

# Synthetic events in Reactjs

Uladzimir Dziomin

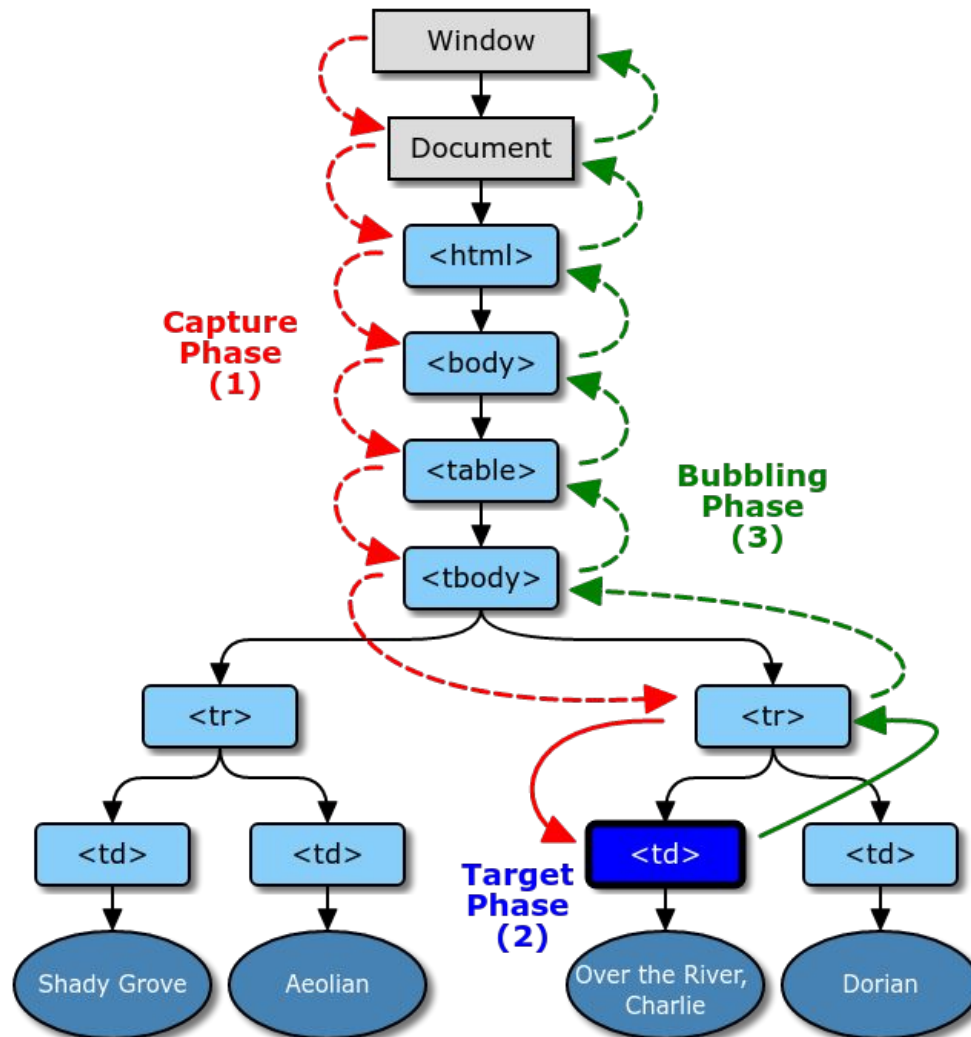
Captain obviously



# Event is...

HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document

# DOM events bubbling / capturing



# Bubbling/capturing

- `addEventListener('eventName', handler, isCapturing);`
  - Last argument set bubbling/capture event capturing phase
- `<div onclick='handler'></div>`
  - Always bubbling

# Dom event

- `event.preventDefault()`
- `event.stopPropagation()`
- `event.stopImmediatePropagation()`
  - `x.addEventListener("click", myFunction);`
  - `x.addEventListener("click", someOtherFunction);`

# Event delegation

- Attach event to the **window** or **document** and listen it
- Use **event.target** to check what element caused the event

# Event pooling

- All events will be added to some pool
- When event was handle then clear it and return to the pool



Reactjs

# How events work in reactjs

- React uses top-level event delegation
- React doesn't attach the events to the real DOM nodes
- React uses Synthetic Events to process events within virtual DOM
- **addEventListener** will attach events to DOM nodes
- you have **nativeEvent** reference inside synthetic events

# How events work in reactjs

- React attaches 63 events to one handler
  - scroll\*, focus, blur are captured
- <https://github.com/facebook/fbjs> is used to attach events
- **One per event** listener



# React event delegation

- Delegation listener is attached to the **document**
- When it receives an event it dispatches a synthetic event across the virtual DOM

# React event pooling

```
if (event) {  
    EventPluginUtils.executeDispatchesInOrder(event, simulated);  
  
    if (!event.isPersistent()) {  
        event.constructor.release(event);  
    }  
}
```

# Pooling problem

- Event can be cleared before you process it
  - when your handler puts your request to the queue

# EventPluginHub

- **extractEvents** - extracts synthetic events that will be queued and dispatched
- **eventTypes** - is the name that is used to register a listener
- **executeDispatch** - overloads the way how event will be dispatched



# Synthetic event

- Synthetic events are dispatched by event plugins of delegation handler
- React's event system and native DOM events are isolated
  - Your component will be the last to hear the event.
- Use react's event system as much as you can.

