CIS 371 Web Application Programming

TypeScript I

Transition from Java to TypeScript

Lecturer: Dr. Yong Zhuang

Based on the original version by Professor Hans Dulimarta.

Official Reference:
The TypeScript Handbook

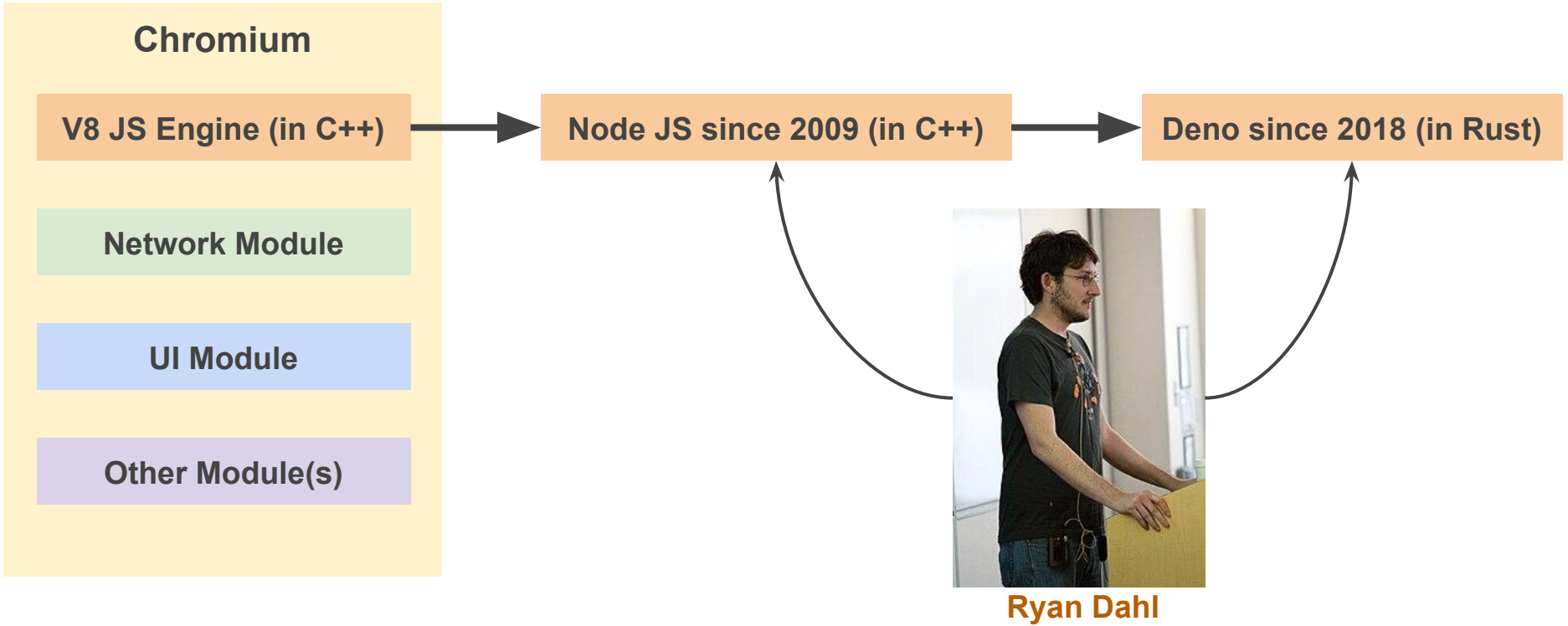
JavaScript and TypeScript

JS Edition	Release Date	Code Name	TypeScript Version
1st-4th	June 1997, Jun 1998, Dec 1999		
5th	June 2011		TypeScript 0.x (2012-2013)
6th	June 2015	ECMAScript 6 or ES2015	
7th	June 2016	ECMAScript 2016	TypeScript 2.0
8th	June 2017	ECMAScript 2017	
9th	June 2018	ECMAScript 2018	TypeScript 3.0
10th	June 2019	ECMAScript 2019	
11th	June 2020	ECMAScript 2020	TypeScript 4.0

