**# Install the tools (after NPM):**

```
sudo npm install -g bower gulp karma
```

**# Start a project:**

```
npm init         answer the prompts with defaults
npm install --save bower karma
npm install --save gulp gulp-util gulp-plumber gulp-watch
npm install --save gulp-uglify gulp-concat gulp-connect

bower init       answer the prompts with defaults

# tell Bower where to put libraries
echo "{"directory":"www/lib"}" >.bowerrc

bower install --save bootstrap jquery
bower install --save angular angular-route
bower install --save angular-animate angular-mocks


# run a build:
gulp
```

**# VERY OPTIONAL:**

```
# avoiding “sudo” or “run as admin”:

npm config set prefix c:\whatever -g
\
# prove to yourself that NPM works:
rm -rf node_modules
npm install


# if you already have bower.json populated:
bower install


# Git at the command line:

git init           make this a Git project
git add -n .       see what it will add
vi .gitignore      adjust ignores
git add .          actually add
git commit         commit
```

# How Many Watches/Bindings Is Too Many?

Some browsers are faster than others. Older mobile browsers are slow.

Complex watches are more expensive than simple watches.

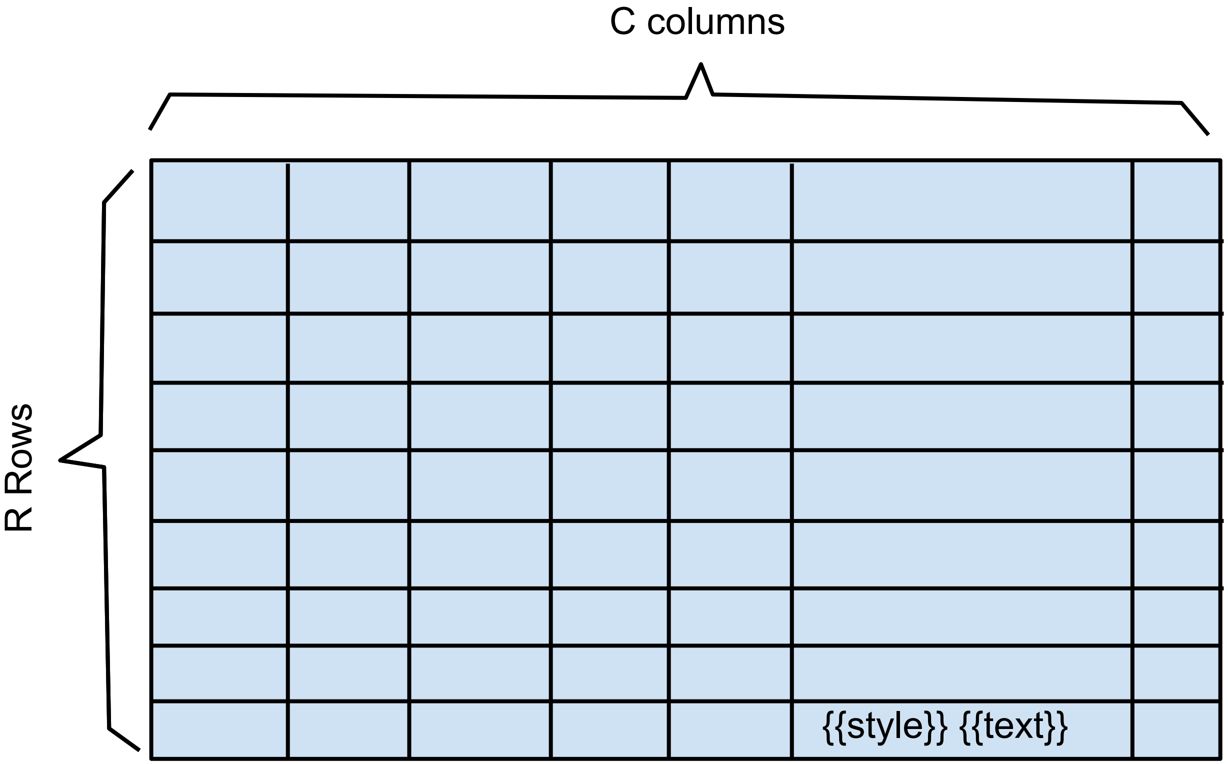
Guidelines:

100s = OK
1000s = SLOW

Hard to get in trouble with manually placed bindings.

