Programming Methodology

Lab 4: Testing and Mental Models

Wednesday February 25, 2026

Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises.
  - Please sit with your group from last week.
- Discussion leaders will walk around and answer questions.
- Solutions to exercises will be reviewed as a class.
- Attendance will be taken at the end.

Reminders

- Midterm 1 is next Wednesday (3/4) from 7-9 pm.
  - Previous exams can be found under “Modules” on Canvas.
- Homework 4 is due tonight (2/25) at 11:59pm.
  - Come to office hours for help!
- Homework 5 will be released after the midterm.
- If you need to miss lab and have a valid reason according to the syllabus (medical, other personal) please fill out the lab excusal form on Canvas before the start time of your lab.
  - Waking up late or the bus being late are NOT valid reasons to miss lab.

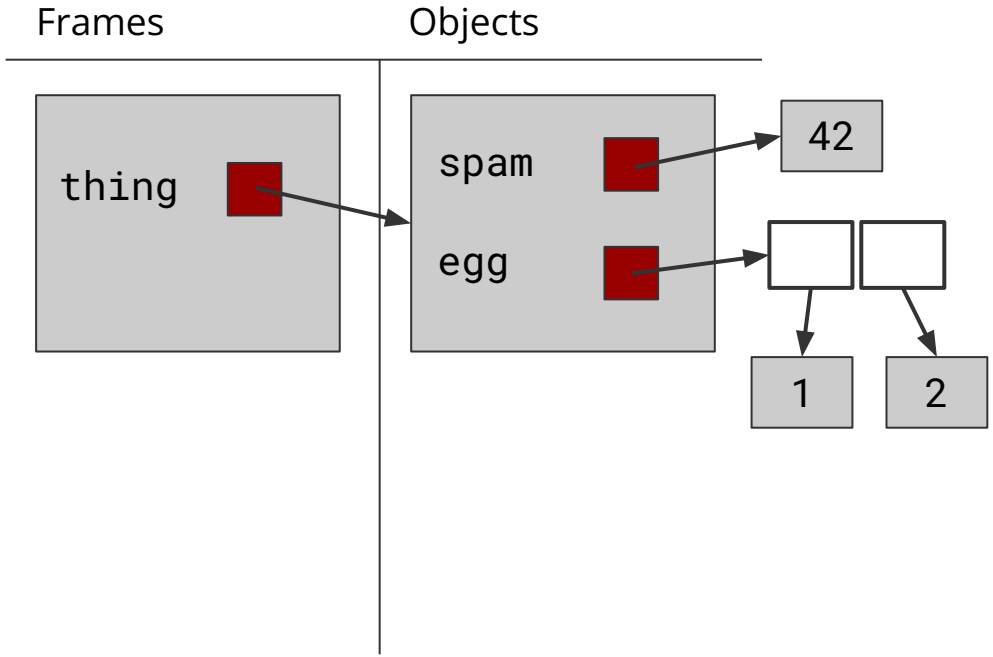
Today's Goals

- Practice creating mental models
- Practice test design and using Jest

Exercise 1: Mental Models

The memory diagram for the following code is depicted on the right. It uses reference semantics for all types.

```
1 let thing = {
2     spam: 42,
3     egg: [1, 2]
4 };
```



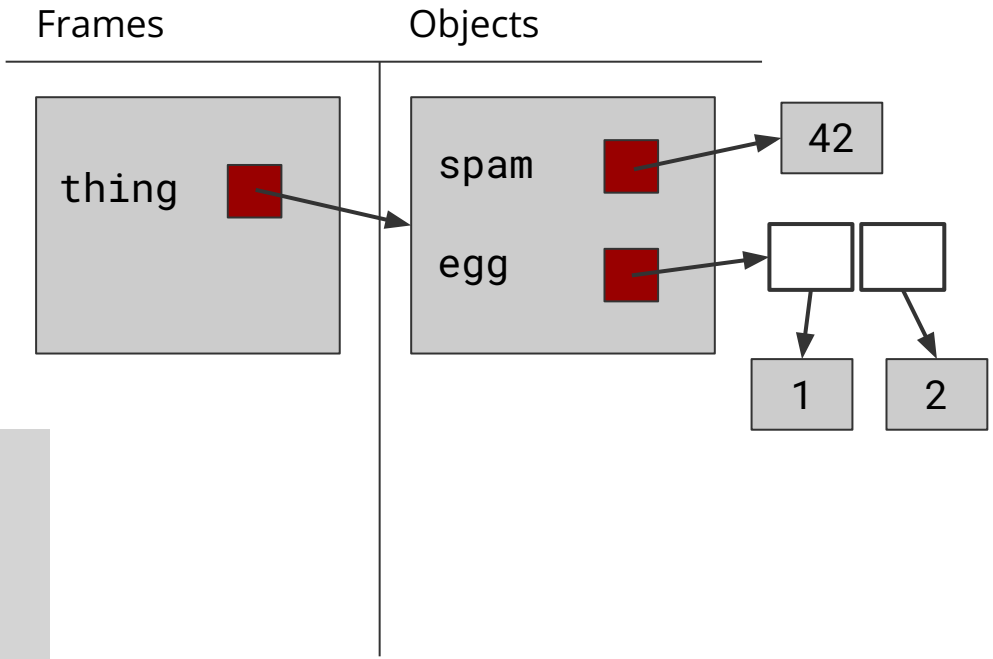
Exercise 1: Mental Models

```
1 let thing = {
2     spam: 42,
3     egg: [1, 2]
4 };
```

Draw the updated memory diagram after running lines 5-10 below.

