Виды ПО

ПО

Прикладное

Системное

Утилиты

Библиотеки

Трансляторы

Демоны

ОС

Структура ПО

Прикладное ПО

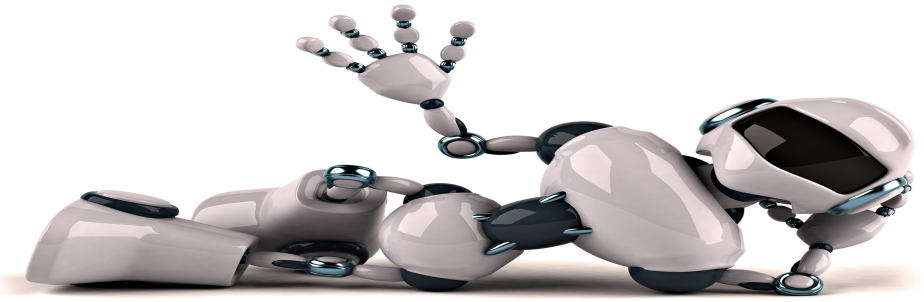
Системное ПО

ОС

АППАРАТУРА

Функции ОС

- запуск и исполнение программ
- управление памятью
- работа с периферийными устройствами
- программный интерфейса
- параллельное выполнение программ
- организация взаимодействия процессов
- защита системных и пользовательских ресурсов
- аутентификация и авторизация



Дисковые ОС

Выполняет первые 4 функции.

Загружает пользовательскую программу в память и передает ей управление, после чего программа делает с системой все, что ей заблагорассудится.

MS DOS, CP/M, Win 95/98

Универсальные ОС

К этому классу относятся системы, берущие на себя выполнение всех вышеперечисленных функций. Разделение на ОС и ДОС идет, по-видимому, от систем IBM DOS/360 и OS/360 для больших компьютеров этой фирмы.

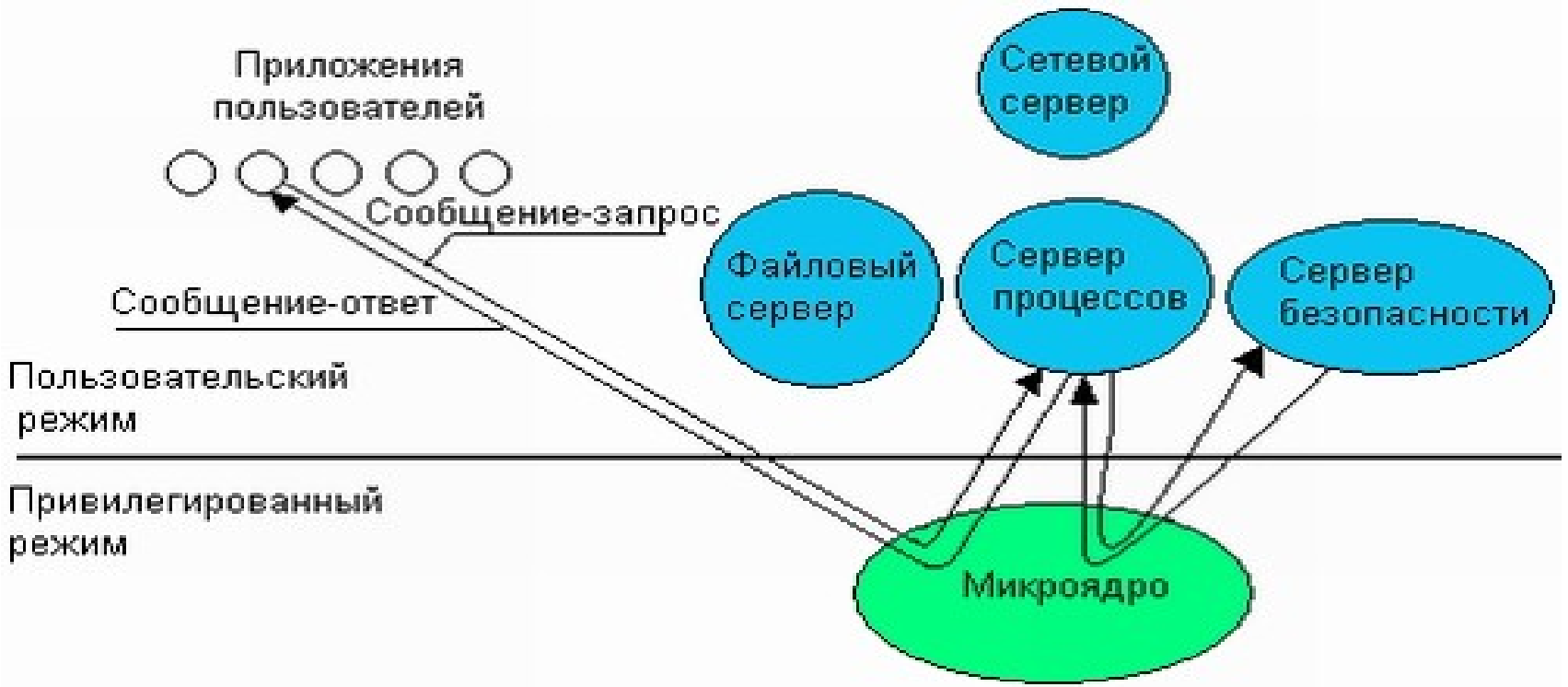
Win NT/2000/XP, Vista/7/8/2008...

OS/2, Unix

Виды ядер

- Монолитное
- Микроядро
- Наноядро
- Гибридное
- Экзоядро

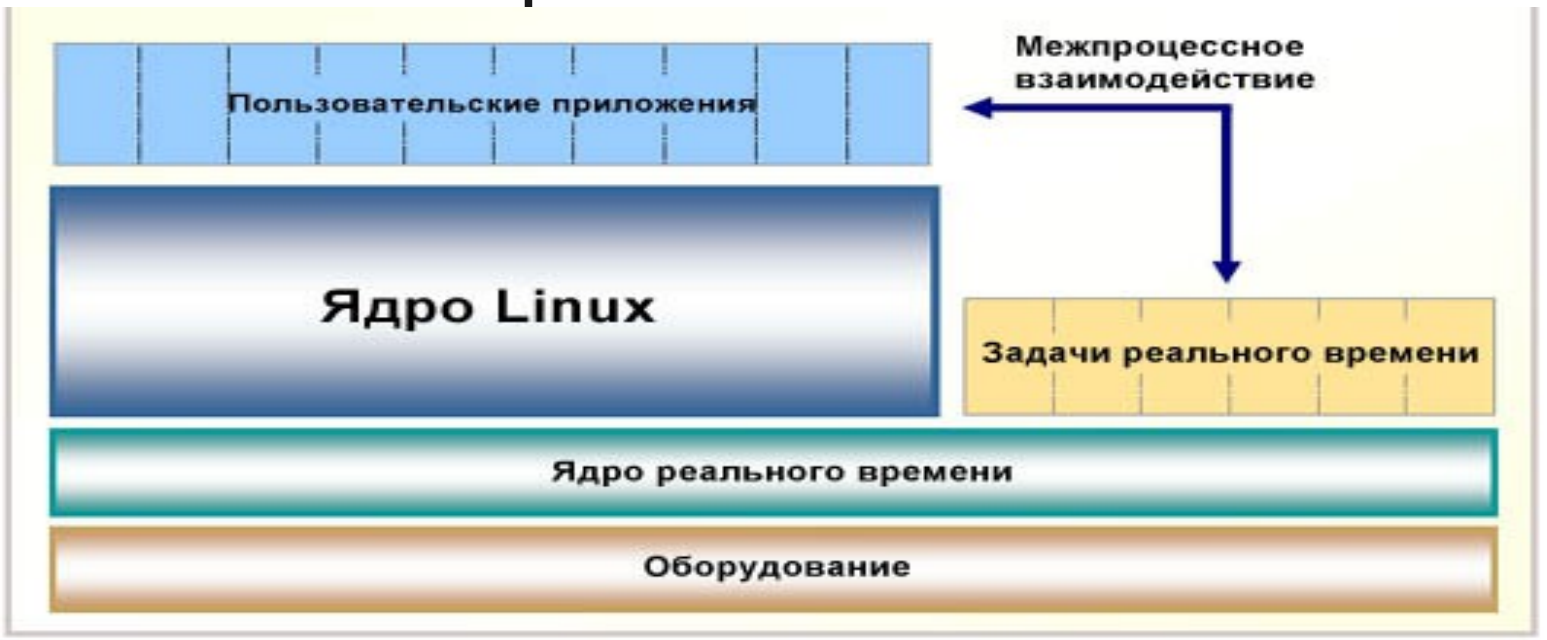
Микроядерные ОС



QNX, Darwin (Mac OS X, iOS), Mach (Hurd, Nextstep)

ОС реального времени

Предназначены для облегчения разработки так называемых приложений реального времени — программ, управляющих некомпьютерным по природе оборудованием, часто с очень жесткими ограничениями по времени.



QNX, ОСРВ, Linux

Виртуальные машины

- Эмуляторы ОС
- Виртуальные машины
- Контейнеры (виртуальное окружение)

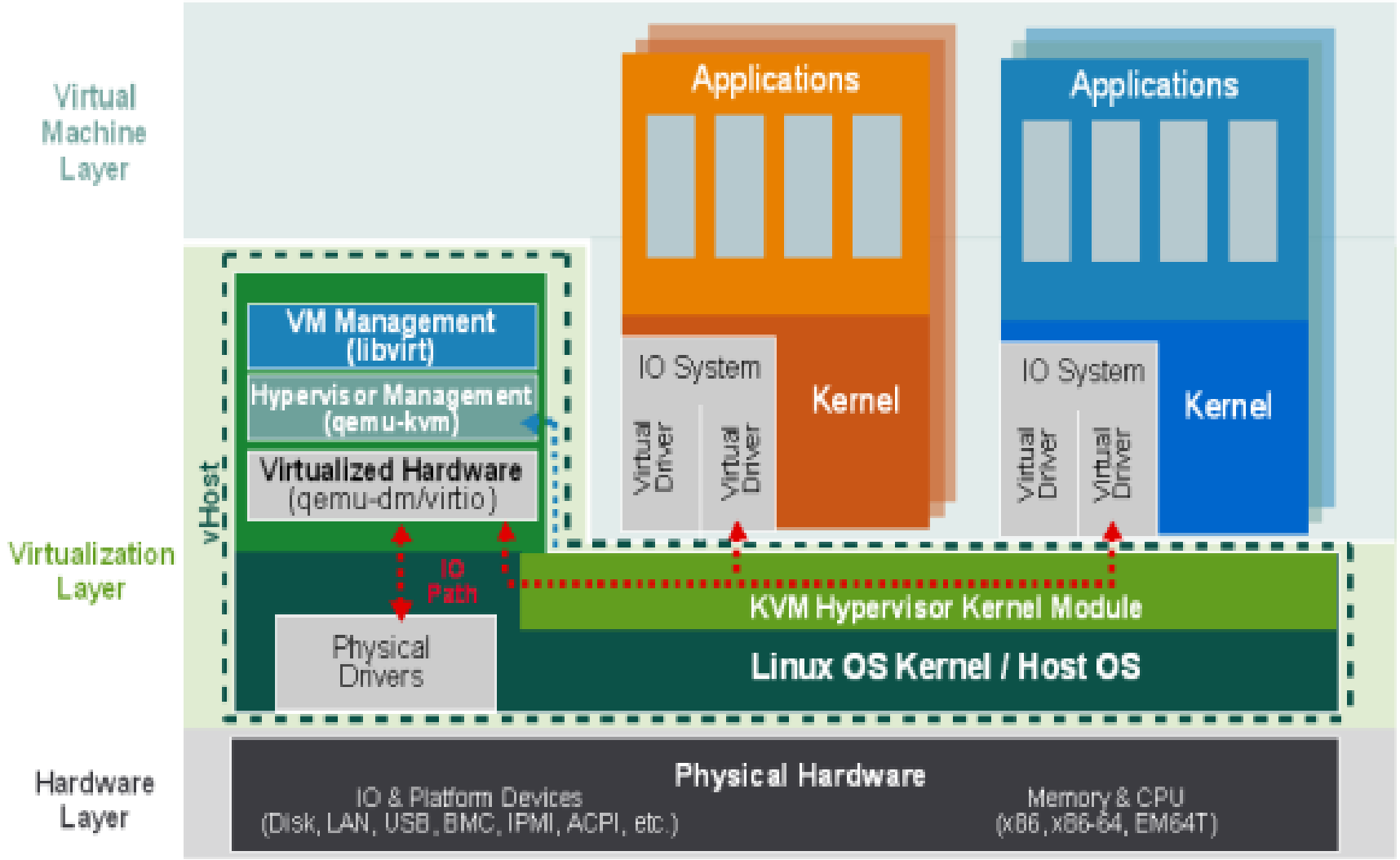
Эмуляторы ОС

Запуск приложения откомпилированной для одной ОС в другой ОС

Wine
Is
Not
Emulator

Виртуальные машины

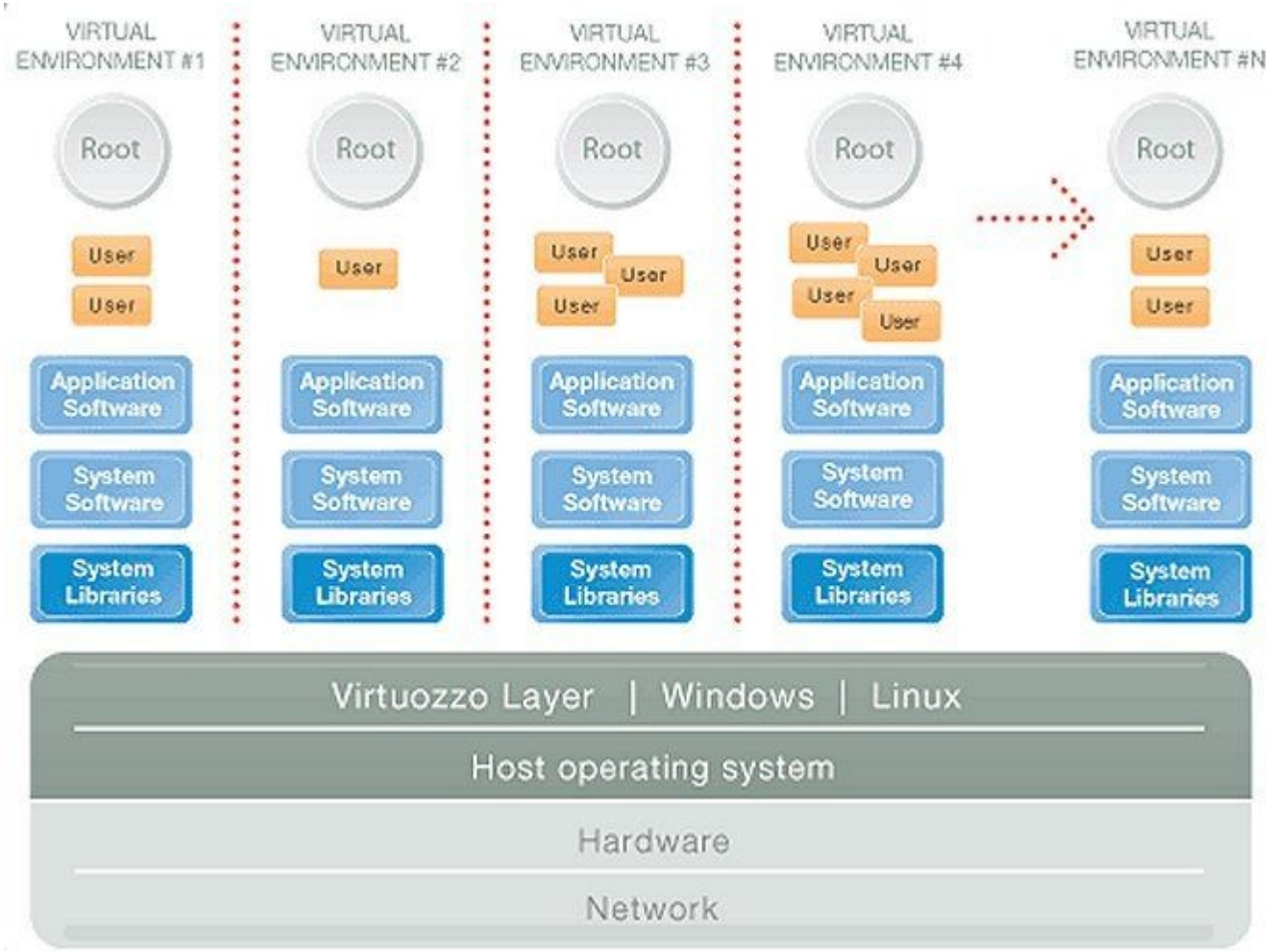
KVM



KVM, XEN, VM Ware, Hyper V

Контейнеры

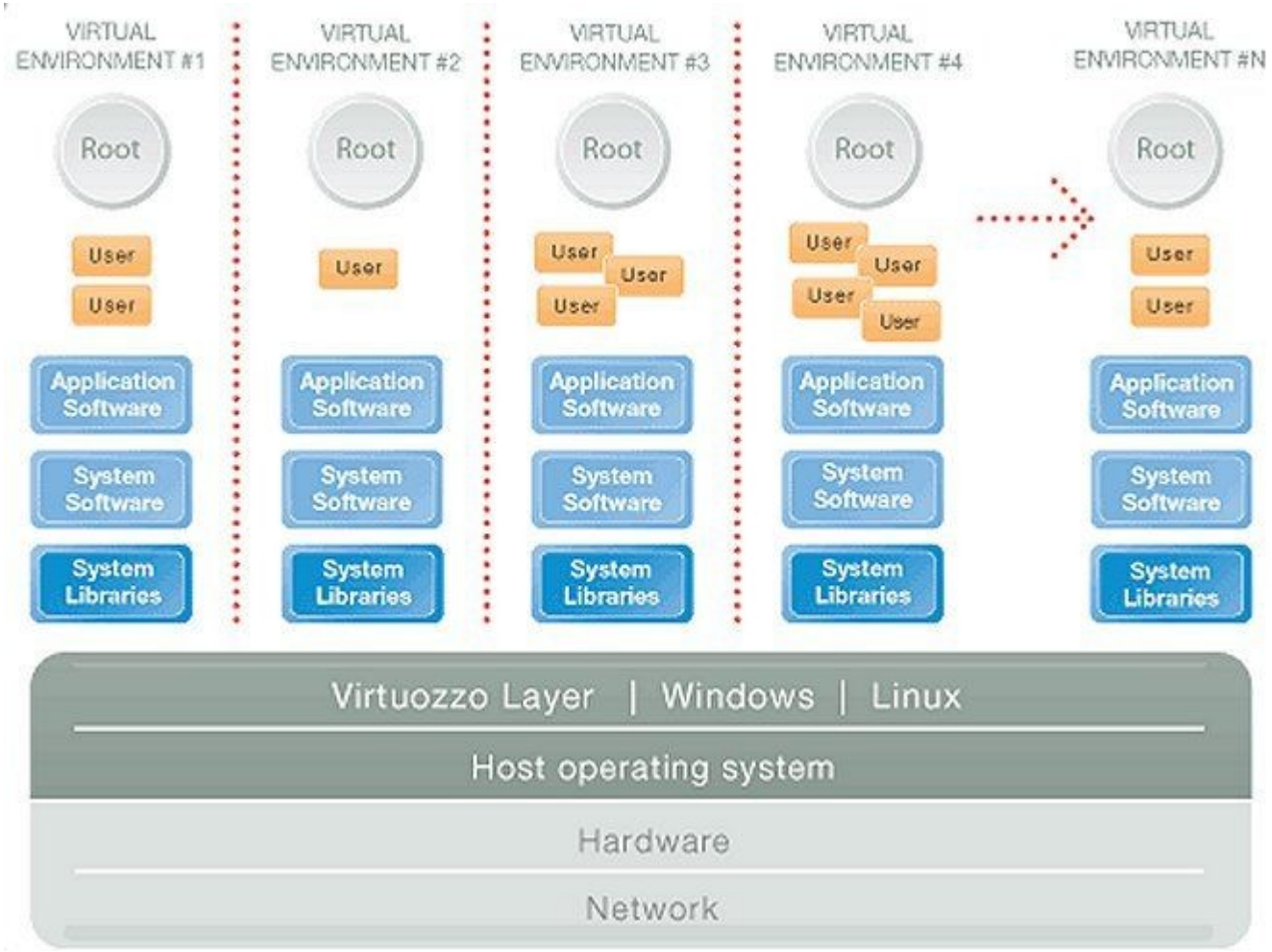
Виртуальное окружение, virtual enviroment



