ANGULAR MODULARITY
(NgModules)

Naveen Pete

Saturday, January 20, 2018

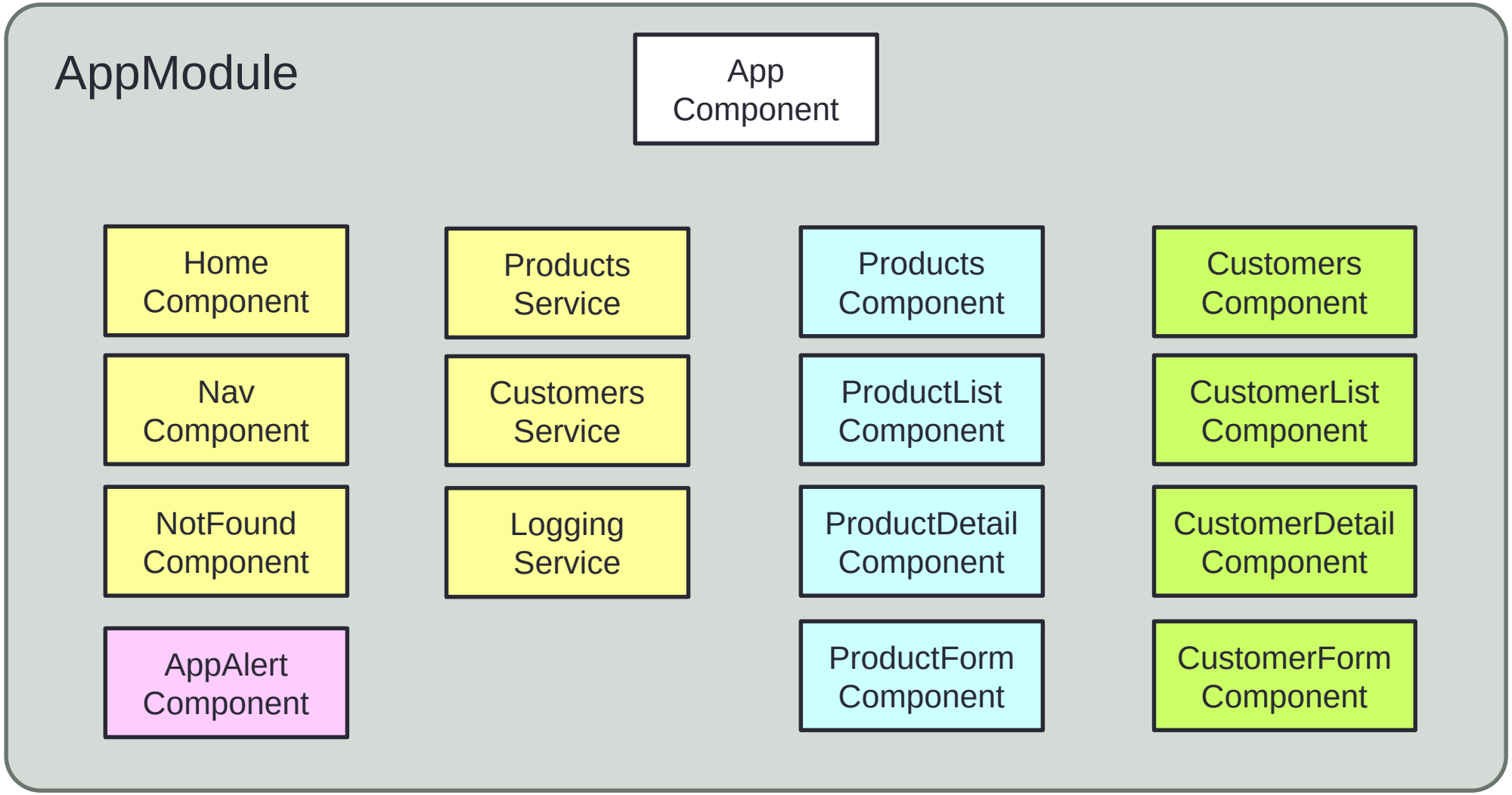
Agenda

- Modules
- A Typical Angular App – Store App
- Angular Modules
- @NgModule Decorator
- Routing Module
- Feature Module
- Sharing Module
- Organizing Core Components of an App
- Points to Note
- Q & A

