CLR ARCHITECTURE

Построение графа достижимости объектов сборщиком мусора в .NET

Внутренняя архитектура CLR

- GC Roots
- Mark Phase
- Generations

Memory Management · Graph Theory · Performance Optimization

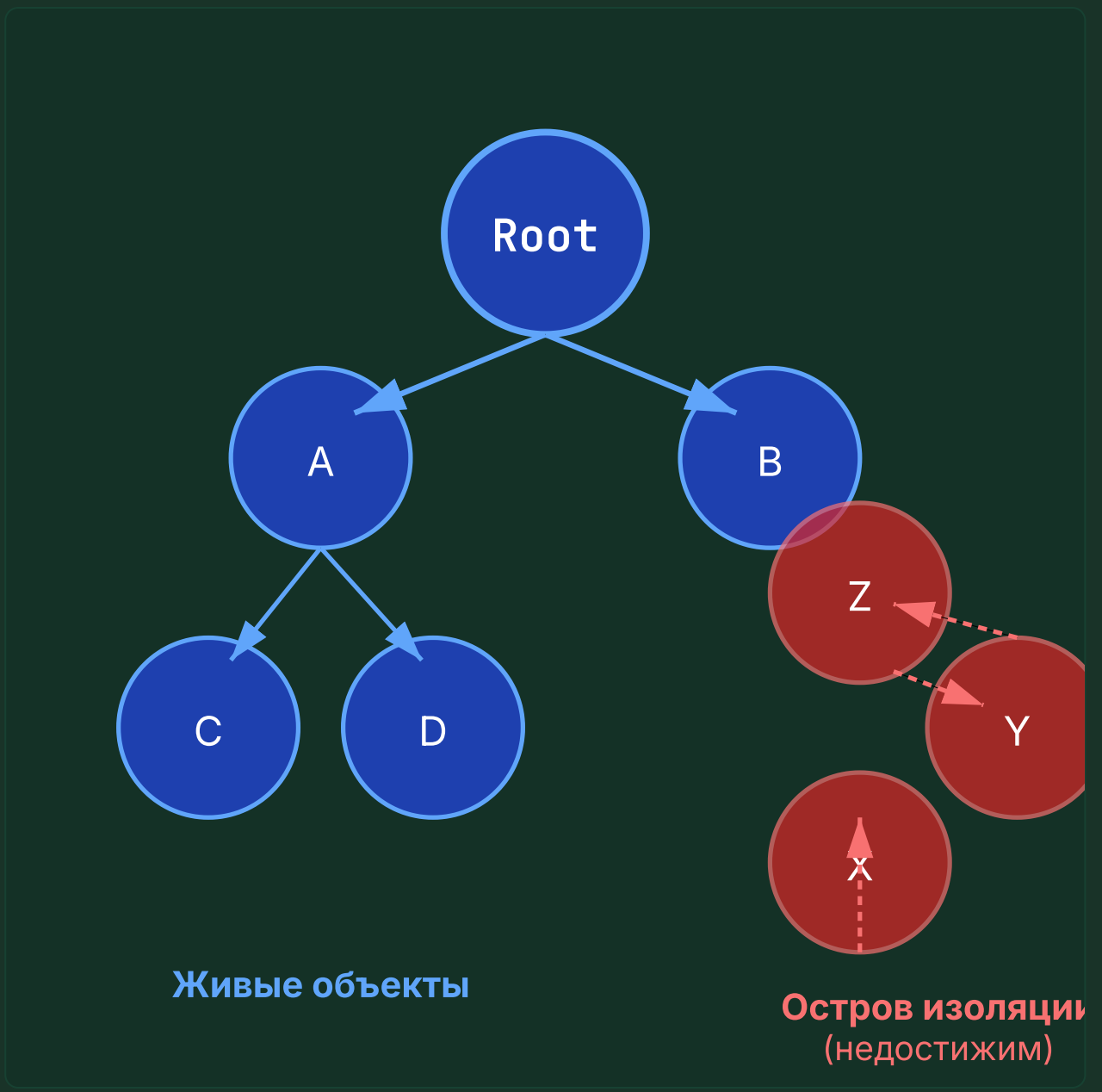
Понятие графа объектов

Узлы (Nodes)

