Synode: Understanding and Automatically Preventing Injection Attacks on Node.js

Cristian-Alexandru Staicu[1] Michael Pradel[1]
Ben Livshits[2]

[1]TU Darmstadt

[2]Imperial College London, Brave Software

February 20, 2018

SOLA SoftwareLab

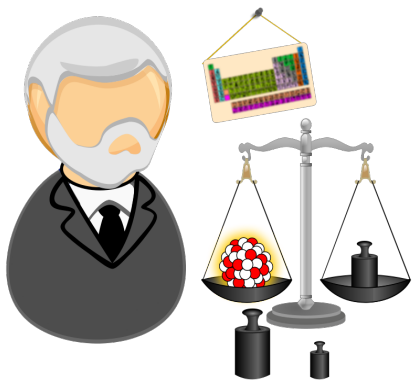
TECHNISCHE UNIVERSITÄT DARMSTADT

This Talk

Node.JS and Injections

Empirical Study

Synode



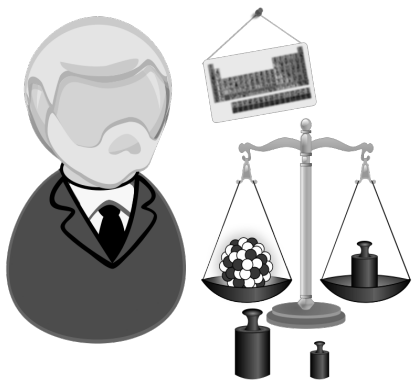
Evaluation

This Talk

Node.JS and Injections

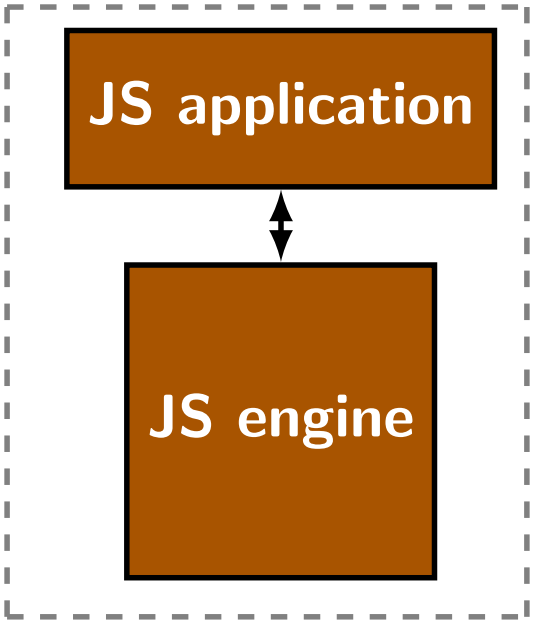
Empirical Study

Synode

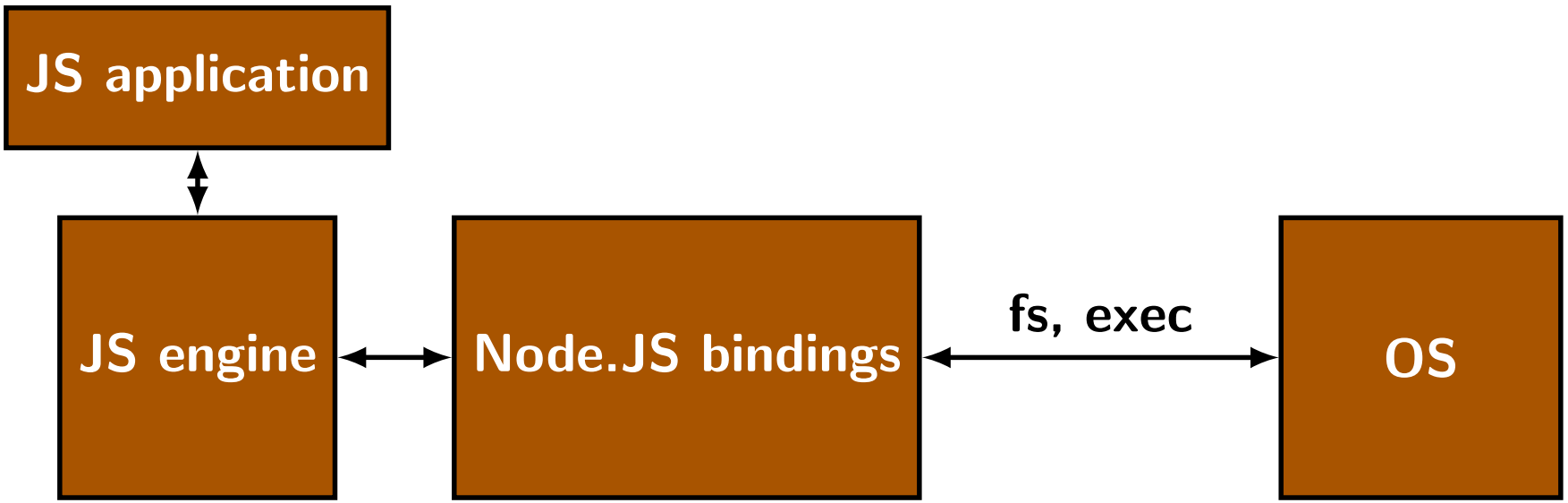


Evaluation

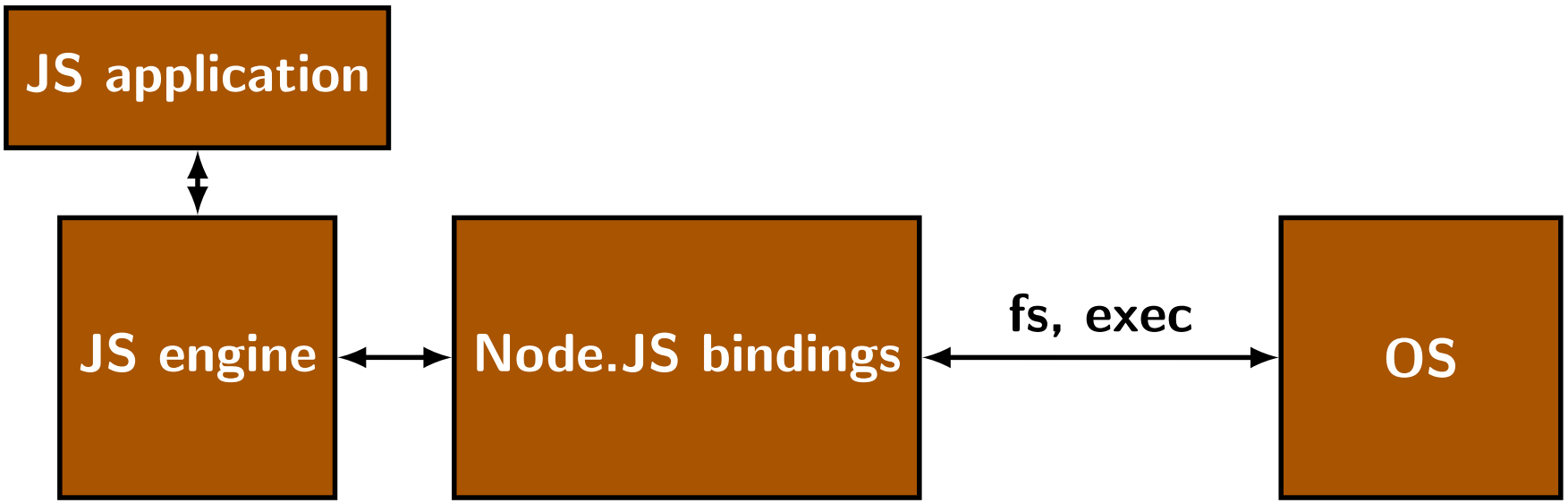
Node.js 101



Node.js 101