Modules

- Great way to organize an app
- Allow an app to extend with capabilities from external libraries
- Structure code in a better way
- Make the app code more maintainable
- Promote reuse of code
- Increase performance of the app
- Decrease file size

A Typical Angular App – Store App



Angular Modules

- By default, all Angular apps contain one module – AppModule

- An Angular module
  - consolidates components, directives, and pipes into cohesive blocks of functionality
    - Focused on a feature, app business domain, workflow or common collection of utilities
  - bundles related functionalities that can be imported and reused
  - can be eagerly loaded or lazy loaded asynchronously
  - is a simple TypeScript class that is decorated using @NgModule decorator

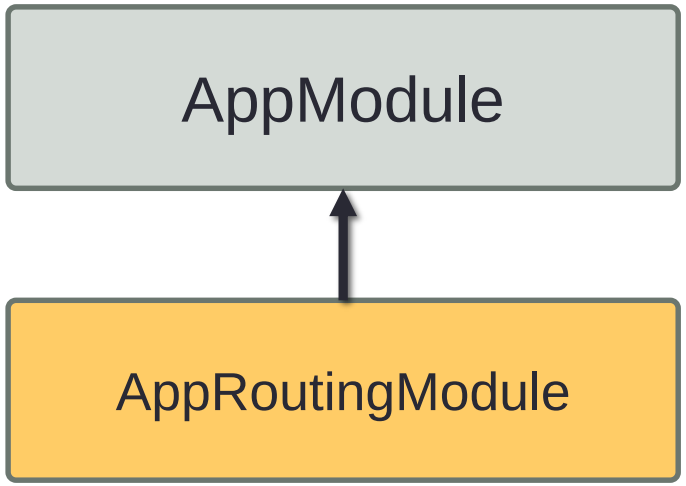
```
// To create a module using Angular CLI
C:\store-app> ng generate module customers

// To create a component within Customers module
C:\store-app> ng generate component customers/customer-dashboard
```

@NgModule Decorator

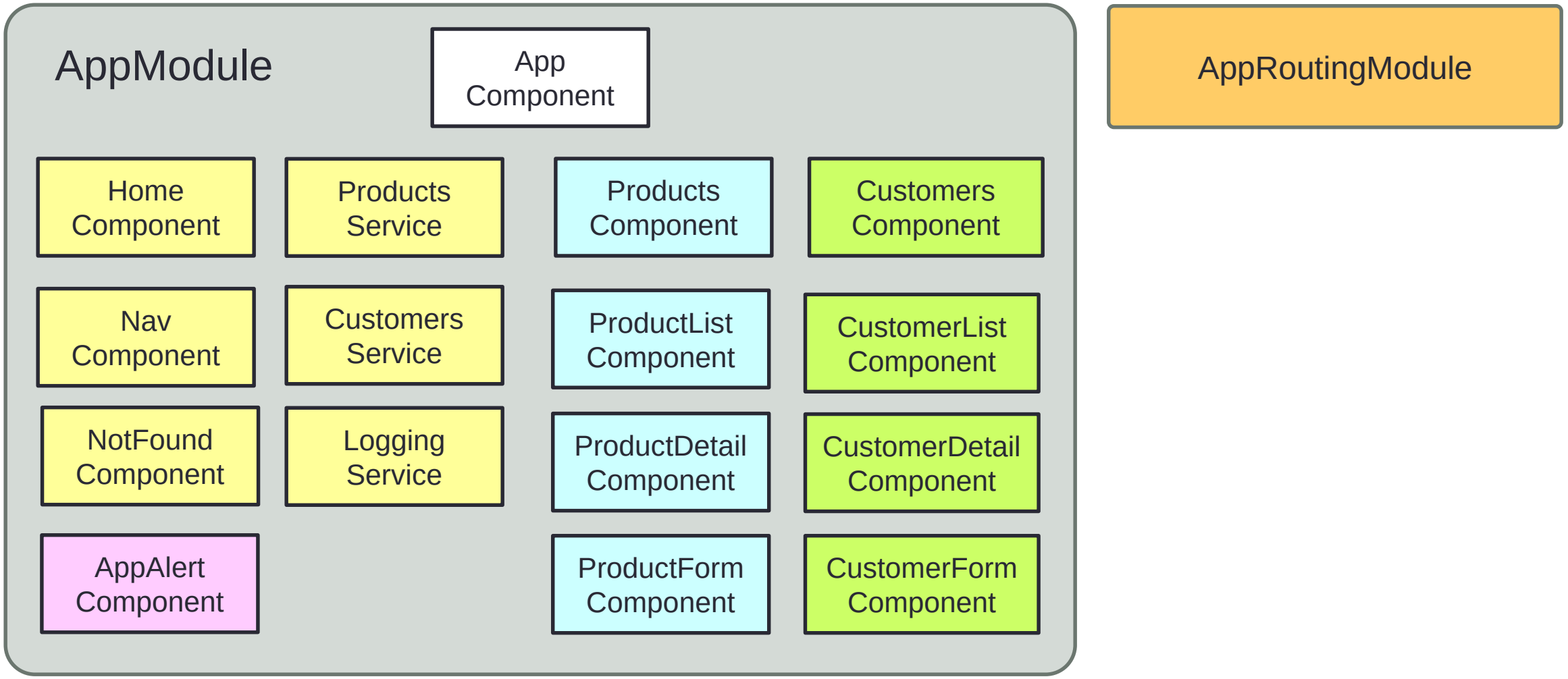
- Specifies metadata for an Angular module using a plain JS object

