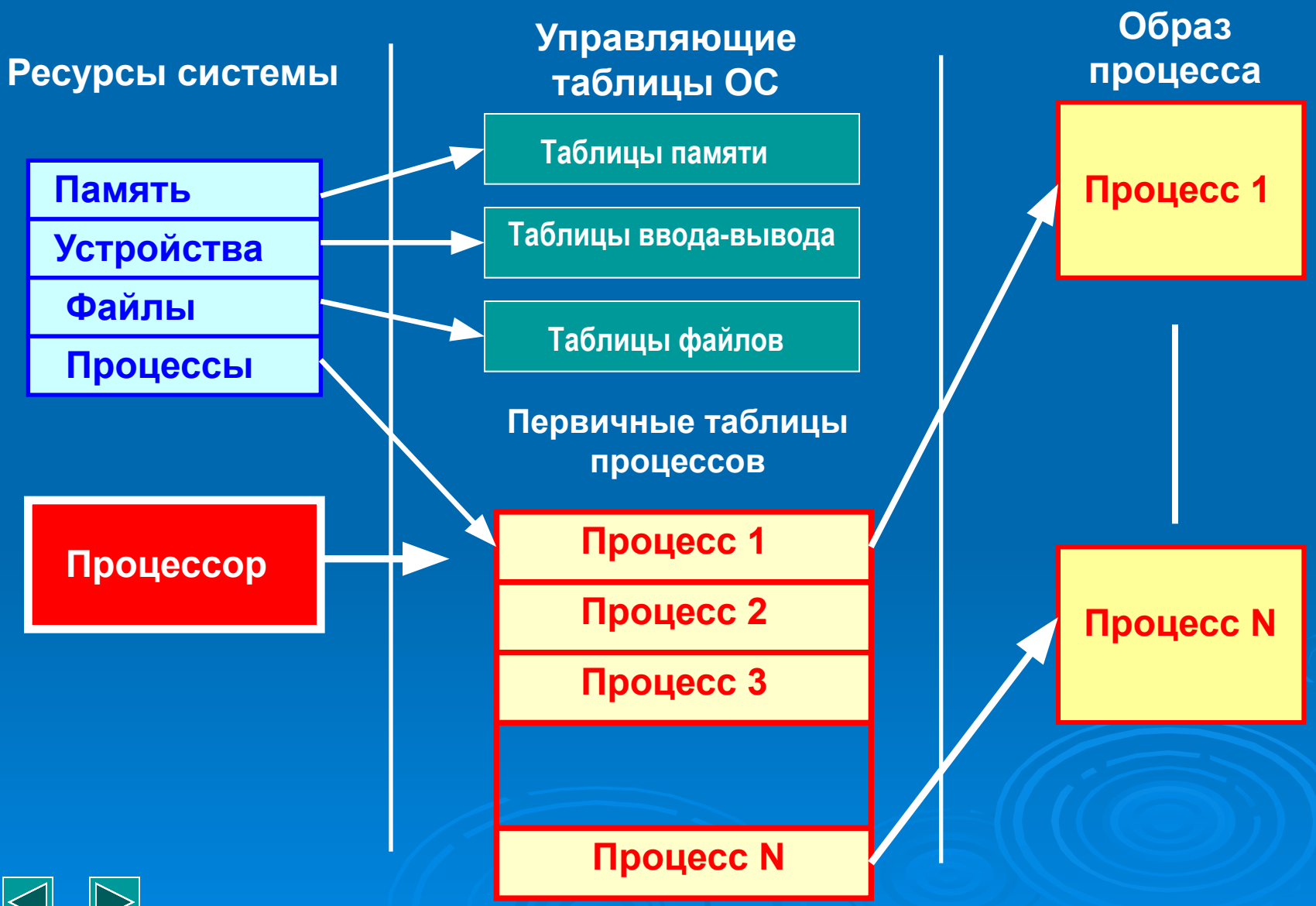
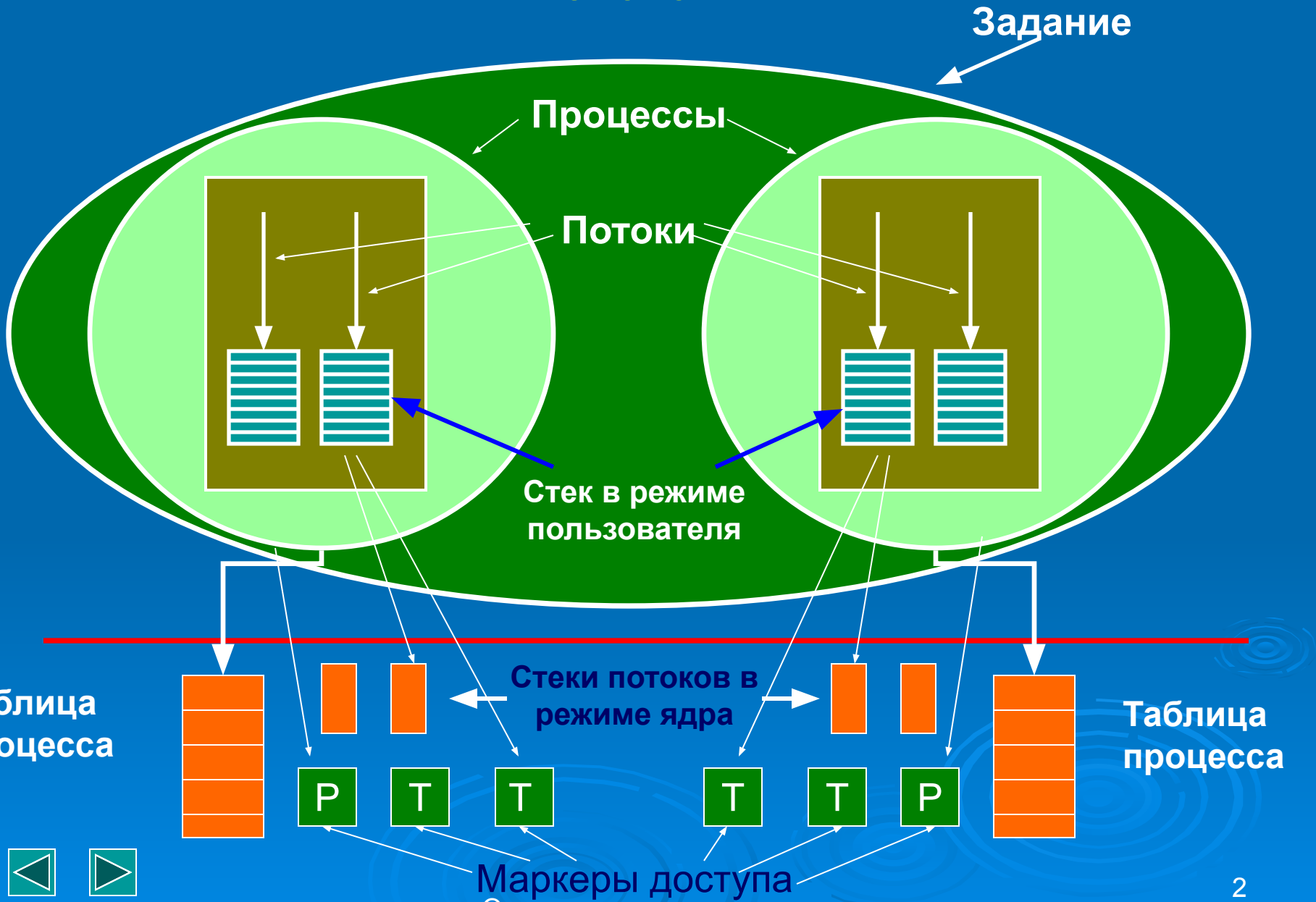
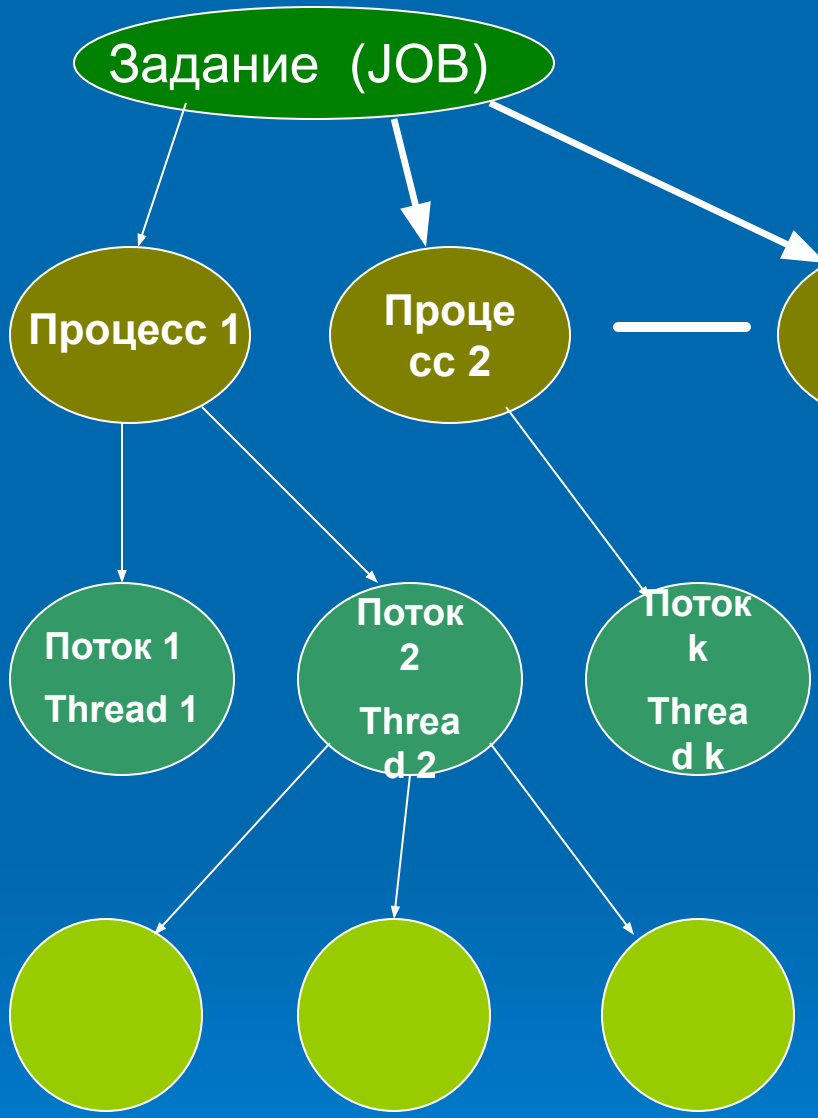
2.1. Концепция процессов и потоков. Задания, процессы, потоки (нити), волокна



