SAILING ON HOT STREAMS

Reactive programming in realtime applications

Tyrone Tudehope

WHAT IS A STREAM?

A `sequence` of ongoing `events` ordered in time.

- Andre Staltz

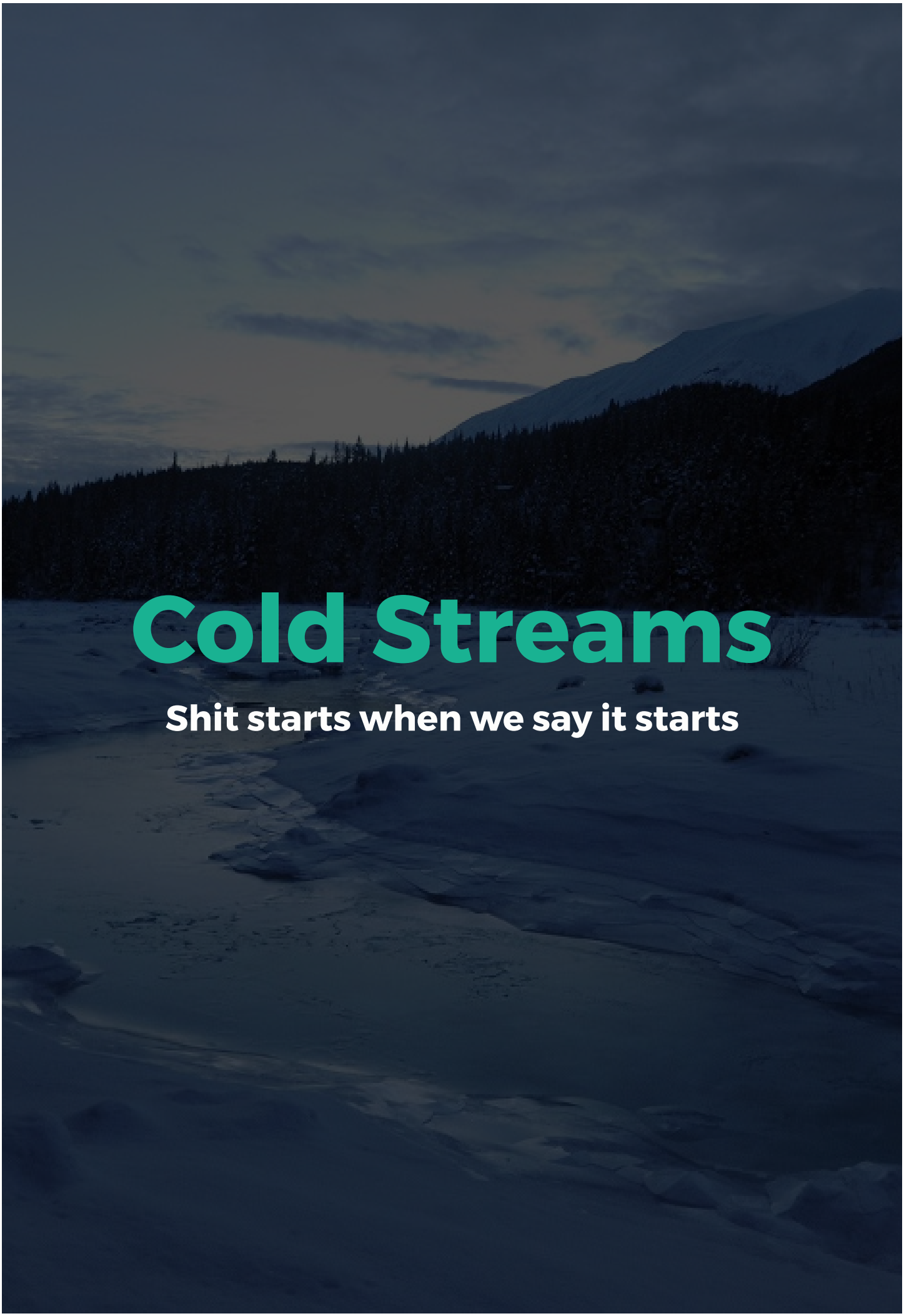
`React` to events `asynchronously` in a `declarative` and `composable` manner

