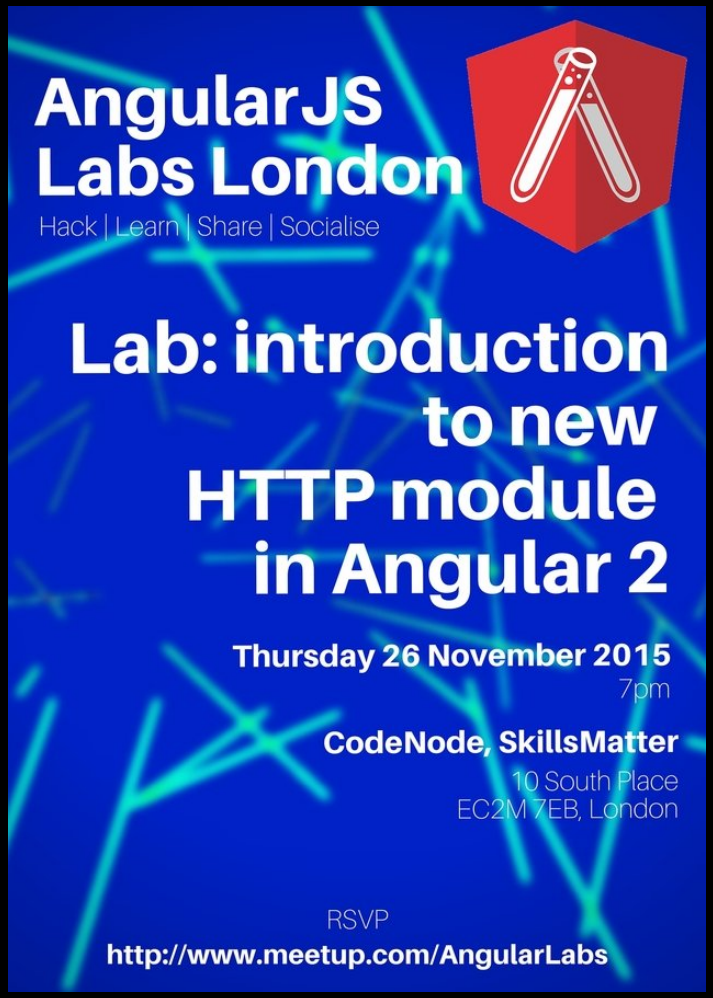
RxJS 5 In-depth

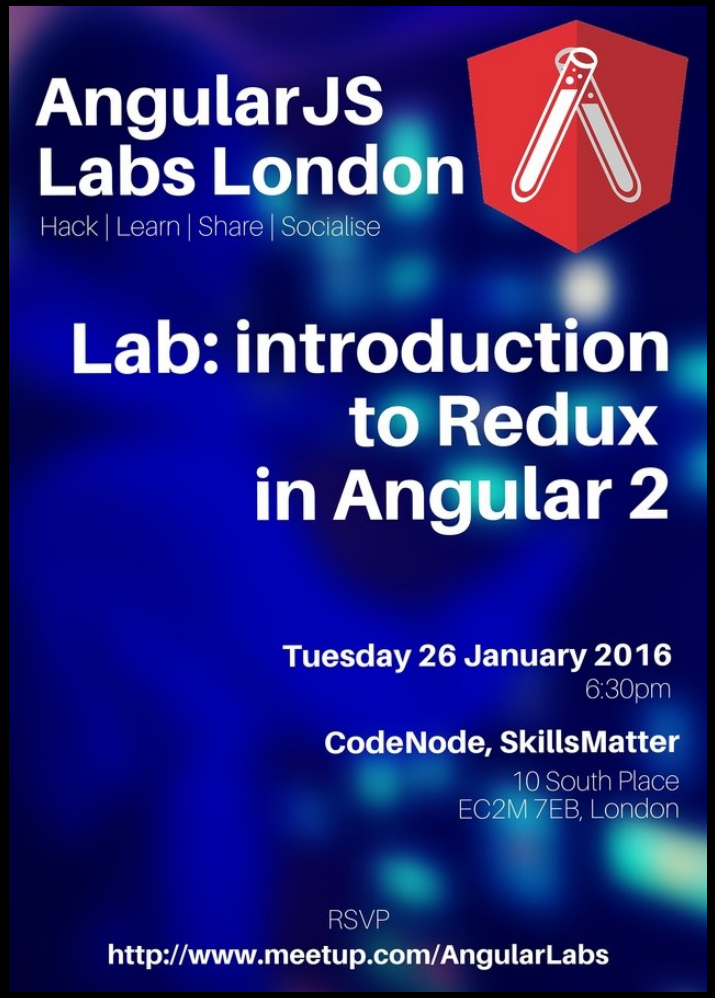
by Gerard Sans (@gerardsans)