- Ссылочные объекты в Managed Heap
- Созданы через `new ClassName()`
- Содержат MethodTable — паспорт объекта

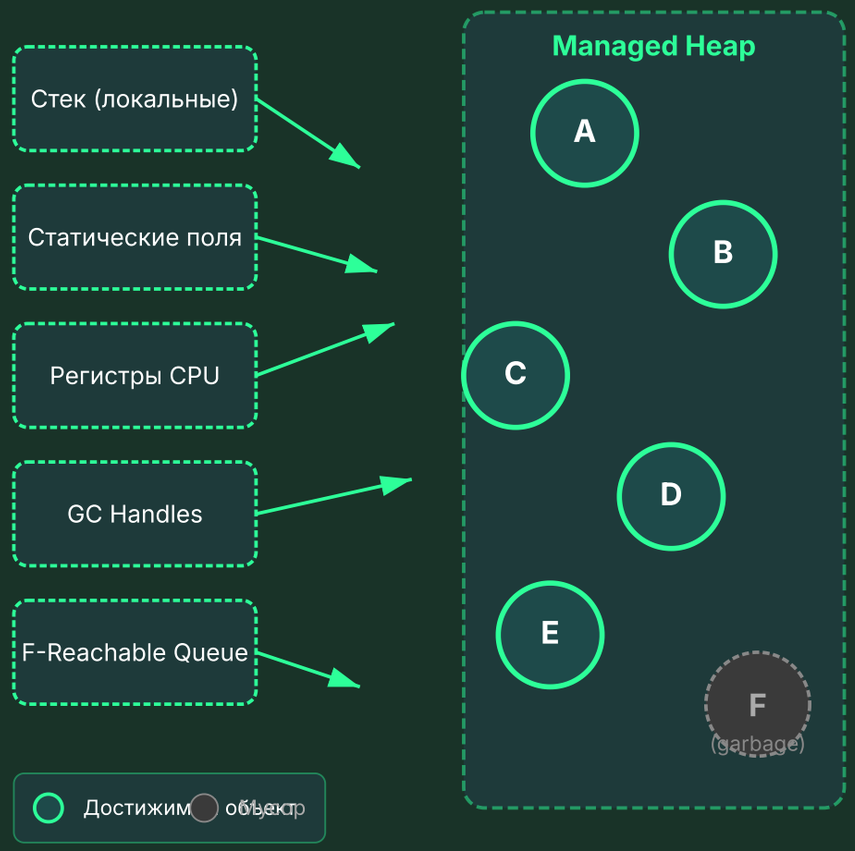
Ребра (Edges)

- Связи через поля и свойства
- Ориентированные: `Родитель → Потомок`
- 4/8 байт — адрес в памяти



Ключевой принцип: Объект жив только если на него указывает стрелка от живого объекта

Корни GC (GC Roots)



Точки входа в граф объектов

Стек потоков (Thread Stacks)

Локальные переменные методов • JIT-оптимизация корней

Статические поля

Опасные корни • Утечки памяти • Живут до завершения процесса

Регистры процессора

Адреса в регистрах CPU • Сканирование контекста

GC Handles

Взаимодействие с неуправляемым кодом • Pinned Objects

F-Reachable Queue

Очередь финализации • "Зомби"-объекты

Фаза Маркировки (Mark Phase)

