JavaScript: Introduction, Types

Lecture 22

History

- Developed by Netscape
  - "LiveScript", then renamed JavaScript
  - *Nothing* to do with Java!
- Motivation: client-side execution in browser
  - Interpreted
- Standardized by ECMA ("ECMAScript")
  - Big update v6 in 2015, ie ES6 (aka ES2015)
  - Now annual updates, every June
  - After ES6, named with year (eg ES2024)
- Has become popular outside of browsers
  - Node.js
- Translation target for other languages:
  - Syntax: CoffeeScript
  - Static types: Dart (Google), TypeScript (MS)

Client-Side Execution


```
GET /news/index.php HTTP/1.1
Host: www.osu.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu;…etc
```

Request

```
<!DOCTYPE html>
<html lang="en">
  <head><title>My Page</title>
  <meta charset="utf-8" />
  …
```

Client-Side Execution


```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Something Short and Sweet</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <p>
      Hello <a href="planet.html">World</a>!
      <br />
      <img src="globe.png" alt="a globe"/>
    </p>
  </body>
</html>
```

Client-Side Execution


```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Something Short and Sweet</title>
    <meta charset="utf-8" />
    <script>
      window.alert("Annoying!");
    </script>
  </head>
  <body>
    <p>
      Hello <a href="planet.html">World</a>!
      <br />
      <img src="globe.png" alt="a globe"/>
    </p>
  </body>
</html>
```

Including Scripts

- Head: executed *before* body displays
  - Script (source) can be explicitly included

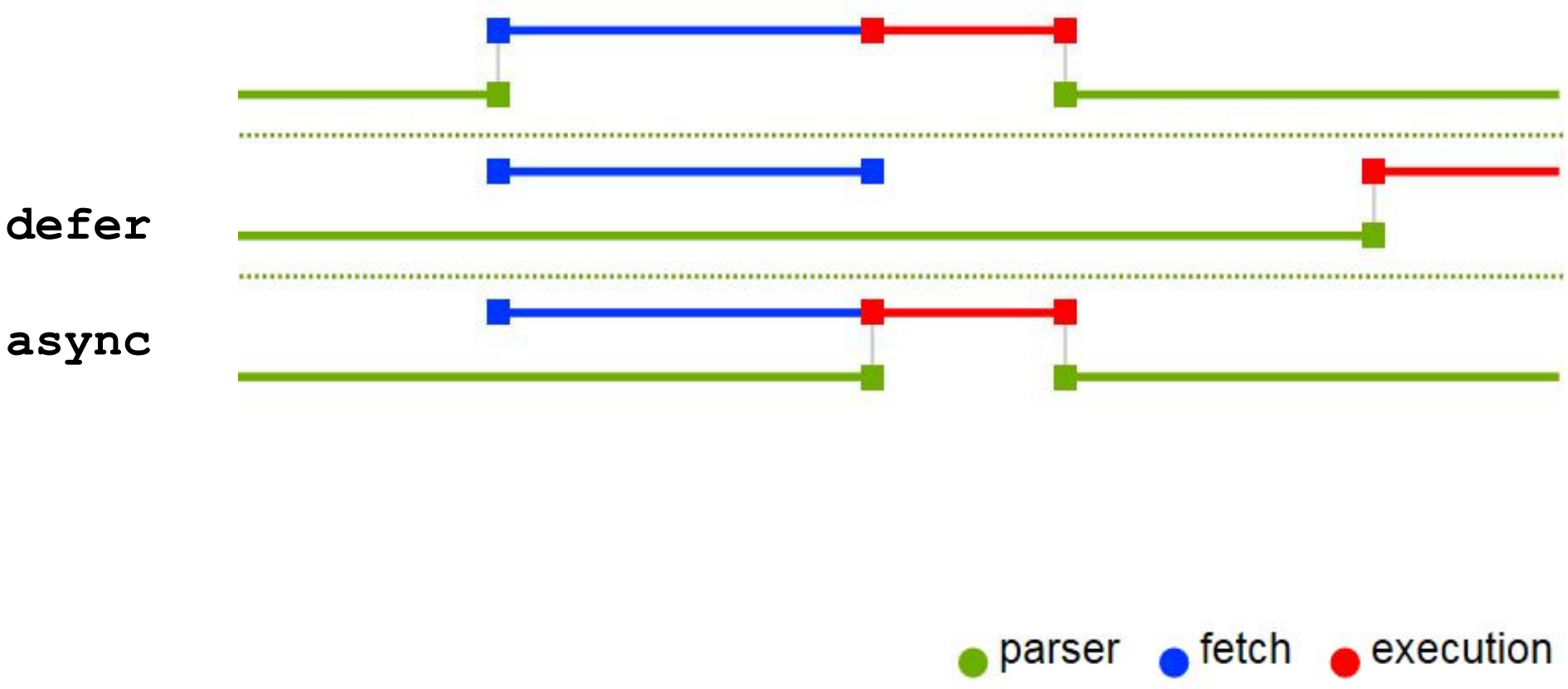
```
<script>
  console.info("hi");
  ...
</script>
```