Note: "a += b" is equivalent to "a = a + b" in TypeScript.

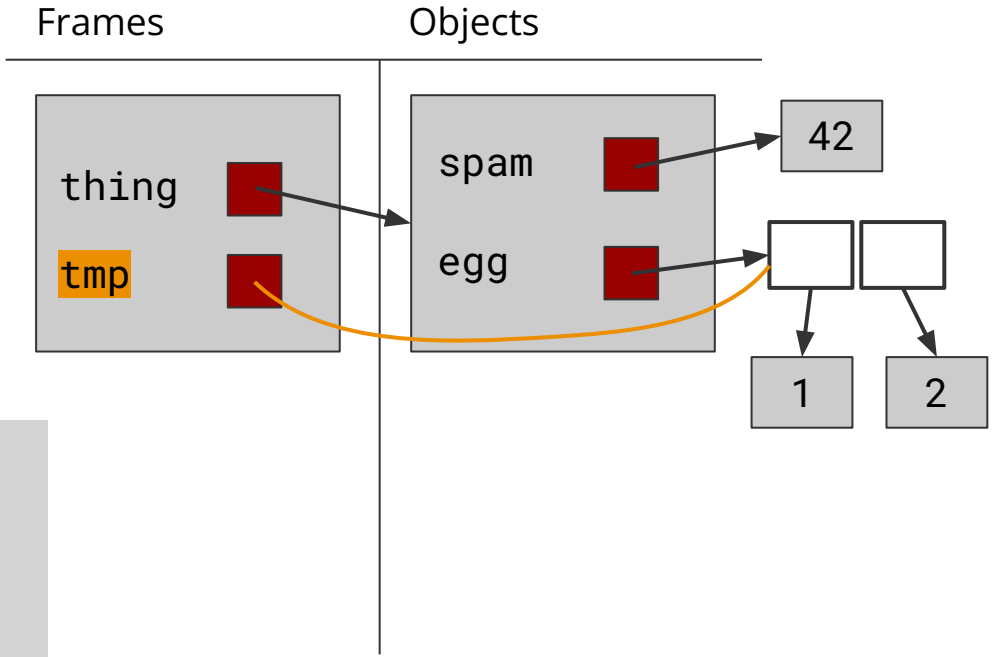
```
5  let tmp = thing.egg;
6  thing.egg[0] += 3;
7  thing.egg = thing.spam;
8  thing.spam = tmp;
9  thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```



Exercise 1: Solution

Line 5:

- variable `tmp` is initialized
- the value stored in `thing.egg` (reference to array `[1,2]`) is stored in `tmp`

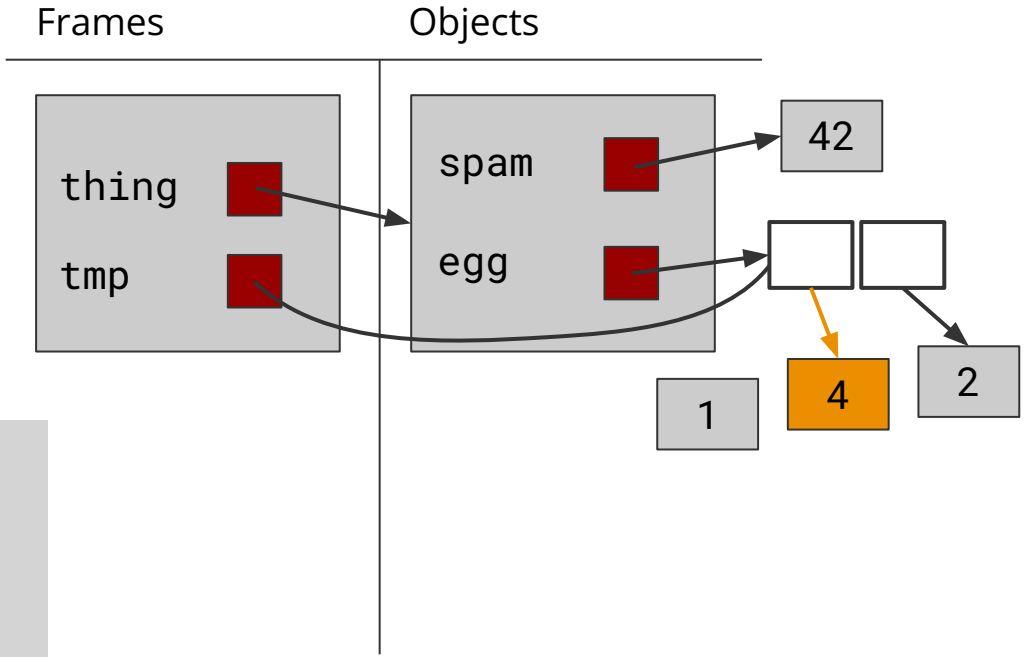



