

# Having Fun with ES Proxies

Clarke's Third Law in Action

A talk by [Christophe Porteneuve](#) at [dotJS 2025](#)

# whoami

```
const christophe = {
  family: { wife: '👩 Élodie', sons: ['👦 Maxence', '👦 Elliott'] },
  city: 'Paris, FR',
  company: {
    name: 'Doctolib', // careers.doctolib.fr 😊
    hiring: true,
    superCool: true
  },
  webDevSince: 1995, // 🎉
  mightBeKnownFor: [
    'Prototype.js',
    'Prototype and Script.aculo.us ("The Bungie Book")',
    'dotJS',
    'Paris Web',
    'NodeSchool Paris',
  ],
}
```

# About these slides...

These are interactive. Hover on the bottom-left corner for the toolbar.

You also get a lot of keyboard shortcuts:

- `f` to toggle **fullscreen**
- `d` to toggle **dark mode**
- `g` to go to any slide directly, by number or fuzzy title match
- `o` to toggle the **overview** layout (see all slides as thumbs)
- `←` / `→` / `Space` to navigate step-by-step
- `↑` / `↓` to navigate slide-by-slide
- `Home` / `End` (or `Cmd` `↑` / `Cmd` `↓`) to go to first / last slide
- Hover on the bottom-left corner for a toolbar (including a PDF download link!)
- Many code examples have links to their source files so you can play with the whole implementation



# Lay of the land

You likely haven't used this stuff yet.

# What are proxies?

They're **not** network proxies 😊

- A proxy is a **wrapper** around an object (which could be a function)
- It can **intercept any interaction** your code has with that object... through the wrapper.
- It's the latest step in **JS metaprogramming**
- **Can't be polyfilled.**
- Showed up in **ES2015**, available across browsers since Sep 2016, and Node 6+.
- Deemed **baseline** for a long time.

# You build one like this

```
new Proxy(obj, {  
  // handler object, usually provides "traps": get, set, has, apply, etc.  
})
```

- **Target** = the proxied object
- **Prop / Key** = property being used (looked up, read, written to or whose descriptor is defined)
- **Value** = property value being set
- For `apply` and `construct` you get arguments, `apply` also has the binding (`this`)

# You build one like this

```
new Proxy(obj, {  
    // handler object, usually provides "traps": get, set, has, apply, etc.  
})
```

- **Target** = the proxied object
- **Prop / Key** = property being used (looked up, read, written to or whose descriptor is defined)
- **Value** = property value being set
- For `apply` and `construct` you get arguments, `apply` also has the binding (`this`)

`Reflect` provides all the **built-in behaviors** for every **trap**, so we can delegate to them.