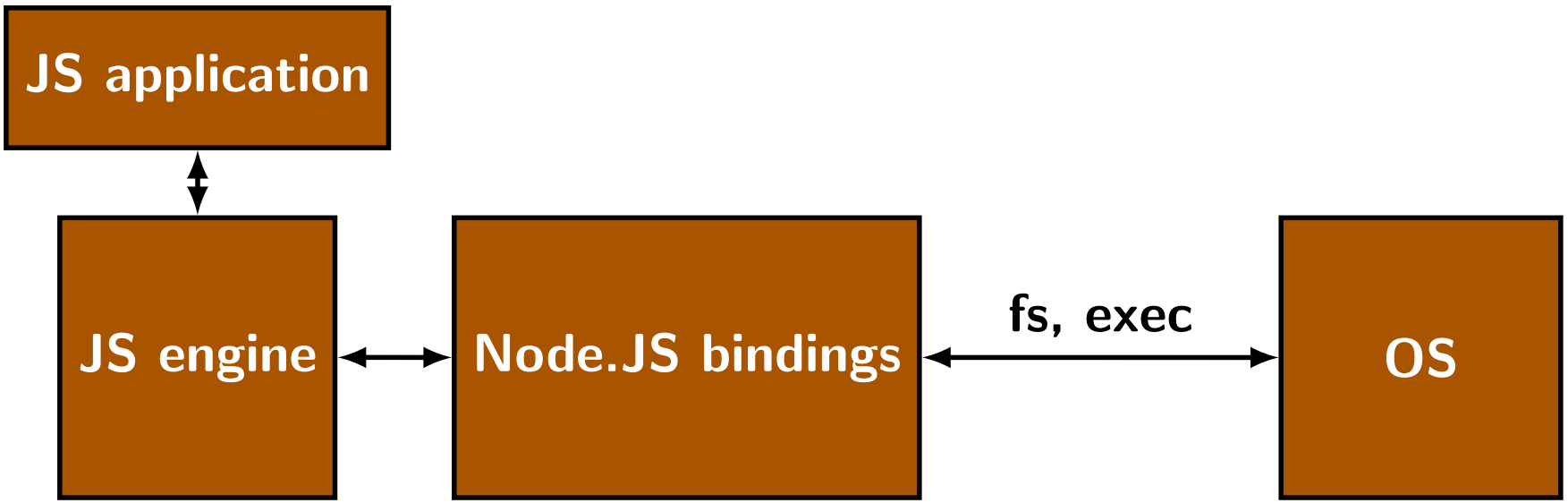


Node.js 101



Node Package Manager

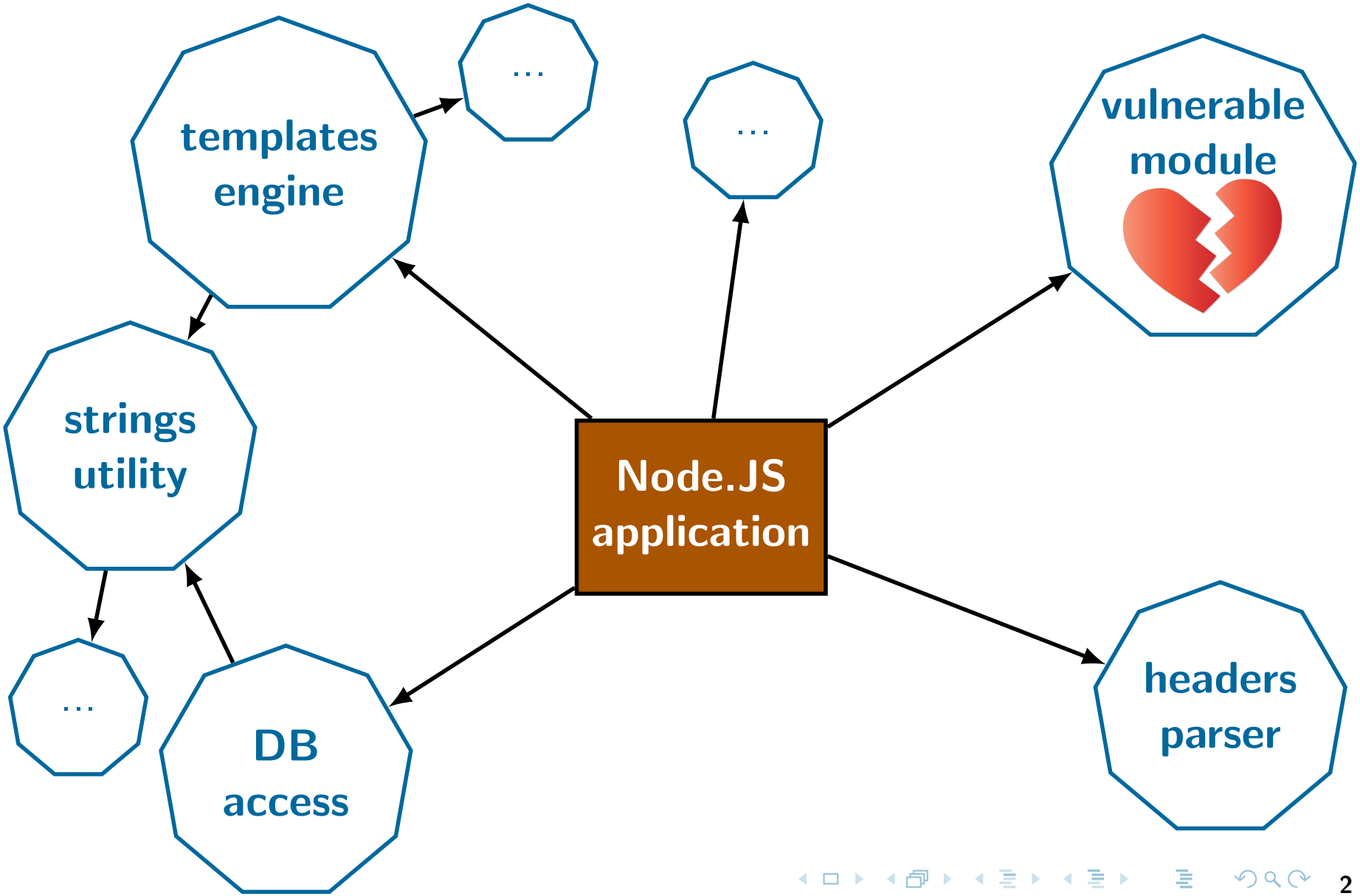
Node.js 101



Node Package Manager

Node Security Project

Typical Node.JS Application



Running Example

```
function backupFile(name, ext) {
  var cmd = [];
  cmd.push("cp");
  cmd.push(name + "." + ext);
  cmd.push("~/.localBackup/");

  exec(cmd.join(" "));
}
```

Running Example

```
function backupFile(name, ext) {
  var cmd = [];
  cmd.push("cp");
  cmd.push(name + "." + ext);
  cmd.push("~/.localBackup/");

  exec(cmd.join(" "));
}
```

Malicious Payload

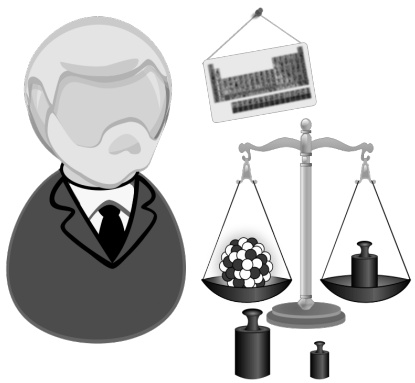
```
backupFile("-h && rm -rf * && echo ", "")
```