```
5  let tmp = thing.egg;
6  thing.egg[0] += 3;
7  thing.egg = thing.spam;
8  thing.spam = tmp;
9  thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```

Exercise 1: Solution

Line 6:

- the element at index `0` in the array referenced by `thing.egg` is incremented by 3

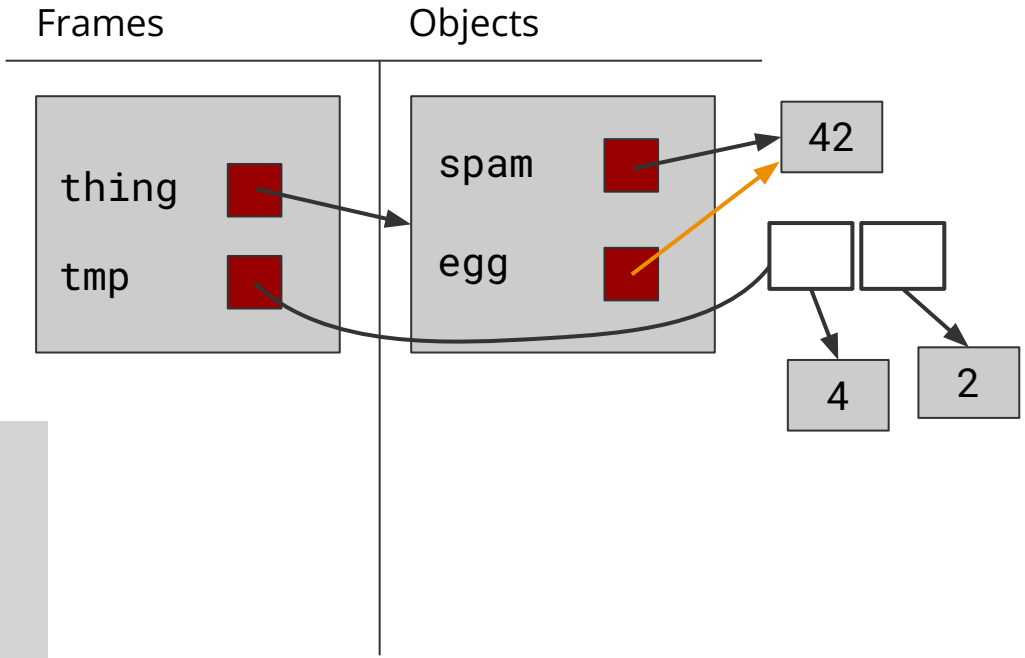



```
5  let tmp = thing.egg;
6  thing.egg[0] += 3;
7  thing.egg = thing.spam;
8  thing.spam = tmp;
9  thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```

Exercise 1: Solution

Line 7:

- the `egg` property of object `thing` is set to the value stored in `thing.spam`, which is the number 42

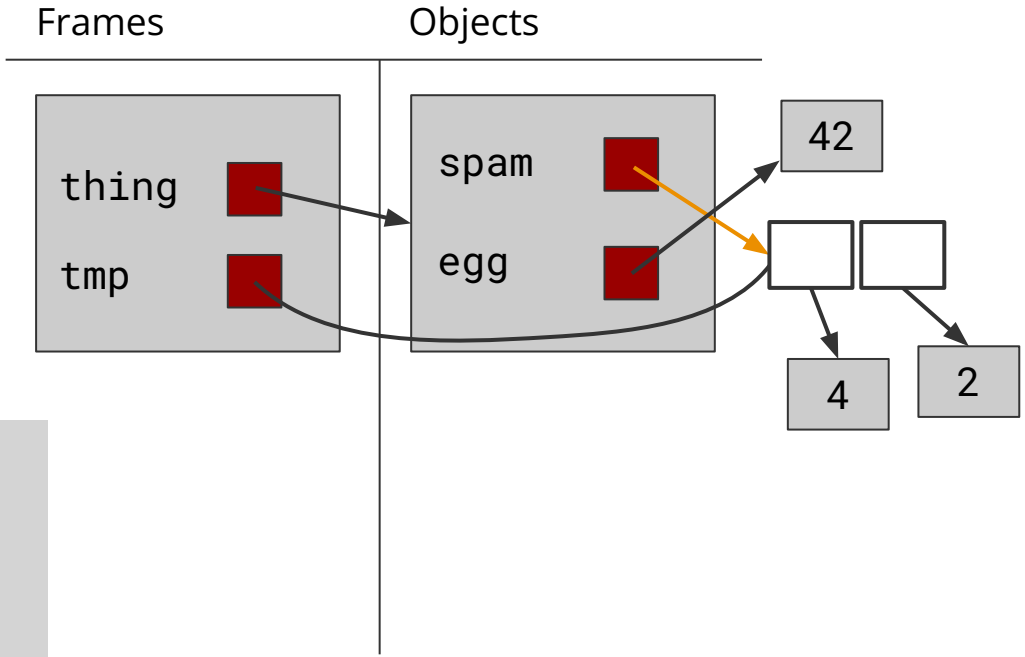