Взаимосвязь между заданиями, процессами и потоками



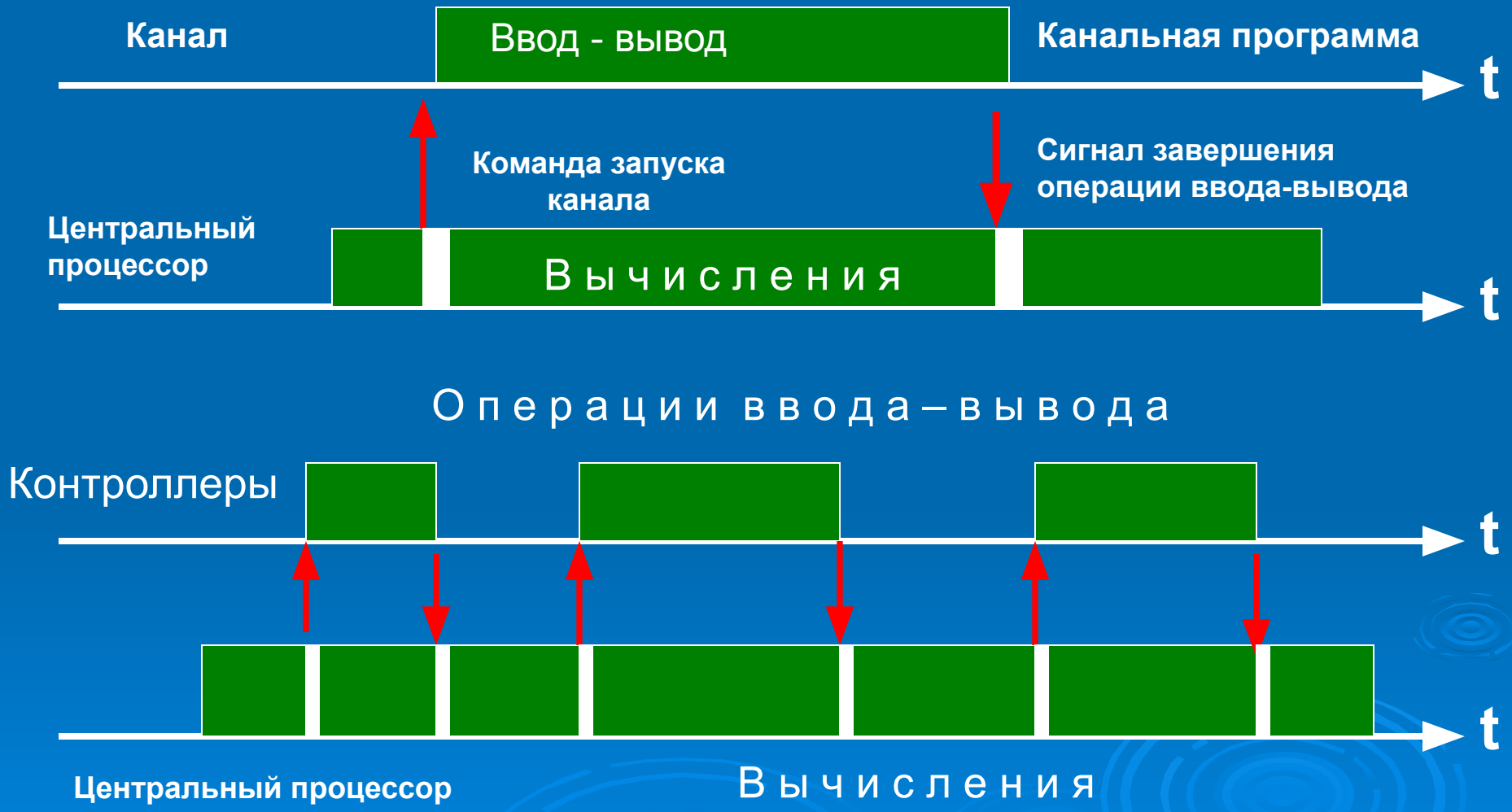


Объекты

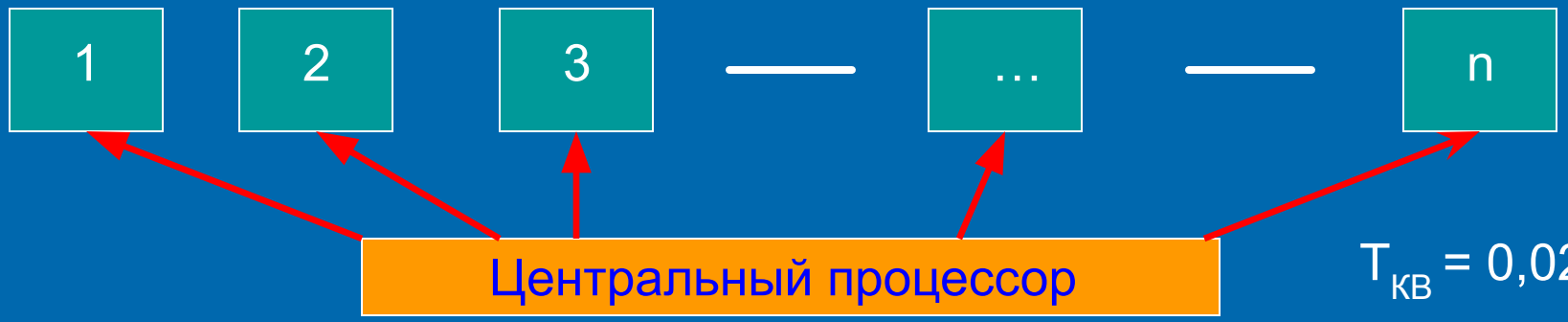
Название	Описание
Задание	Набор процессов с общими квотами и лимитами
Процесс	Контейнер для ресурсов и потоков
Поток	Исполнение кода в процессе
Волокно (нить)	Облегченный поток, полностью управляемый в пространстве пользователя

2.2. Мультипрограммирование. Формы многопрограммной работы

2.2.1. Мультипрограммирование в системах пакетной обработки



2.2.2. Мультипрограммирование в системах разделения времени



$T_{кв}$ = 0,02 мс

2.2.3. Мультипрограммирование в системах реального времени

1. **Управление техническими объектами, технологическими процессами, системами обслуживания и т. п.**
2. **Фиксированный набор заранее разработанных задач.**
3. **Жесткие ограничения на время обслуживания.**
4. **Режим типа запрос – ответ.**

2.2.4. Мультипроцессорная обработка

1. Операционные системы : Windows NT/2000/2003, Sun Solaris 2/x, Santa Cruz Operations Open Server 3.x, OS/2 и др.
2. Симметричная архитектура и асимметричная архитектура.

2.3.2. Роль процессов, потоков и волокон в мультипрограммировании

1. Отдельный процесс не может быть выполнен быстрее, чем в однопрограммном режиме.
2. Сложно создать программу, реализующую параллелизм в рамках одного процесса.
3. Стандартные средства современных ОС не позволяют создать для одного приложения несколько процессов для параллельных работ.
4. Многопоточная обработка позволяет распараллелить вычисления в рамках одного процесса.
5. Многопоточная (multithreading) обработка эффективна в многопроцессорных вычислительных системах.
6. Использование аппарата волокон (Windows 2000) повышает эффективность мультипрограммирования за счет сокращения переключения процессов, но увеличивает трудоемкость разработки приложений.

2.3. Управление процессами и потоками

2.3.1. Основные функции управления процессами и потоками

1. Создание процессов и потоков.
2. Обеспечение процессов и потоков необходимыми ресурсами.
3. Изоляция процессов.
4. Планирование выполнения процессов и потоков.
5. Диспетчеризация потоков.
6. Синхронизация процессов и потоков.
7. Завершение и уничтожение процессов и потоков.

События, приводящие к созданию процессов:

1. Инициализация (загрузка) ОС.
2. Запрос процесса на создание дочернего процесса.
3. Запрос пользователя на создание процесса (например, при входе в систему в интерактивном режиме).
4. Инициирование пакетного задания.
5. Создание операционной системой процесса какой-либо службы.

2.4. Создание процессов и потоков. Модели процессов и потоков

2.4.1.Процессы и их модели

Образ процесса: программа, данные, стек и атрибуты процесса

Информация	Описание
Данные пользователя	Изменяемая часть пользовательского адресного пространства (данные программы, пользовательский стек и модифицируемый код)
Пользовательская программа	Программа, которую нужно выполнить
Системный стек	Один или несколько системных стеков для хранения параметров и адресов вызова процедур и системных служб
Управляющий блок процесса	Данные, необходимые ОС для управления процессом: 1) дескриптор процесса, 2) контекст процесса

Дескриптор процесса содержит:

1. **Информацию по идентификации процесса (идентификатор процесса, идентификатор пользователя, идентификаторы родительского и дочерних процессов).**
2. **Информацию по состоянию процесса**
3. **Информацию, используемую для управления процессом**

Информация по состоянию и управлению процессом

- Состояние процесса, определяющее его готовность к выполнению (выполняющийся, готовый к выполнению, ожидающий события, приостановленный);
- Данные о приоритете (текущий, по умолчанию, максимально возможный);
- Информация о событиях – идентификация события, наступление которого позволит продолжить выполнение процесса;
- Указатели, позволяющие определить расположение образа процесса в оперативной памяти и на диске;
- Указатели на другие процессы (находящиеся в очереди на выполнение);
- Флаги,сигналы и сообщения, имеющие отношение к обмену информацией между двумя независимыми процессами;
- Данные о привилегиях, определяющие прав доступа к определенной области памяти или возможности выполнять определенные виды команд, использовать системные утилиты и службы;
- Указатели на ресурсы, которыми управляет процесс;
- Сведения по использованию ресурсов и процессора;
- Информация, связанная с планированием.

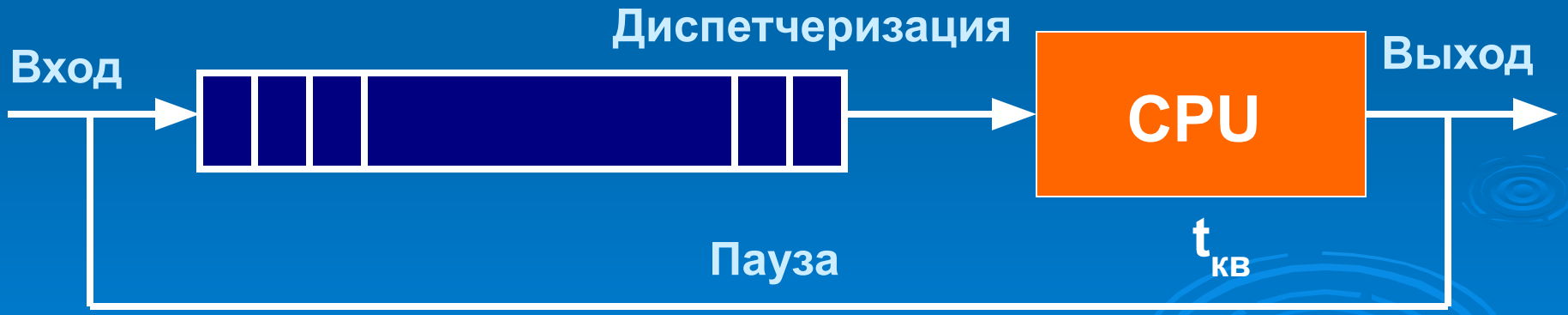
КОНТЕКСТ ПРОЦЕССА

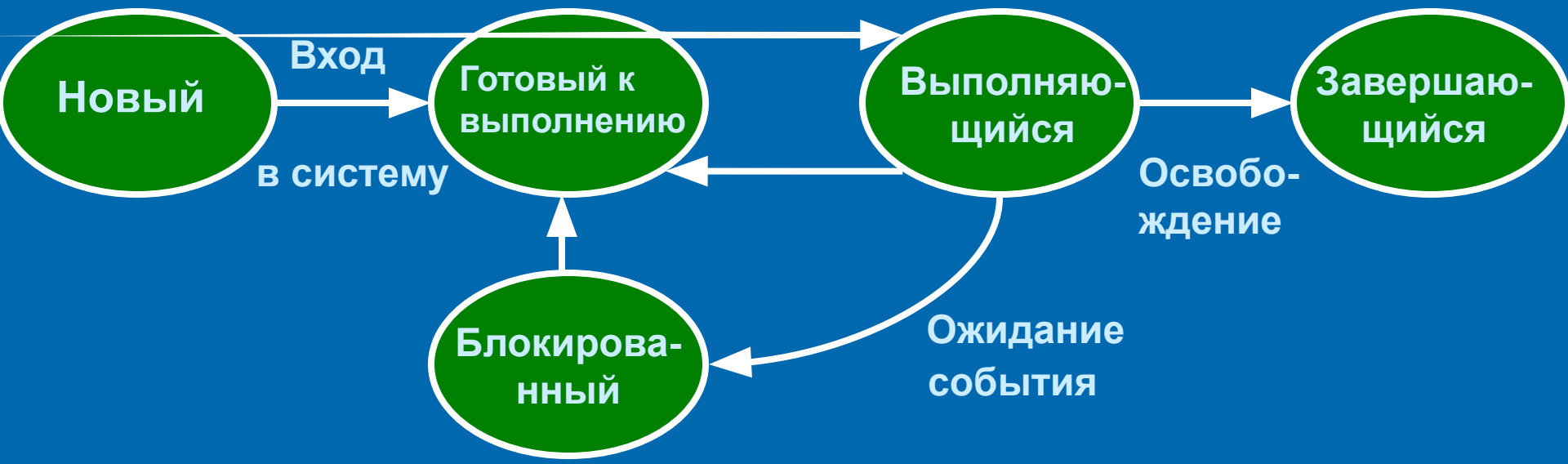
- Содержимое регистров процессора, доступных пользователю (обычно 8 – 32 регистра и до 100 регистров в RISC – процессорах);
- Содержимое счетчика команд;
- Состояние управляющих регистров и регистров состояния;
- Коды условия, отражающие результат выполнения последней арифметической или логической операции (например, равенство нулю,переполнение);
- Указатели вершин стеков,хранящие параметры и адреса вызова процедур и системных служб.

