

# Dependency Injector

**3. Injector Trees - Injectors  
arranged in two trees**

# Injector trees

- Angular arrange the injectors in 2 trees
  - Element injector tree
  - Module injector tree

# Element injector tree

- Made out from ElementInjector
- Every time you supply value in **providers** array in a **Component/Directive** metadata you add a child to the tree
- Every DOM element can have an Injector.
- When asking for data from the **DI** inside a **Component/Directive** angular will traverse the tree starting from the closes **ElementInjector** and moving up till it finds the data

# Module injector tree

- Made out from **ModuleInjectors**
- This tree is made from the root injector
- Every time we load a lazy loaded module, we add a child to the tree
- Every injector here is loaded with:
  - **providers** - array
  - **imports** - array recursively loading from other modules providers array

# DI algorithm - Part 1 - find the tree

- To understand how the DI works we first need to understand what tree will the search start
- If we are asking in the constructor of a **Component/Directive/Pipe** angular will start from the nearest injector in the **Element Tree**
- If we are asking the DI in a constructor of a module, angular will search the nearest **Injector** in the **Module Tree**
- If we are in a service it depends on which of the above requested for the service recursively.

# DI algorithm - Part 2 - Traverse the tree

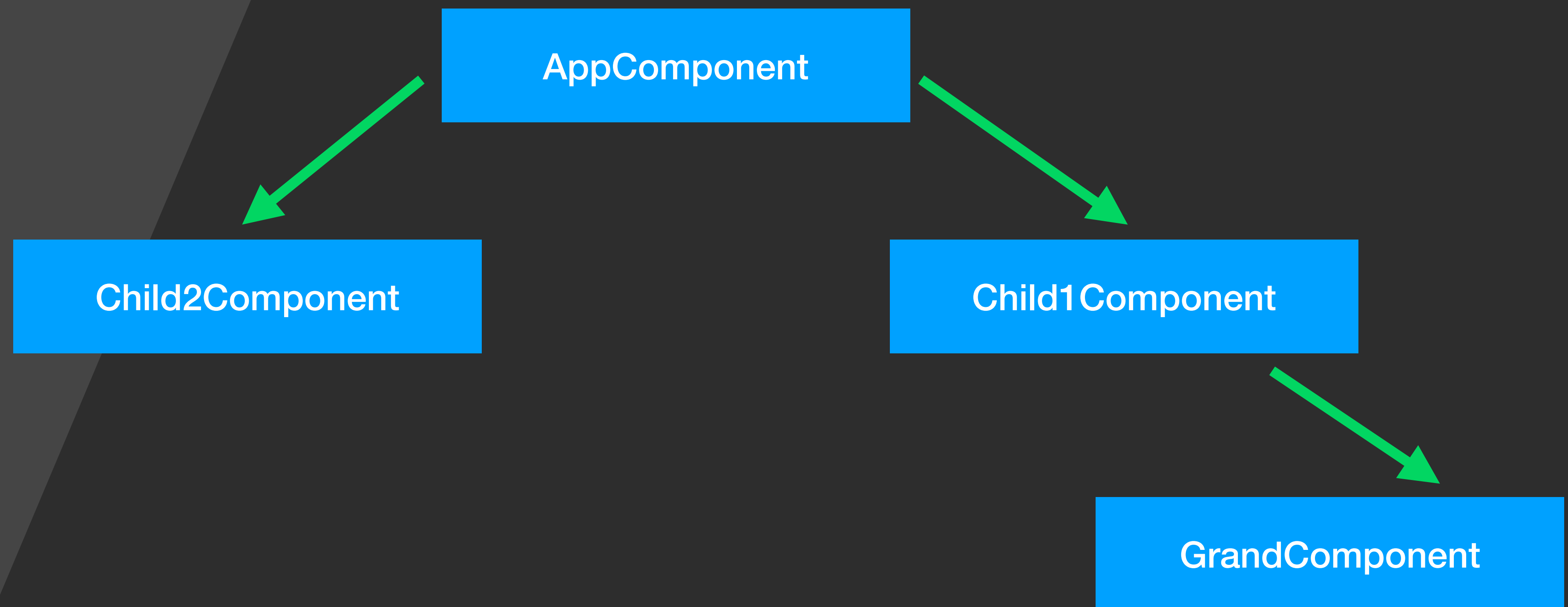
- After finding the tree and the nearest injector start traversing the tree asking each injector if it can find the data we are looking for
- If we are on the Element tree and the data was not found we start looking in the module tree starting again with the nearest module injector to whomever originated the request

