Recolección de Basura en D

Leandro Lucarella

Facultad de Ingeniería, UBA

Diciembre 2010

Motivación

- Recolección de basura
- Lenguaje de programación D
- Utilidad $\rightarrow$ Software Libre $\rightarrow$ Contribución

Introducción

¿Qué?

- Administración automática de memoria

¿Para qué?

- Simplificar interfaces
- Mejorar eficiencia (!)
- Evitar errores de memoria
  - *Dangling pointers*
  - *Memory leaks*
  - *Double free*

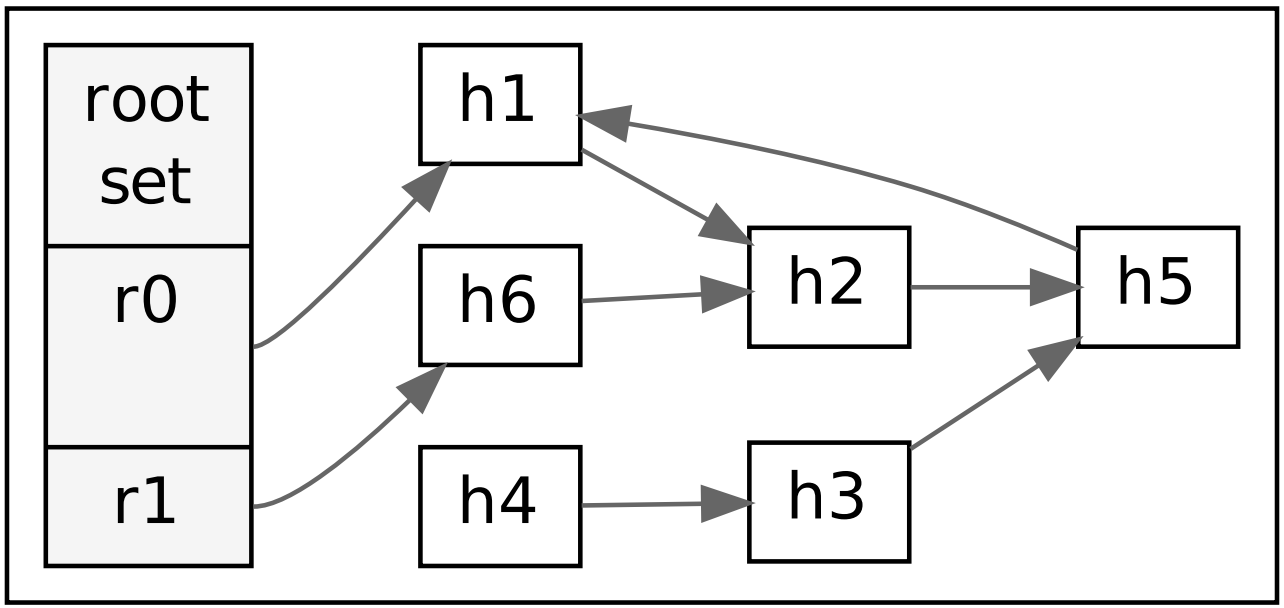
¿Cómo?

