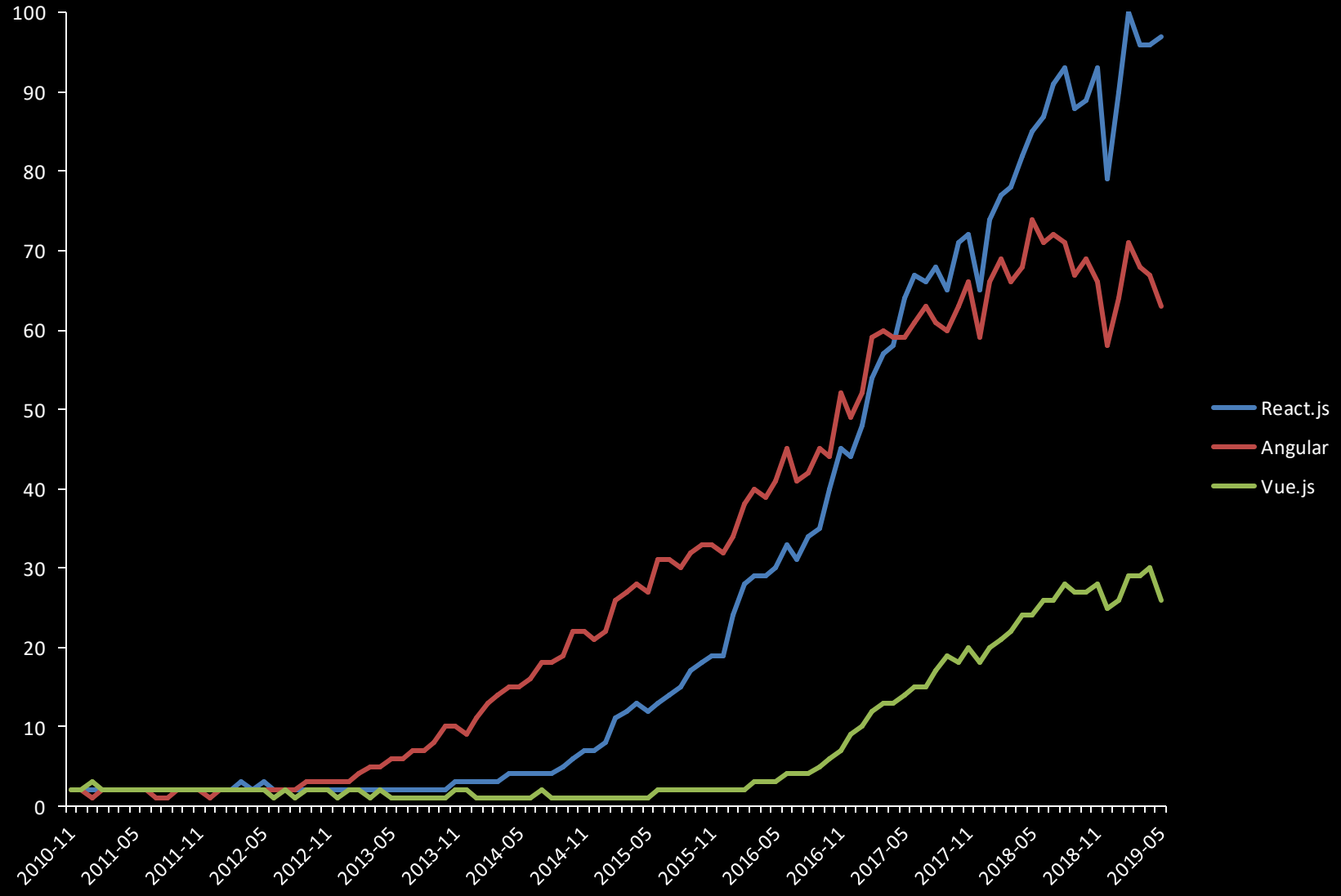
Internet Applications

React

JavaScript frameworks landscape



Key characteristics

- A library for creating UIs
- Created and sustained by Facebook
- Declarative
- Based on components
- Everything is written in JavaScript

Who uses React?

as well as **hundreds** of other companies…

Hello, world!

```
<div id="root"></div>

<script type="text/babel">
  const rootElement = document.getElementById('root’);
  const root = ReactDOM.createRoot(rootElement);
  root.render(<h1>Hello, world!</h1>);
</script>
```

JSX

- JavaScript expression

```
const element = <h1>Hello, world!</h1>;
```

- Expressions in curly braces

```
const element = <h1>2 + 2 = {2 + 2}</h1>;
```

- Attributes

```
const element = <img src={user.avatarUrl}></img>;
```

- camelCase notation for attributes (e.g., className)

JSX

- Text displayed as string
- Transpilation

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

Component

- A function returning a React element

```
function Welcome({name}) {
  return <h1>Hello, {name}</h1>;
}
```

Component

- Usage

```
const element = <Welcome name="Maciej" />;
```

- Names begin with capital letters
- Passing data using arguments
- Has to be a pure function w.r.t. arguments (they are read-only)

State

```
function User({userName}) {
  const [counter, setCounter] = React.useState(0);

  return (
    <div>
      Welcome, {userName}!
    </div>
  );
}


const rootElement = document.getElementById('root');
const root = ReactDOM.createRoot(rootElement);
root.render(<User userName="Maciej" />);
```

Events

```
function User({userName}) {
  const [counter, setCounter] = React.useState(0);

  const count = () => {
    setCounter(counter + 1);
  };

  return (
    <div>
      <div>
        Welcome, {userName}!
        This is your {counter} click.
      </div>
      <button onClick={count}>Count</button>
    </div>
  );
}
```

Lists

```
function User({name}) {
  return <li>{name}</li>
}

function UsersList({users}) {
  return (
    <div>
      <h1>List of users</h1>
      <ol>
        {users.map(user => <User key={user} name={user} />)}
      </ol>
    </div>
  )
}

const rootElement = document.getElementById("root");
const root = ReactDOM.createRoot(rootElement);
root.render(
  <UsersList users={["Dante", "Patrokles", "Piotr"]} />
);
```

Forms

- The problem with state
- Controlled components

Forms

```
function UserForm() {
  const [name, setName] = React.useState('');

  const handleChange = (event) => {
    setName(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    //...
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={name} onChange={handleChange} />
      <input type="submit" value="Login" />
    </form>
  );
}
```

Component lifecycle

```
function MyComponent() {
  React.useEffect(() => {
    // Code to run after component mounts
    console.log('Component mounted');

    // Optional cleanup function
    return () => {
      console.log('Component unmounting');
    };
  }, []); // Runs once after initial render

  return <div>My Component</div>;
}
```

Component lifecycle

```
function UserProfile({ userId }) {
  const [userData, setUserData] = React.useState(null);

  React.useEffect(() => {
    console.log('userId changed to:', userId);
    fetchUserData(userId).then(data => setUserData(data));
  }, [userId]); // Runs when these values change

  return <div>{userData ? userData.name : 'Loading...'}</div>;
}
```

Virtual DOM

1. Virtual DOM gets updated.
2. Comparing current V-DOM with previous version.
3. React looks for changes.
4. Changes are propagated to true DOM.
5. Changes in DOM cause change in the application interface.

Quickstart

- https://react.dev/learn/installation