Understanding and Configuring Dependency Injection

Brice Wilson

@brice_wilson www.BriceWilson.net

Overview

What is dependency injection?

Providers

Injectors

Deciding where to provide services

Creating a Dependency

```
export class DashboardComponent {

}
```

Creating a Dependency

```
export class DashboardComponent {

  dataService: DataService;





}
```

Creating a Dependency

```
export class DashboardComponent {
  dataService: DataService;
  constructor() {
    this.dataService = new DataService();
  }
}
```

Creating a Dependency

```
export class DashboardComponent {
  dataService: DataService;
  constructor() {
    this.dataService = new DataService();
  }
}
```

Injecting a Dependency

```
export class DashboardComponent {



}
```

Injecting a Dependency

```
export class DashboardComponent {
  constructor(private dataService: DataService) {
  }
}
```

Injecting a Dependency

```
export class DashboardComponent {
  constructor(private dataService: DataService) {
  }
}
```

Injecting a Dependency

```
export class DashboardComponent {
  constructor(private dataService: DataService) {
  }
}
```

Why Is Dependency Injection Important?

Loosely coupled code

More flexible code

Easier to test

Providers

A provider tells an injector *how to create the service*.

Angular Documentation

Configure an injector with a service provider

https://angular.io/guide/dependency-injection#configure-an-injector-with-a-service-provider

Provider Tokens

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [DataService]
})
export class AppModule { }
```

Provider Tokens

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [DataService]
})
export class AppModule { }
```

Provider Tokens

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [DataService]
})
export class AppModule { }
```

Token

Provider Tokens

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [DataService]
})
export class AppModule { }
```

Token

Token

```
export class DashboardComponent {
  constructor(private dataService: DataService) { }
}
```

Provider Recipes

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [



  ]
})
export class AppModule { }
```

Provider Recipes

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [
    DataService,
    { provide: LoggerService, useClass: LoggerService }
  ]
})
export class AppModule { }
```

Provider Recipes

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [
    DataService,
    { provide: LoggerService, useClass: LoggerService }
  ]
})
export class AppModule { }
```

Token

Recipe

Provider Recipes

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [
    DataService,
    { provide: LoggerService, useClass: LoggerService }
  ]
})
export class AppModule { }
```

Provider Recipes

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [
    DataService,
    { provide: LoggerService, useClass: LoggerService }
  ]
})
export class AppModule { }
```

Provider Recipes

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [
    DataService,
    { provide: LoggerService, useClass: LoggerService }
  ]
})
export class AppModule { }
```

Provider Recipes

```
@NgModule({
  declarations: [AppComponent,DashboardComponent],
  bootstrap: [AppComponent],
  providers: [
    DataService,
    { provide: LoggerService, useClass: LoggerService }
  ]
})
export class AppModule { }
```

Demo

Multiple ways to provide services

Injectors

The Roles of Injectors

- Deliver provided services when they're requested via constructor injection
- Maintain a single instance of each service provided
- Determine what to inject based on emitted metadata
- Delegate injection to parent injectors if necessary

The Importance of Metadata

The Importance of Metadata

Provides information about parameters to injectors

```
{
  "compilerOptions": {
    "outDir": "./dist/out-tsc",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5"
  }
}
```

The Importance of Metadata

Provides information about parameters to injectors

Enabled with the “emitDecoratorMetadata” compiler option

```
{
  "compilerOptions": {
    "outDir": "./dist/out-tsc",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5"
  }
}
```

The Importance of Metadata

Provides information about parameters to injectors

Enabled with the “emitDecoratorMetadata” compiler option

```
{
  "compilerOptions": {
    "outDir": "./dist/out-tsc",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5"
  }
}
```

The Importance of Metadata

Provides information about parameters to injectors

Enabled with the “emitDecoratorMetadata” compiler option

```
{
  "compilerOptions": {
    "outDir": "./dist/out-tsc",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5"
  }
}
```

The Importance of Metadata

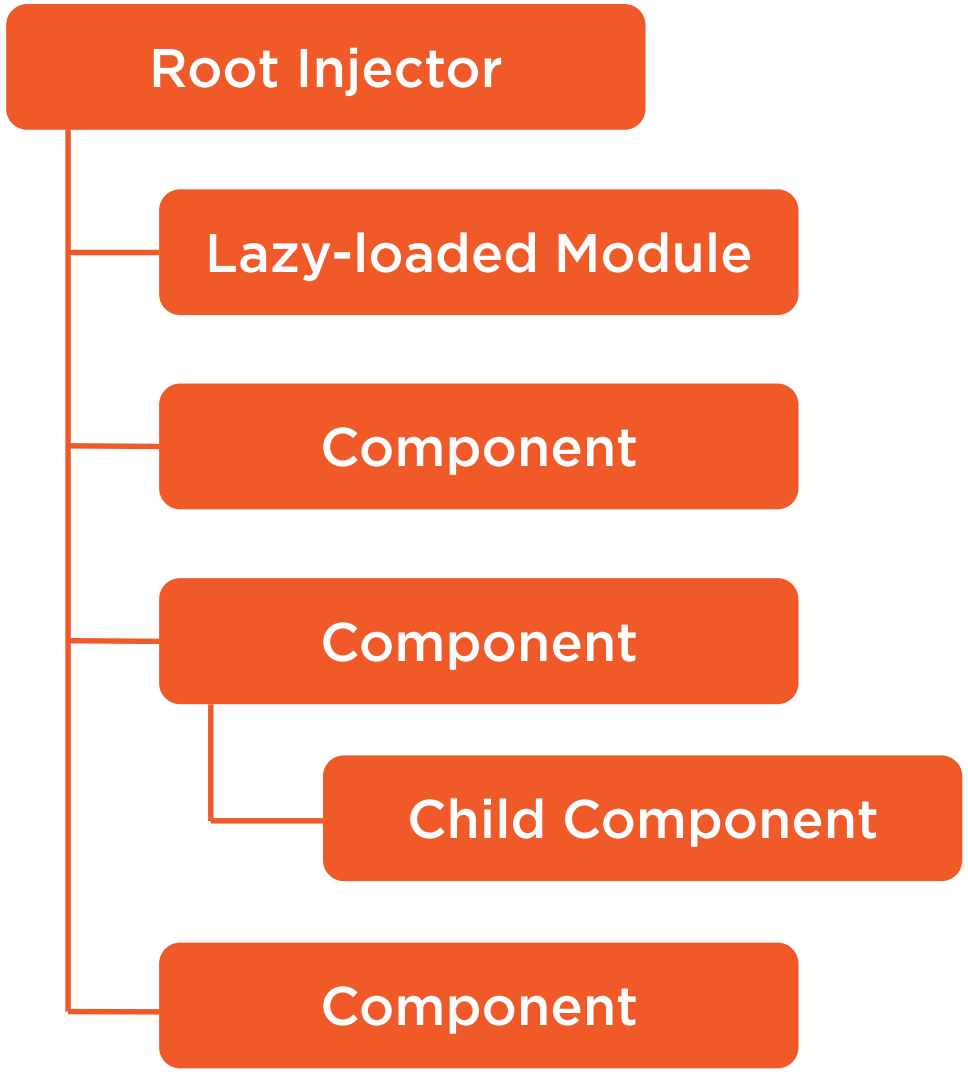
Provides information about parameters to injectors

