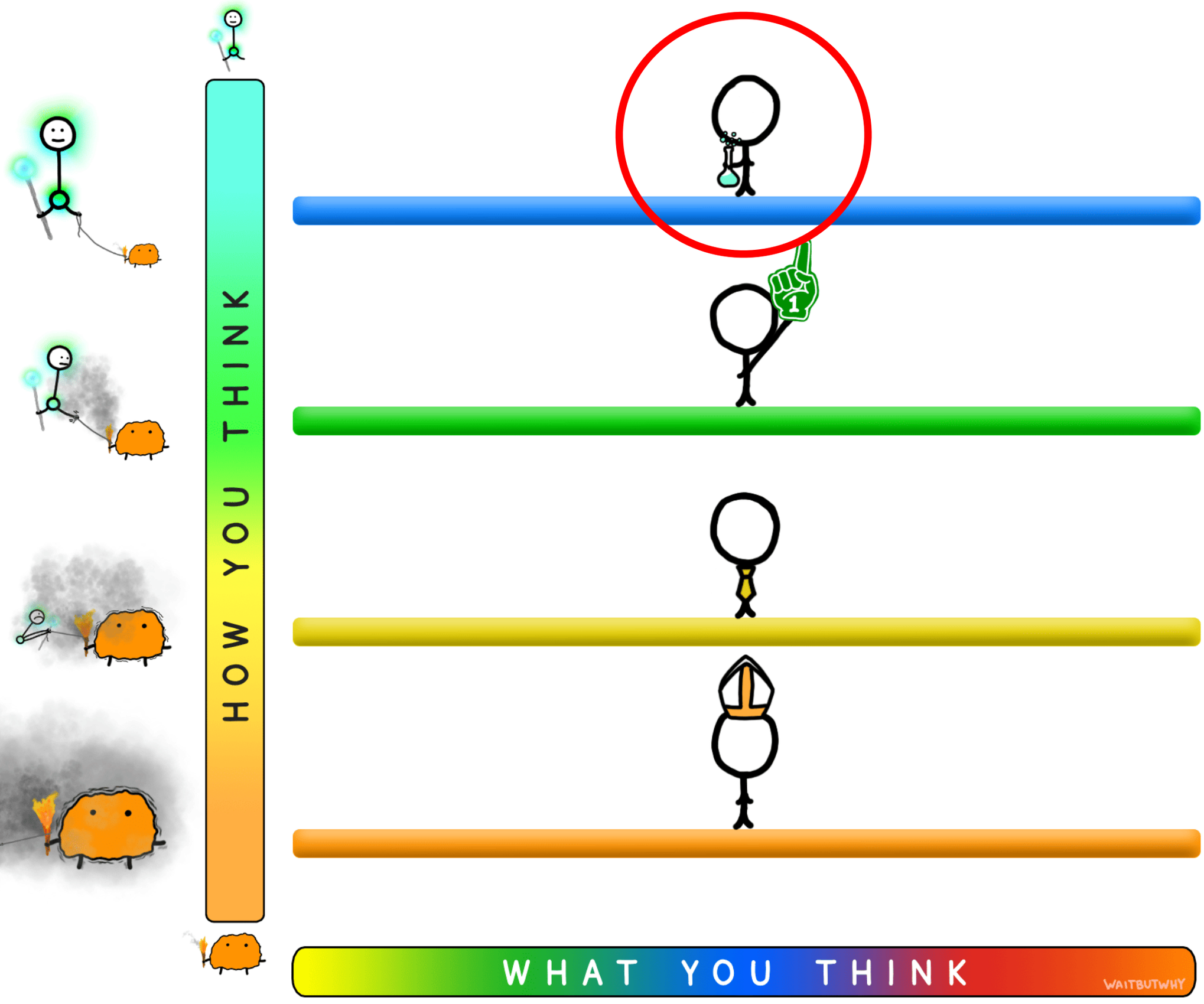
I  JamStack

BUT

Let’s Get Real


HOW YOU THINK
WHAT YOU THINK
WAITBUTWHY

The Problems with Static Architecture

1

Unbounded Page Growth

A Static Site Axiom

The more unique pages your static site has, the slower it compiles.

