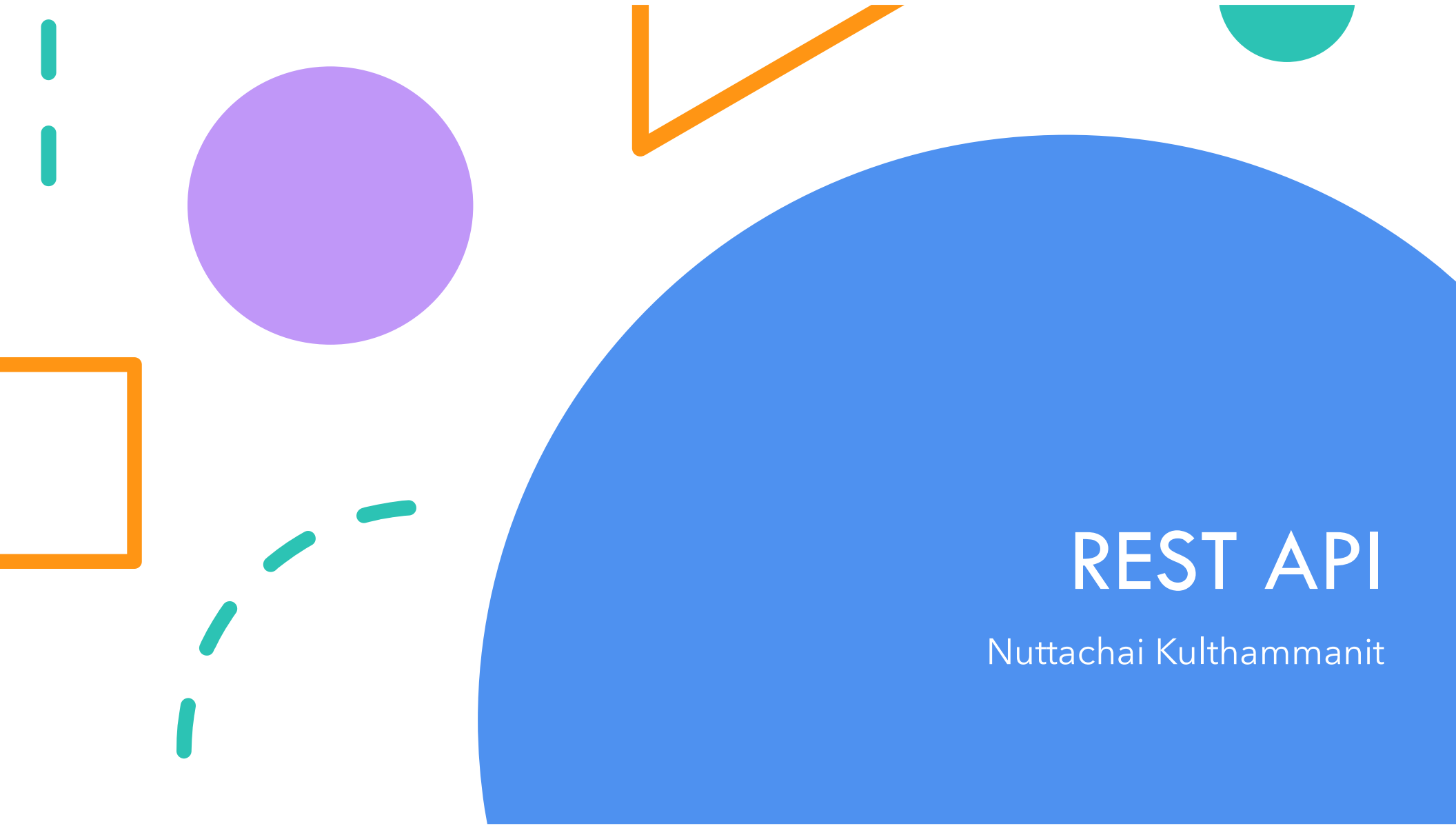

REST API
Nuttachai Kulthammanit

Topics

- สร้าง REST APIs / Web Services ด้วย Spring
- หลักการ REST, JSON และ HTTP messaging
- การใช้ Postman
- สร้าง REST APIs สำหรับ CRUD กับ Database

Rest APIs - REST Web Service

Chapter 1

สิ่งที่เราจะได้เรียนรู้คือ วิธีการ...

- สร้าง REST APIs / Web Services ด้วย Spring
- พูดคุยเกี่ยวกับแนวคิด REST, JSON และ HTTP messaging
- ติดตั้งเครื่องมือ REST client : Postman
- พัฒนา REST APIs / Web Services ด้วย @RestController
- สร้าง CRUD APIs และเชื่อมต่อไปยังฐานข้อมูลด้วย Spring REST

ผลลัพธ์ที่จะได้

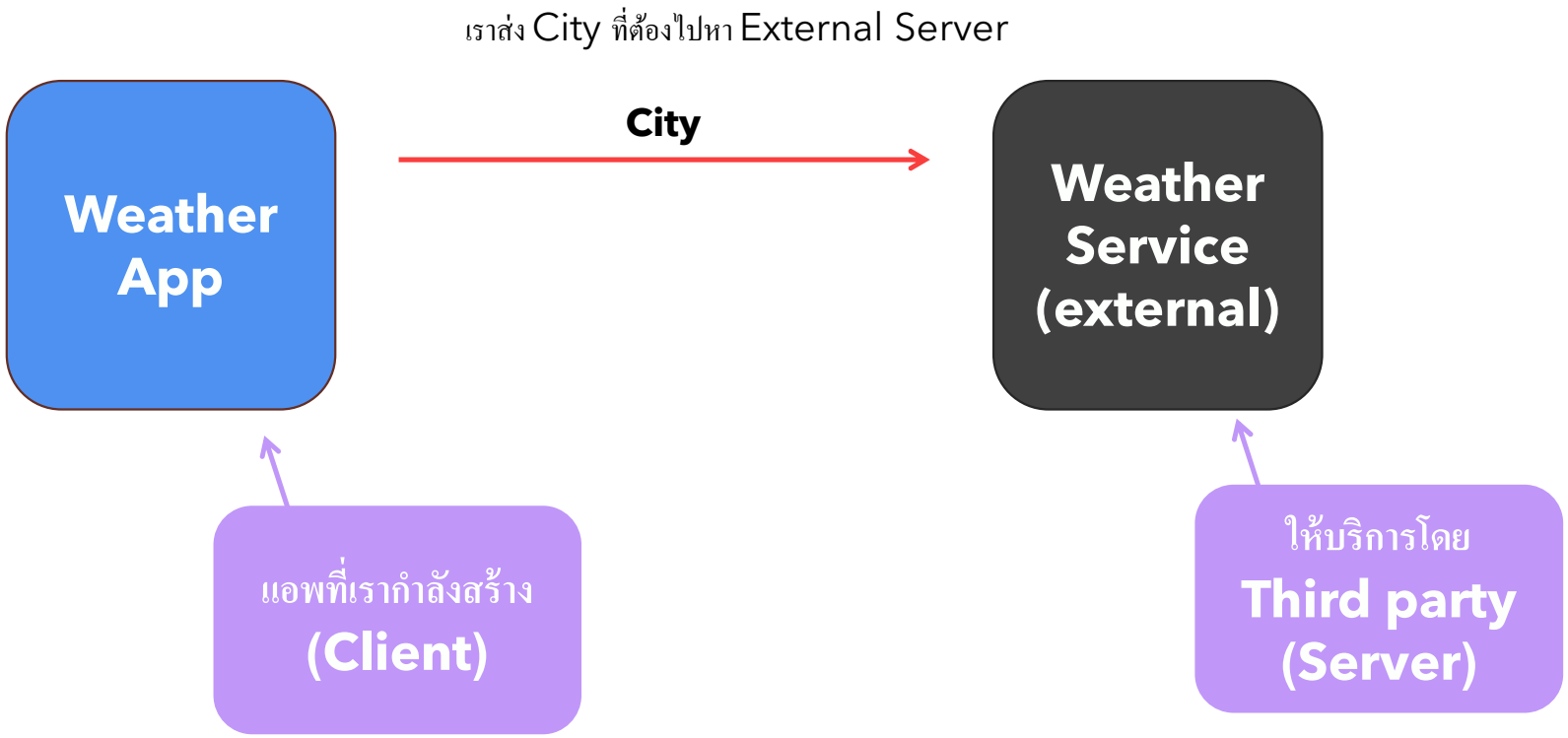
- ในบทนี้เราจะเรียน Spring REST เบื้องต้น
- แต่สามารถดูคู่มืออ้างอิง Spring เพิ่มเติมได้ที่ลิงค์ด้านล่าง

https://spring.io/projects/spring-framework

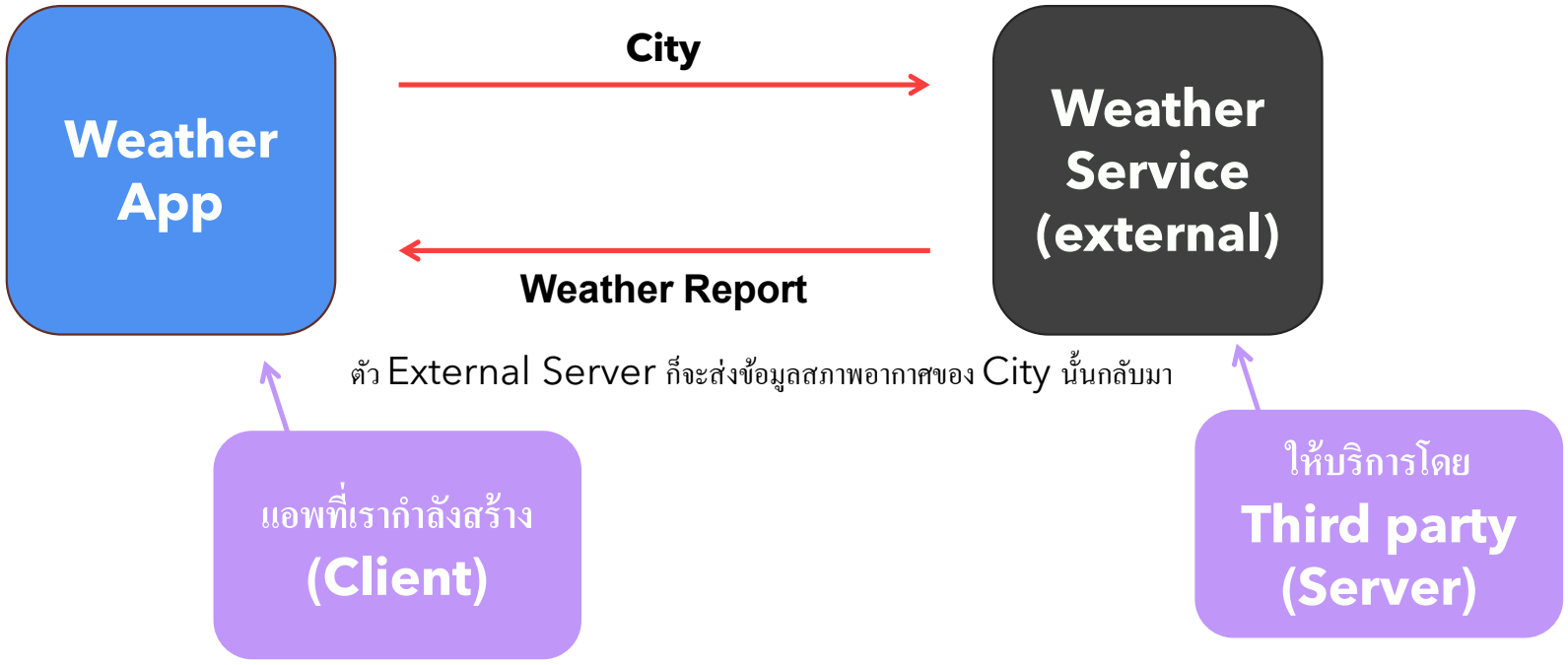
ตัวอย่าง Application

- สมมติเราจะสร้าง Application ที่ให้รายงานสภาพอากาศ
- เพราะฉะนั้นเราต้องดึงข้อมูลสภาพอากาศจาก External Service

ภาพรวมของแอพพลิเคชัน

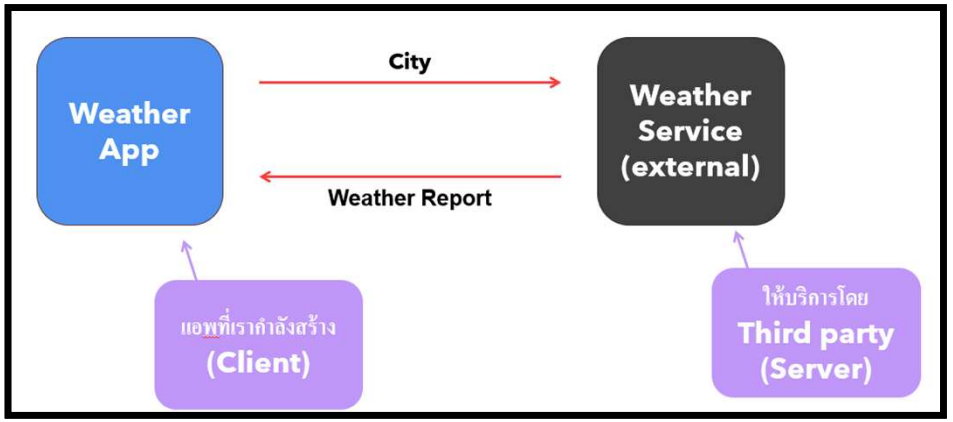


ภาพรวมของแอพพลิเคชัน



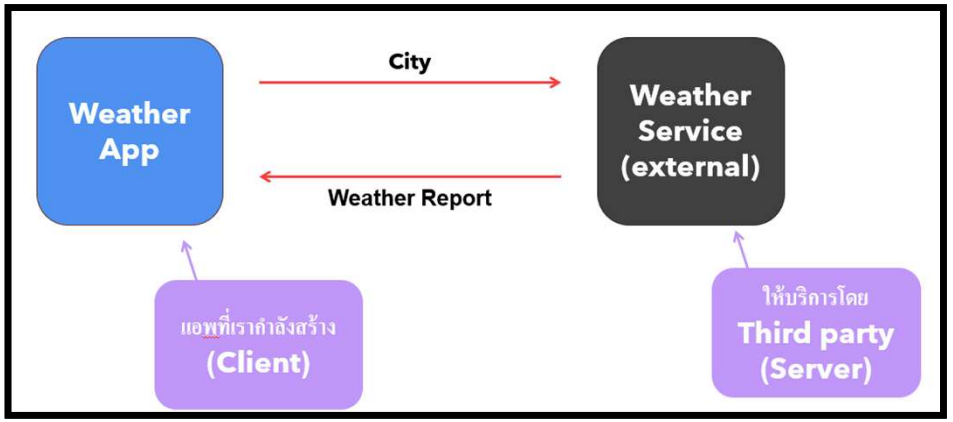
แต่เรายังมีคำถามอยู่ เช่น

- เราจะเชื่อมต่อกับ Weather Service ยังไง
- เราจะใช้ภาษาอะไร
- รูปแบบข้อมูล (data format) คืออะไร?



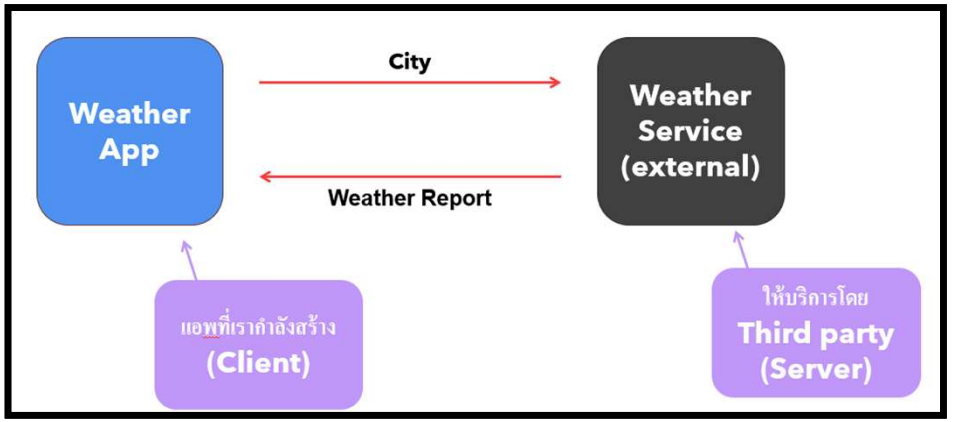
คำตอบ

- **เราจะเชื่อมต่อกับ Weather Service ยังไง**
  - เราจะใช้ REST API ผ่าน HTTP
  - REST: REpresentational State Transfer
  - เป็นวิธีการที่ Lightweight สำหรับการการติดต่อสื่อสารระหว่าง แอพพลิเคชัน



คำตอบ

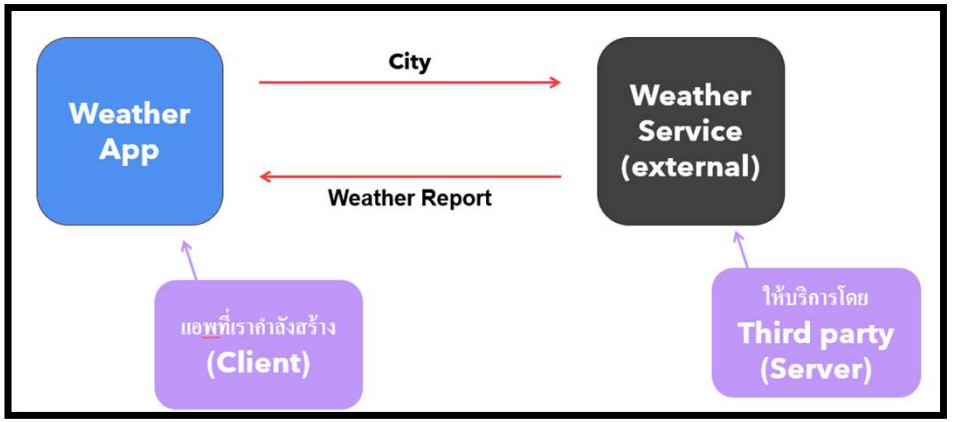
- **เราจะใช้ภาษาอะไร**
  - REST ไม่ได้ขึ้นอยู่กับ ภาษา
  - ไคลเอนต์สามารถใช้ภาษาใดก็ได้
  - เซิร์ฟเวอร์สามารถใช้ภาษาใดก็ได้เช่นกัน



คำตอบ

- **รูปแบบข้อมูลคืออะไร?**
  - แอพพลิเคชัน REST สามารถใช้ข้อมูลรูปแบบใดก็ได้
  - โดยทั่วไปจะเห็น XML และ JSON

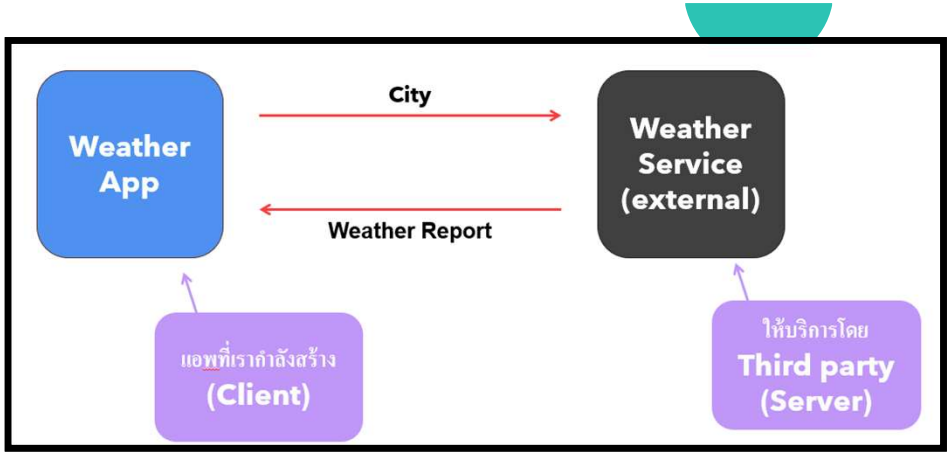
  - JSON เป็นที่นิยมมากที่สุดในปัจจุบัน
    - JavaScript Object Notation



วิธีการเรียกใช้ External Service

- ใช้ Weather Service API ออนไลน์ที่จัดทำโดย: openweathermap.org
- ให้ข้อมูลสภาพอากาศผ่านทาง API
- ข้อมูลมีหลายรูปแบบ: JSON, XML เป็นต้น …

การเรียก Weather Service

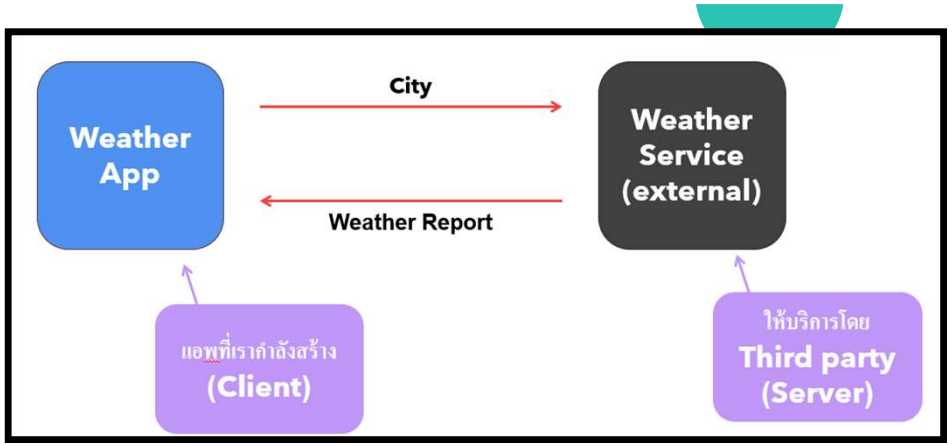


- จาก API Documentation เราสามารถเรียกใช้โดย:
  - ผ่านในชื่อเมือง

```
api.openweathermap.org/data/2.5/weather?q={city name}

OR

api.openweathermap.org/data/2.5/weather?q={city name},{country code}
```

รูปแบบของ Response

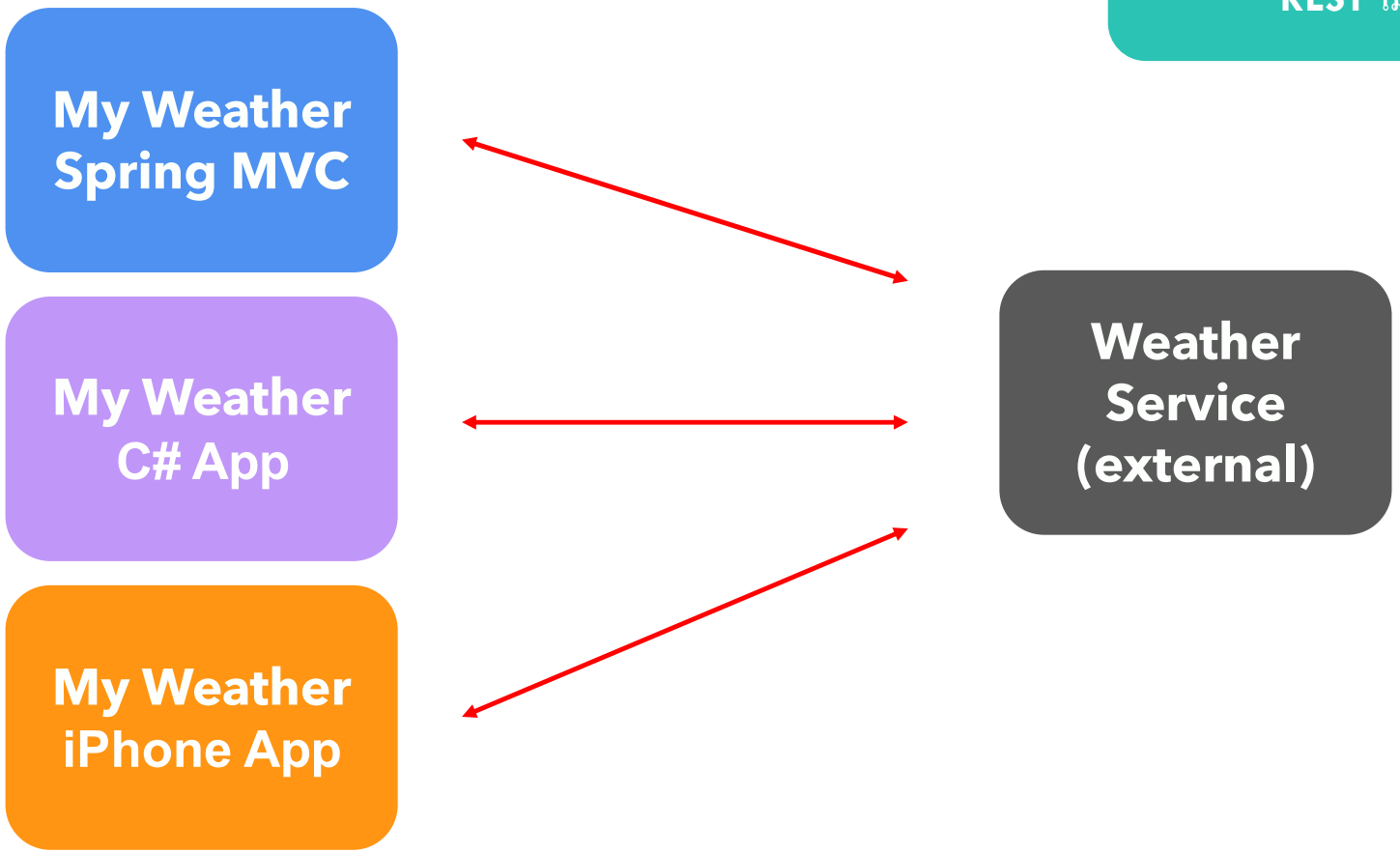
- Weather Service ตอบกลับด้วย JSON



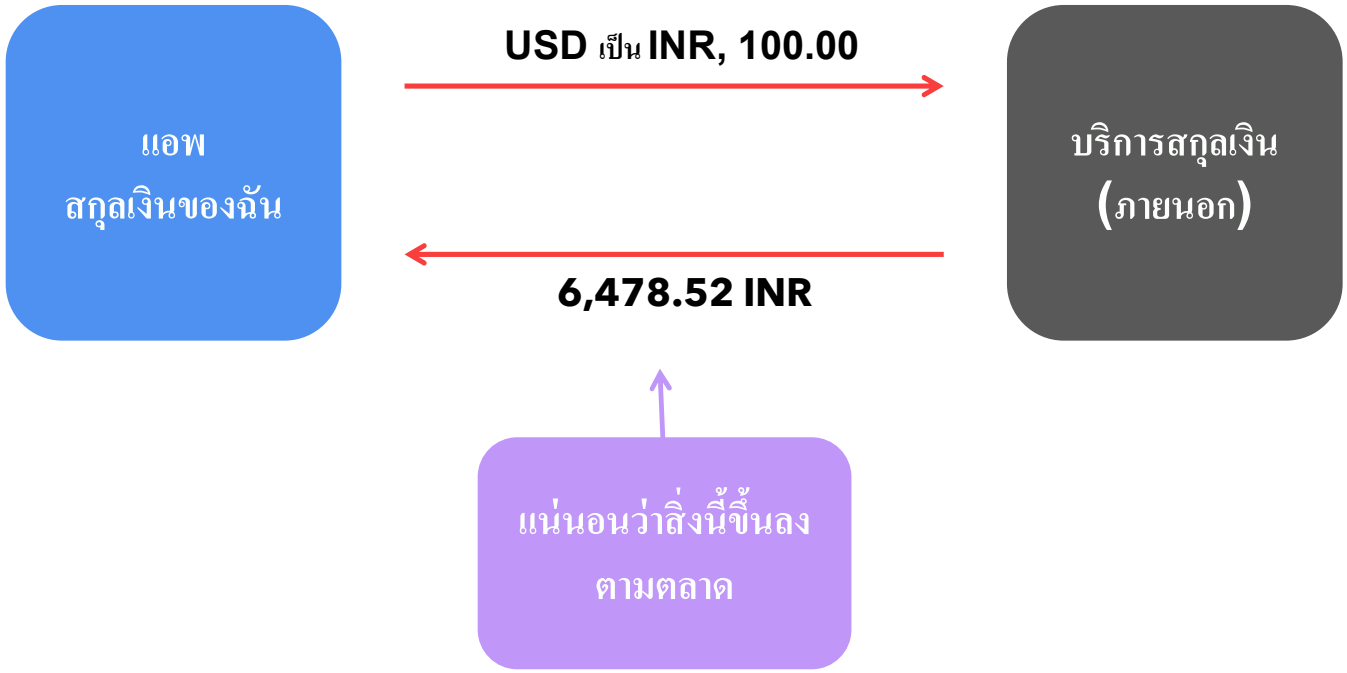

```
{
    "temp":14,
    "temp_min":11,
    "temp_max":17,
    "humidity":81,
    "name":"London"
}
```

App จะเขียนด้วยภาษาใดก็ได้

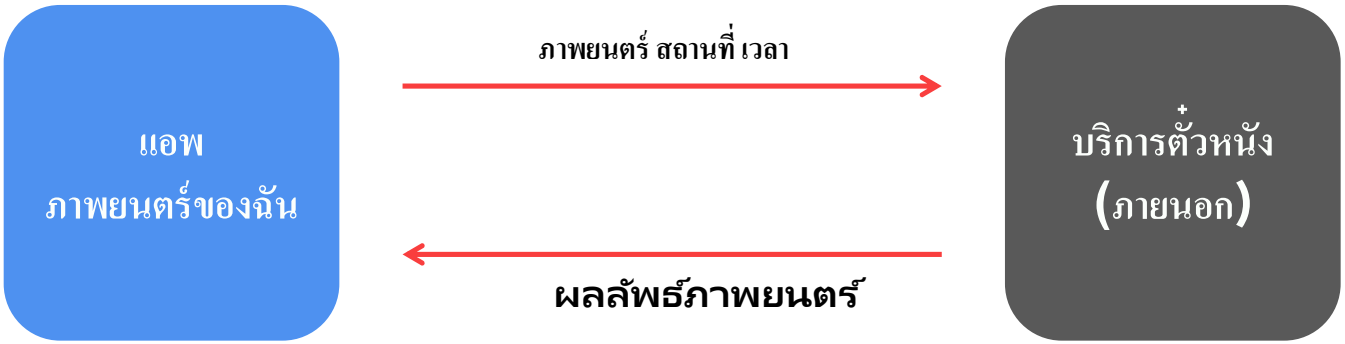
จำไว้ว่า:
การเรียก **REST** สามารถทำได้ผ่าน **HTTP**
REST ไม่ขึ้นกับภาษา



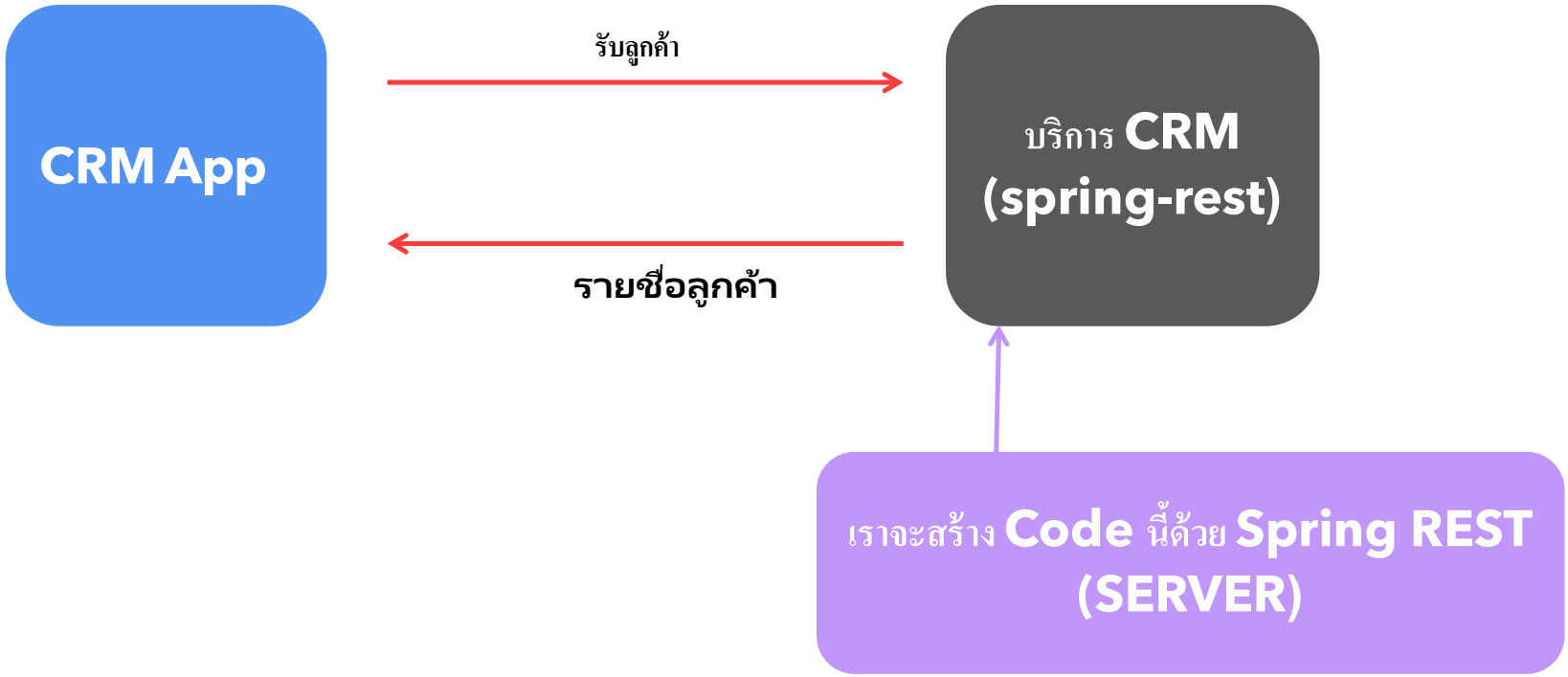
ตัวอย่าง - แอพแปลงสกุลเงิน



ตัวอย่าง - แอพตั๋วหนัง



ตัวอย่าง - แอพผู้จัดการฝ่ายลูกค้าสัมพันธ์ (CRM)



เราเรียกสิ่งนี้ว่าอย่างไร?

REST API

REST
Web Services

REST Services

RESTful API

RESTful
Web Services

RESTful Services

โดยทั่วไปแล้ว ทั้งหมดมีความหมายเหมือนกัน

เราเรียกสิ่งนี้ว่าอย่างไร?

REST API

REST
Web Services

REST Services

RESTful API

RESTful
Web Services

RESTful Services

โดยทั่วไปแล้ว ทั้งหมดมีความหมายเหมือนกัน

พื้นฐาน
JSON - Syntax

Chapter 2

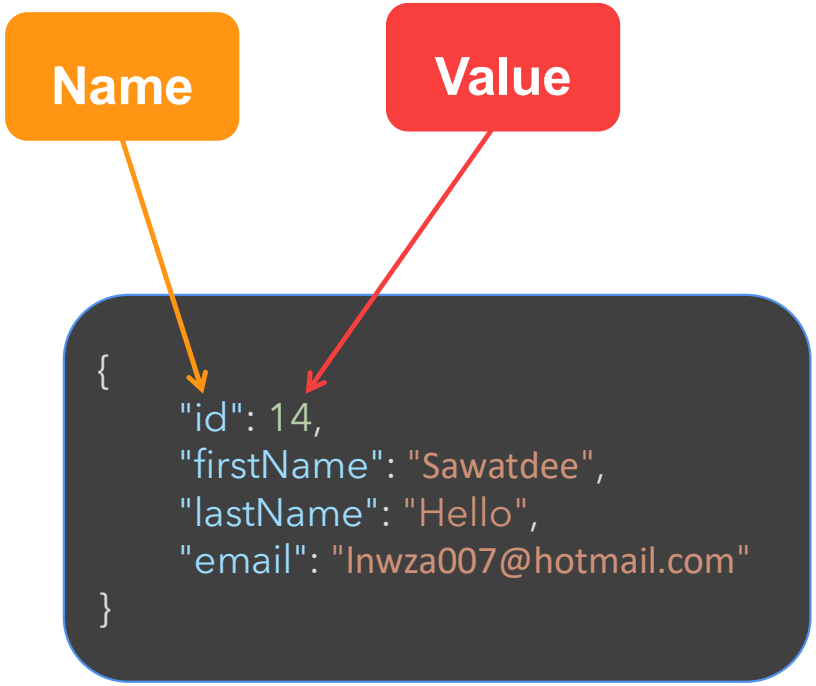
JSON คืออะไร

- ย่อมาจาก **J**ava**S**cript **O**bject **N**otation
- รูปแบบข้อมูลน้ำหนักเบาสำหรับการจัดเก็บและแลกเปลี่ยนข้อมูล
- ไม่ขึ้นกับภาษา ไม่ใช่แค่สำหรับ JavaScript
- ใช้กับโปรแกรมภาษาใดก็ได้: Java, C#, Python เป็นต้น …

JSON เป็นเพียง plain text

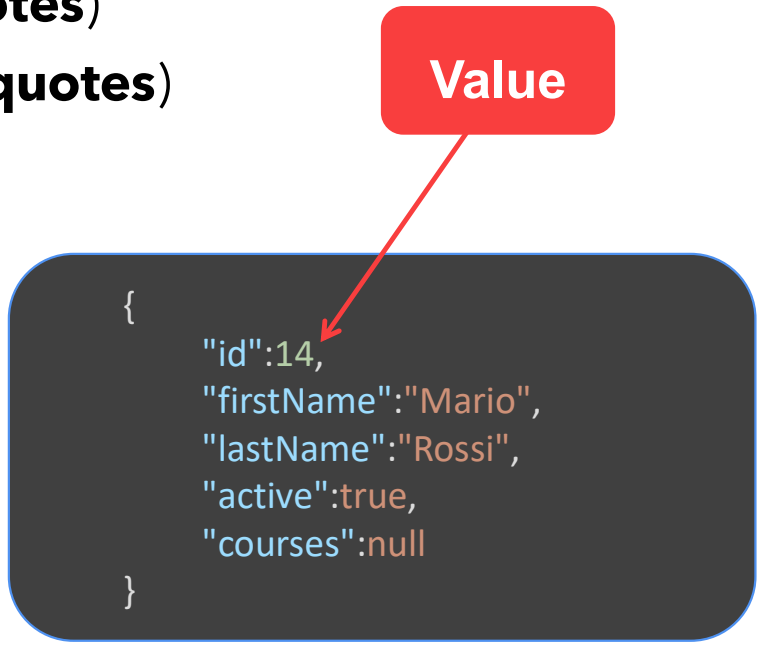
ตัวอย่าง JSON เบื้องต้น

- เครื่องหมายปีกกาเป็นการประกาศ object ใน JSON
- สมาชิกของ object จะเป็นเป็นคู่ name / value
  - คั่นด้วยเครื่องหมายทวิภาค " : " (**colons**)
- name ต้องอยู่ในเครื่องหมายอัญประกาศ " " (**double-quotes**) เสมอ

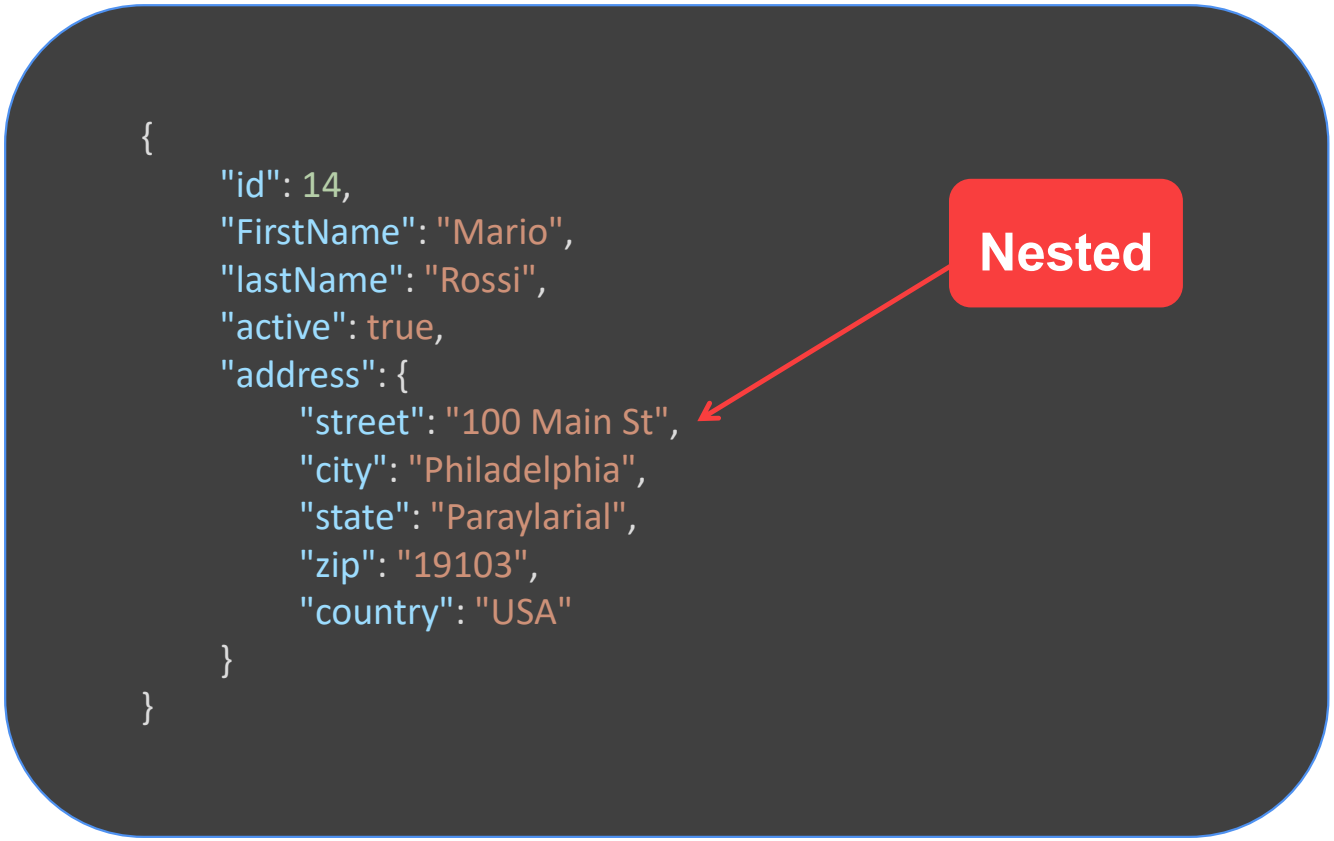


JSON Values

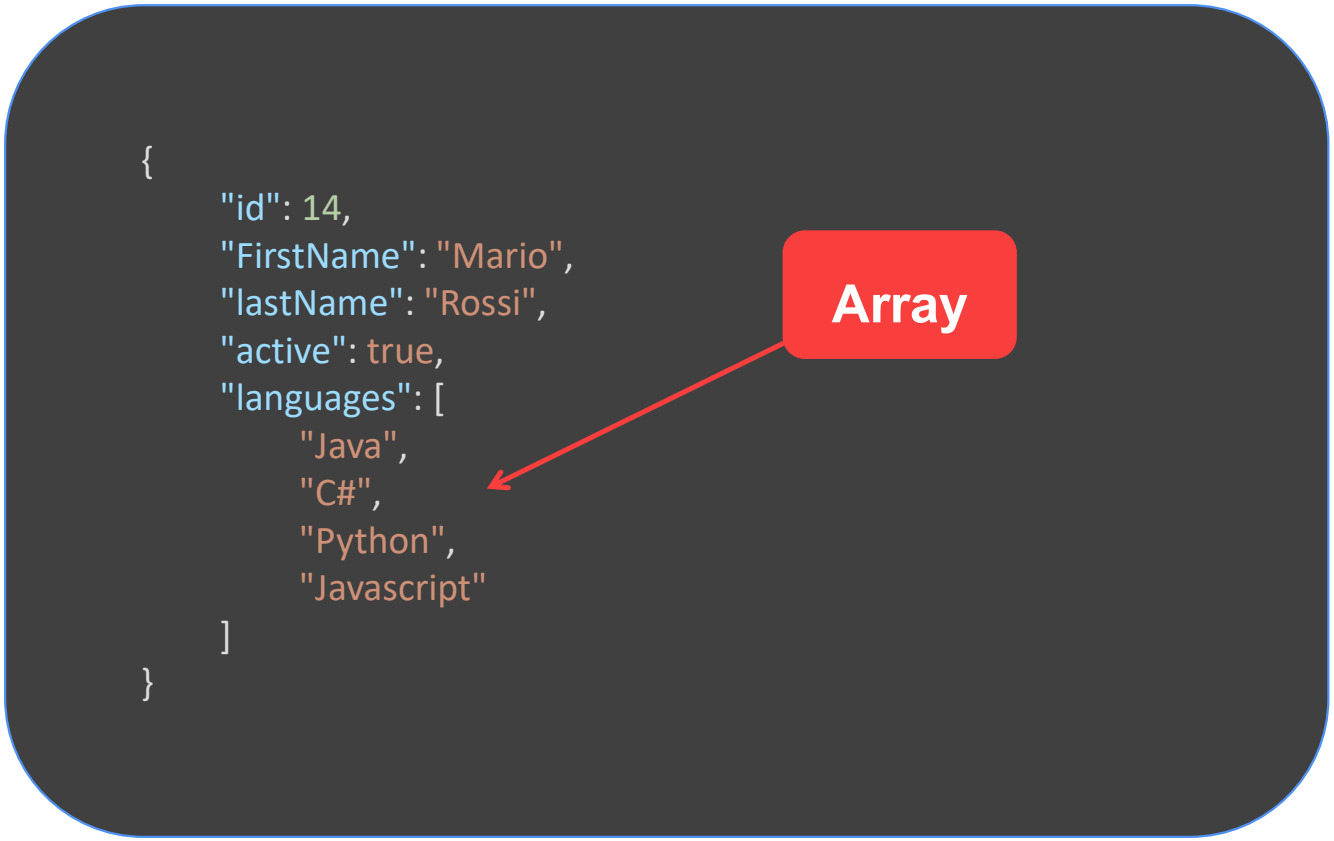
- ตัวเลข Number : ไม่มีเครื่องหมายอัญประกาศ (**double-quotes**)
- ตัวหนังสือ String : ต้องอยู่ในเครื่องหมายอัญประกาศ (**double-quotes**)
- Boolean: **true**, **false**
- Nested JSON object
- Array
- null



Nested JSON Object



JSON Array


{
"id": 14,
"FirstName": "Mario",
"lastName": "Rossi",
"active": true,
"languages": [
"Java",
"C#",
"Python",
"Javascript"
]
}
Array

พื้นฐาน
REST HTTP

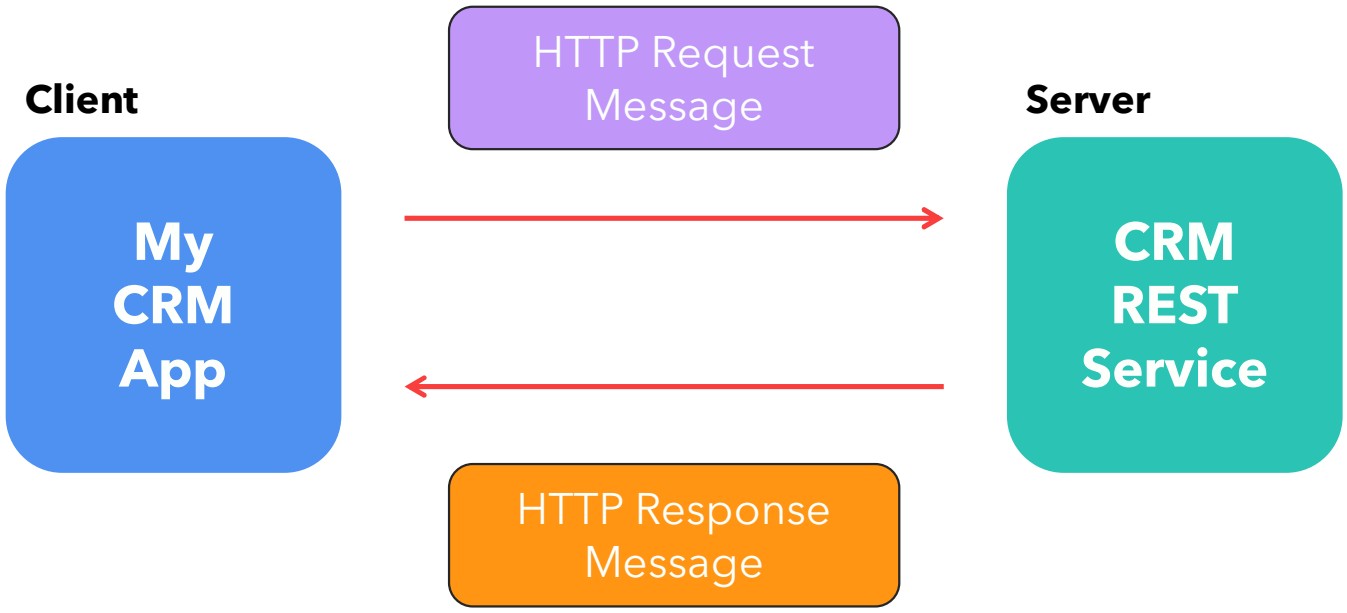
Chapter 3

REST over HTTP

- การใช้ REST จะใช้บ่อยที่สุดคือ การใช้ผ่าน HTTP
- ใช้ HTTP methods สำหรับการดำเนินการ CRUD

