FRP (Rx way)

MARCH 2017

AGENDA

- MOTIVATION
- FUNCTIONAL REACTIVE PROGRAMMING INTRO
- DEMO

MOORE'S LAW

Simplified version

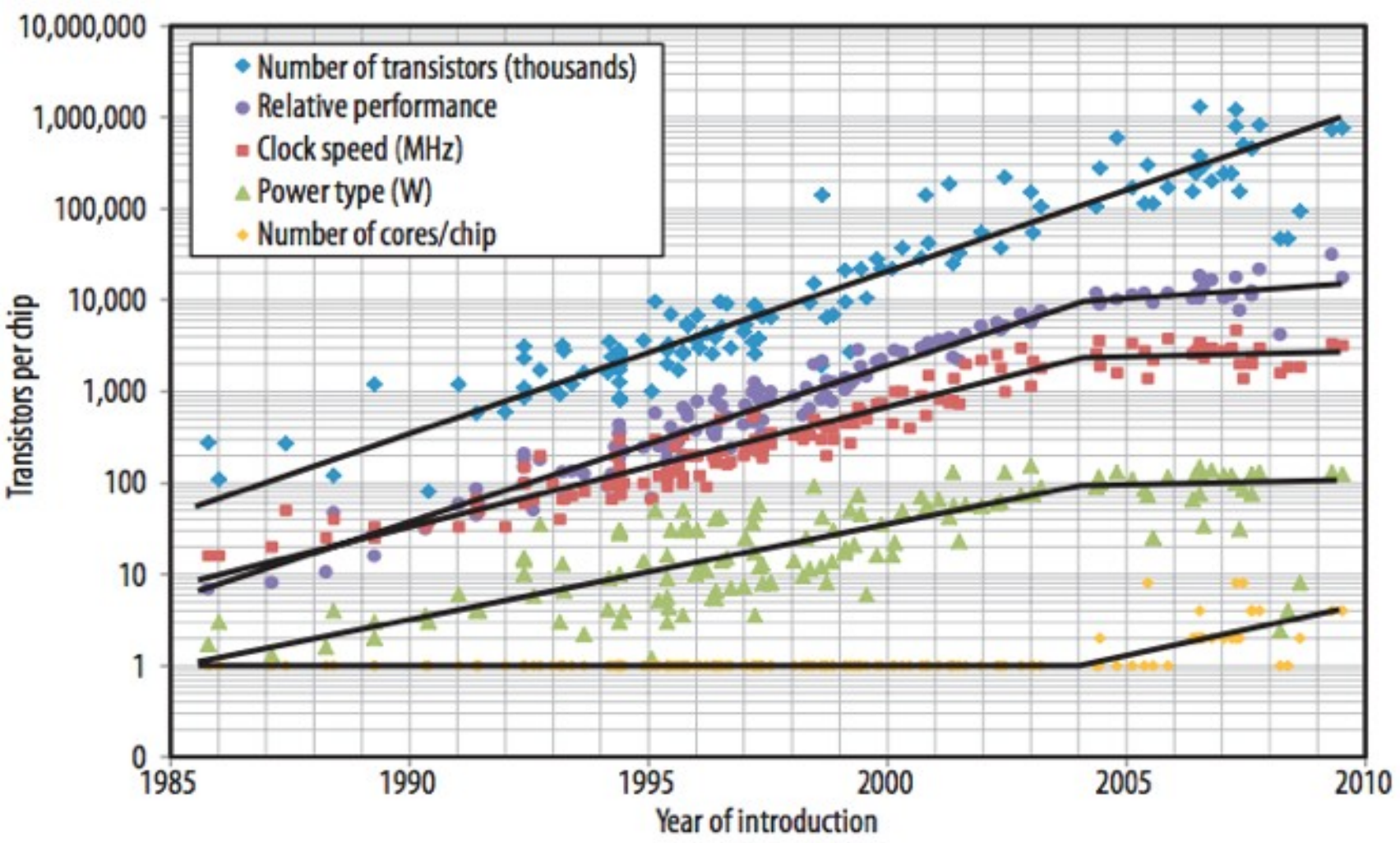
Processor speeds, or overall processing power for computers will double every two years.

You could count on hardware improvements and wait for your app's performance to increase

Real version

*The number of **transistors** on an affordable CPU would double every two years*

WHAT HAPPENED WITH THOSE EXTRA TRANSISTORS?



