

A JavaScript Framework for Presentations and Animations on Computer Science

Laszlo Korte

Bachelor Thesis
Technical Aspects of Multimodal Systems
Department of Informatics
University of Hamburg

Outline

1. Introduction
2. Web Platform
 - The Browser
 - NodeJS
 - Workflow
 - Graphical User Interface
3. Components
 - Demos
4. Architecture

Introduction

Motivation

Motivation

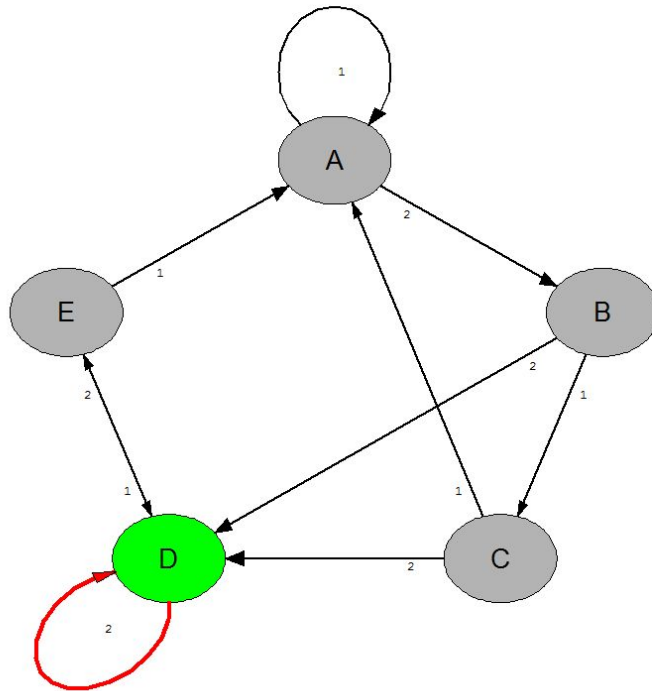
- Computer Science deals with abstract topics
- Hard to grasp without trying
- Short feedback cycle for building intuition

→ Interactive learning experience

Matlab Components

- **Boolean expression** parsing and evaluation
- **Binary number** encoding and decoding
- **Finate-state machine** simulation
- **Algorithm** visualisation
- ...

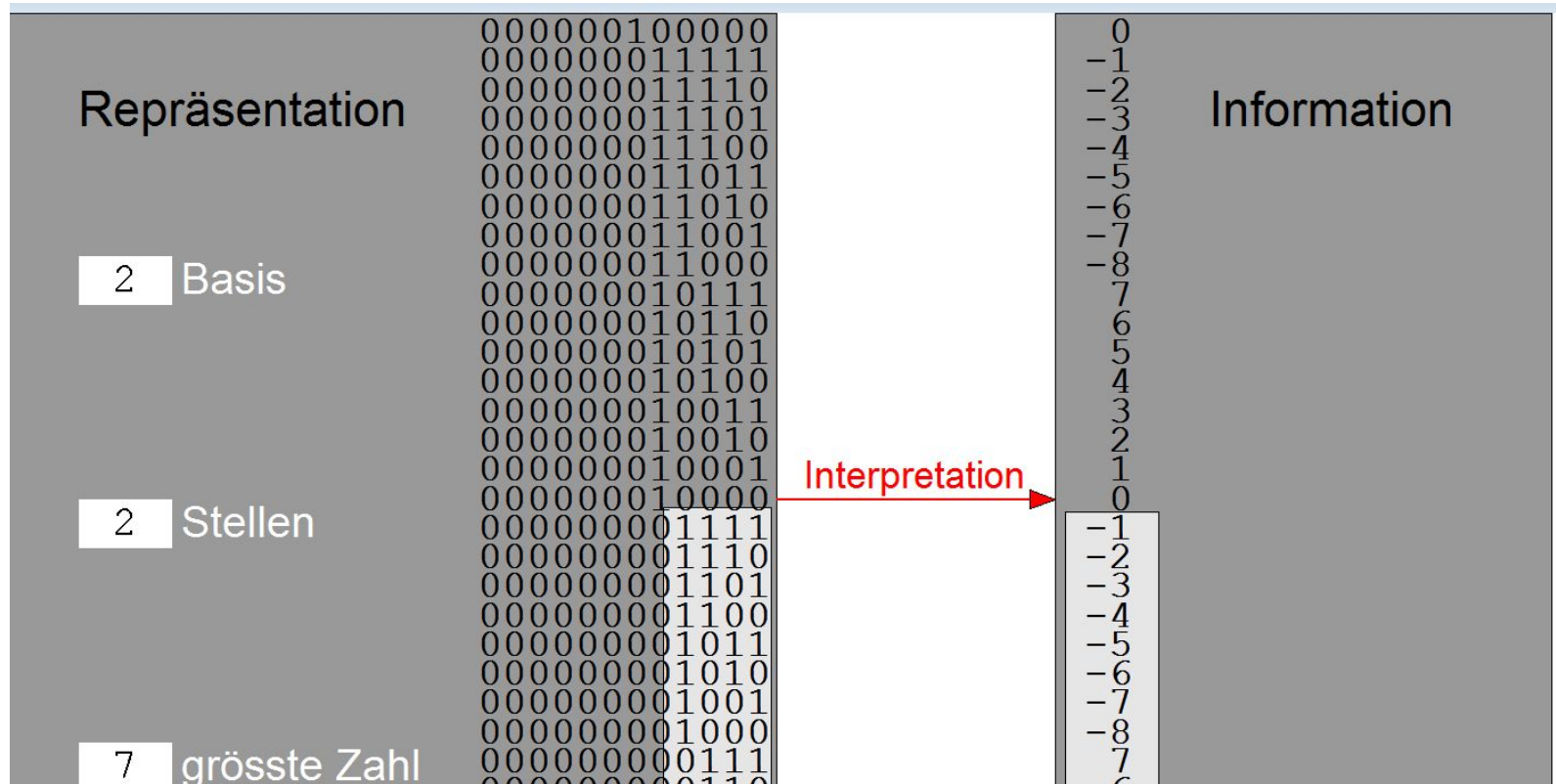
Matlab Components - FSM



Zustand	Index der Eingabe	
	1	2
A	A	B
B	C	D
C	A	D
D	E	D

finite-state machine simulator

Matlab Components - number encoding



number "circle"

Matlab Components - base conversion

Algorithmus

rechentechisch

darzustellende Zahl x

12345678

Basis q

16

a := x

n = 7

a = 0

while a > 0

(a > 0) = 0

y_n := a mod q

a := a div q

end



Takt

000000000BC614E
Resultat

visualisation of an algorithm for base conversion

Java Applets

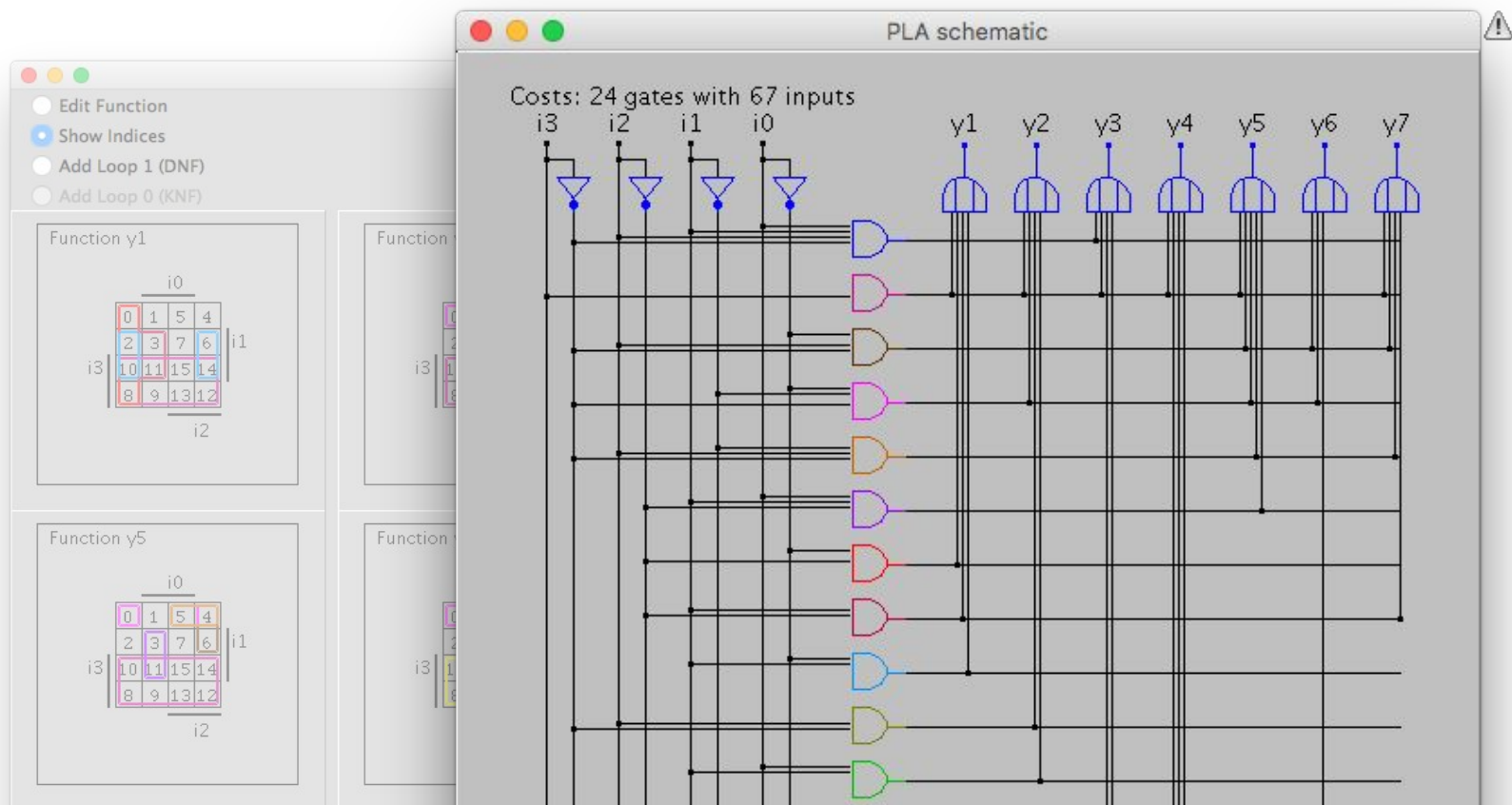
- Karnaugh map editor
- Advanced FSM editor

