input cannot be null");
      matchesPattern(value.toLowerCase(),
        "^(?=[a-z0-9.@]{15,77}$)[a-z0-9]+\\.?[a-z0-9]+@\\bhospital.com$");

      this.value = value.toLowerCase();
   }
   ...
}
```

Testing with Extreme Input

- Testing the extreme is all about identifying weaknesses in the design that makes the application break or behave strangely when handling extreme values.

```
@TestFactory
Stream<DynamicTest> should_reject_extreme_input() {
   return Stream.<Supplier<String>>of(
         () -> repeat("x", 10000),
         () -> repeat("x", 100000),
         () -> repeat("x", 1000000),
         () -> repeat("x", 10000000),
         () -> repeat("x", 20000000),
         () -> repeat("x", 40000000))
         .map(input -> dynamicTest("Rejecting extreme input",
                       assertInvalidEmail(input.get())));
}
```

Inefficient Backtracking

```
"^(?=[a-z0-9.@]{15,77}$)[a-z0-9]+\\.?[a-z0-9]+@\\bhospital.com$"
```

V.S

```
“^[a-z0-9]+\\.?[a-z0-9]+@\\bhospital.com$"
```

What we'll cover today…

- ✓ Domain Primitives
- ✓ Entity Snapshots
- ✓ Dealing with Legacy Code
- ✓ Security in your Delivery Pipeline
- ● **Domain DoS Attacks**

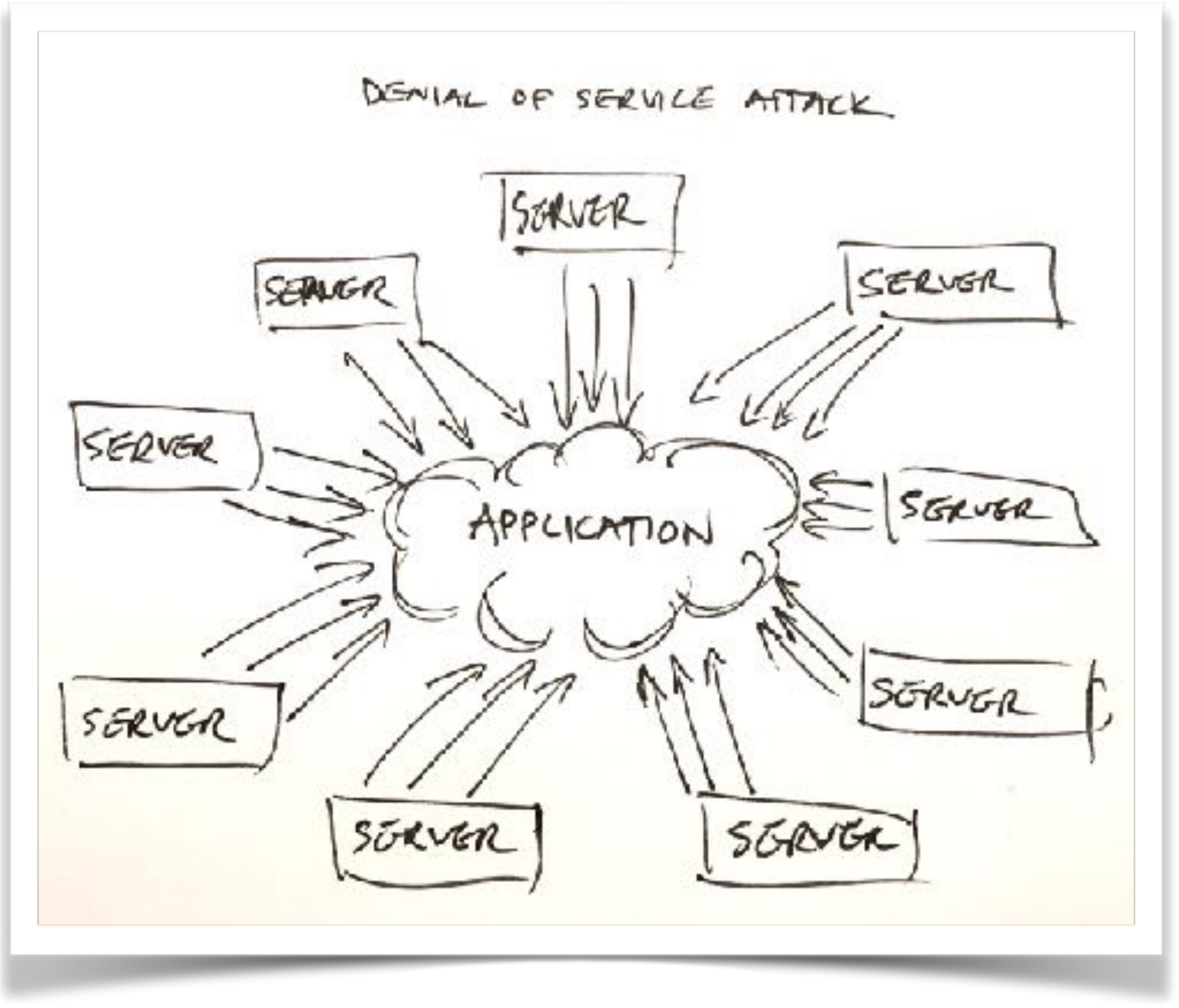
Design patterns

Security & tests

Upcoming threat

DoS Attacks

- The main objective of a DoS attack is to prevent availability of a system's services
- A DoS attack doesn't require heavy load to be successful (asymmetric)



Domain DoS Attacks

A DoS attack caused by utilizing ***domain*** *rules in a malicious way is called a* ***Domain DoS***

Domain DoS Example - The Hotel

We need great customer service!

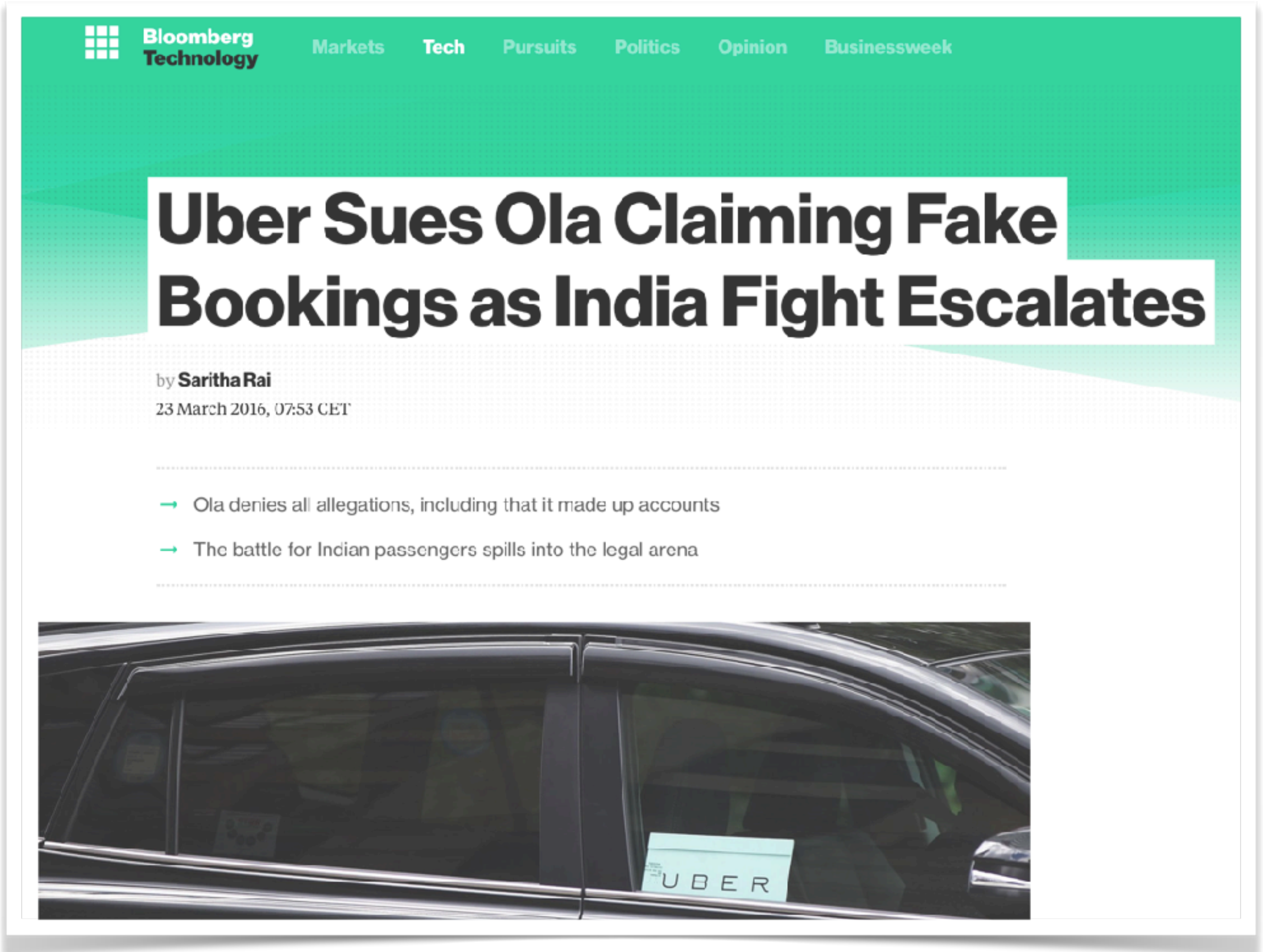
Full refund if cancelled before 4 p.m.