- Properties
  - declarations: []
    - specifies a list of components, directives, and pipes that belong to this module
  - imports: []
    - specifies a list of modules whose exported components, directives, pipes should be available to templates in this module
  - providers: []
    - specifies services that the components, directives and pipes can use within the app
  - bootstrap: []
    - defines the components that should be bootstrapped when this module is bootstrapped
    - defines the root component of the app
  - exports: []
    - makes some of the components, directives, and pipes public so that other module's component templates can use them

Routing Module



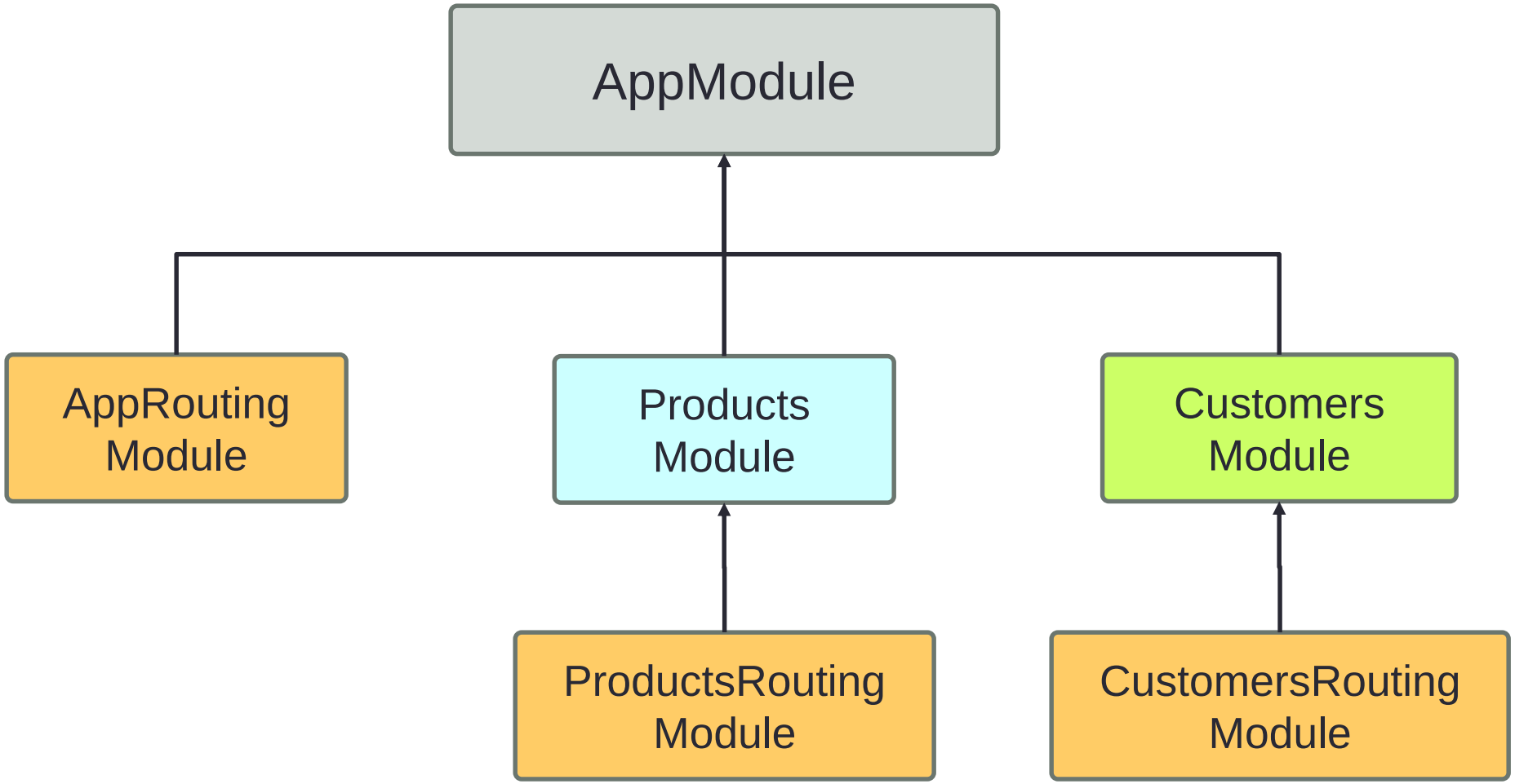
Routing Module



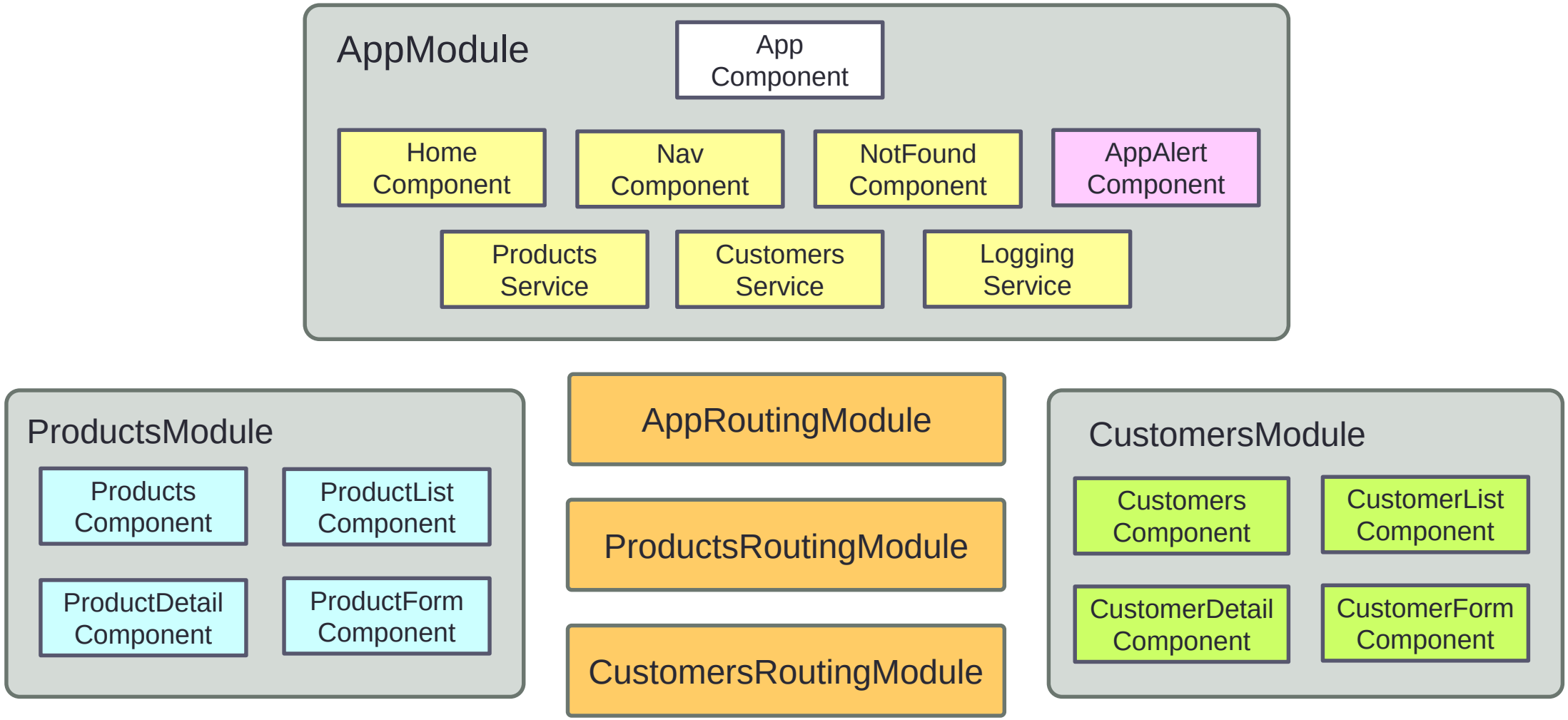
Routing Module

- Provides routing configuration for another module
- Separates routing concerns from its companion module
- Defines routes
- Adds router configuration to the module’s imports
- Re-exports the configured RouterModule
- Does not have its own declarations
- Should be imported only by its companion module
- Named after its companion module using the suffix “Routing”
  - E.g.
    - ProductsRoutingModule
    - CustomersRoutingModule
    - OrdersRoutingModule

