Операционные Системы Реального Времени

Максим Петрович Червинский

email: Maxim.Tchervinsky@motorola.com

(8 академических часов)

План (1)

1. **Введение**
2. **Определение “реального времени”**
    - **2.1 Жесткое реальное время (hard)**
    - **2.2 Реальное время с допусками (soft)**
    - **2.3 Комбинированное реальное время (firm)**
    - **2.4 Классификация и примеры событий**
3. **История развития встроенных ОС**
    - **3.1 Временной циклический исполнитель (cyclic executive)**
    - **3.2 Система, управляемая прерываниями (interrupt-driven executive)**
    - **3.3 Приоритетный планировщик, управляемый событиями (event-driven priority-based scheduler)**
4. **Характеристики встроенных ОС**

План (2)

5. Базовые объекты

- **5.1 задачи**
- **5.2 обработчики прерываний**
- **5.3 ресурсы (семафоры)**
- **5.4 сообщения**
- **5.5 события (флаги, сигналы)**
- **5.6 таймеры и счетчики**

6. Планирование и Диспетчеризация

7. Типы планирования:

- **7.1 невытесняющее (non pre-empted)**
- **7.2 вытесняющее (pre-empted)**
- **7.3 круговое (round-robin)**
- **7.4 квантование времени (time-sliced)**
- **7.5 переключения по времени (time-triggered)**

8. Управление задачами

9. Ждущие задачи

План (3)

10. Обслуживание прерываний:

- **10.1 вложенные прерывания**
- **10.2 немедленное выполнение сервиса ОС**
- **10.3 задержанное выполнения сервисов ОС**
- **10.4 отложенное выполнение сервисов ОС**
- **10.5 ограничение сервисов ОС**
- **10.6 атомарные операции в ОС**

11. Разделяемые ресурсы (семафоры)

- **11.1 P/V семафоры и связанные с ними проблемы**
- **11.2 Протокол маскирования прерываний (IMP)**
- **11.3 Протокол наследования приоритетов (PIP)**
- **11.4 Протокол высшего приоритета (HLP)**

План (4)

12. Техника назначения приоритетов:

- **12.1 Последовательное увеличение приоритетов (RMA)**
- **12.2 Приоритетное планирование с учетом ближайших сроков (EDF)**

13. Приоритетное планирование с пороговым вытеснением (PTS)

14. Сетевая передача данных

- **14.1 Физический уровень и уровень доступа на примере CAN**
- **14.2 Управление сетью**

15. Примеры Операционных Систем:

- **15.1 OSEK OS**
- **15.2 Real-Time Linux**

Библиография

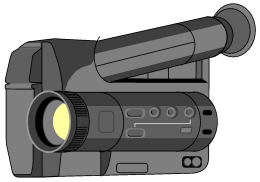
1. Real-Time Systems. Design Principles for Distributed Embedded Applications, by Hermann Kopetz, 1997, ISBN 0-7923-9894-7
2. A Practitioner's Handbook for Real-Time Analysis, by Mark H. Klein, Thomas Ralya, Bill Pollak, Ray Obenza, 1993, ISBN 0-7923-9361-9
3. Real-Time Systems, The International Journal of Time-Critical Computing Systems, ISSN 0922-6443
4. Программирование для вычислительных систем реального времени, Дж. Мартин, «Наука», 1975, УДК 519.95
5. Проектирование операционных систем для малых ЭВМ, С.Кейслер, «Мир», 1986, УДК 681.142.2
6. Разработка програмных средств для встроенных систем, Никифоров В. В., Учеб. пособие. СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2000, УДК 681.325.5-181.4, ISBN 5-7629-0340-0

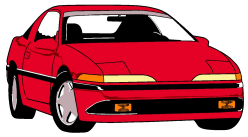
Ресурсы Интернет

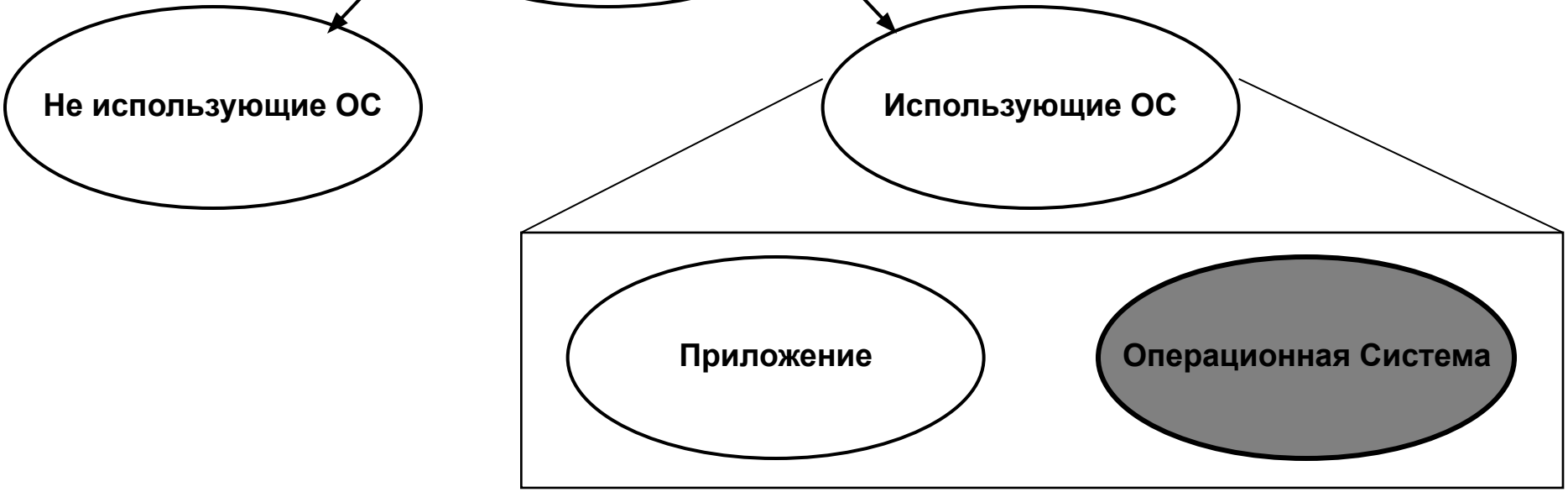
A. News: comp.realtime, comp.arch.embedded
B. http://www.embedded.com (Embedded System Programming, ESP)
C. http://www.dedicated-systems.com (including Dedicated Systems Magazine)
D. "Deadline Monotonic Analysis", Ken Tindell, ESP, vol. 13, #6, june 2000: http://www.embedded.com/2000/0006/0006feat1.htm
E. "Get by Without RTOS", Michael Melkonian, ESP, vol. 13, #10, september 2000: http://www.embedded.com/2000/0009/0009feat4.htm
F."Операционные системы реального времени", Жданов А.А., ЗАО "РТСофт", Москва, "PCWeek", N 8, 1999: http://www.rtsoft.ru/pressa/text027.html
G. "Full" list of RTOS: http://www.realtime-info.be/encyc/market/rtos/rtos.htm
H. OSEK Official Web site: www.osek-vdx.org

1. Введение (1)

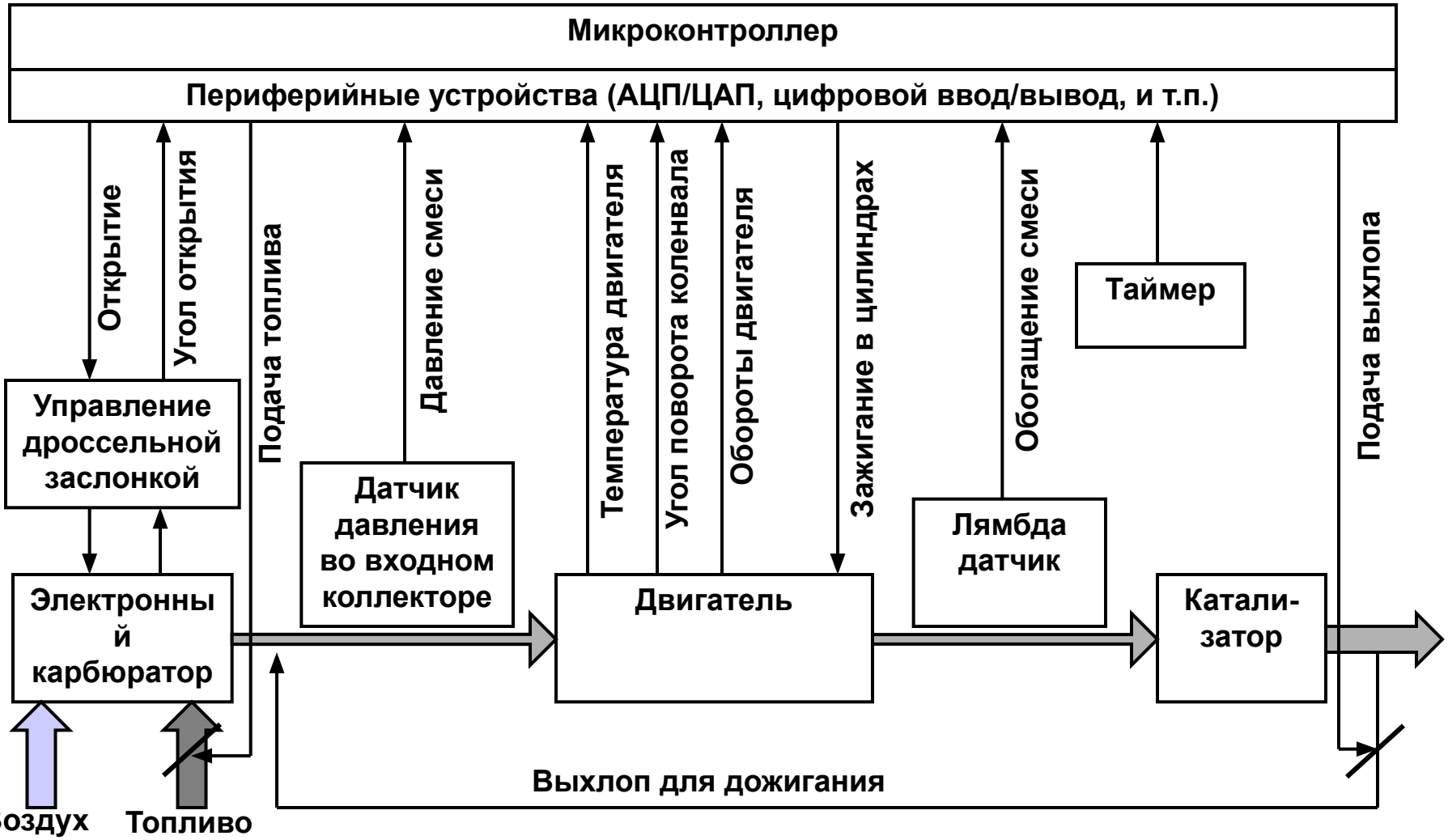
Встроенные системы (embedded systems) - программные системы, встраиваемые в оборудование (автомобили, бытовую технику, аудио и видео технику, станки, и т.п.)







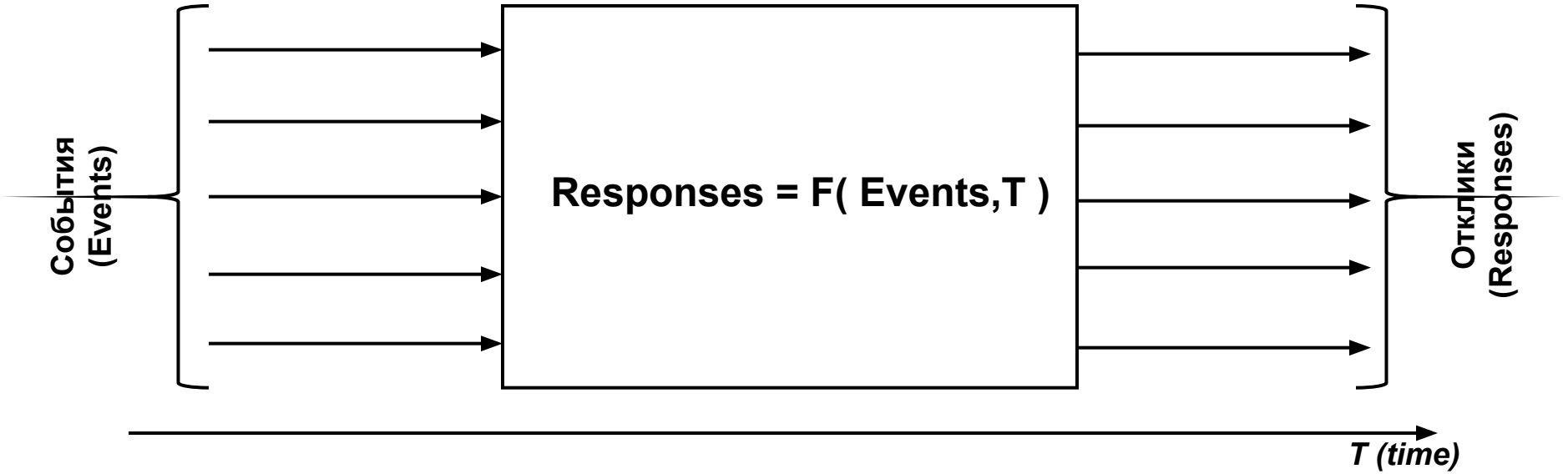
1. Введение (2)



Система управления двигателем обеспечивает наилучшее потребление топлива и оптимальную мощность двигателя при соблюдении требований по защите окружающей среды на всех режимах работы двигателя. Работает в режимах с обратной связью и без обратной связи.

2. Определение «реального времени» (1)

Система (приложение) реального времени - программная система, в которой корректность работы зависит не только от результатов вычислений, но также от времени получения этих результатов.

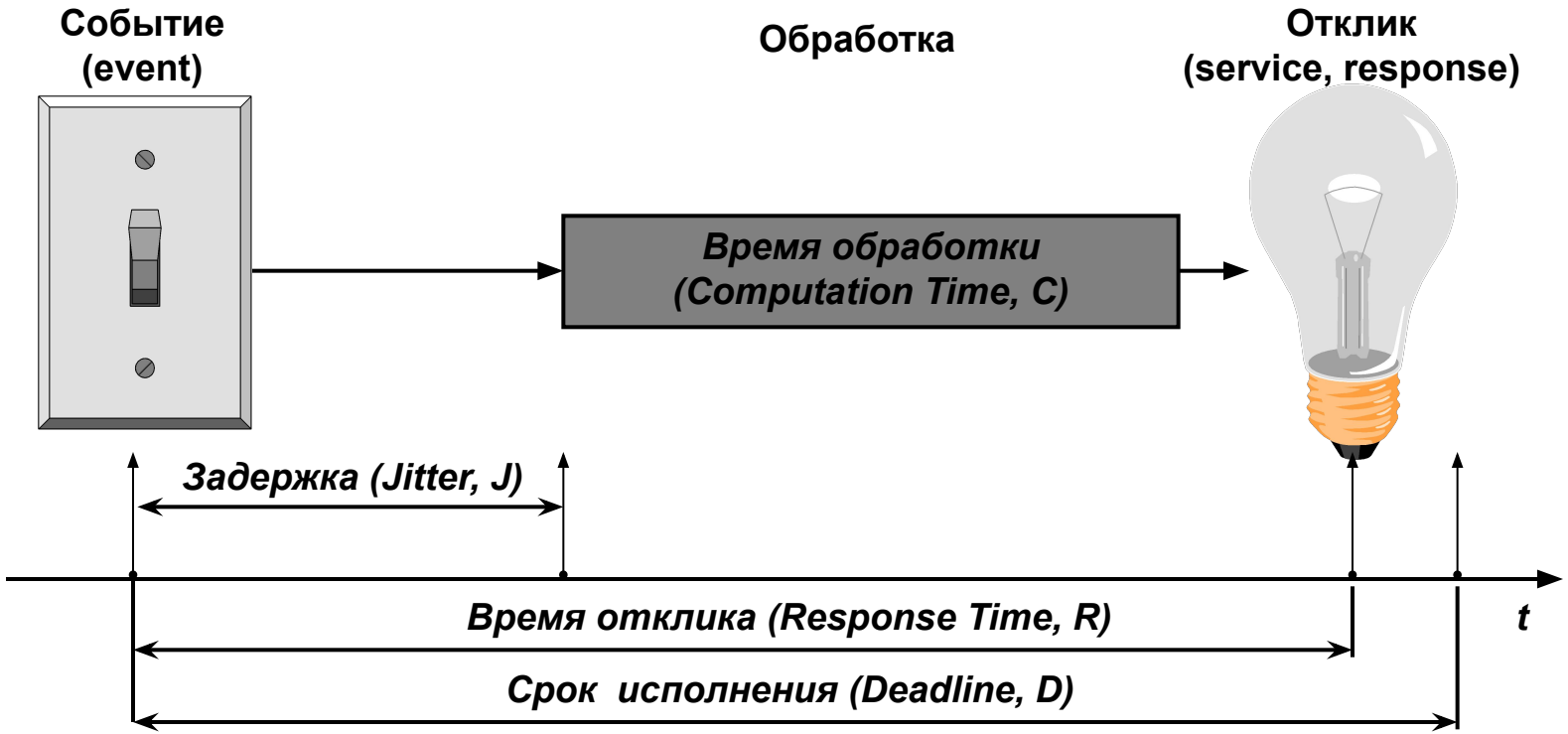


Система должна завершить обработку события (выработать отклик) не позднее заранее определенного момента времени.

Система управляет обработкой большого количества разных событий.

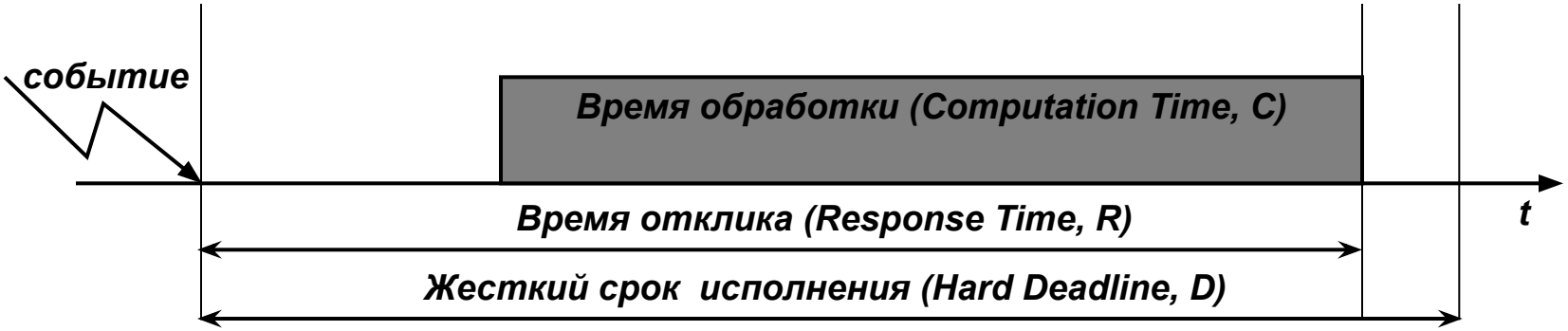
2. Определение «реального времени» (2)

- **Реальное время определяется соотношением срока исполнения и временем отклика.**
- **Реальное время не зависит от того, “быстрая” система или “медленная” (то есть не зависит от единиц измерения времени).**



Обработка “в реальном времени” означает “вовремя”

2.1 Жесткое реальное время



Жесткое реальное время (hard real time) требует, чтобы время отклика никогда не превышало срок исполнения (т.е. R меньше либо равно D).

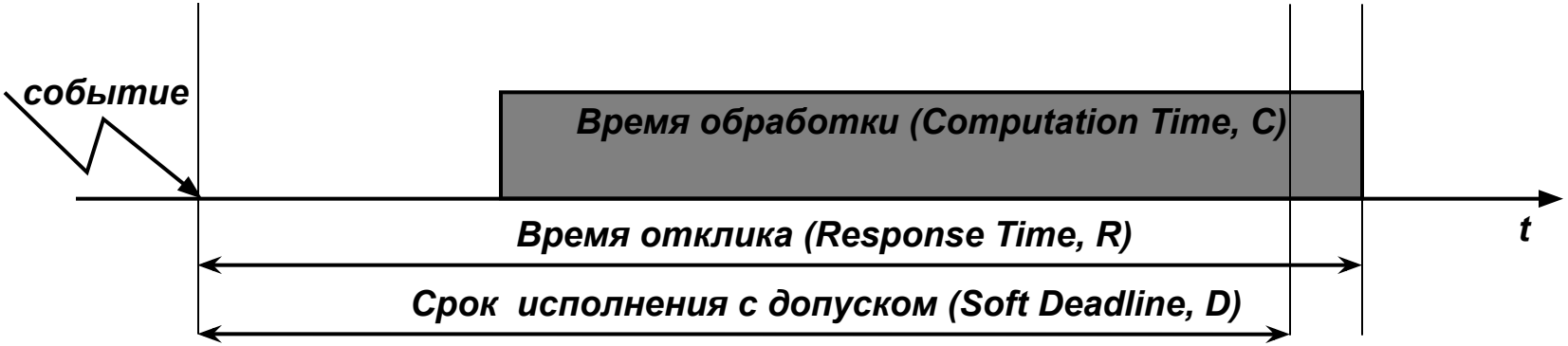
В случае, если срок исполнения истекает, а отклик не был выработан, происходит фатальный отказ системы.

Примеры:

- **система управления двигателем**
- **система торможения**
- **подушка безопасности**

Требуется в большинстве встроенных приложений!

2.2 Реальное время с допусками



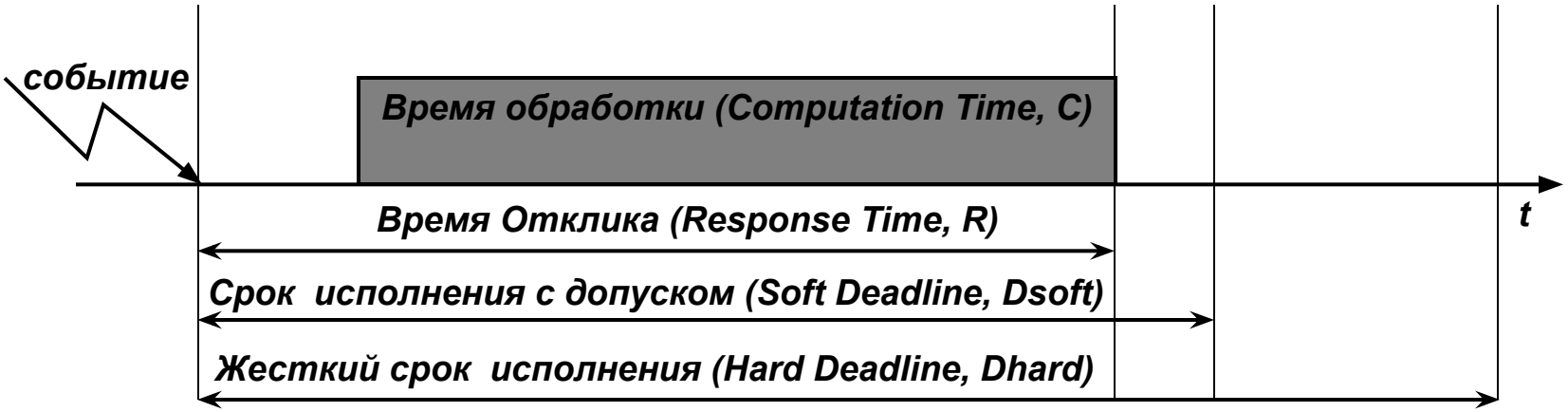
Реальное время с допусками (soft real time) допускает флуктуации времени отклика при условии, что среднее время отклика равно сроку исполнения (т.е. R в среднем равно D).

Система работает хуже (деградирует), но сохраняет работоспособность даже если срок исполнения иногда просрочен.

Примеры:

- **экранный редактор**
- **сеть передачи данных**
- **сервер базы данных**

2.3 Комбинированное реальное время

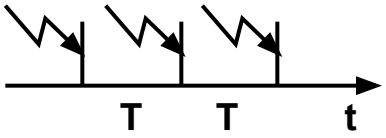
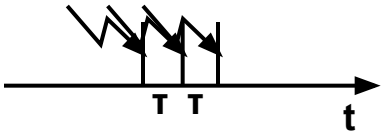


Комбинированное реальное время (firm real time) комбинирует два срока выполнения - короткого «с допуском» и более длинного «жесткого» (т.е. R в среднем равно Dsoft, но меньше либо равно Dhard).

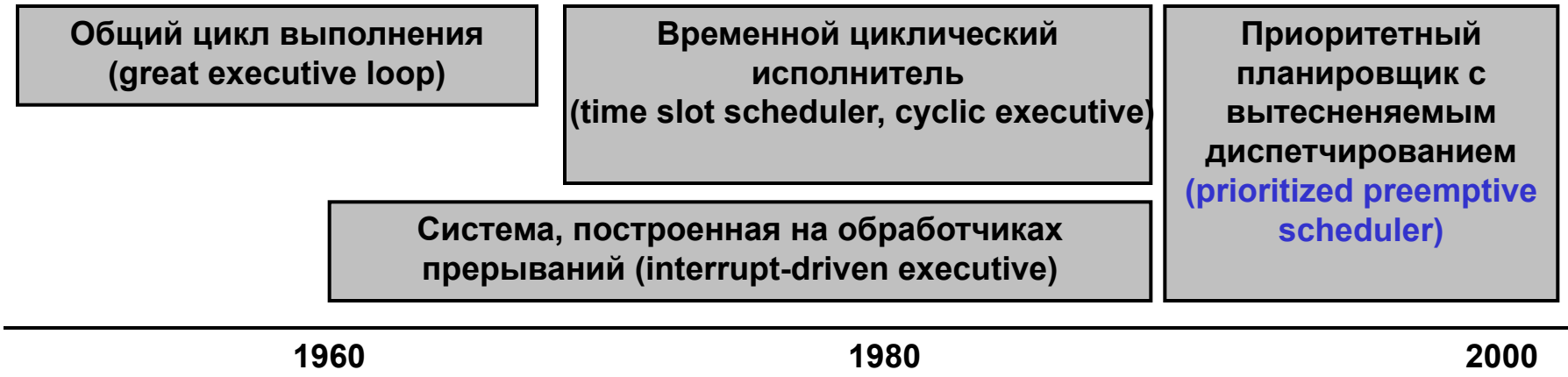
Примеры:

- **мульти-медиа приложения**
- **высоко-скоростные сети передачи данных**

2.4 Классификация и примеры событий

По времени возникновения / По типу возникновения	Периодические (periodic)	Спорадические (sporadic)	Апериодические (aperiodic)
	 Фиксированный период возникновения T	 Миниальный интервал между возникновением ограничен некоторым значением т	 Миниальный интервал между возникновением может быть любым
Внешние события - изменения состояния внешней среды (environmental)	• Периодическое поступление сетевого сообщения (напр. t° двигателя)	• Нажатие клавиши на клавиатуре (аппаратная защита от слишком частого нажатия)	• Сбой аппаратуры - генерация прерывания (Appolo-11) • Атака хакеров на Web-сервер
Внутренние события - изменения состояния внутри системы. (internal)	• Программная защита от зацикливания - периодический опрос состояния задач	• Коррекция курса самолета в случае предсказания коллизии (результат вычислений)	• Ошибка программы - бит события задачи не сбрасывается (всегда установлен)
Временные события (timed)	• Системный таймер	• Программируеиый интервальный таймер	• Одновременное истечение тайм-аутов передачи нескольких сетевых сообщений

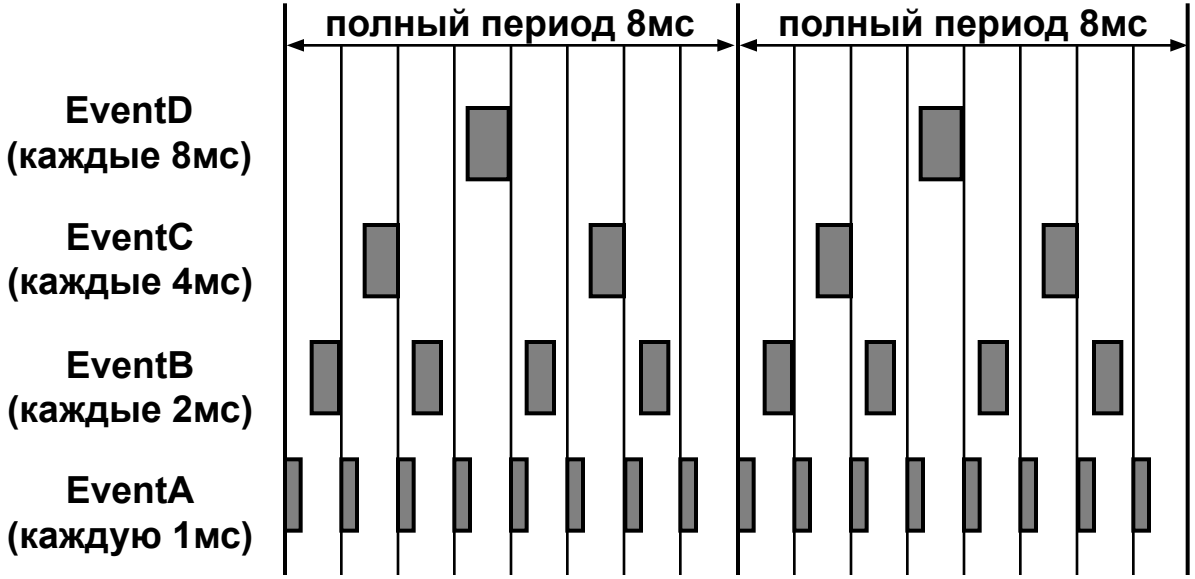
3. История развития встроенных ОС



Для того, чтобы ОС могла использоваться как основа приложения реального времени, она должна удовлетворять требованиям:

- **исполнимость**: предсказуемость работы приложения жесткого реального времени при любой (допустимой) нагрузке
- **одновременная обработка событий** различного типа и времени возникновения, и сроками исполнения (в том числе с существенно различными периодами и сроками исполнения)
- относительная **простота модификации** приложения при добавлении событий или изменении параметров событий (например, при уменьшении периода событий)
- **надежность** (отсутствие сбоев и крахов)
- минимально возможное **потребление ресурсов** - памяти и процессорного времени

3.1 Временной циклический исполнитель



Временной циклический исполнитель (cyclic executive).

Обработка событий привязана к временным промежуткам (таймерным слотам).

Преимущества:
- исполнимость (несложная проверка исполнимости худшего случая);
- надежность – обработчики вызываются как функции;
- небольшие расходы памяти процессора.

Недостатки:
- большие накладные расходы загрузки процессора - плохое его использование из-за частой проверки событий - особенно редких с коротким сроком исполнения (например, сигнала от датчика лобового удара);
- сложность модификации (при добавлении событий изменяется график, иногда нужно разбивать обработчик на несколько более коротких);
- невозможность приоритетного вытеснения обработки для обслуживания срочного события (по прерыванию).