Algoritmos Clásicos

- Conteo de referencias
- Copia de semi-espacio
- Marcado y barrido

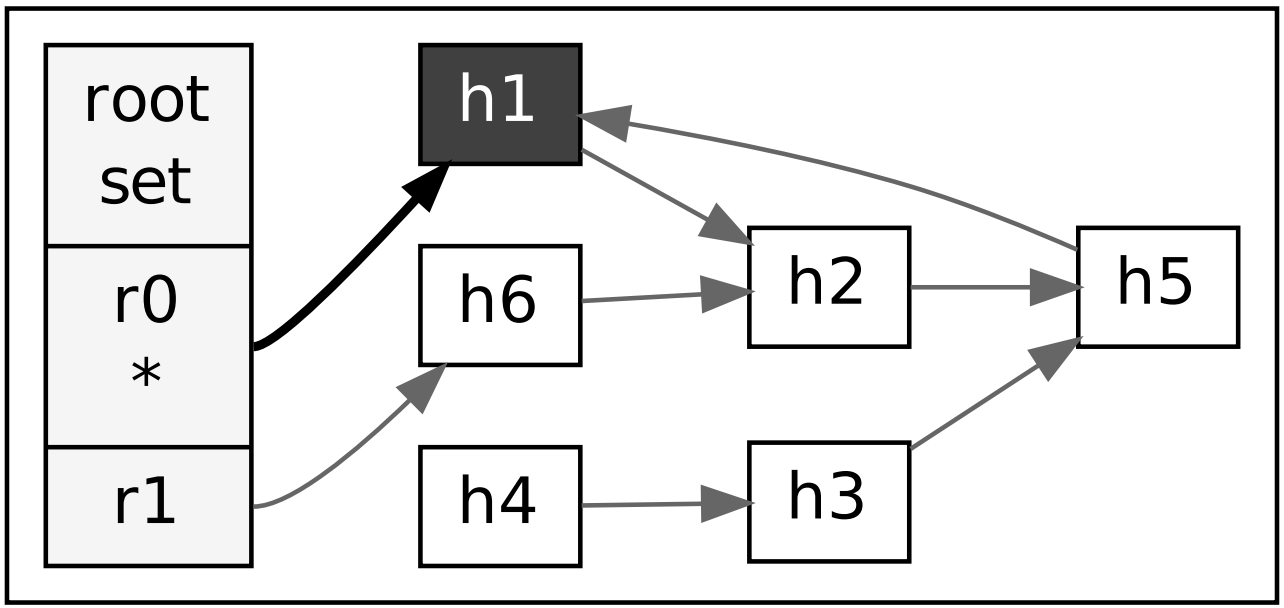
Algoritmos Clásicos

- Conteo de referencias
- Copia de semi-espacio
- Marcado y barrido



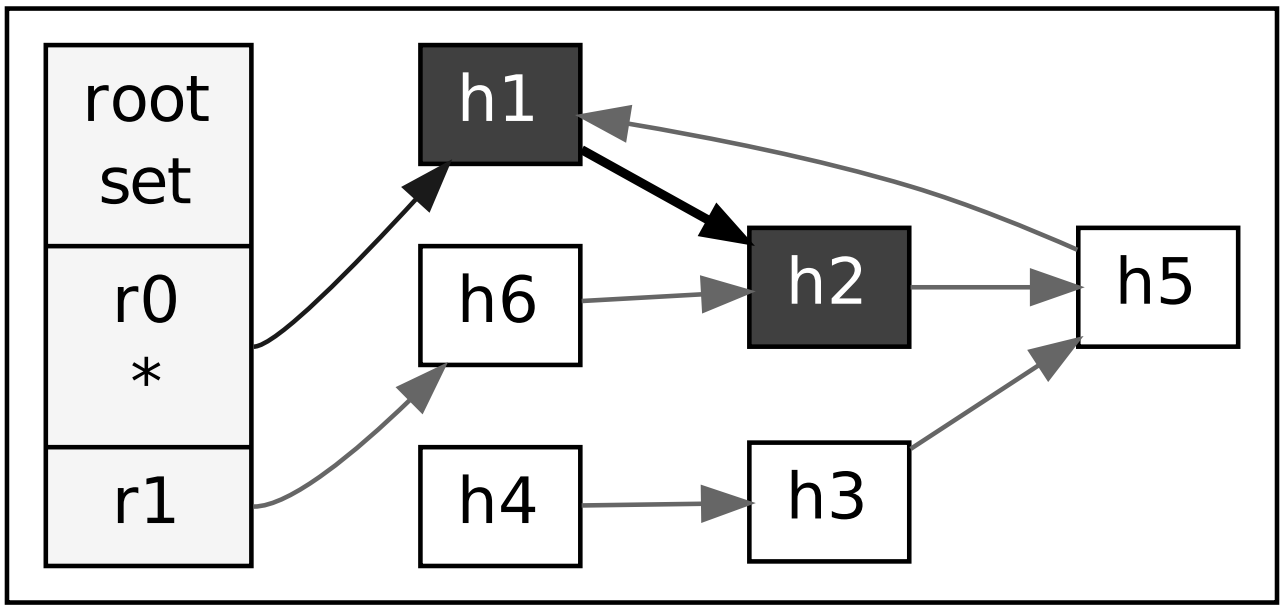
Algoritmos Clásicos

- Conteo de referencias
- Copia de semi-espacio
- Marcado y barrido



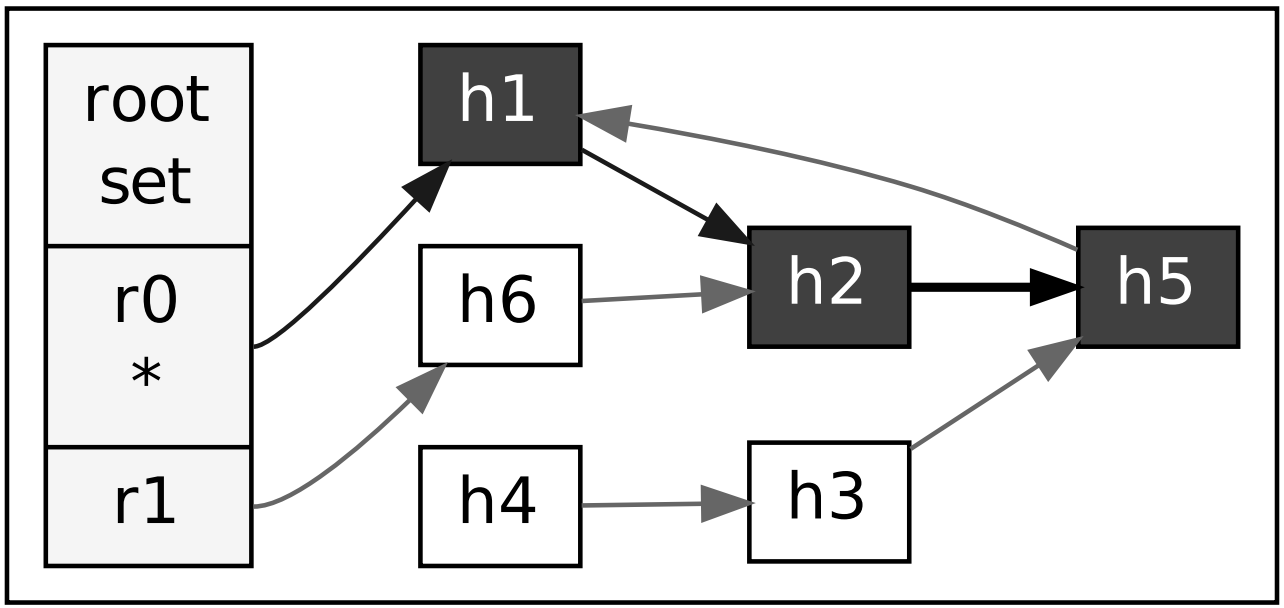
Algoritmos Clásicos

- Conteo de referencias
- Copia de semi-espacio
- Marcado y barrido



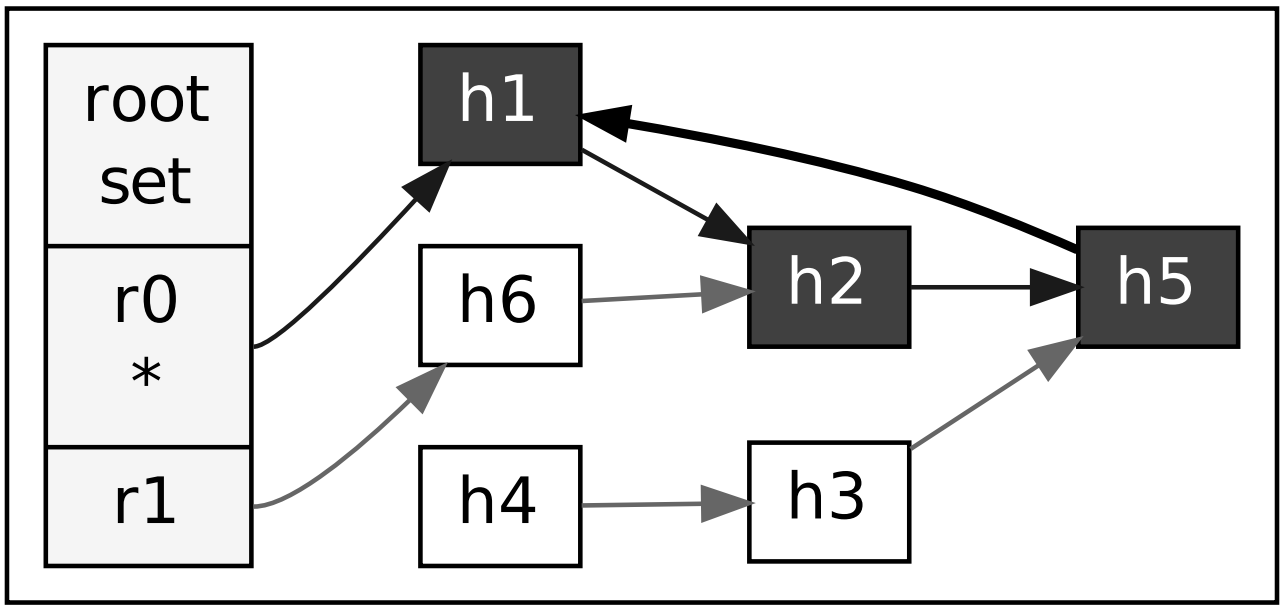
Algoritmos Clásicos

- Conteo de referencias
- Copia de semi-espacio
- Marcado y barrido



