Nick Nisi
KCDC 2024

Unleashing the TypeScript Compiler
Adventures in practical code exploration and modification
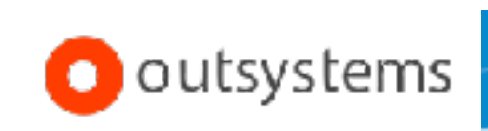
Who is this talk for?
Software Developers

- TypeScript developers

- Developers interested in Developer Experience (DX)

- Developers facing large refactors

- Anyone who wants to make some **really cool tools!**

This is a story
Eternally doomed to strive for laziness

- When I joined my last team, I broke everything on my first PR 💀

    - Code scanner to find unused i18n keys

        - I didn't account for statements like `t(keys[name]);` or `t(`core.${name}`);`

- Find untranslated text in an app

- Rewrite tests to use a custom renderer

- Various scripts to analyze and report

Aren't there better tools?
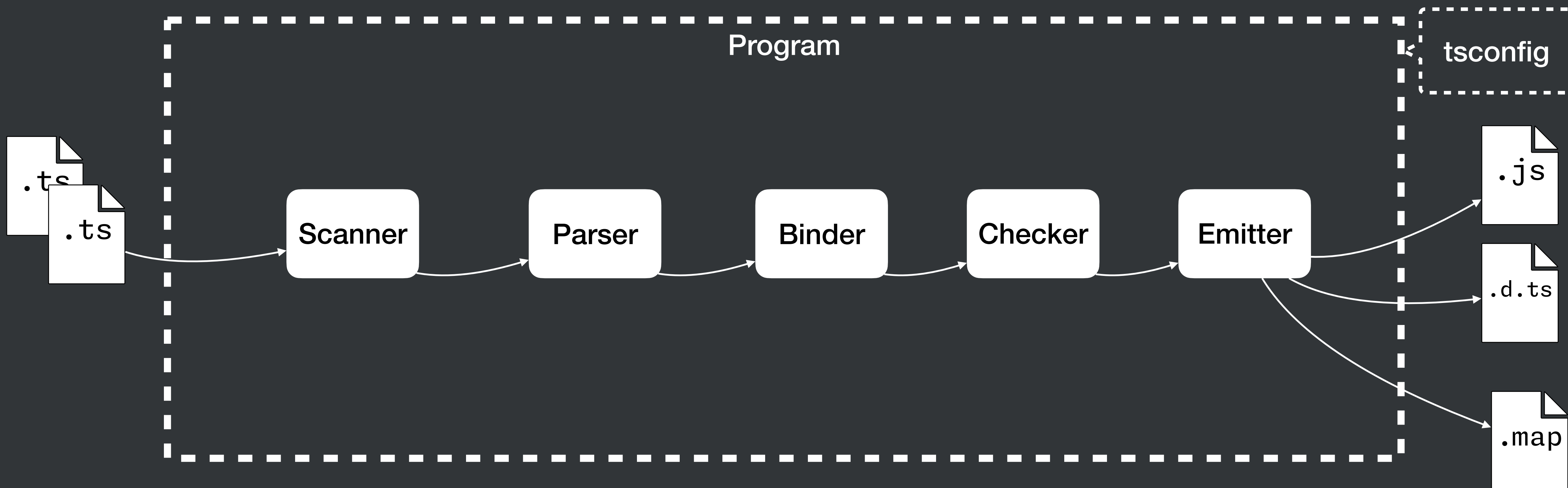## Maybe. Shut up.

- Used tools like <u>jscodeshift</u> to write codemods. It's fine.

- Wrote automated upgrader tool for <u>Dojo Toolkit</u> back in the day

- New tools like <u>ast-grep</u> are wonderful for querying and one-offs codemods

- Meanwhile, I wanted to learn more about the **TypeScript compiler**…
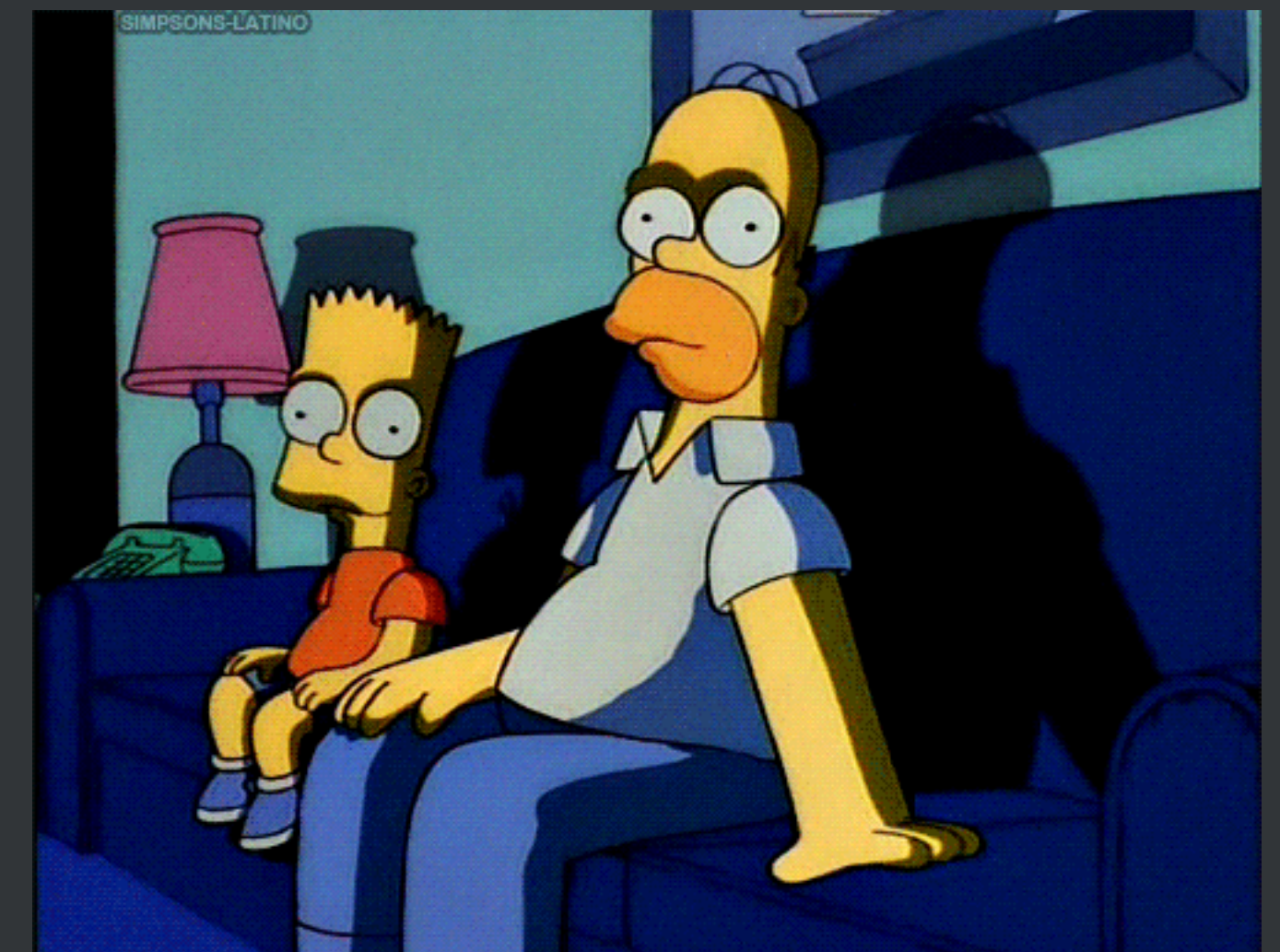
TypeScript **Compiler**

