PYCON
KOREA 2017

Python Asynchronous Programming with Salt Stack (tornado, asyncio) and RxPY

PyCon Korea 2017

Kim Sol
kstreee@gmail.com

PYCON
KOREA 2017

Python Asynchronous Programming with Salt Stack (tornado, asyncio) and RxPY

Kim Sol
kstreee@gmail.com

NHN ENTERTAINMENT

Interest

The safeness of scalable—code and performance—programs.

Interest

Methods to Build Safe and Scalable Programs

1. Born to be Googler

2. Software Analysis, Checking errors before execution
   Static Analysis, Abstract Interpretation, Sparrow, FB Infer

3. Software Verification, Implementing programs with robust mathematical basis
   Coq, Machine-checked Proofs, Type Inference, Robust Type Systems, …

4. Frameworks, Using safeness & productivity of frameworks
   Asynchronous Frameworks, Reactive Programming, …

Interest

Methods to Build Safe and Scalable Programs

1. Born to be Googler

2. Software Analysis, Checking errors before execution
   Static Analysis, Abstract Interpretation, Sparrow, **FB Infer**

3. Software Verification, Implementing programs with robust mathematical basis
   Coq, Machine-checked Proofs, Type Inference, Robust Type Systems, …

4. Frameworks, Using safeness & productivity of frameworks
   Asynchronous Frameworks, **Reactive Programming**, …

What I have been doing in NHN Ent.

Interest

Methods to Build Safe and Scalable Programs

1. Born to be Googler

2. Software Analysis, Checking errors before execution
   Static Analysis, Abstract Interpretation, Sparrow, FB Infer

3. Software Verification, Implementing programs with robust mathematical basis
   Coq, Machine-checked Proofs, Type Inference, Robust Type Systems, …

4. Frameworks, Using safeness & productivity of frameworks
   👉 **Asynchronous Frameworks**, **Reactive Programming**, …

What I will discuss in this talk

INDEX

1. Preliminary of Asynchronous Programming

Preliminary
Asynchronous Programming
When do we have to use Async?
Why Async Frameworks Matters?

2. Async Frameworks Details

Code Scalability
Reactive Programming (RxPY)
Why Reactive Programming Matters?

INDEX

1. Preliminary of Asynchronous Programming

Preliminary
Asynchronous Programming
When do we have to use Async?
Why Async Frameworks Matters?

2. Async Frameworks Details

Code Scalability
Reactive Programming (RxPY)
Why Reactive Programming Matters?

INDEX

1. Preliminary of Asynchronous Programming

Preliminary

Asynchronous Programming

When do we have to use Async?

Why Async Frameworks Matters?

2. Async Frameworks Details

Code Scalability

Reactive Programming (RxPY)

Why Reactive Programming Matters?

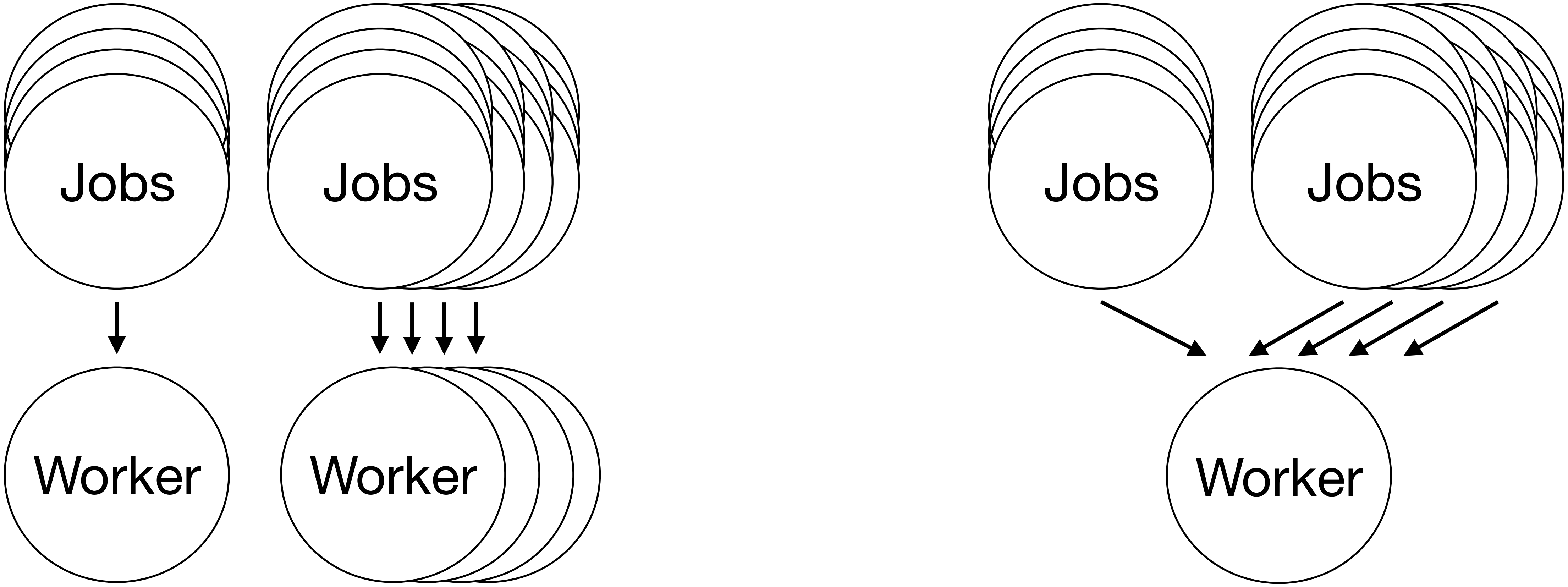
Preliminary

Parallelism	-	Concurrency
Blocking I/O	-	Non-blocking I/O
Synchronous Programming (?)	-	Asynchronous Programming

https://kstreee.github.io/techmemo/async_and_webframework.pdf

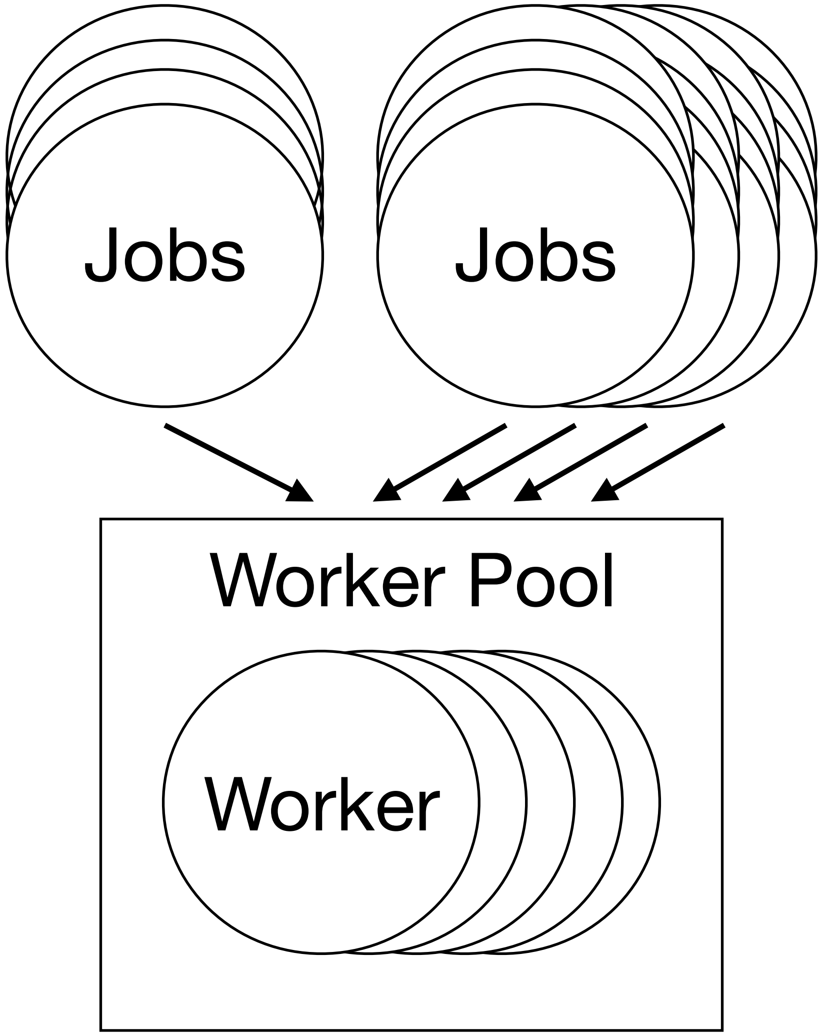
Preliminary

Parallelism - Concurrency



Preliminary

Parallelism + Concurrency



Preliminary

Blocking - Non-blocking I/O

```
while not_finished:
    data = socket.recv(buf_size)
    do_something(data)
```

Preliminary

Blocking - Non-blocking I/O

```
while not_finished:
    try:
        data = socket.recv(buf_size)
        do_something(data)
    except socket.error as e:
        if e.args[0] in _ERRNO_WOULDBLOCK:
            # DO SOMETHING ELSE
```

Preliminary

Asynchronous Programming

```
data = yield tornado.iostream.read_until('\r\n')
```

Preliminary

Asynchronous Programming

```
data = yield tornado.iostream.read_until('\r\n')
```

Implementation Details

Non-blocking socket I/O

```
while True:
    try:
        data = socket.recv(buf_size)
    except socket.error as e:
        if e.args[0] in _ERRNO_WOULDBLOCK:
            # DO SOMETHING ELSE
```

Preliminary

Asynchronous Programming

```
data = yield tornado.iostream.read_until('\r\n')
```

Implementation Details

Could use any other methods, even, those don't need to be non-blocking I/O.

INDEX

1. Preliminary of Asynchronous Programming

Preliminary
Asynchronous Programming
When do we have to use Async?
Why Async Frameworks Matters?

2. Async Frameworks Details

Code Scalability
Reactive Programming (RxPY)
Why Reactive Programming Matters?

Asynchronous Programming

Providing Concurrency by Scheduling Events

Asynchronous Programming

Providing Concurrency by Scheduling Events

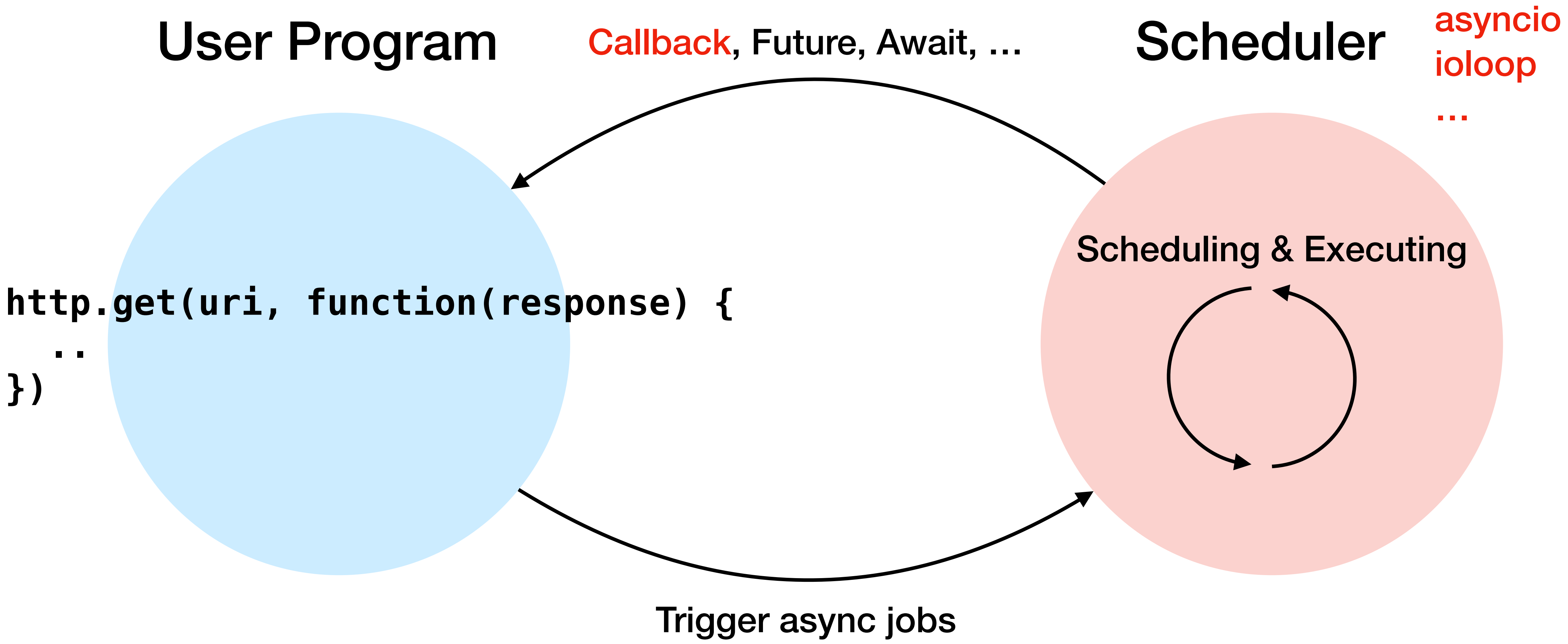
```
data = yield tornado.iostream.write(data)
```

Asynchronous Programming

How to communicate with a scheduler ?