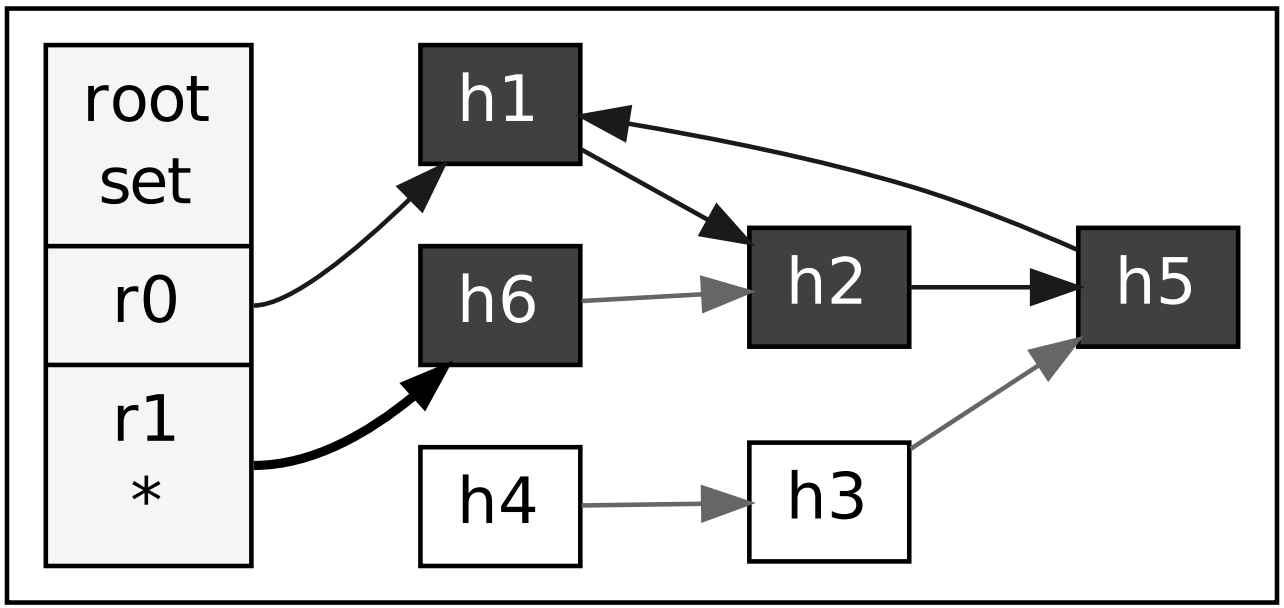
Algoritmos Clásicos

- Conteo de referencias
- Copia de semi-espacio
- Marcado y barrido



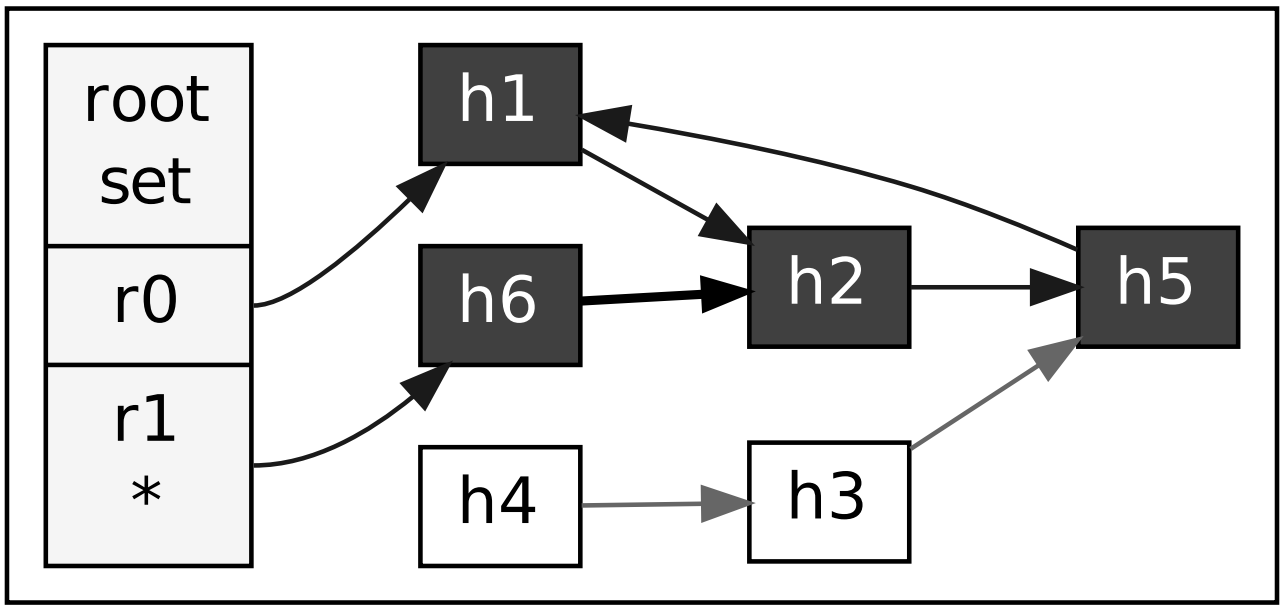
Algoritmos Clásicos

- Conteo de referencias
- Copia de semi-espacio
- Marcado y barrido



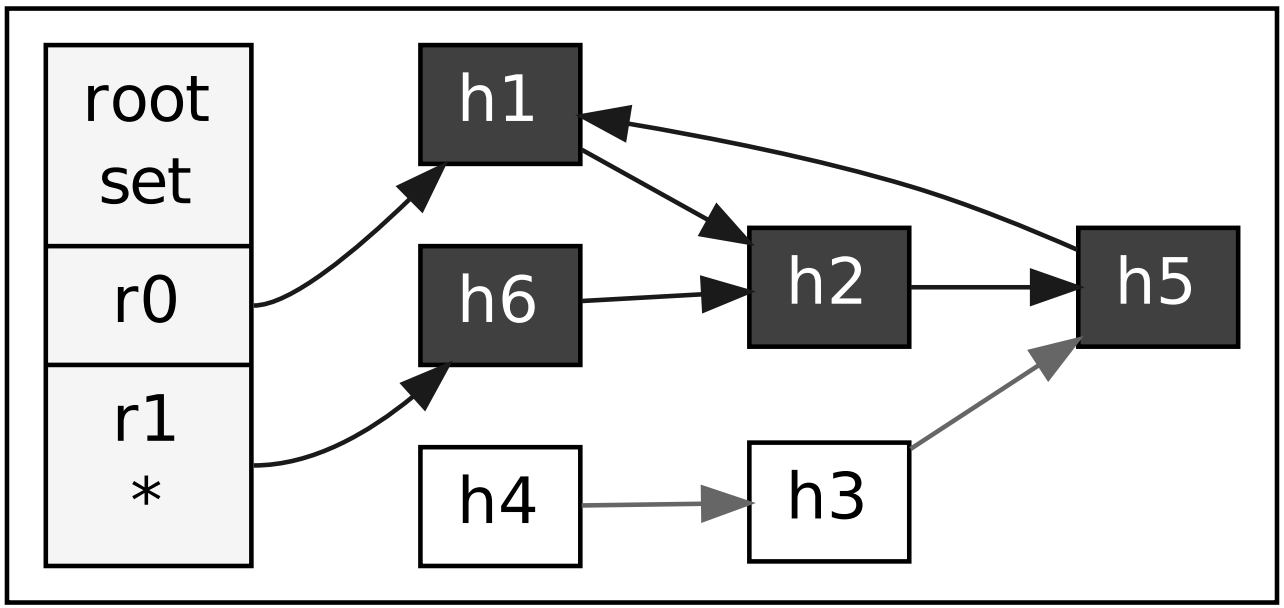
Algoritmos Clásicos

- Conteo de referencias
- Copia de semi-espacio
- Marcado y barrido



Algoritmos Clásicos

- Conteo de referencias
- Copia de semi-espacio
- Marcado y barrido



Estado del Arte

- Medio siglo de investigación y desarrollo (3000+ publicaciones)
- Objetivo
  - ↓ Tiempo total de ejecución
