PROMISE ALL SYNTAX

PROMISE ALL SYNTAX

- “The **`Promise.all()`** method returns a single `Promise` that resolves when all of the promises passed as an iterable have resolved or when the iterable contains no promises. It rejects with the reason of the first promise that rejects. “ - Mozilla Docs

PROMISE ALL EXAMPLE

```
var p1 = Promise.resolve(3);
var p2 = 1337;
var p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("foo");
  }, 100);
});

Promise.all([p1, p2, p3]).then(values => {
  console.log(values); // [3, 1337, "foo"]
});
```

ARRAYS IN JAVASCRIPT

MAP, REDUCE, FILTER

FUN WITH ARROW FUNCTIONS

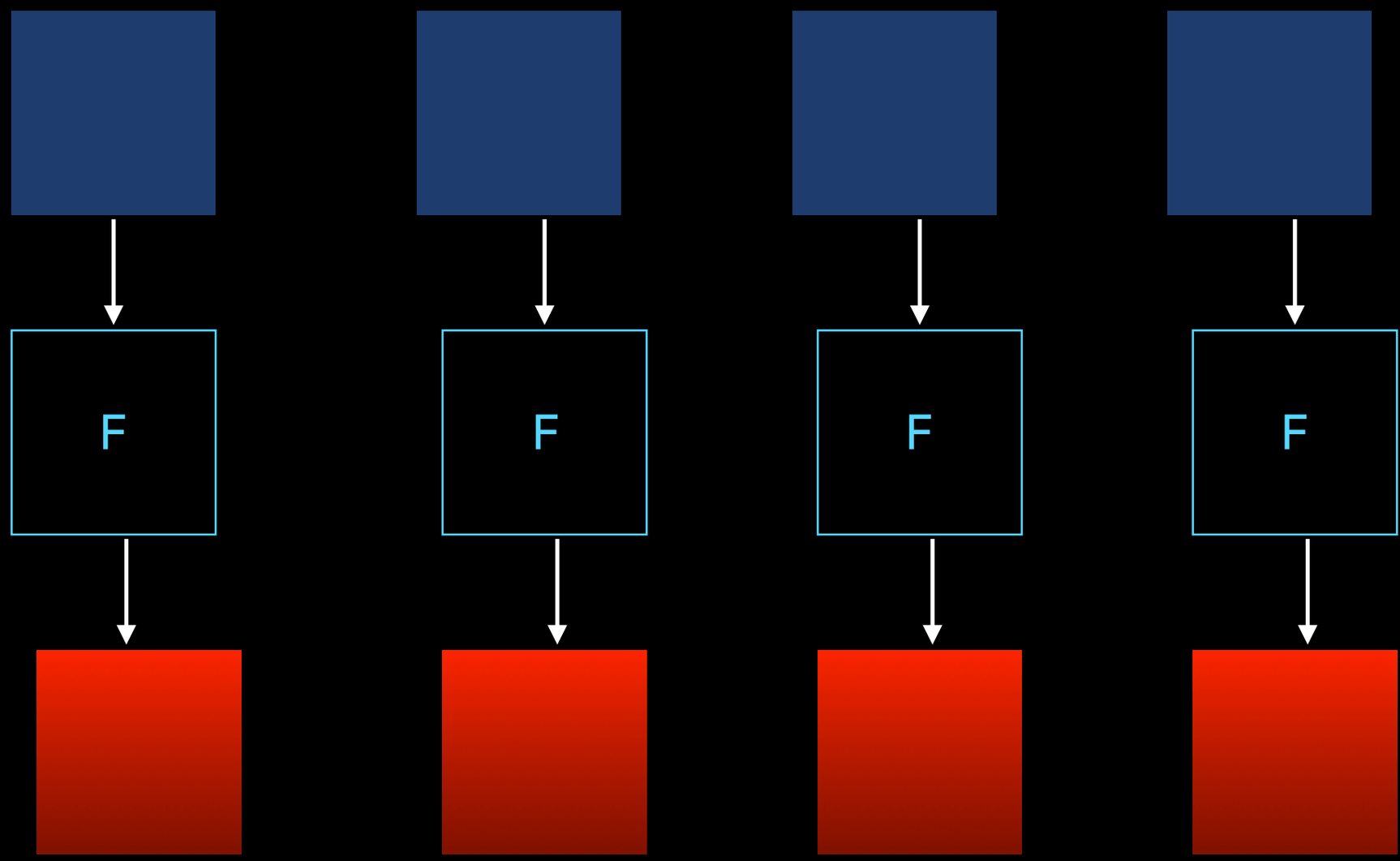
PRINT ALL THE IDS

```
students = [{grade:93, name: ‘DeShea'},
 {grade: 96, name: 'Devin'},
{grade: 90.0, name:'Phylicia'}]
```

```
ids = []
for(let i =0; i < students.length; i++){
    ids.push(students[i].grade)
}

console.log(ids)
```

