

# Reaktywne aplikacje od podstaw

Marek Walkowiak

JUGademy 04.03.2020

**allegro**

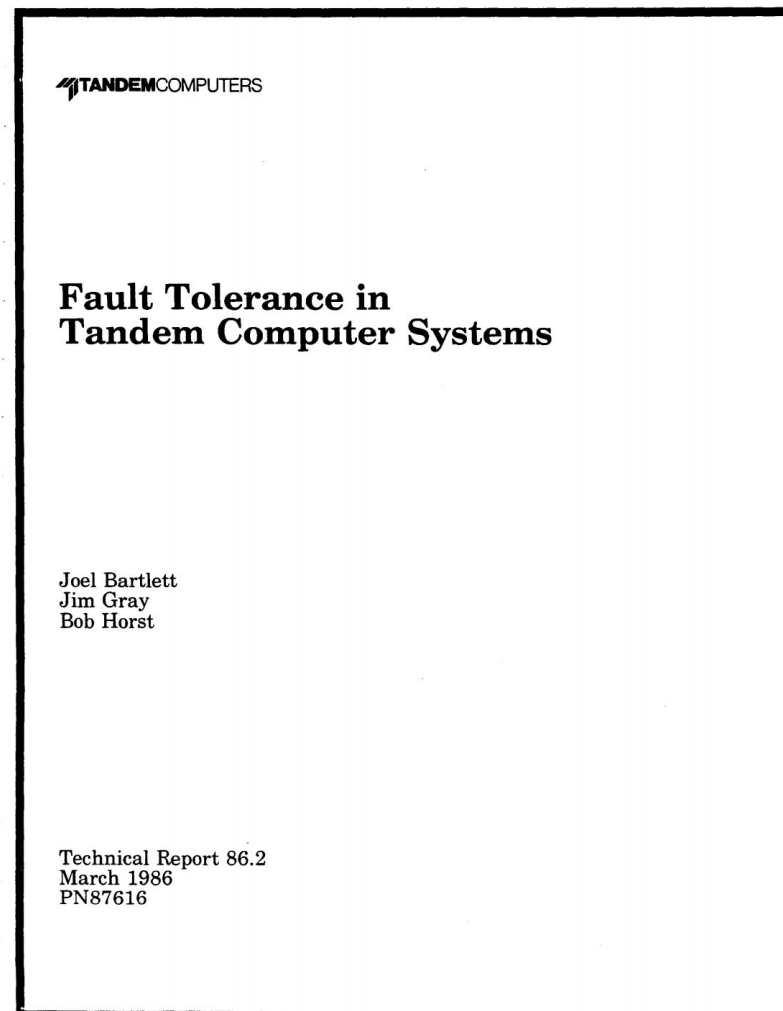
# Agenda

- Po co i dlaczego, czyli rys historyczny
- Reaktywność - o co chodzi?
- Programowanie reaktywne
- Reactive Extensions (Rx\*)
- Aplikacje reaktywne w praktyce

Po co i dlaczego, czyli krótki rys historyczny

# 1970s - 2000s

Różne mądre publikacje i seminaria



Making reliable  
distributed systems  
in the presence of  
software errors

Final version (with corrections) – last update 20 November 2003

Joe Armstrong

A Dissertation submitted to  
the Royal Institute of Technology  
in partial fulfilment of the requirements for  
the degree of Doctor of Technology  
The Royal Institute of Technology  
Stockholm, Sweden

December 2003

Department of Microelectronics and Information Technology



2009

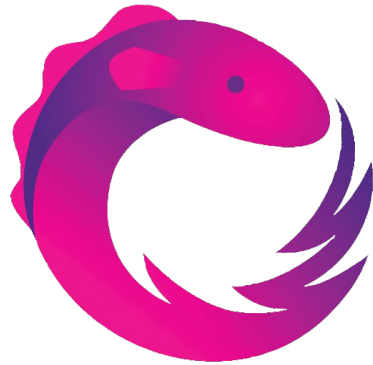
Pierwsza wersja Rx.NET



Erik Meijer

Współtwórca ReactiveX, Rx.NET, RxJava i wielu innych





## 2012

Rx.NET staje się Open Source i ląduje na Githubie. Zaczynają powstawać inne **Reactive Extensions**.

## 2012 - 2015

Reaktywny boom, powstają **RxJava** (Netflix), RxJS i wiele innych. Spisane zostaje **Reactive Manifesto**.



## RxJava

## 2015

Wspólna praca Netflix, Pivotal, Lightbend, Twitter i innych owocuje powstaniem standardu **Reactive Streams** i pierwszych implementacji (Project Reactor).

## 2015 - 2020+

Powstaje **Webflux**, większość języków ma już swoje Rx. PR jest fundamentem wielu technologii frontendowych, mobilnych (**Android**) i serwerowych.



VERT.x



# The Reactive Manifesto

*Published on September 16 2014. (v2.0)*

Organisations working in disparate domains are independently discovering patterns for building software that look the same. These systems are more robust, more resilient, more flexible and better positioned to meet modern demands.

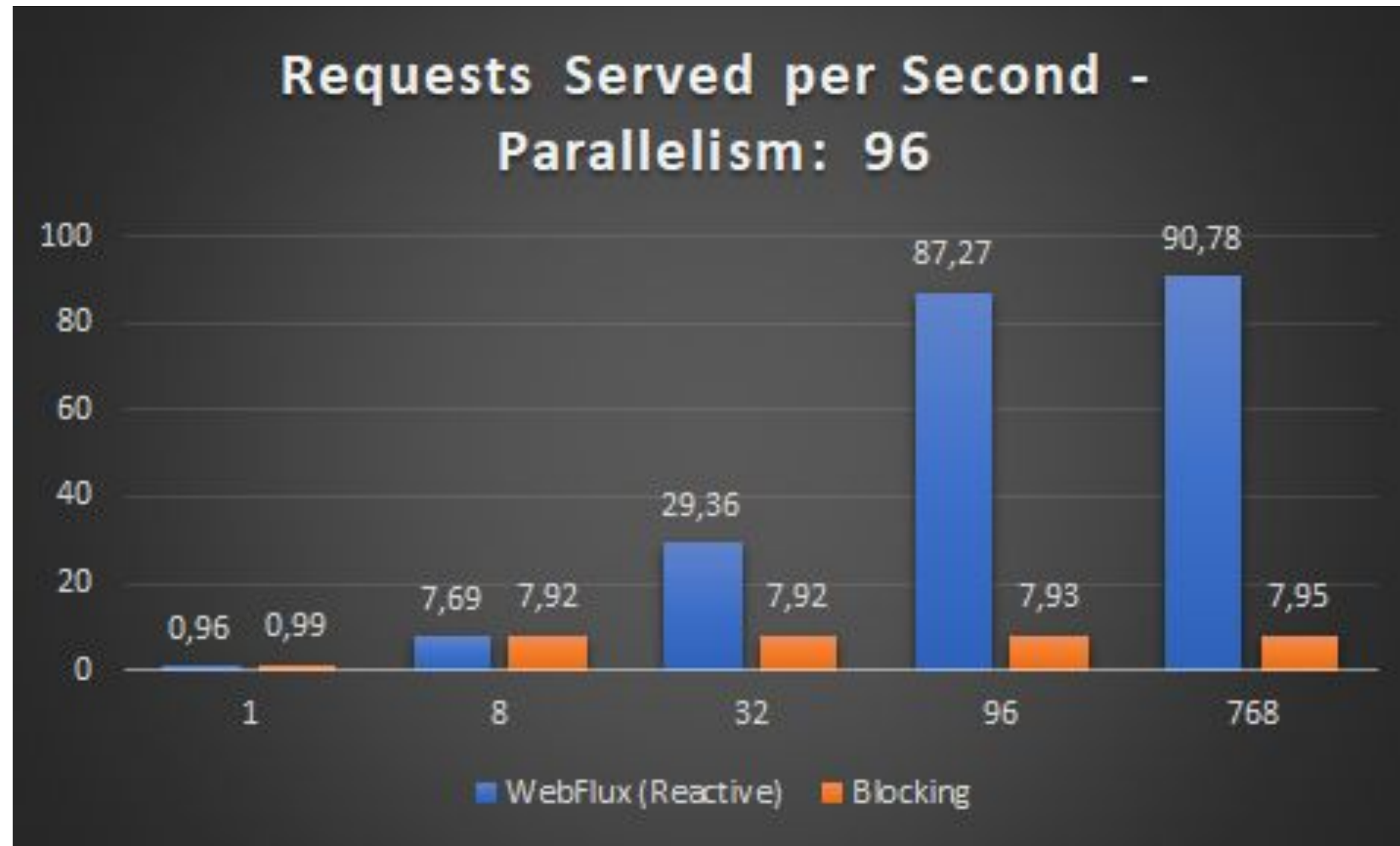
These changes are happening because application requirements have changed dramatically in recent years. Only a few years ago a large application had tens of servers, seconds of response time, hours of offline maintenance and gigabytes of data. Today applications are deployed on everything from mobile devices to cloud-based clusters running thousands of multi-core processors. Users expect millisecond response times and 100% uptime. Data is measured in Petabytes. Today's demands are simply not met by yesterday's software architectures.

We believe that a coherent approach to systems architecture is needed, and we believe that all necessary aspects are already recognised individually: we want systems that are Responsive, Resilient, Elastic and Message Driven. We call these Reactive Systems.

Systems built as Reactive Systems are more flexible, loosely-coupled and scalable. This makes them easier to develop and amenable to change. They are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster. Reactive Systems are highly responsive, giving users effective interactive feedback.



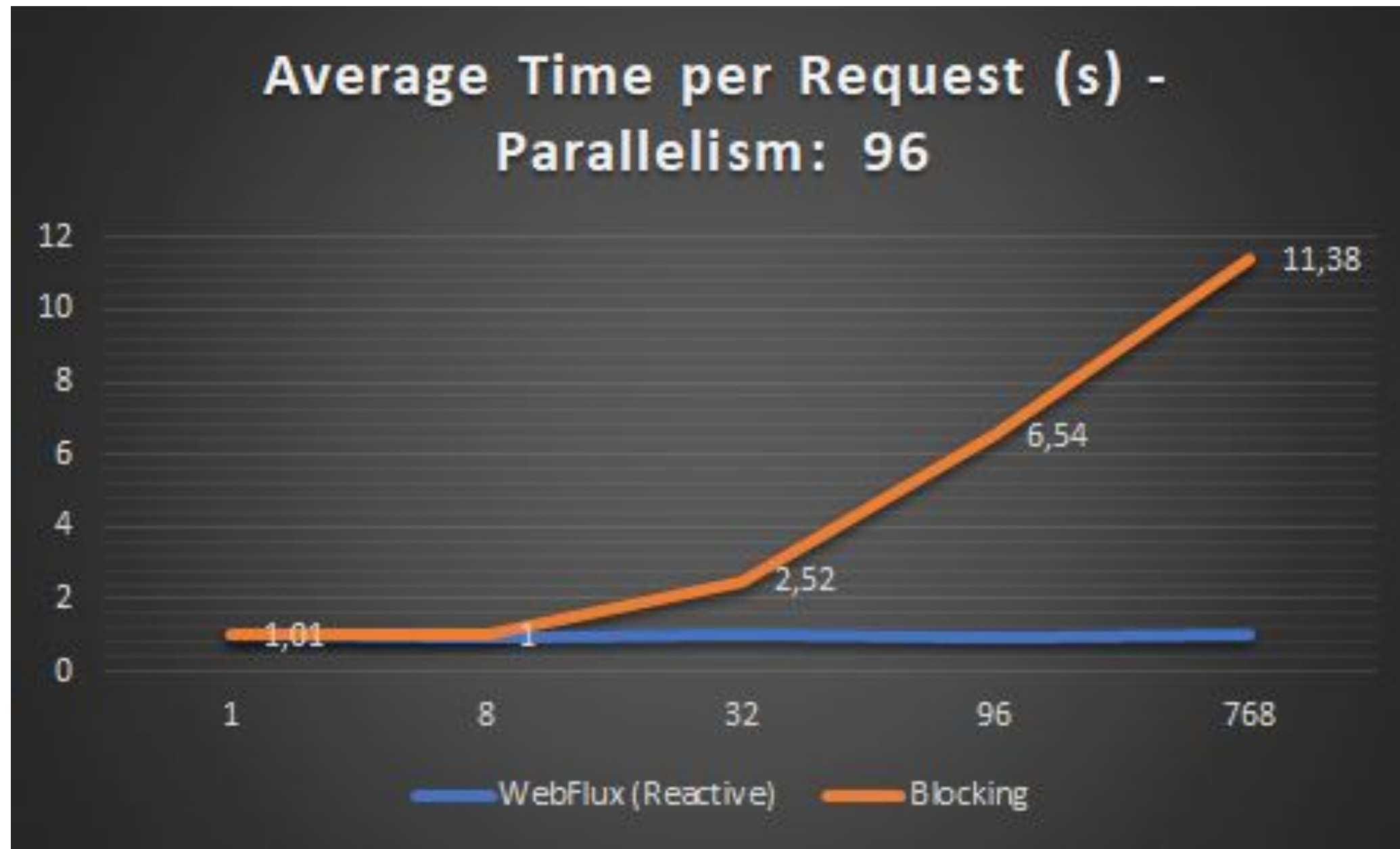
Po co?



**768** równoległych połączeń na **96** wątkach

Więcej = lepiej

Po co?



**768** równoległych połączeń na **96** wątkach

Mniej = lepiej

Reaktywność - o co chodzi?

Stary koncept...

Price	VAT	Service fee	Total
200	46	20	266

Stary koncept...

Price	VAT	Service fee	Total
200	46	20	266

... nowe zastosowania

Unit price:

100

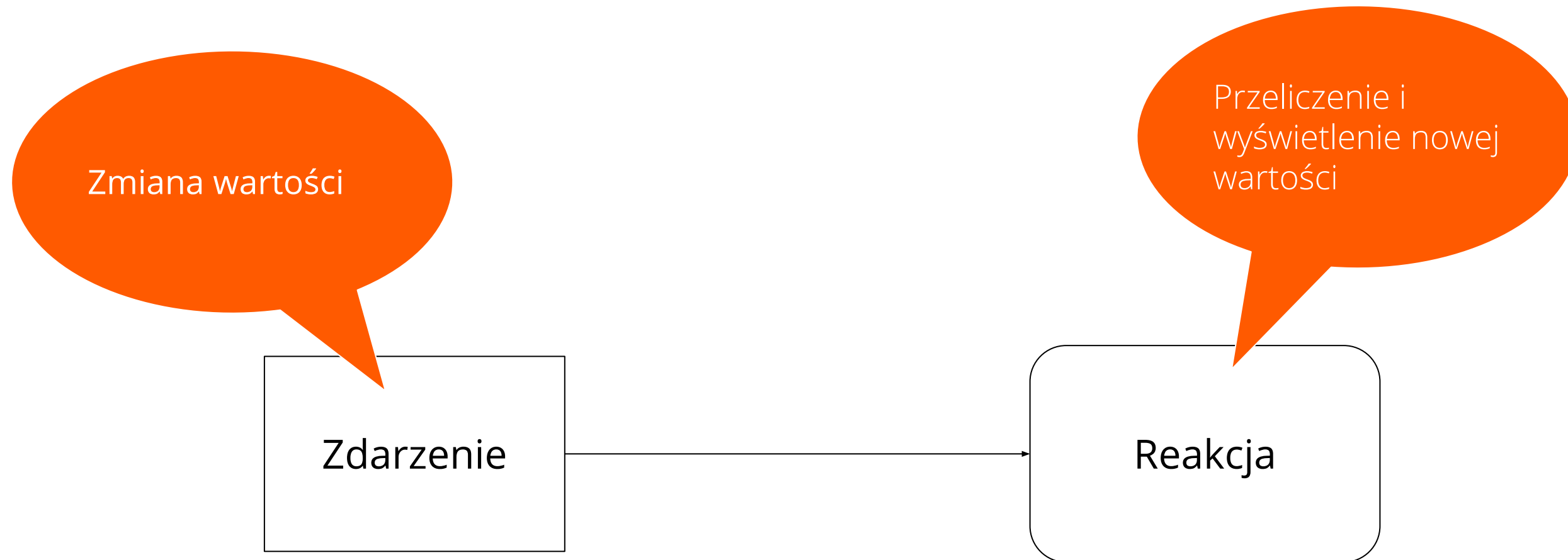
VAT 23%:	23 PLN
Service fee 10%:	10 PLN
<b>Total:</b>	<b>133 PLN</b>

