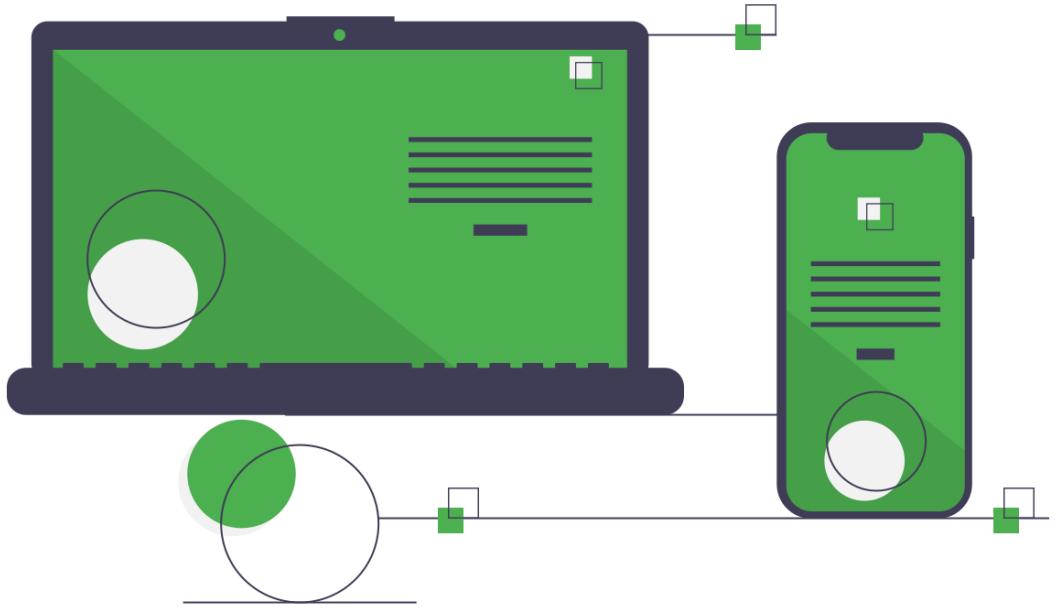

node



# Acerca de node.js

Node.js es una de las tecnologías más populares hoy en día para crear API REST escalables y eficientes. También se utiliza para construir aplicaciones móviles híbridas, aplicaciones de escritorio e incluso Internet de las cosas.

