ОС WINDOWS

Часть 2

Разрядность процессора – это величина, которая определяет размер машинного слова, то есть количество информации, которой процессор обменивается с оперативной памятью. Каждая ячейка ОЗУ имеет адрес и в 32-х битной системе он записывается в виде двоичного кода длиной в 32 символа. (8 бит = 1 байт)

Из простых математических преобразований получаем 2^{32}=4294967296 байт = 4 ГБ.

То есть, в 32-битной системе, ячейки памяти условно расположенные за пределом 4ГБ просто не получат адреса и не будут использоваться.

Если использовать 64-х битную систему, то размер адресуемой памяти 2^{64}= 18446744073709551616 = 16 Эб (эксабайт).

Единица	Аббревиатура	Сколько
бит	б	1
байт	Б	8 бит
килобит	кбит (кб)	1 000 бит
килобайт	КБайт (КБ)	1024 байта
мегабит	мбит (мб)	1 000 килобит
мегабайт	МБайт (МБ)	1024 килобайта
гигабит	гбит (гб)	1 000 мегабит
гигабайт	ГБайт (ГБ)	1024 мегабайта
терабит	тбит (тб)	1 000 гигабит
терабайт	ТБайт (ТБ)	1024 гигабайта

Также вы можете воспользоваться конвертером

Конвертер байт

Перевести

1844674407370955 бит (б)

в

терабайт (ТБ)

Результат конвертирования

18446744073709551616 б = **2097152 ТБ**

Какой объем ОЗУ поддерживают различные версии и выпуски Windows

Version (only in X64 editions **)** **Limit on X64**

- Windows Server 2012 Datacenter 4 TB
- Windows Server 2012 Standard 4 TB
- Windows Server 2012 Essentials 64 GB
- Windows Server 2012 Foundation 32 GB
- Windows Storage Server 2012 Workgroup 32 GB
- Windows Storage Server 2012 Standard 4 TB

Version	**Limit on X86**	**Limit on X64**
Windows 8 Enterprise	4 GB	512 GB
Windows 8 Professional	4 GB	512 GB
Windows 8	4 GB	128 GB

Диспетчер памяти

Максимальный объем физической памяти, поддерживаемый Windows, варьируется от 2 до 1024 Гб в зависимости от версии и редакции Windows. Так как виртуальное адресное пространство может быть больше или меньше объема физической памяти в компьютере, диспетчер управления памятью решает две главные задачи.

- Трансляция, или проецирование (mapping), виртуального адресного пространства процесса на физическую память. Это позволяет ссылаться на корректные адреса физической памяти, когда потоки, выполняемые в контексте процесса, читают и записывают в его виртуальном адресном пространстве. Физически резидентное подмножество виртуального адресного пространства процесса называется рабочим набором (working set).
- Выгрузка части содержимого памяти на диск, когда потоки или системный код пытаются задействовать больший объем физической памяти, чем тот, который имеется в наличии, и загрузка страниц обратно в физическую память по мере необходимости.

3. Несколько ключевых компонентов, работающих в контексте шести различных системных потоков режима ядра.

a) *Диспетчер рабочих наборов* (working set manager) с приоритетом 16. Диспетчер настройки баланса (системный поток, создаваемый ядром) вызывает его раз в секунду или при уменьшении объема свободной памяти ниже определенного порогового значения. Он реализует общие правила управления памятью, например усечение рабочего набора, старение и запись модифицированных страниц.

b) *Поток загрузки и выгрузки стеков* (process/stack swapper) с приоритетом 23- Выгружает (outswapping) и загружает (inswapping) стеки процесса и потока. При необходимости операций со страничным файлом этот поток пробуждается диспетчером рабочих наборов и кодом ядра, отвечающим за планирование.

Диспетчер памяти является частью исполнительной системы Windows, содержится в файле Ntoskrnl.exe и включает следующие **компоненты**.

1. Набор сервисов исполнительной системы для выделения, освобождения и управления виртуальной памятью; большинство этих сервисов доступно через Windows API или интерфейсы драйверов устройств режима ядра.
2. Обработчики ловушек трансляции недействительных адресов (transla-tion-not-vaiid) и нарушений доступа для разрешения аппаратно обнаруживаемых исключений, связанных с управлением памятью, а также загрузки в физическую память необходимых процессу страниц.

c) *Подсистема записи модифицированных страниц* (modified page writer) с приоритетом 17. Записывает измененные страницы, зарегистрированные в списке модифицированных страниц, обратно в соответствующие страничные файлы.

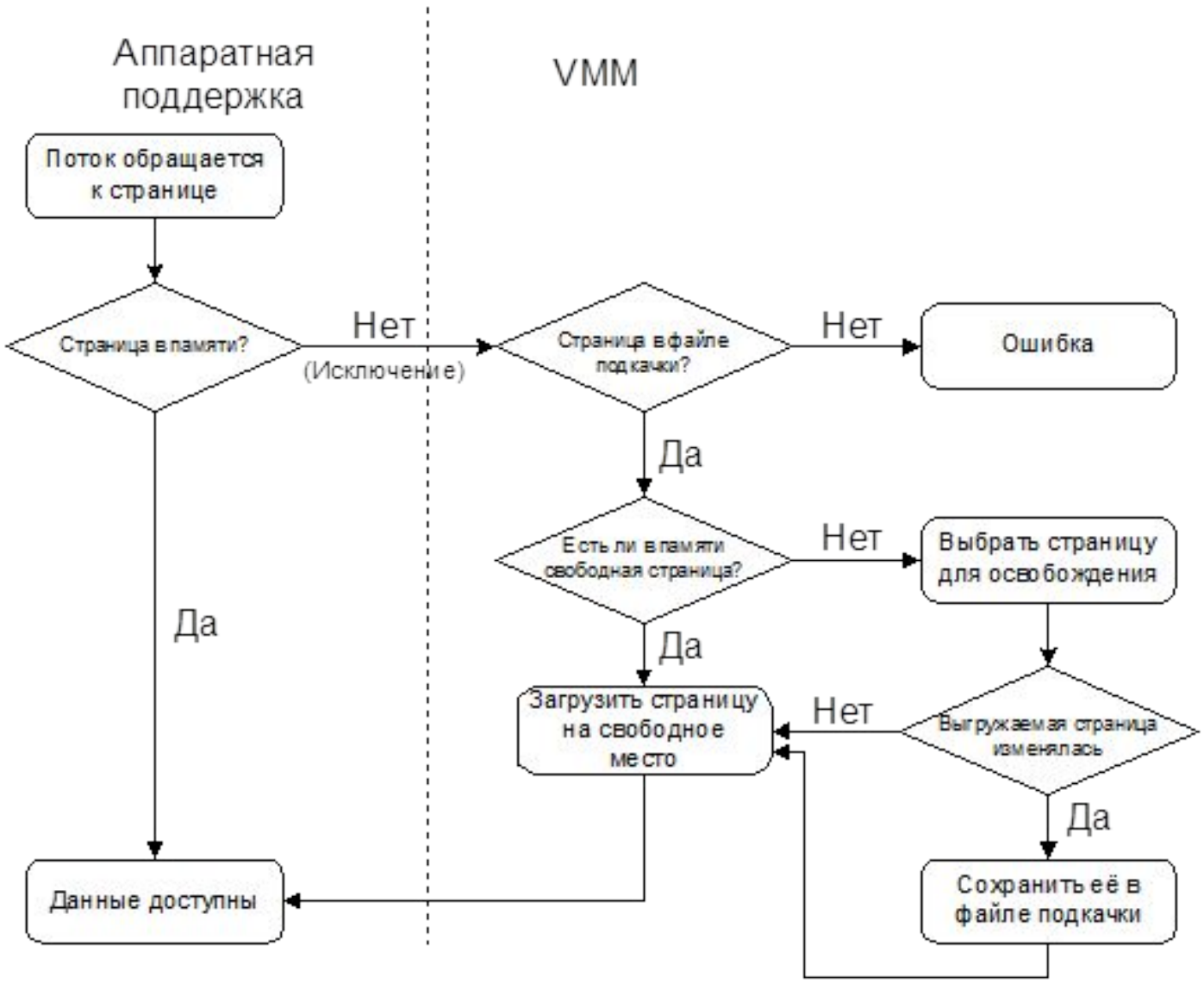
d) *Подсистема записи спроецированных страниц* (mapped page writer) с приоритетом 17. Записывает измененные страницы спроецированных файлов на диск. Пробуждается, когда нужно уменьшить размер списка модифицированных страниц или когда страницы модифицированных файлов находятся в этом списке более 5 минут.

e) *Поток сегмента разыменования* (dereference segment thread) с приоритетом 18. Отвечает за уменьшение размеров системного кэша и изменение размеров страничного файла.

f) *Поток обнуления страниц* (zero page thread) с приоритетом 0. Заполняет нулями страницы, зарегистрированные в списке свободных страниц. (В некоторых случаях обнуление памяти выполняется более скоростной функцией MiZeroInParallel.)

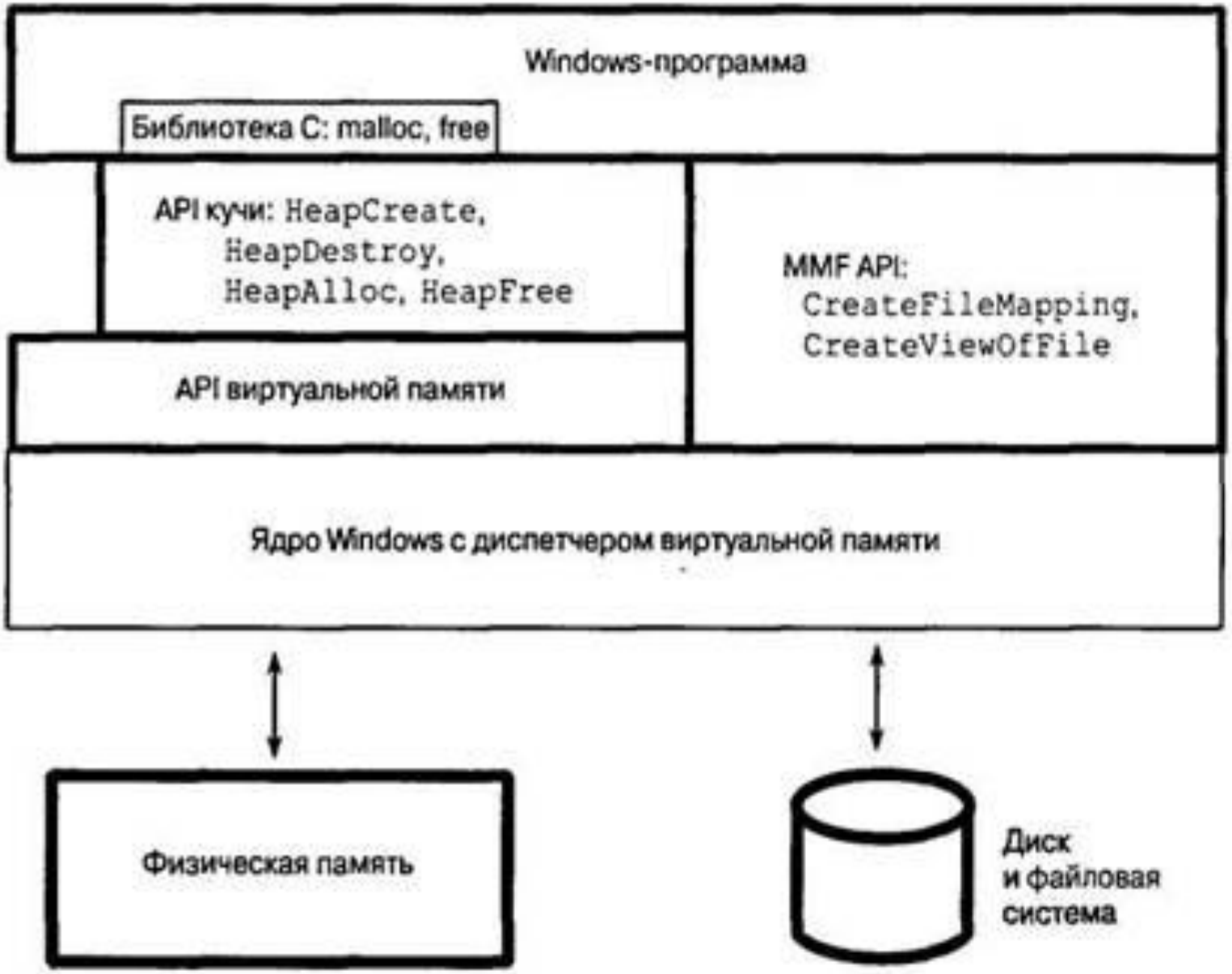
- Диспетчер памяти предоставляет набор **системных сервисов** для выделения и освобождения виртуальной памяти, разделения памяти между процессами, проецирования файлов в память, сброса виртуальных страниц на диск, получения информации о диапазоне виртуальных страниц, изменения атрибутов защиты виртуальных страниц и блокировки в памяти. Большинство этих сервисов предоставляется через Windows API. В него входят три группы прикладных функций управления памятью:

1. для операций со страницами виртуальной памяти (Virtualxxx),
2. проецирования файлов в память (CreateFileMapping, MapViewOfFile),
3. управления кучами (Heapxxx, а также функции из старых версий интерфейса — Localxxx и Globalxxx).



