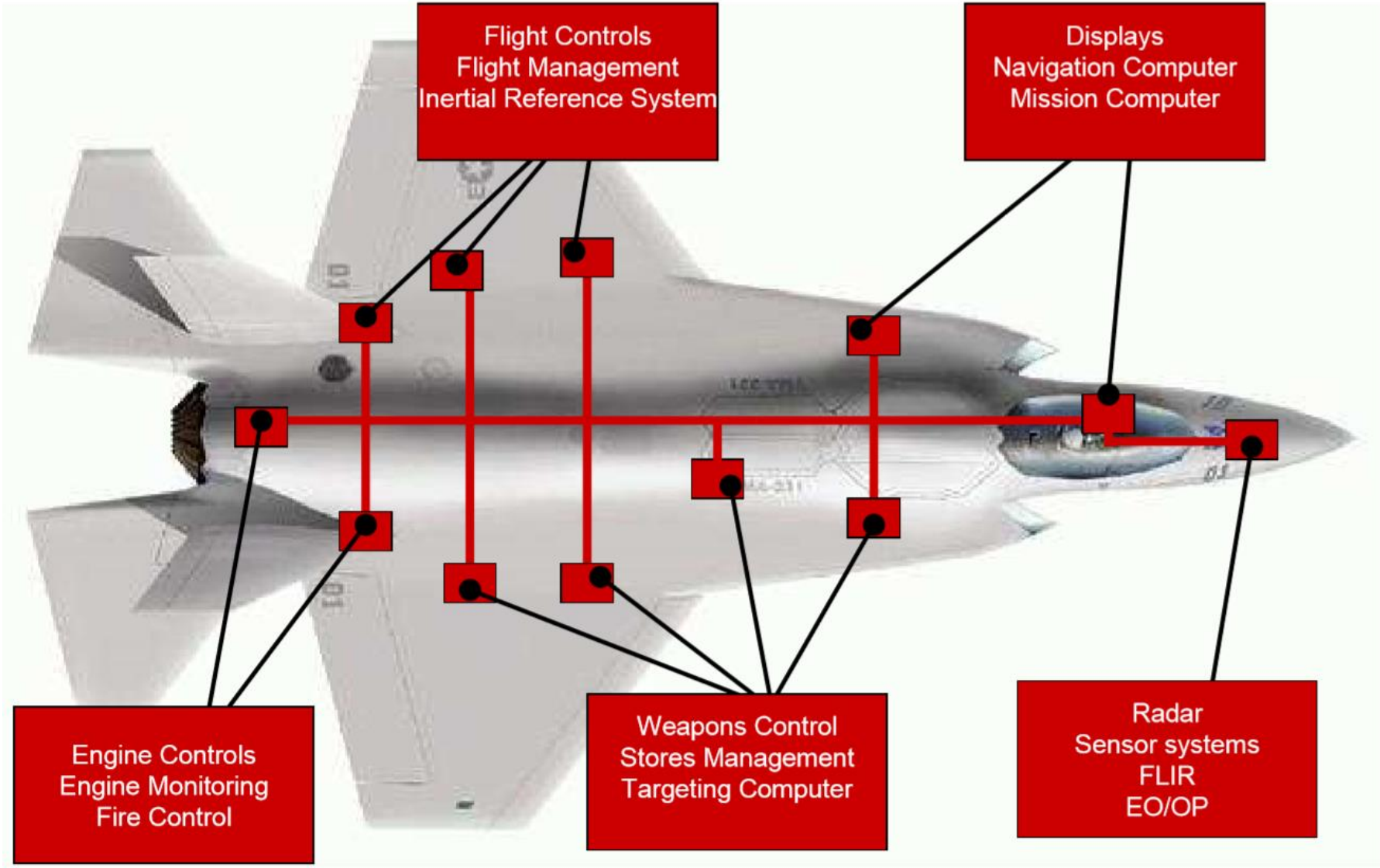
ВСТРОЕННЫЕ ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 4:

Статико-динамическое планирование вычислений в системах интегрированной модульной авионики

