Overview of Backpressure Models in the Project Reactor Flux

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand key classes in the Project Reactor API
- Know how Project Reactor Flux supports backpressure

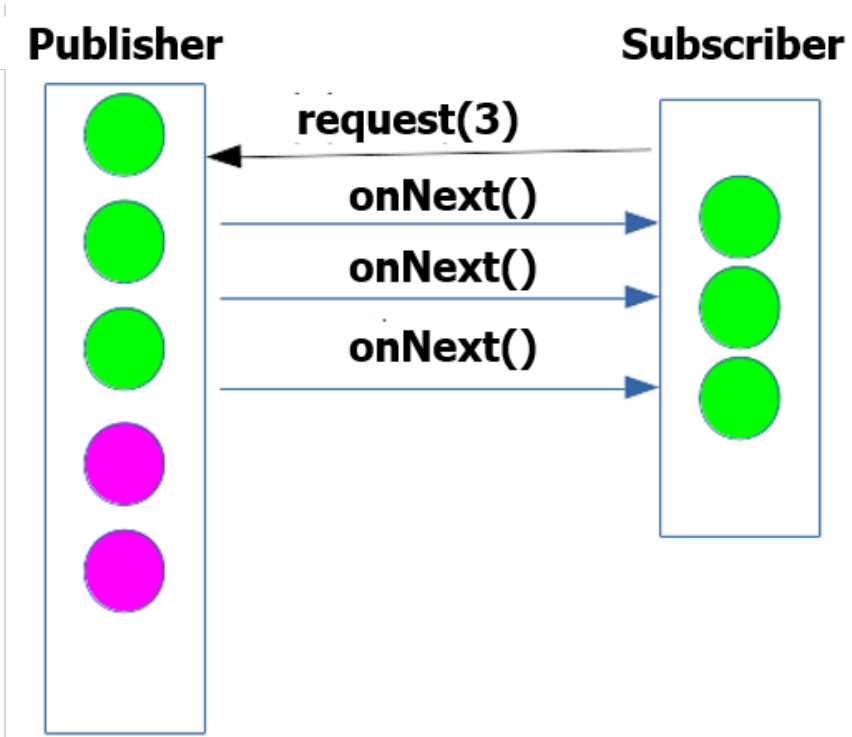
```
public static enum FluxSink.OverflowStrategy
extends Enum<FluxSink.OverflowStrategy>
```

Enumeration for backpressure handling.

Enum Constant Summary

Enum Constants

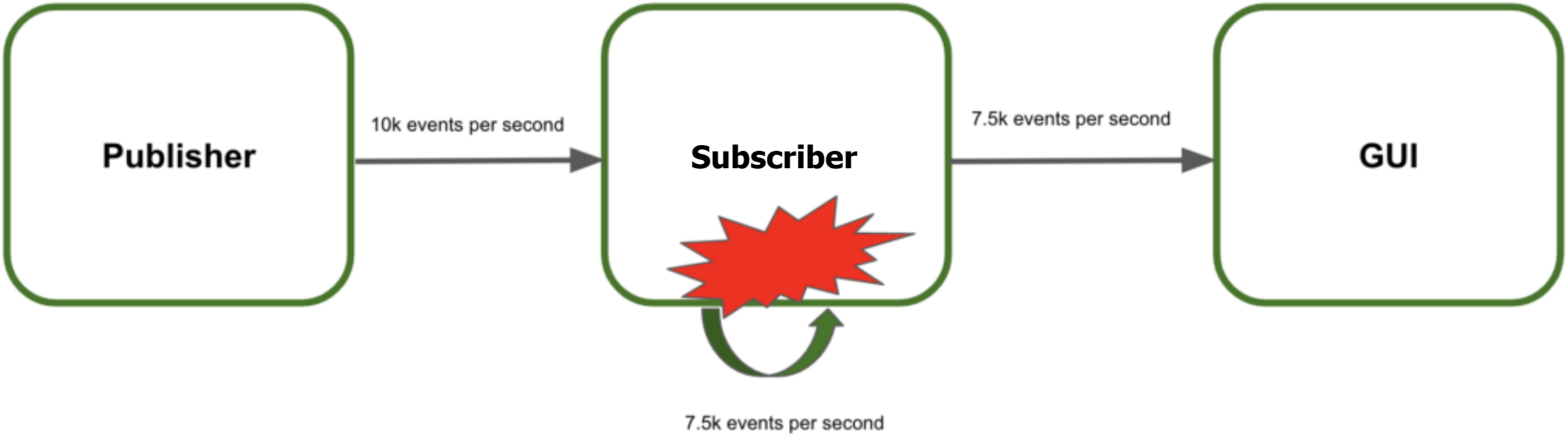
Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.