  - Script can be linked in from external file

```
<script src="MyProgram.js"></script>
```

  - Recall: linking to CSS
- Inline: executed as body is displayed
- Browser blocks while downloading
  - Common advice: put scripts at *end of body*
  - Modern advice: use **<script src="…" async>**

Async/defer Downloading



Some Objects Provided Implicitly

- Some objects are created implicitly by the execution environment (browser)
- Document object (**`document`**)
  - **`document`**`.writeln()` puts output in body
- Window object (**`window`**)
  - Refers to browser's display window
  - Alert method pops up a dialogue

```
window.alert("Say \"cheese\"!");
```

  - Prompt method pops up a dialogue

```
name = window.prompt("Enter name");
```

Demo with Popups

- See: codepen.io/cse3901/pen/BYqqPb
  - Alert window
  - Prompt window
  - Console output (info, warn, error)
- Notice:
  - HTML body is empty
  - Settings > Auto-update preview (Off)

Familiar (Java) Minor Syntax

- Statement separator `;`
  - Wrinkle: `;`'s are optional!
    - Implicitly automatically inserted
    - But clearer and safer to include explicitly
- Statement blocks `{…}`
- Parentheses in expressions `(…)`
- Comments `//` and `/*…*/`

Familiar (Java) Operators

- Arithmetic (numbers are floats)
  - `+ - * / %`
  - Wrinkles:
    - No diff in `/` between ints and floats!
    - `%` works on floats!
- Relational
  - `< > <= >=`
  - `== !=`
  - Wrinkle: `===  !==`
- Logical
  - `&& || !`