Prerequisites

- Software Required
  - NodeJS
    - node: for running JavaScript in a non-browser environment
    - npm (Node Package Manager): for installing JS/TS libraries
  - TypeScript
    - ts-node
    - tsc (TypeScript transpiler to JavaScript)

Node JS & Deno

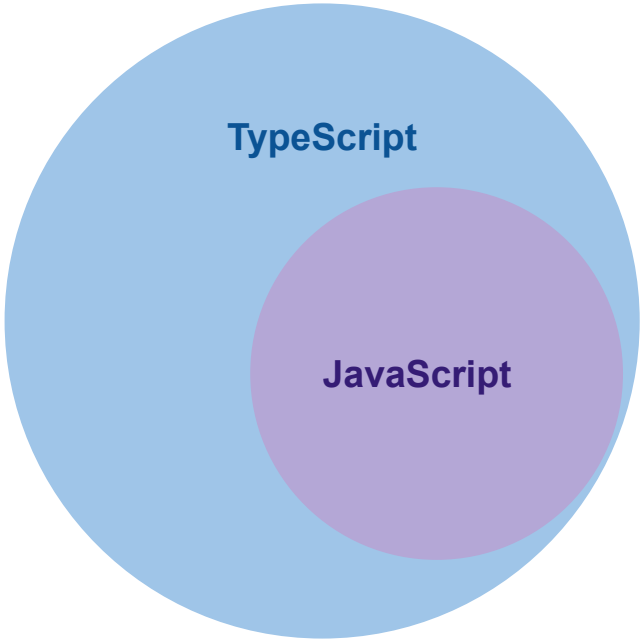

Chromium
V8 JS Engine (in C++)
Network Module
UI Module
Other Module(s)
Node JS since 2009 (in C++)
Deno since 2018 (in Rust)
Ryan Dahl

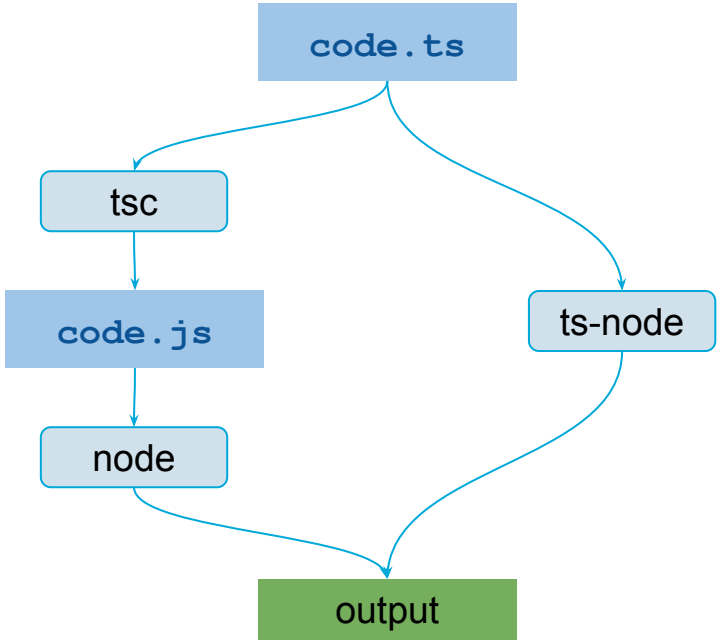
Initial Setup For Node JS

- Using Docker
  - Please refer to the Docker slides for detailed steps on setting up NodeJS within a Docker container.
- Direct Installation
  - Download NodeJS (from https://nodejs.org)
    - Choose the LTS (Long Term Support) version.
- Verify Your Installation
    - Once installed, open a terminal, command prompt, console, or PowerShell.

```
node -v  # version 14.x.x (or newer)
npm -v   # version 7.x.x (or newer)
npx -v   # version 7.x.x (or newer)
```

