Занятие 3
Архитектура приложений реального времени

Разделы:

A. Архитектура приложений на ведущем и целевом устройствах

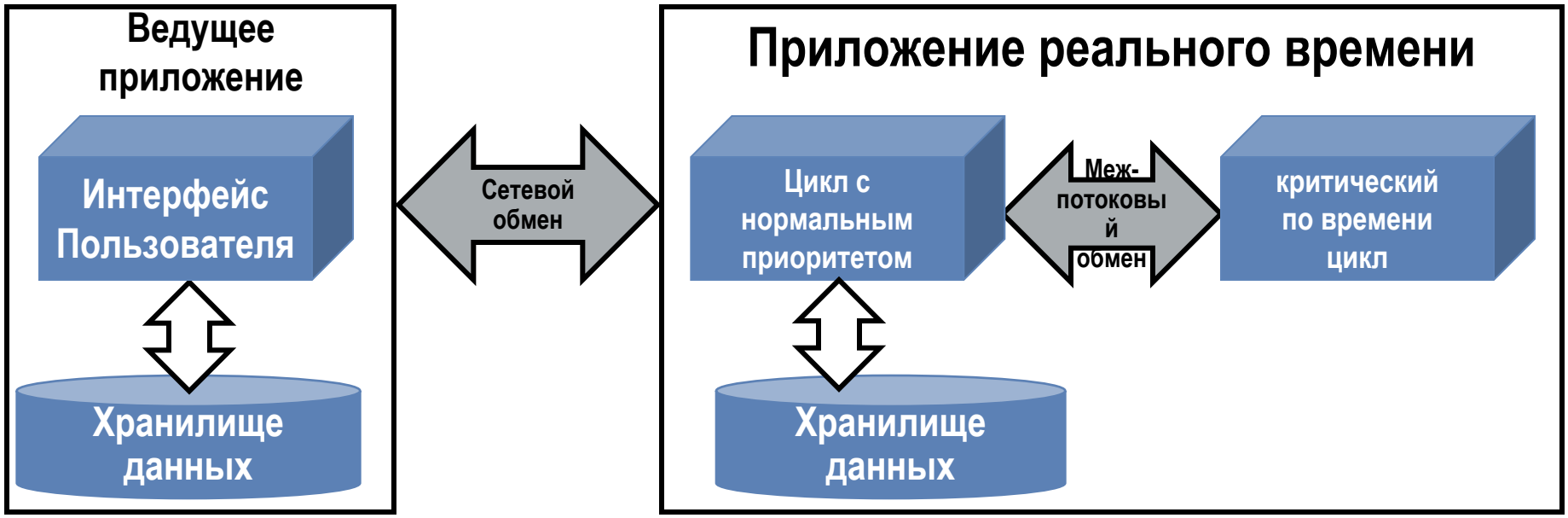
B. Многопоточность

C. Режим ожидания (перерыв)

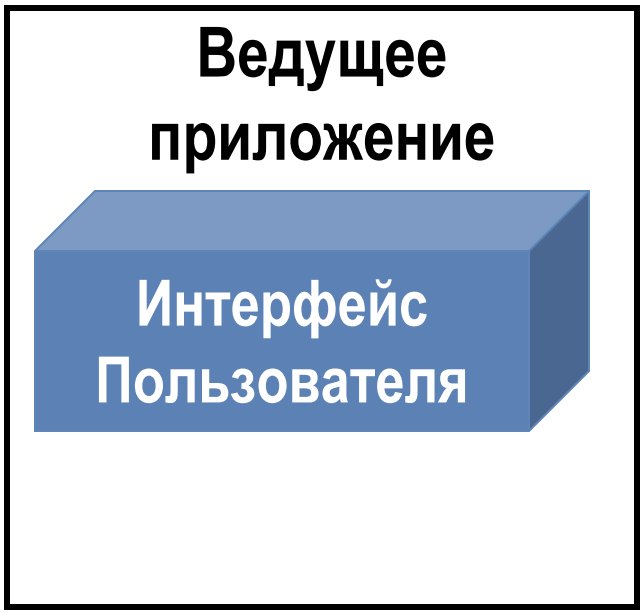
D. Детерминизм

E. Обмен данными между потоками (Threads)

Архитектура приложений реального времени



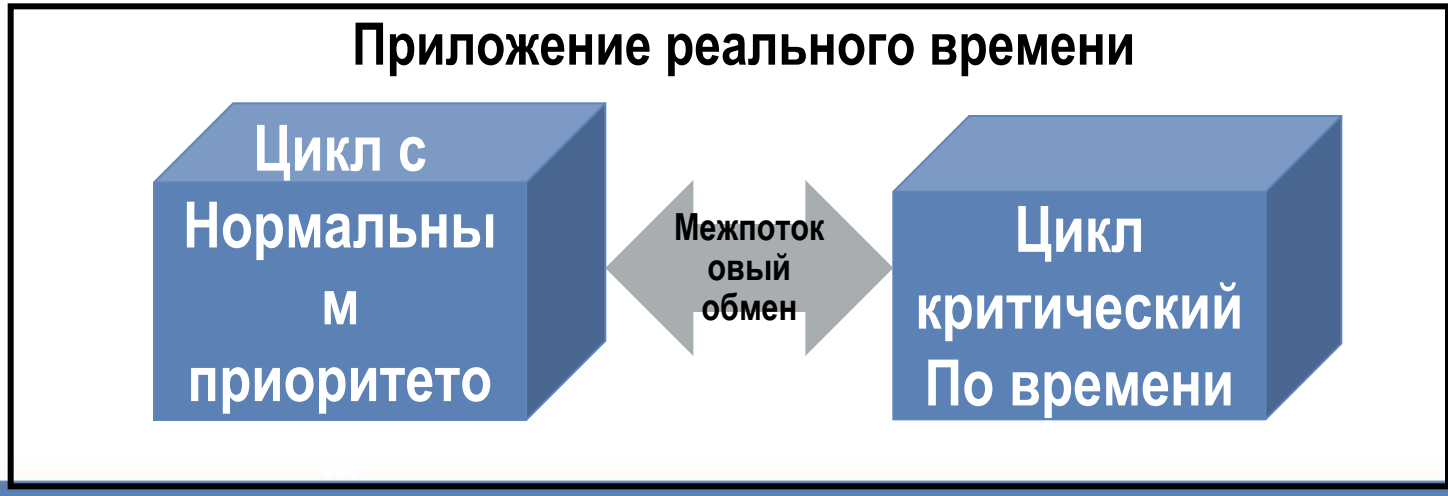
Ведущее приложение



- Выполняется на компьютере
- Отвечает за задачи, не требующие детерминизма:
  - Обмен данными с контроллером
    - параметры поступающие в приложение или отображающиеся на интерфейсе пользователя
    - запрос и получение результатов
  - Запись данных на жесткий диск
  - Анализ данных
  - Сетевой обмен и пересылка данных

Приложение реального времени

- Процессы выполняются с высоким и низким приоритетами. Процесс, который должен быть детерминирован по времени, имеет критический приоритет – все остальные процессы имеют более низкий приоритет
- Многопоточность (multithreading) позволяет задать приоритет каждого процесса.



Что такое многопоточность?

- Расширение подхода многозадачности
  - Многозадачность – возможность операционной системы быстро переключаться между выполняемыми задачами
  - Задача это, как правило, целое приложение, например, такое как LabVIEW
- Многопоточность расширяет возможности многозадачности при работе с приложением:
  - Отделяет определенные операции, выполняемые приложением и помещает их в потоки
  - Делит процессорное время между потоками
  - Позволяет назначать приоритеты

Преимущества многопоточности

Разделение между задачами критичными и некритичными по времени.

Некоторые задачи критические по времени

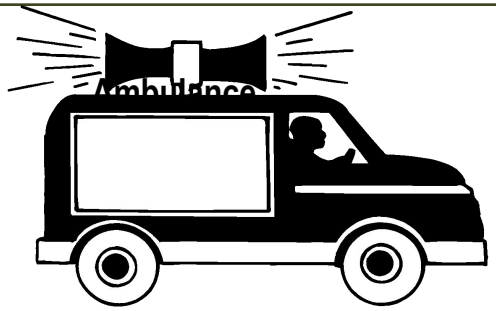
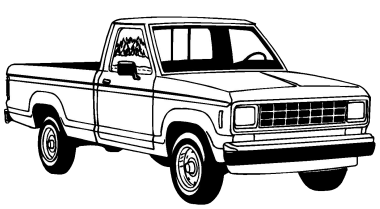
- Цикл управления
- Безопасный мониторинг

Некоторые задачи некритические по времени

- Сетевой обмен
- Запись данных на диск

Требования реального времени заставляют операционною систему распределять процессорное время по приоритетам и в соответствии с жестким расписанием.

Простая модель многопоточной системы реального времени

	Приоритет критический по времени (только один VI)	Нормальный приоритет
Задачи		 

Оператор
(Операционная система)

Один механик
(Процессор)

Расписание Потоков

Циклическое расписание

Потоки одинакового приоритета получают одинаковые кванты времени при обработке на центральном процессоре. Может потребоваться несколько кругов (циклов) полного выполнения задачи за которую отвечает выделенный поток

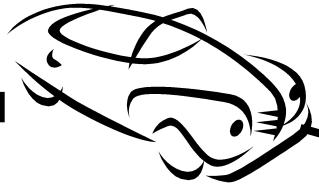
Расписание по приоритетному прерыванию

Поток с высоким приоритетом мгновенно приостанавливает обработку всех потоков с более низкими приоритетами

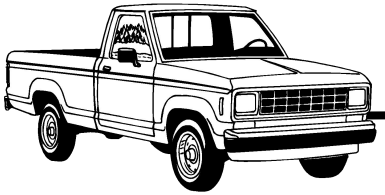
Циклическое расписание

Оператор
(планировщик)