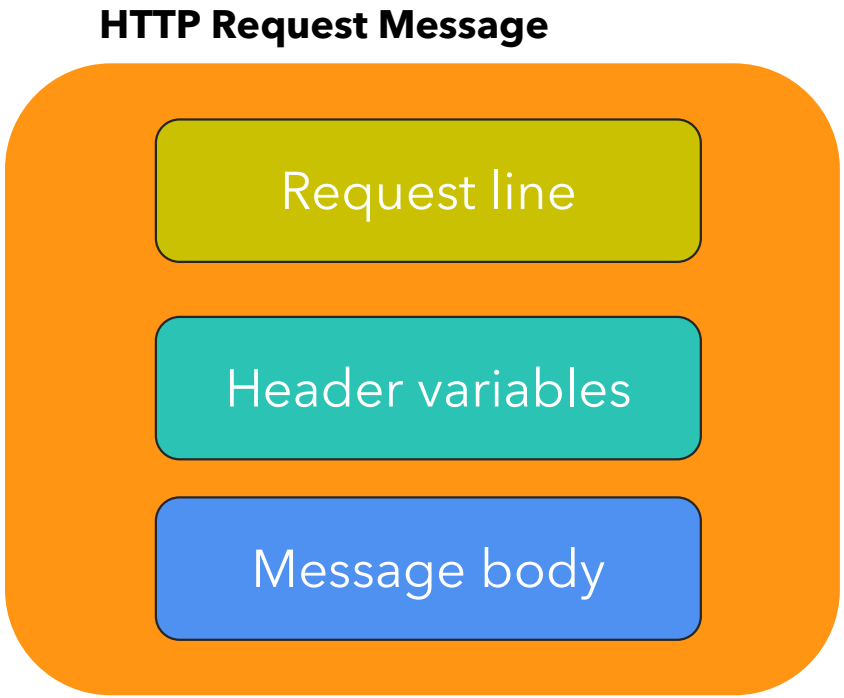
HTTP Method	CRUD Operation
POST	สร้าง **entity** ใหม่ (Create)
GET	อ่านรายชื่อ **entities** หรือ **entity** เดียว (Read)
PUT	แก้ไขข้อมูล **entity** (Update)
DELETE	ลบข้อมูล entity (Delete)

HTTP Message



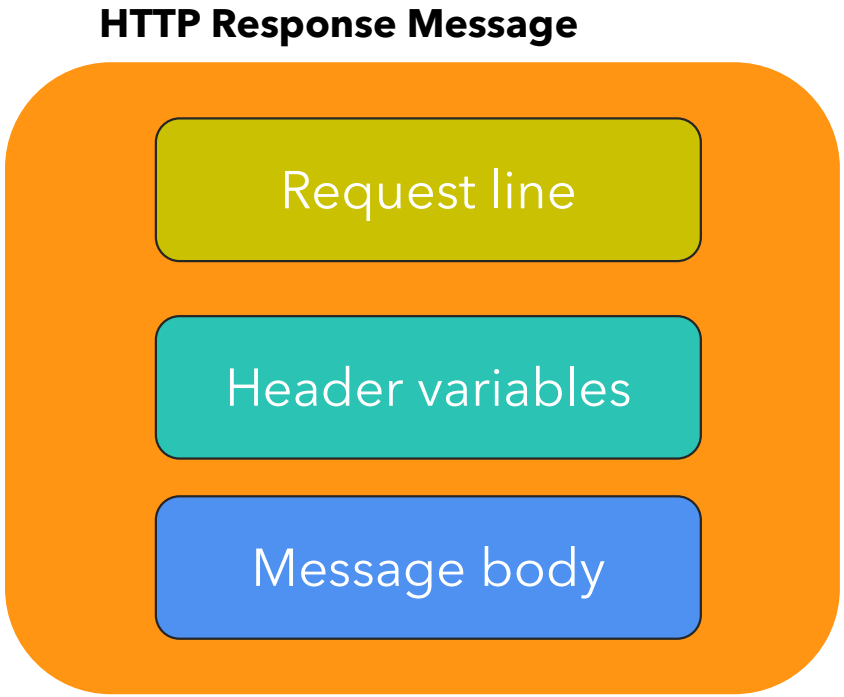
HTTP Request Message

- Request line: HTTP Command
- Header variables: metadata
- Message body: contents



HTTP Response Message

- Response line: server protocol and status code
- Header variables: metadata
- Message body: Contents



HTTP Response – Status Codes

ช่วงรหัส	คำอธิบาย
100 - 199	ให้ข้อมูล (Informational)
200 – 299	สำเร็จ (Successful)
300 – 399	การเปลี่ยนเส้นทาง (Redirect)
400 - 499	Error ของฝั่ง client (Client error)
500 - 599	Error ของฝั่ง server (Server error)

401 จำเป็นต้องมีการตรวจสอบสิทธิ์
404 ไม่พบไฟล์

500 ข้อผิดพลาด
ภายใน server

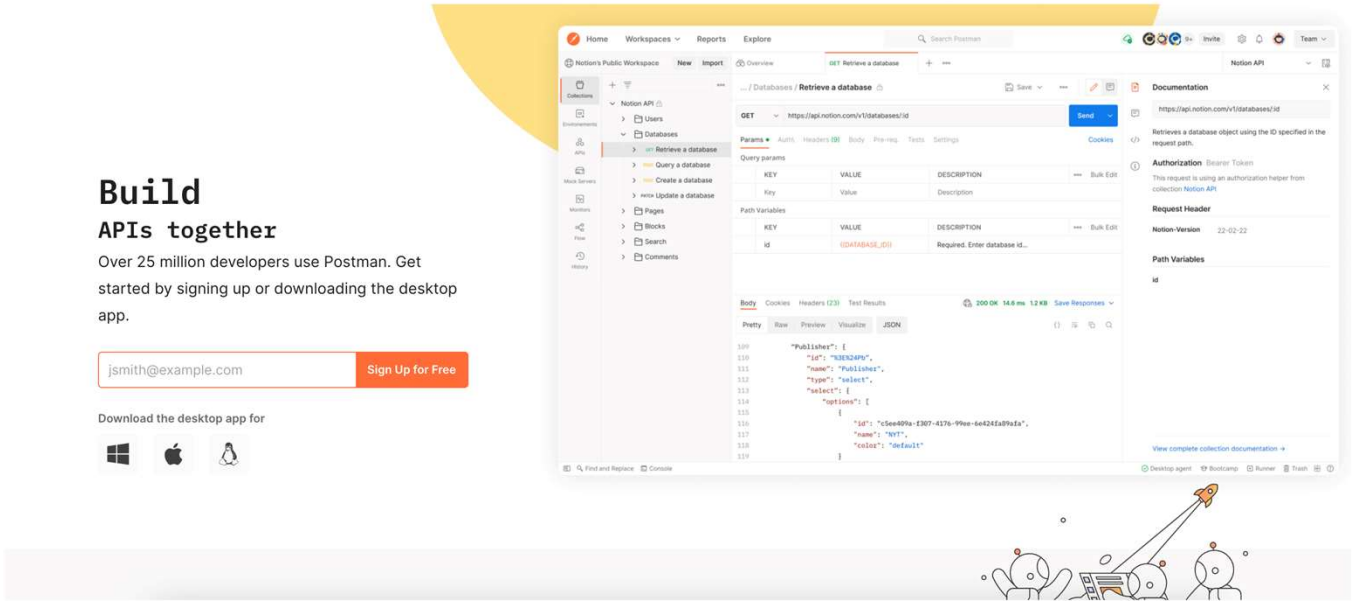
เนื้อหาประเภท MIME

- รูปแบบข้อความอธิบายตามประเภทเนื้อหา MIME
  - **M**ultipurpose **I**nternet **M**ail-**E**xtension
- Basic Syntax : Type/Sub-type
- ตัวอย่าง
  - text/html, text/plain
  - application/json, application/xml, ...

Client Tool

- เราต้องการ **client tool** ในการคุยกับ API
- ส่งคำขอ HTTP ไปยัง REST Web Service / API
- มีเครื่องมือให้ใช้มากมาย เช่น curl, Postman ฯลฯ ...

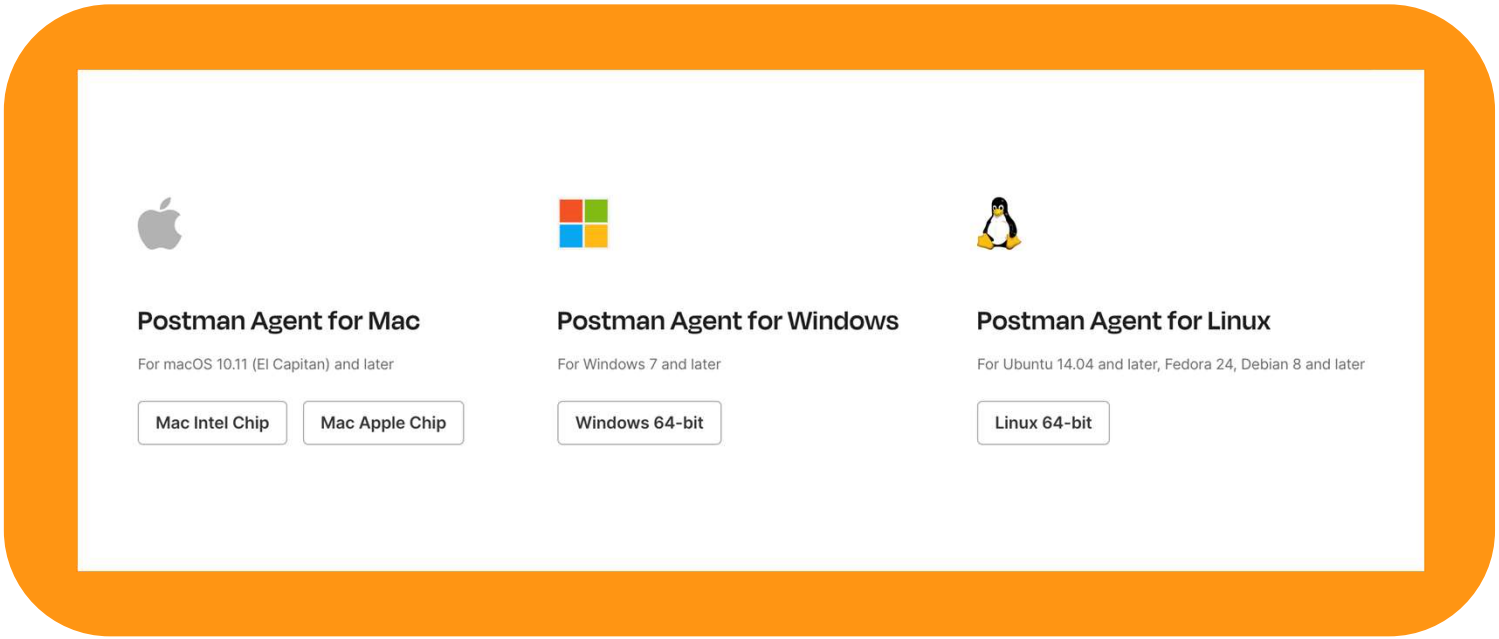
Postman



www.postman.com

การติดตั้ง Postman

www.postman.com



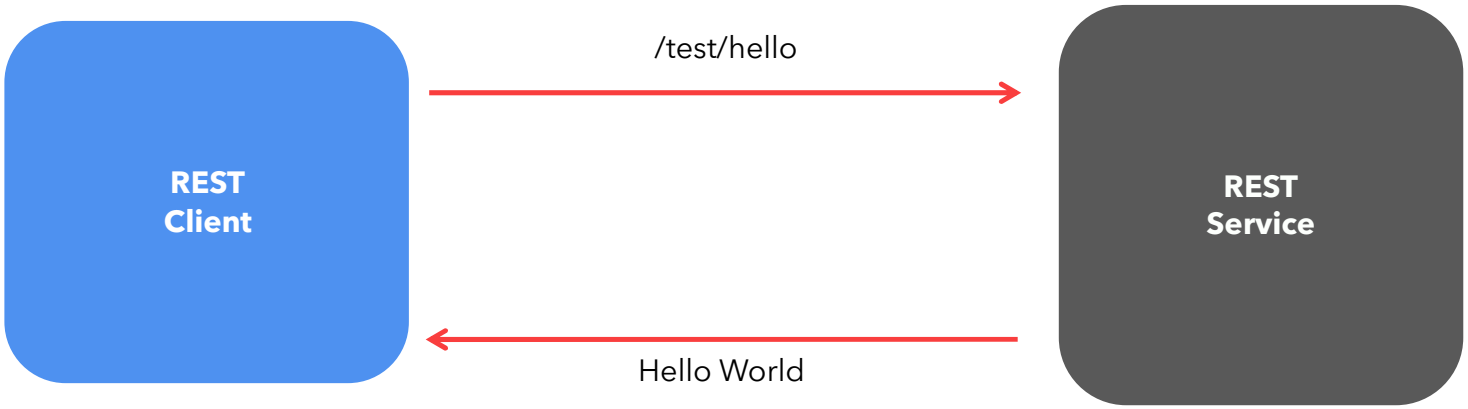
ลองใช้ POSTMAN

- Mini-workshop: ลองใช้ Postman

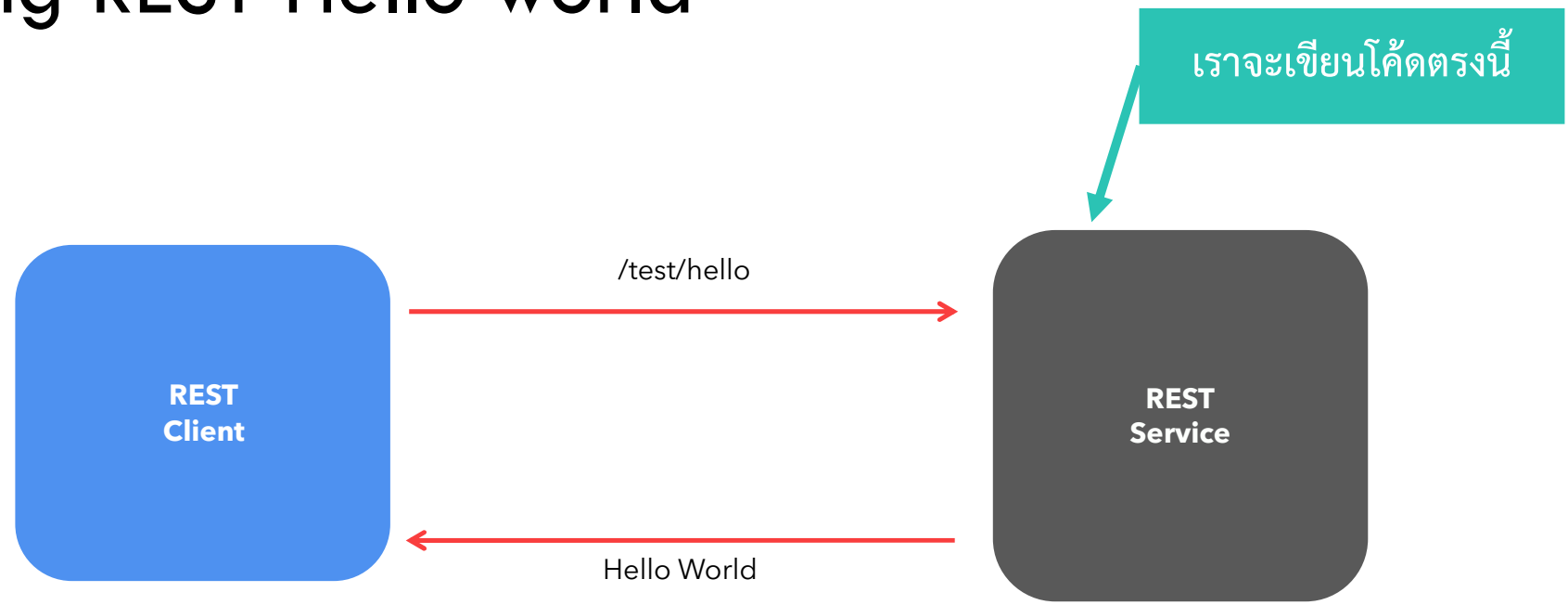
- Ref: lab-s5-l0


Spring Boot REST Controller
Chapter 4

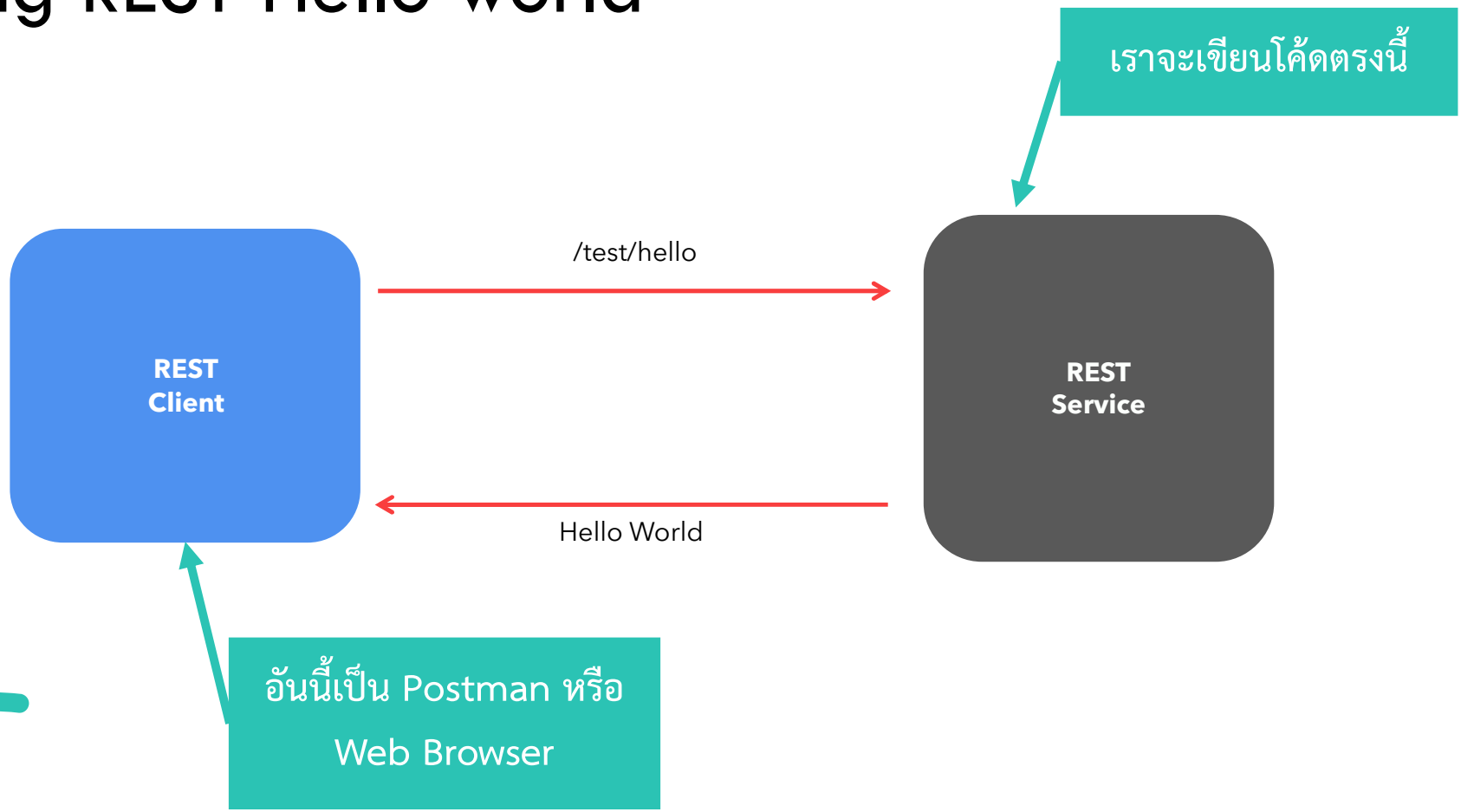
Spring REST Hello world



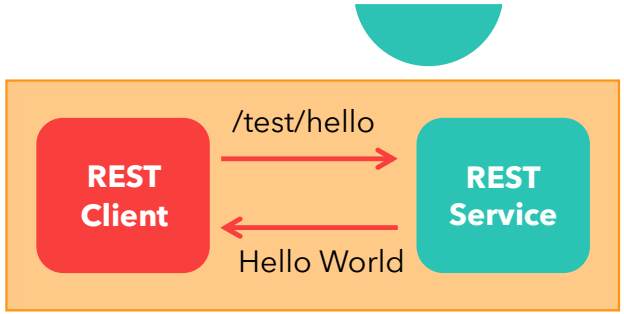
Spring REST Hello world



Spring REST Hello world



Spring REST Controller



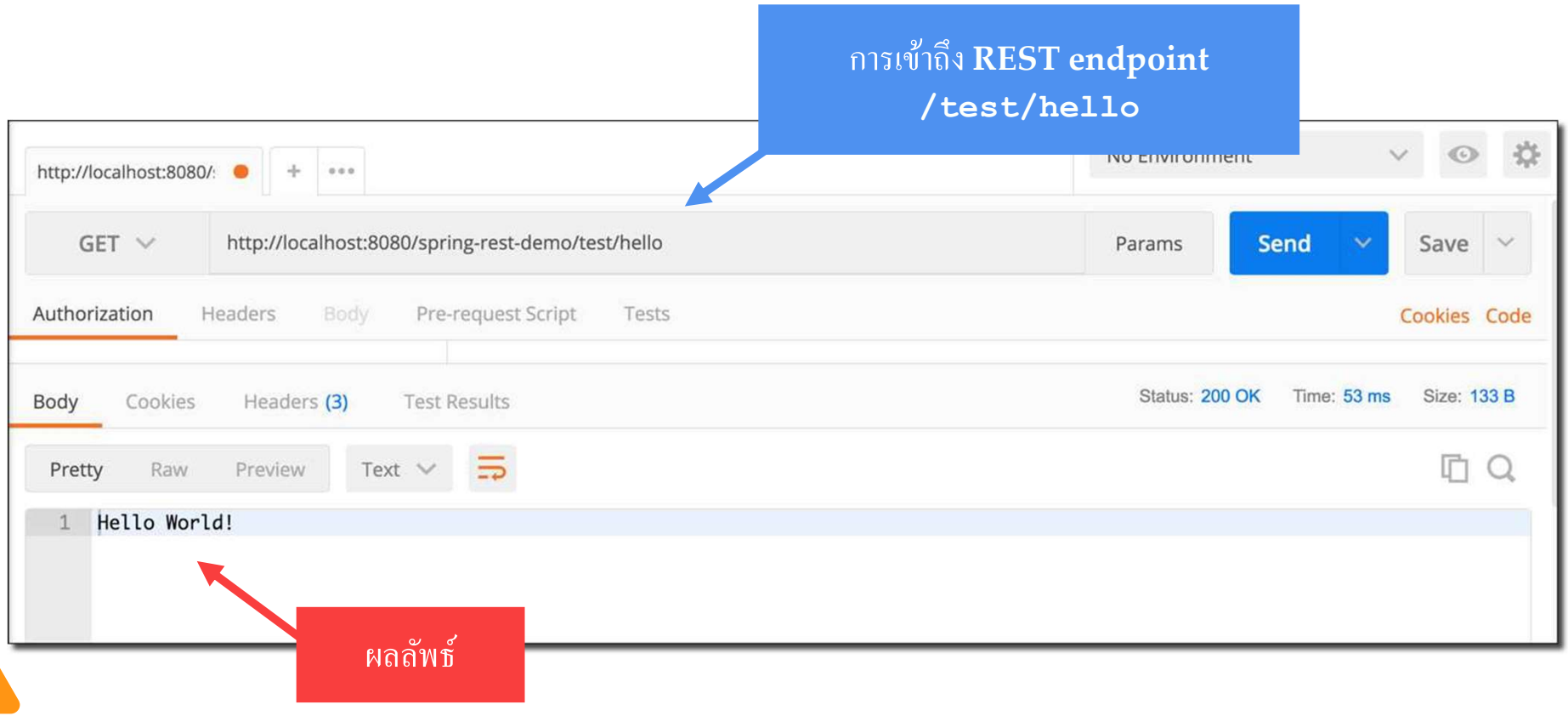
เพิ่ม REST support

```
@RestController
@RequestMapping("/test")
public class DemoRestControl

    @GetMapping('/hello')
    public String sayHello() {
        return "Hello World!";
    }
}
```

เข้าถึง REST endpoint
/test/hello

Return content
ไปให้ Client

Testing with REST Client - Postman



Testing with REST Client – Web Browser

Web Browser vs Postman

- สำหรับการทดสอบ REST แบบง่ายสำหรับการ GET requests
  - Web Browser and Postman มีความคล้ายกัน

- อย่างไรก็ตาม สำหรับการทดสอบ REST ขั้นสูง: POST, PUT และอื่น ๆ ...
  - Postman มีการสนับสนุนที่ดีกว่า
  - การ POSTing ข้อมูล JSON, content type
  - สามารถกำหนด HTTP request headers , Authentication ฯลฯ ...

POSTMAN

วิธีการทำ

ทีละขั้นตอน

1. เพิ่ม Spring Boot Starter Web เป็น Maven dependency

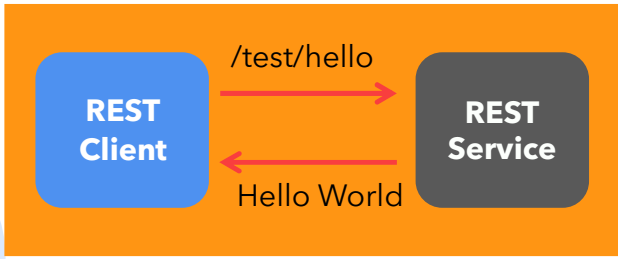
2. สร้าง Spring REST Service โดยการใช้ @RestController

ขั้นตอนแรก: เพิ่ม Marven Dependency

```xml
<!-- Add Spring Boot Starter Web -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

หรือในเว็บไซต์ **Spring Initializr**
สามารถเลือก **“Web” dependency** ได้

ขั้นตอนสอง: สร้าง Spring REST Service




```java
@RestController
@RequestMapping ("/test")
public class DemoRestController {

    @GetMapping ("/hello")
    public String sayHello() {
        return "Hello World!";
    }

