Higher-order Functions Review

Lecture 22 – Spring 2019

1. Which functions are actually Funcy?

```
interface Funcy<A, B> {
    (a: B, b: A): B;
}

let a: Funcy<number, string> = (s: string, t: number): string => {
    return "?";
};

let b: Funcy<string, number> = (s: string, t: number): string => {
    return "?";
};

let c: Funcy<string, number> = (s: number, t: string): number => {
    return 1;
};

let d: Funcy<number, number> = (s: number, t: number): number => {
    return 1;
};
```

2. Of functions of b, c, d, and e, which are not exactly equivalent to function a?

```
let a: Transform<string, number> = (input: string): number => {
   return input.length;
};

let b: Transform<string, number> = (input) => {
   return input.length;
};

let c: Transform<string, number> = (input) => input.length;

let d = (input: string): number => {
   return input.length;
};

let e = (input: string): number => input.length;
```

3. What is the result of this call to `filter`?

```
filter(listify(1, 2, 3, 4), (item) => item % 2 === 1)
```

4. What is the result of this call to `map`?

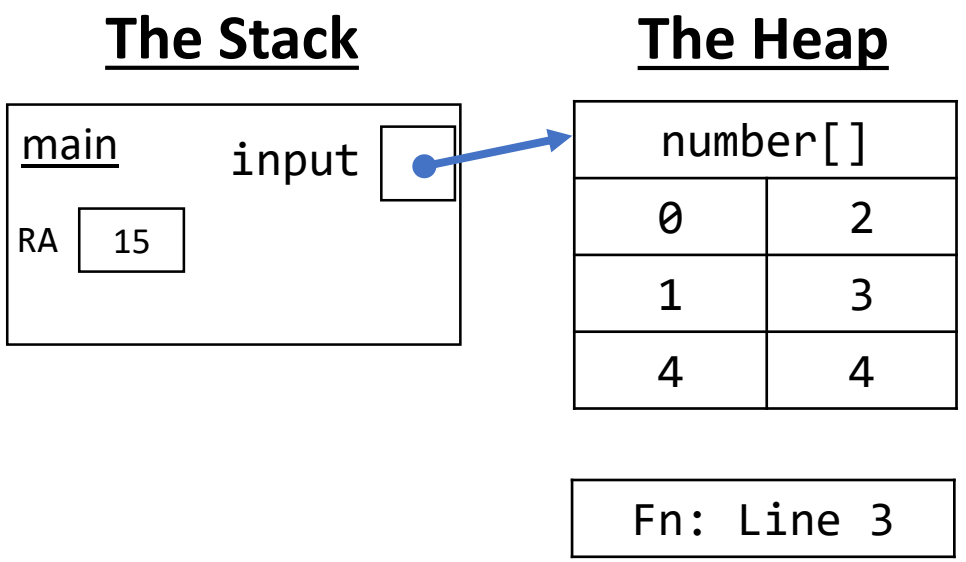
```
map(listify(1, 3), (item) => item * 2)
```

