

CIS 658 Web Architectures

TypeScript I

Transition from Java to TypeScript



GRAND VALLEY
STATE UNIVERSITY®

Lecturer: **Dr. Yong Zhuang**

Official Reference:
The TypeScript Handbook

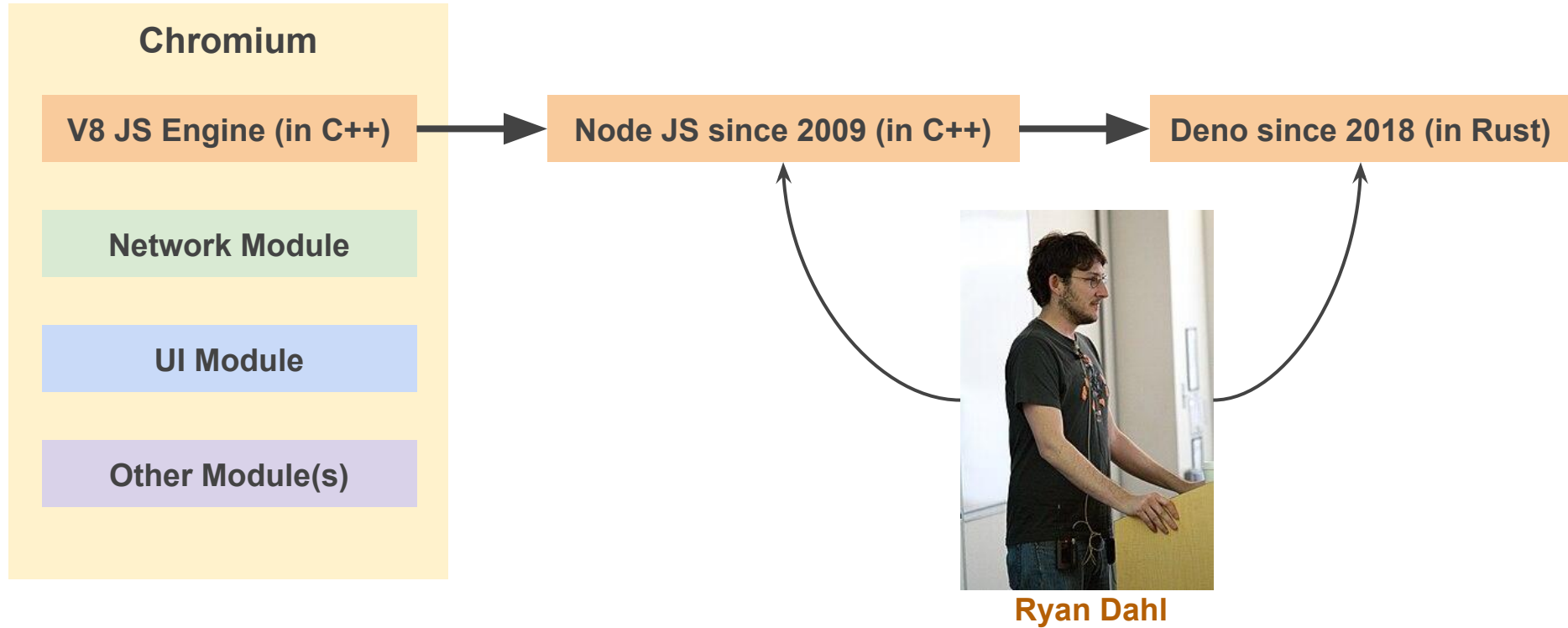
JavaScript and TypeScript

JS Edition	Release Date	Code Name	TypeScript Version
1st–4th	June 1997, Jun 1998, Dec 1999	–	–
5th	June 2011	ECMAScript 5	TypeScript 0.x (2012–2013)
6th	June 2015	ECMAScript 6 (ES2015)	TypeScript 1.x
7th	June 2016	ECMAScript 2016 (ES2016)	TypeScript 2.0
8th	June 2017	ECMAScript 2017 (ES2017)	TypeScript 2.3–2.6
9th	June 2018	ECMAScript 2018 (ES2018)	TypeScript 3.0
10th	June 2019	ECMAScript 2019 (ES2019)	TypeScript 3.x
11th	June 2020	ECMAScript 2020 (ES2020)	TypeScript 4.0
12th	June 2021	ECMAScript 2021 (ES2021)	TypeScript 4.3–4.5
13th	June 2022	ECMAScript 2022 (ES2022)	TypeScript 4.7
14th	June 2023	ECMAScript 2023 (ES2023)	TypeScript 5.0
15th	June 2024	ECMAScript 2024 (ES2024)	TypeScript 5.4+