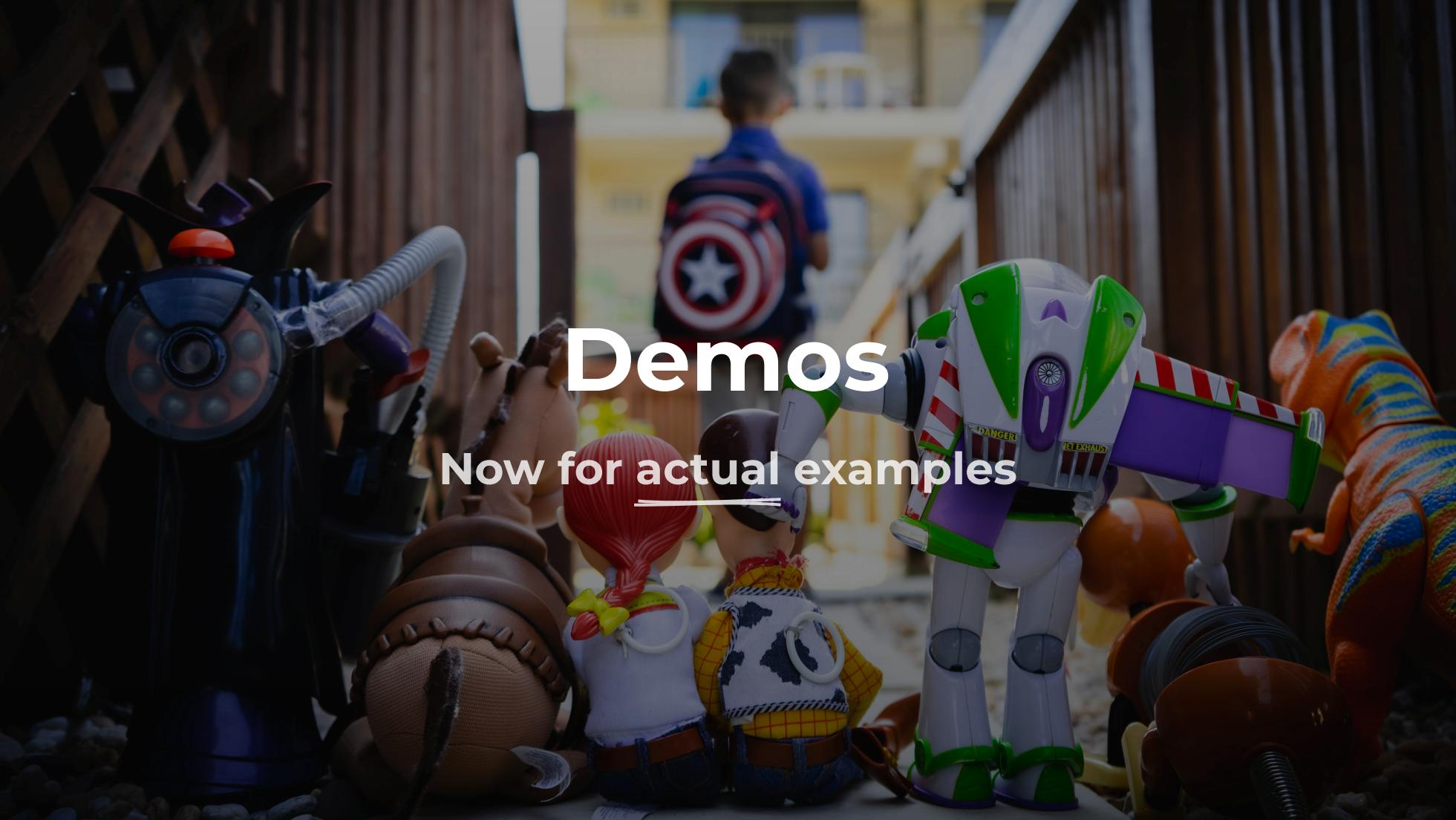
# Here's a dumb example

```
1 const conference = { name: 'dotJS', year: 2025 }
2 const proxiedConf = new Proxy(conference, {
3   get(target, key, receiver) {
4     console.log(`Reading ${key as string} off`, target)
5     return Reflect.get(target, key, receiver)
6   },
7 })
8
9 // console.log(proxiedConf.name)
```



# Demos

Now for actual examples

A photograph of a child's playroom. In the foreground, several Toy Story characters are scattered on the floor: Mr. Potato Head, Jessie, Woody, Hamm the piggy bank, and Buzz Lightyear. A young boy with a Captain America backpack is seen from behind, walking away from the camera towards a yellow building. The room has wooden walls and a window.

# Demos

Now for actual examples

# “Type-safe” object properties

```
1  function makeTypeSafish(obj: object) {
2      return new Proxy(obj, {
3          set(target, key, value) {
4              if (key in target) {
5                  const knownType = typeof target[key];
6
7                  if (knownType !== typeof value) {
8                      throw new Error(`Property ${key as string} expects a ${knownType}`);
9                  }
10             }
11
12             return Reflect.set(target, key, value);
13         },
14     });
15 }
```

» [type-safe-properties.ts](#)

# “Type-safe” object properties

```
1  function makeTypeSafish(obj: object) {
2    return new Proxy(obj, {
3      set(target, key, value) {
4        if (key in target) {
5          const knownType = typeof target[key];
6
7          if (knownType !== typeof value) {
8            throw new Error(`Property ${key as string} expects a ${knownType}`);
9          }
10         }
11
12         return Reflect.set(target, key, value);
13       },
14     });
15   }
```