Feature Module



Feature Module

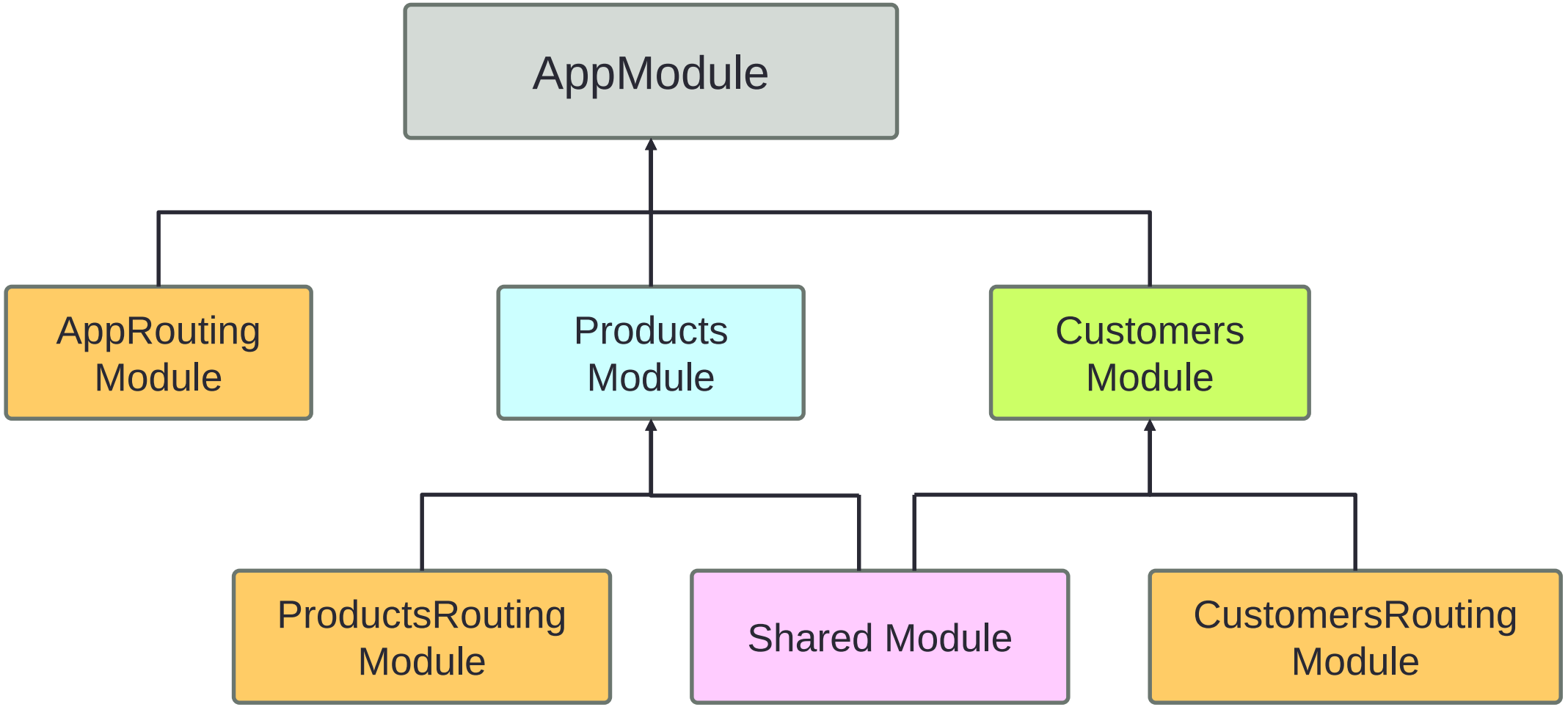


Feature Module

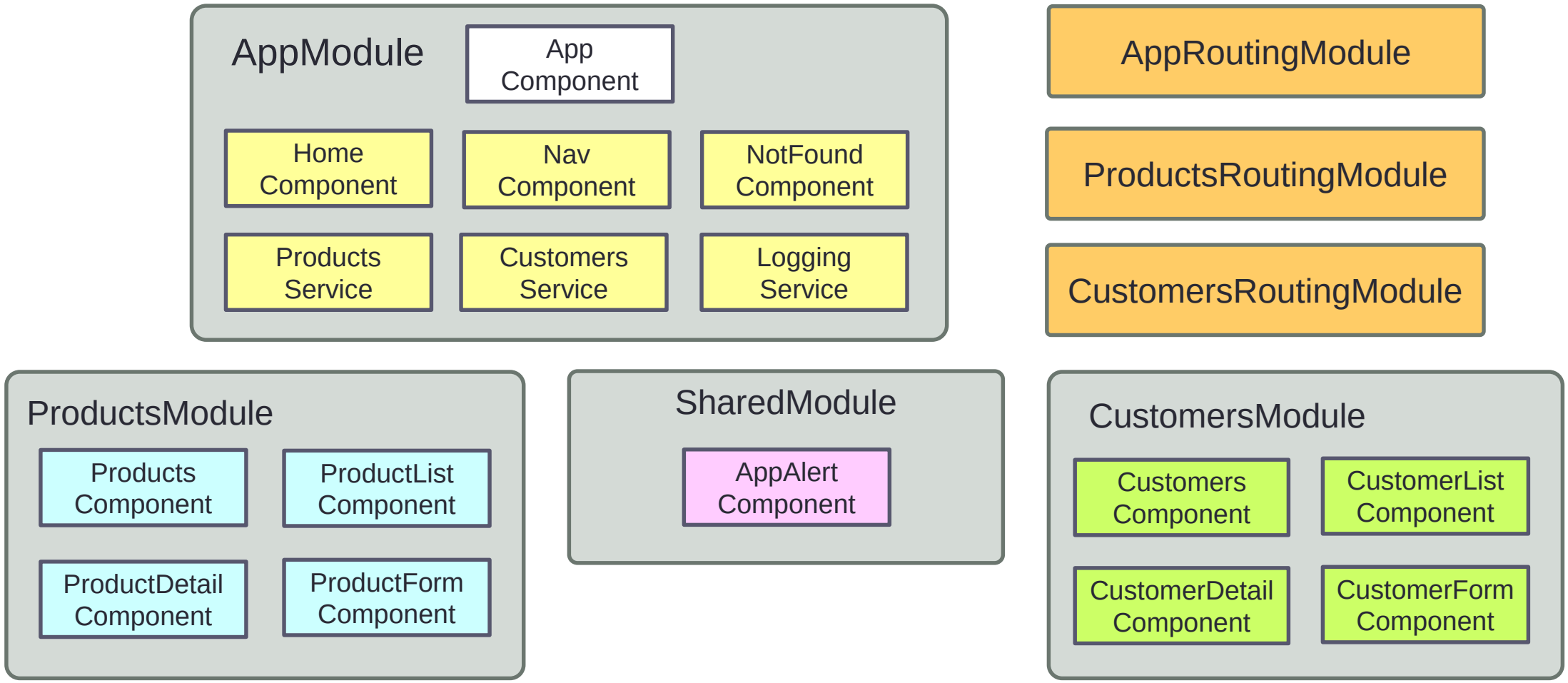
- An organizational best practice
- Organizes code relevant for a specific feature
- Helps you partition the app into focused areas
- Keeps code related to a specific functionality or feature separate from other code
- Collaborates with the root module and other modules

