

GRAPH THEORY

A massive, bright orange and yellow explosion dominates the center of the image, set against a backdrop of a clear blue sky and a green, hilly landscape. The explosion is highly detailed, with intense fire and smoke billowing upwards and outwards. The text 'MIND. EXPLOSION.' is overlaid in the center of the explosion.

MIND. EXPLOSION.

Who are you exactly?

And why should I pay attention?

Founder & CEO at



nodejitsu

I'm on the Internet!

@indexzero

AND RECENTLY...

LITERALLY A

MASTER
OF HCI



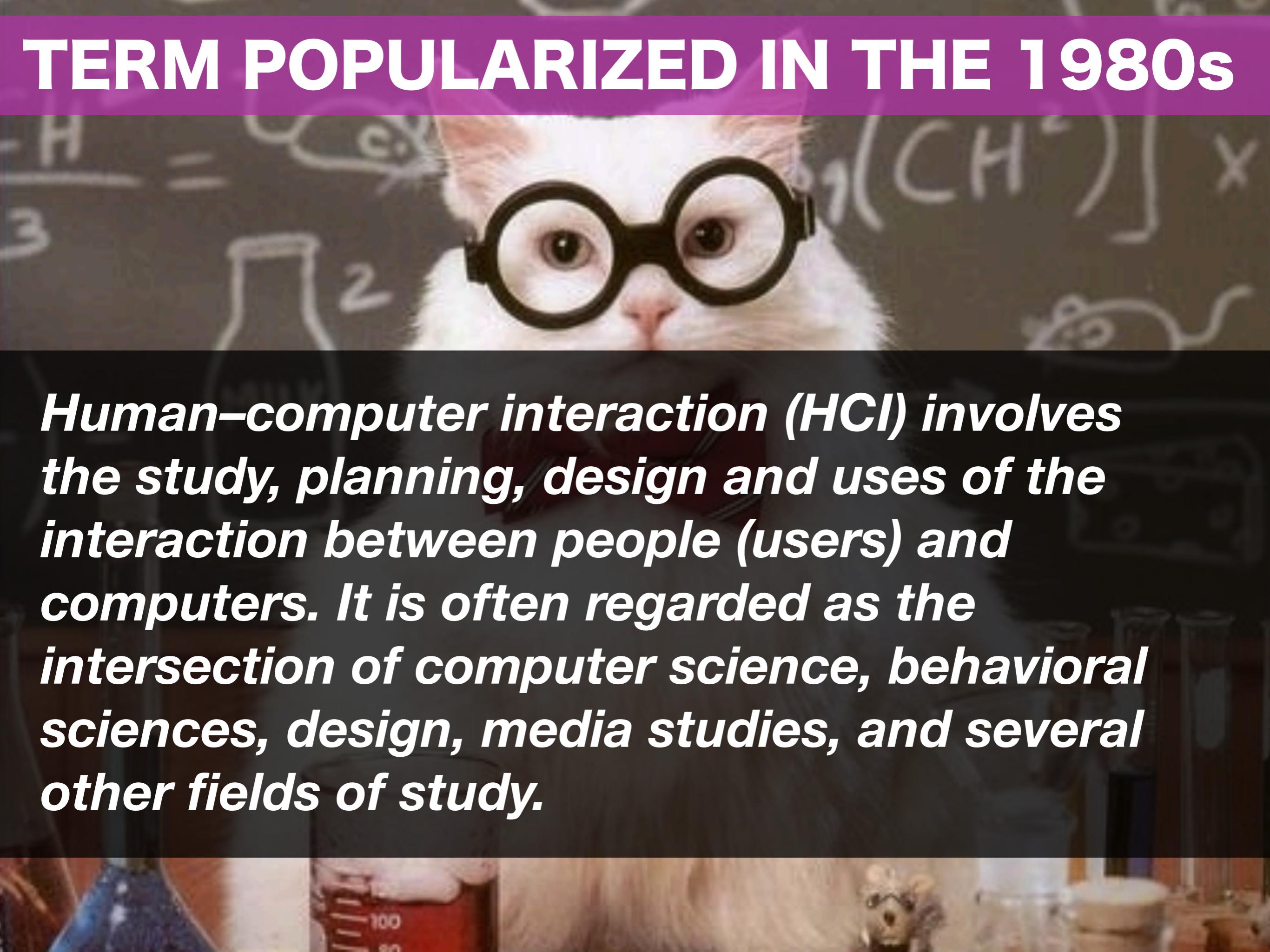
HUMAN-COMPUTER INTERACTION

THE ART OF

MANAGING
STRATEGY

MOVE YOUR COMPETITION
WITH YOUR TACTICS

TERM POPULARIZED IN THE 1980s

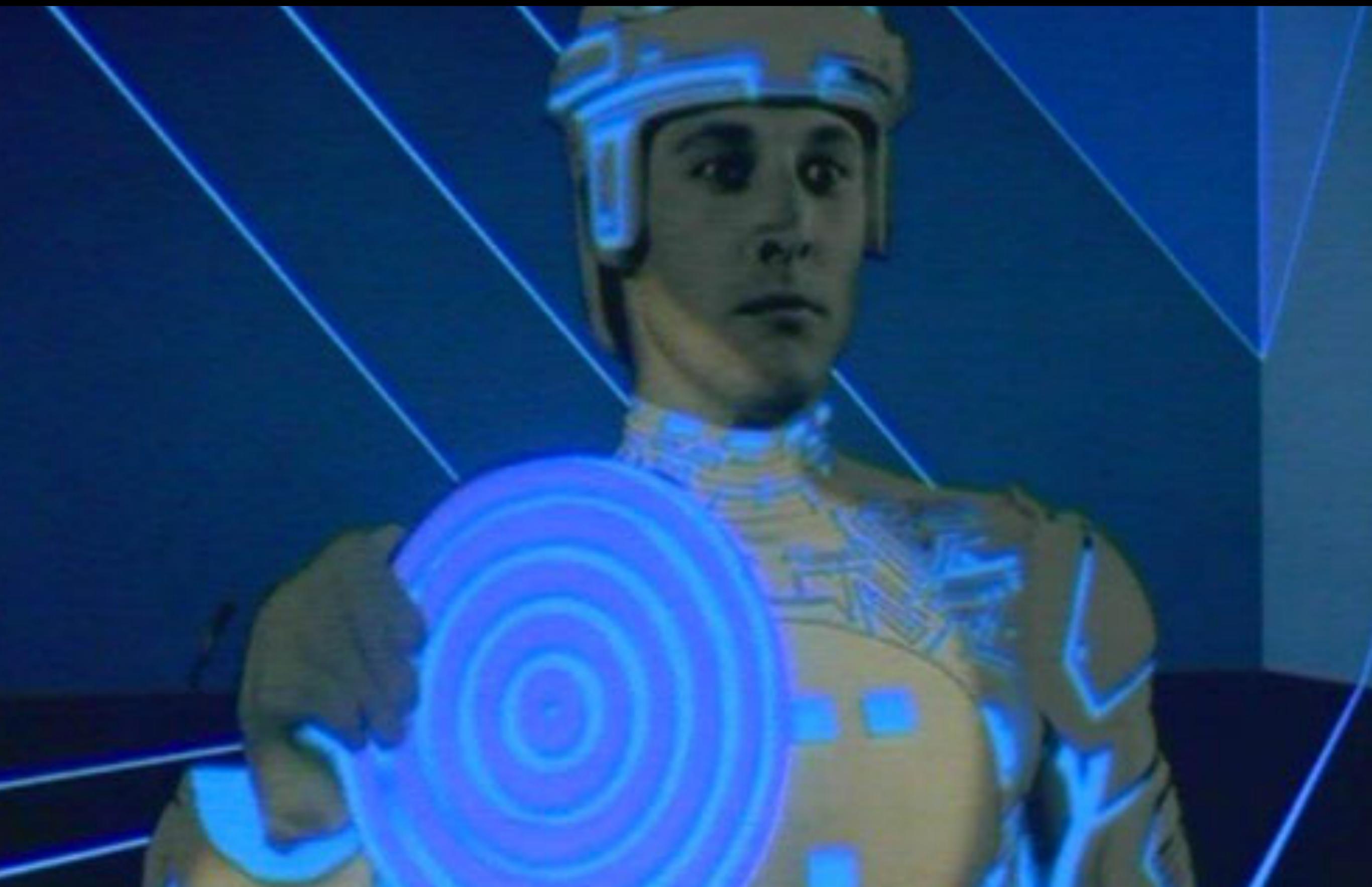


Human-computer interaction (HCI) involves the study, planning, design and uses of the interaction between people (users) and computers. It is often regarded as the intersection of computer science, behavioral sciences, design, media studies, and several other fields of study.

A glowing blue Tron-style character in a dynamic pose, set against a dark background with glowing lines.

HCI FIGHTS FOR THE USER

WELL ... SORT OF ...



HCI & Static Analysis?

Why the what precisely?

Toward decreasing mean decision-time in Open Source with com(STAT)²

Charlie Robbins, Jane Kim, William McAuliffe

Columbia University

New York, NY, 10027, USA

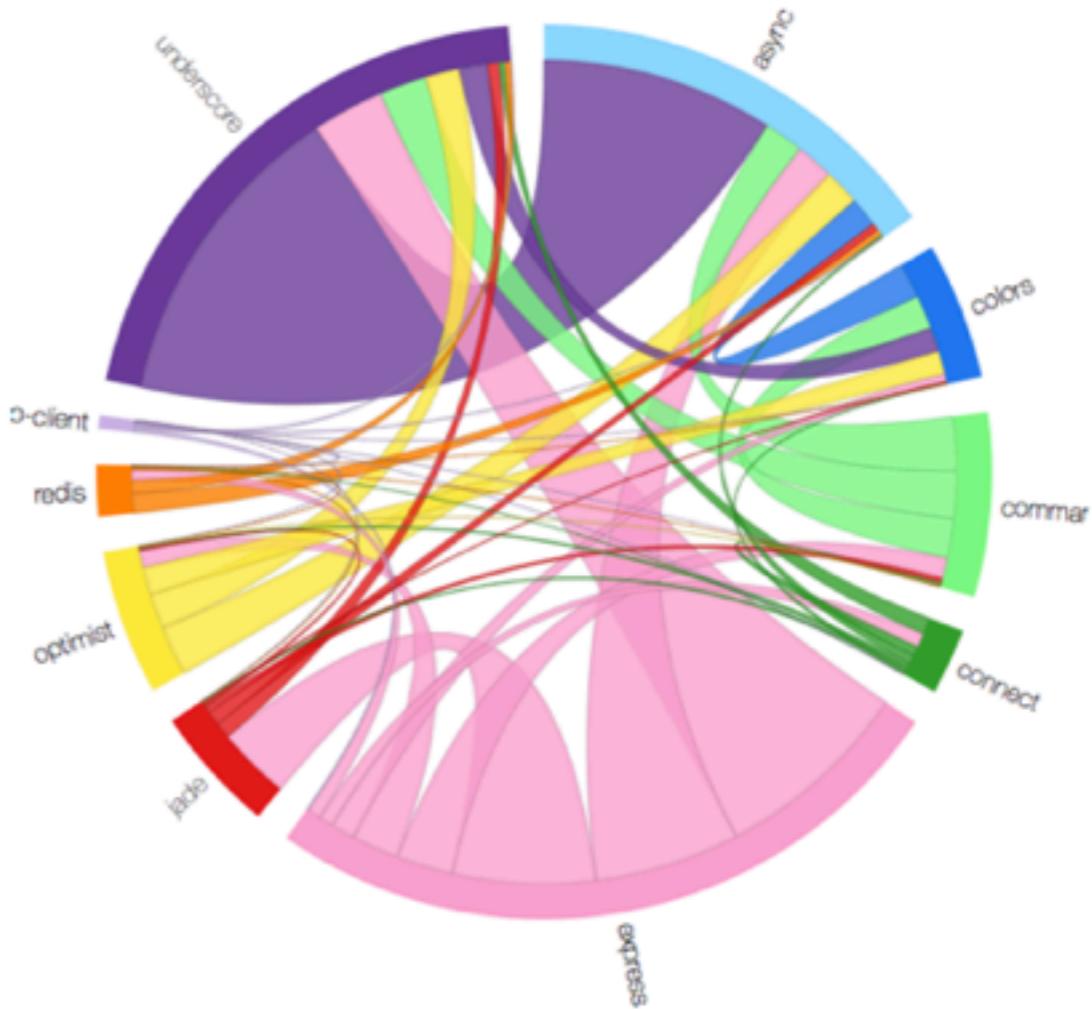
{ cjr2139, jk3316, wom2102 } @ columbia.edu

May 14, 2014

Abstract

Modules, small reusable pieces of code, have long been the goal of sustainable Software Engineering practices. In recent years in some circles, particularly Javascript, this goal has become a reality. Having these large Javascript applications depend on hundreds of small modules presents its own decision-making challenges for module authors. Most are flying completely blind with respect to how their module(s) are being used and what constitutes a “breaking change” for their user base.

This decision-making for mature projects often takes more time than the engineering work itself. This paper presents a COMprehensive STATistical STATIC analysis interface, com(STAT)², that aims to decrease the mean decision time for module authors and evaluates it on well-known authors in the popular Javascript environment of Node.js and npm (Node Packaged Modules).