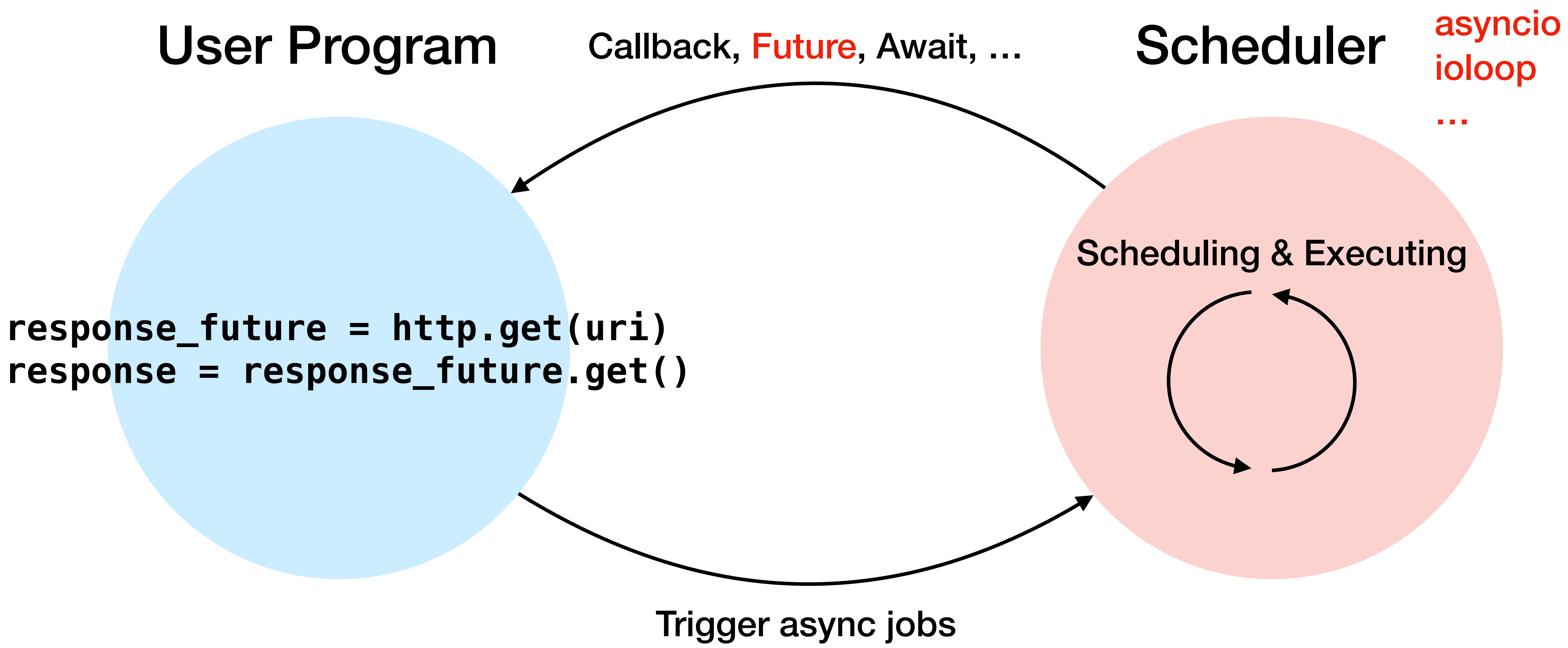
ex) Callback, Future, Promise, Await…

Asynchronous Programming



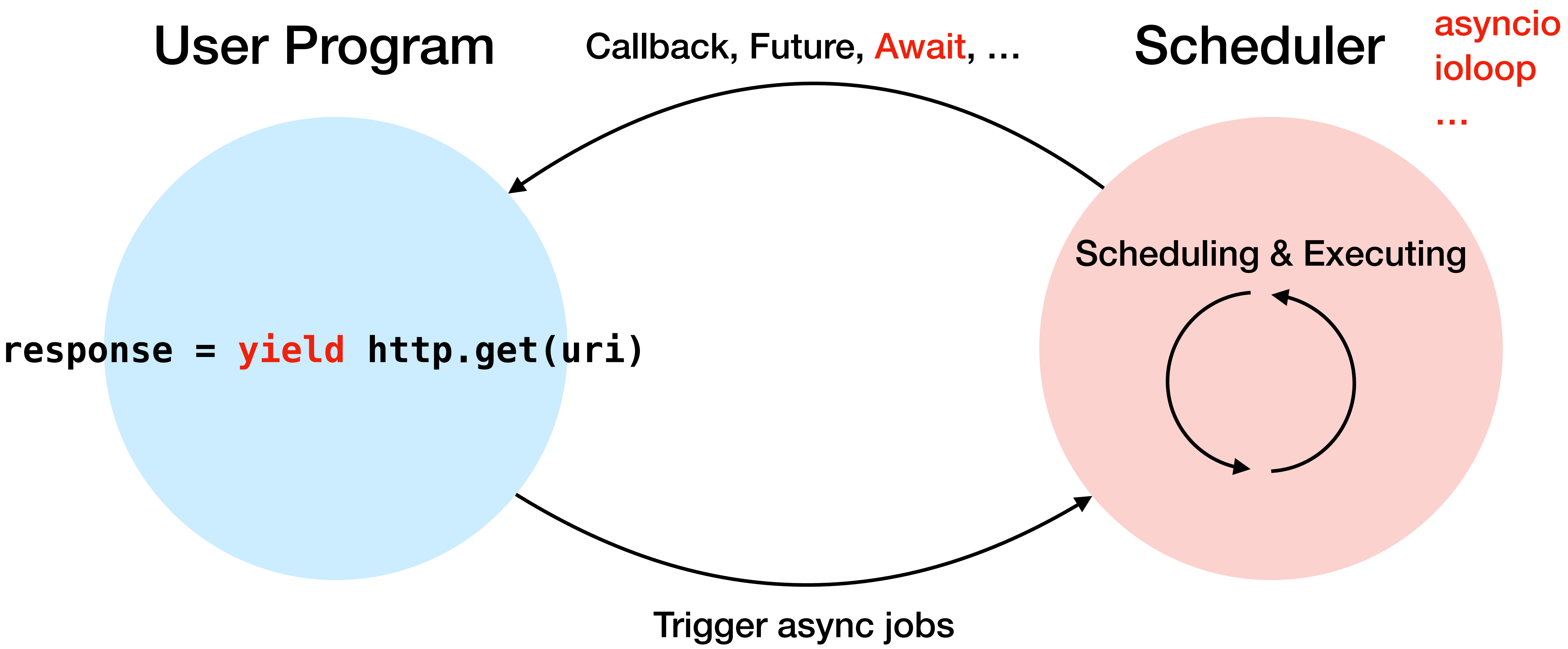
```
http.get(uri, function(response) {
   ..
})
```

Asynchronous Programming



```
response_future = http.get(uri)
response = response_future.get()
```

Asynchronous Programming



User Program

Callback, Future, Await, ...

Scheduler

asyncio
ioloop
...

```
response = yield http.get(uri)
```

Scheduling & Executing

Trigger async jobs

INDEX

1. Preliminary of Asynchronous Programming

Preliminary
Asynchronous Programming
When do we have to use Async?
Why Async Frameworks Matters?

2. Async Frameworks Details

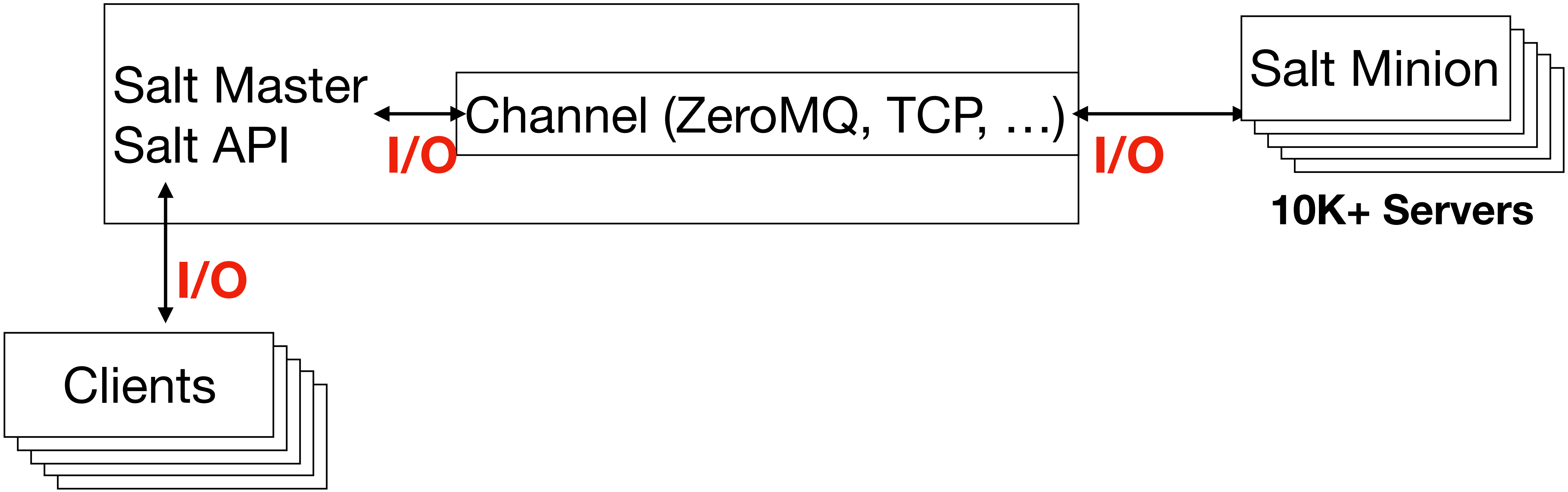
Code Scalability
Reactive Programming (RxPY)
Why Reactive Programming Matters?

When do we have to use Async?

Massive I/O

When do we have to use Async?

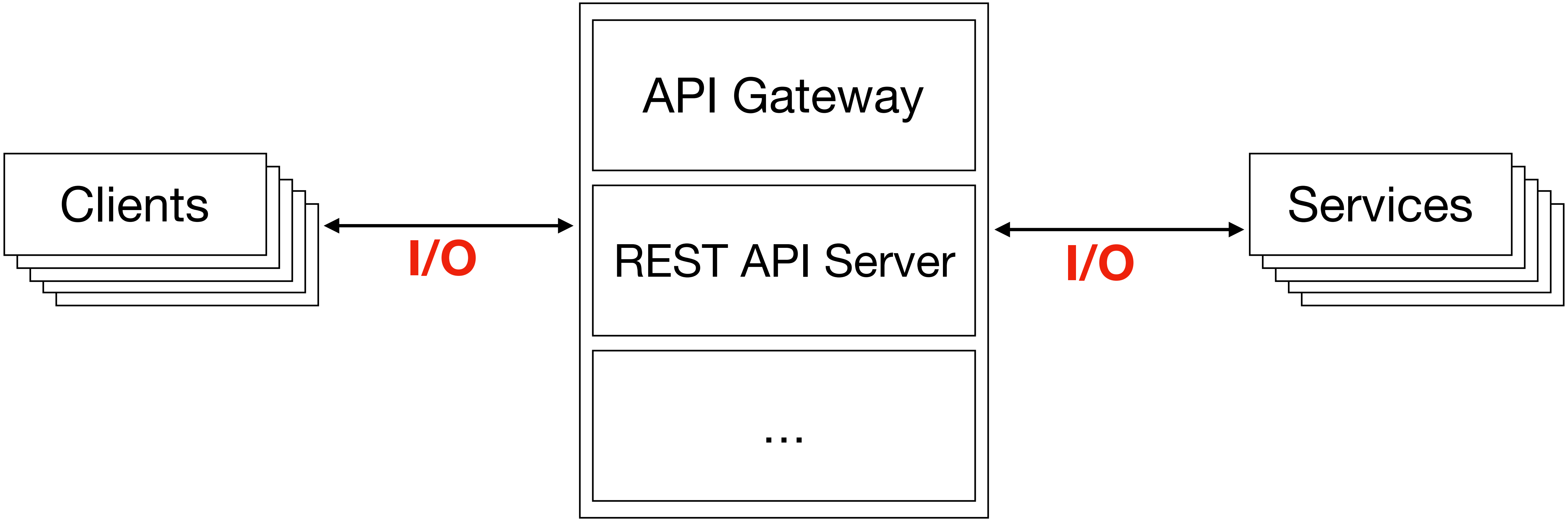
Salt Stack with Tornado in NHN Ent.