Нормальный

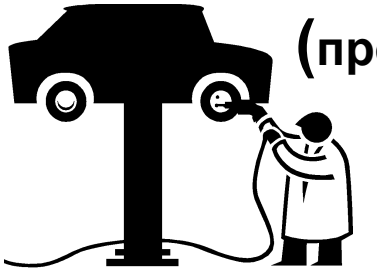


Нормальный



Нормальный

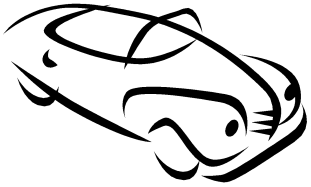
Механик
(процессор)



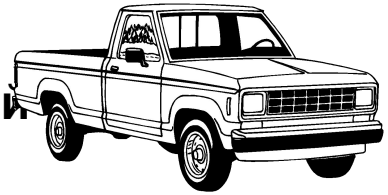
Расписание по приоритетному прерыванию

критический по времени

Нормальный



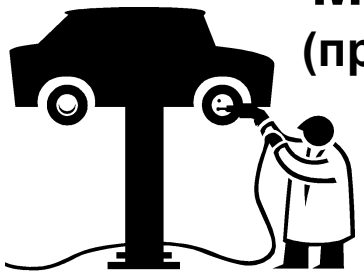
Нормальный



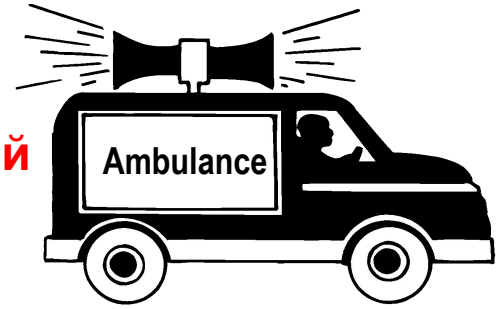
Нормальный

Оператор
(планировщик)

Механик
(процессор)

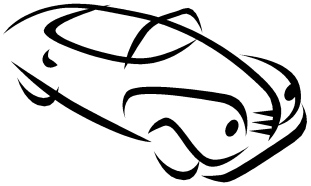


Расписание LabVIEW Real-Time

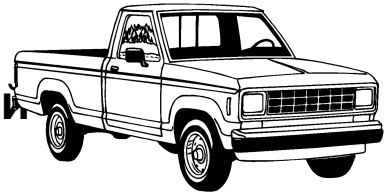


критический

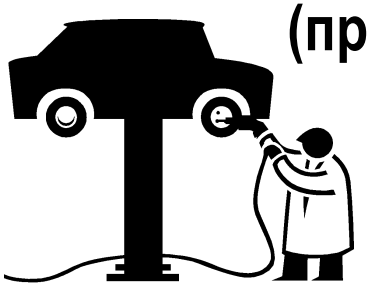
Оператор
(планировщик)



Нормальный



Нормальный

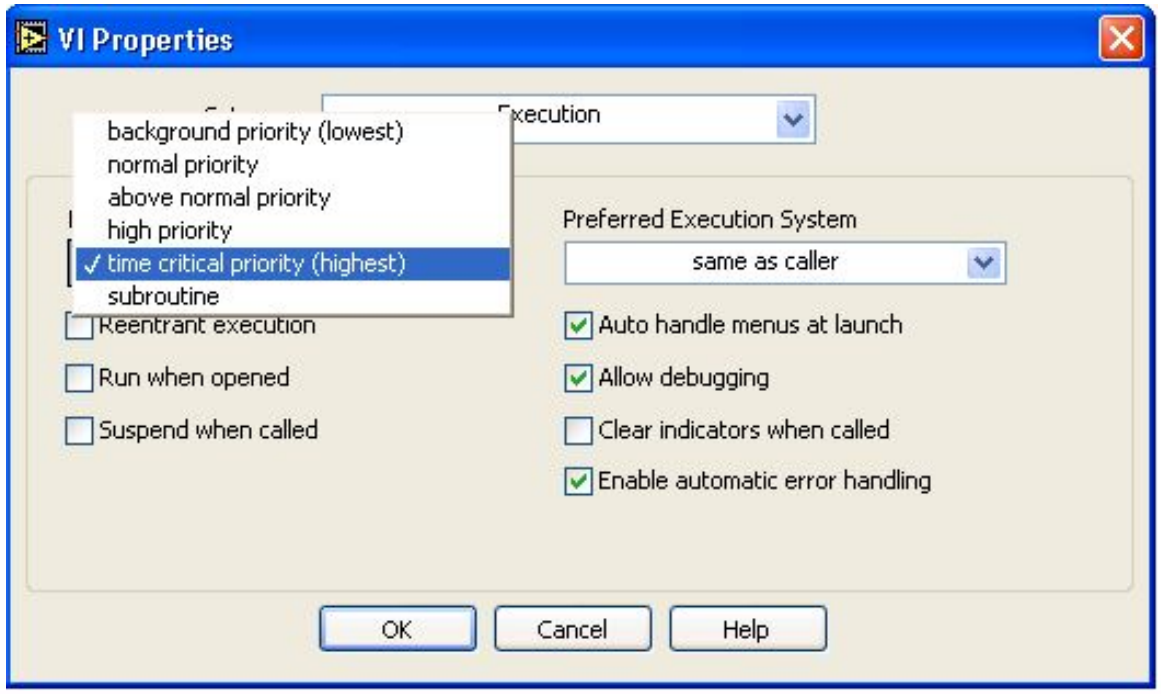


Механик
(процессор)