```
5  let tmp = thing.egg;
6  thing.egg[0] += 3;
7  thing.egg = thing.spam;
8  thing.spam = tmp;
9  thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```

Exercise 1: Solution

Line 8:

- the `spam` property of object `thing` is set to the value stored in `tmp`, which is the reference to the array `[4,2]`



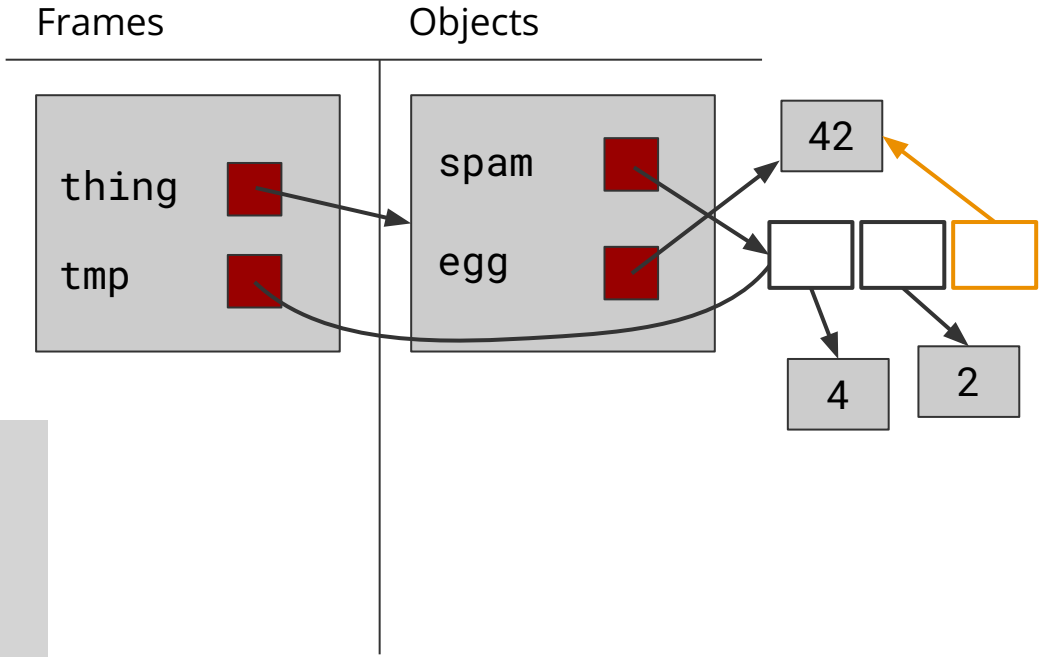
Note: lines 7 and 8 would error in Typescript!

```
5  let tmp = thing.egg;
6  thing.egg[0] += 3;
7  thing.egg = thing.spam;
8  thing.spam = tmp;
9  thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```

Exercise 1: Solution

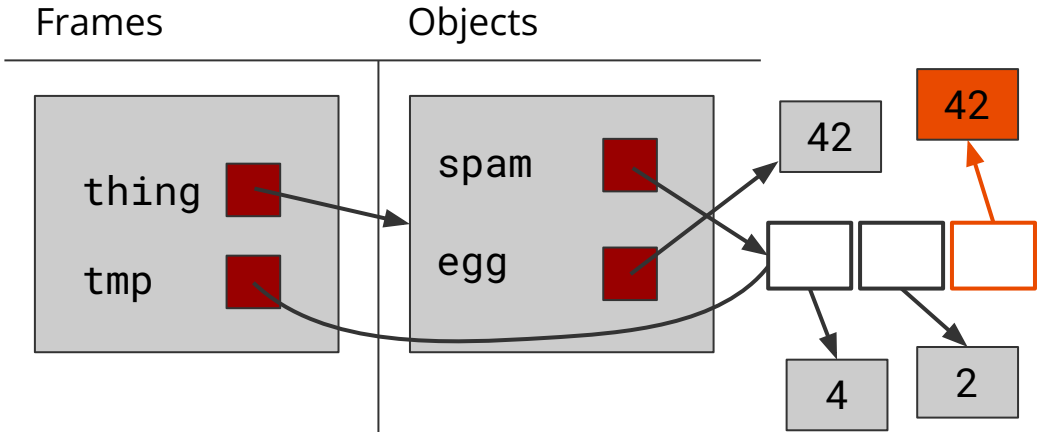
Line 9:

- the value stored in `thing.egg` (42) is added as an element to the array referenced by `thing.spam`