# When Synthetic events are not enough?

- **resize, hashchange, transitionend, ... events**
- React component is inserted into a page containing other event listeners
  - If an event happens inside your React component, your component will be the last to hear the event

# Adding DOM events is the same

- `componentDidMount`:
  - `window.addEventListener('resize', this.closeTooltip);`
- `componentWillUnmount`:
  - `window.removeEventListener('resize', this.closeModal);`

# But how to stop it for real DOM?

`event.stopPropagation()` - the synthetic event can't stop real DOM events...



....hmmmm

**BUT how to stop it for real DOM?**

`reactEvent.nativeEvent.stopPropagation()`

... still can't stop the propagation



Only one chance...

`reactEvent.nativeEvent.stopImmediatePropagation()`

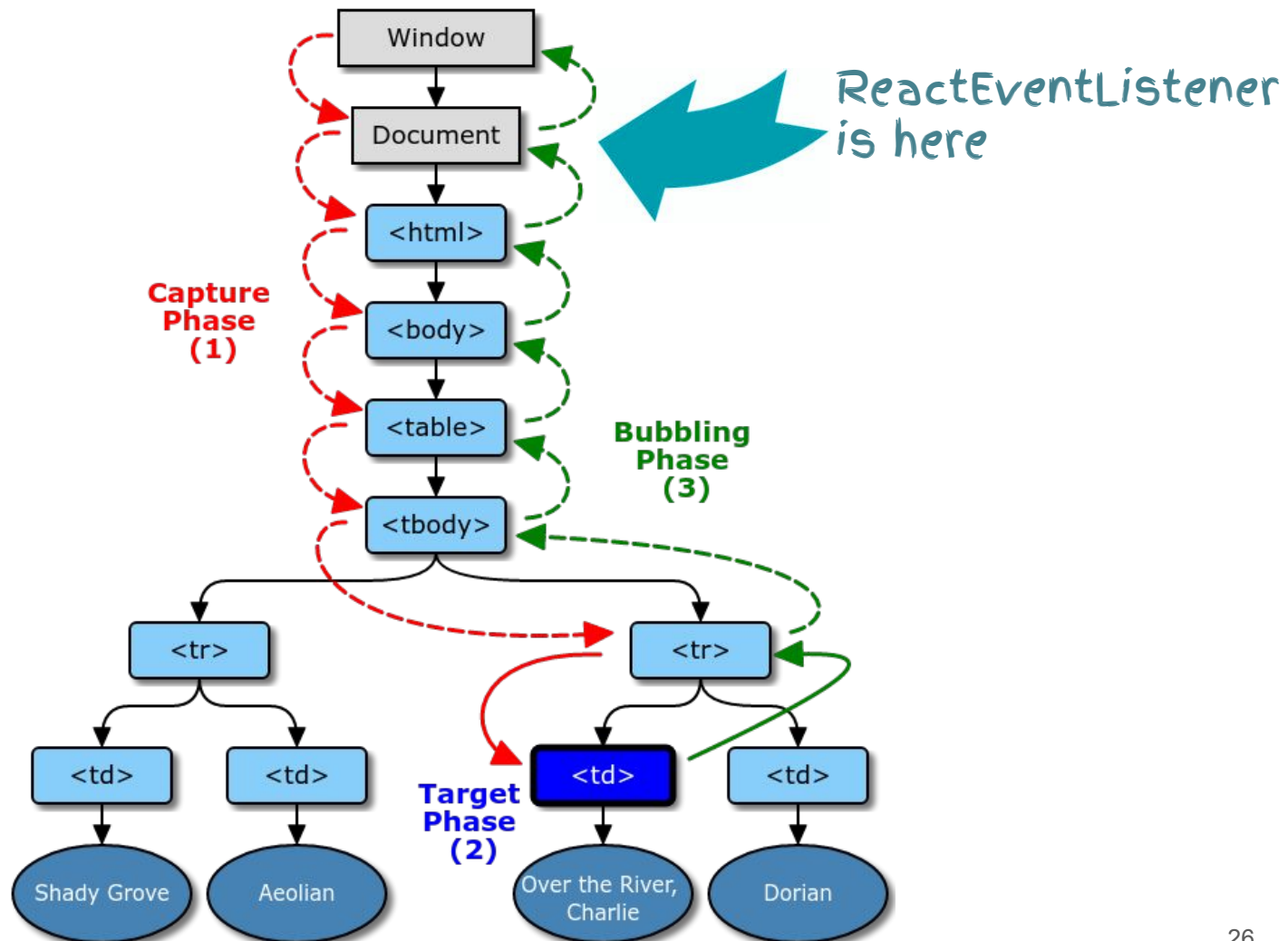
...will work only for event connected to **document** and above



# Does stopping the native event stop react's event?

- Yes... but it stops **all** react events

# DOM events bubbling / capturing



# Conclusion

- React takes care on events handling
- No connected to DOM events => no worrying
- But if you need complex behaviour you need to know it under the hood

Thank you!

It's questions time :)