Easy to get in trouble with (nested) ng-repeat.

An easy way to get in trouble: naive table/grid.



C columns

R Rows

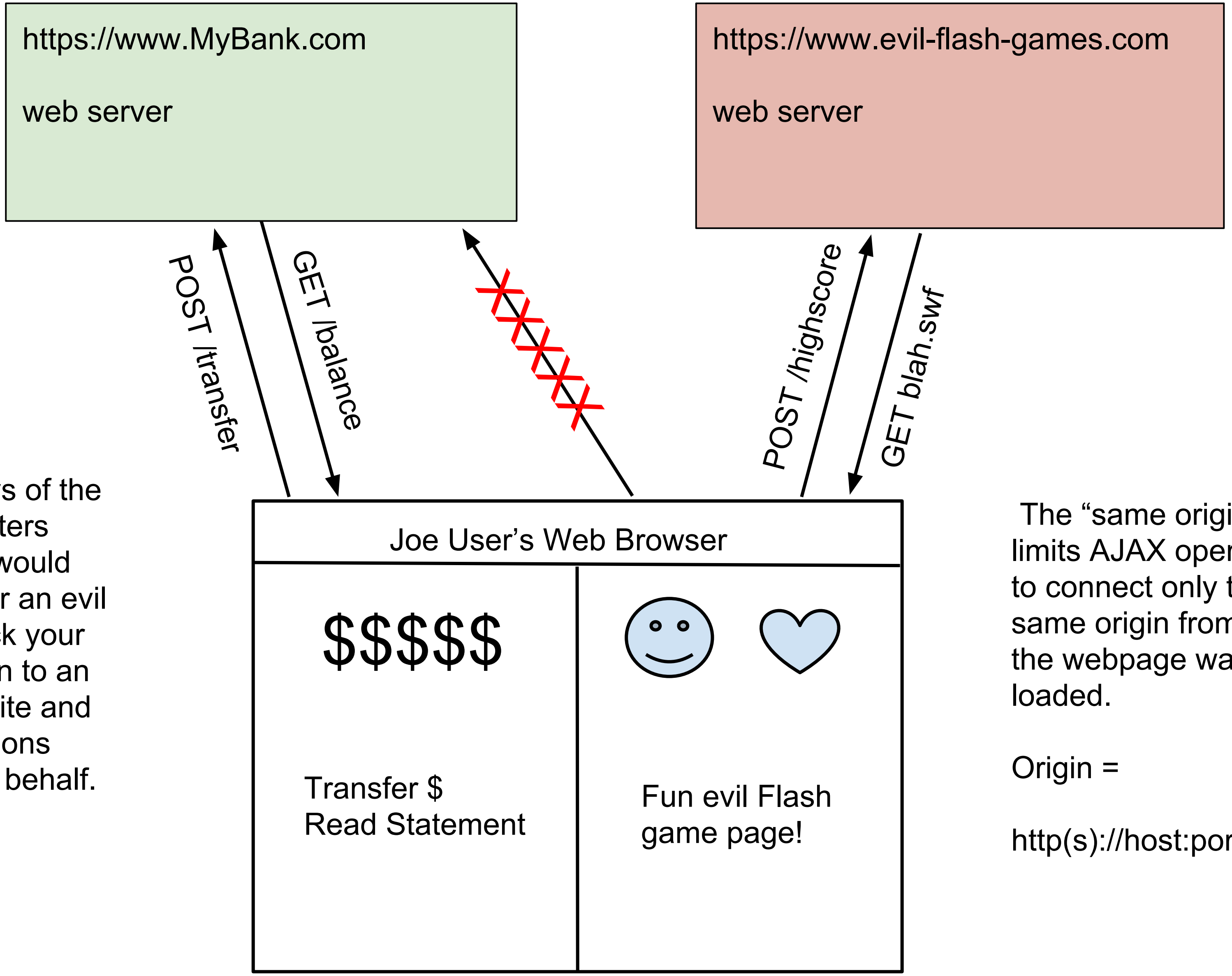
{{style}} {{text}}

R * C * 2 watches per cell = total number of watches

Example:  100 * 20 * 2 = 4000 watches   = SLOW

# CORS 1/2 : Same Origin Policy

In the early days of the web, implementers realized that it would be very easy