Motivation for Back pressure Mechanisms

Motivation for Backpressure Mechanisms

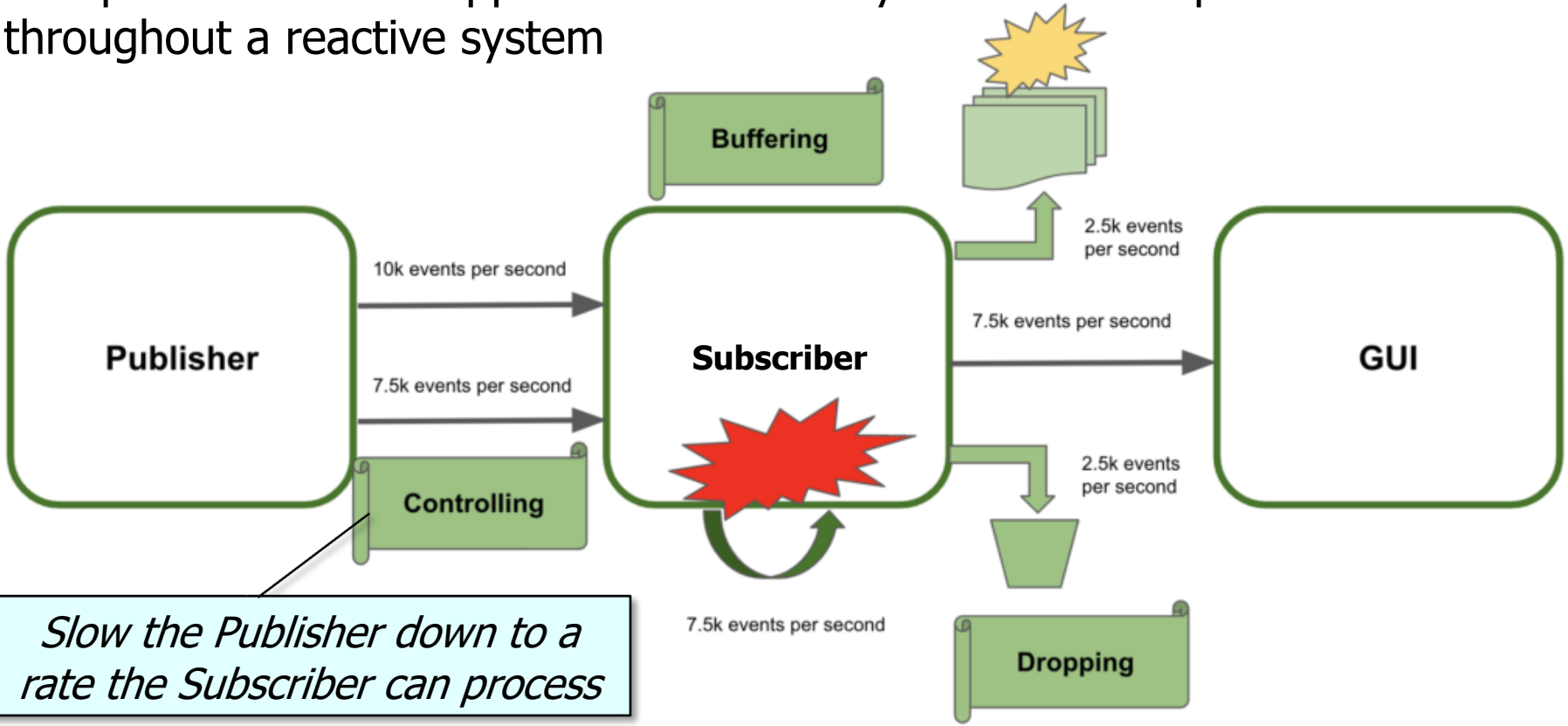
- Backpressure is needed for systems where Publisher(s) supply events faster than Subscriber(s) can consume them



See www.baeldung.com/spring-webflux-backpressure

Motivation for Backpressure Mechanisms

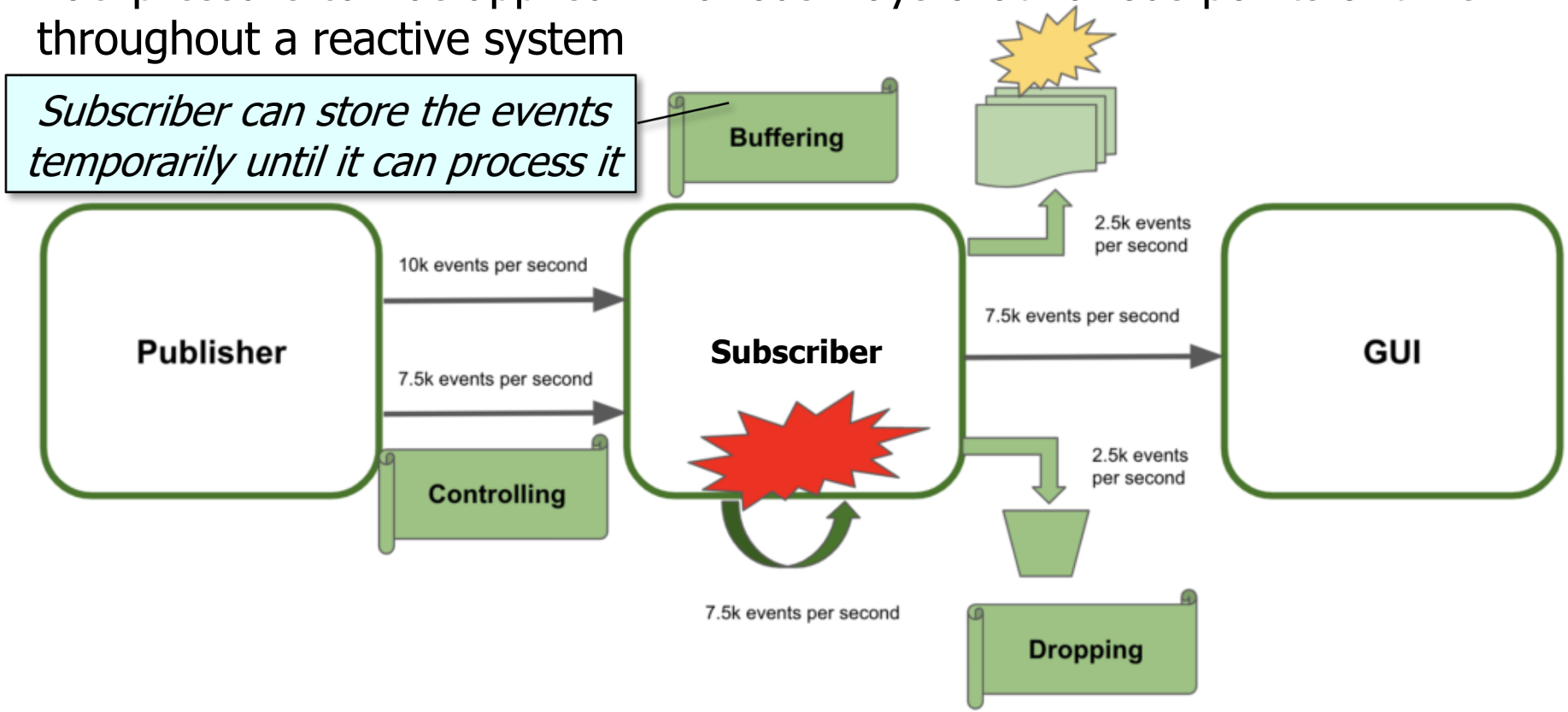
- Backpressure can be applied in various ways & at various points of time throughout a reactive system