As your organization grows, you will continue adding more unique pages.

Therefore as your organization grows, your site’s compilation time gets slower.

The more unique pages your static site has, the slower it compiles.

But, Hugo! - Hugo is great & fast but this is still true

But, Gatsby Cloud! - This is a paid, remote only workaround

As your organization grows, you will continue adding more unique pages.

But, we have no blog! - Targeted marketing, custom audiences…

But, it will be a long time! - Ok, still true

But, we won’t grow! - Then you don’t have scale problems

2

Dynamic Functionality

Some Examples

Forms

Email Signups

Clearbit Auto Form Fill

User Login

API routes to protect access tokens

API routes to combine different api sends & responses

IP-based location lookup on request

How To Fix This

1. Pay a third-party service provider
2. Make it yourself, run a server with an API

3

Content Preview

4

Fast-changing, Dynamically Rendered Public Pages

The Stress of Web Architec Decisions: The Game!

All-in Dynamic

- ✓ no limits
- ✓ no refactors later
- ✗ complexity when not needed
- ✗ slower time to market
- ✗ more expensive

Worst Case Scenario

Competitors beat you to market with a simpler, faster option that costed them less. Your resources are drained due to the higher costs and complexity.

All-in Static

- ✓ fast, easy start
- ✓ low cost & complexity
- ✗ limitations as discussed
- ✗ getting around them is brutal

Worst Case Scenario

As your product matures, you need crazy workarounds to the limitations of static, which are more expensive and cause more velocity reduction than a dynamic build, competition crushes you over time by being more efficient.

Start Static, Refactor to Dynamic

- ✓ no limits
- ✓ fast, easy start
- ✗ enormously expensive refactor

Worst Case Scenario

You make it to market and start maturing, but your growth stalls and expenses skyrocket as you begin to refactor, drains your resources and competition outpaces you drastically.

Time for a vote:

Which Do You Pick?

One more:

Who Was 100% Confident In Their Vote?

Lets be real:

These Options Are All Awful

The Solution


NEXT
.JS

The Plan:

You Dont Need to Choose Static or Dynamic

Stage One:

One Codebase, Deploy as Static or Dynamic

Example:

Stage Two:

Choose Static or Dynamic Per Route

getStaticProps

getServerProps

getStaticPaths


Static
Limitations, Be Gone!

SPLIT BABEL COMPILATION & EXPORT

MULTI-THREADED EXPORT

1

Unbounded Page Growth

SWAP ANY ROUTE TO DYNAMIC

"API" FOLDER

2

LAMBDA DEPLOYMENT

Dynamic Functionality

FULL SERVER RENDER

DEPLOY AS DYNAMIC

3

Content Preview

EVEN MORE GOODNESS SOON!

4

Fast-changing, Dynamically Rendered Public Pages

USE GETSERVERPROPS

Table Stakes

- ✓ Route-based code splitting, including css
- ✓ Crazy webpack bundle optimization
- ✓ SSR setup out of the box
- ✓ Load data in any way, from any source
- ✓ Client or server side routing, filesystem-based
- ✓ Automatic prefetching
- ✓ Easily customize with config & plugins
- ✓ Automatic typescript config

I 💖 NextJS

BUT

Every Tool Has Its Weaknesses

- ✓ Growing companies
- ✓ Venture-funded companies
- ✓ Web applications

- ✗ Personal websites
- ✗ Small/medium blogs
- ✗ Small business website
- ✗ Informational website

At Least One
ull Time Web De

NextJS is changing fast

It is way more complicated than straight html/css/js

Not even close to future-proof

Thank You!