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



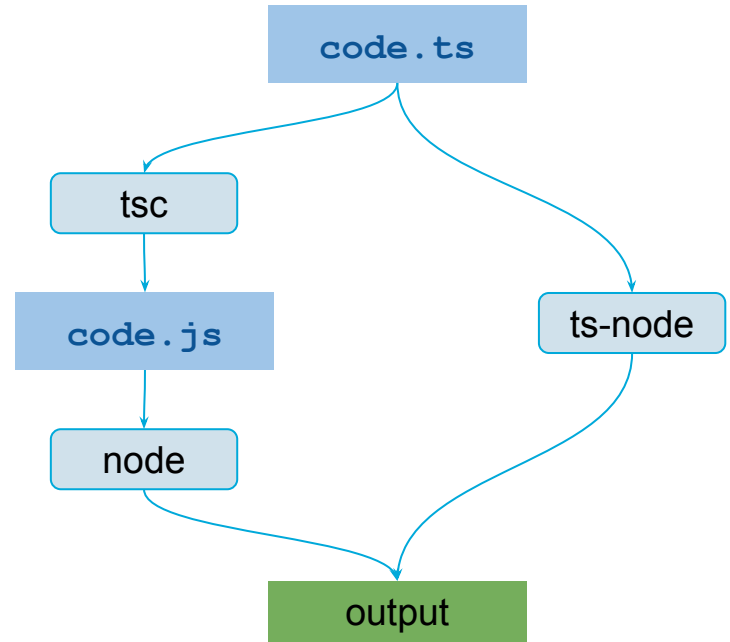
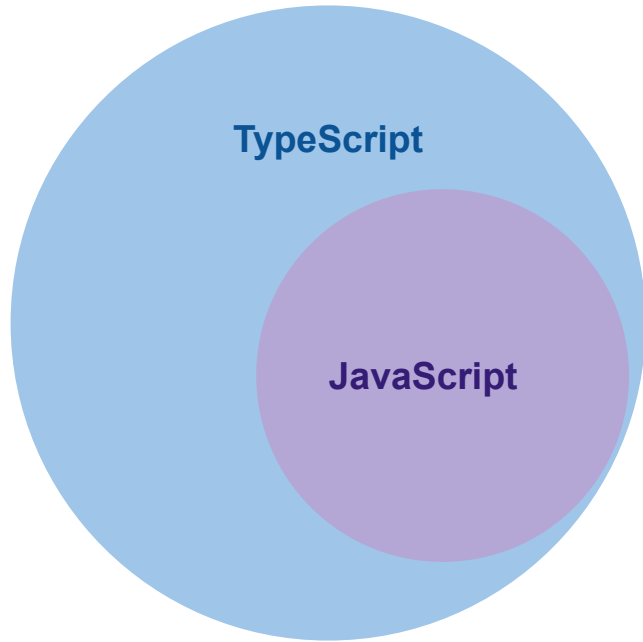
Ryan Dahl

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: Benefits

TypeScript is a super set of JavaScript that provides static typing. With static typing we reap a number of benefits including:

- Tighter integration with IDE
- Better Refactoring with Errors at Compile Time
- And Self Documented Code that's Easier to Understand

JS: Dynamic typing vs. TS: Static typing

Dynamic typing

```
let age = 30;
age = 'thirty';

const car = { wheels: 4, doors: 4 };

car.isElectric = false;
delete car.doors;
```

```
let age: number = 30;
age = 'thirty';
// Error: Type 'string' is not assignable to type 'number'
```

```
const car = { wheels: 4, doors: 4 };
```

```
car.isElectric = false;
```

```
// Error: Property 'isElectric' does not exist on type '{ wheels: number; doors: number; }'
```

```
delete car.doors; // Error: 'doors' is not deletable
```

Static typing

Tighter integration with IDE

```
interface User {  
  id: number;  
  name: string;  
  email: string;  
}  
  
function getUser(id: number): User {  
  return {  
    id,  
    name: 'Alice Smith',  
    email: 'alice.smith@example.com',  
  };  
}  
  
const user = getUser(1);  
console.log(user);
```

- email
- id
- name

Better Refactoring with Errors at Compile Time

```
interface User {
  id: number;
  name: string;
  email: string;
}

function getUser(id: number): User {
  return {
    id,
    name: 'Alice Smith',
    email: 'alice.smith@example.com'.
  };
}

const user2: User = {
  id: 1,
  name: 'Alice Smith',
  email: 'alice.smith@example.com',
  age: 12;
};
```

Property 'age' does not exist on type 'User'. ts(2339)

any

View Problem (Alt+F8) Quick Fix... (Ctrl+.)