[9]

Book all empty rooms

No rooms available for ordinary customers

Uber vs Ola


Bloomberg Technology Markets Tech Pursuits Politics Opinion Businessweek

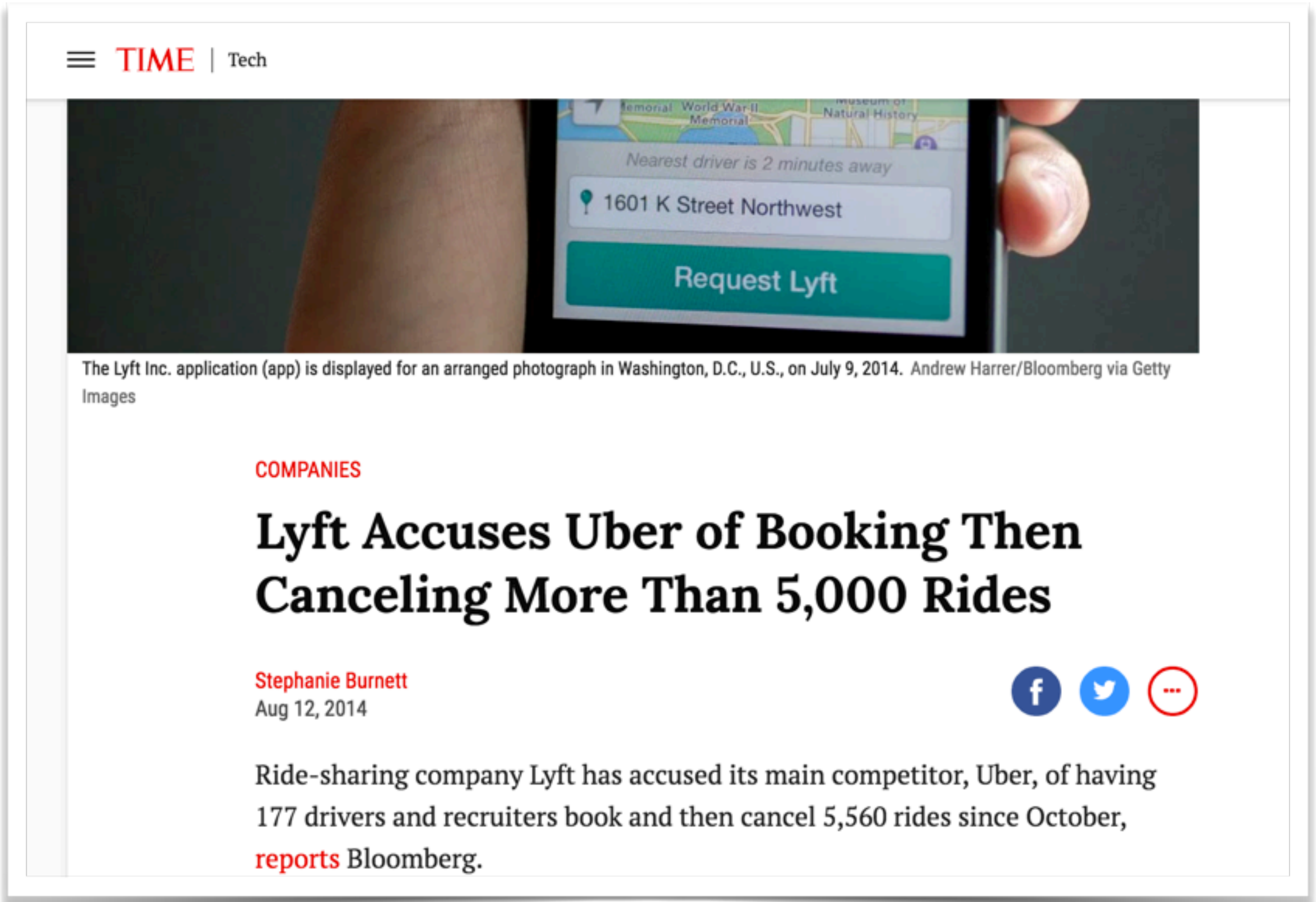
Uber Sues Ola Claiming Fake Bookings as India Fight Escalates

by Saritha Rai
23 March 2016, 07:53 CET

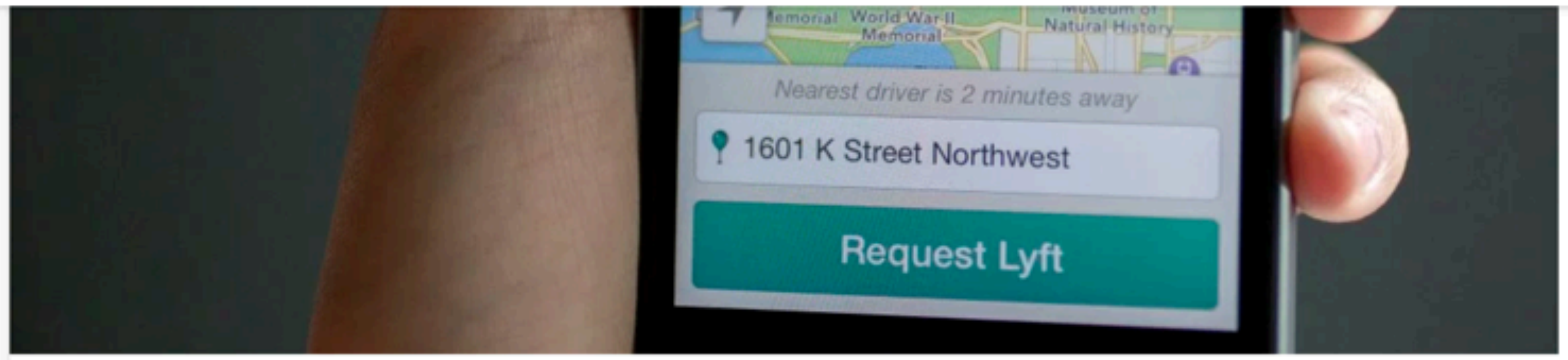
- Ola denies all allegations, including that it made up accounts
- The battle for Indian passengers spills into the legal arena

[4]

Lyft vs Uber



TIME | Tech



The Lyft Inc. application (app) is displayed for an arranged photograph in Washington, D.C., U.S., on July 9, 2014. Andrew Harrer/Bloomberg via Getty Images

COMPANIES

Lyft Accuses Uber of Booking Then Canceling More Than 5,000 Rides

Stephanie Burnett
Aug 12, 2014

Ride-sharing company Lyft has accused its main competitor, Uber, of having 177 drivers and recruiters book and then cancel 5,560 rides since October, reports Bloomberg.