VMM использует следующий алгоритм организации доступа к данным

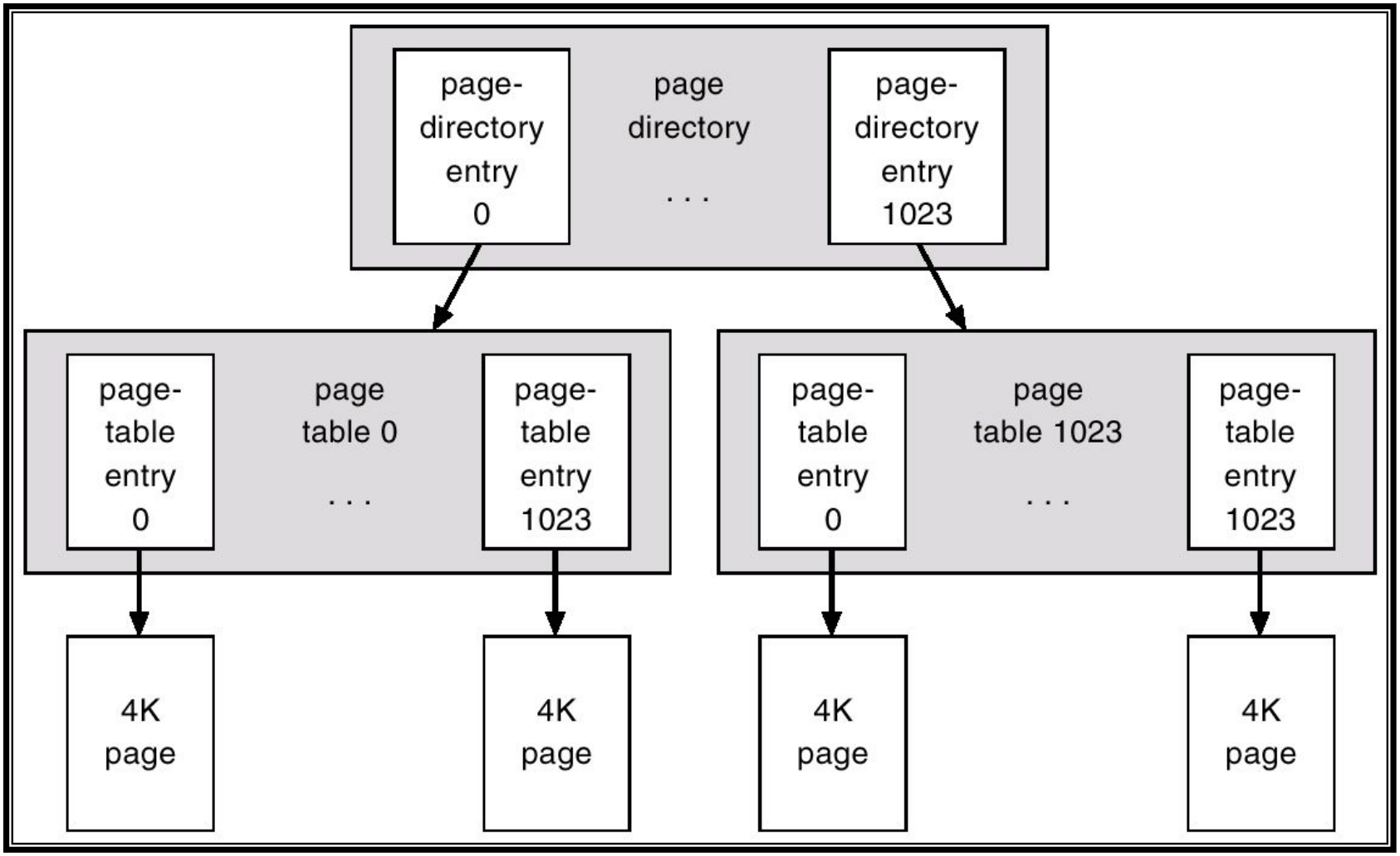
Архитектура системы управления памятью Windows



Executive — менеджер виртуальной памяти

- При проектировании менеджера виртуальной памяти предполагалось, что процессор поддерживает для отображения виртуальных адресов в физические механизм страничной организации, прозрачный кэш для многопроцессорных систем, а также алиасы для виртуальных адресов.
- VM – менеджер в Windows 2000 использует страничную организацию с размером страницы 4 KB (4096) .
- Используется двухуровневая схема выделения памяти.
  - На первом шаге резервируется адресное пространство процесса.
  - На втором шаге выделяется пространство в файле подкачки (paging file).

Распределение виртуальной памяти



Менеджер виртуальной памяти

- **Трансляция виртуальных адресов в Windows 2000 использует несколько структур данных.**
  - **Каждый процесс имеет каталог страниц (*page directory*), содержащий 1024 *элемента* размером по 4 байта.**
  - **Каждый элемент каталога ссылается на *таблицу страниц*, которая содержит 1024 *элемента* (page table entries - PTEs) размером по 4 байта.**
  - **Каждый PTE ссылается на фрейм страницы (4 KB) в физической памяти.**
- **Ссылка на элемент всегда занимает 10 битов (0..1023).**
- **Это свойство используется при трансляции виртуальных адресов в физические.**
- **Станица может находиться в следующих состояниях: valid, zeroed, free, standby, modified, bad.**

Большие и малые страницы

- Виртуальное адресное пространство делится на единицы, называемые страницами. Это вызвано тем, что аппаратный блок управления памятью транслирует виртуальные адреса в физические по страницам. Поэтому страница — наименьшая единица защиты на аппаратном уровне.
- Страницы бывают двух размеров: малого и большого. Реальный размер зависит от аппаратной платформы

Таблица. Размеры страниц

Архитектура	Размер малой страницы	Размер большой страницы
x86	4 Кб	4 Мб
x64	4 Кб	2 Мб
IA64	8 Кб	16 Мб

- Преимущество больших страниц — скорость трансляции адресов для ссылок на другие данные в большой странице. Дело в том, что первая ссылка на любой байт внутри большой страницы заставляет аппаратный ассоциативный буфер трансляции (TLB) загружать в свой кэш информацию, необходимую для трансляции ссылок на любые другие байты в этой большой странице. При использовании малых страниц для того же диапазона виртуальных адресов требуется больше элементов TLB.
- Чтобы задействовать преимущества больших страниц в системах с достаточным объемом памяти, Windows проецирует на такие страницы базовые образы операционной системы (Ntoskrnl.exe и Hal.dll) и базовые системные данные (например, структуры данных, описывающие состояние каждой страницы физической памяти).

- Windows также автоматически проецирует на большие страницы запросы объемного ввода-вывода, если запрос удовлетворяет длине и выравниванию для большой страницы.
- Один из **побочных эффектов** применения больших страниц заключается в следующем. Так как аппаратная защита памяти оперирует страницами как наименьшей единицей, то, если на большой странице содержатся код только для чтения и данные для записи/чтения, она должна быть помечена как доступная для чтения и записи, т. е. код станет открытым для записи.

- Страницы в адресном пространстве процесса могут быть:
- **свободными (free**),
- **зарезервированными (reserved)**
- **переданными (committed**).
- Приложения могут резервировать (reserve) адресное пространство и передавать память (commit) зарезервированным страницам по мере необходимости. Резервировать страницы и передавать им память можно и одним вызовом. Эти сервисы предоставляются через Windows-функции VirtualAlloc и VirtualAllocEx.

Невыгружаемый пул

Ядро и драйверы устройств используют невыгружаемый пул для хранения данных, к которым можно обратиться в случае, когда система не может обработать страничные ошибки.

Ядро входит в такой режим в случаях, когда оно выполняет процедуры обработки прерываний (**ISR**) и отложенные вызовы процедур (**DPC)** - функции, связанные с аппаратными прерываниями.

Выгружаемый пул

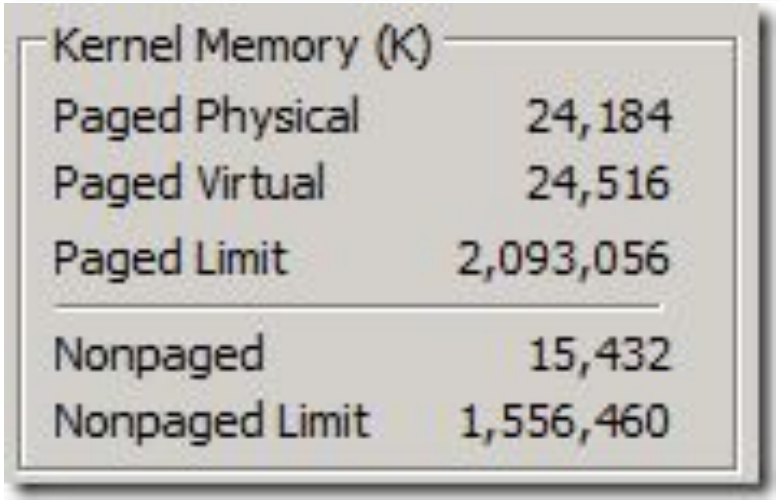
Выгружаемый пул, получил свое название потому, что **Windows** может записать данные, которые он хранит, в файл подкачки, позволяя тем самым использовать физическую память, которая при этом освобождается, в других целях. Структуры данных, которые представлены в виде отображаемых на память файлов, называемых секциями, также хранятся в выгружаемом пуле.

Просмотр информации об использовании выгружаемого и невыгружаемого пулов

Есть три счетчика, показывающих информацию об использовании пула:

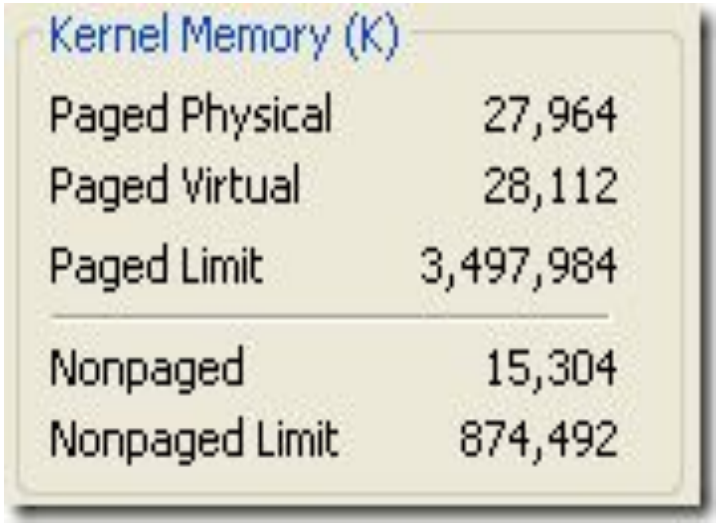
- *Пул невыгружаемых байтов*;
- *Пул выгружаемых байтов (виртуальный размер выгружаемого пула)*;
- *Пул выгружаемых резидентных байтов*.

Ограничения на размер невыгружаемого пула

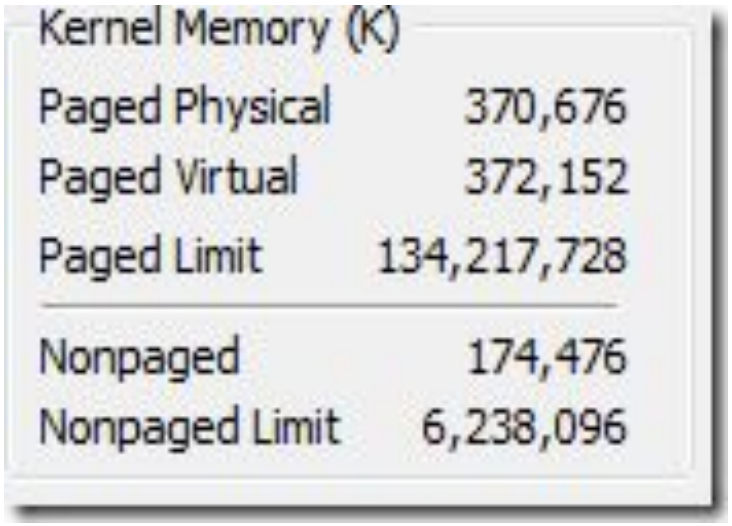


32-х битный
Windows Server 2008
с 2 Гб ОЗУ

Для **32-х битной Windows XP** данный предел вычисляется исходя из того, сколько адресного пространства выделено другим ресурсам, в особенности таблице **PTE**, с максимальным значением в **491Мб**.



64-х битная
Windows XP
с 2 Гб ОЗУ

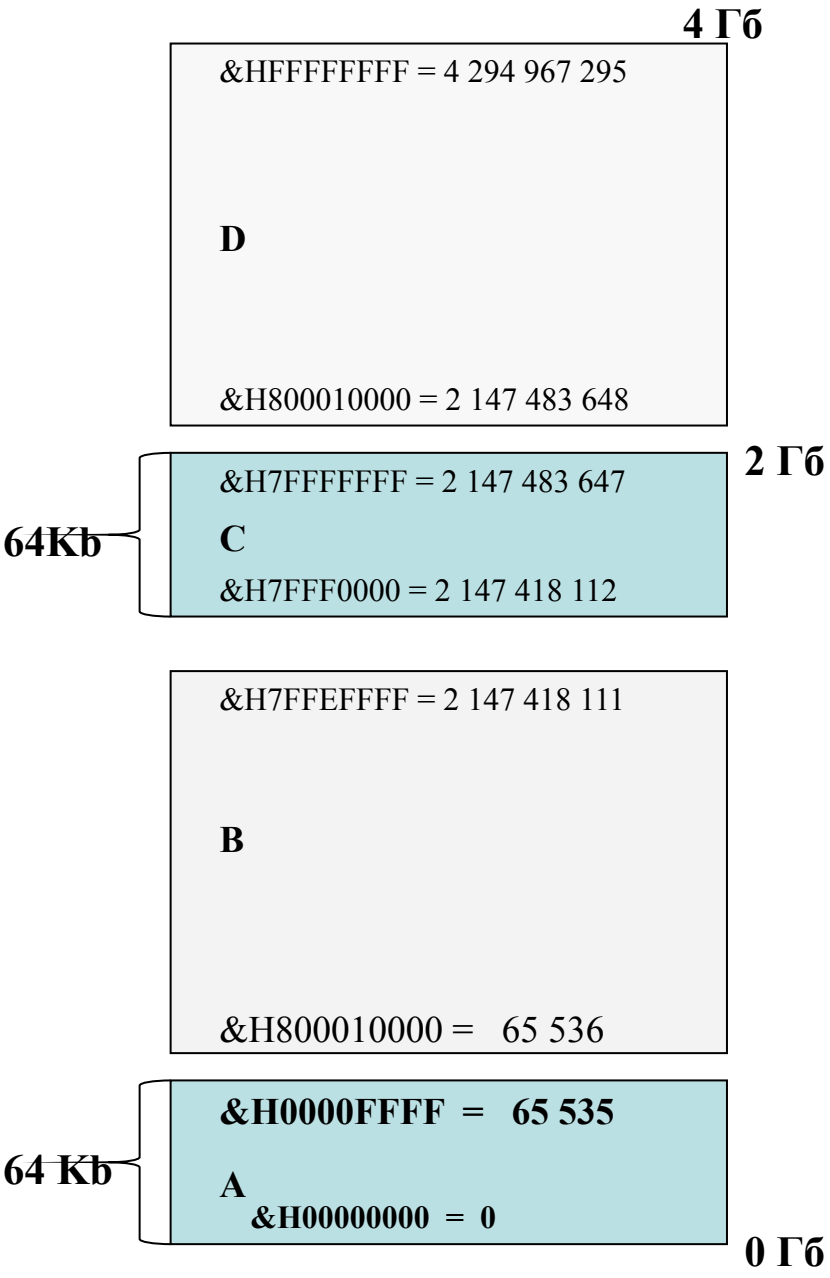


64-битная
Windows 7 с
8 Гб памяти

Распределение ВАП

Виртуальная память 32-разрядной версии Windows обеспечивает каждому процессу 4 Гбайт виртуального адресного пространства (ВАП) (2 Гбайт – ОС, 2 Гбайт – пользовательская программа).