# Reaktywność jako reakcja na zdarzenia

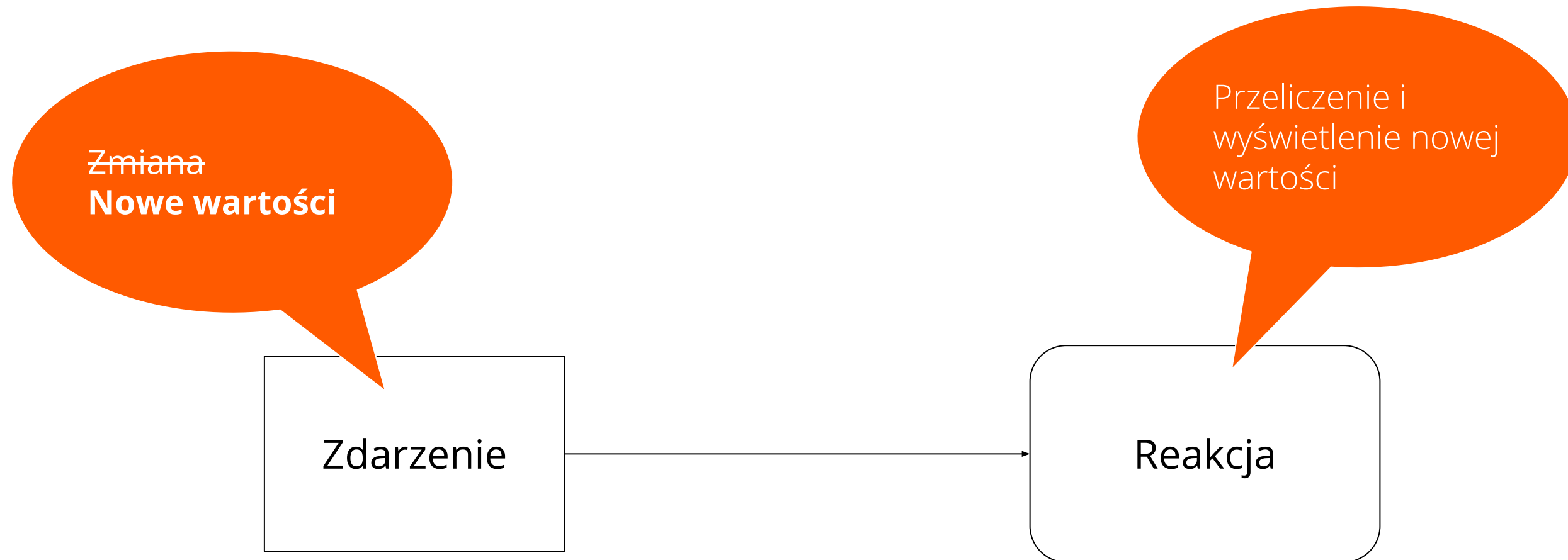




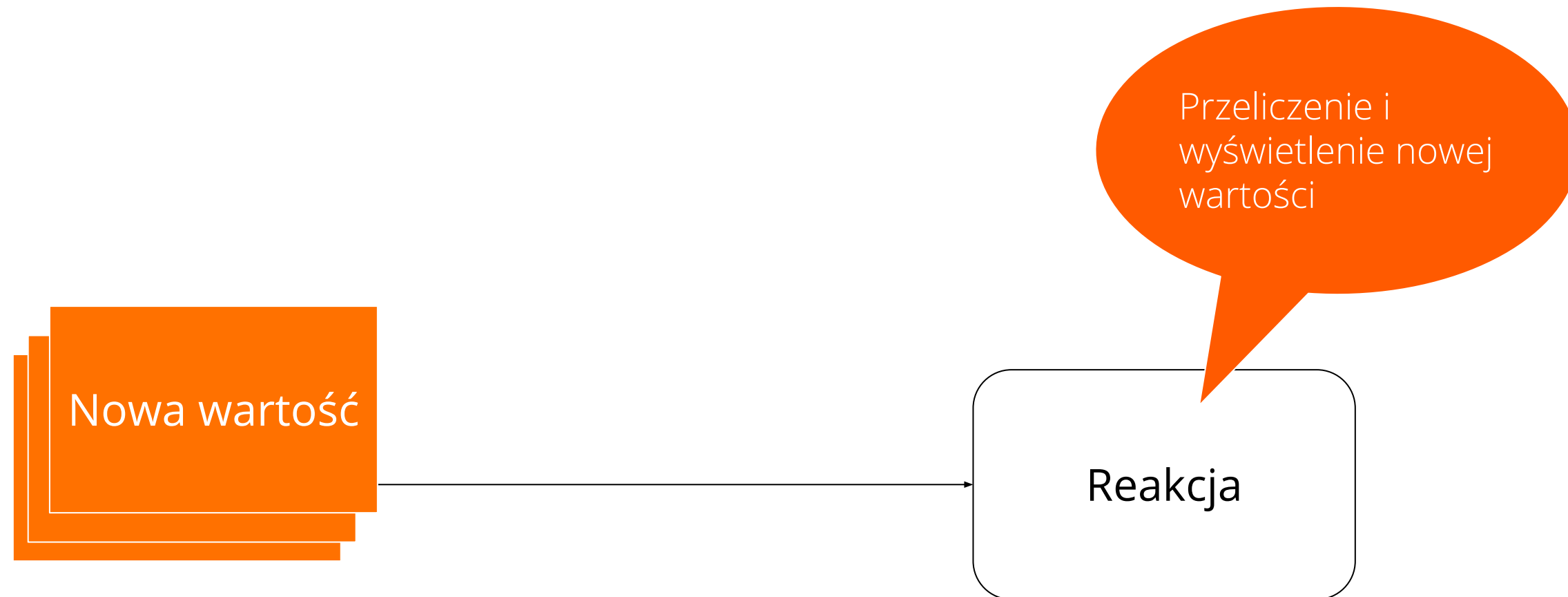
# Reaktywność jako reakcja na zdarzenia



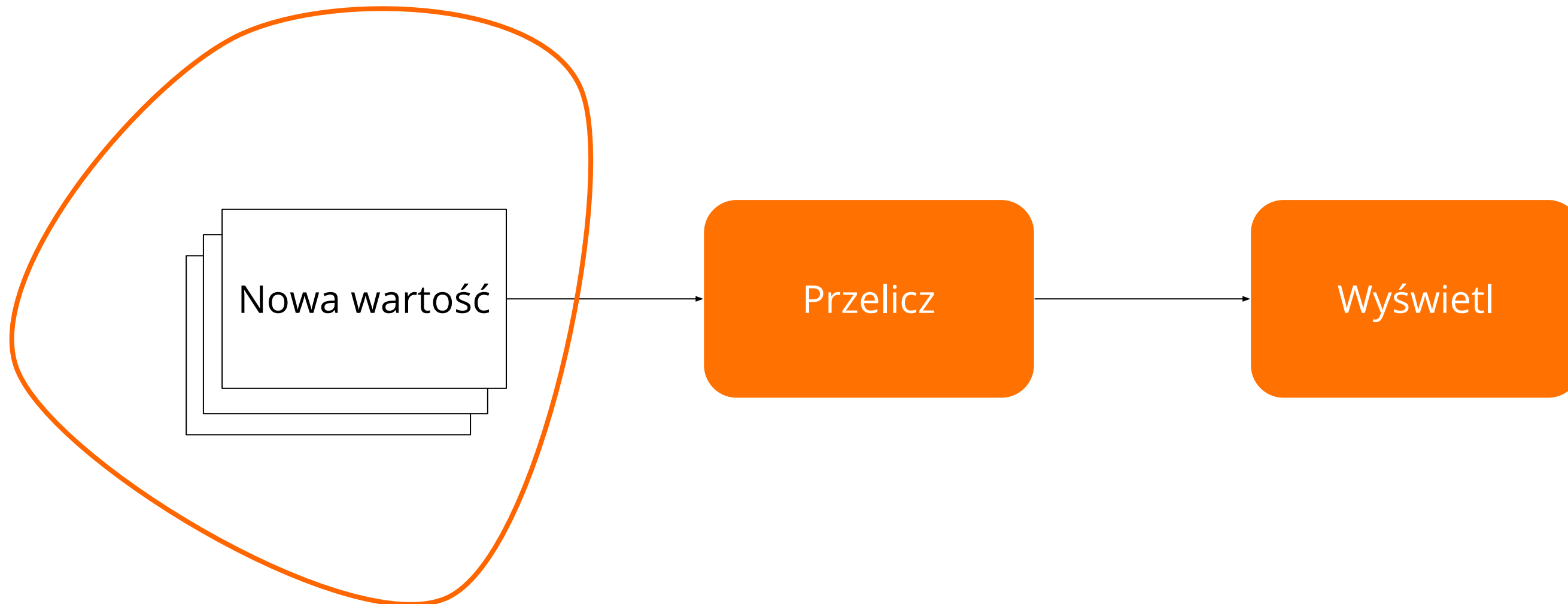
# Reaktywność jako reakcja na zdarzenia



# Reaktywność jako reakcja na zdarzenia



# Reaktywność jako reakcja na zdarzenia

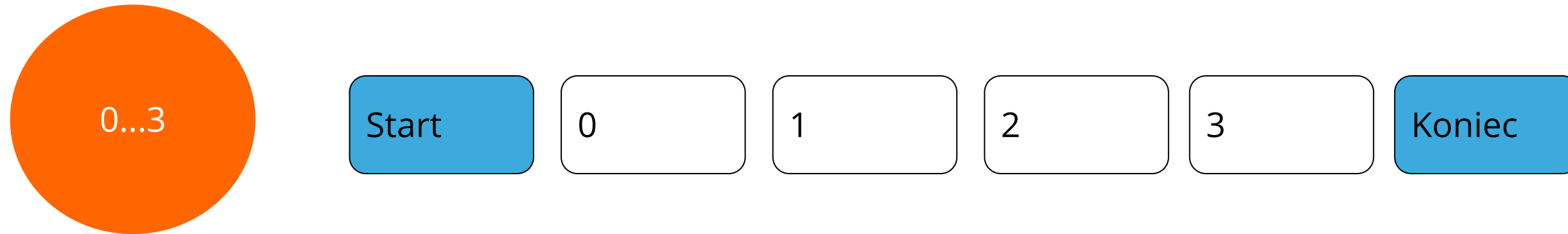


**Strumień wartości!**

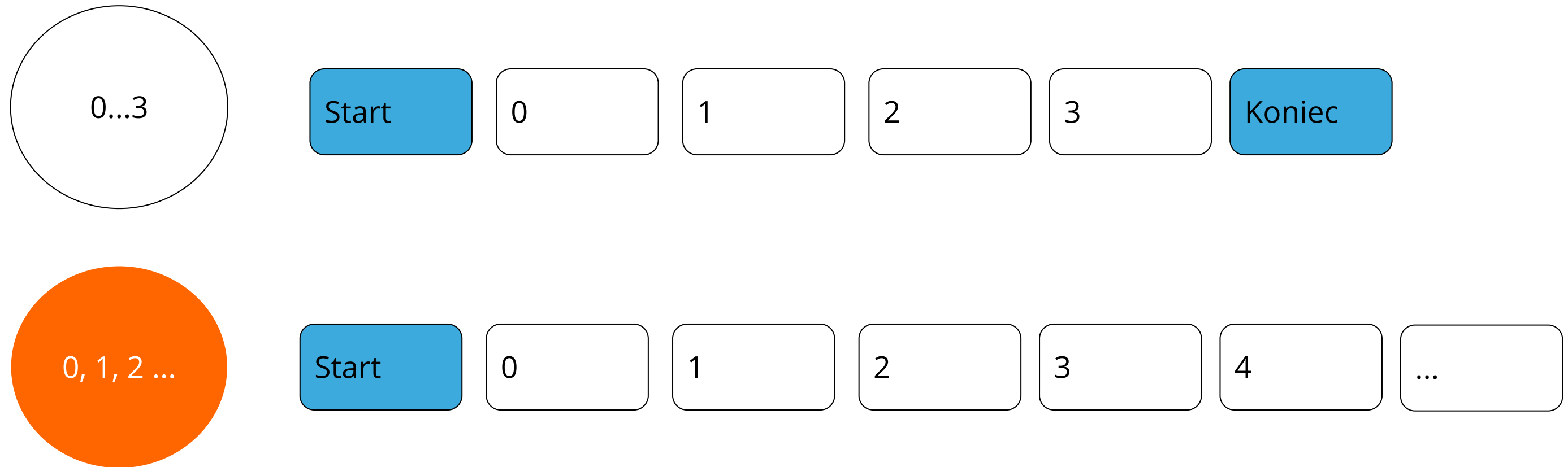
# Reaktywność jako reakcja na zdarzenia



# Strumień skończony i nieskończony

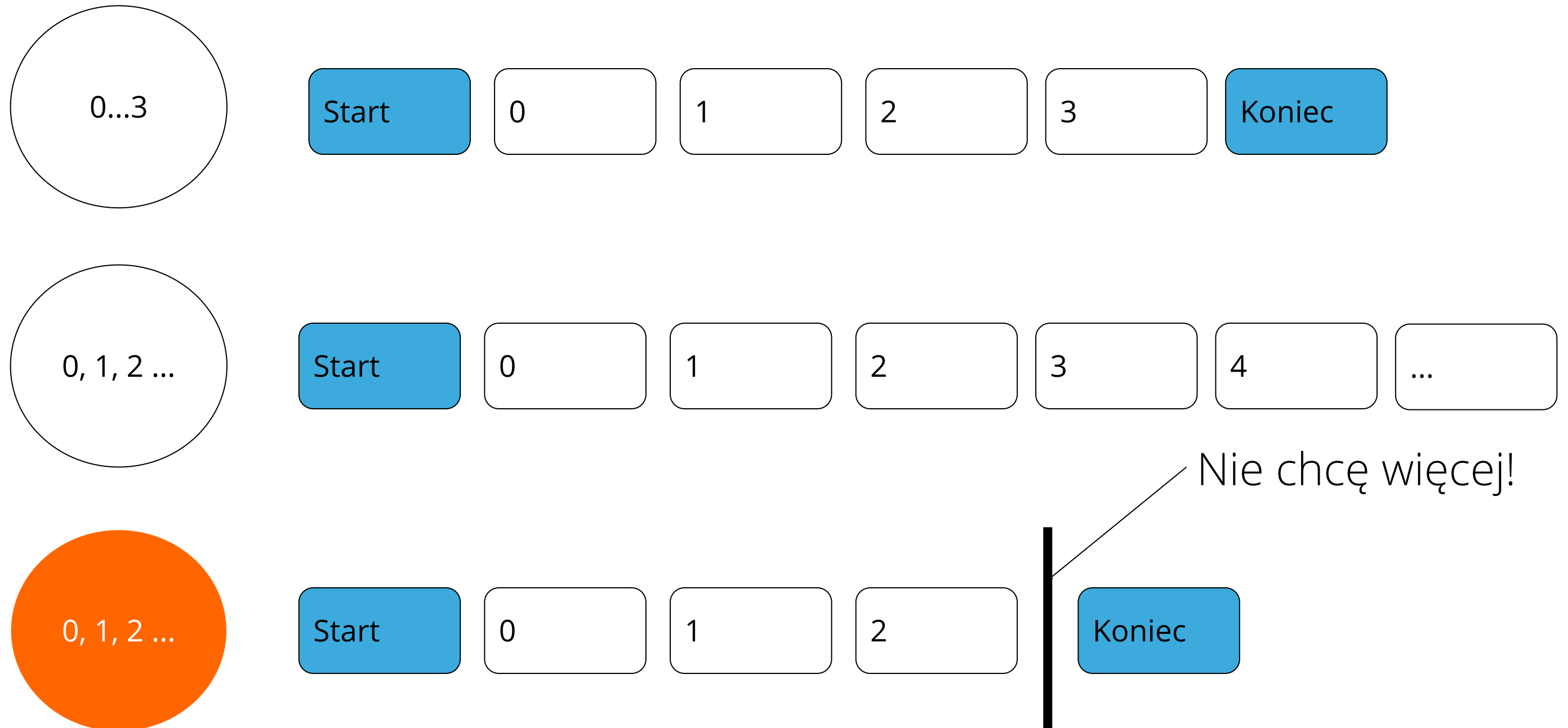


# Strumień skończony i nieskończony

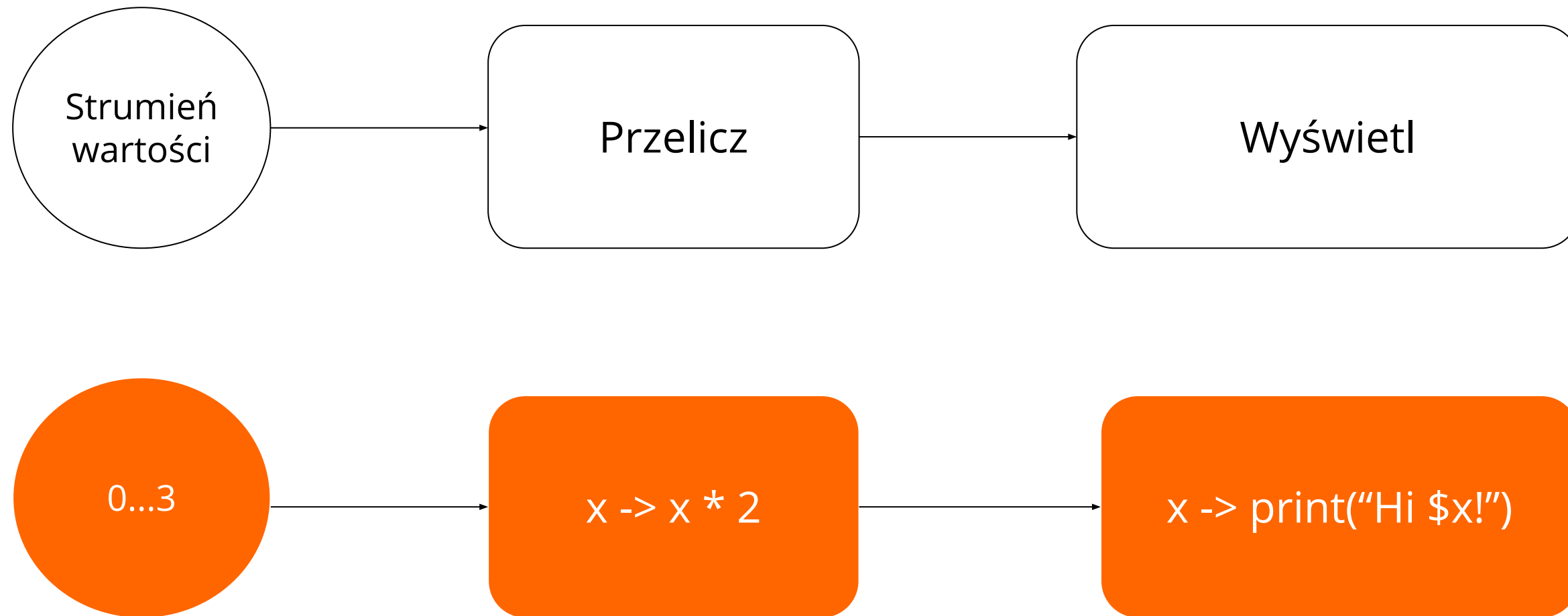




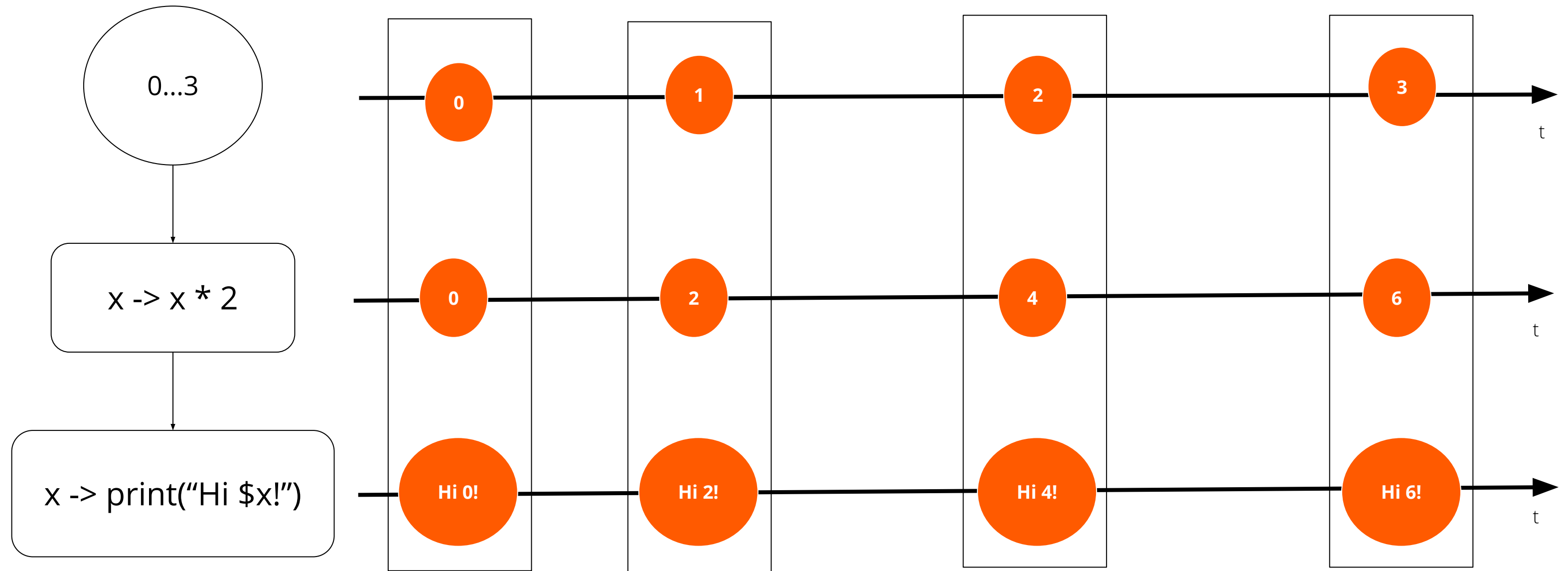
# Strumień skończony i nieskończony



# Reaktywność jako reakcja na zdarzenia



# Reaktywność jako reakcja na zdarzenia



# Programowanie reaktywne

# Programowanie reaktywne - czym jest?

*In computing, **reactive programming** is a declarative programming paradigm concerned with data streams and the propagation of change. With this paradigm it is possible to express static (e.g., arrays) or dynamic (e.g., event emitters) data streams with ease, and also communicate that an inferred dependency within the associated execution model exists, which facilitates the automatic propagation of the changed data flow.*

~ Wikipedia

# Programowanie reaktywne - czym jest? [--human-readable]

***Programowanie reaktywne*** to sposób pisania kodu, w którym traktujemy wszystko co się da jako strumień danych, który staramy się obsłużyć w sposób nieblokujący.