}
```

จัดการกับ **HTTP GET requests**

สร้าง Spring REST Service

- Mini-workshop: สร้าง Spring REST Service

- Ref: lab-s5-l1

การเชื่อมโยงข้อมูล

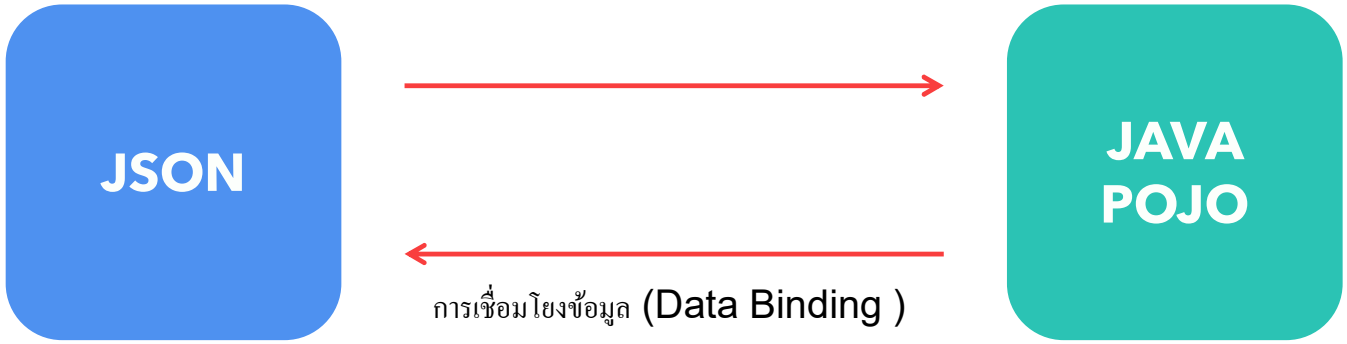
Java JSON

(Java JSON Data Binding)

Chapter 5

Java JSON Data Binding

- Data binding เป็นกระบวนการของการแปลงข้อมูล JSON เป็น Java POJO



เรียกอีกอย่างว่า

Mapping
Serialization / Deserialization
Marshalling / Unmarshalling

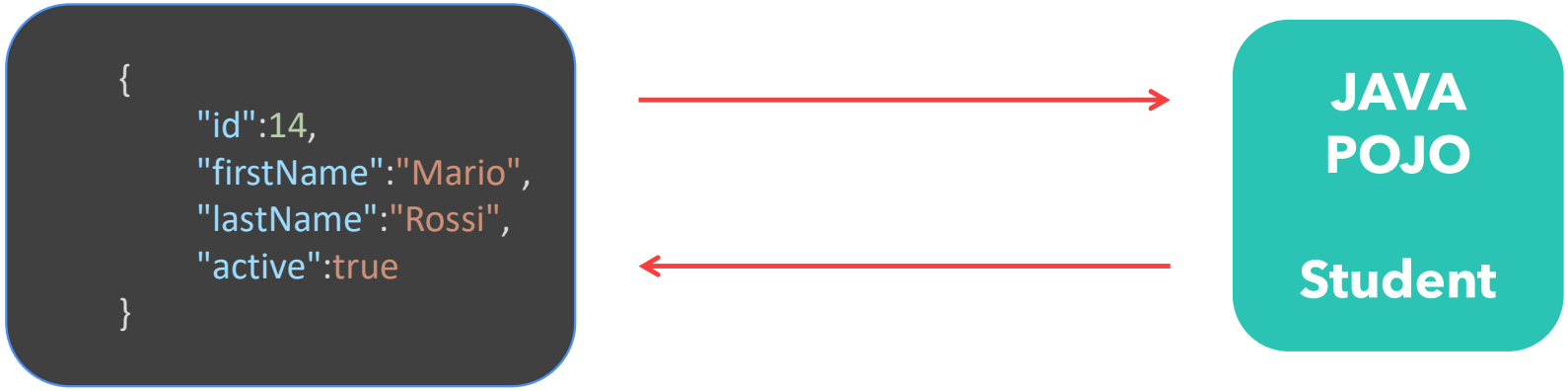
JSON Data Binding โดย Jackson

- Spring ใช้ Jackson Project ทำงานอยู่เบื้องหลัง
- Jackson จัดการการ data binding ระหว่าง JSON และ Java POJO
- รายละเอียดเกี่ยวกับโปรเจกต์ Jackson

https://github.com/FasterXML/jackson-databind

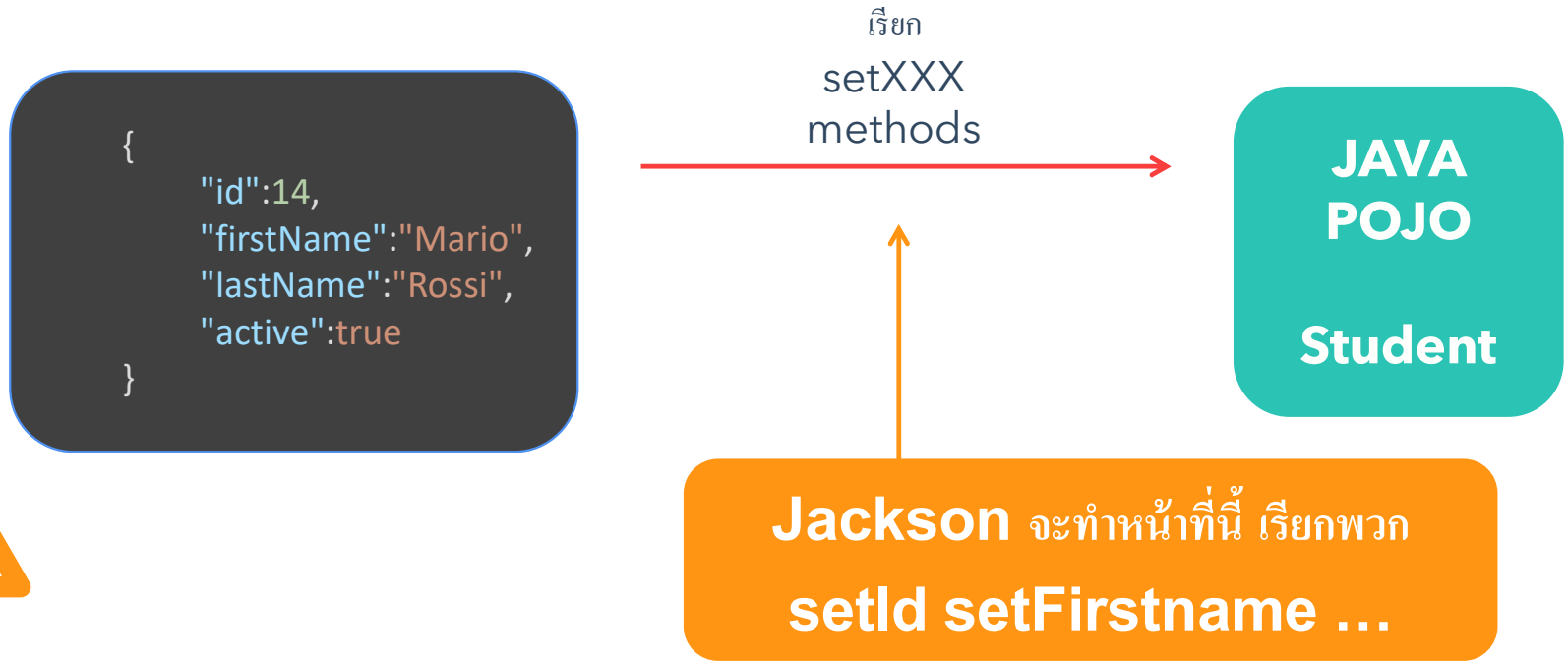
JSON Data Binding โดย Jackson

- โดยค่าเริ่มต้น Jackson จะเรียกใช้เมธอด getter/setter ที่เหมาะสม



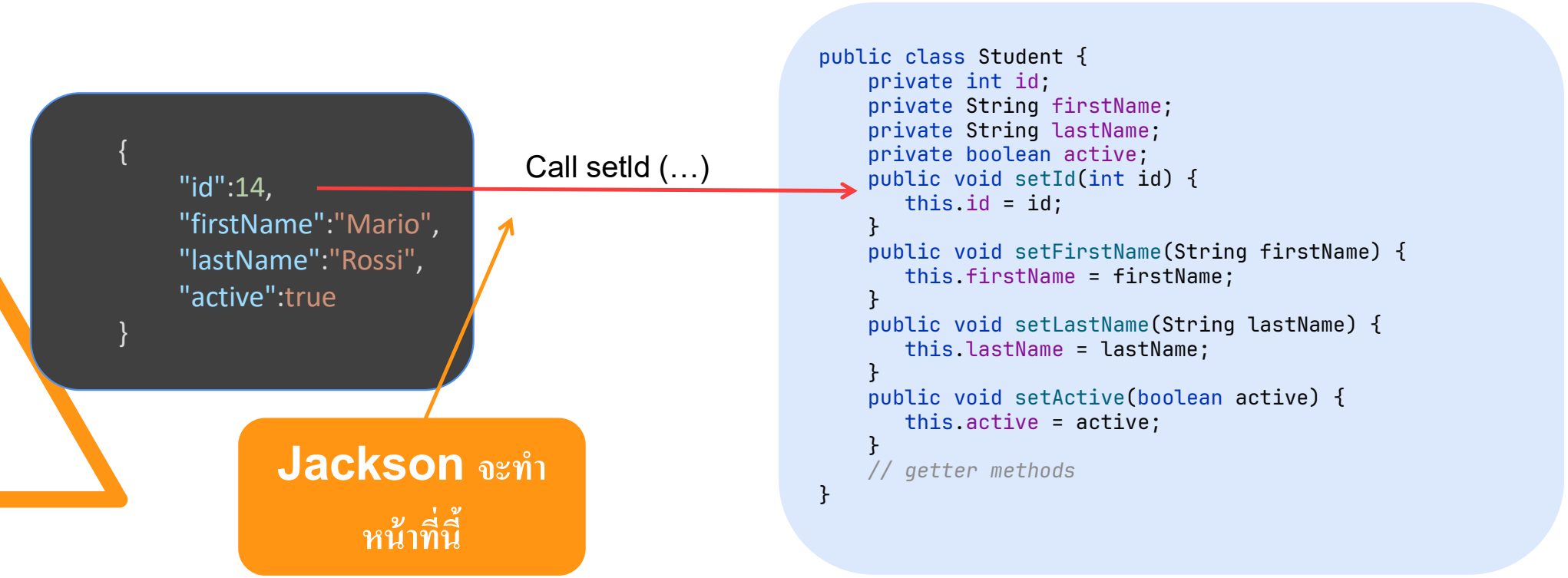
JSON Data Binding โดย Jackson

- โดยค่าเริ่มต้น Jackson จะเรียกใช้เมธอด getter/setter ที่เหมาะสม



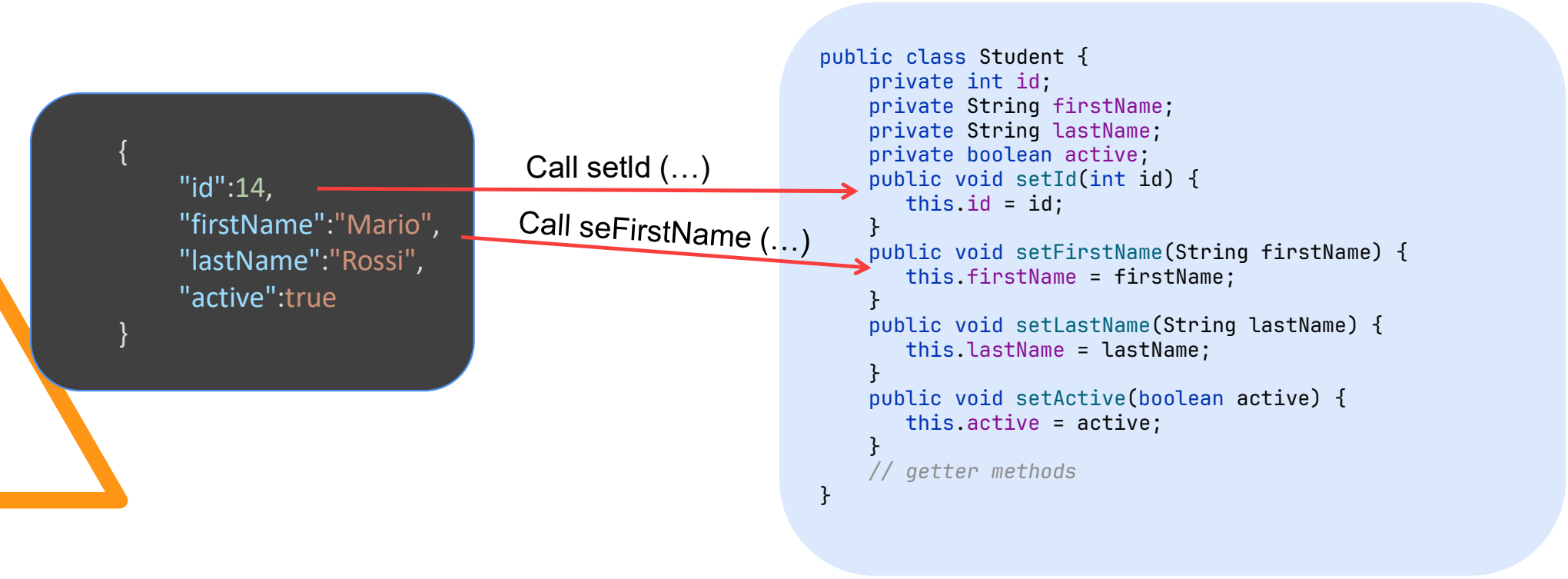
JSON Data Binding โดย Jackson

- โดยค่าเริ่มต้น Jackson จะเรียกใช้เมธอด getter/setter ที่เหมาะสม



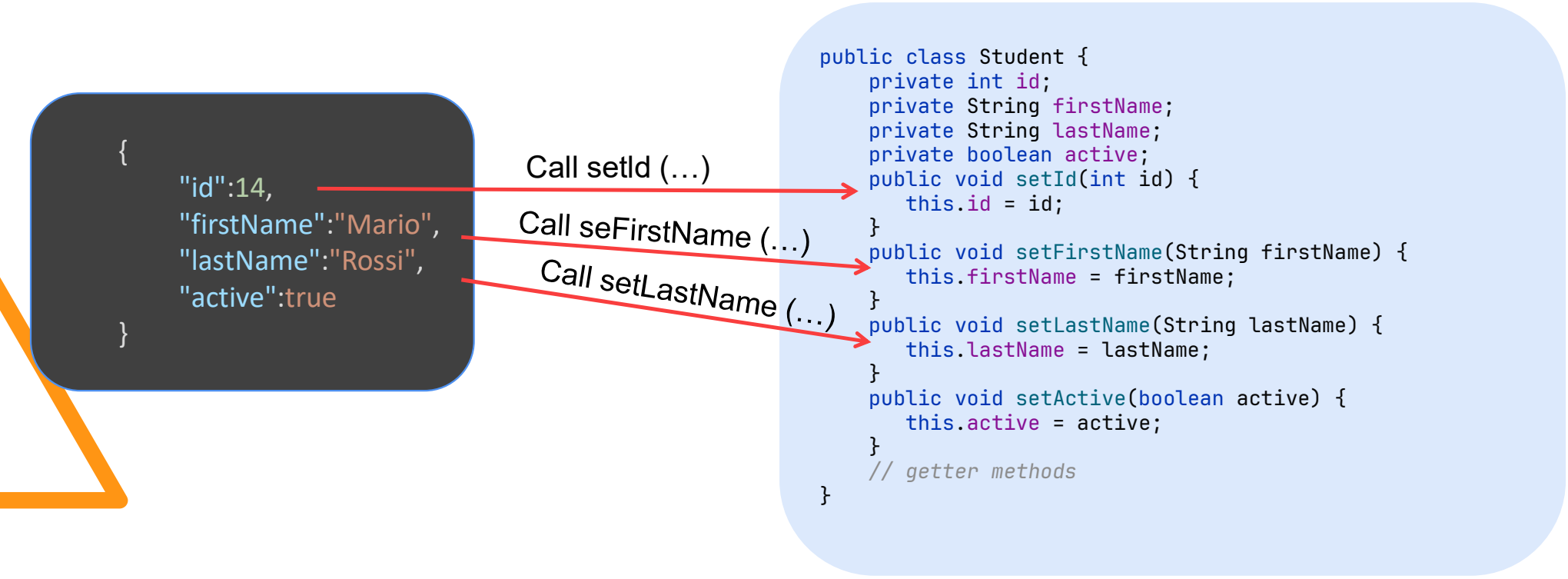
JSON Data Binding โดย Jackson

- โดยค่าเริ่มต้น Jackson จะเรียกใช้เมธอด getter/setter ที่เหมาะสม



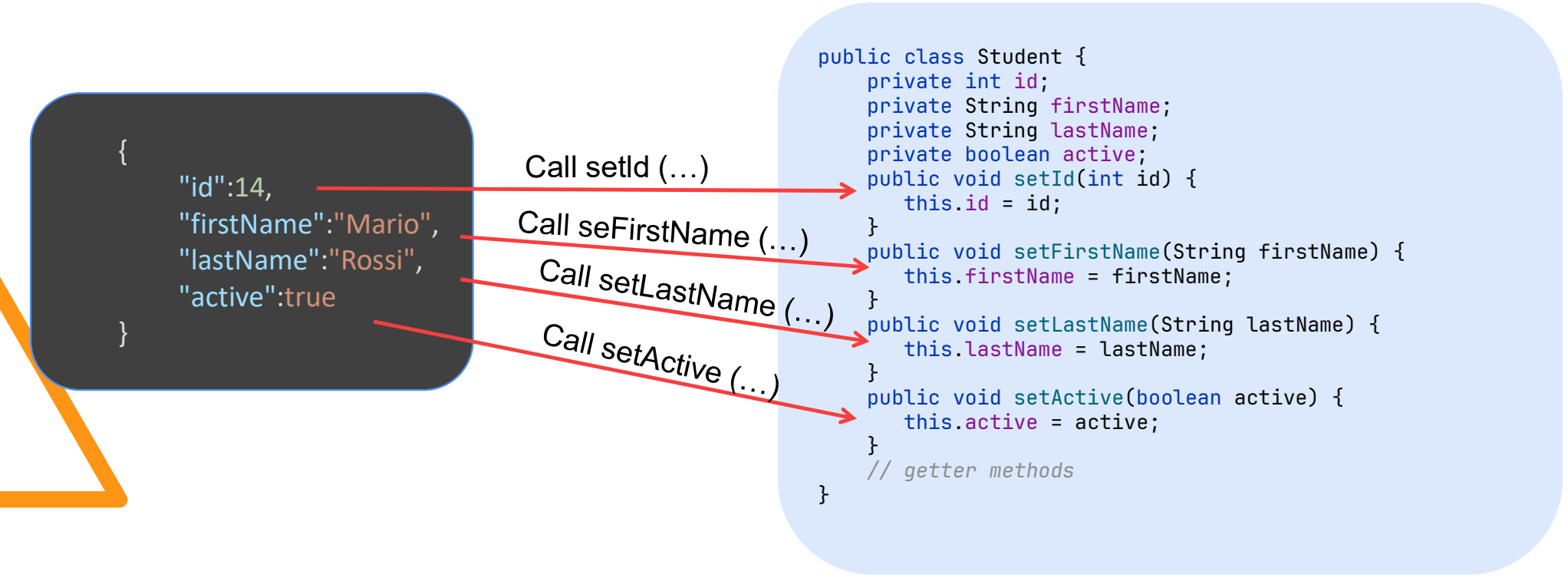
JSON Data Binding โดย Jackson

- โดยค่าเริ่มต้น Jackson จะเรียกใช้เมธอด getter/setter ที่เหมาะสม



JSON Data Binding โดย Jackson

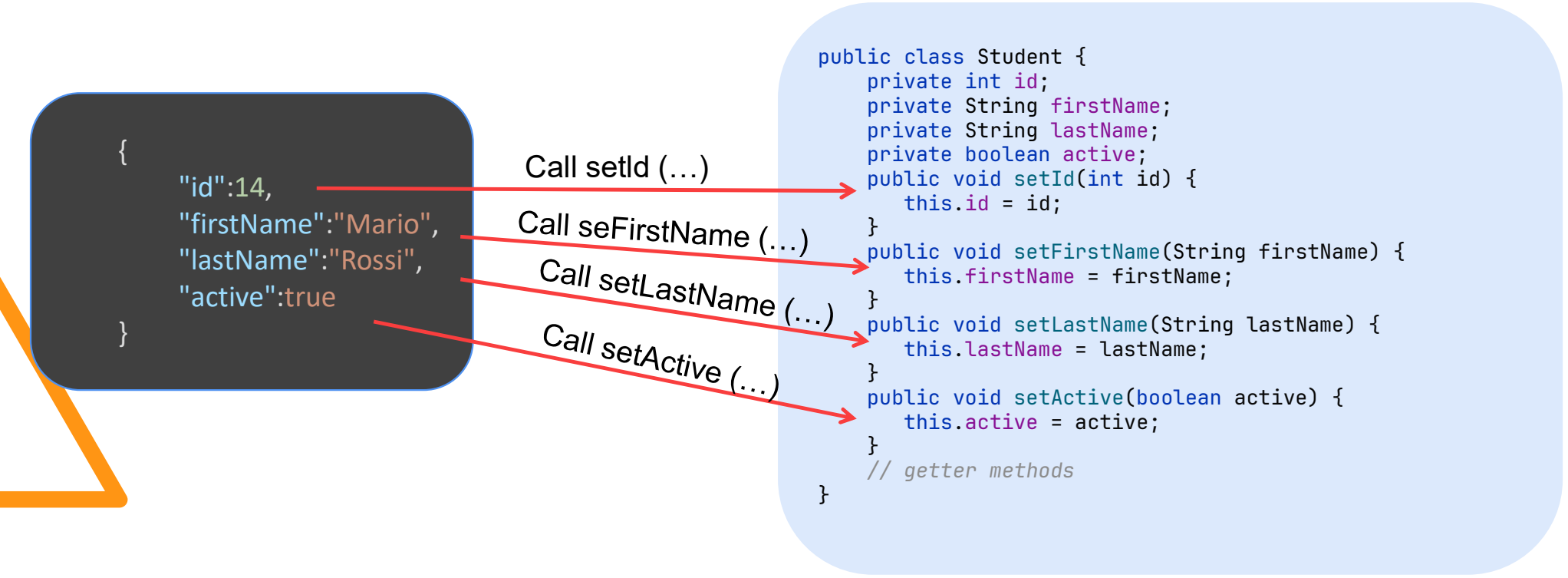
- โดยค่าเริ่มต้น Jackson จะเรียกใช้เมธอด getter/setter ที่เหมาะสม



JSON Data Binding โดย Jackson

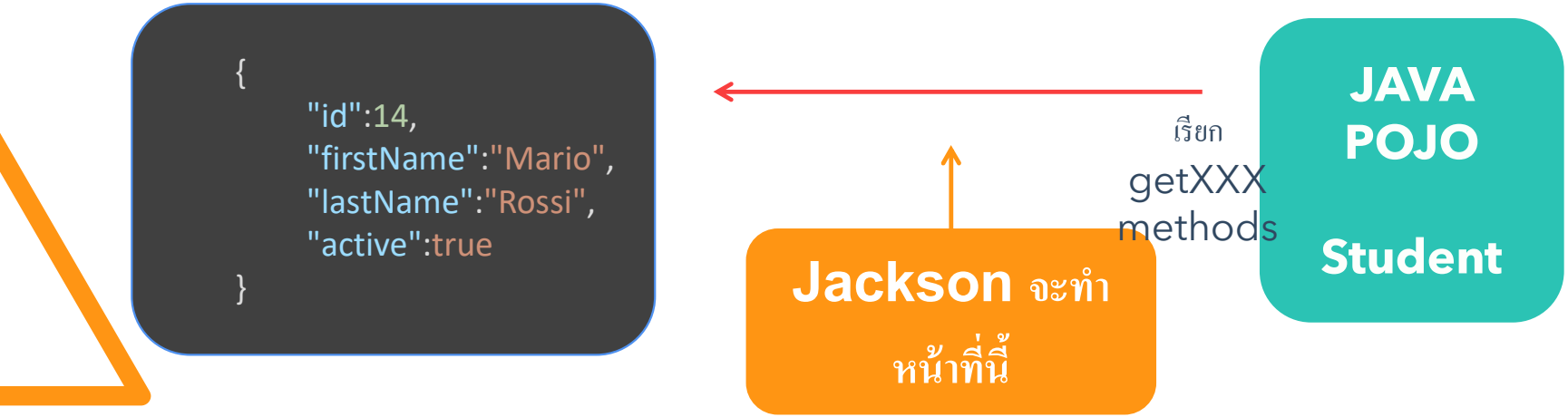
หมายเหตุ: Jackson จะเรียกใช้เมธอด **setXXX** เท่านั้น จะไม่เข้าถึง **private field** โดยตรง

- โดยค่าเริ่มต้น Jackson จะเรียกใช้เมธอด getter/setter ที่เหมาะสม



JSON Data Binding โดย Jackson

- ในทางกลับกัน ถ้าเราจะแปลง POJO เป็น JSON
- โดยค่าเริ่มต้น Jackson จะเรียกใช้เมธอด getter/setter ที่เหมาะสม



Spring และ Jackson การสนับสนุน

เป็นการทำงานเบื้องหลังที่เกิดขึ้นโดยอัตโนมัติ

- เมื่อสร้างแอพพลิเคชั่น Spring REST
- Spring จะมีการรวม Jackson มาใน App โดยอัตโนมัติ
- ข้อมูล JSON ที่ส่งผ่านไปยัง REST controller จะถูกแปลงเป็น POJO
- object Java ที่ถูกส่งคืนจาก REST Controller จะถูกแปลงเป็น JSON

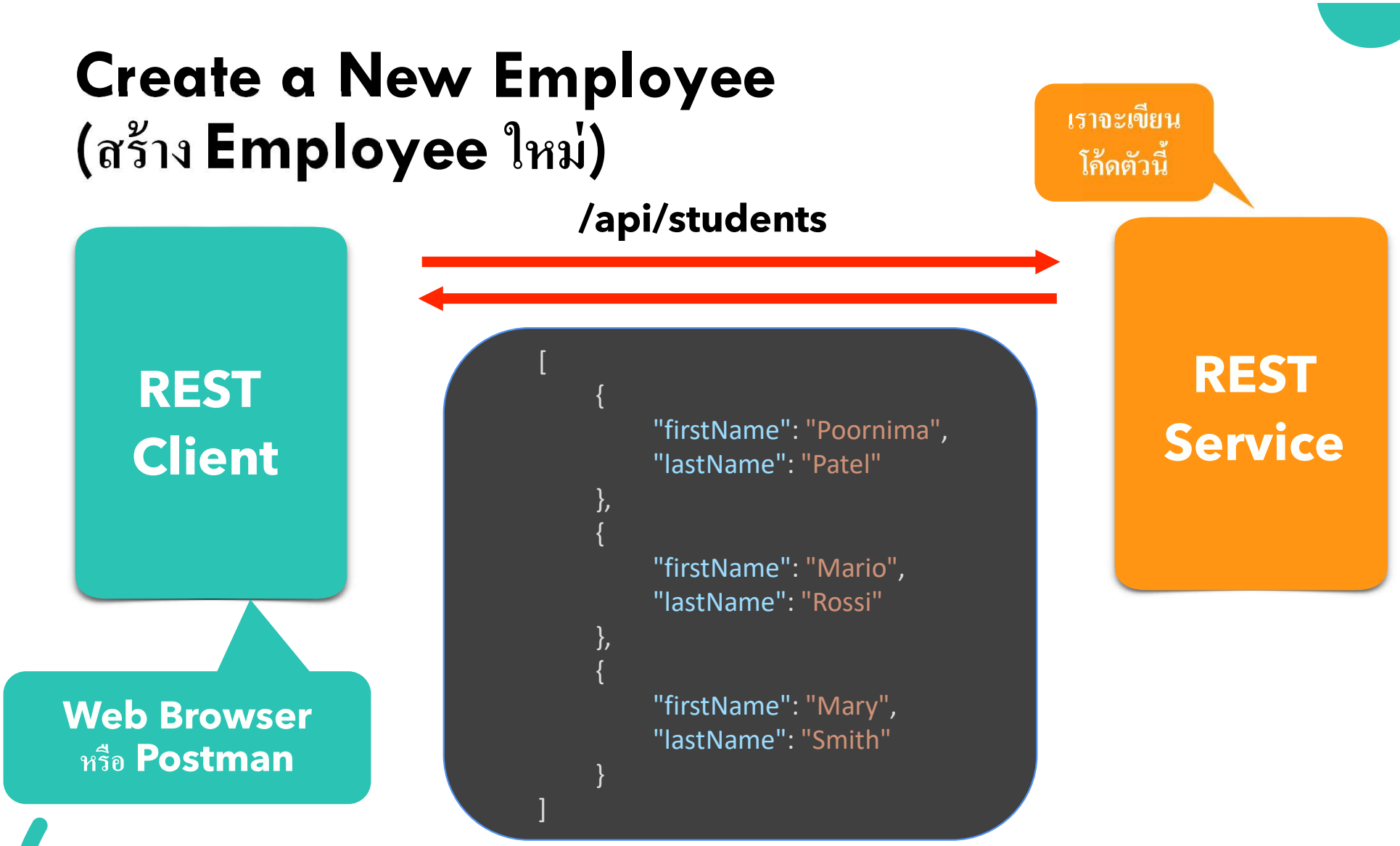

Spring REST
Service - Students
Chapter 6

สร้าง Service ใหม่

- ส่งคืนรายชื่อนักเรียนทั้งหมด

GET	/api/students	Return a list of stidents

Create a New Employee
(สร้าง Employee ใหม่)



แปลง Java POJO เป็น JSON

- REST Service จะส่ง List<Student> ออกไป
- ต้องแปลง List<Student> เป็น JSON
- Jackson สามารถช่วยเราในเรื่องนี้

Spring และ Jackson การสนับสนุน

เกิดขึ้นโดยอัตโนมัติ

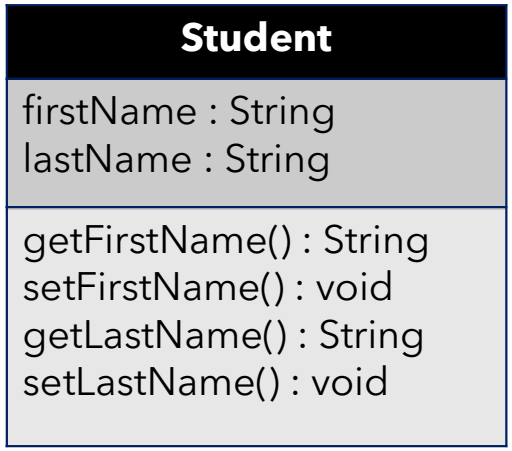
โดยการทำงานเบื้องหลัง

- Spring จะจัดการ Jackson Integration โดยอัตโนมัติ
- ข้อมูล JSON ที่เข้ามาใน REST controller จะถูกแปลงเป็น POJO
- object Java ที่ถูกส่งออกไปจาก REST Controller จะถูกแปลงเป็น JSON

Student POJO (class)

Java POJO

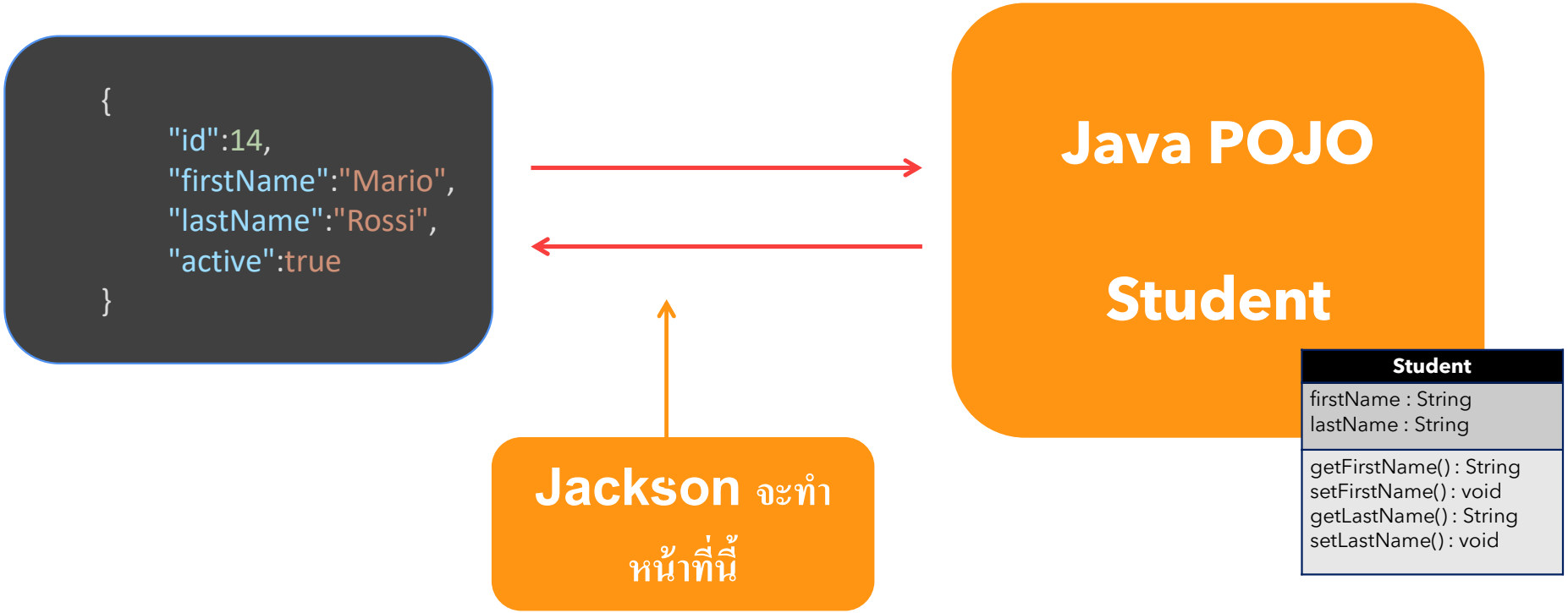
Student



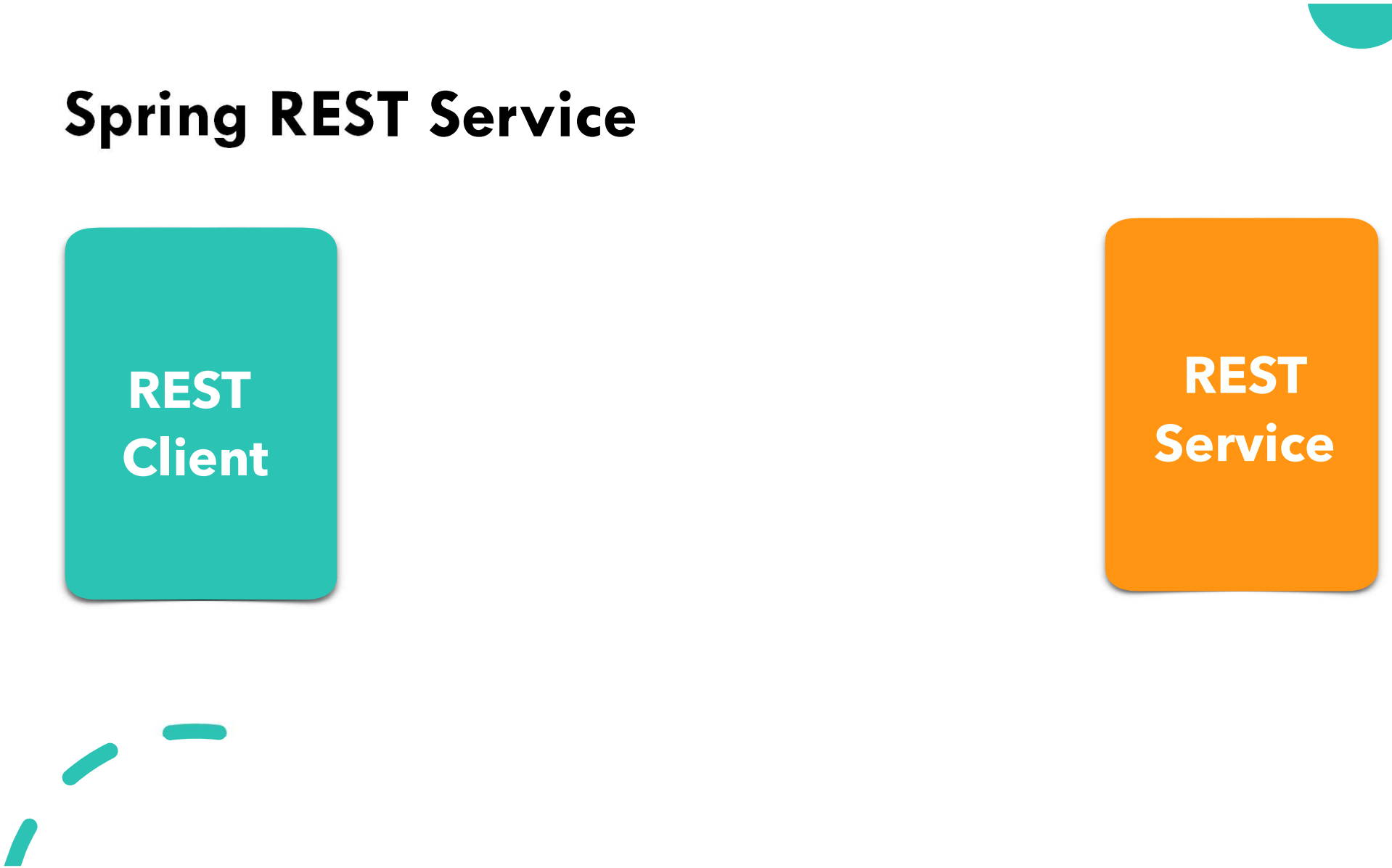
Jackson Data Binding

ทบทวน

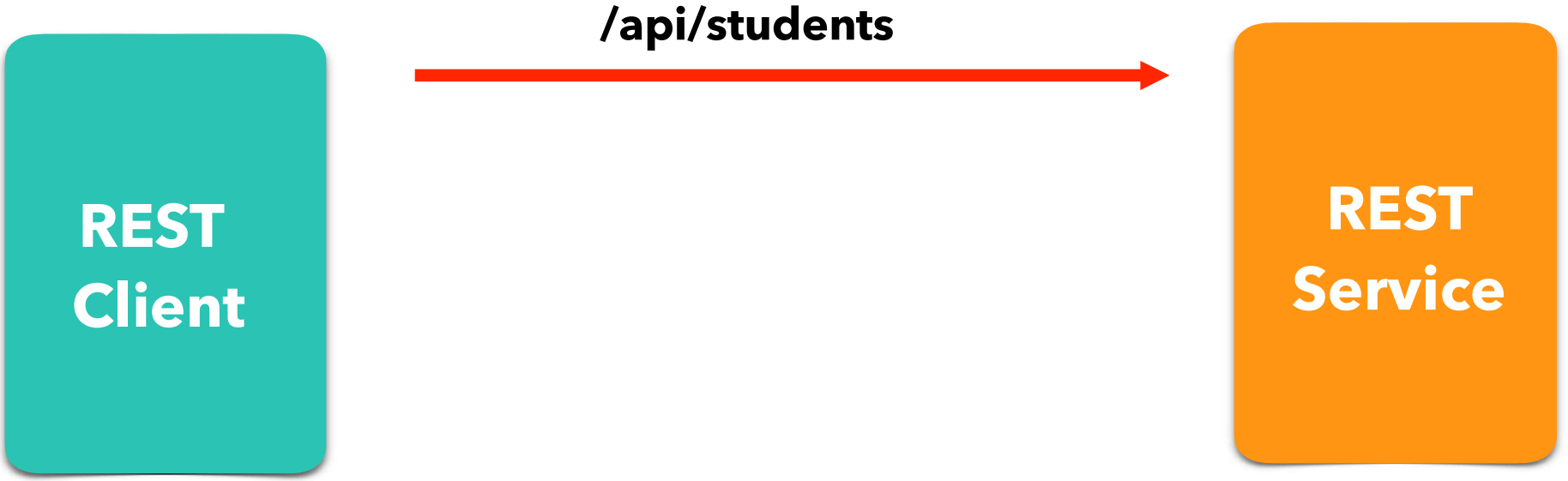
- Jackson จะเรียกใช้เมธอด getter/setter ที่เหมาะสม



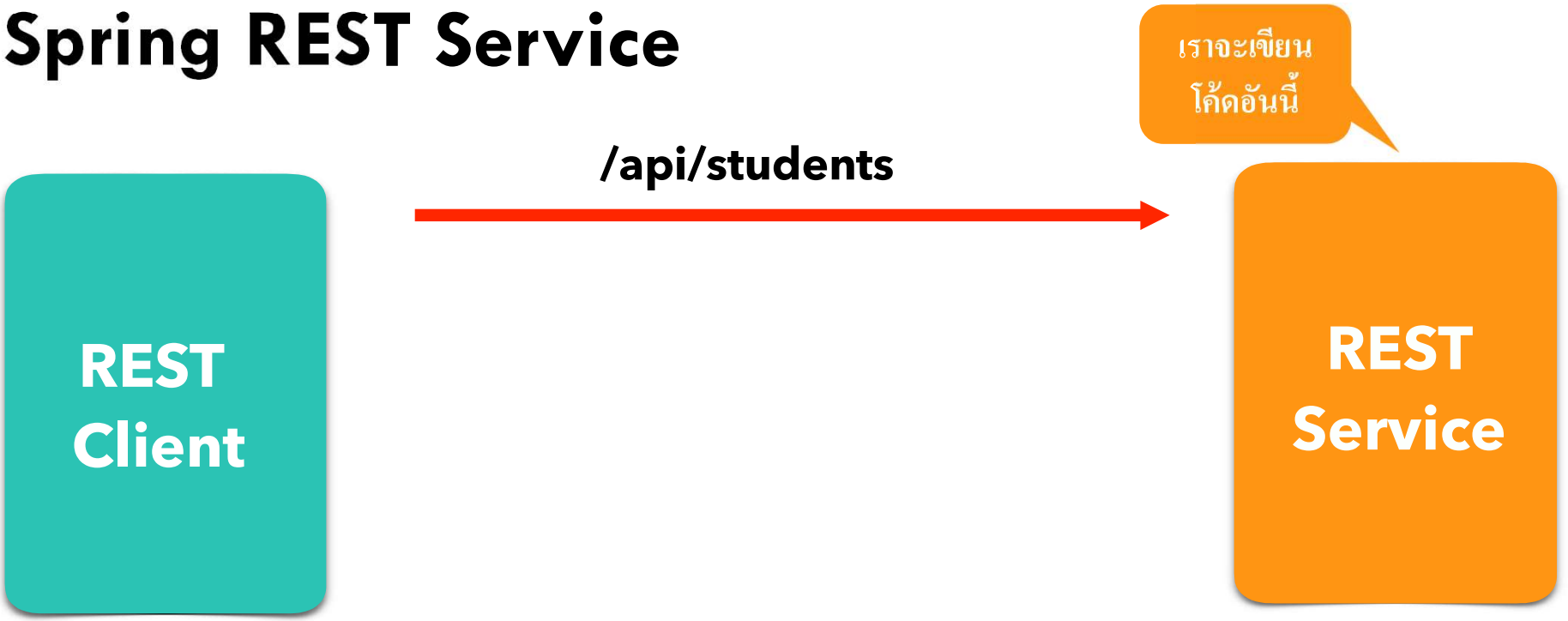
Spring REST Service



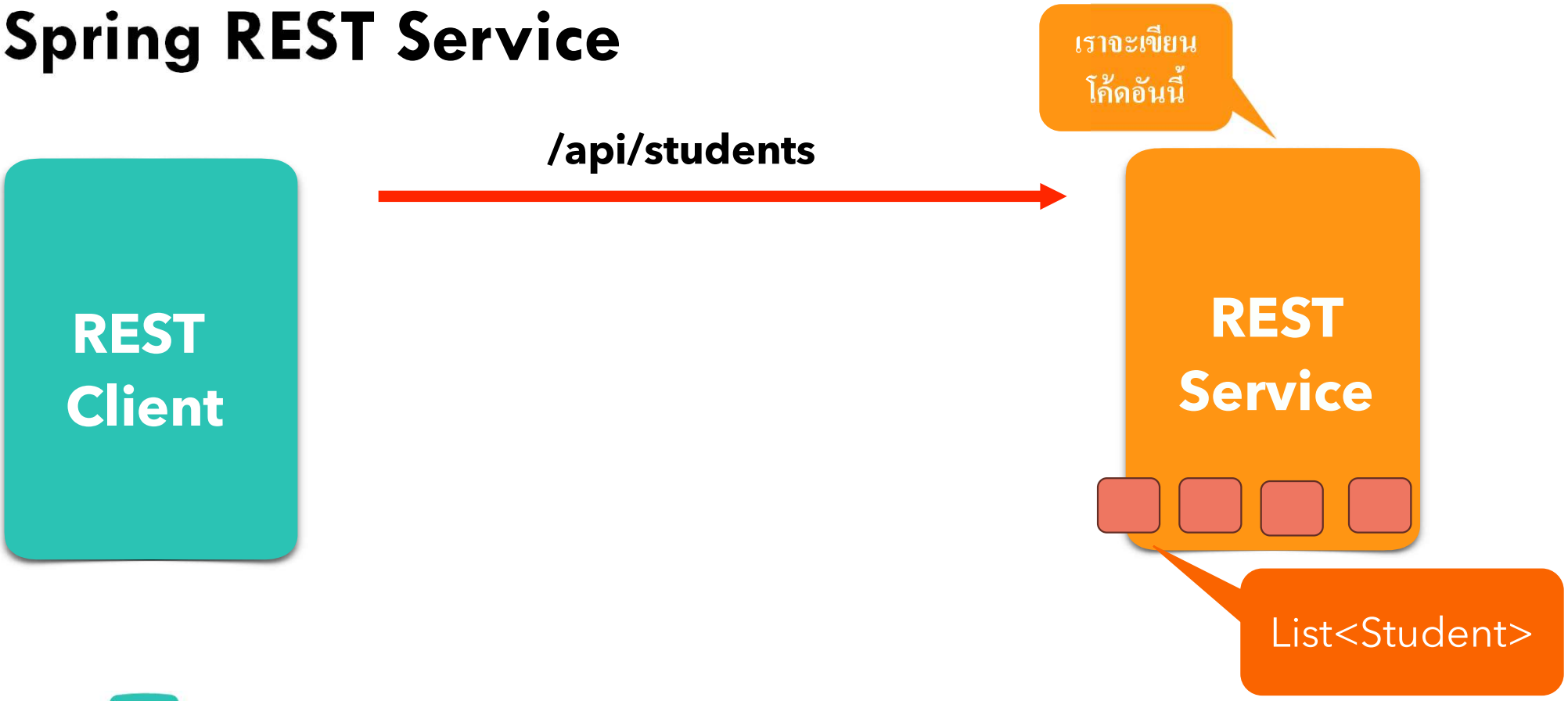
Spring REST Service



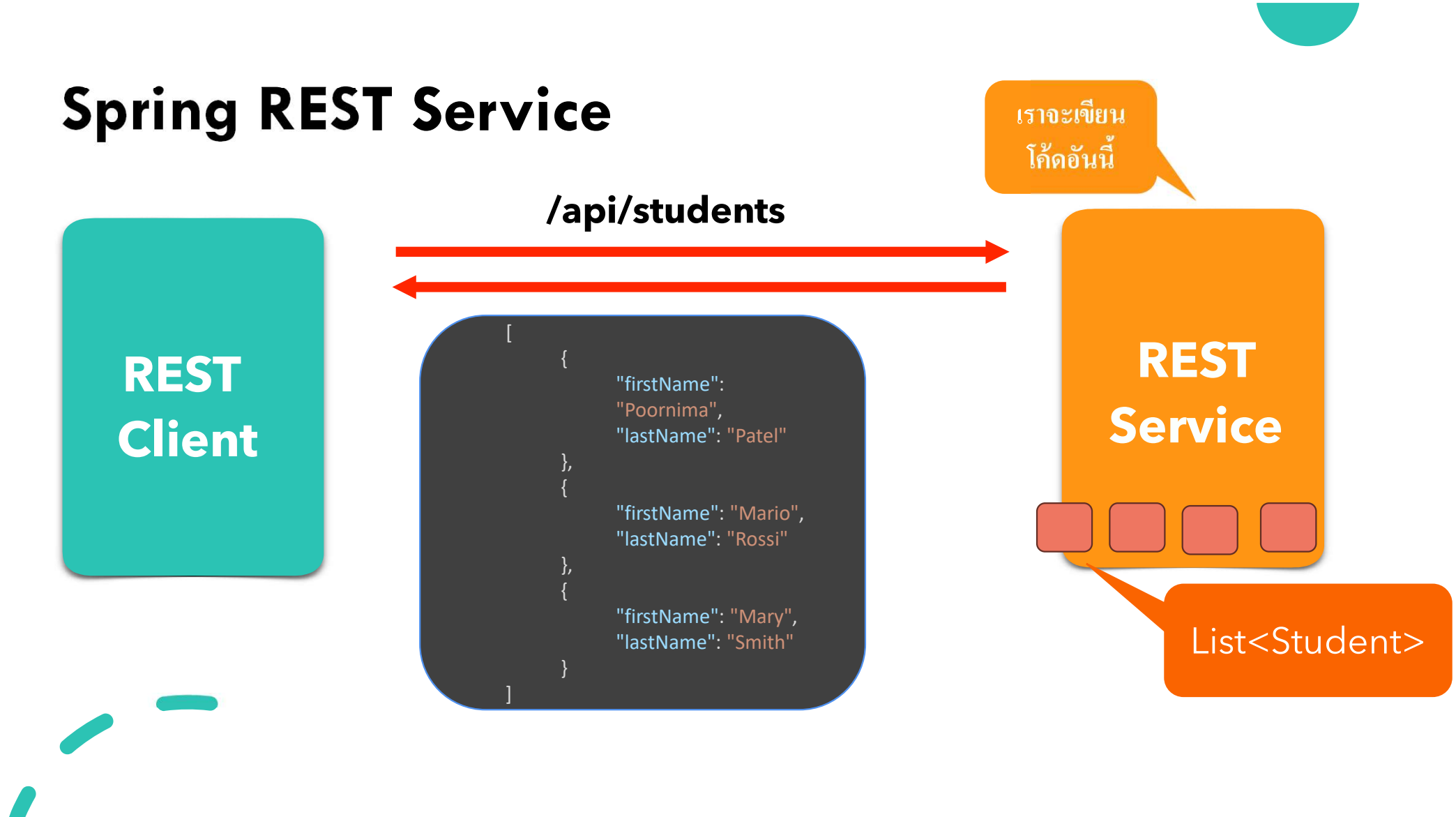
Spring REST Service



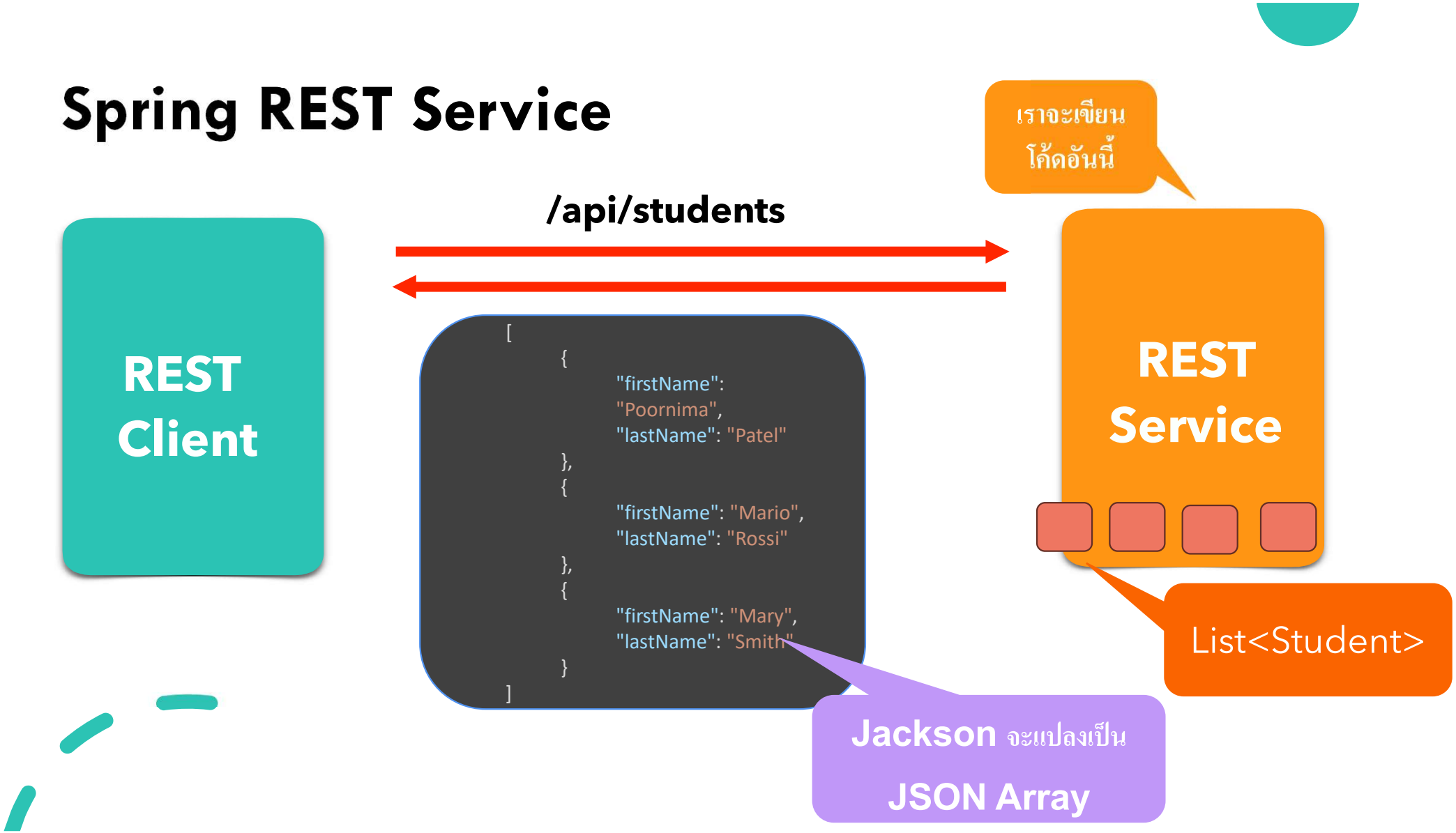
Spring REST Service



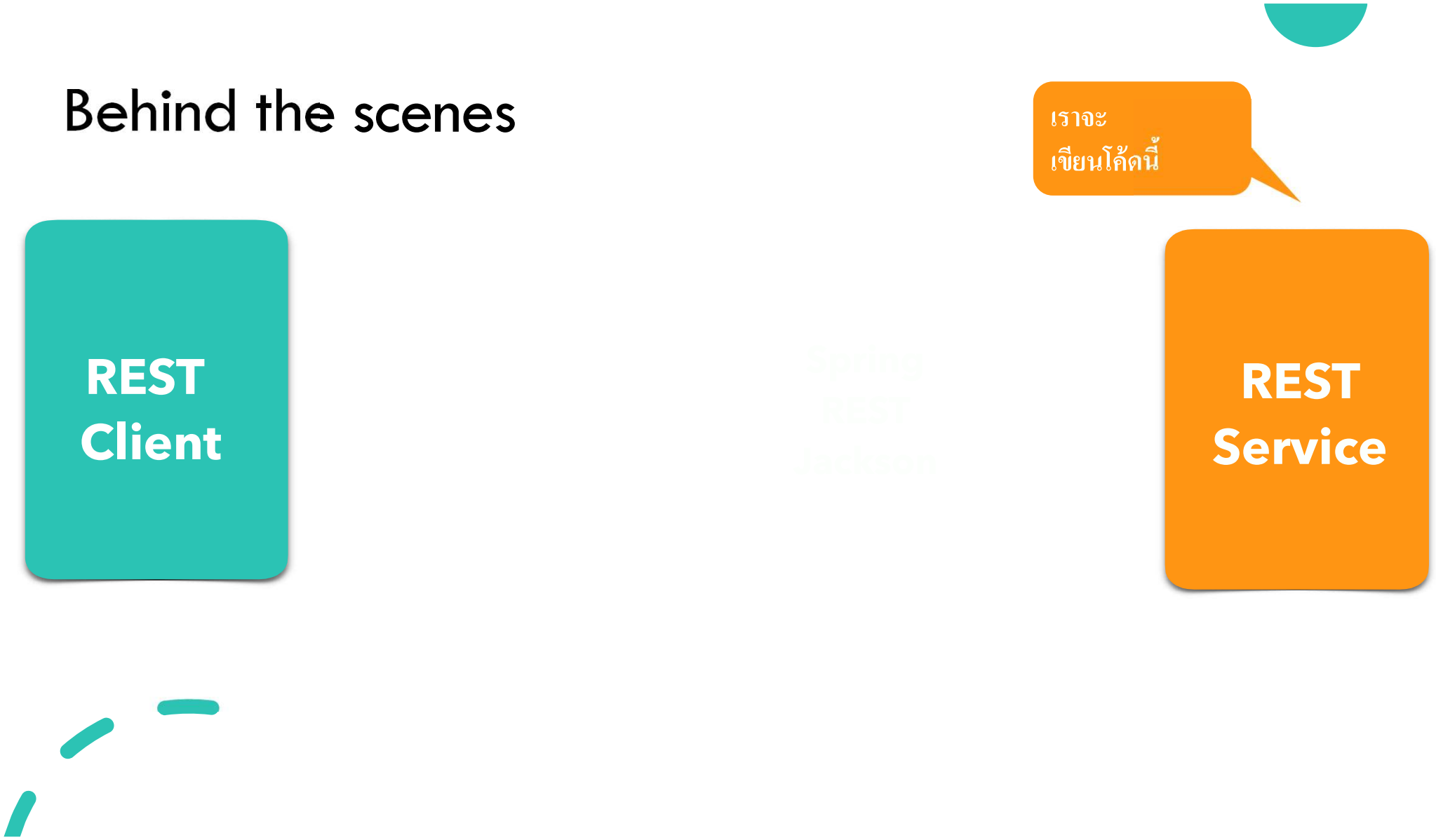
Spring REST Service



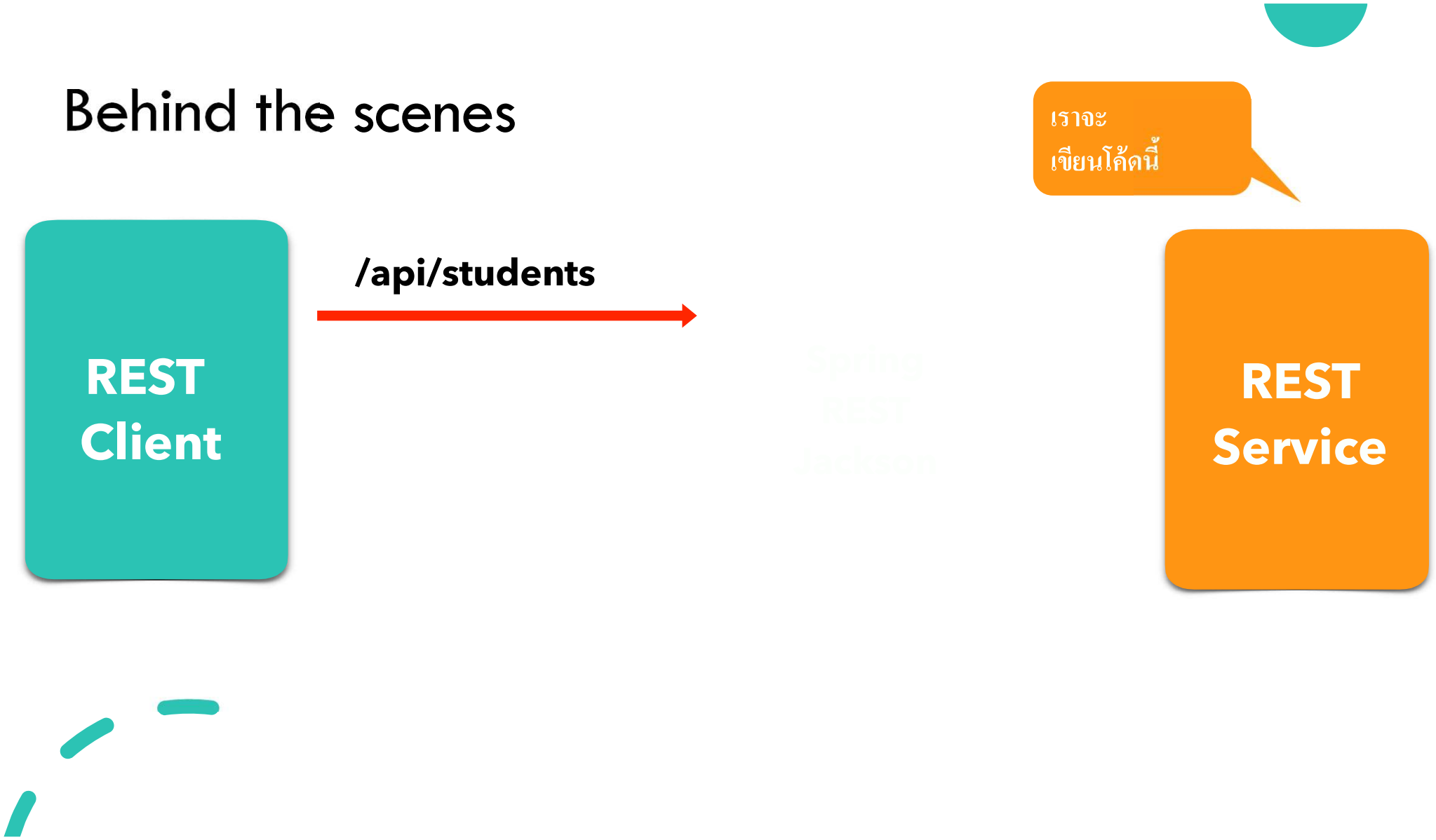
Spring REST Service



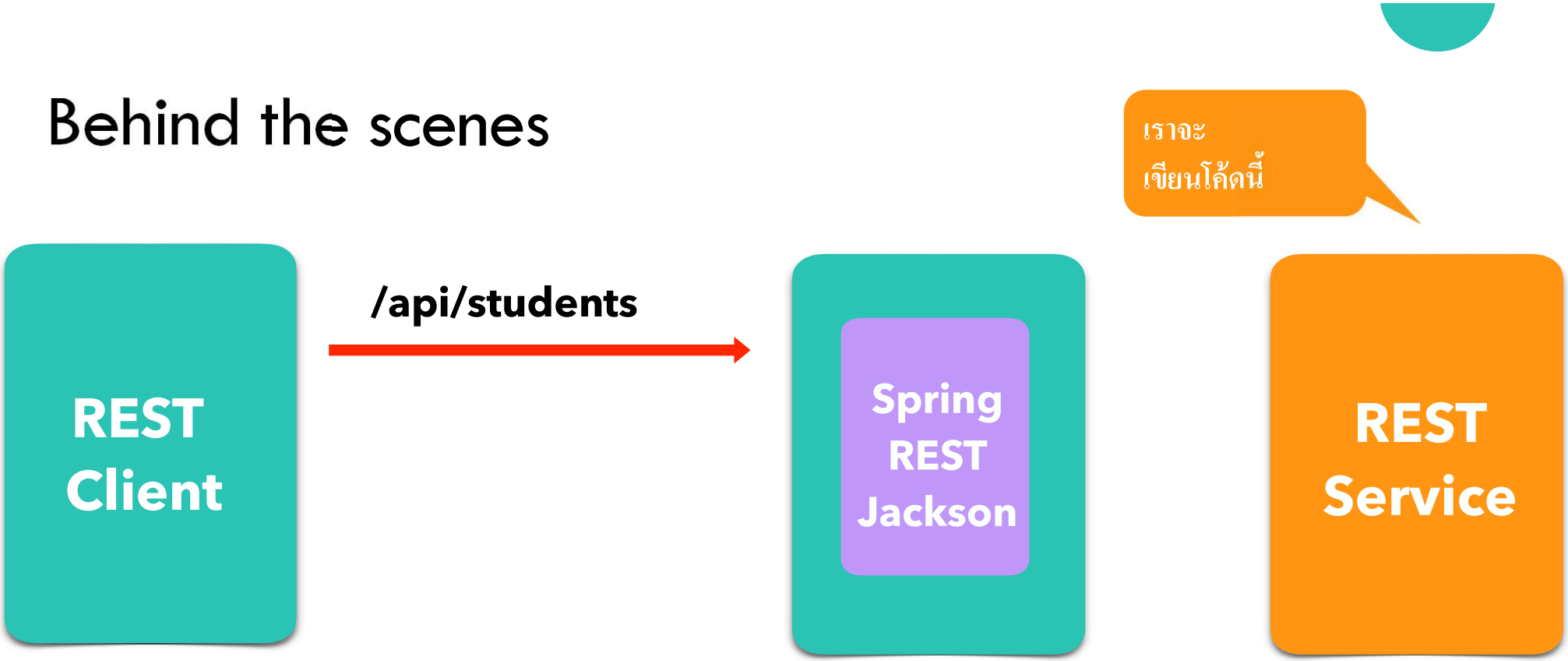
Behind the scenes



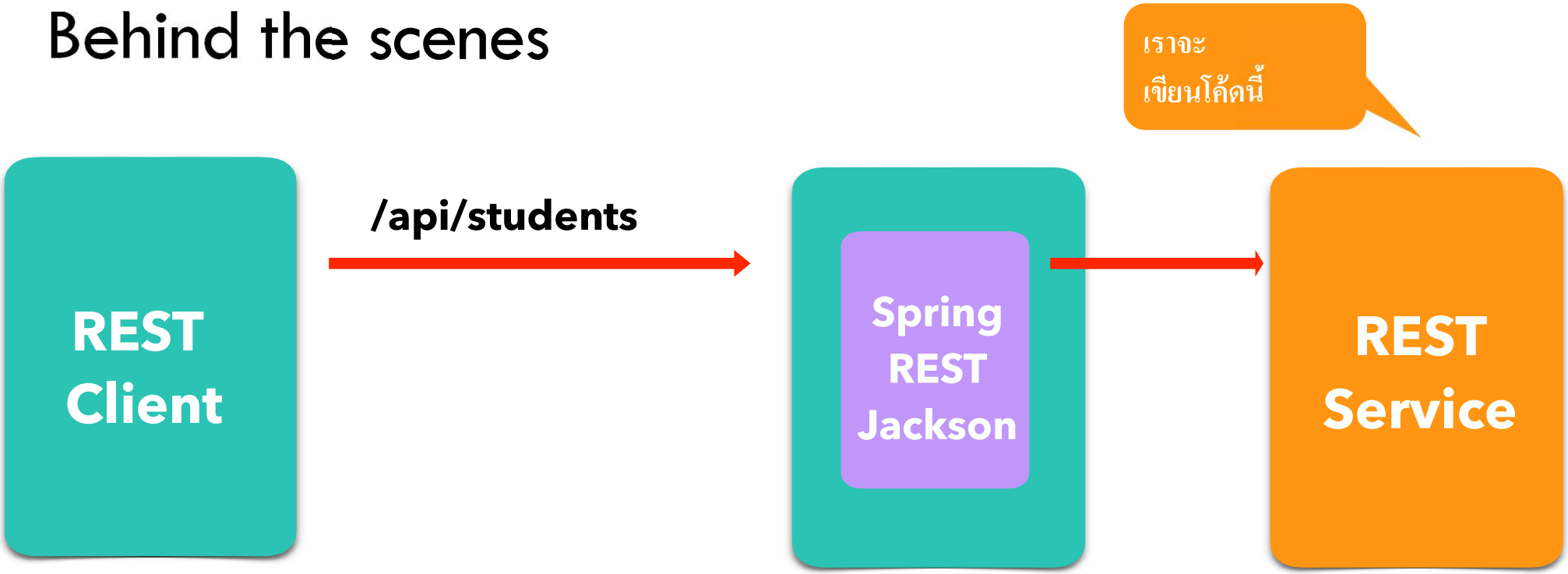
Behind the scenes



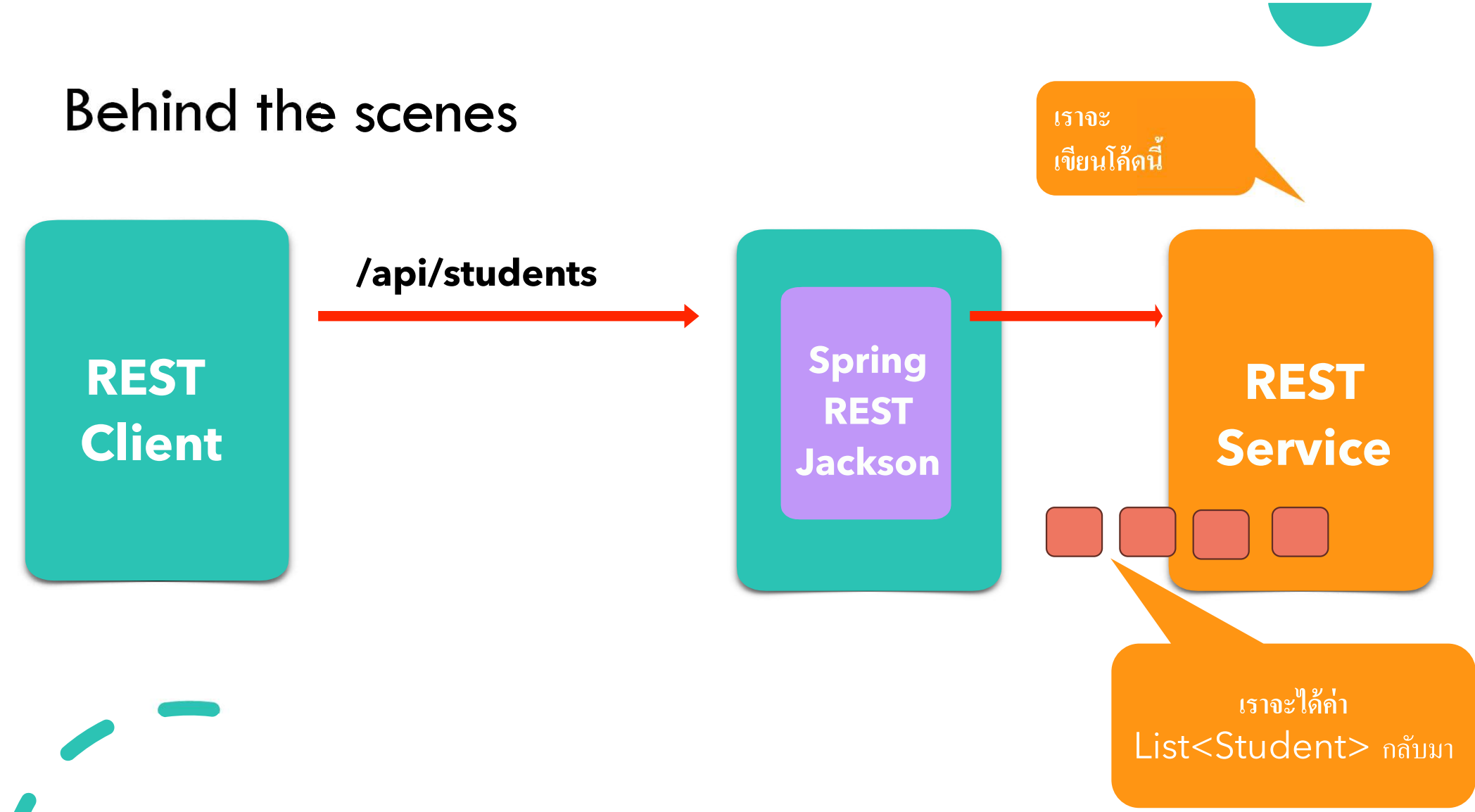
Behind the scenes



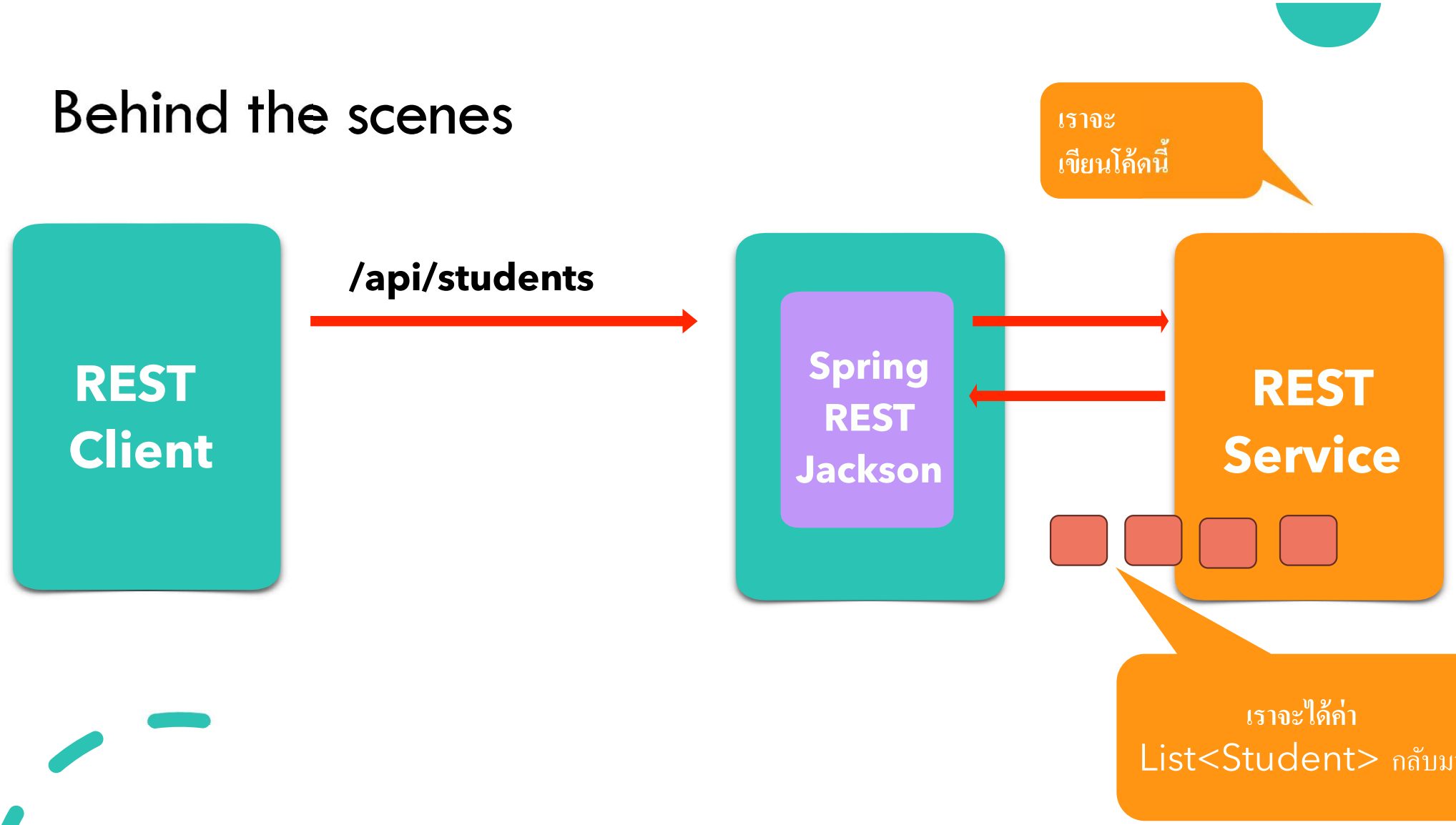
Behind the scenes



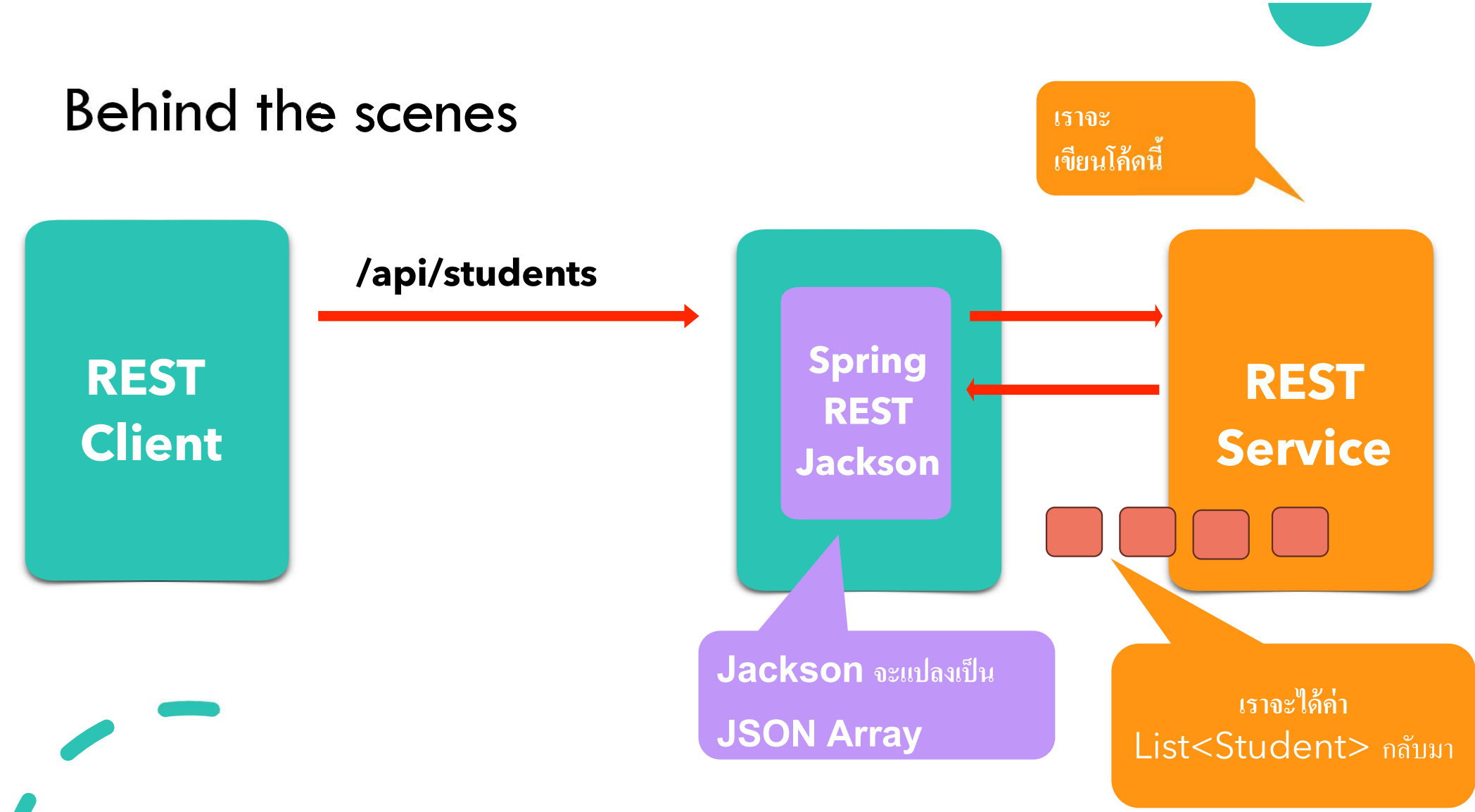
Behind the scenes



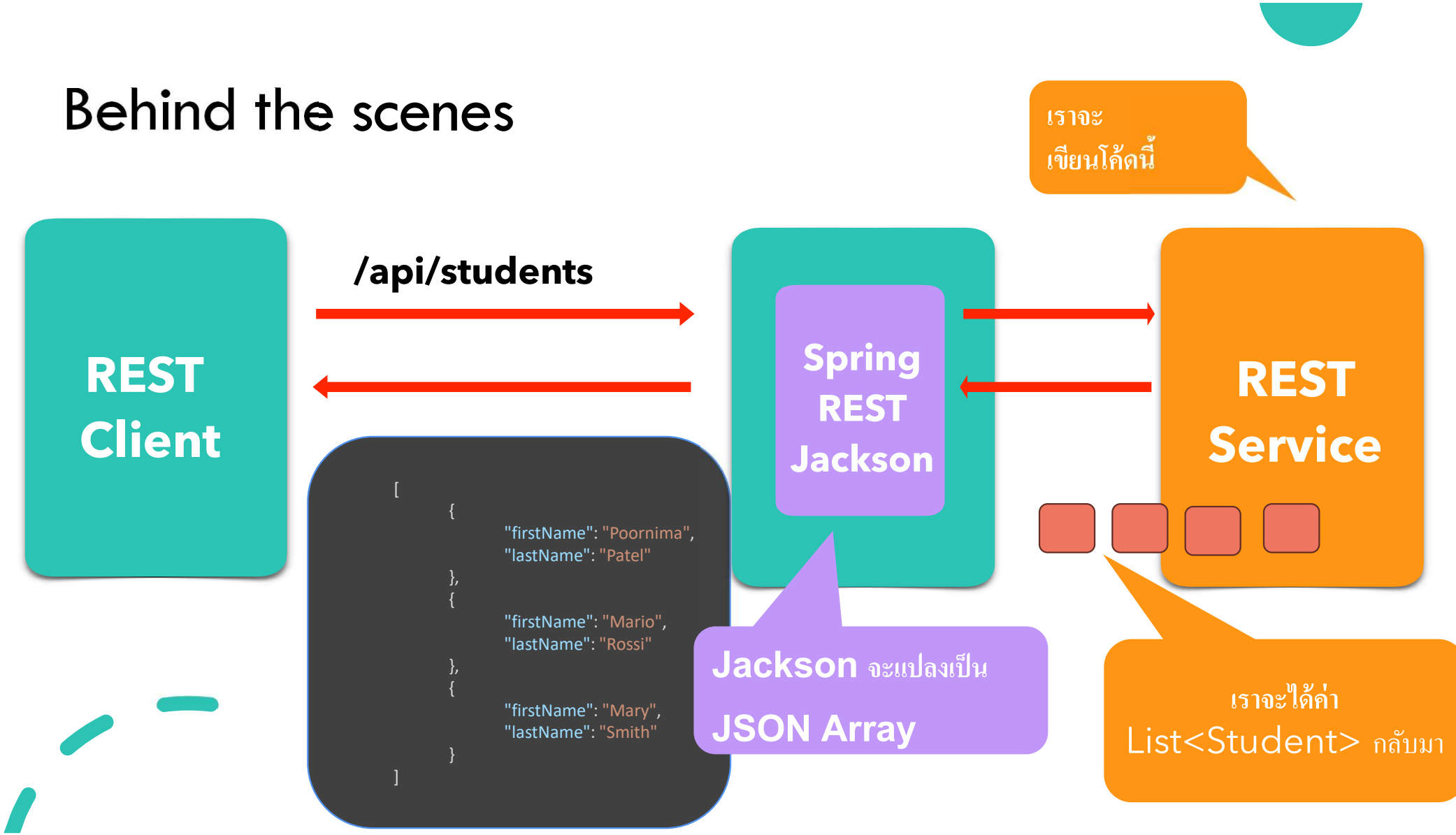
Behind the scenes



Behind the scenes



Behind the scenes



ขั้นตอนการพัฒนา

1. สร้าง Java POJO class สำหรับ **Student**

2. สร้าง Spring REST Service โดยใช้ **@RestController**

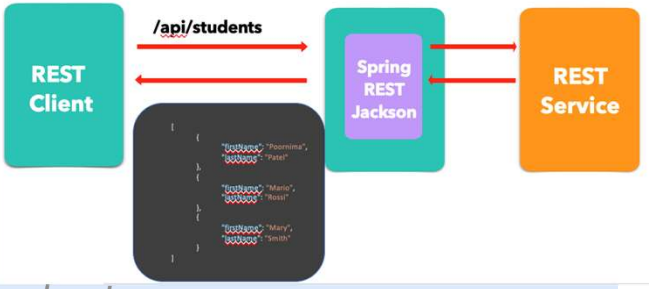
ขั้นแรก: สร้าง Java POJO class สำหรับ **Student**

Student
firstName : String lastName : String
getFirstName() : String setFirstName() : void getLastName() : String setLastName() : void

Fields
Constructors
Getter/Setters