for an evil website to hijack your login/connection to an important website and execute operations silently on your behalf.



https://www.MyBank.com

web server

https://www.evil-flash-games.com

web server

POST /transfer

GET /balance

POST /highscore

GET blah.swf

Joe User’s Web Browser

$$$$$

Transfer $
Read Statement

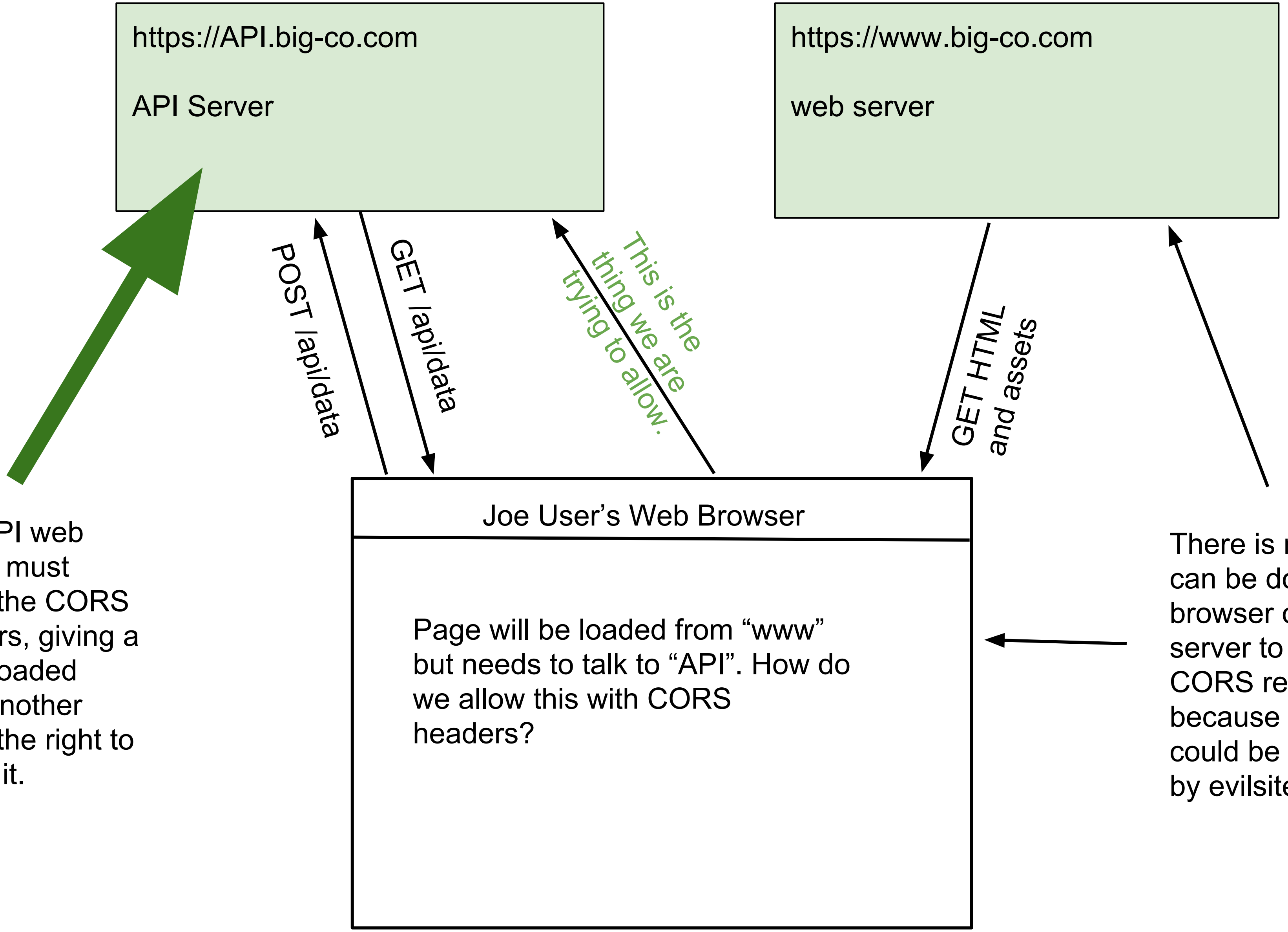
Fun evil Flash game page!

