React Performance Playbook

Introduction to React Performance Optimization

Kamran Ayub

Helping React developers create snappy apps

@kamranayub | www.kamranayub.com

Version Check

Live Video

Live Video

Web performance optimization is no longer a choice.

Live Video

100ms = **1%**

latency fewer sales

Greg Linden (Amazon), 2009. "Make Data Useful"

1s latency = **7%** loss in conversion

Akamai, 2015

1s latency = **7%** loss in conversion

For an e-commerce store doing $100,000 per day in revenue,

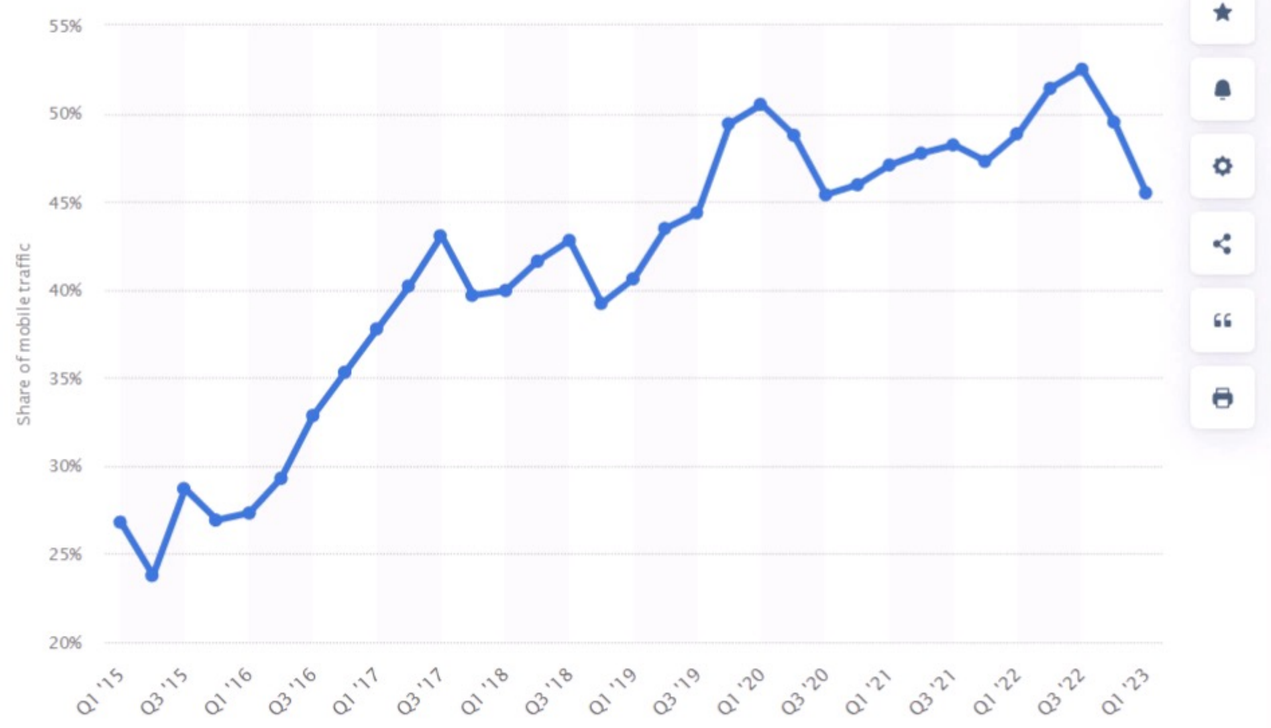
that's **$2.5M** lost per year

Akamai, 2015

Internet › Reach & Traffic

PREMIUM

Percentage of mobile device website traffic in the United States from 1st quarter 2015 to 1st quarter 2023



© Statista 2023

Additional Information

Show source

DOWNLOAD

PDF XLS PNG PPT

Source

→ Show sources information

→ Show publisher information

→ Use Ask Statista Research Service

Release date

April 2023

Region

United States

Survey time period

Q1 2015 to Q1 2023

Special properties

excluding desktop and tablet devices

Citation formats

→ View options

Percentage of mobile device website traffic in the United States from 1st quarter 2015 to 1st quarter 2023, Statista

20%

Reduction in abandonment rates

since Google made page speed a ranking factor

Google, 2019, "User experience improvements with page speed in mobile search"

When page load time increases from...	bounce probability increases by...
1s to 3s	+32%
1s to 5s	+90%
1s to 6s	+106%
1s to 10s	+123%

Google/SOASTA Research, 2017

Better performance means better SEO visibility

1s more load time = **4.6%** fewer views

Financial Times, 2016. "A Faster FT.com"