~ Marek

# Blocking IO vs Non-blocking IO

Pobierz zagregowaną listę lotów z Poznania do Rzymu między 7.03.2020, a 20.03.2020

Http = IO



# Blocking IO vs Non-blocking IO

A solid orange horizontal bar spanning most of the width of the slide.

IO (HTTP)

# Blocking IO vs Non-blocking IO

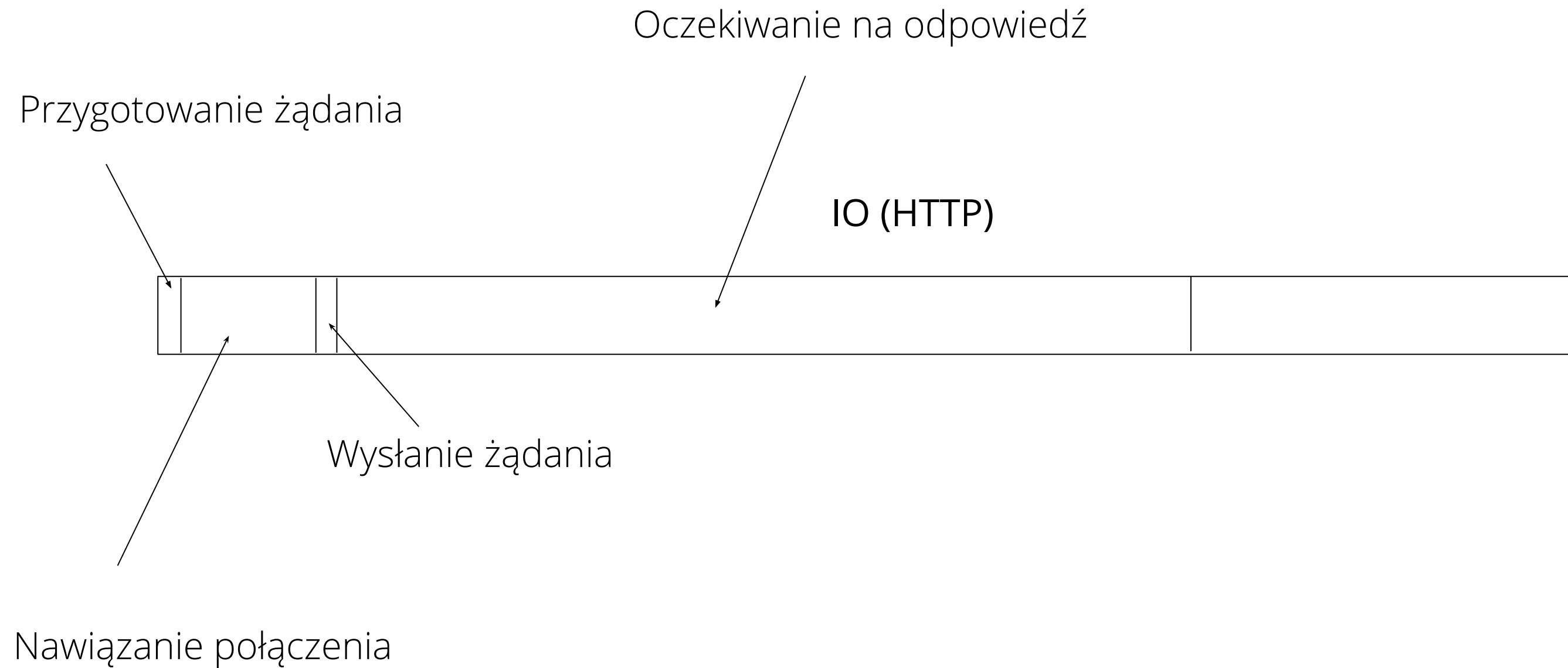
Przygotowanie żądania

IO (HTTP)