Toast Cloud Products, (TC Deploy, RDS, …),
Other NHN Ent. Internal Systems, …
Total : 10+ Systems

When do we have to use Async?

Services with Massive I/O



INDEX

1. Preliminary of Asynchronous Programming

Preliminary
Asynchronous Programming
When do we have to use Async?
Why Async Frameworks Matters?

2. Async Frameworks Details

Code Scalability
Reactive Programming (RxPY)
Why Reactive Programming Matters?

Why Async Frameworks Matters?

The safeness of scalable—code and performance—programs.

Why Async Frameworks Matters?

Methods to Build Safe and Scalable Programs

1. Born to be Googler

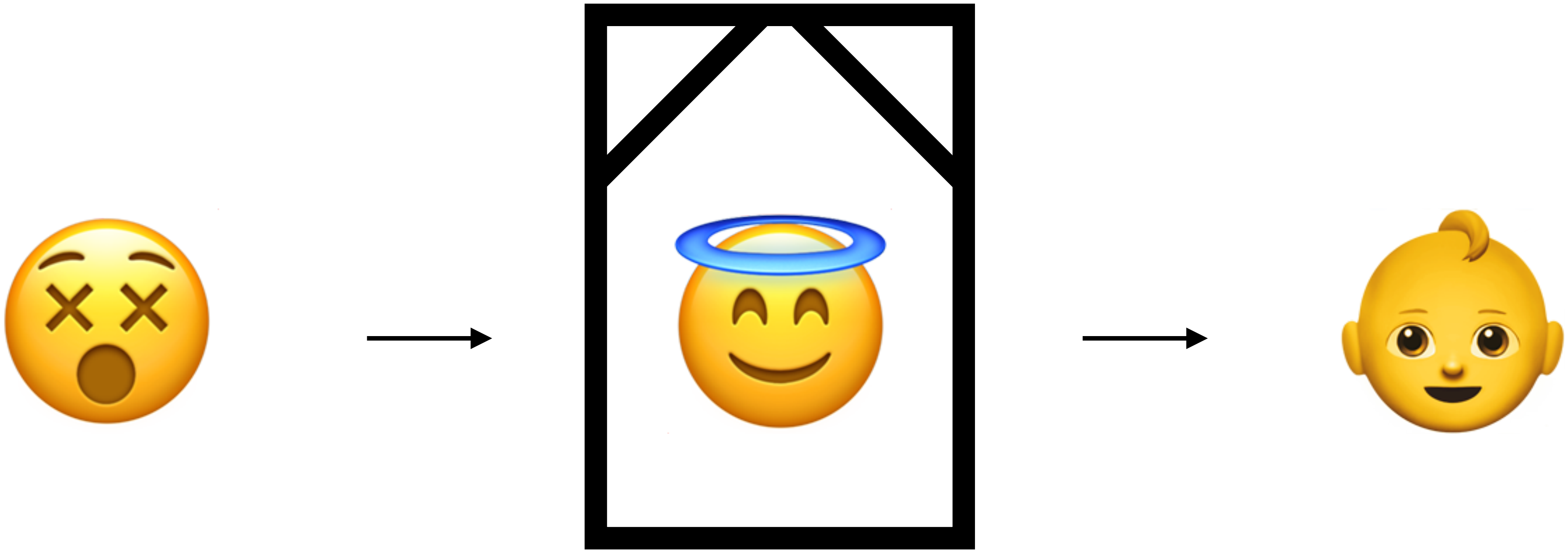
2. Software Analysis, Checking errors before execution
   Static Analysis, Abstract Interpretation, Sparrow, FB Infer

3. Software Verification, Implementing programs with robust mathematical basis
   Coq, Machine-checked Proofs, Type Inference, Robust Type Systems, …

👉 4. Frameworks, Using safeness & productivity of frameworks
   Asynchronous Frameworks, Reactive Programming, …

Why Async Frameworks Matters?

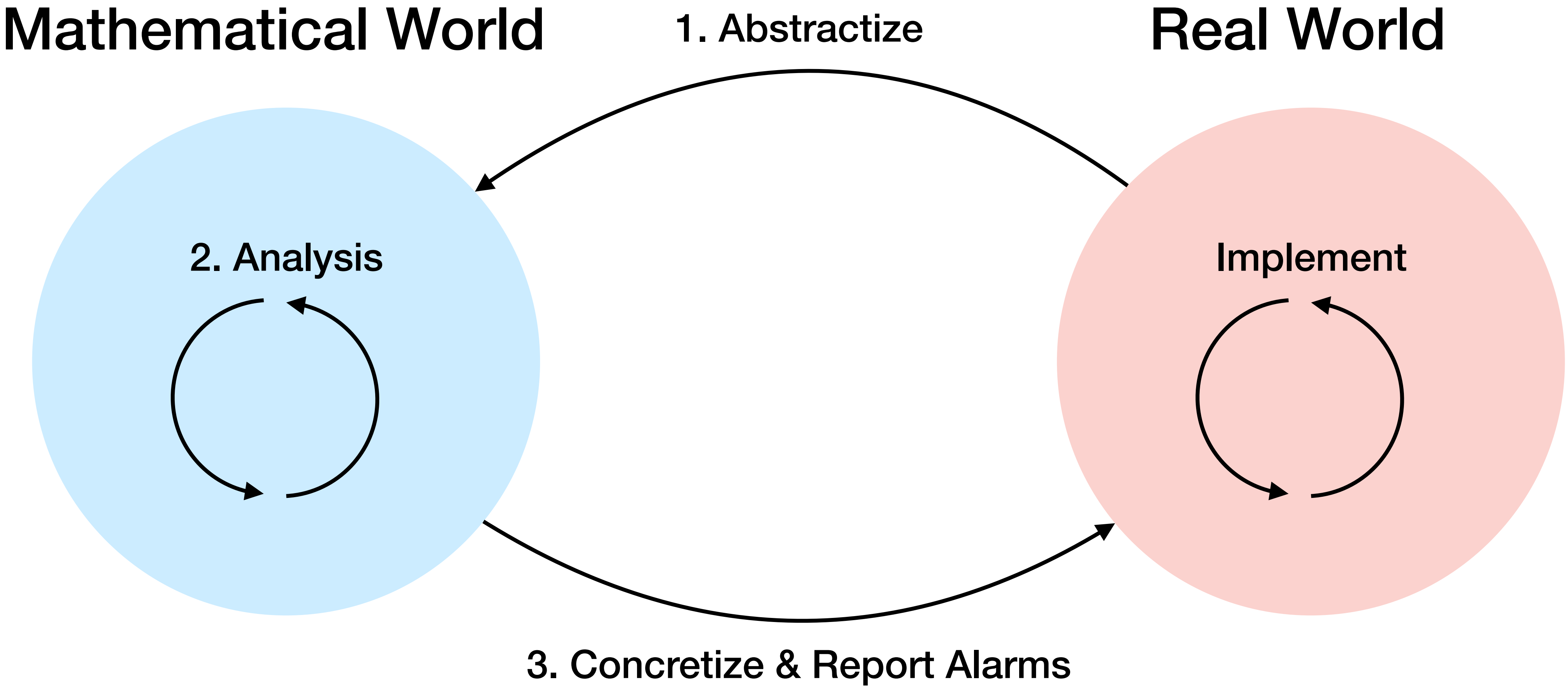
1. Born to be Googler … ?



Why Async Frameworks Matters?

2. Software Analysis, Checking errors before execution

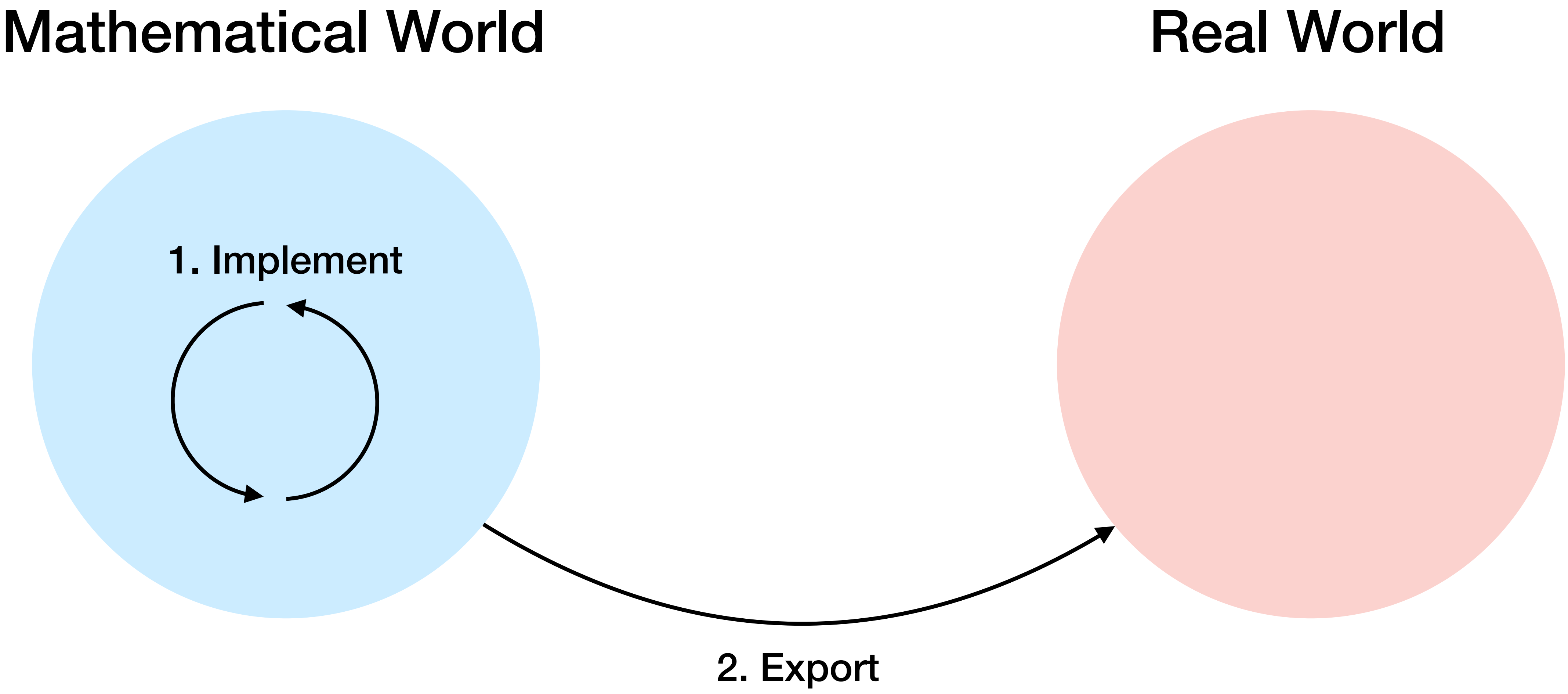
Static Analysis, Abstract Interpretation, Sparrow, FB Infer



Why Async Frameworks Matters?