Enabled with the “emitDecoratorMetadata” compiler option

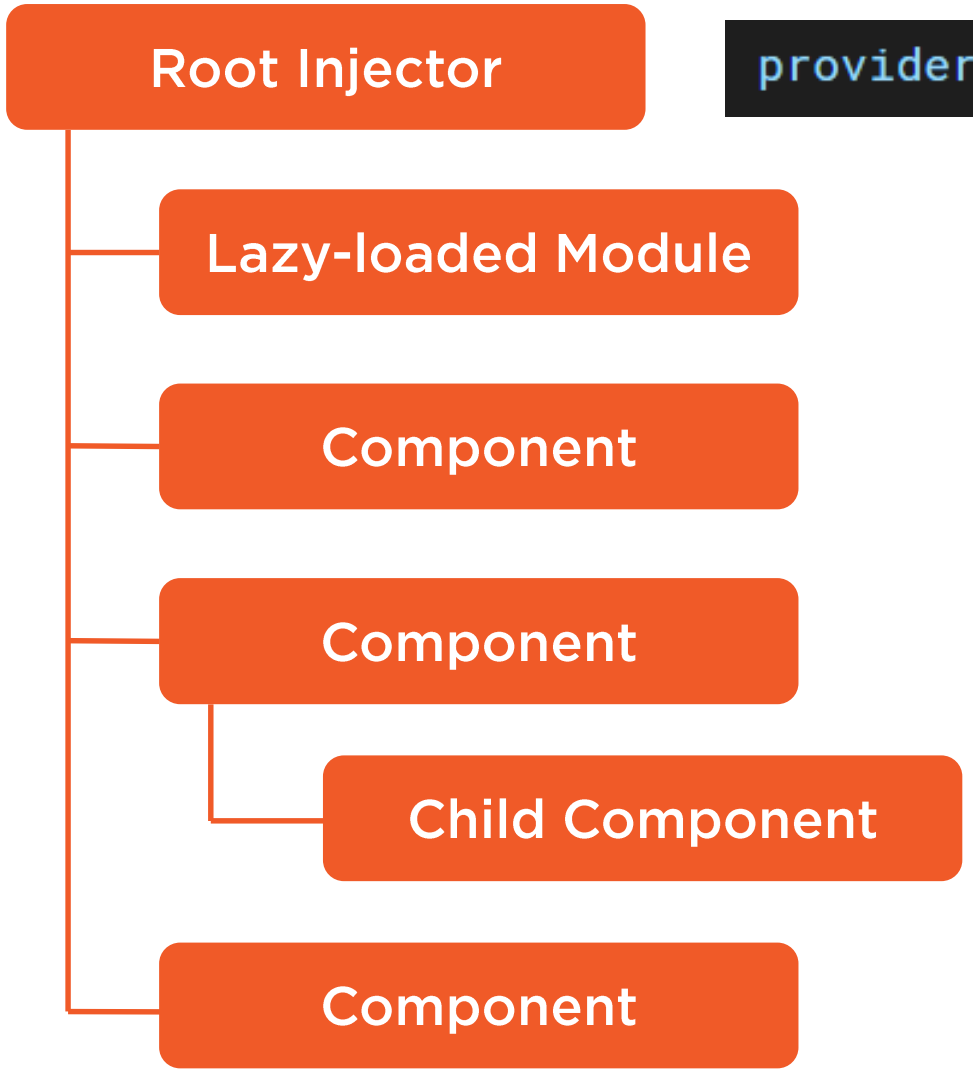
@Injectable() and @Component() decorator added to output metadata

