# ES2016 Observable

# 6/13/2004: Async Generators Proposed for ES2016

# Async Generator Goals

- Composable interface for web's push stream APIs (Observable)
- Syntax for producing and consuming push streams (yield, for...on)
- Syntax support for backpressure (await + for...on)

# Async Generators

```
async function* getPriceSpikes(stockSymbol, threshold) {
  let delta,
    oldPrice,
    price;

  for(let price on new WebSocket("ws://www.fakedomain.com/stockstream/" + stockSymbol)) {
    if (oldPrice == null) {
      oldPrice = price;
    }
    else {
      delta = Math.abs(price - oldPrice);
      oldPrice = price;
      if (delta > threshold) {
        yield {price, oldPrice, delta};
      }
    }
  }
}
```

# Issues with Async Gen Proposal

- `async function*() ->` ?
- Push not *necessarily* Async
- No static rejection of await if backpressure not supported

# Issues with Async Gen Proposal

```
async function* getPriceSpikes(stockSymbol, threshold) {
  let delta,
    oldPrice,
    price;

  for(let price on new WebSocket("ws://www.fakedomain.com/stockstream/" + stockSymbol)) {
    if (oldPrice == null) {
      oldPrice = price;
    }
    else {
      delta = Math.abs(price - oldPrice);
      oldPrice = price;
      if (delta > threshold) {
        yield {price, oldPrice, delta};
      }
    }
  }
}
```

AsyncIterator, Observable, or AsyncObservable?

await inside for...on
not statically rejected.

# Observable

- Producer in control
- No backpressure
- Ergonomically and efficiently models...
  - Events
  - Websockets
  - Server-sent events
  - setInterval

# Async Iterator

- Consumer in control
- Natural Backpressure
- Ergonomically models...
  - IO

# Conclusion

**Async Iterators and Observables both ergonomic/efficient for different use cases**

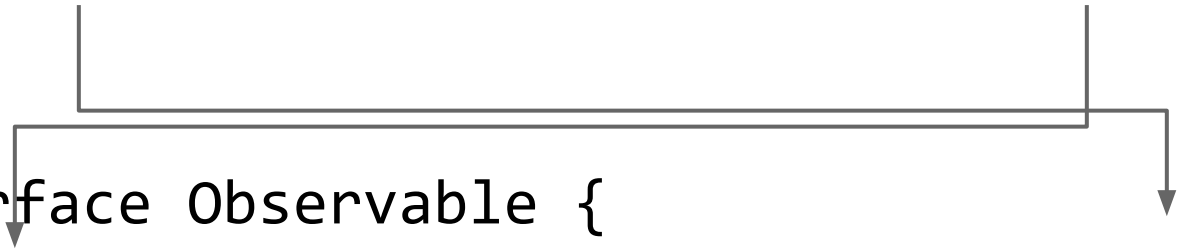
# Strategy

1. ~~Explore Space~~
2. Prioritize
3. Work incrementally

# Proposal: Observable Class in ES2016

# Observable Contract

```
interface Iterable {
   Generator    [Symbol.iterator](void)
}

interface Observable {
   Subscription [Symbol.observer](Generator)
}
```



# Implementing the Contract

```
Array.prototype[Symbol.observer] =
      function(generator) {
            let iterResult;
            try {
                  for(let count = 0; count < this.length; count++) {
                        iterResult = generator.next(this[count]);
                        if (iterResult && iterResult.done) {
                              break;
                        }
                  }
                  if (!iterResult || !iterResult.done) generator.return();
            } catch(e) {
                  generator.throw(e);
            }
            return { unsubscribe() { /* noop */ } };
      };
```

# Implementing the Contract (cont)

```
WebSocket.prototype[Symbol.observer] = function(generator) {
    let unsubscribe,
       handlers = {
            message: v => {
                try {
            let iterResult = generator.next(v);
            if (iterResult && iterResult.done) unsubscribe();
                } catch(e) {
                    unsubscribe();
                    generator.throw(e);
                }
            },
            error: e => { unsubscribe(); generator.throw(e); },
            close: v => { unsubscribe(); generator.return(v); };
        };
    unsubscribe = () => {
        [“message”, “error”, “close”].forEach(message => this.removeEventListener(message, handlers[message]));
    };
    [“message”, “error”, “close”].forEach(message => this.addEventListener(message, handlers[message]));
    return { unsubscribe };
};
```

# Proposal: Observable Class

```
class Observable {
  constructor(observerFnDefn) {
    this._observerFnDefn = observerFnDefn;
  }

  [Symbol.observer](generator) { // schedules subscription immediately
    return this._observerFnDefn(generator);
  }
  static from(observableContractIterableArrayLikeOrPromise): Observable
  // schedules subscription as job
  subscribe(generatorOrNextFn, optionalErrorFn, optionalReturnFn): Subscription
  forEach(action): Promise
}
```

