KONKURENTNÉ PROGRAMOVANIE

Cvičenie 11 : Reaktívne programovanie 2

Iné generátory dát

- knižnice pre MongoDB, Cassandra, Redis, Couchbase, relačné DB (R2DBC)
  - zasielanie hodnôt zo select-ov
  - získanie odozvy ukončenia insert-ov, update-ov a ďalších operácií
- WebFlux
  - reaktívne REST API cez eventsQueue
    - Vlákna nespia, kým dôjdu dáta z pezistentnej vrstvy
  - reaktívny WebSocket
- WebClient
  - reaktívny HTTP klient
- Processory (Sinks)
  - Keď generovanie dát je závislé od externých udalostí

Prúdové operátory

- Zmena prichádzajúcich dátových objektov
- Odchytávanie udalosti uprostred prúdu
  - Prichádzajúce dáta a výnimky sú preposielané ďalej
- Delenie a spájanie viacerých prúdov
- Synchrónne vrátenie dát
- Práca s časom

Mapovanie elementov

- map(x -> zmeň(x))
  - Vezme dátový objekt a na základe neho vráti zmenený, alebo úplne iný objekt
- timestamp()
  - Vezme objekt x a vráti Tuple.of(long_timestamp, x)
- index()
  - prevezme i-ty objekt x z prúdu a vráti Tuple.of(i, x)

Filtrovanie elementov

- filter(x -> podmienka(x))
  - Vezme dátový objekt x
    - Ak podmienka vráti true, pošle ho ďalej, inak ho zahodí
- ignoreElements()
  - Zahodí všetky objekty pustí len chyby a complete
- takeLast()
  - Pustí iba posledný objekt pred udalosťou complete
- elementAt(n)
  - Pustí iba n-tý objekt prúdu a zavrie prúd
  - pošle IndexOutOfBoundsException, ak prúd je kratší
- single()
  - Ak príde jediný objekt, pustí ho
  - Ak nepríde objekt a prúd sa uzavrie, pošle NoSuchElementException
  - Ak príde druhý objekt, pošle IndexOutOfBoundsException

Filtrovanie elementov 2

- take(n)
  - Pustí maximálne n objektov, potom zavrie prúd
- skip(n)
  - Prvých n objektov zahodí, ostatné objekty púšťa ďalej
- takeLast(n)
  - Po zatvorení vstupného prúdu pošle posledných n objektov
- skipLast(n)
  - Po prijatí (n+1). objektu posiela prvý, po (n+2). posiela druhý, ... tak, aby posledných n neposlal
- take(Duration)
  - Púšťa objekty pokiaľ neubehne daný časový interval od otvorenia prúdu, potom zavrie prúd
- skip(Duration)
  - Daný časový interval od otvorenia prúdu zahodí všetko a potom už objekty púšťa

Filtrovanie elementov 3

- takeUntil(x -> podmienka(x))
  - Pokiaľ platí podmienka objekty púšťa, ak vráti false, zavrie prúd
- skipUntil(x -> podmienka(x))
  - Pokiaľ neplatí podmienka objekty nepúšťa, po prvom true, už púšťa všetky
- takeUntilOther(Publisher)
  - Púšťa objekty pokiaľ nepríde prvý objekt z druhého prúdu, potom zavrie svoj prúd
- skipUntilOther(Publisher)
  - Zahadzuje objekty pokiaľ nepríde prvý objekt z druhého prúdu, potom už objekty púšťa

Filtrovanie elementov

- distinct()
  - Pošle ďalej iba elementy, ktoré sa v prúde ešte nevyskytli (pozor na pamäť)
  - 1,1,2,2,2,3,1,2,2,2,3,3,4 -> 1,2,3,4
- distinctUntilChanged()
  - Z každej podpostupnosti rovnakých objektov pošle iba prvý
  - 1,1,2,2,2,3,1,2,2,2,3,3,4 -> 1,2,3,1,2,3,4

Analýza prúdu

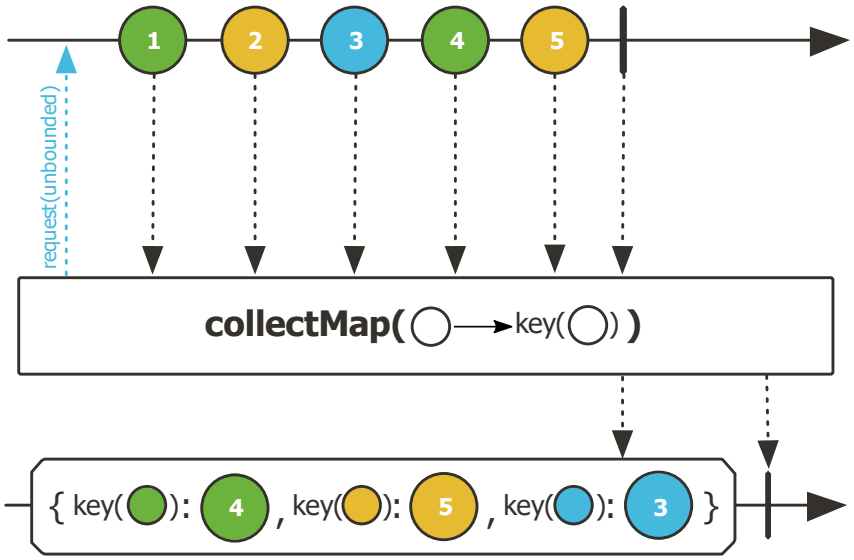
- Všetky tieto metódy vracajú Mono
- count()
  - pošle počtet objektov v predchádzajúcom prúde po jeho zatvorení
- all(x -> podmienka(x))
  - Pošle true ak všetky prvky splnili podmienku
  - Po prvom, čo nesplnil zatvorí vstupný prúd a pošle false
- any(x -> podmienka(x))
  - Pošle false ak všetky prvky nesplnili podmienku
  - Po prvom, čo splnil zatvorí vstupný prúd a pošle true
- hasElements()
  - Pošle true, ak vstup pošle prvý objekt a zatvorí vstupný prúd
  - Ak sa vstupný prúd zavrie bez prvku, posiela sa false
- hasElement(e)
  - Pošle true, ak príde rovnaký objekt a zatvorí vstupný prúd
  - Ak sa vstupný prúd zavrie, posiela sa false

