Lecture 05: Cellular Automata & Interactions

COSC 225: Algorithms and Visualization

Spring, 2023

Outline

1. Cellular Automata
2. Activity: Rule 90
3. JavaScript and Node.js
4. CSS Animations
5. JS Events

Last Time: JavaScript

```
<!doctype html>
<html lang=en>
  <head>
    <meta charset=utf-8>
    <title>Page Title</title>
    <script src="hello.js"></script>
    <script>
          ...javascript code here...
    </script>
  </head>
</html>
```

Basic Tasks

- Get an element in the document (`selector` is like CSS selector)

```
const someElement document.querySelector("selector");
```

  first element in document matching selector is returned

- Create an element (`some-tag` is desired tag of element)

```
let myElement = document.createElement("some-tag");
```

- Add text to element

```
myElement.textContent = "some text";
```

- Add element as child of another

```
someElement.appendChild(myElement);
```

Adding Style

If `someElement` is an element, we can...

- set an id

```
someElement.id = "some-id";
```

- add a class

```
someElement.classList.add("some-class");
```

- add a style

```
someElement.style.backgroundColor = "rgb(200,200,200)";
```

This Week

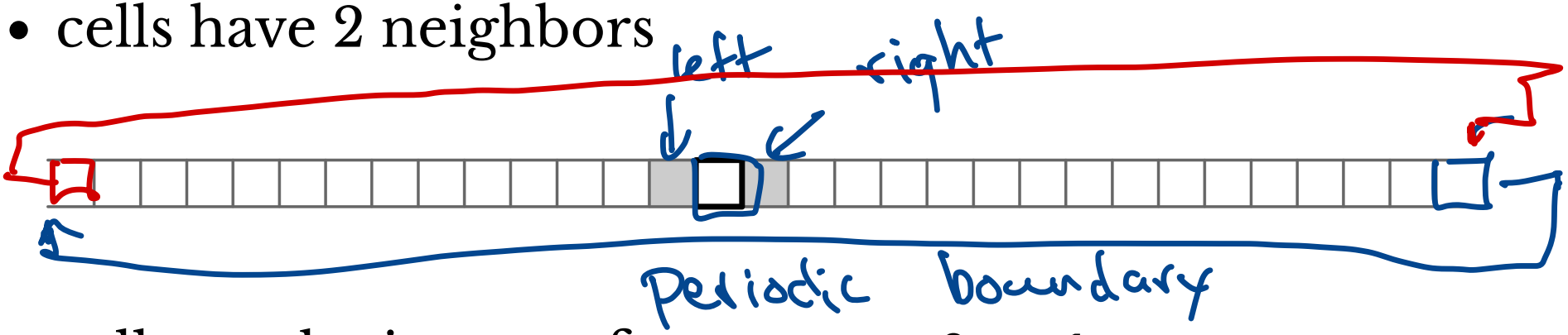
Cellular Automata

- use JavaScript to make generative graphics

Cellular Automata (1D, 2 State)

A cellular automaton consists of

- a (circular) array of **cells**
- cells have 2 neighbors



- cells can be in one of two states: 0 or 1
- a **configuration** assigns states to each cell