3s more load time = **7.9%** fewer views

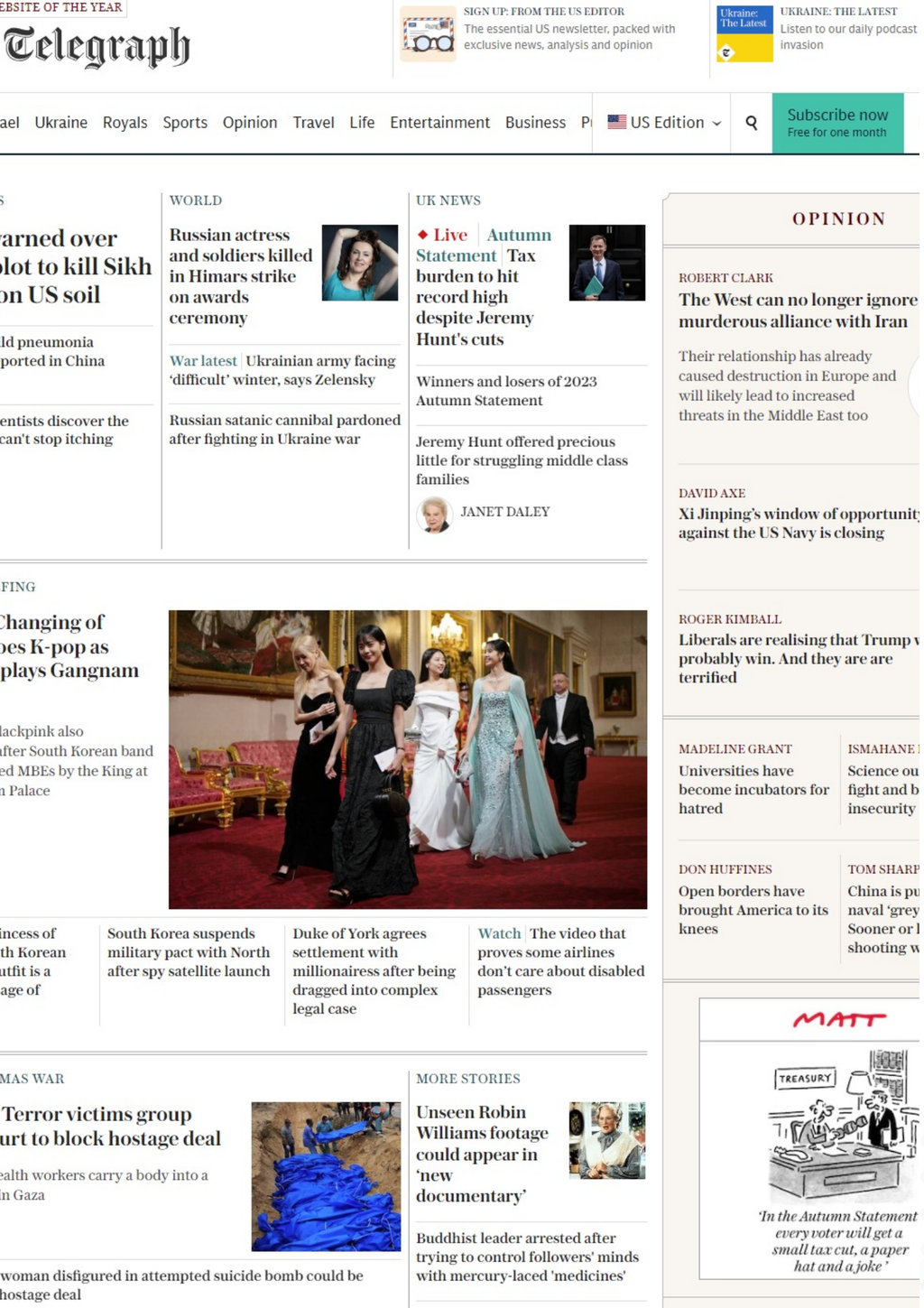
Financial Times, 2016. "A Faster FT.com"

1s more load time = **10%** fewer visitors

BBC, 2018. "How the BBC Builds Websites That Scale"

Live Video

Use data-driven decision making to justify performance optimizations.



Optimizely and Telegraph Performance Case Study

Stakeholders wanted to keep adding third-party tracking scripts

Engineering team wanted to model how performance affects revenue

Artificially slowed down page load through A/B testing

A page load delay of...

4s ⋯⋯⋯⋯▸

decreased page views by...

-11.02%

Optimizely, 2018. "How Does Page Load Time Impact Engagement?"

A page load delay of…	decreased page views by…
4s	-11.02%
8s	-17.52%
16s	-20.53%
20s	-44.19%

Optimizely, 2018. "How Does Page Load Time Impact Engagement?"

Live Video

KPI	Dealer Masters (Built on Gatsby)	Average Competitor	Percent Difference
Bounce Rate	28.75%	47.85%	-39.92%
Overall Load Speed	3.43s	7.43s	-53.81%
CPC Bounce Rate	41.4%	58.37%	-29.04%
Organic Bounce Rate	23.11%	29.94%	-22.82%

GatsbyJS, 2022. "What We Learned About Web Performance From Building 100+ Auto Dealership Websites With Gatsby"

1s = **1.6%**

more load time higher bounce rate

GatsbyJS, 2022. "What We Learned About Web Performance From Building 100+ Auto Dealership Websites With Gatsby"