OpenVZ, LXC, FreeBSD jail, Solaris Containers

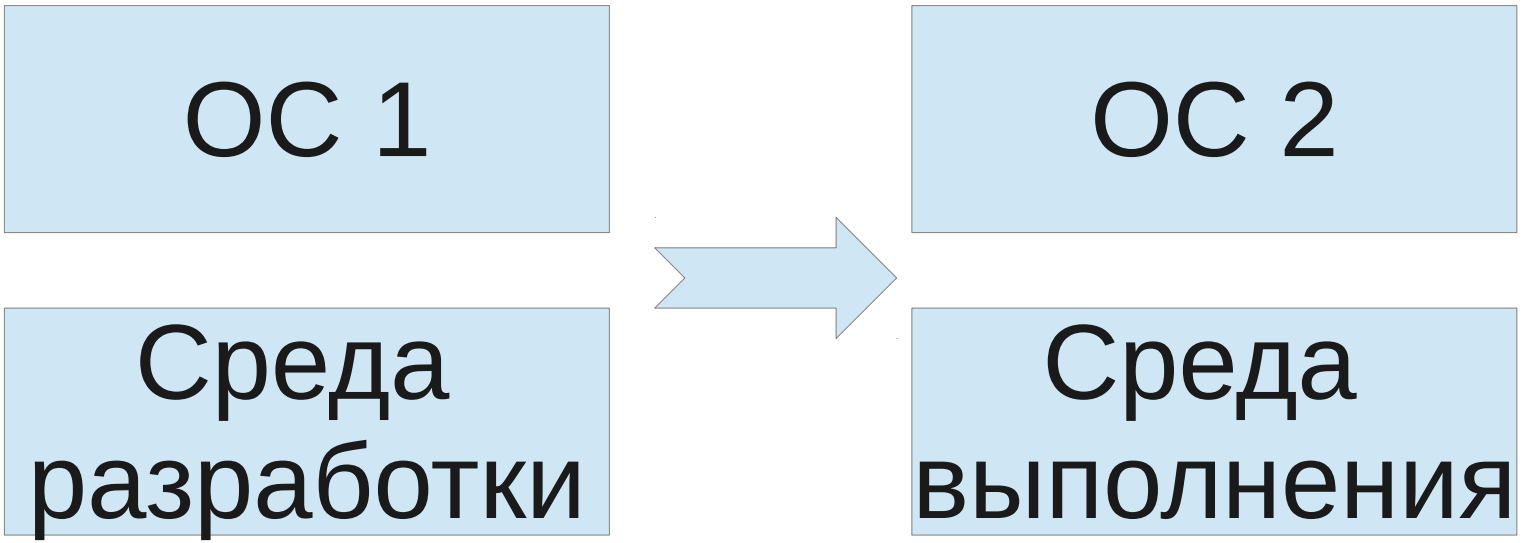
Контейнеры

Виртуальное окружение, virtual enviroment



OpenVZ, LXC, FreeBSD jail, Solaris Containers

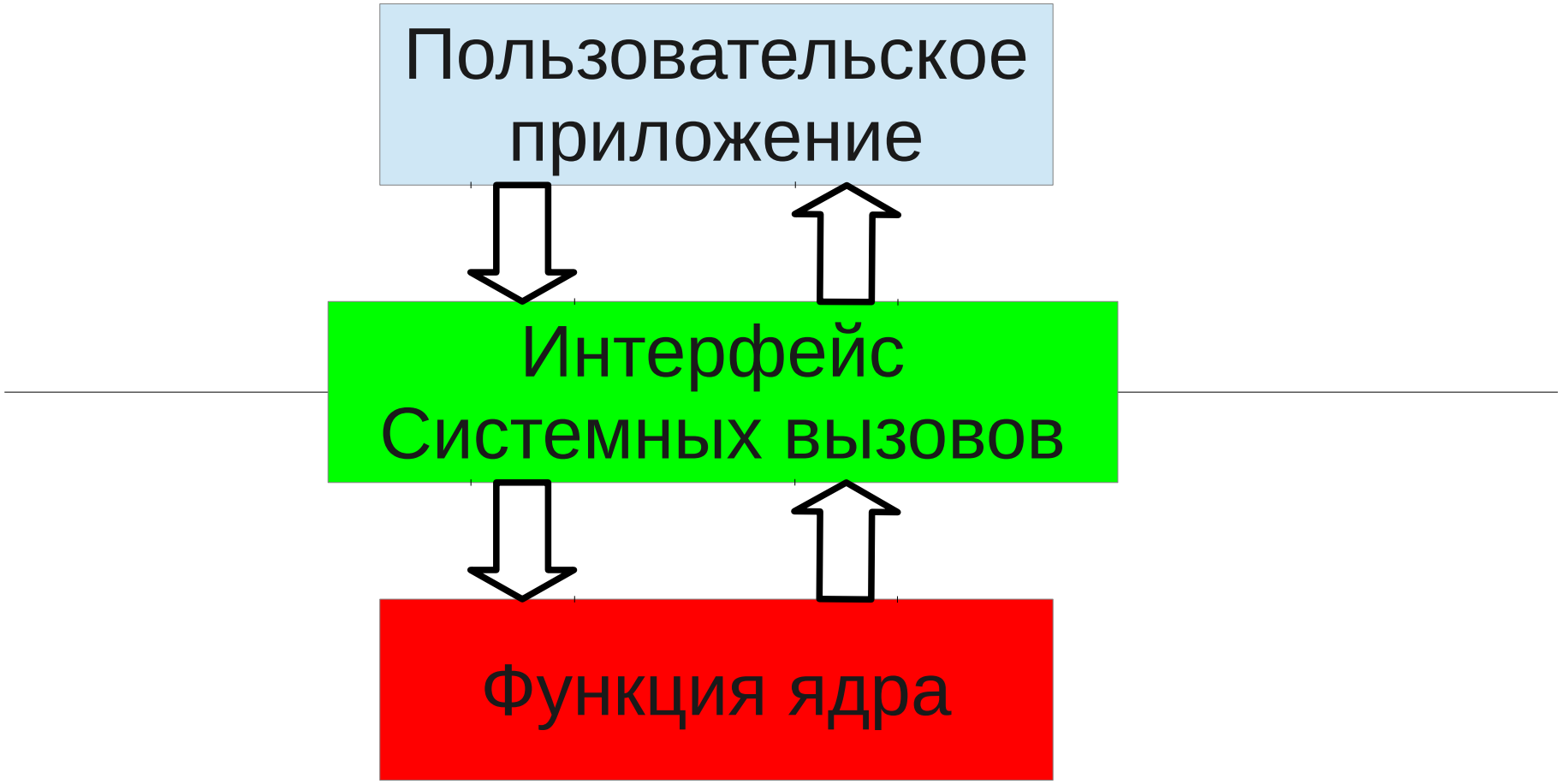
Средства кросс-разработки



Android SDK, iOS SDK

Структура ОС

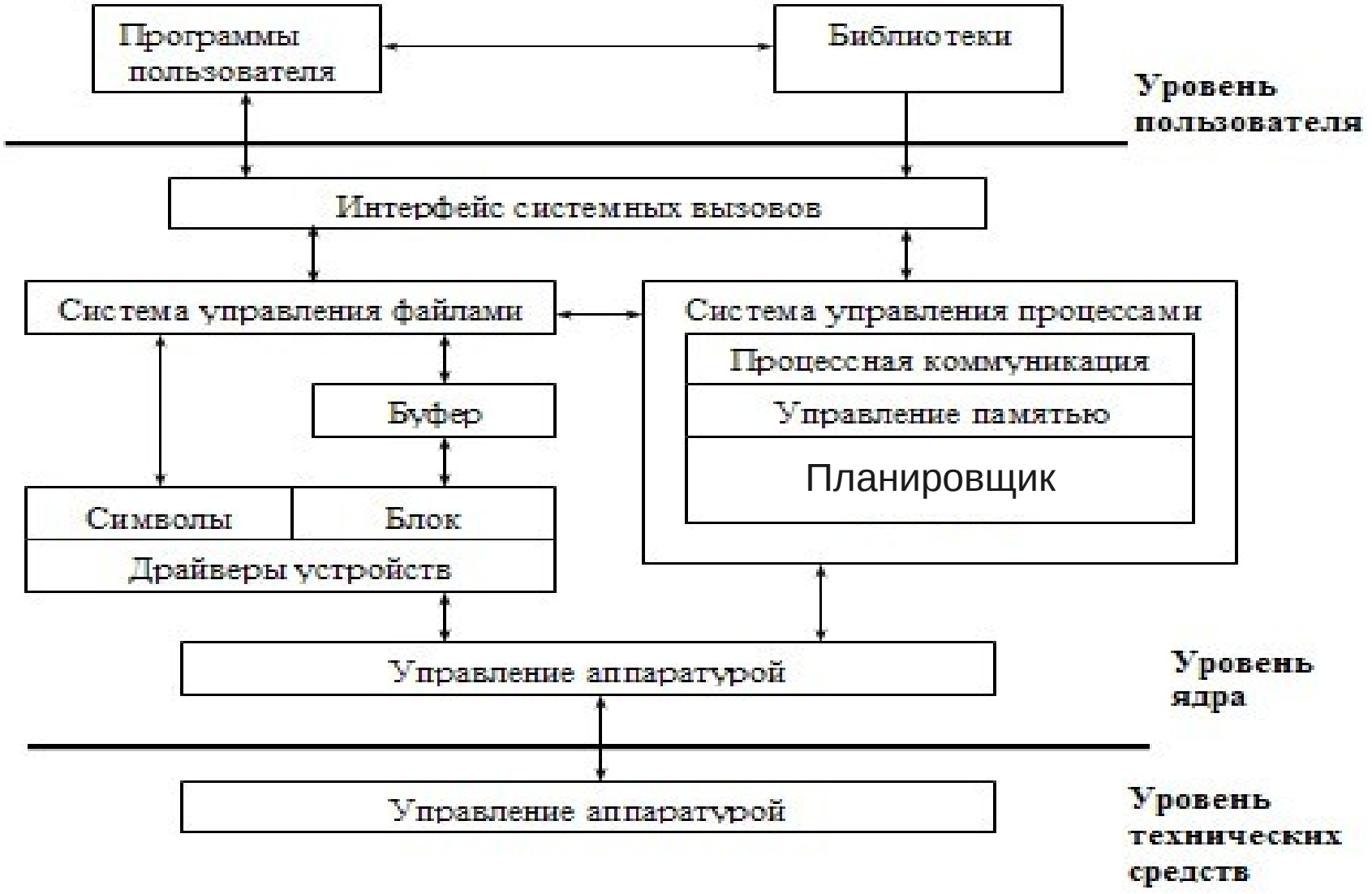
Режимы задачи и ядра ОС



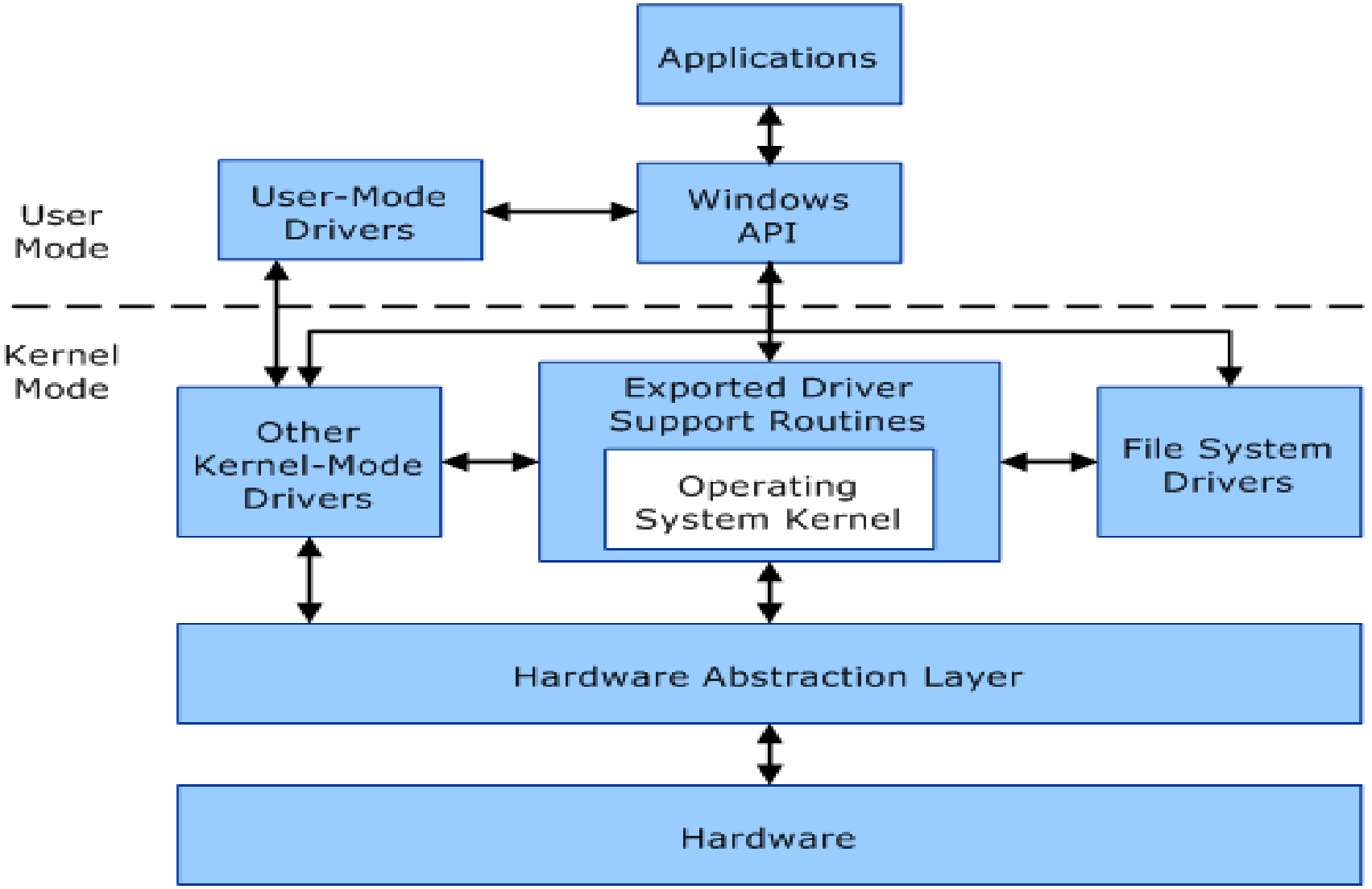
Возможности в режимах З и Я

Режим задачи (user mode)	Режим ядра (kernel mode)
Доступно адресное пространство только самого процесса	Доступно все адресное пространство
Перейти в режим Я (системный вызов)	Выполнить системный вызов
Нет возможности вносить управлять исполнением системного вызова	Результат системного вызова поместить в адресное пространство процесса
При прерывании неконтролируемый процессом переход в режим Я	После обработки прерывания возврат в режим З

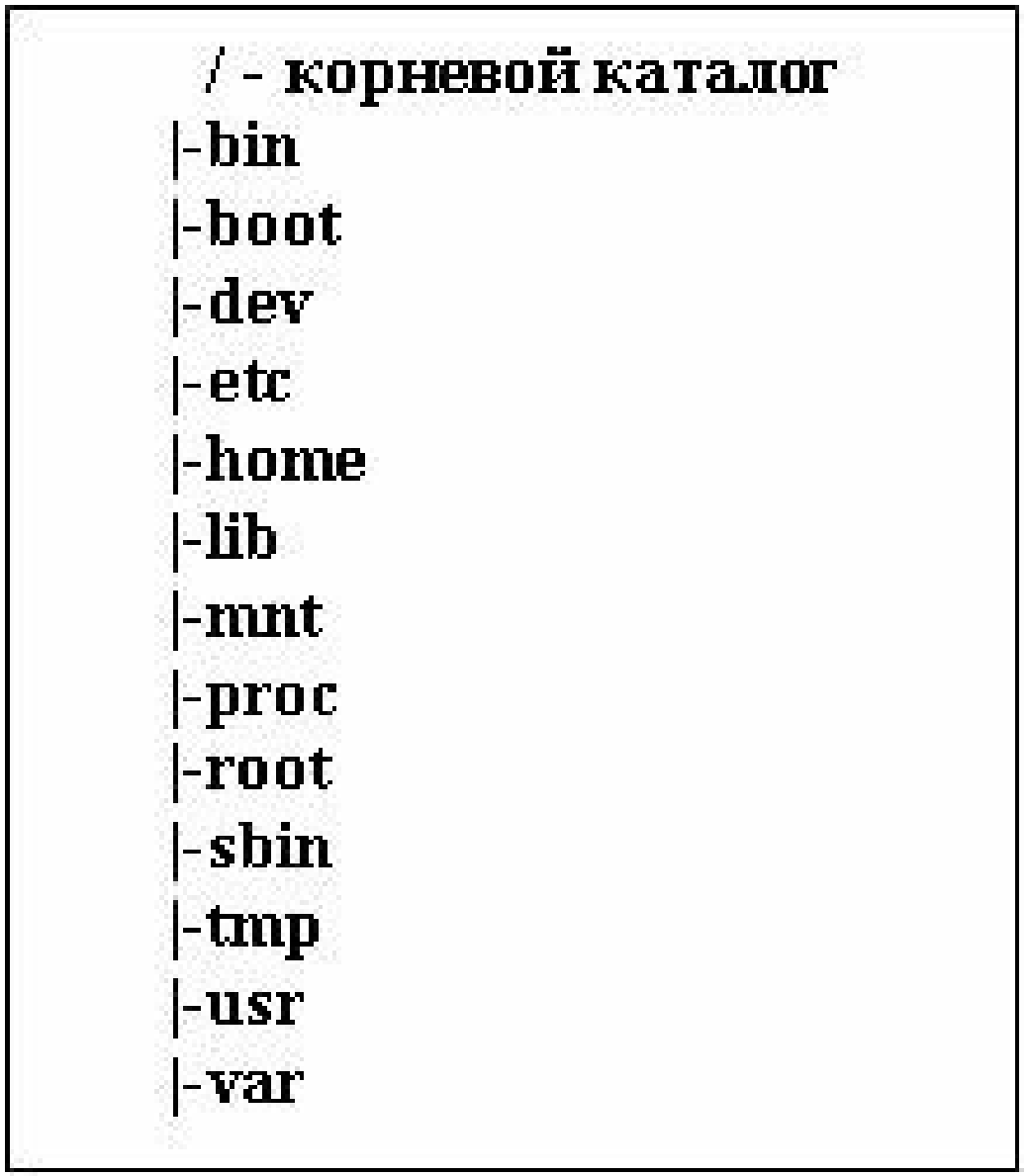
Блок-схема ядра



Блок-схема windows



Структура каталогов Unix

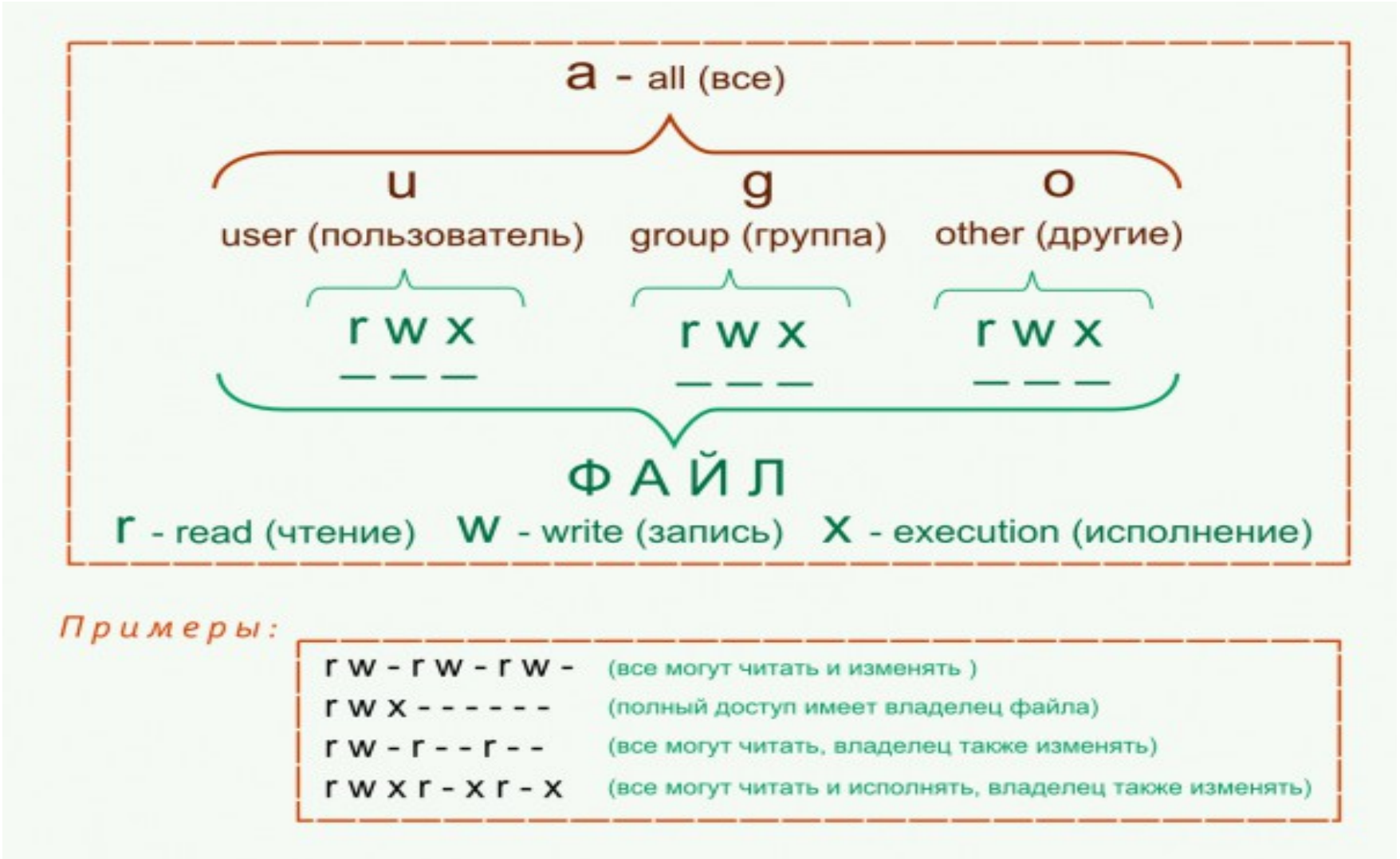


Имена каталогов Unix

.. — вышестоящий каталог
. — текущий каталог
/ — корневой каталог

```
$ pwd
/tmp/someplace/t1
$ cd ..
$ pwd
/tmp/someplace
$ cd .
$ pwd
/tmp/someplace
```

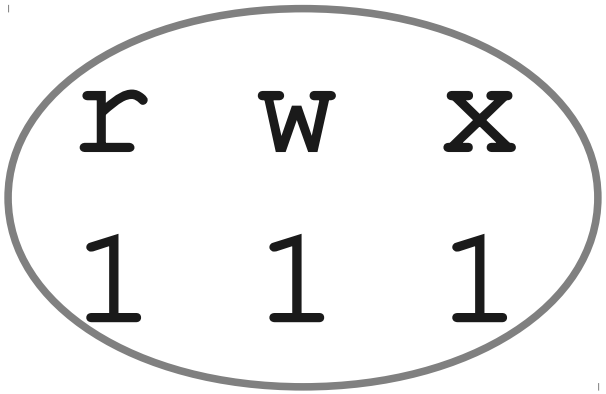
Права доступа файлов



Примеры:

r w - r w - r w -	(все могут читать и изменять)
r w x - - - - - -	(полный доступ имеет владелец файла)
r w - r - - r - -	(все могут читать, владелец также изменять)
r w x r - x r - x	(все могут читать и исполнять, владелец также изменять)

