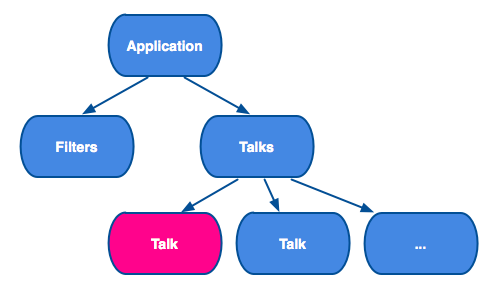
* **Angular 2 separates updating the application model and reflecting the state of the model in the view into two distinct phases.**
* The developer is responsible for updating the application model. Angular, by means of change detection, is responsible for reflecting the state of the model in the view.
* **Event bindings**, which can be added using the () syntax, can be used to capture a browser event execute some function on a component. So they trigger the first phase.
* **Property bindings**, which can be added using the [] syntax, should be used only for reflecting the state of the model in the view.
  1. **First, nothing has changed in the view. Only the model has been updated.**
  2. **Next, at the end of the VM turn, change detection kicks in to propagate changes in the view.**

**First, change detection goes through every component in the component tree to check if the model it depends on changed. And if it did, it will update the component.**



Then, the framework updates the DOM.

Note, the framework has used change detection and property bindings to execute this phase.

**Why?**

Now, when we have understood how we’d separated the two phases, let’s talk about why we did it.

#### Predictability

1. **Angular 2 makes reasoning about the component easier because it limits the number of ways it can be modified, and makes this modification predictable.**
2. In Angular 2, you know that Parent will always be updated before Child, and we also know that a property cannot be updated more than once.
3. In Angular 1.x the update can be interleaved. How many times prop1 is updated in the same digest run? It can more than 1. This makes implementing certain patterns very difficult. It also makes troubleshooting certain types of problems challenging: the static template just does not give you enough information to see what is going on. You have to know how the model gets updated. For instance,**it is natural to have cycles in your application model**.
4. It is hard to achieve this kind of guarantee in Angular 1.x.. First, talk=new Talk() could potentially result in cascading changes if the talk were two-way bound to some property on the parent component. Second, since the only Angular 1.x way of dealing with forms is to create an ngModel binding, it would be hard to keep Talk immutable.

#### Performance

A big part of the separation is that it allows us to constrain the view state propagation. This makes the system more predictable, but it also makes it a lot more performant. For example, the fact that the change detection graph in Angular 2 can be modeled as a tree allowed us to get rid of digest TTL. Now the system gets stable after a single pass.

**How Does Angular Enforce It?**

What happens if I try to break the separation? What if I try to change the application model inside a setter that is invoked by the change detection system?

* Angular tries to make sure that the setter you define for you component only updates the view state of this component or its children, and not the application model.
* To do that Angular will check your bindings twice in developer mode.
* First time to propagate changes, and second time to make sure there are no changes.
* If it finds a change during the second pass, it means that one of your setters updated the application model, the framework will throw an exception, giving you the feedback you need to ensure it runs properly in production.

**What About Angular 1.x?**

Although Angular 1.x was not built with this separation in mind, you can still write your code in this way to make your migration easier. Here’s a few things you can do:

* Prefer one-way data-bindings to two-way data-bindings.
* Do not update application state inside watch functions.
* Be explicit about how data flows in your application. For example, try to implement unidirectional data-flow. Read more about it [here](http://victorsavkin.com/post/99998937651/building-angular-apps-using-flux-architecture).

**Summary**

* Angular 2 separates updating the application model and updating the view.
* Event bindings are used to update the application model.
* Change detection uses property bindings to update the view. Updating the view is unidirectional and top-down. This makes the system a lot more predictable and performant.
* Angular 2 embraces unidirectional data-flow.
* You can use the same mindset when building Angular 1.x applications.

References:

https://vsavkin.com/two-phases-of-angular-2-applications-fda2517604be#.k8lh5kmm2