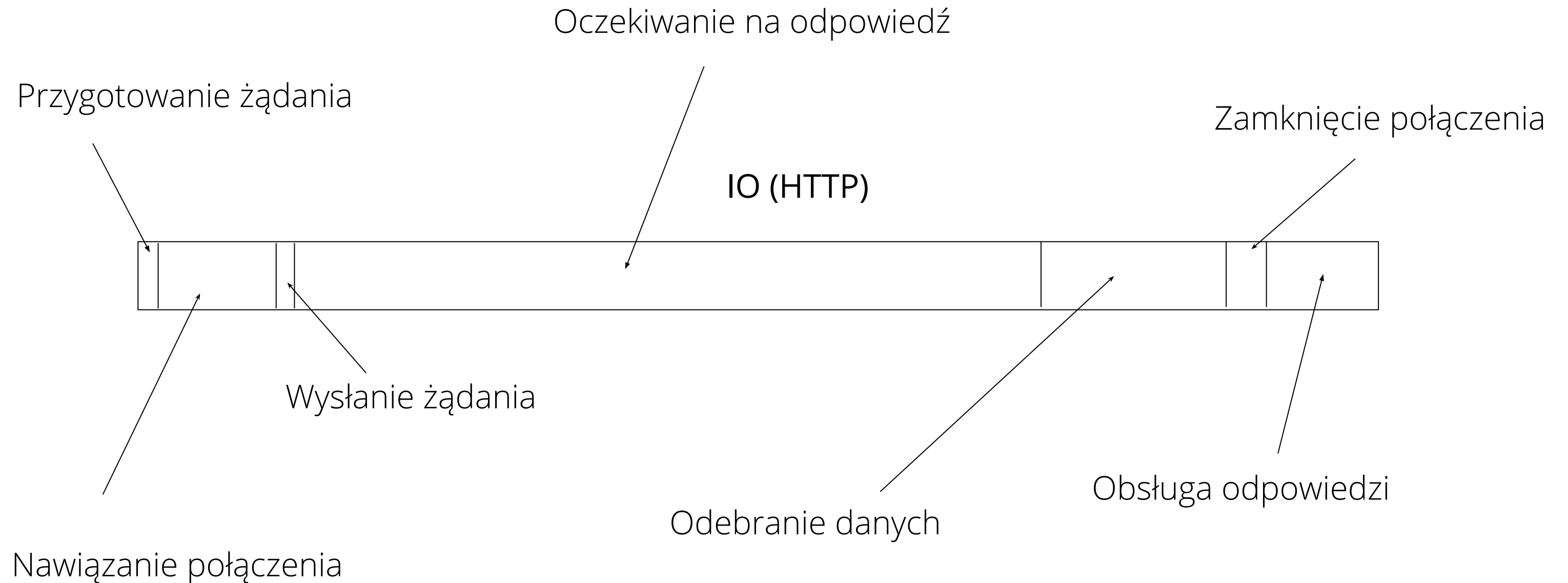
Wysłanie żądania

Nawiązanie połączenia

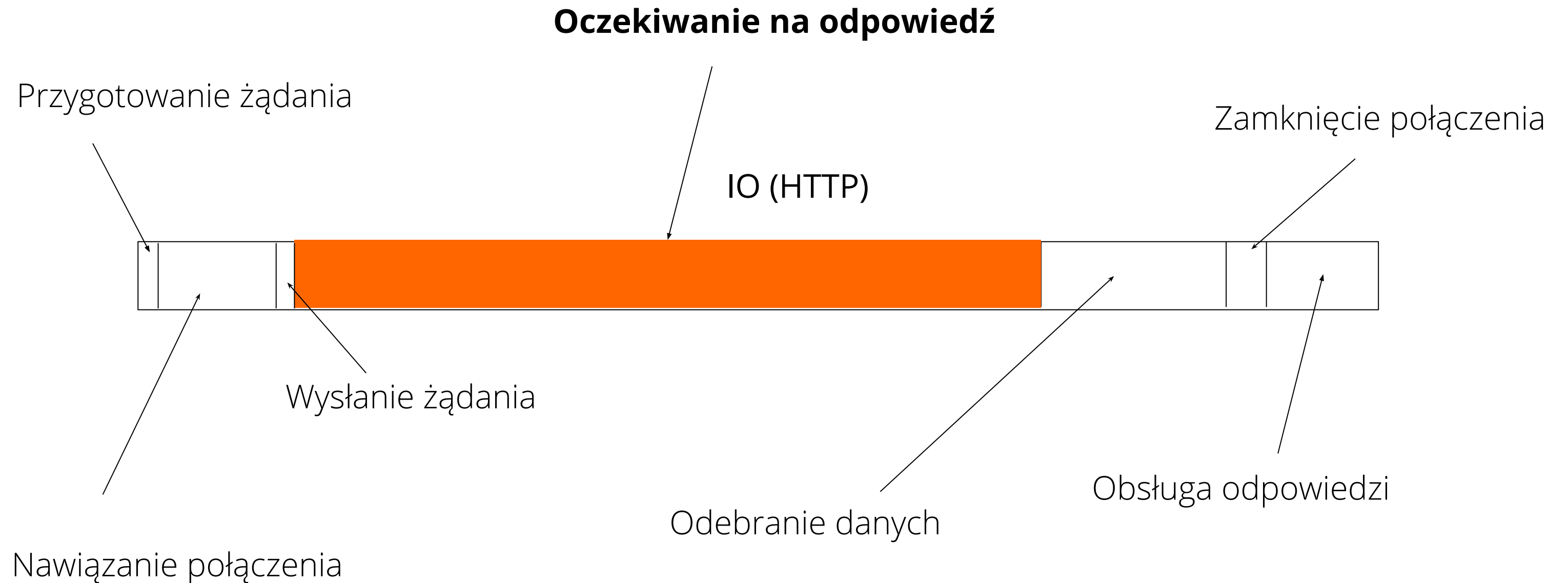
# Blocking IO vs Non-blocking IO



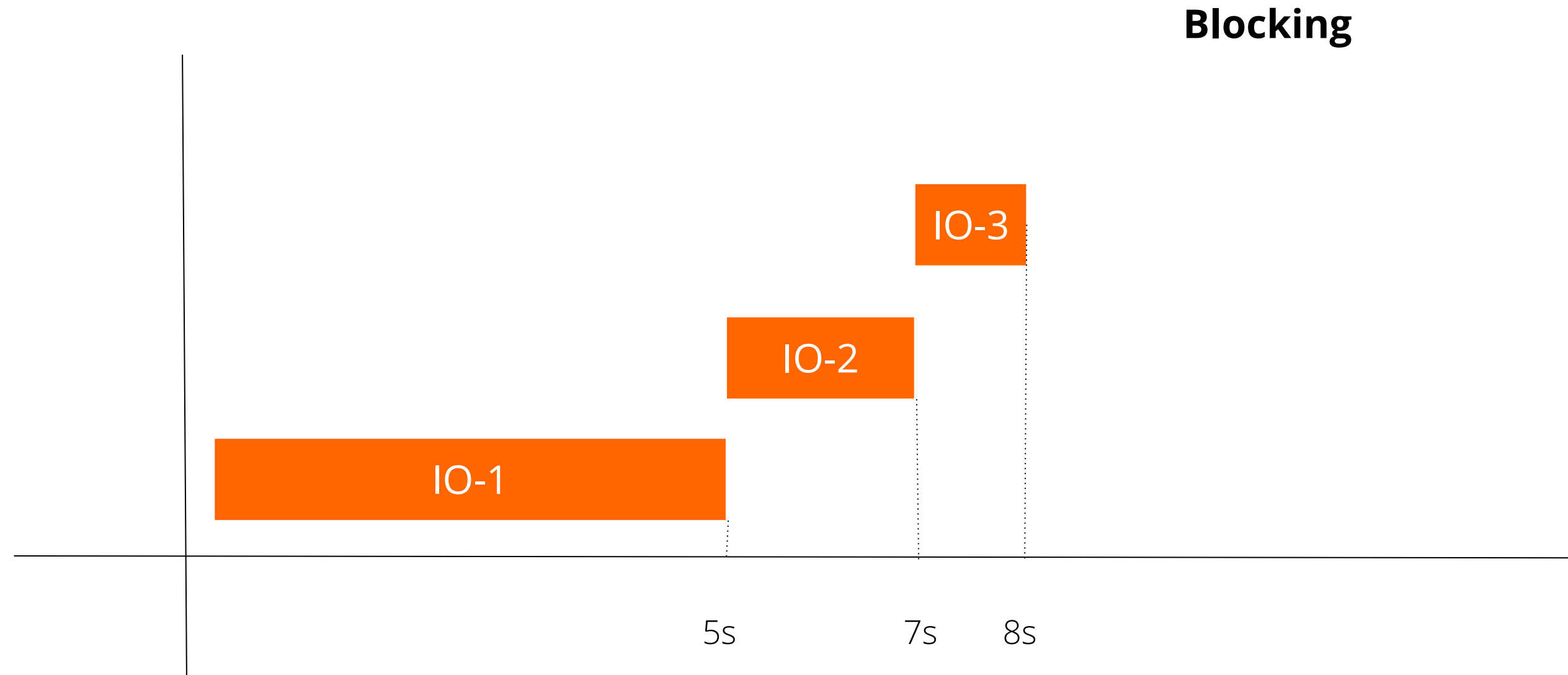
# Blocking IO vs Non-blocking IO



# Blocking IO vs Non-blocking IO

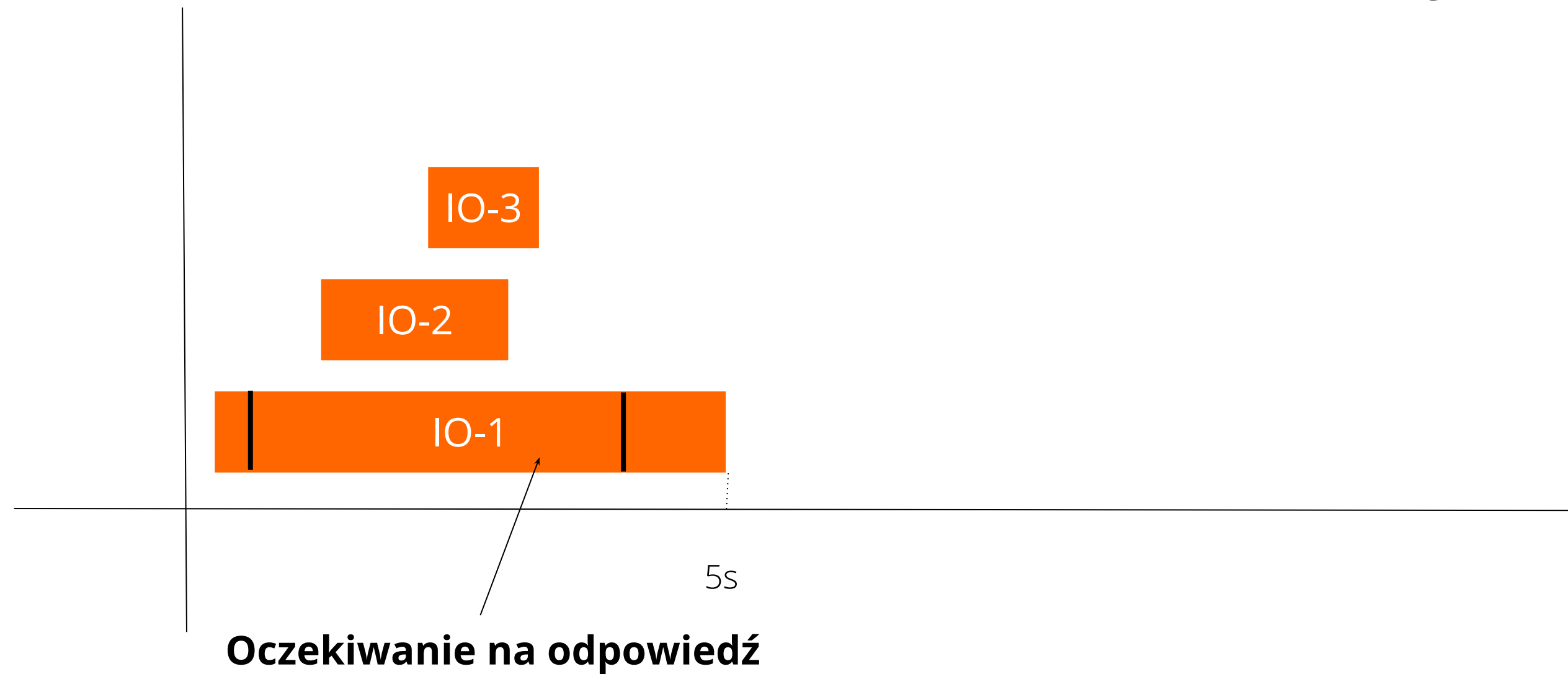


# Blocking IO vs Non-blocking IO



# Blocking IO vs Non-blocking IO

**Non-blocking**

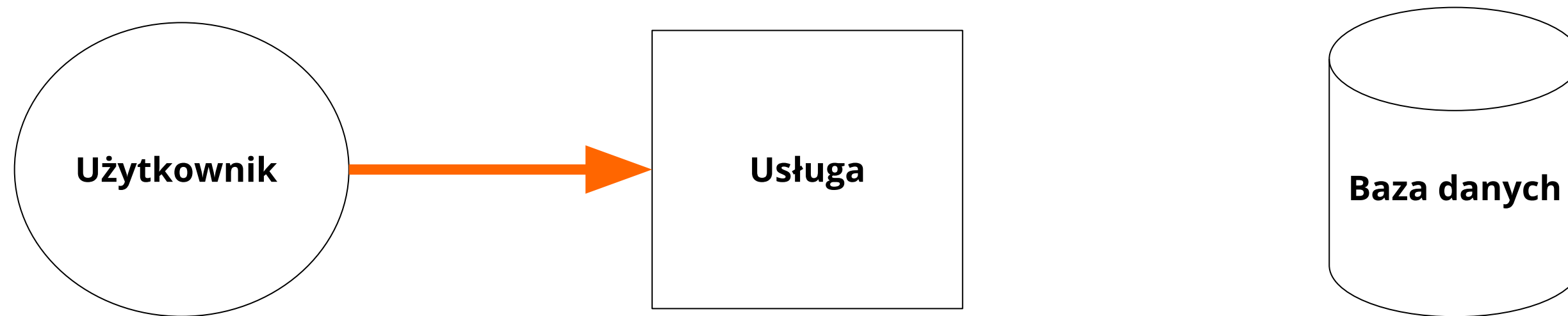




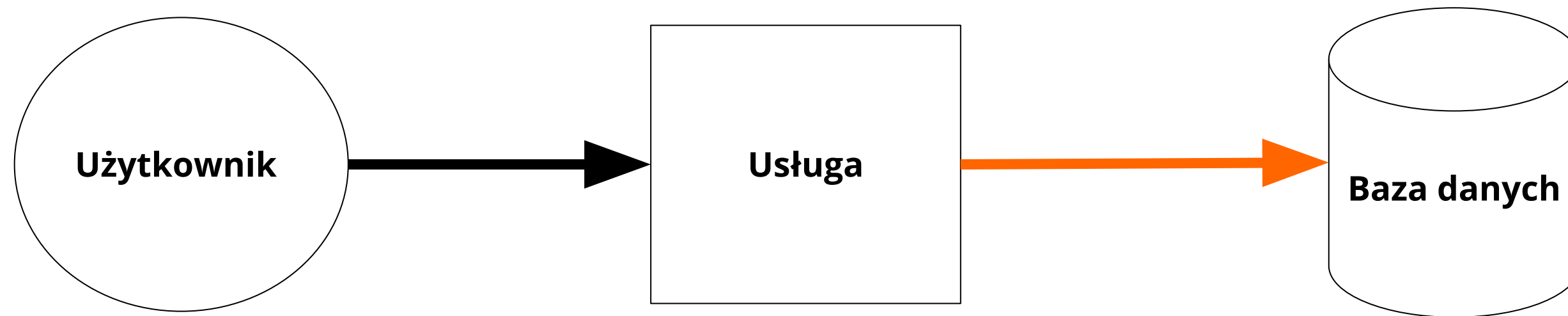
# Blocking IO



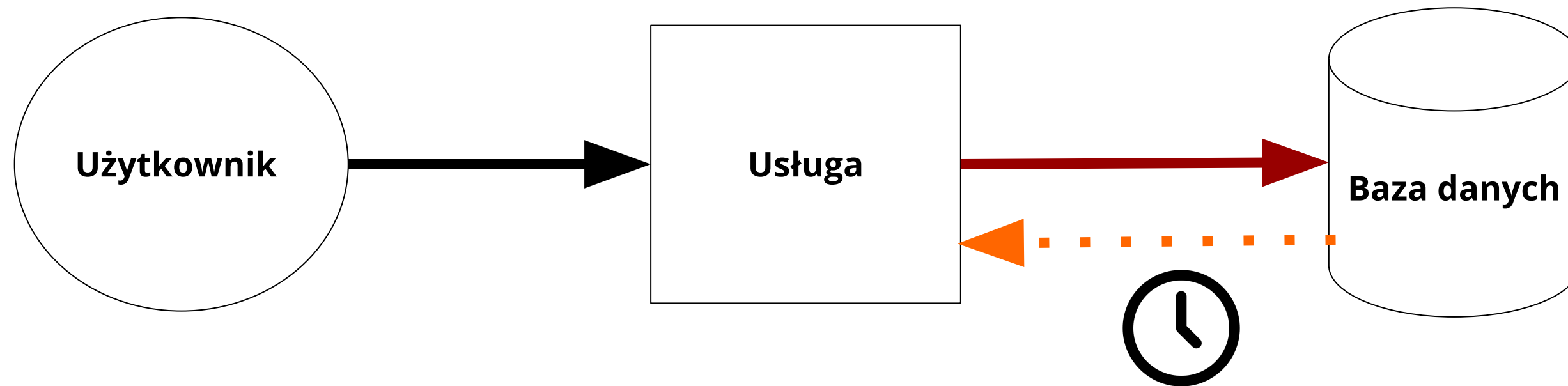
# Blocking IO



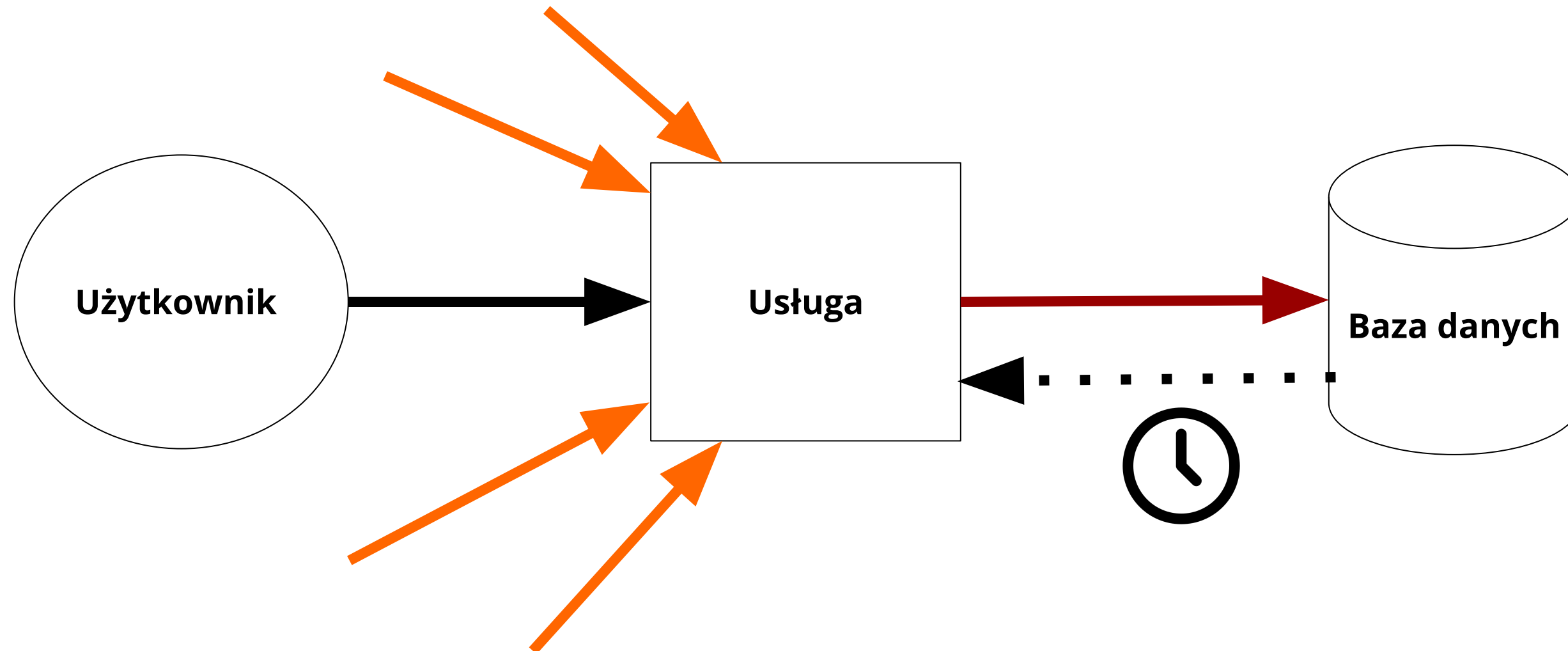
# Blocking IO



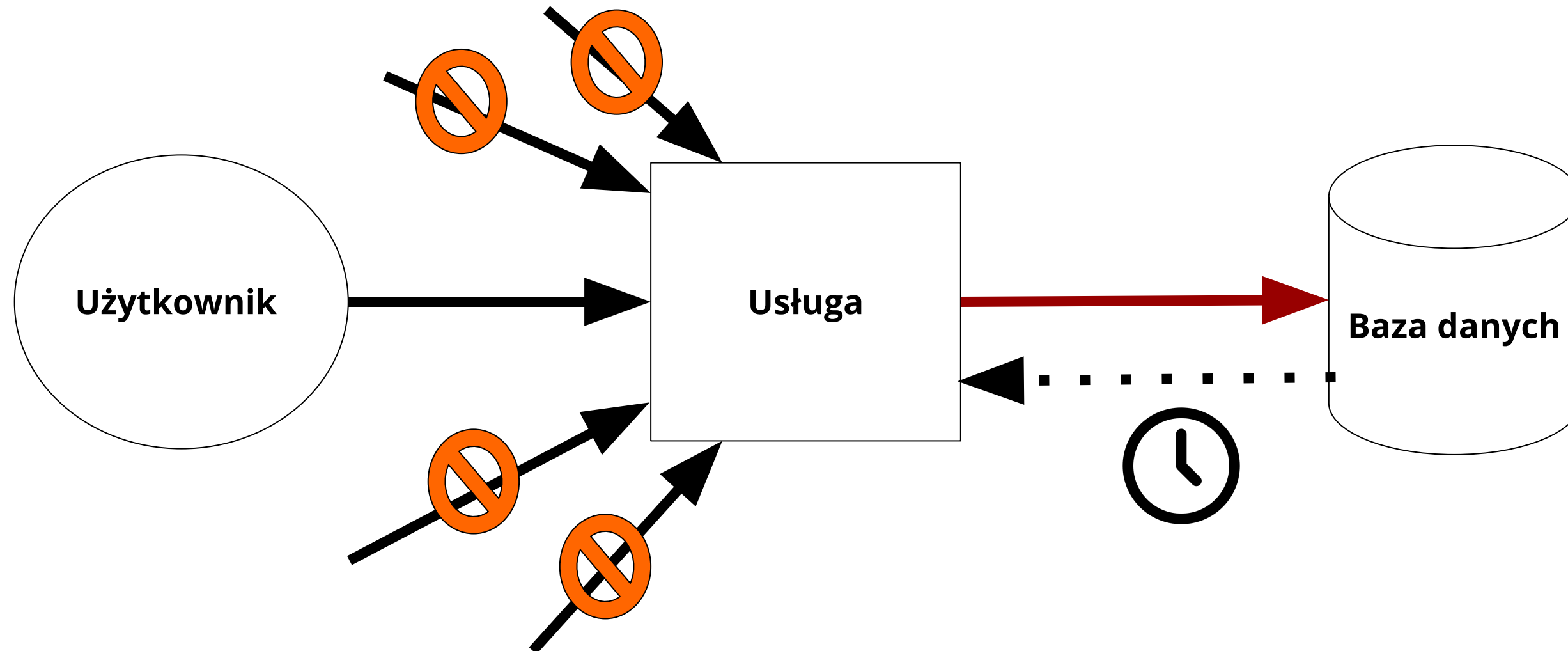
# Blocking IO



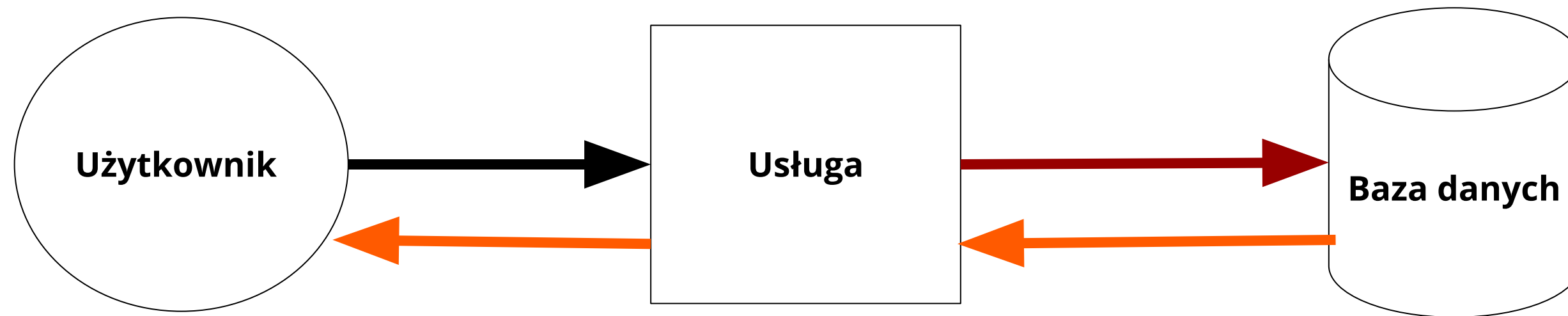
# Blocking IO



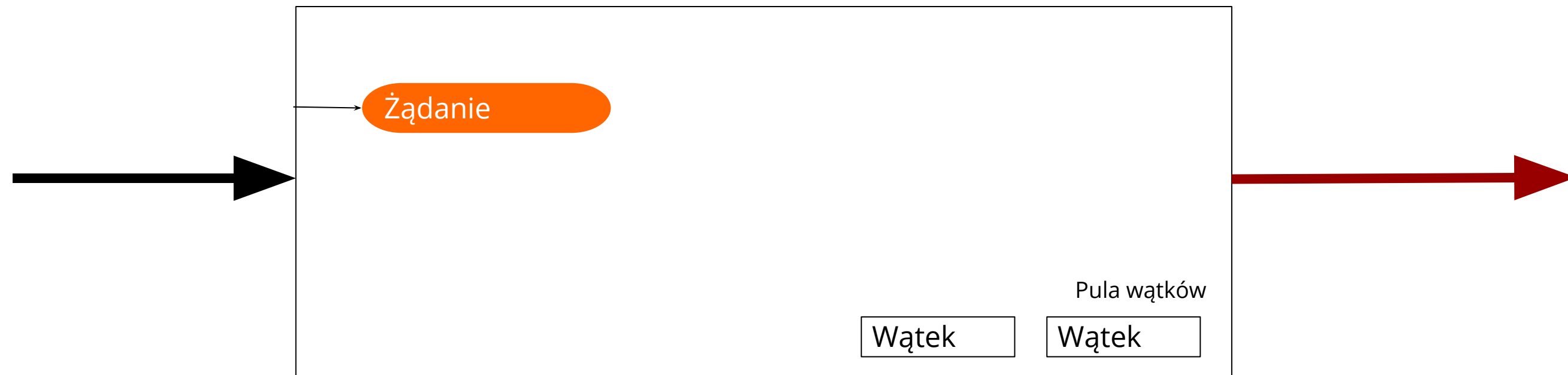
# Blocking IO



# Blocking IO

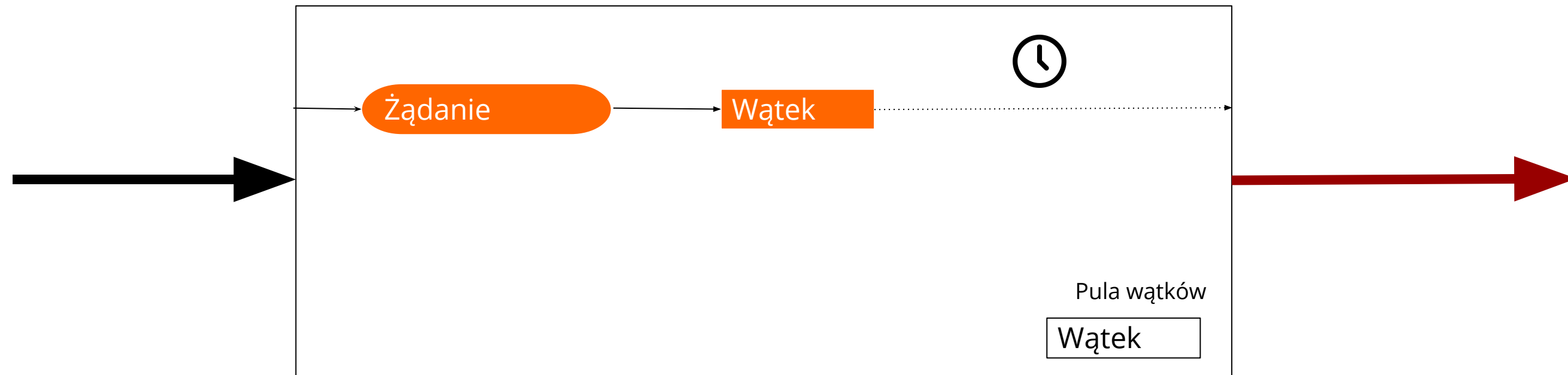


# Blocking IO

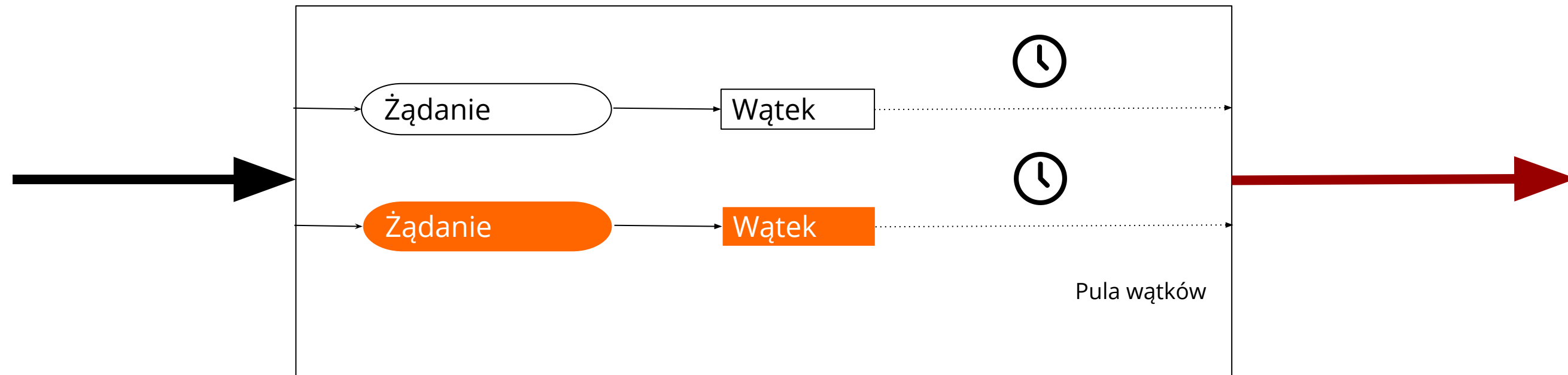




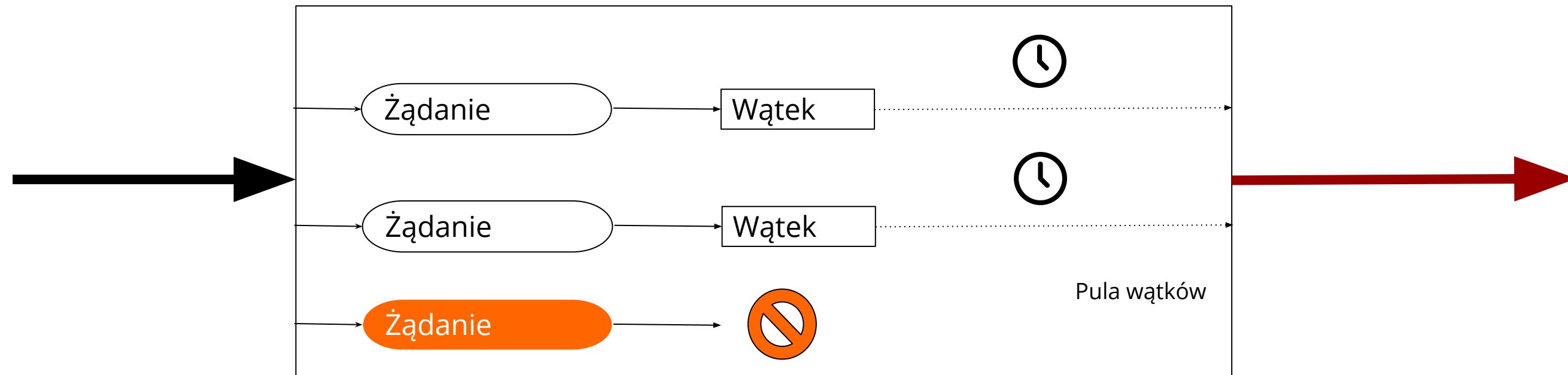
# Blocking IO



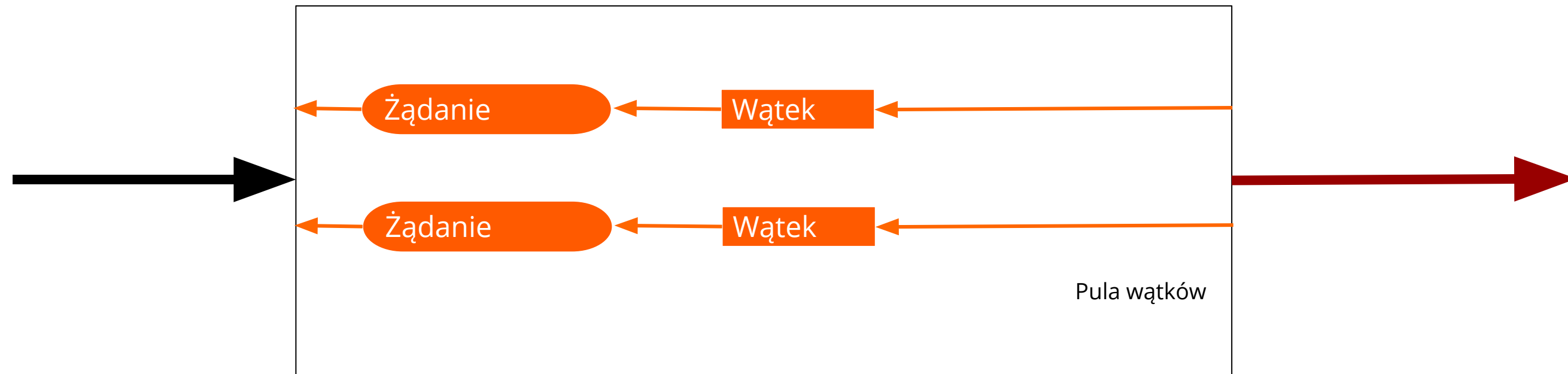
# Blocking IO



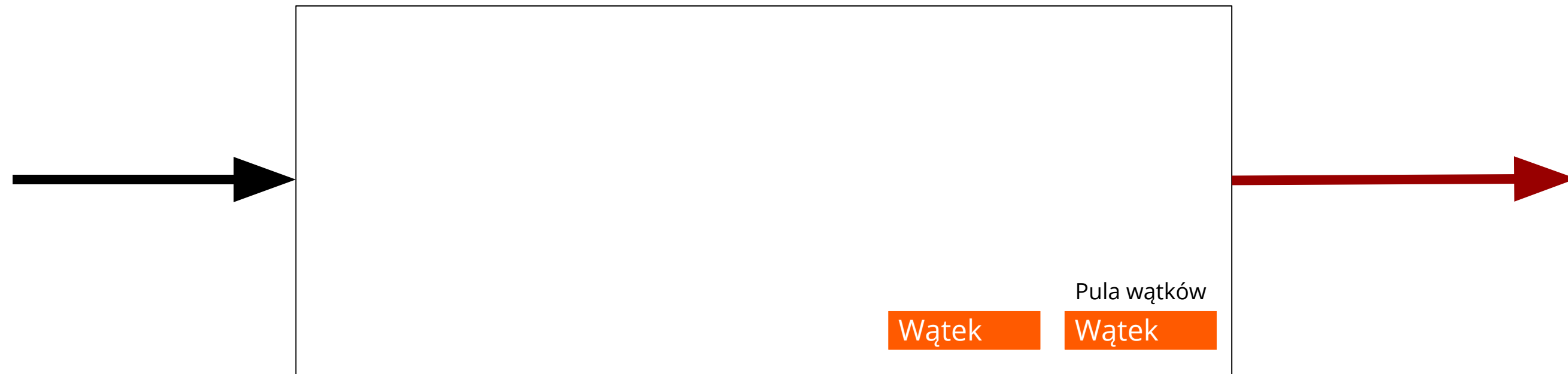
# Blocking IO



# Blocking IO



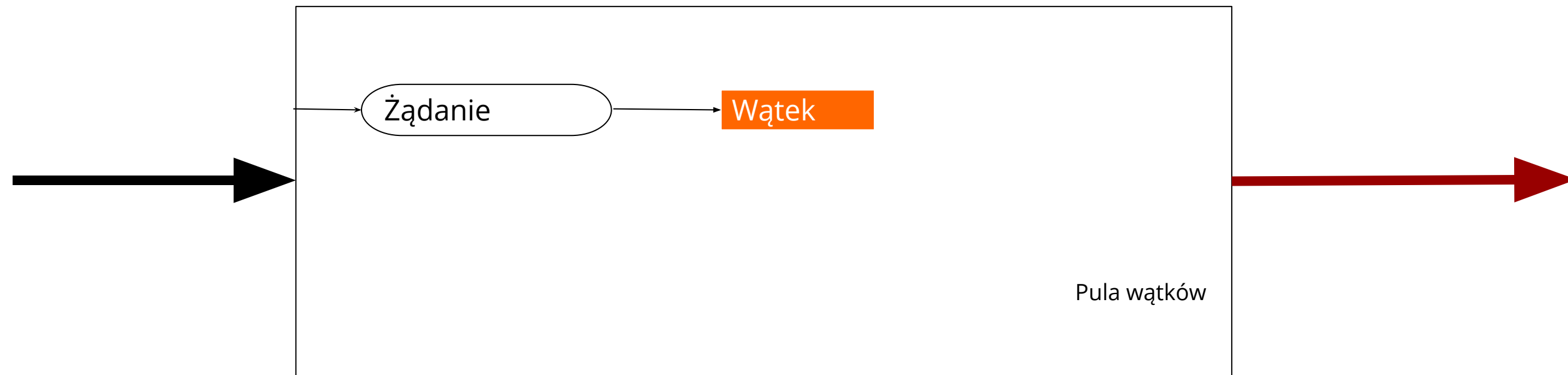
# Blocking IO



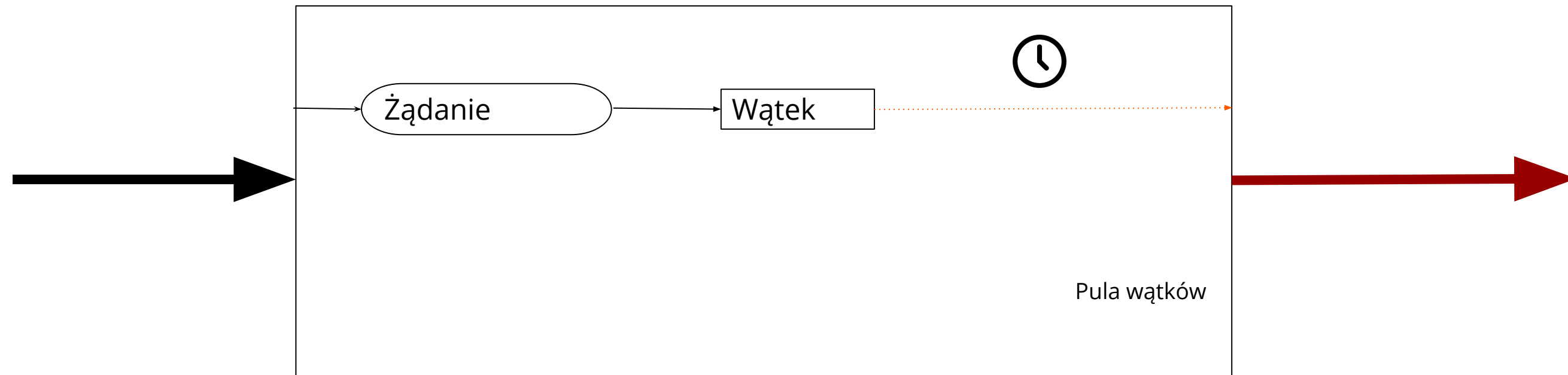
# Non-blocking IO



# Non-blocking IO

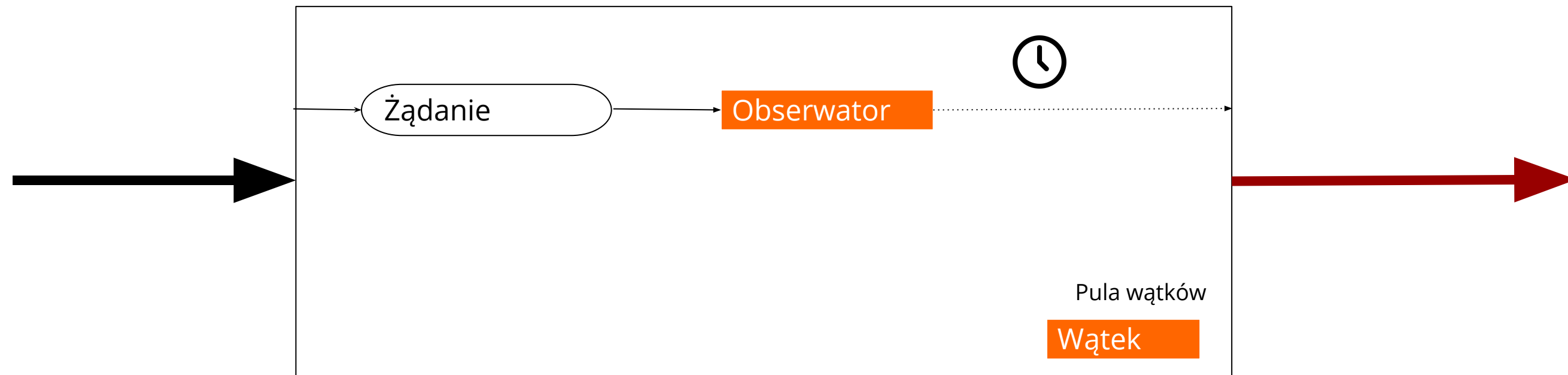


# Non-blocking IO

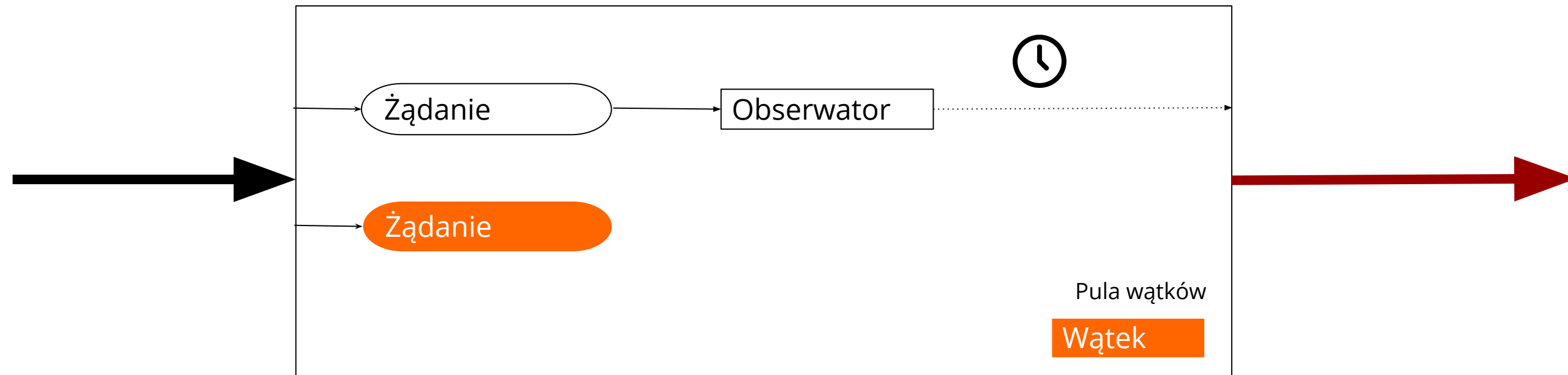




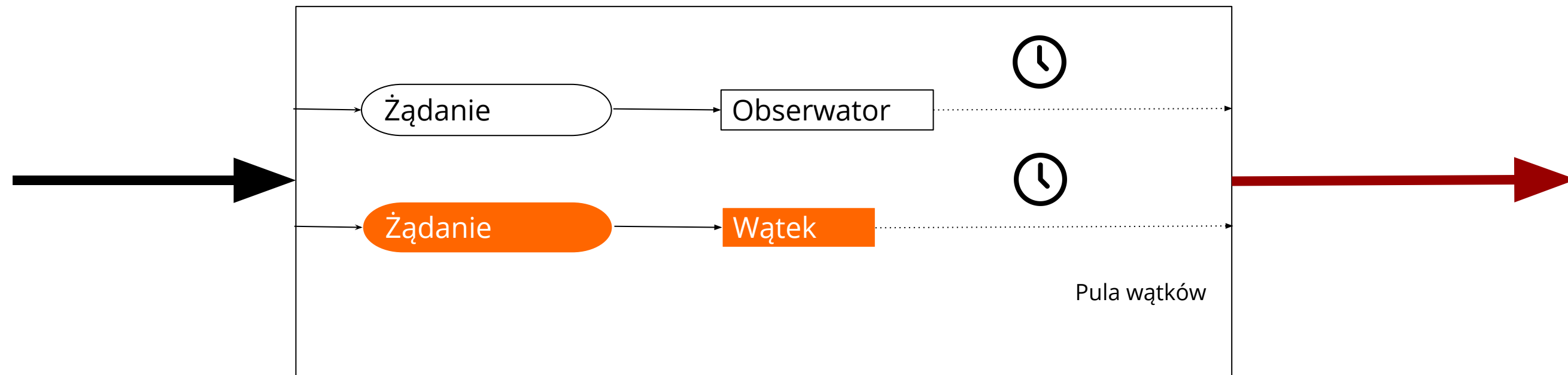
# Non-blocking IO



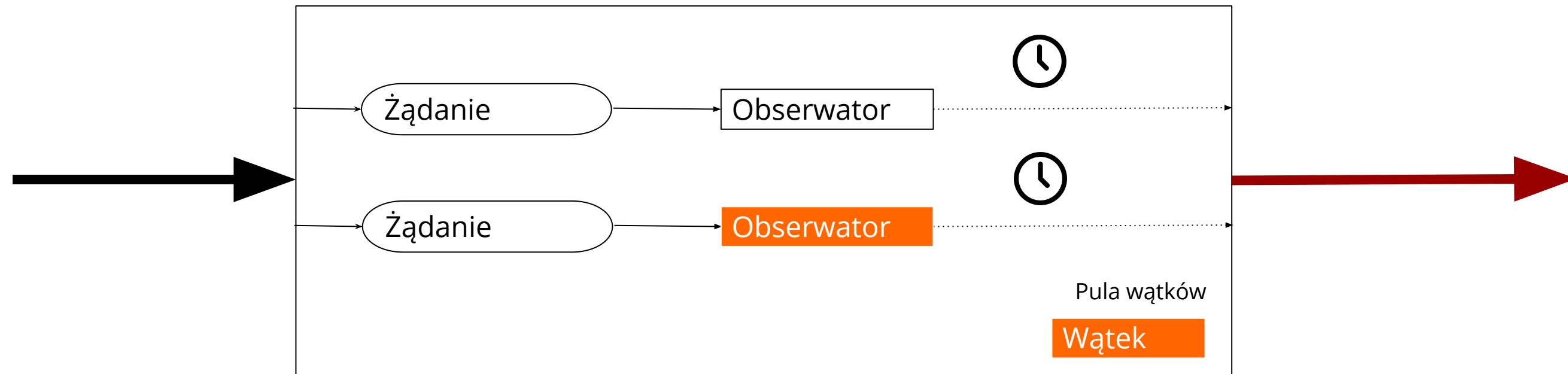
# Non-blocking IO



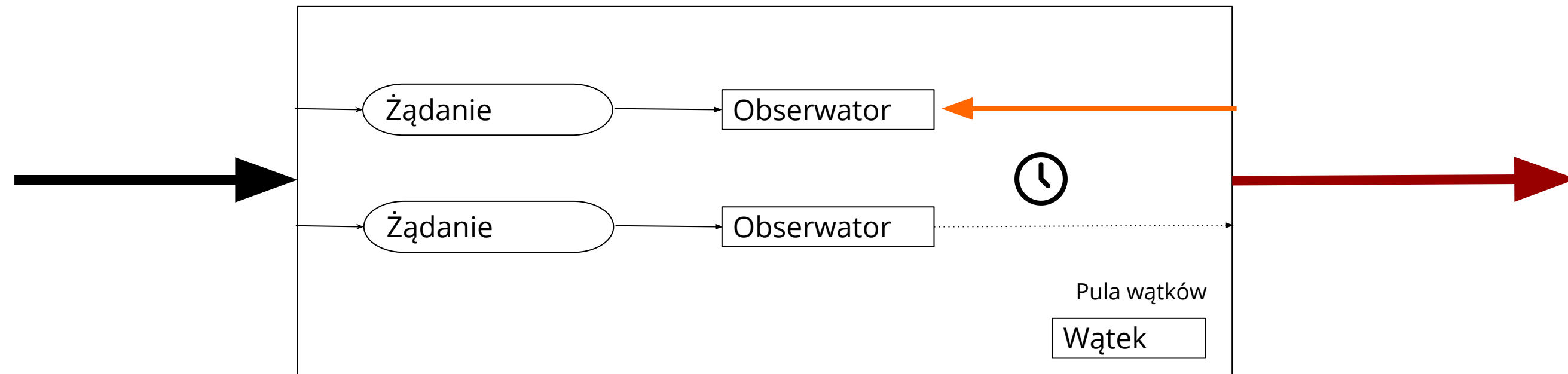
# Non-blocking IO



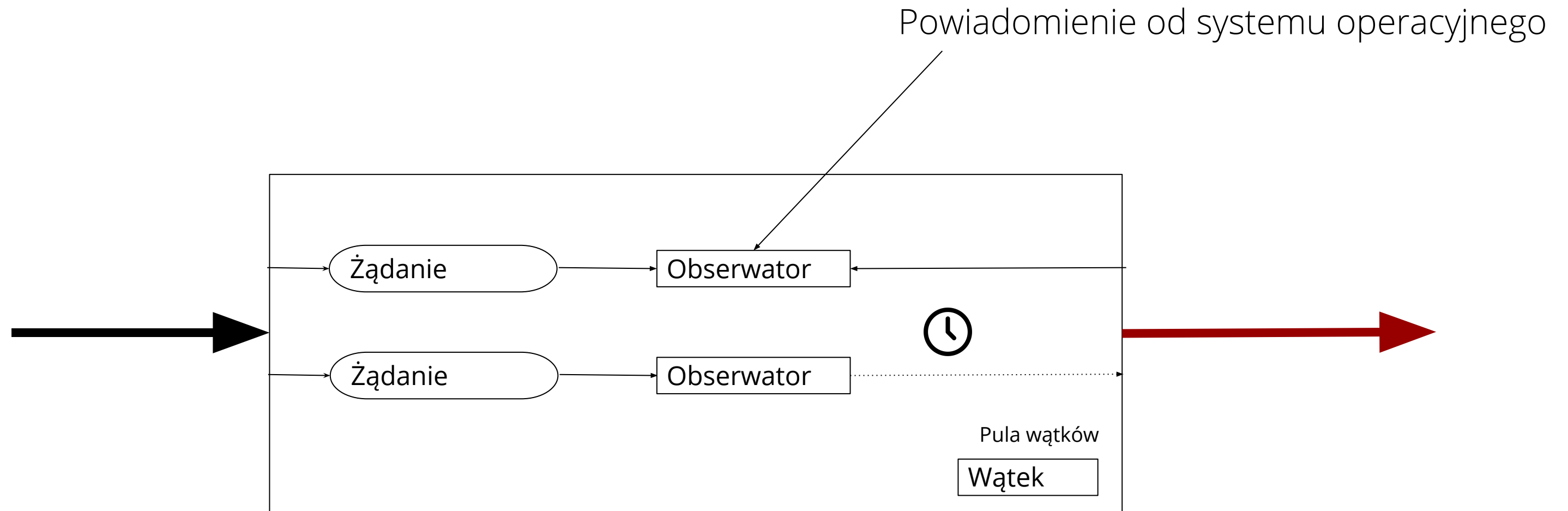
# Non-blocking IO



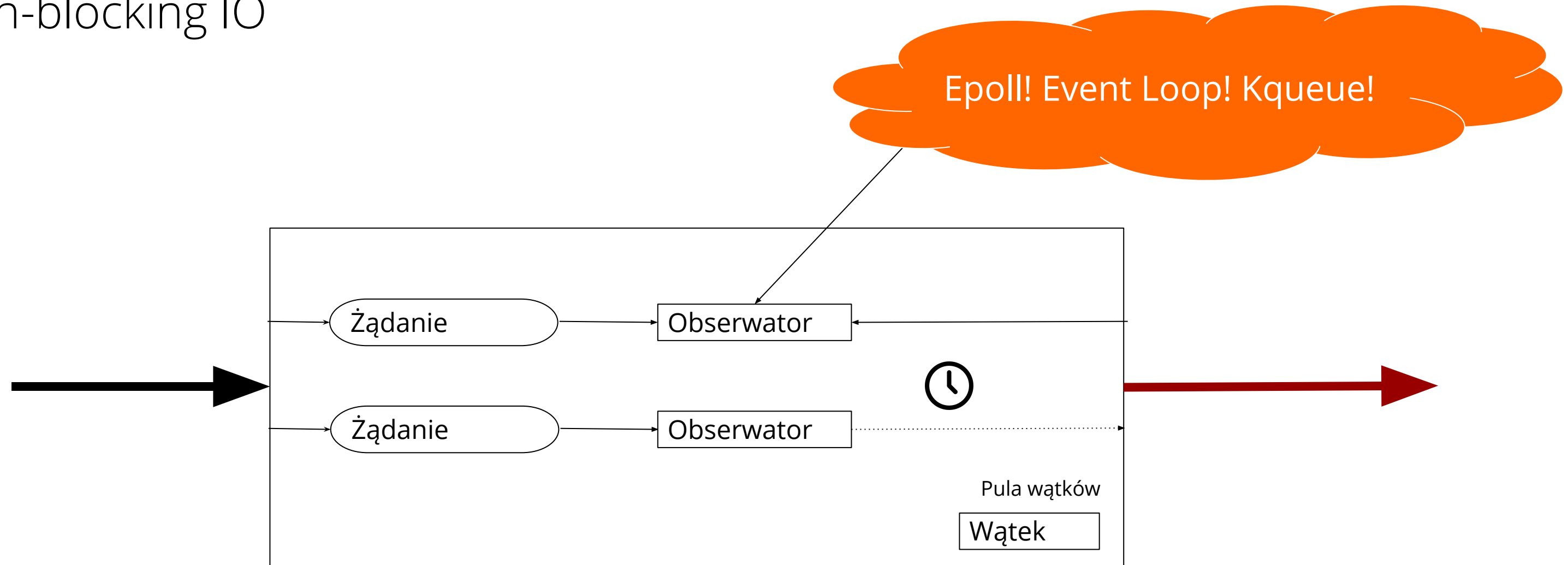
# Non-blocking IO



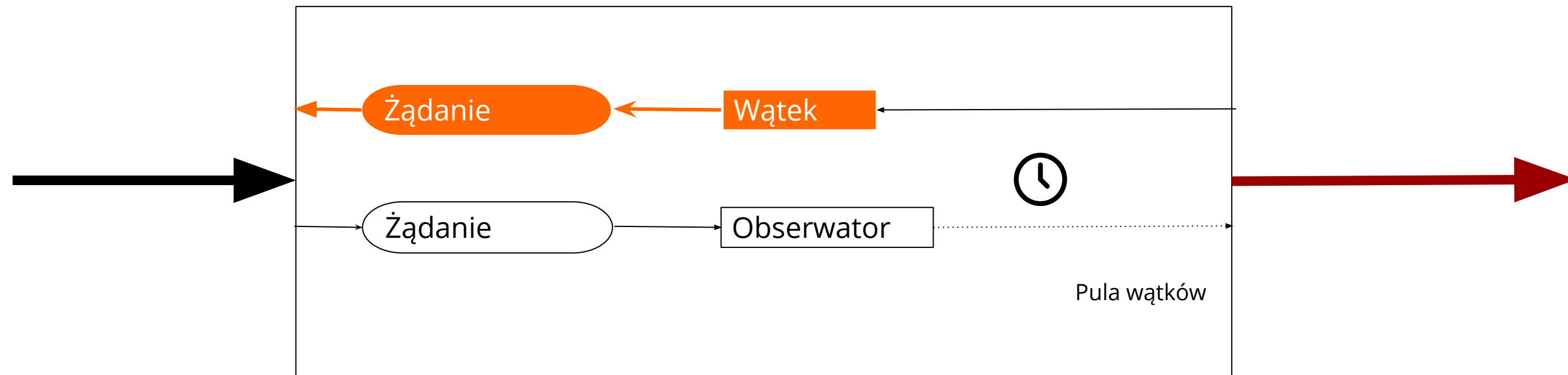
# Non-blocking IO



# Non-blocking IO

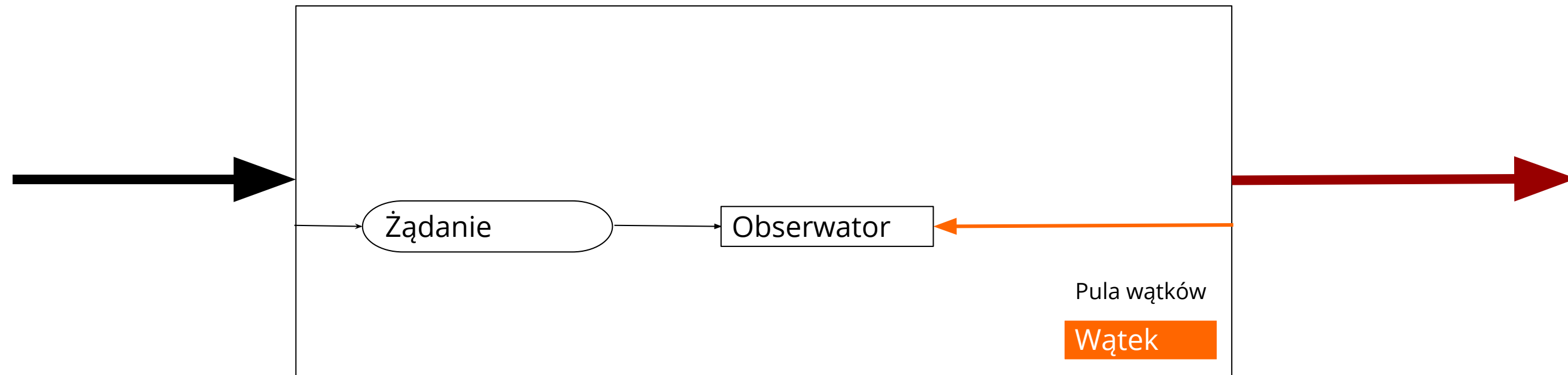


# Non-blocking IO

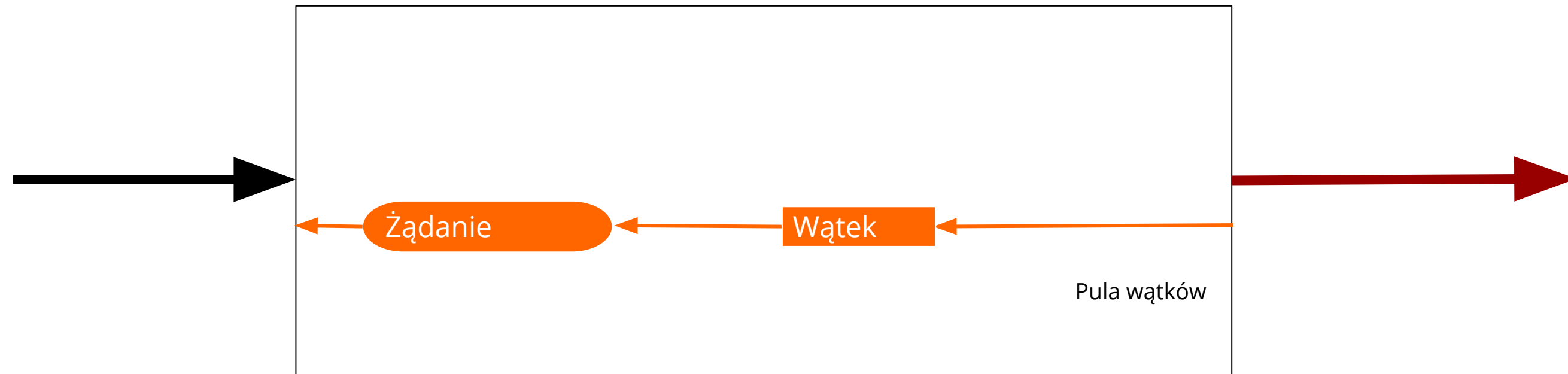




# Non-blocking IO




# Non-blocking IO



# Non-blocking IO



# Strumienie - przykład



Wszystkie kategorie
SZUKAJ

☐ szukaj u jednego sprzedawcy