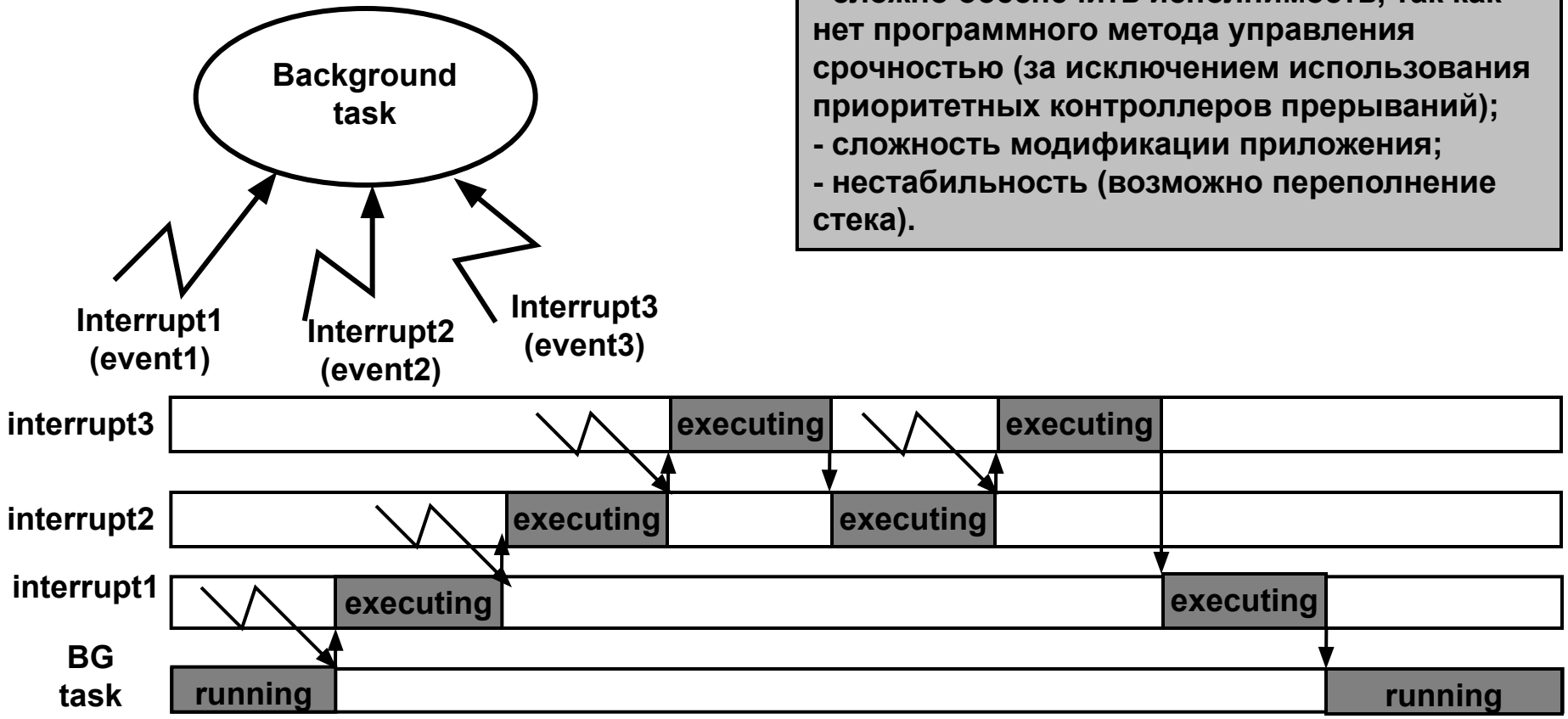
3.2 Система, управляемая прерываниями

Система, построенная на обработчиках прерываний (interrupt-driven executive).

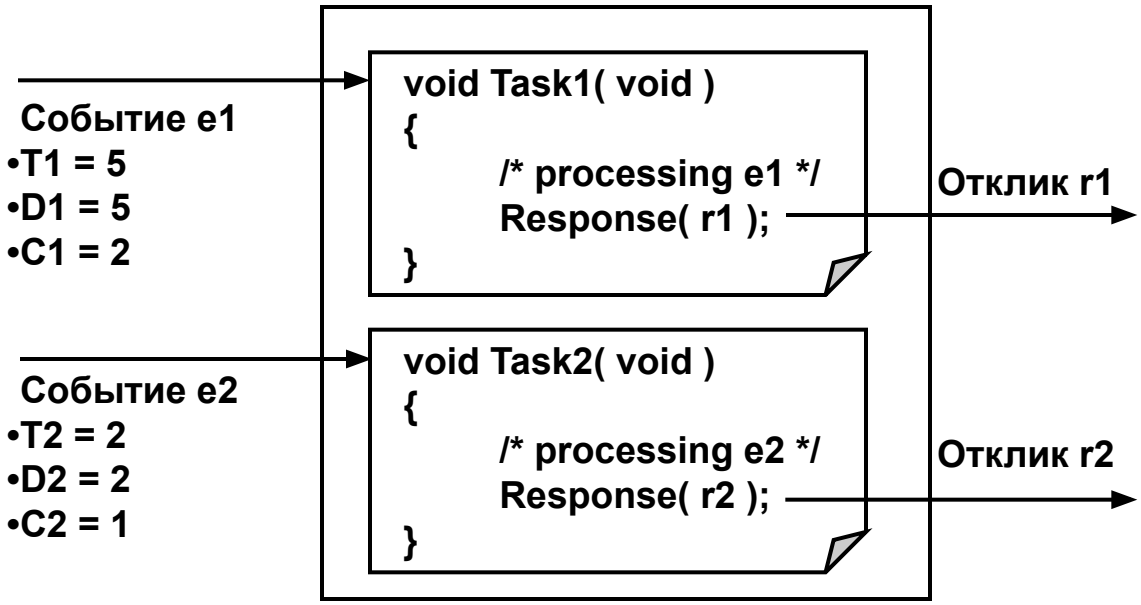
Обработка событий выполняется вложенными обработчиками прерываний.

Преимущества:
- управляется событиями;
- небольшое потребление памяти и процессорного времени.

Недостатки:
- сложно обеспечить исполнимость, так как нет программного метода управления срочностью (за исключением использования приоритетных контроллеров прерываний);
- сложность модификации приложения;
- нестабильность (возможно переполнение стека).



3.3 Приоритетный планировщик (1)



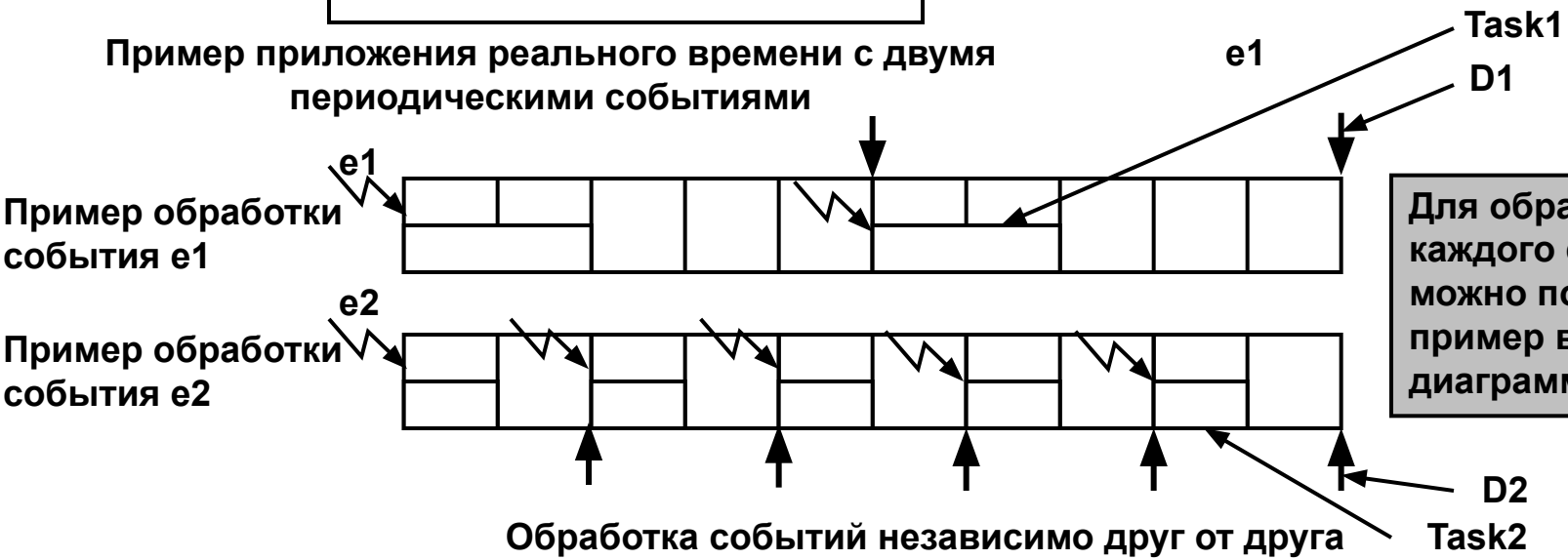
Пример приложения реального времени с двумя периодическими событиями

Для каждого события создается обработчик - например, функция на языке C.

Эта функция называется задачей (task).

Задачи не связаны друг с другом.

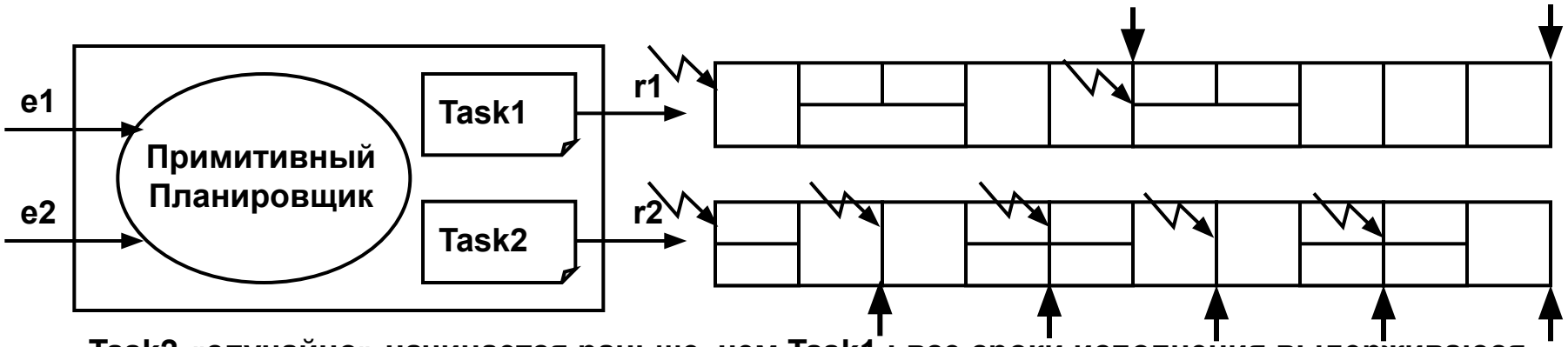
Задачи активизируются событиями, поддерживая концепцию «система управляется событиями» (event-driven).



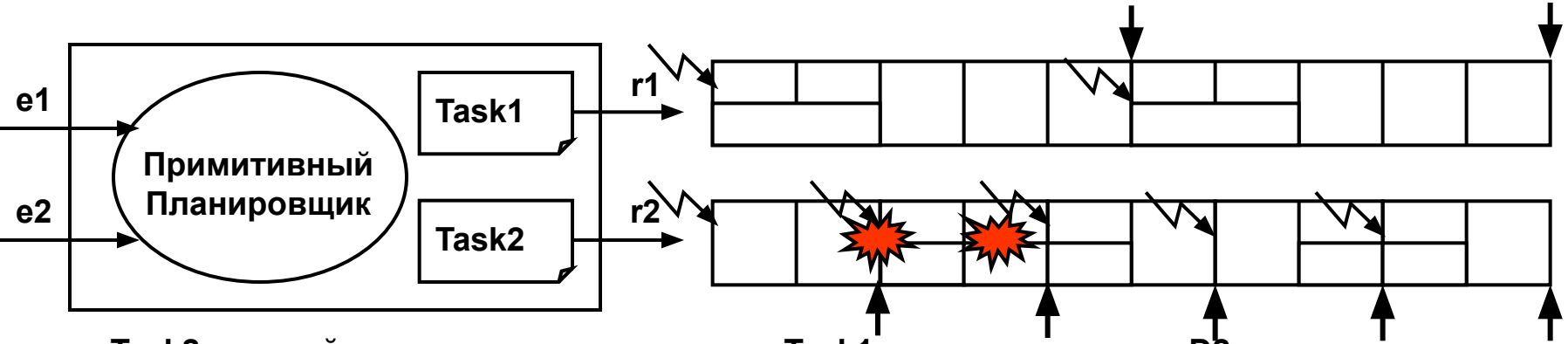
Обработка событий независимо друг от друга

Для обработки каждого события можно построить пример временной диаграммы.

3.3 Приоритетный планировщик (2)



Task2 «случайно» начинается раньше, чем Task1 : все сроки исполнения выдерживаются

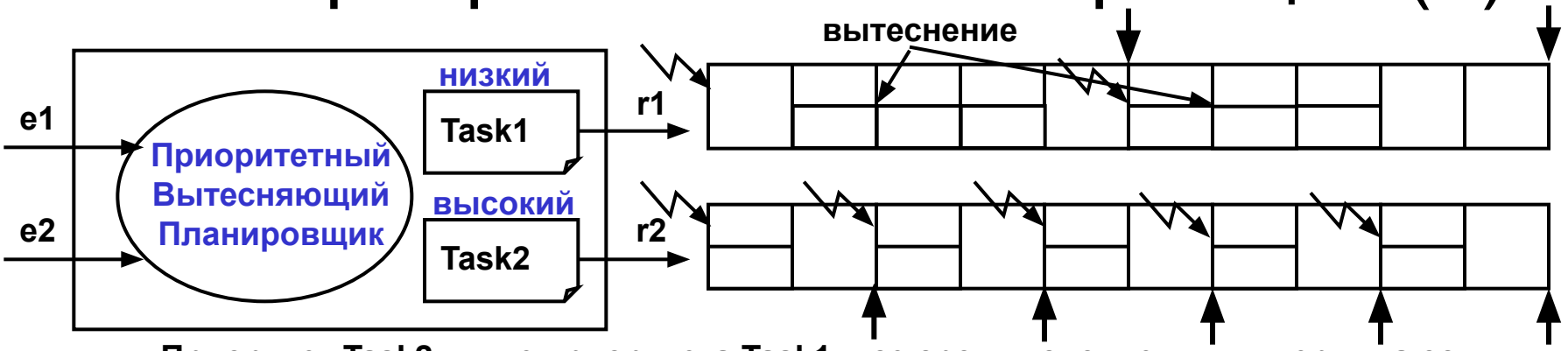


Task2 «случайно» начинается позже, чем Task1 : срок исполнения D2 нарушается и даже может быть пропущена обработка события.

- **В том случае, если планировщик не поддерживает приоритетность выполнения задач, нет гарантии, что сроки исполнения будут выдержаны, потому что сценарий выполнения зависит от того, обработка какого события начнется раньше.**
- **Следовательно, примитивный планировщик не годится для систем реального времени.**

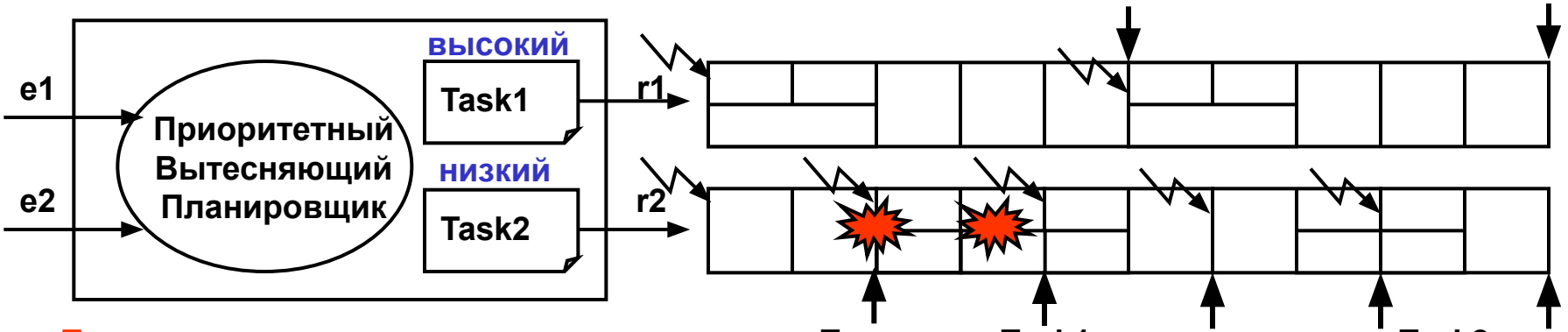
нарушение срока исполнения или пропуск обработки события

3.3 Приоритетный планировщик (3)



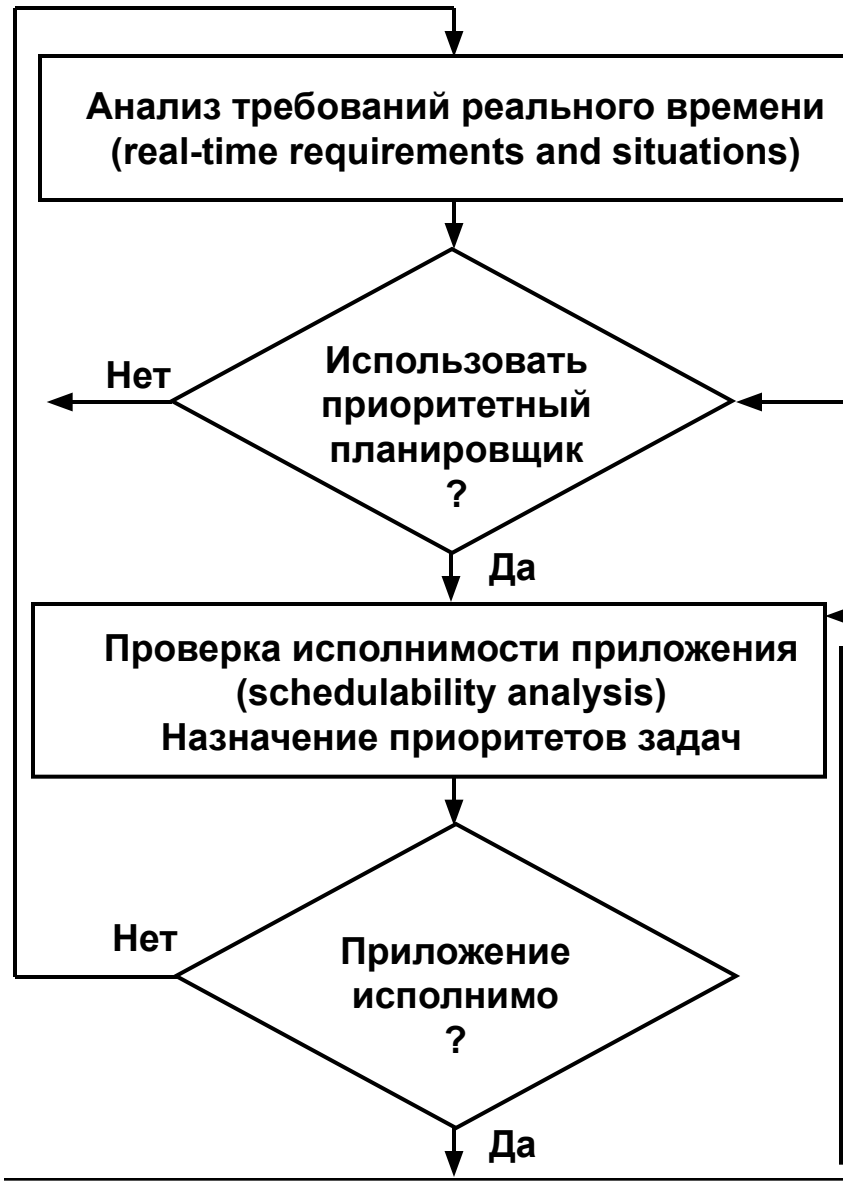
Приоритет Task2 выше приоритета Task1: все сроки исполнения выдерживаются

- **Приоритетный вытесняющий планировщик для задач с фиксированными приоритетами позволяет добиться гарантированного соблюдения сроков исполнения (исполнимости) при некотором оптимальном способе назначения приоритетов. Этот факт подтверждается математическим аппаратом.**
- **Точный график исполнения не создается - оцениваются только возможные сценарии.**
- **Поэтому имеет смысл разрабатывать приложение реального времени на основе исполнителя реального времени (например, приоритетного планировщика).**



Приоритеты должны назначаться правильно. Приоритет Task1 выше приоритета Task2: срок исполнения D2 нарушается, и даже может быть пропущена обработка события.

3.3 Приоритетный планировщик (4)



Анализ характеристик событий.
Анализ сроков исполнения.
Проектирование обработчиков событий (задач).
Оценка времен обработки событий.

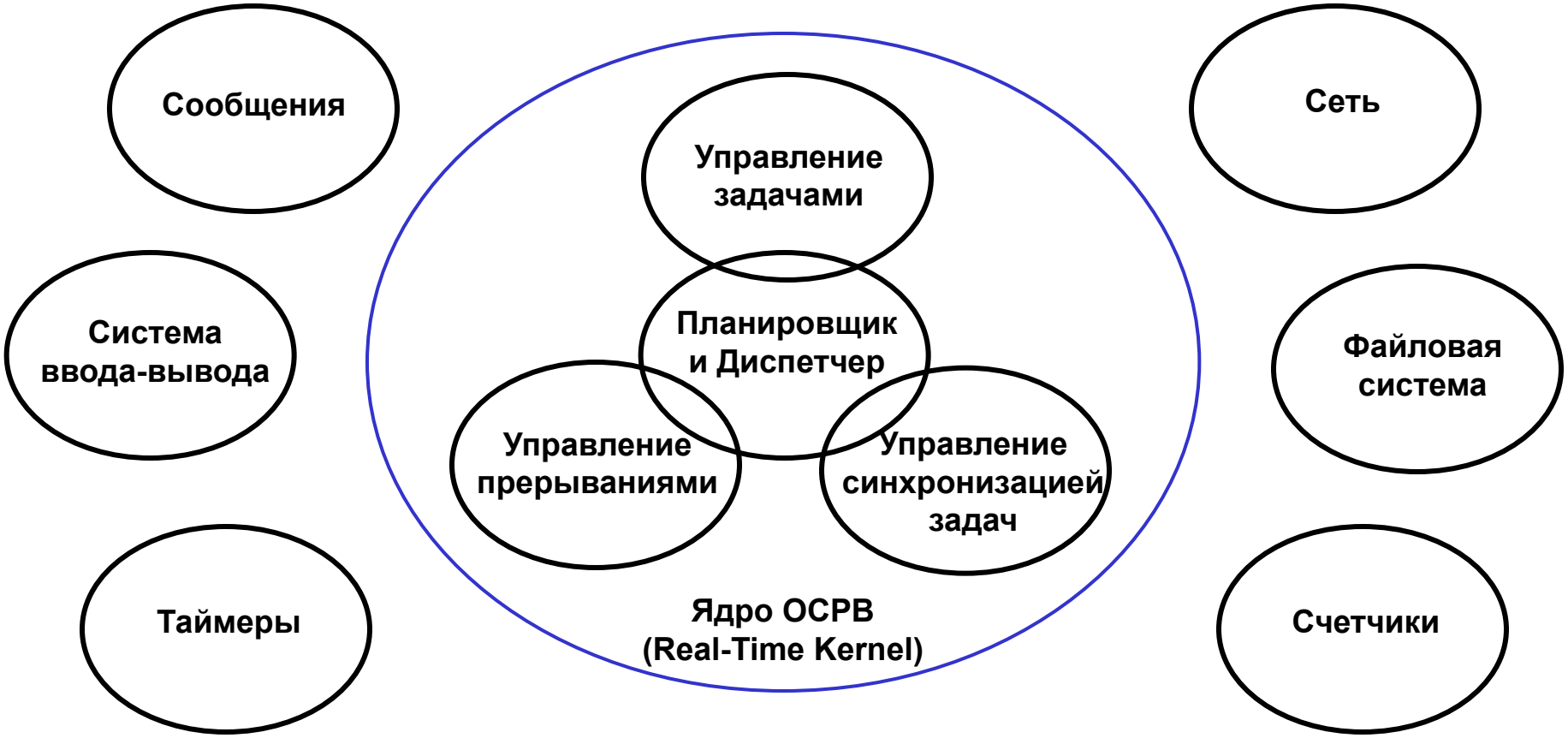
Оценка накладных расходов.
Прогноз модификаций приложения.
Анализ стоимости.

Обычно используется метод назначения приоритетов "чем меньше срок исполнения, тем выше приоритет" (RMS).

Преимущества:
- математически доказанные методы, гарантирующие выполнение требований реального времени (для периодических и спорадических событий);
- поддержка концепции «система управляется событиями»;
- простота модификации приложения.

Недостатки:
- накладные расходы на работу собственно планировщика.

4. Характеристики встроенных ОС (1)



Операционная система реального времени - программа, распределяющая вычислительные ресурсы таким образом, чтобы обеспечить выполнение требований реального времени для приложения, использующего ОСРВ.

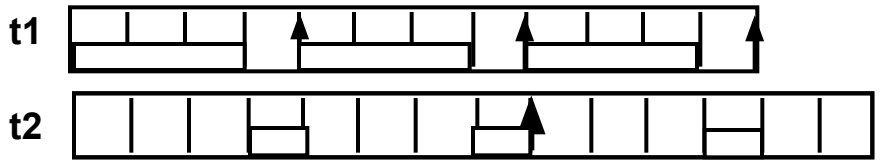
4. Характеристики встроенных ОС (2)

Характеристика	ОС общего назначения (ОСОН) (Windows NT, Unix)	Встроенные ОС реального времени
• Производительность	• “Равные” условия для всех задач	• “Привилегированные” условия для срочных задач
• Реактивность	• “Быстрый” усредненный отклик	• Обеспечение ответа реального времени
• Планирование	• Разделение времени; нестрого-“приоритетное”; приоритет коротким задачам	• Обычно приоритетное вытесняющее с большим количеством приоритетов
• Инверсия приоритетов задач	• Возможна; в том числе неограниченная по времени (P/V семафоры)	• Исключена (кроме ограниченной при доступе к разделяемым ресурсам)
• Объем занимаемой памяти	• Несколько мегабайт кода, около мегабайта данных	• Несколько килобайт кода, около сотни байт данных
• Время исполнения сервисов	• Сотни микросекунд	• Единицы и десятки микросекунд
• Предсказуемость	• “Отсутствует” (при перегрузке)	• Гарантируется всегда

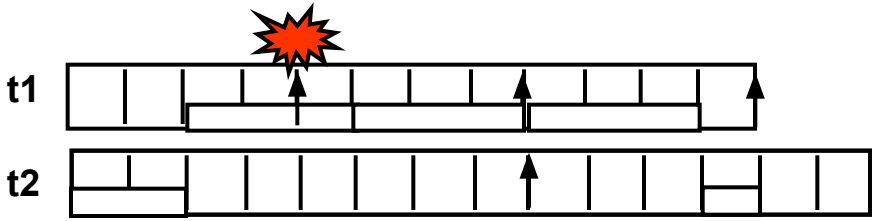
4. Характеристики встроенных ОС (3)

Приоритетное вытесняющее планирование в ОСРВ и планирование «приоритет коротким задачам» в ОСОН

t1: T1=D1=4, C1=3; t2: T2=D2=8, C2=2;



RMS (приоритет t1 выше приоритета t2): приложение всегда исполнимо

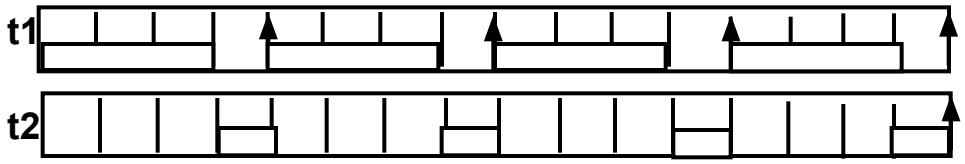


Short Job First (приоритет t1 ниже приоритета t2): не выдерживается срок исполнения D1

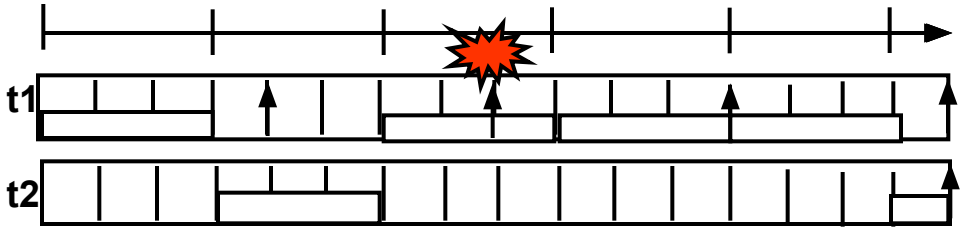
Планирование «приоритет коротким задачам» не учитывает сроки исполнения задач, и поэтому не применяется в ОСРВ.