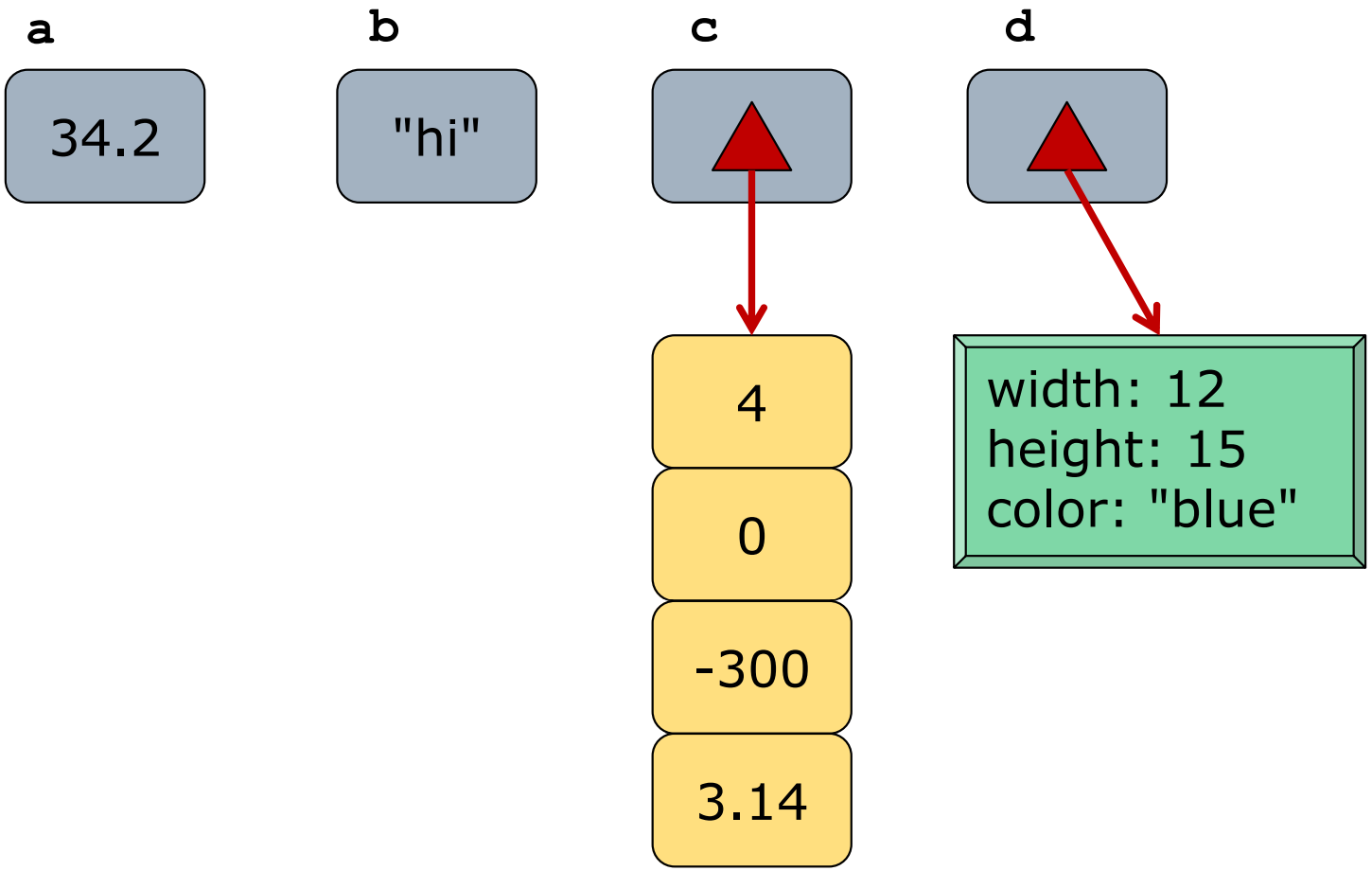
Familiar (Java) Statements

- Assignment
  - `=`
  - `+= -= *= /= %=`
  - `++ --` (pre and post)
- Conditionals
  - `if (…)`, `if (…) … else`
  - `switch (c)`
    `case 'a': … case 'b': … default;`
- Iteration
  - `while (…)`, `do…while(…)`
  - `for (…;…;…)`
  - `break`, `continue`

Primitive vs Reference Types

- Distinction is similar to Java
- A variable is a "slot" in memory
- A variable can be *primitive*
  - The slot holds the value itself
  - Boolean, number, string, null, undefined
  - Since ECMAScript 2015 (ES6): symbols
- A variable can be a *reference*
  - The slot holds a pointer to the value
  - Arrays and objects (including functions!)