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

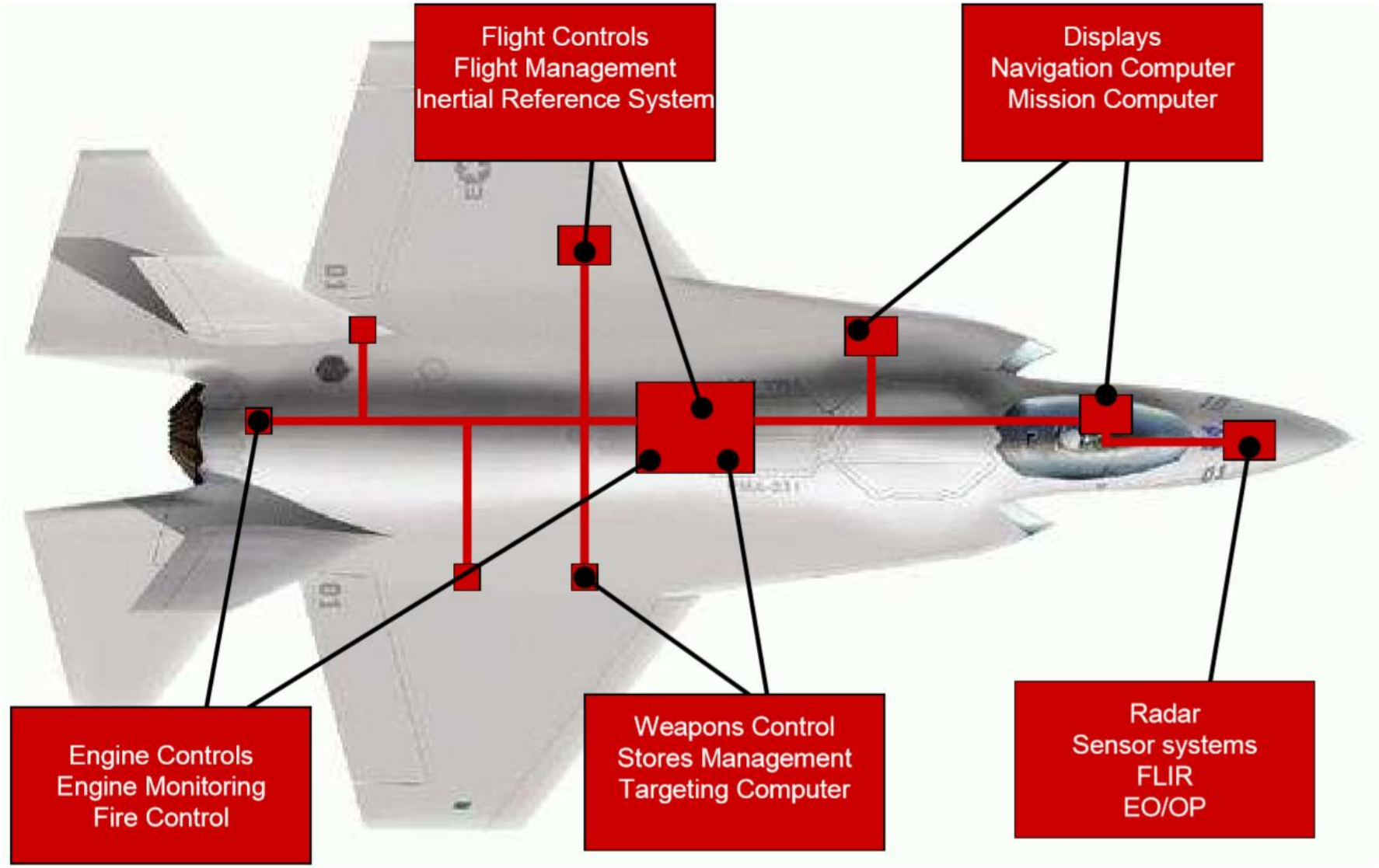
Федеративная ИУС РВ



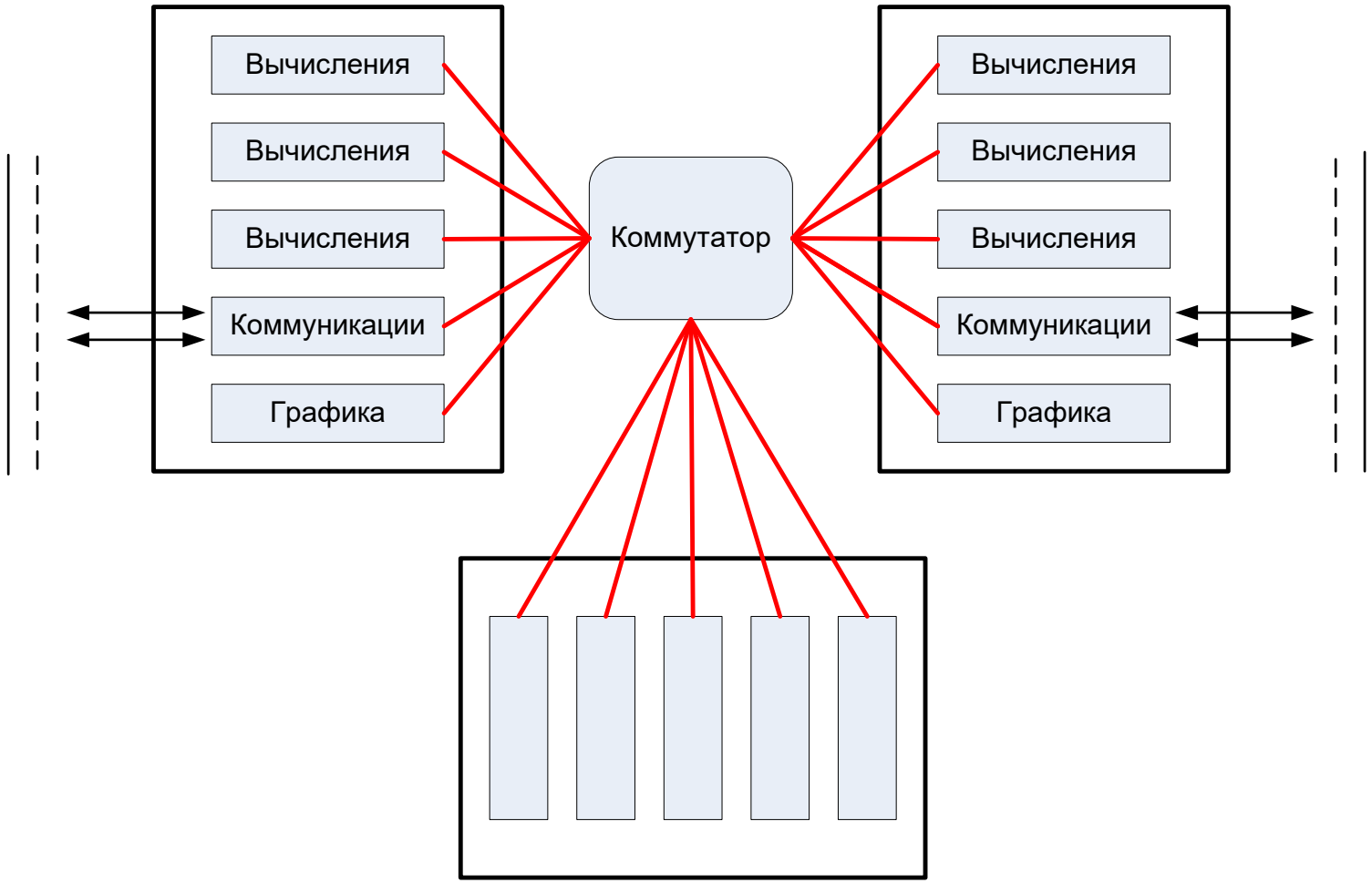
Федеративная ИУС РВ

- Специализированные блоки
  - по назначению
  - по архитектуре
- ПО различных подсистем – на различных блоках
  - изоляция по памяти
  - нет конкуренции подсистем за процессорное время
- Недостатки:
  - низкая переносимость и повторная используемость ПО
  - «зоопарк» процессорных архитектур и программных интерфейсов