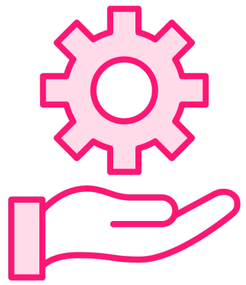
Live Video

Business Impact of Performance Optimization

Better brand perception

Business Impact of Performance Optimization

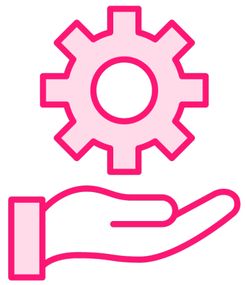
Better brand perception



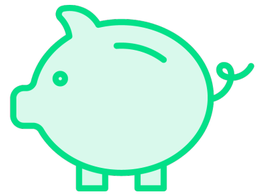
Lower tech debt

Business Impact of Performance Optimization

Better brand perception



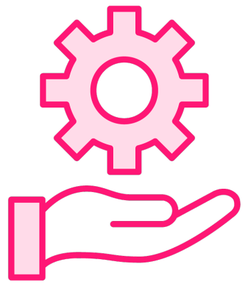
Lower tech debt



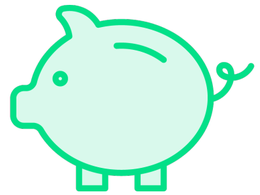
Lower operational cost

Business Impact of Performance Optimization

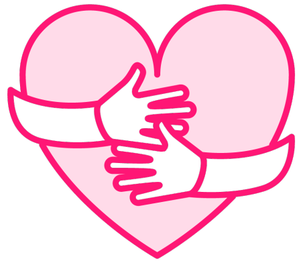
Better brand perception



Lower tech debt



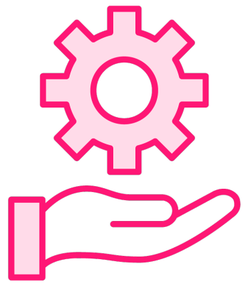
Lower operational cost



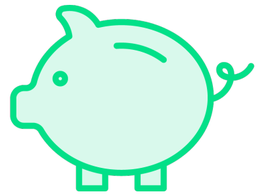
Deeper engagement

Business Impact of Performance Optimization

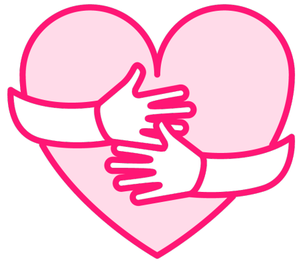
Better brand perception



Lower tech debt



Lower operational cost

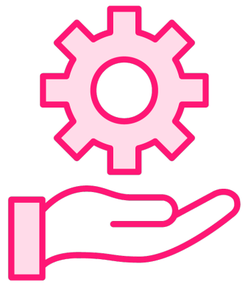


Deeper engagement

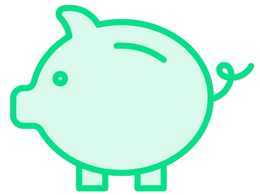
Higher conversion

Business Impact of Performance Optimization

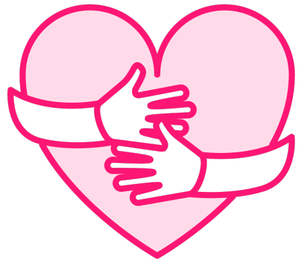
Better brand perception



Lower tech debt



Lower operational cost



Deeper engagement

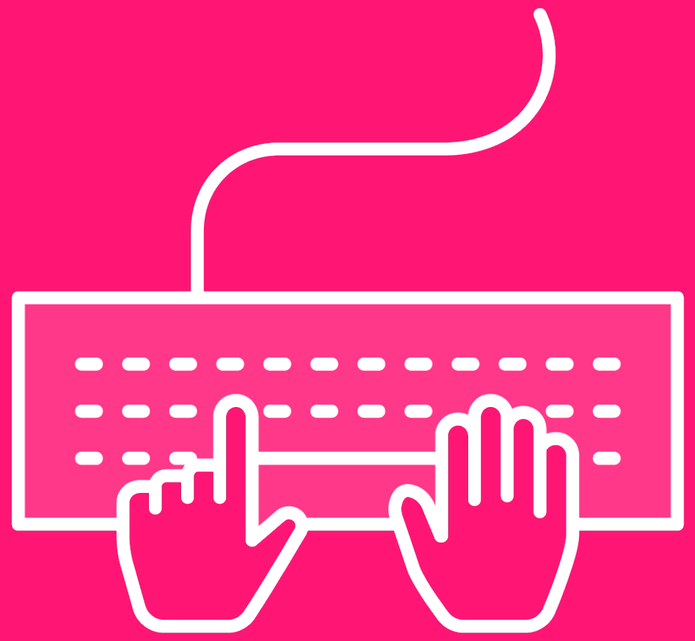
Higher conversion

Higher revenue

Live Video: Building faster React apps is good for business

What to Expect

Feel free to jump around. Each module starts simple and ends with more advanced concepts.



Performance in Action

Access the full demo app at:
bit.ly/PSReactPerfDemo

Where to find the demo files

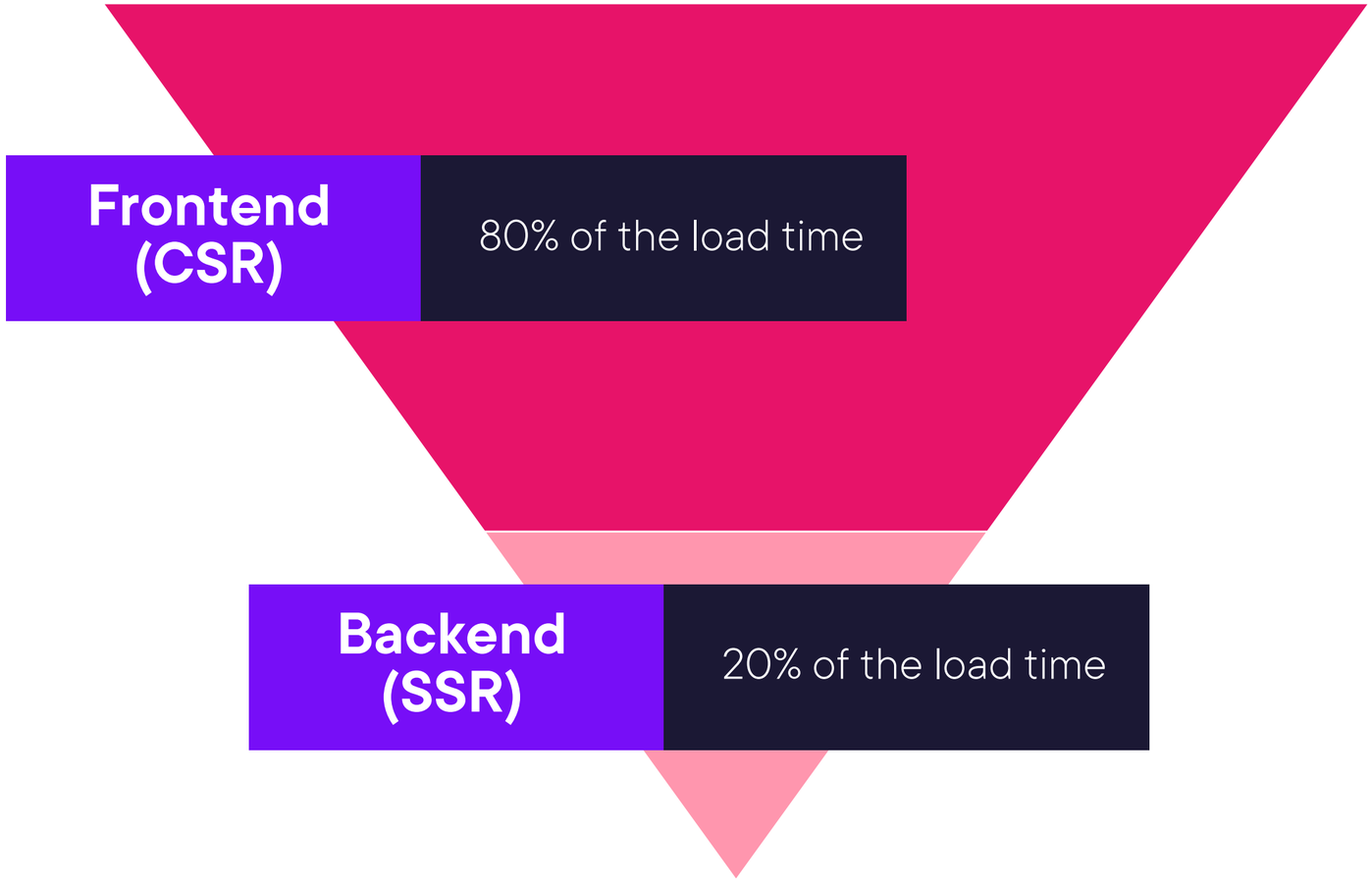
Inside the Pluralsight app, under "Exercise Files"

