SIDE EFFECTS

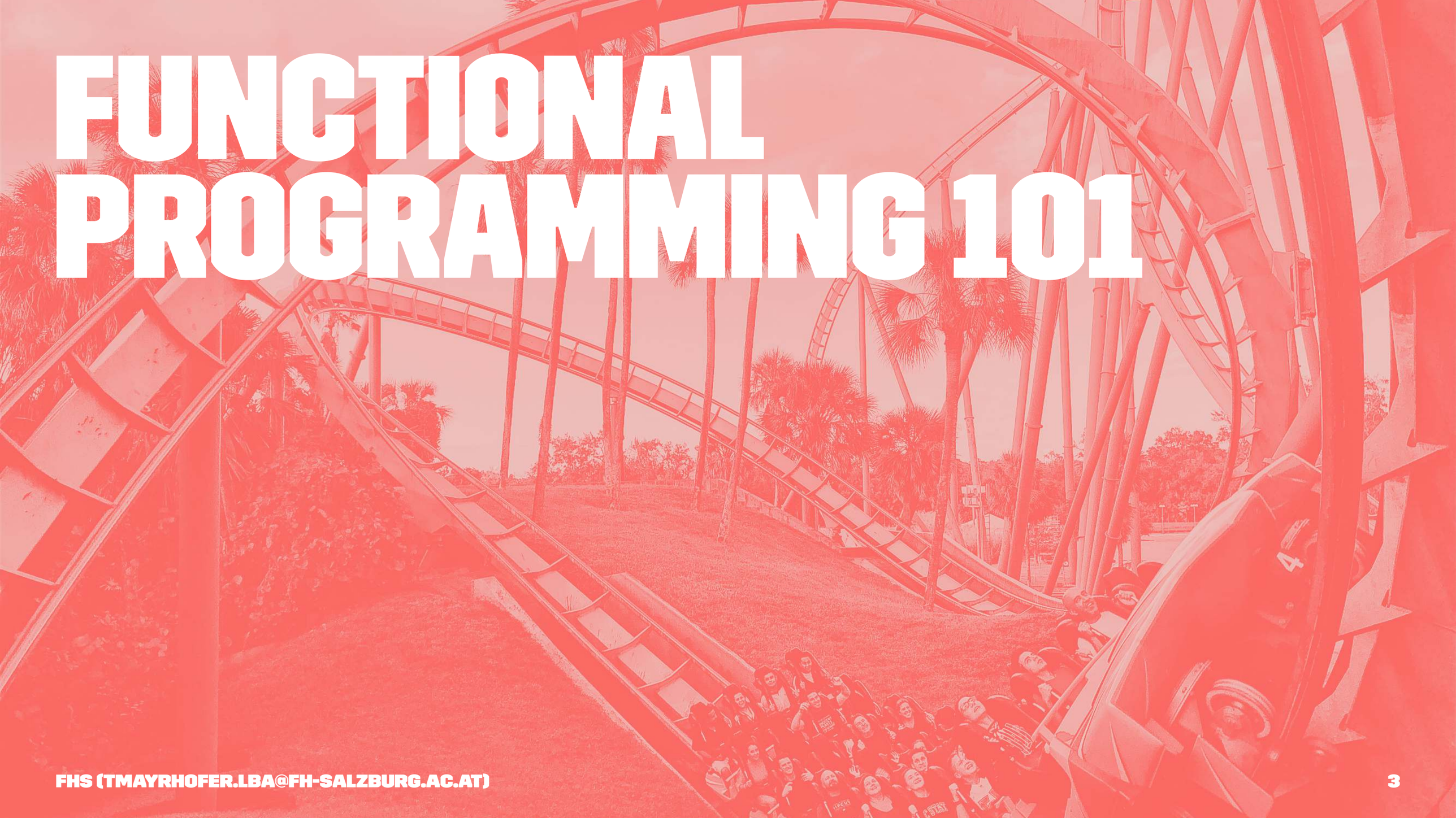
ROADMAP

» Functional Programming 101

» Side Effects with React

» State Management

FUNCTIONAL PROGRAMMING 101



FUNCTIONAL PROGRAMMING 101

» Immutability

» Pure functions

» Side Effect

FUNCTIONAL PROGRAMMING 101
WHAT IS FUNCTIONAL PROGRAMMING

"Applications developed in a functional style use side-effect free functions as their main building blocks. (Made up definition by myself)"