ИУС РВ с архитектурой ИМА



ИУС РВ с архитектурой ИМА с сетью на базе коммутатора



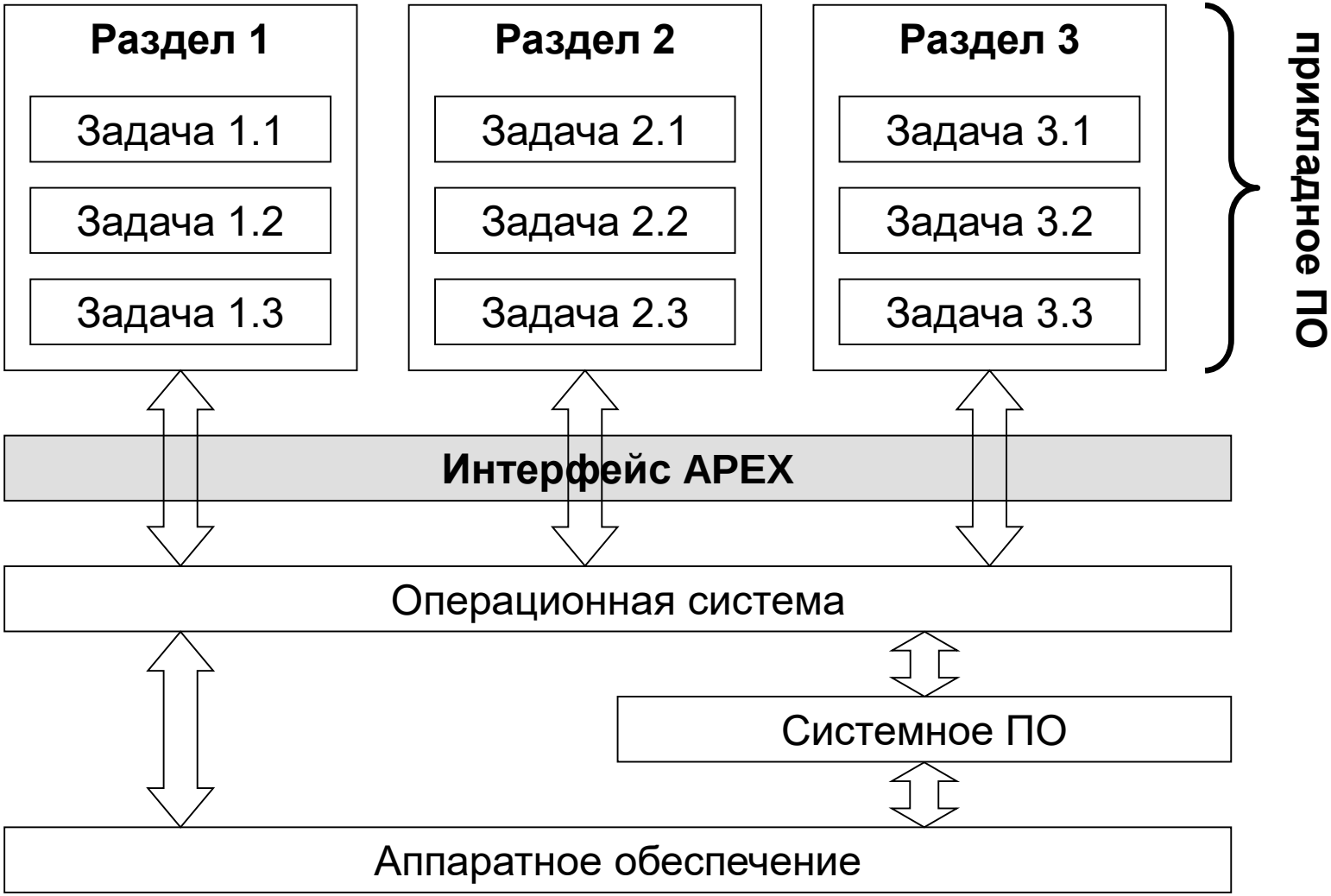
ИУС РВ с архитектурой ИМА

- Логически единый распределенный вычислитель
  - унифицированные модули нескольких типов
  - единая архитектура процессоров
  - унифицированный программный интерфейс
- Разделение вычислительных ресурсов между ПО различных подсистем
- Проблемы:
  - изоляция по памяти
  - разделение процессорного времени
- Решение: каждой подсистеме → раздел

ИМА и разработка ПО

- Особенности:
  - Стандартное API со стороны ОС
  - Статическое разделение времени, памяти и ресурсов
- Преимущества:
  - Надежность
  - Переносимость
  - Возможность повторного использования
  - Модульность
  - Упрощение верификации и сертификации

Структура ПО. Интерфейс APEX



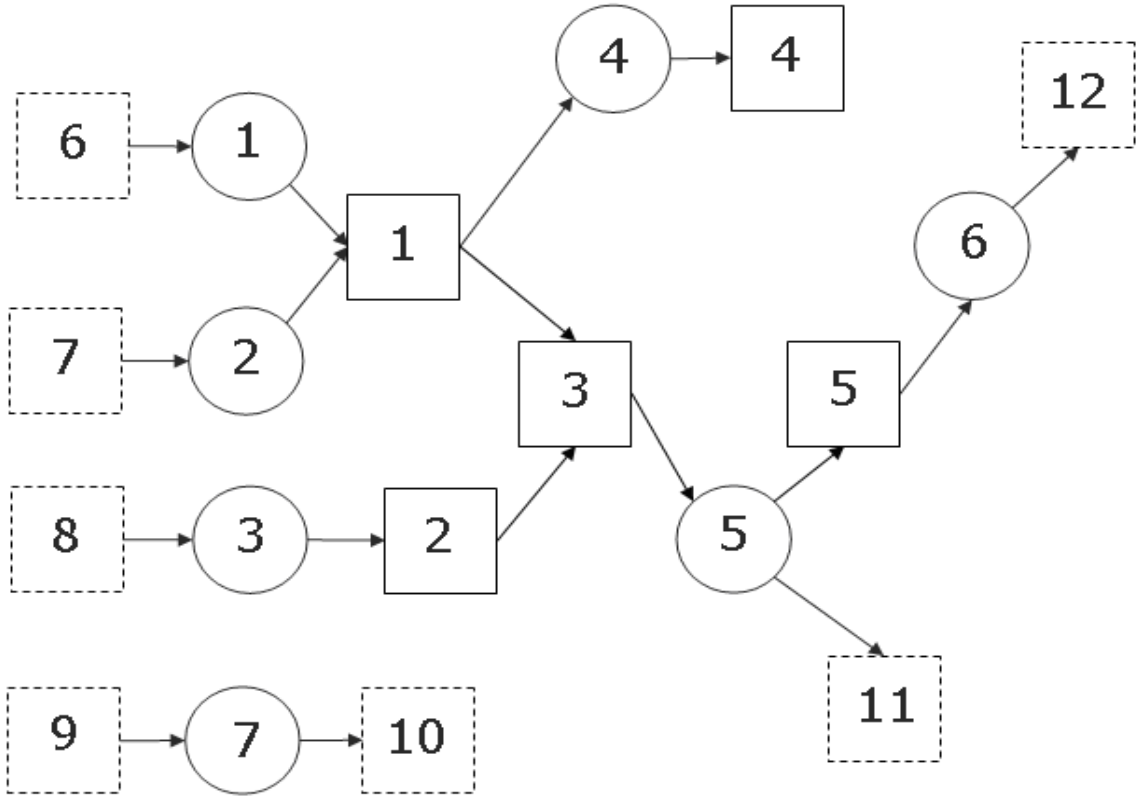
Взаимодействие между разделами

- Порты с очередью сообщений
- Порты с перезаписью сообщений
  - Буфер фиксированной длины
  - Сообщение в буфере перезаписывается
  - Отправка с заданным периодом

Взаимодействие внутри разделов

- Обмен данными
  - Передача сообщений
  - Общая память
- Механизмы синхронизации
  - Семафоры
  - События

