

O'REILLY®

Velocity

CONFERENCE

BUILD RESILIENT SYSTEMS AT SCALE

velocity.oreilly.com.cn

#velocityconf

YAHOO! Front Page: Use React with Universal Flux 如何使用React和同构Flux

Lingyan Zhu

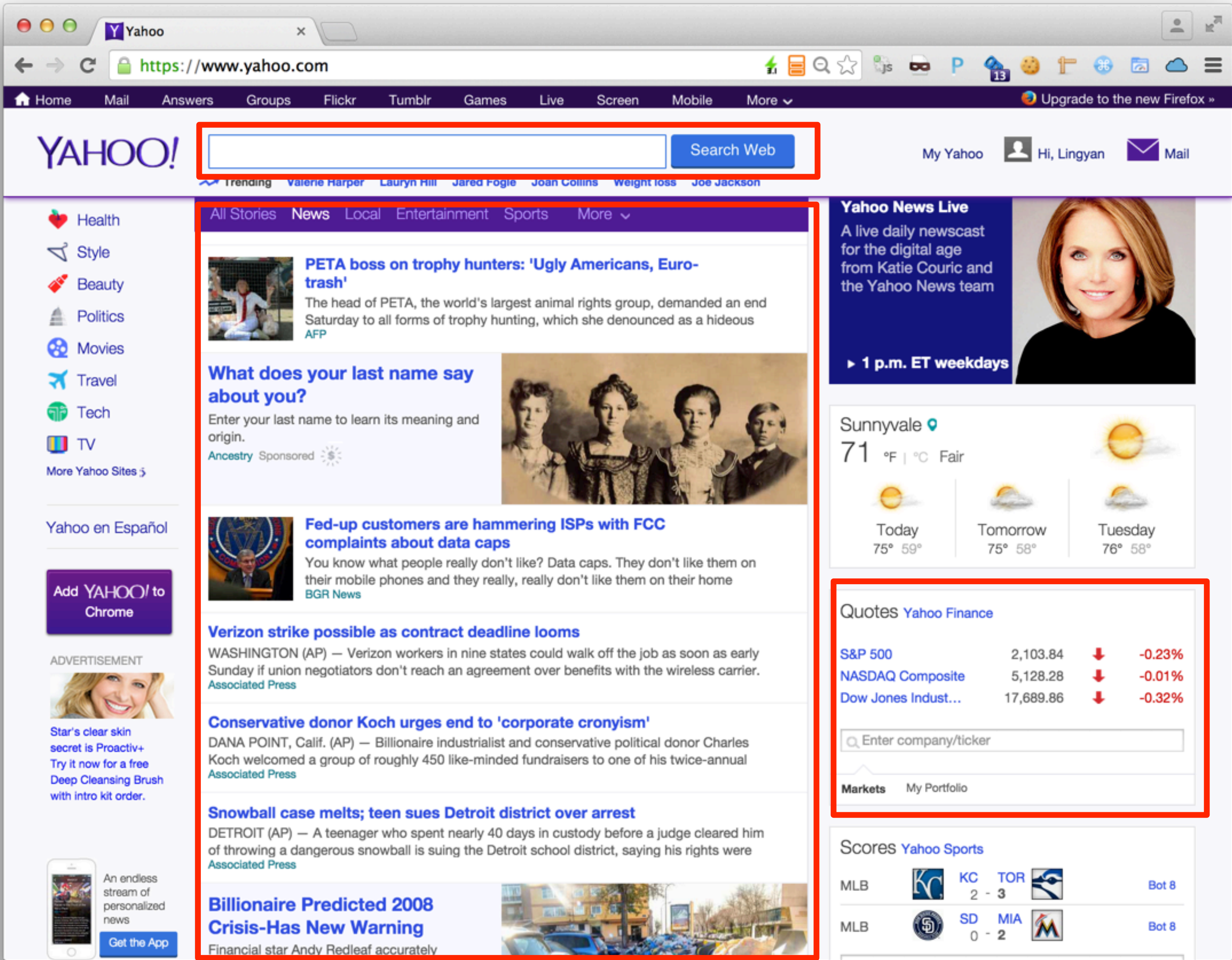
Start with a Poll

- Have you heard of React and Flux before this conference?
- Have you used (or are using) React and Flux for your product(s)?
- Have you used React and Flux on both client and server?

Agenda

- Yahoo! Front Page's main use case
- How do we use React and Flux on both client and server
- Where we are today
- Lessons learned, tips & tricks
 - Developer productivity
 - Performance
 - Other random bits

What is our main use case?