```
public class Student {
    private String firstName;
    private String lastName;
    public Student() {
    }
    public Student(String firstName, String lastName) {
       this.firstName = firstName;
       this.lastName = lastName;
    }
    public String getFirstName() {
       return firstName;
    }
    public void setFirstName(String firstName) {
       this.firstName = firstName;
    }
    public String getLastName() {
       return lastName;
    }
    public void setLastName(String lastName) {
       this.lastName = lastName;
    }
}
```

ขั้นสอง: สร้าง Spring REST Service โดยใช้ @RestController




```java
@RestController
@RequestMapping ("/api")
public class StudentRestController {
    // define endpoint for "/students" - return list of students
    @GetMapping ("/students")
    public List<Student> getStudents() {
        List<Student> theStudents = new ArrayList <>();
        theStudents.add(new Student("Poornima", "Patel"));
        theStudents.add(new Student("Mario", "Rossi"));
        theStudents.add(new Student("Mary", "Smith"));

        return theStudents;
    }
}
```

ตอนนี้เราจะ Hardcode ไปก่อน

เดี๋ยวมาเพิ่ม DB ในภายหลัง

Jackson จะแปลง List<Student> เป็นJSON

Spring REST Service กับ POJO

- Mini-workshop: Spring REST Service กับ POJO

- Ref: lab-s5-l2

Spring REST – Path Variables

Chapter 7

Path Variabless

- ค้นหานักเรียนหนึ่งคน โดยใช้ id

GET /api/students/{**studentId**} ผลลัพธ์ที่ได้จะเป็นนักเรียนหนึ่งคน

Path Variabless

- ค้นหานักเรียนหนึ่งคน โดยใช้ id

GET /api/students/{**studentId**} ผลลัพธ์ที่ได้จะเป็นนักเรียนหนึ่งคน

สิ่งนี้เรียกว่า
"path variable"

Path Variabless

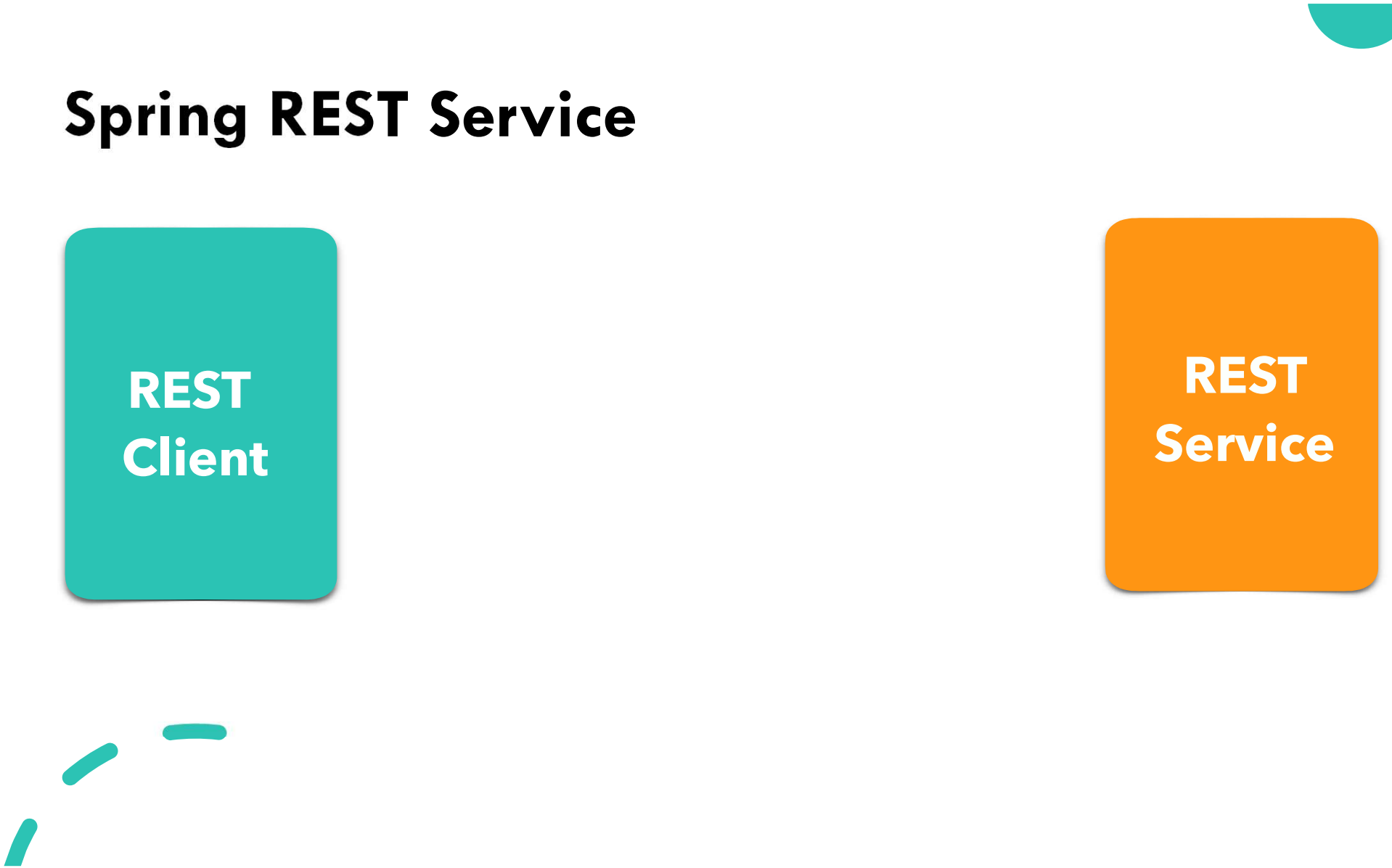
- ค้นหานักเรียนหนึ่งคน โดยใช้ id

GET /api/students/{**studentId**} ผลลัพธ์ที่ได้จะเป็นนักเรียนหนึ่งคน

/api/students/0
/api/students/1
/api/students/2

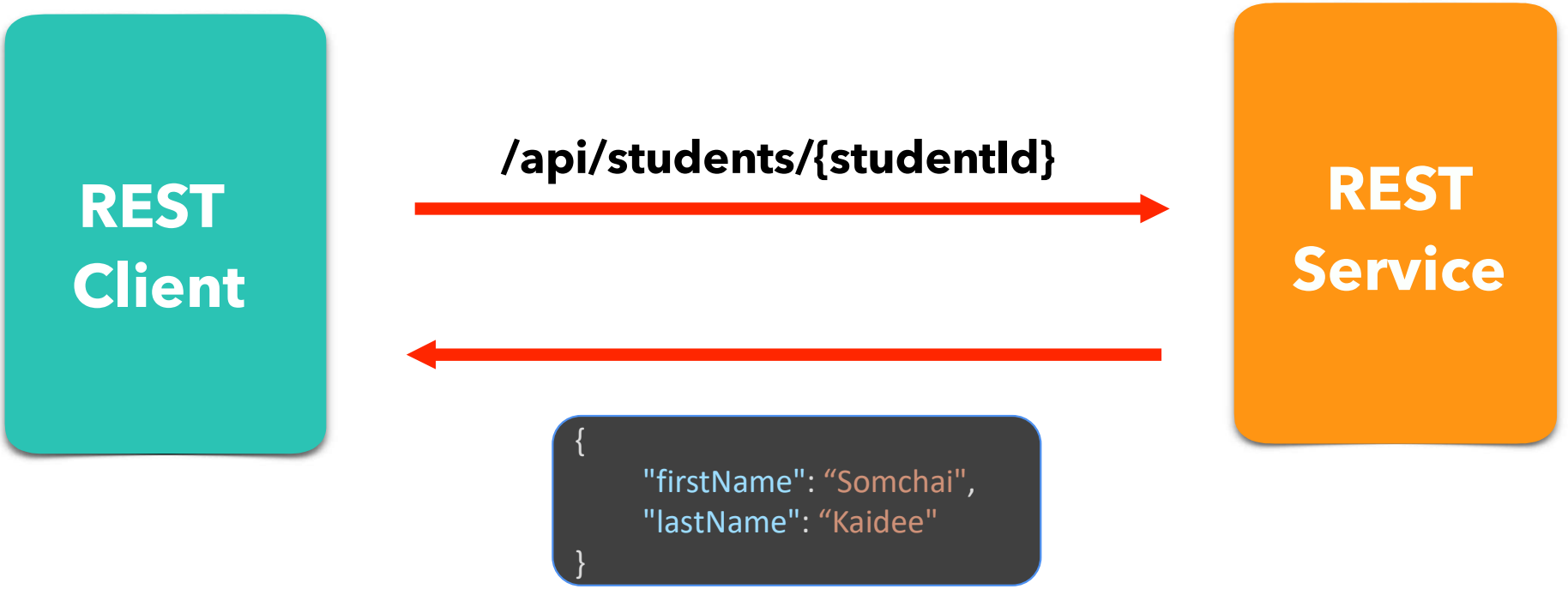
สิ่งนี้เรียกว่า
"path variable"

Spring REST Service

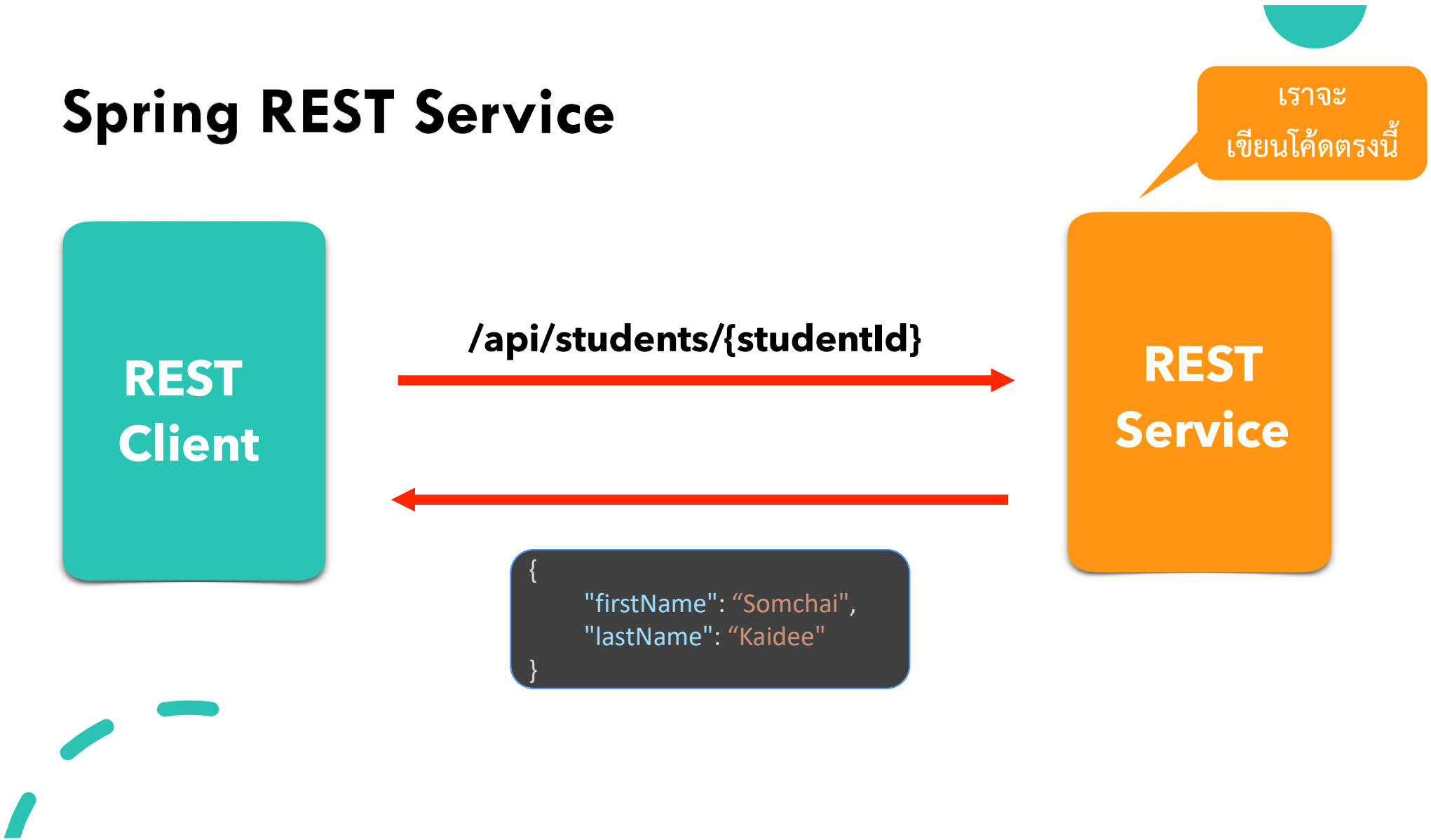


Spring REST Service

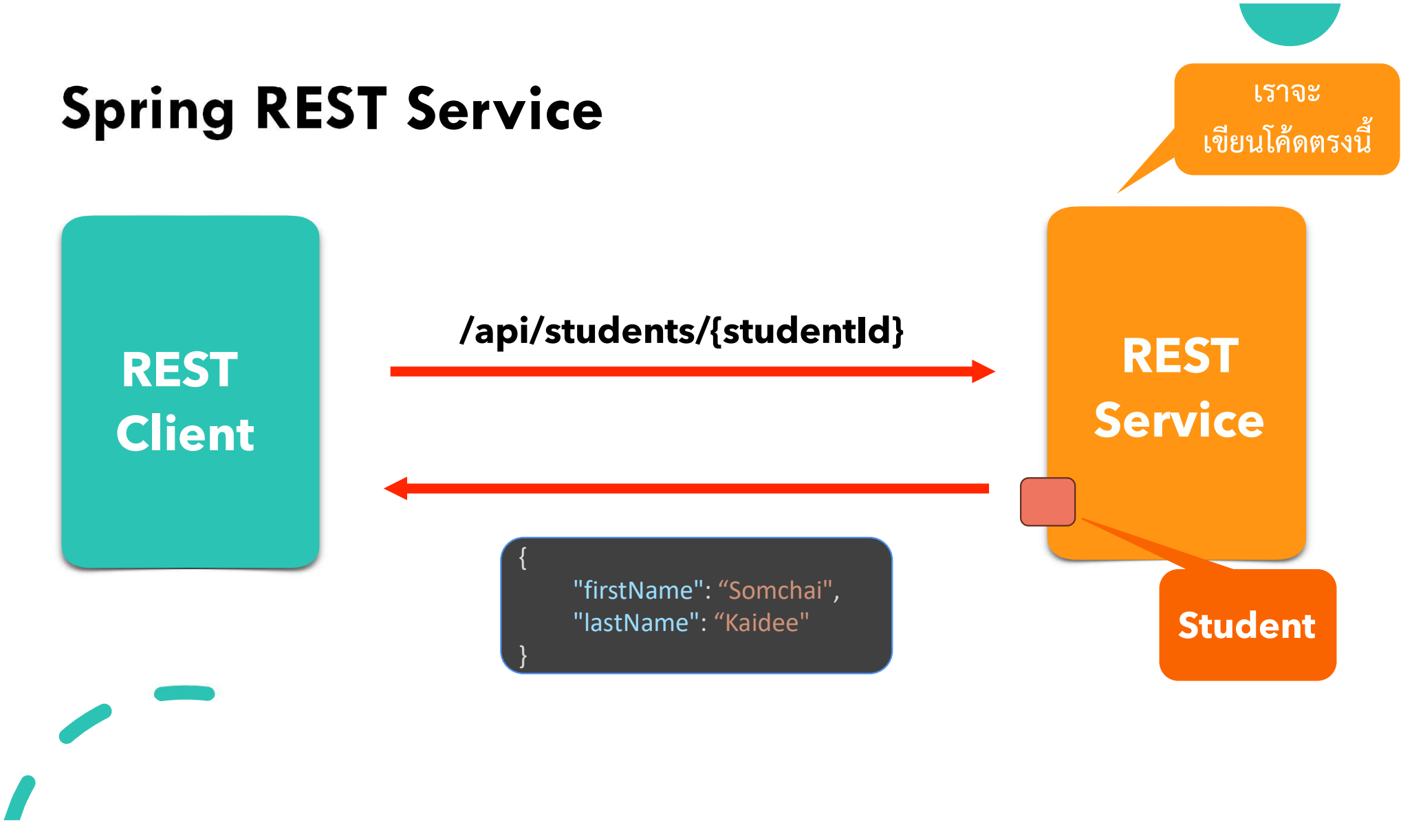
Spring REST Service



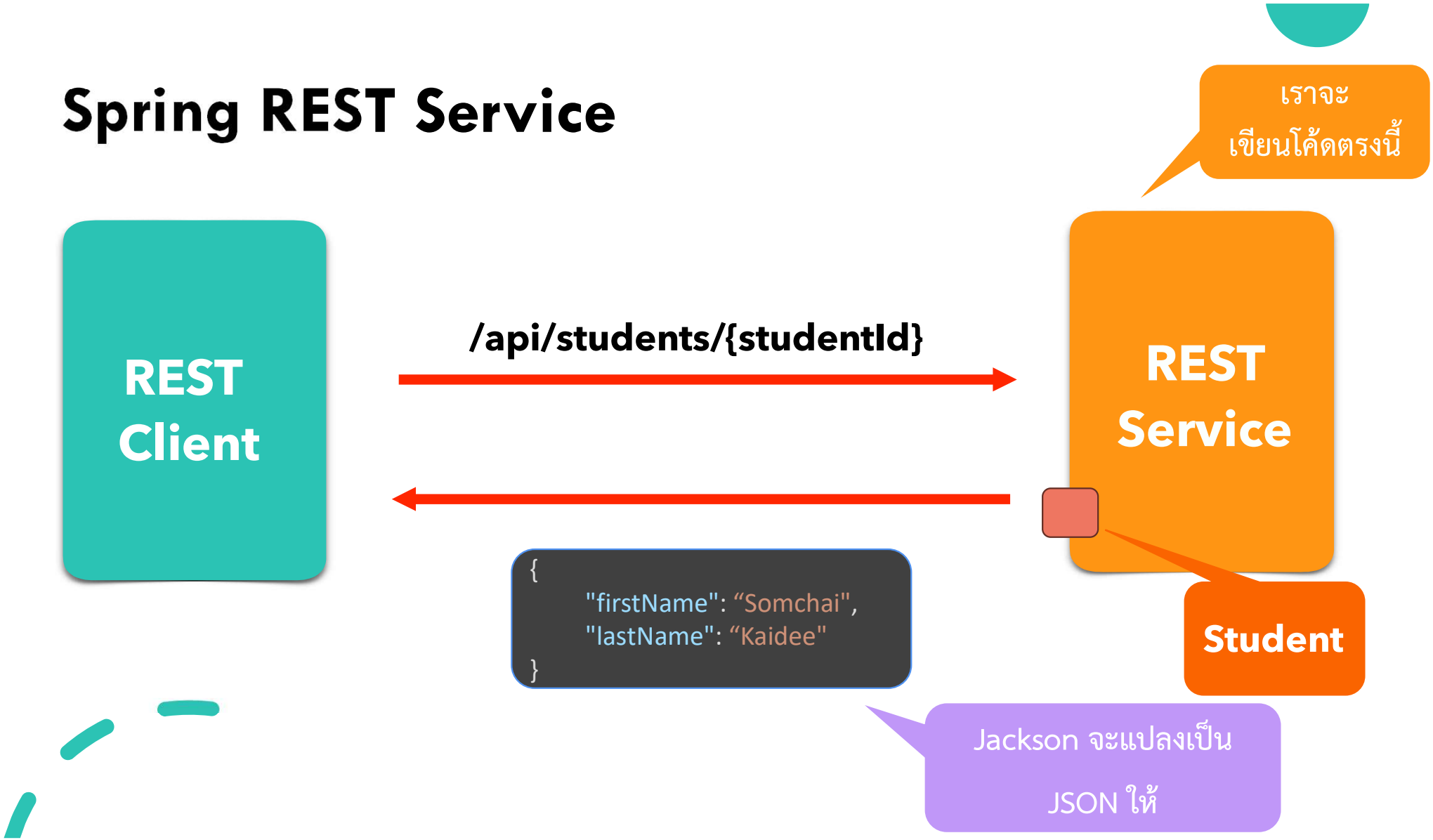
Spring REST Service



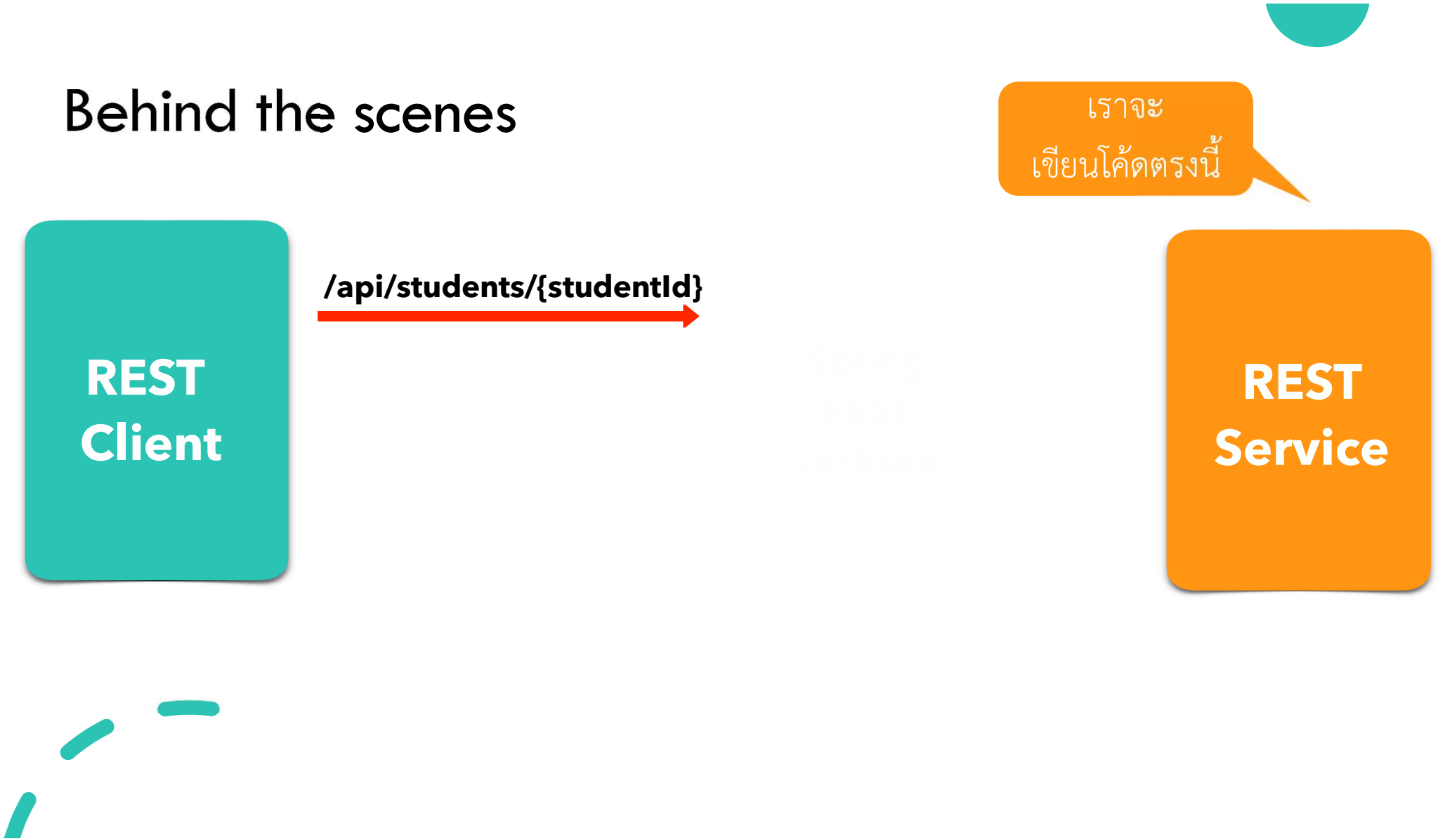
Spring REST Service



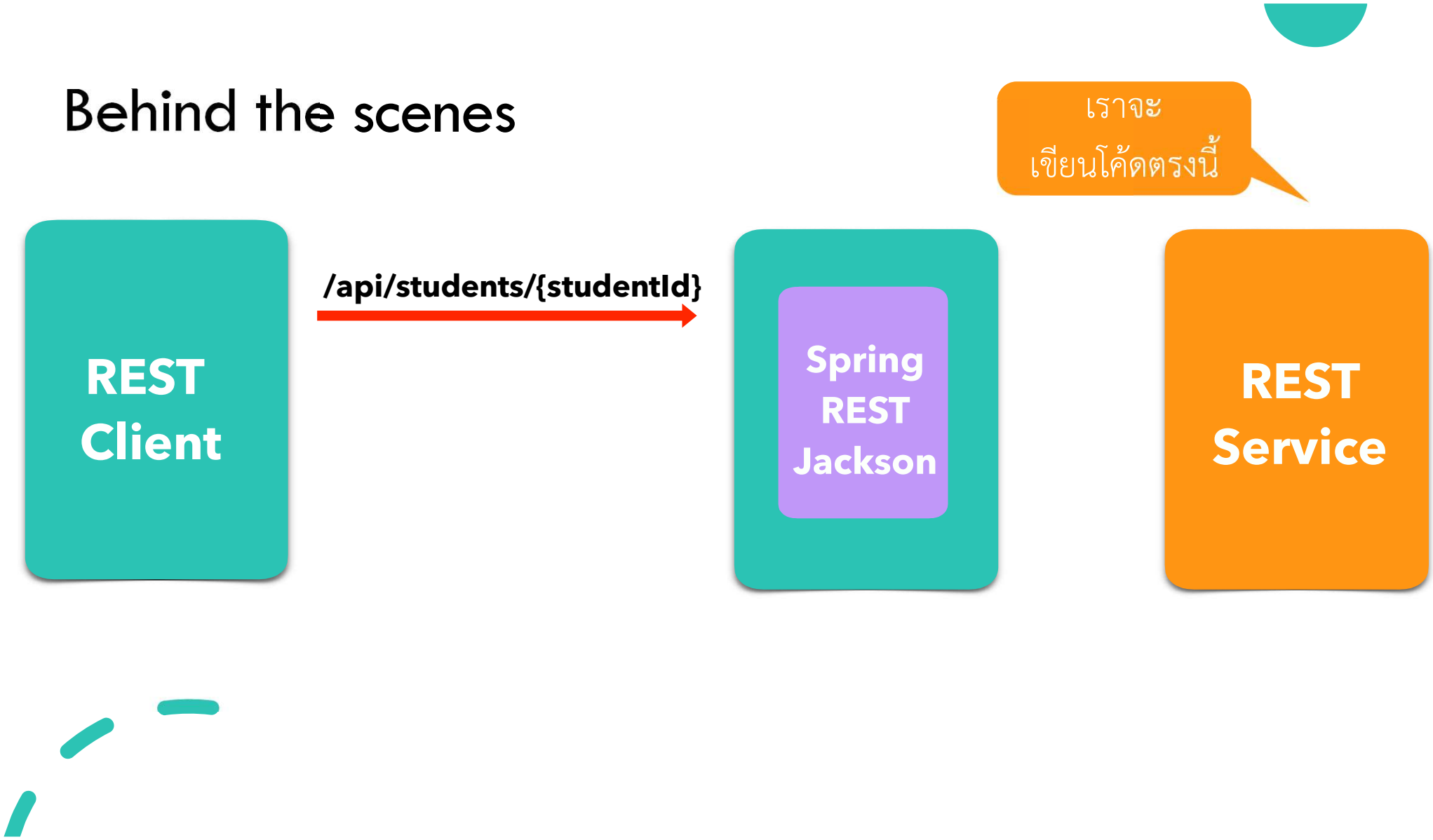
Spring REST Service



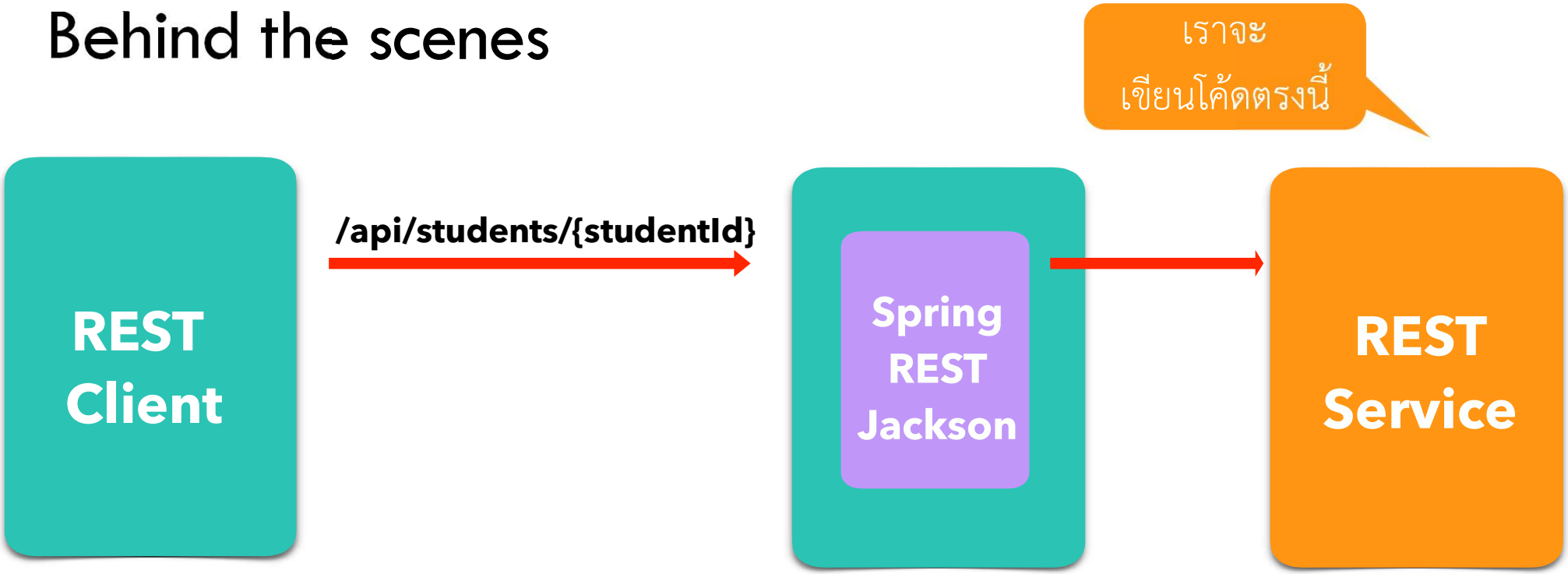
Behind the scenes



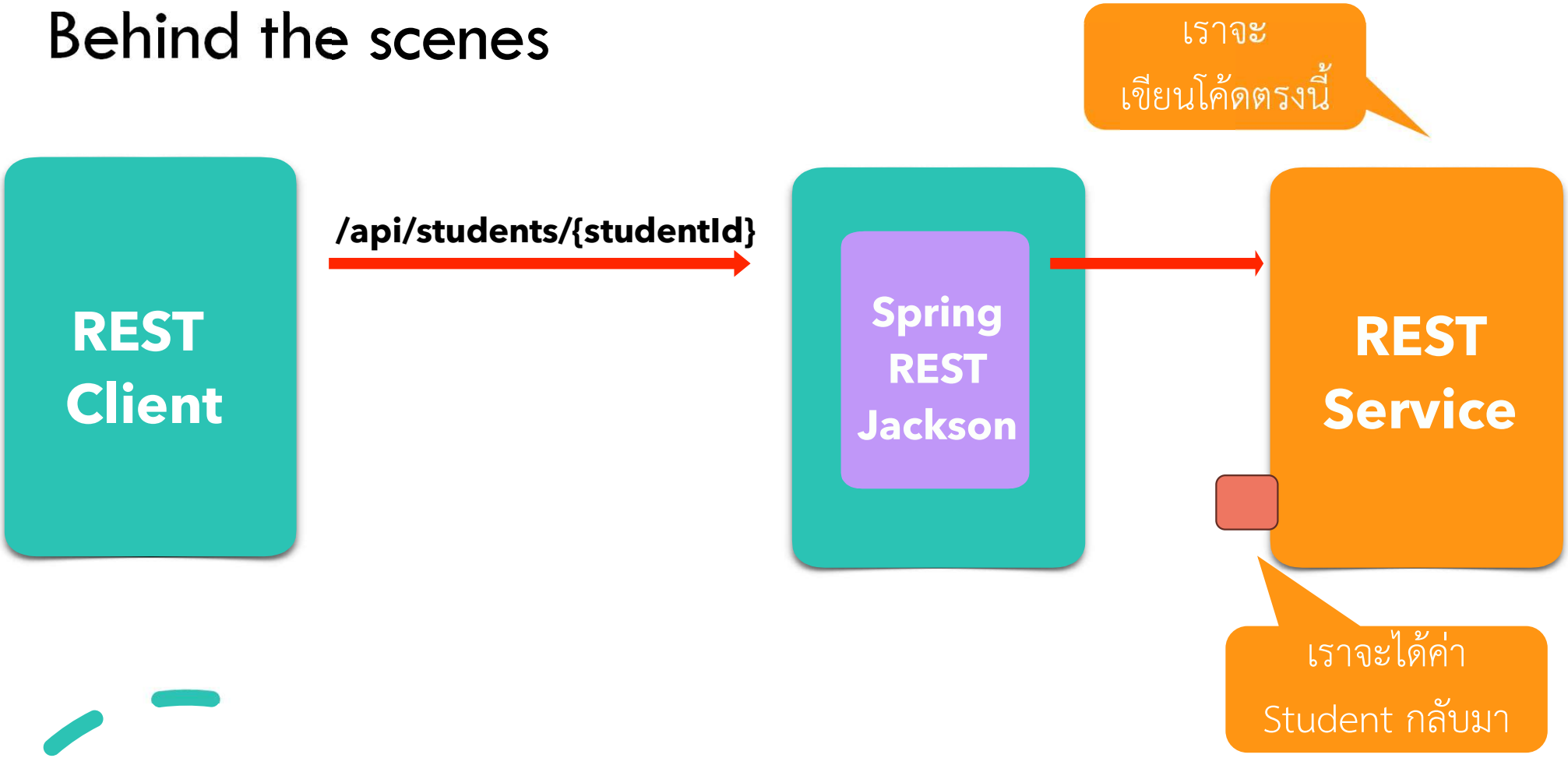
Behind the scenes



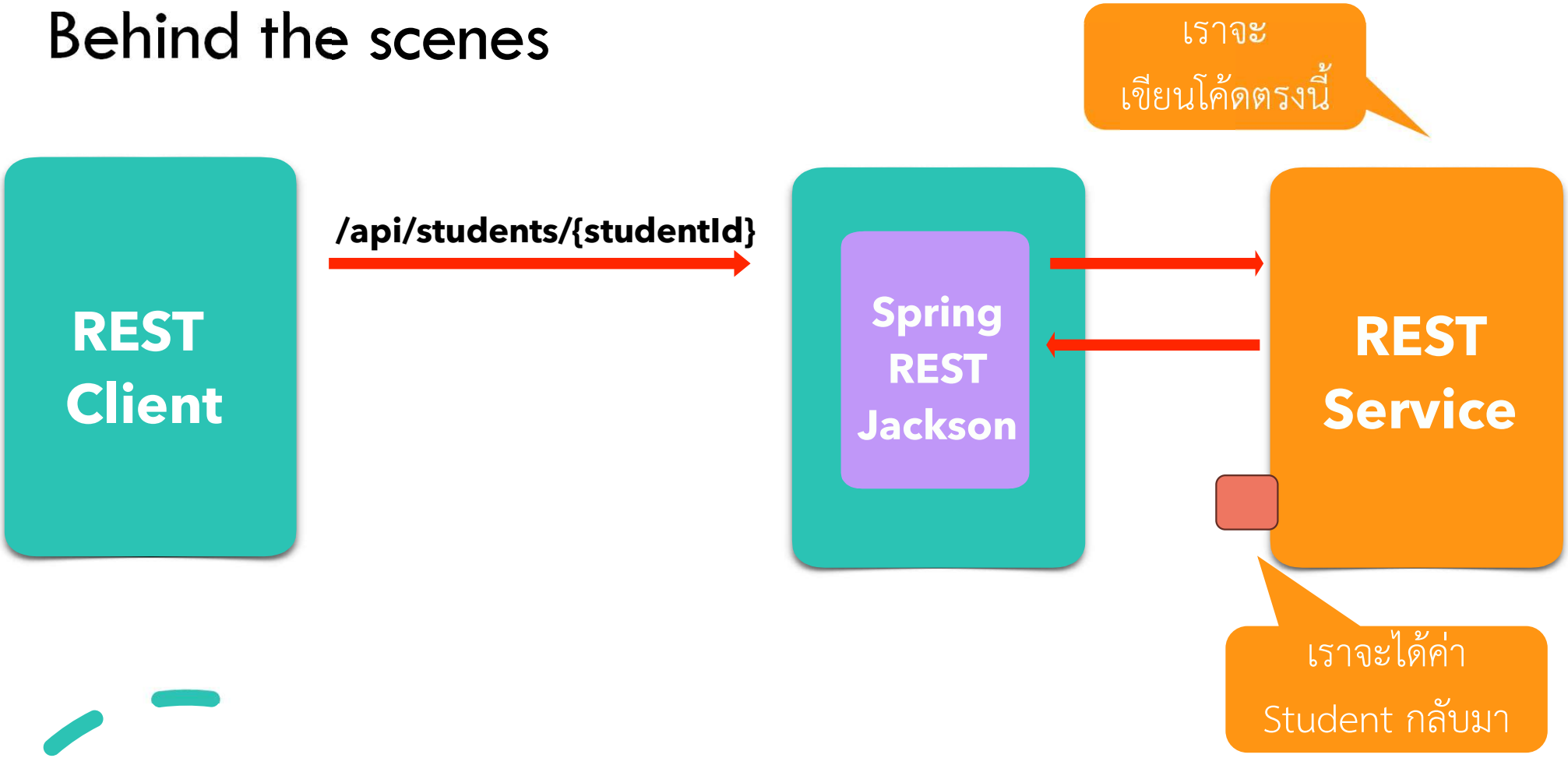
Behind the scenes



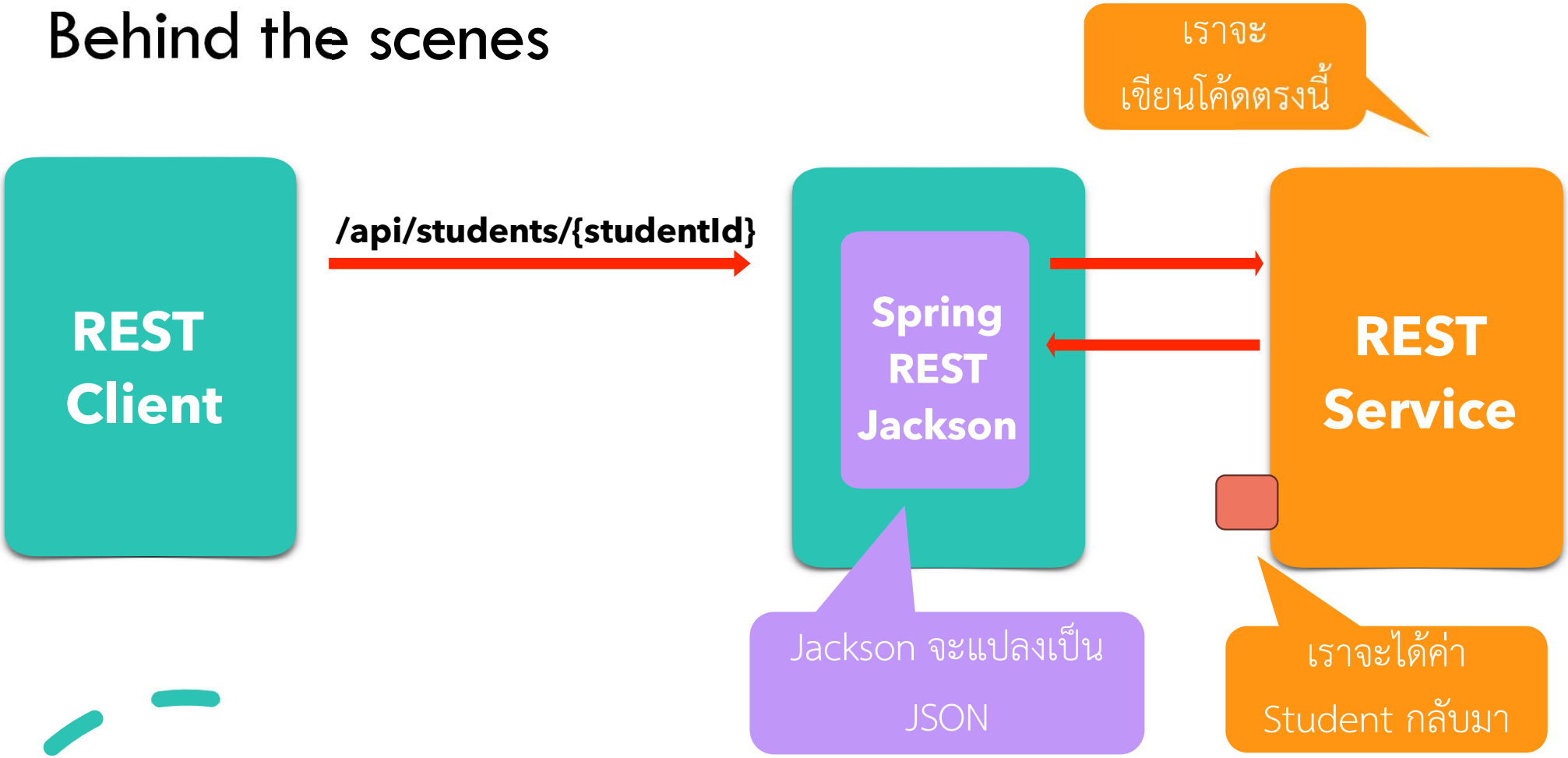
Behind the scenes



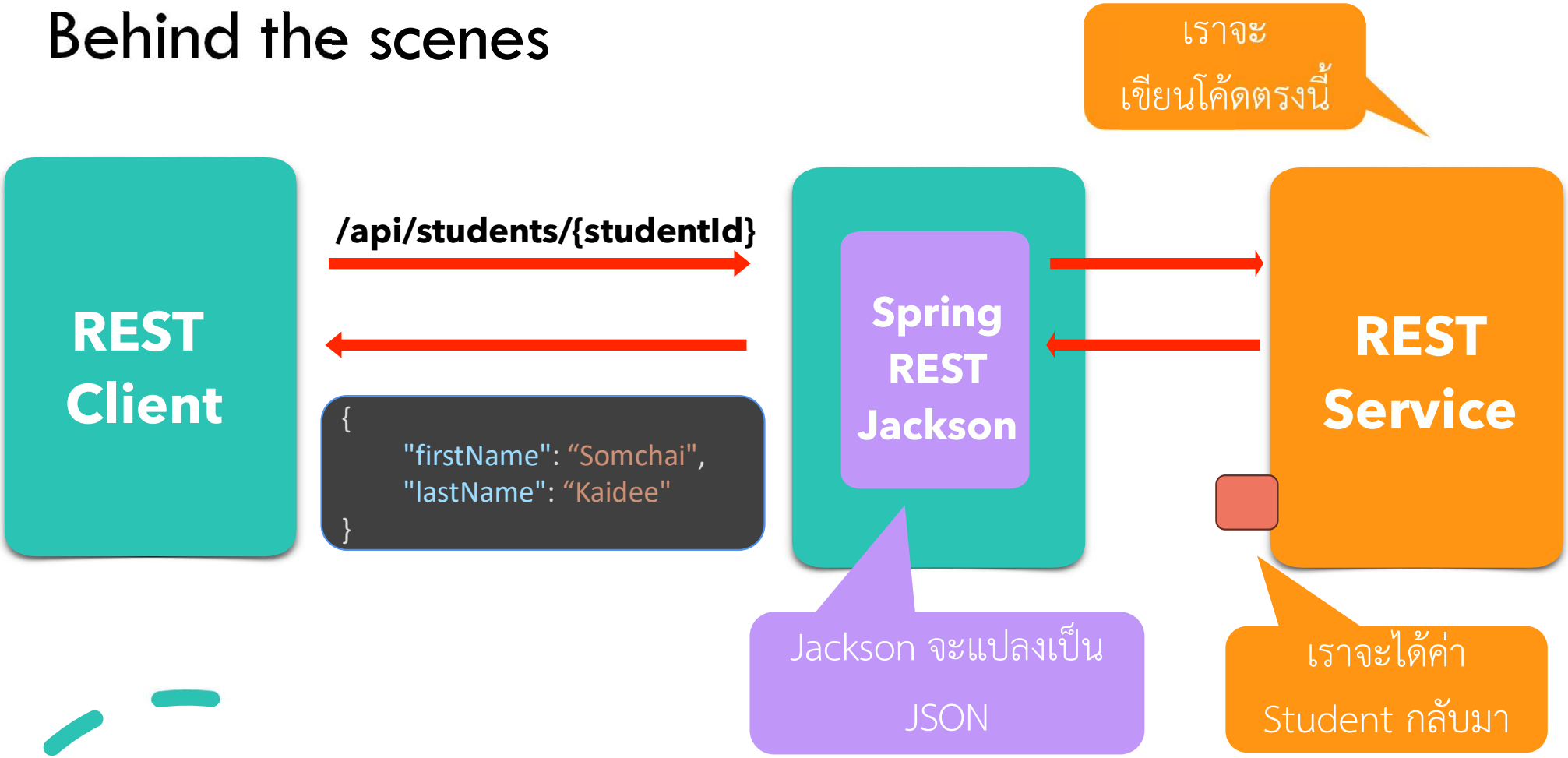
Behind the scenes



Behind the scenes



Behind the scenes



ขั้นตอนการพัฒนา

ทีละขั้นตอน

1. เพิ่ม request mapping ไปยัง Spring REST Service
   - เชื่อม path variable กับพารามิเตอร์เมธอดโดยใช้ @PathVariable

ขั้นแรก: เพิ่ม Request Mapping