QCon LONDON

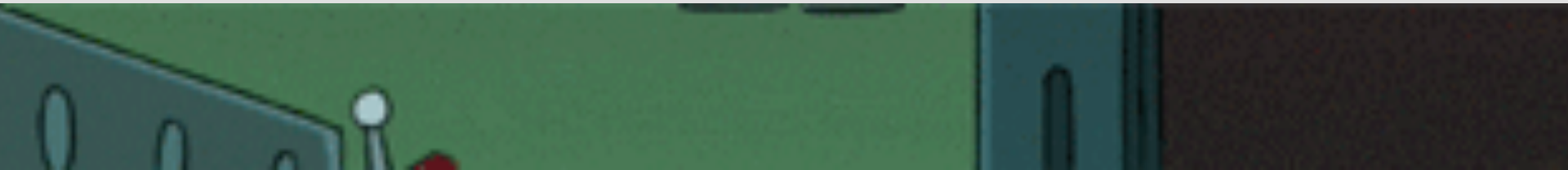
A little about me

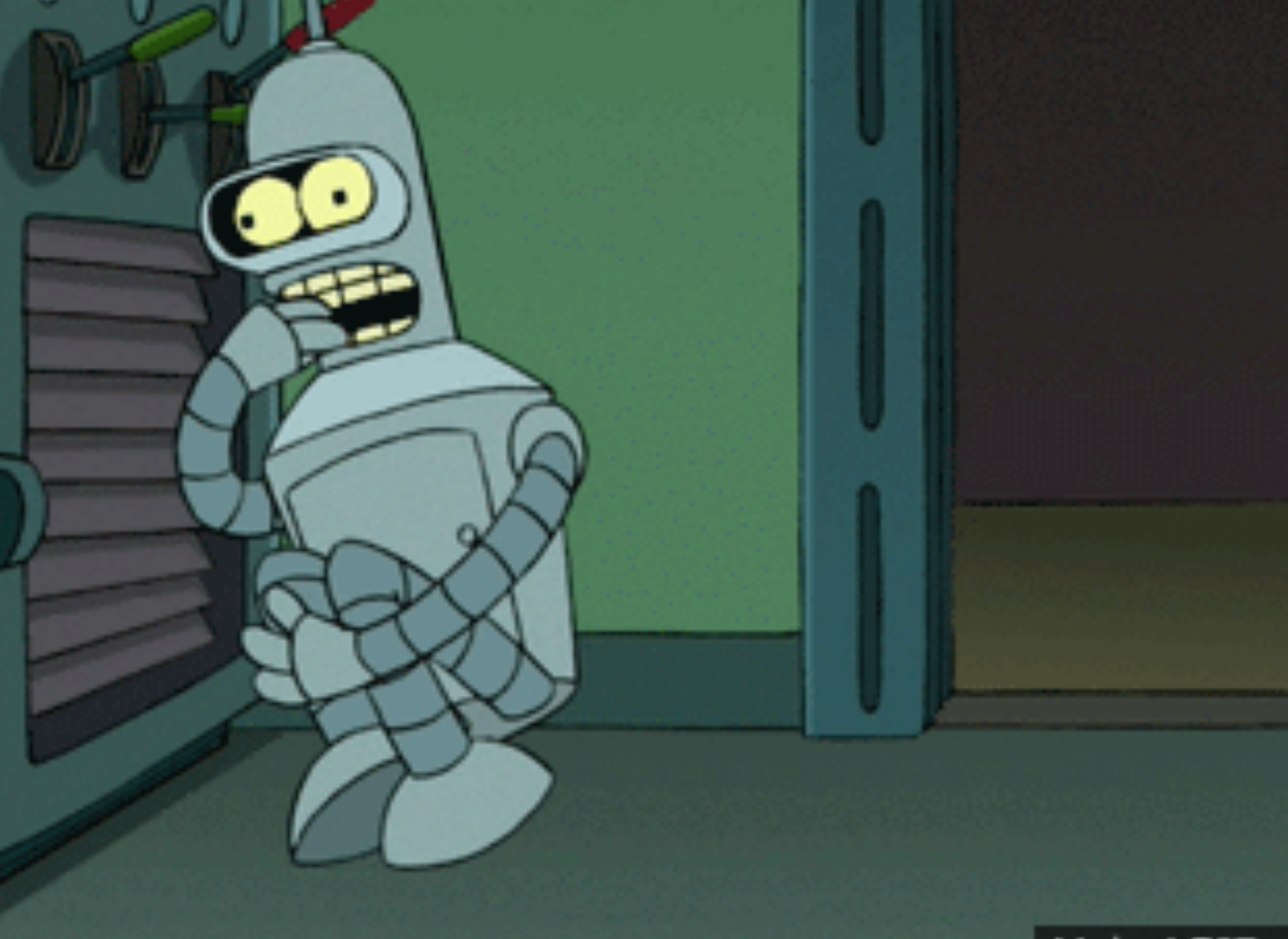
a bit more...