# Observable Usage

```
let subscription =
    Observable.
        from(new Websocket(“/JNJ/prices”)).
        // job scheduled on subscription
        subscribe(
            price => console.log(price));
```

# Observable Usage

```
let subscription =
    Observable.
        from(new Websocket(“/JNJ/prices”)).
        // job scheduled on subscription
        subscribe(
            price => console.log(price),
            error => console.error(error),
            () => console.log(“done”));
```

# Observable Usage

```
let subscription =
    Observable.
        from(new Websocket(“/JNJ/prices”)).
        // job scheduled on subscription
        subscribe({
            next(price) { console.log(price); },
            throw(error) { console.error(error); },
            return() { console.log(“done”); }
        });
```

# Observable Class: Goals

- Standard push stream contract for interoperability
  - Combinator Libs (RxJS, Most.js)
  - Interested MVC Frameworks (Angular 2, React)
  - W3C
- Efficient Composition using combinators
- Ergonomic Consumption of push streams

# Use cases

- Common Interface for push streams
  - Events
  - Web sockets
- Push Stream Composition
- Polymorphic push stream composition
- Cooperative push stream processing
- TCP syntax for push stream processing

# UC #1: Interface for Push Streams

## Websockets

```
let sub = Observable.
  from(webSocket).
  subscribe({
    next(v) { console.log(v); },
    throw(e) { console.error(e); }
    return() { console.log('done') }
  });


// at some later time:
sub.unsubscribe();
```

## DOM events

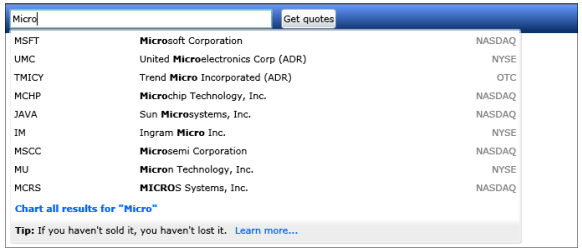
```
let keypresses =
  fromEvent(
    element,
    "keypress",
    false, // useCapture,
    e => e.stopPropagation());


let sub = keypresses.subscribe(
  e => console.log(e.keyCode));


// at some later time:
sub.unsubscribe();
```

# Use Case #2: Event Composition

```
let preventDefaultHandler = e => e.preventDefault();

function getDrags(element) {
  let mouseDowns = fromEvent(element, “mousedown”, false, preventDefaultHandler);
  let mouseMoves = fromEvent(document.body, “mousemove”, false, preventDefaultHandler);
  let mouseUps = fromEvent(document.body, “mouseup”, false, preventDefaultHandler);

  return mouseDowns.flatMap(mouseDownEvent => mouseMoves.takeUntil(mouseUps));
}

getDrags(imageElement).subscribe(mouseDownEvent => {
  imageElement.style.left = mouseDownEvent.offsetX;
  imageElement.style.top = mouseDownEvent.offsetY;
});
```

# Use Case #2: Event Composition

```
let keypresses = fromEvent(textbox, “keypress”, false);
let searchResultSets =
  keypresses.
    map(() => Observable.from(getSearchResultSet(textbox.value))).
    switchLatest();

searchResultSets.
  subscribe(
    searchResultSet => resultsTemplate.apply(resultsDiv, searchResultSet),
    error => alert(‘Server is down’));
```



# Adapting to DOM Events to Observables

```
function fromEvent(element, eventName, useCapture, eventHandler) {
      return new Observable((generator) => {
            let subscription = {
                unsubscribe(){ element.removeEventListener(eventName, handler); }
            },
            handler = function(e) {
                let iterResult;
                eventHandler(e);
                ((iterResult = generator.next(e)) && iterResult.done && subscription.unsubscribe();
            };

            element.addEventListener(eventName, handler, useCapture);
            return subscription;
      });
};
```

# UC #3: Coop Push Stream Processing

```
Observable.
    from(websocket).
    map(JSON.stringify).
    subscribe((function*() {
        let token = function.sent;
        do {
            // parse logic
        } while(token = yield);
    }()));
```

# UC #4: Consumption in async fns

```
async function getPriceHigh(stock) {
    let delta,
        high = Number.MIN_VALUE;

    let socket = new WebSocket(“/dailyPrices/" + stock);
    await Observable.from(socket).forEach(price => {
        if (price > high) {
            high = price;
        }
    });

    return high;
}
```

# UC #4: TCP with for...on

```
async function getNextPriceSpike(stock, threshold) {
    let delta,
       oldPrice,
       price;
    for(let price on new WebSocket(“/prices/" + stock)) {
       if (oldPrice == null) {
         oldPrice = price;
       }
       else {
         delta = Math.abs(price - oldPrice);
          oldPrice = price;
         if (delta > threshold) {
            return { price: price, oldPrice: oldPrice };
         }
       }
   }
}
```

