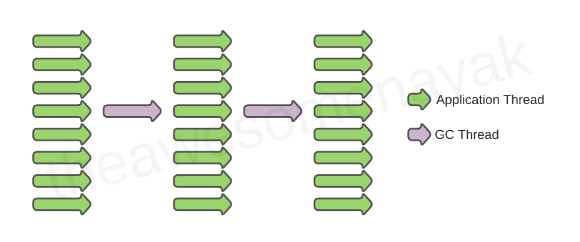
## Implementações de GC

* O padrão JVM define 5 implementações de GC, serial GC, parallel GC, CMS GC, G1 GC e ZGC
* Podem existir implementações específicas de um vendor

### Serial GC

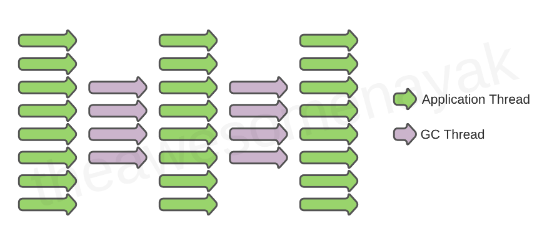
* Usa uma thread para fazer GC, todas as fases são realizadas de maneira sequencial e causam pausa na execução



* Na young gen faz marking com stop-the-world e sweep with copying
* Na old gen faz marking com stop-the-world e sweep with compacting
* Só é recomendado em programas pequenos e single thread

#### Parallel GC ou Throughput Collector

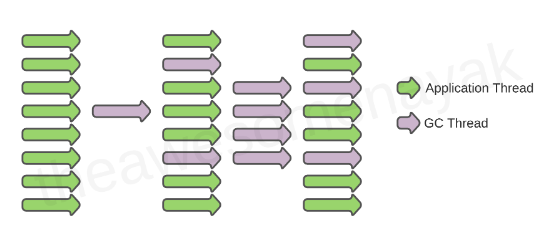
* Padrão de Java 8. É serial GC, mas usando múltiplas threads para fazer mark/sweep



* Na young gen faz marking com stop-the-world e sweep with copying. Na old gen faz marking com stop-the-world e sweep with compacting
* O paralelismo faz com que mark e sweep sejam mais rápidos
* É usado em ambientes multi-thread, em aplicações pequenas

### Concurrent Mark-Sweep (CMS)

* Também é chamado de concurrent low pause collector
* O objetivo dessa implementação é minimizar as pausas de GC na old gen e fazer a coleta de maneira quase totalmente concorrente com as threads da aplicação
* O foco de CMS é na old gen, a coleta na young gen usa parallel ou serial GC
* Na old gen stop-the-world acontece duas vezes e dura pouco (marking/final marking)
* Nem toda aplicação aguentará compartilhar recursos com o GC enquanto roda, mas se esse for o caso a eficiência é maior que parallel/serial

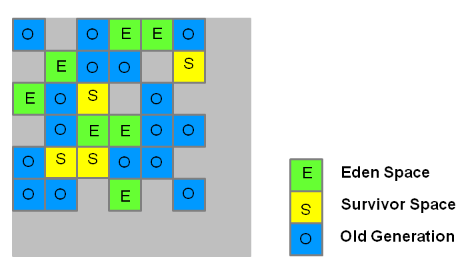


#### Major GC

1. **Initial mark (stop-the-world de curta duração):** Marca objetos na old generation que possuem referência, mesmo que a partir de objetos na young gen
2. **Concurrent Marking:** De maneira concorrente, marca mais objetos
3. **Remark (stop-the-world):** Pausa a execução e encontra objetos que foram perdidos durante a fase 2 devido a execução concorrente, garantindo que nenhum objeto live seja removido por acidente
4. **Concurrent Sweep:** Faz a coleta de objetos sem referência, os objetos removidos normalmente não são movidos e o espaço fica vazio, causando fragmentação eventualmente
   * **Concurrent mode failure:** Ocorre quando o sweep não consegue liberar memória para alocar a um objeto criado, nesse caso ocorrerá um stop-the-world para compactar a old gen após o sweep
5. **Resetting:** Limpa as estruturas usadas e prepara para a próxima coleta