AngularJS
Labs London
Hack | Learn | Share | Socialise
Lab: introduction to new HTTP module in Angular 2
Thursday 26 November 2015
7pm
CodeNode, SkillsMatter
10 South Place
EC2M 7EB, London
RSVP
http://www.meetup.com/AngularLabs


AngularJS
Labs London
Hack | Learn | Share | Socialise
Lab: introduction to Redux in Angular 2
Tuesday 26 January 2016
6:30pm
CodeNode, SkillsMatter
10 South Place
EC2M 7EB, London
RSVP
http://www.meetup.com/AngularLabs

Asynchronous Data Streams





Asynchronous Data Streams

will happen some time in the future

Asynchronous **Data** Streams

raw information

Asynchronous Data **Streams**

values made available over time

Examples

Stream

Array

Pull vs Push

Pull	Push
Arrays, Generators, Iterables	DOM Events Promises Observables
synchronous	asynchronous

Pull Example

```
// Iterable/Iterator
let iterator = [1, 2, 3].values();

console.log(iterator.next()); // {"value":1,"done":
console.log(iterator.next()); // {"value":2,"done":
console.log(iterator.next()); // {"value":3,"done":
console.log(iterator.next()); // {"done":true}

for (let x of [1, 2, 3]) {
  console.log(x);
}
```

Push Examples

```
// DOM Events
var image = document.getElementById('avatar');
image.addEventListener('load', successHandler);
image.addEventListener('error', errorHandler);

// Promise (single value)
get('languages.json')
  .then(successHandler, errorHandler);
```

Streams timeline

- pipes (Unix 3, 1973)

streams (Node.js, 2009)

observables (Microsoft, 2009)

observables (Angular 2, 2014)

101 Arrays

- Array Extras (ES5)
  - .forEach(), .map(), .filter() and .reduce()
- Composition

forEach

```
var team = [
  { name: "Igor Minar", commits: 259 },
  { name: "Jeff Cross", commits: 105 },
  { name: "Brian Ford", commits: 143 }
];

for(var i=0, ii=team.length; i<ii; i+=1){
  console.log(team[i].name);
}

team.forEach( member => console.log(member.name) );

// Igor Minar
// Jeff Cross
// Brian Ford
```

map

```
var team = [
  { name: "Igor Minar", commits: 259 },
  { name: "Jeff Cross", commits: 105 },
  { name: "Brian Ford", commits: 143 }
];

var newTeam = [];
for(var i=0, ii=team.length; i<ii; i+=1){
  newTeam.push({ name: team[i].name });
}

var onlyNames = team.map(
    member => ({ name: member.name })
);
```

