
NATER JORDE
SENIOR WEB APPLICATION DEVELOPER
HTTPS://WWW.CODE42.COM
HTTPS://GITHUB.COM/NATERGJ
OUR WEB APPLICATION JOURNEY
DENVER STARTUP WEEK 2017

00:00:00

TITLE SPONSORS

TRACK SPONSORS

“Thank you for joining us at the sixth annual Denver Startup Week!”
“Denver Startup Week is the largest free entrepreneurial event of its kind in North America.”
“Thank you to our title sponsors Aging 2.0, Comcast, Chase for Business, the Downtown Denver Partnership and WeWork.”
“This session is part of the Developer Track, one of six programming tracks aimed at supporting the entire entrepreneurial team.”
“Be sure to share your experience at this session online using #DENStartupWeek!” (feel free to add your own hashtag too)

DEN
STARTUP WEEK

HEADLINE SPONSORS

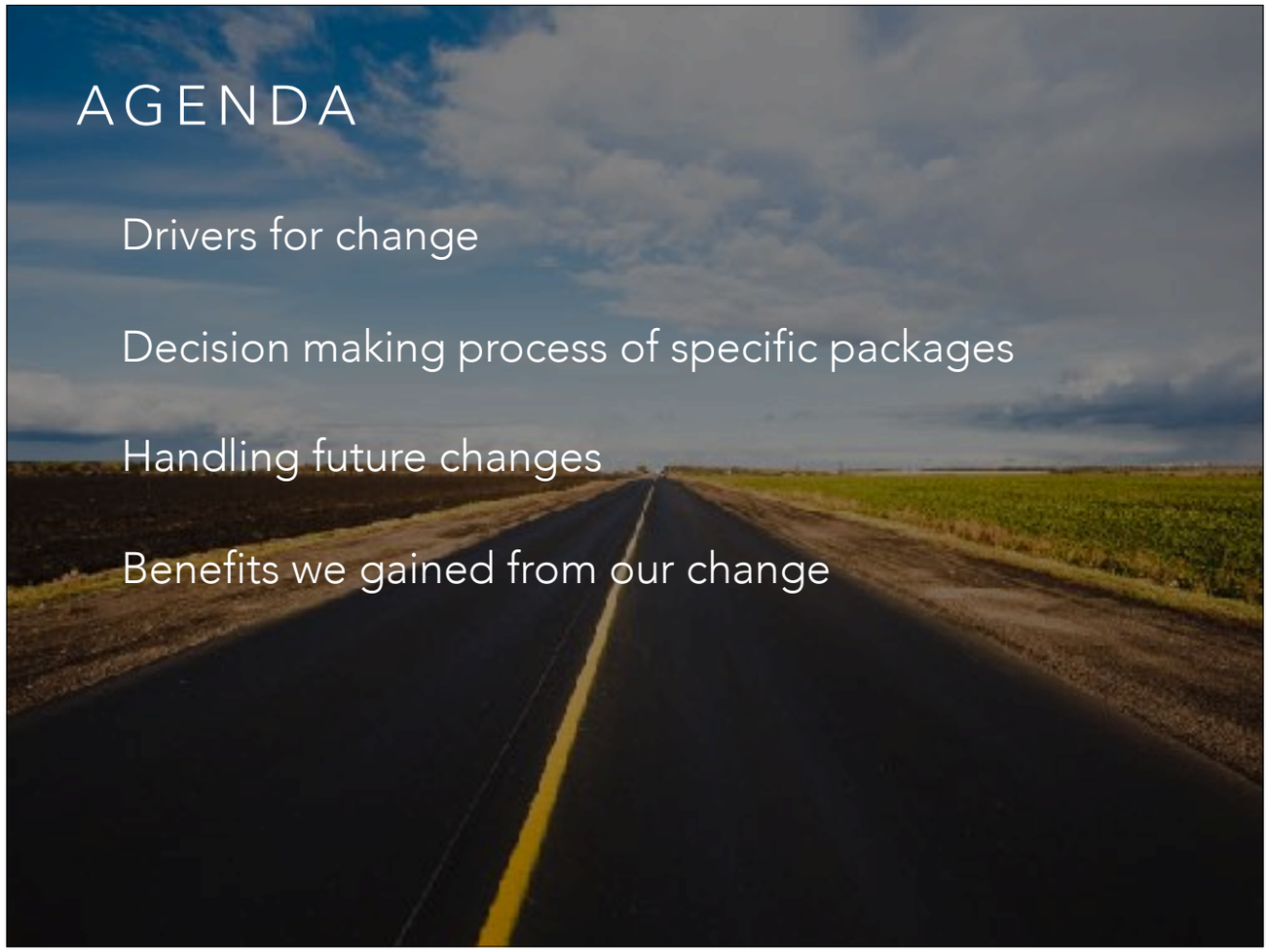
PARTNER SPONSORS

Baker Hostetler
Bradford, LTD
Capital One
Coastal Cloud
Colorado Impact Fund
Connect For Health Colorado
Cooley
Corus360
EKS & H
Event Integrity
Full Contact
Gary Community Investments
General Assembly
Groundfloor Media / CenterTable
Guiceworks
ImageSeller
Ink Monstr
Intelivideo
Luna Gourmet Coffee & Tea
Nanno
Office of Economic Development
Pass Gas Denver
Slalom
Wazee Digital

MEMBER SPONSORS

Accenture
Bridgepoint Education
Butler Snow
Delta Tables
Hogan Lovells
Meyer Law
Name.com
Slifer Smith & Frampton
SoGnar
Sounddown
Swiftpage
The Denver Foundation
Zipcar

Before we get started. Who here is
Web Developer?
Development Manager?
Product Owner?
Project Manager?
Designer?



05:00:00

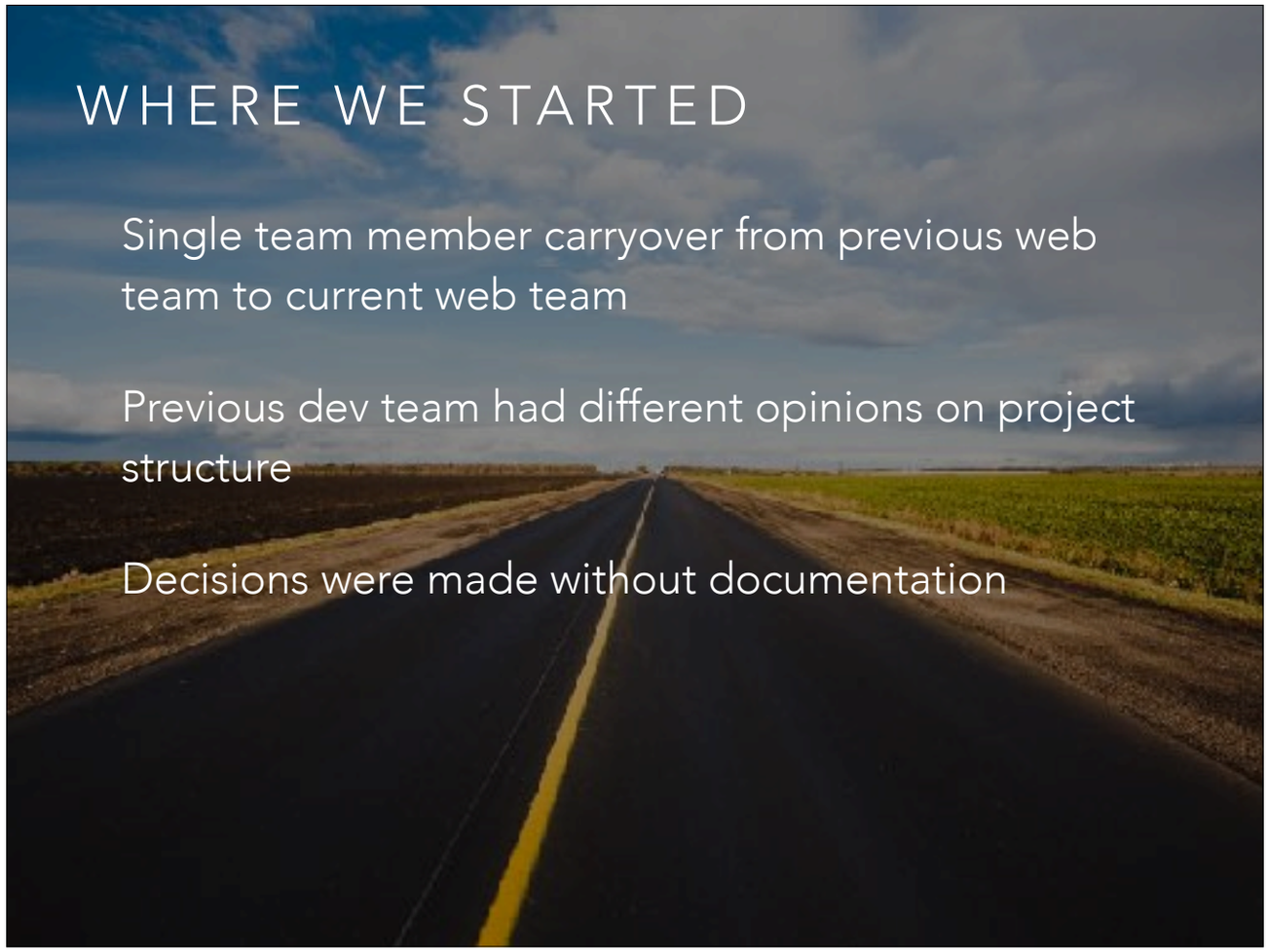
What we'll be covering today:

Drivers for change: What is it that causes us to go on these journeys of change?

Decision making process for specific subset of packages: What we considered when choosing or implementing specific packages

Handling future changes: How do we not become stagnant again?

Benefits we gained from our change: How we're working better.



07:00:00
one year ago:

Single team member carryover from previous web team to current web team:
for many reasons, we had a large turn-over. The new development team (fresh eyes) exposed that our web tech stack as of a year ago and its associated process was not as maintainable as management had thought

Previous dev team had different opinions on project structure:
Ohhhh-pinions. moving on without buy-in is difficult. If I can’t understand why something is done a certain way, I can’t justify continuing to do it that way.

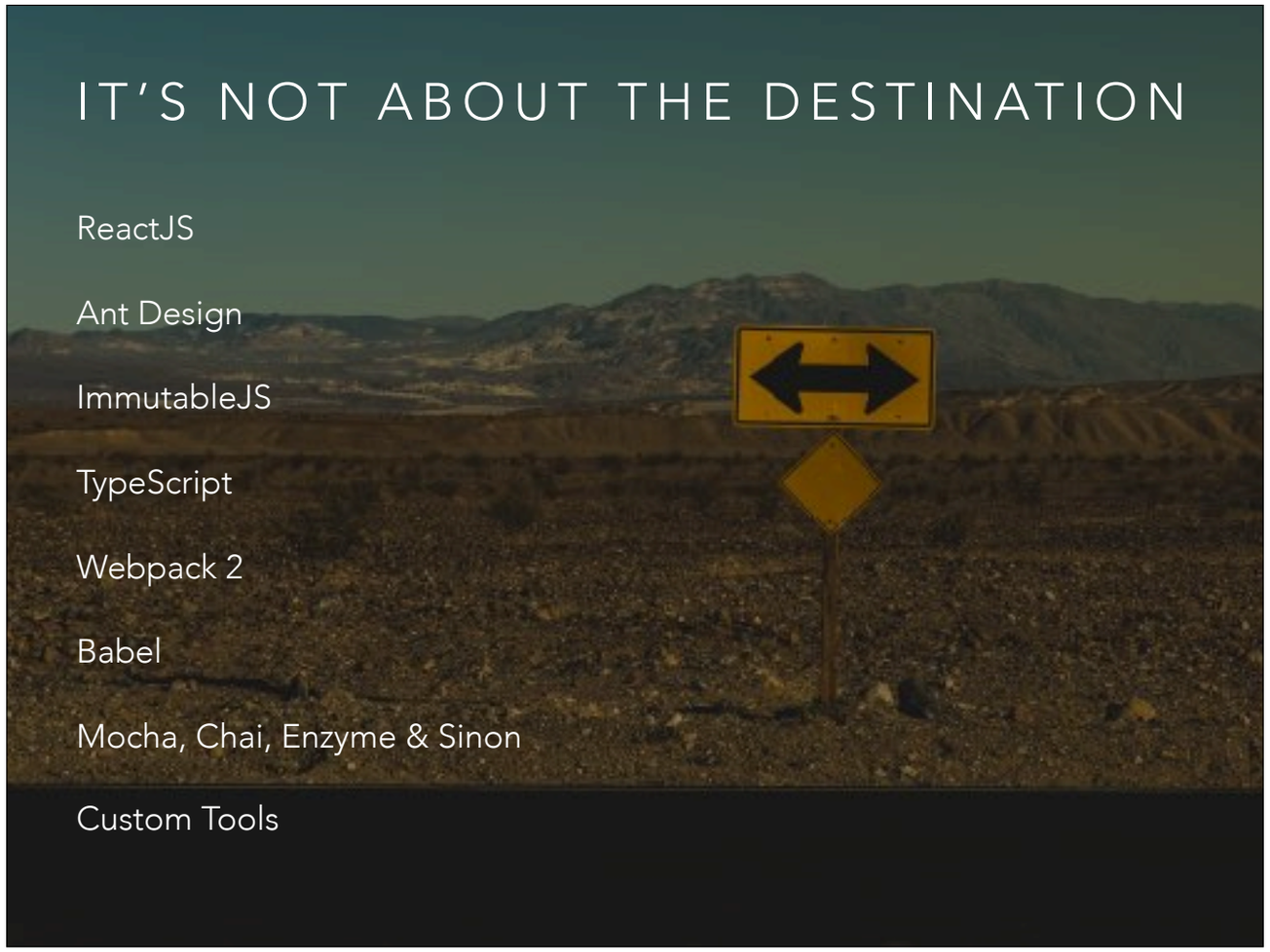
Decisions were made without documentation:
Opinions were unknown to many of the new developers, rational was lost

10:00:00
Before starting a journey, identify what a successful journey looks like.

What problems are we trying to solve?:
change should be driven by improvement.
How do we identify issues we need to address.
What is the time commitment to maintain our current way of doing something vs a different way of doing something
- consider both up-front change [investment] and later time saved [return]

What's working that we want to retain?:
Change for change sake is counter-productive. Before making decisions on change, make certain that the cost of change is worth the return on your time investment.
statements like "I don't like how that's done" are not valid enough reasons?
Just because one way of doing something is different than what you may be use to, does not make it 'bad' or 'wrong'

When and how do we make changes?:
We didn't want to start an aimless journey
All change must be intentional and directed (they must address an issue)
Every change needs to progress the journey forward
Every change needs to be in the same direction (tech debt)
Detours should be researched and validated (agile spikes adding to tech debt to reduce un-intended consequences)



13:30:00
This talk is less about convincing you of the right tech stack and more about the process we went through to figure out that this is the best tech stack for us today.