» [type-safe-properties.ts](#)

# “Type-safe” object properties

```
1  function makeTypeSafish(obj: object) {
2    return new Proxy(obj, {
3      set(target, key, value) {
4        if (key in target) {
5          const knownType = typeof target[key];
6
7          if (knownType !== typeof value) {
8            throw new Error(`Property ${key as string} expects a ${knownType}`);
9          }
10         }
11
12         return Reflect.set(target, key, value);
13       },
14     });
15   }
```

» [type-safe-properties.ts](#)

# “Type-safe” object properties

```
1  function makeTypeSafish(obj: object) {
2    return new Proxy(obj, {
3      set(target, key, value) {
4        if (key in target) {
5          const knownType = typeof target[key];
6
7          if (knownType !== typeof value) {
8            throw new Error(`Property ${key as string} expects a ${knownType}`);
9          }
10         }
11
12         return Reflect.set(target, key, value);
13       },
14     });
15   }
```

» [type-safe-properties.ts](#)

# “Type-safe” object properties

```
const bestConf = makeTypeSafish({ name: 'dotJS' })
```

# “Type-safe” object properties

```
const bestConf = makeTypeSafish({ name: 'dotJS' })  
  
bestConf.name = 42  
// ⚡ Throws with "Property name expects a string"
```

# “Type-safe” object properties

```
const bestConf = makeTypeSafish({ name: 'dotJS' })  
  
bestConf.edition = 10  
// ✅ OK  
  
bestConf.edition = '10th'  
// 💥 Throws with "Property edition expects a number"
```

# Shout-out: tpyo

The lib that defeats typos 😍

```
1 import tpyo from 'tpyo'  
2  
3 const woah = tpyo(['dotAI', 'dotJS'])  
4 woah.pousse('dotCSS', 'dotSwift')  
5 console.log(woah.longueur, woah.avril(3), woah.plop(), woah.shot())
```



```
4, dotSwift, dotSwift, dotAI
```

Finds the closest existing property using Levenshtein distances 😎

# Negative array indices

```
1  function allowNegativeIndices(arr: Array<unknown>) {
2    return new Proxy(arr, {
3      get(target, prop, receiver) {
4        const index = Number(prop);
5        if (index < 0) prop = String(index + target.length);
6        return Reflect.get(target, prop, receiver);
7      },
8      set(target, prop, value, receiver) {
9        const index = Number(prop);
10       if (index < 0) prop = String(index + target.length);
11       return Reflect.set(target, prop, value, receiver);
12     },
13   });
14 }
```

» [negative-array-playground.ts](#)

# Negative array indices

```
1  function allowNegativeIndices(arr: Array<unknown>) {
2    return new Proxy(arr, {
3      get(target, prop, receiver) {
4        const index = Number(prop);
5        if (index < 0) prop = String(index + target.length);
6        return Reflect.get(target, prop, receiver);
7      },
8      set(target, prop, value, receiver) {
9        const index = Number(prop);
10       if (index < 0) prop = String(index + target.length);
11       return Reflect.set(target, prop, value, receiver);
12     },
13   });
14 }
```

» [negative-array-playground.ts](#)

# Negative array indices

```
const allTheDots = allowNegativeIndices([
  'dotJS', 'dotAI',
  'dotRB', 'dotCSS', 'dotScale', 'dotSecurity', 'dotSwift',
])
allTheDots[0] // => 'dotJS'
allTheDots[-1] // => 'dotSwift'
allTheDots[-2] // => 'dotSecurity'
```

# Negative array indices

```
const allTheDots = allowNegativeIndices([
  'dotJS', 'dotAI',
  'dotRB', 'dotCSS', 'dotScale', 'dotSecurity', 'dotSwift',
])
allTheDots[0] // => 'dotJS'
allTheDots[-1] // => 'dotSwift'
allTheDots[-2] // => 'dotSecurity'
allTheDots[-3] = 'dotMontéeÀLÉchelle' // Oh là là the frenchness 🇫🇷
```