Планирование с немедленным вытеснением в ОСРВ и планирование по таймеру в ОСОН

t1: T1=D1=4, C1=3; t2: T2=D2=16, C2=4;



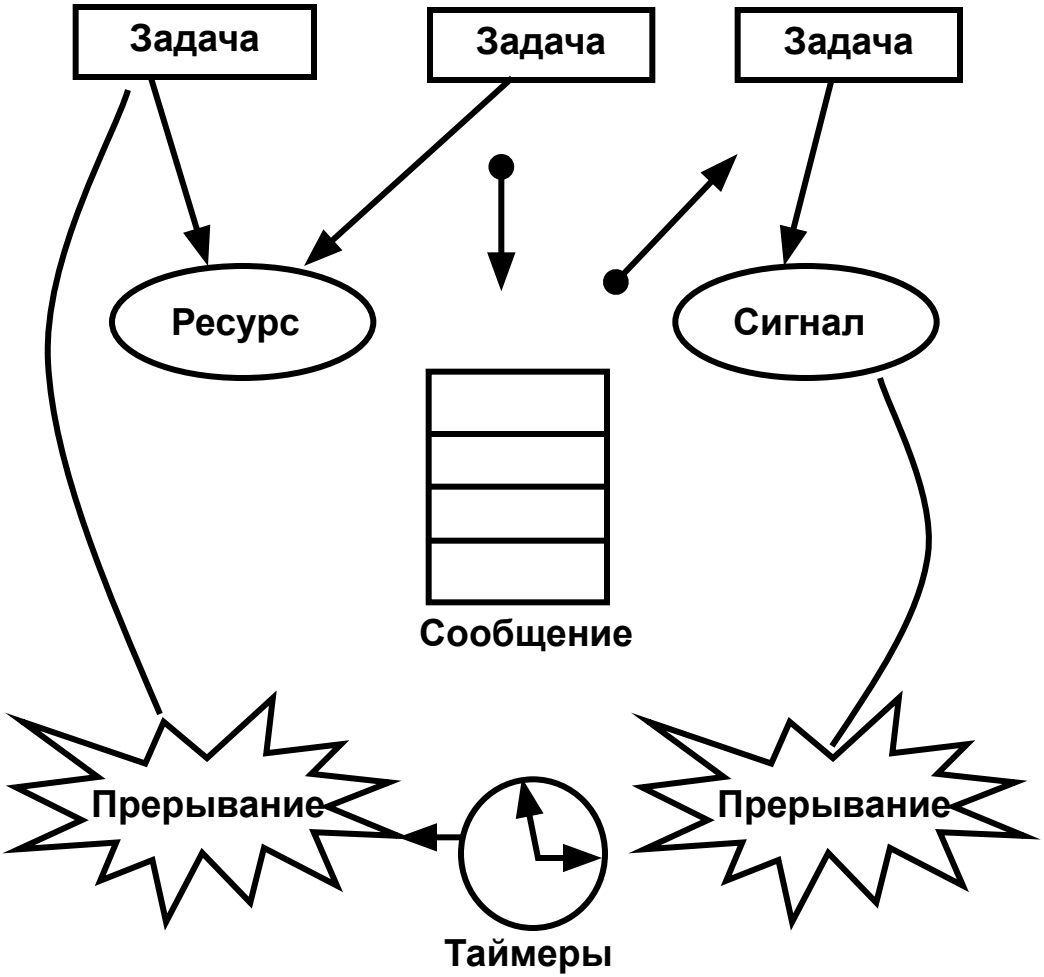
Немедленное вытеснение: приложение всегда исполнимо



Планирование по таймеру с периодом равным 3: не выдерживается срок исполнения D1

Планирование по таймеру приводит к инверсии приоритетов, и, как следствие, к нарушению сроков исполнения. Обычно не применяется в ОСРВ.

5. Базовые объекты (1)



Задачи

Обработчики прерываний

Семафоры для доступа к разделяемым ресурсам

Сообщения

События (флаги, сигналы)

Таймеры и счетчики

5.1 Задачи

```
void TaskA_Entry( void )
{
        /* processing */

        Activate( TaskB );

        Terminate();
}

void TaskB_Entry( void )
{
        /* processing */

        Terminate();
}
```

Задача (task) - единица обработки, выполняющаяся конкурентно с другими задачами.

Задачи являются основным средством обработки внутренних событий.

Задача имеет некоторое значение приоритета, определяющее ее относительные претензии на захват процессора. Эти претензии удовлетворяются ОС по определенному алгоритму. Вместо приоритета может использоваться значение срока исполнения.

Задача имеет точку входа (entry point).

Задача обычно является функцией, записанной на языке С (С++) и идентифицируется некоторым идентификатором (языка С).

Задача обычно выполняет вызовы ОС для взаимодействия с другими задачами (хотя бы один вызов завершения задачи).

Обычно задача встроенной ОС эквивалентна thread обычных ОС, т.е. работает в одном адресном пространстве с другими задачами.

5.2 Обработчики прерываний

```
void interrupt Isr1( void )
{
        ISREntry();

        /* processing */

        ISRActivate( TaskB );

        ISRExit();
}
```

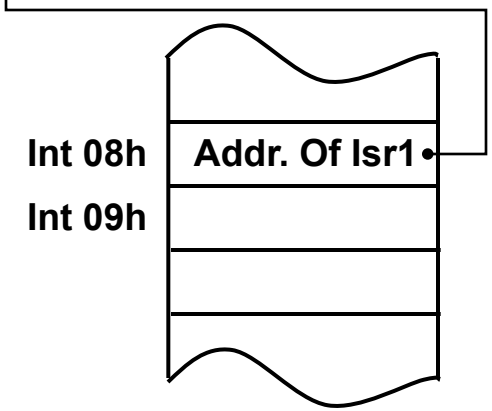


Таблица векторов прерываний

Обработчик прерываний (interrupt service routine, software interrupt handler) - единица обработки, инициированная аппаратным прерыванием асинхронно по отношению к выполнению задач и самой ОС.

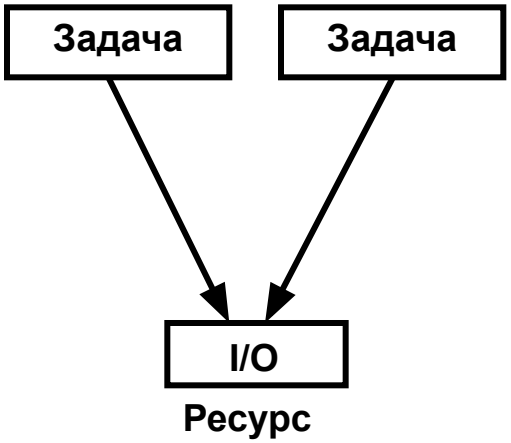
Обработчики прерываний являются основным средством обнаружения возникновения внешних и временных событий.

Обработчики прерываний занимают процессор в соответствии с алгоритмом планирования, поддерживаемым аппаратурой.

Для выполнения вызовов ОС обработчик прерываний обычно включает специальный пролог и эпилог, которые обеспечивают необходимый контекст выполнения.

Обычно обработчики прерываний выполняют только предварительное обслуживание событий, и, генерируя внутреннее событие, планируют задачу для завершения обработки событий и выработки откликов.

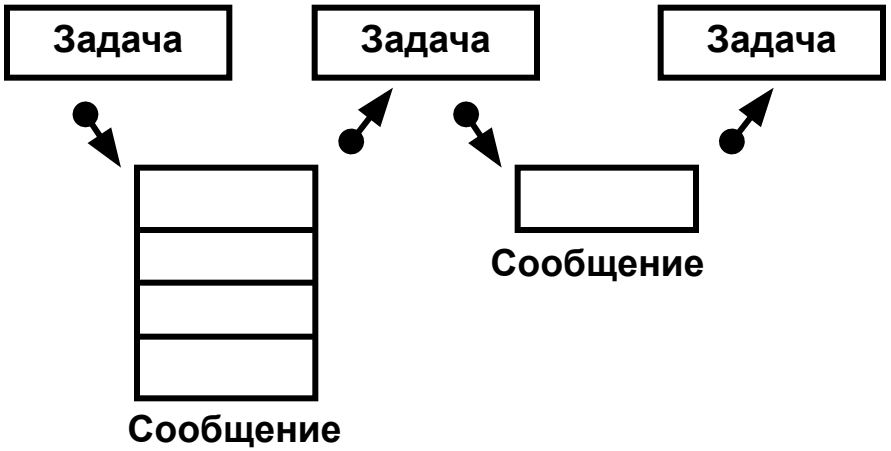
5.3 Ресурсы



Семафоры предназначены для взаимосключающего доступа задач (и обработчиков прерываний) к критическим секциям кода, т.е. к разделяемым ресурсам.

Специальные протоколы доступа к семафорам применяются для исключения блокировок, тупиковых ситуаций, и неограниченной инверсии приоритетов.

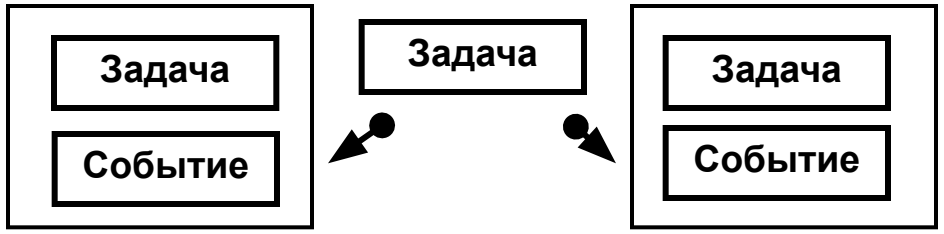
5.4 Сообщения



Сообщения (messages) предназначены для обмена данными любого типа между задачами и обработчиками прерываний.

Сообщения обычно копируются в системную область.

5.5 События (флаги, сигналы)

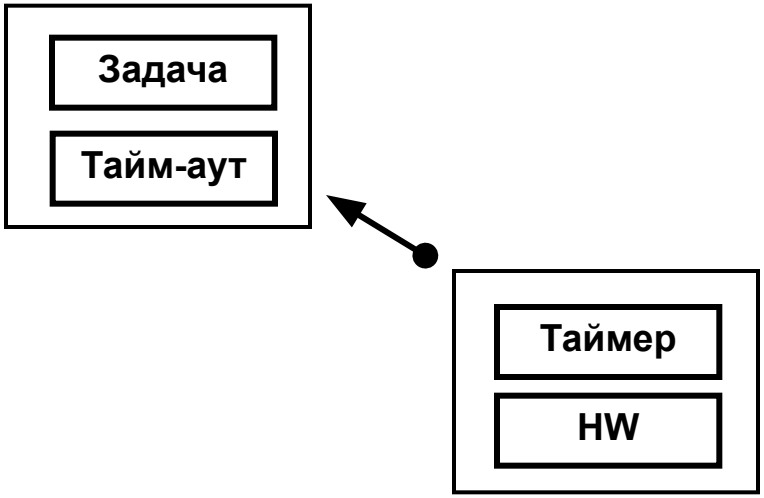


События (events) предназначены для обмена двоичными данными между задачами и обработчиками прерываний.

События также реализуются в виде флагов (masks, flags) или сигналов (signals).

События обычно принадлежат задачам (не разделяются между задачами).

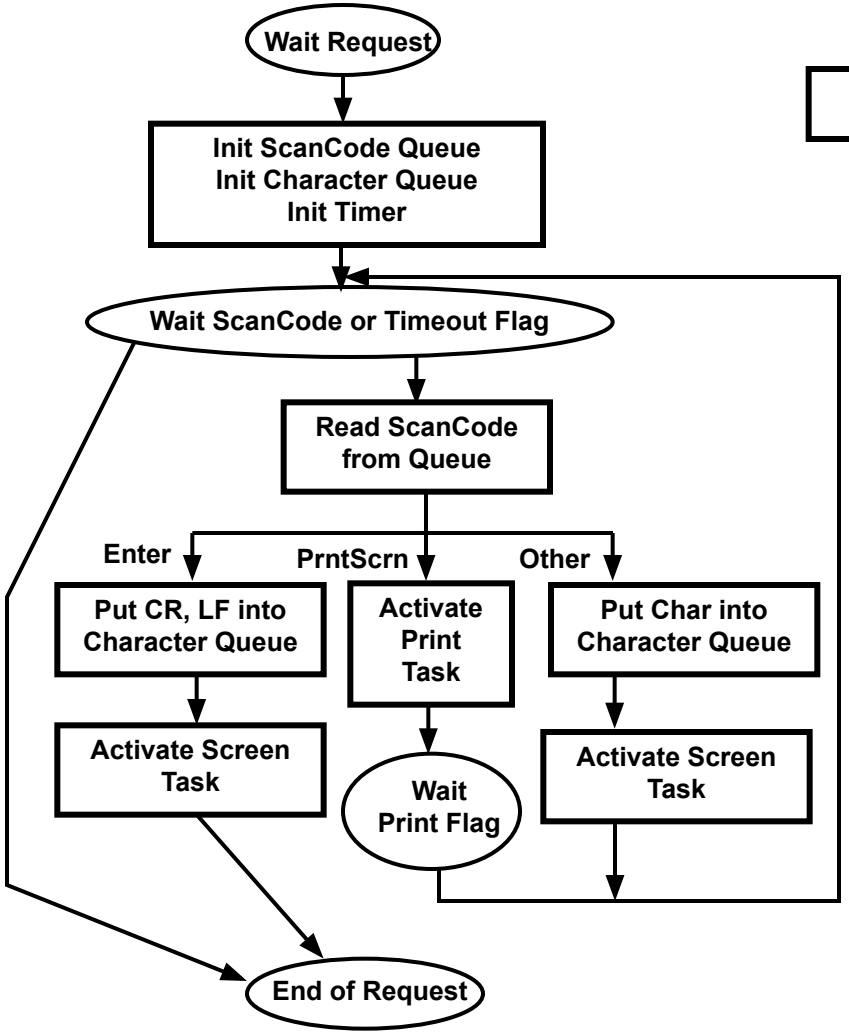
5.6 Таймеры и счетчики



Таймеры (timers) предназначены для задания временных интервалов для задач, а также подсчета абсолютного значения времени.

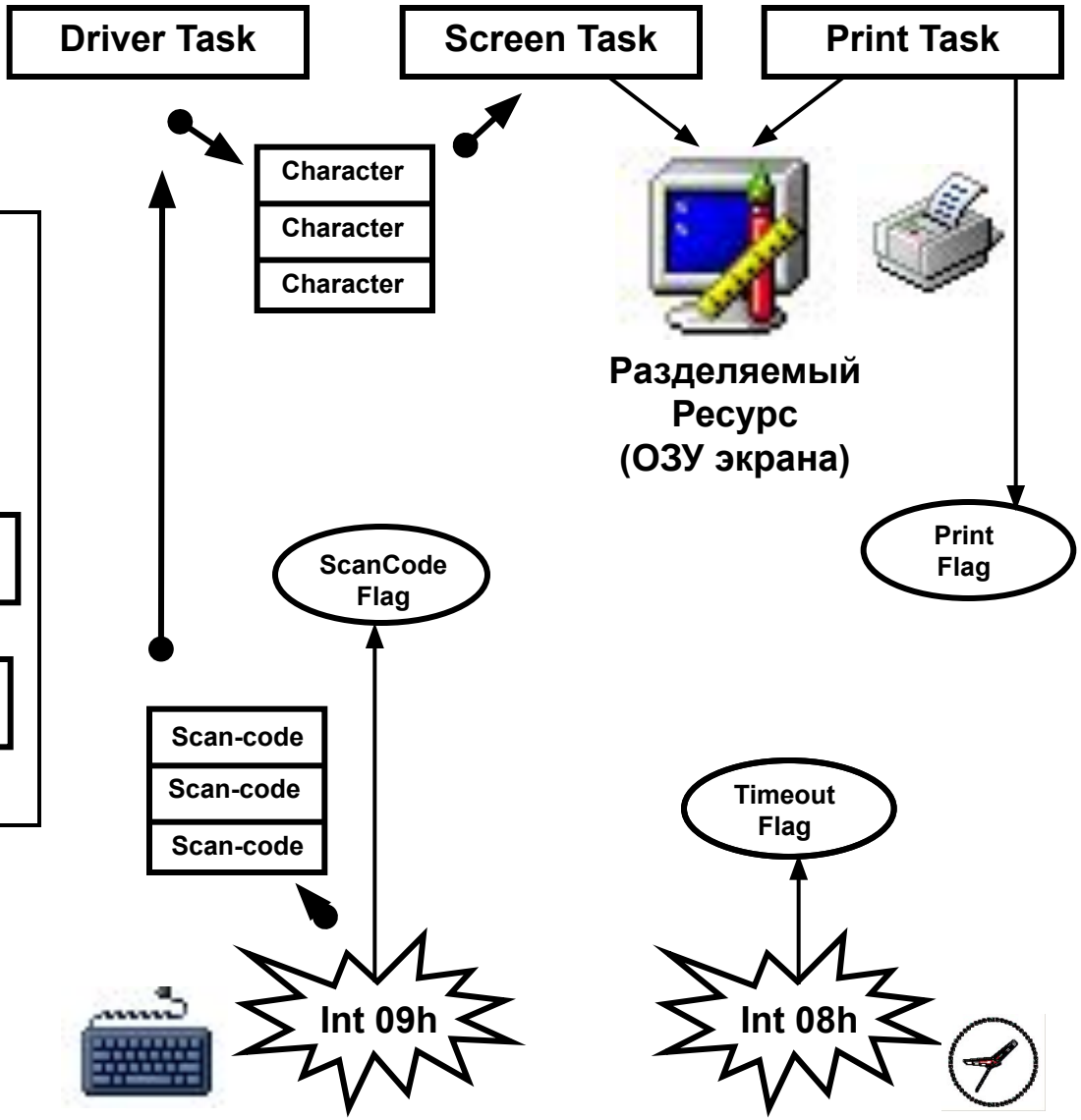
Счетчики (counters) предназначены для отслеживания абсолютного значения или перемещения механических устройств (например, угла поворота вала).

5.7 Базовые объекты (пример)

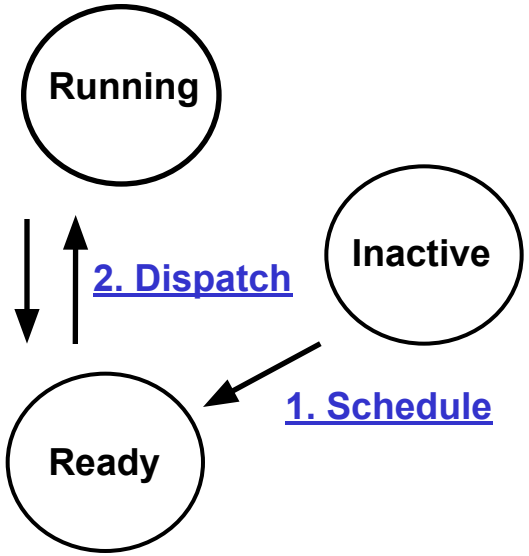


Алгоритм Driver Task

Пример драйвера терминала

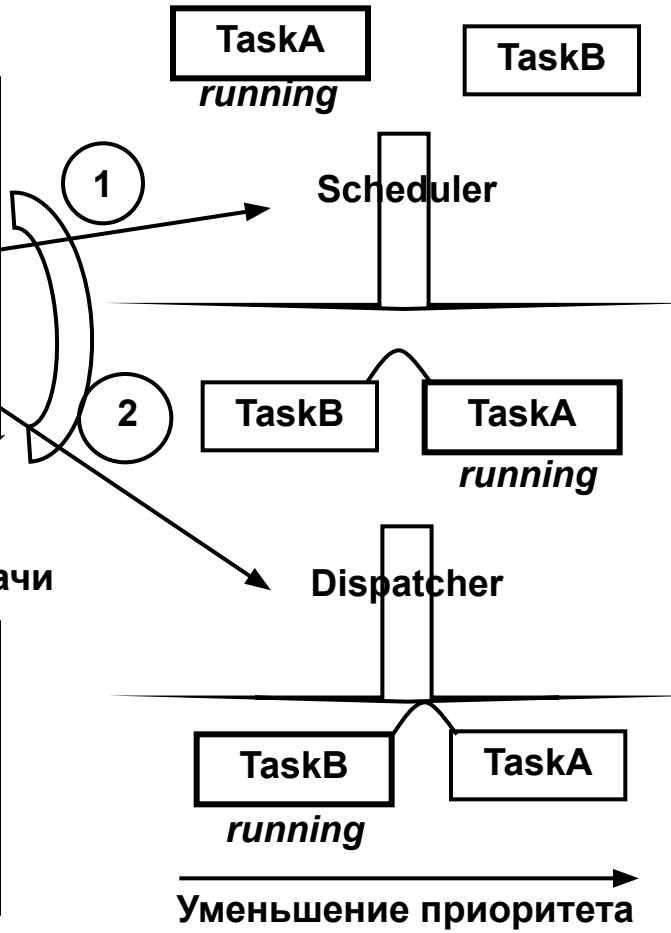


6. Планирование и диспетчеризация (1)



Состояния задач

```
void TaskA_Entry(void)
{
Activate( TaskB );
}
```



Запуск более приоритетной задачи

Планирование состоит из двух шагов, выполняющихся друг за другом:
1. Собственно **планирование** (scheduling), т.е. составления расписания.
2. **Диспетчеризация** (dispatching) - выбора задачи из списка и назначения ей процессора (переключение контекста).

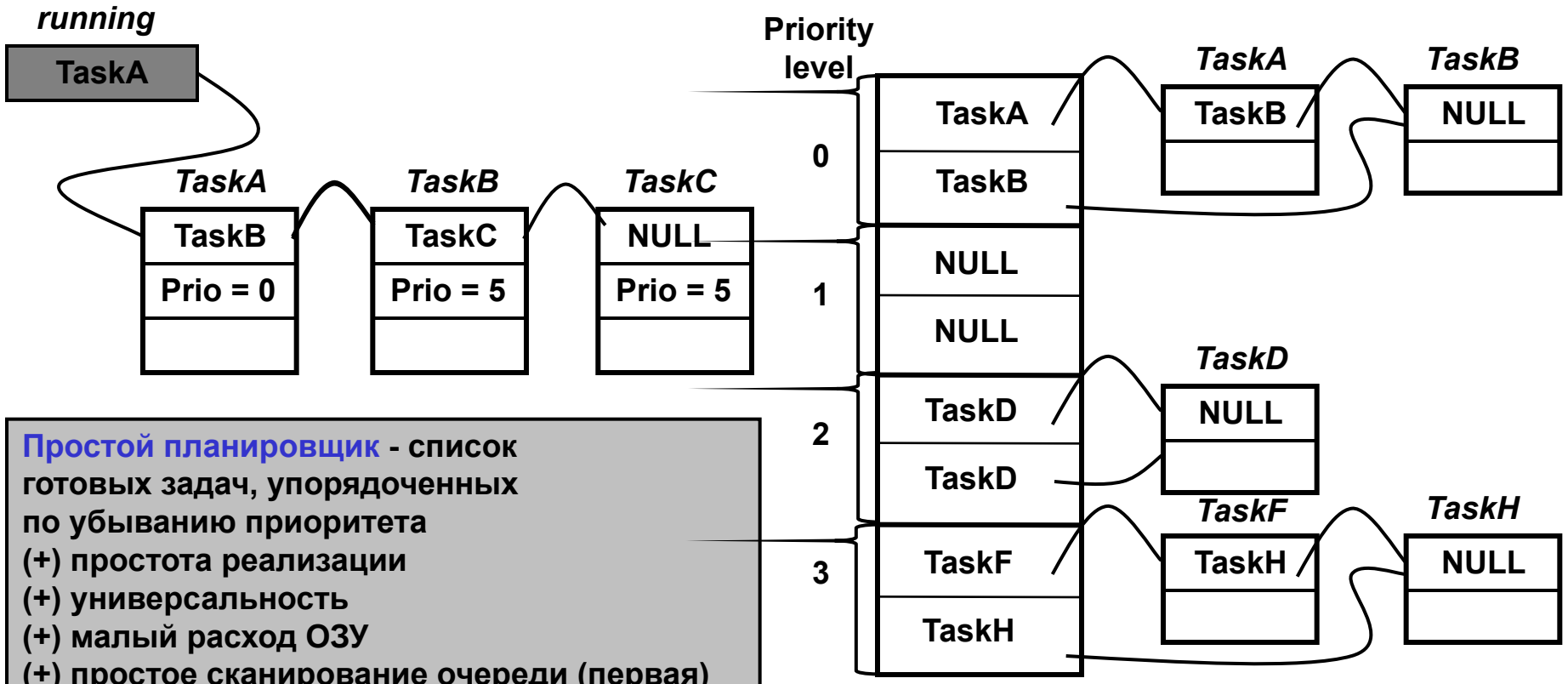
Для гарантирования соблюдения сроков исполнения **не должно быть инверсии приоритетов**, поэтому диспетчеризация должна выполняться немедленно после планирования **(не по таймеру!)**.

6. Планирование и диспетчеризация (2)

Планирование и диспетчеризация в системах реального времени должно удовлетворять следующим требованиям:

- **строгое соблюдение дисциплины планирования**
- **полное исключение инверсии приоритетов между задачами (за исключением специальных планировщиков – например, невытесняющих)**
- **сохранение контекста задачи при вытеснении ее с процессора**
- **восстановление контекста задачи при назначении ей процесоора**
- **минимально возможное потребление ресурсов - памяти и процессорного времени**

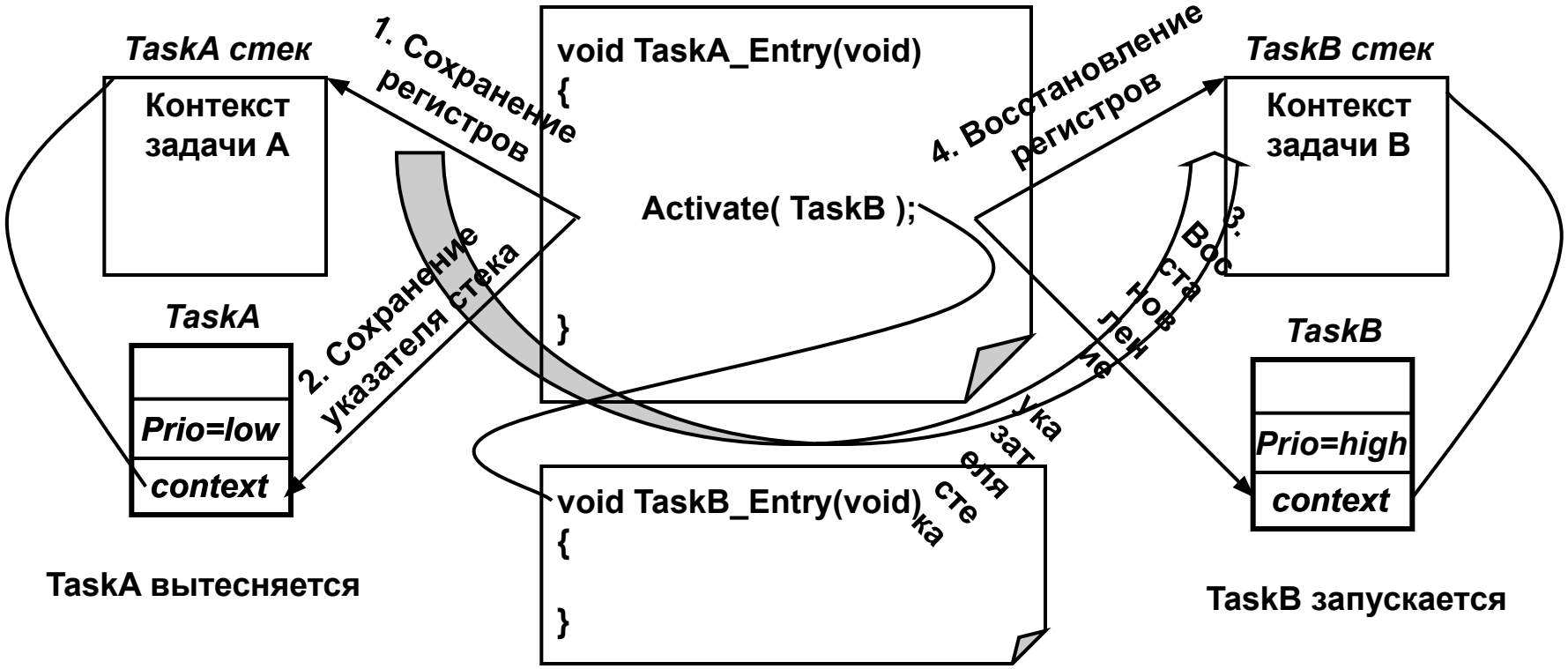
6. Планирование и диспетчеризация (3)



Простой планировщик - список готовых задач, упорядоченных по убыванию приоритета
(+) простота реализации
(+) универсальность
(+) малый расход ОЗУ
(+) простое сканирование очереди (первая)
(-) время постановки в очередь варьируется в зависимости от приоритета задачи и числа задач в очереди

«POSIX» планировщик поддерживает списки задач для каждого приоритета
(+) быстрая постановка в очередь
(+) время постановки в очередь всегда одинаково
(-) большой расход ОЗУ
(-) требуется цикл сканирования очередей

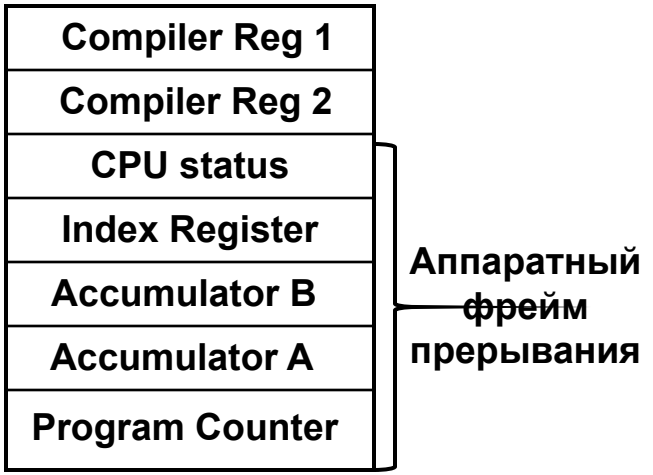
6. Планирование и диспетчеризация (4)



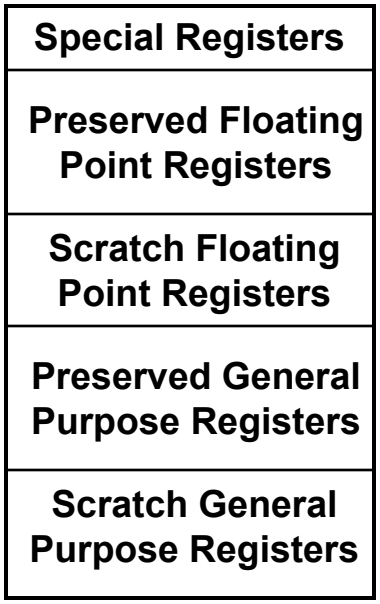
«Кажущийся» вызов задачи В из задачи А осуществляется с помощью:
1. сохранения значений регистров процессора на стеке выполняющейся задачи А,
2. запоминания указателя стека в описателе задачи А (для того, чтобы впоследствии продолжить выполнение задачи А),
3. загрузки указателя стека процессора из описателя задачи В,
4. восстановления значений регистров процессора со стека задачи В.

Переключение контекста задач при запуске высокоприоритетной задачи из низкоприоритетной задачи (вытеснение).