  - ↓ Cantidad de recolecciones
  - ↓ Tiempo de recolección
  - ↓ Tiempo (máximo) de pausa
- Técnicas
  - Particiones
  - Concurrencia
  - Organización de memoria
  - Precisión
  - Análisis estático

Características Generales

- Sintaxis tipo C/C++
- Compilado
- Sistema de tipos estático
- Multi-paradigma

Paradigmas

- Programación de bajo nivel (*system-programming*) ← C/C++
  - `asm`
  - `union`
  - `extern (C)`
  - `malloc()`

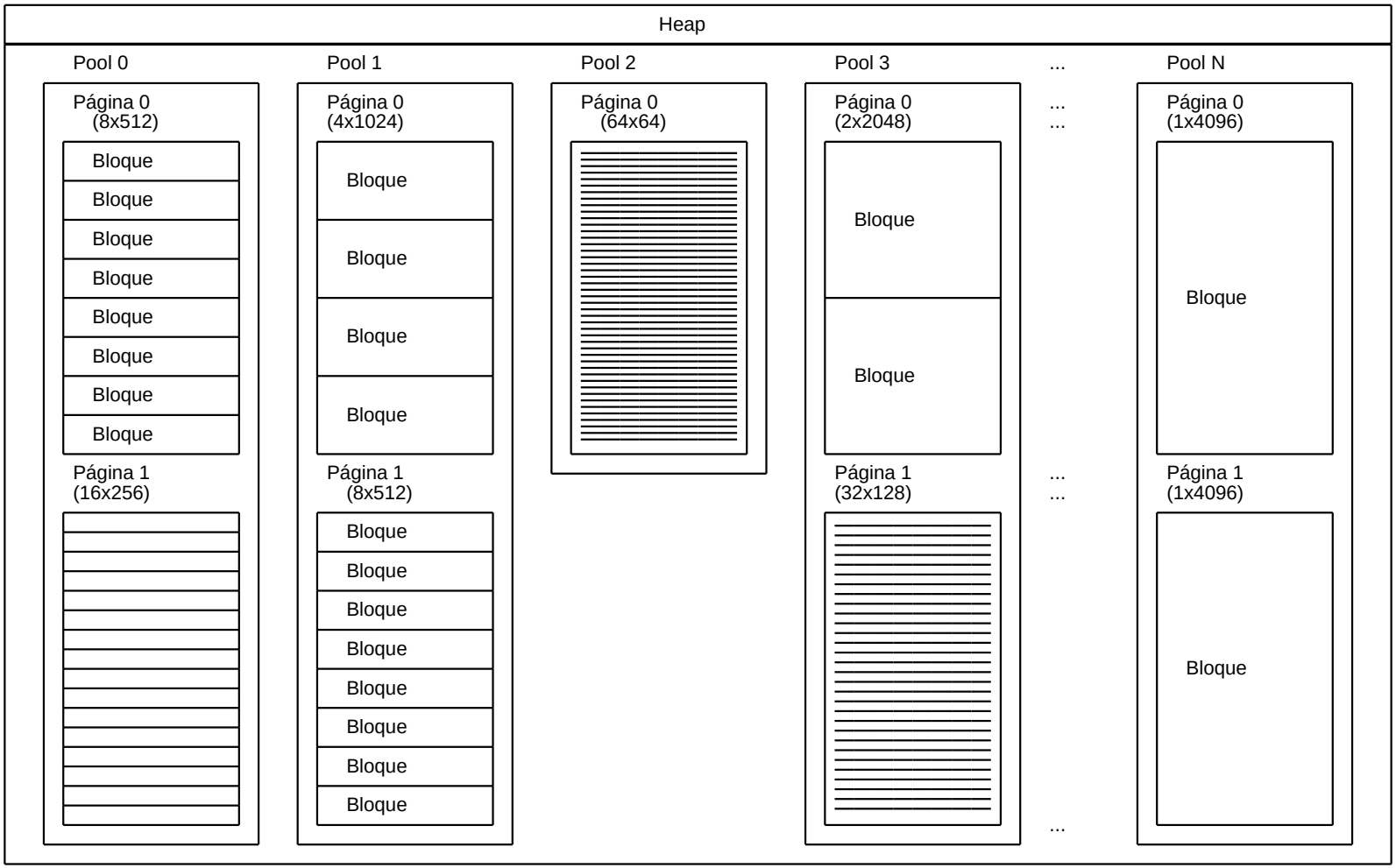
  → Conservativo + Manipulación de *root set*
- Programación de alto nivel ← Python/Ruby/Perl
  - *GC*
  - `T[], T[K]`