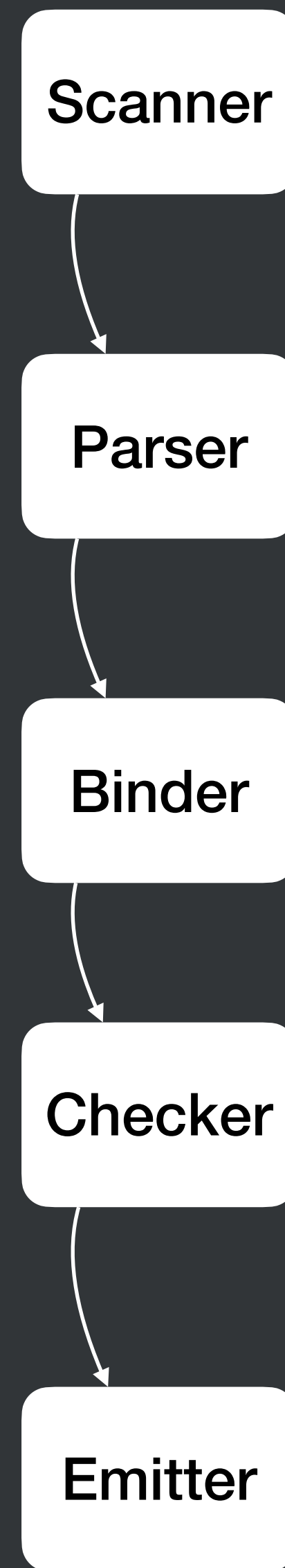- **TypeScript** is strongly-typed superset of the JavaScript programing language

  - Adds static types

  - Enhances code quality and developer efficiency by catching errors at compile time and provides for robust developer tooling through a language server

- The **TypeScript Compiler** is a tool that transforms TypeScript into JavaScript code

  - Language Service API

  - Customizing module resolutions

The TypeScript Compiler

Program

tsconfig

.ts
.ts

Scanner → Parser → Binder → Checker → Emitter

.js
.d.ts
.map

👨 @nicknisi .com

Program

tsconfig

.ts

.ts

Scanner → *Tokens* → Parser → *AST* → Binder → *Symbols* → Checker → *Type checks* → Emitter