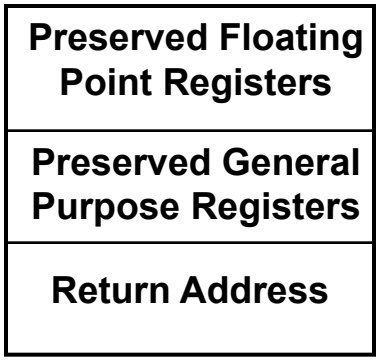
6. Планирование и диспетчеризация (5)



Контекст задачи для типичного 8-битного процессора (~9 байт)



Контекст задачи для типичного 32-битного процессора (~64+ байта)



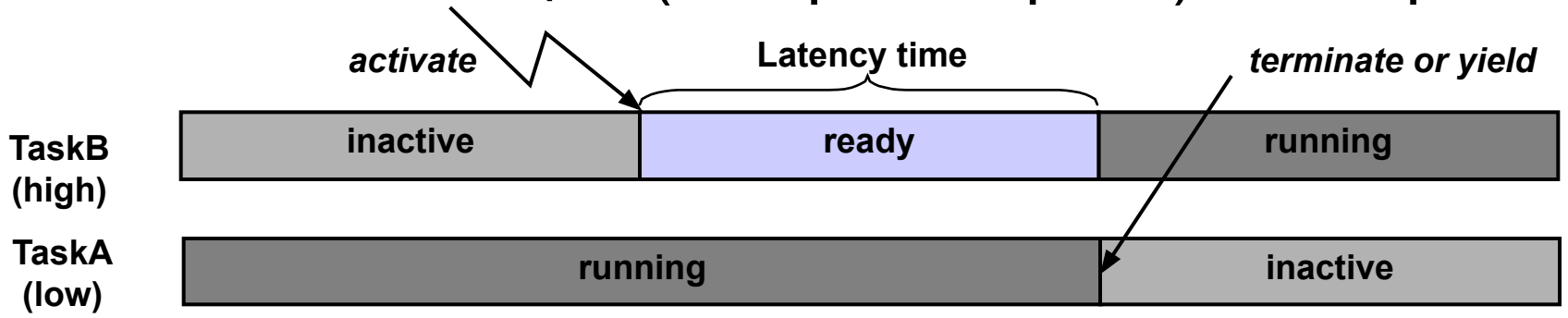
Оптимизированный контекст задачи для вызова задачи как функции (~2+ байта).

Применяется в случае использования «общего стека» для всех задач.

- **Контекст задачи обычно сохраняется на стеке задачи**
- **Указатель на контекст (вершина стека) заносится в описатель задачи**
- **Переключение контекста часто выполняется с помощью команд процессора, предназначенных для обработки прерываний («программное прерывание», «возврат из прерывания»)**

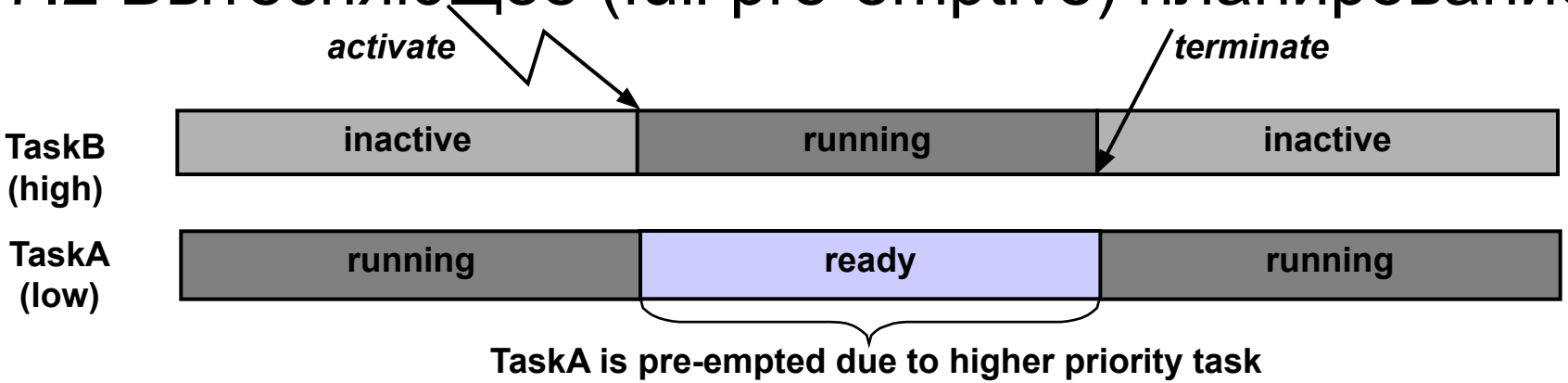
7. Типы планирования (1)

7.1 Невытесняющее (non pre-emptive) планирование



• Обычно применяется для быстрой обработки события (чтобы не тратить время на «лишнии» переключения контекста)

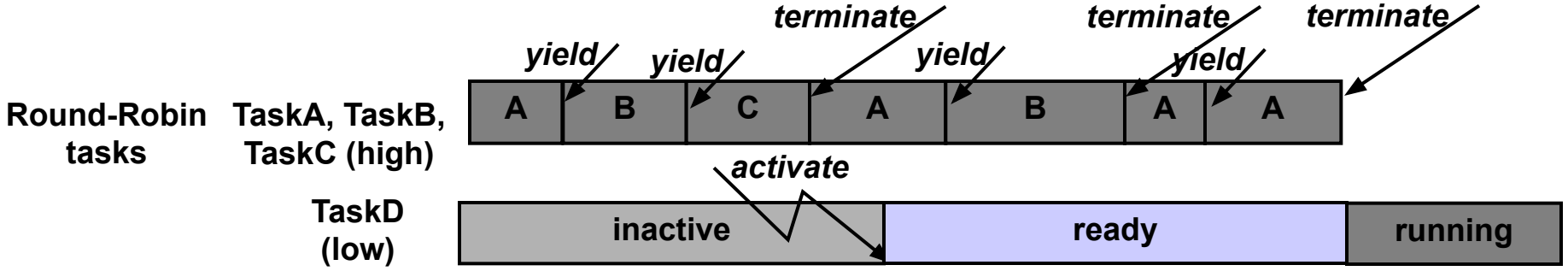
7.2 Вытесняющее (full pre-emptive) планирование



• Основной тип планирования в системах жесткого реального времени

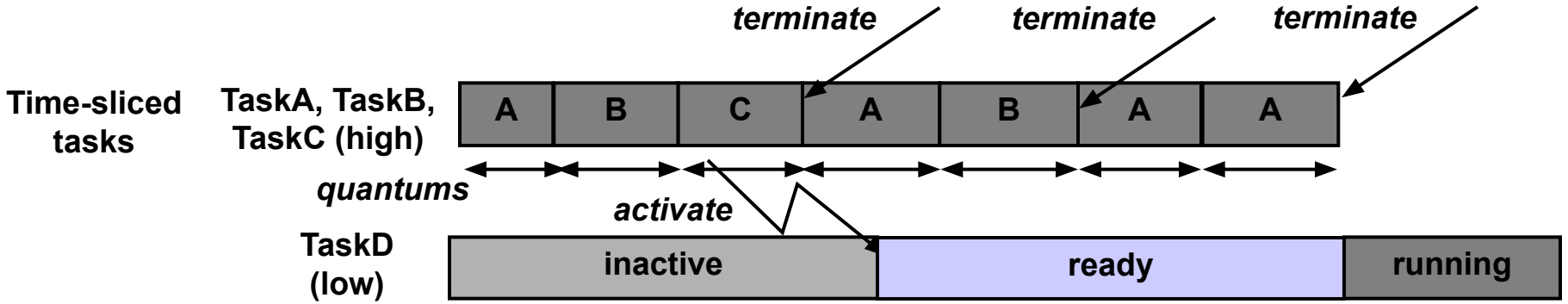
7. Типы планирования (2)

7.3 Круговое (Round-Robin) планирование



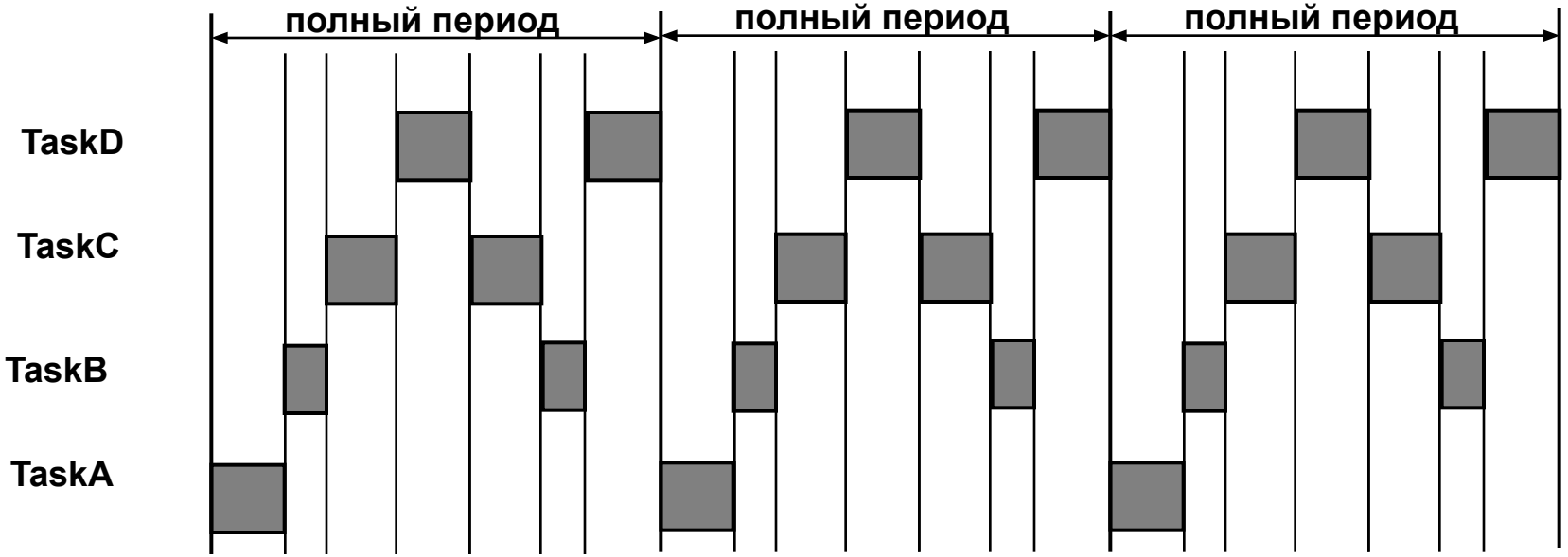
- Обычно применяется как замена «общего цикла выполнения»
- ОС не влияет на вытеснение задач и на время выполнения задач

7.4 Планирование с квантованием (time-sliced)



- Похоже на round-robin, но вытеснение задач происходит принудительно по истечении кванта времени (таким образом, ОС влияет на время выполнения задач)

7. Типы планирования (3)
7.5 Time-triggered scheduling (X-by-wire)



Свойства time-triggered планирования:
1. **Детерминированность (replica determinism).**
2. **Двойное и тройное исполнение задачи для сравнения результата вычислений.**
3. **Жесткий (непериодический) график задач строится до исполнения (off-line).**
4. **Прерывания разрешаются только в определенные моменты времени или не разрешаются во время полного цикла выполнения.**
5. **Возможна исключительно эффективная реализация исполнителя графика.**

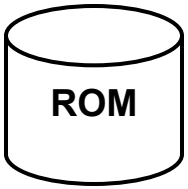
Но: построение оптимального off-line графика выполнения в общем случае относится к классу NP-complete задач, и, следовательно, практически неосуществимо.
Европейский проект реализации в различных областях: http://www.setta.org

8. Управление задачами (1)

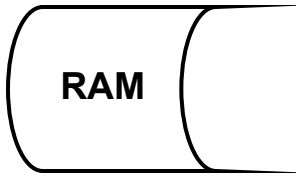
Для того, чтобы обработка внутренних событий выполнялась в реальном времени, механизмы управления задачами должны удовлетворять следующим требованиям :

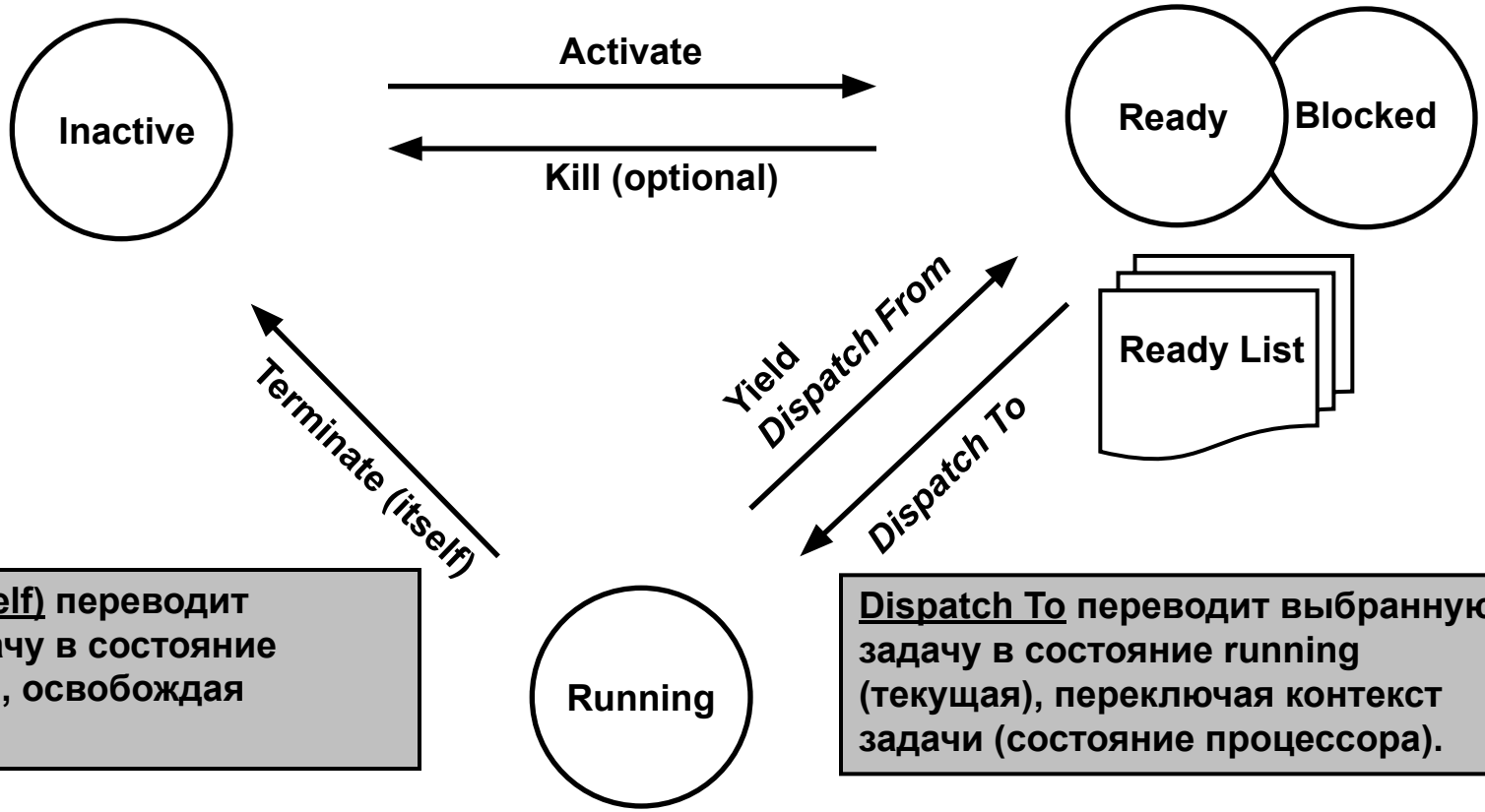
- **поддержка задач однократного выполнения (без состояния ожидания)**
- **поддержка задач с состоянием ожидания (нужны для естественной реализация машины состояний)**
- **статическое создание задачи (обычно off-line)**
- **минимально возможное потребление ресурсов - памяти и процессорного времени**

8. Управление задачами (2)



Activate создает в ОЗУ управляющий блок задачи пользуясь описателем, находящимся в ПЗУ (подобно загрузке исполняемого файла с диска в память).

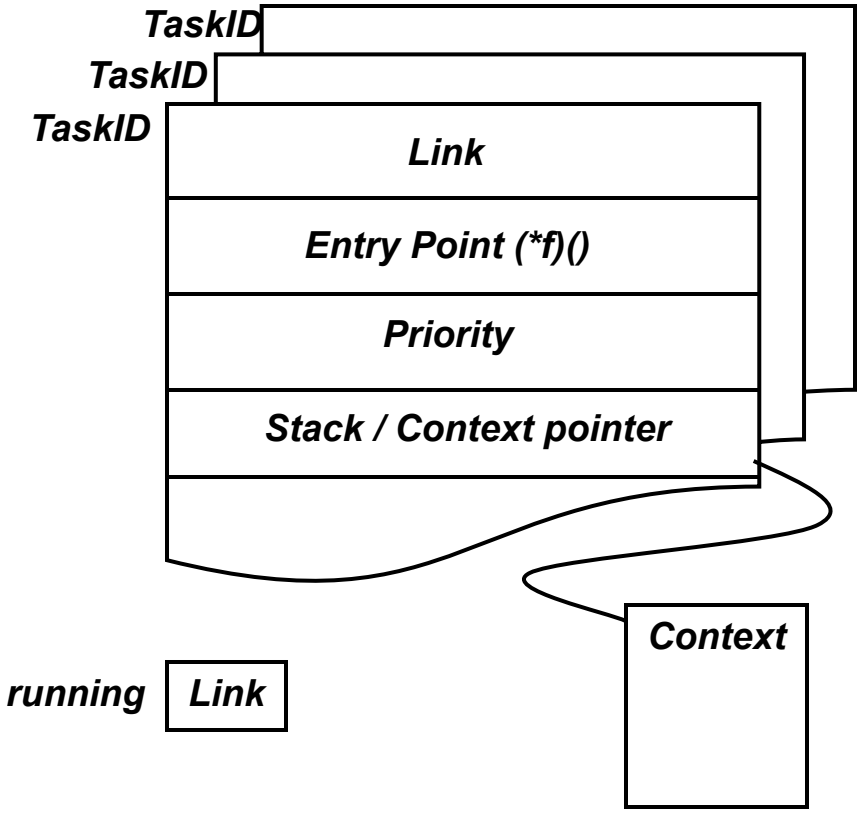




Terminate (itself) переводит текущую задачу в состояние неактивности, освобождая области ОЗУ.

Dispatch To переводит выбранную задачу в состояние running (текущая), переключая контекст задачи (состояние процессора).

8. Управление задачами (3)

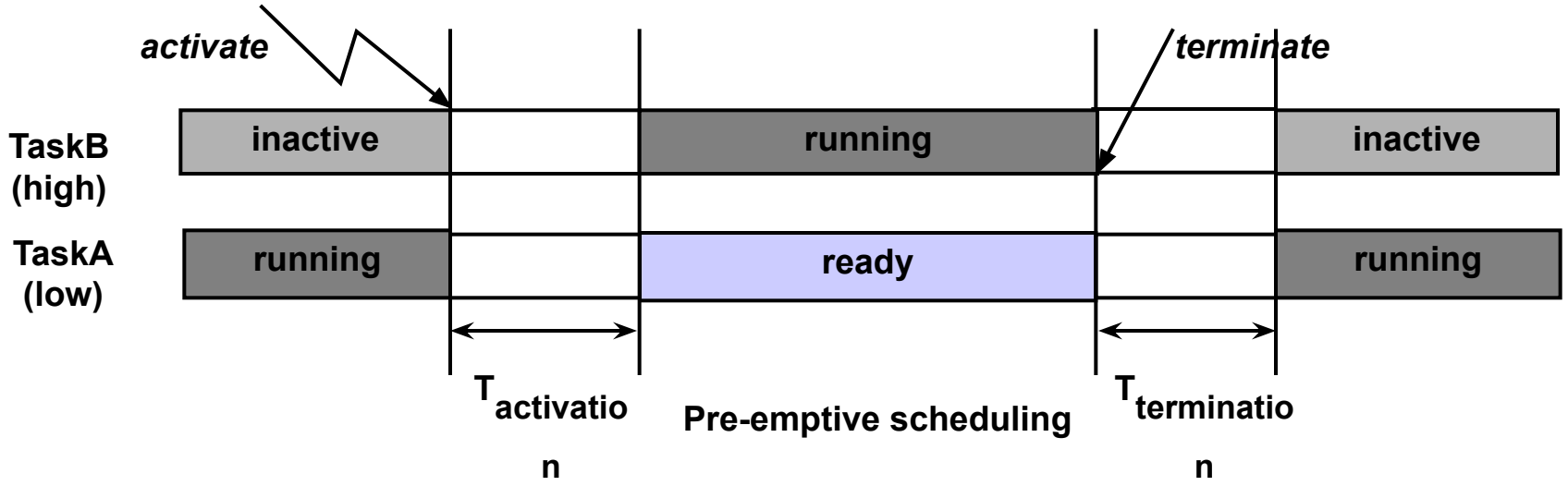


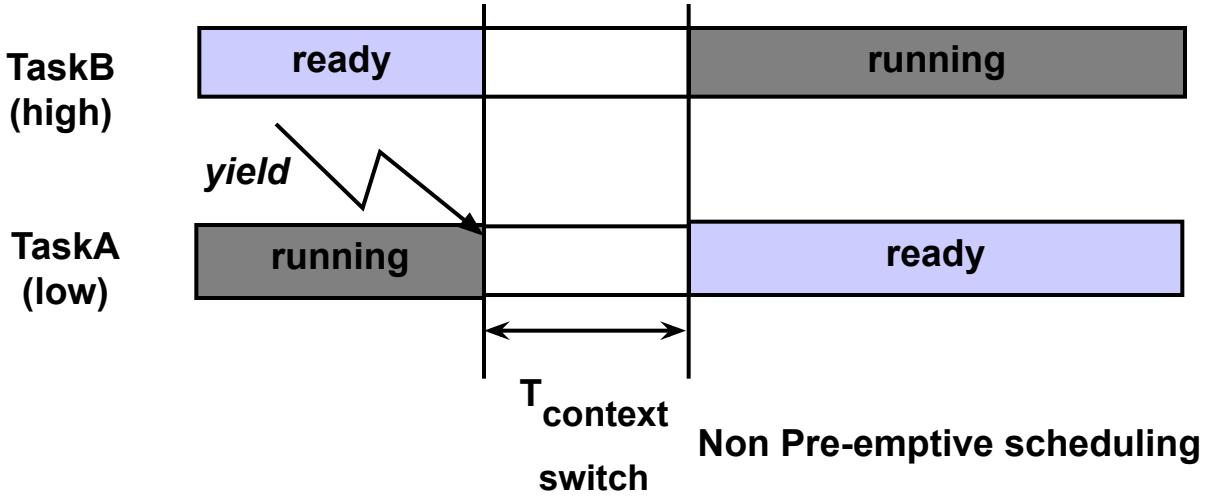
Структуры данных для управления задачами

- **Activate(TaskID) - планирует задачу (inactive -> ready), и вызывает диспетчер.**
- **Terminate() - задача завершает саму себя (running -> inactive), и вызывает диспетчер.**
- **Yield() - задача освобождает процессор для более приоритетной задачи, (running -> ready), и вызывает диспетчер.**
- **IdleLoop() - “for(;;) { }” (в некоторых системах эту роль выполняет низкоприоритетная задача) («jump *»)**

Сервисы для управления задачами

8. Управление задачами (4)

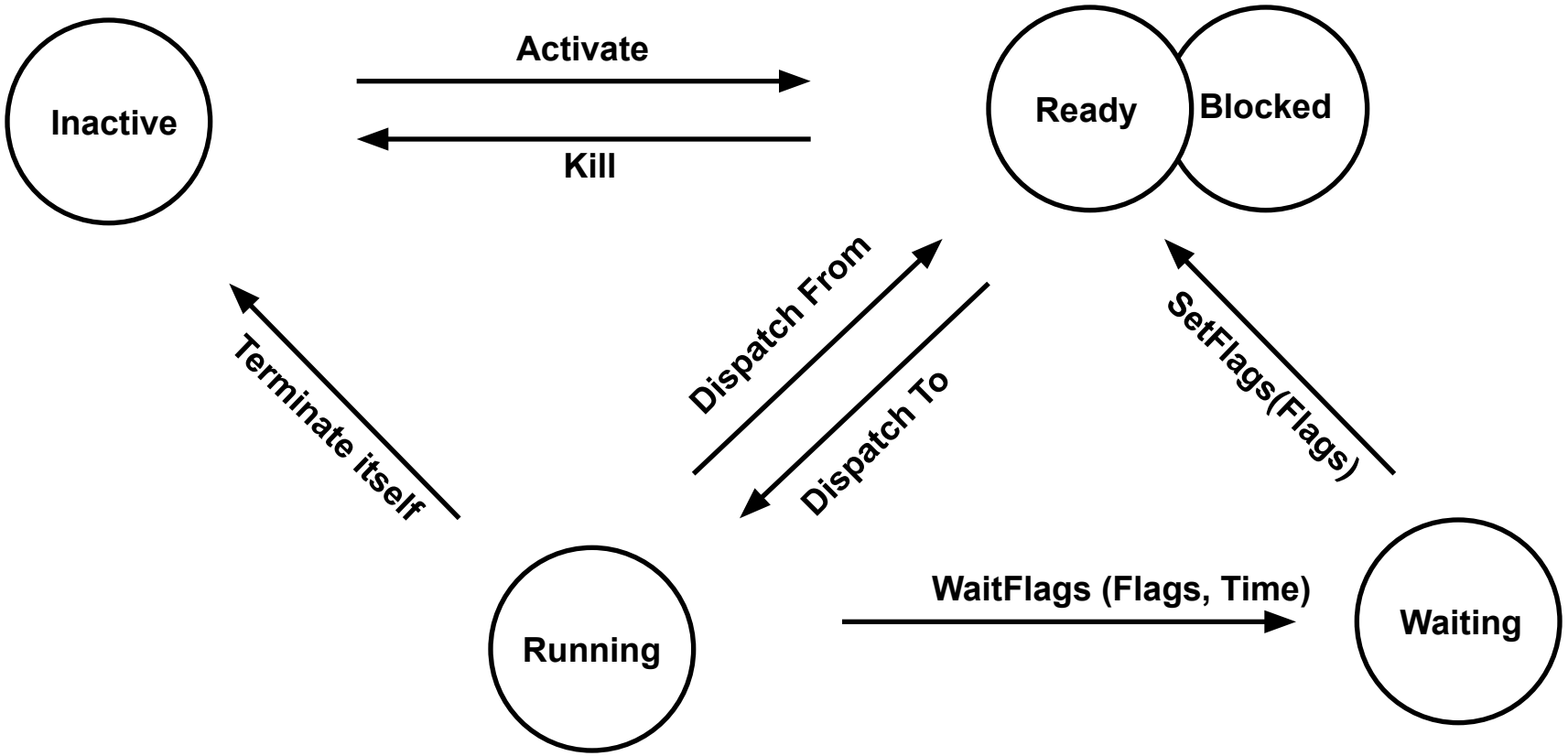




- $T_{activation\ latency}$
- $T_{termination\ latency}$
- $T_{context\ switch}$

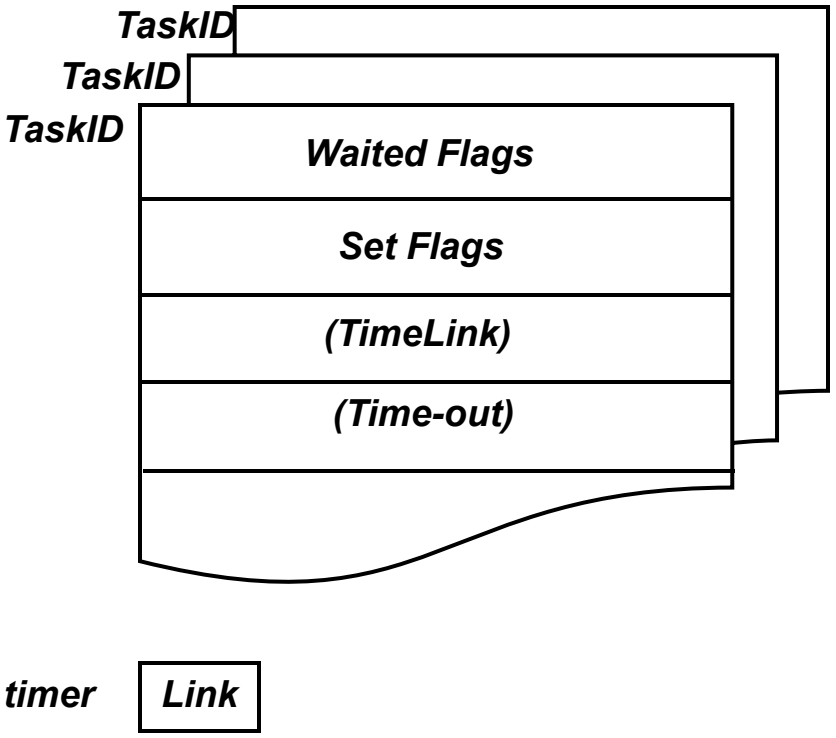
Базовые временные характеристики управления задачами

9. Ждущие задачи (1)



Waiting означает, что задача ждет какого-то события (флага или момента времени). В этом состоянии задача сохраняет свой контекст и локальные переменные для максимально быстрого перевода в состояние готовности (ready) и затем продолжения выполнения (running).