Кодирование прав доступа



r	w	x	r	–	x	–	–	x
1	1	1	1	0	1	1	0	0
	7			5			1	

Примеры установки прав доступа

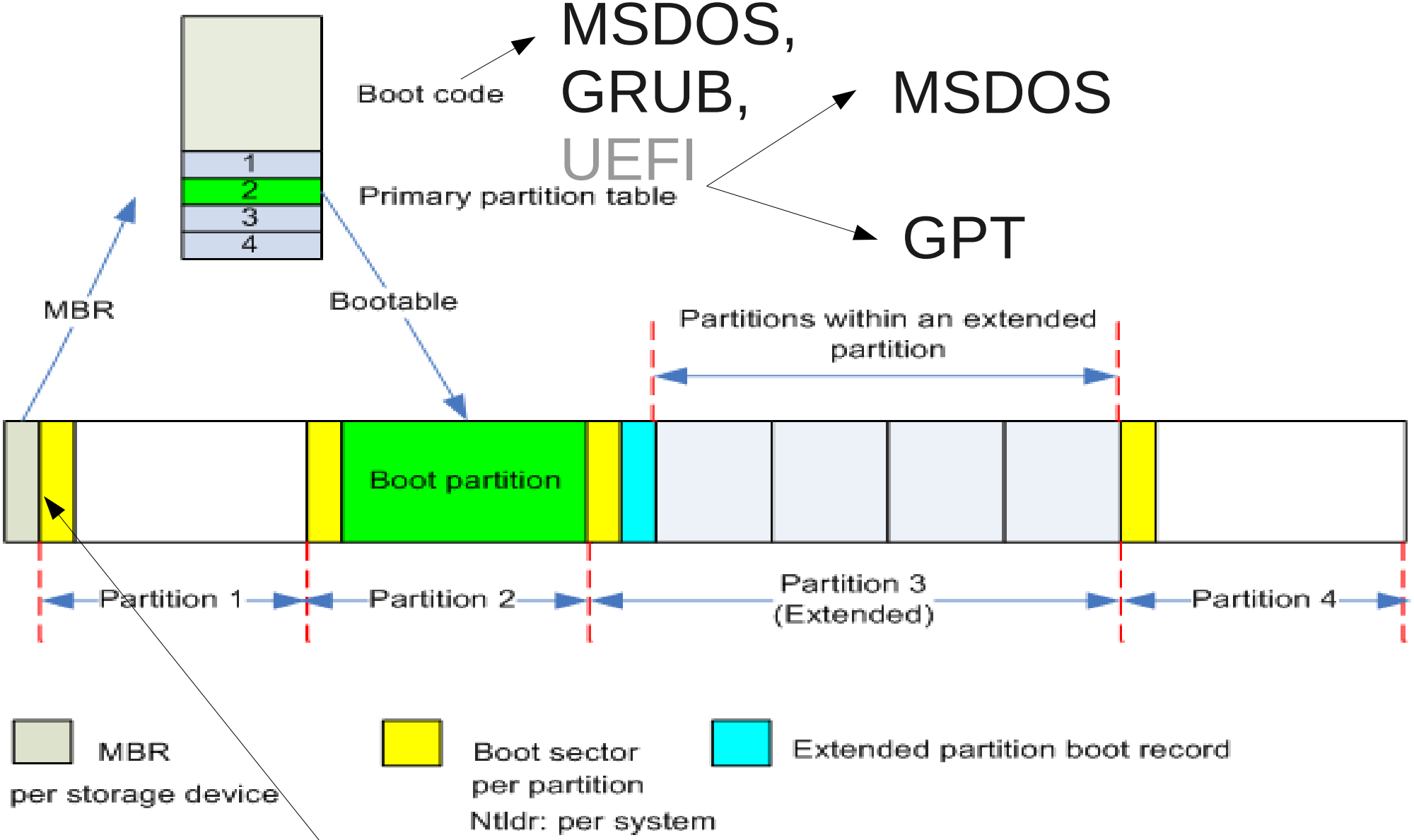
```
$ ls -l f1.txt
-rw-r--r-- 1 kdb kdb 10 Янв 29 22:12 f1.txt
$ chmod 755 f1.txt
$ ls -l f1.txt
-rwxr-xr-x 1 kdb kdb 10 Янв 29 22:12 f1.txt
$ chmod g+w f1.txt
$ ls -l f1.txt
-rwxrwxr-x 1 kdb kdb 10 Янв 29 22:12 f1.txt
```

«Дисковая» память

MSDOS,
GRUB,
UEFI

MSDOS

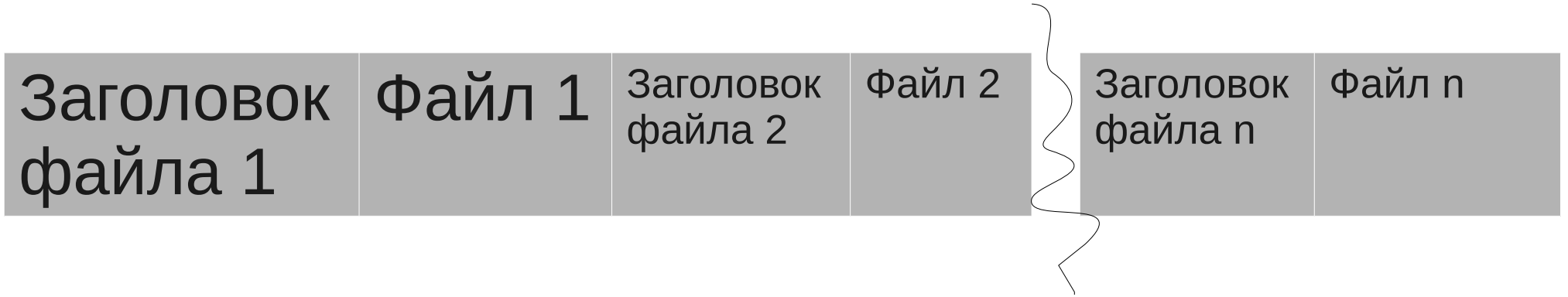
GPT



Для MSDOS начало в 62*512

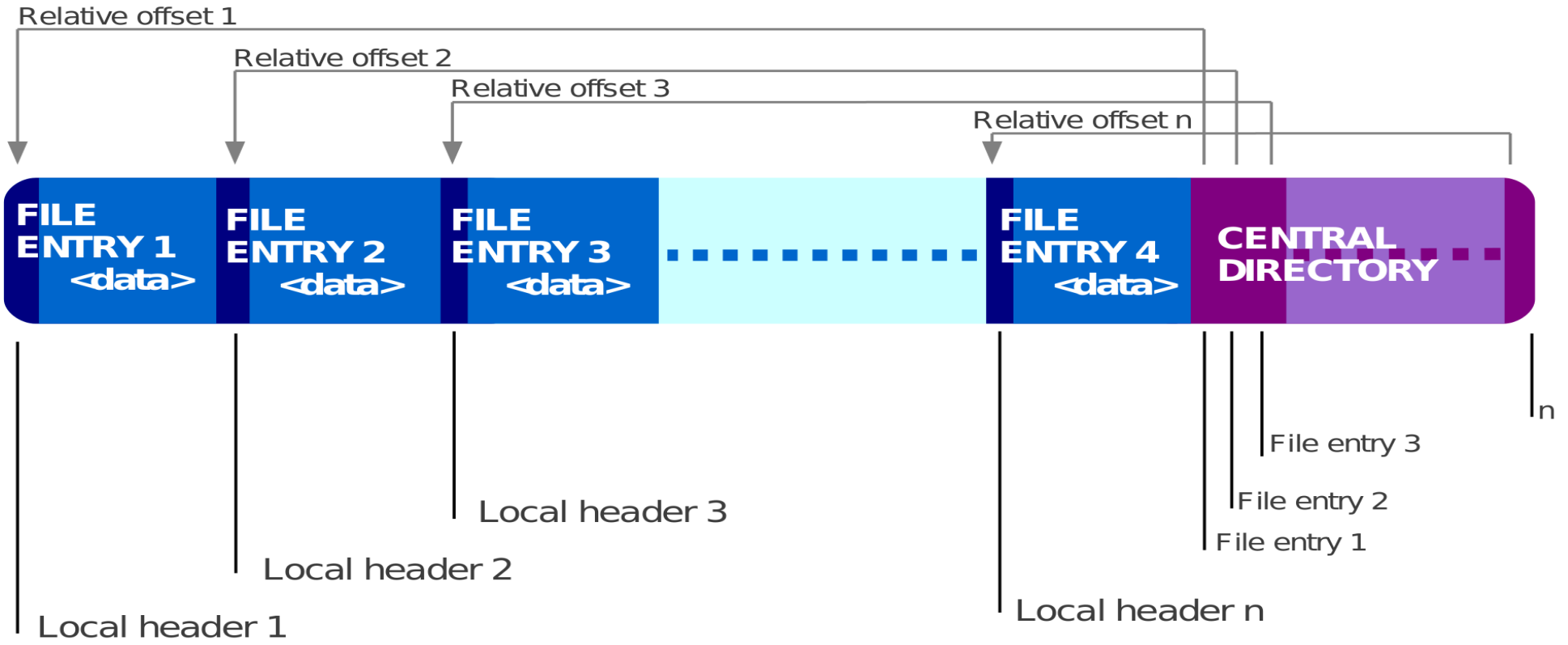
ФС TAR

Type ARchive



- низкая скорость поиска
- низкая скорость произвольно доступа
- отсутствие возможности фрагментации
+ сбоеустойчивость
+ скорость полного последовательно чтения

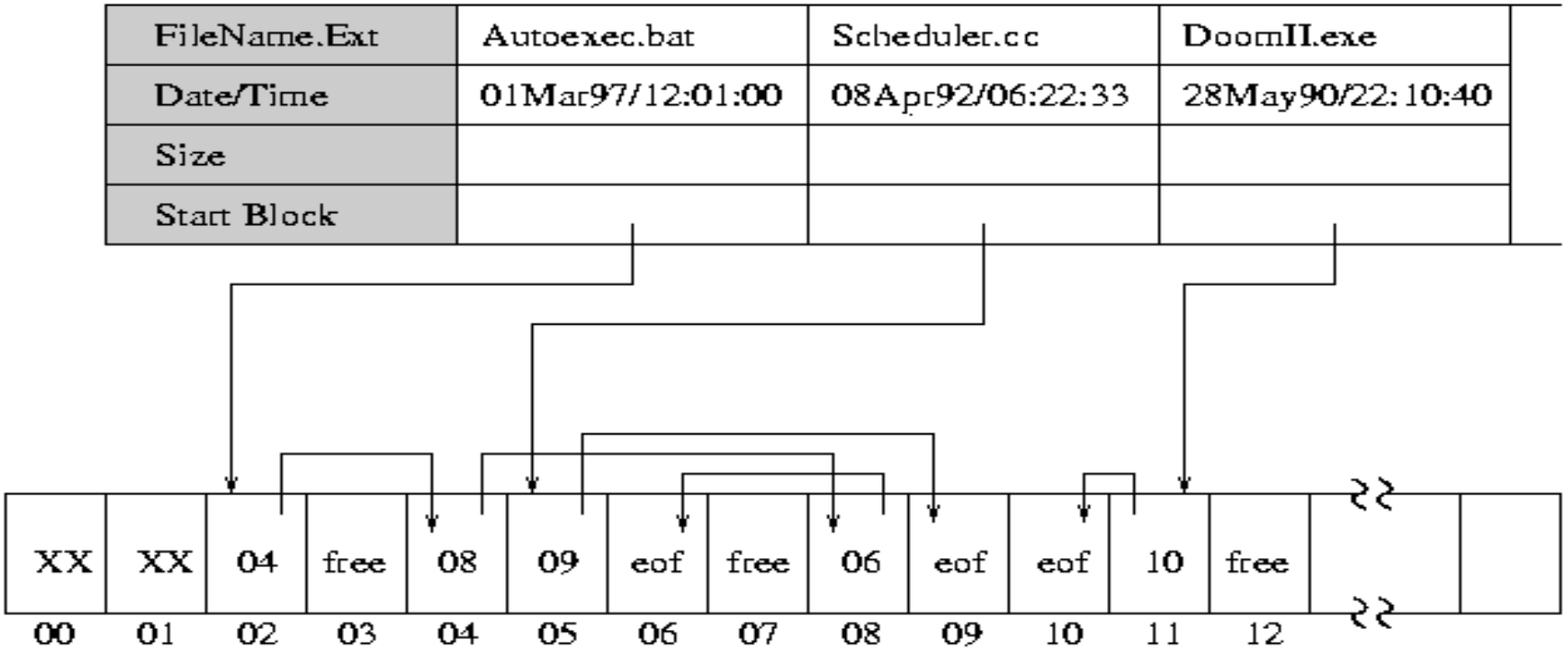
ФС RT11, ZIP



- низкая скорость произвольно доступа
- отсутствие возможности фрагментации
+ скорость полного последовательно чтения

FAT (12,16,32)

MS/DOS Directory Entires



- низкая производительность
+ возможна фрагментации
+ простота — сбоеустойчивость

s5fs

UNIX SYSTEM V

Суперблок	Таблица инодов	Блоки данных

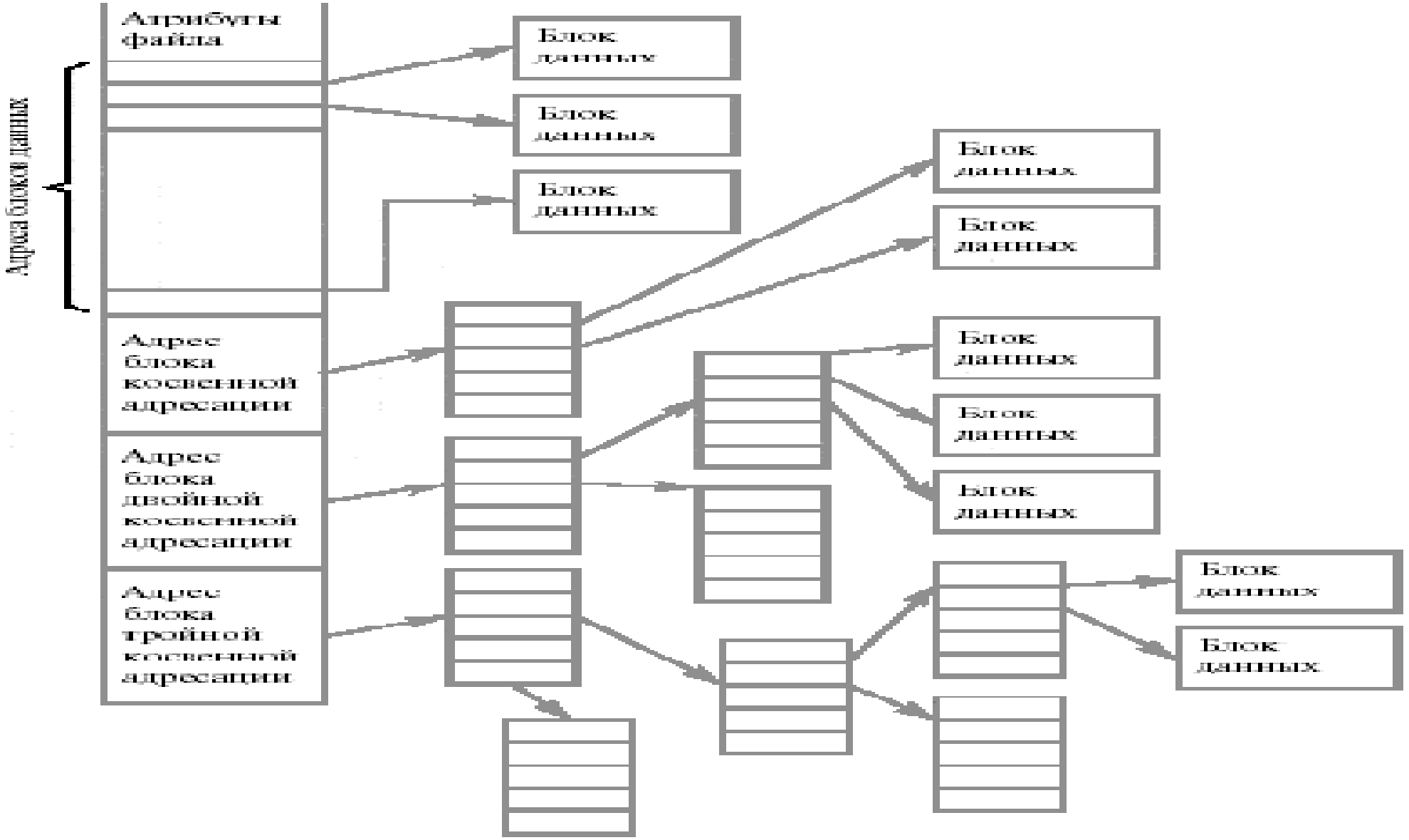
В суперблоке содержится информация о файловой системе в целом:

- тип,
- размер в блоках,
- размер блока (512, 1024, 2048),
- число свободных блоков,
- число свободных инодов,
- список свободных блоков,
- список свободных инодов.

Структура инода

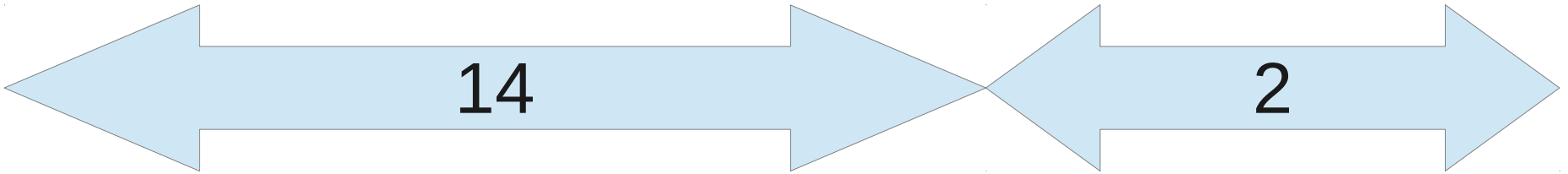
- тип файла,
- время доступа, создания и последней модификации файла,
- моды доступа (чтение, запись, исполнение),
- идентификаторы пользователя и группы(UID, GID),
- число ссылок,
- размер файла (или номера устройств),
- прямые ссылки на блоки данных -- 10 штук,
- косвенная ссылка на блоки данных,
- ссылка двойной косвенности на блоки данных,
- ссылка тройной косвенности на блоки данных.

Ссылки инода



Структура каталога s5fs

ИМЯ ФАЙЛА	НОМЕР ИНОДА



F1.txt	6
F2.txt	9
somedir	8
F11.txt	16
F1	6

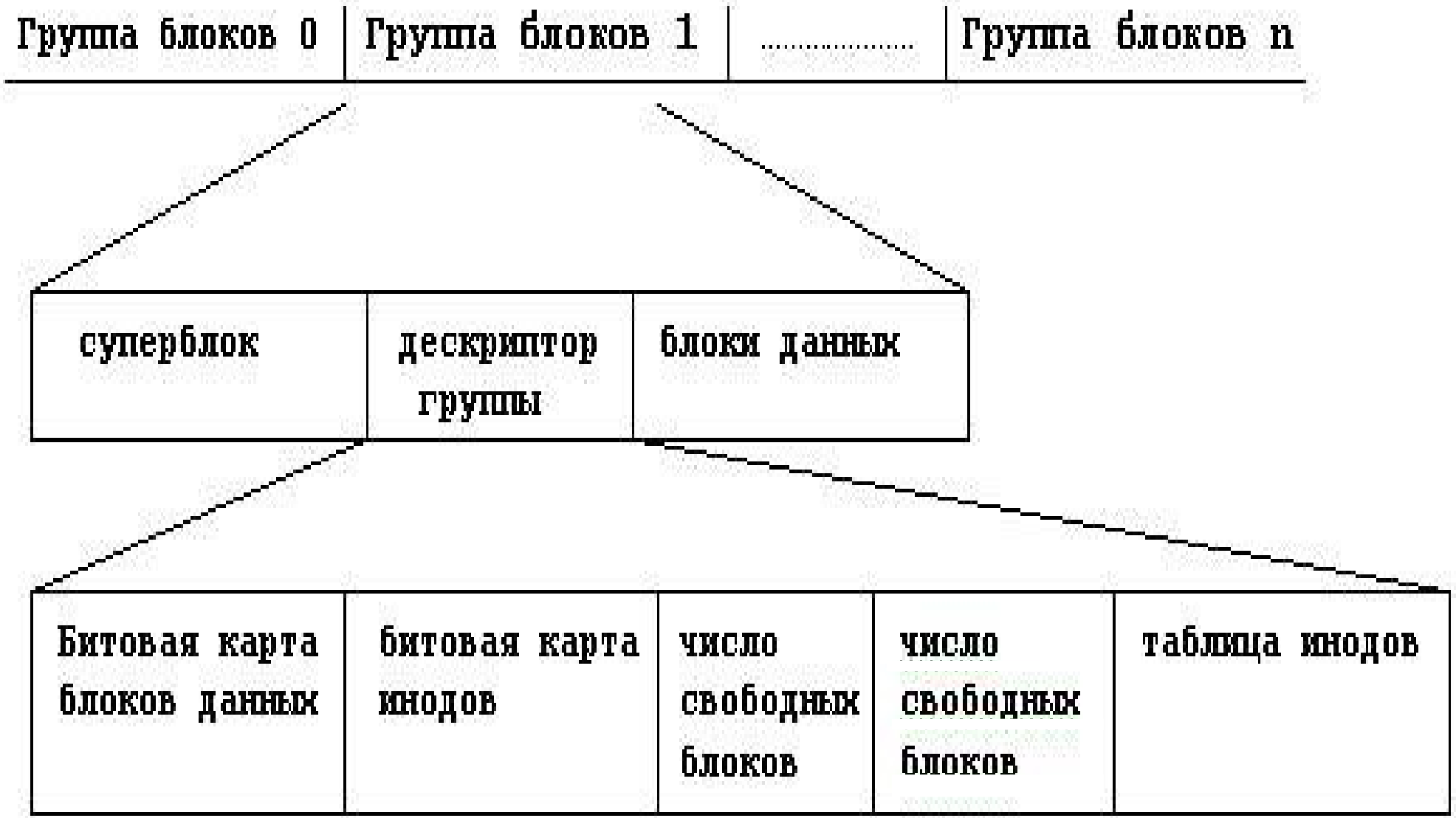
Почему в Си нет функции «удалить файл»?

Достоинства/недостатки s5fs

СУПЕРБЛОК	ИНОДЫ	БЛОКИ ДАННЫХ
тип, размер в блоках, размер блока, число свободных блоков, число свободных инодов, список свободных блоков, список свободных инодов	тип файла, число ссылок, время, моды доступа, UID, GID, размер файла, ссылки на блоки данных	

+ хорошая производительность (особенно для небольших файлов)
+ возможность разграничения доступа к файлам
- низкая сбоеустойчивость
- списки свободных блоков/инодов
- короткие имена файлов

Ext2(3,4)


Группа блоков 0
Группа блоков 1
.....
Группа блоков n
суперблок
дескриптор группы
блоки данных
Битовая карта блоков данных
битовая карта инодов
число свободных блоков
число свободных блоков
таблица инодов

NTFS

Файловую систему NTFS принято описывать как сложную реляционную базу данных, обескураживающую грандиозностью своего архитектурного замысла не одно поколение начинающих исследователей. NTFS похожа на огромный, окутанный мраком лабиринт, в котором очень легко заблудиться.

В NTFS все служебные структуры представлены файлами.

inode	Имя файла	Описание
0	$MFT	главная файловая таблица (Master File Table, MFT)
1	$MFTMirr	резервная копия первых четырех элементов 4 MFT
2	$LogFile	журнал транзакций (transactional logging file)
3	$Volume	серийный номер, время создания, dirty flag (флаг не сброшенного кэша) тома
4	$AttrDef	определение атрибутов
5	. (точка)	корневой каталог (root directory) тома
6	$Bitmap	карта свободного/занятого пространства
7	$Boot	загрузочная записи (boot record) тома
8	$BadClus	список плохих кластеров (bad clusters) тома

MFT-зона содержит служебные файлы, включая сам MFT.
Размер по умолчанию — 12,5%

MFT состоит из файловых записей — аналог инодов.

Размер одной записи — 1к.
Возможно дополнительная файловая запись.