Big Scale

**Super Fast
Page Serving**

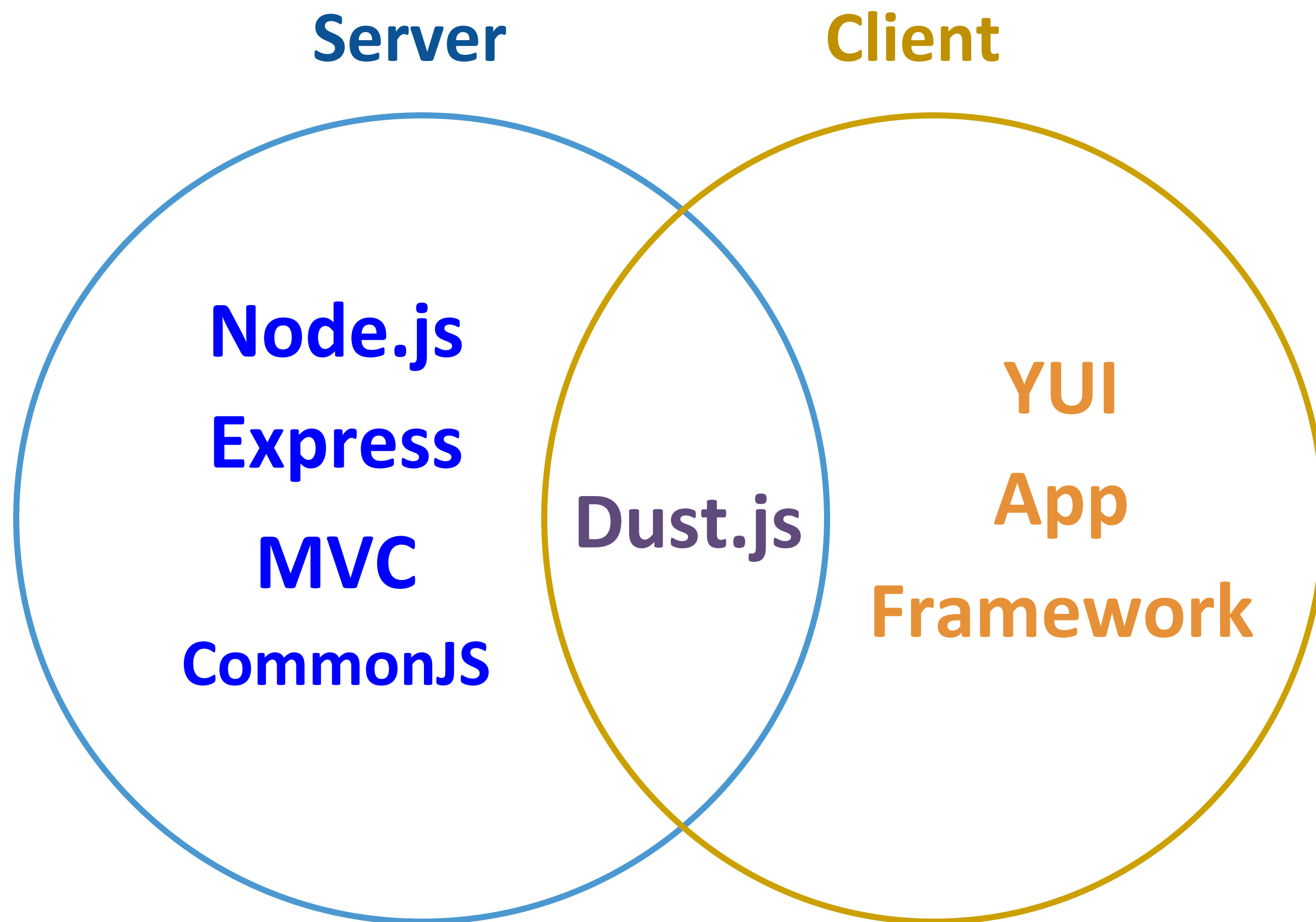
Reliable Page Serving

Fast Interaction

Composable

Live

Prior to React/Flux...



Only share templates between client and server

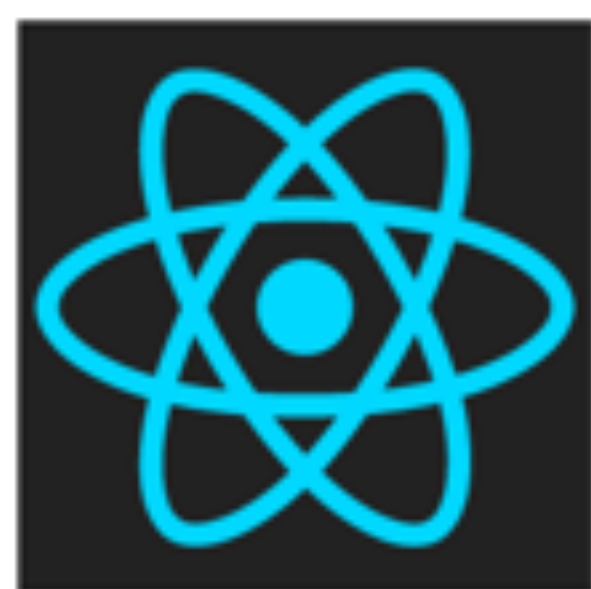
Much better server side performance

Introduced client side navigation for front page on smartphone, for sub-second content consumption

Different MVC on client and server

Client side performance still not ideal

	Developer Productivity	Client Side Performance	Solution
YUI is a Big Abstract Layer	✓	✗	Standard JS API & Polyfills
YUI App Framework	✓	✗	Flux
Different MVC on Client and Server	✗	--	React & Universal Flux



React


```

var Timer = React.createClass({
  getInitialState: function() {
    return {secondsElapsed: 0};
  },
  reset: function() {
    this.setState({secondsElapsed: 0});
  },
  tick: function() {
    this.setState({secondsElapsed: this.state.secondsElapsed + 1});
  },
  componentDidMount: function() {
    this.interval = setInterval(this.tick, 1000);
  },
  componentWillUnmount: function() {
    clearInterval(this.interval);
  },
  render: function() {
    return (
      <div>
        <span>Seconds Elapsed: {this.state.secondsElapsed}</span>
        <button onClick={this.reset}>Reset</button>
      </div>
    );
  }
});

React.render(<Timer />, document.getElementById('example'));

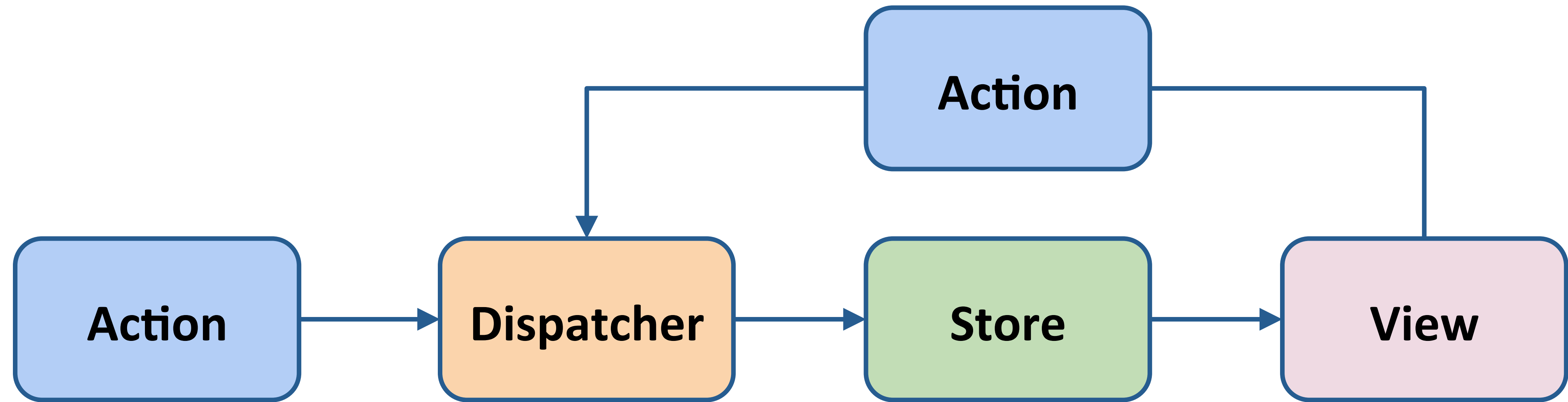
```

- JavaScript, no special templating language. JSX.
- Markup & interaction in the same file
- Smart DOM diff-ing for efficient DOM updates
- Event Delegation
- Composable
- Sharing is easy

<https://facebook.github.io/react/>



- Simple, Plain JavaScript
- Reusable Components
- Well-defined Component Life Cycle
- Declarative
- Virtual DOM
- Server-Side Ready
- Composable
- Built-in Event Delegation
- Truly Universal, esp. with react 0.14



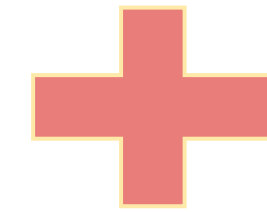
- Single Direction
- Light Weight
- Server Friendly

We want to use the same React and Flux
on server!