4. What is printed when the following code executes?

```
01| let main = async () => {
02|    let input = [2, 3, 4];
03|    let result = reduce(input, (m, x) => m * x, 1);
04|    print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```

Tracing higher order functions.

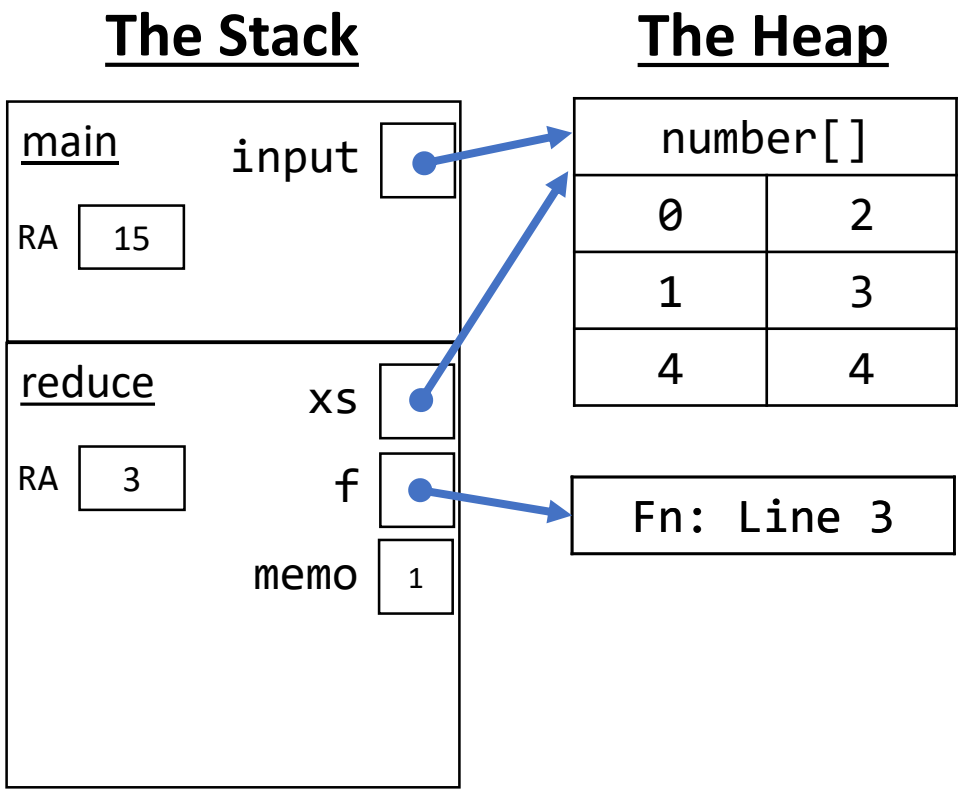
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```



- Fast-forwarding to the point of the call to *reduce*.
- Globals are omitted for space. Both *main* and *reduce* would be names bound in the Global frame.
- Notice the 2nd argument is an anonymous function. Prior to the call to reduce, it is established on the heap. It has no name bound in the main frame, though.

Passing anonymous functions.

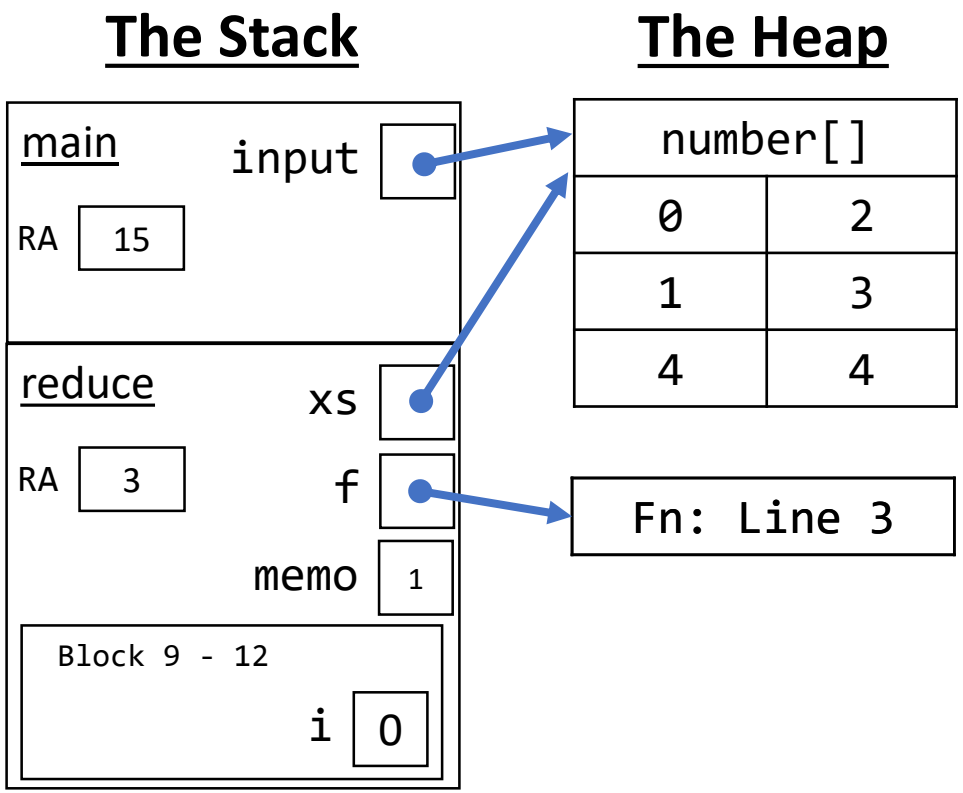
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```



- Notice in the call to reduce, the name f is bound to the function defined on line 3. This is how we'll know where to jump when the **f** function is called.

Establishing for loop block.

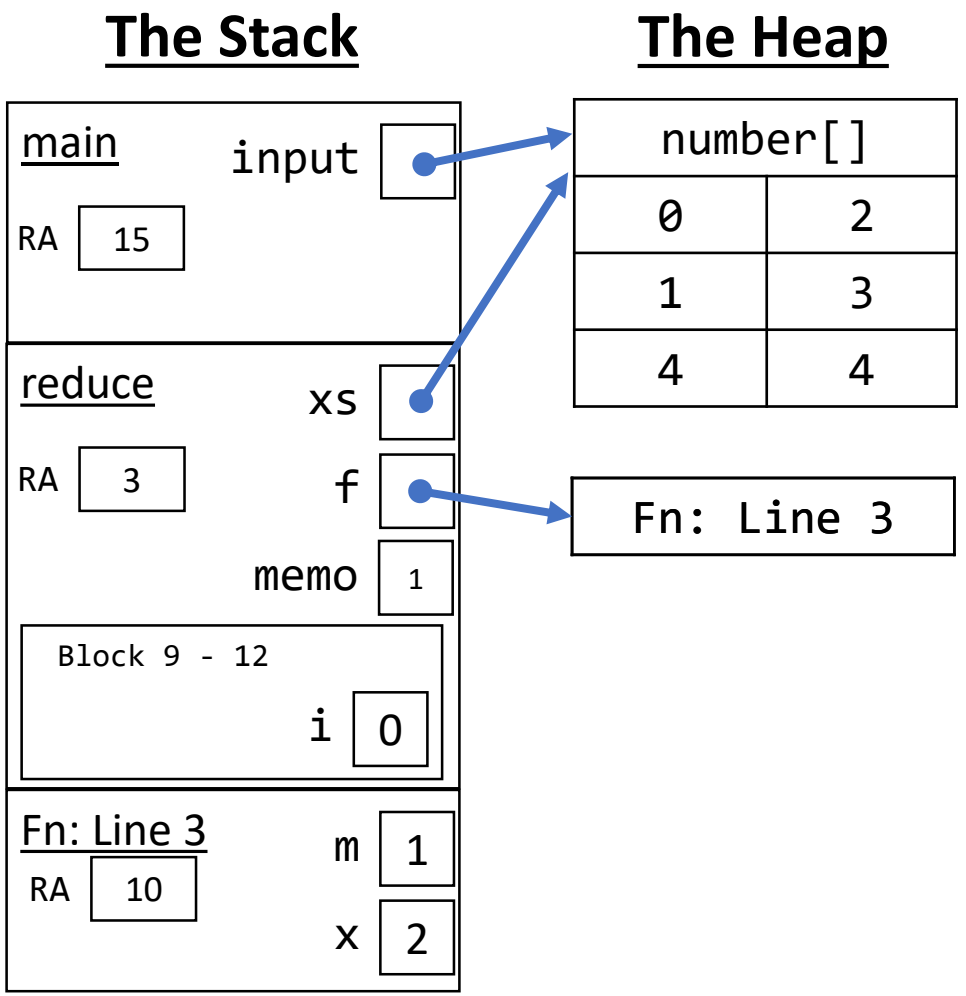
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```



- Entering the for loop, the variable i is established in the for loop's block. It is initialized to 0.

Calling the function *f*.

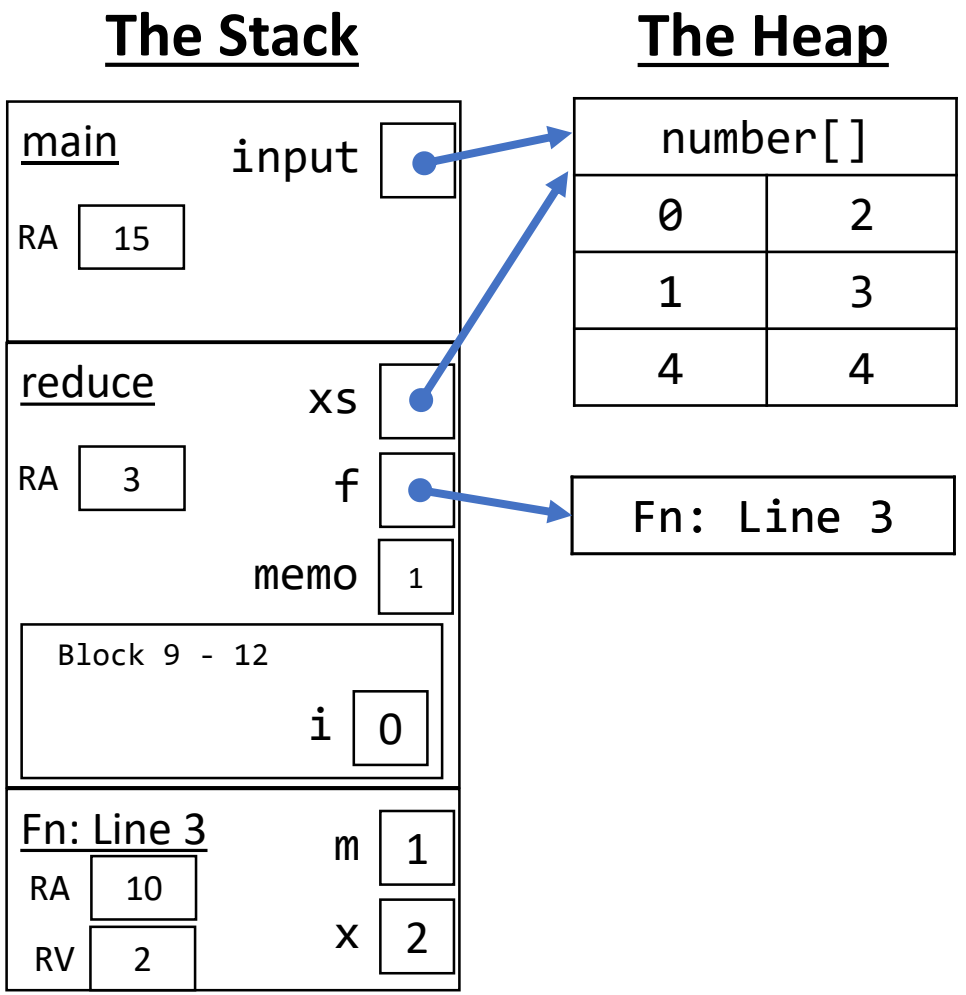
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   retu    memo;
13| };
14|
15| main();
```



- Notice here a call to the function *f* is encountered.
- We know *f* is bound to the function definition on line 3:
  `(m, x) => m * x`
- So, we establish a frame for f.