Self-Documented Code

```
type Currency = 'USD' | 'EUR' | 'GBP';
```

```
interface Product {  
  name: string;  
  price: number;  
  currency: Currency;  
}
```

```
function displayProduct(product: Product): string {  
  return `${product.name} costs ${product.price} ${product.currency}`;  
}
```

```
const product: Product = {  
  name: 'Wireless Mouse',  
  price: 29.99,  
  currency: 'USD',  
};
```

```
console.log(displayProduct(product));
```

currency
name
price

(property) Product.name: string

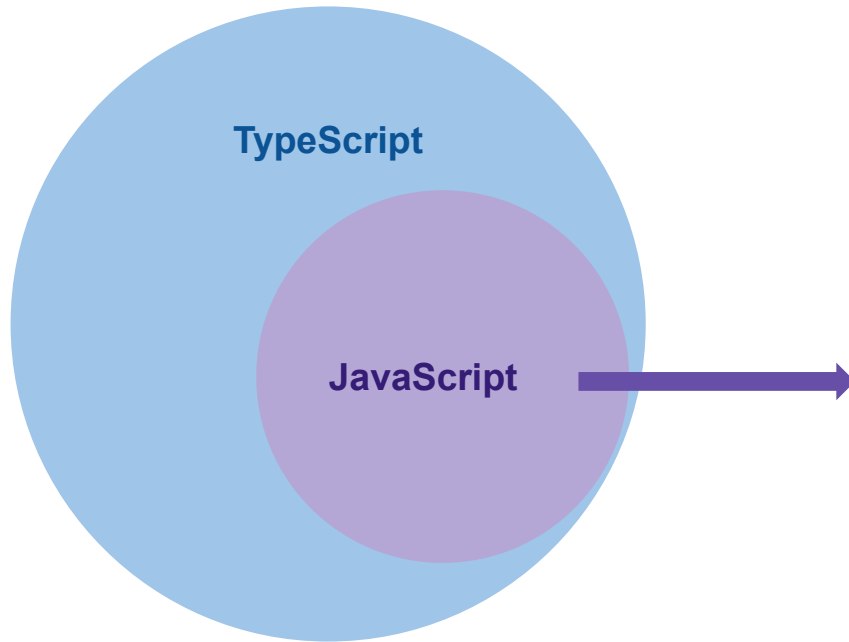
×

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

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

Demo.java

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

hello.ts

```
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.java

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

say.ts

Functions vs. Methods

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

function-demo.ts

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

method-demo.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
<code>boolean isHidden</code>	<code>let isHidden: boolean;</code>
<code>boolean isHidden = false;</code>	<code>let isHidden = false;</code>
<code>final String name = "Tom";</code>	<code>const name = "Tom";</code>
<code>float taxRate;</code>	<code>let taxRate: number;</code>
<code>short distance;</code>	<code>let distance: number;</code>

- 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
```

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

SmartCast

== 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
<i>With internal type conversion</i>		<i>No type conversion</i>	