Transpiled code

.js

.d.ts

.map

🧔 @nicknisi.com

Program

tsconfig

.ts
.ts

Scanner
Tokens

Parser
AST

Binder
AST

Checker
Symbols   Type checks

Change
AST

Emitter

Improved
TypeScript

.js

.d.ts

Transpiled code

.map

👤 @nicknisi.com

Program

- Scanner
  - ↓ *Tokens*
- Parser
  - ↓ *AST*
- Change AST
  - ↓
- Emitter
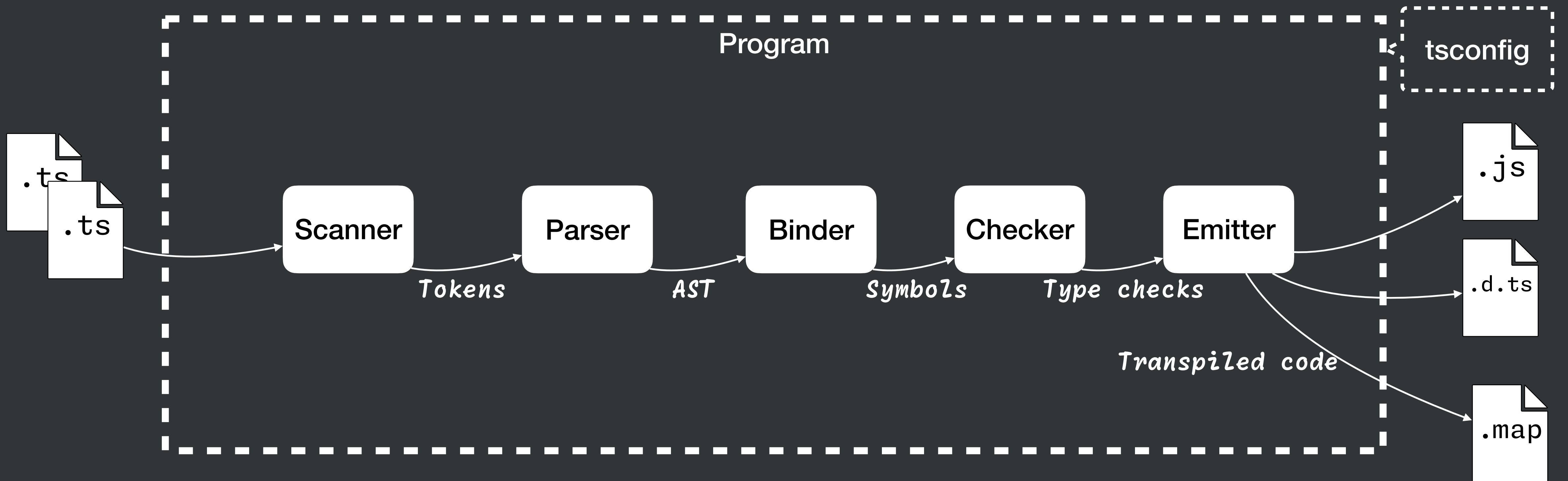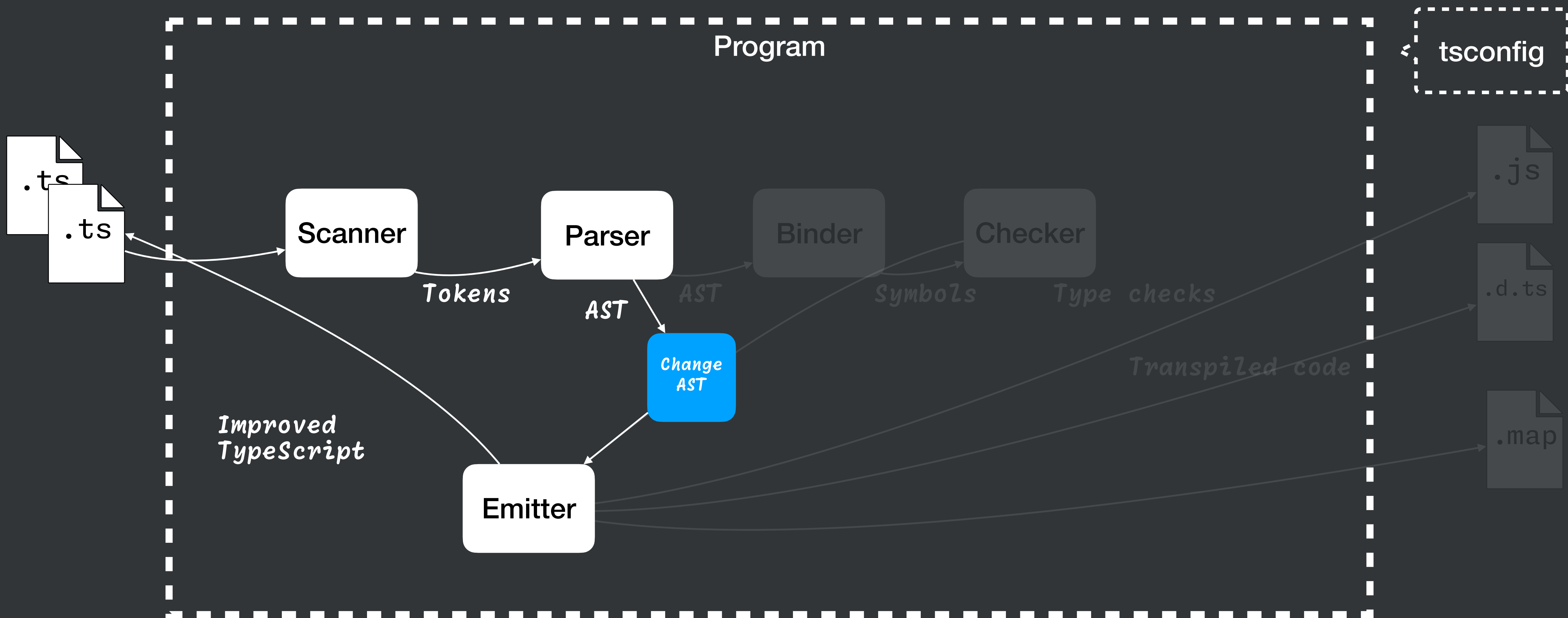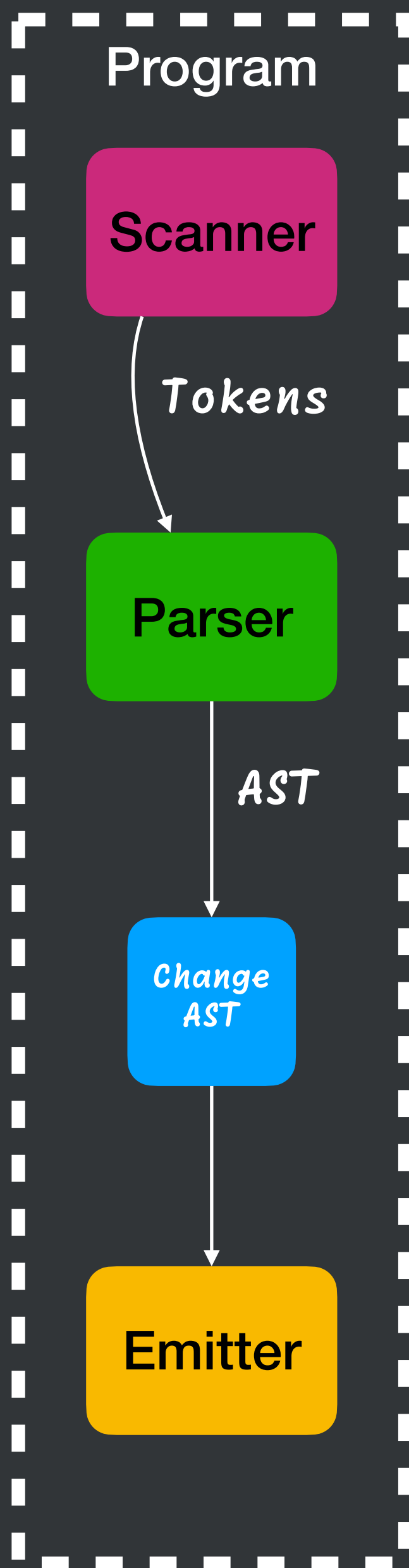
```typescript
import { writeFileSync } from "fs";
import ts from "typescript";

const program = ts.createProgram(["main.ts"], {
  module: ts.ModuleKind.ESNext,
  noImplicitAny: true,
});

const rootFileNames = program.getRootFileNames();

for (const rootFileName of rootFileNames) {
  // get the AST
  const ast = program.getSourceFile(rootFileName);

  // Perform transformations on the AST

  // write out the results
  const printer = ts.createPrinter({ newLine: ts.NewLineKind.LineFeed });
  writeFileSync(rootFileName, printer.printFile(ast));
}
```
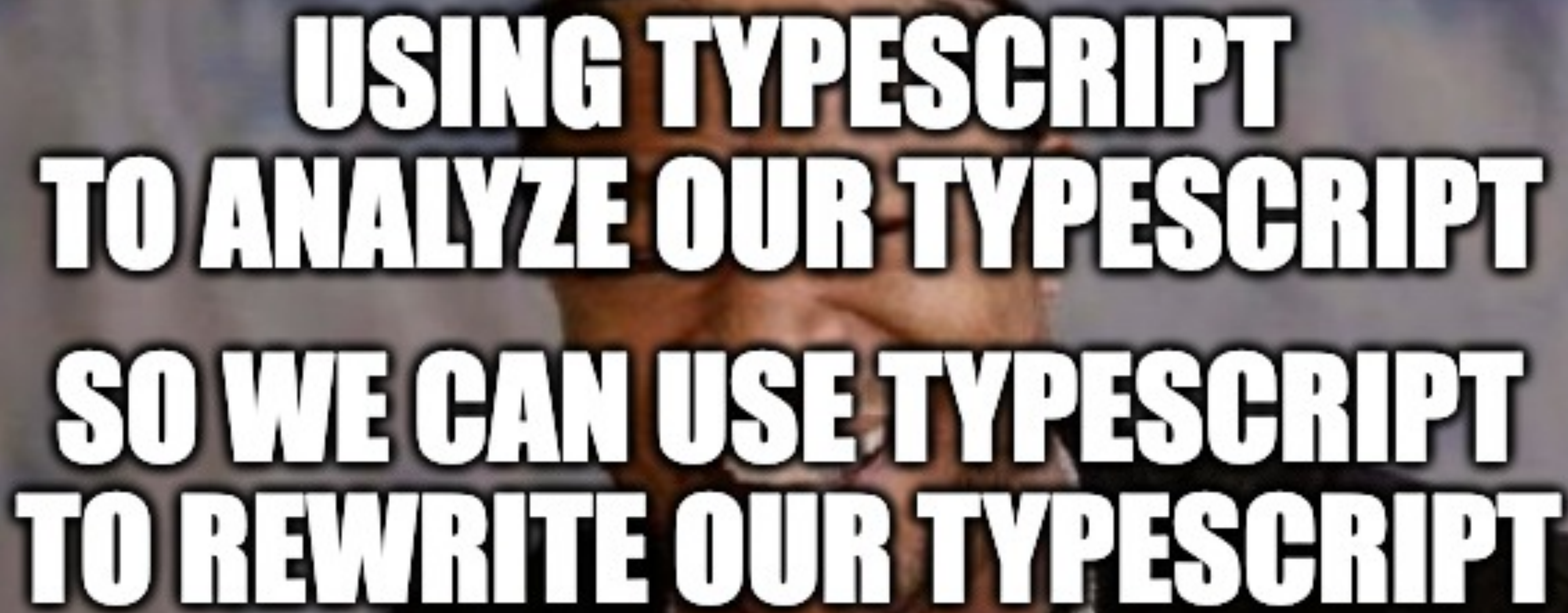
🧑 @nicknisi.com