Kombinovanie vstupných hodnôt

- sort() a sort(comparator)
  - Po uzavretí vstupného prúdu, generuje nový výstupný prúd, kde sa posielajú prijaté hodnoty už usporiadané
- reduce(init_acc, (acc, elem) -> new_acc)
  - Pre každý element zo vstupného prúdu zavolá druhú funkciu, ktorá vracia nový medzivýsledok acc
  - Prvý parameter je iniciálna hodnota medzivýsledku acc
  - Výsledkom je Mono s poslednou vrátenou hodnotou medzivýsledku
- reduce((acc, elem) -> new_acc)
  - iniciálna hodnota medzivýsledku je prvý element, vnútorná funkcia sa spúšťa až pre druhý element vstupného prúdu
- scan(init_acc, (acc, elem) -> new_acc) a scan((acc, elem) -> new_acc)
  - Ako reduce, len výsledkom je Flux so všetkými medzivýsledkami

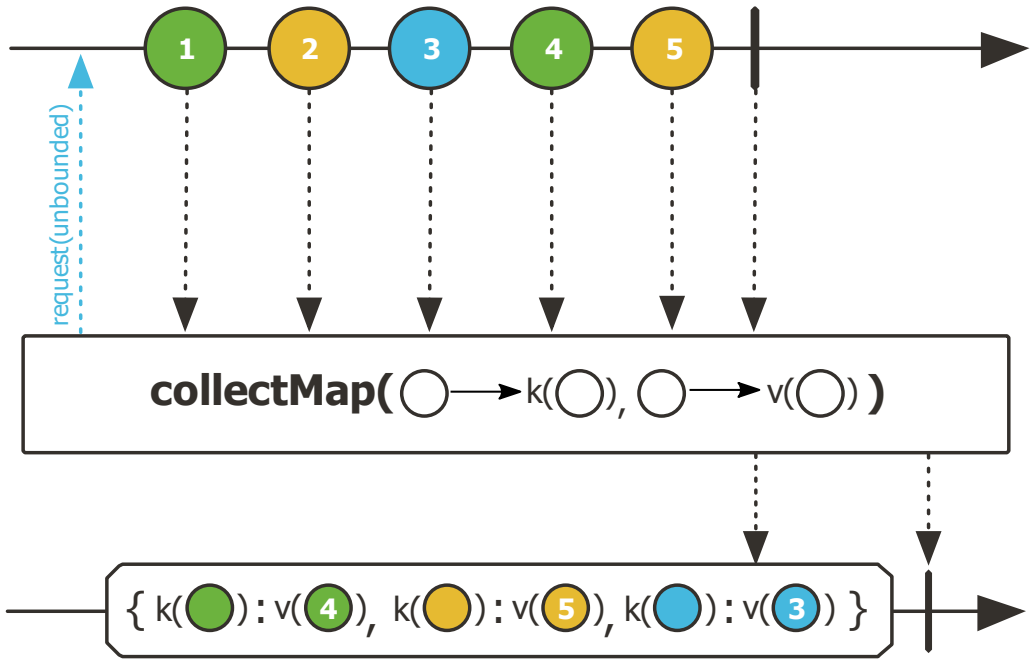
Vyzbieranie hodnôt do kolekcie

- Meníme Flux na Mono
  - Keď sa Flux zavrie, operátor posiela jediný objekt – kolekciu s prijatými hodnotami do výstupného prúdu
  - Keď príde chyba,
- collectMap(x ->vráťKľúč(x))
  - Výsledná mapa priradí každému kľúču posledný objekt, pre ktorý sme tento kľúč vrátili