Java Applets - Karnaugh map



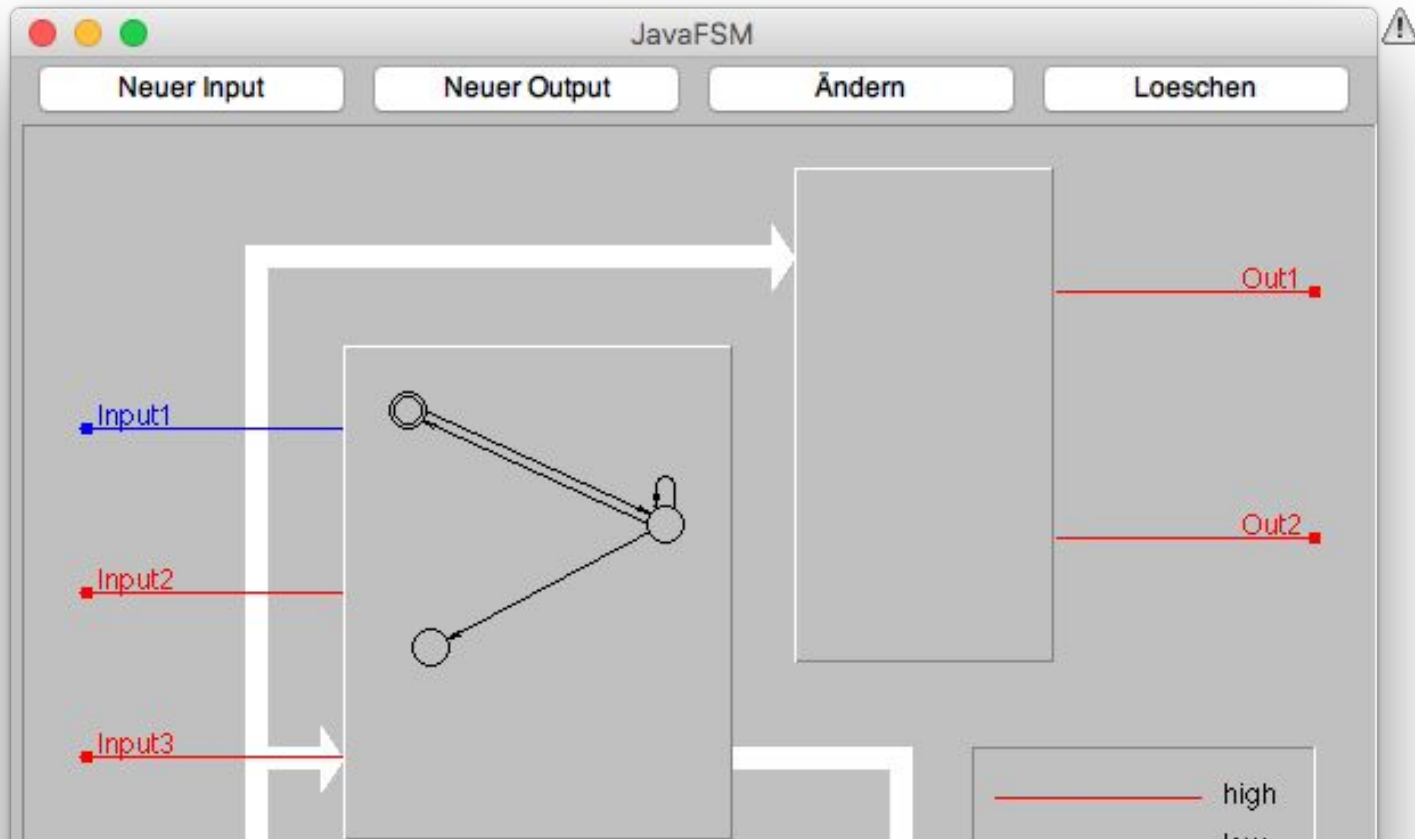
Karnaugh map editor

Java Applets - Karnaugh map



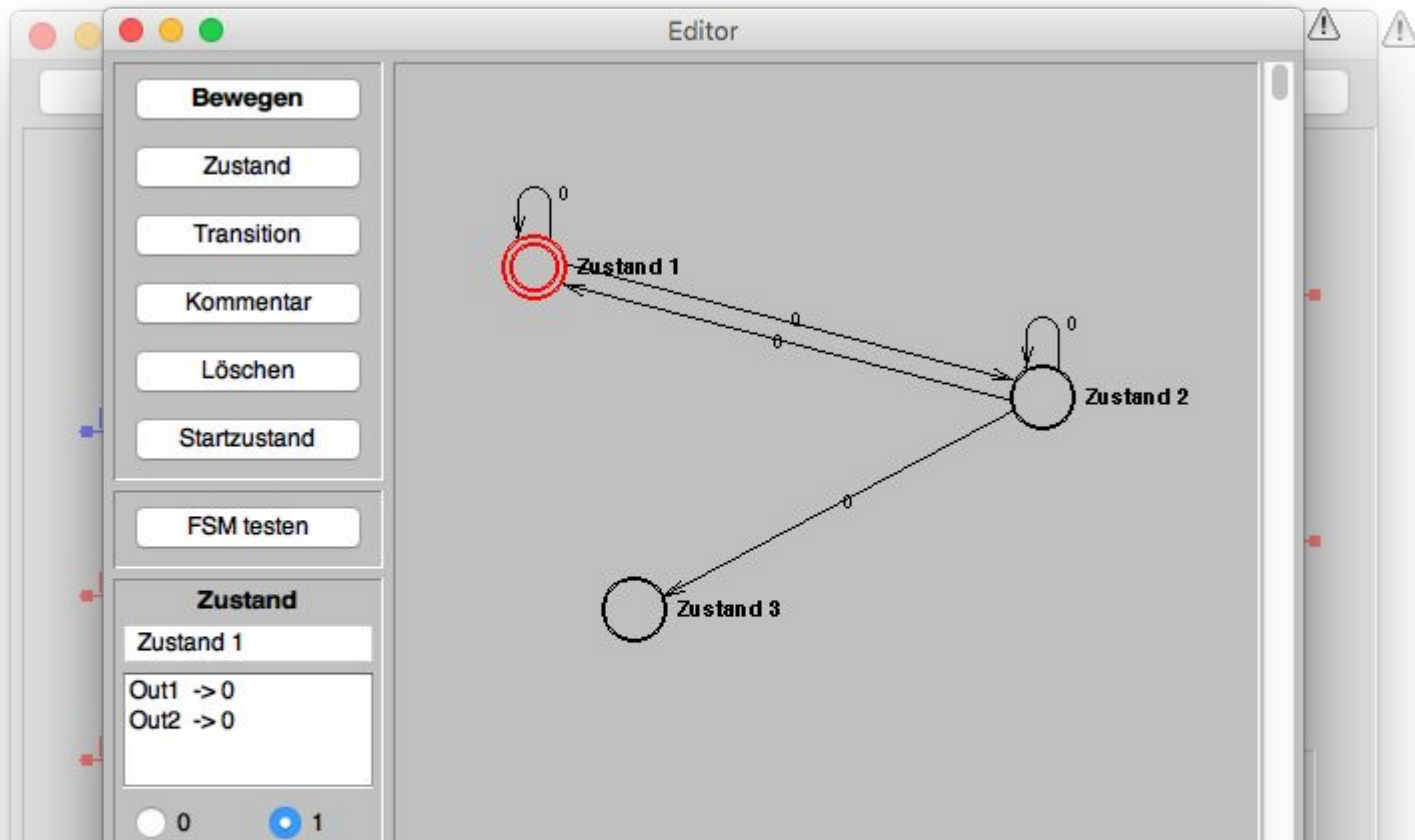
Karnaugh map editor

Java Applets - FSM



Finite-state machine editor

Java Applets - FSM



Finite-state machine editor

Issues

- Matlab
 - Not free, licensing restrictions
 - Not trouble-free with current Matlab version
 - Not on mobile devices

Issues

- Matlab

- Not free, licensing restrictions
- Not trouble-free with current Matlab version
- Not on mobile devices

- Java Applets

- Not on iOS (iPad, iPhone)
- Not usable on touch screens

Introduction

Objective

Objective

- A new collection of components
 - no licensing issues
 - not limited to a small set of devices
- The web as **free** and **open** platform
- **Touch screen** support
- Support a wide range of **screen sizes**

Objective - components

- Karnaugh map editor
- Finite-state machine editor
- Boolean expression editor
- Number circle
- ...

→ **Expandable later on**

Web Platform

The Browser

The Browser

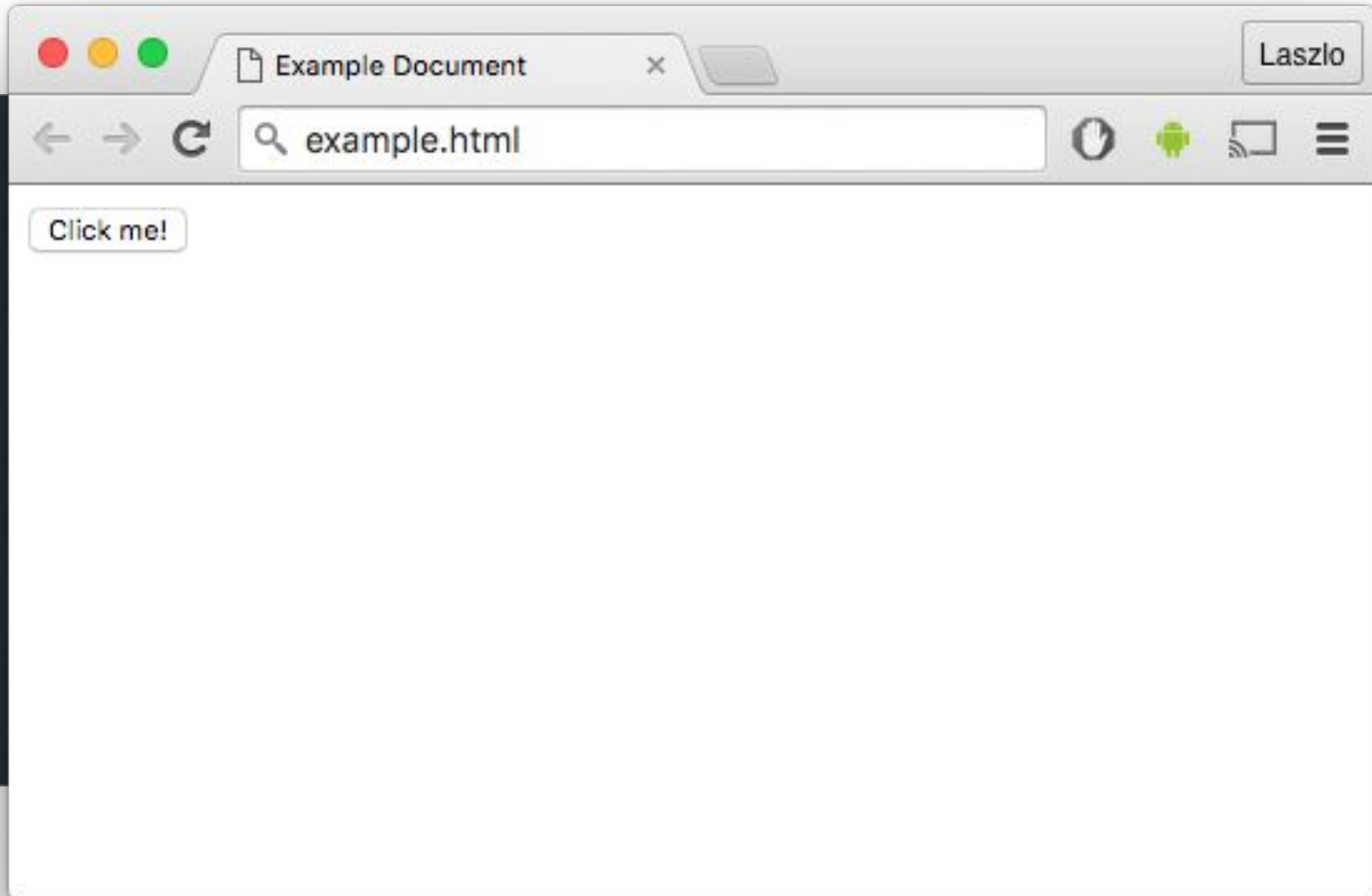
- Display HyperText Markup Language Documents (HTML)
- Documents can be styled via Stylesheets (CSS)
- Documents may contain code to execute (JavaScript)

The Browser - example HTML document

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Example Document</title>
5 </head>
6 <body>
7     <button>Click me!</button>
8 </body>
9 </html>
```

index.html

The Browser - example HTML document



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Example Document</title>
5   <style>
6     button {
7       width: 100%;
8       background-color: black;
9       color: #fff;
10      font-size: 24pt;
11    }
12  </style>
13 </head>
14 <body>
15   <button>Click me!</button>
16 </body>
17 </html>
```



+ CSS

```
1 <!DOCTYPE html>
```

```
2 <html>
```

```
3 <
```

```
4
```

```
5 <
```

```
6 <
```

```
7
```



```
10
```

```
11
```

```
12
```