endava

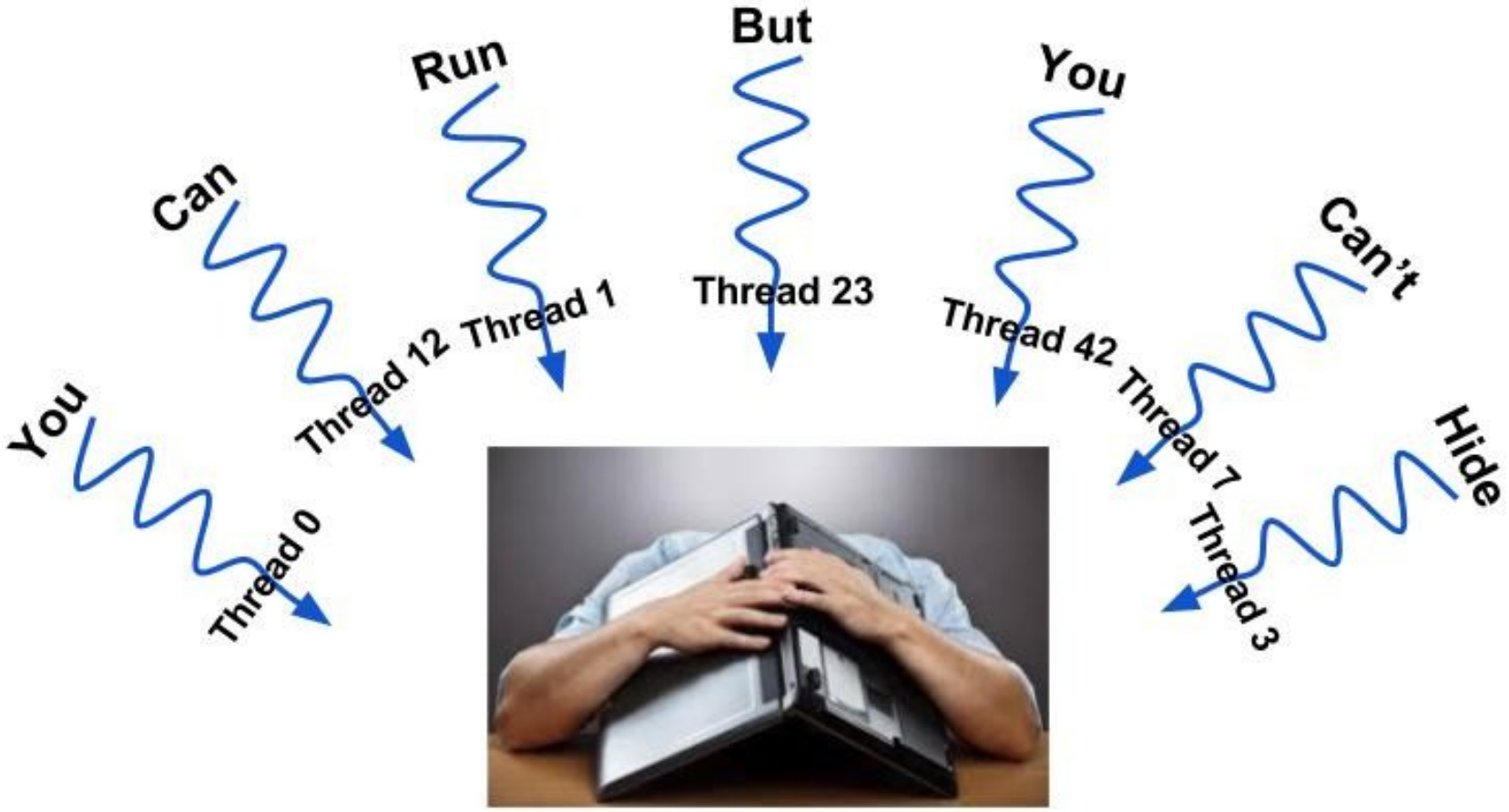
PARALLEL PROGRAMMING

Parallel programming is the only way to use hardware improvement to increase performance in the future!

CONCURRENT PROGRAMMING

- Internet:
  - December 2000 … 361 millions of users … 5.8% of world population
  - September 2016 .... 3675 millions of users … 50.1% of world population
- Web programming
- Communication with external devices
  - Sensors
  - Bluetooth
  - …

MULTITHREADING IS UNAVOIDABLE



endava

SHARED MUTABLE STATES

PROBLEM:

- Some states are “invalid” and can crash or block app
- Multiple points of access => hard to control states

PREVENTION:

- Access control - setters/getters (OOP)

SHARED MUTABLE STATES + MULTITHREADING

NONDETERMINISTIC BEHAVIOUR => PROGRAMMING HELL

Without multithreading, bugs are 100% reproducible.

- all conditions are under our control => we control execution order

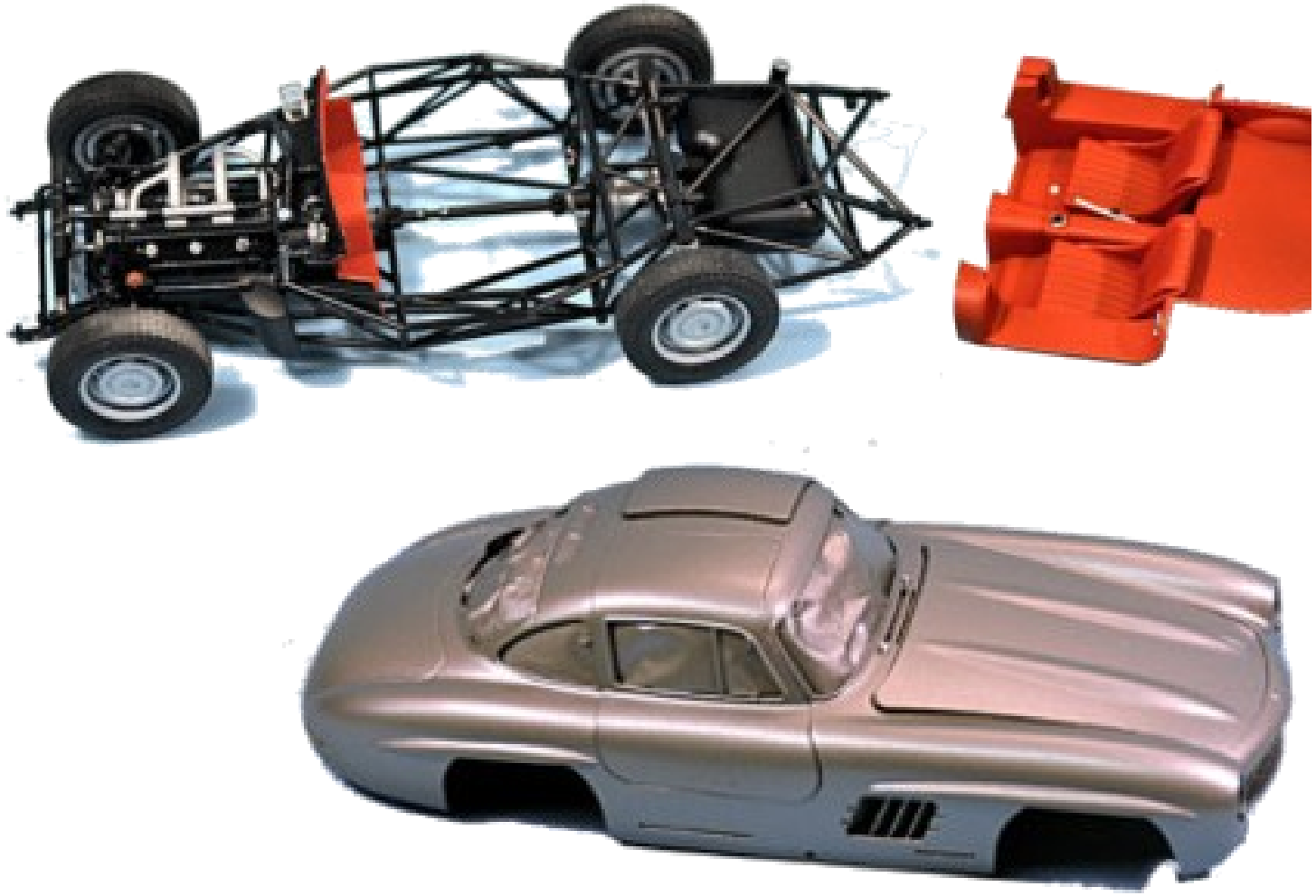
With multithreading, bugs are **intermittent** (<< 100%)