Нормальный

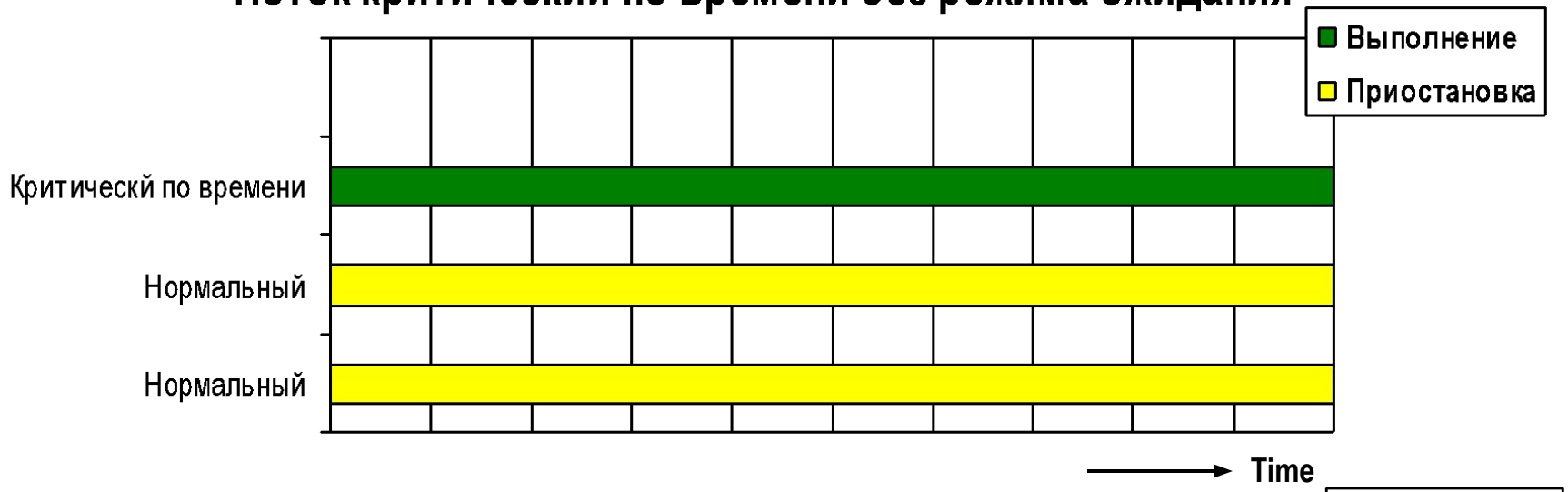
Планировщик LabVIEW Real-Time

- Комбинация циклического расписания и расписания по приоритетному прерыванию.
- Назначение одного критичного по времени ВП



Режим ожидания

Поток критический по времени без режима ожидания



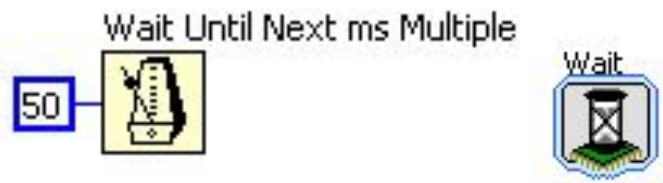
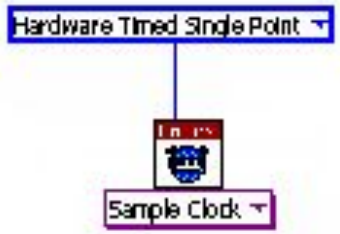
Режим ожидания добавлен в поток критический по времени

Зависание


Зависание потков из-за отсутствия режима ожидания
Выполнение
Приостановка
A. Критический по времени
B. Выше Нормального
C. Нормальный
Time
Режим ожидания добавлен—все еще возможно зависание отдельных потоков
Перерыв
Выполнение
Приостановка
A. Критический по времени
B. Выше Нормального
C. Нормальный

Режим ожидания

Приостановка выполнения VI или потока

Программный метод	Аппаратный метод
	
Использует таймер операционной системы для задания времени выполнения цикла программы.	Использует аппаратные таймеры или таймер процессора для задания времени выполнения цикла.