```
5  let tmp = thing.egg;
6  thing.egg[0] += 3;
7  thing.egg = thing.spam;
8  thing.spam = tmp;
9  thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```

Exercise 1: Solution




```
5  let tmp = thing.egg;
6  thing.egg[0] += 3;
7  thing.egg = thing.spam;
8  thing.spam = tmp;
9  thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```

Why is the diagram on the left **incorrect**? Why is it **incorrect** for `thing.spam[2]` to point to a different 42?

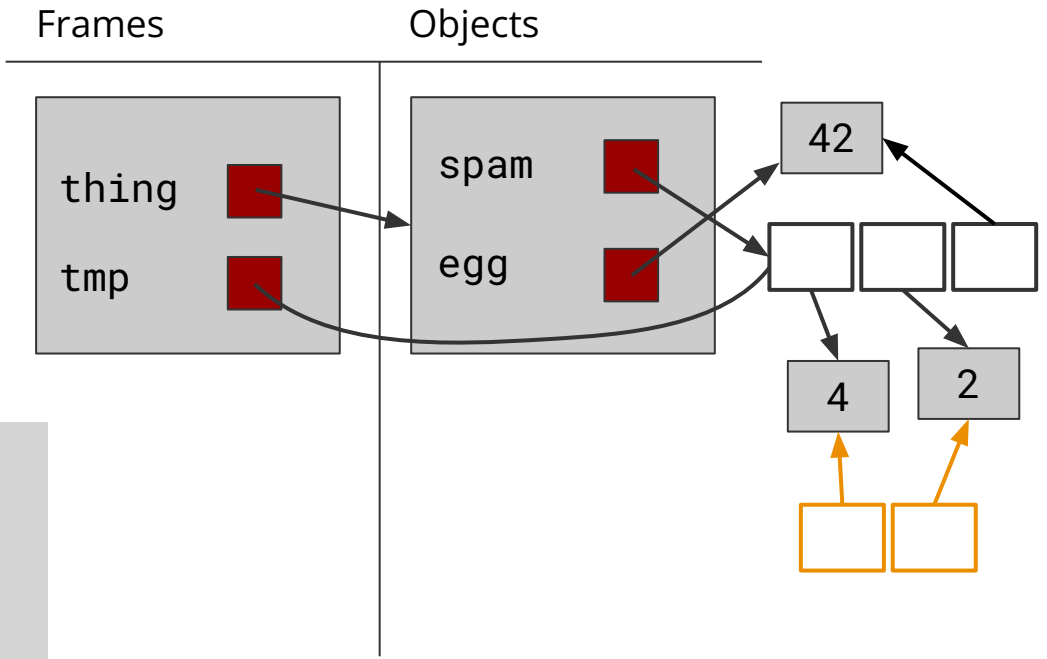
`thing.spam.push(thing.egg)` means we are adding a new element to the array referenced by `thing.spam`, where this element points to the same value as `thing.egg`. So, it should point to the **same** 42 as `thing.egg`.

Exercise 1: Solution

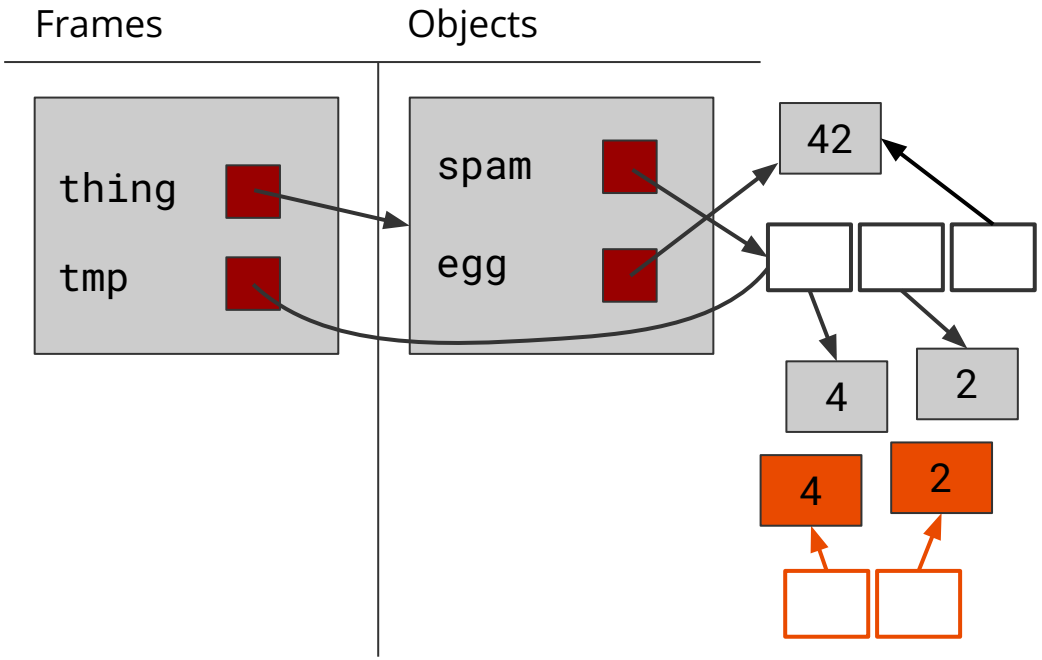
Line 10:

- `filter` is called on the array referenced by `thing.spam` (`[4,2,42]`)
- a new array is created, with only the elements less than 5 (`[4,2]`)
  - the reference to this array is never stored