3. Software Verification, Implementing programs with robust mathematical basis

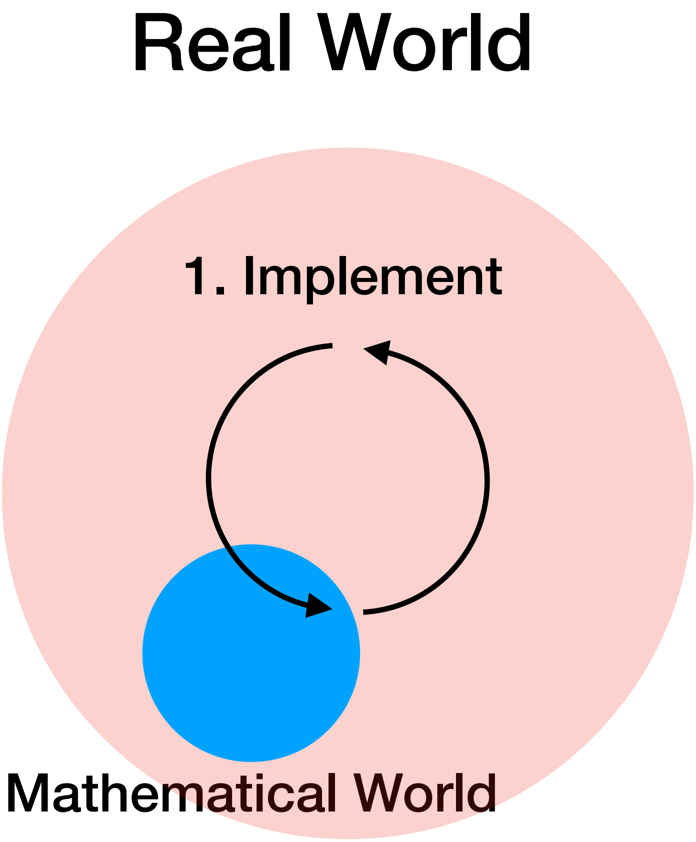
Coq, Machine-checked Proofs, Type Inference, Robust Type Systems, ...



Why Async Frameworks Matters?

4. Frameworks, Using safeness & productivity of frameworks

Asynchronous Frameworks, Reactive Programming, …



INDEX

1. Preliminary of Asynchronous Programming

Preliminary
Asynchronous Programming
When do we have to use Async?
Why Async Frameworks Matters?

2. Async Frameworks Details

Code Scalability
Reactive Programming (RxPY)
Why Reactive Programming Matters?

INDEX

1. Preliminary of Asynchronous Programming

Preliminary
Asynchronous Programming
When do we have to use Async?
Why Async Frameworks Matters?

2. Async Frameworks Details

Code Scalability
Reactive Programming (RxPY)
Why Reactive Programming Matters?

Code Scalability

Everything must be compositional

Code Scalability

Remind

Asynchronous Frameworks (tornado, asyncio)

```
data = yield tornado.iostream.read_until(‘\r\n’)
```

much more compositional

non-blocking I/O

```
while True:
    try:
        data = socket.recv(buf_size)
    except socket.error as e:
        if e.args[0] in _ERRNO_WOULDBLOCK:
            # DO SOMETHING ELSE
```

Code Scalability

Asynchronous Frameworks (tornado, asyncio)

```
data = yield tornado.iostream.read_until(‘\r\n’)
```

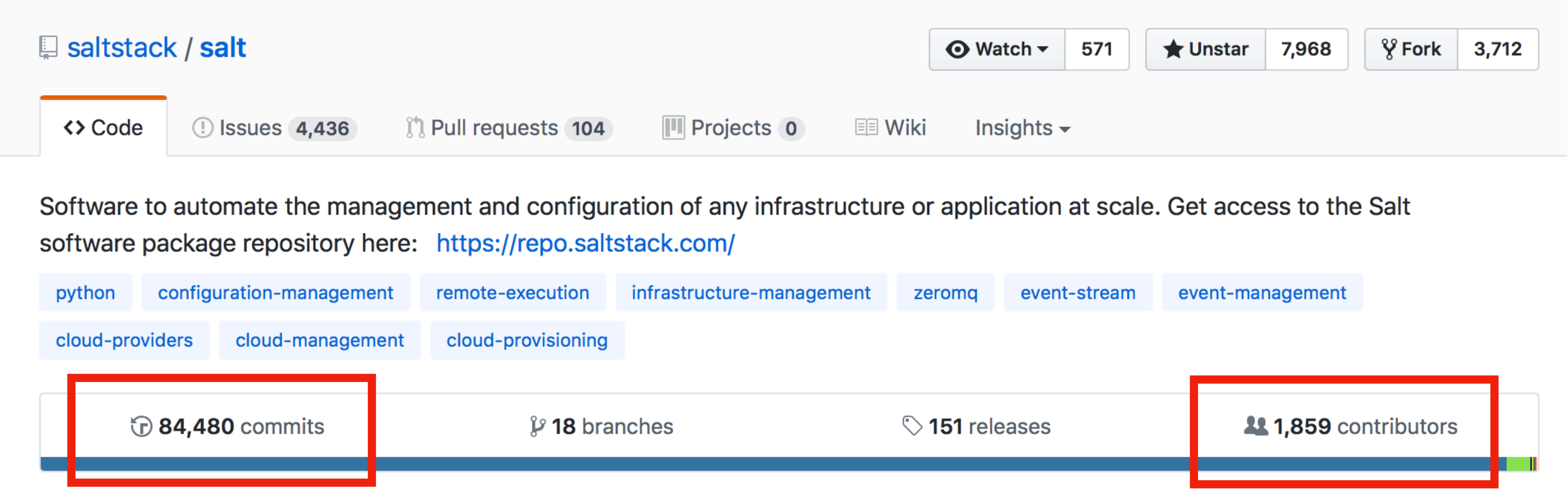
Is it enough?

NO!

Not enough for us

Code Scalability

Not enough for us



saltstack / salt

Watch 571 | Unstar 7,968 | Fork 3,712

<> Code | Issues 4,436 | Pull requests 104 | Projects 0 | Wiki | Insights

Software to automate the management and configuration of any infrastructure or application at scale. Get access to the Salt software package repository here: https://repo.saltstack.com/

python | configuration-management | remote-execution | infrastructure-management | zeromq | event-stream | event-management | cloud-providers | cloud-management | cloud-provisioning

84,480 commits | 18 branches | 151 releases | 1,859 contributors

Code Scalability

Comparison between (tornado, asyncio) and RxPY

Multiple Async HTTP Calls
Multiple Async HTTP Calls, Take Fastest Response
Sleep between Jobs & Global Timeout

Code Scalability

Multiple Async HTTP Calls, Torando, asyncio

```
res1 = yield service1_api_call()
res2 = yield service2_api_call()
res3 = yield service3_api_call()
response = res1 + res2 + res3
```

Must be refactored

Code Scalability

Multiple Async HTTP Calls, Torando, asyncio

```
futures = [service1_api_call(),
           service2_api_call(),
           service3_api_call()]

response = []
for future in futures:
    response.append((yield future))
```

Code Scalability

Multiple Async HTTP Calls, RxPY

```
Observable.merge(service1_api_call(),
                 service2_api_call(),
                 service3_api_call())
          .map(lambda response: ...)
```

Code Scalability

Multiple Async HTTP Calls, Take Fastest Response, Torando, asyncio

```
futures = [server1_api_call(),
           server2_api_call(),
           server3_api_call()]

response = None
for future in futures:
    response = yield future
    break
```

...?

Must be refactored

Code Scalability

Multiple Async HTTP Calls, Take Fastest Response, Torando, asyncio

```python
class Any(Future):
    def __init__(self, futures):
        super(Any, self).__init__()
        for future in futures:
            future.add_done_callback(self.done_callback)

    def done_callback(self, future):
        if not self.done():
            self.set_result(future)

futures = Any(system1_api_call(),
              system2_api_call(),
              system3_api_call())
response = yield futures
```

JUGGLING FUTURES

https://github.com/saltstack/salt/blob/b7cd30d3ee919fcf3f03a1afe349fb68b357cd99/salt/netapi/rest_tornado/saltnado.py#L269-L281

Code Scalability

Multiple Async HTTP Calls, Take Fastest Response, RxPY

```
Observable.merge(server1_api_call(),
                 server2_api_call(),
                 server3_api_call())
          .take(1)
          .map(lambda response: ...)
```

Code Scalability

More Complex Examples, RxPY

```
res1 = Observable.merge(service1_server1_api_call(),
                        service1_server2_api_call(),
                        service1_server3_api_call())
                 .take(1)

res2 = Observable.merge(service2_server1_api_call(),
                        service2_server2_api_call(),
                        service2_server3_api_call())
                 .take(1)

Observable.merge(res1, res2)
          .map(lambda response: ...)
```

Code Scalability

More Complex Examples, Torando, asyncio

…?

Code Scalability

Sleep between Jobs & Global Timeout, tornado, asyncio

```
elapsed_time = 0
for item in many_items:
    begin_time = time.time()
    yield gen.sleep(1000)
    yield insert_to_db(item)
    elapsed_time += time.time() - begin_time
    if elapsed_time > many_items.length * 1000 + MARGIN:
        raise Exception(...)
```

Code Scalability

Sleep between Jobs & Global Timeout, RxPY

```
Observable.from(many_items)
          .zip(Observable.interval(1000), lambda (data, interval): data)
          .flat_map(insert_to_db)
          .timeout(many_items.length * 1000 + MARGIN)
          .subscribe(success, exception, completion)
```

Code Scalability

Reactive Programming (RxPY)

much more compositional

Asynchronous Frameworks (tornado, asyncio)

```
data = yield tornado.iostream.read_until(‘\r\n’)
```

much more compositional

non-blocking I/O

```
while True:
    try:
        data = socket.recv(buf_size)
    except socket.error as e:
        if e.args[0] in _ERRNO_WOULDBLOCK:
            # DO SOMETHING ELSE
```

INDEX

1. Preliminary of Asynchronous Programming

Preliminary
Asynchronous Programming
When do we have to use Async?
Why Async Frameworks Matters?

2. Async Frameworks Details

Code Scalability
Reactive Programming (RxPY)
Why Reactive Programming Matters?

Reactive Programming

Providing Concurrency by Scheduling Events

+

Providing Operators by Calling Functions

Reduce Complexity using functional programming patterns, disciplines

Reactive Programming

Rx Operators

Observable<Data>

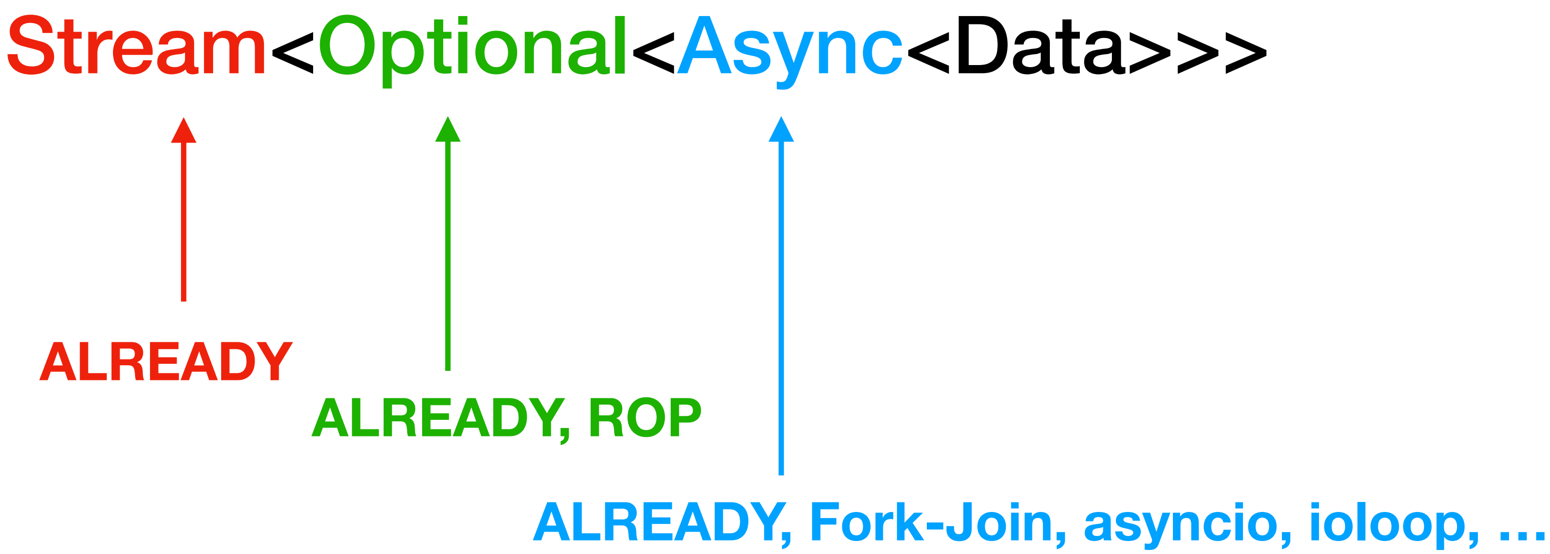
+

Operators

timer, defer, interval, repeat, just,
map, flat_map, buffer,
filter, debounce, last, skip,
zip, merge,
catch_exception, retry,
delay, timeout,
reduce, average, max, min, count,
…