  → Punteros interiores
- Orientación a objetos ← Java
  - `~this()`

  → Finalización

Organización del Heap

Heap → *Pools* → Páginas → Bloques + Listas de libres



Bloques

- Tamaño fijo (por página)
  - Potencias de 2
  - De 16 a 4096 bytes
  - Más de 4096 (una página)
    - Objeto grande
    - Múltiplo de páginas: 4096, 8192, ...
    - En páginas contiguas (y mismo *pool*)
- Indicadores (*bit sets* en *pool*)
  - Marcado
    - *mark*
    - *scan*
    - *noscan*
  - Barrido
    - *free*
    - *finals*

Algoritmo

- Marcado y barrido
  - Marcado iterativo
- Conservativo
  - Con una pizca de *precisión* (`NO_SCAN`)
- *Stop-the-world*
  - Durante el marcado, en teoría
- *Lock* global
  - Muy propenso a extender el tiempo de *stop-the-world* en la práctica

Lo Bueno

- Anda :)
- Organización del *heap* (*two-level allocation*)
- Marcado iterativo (!*overflow*)
- *Bit set* para indicadores (caché)

(bueno != perfecto)

Lo Malo y lo Feo

Lo malo

- $\downarrow$ Configurabilidad (*no silver bullet*)
- $\downarrow$ Precisión (información de tipos) $\rightarrow$ Memoria inmortal
- $\downarrow$ Concurrencia $\rightarrow$ Grandes pausas
- $\downarrow$ Control sobre el factor de ocupación del *heap* $\rightarrow$ casos patológicos

Lo feo

- El código (complejo, intrincado, duplicado, poco documentado) $\rightarrow$ Difícil de mantener, modificar y mejorar

fork(2)

- Hijo *nace* con una *fotografía* de la memoria del padre
- Aisla modificaciones en la memoria de padre e hijo
- Minimiza copia efectiva de memoria (*COW*)
- Comienza con un solo hilo (el que llamó a `fork(2)`)
- Muy eficiente

Algoritmo Principal

- Basado en el trabajo de Gustavo Rodriguez-Rivera y Vince Russo (*Non-intrusive Cloning Garbage Collector with Stock Operating System Support*)
- Minimiza tiempo de pausa realizando fase de marcado concurrente vía `fork(2)`
- Proceso padre sigue corriendo el programa
- Proceso hijo realiza fase de marcado
- Se comunican resultados vía memoria compartida
- Sincronización mínima (`fork(2)` + `waitpid(2)`)

Problemas

- Hilo que disparó la recolección bloqueado hasta fin de recolección completa (marcado concurrente inclusive)
- Otros hilos potencialmente bloqueados durante toda la recolección también (*lock* global)

$\rightarrow$ Tiempo de pausa en la práctica ~= tiempo total de recolección

Eager Allocation

- Crea un nuevo *pool* de memoria antes de lanzar el marcado concurrente
- Devuelve memoria del nuevo *pool* al programa mientras termina el marcado concurrente
- Permite al programa (todos sus hilos) seguir trabajando mientras se realiza el marcado concurrente
- Compromiso
  ↑ Consumo de memoria
  ↓ Tiempo de pausa real

Early Collection

- Dispara una recolección *preventiva* antes de que se agote la memoria
- Permite al programa (todos sus hilos) seguir trabajando mientras la recolección *preventiva* está en progreso
- Si se agota la memoria antes de que la recolección *preventiva* finalice, se vuelve a bloquear
- Combinable con *eager allocation* para evitar bloquear
- Pueden realizarse más recolecciones de las necesarias
- Compromiso
  ↑ Consumo de procesador (potencialmente)
  ↓ Tiempo de pausa real (no garantizado)

Precisión

Adaptación del trabajo de Vincent Lang y David Simcha:

- Compilador genera información sobre ubicación de los punteros para cada tipo de dato
  - Indica si una *palabra* debe ser escaneada
  - Indica si una palabra es un puntero
- Se pasa esa información al recolector al momento de pedir memoria
- Recolector original utiliza esa información
  - Almacena un puntero a la información al final del bloque
  - Utiliza la información para escanear solo palabras que son punteros (con seguridad o potencialmente)

Optimizaciones y Otras Mejoras Menores

- Mejora del factor de ocupación del *heap*
- Caché de consultas críticas para acelerar cuellos de botella
- Reestructuración, modularización, simplificación y limpieza del código
- Pre-asignación de memoria
- Optimizaciones algorítmicas sobre búsquedas frecuentes
- Registro de pedidos de memoria y recolecciones realizadas

Configurabilidad

- Configurable en *tiempo de arranque*
- Vía variable de entorno (`D_GC_OPTS=fork=0 ./prog`)
- Viejas opciones convertidas
  - `mem_stop`
  - `sentinel`
- Nuevas opciones
  - `pre_alloc`
  - `min_free`
  - `malloc_stats_file`
  - `collect_stats_file`
  - `conservative`
  - `fork`
  - `eager_alloc`
  - `early_collect`

