一起来造个轮子（三）

基于Virtual Dom的组件框架

luckyadam

1 更新机制

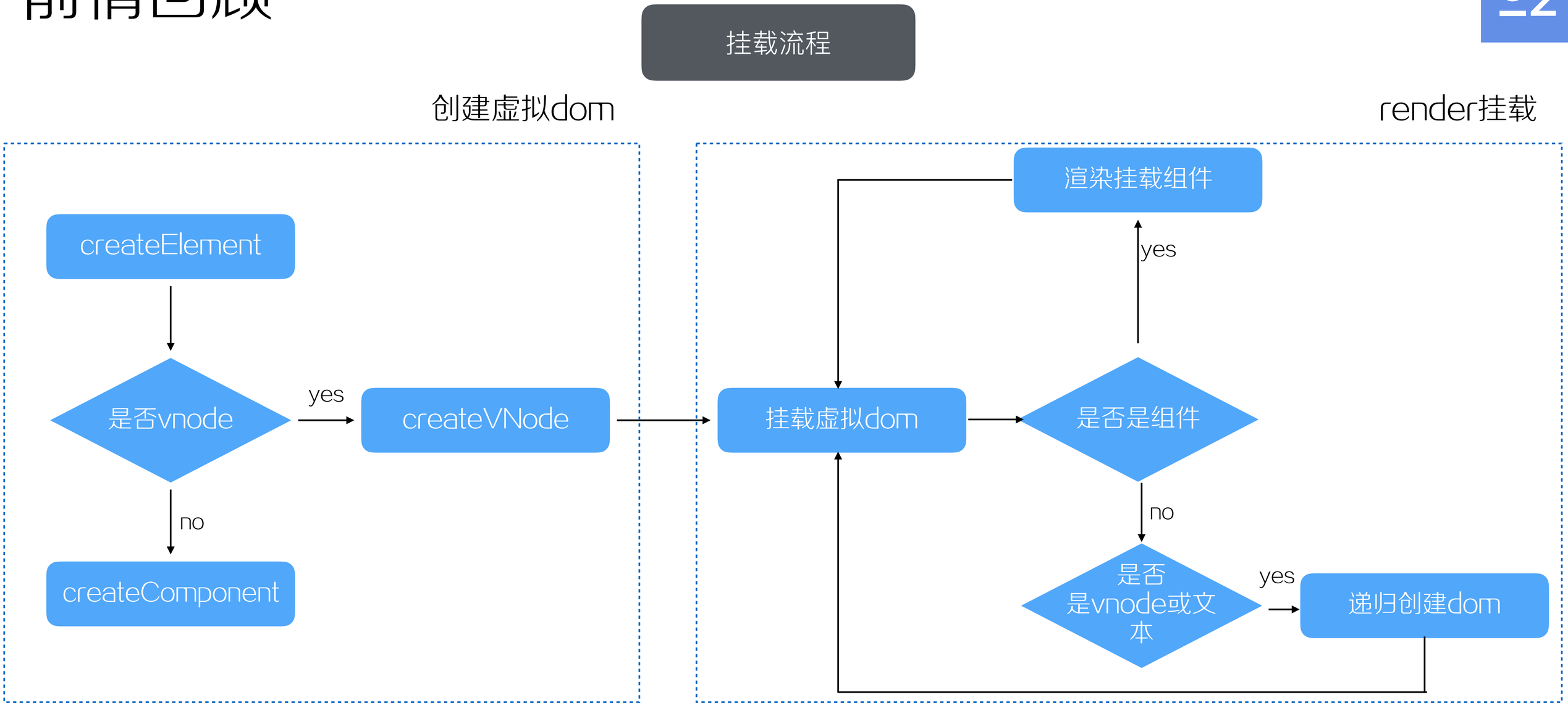
2 事件处理

3 SVG处理

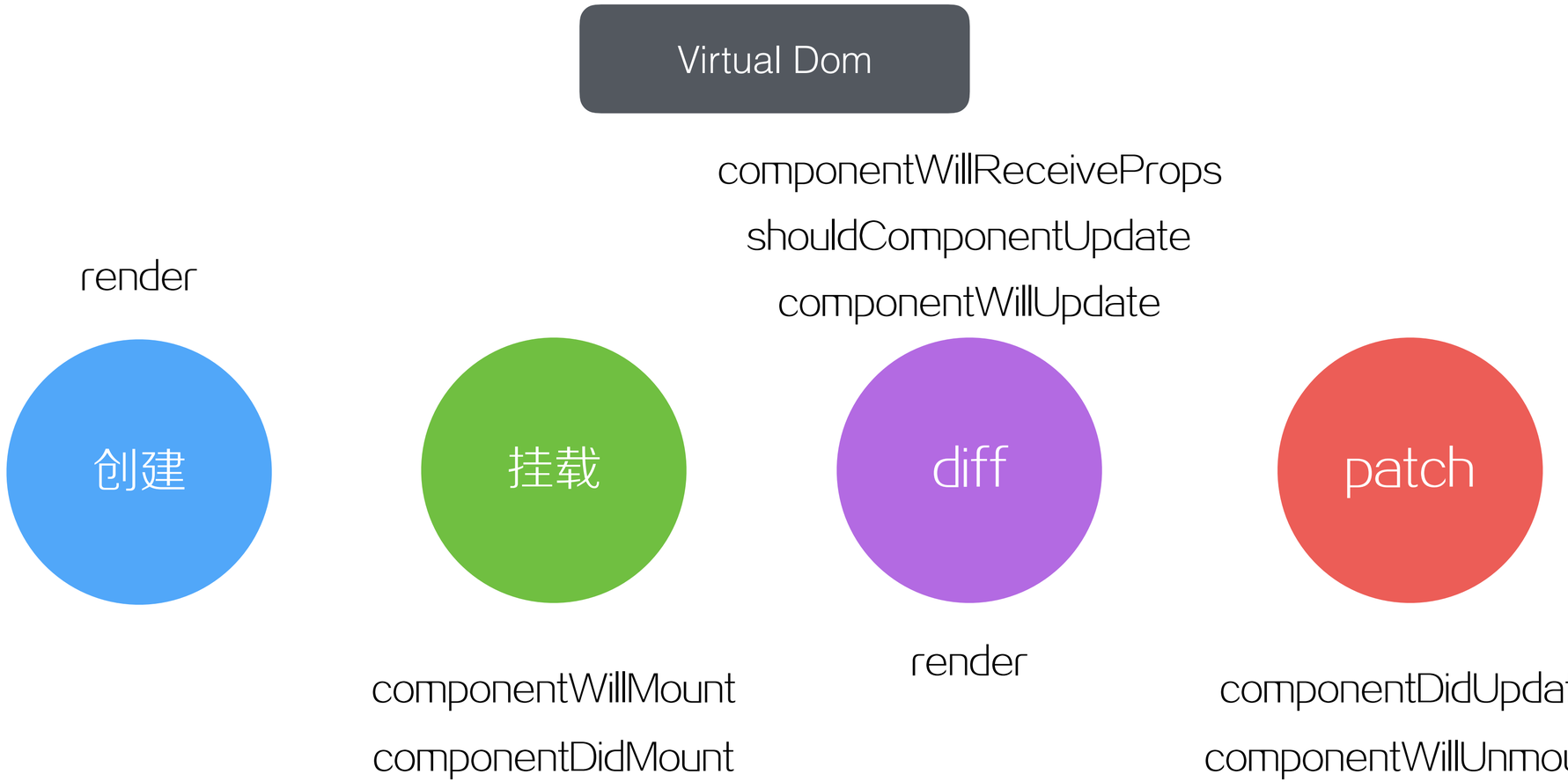
前情回顾

前情回顾

挂载流程


创建虚拟dom
render挂载
createElement
是否vnode
yes
createVNode
no
createComponent
挂载虚拟dom
渲染挂载组件
yes
是否是组件
no
是否
是vnode或文
本
yes
递归创建dom