May not always be possible, especially for cyber-physical systems

Motivation for Backpressure Mechanisms

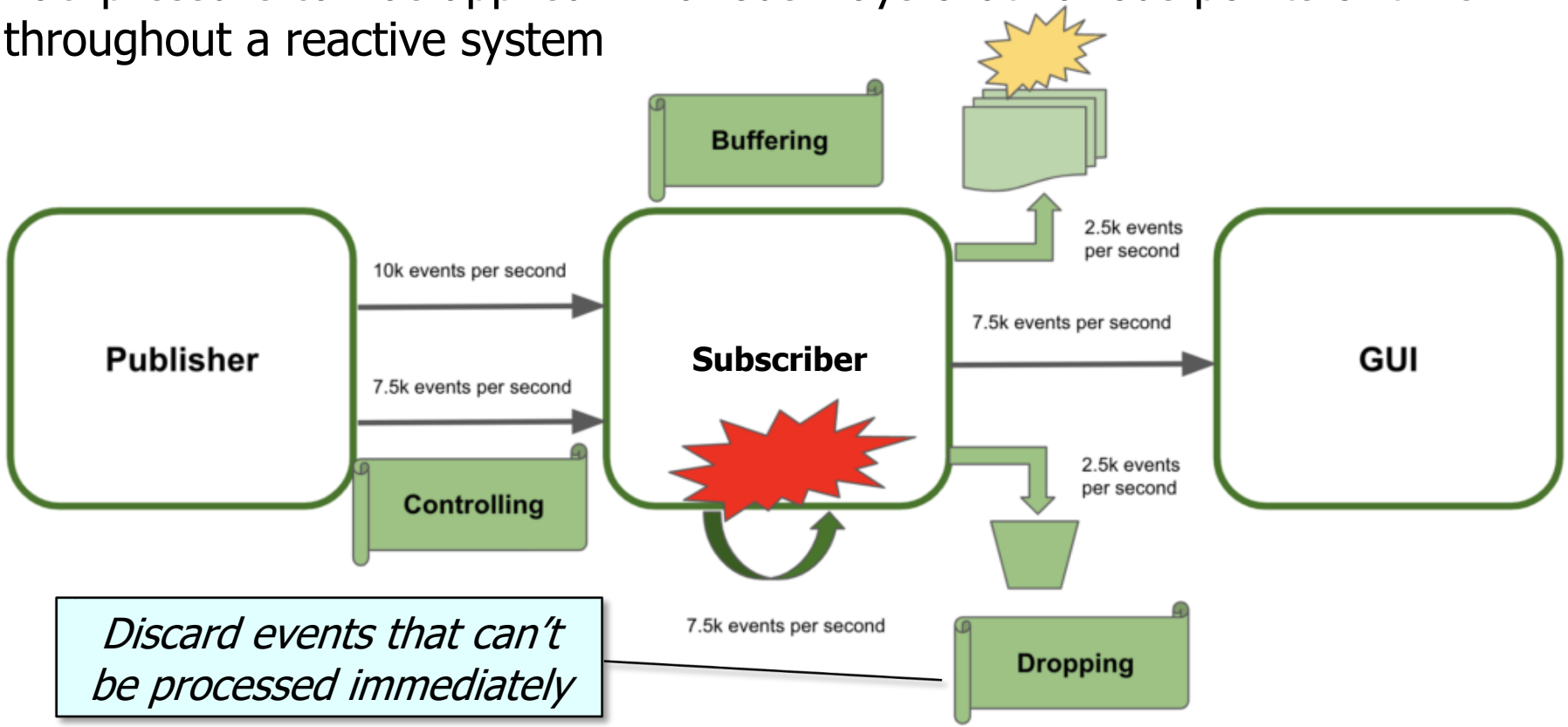
- Backpressure can be applied in various ways & at various points of time throughout a reactive system



May eventually cause "out-of-memory" exceptions!

Motivation for Backpressure Mechanisms

- Backpressure can be applied in various ways & at various points of time throughout a reactive system

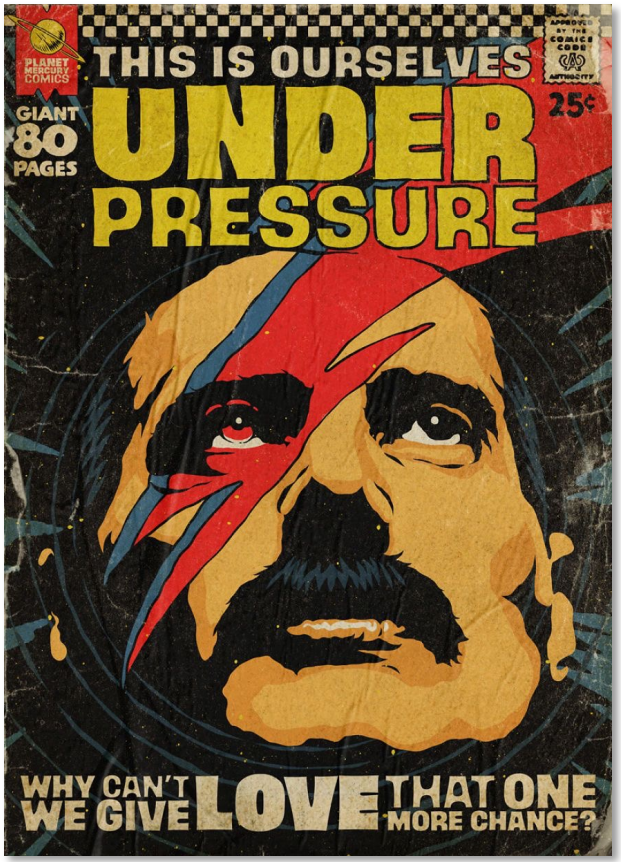


May be problematic if all events contain valuable data

Overview of Backpressure in Project Reactor Flux

Overview of Backpressure in Project Reactor Flux

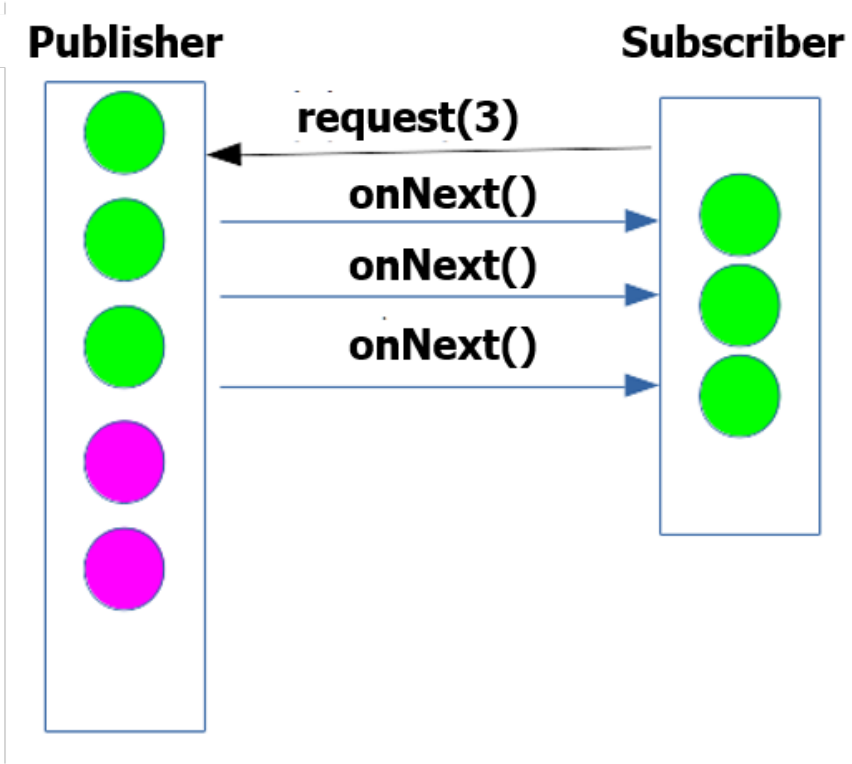
- Project Reactor Flux supports several types of backpressure