Reactive Programming

Essence of Observable



Stream<Optional<Async<Data>>>

ALREADY

ALREADY, ROP

ALREADY, Fork-Join, asyncio, ioloop, …

Reactive Programming

Essence of Observable

Rx : Stream<Optional<Async<Data>>>
→ (Data → Stream<Optional<Async<Data>>>)
→ Stream<Optional<Async<Data>>>

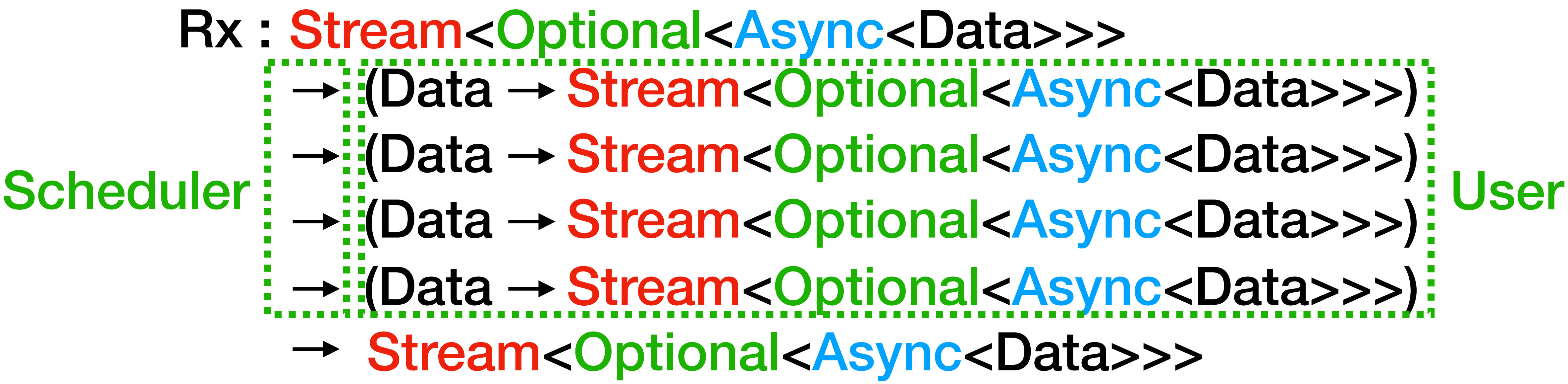
Reactive Programming

Essence of Observable

Rx : Stream<Optional<Async<Data>>>
→ (Data → Stream<Optional<Async<Data>>>)
→ (Data → Stream<Optional<Async<Data>>>)
→ (Data → Stream<Optional<Async<Data>>>)
→ (Data → Stream<Optional<Async<Data>>>)
→ Stream<Optional<Async<Data>>>

Reactive Programming

Essence of Observable



Rx : Stream<Optional<Async<Data>>>
→ (Data → Stream<Optional<Async<Data>>>)
→ (Data → Stream<Optional<Async<Data>>>)
→ (Data → Stream<Optional<Async<Data>>>)
→ (Data → Stream<Optional<Async<Data>>>)
→ Stream<Optional<Async<Data>>>

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
          .filter(is_student)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

1. Stream (infinite data structure based on Co-induction)

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
          .filter(is_student)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

2. Some other obvious examples of compositionality

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
          .filter(is_student)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

2. Some other obvious examples of compositionality

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
          .filter(is_teacher)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

2. Some other obvious examples of compositionality

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
  ───────→.filter(super_super_complex_logic)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

3. Compositionality of Operators

```
 operator =  some_other_operator : Observable<a> -> * -> Observable<b>

feature_x = Observable.from(get_people_by_async_task)
                      .filter(non_empty)
                      .filter(is_student)
                      .map(extract_student_id)
                      .flat_map(get_person_detail_by_async_task)
                      .map(extract_name)
                      .some_other_operator(. . .)


feature_y = Observable.from(get_people_by_async_task)
                      . (. . .)
                      .flat_map(feature_x)
                      . (. . .)
                      .some_other_operator(. . .)
                      .subscription(success, error, complete)
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

4. Functional Programming Patterns (ROP), First Class Effect

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
          .filter(is_student)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

https://fsharpforfunandprofit.com/rop/

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
          .filter(is_student)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

https://www.cl.cam.ac.uk/teaching/1415/L28/monads.pdf

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
          .filter(is_student)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

```
from : a -> Observable<a>
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
          .filter(is_student)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

```
  filter : Observable<a> -> (a -> bool) -> Observable<a>
     map : Observable<a> -> (a -> b) -> Observable<b>
flat_map : Observable<a> -> (a -> Observable<b>) -> Observable<b>
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

```
Observable.from(get_people_by_async_task)
          .filter(non_empty)
          .filter(is_student)
          .map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .map(extract_name)
          .subscription(success, error, complete)
```

```
operators : Observable<a> -> * -> Observable<b>    ≃ flat_map
```

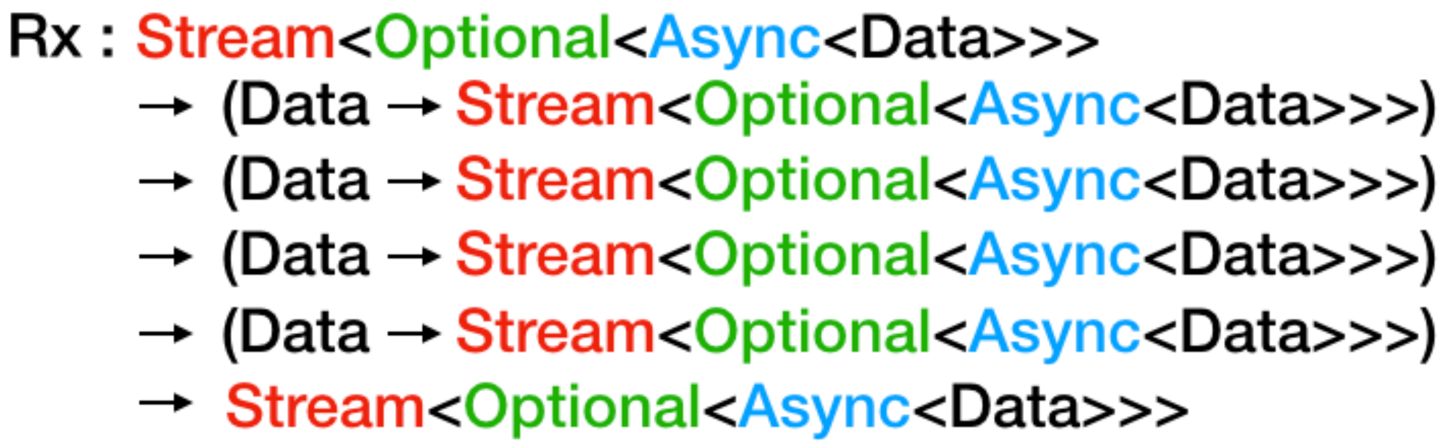
Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

```
Observable.return(get_people_by_async_task)
          .flat_map(filter_non_empty)
          .flat_map(filter_is_student)
          .flat_map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .flat_map(extract_name)
          .subscription(success, error, complete)
```

```
   return : a -> Observable<a>
 flat_map : Observable<a>
            -> (a -> Observable<b>)
            -> Observable<b>
```



```
Rx : Stream<Optional<Async<Data>>>
     → (Data → Stream<Optional<Async<Data>>>)
     → (Data → Stream<Optional<Async<Data>>>)
     → (Data → Stream<Optional<Async<Data>>>)
     → (Data → Stream<Optional<Async<Data>>>)
     → Stream<Optional<Async<Data>>>
```

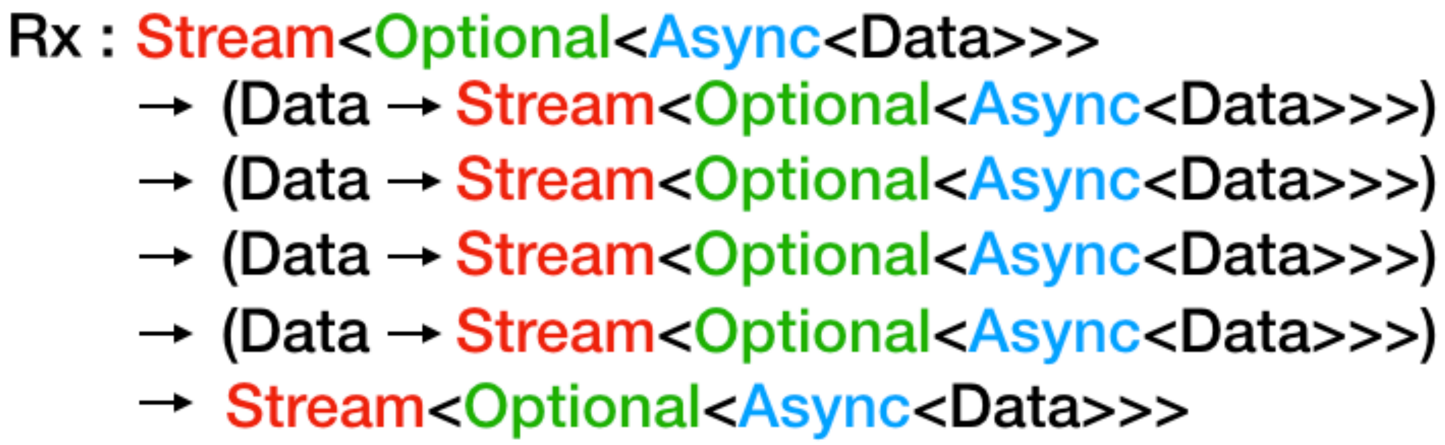
Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

```
Stream.return(Optional.return(Async.return(get_people_by_async_task)))
      .flat_map(Optional.flat_map(Async.flat_map(filter_non_empty)))
      .flat_map(Optional.flat_map(Async.flat_map(filter_is_student)))
      .flat_map(Optional.flat_map(Async.flat_map(extract_student_id)))
      .flat_map(Optional.flat_map(Async.flat_map(get_person_detail_by_async_task)))
      .flat_map(Optional.flat_map(Async.flat_map(extract_name)))
```

```
  return : a -> Stream<Optional<Async<<<a>>>
flat_map : Stream<Optional<Async<<<a>>>
           -> (a -> Stream<Optional<Async<<<b>>>)
           -> Stream<Optional<Async<<<b>>>
```



```
Rx : Stream<Optional<Async<Data>>>
     → (Data → Stream<Optional<Async<Data>>>)
     → (Data → Stream<Optional<Async<Data>>>)
     → (Data → Stream<Optional<Async<Data>>>)
     → (Data → Stream<Optional<Async<Data>>>)
     → Stream<Optional<Async<Data>>>
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

```
Observable.return(get_people_by_async_task)
          .flat_map(filter_non_empty)
          .flat_map(filter_is_student)
          .flat_map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .flat_map(extract_name)
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

In Scala

```
for {
  data <- get_people_by_async_task()
  data <- filter_non_empty(data)
  data <- filter_is_student(data)
  data <- extract_student_id(data)
  data <- get_person_detail_by_async_task(data)
  data <- extract_name(data)
} yield {
  . . .
}
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

In Haskell

```
do data <- get_people_by_async_task
   data <- filter_non_empty data
   data <- filter_is_student data
   data <- extract_student_id data
   data <- get_person_detail_by_async_task data
   data <- extract_name data
   data
. . .
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

In OCaml

```
let%lwt data = get_people_by_async_task in
let%lwt data = filter_non_empty data in
let%lwt data = filter_is_student data in
let%lwt data = extract_student_id data in
let%lwt data = get_person_detail_by_async_task data in
let%lwt data = extract_name data in
. . .
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

5. Abstracted Data Type based on Type Theory - First Class Effect

In Java

```
Observable.return(get_people_by_async_task)
          .flat_map(SomeObj::filter_non_empty)
          .flat_map(SomeObj::filter_is_student)
          .flat_map(SomeObj::extract_student_id)
          .flat_map(SomeObj::get_person_detail_by_async_task)
          .flat_map(SomeObj::extract_name)
```

