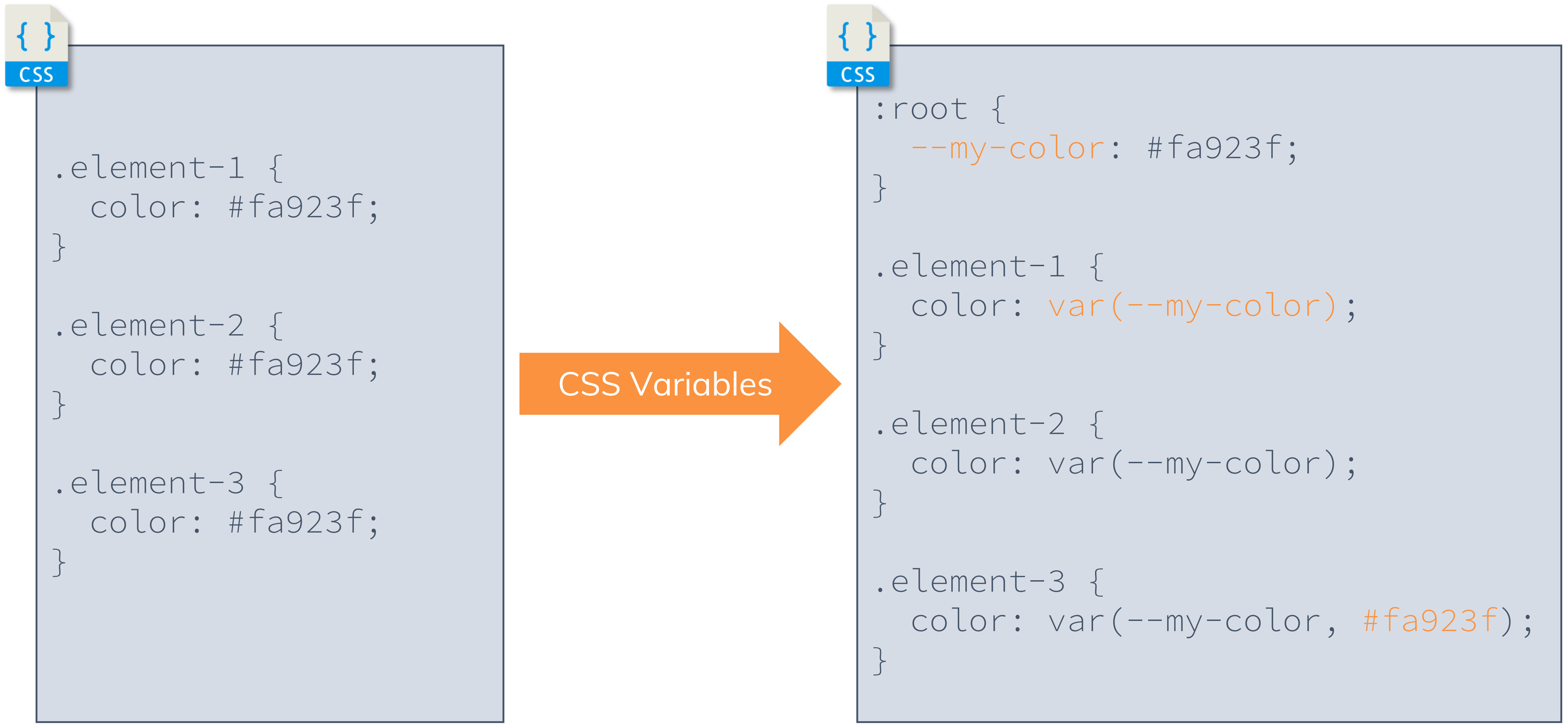
CSS Variables



```css
.element-1 {
  color: #fa923f;
}

.element-2 {
  color: #fa923f;
}

.element-3 {
  color: #fa923f;
}
```

```css
:root {
  --my-color: #fa923f;
}

.element-1 {
  color: var(--my-color);
}

.element-2 {
  color: var(--my-color);
}

.element-3 {
  color: var(--my-color, #fa923f);
}
```

Vendor Prefixes

Browsers implement new Features Differently and at different Speed

```css
.container {
  display: -webkit-box;
  display: -ms-flexbox;
  display: -webkit-flex;
  display: flex;
}
```