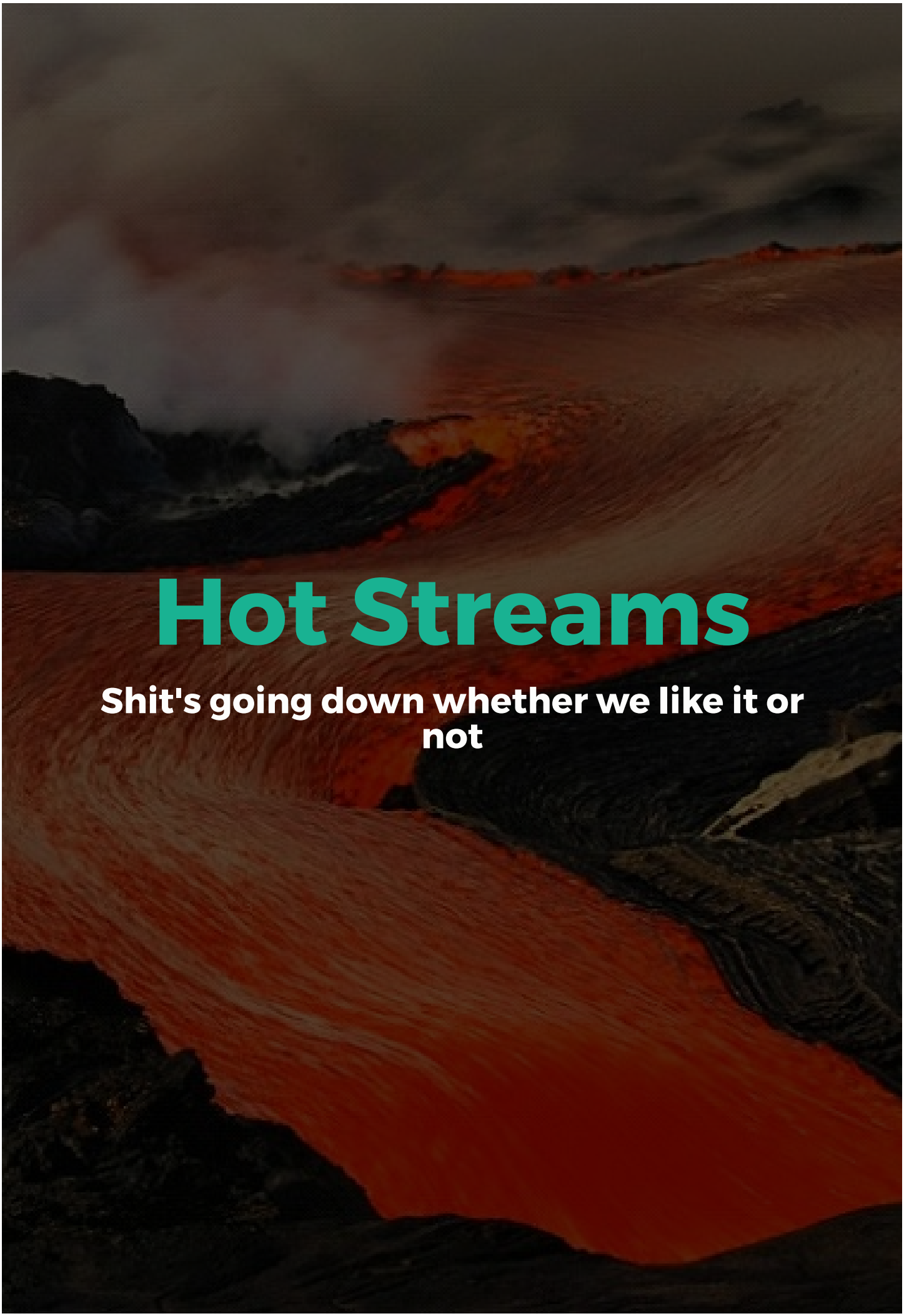
THE WORLD AS EVENTS

- User interaction `element.onclick`
- Filesystem changes `fs.watch`
- Arrays `[1, 2, 3]`
- Single value `{ foo: 'bar' }`
- *Everything*

HOW DO WE USE THEM?

- Define a producer
- Declare steps: `mapReduce`
- Subscribe


Cold Streams
Shit starts when we say it starts


Hot Streams
Shit's going down whether we like it or not

AN IMPERATIVE APPROACH

```
const arr = []
setInterval(() => {
  arr.push(arr.length + 1)
  let sumProduct = 0;
  for (x of arr) {
    if(x % 2 !== 0) {
      sumProduct += x * 10
    }
  }
  console.log(sumProduct)
}, 1000)

// 10
// 40
// 90
// ...
```

LET'S TRY W/ STREAMS

```
xs
.periodic(1000) // Produce
/* mapReduce */
.filter(isOdd)
.map(productOf(10))
.fold(sum, 0)
.subscribe({ // Observe
  next: x => console.log(x)
})

// 10
// 40
// 90
// ...
```

A STEP FURTHER...

```
const count$ = xs.periodic(1000) // Produce

count$
.compose(sumOddProductOf(10))
.subscribe(logger('Odd')) // Observe

count$
.compose(sumPrimeProductOf(5))
.subscribe(logger('Prime')) // Observe

// Odd: 10
// Prime: 10
// Odd: 40
// Prime: 25
// Odd: 90
// Prime: 50
```

FULLSTACK? YES PLEASE!

- Same paradigms
- Explicit & Predictable
- Testable
- Focus on `what` not `how`

CYCLE.JS

- Reduce app to a `single pure function`
- Everything is streams
- Drivers

XSTREAM

- **Immutable**
- **Pure**
- **Hot streams only**

YACHT UI

- Monitor instrumentation
- Control the rudder remotely
- Realtime

MONITORING INSTRUMENTATION

```
// Messages from server over socket
const vdom$ = sources.Socket
  .select('message')
  .compose(pickSensors)
  .map(sensors =>
    div([
      div(sensors.bearing),
      div(sensors.boatSpeed),
      div(sensors.windDirection),
      div(sensors.windSpeed),
    ])
  )

return {
  DOM: vdom$
}
```

CHANGING COURSE

```
// Messages from client
const steer$ = sources.Socket
  .select('message')
  .map(angleProp)
  .map(notify('./interface/steer'))

// Messages to client
const sensors$ = sources.Boat.watch
  .compose(combineSensors)
  .compose(emitUpdate)

return {
  Socket: sensors$,
  Boat: steer$
}
```

DEMO

#AMA

Source + Slides on Github