# Negative array indices

```
1 const allTheDots = allowNegativeIndices([
2   'dotJS', 'dotAI',
3   'dotRB', 'dotCSS', 'dotScale', 'dotSecurity', 'dotSwift',
4 ])
5
6 allTheDots[0] // => 'dotJS'
7 allTheDots[-1] // => 'dotSwift'
8 allTheDots[-2] // => 'dotSecurity'
9 allTheDots[-3] = 'dotMontéeÀLÉchelle'
10 console.log(allTheDots[4])
11 console.log(Object.keys(allTheDots))
12
```



```
dotMontéeÀLÉchelle
["0", "1", "2", "3", "4", "5", "6"]
```

» [negative-array-playground.ts](#)

# Mini observables

```
1 type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
2
3 function makeObservable(obj: object) {
4   const subscribers = new Set<Subscriber>()
5   const proxy = new Proxy(obj, {
6     ...
7     set(target, key, value, receiver) {
8       const previous = target[key as keyof typeof target]
9       const result = Reflect.set(target, key, value, receiver)
10      subscribers.forEach((sub) => sub({ key, previous, value }))
11      return result
12    }
13  }) as Subscribable
14
15  proxy.$subscribe = ...
16
17  return proxy
18 }
```

» [mini-observables.ts](#)

# Mini observables

```
1 type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
2
3 function makeObservable(obj: object) {
4   const subscribers = new Set<Subscriber>()
5   const proxy = new Proxy(obj, {
6     ...
7     set(target, key, value, receiver) {
8       const previous = target[key as keyof typeof target]
9       const result = Reflect.set(target, key, value, receiver)
10      subscribers.forEach((sub) => sub({ key, previous, value }))
11      return result
12    }
13  }) as Subscribable
14
15  proxy.$subscribe = ...
16
17  return proxy
18}
```

» [mini-observables.ts](#)

# Mini observables

```
1 type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
2
3 function makeObservable(obj: object) {
4   const subscribers = new Set<Subscriber>()
5   const proxy = new Proxy(obj, {
6     ...
7     set(target, key, value, receiver) {
8       const previous = target[key as keyof typeof target]
9       const result = Reflect.set(target, key, value, receiver)
10      subscribers.forEach((sub) => sub({ key, previous, value }))
11      return result
12    }
13  }) as Subscribable
14
15  proxy.$subscribe = ...
16
17  return proxy
18}
```

» [mini-observables.ts](#)

# Mini observables

```
1 type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
2
3 function makeObservable(obj: object) {
4   const subscribers = new Set<Subscriber>()
5   const proxy = new Proxy(obj, {
6     ...
7     set(target, key, value, receiver) {
8       const previous = target[key as keyof typeof target]
9       const result = Reflect.set(target, key, value, receiver)
10      subscribers.forEach((sub) => sub({ key, previous, value }))
11      return result
12    }
13  }) as Subscribable
14
15  proxy.$subscribe = ...
16
17  return proxy
18}
```

» [mini-observables.ts](#)

# Mini observables

```
1  type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
2
3  function makeObservable(obj: object) {
4    const subscribers = new Set<Subscriber>()
5    const proxy = new Proxy(obj, {
6      ...
7      set(target, key, value, receiver) {
8        const previous = target[key as keyof typeof target]
9        const result = Reflect.set(target, key, value, receiver)
10       subscribers.forEach((sub) => sub({ key, previous, value }))
11       return result
12     }
13   }) as Subscribable
14
15   proxy.$subscribe = ...
16
17   return proxy
18 }
```

» [mini-observables.ts](#)

# Mini observables

```
1  type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
2
3  function makeObservable(obj: object) {
4    const subscribers = new Set<Subscriber>()
5    const proxy = new Proxy(obj, {
6      deleteProperty(target, key) {
7        const previous = target[key as keyof typeof target]
8        const result = Reflect.deleteProperty(target, key)
9        subscribers.forEach((sub) => sub({ key, previous, deleted: true }))
10       return result
11     },
12     ...
13   }) as Subscribable
14
15   proxy.$subscribe = ...
16
17   return proxy
18 }
```

» [mini-observables.ts](#)