Рабочая нагрузка

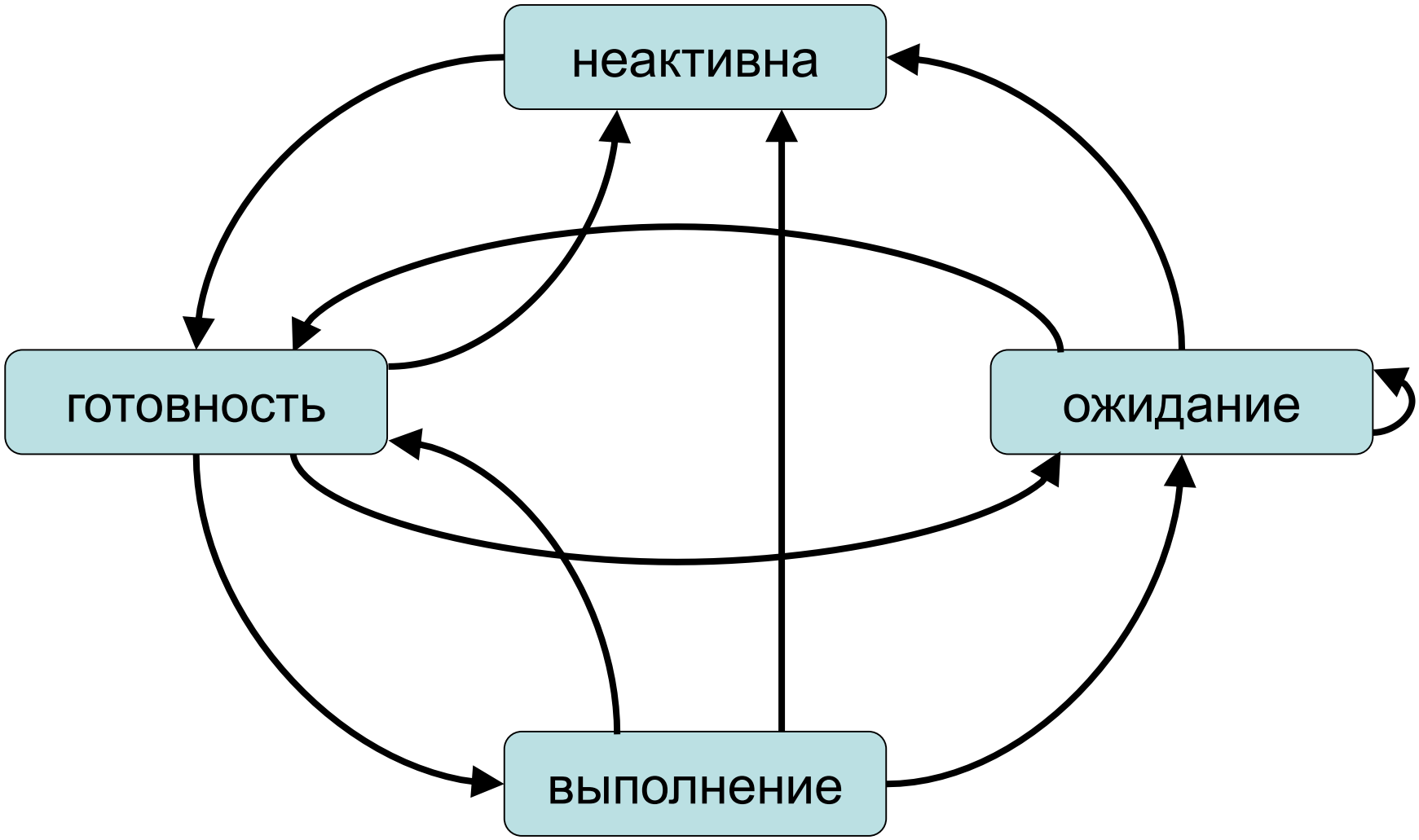


- периодические задачи
- сообщения
- зависимости по данным

Синхронизация на практике

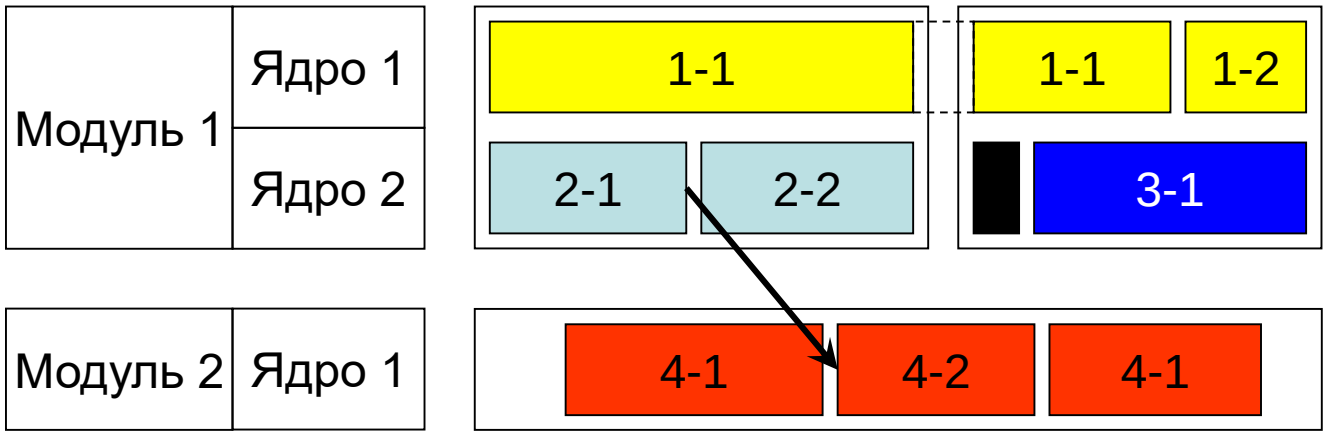
- Ожидание данных только от задач с той же частотой
  - «синхронная» зависимость по данным
- Отправитель выполняется чаще получателя:
  - на входе получателя почти всегда актуальные данные
  - ожидать нет смысла
- Отправитель выполняется реже получателя
  - ожидание нарушает требования по частоте для задачи-получателя