- some conditions are out of our control => execution order is unpredictable

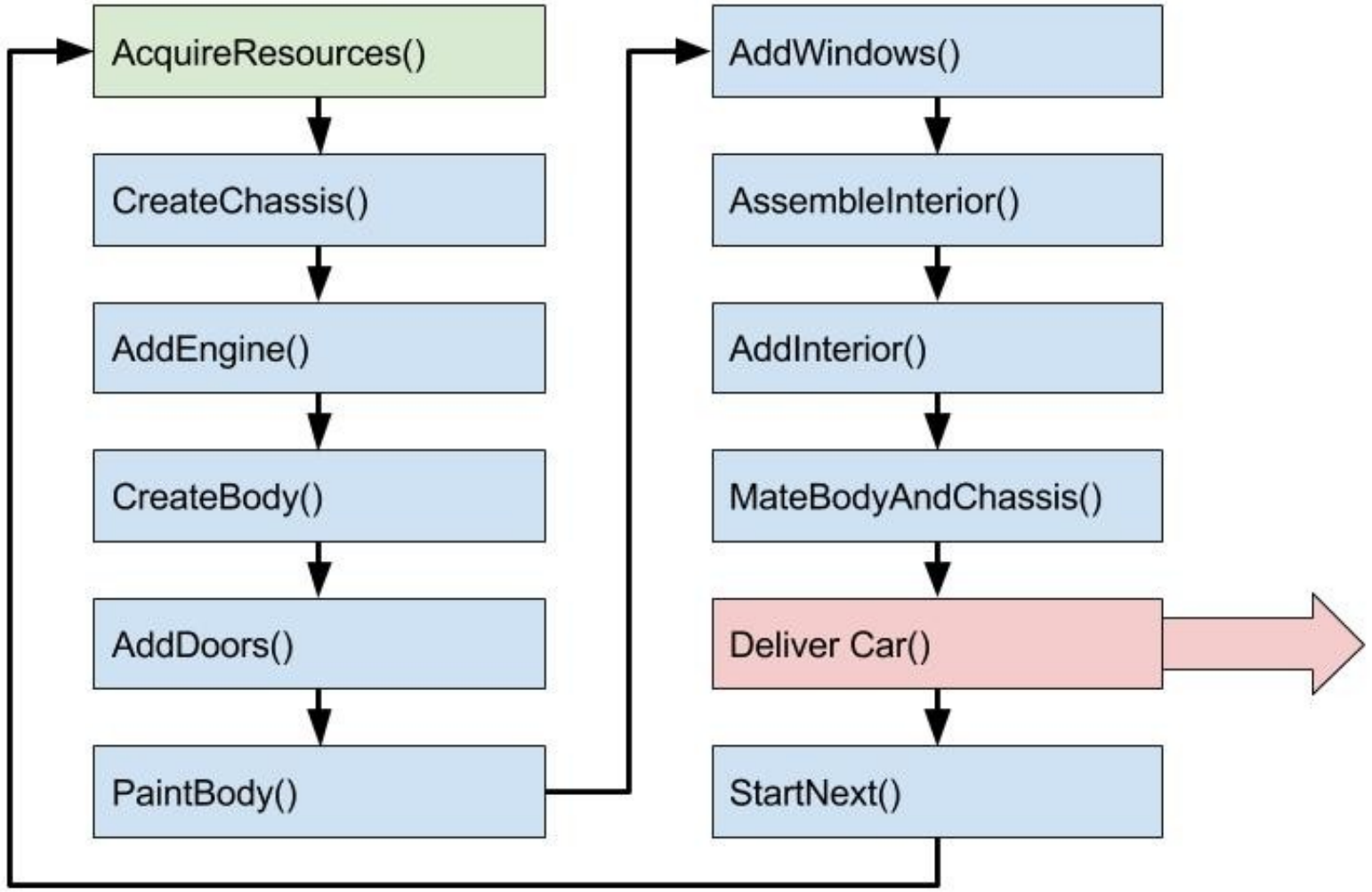
SOLUTIONS ?

- Remove Shared Mutable States completely (???)
  - Functional programming
- Remove Shared Mutable States as much as possible
  - Remove unnecessary states
  - Make const everything that can be const
  - Use some Functional programming techniques (work with collections, higher order functions)