Evaluating the anonymous function.

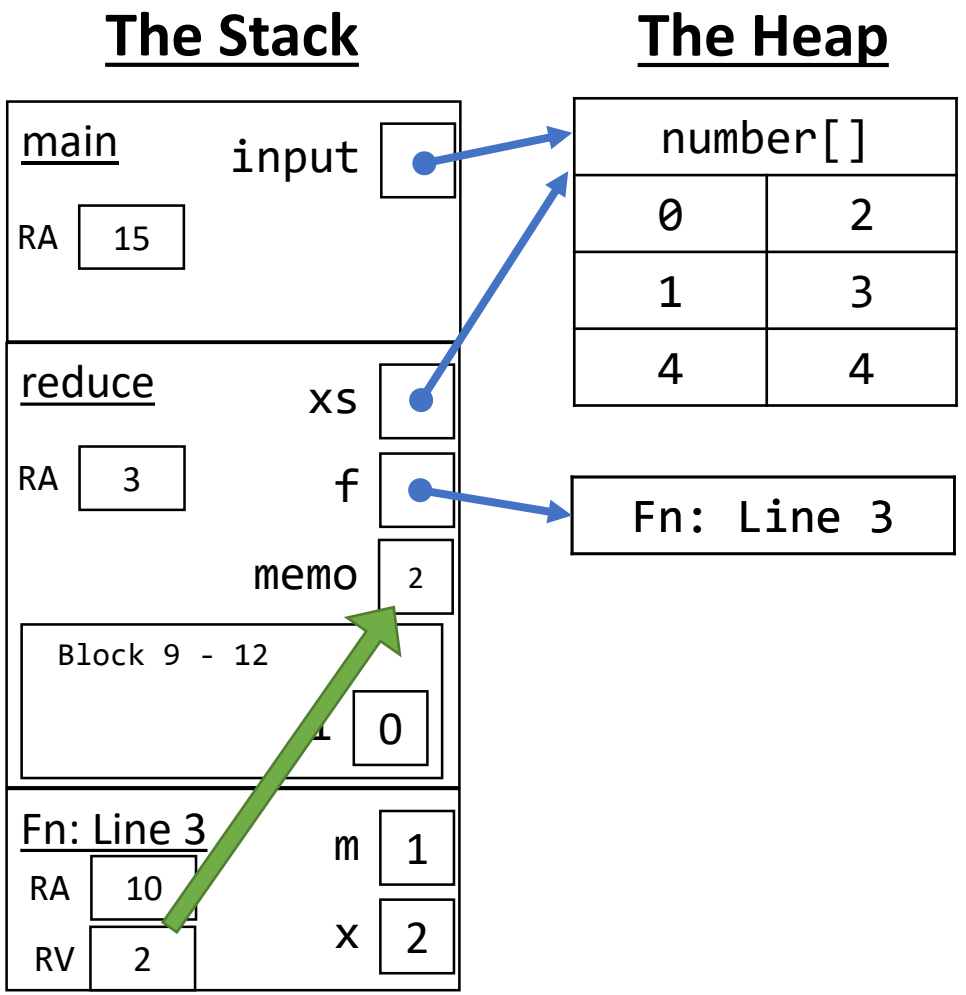
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```



- The function is written in shorthand notation implying the return value is going to be m * x
- The value of 2 is returned to line 10.

