Automated Bug Detection for JavaScript

Anders Møller

CENTER FOR ADVANCED SOFTWARE ANALYSIS

http://casa.au.dk/

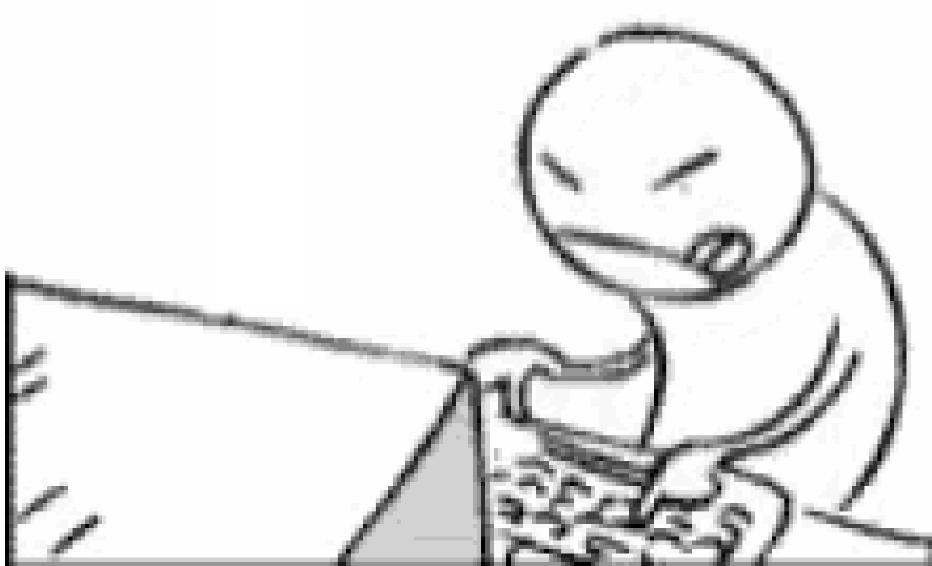