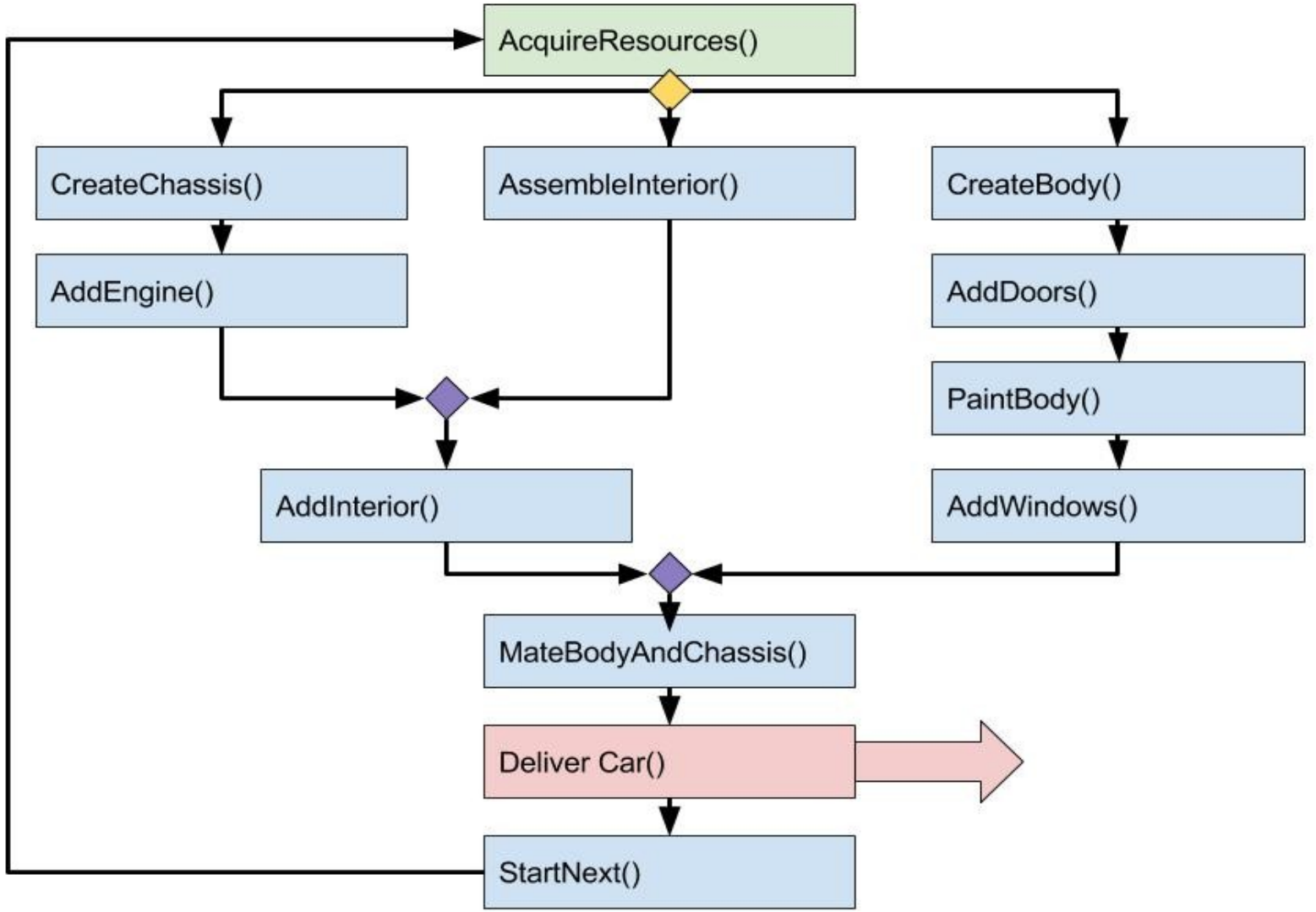
CAR ASSEMBLY



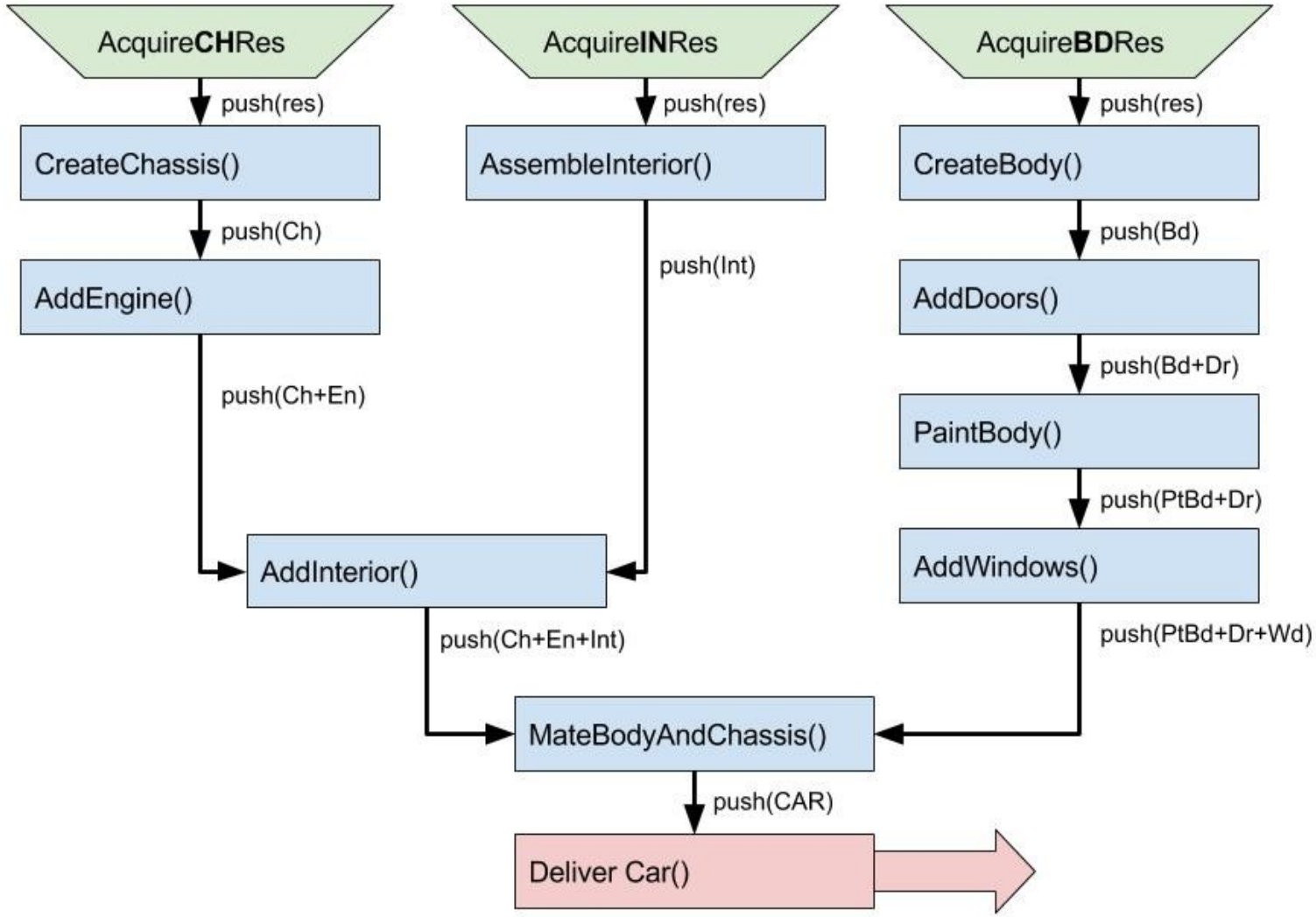
CREATE CAR EXAMPLE: SINGLE-THREADED



CREATE CAR EXAMPLE: MULTI-THREADED



CREATE CAR EXAMPLE: FRP

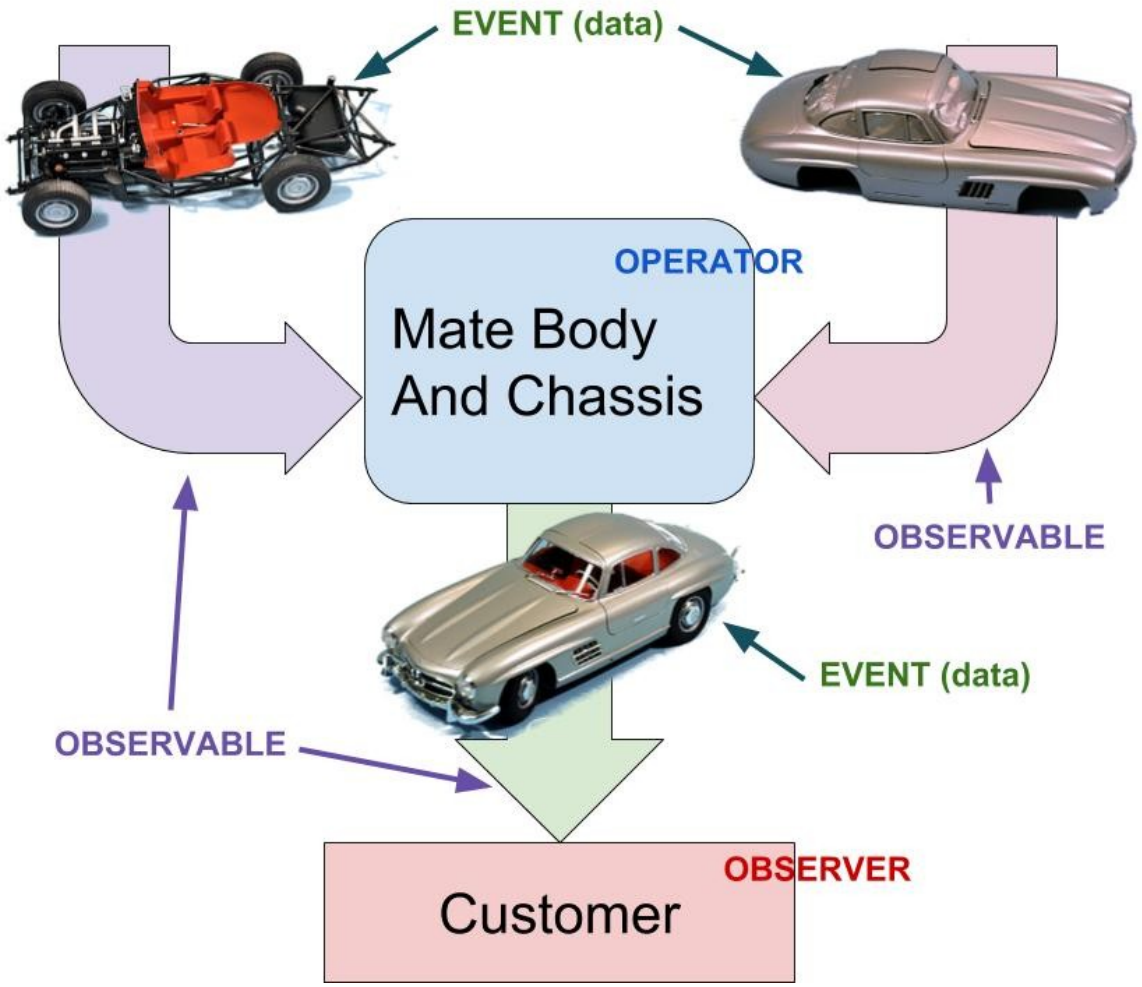


endava

AGENDA

- MOTIVATION
- FUNCTIONAL REACTIVE PROGRAMMING INTRO
- DEMO

MEET THE OBSERVABLE



endava

MEET THE OBSERVABLE

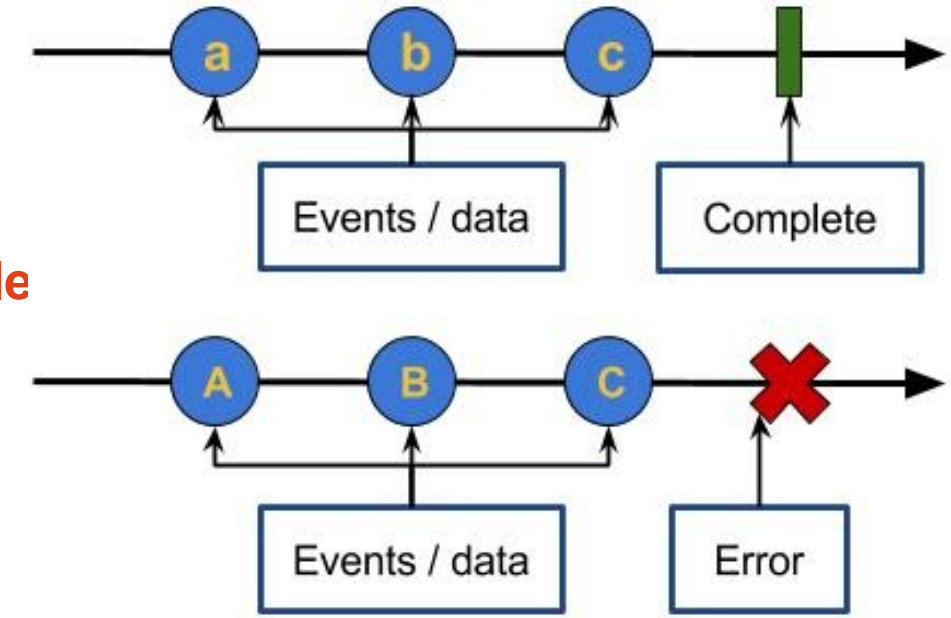
Observable is the object that represent the stream of data

Observer can subscribe to an **Observable**