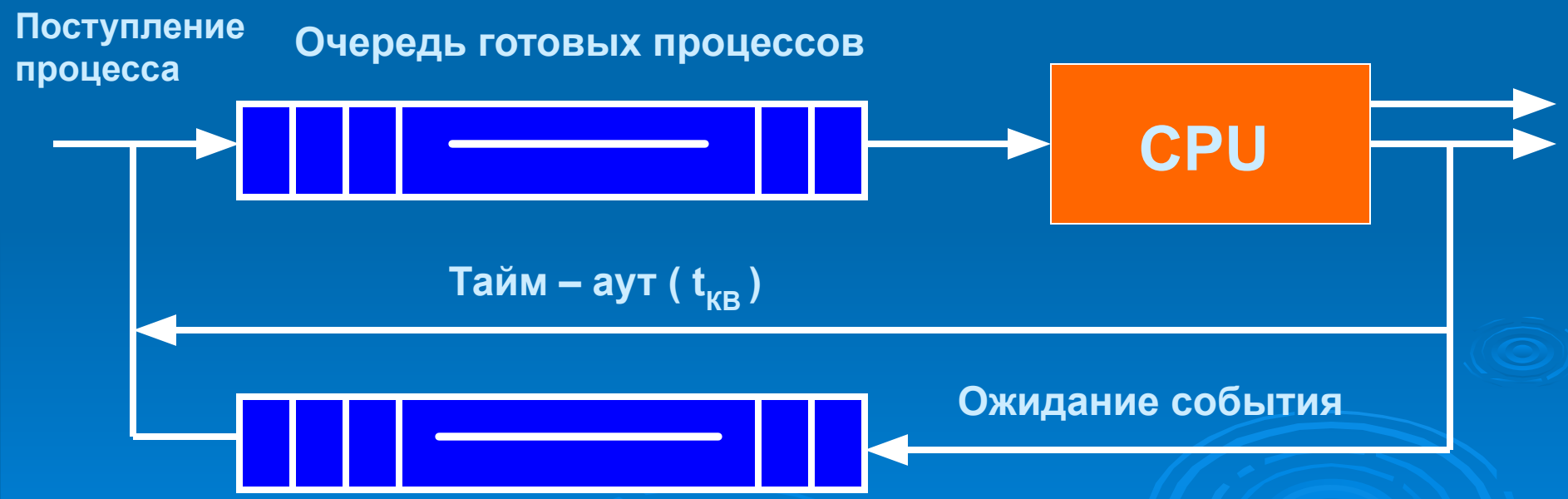
Значительная часть этой информации фиксируется в виде слова состояния программы PSW (program status word – EFLAGS в процессоре Pentium).

Простейшая модель процесса

Граф состояний и переходов




Новый
Вход
в систему
Готовый к выполнению
Выполняю-щийся
Завершаю-щийся
Освобо-ждение
Блокирова-нный
Ожидание события


Поступление процесса
Очередь готовых процессов
CPU
Тайм – аут ($t_{кв}$)
Ожидание события

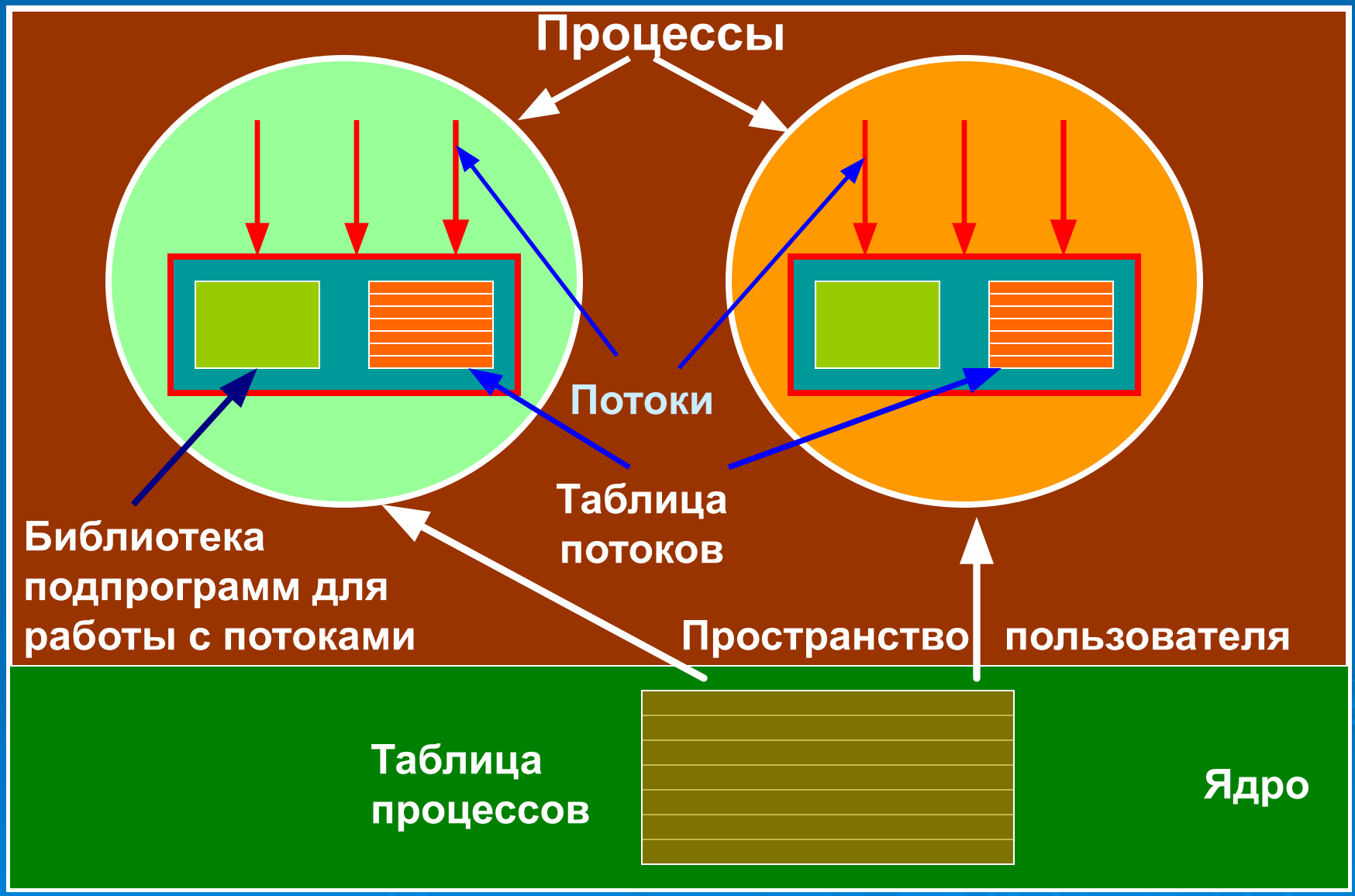
2.4.2. Потоки и их модели

Описатель потока: блок управления потоком и контекст потока (в многопоточной системе процессы контекстов не имеют).

Способы реализации пакета потоков:

- в пространстве пользователя (user – level threads – ULT);
- в ядре (kernel – level threads – KLT).

Поток на уровне пользователя (в пользовательском пространстве)



Поток на уровне пользователя

ДОСТОИНСТВА:

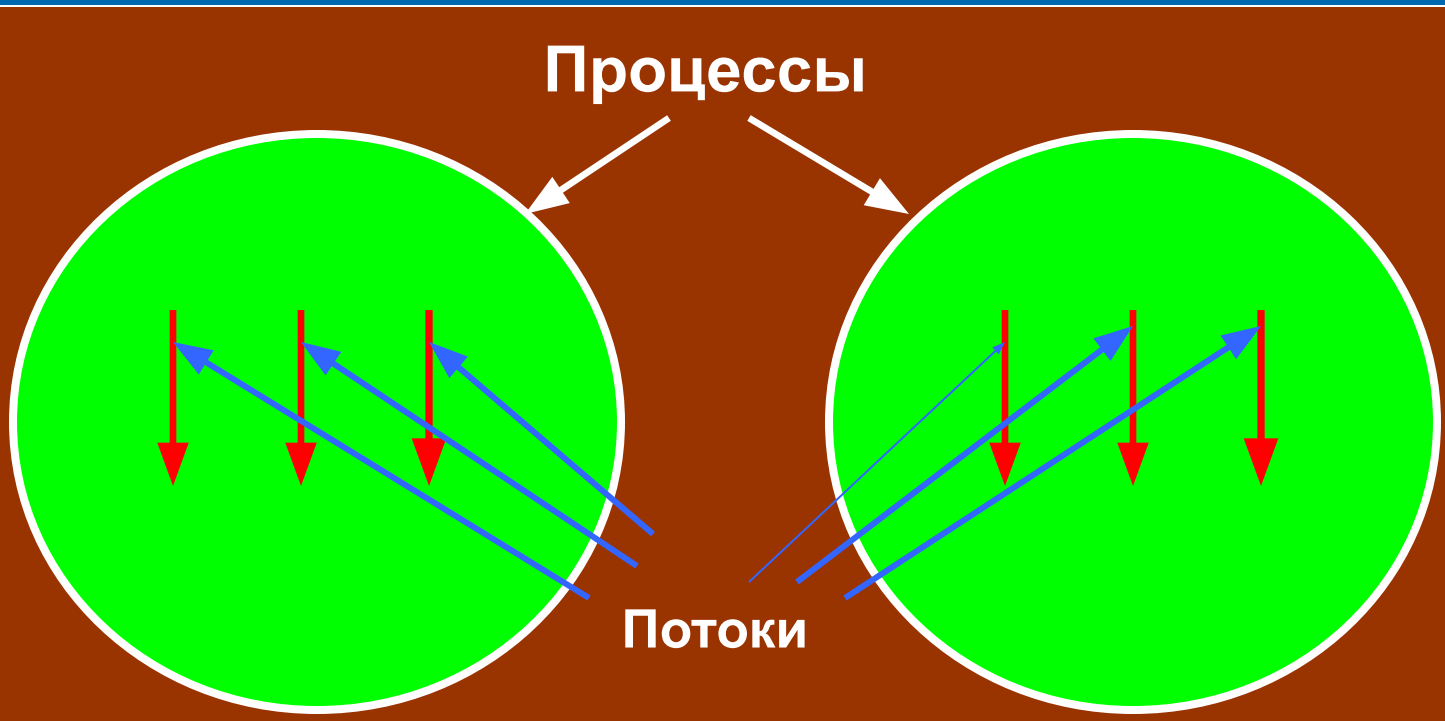
- можно реализовать в ОС, не поддерживающей потоки без каких-либо изменений в ОС;
- высокая производительность, поскольку процессу не нужно переключаться в режим ядра и обратно;
- ядро о потоках ничего не знает и управляет однопоточными процессами;
- имеется возможность использования любых алгоритмов планирования потоков с учетом их специфики;
- управление потоками возлагается на программу пользователя.

Поток на уровне пользователя

НЕДОСТАТКИ:

- системный вызов блокирует не только работающий поток, но и все потоки того процесса, к которому он относится;
- приложение не может работать в многопроцессорном режиме, так как ядро закрепляет за каждым процессом только один процессор;
- при запуске одного потока ни один другой поток в рамках одного процесса не будет запущен пока первый добровольно не отдаст процессор;
- внутри одного потока нет прерываний по таймеру, в результате чего невозможно создать планировщик по таймеру для поочередного выполнения потоков.

Поток на уровне ядра



Поток на уровне ядра

ДОСТОИНСТВА:

- возможно планирование работы нескольких потоков одного и того же процесса на нескольких процессорах;
- реализуется мультипрограммирование в рамках всех процессов (в том числе одного);
- при блокировании одного из потоков процесса ядро может выбрать другой поток этого же (или другого процесса);
- процедуры ядра могут быть многопоточными.