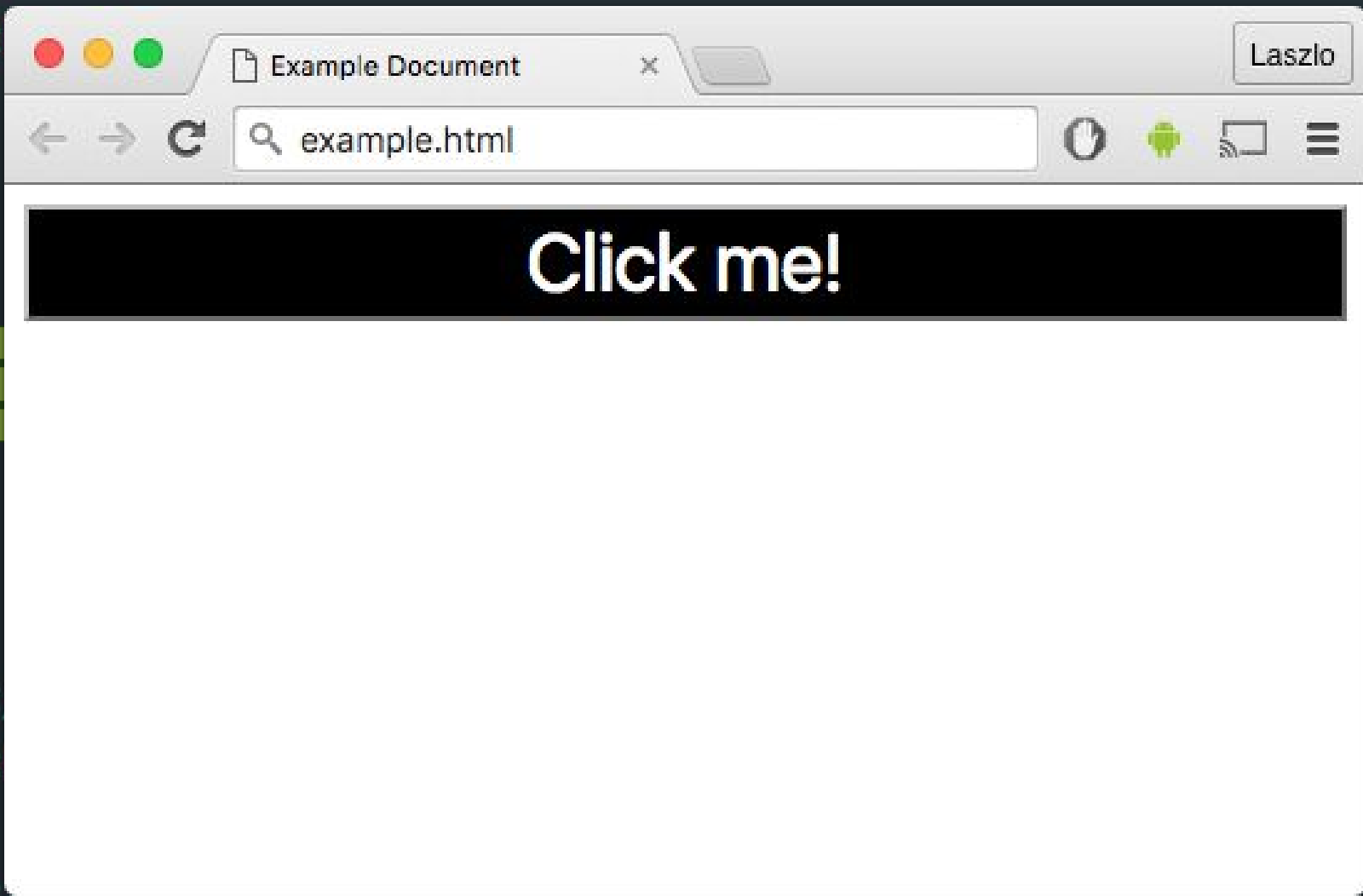
```
13 <
```

```
14 <
```


```
15
```

```
16 </body>
```

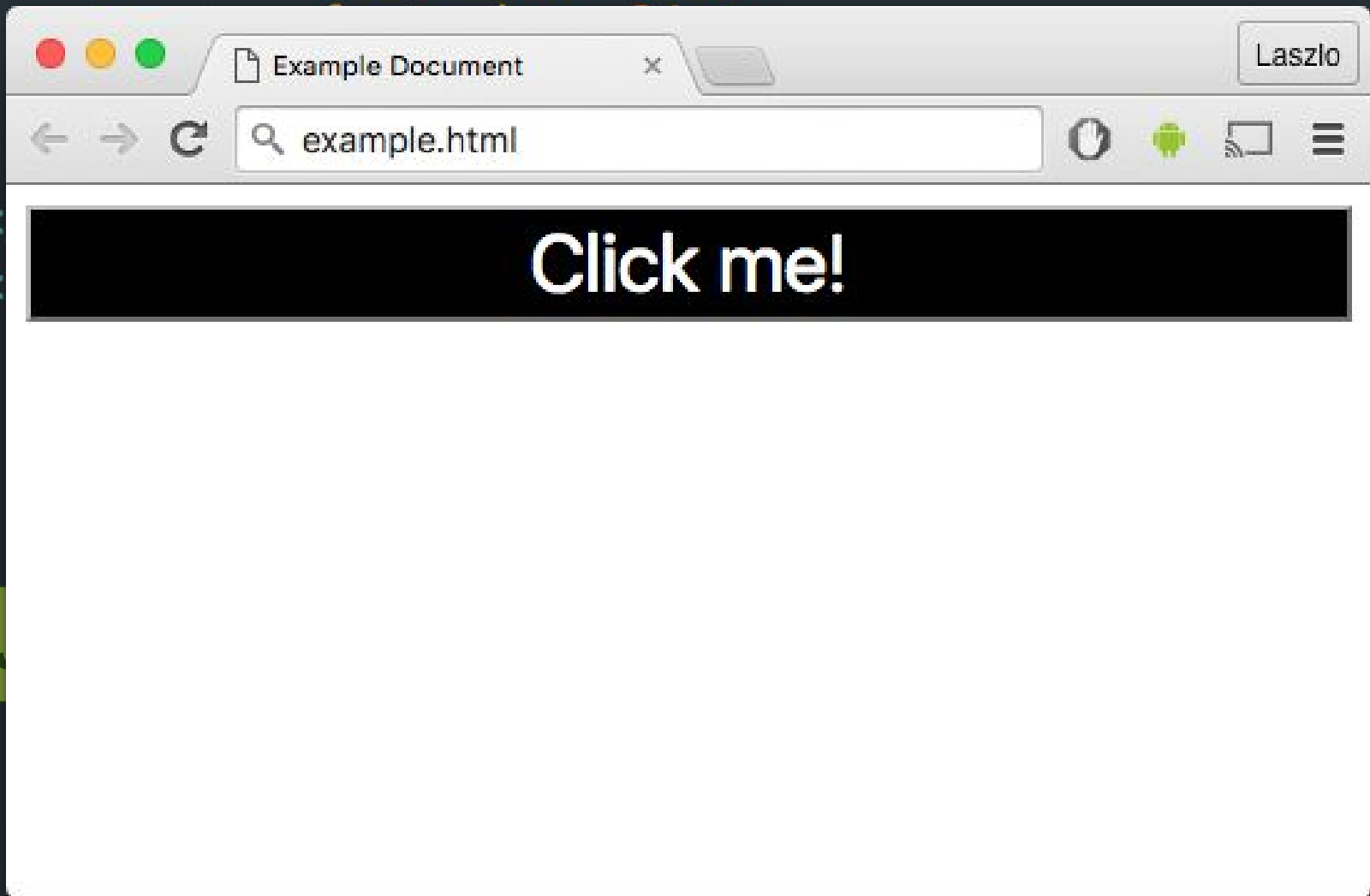
```
17 </html>
```




```
9         color: #fff;
10        font-size: 24pt;
11    }
12    </style>
13 </head>
14 <body>
15     <button>Click me!</button>
16     <script>
17         var count = 0;
18         document.querySelector('button')
19             .addEventListener('click', function (e) {
20             e.target.innerHTML =
21                 'Click me! (' + (++count) + ')';
22             })
23     </script>
24 </body>
25 </html>
```



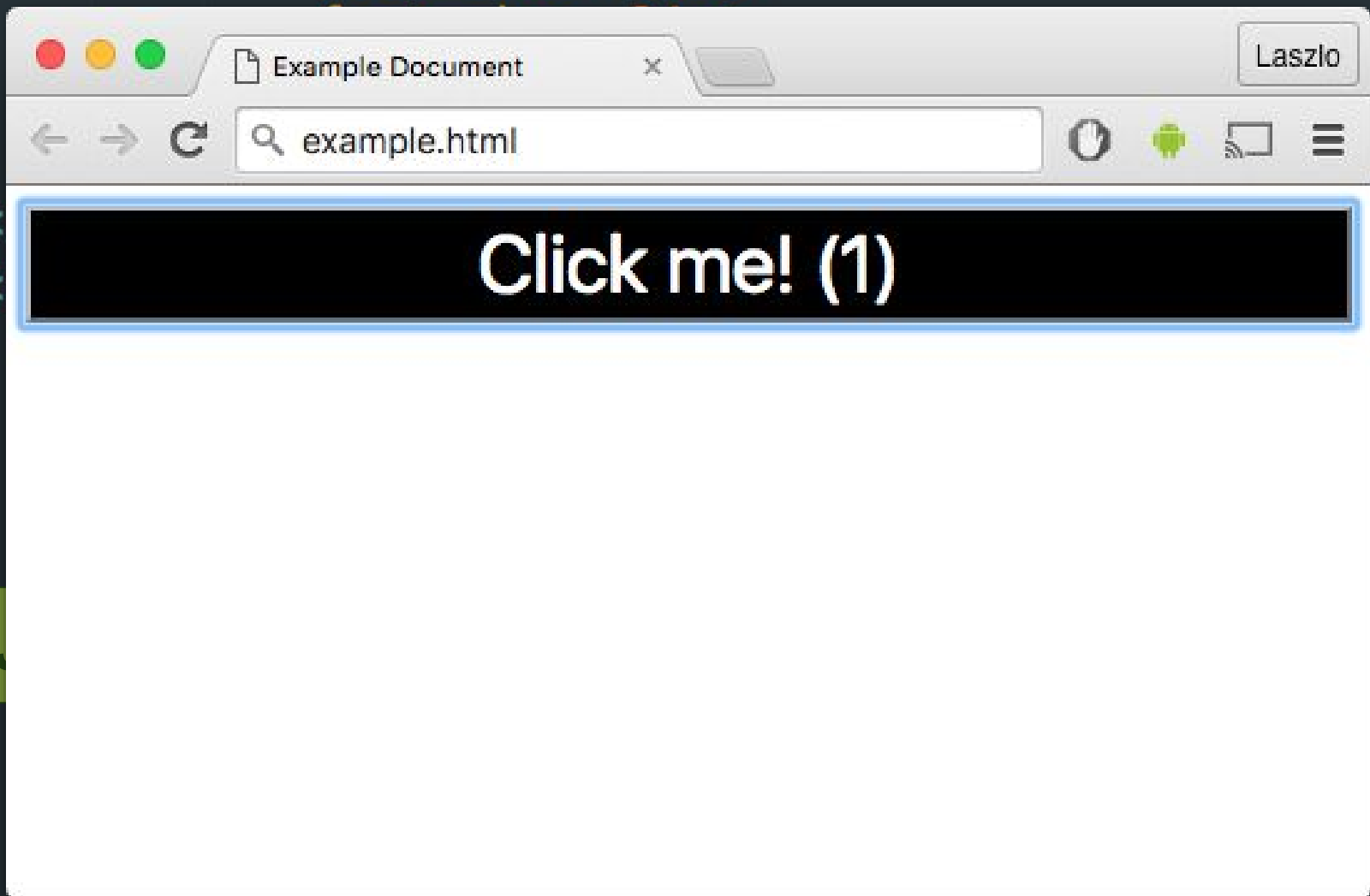
```
color: #fff;
```



```
</body>
```

```
</html>
```