TypeScript vs. JavaScript





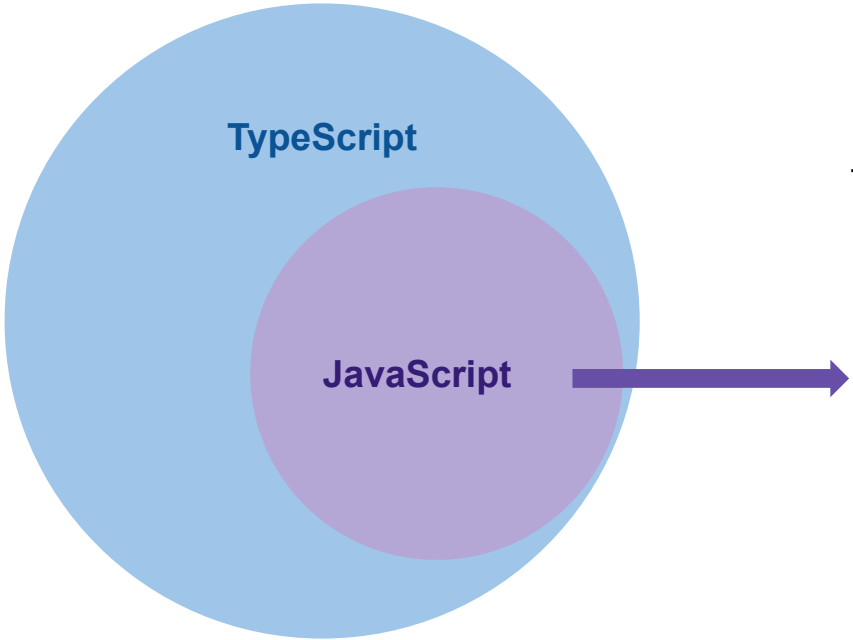
TypeScript adds syntax on top of JavaScript, allowing developers to add types.

TypeScript: Online PlayGround

https://typescriptlang.org/play

https://replit.com/languages/typescript

JavaScript: Online Reference



Use Mozilla Developer Network (MDN Web Docs) to lookup JS objects/classes:

- *Array,*
- *BigInt,*
- *Map,*
- *Number,*
- *String,*
- *Promise,*
- *etc…*

Initialize a NodeJS (and TypeScript) Project

```
# Create a new (sub) directory
mkdir my-first-node-project
cd my-first-node-project

npm init -y                     # Creates package.json
npm install -D typescript       # Add typescript as development dependencies
npm install -D ts-node npx
tsc -init                       # Creates tsconfig.json

# Create hello.ts
…

# Run option 1: Use ts-node
npx ts-node hello.ts
# Run option 2: Use tsc and node
npx tsc hello.ts
node hello.js
```

package.json

```
{
  "name": "sample-project",
  "version": "1.0.0",
  "author": "name <name.org>",
  "licence": "MIT",
  "dependencies": {
    "@js-joda/core": "3.0.1",
    "axios": "0.25.3",
    . . .
  },
  "devDependencies": {
    "ts-node": "x.y.z",
    "Nodemon": "x.y.z"
  }
}
```

Libraries needed to run/deploy your app

Libraries needed only during development of your app

Hello World: Java vs. TypeScript

Demo.java

```
class Demo {
  public sayHello() {
    System.out.println("Hello, JS");
  }
}
```

hello.ts

```
console.log("Hello World");
console.error("Hello Again");
```

Say.java

```
class Say {
  public static void main(String[] argos) {
    System.out.println("Hello World!");
    System.err.println("Hello again");
    Demo d = new Demo();
    d.sayHello();
  }
}
```

say.ts

```
class Demo {
  sayHello(): void {
    console.log("Hello, TS");
  }
}

const d:Demo = new Demo();
d.sayHello();
```

Functions vs. Methods

function-demo.ts

```
// Stand-alone fn
function sayHello(): void {
  console.log("Hello, TS");
}
```

method-demo.ts

```
class Demo {
  // a method of a class
  sayHello(): void {
    console.log("Hello, TS");
  }
}
```

- **Use *function*** keyword for standalone functions
- **No *function*** keyword methods in a class