```java
@RestController
@RequestMapping("/api")
public class StudentRestController {

    // define endpoint for "/students/{studentId}" - return student at index
    @GetMapping("/students/{studentId}")
    public Student getStudent(@PathVariable int studentId) {
        List<Student> theStudents = new ArrayList<>();
        // populate theStudents

        ...

        return theStudents.get(studentId);
    }

}
```

เชื่อม path variable
(โดย default ต้องตรงกัน)

ขั้นแรก: เพิ่ม Request Mapping


```
@RestController
@RequestMapping("/api")
public class StudentRestController {

    // define endpoint for "/students/{studentId}" - return student at index
    @GetMapping("/students/{studentId}")
    public Student getStudent(@PathVariable int studentId) {
        List<Student> theStudents = new ArrayList<>();
        // populate theStudents

        ...

        return theStudents.get(studentId);
    }

}
```

เชื่อม path variable
(โดย default ต้องตรงกัน)

Jackson จะแปลงเป็น
JSON

Spring REST – Path variables

- Mini-workshop: Spring REST – Path variables

- Ref: lab-s5-l3

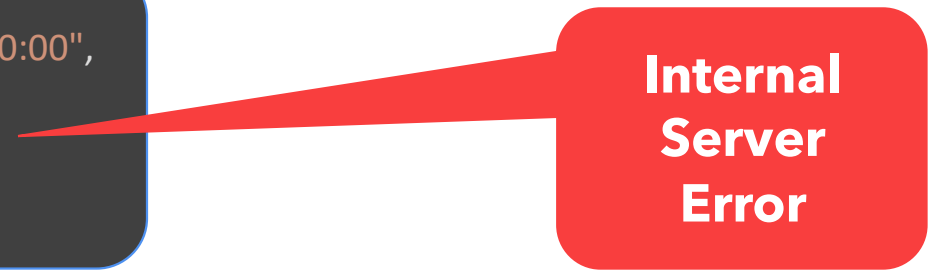
Spring REST – Exception Handling

Chapter 8

ปัญหาที่เกิดขึ้นกับเรา

- Bad student id of 9999 …


```
{
    "timestamp": "T14:07:41.767+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "path": "/api/students/9999"
}
```

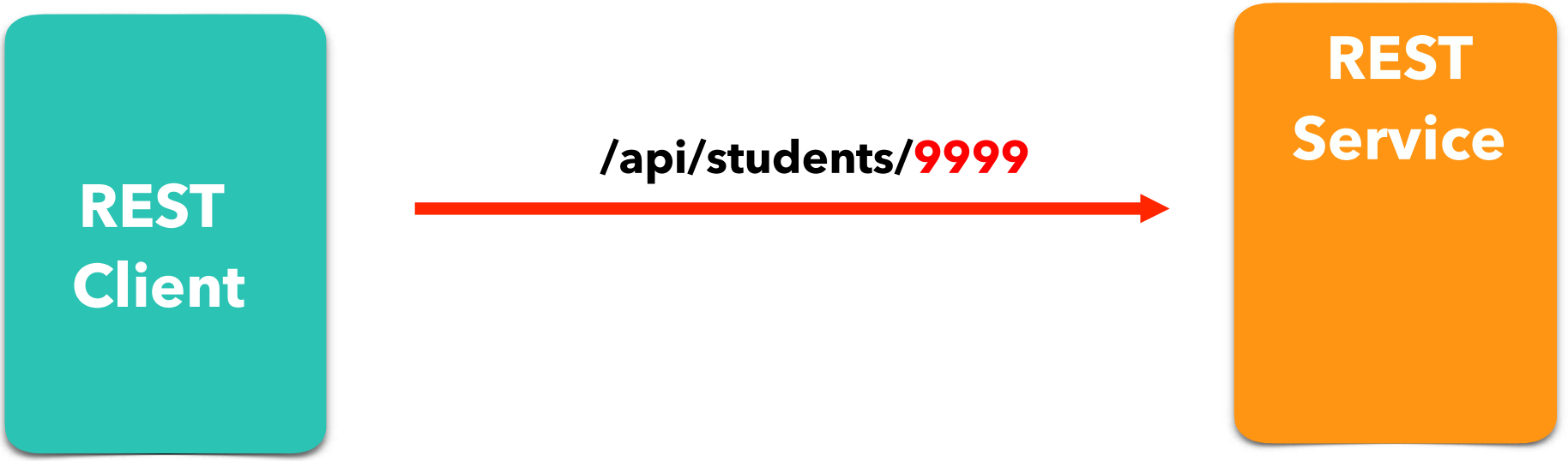



สิ่งที่เราต้องการจริงๆ คือ

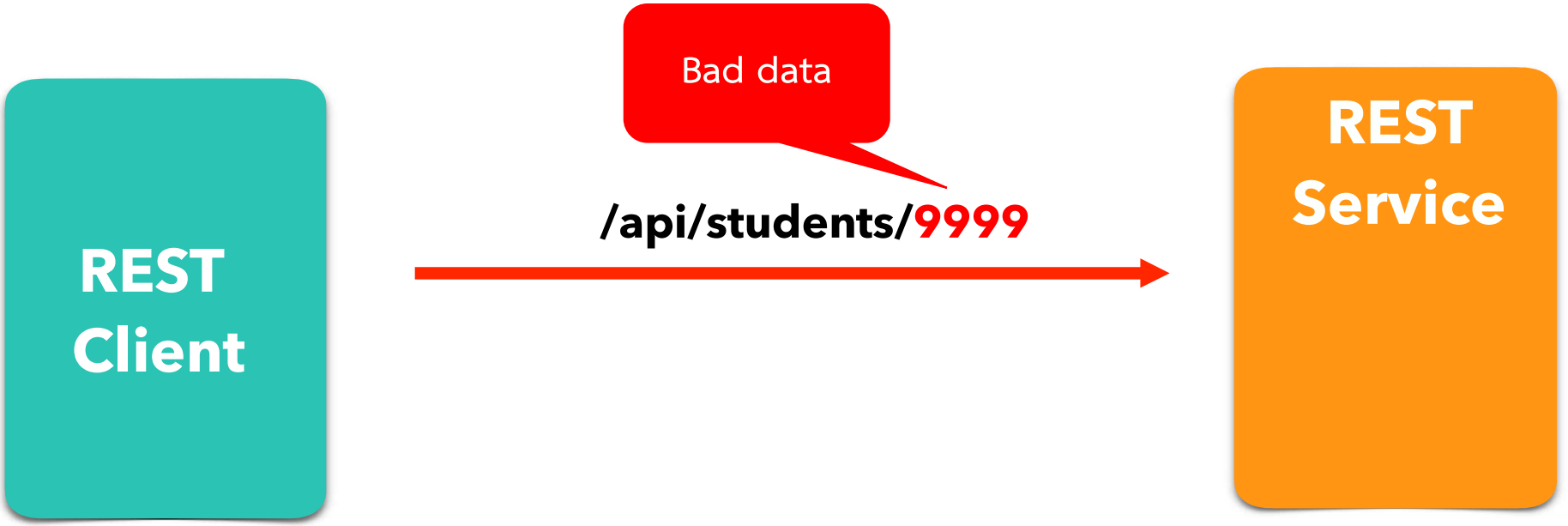
- Handle ตัว Exception และส่งออกไปเป็น JSON ในรูปแบบที่เราต้องการ (ข้อความที่อ่านรู้เรื่อง)


```
{
    "status": 404,
    "message": "Student id not found 9999",
    "timestamp": 1526149650271
}
```

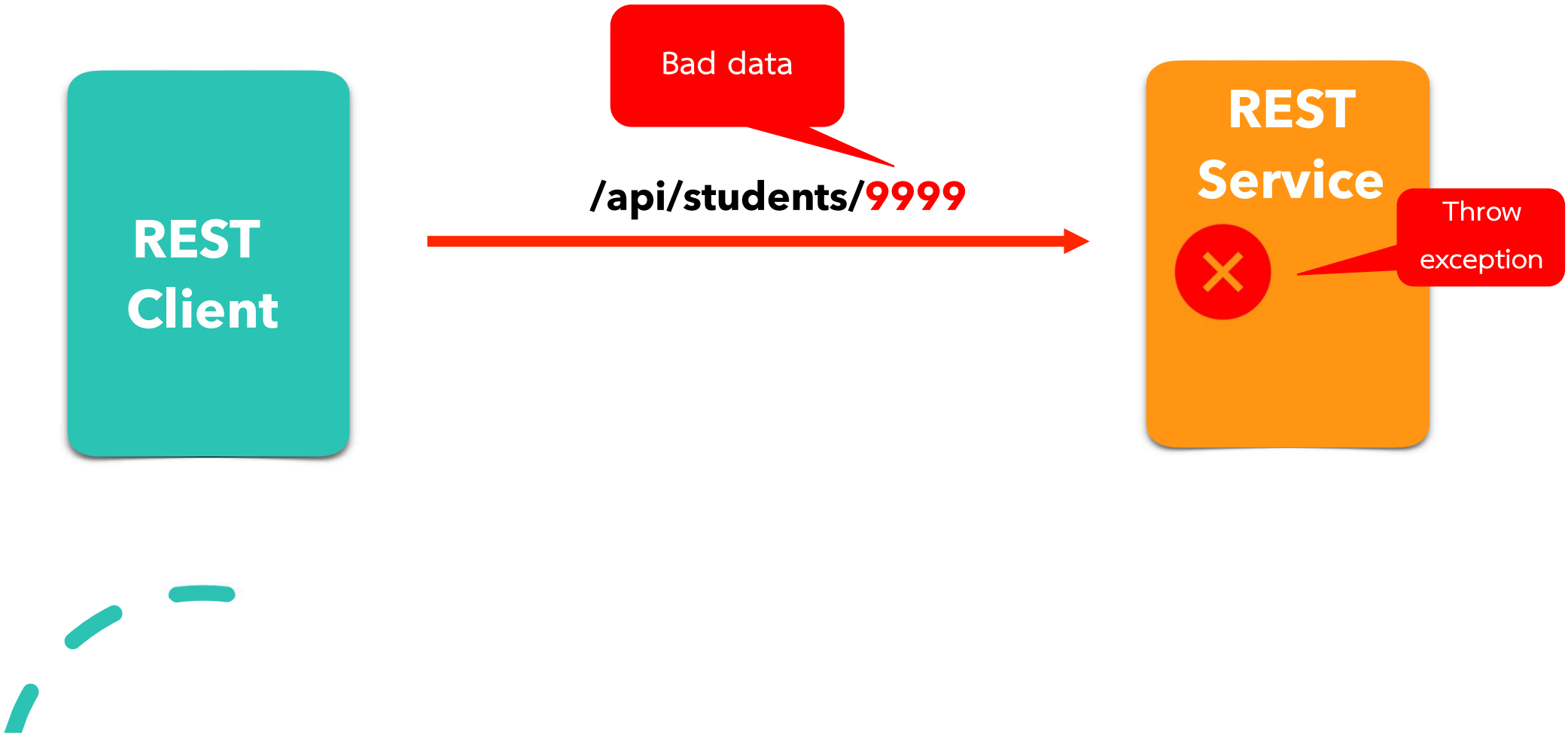
Spring REST Exception Handling



Spring REST Exception Handling



Spring REST Exception Handling

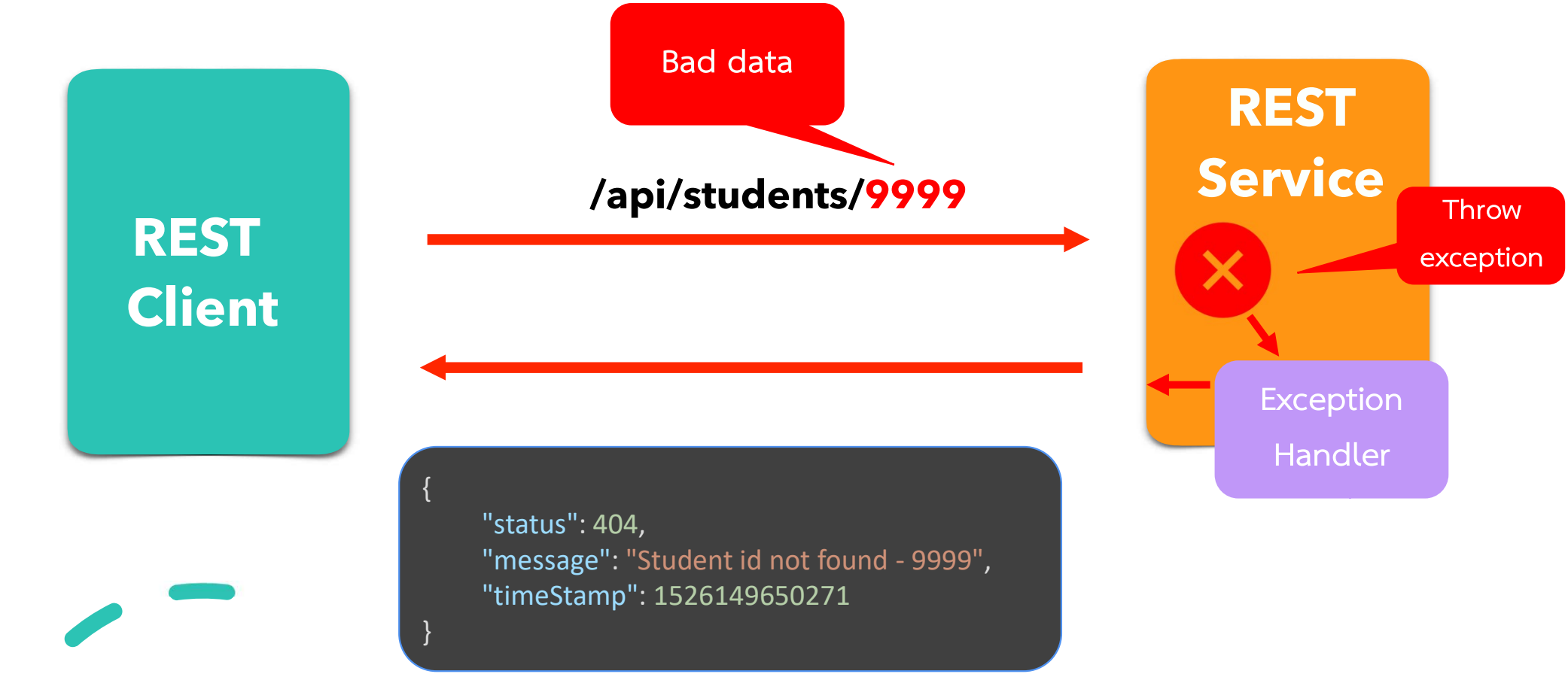


Spring REST Exception Handling

Spring REST Exception Handling

Spring REST Exception Handling

Spring REST Exception Handling

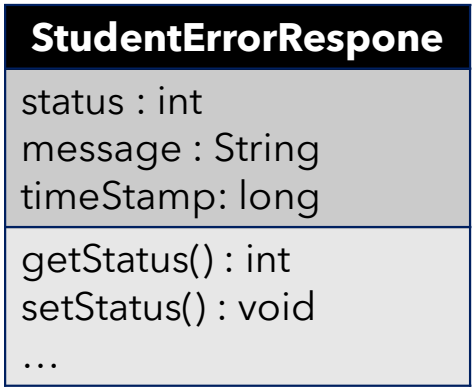


ขั้นตอนการพัฒนา

1. สร้าง custom error response class
2. สร้าง custom exception class
3. เพิ่ม Logic ใน REST service สำหรับการส่ง exception หากไม่เจอ
4. เพิ่ม exception handler โดยใช้ @ExceptionHandler

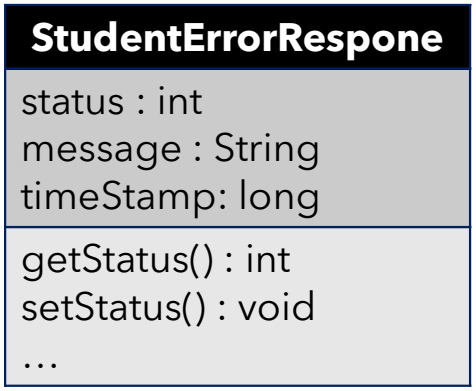
ขั้นแรก: สร้าง **custom error response class**

- custom error response class จะถูกส่งกลับไปยัง Client ในรูปแบบ JSON
- เราจะกำหนดเป็น Java class (POJO)



ขั้นแรก: สร้าง **custom error response class**

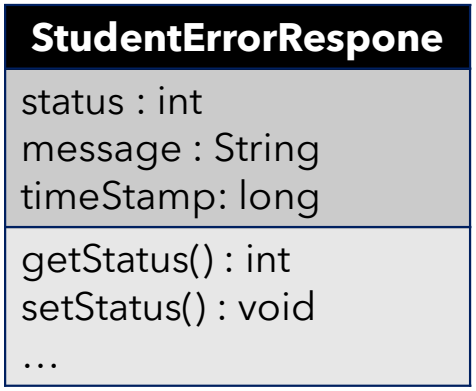
- custom error response class จะถูกส่งกลับไปยัง Client ในรูปแบบ JSON
- เราจะกำหนดเป็น Java class (POJO)
- Jackson จะจัดการแปลงเป็น JSON




```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

ขั้นแรก: สร้าง **custom error response class**

- custom error response class จะถูกส่งกลับไปยัง Client ในรูปแบบ JSON
- เราจะกำหนดเป็น Java class (POJO)
- Jackson จะจัดการแปลงเป็น JSON

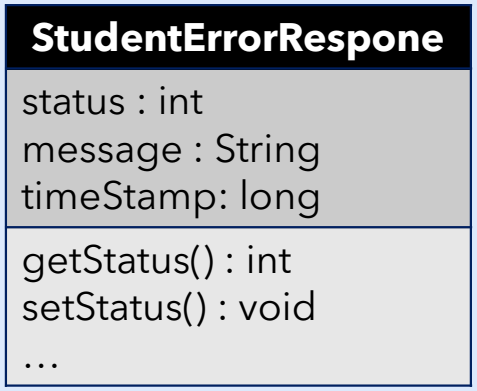


เราสามารถกำหนดเป็น **Fields** อะไรก็ได้

```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

ขั้นแรก: สร้าง custom error response class

```
public class StudentErrorResponse
{
    private int    status;
    private String message;
    private long   timeStamp;

    // constructors

    // getters / setters
}
```




```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

ขั้นตอนสอง: สร้าง **custom exception class**

- custom student exception จะถูกใช้โดย REST service ของเรา
- ในโค้ดของเรา หากเราไม่พบนักเรียน เราจะ throw an exception
- ต้อง custom student exception class
  - `StudentNotFoundException`

ขั้นตอนสอง: สร้าง custom exception class

```
public class StudentNotFoundException extends RuntimeException
{
    public StudentNotFoundException( String message )
    {
       super( message );
    }
}
```

เรียก super class constructor

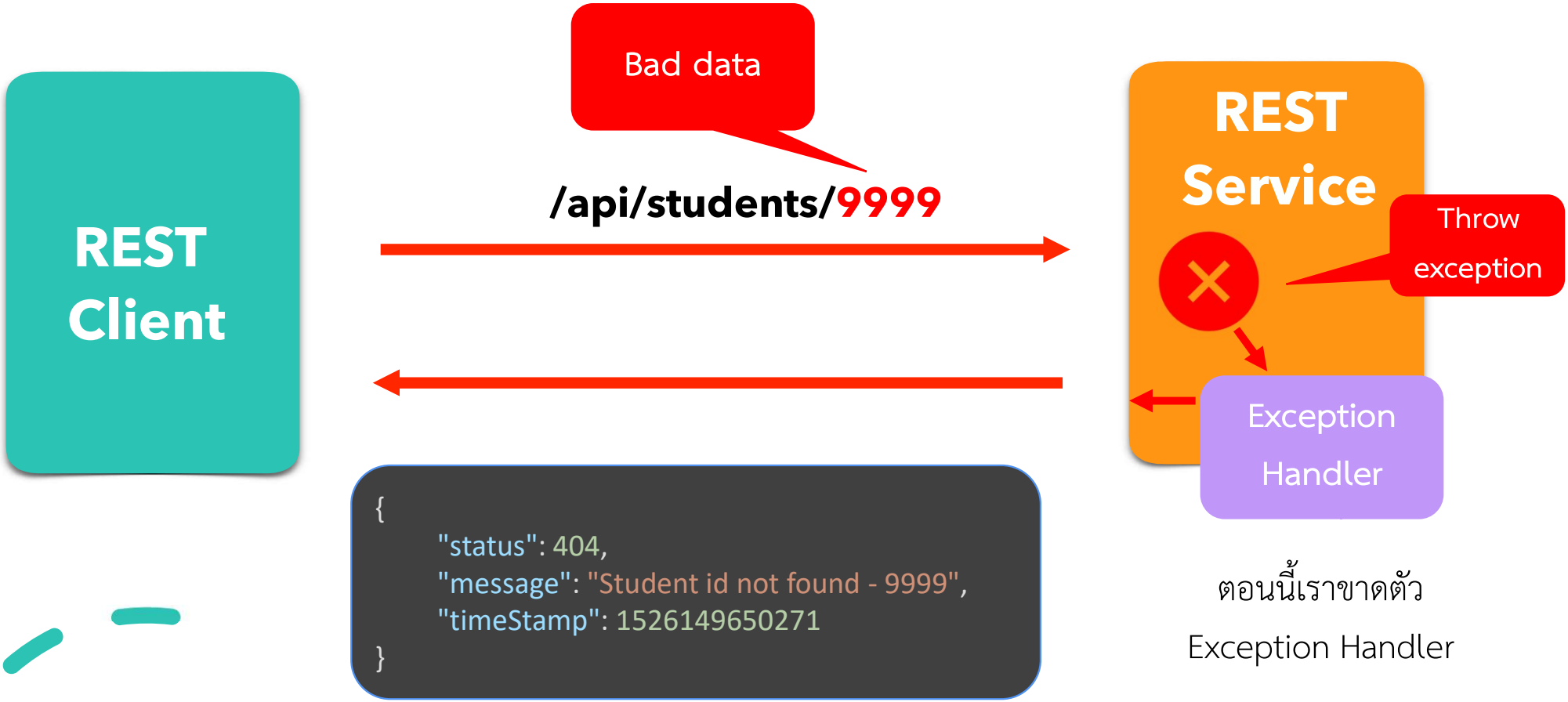
ขั้นตอนสาม: แก้ไข REST service เพื่อส่ง exception

```java
@RestController
@RequestMapping ( "/api" )
public class StudentRestController
{
    @GetMapping("/students/{studentId}")
    public Student getStudent(@PathVariable int studentId) {

       // check the studentId against list size

       if ( (studentId >= theStudents.size()) || (studentId < 0) ) {
          throw new StudentNotFoundException("Student id not found - " + studentId);
       }

       return theStudents.get(studentId);
    }
    ...
}
```

Happy path

Throw exception

Spring REST Exception Handling



ตอนนี้เราขาดตัว Exception Handler

ขั้นตอนสี่: เพิ่ม exception handler method

- กำหนด exception handler method(s) ด้วย annotations @ExceptionHandler
- Exception handler จะส่งคืน ResponseEntity
- ResponseEntity เป็น wrapper สำหรับ HTTP response object
- ResponseEntity จะมี properties หลายอย่างให้เราใช้:
  - HTTP status code, HTTP headers and Response body

ขั้นตอนสี่: เพิ่ม exception handler method

```
@RequestMapping("/api")
public class StudentRestController {

    ...

    @ExceptionHandler
    public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

       StudentErrorResponse error = new StudentErrorResponse();
       error.setStatus( HttpStatus.NOT_FOUND.value());
       error.setMessage(exc.getMessage());
       error.setTimeStamp(System.currentTimeMillis());

       return new ResponseEntity <>(error, HttpStatus.NOT_FOUND);
    }
}
```

ขั้นตอนสี่: เพิ่ม exception handler method

บอกว่าเป็น Exception handler method

```
@Reque      ing("/api")
public      s StudentRestController {

    ...

    @ExceptionHandler
    public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

       StudentErrorResponse error = new StudentErrorResponse();
       error.setStatus( HttpStatus.NOT_FOUND.value());
       error.setMessage(exc.getMessage());
       error.setTimeStamp(System.currentTimeMillis());

       return new ResponseEntity <>(error, HttpStatus.NOT_FOUND);
    }
}
```

ขั้นตอนสี่: เพิ่ม exception handler method

บอกว่าเป็น Exception handler method

Type ของ response body

```
@Reque      ing("/api")
public      s StudentRestController {

    ...

    @ExceptionHandler
    public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

       StudentErrorResponse error = new StudentErrorResponse();
       error.setStatus( HttpStatus.NOT_FOUND.value());
       error.setMessage(exc.getMessage());
       error.setTimeStamp(System.currentTimeMillis());

       return new ResponseEntity <>(error, HttpStatus.NOT_FOUND);
    }
}
```

ขั้นตอนสี่: เพิ่ม exception handler method

บอกว่าเป็น Exception handler method

Type ของ response body

Type ของ Exception ในการ handle / catch

```
@Reque      ing("/api")
public      s StudentRestController {

    ...

    @ExceptionHandler
    public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

       StudentErrorResponse error = new StudentErrorResponse();
       error.setStatus( HttpStatus.NOT_FOUND.value());
       error.setMessage(exc.getMessage());
       error.setTimeStamp(System.currentTimeMillis());

       return new ResponseEntity <>(error, HttpStatus.NOT_FOUND);
    }
}
```

ขั้นตอนสี่: เพิ่ม exception handler method

StudentErrorRespone
status : int message : String timeStamp: long
getStatus() : int setStatus() : void …

```
@RequestMapping("/api")
public class StudentRestController {

    ...

    @ExceptionHandler
    public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

       StudentErrorResponse error = new StudentErrorResponse();
       error.setStatus( HttpStatus.NOT_FOUND.value());
       error.setMessage(exc.getMessage());
       error.setTimeStamp(System.currentTimeMillis());

       return new ResponseEntity <>(error, HttpStatus.NOT_FOUND);
    }
}
```

ขั้นตอนสี่: เพิ่ม exception handler method

StudentErrorRespone

status : int
message : String
timeStamp: long

getStatus() : int
setStatus() : void
…

```
@RequestMapping("/api")
public class StudentRestController {

    ...

    @ExceptionHandler
    public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

       StudentErrorResponse error = new StudentErrorResponse();
       error.setStatus( HttpStatus.NOT_FOUND.value());
       error.setMessage(exc.getMessage());
       error.setTimeStamp(System.currentTimeMillis());

       return new ResponseEntity <>(error, HttpStatus.NOT_FOUND);
    }
}
```

Body

Status Code

ขั้นตอนสี่: เพิ่ม exception handler method

```
@RequestMapping("/api")
public class StudentRestController {

    ...

    @ExceptionHandler
    public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

       StudentErrorResponse error = new StudentErrorResponse();
       error.setStatus( HttpStatus.NOT_FOUND.value());
       error.setMessage(exc.getMessage());
       error.setTimeStamp(System.currentTimeMillis());

       return new ResponseEntity <>(error, HttpStatus.NOT_FOUND);
    }
}
```

Body

Status Code

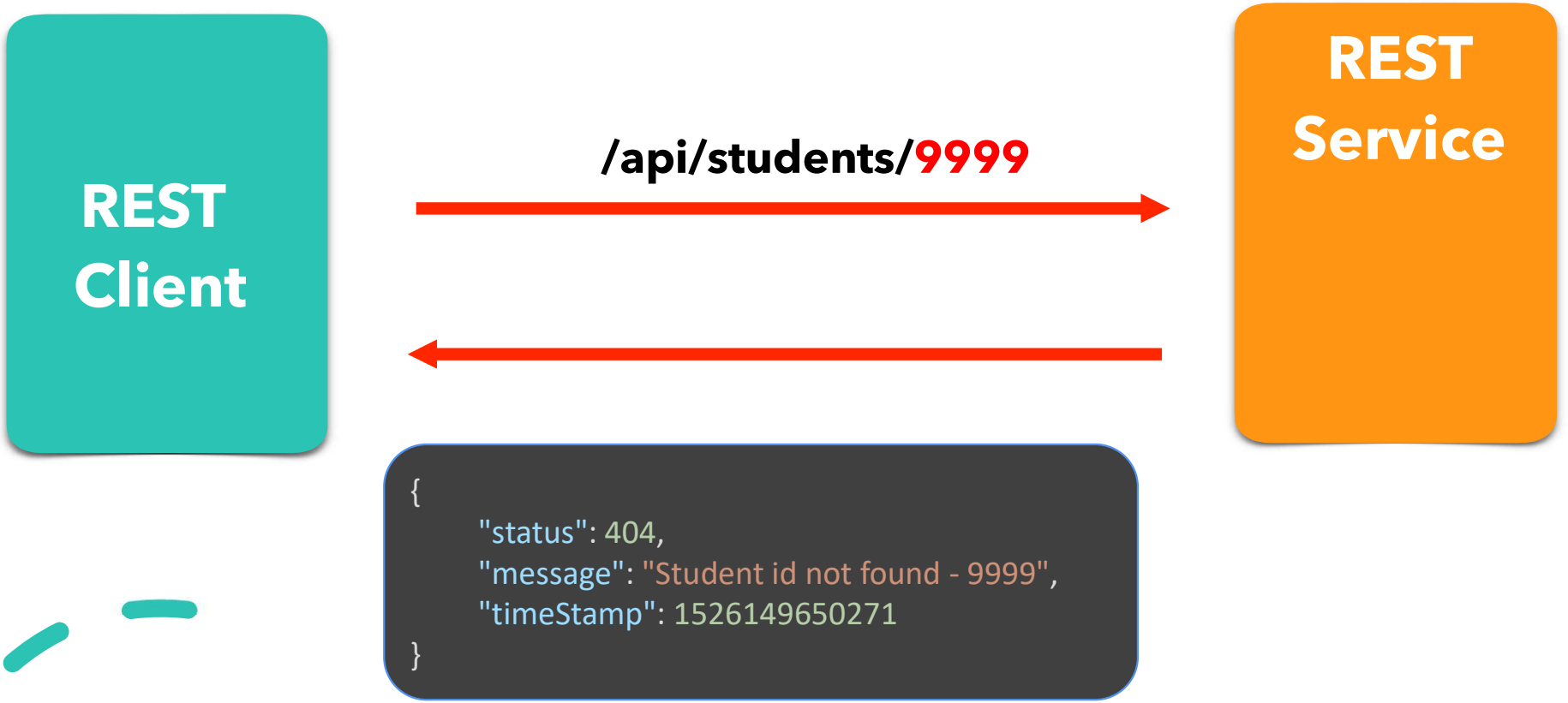
StudentErrorRespone

status : int
message : String
timeStamp: long

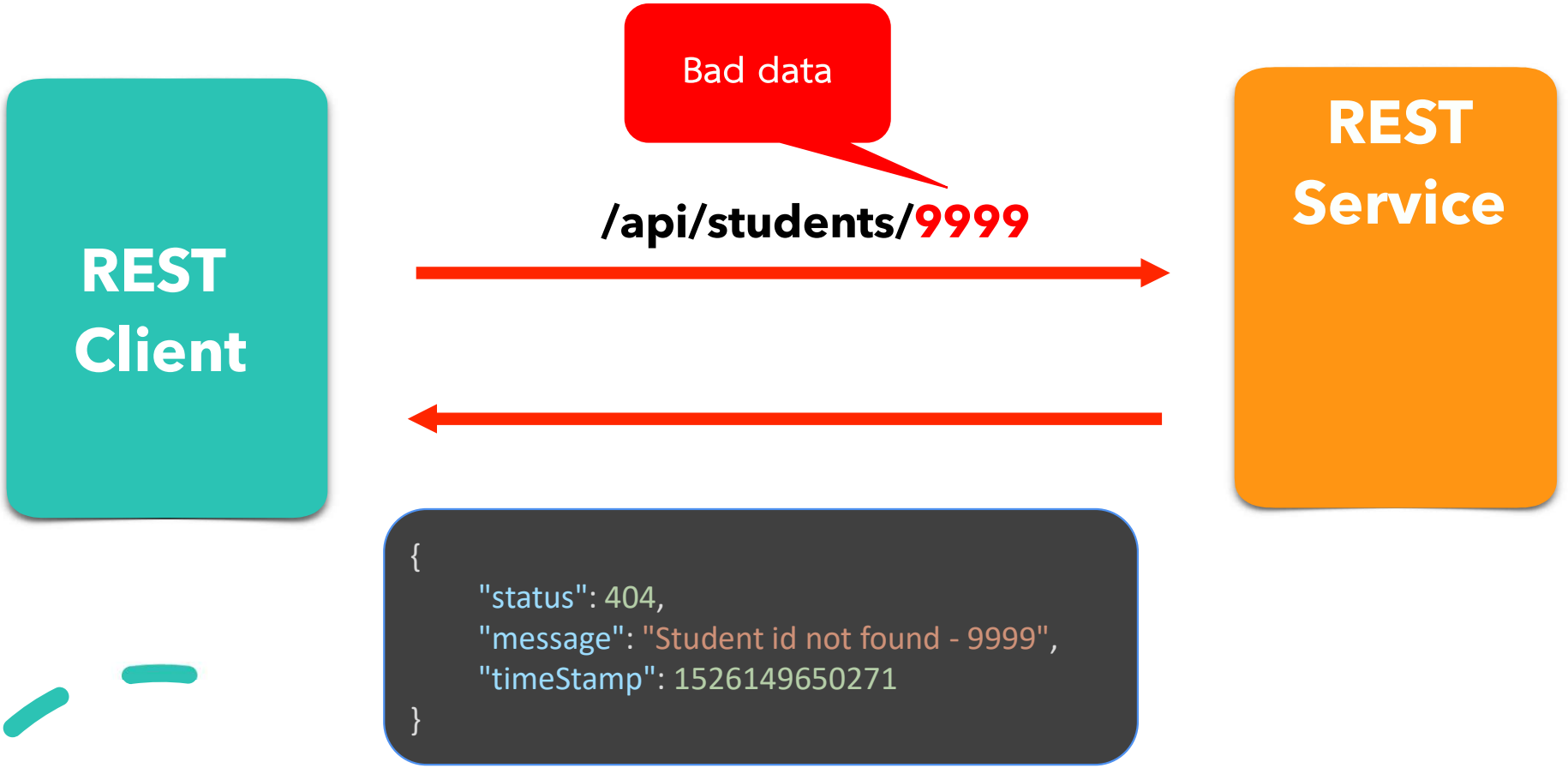
getStatus() : int
setStatus() : void
…

```
{
        "status": 404,
        "message": "Student id not found - 9999",
        "timeStamp": 1526149650271
}
```