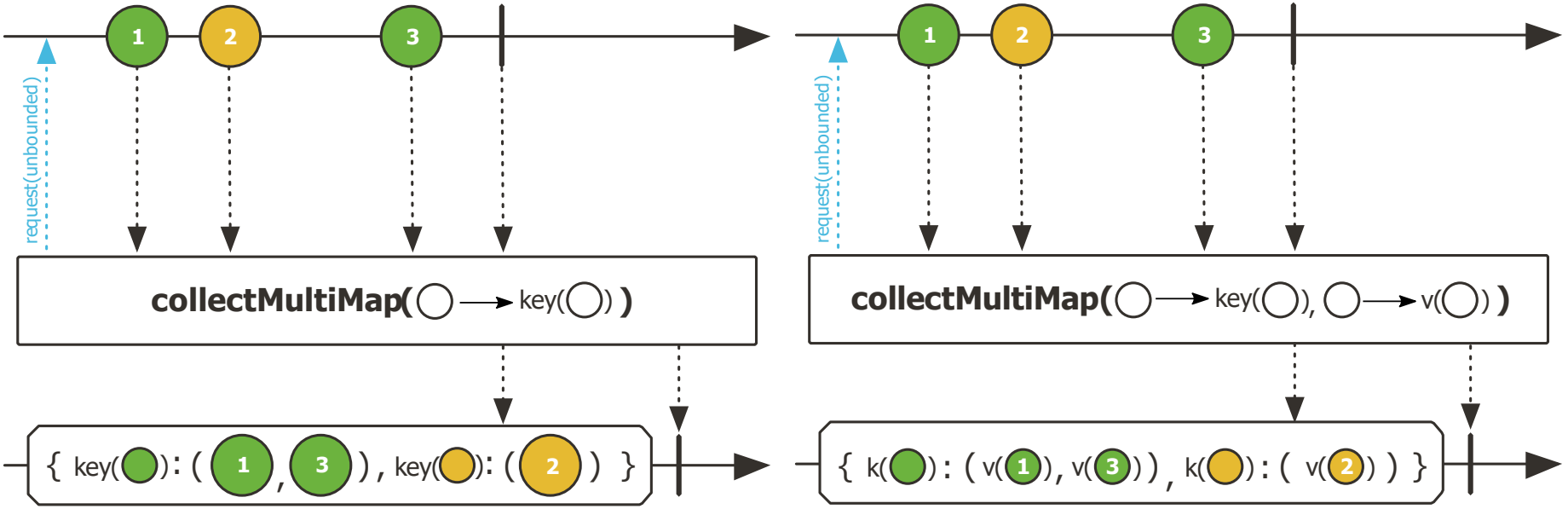
Vyzbieranie hodnôt do kolekcie 2

- collectMap(x ->vráťKľúč(x), x->vráťHodnotu(x))
  - Výsledná mapa priradí každému kľúču poslednú hodnotu z objektu, pre ktorý sme tento kľúč vrátili



Vyzbieranie hodnôt do kolekcie 3

- collectMultimap(x ->vráťKľúč(x))
- collectMultimap(x ->vráťKľúč(x), x->vráťHodnotu(x))
  - Výsledná mapa priradí každému kľúču kolekciu hodnôt z objektov, pre ktoré sme tento kľúč vrátili



Vyzbieranie hodnôt do kolekcie 4

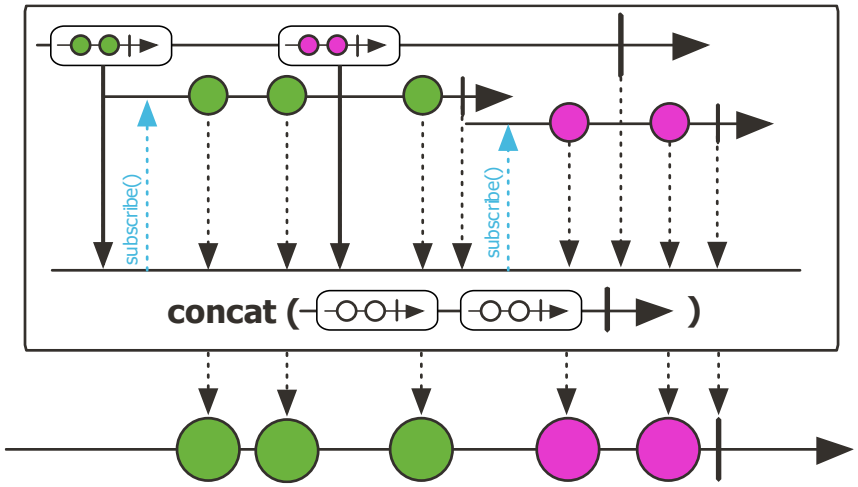
- collectList()
  - Pošle hodnoty ako zoznam
- collectSortedList() a collectSortedList(Comparator)
  - Pošle usporiadaný zoznam prijatých hodnôt
- collect(Collector)
  - Použije na vytvorenie kolekcie ľubovoľný kolektor implementujúci java.util.stream.Collector
- collect(() ->init_acc, (acc, value) -> {})
  - Pošle object acc, keď sa vstupný prúd zavrie
  - Predpokladá sa, že v druhej funkcii pridávame value do acc

Napájanie prúdov

- Tieto operátory ignorujú dáta zo vstupných prúdov, len čakajú, kedy sa zatvoria, aby spustili ďalšie
  - Flux.then(Mono)
  - Flux.thenMany(Publisher)
  - Flux.thenEmpty(Publisher)
  - Mono.then(Mono)
  - Mono.thenMany(Publisher)
  - Mono.thenEmpty(Publisher)
  - Mono.thenReturn(value)
    - Vráti Mono s danou hodnotou

