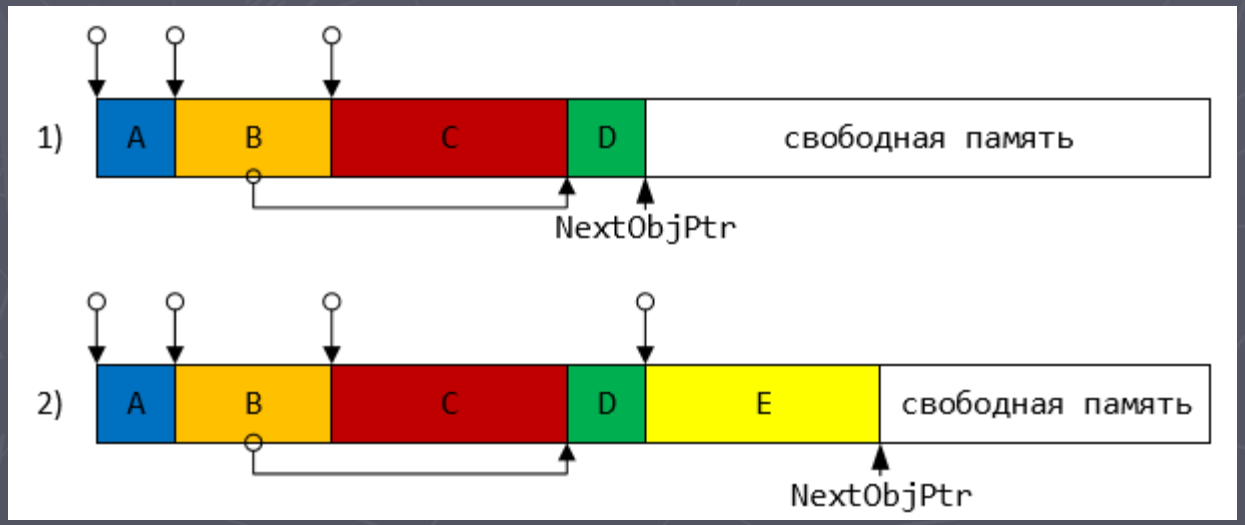
Жизненный цикл объектов

Жизненный цикл объектов

- **Типы платформы .NET: *ссылочные* и *типы значений*.**
- **Локальная переменная типа значений:**
  - **создаётся в стеке;**
  - **время жизни = время работы метода (после выхода из метода память в стеке, занимаемая переменной, автоматически освободится).**
- ***) в реальности стеков несколько (многопоточность)**

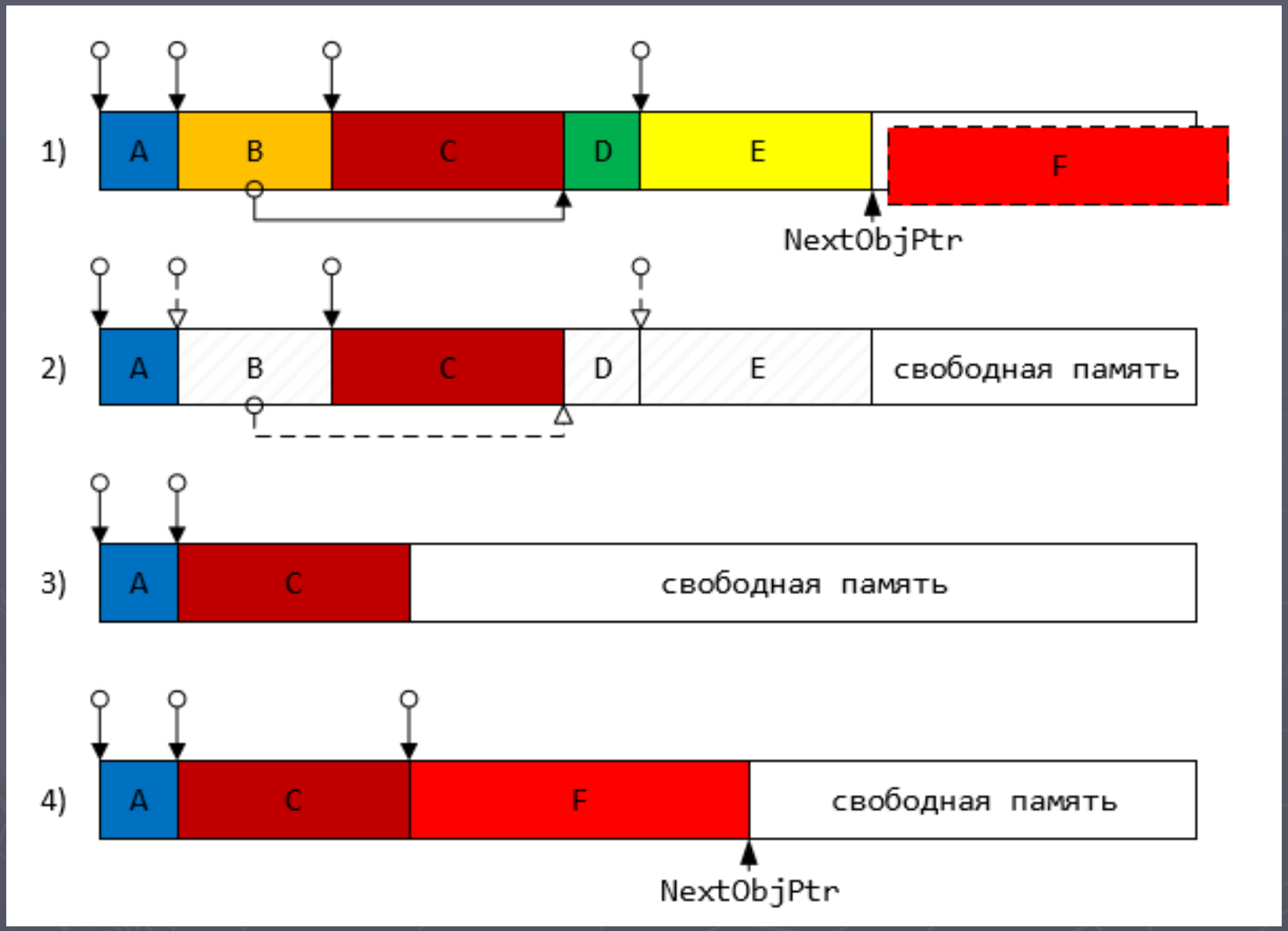
Управляемая куча (managed heap)