Operator is basically a function that accept **Observable** as operand, instead of the data.

Observable can send three types of signals:

- **Value** signal
- **Completed** signal
- **Error** signal



endava

OBSERVABLE VS ITERABLE

Iterable:

A container object capable of returning its members one at a time.

It is possible to implement all functional operators for work with enumerable collections to work with observables ! (i.e. map, zip, filter ...)

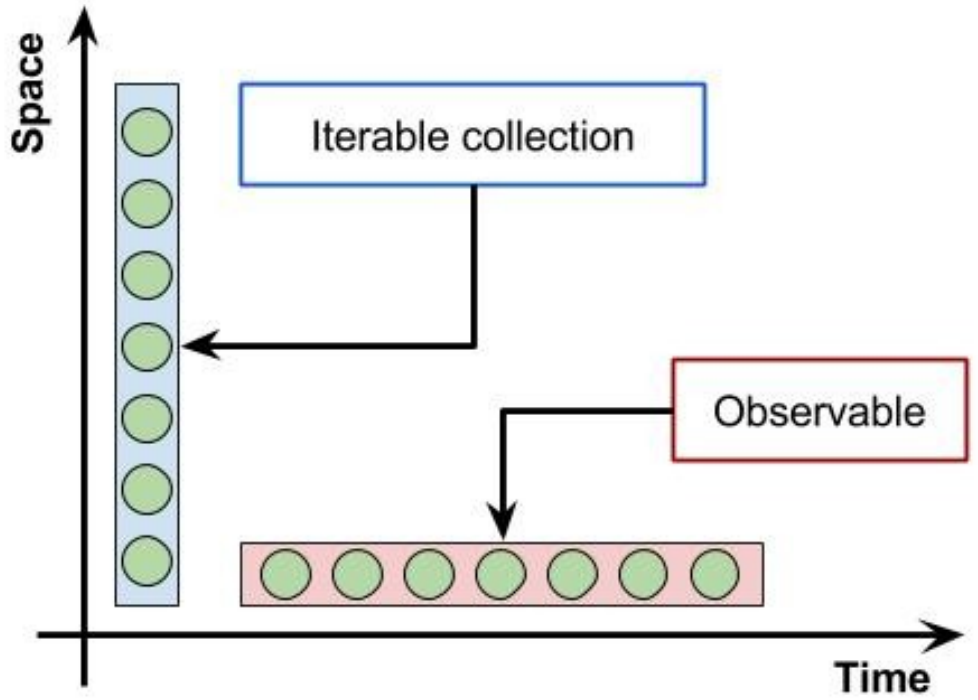
For example:

someIterable*.map**(someFunction) -> ***newIterable

someObservable*.map**(someFunction) -> ***newObservable

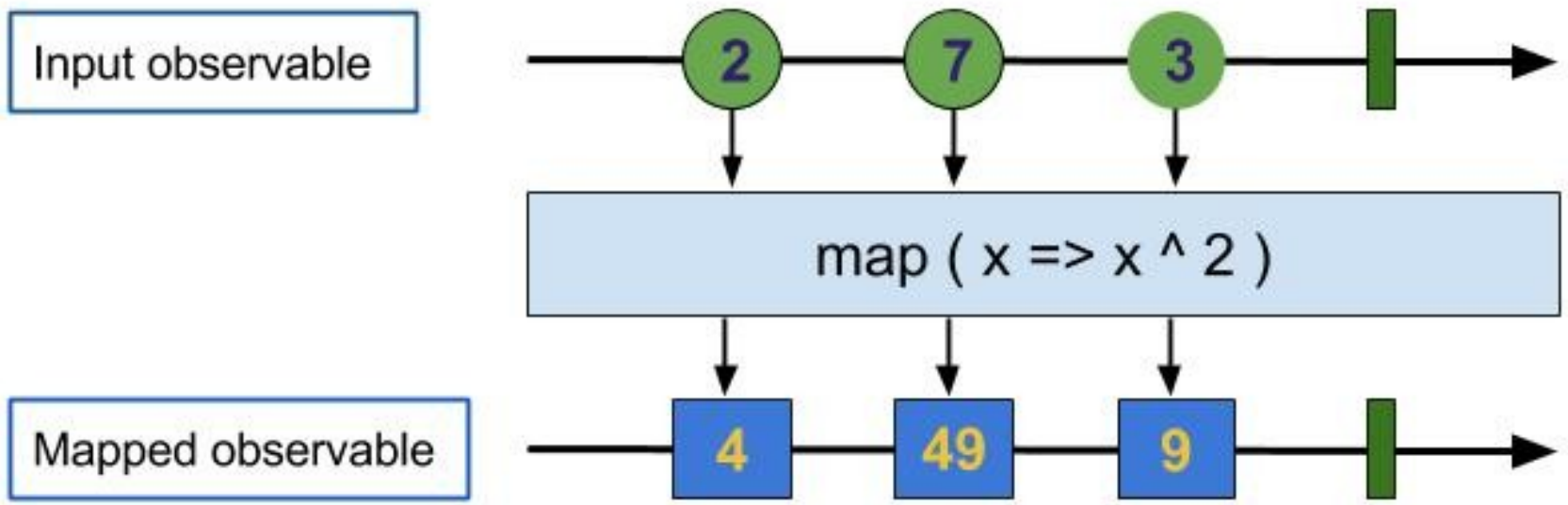
OBSERVABLE VS ITERABLE

- At one point of time ALL of the **Iterable** elements are accessible
- Only one element of the **Observable** is accessible at the moment of its creation
- **Iterable** have a pull based access
- **Observable** have a push based access
- It is possible to convert from **Observable** to **Iterable** and vice versa



endava

MAP OPERATOR

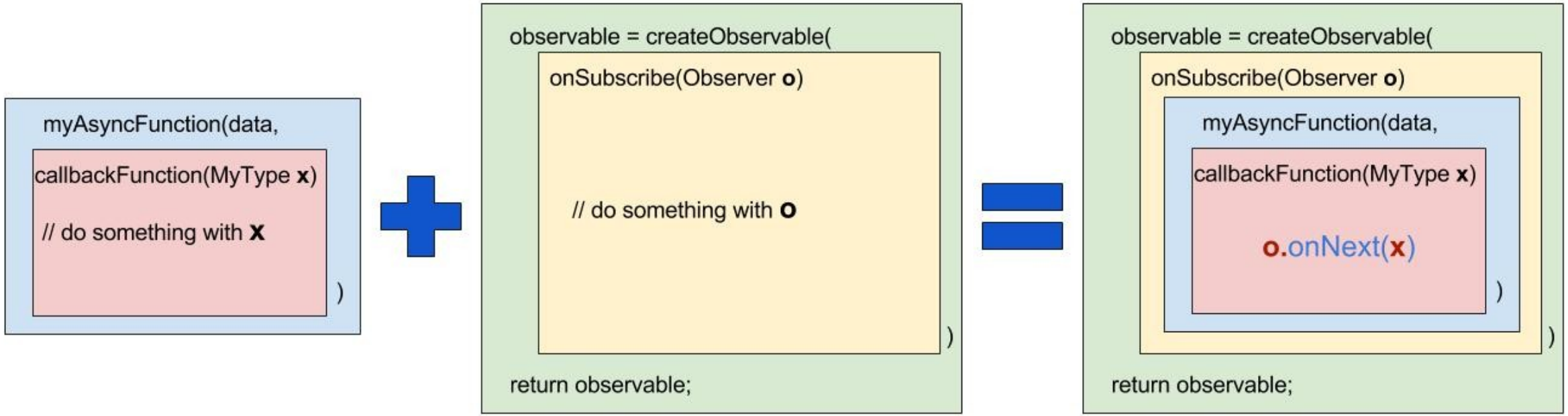


endava

CALLBACK VS OBSERVABLE (BRIDGING)

void *myAsyncFunction*(**Data** someData, **Callback<MyType>** myCallback);