See jstobigdata.com/java/backpressure-in-project-reactor

Overview of Backpressure in Project Reactor Flux

- Project Reactor Flux supports several types of backpressure, e.g.
  - Backpressure-aware Subscriber(s) can inform Publisher(s) how much data they can consume



Overview of Backpressure in Project Reactor Flux

- Project Reactor Flux supports several types of backpressure, e.g.
  - Backpressure-aware Subscriber(s) can inform Publisher(s) how much data they can consume
    - The goal is to avoid overwhelming memory/processing resources
      - i.e., flow-control Publisher(s) so they don't generate events faster than Subscriber(s) can consume them

See www.appsdeveloperblog.com/implementing-backpressure-in-project-reactor

Overview of Backpressure in Project Reactor Flux

- Project Reactor Flux supports several types of backpressure, e.g.
  - Backpressure-aware Subscriber(s) can inform Publisher(s) how much data they can consume
    - The goal is to avoid overwhelming memory/processing resources
  - Requires Publisher(s) & Subscriber(s) to interact

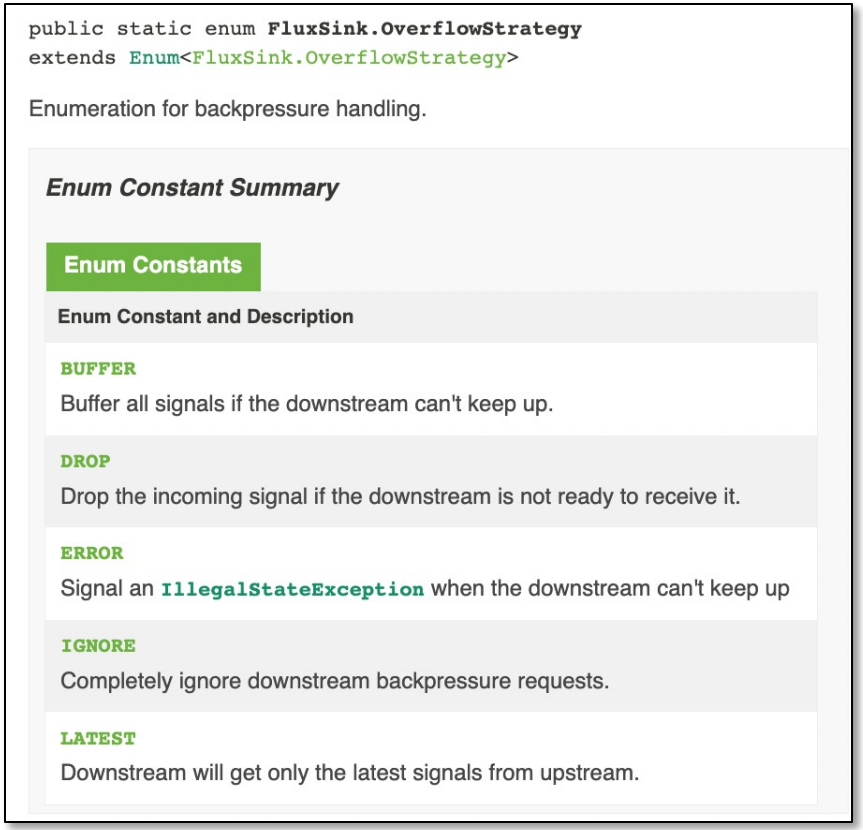
```
void onSubscribe
   (Subscription subscription){
     mSubscription =
       subscription;

     subscription
       .request(mRequestSize);
}
```

Subscriber(s) call the request() method on a Subscription passed by Publisher(s) to Subscriber(s) via the onSubscribe() hook method

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/flux/ex4

Overview of Backpressure in Project Reactor Flux

- Project Reactor Flux supports several types of backpressure, e.g.
  - Backpressure-aware Subscriber(s) can inform publisher(s) how much data they can consume
  - Non-backpressure-aware Subscriber(s) can apply an overflow strategy if they can't keep up with faster Publisher(s)



```
public static enum FluxSink.OverflowStrategy
extends Enum<FluxSink.OverflowStrategy>
```

Enumeration for backpressure handling.

Enum Constant Summary

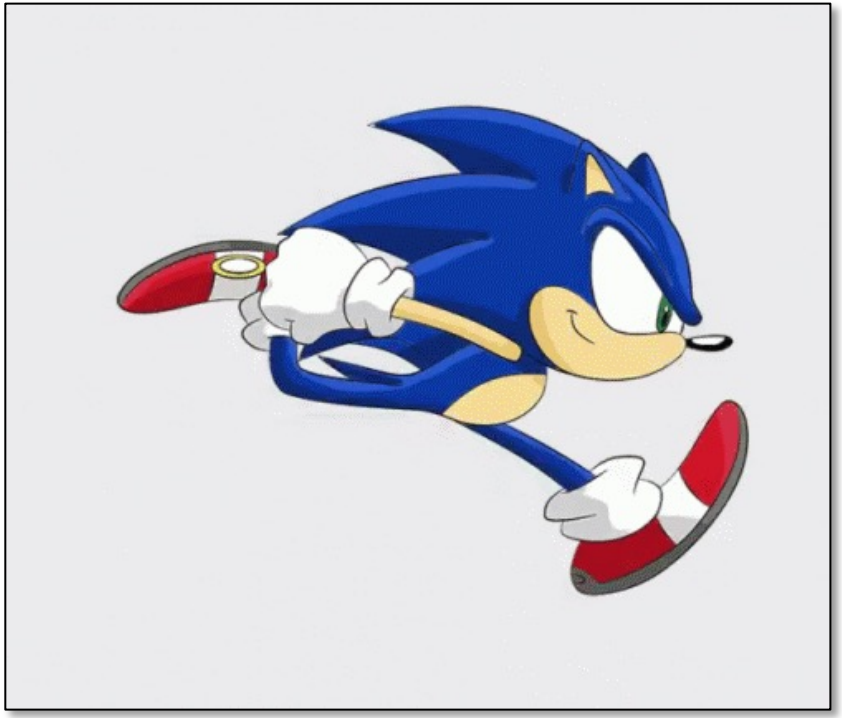
Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