64-разрядная версия Windows предоставляет процессам намного более обширное ВАП: 7152 Гбайт на системах IA-64 и 8192 Гбайт на системах x64.



Зарезервировано Windows для исполнительной системы Windows, ядра и драйверов устройств.
Недоступно в пользовательском режиме.
(2 Гб)

Используется для некорректно инициализированных указателей.
Недоступно в пользовательском режиме. (64 Кб)

Адресное пространство процессов содержит прикладные модули EXE и DLL, Win32 DLL (kernel32.dll, user.dll и т.д.), файлы, отображаемые в память.
Доступно в пользовательском режиме. (2 Гб – 128 Кб)

Используется для некорректно инициализированных указателей.
Недоступно в пользовательском режиме. (64 Кб)

Схема адресного пространств для 32-разрядной версии ОС Windows

Схемы адресных пространств для 64-разрядной версии Windows

Структура 64-разрядного адресного пространства

Регион	IA64	x64	x86
Адресное пространство процесса	7152 Гб	8192 Гб	2–3 Гб
Пространство системных PTE	128 Гб	128 Гб	1,2 Гб
Системный кэш	128 Гб	128 Гб	960 Мб
Пул подкачиваемой памяти	128 Гб	128 Гб	470–650 Мб
Пул неподкачиваемой памяти	128 Гб	128 Гб	256 Мб

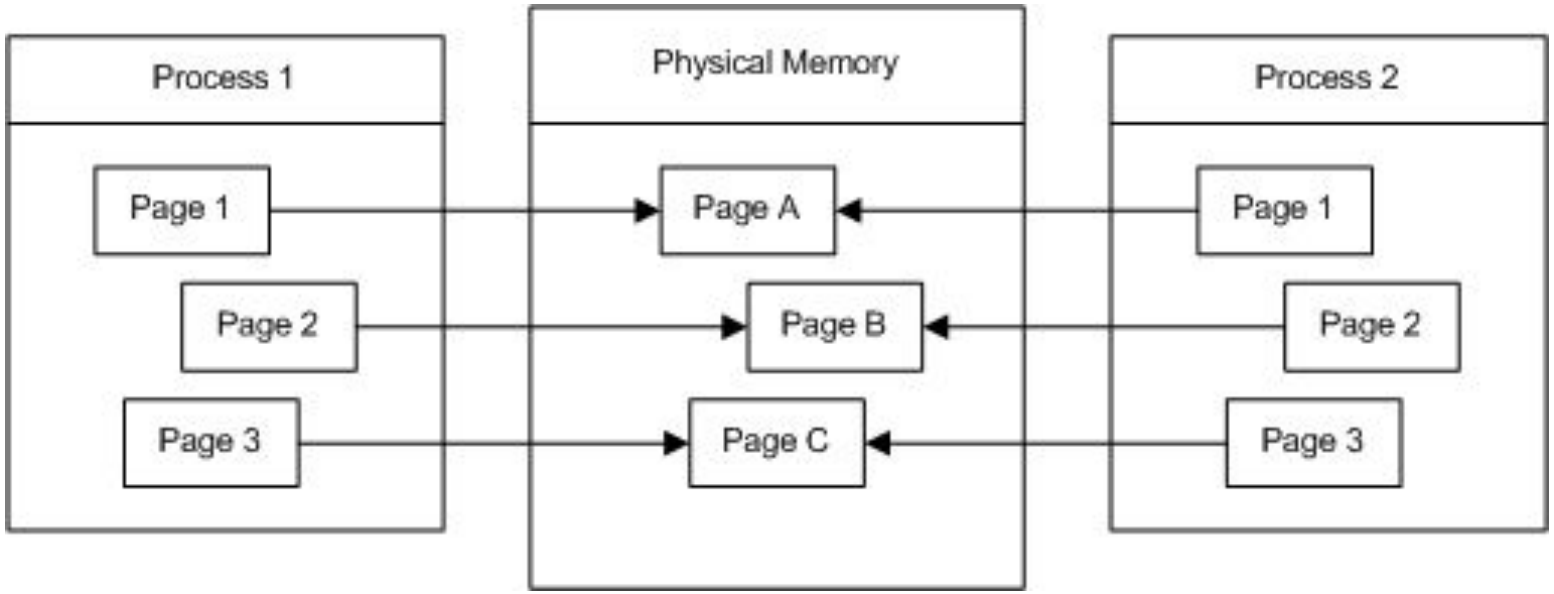
0000000000000000 000007FFFFFEFFF	Адреса пользовательского режима (8 Тб минус 64 Кб)
000007FFFFF0000 000007FFFFFFFFF	Недоступный регион размером 64 Кб
	.
FFFF080000000000	Начало системного пространства
FFFFF68000000000	Карта четырехуровневых таблиц страниц (512 Гб)
FFFFF70000000000	Гиперпространство — списки рабочих наборов и индивидуальные для каждого процесса структуры управления памятью, проецируемые на этот 512-гигабайтный регион
FFFFF78000000000	Разделяемая системная страница
FFFFF78000001000	В этом регионе (размером 512 Гб за вычетом 4 Кб) находится информация рабочего набора системного кэша
	. .
FFFFF80000000000	Проекции, инициализируемые загрузчиком
FFFFF90000000000	Пространство сеанса Регион размером 512 Гб
FFFFF98000000000	В этом 1-терабайтном регионе находится системный кэш Только для доступа в режиме ядра
FFFFFA8000000000	Начало системной области пула подкачиваемой памяти (128 Гб) Только для доступа в режиме ядра
	Проецируемые системой представления начинаются сразу за пулом подкачиваемой памяти. По умолчанию — 104 Мб, размер может быть изменен через реестр, но предельный размер равен 8 Гб
FFFFFAA000000000	Пул системных PTE (128 Гб) Только для доступа в режиме ядра
FFFFFAC000000000	Пул неподкачиваемой памяти (128 Гб) Только для доступа в режиме ядра Системная область пула неподкачиваемой памяти
FFFFFADFFFFFFFF	
FFFFFFFF80000000 FFFFFFFFFFFFFFFF	Зарезервирован для HAL (2 Гб)

Защита памяти

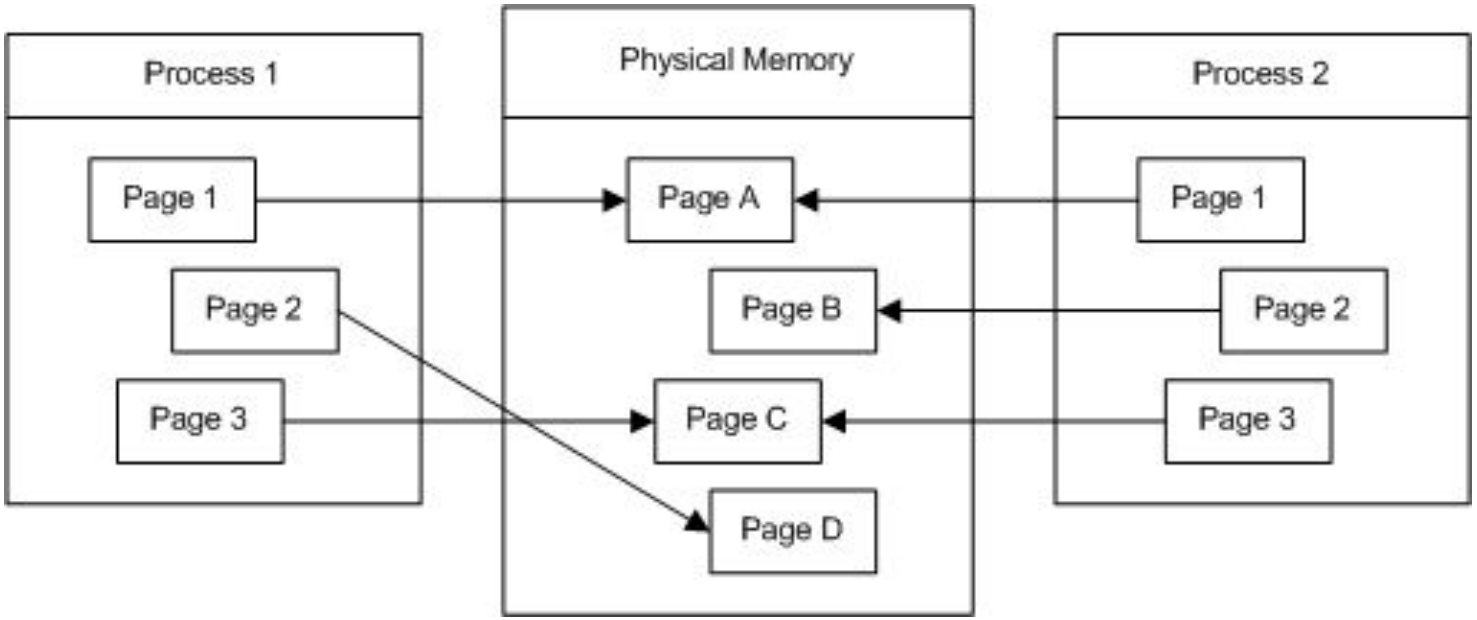
- PAGE_READONLY присваивает доступ «только для чтения» выделенной виртуальной памяти;
- PAGE_READWRITE назначает доступ «чтение-запись» выделенной виртуальной памяти;
- PAGE_WRITECOPY устанавливает доступ «запись копированием» (copy-on-write) выделенной виртуальной памяти.
- PAGE_EXECUTE разрешает доступ «выполнение» выделенной виртуальной памяти. Тем не менее, любая попытка чтения - записи этой памяти приведет к нарушению доступа;
- PAGE_EXECUTE_READ назначает доступ «выполнение» и «чтение»;
- PAGE_EXECUTE_READWRITE разрешает доступ «выполнение», «чтение» и «запись»;
- PAGE_EXECUTE_WRITECOPY присваивает доступ «выполнение», «чтение» и «запись копированием»;
- PAGE_NOACCESS запрещает все виды доступа к выделенной виртуальной памяти.

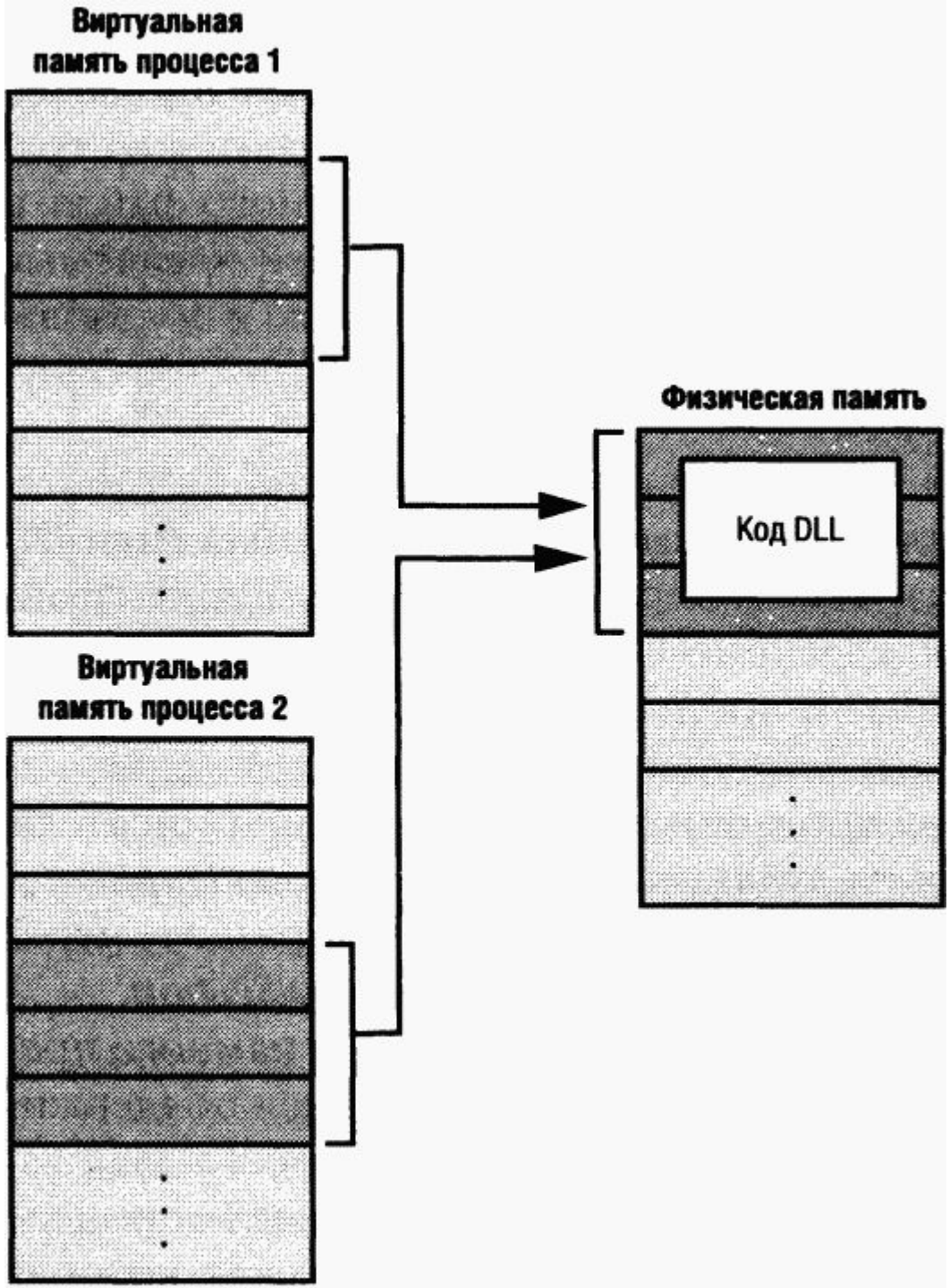
Copy-on-Write Protection

- Позволяет нескольким процессам организовать их карту виртуального адресного пространства (ВАП) таким образом, что они используют **одну** физическую страницу, пока один из процессов не изменяет страницу.
- Эта технология, называемая «*ленивые вычисления*», позволяет системе экономить физическую память и время, выполняя операцию, только когда это станет абсолютно необходимо. Например, два процесса загрузили страницы с одного DLL в свои ВАП. Эти страницы виртуальной памяти отображаются в той же физической странице памяти для обоих процессов. Пока ни один процесс не пишет на эти страницы, они могут отображаться на те же физические страницы.