```
5  let tmp = thing.egg;
6  thing.egg[0] += 3;
7  thing.egg = thing.spam;
8  thing.spam = tmp;
9  thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```



Exercise 1: Solution

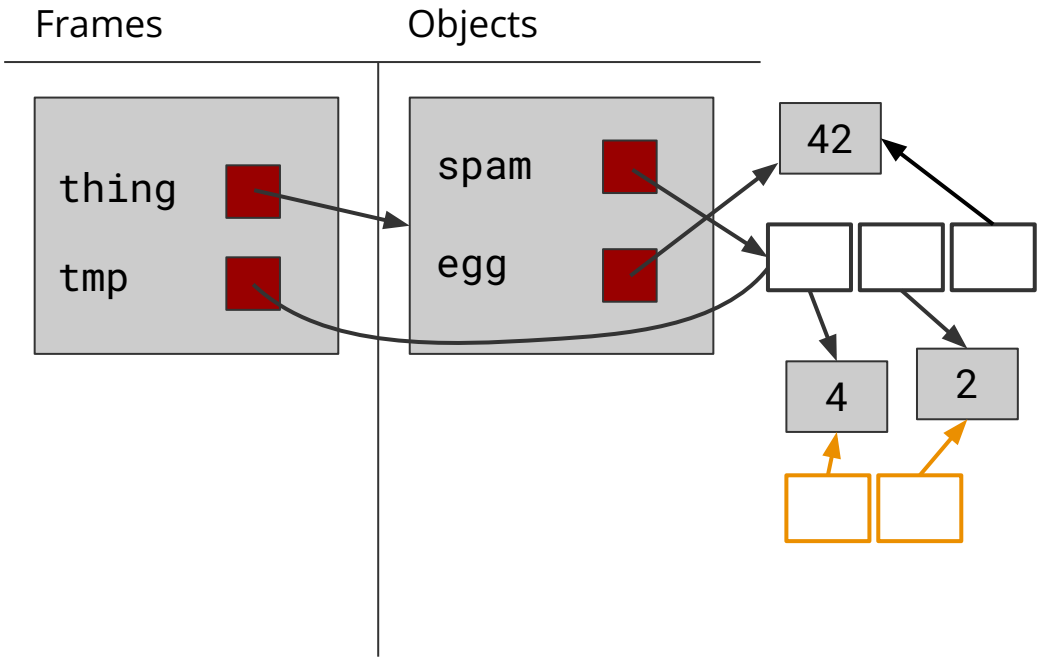


Why is the diagram on the left **incorrect**? Why is it **incorrect** for the elements of the new array to point to different numbers than those pointed to by the existing array?

With the way filter works, the newly created array will still point to the same numbers as the original array.

```
5  let tmp = thing.egg;
6  thing.egg[0] += 3;
7  thing.egg = thing.spam;
8  thing.spam = tmp;
9  thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```

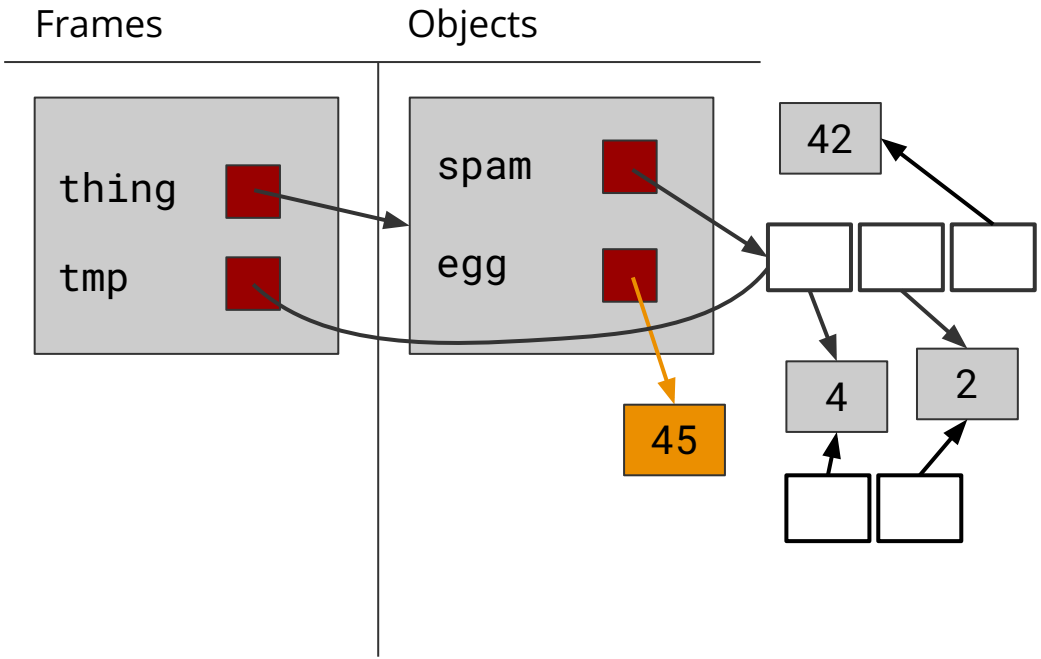
Exercise 1: Solution



What would happen if we now ran the following lines of code?