USING TYPESCRIPT
TO ANALYZE OUR TYPESCRIPT
SO WE CAN USE TYPESCRIPT
TO REWRITE OUR TYPESCRIPT

Abstract Syntax Trees

ASTs

- An intermediate representation of source code as a tree structure

- Starts with a root node that points to other nodes, which point to others…

- Nodes represent constructs occurring in the source code

  - Literals, operators, identifiers

- Edges represent relationships between nodes

```
const bestPodcast = "JS Party";
```


VariableStatement

VariableDeclarationList

VariableDeclaration

Identifier

SourceFile

StringLiteral

EndOfFileToken
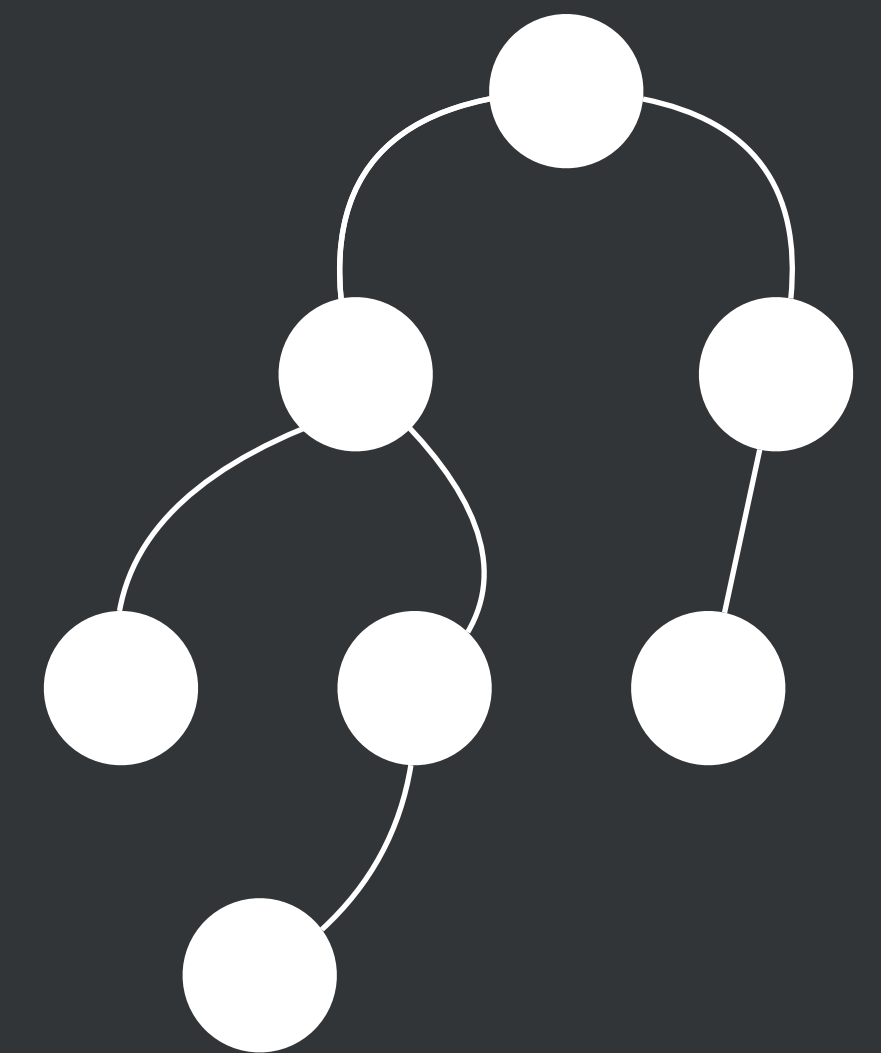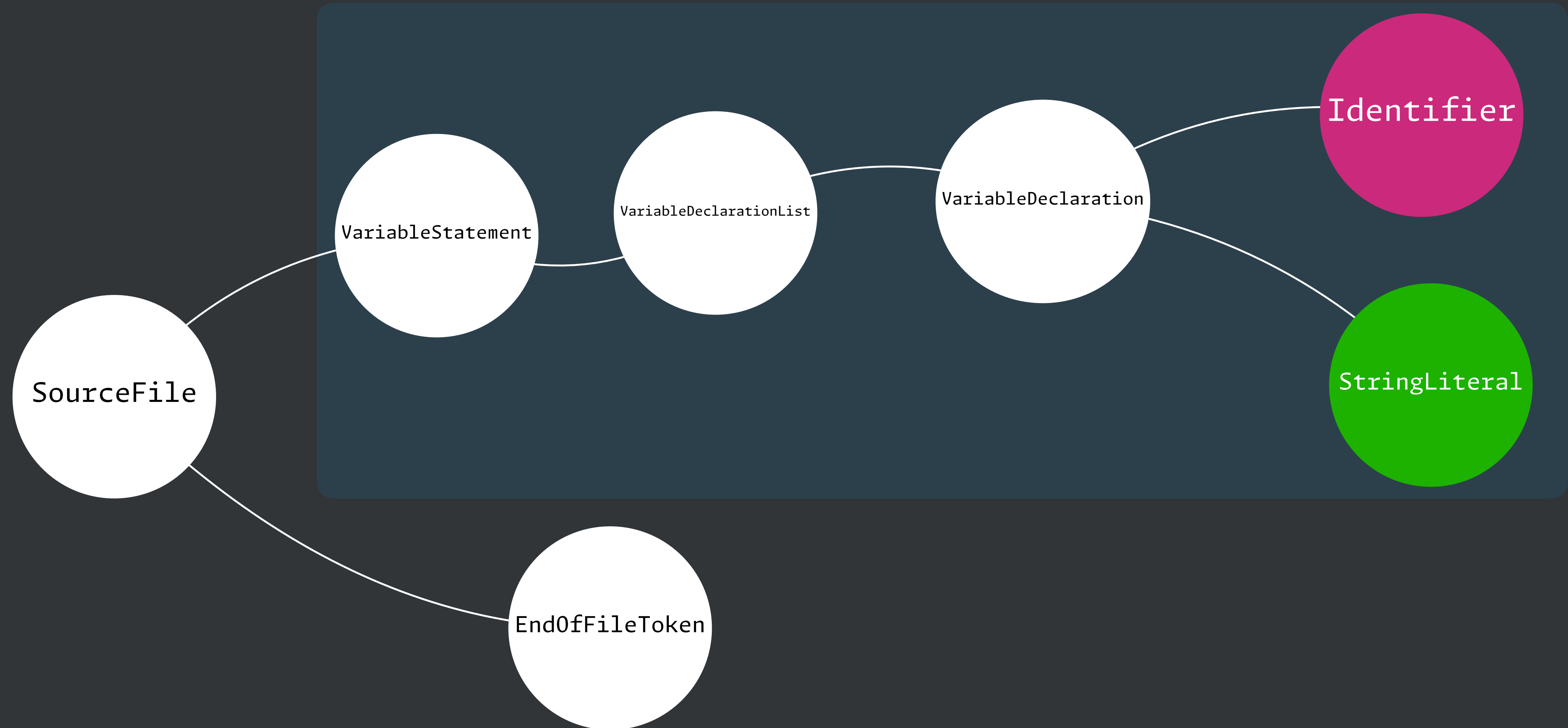
@nicknisi.com

```
const bestPodcast = "JS Party";
alert(bestPodcast);
```