Hierarchical Injectors

Hierarchical Injectors

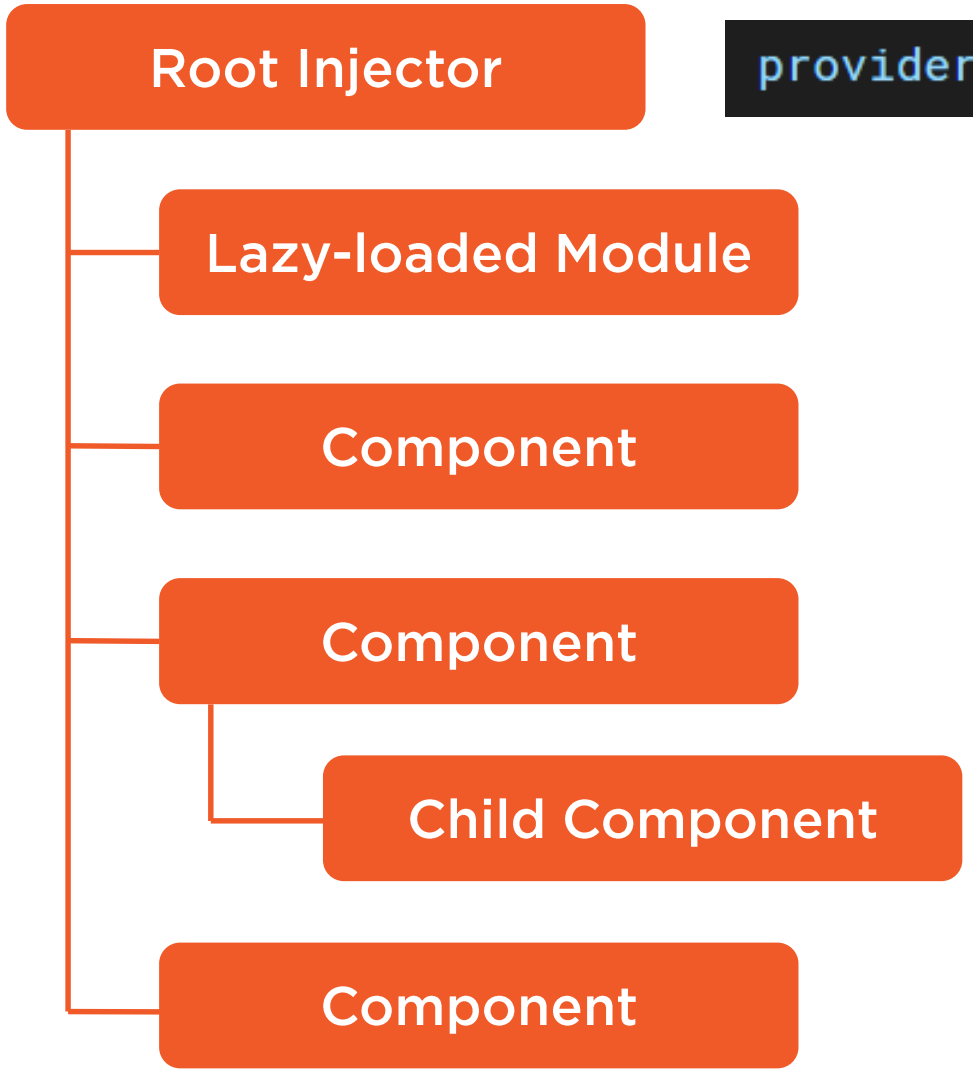


Hierarchical Injectors




```
providers: [LoggerService]
```

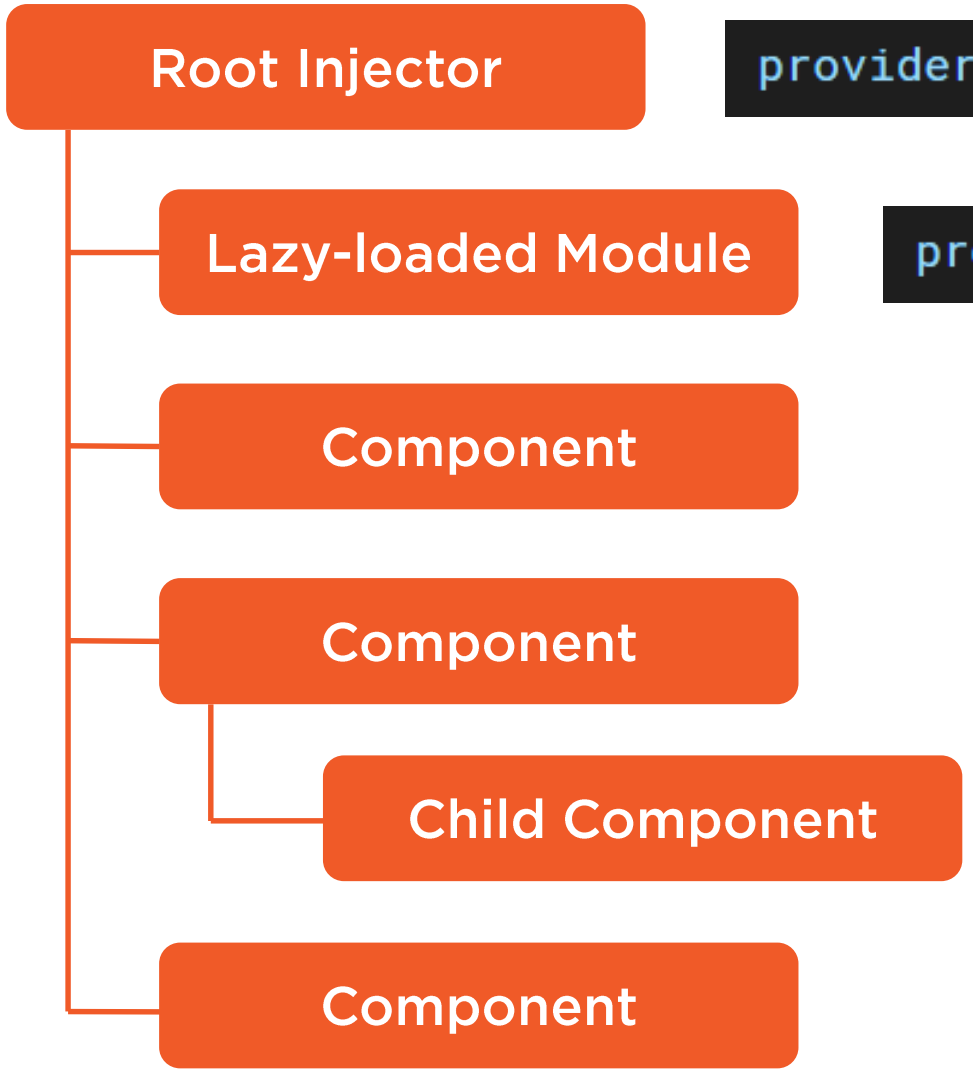
Hierarchical Injectors




```
providers: [LoggerService]
```

```
@Injectable({
  providedIn: 'root'
})
```

Hierarchical Injectors

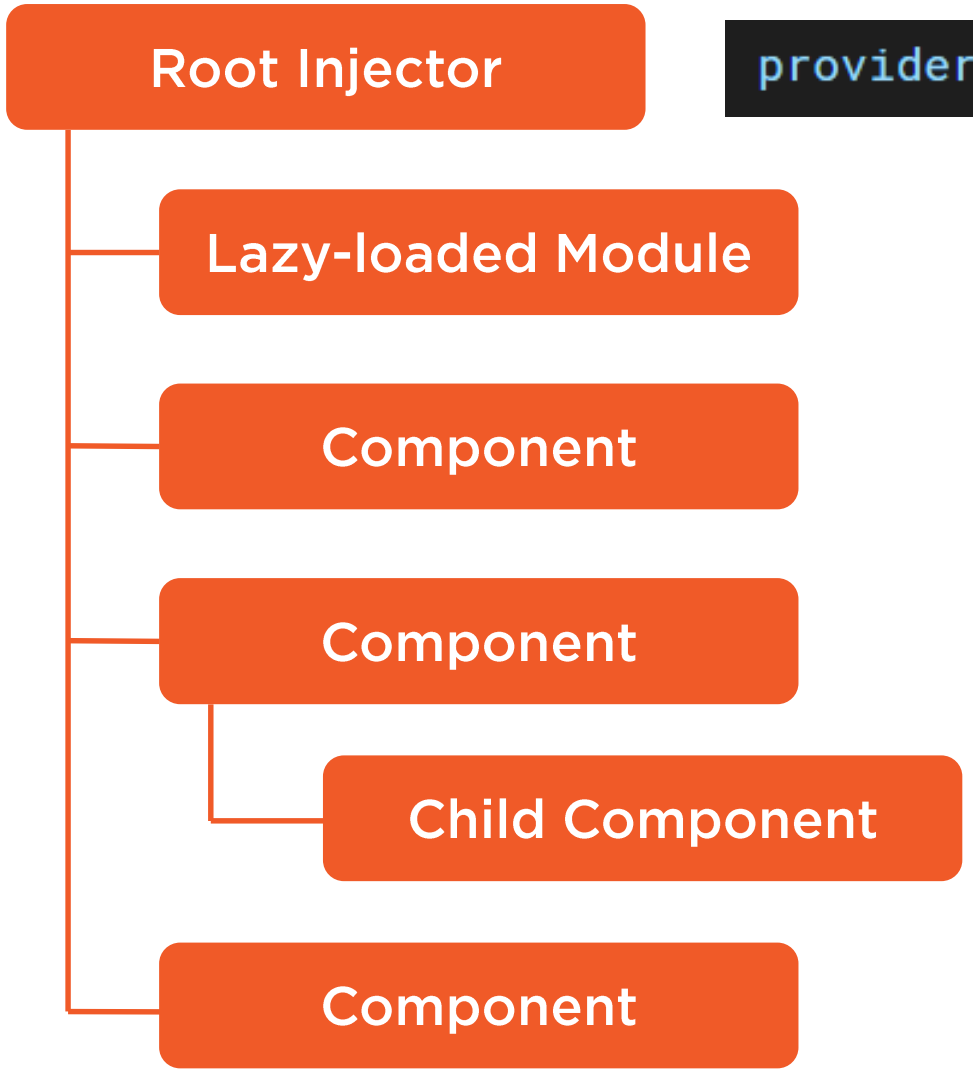



```
providers: [LoggerService]
```

```
@Injectable({
  providedIn: 'root'
})
```

```
providers: [LoggerService]
```