NOWOŚĆ

DZIAŁY
KORZYŚCI

Elektronika

Moda

Dom i ogród

Supermarket

Dziecko

Uroda

Zdrowie

Kultura i rozrywka

Sport i turystyka

Motoryzacja

Ogłoszenia i usługi


Kolekcje i sztuka

Firma

eBilet.pl


Allegro Lokalnie

WSZYSTKIE KATEGORIE




Dzień Kobiet już-tuż
LEGO® Promocja!
Pampers -46%
Dla niej
Niezawodne spawarki

Warto zobaczyć




Limitowana  
Smart!  
okazja


Codziennie o 20:00!



Kuchnia marzeń




Wiosna w ogrodzie




8 MARCA

Na Dzień Kobiet




Atrakcje dla rodziny




Akademia Alleg


Cześć Marek!



Nowość! Allegro Smart także z kurierem!




Wymień Monety i płać mniej za zakupy. Liczba Monet: 13



Oceń produkty: 16

Okazja wybrana dla Ciebie



-8%
24,99-zł

22,99 zł

SMART z kurierem

Stażystka Alicja Sinicka

26 osób kupiło


ZOBACZ WIĘCEJ OKAZJI


Weekendowe okazje kończą się za:


07:12:14

Trafiony prezent na Dzień Kobiet

-34%
429,99-zł







# Strumienie - przykład

The screenshot displays the Allegro homepage with a focus on the 'Dzień Kobiet' (Women's Day) promotion. The search bar at the top is highlighted with an orange border, showing the placeholder 'czego szukasz?' and a 'SZUKAJ' button. Below the search bar, there's a checkbox for 'szukaj u jednego sprzedawcy' and a 'NOWOŚĆ' tag. The left sidebar lists various categories under 'DZIAŁY' and 'KORZYŚCI'. The main content area features a large banner for 'Dzień Kobiet' with a watch and jewelry, followed by a row of smaller promotional tiles for 'Dzień Kobiet tuż-tuż', 'LEGO® Promocja!', 'Pampers -46%', 'Dla niej', and 'Niezawodne spawarki'. Below this is a 'Warto zobaczyć' section with six circular icons representing different offers. The right sidebar contains a 'Cześć Marek!' section with three items and an 'Okazja wybrana dla Ciebie' section featuring a book. At the bottom, there are two more promotional banners: 'Weekendowe okazje kończą się za:' with a countdown timer and 'Trafiony prezent na Dzień Kobiet' with a discount tag.