Состояния задач в ARINC653

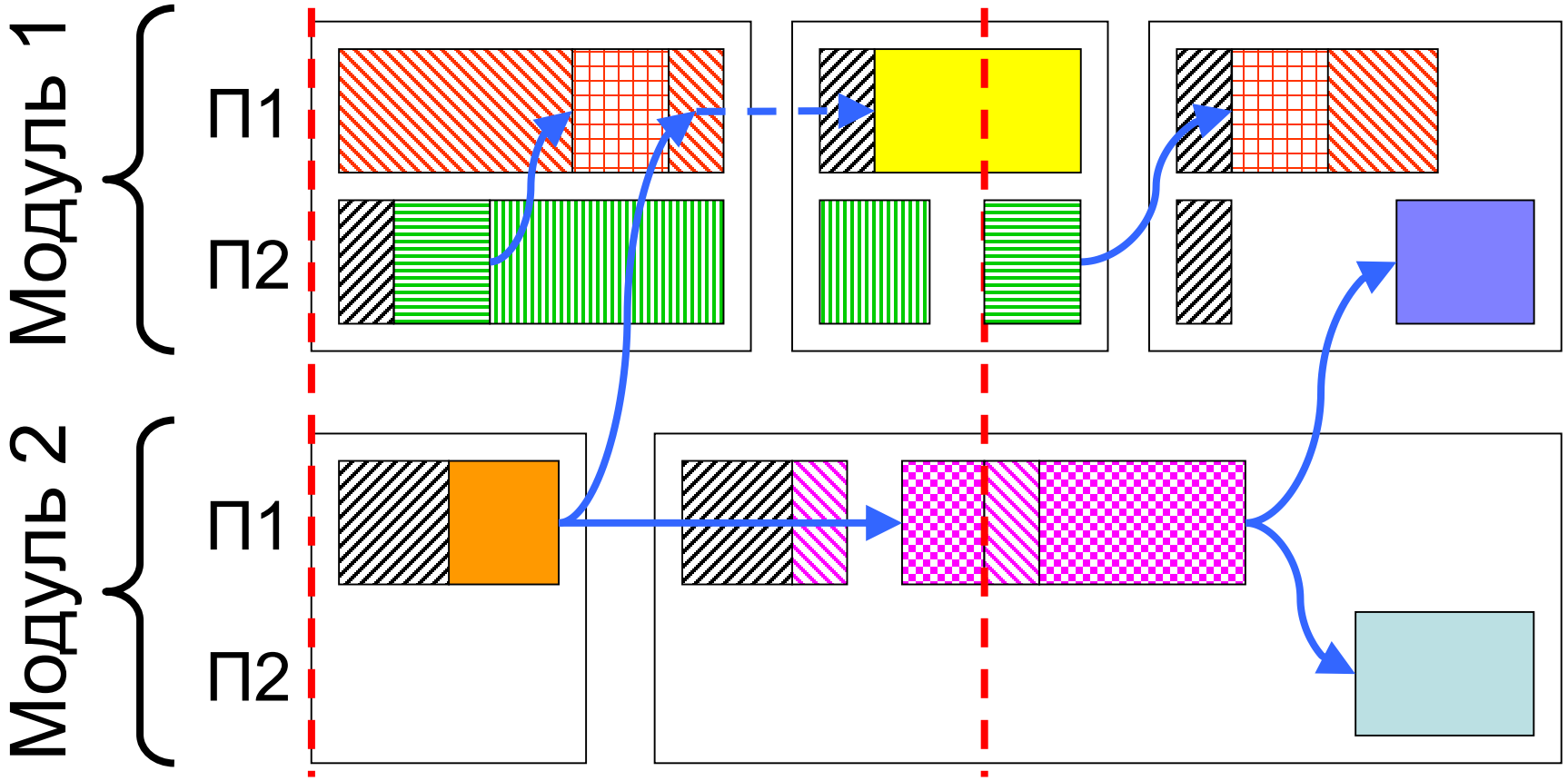


Выполнение задач в системе

- Задачи раздела выполняются в рамках окон
  - статическое расписание окон
  - границы окон одинаковы для всех ядер одного модуля
- Между разделами происходит переключение контекста
- Работы в окне: динамическое планирование
  - очередь выполнения
  - приоритеты
  - вытеснение
  - ожидание входных данных
- Незавершенная работа может быть возобновлена в следующем окне



Пример расписания



Задача планирования: входные данные

- Описание системы ИМА
  - набор модулей
  - модуль → набор и типы процессорных ядер
  - ядро → верхняя граница загрузки
- Описание рабочей нагрузки
  - наборы задач, сообщений, разделов
  - задача → период, приоритет, WCET (для типа ядра)
  - раздел → задачи, допустимые ядра
  - сообщение → отправитель, получатель, размер, длительность передачи (через память, через сеть)
  - *свободные задачи,* допустимые ядра

Задача планирования: цели

- Составить разделы из свободных задач
  - Трафик между разделами → min
  - Загрузка ядра разделом ≤ U_{max}

    (при выполнении раздела целиком на одном ядре)
- Привязать разделы к ядрам
  - Трафик между модулями → min

    (минимизация загрузки сети)
  - Загрузка ядра ≤ U_{max}(ядро)
  - Привязка к допустимым ядрам
  - Выполнение условий динамической планируемости
  - Инкрементальная привязка (расширение прежней)
- Построить расписание окон для каждого ядра
  - Корректность расписание проверяется моделированием работы динамического планировщика
  - Расписание считается корректным, если все задачи выполняются в пределах директивных сроков (Д.С. = период) при длительностях выполнения, равных WCET

$$\sum_{i=0}^{n} T_i F_i \le n\left(2^{\frac{1}{n}} - 1\right)$$

Ограничения корректности

- Раздел привязан ровно к одному ядру
- Не более одного раздела в окне
- Окна не пересекаются по времени
- Работы выполняются в рамках окон раздела
- В каждый момент времени выполняется не более одной работы
- Все работы выполняются полностью
- Выполняются зависимости по данным
- Соблюдаются приоритеты

Жадный алгоритм привязки разделов к процессорным ядрам

1. Выбрать непривязанный раздел *P* с максимальным трафиком между *P* и привязанными разделами.
2. Для каждого модуля рассчитать трафик между *P* и разделами, привязанными к ядрам других модулей.
3. Упорядочить модули по убыванию этого трафика.
4. В цикле по модулям:
   - если раздел *P* может быть привязан к какому-либо ядру данного модуля без нарушения ограничений на загрузку ядра, то привязать *P* к этому ядру и выйти из цикла;
   - иначе, если разделы на данном модуле могут быть перераспределены между ядрами этого модуля так, чтобы в достаточной мере «разгрузить» одно из ядер, то выполнить перераспределение, привязать *P* к этому ядру и выйти из цикла;
5. Если на шаге 4 не выбрано никакое ядро, то **стоп** (неуспех).
6. Если остались непривязанные разделы, то перейти к шагу 1, иначе **стоп** (успех).

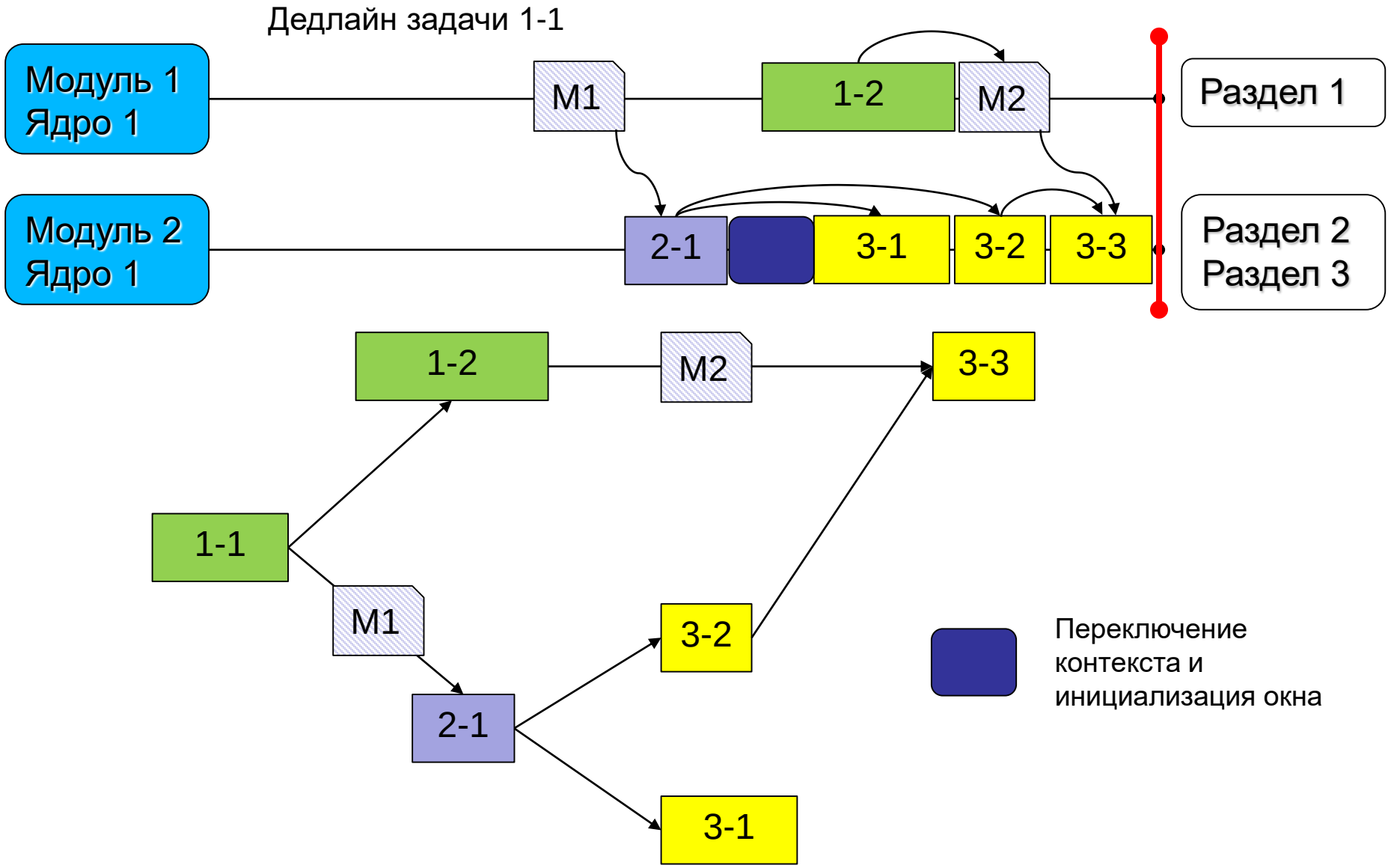
Альтернативные алгоритмы привязки разделов к процессорным ядрам