Reactive Programming

Reduce Complexity using functional programming patterns, disciplines

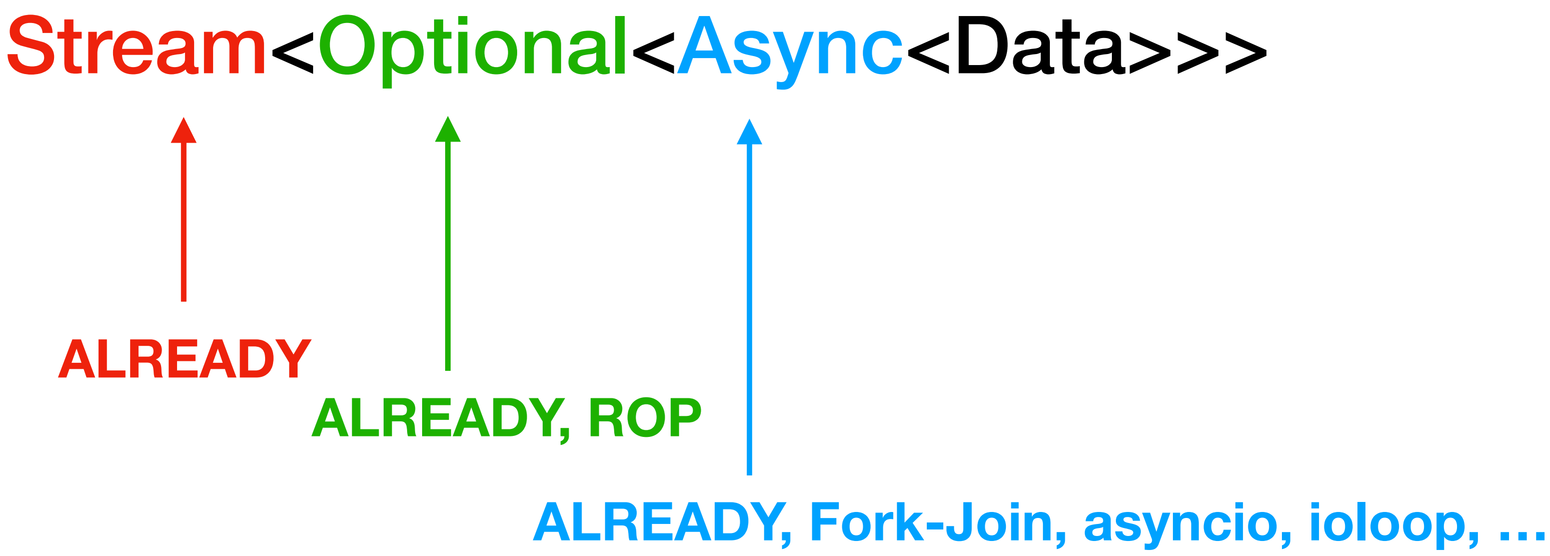
5. Abstracted Data Type based on Type Theory - First Class Effect

In Python

```
Observable.return(get_people_by_async_task)
          .flat_map(filter_non_empty)
          .flat_map(filter_is_student)
          .flat_map(extract_student_id)
          .flat_map(get_person_detail_by_async_task)
          .flat_map(extract_name)
```

Reactive Programming

Essence of Observable



Stream<Optional<Async<Data>>>

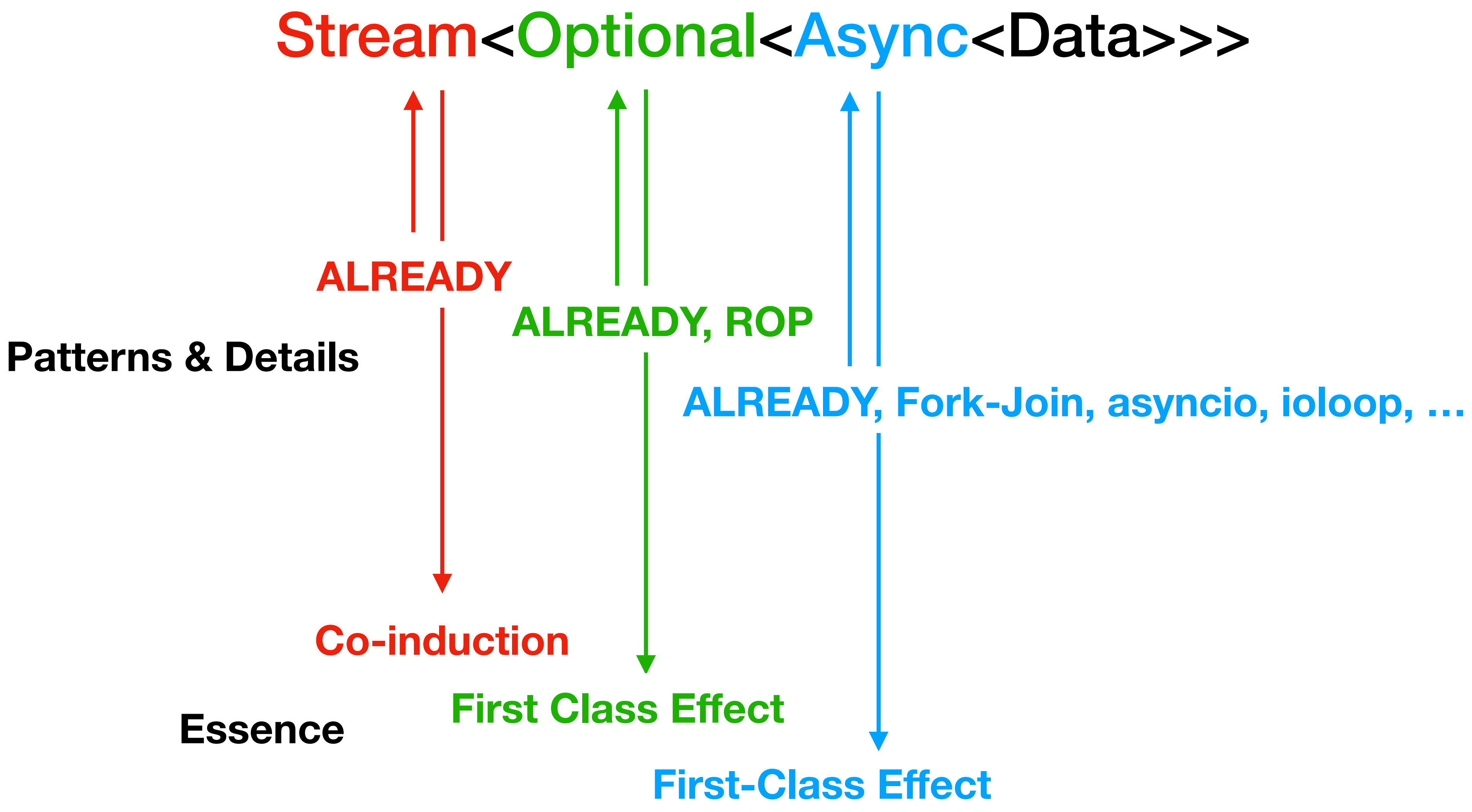
ALREADY

ALREADY, ROP

ALREADY, Fork-Join, asyncio, ioloop, …

Reactive Programming

Essence of Observable



Stream<Optional<Async<Data>>>

ALREADY

ALREADY, ROP

Patterns & Details

ALREADY, Fork-Join, asyncio, ioloop, …

Co-induction

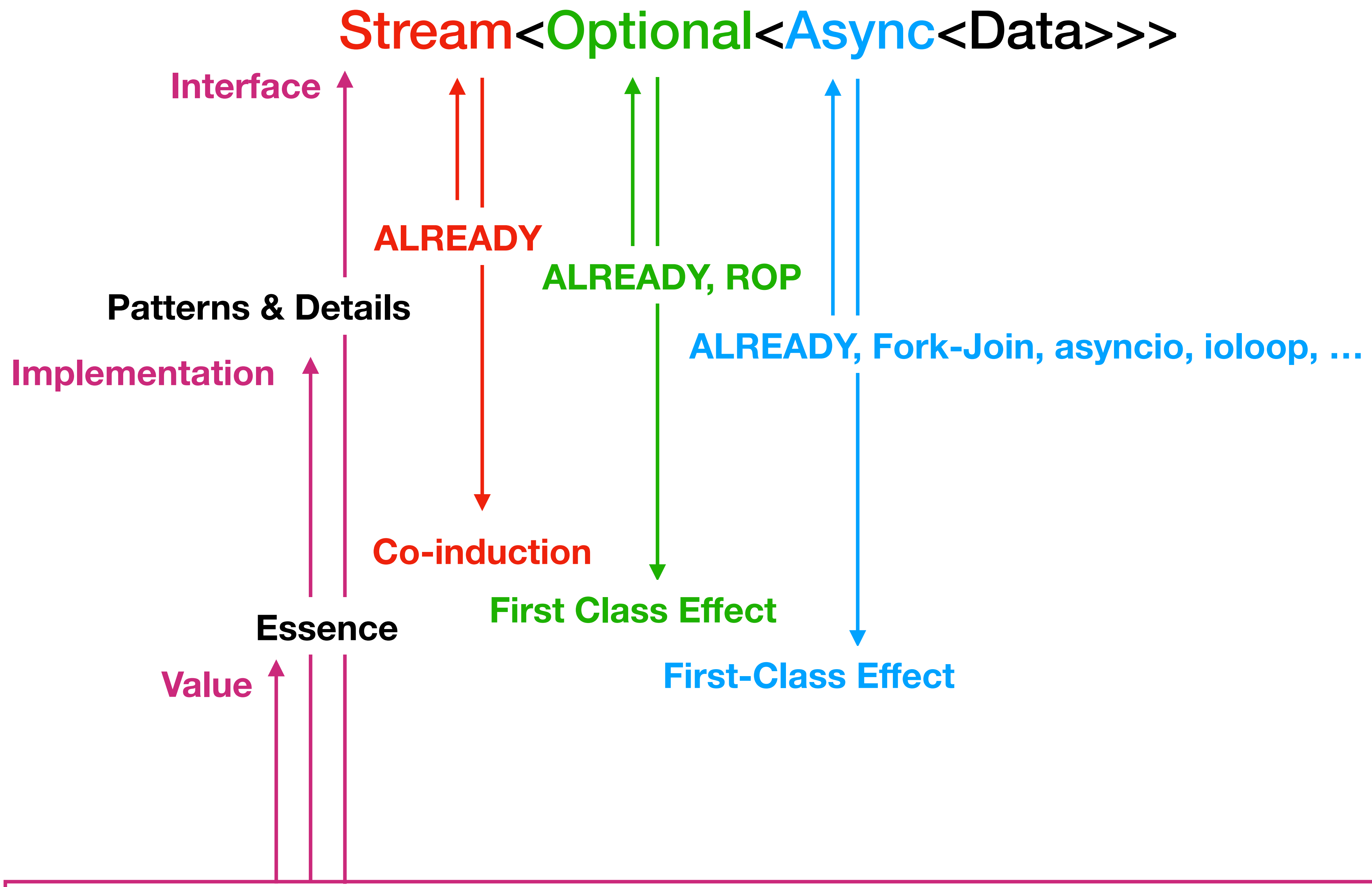
First Class Effect

Essence

First-Class Effect

Reactive Programming

Essence of Observable



Stream<Optional<Async<Data>>>

Interface

ALREADY

ALREADY, ROP

Patterns & Details

ALREADY, Fork-Join, asyncio, ioloop, …

Implementation

Co-induction

First Class Effect

Essence

First-Class Effect

Value

What we can obtain by following the principles can be explained based on mathematics.

INDEX

1. Preliminary of Asynchronous Programming

Preliminary
Asynchronous Programming
When do we have to use Async?
Why Async Frameworks Matters?

2. Async Frameworks Details

Code Scalability
Reactive Programming (RxPY)
Why Reactive Programming Matters?

Why Reactive Programming Matters?

4. Frameworks, Using safeness & productivity of frameworks

Asynchronous Frameworks, Reactive Programming, …

Interface

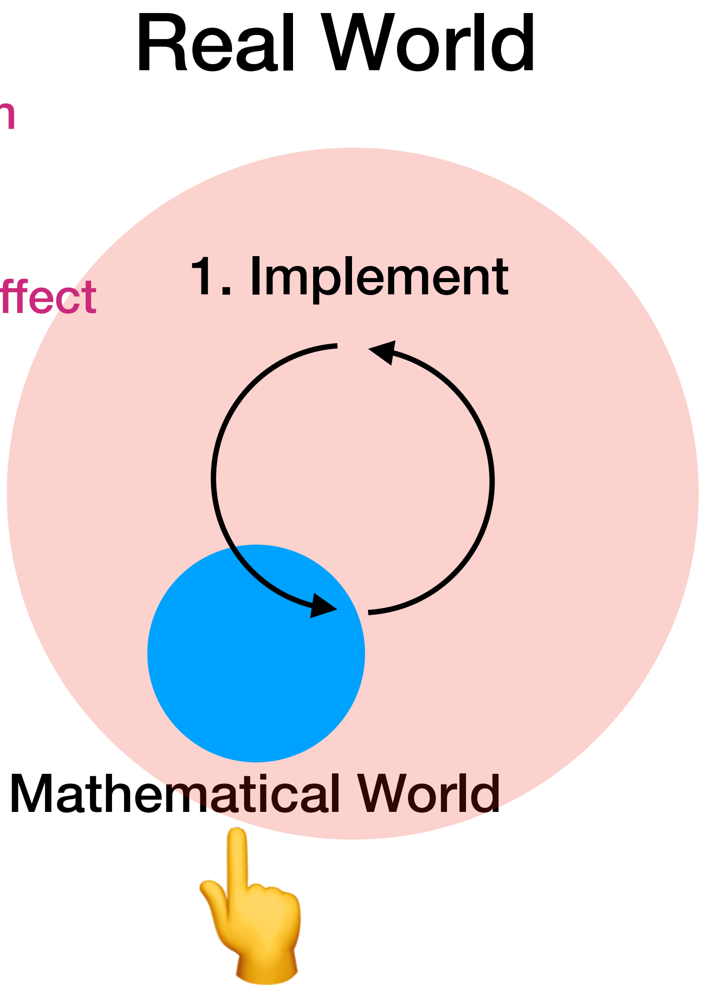
Syntax
Type Systems
Syntactic Sugars
Usage of Frameworks
Usage of Libraries
Programming Patterns
. . .

