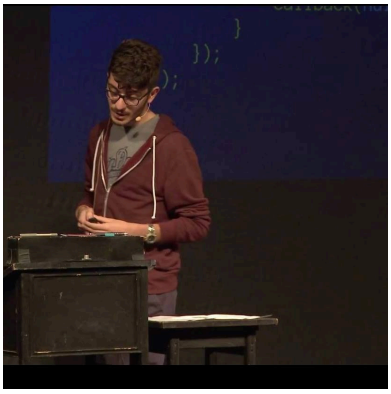
CSP IN JAVASCRIPT



VINCENZO CHIANESE

BUGS INTRODUCER AT APIARY

- https://github.com/XVincentX
- @D3DVincent
- https://vncz.js.org

COMMUNICATING SEQUENTIAL PROCESSES

THE PROBLEM:
SHARING THE MEMORY

- Non determinism
- Race conditions
- Deadlocks

- Hard to **reproduce**
- Hard to **debug**
- Hard to **test**

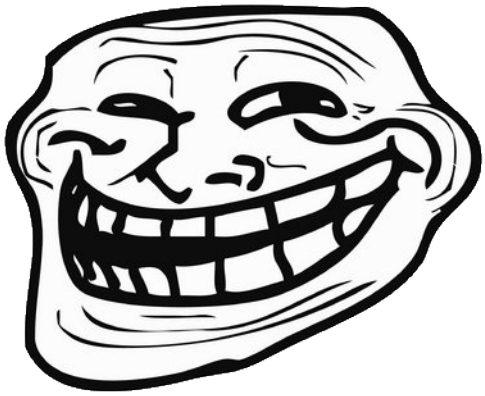
SOLUTIONS

- Mutexes
- Semaphores
- Atomic operations
- ...and a lot more

- **Complexity** and **overhead**
- **Leaky**
- Hard to **compose**

Wait a moment... 

We're using Javascript! Problem solved.



- Any of you, now

HIGH LEVEL TASKS EXAMPLE

- Ajax request - 1, 2, 3, 4
- Rendering - 1, 2

- 1
- 2
- 3
- 4 done!
- 1
- 2 done!
- 😱

- 1
- 1
- 2
- 2 done!
- 3
- 4 done!
- 🌚

COORDINATION OF CONCURRENCY

Callbacks

```
function publish() {
  getProduct('F010', function (err, product) {
    if (err)
      handleError(err);
    else
      getProductSales(product, function (err, sales) {
        if (err)
          handleError(err);
        else
          publishSales(sales, function (err, res) {
            if (err)
              handleError(err);
            else {
              // Do finally something useful
            }
          });
      });
  });
}
```

http://slideslive.com/38894521/from-callbacks-to-promises

Promises

```
function publish() {
  return getProduct('F010')
    .then(getSales)
    .then(publisSales)
    .then(undefined, handleError);
}
```

Generators

```
co(function* publish() {
  try {
    const product = yield getProduct('F010');
    const sales = yield getSales(product);
    yield publishSales(sales);
  } catch (e) {
    handleError(e);
  }
});
```

Async

```
async function publish() {
  try {
    const product = await getProduct('F010');
    const sales = await getSales(product);
    await publishSales(sales);
  } catch (e) {
    handleError(e);
  }
}
```

PROMISES STILL HAVE SOME LIMITATIONS

Promises and events

```
const p1 = new Promise((resolve, reject) => {
  $('#btn').on('click', (evt) => {
    const className = evt.target.className;
    if (className === "tinapacchetella") {
      resolve(className);
    } else {
      reject();
    }
  });
});

p1.then((className) => alert(className));
```

Demo

Promises and events

```
$('#btn2').on('click', (evt) => {
  const p1 = new Promise((resolve, reject) => {
    const className = evt.target.className;
    if (className === "tinapacchetella") {
      resolve(className);
    } else {
      reject();
    };
  });

  p1.then((className) => alert(className));
});
```

Demo

High order pattern

- Observable (RxJs)
- Communicating Sequential Processes

Pipes

- They are **simple** `cat file | grep search`
- They are **composable**
- They are **parallel**
- It does **not leak**

C.S.P.

COMMUNICATING SEQUENTIAL PROCESSES

```
<command> :.--- <simple command>l<structured command>
<simple command> :.--- <null command>l<assignment command>
I<input command>l<output command>
<structured command> :.--- <alternative command>
I<repetitive command>l<parallel command>
<null command> :.--- skip
<command list> :.--- {<declaration>; I<command>;} <command>
```