See projectreactor.io/docs/core/release/api/reactor/core/publisher/FluxSink.OverflowStrategy.html

Overview of Backpressure in Project Reactor Flux

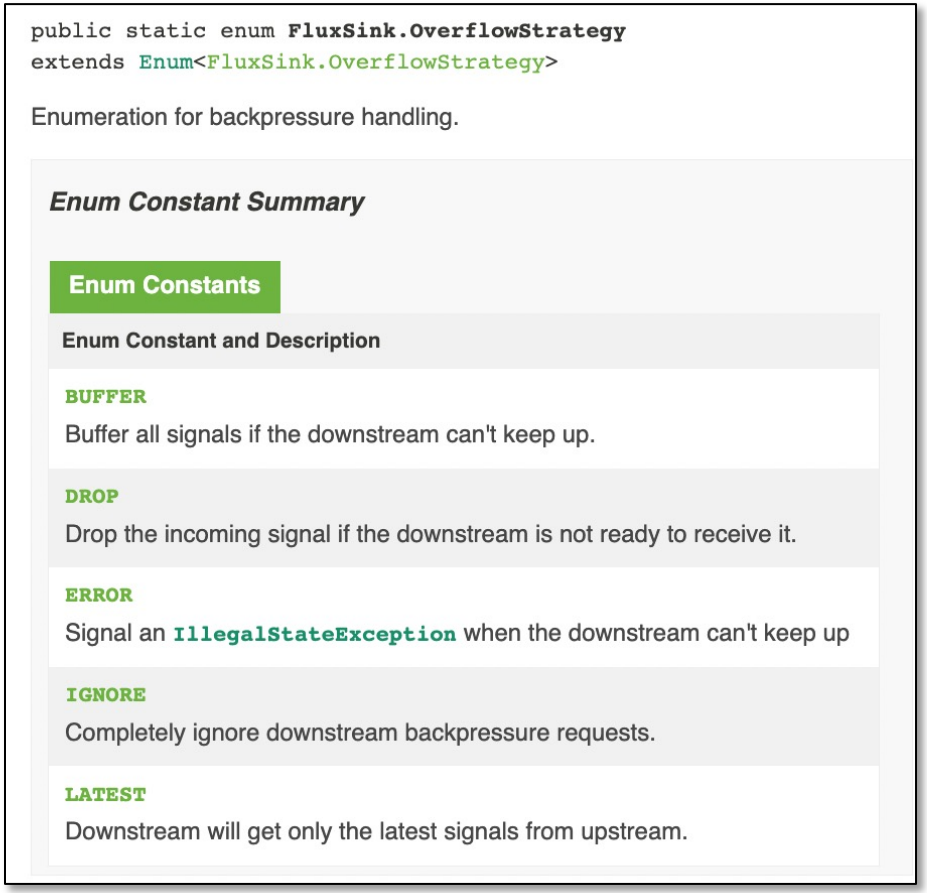
- Project Reactor Flux supports several types of backpressure, e.g.
  - Backpressure-aware Subscriber(s) can inform publisher(s) how much data they can consume
  - Non-backpressure-aware Subscriber(s) can apply an overflow strategy if they can't keep up with faster Publisher(s)
    - i.e., non-flow-controlled Publisher(s)



Overview of Flux Over flow Strategies

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received



public static enum **FluxSink.OverflowStrategy**
extends Enum<FluxSink.OverflowStrategy>

Enumeration for backpressure handling.

Enum Constant Summary

Enum Constants

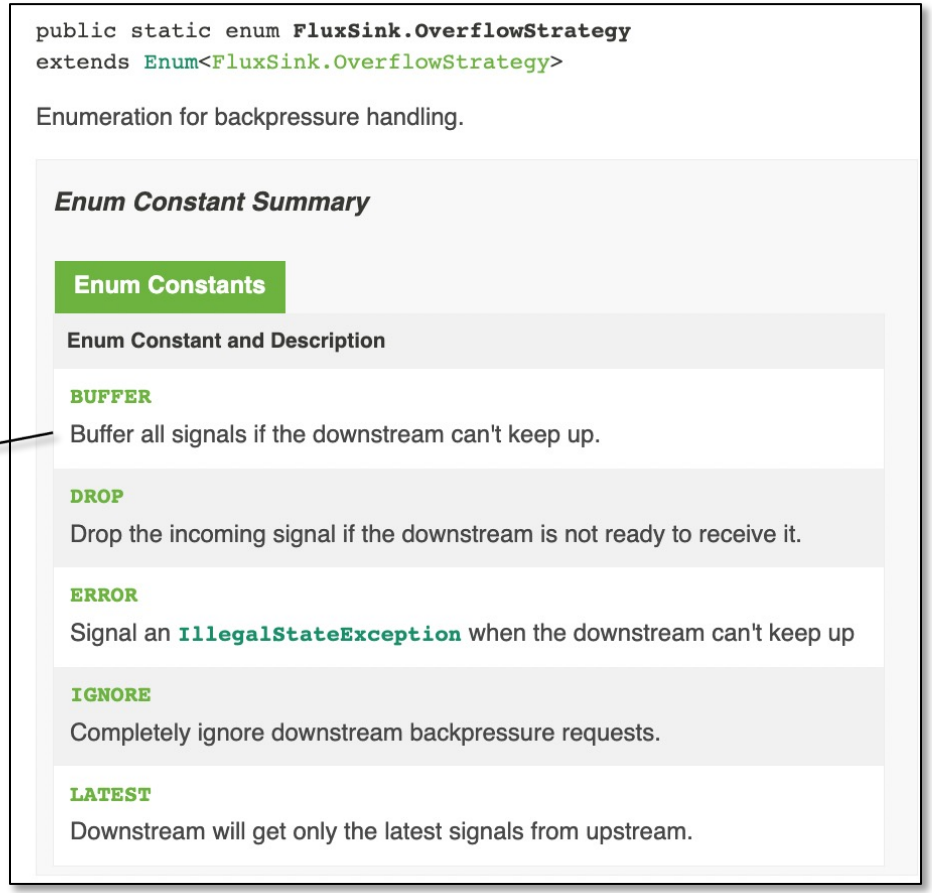
Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an **IllegalStateException** when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

See projectreactor.io/docs/core/release/api/reactor/core/publisher/FluxSink.OverflowStrategy.html

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

All values are buffered so that subscriber can receive all values



```
public static enum FluxSink.OverflowStrategy
extends Enum<FluxSink.OverflowStrategy>
```

Enumeration for backpressure handling.