**allegro**

czego szukasz? Wszystkie kategorie **SZUKAJ**

☐ szukaj u jednego sprzedawcy **NOWOŚĆ**

**DZIAŁY** KORZYŚCI

- Elektronika
- Moda
- Dom i ogród
- Supermarket
- Dziecko
- Uroda
- Zdrowie
- Kultura i rozrywka
- Sport i turystyka
- Motoryzacja
- Ogłoszenia i usługi
- Kolekcje i sztuka
- Firma
- eBilet.pl
- Allegro Lokalnie

**WSZYSTKIE KATEGORIE**

**MARCA Dzień Kobiet**

**SPRAWDŹ** Biżuteria i zegarki

Dzień Kobiet tuż-tuż LEGO® Promocja! Pampers -46% Dla niej Niezawodne spawarki

**Warto zobaczyć**

- Limitowana Smart! okazja Codziennie o 20:00!
- Kuchnia marzeń
- Wiosna w ogrodzie
- Na Dzień Kobiet
- Atrakcje dla rodziny
- Akademia Alleg

**Cześć Marek!**

- Nowość! Allegro Smart także z kurierem!
- Wymień Monety i płać mniej za zakupy. Liczba Monet: 13
- Oceń produkty: 16

**Okazja wybrana dla Ciebie**

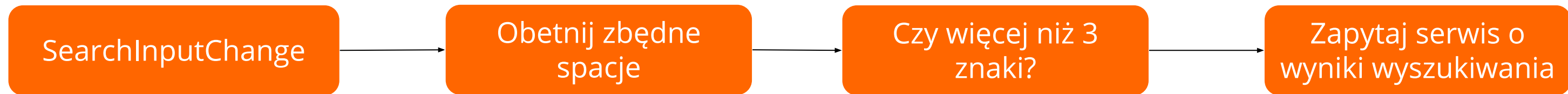
**STAŻYSTKA** -8% 24,99 zł **22,99 zł SMART** z kurierem  
Stażystka Alicja Sinicka  
26 osób kupiło

**ZOBACZ WIĘCEJ OKAZJI**

**Weekendowe okazje kończą się za:** 07:12:14

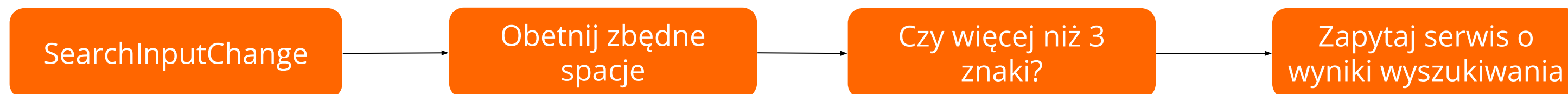
**Trafiony prezent na Dzień Kobiet** -34% 129,99 zł

# Strumienie - przykład



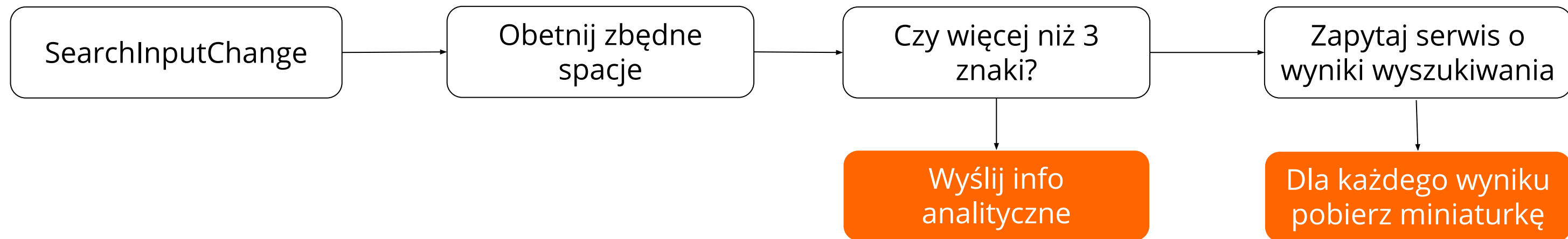


## Strumienie - przykład



```
this.input.addEventListener('change', (event) => {  
  const newValue = event.target.value;  
  const trimmedValue = newValue.trim();  
  if (trimmedValue.length > 3) {  
    this.results = this.getResults(trimmedValue);  
  }  
});
```

# Strumienie - przykład



```
this.input.addEventListener('change', (event) => {
  const newValue = event.target.value;
  const trimmedValue = newValue.trim();
  if (trimmedValue.length > 3) {
    this.getResults(trimmedValue, func: (results) => {
      this.results.clear();
      results.forEach( callbackfn: res => {
        this.getThumbnail(res, param: thumb => {
          this.results.add(new ResultWithThumbnail(res, thumb));
        });
      });
    });

    this.sendAnalytics(trimmedValue.length);
  }
});
```