AARHUS UNIVERSITY

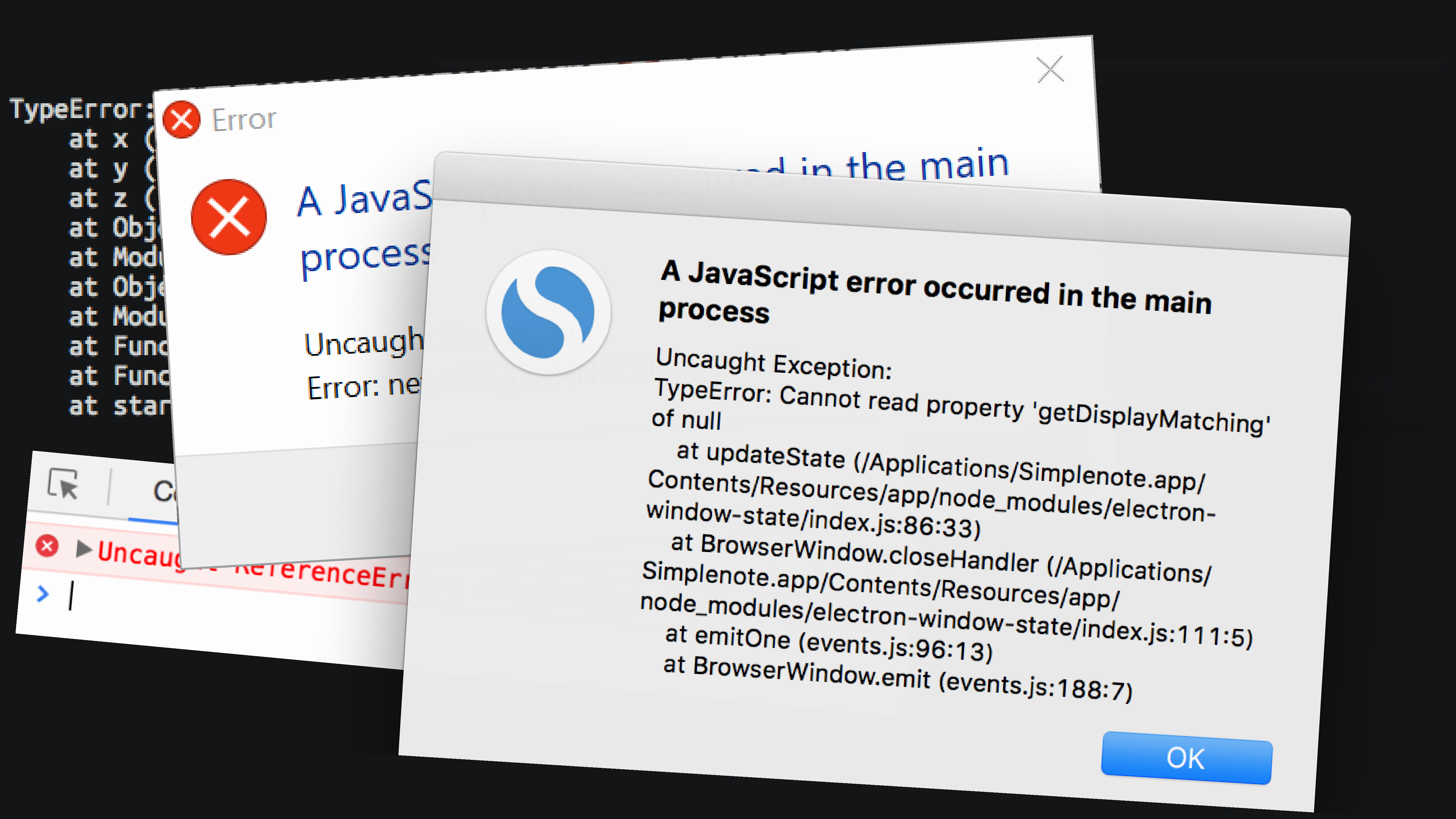
JS JavaScript

- The most popular programming language