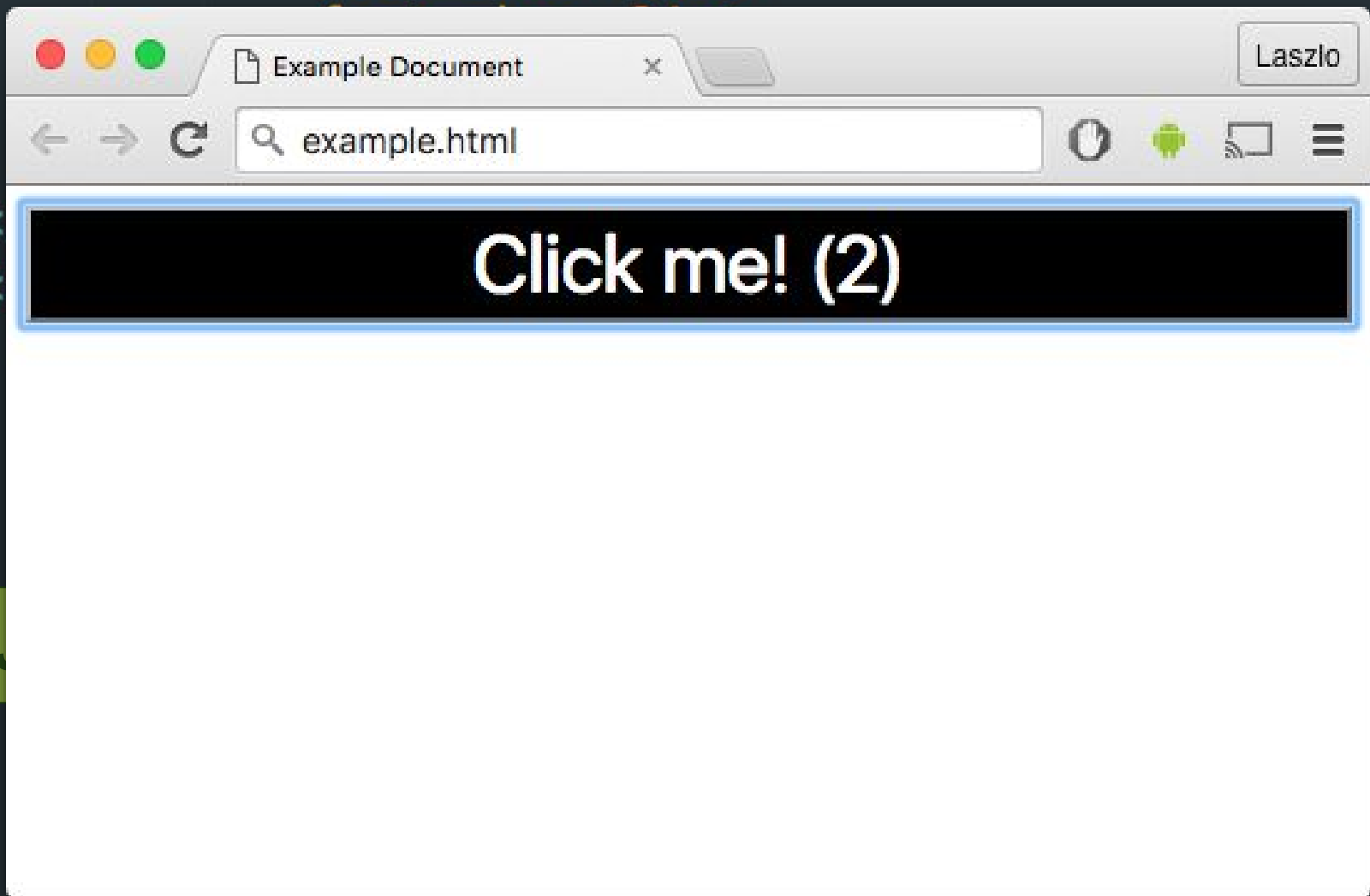
```
color: #fff;
```



```
</body>
```

```
</html>
```

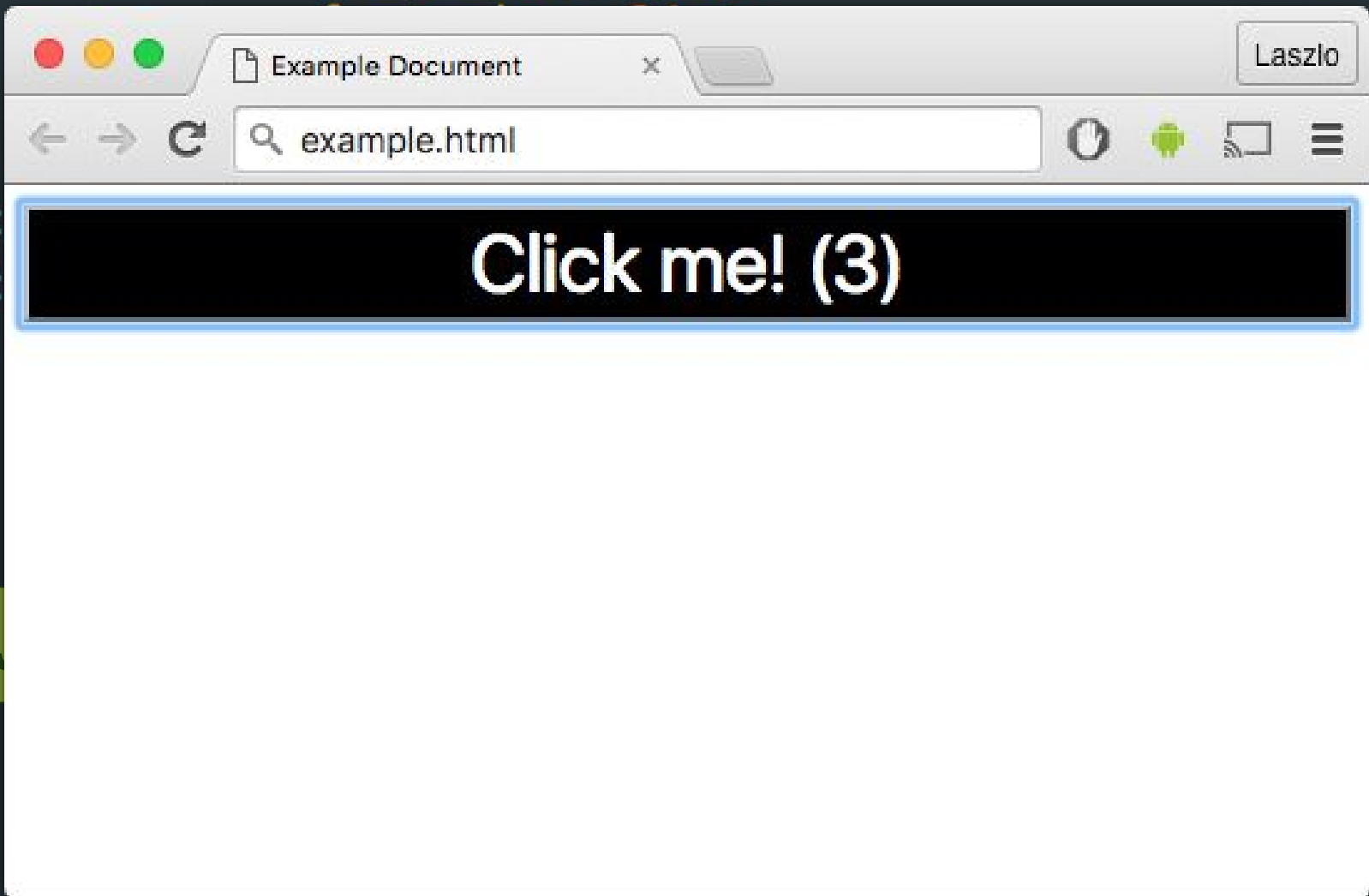
```
color: #fff;
```



```
</body>
```

```
</html>
```

```
color: #fff;
```



```
</body>
```

```
</html>
```

The Browser - example document

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Example Document</title>
5     <link rel="stylesheet" href="./layout.css">
6 </head>
7 <body>
8     <button>Click me!</button>
9     <script src="./main.js"></script>
10 </body>
11 </html>
```

index.html

The Browser

- HTML Document is the **entry point**
 - links to/embeds stylesheets
 - links to/embeds javascripts
 - Most of the application is defined in JavaScript (JS)
- start with an almost empty document,
build everything in JS

The Browser - Document Object Model

- DOM API allows querying and modifying the whole document

The Browser - JavaScript

- No module system
 - difficult to split application into modules
- No static type system
- Weird semantics
 - $5 + "5" === "55"$
 - $5 * "5" === 25$
- API differences across browsers

Web Platform
NodeJS

NodeJS



Source: <https://nodejs.org/en/about/resources/>

- JavaScript runtime environment
 - Based on Chrome's JavaScript engine V8
 - Run JavaScript outside the browser

→ Large ecosystem of open source libraries and tools

NodeJS - Package Manager






Source: <https://www.npmjs.com/>

- Command line tool for managing dependencies
 - Like maven for Java or easy_install for python
- Access to **>250.000** libraries

Web Platform
Workflow

Workflow - Tools

- BabelJS 
 - Access to future JS features (eg. module system)
- ESLint 
 - Check source code for common errors
- Webpack 
 - Module bundler
 - combine multiple JS files and libraries into one file

Workflow - Babel

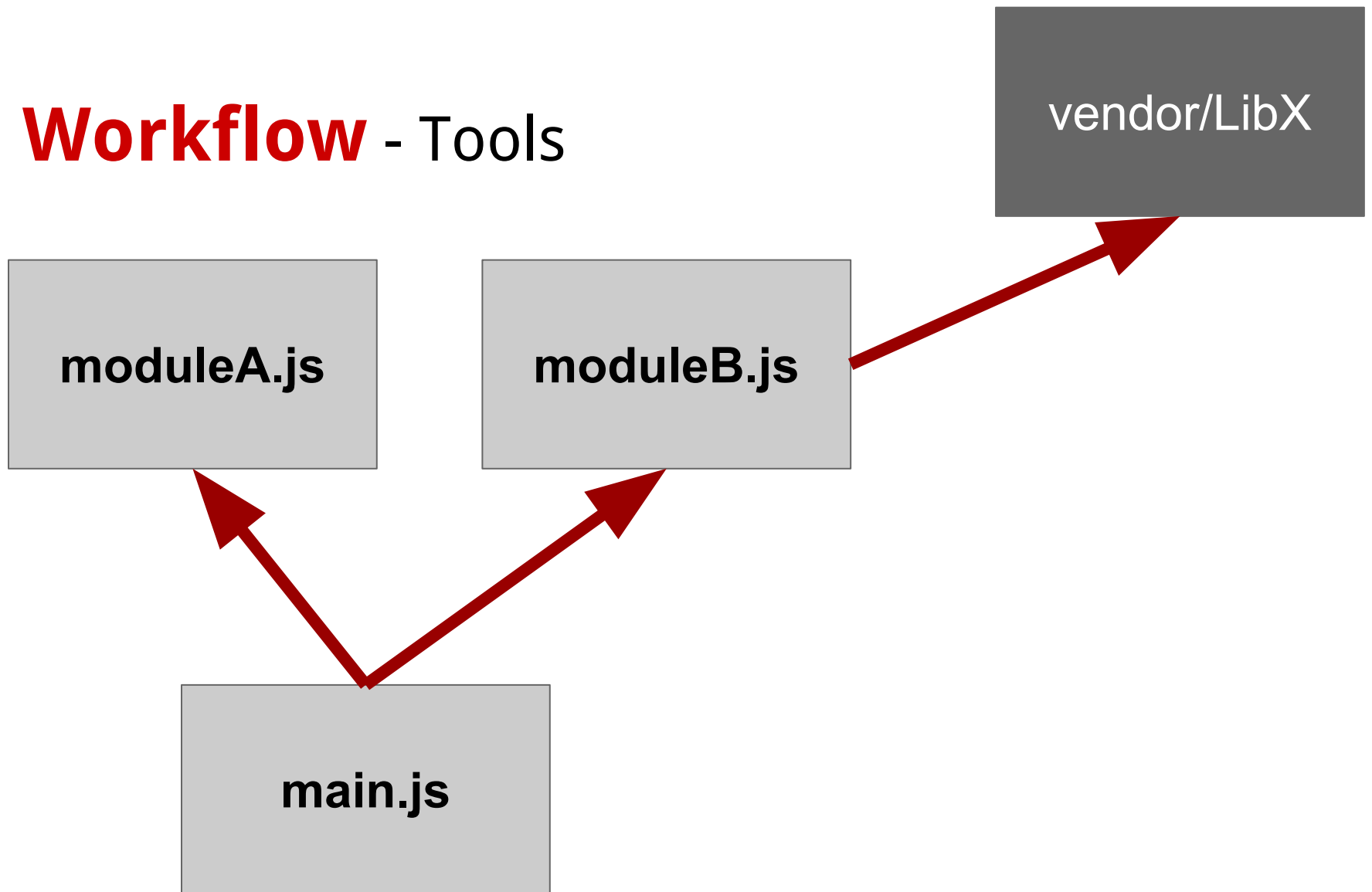


Source: <https://github.com/babel/babel>