Updating memo

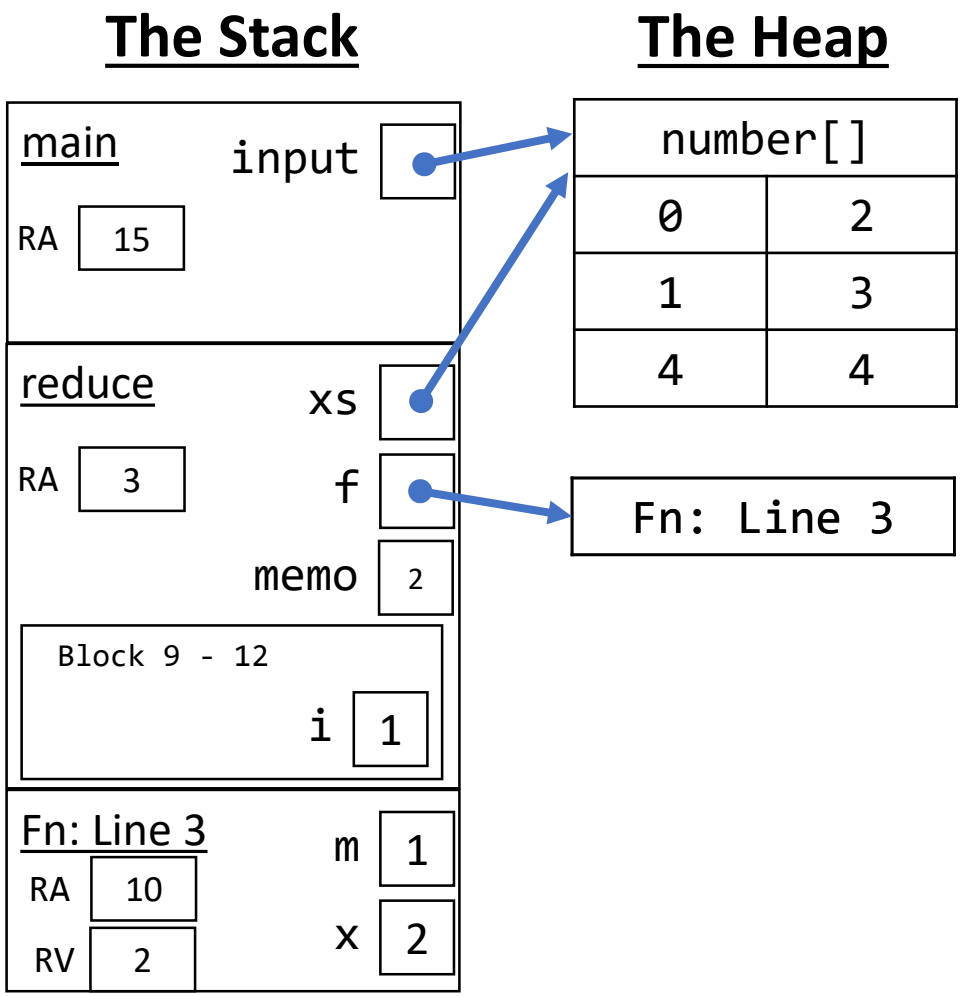
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|
12|   return memo;
13| };
14|
15| main();
```



- Memo is updated to the return value of the call to *f*.

Updating i

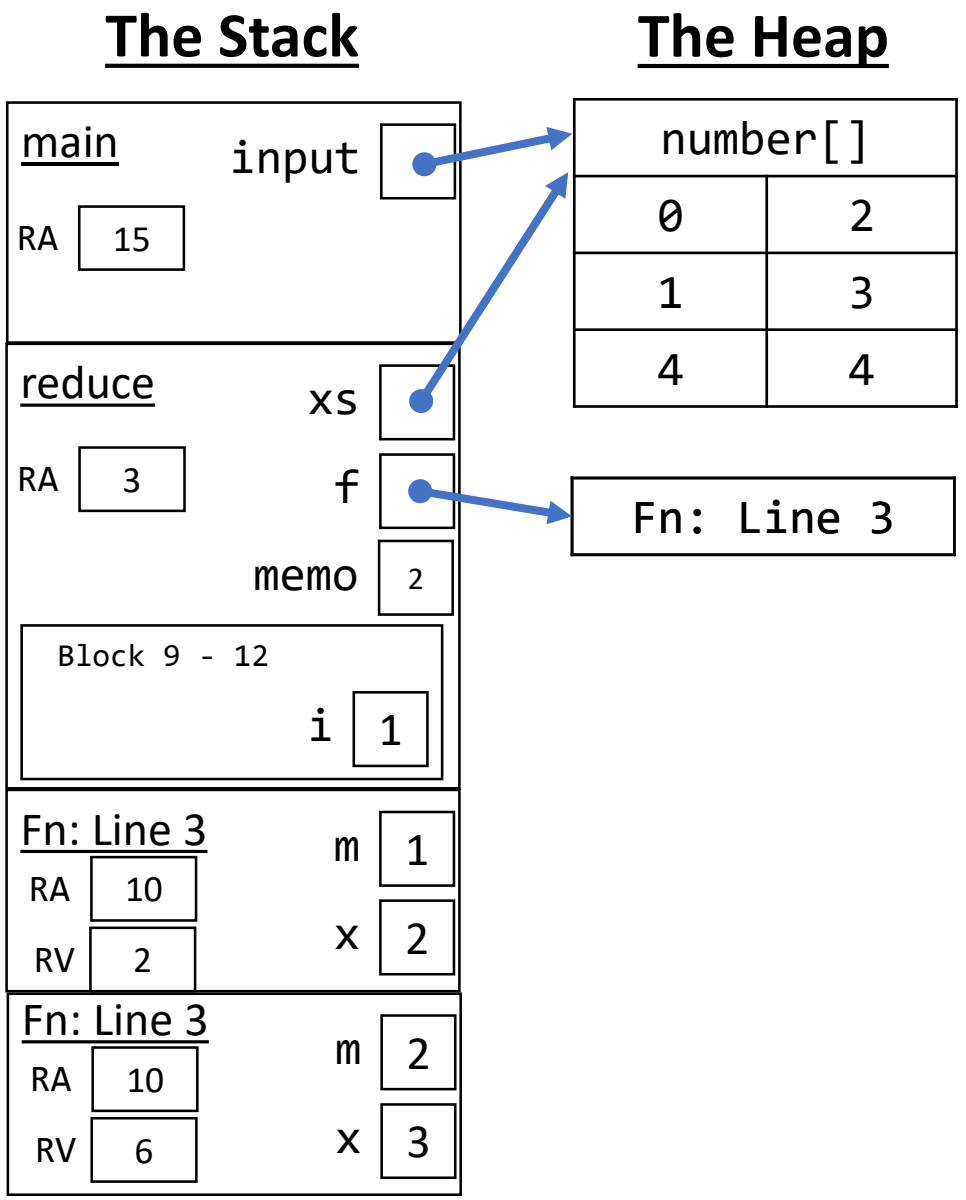
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```



- We're ready to move to the next index of our array.

Calling *f*, again.

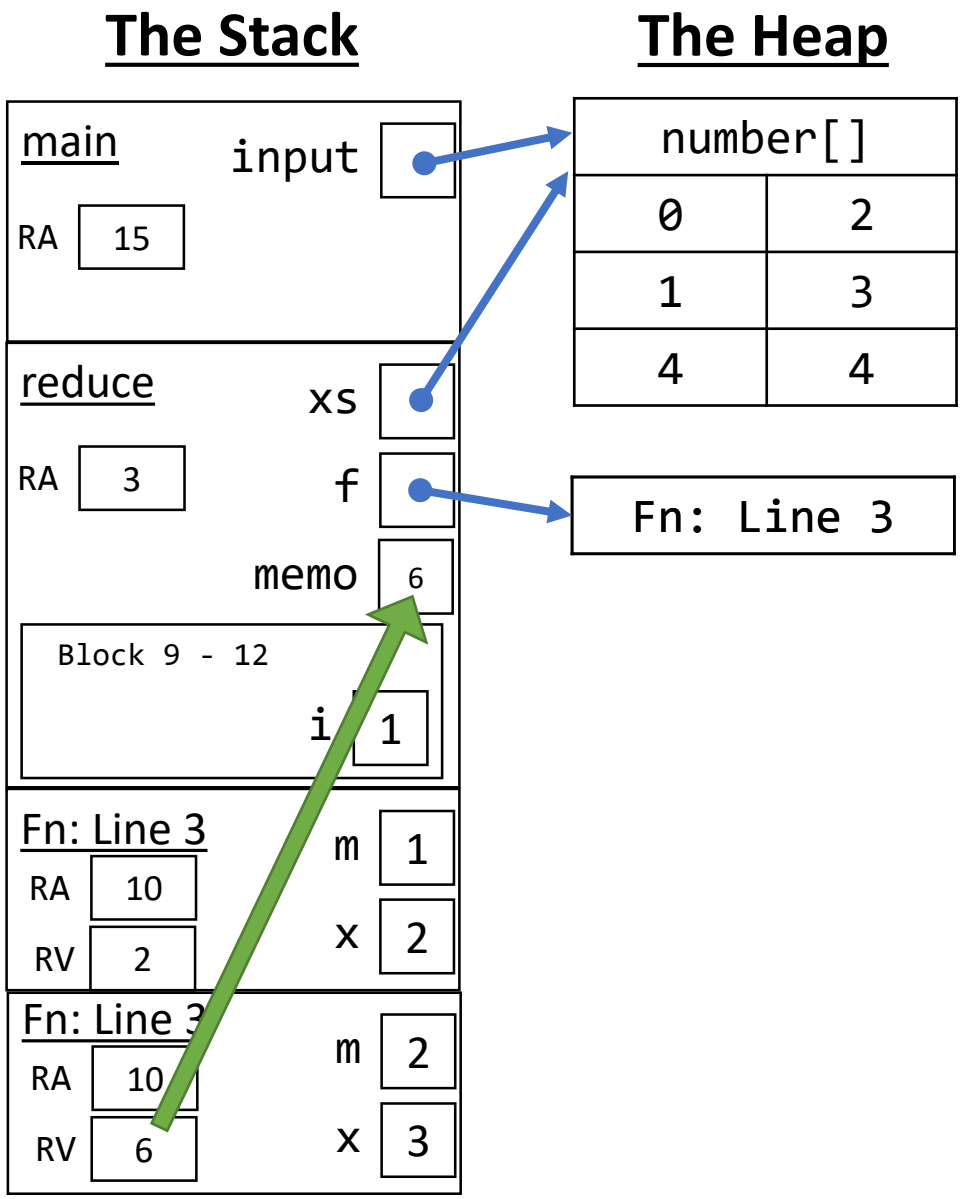