# El mundo antes de Node

# Como se hacían aplicaciones Web en los 90`s?

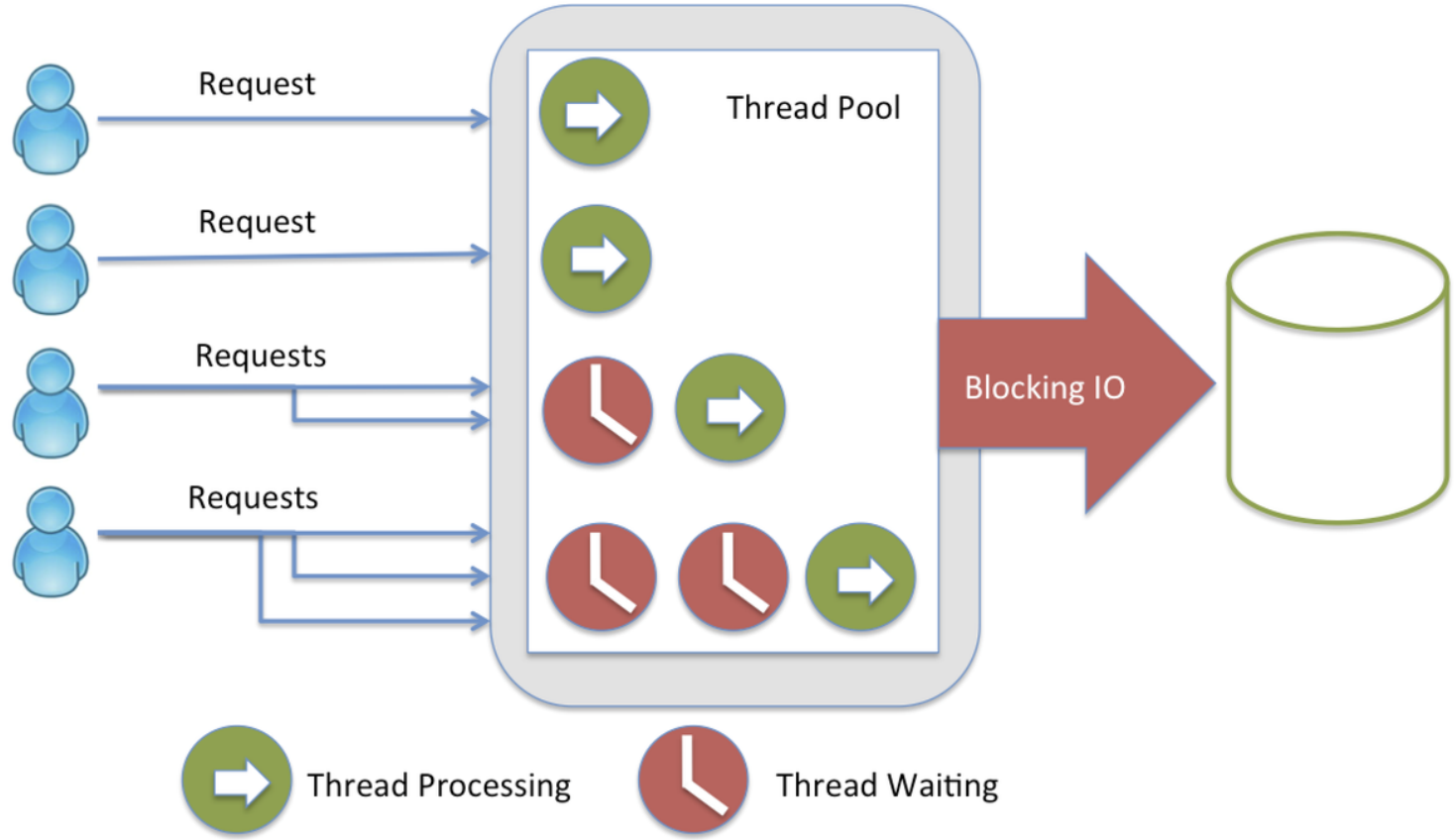
Las aplicaciones web se escribían en un modelo cliente/servidor donde el cliente solicitaba recursos y el servidor se los daba; este solo respondía cuando el cliente lo solicitaba y cerraba la conexión después de cada petición, cada petición era atendida una después de otra, cada petición era atendida por un hilo, ¿un hilo?

# Thread [hilo]

Un hilo es tiempo y recursos que la CPU da para ejecutar una pequeña unidad de instrucciones. Dicho esto, el servidor atiende múltiples solicitudes a la vez, una por subproceso (también llamado thread-per-request model modelo de subproceso por solicitud)

# Servidor multiproceso

## Multi Threaded Server



Para atender N solicitudes de una vez, el servidor necesita N subprocesos. Si el servidor recibe la solicitud N + 1, debe esperar hasta que alguno de esos N subprocesos esté disponible.