  (according to Stack Overflow's 2019 survey)

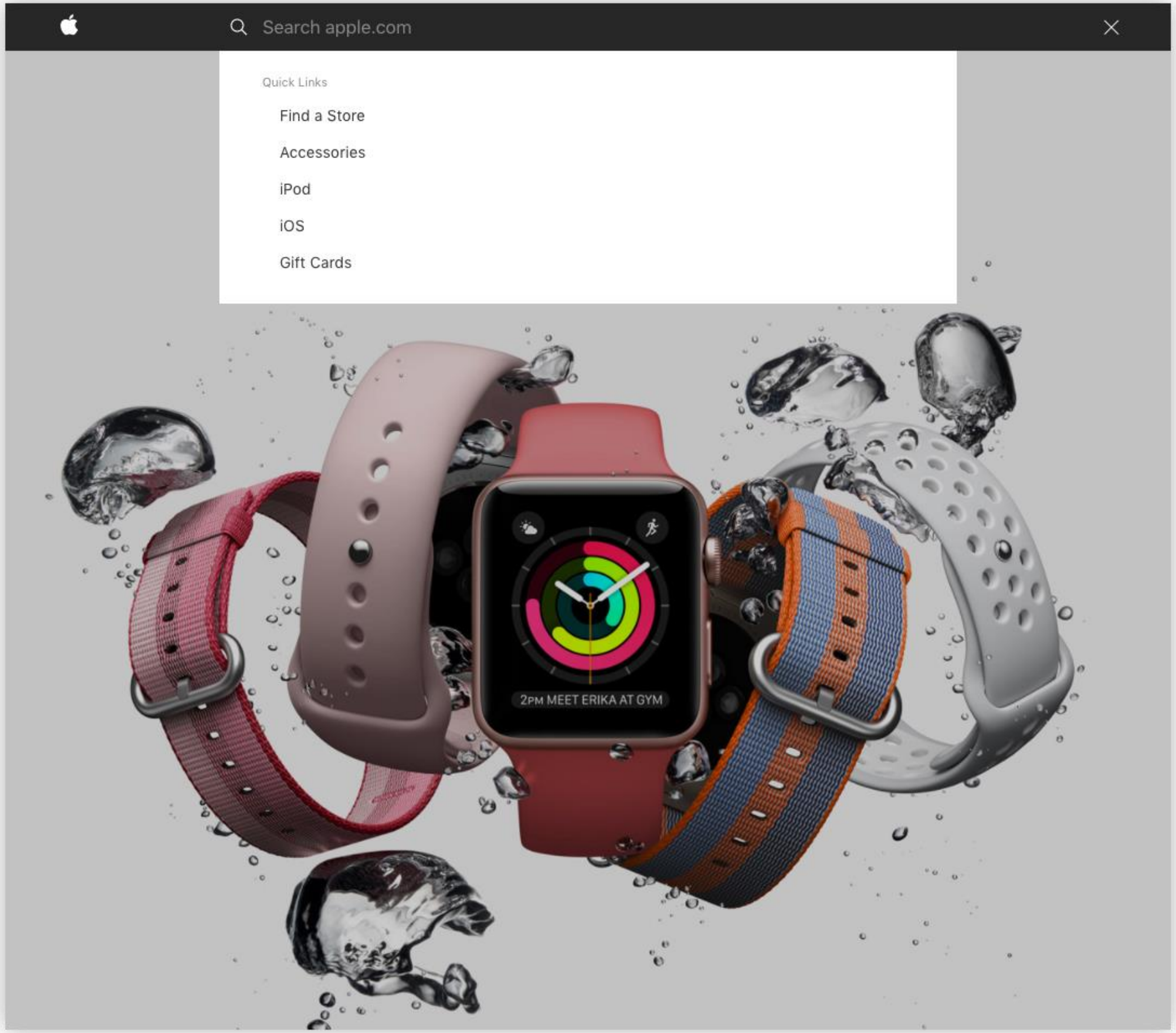
- **Object-oriented + functional + event-driven**

