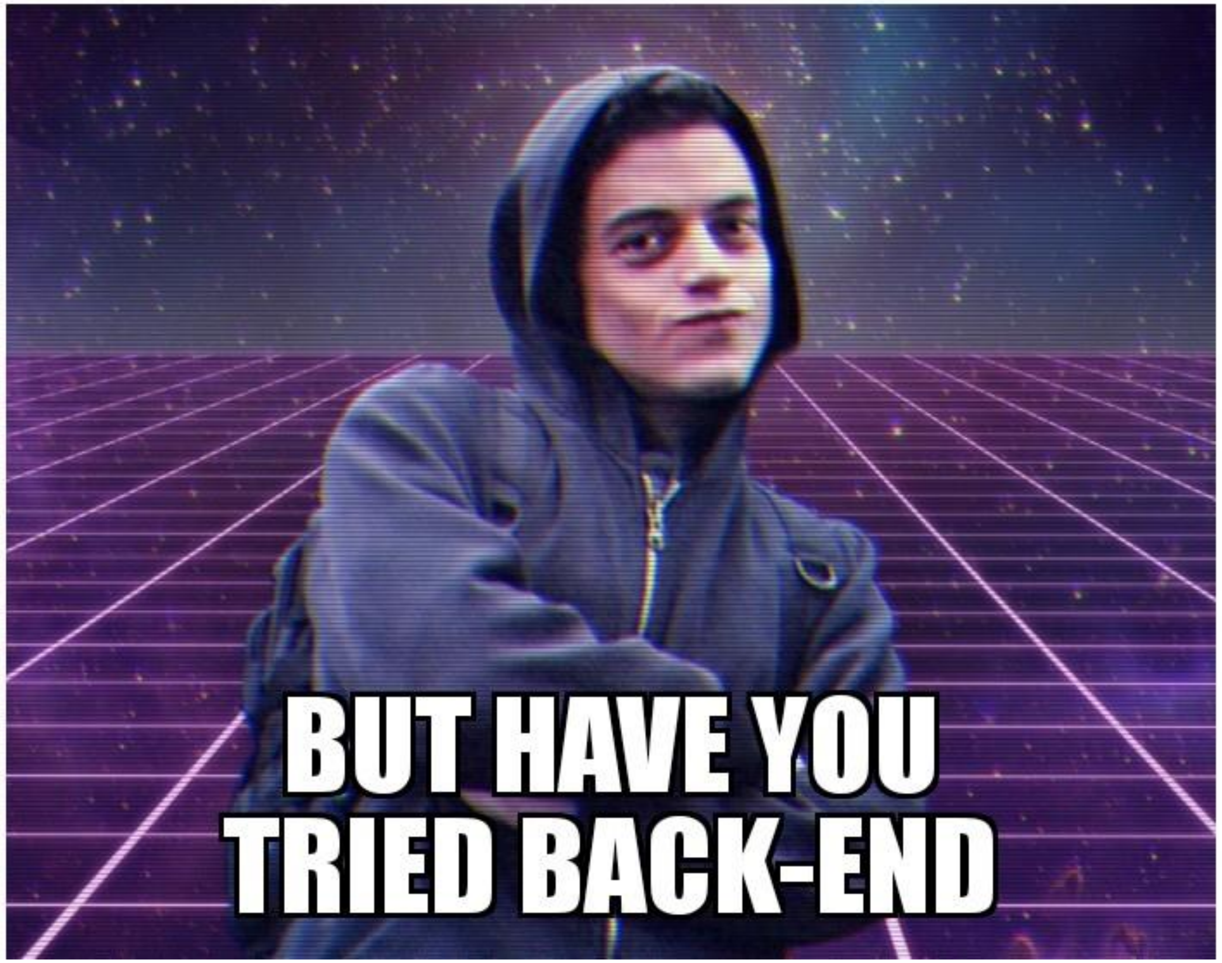
YEAH FRONT-END IS COOL



Introduction to Buffer Module

Buffer module is **commonly used to work with binary data directly**. It is a part of Node.js and allows you to work with raw binary data in a variety of encodings.