The “same origin policy” limits AJAX operations to connect only to the same origin from which the webpage was loaded.

Origin =

http(s)://host:port/

# CORS 2/2 : CORS headers, where do they go?



https://API.big-co.com

API Server

https://www.big-co.com

web server

POST /api/data

GET /api/data

This is the thing we are trying to allow.

GET HTML and assets

Joe User’s Web Browser

Page will be loaded from “www” but needs to talk to “API”. How do we allow this with CORS headers?

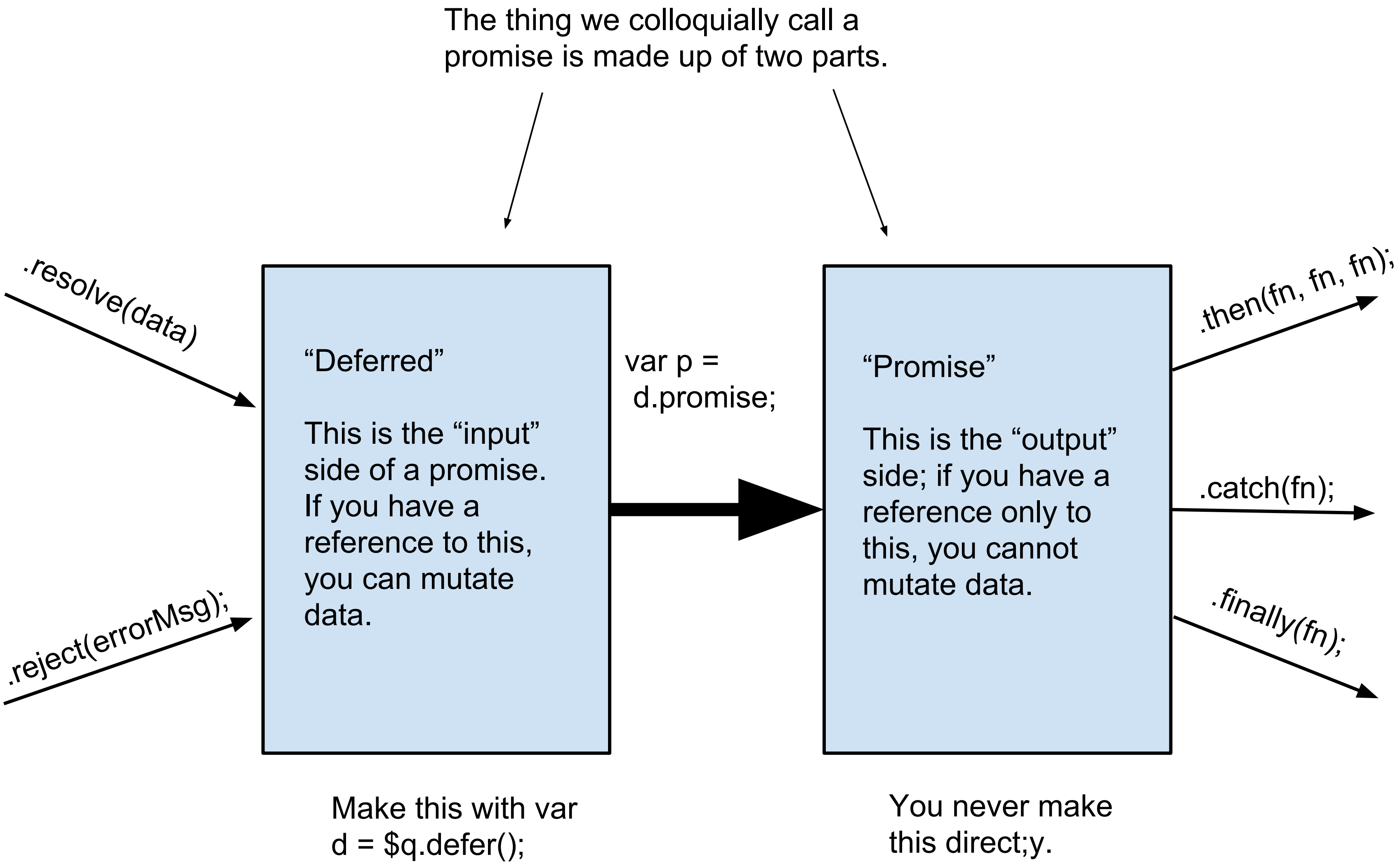
The API web server must serve the CORS headers, giving a page loaded from another origin the right to talk to it.