- ***Testing and evolving JavaScript programs is difficult!***

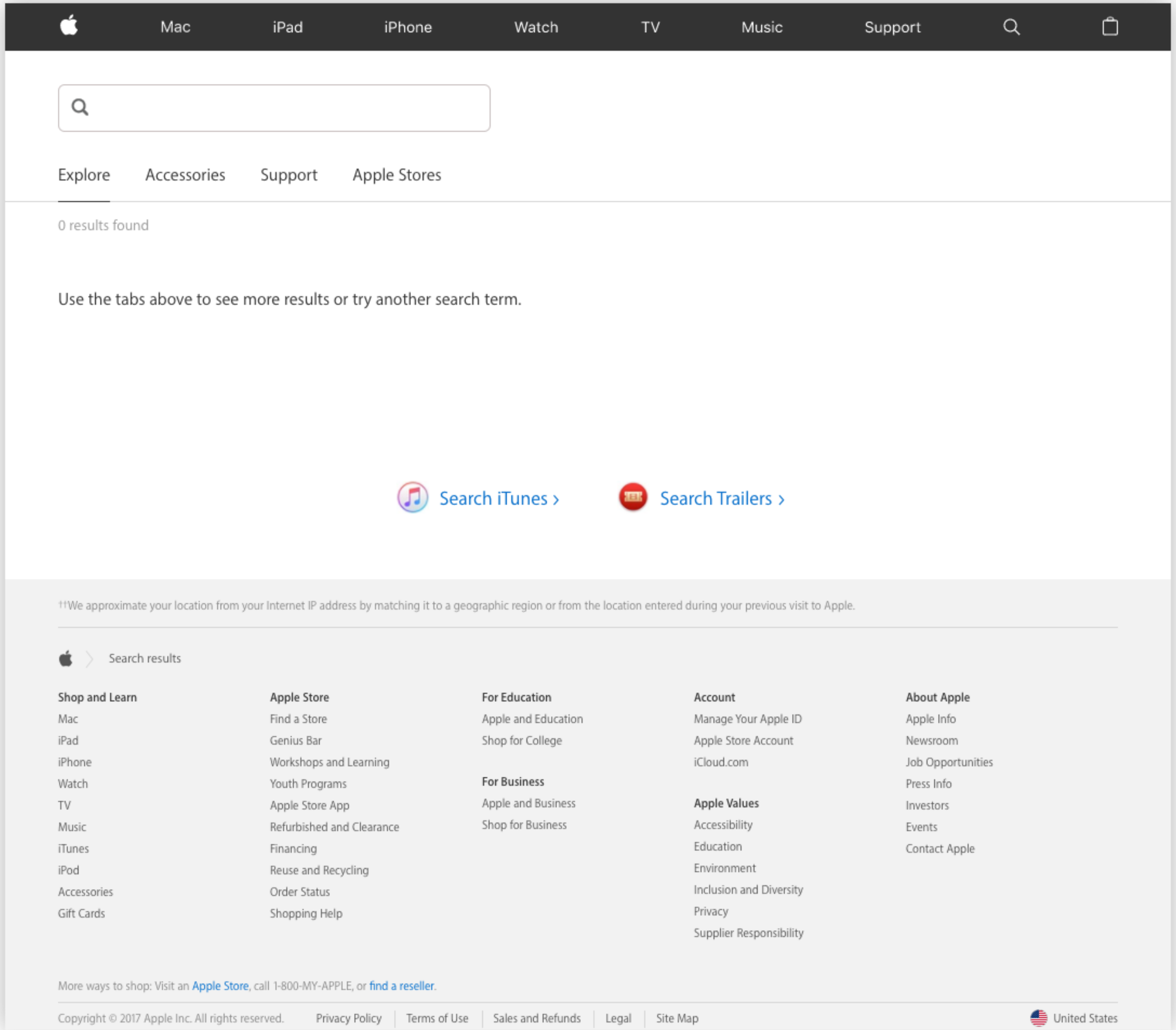



Error
A JavaScript error occurred in the main process
Uncaught Exception:
TypeError: Cannot read property 'getDisplayMatching' of null
at updateState (/Applications/Simplenote.app/Contents/Resources/app/node_modules/electron-window-state/index.js:86:33)
at BrowserWindow.closeHandler (/Applications/Simplenote.app/Contents/Resources/app/node_modules/electron-window-state/index.js:111:5)
at emitOne (events.js:96:13)
at BrowserWindow.emit (events.js:188:7)
OK