1 Establishing the Expert Problem Domain

HCII

WITH

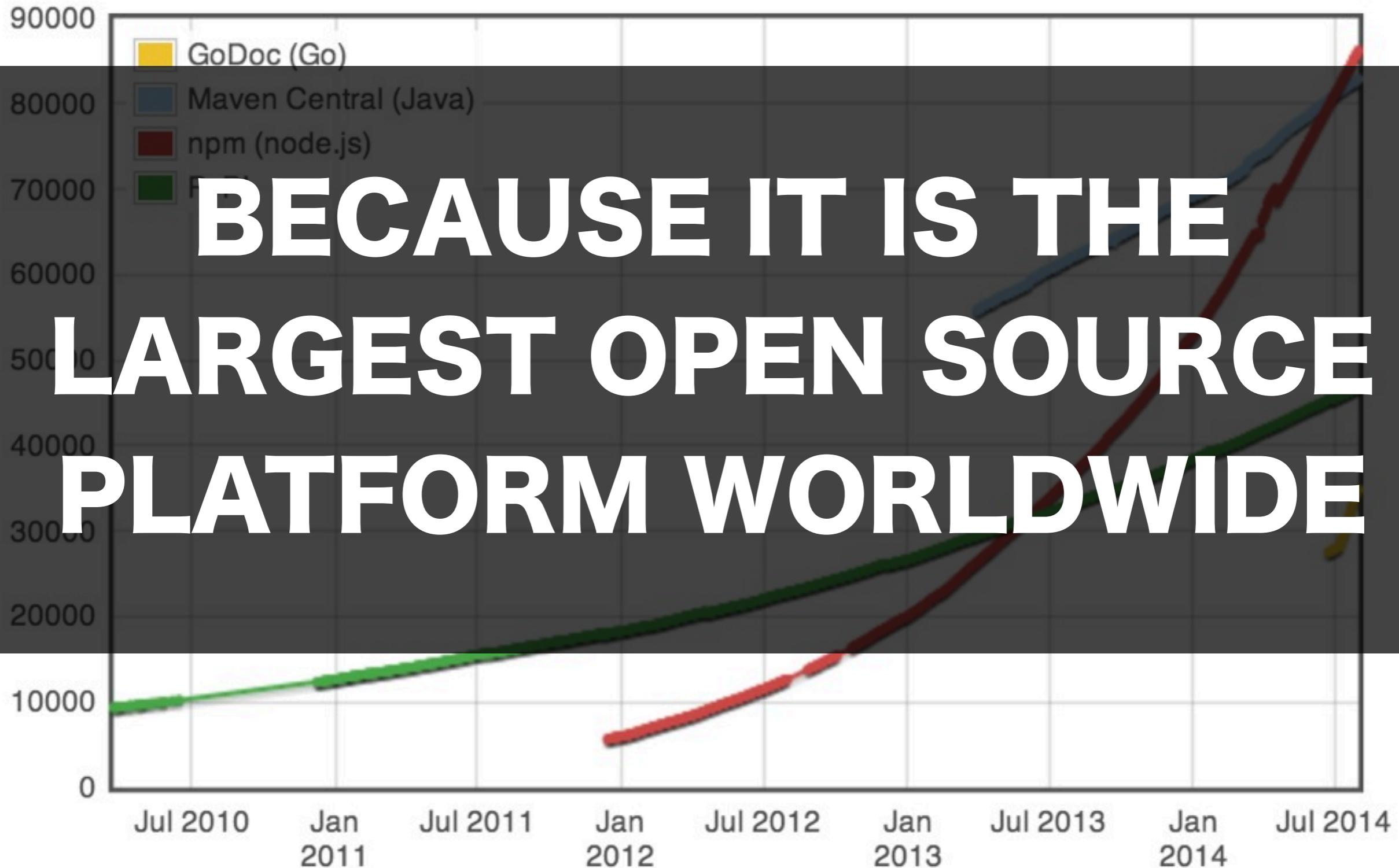




I CAN HAZ USER?

Why is **npm** a worthy HCI topic?

Because reasons I guess maybe?



Why is **npm** a worthy HCI topic?

How soon exactly?

AS OF 06/26/14

npm

80,353

-323

173 / day

Maven

80,676

+323

75 / day

ROUGHLY BY TUESDAY.

YUP.



**PLEASE. TELL ME ABOUT
YOUR RESEARCH STUDY**

Why is a worthy HCI topic?

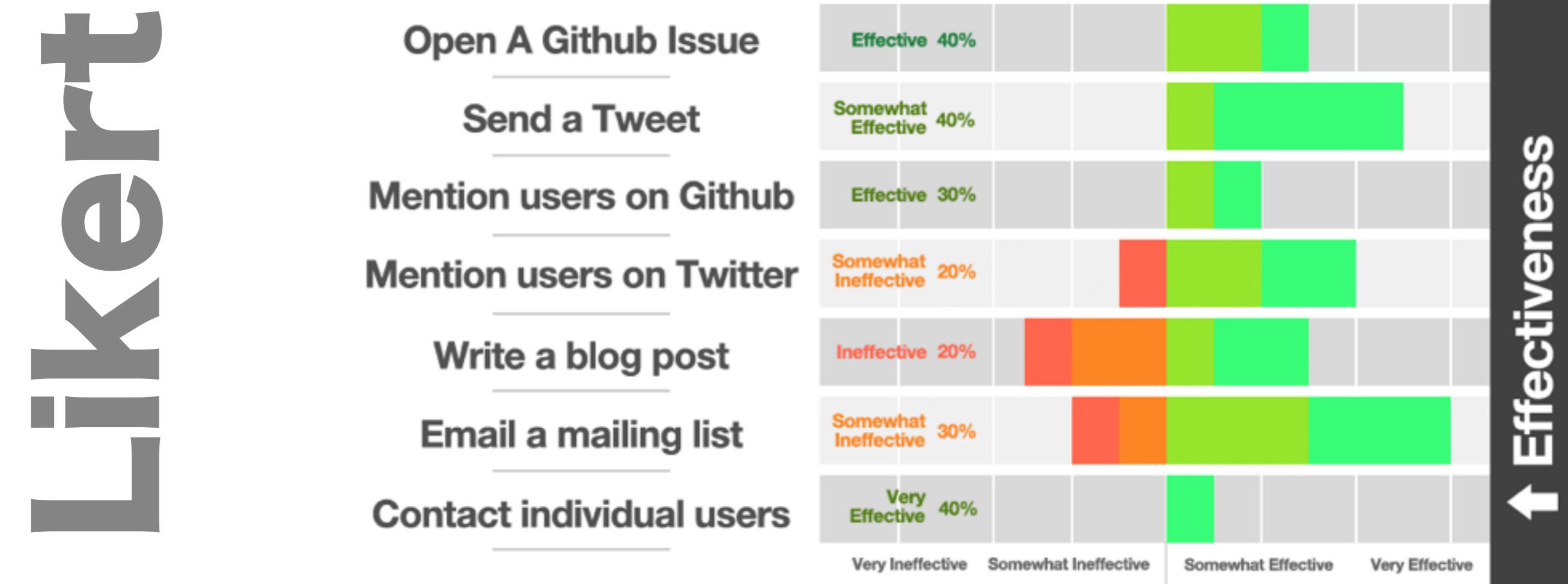
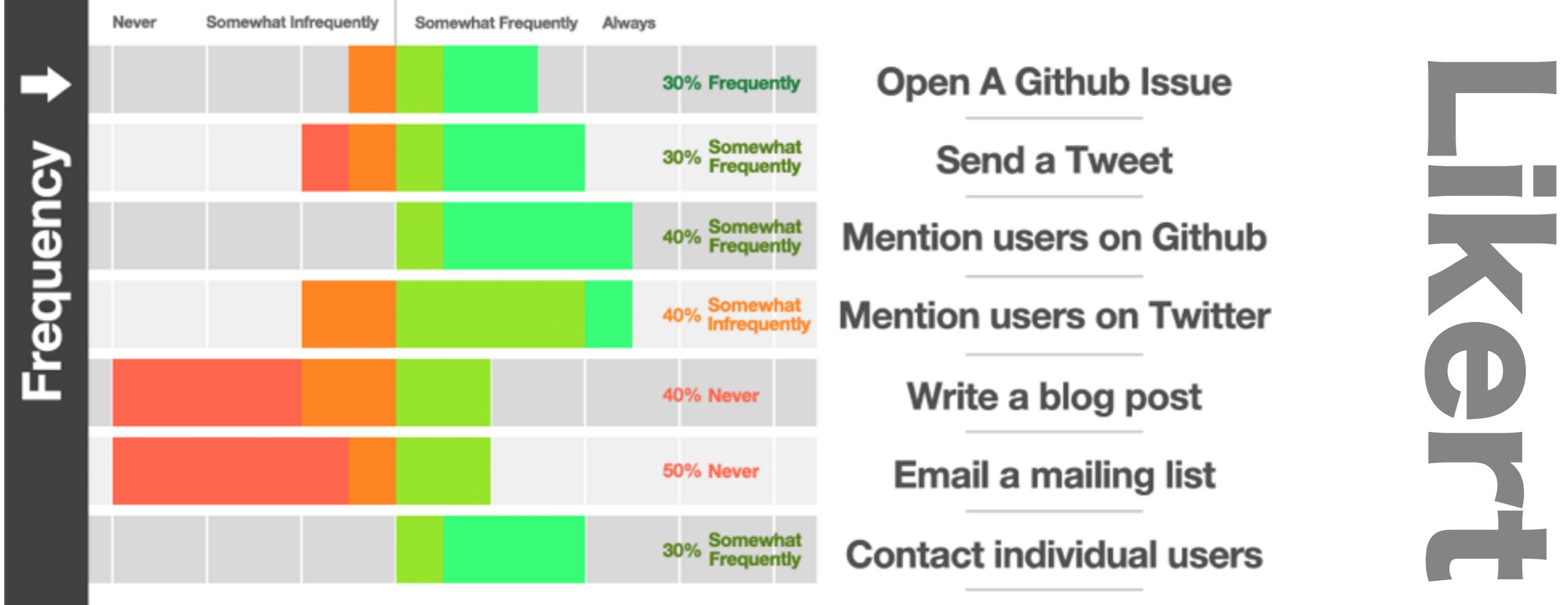
Why the what precisely?

**SMALL GROUP
OF 8 EXPERTS**



Likert Scales.

Seriously, why does this need a name?



Why is **QPM** a worthy HCI topic?

Why the what precisely?

MOST EFFECTIVE
METHODS USED
MOST INFREQUENTLY

MODULE AUTHORS
AND DEVELOPERS
HAVE QUESTIONS
ABOUT MODULES
BEING DEVELOPED

A glowing blue Tron character, likely Clu, is shown from the waist up, standing in a dark, futuristic environment. The character is wearing a glowing blue suit with a visor and glowing blue lines on their arms. The background is dark with some glowing blue lines and shapes.