This Talk

Node.JS and Injections

Empirical Study

Synode



Evaluation

npm Codebase

236,337
packages

2.471
average number of package dependences

816,840,082
lines of JavaScript code

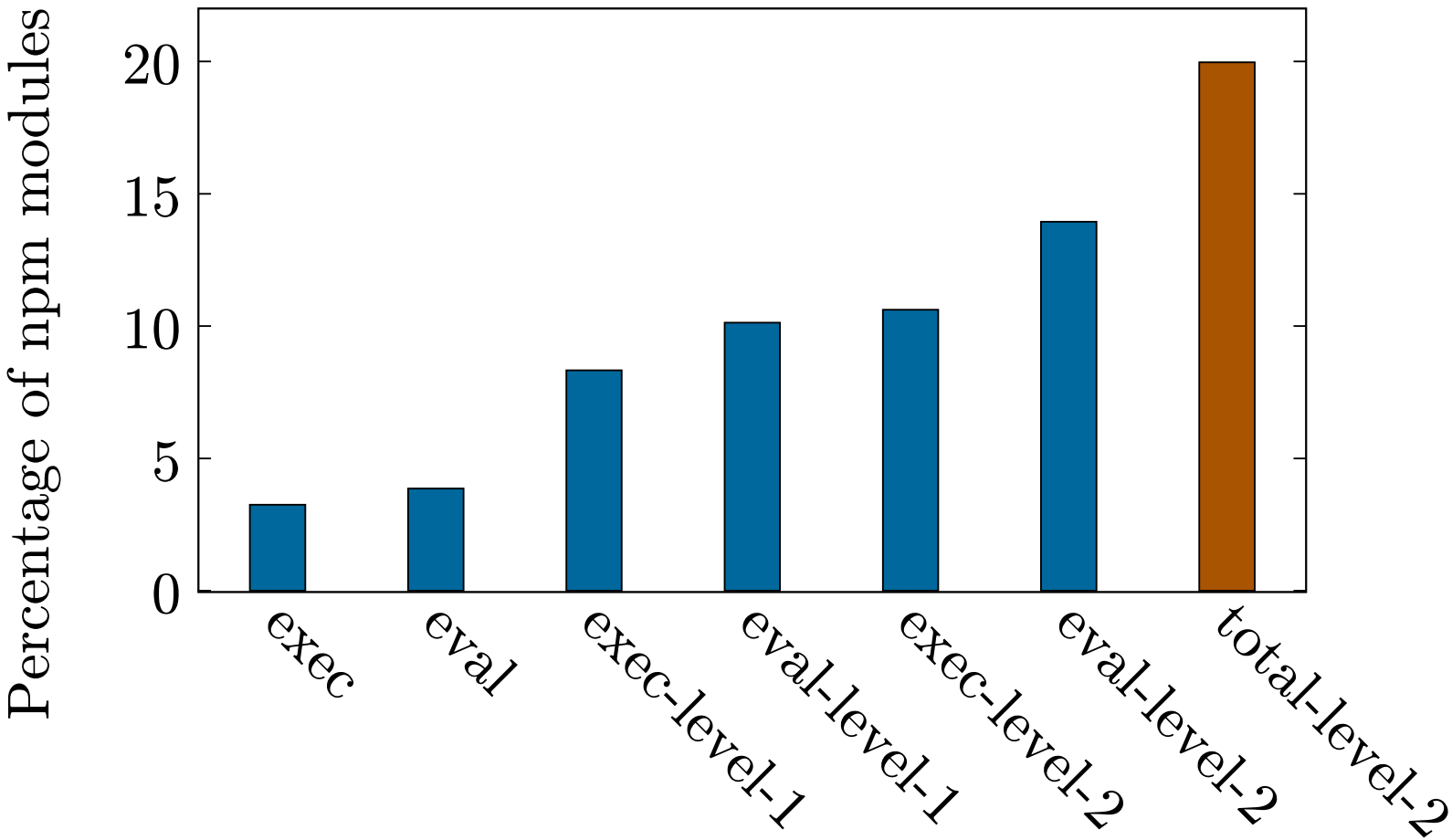
>40,000
C files

7,685
number of packages containing exec

9,110
number of packages containing eval

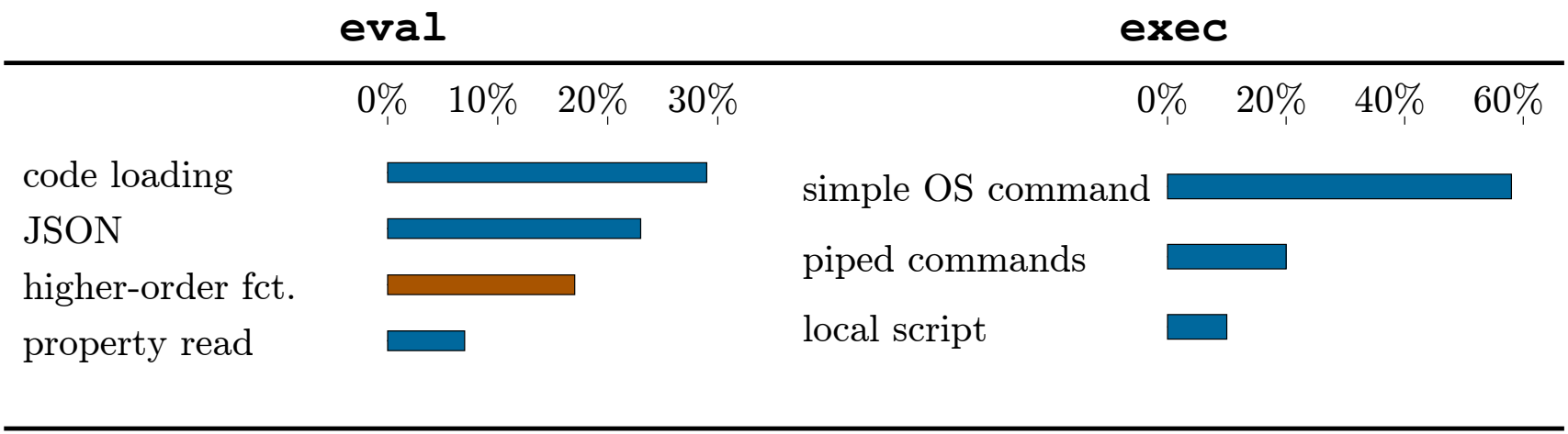
February 2016

Dependences on Injection APIs



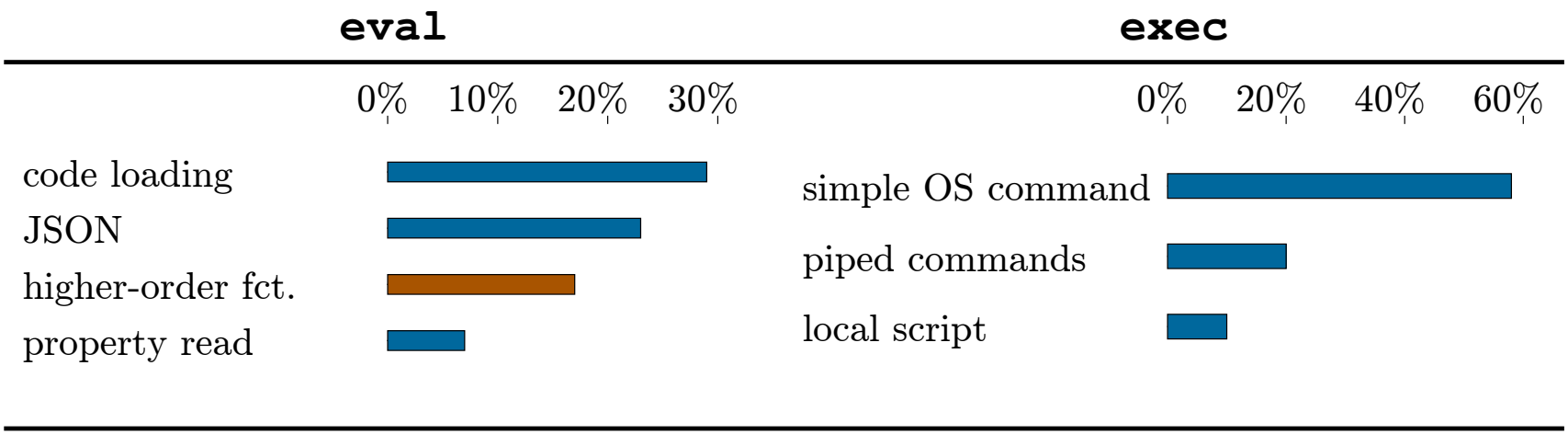
Data Passed to Injection APIs

Manual inspection of **150** call sites



Data Passed to Injection APIs

Manual inspection of **150** call sites



58% contain user-controlled data, out of which:

- **90%** perform no check on this data
- **9%** use regular expressions

Submitted Bug Reports

Affected module	Confirmed	Time until fixed
mixin-pro	yes	1 day
modulify	no	–
proto	yes	155 days*
mongoosify	yes	73 days
summit	yes	–
microservicebus.node	yes	–
mobile-icon-resizer	yes	2 days
m-log	–	–
mongo-edit	–	–
mongo-parse	yes	–
mock2easy	–	–
mongui	–	–
m2m-supervisor	–	–
nd-validator	–	–
nameless-cli	–	–
node-mypeople	–	–
mongoosemask	–	–
kmc	–	–
mod	–	–
growl	yes	–

– indicates a lack response and * an incomplete fix

180 days after reporting

Lessons Learned

multiple dependences

on average each module has 2.5 direct dependences

no sanitization

only 9% use sanitization, often broken

unresponsive developers

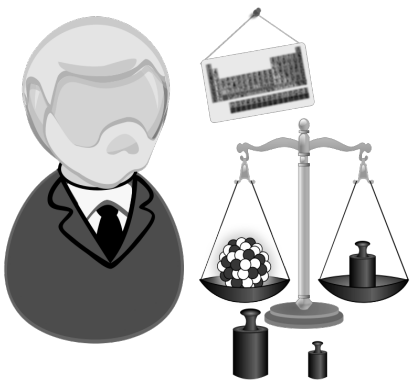
within six months only 25% of the issues were fixed