filter

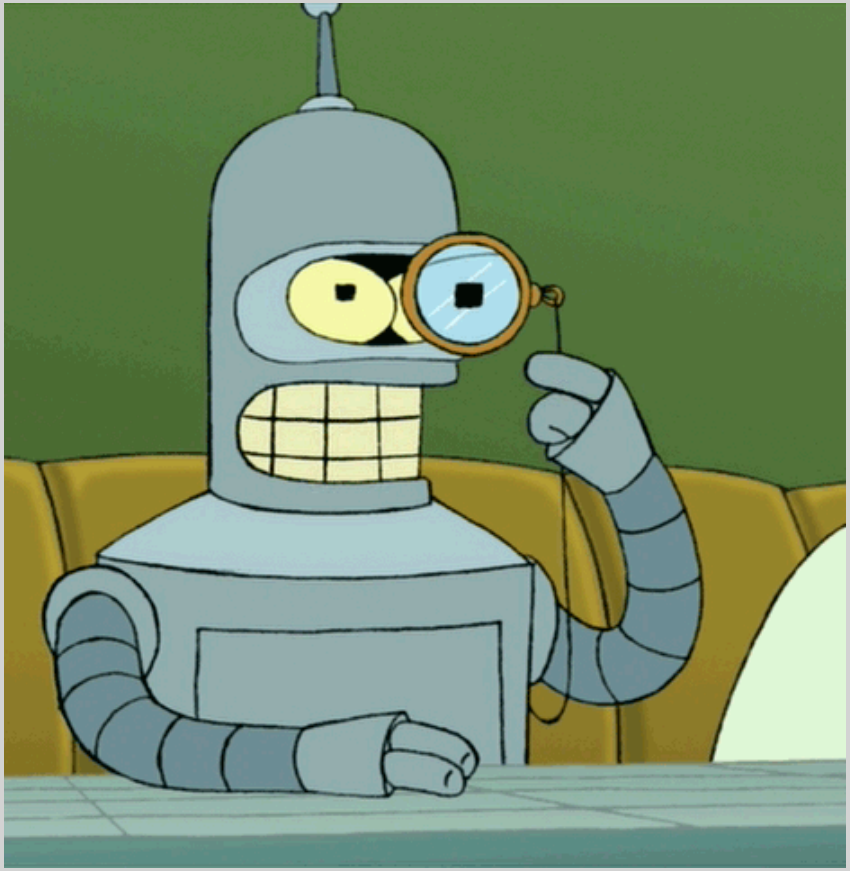
```
var team = [
  { name: "Igor Minar", commits: 259 },
  { name: "Jeff Cross", commits: 105 },
  { name: "Brian Ford", commits: 143 }
];
var onlyOver120Commits = [];
for(var i=0, ii=team.length; i<ii; i+=1){
  if (team[i].commits>120) {
    onlyOver120Commits.push(team[i]);
  }
}
var onlyOver120Commits = team.filter(
    member => member.commits>120
);
```

reduce

```
var team = [
  { name: "Igor Minar", commits: 259 },
  { name: "Jeff Cross", commits: 105 },
  { name: "Brian Ford", commits: 143 }
];
var total = 0; // initial value
for(var i=0, ii=team.length; i<ii; i+=1){
  total = total + team[i].commits;
}
var total = team.reduce(
  (total, member) => total + member.commits
, 0); // initial value

// 507
```

Composition

```
var over120Commits = x => x.commits>120;
var memberName = x => x.name;
var toUpperCase = x => x.toUpperCase();
var log = x => console.log(x);

team
   .filter(over120Commits)
   .map(memberName)
   .map(toUpperCase)
   .forEach(log);

// IGOR MINAR
// BRIAN FORD
```



RxJS 5

Main Contributors

@AndreStaltz

@BenLesh

@_ojkwon

@robwormald

@trxcllnt

Observable

```
//Observable constructor
let obs$ = new Observable(observer => {
  try {
    //pushing values
    observer.next(1);
    observer.next(2);
    observer.next(3);

    //complete stream
    observer.complete();
  }
  catch(e) {
    //error handling
    observer.error(e);
  }
});
```

Basic Stream

```
//ASCII Marble Diagram

----0----1----2----3---->   Observable.interval(1000);
----1----2----3|            Observable.fromArray([1,2,3])
----#                       Observable.of(1,2).do(x => th

---> is the timeline
0, 1, 2, 3 are emitted values
# is an error
| is the 'completed' signal
```