# UC #5: Safe(r) Consumption via Subscription Job Scheduling

```
function runCode() {
    let state = 0;
    observable.subscribe(() => {
        // state guaranteed to be 10
        state++;
    });
    state = 10
}
```

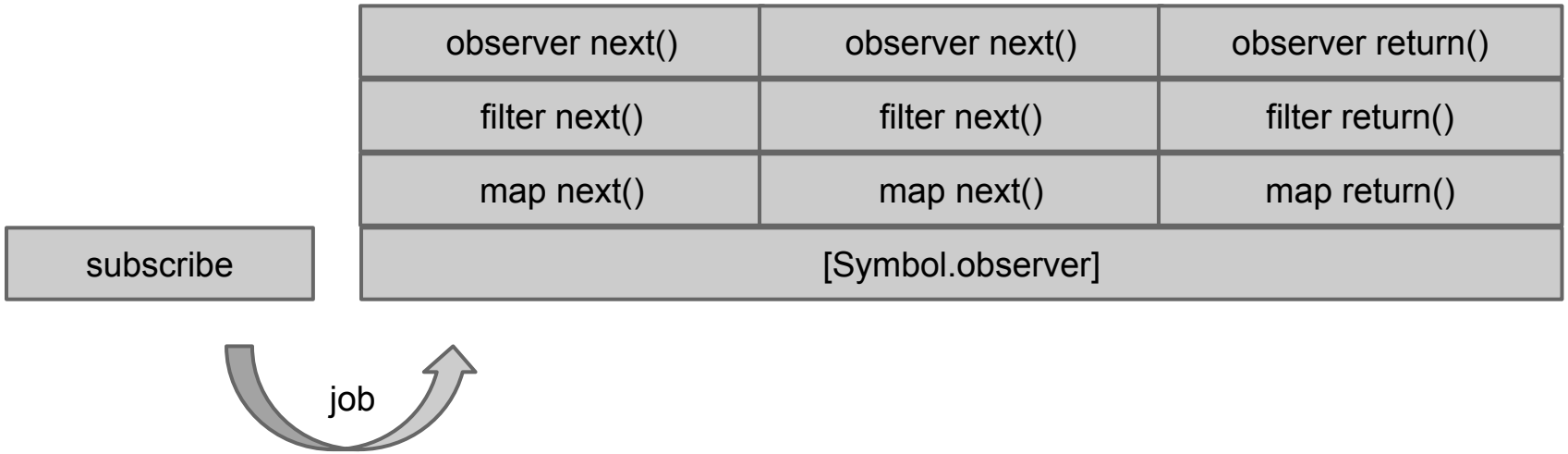
# Efficient Composition via [Symbol.observer]

```
Observable.prototype.map = function(projection) {
    return new Observable(observer => {
        return this[Symbol.observer]({
            next(value) { return observer.next(projection(value));},
            throw(error) { return observer.throw(error); },
            return(value) { return observer.return(value); }
        });
    });
};
```

# Efficient Composition, Safe(r) Consumption

```
[1,2,3].
    map(x => x + 1).
    filter(x => x > 1).
    subscribe(x => console.log(x));
```

# Efficient Composition, Safe(r) consumption



# Leaving Space for Async Iterators

```
async function* getStocks() {
  let reader = new
    AsyncFileReader(“stocks.txt”);
  try {
    while(!reader.eof) {
      let line = await reader.readLine();
      yield JSON.parse(line);
    }
  } finally {
    await reader.close();
  }
}
```

```
async function writeStockInfos() {
  let writer = new
    AsyncFileWriter(“stocksAndPrices.txt”);
  try {
    for async(let stock of getStocks()) {
      await writer.write(line);
    }
  } finally {
    await writer.close();
  }
}
```

# Leaving Space for Async Observables

```
async function*> getStocks() {
  let reader = new
    AsyncFileReader(“stocks.txt”);
  try {
    while(!reader.eof) {
      let line = await reader.readLine();
      yield JSON.parse(line);
    }
  } finally {
    await reader.close();
  }
}
```

```
async function writeStockInfos() {
  let writer = new
    AsyncFileWriter(“stocksAndPrices.txt”);
  try {
    for async(let stock on getStocks()) {
      await writer.write(line);
    }
  } finally {
    await writer.close();
  }
}
```

# Leaving Space for Push Generators

```
function*> getPriceSpikes(stockSymbol, threshold) {
  let delta,
    oldPrice,
    price;

  for(let price on new WebSocket("ws://www.fakedomain.com/stockstream/" + stockSymbol)) {
    if (oldPrice == null) {
      oldPrice = price;
    }
    else {
      delta = Math.abs(price - oldPrice);
      oldPrice = price;
      if (delta > threshold) {
        yield {price, oldPrice, delta};
      }
    }
  }
}
```

Observable