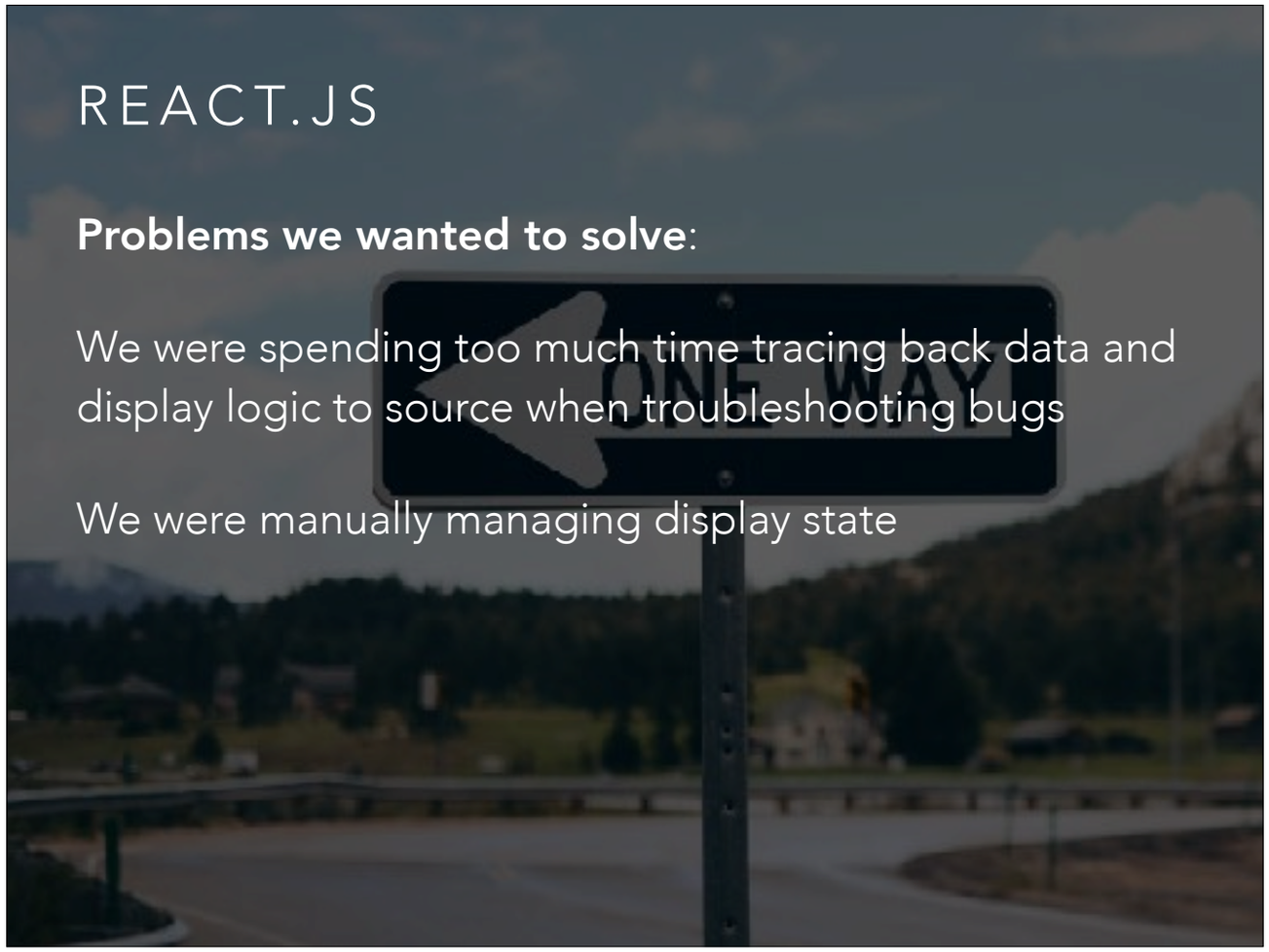
When i say “It’s not about the destination” i mean
The destination is the product and the project should be product agnostic.
As a web dev team, our products are web applications used to interact with our services.
Our project is the technology used to deliver that product.

When making decisions we keep these points in mind:
Technology stack must be flexible to handle any application (destination) that we may be asked to develop

Applications must be able to be supported by posterity (use well documented, well supported tech stack)

Applications must be testable (testing tools are listed last, but that does not mean they are least important)

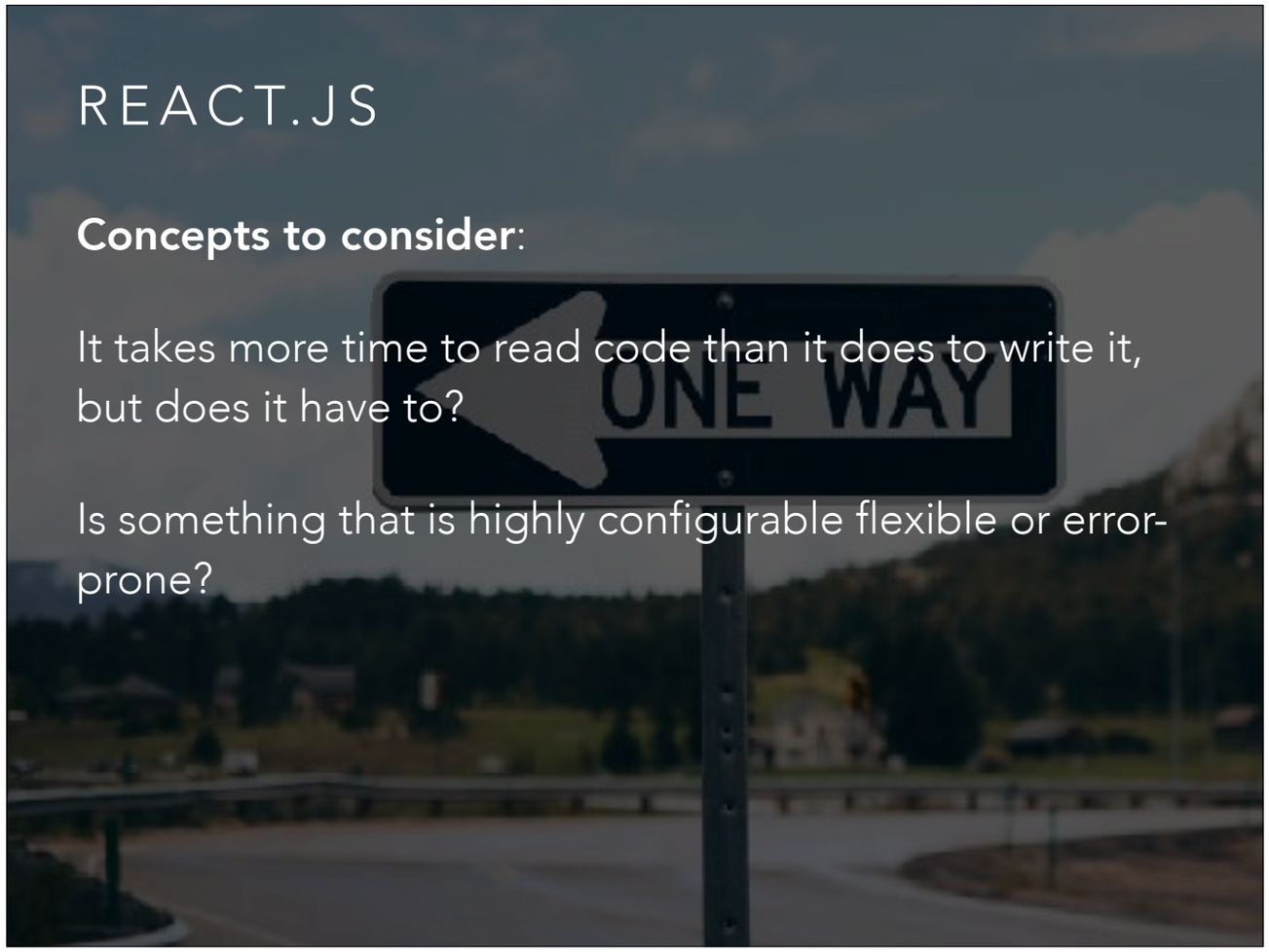
Applications must be able to evolve with the web. i.e. We should be able to swap out any component without a complete project rewrite. (not to say our older apps won’t need a complete rewrite depending on the swapped component)



14:30:00
One way sign: one way data binding for components

We were spending to much time tracing back data and display logic to source when troubleshooting bugs:
explicit actions are component props which also makes troubleshooting event triggers easier

We were manually managing display state:
display state was done with outdated best practices.
too much reliance on jQuery UI and jQuery selectors.
Re-used jQuery objects were updated inadvertently when selectors were too broad



16:00:00

Concepts to consider when choosing an application framework/library.

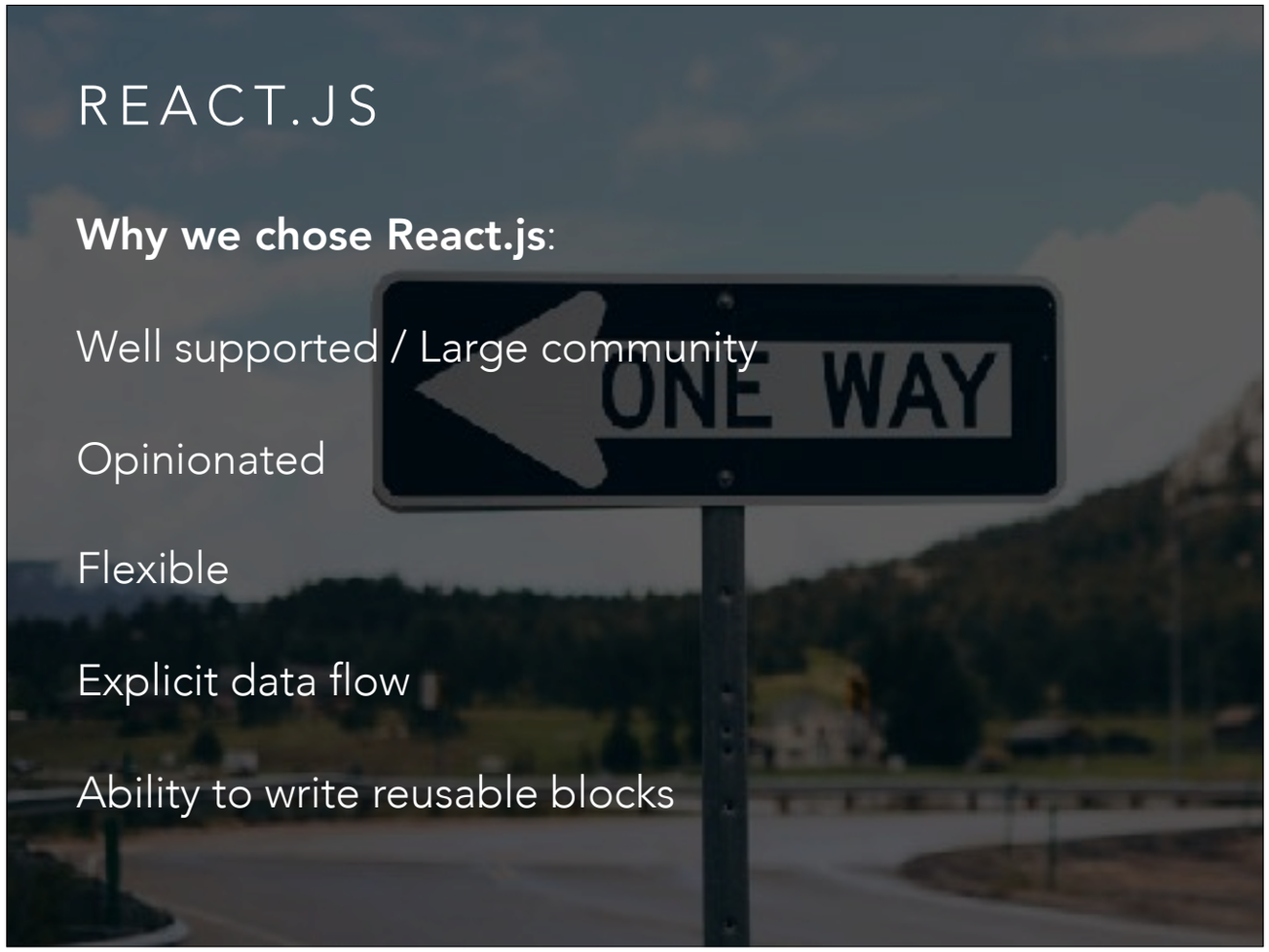
It takes more time to read code than it does to write it, but does it have to?
Can we find a library with a simple and consistent structure so each part of our code follows a similar syntax?
Can we find a library that allows us to more easily write every new feature for someone who will need to work on the code having never seen it.
Can we find a library that allows us to easily separate display from logic.

Is something that is highly configurable flexible or error-prone?
Is there a library that is extensible that isn't necessarily configurable
If you've written nested switch statements or a complex if/else chain, you're probably doing something wrong
Write smaller, more explicit blocks rather than larger, all encompassing components
Higher Order Components were a big benefit with React

20:00:00

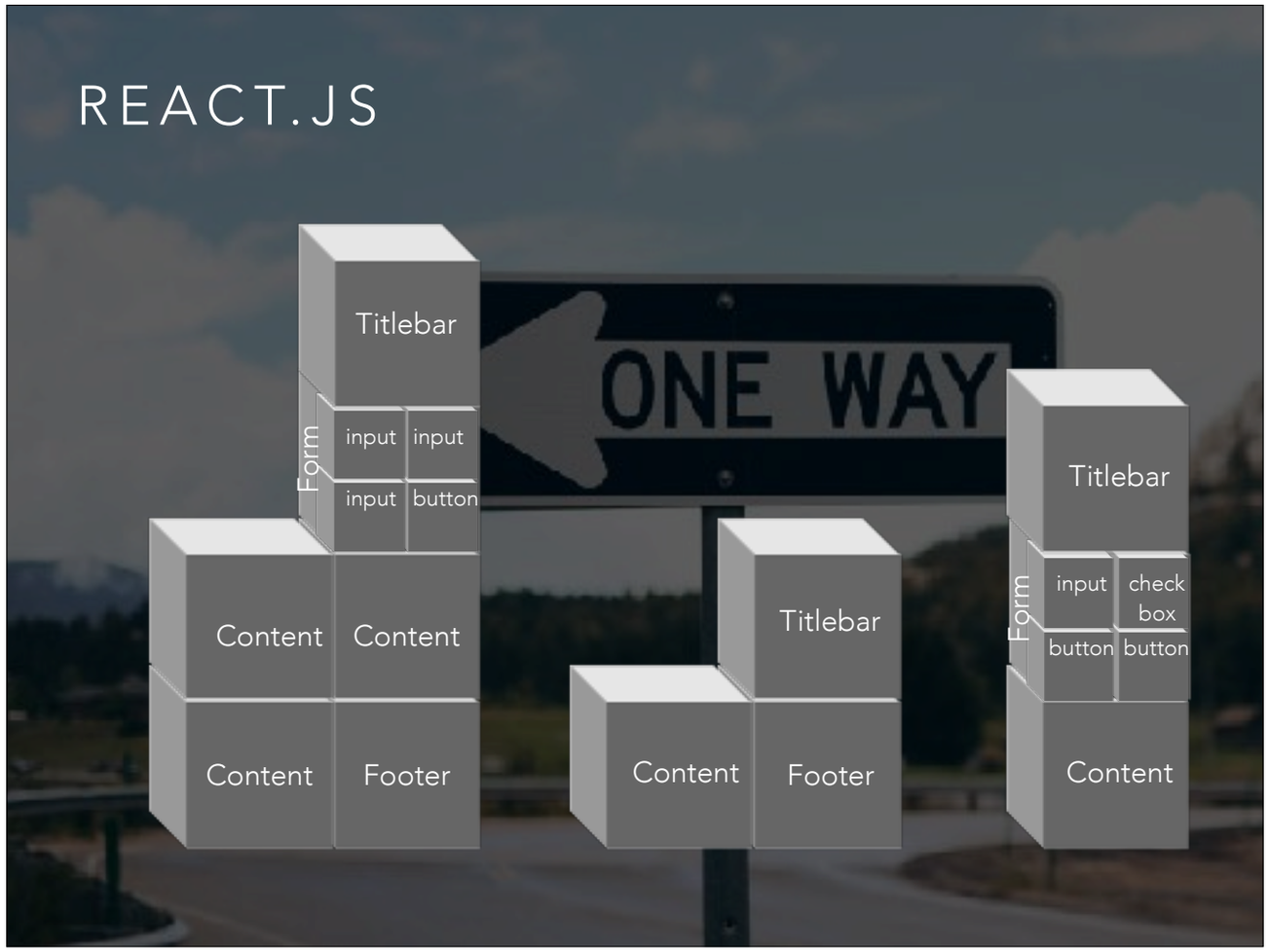
What we looked at:
Library vs Framework. Did we want a “does everything” framework?
We didn’t look at too much. React was a clear winner for us. Not going in too deep. The internet is littered with comparisons

Who here is familiar with ReactJS?