What Apple's programmers expect you to see when clicking the icon on apple.com



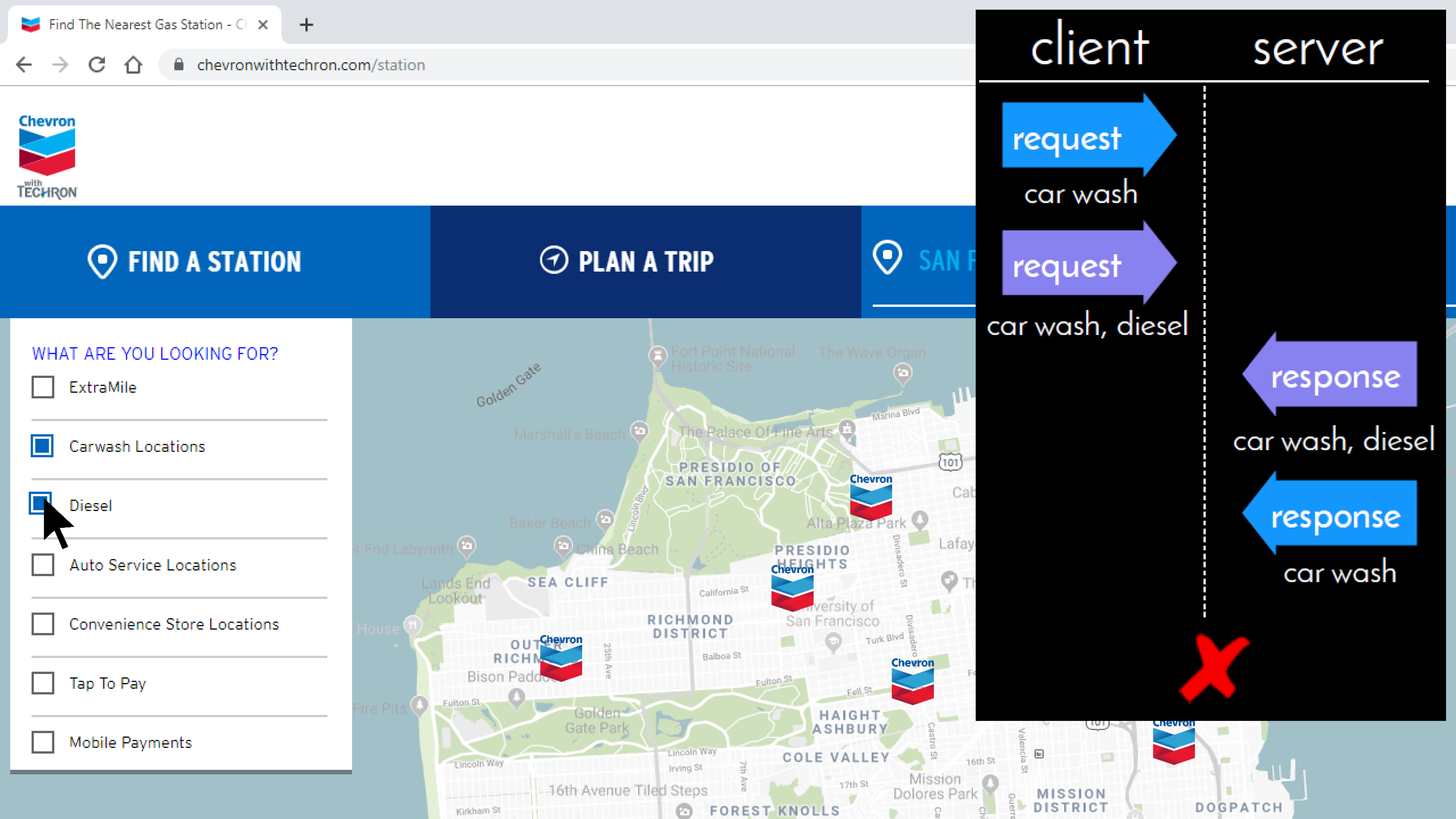
What you see if you click before the page is fully loaded



Research in JavaScript at Aarhus University

European Research Council
Established by the European Commission
Supporting top researchers from anywhere in the world

- Static analysis
- Automated testing



Find The Nearest Gas Station - Cl
chevronwithtechron.com/station
Chevron
with TECHRON
FIND A STATION
PLAN A TRIP
WHAT ARE YOU LOOKING FOR?
ExtraMile
Carwash Locations
Diesel
Auto Service Locations
Convenience Store Locations
Tap To Pay
Mobile Payments
client
server
request
car wash
request
car wash, diesel
response
car wash, diesel
response
car wash

An interesting research challenge:

How to automatically detect
event-race errors
in JavaScript programs?

Detecting event-race errors automatically

Our approach (simplified):

1. Run instrumented program, record event sequence
2. Analyze the event sequence to predict possible problems
3. Simulate reordered event sequences that have
   - the same order of user events (mouse clicks, etc.), but
   - different order of system events (timeouts, HTTP responses, etc.)
4. Look for crashes and other observable differences

Research papers and prototype tools:

https://github.com/cs-au-dk/initracer

https://github.com/cs-au-dk/ajaxracer

- npm is a package manager for JavaScript
- The world's largest software registry
  - >1M packages, mostly libraries
  - some have >10M weekly downloads
  - many packages evolve rapidly, with new features, bugfixes, etc.
  - an application *A* may depend on library *B* that depends on library *C*, etc.

Semantic Versioning

Example: v1.1.0 → v1.2.0 is a *minor* update
that should not cause problems for clients of the package

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards-compatible manner, and
3. PATCH version when you make backwards-compatible bug fixes.

https://semver.org/

How does the package developer know whether a change is an "incompatible API change"?

What is the API of a JavaScript library?