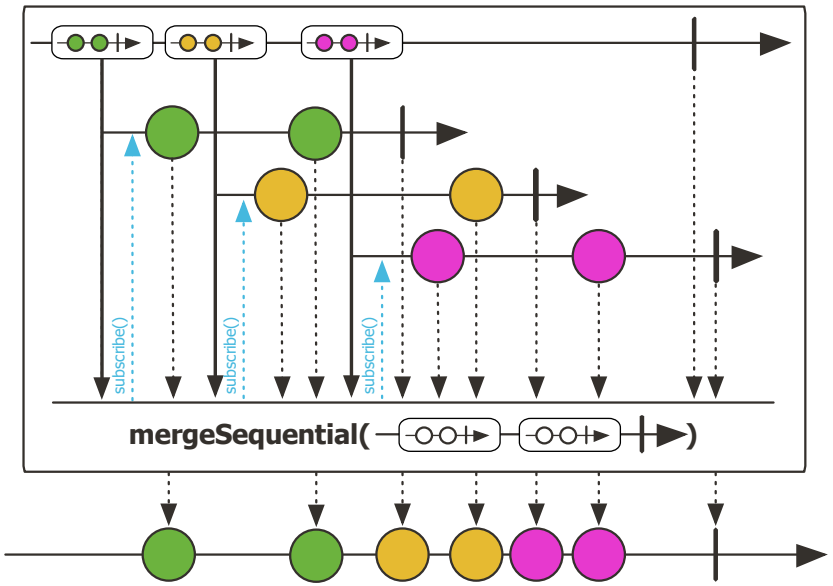
Spájanie prúdov

- Flux.concat(listPublisherov) a Flux.concat(publisher1, publisher2, ...)
  - Postupne robí subscribe na každý kanál, keď sa i-ty uzavrie, spustí (i+1).kanál
  - Čo posielajú vnútorné prúdy, hneď preposielam do výstupného prúdu
- Flux.concat(PublisherPublisherov)
  - Keď pritečie i-ty kanál, urobím na ňom subscribe a preposielam dáta z neho na výstup. Keď prúd i-teho kanála uzavrie, tak urobím subscribe na (i+1). kanáli čo pritiekol po ňom, alebo naňho "počkám".



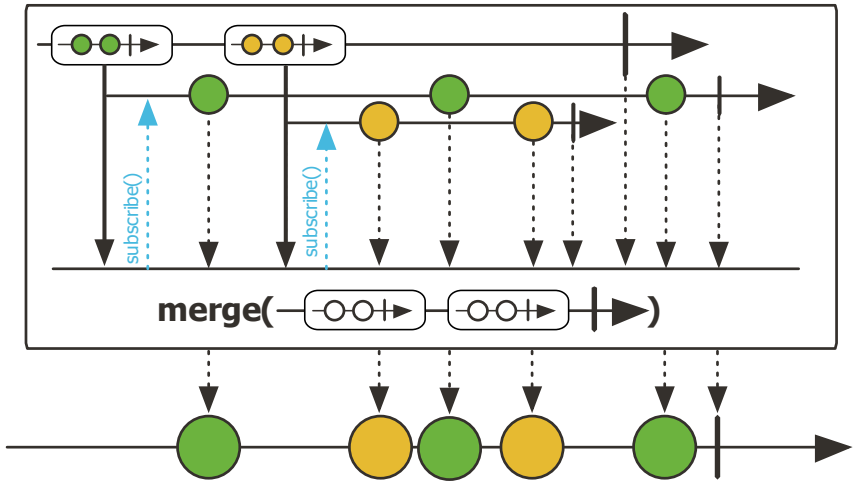
Spájanie prúdov

- Flux.mergeSequential(listPublisherov) a Flux.mergeSequential(publisher1, publisher2, ...)
  - Spraví subscribe na každý kanál, zavrie, keď všetky vstupné zavrú
  - Dáta z i-teho prúdu pošle až keď (i-1). prúd zavrie
- Flux.mergeSequential(PublisherPublisherov)
  - Urobím subscribe na každý kanál, hneď keď príde
  - Dáta z i-teho prúdu pošle až keď (i-1). prúd zavrie