**HCI TRON MUST KNOW
HOW DO WE DO BETTER?**

GRAPHS

RULE

EVERYTHING

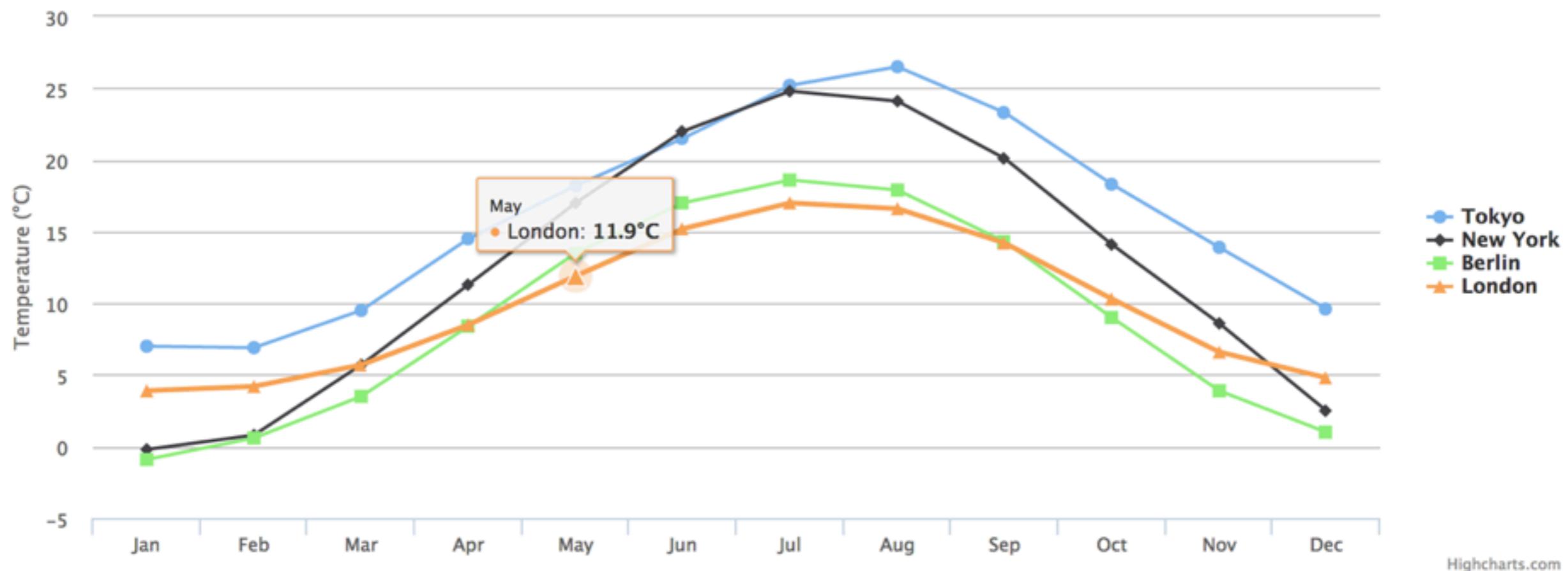
AROUND

ME

NOT THIS KIND

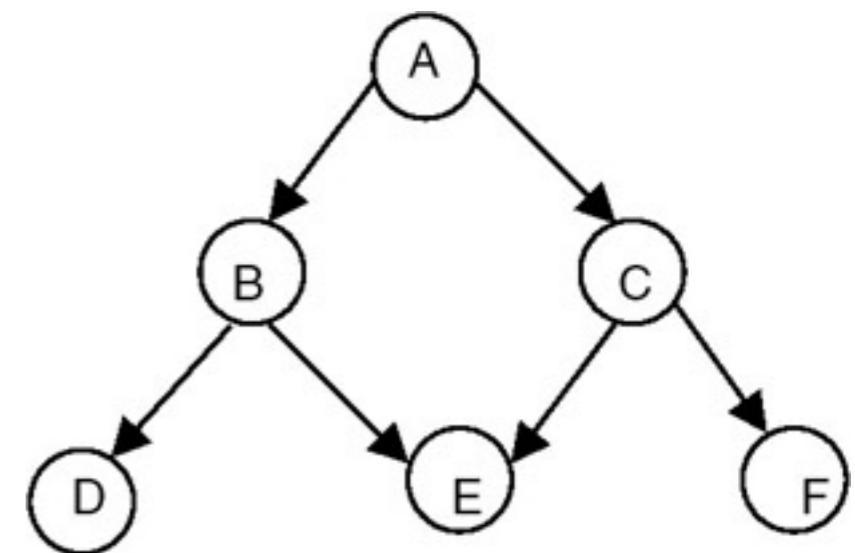
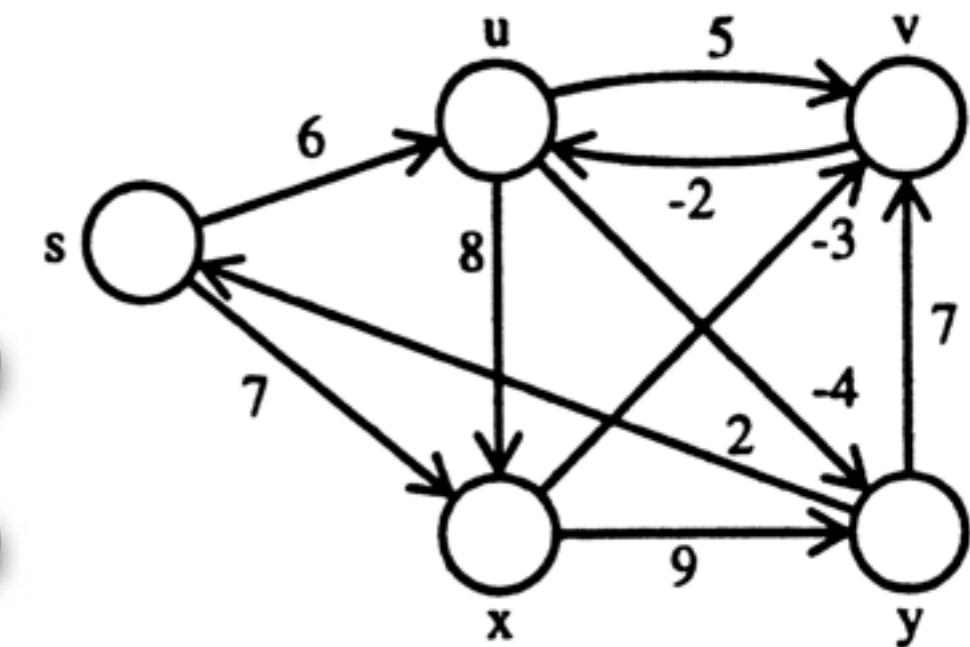
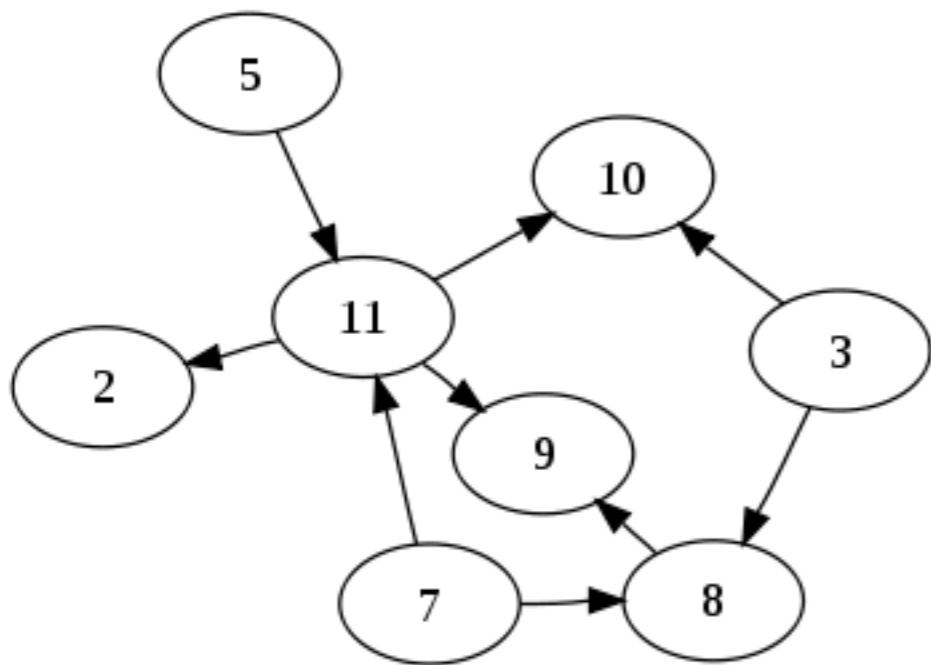
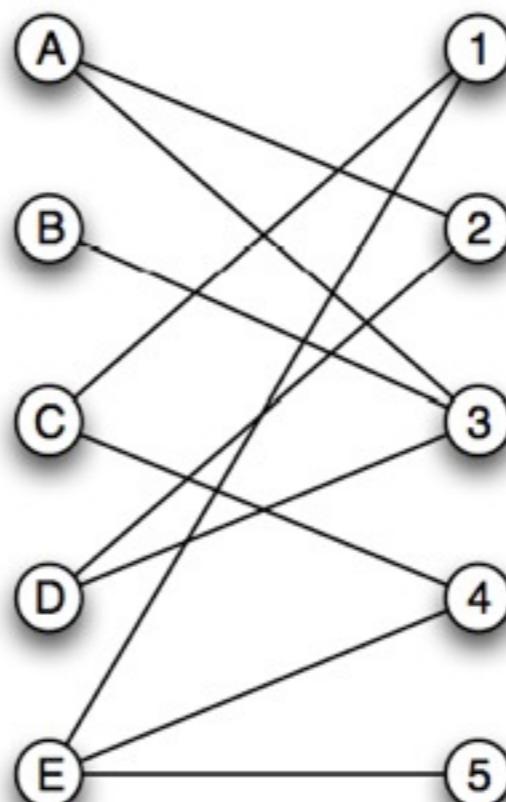
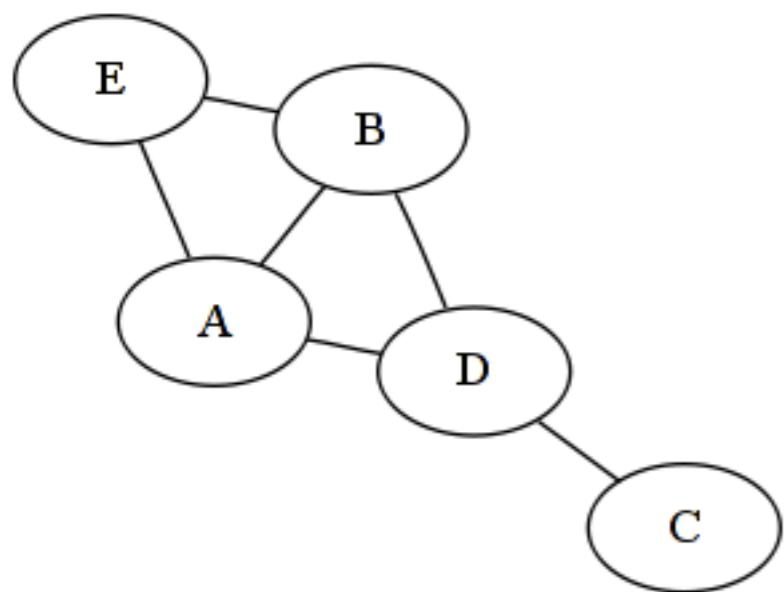
Monthly Average Temperature

Source: WorldClimate.com



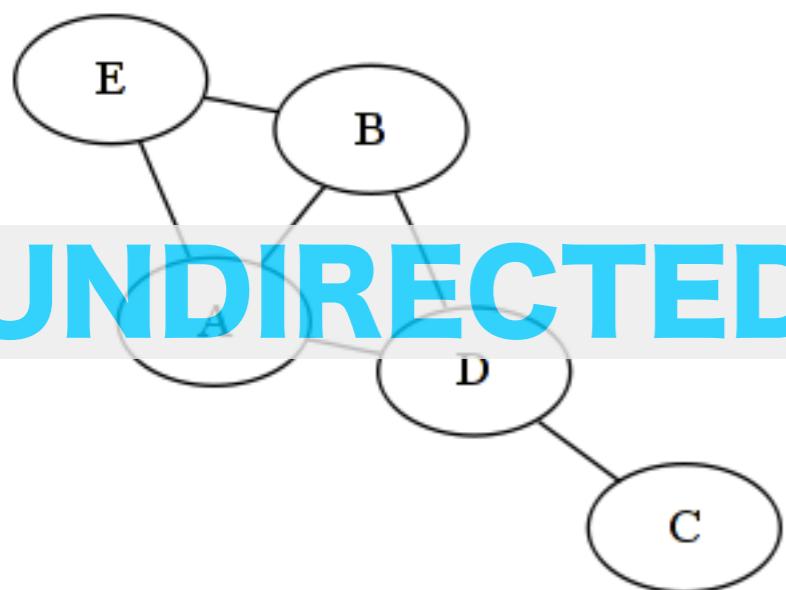
OF GRAPH

THESE KINDS

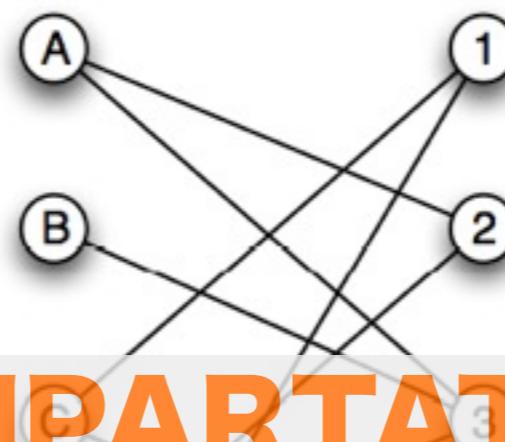


OF GRAPHS

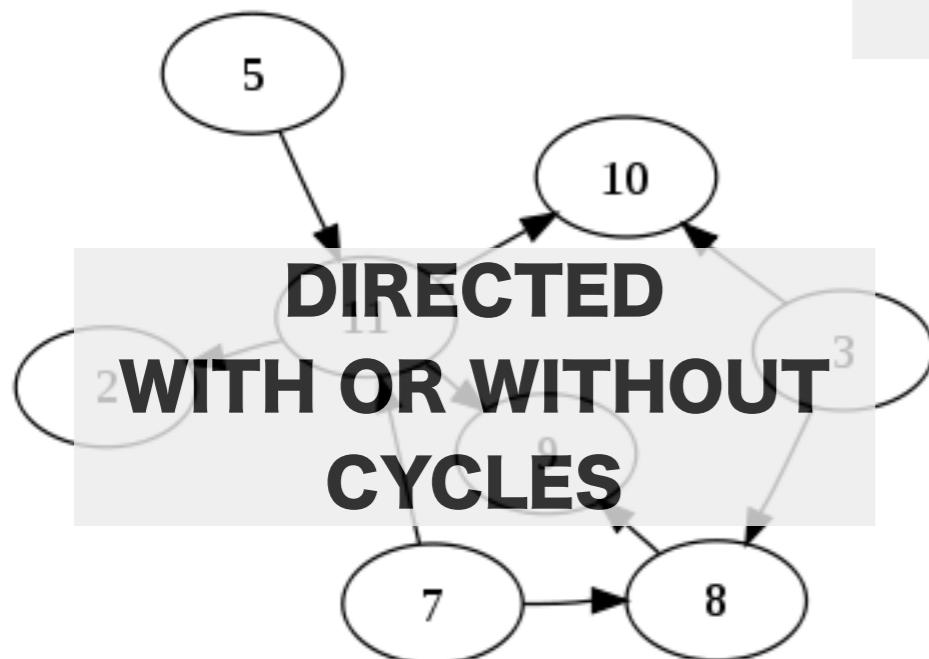
ALL OF THESE GRAPHS



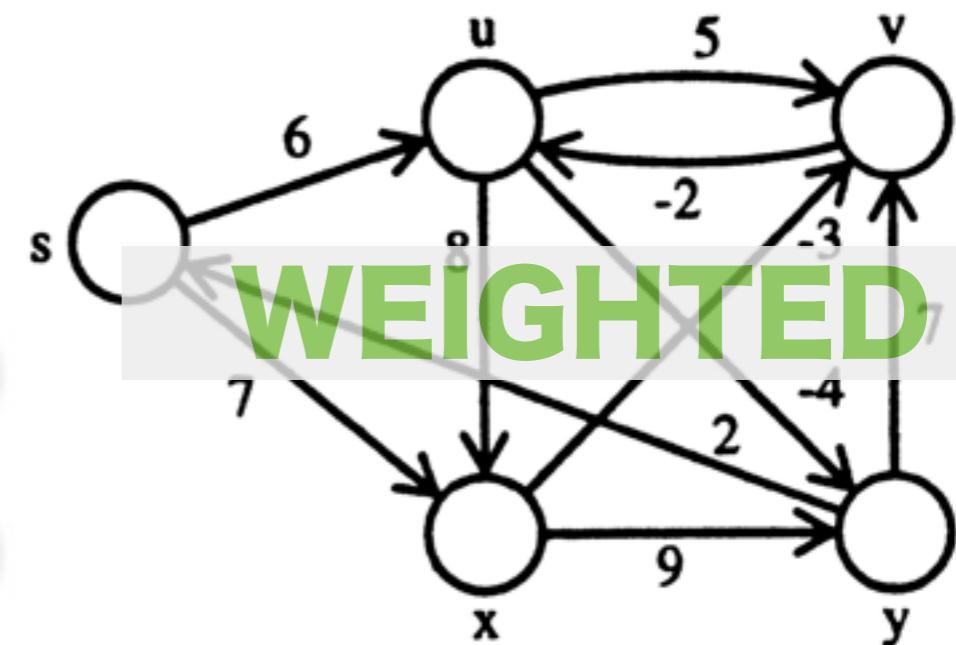
UNDIRECTED



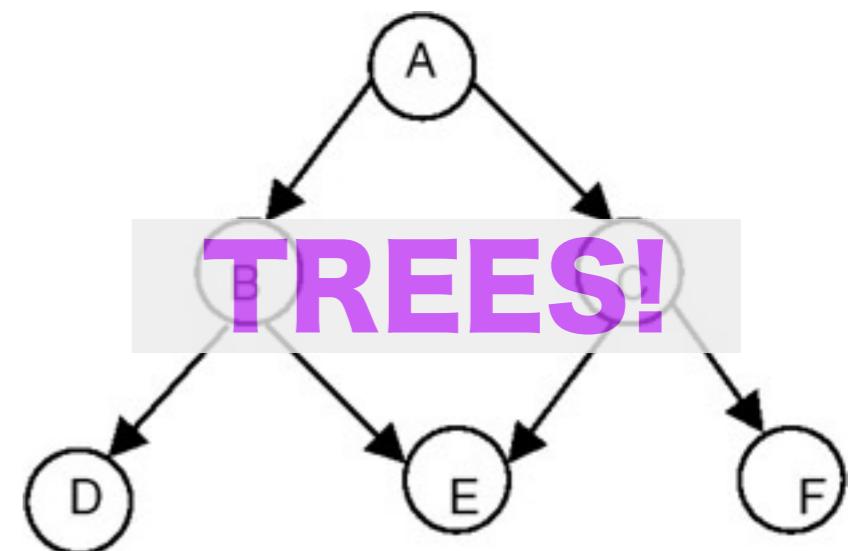
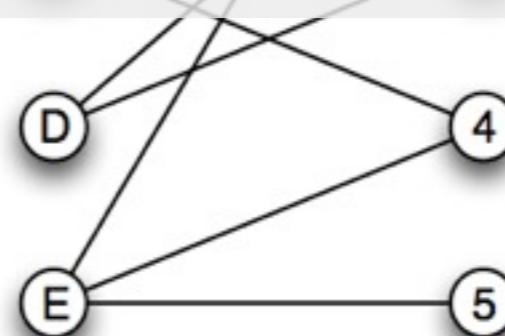
BIPARTITE



DIRECTED
WITH OR WITHOUT
CYCLES



WEIGHTED



TREES!