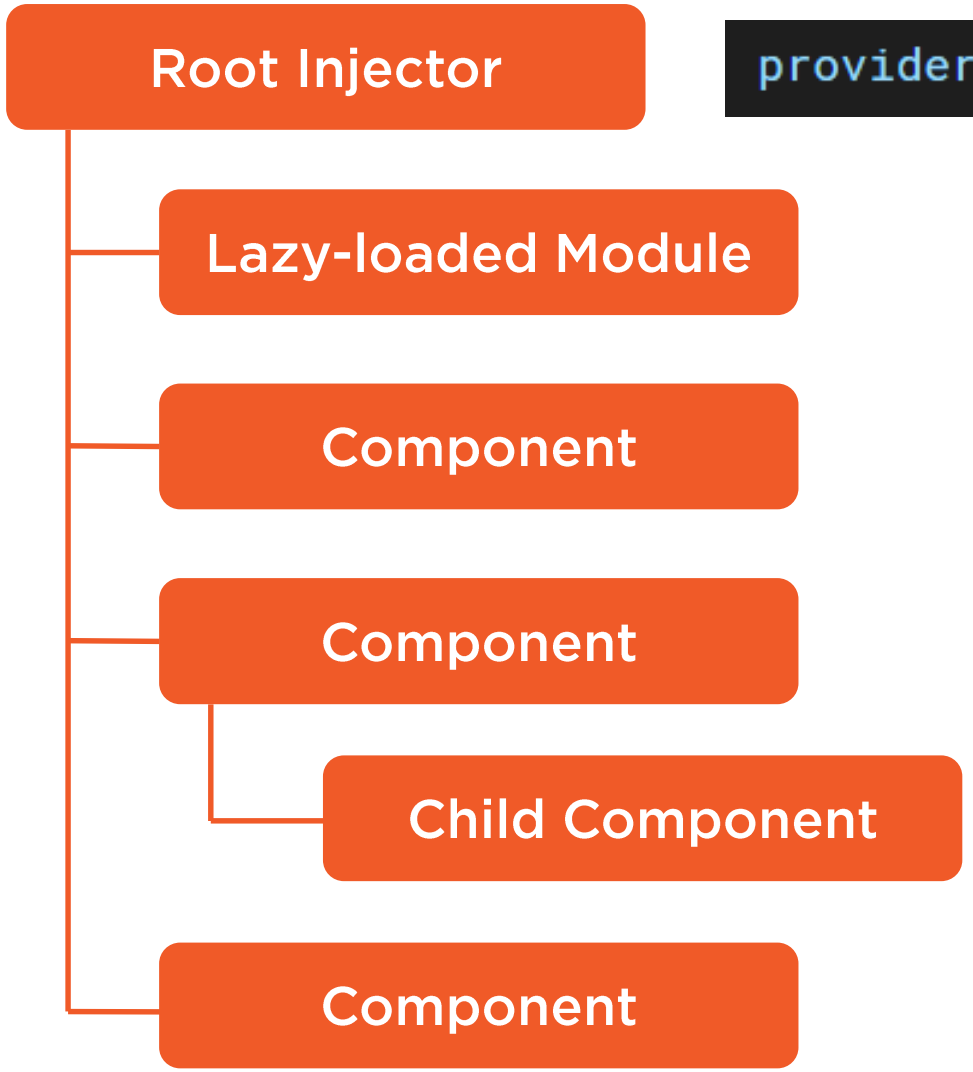
Hierarchical Injectors




```
providers: [LoggerService]
```

```
@Injectable({
  providedIn: 'root'
})
```

Hierarchical Injectors

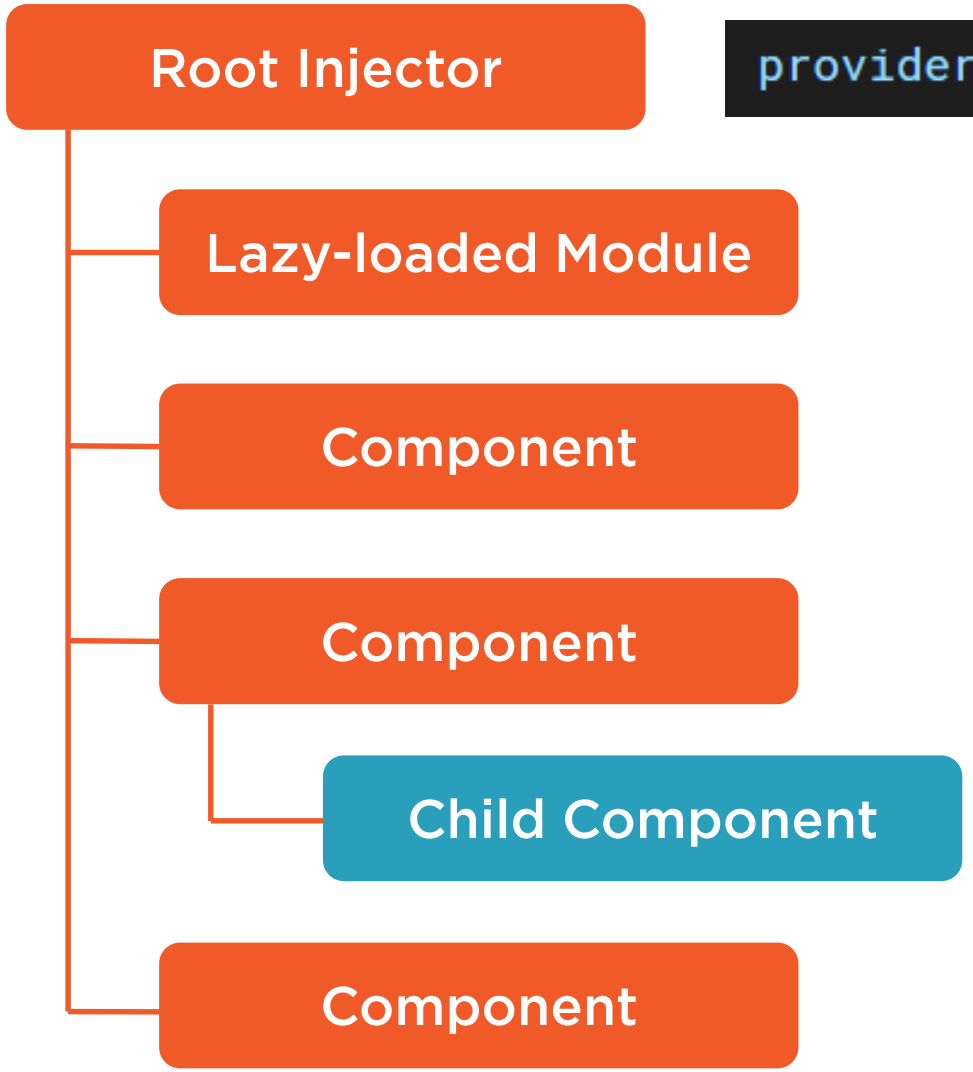



```
providers: [LoggerService]
```

```
@Injectable({
  providedIn: 'root'
})
```

```
constructor(private loggerService: LoggerService) { }
```

Hierarchical Injectors

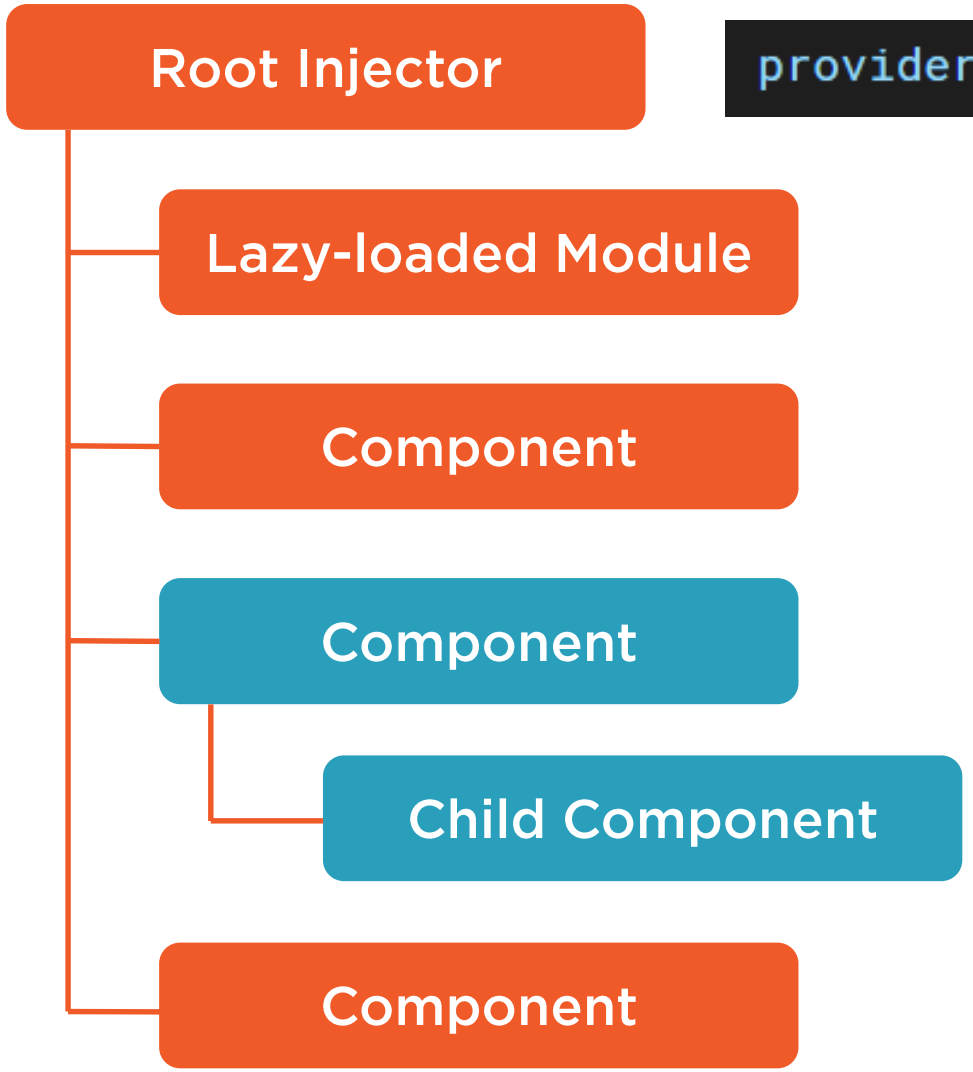



```
providers: [LoggerService]
```

```
@Injectable({
  providedIn: 'root'
})
```

```
constructor(private loggerService: LoggerService) { }
```