Режим ожидания и критический по времени приоритет

Следующие особенности уникальны для LabVIEW Real-Time:

- Если любой **критический по времени** VI прерывает свое исполнение в режиме ожидания, то **весь поток** переводится в режим ожидания.
- Не используйте параллельные циклы в критических по времени VI, многозадачность не будет работать.
- Если параллельные циклы имеют разную частоту, не используйте критические по времени VI — используйте специальный циклы с тактированием (timed loop)

Упражнение 3-1

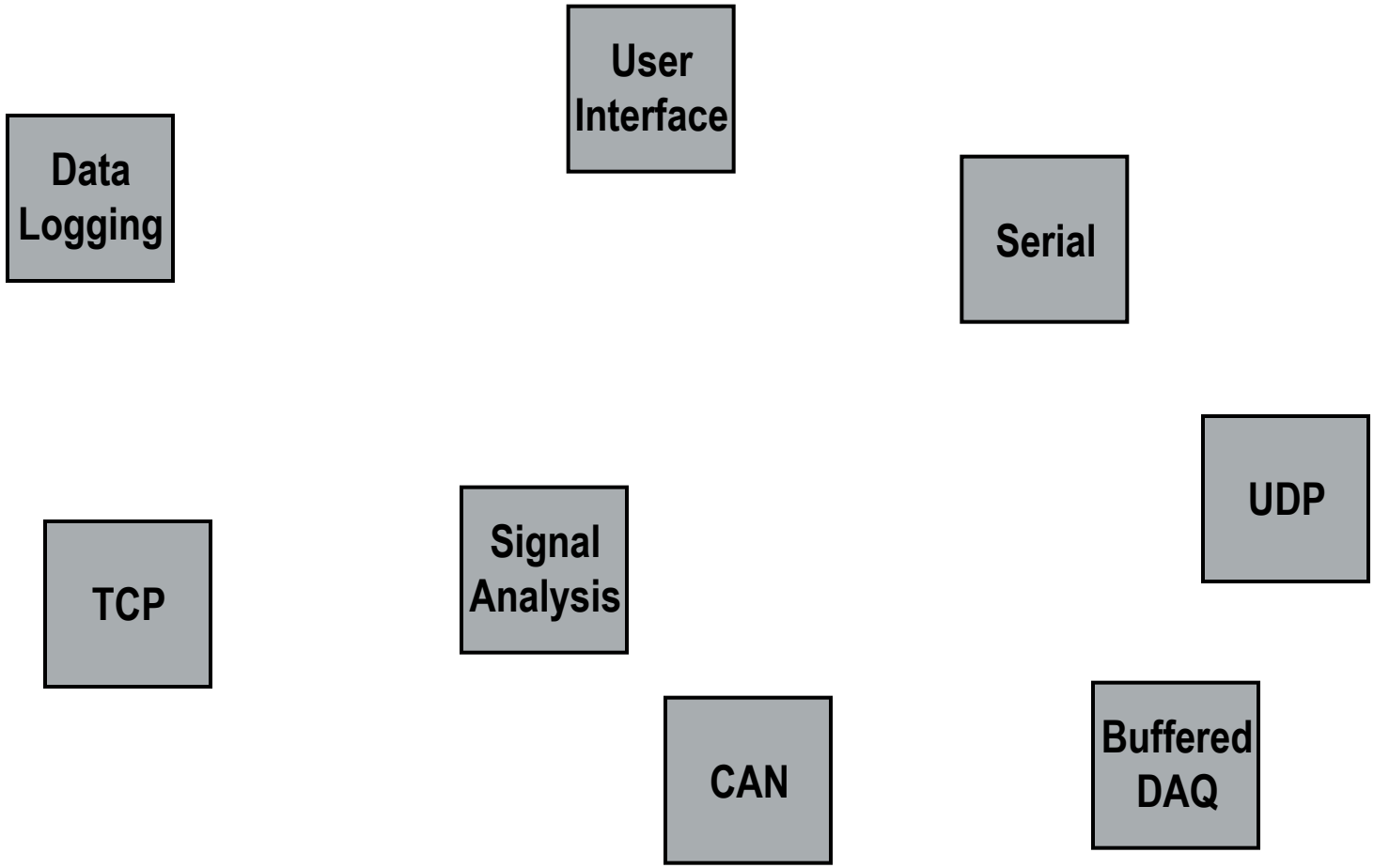
Работа с приоритетами

Время на выполнение: 10 минут.