Original paper 1978

PROCESSES

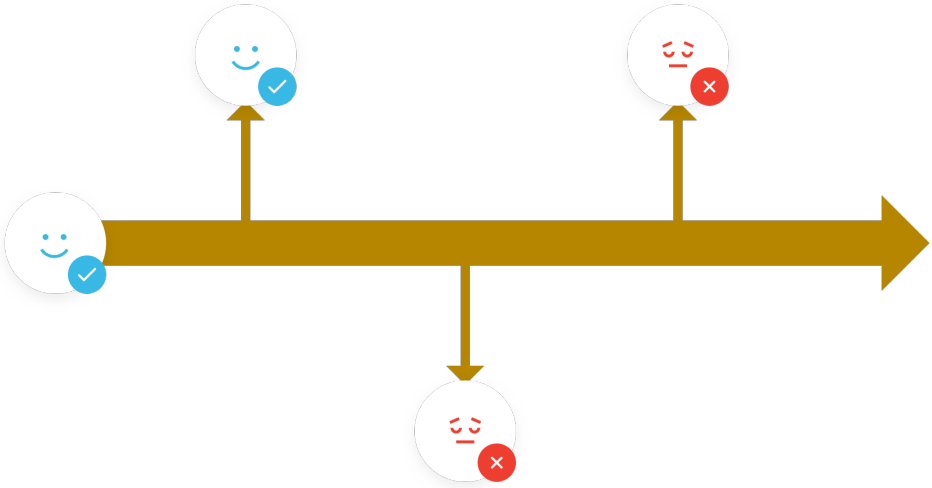
- No **OS processes**
- They **do not** share memory

CHANNELS

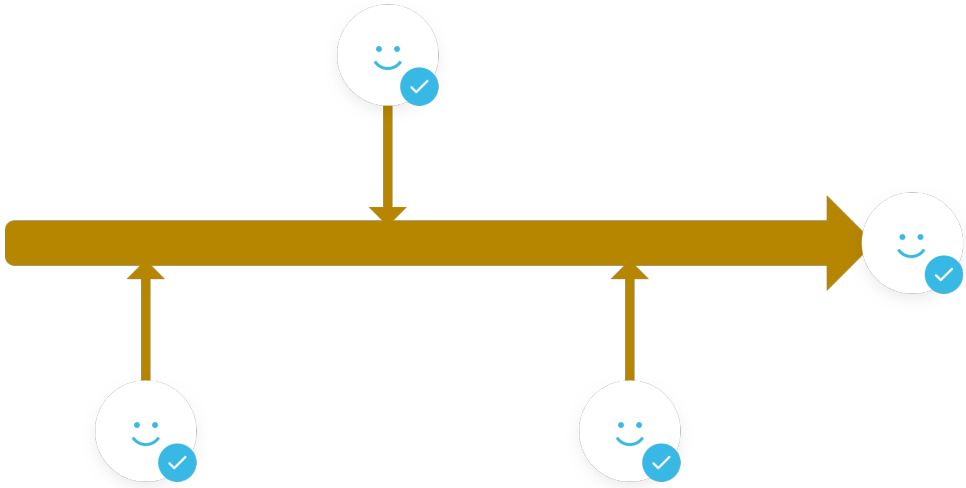
- Pass **data structures**
- Instrinsic configurable **blocking** mechanism

ONE TO ONE

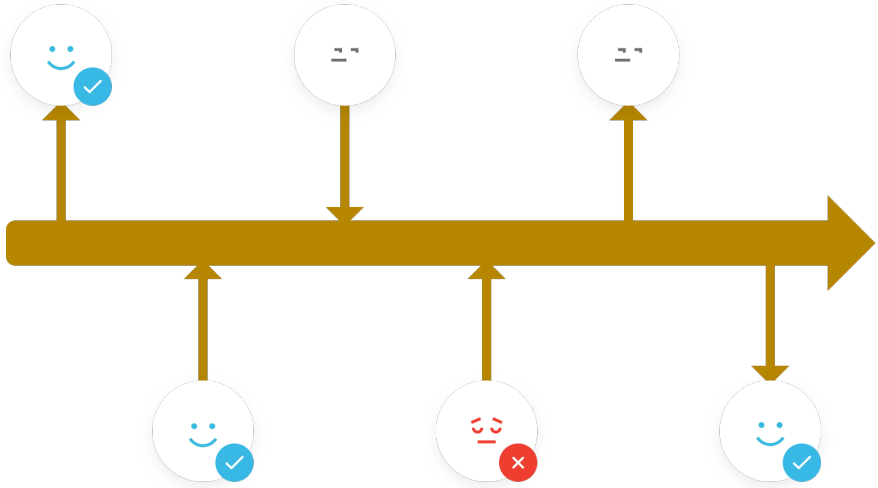
ONE TO MANY



MANY TO ONE



MANY TO MANY



Time to code!

- **Basic** home made example
- CSP to coordinate **callbacks**
- CSP to handle DOM **events**

CSP RECAP

- Solid **theory** concepts (it works)
- **Simple** imperative API compared to RxJs
- Built in **backpressure**

- **Transducer** support
- **Highest** order pattern
- Used in **production** (CircleCI)

CSP CAVEATS

- Idea from **future**
- Different, **opinionated** implementations
- Lack of **integrations**


WE WANT YOU!

THANKS!