9. Ждущие задачи (2)



Дополнительные структуры данных для управления ждущими задачами

- **WaitFlags(Flags) - сравнивает состояние установленных флагов и аргумента. Если они совпадают, то задача продолжает выполнение. Если нужные флаги не установлены, задача переходит в состояние ожидания (running -> waiting), и вызывает диспетчер.**

- **SetFlags(Flags) - сравнивает состояние ожидаемых флагов и аргумента. Если они не совпадают, то задача остается в состоянии ожидания. Если ожидаемые флаги установлены, задача планируется (waiting -> ready), и вызывается диспетчер.**

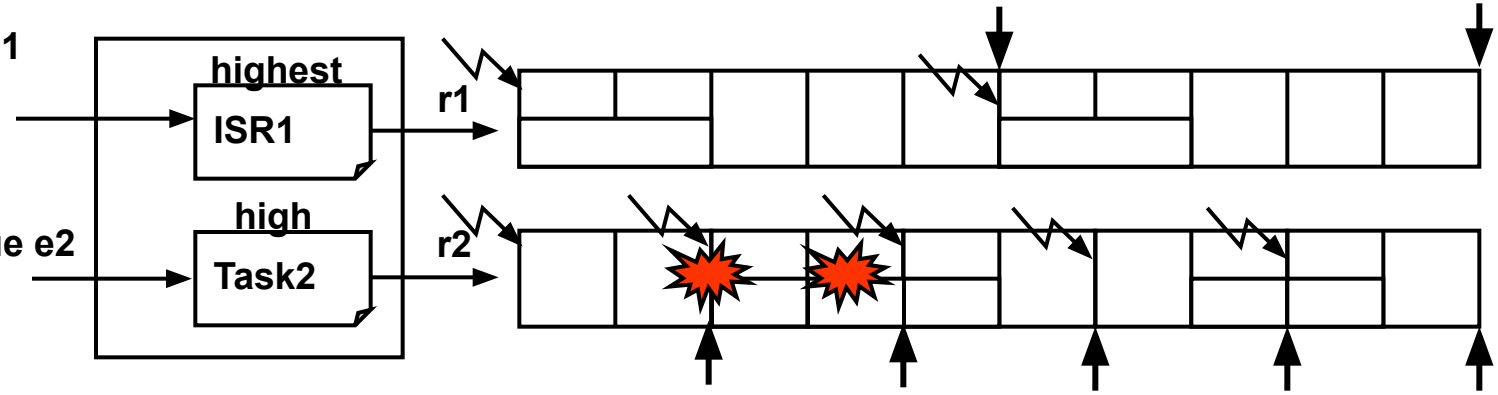
- **Если состояние ожидания связано с тайм-аутом, ожидающая задача находится в таймерной очереди.**

Дополнительные сервисы для управления ждущими задачами

10. Обслуживание прерываний (1)

Внешнее событие e1
- **T1 = 5**
- **D1 = 5**
- **C1 = 2**

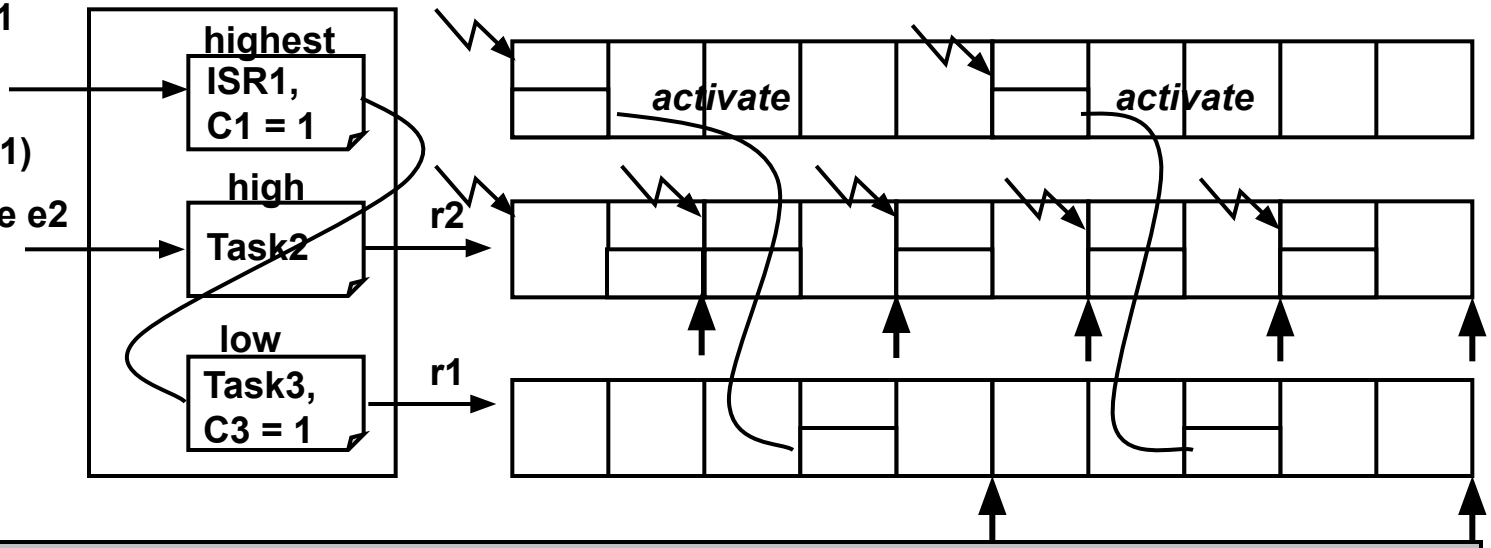
Внутреннее событие e2
- **T2 = 2**
- **D2 = 2**
- **C2 = 1**



Приложение неисполнимо, так как обработчик прерывания аппаратно вытесняет задачу с процессора, хотя и обрабатывает событие с большим сроком исполнения.

Внешнее событие e1
- **T1 = 5**
- **D1 = 5**
- **C = 2 = (C1=1) + (C3=1)**

Внутреннее событие e2
- **T2 = 2**
- **D2 = 2**
- **C2 = 1**



Обработка события распределена между обработчиком прерывания и задачей. Приложение стало исполнимым, потому что приоритетом задачи можно управлять.

10. Обслуживание прерываний (2)

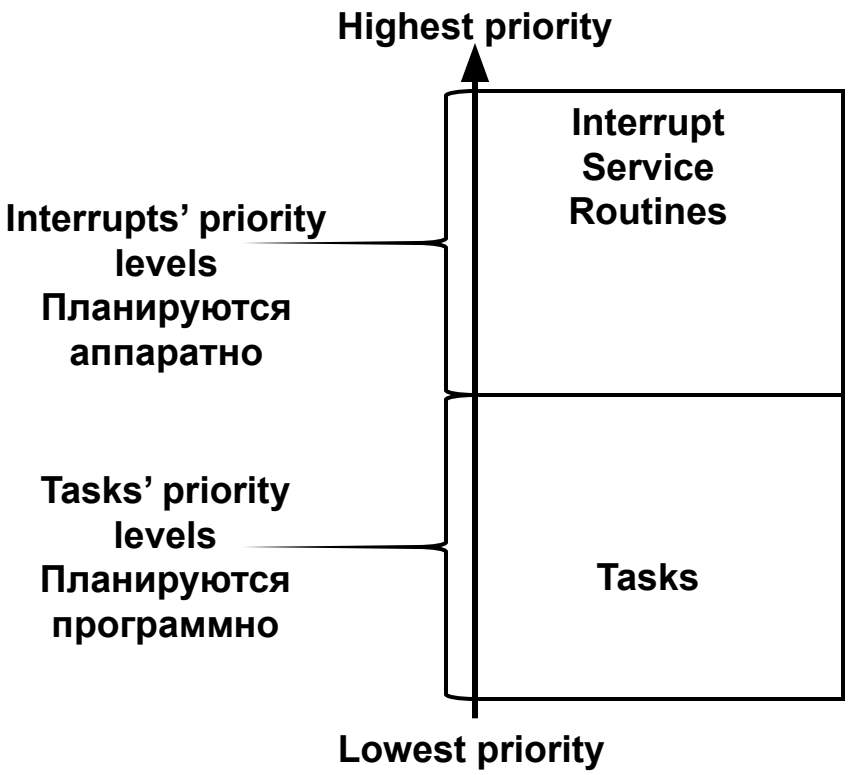


Схема приоритетов, не поддерживающая перекрытие приоритетов задач и обработчиков прерывания (например, OSEK/VDX OS)

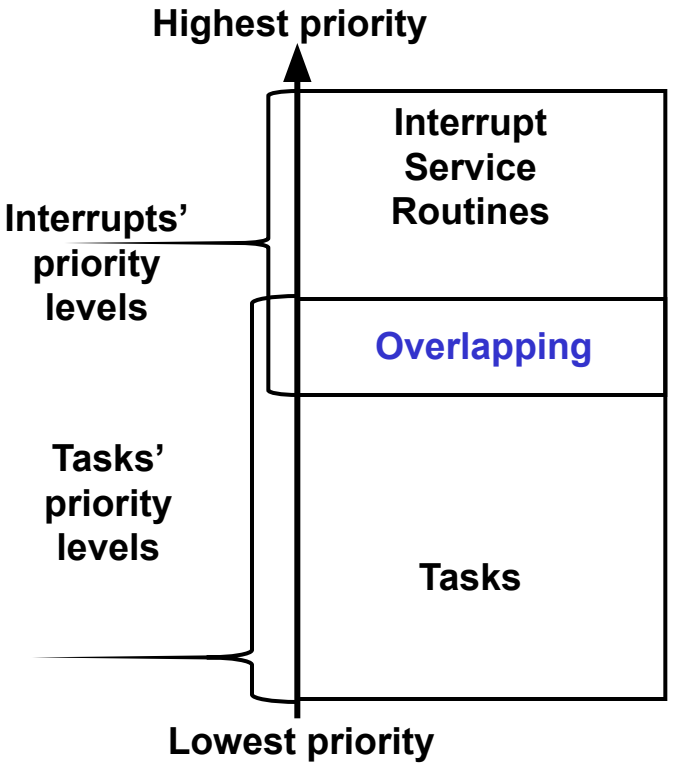


Схема приоритетов, поддерживающая перекрытие приоритетов задач и обработчиков прерывания (например, ERCOS)

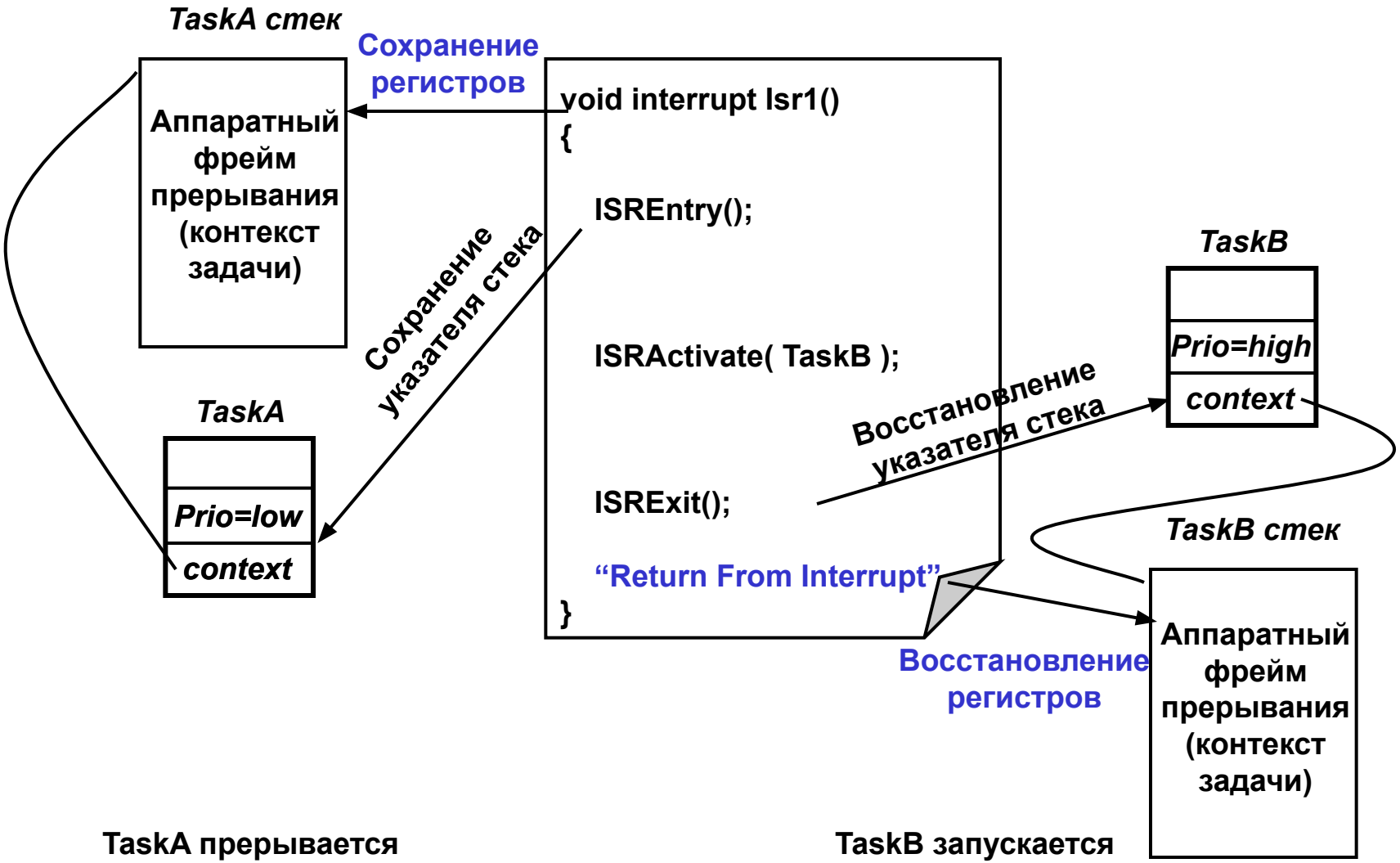
Назначение приоритетов в соответствии со сроками исполнения может приводить к перекрытию приоритетов задач и обработчиков прерываний.

10. Обслуживание прерываний (3)

Для того, чтобы обработка внешних и таймерных событий выполнялась в реальном времени, механизмы обслуживания прерываний должны удовлетворять следующим требованиям:

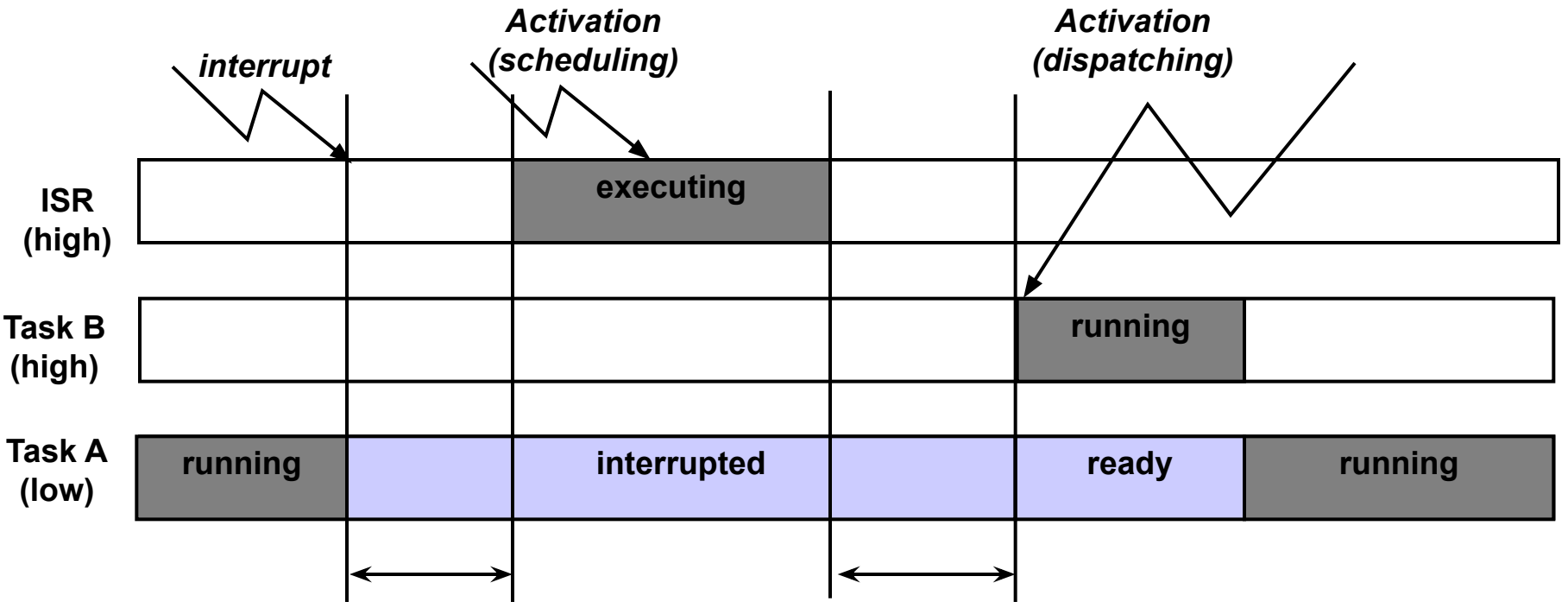
- **выполнение сервисов ОС- включая запуск задач - из обработчиков прерываний (позволяет перенести значительную часть обработки события из обработчика прерывания в задачу, улучшив исполнимость приложения)**
- **исключение инверсии приоритетов между обработчиками прерываний и задачами**
- **целостность данных и операций операционной системы при возникновении и обработки прерываний (атомарные операции ядра системы)**
- **высокая реактивность, т.е. минимально возможное время реакции системы на возникновение внешнего прерывания (interrupt latency)**
- **обработка нескольких десятков источников прерываний**

10. Обслуживание прерываний (4)



Переключение контекста задач при запуске высокоприоритетной задачи из обработчика прерывания.

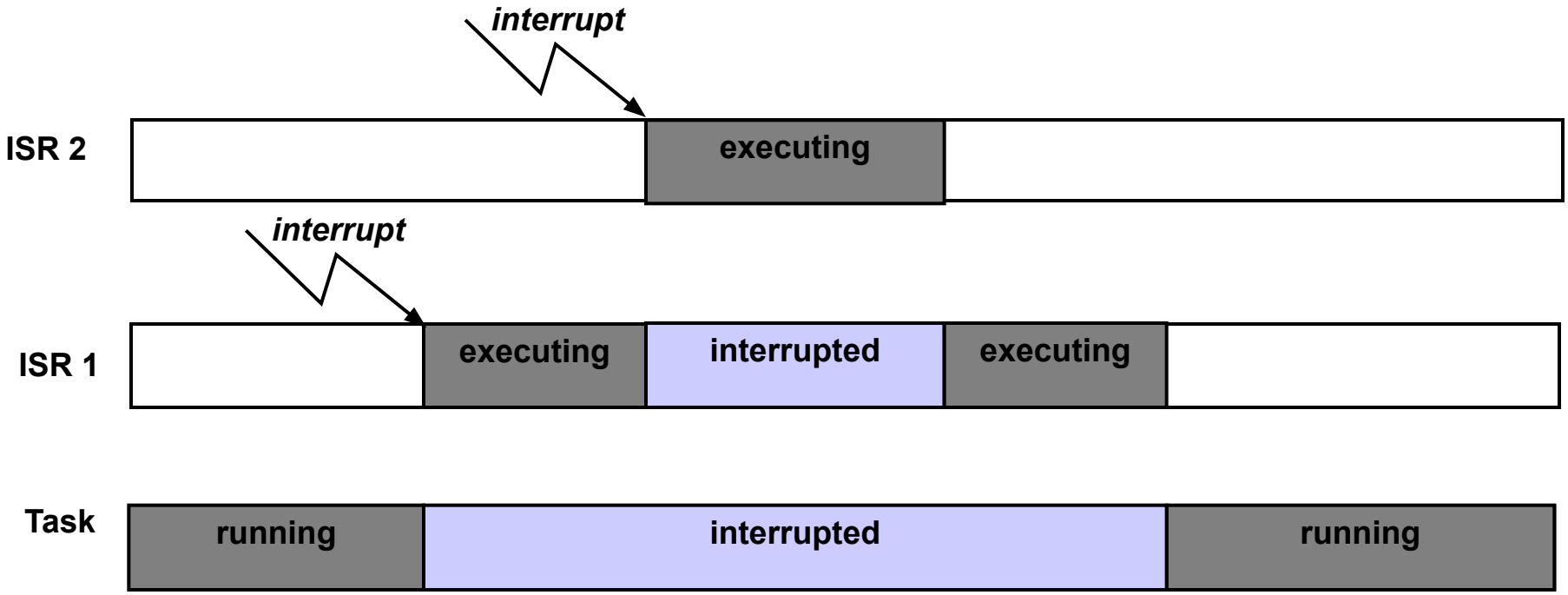
10. Обслуживание прерываний (5)



- $T_{interrupt\ entry\ latency}$
- $T_{interrupt\ switch\ latency}$

Базовые временные характеристики обслуживания прерываний

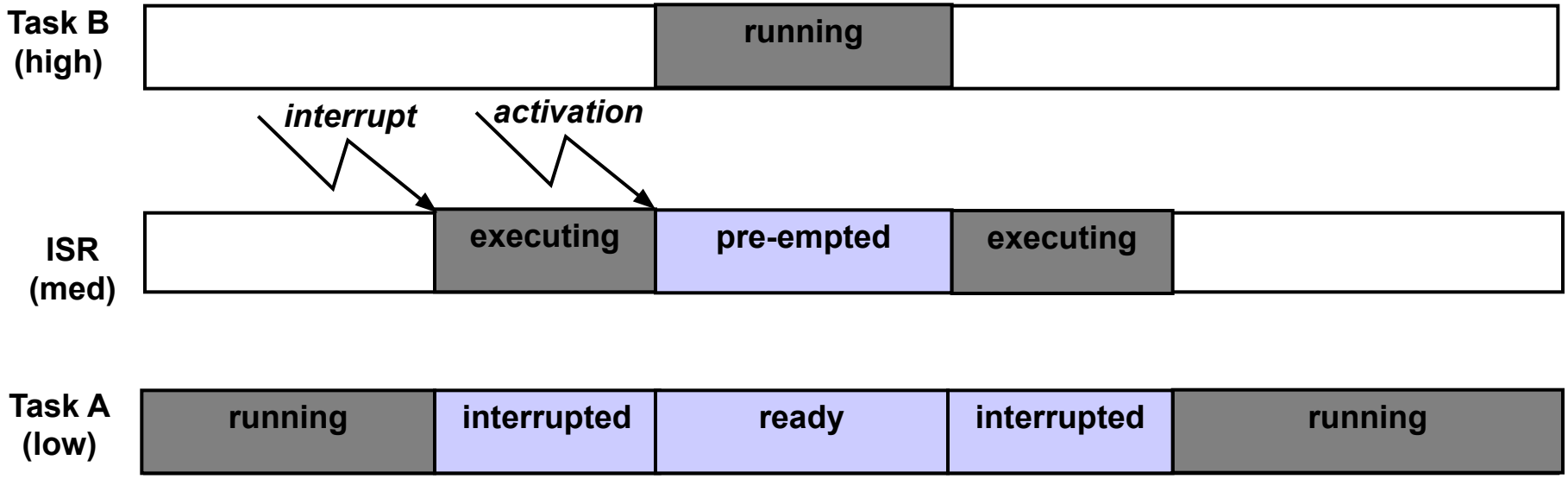
10.1 Вложенные прерывания



Вложенные прерывания улучшают возможность системы по одновременной обработки нескольких источников прерываний.
Вложенные прерывания обычно поддерживаются операционной системой в случае, если аппаратура имеет многоуровневую приоритетную систему прерываний (Motorola 68K, PowerPC, Siemens C167), так как при этом фактически выполняется приоритетное вытесняющее планирование обработчиков прерываний (т.е. контроллер прерываний работает как аппаратный планировщик и диспетчер).

Вложенные обработчики без поддержки приоритетов ухудшают исполнимость приложений, так как фактически выполняются в режиме “случайного” вытеснения.

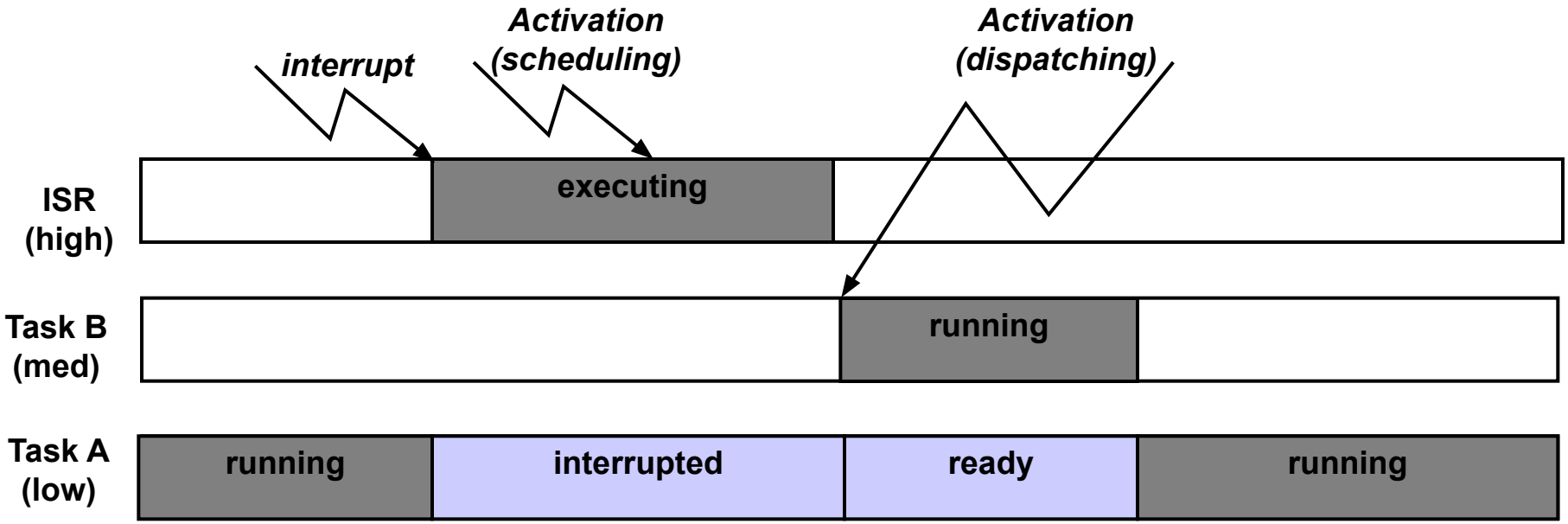
10.2 Немедленное выполнение сервиса ОС



Немедленное выполнение сервиса ОС необходимо в ОС, где поддерживается схема приоритетов, которая позволяет задачам иметь приоритет выше, чем приоритет обработчика прерывания (например, ERCOS) - потому что гарантирует отсутствие инверсии приоритетов. Задача, планируемая из обработчика прерываний, может быть приоритетнее самого обработчика, и должна немедленно вытеснить его с процессора.

Обычно не поддерживается во встроенных ОС из-за сравнительно сложной реализации.

10.3 Задержанное выполнение сервисов ОС (1)



Задержанное выполнение сервиса ОС обычно означает, что планирование задачи выполняется во время выполнения обработчика прерывания, а переключение (диспетчирование) задачи выполняется по завершению обработчика прерывания.

Такая схема поддерживается многими ОС (например, OSEK/VDX OS), так как соответствует схеме приоритетов “обработчик прерывания имеет более высокий приоритет, чем задача”. Конечно, правильное назначение приоритетов - ответственность разработчика приложения.

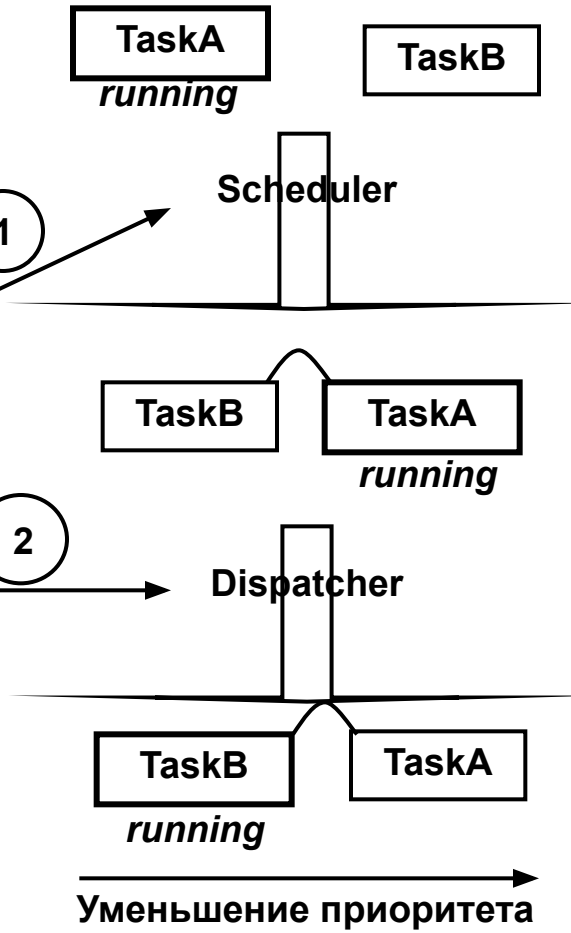
10.3 Задержанное выполнение сервисов ОС (2)

Планирование в случае задержанного выполнения сервиса состоит из двух раздельных шагов:

1. Собственно планирование, т.е. составление расписания.

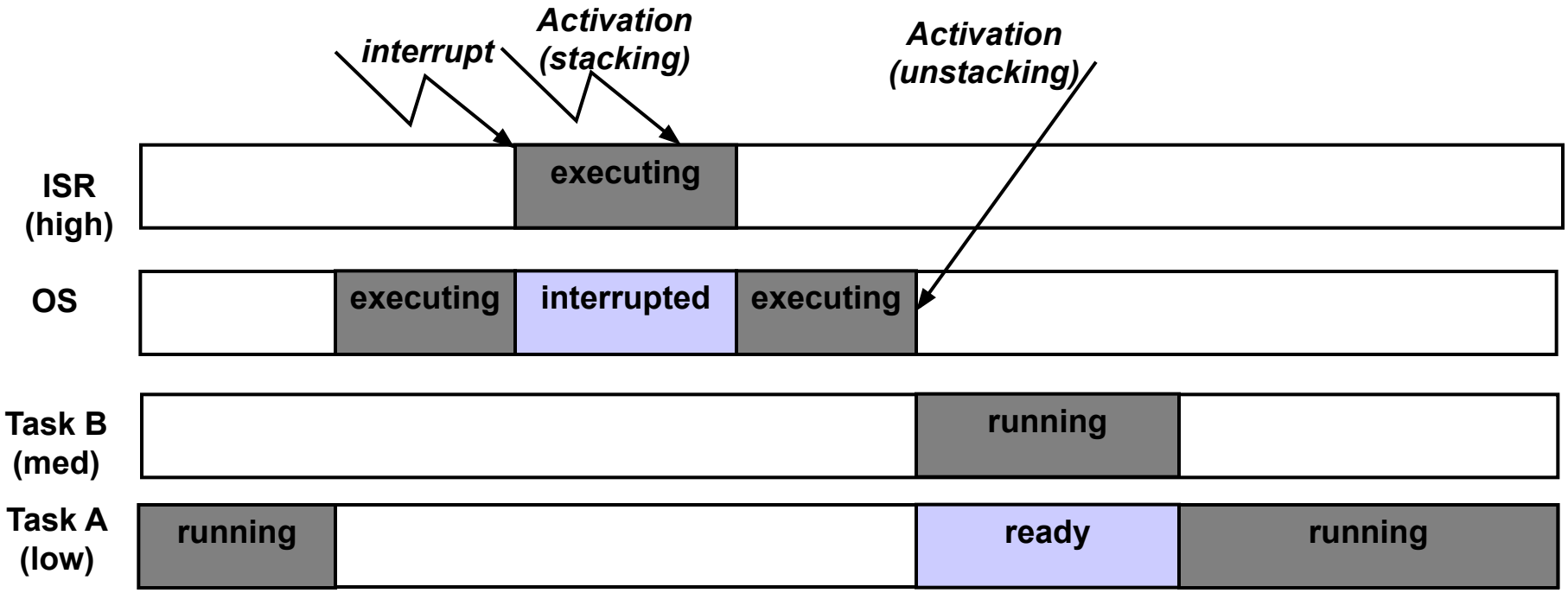
2. Диспетчирование (dispatching) - выбор задачи из списка и назначения ей процессора (переключение контекста). Выполняется по завершению обработчика прерывания!

```
void interrupt Isr1()
{
   ISREntry();

   ISRActivate( TaskB );

   ISRExit();

}
```



В случае обработки нескольких вложенных обработчиков прерываний (т.е. прерывающих друг друга) диспетчирование выполняется по завершению самого внешнего обработчика прерывания.

10.4 Отложенное выполнение сервисов ОС



Отложенное выполнение сервиса ОС обычно означает, что вызов сервиса ОС откладывается до момента окончания обработчика прерывания или завершения прерванного выполнения сервиса ОС. По окончании выполнения обработчика выполняется вызов сервиса.