**Search Engine
Optimization**



**Legacy
Browser
Support**



**User
Perceived
Performance**

Perfect Match



React

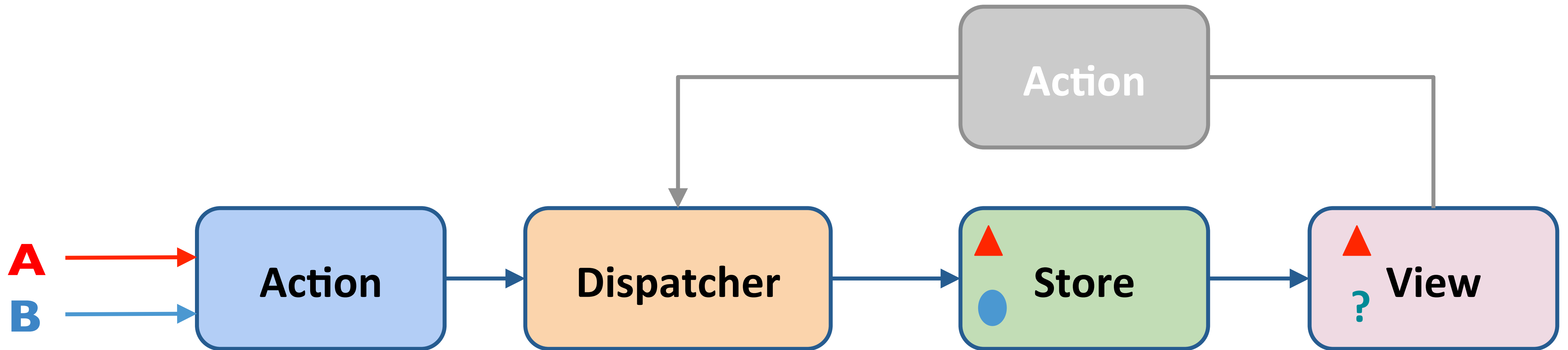


React on the Server

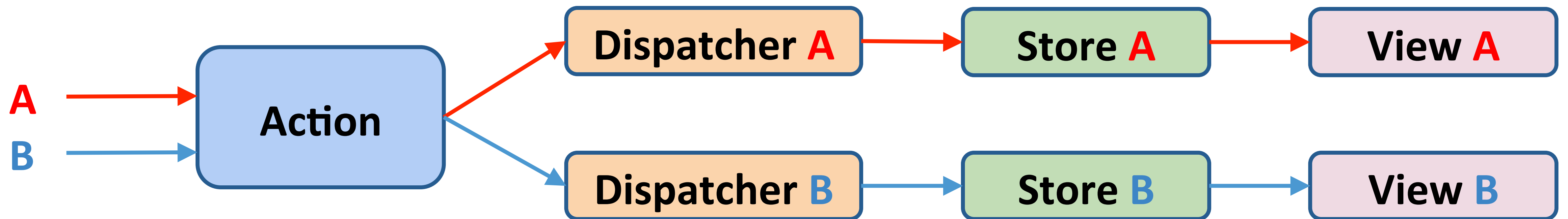
```
// server.js
var express = require('express');
var server = express();
var Timer= require('./Timer');

server.get('/', function (req, res, next) {
  var html = React.renderToString(Timer());
  res.header('Content-Type', 'text/html;
  charset=utf-8');
  res.send(html);
});
server.listen(8080);
```