RxMarbles

Observable helpers

```
//Observable creation helpers

Observable.of(1);                              // 1|
Observable.of(1,2,3).delay(100);               // ---1---2---

Observable.from(promise);
Observable.from(numbers$);
Observable.fromArray([1,2,3]);                 // ---1---2---

Observable.fromEvent(inputDOMElement, 'keyup');
```

Subscribe

```
Observable.subscribe(
  /* next */       x => console.log(x),
  /* error */      x => console.log('#'),
  /* complete */   () => console.log('|')
);

Observable.subscribe({
  next:  x => console.log(x),
  error: x => console.log('#'),
  complete: () => console.log('|')
});
```

Hot vs Cold

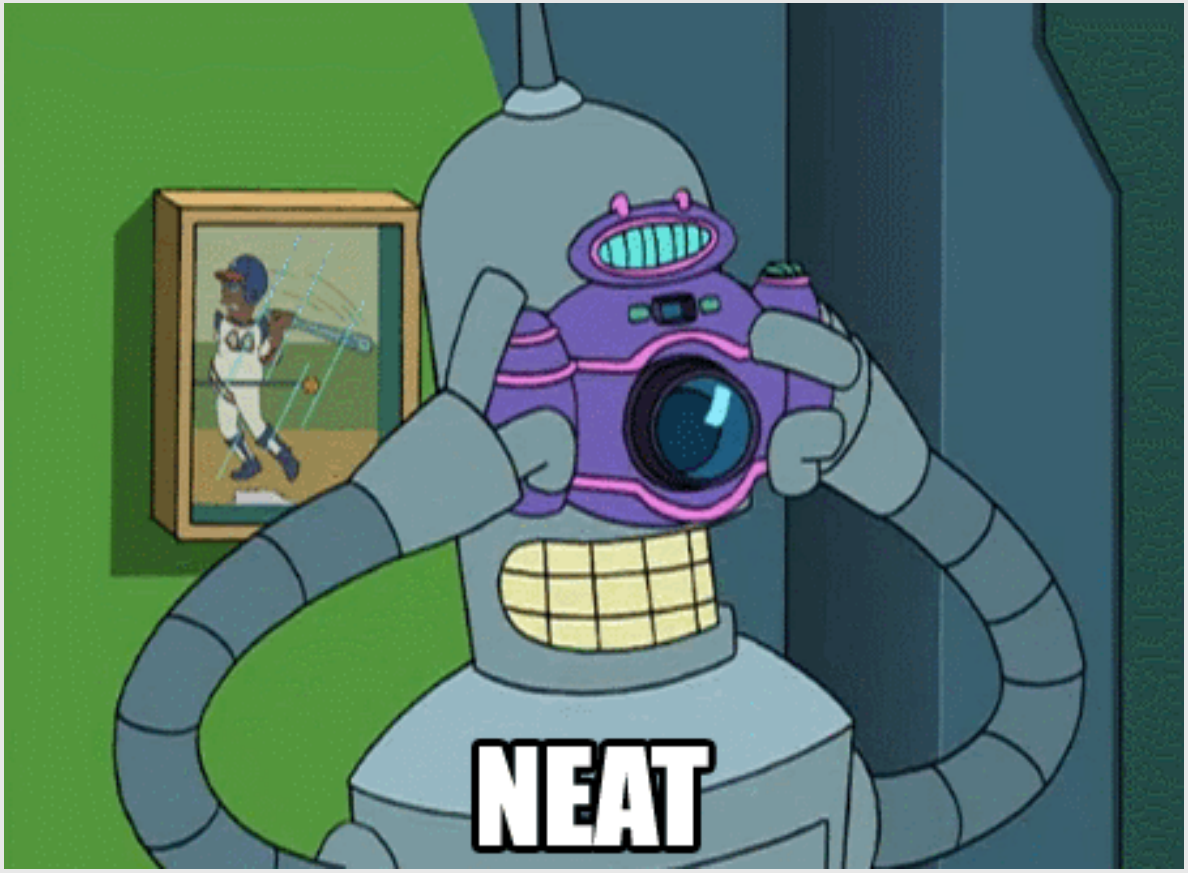
Hot	Cold
obs.shared()	default
broadcasted	pre-recorded
subscribers synced	subscribers not synced

Unsubscribe

```
var subscriber = Observable.subscribe(
  twit => feed.push(twit),
  error => console.log(error),
  () => console.log('done')
);

subscriber.unsubscribe();
```