# Mini observables

```
1 type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
2
3 function makeObservable(obj: object) {
4   const subscribers = new Set<Subscriber>()
5   const proxy = new Proxy(obj, {
6     deleteProperty(target, key) {
7       const previous = target[key as keyof typeof target]
8       const result = Reflect.deleteProperty(target, key)
9       subscribers.forEach((sub) => sub({ key, previous, deleted: true }))
10      return result
11    },
12    ...
13  }) as Subscribable
14
15  proxy.$subscribe = ...
16
17  return proxy
18 }
```

» [mini-observables.ts](#)

# Mini observables

```
1 type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
2
3 function makeObservable(obj: object) {
4   const subscribers = new Set<Subscriber>()
5   const proxy = new Proxy(obj, {
6     deleteProperty(target, key) {
7       const previous = target[key as keyof typeof target]
8       const result = Reflect.deleteProperty(target, key)
9       subscribers.forEach((sub) => sub({ key, previous, deleted: true }))
10      return result
11    },
12    ...
13  }) as Subscribable
14
15  proxy.$subscribe = ...
16
17  return proxy
18 }
```

» [mini-observables.ts](#)

# Mini observables

```
1  type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
2
3  function makeObservable(obj: object) {
4    const subscribers = new Set<Subscriber>()
5    const proxy = new Proxy(obj, {
6      deleteProperty(target, key) {
7        const previous = target[key as keyof typeof target]
8        const result = Reflect.deleteProperty(target, key)
9        subscribers.forEach((sub) => sub({ key, previous, deleted: true }))
10       return result
11     },
12     ...
13   }) as Subscribable
14
15   proxy.$subscribe = ...
16
17   return proxy
18 }
```

» [mini-observables.ts](#)

# Mini observables

```
1 type Notification =
2   | { key: PropertyKey, previous: unknown, value: unknown }
3   | { key: PropertyKey, previous: unknown, deleted: true }
4 type Subscriber = (notif: Notification) => void
5 type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }
6
7 function makeObservable(obj: object) {
8   const subscribers = new Set<Subscriber>()
9   const proxy = ...
10
11   proxy.$subscribe = (subscriber) => {
12     subscribers.add(subscriber)
13     return proxy
14   }
15
16   return proxy
17 }
```

» [mini-observables.ts](#)

# Mini observables

```
1 const propLog: PropertyKey[] = []
2 const foo = makeObservable({ name: 'dotJS', ranking: '👍' })
3   .$subscribe(console.debug)
4   .$subscribe(({ key }) => propLog.push(key))
5
6 // foo.name += ' 2025'
7 // foo.ranking = '⭐'
8 // foo.hasAfterParty = true
9 // delete foo.hasAfterParty
10 // console.log(propLog)
11
```



» [mini-observables.ts](#)

# Shout-out: Vue.js 3 relies on ES proxies

```
<script setup>
import { reactive } from 'vue'

const state = reactive({ count: 0 })
</script>

<template>
  <button @click="state.count += 1">
    {{ state.count }}
  </button>
</template>
```

It's more powerful and performant than Vue 2's reactivity. For instance, no custom syntax for reacting to property deletion (`deleteProperty` trap), no need to have all properties defined from the start either (the `set` trap triggers on both existing and new properties).

# TTL Refs

```
1  function makeTTL(fx: Function, ttlMS: number) {
2    const expiry = Date.now() + ttlMS
3    const fxName = fx.name || fx['displayName'] || 'Function'
4
5    return new Proxy(fx, {
6      apply(target, thisArg, args) {
7        if (Date.now() ≥ expiry) {
8          throw new Error(`#${fxName} cannot be called anymore (${ttlMS}ms TTL expired)`)
9        }
10
11      return Reflect.apply(target, thisArg, args)
12    }
13  })
14}
```

» [ttl-refs.ts](#)