- E.g., ProductsModule, CustomersModule, OrdersModule, AuthModule

Sharing Module



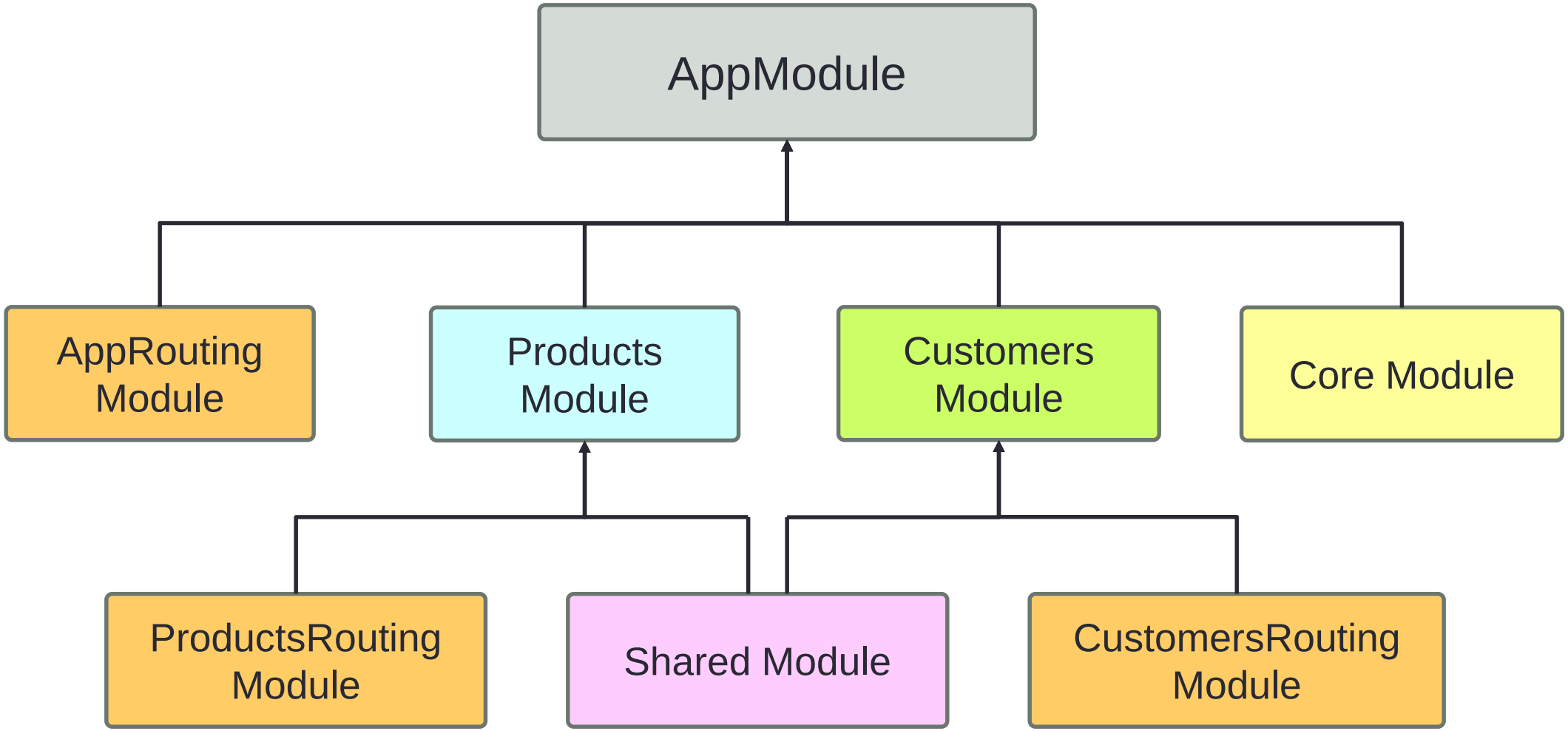
Sharing Module



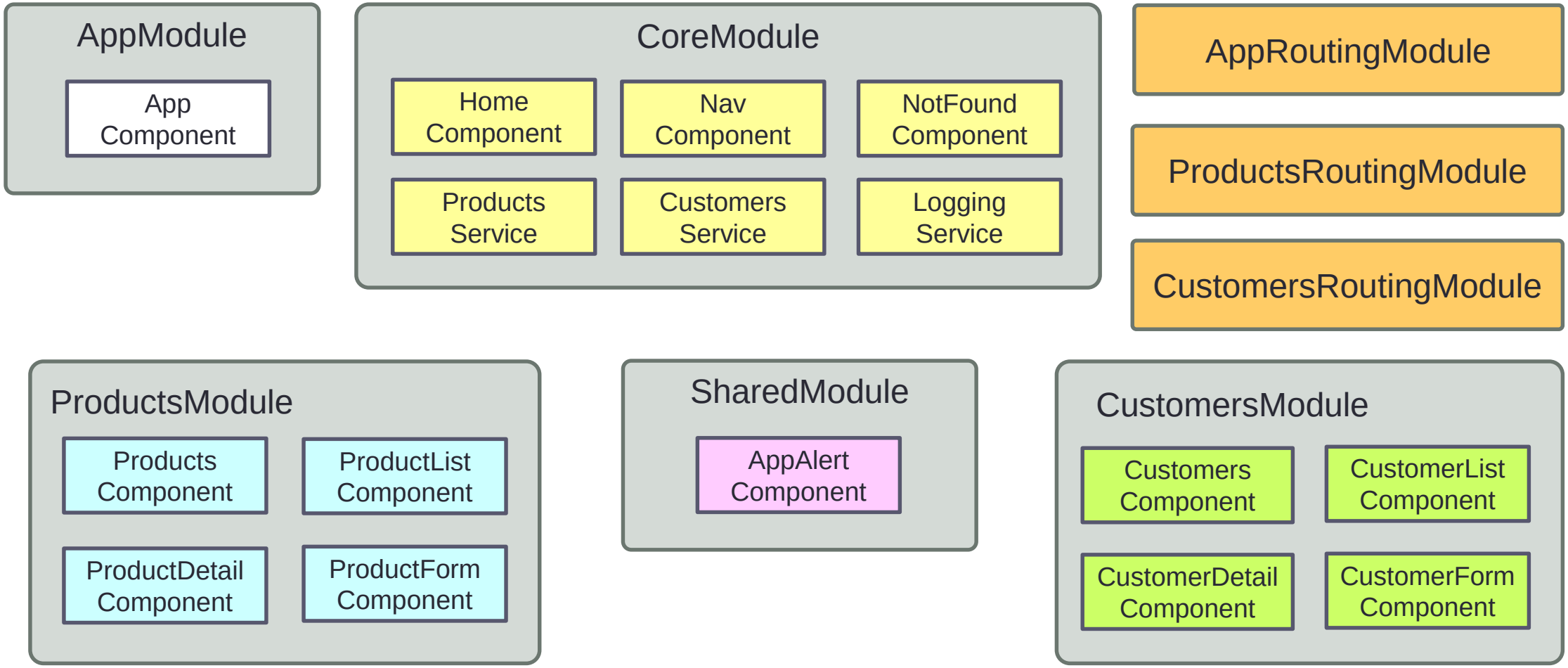
Sharing Module

- Organizes commonly used directives, pipes, and components into a module
- Makes components, directives, and pipes available to other modules of the app
- Mostly consists of declarations, most of them exported
- Import in any module whose component templates need the components
- Promotes reuse of code

Organizing Core Components of an App



Organizing Core Components of an App



Points to Note

- The AppRoutingModule adds router configuration to its imports with
  - RouterModule.forRoot(routes)
- All other routing modules are children that import
  - RouterModule.forChild(routes)
- Feature modules import CommonModule instead of BrowserModule
  - CommonModule contains information for common directives such as ngIf and ngFor which are needed in most templates

Q & A

- Thank you!