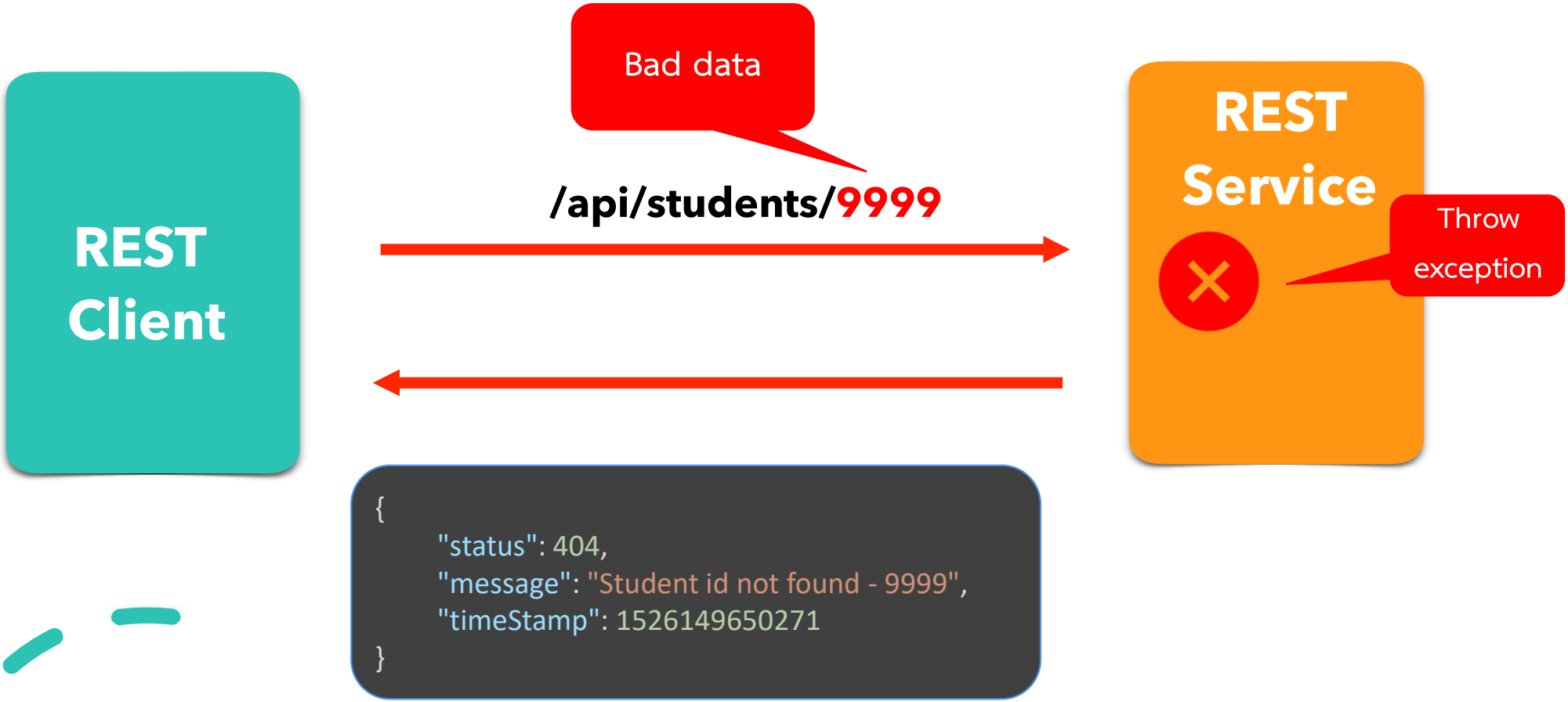
Spring REST Exception Handling



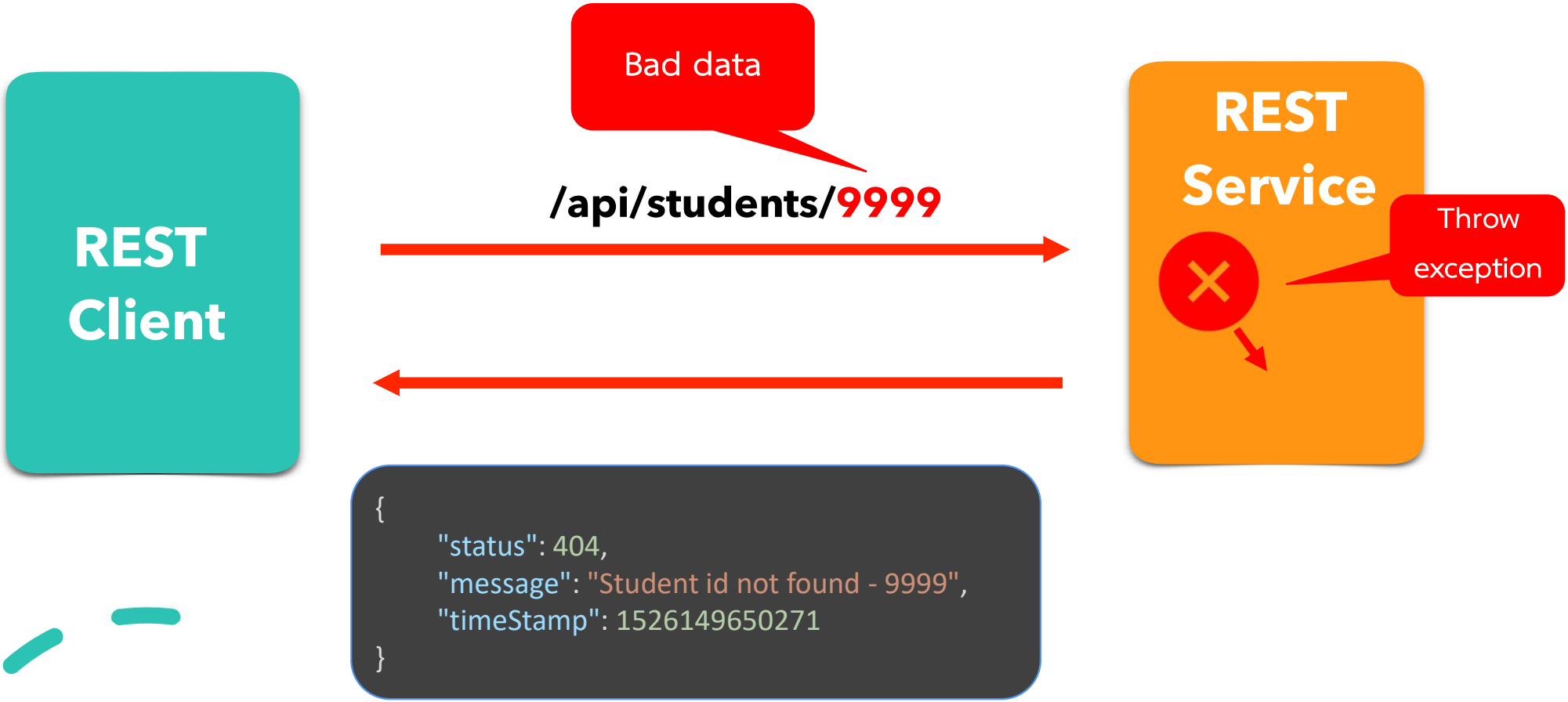
Spring REST Exception Handling



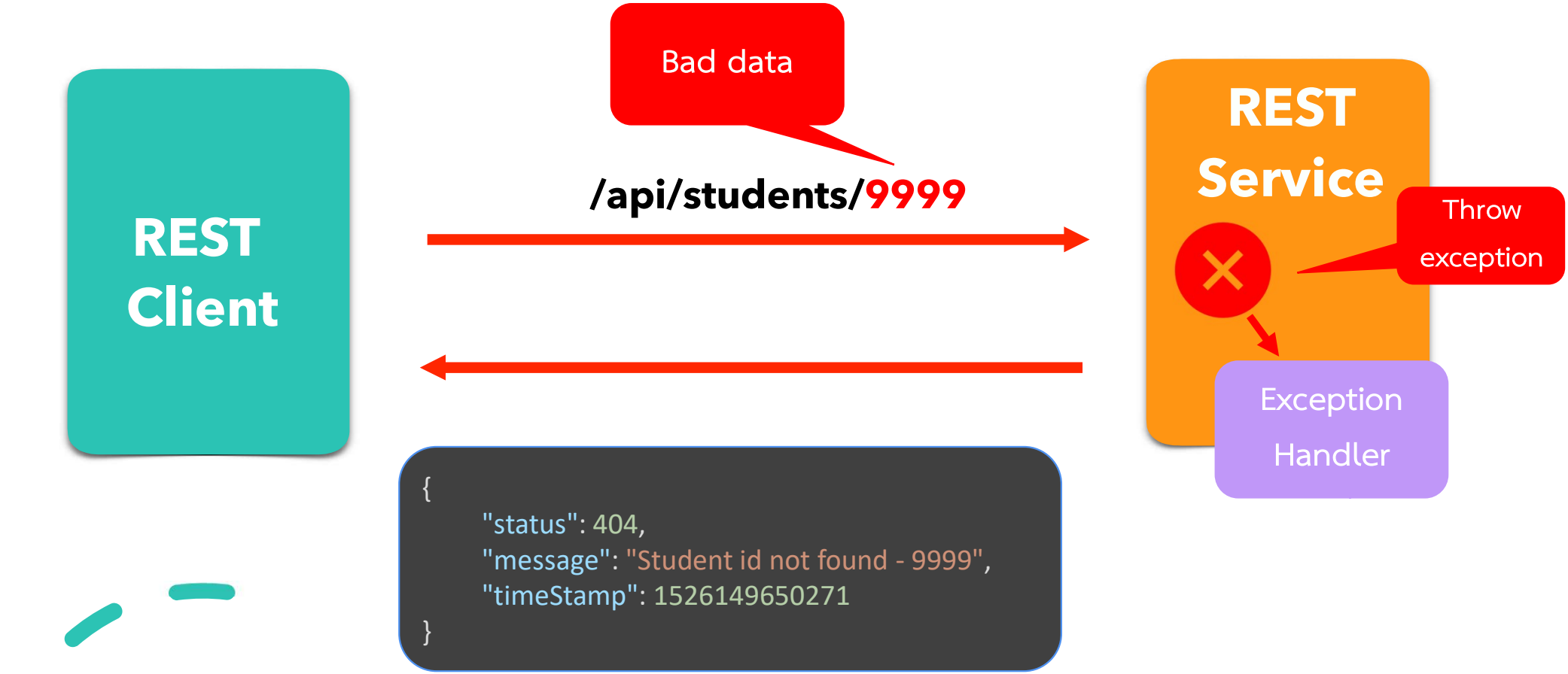
Spring REST Exception Handling



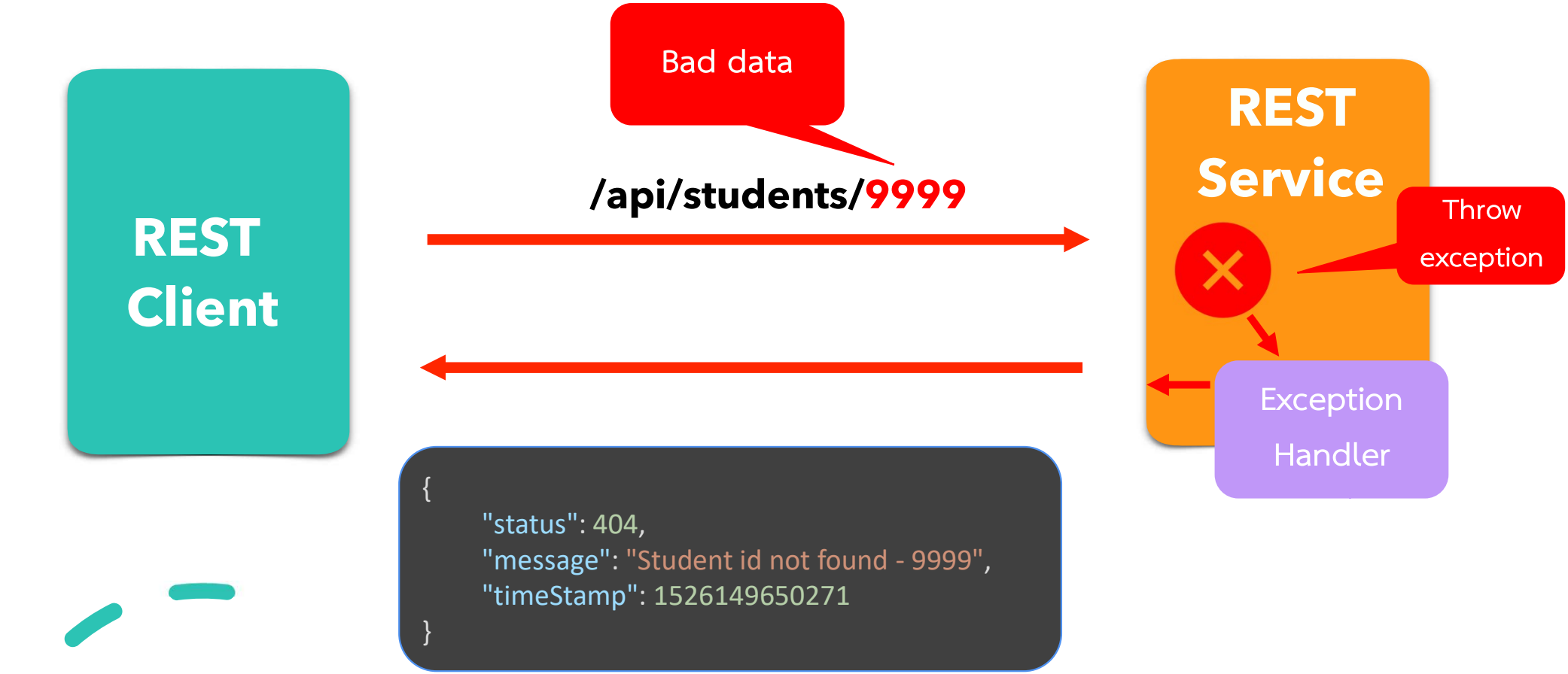
Spring REST Exception Handling



Spring REST Exception Handling



Spring REST Exception Handling



Spring REST – Exception handler

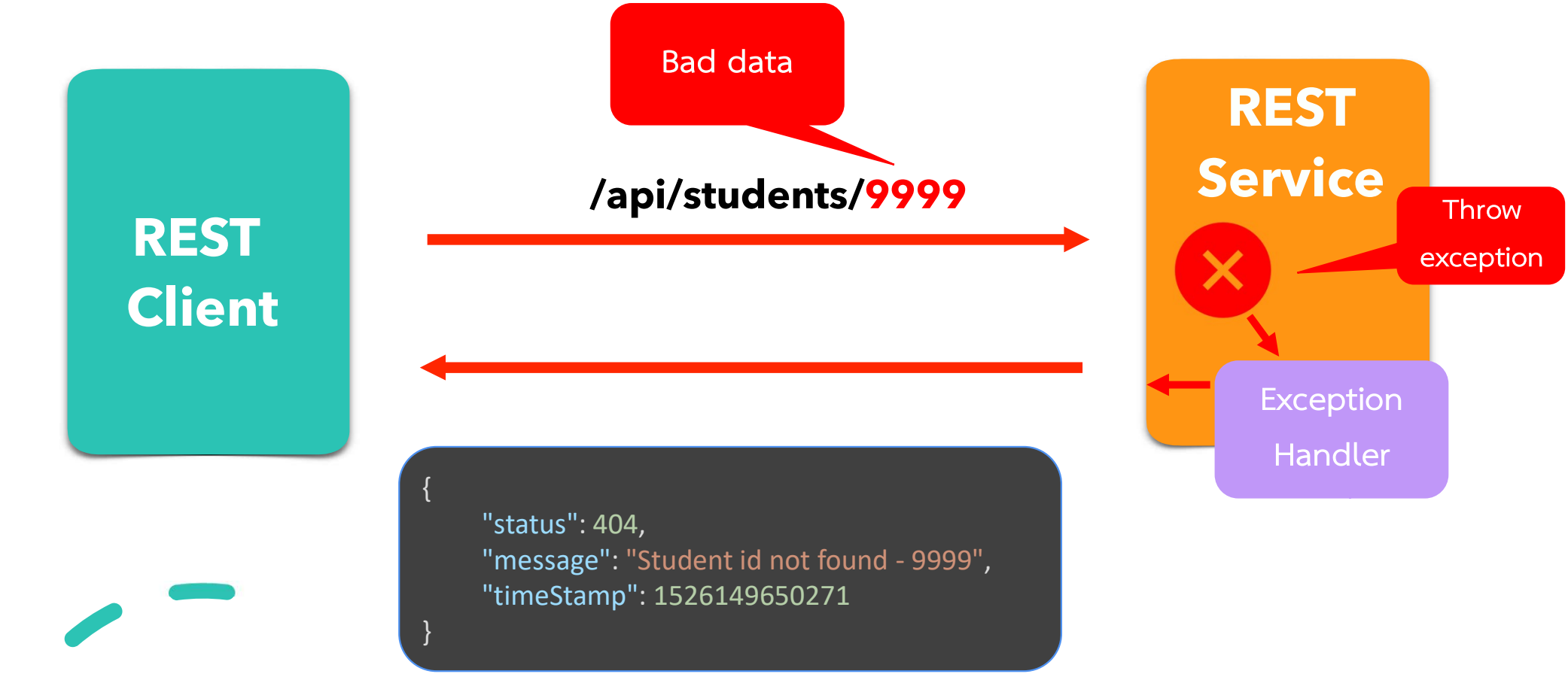
- Mini-workshop: Spring REST – Exception handler

- Ref: lab-s5-l4

Spring REST – Global Exception Handling

Chapter 9

Spring REST Exception Handling



มันได้ผล แต่...

- Exception handler code มีไว้ Handle REST Controller เป็นตัว ๆ ไปเท่านั้น
- Controller อื่นก็ต้องเขียน Handler Exception ของตัวเองขึ้นมา

- เราจึงต้องการ global exception handlers
  - Controller ทุกตัวใช้โค้ดเดียวกัน (มีการ reuse)
  - จัดการง่ายเพราะรวมอยู่ในที่เดียว

Project ขนาดใหญ่ จะมี Controller เยอะมาก เราทำแบบนี้ทุก Controller ไม่ไหว

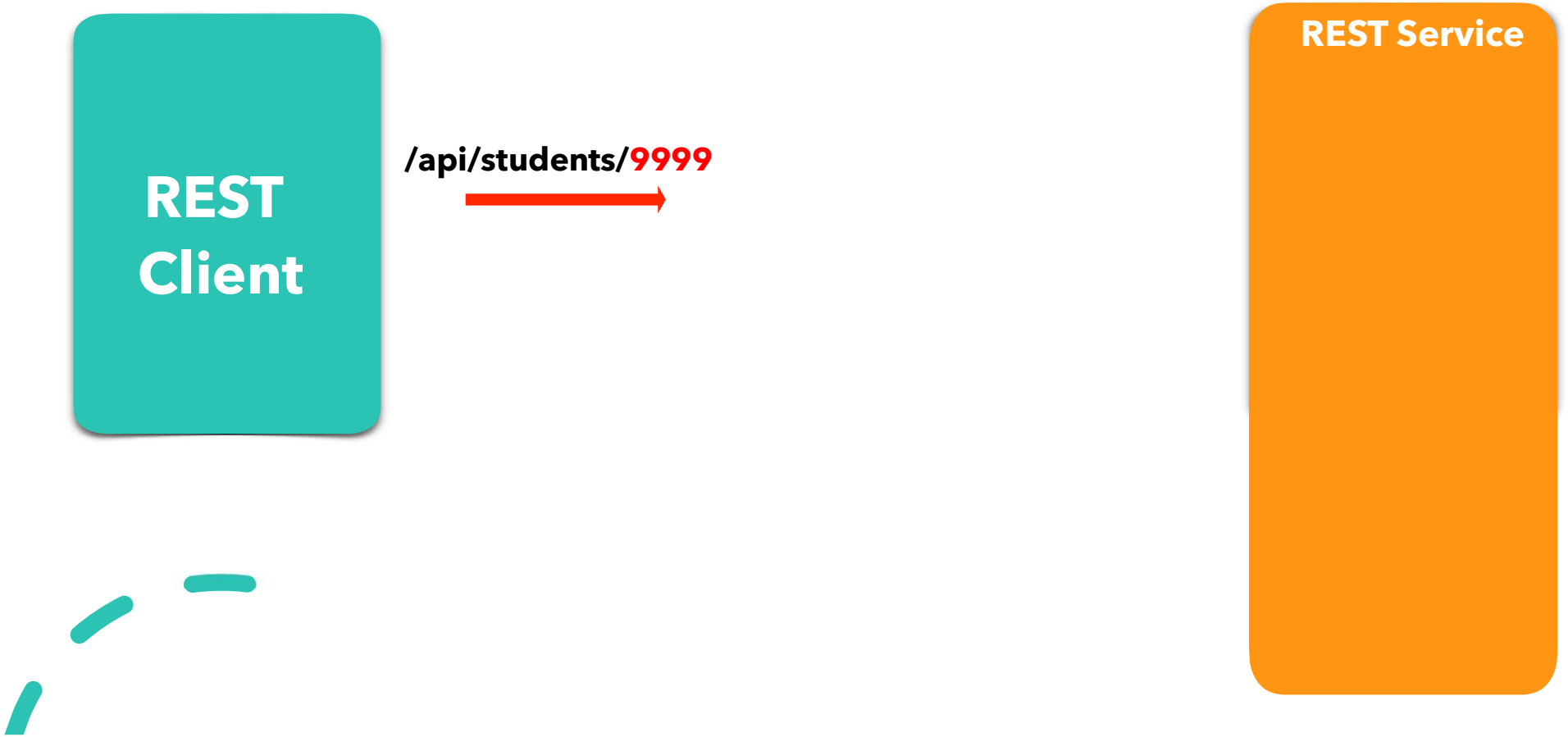
Spring @ControllerAdvice

- @ControllerAdvice คล้ายกับ interceptor/filter
- Pre-process ตัว Request ก่อนที่จะไปยัง Controller
- Post-process ตัว Response จาก handle exceptions
- เหมาะสำหรับใช้กับ Global exception handling

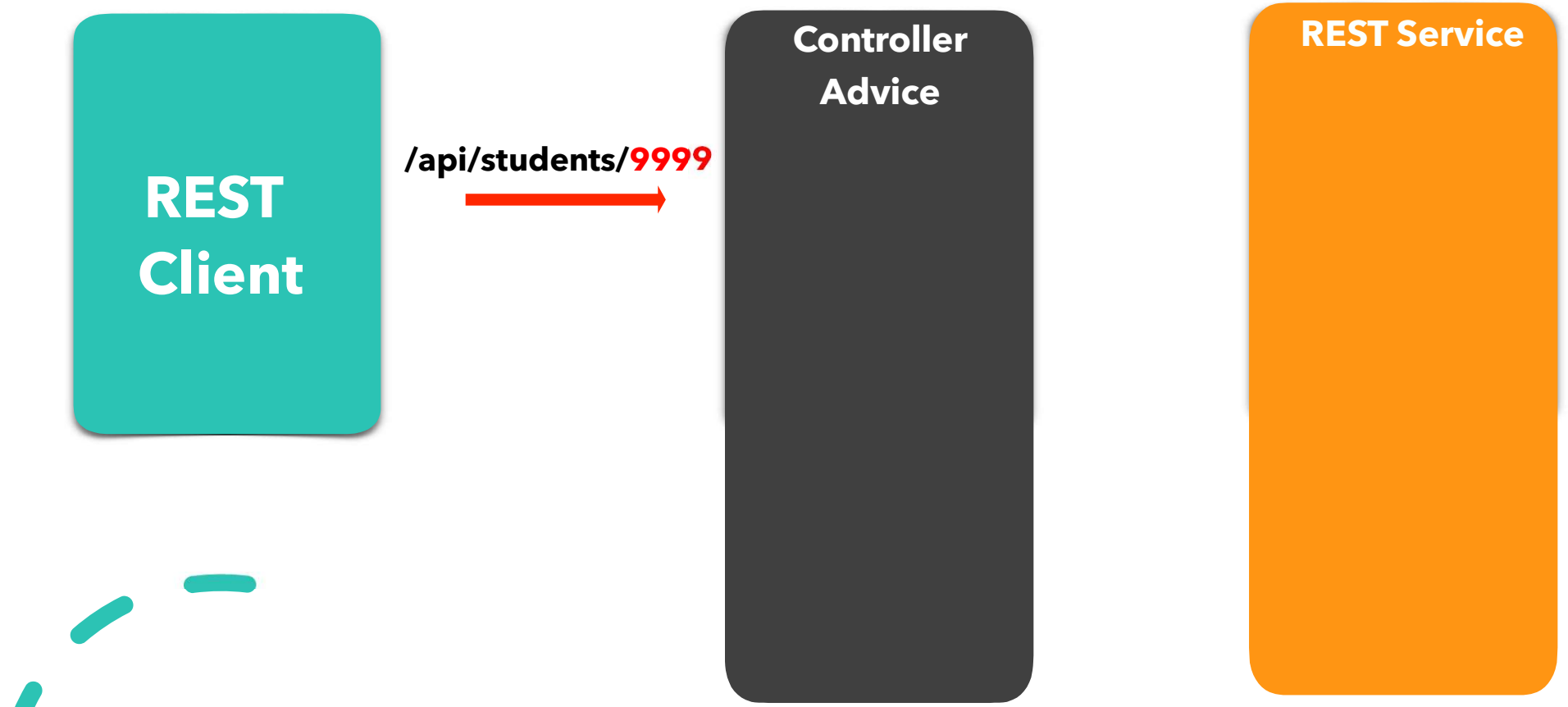
เป็น Concept ของ AOP

Spring REST Exception Handling

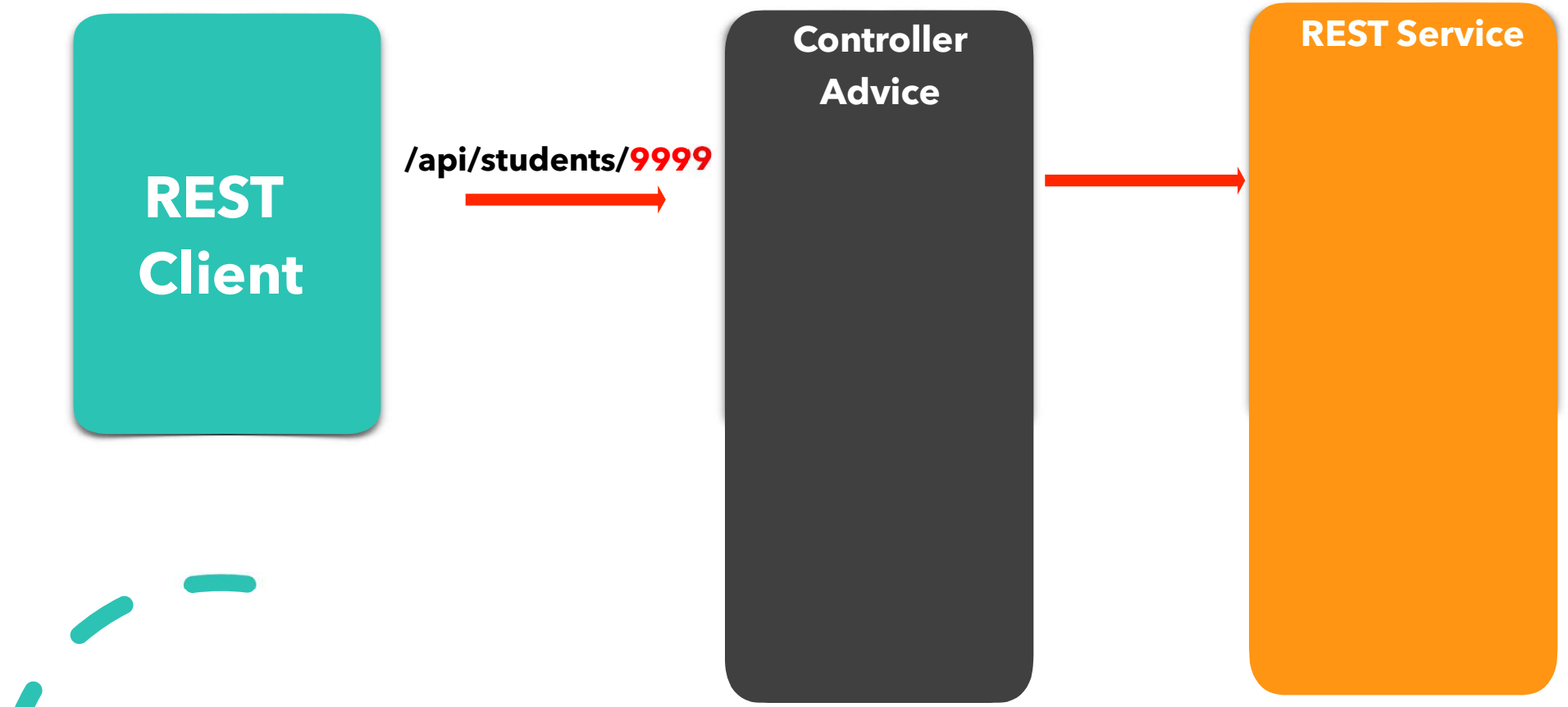
Spring REST Exception Handling



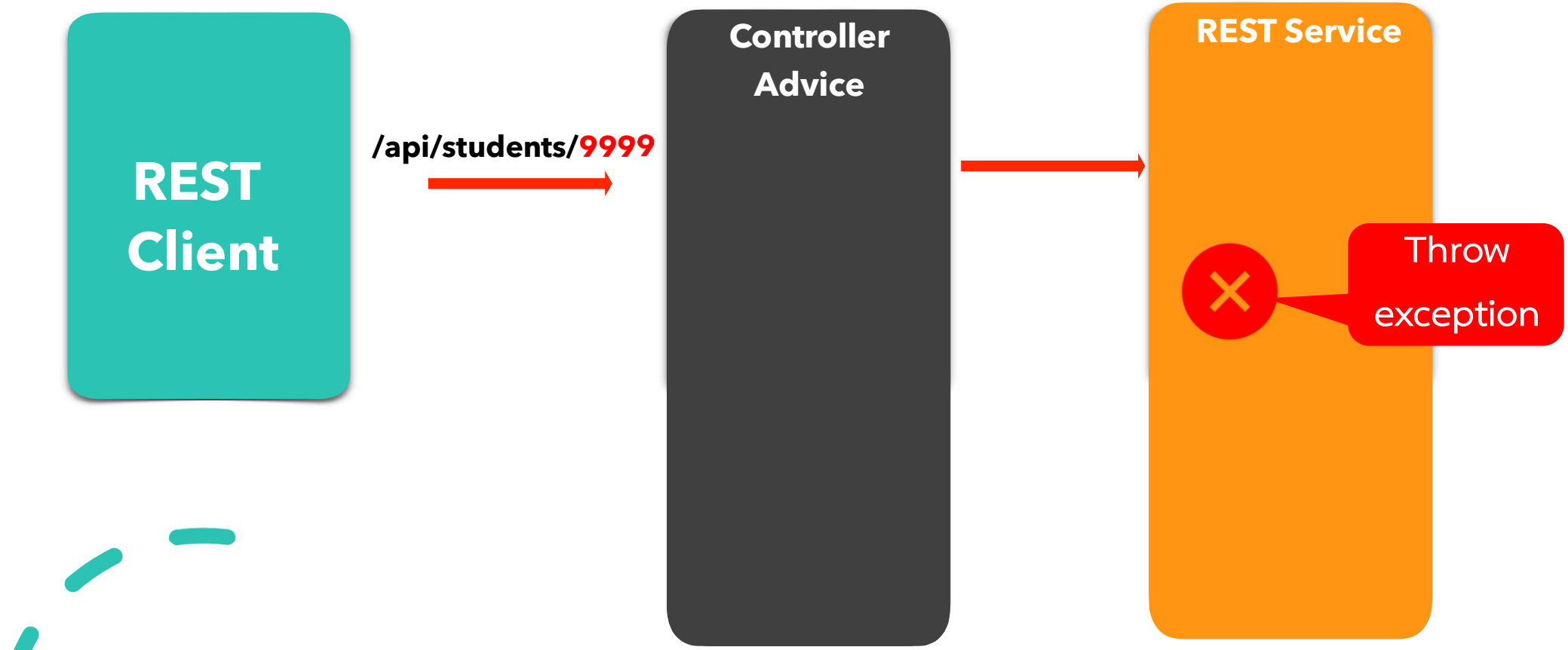
Spring REST Exception Handling



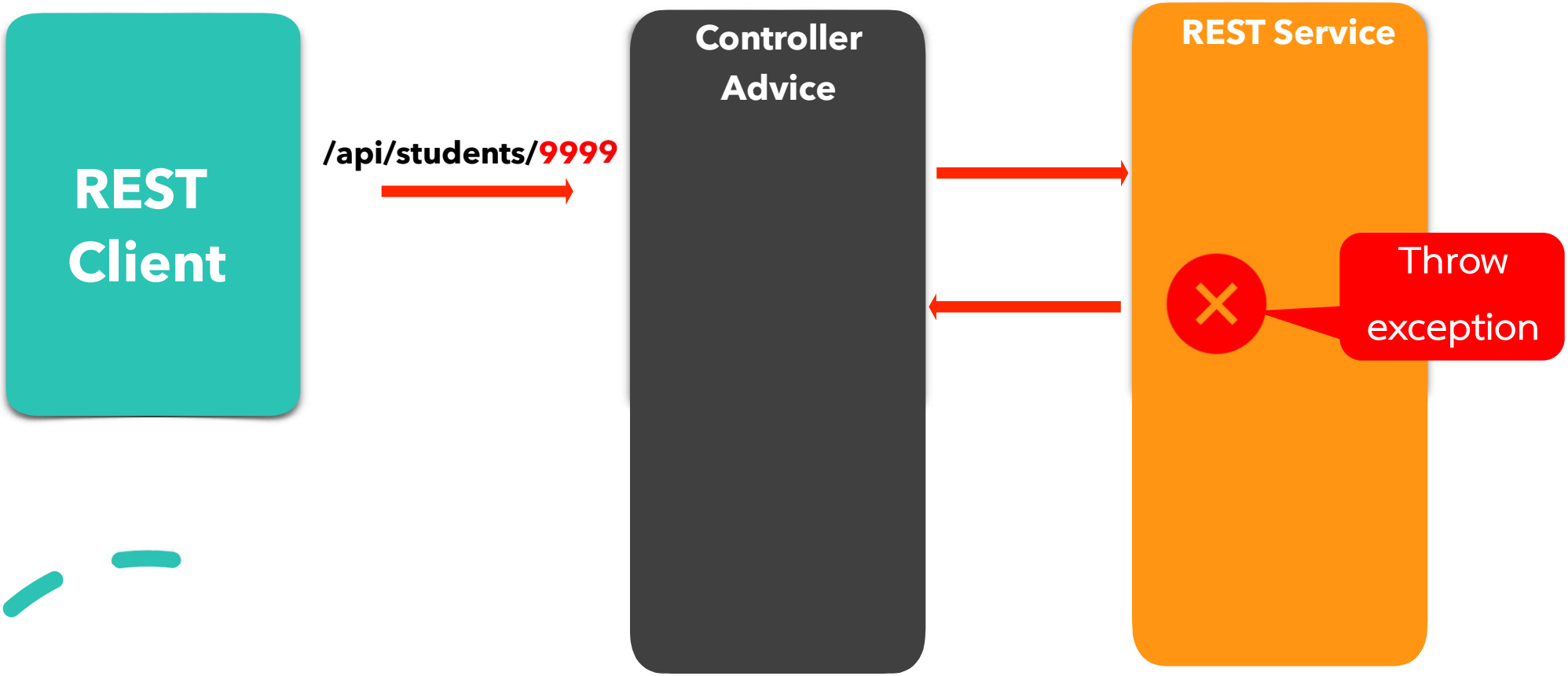
Spring REST Exception Handling



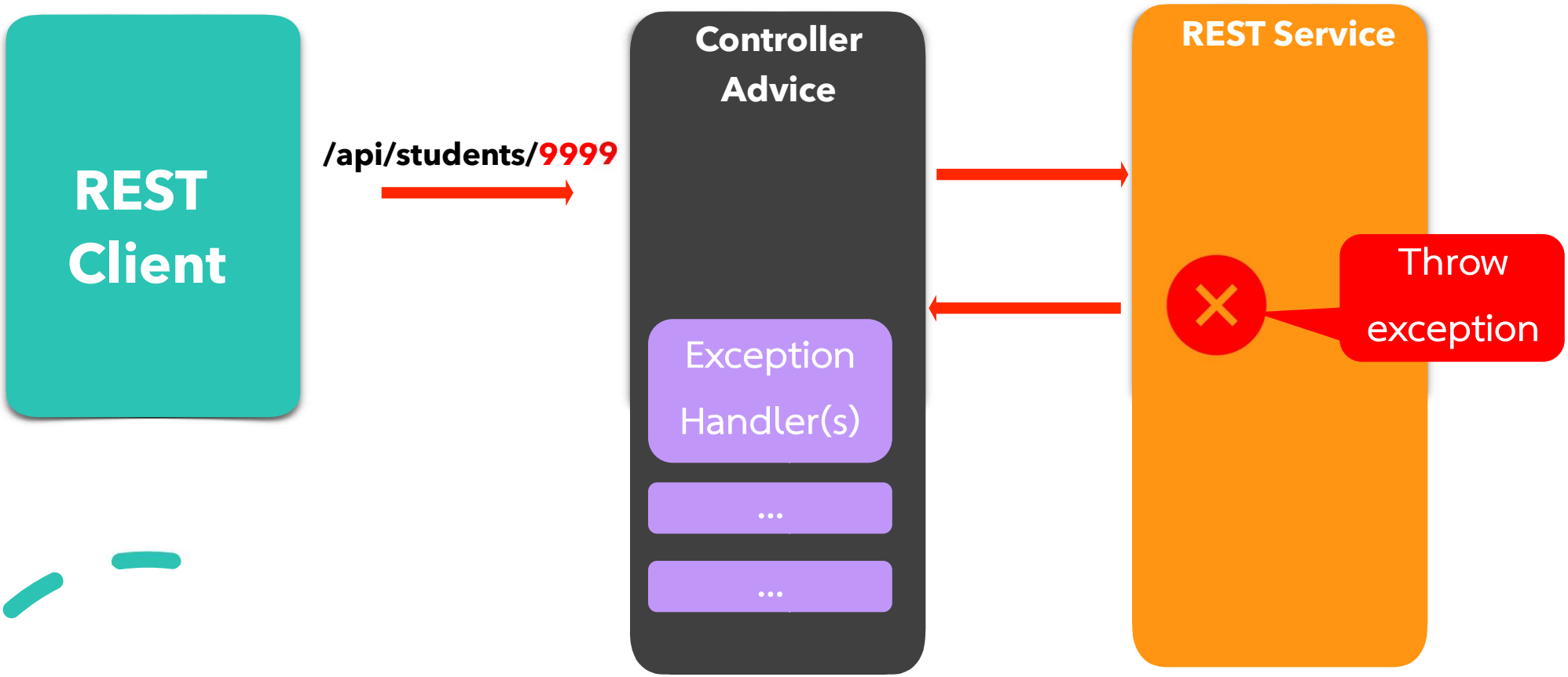
Spring REST Exception Handling



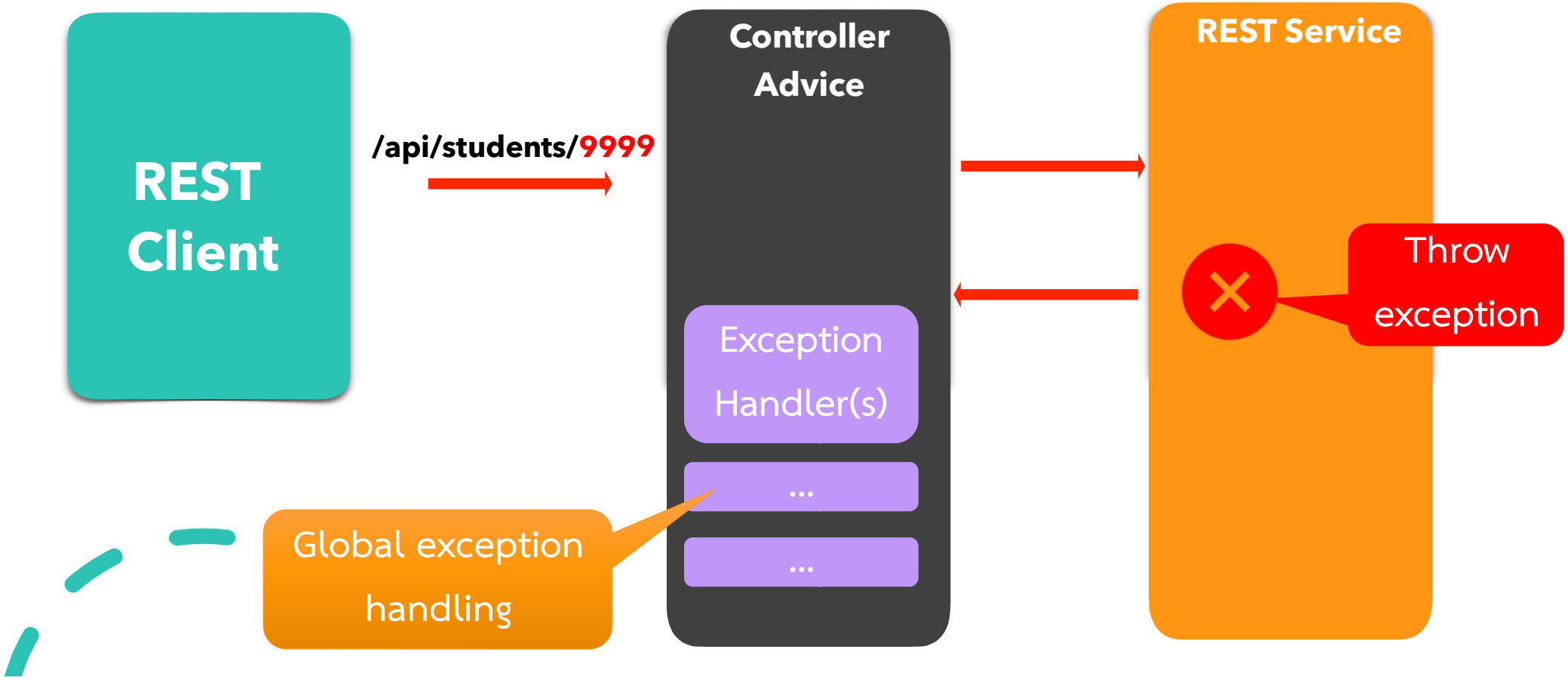
Spring REST Exception Handling



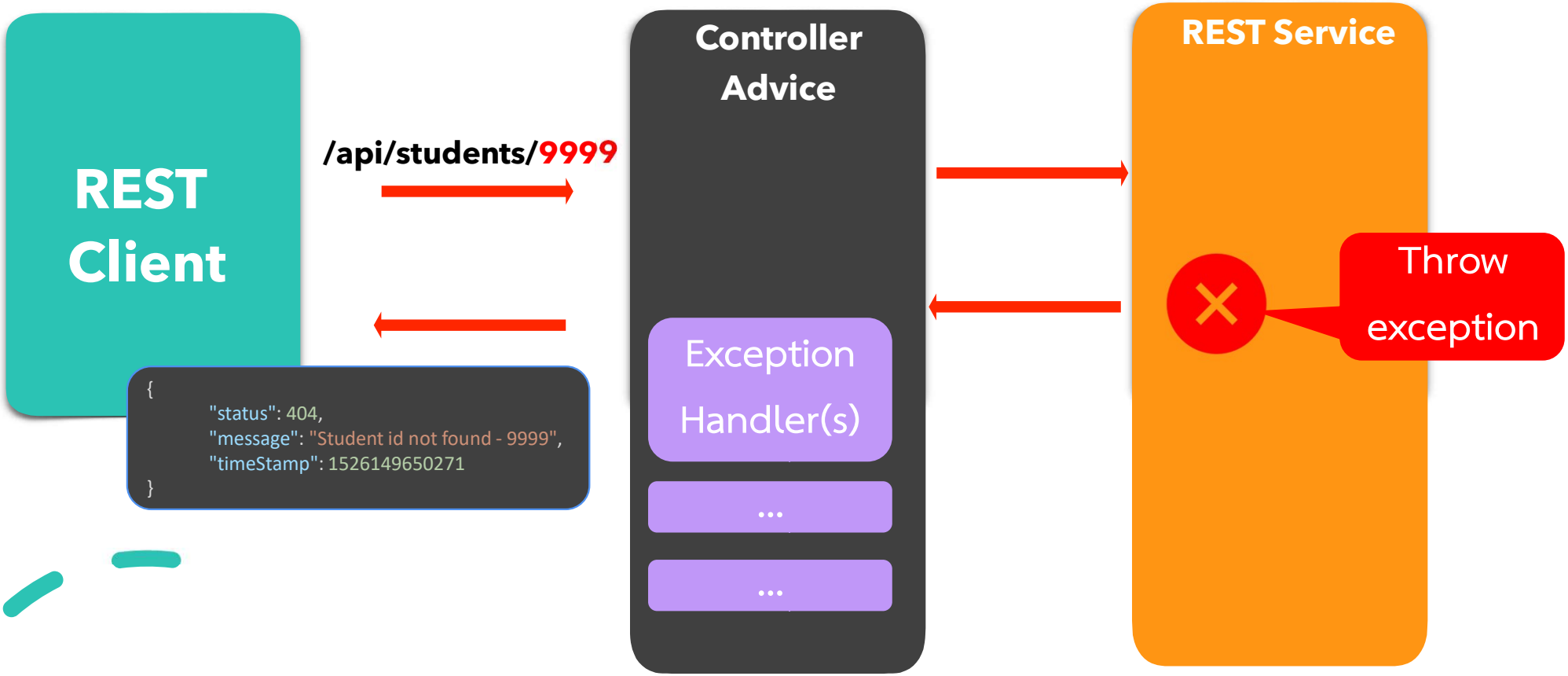
Spring REST Exception Handling



Spring REST Exception Handling



Spring REST Exception Handling



ขั้นตอนการพัฒนา

1. สร้าง @ControllerAdvice ใหม่
2. Refactor REST Service … ลบ exception handling code
3. เพิ่ม exception handling code ใน @ControllerAdvice แทน

ขั้นแรก: สร้าง @ControllerAdvice ใหม่

```
@ControllerAdvice
public class StudentRestExceptionHandler {
    ...
}
```

ขั้นสอง: Refactor ลบ exception handling code

```
@RestController
@RequestMapping ( "/api" )
public class StudentRestController
{
    ...
    @ExceptionHandler
    public ResponseEntity < StudentErrorResponse > handleException( StudentNotFoundException ex )
    {
       StudentErrorResponse error = new StudentErrorResponse();

       error.setStatus( HttpStatus.NOT_FOUND.value() );
       error.setMessage( ex.getMessage() );
       error.setTimeStamp( System.currentTimeMillis() );

       return new ResponseEntity <>( error , HttpStatus.NOT_FOUND );
    }
}
```

ขั้นสอง: Refactor ลบ exception handling code

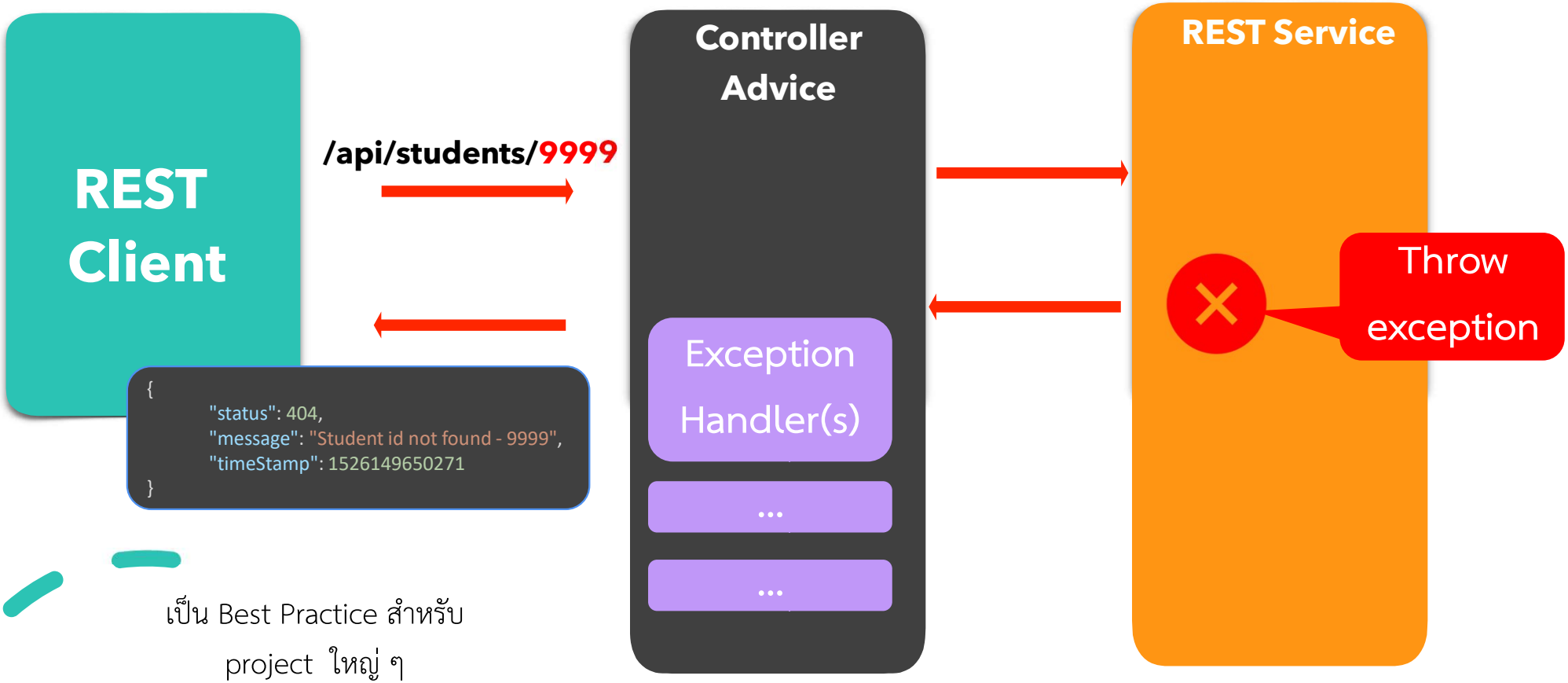
```
@RestController
@RequestMapping ( "/api" )
public class StudentRestController
{
    ...
    @ExceptionHandler
    public ResponseEntity < StudentError                xception( StudentNotFoundException ex )
    {
       StudentErrorResponse error = ne          ntE   rR      ();

       error.setStatus( HttpStatus.NO           alue()
       error.setMessage( ex.getMessage
       error.setTimeStamp( System.curre

       return new ResponseEntity <>( error                NOT_FOUND );
    }
```