FILE Record
Header ; заголовок
Attribute 1 ; атрибут 1
Attribute 2 ; атрибут 2
...
End Marker (FFFFFFFFh) ; маркер конца

Заголовок:
- Число ссылок
- Тип (файл,каталог, пусто)
- Размер записи

Атрибуты

Резидентные

Нерезидентные

Заголовок •Тип атрибута •Длина •Нерезидентный флаг •Длина имени атрибута •Имя атрибута	Тело или Список отрезков

Типы атрибутов

Условное обозначение	Описание
$STANDARD_INFORMATION	стандартная информация о файле (время, права доступа)
$ATTRIBUTE_LIST	список атрибутов
$FILE_NAME	полное имя файла
$VOLUME_VERSION	версия тома
$OBJECT_ID	уникальный GUID и прочие ID
$SECURITY_DESCRIPTOR	дескриптор безопасности и списки прав доступа (ACL)
$VOLUME_NAME	имя тома
$VOLUME_INFORMATION	информация о томе
$DATA	основные данные файла
$INDEX_ROOT	корень индексов
$INDEX_ALLOCATION	ветви (sub-nodes) индекса
$BITMAP	карта свободного пространства
$SYMBOLIC_LINK	символическая связь

Списки отрезков

Тела нерезидентных атрибутов хранятся на диске в одной или нескольких кластерных цепочках, называемых отрезками. Отрезком называется последовательность смежных кластеров, характеризующаяся номером начального кластера и длиной.

Смещение в нибблах (полубайтах)	**Размер в нибблах**	**Описание**
0	1	размер поля длины (L)
1	1	размер поля начального кластера (S)
2	2*L	количество кластеров в отрезке
2+2*L	2*S	номер начального кластера отрезка
конец	1	0x00

Примеры отрезков

21 18 34 56 00

1 — размер поля длины
2 — размер поля начального кластера
18 — число кластеров в отрезке
5634 — номер начального кластера

31 38 73 25 34 32 14 01 E5 11 02 31 42 AA 00 03 00

1. 342573 + 38
2. 0211E5 + 114
3. 0300AA + 42

Журналируемые ФС

Операции с файловой системой не атомарны.

Пример: удаление файла

Уменьшение числа ссылок на инод

Ссылки на блоки файла - в список свободных

Номер инода - в список свободных

Возможны сбои:
- Снятие питания
- Отключение накопителя
- Повреждение накопителя

Следствие сбоя - нарушение целостности ФС

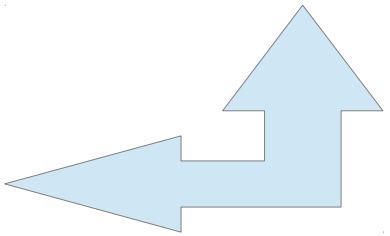
Журналируемые ФС

Восстановление целостности

Проверка ФС после сбоя

Откат состояния до операции, во время которой произошел сбой

Требуется хранить информацию о ходе выполнении операции — журнал **транзакций**



Журналируемые ФС

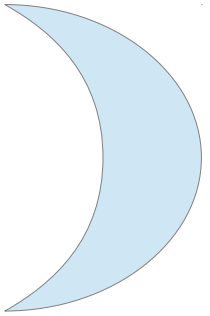
- Ext3, ext4
- Ntfs
- Btrfs (Oracle)
- ZFS (Sun, Oracle)

Полезные свойства ФС

- Copy-on-write
- Snapshot
- Encryption

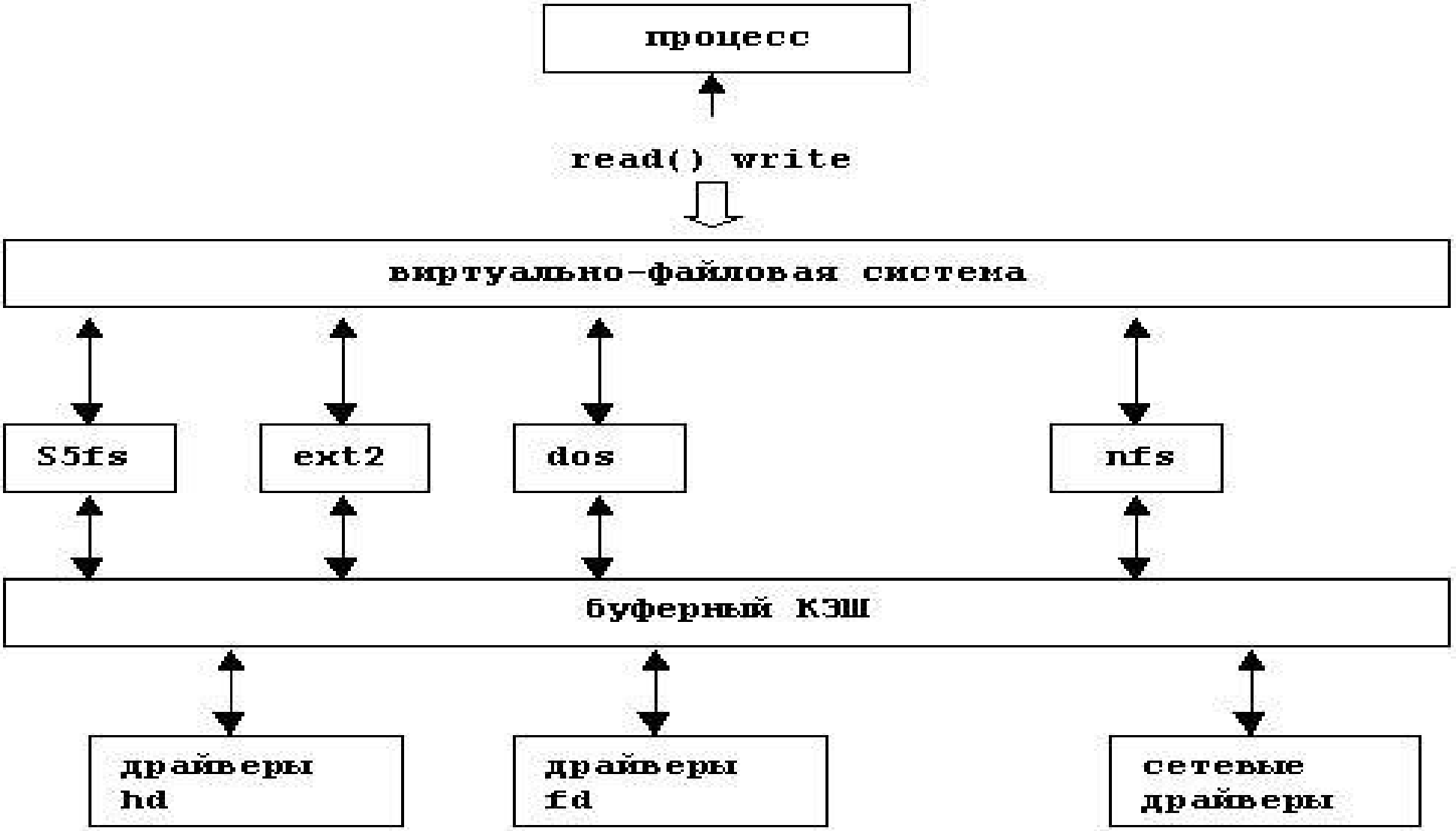
Не совсем файловые системы

- Shared disk file systems
- Distributed file systems
- Distributed fault-tolerant file systems
- Distributed parallel file systems
- Distributed parallel fault-tolerant file systems
- RAID
- LVM

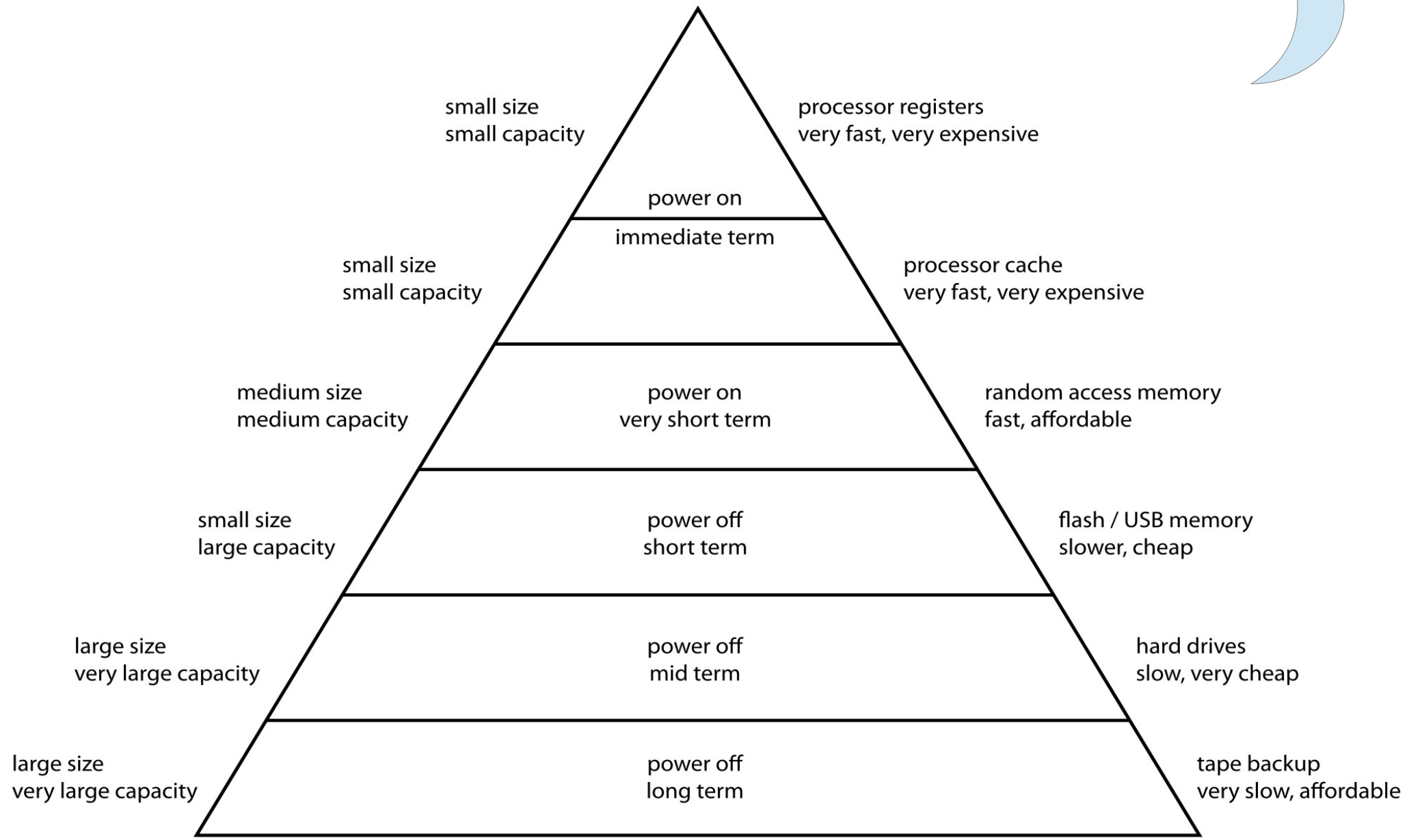


VFS

Unix. Java, Windows — на уровне API

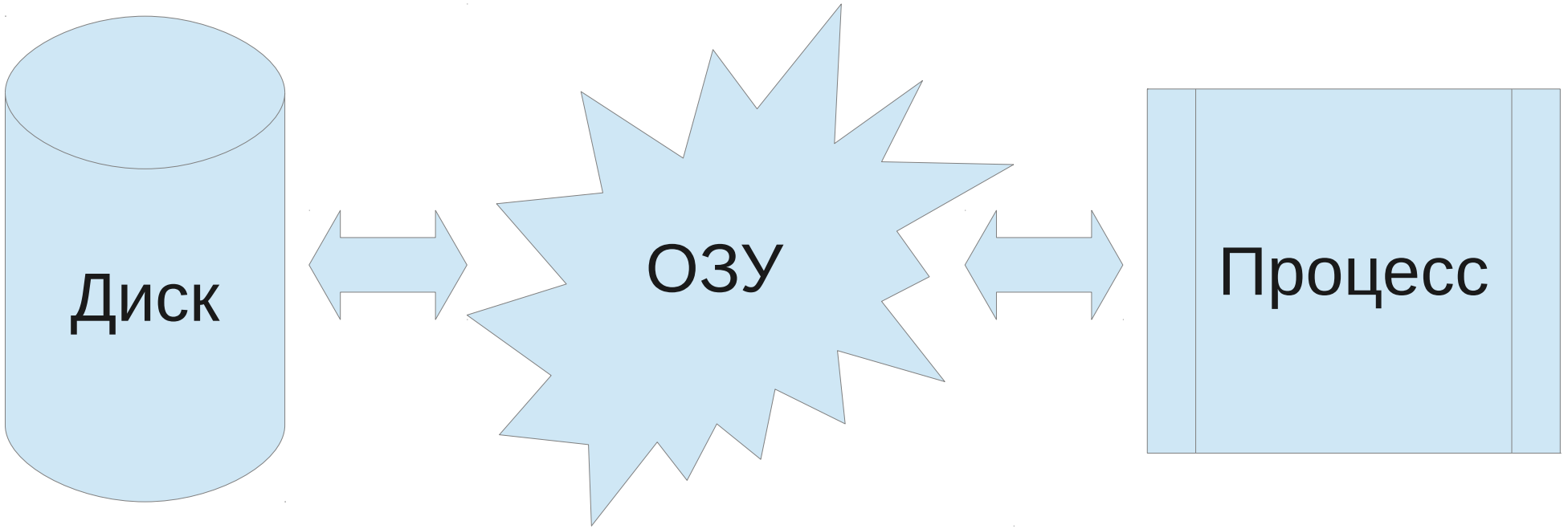


Иерархия памяти

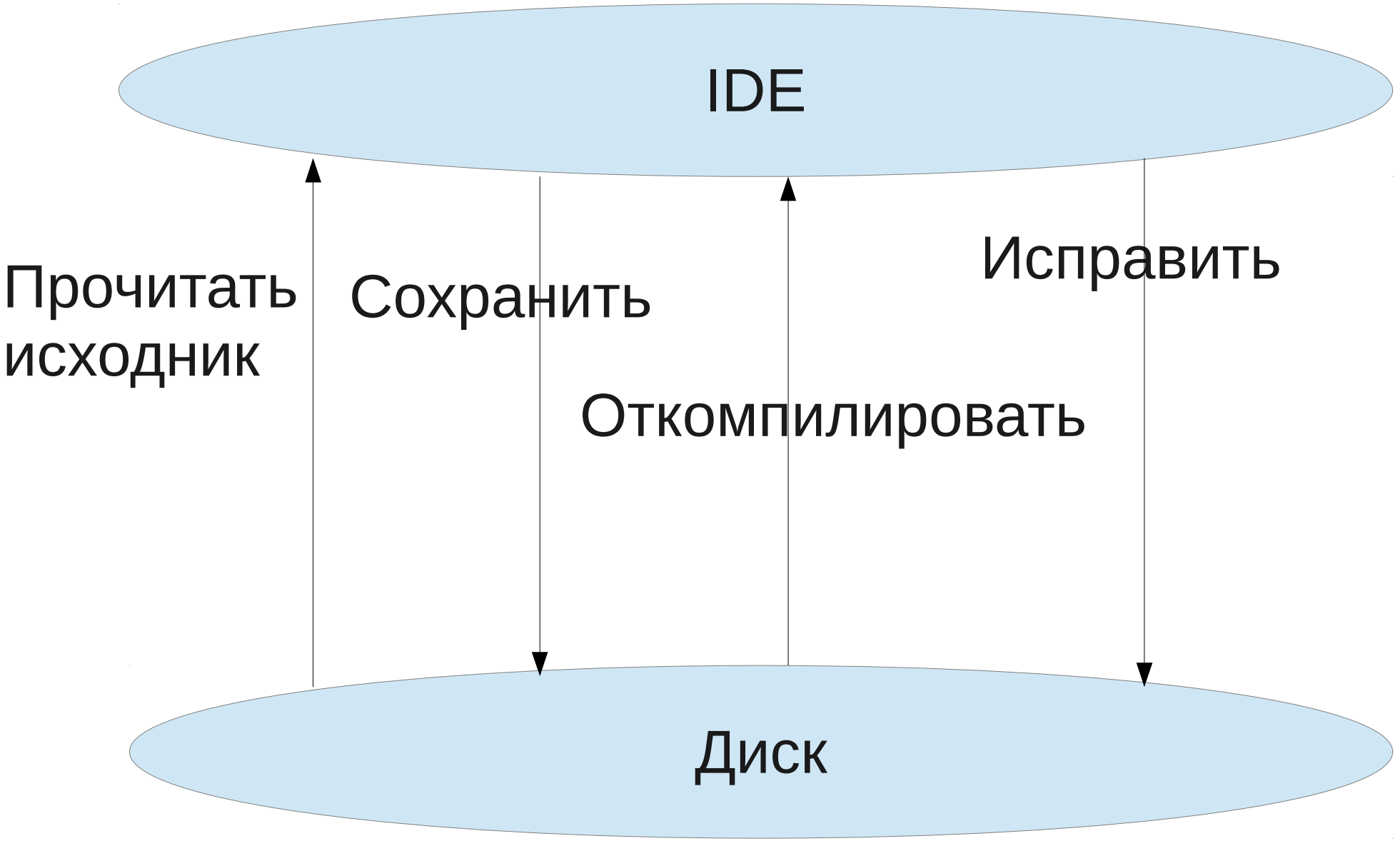


Буферный кэш

Часть оперативной память (адресное пространство ядра), предназначенная для снижения взаимодействия процессов с диском.

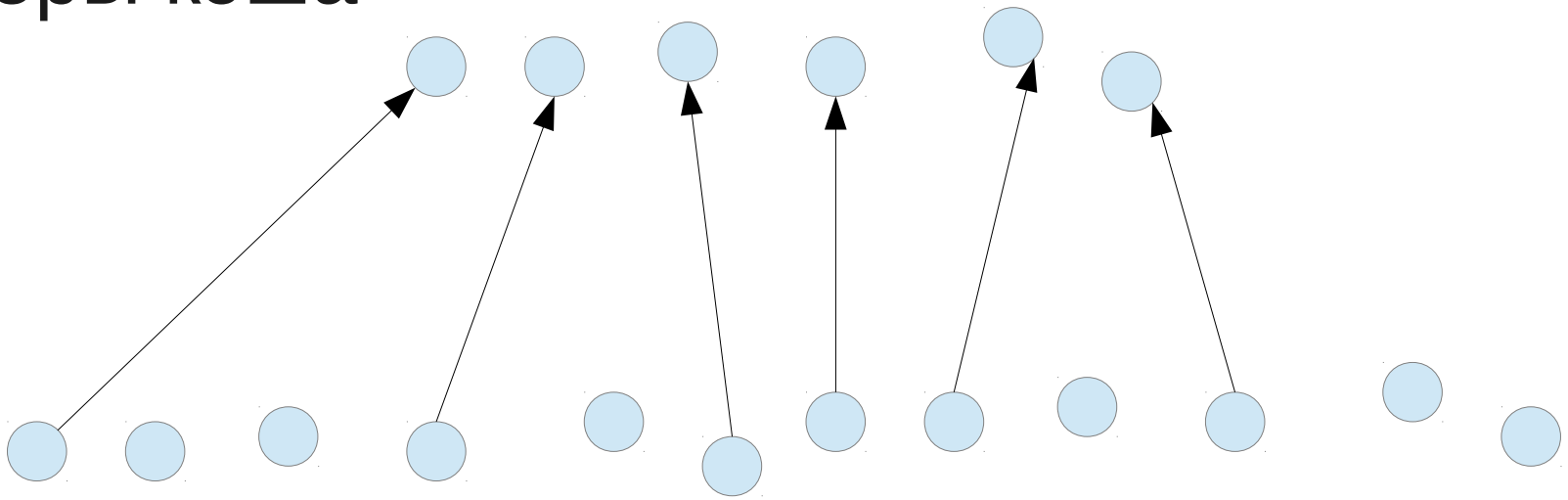


Пример взаимодействия с диском



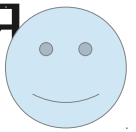
Буферы ⟷ блоки

Буферы кэша

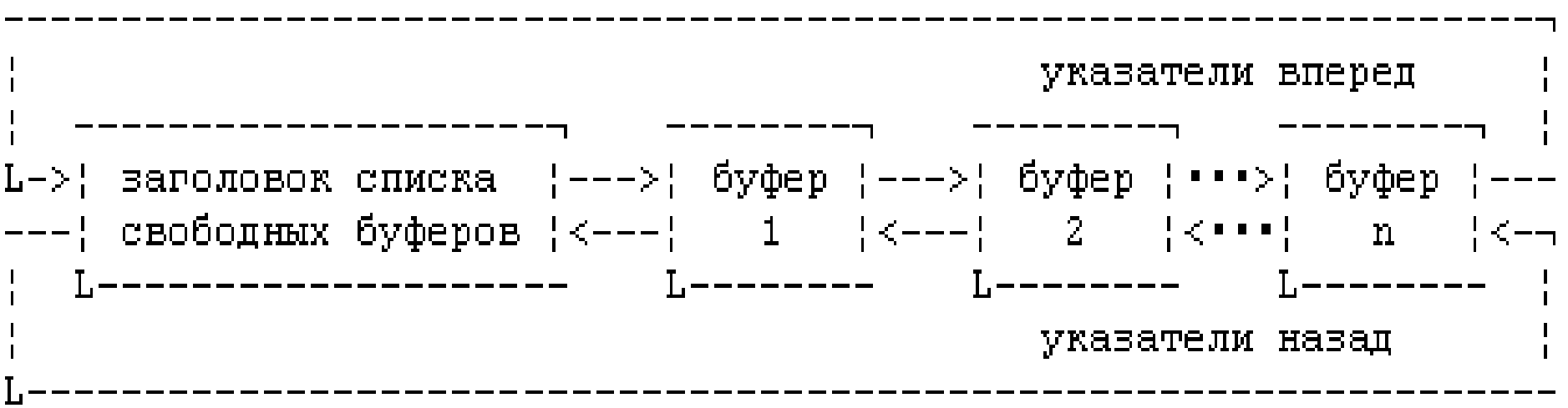


Блоки диска

Свойства соответствия



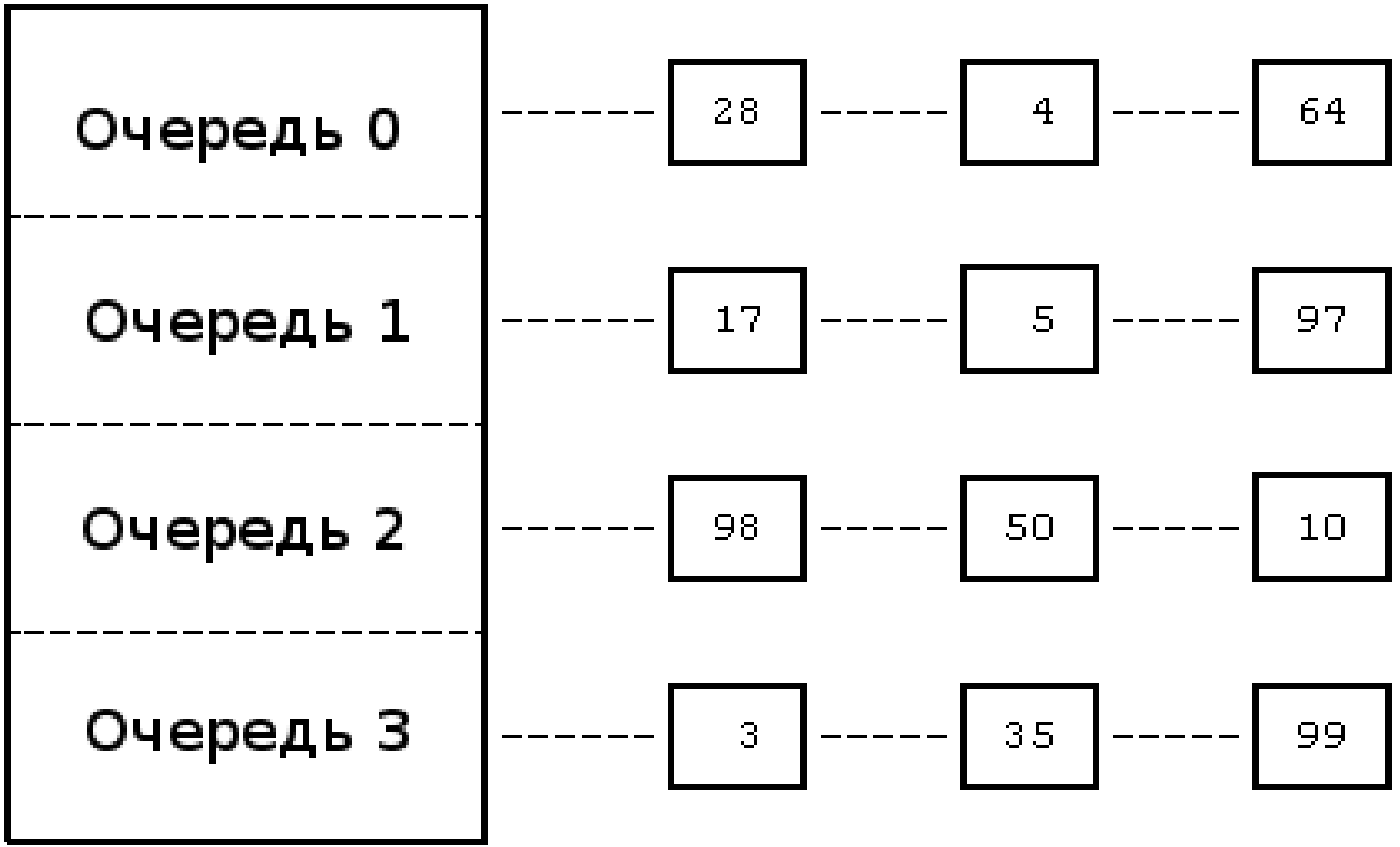
Список свободных буферов



- буфер под новый блок берется из начала
- буферы с поюзанными блоками помещаются в конец

Что это дает?
Зачем двунаправленное кольцо?