SourceFile

VariableStatement
VariableDeclarationList
VariableDeclaration
Identifier
StringLiteral

ExpressionStatement
CallExpression
Identifier
Identifier

EndOfFileToken
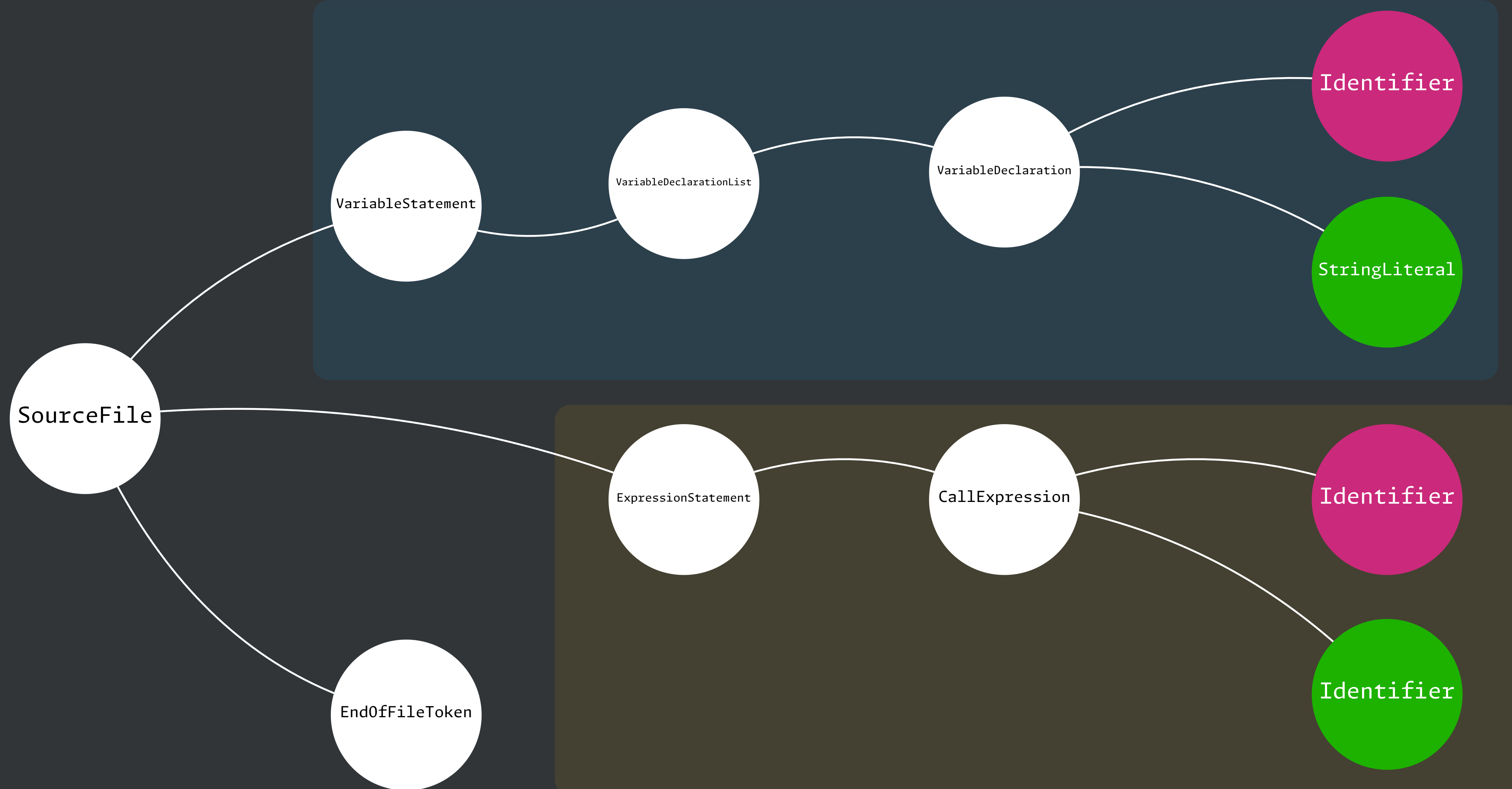
@nicknisi.com

JS Party

Close

jsparty.fm

👨 @nicknisi.com

Using ASTs

Why are we looking at ASTs?
Because parsing gets to the root of the code! 🌳

- ASTs let us **find** with precision

  - Find exactly where a function is called

  - Ignore commented-out code

  - Determine what a variable is named and then follow its usage throughout a file

  - Count exactly how many times a function is called in a codebase

- ASTs let us **replace** with precision

  - Change variable names

  - Replace deprecated API calls

Another experiment at work
Another attempt at i18n mods

- Find all `JsxText` nodes that aren't whitespace

- Count them to get an idea of how much of an app isn't translated

  - Assume that the existence of this text indicates non-translated text

  - Might be an approximation, but the script can be refined and re-run

A simple example: English strings

```tsx
import { useTranslation } from 'react-i18next';

export default function Main() {
  const { t } = useTranslation();


  return (
   <div>
    <h1>My App</h1>
    <p>{t('Hello, World!')}</p>
   </div>
  );
}
```

- Generates 639 lines of AST JSON

- We're specifically interested in `JsxText`, or the "plain English" in the file.

- Not interested in "plain English" passed to the `t` function, for example.

```
SourceFile
 ▼ImportDeclaration
  ▼ImportClause
   ▼NamedImports
    ▼ImportSpecifier
       Identifier
     StringLiteral
 ▼FunctionDeclaration
    ExportKeyword
    DefaultKeyword
    Identifier
  ▼Block
   ▼VariableStatement
    ▼VariableDeclarationList
     ▼VariableDeclaration
      ▼ObjectBindingPattern
       ▼BindingElement
          Identifier
      ▼CallExpression
          Identifier
   ▼ReturnStatement
    ▼ParenthesizedExpression
     ▼JsxElement