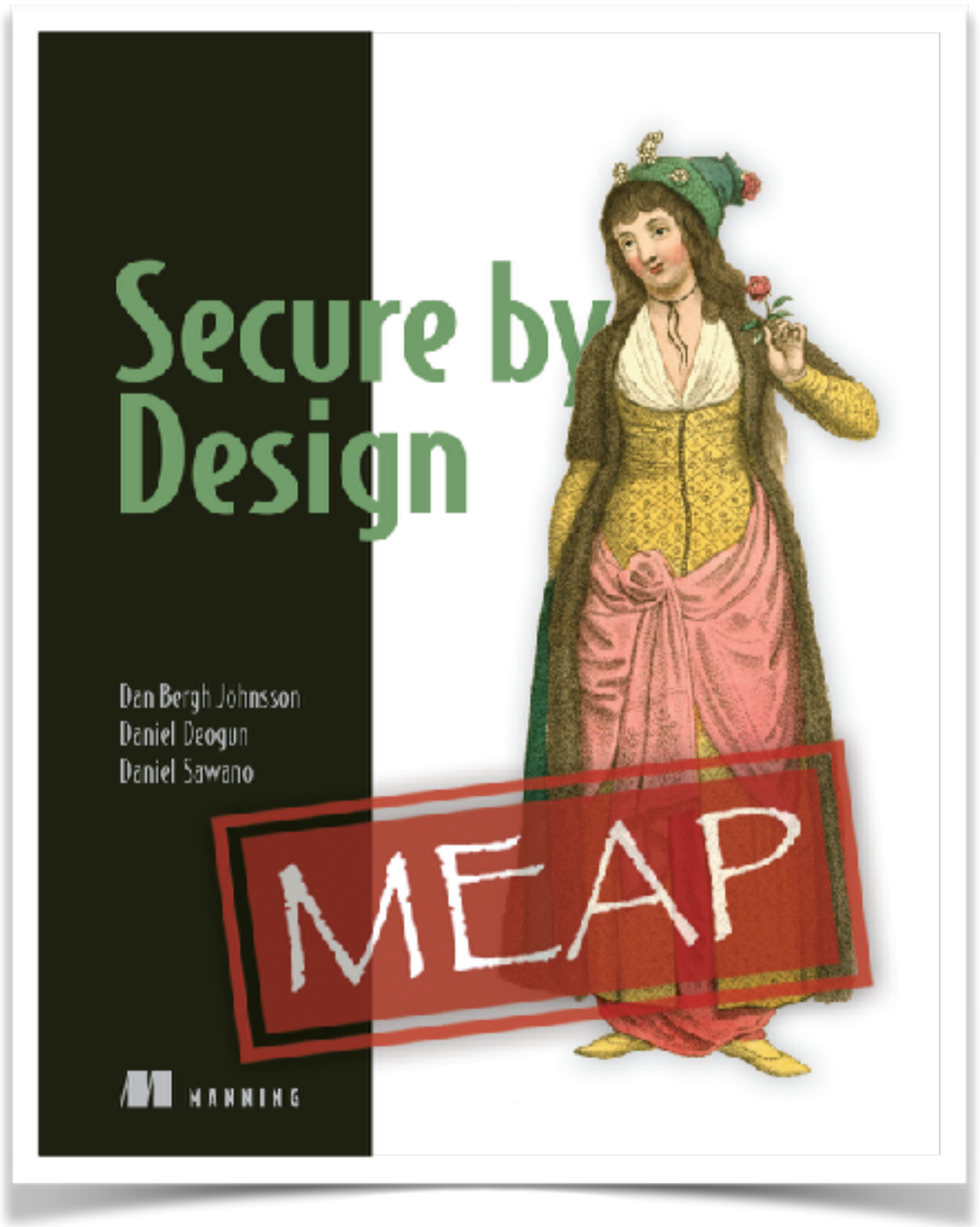
[5]

@DanielDeogun @DanielSawano #DevoxxPL

Key Takeaway

By focusing on good design principles you can create secure software without constantly thinking about security.

Questions & More



Want a free ebook?

Catch us in the break!

[2]

URL: http://bit.ly/secure-by-design
Discount code: ctwdevoxxpl17 (40% off)

References

[1] https://www.flickr.com/photos/stewart/461099066 by Stewart Butterfield under license https://creativecommons.org/licenses/by/2.0/
[2] https://flic.kr/p/9ksxQa by Damián Navas under license https://creativecommons.org/licenses/by-nc-nd/2.0/
[3] https://flic.kr/p/nEZKMd by Graeme Fowler under license https://creativecommons.org/licenses/by/2.0/
[4] Uber vs Ola, https://www.bloomberg.com/news/articles/2016-03-23/uber-sues-ola-claiming-fake-bookings-as-india-fight-escalates
[5] Lyft vs Uber, http://time.com/3102548/lyft-uber-cancelling-rides/
[6] 3d key, https://flic.kr/p/e9qfrf by Yoel Ben-Avraham under license https://creativecommons.org/licenses/by-nd/2.0/
[7] Building blocks, https://flic.kr/p/agPw7C by Tiffany Terry under license https://creativecommons.org/licenses/by/2.0/
[8] Doctors Stock Photo, https://flic.kr/p/HNJUzV, by Sergio Santos under license https://creativecommons.org/licenses/by/2.0/
[9] "Anonymous" - Icon made by Egor Rumyantsev from www.flaticon.com - CC 3.0