Хэш - очереди



Хэш-функция?

5 механизмов getblck

1. Ядро **обнаруживает** блок в хеш-очереди, соответствующий ему буфер свободен.
2. Ядро **не может** обнаружить блок в хеш-очереди, поэтому оно выделяет буфер из списка свободных буферов.
3. Ядро **не может** обнаружить блок в хеш-очереди, в списке свободных только буферы в состоянии "**занят на время записи**".
4. Ядро **не может** обнаружить блок в хеш-очереди, а **список свободных пуст**.
5. Ядро **обнаруживает** блок в хеш-очереди, но его буфер в настоящий момент **занят**.

Ядро обнаруживает блок в хеш-очереди, соответствующий ему буфер свободен

Найти

Выдать буфер

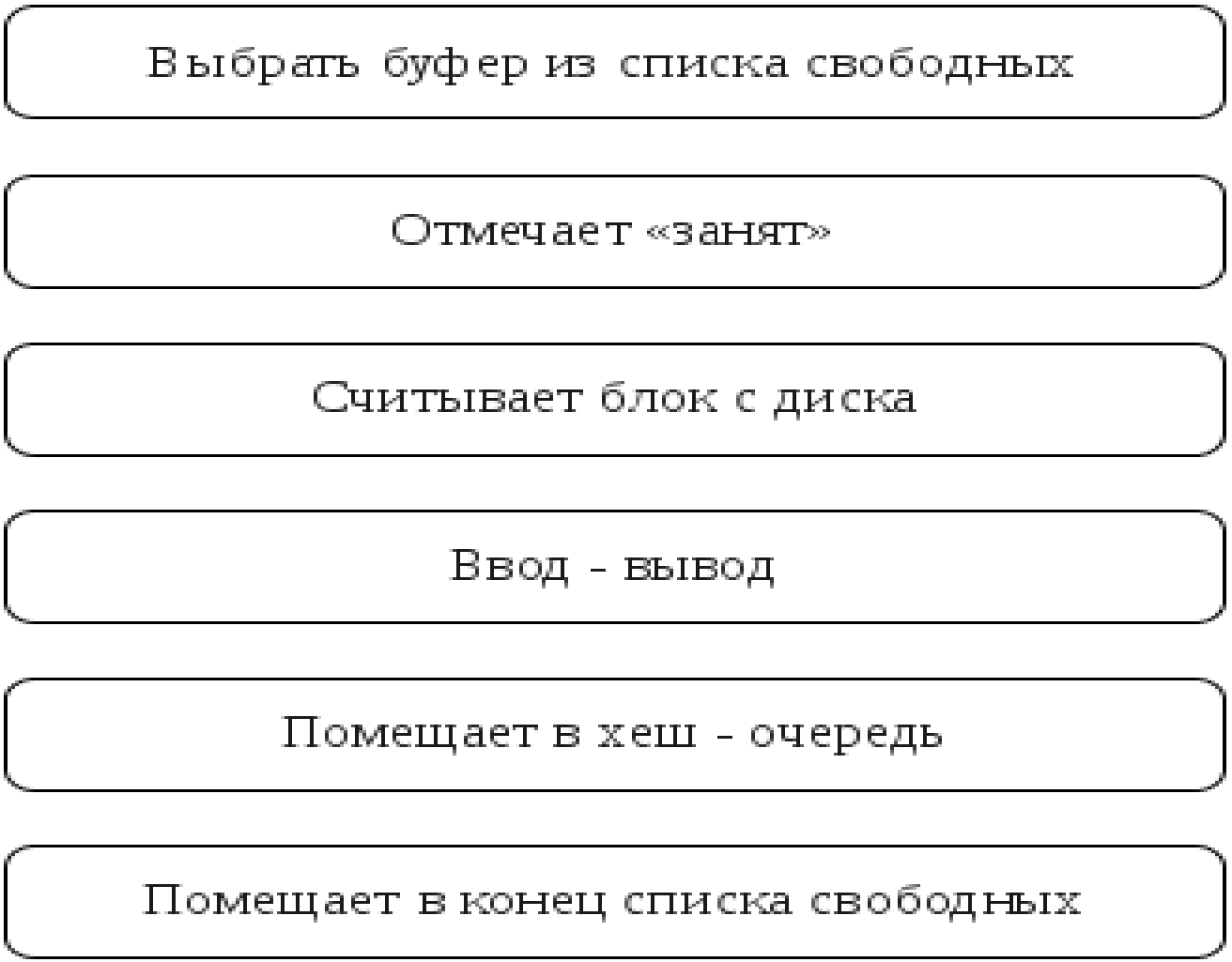
Пометить «занят»

Удалить из списка свободных

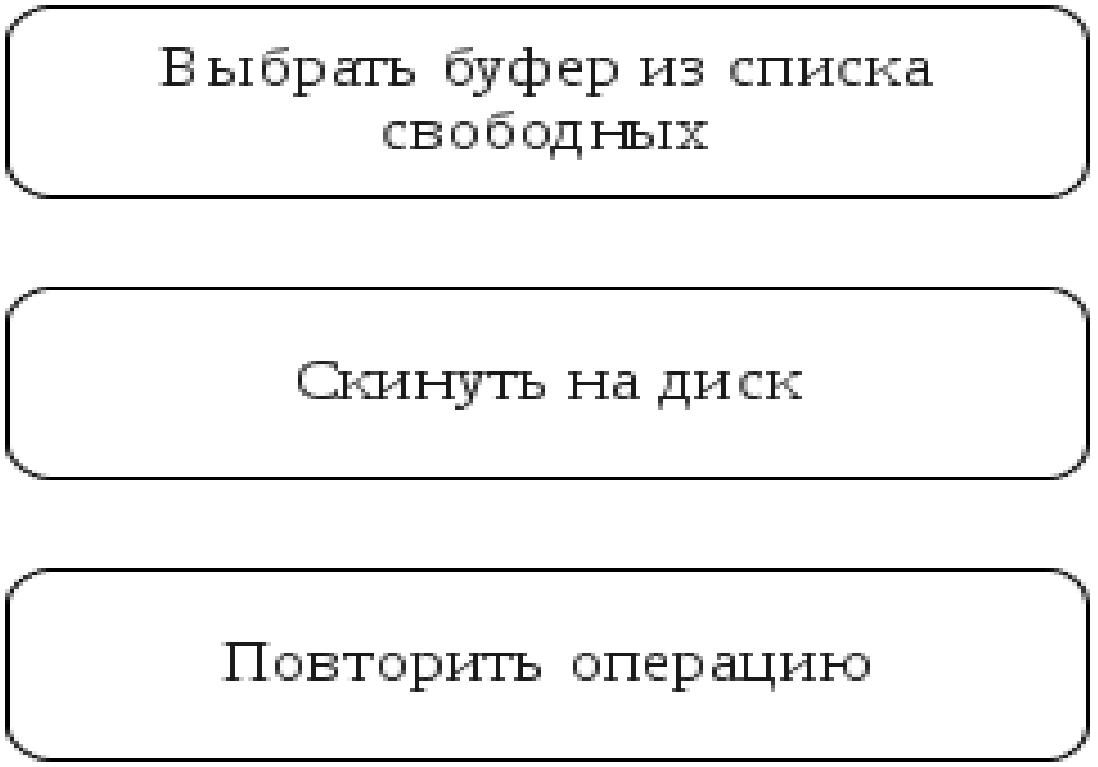
Ввод - вывод

Помещает в список свободных

Ядро не может обнаружить блок в хеш-очереди, поэтому оно выделяет буфер из списка свободных буферов.



Ядро не может обнаружить блок в хеш-очереди, в списке свободных только буферы в состоянии "занят на время записи"



Ядро не может обнаружить блок в хеш-очереди, а список свободных пуст

Ядро обнаруживает блок в хеш-очереди, но его буфер в настоящий момент занят

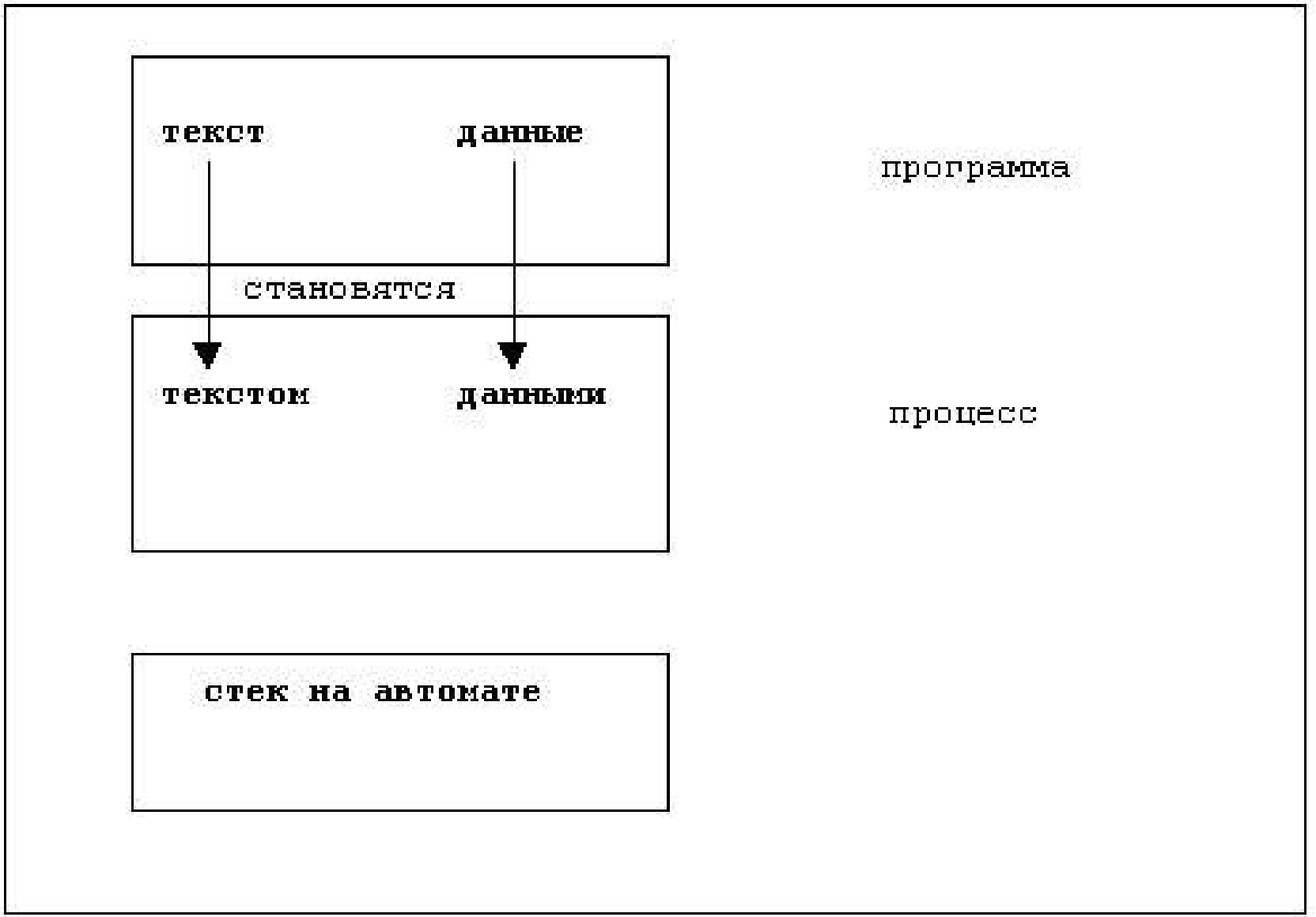
Особенности буферного кэша

+ увеличение производительности системы при многократном использовании одних и тех же файлов
+ унификация доступа к блочным устройствам
- пробой кэша 
- потеря данных при сбое

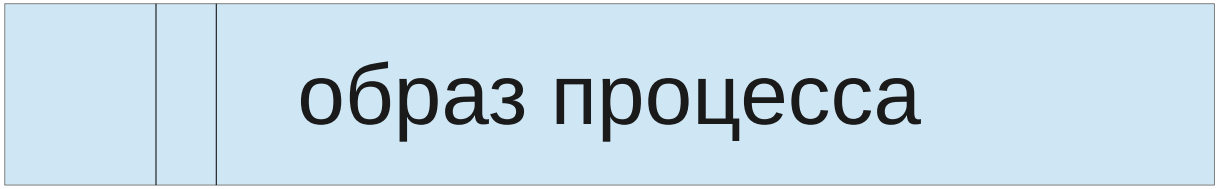
Понятие процесса

- Процесс
- Задача
- Легкий процесс (lightweight)
- Нить (thread)
- Тяжелый процесс (havy)
- Образ процесса
- Зомби
- Демоны

Образ процесса

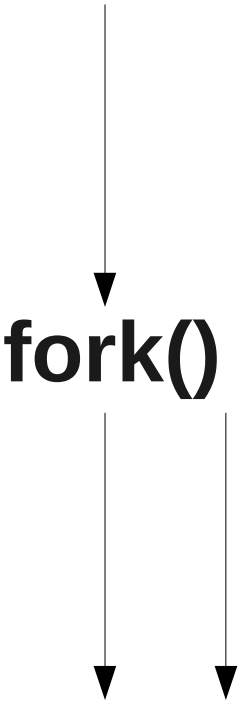


Нити

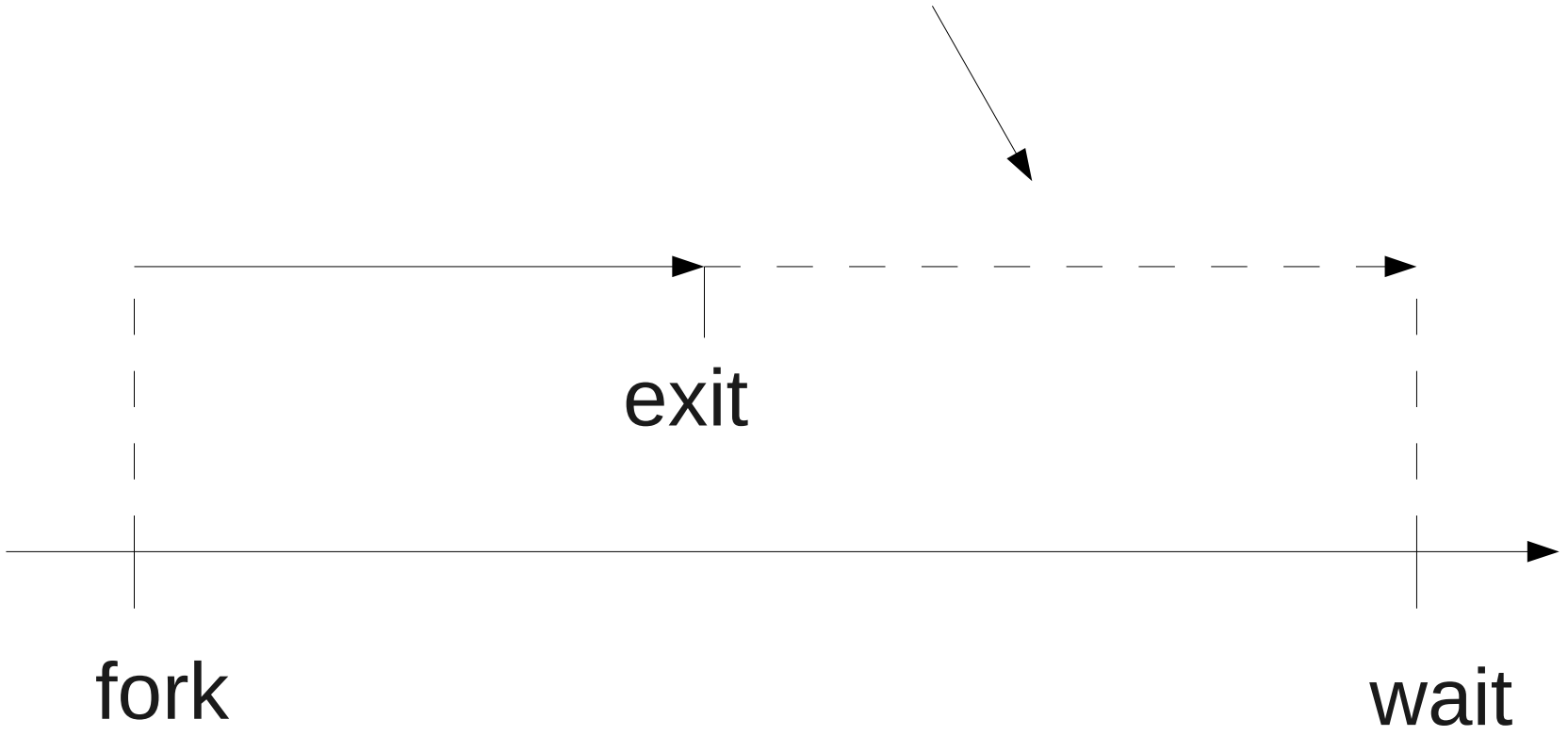


Тяжелый процесс

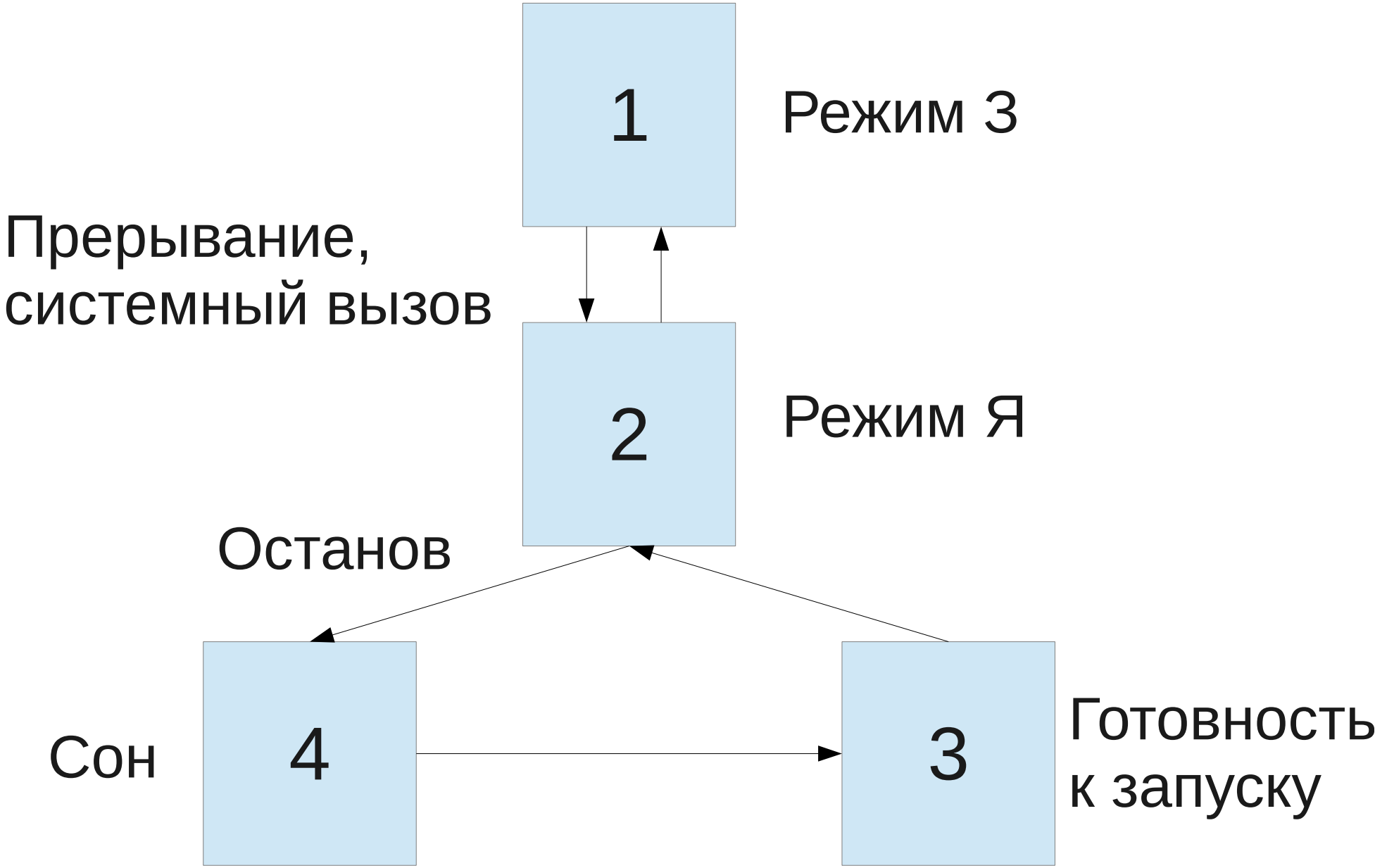
```
....
pid=fork();
if( pid == 0 )
{
 Родительский процесс
 wait(0);
}
{
Порожденный процесс
}
```