But why to work with binary data directly?

- **Performance**: Operations on buffers are generally faster compared to string manipulation when dealing with binary data, especially when working with large datasets.
- **Flexibility**: Buffers can be resized, sliced, concatenated, and manipulated in various ways to suit different use cases.
- **Interoperability:** Buffers can be easily converted to and from other data types, such as strings, arrays, and TypedArrays, facilitating interoperability with different parts of an application or external systems.

Creating a Buffer:

```
// Creating a Buffer of size 4  bytes
const buffer = Buffer.alloc(4);
console.log(buffer);
```

Writing and Reading Data:

- You can write data into the buffer and read it back:

```
// Writing data to the buffer
buffer.write('Hello', 'utf-8');
// Reading data from the buffer
const data = buffer.toString('utf-8');
console.log(data); // Output: Hello
```

Concatenating Buffers:

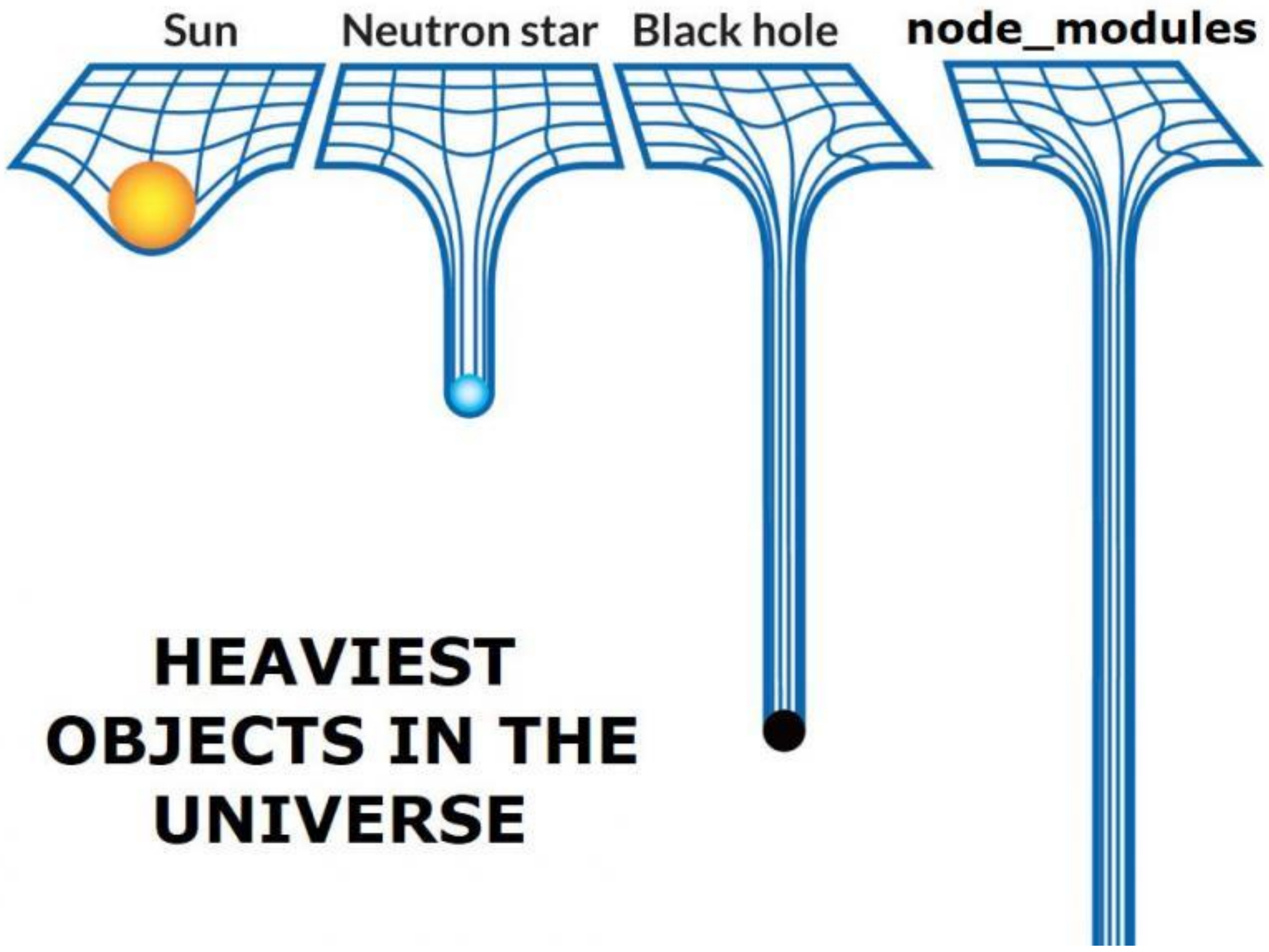
```
const buffer1 = Buffer.from('Hello', 'utf-8');
const buffer2 = Buffer.from(' World', 'utf-8');

// Concatenating buffers
const concatenatedBuffer = Buffer.concat([buffer1,
buffer2]);
console.log(concatenatedBuffer.toString('utf-8'));
// Output: Hello World
```

