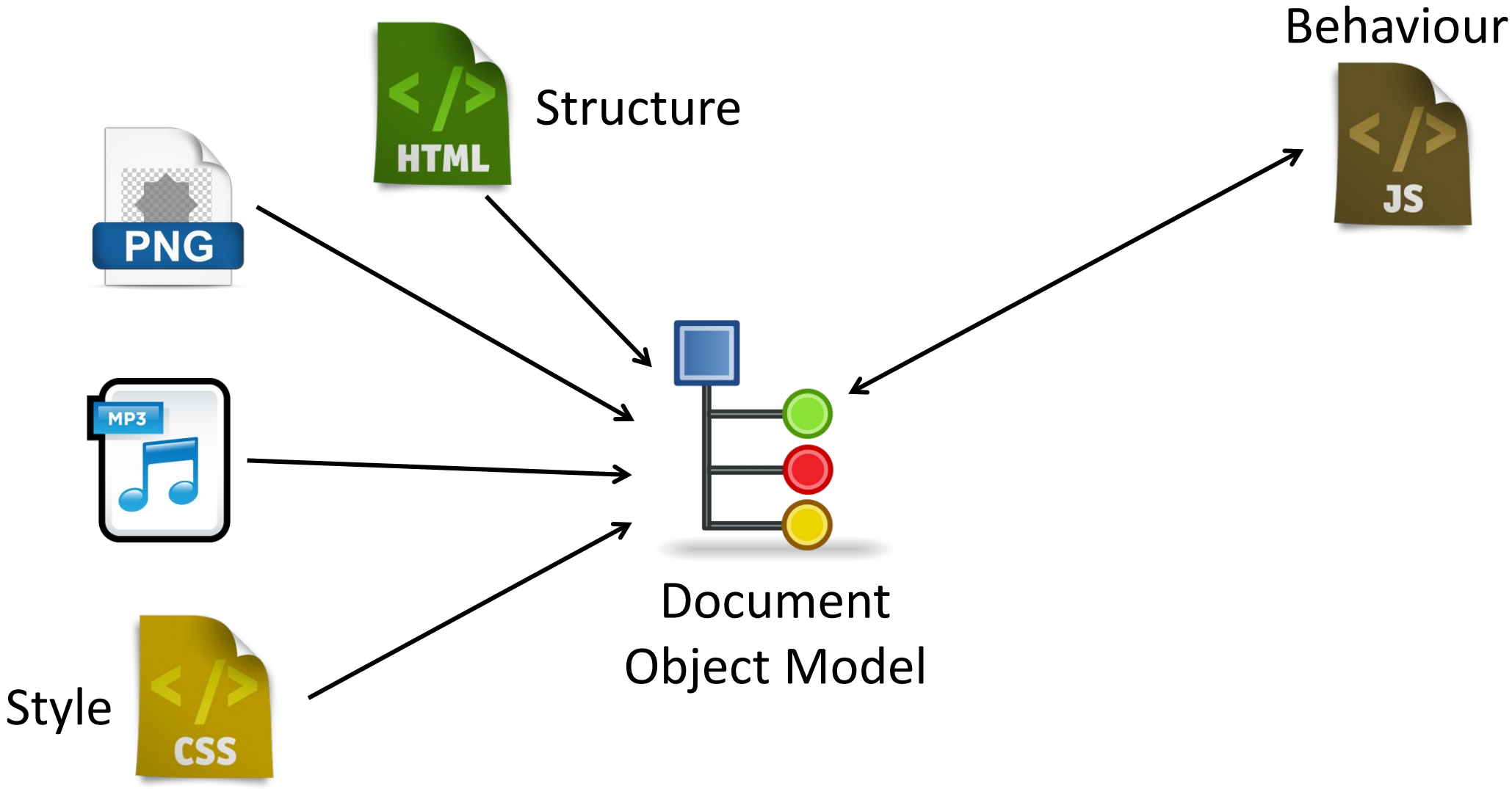
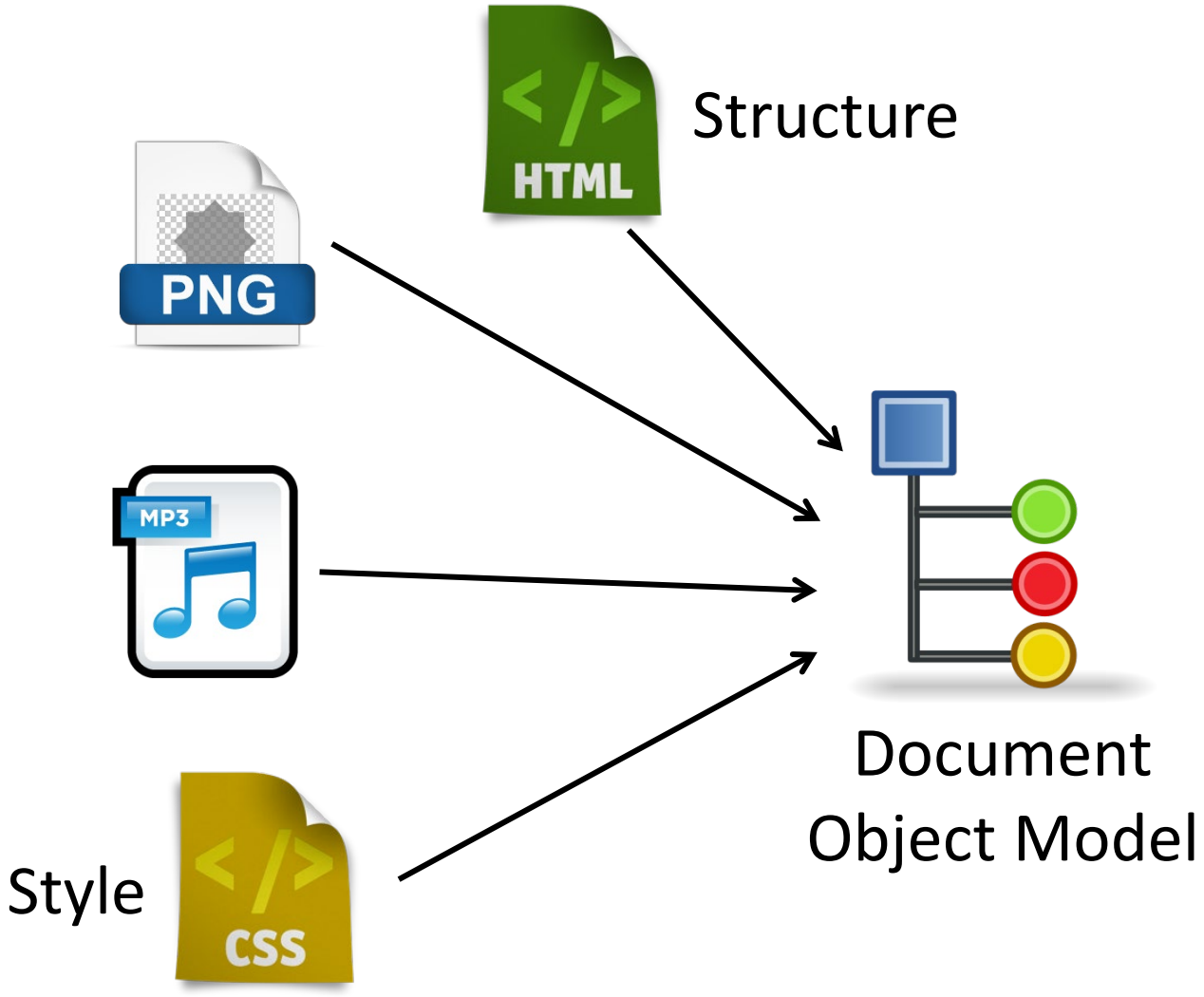
Day 31