ARE DIFFERENT

**TRANSPORTATION
NETWORKS**

INFORMATION NETWORKS

MOLECULAR CHEMISTRY

WIRELESS NETWORKS

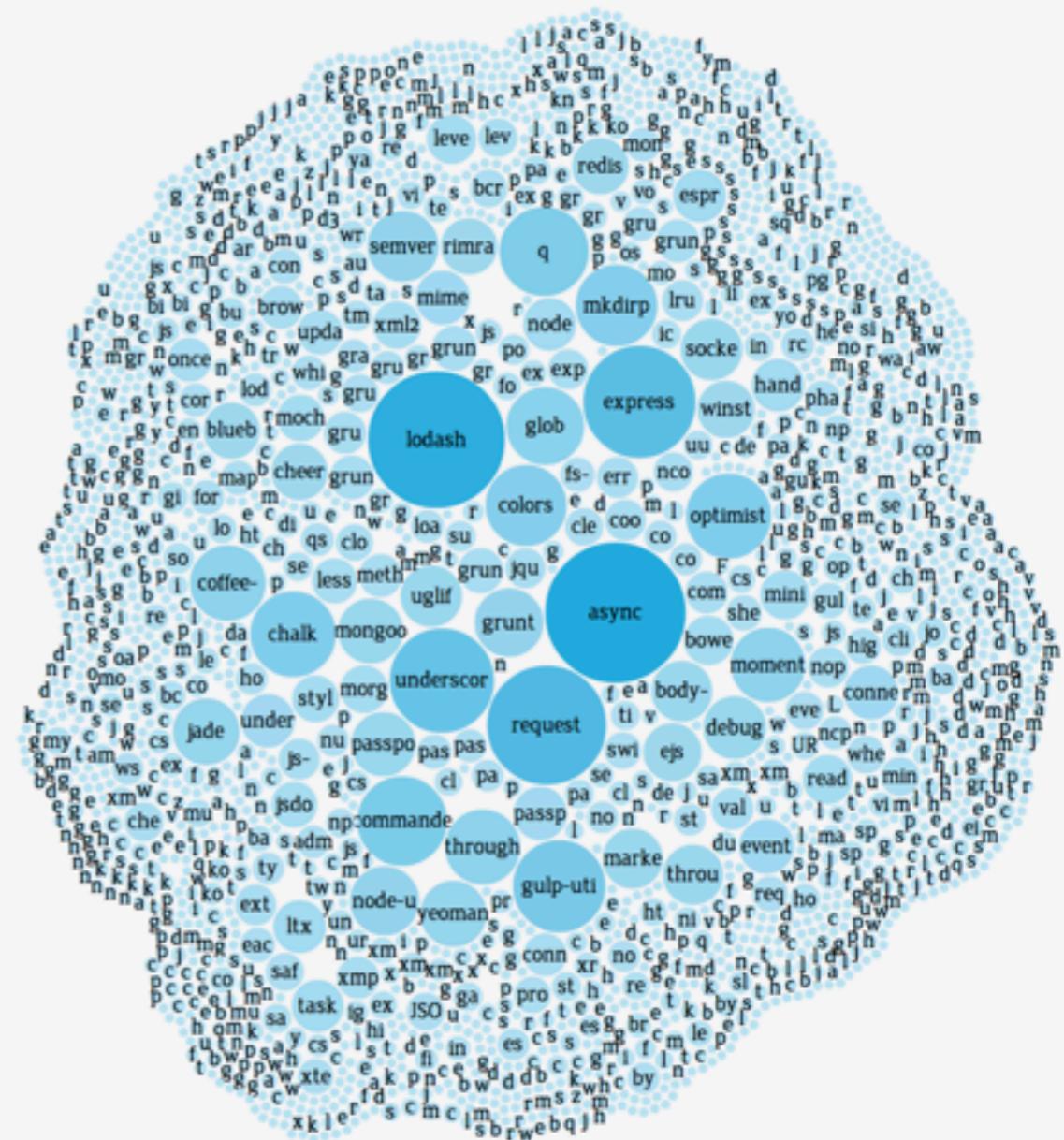
MAJOR LEAGUE BASEBALL

DEPENDENCY MANAGEMENT

Dependency graphs

The $G = (V, E)$ kind of graph

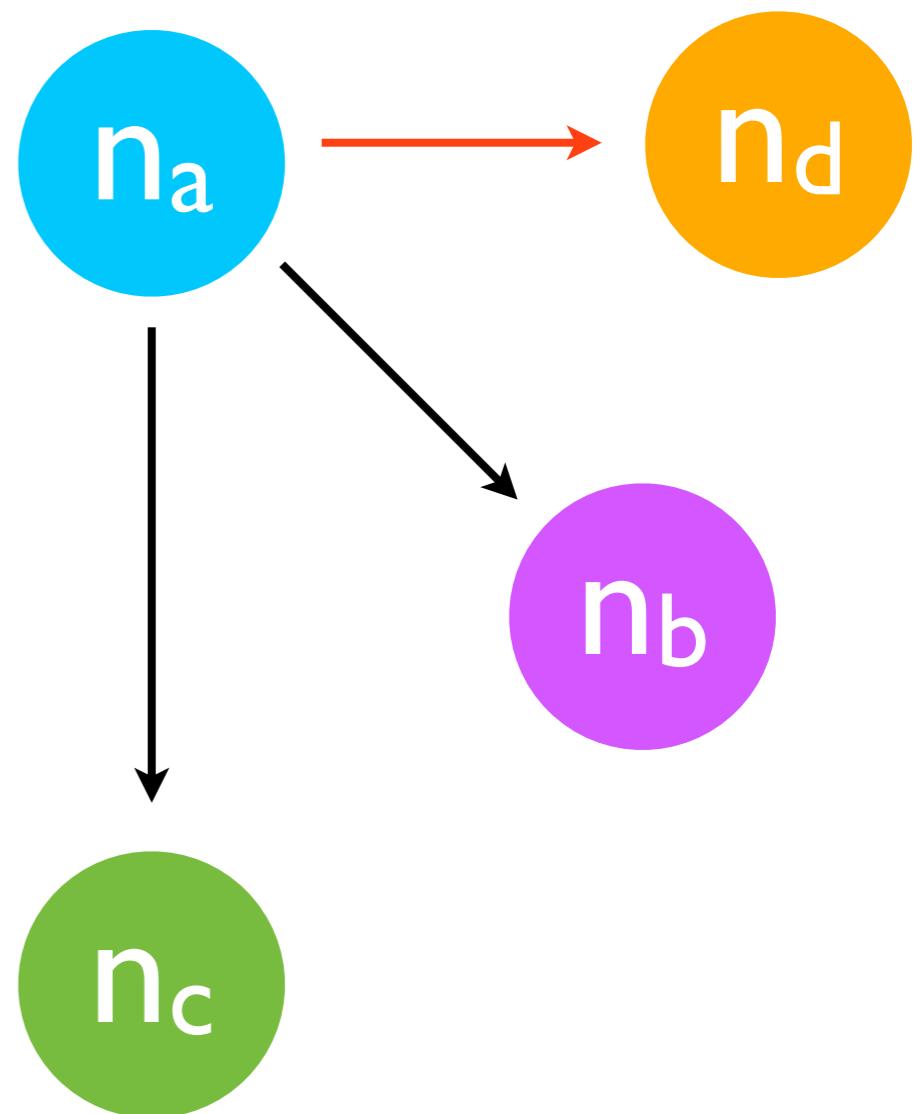
- Basic graph representation is not extremely helpful for visualizing package relationships.
 - But it does provide a basic structure for a graph search problem.



Dependency graphs

The $G = (V, E)$ kind of graph

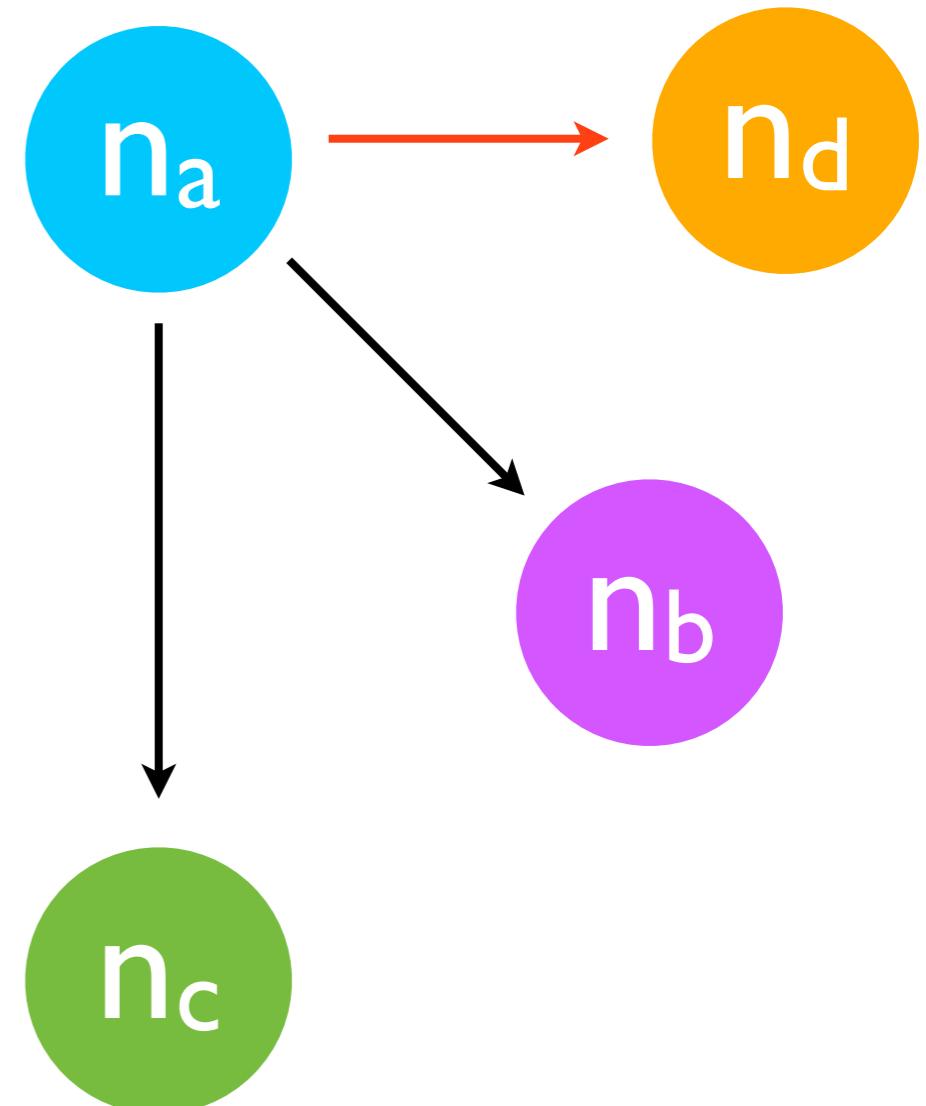
- To build our graph, G , we add a node (or vertex) for every package.
- We then add a colored edge from any node n_A to n_B if package A depends on package B.
- Edges are colored by dependency type:
dependencies or
devDependencies



Dependency graphs

The $G = (V, E)$ kind of graph

```
{  
  "name": "package-a",  
  "dependencies": {  
    "package-b": "~1.0.4",  
    "package-c": "~2.1.3"  
  },  
  "devDependencies": {  
    "package-d": "~3.1.2"  
  },  
  "main": "./index.js"  
}
```



Now imagine this for 80,000+ packages!

A massive, bright orange and yellow explosion dominates the center of the image, set against a backdrop of a clear blue sky and a hilly, grassy landscape. The explosion is highly detailed, showing intense fire and smoke. The text 'ANOTHER. MIND. EXPLOSION.' is overlaid in large, white, sans-serif capital letters.

ANOTHER. MIND. EXPLOSION.

Actually.
It's not so bad.