      ▼JsxOpeningElement
          Identifier
        JsxAttributes
        JsxText
      ▼JpxElement
       ▼JsxOpeningElement
          Identifier
        JsxAttributes
        JsxText
       ▼JsxClosingElement
          Identifier
        JsxText
      ▼JsxElement
       ▼JsxOpeningElement
          Identifier
        JsxAttributes
       ▼JsxExpression
        ▼CallExpression
          Identifier
          StringLiteral
       ▼JsxClosingElement
          Identifier
        JsxText
       ▼JsxClosingElement
          Identifier
     EndOfFileToken
```

src/examples/i18n/main.tsx

🧑 @nicknisi.com

📁 src > 📁 examples > 📁 i18n > ⚛ main.tsx > {} import

```tsx
1  import { useTranslation } from 'react-i18next';

2  export default function Main() {
3    const { t } = useTranslation();

5    return (
6      <div>
7        <h1>My App</h1>
8        <p>{t('Hello, World!')}</p>
9      </div>
10   );
11 }
```
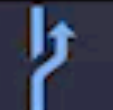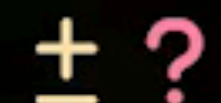
± ? main

```typescript
import ts from 'typescript';

function getDescendantsOfKind<T extends ts.Node>(node: ts.Node, kind: ts.SyntaxKind): T[] {
  const descendants: T[] = [];

  function visit(node: ts.Node) {
    if (node.kind === kind) {
      descendants.push(node as T);
    }
    ts.forEachChild(node, visit);
  }

  visit(node);
  return descendants;
}
```
(We'll talk about simplifying this with ts-morph later)
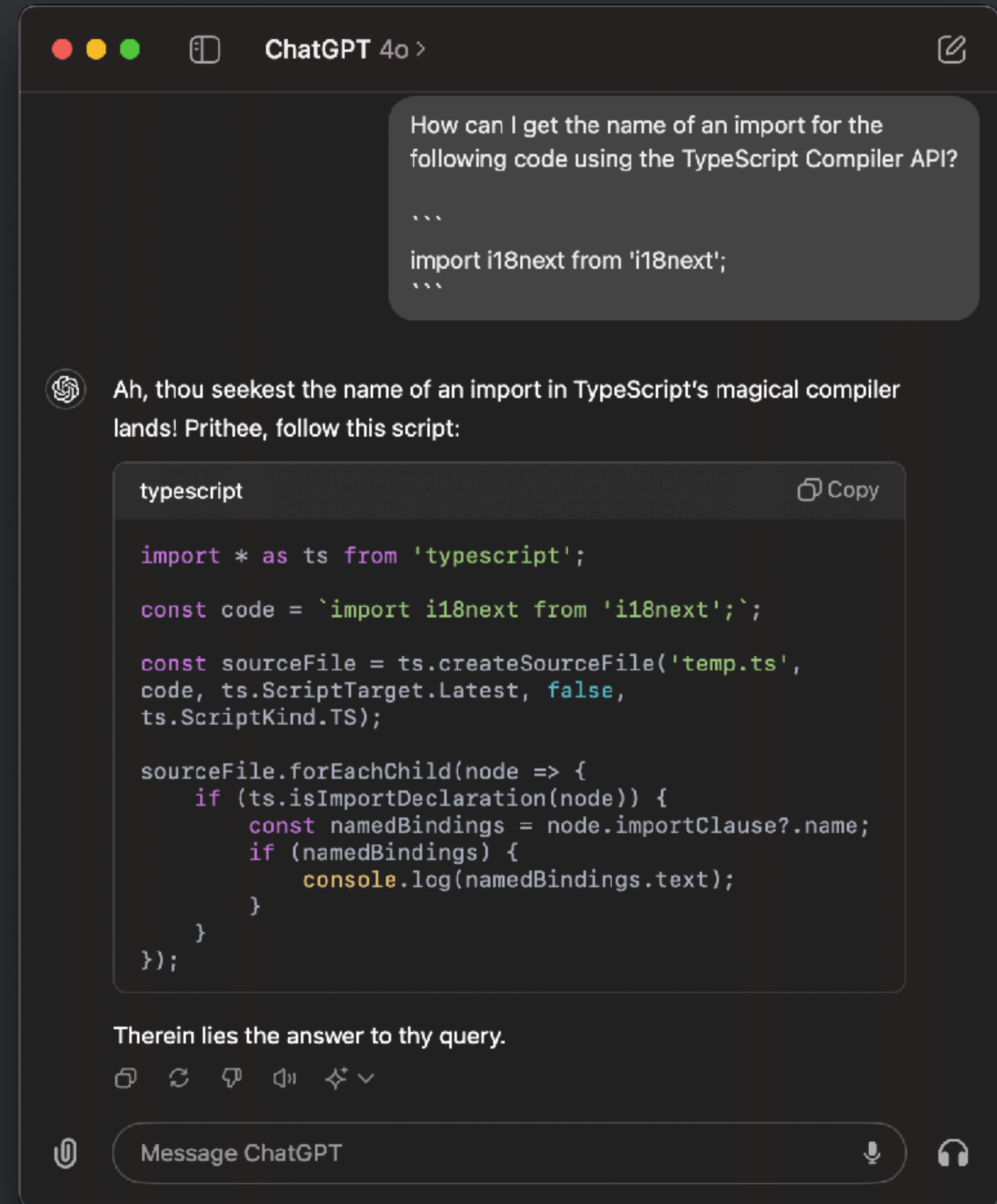
```typescript
const clean = (text: string) => text.replace(/[.,\/#!$%\^&\*;:{}=\-_~()]/g, '').trim();

const program = ts.createProgram(['src/examples/i18n/main.tsx'], {
  module: ts.ModuleKind.ESNext,
  jsx: ts.JsxEmit.React,
  target: ts.ScriptTarget.ESNext,
});

const filename = program.getRootFileNames()[0]!;
const ast = program.getSourceFile(filename)!;
const texts = getDescendantsOfKind<ts.JsxText>(ast, ts.SyntaxKind.JsxText).filter(text => clean(text.text));
texts.forEach(text => console.log(`JSX Text: ${text.text}`));
```