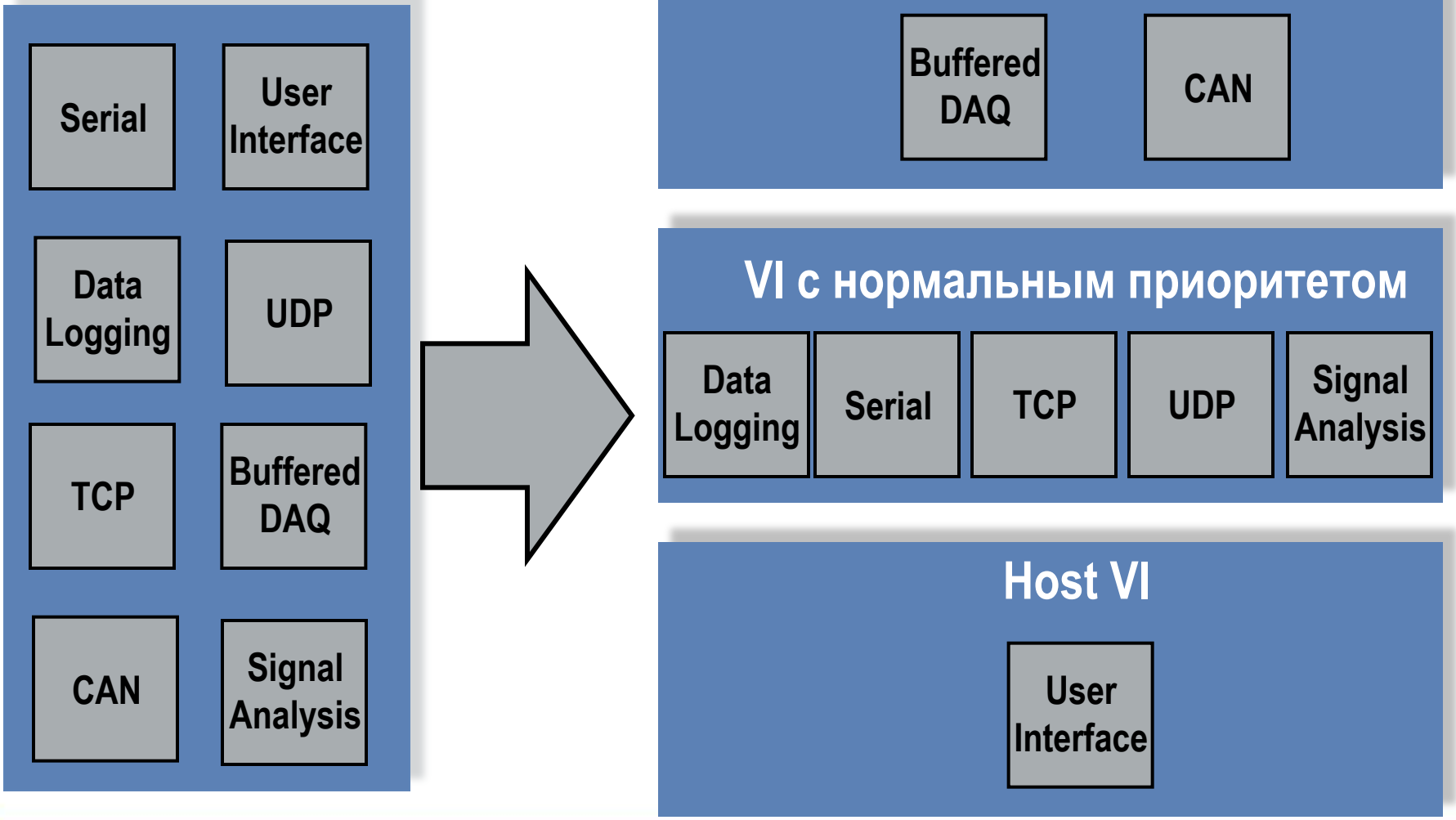
OBJECTIVE

Настроить приоритеты и освоить влияние приоритета на работу VI

Упражнение в классе – выбор приоритета



Возможное Решение



Детерминизм. Оптимизация приложения.

- Выбирайте подходящие аппаратные средства
- Избегайте использования ресурсов с общим доступом
- Избегайте перевыделения памяти
- Избегайте вызовов subVI в цикле
- Отключайте ненужные опции
- Используйте только один VI с критическим приоритетом

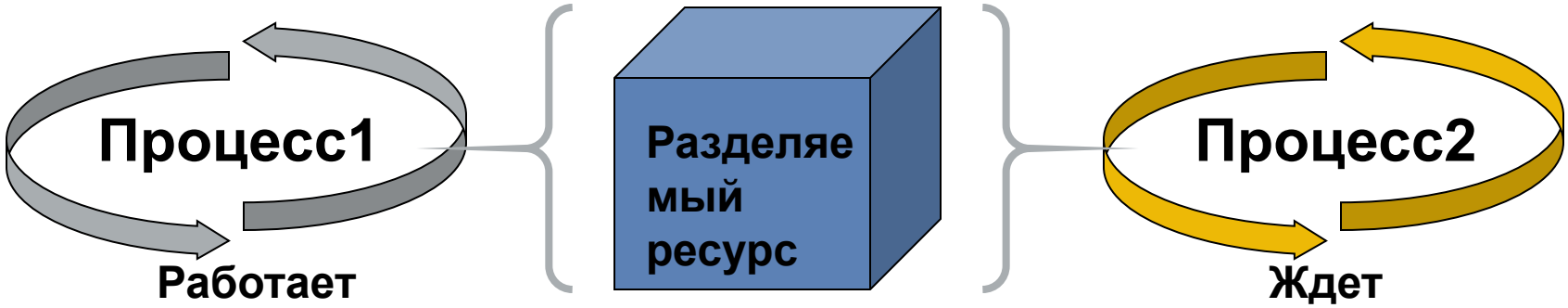
Ресурсы с общим доступом

- *Ресурсы с общим доступом* в LabVIEW Real-Time - это ресурсы, которые могут быть использованы только одним процессом в один и тот же момент времени
- Ресурсами с общим доступом являются:

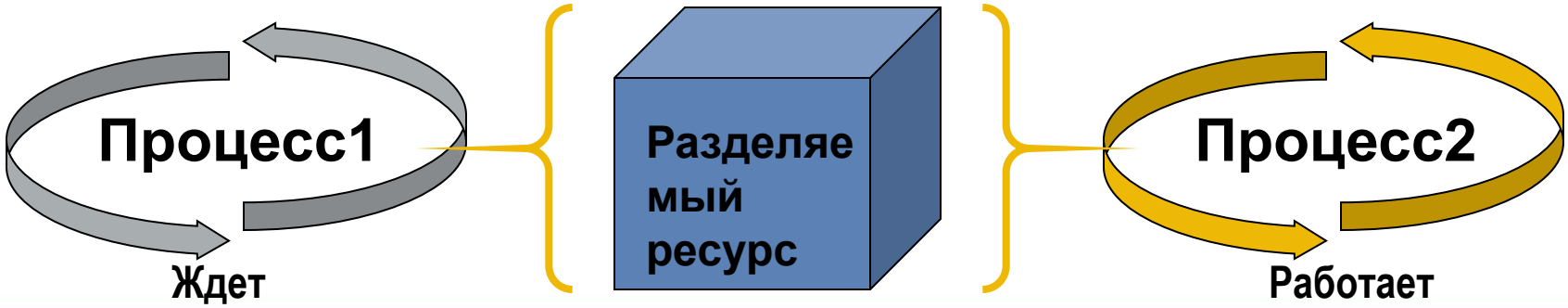
- Глобальные переменные
- Менеджер памяти Real-Time
- Однопоточные DLL
- Сетевой код (TCP/IP, UDP, VI Server) *
- Переменные с общим доступом (Shared Variable)
- subVI, не поддерживающие параллельные вызовы
- Семафоры
- Файлы

Ресурсы с общим доступом

Перед тем как процесс начнет использовать ресурс он должен получить мьютекс (mutex) – флаг предотвращения одновременного доступа



После того как процесс1 завершит работу, процесс2 может начать работу



Ресурсы с общим доступом -приоритеты

Критический приоритет *ждет*

Инверсия приоритетов:

VI с нормальным приоритетом блокировал VI с критическим приоритетом, захватив мьютекс разделяемого ресурса