- Размещение объектов в управляемой куче происходит последовательно
- CLR поддерживает указатель на свободное место в куче



Алгоритм сборки мусора

- строится граф используемых объектов
  - корневые объекты
- выясняется реально занимаемая этими объектами память
- дефрагментация кучи – используемые объекты перераспределяются так, чтобы занимаемая ими память составляла единый блок в начале кучи


1)
A
B
C
D
E
F
NextObjPtr
2)
A
B
C
D
E
свободная память
3)
A
C
свободная память
4)
A
C
F
свободная память
NextObjPtr

Оптимизации

- 1) объекты размером 85000 и более байтов размещаются в отдельной управляемой куче больших объектов (Large Object Heap). –Gen2
  - не дефрагментируется
- 2) для малых объектов выделяется три поколения – Gen0, Gen1 и Gen2.

Если в Gen1 произошла сборка мусора, то «пережившие» её объекты переходят в поколение Gen2

Вначале все объекты

После первой сборки мусора «пережившие» её объекты переходят в поколение

статический класс System.GC

- Collect() – вызывает принудительную сборку мусора в программе.
- GetGeneration() – возвращает номер поколения для указанного объекта;
- GetTotalMemory() – возвращает количество используемой памяти в управляемой куче;
- SuppressFinalize() – подавляет вызов финализатора для объекта;
- WaitForPendingFinalizers() – приостанавливает текущий поток выполнения, пока не будут выполнены все финализаторы освобождаемых объектов.

Финализаторы и интерфейс IDisposable

- System.Object содержит виртуальный метод Finalize().

- Класс (но не структура!) может переопределить этот метод для освобождения неуправляемых ресурсов.

Финализаторы – работа

- Объект класса с методом `Finalize()` обрабатывается особо при размещении в куче и при сборке мусора.
- **Размещение**: ссылка на объект запоминается в специальной внутренней структуре CLR.
- Сборка мусора: ссылка на объект перемещается в *очередь завершения* (freachable queue). Затем в отдельном потоке выполнения у объектов из очереди вызывается `Finalize()`, после этого ссылка на объект удаляется из очереди завершения.

- не позволяет явно переопределить
- ~имя-класса()
- не имеет параметров и модификаторов доступа (считается, что у него модификатор доступа protected).
- При наследовании в финализатор класса-наследника автоматически подставляется вызов финализатора класса-предка.

```
public class ClassWithFinalizer
{
    public void DoSomething()
    {
        Console.WriteLine("I am working...");
    }

    ~ClassWithFinalizer()
    {
        // здесь должен быть код освобождения неуправляемых ресурсов
        Console.WriteLine("Bye!");
    }
}
```

IDisposable

- Для освобождения управляемых ресурсов (т. е. ресурсов платформы .NET) программист может описать в классе метод, который следует вызывать вручную, когда ресурс больше не нужен.
- IDisposable - Dispose()

```
public class ClassWithDispose : IDisposable
{
    public void DoSomething()
    {
        Console.WriteLine("I am working...");
    }

    public void Dispose()
    {
        // здесь должен быть код освобождения управляемых ресурсов
        Console.WriteLine("Bye!");
    }
}
```

```
using (ClassWithDispose x = new ClassWithDispose())
{
    x.DoSomething();
    // компилятор C# поместит сюда вызов x.Dispose()
}
```

Слабые ссылки (weak reference)

- особый вид ссылки на объект в системах со сборкой мусора
- представлены классами System.WeakReference System.WeakReference<T>

```
var weak = new WeakReference(new StringBuilder("Test"));
```

- он рассматривается алгоритмом сборки мусора как подлежащий удалению

```
if (weak.IsAlive)
{
    Console.WriteLine(weak.Target);          // Test
}
GC.Collect();
Console.WriteLine(weak.Target == null);     // True
```

Одно из применений слабых ссылок – построение кэшей больших объектов. При кэшировании формируется слабая ссылка на объект.