```
1 import doStuff from './moduleA.js';  
2 import someFunction from './moduleB.js';  
3  
4 // ...
```

ES6 module import syntax

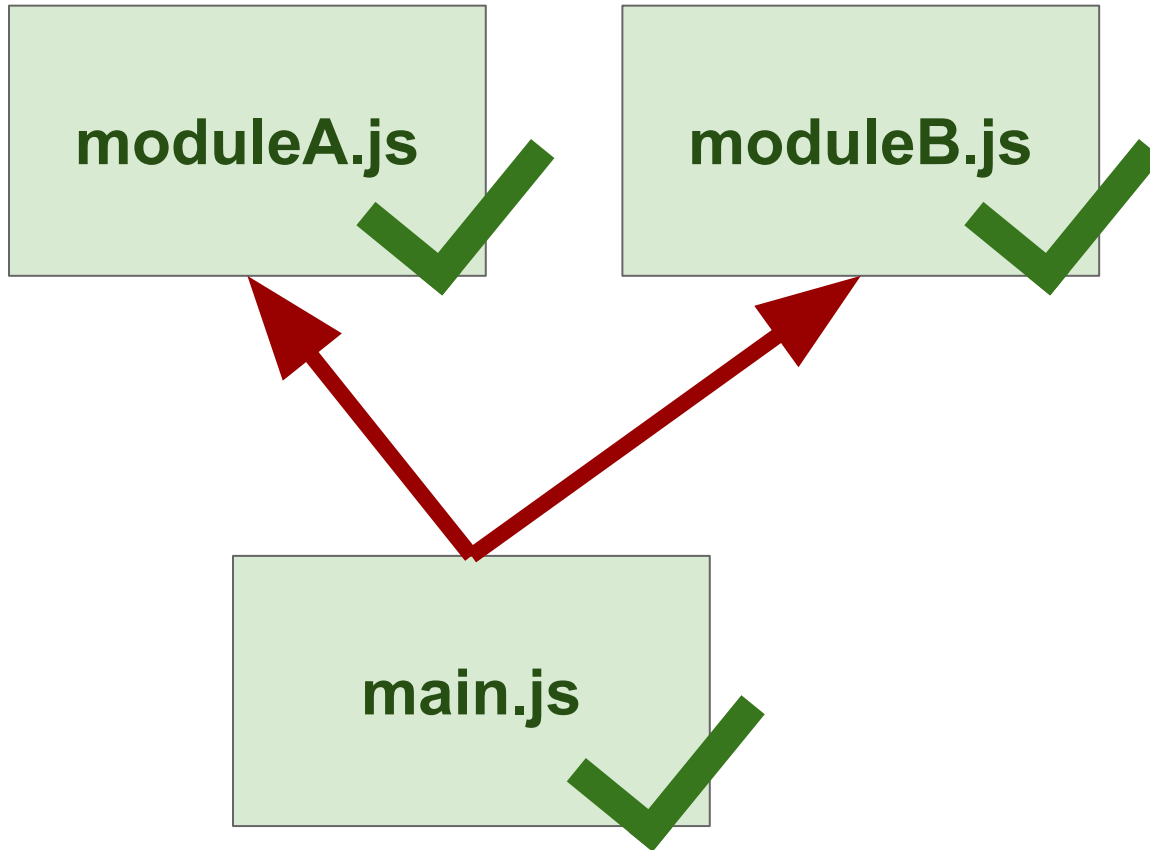
Workflow - Tools



Workflow - ESLint



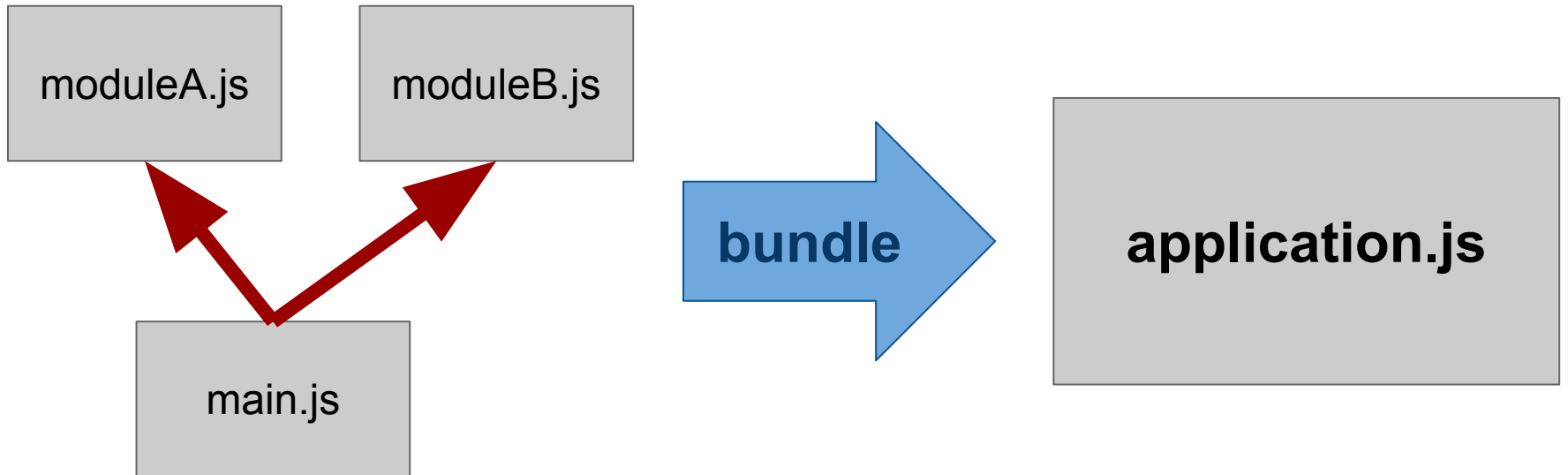
Source: <http://eslint.org/>



Workflow - Webpack



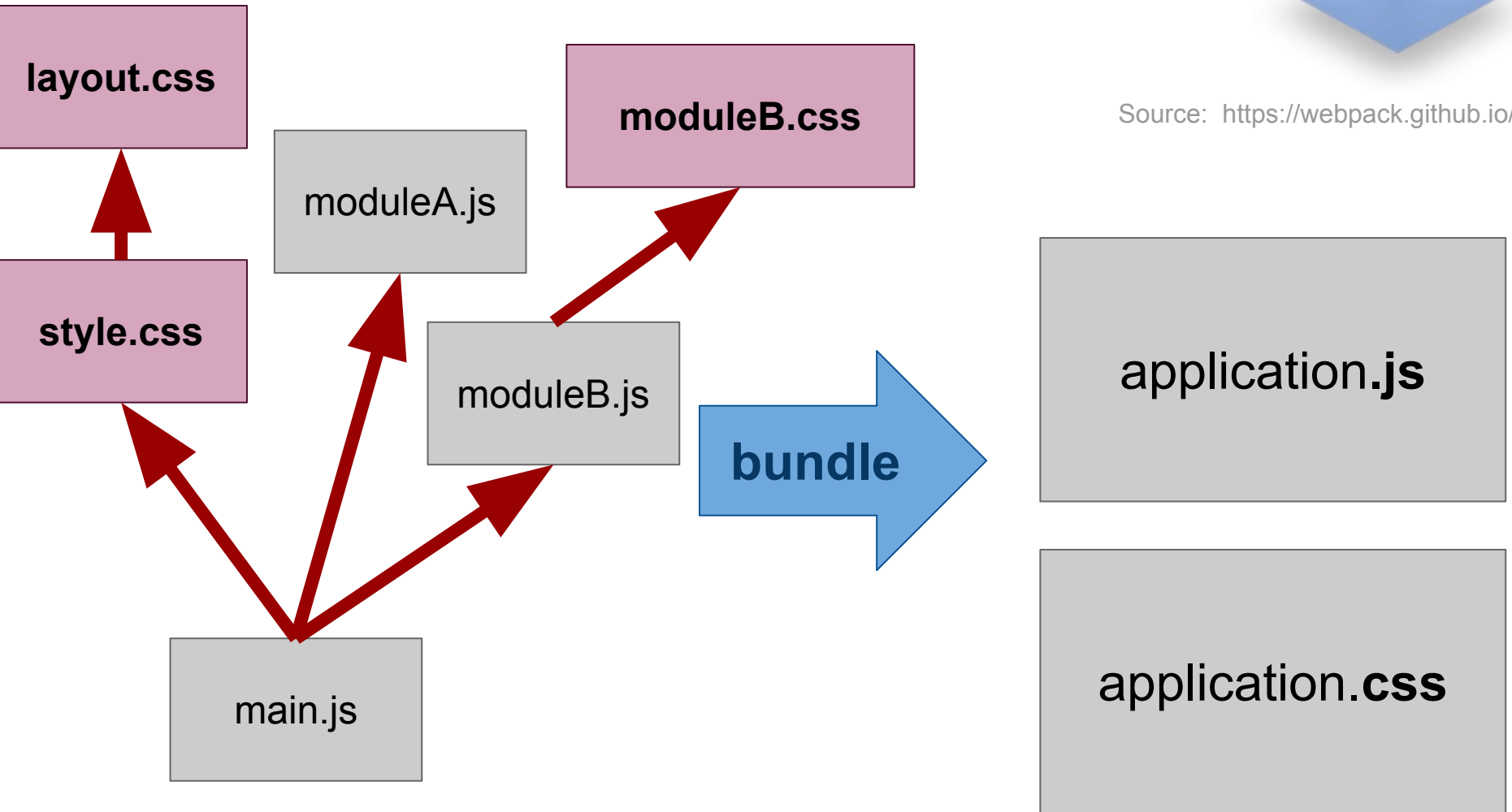
Source: <https://webpack.github.io/>



Workflow - Webpack



Source: <https://webpack.github.io/>



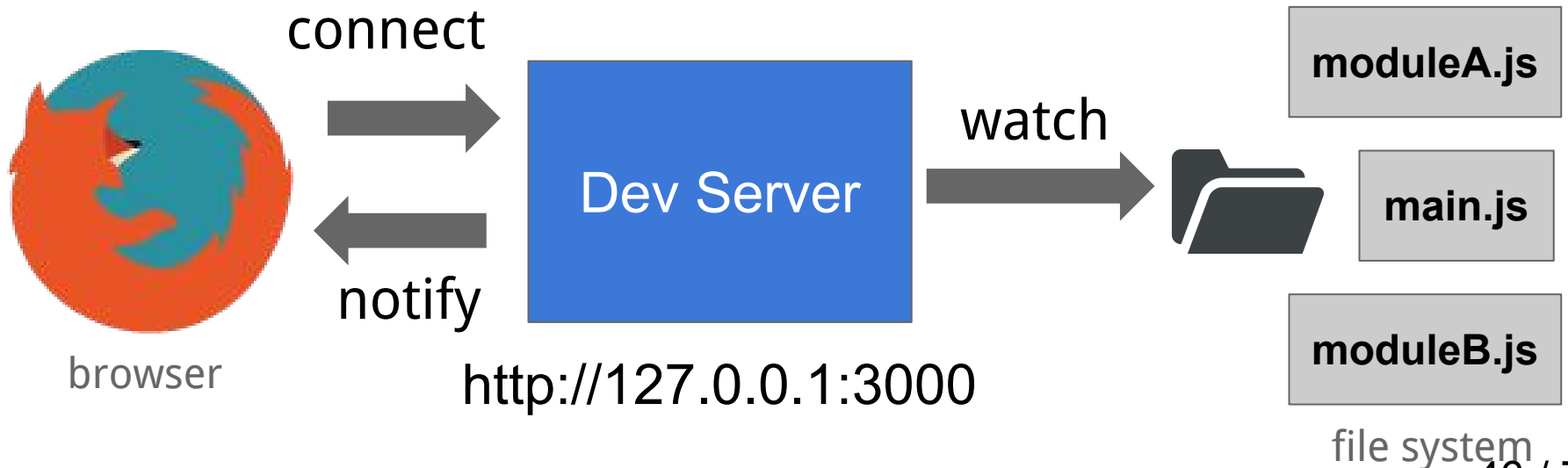
Workflow - Webpack Dev Server

Workflow - Webpack Dev Server

- No manual recompile
- Watches the file system for changes
- Pushes changes to browser (via websocket)

Workflow - Webpack Dev Server

- No manual recompile
- Watches the file system for changes
- Pushes changes to browser (via websocket)



Web Platform

Graphical User Interface

GUI - Graphical User Interface

- JavaScript provides multiple APIs for building GUIs

→ HTML / CSS

→ Canvas (2D bitmap)

→ Scalable Vector Graphics (SVG)

GUI - HTML / CSS

- Layout calculations are done by the browser
 - text wrapping, adjusting height of elements
- Native support for responsive layouts
- API for user interactions

- Only rectangular shapes

GUI - Canvas (2D Bitmap)

- Fine grained control on a pixel level
- All kinds of shapes (arcs, bézier curves...)

- Hard to support multiple screen sizes
- Hittesting needs to be done manually

GUI - Scalable Vector Graphics (SVG)

- XML based format for vector graphics
- Embeddable into HTML
- All kinds of shapes (arcs, bézier curves...)
- Support for Stylesheets
- API for user interactions (event handling)
- sharp on all display sizes
- **Manual layout** (text wrapping, element height)

GUI

- HTML / CSS for base layout
- SVG for advanced visualisations and icons
- no Canvas

Components

Demo

- Logic expression editor
- Logic expression checker
- Karnaugh map editor
- LED editor
- Number circle
- FSM editor

<https://thesis.laszlokorte.de/>

Demo - Logic expression editor

- Parse boolean expression
 - In various dialects
- Generate a function table
- Generate an operator tree
 - Color branches
- Enter multiple expressions

thesis.laszlokorte.de/demo/logic-editor.html

Demo - Logic expression editor - Details

- Expressions are parsed by PEG.js via parsing expression grammar
- Tree is drawn via Buchheim algorithm
(C. Buchheim, M. J Unger, and S. Leipert. Improving Walker's algorithm to run in linear time.)
- Labelled expressions are sorted topologically