JavaScript is a highly dynamic programming language

- Impossible to determine library interface directly from the syntactic structure of the library code
- No type annotations
- No clear distinction between public and private

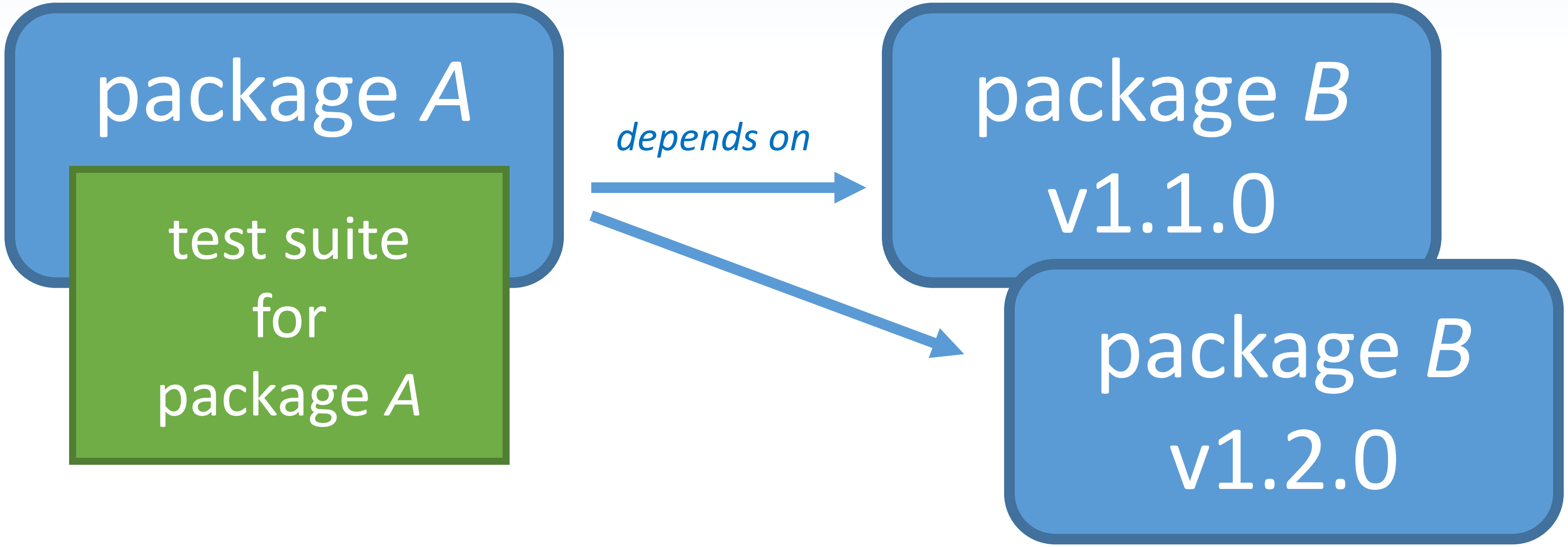
In comparison, Java has static type checking

- Just re-compiling the client reveals many breaking changes
- But only type-related, not semantic properties

Another interesting research challenge:

How to automatically detect
breaking changes
in JavaScript library updates?