20:30:00

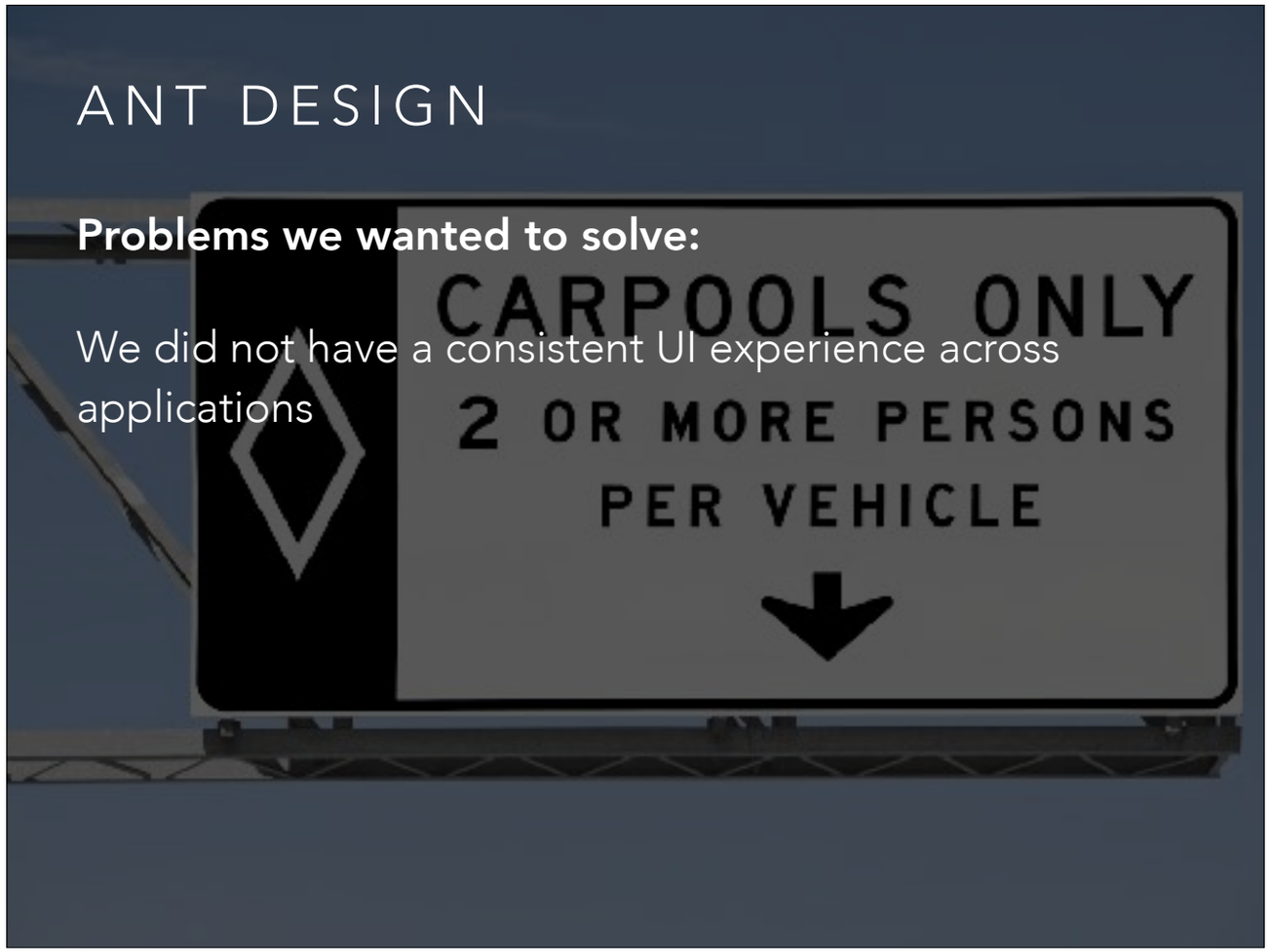
Why we chose what we did:
well supported - lots of resources, tutorials, etc
large community - helpful resources are available, stack overflow, meetups, lots of free tutorials, etc
fairly opinionated - best practices are clearly defined on the support page, no need for exploration. Makes code more maintainable and easier to read by posterity.
flexible - because it is minimal, it can be used to build anything we could need
explicit data flow - much easier to troubleshoot data or settings related bugs. even more explicit with Redux
ability to write reusable blocks - eventually building applications is simply combining blocks, makes for much faster app development (see graphic on next slide)



22:30:00

Reused:
Titlebar
Form (Higher Order Component) - we use Ant Design form (later slides)
Form elements (independent of form)
Content component can possibly be reused as HOC (like Ant Design Layout)
Footer

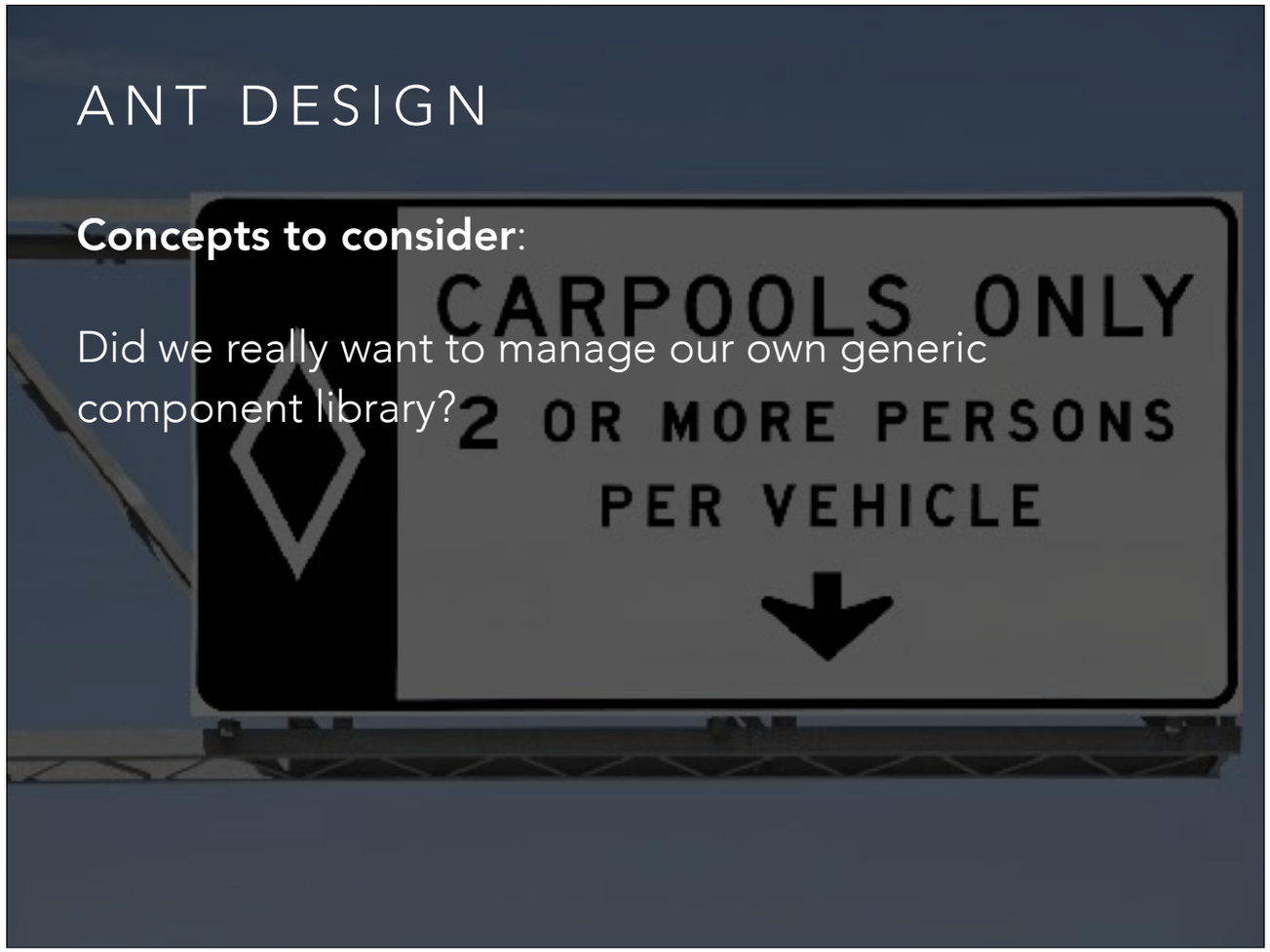
Development time decreased dramatically.
With React, we can write the code for an alpha of a web app in a single sprint.
Iterative development with UX and our Product owners is also much easier and faster. Adjustments are often done in real-time.



23:30:00
Who here as heard of Ant Design?
alibaba project
wraps another project, react-components, in design elements

Topic:
Carpools Only: Restrict the UI lane to disallow “anything goes” in terms of UX. We want to stay in the fast lane. The fast lane has restrictions.

We did not have a consistent UI experience across applications:



24:00:00

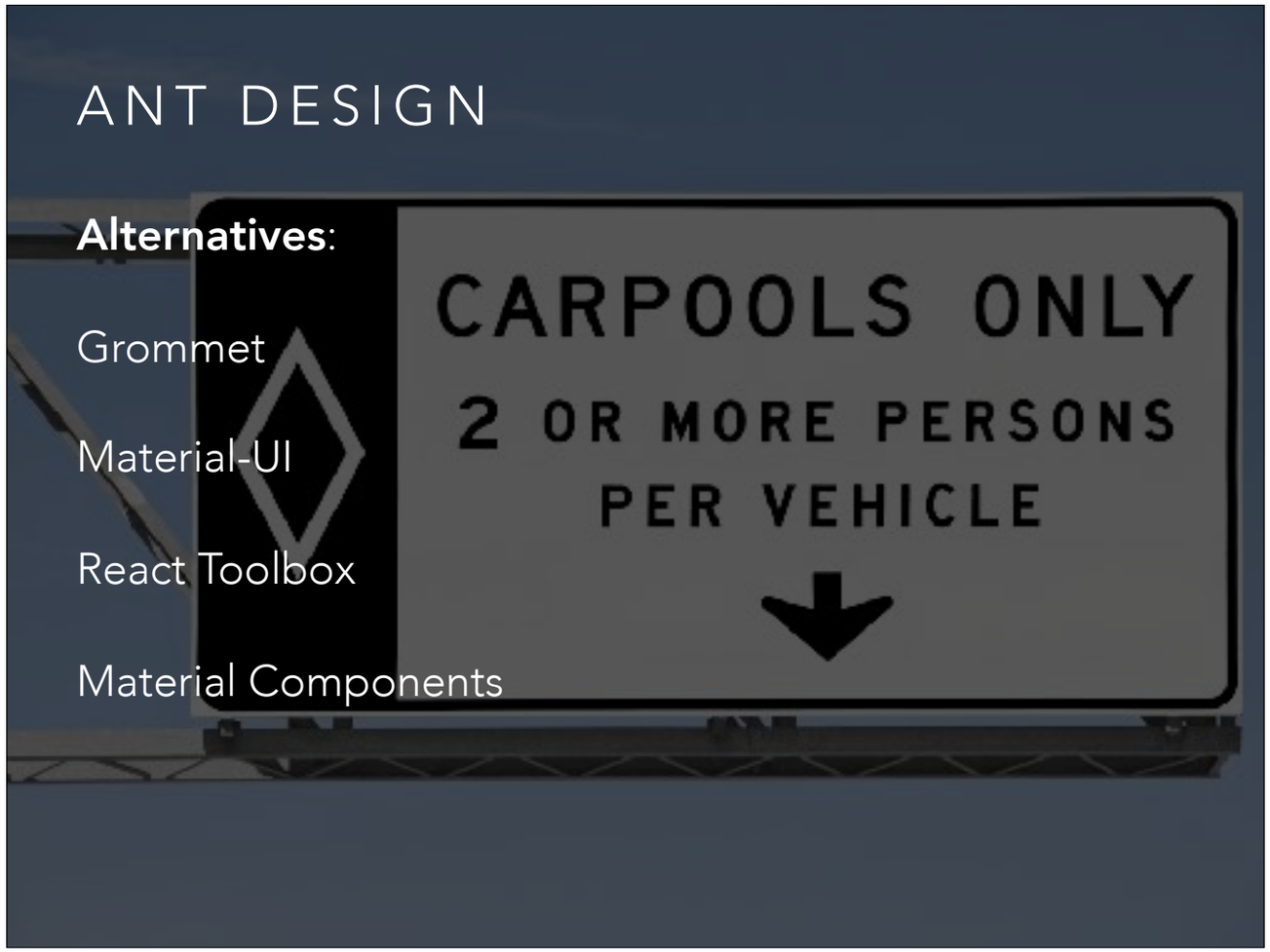
Concepts to consider:
Did we really want to manage our own generic component library?

What is the value that our company is creating? Is it unique user interface components? (buttons, input fields, date selectors, and those types of things)

This doesn’t mean we ignore User Experience, our UX team cares greatly about usability.

Our focus is not to re-invent web controls.
(Who remembers the web of the late 90s? https://www.warnerbros.com/archive/spacejam/movie/jam.htm)

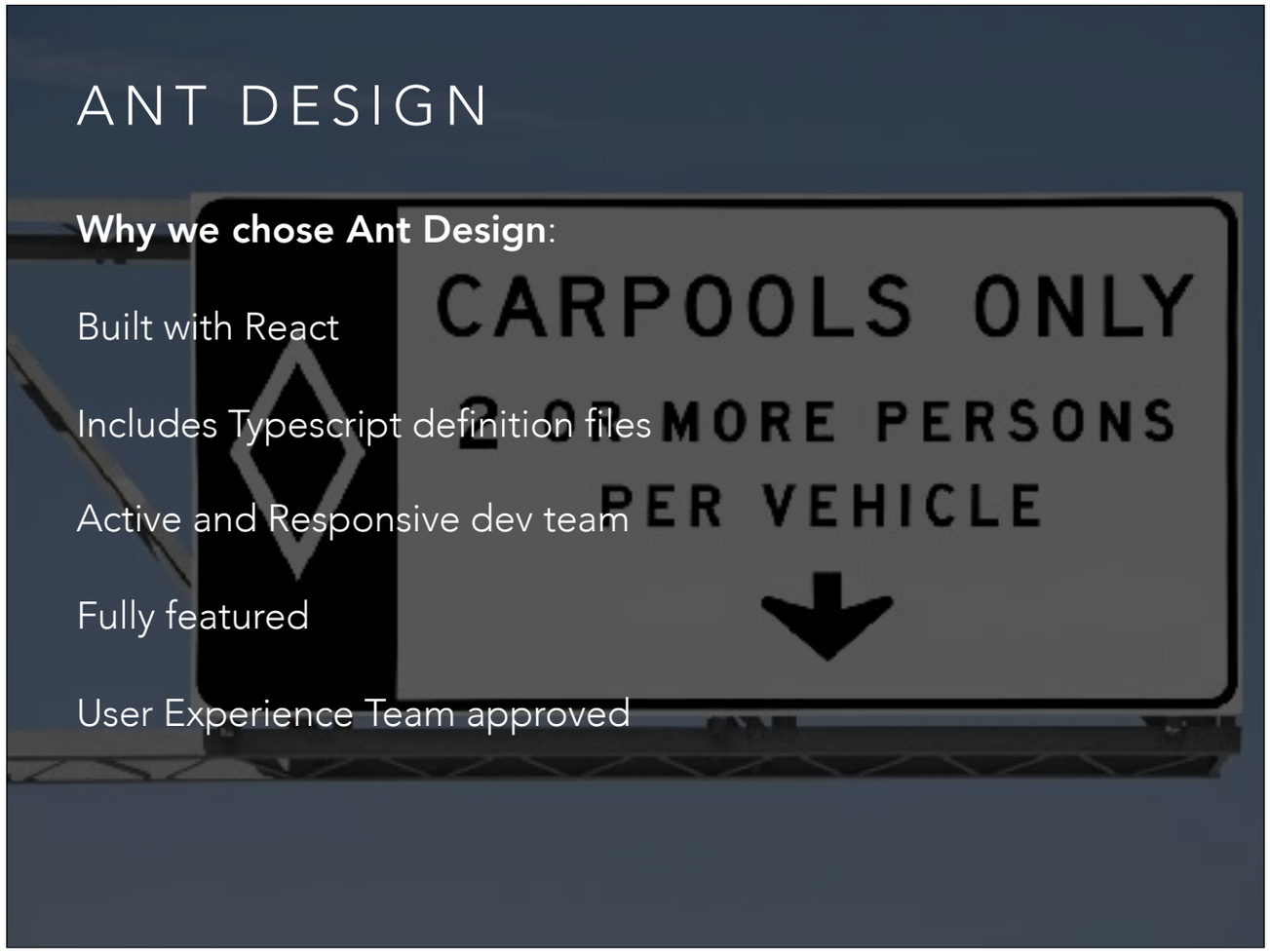
Once we realize we don’t want to have our own library,
Are the libraries that fit into our development stack better than others?