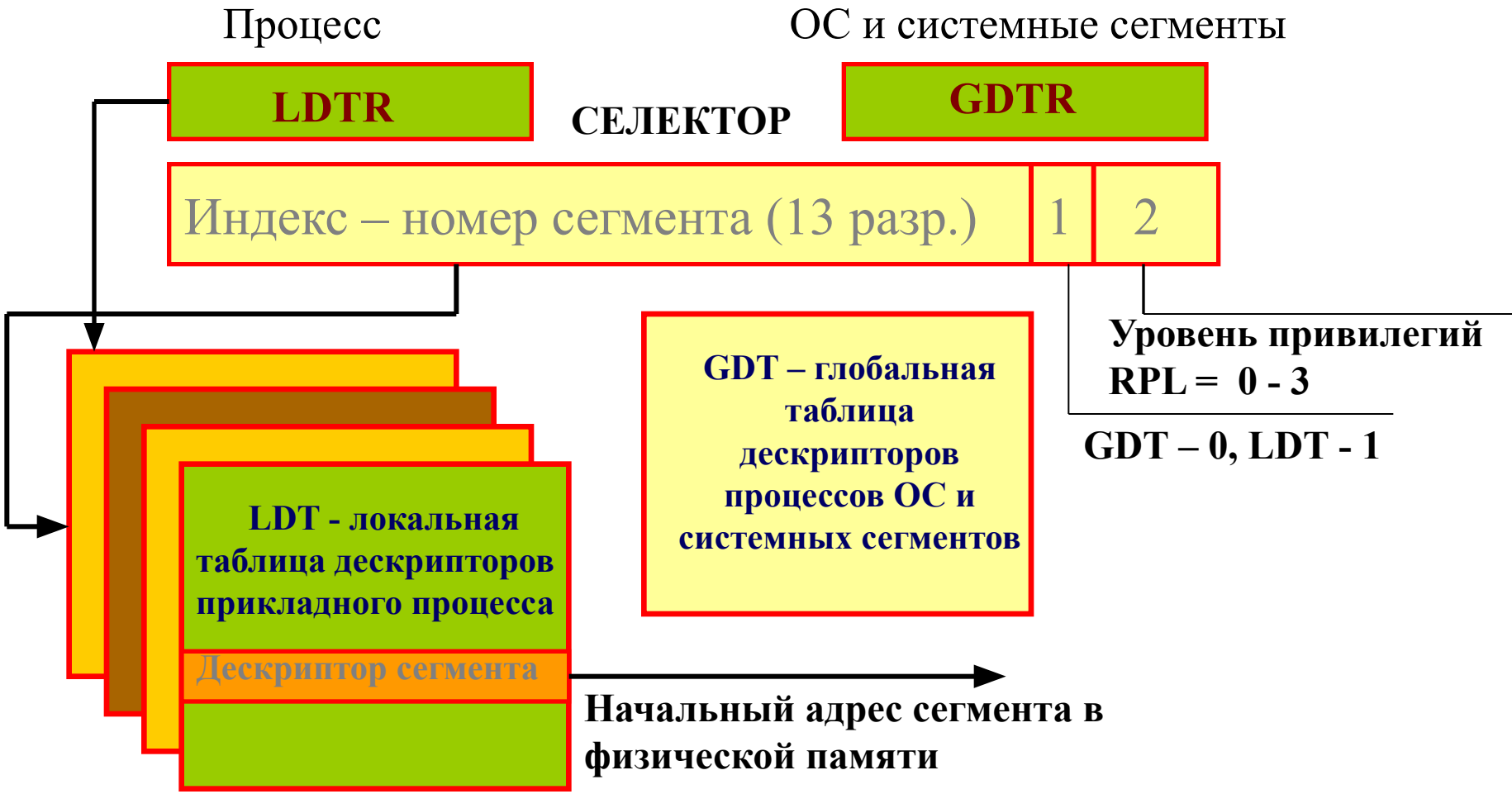
- Если процесс1 записывает в одну из этих страниц, содержание физической страницы будет скопировано на другую физическую страницу и карта виртуальной памяти для процесса1 обновляется. Оба процесса теперь имеют свой собственный экземпляр страницы в физической памяти.



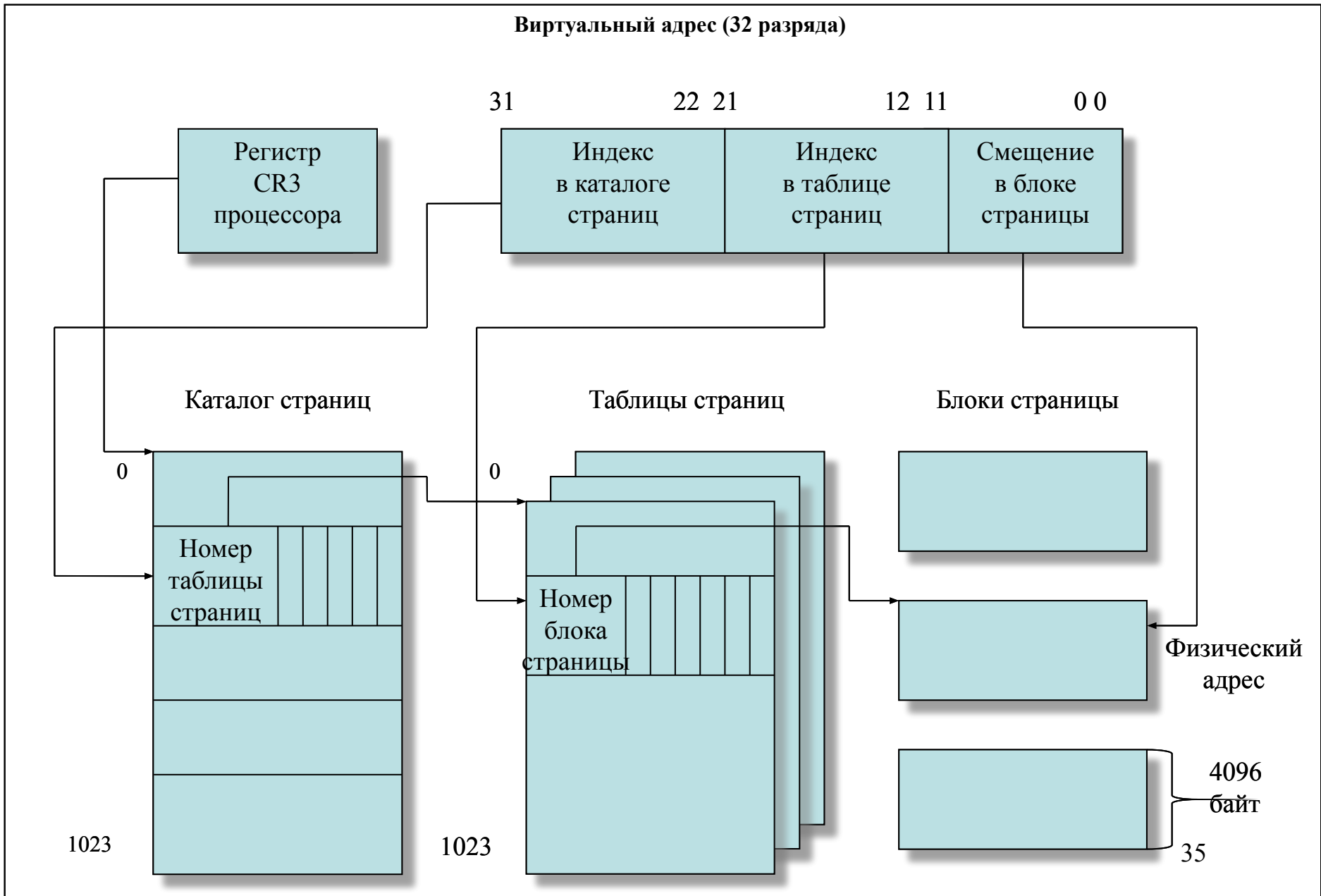


Windows поддерживает механизм разделения памяти.
Разделяемой (shared memory) называется память, видимая более чем одному процессу или присутствующая в виртуальном адресном пространстве более чем одного процесса. Например, если два процесса используют одну и ту же DLL, есть смысл загрузить ее код в физическую память лишь один раз и сделать ее доступной всем процессам, проецирующим эту DLL

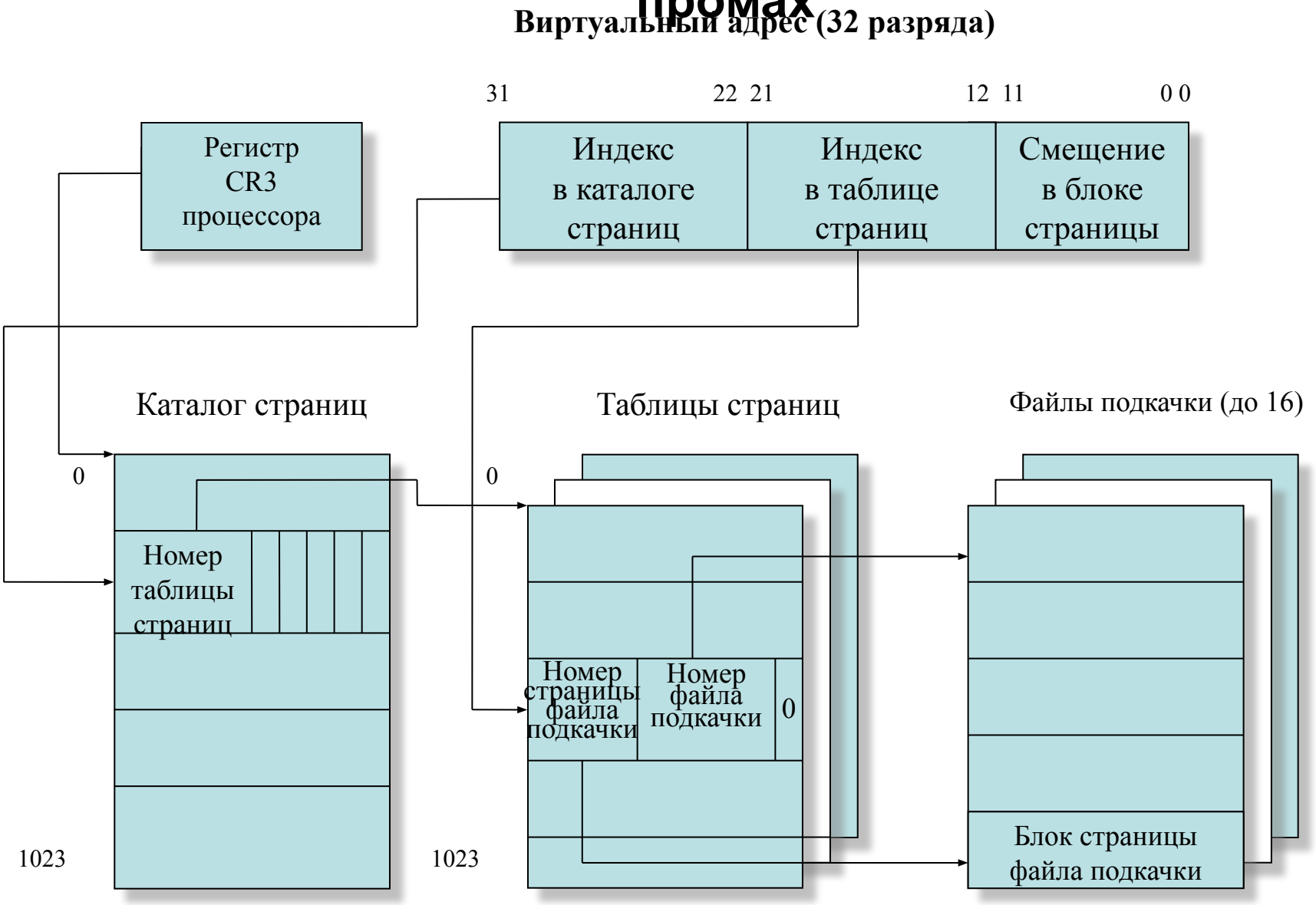
Виртуальная память Windows обеспечивает каждому процессу: 1. 4 Гбайт виртуального адресного пространства (2 Гбайт – ОС, 2 Гбайт – пользовательская программа). 2. 16 К независимых сегментов (8к локальных и 8К глобальных).