🧑 @nicknisi.com

src/examples/i18n/ast.ts

Getting help with ASTs

- [ASTExplorer](#)

- [TS AST Viewer](#)

- ChatGPT is actually pretty good at it

```
1 const bestPodcast = "JS Party";
2 alert(bestPodcast);
```

Tree | JSON | 1ms

```
1 {
2     "pos": 0,
3     "end": 51,
4     "flags": 0,
5     "modifierFlagsCache": 0,
6     "transformFlags": 4457472,
7     "kind": 308,
8     "statements": [
9         {
10             "pos": 0,
11             "end": 31,
12             "flags": 0,
13             "modifierFlagsCache": 0,
14             "transformFlags": 4457472,
15             "parent": "[Circular ~]",
16             "kind": 240,
17             "declarationList": {
18                 "pos": 0,
19                 "end": 30,
20                 "flags": 2,
21                 "modifierFlagsCache": 0,
22                 "transformFlags": 4457472,
23                 "parent": "[Circular ~.statements.0]",
24                 "kind": 258,
25                 "declarations": [
26                     {
27                         "pos": 5,
28                         "end": 30,
```

Mo' MUI Mo' Problems
Muy MUI Problems - let's get back to the story

- 🙈 Our main app is written in TS and React, using Material-UI v4.x

- 👴 MUI 4 is using old versions of everything (TypeScript 3 😱, React 16 😱)

- 💄 MUI 4 uses JSS, MUI 5 is emotion

- 🌕 We were sowing wild oats, experimenting with things like RSC and Next.js

- 🍹 We also heard the good word about Tailwind

How do we justify the work?

Tailwind is great

CHANGE MY MIND

Analyze first

We need to know what we're getting into

Analyzing a codebase

- Create a list of all files that are using MUI components

- Get a list of MUI components used in those files, sorted by most used

- Make a reporting tool to provide data about the number of component changes that can be automated away

  - Help determine that `<Grid />` and `<Box />` usage is high in all components

```typescript
let used = 0;
files.forEach(file => {
  const source = readFile(join(MAIN_PATH, file), 'utf8');
  const ast = ts.createSourceFile(file, source, ts.ScriptTarget.Latest, true);
  ts.forEachChild(ast, node => {
    if (node.kind === ts.SyntaxKind.ImportDeclaration) {
      const importDeclaration = node as ts.ImportDeclaration;
      const importPath = importDeclaration.moduleSpecifier.getText().replace(/[\'\"]/g, '');
      if (importPath === '@material-ui/core') {
        ++used;
        const { importClause } = importDeclaration;

        if (importClause) {
          importClause.namedBindings?.forEachChild(node => {
            const importSpecifier = node as ts.ImportSpecifier;
            const importName = importSpecifier.name.getText();
            if (importName && !importName.endsWith('Props')) {
              const value = components.get(importName) ?? 0;
              components.set(importName, value + 1);

              const project = file.split('/').slice(0, 3).join('/');
              const projectValue = projects.get(project) ?? 0;
              projects.set(project, projectValue + 1);
            }
          });
        }
      }
    }
  });
});
```

🧑 @nicknisi.com

scripts/bin/analyze.ts

```
▷ npx analyze
```

```
Here's the total usage I found, per component exported by '@material-ui/core':

Total usage per component:


Component      | Count
------------   | -----
Box            | 3
Grid           | 2
ThemeProvider  | 1
SvgIcon        | 1


Overall, there are 7 components imported from "@material-ui/core" across the app.

Per-component breakdown:


File                        | Count
--------------------------- | -----
src/components/ProTip.tsx    | 3
src/components/Copyright.tsx | 2
src/main.tsx                 | 1
src/App.tsx                  | 1


Material UI components are used in 80% (4 of 5) of the components in the app.

~/Developer/ts-compiler-api-talk ⓙ v20.10.0
▷ █
```

🧑 @nicknisi.com

$ npx analyze

Reality check
What was the run like in the real repo?

- `<Box />` imported in 273 of 654 files

- `<Grid />` imported in 114 of 654 files

😱 A single developer writing a single script to focus on just these two components to start can do 39% of the conversion by themselves.