- Метод ветвей и границ
  - Критерий отсечения: загрузка ядра превышает допустимую, или ранее найдено лучшее решение
  - Недостаток: время выполнения на реальных данных
- Упаковка в контейнеры
  - Ядро = контейнер
    - емкость = U_{max}(ядро)
  - Раздел = объект
    - объем = вклад раздела в загрузку ядра
    - стоимость = трафик, становящийся «внутренним» для модуля в случае привязки раздела к ядру этого модуля
  - Проблемы:
    - объем объекта зависит от выбора контейнера (длительность выполнения задачи зависит от типа ядра)
    - стоимость объекта зависит от расположения других объектов

Построение расписания: схема алгоритма

1. Построение графа зависимостей работ
2. Уточнение директивных интервалов работ на основе зависимостей
3. Построение последовательностей выполнения работ «слева направо», параллельно для всех ядер
4. Построение расписания окон на основе построенной последовательности работ
5. Назначение приоритетов задач на основе построенной последовательности работ

Уточнение директивного срока



Планирование «слева направо»

1. Работа помещается в список готовых для выполнения, если:
   - Начался ее директивный интервал
   - Получены все синхронные входящие сообщения для данной работы
2. Очередная работа для размещения выбирается из списка:
   - Если отсутствуют готовые работы с большим приоритетом
   - По критерию EDF (если приоритет еще не определен)

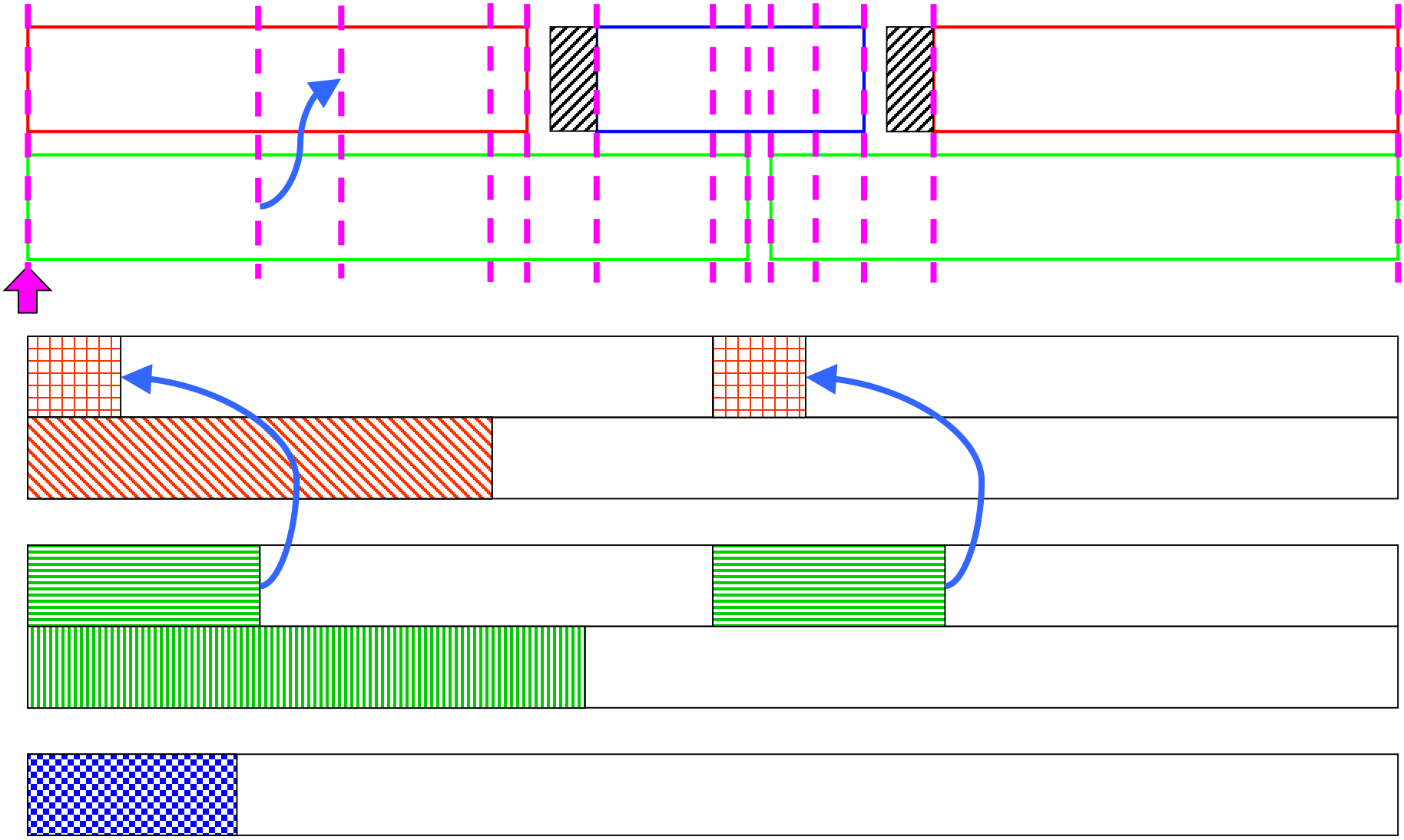
Модель вычислительного процесса

- Служит для проверки корректности расписания окон
- Модель основана на событиях
- Схема работы:
  1. Создание начальных событий
  2. Выбор событий с минимальным временем
  3. Обновление списка готовых к выполнению работ
  4. Обработка выбранных событий
  5. Если список событий не пуст, переход к п.2