前情回顾

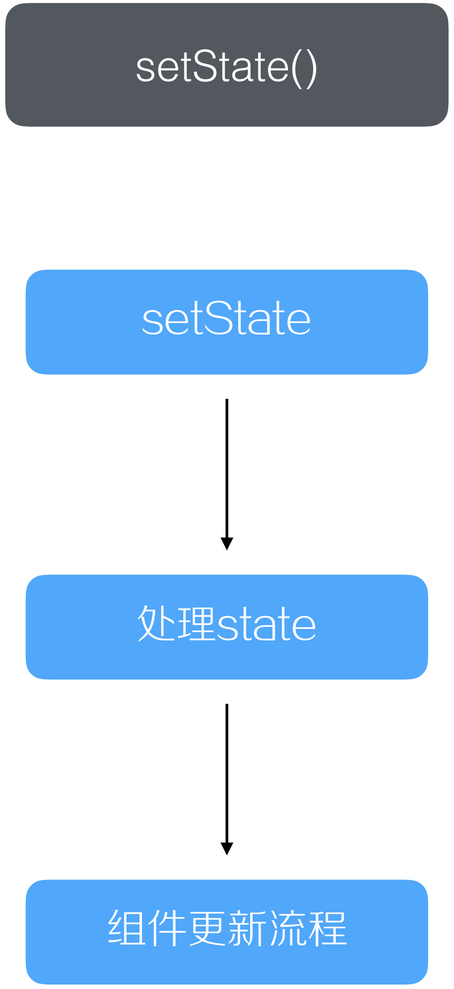

Virtual Dom
componentWillReceiveProps
shouldComponentUpdate
componentWillUpdate
render
创建
挂载
diff
patch
componentWillMount
componentDidMount
render
componentDidUpdate
componentWillUnmount