### Garbage First (G1)

* Suportado desde Java 7 e o padrão a partir de Java 9 tem foco em máquinas com múltiplos processadores, permitindo ter uma quantidade mínima de pausas e mantendo um throughput alto
* O heap é dividido em regiões de tamanho variável (1-32mb), que podem ser usadas para a eden spaces, survivor spaces ou para a old gen.



* Possui uma lista que contem cada região e a quantidade de objetos não alcançáveis nelas chamada collection set.
  + Regiões com mais objetos para coleta são priorizadas para GC, e por isso que é chamado garbage first
* Usa sweep with copying e faz stop-the-word curtos para compactar as regiões no heap (não os espaços dentro das regiões, mas as regiões espalhadas no heap)
* **Mixed garbage collection:** minor gc e major gc são feitas ao mesmo tempo para melhorar a utilização do heap e evitar as pausas longas durante o major gc
* **Pause time target (VM arg):** É um parâmetro da JVM que define a periodização ideal para os ciclos de GC, o comportamento de G1 será modificado para atingir esse tempo ideal
* **Humongous Objects:** São objetos que ocupam mais da metade do armazenamento de uma região, eles são mantidos em uma área separa do heap a e verificados durante concurrent mark
* **String deduplication (VM arg):** Verifica objetos string e quando encontra strings iguais atualiza sua referência para apontar para o mesmo char array, fazendo com que o char array repetido seja elegível para GC (é concorrente)
* Assim como CMS, se a aplicação tenta alocar mais memória do que está sendo liberada é necessário fazer um stop-the-world para liberar mais
* Ideal para aplicações que tem uma quantidade de memória disponível alta e necessitam de alto desempenho

#### Minor e Major GC

1. **Initial mark (stop-the-world):** Marca survivor regions que podem ter objetos com referência para objetos na old gen
2. **Root region scan:** Verifica as survivor regions marcadas anteriormente para encontrar os objetos com referência para objetos na old gen, essa fase deve ser concluída antes de um gc na young gen
3. **Concurrent marking (todo o heap):** Verifica todas regiões do heap e marca objetos com referência sem pausar a execução. Pode ser interrompido por young gc
4. **Mark (stop-the-world):** Usando um algorítimo chamado snapshot-at-the-beggining (mais rápido) faz o resto do marking rapidamente
5. **Cleanup (stop-the-world e concorrente):** Conta objetos live em cada região e verifica os collection sets (stop-the-world), depois limpa as regiões com sweep with copying e coloca-as na free list
   * **Free list:** Uma região que foi limpa anteriormente (source) por sweep with copying é uma free region, as free list guarda essas regiões
6. **Copying (stop-the-world):** Pausas para copiar objetos para free regions ou para fazer sweep em source regions. Pode ser feito em young gen ou young gen e old gen ao mesmo tempo
7. Após a coleta ser finalizada as regiões são compactadas no heap

### Outros GC

* Epsilon Garbage Collector: Não faz nenhuma coleta de GC, quando o heap fica cheio ocorre OutOfMemoryError e a JVM para, é usado apenas quando a aplicação não necessita de coleta
* Shenandoah (JDK 12): Usa os mesmos princípios de G1, porém evacuate é feito sem nenhuma pausa, o processo de GC ocorre em paralelo a execução principal, tornando possível fazer GC assim que um candidato é encontrado, mas o custo disso é um maior uso de CPU
* ZGC (JDK 15): É para aplicações que precisam de latência mínima, todo o processo de GC é feito em paralelo com a execução, o diferencial são as pausas de execução de até 10 ms