26:30:00

What we looked at:

For any third party library, appropriate license was a necessity. Prefer MIT or Apache 2.0
Was library designed for desktop or mobile first? For our apps, desktop first.
Can we customize when needed?
Can we extend?
Can custom components intermingle?
How active is the project? is it well supported/maintained?
Is library development team open to PR's? (we really don't want to maintain a fork)



27:30:00

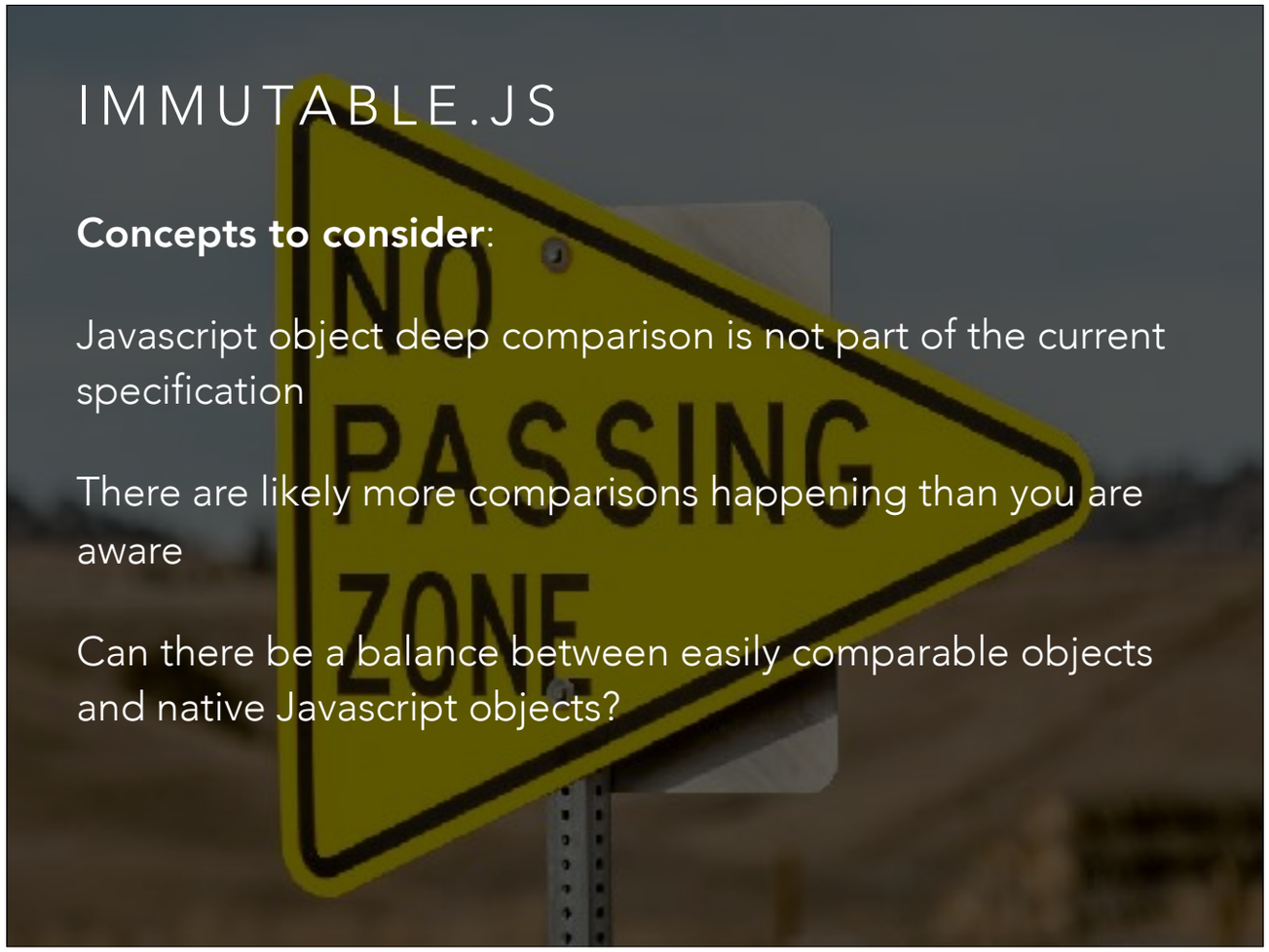
Why we chose what we did:
React components fits into our development stack.
Full typescript definitions.
Very active project.
Library development team very receptive to PR’s, usually reviewed and merged within 2 days.
Components are very customizable, both visually with themes and functionally with event handling.
Fully featured library.
UX team approved!

28:00:00

No passing zone: don’t change lanes if you don’t have to

redraws occurred after equality checks return false, if there was even an equality check to begin with.

Ask familiarity with techniques of Immutability in Javascript (go to next slide)



30:00:00

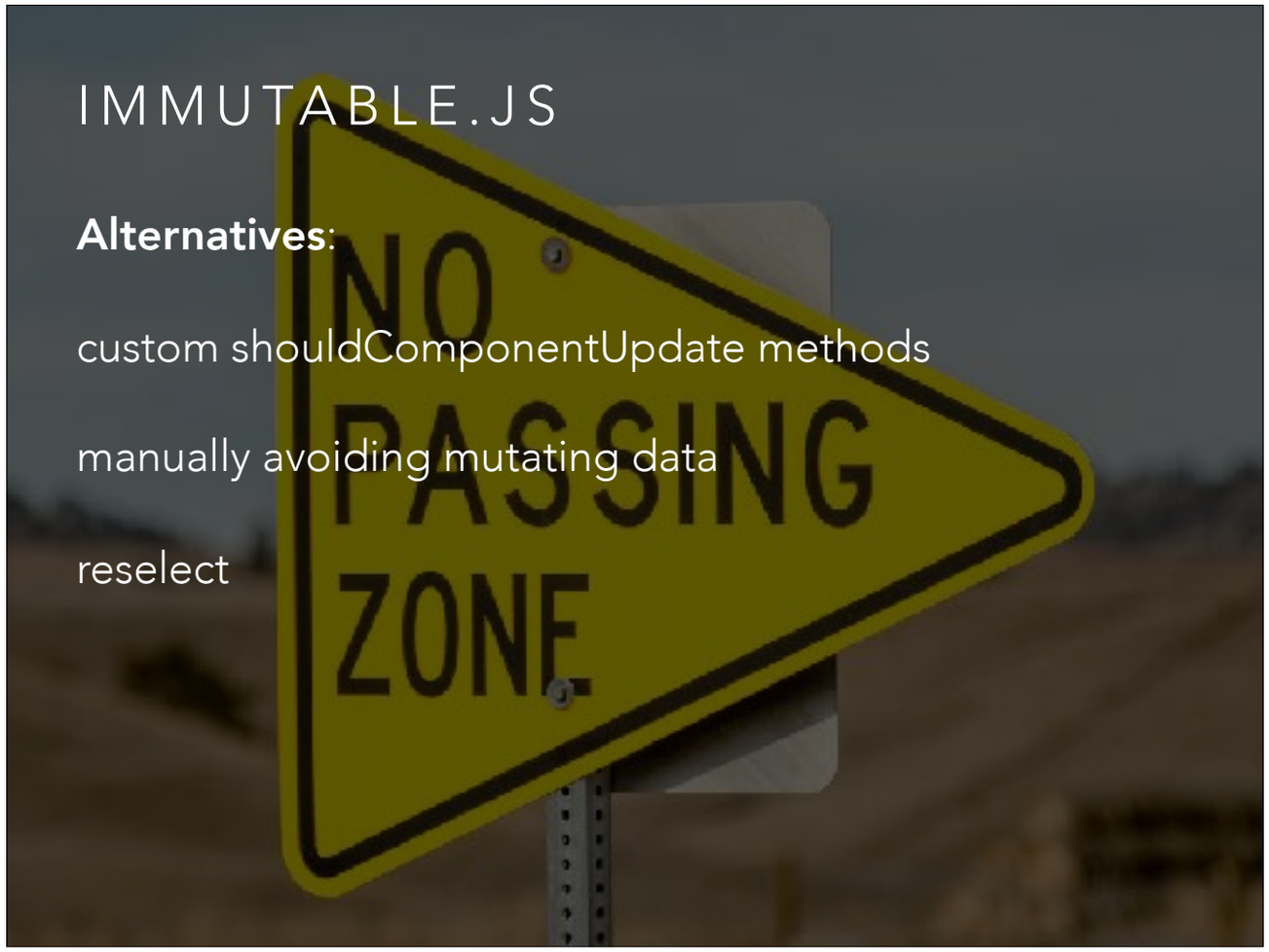
Concepts to consider:
Javascript object deep comparison is not part of the current specification
There are likely more comparisons happening than you are aware

https://github.com/garbles/why-did-you-update (react specific)

Our decision on what was best for us right now was deeply linked to another decision about React and Redux

Is the conversion to an easily comparable object just as expensive as a custom deep comparison between native objects?
Immutable toJS and fromJS are very time consuming. Map/toObject, List/toArray much more efficient, doesn't do deep conversion

Do I have to go 'all in' with a comparable object solution?



31:00:00
What we looked at:

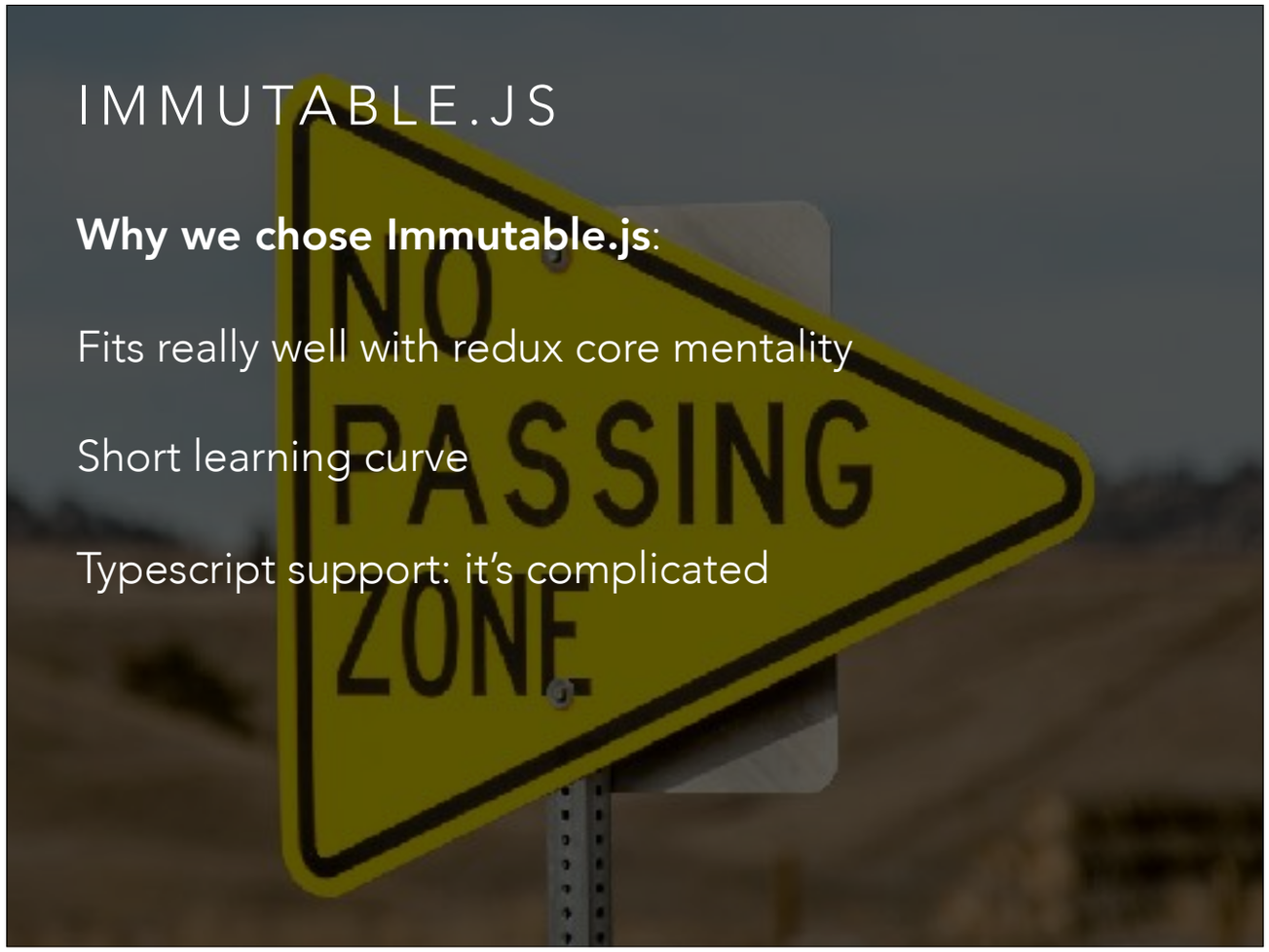
custom shouldComponentUpdate methods: too much room for error

manually avoiding mutating data: Object.assign(), spread operators

reselect: works great for derived data, most of our data is not derived. Much of that load is handled by api server

integration with redux and moving away from flux

We looked for what gave us the most benefit for the least overhead.

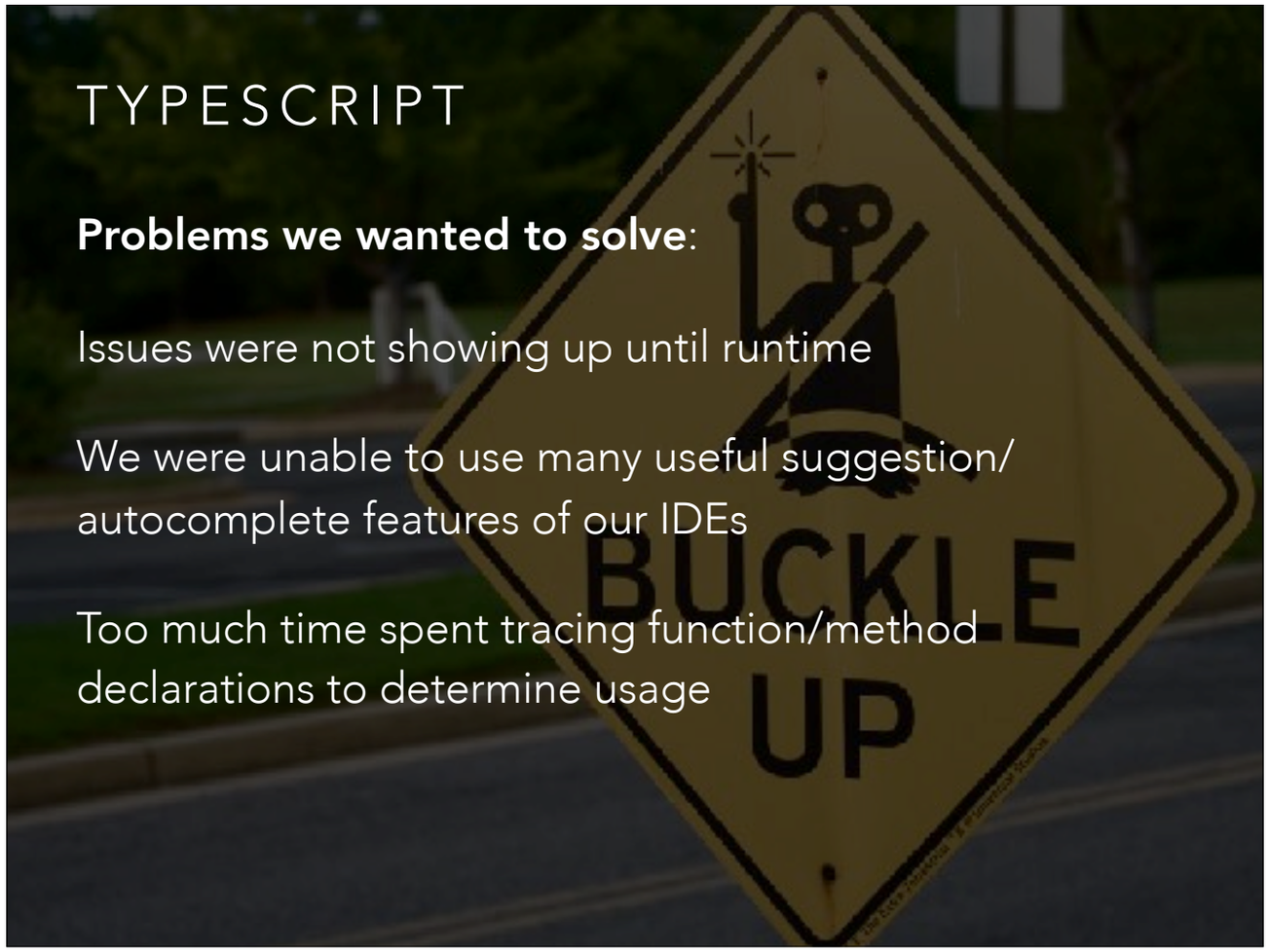


32:00:00
Why we chose what we did:

redux core mentality: reducer is ‘pure’ function. State is always replaced, never modified.

Short learning curve. Helps solve same troubleshooting issue as React. (just don’t rely on toJS() and fromJS())

downside: Custom typescript definitions for every type of Map, no inference. We needed to write a generic interface for Map.



