Performance y Escalamiento de Node

Fernando Florenzano
Matias Junemann

IIC3585 - Diseño Avanzado de Aplicaciones Web

Performance

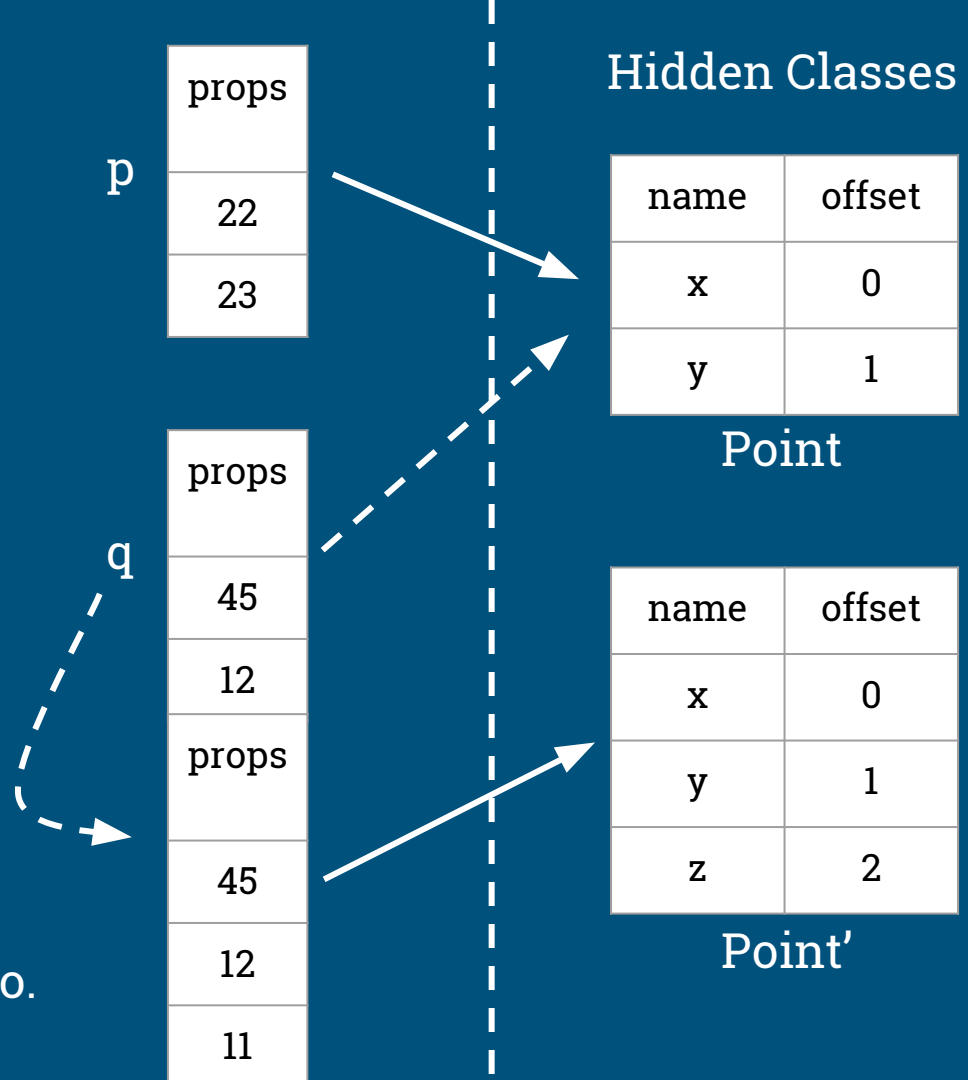
Just-in-time compilation utilizando V8

JIT de V8

- Produce código de máquina más eficiente desde código JavaScript.
- Lo genera en tiempo de ejecución.
- No genera código intermedio o byte-code.

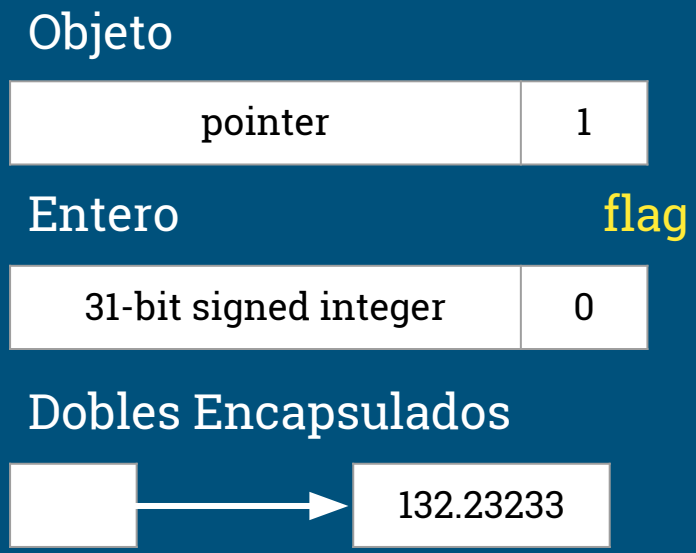
```
function Point(x,y){
    this.x = x;
    this.y = y:
}
var p = new Point(22,23);
var q = new Point(45,12);

q.z = 11;
```