Generalidades

- Múltiples corridas (20-50)
  - Minimizar error en la medición
  - Resultados expresados en función de:
    - Mínimo
    - Media
    - Máximo
    - Desvío estándar
- Minimizar variación entre corridas
  - `cpufreq-set(1)`
  - `nice(1)`
  - `ionice(1)`
- 4 *cores*

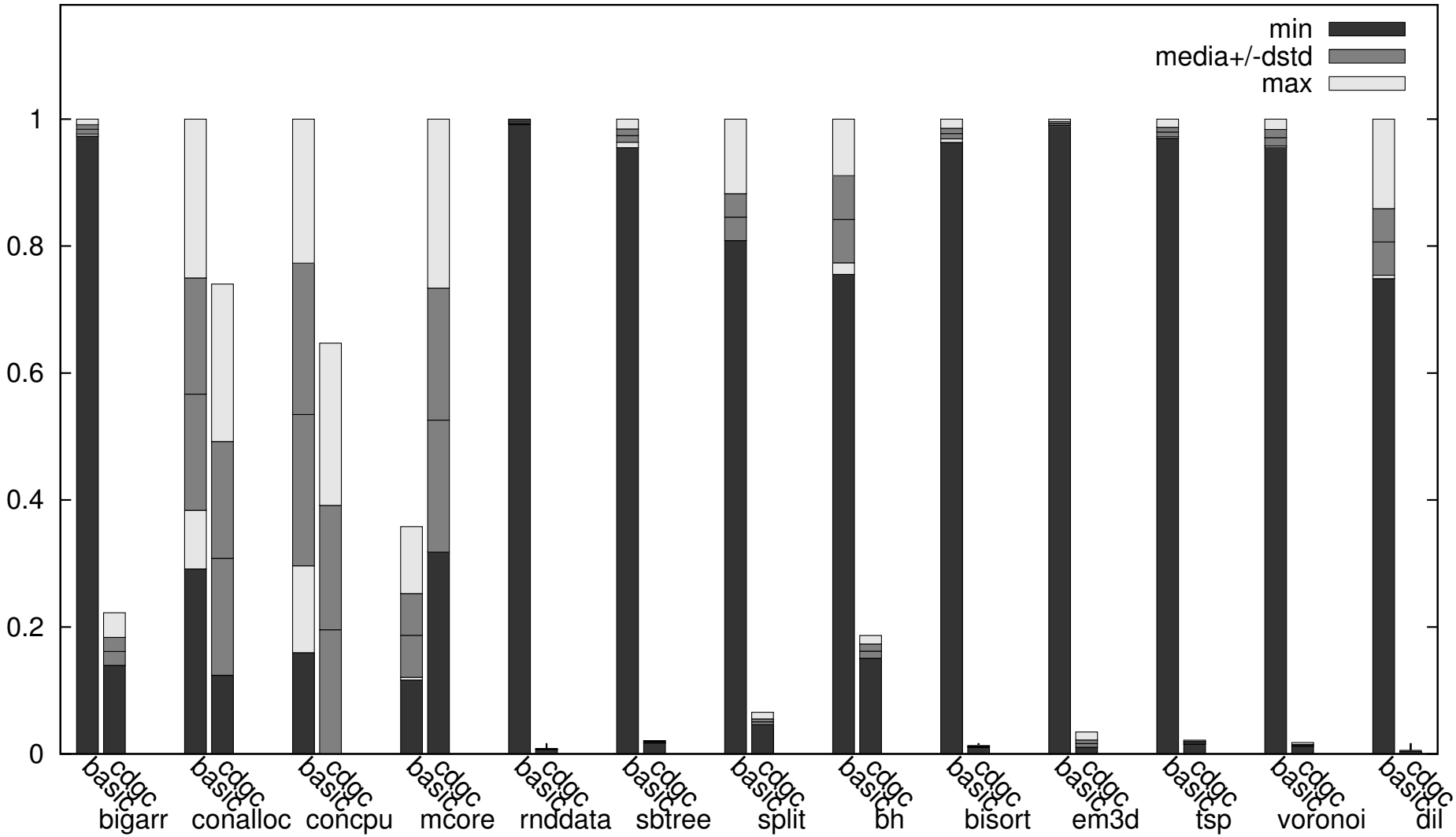
Programas

- Triviales (7)
  - Ejercitar aspectos puntuales
  - No realizan una tarea útil
  - Casos patológicos
- Programas pequeños - *Olden Benchmark* (5)
  - Relativamente pequeños (400-1000 *SLOC*)
  - Realizan una tarea útil
  - Manipulan mucho listas y árboles asignando mucha memoria
  - No son ideales para probar un *GC*
- Programas reales - Dil (1)
  - Compilador de D escrito en D
  - Grande y complejo (32K+ *SLOC*, 86 módulos, 300+ *clases*)
  - Programado sin (limitaciones ni ventajas del) *GC* en mente
  - Manipulación de *strings*, arreglos dinámicos y asociativos