33:00:00

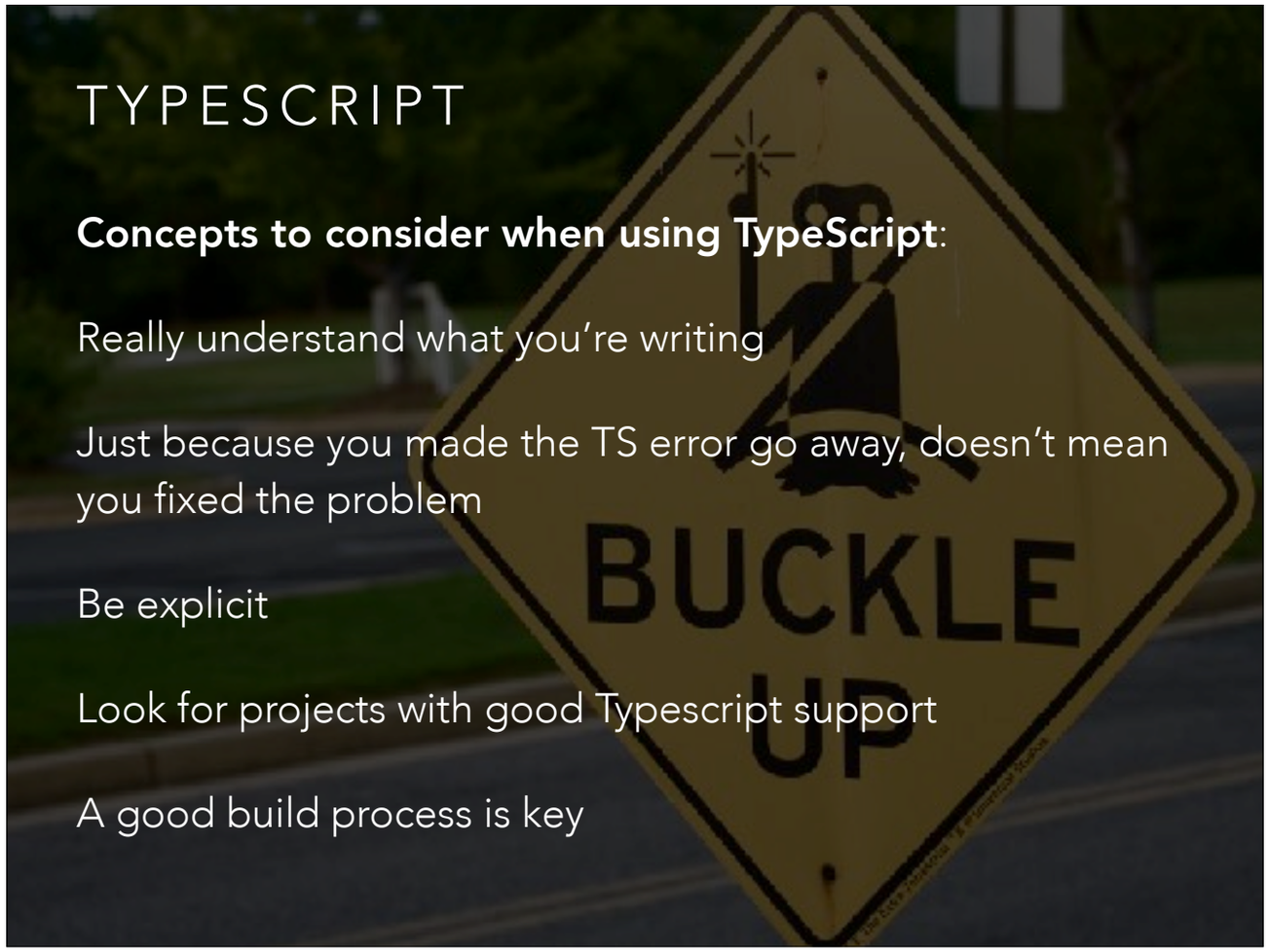
Buckle Up: be safe

Ask who writes web apps in language other than vanilla javascript? ask how uses TypeScript?

There are a LOT of languages that compile to javascript (wiki page on coffee script GitHub page). MANY have come and gone. Delay between new JS features and transpilation support.

How likely is language to stay around?
Backed by Microsoft (large corporate backed languages tend to stay)
can it co-exist with current code?
Can be mixed and matched with Javascript, great for slowly migrating apps to TypeScript
is it difficult to learn?
Syntax is familiar to Javascript devs

how good is the IDE support?
Visual Studio Code - free cross-platform IDE that integrates very well with TypeScript

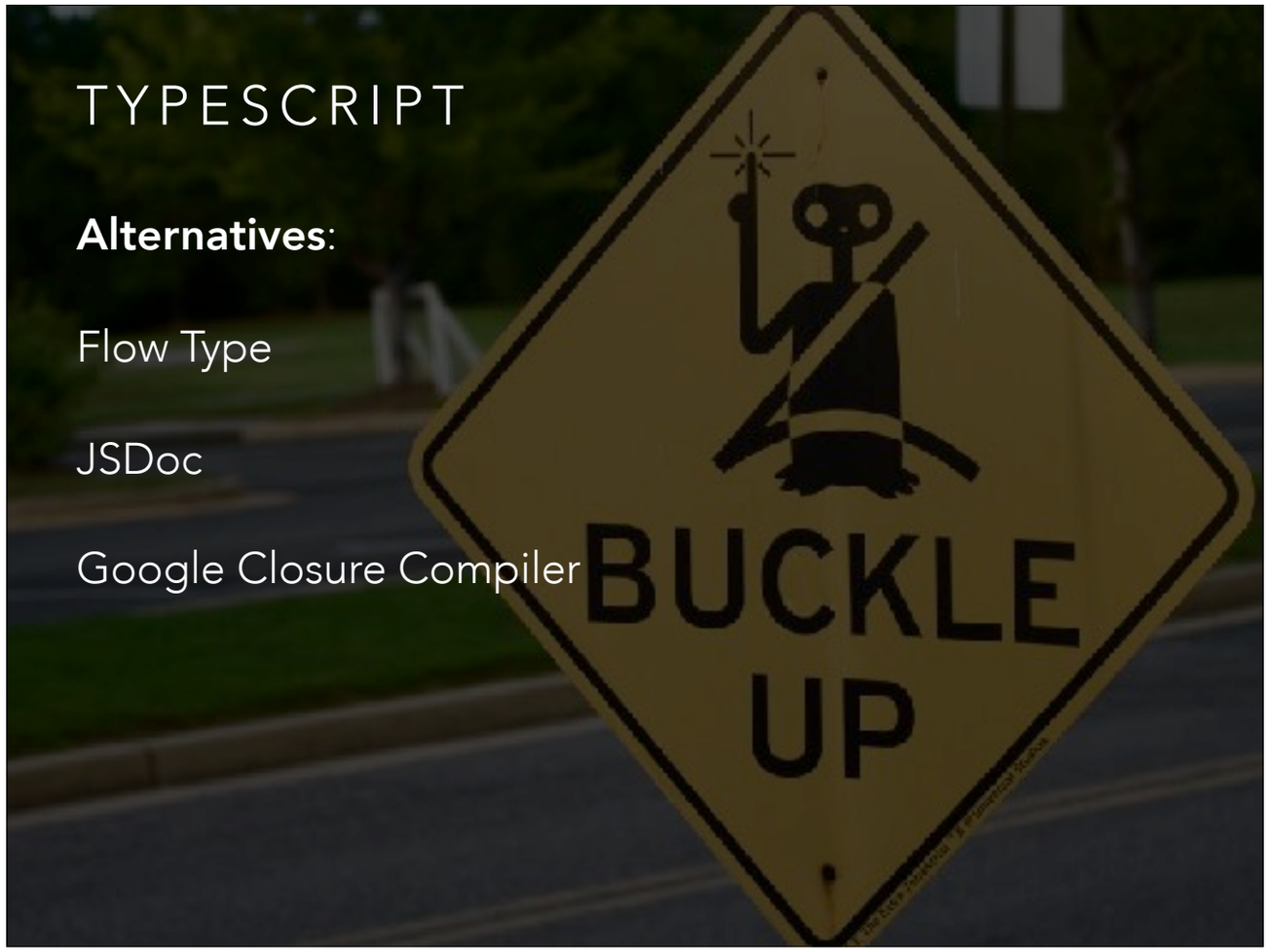


35:30:00
Concepts to consider (when working with TypeScript - not about choosing a language):
A typed language is not a Silver Bullet

understand what you're writing: Write complete typescript definitions, the more time you spend up front thinking about your definitions, the more time you'll save later
Typescript won't protect you as well if you loosely type everything. Try to be explicit
Typescript still doesn't protect you from mistakes you make when writing your typescript

DefinitelyType @types scoped packages are helpful, but often times are incomplete. Version mismatches between libraries and type definitions can be frustrating.

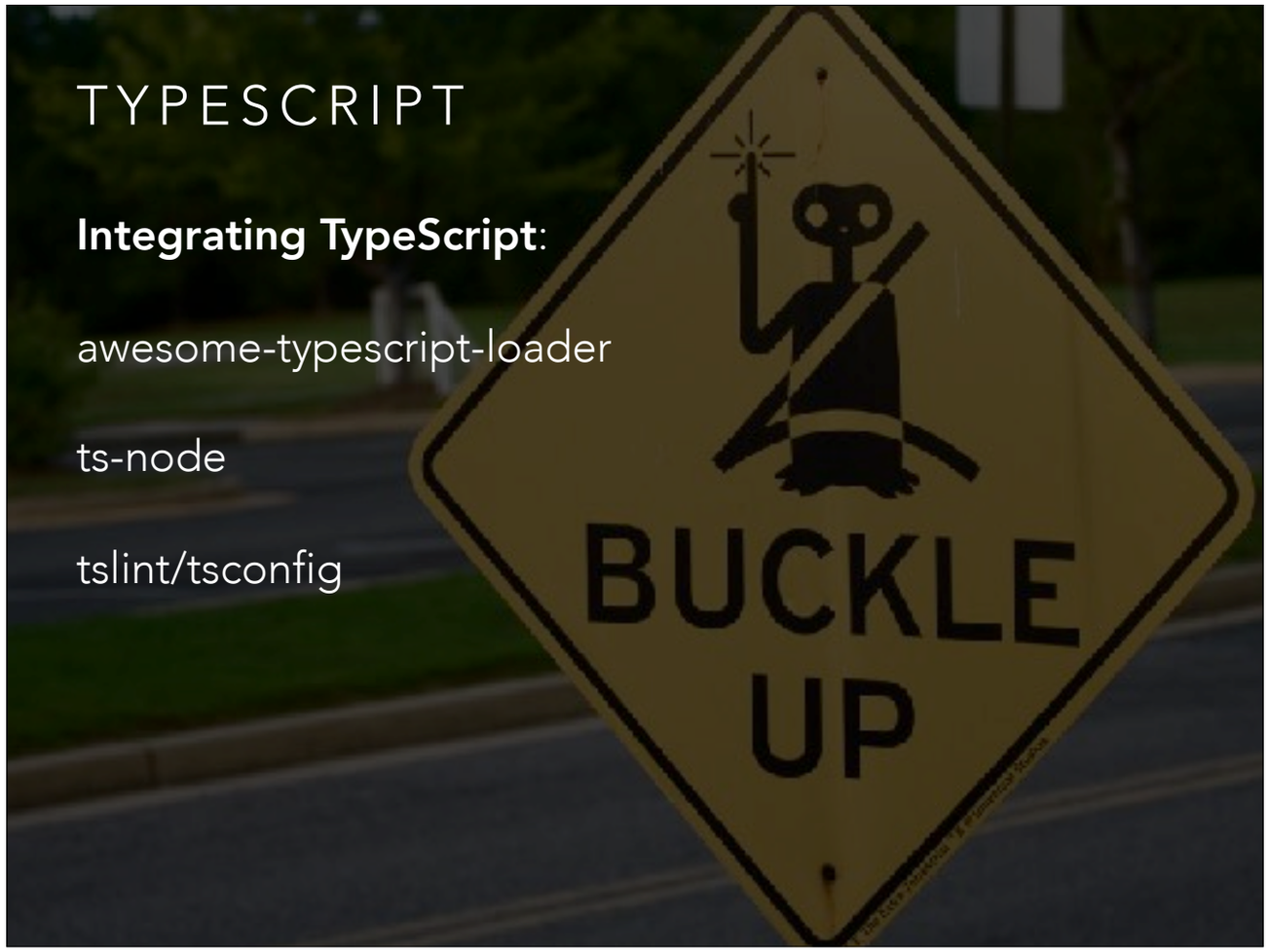
Typescript compile targets and processes can be fully self contained, but often there are trade offs for that. i.e. test watchers, minified code,



36:00:00
Flow type:
more bugs introduced than fixed
relatively new
less activity than typescript

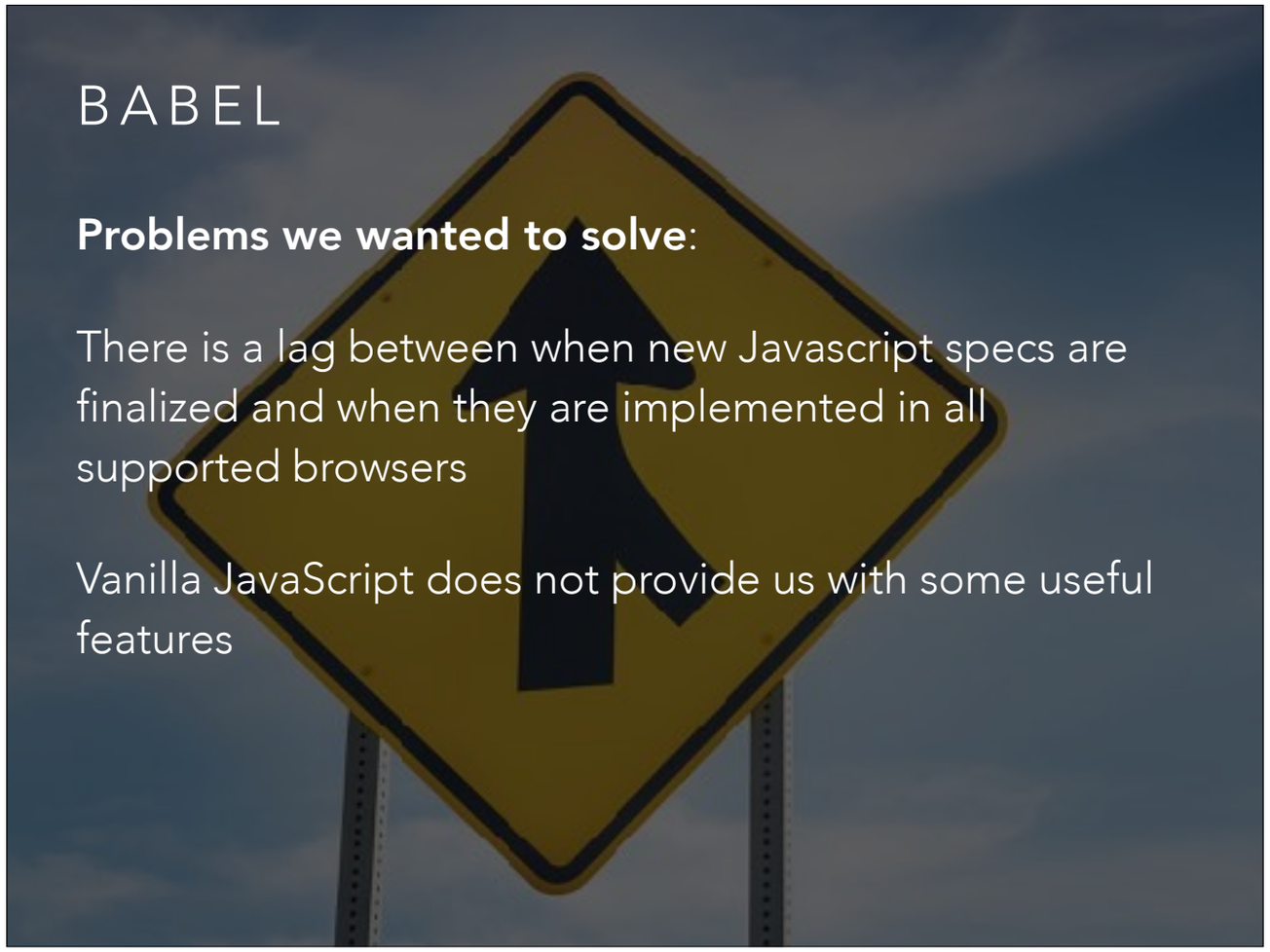
jsdoc:
too much documentation reduces readability

closure:
design decisions are becoming outdated



37:00:00
Integrating Typescript

since TypeScript is a compile to javascript language, how did we get typescript to “run”
awesome-typescript-loader (webpack pluging - later slides)
ts-node (has a register method similar to babel)
tslint/tsconfig (clean code)