There is nothing that can be done from the browser or asset server to enable CORS requests, because if there was, it could be used easily by evilsite.org.

| Language | Depend. Mgmt | Build | Dep. file | Setup | |
|---|---|---|---|---|---|
| Java | Maven | Maven | pom.xml | | no such feature |
| **Node / JS** | **NPM** | **Grunt / Gulp** | **package.json** | npm init | **npm install --save packagename** |
| **Client side JS** | **Bower** | **Grunt / Gulp** | **bower.json** | bower init | **bower install --save packagename** |
| C# | nuget | MSBuild | nuspec.xml | | |
| Ruby | Gem | Bundle | Gemfile | | |
| Clojure | Leiningen | Leiningen | project.clj | | no such feature |
| Perl | CPAN | - | ? | | cpanm |
| PHP | Composer | - | ? | | |
| Python | | | | | |

oasisdigital.com

Understanding the AngularJS $q Promise API



The thing we colloquially call a promise is made up of two parts.

.resolve(data)

.reject(errorMsg);

“Deferred”

This is the “input” side of a promise. If you have a reference to this, you can mutate data.

var p =
 d.promise;

“Promise”

This is the “output” side; if you have a reference only to this, you cannot mutate data.

.then(fn, fn, fn);

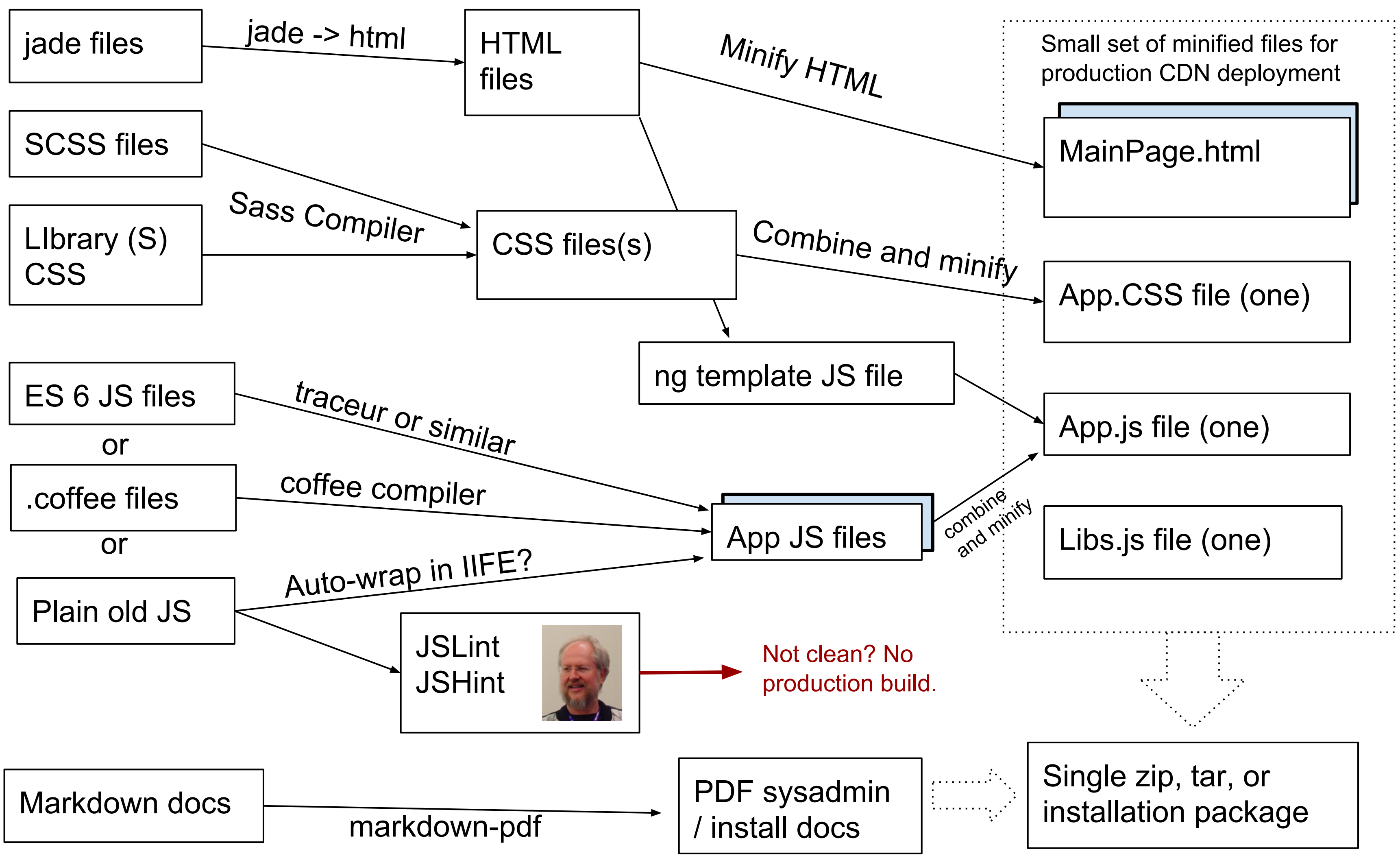
.catch(fn);

.finally(fn);

Make this with var
d = $q.defer();

You never make
this direct;y.

# Typical AngularJS Client-app build process - simplified



jade files

jade -> html

HTML files

Minify HTML

Small set of minified files for production CDN deployment

MainPage.html

SCSS files

Sass Compiler

LIbrary (S) CSS

CSS files(s)

Combine and minify

App.CSS file (one)

ng template JS file

App.js file (one)

ES 6 JS files

traceur or similar

or

.coffee files

coffee compiler

or

App JS files

combine and minify

Libs.js file (one)

Plain old JS

Auto-wrap in IIFE?

JSLint JSHint

Not clean? No production build.