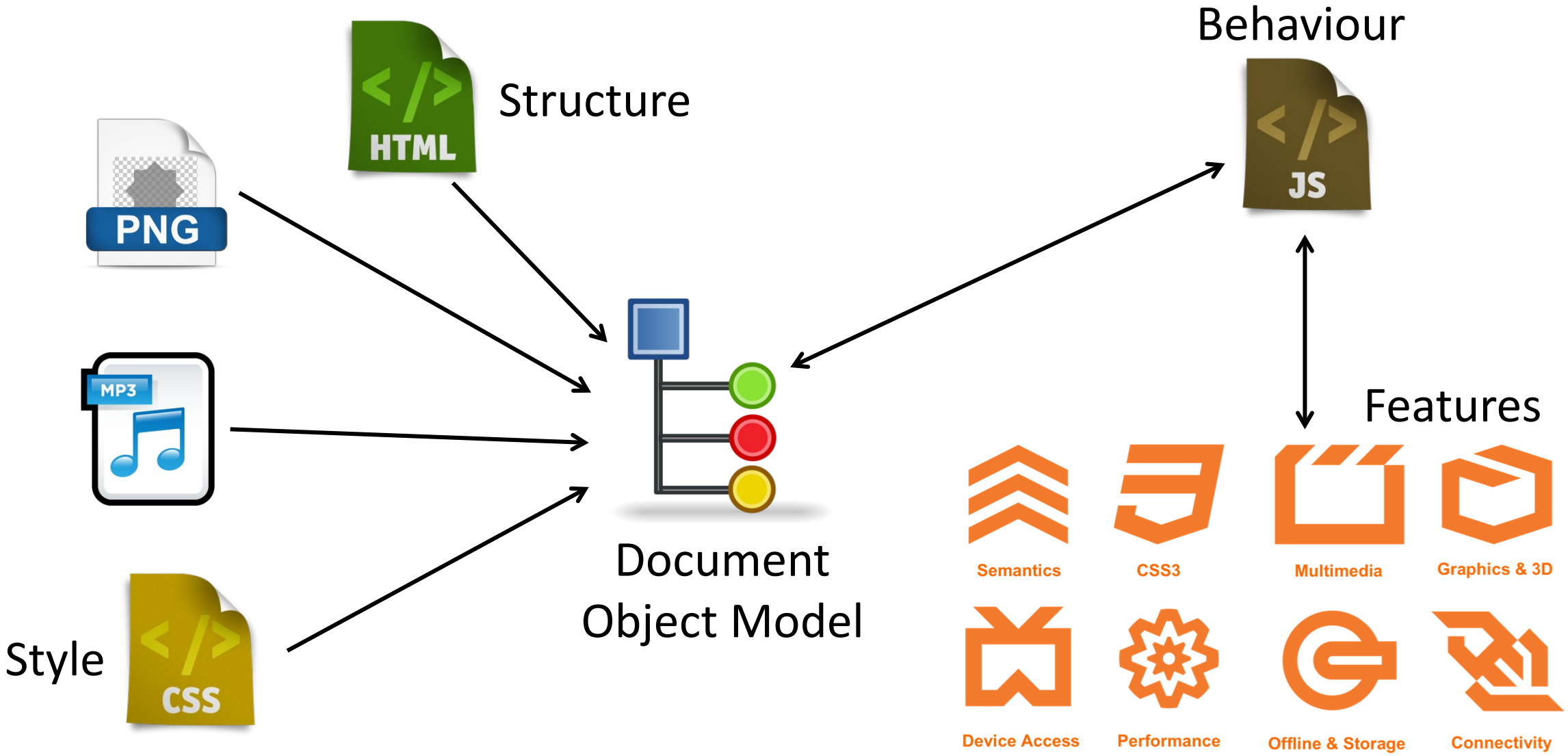
HTML5 Application



HTML5 Application



HTML5 Application



The MEAN Stack

The MEAN Stack

Browser

Angular HTML, CSS, JavaScript

HTML5 Application

Server

JavaScript

Web Application

Persistence

Database

Environment Setup

- Install TypeScript

```
npm install -g typescript
```

```
sudo npm install -g typescript
```

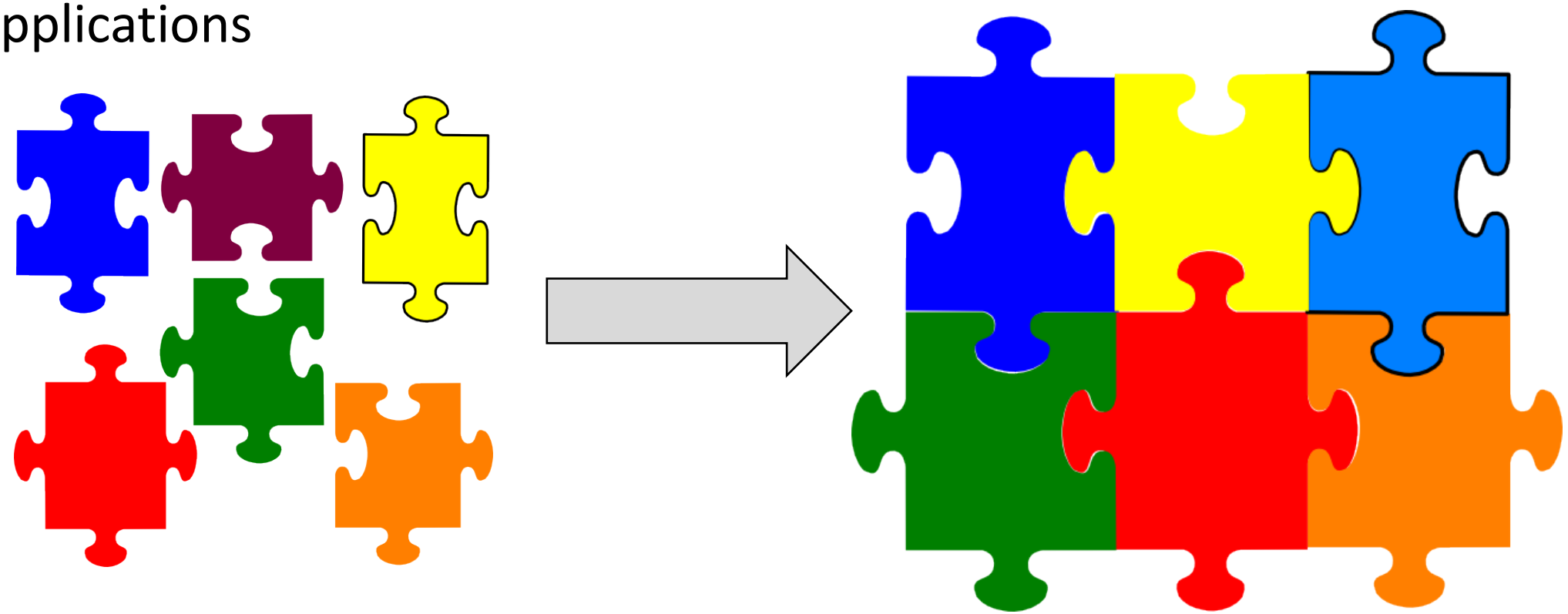
- Install Angular CLI
  - https://cli.angular.io

```
npm install -g @angular/cli
```

```
sudo npm install -g @angular/cli
```