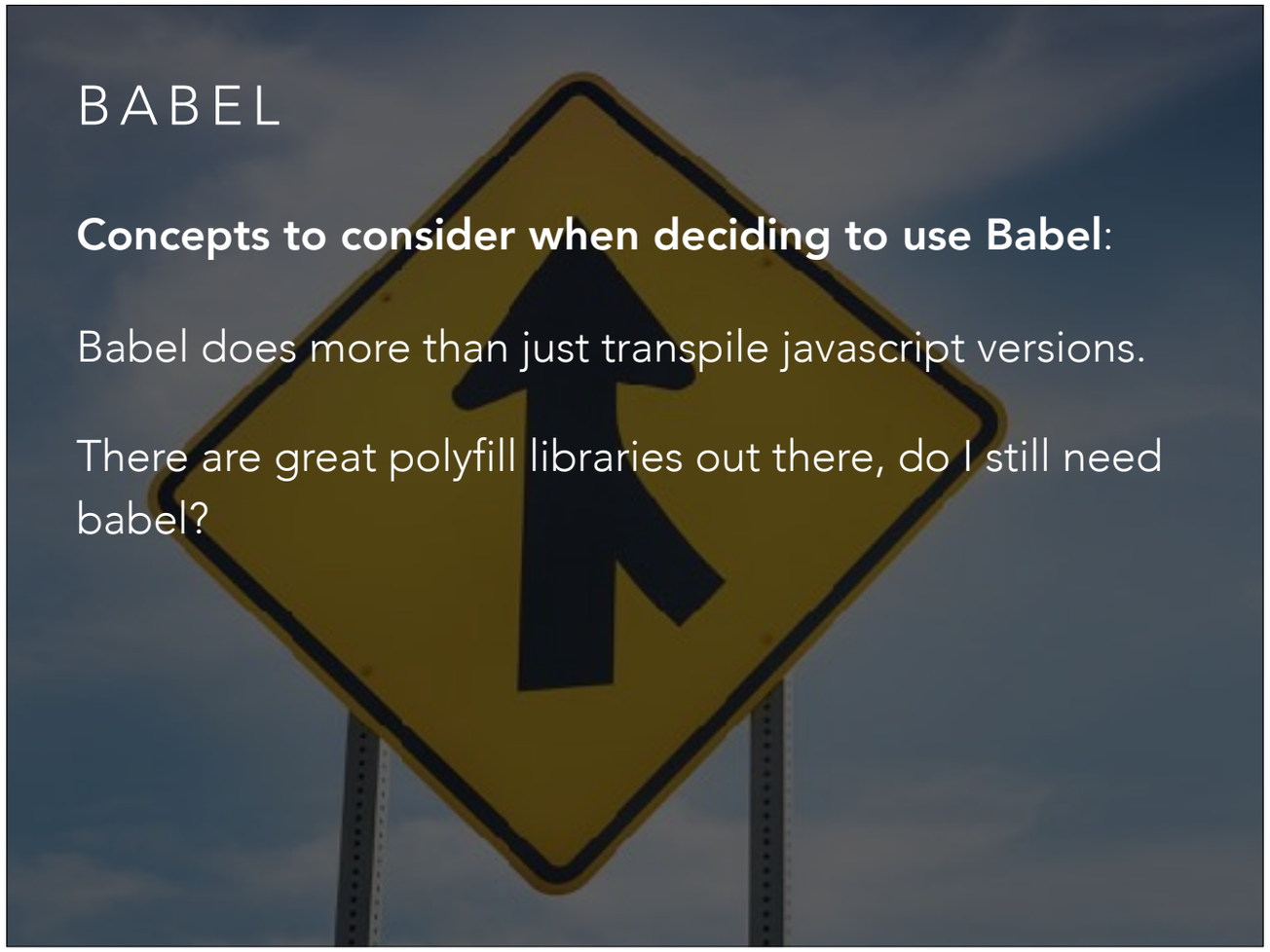


39:00:00

lane merge sign: bringing in new features before all web browsers support them.

There is a lag between when new Javascript specs are finalized and when they are implemented in all supported browsers

Vanilla JavaScript does not provide us with some useful features:
Babel plugin architecture allows us to expand on the language. We use this in our build process (later slides)

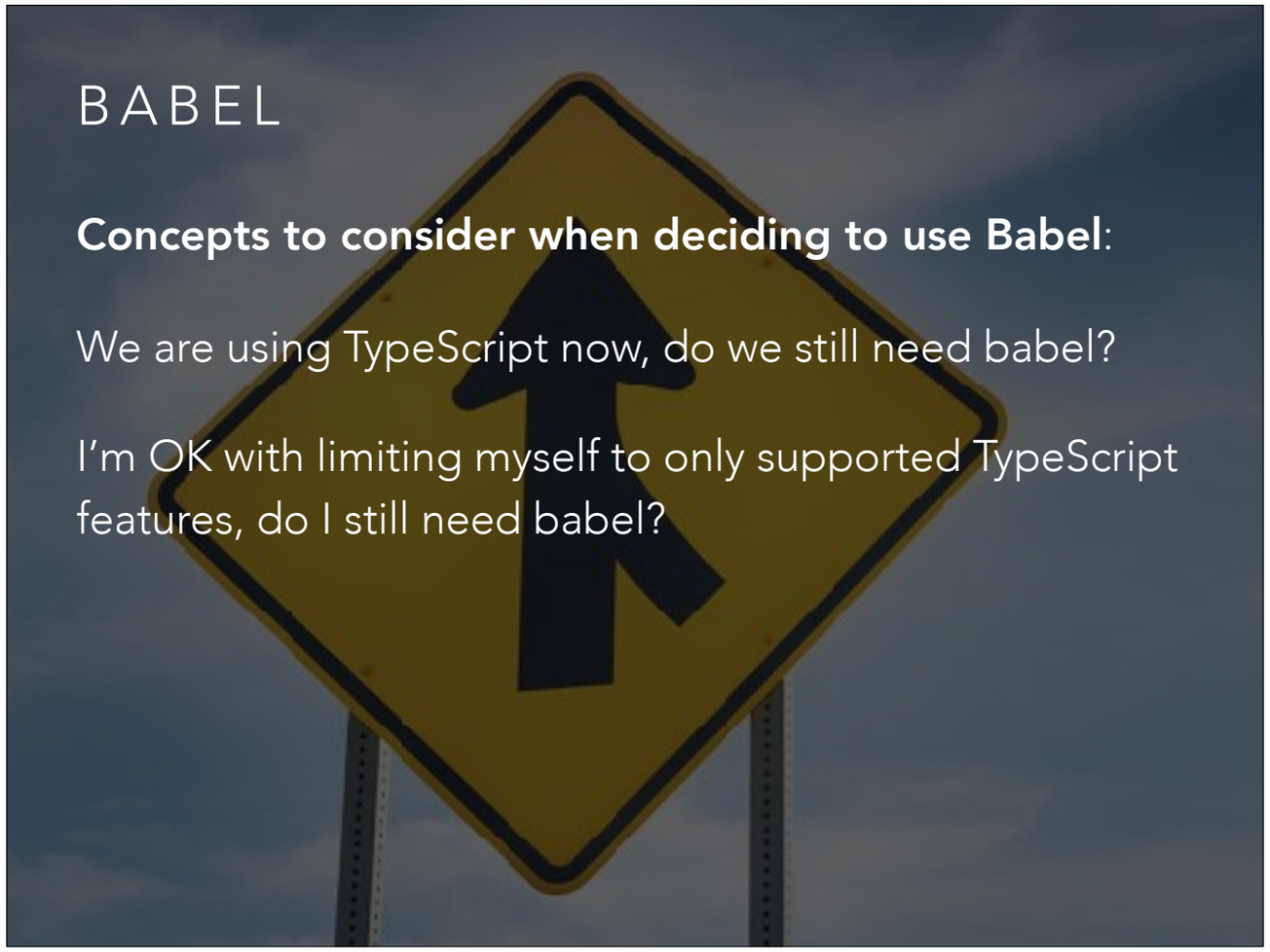


39:30:00

Concepts to consider:

Babel is more than just a new feature transpiler. There are many useful babel plugins

Polyfills work OK, but they still add code.
Focus on the consumer. less code for them is better even if it’s more work for you.
manually Polyfill individual WebAPIs not ES features



40:30:00

Concepts to consider when deciding to use Babel:
We are using TypeScript now, do we still need babel?
We tried to go all Typescript, but abandoned that because we were loosing out on too many nice features

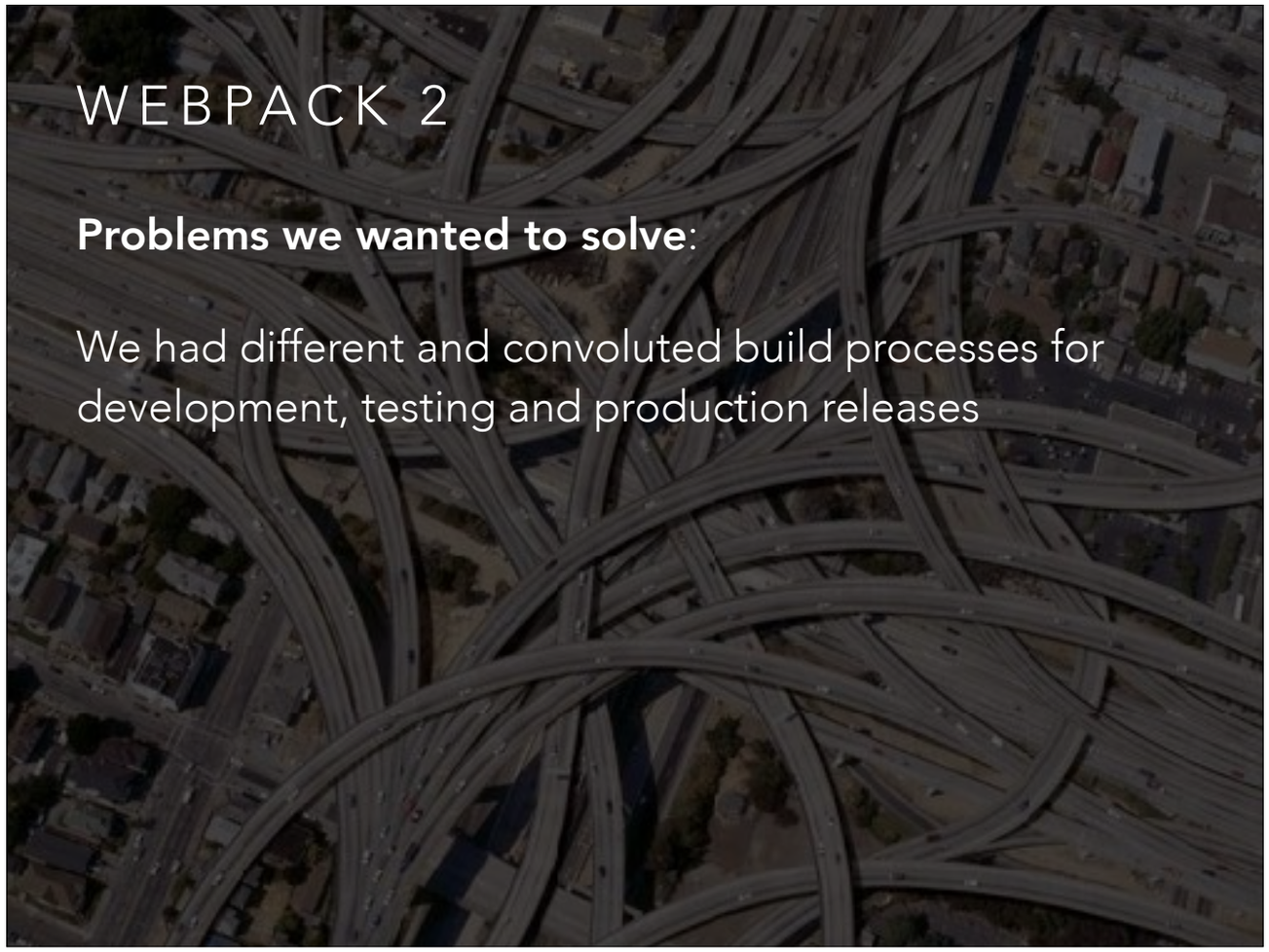
I'm OK with limiting myself to only supported TypeScript features, do I still need babel?
Typescript and babel: TypeScript does not support transpiling ES6 features when targeting ES5. i.e. Object.assign()

41:30:00
Why we chose what we did:

Tree shaking relies on ES6 modules so we could not set our Typescript target to ES5
Minification/Uglify requires ES5 code
Basically our webpack had to accept ES6 code and something had to convert it to ES5 before it got to Uglify

Babel import plugin allows us to include scss/less files in javascript components and it only includes the css for the components we use (useful for ant design). It can also treat some ES5 libraries like ES6 modules (useful for lodash). it GREATLY reduces final build size

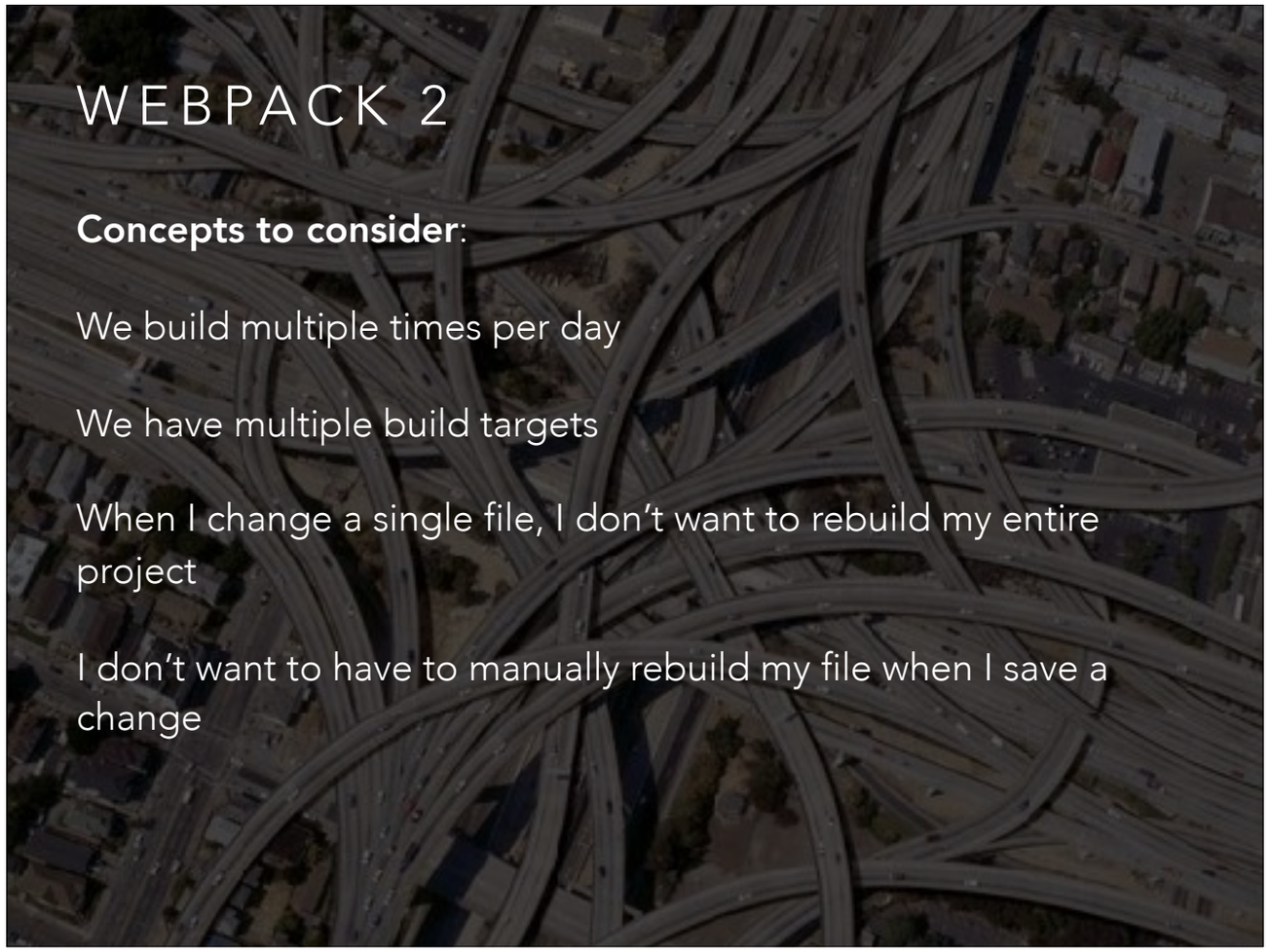
Object.assign() is not part of Typescript. Object spread did not arrive until Typescript 2.1



43:00:00
Intersection: Multiple parts combined to make a whole

our build process with grunt was a multi-minute process. multiplied by the times each day we make changes, that was a huge time sink

Good build processes are critical IMHO. Story: I was overseeing a development intern this past summer. The first few weeks of him being in the office was setting up and understanding all the parts of setting up a good build process. He hated it. Later he saw just how much time he was saving when building features and just how fast he could iterate over different implementations.