This Talk

Node.JS and Injections

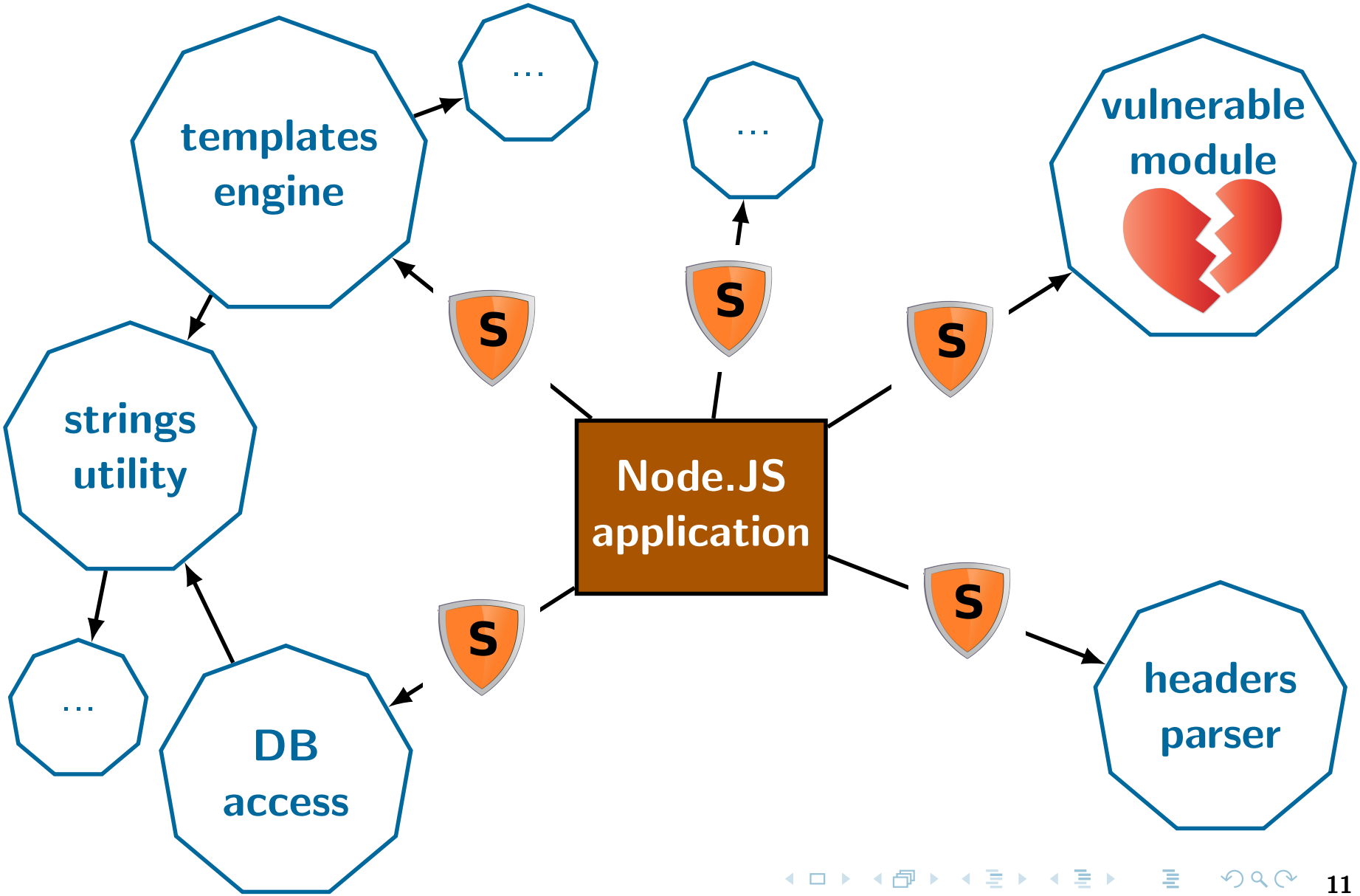
Empirical Study

Synode

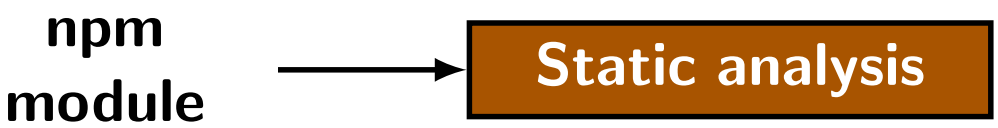


Evaluation

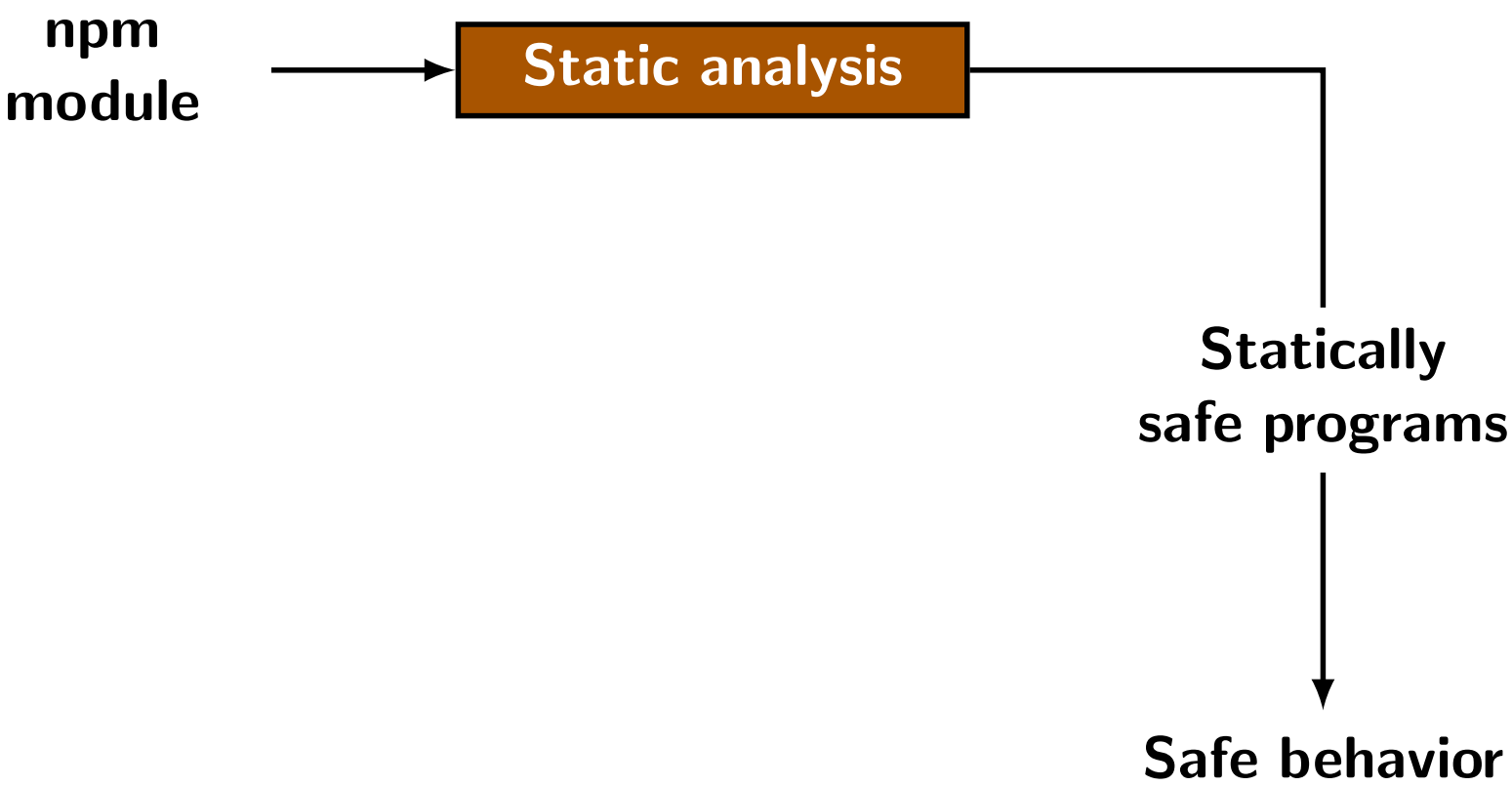
Safe Use of Modules with Synode



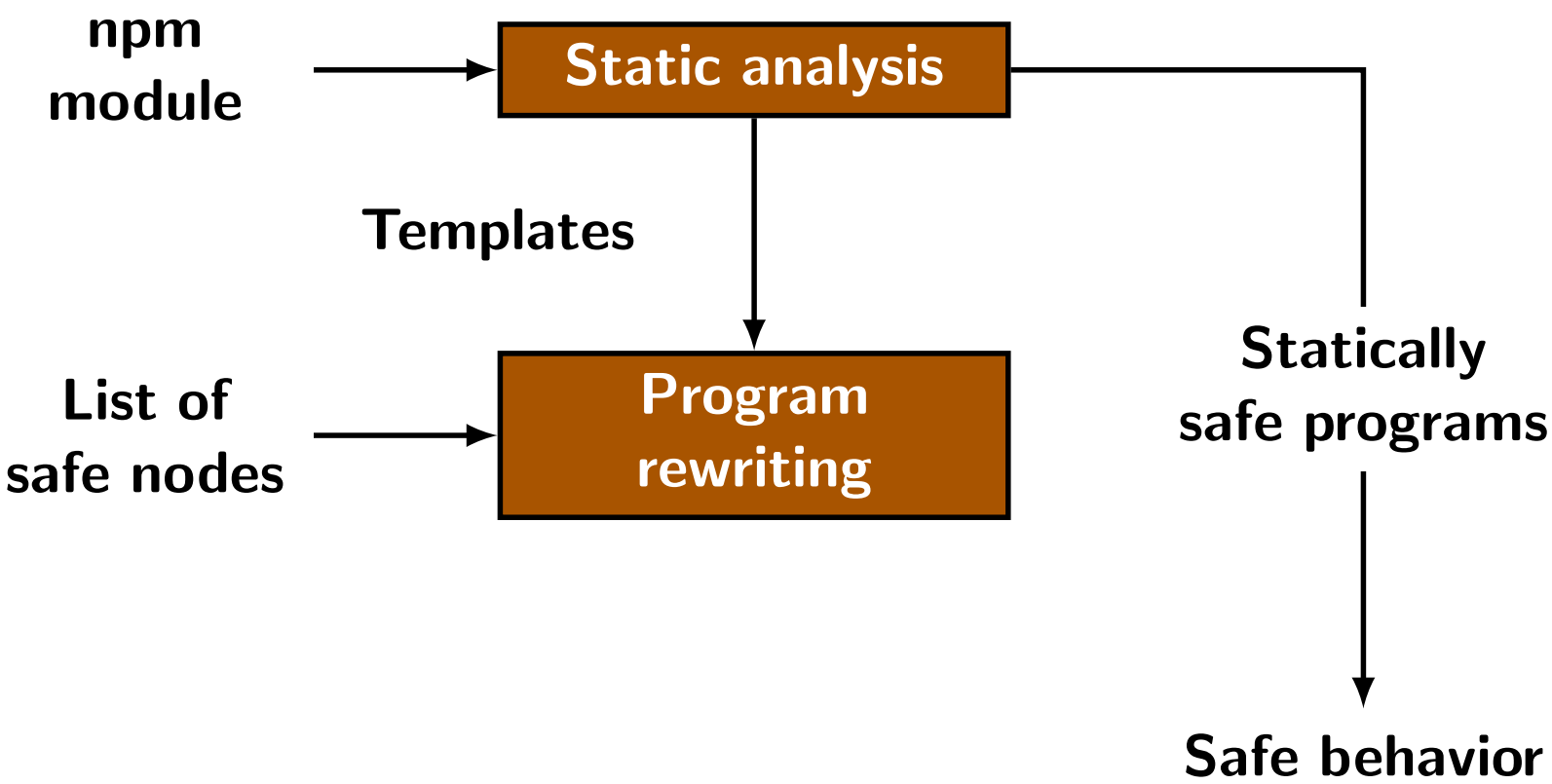
Overview of Synode



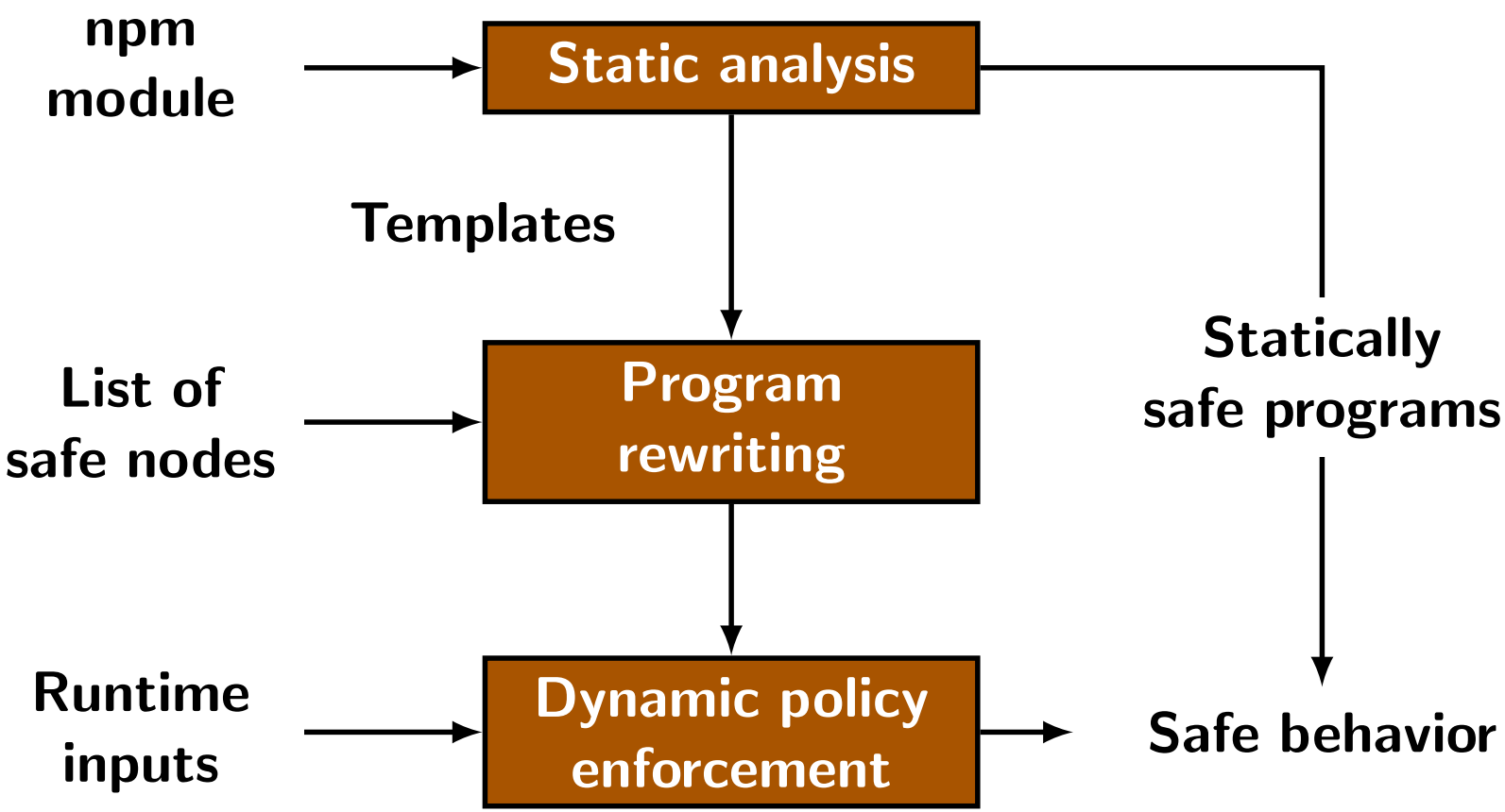
Overview of Synode



Overview of Synode



Overview of Synode



Static Phase

1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime

Static Phase

1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime

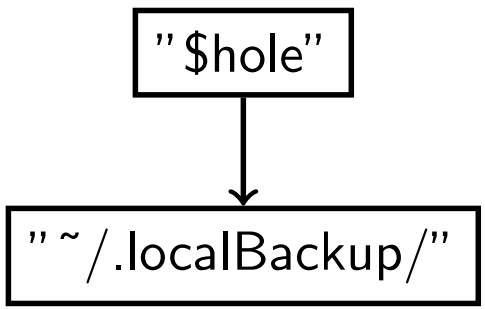
```
function backupFile(name, ext){
    var cmd = [];
    cmd.push("cp");