FUNCTIONAL PROGRAMMING 101
WHY FUNCTIONAL PROGRAMMING

- More testable
  - pure functions simplify testing
- Declarative APIs which are easier to reason about
- Easy concurrency because of statelessness and immutability
  - State is pushed out of the application core to the boundaries

FUNCTIONAL PROGRAMMING - IMMUTABILITY

“An immutable data structure is an object that doesn't allow us to change its value. (Remo H. Jansen)”

FUNCTIONAL PROGRAMMING - IMMUTABILITY

IMMUTABLE OBJECTS IN JS

```
const immutableObject = Object.freeze({ test: 1 })
immutableObject.test = 10
console.log(immutableObject) // => { test: 1 }
```

FUNCTIONAL PROGRAMMING - IMMUTABILITY
CHANGING AN IMMUTABLE VALUE

```
const immutableObject = Object.freeze({ a: 1, b: 2 })
const updatedObject = Object.freeze({ ...immutableObject, a: 2 })
console.log(updatedObject) // => { a: 2, b: 2 }
```

FUNCTIONAL PROGRAMMING - IMMUTABILITY

UNFREEZE AN OBJECT

```
const immutableObject = Object.freeze({ test: 1 })
const unfrozenCopy = { ...immutableObject }
```

FUNCTIONAL PROGRAMMING - IMMUTABILITY

WHY IMMUTABILITY

» race conditions impossible

» state of the application is easier to reason about

» easier to test

FUNCTIONAL PROGRAMMING - SIDE EFFECTS

“A side effect is a change of system state or observable interaction with the outside world that occurs during the calculation of a result. (Chris Barbour)”

FUNCTIONAL PROGRAMMING - SIDE EFFECTS

SOME SIDE EFFECTS

» DB/HTTP calls

» changing the file system

» querying the DOM

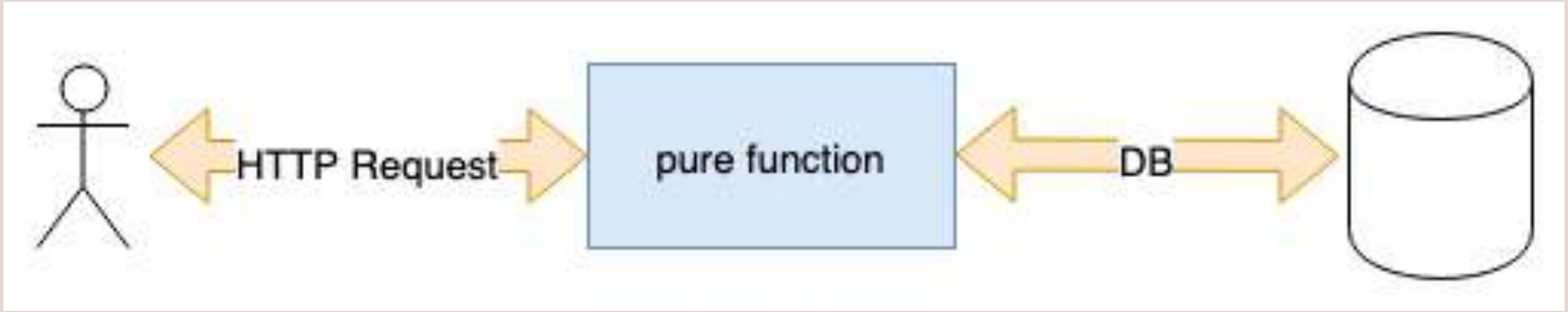
» printing/logging

» accessing system state (eg. Clock, Geolocation,...)

FUNCTIONAL PROGRAMMING - SIDE EFFECTS
WHERE TO DEAL WITH SIDE EFFECTS

» Moved to the boundaries of the system

» Business logic stays pure functional



FUNCTIONAL PROGRAMMING - PURE FUNCTIONS

- A function is considered pure when:
  - for the same input it always returns the same output
  - it has no side effects
  - no mutation of non-local state

```
const add = (a, b) => a + b
```

FUNCTIONAL PROGRAMMING - PURE FUNCTIONS

PURE OR IN-PURE

```
const array = [1, 2, 3, 4, 5, 6]
const fn1 = (array) => array.slice(0, 3)
const fn2 = (array) => array.splice(0, 3)
const fn3 = (array) => array.shift()
const fn4 = (array) => array.pop()
const fn5 = (array) => array.sort((a, b) => a - b)
const fn6 = (array) => [...array].sort((a, b) => a - b)
const fn7 = (array) => array.map((item) => item * 2)
const fn8 = (array) => array.forEach((item) => console.log(item))
```