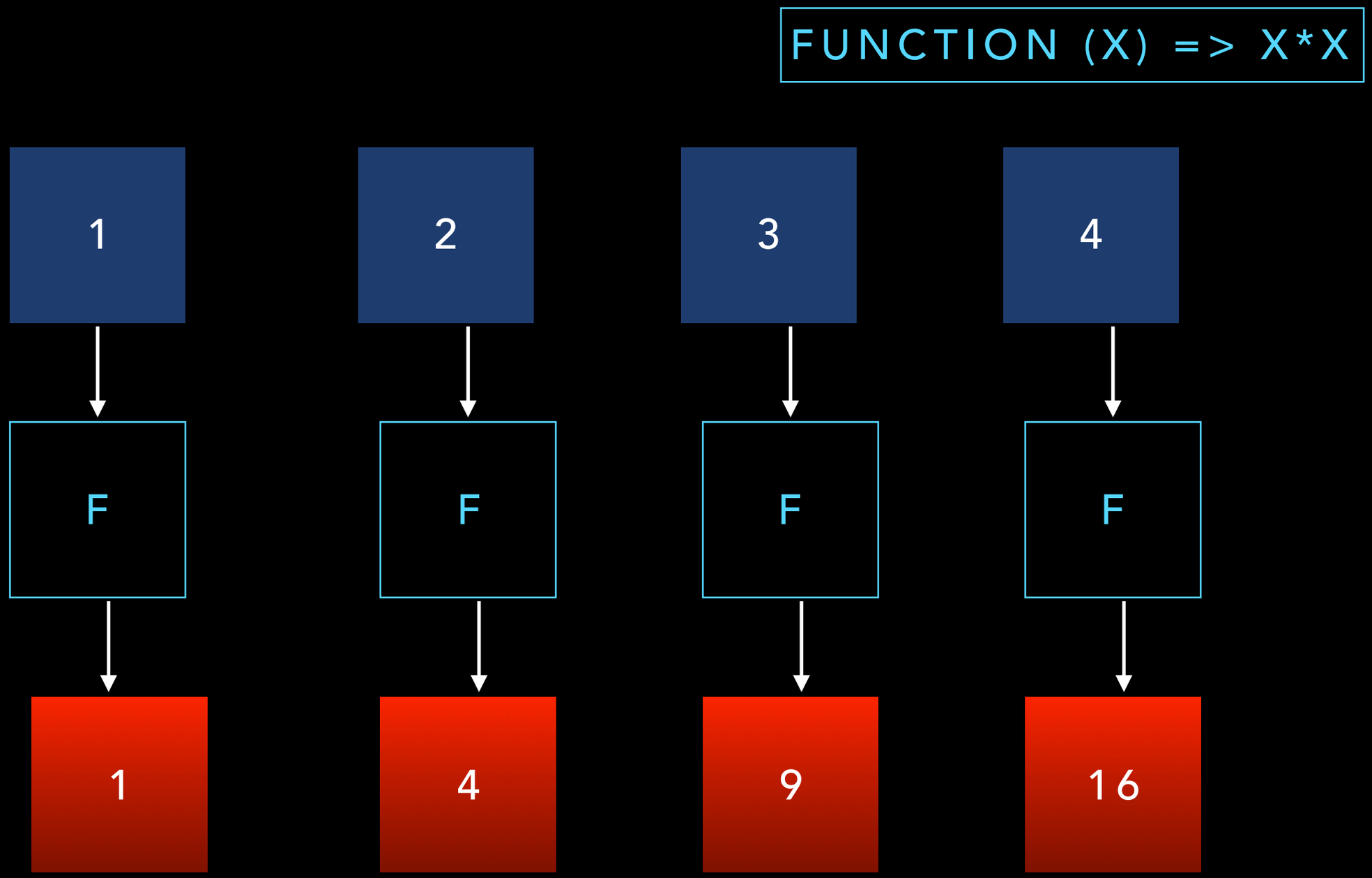
INTRODUCED THIS NEW SYNTAX OF ARRAY.PUSH