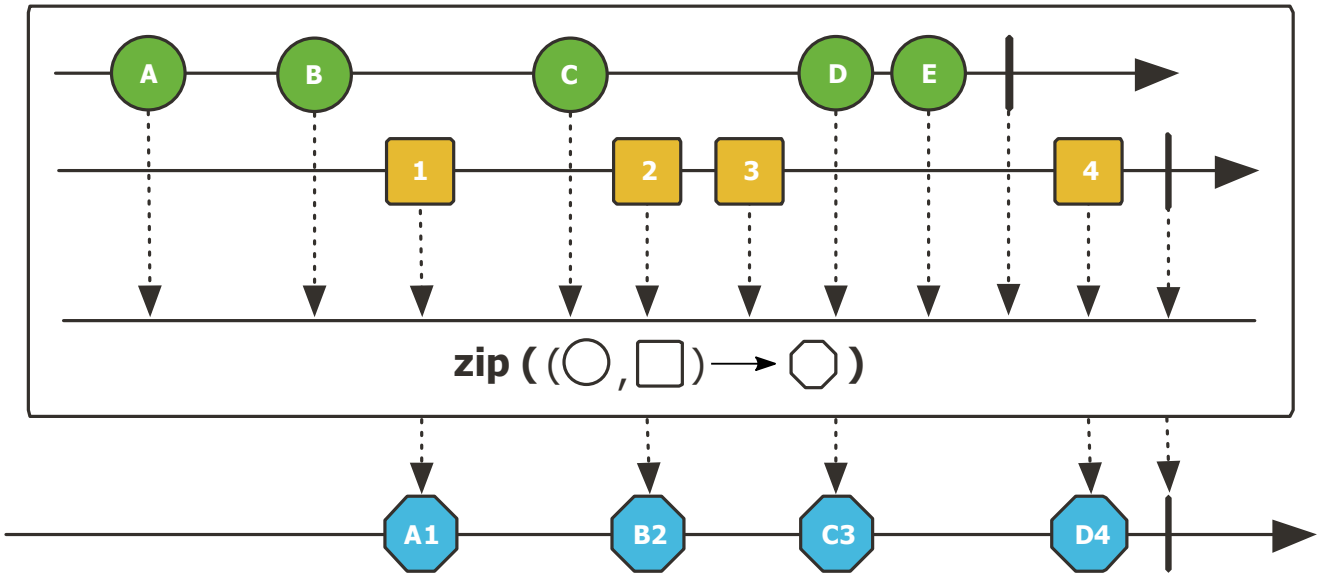
Spájanie prúdov

- Flux.merge(listPublisherov) a Flux.merge(publisher1, publisher2, ...)
  - Spraví subscribe na každý kanál, zavrie, keď všetky vstupné zavrú
  - Čo posielajú vnútorné prúdy, hneď preposielam do výstupného prúdu
- Flux.merge(PublisherPublisherov)
  - Urobím subscribe na každý kanál, hneď keď príde
  - Čo posielajú vnútorné prúdy, hneď preposielam do výstupného prúdu



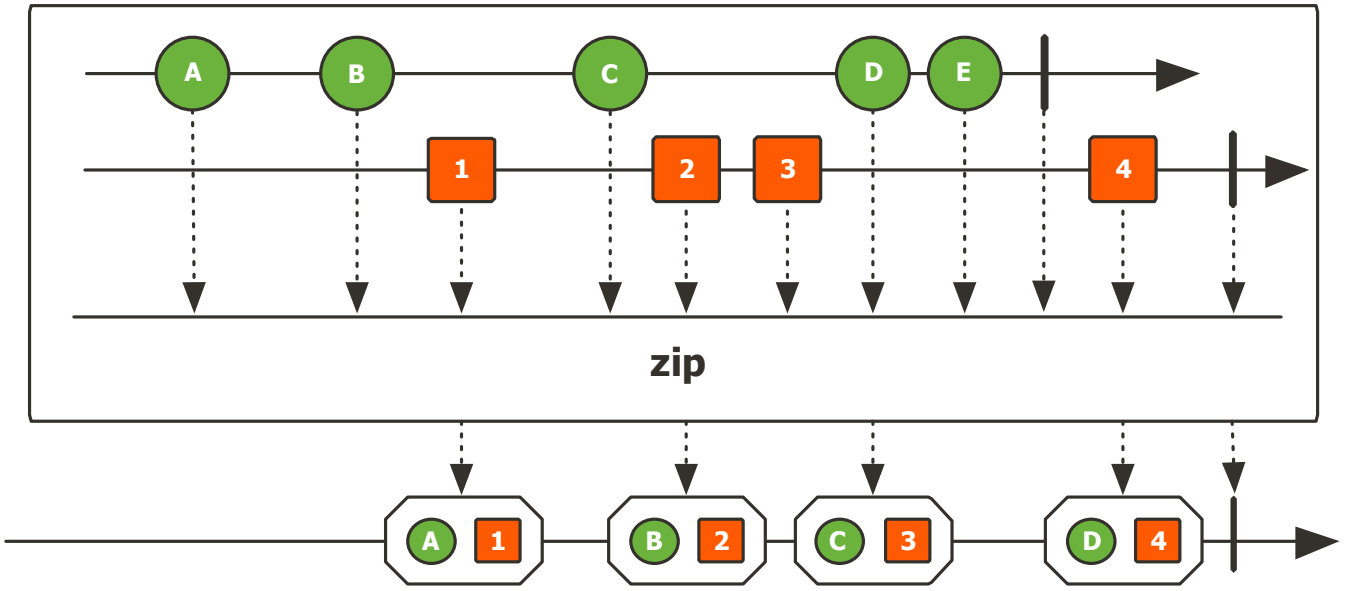
Spájanie prúdov

- Flux.zip(listPublisherov, ([v1,v2,..]) -> v)
  - Spraví subscribe na každý kanál, keď prídú i-te objekty zo vstupných prúdov, pošle na výstup výsledok vnútornej funkcie



