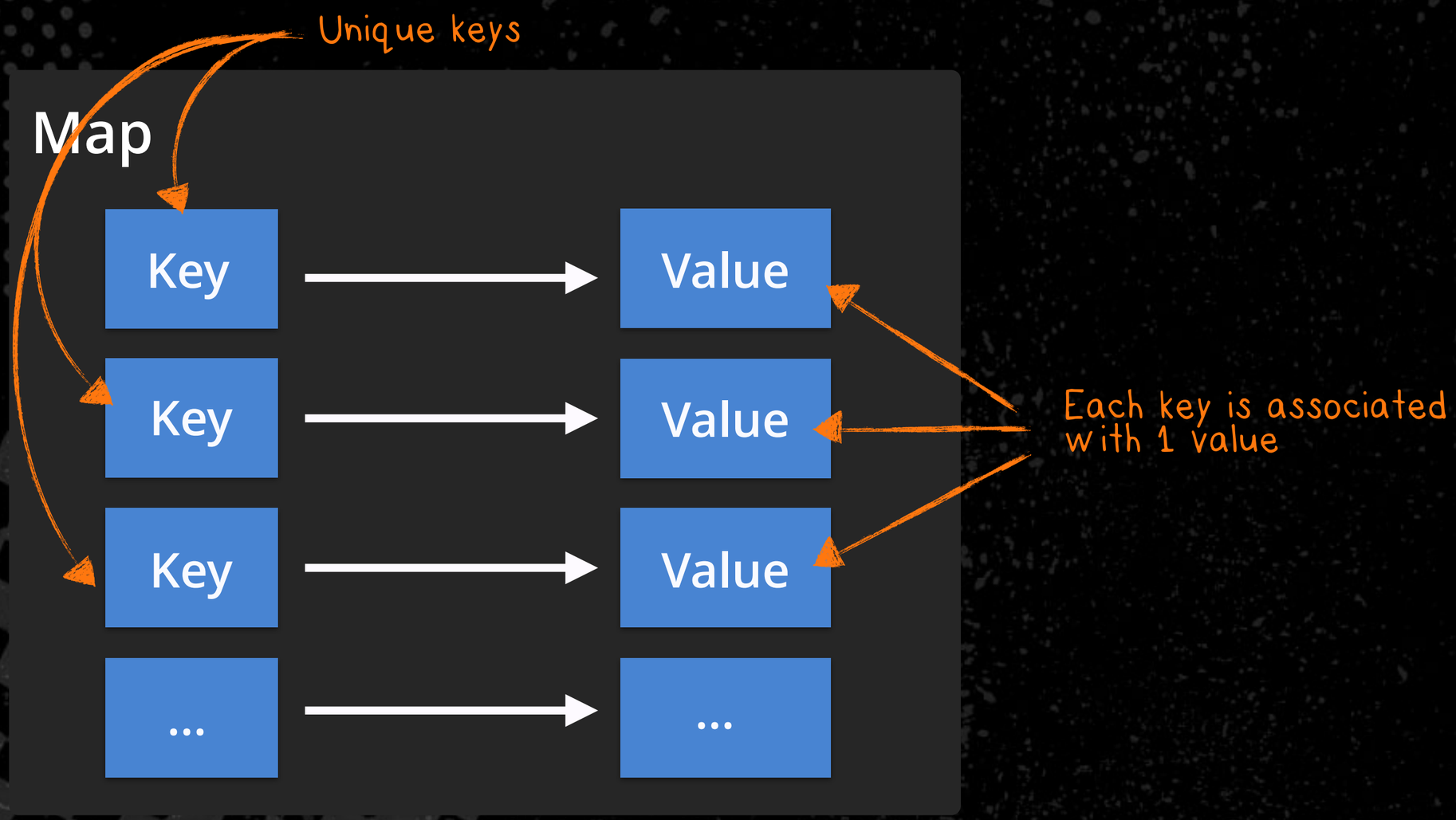
Maps

Level 4 – Section 2

The Map Data Structure

Maps are a **data structure** composed of a collection of **key/value** pairs. They are very useful to store simple data, such as property values.