# Strumienie - callback that!



# Callbacks - problemy

Czytelność

X oraz Y

Agregacje strumieni

X lub Y

Rezygnowanie z operacji

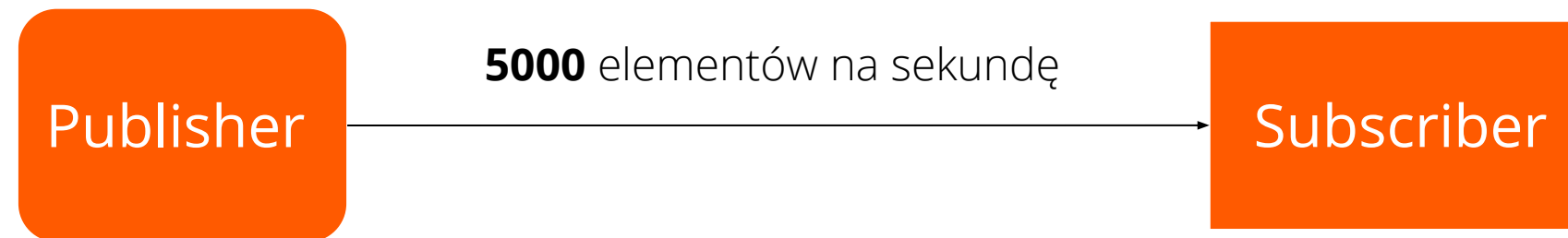
Backpressure

Powtórzenia

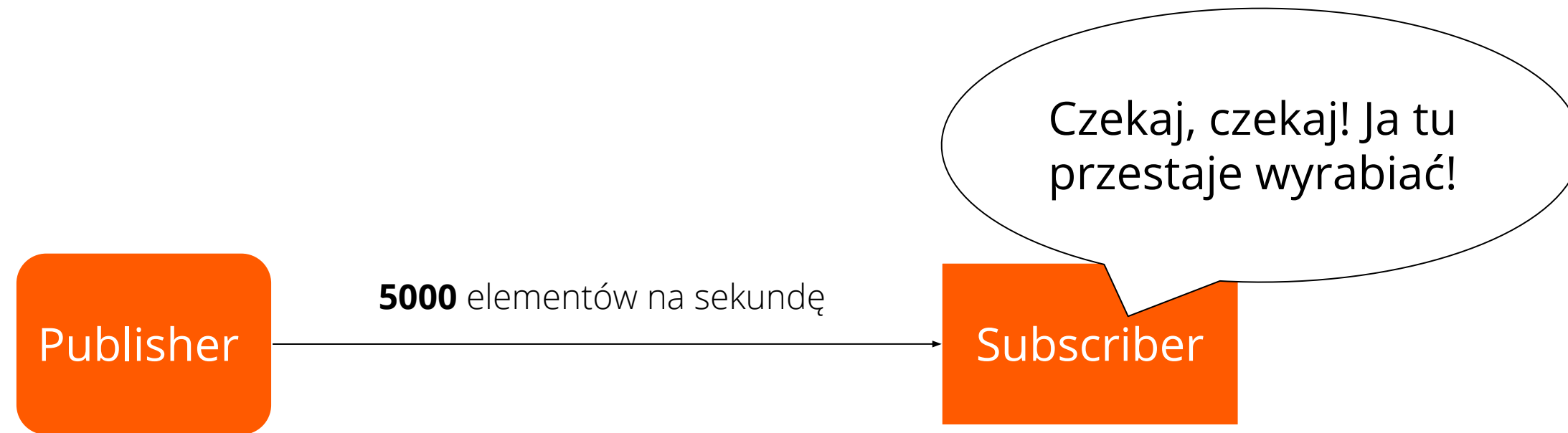


Operacje na oknach

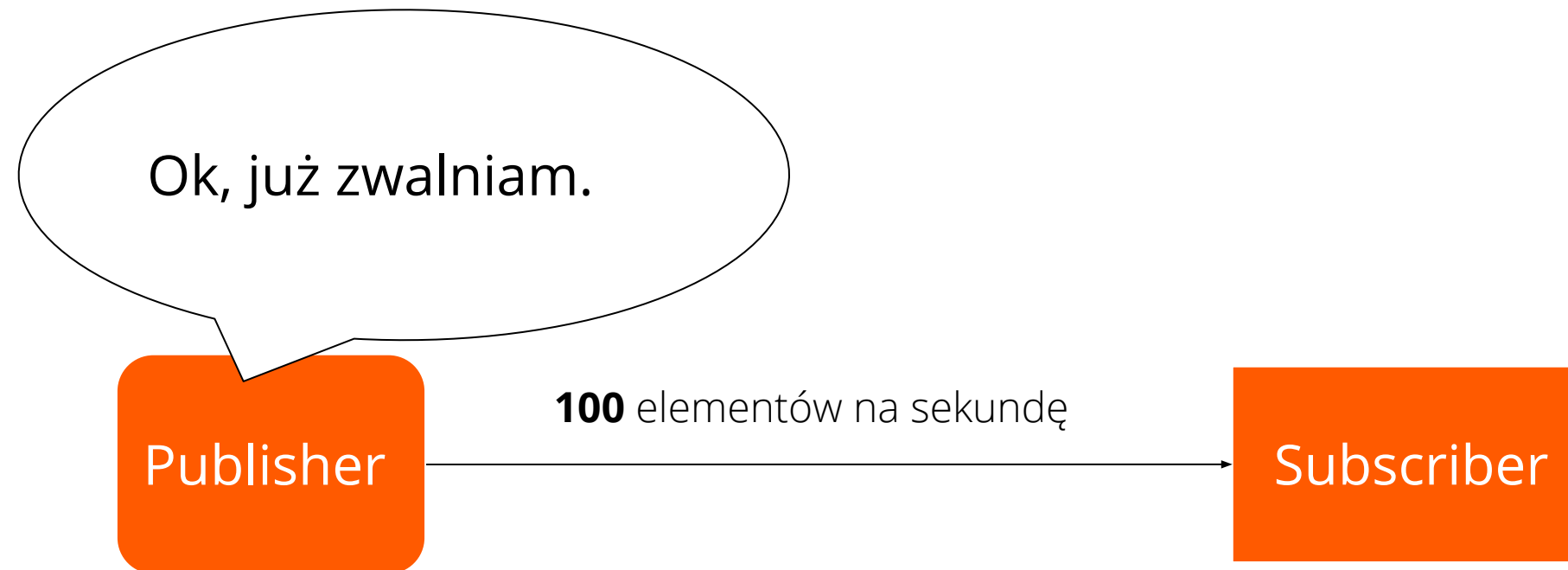
# Backpressure



# Backpressure



# Backpressure

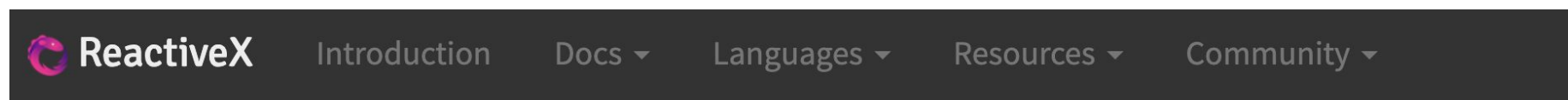


# Backpressure



# Reactive Extensions

# Reactive Extensions (Rx\* / ReactiveX)



## Languages

- Java: [RxJava](#)
- JavaScript: [RxJS](#)
- C#: [Rx.NET](#)
- C#(Unity): [UniRx](#)
- Scala: [RxScala](#)
- Clojure: [RxClojure](#)
- C++: [RxCpp](#)
- Lua: [RxLua](#)
- Ruby: [Rx.rb](#)
- Python: [RxPY](#)
- Go: [RxGo](#)
- Groovy: [RxGroovy](#)
- JRuby: [RxJRuby](#)
- Kotlin: [RxKotlin](#)
- Swift: [RxSwift](#)
- PHP: [RxPHP](#)
- Elixir: [reaxive](#)
- Dart: [RxDart](#)

## ReactiveX for platforms and frameworks

- [RxNetty](#)
- [RxAndroid](#)
- [RxCocoa](#)



# Reactive Extensions vs Reactive Streams



**Zbiór bibliotek** do programowania reaktywnego. Mogą, ale nie muszą być kompatybilne z Reactive Streams

Przykłady:

- RxJava 2,
- RxJS,
- Rx.NET

**Standard** uspójniający obsługę asynchronicznych strumieni danych w nieblokujący sposób, zapewniając przy tym backpressure.

Przykłady implementacji:

- Project Reactor
- Java 9 Flow API
- Akka Streams
- RxJava 2

## Przykład - Project Reactor

```
Flux.just(5, 10, 15)
    .filter(x -> x > 5)
    .map(x -> x + 1)
    .subscribe(x -> logger.info("Got an integer: " + x));
```

## Przykład - Project Reactor

```
Flux.just(5, 10, 15)
    .filter(x -> x > 5)
    .map(x -> x + 1)
    .subscribe(x -> logger.info("Got an integer: " + x));
```

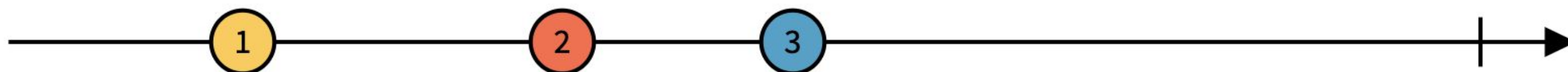
**Functional Reactive Programming**  
proste funkcje bez efektów ubocznych

# Operator

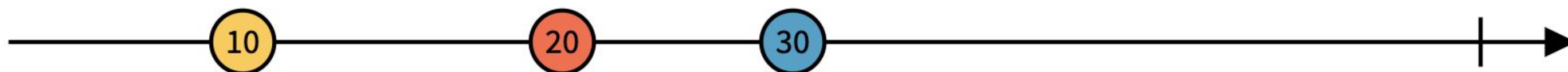
Typ operacji	RxJs 5	RxJava 2	Project Reactor
Tworzenie	<i>of(42, 43, 44)</i>	<i>Flowable.just(42, 43, 44)</i>	<i>Flux.just(42, 43, 44)</i>
Transformacja	<i>obs.map(x =&gt; x + 1)</i>	<i>f.map(x -&gt; x + 1)</i>	<i>f.map(x -&gt; x + 1)</i>
Filtrowanie	<i>obs.filter(x =&gt; x &gt; 5)</i>	<i>f.filter(x -&gt; x &gt; 5)</i>	<i>f.filter(x -&gt; x &gt; 5)</i>
Łączenie	<i>zip(obs1, obs2)</i>	<i>f1.zipWith(f2)</i>	<i>f1.zipWith(f2)</i>
Blokujące pobieranie	Brak	<i>f.blockingFirst()</i>	<i>f.blockFirst()</i>
Nieblokujące reagowanie	<i>obs.subscribe()</i>	<i>f.subscribe()</i>	<i>f.subscribe()</i>

# RxMarbles

## Interactive diagrams of Rx Observables



`map(x => 10 * x)`



<https://rxmarbles.com/>

Czy wszystko napisane w Rx\* jest **nieblokujące**?

Czy wszystko napisane w Rx\* jest **nieblokujące**?

# Nie!

Blokujące

```
integer = Flux.just(5, 10, 15)
    .filter(x -> x > 5)
    .map(x -> x + 1)
    .blockFirst();
logger.info("Got an integer: " + integer);
```

```
Flux.just(5, 10, 15)
    .filter(x -> x > 5)
    .map(x -> x + 1)
    .subscribe(x -> logger.info("Got an integer: " + x));
```

Nieblokujące

Czy wszystko napisane w Rx\* jest **zrównoleglone**?



Czy wszystko napisane w Rx\* jest **zrównoleglone**?

# Nie!

Domyślnie na tym samym wątku

```
Flux.just(5, 10, 15)
    .filter(x -> x > 5)
    .map(x -> x + 1)
    .subscribe(x -> logger.info("Got an integer: " + x));
```

Pula wątków

```
Flux.just(5, 10, 15)
    .filter(x -> x > 5)
    .map(x -> x + 1)
    .subscribeOn(Schedulers.parallel())
    .subscribe(x -> logger.info("Got an integer: " + x));
```

Czy wszystko napisane w Rx\* jest **asynchroniczne**?

Czy wszystko napisane w Rx\* jest **asynchroniczne**?

# Nie!

Eager: żądanie od razu, niezależnie od subskrypcji

```
Mono.just(httpGet(endpoint))  
    .subscribe(x -> logger.info("Got a response: " + x));
```

Lazy: żądanie w momencie subskrypcji

```
Mono.fromCallable(() -> httpGet(endpoint))  
    .subscribe(x -> logger.info("Got a response: " + x));
```

Jeśli bardzo chcemy, możemy pisać **blokujący**,  
**synchroniczny** i **niezrównoleglony** kod korzystając z  
Reactive Extensions (lub Reactive Streams).

# Cold stream (cold observable)

- Nie opublikują żadnych danych dopóki się ich bardzo grzecznie nie poprosi. Same z siebie nigdy nie zrobią niczego.
- Każda subskrypcja dostanie te same wartości i będą to wszystkie wartości, jakie były do wyprodukowania.

## Przykłady użycia

- Odczyt plików
- żądania HTTP do zewnętrznych serwisów,
- nakładanie chronologicznych zmian (np. migracja bazy).

## Cold stream (cold observable)

```
const source =  
  interval(1000)  
    .pipe(take(5));  
  
source  
  .subscribe(val => logFirst(val))  
  
setTimeout( () => {  
  source  
    .subscribe(val => logSecond(val))  
}, 2000)
```

[first] 0

[first] 1

[first] 2

[second] 0

[first] 3

[second] 1

[first] 4

[second] 2

[second] 3

[second] 4

# Hot stream (hot observable)

- Produkuje wartości niezależnie od tego, czy ktoś je czyta, czy nie.
- Nowe subskrypcje dostają tylko te wartości, które powstały od chwili utworzenia subskrypcji
- Producent nie jest tworzony od zera przy każdej subskrypcji

## Przykłady użycia

- Websockety

## Hot stream (hot observable)

```
const source =  
  interval(1000)  
    .pipe(  
      take(5),  
      publish()  
    );  
  
source  
  .subscribe(val => logFirst(val))  
  
setTimeout( () => {  
  source  
    .subscribe(val => logSecond(val))  
}, 3000)
```