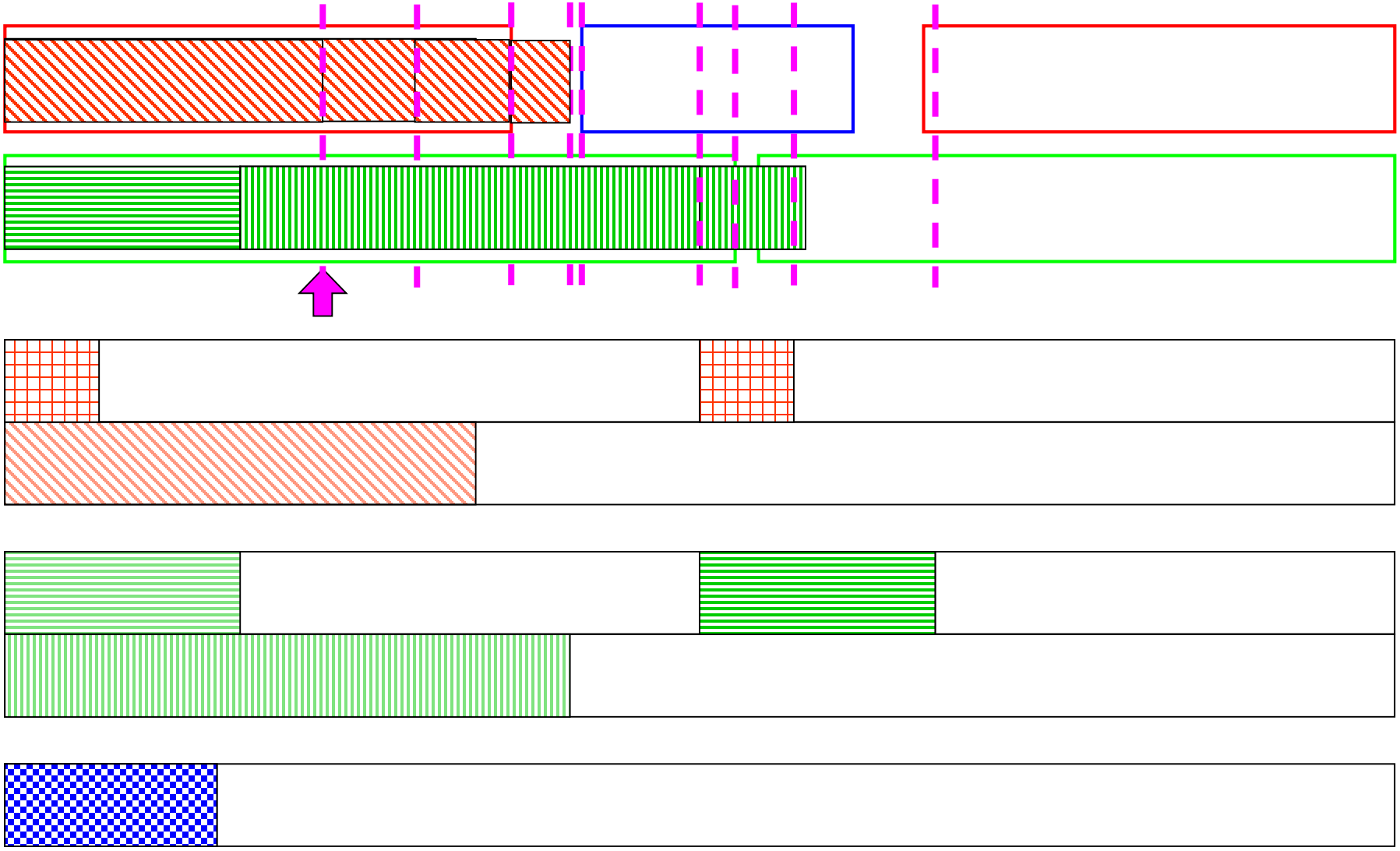
Типы событий

- Открытие окна
- Закрытие окна
- Начало директивного интервала
- Завершение работы
- Доставка сообщения