Updating Rule

In a single **step**, each cell updates its state based on

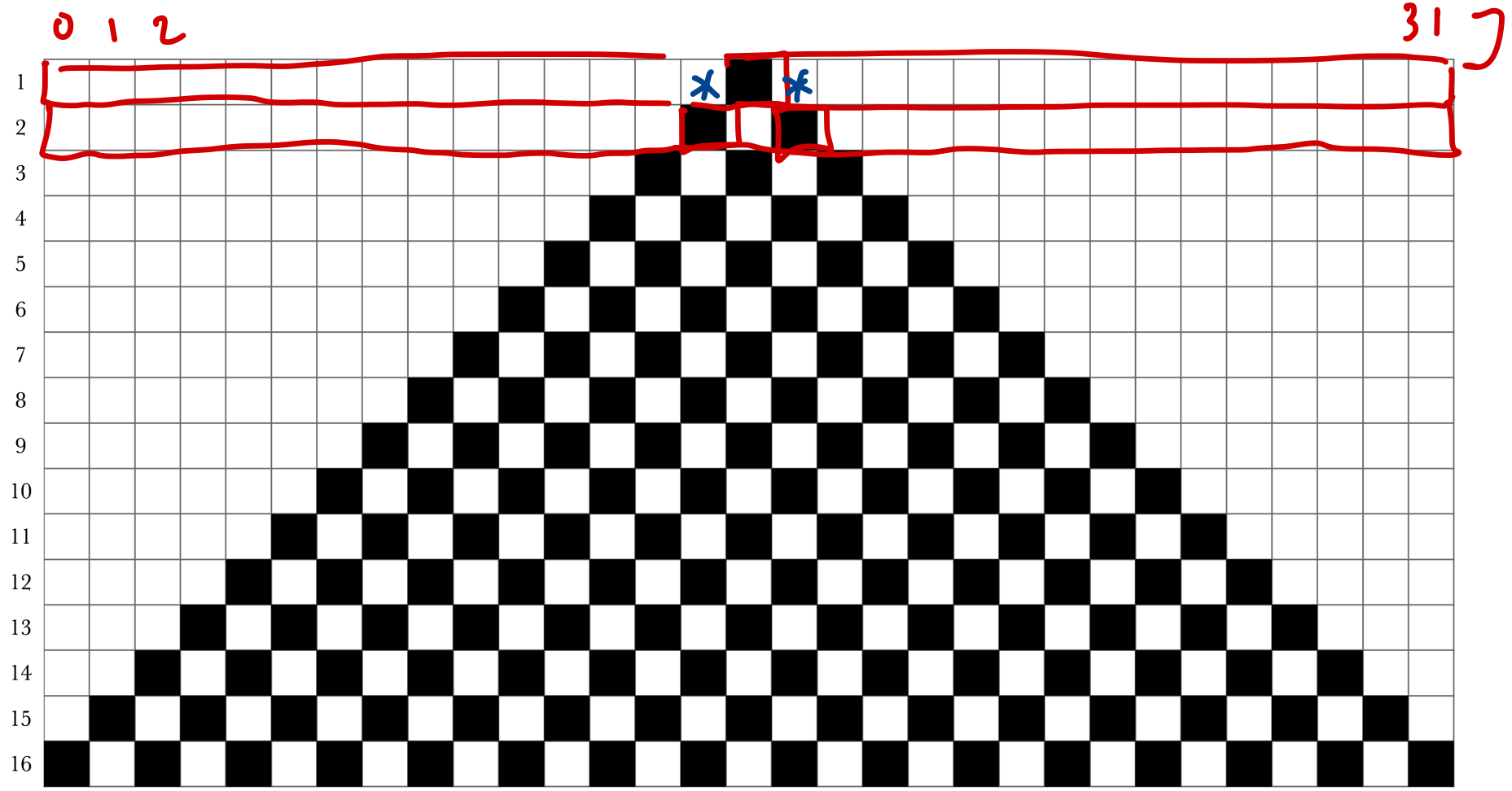
- its current state
- state of neighboring nodes

Space-time diagram shows evolution over time

- each row is a configuration

Example

Update rule: update to `1` if either neighbor was 1, update to `0` otherwise



All Possible Rules

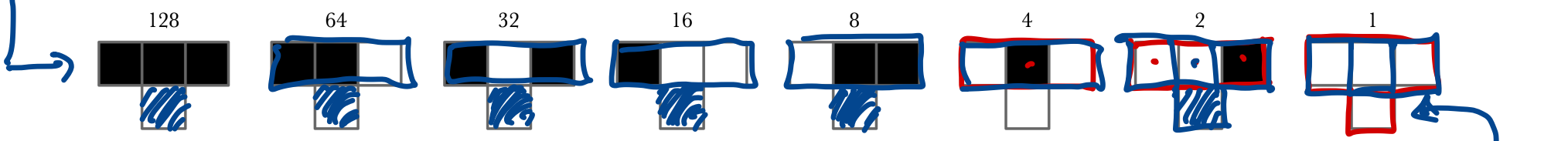
Updated state depends on 3 states:

- left neighbor's state
- own state
- right neighbor's state

There are $8 = 2^3$ possibilities that must be considered

A **rule** determines update state for each possibility

current state



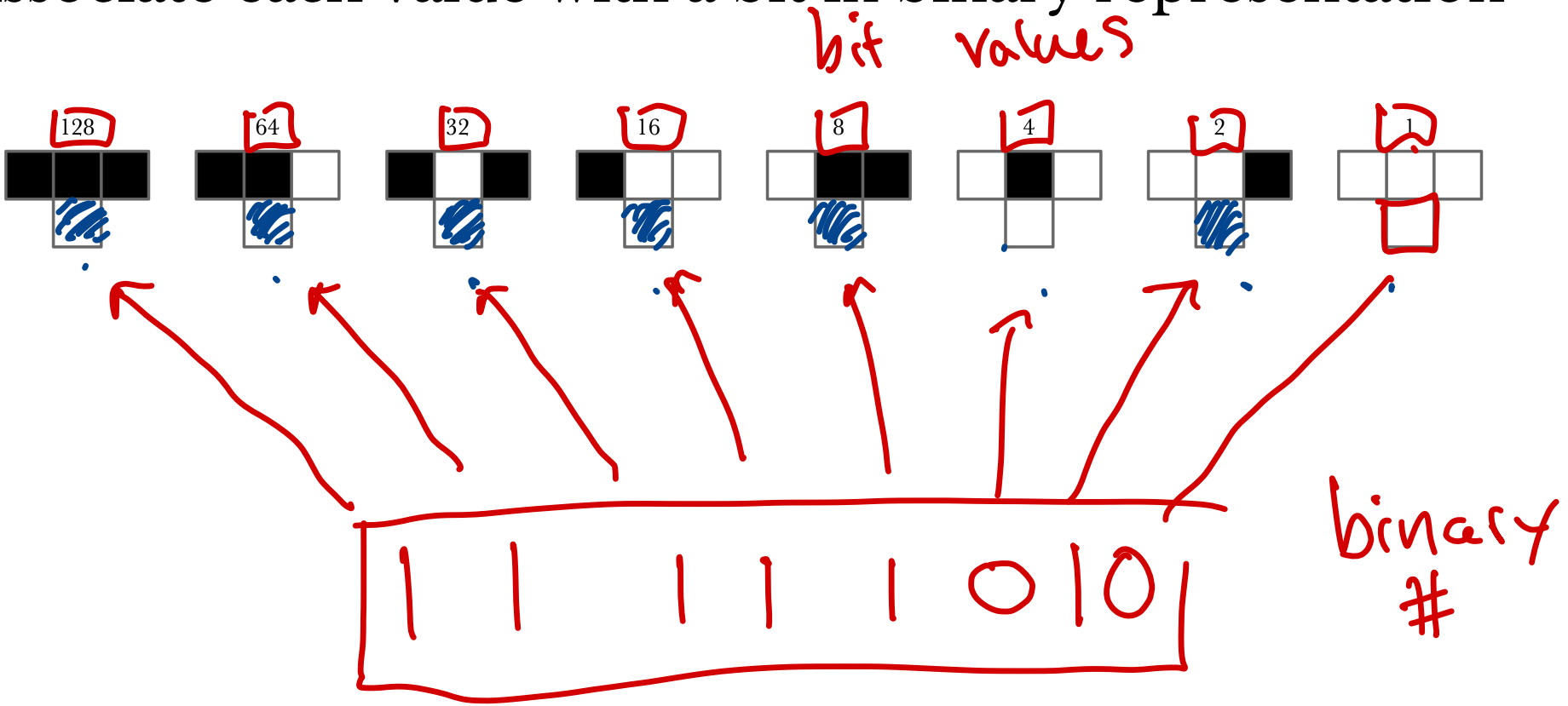
Example: update to 1 if either is 1

middle cell's new state

Naming Convention

$2^8 = 256$

Associate each value with a bit in binary representation



Convention: Wolfram "A New Kind of Science"