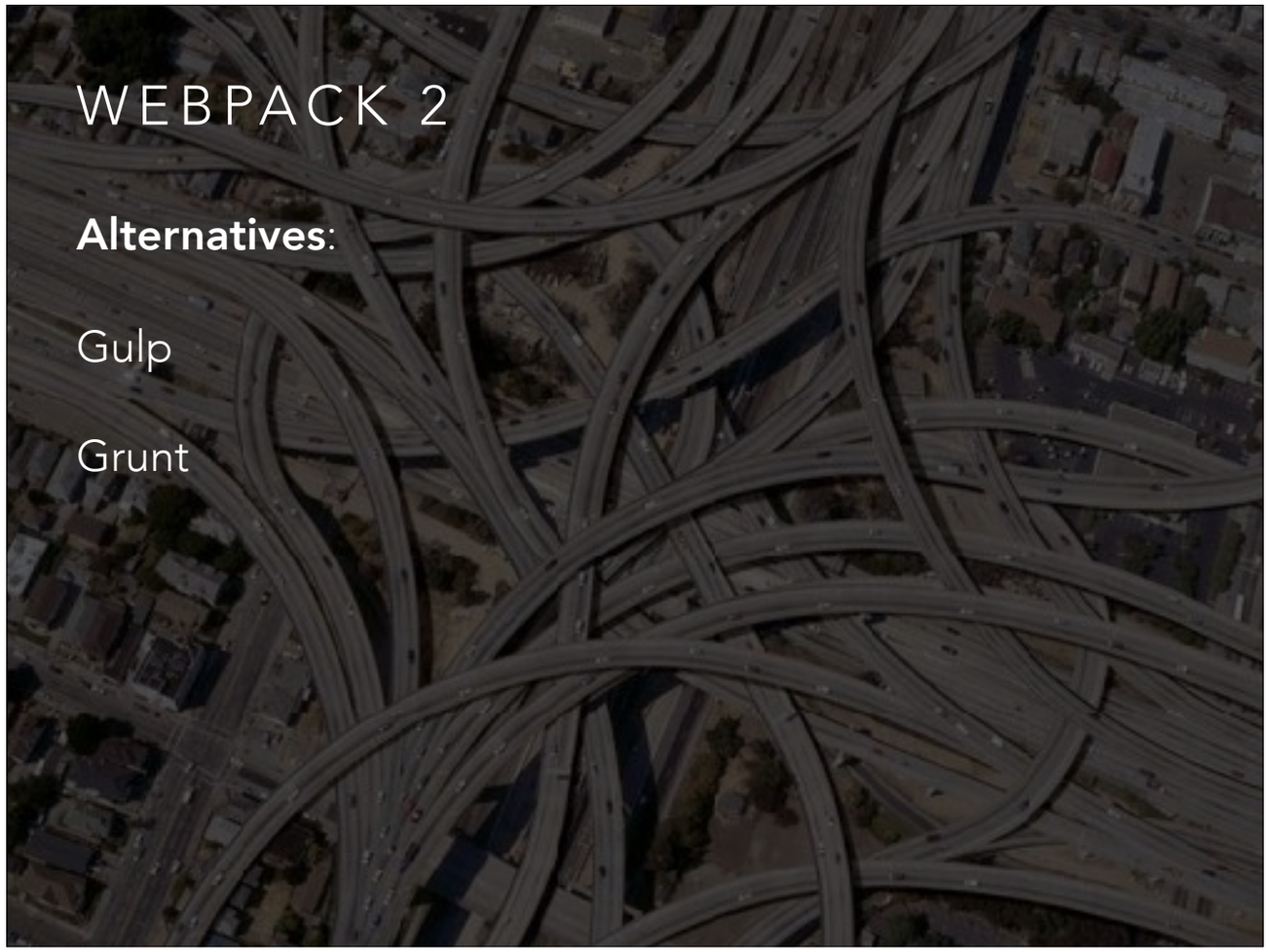


44:00:00
Concepts to consider when looking at your build tools:
We build multiple times per day:
Build tools are not used only right before deployment

We have multiple build targets:
Build tools need to easily be able to switch between dev, test and prod environments

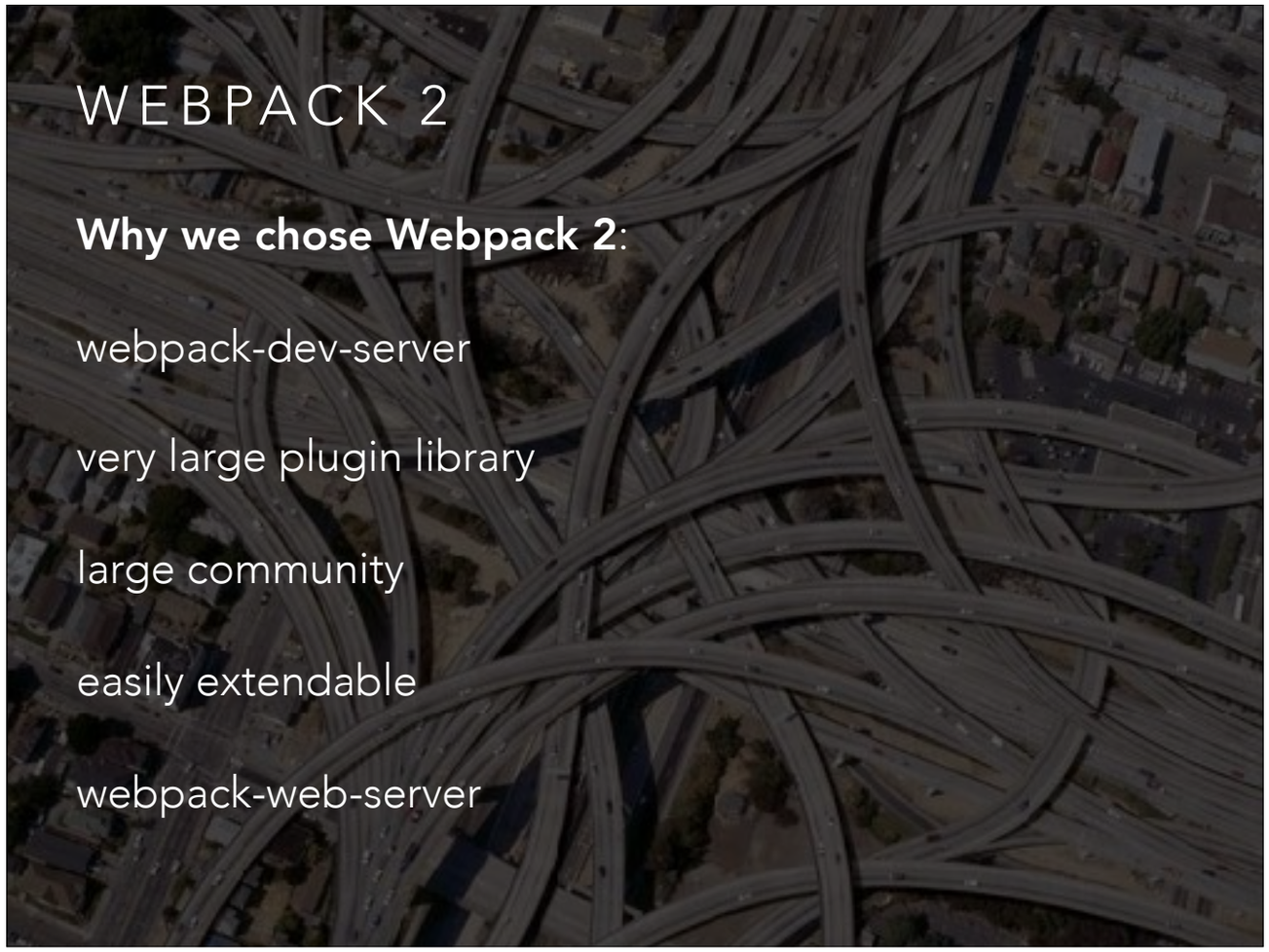
When I change a single file, I don’t want to rebuild my entire project:
our projects can contain hundreds of files, do I need to re-evaluate each one when I make a single change?

I don’t want to have to manually rebuild my file when I save a change:
Build tools should help speed up development



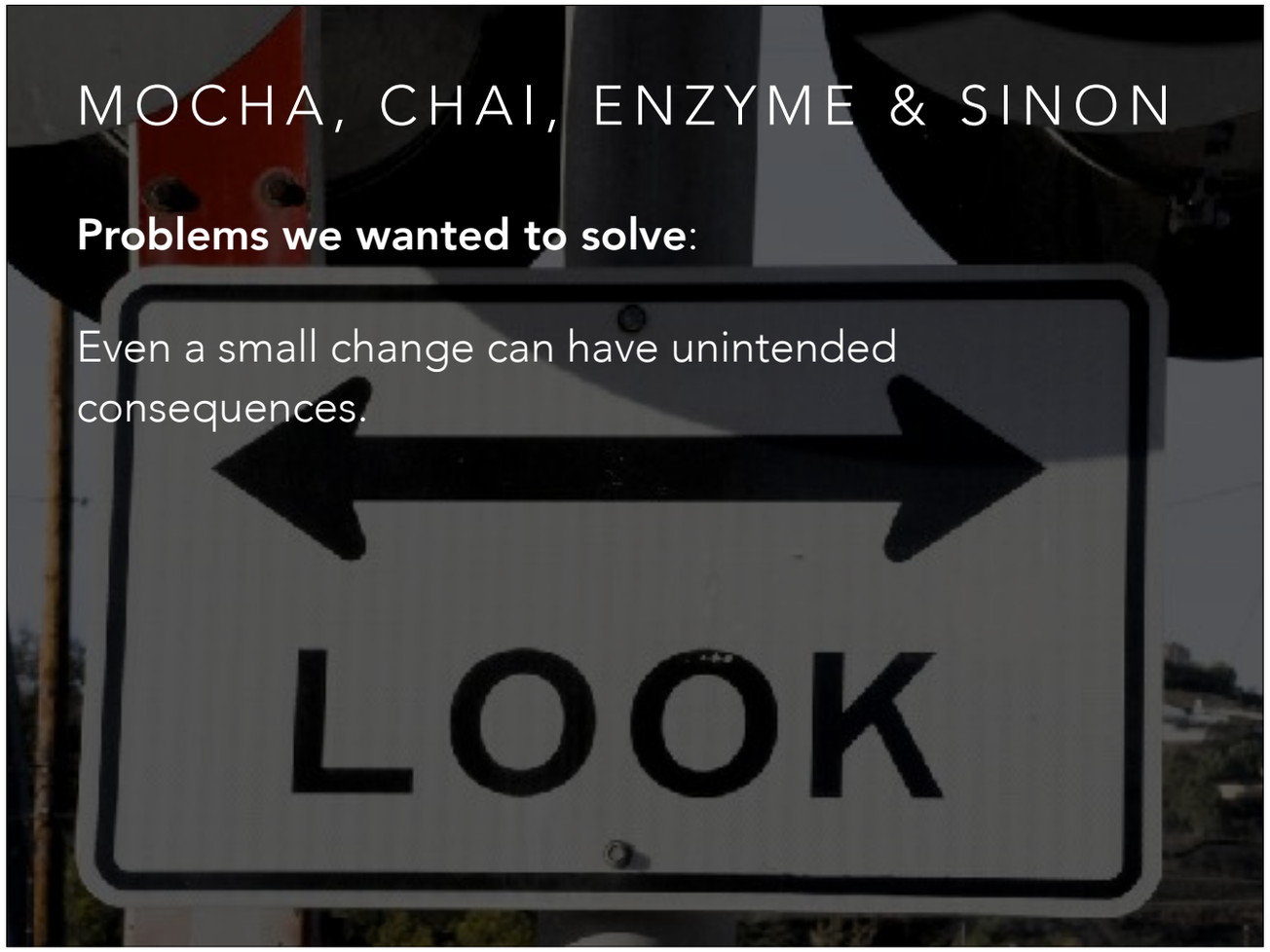
45:00:00
What we looked at:
Not going to spend a lot of time, internet comparisons galore

summary: webpack2 gave us a really nice balance between complexity and functionality

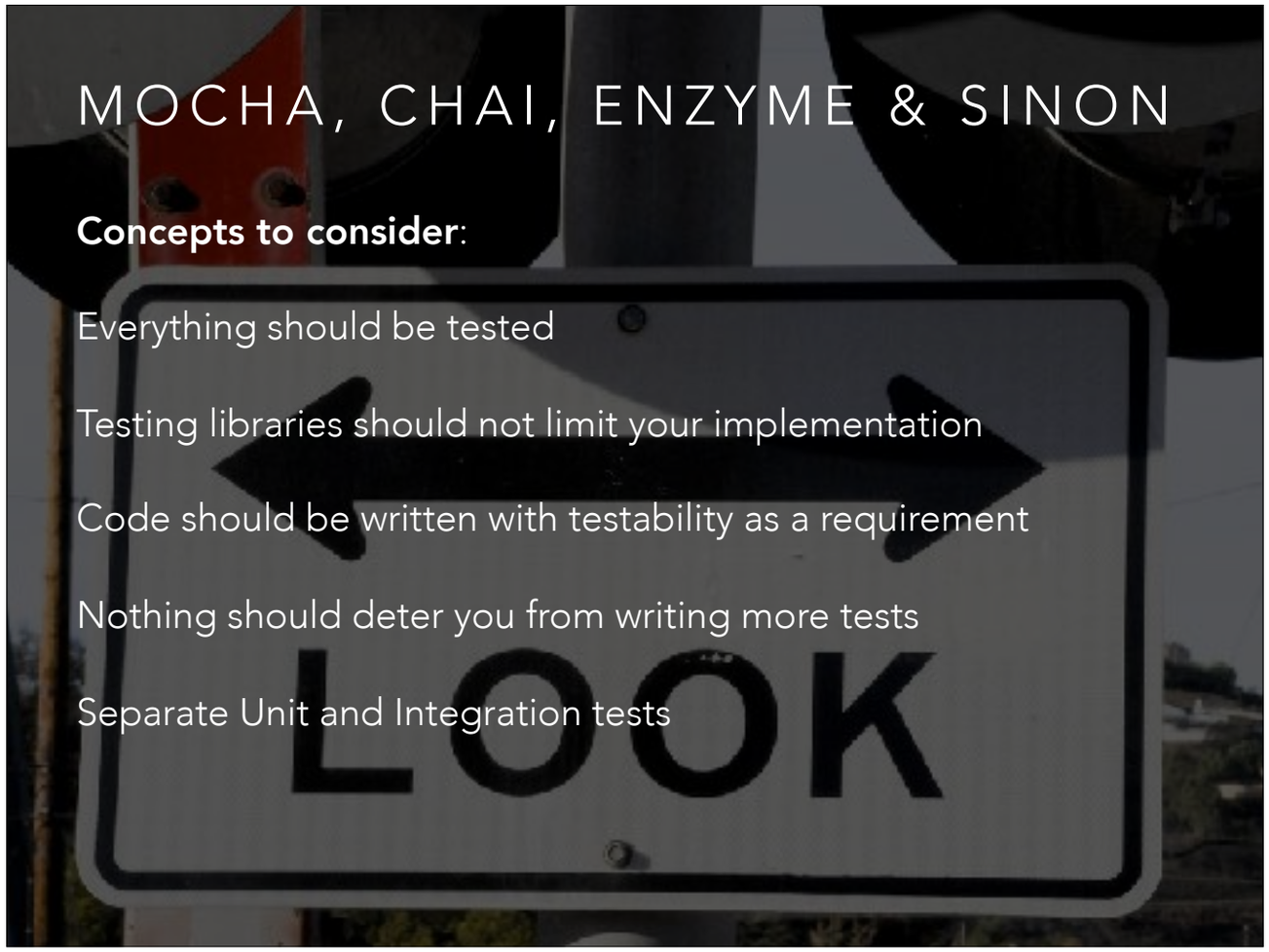


45:30:00

Why we chose what we did:
devpack-web-server is something that I never want to give up
I have yet to need a plugin that wasn’t already written
lower learning curve than gulp/grunt imho
i also think that it is easier to maintain build configuration file
webpack-dev-server



46:30:00
Look both ways. A good developer looks both ways before crossing a one way street.