1

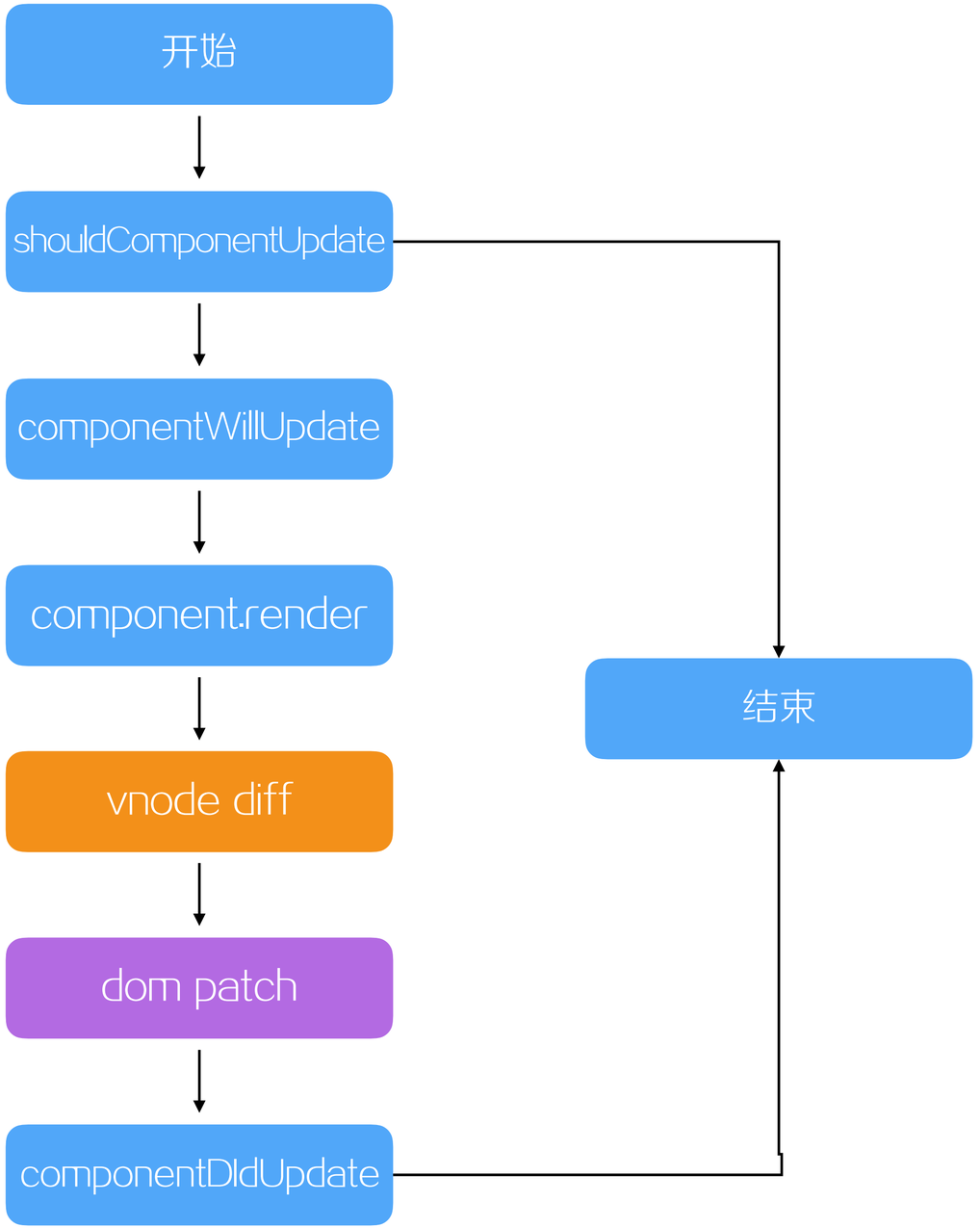
更新机制

更新机制

02



更新机制


开始
shouldComponentUpdate
componentWillUpdate
component.render
vnode diff
dom patch
componentDIdUpdate
结束

更新机制

组件更新

```
function updateComponent (component) {
  const lastDom = component.dom
  const state = component.state
  const props = component.props
  if (component.shouldComponentUpdate
    && component.shouldComponentUpdate(props, state) === false) {
    return
  }
  if (component.componentWillUpdate) {
    component.componentWillUpdate(props, state)
  }
  const lastRendered = component._rendered
  const rendered = component.render()
  const patches = diff(lastRendered, rendered)
  component.dom = patch(lastDom, patches)
  if (component.componentDidUpdate) {
    component.componentDidUpdate(props, state, context)
  }
}
```