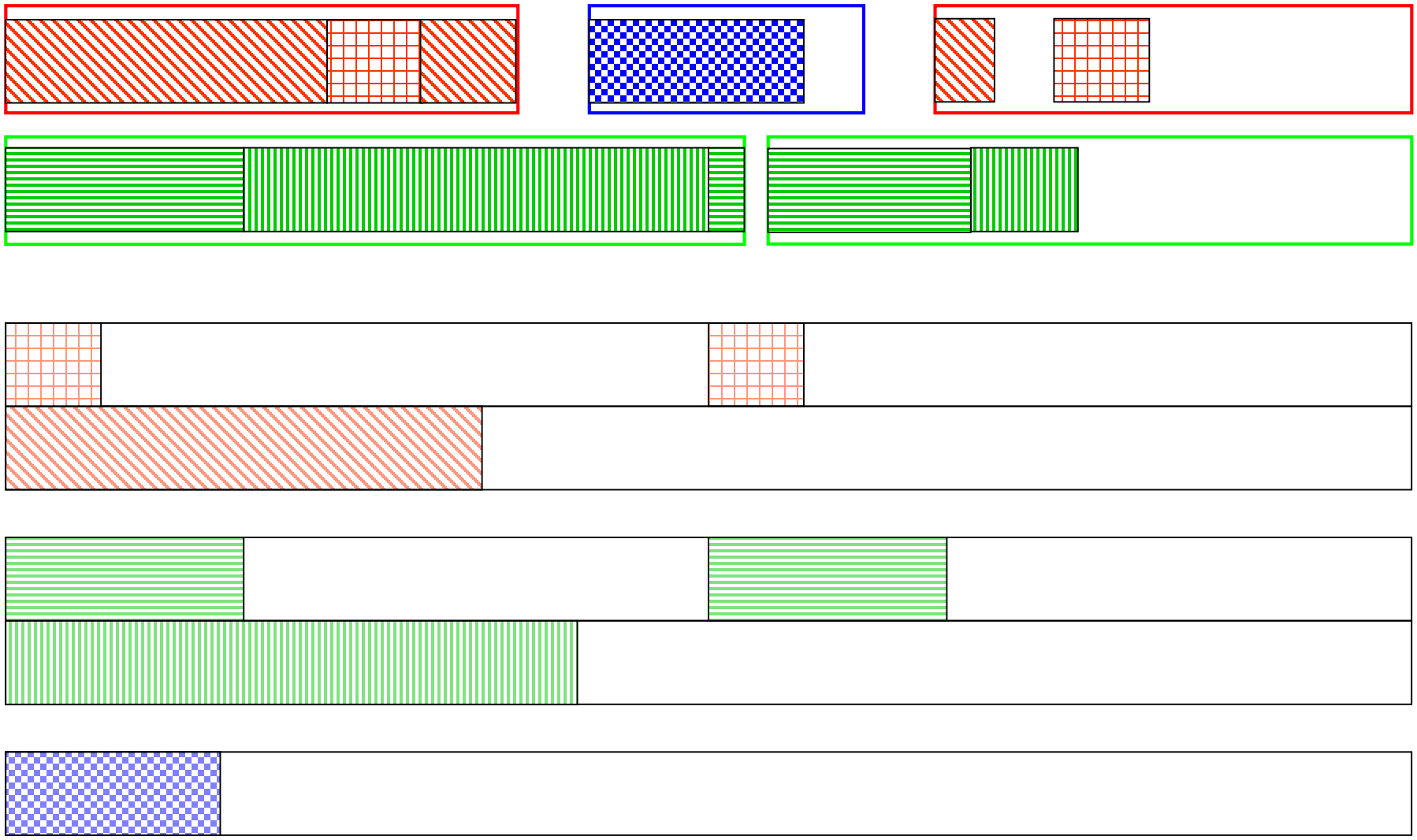
Работа модели



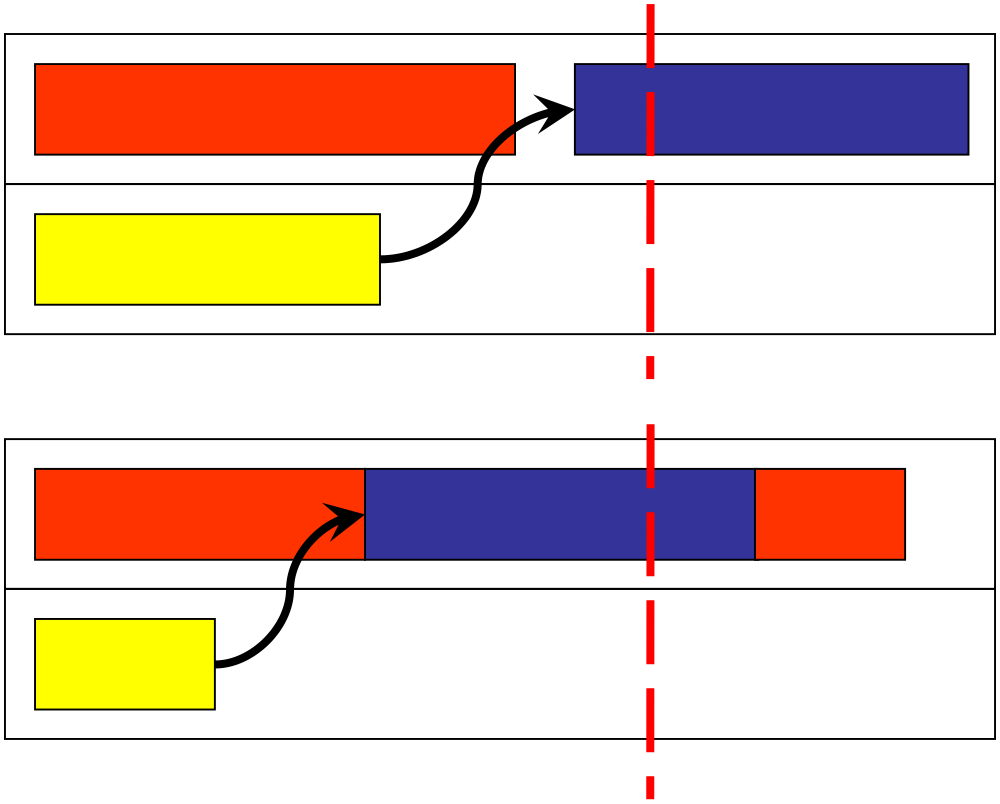
Работа модели



Работа модели



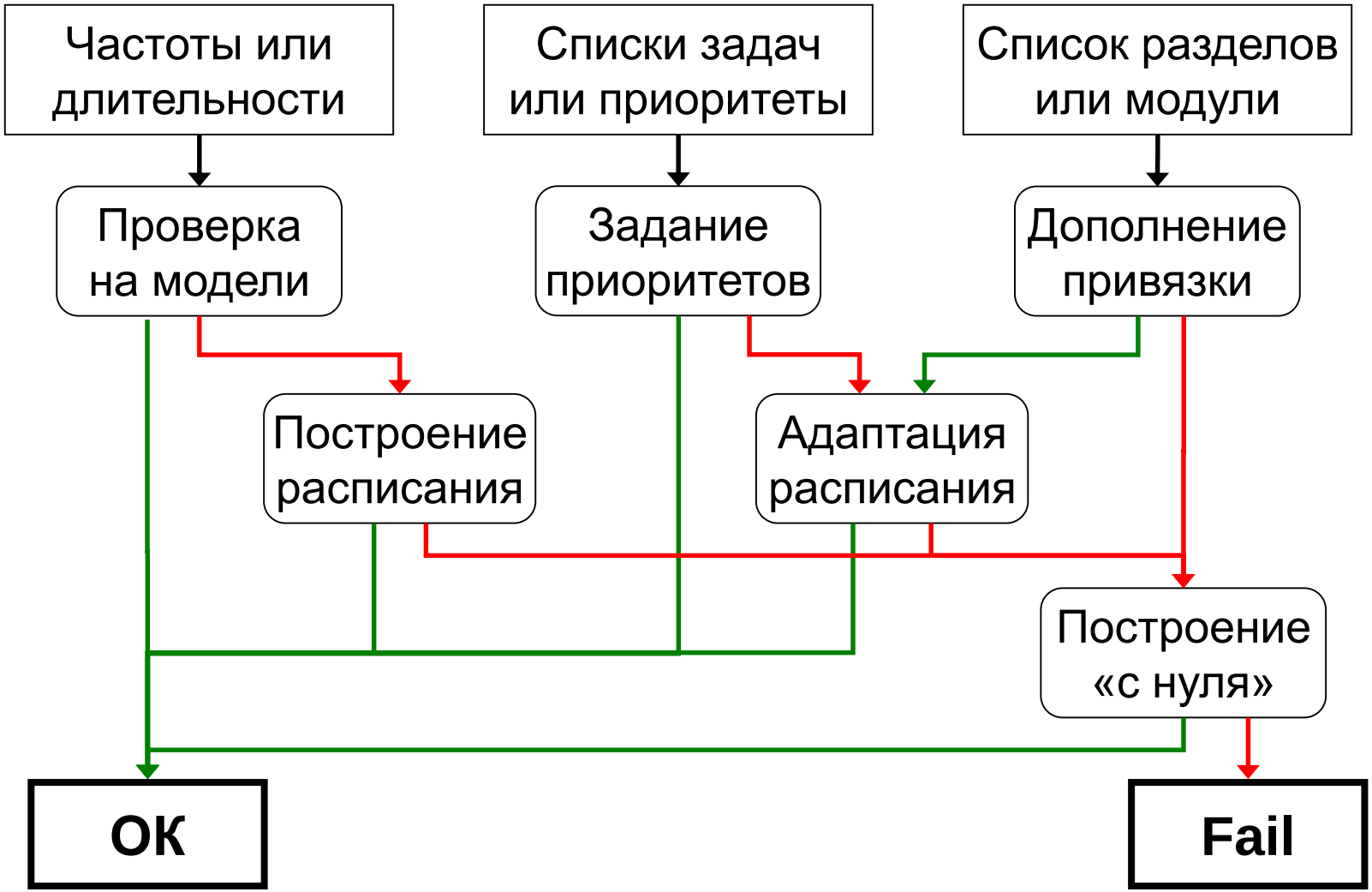
Недостатки модели



Решение:

не выполнять низкоприоритетные задачи, пока есть высокоприоритетные, ожидающие сообщений

Адаптация к изменениям



Перспективные задачи

- Построение расписания с минимизацией затрат на переключение контекста
- Корректировка распределения разделов по ядрам при неуспешном построении расписания
- Построение расписания, гарантированно корректного при временах выполнения задач, меньших чем WCET
- Минимизация сетевой загрузки с учётом синхронности зависимостей по данным
  - сообщения, которые ожидает задача-получатель, следует передавать через память (=> привязка отправителя и получателя к одному модулю)
- Совместное планирование задач и конфигурирование сети

Спасибо за внимание