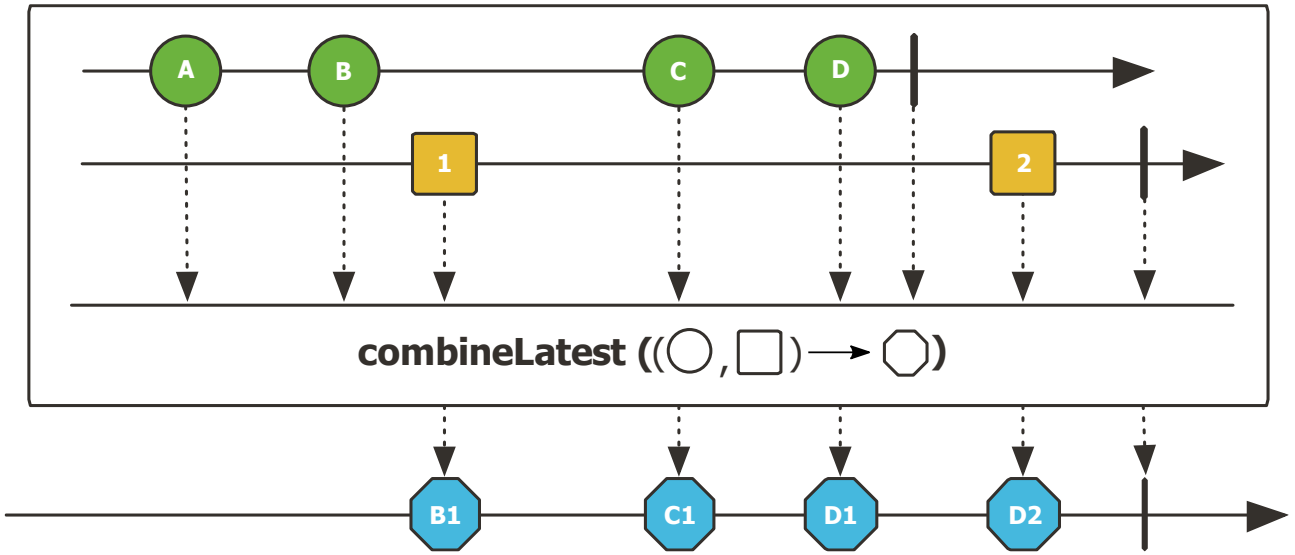
Spájanie prúdov

- Flux.zip(publisher1, publisher2, ...)
  - Spraví subscribe na každý kanál, keď prídú i-te objekty zo vstupných prúdov, pošle na výstup objekt typu Tuple



Spájanie prúdov

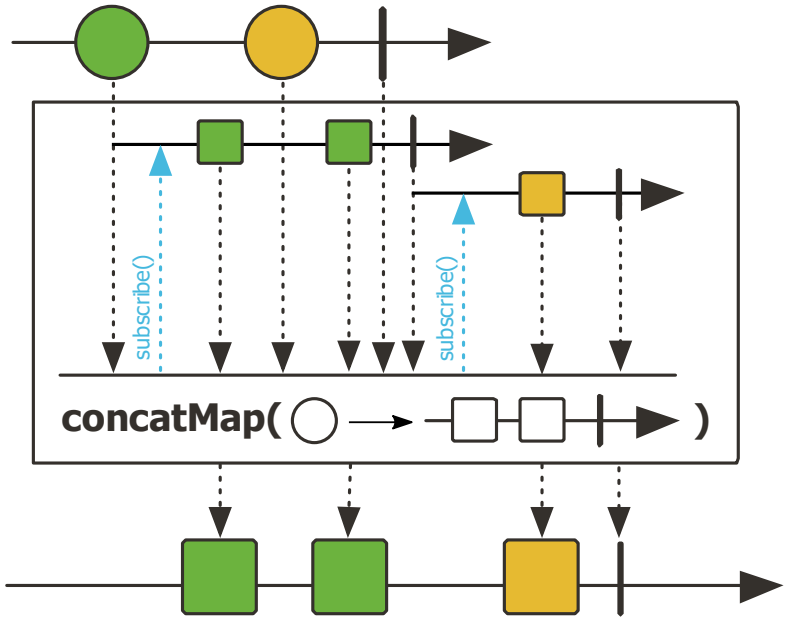
- Flux.combineLatest(listPublisherov, ([v1,v2,..]) -> v) a
- Flux.combineLatest(publisher1, publisher2,..., ([v1,v2,..]) -> v)
  - Princíp je rovnaký ako pri zip, ale posielajú sa n-tice z posledných hodnôt, hneď ako niektorý zo vstupov pošle dáta



Mapovanie cez prúdy

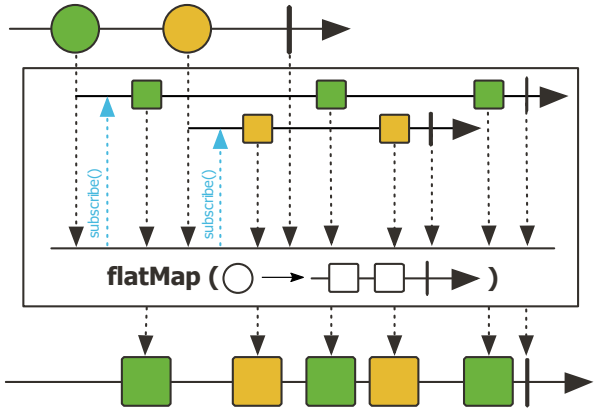
- Veľmi silný nástroj
- Vieme zaradiť do kanálu viaceré dlhotrvajúce operácie
  - Čítanie z internetu, databázy, ...
- Napr. ak vstup je prúd používateľských udalostí v GUI, po ktorých potrebujem komunikovať so serverom
  - Výsledky komunikácie môžem spracovať v mnou určenom poradí