CONCEPT OF MAP



Apply some function F to every value in the array

CONCEPT OF MAP



Apply some function F to every value in the array

MAP FUNCTIONALITY

```
var grades = students.map(function (student) {
    return student.grade
  });

  console.log("map grades "+ grades)
```

MAP FUNCTION EVERY VALUE IN THE ARRAY

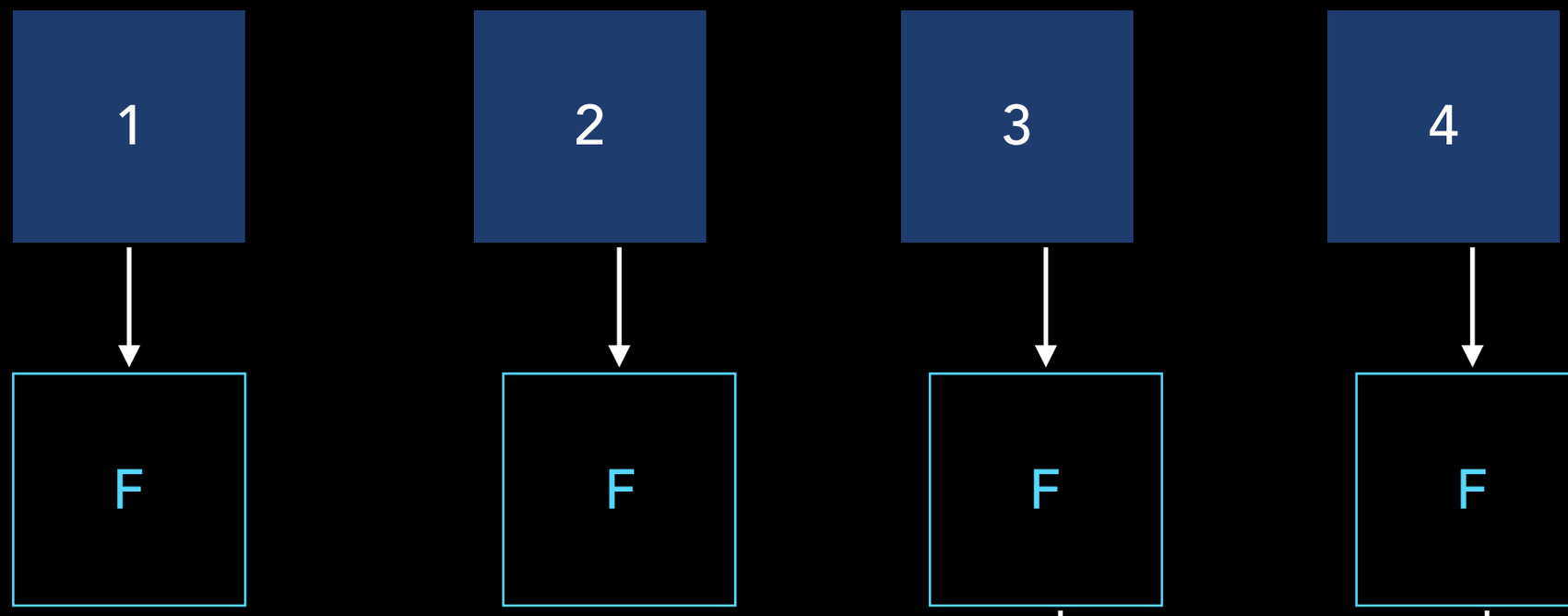
```
var grades = students.map(student => student.grade)

console.log("map grades "+ grades)
```