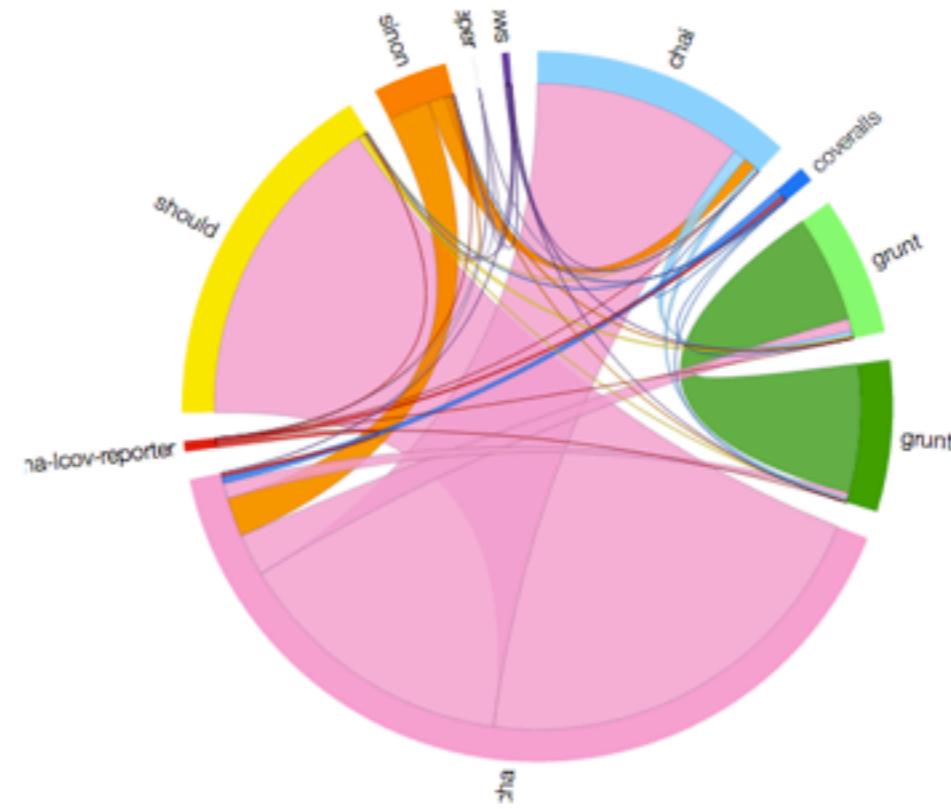
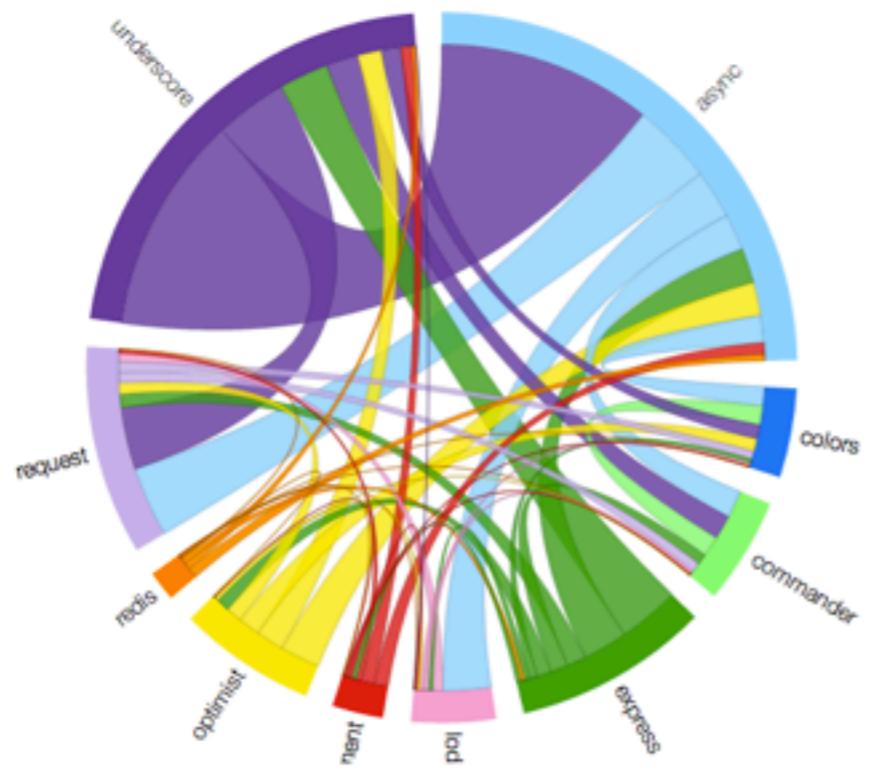


Dependency graphs

Ok, so what is this useful for?

- There are tons of useful applications of dependency graphs.
- Lets consider two.

First: Codependencies



Codependencies?

Well, technically co(*)dependencies.

- Codependencies answer the question “*people who depend on package A also depend on ...*”

["package", "codep", "thru"]

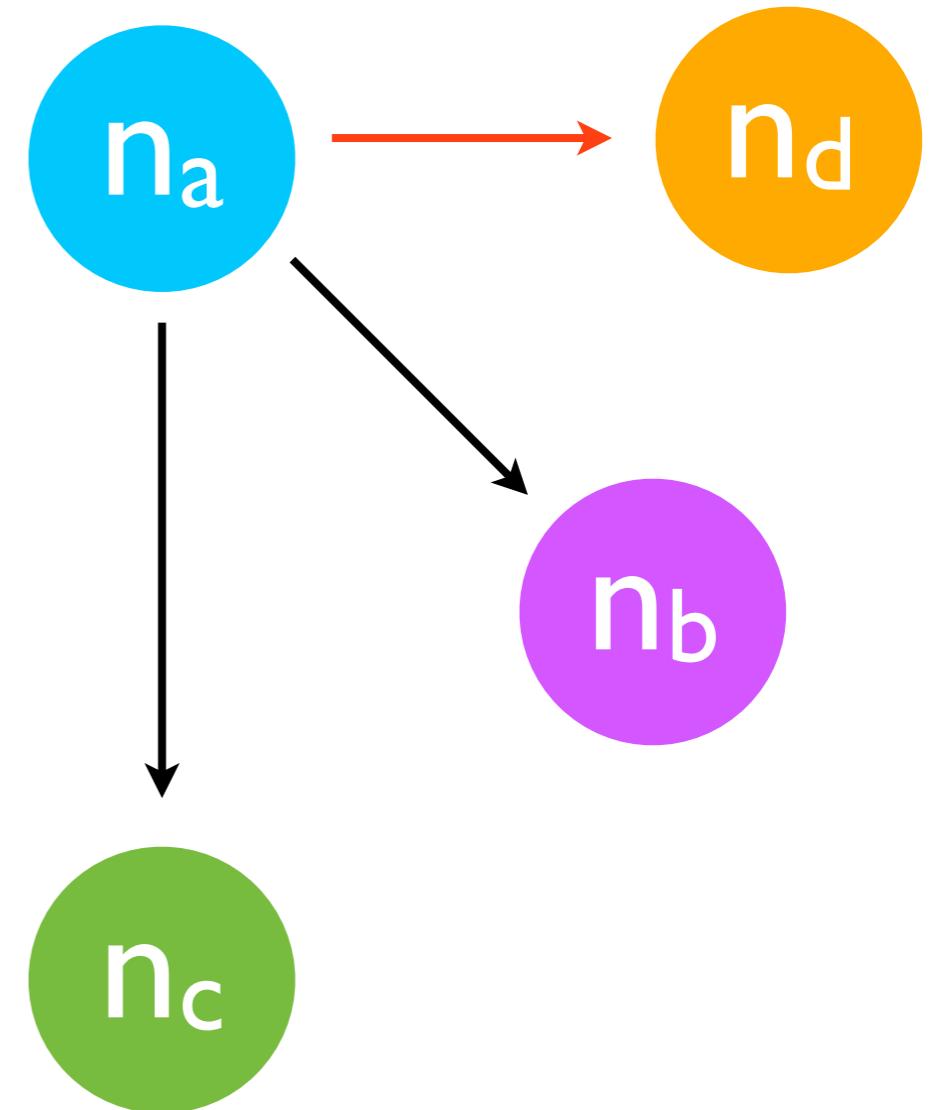


<3 CouchDB!

Dependency graphs

The $G = (V, E)$ kind of graph

```
{  
  "name": "package-a",  
  "dependencies": {  
    "package-b": "~1.0.4",  
    "package-c": "~2.1.3"  
  },  
  "devDependencies": {  
    "package-d": "~3.1.2"  
  },  
  "main": "./index.js"  
}
```



Here we would say that package-b and package-c have a codependency relationship thru package-a

Codependencies?

Thanks CouchDB!

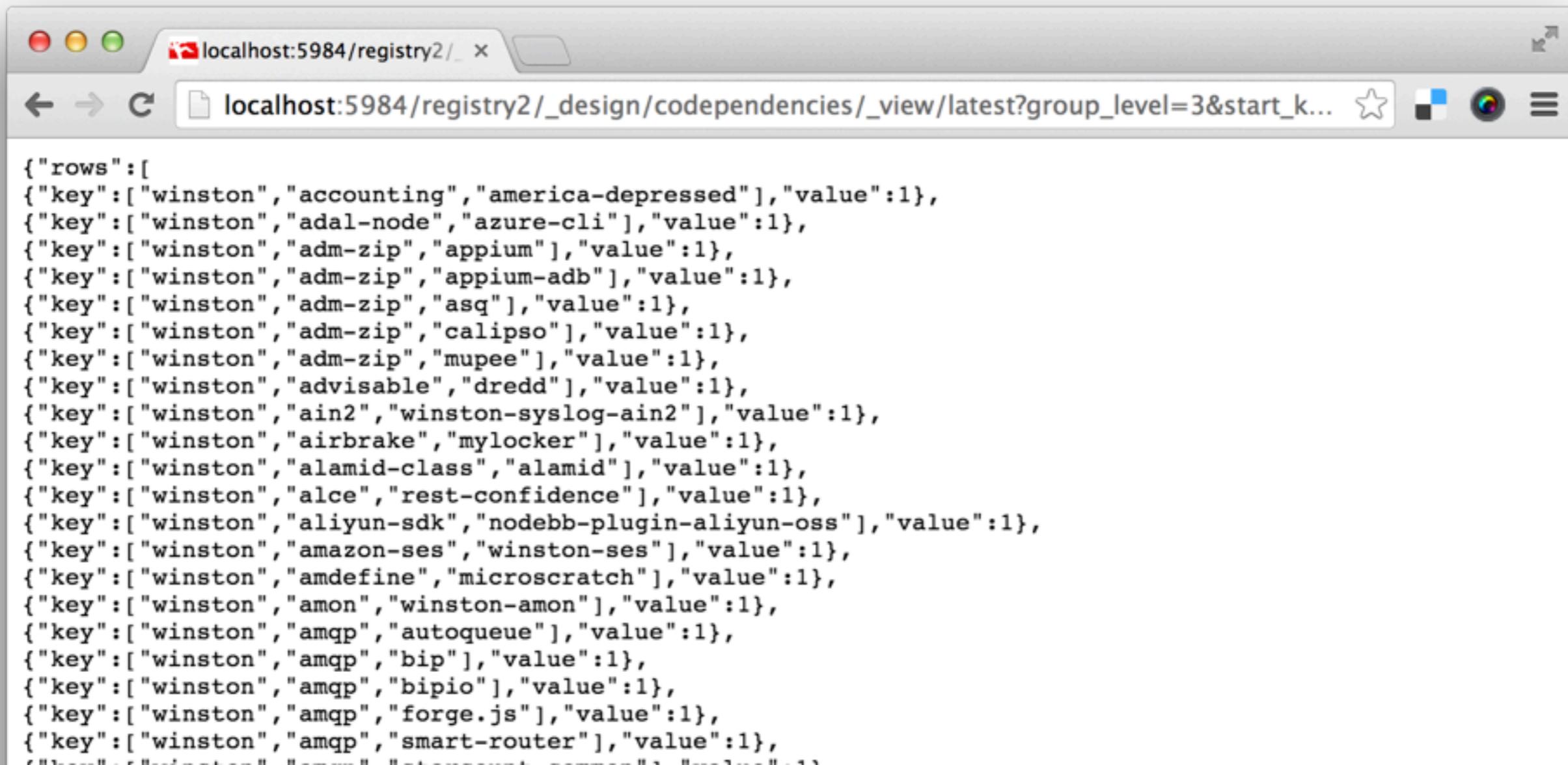
- We can get all of this from a **simple CouchDB view.**

```
for (var dep in d) {  
  dep = dep.trim();  
  for (var codep in d) {  
    codep = codep.trim();  
    if (dep !== codep) {  
      emit([dep, codep, doc._id], 1)  
    }  
  }  
}
```

Codependencies

Thanks CouchDB!

```
group_level=3
&start_key=["winston"]
&end_key=["winston",{}]
```



```
{"rows": [
  {"key": ["winston", "accounting", "america-depressed"], "value": 1},
  {"key": ["winston", "adal-node", "azure-cli"], "value": 1},
  {"key": ["winston", "adm-zip", "appium"], "value": 1},
  {"key": ["winston", "adm-zip", "appium-adb"], "value": 1},
  {"key": ["winston", "adm-zip", "asq"], "value": 1},
  {"key": ["winston", "adm-zip", "calipso"], "value": 1},
  {"key": ["winston", "adm-zip", "mupee"], "value": 1},
  {"key": ["winston", "advisable", "dredd"], "value": 1},
  {"key": ["winston", "ain2", "winston-syslog-ain2"], "value": 1},
  {"key": ["winston", "airbrake", "mylocker"], "value": 1},
  {"key": ["winston", "alamid-class", "alamid"], "value": 1},
  {"key": ["winston", "alce", "rest-confidence"], "value": 1},
  {"key": ["winston", "aliyun-sdk", "nodebb-plugin-aliyun-oss"], "value": 1},
  {"key": ["winston", "amazon-ses", "winston-ses"], "value": 1},
  {"key": ["winston", "amdefine", "microscratch"], "value": 1},
  {"key": ["winston", "amon", "winston-amon"], "value": 1},
  {"key": ["winston", "amqp", "autoqueue"], "value": 1},
  {"key": ["winston", "amqp", "bip"], "value": 1},
  {"key": ["winston", "amqp", "bipio"], "value": 1},
  {"key": ["winston", "amqp", "forge.js"], "value": 1},
  {"key": ["winston", "amqp", "smart-router"], "value": 1},
  {"key": ["winston", "amon", "winston-amon"], "value": 1}
]}
```

Codependencies

The meat of the analysis

- So this is all well and good, but what the heck are you doing?!?!