FUNCTIONAL PROGRAMMING - PURE FUNCTIONS

PURE OR IN-PURE

```
const array = [1, 2, 3, 4, 5, 6]
const fn1 = (array) => array.slice(0, 3) // ✅
const fn2 = (array) => array.splice(0, 3) // 🚫
const fn3 = (array) => array.shift() // 🚫
const fn4 = (array) => array.pop() // 🚫
const fn5 = (array) => array.sort((a, b) => a - b) // 🚫
const fn6 = (array) => [...array].sort((a, b) => a - b) // ✅
const fn7 = (array) => array.map((item) => item * 2) // ✅
const fn8 = (array) => array.forEach((item) => console.log(item)) // 🚫
```

FUNCTIONAL PROGRAMMING - PURE FUNCTIONS
PURE OR IN-PURE

```
const config = { minimumAge: 18 }
const isAllowedToDrink = (age) => age >= config.minimumAge
```

```
const config = { minimumAge: 18 }
const isAllowedToDrink = (age) => age >= config.minimumAge
```

FUNCTIONAL PROGRAMMING - PURE FUNCTIONS
PURE OR IN-PURE

```
const config = { minimumAge: 18 }
const isAllowedToDrink = (age) => age >= config.minimumAge
```

FUNCTIONAL PROGRAMMING - PURE FUNCTIONS
PURE OR IN-PURE

```
const config = { minimumAge: 18 }
const isAllowedToDrink = (age) => age >= config.minimumAge

// both are not pure. const saves the pointer. config is still mutable
isAllowedToDrink(18) // true
config.minimumAge = 19
isAllowedToDrink(18) // false
```

FUNCTIONAL PROGRAMMING - PURE FUNCTIONS
PURE OR INPURE? 2/2

```
const config = Object.freeze({ minimumAge: 18 })
const isAllowedToDrink = (age) => age >= config.minimumAge

// freezing the config makes the function pure
isAllowedToDrink(18) // true
config.minimumAge = 19
isAllowedToDrink(18) // true
```

FUNCTIONAL PROGRAMMING - SUMMARY

- Immutability
  - Object can't be changed after its creation
- Side-Effects
  - Communication with the outside world (eg. db, http, ...)
- Pure-Functions
  - returns the same output for the same input

SIDE EFFECTS WITH REACT

SIDE EFFECTS WITH REACT
RECAP USEEFFECT

“The Effect Hook lets you perform side effects in function components”

SIDE EFFECTS WITH REACT
RECAP USEEFFECT

```
// Executed on every rerender
useEffect(() => {})

// Executed when component rendered initially
useEffect(() => {}, [])

// Executed when component rendered initially
// and when variable changes.
useEffect(() => {}, [variable])

// Cleanup when component unmounts (eg. eventHandlers, setInterval/setTimeout)
useEffect(() => {
  // do something fancy
  return () => { console.log('cleanup') }
}, [variable])
```

SIDE EFFECTS WITH REACT
RECAP USEEFFECT

```
export const MoneyTransactions = () => {
  const [moneyTransactions, setMoneyTransactions] = useState([])
  useEffect(() => {                                          // 1)
    fetch("http://localhost:3001/money-transaction")         // 2)
      .then((response) => response.json())                   // 3)
      .then((json) => setMoneyTransactions(json))            // 4)
  }, [])                                                     // 5)

  // 1) define the useEffect hook
  // 2) make the HTTP request to the backend
  // 3) get the JSON from the response
  // 4) set the response as state
  // 5) call the effect when the component is mounted

  // ... remaining component
}
```

SIDE EFFECTS WITH REACT
EXTRACT INTO CUSTOM HOOK

```
const useHTTPEffect = (endpoint) => {
  const [response, setResponse] = useState([])
  useEffect(() => {
    fetch(endpoint)
      .then((response) => response.json())
      .then((json) => setResponse(json))
  }, [endpoint])

  return response;
}

export const MoneyTransactions = () => {
  const moneyTransactions = useHttpEffect("http://localhost:3001/money-transaction")
  const users = useHttpEffect("http://localhost:3001/user")

  // ...
}
```

TASK (30 MIN)
FETCH AND DISPLAY MONEY-TRANSACTIONS

- npm i json-server -D
- add npm script
  - "start:server": "json-server --watch db.json --port 3001",
- add db.json
- fetch/display money-transactions and users

FEEDBACK

» Questions: tmayrhofer.lba@fh-salzburg.ac.at

» https://s.surveyplanet.com/x1ibwm85