Implementation

Compiler
Type Checkers
Frameworks
Libraries
Implementation Details
Engineerings
. . .

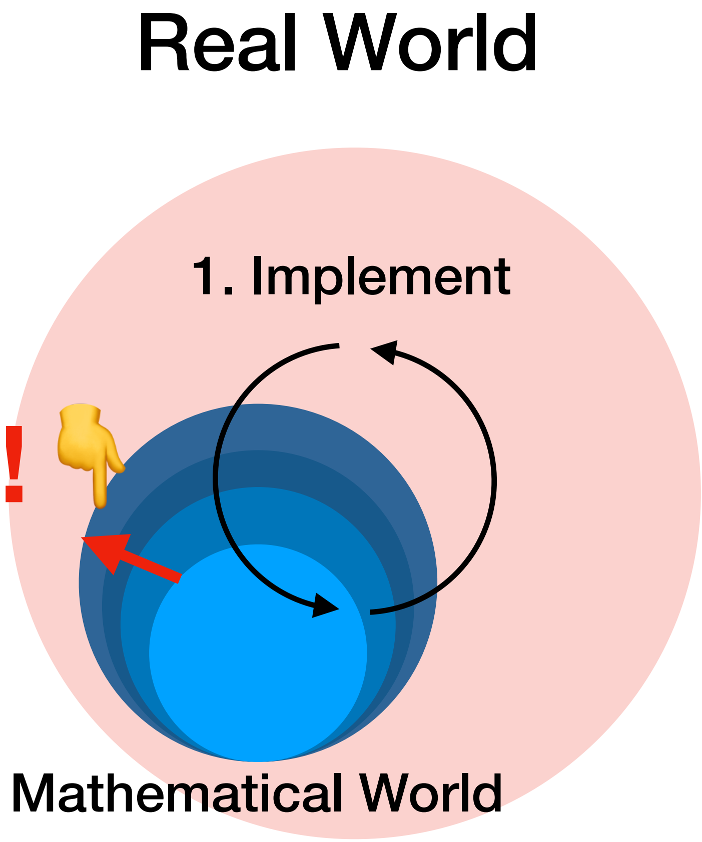
Essence

Curry-Howard Isomorphism
Type Theory
Category Theory
Monads, First Class Side Effect
Propositional Logics
. . .



What we can obtain by following the principles can be explained based on mathematics.

Why Reactive Programming Matters?



Real World

1. Implement

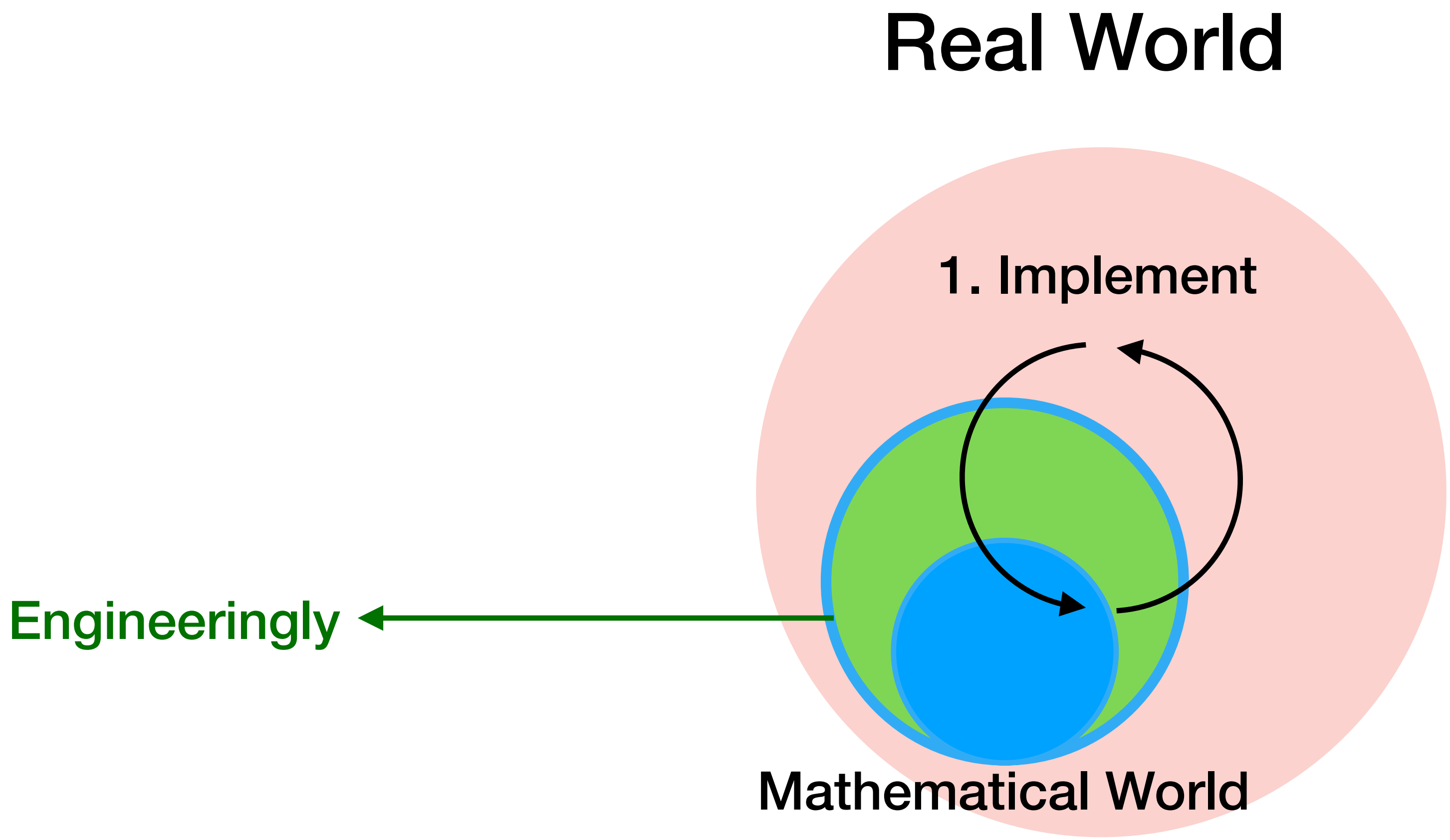
Interesting! 👇

Mathematical World

Why Reactive Programming Matters?

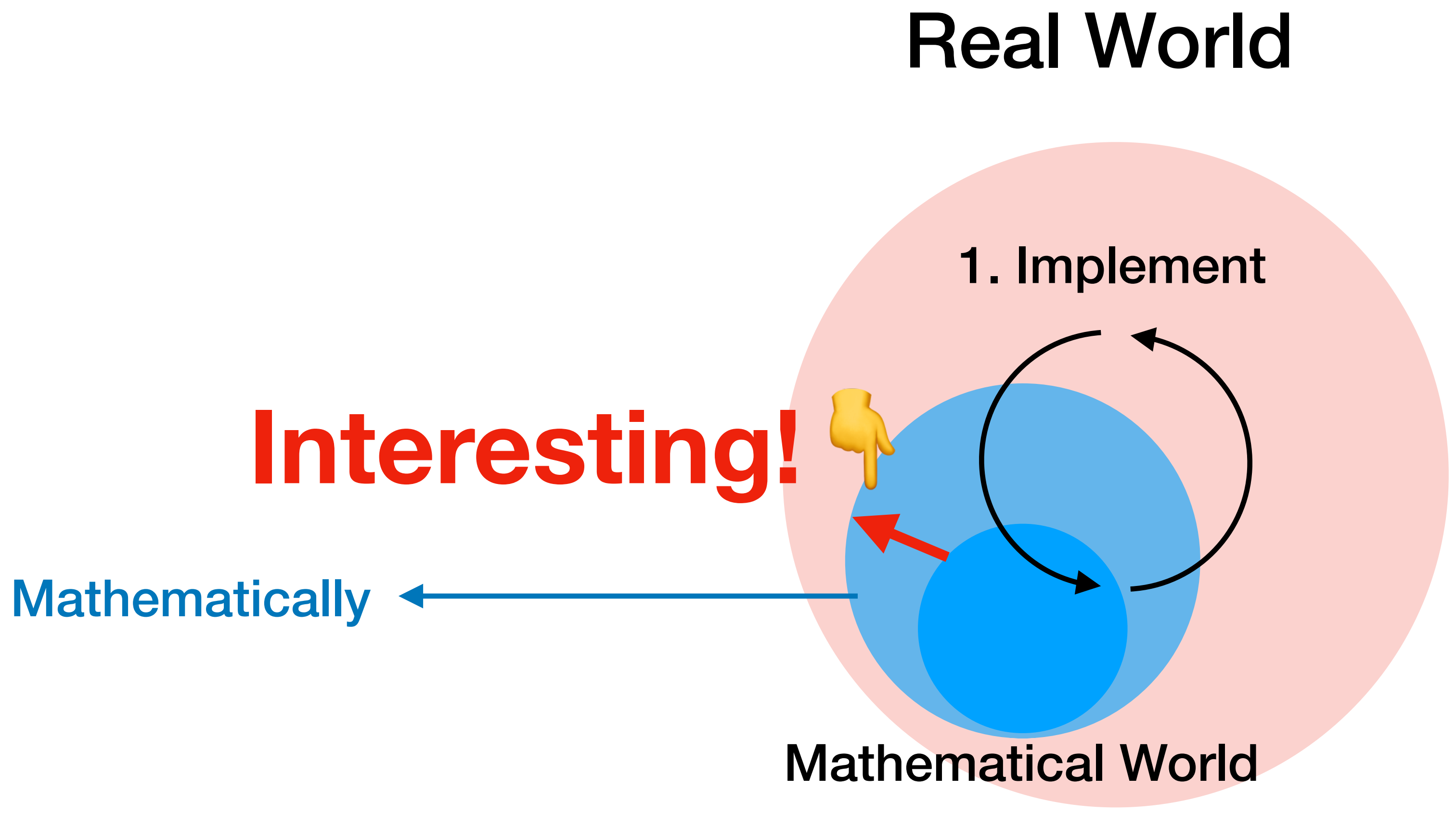
Just Use!