Markdown docs

markdown-pdf

PDF sysadmin / install docs

Single zip, tar, or installation package

All of the above is for a web SPA; a project typically has a server side also.

# SSH to GitHub.com Through a Corporate Network

by **Paul Spears**, Oasis Digital Angular Boot Camp instructor

A large portion of modern web application development involves downloading and exploring the capabilities third party tools and libraries. At the center of this tooling is github.com. With over 4 million users GitHub is the most popular resource for publicly sharing code. GitHub has been integrated into the heart of many tools and access to such a resource is invaluable. Certain network environments can prevent this.

I use NPM to manage my application, tooling, dependencies. NPM looks at the dependencies of various node modules and resolves them automatically. This is done by cloning them from remote repositories, often, GitHub. If this process is performed on a corporate network with strict access policies a user may encounter any number of possible errors. The most common is `connect to host github.com port 22: Bad file number.` or `github.com[0: 207.97.227.239]: errno=Connection timed out fatal: unable to connect a socket (Connection timed out)`

This is usually an indication that SSH traffic is not allowed on the current network.

Lets spend a minute analyzing what is going on when we encounter this error using NPM so that we know what to do about it. When we attempt to install a tool or library using NPM a list of dependencies and their corresponding repositories are calculated. In this situation one of the dependencies supplied a repository URL that indicates that Git should use SSH to fetch the code. When Git attempts to access this repository the network firewall/proxy denies the request.

So what do we do to resolve this issue? Normally we would simply reissue the request using the projects HTTPS address. Unfortunately, we do not alway have access to the dependency list of the module we are trying to install and have to accept the URL we are given. That leaves us with one of two choices. One, use URL rewriting in Git. This is by far the easiest approach to implement. The premise is simple. We can instruct tell Git to match URLs and redirect the request to another URL of our choice. For our purposes the command to achieve this is as follows:

```
git config --global url."https://github".insteadOf git://github
```

The previous command adds an entry to our git configuration that will send all Git traffic intended for git://github to the https equivalent. Now if a dependency requests an ssh or git protocol

Decisions were made in three places that stack to create a nasty mess. Some dependencies for node_modules only list a github.com SSH repo URL. This forces git to use SSH. If the HTTPS URL was specified none of this would be necessary. Secondly, GitHub requires an SSH key regardless of who is making the request or what is being requested. This forces us to setup an account and SSH Key. Lastly, with a large number of corporate networks, proxies and firewalls often disallow the use of SSH traffic and an HTTPS redirection is needed.

Sign up for a github.com. Yes, I know that sounds ridiculous but github only allows SSH traffic with a properly configured SSH key. Even when using git@github.com To do this you will need to create an account and add your public SSH key to it.

Generate your SSH key. Using your git bash prompt (in Windows) issue the following commands:

```
ssh-keygen -t rsa -C "your_email@example.com"
```

```
ssh-agent -s
```

```
//This may not work but that is usually OK
ssh-add ~/.ssh/id_rsa
```

```
clip < ~/.ssh/id_rsa.pub
```

Your SSH key is now on the clipboard. Add it to your Github account.

Now we need to inform SSH to tunnel through HTTPS

find or create the following file ~/.ssh/config and add the following:

```
ProxyCommand /bin/connect.exe -H proxy.server.name:3128 %h %p (for Windows using Git Bash)
```

```
ProxyCommand nc -x MY_PROXY_HOST:MY_PROXY_PORT %h %p (for *nix)
```

```
Host github.com
  Hostname ssh.github.com
  Port 443
```

For the proxy URL if a username and password is required, the user should be specified but the password should be entered when prompted. (This likely depends upon the proxy in use).

https://help.github.com/articles/generating-ssh-keys
https://help.github.com/articles/using-ssh-over-the-https-port
http://stackoverflow.com/questions/5103083/problem-of-testing-ssh-in-git-behind-proxy-on-window-7