ขั้นสาม: เพิ่ม exception handling code ใน @ControllerAdvice

เหมือนกับใน Controller เลย

```
@ControllerAdvice
public class StudentRestExceptionHandler {

    @ExceptionHandler
    public ResponseEntity < StudentErrorResponse > handleException( StudentNotFoundException ex )
    {
        StudentErrorResponse error = new StudentErrorResponse();

        error.setStatus( HttpStatus.NOT_FOUND.value() );
        error.setMessage( ex.getMessage() );
        error.setTimeStamp( System.currentTimeMillis() );

        return new ResponseEntity <>( error , HttpStatus.NOT_FOUND );
    }
}
```

Spring REST Exception Handling



เป็น Best Practice สำหรับ project ใหญ่ ๆ

Spring REST – Exception handler

- Mini-workshop: Spring REST – Exception handler

- Ref: lab-s5-l5

การออกแบบ Spring REST API

Chapter 10

การออกแบบ REST API

- สำหรับโปรเจกต์แบบเรียลไทม์ ใครจะใช้ API ของเราบ้าง ?
- นอกจากนี้ พวกเขาจะใช้ API ของเราได้อย่างไร ?
- ออกแบบ API โดยอ้างอิงจาก requirement

ขั้นตอนการออกแบบ API

1. รีวิว API Requirement
2. กำหนด resource และ entity ที่ต้องใช้
3. กำหนด methods HTTP ให้สอดคล้องกับ action ที่ทำกับ resource

ขั้นตอนแรก: รีวิว API Requirement

จากหัวหน้า

สร้าง **REST API** สำหรับเก็บข้อมูลพนักงาน

สิ่งที่ **REST** ควรจะทำได้

- ให้ข้อมูล รายชื่อพนักงานได้หลายคน
- ให้ข้อมูล พนักงานคนเดียว ตามรหัส
- เพิ่มพนักงานใหม่
- เปลี่ยนแปลงพนักงาน
- ลบพนักงาน

ขั้นตอนสอง:กำหนด **resource** และ **entity** ที่ต้องใช้

- ในการระบุ resource หลัก / entity ให้มองหา "คำนาม" ที่เด่นชัดที่สุด
- สำหรับโปรเจกต์ของเราคือ “employee”
- Convention คือใช้ plural ของ resource / entity: **employees**

```
/api/employees
```

ขั้นตอนสาม:

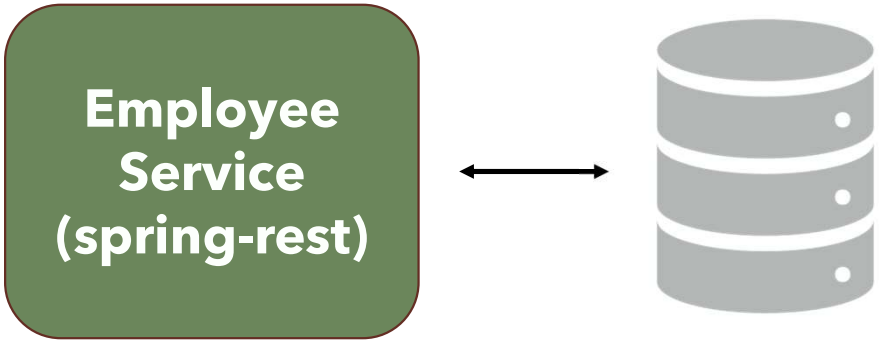
ใช้ methods HTTP เพื่อกำหนดการดำเนินการกับresource

HTTP Method	CRUD Action
POST	สร้าง **entity** ใหม่ (Create)
GET	อ่านรายชื่อ **entities** หรือ **entity** เดียว (Read)
PUT	แก้ไขข้อมูล **entity** ที่มีอยู่ (Update)
DELETE	ลบข้อมูล entity ที่มีอยู่ (Delete)

Full CRUD

Employee Real-Time Project

HTTP Method	CRUD Action	CRUD Action
POST	/api/employees	สร้าง พนักงาน ใหม่ (Create)
GET	/api/employees	อ่านรายชื่อพนักงานหลายคน (Read)
GET	/api/employees/{employeeId}	อ่านรายชื่อพนักงานคนเดียว (Read)
PUT	/api/employees	แก้ไขข้อมูลพนักงานที่มีอยู่ (Update)
DELETE	/api/employees/{employeeId}	ลบข้อมูลพนักงานที่มีอยู่ (Delete)



Anti-Patterns

- ไม่ควรทำสิ่งนี้ … รูปแบบนี้เป็นรูปแบบที่ขัดกับ REST เป็น bad practice

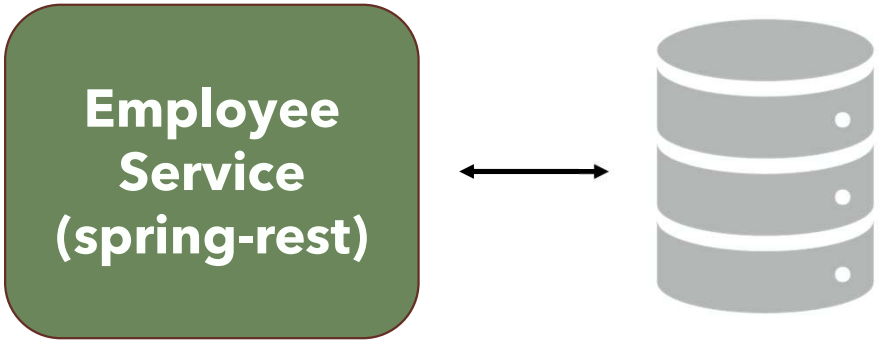
/api/employeesList
/api/deleteEmployee
/api/addEmployee
/api/updateEmployee

ไม่ควรใส่ action ใน endpoint

ควรใช้ **HTTP methods** แทน ในการกำหนดการกระทำ หรือ **action**

Employee Real-Time Project

HTTP Method	CRUD Action	CRUD Action
POST	/api/employees	สร้าง พนักงาน ใหม่ (Create)
GET	/api/employees	อ่านรายชื่อพนักงานหลายคน (Read)
GET	/api/employees/{employeeId}	อ่านรายชื่อพนักงานคนเดียว (Read)
PUT	/api/employees	แก้ไขข้อมูลพนักงานที่มีอยู่ (Update)
DELETE	/api/employees/{employeeId}	ลบข้อมูลพนักงานที่มีอยู่ (Delete)



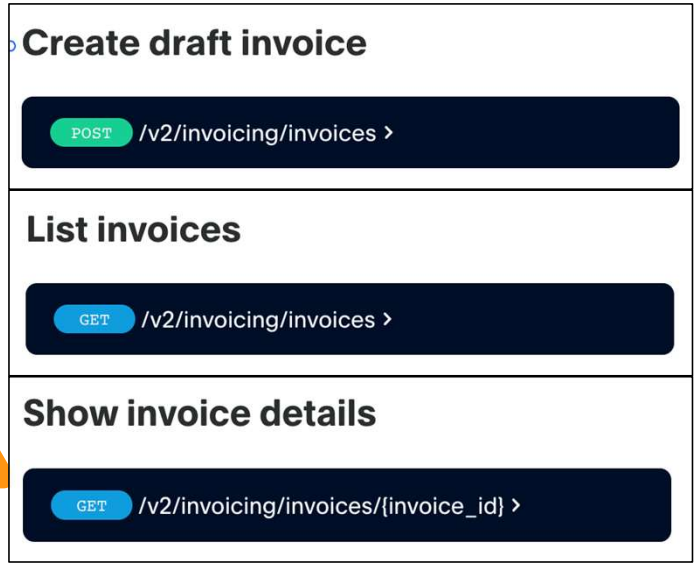
ตัวอย่าง API เพิ่มเติม

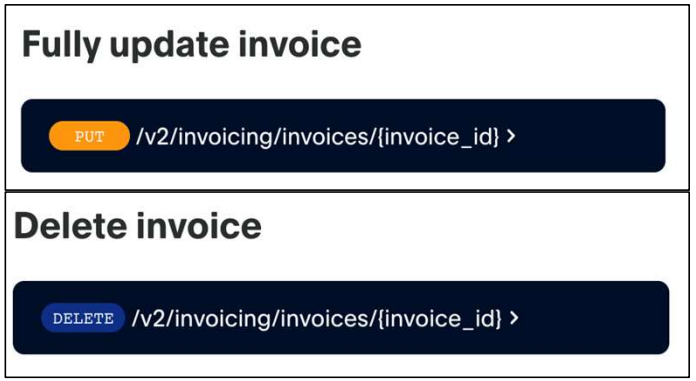
- ในส่วนต่อไปนี้ เราจะมาดู API จากโปรเจกต์อื่น ๆ
- Paypal
- GitHub
- SalesForce

PayPal

API การออกใบแจ้งหนี้ของ PayPal (PayPal Invoicing API)

- https://developer.paypal.com/docs/api/invoicing/v2/





GitHub

API คลังข้อมูลของ GitHub (GitHub Repositories API)

- https://developer.paypal.com/docs/api/invoicing/v2/

Create a new repository

```
POST /user/repos
```

List your repositories

```
GET /user/repos
```

Delete a repository

```
DELETE /repos/:owner/:repo
```

Get a repository

```
GET /repos/:owner/:repo
```

SalesForce

อุตสาหกรรม REST API

- https://sforce.co/2J40ALH

Retrieve All Individuals

GET /services/apexrest/v1/individual/

Retrieve One Individual

GET /services/apexrest/v1/individual/{individual_id}

Create an individual

POST /services/apexrest/clinic01/v1/individual/

Update an individual

PUT /services/apexrest/clinic01/v1/individual/

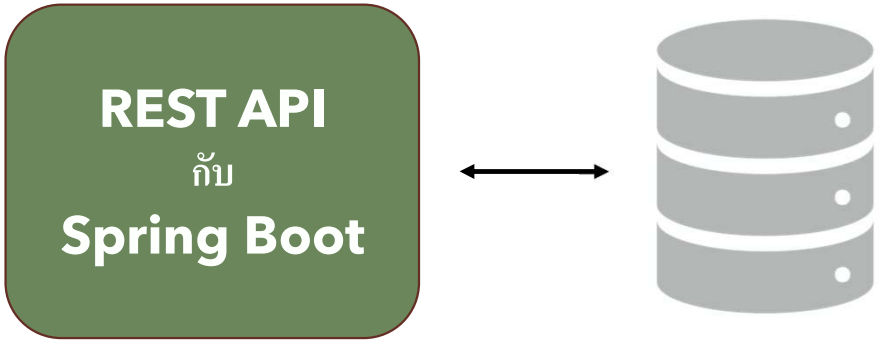
Spring Boot REST API Real Time Project

Chapter 11

Project

REST API กับ Spring Boot

ที่เชื่อมต่อกับฐานข้อมูล



ข้อความต้องการ API

จากหัวหน้า

สร้าง **REST API** สำหรับเก็บข้อมูลพนักงาน

สิ่งที่ **REST** ควรจะทำได้

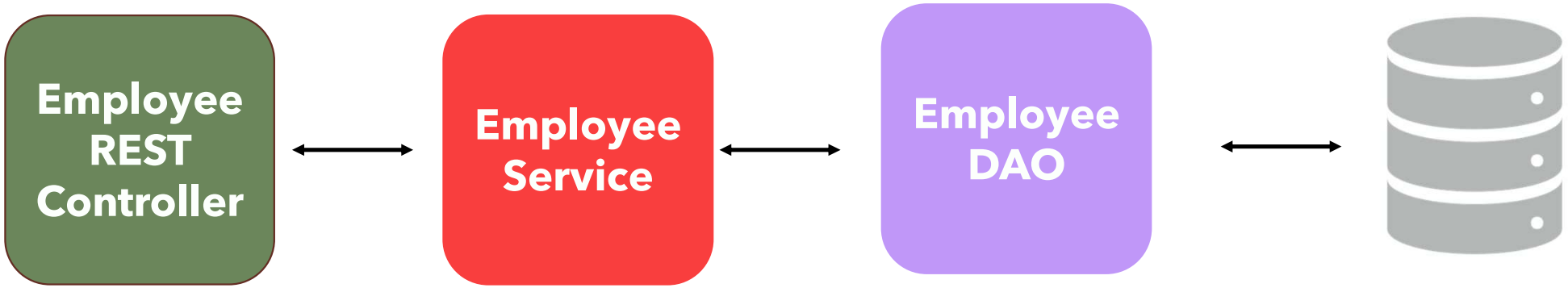
- ให้ข้อมูล รายชื่อพนักงานได้หลายคน
- ให้ข้อมูล พนักงานคนเดียว ตามรหัส
- เพิ่มพนักงานใหม่
- เปลี่ยนแปลงพนักงาน
- ลบพนักงาน

ขั้นตอนการพัฒนา

ทีละขั้นตอน

1. Setup ตัว Database
2. สร้างโปรเจกต์ Spring Boot โดยใช้ Spring Initializr
3. อ่าน(Get) รายชื่อพนักงาน
4. อ่าน(Get) พนักงานคนเดียวตาม ID
5. เพิ่ม(Add) พนักงานใหม่
6. แก้ไข/เปลี่ยนแปลง(Update) พนักงานที่มีอยู่
7. ลบ(Delete) พนักงานที่มีอยู่

สถาปัตยกรรมแอพพลิเคชัน



สร้าง โปรเจกต์ Spring Boot

Chapter 12

ขั้นตอนการพัฒนา

ทีละขั้นตอน

1. ตั้งค่าฐานข้อมูลเป็น Dev Environment
2. สร้างโปรเจกต์ Spring Boot โดยใช้ Spring Initializr
3. รับ(Get) รายชื่อพนักงาน
4. รับ(Get) พนักงานคนเดียวตาม ID
5. เพิ่ม(Add) พนักงานใหม่
6. แก้ไข/เปลี่ยนแปลง(Update) พนักงานที่มีอยู่
7. ลบ(Delete) พนักงานที่มีอยู่

หน้าที่ของเรา

สร้าง Spring boot Project สำหรับการทำ REST

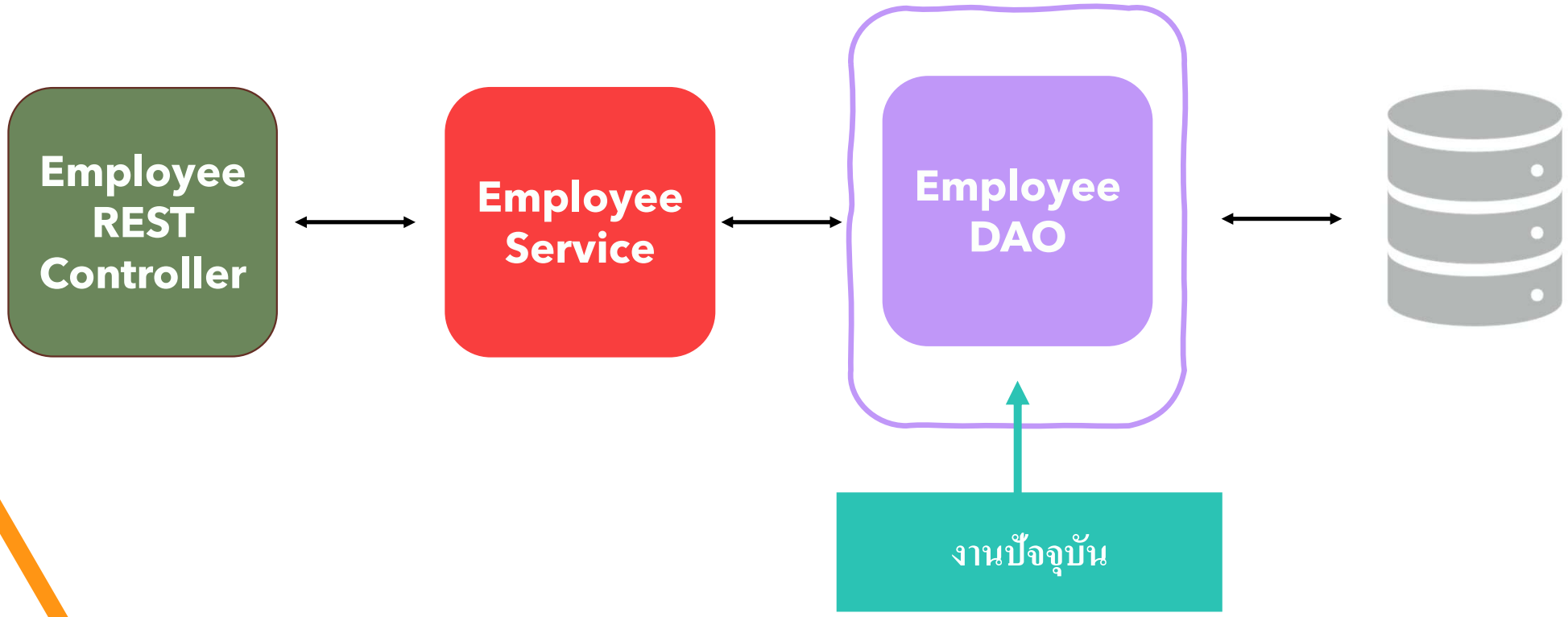
- Mini-workshop: สร้าง Spring boot Project สำหรับการทำ REST

- Ref: lab-s5-l6

สร้าง DAO ใน Spring Boot

Chapter 13

สถาปัตยกรรมแอพพลิเคชัน



ขั้นตอนการพัฒนา

ทีละขั้นตอน

1. ตั้งค่าฐานข้อมูลเป็น Dev Environment
2. สร้างโปรเจกต์ Spring Boot โดยใช้ Spring Initializr
3. รับ(Get) รายชื่อพนักงาน
4. รับ(Get) พนักงานคนเดียวตาม ID
5. เพิ่ม(Add) พนักงานใหม่
6. แก้ไข/เปลี่ยนแปลง(Update) พนักงานที่มีอยู่
7. ลบ(Delete) พนักงานที่มีอยู่

สร้างเลเยอร์ DAO สำหรับการทำ CRUD REST

DAO Interface

```
public interface EmployeeDAO
{
    List < Employee > findAll();
}
```

DAO Impl

```java
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO
{
    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl( EntityManager theEntityManager )
    {
       entityManager = theEntityManager;
    }
    ...
}
```

Interface เดียวกัน สำหรับ API เพื่อความ consistency

สร้างให้อัตโนมัติ โดย Spring Boot

Constructor injection

Get รายชื่อพนักงาน

```
@Override
public List < Employee > findAll()
{
    // create a query
    TypedQuery < Employee > theQuery =
          entityManager.createQuery( "from Employee" , Employee.class );

    // execute query and get result list
    List < Employee > employees = theQuery.getResultList();

    // return the results
    return employees;
}
```

ขั้นตอนการพัฒนา

1. ใส่ config เกี่ยวกับ database ใน application.properties
2. สร้าง Employee Entity
3. สร้าง DAO interface
4. สร้าง DAO implementation
5. สร้าง REST controller เพื่อใช้ DAO

การสร้าง REST + DAO

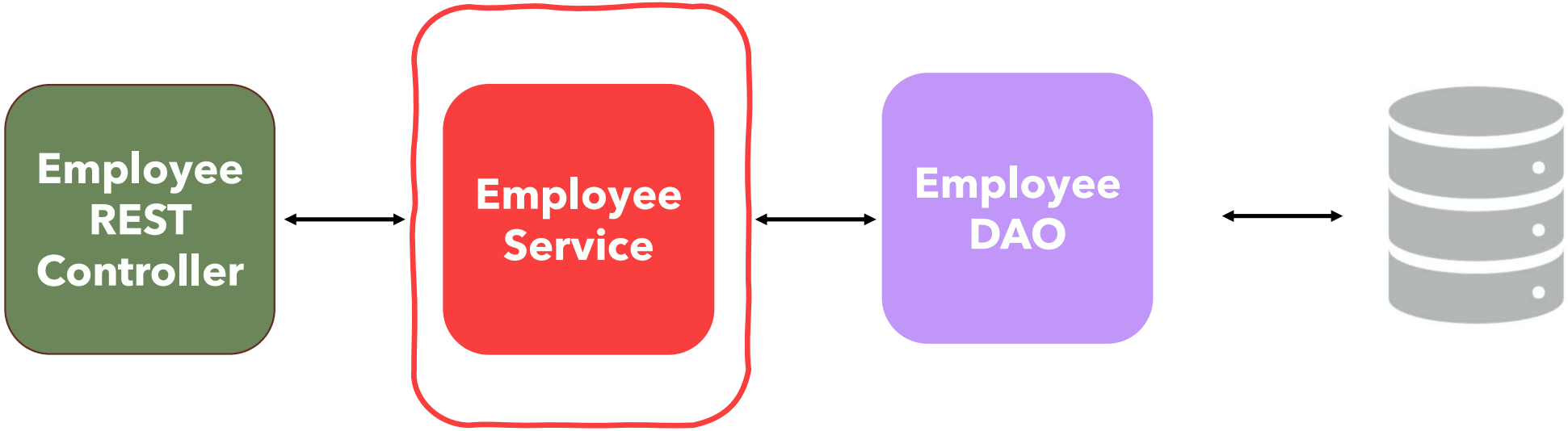
- Mini-workshop: การสร้าง REST + DAO

- Ref: lab-s5-l7

Define Services with @Service

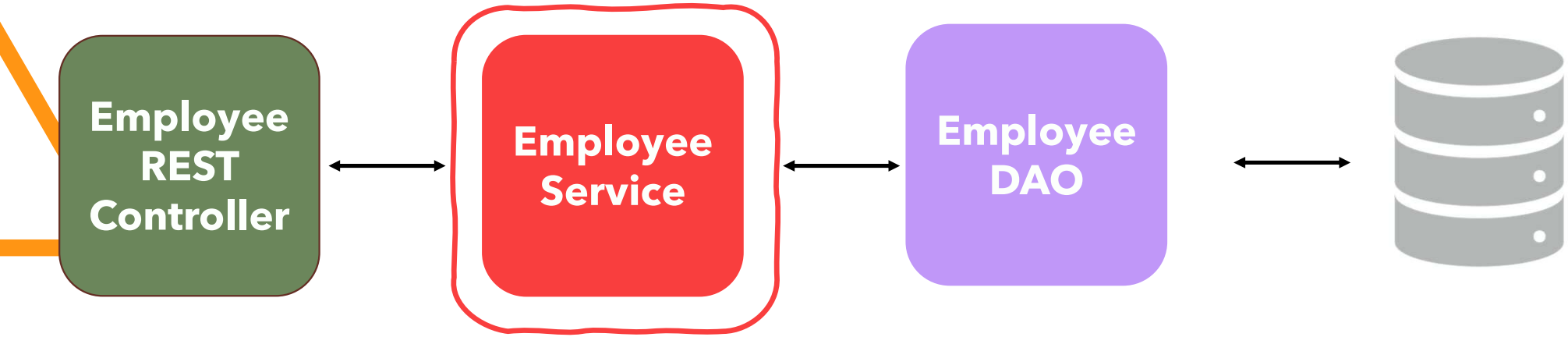
Chapter 14

Refactor: เพิ่ม Service Layer

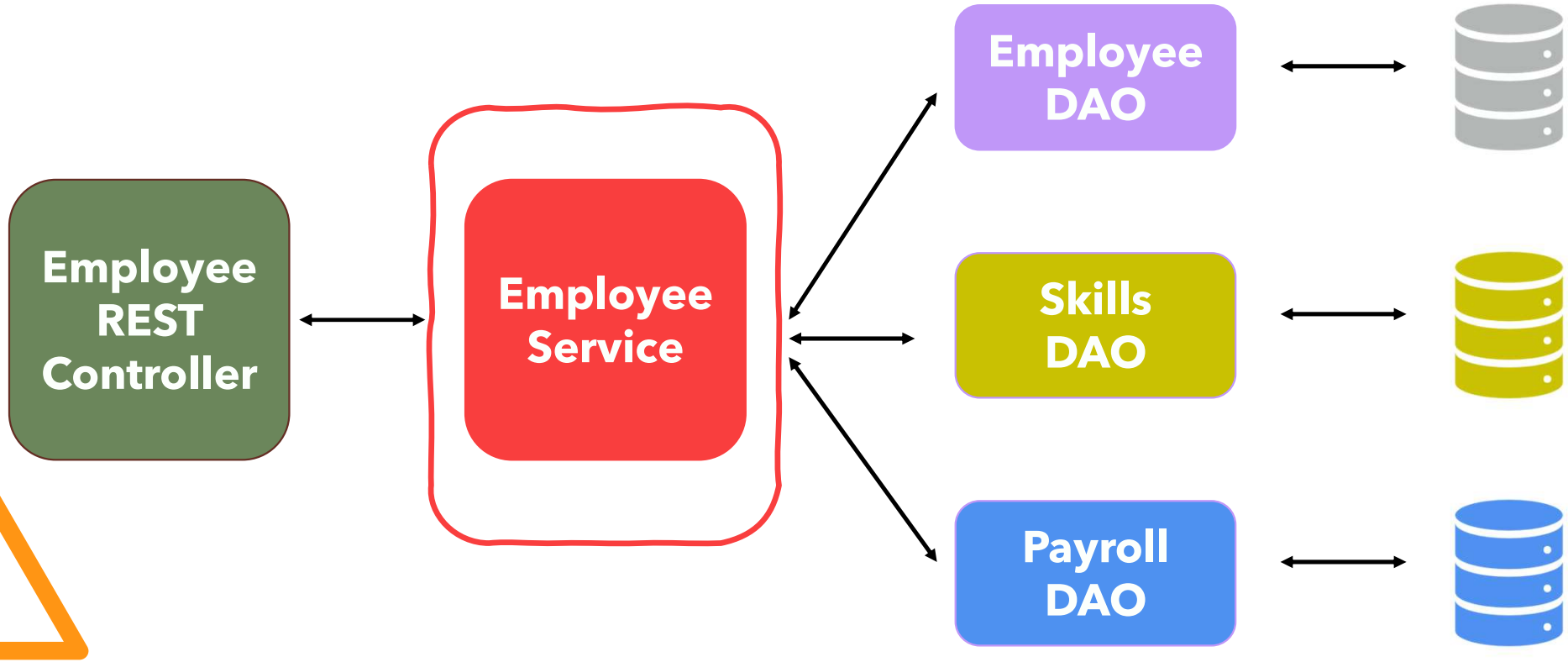


วัตถุประสงค์ของ Service Layer

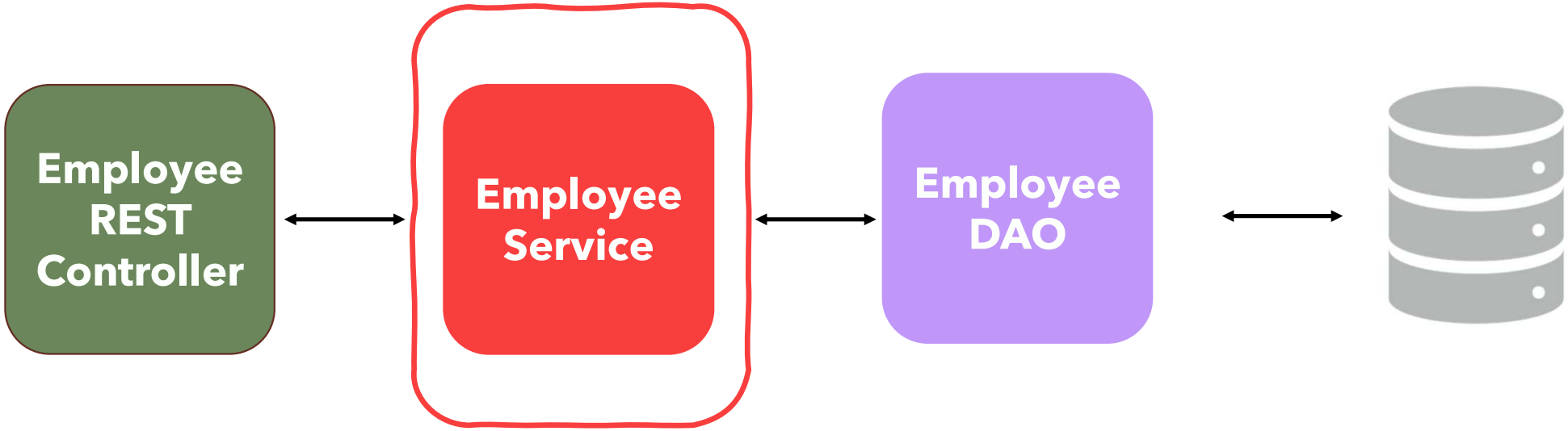
- **Service Facade** design pattern
- เป็น layer กลางสำหรับ กำหนด business logic
- Integrate ข้อมูลจากหลาย sources (DAO/repositories)



Integrate ข้อมูลจากหลาย sources

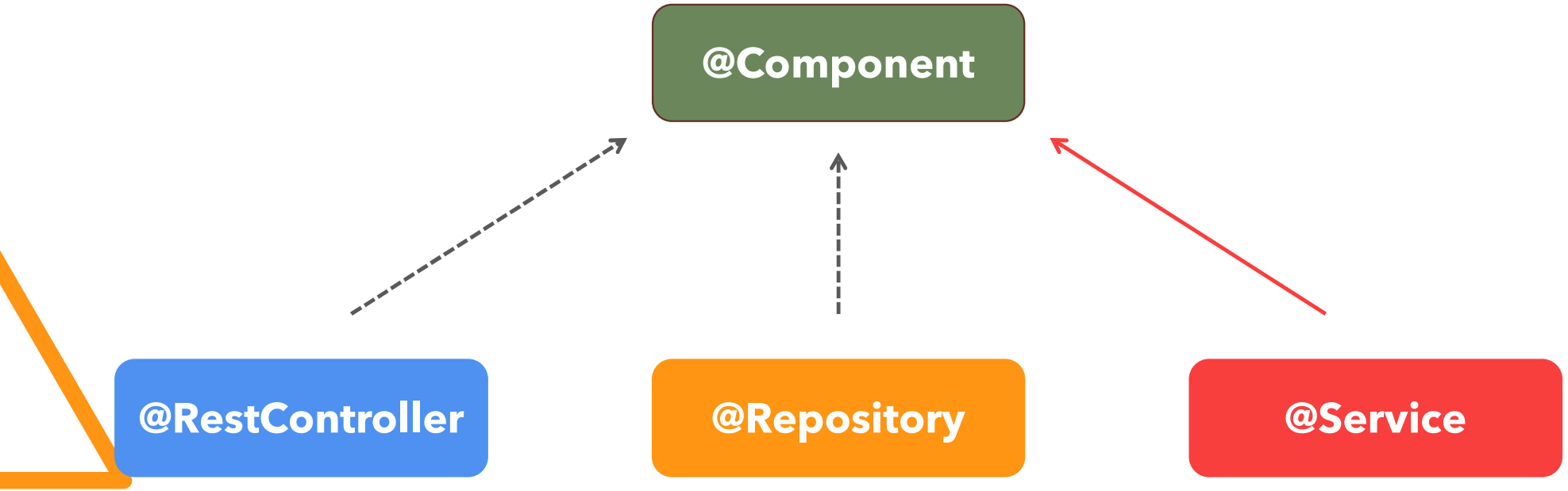


Most Times – Delegate Calls



Specialized Annotation สำหรับ Services

- Spring จะจัดเตรียม @Service annotation



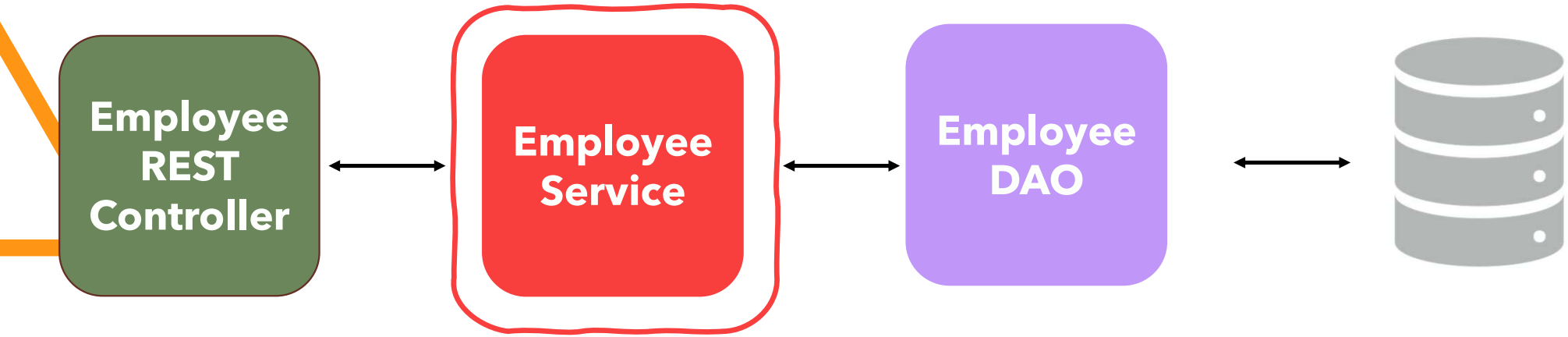
Specialized Annotation สำหรับ Services

- **@Service** นำไปปรับใช้กับ การใช้งาน Service

- Spring จะลงทะเบียนการใช้ Service โดยอัตโนมัติ
  - ด้วย Component-scanning

Employee Service

- ขั้นแรก การกำหนด Service interface
- ขั้นสอง การกำหนดการใช้งาน Service
  - inject the Employee DAO



ขั้นแรก การกำหนด Service interface

```
public interface EmployeeService {

        List<Employee> findAll();

}
```

ขั้นสอง การกำหนดการใช้งาน Service

@service – เปิดใช้งานการสแกน component

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

 // inject EmployeeDAO ...

    @Override
    public List<Employee> findAll() {
      return employeeDAO.findAll();
 }
}
```

การสร้าง Service Layer

- Mini-workshop: การสร้าง Service Layer

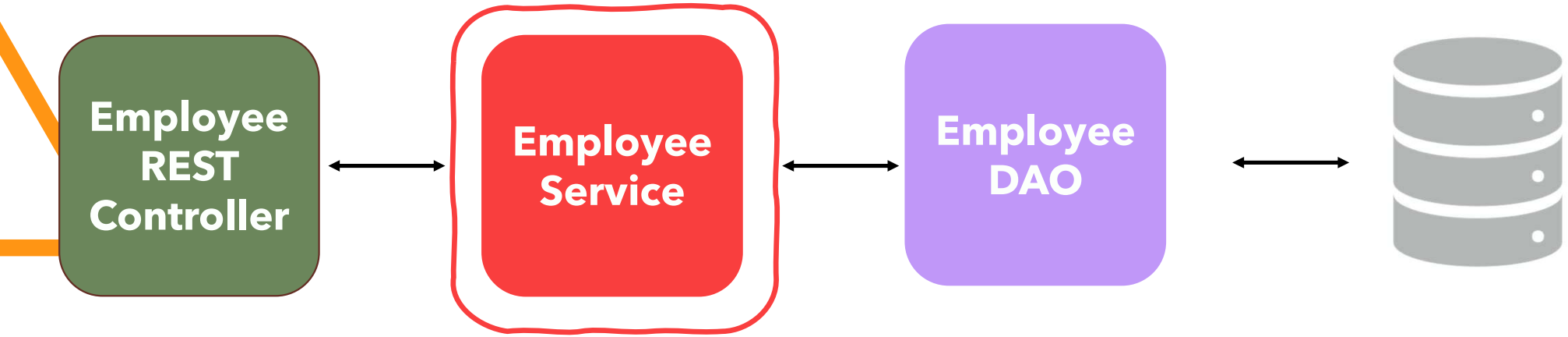
- Ref: lab-s5-l8

DAO: การค้นหา(Find) เพิ่ม(Add) แก้ไข(Update) และลบ(Delete)

Chapter 15

Service Layer – แนวทางการทำที่ดีที่สุด (Best Practice)

- Best practice คือ การนำ transactional ไปใช้ที่ service layer
- เป็นหน้าที่ของ service layer’s ในการจัดการ transactional
- สำหรับการใช้งาน Code
  - ใช้ @Transactional กับ service methods
  - ลบ @Transactional บน DAO methods ที่มีอยู่

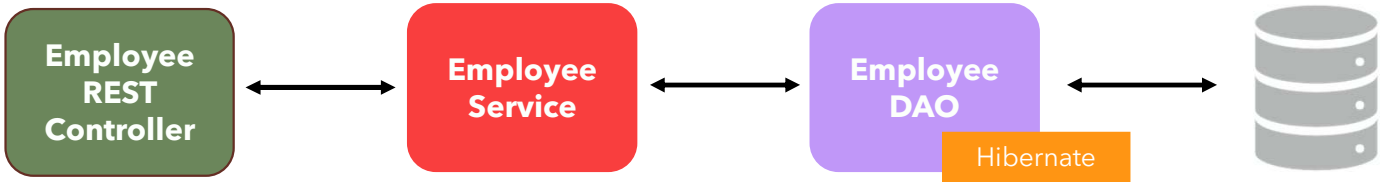


ขั้นตอนการพัฒนา

ทีละขั้นตอน

1. ตั้งค่าฐานข้อมูลเป็น Dev Environment
2. สร้างโปรเจกต์ Spring Boot โดยใช้ Spring Initializr
3. รับ(Get)รายชื่อพนักงาน
4. รับ(Get)พนักงานคนเดียวตาม ID
5. เพิ่ม(Add)พนักงานใหม่
6. แก้ไข/เปลี่ยนแปลง(Update) พนักงานที่มีอยู่
7. ลบ(Delete)พนักงานที่มีอยู่

DAO methods



DAO: Get (รับ) employee คนเดียว

```java
@Override
public Employee findById(int id) {

    Employee employee = entityManager.find( Employee.class, id );

    return employee;
}
```

DAO: Add (เพิ่ม) หรือ Update (แก้ไข) employee

Note: เราไม่ได้ใช้ @Transactional ที่ DAO Layer โดยจะจัดการที่ Service layer

if id == 0 then save/insert
else update

```
@Override
public Employee save(Employee theEmployee) {

    // save or update the employee
    Employee dbEmployee = entityManager.merge(theEmployee);

    // return dbEmployee
    return dbEmployee;
    }
```

ส่งคืน dbEmployee
ที่มี id จากฐานข้อมูล (ในกรณีเพิ่ม)

DAO: Delete (ลบ) employee ที่มีอยู่

Note: เราไม่ได้ใช้ **@Transactional** ที่ **DAO Layer** โดยจะจัดการที่ **Service layer**

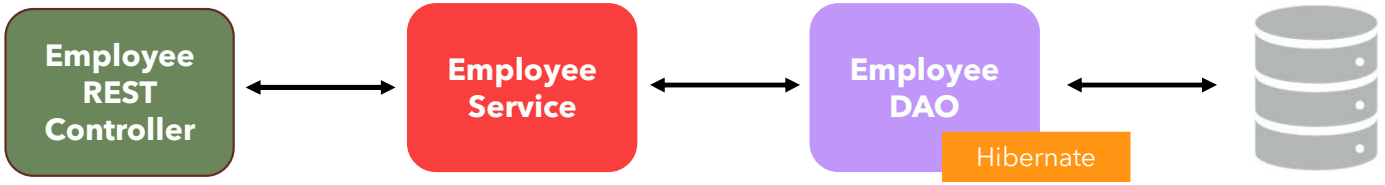
```
@Override
public void deleteById(int theId) {

    // find the employee by id
    Employee theEmployee = entityManager.find(Employee.class, theId);

