**React Components Are Simple**

Components are responsible for small isolated chunks of UI, they accept data and render consistently, every time. Front-end developers, back-end developers and designers can easily understand and contribute to components because they look like HTML and the minimal API footprint means that you can learn what you need to know about components in a day and start contributing right away.

**Server Rendered HTML**

React makes server-side rendering trivial because components can be thought of as functions that take data and return HTML. This design makes it easy to integrate server-side rendering into any server-side programming language or framework.

In the early days of client-side MVC we broke the web with things like hash bang routing and huge loading times before anything made its way to the screen. We also broke search engine crawlers by rendering everything with JavaScript when the page had loaded. Since then we’ve learned from our mistakes and are taking these core tenets of the web seriously again — URLs, server rendered HTML and fast load times.

React shines here where other frameworks have struggled.

**DOM Updates Are Messy**

Backbone was an important milestone for JavaScript. As the first and most prominent effort to bring MVC to the client-side, it showed us a new way that we could structure our applications. One of the first things it mentioned in its documentation is that it’s a bad idea to maintain your state in the DOM. When the DOM remains the source of truth your app quickly becomes brittle and hard to follow. It becomes impossible to know which piece of code changed which element. Backbone encouraged an ideal of always re-rendering views based on the current state of the world, React enforces this same idea with it’s pattern of one-way data flow.

React components don’t define the transition between states. Instead, they explicitly render the view based on their current state, completely eliminating this task of manually tweaking the DOM. Its one-way data flow prevents the DOM from being the source of truth.

Admittedly, this makes certain tasks such as animation more difficult because those are cases where you do want to define transitions between states. For the vast majority of cases though, it’s much simpler to only concern yourself with the final state of how the component should be rendered.

**The Future**

React will continue to grow in popularity and we’ll see more supporting tools and projects. As the ecosystem around React matures, the library may change but the core ideas of one-way data flow, component hierarchies, explicit renders and virtual DOM reconciliation will live on.

React Native has shown how a simple view layer can be re-purposed to construct other types of UI as well. There’s been a shift in the industry towards this pattern of building UI’s and it’s not going away any time soon.

In short, React has won and the future is bright for us.