Зомби



Состояние процесса



Диспетчеризация процессов Способы задания многозадачности

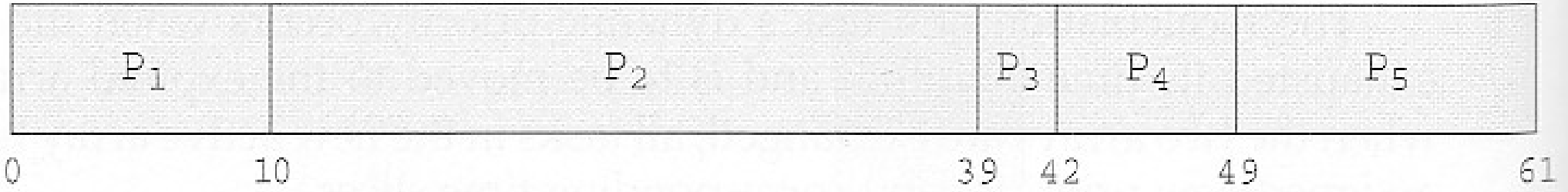
- Пакетный режим
- Невытесняющая (кооперативная)
- Вытесняющая (time-sharing)

Способы организации очередей

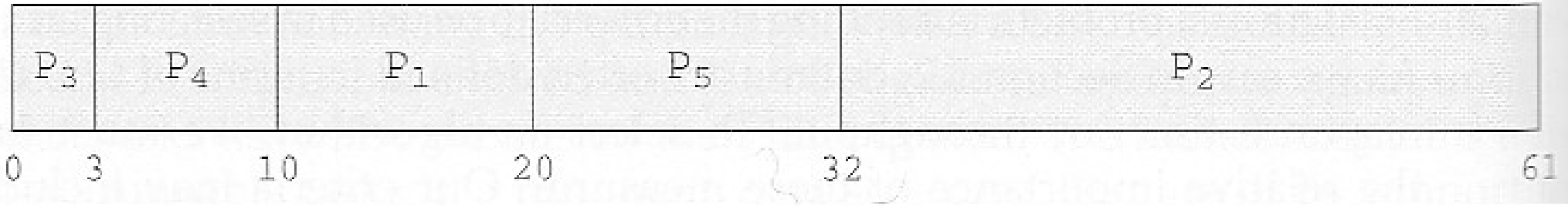
FCFS (fifo) — first come first served
SJF — shortest job first
RR — Round Robin

SJF vs FCFS

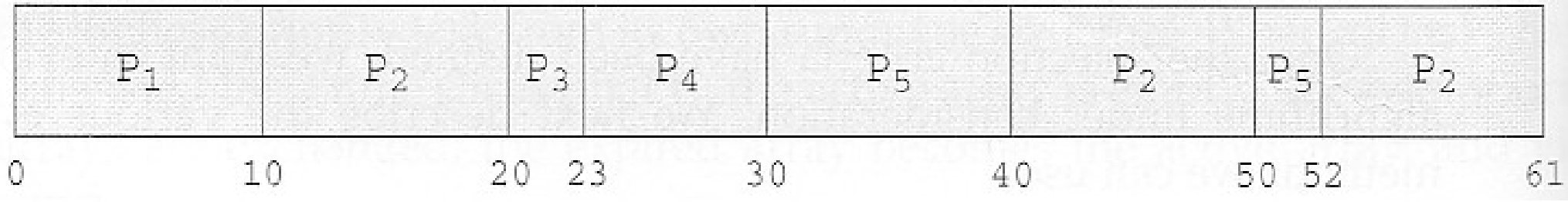
FCFS:



Non-preemptive SJF:

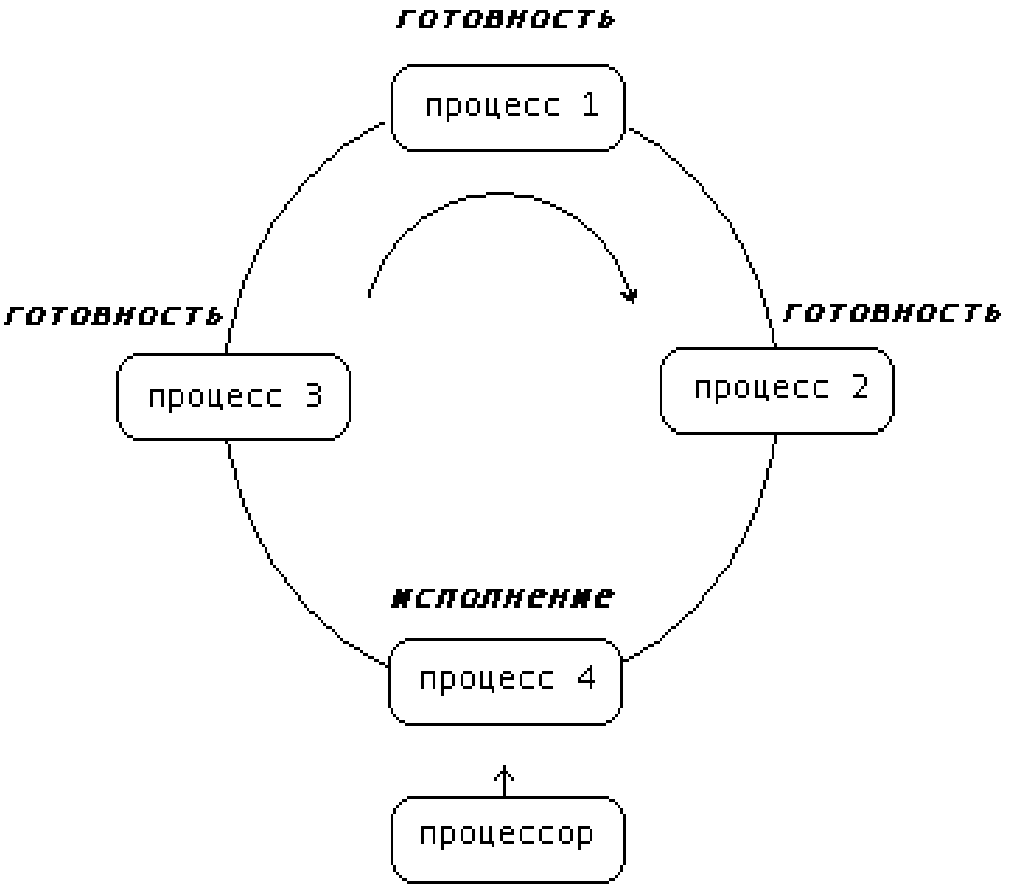


Round Robin:

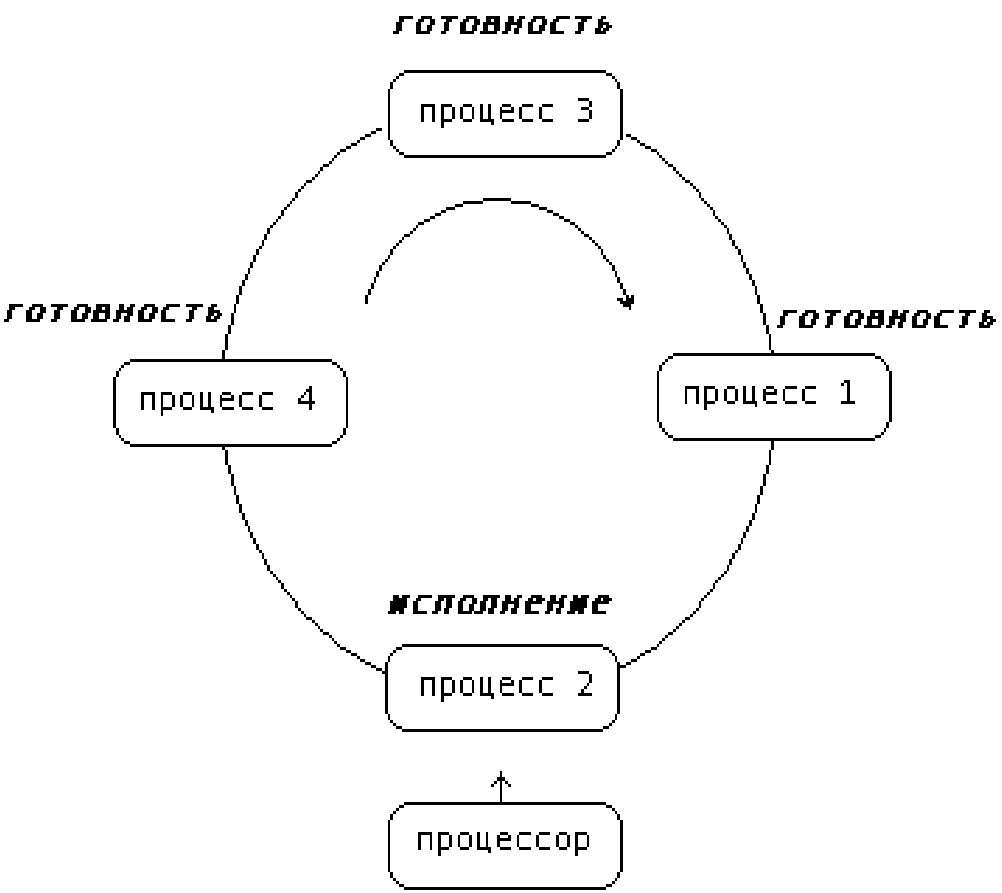


Среднее время ожидания?

RR



Начальный момент времени



По прошествии кванта

Приоритетное планирование

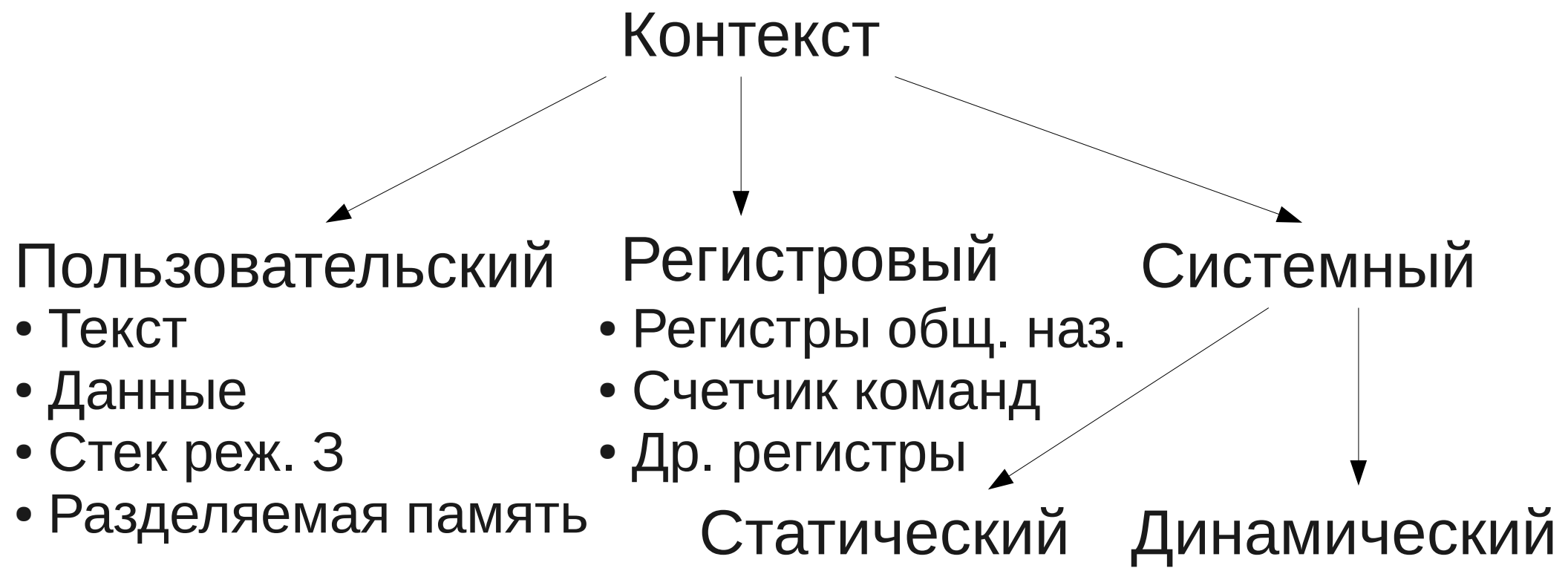
При приоритетном планировании каждому процессу присваивается определенное числовое значение — приоритет, в соответствии с которым ему выделяется процессор. Процессы с одинаковыми приоритетами планируются в порядке FCFS. Для алгоритма SJF в качестве такого приоритета выступает оценка продолжительности следующего CPU burst.

Многоуровневые очереди

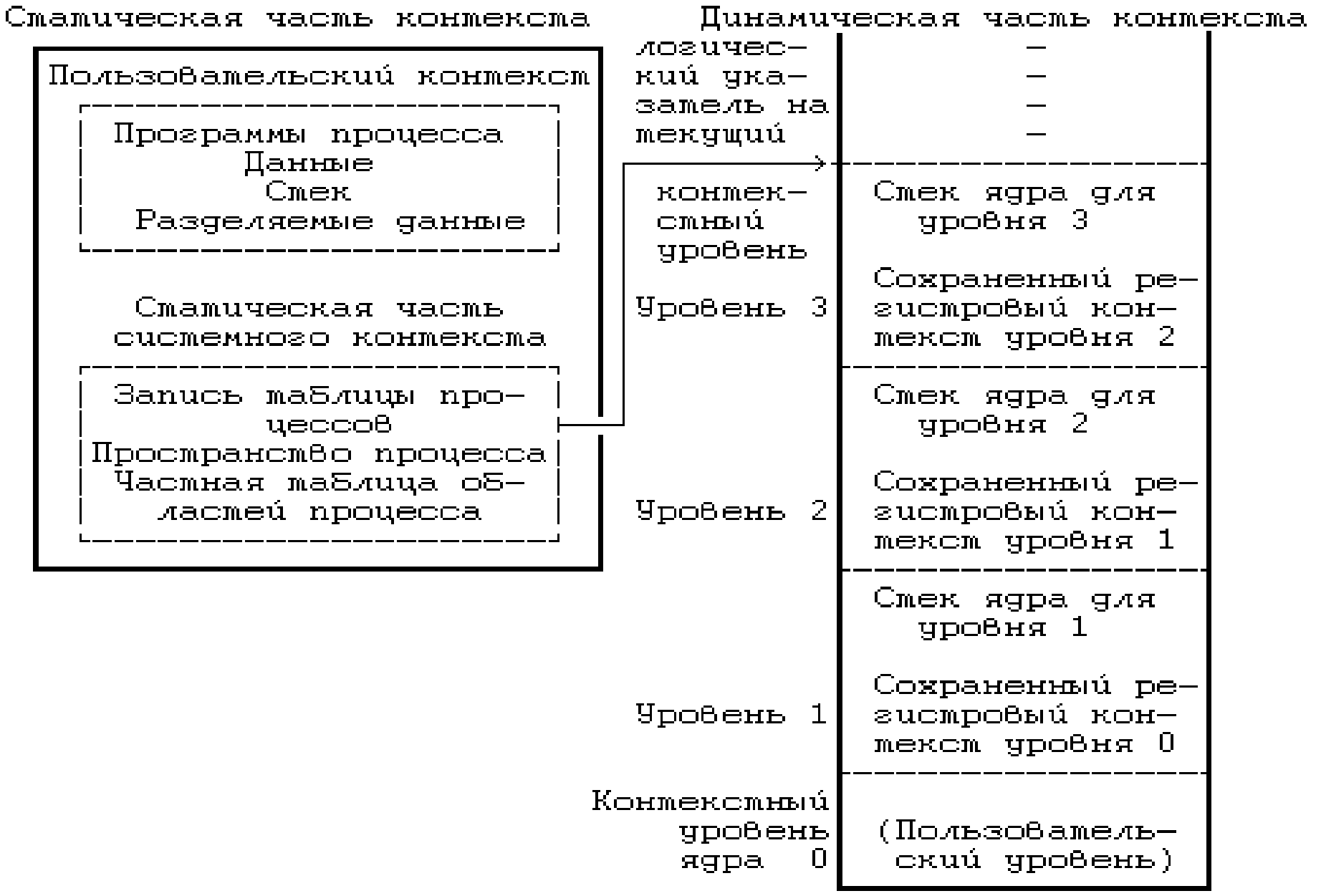
Контекст процесса

Контекстом процесса называется его состояние, определяемое текстом, значениями глобальных переменных пользователя и информационными структурами, значениями используемых машинных регистров, значениями, хранимыми в позиции таблицы процессов и в адресном пространстве задачи, а также содержимым стеков задачи и ядра, относящихся к данному процессу.

Уровни контекста



Слои контекста



Случаи сохранения контекста

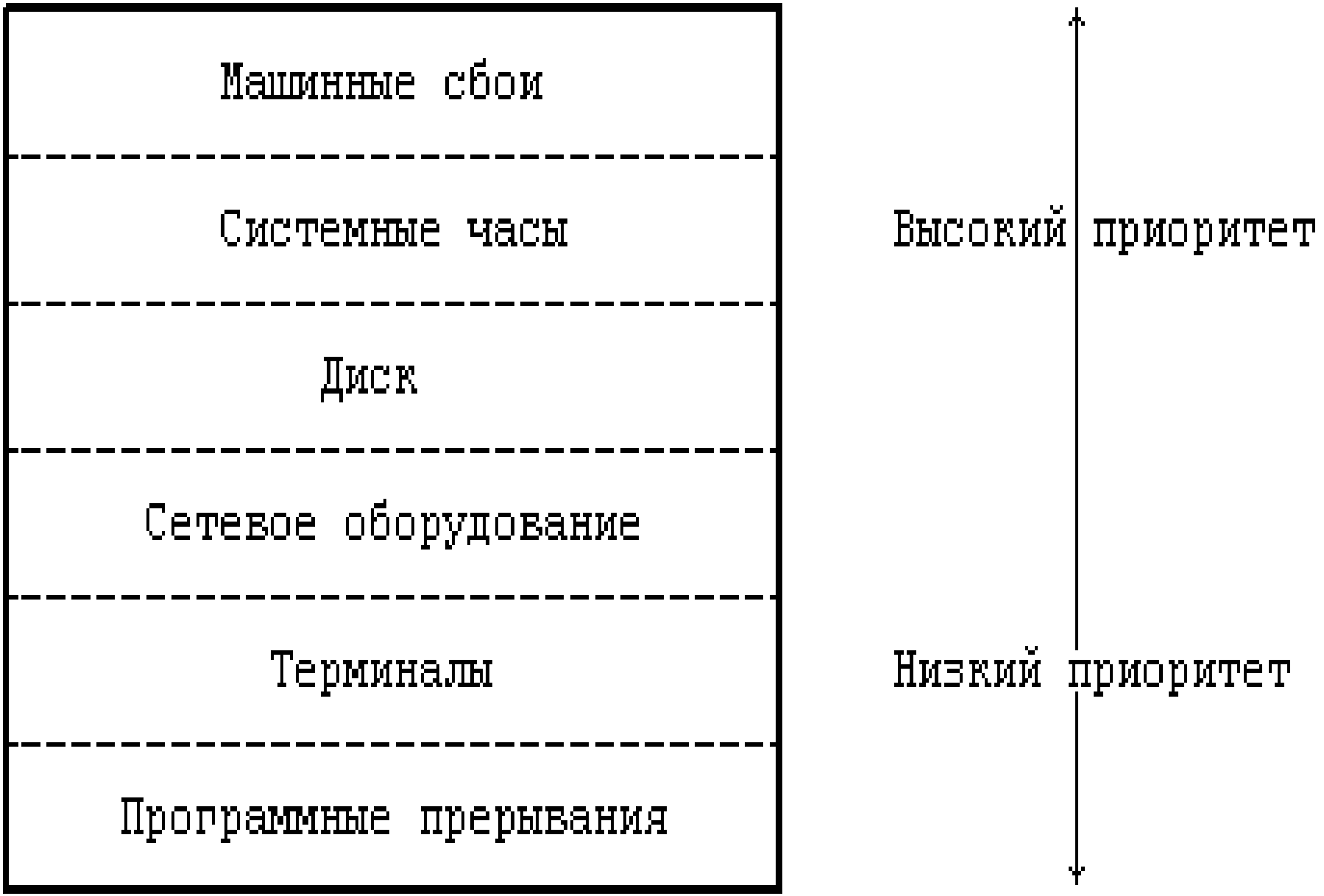
- Прерывание
- Системный вызов
- Переключение контекста

Сохранение контекста. Прерывание

Сохранение контекста. Системный вызов

Системный вызов = Программное прерывание

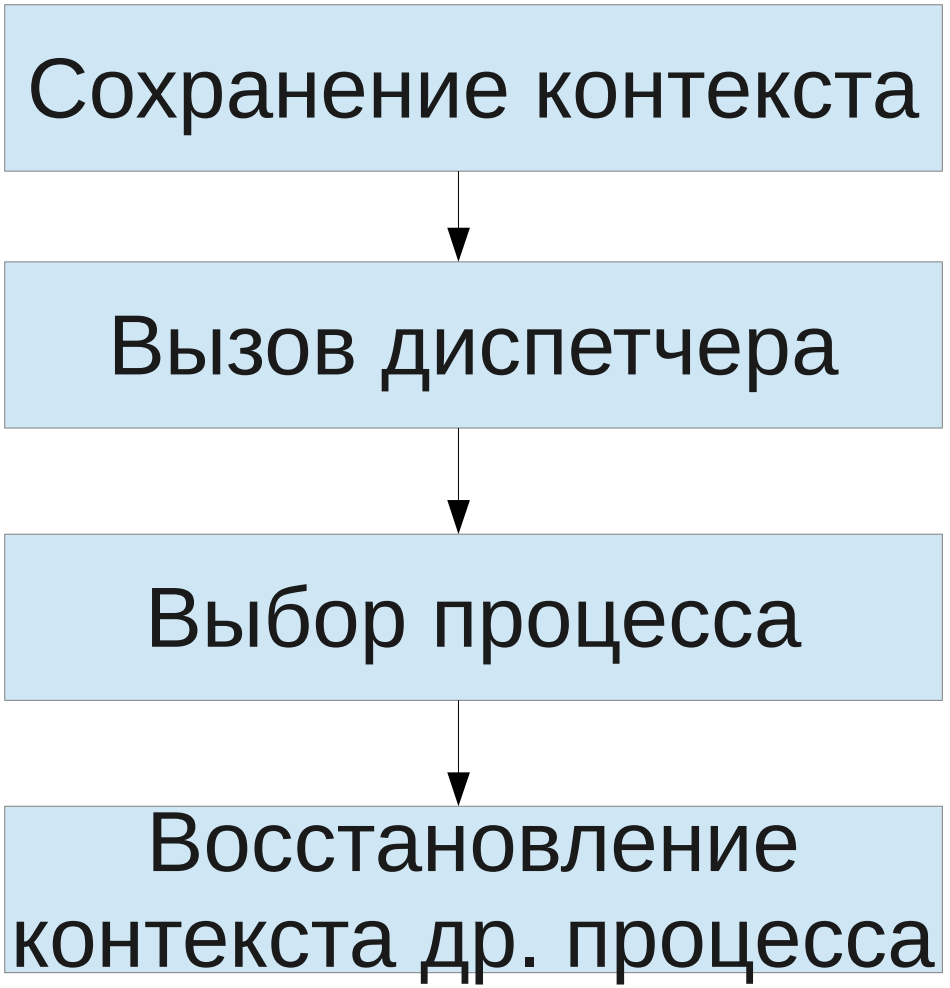
Приоритеты прерываний



Сохранение контекста. Переключение

- процесс переходит в состояние сна
- процесс делает системный вызов exit и завершает работу
- процесс переходит в режим задачи после обработки прерывания
- процесс переходит в режим задачи после системного вызова

Сохранение контекста. Переключение



Способы адресации

- Физическая адресация
- Виртуальная адресация

Недостатки физической адресации

Программа размещается в памяти в соответствии со сгенерированными компилятором адресами

- Нет защиты памяти
- Вопрос размещения программы в памяти решается на этапе компиляции
- Невозможно запустить несколько экземпляров одной программы

Виртуальная адресация

Компилятор генерирует последовательные адреса (виртуальные).