SIMPLY THE EXPRESSION USING ARROW NOTATION

FILTER

FUNCTION (X) => X > 2

Old Array



New Array

APPLY SOME FUNCTION F TO EVERY VALUE IN THE ARRAY
ONLY KEEP VALUES WHERE FUNCTION IS TRUE

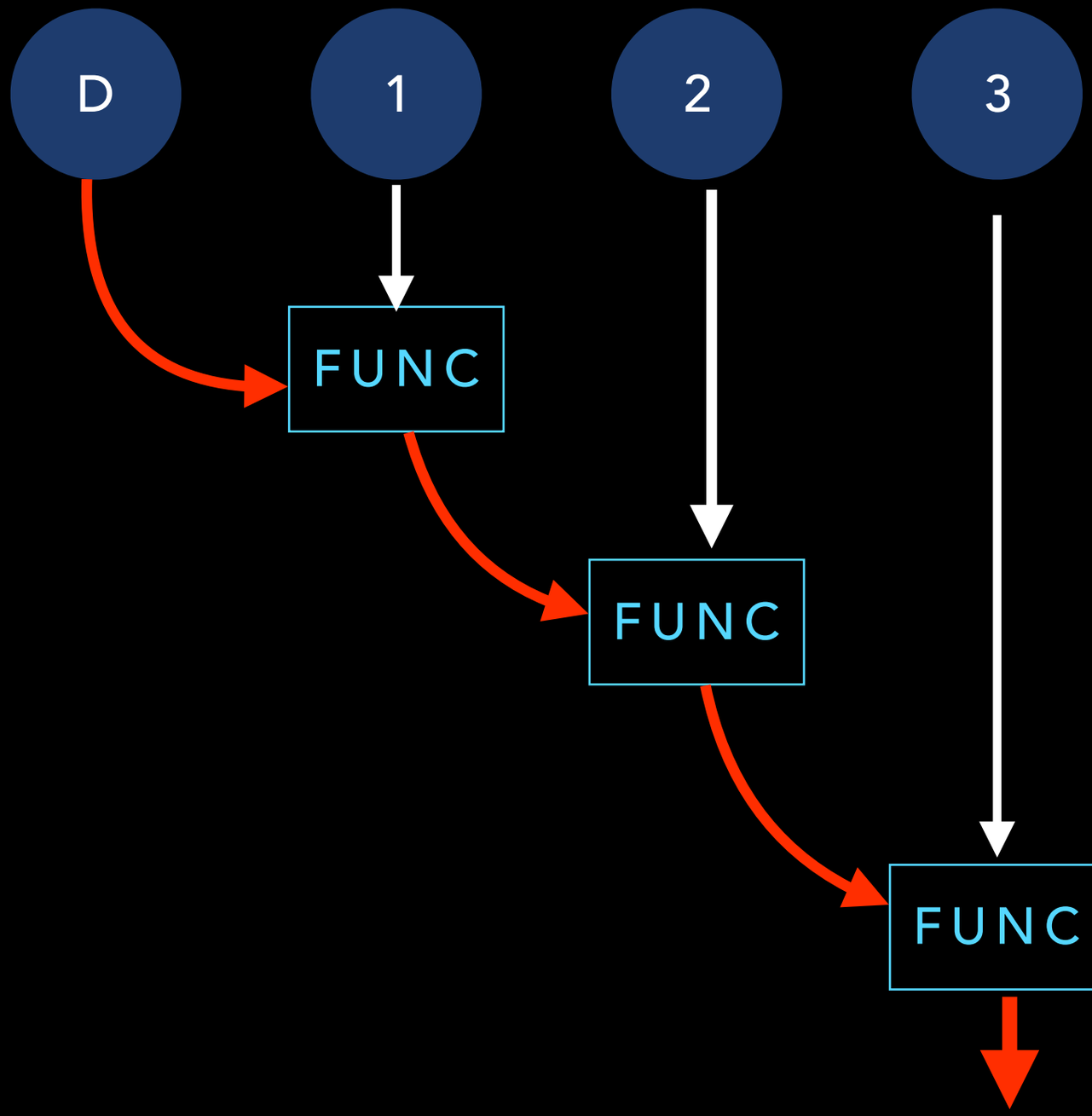
FILTER

```
above90 = students.filter(student => {
return student.grade > 90
})
console.log(above90)
```

```
[ { grade: 93, name: 'DeShea' }, { grade: 96, name: 'Devin' } ]
```