Hierarchical Injectors

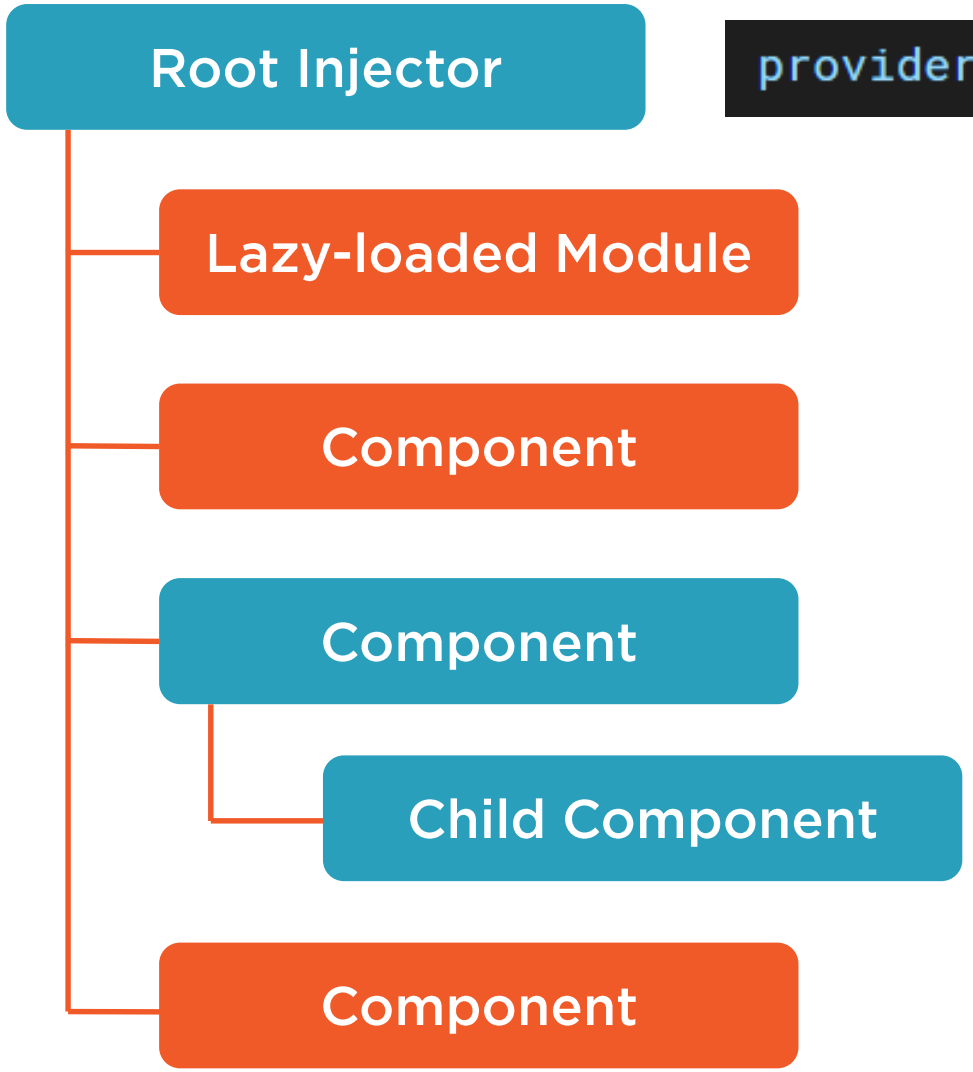



```
providers: [LoggerService]
```

```
@Injectable({
  providedIn: 'root'
})
```

```
constructor(private loggerService: LoggerService) { }
```