Data Types

Java	TypeScript
boolean	boolean
char	string
String	string
float, double	number
short, int, long	number
	any (no type checking)
	unknown (strict type checking)

Variable Declaration

Java	TypeScript
boolean isHidden	let isHidden: boolean;
boolean isHidden = false;	let isHidden = false;
final String name = “Tom”;	const name = “Tom”
float taxRate;	let taxRate: number;
short distance;	let distance: number;

- Use let for mutable variables
- Use const for immutable “variables”

Explicit type is not required when the compiler can infer the type from the surrounding context

Variable Declarations (uninitialized)

```
// Java (inside a class)
boolean isDarkMode;       // init to false
String lang;              // init to null
float total;              // init to 0.0f
```

```
// TypeScript (anywhere)
let isDarkMode: boolean; // init to undefined
let lang: string;        // init to undefined
let total: number;       // init to undefined
```

null is different from undefined

TS Unions: multiple types

```
// TypeScript
let a: number | boolean;                    // init to undefined
let b: string | number | null = 6.5;        // current type is number

a = "can't do this";                        // Error, can't take a string type
a = false;                                  // current type is boolean

console.log(typeof b);                      // output "number"

b = "hello";
console.log(typeof b);                      // output "string"
```

Use this feature in conjunction with typeof test at runtime

Type Assertions (or Typecast)

```
function doOne (x: number | string | null): void {
  console.debug(x.toUpperCase()); // Compile ERROR: toUpperCase() does not exist for number
  console.debug(x * 10);          // Compile ERROR: incorrect type for arithmetic
}
```

```
function doOne (x: number | string | null): void {
  console.debug((x as string).toUpperCase());
}
```

```
doOne("five"); // Output FIVE
doOne(5); // Runtime crash!
```

```
function doOne (x: number | string | null): void {
  console.debug((x as number) * 3);
}
```

```
doOne("five"); // Runtime crash!
doOne(5); // Output 15
```

SmartCast

```
function doOne (x: number | string | null): void {
  if (typeof x === 'number') console.debug(x * 3);
  else if (typeof x === 'string') console.debug(x.toUpperCase());
}
```

== vs ===

==		===	
5 == “5”	true	5 === “5”	false
0.123 == “0.123”	true	0.123 === “0.123”	false
1 == true	true	1 === true	false
5 == true	false	5 === true	false
0 == false	true	0 === false	false
“0” == false	true	“0” === false	false
“1” == true	true	“1” === true	false
With internal type conversion		***No type conversion***	

Arrays

```
// Create with initial values and capacity
const primes: number[] = [31, 43, 19];
const alsoPrimes: Array<number> = [31, 43, 19];

for (let k = 0; k < primes.length; k++) {
  console.debug("At", k, primes[k]);
}

// At 0 31
// At 1 43
// At 2 19
```

```
// Initialize with capacity
const values: number[] = new Array(5);
const nums = new Array<number>(7);

console.log(values.length);        // output 5
console.log(nums.length);          // output 7

console.debug(values[0]);          // Output "undefined"
```

Arrays: for, for-in vs. for-of

```
const fruits = ["Apple", "Banana", "Cherry"];
```

```
for (let k = 0; k < fruits.length; k++) {
  console.debug("At", k, fruits[k]);
}
```

```
for (let k in fruits) {
  console.debug("At", k, fruits[k]);
}
```