Simulating CA by Hand (or JavaScript)

Input:

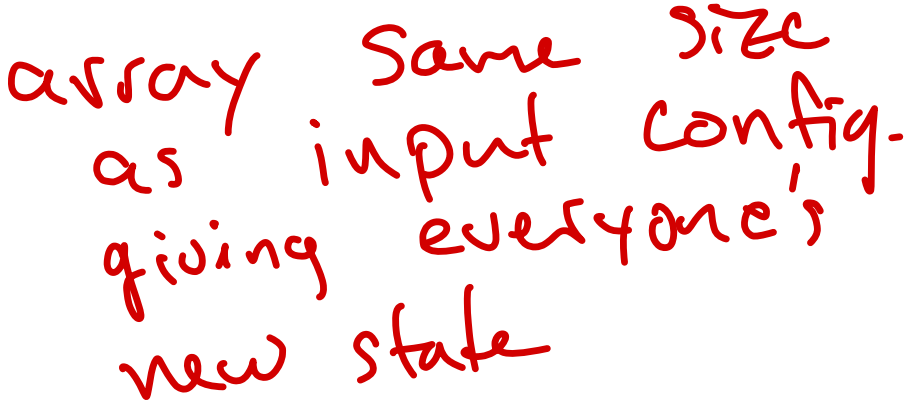
- rule (a number from `0` to `255`)
- configuration (array of `0`s and `1`s)

Output:

- updated configuration

To draw space-time diagram:

- do this repeatedly



Activity

Apply Rule 90!

Assignment 04

Visualize cellular automata to make a cool site!

- must include `cellular-automata.js`
  - must have method `applyRule(config, rule)`
  - `config` a `0`-`1` array
  - `rule` a number from `0` to `255`
  - application has **periodic boundary conditions**

HW 04 Demo

Running and Debugging JavaScript

Node.js

- a JavaScript runtime environment
- run JavaScript outside of a web browser

We'll use Node.js to test your assignment submission!

Node.js Example

- Running .js file from command line
- Interactive mode!
  - .load file
  - .help
  - .exit

Interactions

CSS Interactions

In Assignment 03, you added interactions to your grid with the `:hover` pseudo-class:

```
.tile:hover {
    border-color: white !important;
    z-index: 100;
}
```

CSS Transformations

CSS can do more interesting transformations:

- `scale(x-scale, y-scale)`
- `rotate(amount)` (deg)
- `translate(x-amount, y-amount)` (px)
- `skew(x-skew, y-skew)` (deg)

For example:

```
.tile:hover {
    transform: [transformations];
}
```

[Color grid demo]

Those are cool, but...

...I'd like to see some motion...

Those are cool, but...

...I'd like to see some motion...

CSS can do animations too!

```
.tile:hover {
    animation-name: some-animation;
    animation-duration: 1s;
    animation-iteration-count: 1;
}

@keyframes some-animation {
    from {
        /* initial state */}
    to {
            /* final state */}
}
```

Check it out!

More Interactions!

To have more **robust** interactions, we need JavaScript

- execute methods in response to **events**

Add an **event listener** to an element

```
let box = document.querySelector('#some-box');
box.addEventListener('event-name', someMethod(e));
```

Some Events:

- `"click"` element is clicked
- `"mouseover"`, `"mouseout"`
- keyboard events

...there are a lot!

Events also have properties:

- `e.target` the element that event happened to
- `e.clientX`, `e.clientY` relative coordinates of where mouse cursor was when the event occured

Demo

Let’s make our grid more interactive!

Next Time

- Scalable Vector Graphics (SVG)
- Objects in JavaScript