Stream Module

Akash Pundir

System Programming –I

School of Computer Science and Engineering



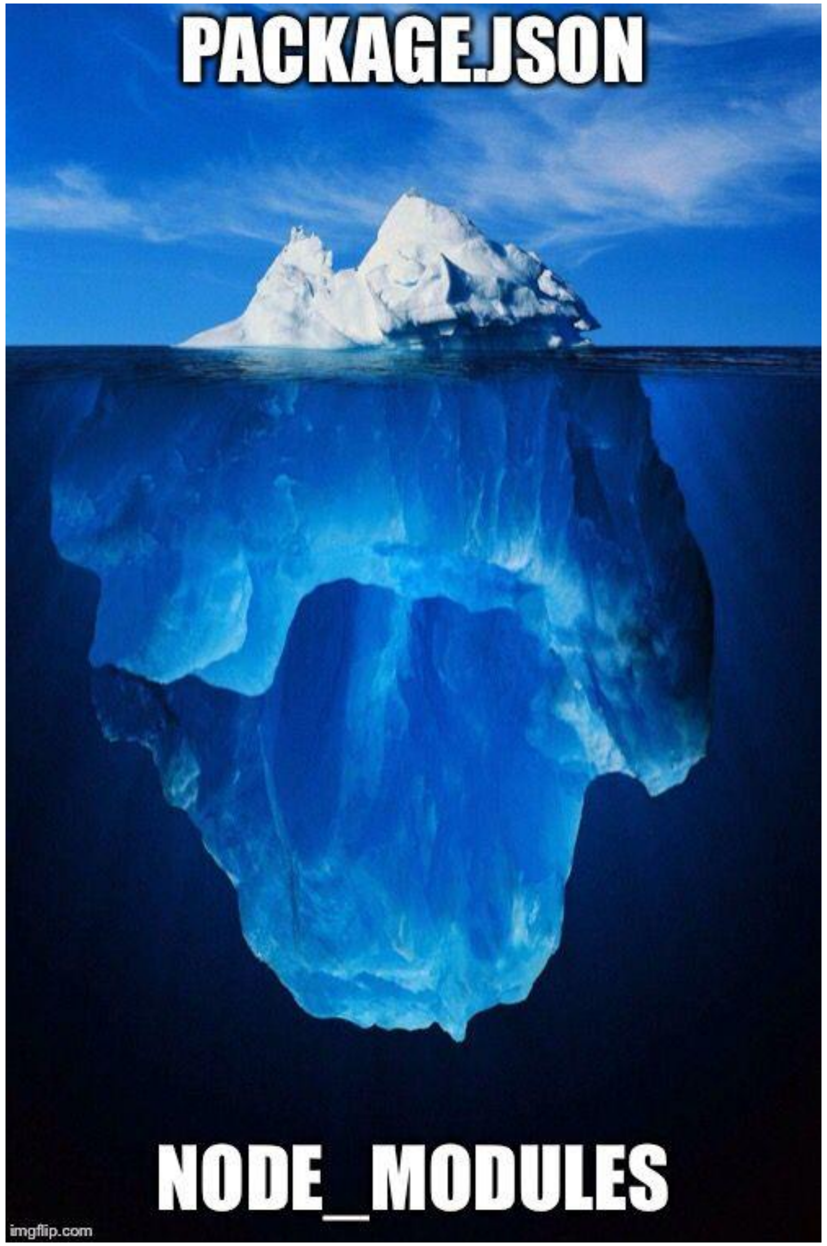
HEAVIEST OBJECTS IN THE UNIVERSE

Whaaat is this, Stream?

Streams are **sequences of data made available over time. Rather than reading or writing all the data at once, streams allow you to process data piece by piece**, which is particularly useful when dealing with large datasets or when real-time processing is required.

So, Stream Module….

In essence, the Stream Module enables efficient handling of data streams, allowing developers to process data in smaller, manageable chunks, leading to better performance.


PACKAGE.JSON
NODE_MODULES
imgflip.com

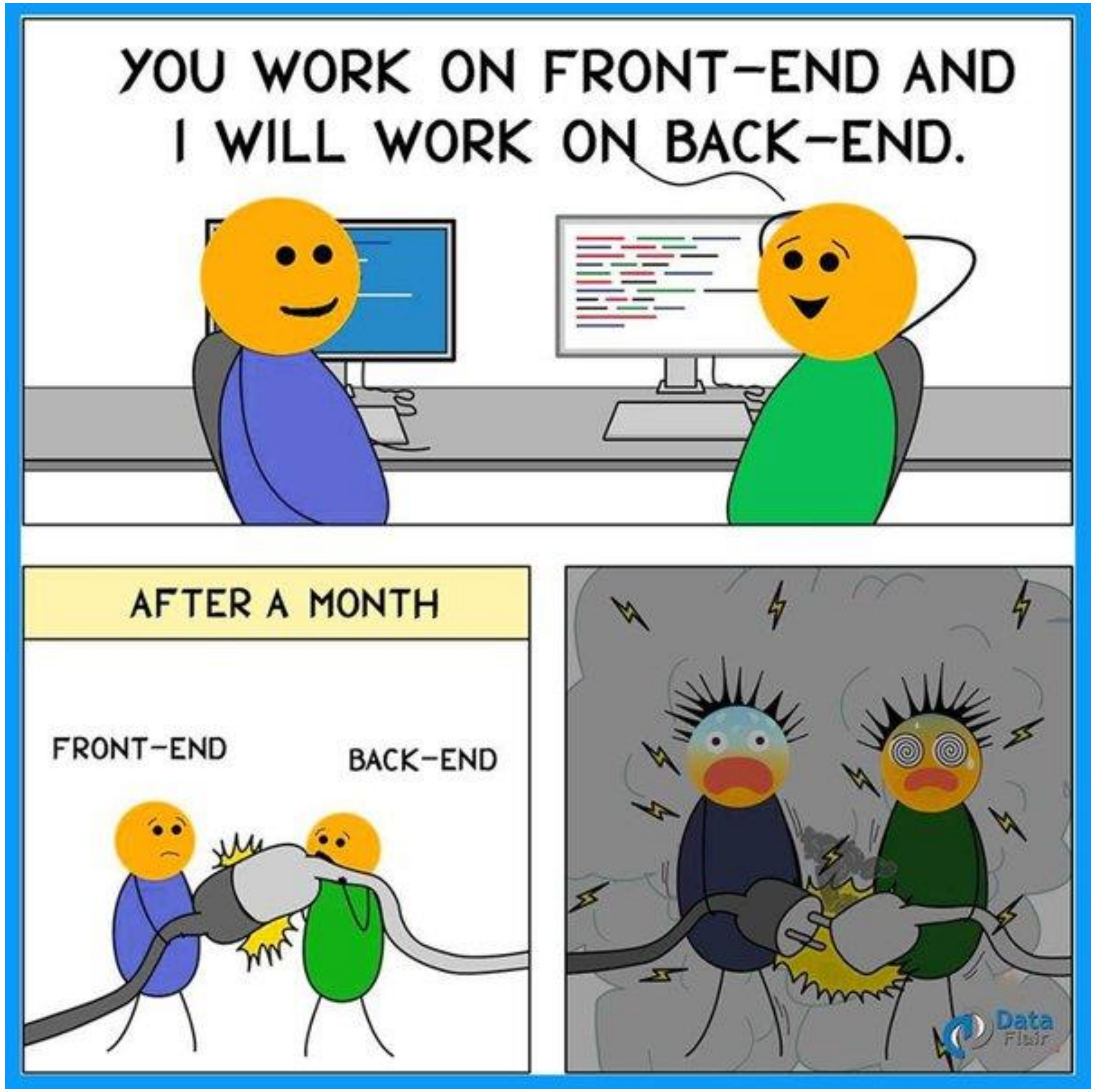
Reading data

```
const fs = require('fs');

// Create a readable stream with a smaller chunk size (e.g., 64 bytes)
const readableStream = fs.createReadStream('example.txt', { encoding: 'utf8', highWaterMark: 64 });

// Listen for the 'data' event, which indicates that a chunk of data is available
readableStream.on('data', (chunk) => {
    console.log('Received chunk of data:');
    console.log(chunk);
});

// Listen for the 'end' event, which indicates that all data has been read
readableStream.on('end', () => {
    console.log('Finished reading data from the file.');
});

// Listen for the 'error' event, in case of any errors during reading
readableStream.on('error', (err) => {
    console.error('Error reading data:', err);
});
```