Demo - Logic expression checker

- Simplified GUI for comparing two expressions
- Highlight differences in function table

thesis.laszlokorte.de/demo/logic-checker.html

Demo - Karnaugh map editor

- Create Karnaugh maps of various sizes
- Creating loops on a touch screen
- Show Programmable Logic Array
- Import and export expressions

thesis.laszlokorte.de/demo/kvd-editor.html

Demo - Karnaugh map editor - details

- PLA view is it's own component

Demo - LED editor

- Build function tables for 7-segment display...
- ... or other displays

thesis.laszlokorte.de/demo/led-editor.html

Demo - Number circle

- Arrange binary numbers in a circle
- Adjusting the circle size
- Compare various encodings
- Highlight overflows

thesis.laszlokorte.de/demo/number-circle.html

Demo - Finite-state machine editor

- Edit Moore and Mealy machines
- Edit transition graph
- Not yet implemented:
 - Accept logic expressions as transition conditions
 - simulating the machine

thesis.laszlokorte.de/demo/fsm-editor.html

Architecture

Architecture

- Based on Functional Reactive Programming
- Each component is a pure function, that transforms input to output
- Components do not modify any outside state
- inspired by Cycle.js library

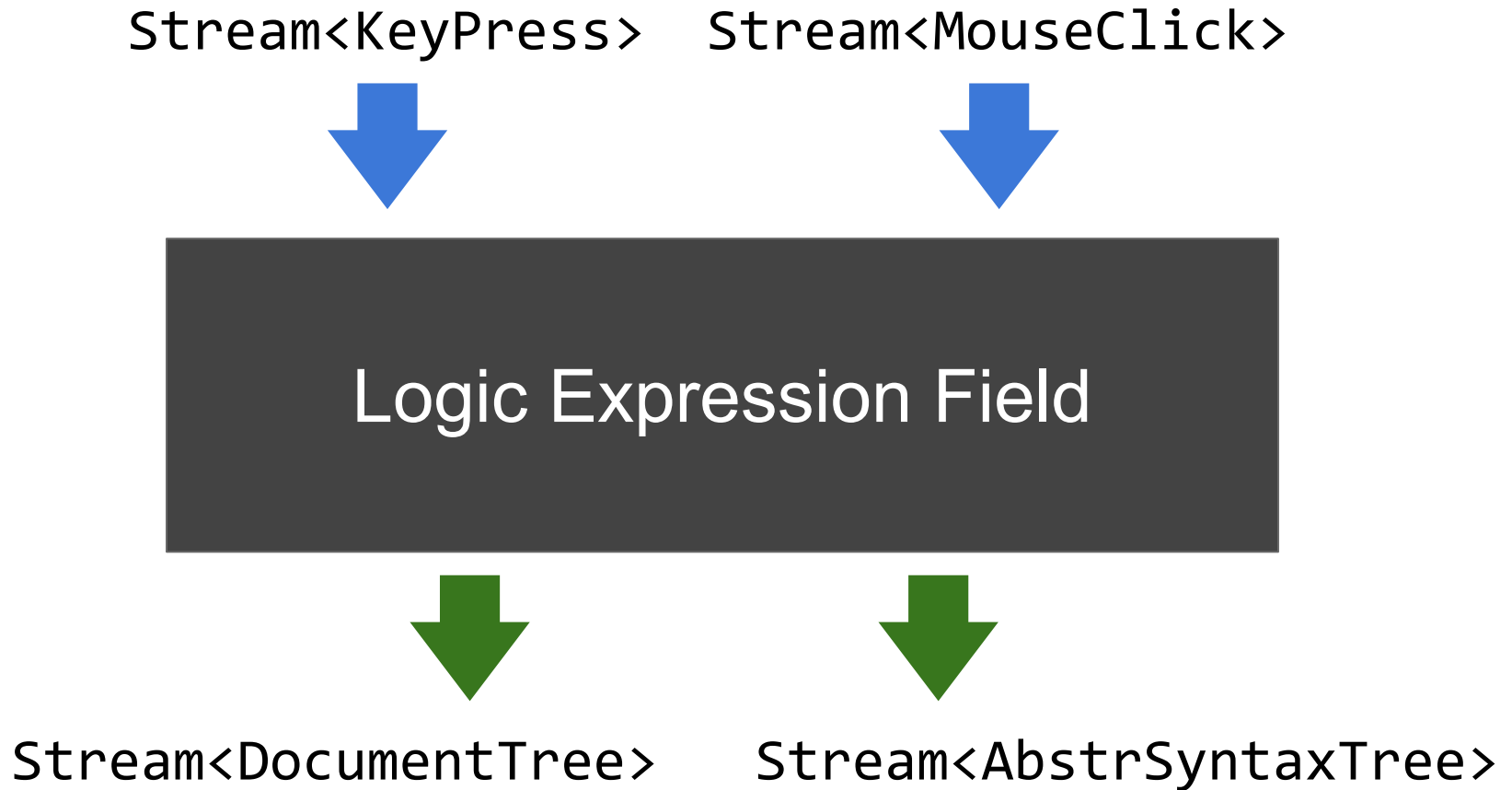


Source: <http://cycle.js.org>

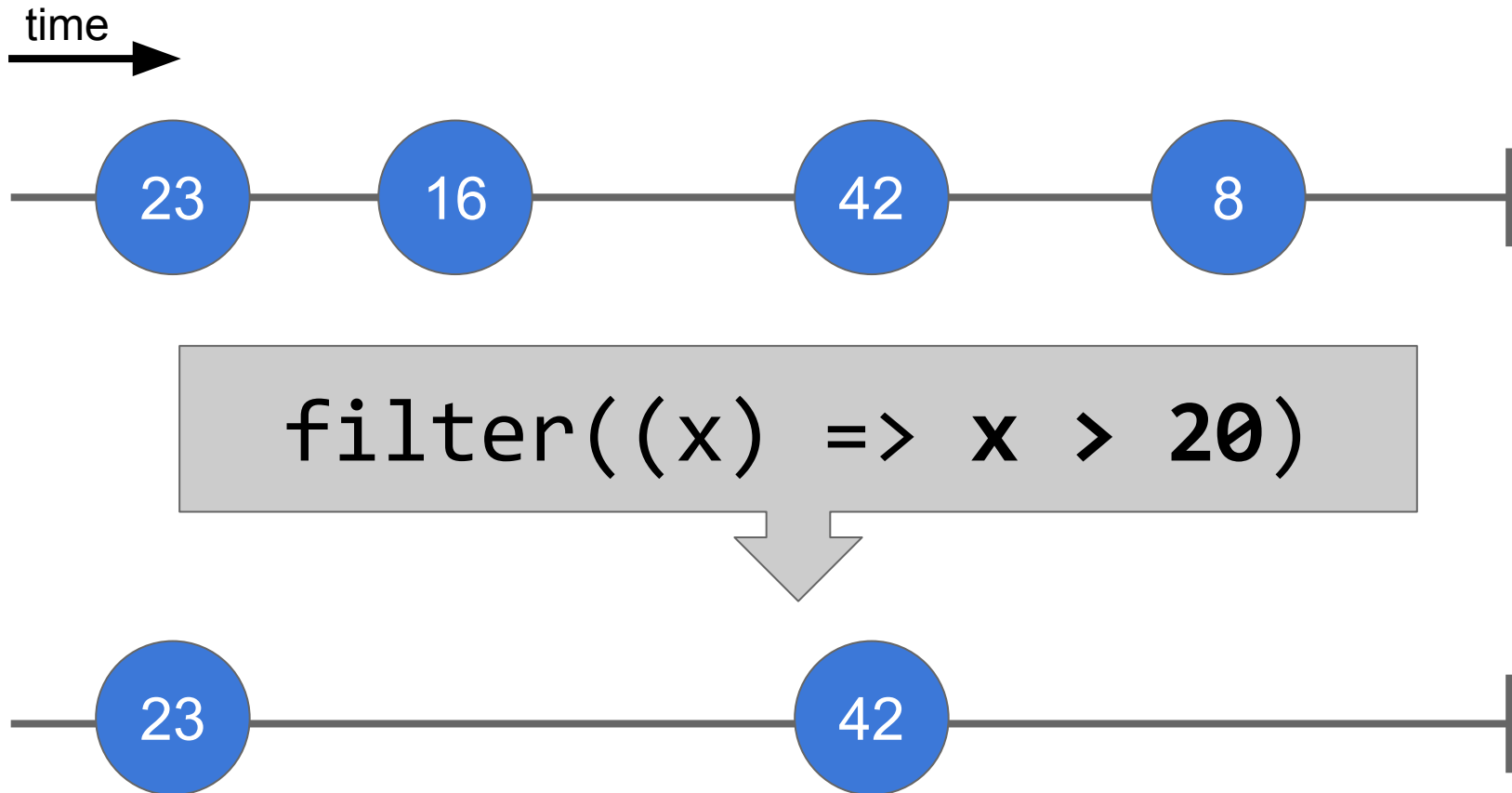
Architecture - Functional Reactive Programming

- **Events** and **state** are modelled as streams
- Streams represent data over time
- A component takes stream of events (clicks, key presses...) as input
- A component produces a stream of state (document state) as output

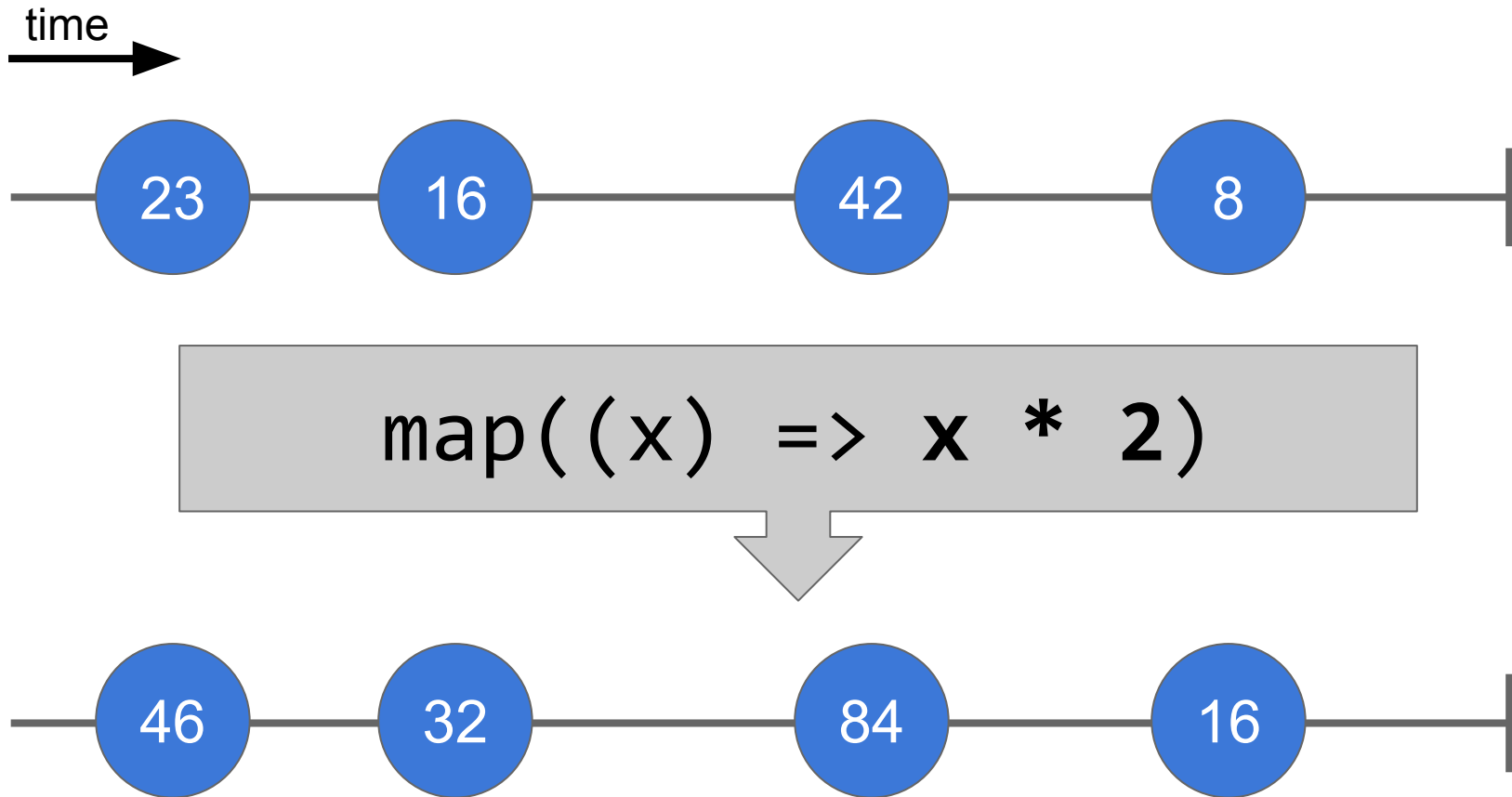
Architecture - Functional Reactive Programming



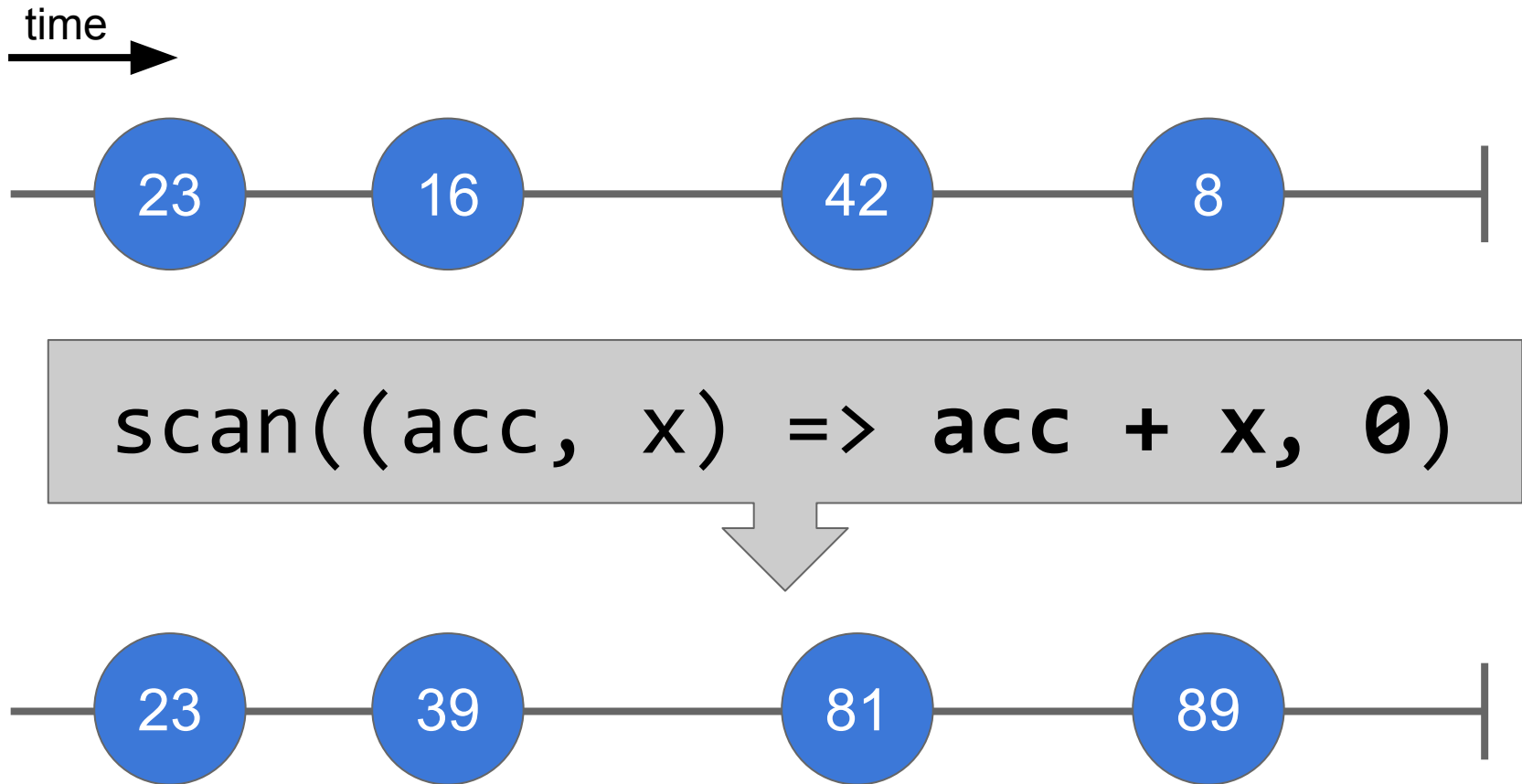
Architecture - Functional Reactive Programming



Architecture - Functional Reactive Programming



Architecture - Functional Reactive Programming



Architecture - Functional Reactive Programming

- many more operators available (reactivex.io)
- The whole GUI is built via stream transformation
- especially drag-drop and touch gestures are easy to build on top of event streams

Conclusion

Conclusion

- Components work across all major browsers:
 - Chrome, Firefox, Opera, Safari, IE11, Edge
- usable on various screen sizes
 - (sharp on retina screens)
- usable on touch screens

Conclusion

Stats:

- ~12.000 lines of JavaScript
- 217 JavaScript files
- 610 Git Commits

Conclusion

Potential improvements:

- Performance could be improved
- Printing support would be nice
 - eg. export expression tree or function table as PDF
- Expression parser via command line
- Splitting component into separate projects

Conclusion

Potential improvements (Cont.):

- Output file size could be further reduced
 - by tree shaking vendor libraries
 - currently ~500KB for Karnaugh maps
- For long term maintenance use a stronger language than JavaScript
 - eg. PureScript or TypeScript

Conclusion

Potential improvements (Cont.):

- No animations yet!
 - easy to built via stream transformation...
 - ... or CSS animations

Thank you for your attention! Questions?

- **Web Platform**
 - The Browser
 - NodeJS, NPM
 - Workflow (Babel, ESLint, Webpack)
 - Graphical User Interface (HTML, Canvas, SVG)
- **Components**
 - Logic Expression editor/checker
 - Karnaugh Maps
 - LED editor
 - Number Circle
 - FSM editor
- **Architecture**

Bonus - Karnaugh map recursion

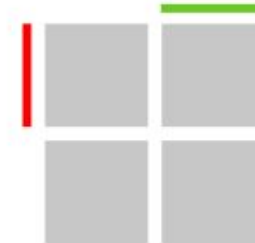
0 Inputs



1 input



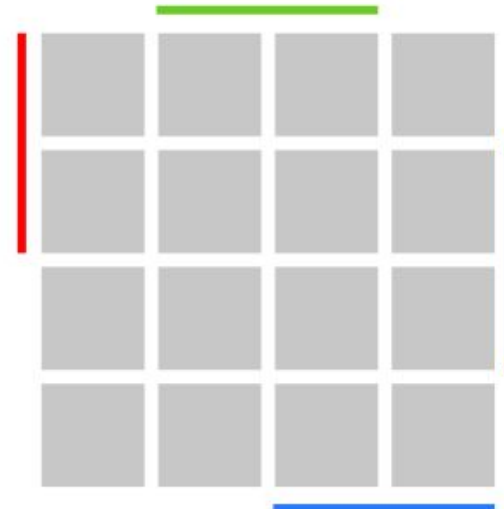
2 inputs



3 inputs

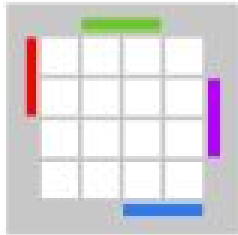


4 inputs

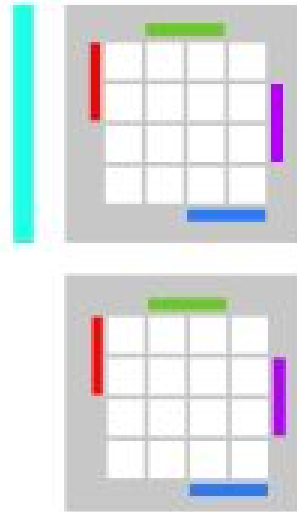


Bonus - Karnaugh map recursion (Cont.)

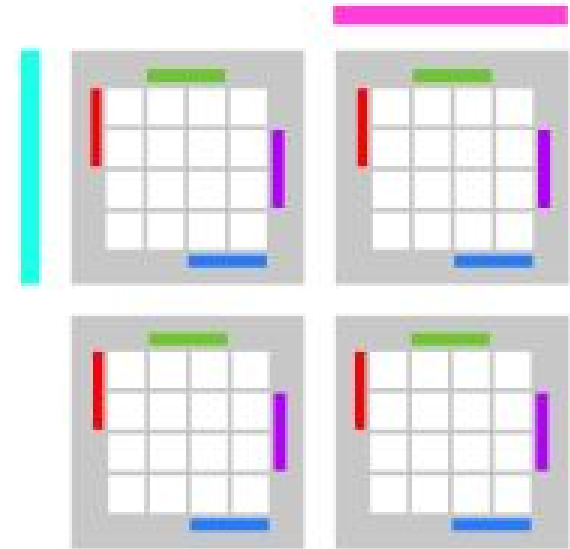
4 inputs



5 input

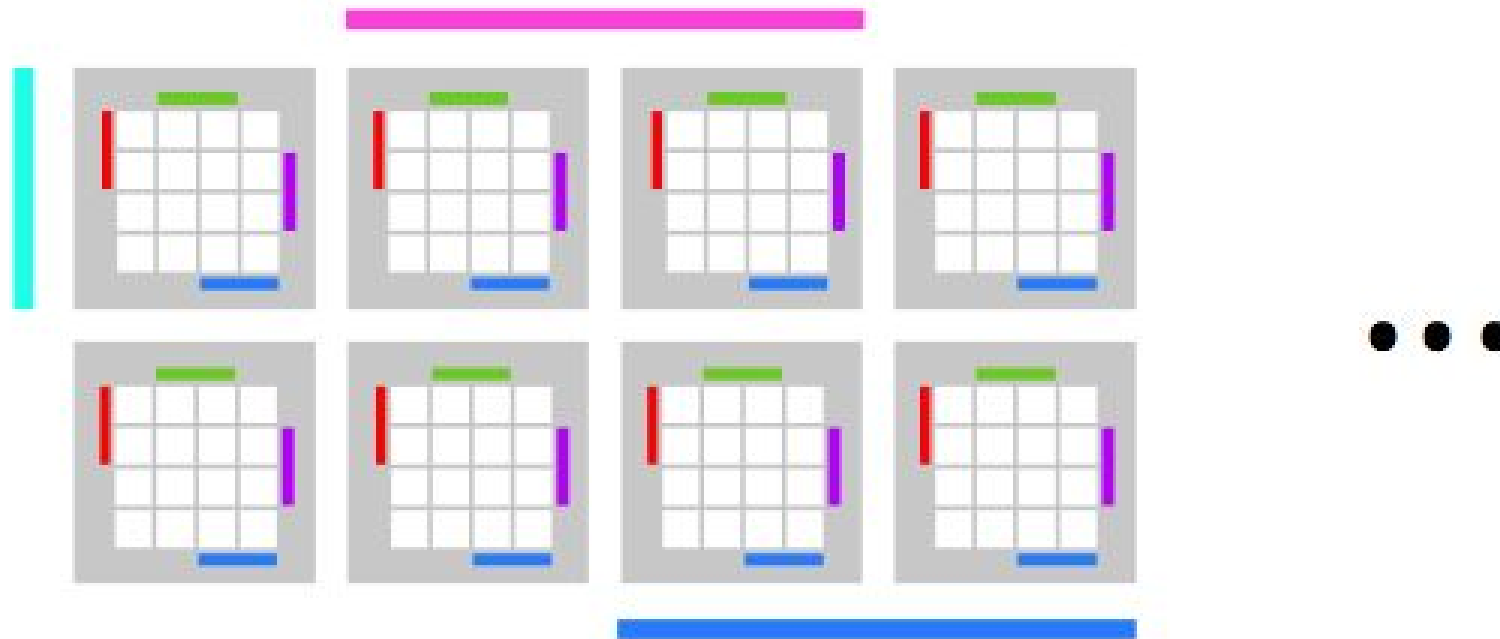


6 inputs



Bonus - Karnaugh map recursion (Cont. 2)

7 inputs



Bonus - Karnaugh map drag-'n-drop

1 Dimensional

Center Pair



Side Pair



Full Pairing

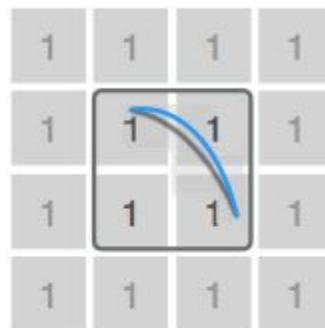


Outer Pairing

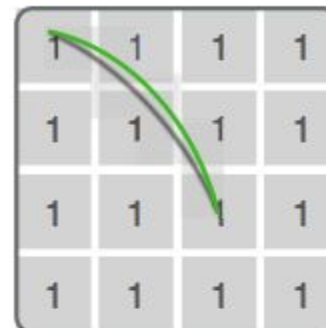


2 Dimensional

Center Pair



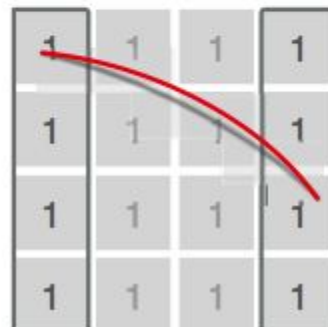
Full Pairing



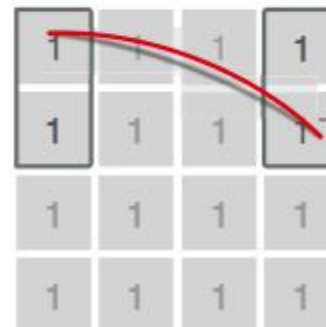
Outer Pairing



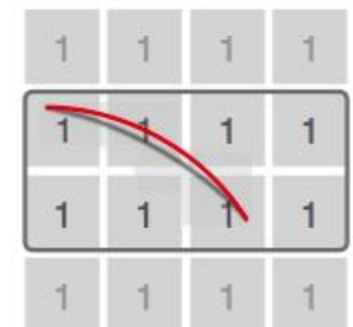
Mix: Full Outer Row Pair



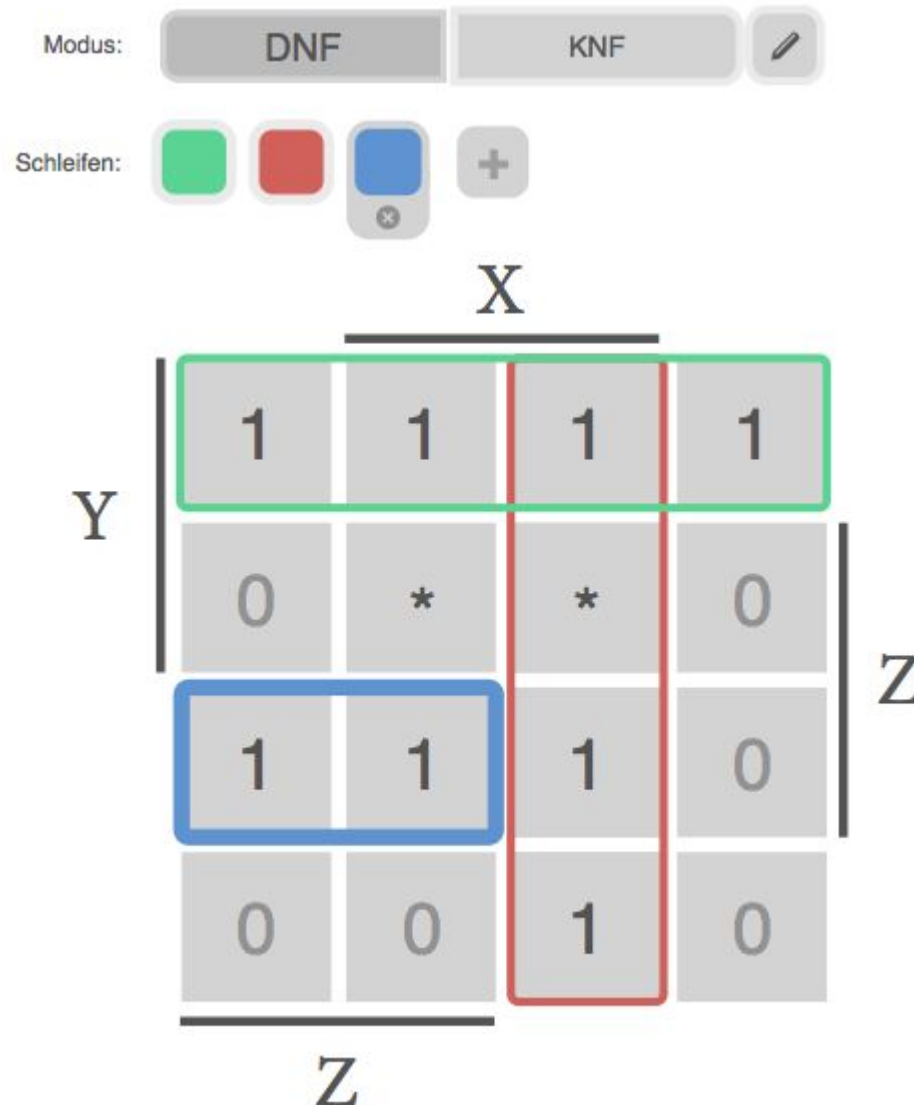
Mix: Outer Row Pair



Mix: Inner Row Pair



Bonus - Karnaugh map layout #1



Bonus - Karnaugh map layout #2

The screenshot shows a web-based Karnaugh map tool. At the top, there are controls for the number of inputs, currently set to 4. Below that, there are buttons for 'Loops:' and 'DNF', and a row of colored buttons for selecting different loop types. The main area displays four 4x4 Karnaugh maps labeled 'Output 1' through 'Output 4'. 'Output 3' is selected, and its map is shown in detail below. The map is a 4x4 grid with cells containing 0 or 1. A loop labeled 'B' is drawn around the top two rows. The map is also labeled with 'A' on the left and 'C' at the bottom.