Hidden Classes

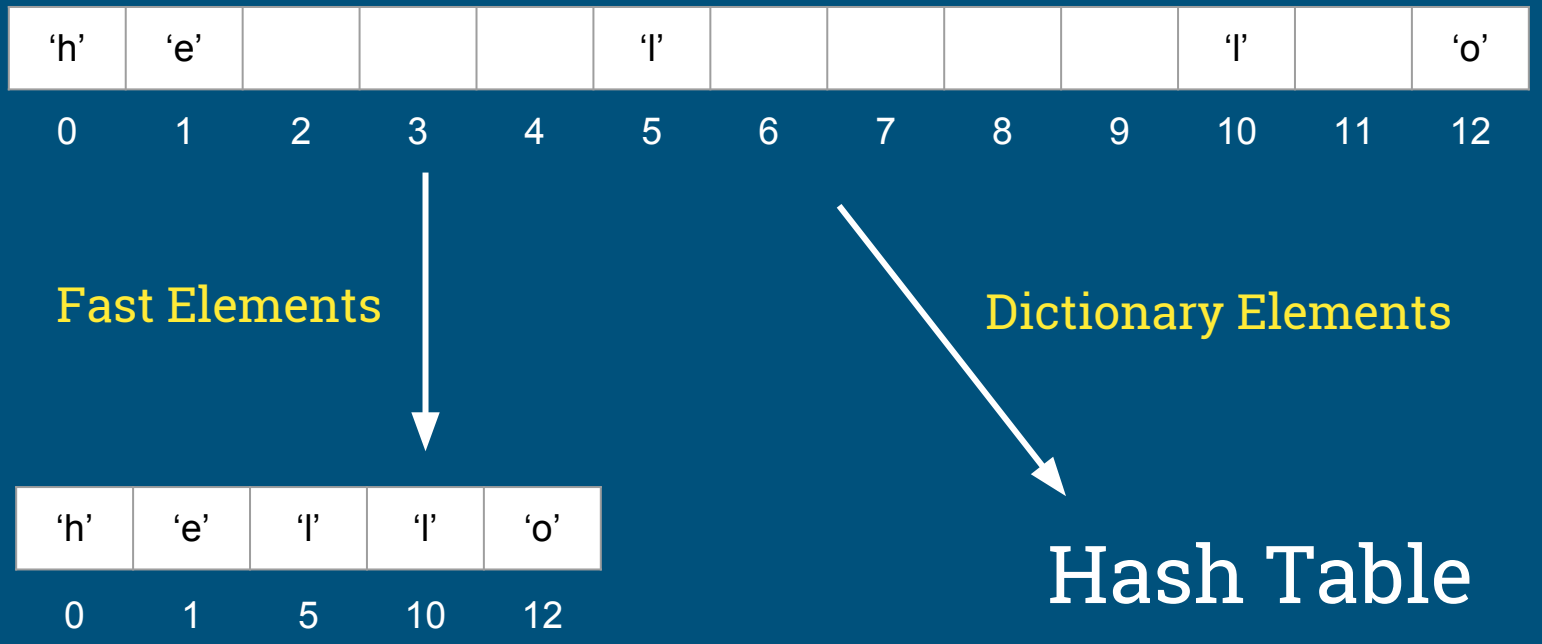
Alternativa a objetos diccionario.



Representando objetos de JS con 32-bits

Tagged Values

Representación eficiente de valores.



Fast Elements and Dictionary Elements

Manejo de Arreglos grandes o esparcidos.

tips!

- Inicializar completamente objetos con constructores.
- Siempre inicializar en el mismo orden.
- Funciones con argumentos siempre utilizarlos con el mismo tipo de argumentos, utilizará la misma Hidden Class.
- Preferir enteros de 31-bits.
- Usar índices contiguos desde 0 en arreglos.
- No crear arreglos gigantes de un golpe, hazlo crecer a medida que es necesario.
- No eliminar elementos de arreglos.
- No cargar elementos no inicializados de arreglos.