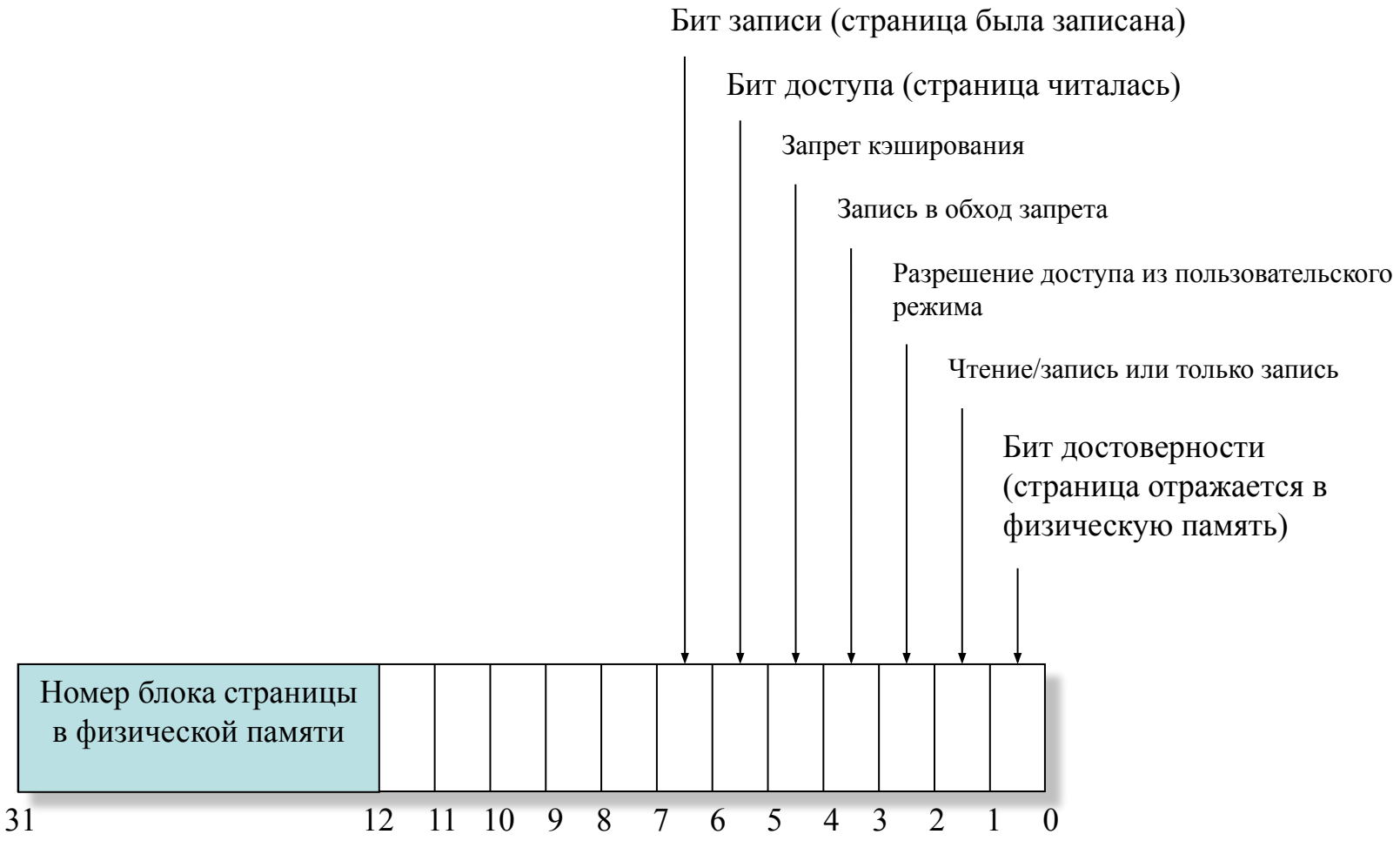


Преобразование виртуальных адресов в физические: попадание



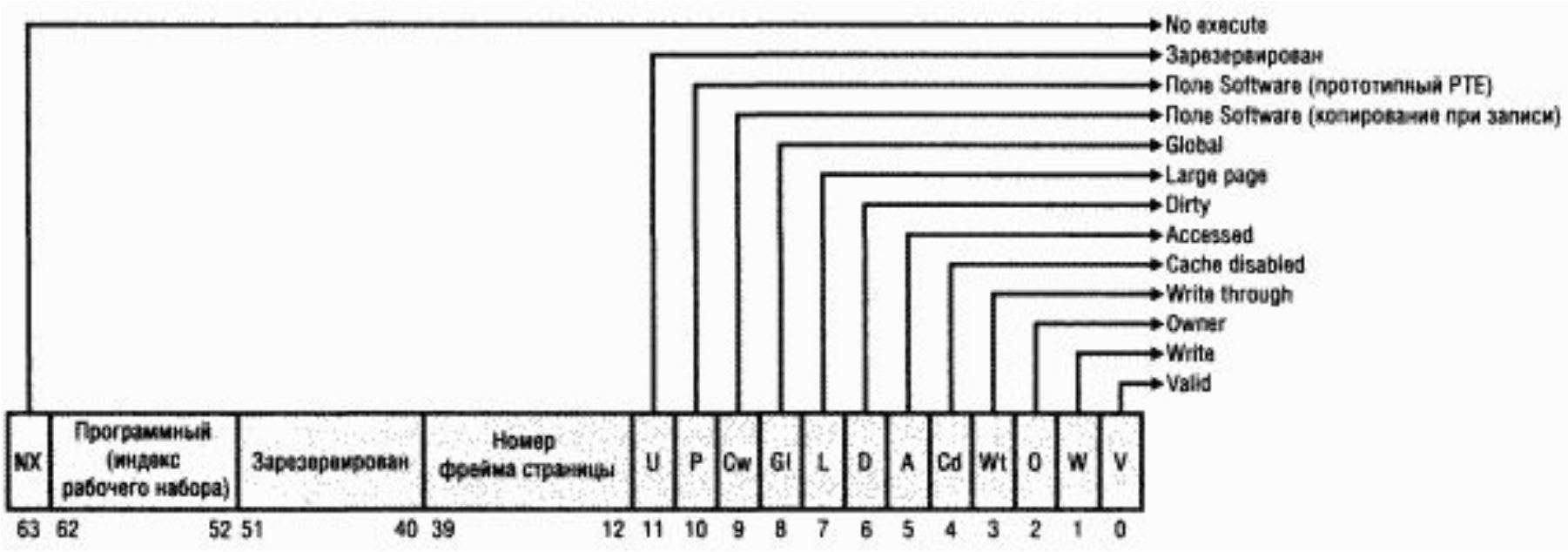
Преобразование виртуальных адресов в физические: промах




Бит записи (страница была записана)
Бит доступа (страница читалась)
Запрет кэширования
Запись в обход запрета
Разрешение доступа из пользовательского режима
Чтение/запись или только запись
Бит достоверности (страница отражается в физическую память)
Номер блока страницы в физической памяти
31
12
11
10
9
8
7
6
5
4
3
2
1
0

Трансляция виртуальных адресов на платформе x64

64-разрядная Windows применяет четырехуровневую схему таблиц страниц. У каждого процесса имеется расширенный каталог страниц верхнего уровня (называемый картой страниц уровня 4), содержащий 512 указателей на структуру третьего уровня — родительский каталог страниц. Каждый родительский каталог страниц хранит 512 указателей на каталоги страниц второго уровня, а те содержат по 512 указателей на индивидуальные таблицы страниц. Наконец, таблицы страниц (в каждой из которых 512 PTE) указывают на страницы в памяти.



Проблемы ручного управления памятью

Традиционным для директивных языков способом управления памятью является ручной:

1. Для создания объекта в динамической памяти программист явно вызывает команду выделения памяти. Эта команда возвращает указатель на выделенную область памяти, который сохраняется и используется для доступа к ней.
2. До тех пор, пока созданный объект нужен для работы программы, программа обращается к нему через ранее сохранённый указатель.
3. Когда надобность в объекте проходит, программист явно вызывает команду освобождения памяти, передавая ей указатель на удаляемый объект.

В любом языке, допускающем создание объектов в динамической памяти, потенциально возможны две проблемы: висячие ссылки и утечки памяти.

Явное освобождение памяти

Программист явно указывает места, где освобождается память: есть операции выделения и освобождения памяти.

	Выделение памяти	Освобождение памяти
C	`malloc`	`free`
C++	`new`	`delete`

Автоматическое управление памятью

Есть только операция выделения памяти, *сборщик мусора* обнаруживает неиспользуемую память и освобождает ее.

Примеры языков:

Java, C#, Lisp, Standard ML, Ruby, Python

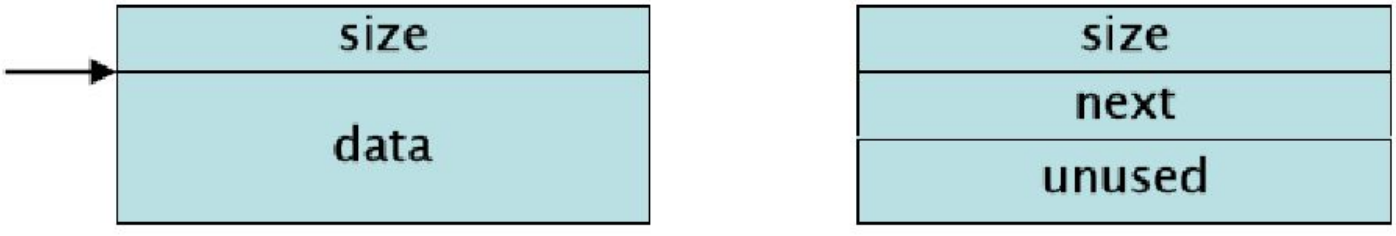
Сборка мусора

- В программировании **сборка мусора** (устоявшийся термин, с точки зрения русского языка правильнее «сбор мусора», англ. garbage collection, **GC**) — одна из форм автоматического управления памятью. Специальный код, называемый сборщиком мусора, периодически освобождает память, удаляя объекты, которые уже не будут востребованы приложением.

Явное управление памятью

метод первого подходящего

Возможное строение объекта:



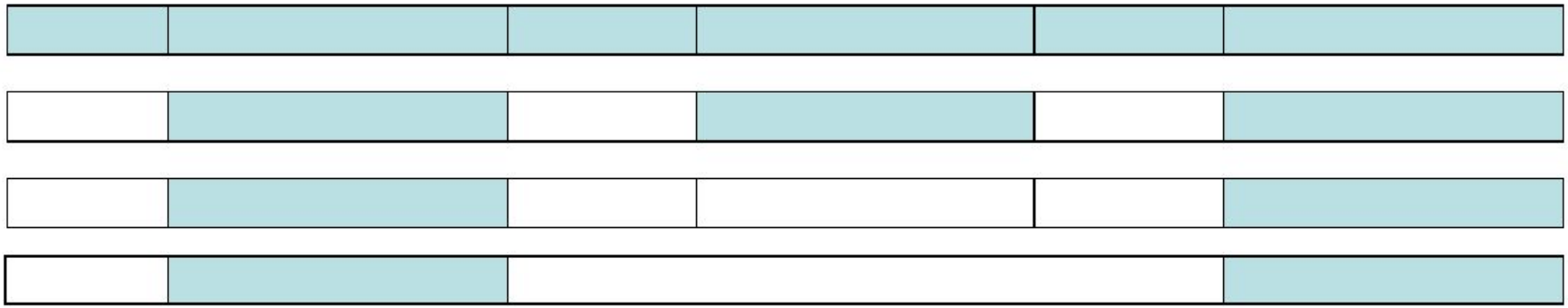
Используемый объект

Свободный объект

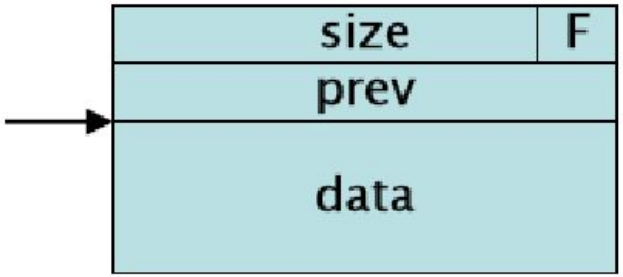
Free-list – список свободных объектов

Метод первого подходящего: проходим по free-list и выбираем первый объект подходящего (не меньшего) размера, остаток помещаем во free-list.

Фрагментация кучи



При освобождении объекта нужно производить слияние между соседними свободными объектами.



F – признак свободен ли объект

prev – предыдущий по адресу объект

Следующий по адресу объект можно получить прибавив size к адресу объекта

Явное освобождение памяти

Программист явно указывает места, где освобождается память: есть операции выделения и освобождения памяти.

	Выделение памяти	Освобождение памяти
C	`malloc`	`free`
C++	`new`	`delete`

Автоматическое управление памятью

Есть только операция выделения памяти, *сборщик мусора* обнаруживает неиспользуемую память и освобождает ее.

Примеры языков:

Java, C#, Lisp, Standard ML, Ruby, Python

Висячая ссылка (англ. dangling pointer)

Висячая ссылка — это оставшаяся в использовании ссылка на объект, который уже удалён. После удаления объекта все сохранившиеся в программе ссылки на него становятся «висячими». Память, занимаемая ранее объектом, может быть передана операционной системе и стать недоступной, или быть использована для размещения нового объекта в той же программе. В первом случае попытка обратиться по «повисшей» ссылке приведёт к срабатыванию механизма защиты памяти и аварийной остановке программы, а во втором — к непредсказуемым последствиям.

Появление висячих ссылок обычно становится следствием неправильной оценки времени жизни объекта: программист вызывает команду удаления объекта до того, как его использование прекратится

Утечка памяти (англ. memory leak)
— процесс неконтролируемого уменьшения объёма свободной оперативной памяти (RAM) компьютера, связанный с ошибками в работающих программах, вовремя не освобождающих ненужные уже участки памяти, или с ошибками системных служб контроля памяти.
Создав объект в динамической памяти, программист может не удалить его после завершения использования. Если ссылающейся на объект переменной будет присвоено новое значение и на объект нет других ссылок, он становится программно недоступным, но продолжает занимать память, поскольку команда его удаления не вызывалась. Такая ситуация и называется утечкой памяти.

Рассмотрим следующий фрагмент кода на C++:

```
/*1*/ char *pointer = NULL;
/*2*/ for( int i = 0; i < 10; i++ ) {
/*3*/   pointer = new char[100];
/*4*/ }
/*5*/ delete [] pointer;
```

В этом примере на 3-й строке создается объект в динамической памяти. Код на 3-й строке выполняется 10 раз, причём каждый следующий раз адрес нового объекта перезаписывает значение, хранящееся в указателе pointer. На 5-й строке выполняется удаление объекта, созданного на последней итерации цикла. Однако первые 9 объектов остаются в динамической памяти, и одновременно в программе не остаётся переменных, которые бы хранили адреса этих объектов. Т.е. в 5-й строке невозможно ни получить доступ к первым 9 объектам, ни удалить их.