for-in

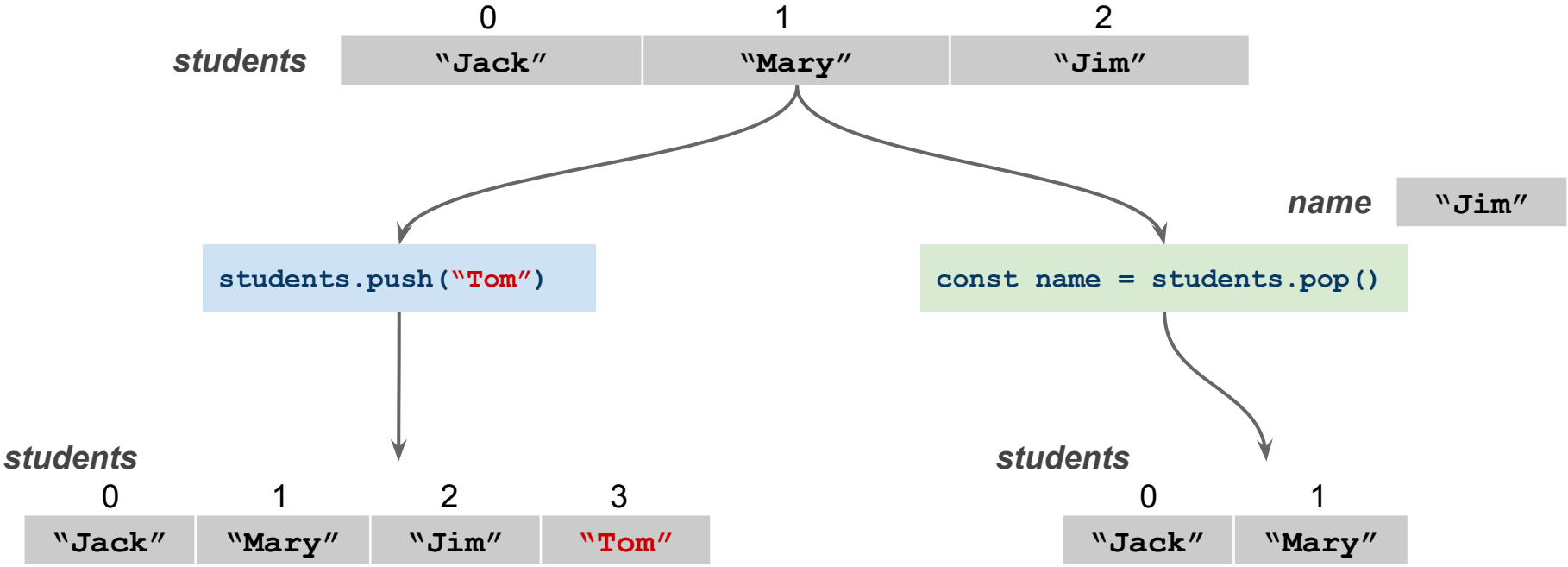
```
At 0 Apple
At 1 Banana
At 2 Cherry
```

```
for (let f of fruits) {
  console.debug(f);
}
```