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

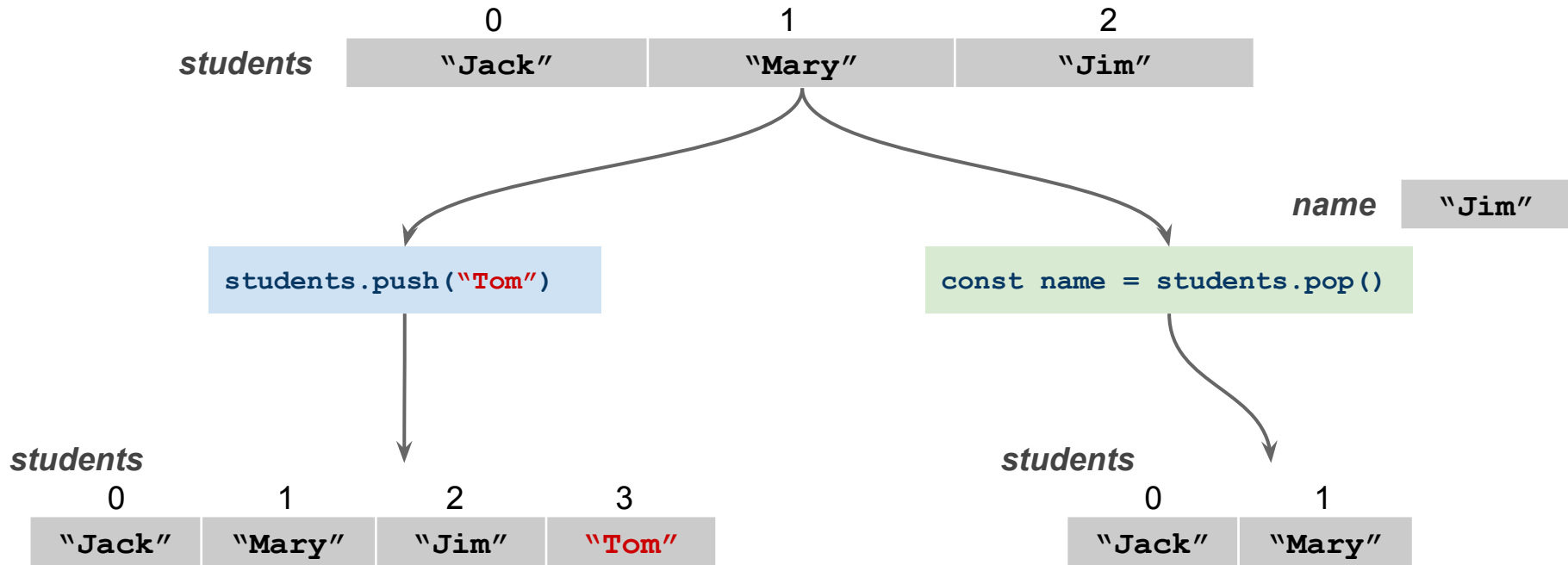
```
for (let f of fruits) {  
  console.debug(f);  
}
```