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```



- Notice this time, the value being assigned to parameter m is 2 because it's the current memo.
- Let's go ahead and evaluate the function, as well. We know 2 * 3 is 6 so the return value is 6.

Updating memo

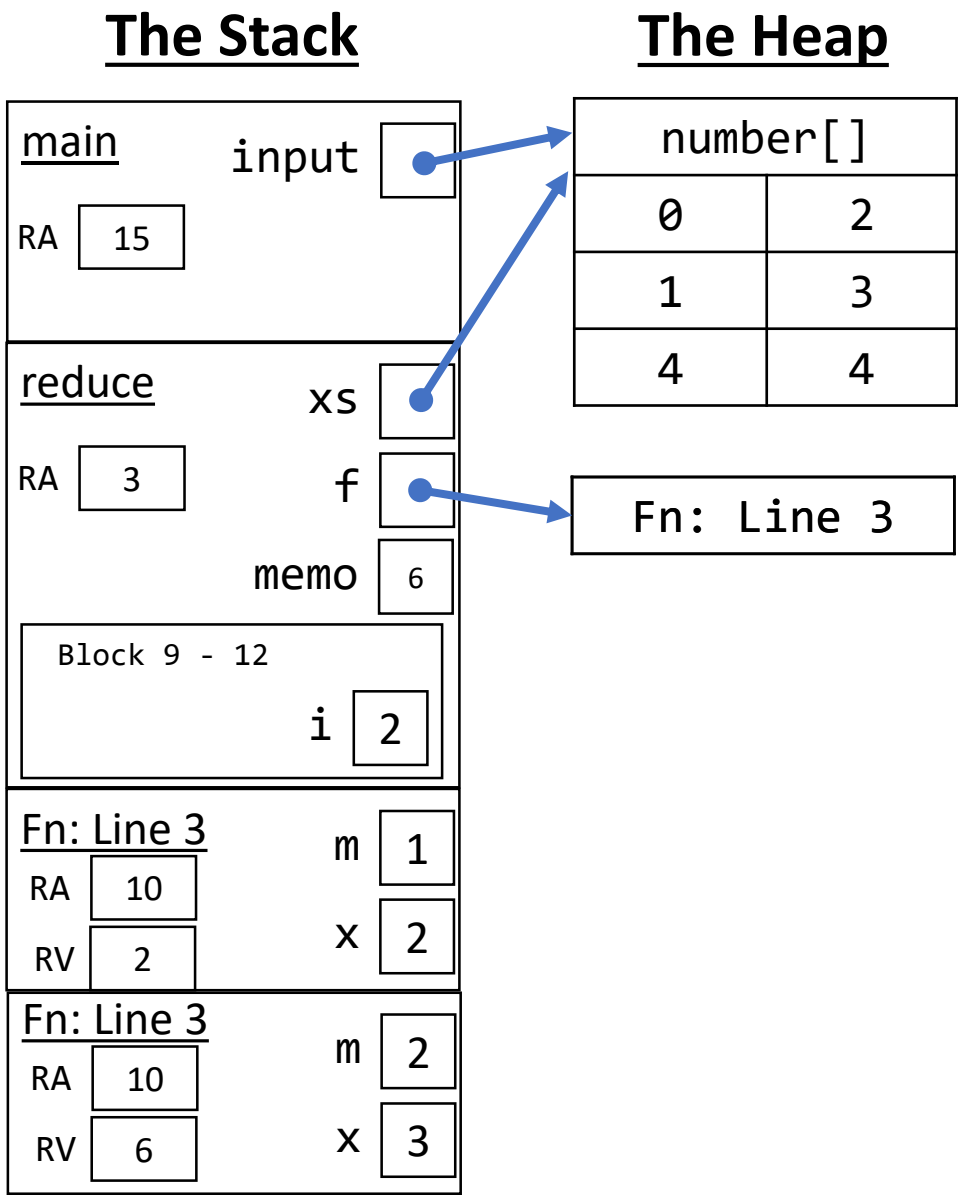
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|
12|   return memo;
13| };
14|
15| main();
```