Dependency Injection, Flux, Redux (Redux-React, Revue, …), …



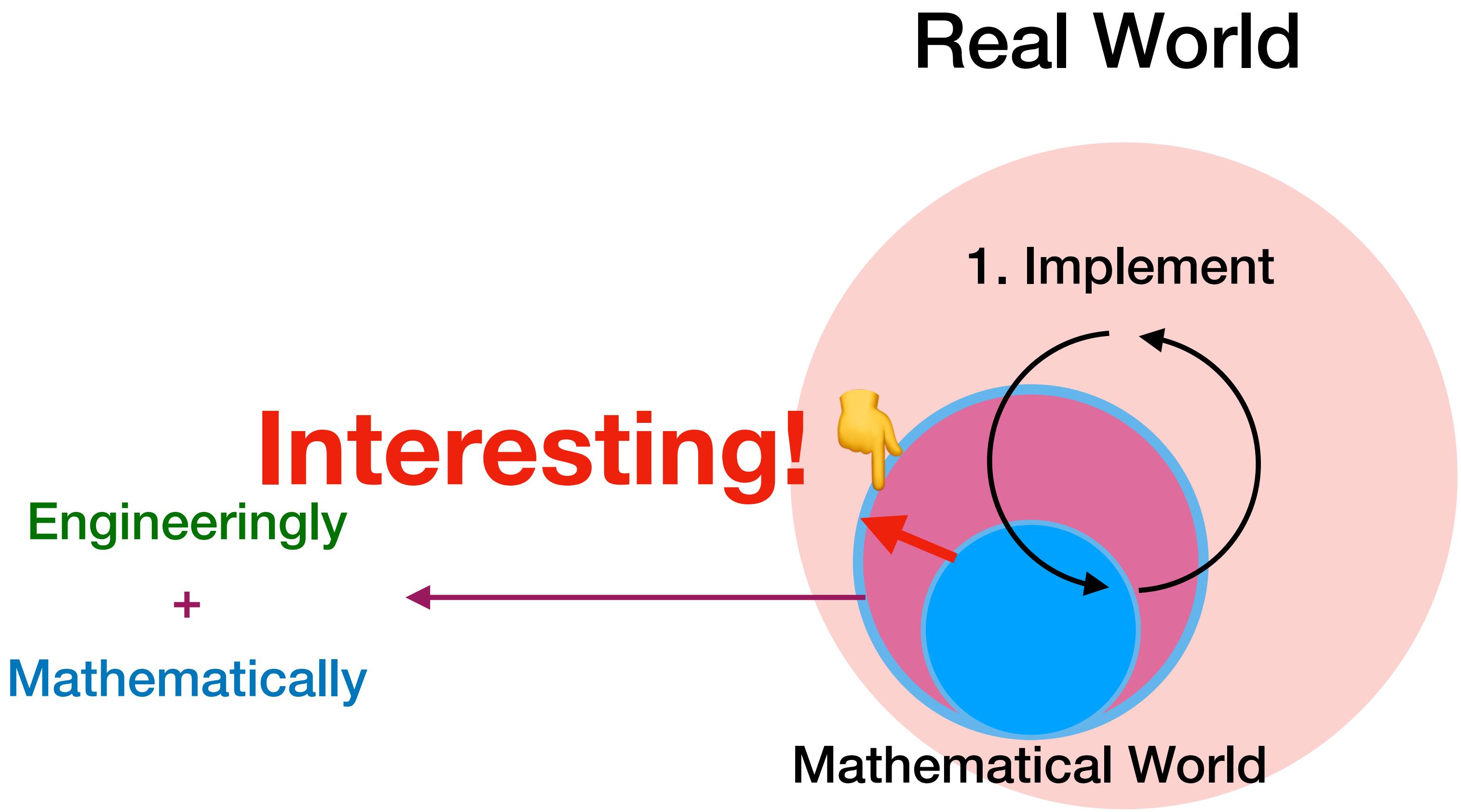
Why Reactive Programming Matters?

Advanced Type System (Scala DOT, …),
GADT, Dependent Types, Ur/Web, …



Why Reactive Programming Matters?

Reactive Programming (RxPY, RxJava, ...)



Real World

1. Implement

Interesting! 👇

Engineeringly

+

Mathematically

Mathematical World

SUMMARY IN 3 SLIDES

Code Scalability

Reactive Programming (RxPY)

much more compositional

Asynchronous Frameworks (tornado, asyncio)

```
data = yield tornado.iostream.read_until(‘\r\n’)
```

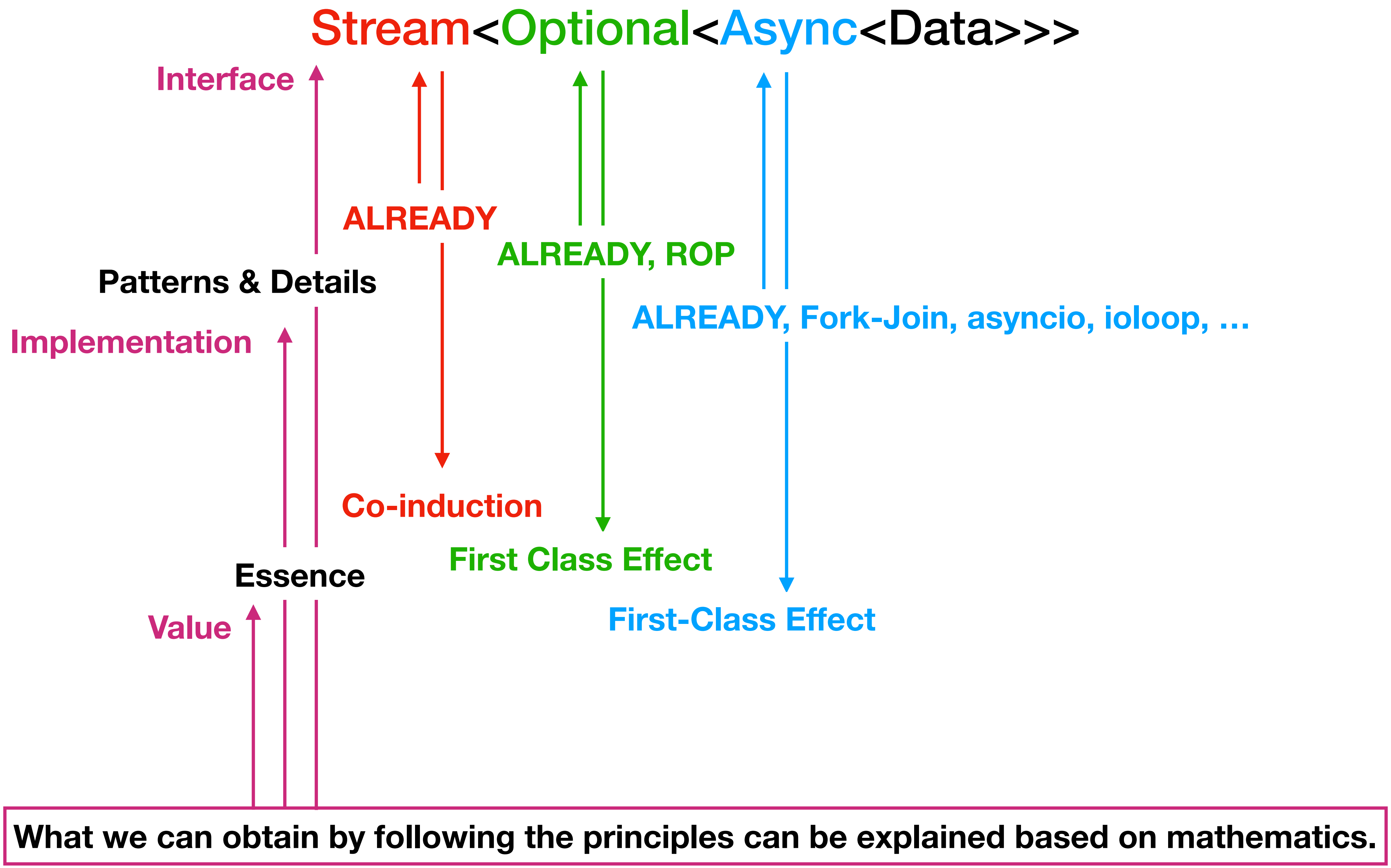
much more compositional

non-blocking I/O

```
while True:
    try:
        data = socket.recv(buf_size)
    except socket.error as e:
        if e.args[0] in _ERRNO_WOULDBLOCK:
            # DO SOMETHING ELSE
```

Reactive Programming

Essence of Observable



Stream<Optional<Async<Data>>>

Interface

ALREADY

ALREADY, ROP

Patterns & Details

ALREADY, Fork-Join, asyncio, ioloop, …

Implementation

Co-induction

First Class Effect

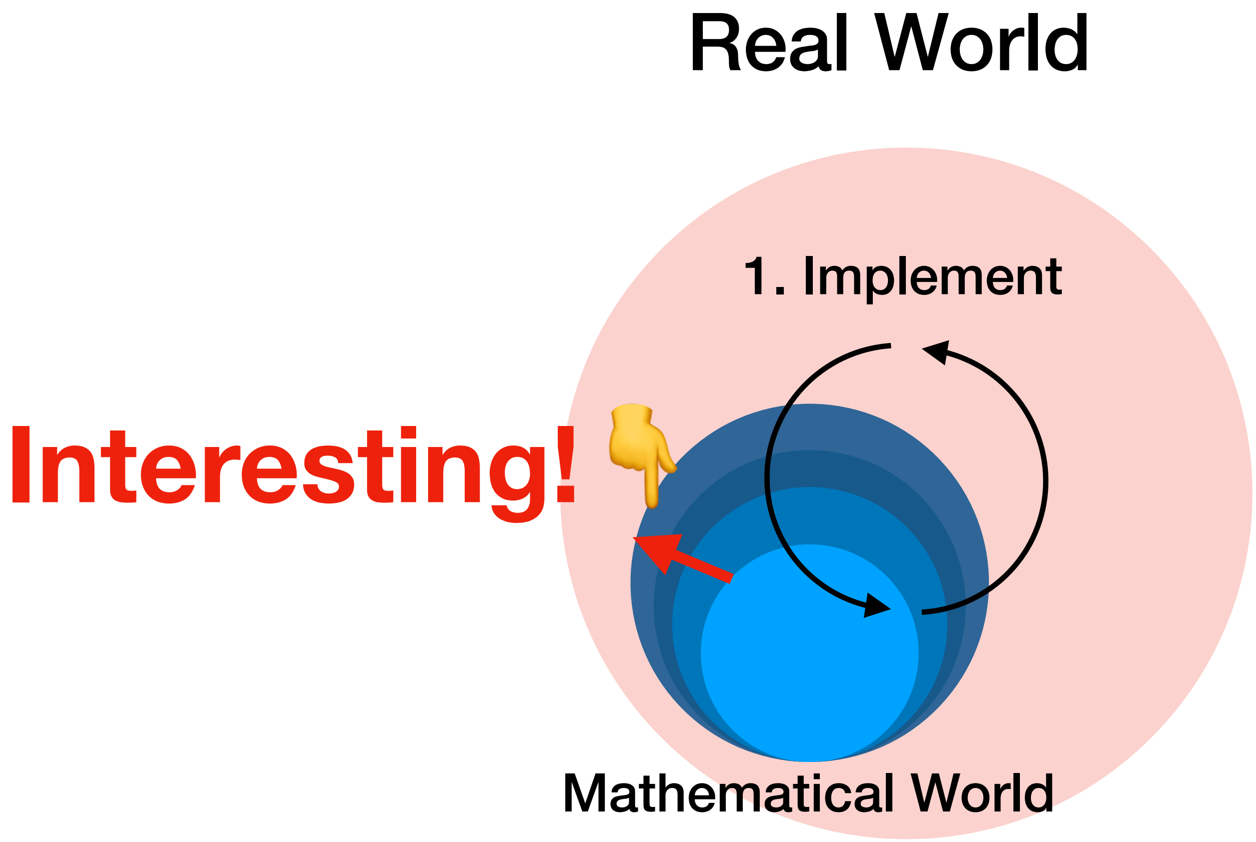
Essence

First-Class Effect

Value

What we can obtain by following the principles can be explained based on mathematics.

Why Reactive Programming Matters?



References & Further More

Salt Stack & RxPY

- Salt Stack : https://github.com/saltstack/salt
- RxPY : https://github.com/reactivex/rxpy

Asynchronous Programming & Reactive Programming

- The introduction to Reactive Programming you've been missing : https://gist.github.com/staltz/868e7e9bc2a7b8c1f754
- Your mouse is a database : http://queue.acm.org/detail.cfm?id=2169076
- Lwt: a Cooperative Thread Library : https://www.irif.fr/~Vouillon/publi/lwt.pdf
- Optimizing the Netflix API : https://medium.com/netflix-techblog/optimizing-the-netflix-api-5c9ac715cf19
- There is no Fork: an Abstraction for Efficient, Concurrent, and Concise Data Access : https://research.fb.com/publications/there-is-no-fork-an-abstraction-for-efficient-concurrent-and-concise-data-access/

Advanced Materials

- Functional Program Design in Scala : https://www.coursera.org/learn/progfun2
- Advanced Functional Programming : https://www.cl.cam.ac.uk/teaching/1415/L28/materials.html
- "Mostly functional" programming does not work : http://queue.acm.org/detail.cfm?ref=rss&id=2611829
- Railway Oriented Programming : https://fsharpforfunandprofit.com/rop/
- First-Class Effect : https://www.cl.cam.ac.uk/teaching/1415/L28/monads.pdf
- Moving fast with software verification : https://research.fb.com/wp-content/uploads/2016/11/publication00124_download0001.pdf?

NHN ENTERTAINMENT

We Are Hiring

👉 플랫폼 개발자 👈

Q & A

kstreee@gmail.com