执行各个生命周期方法

生成新的虚拟dom

两次虚拟dom之间diff

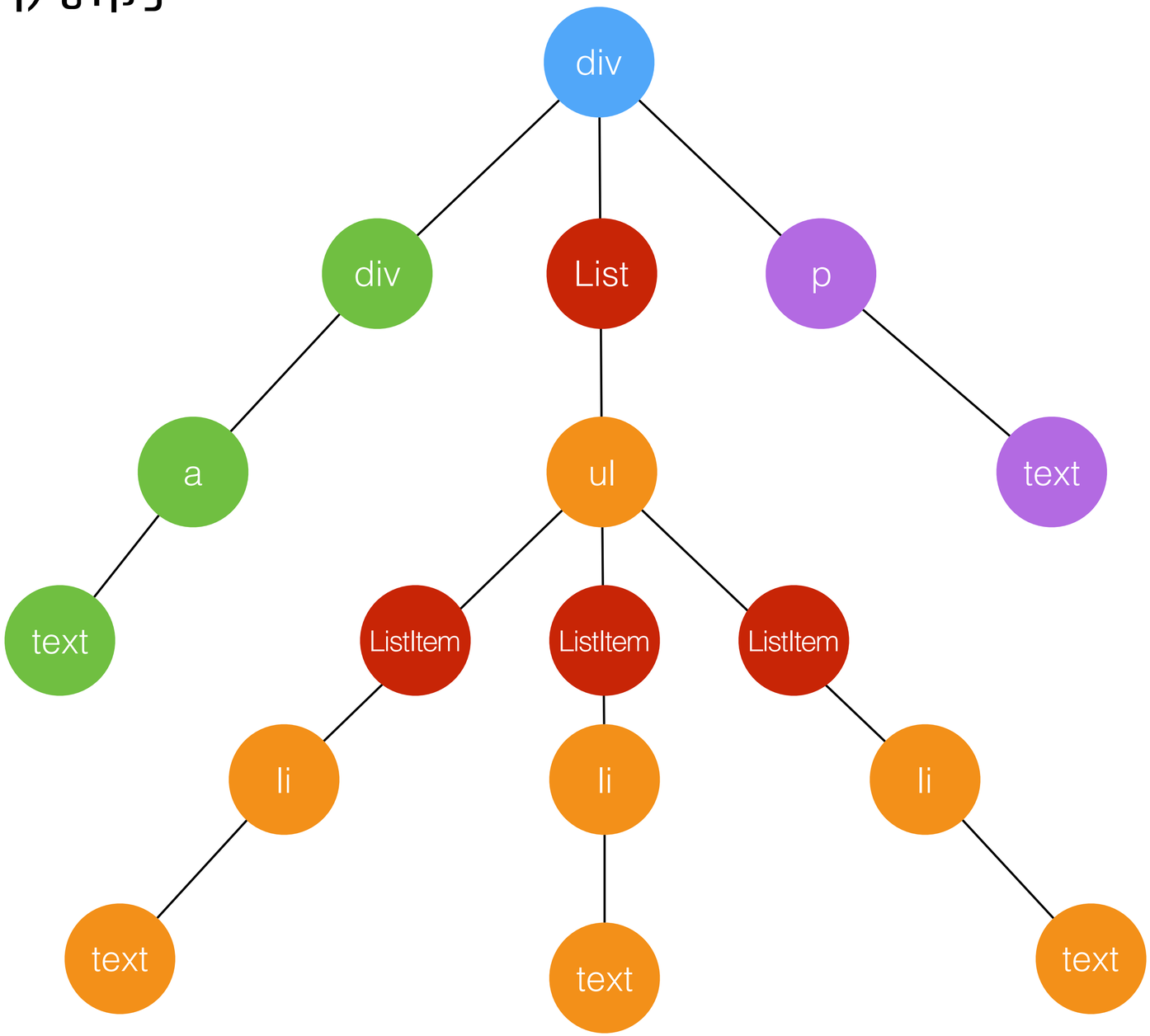
更新差异

更新机制

02

组件diff?

更新机制



vnode节点包含组件

组件包含vnode节点

递归创建

更新机制

diff

```
if (isComponent(b)) {
  apply = appendPatch(apply, {type: 'component', patch: b, old: a})
}
```

diff新增component类型

更新机制

patch

```
function patchComponent (domNode, oldNode, newNode) {
  let newDom
  if (isComponent(oldNode) && oldNode.constructor.name === newNode.constructor.name) {
    newDom = reRenderComponent(oldNode, newNode)
  } else {
    newDom = createDOMNode(newNode)
  }
  const parentNode = domNode.parentNode
  if (parentNode && domNode !== newDom) {
    parentNode.replaceChild(newDom, domNode)
  }
  return newDom
}
```