Stop-The-World

Остановка потоков для подсчета живых объектов

1 **Берем корень**

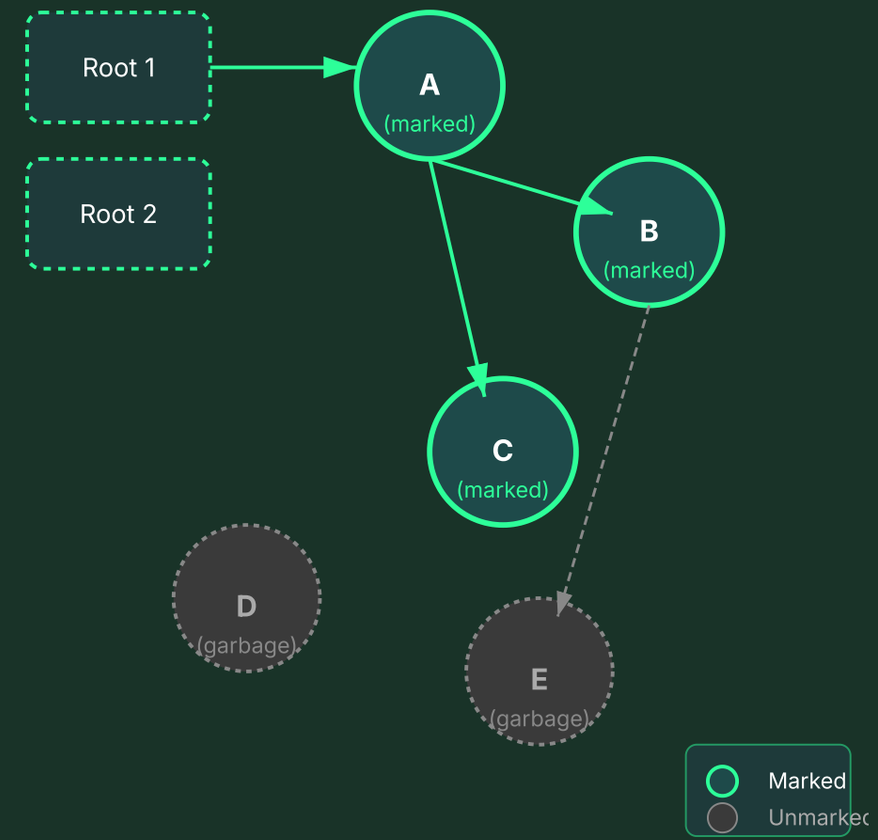
Переходим по ссылке к объекту в куче

2 **Ставим "галочку"**

Marked bit = 1 • Используем MethodTable

3 **Обходим связи**

Идем вглубь по ссылкам • Проверяем на зацикливание



Результат фазы

Бит = 1 → Живой объект
Бит = 0 → Мусор

Проблема поколений (Generations)

Гипотеза поколений

Большинство объектов умирают молодыми

Gen 0 — "Ясли"

Fast

Новые объекты • Молниеносная сборка • Большинство — мусор

Gen 1 — Буферная зона

Medium

Пережили одну проверку • Промежуточное поколение

Gen 2 — "Дом престарелых"