What is Angular?

- Is a component based Typescript framework for developing HTML5 applications



Individual reusable components

Assemble components into application

Image from https://www.clker.com/clipart-jigsaw-puzzle-6-pieces-5.html

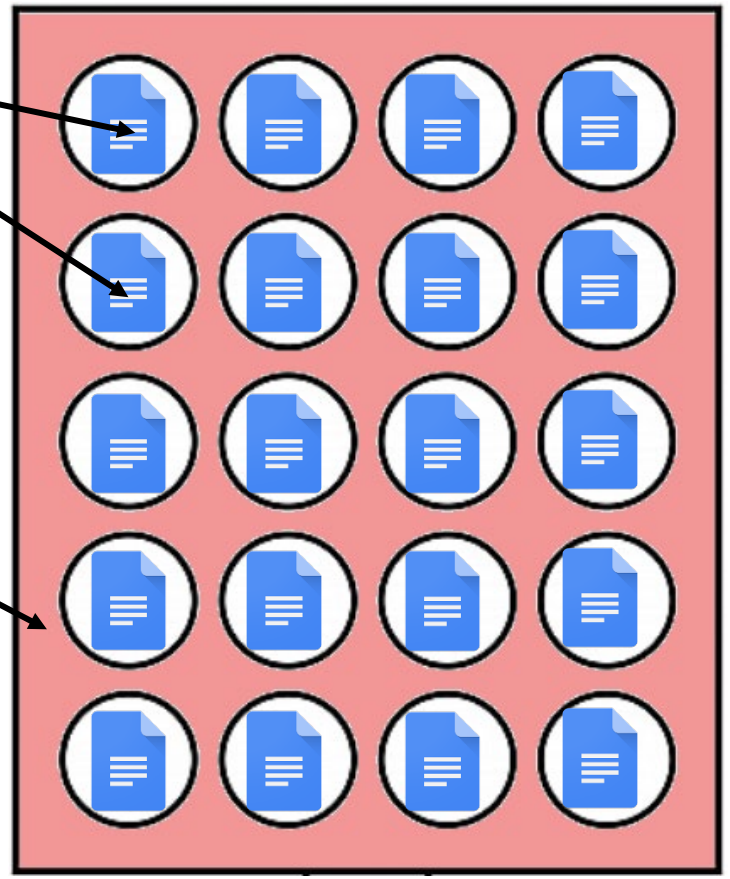
Library vs Framework

Library

Your program uses the library

The framework use your code

You write this/these



Framework

Angular Framework

You write theses

Angular provided these

The Web/Browser

Angular Workflow

- Generate a new Angular application

```
ng new <app name>
```

- Add additional libraries

```
npm install --save <module>
```

- Start development server

```
ng serve -o
```

Angular Workflow

- Generate one or more components

```
ng generate component <name>
```

- Write one or more service
- Build final application

```
ng build --prod
```

Angular Project Directory

- Generated by ng new
- Important files
  - `package.json` - list of installed modules in `node_modules`
  - `angular.json` - CLI configuration file
  - `src` - application source

`src` Directory

- Bootstrap
  - `index.html`, `main.ts`
- Global stylesheet
  - `styles.css`
- `asset` directory for images, etc.
- `app` directory
  - `app.module.ts`
  - `app.component.ts`, `app.component.css`, `app.component.html`

Application Structure

app.component.html

```
<h1>hello, world</h1>
```

app.component.css

```
h1 {
  color: red;
}
```

app.component.ts

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: [ 'app.component.css' ]
})
```

index.html