47:00:00
Concepts to consider when choosing testing libraries:

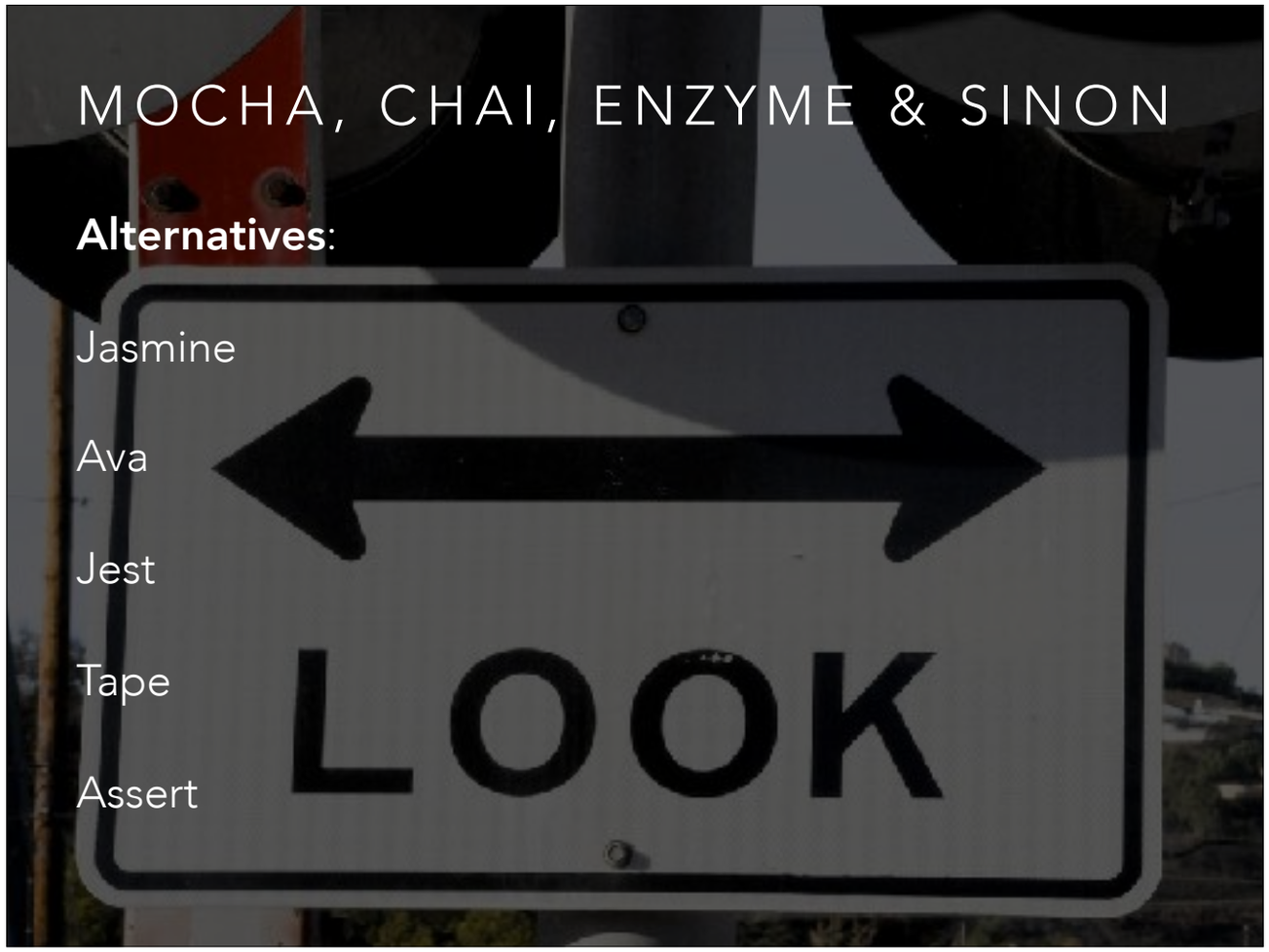
Everything should be tested:
“That scenario is not possible” is a lie

Testing libraries should not limit your implementation:
asynchronous calls, promises, async await, etc, should be fully supported

Code should be written with testability as a requirement:
This is a strongly held belief, our testing infrastructure should help enable this.

nothing deter from testing:
If testing slows drastically as you write tests, you’re using the wrong library or need to write code that is more testable

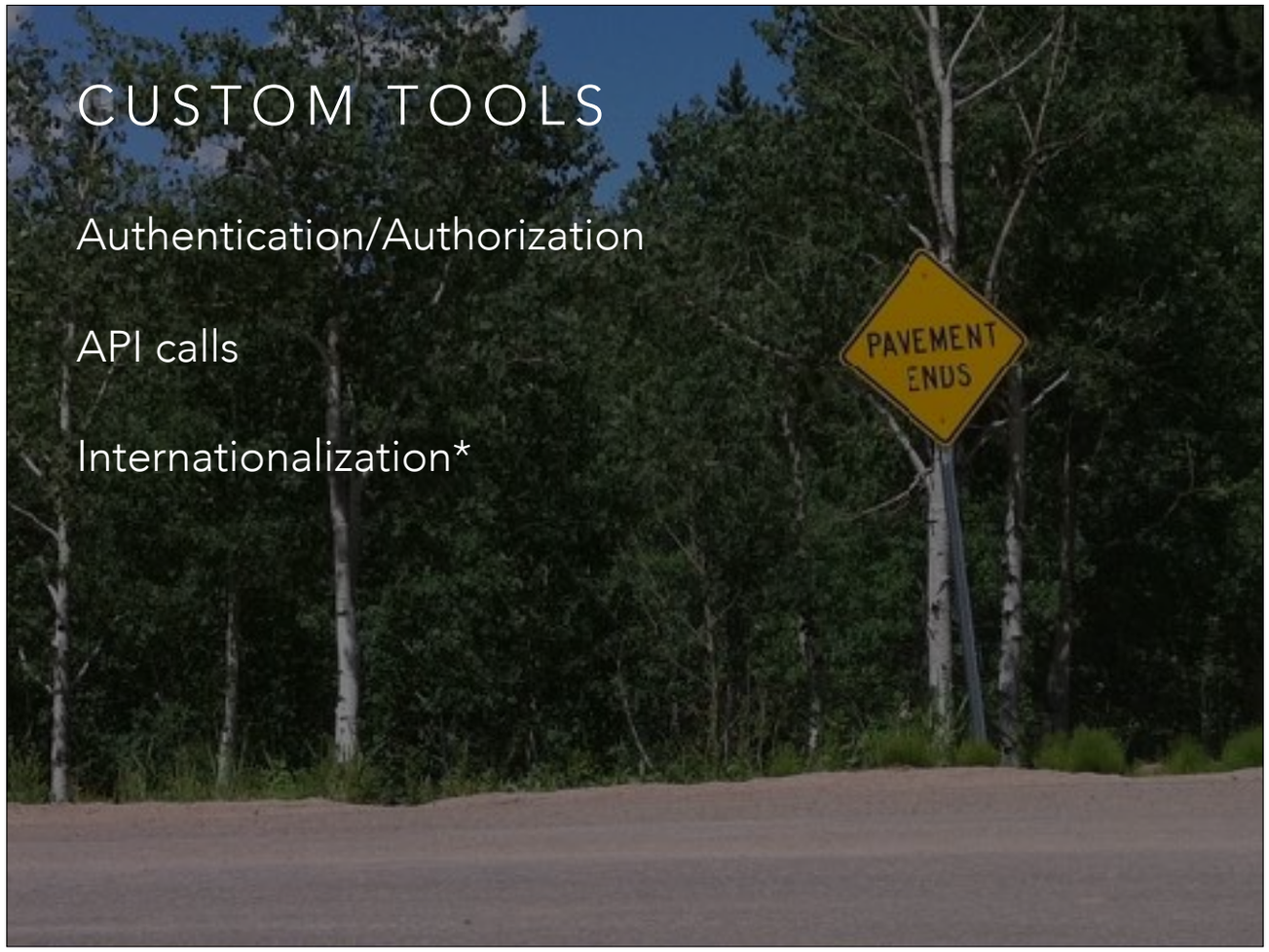
Integration Tests run by Continuous Integration environment:
We did not choose an all-encompassing testing tool. This is very good at a specific part.



48:30:00
What we looked at:
Mocha : test-runner
Chai : assertion library
Enzyme: react rendering tool
Sinon: Mock/Stub tool

Won’t go into detail about each. Lots of comparisons on the internet.
Each can be substituted out if they end up not fulfilling needs
Some packages were all-inclusive, but that meant they could be restrictive
Some were focused on React, but that focus came at a speed cost

49:00:00
Why we chose what we did:
None of the parts tried to be the other, they did what they did the best that they could
All of the parts work great together
code coverage and instrumenting with istanbul works great with mocha

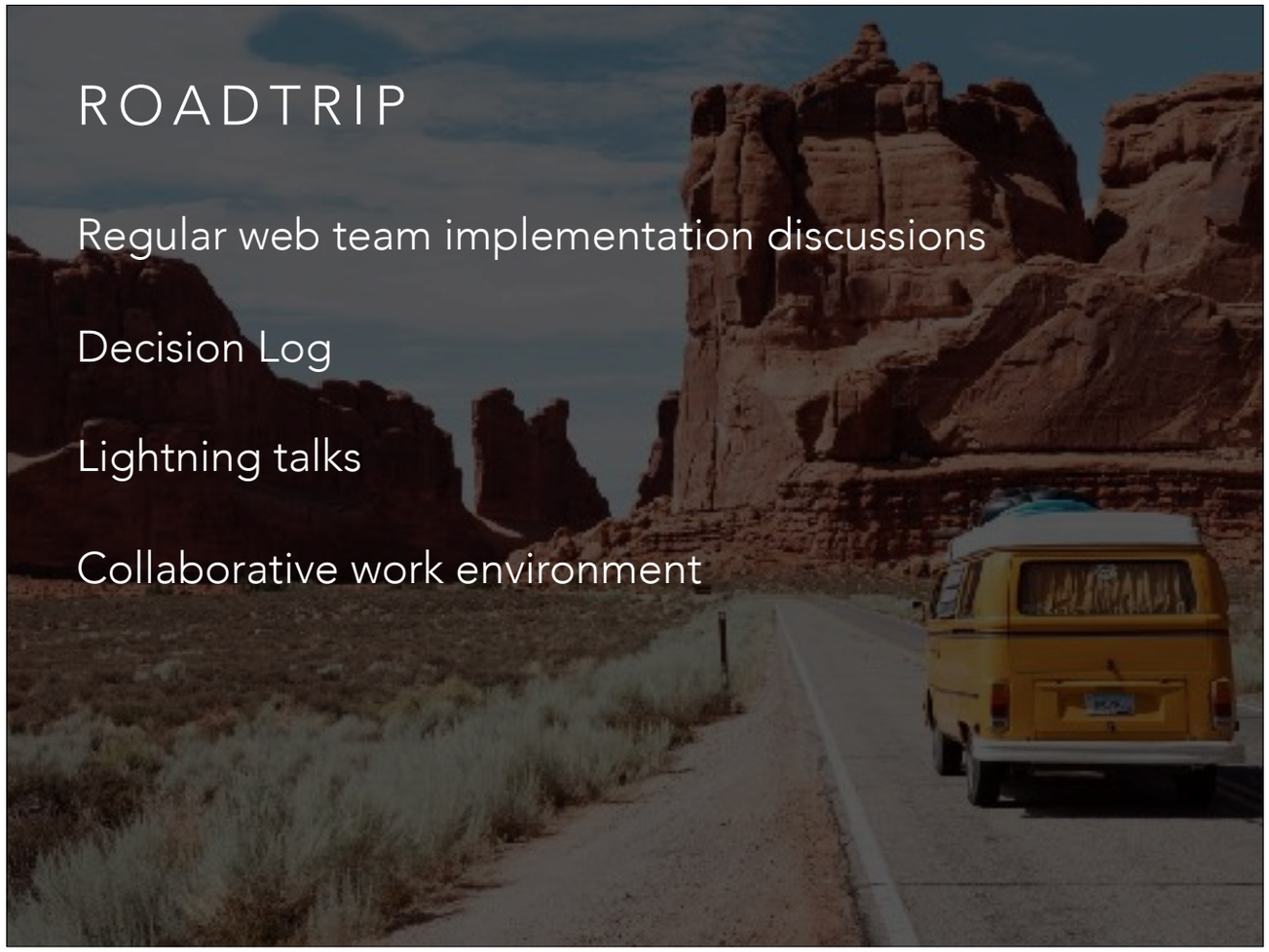


50:00:00
The things we didn’t change

Authentication
relies to heavily on custom server side authentications/authorizations to use generic FE library

API calls
Redux middleware
lots of learning on this front, come find me if you’d like to chat about redux and asynchronous data handling

Internationalization
we established translations process with custom i18n library. No need to change yet, although ant design may have us rethink that.
our journey is never complete, we are constantly re-evaluating where we are. i18n is great example
This is a working presentation. It will likely change, every year.



52:00:00
ROADTRIP: How we decide where to go.

Regular web team implementation discussions
We use PR comments to define agenda for next meeting
limit meetings to no more than 4 topics, they'll take up an hour.
No lone wolves
healthy discussion and disagreement is beneficial to all with an open mind
prevents us from falling into the "we do it that way because that's how we've always done it"

Decision Log
record decisions for posterity and/or review

Lightening talks:
knowledge is best shared
necessity for management to understand that this needs to be built into sprint scheduling. sprint flex time is a necessity and completely pays for itself in the end.

Collaborative work environment
ask for help on chat channel, standup desks, whiteboards everywhere

54:00:00
RE-ROUTING: when what we decided no longer suits our needs

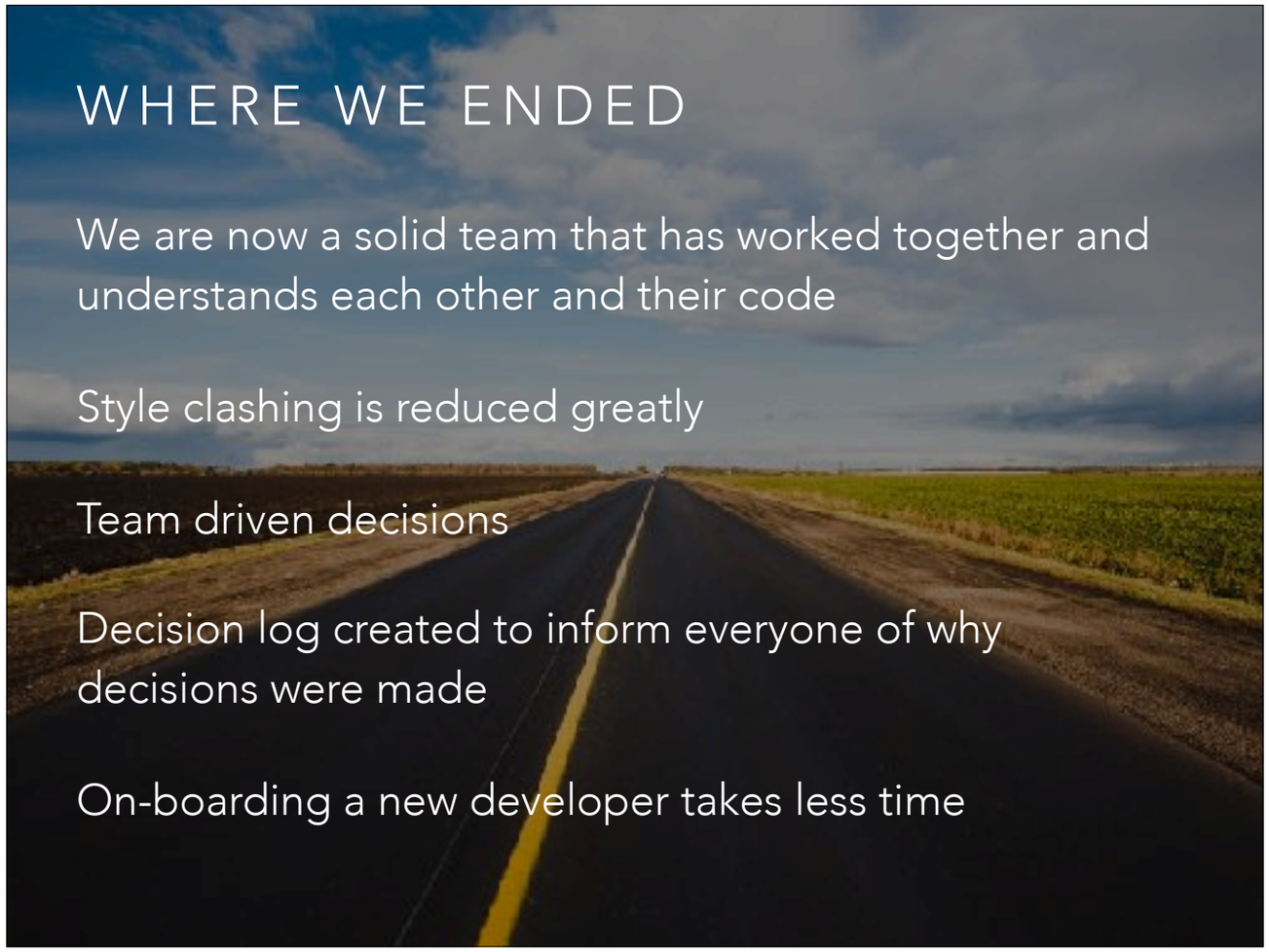
Updating the project infrastructure: watch out for unintended results.

Project regression: (list of current functionality that we can’t live without)
test:watch (tricky with TypeScript)
webpack-dev-server and hot reloading

development wins cannot be at expense of prod builds (cache busting, file minification)

regression testing each of the build targets (prod, dev, test)
source mapped tests and dev builds, no source map prod builds

run your integration tests: (things that unit tests can’t cover)
multi-browser feature support
css preprocessor output manual verification, things that not even integration tests can test



56:00:00

We are now a solid team that has worked together and understands each other and their code
Style clashing is reduced greatly
Team driven decisions
Decision log created to inform everyone of why decisions were made
On-boarding a new developer takes less time

summary: deliverability, maintainability and expandability of our product has been greatly increased

58:00:00

Please find me if you disagree with anything I’ve gone over today, I’d enjoy discussing

https://github.com/natergj/webapp-base-project