Primitive vs Reference Types



Primitives: Checking Equality

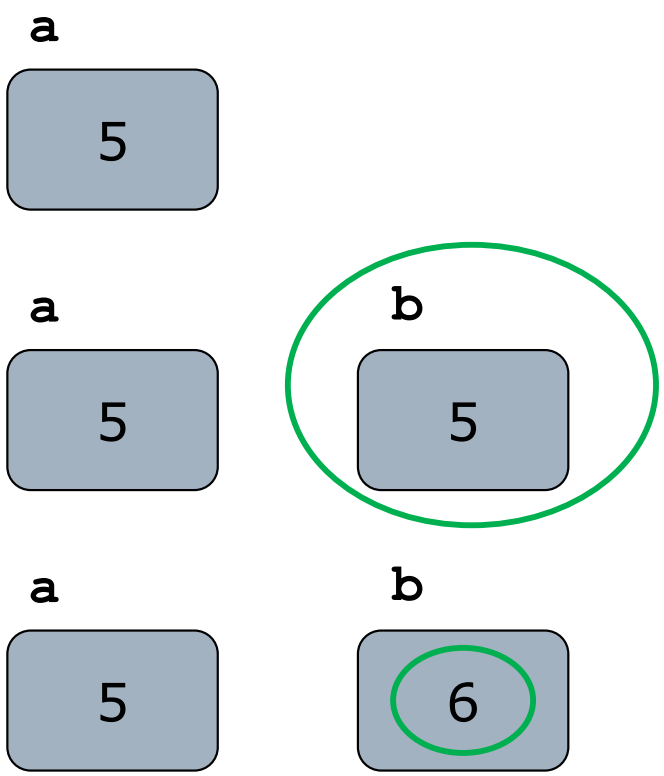

```
let a = 5;
let b = 5;
let c = 7;

if (a == b)… //=> true, equal slots
if (a == c)… //=> false

let x = "hello";
let y = "hello";

if (x == y)… //=> true! cf. Java
```

Primitives: Assignment is Copy


```
let a = 5;
let b = a; // copy contents of slot

b++;

if (a == 5)… //=> true, a unchanged
```

Assignment is Copy (of Slot)


```
let a = 5;
let b = a;

b++;

if (a == 5)…
```



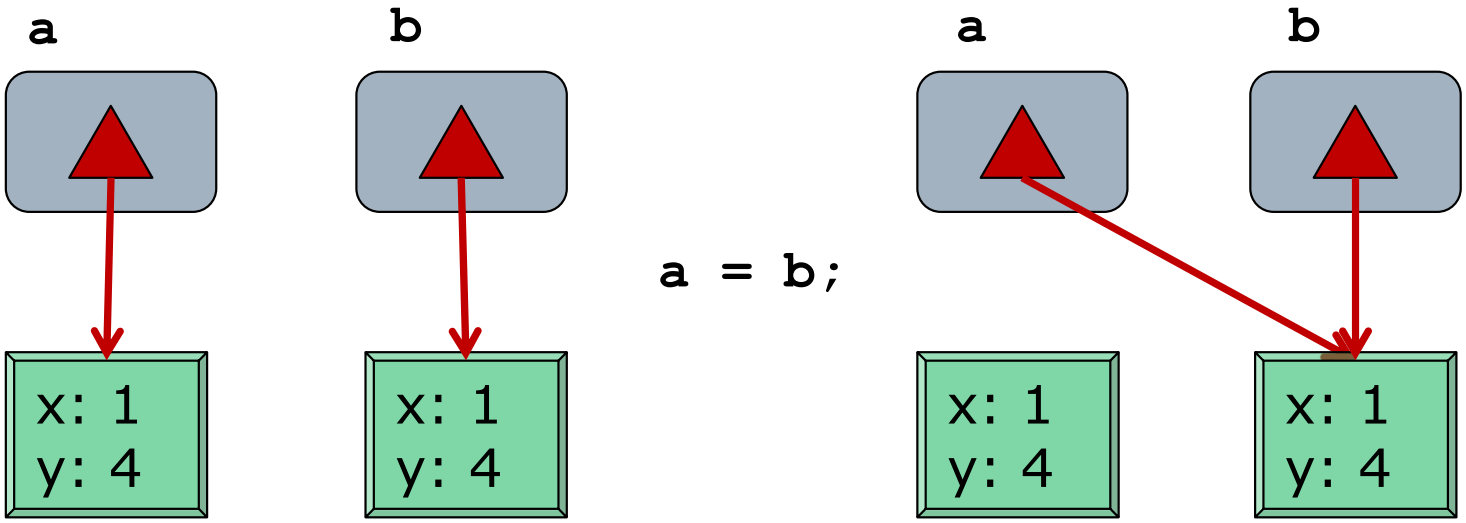
Primitives: Argument Passing


```
function inc (param) {
  param++;
}

let a = 5;
inc(a); // copy contents of slot
if (a == 5)… //=> true
```