For module **NAME** generate a matrix by:

- **Rank** codependencies based on number of times they appear
- For each codependency **C** in the **SET** of the top **N**:
Rank **SET** – **{C}** by number of times they appear to create **Row[C]**

Codependencies

Understanding codependencies through winston

- This last step yields a matrix for the codependency relationship:

	asyn...	expr...	opti...	requ...	unde...
asyn...	0.0000	553.0000	534.0000	837.0000	1359.0000
expr...	553.0000	0.0000	314.0000	365.0000	648.0000
opti...	534.0000	314.0000	0.0000	335.0000	448.0000
requ...	837.0000	365.0000	335.0000	0.0000	786.0000
unde...	1359.0000	648.0000	448.0000	786.0000	0.0000

Codependencies

Understanding codependencies through winston

- Now we need to weight the matrix based on the overall appearance of these codependencies

240		async		0.2761
207		underscore		0.2382
163		express		0.1875
133		request		0.1530
126		optimist		0.1449
869	total			

Codependencies

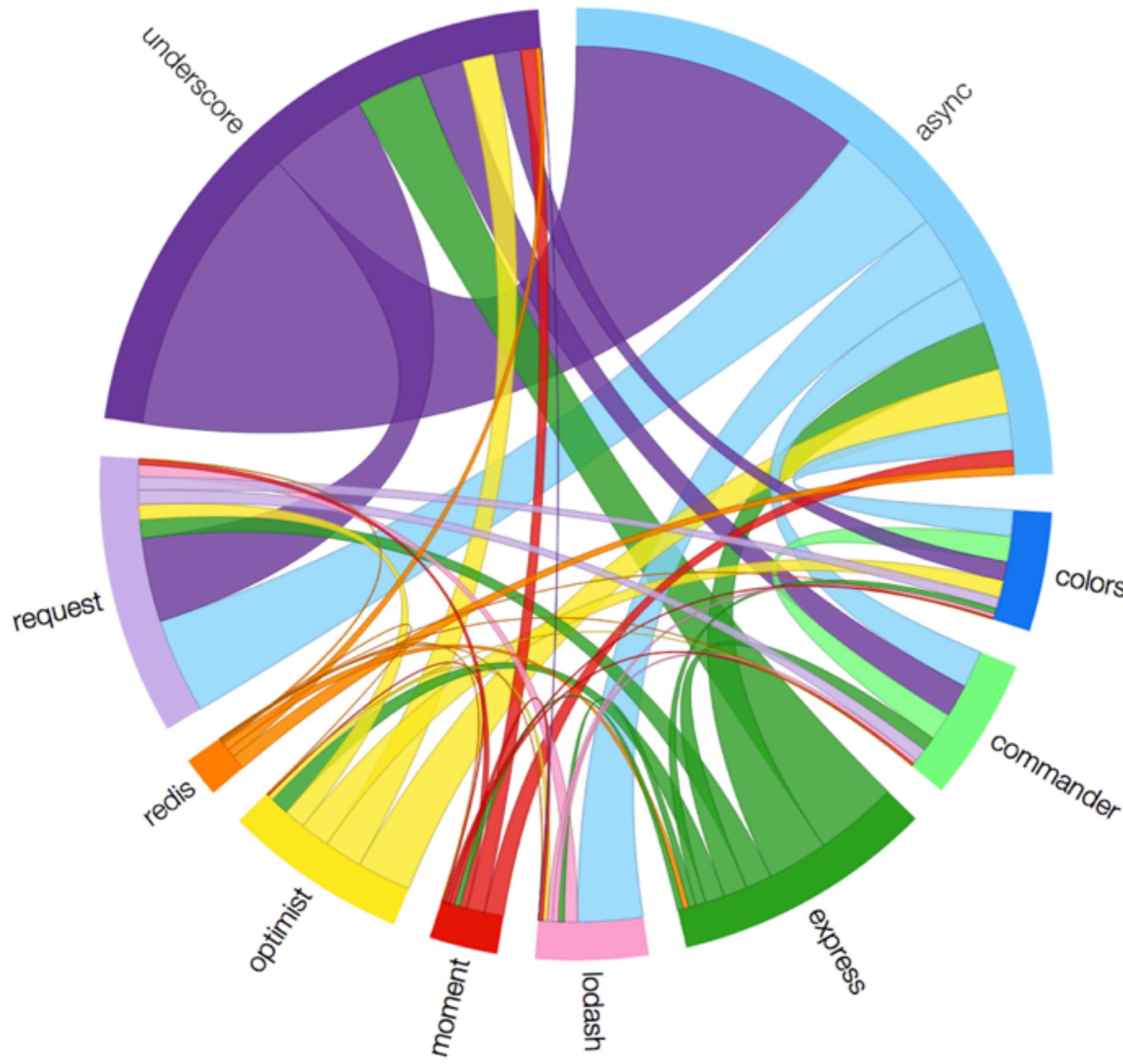
Understanding codependencies through winston

- To do this we go and calculate the codependencies for each of the members

ASYNC

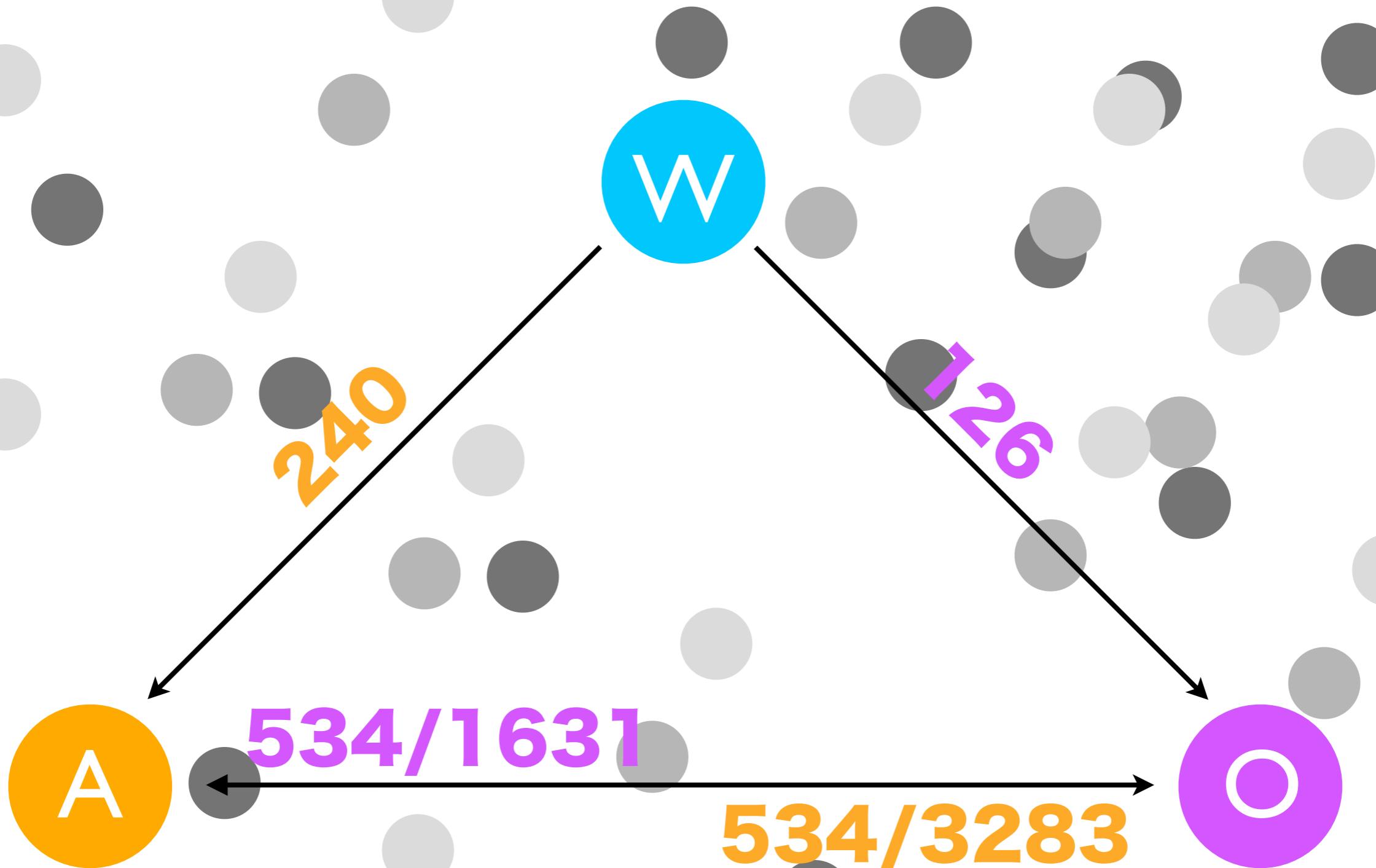
1359		underscore	0.4139
837		request	0.2549
553		express	0.1684
534		optimist	0.1626
3283	total		

AND THUS MEANINGFUL GRAPHS





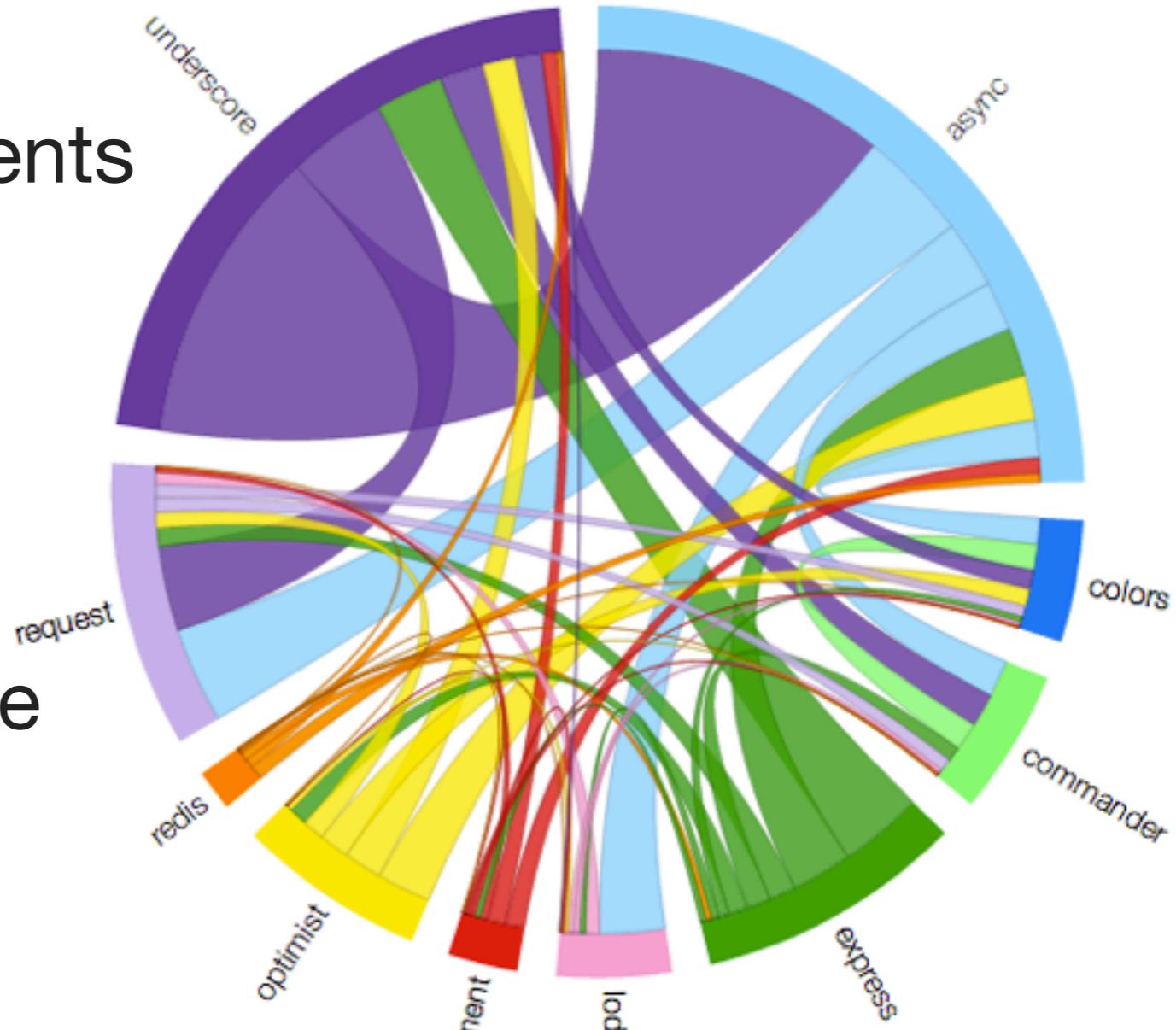
WAT?