- The return value is sent back to the assignment statement on line 10.

Updating i

```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```

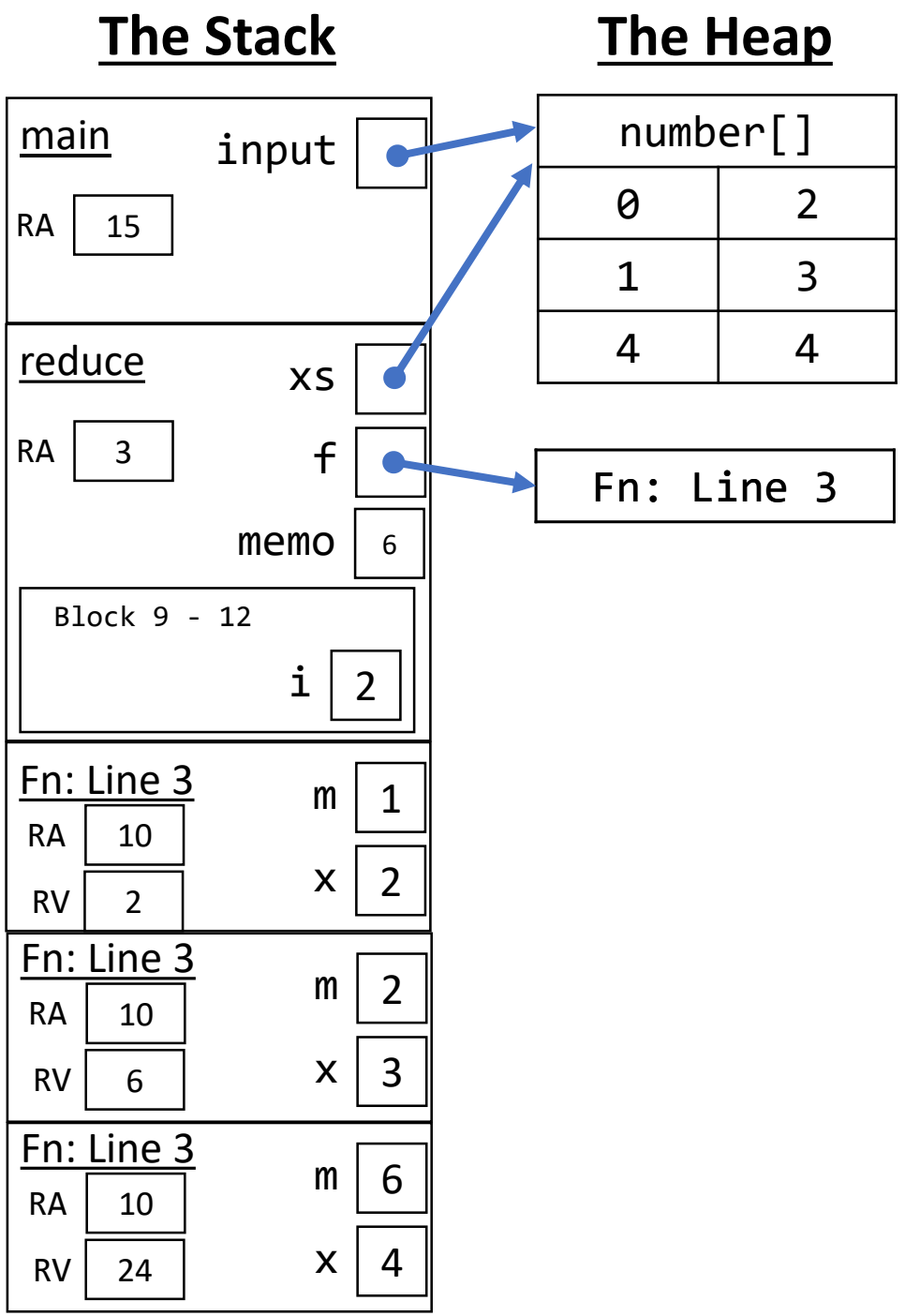


- The return value is sent back to the assignment statement on line 10.

Calling *f*, again.

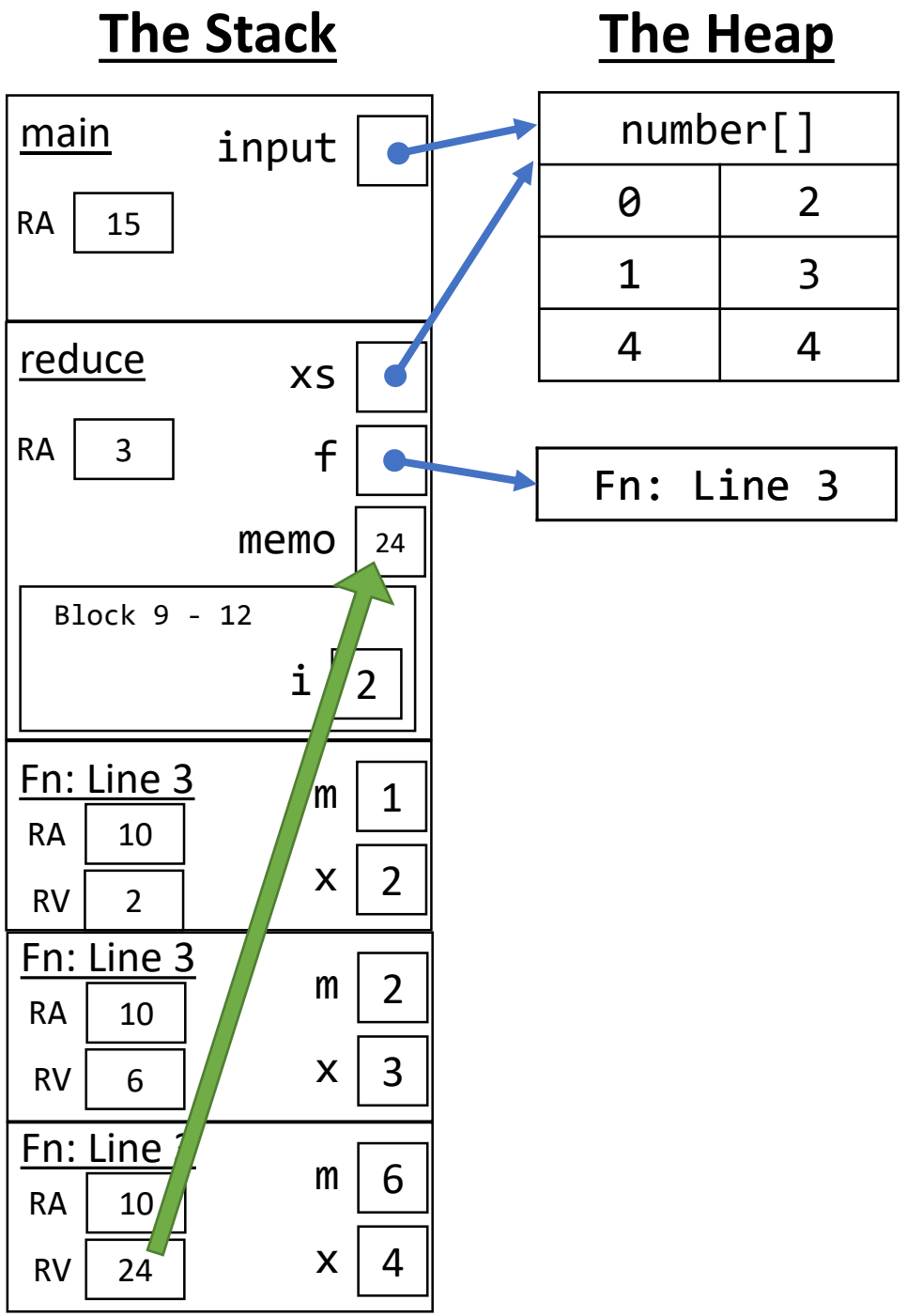
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   retu    emo;
13| };
14|
15| main();
```

- Once again, we're using the latest value of memo and evaluating this function in a single slide. The return value of 24 is…