Такая схема позволяет разрешать прерывания во время выполнения сервиса ОС, уменьшая interrupt latency, и поддерживается многими встроенными ОСРВ (например, RTXC).

10.5 Ограничение сервисов ОС

Многие ОС не позволяют выполнять все сервисы ядра из обработчиков прерывания. Например, разрешается только переводить задачу из ждущего состояния в готовое (SetFlags). Это делается для того, чтобы обеспечить быстрое выполнение обработчиков прерывания за счет уменьшения объемов обработки сервиса (например, RTXC разрешает только одну операцию в обработчиках прерываний).

10.6 Атомарные операции в ОС

```
void Activate( TASK task )
{
  DisableInterrupts();

  Schedule( task );


  Dispatch();

  EnableInterrupts();

}
```

⇐

Обычно атомарные (неделимые) операции в ядре выполняются с помощью запрещения / разрешения прерываний. Это сделано для того, чтобы обработчики прерываний не могли нарушить целостность данных ядра - например, списков планировщика.

⇒

В то же время для улучшения реактивности ядра желательно уменьшить время исполнения атомарных операций - например, разделив их на две операции.

```
void Activate( TASK task )
{

  DisableInterrupts();
  Schedule( task );
  EnableInterrupts();

  DisableInterrupts();
  Dispatch();
  EnableInterrupts();

}
```

11. Разделяемые ресурсы (семафоры)

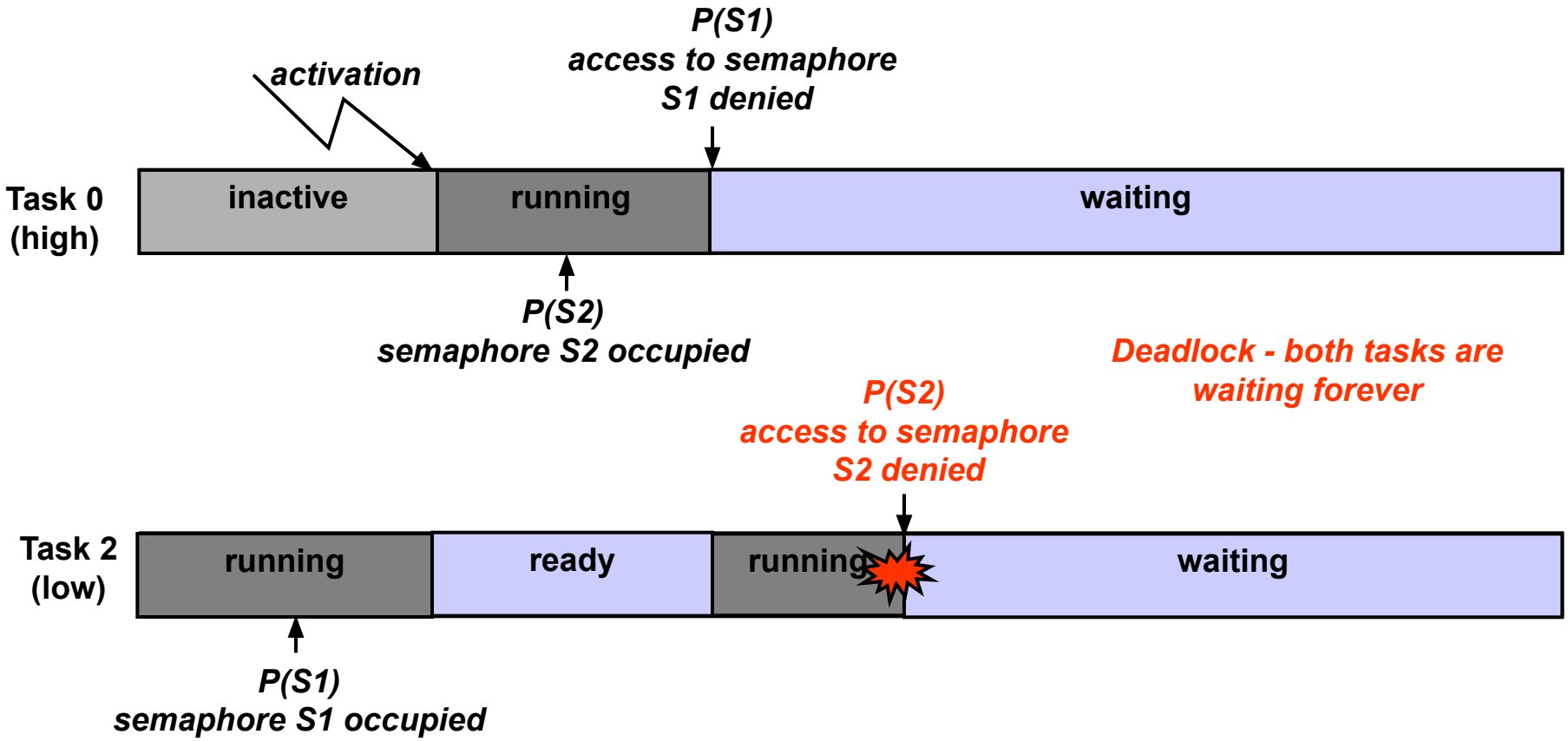
```
void TaskA_Entry(void)
{
  /* Entry Critical Section */
  RequestResource( ResX );
  /* write shared variable */
  X = 1;

  /* Exit Critical Section */
  ReleaseResource( ResX );
}
```

Доступ к критической секции управляется семафором ресурса ResX

```
void TaskB_Entry(void)
{
  /* Entry Critical Section */
  RequestResource( ResX );
  /* read shared variable */
  if( X == 1 )  Y = 0;

  /* Exit Critical Section */
  ReleaseResource( ResX );
}
```

Критическая секция кода для доступа к разделяемым переменным, устройствам ввода-вывода, и т.п.

11.1 P/V семафоры и проблемы (1)



Тупик при взаимном захвате двух P/V семафоров

11.1 P/V семафоры и проблемы (2)

Исправление дефектов на Марсе.

4 Июля 1997 года Mars Pathfinder приземлился на Марсе.
Ключевым компонентом системы была ОС реального времени VxWorks® (WindRiver).

Для сбора данных в реальном времени использовались камеры, управление которыми программно синхронизировалось. Вскоре после начала сбора данных система стала переходить в состояние сброса (reset).

Оказалось, что синхронизация задач, осуществляемая с помощью разделяемых ресурсов, приводила к задержкам, которые превышали тайм-аут, и система производила сброс. Причиной задержек являлась затяжная инверсия приоритетов (unbounded priority inversion).

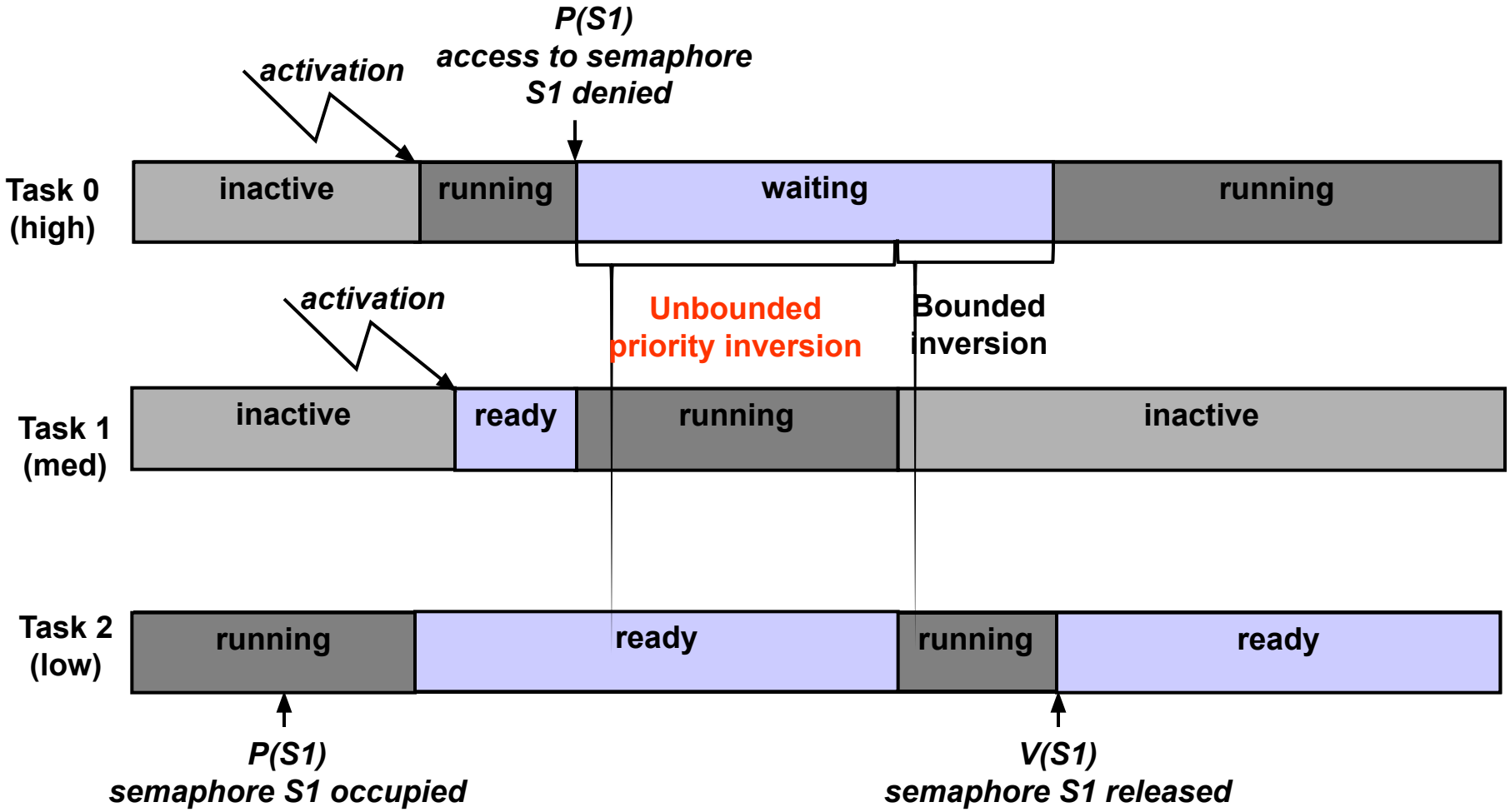
Высоко-приоритетная задача с коротким сроком исполнения через pipe обменивалась данными с низкоприоритетной задачей. Pipe была реализована с помощью семафора.

VxWorks поддерживает протокол наследования приоритетов для разделяемых ресурсов как опцию, но эта опция не была выбрана специалистами NASA. Теория стала реальностью.

Решение: специалисты NASA послали команду "установить бит" в глобальной переменной для использования протокола наследования приоритетов. После этого система работала без сбоев до истечения срока службы батарей.

Полная версия http://www.wrs.com/products/html/jpl.html
Комментарий http://www.embedded.com/2000/0006/0006feat1.htm

11.1 P/V семафоры и проблемы (3)



Затяжная инверсия приоритетов (Unbounded Priority Inversion)

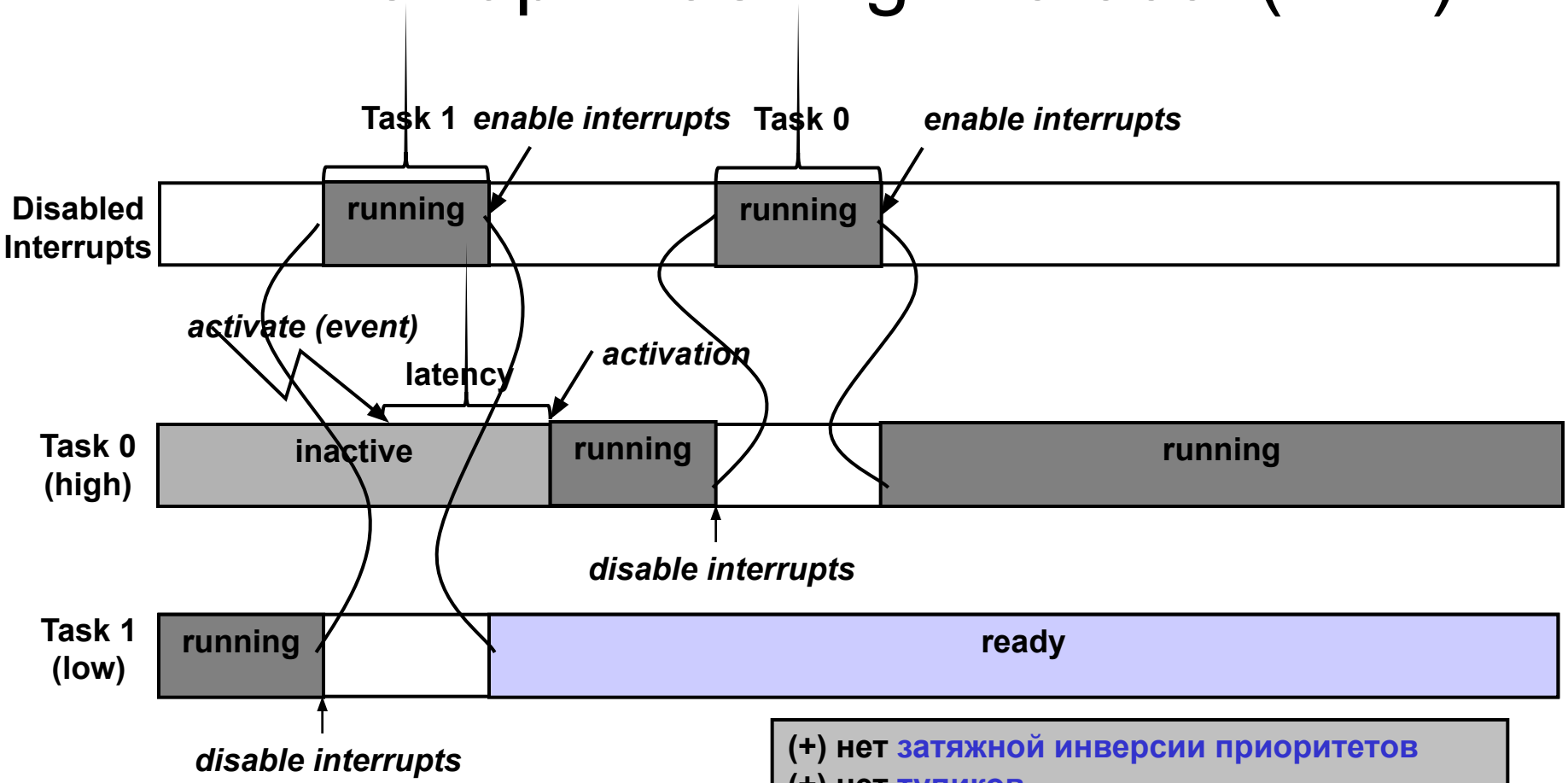
11.1 P/V семафоры и проблемы (4)

Для того, чтобы семафоры могли использоваться во встроенных приложениях реального времени, механизм поддержки семафоров должен удовлетворять следующим требованиям:

- **предотвращение тупиков**
- **предотвращение затяжной инверсии приоритетов (ограничение инверсии по времени)**
- **минимизация влияния на задачи, не разделяющих критическую секцию**
- **минимально возможное потребление ресурсов - памяти и процессорного времени**

Механизм \ Требование	Тупики	Затяжная инверсия приоритетов	Влияние на «другие» задачи	Потребление ресурсов
Протокол маскирования прерываний (IMP)	Предотвращает	Предотвращает	Значительное	Минимальное
Протокол наследования приоритетов (PIP)	Не предотвращает	Предотвращает	Незначительное	Незначительное
Протокол потолка приоритетов (PCP)	Предотвращает	Предотвращает	Умеренное	Значительное (сложный)
Протокол высшего приоритета (HLP)	Предотвращает	Предотвращает	Умеренное	Незначительное

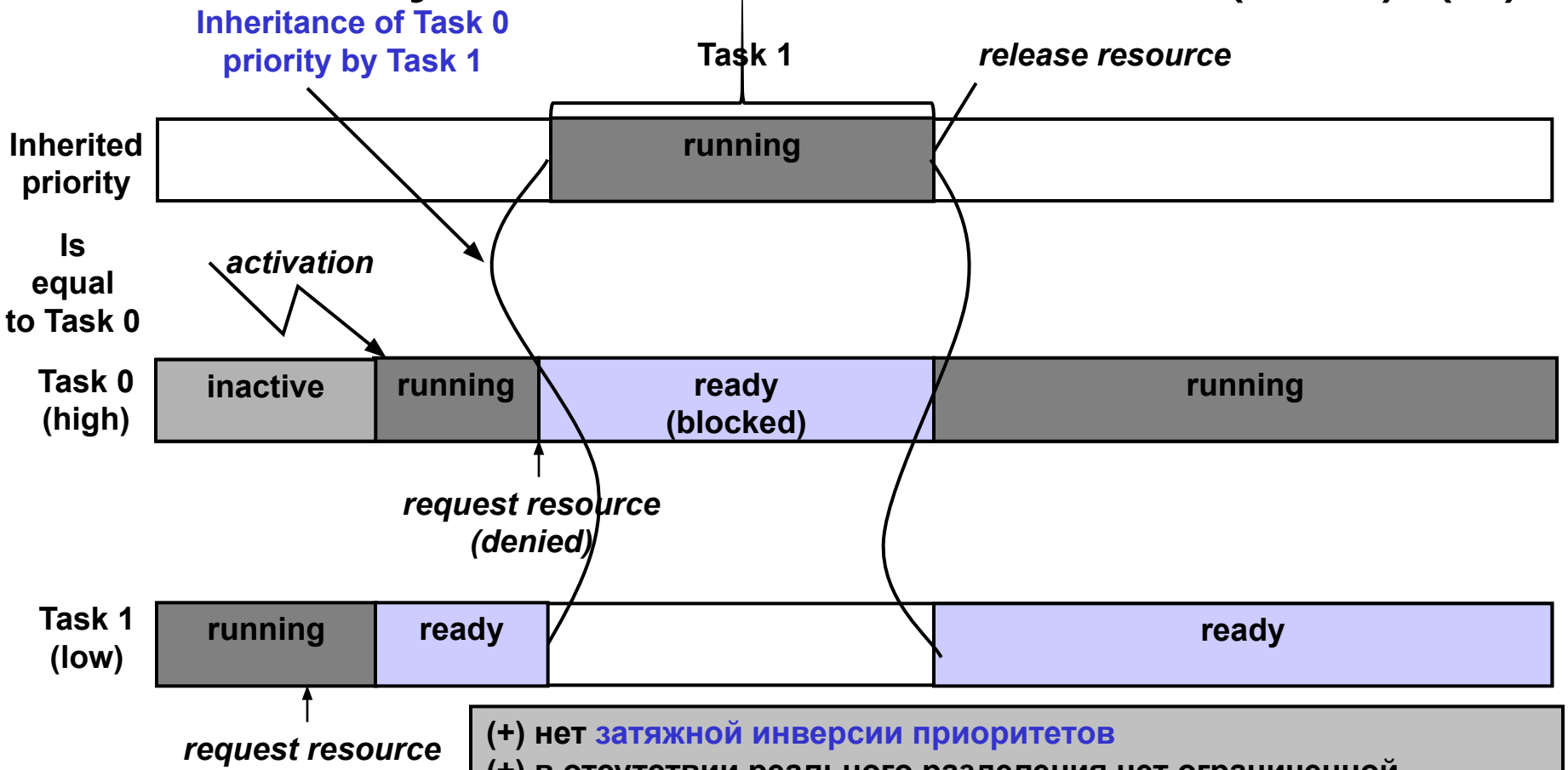
11.2 Interrupt Masking Protocol (IMP)



В некоторых операционных системах вместо запрещения прерывания используется запрещение вытеснения задач (disable preemption).

(+) нет затяжной инверсии приоритетов
(+) нет тупиков
(+) простота реализации
(+) минимальные накладные расходы
(-) ухудшается реакция на возникновение прерывания (увеличивается latency)
(-) критическая секция распространяется на все задачи!

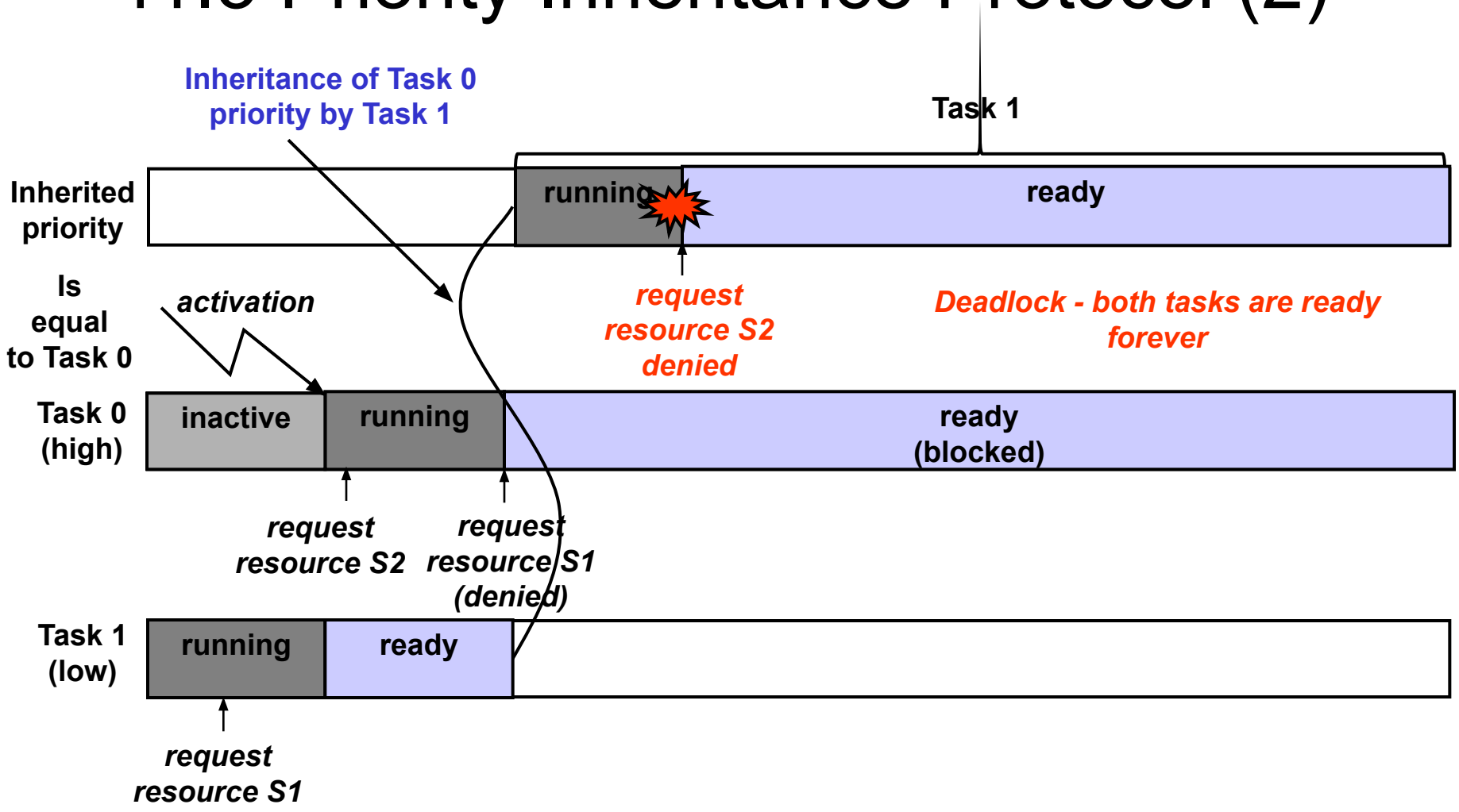
11.3 Priority Inheritance Protocol (PIP) (1)



Release resource вызывает диспетчирование задач!

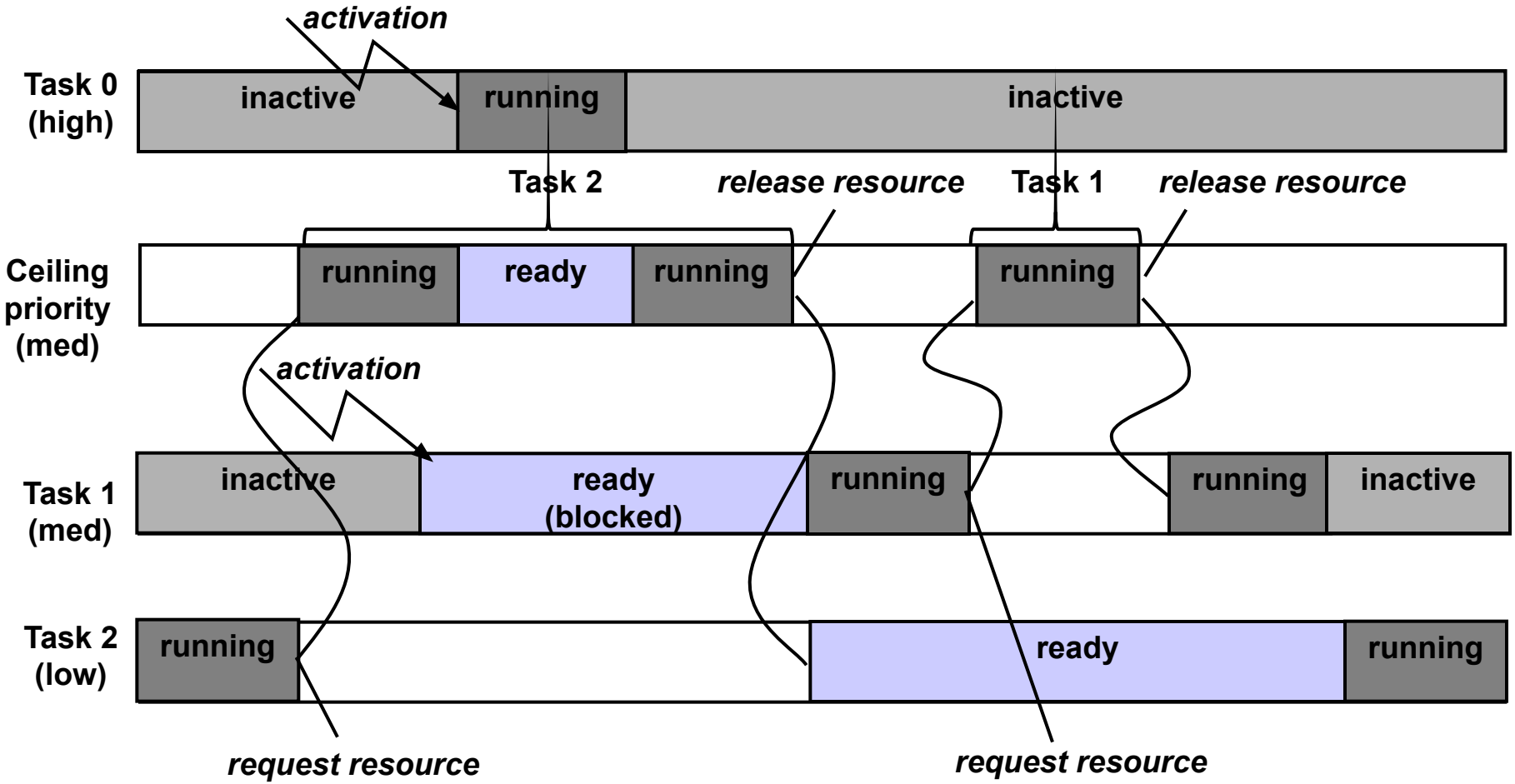
(+) нет затяжной инверсии приоритетов
(+) в отсутствии реального разделения нет ограниченной инверсии приоритетов (не очень важно, так как при анализе всегда интересует худший случай, а он подразумевает разделение)
(+) не требуется вычислений до выполнения (т.е. нет off-line обработки).
(-) тупиковые ситуации возможны

11.3 Priority Inheritance Protocol (2)



Тупик при взаимном захвате двух семафоров (ресурсов) в случае протокола наследования приоритетов

11.4 Highest Locker Protocol (HLP) (1)

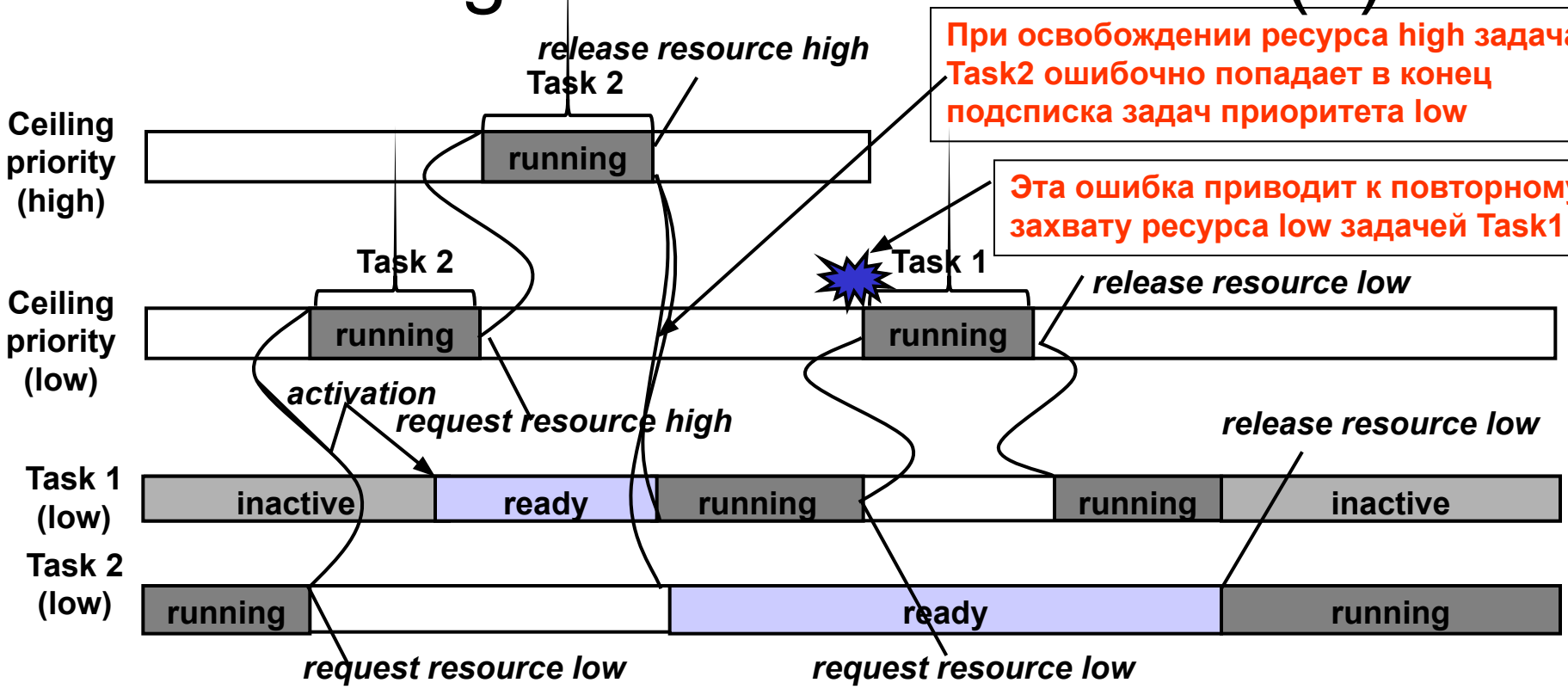


(+) нет затяжной инверсии приоритетов
(+) нет тупиковых ситуаций
(+) относительная простота реализации
(+) нет влияет на более приоритетные задачи

(-) требуется вычисление приоритетов до выполнения (т.е. off-line обработка).
(-) ограниченная инверсия приоритетов

При освобождении ресурса должна выполняться диспетчеризация задач!