```
11 thing.egg += 3;
12 thing.spam[0] += 3;
13 tmp[1] += 3;
```

Exercise 1: Solution

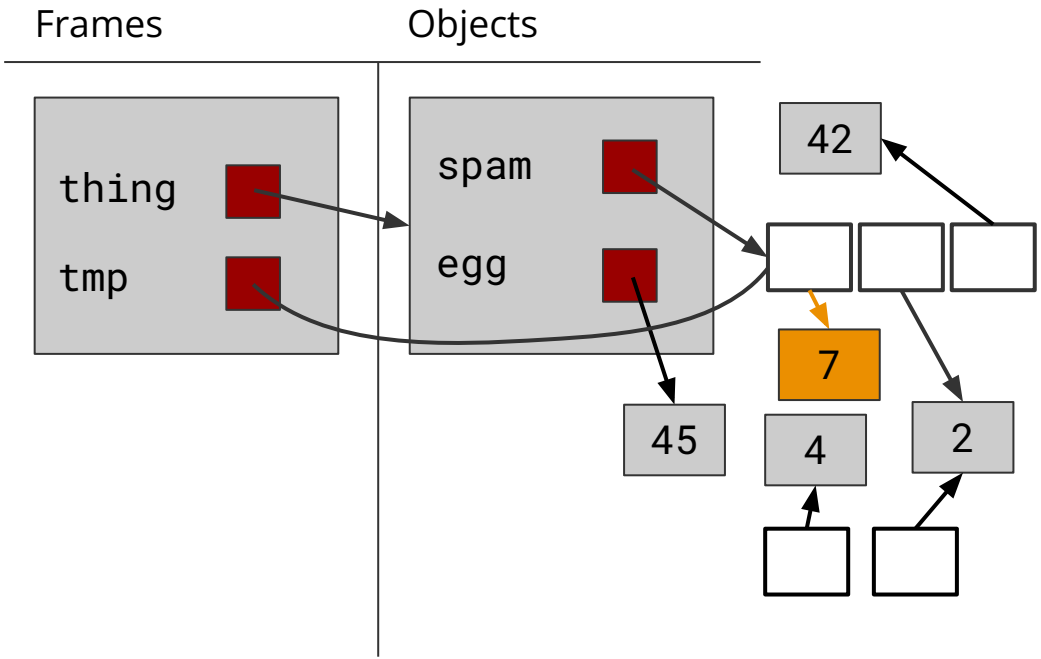


Line 11:

- we increment `thing.egg` by 3, meaning `thing.egg` now points to a different number (45)
  - this does not affect `thing.spam[2]`
  - the 42 that `thing.egg` originally pointed to is not modified to become 45, as numbers are immutable

```
11 thing.egg += 3;
12 thing.spam[0] += 3;
13 tmp[1] += 3;
```

Exercise 1: Solution

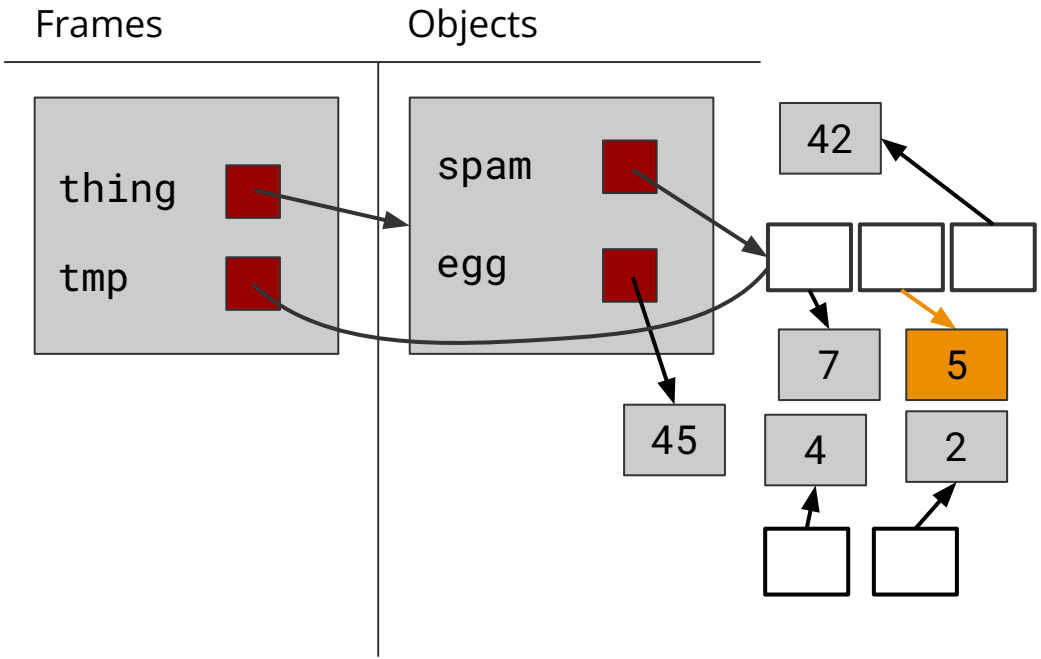


Line 12:

- we increment the first element of the array referenced by `thing.spam` by 3
- now, `thing.spam[0]` points to 7
- similar reasoning applies from the previous slide for why the original 4 is not modified