针对component类型进行处理

更新机制

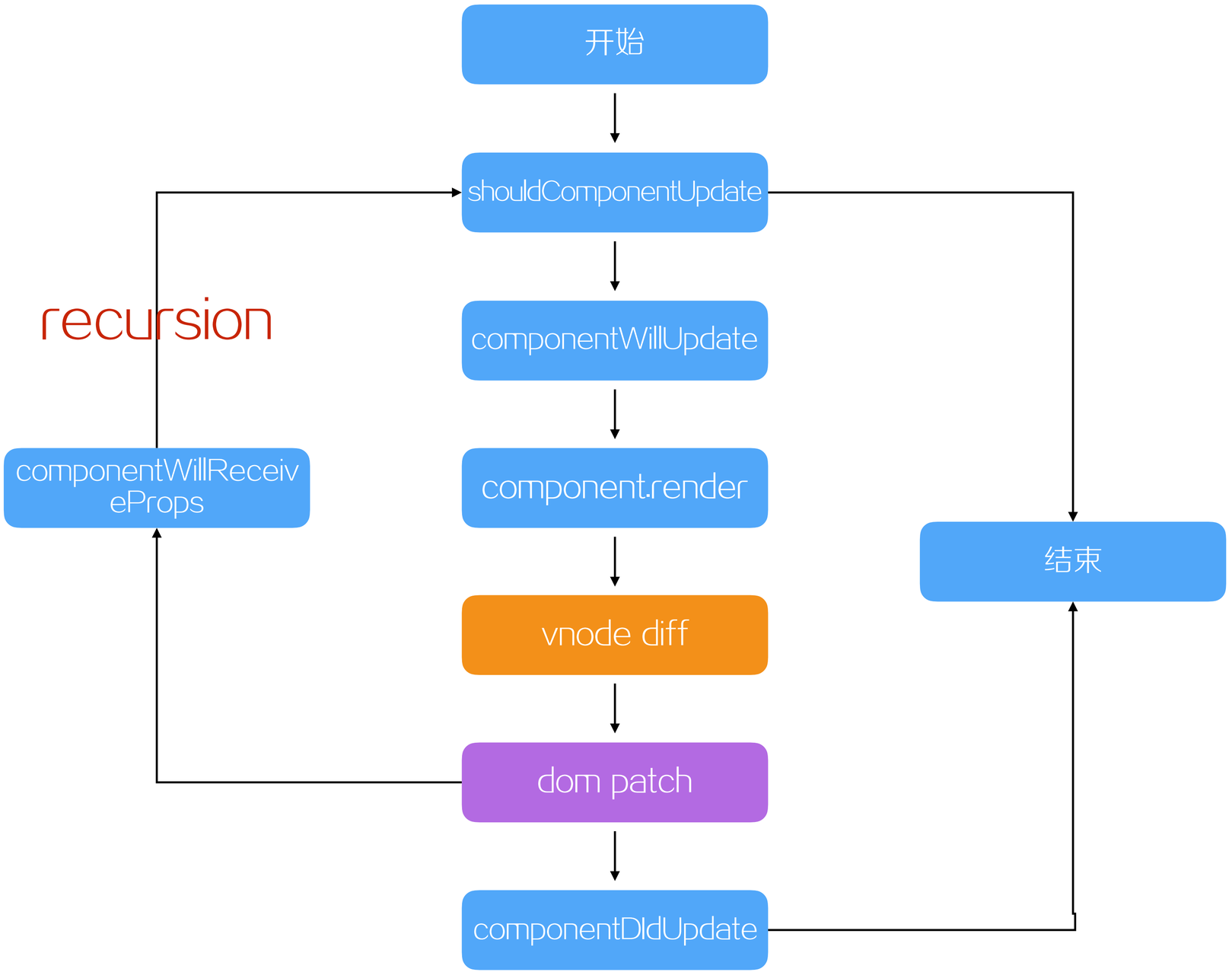
patch

```
function reRenderComponent (prev, curr) {
  const nextProps = curr.props
  if (prev.componentWillReceiveProps) {
    prev.componentWillReceiveProps(nextProps)
  }
  prev.props = nextProps
  updateComponent(prev)
  return prev.dom
}
```

触发子组件的更新

componentWillReceiveProps

更新机制


开始
shouldComponentUpdate
componentWillUpdate
component.render
vnode diff
dom patch
componentDIdUpdate
结束
componentWillReceiveProps
recursion

更新机制

神奇的setState()

```
class SetStateSample extends React.Component {
  constructor () {
    super()
    this.state = {
      count: 0
    }
  }

  componentDidMount () {
    this.setState({ count: this.state.count + 1 })
    console.log(this.state.count)

    this.setState({ count: this.state.count + 1 })
    console.log(this.state.count)
  }

  componentWillUpdate () {
    console.log('componentWillUpdate')
  }

  render () {
    return null
  }
}
```

两次console.log()分别输出什么?

componentWillUpdate输出几次?