Hierarchical Injectors

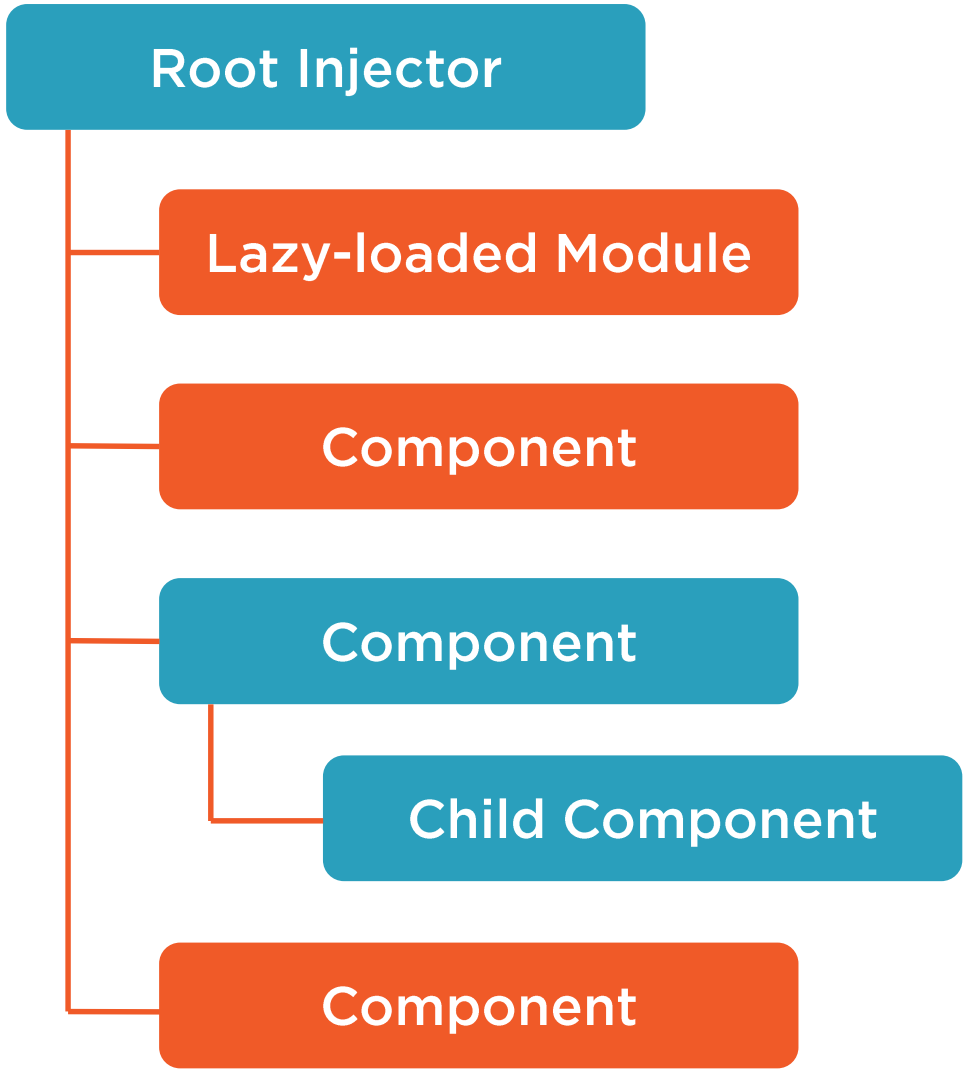



```
providers: [LoggerService]
```

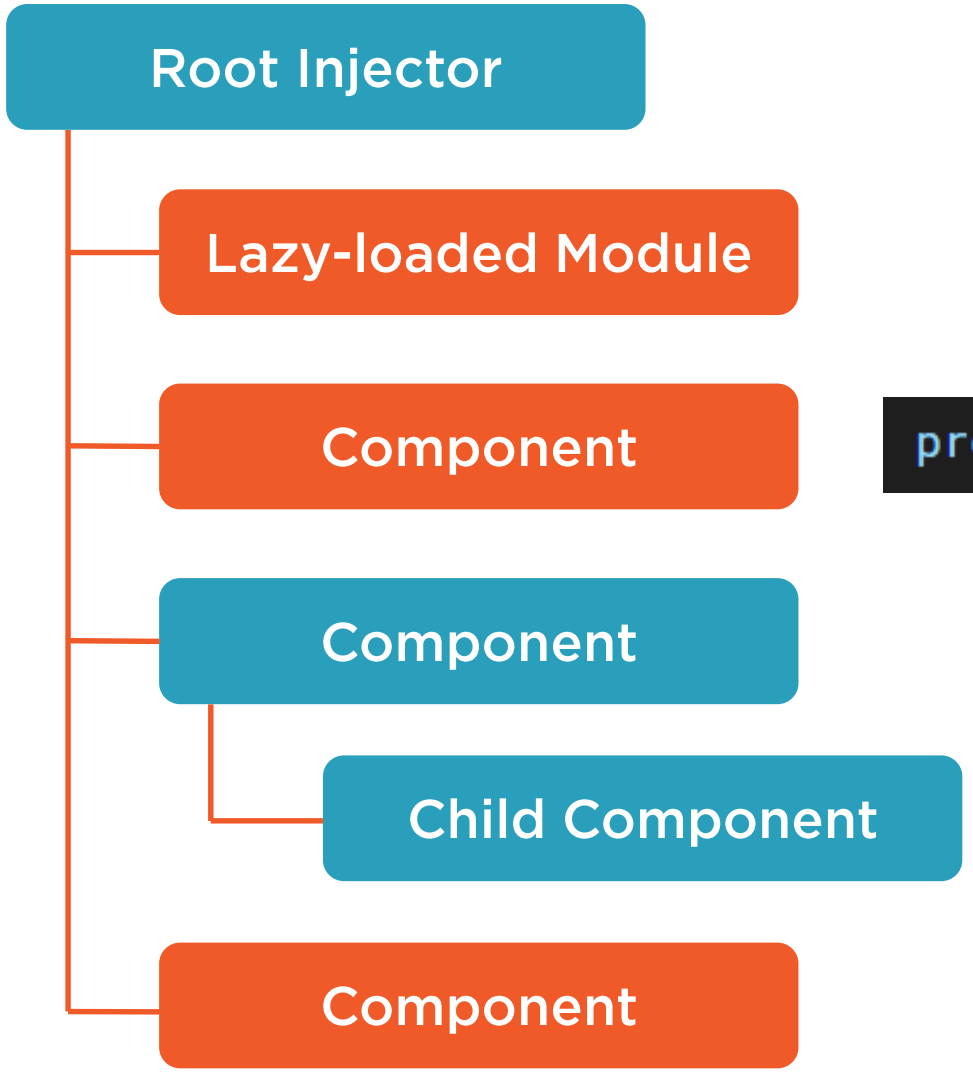
```
@Injectable({
  providedIn: 'root'
})
```

```
constructor(private loggerService: LoggerService) { }
```