On GitHub, at bit.ly/PSReactPerfRepo (case-sensitive)

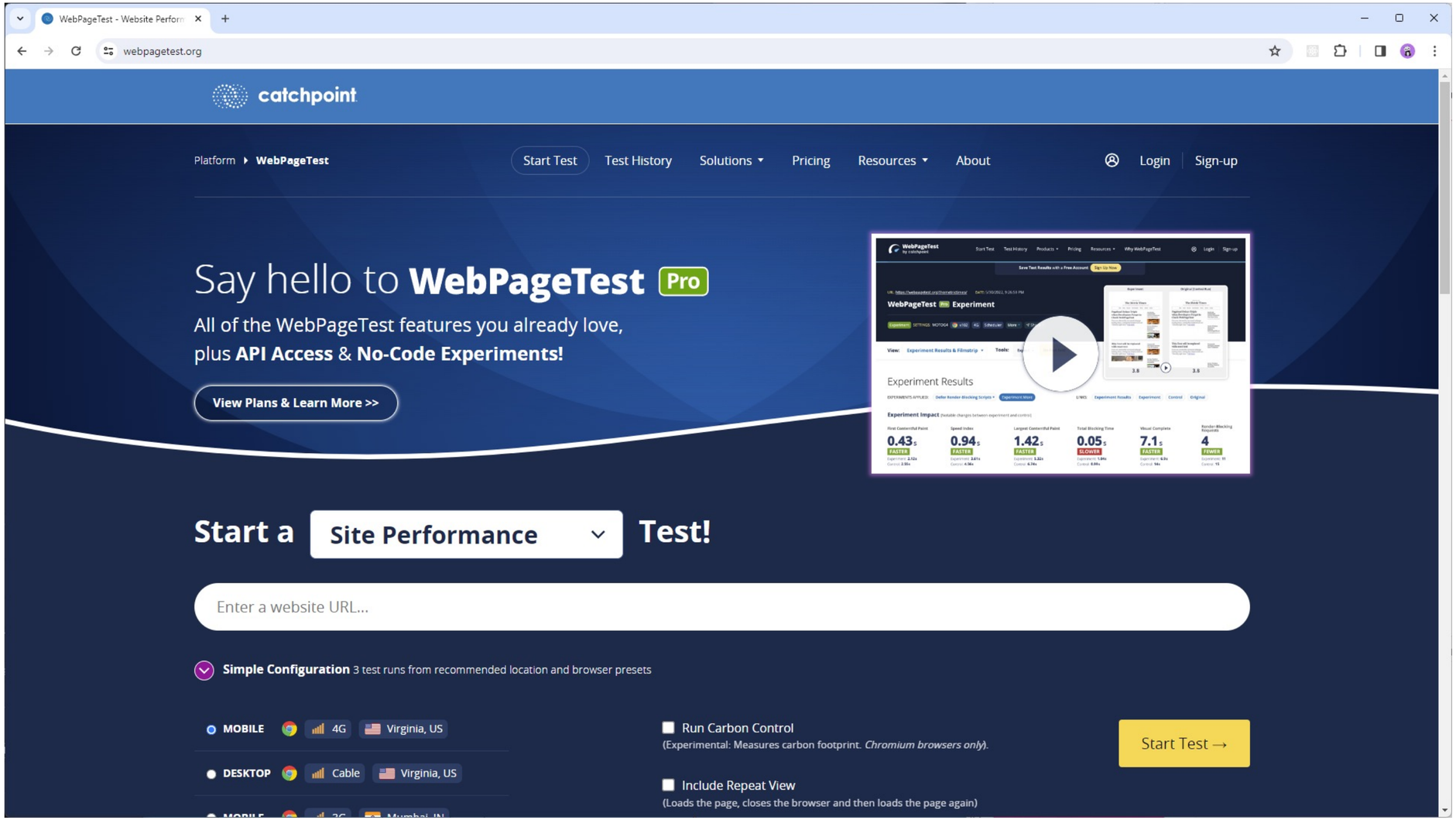
Live Video Outro

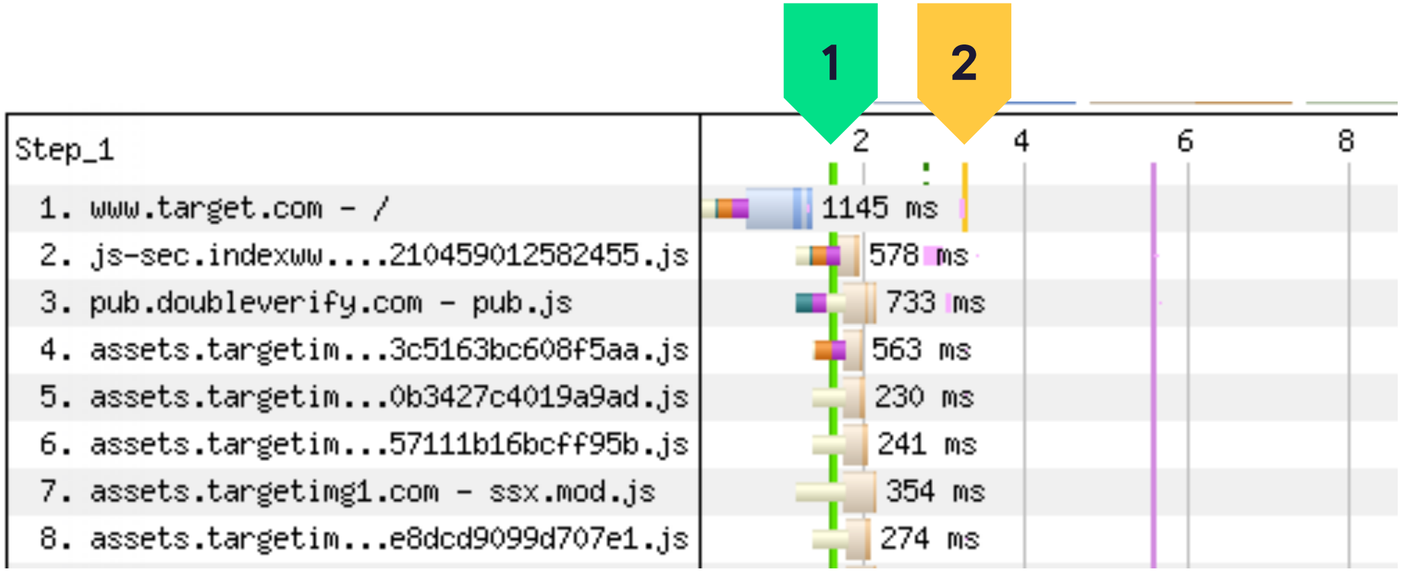
Avoiding the Pitfalls of Performance

80/20 Performance Rule of Thumb?



WebPageTest B-Roll Footage





Target.com

1 **Time-to-First Byte (TTFB)** 1.1s

2 **Largest Contentful Paint (LCP)** 2.7s