Observable<MyType> *myFrpAsyncFunction*(**Data** someData);



endava

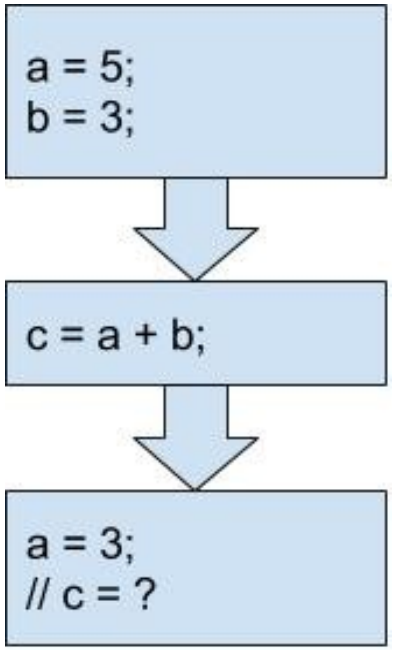
AGENDA

- MOTIVATION
- FUNCTIONAL REACTIVE PROGRAMMING INTRO
- DEMO

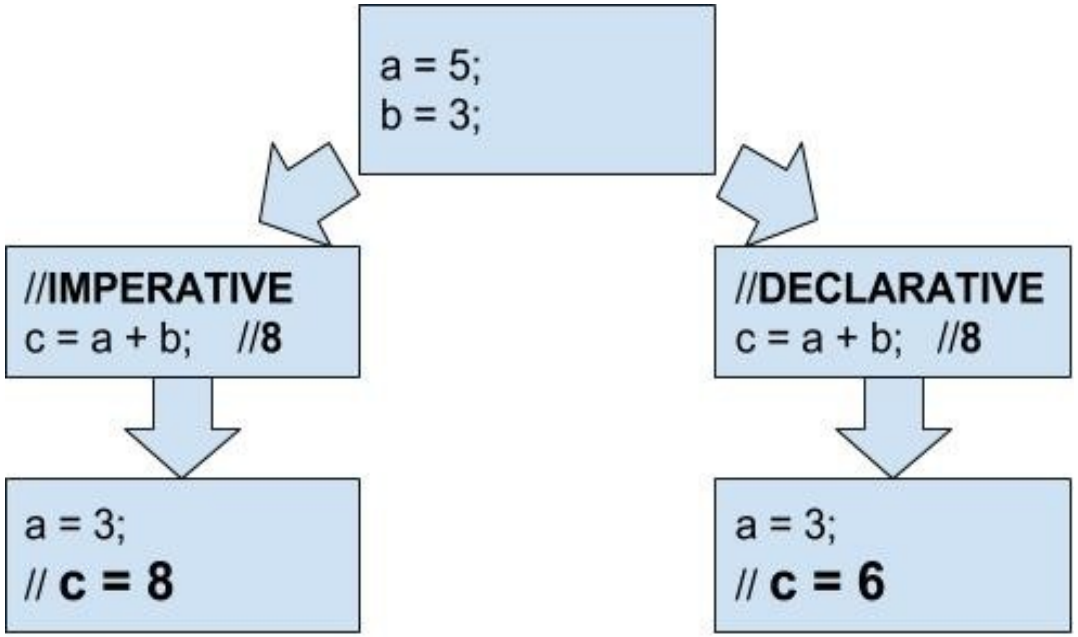
DEMO: DOWNLOAD IMAGE

endava

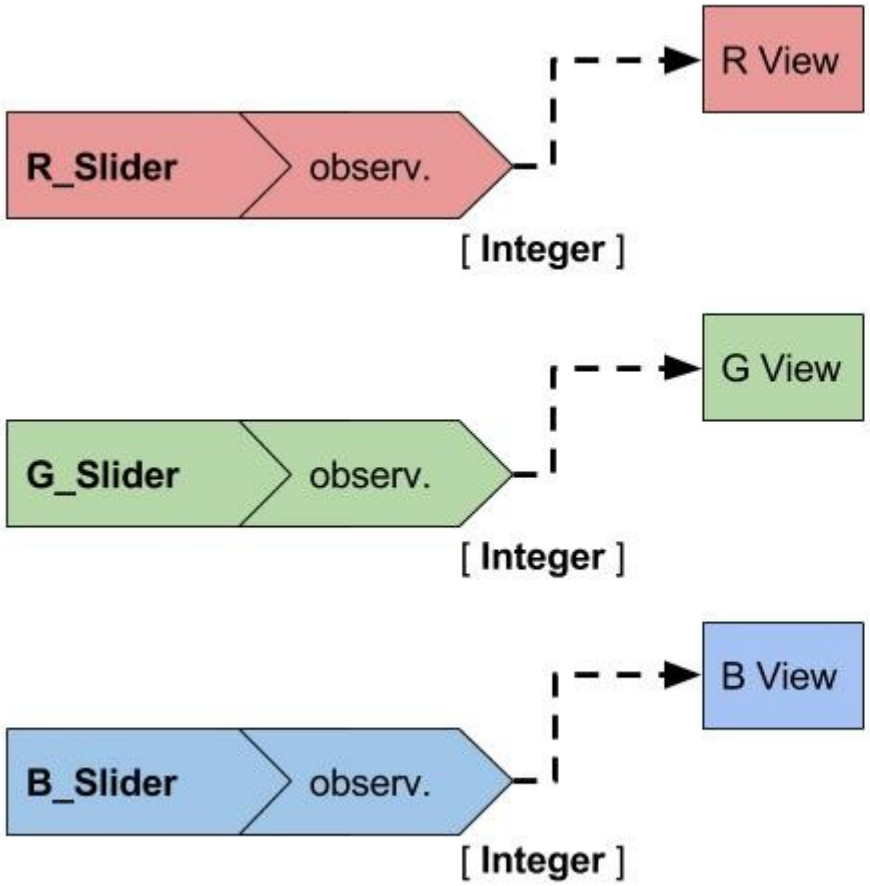
IMPERATIVE VS. DECLARATIVE