```
<app-root></app-root>
```

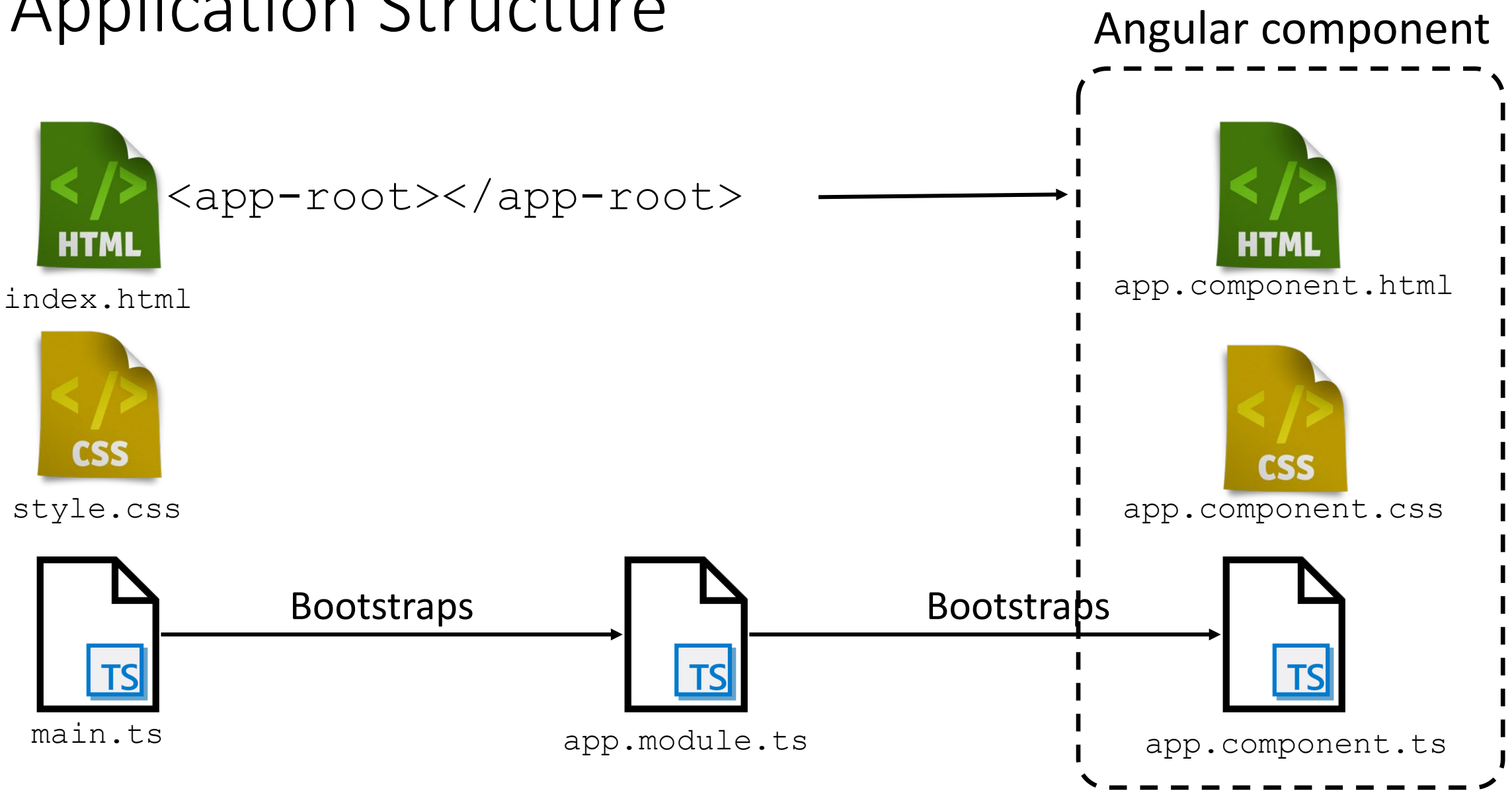
app.module.ts

```
@NgModule({
  ...
  bootstrap: [ AppComponent ]
})
```

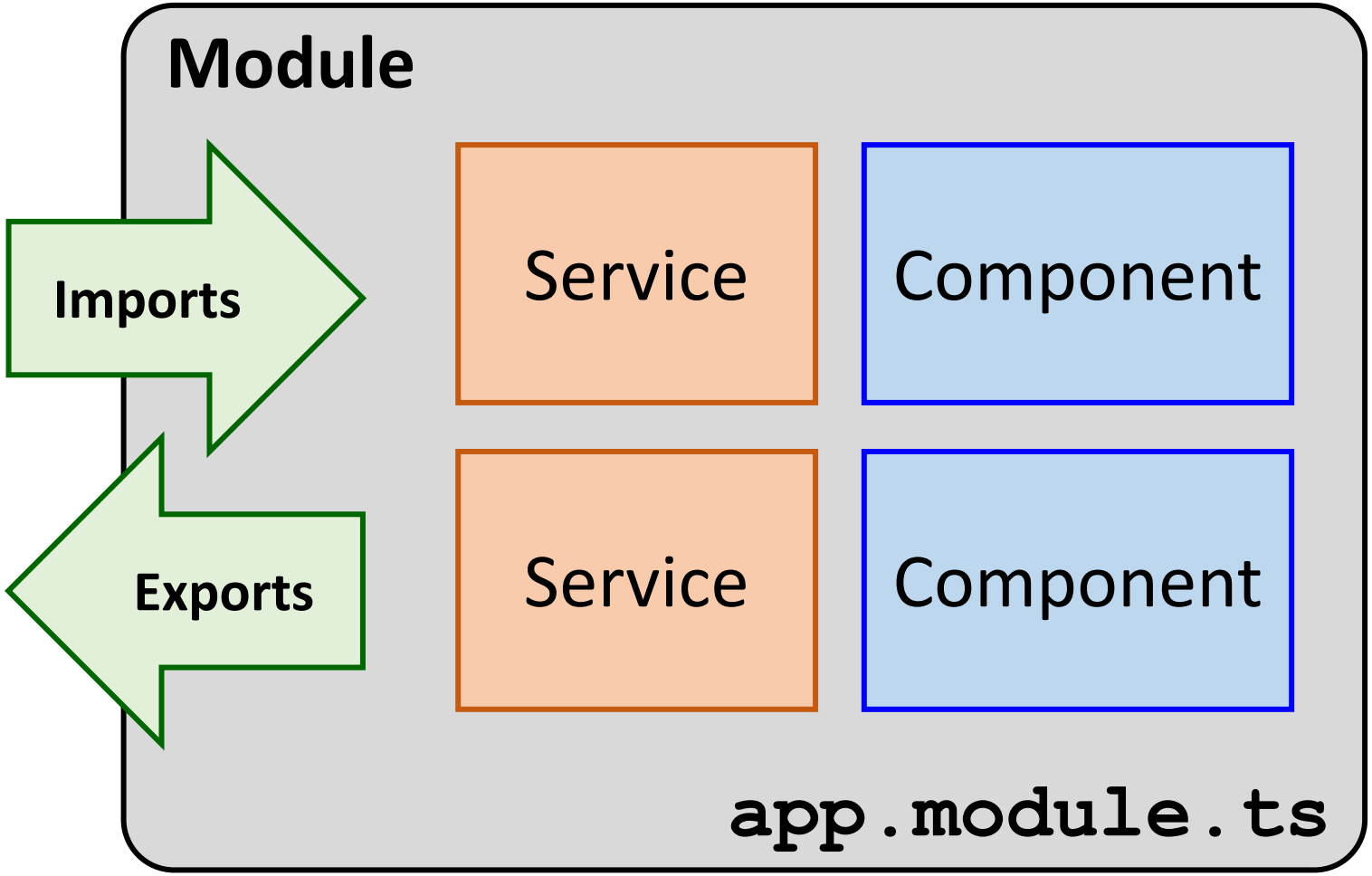
main.ts