for-of

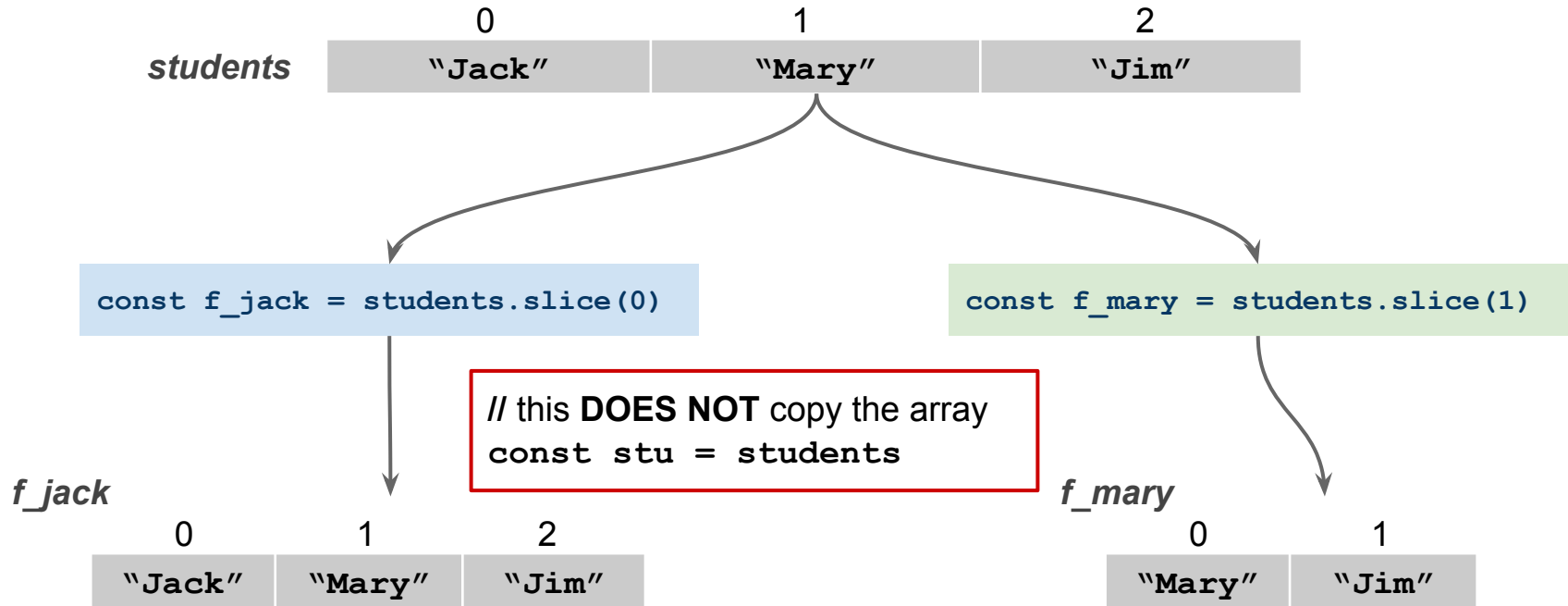
At 0 Apple
At 1 Banana
At 2 Cherry

Apple
Banana
Cherry

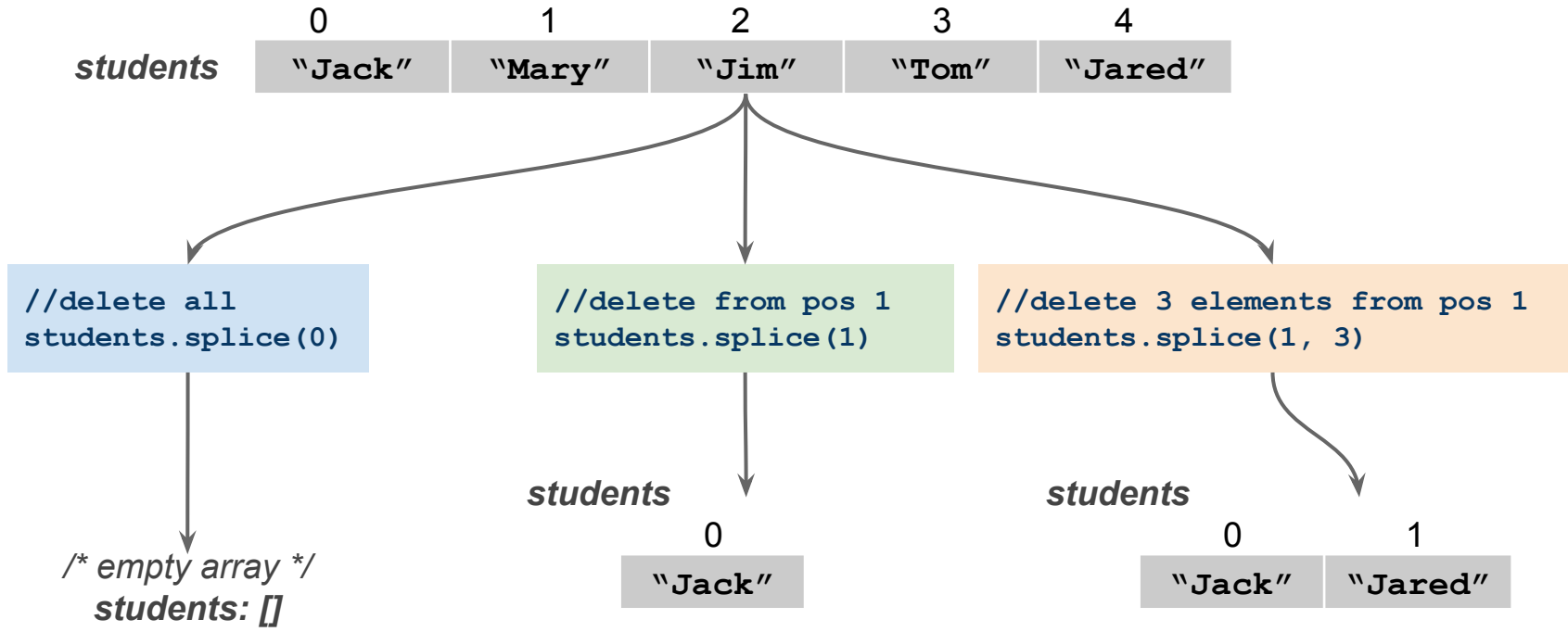
Array: `.push()` and `.pop()`



Array: `.slice()` creates a copy



Array: `.splice()` delete elements



Array: `.splice()` replaces elements

students

0	1	2	3	4
"Jack"	"Mary"	"Jim"	"Tom"	"Jared"

```
//delete 3 and insert "Bob"  
students.splice(1, 3, "Bob")
```

students

0	1	2
"Jack"	"Bob"	"Jared"

```
//delete 3 elements and insert "Bob" & "Cook"  
students.splice(1, 3, "Bob", "Cook")
```

students

0	1	2	3
"Jack"	"Bob"	"Cook"	"Jared"