Enum Constant Summary

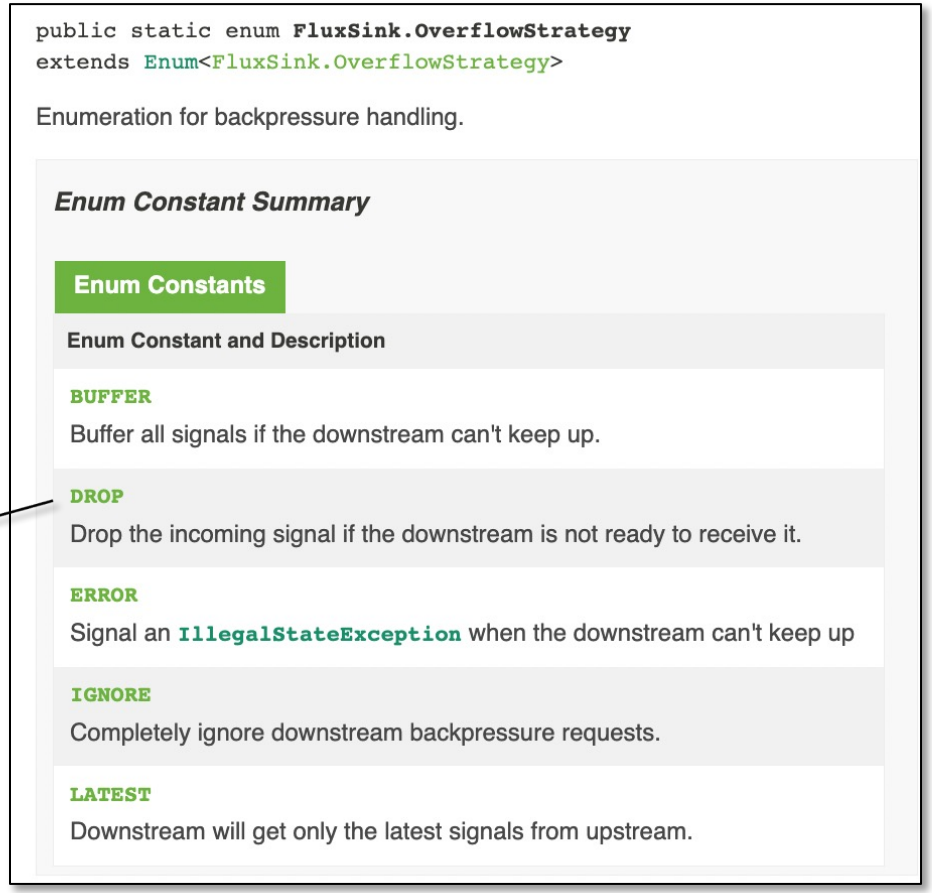
Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

Drop the most recent onNext() value if the downstream can't keep up because its too slow



public static enum **FluxSink.OverflowStrategy**
extends Enum<FluxSink.OverflowStrategy>

Enumeration for backpressure handling.

Enum Constant Summary

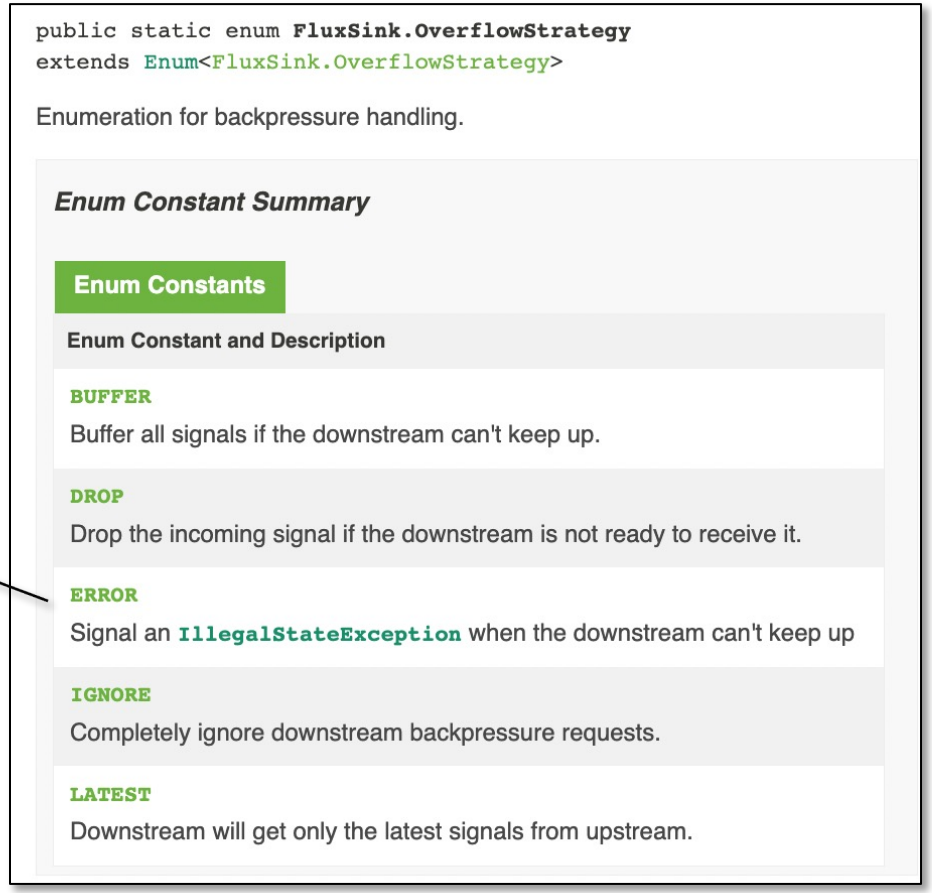
Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an **IllegalStateException** when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

Throw the OverflowException if the downstream can't keep up due to slowness



public static enum **FluxSink.OverflowStrategy**
extends Enum<FluxSink.OverflowStrategy>

Enumeration for backpressure handling.