```
platformBrowserDynamic()
  .bootstrapModule(AppModule)
```

Application Structure



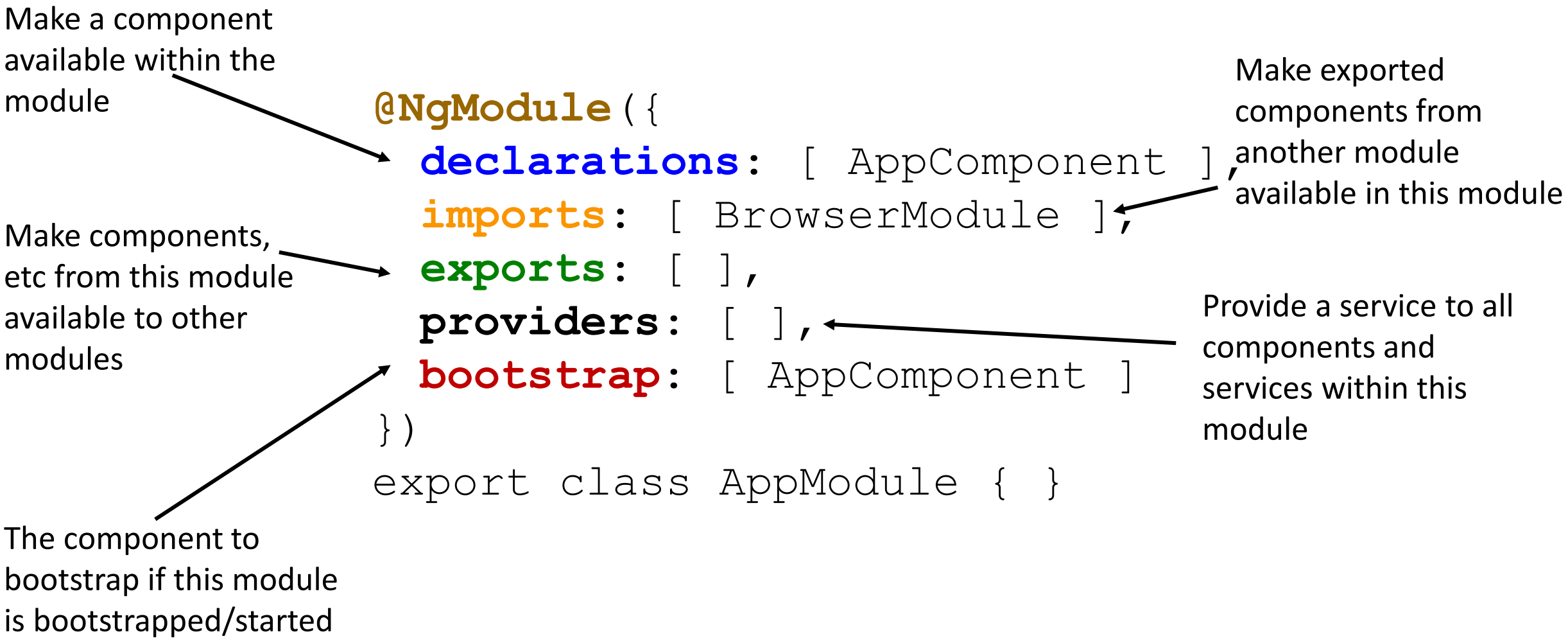
Module


Module
Imports
Exports
Service
Component
Service
Component
app.module.ts

Module

- A logical grouping of components, services, directives, pipes etc.
- Modules are opaque
  - The internals of a module is not visible to external unless a module chooses to export it
- Modules can use components from other modules by importing them
- `@NgModule` annotation is used to declare a class as module

Declaring a Module



Component

- Components are reusable software functionalities
- UI building blocks
  - Controls an area of the screen
  - Eg. a registration form
- A component is made up of
  - HTML fragment - structure
  - CSS - style
  - TypeScript class - behaviour
- Components are created with the `@Component` annotation

Generating a Component

```
ng g c components/cart --spec false --flat
```

Updates AppModule

Generates these

```
@NgModule({
  declarations: [CartComponent ],
})
export class AppModule {
```

app.module.ts

```
src/app/components
  cart.component.ts
  cart.component.html
  cart.component.css
```

Component class

cart.component.ts

```
@Component({
  selector: 'app-cart',
  templateUrl: './cart.component.html',
  styleUrls: [ './cart.component.css' ]
})
export class CartComponent {
```

Component

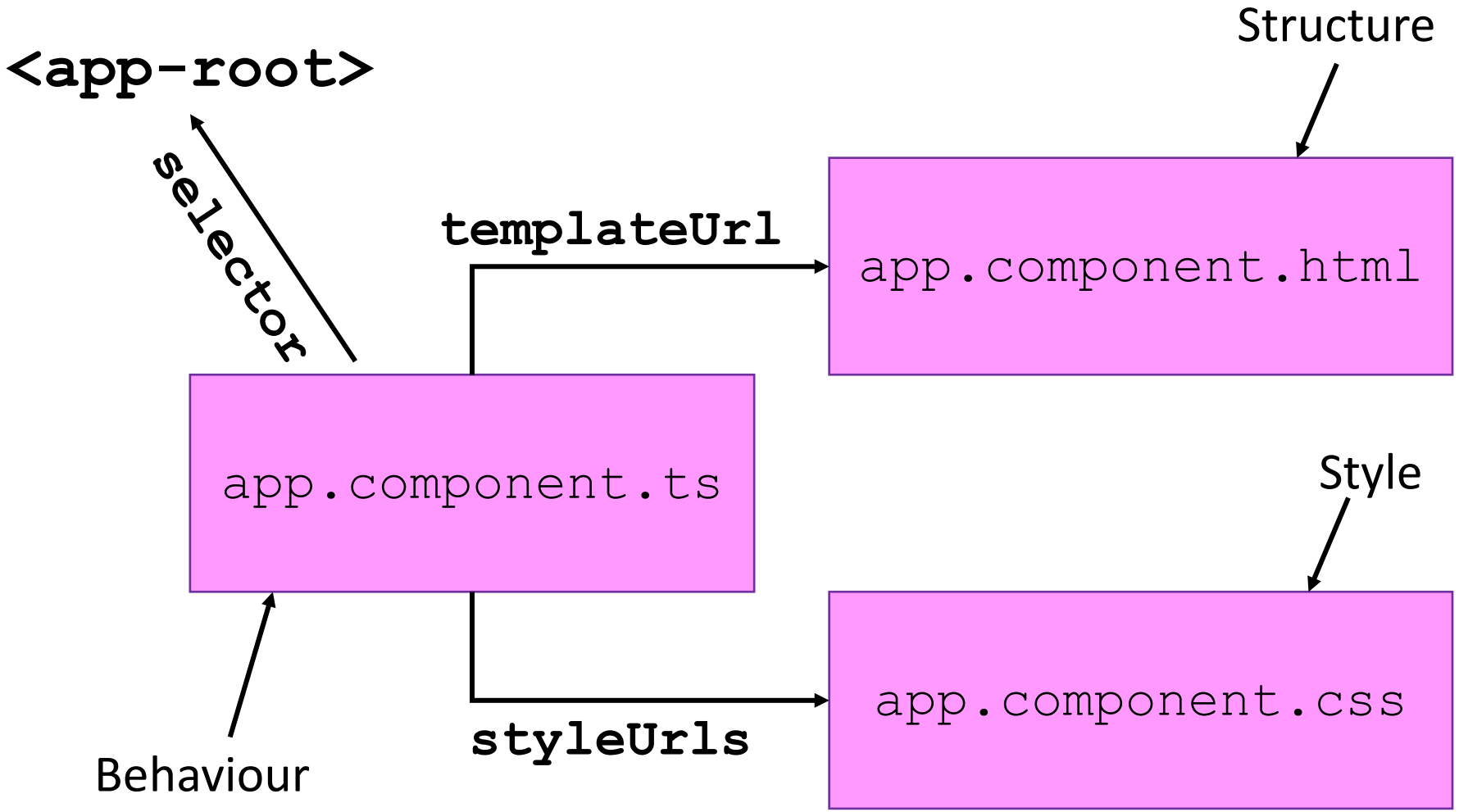
The ‘tag’ that instantiates this component

The HTML that defines the component’s structure

One or more CSS that defines the component’s style