# TTL Refs

```
1  function makeTTL(fx: Function, ttlMS: number) {
2      const expiry = Date.now() + ttlMS
3      const fxName = fx.name || fx['displayName'] || 'Function'
4
5      return new Proxy(fx, {
6          apply(target, thisArg, args) {
7              if (Date.now() ≥ expiry) {
8                  throw new Error(`#${fxName} cannot be called anymore (${ttlMS}ms TTL expired)`)
9              }
10
11             return Reflect.apply(target, thisArg, args)
12         }
13     })
14 }
```

» [ttl-refs.ts](#)

# TTL Refs

```
1  function sayHi(whom: string) { console.log(`Hi ${whom}!`) }
2  const sayHiFast = makeTTL(sayHi, 500)
3
4  sayHiFast('John')
5  // ✅ Hi John
6
7  setTimeout(() => sayHiFast('Nessrine'), 400)
8  // ✅ Hi Nessrine!
9
10 setTimeout(() => sayHiFast('Sylvain'), 500)
11 // ❗ sayHi cannot be called anymore (500ms TTL expired)
12
```



```
Hi John!
+416ms: Hi Nessrine!
+517ms: Error: "sayHi cannot be called anymore (500ms TTL expired)"
```

» [ttl-refs.ts](#)

# “URL builder”

```
1 // We aim for this:  
2 // const builder = makeURLBuilder('https://api.acme.biz')  
3 // builder.buy.bacon.and.eggs() => https://api.acme.biz/buy/bacon/and/eggs  
4  
5 function makeURLBuilder(domain: string): URLBuilder {  
6   const parts: string[] = []  
7  
8   return new Proxy(buildURL, {  
9     has: () => true,  
10    get(_target, prop, receiver) {  
11      parts.push(prop as string)  
12      return receiver  
13    },  
14  }) as URLBuilder  
15  
16  ...  
17}
```

» [url-builder.ts](#)

# “URL builder”

```
1 // We aim for this:  
2 // const builder = makeURLBuilder('https://api.acme.biz')  
3 // builder.buy.bacon.and.eggs() => https://api.acme.biz/buy/bacon/and/eggs  
4  
5 function makeURLBuilder(domain: string): URLBuilder {  
6   const parts: string[] = []  
7  
8   return new Proxy(buildURL, {  
9     has: () => true,  
10    get(_target, prop, receiver) {  
11      parts.push(prop as string)  
12      return receiver  
13    },  
14  }) as URLBuilder  
15  
16  ...  
17}
```

» [url-builder.ts](#)

# “URL builder”

```
1 // We aim for this:  
2 // const builder = makeURLBuilder('https://api.acme.biz')  
3 // builder.buy.bacon.and.eggs() => https://api.acme.biz/buy/bacon/and/eggs  
4  
5 function makeURLBuilder(domain: string): URLBuilder {  
6   const parts: string[] = []  
7  
8   return new Proxy(buildURL, {  
9     has: () => true,  
10    get(_target, prop, receiver) {  
11      parts.push(prop as string)  
12      return receiver  
13    },  
14  }) as URLBuilder  
15  
16  ...  
17}
```

» [url-builder.ts](#)

# “URL builder”

```
1 // We aim for this:  
2 // const builder = makeURLBuilder('https://api.acme.biz')  
3 // builder.buy.bacon.and.eggs() => https://api.acme.biz/buy/bacon/and/eggs  
4  
5 function makeURLBuilder(domain: string): URLBuilder {  
6   const parts: string[] = []  
7  
8   ...  
9  
10  function buildURL(qs: object = {}) {  
11    const url = new URL([domain, ...parts].join('/'))  
12    for (const [key, value] of Object.entries(qs)) {  
13      url.searchParams.append(key, value)  
14    }  
15    parts.length = 0  
16    return url.toString()  
17  }  
18}
```

» [url-builder.ts](#)

# “URL builder”

```
1 const builder = makeURLBuilder('https://api.acme.biz')
2 console.log(builder.buy.bacon.and.eggs())
3
4 console.log(builder.categories[124].discounts({ mode: 'sale', pageSize: 10 }))
5
```



<https://api.acme.biz/buy/bacon/and/eggs>

<https://api.acme.biz/categories/124/discounts?mode=sale&pageSize=10>

» [url-builder.ts](#)