Enum Constant Summary

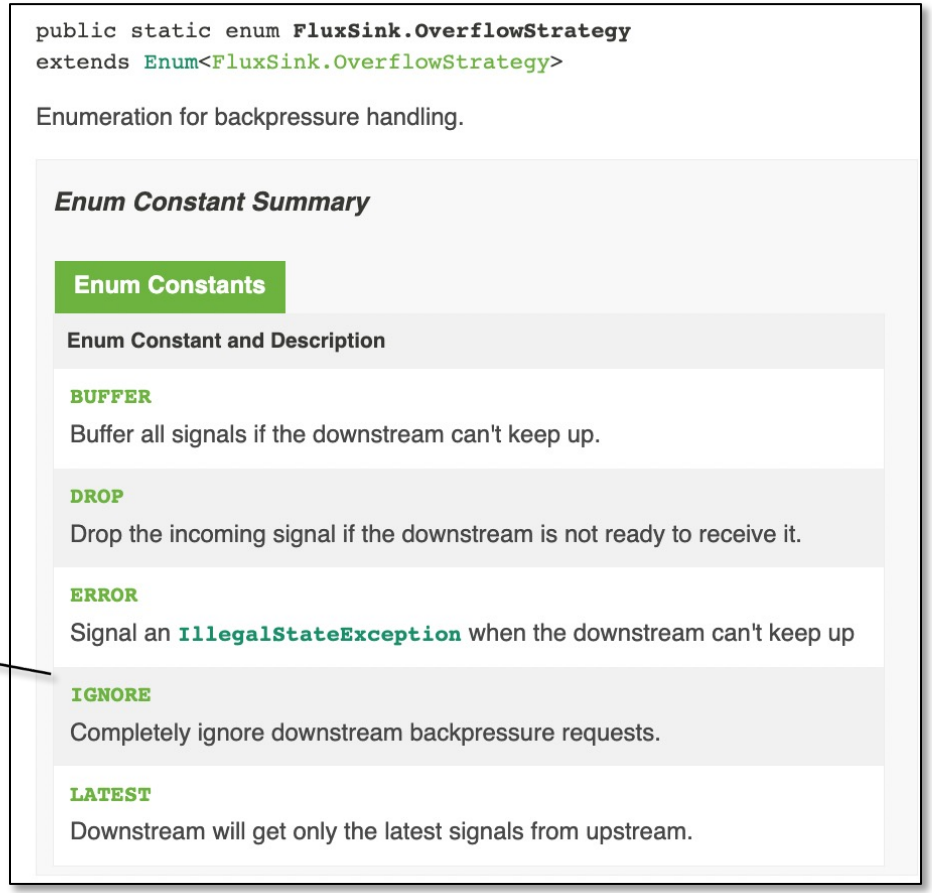
Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an **IllegalStateException** when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

There is no buffering or dropping, so Subscriber(s) must handle overflow of they will receive an error



```
public static enum FluxSink.OverflowStrategy
extends Enum<FluxSink.OverflowStrategy>
```

Enumeration for backpressure handling.

Enum Constant Summary

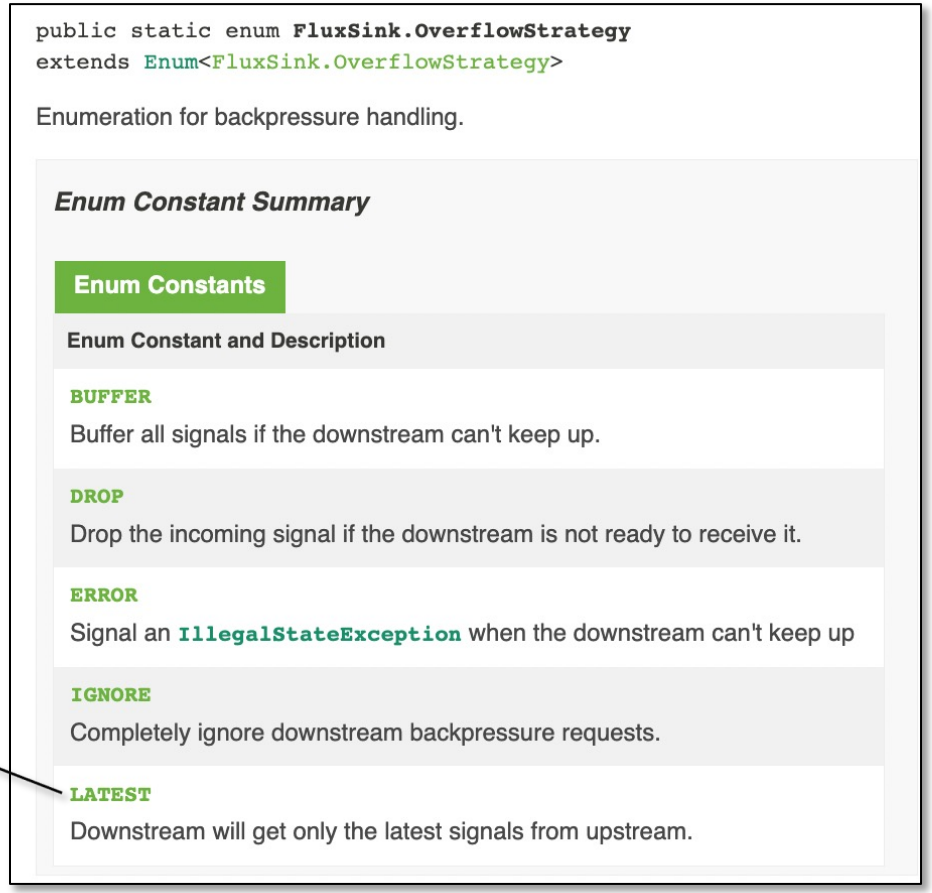
Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

Only key the latest onNext() value, overwriting any previous value if the downstream can't keep up because its too slow



```
public static enum FluxSink.OverflowStrategy
extends Enum<FluxSink.OverflowStrategy>
```

Enumeration for backpressure handling.

Enum Constant Summary

Enum Constants

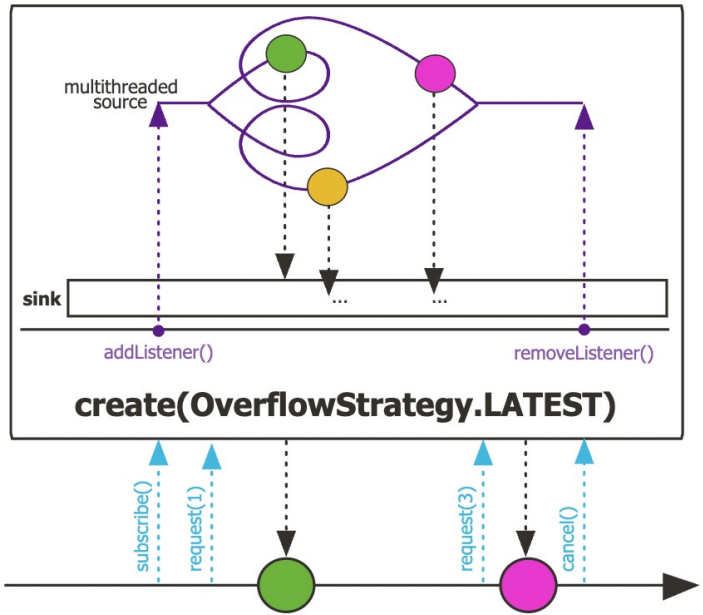
Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator

```
public static <T> Flux<T> create(Consumer<? super FluxSink<T>> emitter,
                                 FluxSink.OverflowStrategy backpressure)
```

Programmatically create a `Flux` with the capability of emitting multiple elements in a synchronous or asynchronous manner through the `FluxSink` API. This includes emitting elements from multiple threads.