Hierarchical Injectors

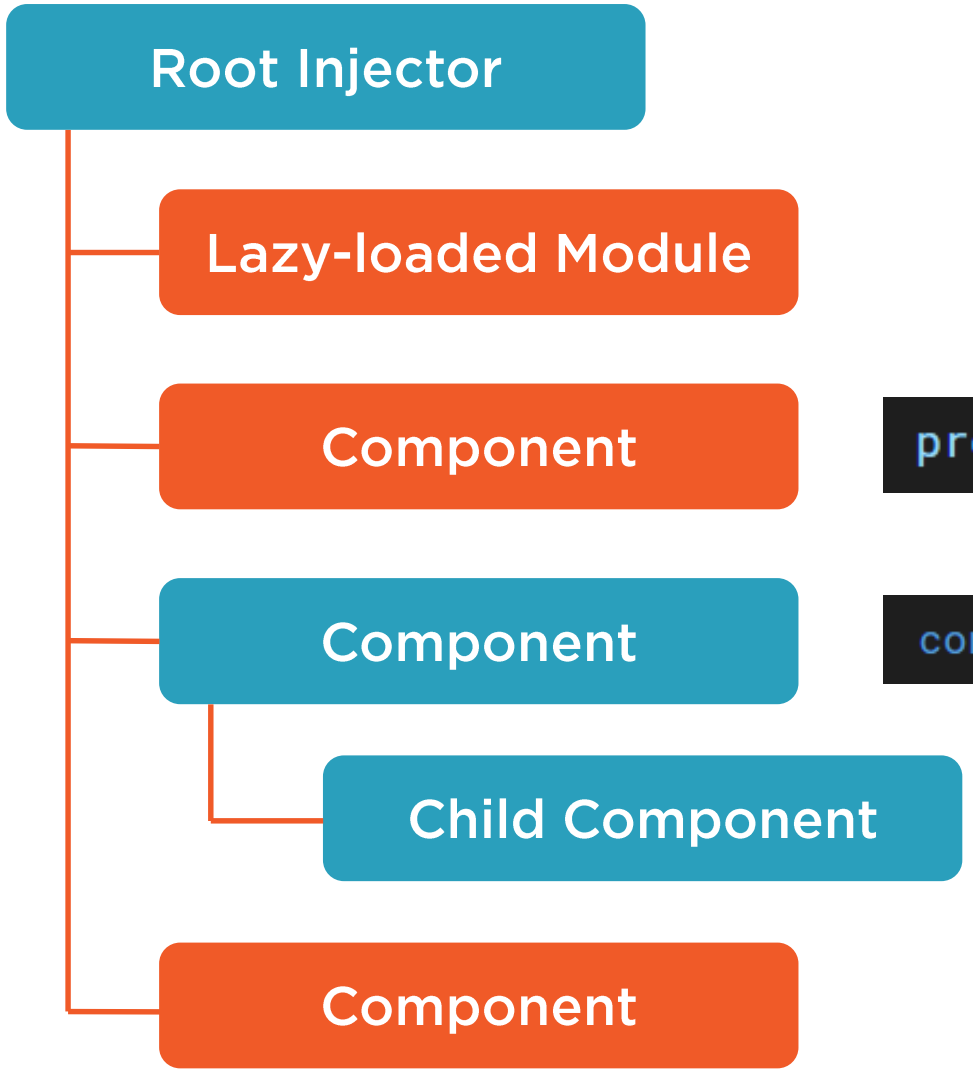


Hierarchical Injectors




```
providers: [LoggerService]
```

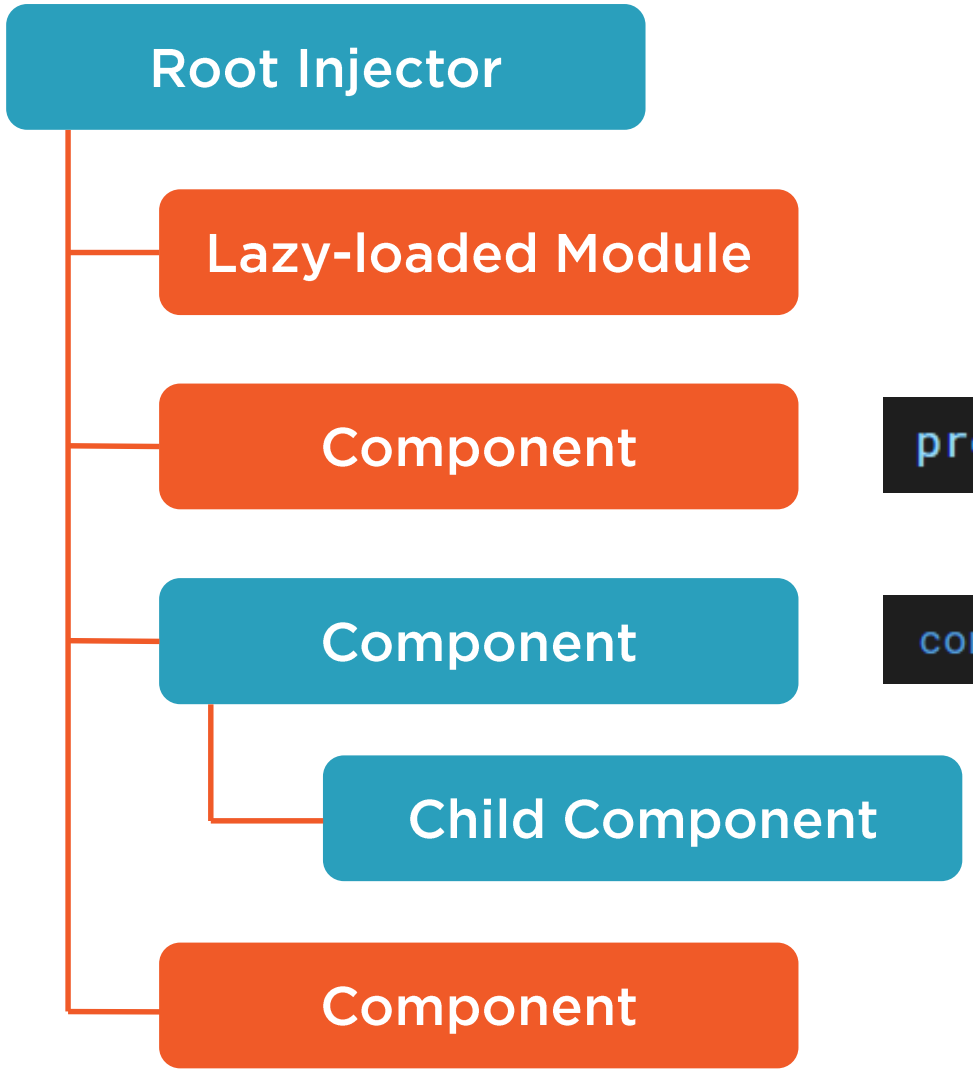
Hierarchical Injectors




```
providers: [LoggerService]
```

```
constructor(private loggerService: LoggerService) { }
```

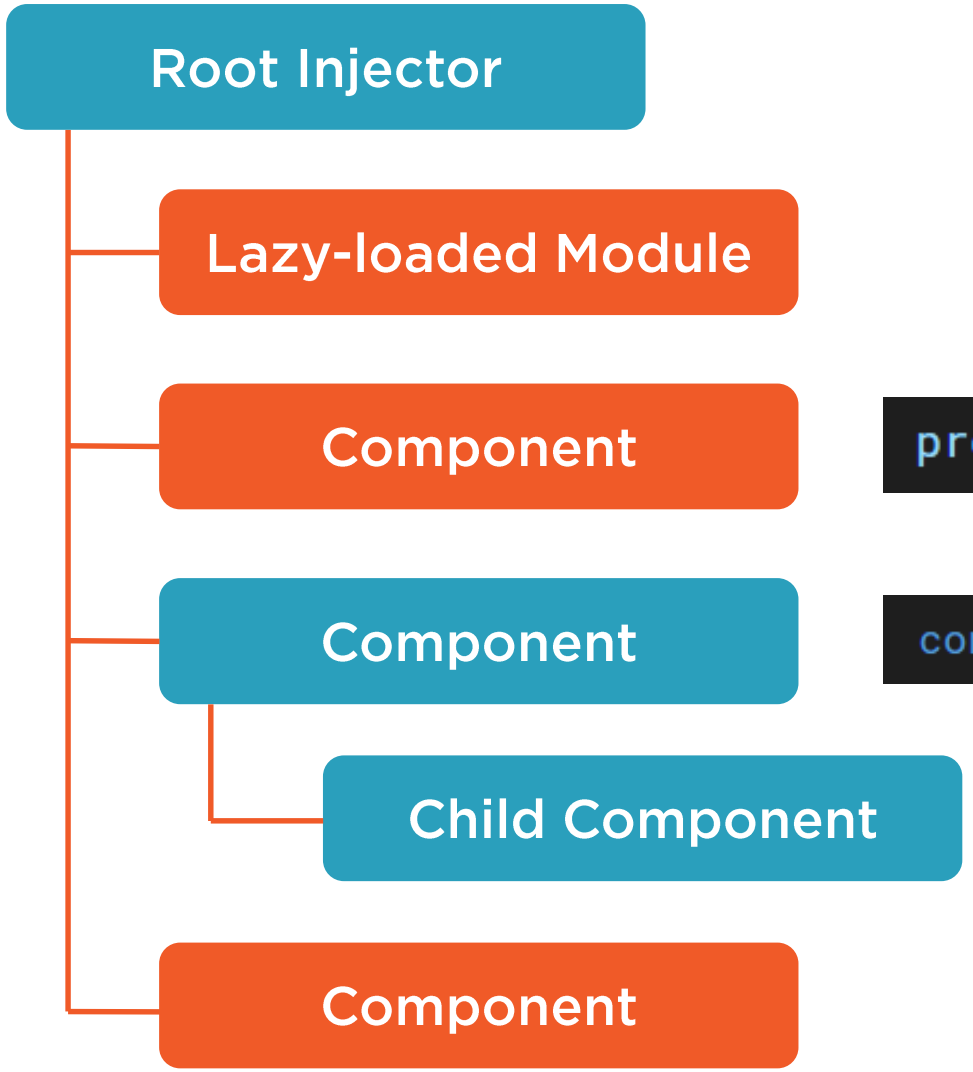
Hierarchical Injectors




```
providers: [LoggerService]
```

```
constructor(private loggerService: LoggerService) { }
```