更新机制

神奇的setState()

异步

批量

更新机制

神奇的setState()

```
setState (state, callback) {
  if (state) {
    (this._pendingStates = (this._pendingStates || [])).push(state)
  }
  if (typeof callback === 'function') {
    (this._pendingCallbacks = (this._pendingCallbacks || [])).push(callback)
  }
  updateComponent(this)
}

getState () {
  const { _pendingStates = [], state, props } = this
  if (!_pendingStates.length) {
    return state
  }
  const stateClone = Object.assign({}, state)
  const queue = _pendingStates.concat()
  this._pendingStates.length = 0
  queue.forEach(nextState => {
    if (typeof nextState === 'function') {
      nextState = nextState.call(this, state, props)
    }
    Object.assign(stateClone, nextState)
  })
  return stateClone
}
```

将新的state加入队列pendingStates中

需要用到state的时候进行处理

组件更新异步调用

更新机制

神奇的setState()

componentWillMount/componentWillReceiveProps

调用this.setState()不会导致组件更新

通过component._disable来判断是否需要执行组件更新

更新机制

02

神奇的setState()

Under-the-hood-ReactJS

更新机制

02

异步更新机制

componentDidMount

组件异步更新

更新机制

02

异步更新机制

Promise/MutationObserver/setTimeout

依次降级

2

事件处理

事件处理

02

SyntheticEvent

事件处理

SyntheticEvent

事件几乎都代理到document

跨浏览器

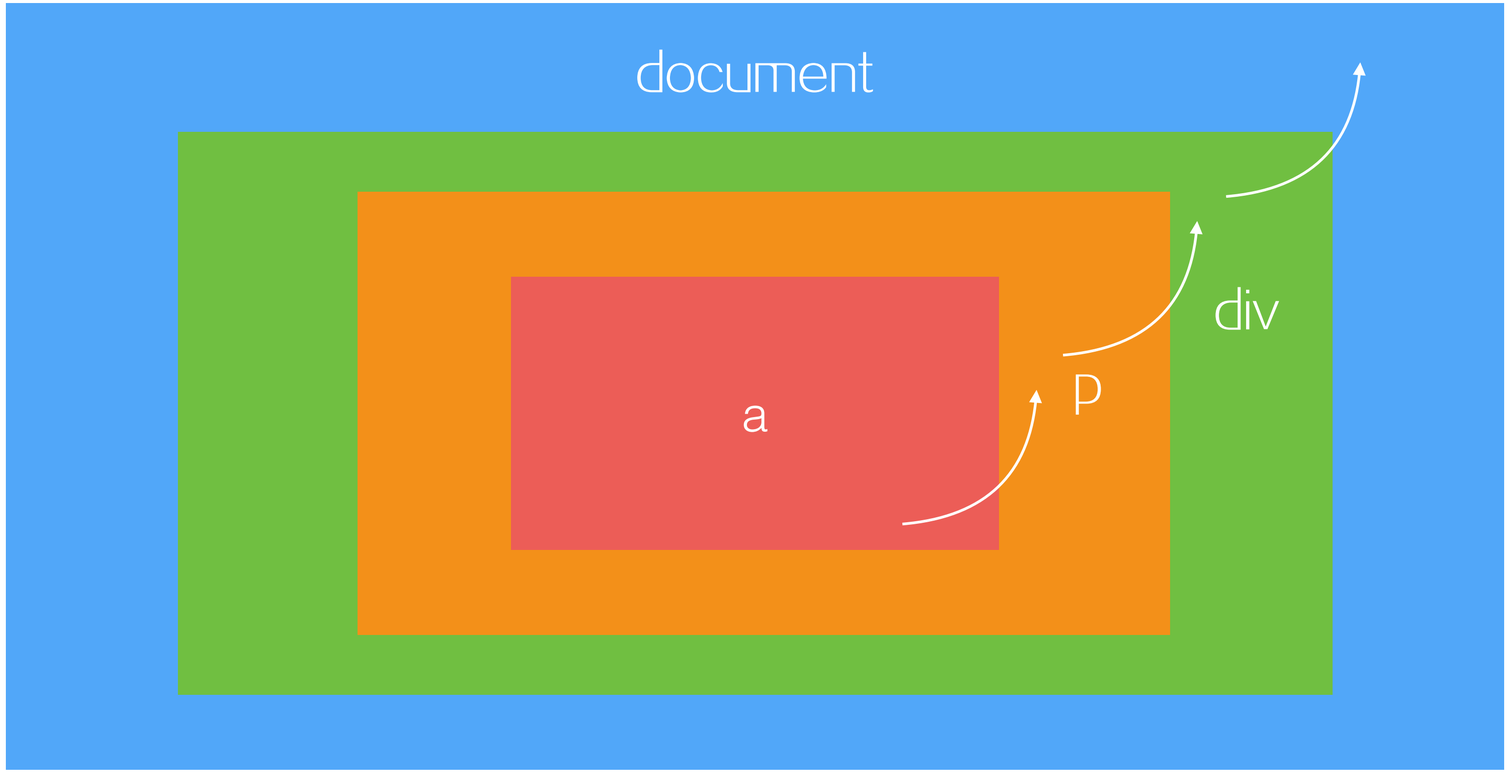
事件对象非原生

事件自动代理自动销毁

实现机制非常复杂！！！

事件处理

事件代理



冒泡机制

事件处理

事件代理

const delegateEvents ={}

```
{
  'onclick': {
    items: Map {
      node: domNode,
      handler: eventHandler
    },
    event: delegateHandler
  }
}
```

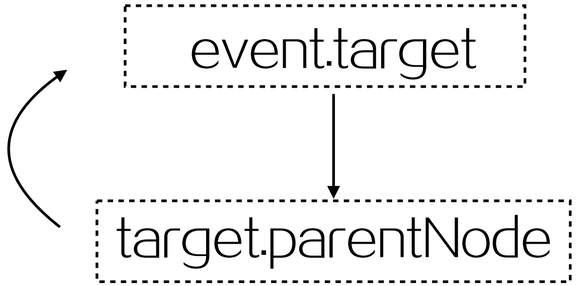
事件处理

02

事件代理

冒泡事件

document.addEventListener
(eventName, delegateHandler, false)



非冒泡事件

node.addEventListener
(eventName, delegateHandler, false)

onmouseenter

onmouseleave

...

3

SVG处理

SVG处理

SVG

```svg