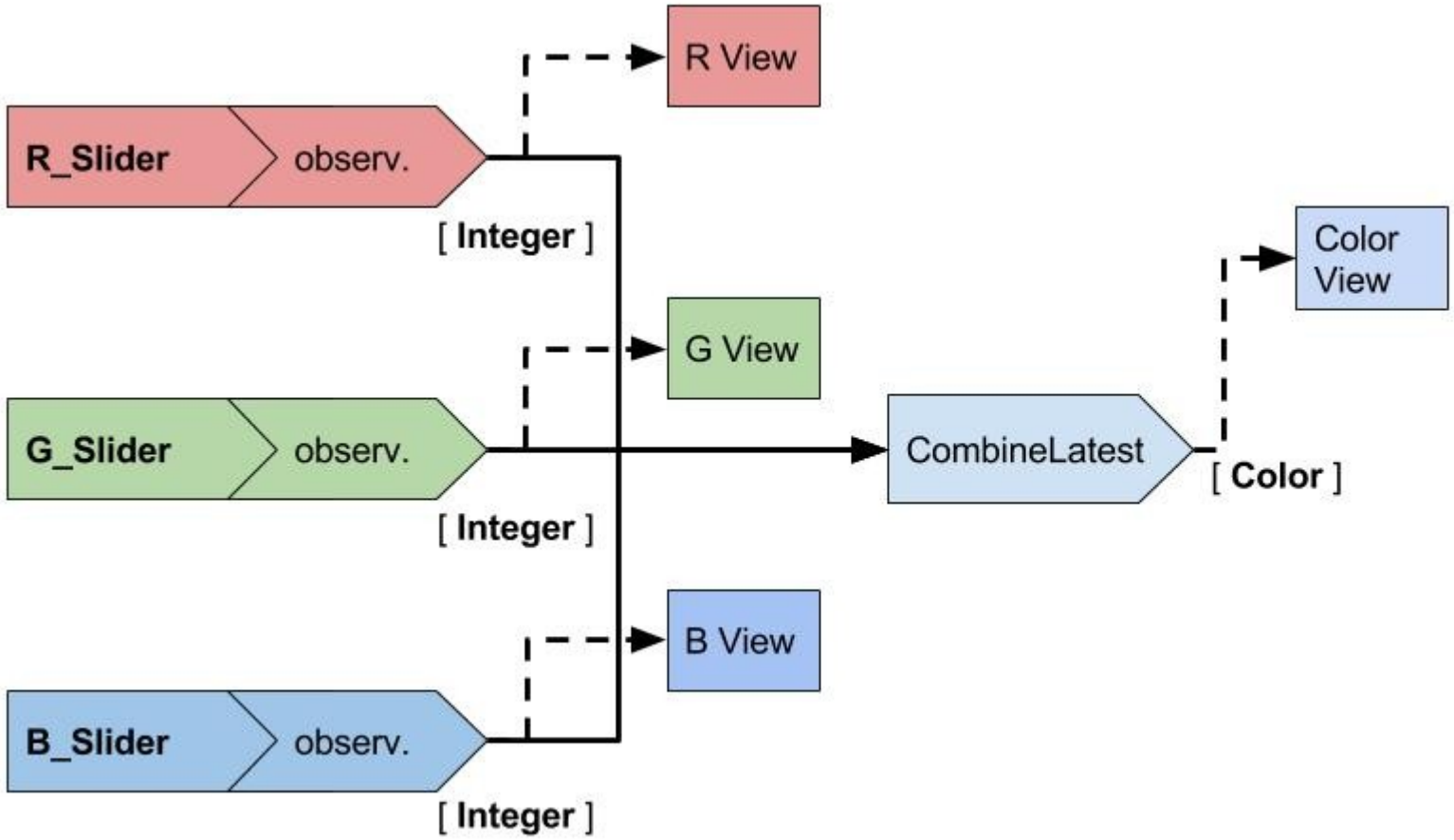
IMPERATIVE VS. DECLARATIVE



DEMO: RGB SLIDERS (PART 1)

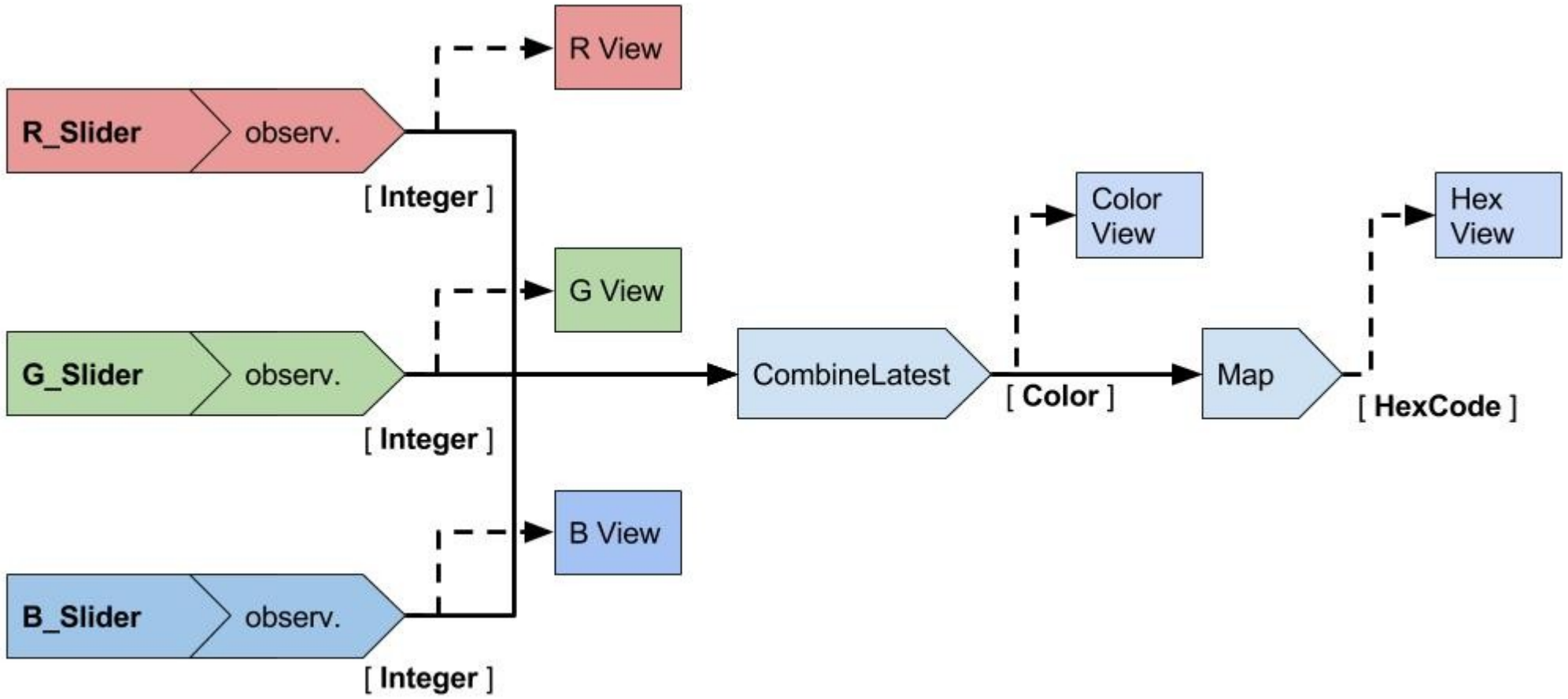


DEMO: RGB SLIDERS (PART 2)



endava

DEMO: RGB SLIDERS (PART 3)



endava

EXTERNAL LINKS

http://reactivex.io/

http://rxmarbles.com/

RASKO GOJKOVIC

SENIOR IOS DEVELOPER

SLOBODAN PAVIC

ANDROID DEVELOPER