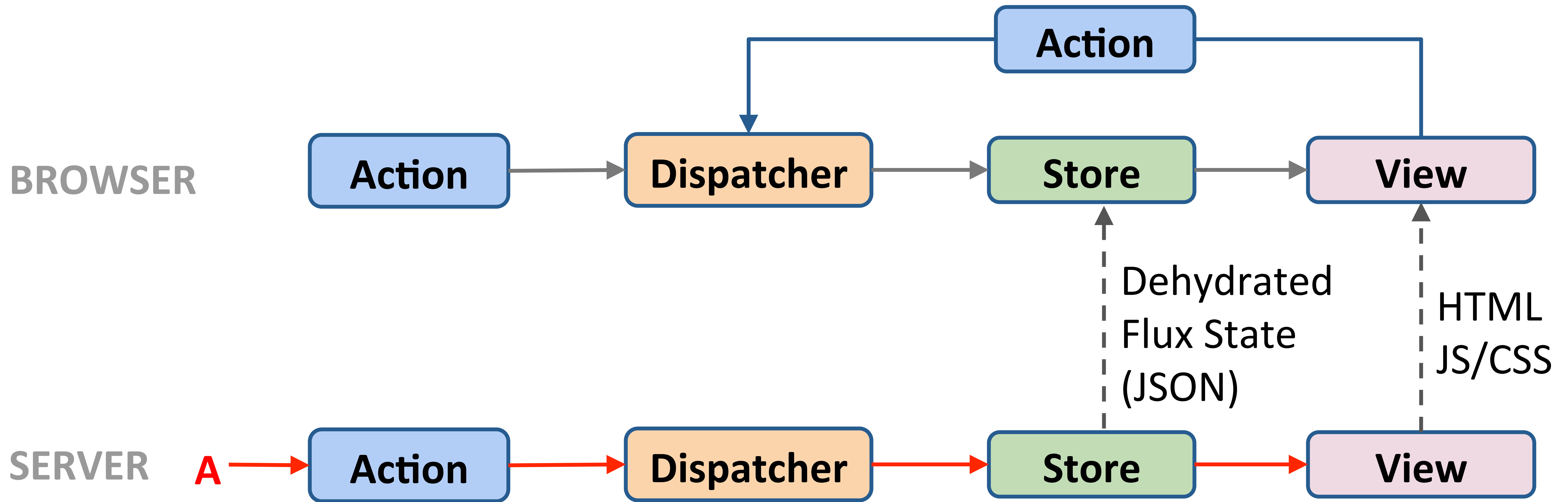
Bring Flux to the Server



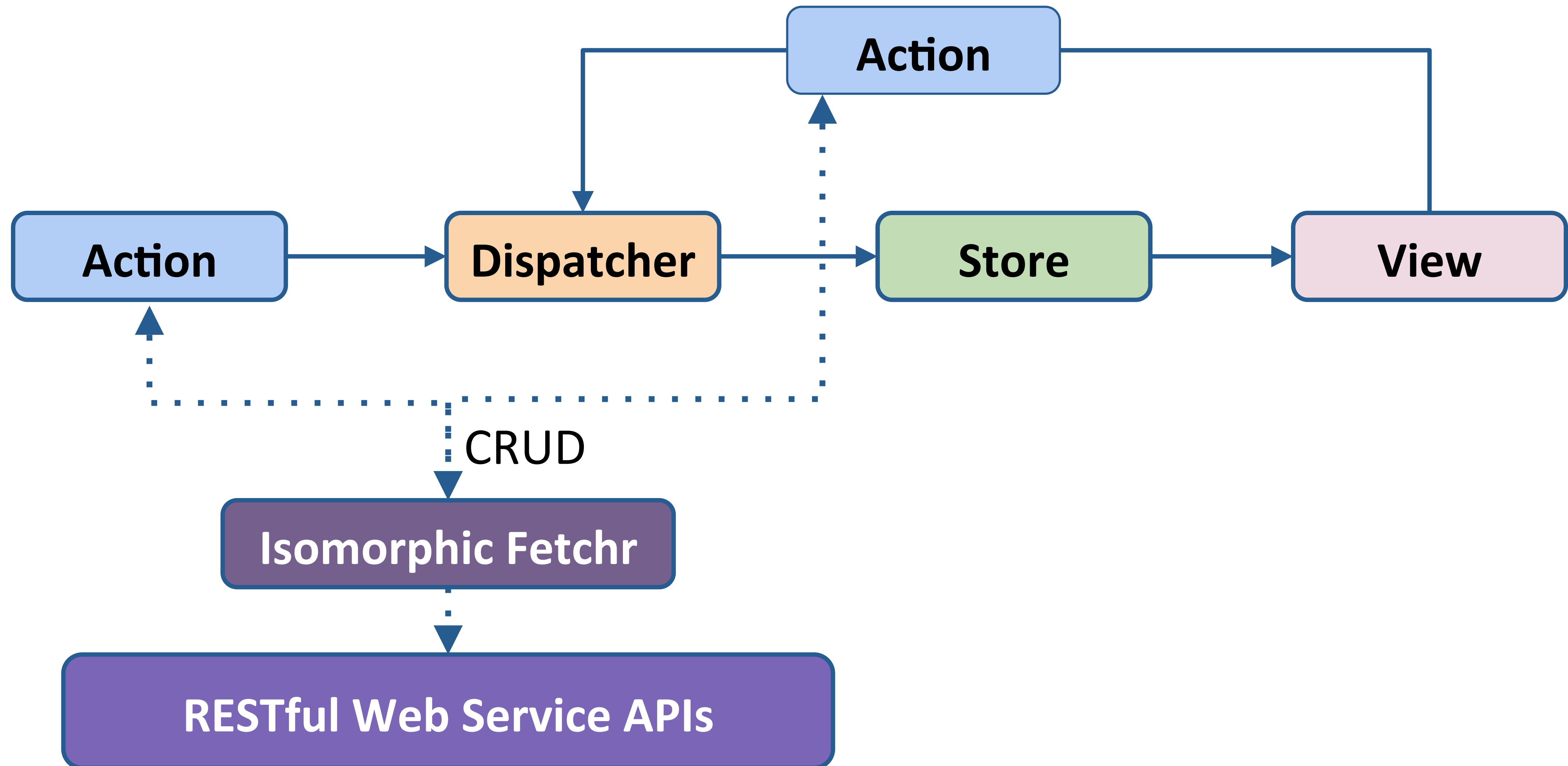
Bring Flux to the Server



Bridging Server and Client



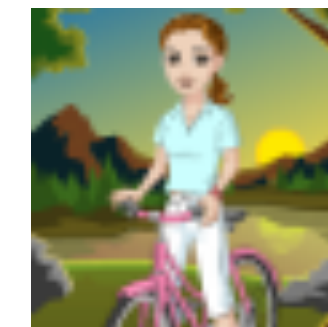
Accessing Data





`http://fluxible.io`

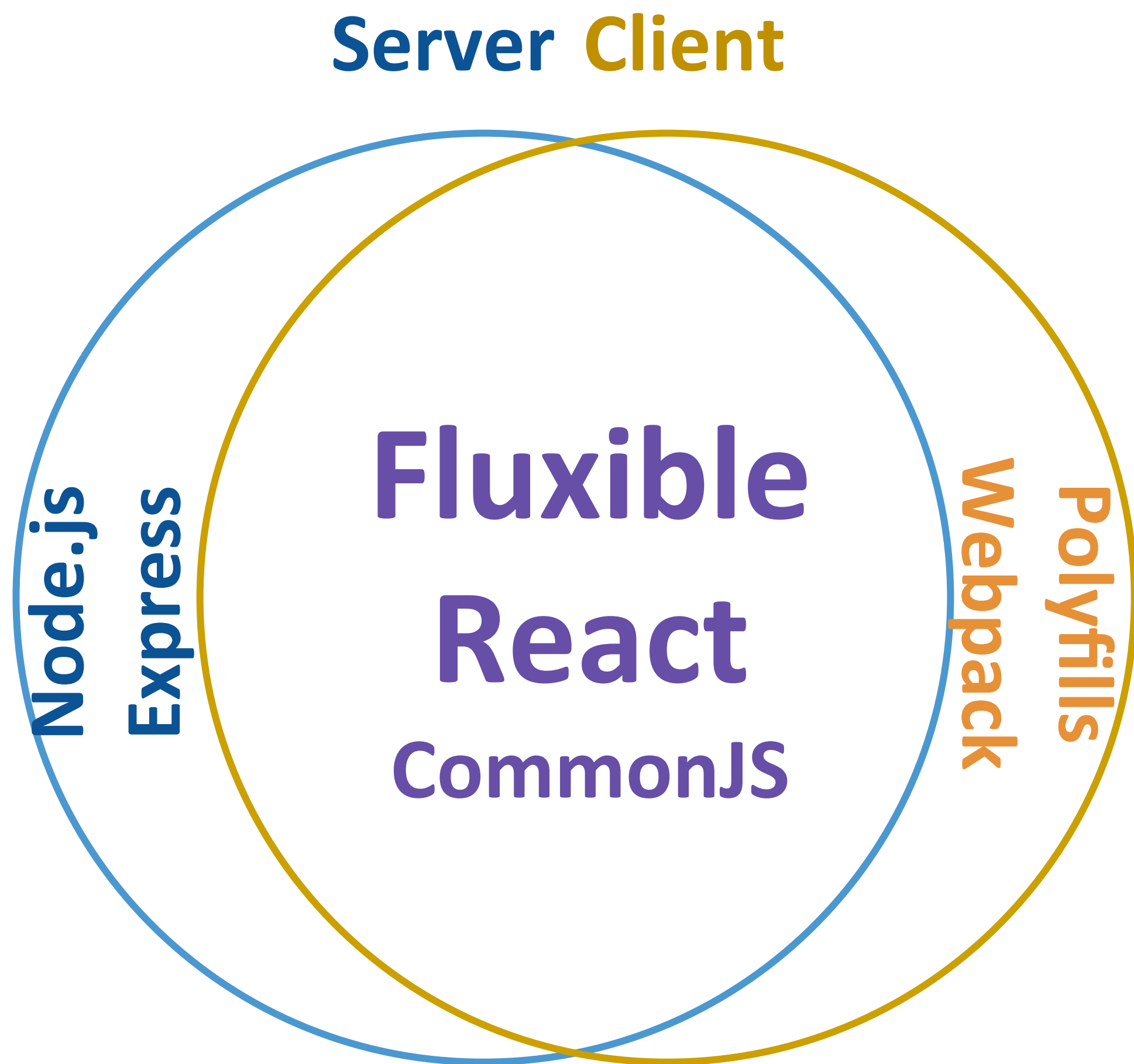
Core Fluxible Libraries on <https://github.com/yahoo>:
dispatchr, fetchr, fluxible, fluxible-router, routr



Additional React related Libraries on <https://github.com/yahoo>:

react-i13n, react-intl

With React & Universal Flux



Same MVC and routing logic on client and server

Better client side performance

Good server side performance, continue improving

Open source contributions:

- <http://fluxible.io>
- <https://github.com/yahoo/react-i13n/>
- <https://github.com/yahoo/react-intl/>
(by our previous YUI team)

Yahoo Sites Using this stack

Already in production:

- Taiwan Mobile Front Page: <https://tw.mobi.yahoo.com/>
- Sports Daily Fantasy App: <https://sports.yahoo.com/dailyfantasy>
- Beauty digital magazines: <https://www.yahoo.com/beauty>
- International magazines, e.g.:
 - <https://uk.celebrity.yahoo.com/>
 - <https://uk.style.yahoo.com/>
 - <https://fr.people.yahoo.com/>
- Search SDK App: <https://developer.yahoo.com/search-sdk/apps/>