    cmd.push(name + "." + ext);
    cmd.push("~/.localBackup/");
    exec(cmd.join(" "));
}
```

"$hole"

Static Phase

1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime

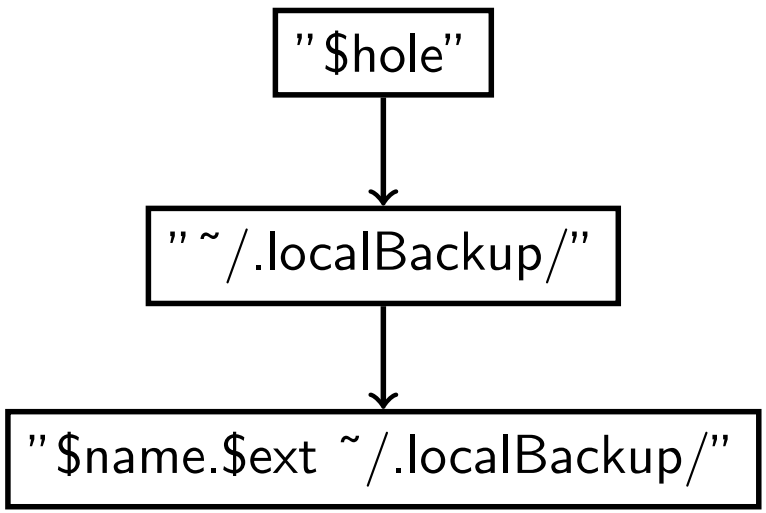
```
function backupFile(name, ext){
    var cmd = [];
    cmd.push("cp");
    cmd.push(name + "." + ext);
    cmd.push("~/.localBackup/");
    exec(cmd.join(" "));
}
```



Static Phase

1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime

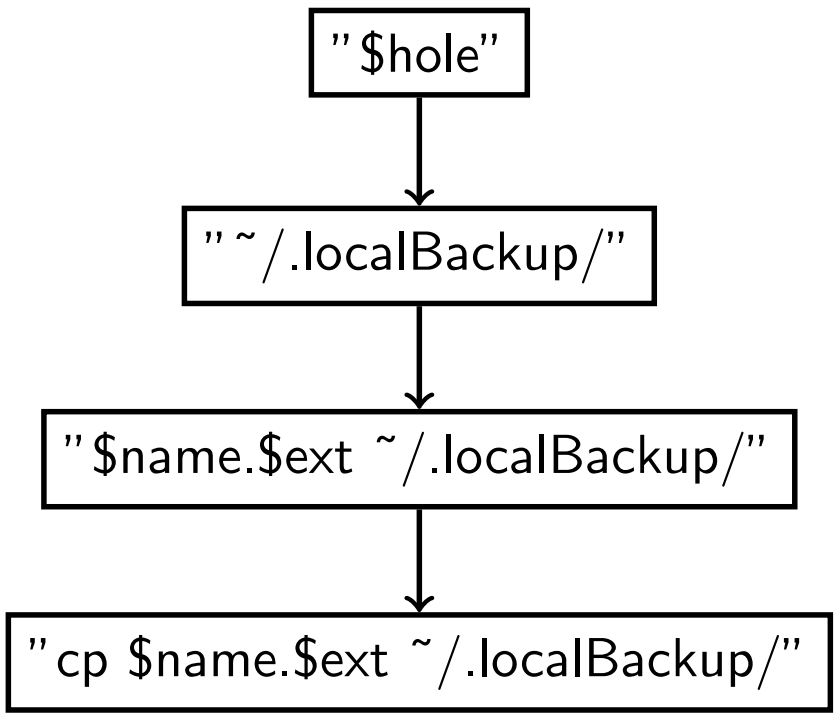
```
function backupFile(name, ext){
    var cmd = [];
    cmd.push("cp");
    cmd.push(name + "." + ext);
    cmd.push("~/.localBackup/");
    exec(cmd.join(" "));
}
```