References: Equality/Assignment


```
let a = {x:1, y:4}; // a new object
let b = {x:1, y:4}; // a new object

if (a == b)… //=> false

a = b; // copy contents of slot

if (a == b)… //=> true
```

Assignment is Copy (of Slot)



`a != b`

`a == b`

References: Argument Passing


```
function inc (param) {
  param.x++;
}

let a = {x: 1, y: 4};
inc(a); // copy contents of slot
if (a.x == 2)… //=> ??
```

References: Argument Passing


```
function inc (param) {
  param = {x: 2, y: 7};
}

let a = {x: 1, y: 4};
inc(a); // copy contents of slot
if (a.x == 2)… //=> ??
```

Wrinkle: == vs ===

- Recall + operator in Java
  - Concatenation between strings
  - Addition between numbers
  - `3 + "4"` also works! Results in `"34"`
- Similarly, JavaScript == (!=) tries to make types match
  - `3 == "3"` is true!
- To prevent implicit type conversion, use === (!==)
  - `3 === "3"` is false
- More on type conversion later…

Demo: Iteration

- See: codepen.io/cse3901/pen/Jpmejp
- Table generated by Javascript
  - Prompt for initial value
  - Calculate interest series
  - Print out a row of table for each year

Static vs Dynamic Types

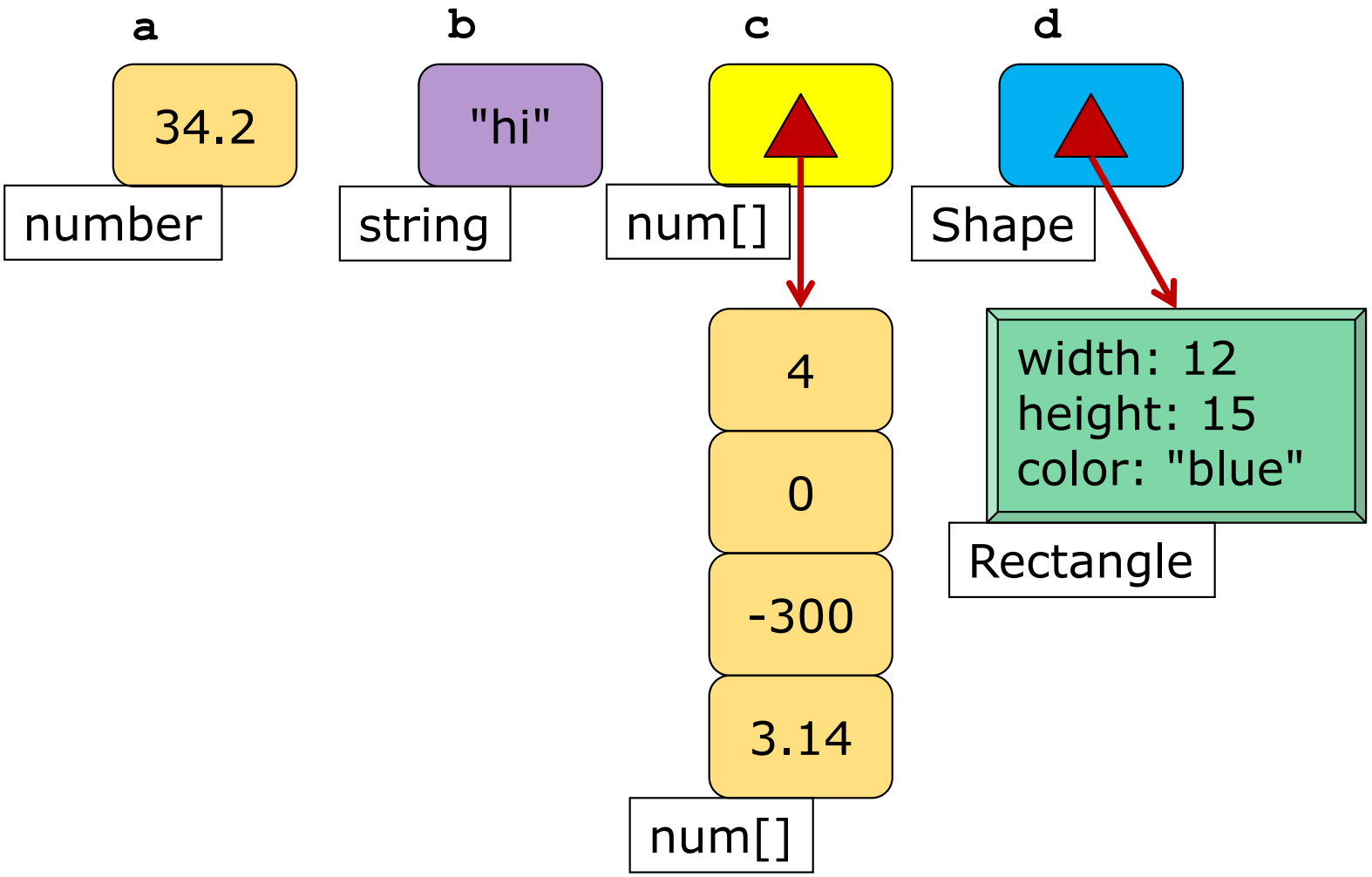
- Static: known at compile time
  - *e.g.*, C, C++, Java, Ada

```
int x
char[] a
FluffyCloud t
void* d
```