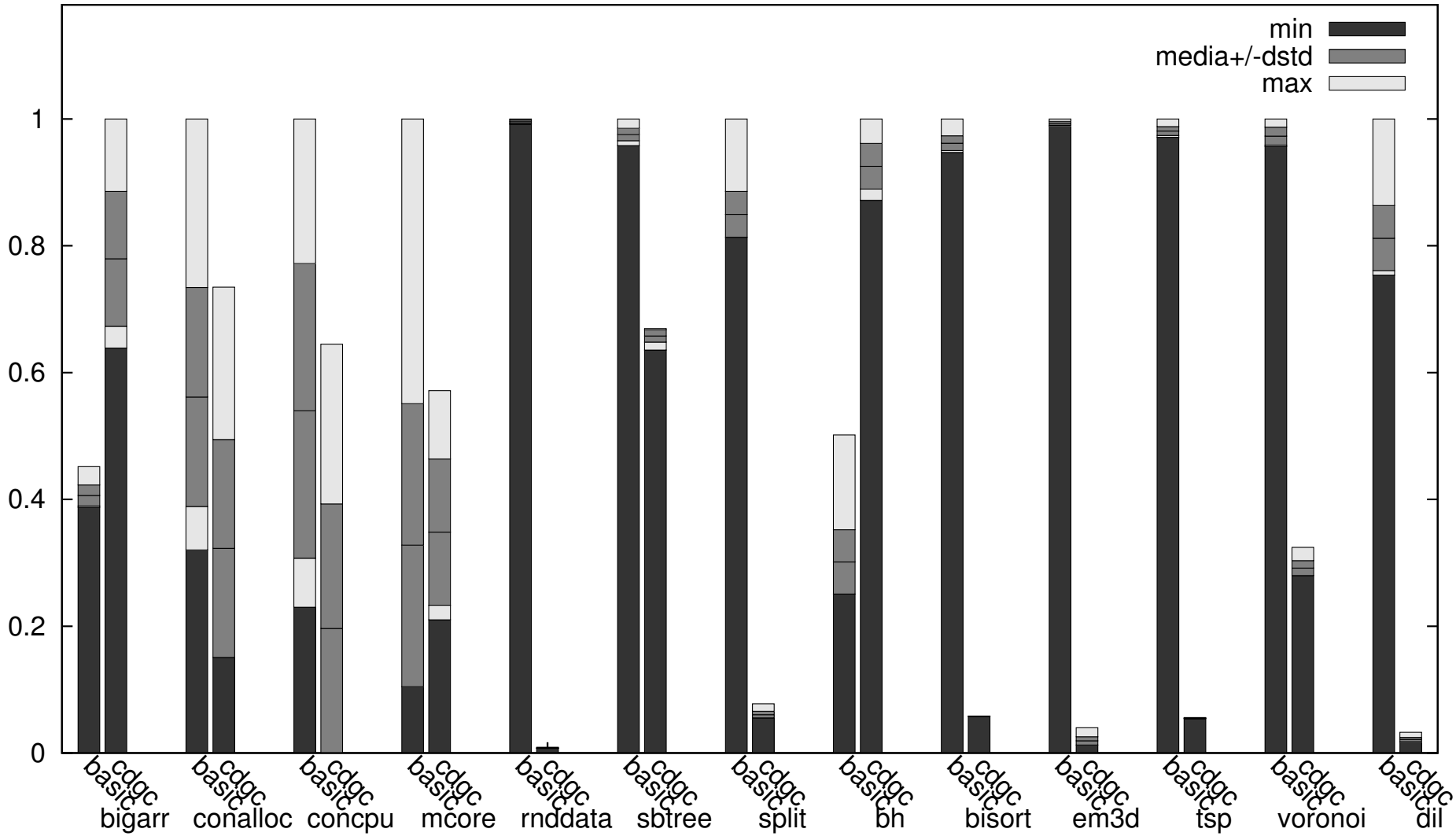
Métricas

- Tiempo total de ejecución
- Tiempo máximo de *stop-the-world*
- Tiempo máximo de pausa real
- Cantidad máxima de memoria utilizada