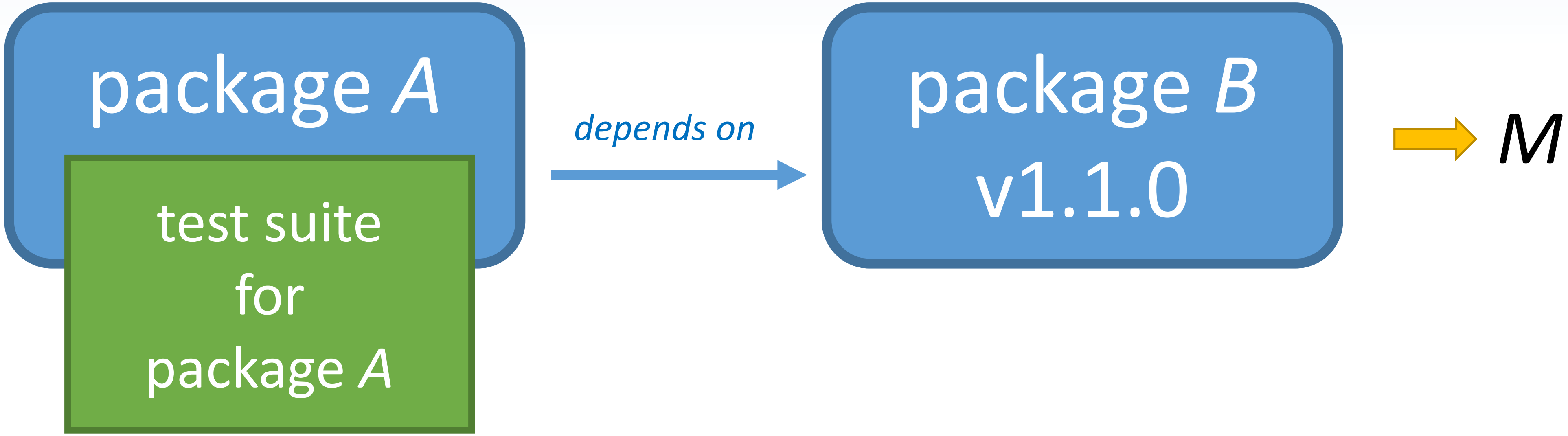
Using client test suites to detect breaking changes



If *A*’s test suite succeeds with v1.1.0 of *B* but fails with v1.2.0 then the update likely contains an “incompatible API change”

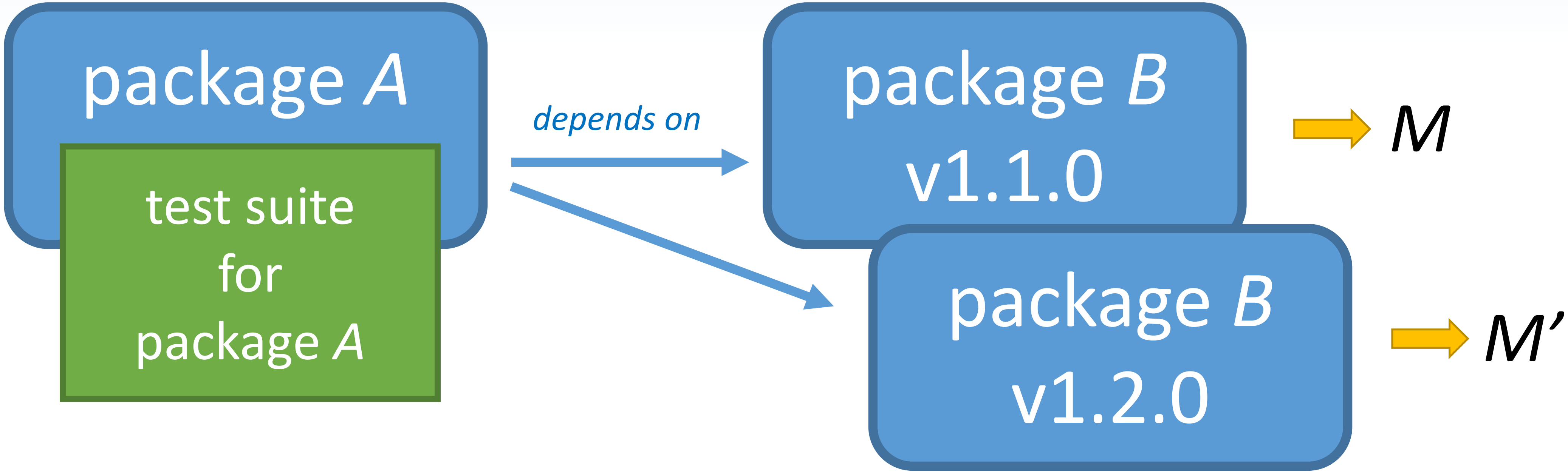
Example: the Lodash package has more than 50 000 direct clients in npm, many with test suites