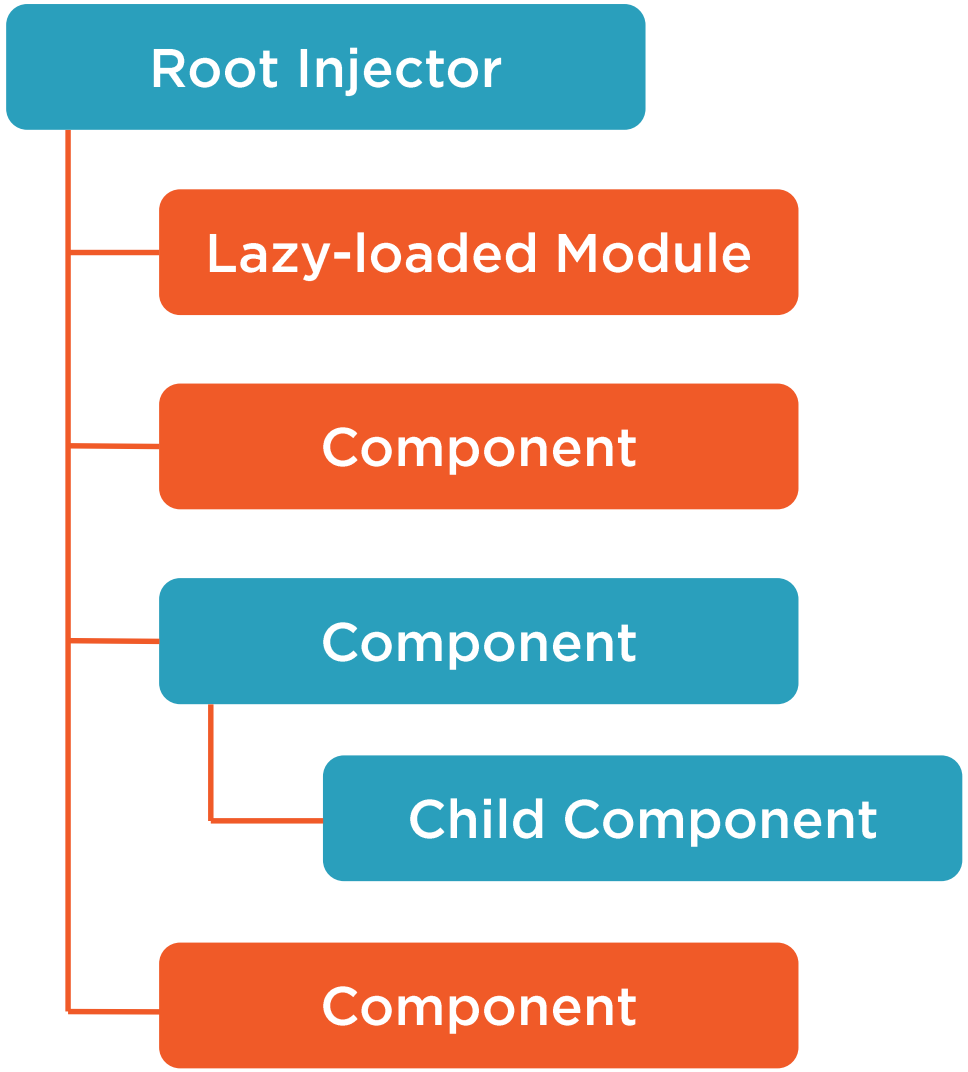
Hierarchical Injectors



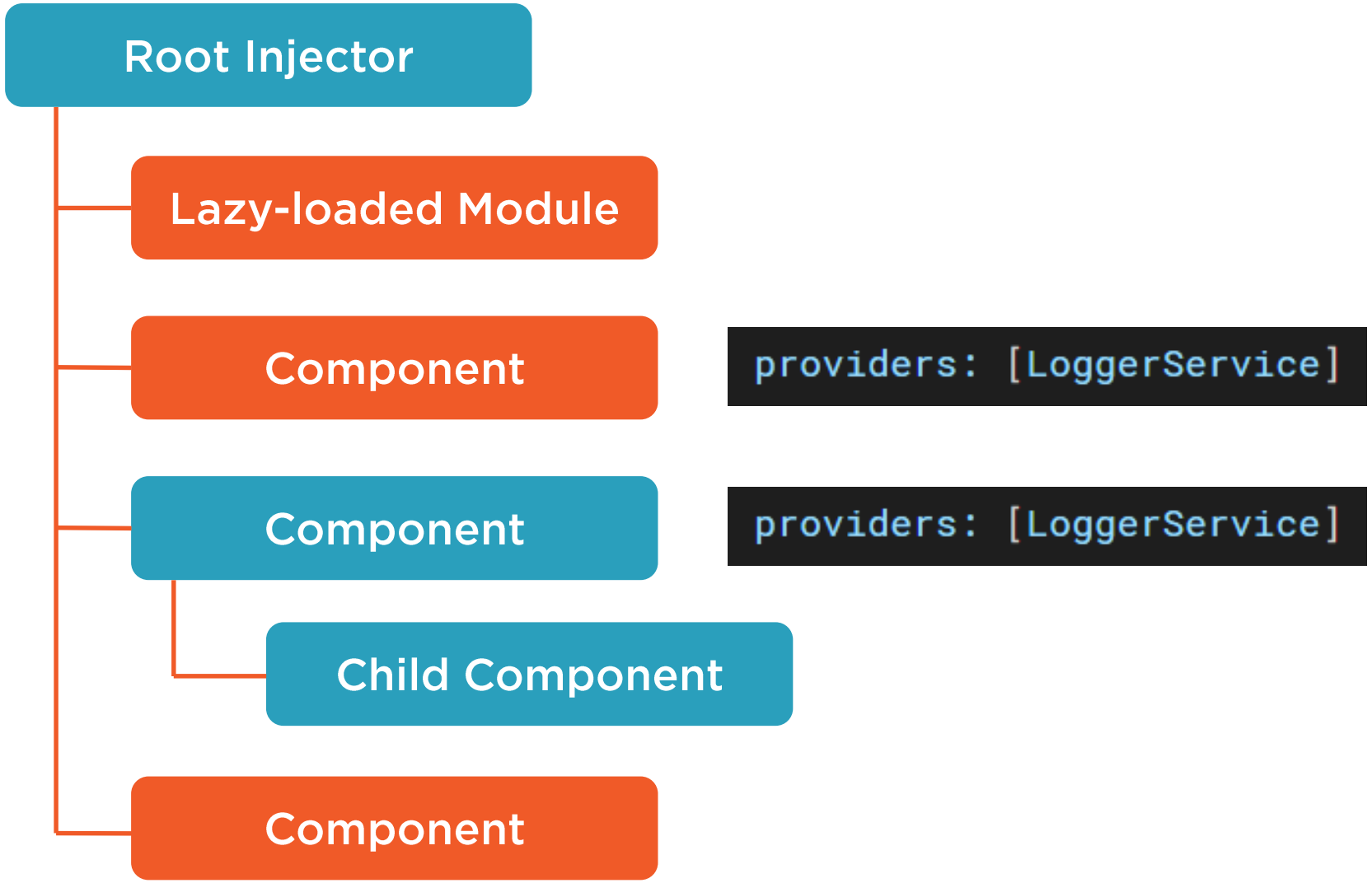

```
providers: [LoggerService]
```

```
constructor(private loggerService: LoggerService) { }
```

Hierarchical Injectors



Hierarchical Injectors



Demo

Hierarchical injectors

Deciding Where to Provide Services

- Provide in the root injector if needed everywhere
- Provide at the root AppModule rather the root AppComponent
- Provide component-specific services directly to component
- Consider creating a core module

Demo

Providing feature services

Demo

Creating a core module

Summary

- Providers
- Injectors
- Recipes for providers
- Injector hierarchy
- Deliver the right service at the right time