Frontend (CSR)

60/40

Backend (SSR)

Target.com

Time-to-First Byte (TTFB)

1.1s

Largest Contentful Paint (LCP)

2.7s

Frontend (CSR)

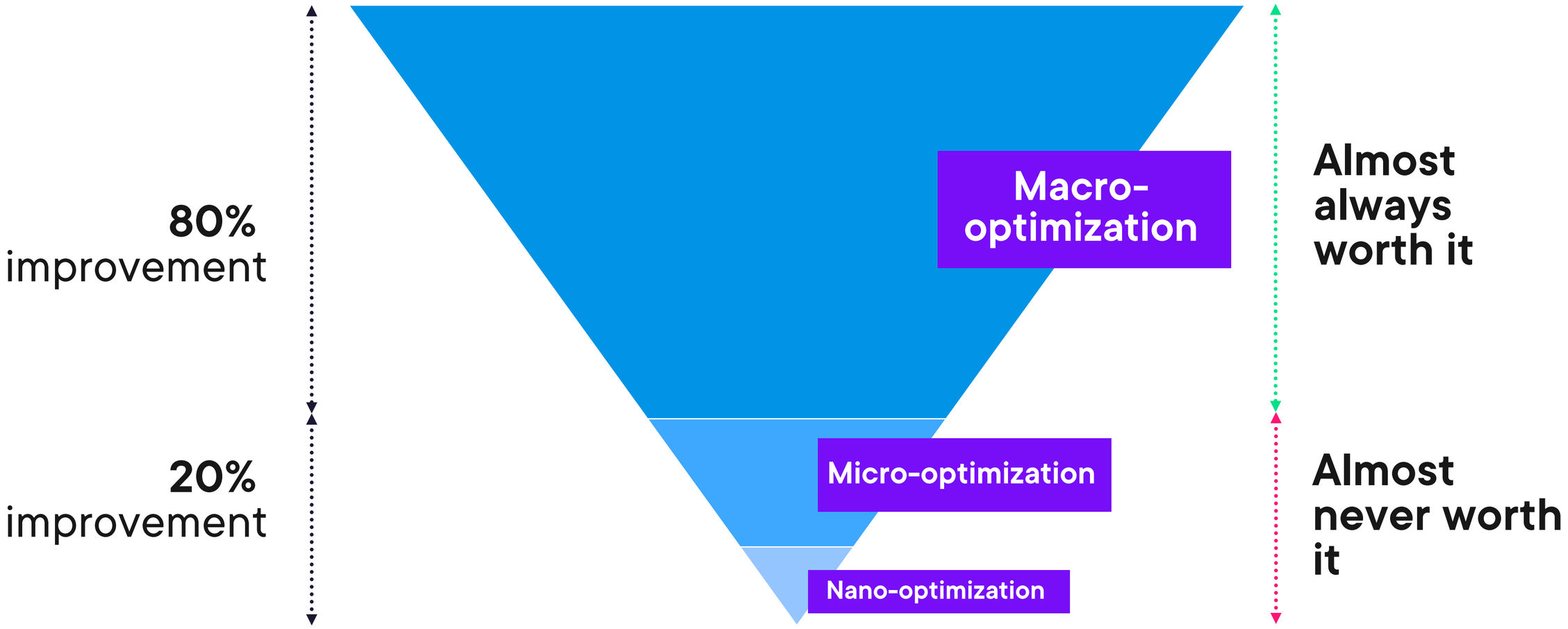
70/30

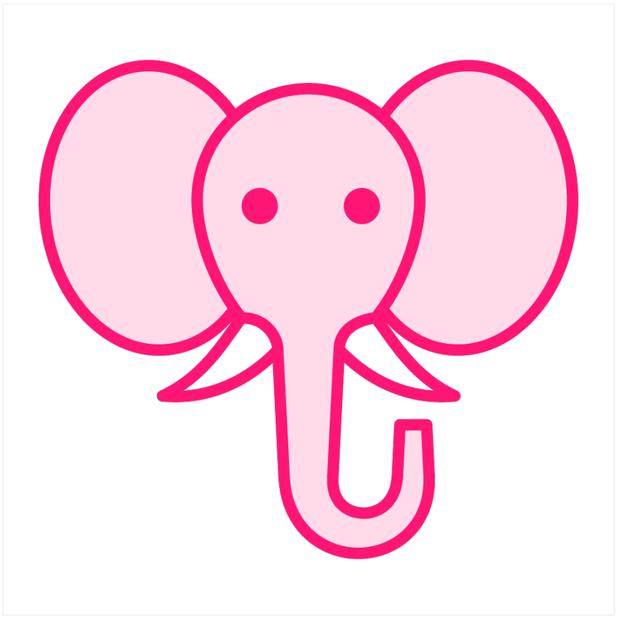
Backend (SSR)

Airbnb.com

Don’t just rely on “rules of thumb.” Measure your application’s performance to decide where to focus.