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <circle cx="100" cy="50" r="40" stroke="black" stroke-width="2" fill="red"/>
</svg>
```

XML

SVG处理

SVG

namespace

http://www.w3.org/2000/svg

document.createElementNS(namespace, tagName)

SVG处理

02

SVG

```svg
<svg>
  <use xlink:href='#icon_s' />
</svg>
```

jsx写法

```svg
<svg>
  <use xlinkHref='#icon_s' />
</svg>
```

SVG处理

SVG

```svg
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <circle cx="100" cy="50" r="40" stroke="black" stroke-width="2" fill="red"/>
</svg>
```


```jsx
<svg width='100%' height='100%'>
  <circle cx='100' cy='50' r='40' stroke='black' strokeWidth='2' fill='red' />
</svg>
```

SVG处理

SVG

```
xChannelSelector: 'xChannelSelector',
xlinkActuate: 'xlink:actuate',
xlinkArcrole: 'xlink:arcrole',
xlinkHref: 'xlink:href',
xlinkRole: 'xlink:role',
xlinkShow: 'xlink:show',
xlinkTitle: 'xlink:title',
xlinkType: 'xlink:type',
xmlBase: 'xml:base',
xmlId: 'xml:id',
xmlns: 0,
xmlnsXlink: 'xmlns:xlink',
xmlLang: 'xml:lang',
xmlSpace: 'xml:space',
```

SVG处理

SVG

特殊属性设置

setAttributeNS(namespace, propName, propValue)

```
evEvent: 'http://www.w3.org/2001/xml-events',
xlinkActuate: 'http://www.w3.org/1999/xlink',
xlinkArcrole: 'http://www.w3.org/1999/xlink',
xlinkHref: 'http://www.w3.org/1999/xlink',
xlinkRole: 'http://www.w3.org/1999/xlink',
xlinkShow: 'http://www.w3.org/1999/xlink',
xlinkTitle: 'http://www.w3.org/1999/xlink',
xlinkType: 'http://www.w3.org/1999/xlink',
xmlBase: 'http://www.w3.org/XML/1998/namespace',
xmlId: 'http://www.w3.org/XML/1998/namespace',
xmlLang: 'http://www.w3.org/XML/1998/namespace',
xmlSpace: 'http://www.w3.org/XML/1998/namespace'
```

Nerv

http://github.com/o2team/nerv

课后作业

实现一个可用的组件框架

02

OPEN ORIENTED

凹 凸 实 验 室