Codependencies

Reading the tea leaves of dense data visualization

- **The size of the arc** represents the degree of the codependency relationship with the parent module.
- **The size of the chord** represents the degree of the codependency relationship between each pair.
- **The color of the chord** represents the “dominant” module between the pair.



winston



PAUSE FOR
DEMO

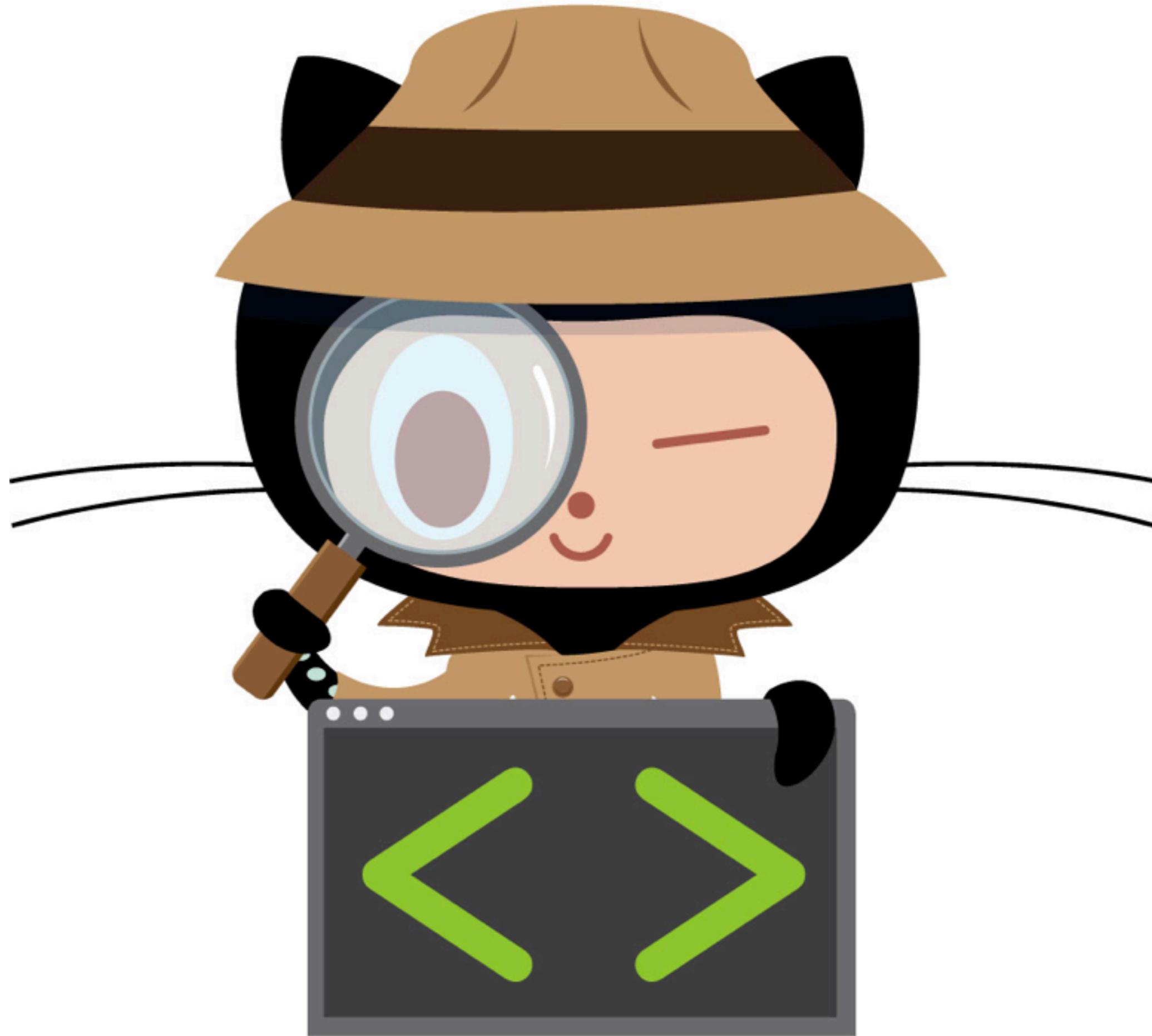
COMEDY C

ALL OPEN SOURCE

Seriously... what else do people do?

indexzero/npm-codependencies

indexzero/npm-comp-stat-www

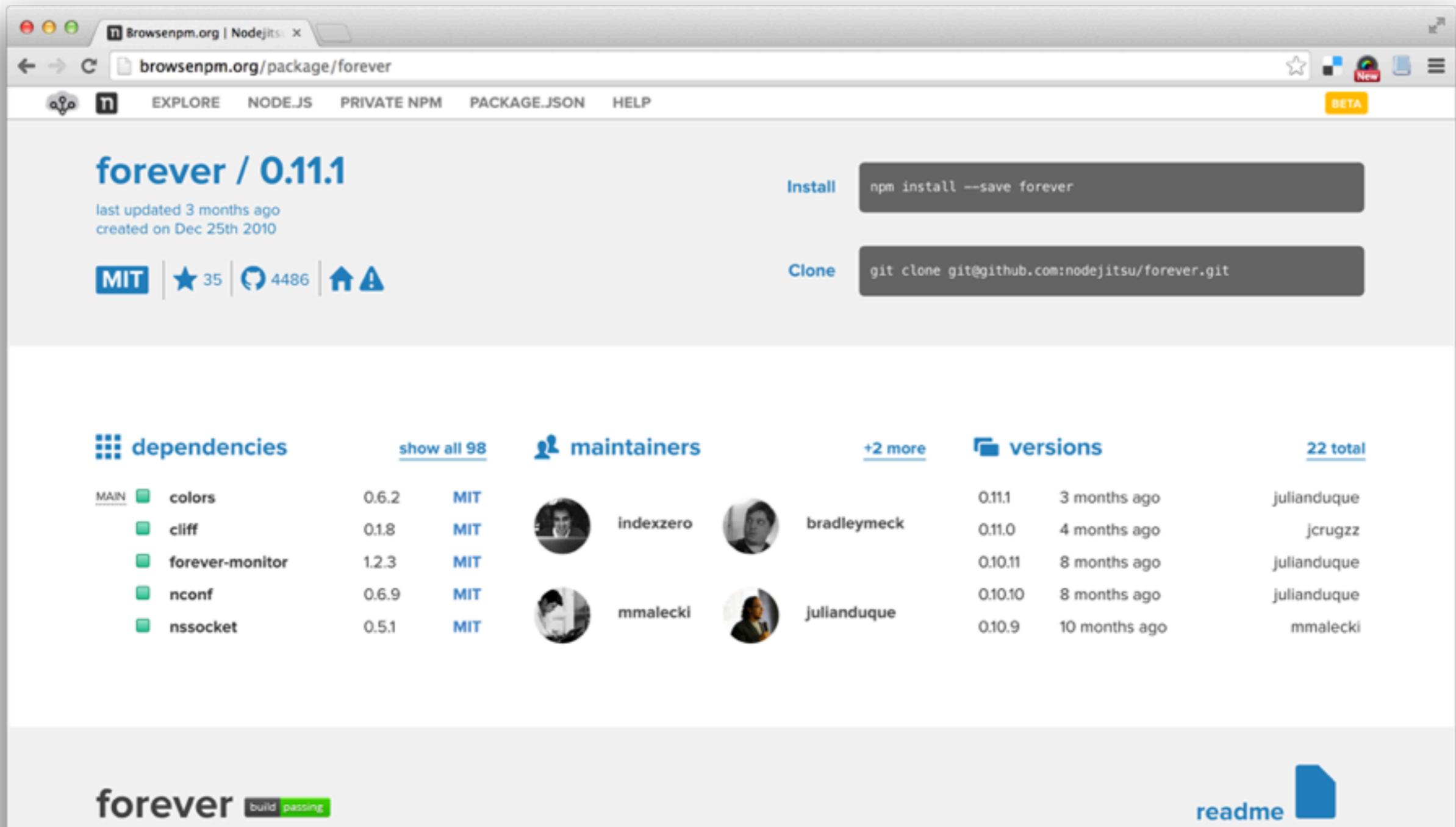






browsenpm.org

github.com/nodejitsu/browsenpm.org



The screenshot shows a web browser window with the URL `browsenpm.org/package/forever` in the address bar. The page is titled "forever / 0.11.1". It displays the package's dependencies, maintainers, and versions. The "Install" section contains the command `npm install --save forever`, and the "Clone" section contains the command `git clone git@github.com:nodejitsu/forever.git`. The page is labeled "BETA" in the top right corner.

dependencies	show all 98	maintainers	+2 more	versions	22 total		
colors	0.6.2	MIT	indexzero	bradleymeck	0.11.1	3 months ago	julianduque
cliff	0.1.8	MIT			0.11.0	4 months ago	jcrugzz
forever-monitor	1.2.3	MIT			0.10.11	8 months ago	julianduque
nconf	0.6.9	MIT	mmalecki	julianduque	0.10.10	8 months ago	julianduque
nssocket	0.5.1	MIT			0.10.9	10 months ago	mmalecki

forever build passing

readme 

**BUT! THAT'S
NOT ALL!**



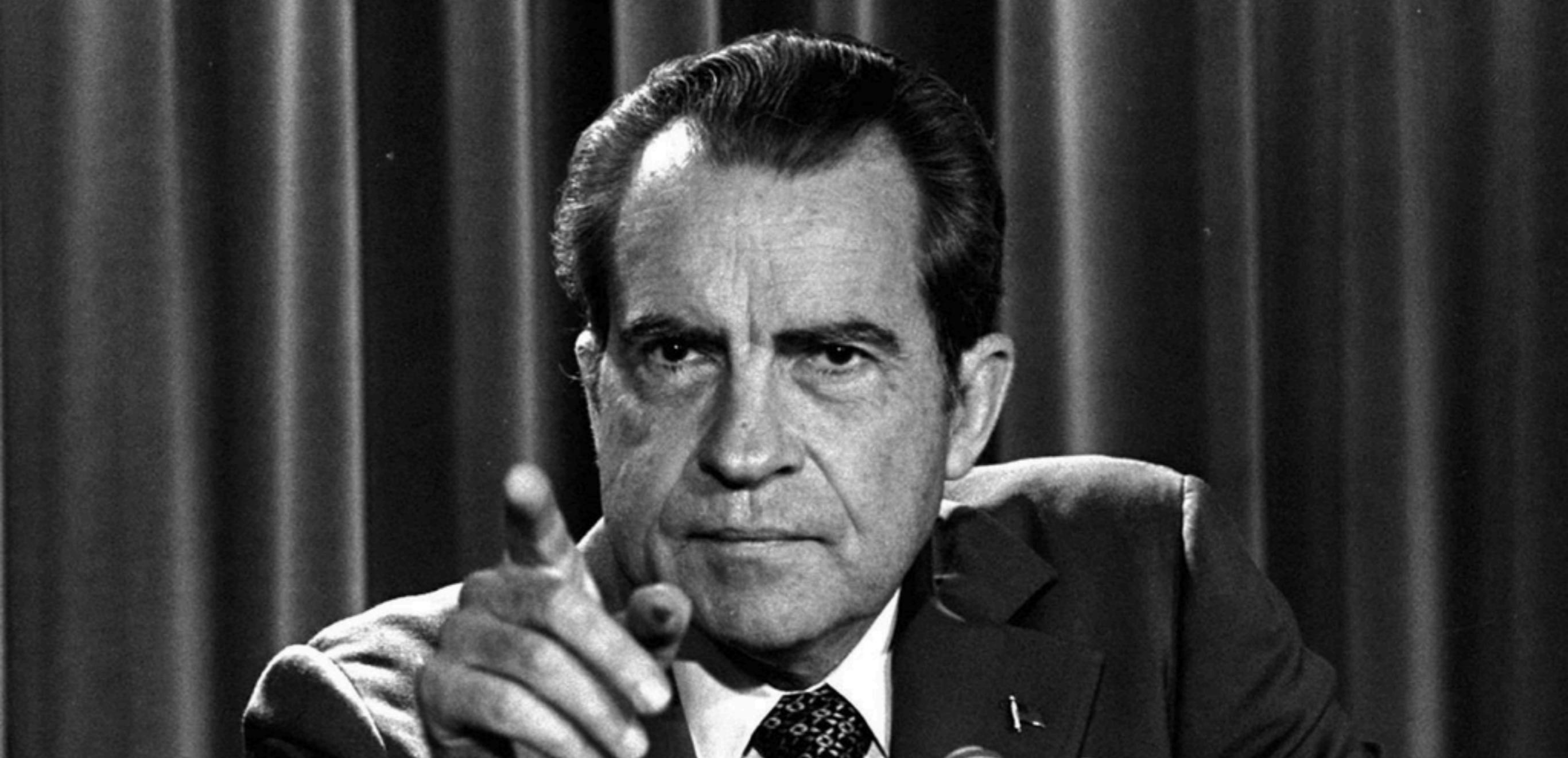


**NO, NOT TOP HATS
AND MONACLES**

I should use static analysis with **npm**

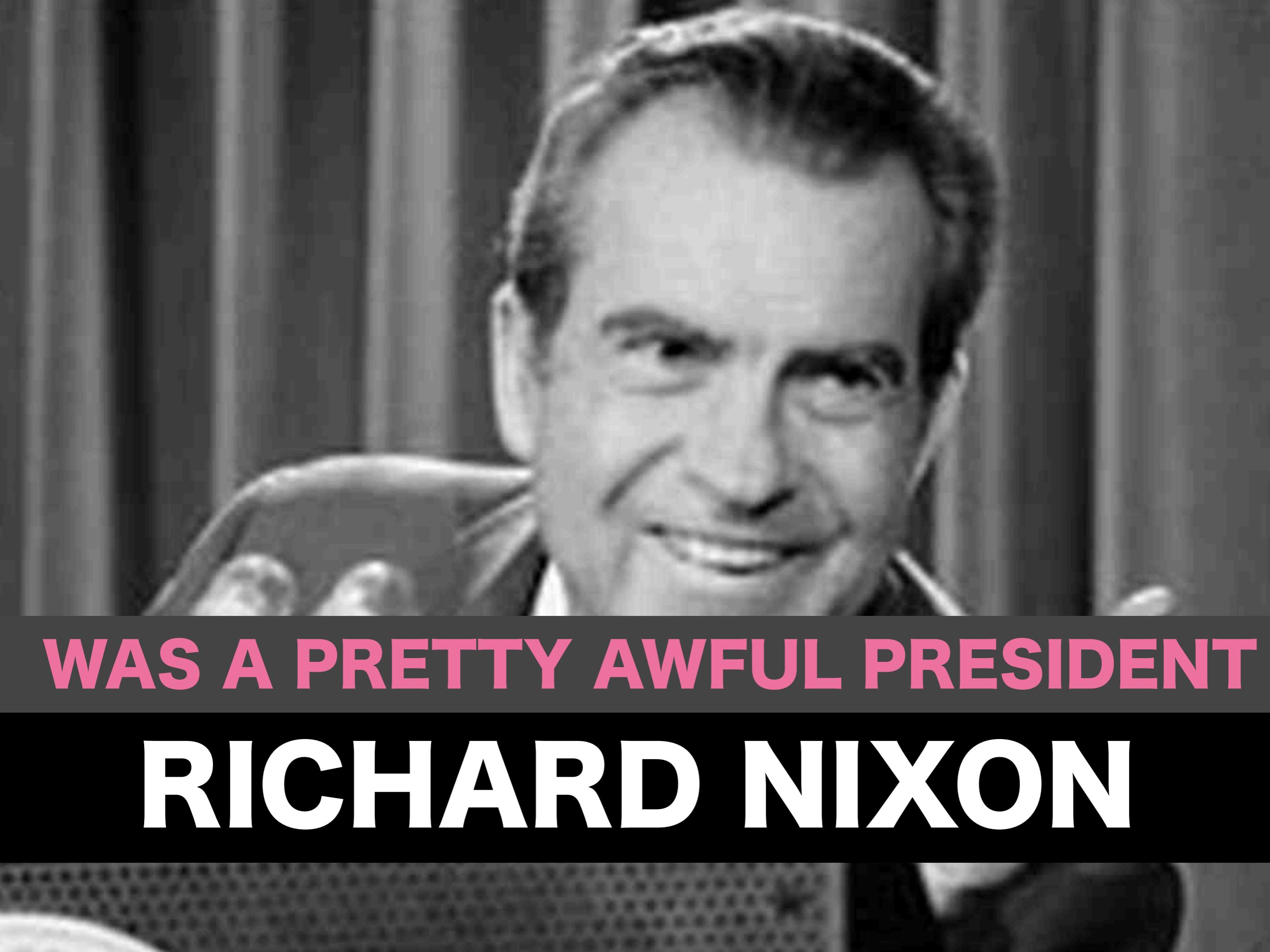
ANALYZE ALL THE THINGS!