[first] 0

[first] 1

[first] 2

[second] 2

[first] 3

[second] 3

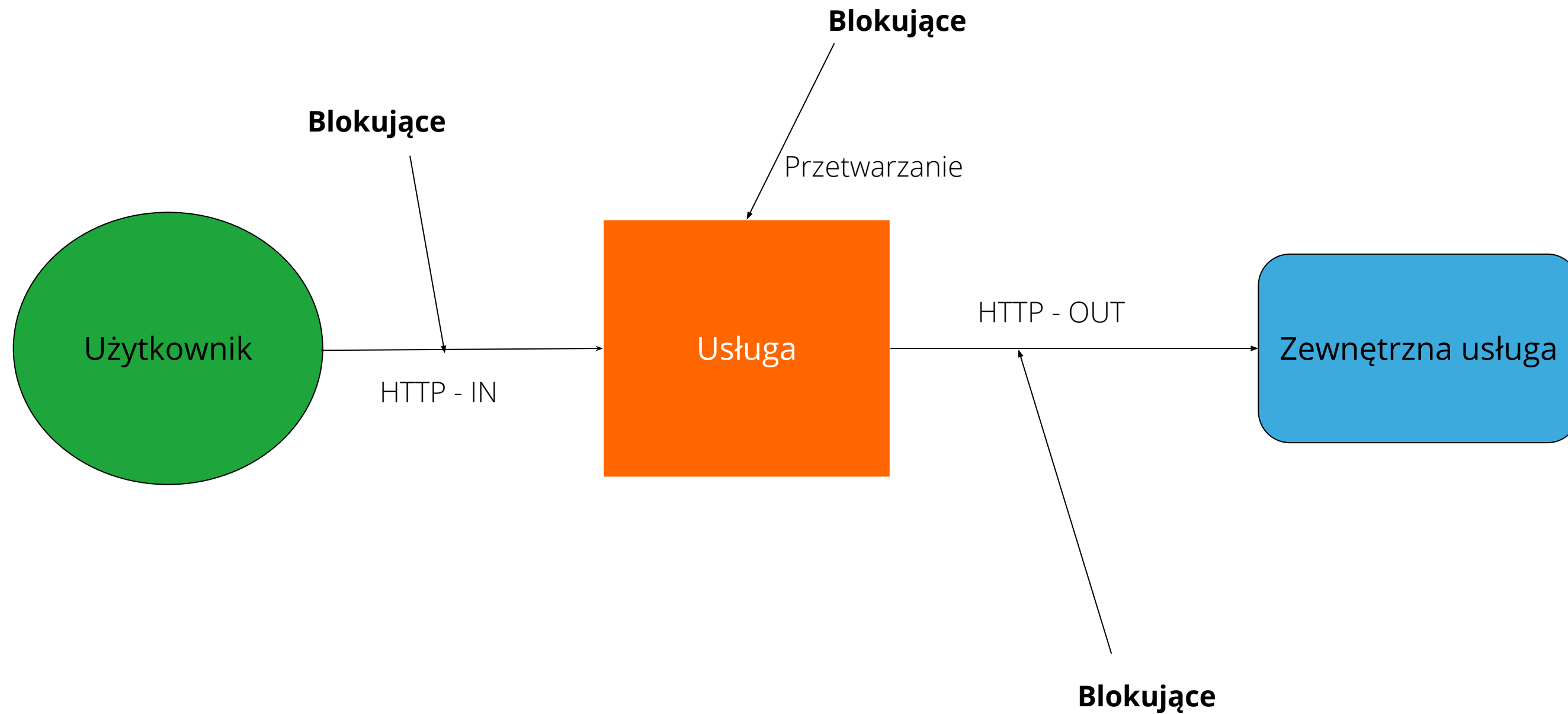
[first] 4

[second] 4

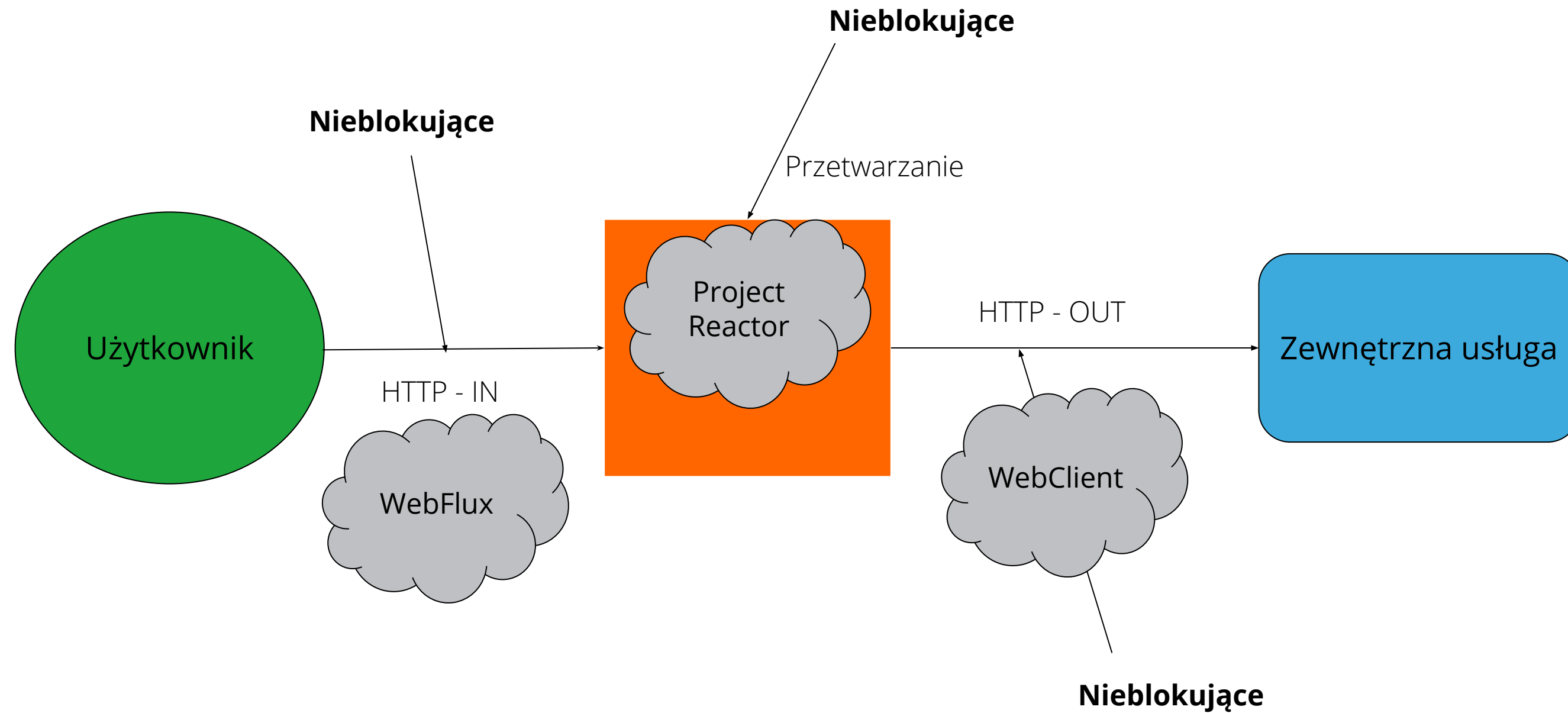


# Aplikacje reaktywne w praktyce

# Architektura blokująca



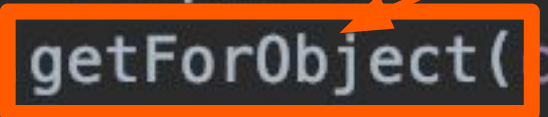
# Architektura nieblokująca



# WebClient - reaktywny klient HTTP

## RestTemplate

```
Collection<Apartment> response =  
    restTemplate.getForObject(config.getUrl(),  
        ApartmentCollection.class).getApartments();
```



Blokujący

Nie musimy sami tworzyć strumienia

## WebClient

(domyślnie Netty)

```
Mono<Collection<Apartment>> map = webClient  
    .get()  
    .uri(URI.create(config.getUrl()))  
    .retrieve()  
    .bodyToMono(ApartmentCollection.class)  
    .map(ApartmentCollection::getApartments);
```



Nieblokujący

## WebFlux - reaktywne MVC

## Tomcat (Servlet)

```
@GetMapping  
private ApartmentCollection getApartments() {  
    return apartmentService.getApartments().block();  
}
```

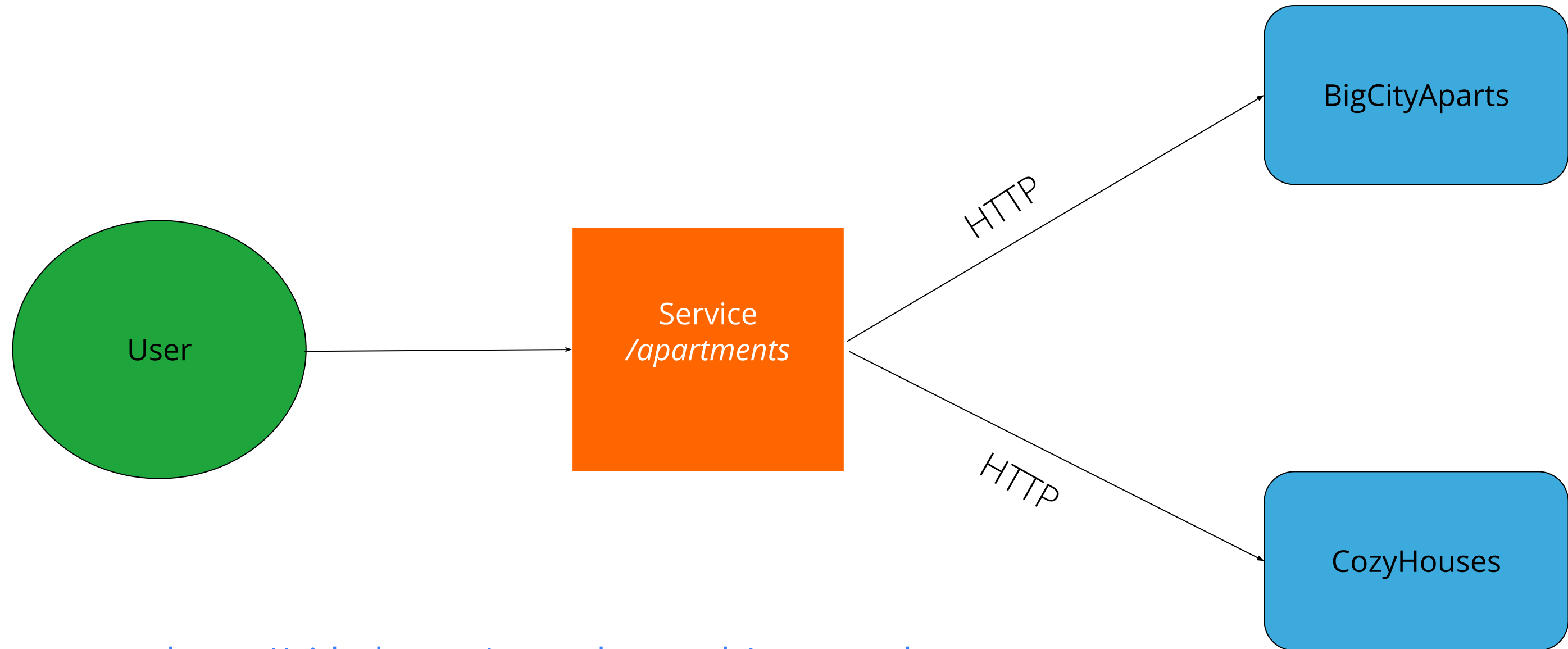
Blokujący

## Netty (reactive)

```
@GetMapping  
private Mono<ApartmentCollection> getApartments() {  
    return apartmentService.getApartments();  
}
```

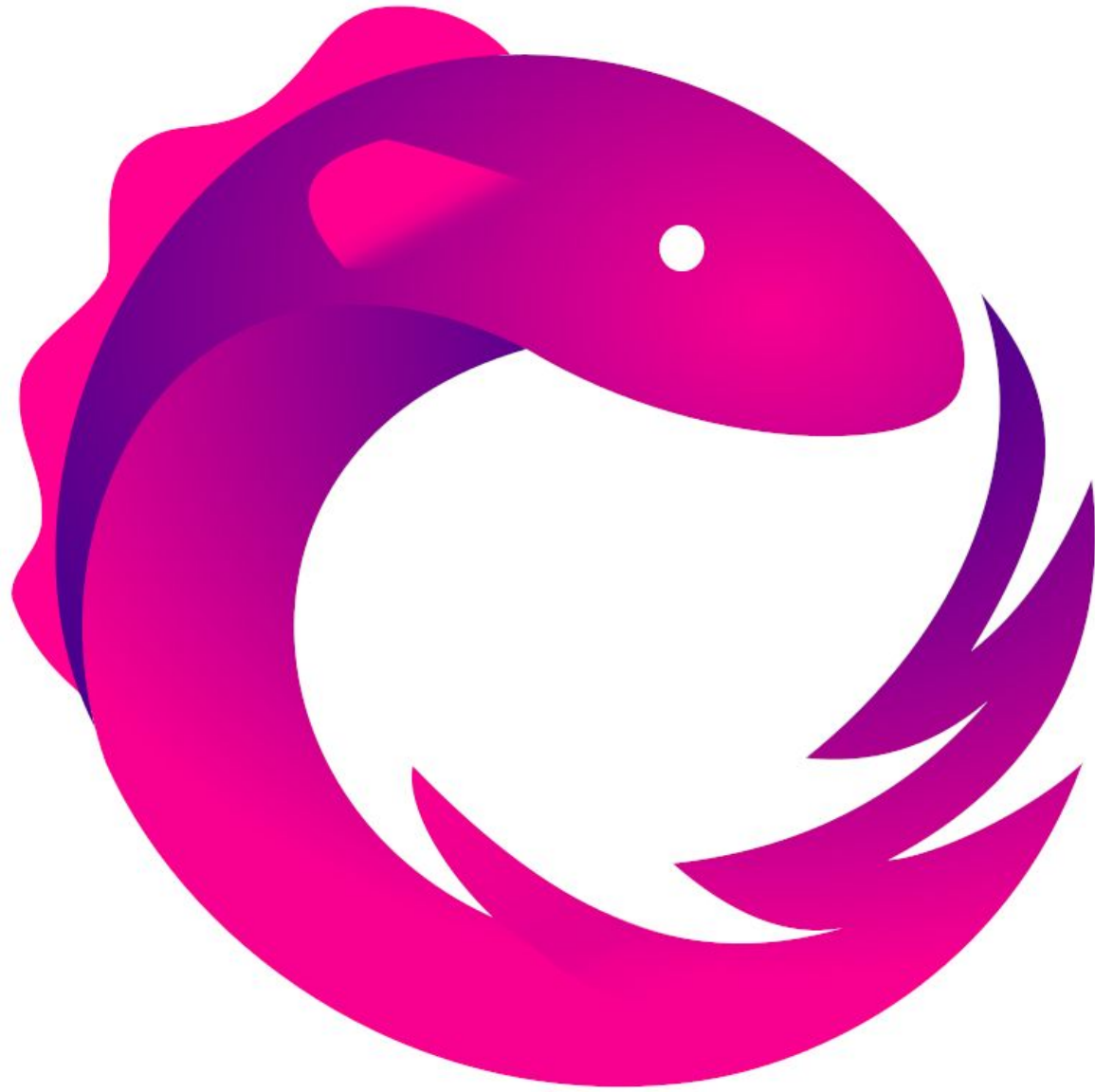
Nieblokujący

## Demo



[https://github.com/mareckmareck/reactor\\_demo](https://github.com/mareckmareck/reactor_demo)

## Przykład z życia (RxJS)





## Przykład z życia (RxJS)

```
private getCartOrdersCountByToken(token: string) {  
    return this.getOrderByToken(token)  
        .pipe(switchMap( project: (order) => this.getCartOrdersCount(order)))  
        .subscribe( next: response => {  
            this.order.cartOrdersCount = response.body.cartOrdersCount;  
        });  
}
```



## Przykład z życia (RxJS)

```
private getCartOrdersCountByToken(token: string) {  
    return this.getOrderByToken(token)  
        .pipe(switchMap( project: (order) => this.getCartOrdersCount(order)))  
        .subscribe( next: response => {  
            this.order.cartOrdersCount = response.body.cartOrdersCount;  
        });  
}
```

```
getCartOrdersCount(token: string): Observable<HttpResponse<OrderDetails>> {  
    const headers = {...};  
  
    return this.http.get<OrderDetails>(  
        url: this.config.get().getApiUrl() + 'orders',  
        {headers: headers...});  
}
```

## Przykład z życia (Project Reactor)



## Przykład z życia (Project Reactor)

```
public Mono<Void> processOrders(Collection<Order> orders) {  
    return Flux.fromIterable(orders)  
        .doOnNext(this::process)  
        .doOnNext(o -> addReminderStatus(o, SENT))  
        .doOnError(this::handleError)  
        .doOnComplete(() -> logProcessed(orders))  
        .doFinally(this::logAudit)  
        .subscribeOn(scheduler)  
        .then();  
}
```



## Przykład z życia - Hermes (Project Reactor)

```
@Override
public Mono<HermesResponse> send(Uri uri, HermesMessage message) {
    return webClient.post() WebClient.RequestBodyUriSpec
        .uri(uri) WebClient.RequestBodySpec
        .syncBody(message.getBody()) WebClient.RequestHeadersSpec<capture of ?>
        .headers(httpHeaders -> httpHeaders.setAll(message.getHeaders())) capture of ?
        .exchange() Mono<ClientResponse>
        .flatMap(response -> handleResponse(message, response));
}

private Mono<HermesResponse> handleResponse(HermesMessage message, ClientResponse response) {
    return response
        .bodyToMono(String.class)
        .switchIfEmpty(NO_BODY)
        .map(body -> buildHermesResponse(message, response, body));
}

private HermesResponse buildHermesResponse(HermesMessage message, ClientResponse response, String body) {
    return hermesResponse(message)
        .withBody(body)
        .withHttpStatus(response.rawStatusCode())
        .withHeaderSupplier(header ->
            convertToCaseInsensitiveMap(response.headers().asHttpHeaders().toSingleValueMap())
            .get(header))
        .build();
}
```

## Przykład z życia - what the... (Project Reactor)

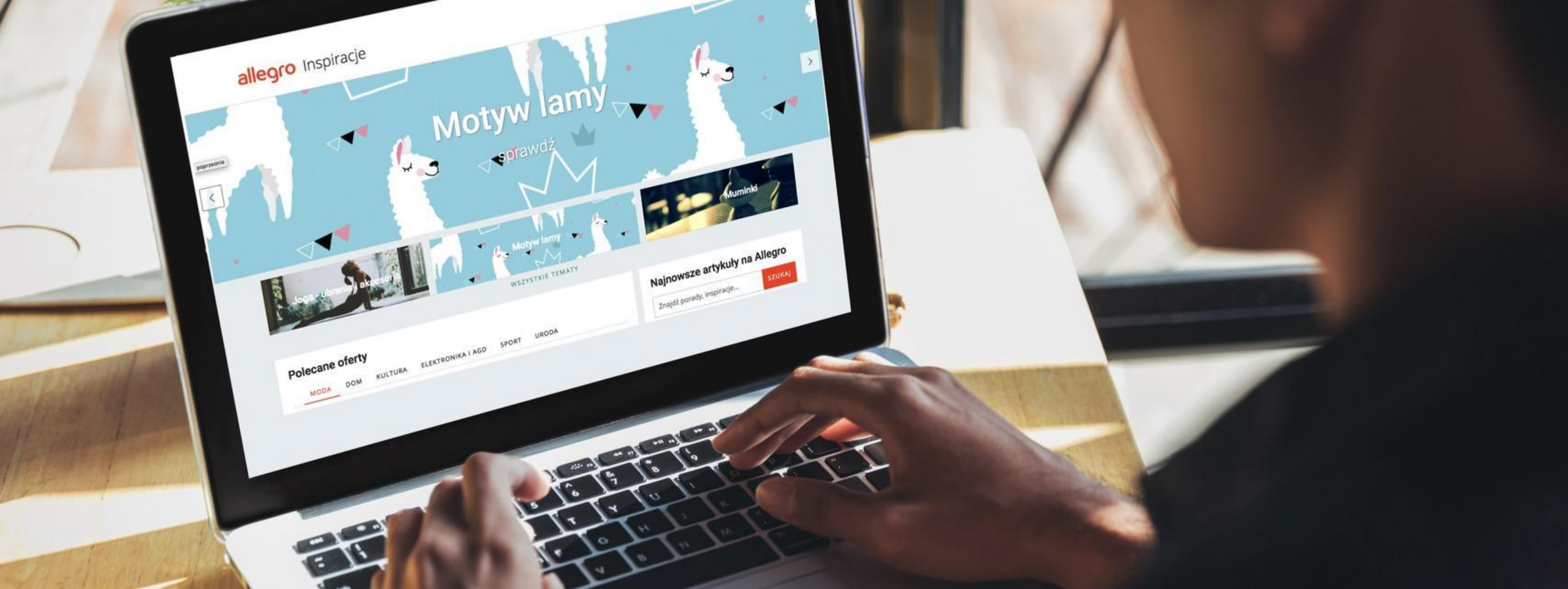
```
Flux<UserEntity> recountUserEntities(EntityRequest entityRequest) {
    String userId = entityRequest.getUserId();
    Country country = entityRequest.getCountry();
    LocalDateTime gracePeriodApplied = currentTimeSupplier.get().minusDays(entityProperties.getGracePeriod());

    return Mono.justOrEmpty(entityRepository.findFirstByUserId(userId))
        .switchIfEmpty(Mono.error(new EntityNotFoundException(userId, country)))
        .map(Entity::getUserId)
        .flatMapMany(uId ->
            Flux.zip(Mono.just(entityRequest.getUserId())
                .flatMapMany(opId ->
                    Flux.zip(getMonthlyAggregatedRates(entityRequest),
                        counterService.countByUserIdAndFlagged(uId, country, gracePeriodApplied),
                        counterService.countByUserIdAndNotFlagged(uId, country, gracePeriodApplied),
                        counterService.countByUserIdAndCountryFlagged(uId, country),
                        counterService.countByUserIdAndCountryNotFlagged(uId, country)))
                .map(UserEntityAggregateMapper::map)
                .map(UserEntityMapper::composeEntity)
                .subscribeOn(scheduler),
                    recountAggregateEntity(entityRequest))
                .map(entity -> composeFull(entity, uId, country))
                .subscribeOn(scheduler)
            );
        );
}
```

# Wady programowania reaktywnego

- kod jest znacznie trudniejszy w pisaniu/czytaniu niż w podejściu imperatywnym,
- debugowanie Rx\* jest zadaniem na całe dnie,
- bardzo łatwo zapomnieć np. o ustawieniu puli wątków czy poprawnej obsłudze błędów,
- niezliczone kruczki i pułapki,
- kod jest dużo cięższy do testowania niż przy programowaniu imperatywnym.





Dzięki za uwagę! Pytania?

**allegro**