The Performance Optimization Hierarchy



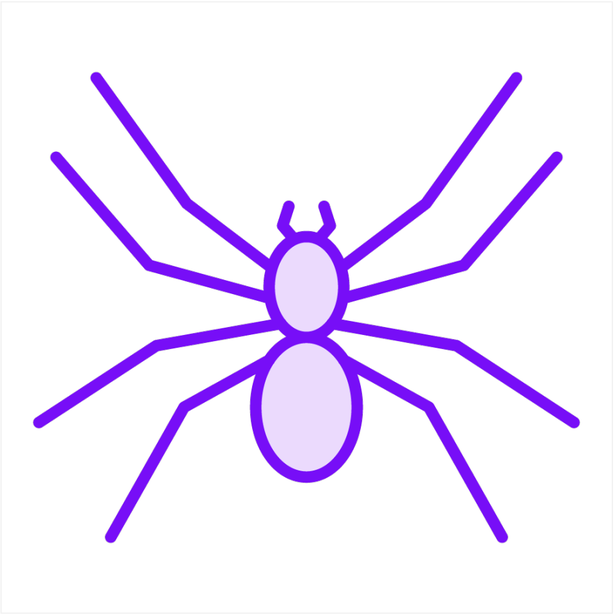


Macro optimizations

Contributes a significant performance improvement (on the order of seconds)

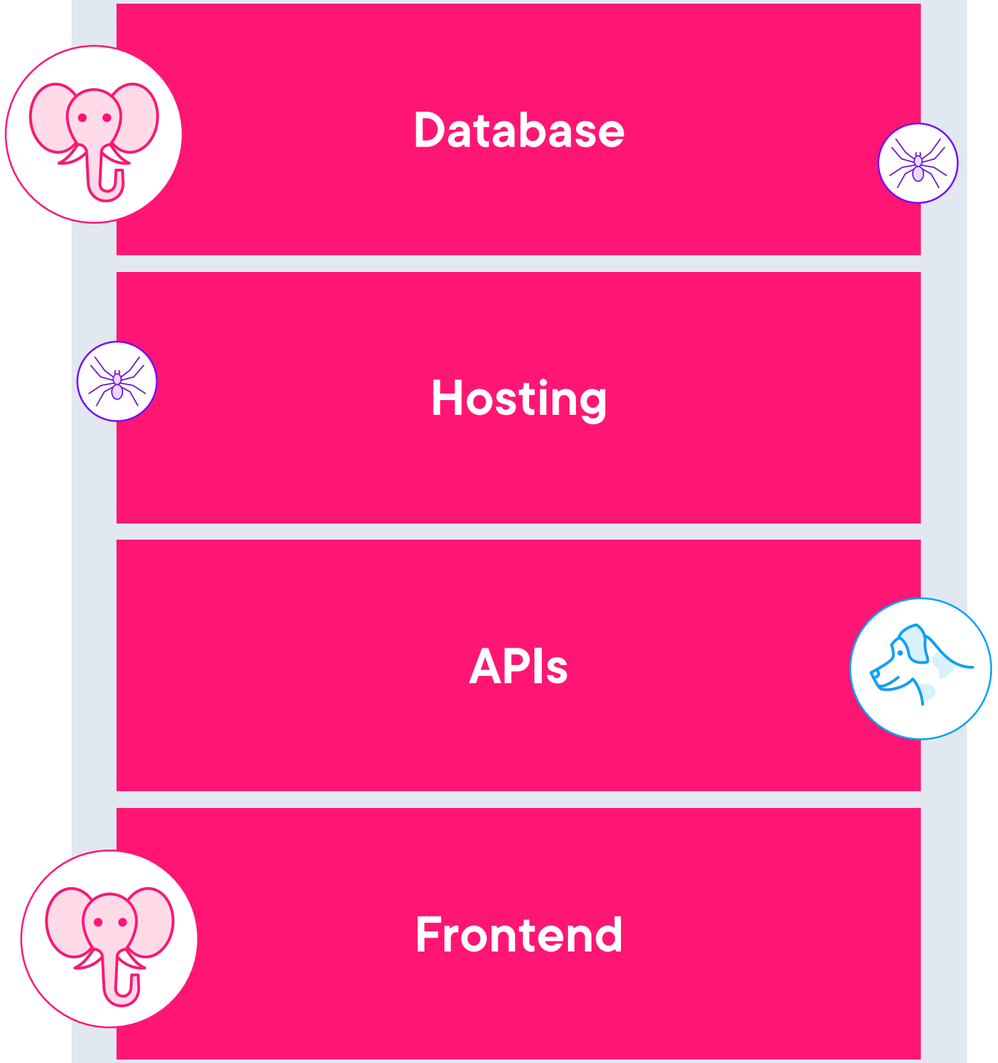
Micro optimizations

Contributes a modest improvement (on the order of 100s of milliseconds)



Nano optimizations

Contributes a minor improvement (on the order of 10s of milliseconds)


Database
Hosting
APIs
Frontend

Example: The Tale of The Slow Spread Operator

```
export const selectProductAttributes = createSelector(
  selectAttributes, selectProducts, (attr, products) => {
  // complex merging logic
});
```