```
@Component({
  selector: ‘app-root,
  templateUrl: ‘./app.component.html’,
  styleUrls: [
  ‘./app.component.css’
  ]
})
export class AppComponent {
  ...
}
```

Component


<app-root>
selector
templateUrl
app.component.html
Structure
app.component.ts
Style
app.component.css
styleUrls
Behaviour

Accessing Properties

- Properties/members in the component class can be accessible by its template
- Properties are displayed by `{{ }}`
- Changes in the properties are immediately reflected in the template

```
@Component({
  templateUrl:
    'app.component.html'
    ...
})
export class AppComponent {
  title = 'hello, world';
}
```

```
<h1> {{ title }} </h1>
```

Angular expression

Property Binding

- Any HTML attribute can be bound to the component's properties by surrounding the attribute name with [ ]

```
<input [value]="name">


<button type="button" [disabled]="isDisabled">
</button>


<p [style.font-size]="fontSize">
  {{ greetings }}
</p>
```

Event Binding

- HTML element generate events
  - Clicked, mouse hover, value changed, etc
  - See https://developer.mozilla.org/en-US/docs/Web/Events
- Bind HTML event with `( )` to a method/function in the component's class
  - Drop the on when binding to events
  - eg `onClick` becomes `(click)`
- Pass `$event` into the function to get the event object

Event Binding

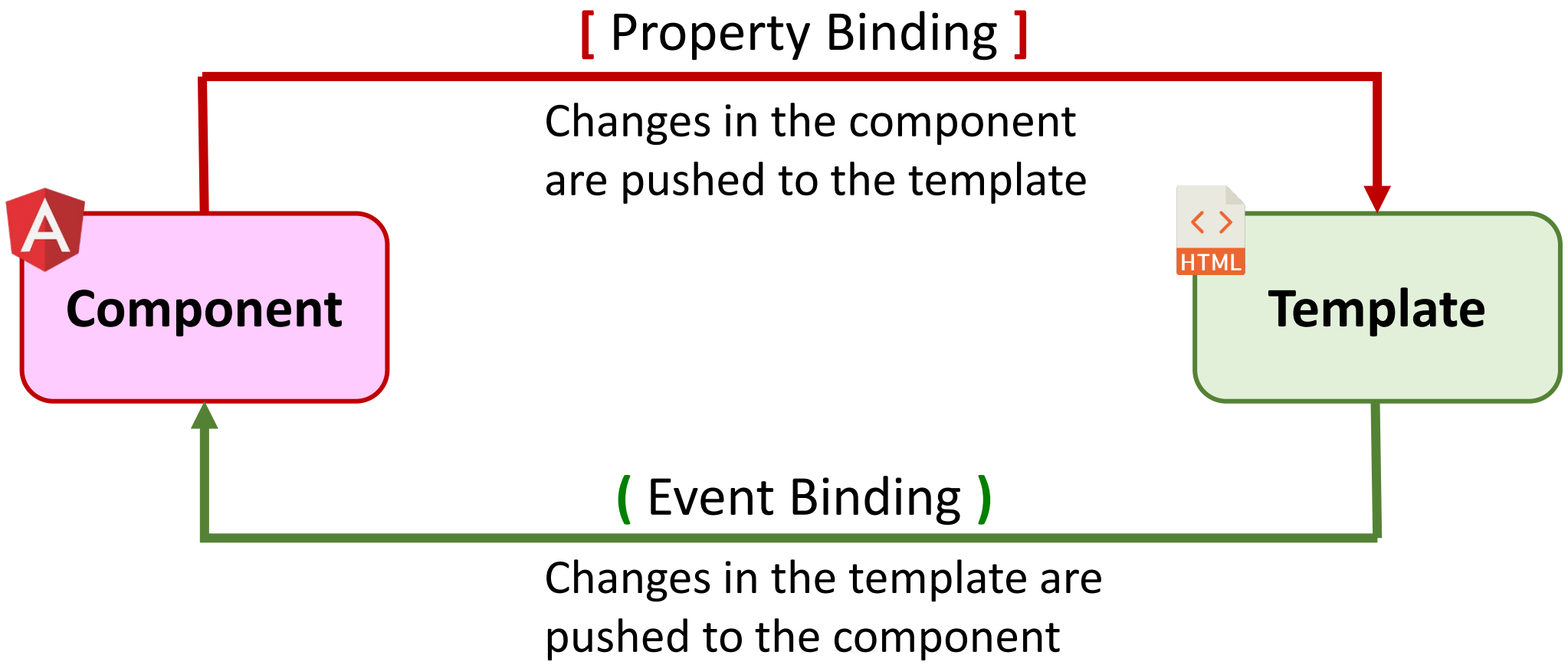
```
<p [style.font-size]="fontSize">
  {{ greetings }}
</p>
<input type="range"
  min="1" max="10" step="0.1"
  (change)="fontSizeChanged($event)">
```

This is DOM event object. See https://developer.mozilla.org/en-US/docs/Web/API/Event

Method is called whenever the value of the slider changes