НЕДОСТАТКИ:

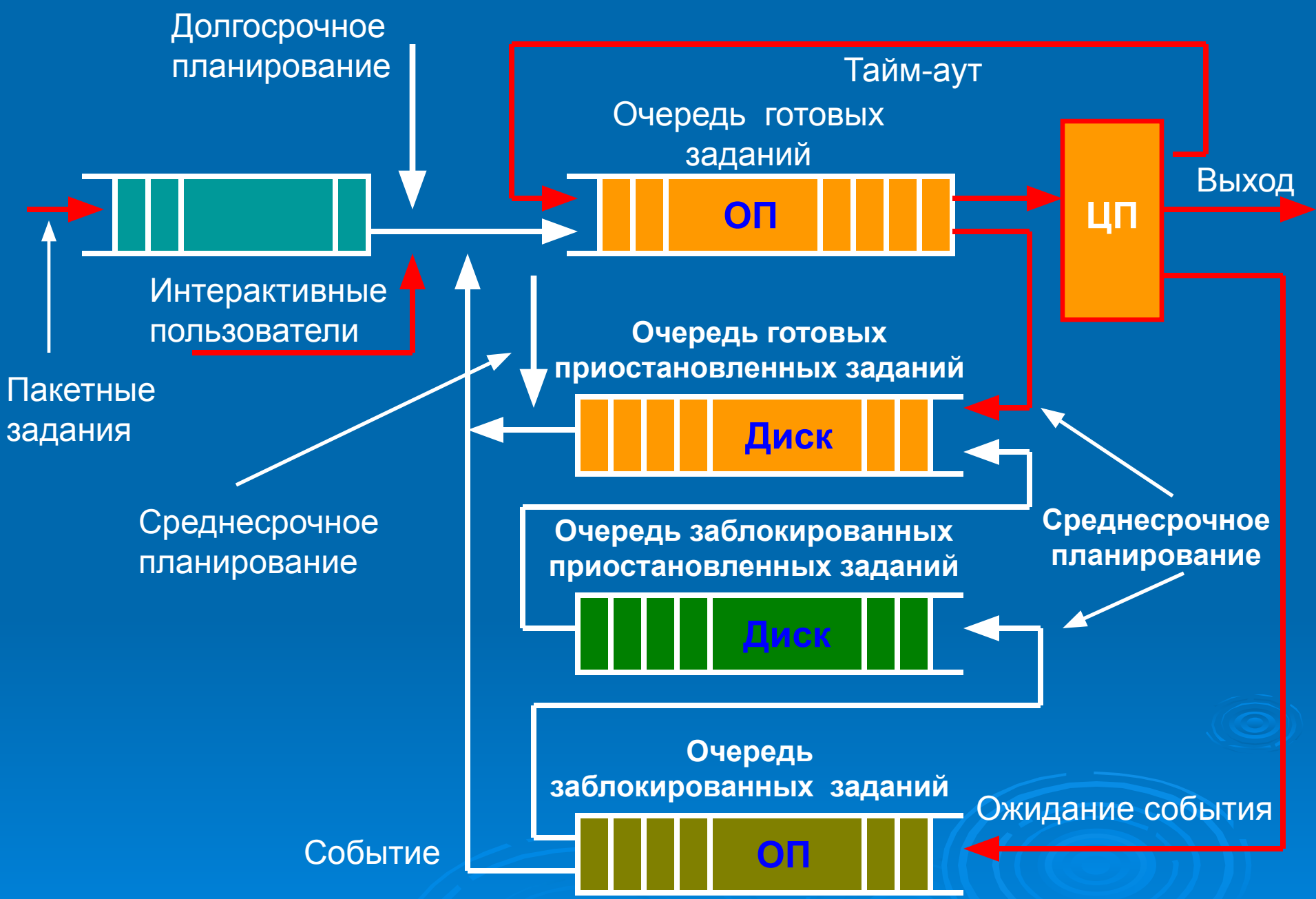
Необходимость двукратного переключения режима пользователь – ядро, ядро – пользователь для передачи управления от одного потока к другому в рамках одного и того же процесса.

2.5. Планирование заданий, процессов и потоков

1. Виды планирования

Вид планирования	Выполняемые функции
Долгосрочное	Решение о добавлении задания (процесса) в пул выполняемых в системе
Среднесрочное	Решение о добавлении процесса к числу процессов полностью или частично размещенных в основной памяти
Краткосрочное	Решение о том, какой из доступных процессов (потоков) будет выполняться процессором
Планирование ввода-вывода	Решение о том, какой из запросов процессов (потоков) на операцию ввода-вывода будет выполняться свободным устройством ввода-вывода

Схема планирования с учетом очередей заданий (процессов)


Долгосрочное планирование
Тайм-аут
Очередь готовых заданий
ОП
ЦП
Выход
Интерактивные пользователи
Пакетные задания
Очередь готовых приостановленных заданий
Диск
Среднесрочное планирование
Очередь заблокированных приостановленных заданий
Диск
Среднесрочное планирование
Очередь заблокированных заданий
ОП
Ожидание события
Событие

Типичный граф состояния потока

Алгоритмы планирования потоков

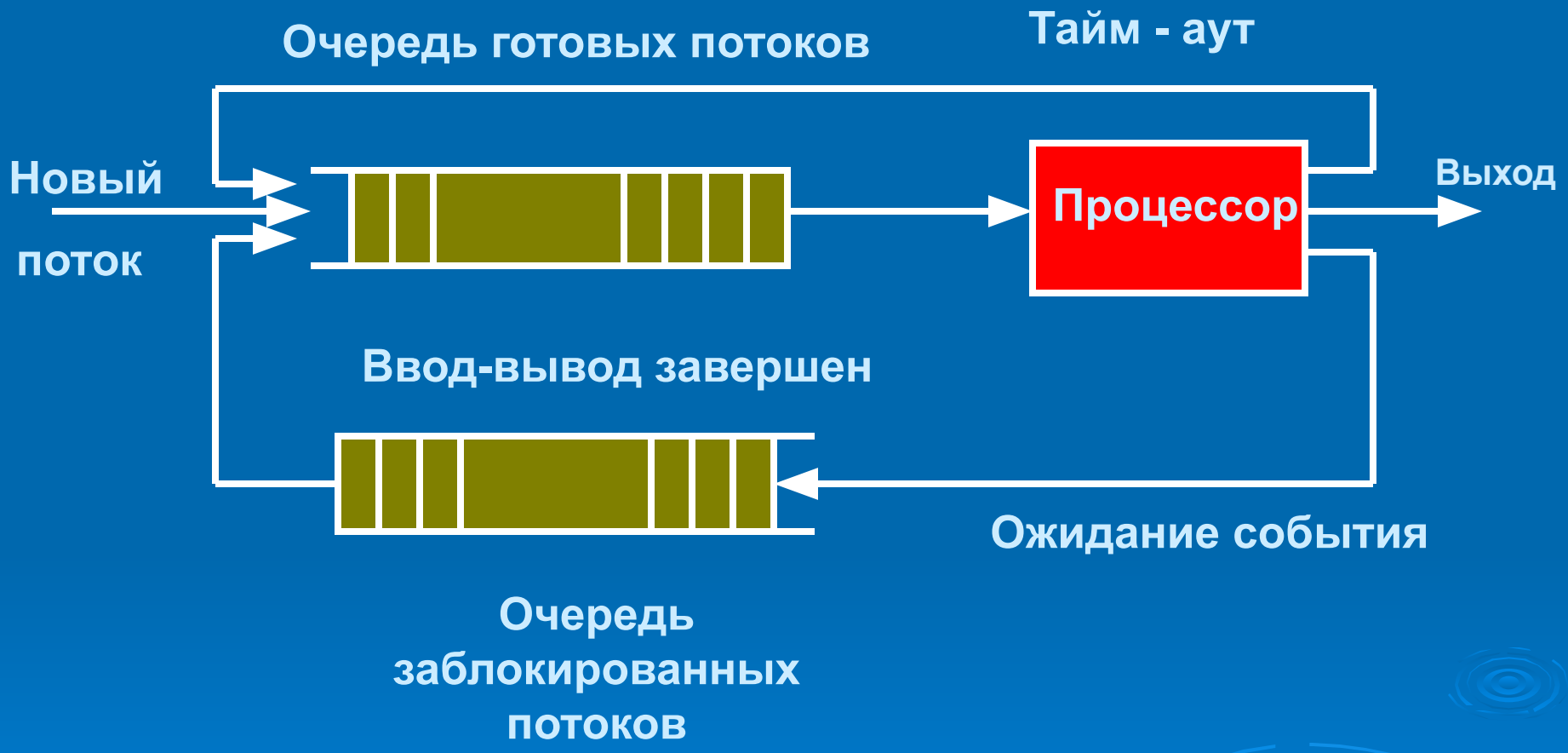
1. Невытесняющие (non-preemptive)

- планирование распределяется между ОС и прикладными программами;
- необходимость частых передач управлений ОС, в противном случае возможна монополизация процессора приложением;
- зависания приложений могут привести к краху системы

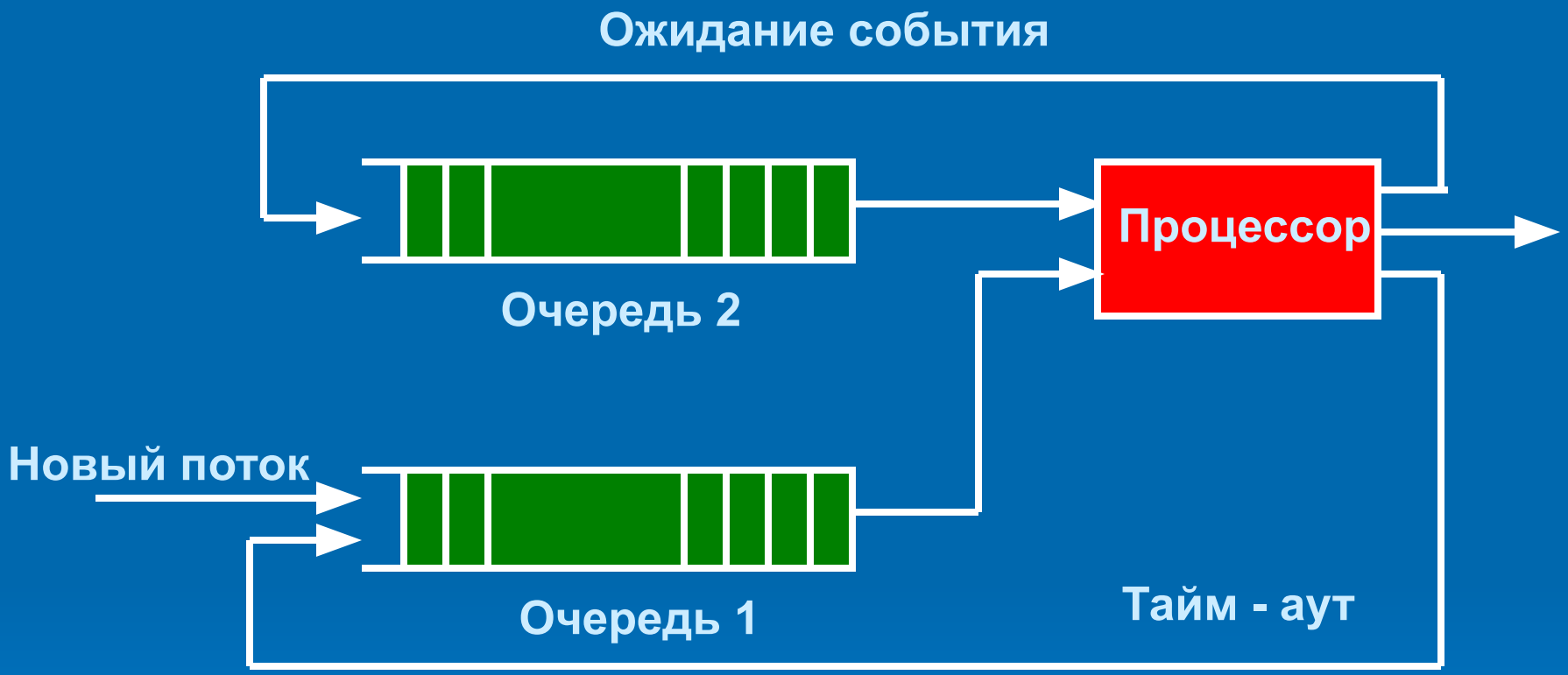
2. Вытесняющие (preemptive)

- функции планирования сосредоточены в ОС;
- планирование на основе квантования процессорного времени;
- планирование на основе приоритетов потоков: статических, динамических, абсолютных, относительных, смешанных;

