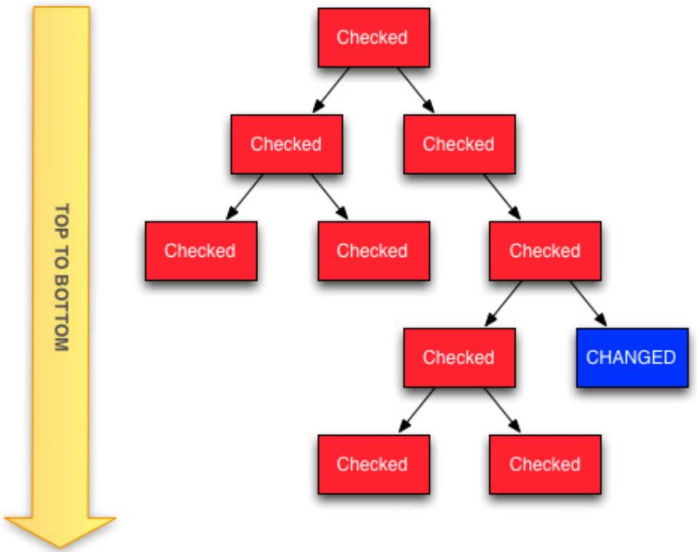
# Angular Change Detection

# Default Change Detection Mechanism

- 每个组件都有一个change detector
- 一个网页应用有很多组件
- 组件之间相互依赖，形成树状关系
- change detector也遵循这样的树状关系
- 任意一个detector的变化会导致整棵树从上到下的check.



Default change detection

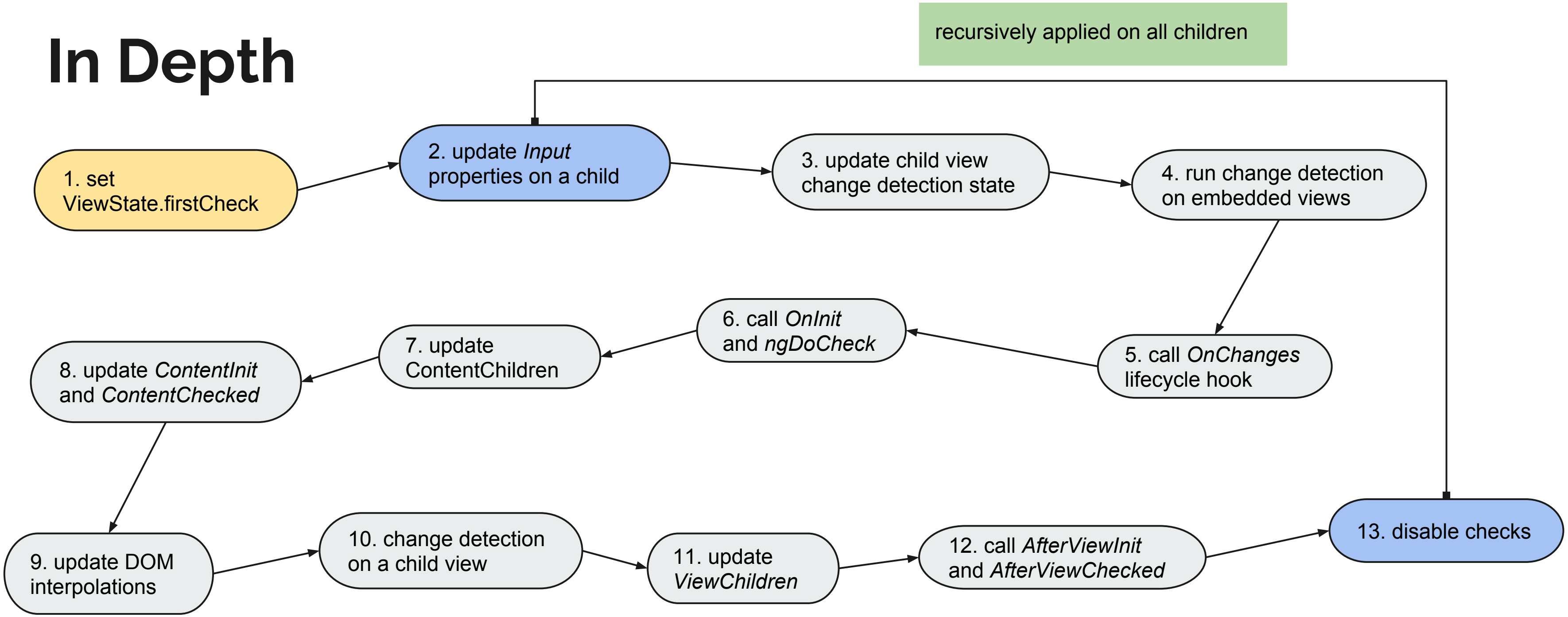
# Customize Change Detection

- ❑ 默认的检测机制可以进一步优化
- ❑ 可以配置一个组件的ChangeDetectionStrategy来自定义检测机制
- ❑ Example: 当Input改变时检测机制 - OnPush Strategy

# Zones

- Zones 是Angular用来实现检测机制的库
  - DOM
  - HTTP
  - Timer
- 有些可能不受控制的情景
  - Third party library that runs asynchronously
  - Immutable data
  - Observable
- Solution: OnPush Strategy

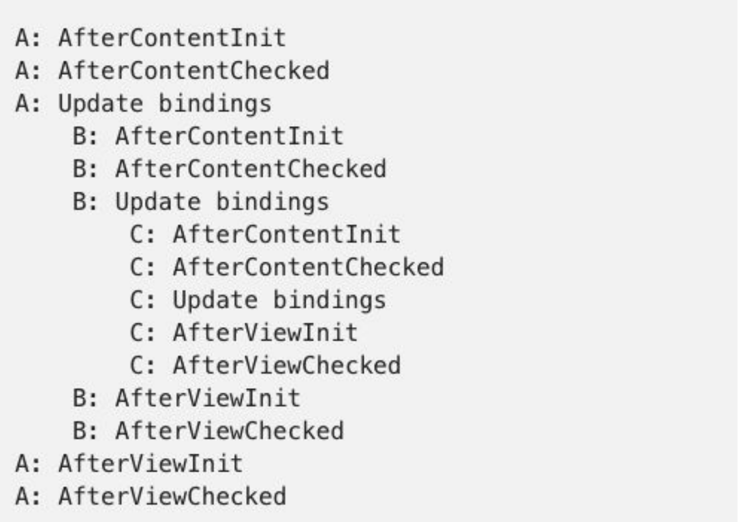
# In Depth


recursively applied on all children
1. set ViewState.firstCheck
2. update *Input* properties on a child
3. update child view change detection state
4. run change detection on embedded views
5. call *OnChanges* lifecycle hook
6. call *OnInit* and *ngDoCheck*
7. update ContentChildren
8. update *ContentInit* and *ContentChecked*
9. update DOM interpolations
10. change detection on a child view
11. update *ViewChildren*
12. call *AfterViewInit* and *AfterViewChecked*
13. disable checks

# In Depth

如果有一个A->B->C的继承结构，

那么lifecycle hook的调用顺序是这样的：



```
A: AfterContentInit
A: AfterContentChecked
A: Update bindings
    B: AfterContentInit
    B: AfterContentChecked
    B: Update bindings
        C: AfterContentInit
        C: AfterContentChecked
        C: Update bindings
        C: AfterViewInit
        C: AfterViewChecked
    B: AfterViewInit
    B: AfterViewChecked
A: AfterViewInit
A: AfterViewChecked
```

# Reference

- Everything you need to know about change detection in Angular
- The Mechanics of DOM updates in Angular