    // delete the employee
    entityManager.remove(theEmployee);
}
```

ขั้นตอนการพัฒนา

ทีละขั้นตอน

1. ตั้งค่าฐานข้อมูลเป็น Dev Environment
2. สร้างโปรเจกต์ Spring Boot โดยใช้ Spring Initializr
3. รับ(Get) รายชื่อพนักงาน
4. รับ(Get) พนักงานคนเดียวตาม ID
5. เพิ่ม(Add) พนักงานใหม่
6. แก้ไข/เปลี่ยนแปลง(Update) พนักงานที่มีอยู่
7. ลบ(Delete) พนักงานที่มีอยู่

DAO methods



CRUD ตัวที่เหลือ

- Mini-workshop: CRUD ตัวที่เหลือ

- Ref: lab-s5-l9

Rest Controller Methods ค้นหา (Find) และ เพิ่ม (Add)

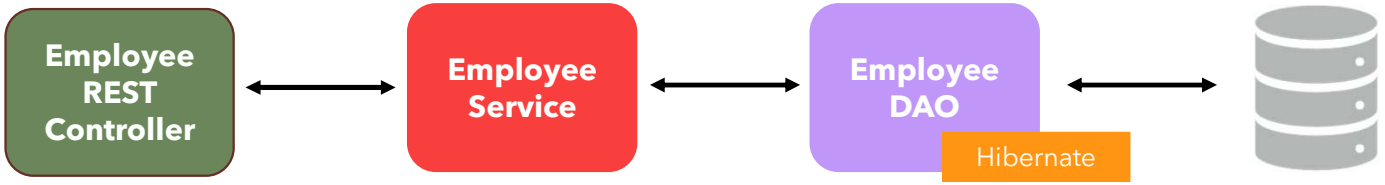
Chapter 16

ขั้นตอนการพัฒนา

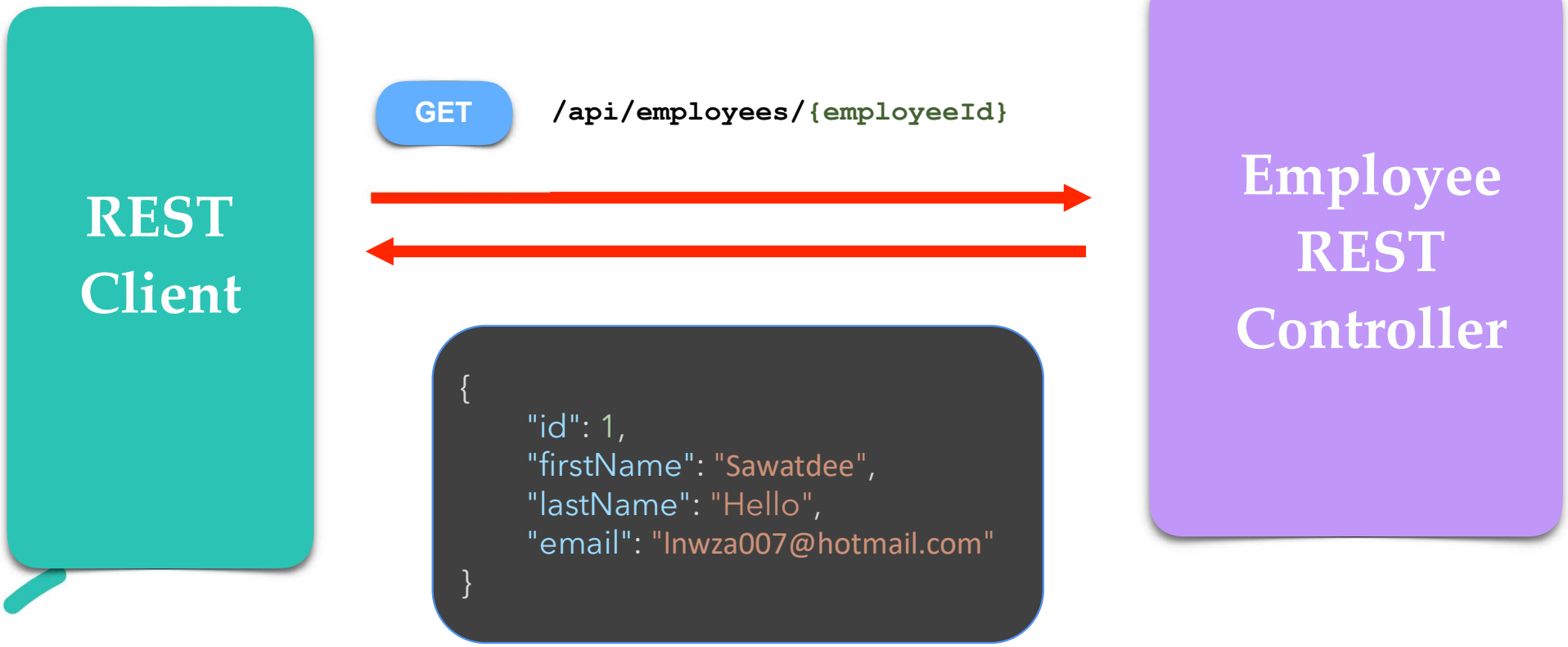
ทีละขั้นตอน

1. ตั้งค่าฐานข้อมูลเป็น Dev Environment
2. สร้างโปรเจกต์ Spring Boot โดยใช้ Spring Initializr
3. รับ(Get) รายชื่อพนักงาน
4. รับ(Get) พนักงานคนเดียวตาม ID
5. เพิ่ม(Add) พนักงานใหม่
6. แก้ไข/เปลี่ยนแปลง(Update) พนักงานที่มีอยู่
7. ลบ(Delete) พนักงานที่มีอยู่

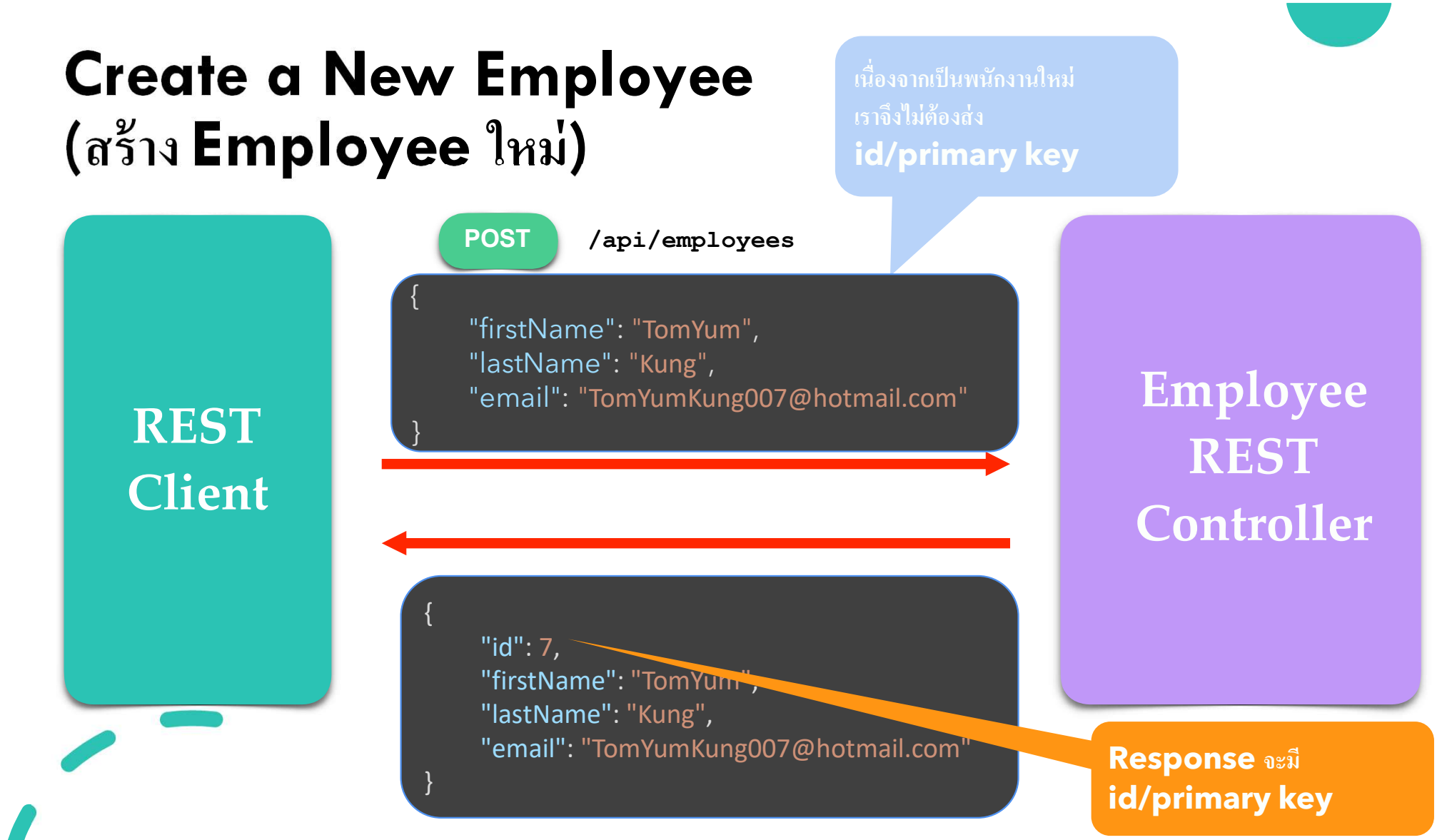
Rest Controller methods



Read a Single Employee
(อ่าน **Employee** คนเดียว)



Create a New Employee (สร้าง Employee ใหม่)

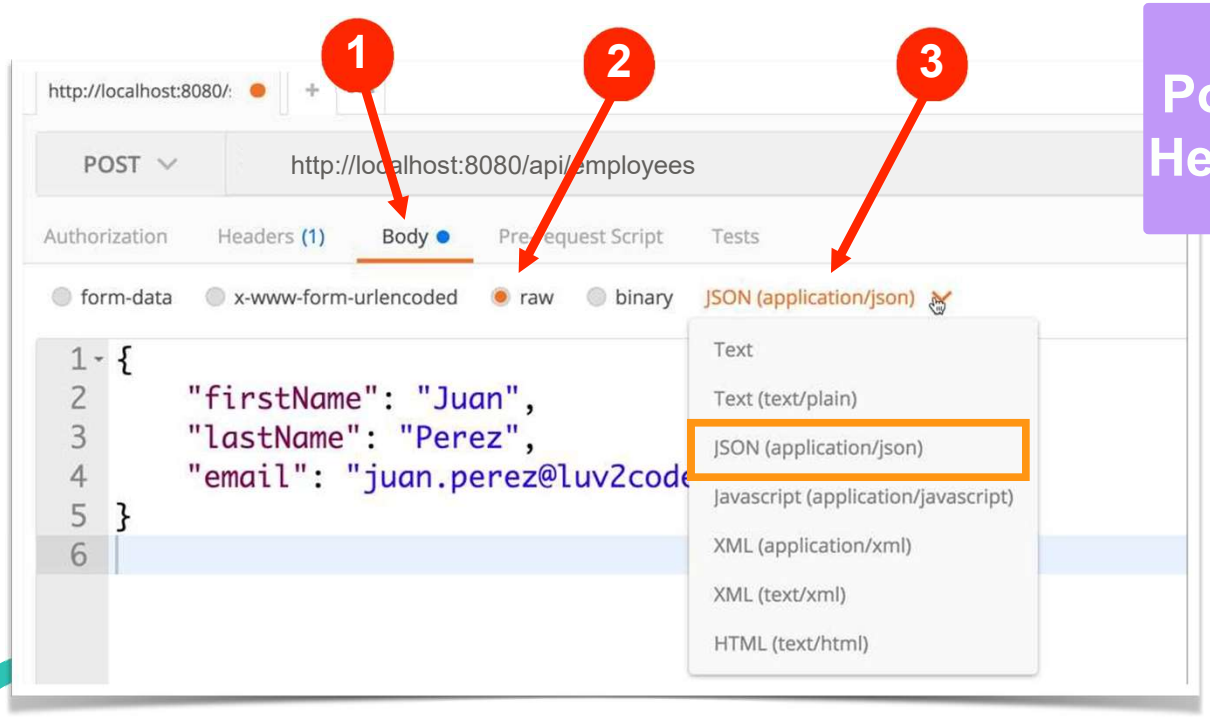


การส่ง JSON ไปยัง Spring REST Controllers

- เมื่อส่งข้อมูล JSON ไปยัง Spring REST Controllers
- เพื่อให้ Controller ประมวลผลข้อมูล JSON ต้องตั้งค่าส่วนหัวคำขอ HTTP
  - Content-type: application/json
- ต้องกำหนดค่า REST client เพื่อส่ง HTTP Request header ที่ถูกต้อง

Postman - การส่ง JSON Request Body

- ต้องตั้งค่าส่วนหัวคำขอ HTTP ใน Postman



Postman จะตั้งค่า HTTP Request Header ที่ถูกต้องไว้ให้ ตามการกำหนดค่าเหล่านี้

CRUD ตัวที่เหลือ - 2

- Mini-workshop: CRUD ตัวที่เหลือ - 2

- Ref: lab-s5-l10

Rest Controller Methods แก้ไข (Update) ลบ (Delete)

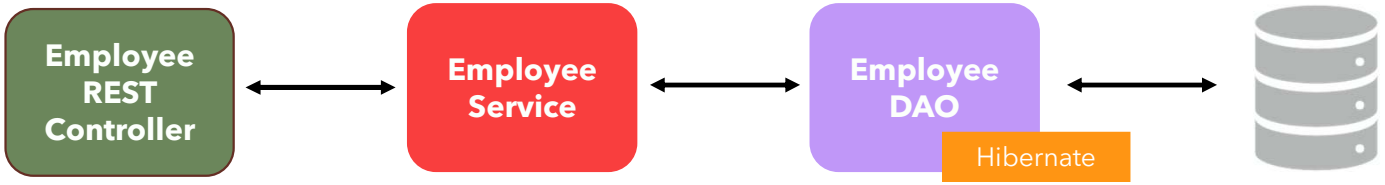
Chapter 17

ขั้นตอนการพัฒนา

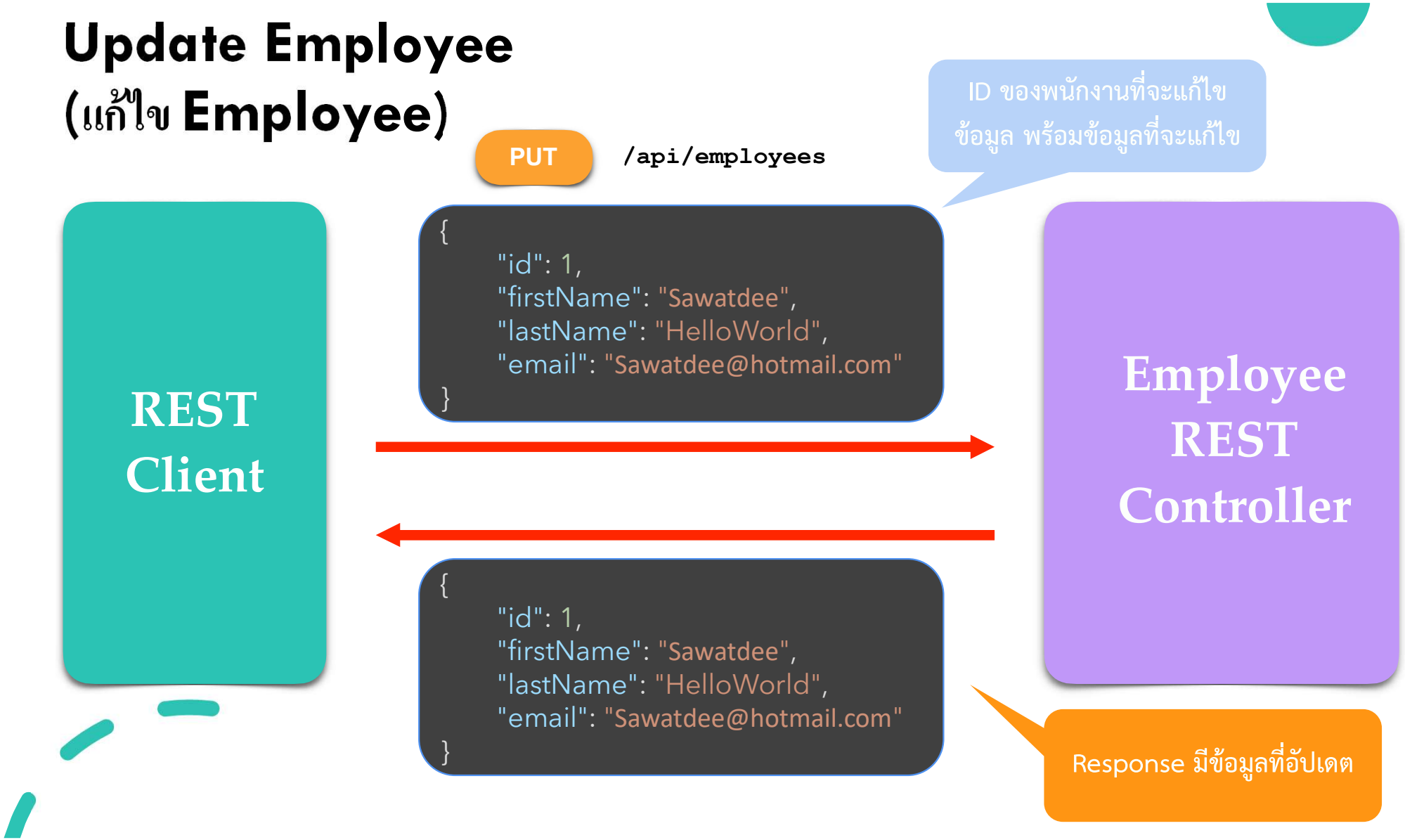
ทีละขั้นตอน

1. ตั้งค่าฐานข้อมูลเป็น Dev Environment
2. สร้างโปรเจกต์ Spring Boot โดยใช้ Spring Initializr
3. รับ(Get) รายชื่อพนักงาน
4. รับ(Get) พนักงานคนเดียวตาม ID
5. เพิ่ม(Add) พนักงานใหม่
6. แก้ไข/เปลี่ยนแปลง(Update) พนักงานที่มีอยู่
7. ลบ(Delete) พนักงานที่มีอยู่

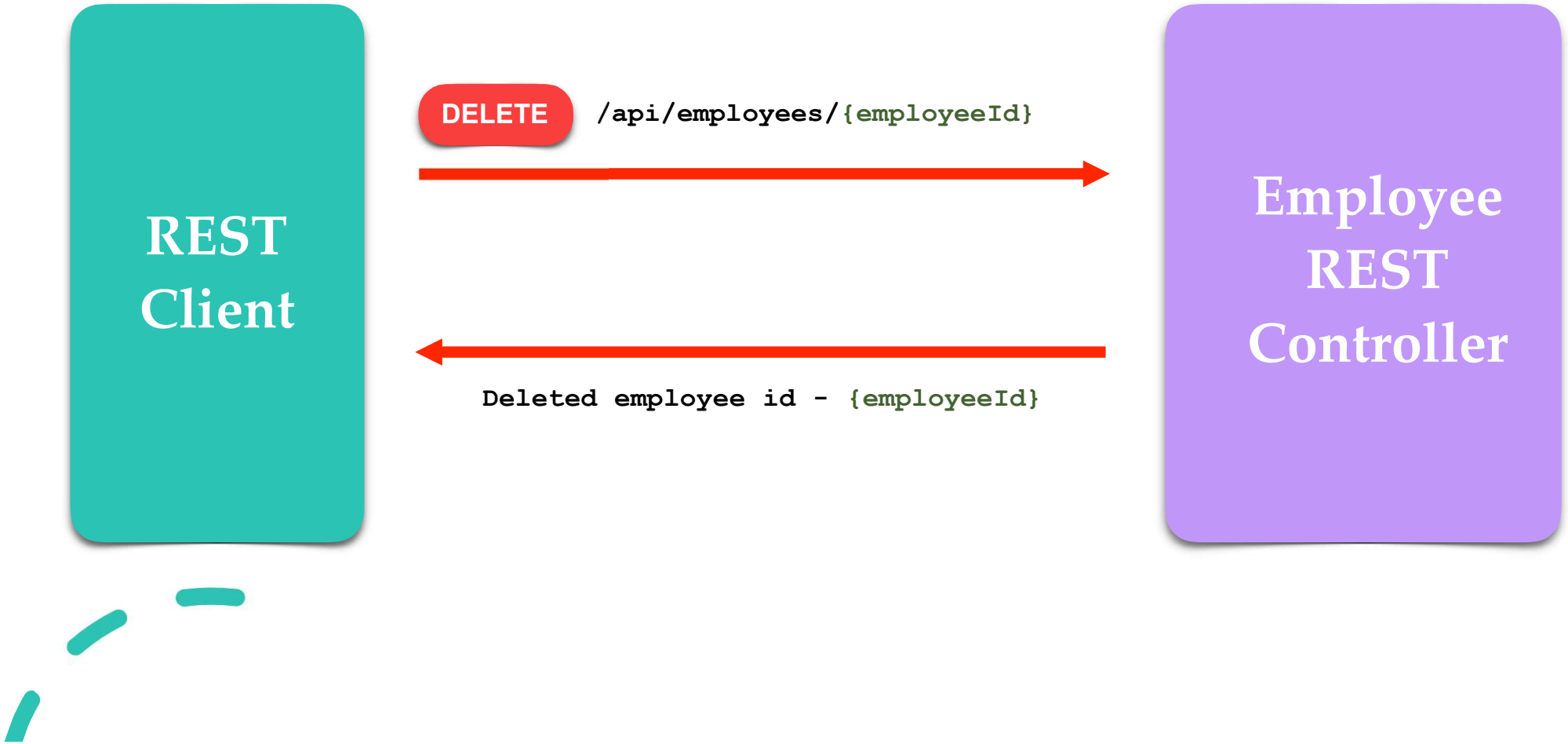
Rest Controller methods



Update Employee
(แก้ไข Employee)



Delete Employee
(ลบ Employee)



CRUD ตัวที่เหลือ - 3

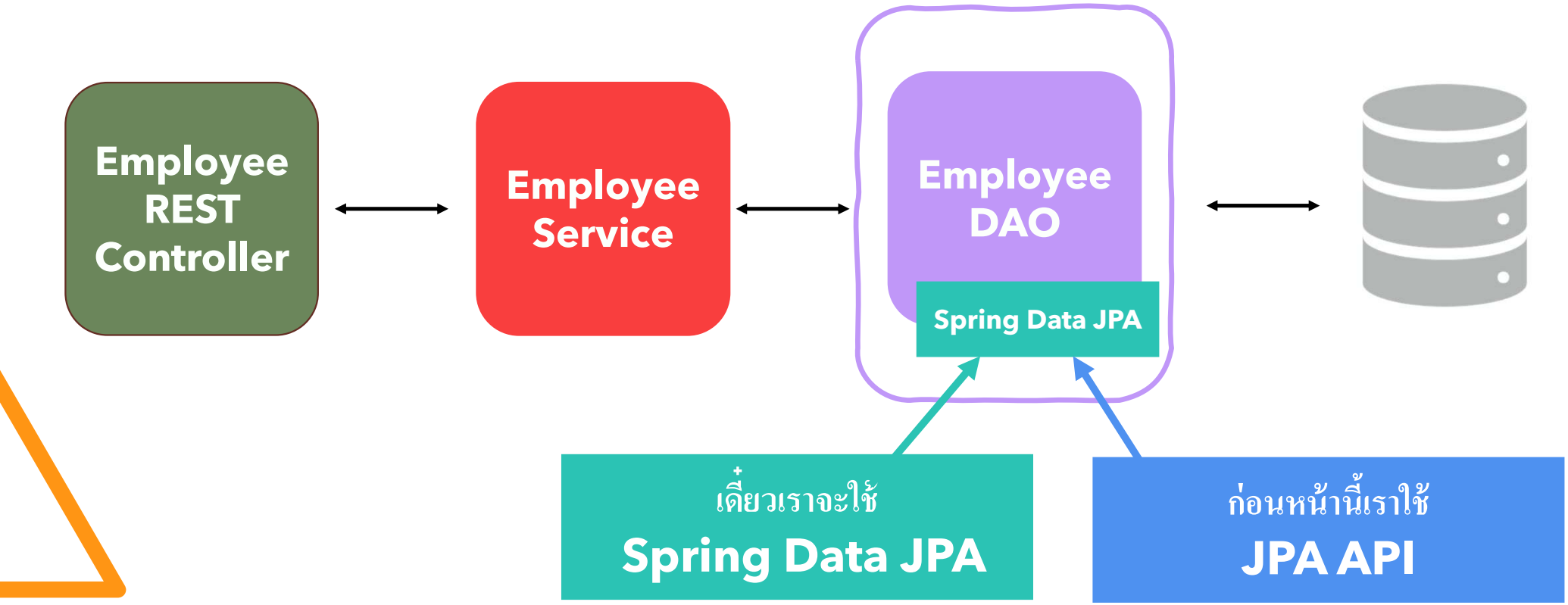
- Mini-workshop: CRUD ตัวที่เหลือ - 3

- Ref: lab-s5-l11

Spring Data JPA
ใน
Spring Boot

Chapter 18

สถาปัตยกรรมแอพพลิเคชัน



ปัญหาที่พบ

- เราได้เห็นวิธีการสร้าง DAO สำหรับ **Employee** ไปแล้ว
- จะเกิดอะไรขึ้น ถ้าหากเราต้องการสร้าง DAO สำหรับ entity อื่นๆ ?
  - **Customer**, **Student**, **Product**, **Book** ...
- เราต้องเขียนโค้ดเหมือนเดิมทั้งหมดอีกไหม???

```java
public interface EmployeeDAO
{
    Employee save(Employee employee);

    List< Employee > findAll();

    Employee findById(Integer id);

    void deleteBy
}
```

```java
@Repository
public class EmployeeDAOImpl implements EmployeeDAO
{
    private EntityManager entityManager;

    public EmployeeDAOImpl( EntityManager entityManager )
    {
       this.entityManager = entityManager;
    }

    @Override
    public Employee save( Employee employee )
    {
       entityManager.merge( employee );

       return employee;
    }

    @Override
    public List < Employee > findAll()
    {
       TypedQuery < Employee > query =
             entityManager.createQuery( "FROM Employee" , Employee.class );

       List < Employee > employees = query.getResultList();

       return employees;
    }
```

การสร้าง DAO

- เราอาจจะสังเกต เห็นรูปแบบการสร้าง DAO แล้ว

```
@Override
public Employee findById( Integer id )
{
    return entityManager.find( Employee.class, id );
}
```

โค้ดส่วนใหญ่
มีความเหมือนกัน

ความแตกต่างเพียงอย่างเดียวคือประเภท
entity และ **primary key**

ประเภท **entity**

primary key

สิ่งที่เราต้องการ

- เราต้องการจะสามารถบอกให้ Spring :

สร้าง **DAO** ให้เรา

เราแค่บอกประเภท **entity** และ **primary key** ของเรา

มีพื้นฐานฟีเจอร์ **CRUD** ทั้งหมด

Diagram ที่เราต้องการ

ตัวอย่าง

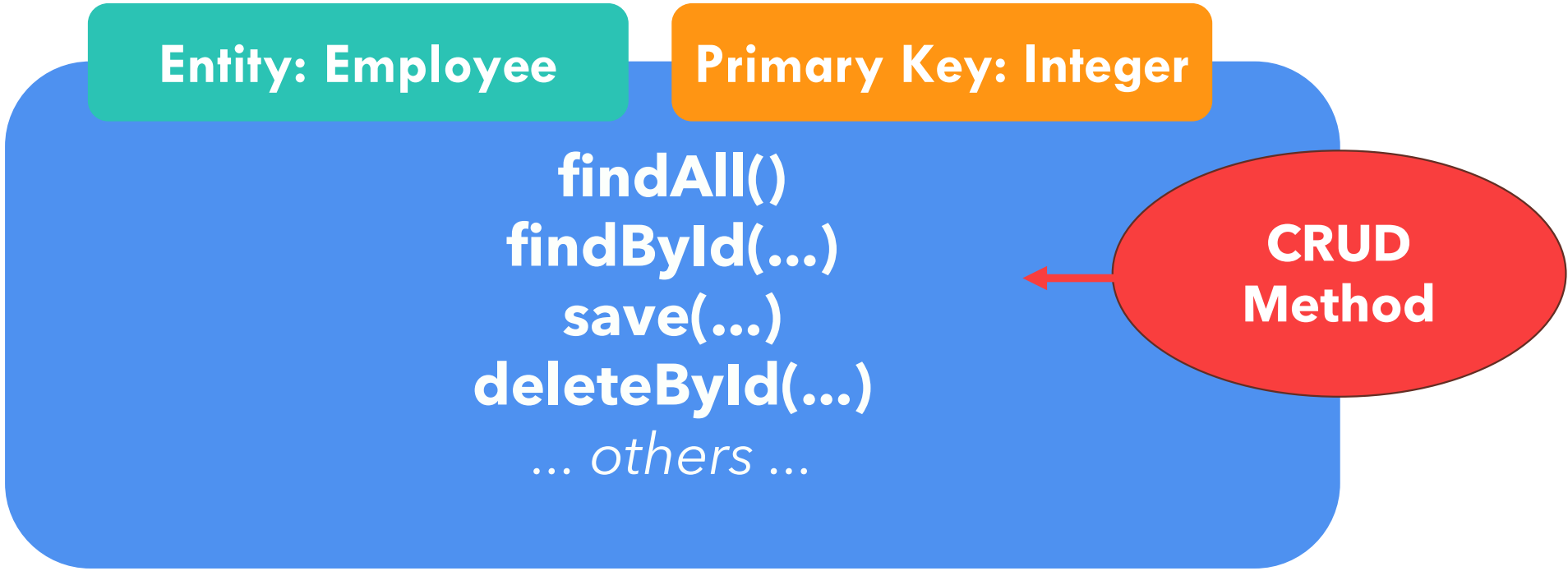


Diagram ที่เราต้องการ

ตัวอย่าง

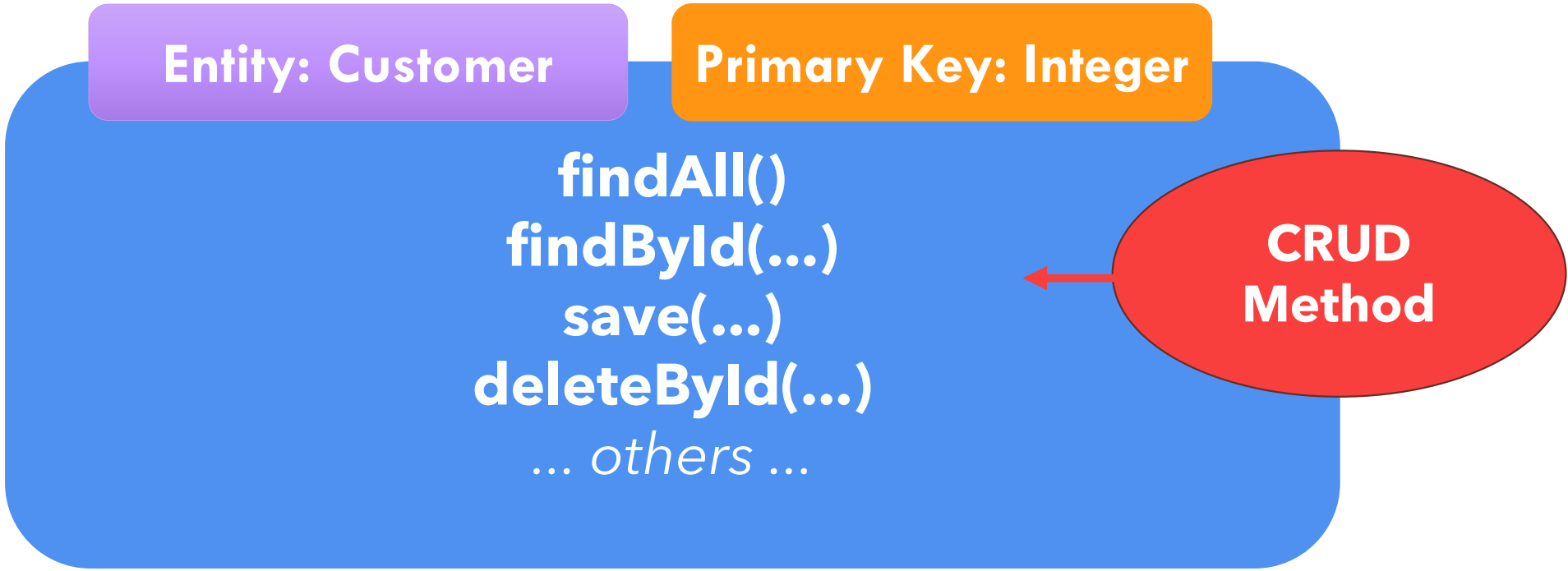
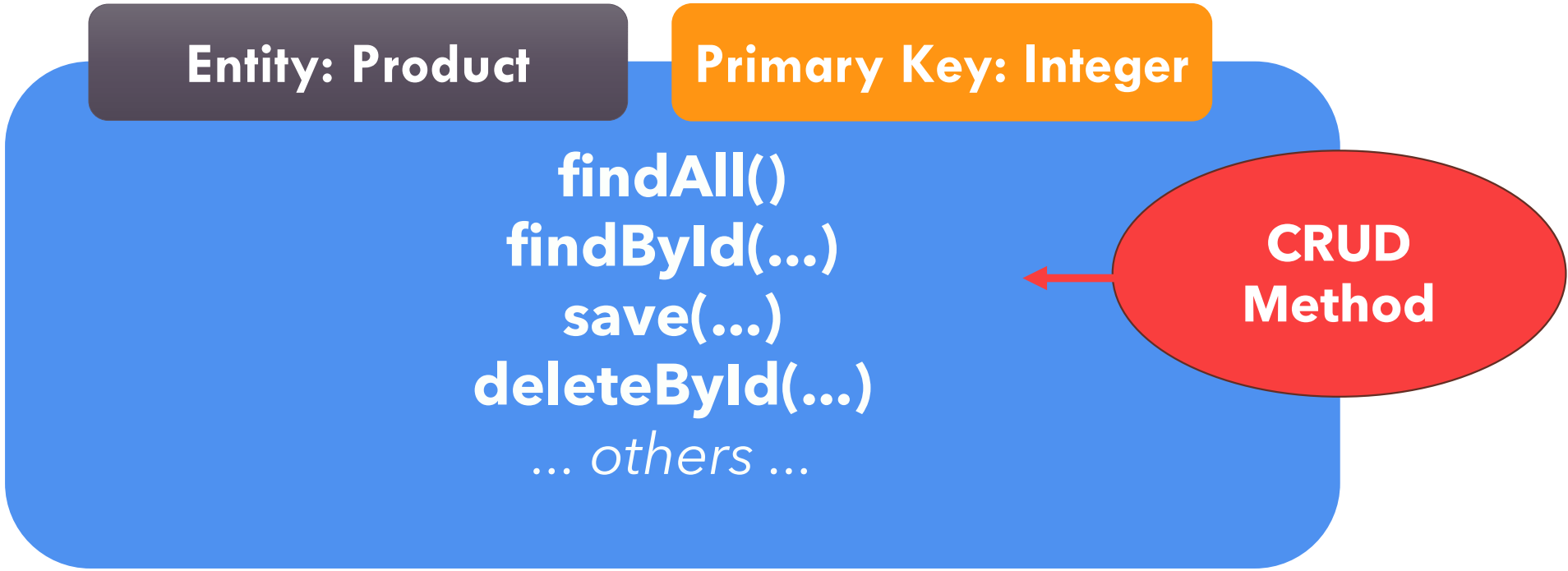


Diagram ที่เราต้องการ

ตัวอย่าง



Spring Data JPA - วิธีแก้ปัญหา

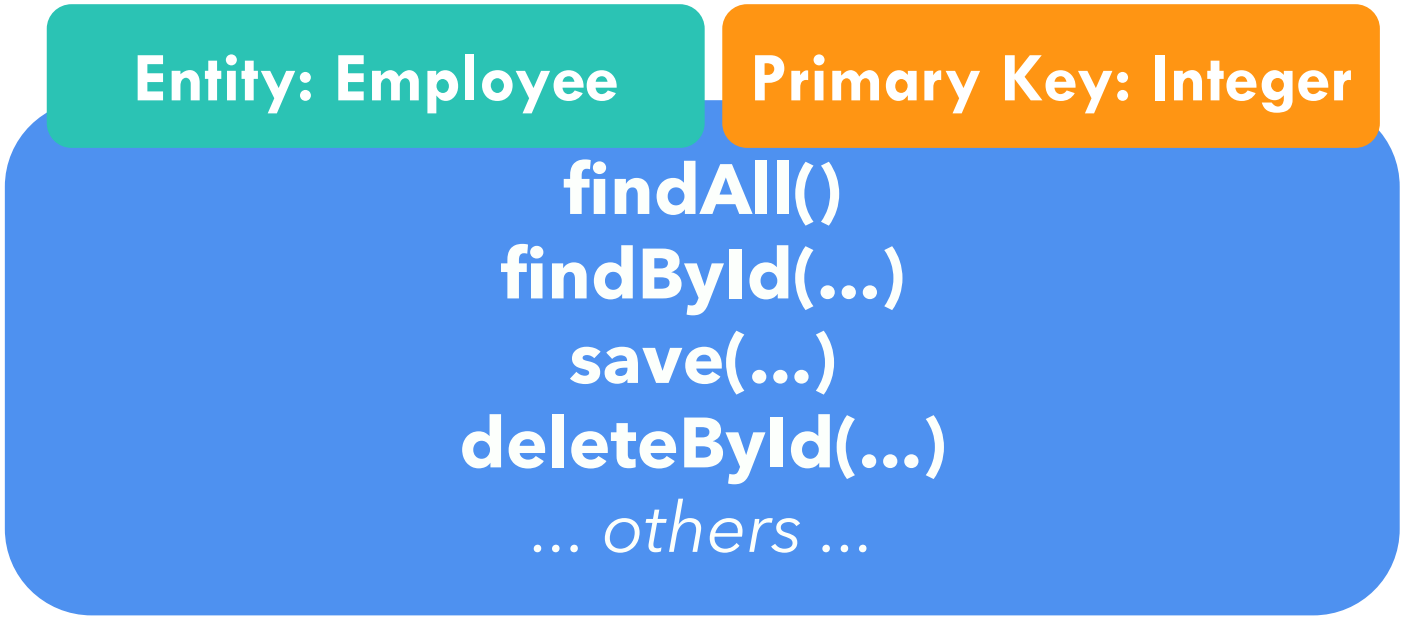
https://spring.io/projects/spring-data-jpa

- **Spring Data JPA** จะมาแก้ปัญหานี้ !!!
- สร้าง DAO และ บอกประเภท entity และ primary key ของเรา
- Spring จะทำการสร้าง CRUD Method ให้เรานำไปใช้งาน
  - ช่วยลดโค้ด DAO boiler-plate

การเขียนโค้ดจะน้อยมากกว่า 70% ... ขึ้นอยู่กับกรณีการใช้งาน

JPARepository

- Spring Data JPA มี Interface: **JpaRepository**
- แค่เราใส่ Entity และ Type ของ Primary key ใน interface นี้



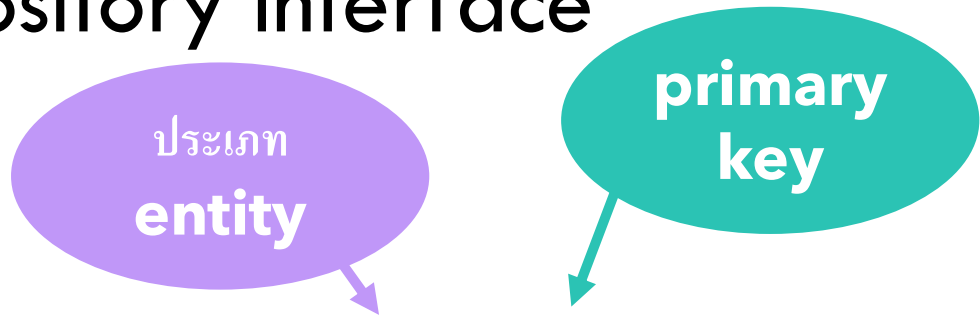
ขั้นตอนการพัฒนา

ทีละขั้นตอน

- สร้าง Interface ที่ Extend **JpaRepository** interface
- Interface นี้เราจะเรียกว่า Repository
- ใช้ Repository นี้ในแอพของเรา

ไม่จำเป็นต้อง
มีการใช้งาน **class**

ขั้นตอนแรก: Extend JpaRepository interface



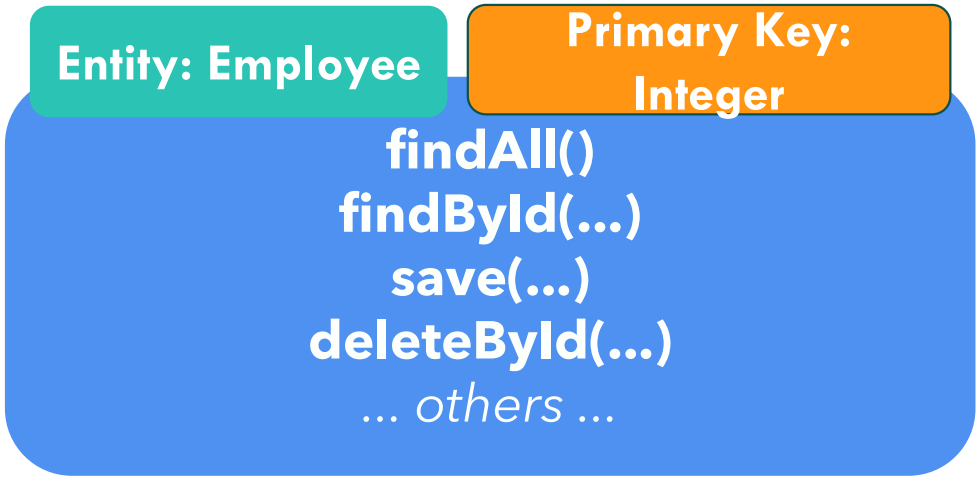

```
public interface EmployeeRepository extends JpaRepository< Employee, Integer >
{
    // No need to code here
}
```

ขั้นตอนแรก: Extend JpaRepository interface

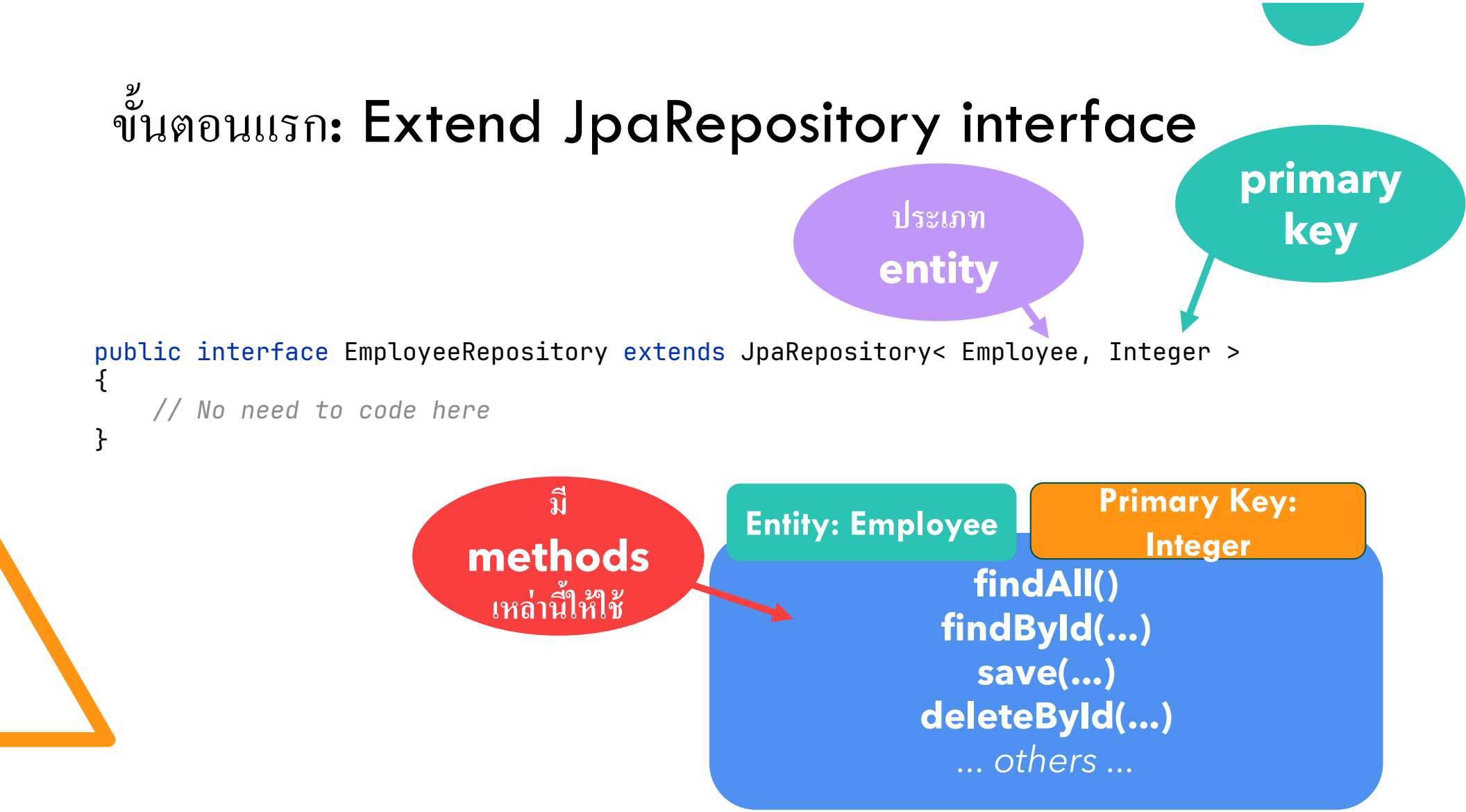
ประเภท entity

primary key

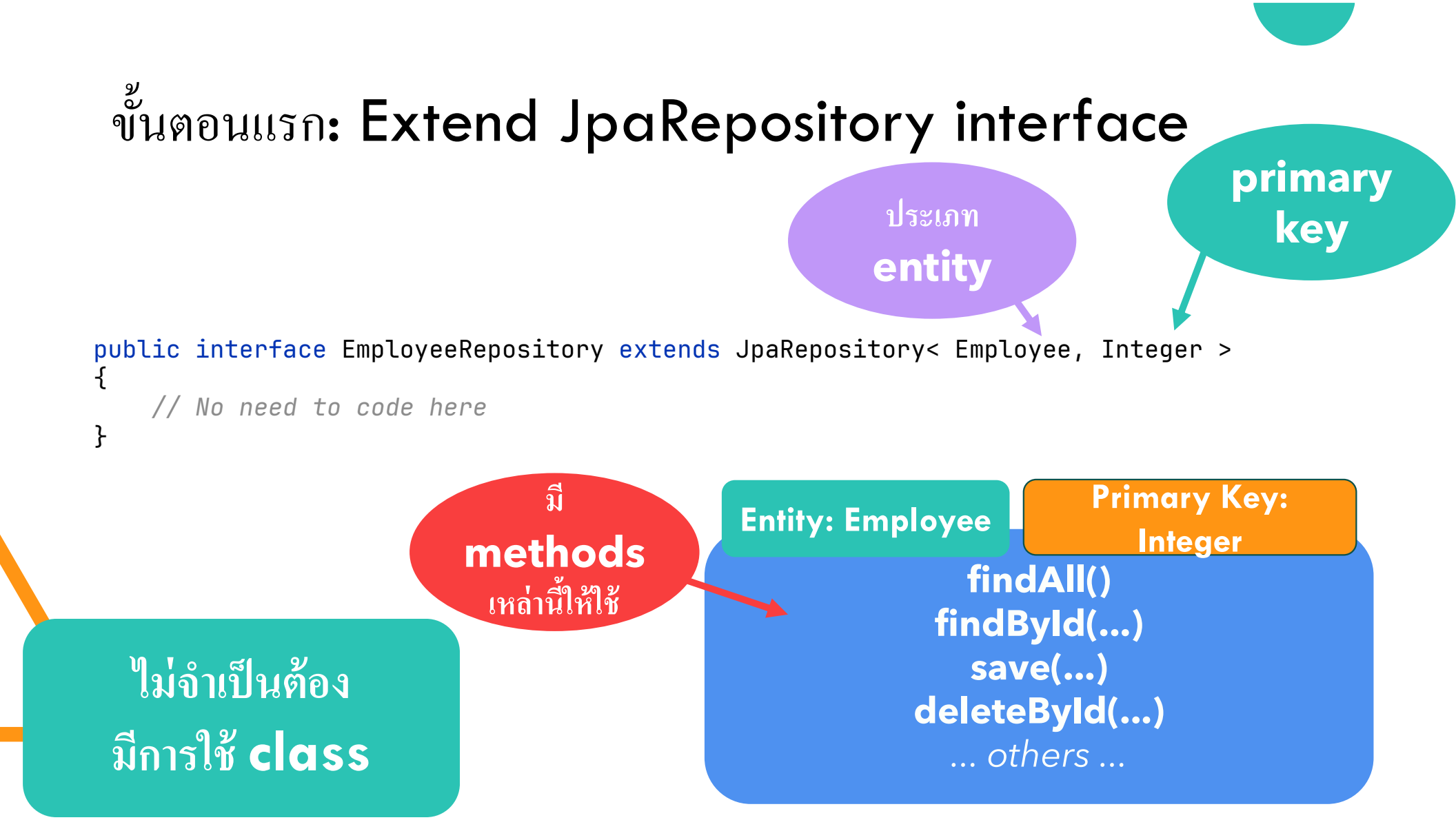
```
public interface EmployeeRepository extends JpaRepository< Employee, Integer >
{
    // No need to code here
}
```



ขั้นตอนแรก: Extend JpaRepository interface



ขั้นตอนแรก: Extend JpaRepository interface



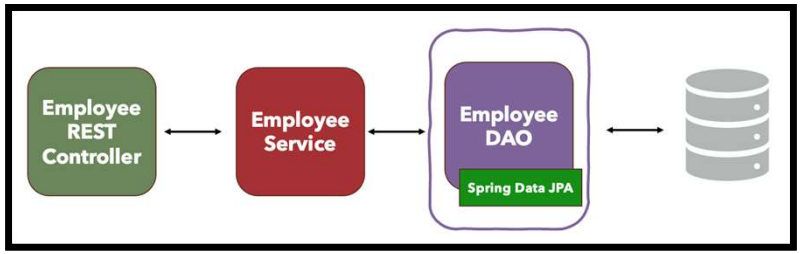
JpaRepository Doc

- รายการ methods ทั้งหมดสามารถ ดูที่ได้ที่ JavaDoc สำหรับ JpaRepository

JpaRepository (Spring Data JPA Parent 3.1.1 API)

ขั้นตอนสอง: ใช้ Repository ข้อมูลของเรา ในแอพของเรา

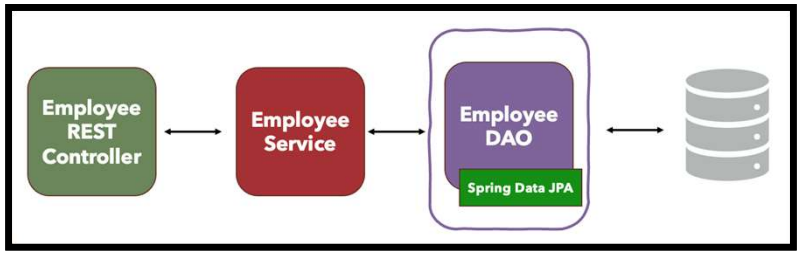
```
@Service
public class EmployeeServiceImpl implements EmployeeService
{
    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl( EmployeeRepository employeeRepository )
    {
       this.employeeRepository = employeeRepository;
    }
```



ขั้นตอนสอง: ใช้ Repository ข้อมูลของเรา ในแอพของเรา

```
@Service
public class EmployeeServiceImpl implements EmployeeService
{
    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl( EmployeeRepository employeeRepository )
    {
       this.employeeRepository = employeeRepository;
    }

    @Override
    public List < Employee > findAll()
    {
       return employeeRepository.findAll();
    }
    ...
```

Repository
ของเรา



มี Method ที่
พร้อมใช้งานให้ใช้เลย

Minimized Boilerplate Code

ก่อนมี Spring Data JPA

```java
public interface EmployeeDAO
{
    Employee save(Employee employee);

    List< Employee > findAll();

    Employee findById(Integer id);

    void deleteBy
}
```

```java
@Repository
public class EmployeeDAOImpl implements EmployeeDAO
{
    private EntityManager entityManager;

    public EmployeeDAOImpl( EntityManager entityManager )
    {
       this.entityManager = entityManager;
    }

    @Override
    public Employee save( Employee employee )
    {
        ntityManager.merge( employee );

        return employee;
    }

    @Override
    public List < Employee > findAll()
    {

                                "FROM Employee" , Employee.class );

                               ery.getResultList();
```

2 ไฟล์
ใช้โค้ด 30 บรรทัดขึ้นไป

Minimized Boilerplate Code

ก่อนมี Spring Data JPA

```java
public interface EmployeeDAO
{
    Employee save(Employee employee);

    List< Employee > findAll();

    Employee findById(Integer id);

    void deleteBy
}
```

```java
@Repository
public class EmployeeDAOImpl implements EmployeeDAO
{
    private EntityManager entityManager;

    public EmployeeDAOImpl( EntityManager entityManager )
    {
       this.entityManager = entityManager;
    }

    @Override
    public Employee save( Employee employee )
    {
        ntityManager.merge( employee );

        return employee;
    }

    @Override
    public List < Employee > findAll()
    {
                                    "FROM Employee" , Employee.class );
                                    ery.getResultList();
```

2 ไฟล์
ใช้โค้ด 30 บรรทัดขึ้นไป

หลังจากมี Spring Data JPA

```java
public interface EmployeeRepository extends JpaRepository< Employee, Integer >
{
    // No need to code here
}
```

1 ไฟล์
ใช้โค้ด 3 บรรทัด

ไม่จำเป็นต้อง
มีการใช้งาน class

Minimized Boilerplate Code

ก่อนมี Spring Data JPA

```
public interface Employee
{
    Employee save(Employee employee);

    List< Employee > findAll();

    Employee findById(Integer id);

    void deleteBy
}
```

```
@Repository
public class Employee
{
    private EntityManager
    public Employee
    {
        this.entit
    }

    @Overri
    public Em
    {
        ntityManager.merge( employee );

        return employee;
    }

    @Override
    public List < Employee > findAll()

        "FROM Employee" , Employee.class );
        ery.getResultList();
```

2 ไฟล์
ใช้โค้ด 30 บรรทัดขึ้นไป

หลังจากมี Spring Data JPA

```
public interface EmployeeRepository extends JpaRepository< Employee, Integer >
{
    // No need to code here
}
```

1 ไฟล์
ใช้โค้ด 3 บรรทัด

ไม่จำเป็นต้อง
มีการใช้งาน class

Feature เพิ่มเติม

- ฟีเจอร์ขั้นสูง ที่มีให้สำหรับ
  - Extends และ สามารถเพิ่ม custom queries เองด้วย JPQL ได้
  - Query Domain Specific Language (Query DSL)
  - สามารถเขียน custom methods ได้ (low-level coding)

Spring Data JPA - Reference Documentation

Spring Data JPA

- Mini-workshop: Spring Data JPA

- Ref: lab-s5-l12