- concatMap(value -> Publisher)
  - S i-tou vstupnou hodnotou sa vytvorí i-ty vnútorný kanál cez danú funkciu
  - i-ty vnútorný kanál sa subscribe-ne, keď (i-1)-vý vnútorný kanál zavrie
  - Na hlavný výstupný kanál sa posielajú hodnoty zasielané z vnútorných kanálov

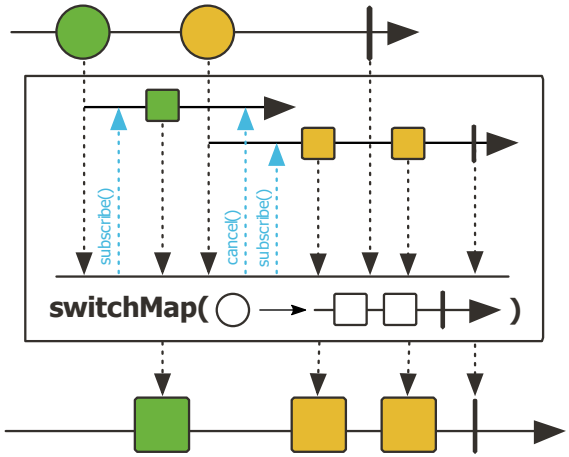


Mapovanie cez prúdy

- flatMap(value -> Publisher)
  - S i-tou vstupnou hodnotou sa vytvorí i-ty vnútorný kanál
  - i-ty vnútorný kanál sa **hneď** subscribe-ne
  - Na hlavný výstupný kanál sa posielajú hodnoty zasielané z vnútorných kanálov



- switchMap(value -> Publisher)
  - S i-tou vstupnou hodnotou sa vytvorí i-ty vnútorný kanál a zruší (i-1)-vý
  - i-ty vnútorný kanál potom subscribe-ne
  - Na hlavný výstupný kanál sa posielajú hodnoty zasielané z vnútorných kanálov



Delenie prúdov

- groupBy(value ->keyMapper(value))
  - Vracia prúd kanálov
  - Každý z odoslaných kanálov ešte treba zvlášť subscribe-núť
    - alebo zasa spojiť a subscribe-núť spojenie
  - Každý kanál, ktorý sa pošle do výstupného prúdu je typu GrouppedFlux, ktorý má oproti bežnému Flux naviac getter na kľúč
  - Hodnoty zo vstupu si vyrátajú kľúč K,
    - Hodnota je preposlaná do výstupného prúdu s kľúčom K
    - Ak taký prúd ešte nebol zaslaný vytvorí sa nový, priradí sa mu kľúč K, pošle sa doňho daná hodnota
    - Hodnoty sa reálne pošlú do prúdu kanála K, až keď je kanál K spustený
- groupBy(value ->keyMapper(value), value->valueMapper(value))
  - To isté akutát do výstupných prúdov sa posielajú nie pôvodné hodnoty ale to čo vráti valueMapper()

```
private String divisibleBy(long value) {
  List<Integer> divisors = Arrays.asList(2,3,5,7,11,13);
  for(Integer divisor: divisors) {
    if (value % divisor == 0) {
      return "Divisible by " + divisor;
    }
  }
  return "Others";
}
```

```
FibonacciFlux.generateFib().take(20).groupBy(value ->divisibleBy(value))
   .subscribe(stream -> {
     stream.collectList().subscribe(value -> {
       System.out.println("key: " + stream.key());
       System.out.println("data: " + value);
     });
   });
}
```

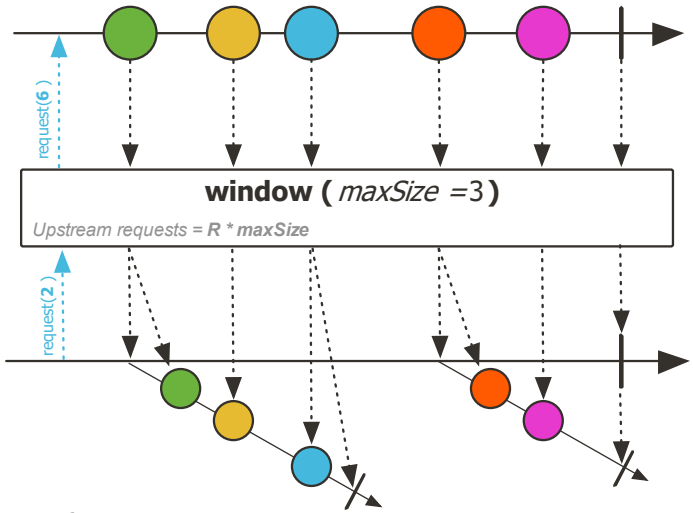
```
key: Others
data: [1, 1, 89, 233, 1597, 4181]
key: Divisible by 13
data: [13, 377]
key: Divisible by 3
data: [3, 21, 987]
key: Divisible by 2
data: [0, 2, 8, 34, 144, 610, 2584]
key: Divisible by 5
data: [5, 55]
```