При загрузке образа процесса в память ядро выделяет свободные участки памяти (физические адреса) и генерирует таблицу соответствия (таблица трансляций, таблица страниц — ТС) между виртуальными адресами и физическими.

Выполняя код (текст) процесса, ЦП (MMU) при обращениях к памяти транслирует виртуальные адреса в физические, используя ТС.

Структура адреса

- Сегментная адресация
- Страничная адресация
- Сегментно-страничная адресация

Пример преобразования адреса

Пример:

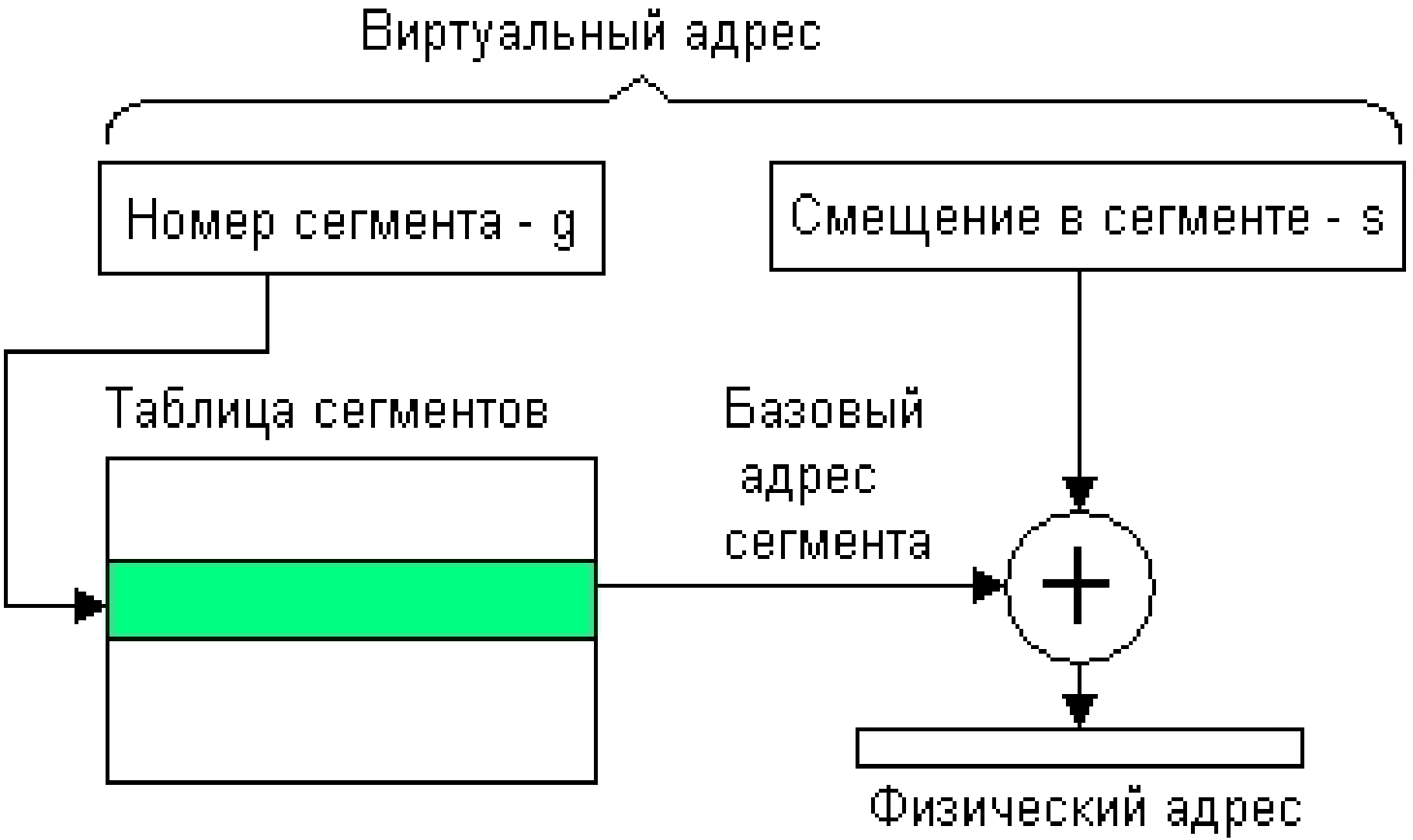
размер страницы — 1 Kb = 2^{10} b

HEX	5	8	4	3	2
BIN	0101	1000	0100	0011	0010
BIN	01	0110	0001	00	0011 0010
HEX	1	6	1	3	2

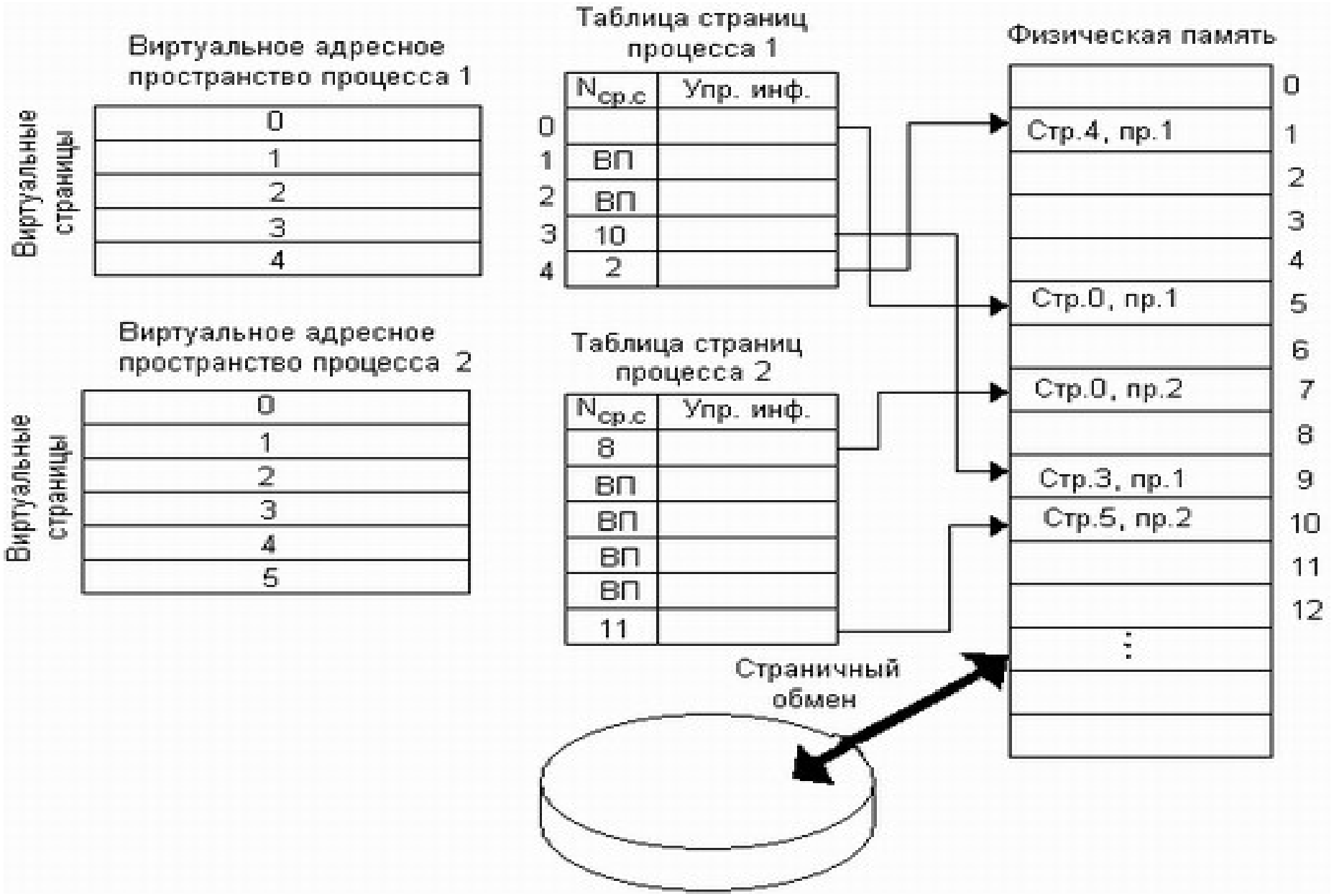
58432 = 161:32

85a23

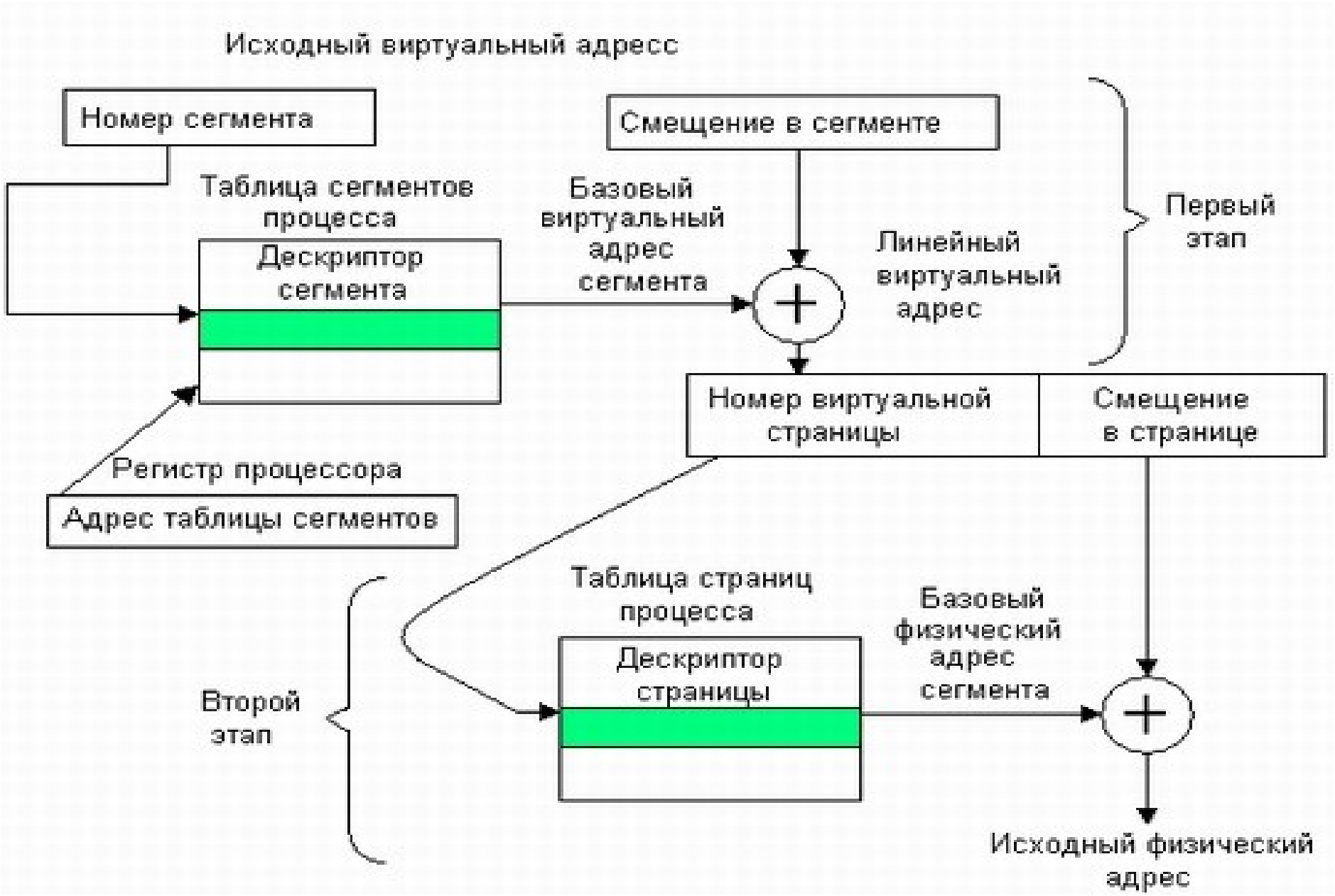
Сегментная адресация



Страничная адресация



Сегментно-страничная адресация

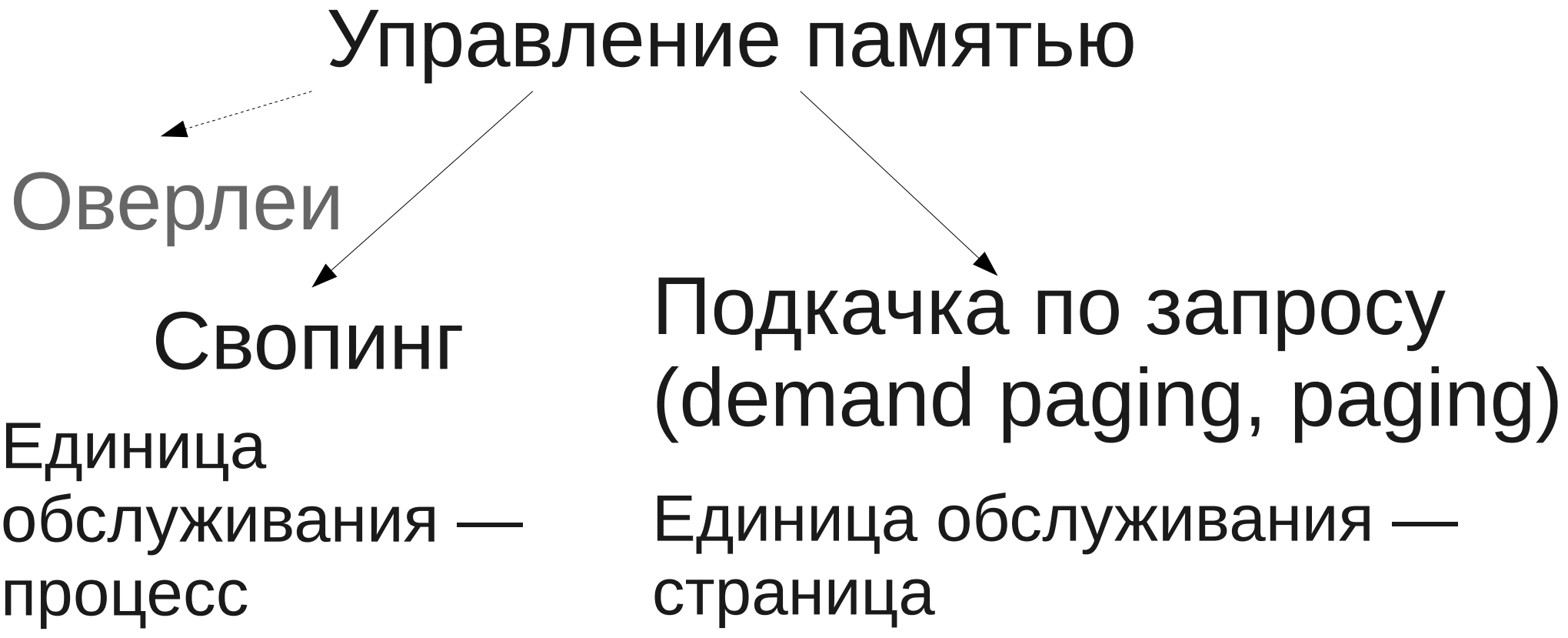


Подсистема управления памятью

1. Решает, какие процессы следует размещать в памяти.
2. Управляет участниками виртуального пространства.
3. Переписывает процессы во внешнюю память.
4. Помещает данные с устройства выгрузки в основную память.

Стратегии управления памятью

Используются при недостатке оперативной памяти



Подкачка по запросу

1) Основная память обменивается с внешней памятью не процессами, а страницами.

2) Этот способ должен иметь аппаратную поддержку: страничную организацию памяти и центральный процессор, имеющий прерываемые команды, должен быть бит упоминания страницы (reference bit), используемый для подсчета возраста страницы.

3) Отсутствуют ограничения на размер процесса, обусловленные объемом физической памяти.

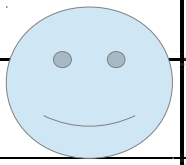
Рабочее множество процесса

Рабочее множество процесса - это совокупность страниц, использованных процессом в последних n-ссылках. Где n - окно рабочего множества процессов, число страниц, находящихся в ОП.

Фиксированное окно рабочего множества не используется.

Пример рабочего множества

Последовательность указателей страниц	рабочее множество			
	n = 2	n = 3	n = 4	n = 5
24	24	24		
15	24,15	24,15		
18	15,18	24,15,18		
23	18,23	15,18,23		
24	23,24	18,23,24		
17	24,17	23,24,17		
18	17,18	24,17,18		
24	18,24	17,18,24		
18	24,18	18,24,18		
15	18,15	18,24,15		
число ошибок	9	8	6	5



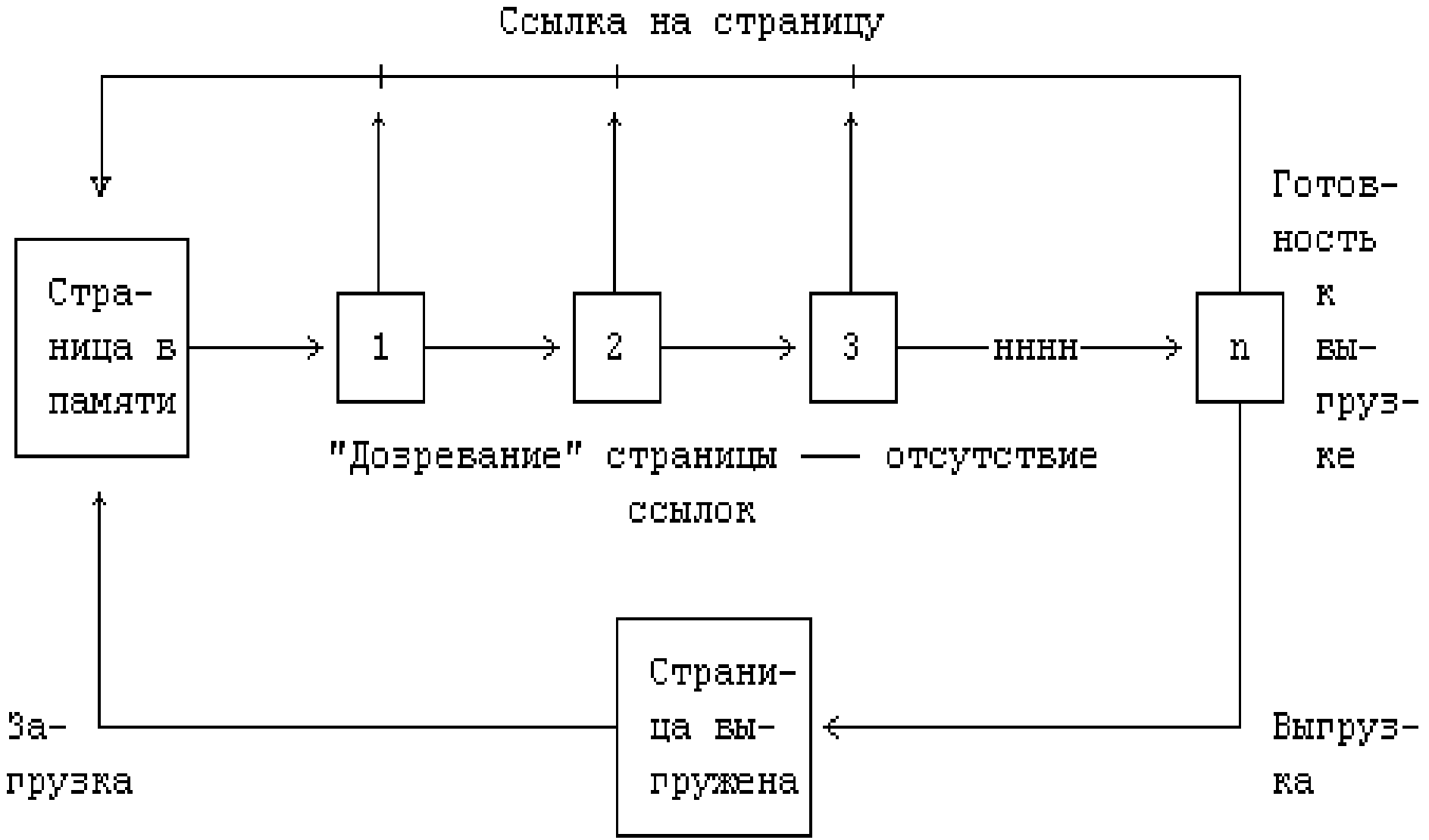
Выгрузка страниц

В случаи недостатка физической памяти, выгружаются страницы, к которым дольше всего не было обращений (обладающие большим возрастом).