IF A MULTIPLE LINE ARROW FUNCTION IS USE IT MUST INCLUDE A RETURN STATEMENT

REDUCE VISUALIZATION

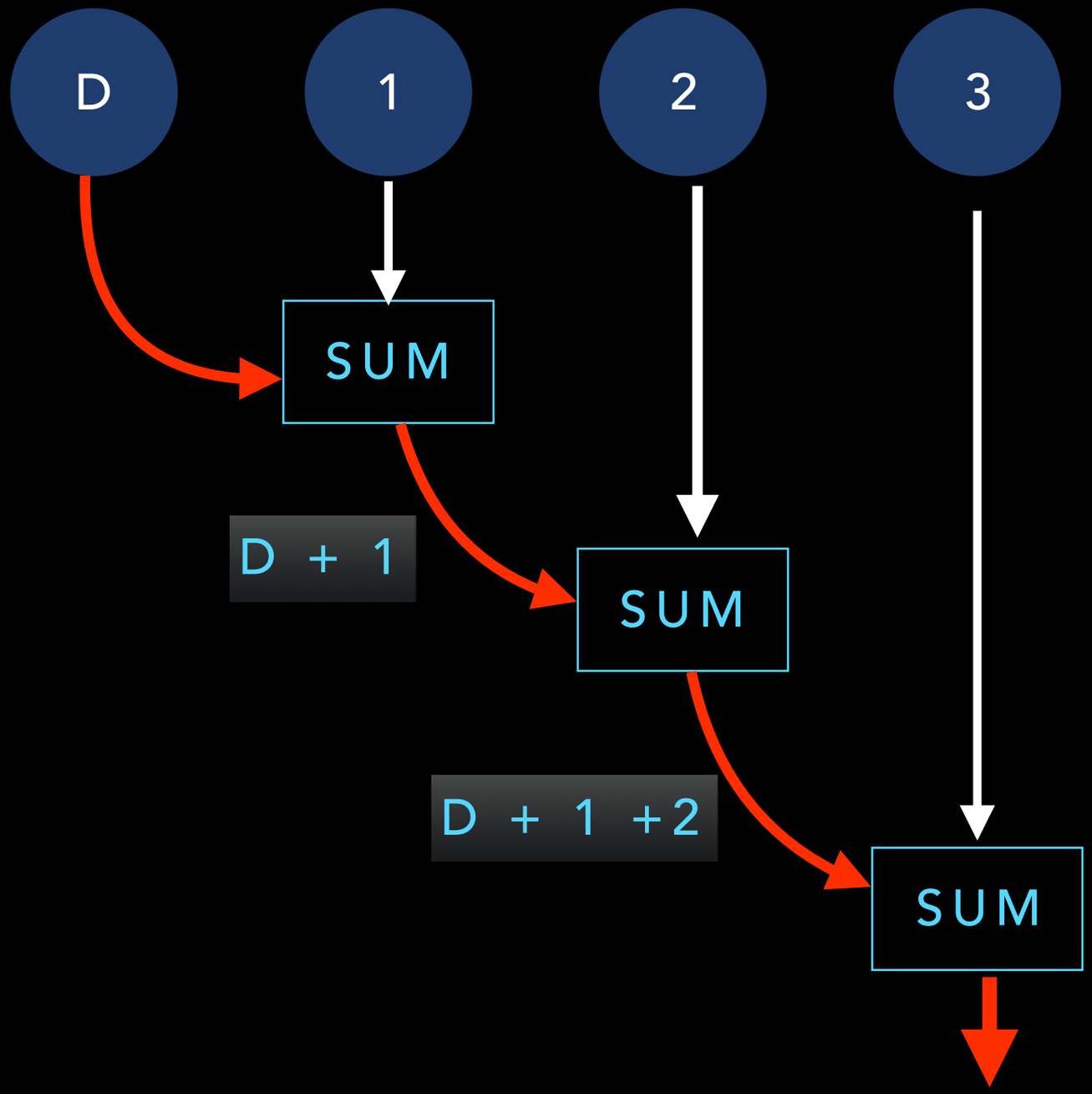


TAKES 2 PARAMETERS

RESULT IS STORE
BACK IN FIRST PARAMETER

(((D+1)+2)+3)

REDUCE VISUALIZATION



TAKES 2 PARAMETERS

RESULT IS STORE
BACK IN FIRST PARAMETER

(((D+1)+2)+3)

REDUCE TO SINGLE VALUE

REDUCE

DEFAULT VALUE

Grade [93, 96, 90]

Sum 93, 189, 279

```
average = grades.reduce((sum, grade) =>sum += grade, 0)/grades.length
console.log(average)
```

THE LAST PARAMETER IS ALWAYS THE VALUE ARRAY

REDUCE QUIZ

WRITE A REDUCTION FINDS THE HIGHEST SCORER

```
highestGrade  = students.reduce((highScorer, student) => {
    return (highScorer.grade || 0) > student.grade ? highScorer :
student;
  });

console.log(highestGrade)
```

ITERATORS

ASYNC INTERATORS

- Iterable Objects are a generalization of arrays

```
let range = {
    from: 1,
    to: 5
};

// We want the for..of to work:
// for(let num of range) ... num=1,2,3,4,5
```

ITERATOR ON OBJECT

```
let range = {
    from: 1,
    to: 5
  };

  // 1. call to for..of initially calls this
  range[Symbol.iterator] = function() {

    // ...it returns the iterator object:
    // 2. Onward, for..of works only with this iterator, asking it for next values
    return {
      current: this.from,
      last: this.to,

      // 3. next() is called on each iteration by the for..of loop
      next() {
        // 4. it should return the value as an object {done:.., value :...}
        if (this.current <= this.last) {
          return { done: false, value: this.current++ };
        } else {
          return { done: true };
        }
      }
    };
  };

  // now it works!
  for (let num of range) {
    console.log(num); // 1, then 2, 3, 4, 5
  }
```

```
let range = {
  from: 1,
  to: 5,

  // for await..of calls this method once in the very beginning
  [Symbol.asyncIterator]() { // (1)
    // ...it returns the iterator object:
    // onward, for await..of works only with that object,
    // asking it for next values using next()
    return {
      current: this.from,
      last: this.to,

      // next() is called on each iteration by the for..of loop
      async next() { // (2)
        // it should return the value as an object {done:.., value :...}
        // (automatically wrapped into a promise by async)

        // can use await inside, do async stuff:
        await new Promise(resolve => setTimeout(resolve, 1000)); // (3)

        if (this.current <= this.last) {
          return { done: false, value: this.current++ };
        } else {
          return { done: true };
        }
      }
    };
  }
};

(async () => {

  for await (let value of range) { // (4)
    alert(value); // 1,2,3,4,5
  }

})()
```