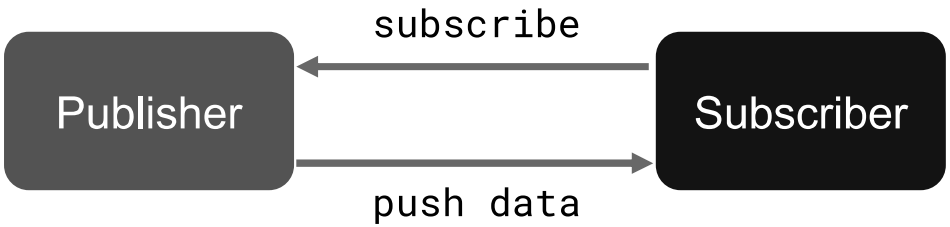
Reactive Streams - Motivation

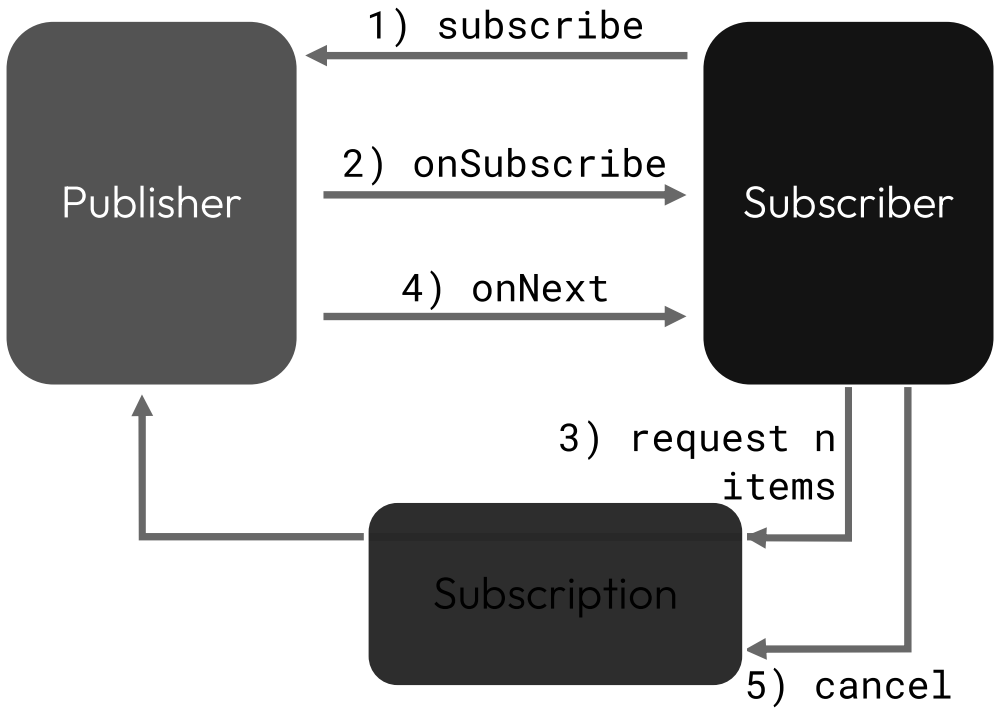
- Ziel: Publisher will Daten zu Subscriber transferieren
- Problem mit Push Strategie:
  - Subscriber überlastet
  - Puffer laufen voll



Reactive Streams - Motivation

Lösung:

- Subscriber fordert nur so viele Daten an, wie er verarbeiten kann (Gegendruck)
- Publisher kann überschüssige Nachrichten
  - warten lassen
  - verwerfen



Grundkonzepte

- interface Flow.Publisher
- interface Flow.Subscriber
- interface Flow.Subscription
- interface Flow.Processor

- class SubmissionPublisher
  - optionaler Puffer je Subscriber (Default-Größe 256 Nachrichten)

Subscriber

```
public class MiniSubscriber implements Flow.Subscriber<String> {

    private Flow.Subscription subscription;

    @Override
    public void onSubscribe(Flow.Subscription subscription) {
        this.subscription = subscription;
        this.subscription.request(1);
    }

    @Override
    public void onNext(String item) {
        this.subscription.request(1);
    }

    @Override
    public void onError(Throwable throwable) { }

    @Override
    public void onComplete() {  }

}
```

- implementiert die vier zentralen Callbacks

SubmissionPublisher

```
// has default buffer size of 256
var pub1 = new SubmissionPublisher<String>();

// custom executor and buffer size
var pub2 = new SubmissionPublisher<String>(
        ForkJoinPool.commonPool(),
        0
);

// custom executor, buffer size and error-handler
var pub3 = new SubmissionPublisher<String>(
        Runnable::run,
        0,
        (subscr, err) -> err.printStackTrace()
);
```

- im JDK vorhandene Klasse
- kann auf drei Arten instanziiert werden (siehe links)
- reicht für viele gängige Anwendungsfälle
- auch als Superklasse nutzbar

Publisher.submit()

```
var publisher = new SubmissionPublisher<String>();
var subscriber = new SimpleSubscriber("subscriber", log);
publisher.subscribe(subscriber);

for (int n = 0; n < 5; n++) {
    String item = "item" + n;
    log.log("Submitting %s", item);
    publisher.submit(item);
}

publisher.close();
```

- `submit()` wartet, bis eine Nachricht an jeden Subscriber zugestellt werden konnte
- `SimpleSubscriber` ist eine selbst erstellte Klasse

Publisher.offer()

```
var publisher = new SubmissionPublisher<String>(
        ForkJoinPool.commonPool(),
        1
);
var subscriber = new SimpleSubscriber("subscriber", log);
publisher.subscribe(subscriber);

for (int n = 0; n < 5; n++) {
    String item = "item" + n;
    log.log("Offering %s", item);
    publisher.offer(item, 100, TimeUnit.MILLISECONDS, null);
}
```

- `offer()` verwirft Nachrichten, wenn Subscriber beschäftigt ist
- optional mit
  - Timeout in Millis, default ist 0
  - Retry Lambda