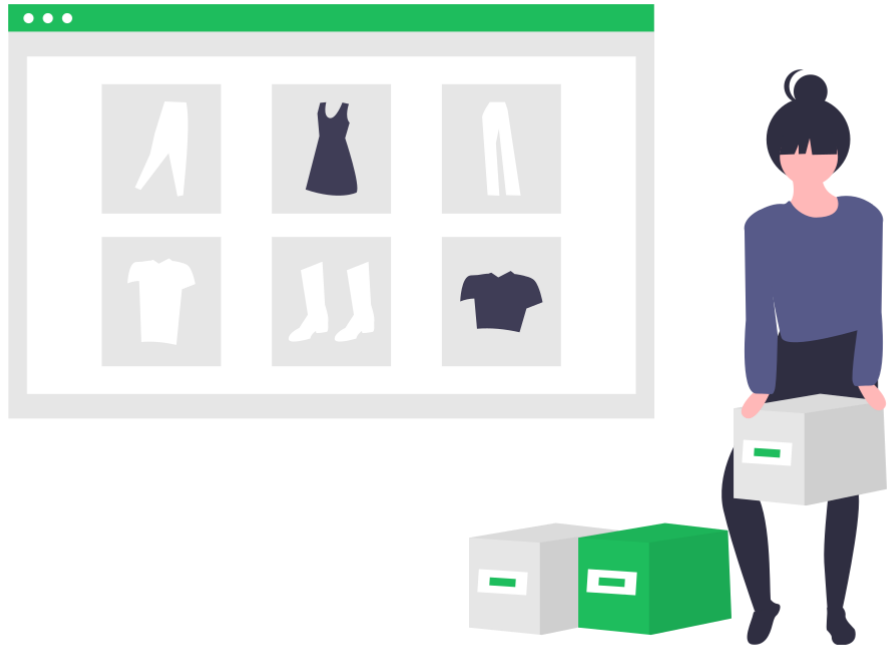

Una forma de resolver esta limitación es agregar más recursos (memoria, núcleos de CPU, etc.) al servidor, pero tal vez no sea una buena idea en absoluto ...
Y, por supuesto, habrá limitaciones tecnológicas.

# Blocking I/O

## [Bloqueo de E/S]

Por que un hilo no puede atender dos o mas peticiones a la vez?

Por el bloqueo de las operaciones de entrada y salida



Supongamos que está desarrollando una tienda en línea y necesita una página donde el usuario pueda ver todos sus productos.

El usuario accede a http://yourstore.com/products y el servidor representa un archivo HTML con todos sus productos desde la base de datos. Bastante simple ¿verdad?

Pero, ¿qué pasa detrás?

1. Cuando el usuario accede a /productos, debe ejecutarse un método o función específicos para atender la solicitud, por lo que un pequeño fragmento de código (tal vez el suyo o de un framework) analiza la URL solicitada y busca el método o la función correctos. El hilo está funcionando. ✓
2. Se ejecuta el método o la función, así como las primeras líneas. El hilo está funcionando. ✓
3. Suponemos que se guardan todos los registros del sistema en un archivo y para asegurarse de que la ruta está ejecutando el método/función correctos, debe registrar un "Método X en ejecución" en un String, esa es una operación de bloqueo de E/S. El hilo está esperando. ✗
4. El registro se guarda y se ejecutan las siguientes líneas. El hilo está funcionando de nuevo. ✓
5. Es hora de ir a la base de datos y obtener todos los productos, una consulta simple como SELECT * FROM productos hace el trabajo pero esa es una operación de bloqueo de E/S. El hilo está esperando. ✗
6. Obtiene una matriz o una lista de todos los productos y los registra. El hilo está esperando. ✗
7. Con esos productos es hora de renderizar una plantilla, pero antes de renderizarla debe leerla primero. El hilo está esperando. ✗
8. El motor de plantillas hace su trabajo y la respuesta se envía al cliente. El hilo está funcionando de nuevo. ✓
9. El hilo es libre.

# ¿Cuan lentas son las operaciones I/O?

| Operación | Nº de ticks en el CPU |
|---|---|
| Registro del CPU | 3 ticks |
| L1 cache | 8 ticks |
| L2 cache | 12 ticks |
| RAM | 150 ticks |
| Disk | 30,000,000 ticks |
| Network | 250,000,000 ticks |

En resumen las operaciones E/S hacen perder tiempo y malgastar recursos a los hilos.

# El problema C10K

Con el pasar del tiempo los servidores eran cada vez mas exigidos y el reto para finales de los 2000`s era resolver el problema C10K que consistía en logran que 10,000 clientes logren conectarse a una sola maquina servidor, el modelo de subproceso por solicitud no podía resolver el problema debido a que:

Las implementaciones de subprocesos nativos asignan aproximadamente 1 MB de memoria por subproceso, por lo que 10k subprocesos requieren 10 GB de RAM solo para la pila de subprocesos y recuerde que estamos a principios de la década de 2000.