Issues With Using Objects as Maps

When using **Objects** as maps, its *keys* are **always converted to strings.**

Two different objects

```
let user1 = { name: "Sam" };
let user2 = { name: "Tyler" };

let totalReplies = {};
totalReplies[user1] = 5;
totalReplies[user2] = 42;

console.log( totalReplies[user1] );
console.log( totalReplies[user2] );

console.log( Object.keys(totalReplies) );
```

Both objects are converted to the string "[object Object]"

```
> 42
> 42
```

```
> ["[object Object]"]
```

Storing Key/Values With Map

The *Map* object is a simple **key/value** data structure. **Any value** may be used as either a key or a value, and objects are **not converted** to strings.

```
let user1 = { name: "Sam" };
let user2 = { name: "Tyler" };

let totalReplies = new Map();
totalReplies.set( user1, 5 );
totalReplies.set( user2, 42 );

console.log( totalReplies.get(user1) );
console.log( totalReplies.get(user2) );
```

Values assigned to different object keys, as expected

```
> 5
> 42
```

We use the get() and set() methods to access values in Maps

Use Maps When Keys Are Unknown Until Runtime

Map

```
let recentPosts = new Map();

createPost(newPost, (data) => {
  recentPosts.set( data.author, data.message );
});
```

Keys unknown until runtime, so... Map!

Object

```
const POSTS_PER_PAGE = 15;

let userSettings = {
  perPage:  POSTS_PER_PAGE,
  showRead: true,
};
```

Keys are previously defined, so... Object!

Use Maps When Types Are the Same

Map

```
let recentPosts = new Map();

createPost(newPost, (data) => {
  recentPosts.set( data.author, data.message );
});

// ...somewhere else in the code
socket.on('new post', function(data){
  recentPosts.set( data.author, data.message );
});
```

All keys are the same type, and
all values are the same type, so Map!

Object

```
const POSTS_PER_PAGE = 15;

let userSettings = {
  perPage:  POSTS_PER_PAGE,
  showRead: true,
};
```

Some values are numeric,
others are boolean, so Object!

Iterating Maps With for...of

Maps are iterable, so they can be used in a *for...of* loop. Each run of the loop returns a **[key, value]** pair for an entry in the *Map.*

```
let mapSettings = new Map();

mapSettings.set( "user", "Sam" );
mapSettings.set( "topic", "ES2015" );
mapSettings.set( "replies", ["Can't wait!", "So Cool"] );

for(let [key, value] of mapSettings){
  console.log(`${key} = ${value}`);
}
```

Remember array destructuring ?

```
> user = Sam
> topic = ES2015
> replies = Can't wait!,So Cool
```

WeakMap

The *WeakMap* is a type of *Map* where **only objects** can be passed as keys. Primitive data types — such as strings, numbers, booleans, etc. — are **not allowed.**

```
let user = {};
let comment = {};

let mapSettings = new WeakMap();
mapSettings.set( user, "user" );
mapSettings.set( comment, "comment" );

console.log( mapSettings.get(user) );
console.log( mapSettings.get(comment) );

mapSettings.set("title", "ES2015");
```

```
> user
> comment
```

```
> Invalid value used as weak map key
```

Primitive data types are not allowed

Working With WeakMaps

All available methods on a *WeakMap* require access to an **object used as a key**.

```
let user = {};

let mapSettings = new WeakMap();
mapSettings.set( user, "ES2015" );

console.log( mapSettings.get(user) );
console.log( mapSettings.has(user) );
console.log( mapSettings.delete(user) );
```

```
> ES2015
> true
> true
```

WeakMaps are **not iterable,** therefore they can't be used with *for...of*

```
for(let [key,value] of mapSettings){
  console.log(`${key} = ${value}`);
}
```

```
> mapSettings[Symbol.iterator] is not a function
```

WeakMaps Are Better With Memory

Individual entries in a *WeakMap* can be **garbage collected** while the *WeakMap* itself still exists.

```
let user = {};

let userStatus = new WeakMap();
userStatus.set( user, "logged" );

//...
someOtherFunction( user );
```

All objects occupy memory space

Object reference passed as key to the WeakMap

Once it returns, user can be garbage collected

WeakMaps don't prevent the garbage collector from collecting objects currently used as keys, but that are no longer referenced anywhere else in the system