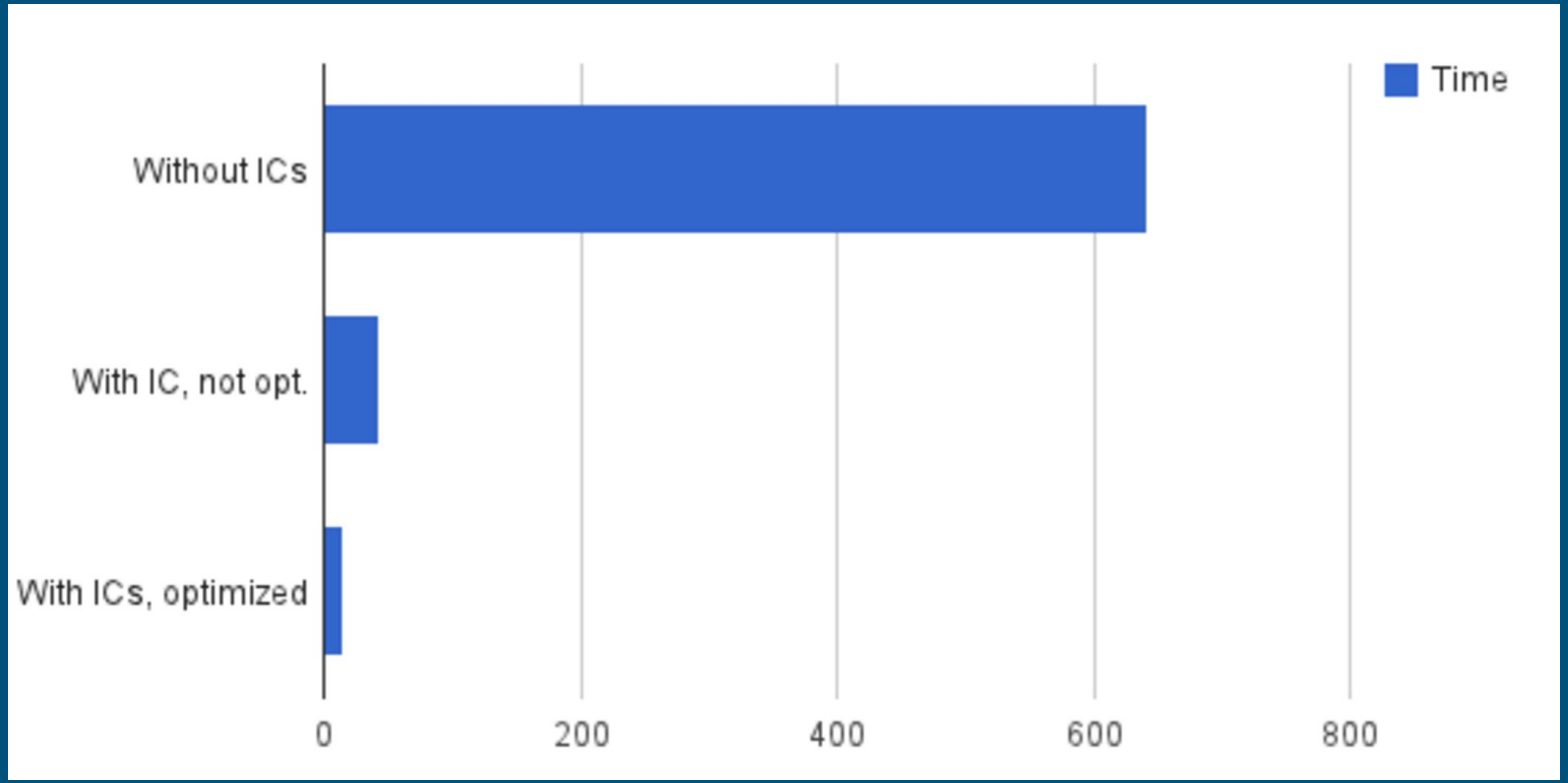
2 compiladores!