Actualmente el problema es resuelto con facilidad atreves de algoritmos o Frameworks, además que los servidores multiplicaron exponencialmente sus recursos a día de hoy, por lo que el problema evoluciono al problema C10M en el que se quiere lograr 10,000,000 de conexiones simultaneas en un mismo servidor.

# JS al rescate

Node.js resuelve el problema C10K ... pero ¿por qué?

El lado del servidor Javascript no era nuevo a principios de la década de 2000, hubo algunas implementaciones en la parte superior de la máquina virtual Java, como RingoJS y AppEngineJS, basadas en el modelo de “subprocesos por solicitud”.

Pero si eso no resolvió el problema de C10K, ¿por qué lo hizo Node.js? Bueno, es porque Javascript tiene un solo subproceso.

# Que es Node.js?

Node.js es una plataforma del lado del servidor construida en el motor Javascript de Google Chrome (motor V8) que compila el código Javascript en código máquina.

Node.js utiliza un modelo de E/S sin bloqueo controlado por eventos que lo hace ligero y eficiente. No es un Framework, no es una Biblioteca, es un entorno de tiempo de ejecución.

# Non-blocking I/O

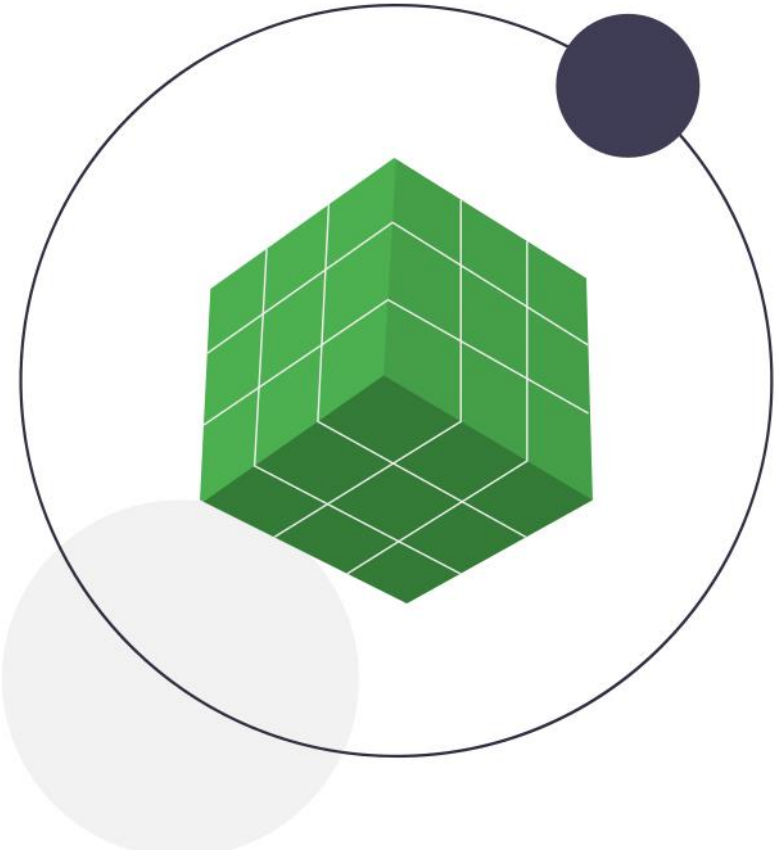
Node.js no tiene bloqueo E/S, lo que significa:

1. El subproceso principal no se bloqueará en las operaciones de E/S.
2. El servidor seguirá atendiendo las solicitudes.
3. Trabajaremos con código asincrónico.