11.4 Highest Locker Protocol (2)



Ошибка реализации HLP при работе с вложенными ресурсами

Для корректной реализации протокола при захвате и освобождении вложенных ресурсов необходимо, чтобы при освобождении ресурса «высокого» приоритета задача помещалась в голову подсписка задач «низкого» приоритета - для того, чтобы она не вытеснялась задачей «низкого» приоритета, которая может захватить неосвобожденный ресурс низкого приоритета!

HLP имеет много синонимов: Priority Protect Protocol (POSIX), Priority Ceiling Emulation Protocol, Immediate Priority Ceiling Protocol, OSEK Priority Ceiling Protocol.

Но: Highest Locker Protocol - это не то же самое, что Priority Ceiling Protocol!

12. Техника назначения приоритетов

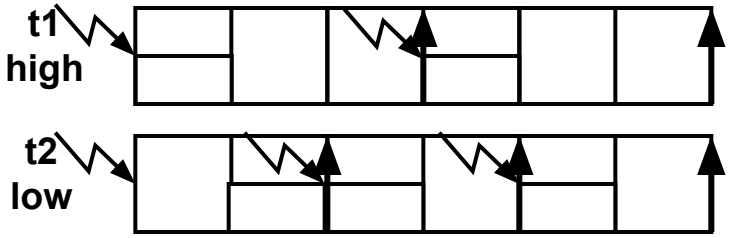
Задачи применения техники назначения приоритетов:
1. Обеспечить исполнимость (schedulability) приложения на протяжении всего времени выполнения (до нескольких лет!).
2. Обеспечить максимальное использование (utilizing) процессорного времени (т.е. экономию вычислительных ресурсов).
3. Рассчитать действительное значение сроков исполнения.

Существуют различные методы, дающие оптимальные результаты для разных наборов задач.

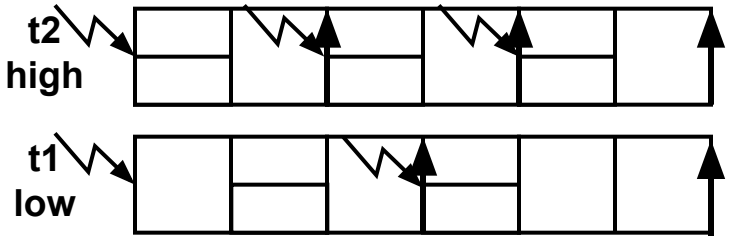
12.1 Rate Monotonic Algorithm (1)

- **Rate Monotonic Scheduling (RMS) оптимален для независимых периодических задач имеющих срок исполнения равным периоду (т.е. задача должна завершиться в пределах периода исполнения, D=T).**
- **Задачам с меньшими периодами назначается больший приоритет.**
- **Теорема Liu и Layland (1973): независимые периодические задачи, планируемые с помощью RMS, всегда удовлетворяют срокам исполнения, если сумма загрузок процессора задачами (C/T) не превышает предела использования (utilization bound).**

t1: T1=D1=3, C1=1; t2: T2=D2=2, C2=1;
U = 1/3 + 1/2 = 0.83(3)

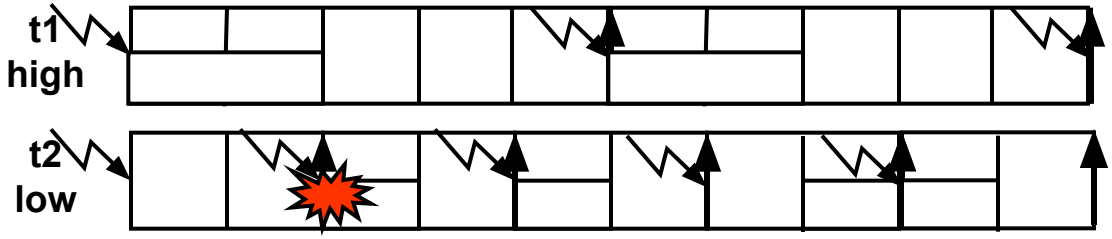


He-RMS: приложение исполнимо

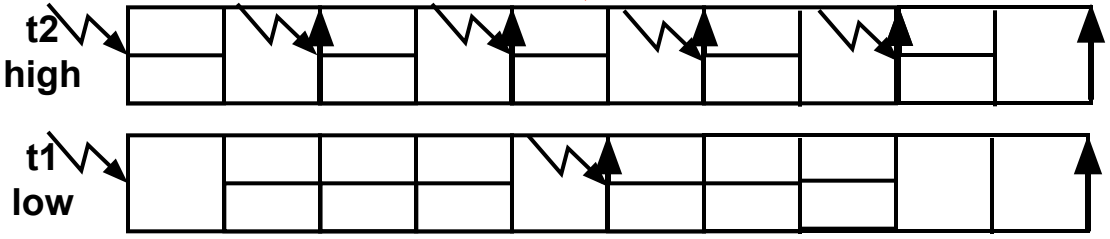


RMS: приложение исполнимо

t1: T1=D1=5, C1=2; t2: T2=D2=2, C2=1;
U = 2/5 + 1/2 = 0.9



He-RMS: приложение неисполнимо!

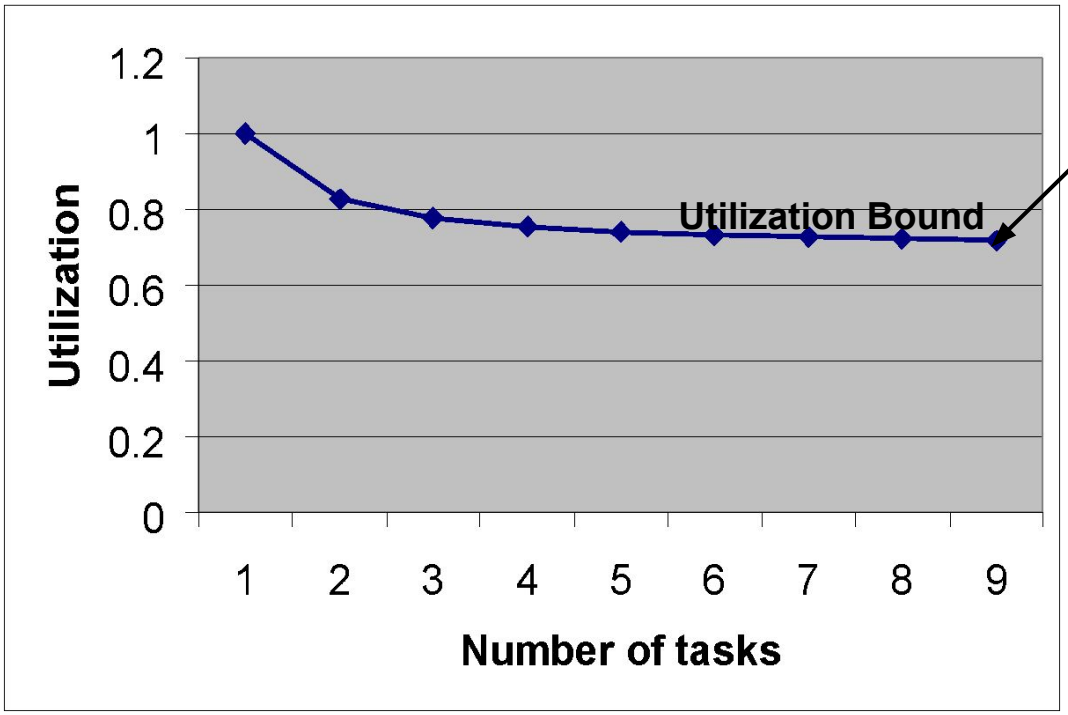


RMS: приложение исполнимо

RMS лучше, чем не-RMS!

нарушение срока исполнения

12.1 Rate Monotonic Algorithm (2)



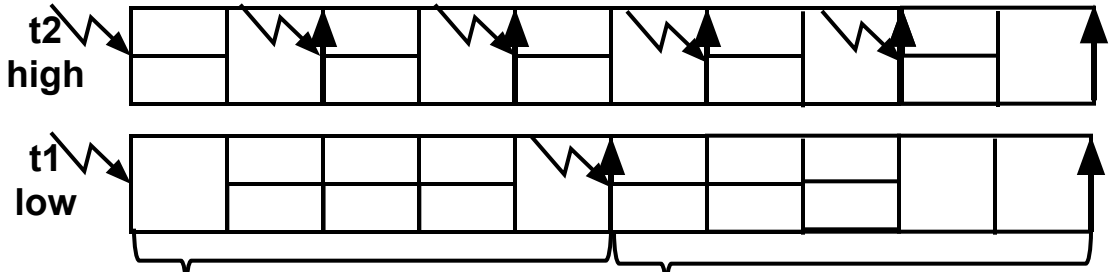
U(9) = 0,720

(+) простая реализация (назначение фиксированных приоритетов)
(-) плохое использование процессора
(-) ОС должна поддерживать много уровней приоритетов (на практике достаточно 32 уровня)

t1: T1=D1=5, C1=2; t2: T2=D2=2, C2=1;
U = 2/5 + 1/2 = 0.9

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \ldots + \frac{C_n}{T_n} <= n\,(2^{1/n} - 1)$$

U(∞) = 0,693 = ln(2)



Для t1 доступно только 40%!

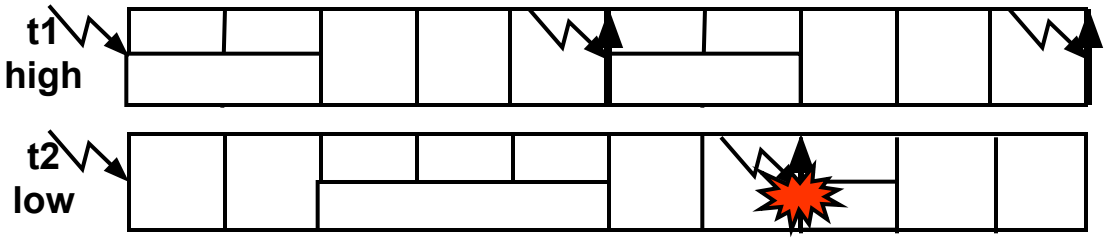
Для t1 доступно 60%

В среднем задаче t1 доступно 50%, но с учетом срока исполнения - только 40%

12.2 Earliest Deadline First

- **Earliest Deadline First оптимален для независимых периодических задач имеющих срок исполнения равным периоду (т.е. задача должна завершиться в пределах периода исполнения, D=T).**
- **EDF используется для приложений, для которых оптимален RMA, но выполняется динамически (и позволяет добиться 100% использования процессора).**
- **Принцип работы: после любого события в системе, которое изменяет набор задач в состоянии готовности (ready), диспетчер назначает высший приоритет задаче с ближайшим сроком исполнения (earliest deadline).**
- **Liu и Layland (1973): независимые периодические задачи, планируемые с помощью EDF, всегда удовлетворяют срокам исполнения, если сумма загрузок (C/T) не превышает 100%.**

t1: T1=D1=5, C1=2; t2: T2=D2=7, C2=4; U=2/5+4/7= 0.9714

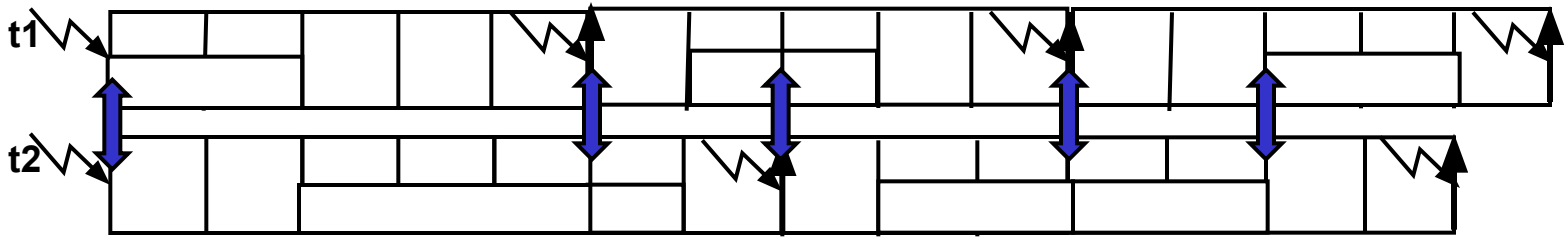


RMS: приложение неисполнимо

(+) полное использование процессора
(-) сложная реализация (динамическое назначение приоритетов)

перепланирование

нарушение срока исполнения

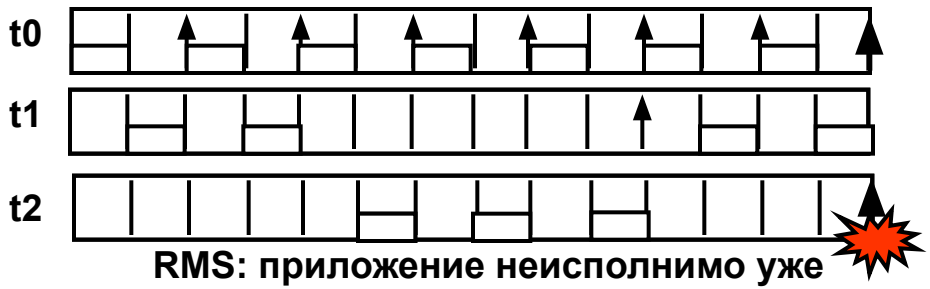


EDF: приложение исполнимо!

13. Preemptive Threshold Scheduling

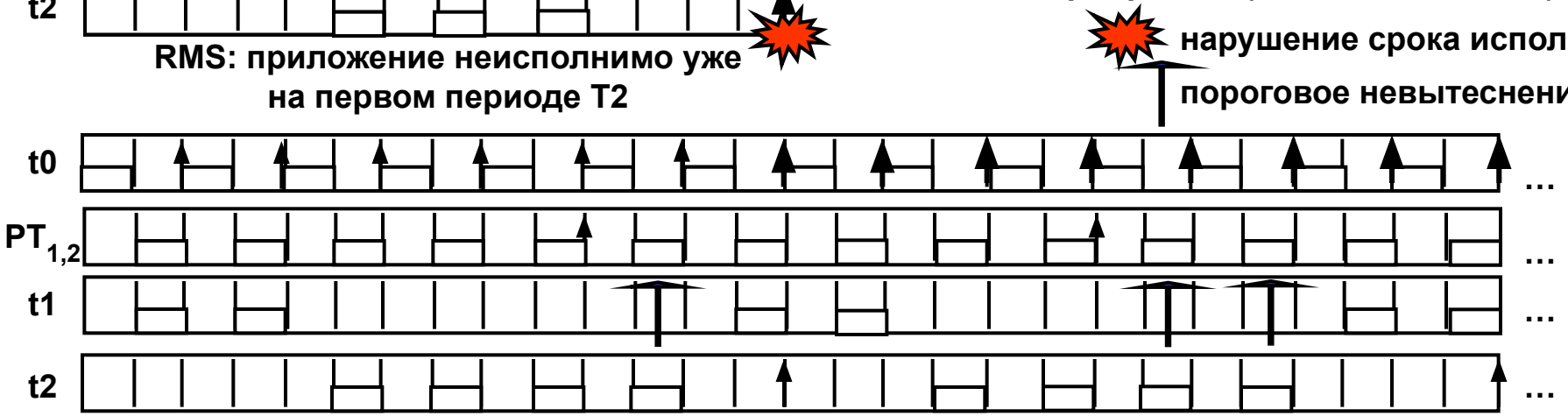
- Можно заметить, что пример для EDF также исполним при невытесняющем планировании.
- Следовательно, можно добиться улучшения планируемости, группируя задачи во взаимо-невытесняемые группы. Такие группы образуются с помощью назначения задачам группы порогового приоритета времени выполнения.
- Задача из группы планируется с исходным приоритетом, а выполняется с пороговым приоритетом, исключая вытеснение задачи другой задачей группы.
- Этот метод называется Preemptive Threshold™, и впервые был использован в ОСРВ ThreadX.
- В реализации метод подобен HLP с критической секцией на все тело задачи.

t0: T0=D0=2, C1=1; t1: T1=D1=10, C1=2; t2: T2=D2=14, C2=4; U=1/2+2/10+4/14= 0.9857



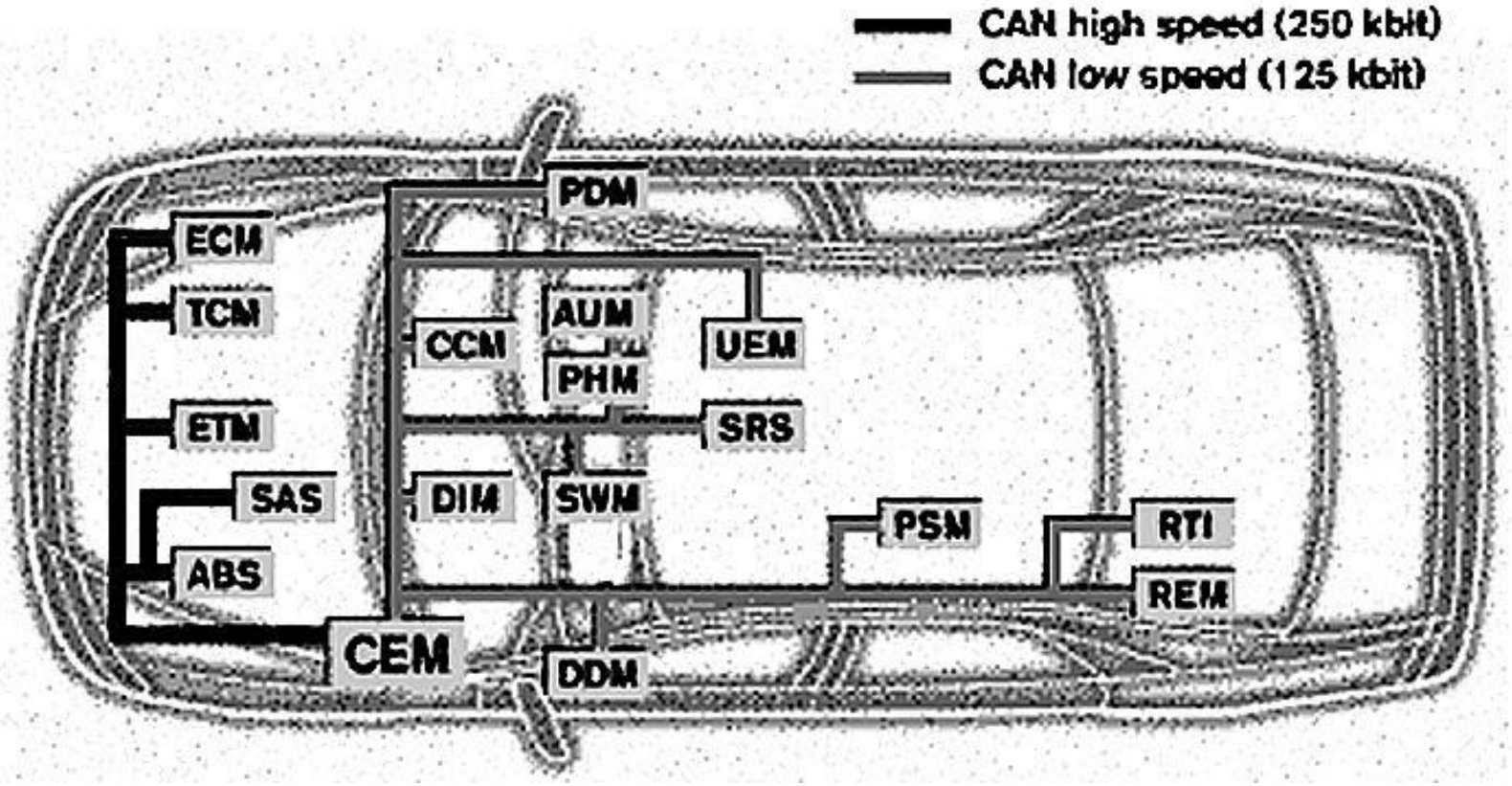
RMS: приложение неисполнимо уже на первом периоде T2

(+) более полное использование процессора, чем в RMS
(+) реализация проще, чем EDF
(-) необходимость вычисления порогового значения приоритета (NP-полная задача)



PTS для группы t1 и t2: приложение исполнимо! (показано только для двух периодов T2)

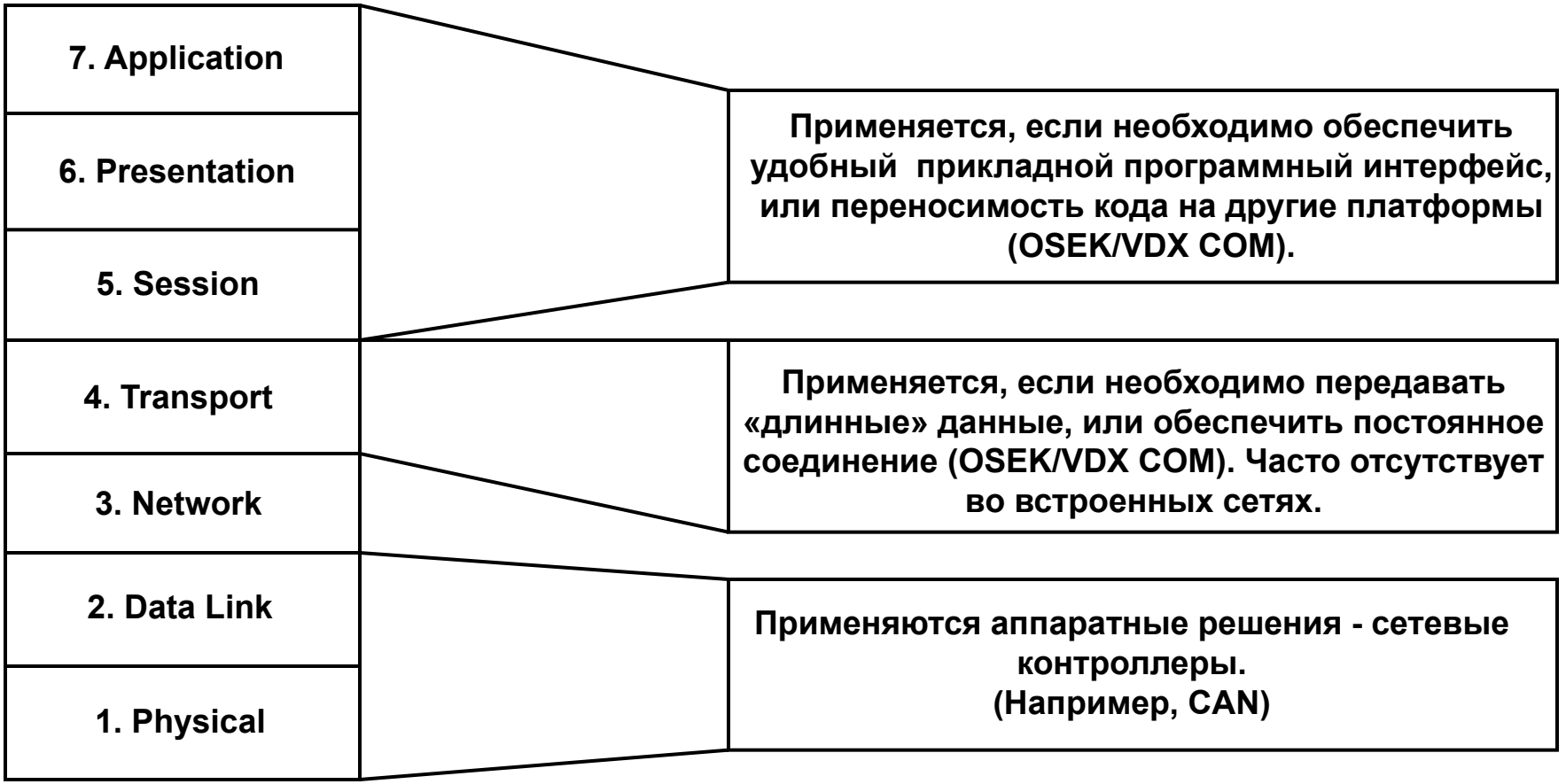
14. Сетевая передача данных (1)



Volvo S80 имеет две сети передачи данных: высокоскоростную для управления двигателем и подвеской, и низкоскоростную для кузовной электроники. Обе CAN сети соединяются шлюзом, и объединяют 18 узлов. Кроме того, в машине есть еще четыре низкоскоростных подсети на основе последовательного интерфейса типа RS232, включенного по схеме шины.

Полная версия: http://www.tech2.volvo.se/reportage/9811electrical/main.htm
http://www.tech2.volvo.se/reportage/9811electrical/anim.htm

14. Сетевая передача данных (2)



ISO/OSI Reference Model

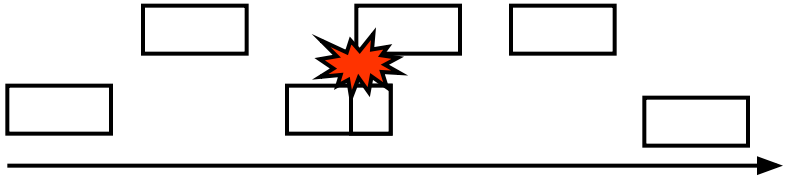
Реальные встроенные сетевые системы

Встроенные системы обычно не поддерживают все уровни сетевой модели, так как требуется надежная быстрая передача «коротких» данных и минимальные накладные расходы.

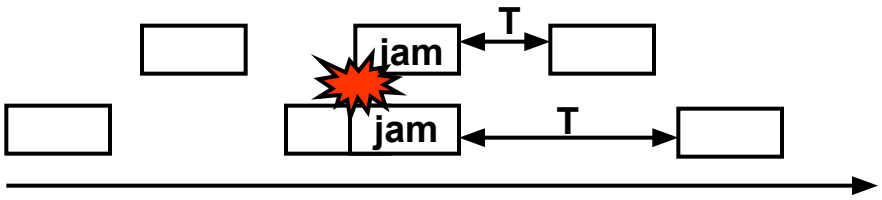
14.1 Физический уровень и уровень доступа на примере CAN (1)

CAN - Controller Area Network - протокол, предназначенный для транспортных средств. Разработка была начата R.Bosch GmbH, Germany в середине 1980-х.

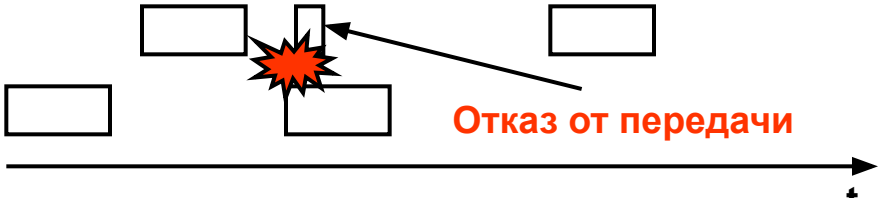
CAN реализует метод доступа CSMA/CD+CR (Carrier Sense, Multiple Access/Collision Detection + Collision Resolution), т.е. CAN корректно разрешает конфликты множественного доступа.



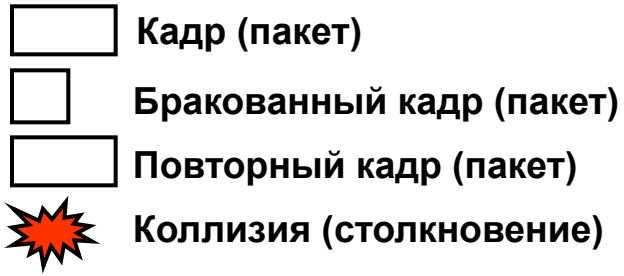
CSMA («ALOHA», ~1950), нет обнаружения коллизий; повтор обеспечивается транспортными протоколами



CSMA/CD (Ethernet, ~1980), есть обнаружение коллизий, нет разрешения коллизий; повтор через псевдо-случайный интервал времени

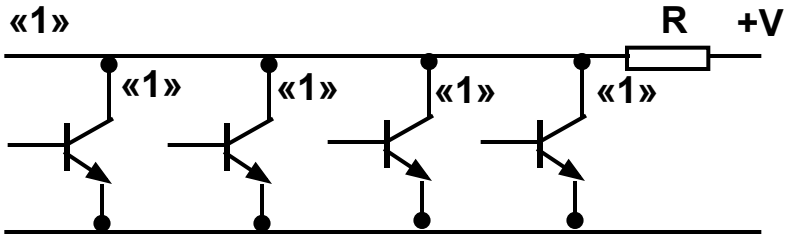


CSMA/CD+CR (CAN), есть обнаружение и разрешение коллизий

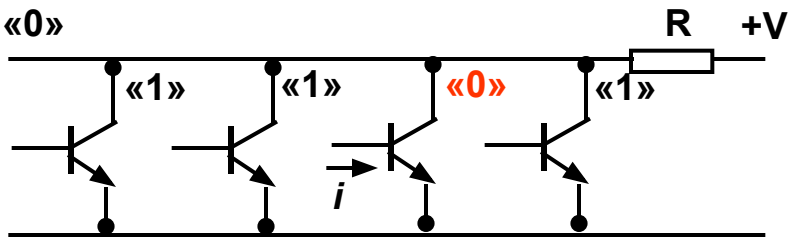


14.1 Физический уровень и уровень доступа на примере CAN (2)

Каждый узел «слушает» шину в процессе передачи заголовка сообщения. Если узел передал «1», а принял «0», то узел отказывается от дальнейшей передачи, так как это означает, что другой узел в этот же момент времени передает более приоритетное сообщение. Так реализуется разрешение конфликтов.