```
11 thing.egg += 3;
12 thing.spam[0] += 3;
13 tmp[1] += 3;
```

Exercise 1: Solution



Line 13:

- we increment the second element of the array referenced by `tmp` by 3
- now, `tmp[1]` points to 5
- similar reasoning applies from the previous slide for why the original 2 referenced by `tmp[1]` is not modified

```
11 thing.egg += 3;
12 thing.spam[0] += 3;
13 tmp[1] += 3;
```

Review : Jest

Jest is the **testing framework** that we use in this course.

```
// each function has a ‘describe’ block
describe(“func”, () => {
    // a ‘describe’ block can have multiple ‘it’ blocks
    // each ‘it’ block represents a test case for the function
    it(“should do something”, () => {
        ...
    });
    ...
});
```

Review : Jest

In addition to global functions, Jest provides robust testing utilities like `expect`.

```
describe("func", () => {
    it("should do something", () => {
        ...
        // expect can be chained with 'matchers'
        expect(a).toEqual(b);
        expect(f).toThrow(e);
    }); // see jestjs.io/docs/expect for more
    ...
});
```

Exercise 2: Testing with Jest

You are given a function `rotateRight<T>(arr: T[], k: number): T[]` that returns a new array containing the elements of `arr` rotated to the right by k steps.

- The function must not mutate the input array.
- k must be a non-negative integer.
- Examples:
  - `rotateRight([1,2,3,4], 1)` returns `[4,1,2,3]`
  - `rotateRight([1,2,3,4], 4)` returns `[1,2,3,4]`

Write comprehensive tests for `rotateRight` using Jest.

Exercise 2: Solution

Start with some **basic cases**.

```
it("should rotate right by 1", () => {
    expect(rotateRight([1, 2, 3, 4], 1)).toEqual([4, 1, 2, 3]);
});

it("should rotate right by 2", () => {
    expect(rotateRight([1, 2, 3, 4], 2)).toEqual([3, 4, 1, 2]);
});
```

Exercise 2: Solution

Consider when the output array is an **exact copy**.

```
it("should return a copy if k = 0", () => {
    const a = [1, 2, 3];
    const b = rotateRight(a, 0);

    expect(b).toEqual([1, 2, 3]);
    expect(b).not.toBe(a); // should be a new array
});

it("should return a copy if k equals length", () => {
    expect(rotateRight([1, 2, 3, 4], 4)).toEqual([1, 2, 3, 4]);
});
```

Exercise 2: Solution

Test **various cases for k**.

```
it("should behave like k = k % length if k > length", () => {
    expect(rotateRight([1, 2, 3, 4], 6)).toEqual([3, 4, 1, 2]);
});

it("should handle very large k values", () => {
    expect(rotateRight([1, 2, 3, 4], 100))
        .toEqual(rotateRight([1, 2, 3, 4], 100 % 4));
  });
```

Exercise 2: Solution

Consider **error cases for k**.

```
it("should throw an error if k is negative", () => {
    expect(() => rotateRight([1, 2, 3], -1)).toThrow();
});

it("should throw an error if k is a float", () => {
    expect(() => rotateRight([1, 2, 3], 1.5)).toThrow();
});

it("should throw an error if k is a negative float", () => {
    expect(() => rotateRight([1, 2, 3], -1.5)).toThrow();
});
```

Exercise 2: Solution

Test **various array lengths and types**.

```
it("should return an empty array for empty input array", () => {
    expect(rotateRight([], 10)).toEqual([]);
});

it("should handle single element arrays", () => {
    expect(rotateRight([42], 5)).toEqual([42]);
});

it("should work with non-numeric types", () => {
    expect(rotateRight(["a", "b", "c"], 1)).toEqual(["c", "a", "b"]);
});
```

Exercise 2: Solution

Finally, test that the **original array is not mutated**.

```
it("should not mutate the input array", () => {
    const a = [1, 2, 3, 4];
    rotateRight(a, 1);

    expect(a).toEqual([1, 2, 3, 4]);
});
```