# The event loop

## [el bucle de eventos]

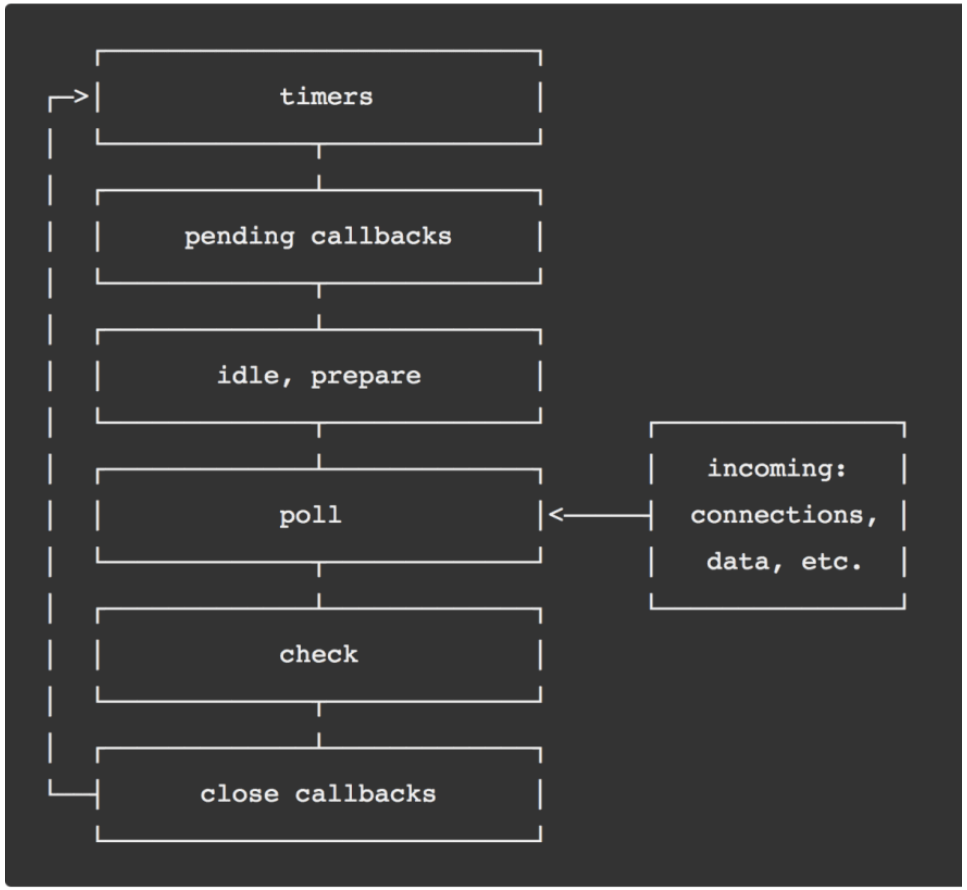
Es un bucle infinito y es el único subproceso disponible en Node.js.
Libuv es una biblioteca en C que implementa este patrón y es parte de los módulos principales de Node.js.
El Event Loop tiene seis fases, la ejecución de todas las fases se llama tick.



# The event phases

## [Fases del bucle de eventos]

- temporizadores: esta fase ejecuta devoluciones de llamada programadas por setTimeout() y setInterval().
- devoluciones de llamada pendientes: ejecuta casi todas las devoluciones de llamada con la excepción de las devoluciones de llamada cercanas, las programadas por temporizadores y setImmediate().
- inactivo, preparado: solo se usa internamente.
- encuesta: recuperar nuevos eventos de E/S; El nodo se bloqueará aquí cuando sea apropiado.
- check: las devoluciones de llamada setImmediate() se invocan aquí.
- cerrar devoluciones de llamada: finalizan las llamadas.



# Qué pasa si el event -loop necesita ejecutar un proceso E/S?

Utiliza un subproceso del sistema operativo de un grupo (a través de la biblioteca libuv) y cuando se realiza el trabajo, la devolución de llamada se pone en cola para ejecutarse en la fase de devoluciones de llamada pendientes.

# Worker Threads

Los Worker Threads son útiles para realizar operaciones JavaScript intensivas en CPU; no recomendadas para E/S, ya que los mecanismos integrados de Node.js para realizar operaciones de forma asincrónica ya lo tratan de manera más eficiente que los subprocesos de trabajo.