Static Phase

1. Intra-procedural backward data flow analysis:
   - Over-approximates strings passed to injection APIs
   - Unknown parts to be filled at runtime

```
function backupFile(name, ext){
    var cmd = [];
    cmd.push("cp");
    cmd.push(name + "." + ext);
    cmd.push("~/.localBackup/");
    exec(cmd.join(" "));
}
```



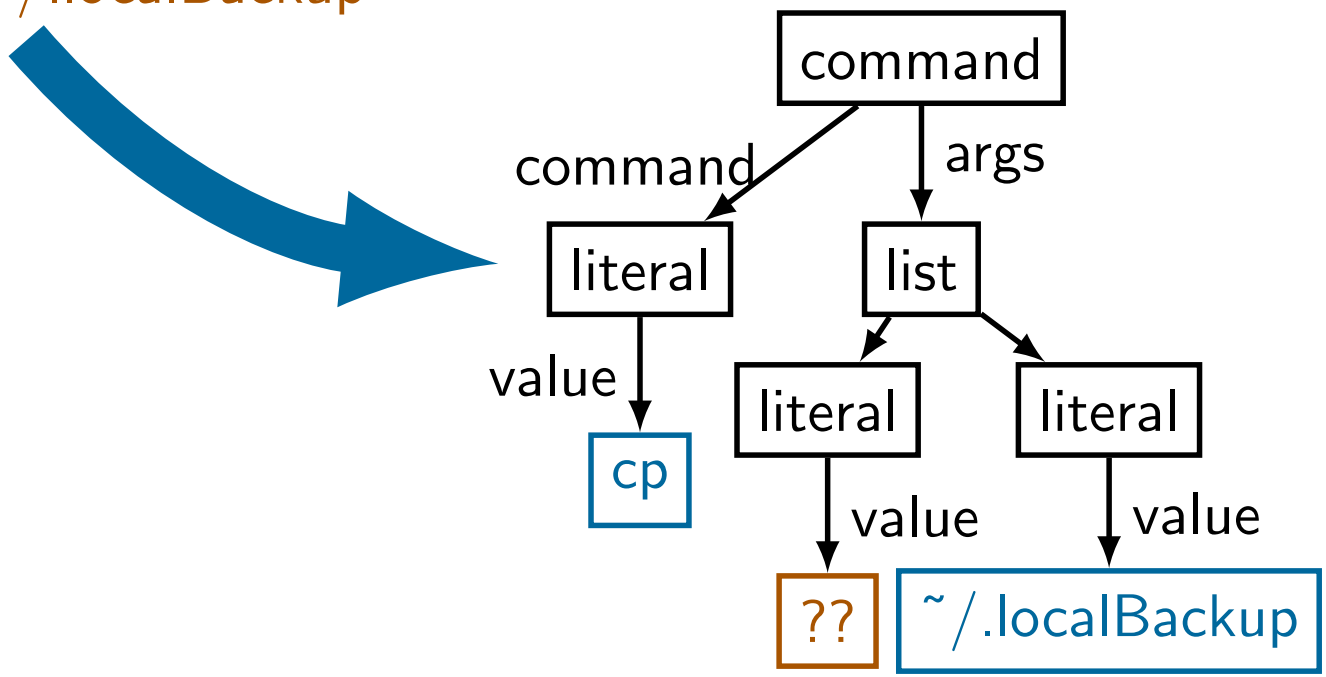
Static Phase

2. Synthesize runtime policy using templates:
   - Enforce structure via partial AST
   - For unknown parts allow only safe nodes

Static Phase

2. Synthesize runtime policy using templates:
   - Enforce structure via partial AST
   - For unknown parts allow only safe nodes

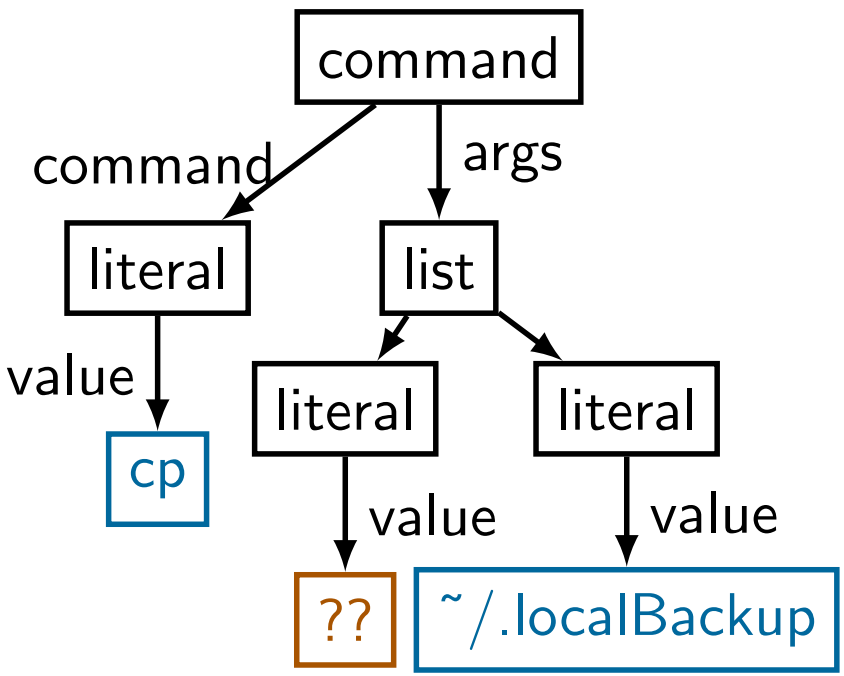
"cp $name.$ext ~/.localBackup"



Runtime Phase

Enforce policy on strings passed to injection APIs

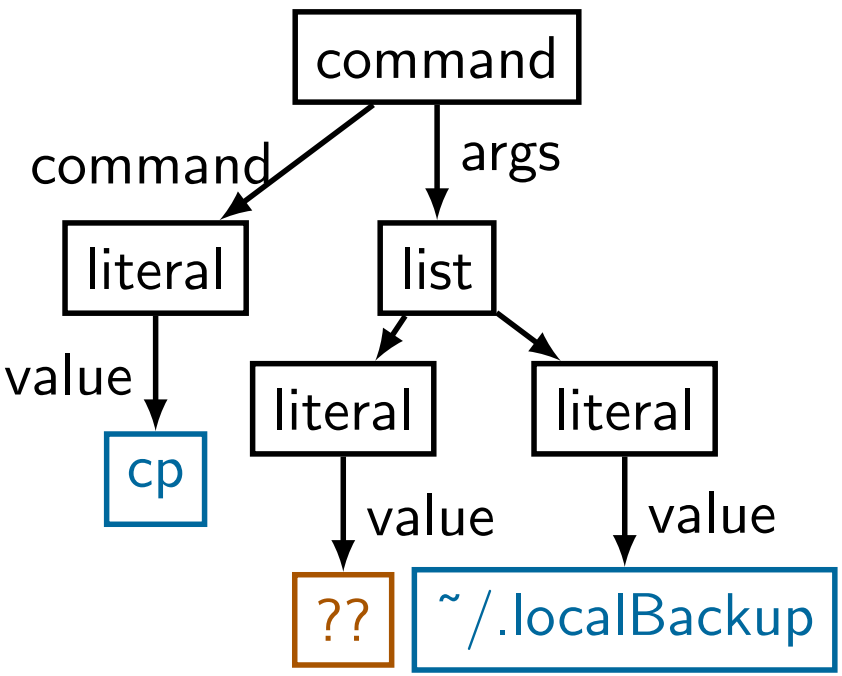
Policy:



Runtime Phase

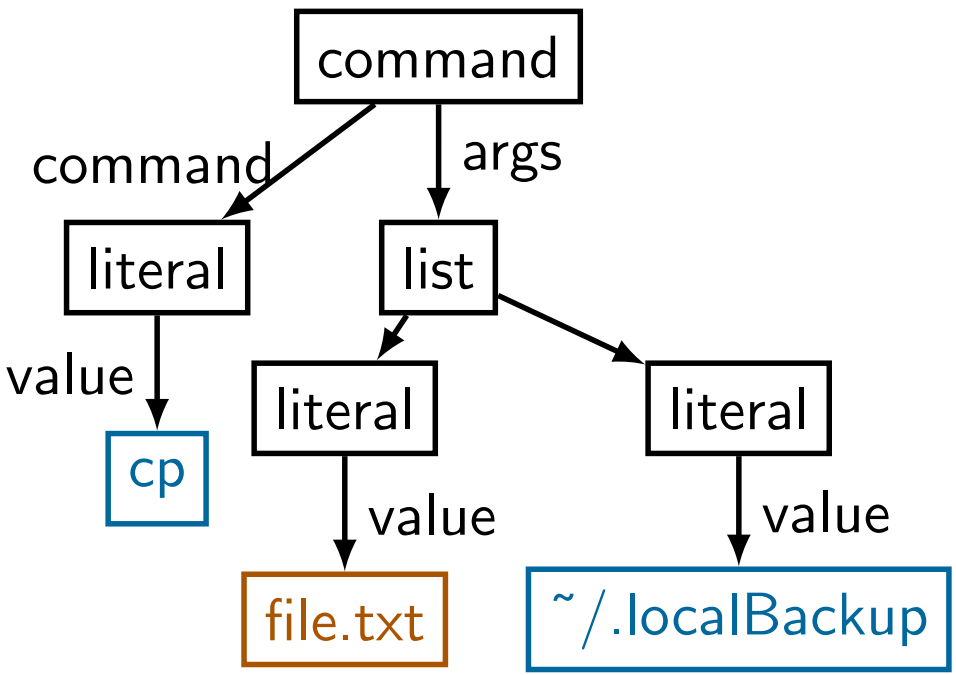
Enforce policy on strings passed to injection APIs

Policy:



Runtime string:

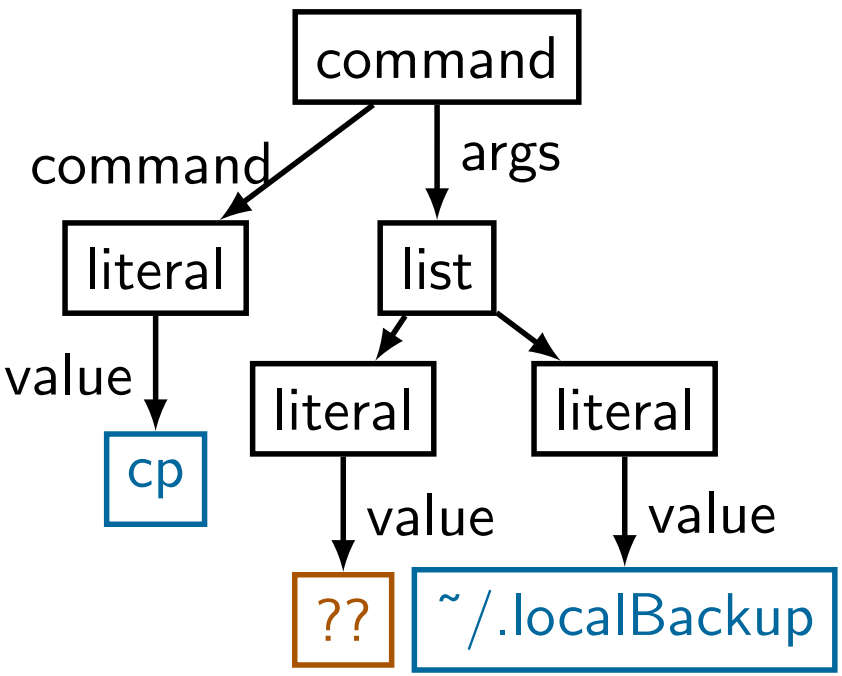
"cp file.txt ~/.localBackup"



Runtime Phase

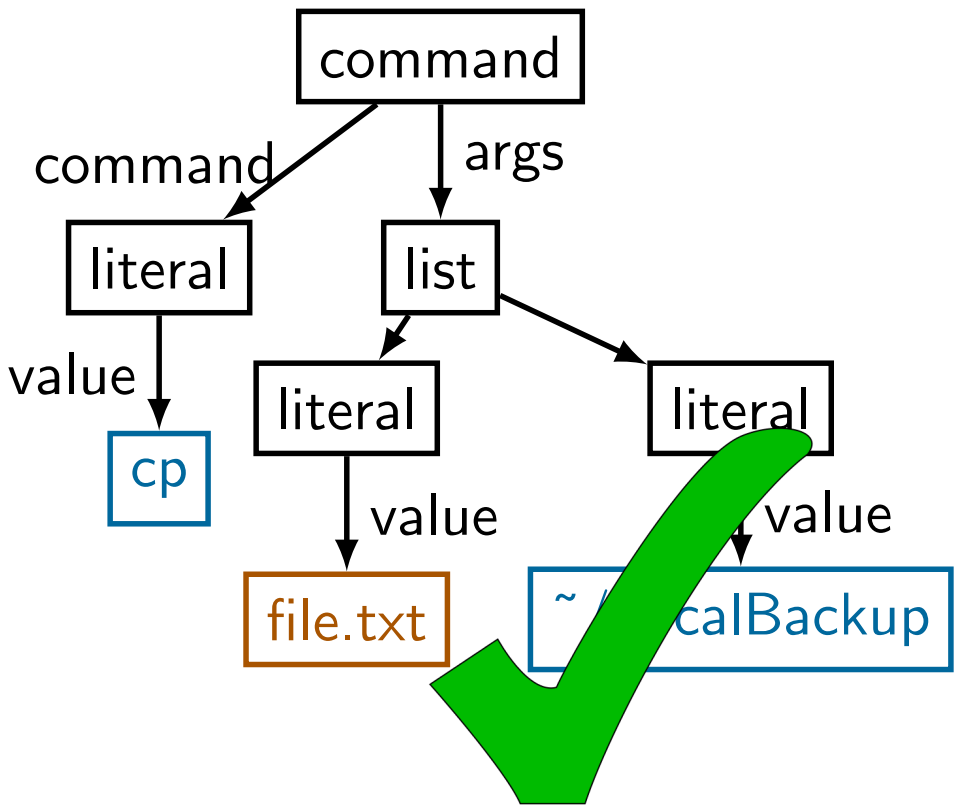
Enforce policy on strings passed to injection APIs

Policy:



Runtime string:

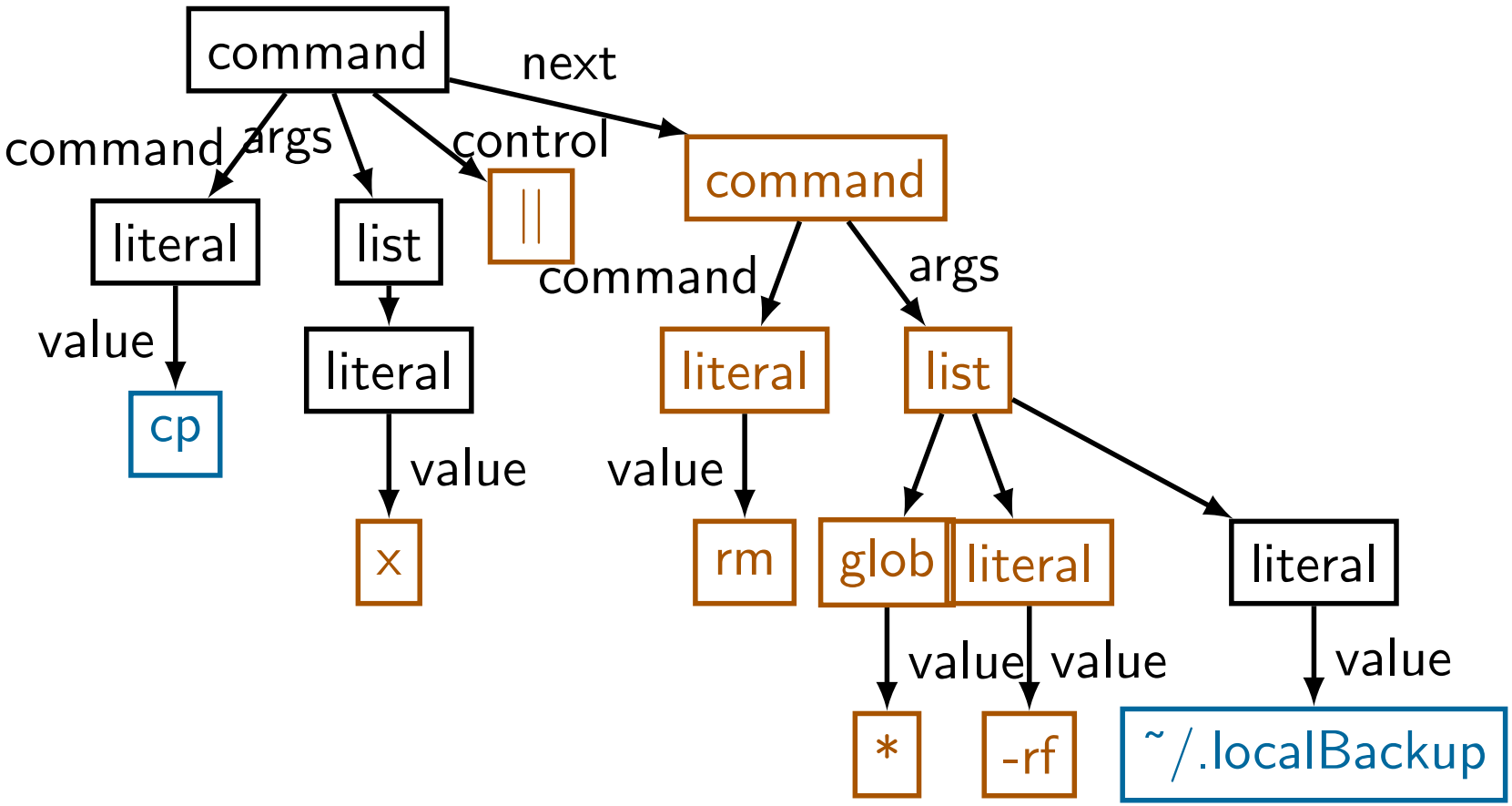
"cp file.txt ~/.localBackup"