Простейший алгоритм планирования, реализующий состояния потока по кадру 27

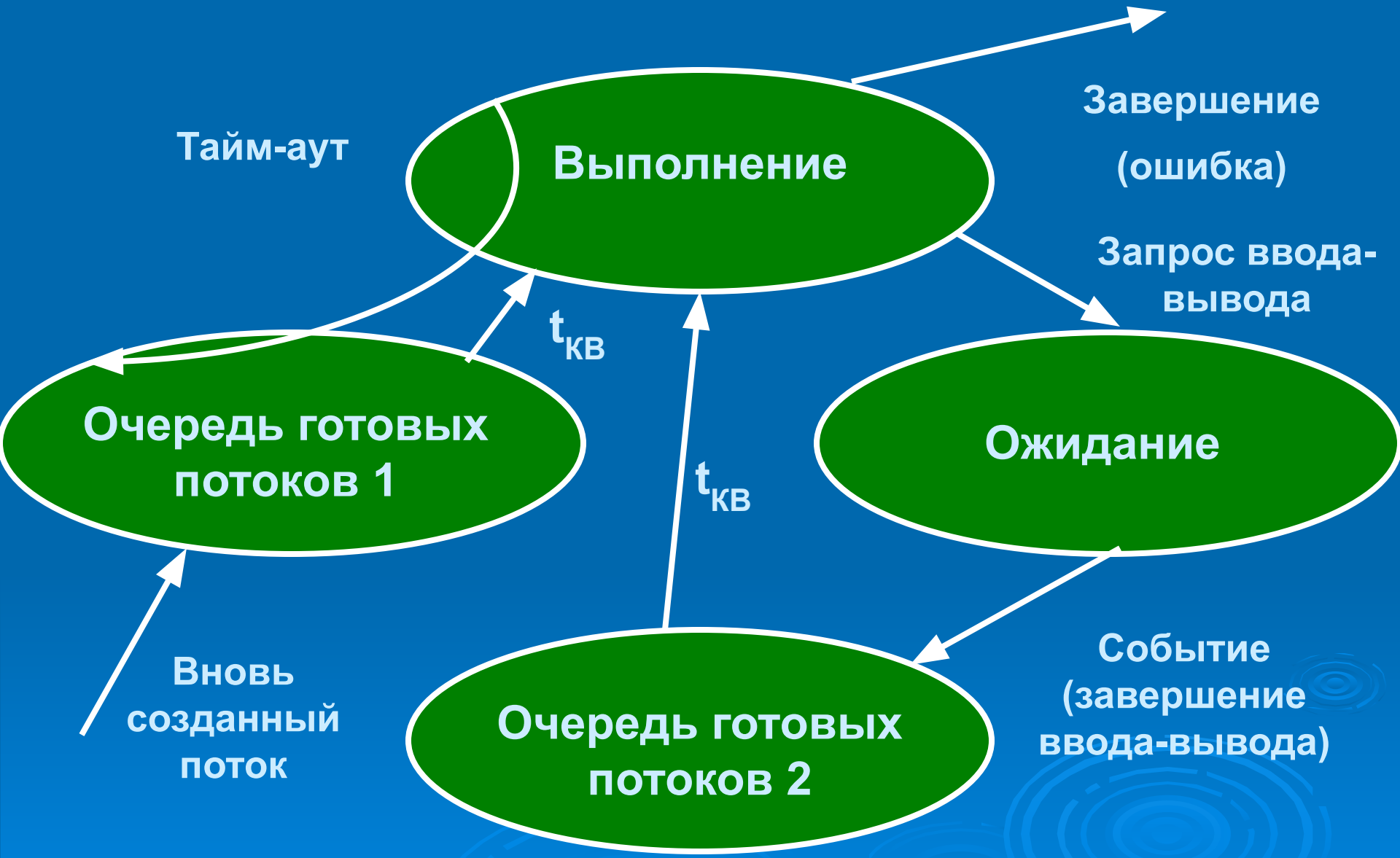


Алгоритм планирования, реализующий предпочтения потокам с интенсивным вводом-выводом

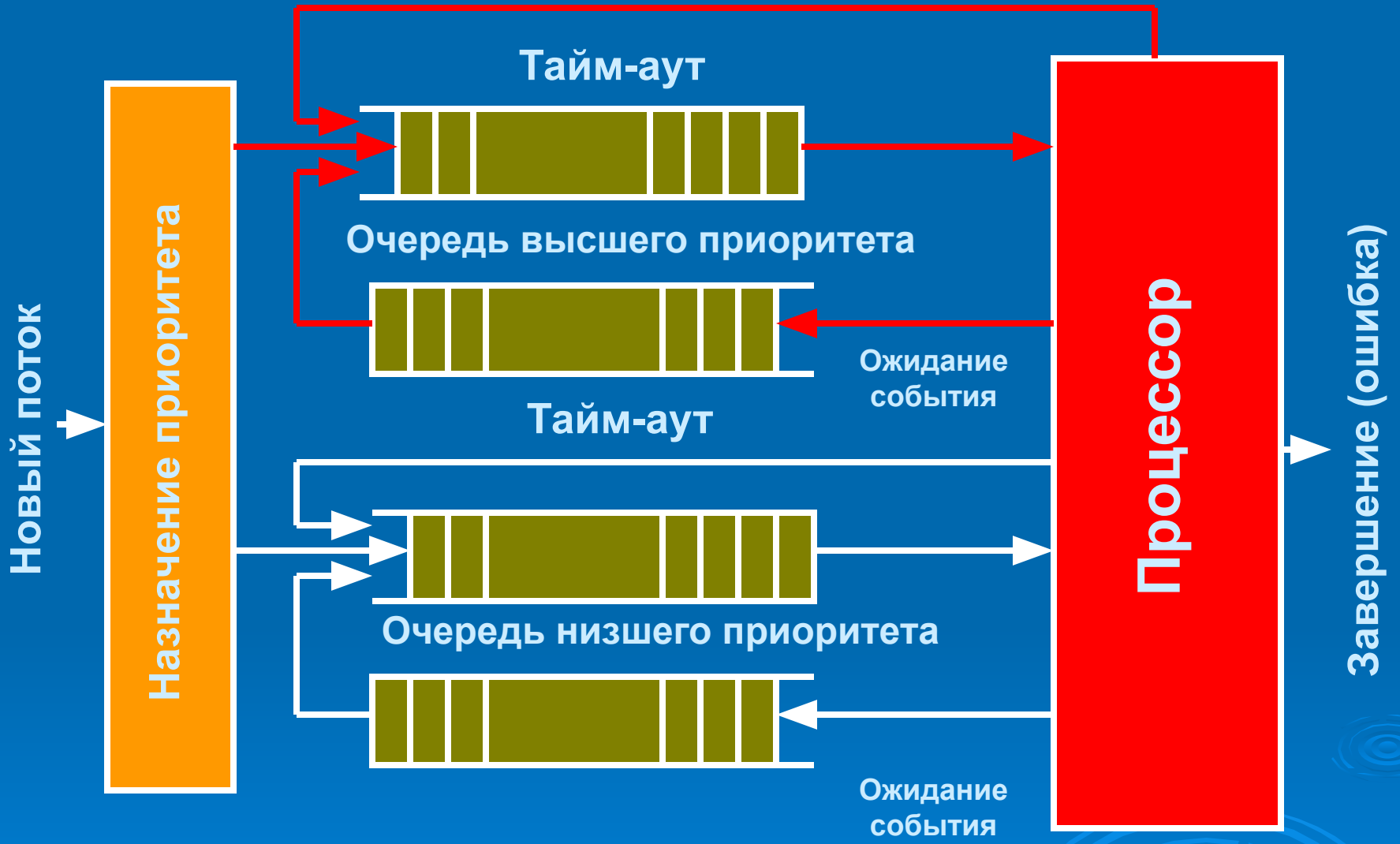


1. Переключение контекстов потоков связано с потерями процессорного времени.
2. С увеличением времени кванта ухудшается обслуживание пользователей.
3. В алгоритмах, основанных на квантовании, ОС не имеет никаких сведений о характеристиках решаемых задач.

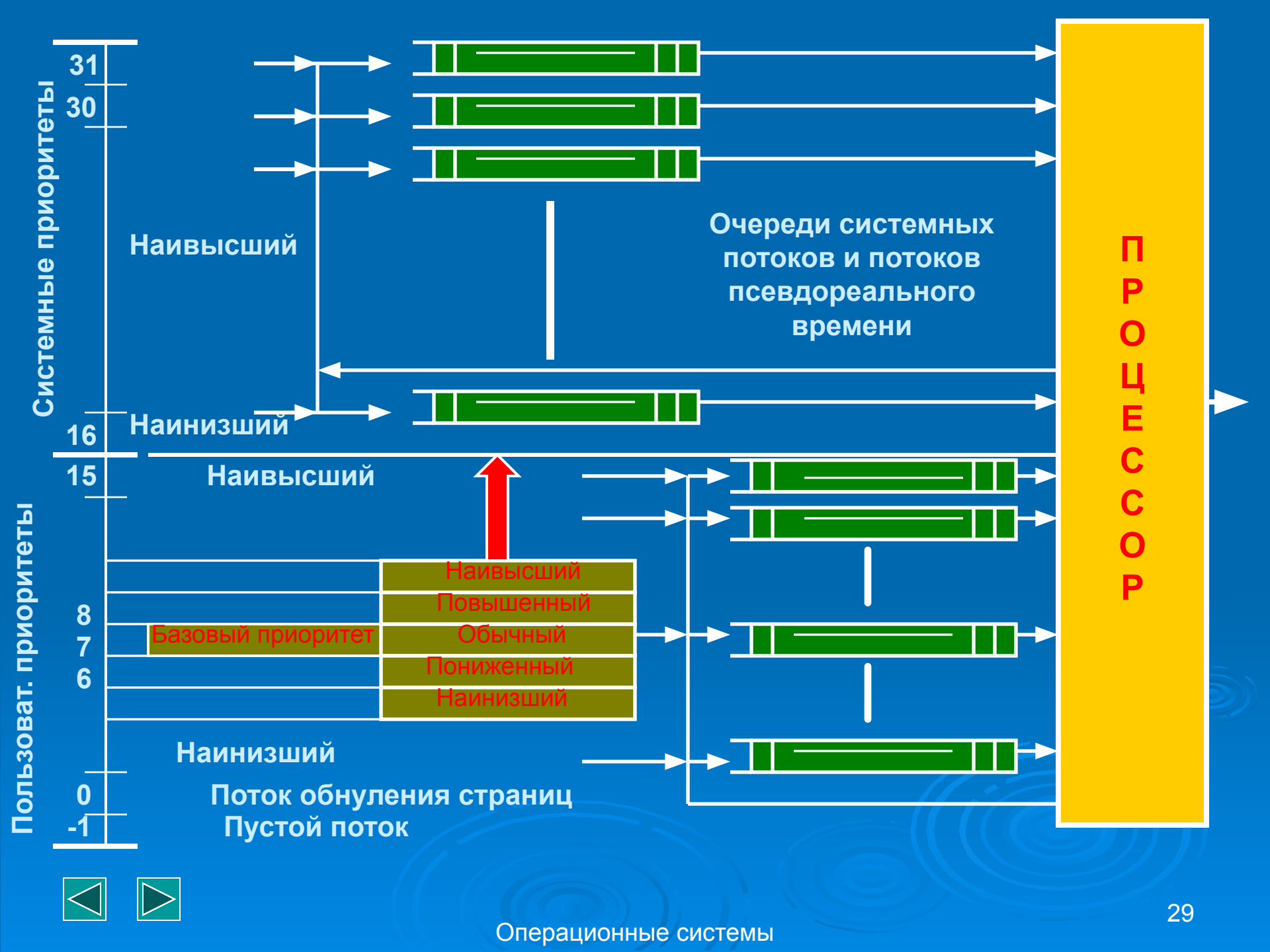
Граф состояния потока



Алгоритмы приоритетного планирования



Приоритетное переключение с квантованием


Системные приоритеты
31
30
Наивысший
16
Наинизший
Очереди системных потоков и потоков псевдореального времени
ПРОЦЕССОР
Пользоват. приоритеты
15
Наивысший
8
7
6
Базовый приоритет
Наивысший
Повышенный
Обычный
Пониженный
Наинизший
Наинизший
0
-1
Поток обнуления страниц
Пустой поток

Изменение базового приоритета потока

Увеличение приоритета

+ 1 – завершение ввода-вывода по диску;
+ 2 – для последовательной линии;
+ 6 – клавиатура;
+ 8 – звуковая карта;
+ 2 – снимается блокировка по семафору (для потока переднего плана);
+ 1 - снимается блокировка по семафору (для потока непереднего плана);
приоритет 15 на 2 кванта процессора, если готовый к выполнению поток простаивает более некоторого директивного времени.

Уменьшение приоритета

- 1 – если полностью использован квант времени процессора (многократно, вплоть до базового приоритета).