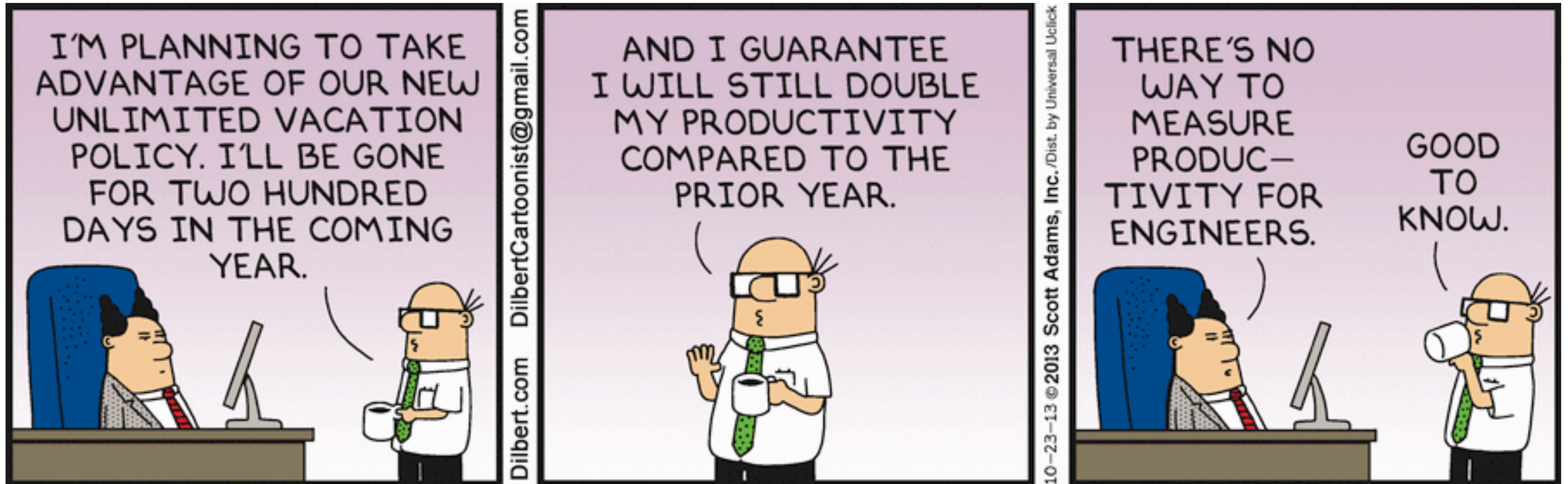
Still improving to meet US front page requirements.

Community Fluxible Sites/Apps

- Docker Hub (Beta): <https://hub-beta.docker.com/>
- Porch: <https://porch.com/>
- <https://blog.cesarandreu.com/>
- <https://www.mailjet.com/passport>
- <https://dice.fm/>
- ...

It's been a year. What have we learned?

Developer Productivity



Developer Productivity

**No more
context switching**

**Component sharing
made easier by React**

**No more data binding
black magic**



Scaffolding
with Yeoman



Managing build
tasks with Grunt for
our internal projects

Build and Deploy



Common Grunt tasks
provided by
framework team

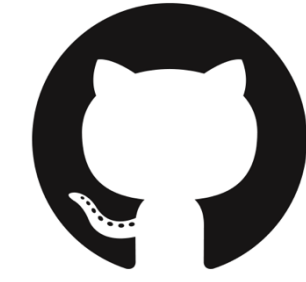


Bundling our
CommonJs modules
with webpack



Internal npm registry
hosting internal
packages

Internal Open Source



“I think one thing we do very well with Node is integrate other people into the community.

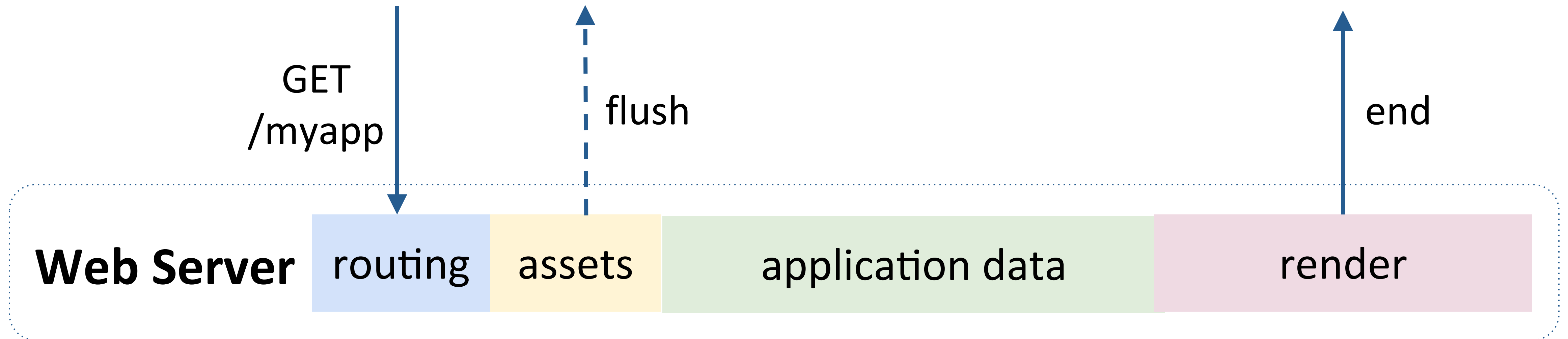
We’re very ***open*** about what we’re working on and trying to bring people into the project.”