Operators

```
// simple operators
map(), filter(), reduce(), scan(), first(), last(), sin
elementAt(), toArray(), isEmpty(), take(), skip(), star

// merging and joining
merge(), mergeMap(flatMap), concat(), concatMap(), swit
switchMap(), zip()

// spliting and grouping
groupBy(), window(), partition()

// buffering
buffer(), throttle(), debounce(), sample()
```

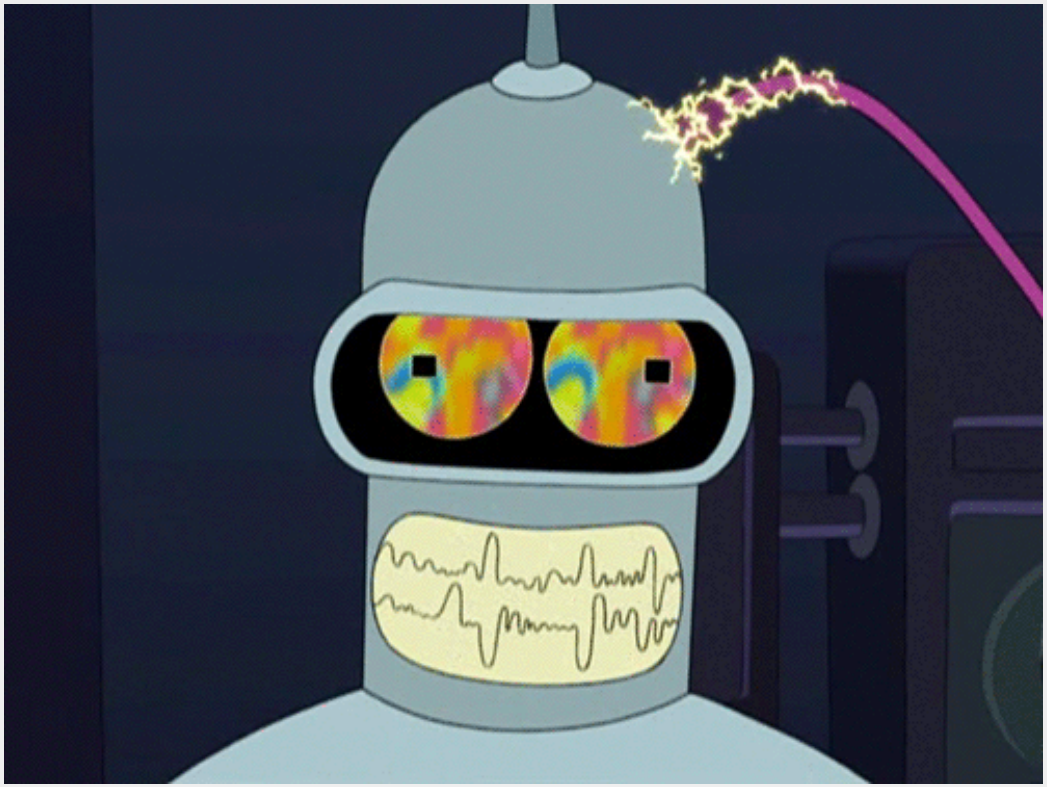

NEAT

Schedulers

```
// Synchronous (default: Scheduler.queue)
Observable.of(1)
  .subscribe({
    next: (x) => console.log(x)
    complete: () => console.log('3')
  });
console.log('2');

// a) 1 2 3
// b) 2 1 3
// c) 1 3 2
// d) 3 2 1
```

Schedulers

```
// Asynchronous
Observable.of(1)
  .observeOn(Scheduler.asap)
  .subscribe({
    next: (x) => console.log(x)
    complete: () => console.log('3')
  });
console.log('2');

// a) 1 2 3
// b) 2 1 3
// c) 1 3 2
// d) 3 2 1
```



Debugging RxJS

- No debugger support yet
- obs.do(x => console.log(x))
- Drawing a marble diagram

RxJS 5 use cases

- Asynchronous processing
- Http
- Forms: controls, validation
- Component events
  - EventEmitter