# Exercise

Injector element tree

# Ex. Injector Element Tree

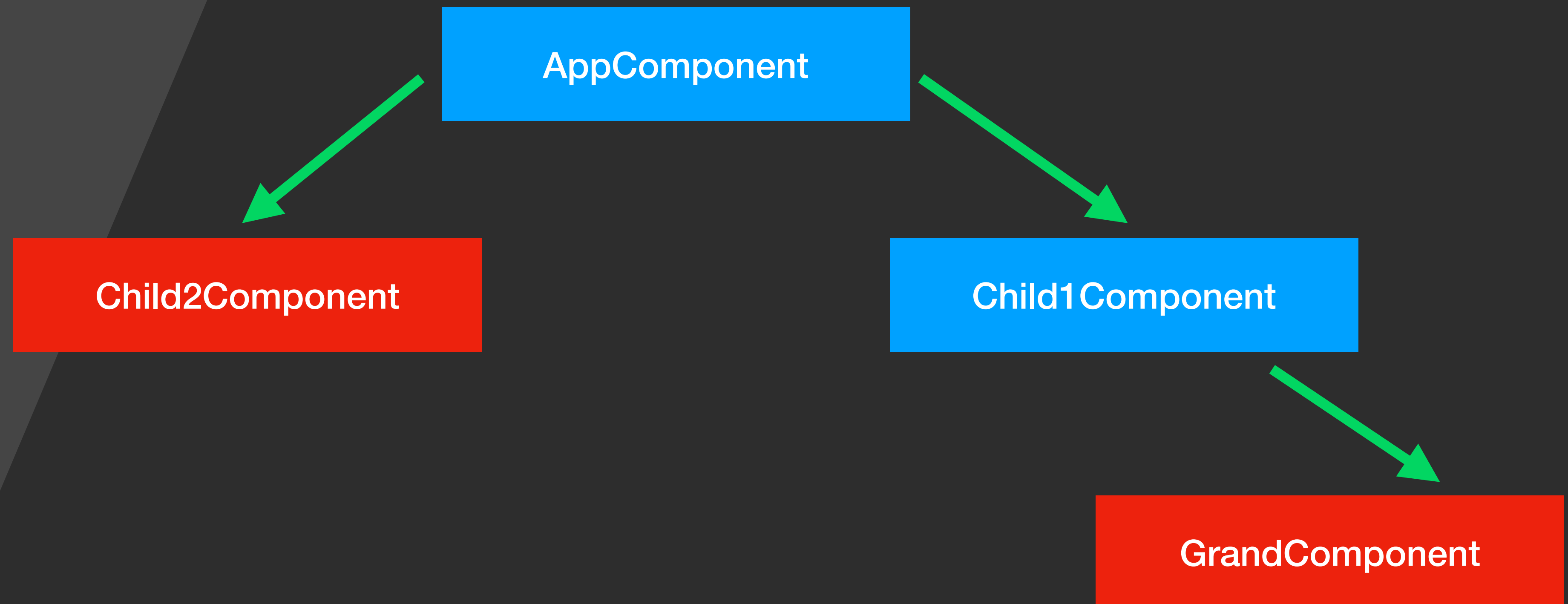
- Create the following component tree, where every component should display a simple message identifying that component from the other components





# Ex. Injector Element Tree

- The blue components should register a message with the Injector
- The “red” components should read the message from the DI
- Based on the algorithm we learned, from where will they get their message?