Dávkové spracovanie

- buffer()
  - Po skončení vstupného prúdu vracia Flux, v ktorom pošle jeden List prijatých objektov
  - a,b,c,d,e,f,g -> [a,b,c,d,e,f,g]
- buffer(maxsize)
  - Po prijatí ďalších maxsize prvkov pošle do výstupného prúdu List s týmito prvkami. Ak sa vstupný prúd zavrie, pošle do výstupného prúdu List so zvyšnými prvkami, ktoré ešte neposlal a zavrie
  - buffer(3) pre a,b,c,d,e,f,g -> [a,b,c], [d,e,f], [g]
- buffer(maxsize, skipsize)
  - Skipsize hovorí po koľkých vstupných prvkoch začína nový List
  - buffer(2,3) pre a,b,c,d,e,f,g -> [a,b], [d,e], [g]
  - buffer(3,2) pre a,b,c,d,e,f,g -> [a,b,c], [c,d,e], [e,f,g], [g]
- bufferUntil(value –> podmienka)
  - Ukončí List, keď platí podmienka
  - bufferUntil(value -> value%2 ==0) pre 1,2,3,5,6,7,9 -> [1,2], [3,5,6], [7,9]
- bufferUntil(value –> podmienka, true)
  - Začne nový List, keď platí podmienka
  - bufferUntil(value -> value%2 ==0, true) pre 1,2,3,5,6,7,9 -> [1], [2,3,5], [6,7,9]

Delenie prúdov 2

- window(maxSize)
  - Posiela prúd prúdov
  - Po prijatí ďalších maxsize prvkov vytvorí nový kanál, do ktorého bude preposielať následné prvky
  - Každý z odoslaných kanálov ešte treba zvlášť subscribe-núť



- Analogicky k variantám buffer metód fungujú aj
  - buffer(maxsize, skipsize)
  - bufferUntil(value –> podmienka)
  - bufferUntil(value –> podmienka, true)

Odchytávanie udalosti uprostred prúdu

- doOnSubscribe(subscription -> {})
  - Na konci kanála bol spustený subscribe
- doOnRequest(count -> {})
  - Od príjemcu sú požadované dáta
- doOnNext(value ->{})
  - Cez prúd pritiekol dátový objekt, posielam ho ďalej
- doOnError(error -> {})
  - Cez prúd pritiekla výnimka, posielam ju ďalej
- doOnComplete(() -> {})
  - Cez prúd už nepôjdu žiadne dáta, zatvára sa
- doOnCancel(() -> {})
  - Príjemca už nechce žiadne dáta, posielame info odosielateľovi
- doOnEach(signal -> {})
  - Nastal subscribe, next, error alebo completion
- doOnTerminate(() –> {})
  - Nastala chyba alebo completion
- doFinally(signalType -> {})
  - Nastala chyba, completion alebo cancel

Transformácia na blokované štruktúry

- blockFirst()
  - Spraví subscribe, zapýta prvok a zaspí. Keď prvok pritečie vráti ho, zavrie prúd. Ak prúd zavrie bez prvku, vráti null.
- blockFirst(timeout)
  - Zaspí na maximálne timeout. Keď prvok pritečie vráti ho, keď nastane timeout vyhodí výnimku RuntimeException
- blockLast()
  - Subscribe-ne kanál a zaspí, kým sa prúd nezavrie. Potom sa zobudí a vráti posledný dátový objekt z prúdu, alebo v prípade prázdneho prúdu vráti null.
- blockLast(timeout)
  - Zaspí na maximálne timeout. Keď sa prúd do timeout-u nezavrie, vyhodí výnimku RuntimeException
- tolterable()
  - Automaticky sa spraví subscribe
  - S každým volaním iterator.next() sa urobí request(1) na zaslanie jedného prvku z prúdu. Pokiaľ tento prvok príde cez prúd volajúce vlákno zaspí.
- tolterable(n)
  - S každým n-tým volaním iterator.next() sa urobí request(n) na zaslanie maximálne n prvkov z prúdu. Pokiaľ chcem next() a z prúdu nič nechodí volajúce vlákno zaspí.
- toStream() a toStream(n)
  - Vyrobí stream z Java Stream API, na pozadí používa tolterable()

Práca s časom (výber)

- interval(period)
  - posiela postupne prirodzené čísla s prestávkami period
- delayElements(delay)
  - Po odoslaní každého prvku neposiela ďalší minimálne daný delay interval
- delaySequence(delay)
  - Posunie celý prúd v čase – každý prvok je preposlaný presne o delay neskôr
- delaySubscrition(delay)
- buffer(časZbierania) a
  buffer(časZbierania, novýBufferKaždých) a
  buffer(časZbierania, timer) a
  buffer(maxBufferSize, maxTime)
  - Spája dáta, ktoré prišli za ďalší časový interval
- ... podobne s window

Práca s časom (výber)

- sample(duration)
  - Po každom duration čase pošle poslednú v ňom prijatú hodnotu, alebo nič
- timeout(duration)
  - Ak cez prúd od naposledy zaslanej hodnoty ubehol daný čas, uzavrie prúd
- timeout(duration, Publisher)
  - Ak cez prúd od naposledy zaslanej hodnoty ubehol daný čas, uzavrie vstupný prúd a namiesto neho urobí subscribe na druhý parameter a začne posielať jeho hodnoty