Updating memo

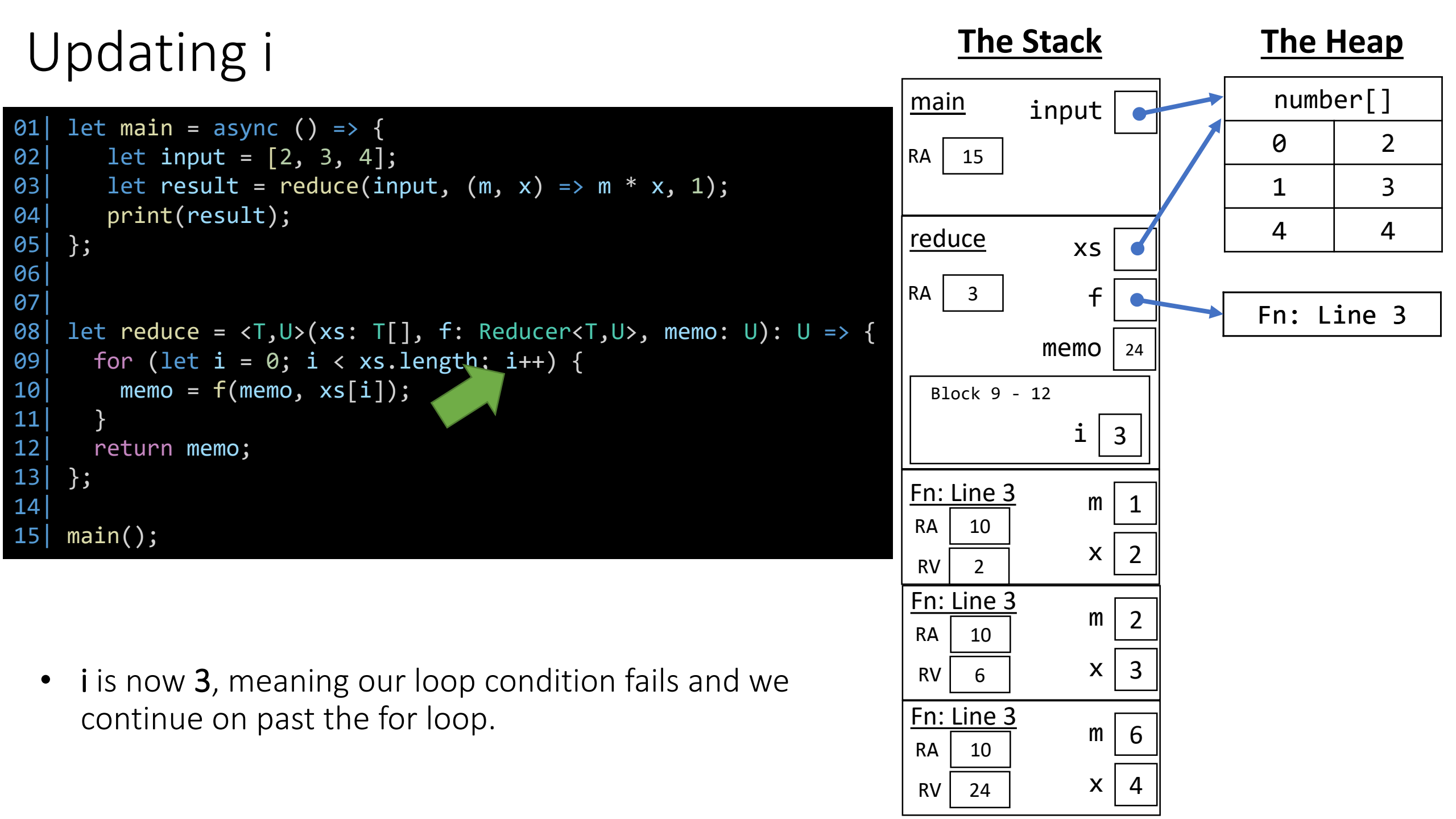
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|
12|   return memo;
13| };
14|
15| main();
```



- Memo is updated to the return value of the Reducer.

Updating i

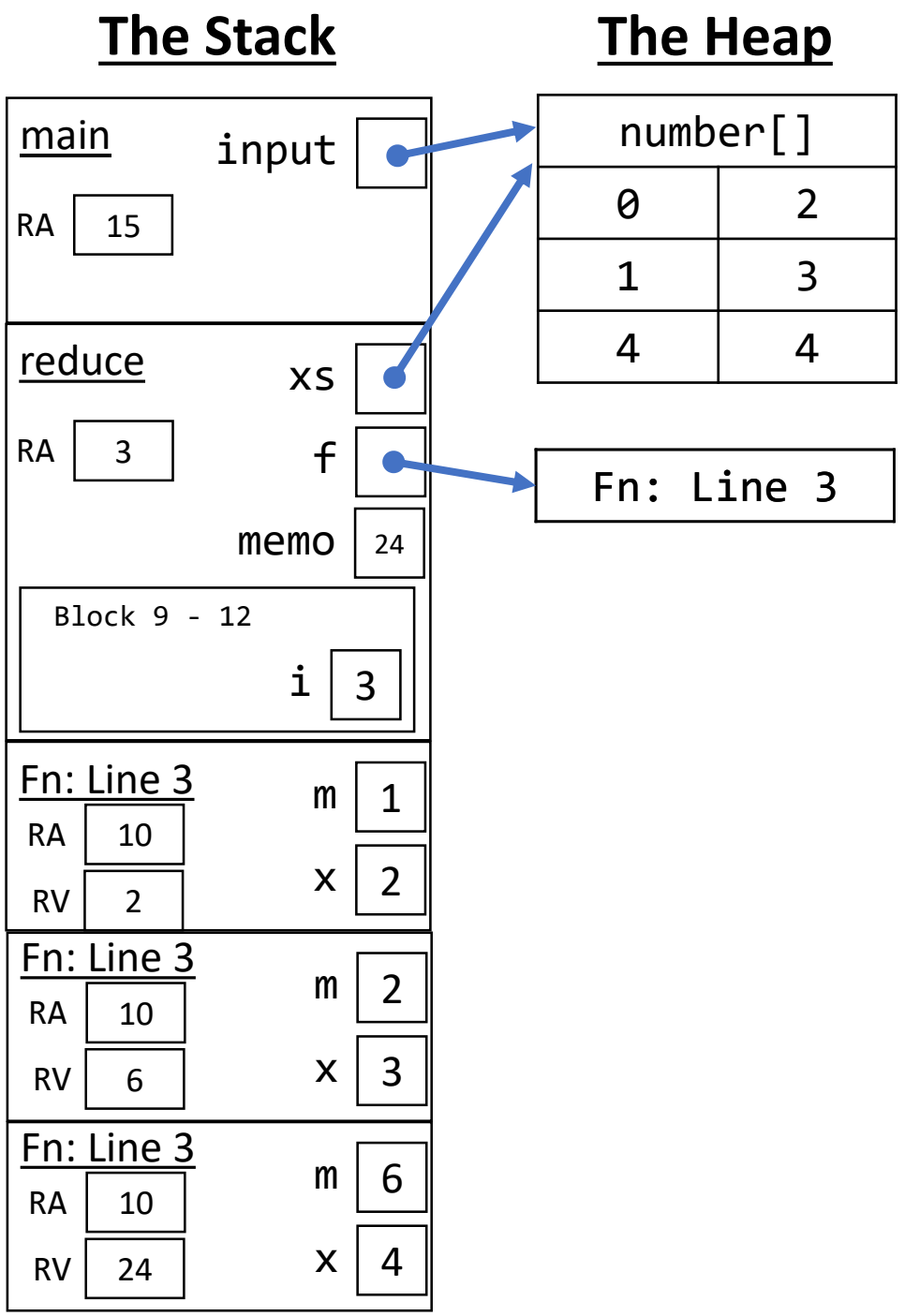
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```



- i is now 3, meaning our loop condition fails and we continue on past the for loop.

Returning memo

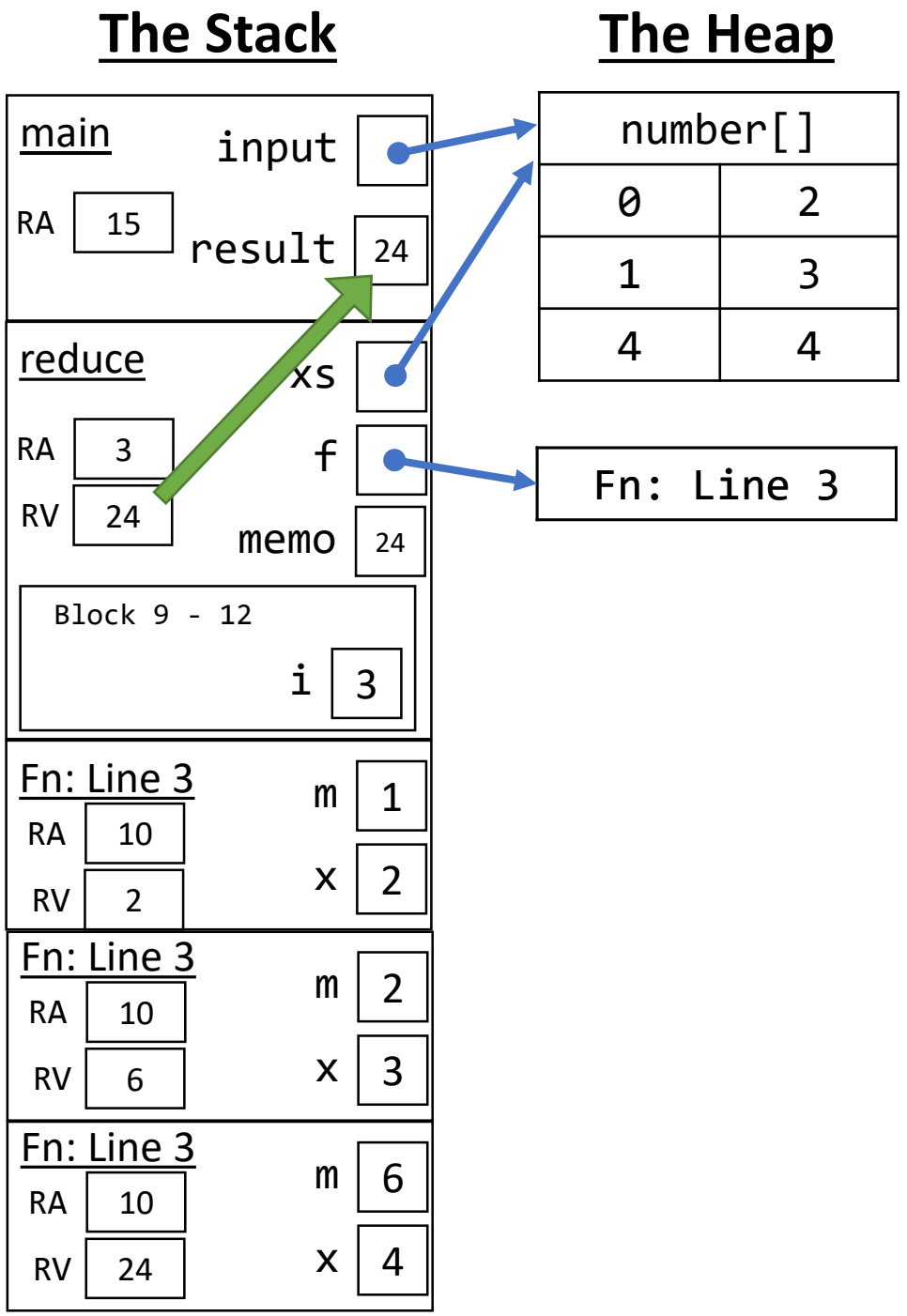
```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13|
14|
15| main();
```



- Notice memo's value now has a value of 24 in it. This will be sent back to line 3.

Initializing result