Работоспособные процессы (потоки)
Выбор для выполнения
Инициализация (0)
Резервный (3)
Переключение
Готовый (1)
Вытеснение
Выполняющийся (2)
Снятие блокировки / возобновление. Ресурсов достаточно
Ресурсов достаточно
Блокировка / Приостановка
Завершение
Ожидание (5)
Транзит (6)
Завершенный (4)
Снятие блокировки. Ресурсов недостаточно
Неработоспособные процессы (потоки)

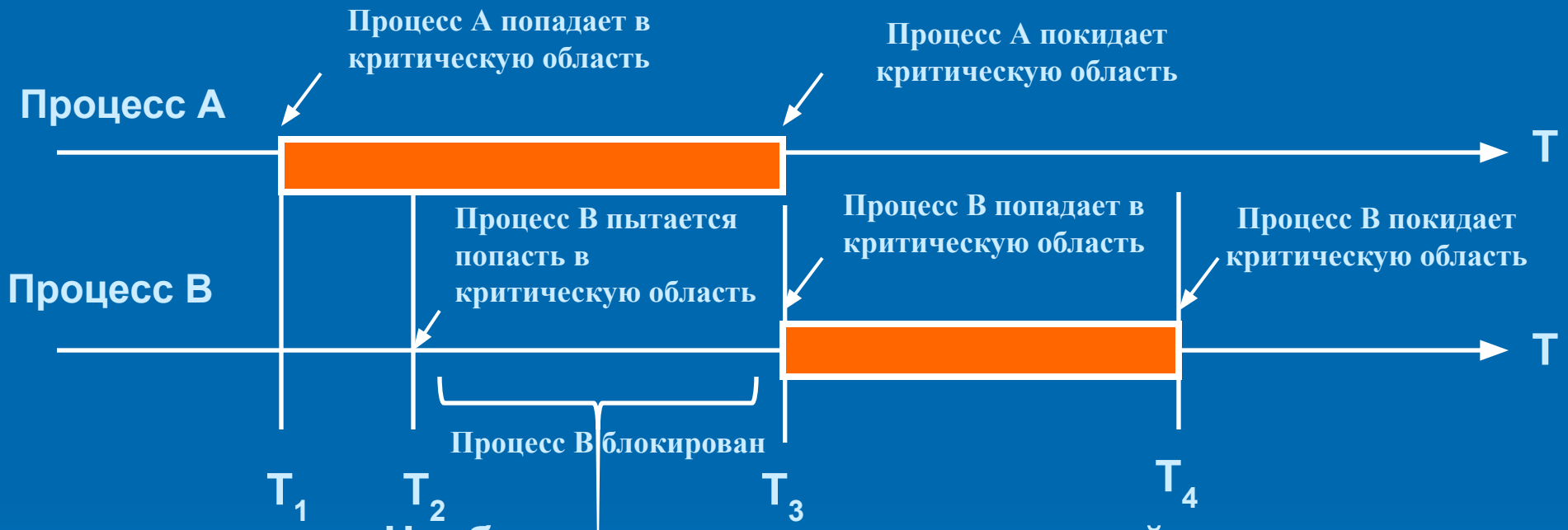
2.6. Взаимодействие и синхронизация процессов и потоков

2.6.1. Проблемы взаимодействия и синхронизации

Степень осведомленности	Взаимосвязь	Влияние одного процесса на другой	Потенциальные проблемы
Процессы не осведомлены друг о друге	Конкуренция	Результат работы одного процес- са не зависит от действий других. Возможно влияние одного про цесса на время работы другого.	Взаимоисключения Взаимоблокировки Голодание
Процессы косвенно осведомлены о наличии друг друга	Сотрудничество с использо- ванием разде- ления	Результат работы одного процесса может зависеть от информации, полученной от других. Возможно влияние одного процесса на время работы другого.	Взаимоисключения Взаимоблокировки Голодание Синхронизация
Процессы непосредственно осведомлены о наличии друг друга	Сотрудничество с использованием связи	Результат работы одного процесса зависит от информации, полученной от других процессов. Возможно влияние одного процесса на время работы другого.	Взаимоблокировки (расходуемые ресурсы) Голодание

2.6.2. Конкуренция процессов в борьбе за ресурсы

Конкуренция – ситуация, когда два или более процессов требуют доступ к одному и тому же ресурсу (принтеру, файлу и т.п.), называемому критическим. Часть программы, использующая критический ресурс, называется критической секцией.



Необходимость взаимоисключений:

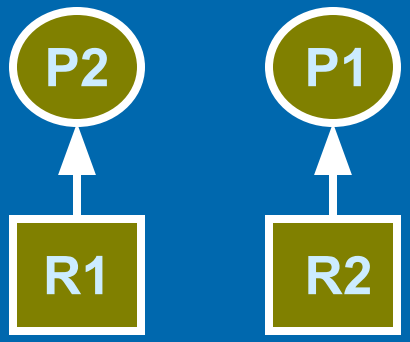
1. **Процессы не должны одновременно находиться в критических областях.**
2. **В программе не должно быть предположений о скорости или количестве процессов.**
3. **Процесс, находящийся вне критической области, не может блокировать другие процессы.**
4. **Невозможна ситуация, в которой процесс вечно ждет попадания в критическую область.**

Взаимоблокировки (тупики, deadlock)

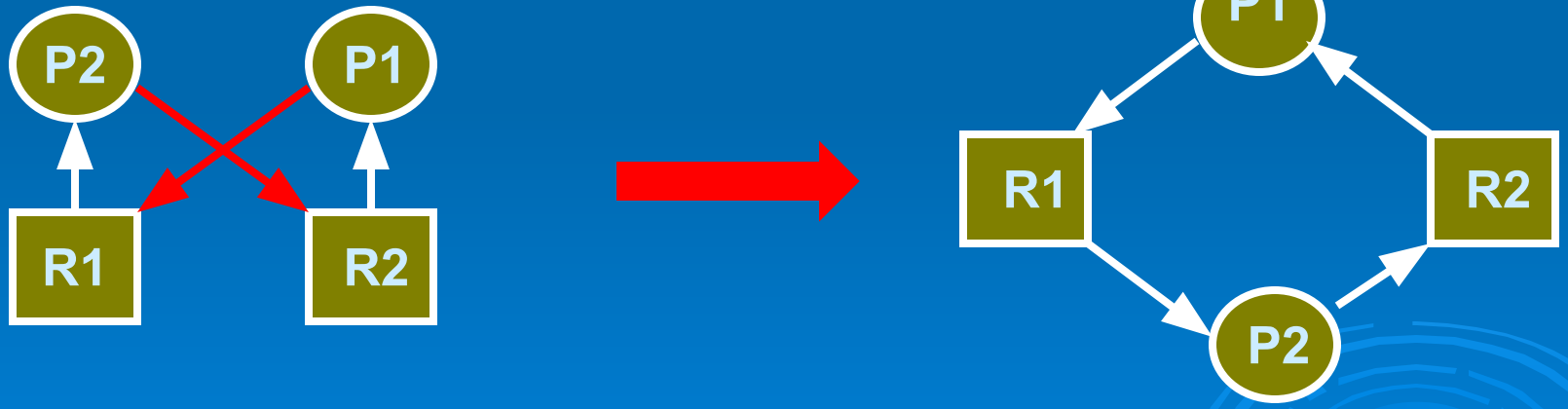
Группа процессов находится в тупиковой ситуации, если каждый процесс из группы ожидает события, которое может вызвать только другой процесс из этой же группы

Процесс

Ресурс

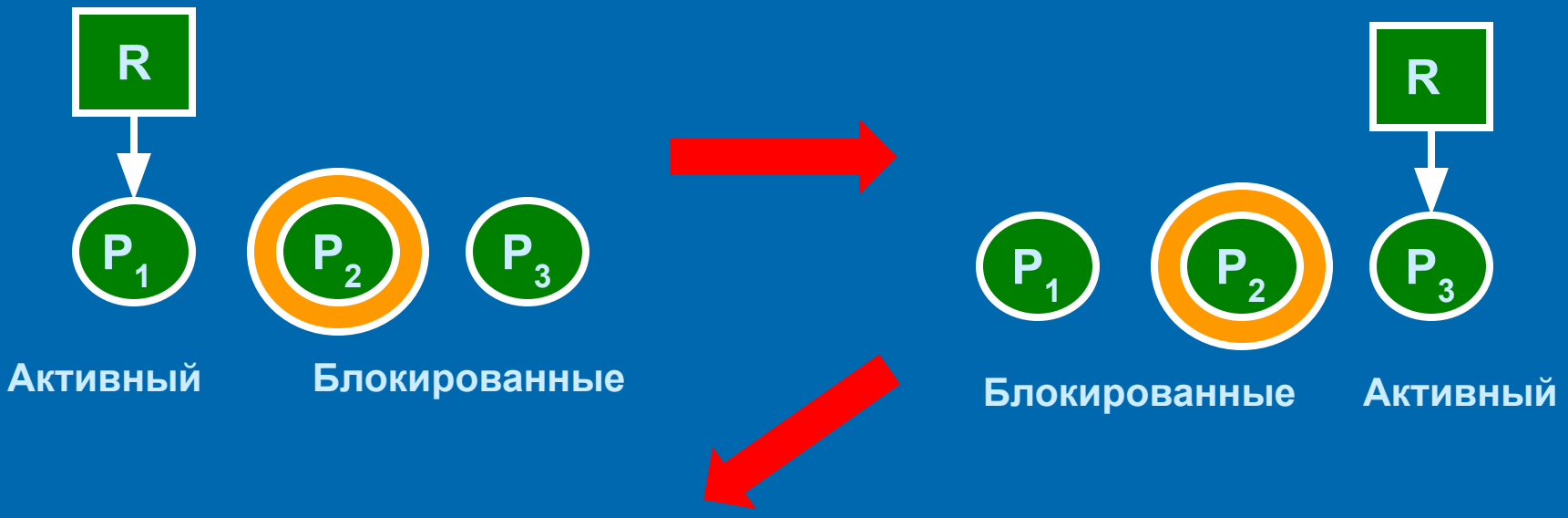


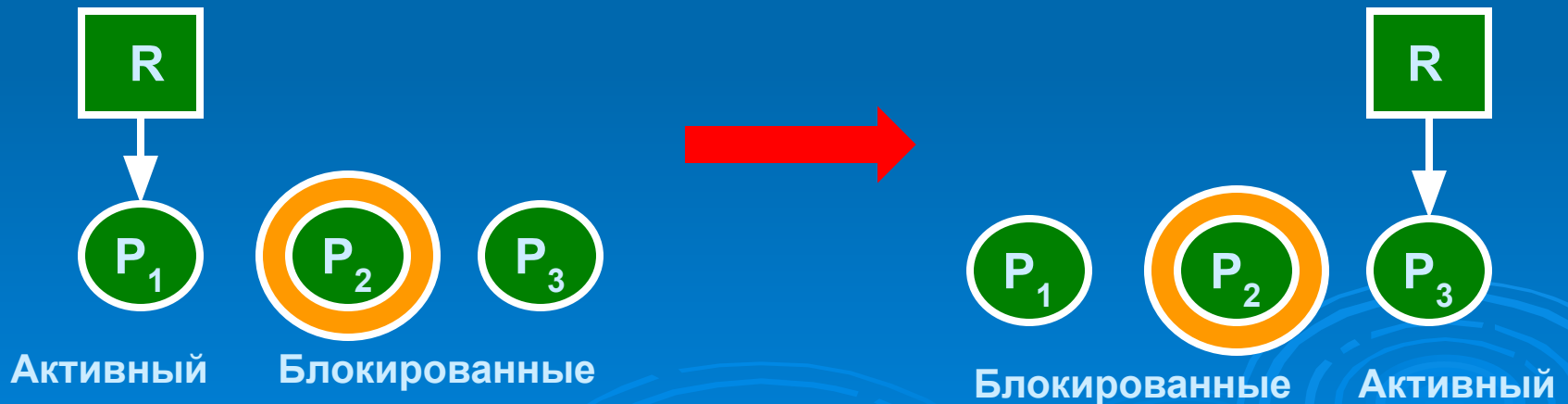
Исходное распределение ресурсов



Тупиковая ситуация

Проблема “голодание”





2.6.3. Сотрудничество с использованием разделения

Процессы, взаимодействующие с другими процессами без наличия явной информации о друг друге, обращаются к разделяемым переменным, к совместно используемым файлам или базам данных.

Проблемы: взаимоисключение, взаимоблокировка, голодание.
Дополнительно: синхронизация процессов для обеспечения согласованности данных

Пример: пусть должно выполняться a = b при начальном значении a = b = 1

1-й вариант: процессы выполняются последовательно

P1: a = a + 1; b = b + 1; P2: b = 2 * b; a = 2 * a;

2-й вариант: процессы прерывают друг друга

P1: a = a + 1; прерывание; P2: b = 2 * b; прерывание;

P1: b = b + 1; прерывание; P2: a = 2 * a;

Согласование нарушено: a = 4, b = 3

Ситуации, в которых два или более процессов обрабатывают разделяемые данные (файлы) и конечный результат зависит от скоростей процессов (потоков), называются гонками.

