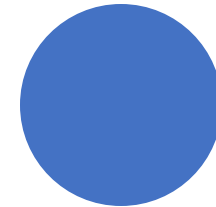


Reactive Clients

Asynchrone Datenverarbeitung

- Früher:
 - Synchrone Verarbeitung --> Programm blockiert beim Warten
 - User-Experience wird geschmälert
- Heute:
 - Asynchrone Verarbeitung --> Hintergrundaufgaben wirklich im Hintergrund

Reactive Clients: Motivation



- Blockieren des Hauptthreads verhindern
- Parallele Datenverarbeitung
- Mithilfe von „async“ und „await“ lesbar wie synchroner Code
- Asynchrone Verarbeitung auch auf Servern wichtig!

Warum asynchron?



Ansätze

Callbacks – Promises -
async/await

Callbacks



- Callback = "Rückruf"
- Rückruffunktion übergeben
- Wird nach Fertigstellung aufgerufen


Callbacks



- Vorteile
 - Gut für einfache asynchrone Vorgänge
 - Sehr verbreitet
- Nachteile
 - "Callback-Hell"
 - Errorhandling pro Callback

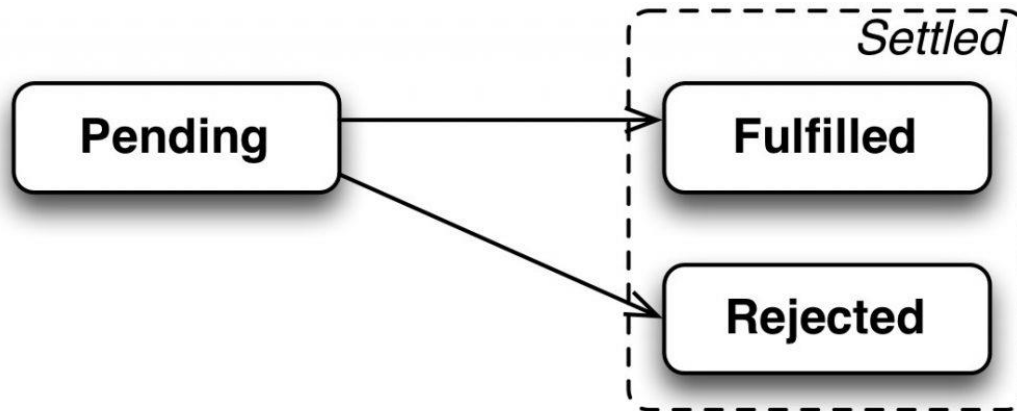
Callback-Hell

```
pan.pourWater(function() {  
  range.bringToBoil(function() {  
    range.lowerHeat(function() {  
      pan.addRice(function() {  
        setTimeout(function() {  
          range.turnOff();  
          serve();  
        }, 15 * 60 * 1000);  
      });  
    });  
  });  
});
```