Example: The Tale of The Slow Spread Operator

```
export const selectProductAttributes = createSelector(
  selectAttributes, selectProducts, (attr, products) => {

  const mergedProduct = {
    ...prevAttributes[productId],
    ...newAttributes[productId],
    ...commonAttributes
  };

});
```

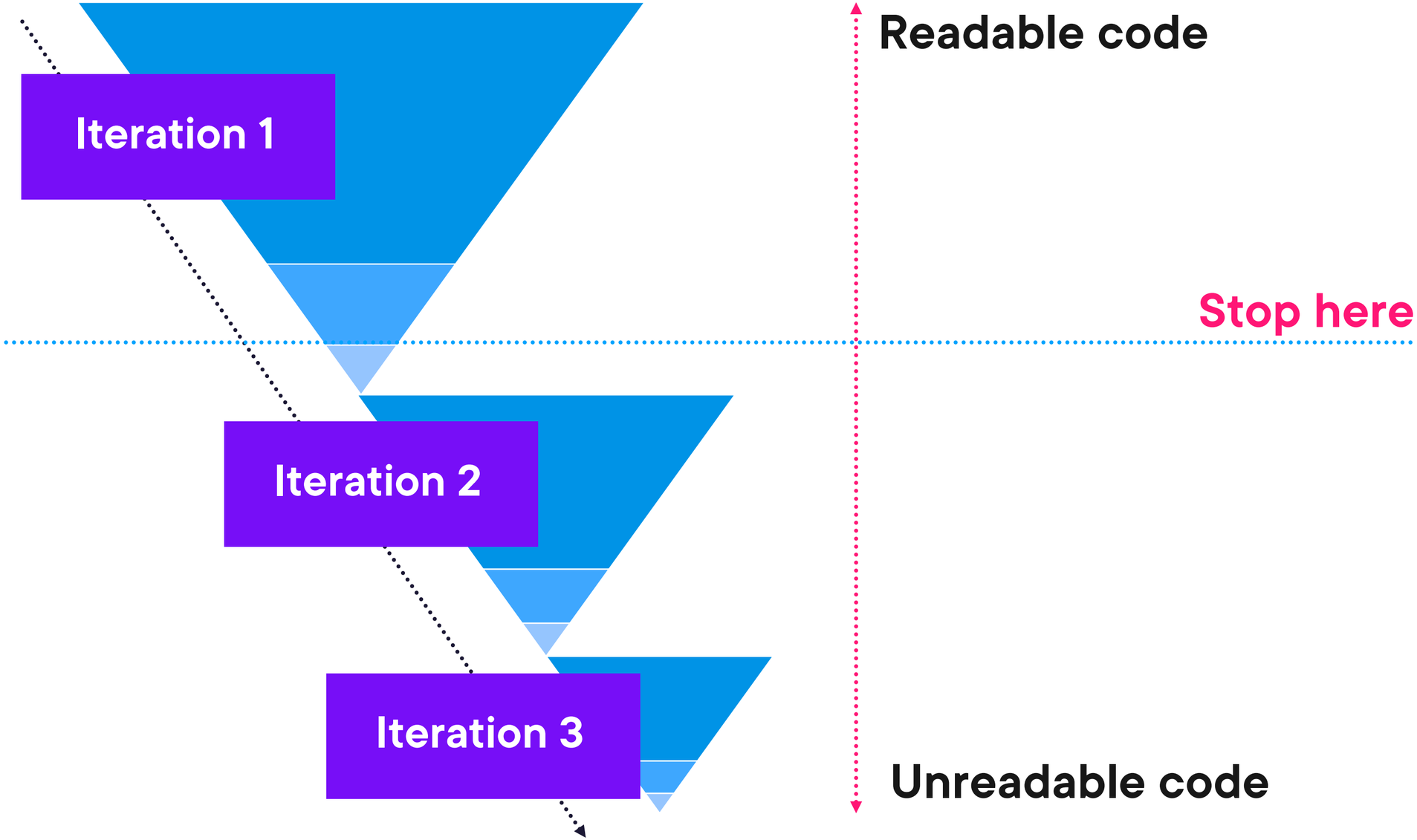
Example: The Tale of The Slow Spread Operator

```
export const selectProductAttributes = createSelector(
  selectAttributes, selectProducts, (attr, products) => {

  const mergedProduct = Object.apply({},
    prevAttributes[productId],
    newAttributes[productId],
    commonAttributes
  );

});
```