	B				
	0	2	6	4	
	1	1	1	1	
	8	10	14	12	
	1	1	1	1	
	9	11	15	13	D
A	1	0	0	1	
	1	0	0	1	
	1	0	0	1	5
					C

Bonus - Karnaugh map layout #3 a

The screenshot shows a Karnaugh map editor interface. At the top, there is a header with a question mark, a "Load Example..." button, a zoom control showing "4 Inputs" with minus and plus signs, and an "Export..." button. Below the header, there are three 4x4 grid icons labeled "O1", "O2", and "O3". The "O3" icon is selected and has a red 'x' mark. To the right of these icons is a plus sign and a button labeled "Schleifen bearbeiten". Below the icons, there are tabs for "DNF" and "KNF".

The main area displays a 4x4 Karnaugh map. The columns are labeled "B" and the rows are labeled "A" and "D". The map contains the following values:

	B			
	0	1	0	0
A	1	0	1	0
	1	1	1	1
	0	1	1	0
	C			

The map shows a pink loop around the row of 1s (row 3) and a green loop around the column of 1s (column 2). The value "03" is entered in a text box below the "O3" icon.

At the bottom right, there are three tabs: "Value", "Decimal Index", and "Binary Index".

Bonus - Karnaugh map layout #3 b

The screenshot shows a digital logic design tool interface. At the top, there is a toolbar with a 'Load Example...' button, a row of colored squares, and an 'Export...' button. Below the toolbar, there are two tabs: 'DNF' (selected) and 'KNF'. A 'Funktion bearbeiten' button is on the left. In the center, three 4x4 Karnaugh maps are shown, labeled O1, O2, and O3. O3 is the active map, showing a 4x4 grid of cells with values 0 or 1. The map is labeled with 'A' on the left, 'B' on top, and 'C' and 'D' on the bottom and right respectively. A pink rectangle highlights a prime implicant covering the entire row where A=1. A green rectangle highlights a prime implicant covering the two columns where B=1. To the right of the map, there are columns for 'Value', 'Decimal Index', and 'Binary Index'. On the right side of the interface, a logic circuit is shown with the title 'Kosten: 6 Gatter mit 22 Ports'. The circuit has four inputs labeled A, B, C, and D. Each input line passes through an inverter. There are four 3-input AND gates. The first three AND gates are connected to the inputs A, B, and C. The fourth AND gate is connected to the inputs A, B, and D. The outputs of all four AND gates are connected to a single 4-input OR gate, which produces the output labeled 'o1'.

	B			
	0	1	0	0
	1	0	1	0
A	1	1	1	1
	0	1	1	0
	C			