# VCR (record & replay)

Here's what we'd like to be able to do.

```
1  const { vcr, replay } = makeVCR()
2  vcr.foo.bar.baz({ name: 'hello' })
3  vcr.foo.quux = 42
4  vcr.foo.doo('amazing').name = 'YOLO'
5
6  const subject: DemoSubject = {
7    foo: {
8      bar: { baz({ name }: { name: string }): { this.name = name } },
9      doo(text: string) { this.kind = text.toUpperCase(); return this },
10     },
11   }
12
13  replay(subject)
```

» [vcr.ts](#)

# VCR (record & replay)

Here's what we'd like to be able to do.

```
1  const { vcr, replay } = makeVCR()
2  vcr.foo.bar.baz({ name: 'hello' })
3  vcr.foo.quux = 42
4  vcr.foo.doo('amazing').name = 'YOLO'
5
6  const subject: DemoSubject = {
7    foo: {
8      bar: { baz({ name }: { name: string }): { this.name = name } },
9      doo(text: string) { this.kind = text.toUpperCase(); return this },
10     },
11   }
12
13  replay(subject)
```

» [vcr.ts](#)

# VCR (record & replay)

Here's what we'd like to be able to do.

```
1  const { vcr, replay } = makeVCR()
2  vcr.foo.bar.baz({ name: 'hello' })
3  vcr.foo.quux = 42
4  vcr.foo.doo('amazing').name = 'YOLO'
5
6  const subject: DemoSubject = {
7    foo: {
8      bar: { baz({ name }: { name: string }): { this.name = name } },
9      doo(text: string) { this.kind = text.toUpperCase(); return this },
10     },
11   }
12
13  replay(subject)
```

» [vcr.ts](#)

# VCR (record & replay)

Here's what we'd like to be able to do.

```
1  const { vcr, replay } = makeVCR()
2  vcr.foo.bar.baz({ name: 'hello' })
3  vcr.foo.quux = 42
4  vcr.foo.doo('amazing').name = 'YOLO'
5
6  const subject: DemoSubject = {
7    foo: {
8      bar: { baz({ name }: { name: string }): { this.name = name } },
9      doo(text: string) { this.kind = text.toUpperCase(); return this },
10     },
11   }
12
13 replay(subject)
```

» [vcr.ts](#)

# VCR (record & replay)

```
1  function makeVCR({
2    pathPrefix = [],
3    ops = [],
4  }: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {
5    const vcr = new Proxy(() => {}, {
6      get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,
7      apply(_, __, args) {
8        const propPath = [...pathPrefix, { args }]
9        ops.push({ kind: 'read', prop: propPath })
10       return makeVCR({ pathPrefix: propPath, ops }).vcr
11     },
12     set(_, prop, value) {
13       ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })
14       return true
15     },
16   }) as VCR
17
18   return { vcr, replay: $replay.bind(null, ops) }
19 }
```

# VCR (record & replay)

```
1  function makeVCR({  
2    pathPrefix = [],  
3    ops = [],  
4  }: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {  
5    const vcr = new Proxy(() => {}, {  
6      get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,  
7      apply(_, __, args) {  
8        const propPath = [...pathPrefix, { args }]  
9        ops.push({ kind: 'read', prop: propPath })  
10       return makeVCR({ pathPrefix: propPath, ops }).vcr  
11     },  
12      set(_, prop, value) {  
13        ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })  
14        return true  
15      },  
16    }) as VCR  
17  
18    return { vcr, replay: $replay.bind(null, ops) }  
19 }
```

# VCR (record & replay)

```
1  function makeVCR({  
2    pathPrefix = [],  
3    ops = [],  
4  }: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {  
5    const vcr = new Proxy(() => {}, {  
6      get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,  
7      apply(_, __, args) {  
8        const propPath = [...pathPrefix, { args }]  
9        ops.push({ kind: 'read', prop: propPath })  
10       return makeVCR({ pathPrefix: propPath, ops }).vcr  
11     },  
12      set(_, prop, value) {  
13        ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })  
14        return true  
15      },  
16    }) as VCR  
17  
18    return { vcr, replay: $replay.bind(null, ops) }  
19 }
```