2.6.4. Методы взаимоисключений

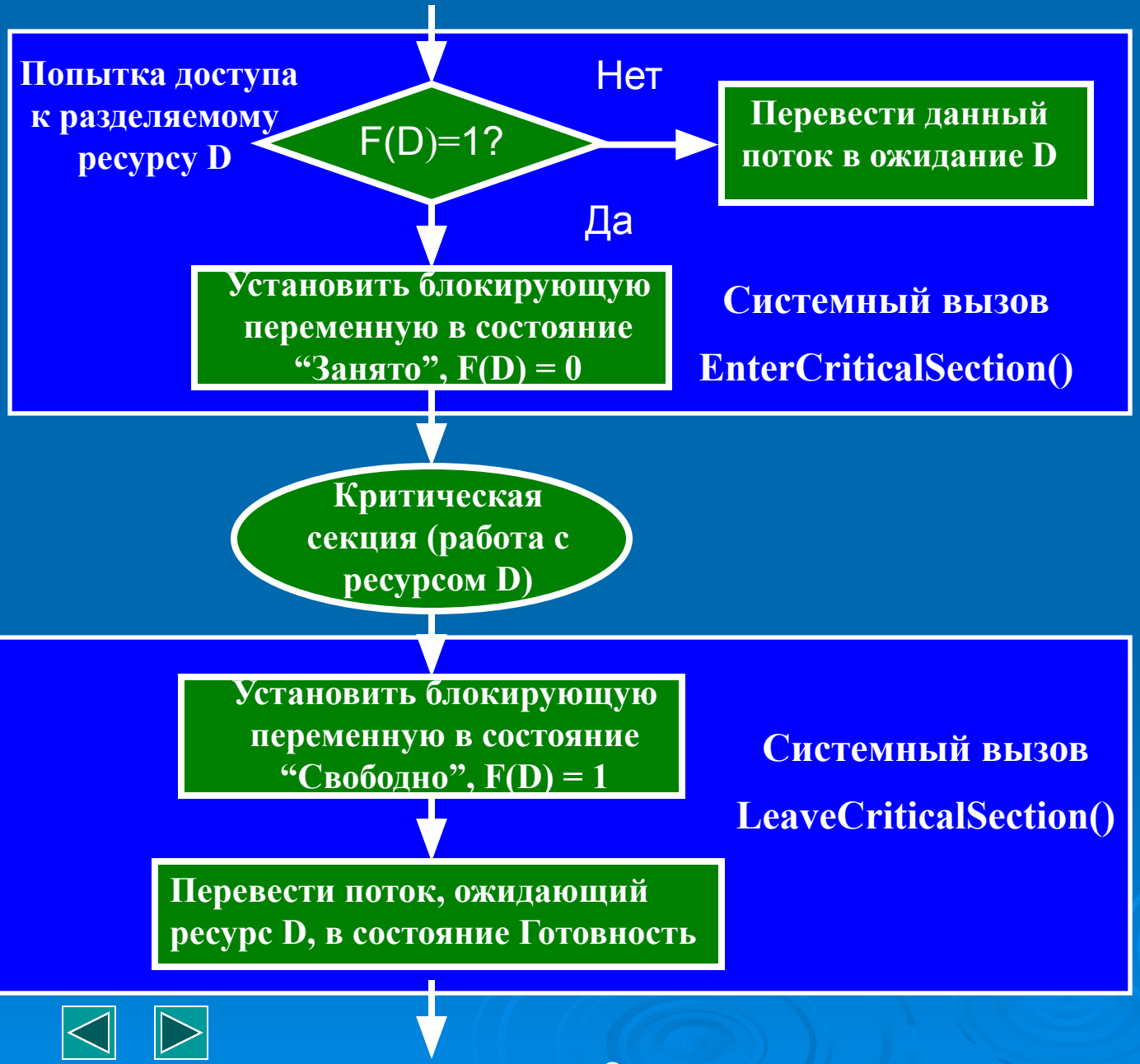
1. **Запрещение прерываний при входе в критическую область** и разрешение прерываний после выхода из критической области. **Достоинства**: простота реализации. **Недостатки**: монополизация процессора, возможный крах ОС при сбое процесса, невозможность использования в многопроцессорных системах.
2. **Блокирующие переменные (программный подход)**

Неделимая операция “проверка-установка”

Команды TC, BTR, BTS процессора Pentium (анализ и присвоение значения логической переменной)

Недостатки: необходимость постоянного опроса другими потоками, требующими тот же ресурс, блокирующей переменной; дополнительные затраты процессорного времени.

3.Использование системных функций входа в критическую секцию



Достоинство: исключается потеря времени процессора на циклическую проверку освобождения занятого ресурса.

Недостаток: растут накладные расходы ОС на по реализации функции входа в критическую секцию и выхода из нее

4. Семафоры Дийкстры (Dijkstra)

Семафор: переменная S, примитивы P (proberen – проверка; down) и V (verhogen – увеличение, up)

V(S) – переменная S увеличивается на 1 единым действием. Выборка, наращивание и запоминание не могут быть прерваны. К переменной S нет доступа во время выполнения этой операции.

P(S) – переменная S уменьшается на 1, если это возможно, составясь в области неотрицательных значений. Если S уменьшить невозможно, поток, выполняющий операцию P, ждет, пока это уменьшение станет возможным. Операция P неделима.

В частном случае семафор S может принимать двоичные значения 0 и 1, превращаясь в блокирующую переменную (двоичный семафор).

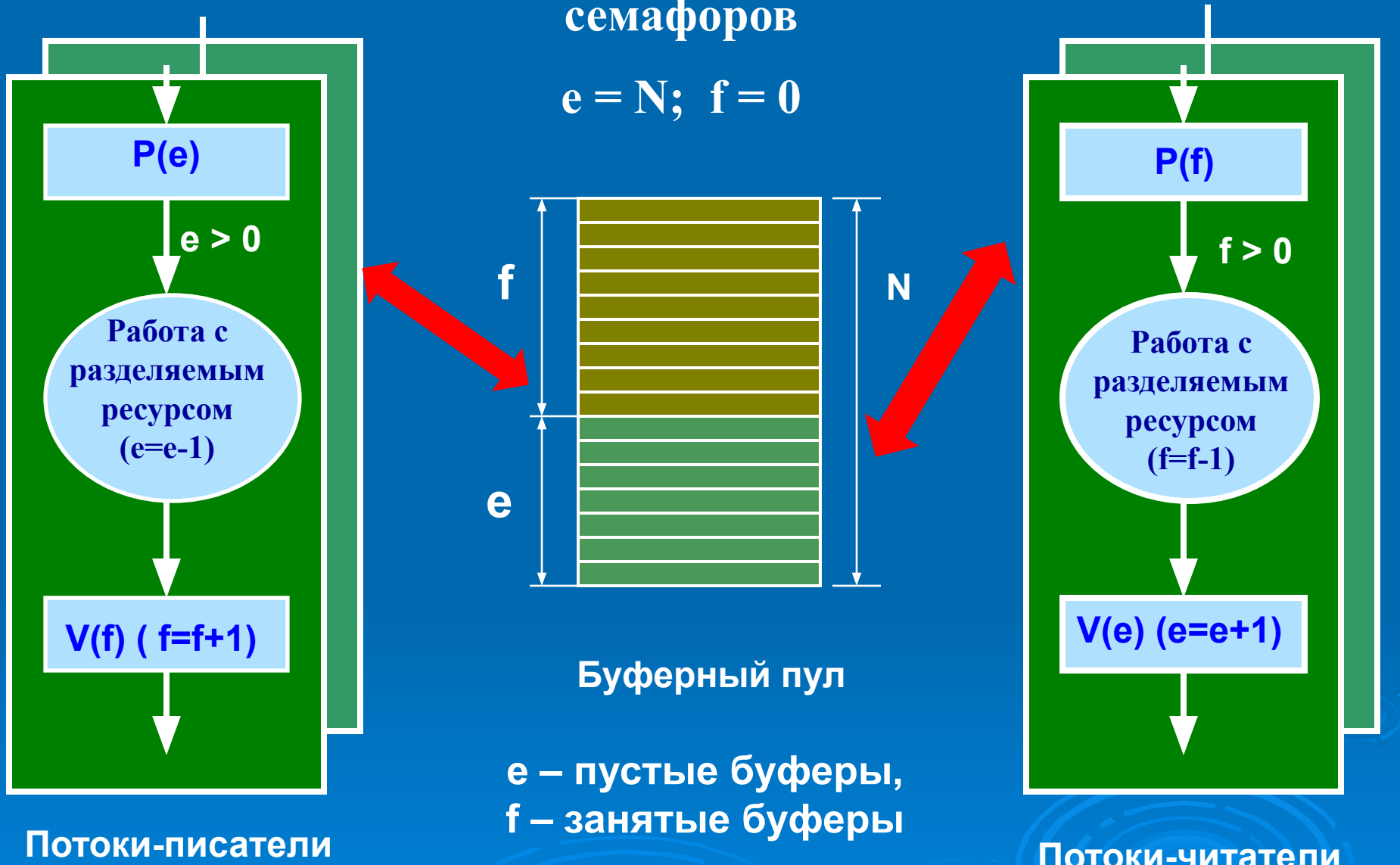
Операция P заключает в себе потенциальную возможность перехода процесса, который ее выполняет, в состояние ожидания (если S = 0).

Операция V может при некоторых обстоятельствах активизировать процесс, приостановленный операцией P.

Для хранения процессов, ожидающих семафоры, используется очередь, работающая по принципу FIFO.

Начальные значения семафоров

$e = N;\ f = 0$



Потоки-писатели

Буферный пул

e – пустые буферы,
f – занятые буферы

Потоки-читатели

2.6.5. Взаимоблокировки (тупики)

Условия возникновения взаимоблокировки (тупиковой) ситуации:

1. Взаимное исключение. Каждый ресурс в данный момент или отдан ровно одному процессу, или недоступен.
2. Условие удержания и ожидания. Процессы, в данный момент удерживающие полученные ранее ресурсы, могут запрашивать новые ресурсы.
3. Отсутствие принудительной выгрузки ресурсов. У процесса нельзя забрать принудительно ранее полученные ресурсы.
4. Условие циклического ожидания. Существует круговая последовательность из двух и более процессов, каждый из которых ждет доступа к ресурсу, удерживаемому следующим членом последовательности.

Стратегии борьбы с взаимоблокировками:

1. Пренебрежение проблемой в целом.
2. Обнаружение и устранение взаимоблокировок (восстановление).
3. Недопущение тупиковых ситуаций с помощью аккуратного распределения ресурсов.
4. Предотвращать с помощью структурного опровержения одного из четырех условий, необходимых для взаимоблокировки

Методы обнаружения взаимоблокировок

1. **В системе один ресурс каждого типа.**

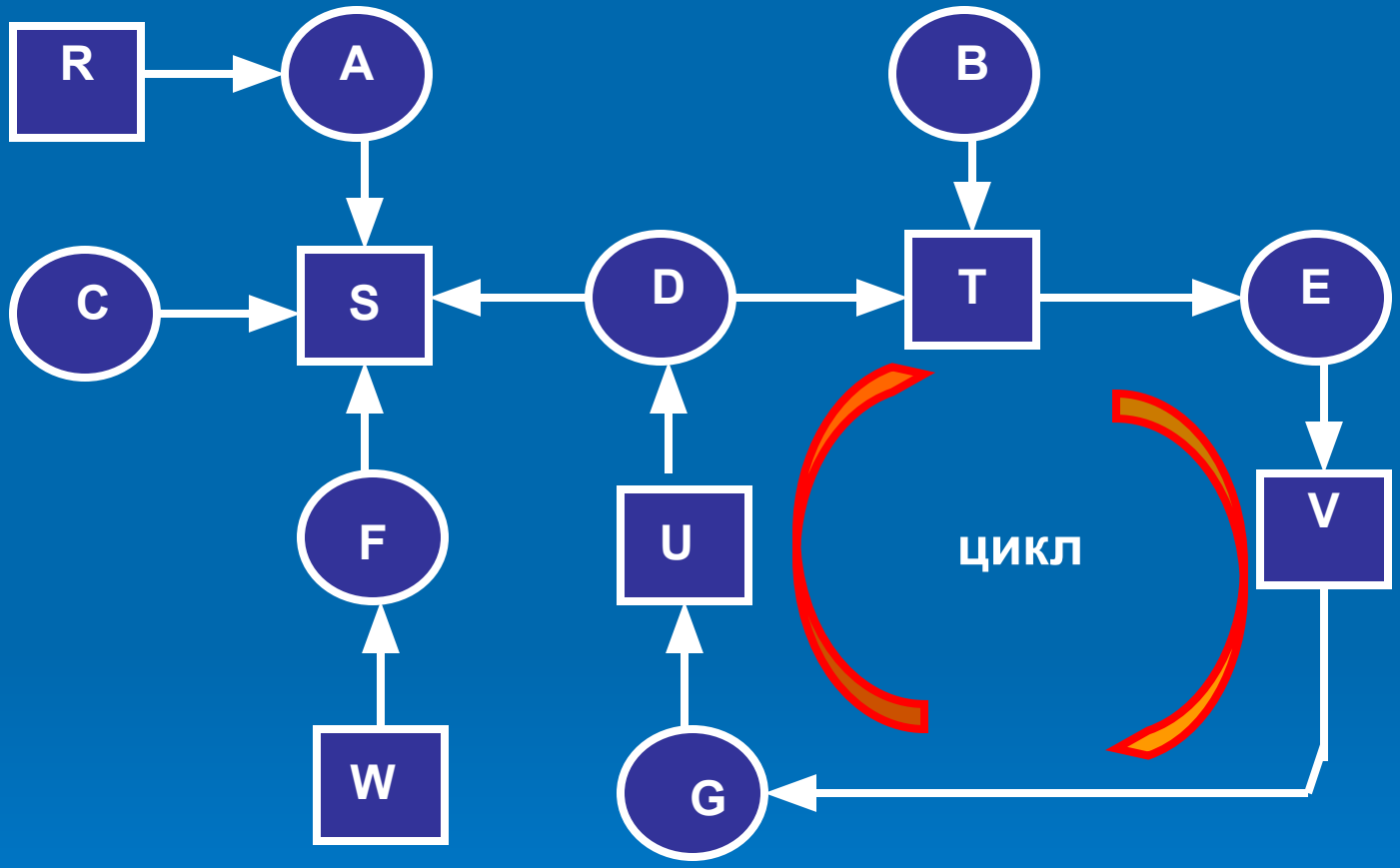
Например, пусть система из семи процессов (A, B, C, D, E, F, G) и шести ресурсов (R, S, T, V, W, U) в некоторый момент соответствует следующему списку:

- **Процесс A занимает ресурс R и хочет получить ресурс S.**
- **Процесс B ничего не использует, но хочет получить ресурс T.**
- **Процесс C ничего не использует, но хочет получить ресурс S.**
- **Процесс D занимает ресурс U и хочет получить ресурсы S и T.**
- **Процесс E занимает ресурс T и хочет получить ресурс V.**
- **Процесс F занимает ресурс W и хочет получить ресурс S.**
- **Процесс G занимает ресурс V и хочет получить ресурс U.**

ВОПРОС: Заблокирована ли эта система и если да, то какие процессы в этом участвуют?