Compilador “Full”

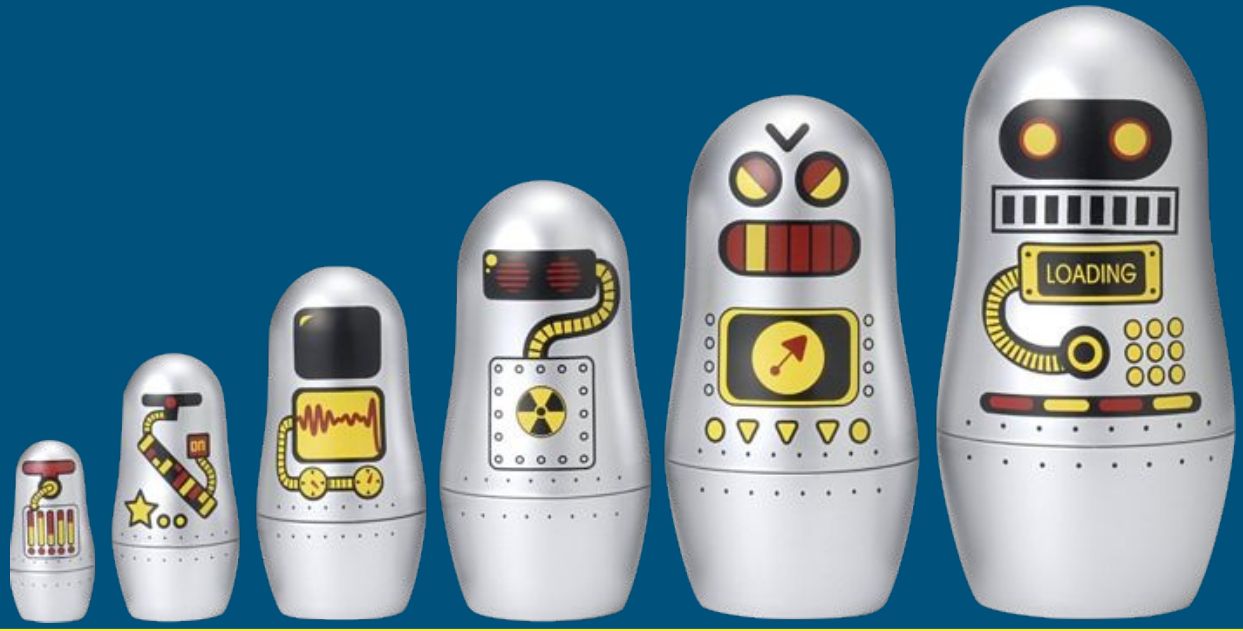
- Produce codigo rápido.
- No toma en cuenta tipos.
- Utiliza Caches Inline (IC) para aprender sobre tipos a medida que ejecuta.
- ICs maneja funciones dependientes de tipo eficientemente.
- Válida suposiciones y luego ejecuta.

Compilador Optimizador

- Re-compila funciones.
- Utiliza los IC para tomar decisiones para optimizar porciones de código.
- No soporta todos los aspectos del lenguaje como **try-catch**.
- Encapsulamiento en funciones permite optimizar secciones no soportados.



Speedup de V8



Escalando Node

Escalando Node

Node.js es single-threaded y usa I/O no-bloqueante, lo que lo permite escalar y soportar decenas de miles de operaciones concurrentes.

So, what could possibly go wrong ?

- No trabaja tan bien sirviendo contenido estático.
- No es tan bueno distribuyendo carga a múltiples servidores.
- Es single-threaded.
- Se generan cuellos de botella.

Escalando Node

- Queremos que el software sea capaz de adaptarse para alcanzar nuevos requisitos de tamaño y alcance para nuevos usuarios.
- Podemos escalar la aplicación de forma: Horizontal o Vertical

Escalamiento Vertical

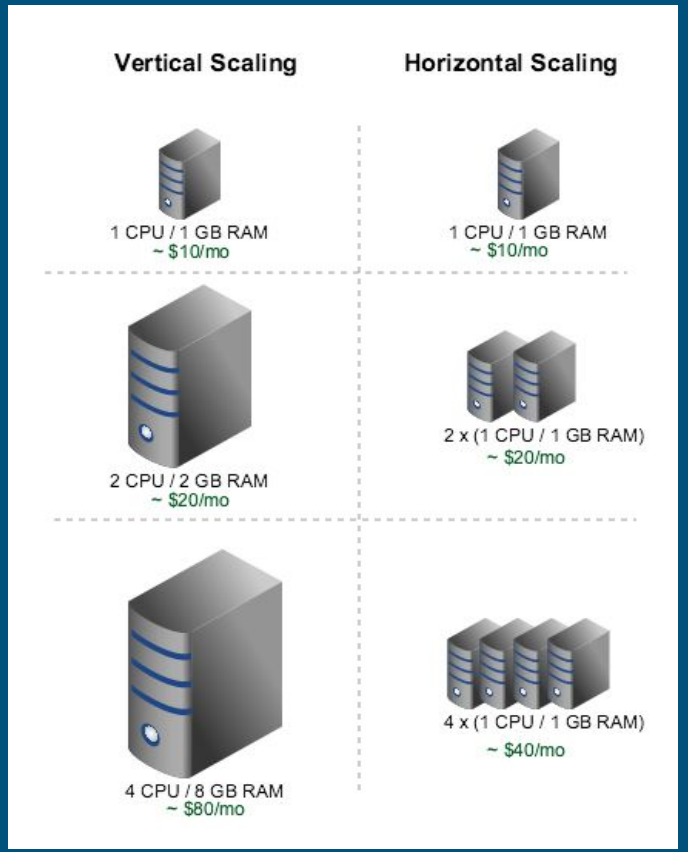
- Mejorar del Hardware
- Static Server
- Reverse Proxy
- Aumentar instancias de la Aplicación