Our approach



- Run *A*'s test suite, ignore the outcomes of the tests, but dynamically monitor all interactions between *A* and *B*
- Build a **model** *M* of the part of *B*'s interface that is used by *A*
  - like typed method signatures in Java for all public methods
- If many packages with test suites depend on *B*, we can learn its entire API

Our approach



- Similarly, build a model *M'* for the new version of *B*
- Compare *M* and *M'* and report incompatibilities
  - Example: calling B.foo() returns a number with v1.1.0, but a string with v1.2.0
- *Amplifying the existing test suites!*

Some experimental results

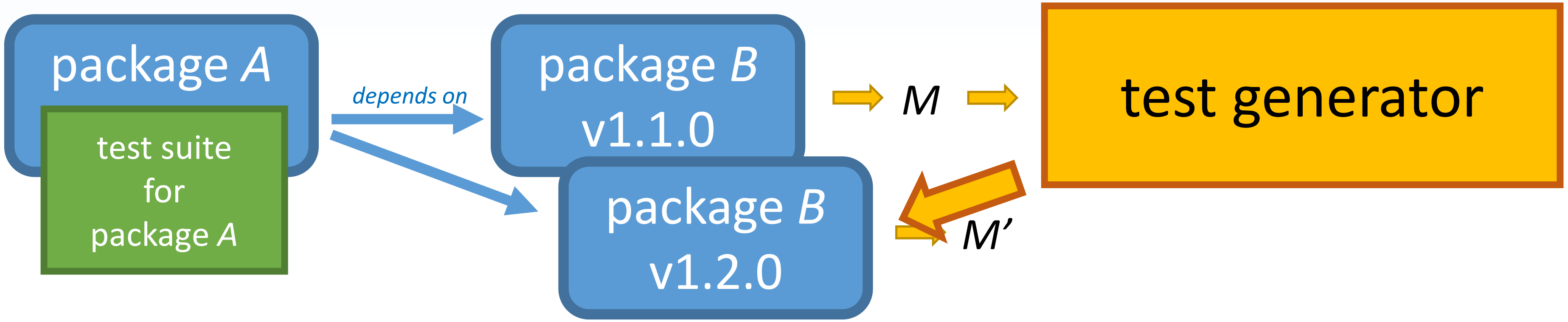
- On 12 popular packages with 389 updates,
  our technique classified more than 90% of the updates correctly
  as major vs. minor

- Automatically detected 26 breaking changes in minor updates!

Research papers and prototype tool:

https://github.com/cs-au-dk/noregrets

Recent work: a model-based testing approach



Instead of producing *M'*, automatically generate tests from *M*

- Faster than running all the test suites!
- Detects even more breaking changes!

Example: big-integer

A test from another library, deposit-iban:

```
const bigInt = require('big-integer');
export function isValidIban(iban) {
  ...
  const bban = ... // '62000000202102329006182700';
  const checkDigitBigInt = bigInt(bban);
  let checkDigitNumber =
    String(98 - checkDigitBigInt.mod(bigInt('97')));
  ...
}
```

Passes with both 1.4.6 and 1.4.7

An automatically generated test by our tool:

```
const bigInt = require('big-integer');
assert(typeof(bigInt('62000000202102329006182700')
                .mod(bigInt('97')).valueOf())
        === "number")
```

Passes with 1.4.6 but fails with 1.4.7 – ***automatically detected a breaking change!***

big-integer v1.4.6

```
function parseValue (v) {
  ...
  return new BigInteger(...);
}
```

big-integer v1.4.7

```
function parseValue (v) {
  if (isPrecise(v)) {
    return new SmallInteger(v);
  }
  ...
  return new BigInteger(...);
}
```

Research in JavaScript at Aarhus University

Lots of exciting research challenges related to JavaScript software development

We're exploring new techniques to automatically detect different kinds of bugs:

- ✶ **event-race errors** in JavaScript applications, caused by unexpected nondeterminism
- ✶ **breaking changes** in JavaScript library updates

+ much more ☺

CENTER FOR ADVANCED SOFTWARE ANALYSIS
http://casa.au.dk/

AARHUS UNIVERSITY