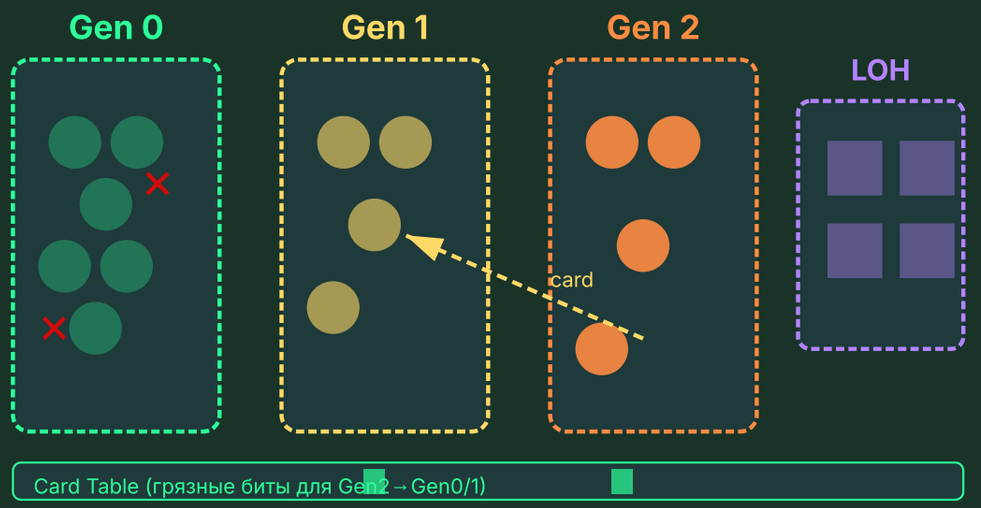
Expensive

Долгожители • Кэши • Синглтоны • Full GC

LOH — Крупные объекты

>85KB

Сразу Gen 2 • Без копирования • Массивы больших данных



⇄ Write Barriers & Card Table

- Отслеживание связей `Старый → Новый`
- Барьер записи при каждом присваивании
- Card Table — карта "грязных" участков

Частичный граф — игнорируем 90% памяти

Сканируем только "грязные" блоки Gen 2

Специальные узлы графа

Pinned Objects

Закрепленные объекты

- Необходимы для `Interop` с неуправляемым кодом
- Неперемещаемые узлы • Адрес фиксирован

Проблемы

- **Фрагментация** памяти — "дырки" перед pinned
- Замедление выделения памяти
- Блокируют Compacting Phase

Weak References

Слабые ссылки

- "Призрачные" связи в графе
- Игнорируются при `Mark Phase`
- Идеальны для **кэширования**

Механизм работы

- GC проверяет после маркировки
- Объект выжил? → Ссылка валидна
- Объект мертв? → `Target = null`

Pinned = Якоря | **Weak** = Призраки

Финализация и Воскрешение

Механизм "Воскрешения"

Шаг 1

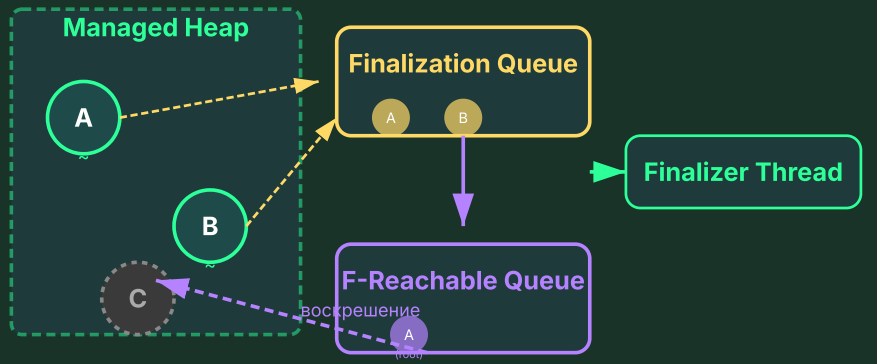
Finalization Queue

Объект в списке финализации

Шаг 2

F-Reachable Queue

Перенос при обнаружении мусора



Объект воскресает!

F-Reachable Queue — это `GC Root`

Объект снова достижим → переходит в Gen 1/2

Двойная жизнь

Мусор становится долгожителем

Задержка

Память освобождается позже

Блокировка

Finalizer Thread может зависнуть

Золотое правило

- **Избегайте** финализаторов
- Используйте `IDisposable`
- Вызывайте `GC.SuppressFinalize(this)`

Конкурентная маркировка (Concurrent Marking)

Три фазы процесса

1 Initial Mark

Короткая пауза STW • Сканирование корней

2 Concurrent Mark

Приложение работает • GC в фоновых потоках

3 Final Mark

Журнал изменений • Докрашивание графа

Low Latency

Незаметные паузы для огромных куч

Ephemeral GC

Gen 0/1 поверх фоновой Gen 2

Write Barriers в действии

Проблема согласованности

Приложение меняет граф, пока GC его обходит

Решение

- Логирование изменений ссылок
- Быстрое докрашивание измененных участков
- Синхронизация в Final Mark фазе

Современный граф — живой организм