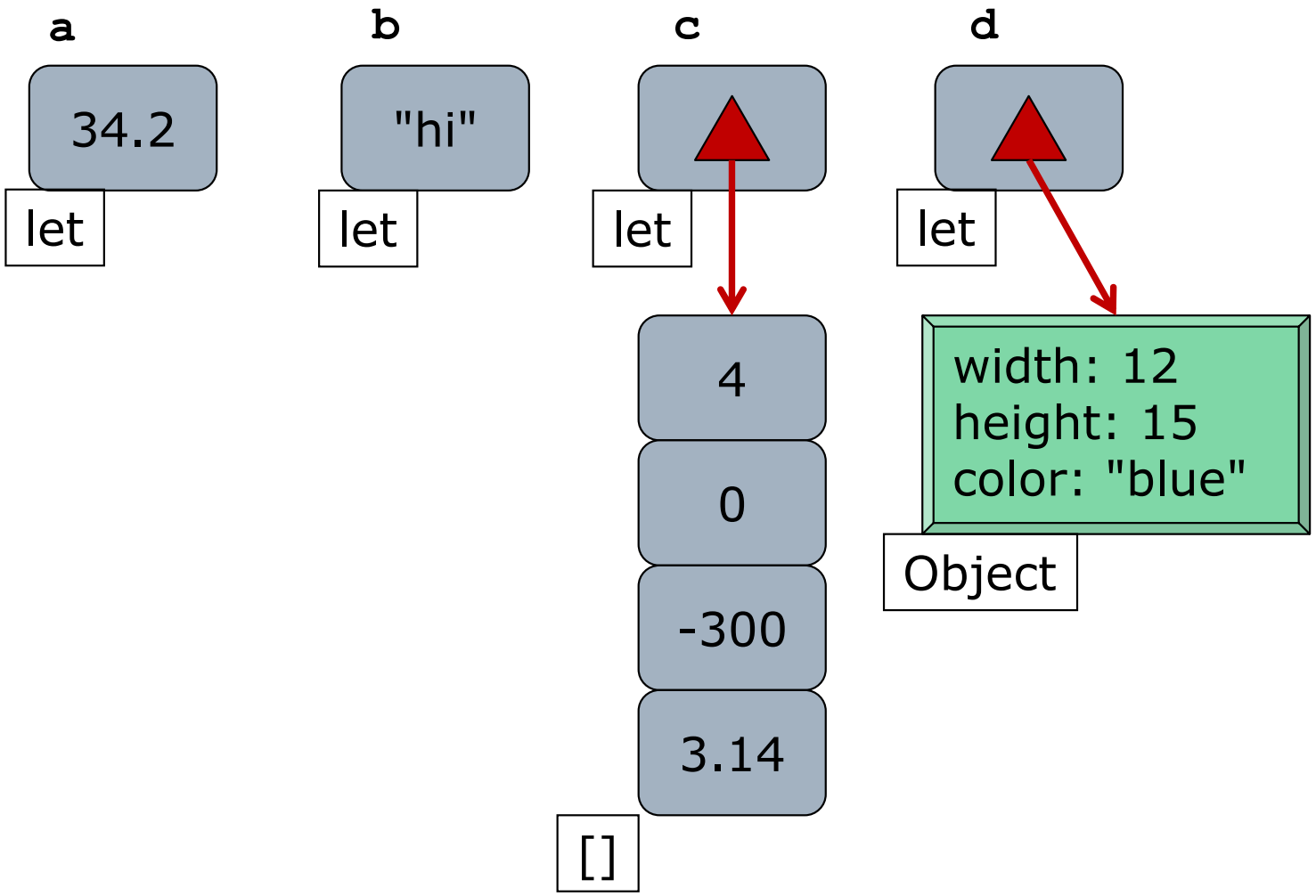
- Dynamic: known only at run time
  - *e.g.*, Python, PHP, Ruby, JavaScript

```
let x
let a
let t
let d
```

Static Types



Dynamic Types



Function Signatures

- Statically typed

```
String parse(char[] s, int i) {… return e;}
out = parse(t, x);
```

  - Parameter types (*i.e.* s and i) are declared
  - Return type (*i.e.* of parse) is declared
  - The *compiler* checks conformance of
    - (Declared) types of arguments (t, x)
    - (Declared) type of return expression (e)
    - (Declared) type of expression *using* parse (out)
- Dynamically typed

```
function parse(s, i) { … }
out = parse(t, x)
```

  - You are on your own!

Changing Types at Run-time

Static Types

```
  //a is undefined
String a;
  //a is null string
a = "hi;
  //compile-time err
a = "hi";
a = 3;
  //compile-time err
a.push();
  //compile-time err
```

Dynamic Types

```
  //a is undeclared
let a;
  //a is undefined
a = "hi;
  //load-time error
a = "hi";
a = 3;
  //a is a number
a.push();
  //run-time error
```

Resources

- MDN (Mozilla Developer Network)
  - developer.mozilla.org/docs/JavaScript
- codepen.io, jsfiddle.net
  - HTML, CSS, Javascript → result
- REPL
  - In VM, at console:

```
$ node
```

  - In a browser: repl.it/languages/javascript
    - Update: free repl.it plan no longer useful
- Class web site (under Resources)
  - Style guides (Airbnb, Google)
  - Books, available online
    - *JavaScript: The Definitive Guide* (Flanagan)
    - *Eloquent JavaScript* (Haverbeke)

Summary

- Executes at client-side, in browser
  - Interpreted (not compiled)
- Basic syntax: operators, statements
- Objects: document, window…
- Types
  - Primitives: boolean, number, string, null, undefined
  - References: arrays, objects (& functions)
- Working with primitives and references
  - Checking equality
  - Assignment
  - Parameter passing
- Dynamic types (vs static types)