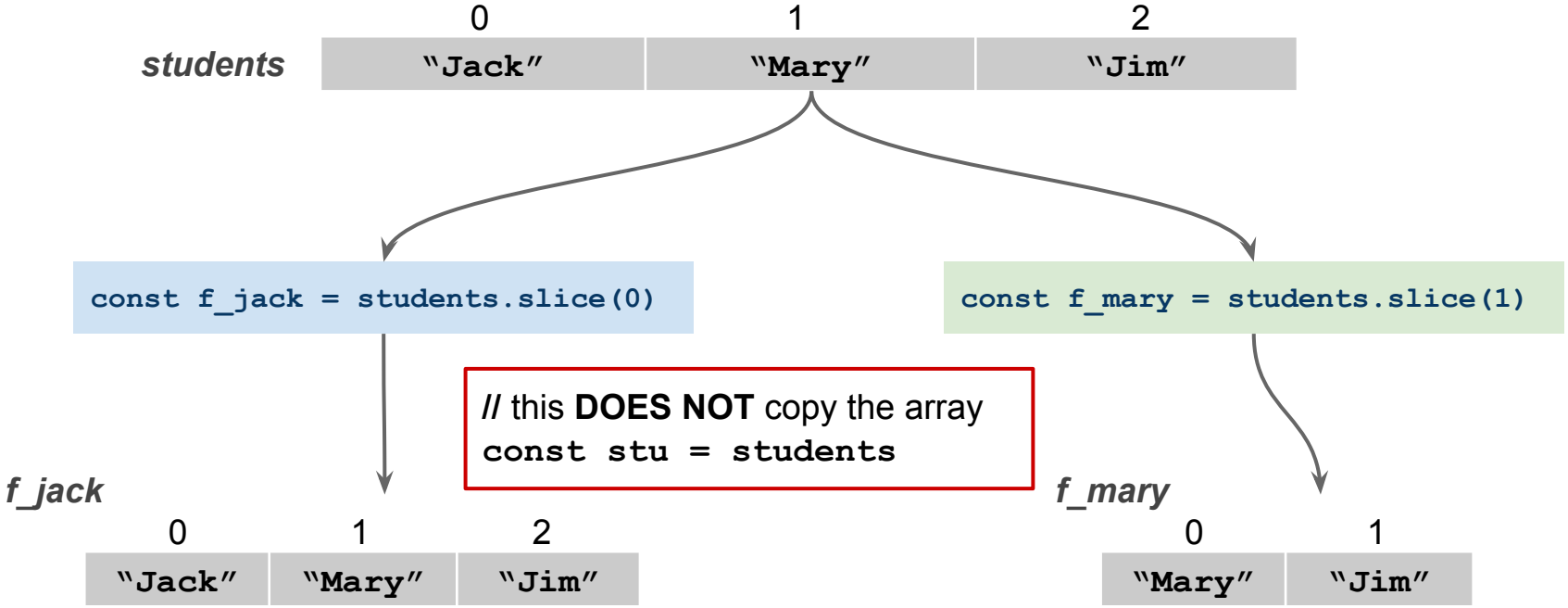
for-of

```
Apple
Banana
Cherry
```

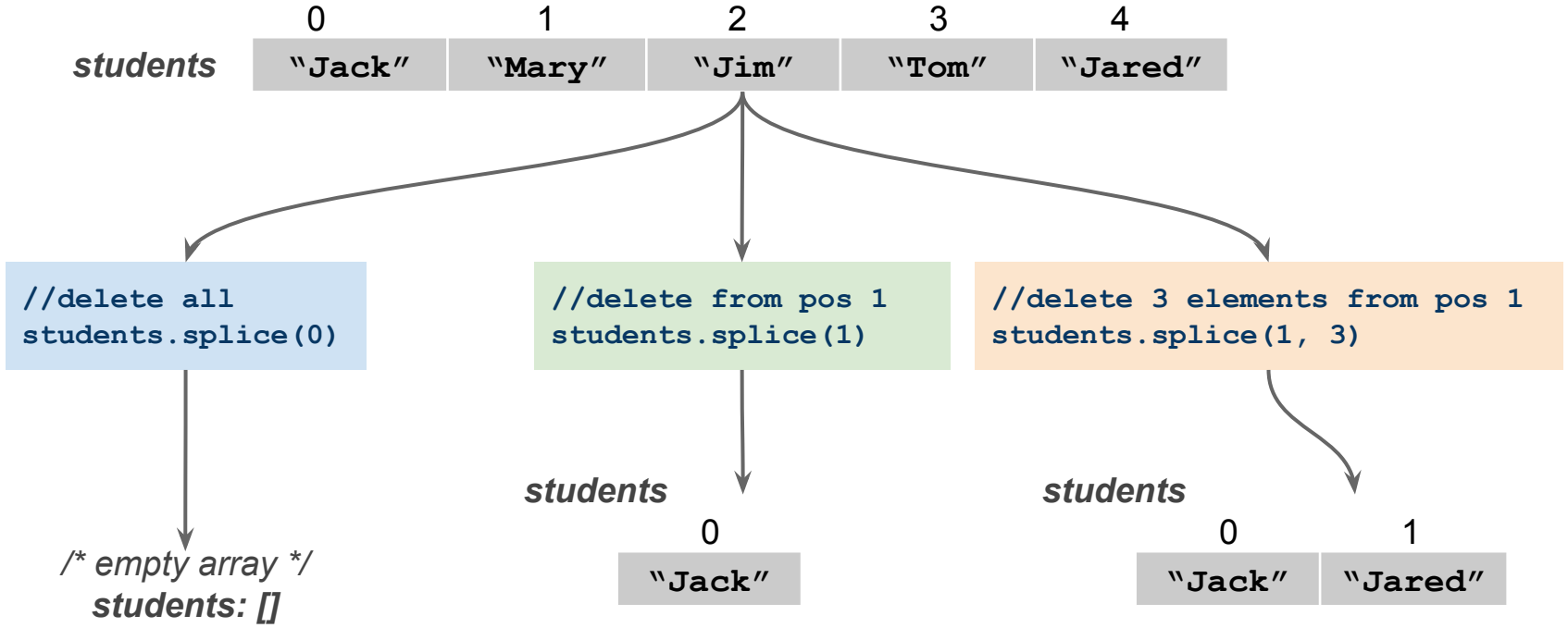
Array: .push() and .pop()