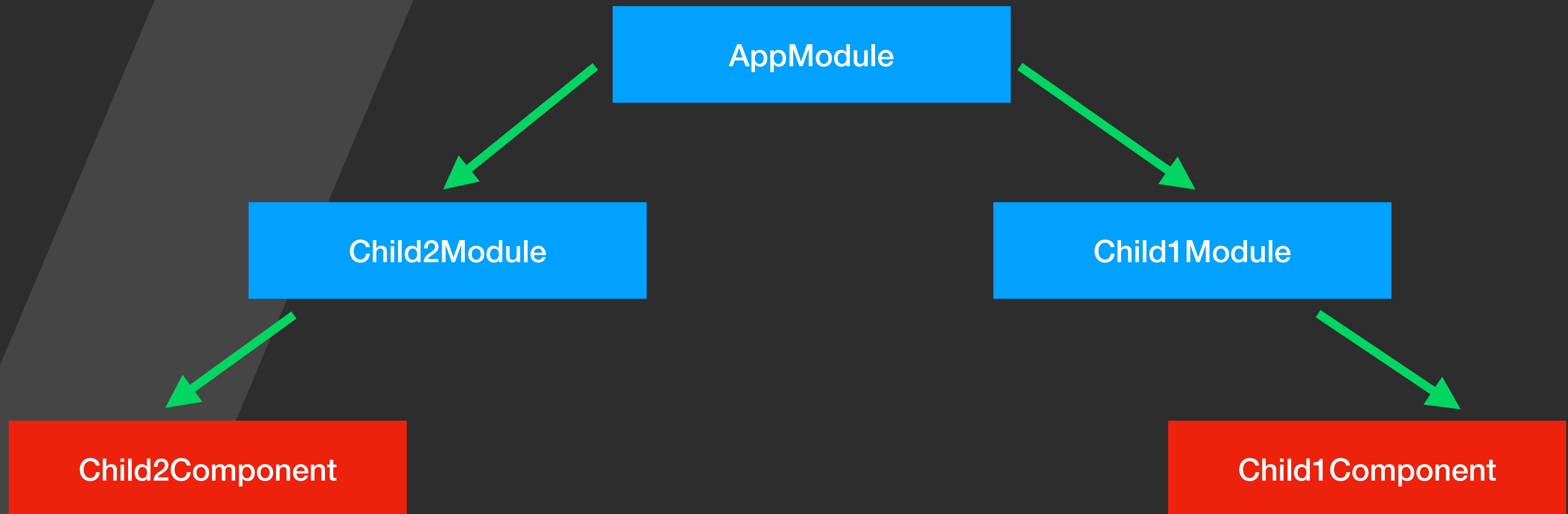


# Exercise

Module injector tree

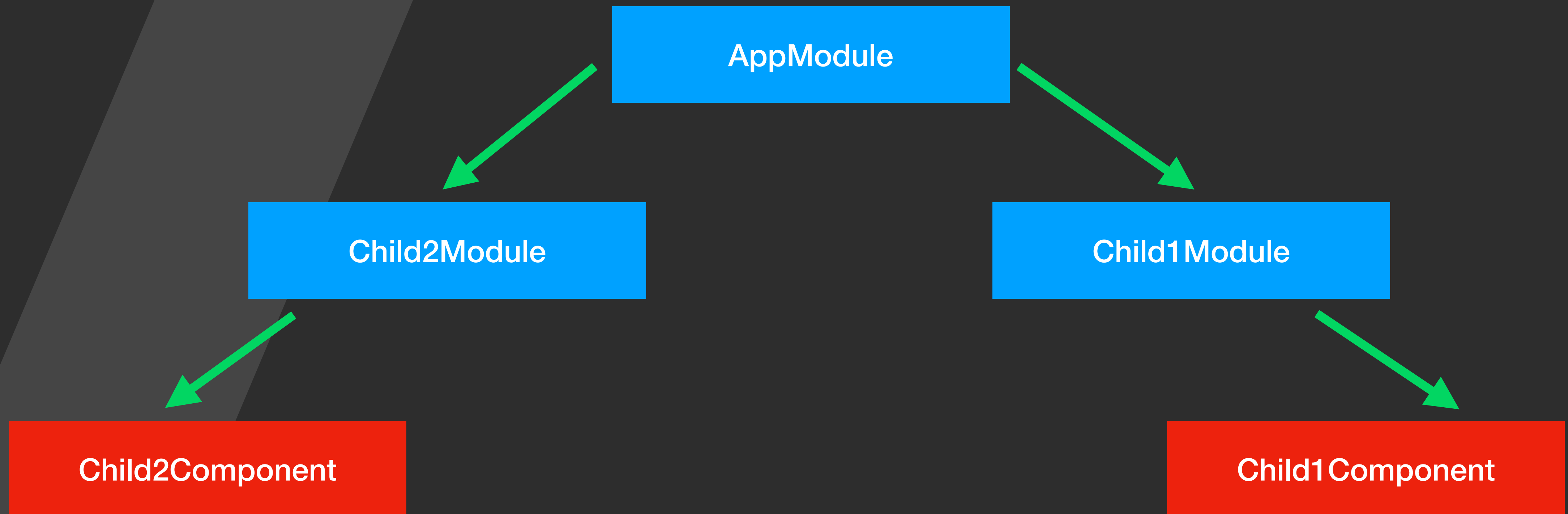
# Ex. Module injector tree

- The blue blocks are lazy loaded modules you need to create
- The red are components in the lazy loaded modules



# Ex. Module injector tree

- Provide a message in the **AppModule**, and in the **Child1Module** Injector
- Ask for that service in the components you created: **Child1Component**, **Child2Component**



# Summary

- Angular is making 2 Trees of injectors: Element and Module
- Angular will traverse the trees looking from the closes injector and looking up the tree
- After finishing with the element tree angular will keep searching in the module tree
- We can fine tune the search algorithm with additional decorators.

# Thank You

**Next Lesson: 3. DI Decorators**