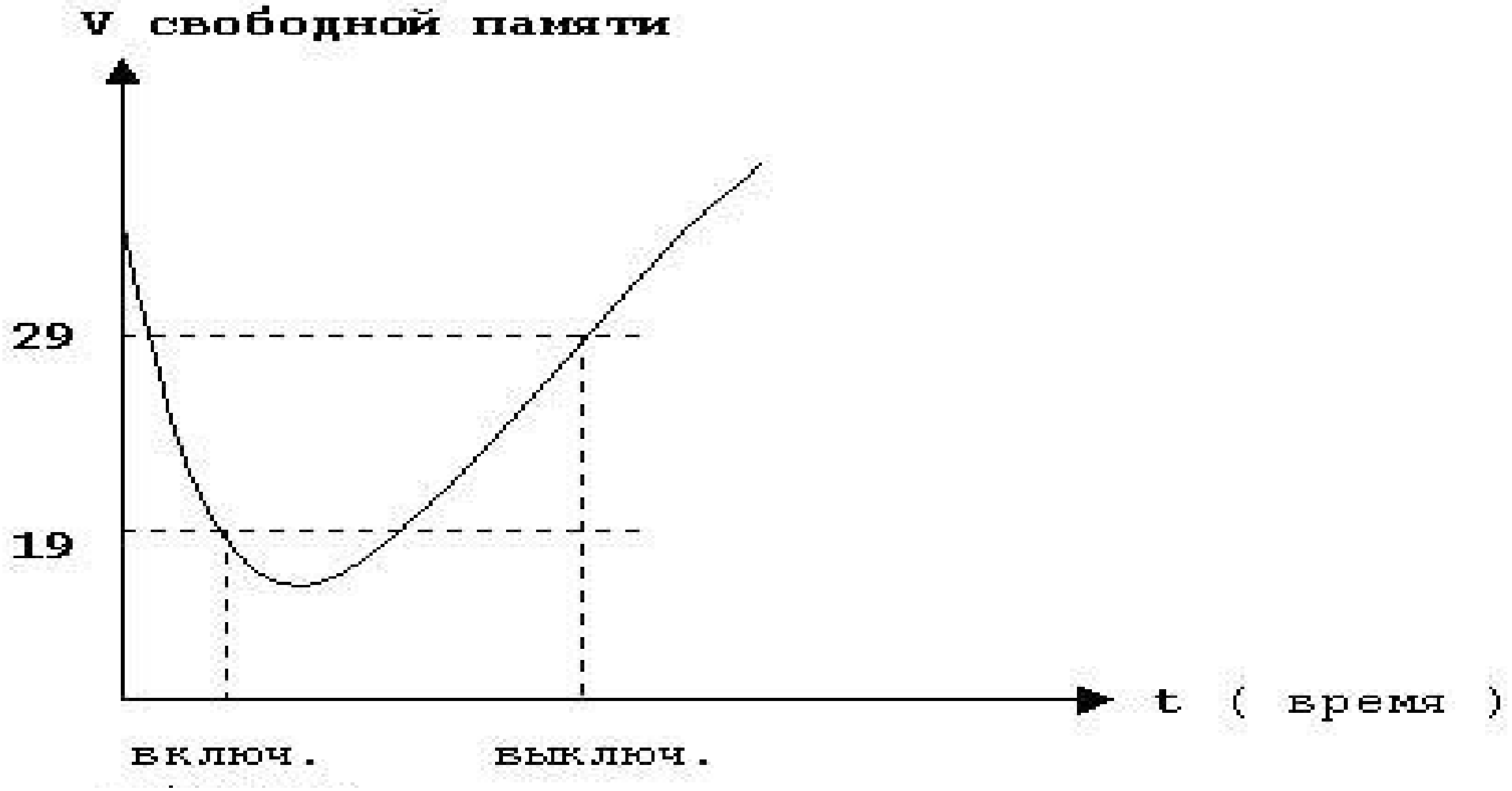
При обращении к странице ЦП устанавливает reference bit (бит упоминания).

Процесс подкачки увеличивает возраст страницы на 1, если reference bit не установлен. В противном случае возраст обнуляется, reference bit сбрасывается.

Пример старения страницы



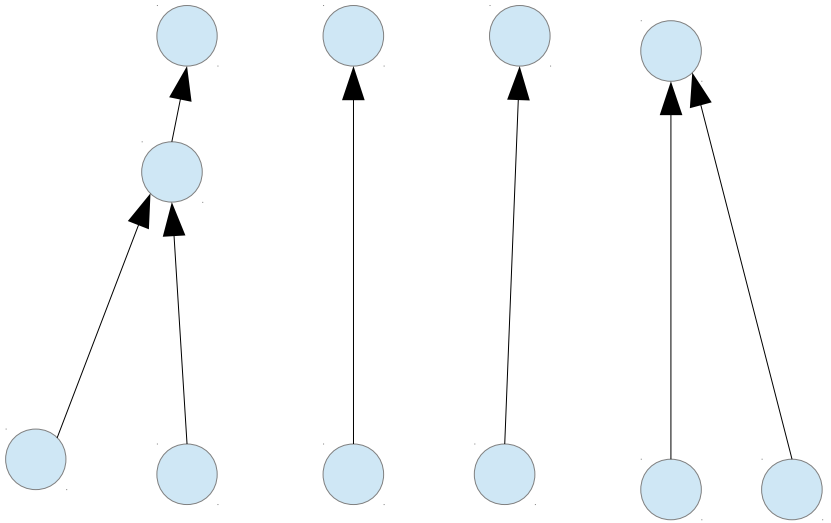
Включение откачки страниц



Управление вводом/выводом

Драйверы — модули ядра, управляющие устройствами.

Драйверы



У-ва

Драйверы бывают не только аппаратных устройств.
Пример: gs, ffmpeg, vlc

Управление вводом/выводом

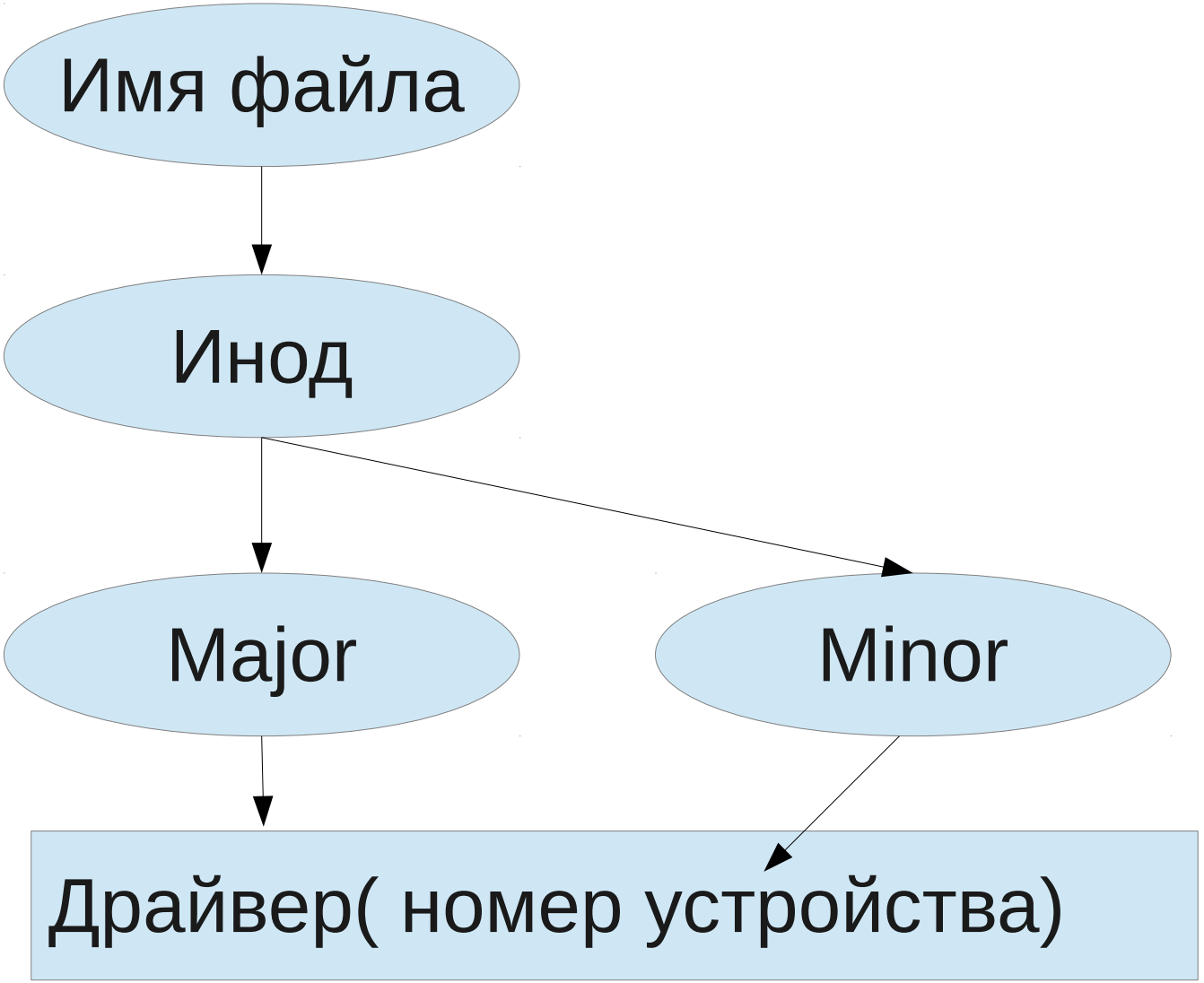
Варианты размещения драйвера:

- Включен в файл ядра
- Отдельный файл
- Виртуальный диск начальной инициализации

Варианты конфигурации устройства (выставления номера прерывания):

1. при подключении модуля
2. под подключение plug-and-play устройств
3. bios
4. перемычками

Обращение к устройству



Получение данных от устройства

- Опрос
- Прерывание
- Контроллер DMA

Опрос устройства

«Программа» вывода

```
copy_from_user(buf, p, n);
for( i=0; i<n; i++)
{
 while( *printer_status != READY ) ;
  *printer_data = p[i];
}
return_to_user();
```

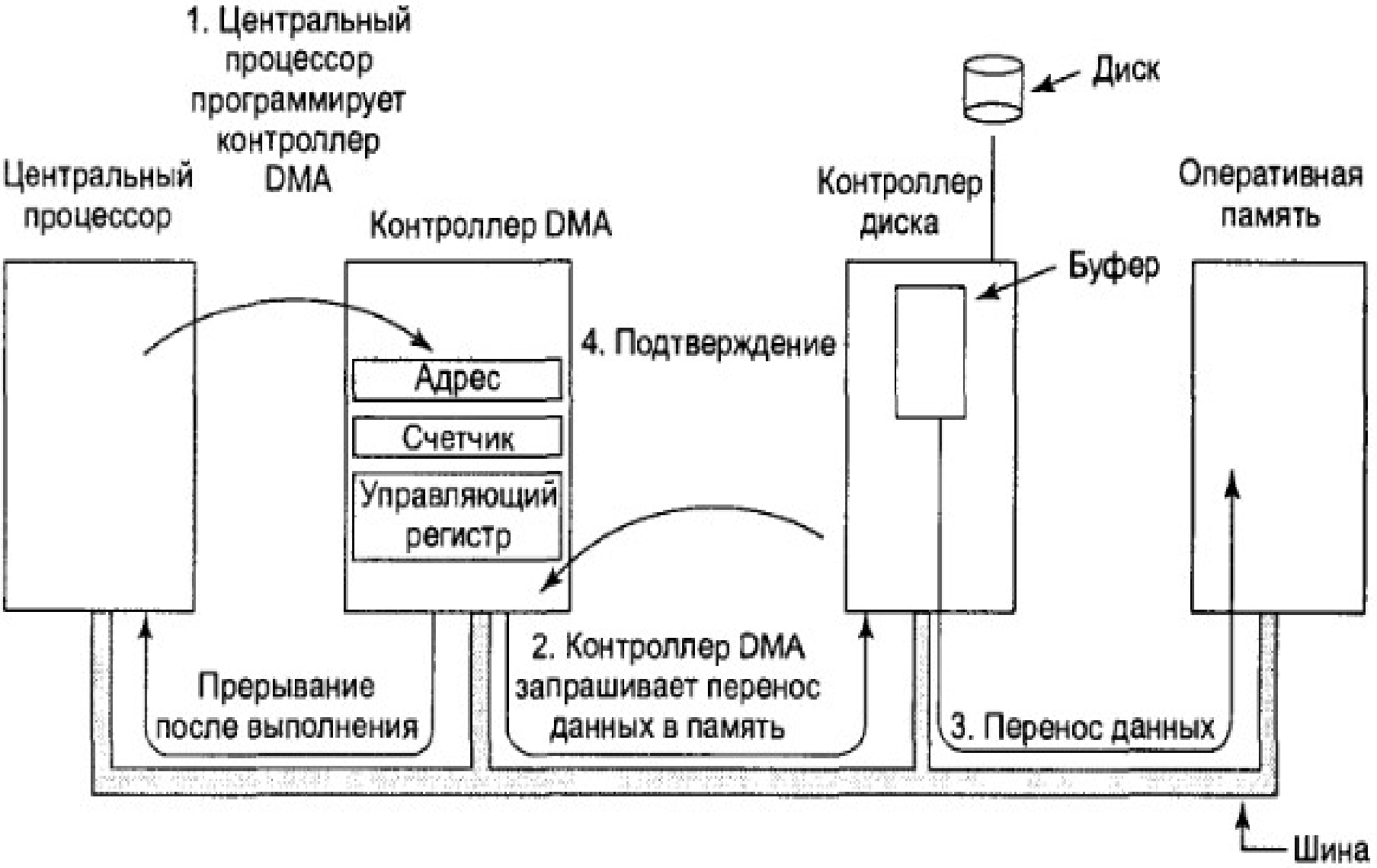
«-» — расходуется процессорное время на цикл опроса готовности устройства

Вывод по прерыванию

```
copy_from_user(buf, p, n);
while( *printer_status != READY ) ;
  *printer_data = p[0];
scheduler();
```

```
if ( n == 0 ) { /* конец */ }
else
{
  *printer_data = p[i];
 n--;
 i = i + 1;
}
return_from_interrupt();
```

Контроллер DMA



Способы взаимодействия процессов

- Сигналы
- Неименованные каналы
- Именованные каналы
- Пакет IPC
- Сокеты
- Отладка

Передача сигнала

kill (<номер процесса>, <номер сигнала>)

SIGHUP (Hang Up) опускание трубки телефона , заверш. управл. процесс

SIGINT (Interact) Ctrl+C прерывание с клавиатуры

QUIT- выход

ILL – неверная инструкция

FPE – деление на 0

KILL – нельзя обработать процессом

SEGV – нарушение сегментации

PIPE – возникновение проблем в конвейере

Обработка сигнала

Сигналы обрабатываются асинхронно. Обработчик сигнала устанавливается с помощью вызова:

signal (<номер сигнала>,<обработчик>)

SIG_INT SIG_DFL указатель на функцию

(игнорирование) (по умолчанию)

Функция – обработчик:

void <имя> (int <номер сигнала>)

Пример обработки сигнала

```
#include <signal.h>
#include <stdio.h>
void obr (int n)
{ printf ("%d",n);
}
main ()
{ signal(SIGUSR2, obr);
while (1);}
```

Неименованный канал

Применяются только при взаимодействии между _родственными_ процессами.

Создается дескриптор, состоящий из двух элементов:

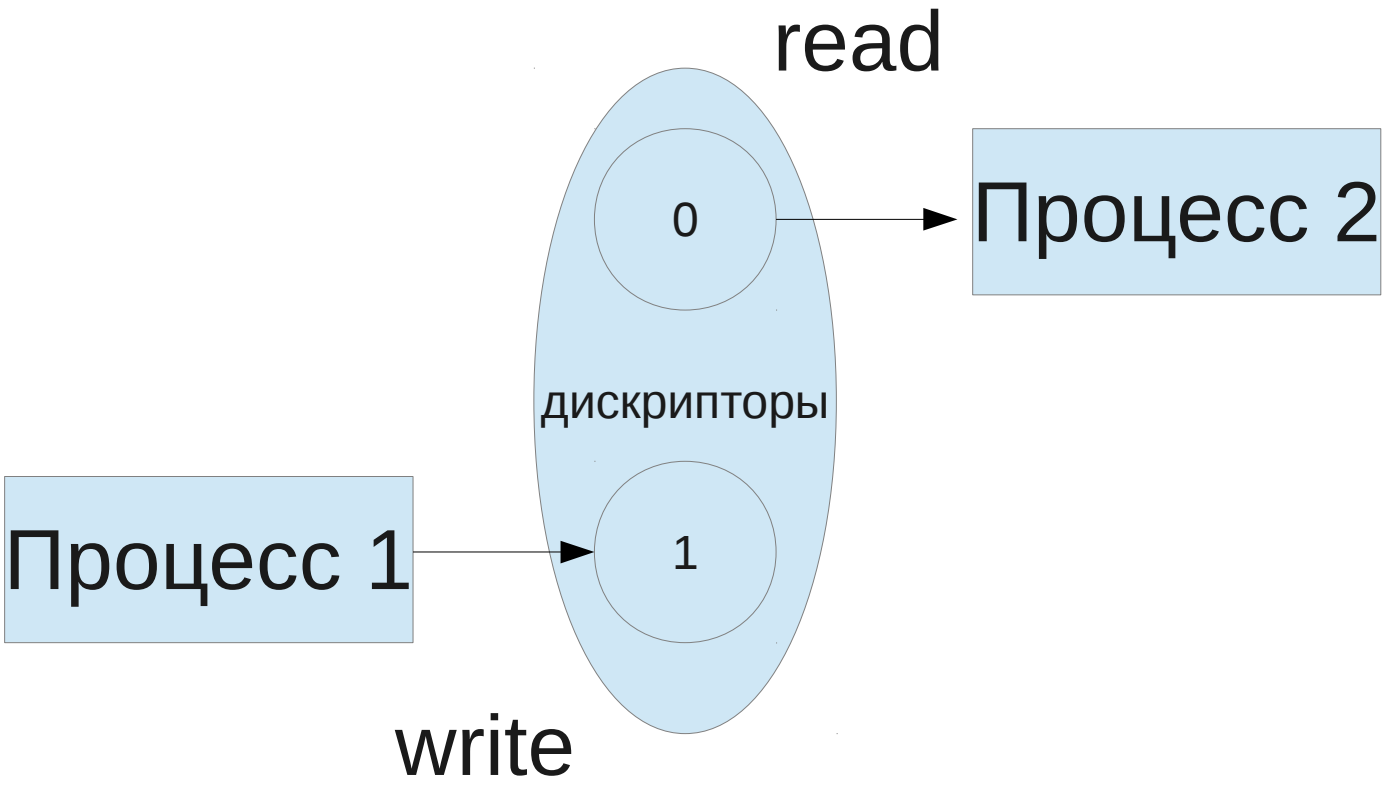
Командой pipe (fdp) он определяется.

pipe - создание неименованного канала

int pipe (fildes)

int fildes [2];

Дескрипторы канала



Именованный канал

При работе с именованным каналом создаем специальный файл:
mknod filename p
, где p - тип файла.

Пакет IPC

В IPC содержится три механизма взаимодействия:

	Сообщения	Память	Семафоры
Создание	msgget	shmget	semget
Настройка	msgctl	shmctl	semctl
Работа	msgrcv msgsnd	shmat shmdt	semop

Сокеты

Сокеты - универсальные методы взаимодействия процессов на основе использования многоуровневых сетевых протоколов.

Сокеты предназначены для работы по сети.

Создание сокета

У сокета 3 атрибута:

1) домен

2) тип

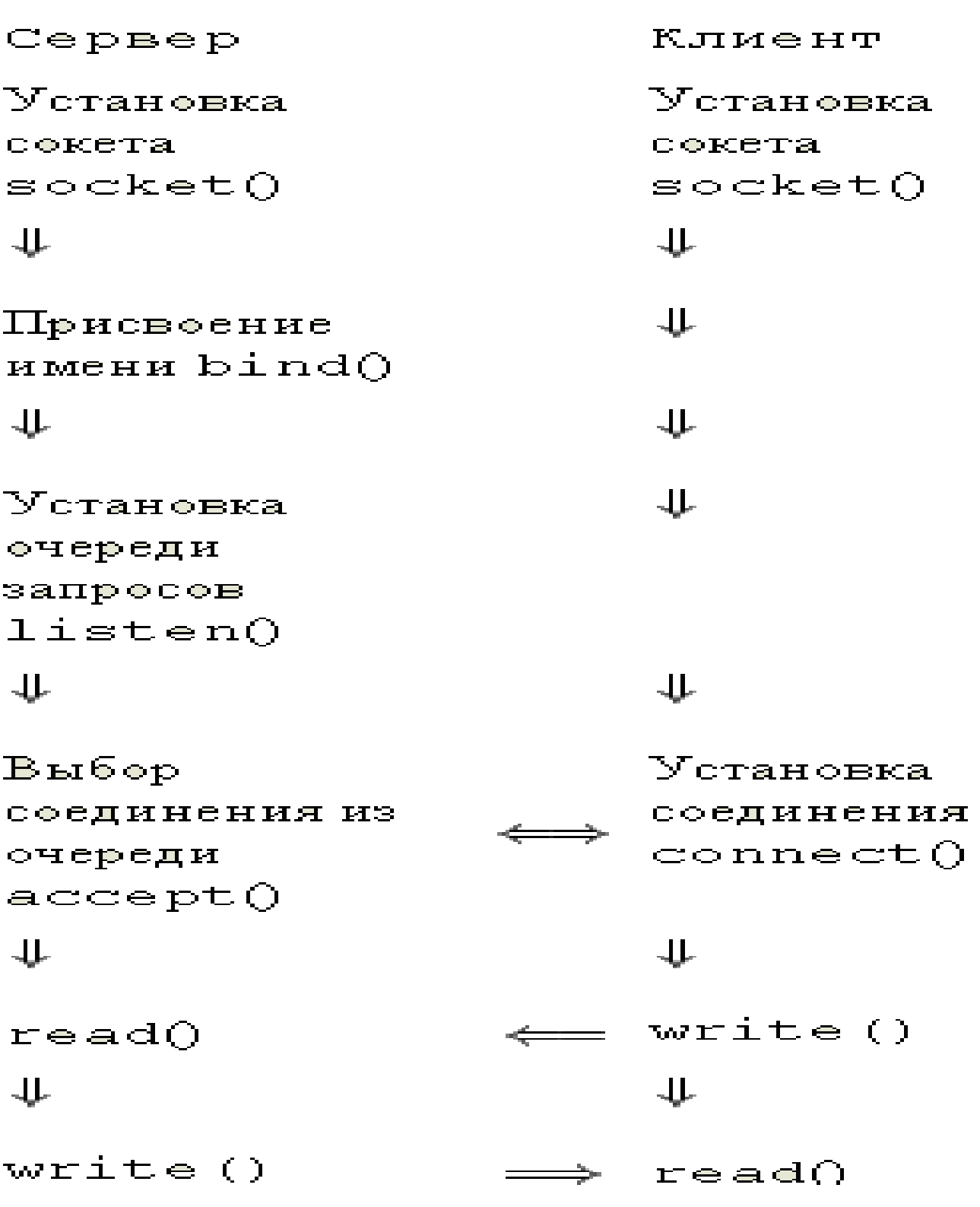
3) протокол

Для создания сокета используется системный вызов socket.

s = socket(domain, type, protocol);

Пример:

s = socket(AF_INET, SOCK_STREAM, 0);


Сервер
Клиент
Установка сокета socket()
Установка сокета socket()
Присвоение имени bind()
Установка очереди запросов listen()
Выбор соединения из очереди accept()
Установка соединения connect()
read()
write ()
write ()
read()