This Flux factory is useful if one wants to adapt some other multi-valued async API and not worry about cancellation and backpressure (which is handled by buffering all signals if the downstream can't keep up).

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#create

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
    - Specify the overflow mode to apply if Subscriber can't keep up with Publisher

```
Flux
   .create(makeEmitter(count,
                               sb),
           FluxSink
           .OverflowStrategy
           .ERROR)

   .flatMap(bf1 ->
      multiplyFraction(bf1,
          sBigReducedFraction,
          Schedulers.parallel(),
          sb))

    .subscribe
      (blockingSubscriber);
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/flux/ex4

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
    - Specify the overflow mode to apply if Subscriber can't keep up with Publisher

```
Flux
   .create(makeEmitter(count,
                       sb),
           FluxSink
           .OverflowStrategy
           .ERROR)

   .flatMap(bf1 ->
      multiplyFraction(bf1,
          sBigReducedFraction,
          Schedulers.parallel(),
          sb))

   .subscribe
     (blockingSubscriber);
```

Rapidly emit a stream of random BigFraction objects in one fell swoop

Overview of Flux Overflow Strategies

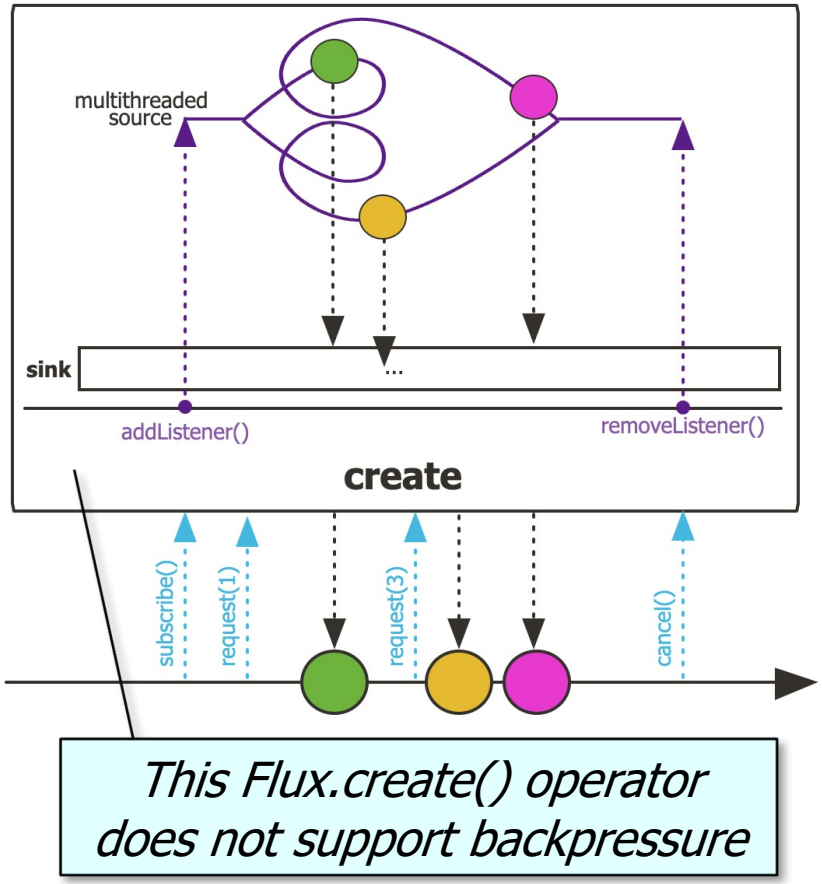
- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
    - Specify the overflow mode to apply if Subscriber can't keep up with Publisher

```
Flux
   .create(makeEmitter(count,
                               sb),
           FluxSink
           .OverflowStrategy
           .ERROR)

   .flatMap(bf1 ->
      multiplyFraction(bf1,
          sBigReducedFraction,
          Schedulers.parallel(),
          sb))

    .subscribe
      (blockingSubscriber);
```

Throw exception when events can't be processed immediately

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
    - Specify the overflow mode to apply if Subscriber can't keep up with Publisher
  - This operator is different than the one param version of Flux.create()



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#create

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
  - They can also be provided via other Flux onBackpressure*() operators

- I want to deal with backpressure "errors" (request max from upstream and apply the strategy when downstream does not produce enough request)…
  - by throwing a special IllegalStateException: Flux#onBackpressureError
  - by dropping excess values: Flux#onBackpressureDrop
    - …except the last one seen: Flux#onBackpressureLatest
  - by buffering excess values (bounded or unbounded): Flux#onBackpressureBuffer
    - …and applying a strategy when bounded buffer also overflows: Flux#onBackpressureBuffer with a BufferOverflowStrategy

See projectreactor.io/docs/core/release/reference/#which.errors

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
  - They can also be provided via other Flux onBackpressure*() operators
    - onBackpressureDrop()
      - Ignore all streamed items that can't be processed until down stream can accept more of them

```
component
   .mouseMoves()
   .onBackpressureDrop()
   .publishOn
      (Schedulers.parallel(),
       1)
   .subscribe(event ->
                compute(event.x,
                        event.y));
```

See Flux.html#onBackpressureDrop

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
  - They can also be provided via other Flux onBackpressure*() operators
    - onBackpressureLatest()
      - Like the DROP strategy, but it keeps the last emitted item

```
component
   .mouseClicks()
   .onBackpressureLatest()
   .publishOn
      (Schedulers.parallel())
   .subscribe(event ->
                compute(event.x,
                        event.y),
                Throwable::
                printStackTrace);
```

See Flux.html#onBackpressureLatest

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
  - They can also be provided via other Flux onBackpressure*() operators
    - onBackpressureBuffer()
      - Creates a bounded or unbounded buffer to hold emitted items that can't be processed by downstream

```
Flux
   .range(1, 1_000_000)
   .onBackpressureBuffer
      (16,
       BufferOverflowStrategy
       .DROP_OLDEST)
   .publishOn
      (Schedulers.parallel())
   .subscribe(e -> { },
              Throwable::
              printStackTrace);
```

See Flux.html#onBackpressureBuffer

End of Overview of Backpressure Models in Project Reactor Flux