Writing Data

```
const fs = require('fs');

// Create a writable stream to write data to a file
const writableStream = fs.createWriteStream('output.txt');

// Data to be written
const data =‘Hello, world’;

writableStream.write(chunk);

// End the writable stream to indicate that no more data will be written
writableStream.end();
```

```
// Listen for the 'finish' event, which indicates that all data has been written
writableStream.on('finish', () => {
    console.log('Finished writing data to the file.');
});

// Listen for the 'error' event, in case of any errors during writing
writableStream.on('error', (err) => {
    console.error('Error writing data:', err);
});
```


YOU WORK ON FRONT-END AND I WILL WORK ON BACK-END.
AFTER A MONTH
FRONT-END
BACK-END
Data Flair

Piping Streams

The process of **connecting the output of one stream to the input of another stream**. This allows you to easily transfer data from one stream to another without manually handling the data flow

```javascript
const fs = require('fs');

// Create a readable stream to read data from a source file
const readableStream = fs.createReadStream('example.txt', 'utf8');

// Create a writable stream to write data to a destination file
const writableStream = fs.createWriteStream('destination.txt');

// Pipe the data from the readable stream to the writable stream
readableStream.pipe(writableStream);

// Listen for the 'finish' event on the writable stream
writableStream.on('finish', () => {
    console.log('Data piped successfully from source to destination.');
});
```

```
// Listen for the 'error' event on the readable
stream, in case of any errors during reading
readableStream.on('error', (err) => {
    console.error('Error reading data:', err);
});

// Listen for the 'error' event on the writable
stream, in case of any errors during writing
writableStream.on('error', (err) => {
    console.error('Error writing data:', err);
});
```

Test your Knowledge

Design a Node.js server using the HTTP and FS modules to efficiently read the contents of a file ('example.txt') and stream it to another file ('example2.txt') when a client accesses the server's root URL ('/')?"

```
const http= require('http');
const fs=require('fs');

http.createServer((req,res)=>{
    if(req.url=='/'){
        readStream=fs.createReadStream('example.txt',{highWat
erMark:8});
        writeStream=fs.createWriteStream('example2.txt');
        readStream.pipe(writeStream);
        res.end('Doneeeee');
    }
}).listen(5000);
```