Support Queries

Some Features just aren‘t implemented (yet) in some Browsers

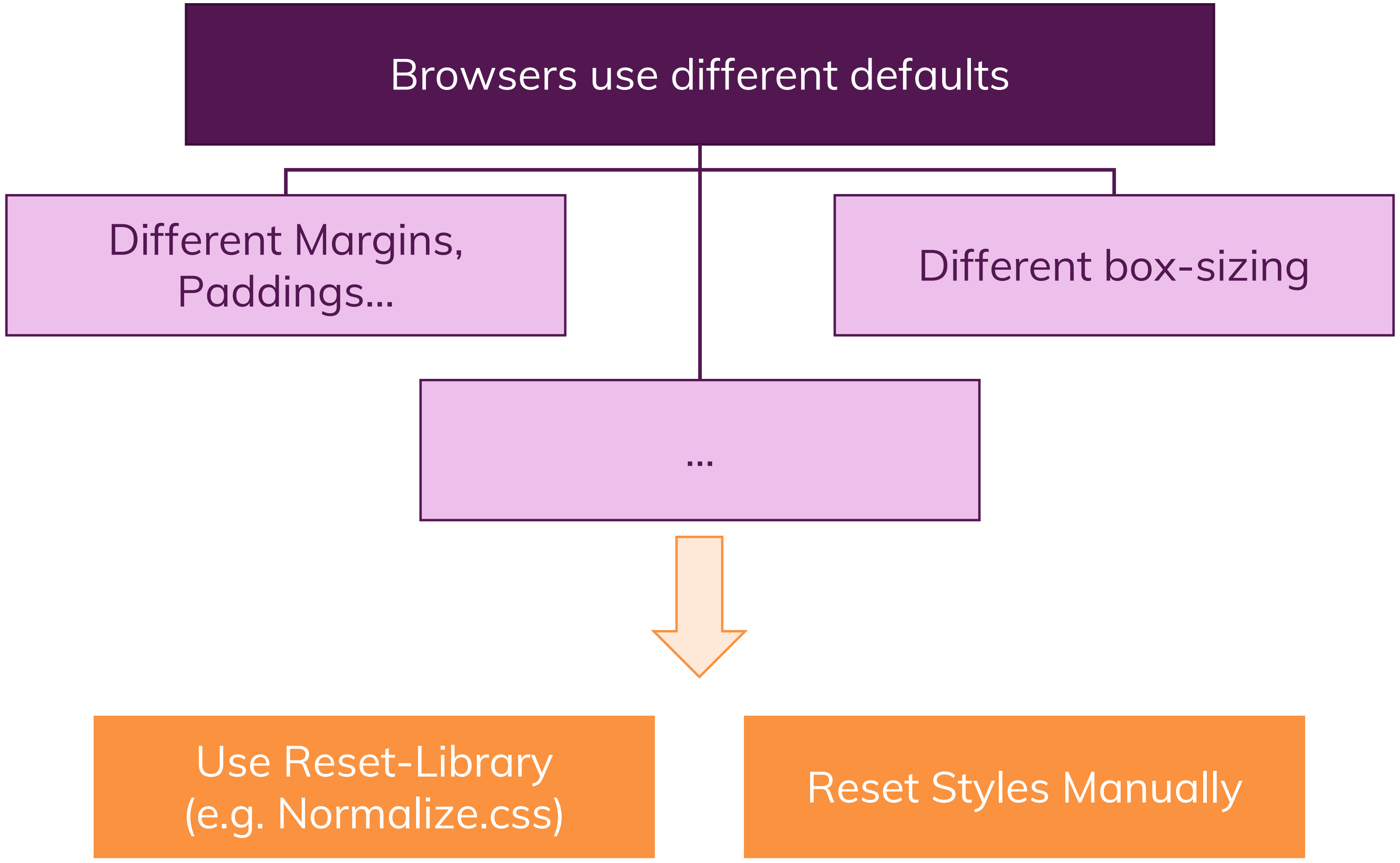
```css
@supports (display: grid) {
  .container {
    display: grid;
  }
}
```

Polyfills

A Polyfill is a JavaScript Package which enables certain CSS Features in Browsers which would not support it otherwise.

Remember: Polyfills come at a cost! The JavaScript has to be loaded and parsed!

Eliminate Cross-Browser Inconsistencies


Browsers use different defaults
Different Margins, Paddings…
Different box-sizing
…
Use Reset-Library (e.g. Normalize.css)
Reset Styles Manually

Choosing Class Names Correctly

Do	Don‘t
Use kebab-case	Use snakeCase
Because CSS is case-insensitive	Because CSS is case-insensitive
Name by feature	Name by style
For example `.page-title`	`.title-blue`

Block Element Modifier (BEM)

A uniform and consistent way of naming your CSS classes

	.	BLOCK	__	ELEMENT	--	MODIFIER
Example	.	menu-main	__	item	--	size-big
Example	.	button	__		--	success

"Vanilla CSS" vs CSS Frameworks

Vanilla CSS

CSS

Write all your styles and layouts on your own

Component Frameworks

Foundation
Start here, build everywhere.

Bootstrap 4

Choose from a rich suite of pre-styled components & utility features/ classes

Utility Frameworks

Tailwind CSS

Build your own styles and layouts with the help of utility features and classes

“Vanilla CSS” vs CSS Frameworks

Vanilla CSS	Component Frameworks	Utility Frameworks
Full Control	Rapid Development	Faster Development
No unnecessary Code	Follow Best Practices	Follow Best Practices
Name Classes as you like	No Need to be an Expert	No Expert Knowledge Needed
Build everything from Scratch	No or Little Control	Little Control
Danger of “bad code”	Unnecessary Overhead Code	Unnecessary Overhead Code
	“All Websites Look the Same”	

Summary

CSS Variables

- `--your-name: 1rem;`
- Define values once, use them multiple times
- Only supported in modern browsers

Naming CSS Classes

- Use kebab-case (e.g. `page-title`) and name classes by feature not by style (e.g. `title-blue`)
- Avoid class name collisions, for example by using BEM class names

Cross-Browser Support

- Browser implement new features differently and with different speed
- Use vendor-prefixes to use cutting-edge features AND support older browsers (partly)
- `@supports` allows you to check for feature-support before using a property
- Polyfills can enable some CSS features which wouldn't work otherwise
- Consider normalizing CSS defaults across browsers

Vanilla CSS vs Frameworks

- Writing all styles from scratch gives you full control but comes with more work and responsibility
- Component frameworks (e.g. Bootstrap 4) allow you to build web pages rapidly but with less control
- Utility frameworks can be a good compromise