- Ryan Dahl

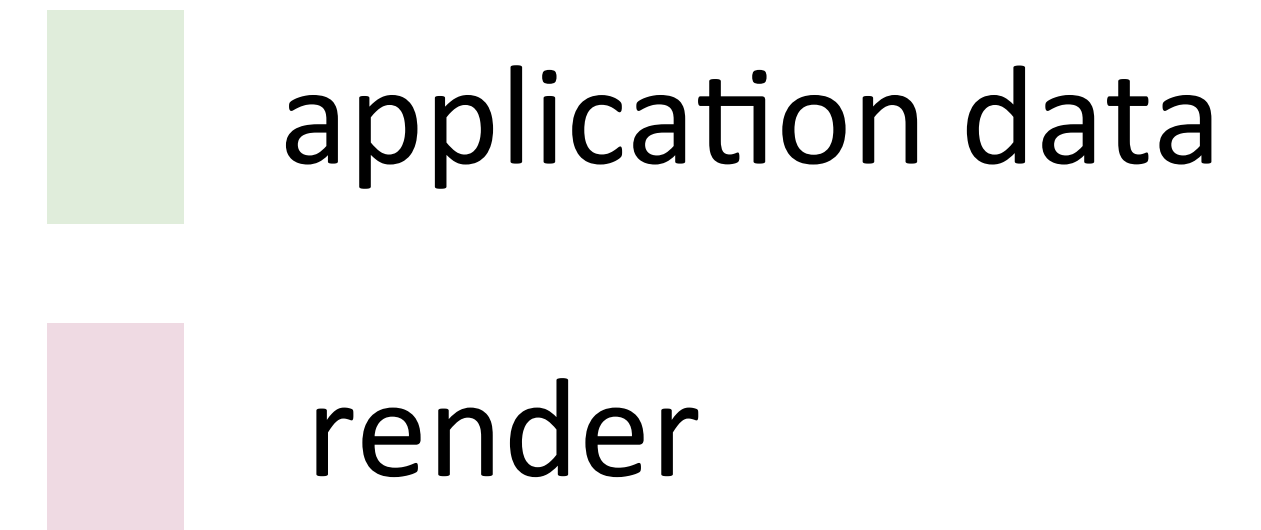
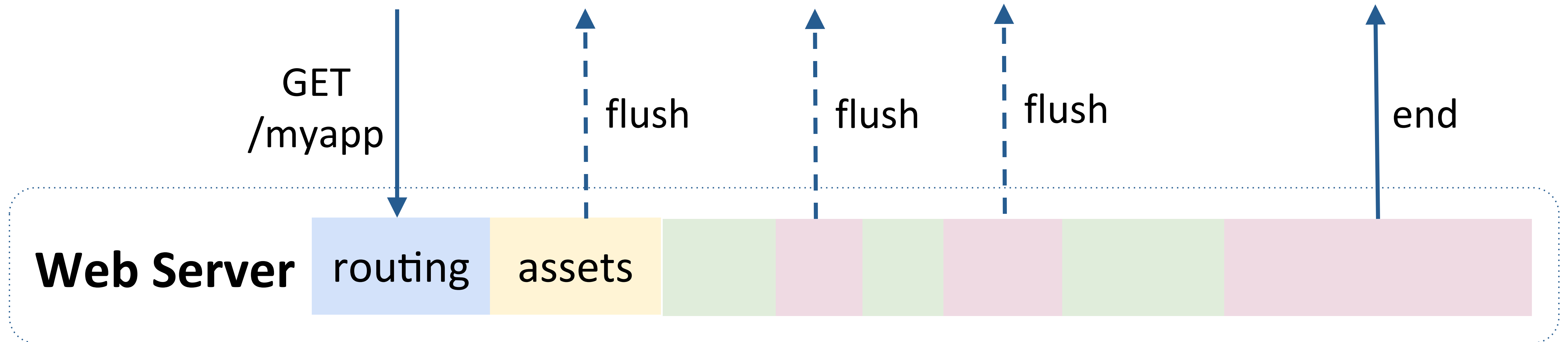
Performance



TTFB



TTFB



Even faster TTFB with ESI

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="http://mycdn.com/1.css">
    <script async src="http://mycdn.com/1.js"></script>
  </head>
  <body>
    <div>Cacheable content</div>
    <esi:include src="http://example.com/1.html" alt="http://
bak.example.com/1.html" onerror="continue"/>
    <esi:include src="http://example.com/2.html" alt="http://
bak.example.com/2.html" onerror="continue"/>
  </body>
</html>
```

Node.js is Single-Threaded

- Great for async I/O tasks
- CPU intensive tasks impact throughput

React Rendering on the Server

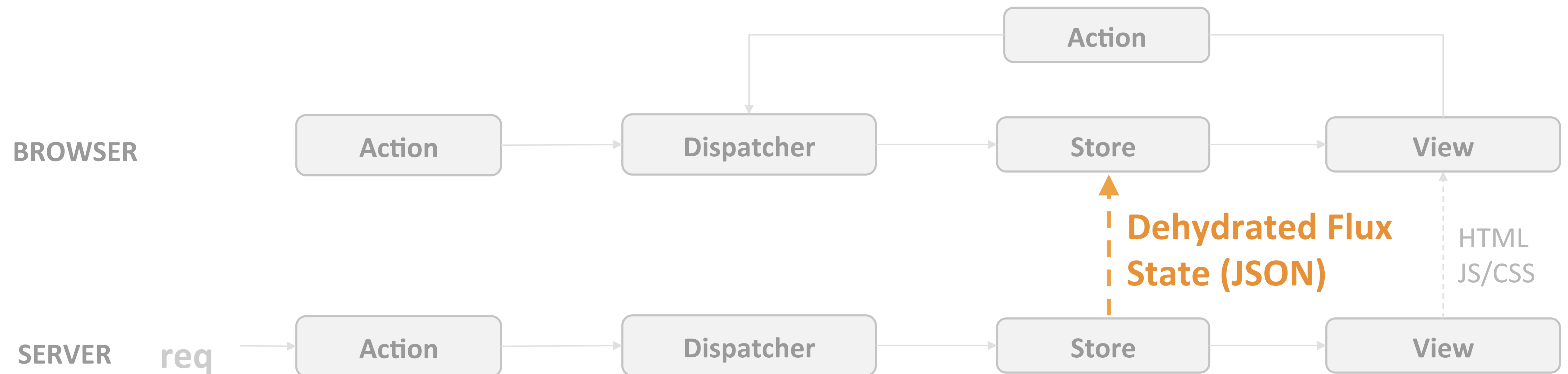
Problems:

- Not originally intended for server side
- As of react 0.13, same rendering path for client and server, i.e. client-only logic also executes on server
- CPU intensive

Tips & Tricks:

- Use minified React for production: 20-30% gain
- Avoid rendering excessive components
- Less component hierarchy for heavily-used components
- Will collaborate with React team on server side performance enhancements

Dehydrated Data Size



Impact:

- CPU intensive
 - JSON serialization
 - Compression
- Memory
- Page weight

Solution:

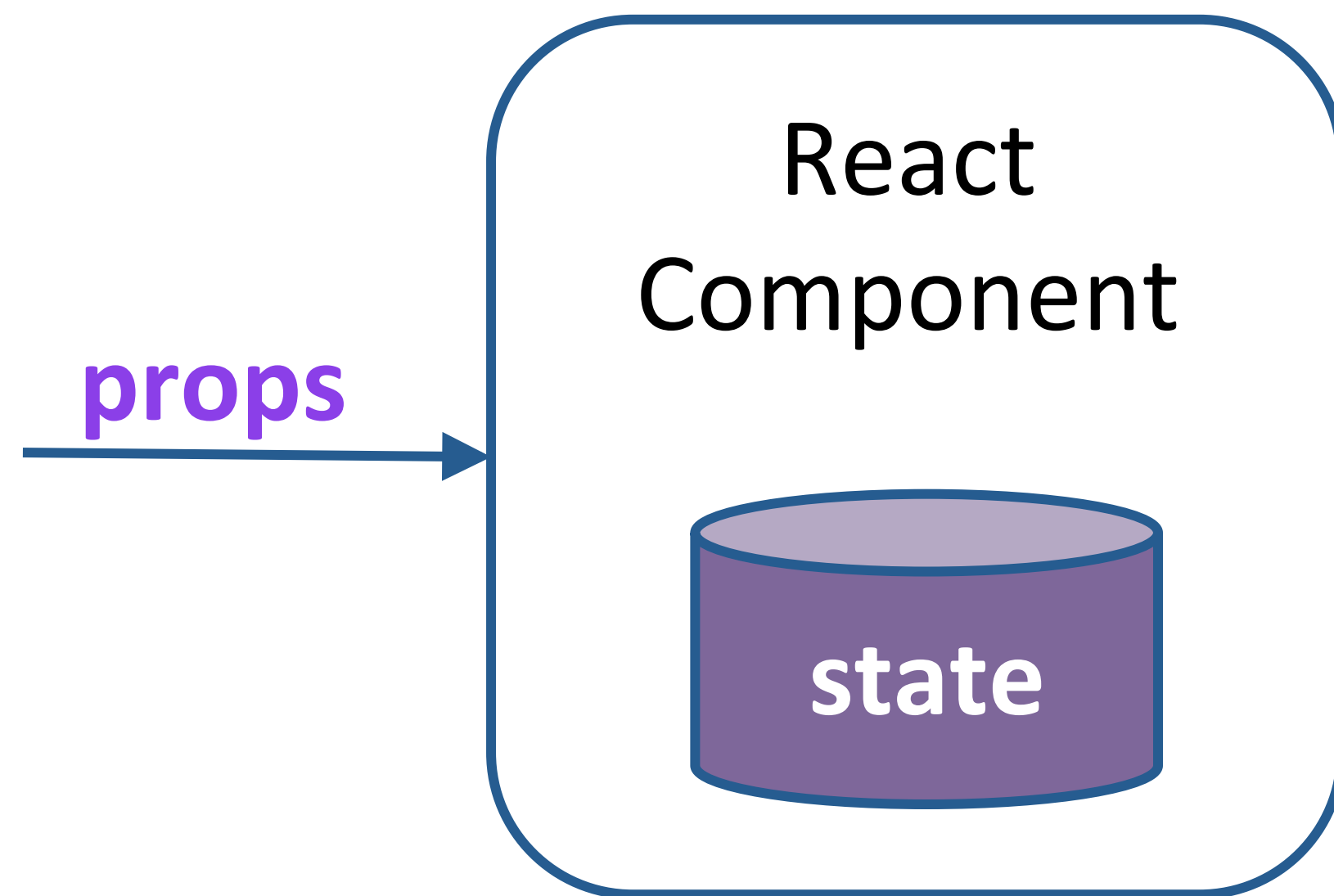
- Do not over-fetch data
- Do not over-dehydrate data
- Design your stores wisely
 - Do not duplicate data

What about Client Side?

Things you Already Know...

- Look out for asset sizes
- Only ship assets you need
- Above-the-fold time (AFT)
- SPDY and HTTP/2

React: Avoid Excessive Re-rendering



- By default, React always re-render the component on prop/state change
- Keep minimal amount of data in the state
- Find over-rendering using React's `Perf.printWasted()`
- Use `shouldComponentUpdate()` to control when to re-render
- Consider using Immutable JS
- Especially important for components with a lot of instances, and components at the top of the hierarchy

