Event Loop

The event loop is the mechanism that allows Node.js to perform non-blocking I/O operations.

Phases:

- Timers: Executes setTimeout() and setInterval()
- Pending Callbacks: I/O callbacks deferred to the next loop
- Poll: Retrieves I/O events; executes I/O
- Check: Executes setImmediate()
- Close Callbacks: Handles close events

Example:

```
console.log('Start');
setTimeout(() => console.log('Timeout'), 0);
setImmediate(() => console.log('Immediate'));
Promise.resolve().then(() => console.log('Promise'));
console.log('End');
```

Streams

Streams are used to read or write data continuously.

Types:

- Readable: Stream to read data
- Writable: Stream to write data
- Duplex: Both readable and writable
- Transform: Modifies data

Example:

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

const readStream = fs.createReadStream('largefile.txt', 'utf8');

readStream.on('data', chunk => console.log('Reading chunk:', chunk.length));
```

Buffers

Buffers handle binary data directly in memory.

Example:

```
const buf = Buffer.from('Hello');

console.log(buf); // <Buffer 48 65 6c 6c 6f>

console.log(buf.toString()); // Hello

buf[0] = 0x68;

console.log(buf.toString()); // hello
```

Child Processes

Allows execution of external processes.

Example (Shell command):

```
const { exec } = require('child_process');

exec('ls', (err, stdout, stderr) => console.log(stdout));
```

Example (Node script):

```
const { fork } = require('child_process');

const child = fork('child.js');

child.on('message', msg => console.log('Parent got:', msg));

child.send({ hello: 'world' });

// child.js:
```

```
process.on('message', msg => {
  console.log('Child got:', msg);
  process.send({ response: 'Hello back!' });
});
```

Clustering

Used to utilize multi-core systems.

Example:

```
const cluster = require('cluster');
const http = require('http');
const os = require('os');
if (cluster.isMaster) {
  const numCPUs = os.cpus().length;
  for (let i = 0; i < numCPUs; i++) cluster.fork();
  cluster.on('exit', worker => console.log(`Worker ${worker.process.pid} died`));
} else {
  http.createServer((req, res) => res.end(`Handled by ${process.pid}`)).listen(3000);
}
```