```
export class AppComponent {
  greetings = 'hello world';
  fontSize = '1em';
  fontSizeChanged($event) {
    this.fontSize = `${$event.target.value}em`;
  }
}
```

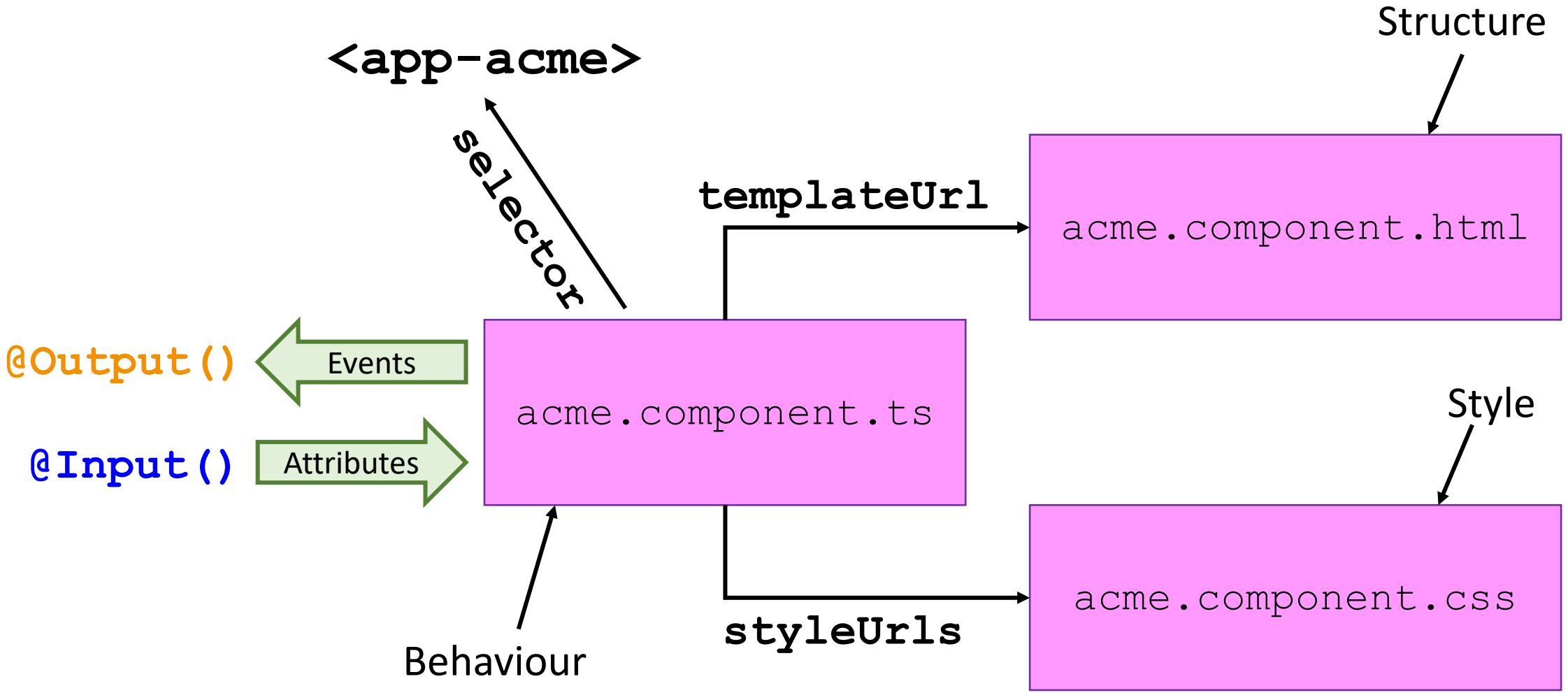
Property and Event Binding



Component

- Components internals are not accessible from the outside of the component
  - Eg accessed by other components
- Declare properties and events
  - To allow binding by other components
- Annotate class member with
  - `@Input()` for attribute
  - `@Output()` for event - `Subjct` type

Component


<app-acme>
selector
Structure
templateUrl
acme.component.html
@Output()
Events
@Input()
Attributes
acme.component.ts
Style
Behaviour
styleUrls
acme.component.css

Example - font-size.component.html

```
<div>
  <h2 [style.font-size]="fontSize">
    {{ message }}
  </h2>
</div>

<div>
  Font size:
  <input type="range" min="1" max="10" step="0.1"
      (change)="fontSizeChanged($event)">
</div>
```

Example - font-size.component.ts

```
@Component({
   selector: 'app-font-size',
   templateUrl: './font-size.component.html',
   styleUrls: [ './font-size.component.css' ]
})
export class FontSizeComponent {
   @Input() message: string;
   @Output() onFontSize = new Subject<number>();
   fontSize: string = '1em';

   fontSizeChanged($event) {
      const fontSize = parseInt($event.target.value);
      this.fontSize = `${fontSize}em`;
      this.onFontSize.next(fontSize);
   }
}
```

Define an externally accessible attribute call `message`

Define an event called `onFontSize`

The event object viz. the value that this event is firing

Fire the event with the latest font size

Example

font-size.component

```
<h2>{{ message }}</h2>
<input (change)="fontSizeChanged($event)">
```

```
export class FontSizeComponent {
  @Input() message;
  @Output() onFontSize = new Subject<number>();
  fontSizeChanged($event) {
    this.onFontSize.next($parseInt(event.target.value));
  }
```

app.component

```
<app-font-size
  [message]="title" (onFontSize)="sizeChanged($event)">
</app-font-size>
```

```
export class AppComponent {
  title = 'hello, world';
  sizeChanged(size) {
    console.log(`font size: ${size}`);
  }
```

@Output

- `@Output` annotation to declare an event to be fired by the component
- Type is `Subject` and an event object type as the type constraint

```
@Output() onEvent = new Subject<string>();
```

- To fire an event

```
this.onEvent.next('hello');
```

Directives

- Angular's way of extending HTML capabilities
  - Eg. conditionally apply CSS to HTML element
- Directives typically starts with `ng`
  - Eg. `ngFor`, `ngIf`, `ngClass`, etc.
- Two types of directives
  - Non structural - enhances an element
  - Structural - add or remves HTML element; prefixed with a `*`
    - eg `*ngIf`

ngClass - Conditional Styling

```
<input type="text"
    [disabled]="isDisabled"
    [ngClass]=
      "{ 'grey-border': isDisabled, 'blue-border': !isDisabled }">
```

Conditionally added these classes based on the boolean variable

*ngIf - Conditional Display

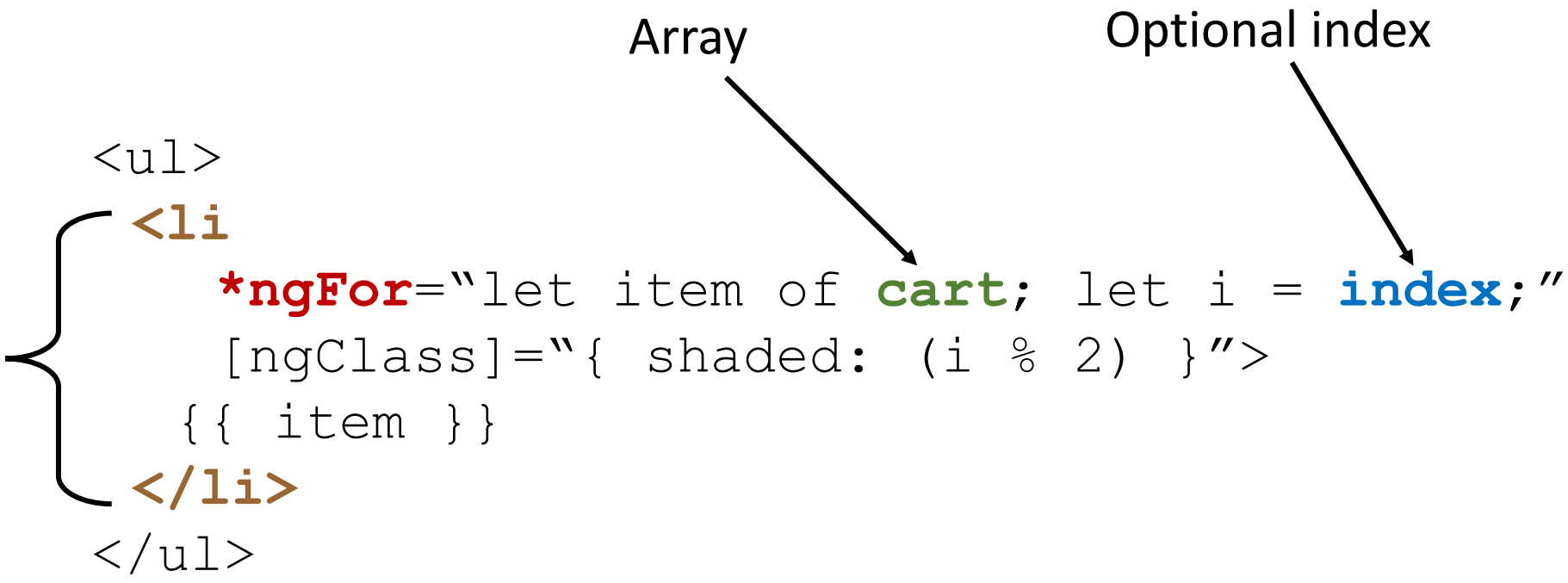
```
<div *ngIf="cart.length <= 0">
  Your cart is empty
</div>