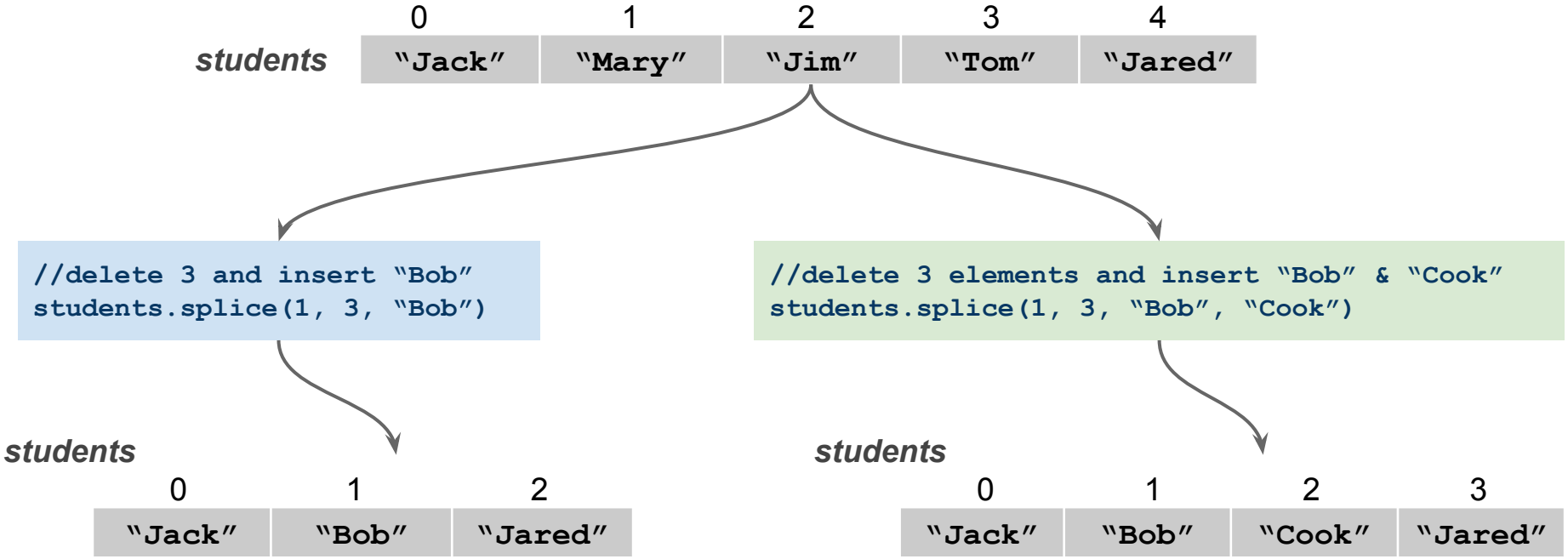
Array: .slice() creates a copy



Array: .splice() delete elements



Array: .splice() replaces elements



Objects

Java Classes and Objects

```
// Java objects must be instantiated from a class
// In Sub.java
class Sub() {
  public String name;
  public int calorie;
}

// In AnotherFile.java
Sub my_order = new Sub();
my_order.name = "Spicy Turkey";
my_order.calorie = 182;

my_order.price = 3.17;  // ERROR!
```

vs.