Достижимость объекта

- определённое множество объектов считается достижимым изначально — корневые объекты, обычно в их число включают все глобальные переменные и объекты, на которые есть ссылки в стеке вызовов;
- любой объект, на который есть ссылка из достижимого объекта, тоже считается достижимым.

«Алгоритм пометок» (Mark and Sweep):

- для каждого объекта хранится бит, указывающий, достижим ли этот объект из программы или нет;
- изначально все объекты, кроме корневых, помечаются как недостижимые;
- рекурсивно просматриваются и помечаются как достижимые объекты ещё не помеченные и до которых можно добраться из корневых объектов по ссылкам;
- те объекты, у которых бит достижимости не был установлен, считаются недостижимыми.

Следует обратить внимание, что, согласно данному алгоритму, если два или более объектов ссылаются друг на друга, но ни на один из этих объектов нет других ссылок, то есть имеет место циклическая ссылка, то вся группа считается недостижимой. Эта особенность алгоритма позволяет гарантированно удалять группы объектов, использование которых прекратилось, но в которых имеются ссылки друг на друга. Такие группы часто называются «islands of isolation» (острова изоляции)

Автоматическое управление памятью: tracing GC

- Объект – *живой*, если он
  (1) корневой, либо
  (2) существует живой объект, ссылающийся на него.

Используемый $\Rightarrow$ живой

Не живой $\Rightarrow$ не используемый

Классический “mark and sweep” алгоритм сборки мусора

- На время сборки мусора работа приложения останавливается (“stop-the-world”)

- 2 фазы:

  маркировка (помечаются живые объекты)

  выметание (непомеченные объекты освобождаются)

Фаза маркировки (mark phase): начальная разметка

Размечаем все корневые объекты и помещаем размеченные объекты в стек разметки

Фаза маркировки: замыкание множества размеченных объектов.

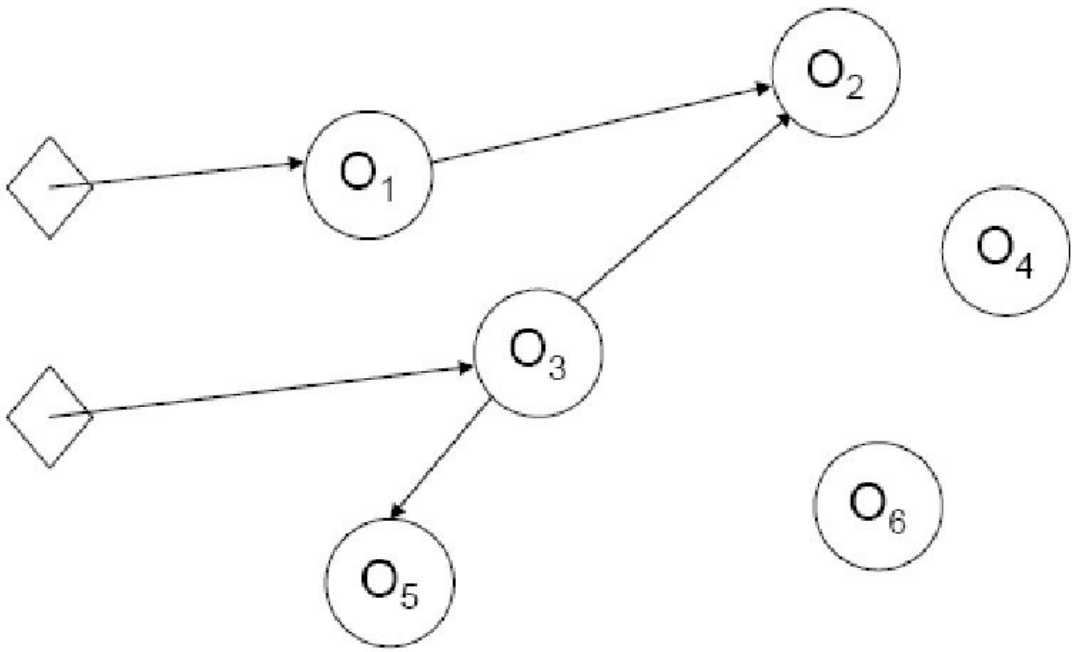
Пока стек разметки не пуст

- достаем из стека очередной объект;
- размечаем все объекты на которые он ссылается;
- помещаем только что размеченные объекты в стек.

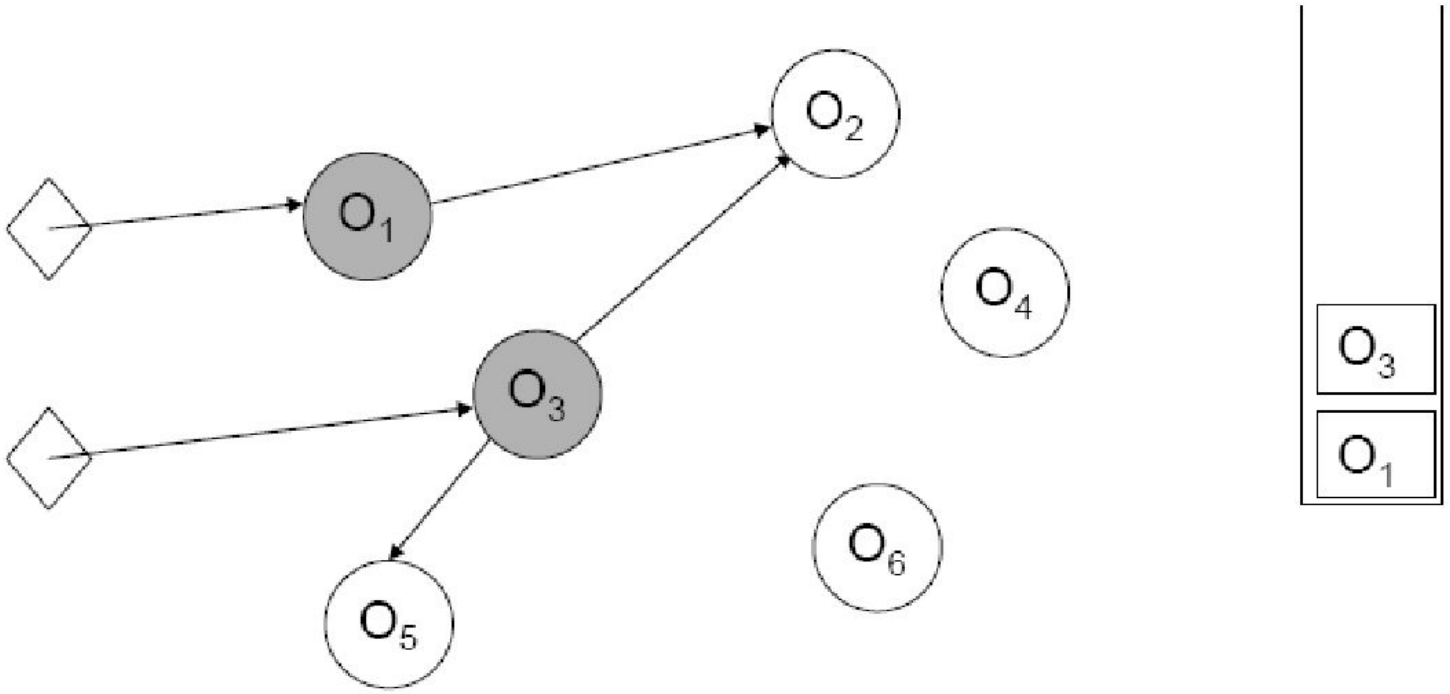
Нотация Дейкстры

- Объект *белого* цвета еще не был обработан.
- Объект *серого* цвета был помечен, однако возможно еще не все объекты, на которые он ссылается (*потомки*) помечены.
- Объект *черного* цвета помечен, и все его потомки также помечены.