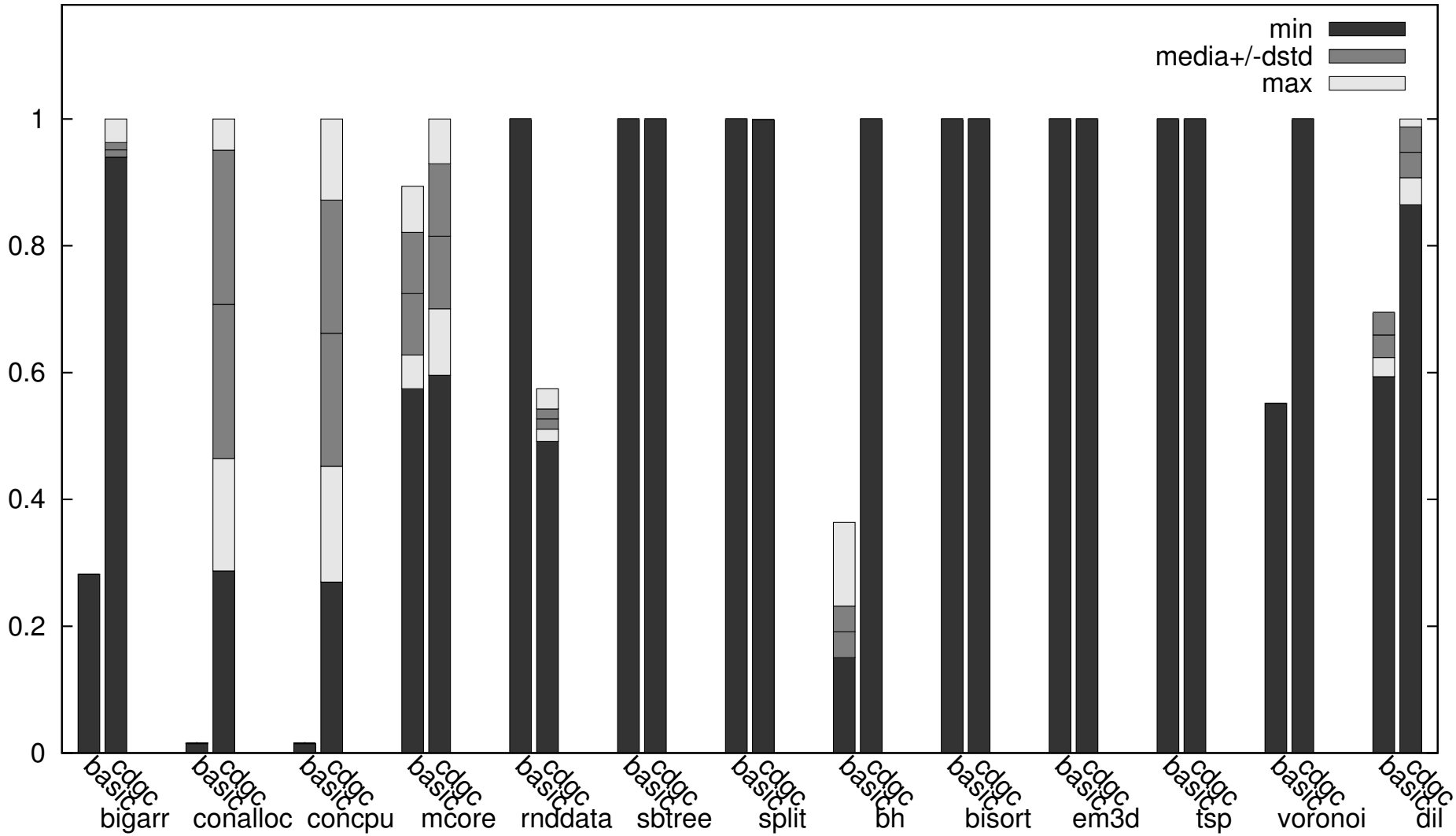
Tiempo Máximo de Stop-The-World



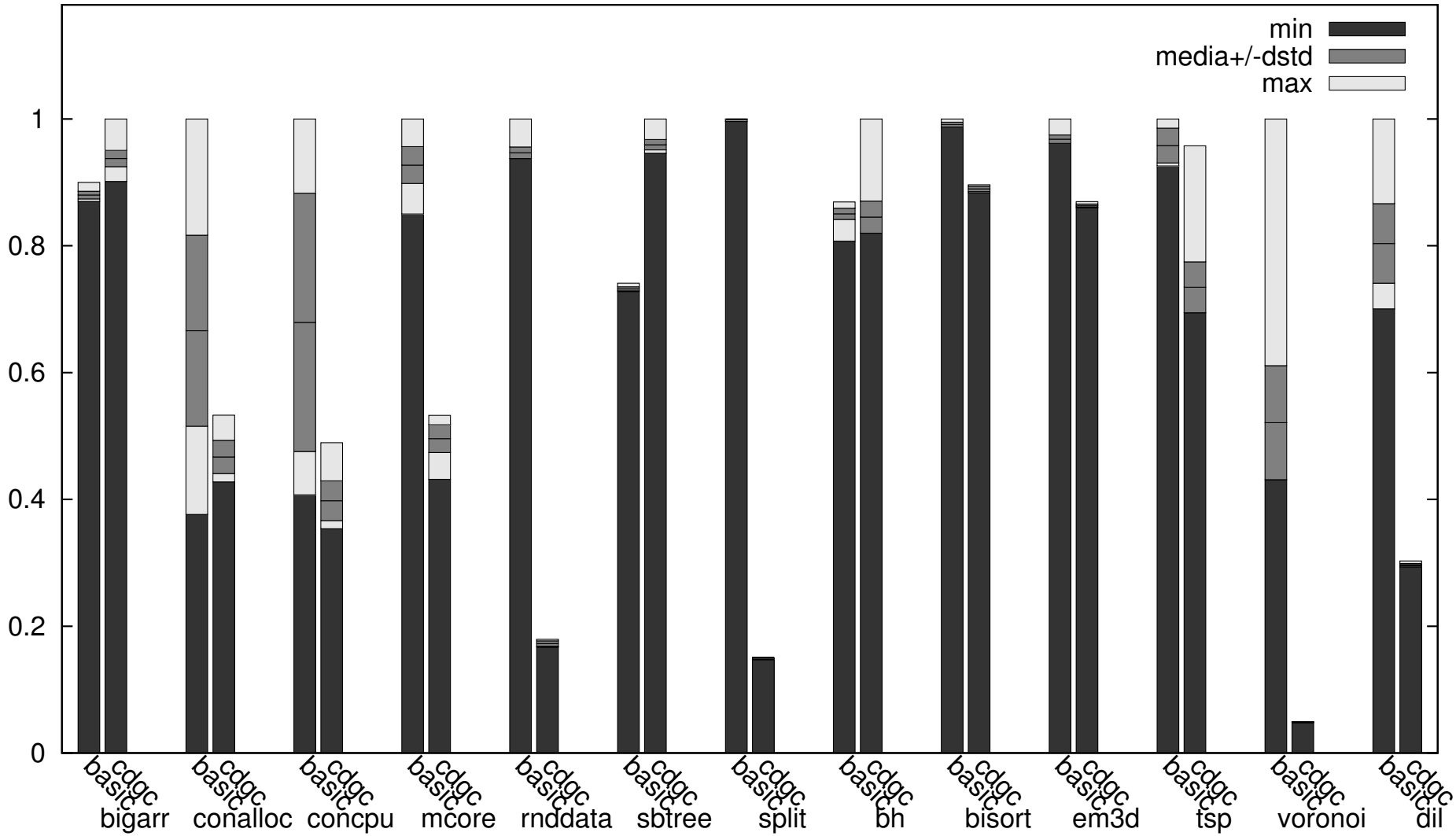
Tiempo Máximo de Pausa Real



Cantidad Máxima de Memoria Utilizada



Tiempo Total de Ejecución



Resumen

- Objetivo principal

  Minimizar tiempo de pausa para programas reales
  Tiempo de pausa de Dil:
  - *Stop-the-world* 160 veces menor (1.66s → 0.01s)
  - Pausa real 40 veces menor (1.7s → 0.045s)
- Objetivo secundario

  No empeorar mucho el recolector actual en ningún aspecto
  Utilización de memoria de Dil:
  50 % mayor (213MiB → 307MiB)
  (mucho *overhead* por marcado preciso)
- Yapa

  Tiempo total de ejecución de Dil:
  Casi 3 veces menor (55s → 20s)

Problemas, Limitaciones y Puntos Pendientes

- Explosión de uso de memoria con *eager allocation*
- Eficiencia del marcado preciso
- Mejorar predicción de *early collection*
- Experimentar con `clone(2)`

Trabajos Relacionados

- *Memory Management in the D Programming Language*
  Vladimir Panteleev. Proyecto de licenciatura, Universitatea Tehnică a Moldovei, 2009.
- *Integrate Precise Heap Scanning Into the GC*
  David Simcha (GC + diseño) y Vincent Lang (compilador). No formal, *bug report*, 2009-2010.
- *Non-intrusive Cloning Garbage Collection with Stock Operating System Support*
  Gustavo Rodriguez-Rivera y Vince Russo. Software Practiceand Experience Volumen 27, Número 8. Agosto 1997.

Trabajos Futuros

- Organización de memoria
- Barrido
- Precisión
- Concurrencia $\rightarrow$ *Lock* global
- Movimiento

Preguntas

¿?

Fin

¡Gracias!