Все выходные ключи закрыты - на шине +V (“1”)



Один выходной ключ открыт - на шине 0 (“0”)
(Схема «Wired-AND» - «монтажное И»)

14.1 Физический уровень и уровень доступа на примере CAN (3)

Так как каждый узел должен прослушивать передачу сообщения от самого удаленного узла, то справедливо следующее неравенство (учитывающее скорость распространения электромагнитной волны в медном проводнике, скорость работы электронных схем, и допустимый сдвиг фазы на 2/3 времени передачи одного бита при встречной передаче кадров от максимально удаленных узлов для среды передачи витой медной пары):

Длина * Скорость <= 40 m * 1 MBit/s

(2/3)*Tbit > 2* (Tline + Treceiver + Ttransceiver),
где Tbit = 1 / (Скорость bit/s)
Treceiver = Ttransceiver = 25E-9 s
Tline = (Длина m) / (2E+8 m/s)

Длина 40 m - Скорость 1 Mbit/s
Длина 400 m - Скорость 0.1 Mbit/s (100 Kbit/s)
Длина 1000 m - Скорость 0.04 Mbit/s (40 Kbit/s)

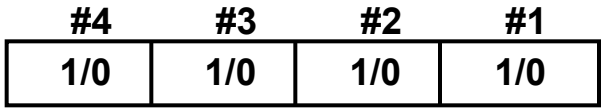
Start Of Frame **Acknowledgment** **End Of Frame**

SOF	Identifier (Priority)	Control (Length)	Data	CRC	ACK	EOF	
1	11 (29)	6	0 - 8 bytes	16	2	7	bits

Примерный формат кадра (frame) CAN.

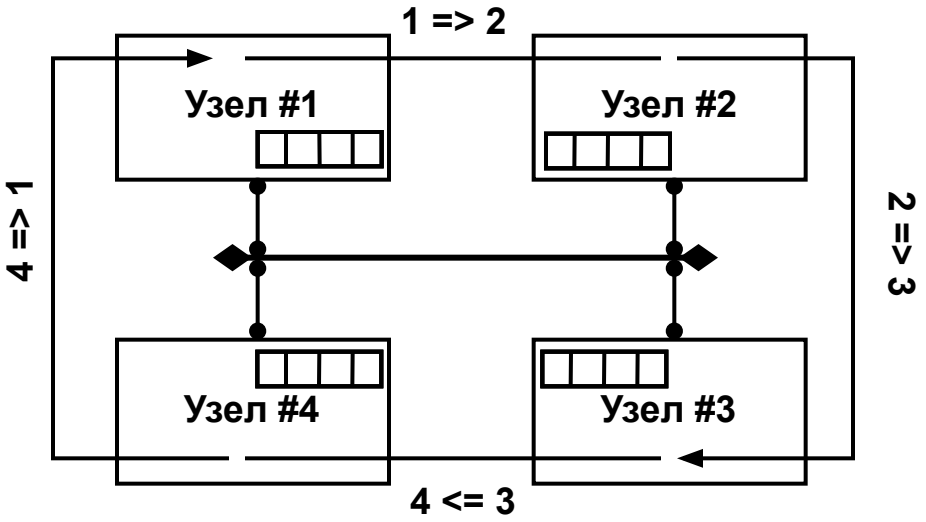
14.2 Управление сетью

Управление сетью (Network Management) предназначено для отслеживания состояний всех узлов сети. Это необходимо для принятия решений о передаче данных узлу или о приеме данных от узла в нормальном режиме работы и для отслеживания сбоев узлов.

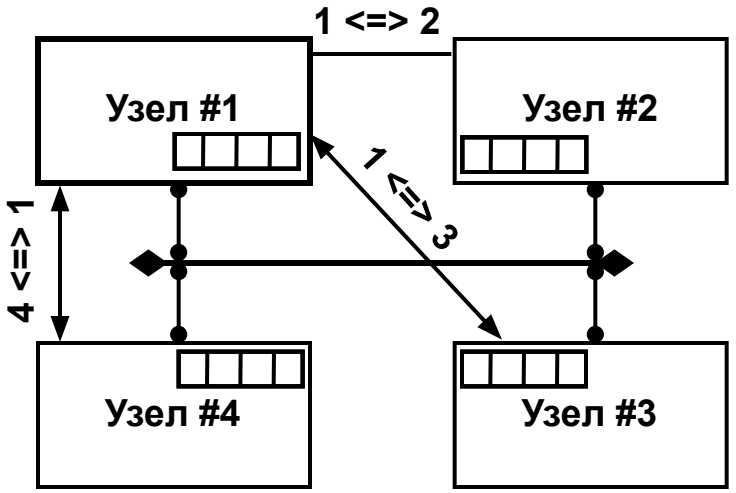


Состояние узлов (1-доступен, 0-не доступен)

Копия хранится в каждом узле

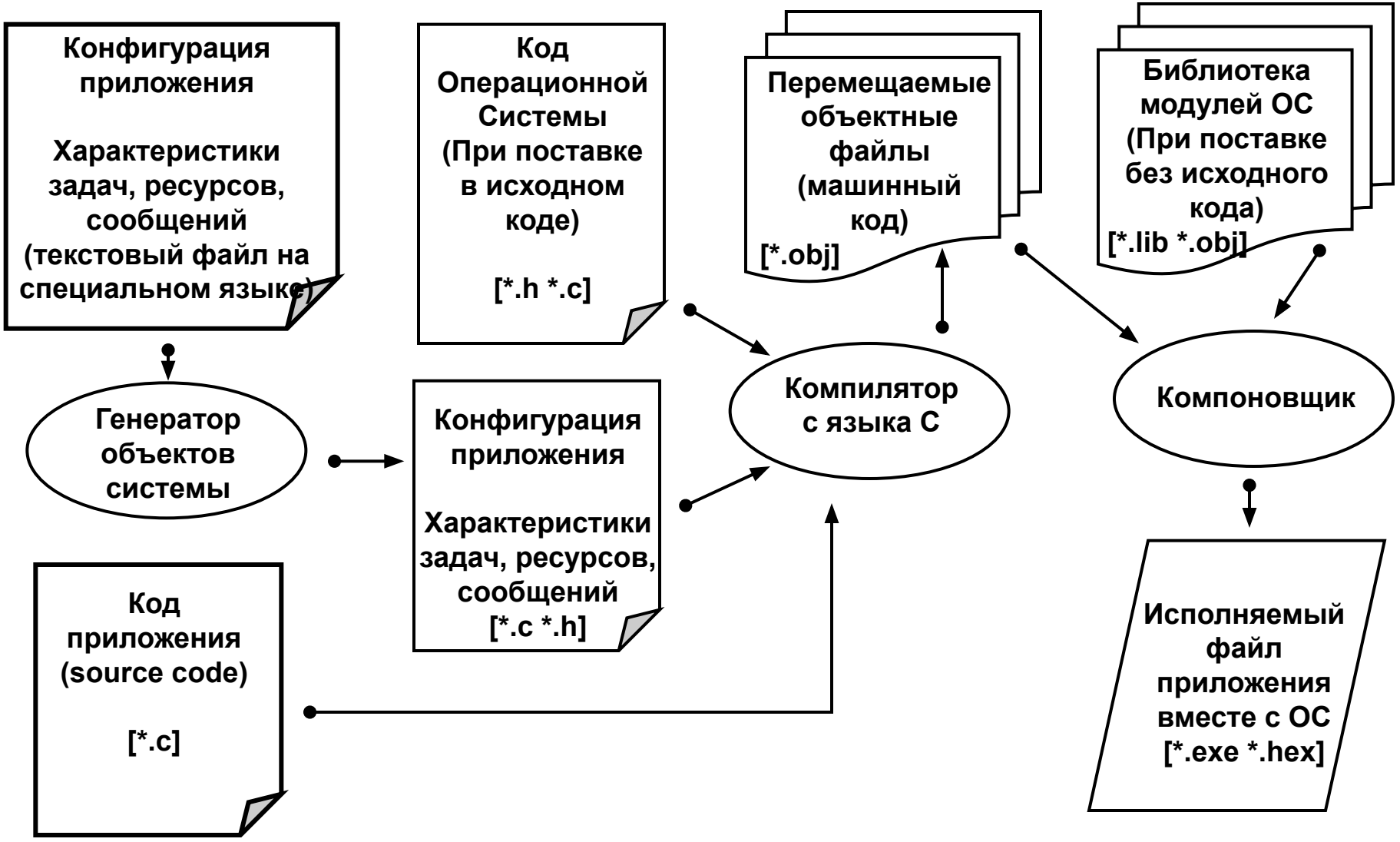


Логическое кольцо (поддерживаются алгоритмы установки и восстановления кольца).



Логическая звезда (один из узлов отслеживает состояние остальных узлов).

15. Примеры Операционных Систем (1)



Упрощенная схема создания встроенного приложения с использованием ОС

15. Примеры Операционных Систем (2)

OSEK/VDX

Спецификации встроенной операционной системы реального времени (OSEK/VDX OS), коммуникационная подсистема (OSEK/VDX COM) и управление сетью (OSEK/VDX NM).
Основная область применения - транспортные средства (автомобили).
Спецификации разрабатываются Европейским консорциумом производителей автомобилей и поставщиков программного обеспечения с 1995 года. Последние версии спецификаций выпущены в конце 2000 года.

В настоящее время существует около десятка коммерчески доступных продуктов, используемых DaimlerChrysler, BMW, и др. в разрабатываемых моделях автомашин.

OSEK Official Web site: www.osek-vdx.org

Real-Time Linux

Модификации ОС общего назначения для применения в приложениях реального времени.

Существует несколько совершенно различных решений, в частности:

1) **Архитектура со интегрированным ядром реального времени (dual-kernel architecture).**
2) **Архитектура с модифицированным планировщиком реального времени (single-kernel architecture).**

The Real-time Linux Software Quick Reference Guide:
http://www.linuxdevices.com/articles/AT8073314981.html

15.1 OSEK/VDX (1)

OSEK = “Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug” (Open systems and the corresponding interfaces for automotive electronics).
VDX = “Vehicle Distributed eXecutive”

OSEK/VDX - совместный проект автомобилестроителей и поставщиков автомобильных приложений, определяющий открытую архитектуру для распределенных систем транспортных средств.

15.1 OSEK/VDX (2)

Функциональные свойства Операционной Системы OSEK/VDX OS

- **Планирование с фиксированными приоритетами**
- **Вытесняемое, невытесняемое, и смешанное диспетчирование задач**

- **Четыре класса соответствия системы (conformance classes)**
- **Поддержка задач с состоянием ожидания через механизм событий (в двух классах соответствия)**
- **Поддержка множественного планирования задач (multiple activation request)**

- **Разделение ресурсов между задачами и обработчиками прерываний с помощью протокола высшего приоритета (называемого OSEK Priority Ceiling Protocol). Поддержка PTS с помощью механизма «внутренних ресурсов»**

- **Межзадачная коммуникация, обеспечивающая прозрачную передачу данных внутри одного устройства и между узлами сети (используются одни и те же вызовы ОС)**

- **Поддержка трех категорий обработчиков прерываний и вложенных прерываний**
- **Задержанное выполнение вызовов ОС из обработчиков прерываний без ограничений вызовов сервисов ОС**

- **Универсальный механизм поддержки таймеров и других считающих устройств с помощью алармов**

- **Поддержка отладочного режима работы, пользовательских обработчиков ошибочной ситуации, переключения контекста задач, старта и останова операционной системы**

15.1 OSEK/VDX (3)

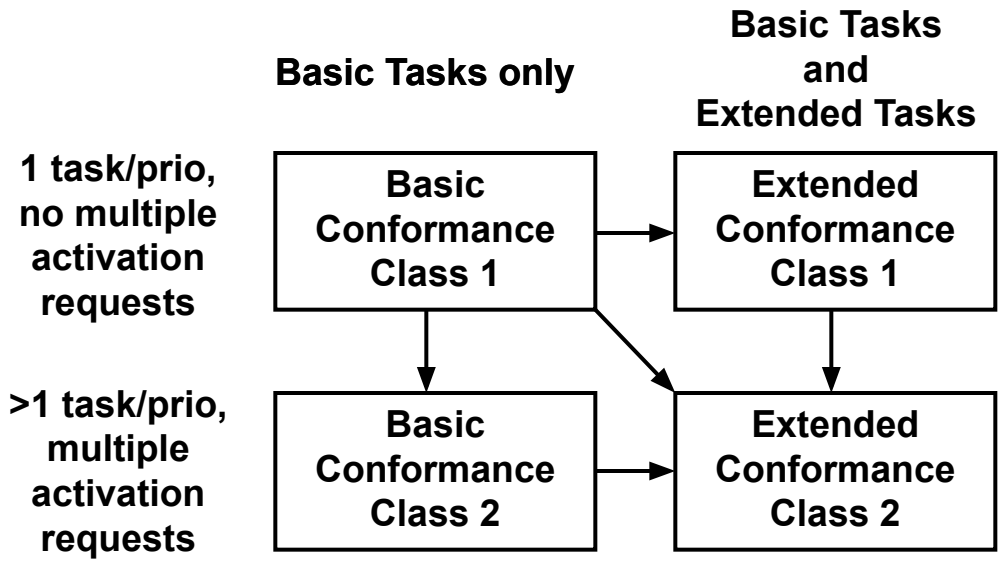


Схема совместимости классов соответствия

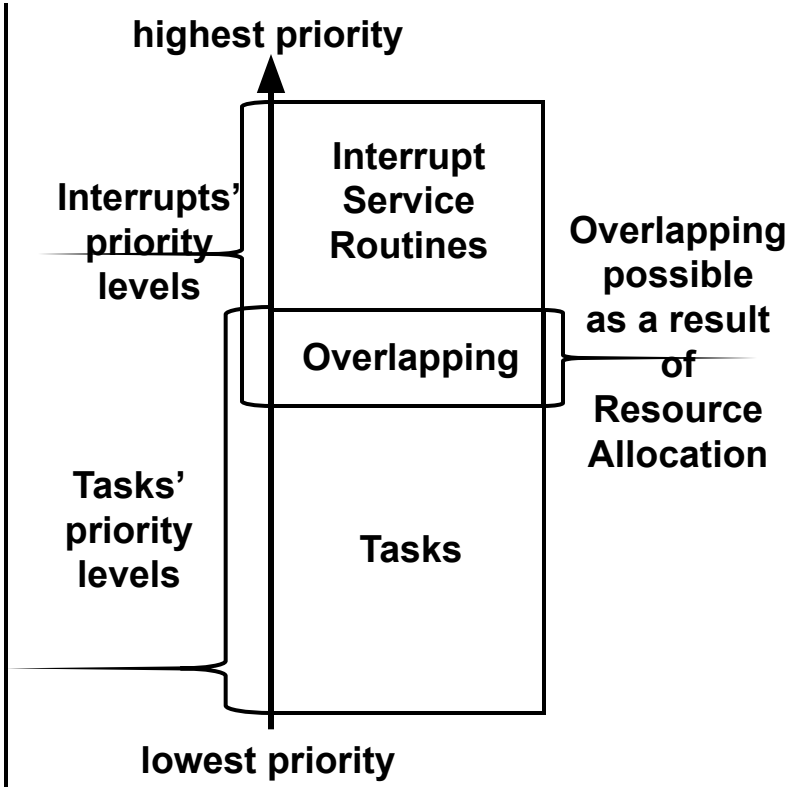
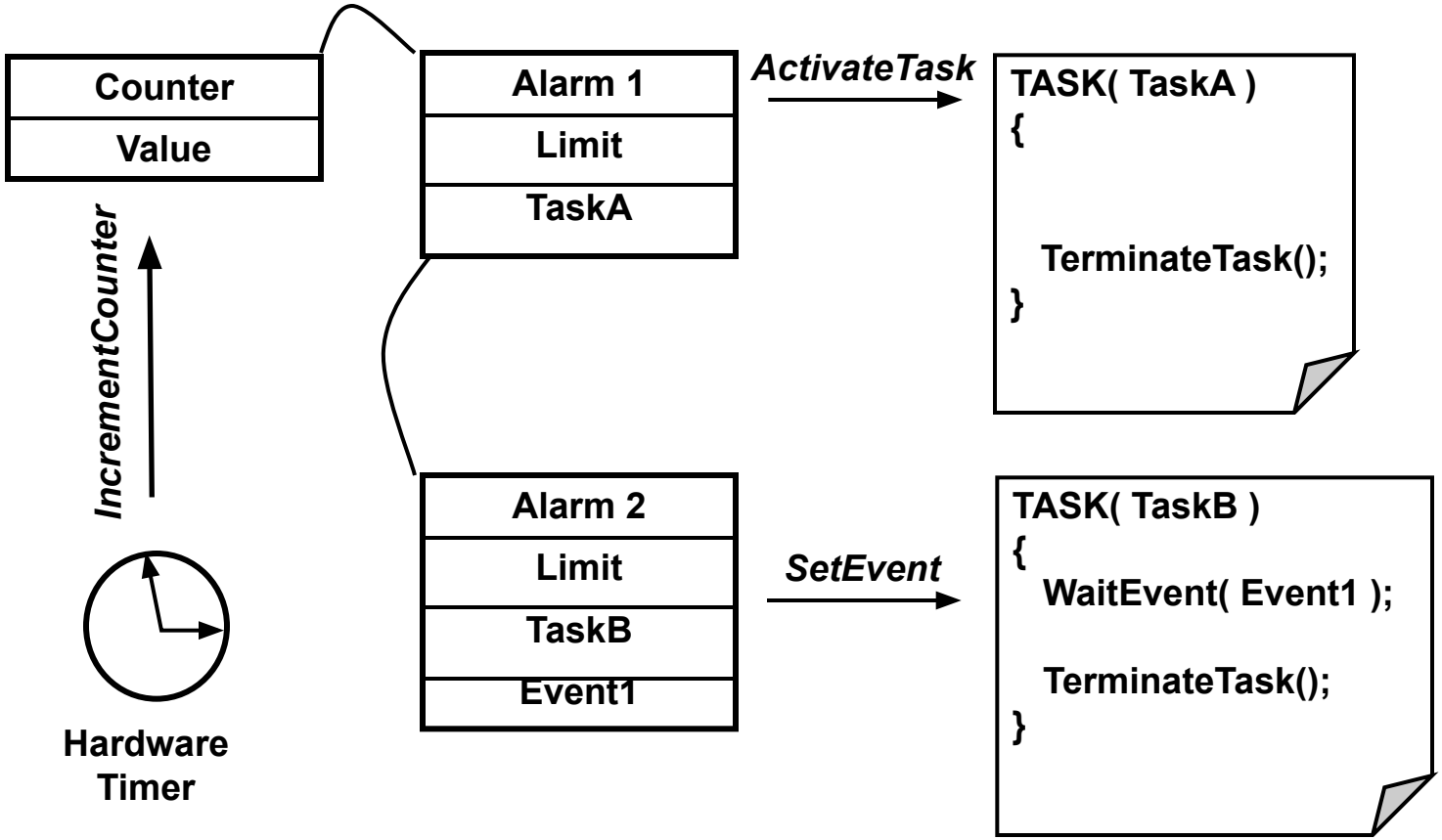


Схема приоритетов поддерживает перекрытие приоритетов задач и обработчиков только во время исполнения задачи.
Задача всегда стартует с приоритетом ниже приоритета обработчиков прерывания.

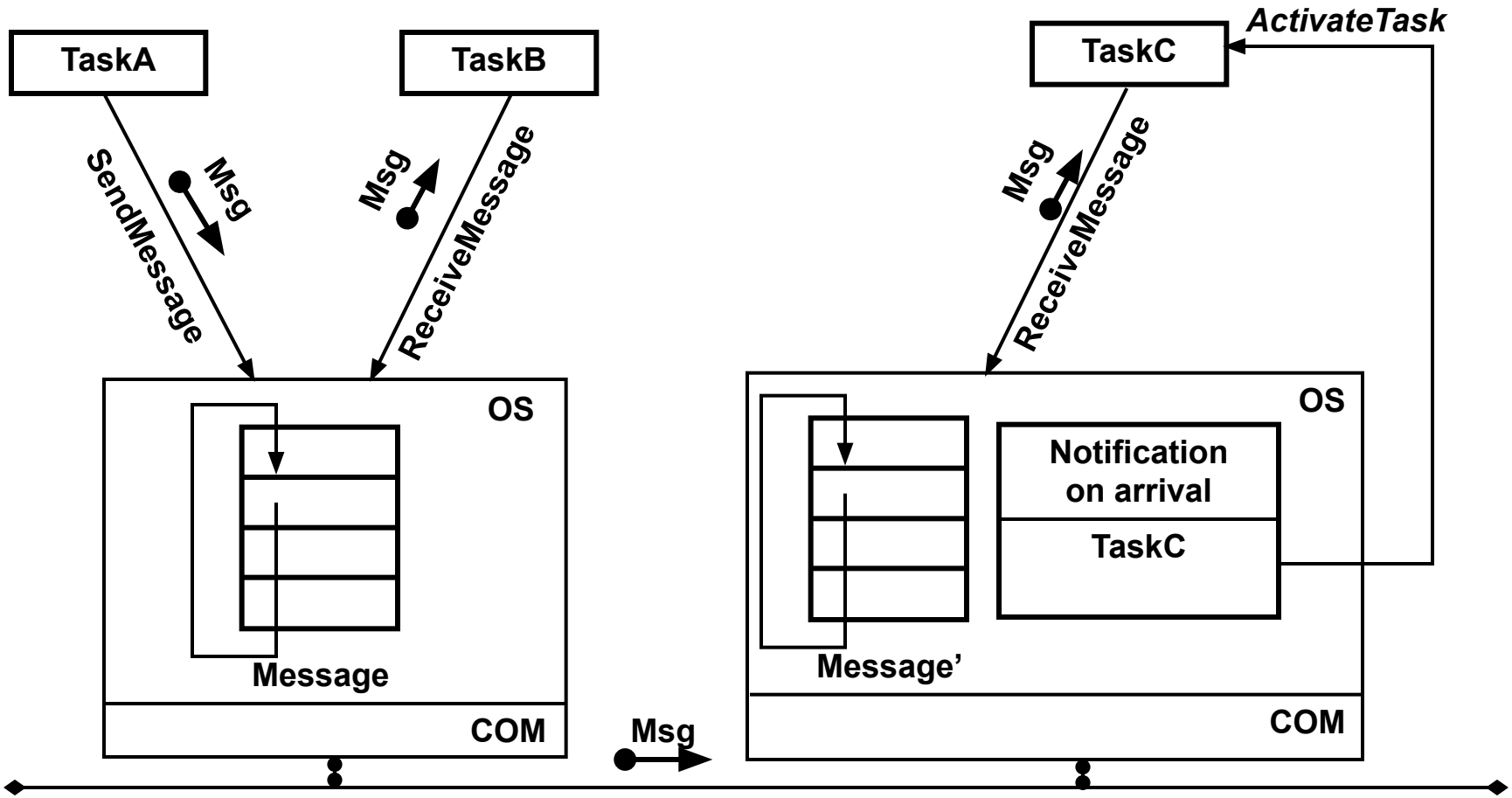
Планирование задач поддерживает принцип FIFO, т.е. равноприоритетные задачи планируются в том порядке, в котором они активизировались (как для первого запуска, так и для множественной активизации).

15.1 OSEK/VDX (4)



Механизм алармов (alarms) предполагает наличие счетчиков (counters), хотя и не специфицирует интерфейс счетчиков.
Алармы могут использоваться как для подсчета времени, так и для измерения углов, линейных перемещений, и т.п.

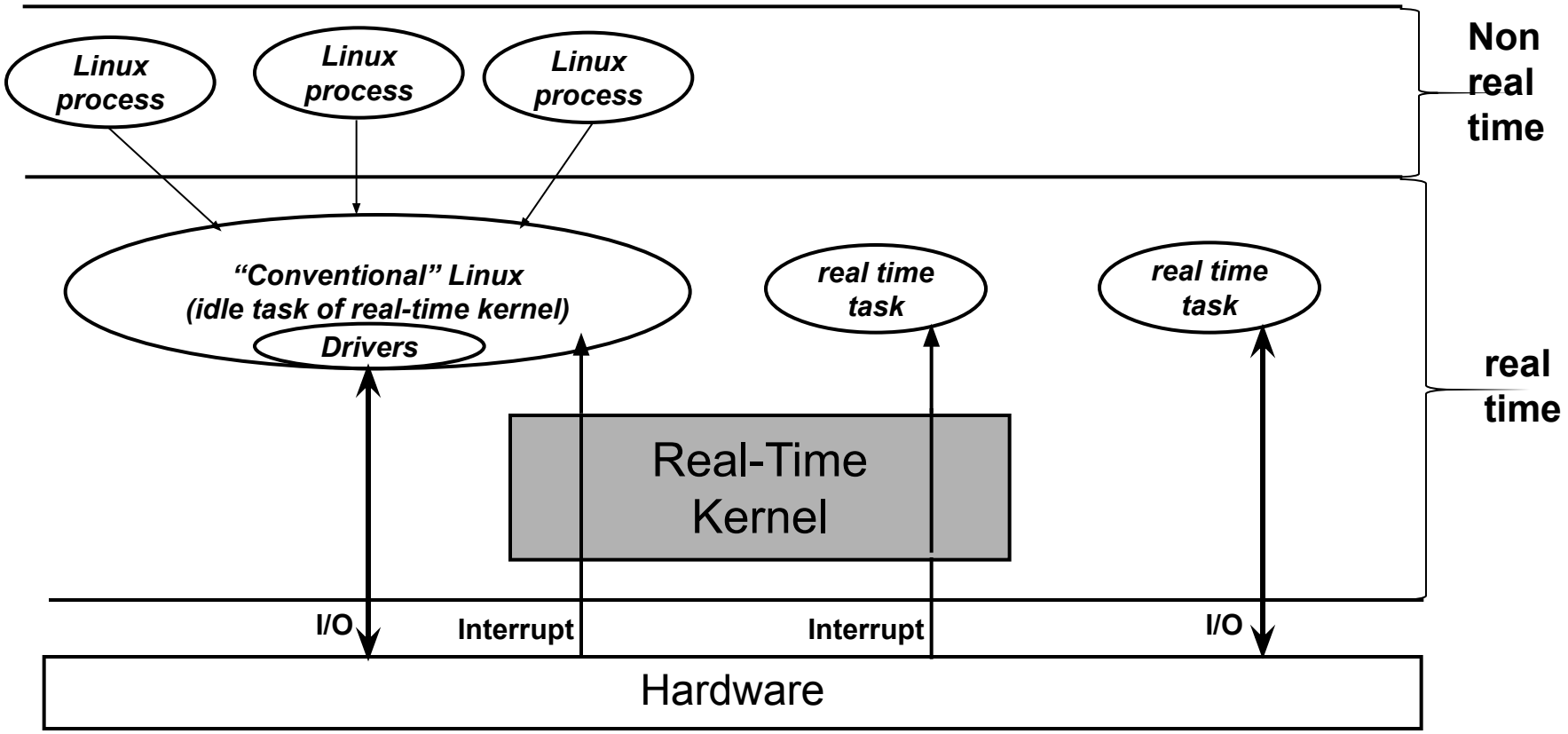
15.1 OSEK/VDX (5)



Сообщения могут передаваться между задачами в одном узле, и также между задачами, находящимися в разных узлах. Один интерфейс используется для локальной и сетевой передачи сообщений.
Нотификация осуществляется как по приему сообщений, так и по не-отправке сообщений.

15.2 Real-Time Linux (1)

Dual-kernel architecture (Embedix from Lineo, RTLinux from FSMLabs)

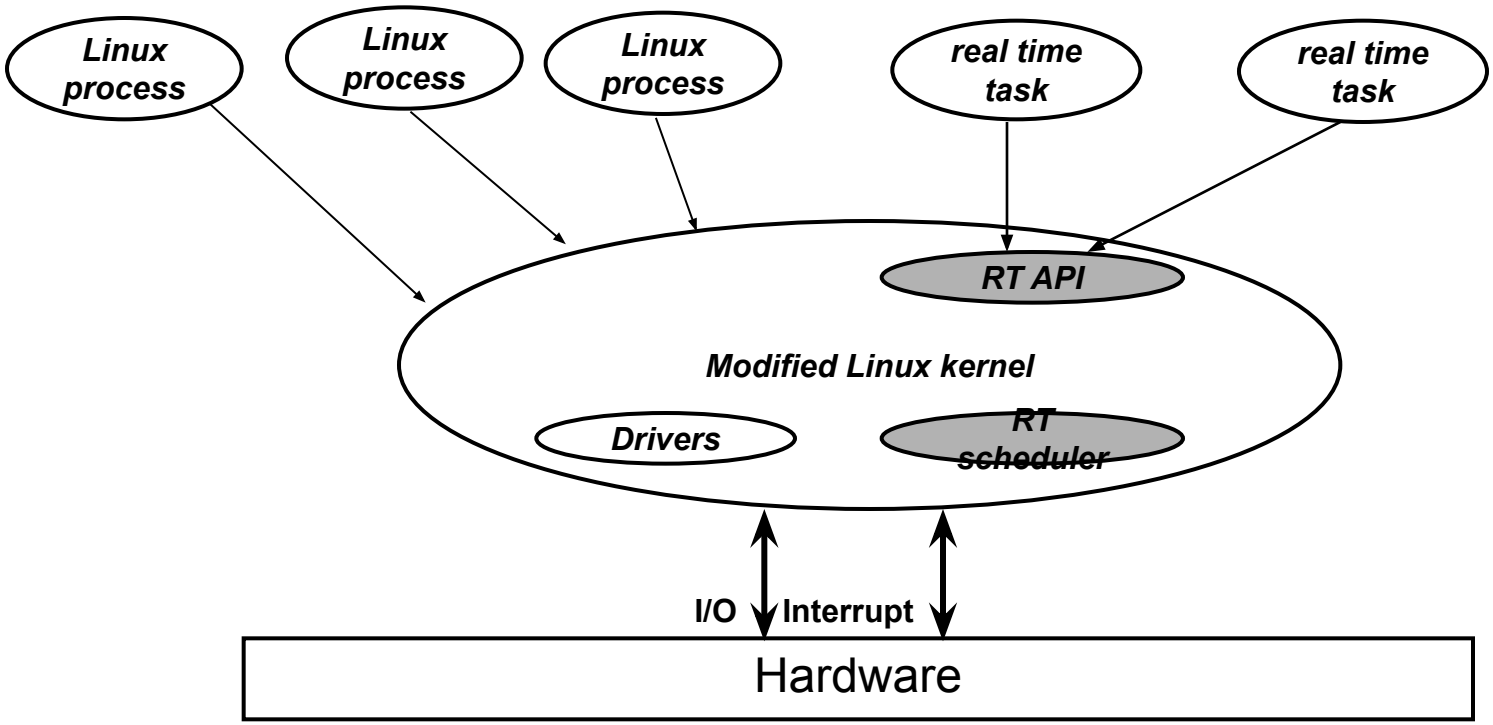


(+) поддерживается жесткое реальное время для RT задач
(+) сроки исполнения в микросекундном диапазоне
(+) используется (почти) немодифицированное ядро Linux

(-) не поддерживается реальное время для процессов Linux

15.2 Real-Time Linux (2)

Single-kernel architecture (HardHat from MontaVista, KURT)



(+) поддерживается жесткое реальное время для Linux задач
(+) возможна интеграция планировщиков
(+) сроки исполнения в диапазоне десятков микросекунд
(+) используется механизмы защиты Linux

(-) требуется модификация ядра
(-) системные модули (драйверы) могут «сводить на нет» характеристики реального времени