Runtime Phase

Runtime string:

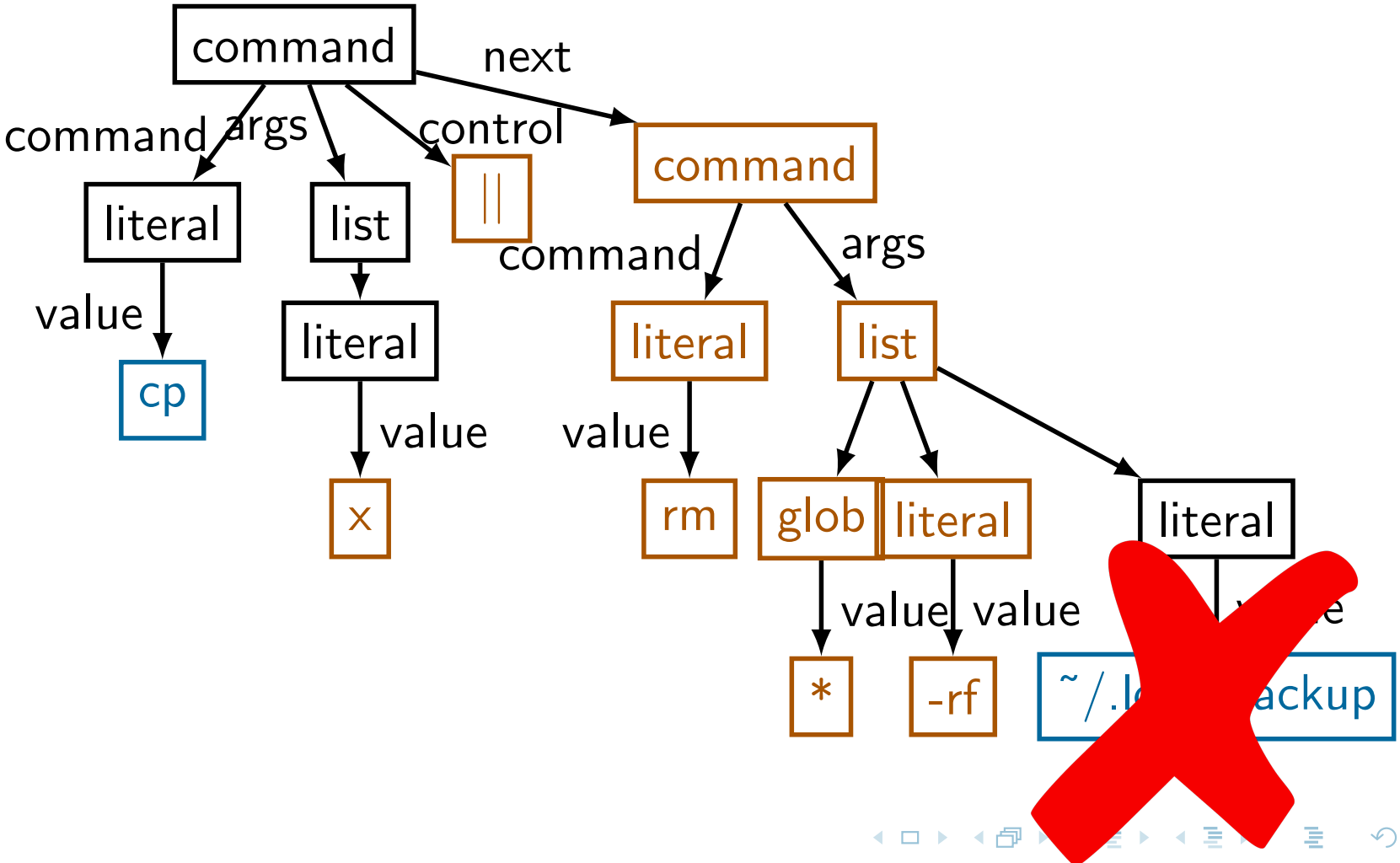
"cp x || rm * -rf ~/.localBackup"



Runtime Phase

Runtime string:

"cp x || rm * -rf ~/.localBackup"

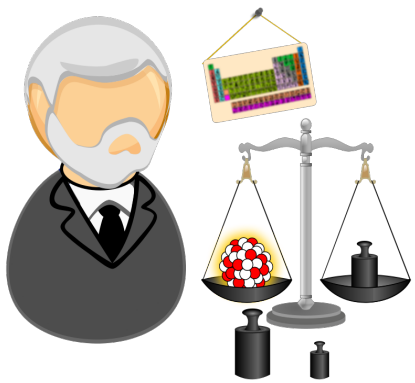


This Talk

Node.JS and Injections

Empirical Study

Synode



Evaluation

Evaluation: Static Phase

Setup

- 51K call sites of injection APIs

Precision

- **36.7%** of the call sites statically safe
- **63.3%** to be checked at runtime

Context

- most call sites have at least:
  - 10 constant characters per template
  - 1 unknown per template

Performance

- 4.4 seconds per module

Evaluation: Runtime Phase

Setup

- 24 modules
- 56 benign and 65 malicious inputs

Results

- **zero** malicious inputs that we do not stop
- five benign inputs that we incorrectly stop
- overhead: 0.74 milliseconds per call

Conclusions

Study of injection vulnerabilities

- First large-scale study of Node.js security
- `exec` and `eval` are prevalent in npm ecosystem
- Developers are slow to react

Conclusions

Study of injection vulnerabilities

- First large-scale study of Node.js security
- `exec` and `eval` are prevalent in npm ecosystem
- Developers are slow to react

Conclusions

Study of injection vulnerabilities

- First large-scale study of Node.js security
- `exec` and `eval` are prevalent in npm ecosystem
- Developers are slow to react

Prevention of injections

- Automatic and easy to deploy
  `https://github.com/sola-da/Synode`
- Small overhead and high accuracy

Conclusions

Study of injection vulnerabilities

- First large-scale study of Node.js security
- `exec` and `eval` are prevalent in npm ecosystem
- Developers are slow to react

Prevention of injections

- Automatic and easy to deploy
  `https://github.com/sola-da/Synode`
- Small overhead and high accuracy

Open challenges

- More precise static analysis
- Automatic generation of attacks

Example Limitation: Array.map()

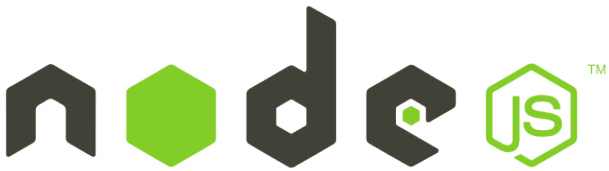
```
var keys = Object.keys(dmenuOpts);
var dArgs = keys.map(function(flag) {
  return '-' + flag + ' "' + dmenuOpts[flag] + '"';
}).join(' ');

var cmd = 'echo | dmenu -p "Password:" ' + dArgs;
exec(cmd);
```

Inferred template

```
'echo | dmenu -p "Password:" $dArgs'
```

Implementation



- Intraprocedural static analysis
- Based on **Google Closure Compiler**
- Policy for unknown parts:
  - **`exec`**: literal
  - **`eval`**: literal, identifier, property, array expression, object expression, member expression, expression statement

Beyond eval and exec

- `vm.runIn*Context()`

```
var vm = require('vm');
vm.runInThisContext(
        "console.log('" + input + "');");
```

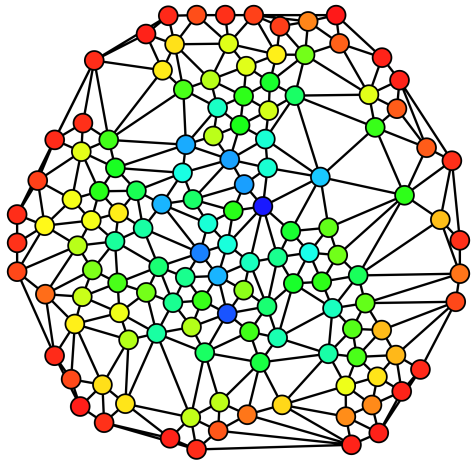
- `execa` module (1,000 dependents)

```
module.exports.shell = function(cmd) {
    args = ['-c', cmd]
    childProcess.spawnSync("/bin/sh", args);
}
```

Why is the Application Domain Unique?

20 out of 66 advisories are injections (Node Security Project)

Bad habits



Unnecessary code reuse (see left-pad)

No sandbox