ОТВЕТ МОЖНО ПОЛУЧИТЬ, ПОСТРОИВ ГРАФ РЕСУРСОВ И ПРОЦЕССОВ.

Граф ресурсов и процессов



2. В системе несколько ресурсов каждого типа.

$P = \{P_1, P_2, \ldots, P_n\}$ – множество процессов, n – число процессов;

$E = \{E_1, E_2, \ldots, E_m\}$ – множество ресурсов, m – число типов ресурсов;

$A = \{A_1, A_2, \ldots, A_m\}$ – вектор свободных ресурсов; $A_J <= E_J$, $j = \overline{1, m}$;

$C = \{c_{I\,J} \mid i = \overline{1, n};\ j = \overline{1, m}\}$ – матрица текущего распределения ресурсов;

$R = \{r_{i\,j} \mid i = \overline{1, n};\ j = \overline{1, m}\}$ – матрица запрашиваемых ресурсов.

Существующие ресурсы

$$E = \{E_1, E_2, \ldots, E_m\}$$

$$\begin{vmatrix} c_{11} & c_{12} & \ldots & c_{1m} \\ c_{21} & c_{21} & \ldots & c_{2m} \\ . & . & \ldots & . \\ c_{n1} & c_{n2} & & c_{nm} \end{vmatrix}$$

Доступные ресурсы

$$A = \{A_1, A_2, \ldots, A_m\}$$

$$\begin{vmatrix} r_{11} & r_{12} & \ldots & r_{1m} \\ r_{21} & r_{22} & \ldots & r_{2m} \\ . & . & \ldots & . \\ r_{n1} & r_{n2} & \ldots & R_{nm} \end{vmatrix}$$

$$\sum_{I=1}^{n} C_{i\,j} + A_j = E_j, \quad j = \overline{1, m}$$

Алгоритм обнаружения тупиков

Основан на сравнении векторов ресурсов. В исходном состоянии все процессы не маркированы (не отмечены). По мере реализации алгоритма на процессы будет ставиться отметка, обозначающая, что они могут закончить свою работу, т. е. не находятся в тупике. После завершения алгоритма любой немаркированный процесс находится в тупиковой ситуации.

Алгоритм

1. Ищется процесс P_i , для которого i – я строка матрицы R меньше вектора A, т. е. $R_i <= A_j$ или $r_{ij} <= A_j$, $j = \overline{1, m}$.
2. Если такой процесс найден, он маркируется, и далее прибавляется I - я строка матрицы C к вектору A, т.е. $A_j := A_j + c_{ij}$, $j = \overline{1, m}$.

   Возврат к шагу 1.
3. Если таких процессов не существует, работа алгоритма заканчивается. Если есть немаркированные процессы, то они попали в тупик.

Методы устранения тупиков

1. Принудительная выгрузка ресурсов. Изъятие ресурса у процесса, передача его другому процессу, а затем возврат ресурса таким образом, что исходный процесс этого “ не замечает” (сложно и чаще всего невозможно).
2. Восстановление через “откат”. Прцессы периодически создают контрольные точки, позволяющие запустить процесс с предыстории. При возникновении тупика процесс, занимающий необходимый ресурс “откатывается” к контрольной точке, после которой он получил ресурс. Если возобновленный процесс вновь попытается получить данный ресурс, он переводится в режим ожидания освобождения этого ресурса.
3. Восстановление путем уничтожения процессов.

Недопущение тупиков путем безопасного распределения ресурсов. Подобные алгоритмы базируются на концепции безопасных состояний. Например, Дейкстрой был разработан алгоритм планирования, позволяющий избегать взаимоблокировок (алгоритм банкира).

2.6.6. Синхронизирующие объекты ОС

Для синхронизации потоков, принадлежащих разным процессам, ОС должна предоставлять потокам системные объекты синхронизации.

К таким объектам относятся *события* (event), *мьютексы* (mutex – mutual exclusion – взаимное исключение), *системные семафоры* и др.

Объект-событие используется для того, чтобы оповестить потоки о том, что некоторые действия завершены.

Мьютекс (простейший двоичный семафор) используется для управления доступом к данным.

Семафоры используются для оповещения свершения последовательности событий.

Для синхронизации используются также “обычные ” объекты ОС: файлы, процессы, потоки

Все объекты синхронизации могут находиться в сигнальном и несигнальном (свободном) состоянии. Поток с помощью системного вызова WAIT(X) может синхронизировать свое выполнение с объектом синхронизации X. С помощью системного вызова SET(X) поток может перевести объект X в сигнальное состояние. Кроме того, в ОС определен набор сигналов для логической связи меду процессами, а также процессами и пользователями (терминалами).

2.7. Аппаратно-программные средства поддержки мультипрограммирования

2.7.1. Системы прерываний

Классы прерываний: внешние, внутренние, программные

1. Внешние прерывания – результат действий пользователя, сигналы от периферийных устройств компьютера и управляемых объектов.
2. Внутренние прерывания – результат появления аварийных ситуаций при выполнении инструкции программы.
3. Программные прерывания – результат выполнения запланированных в программе особых инструкций (системный вызов).

Принципы построения систем прерываний:

- аппаратная поддержка (контроллер прерываний, контроллер DMA, контроллеры внешних устройств, шины подключения внешних устройств, средства микропроцессора);
- векторный, опрашиваемый и комбинированный способы прерываний;
- приоритетный механизм обслуживания (с абсолютными и относительными приоритетами);
- маскирование прерываний;
- диспетчер прерываний и процедуры обслуживания прерываний.

Последовательность действий при обработке прерываний

1. Первичное аппаратное распознавание типа прерывание. Если прерывания запрещены, продолжается текущая программа. В противном случае вызывается диспетчер прерываний и в зависимости от поступившей в процессор информации (вектор прерывания, приоритет и др.) производится вызов процедуры обработки прерывания.

1. Сохраняется некоторая часть контекста прерванного потока, которая позволит возобновить его исполнение после обработки прерывания (обычно слово состояния процессора – регистр EFLAGS в Pentium, регистры общего назначения). Может быть сохранен и полный контекст, если ОС обслуживает прерывание со сменой процесса.
2. В счетчик команд загружается адрес процедуры обработки прерывания и устанавливается новое PSW, которое определяет привилегированный режим работы процессора при обработке прерывания.

1. Маскированием прерываний временно запрещаются прерывания, чтобы не образовалась очередь вложенных друг в друга потоков одной и той же процедуры.

1. После обработки прерывания ядром операционной системы, прерванный контекст восстанавливается (частично аппаратно – PSW, содержимое счетчика команд, частично программно – извлечение данных из стека), снимается обработка прерываний данного типа и работа потока возобновляется с прерванного места.

2.7.2. Системные вызовы

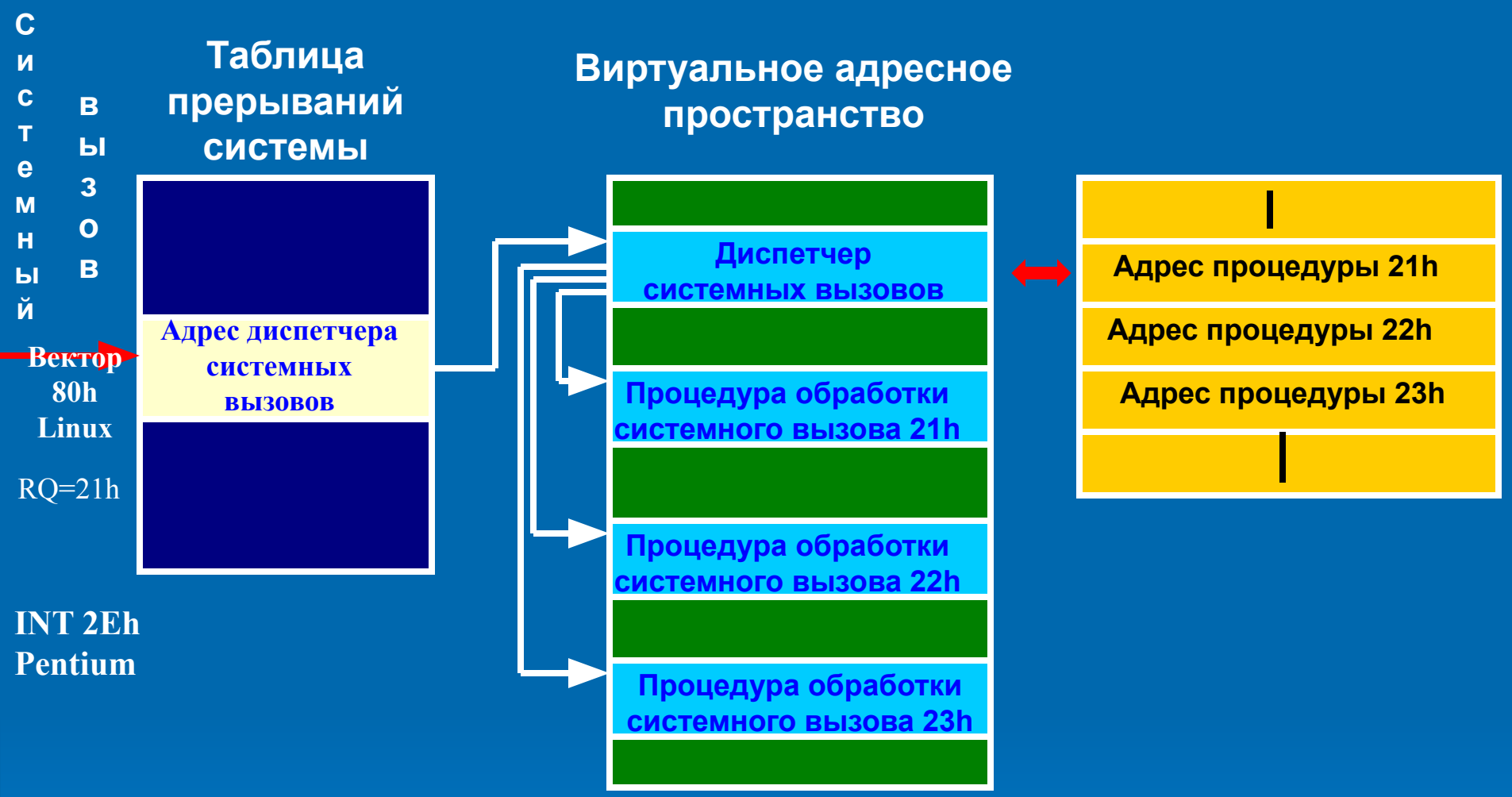
Системный вызов позволяет приложению обратиться к ОС с просьбой выполнить то или иное действие, оформленное как процедура кодового сегмента ОС.

Реализация системных вызовов должна удовлетворять следующим требованиям:

- обеспечивать переключение в привилегированный режим;
- обладать высокой скоростью вызова процедур ОС;
- обеспечивать по возможности единообразное обращение к системным вызовам для всех аппаратных платформ, на которых работает ОС;
- допускать простое расширение системных вызовов;
- обеспечивать контроль со стороны ОС за корректным использованием системных вызовов.

Возможные схемы обслуживания системных вызовов:

1. Децентрализованная –за каждым системным вызовом закреплен свой вектор прерываний. Достоинство – высокая скорость обработки системных вызовов, недостаток – разрастание таблицы векторов прерываний.
2. Централизованная – с помощью диспетчера системных вызовов.



Централизованная схема обработки системных вызовов