Escalamiento Vertical - Problemas

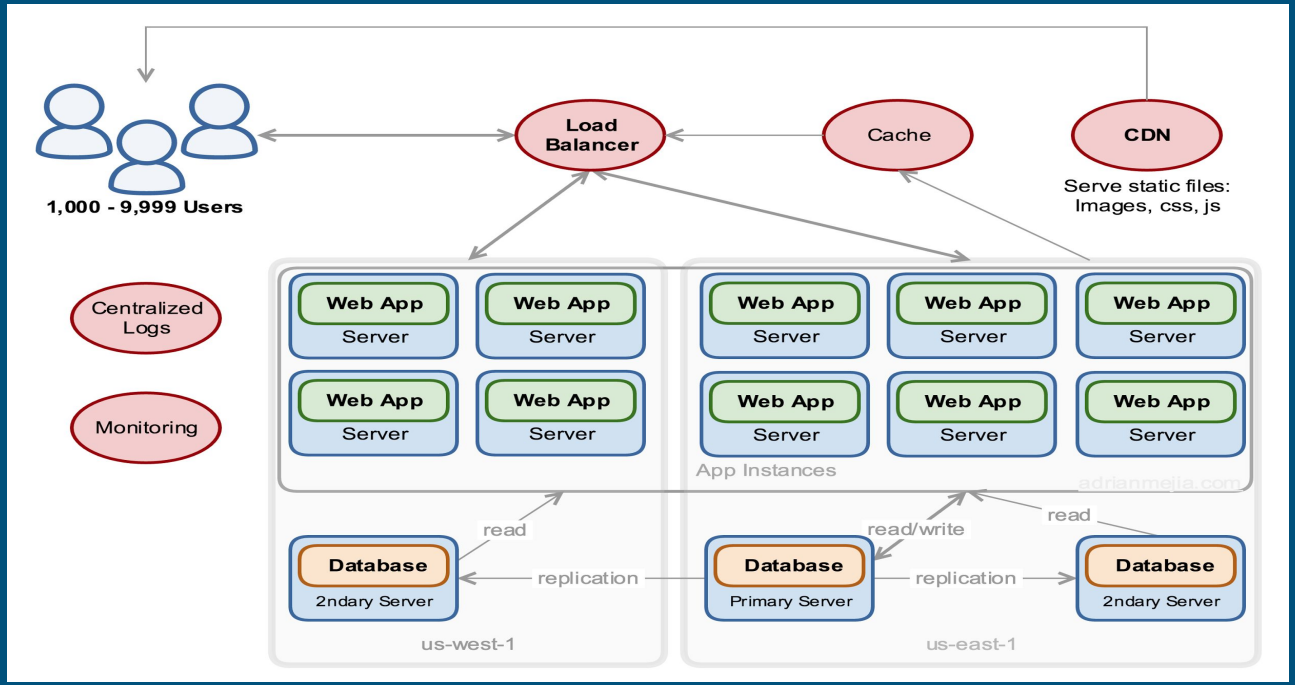
- Disponibilidad
- Centralizado
- Cuello de botella en I/O

Escalamiento Horizontal

- Replicación de las DB’s para manejar el cuello de botella causado por I/O.
- Uso Load Balancers (Nginx, HAProxy).
- Process Management (PM2, StroonLoop, Forever)



Escalamiento Horizontal



Referencias para profundizar

- http://thibaultlaurens.github.io/javascript/2013/04/29/how-the-v8-engine-works/
- http://v8-io12.appspot.com/#1
- http://adrianmejia.com/blog/2016/03/23/how-to-scale-a-nodejs-app-based-on-number-of-users/
- https://www.keithcirkel.co.uk/load-balancing-node-js/