TS Objects

```
// TypeScript (no class needed)
const my_order = {
  name: "Spicy Turkey",
  calorie: 182
}
```

Objects can be created without a class definition

Objects in TypeScript

```
// Typeless objects
const in_a_month = {
  name: "September",
  days: 30
}

const employee_vacation = {
  name: "Bob",  days: 11
}
```

```
// Typed objects
type Monthly = {
  name: string,
  days: number
}
const in_a_month:Monthly {
  name: "September",
  days: 30
}
```

```
type VacationDays = {
  name: string,
  days: number
}
const employee_vacation:VacationDays = {
  name: "Bob",
  days: 11
}
```

Objects with Sub-Objects & Array property

```
type City = {
  name: string,
  population: number,
  geopos: {
    lat: number,
    lon: number
  },
  univs: Array<string>
}
```

```
const ours:City = {
  name: "Grand Rapids",
  population: 198400,
  geopos: {
    lat: 42.9633599,
    lon: -85.6680863
  },
  univs:  [
    "Calvin", "Cornerstone",
    "GVSU"
  ]
}
```

```
const theirs:City = {
  name: "East Lansing",
  population: 48729,
  geopos: {
    lat: 42.737652,
    lon: -84.483788
  },
  univs:  [
    "MSU",
  ]
}
```

```
console.log(ours.name);
for (let u of ours.univs) console.log(u);

console.log(theirs.geopos.lat);
```

```
Grand Rapids
Calvin
Cornerstone
GVSU
42.737652
```

for-in to enumerate object properties

```
const theirs:City = {
  name: "East Lansing",
  population: 48729,
  geopos: {
    lat: 42.737652,
    lon: -84.483788
  },
  univs:  [
    "MSU",
  ]
}
```

```
for (let z in theirs) {
  console.debug(z)
}
```

```
name
population
geopos
univs
```

```
for (let z in theirs) {
  console.debug(z, theirs[z]);
 }                    ^-------^ ERROR
```

```
const eLan = theirs as any;
for (let z in theirs) {
  console.debug(z, "==>", eLan[z])
}
```

```
name ==> East Lansing
population ==> 48729
geopos ==> {lat: 42..., lon: -84..}
univs == > ["MSU"]
```

Array of Objects

```
// In Atom.java
class Atom {
  public String name;
  public weight double;
}

// In AnotherFile.java
ArrayList<Atom> atoms = new ArrayList<>();
Atom a = new Atom("Carbon", 12);
atoms.add(a);
Atom b = new Atom("Oxygen", 16);
atoms.add(b);
atoms.add(new Atom("Natrium", 23);
```

TS: option 1

```
// TypeScript (no class required)
 const atoms = [];
 atoms.push({ name: "Carbon", weight: 12});
 atoms.push({ name: "Oxygen", weight: 16});
 atoms.push({ name: "Natrium", weight: 23});
```

TS: option 2

```
// Or initialize the array
const atoms = [
  { name: "Carbon", weight: 12},
  { name: "Oxygen", weight: 16},
  { name: "Natrium", weight: 23}
];
```

Array of Typed Objects

Typeless array

```
const atoms = [];
atoms.push({ name: "Carbon", weight: 12});
atoms.push({ namme: "Fluor", weight: 12}); // OK
atoms.push({ name: "Oxygen"}); // OK
atoms.push({ name: "Natrium", weight: 23, isMetal: false}); // OK
```

```
// Declare a type
type Atom = {
  name: string,
  weight: number
}
```

Typed array

```
const atoms:Array<Atom> = [];
 atoms.push({ name: "Carbon", weight: 12});
 atoms.push({ namme: "Fluor", weight: 12});        // ERROR: "namme" does not exist
 atoms.push({ name: "Oxygen"});                    // ERROR: property "weight" is missing
 atoms.push({
   name: "Natrium",
   weight: 23,
   isMetal: false});                               // ERROR: "isMetal" does not exist
```