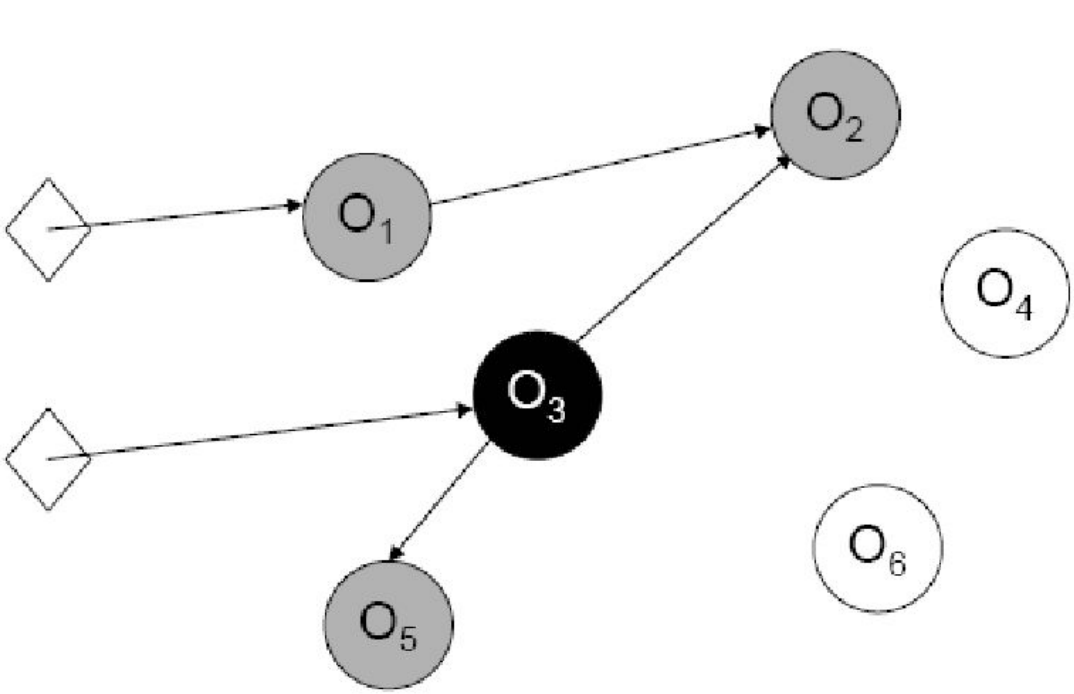
Разметка

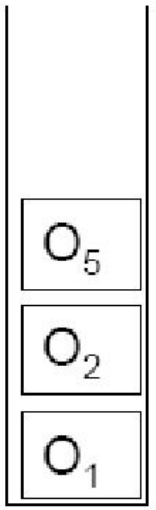


Разметка



Разметка



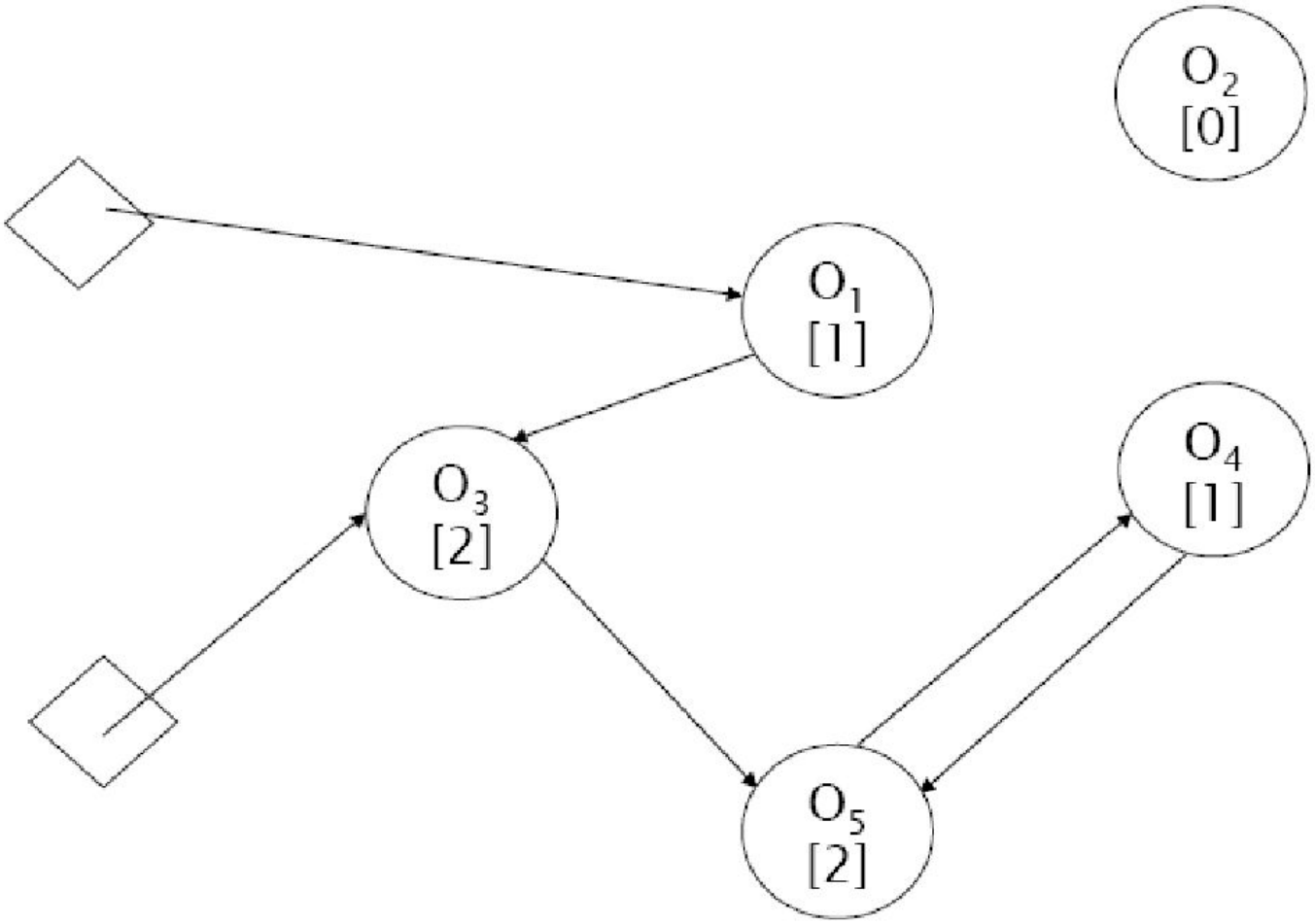


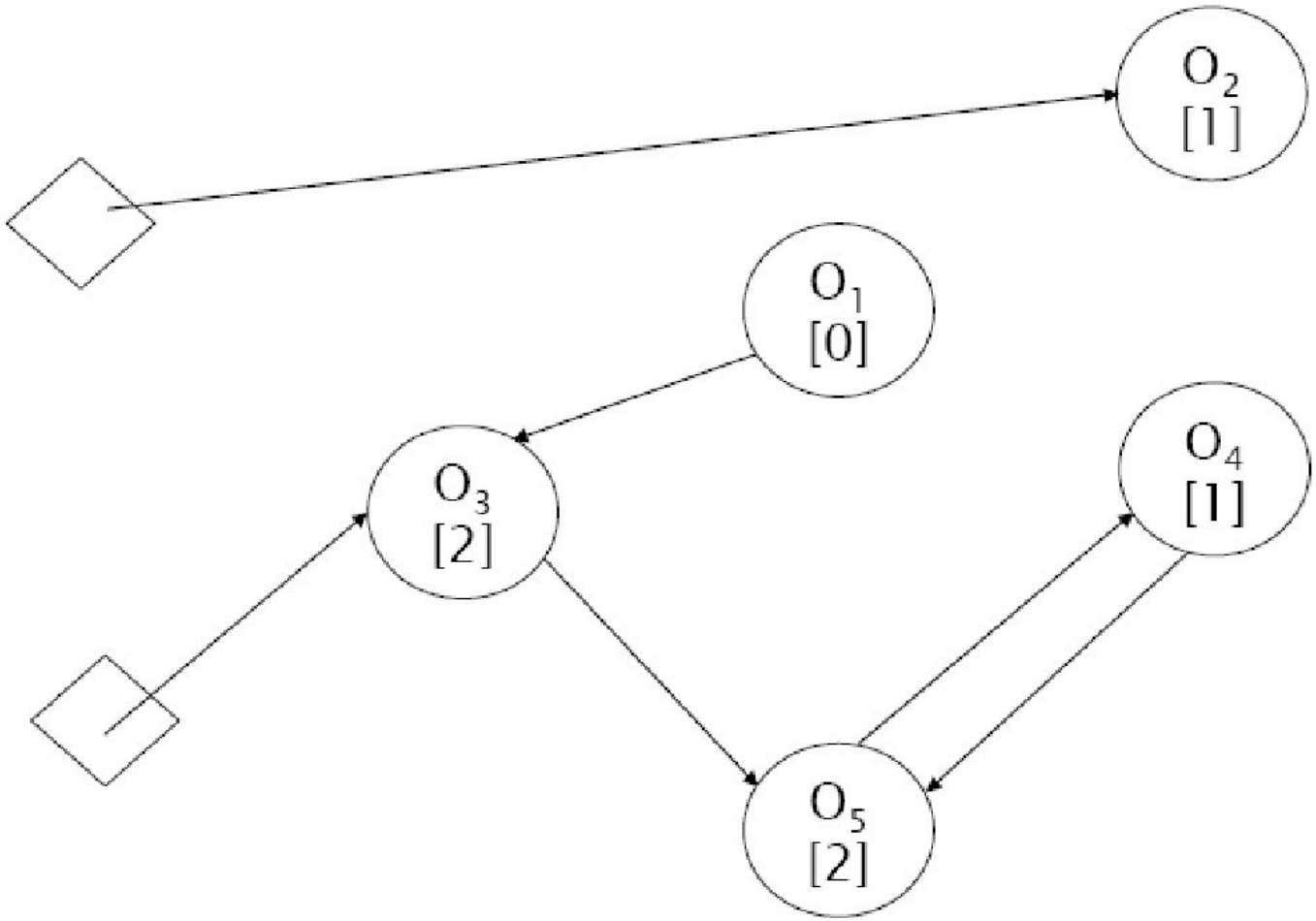
Алгоритм подсчёта ссылок

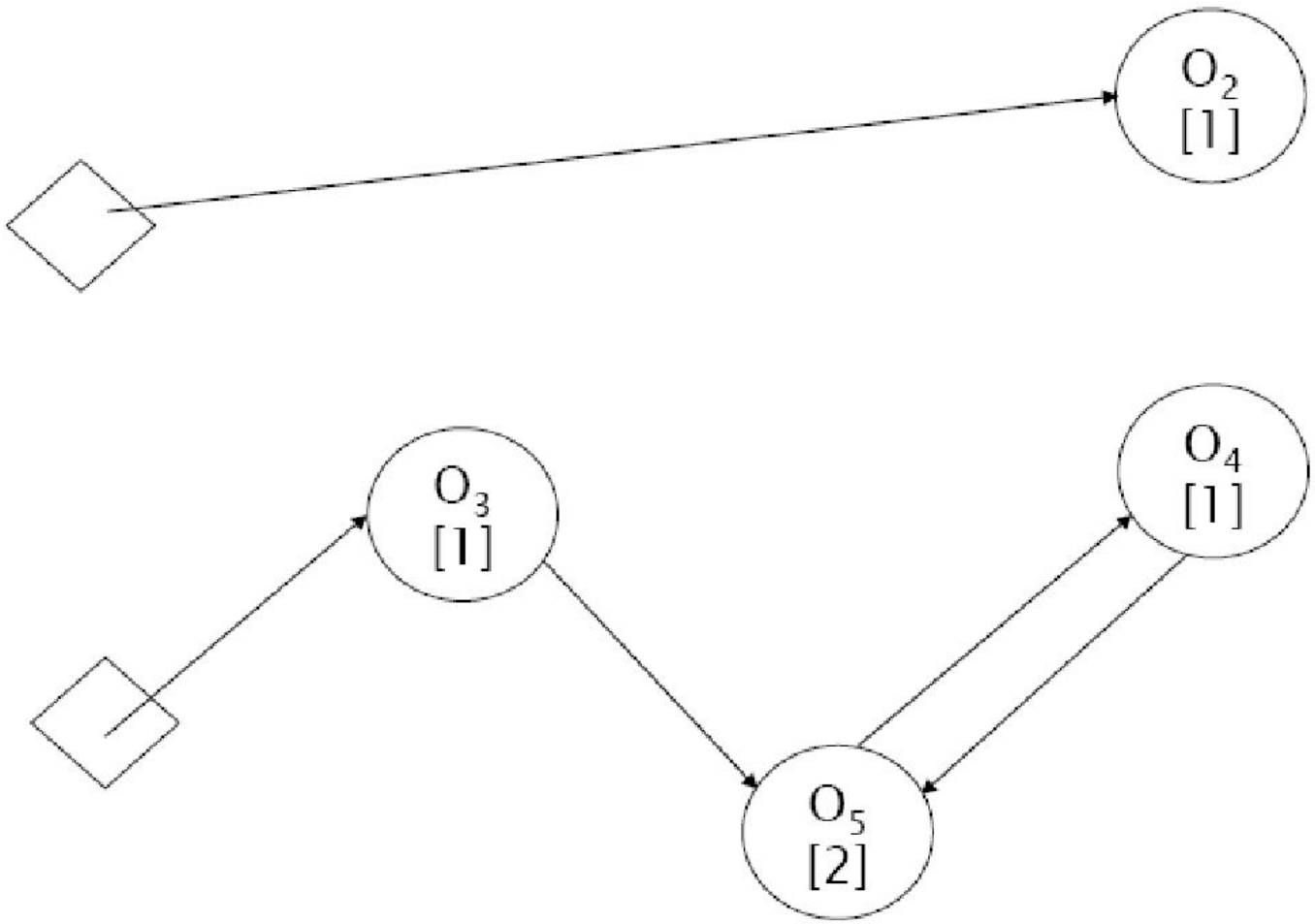
Другой вариант алгоритма определения достижимости — обычный подсчёт ссылок. Его использование замедляет операции присваивания ссылок, но зато определение достижимых объектов тривиально — это все объекты, значение счётчика ссылок которых превышает нуль. Без дополнительных уточнений этот алгоритм, в отличие от предыдущего, не удаляет циклически замкнутые цепочки вышедших из употребления объектов, сохранивших ссылки друг на друга.

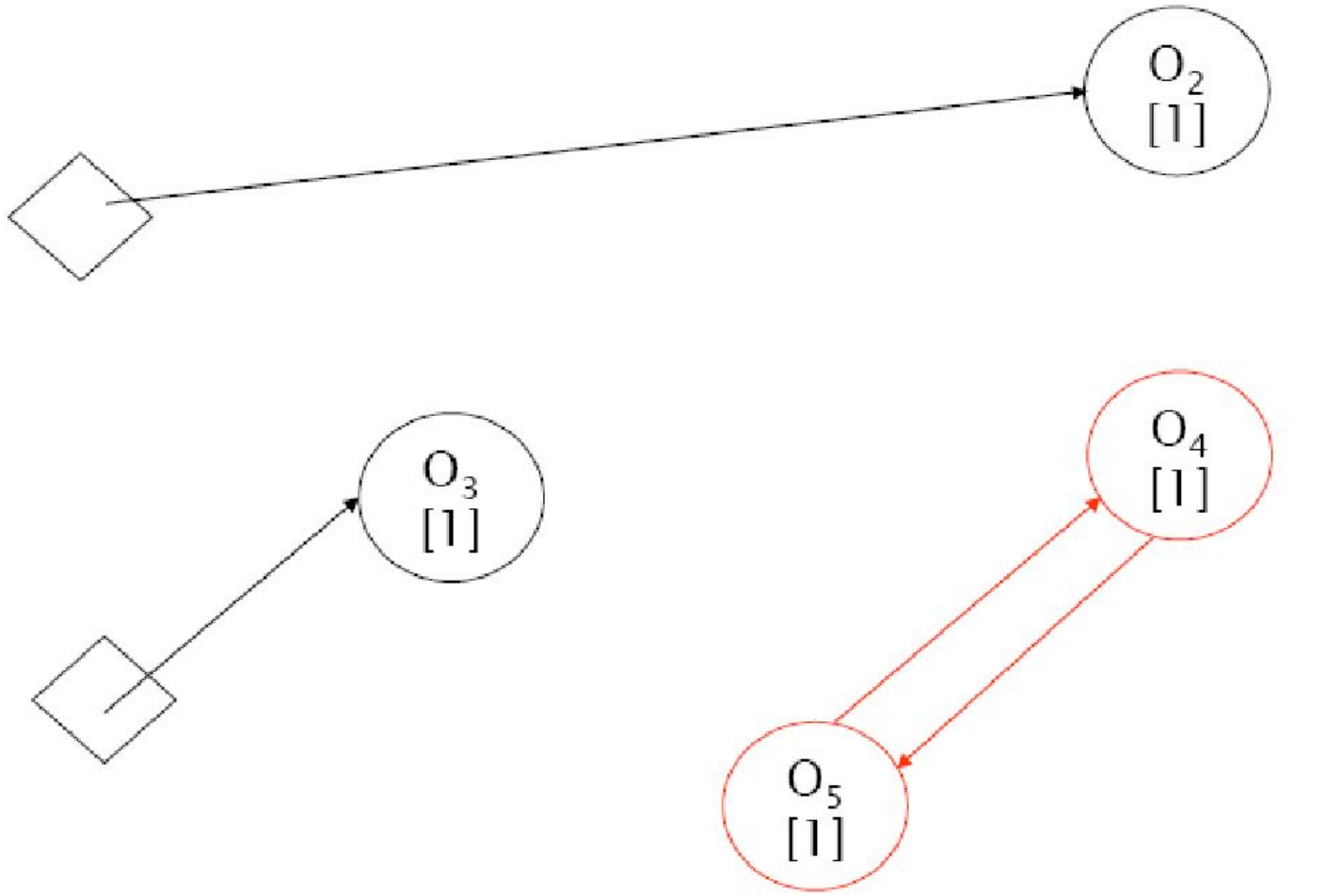
Автоматическое управление памятью: подсчет ссылок

- При создании, с каждым объектом ассоциируется счетчик ссылок на этот объект
- При модификации указателей счетчики корректируются
- Если счетчик достигает 0, объект освобождается


O2
[0]
O1
[1]
O3
[2]
O4
[1]
O5
[2]


O2
[1]
O1
[0]
O3
[2]
O4
[1]
O5
[2]


O2
[1]
O3
[1]
O4
[1]
O5
[2]



Циклический мусор

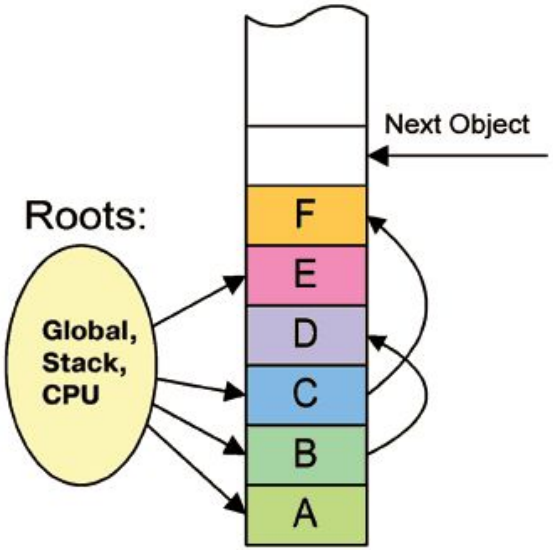
Как только определено множество *недостижимых* объектов, сборщик мусора может освободить память, занимаемую ими.