Making modifications
Say hello to my little friend

- Use the TS Compiler API to rewrite our own TS, back to the files

- Formatting can be lost, but easy to pair with a <u>Prettier</u> run to fix

- Search for `<Grid />` and `<Box />` components and replace them with Tailwind divs

ts-morph
Wrap the TS Compiler API to simplify working with it

- Wrapper around TS Compiler API to simplify working with it

- One downside of transforming with the TS Compiler API:

    - Generating new nodes to replace existing ones can be tedious

- ts-morph simplifies project setup

- Helpers for finding nodes

- Traversal control (up, down, skip)

- Element wrapper provides a `replaceWithText` to simplify new node generation

Let's create a reusable converter

- Convert layout components to divs with Tailwind classes

- Properly handle dynamic expressions

- Clean up unused imports

```
> npx convert -o                                   ☑ kcdc-2024
projectRoot: /Users/nicknisi/Developer/ts-compiler-api-talk/
glob: /Users/nicknisi/Developer/ts-compiler-api-talk/src/**/*.tsx

Converting <Box/> components...

Converting <Grid/> components...

Organizing imports...


    🚀  Converted 13 elements across 5 files

~/Developer/ts-compiler-api-talk ⬡ v20.10.0
▷                                                   ± kcdc-2024
```

1467 elements across 654 files (in a real project)

```
1:

  1  1   import { Box, Grid, SvgIcon, type SvgIconProps } from '@material-ui/core';
     1   import { SvgIcon, type SvgIconProps } from '@material-ui/core';
  2  2   import { useTranslation } from 'react-i18next';
  3  3
  4  4   function LightBulbIcon(props: SvgIconProps) {


12: function LightBulbIcon(props: SvgIconProps) {

 12 12   export default function ProTip() {
 13 13       const { t } = useTranslation();
 14 14       return (
 15                   <Box mt={6} mb={3} color="secondary">
 16                       <Grid container direction="column" alignItems="center" justifyContent="center">
 17                           <Grid item>
     15              <div className="text-secondary mb-3 mt-6">
     16                  <div className="items-center flex flex-wrap flex-col justify-center">
     17                      <div className="flex-auto box-border">
 18 18                          <LightBulbIcon />
 19                       </Grid>
 20                       <Grid item container direction="row" spacing={2} alignItems="center">
 21                           <Grid item>
 22                               <Box color="secondary">
     19                  </div>
:
```

@nicknisi.com

```typescript
export abstract class Converter {
  /**
   * Create a Converter class with a given prop lookup
   */
  static createConverter({ baseClasses = [], lookup }: CreateConverterOptions) {}
  /**
   * The base attributes that can be found on components and should be copied over, verbatim to the new
component.
   */
  static readonly baseAttributes = [] as const;

  constructor(element: JsxElement | JsxSelfClosingElement): Converter;

  /**
   * The base classes that should be added to every component that is converted
   */
  abstract lookup: PropLookup;
  /**
   * A list of base classes that should be added to every component that is converted
   */
  protected readonly baseClasses: string[] = [];
  /**
   * Convert the element into a new element string with the appropriate classes and props
   * @returns The new element string
   */
  convert(): string {}
}
```

🧑 @nicknisi.com   scripts/converter/Converter.ts

The **Grid** component
Flexbox grid component in Material-UI

```jsx
<Grid container spacing={1}>
  <Grid item>
    <a href="https://nicknisi.com">{t('core.copyright')}</a>
  </Grid>
  <Grid item>{year}</Grid>
</Grid>
```

```jsx
<div className="flex flex-wrap gap-1">
  <div className="flex-auto box-border">
    <a href="https://nicknisi.com">{t('core.copyright')}</a>
  </div>
  <div className="flex-auto box-border">{year}</div>
</div>
```

Conversion Steps

- Convert the element into a list of props

- Convert each prop into a class string or a complex class expression

- Get a list of additional props

- Construct a new **`<div>`**

  - Add the new **`className`**

  - Add remaining props back (**`data-test-id`** or **`key`**, for example)

```ts
export default Converter.createConverter({
  baseClasses: [],
  lookup: {
    alignContent: "content",
    alignItems: "items",
    container: () => "flex flex-wrap",
    direction: (_name, value) =>
      value ? `flex-${value.replace(/column/, "col")}` : "",
    item: "flex-auto box-border",
    justify: "justify",
    justifyContent: "justify",
    lg: calculateWidth,
    md: calculateWidth,
    sm: calculateWidth,
    xl: calculateWidth,
    xs: calculateWidth,
    spacing: "gap",
    wrap: "flex-wrap",
    zeroMinWidth: (_name, value) => (value === "true" ? "min-w-0" : ""),
    color: "text",
  },
});
```

Note: this is not an exhaustive list of all `<Grid>` props, only what's used in the project. I checked 😉

🧑 @nicknisi.com   scripts/converter/converters/GridConverter.ts

```typescript
convert(){
  const { attribute } = this;
  let twClass: string = "";
  const initializer = attribute.getInitializer();

  if (!initializer) {
    // this is a boolean prop
    twClass = this.runTransform(undefined);
  } else if (initializer?.isKind(SyntaxKind.StringLiteral)) {
    twClass = this.runTransform(initializer.getLiteralValue());
  } else if (initializer?.isKind(SyntaxKind.JsxExpression)) {
    const expression = initializer.getExpression();

    if (expression?.isKind(SyntaxKind.ObjectLiteralExpression)) {
      twClass = Object.entries(getProperties(expression))
        .map(([key, value]) => `${key}:${this.runTransform(value)}`)
        .join(" ");
    } else if (expression?.isKind(SyntaxKind.ConditionalExpression)) {
      this.isComplexClass = true;
      twClass = this.transformConditionalExpression(expression);
    } else {
      twClass = this.runTransform(expression?.getText());
    }
  }


  return twClass;
}
```

🧑 @nicknisi.com

scripts/convert/Prop.ts

Tips and Takeaways

- The code can be a little dense. ChatGPT is actually pretty good at finding nodes

- You only have to handle the cases in your code base. `analyze` to find those

- Encourage your team to run them as they are making other changes

  - Breaks up conversions into much smaller diffs

  - More likely to actually get reviewed 😉

  - Write tests for your codemods

    - Iteration becomes much faster

- For one-off queries/mods, check out <u>ast-grep</u>

🧍 Thanks!

typescript.fun/compiler-talk

• Freelance TypeScript Enthusiast at dilemmas.dev

• Panelist on the JS Party podcast

• **@nicknisi** on everything social

Talk resources

Speaker feedback

Questions?

🧍 @nicknisi.com