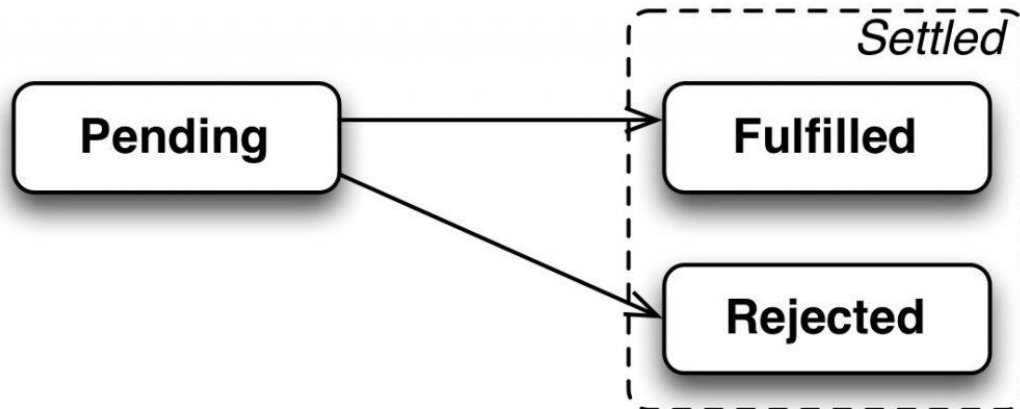
pyramid of doom

Promises



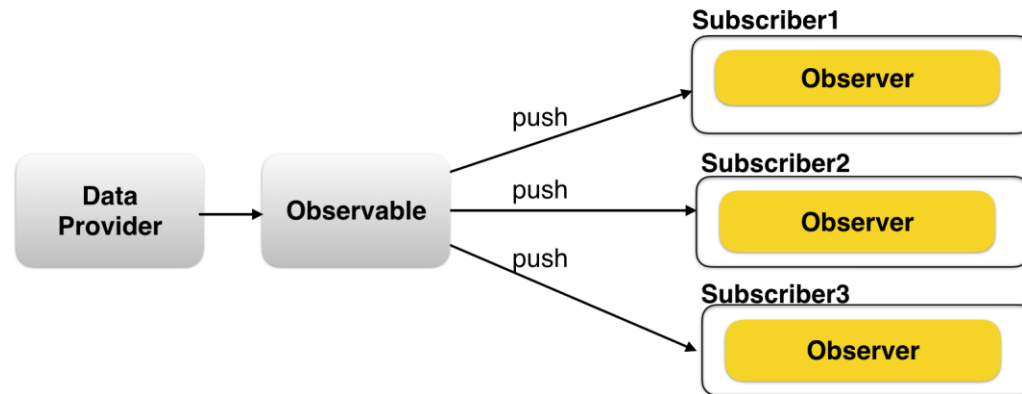
- 3 Zustände
 - "Pending"
 - "Resolved"
 - "Rejected"
- "Versprechen", dass ein Wert zurückgegeben wird.
- Verkettung von asynchronen Vorgängen möglich

Promises

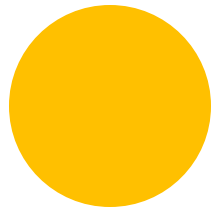
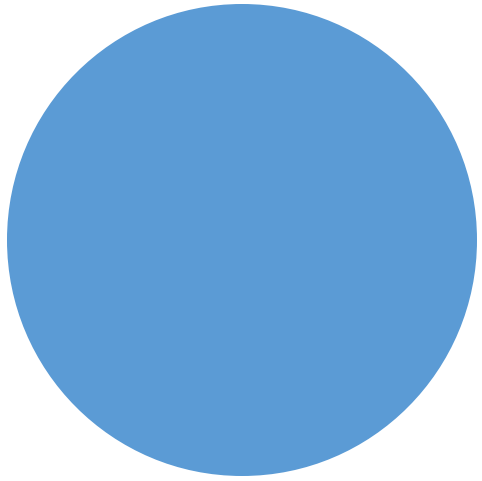


- Vorteile
 - Besser lesbar
 - Einfache Verkettung
 - Ein Errorhandler für alle Operationen
- Nachteile
 - Globale Variablen benötigt
 - Mangelnde Unterstützung

Observables



- Observer registrieren sich beim Observable
- Observable benachrichtigt Observer bei Änderung



Implementationen

Kotlin, JavaScript,
Android

ReactiveX-
Library



- Erhältlich für viele Plattformen (Android, Java, Kotlin, JavaScript ...)
- Baut auf Observables auf, erweitert um Datenstreams

```
CompletableFuture.supplyAsync(() -> 8L)
    .thenApplyAsync(i -> {
        for (long x = 1; x < 29L; x++)
            i = i * x;
        return i;
    })
    .thenAcceptAsync((i) -> System.out.println("Result is " + i));

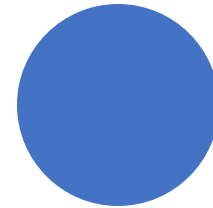
System.out.println("Finished");
```

Output -----

Finished

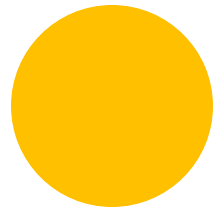
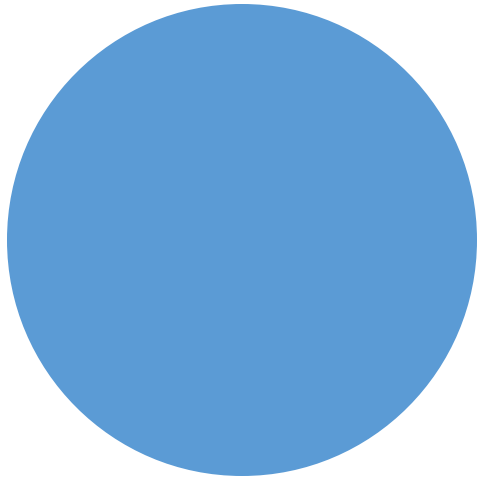
Result is 7594947957393195008

Java



- Promises
- Async await
- Callbacks
- RxJS

Javascript



Es ist Live-Coding-
Zeit!

Bitte VisualStudioCode
starten!