Hello, everybody. I’m delighted to be here today.

So we are going to know more about **Reconciliation in React**.

I’d like to start by mentioning **a few facts about React**.

ReactJS is a JavaScript library for building user interfaces. It was created in 2011 and is maintained by Facebook as an open source project.

React has become highly popular because of its extra simplicity and flexibility. Many companies choose to build their websites using React. The most famous of them are Facebook, Netflix, Airbnb, BBC, PayPal, Twitter, Instagram, Dropbox, Uber, WhatsApp and many others.

Now let’s turn to the matter and find out why React is really so fast.

First of all we should **get acquainted with the concept of the DOM**.

When we open any web page we usually don’t realize what amount of work stands under this process. When we start navigation actually we trigger the chain of events, including DNS lookup, connection to the server setup, safety negotiations, HTTP GET request. The browser doesn’t render the received data onscreen immediately. Having received the first chunk of data it starts parsing it into the DOM.

DOM stands for Document Object Model. It represents the tree-like structure of a document in memory, providing an interface to read and change document’s structure, style, or content.

**The problem with the DOM** is that it is not optimized for dynamic UI applications.

Just imagine that we want to change something in the UI. For example, we want to change the inner HTML of some element with the id elementID. What is going to happen? The browser will have to fulfill many tasks listed in the slide, including updating the DOM and re-rendering the UI.

Updating the DOM itself doesn’t take much time, but rendering and re-rendering the UI is time-consuming and can slow any web application.

Each library or framework tries to solve this problem in its own way. React proposes **the concept called the virtual DOM**.

The virtual DOM (VDOM) is a virtual representation of the DOM (or UI) (like its lightweight copy) that React creates and keeps in memory.

When React wants to render the UI, it is first rendered into the virtual DOM and only then the virtual DOM turns into the real DOM.

So, let’s find out **what the benefits of having the VDOM are**?

* Whenever anything is changed, React saves the current version of the virtual DOM and then updates the Virtual DOM. Now React has 2 versions of the Virtual DOM
* Then React compares both Virtual DOMs and finds out the differences between them in order to understand what should be changed in the real DOM and the UI. This process through which React determines which parts of the DOM need to be changed is called **reconciliation**.
* At the end React updates the real DOM, not the whole DOM, but only those parts that have actually changed. This is very much like applying a patch.

So React can update only the necessary parts of the DOM. This makes a big difference! React’s reputation for performance comes largely from this innovation.

Are you ready to learn more? Take a breath, we are going to dive deeper.

How does React detect changes in the UI?

**React uses observables** instead of dirty checking to detect change. It listens to change events on the state or props and if these events happen (when the props to a component or the state of the component changes) calls the render() function. Such way of changes detecting is really faster than checking the values in the data structure recursively at a regular interval.

A very interesting question is how React compares 2 virtual DOMs?

If the nodes are compared in turn through recursion the complexity of the diff algorithm will be reaching O(n3) – big 0 from n cubed. How terrible is this? If the tree contains 1000 elements we’ll need one billion comparisons. This is far too expensive. To be fast React needs to generate the minimum number of operations to transform one tree into another. React **implements a special heuristic diff algorithm** with the complexity just **O(n)** that is t**he concrete realization of reconciliation. This diff algorithm has 3 strategies or levels.**

1. Tree diff

React traverses the tree using BFS (Breadth First Search). It goes from top to bottom, from left to right, as if it were reading or checking each row (or layer) of nodes. React makes an assumption that the movement of DOM nodes across layers is particularly small and ignores it.

While comparing layers if React finds that any element has been modified, it will, by default, re-render not just this element but the whole subtree under it without checking the depth of the tree. This may sound scary and inefficient but in practice this works fine because changes usually are localized in rather deep elements and destroyed sub-trees are not really big.

2. Component diff

When diffing components or nodes, React first compares their types. If the type has been changed, for example from ‘div’ to ‘span’, React will destroy all the sub-tree under that element and will reconstruct it from scratch. This also concerns the whole components.

If two elements have the same type, React will compare their attributes and update only the modified attributes.

3. Element diff

When the nodes are at the same level, React adopts a very naive approach. It goes through both lists of children at the same time and generates a mutation whenever there's a difference.

Here diff provides 2 types of node operations: **delete and insert. React can also use move operation, but it can’t find by himself that the elements have been moved (like in the second example in the slide). We should help him. How? - we’ll learn at the end of the presentation.**

Besides the efficient diff algorithm React has one more useful tool. It’s **a batch update mechanism**. **This means that updates to the real DOM are sent in batches (groups), instead of sending updates for every single change. So d**uring an event loop, the DOM is being updated just once.

**To make a conclusion we have made sure that React is really fast.**

The last question we are going to answer is **how developers can help React to be fast?**

Let’s return to the example with the list of elements. The difference between 2 code fragments is just in the li element Lemon. What is React going to do? It will re-render all the child elements instead of keeping and moving the  <li>Apple</li> and <li>Banana</li>.

The developer can hint at which child elements may be stable across different renders by providing a key attribute to the element. In the second example we use keys to li elements. Now React knows that the element with the key '3' is the new one, and the elements with the keys '1' and '2' have just moved.

We know that React’s change detection runs when the props to a component or the state of the component changes and  render() function is called by default. We can help React to save some render() calls if we don’t want to change anything by using the special methods componentWillReceiveProps({}) (to check if the current component can use the given property) and shouldComponentUpdate({}) (to check if the current component will update either on state or property change).

**The developers can** hide and display nodes through CSS instead of actually removing and adding DOM nodes **because cross-level operations on the DOM nodes will result in deleting f the whole sub-tree and recreating of a new one, not moving it.**

## The developers can also use a React.PureComponent, make data immutable and use other methods.

I suppose you haven’t got tired with my presentation.

Thank you for your attention.