```
var Timer = React.createClass({
  getInitialState: function() {
    return {secondsElapsed: 0};
  },
  tick: function() {
    this.setState({secondsElapsed: this.state.secondsElapsed + 1});
  },
  componentDidMount: function() {
    this.interval = setInterval(this.tick, 1000);
  },
  componentWillUnmount: function() {
    clearInterval(this.interval);
  },
  shouldComponentUpdate(nextState, nextProps) {
    // render every other tick
    var sinceLast = nextState.secondsElapsed - this.state.secondsElapsed;
    return sinceLast % 2 !== 0;
  },
  render: function() {
    return (
      <div>Seconds Elapsed: {this.state.secondsElapsed}</div>
    );
  }
});
React.render(<Timer />, document.getElementById('example'));
```

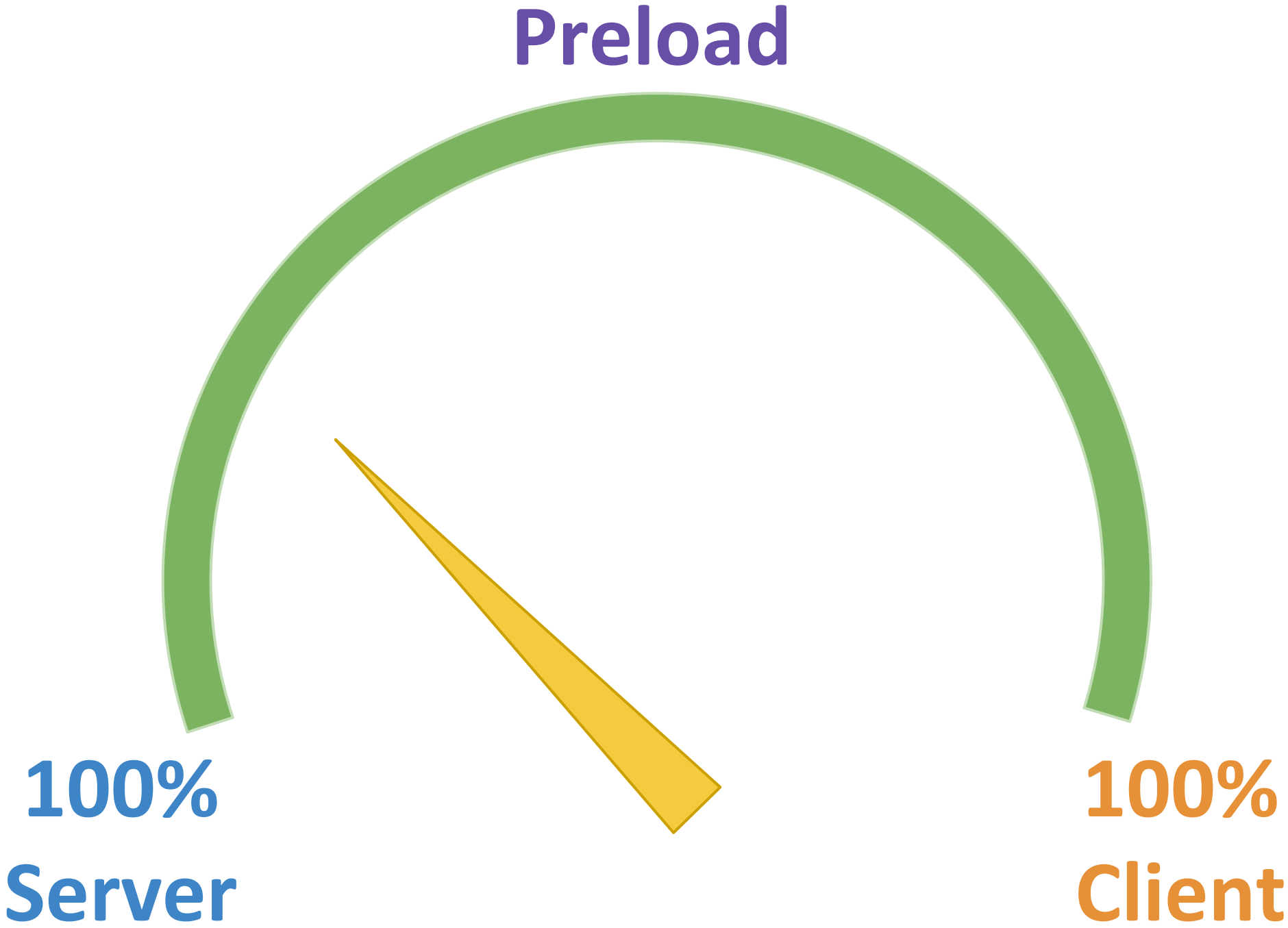
Fast Client Side Navigation

**Optimistically
Render New Route**



**Prefetch
Data & Assets**

Progressive Rendering



	Fetch Data	Render	Example Use Case
100% Server	server	server	Critical modules above the fold
Preload	server	client	Immediately below the fold
100% Client	client	client	Far-bottom or invoked by user action

Imagine: You can start at 100% server and dial your app to more client side, to cope with sudden traffic spike :)

What else?

Deduping Node Modules

Problem:

- Duplicated versions of node modules in the dependency tree

Impact:

- Big memory footprint on node server

Solution:

- use semver wisely
- hoist common packages to root level
- Automate monitoring
- `npm dedupe` is still experimental
- npm 3 will flatten dependency tree. Could be helpful.

Lodash

- Require sub modules directly:

```
require('lodash').merge; // 105KB/minified to browser
```

```
require('lodash/object/merge'); // 17KB/minified to browser
```

- Watch out for `lodash.merge` - It's recursive!
 - Avoid merging big JSON object unless you have to
 - Especially if you are using `lodash.merge` for every request

```
_.merge(object, [sources], [customizer], [thisArg])
```

React Checksum


SERVER

```
<div id="myApp">
  <!-- React.renderToString(React.createElement(MyApp, data)) returns
        checksum generated by using Adler32 from the inner HTML string -->
  <div data-reactid=".pfj8qu0bgg" data-react-checksum="-1057226902">
    ...
  </div>
</div>
```

BROWSER

```
// React renders the markup string on the browser, computes the checksum
// from the markup string, checks the existing checksum from the server
// markup.
// If checksum does not match, React blows away server generated
// markup and rerender entire DOM in browser.
React.render(
  React.createElement(MyApp, data),
  document.getElementById("myApp"));
```

React Checksum

 Warning: React attempted to reuse markup in a container but the checksum was invalid. This generally means that you are using server rendering and the markup generated on the server was not what the client was expecting. React injected new markup to compensate which works but you have lost many of the benefits of server rendering. Instead, figure out why the markup being generated is different on the client or server:
(client) Header.0">Render on **client.**
<div i
(server) Header.0">Render on **server.**
<div i

Common Causes:

- React component's render() is using input besides props and state
- Store data was not dehydrated properly

Impact:

- Could cause jarring user experience

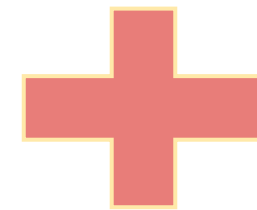
Solutions:

- Fix root cause
- Catch with functional tests

```
var Timer = React.createClass({
  render: function() {
    var silly = (typeof window === 'undefined') ? 'server' : 'client';
    return ( <span>Render on {silly}</span> );
  }
});
```

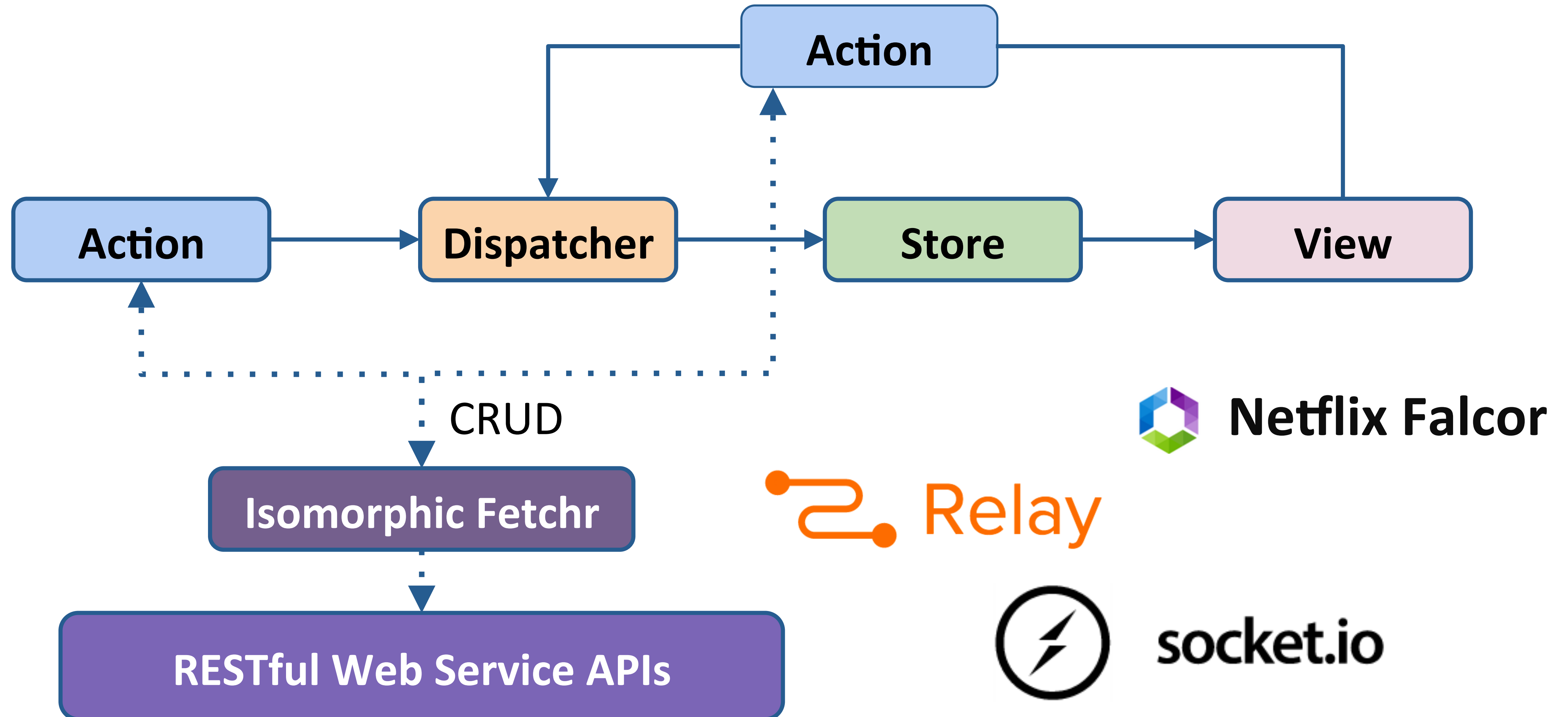
Low Grade Browser

**100%
Server-side
Rendering**



**Super Simple
Client-side
Interaction**

More about Accessing Data



Obvious Things:

- Profile before Optimize
- Automate Benchmarking, Profile Regularly:
 - Your application's characteristics do change over time.
(E.g. Size of your application data can grow over time with new features)

What about CSS

We use: **<http://acss.io/>**



 **<https://github.com/yahoo/atomizer>**



Thank you...

Lingyan Zhu

Front End Architect, Yahoo! Front Page



lingyan



@lingyanzhu