- **Неперемещающий сборщик мусора** быстро освобождает память (поскольку эта операция сводится к пометке соответствующих блоков памяти как свободных), но тратит больше времени на её выделение (поскольку память фрагментируется и при выделении необходимо найти в карте памяти нужное количество блоков подходящего размера).
- **Перемещающий сборщик** требует сравнительно больше времени на этапе сборки мусора (тратится дополнительное время на дефрагментацию памяти и изменение всех ссылок на перемещаемые объекты), зато перемещение позволяет использовать чрезвычайно простой и быстрый алгоритм выделения памяти. При дефрагментации объекты передвигаются так, чтобы разделить всю память на две большие области — занятую и свободную, и сохраняется указатель на их границу. Для выделения новой памяти достаточно лишь переместить эту границу, вернув кусок из начала свободной памяти.

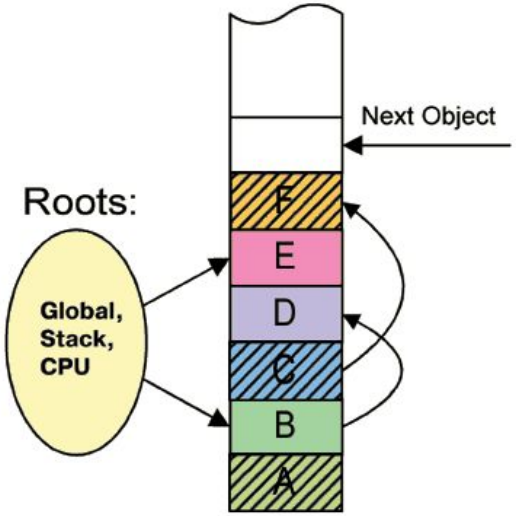
- Для обеспечения высокой скорости доступа к объектам в динамической памяти, объекты, поля которых используются совместно, перемещающий сборщик позволяет размещать в памяти недалеко друг от друга. Тогда они вероятнее окажутся в кэше процессора одновременно, что уменьшит количество обращений к относительно медленному ОЗУ.
- Многие современные сборщики мусора подразделяют все объекты на несколько поколений — серий объектов с близким временем существования. Как только память, выделенная одному из поколений, заканчивается, в этом поколении и во всех более «молодых» производится поиск недостижимых объектов. Все они удаляются, а оставшиеся переводятся в более «старое» поколение.

СБОРЩИК МУСОРА C#

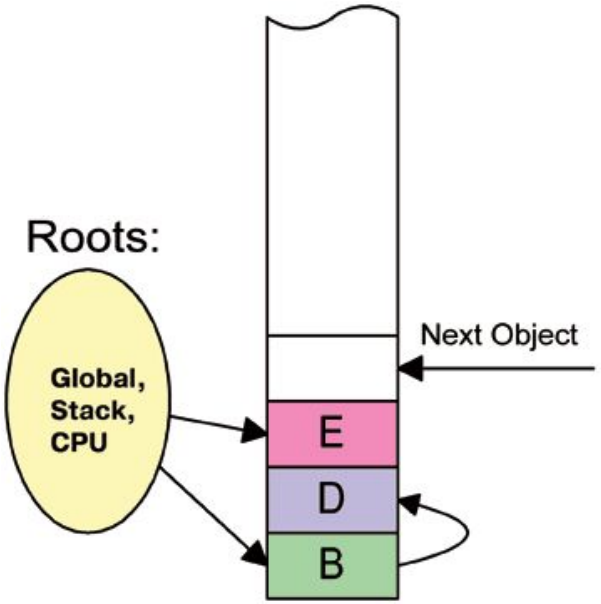
- Алгоритм работы сборщика мусора (garbage collector, GC), являющегося частью CLR, подробно описан в книге Джефри Рихтера «Applied Microsoft .NET Framework Programming».
- Для хранения объектов CLR использует хип, подобный хипу C++, за тем важным исключением, что хип CLR не фрагментирован.
- Выделение объектов производится всегда один за другим в последовательных адресах памяти, что позволяет существенно повысить производительность всего приложения. Когда память заканчивается, в дело вступает процесс, основная задача которого сводится к тому, чтобы освободить место в хипе путём дефрагментации неиспользуемой памяти.
- Первое, что делает GC во время сборки мусора – это принимает решение о том, что все выделенные блоки памяти в программе - это как раз и есть мусор. Далее начинается поиск «живых» указателей на объекты. Microsoft называет эти указатели «roots».
- Найдя «живой» указатель, сборщик мусора помечает объект, на который этот указатель указывает, как всё ещё используемый программой и запрашивает информацию о его структуре, чтобы в свою очередь просканировать уже этот объект на наличие указателей на другие объекты. И так далее, пока все указатели в программе не будут обработаны.



Начальное состояние



Heap после завершения работы с некоторыми объектами



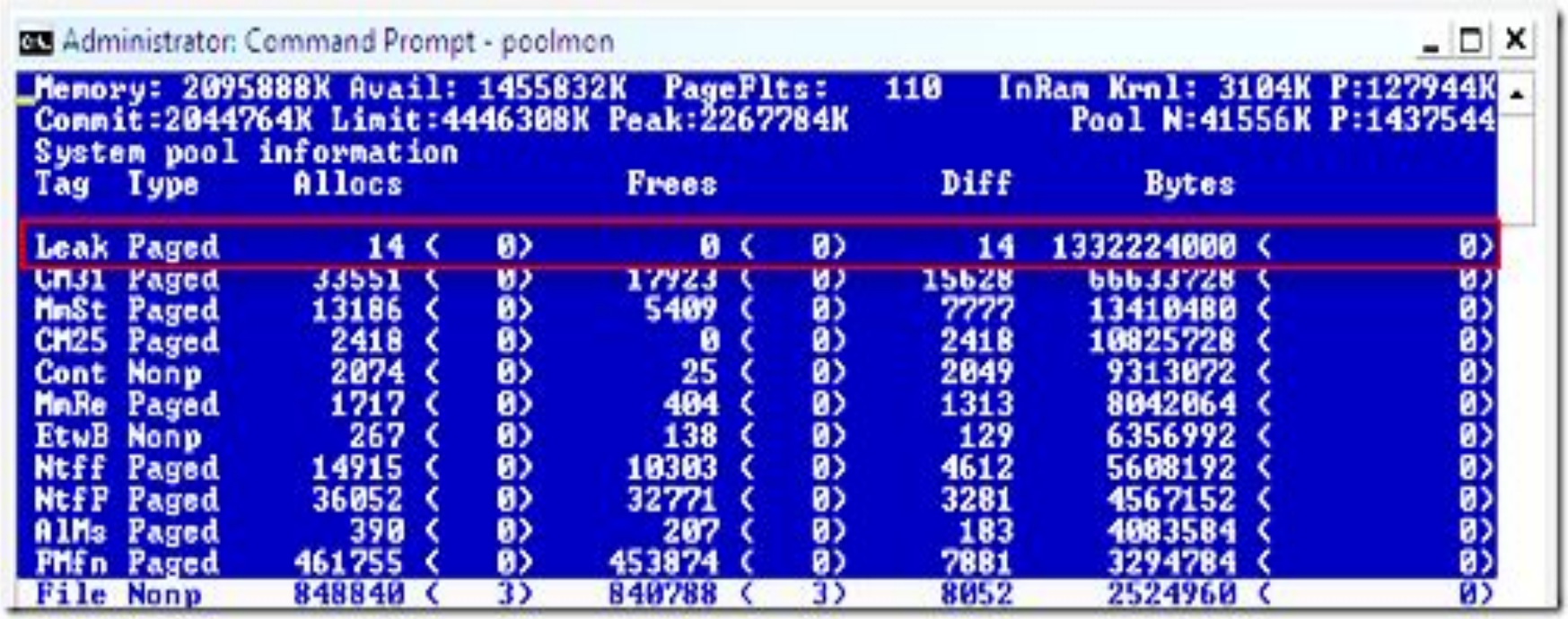
Heap после сборки мусора

Алгоритмы работы GC построены во многом исходя из правил, полученных статистическим и опытным путём.
В частности, одно из таких правил утверждает, что только что созданные «молодые» объекты имеют наиболее короткое время жизни, а живущие уже давно будут жить ещё долго. Именно в соответствии с этим правилом в CLR существуют понятие **поколений** (generations).
Объекты, выжившие после первой сборки мусора и дефрагментации, объявляются первым поколением. В следующий раз, те объекты из первого поколения, которые опять смогли выжить, перемещаются во второе поколение и уже больше не трогаются, выжившие объекты из нулевого поколения перемещаются в первое.
Например, **объекты, пережившие две сборки мусора, остаются в хипе навсегда и GC не занимается их дефрагментацией**.

Отслеживание утечек пула

Poolmon - инструмент из набора **Windows Driver Kit** , показывает число выделенных областей и количество незанятых байтов в этих областях, разделенных по типу пула, а также теги запросов **ExAllocatePoolWithTag**.

Снимок, демонстрирующий работу **Poolmon** на системе, где с помощью **Notmyfault** была организована утечка **14** участков памяти пула, каждый примерно по **100Мб**:



Утилита **Strigs** от **Sysinternals** сохраняет искомые строки в указанном вами файле (искомая строка по умолчанию должна быть не короче 3-х символов, и, так как большинство образов драйверов находятся в директории **%Systemroot%\System32\Drivers**, вы можете открыть командную строку, изменить текущую директорию на указанную и выполнить команду "**strings * | findstr <tag>**". После того, как вы найдете соответствия, вы можете получить информацию о версии драйвера с помощью утилиты **Sigcheck** от **Sysinternals**

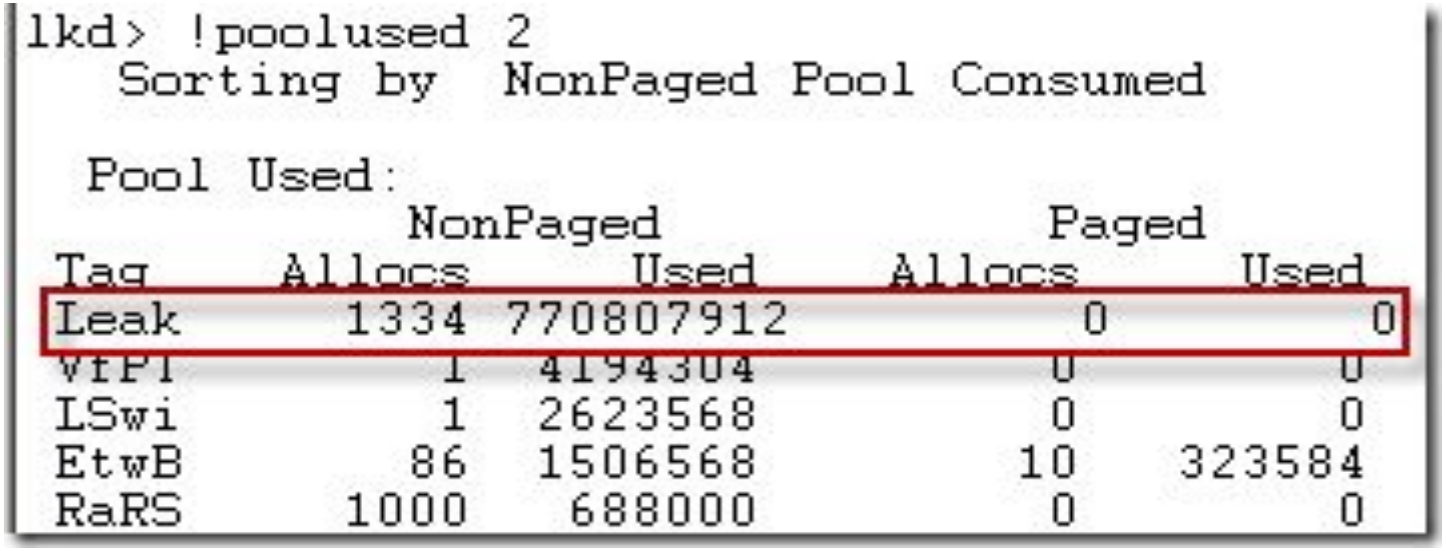
Вот как выглядит процесс поиска драйвера, использующего тег "**Leak**":

```
C:\Windows\System32\drivers>strings * | findstr Leak
C:\Windows\System32\drivers\myfault.sys: Leak

C:\Windows\System32\drivers>sigcheck myfault.sys

sigcheck v1.60 - sigcheck
Copyright (C) 2004-2008 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Windows\System32\drivers\myfault.sys:
        Verified:       Signed
        Signing date:   3:43 PM 3/25/2009
        Strong Name:    Unsigned
        Publisher:      Sysinternals
        Description:    Crash Test Driver
        Product:        Sysinternals Myfault
        Version:        2.0
        File version:   2.0 built by: WinDDK

C:\Windows\System32\drivers>
```

Вот результат запуска команды **!vm** на системе, в которой **Notmyfault** исчерпал невыгружаемый пул:

```
lkd> !vm

*** Virtual Memory Usage ***
      Physical Memory:      262010 (   1048040 Kb)
      Page File: \??\C:\pagefile.sys
        Current:   1355240 Kb  Free Space:   1341636 Kb
        Minimum:   1355240 Kb  Maximum:      4193280 Kb
      Available Pages:       18144 (     72576 Kb)
      ResAvail Pages:        56535 (    226140 Kb)
      Locked IO Pages:           0 (         0 Kb)
      Free System PTEs:     273266 (   1093064 Kb)
      Modified Pages:         1056 (      4224 Kb)
      Modified PF Pages:      1029 (      4116 Kb)
      NonPagedPool Usage:   191968 (    767872 Kb)
      NonPagedPool Max:     193023 (    772092 Kb)
      ********** Excessive NonPaged Pool Usage *****
      PagedPool 0 Usage:      3620 (     14480 Kb)
```

Как только вы убедитесь в наличии утечки, воспользуйтесь командой **!poolused**, чтобы просмотреть информацию об использовании пула, как это было в **Poolmon**. По умолчанию эта команда выдает несортированные данные, так что используйте параметр **1** для сортировки по использованию выгружаемого пула, и **2** - для сортировки по использованию невыгружаемого пула:



```
lkd> !poolused 2
   Sorting by  NonPaged Pool Consumed

  Pool Used:
               NonPaged                  Paged
 Tag     Allocs      Used     Allocs      Used
 Leak      1334 770807912          0         0
 VtPl         1   4194304          0         0
 LSwi         1   2623568          0         0
 EtwB        86   1506568         10    323584
 RaRS      1000    688000          0         0
```