<div *ngIf="cart.length > 0">
  Your cart has {{ cart.length }} item(s)
</div>
```

Conditionally add or remove these elements

*ngFor - Loops

Generate the element that is annotated with *ngFor



Generate HTML element from the contents of an array

Pipes

- Pipes are used in templates to transform values, typically for display
  - Eg. formatting number, date or currency, converting strings to upper case, etc.

```
<div *ngFor="let n of names">
  {{ n | titlecase }}
</div>
```

```
<div *ngFor="let n of names | titlecase">
  {{ n }}
</div>
```

Values to transform

Pipe character

pipe

- Pipes can be combined

Pipe's parameters

```
{{ value | number:'1.1-3' | currency:'SGD':'symbol-narrow' }}
```

10.051 -> SG$10.05

Examples - Pipe

fred to FRED

```
{{ value | uppercase }}
```

heLLo world to Hello World

```
{{ value | titlecase }}
```

10 to 10%

```
{{ value | percent }}
```

3.14159265 to 3.141

```
{{ value | number:'1.1-3' }}
```

new Date() to 04:18 PM GMT+08:00

```
{{ date | date:'hh:mm aa zzzz'
```

https://angular.io/api/common#pipes

['a', 'b', 'c', 'd', 'e'] to ['b', 'c']

```
*ngFor="let let of array | slice:1:3"
```

{ name: 'fred', ... } to JSON.stringify({ name: 'fred', ... })

```
{{ customer | json }}
```

{ name: 'fred', ... } to [{key: 'name', value: 'fred'}, ...]

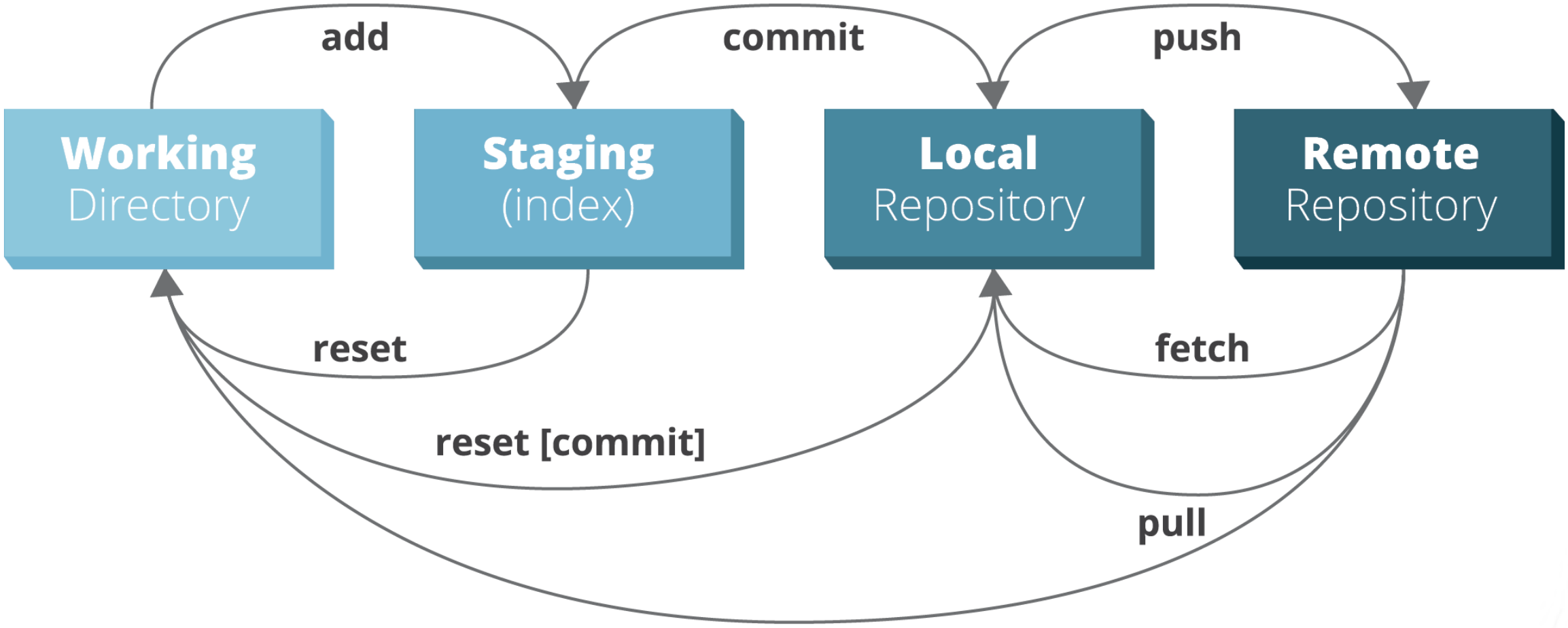
```
<div *ngFor="let c of customers | keyvalue">
  {{ c.key }} - {{ c.value }}
</div>
```

male to him

```
{{ gender | i18nselect:{'male': 'him',
'female': 'her', 'other': 'them' } }}
```

Unused

Git Workflow



Git Commands

- Initialize a directory as a Git repository
  - Not required if project is generated by Angular CLI

```
git init
```

- Add files to the staging area

```
git add .
```

- Commit files to the local repository

```
git commit -m 'commit message'
```

Git Commands

- Push local repository to remote repository

```
git push -u origin master
```

- Adding a remote repository

```
git remote add origin <git repo URL>
```

- Syncing local repo with remote

```
git pull origin master
```