```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```

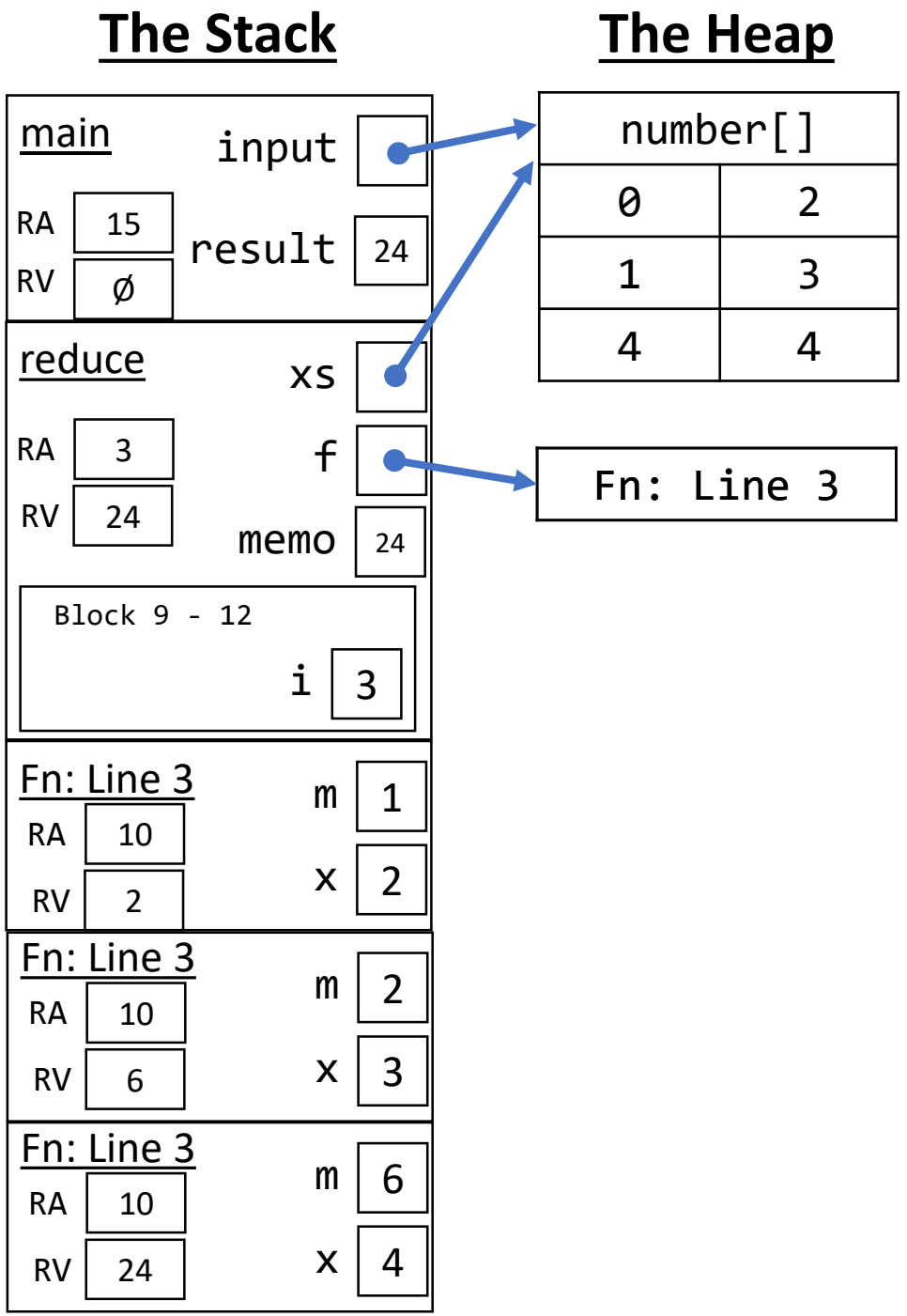


- Notice memo's value now has a value of 24 in it. This will be sent back to line 3.

Printing and Completing

```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, (m, x) => m * x, 1);
04|     print(result);
05| };
06|
07|
08| let reduce = <T,U>(xs: T[], f: Reducer<T,U>, memo: U): U => {
09|   for (let i = 0; i < xs.length; i++) {
10|     memo = f(memo, xs[i]);
11|   }
12|   return memo;
13| };
14|
15| main();
```

- The value 24 is printed out.
- The main function completes.



5. Answer a couple questions related to the signature and an example call to reduce:

Signature:

```
let reduce = <T, U> (xs: Node<T>, f: Reducer<T, U>, memo: U): U => {
```

Call:

```
 reduce(listify(2,4,6), (memo, item) => item + memo, "")
```

6. "Just for funcies" -- Given the code, draw an environment diagram at the breakpoint. Once drawn, answer the questions on PollEv.com

```
00  interface Funcy {
01      (a: number, b: number): number;
02  }
03
04  export let main = async () => {
05      let a = 16;
06      let b = 2;
07      let c = justF((or, funcies) => or - funcies, b, a);
08      // Breakpoint here
09      print(c);
10  };
11
12  let justF = (f: Funcy, a: number, b: number): number => {
13      return f(b / a, a);
14  };
15
16  main();
```

7. Answer the questions on PollEv.

```
let zip = <T> (a: Node<T>, b: Node<T>): Node<T[]> => {
   if (a === null || b === null) {
      return null;
   } else {
      let pair: T[] = [first(a), first(b)];
      return cons(pair, zip(rest(a), rest(b)));
   }
};

let ariana = listify("thank", "next");
let grande = listify("you", "!!!");
let fire = zip(ariana, grande);
```