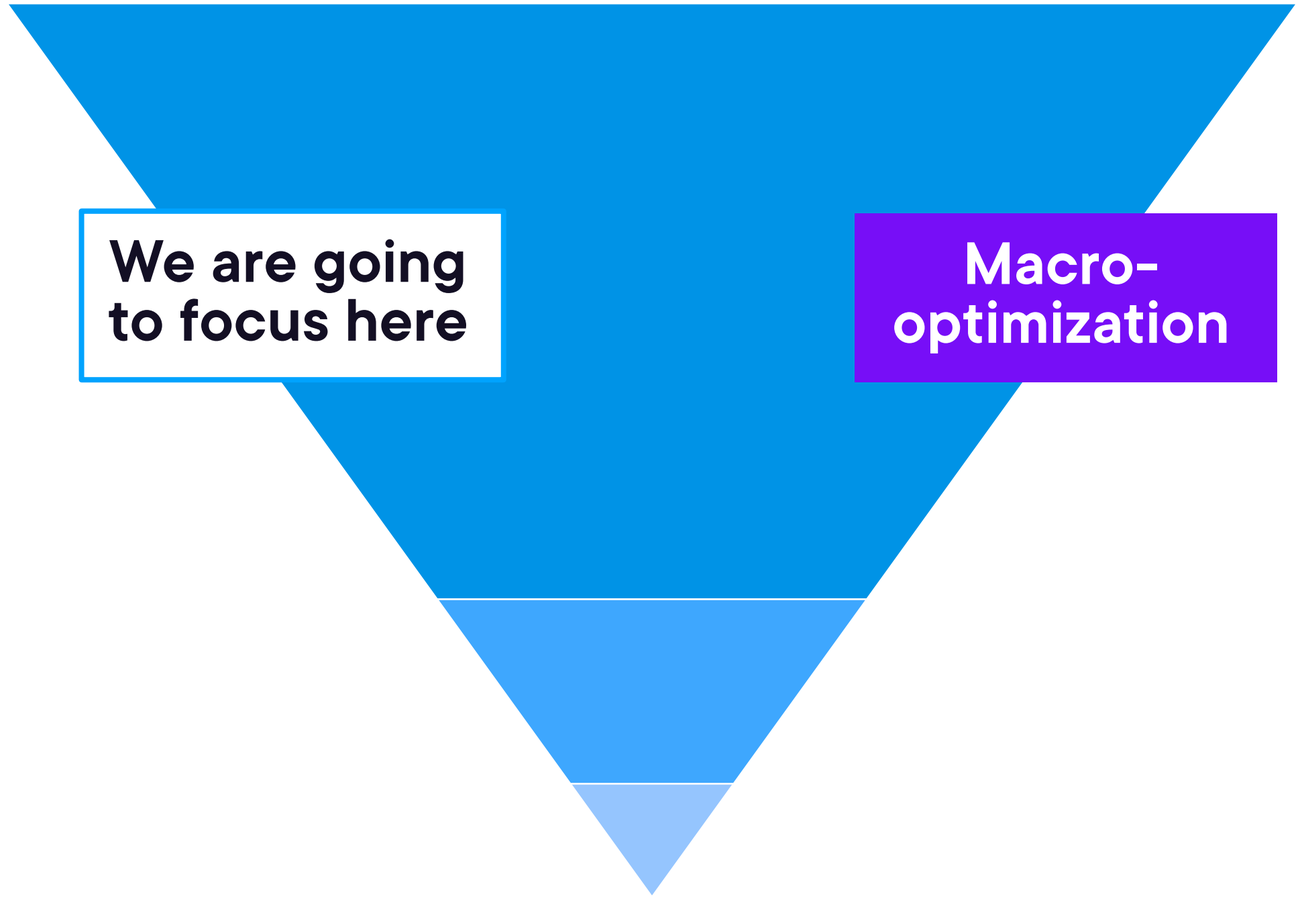
Example: The Tale of The Slow Spread Operator

```
export const selectProductAttributes = createSelector(
  selectAttributes, selectProducts, (attr, products) => {

  // WARNING: Do not use spread (...) operator due to performance issue
  // See: GH-1234 for more details
  const mergedProduct = Object.apply({},
    prevAttributes[productId],
    newAttributes[productId],
    commonAttributes
  );

});
```

Law of Diminishing (Performance) Returns



The Performance Optimization Hierarchy



All potential performance improvements need to take their larger context into account

The Usual Suspects: Get to Know Common React Bottlenecks

Live Video

Changes Triggering Unnecessary Re-renders

The app is slow or janky even when seemingly trivial changes occur.

MyComponent.jsx

```
// Bad Practice
function MyComponent({ list }) {
  return list.map((item, index) =>
    <ChildComponent key={index} item={item} />);
}
```

State or prop changes causing a cascade of unnecessary re-renders in child components.

DOM Thrashing and Manipulation

The UI freezes or lags noticeably, especially during animations or transitions.

useMyAnimation.jsx

```
// Bad Practice
const element =
document.getElementById("myElement");
element.style.height = "100px";
element.style.width = "100px";
```

Directly manipulating the DOM in quick succession, forcing multiple reflows/repaints.

Rendering Large Tables or Lists

Scrolling through a table or list is slow and choppy.

LargeTable.jsx

```
// Bad Practice
function LargeTable({ rows }) {
  return (
    <table>
      {rows.map((row, index) => (
        <tr key={index}>{/* ... */}</tr>
      ))}
    </table>
  );
}
```

Rendering a table with hundreds or thousands of rows without using virtualization.

Inefficient Data Fetching

Data-dependent UI components take too long to render.

UserProfile.jsx

```
// Bad Practice
function UserProfile({ userId }) {
  const [user, setUser] = useState(null);

  // Fetches data every time userId changes,
  // no caching
  useEffect(() => { fetch(`/api/users/${userId}`)
    .then(response => response.json())
    .then(data => setUser(data));
  }, [userId]);


}
```

Fetching data within a component's render method without caching or error handling.

Slow App Loading Time

Users see a blank screen or a loading spinner for a longer than acceptable time.

App.jsx

```
// Bad Practice: No Code Splitting
import ComponentA from './ComponentA';
import ComponentB from './ComponentB';

function App() {
  return (
    <div>
      <ComponentA />
      <ComponentB />
    </div>
  );
}
```

Bundling all your JavaScript together, leading to a large file size and slow initial loading times.

Poorly Optimized State Management

Slow re-renders or unresponsive UI during state updates.

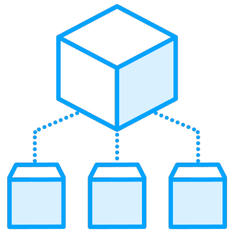
my-reducer.js

```
// Bad Practice
const myReducer = (state, action) => {
  // ...Recalculates everything even if not needed
  return newState;
};
```

Using Redux without proper memoization, causing unnecessary recalculations and re-renders.

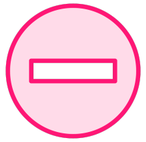
Common React Bottlenecks

Changes triggering unnecessary re-renders

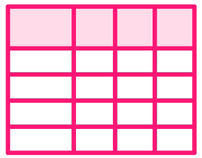


Inefficient data fetching

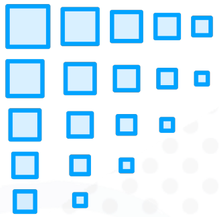
DOM thrashing and manipulation



Slow app loading time



Rendering large lists or tables



Poorly optimized state management

Summary

Today's users expect fast applications which is why speed equals revenue

There are diminishing returns to performance improvements which is why it's important to measure first, then prioritize

React bottlenecks usually stem from unnecessary updates or ineffective memoization