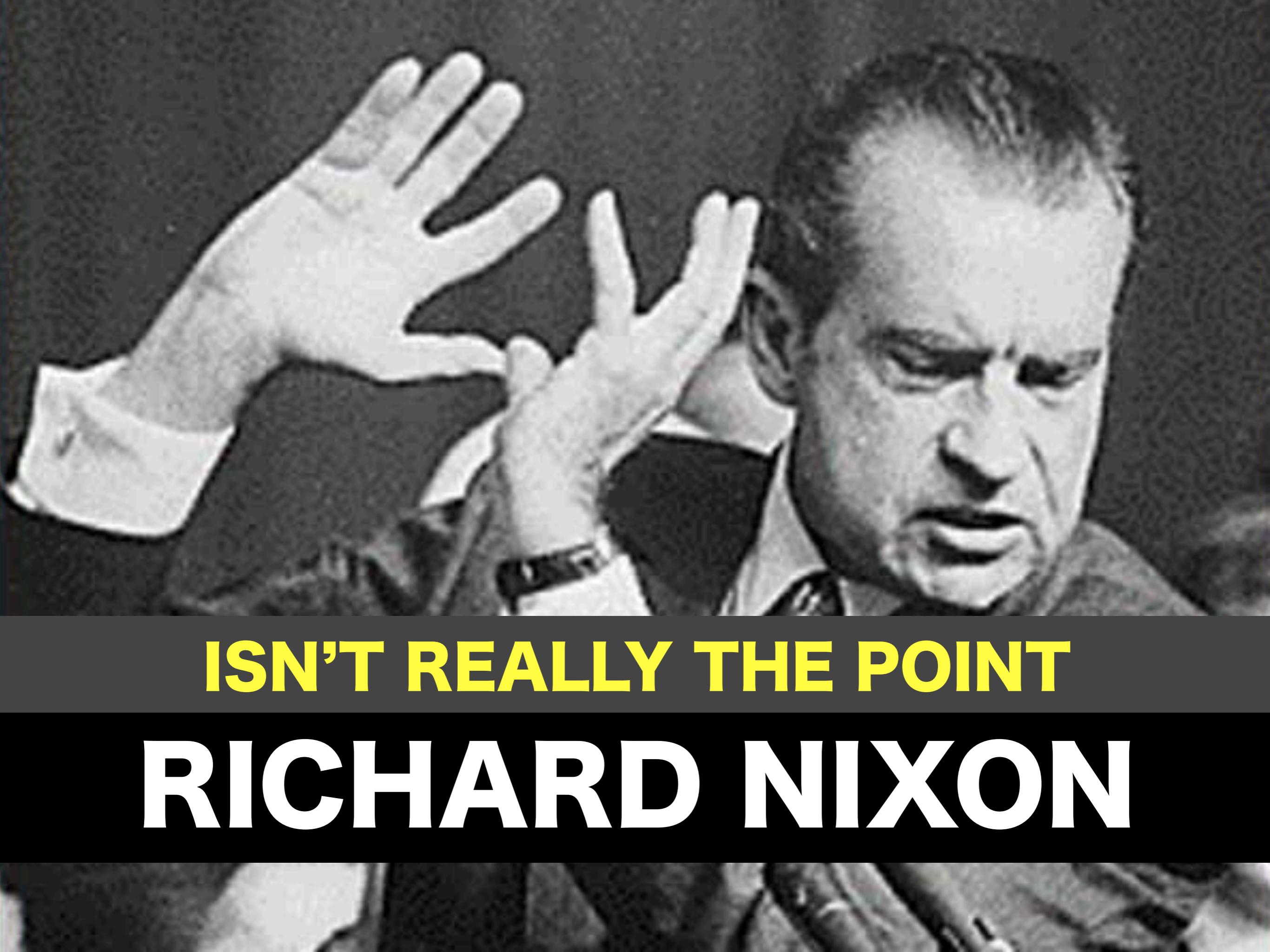


WAS THE 37th PRESIDENT

RICHARD NIXON



WAS A PRETTY AWFUL PRESIDENT
RICHARD NIXON



ISN'T REALLY THE POINT

RICHARD NIXON



GRATUITOUS HEAD IN A JAR

RICHARD NIXON

RICHARD NIXON

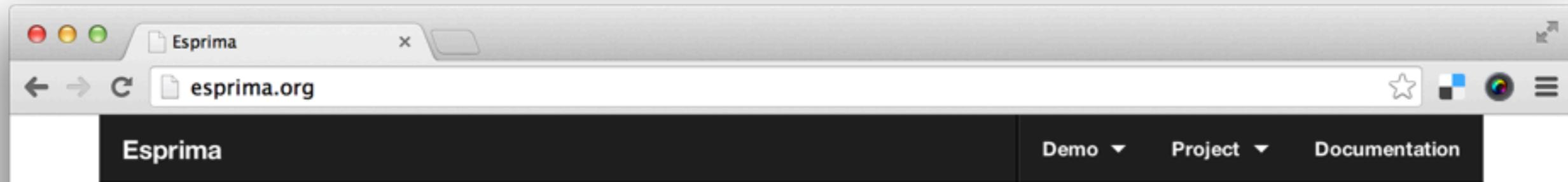
COMEDY
CENTRAL



WELL THE FEAR OF A
SILENT MAJORITY
ANYWAY.....

I should use static analysis with

How the what exactly?



ECMAScript parsing infrastructure for multipurpose analysis

Esprima is a high performance, standard-compliant **ECMAScript** parser written in **ECMAScript** (also popularly known as **JavaScript**).

Features

- Full support for ECMAScript 5.1 ([ECMA-262](#))
- Sensible [syntax tree format](#), compatible with Mozilla [Parser AST](#)
- Optional tracking of syntax node location (index-based and line-column)
- Heavily tested (> 700 [tests](#) with [full code coverage](#))
- [Partial support](#) for ECMAScript 6

Esprima serves as an important **building block** for some JavaScript language tools, from [code instrumentation](#) to [editor autocomplete](#).

```
1 var capitalDb = {  
2   Indonesia: 'Jakarta',  
3   Germany: 'Berlin',  
4   Norway: 'Oslo'  
5 };  
6  
7 // Property completion: "capitalDb." and press Ctrl+Space.  
8 capitalDb.  
9           Germany : String  
          Indonesia : String
```

Once the full syntax tree is obtained, various **static code analysis** can be applied to give an insight to the code: [syntax visualization](#), [code validation](#), [editing autocomplete](#) (with type inferencing) and [many others](#).

Regenerating the code from the syntax tree permits a few different types of **code transformation**, from a simple [rewriting](#) (with specific formatting) to a more complicated [minification](#).

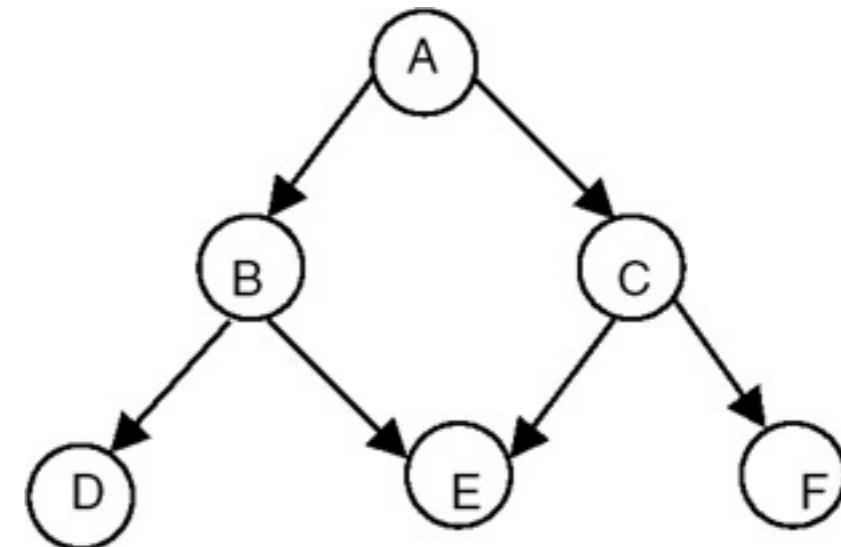
Esprima runs on many popular web browsers, as well as other ECMAScript platforms such as [Rhino](#), [Nashorn](#), and

I should use static analysis with

How do you make the sausage?



ESPRIMA 



CODE

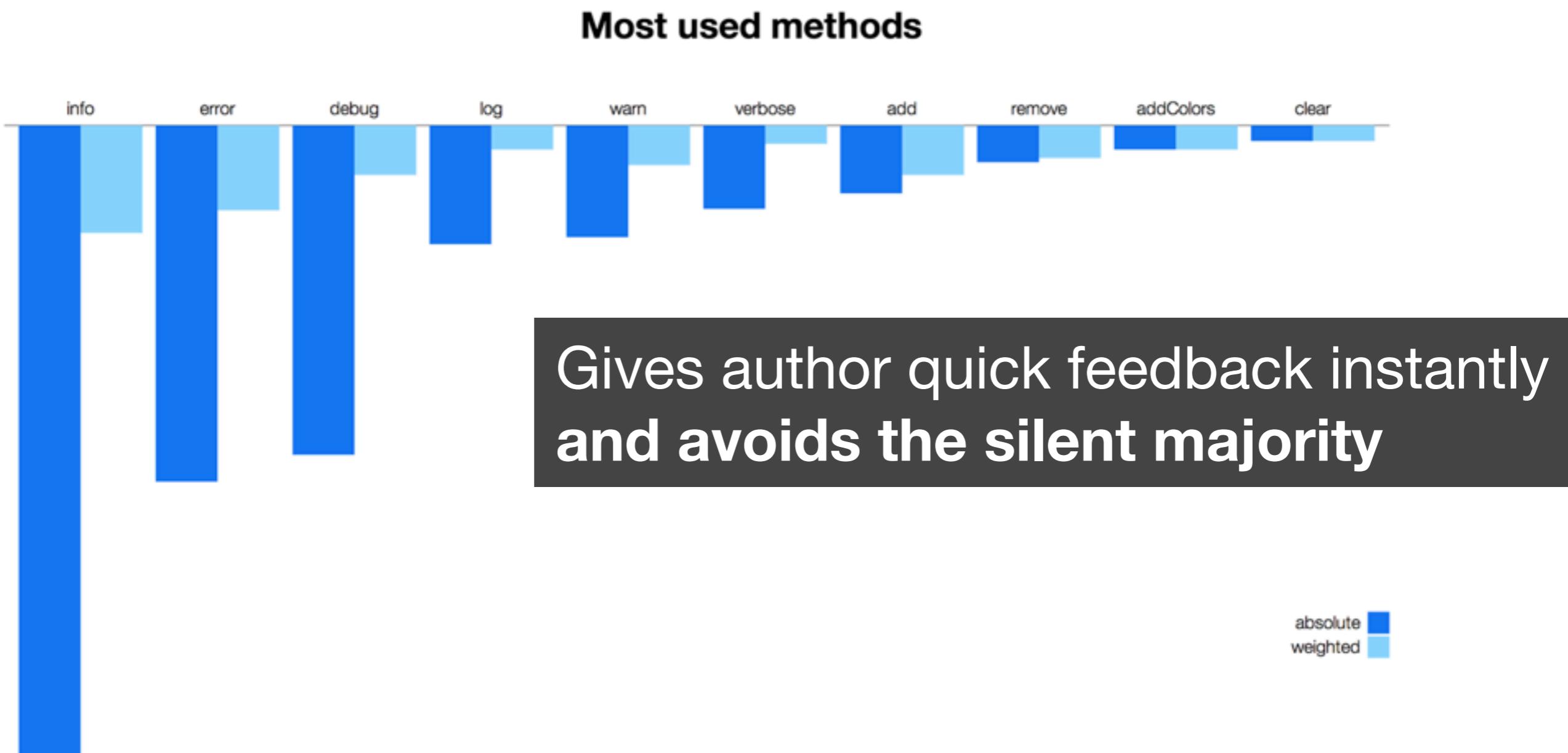
AST

NOW DO THAT FOR
EVERY SOURCE FILE IN
EVERY DEPENDENT MODULE

I should use static analysis with

GET TO THE POINT ALREADY CHARLIE!

Using **esprima** and a **dependency graph** we were able to perform rudimentary analysis of hottest code paths.



ALL OPEN SOURCE

Yup... just like before.

[indexzero/npm-static-stats](https://github.com/indexzero/npm-static-stats)

[indexzero/npm-pipeline](https://github.com/indexzero/npm-pipeline)



nodejitsu

FIN