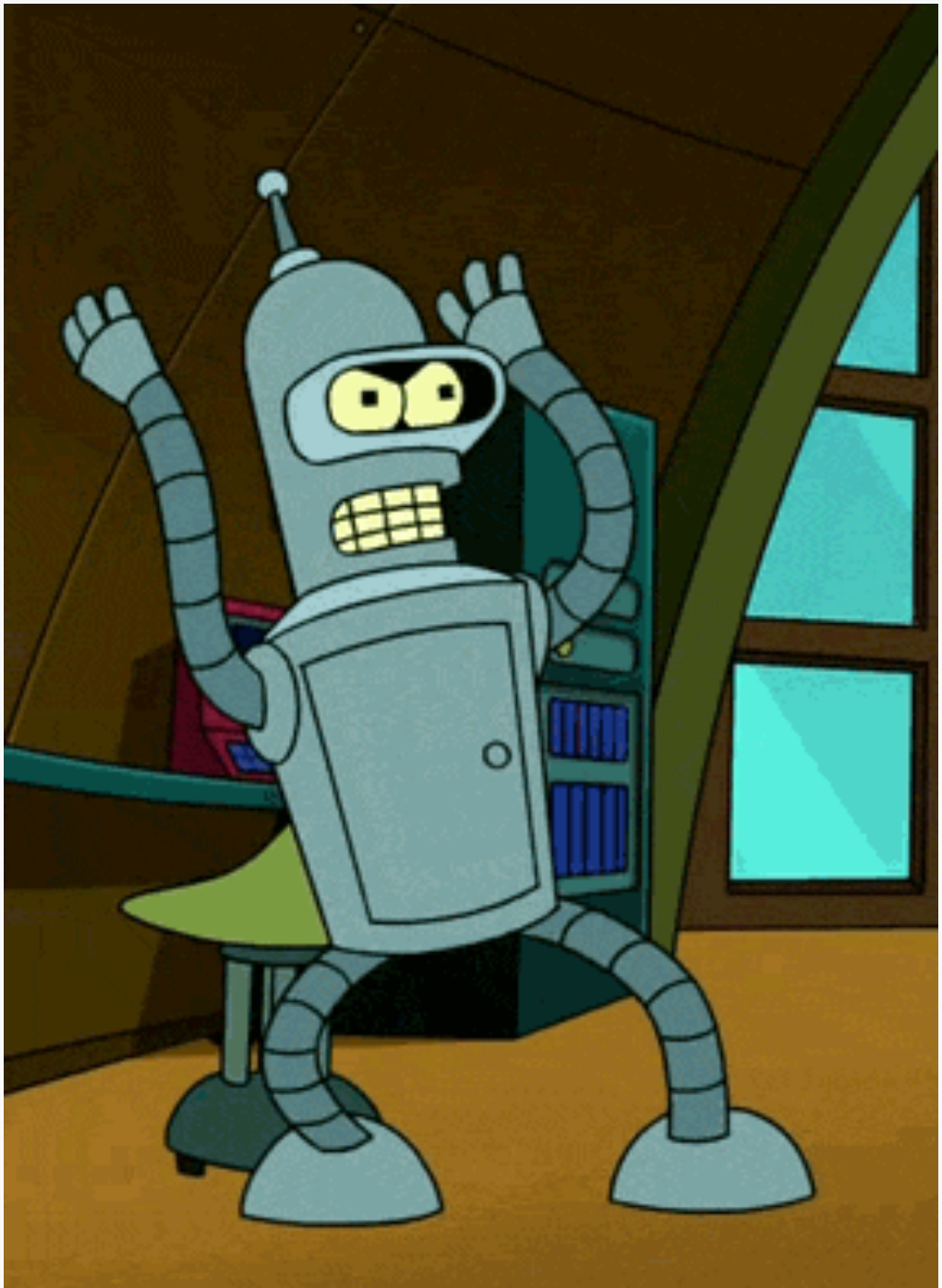
Wikipedia Search

- Http (Jsonp)
- Form: control (valueChanges)
- Async pipe
- RxJS operators
  - flatMap/switchMap
  - retryWhen

plunker

Why Observables?

- Flexible: sync or async
- Powerful operators
- Less code



Want more?

- RxJS 5 (github)
- Reactive Extensions
- Wikipedia Search (plunker)
- RxJS 5 Koans (plunker)

QCon
LONDON

Thanks!