# VCR (record & replay)

```
1  function makeVCR({
2    pathPrefix = [],
3    ops = [],
4  }: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {
5    const vcr = new Proxy(() => {}, {
6      get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,
7      apply(_, __, args) {
8        const propPath = [...pathPrefix, { args }]
9        ops.push({ kind: 'read', prop: propPath })
10       return makeVCR({ pathPrefix: propPath, ops }).vcr
11     },
12      set(_, prop, value) {
13        ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })
14        return true
15      },
16    }) as VCR
17
18    return { vcr, replay: $replay.bind(null, ops) }
19 }
```

# VCR (record & replay)

```
1  function makeVCR({  
2    pathPrefix = [],  
3    ops = [],  
4  }: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {  
5    const vcr = new Proxy(() => {}, {  
6      get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,  
7      apply(_, __, args) {  
8        const propPath = [...pathPrefix, { args }]  
9        ops.push({ kind: 'read', prop: propPath })  
10       return makeVCR({ pathPrefix: propPath, ops }).vcr  
11     },  
12      set(_, prop, value) {  
13        ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })  
14        return true  
15      },  
16    }) as VCR  
17  
18    return { vcr, replay: $replay.bind(null, ops) }  
19 }
```

# VCR (record & replay)



```
1 const { vcr, replay } = makeVCR()
2 vcr.foo.bar.baz({ name: 'hello' })
3 vcr.foo.quux = 42
4 vcr.foo.doo('amazing').name = 'YOLO'
5
6 const subject: DemoSubject = {
7   foo: {
8     bar: { baz({ name }: { name: string }): { this.name = name } },
9     doo(text: string) { this.kind = text.toUpperCase(); return this },
10   },
11 }
12
13 replay(subject)
14 console.log(subject)

{ "foo": { "bar": { "name": "hello" }, "quux": 42, "kind": "AMAZING", "name": "YOLO" } }
```

» [vcr.ts](#)

# Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// Without Immer

const nextState = [...baseState]
nextState[1] = { ...nextState[1], done: true }
nextState.push({ title: 'Tweet about it' })
```

# Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// Without Immer

const nextState = [...baseState]
nextState[1] = { ...nextState[1], done: true }
nextState.push({ title: 'Tweet about it' })
```

# Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// Without Immer

const nextState = [...baseState]
nextState[1] = { ...nextState[1], done: true }
nextState.push({ title: 'Tweet about it' })
```

# Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// Without Immer

const nextState = [...baseState]
nextState[1] = { ...nextState[1], done: true }
nextState.push({ title: 'Tweet about it' })
```

# Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// With Immer
import { produce } from 'immer'

const nextState = produce(baseState, (draft) => {
  draft[1].done = true
  draft.push({ title: 'Tweet about it' })
})

// `baseState` is unchanged
// `nextState` is the deep, copy-on-write variant
```

# Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// With Immer
import { produce } from 'immer'

const nextState = produce(baseState, (draft) => {
  draft[1].done = true
  draft.push({ title: 'Tweet about it' })
})

// `baseState` is unchanged
// `nextState` is the deep, copy-on-write variant
```

# Shout-out: Immer.js is amazing

It's also great in combination with React, with or without Redux. Basic example:

```
const [todos, setTodos] = useImmer([
  { id: 'React', title: 'Learn React', done: true },
  { id: 'Immer', title: 'Try Immer', done: false }
])

function handleToggle(id) {
  setTodos((draft) => {
    const todo = draft.find((todo) => todo.id === id)
    todo.done = !todo.done
  })
}
```

Also has built-in solutions for `useReducer`, Redux reducers, etc.

FIXME: try PDF build FIXME: create repo, deploy, setup GH pages



# Thank you!



These slides are on [bit.ly/dotjs-esproxies](https://bit.ly/dotjs-esproxies)

Christophe: [@porteneuve@piaille.fr](mailto:@porteneuve@piaille.fr) / [LinkedIn](#)

Photo credits: Cloudy sky par [mosi knife](#), Confusion by [Ayo Ogunsende](#) and toys by [Chris Hardy](#), all sources from [Unsplash](#).