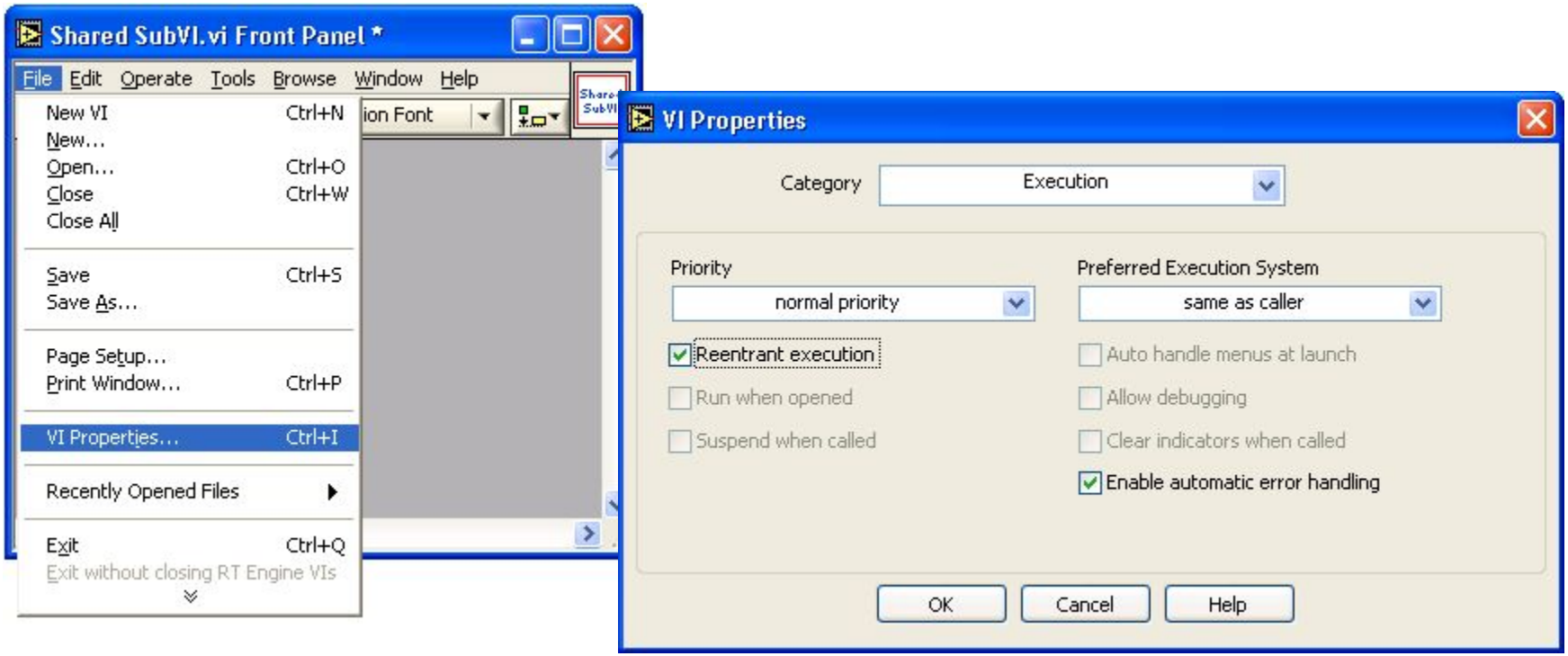
Нормальный приоритет *работает*

Наследование приоритета:

VI с нормальным приоритетом наследует более высокий приоритет до освобождения мьютекса

Ресурсы с общим доступом – (subVI)

Настройте subVI для поддержки параллельных вызовов

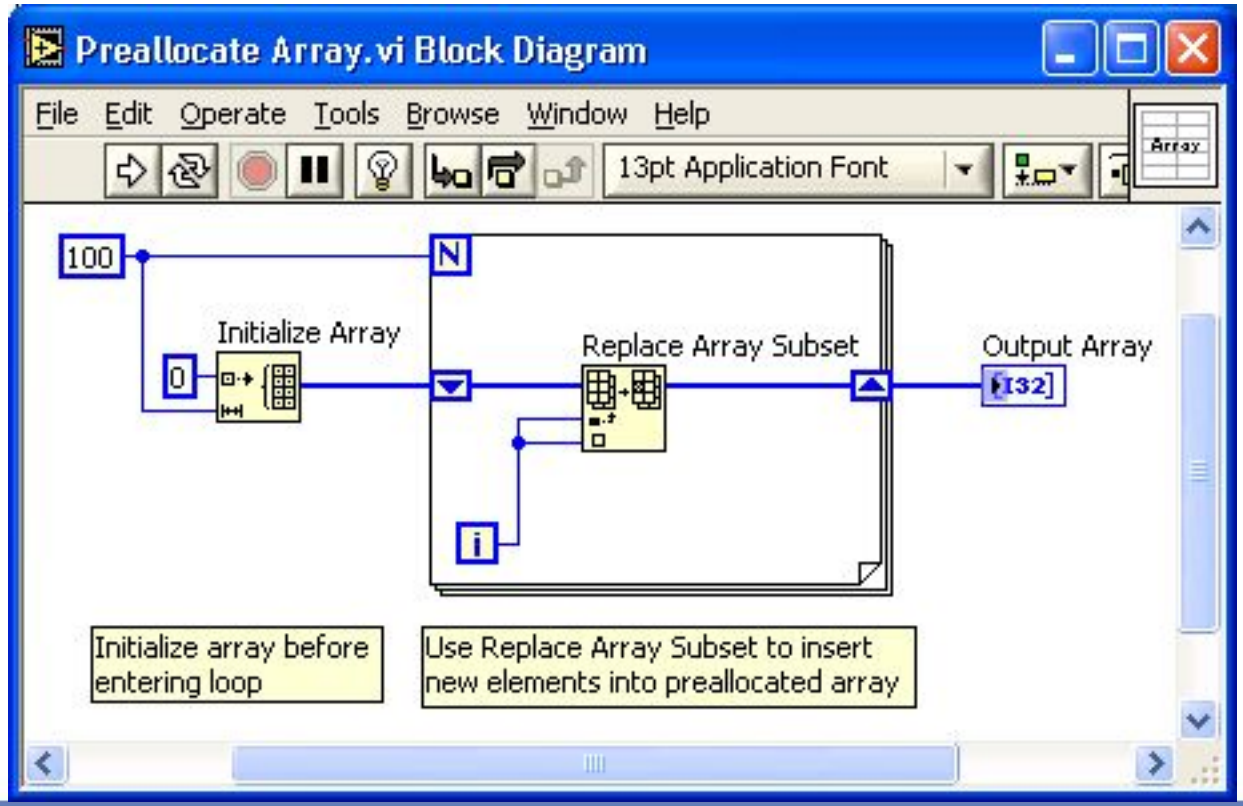


Ресурсы с общим доступом – менеджер памяти

- Менеджер памяти LabVIEW Real-Time выделяет память автоматически
  - Пользователь не должен явно резервировать или освобождать память
  - Это означает, что менеджер памяти просто использовать, но им трудно управлять
- Менеджер памяти LabVIEW Real-Time это разделяемый ресурс
  - Вы должны управлять выделением памяти для того, чтобы избежать конфликтов (свойственных разделяемым ресурсам) с менеджером памяти.
  - Необходимо статическое выделение памяти перед запуском процесса критического по времени

Инициализация массивов

Избегайте динамического выделения памяти для массивов в цикле с критическим приоритетом.



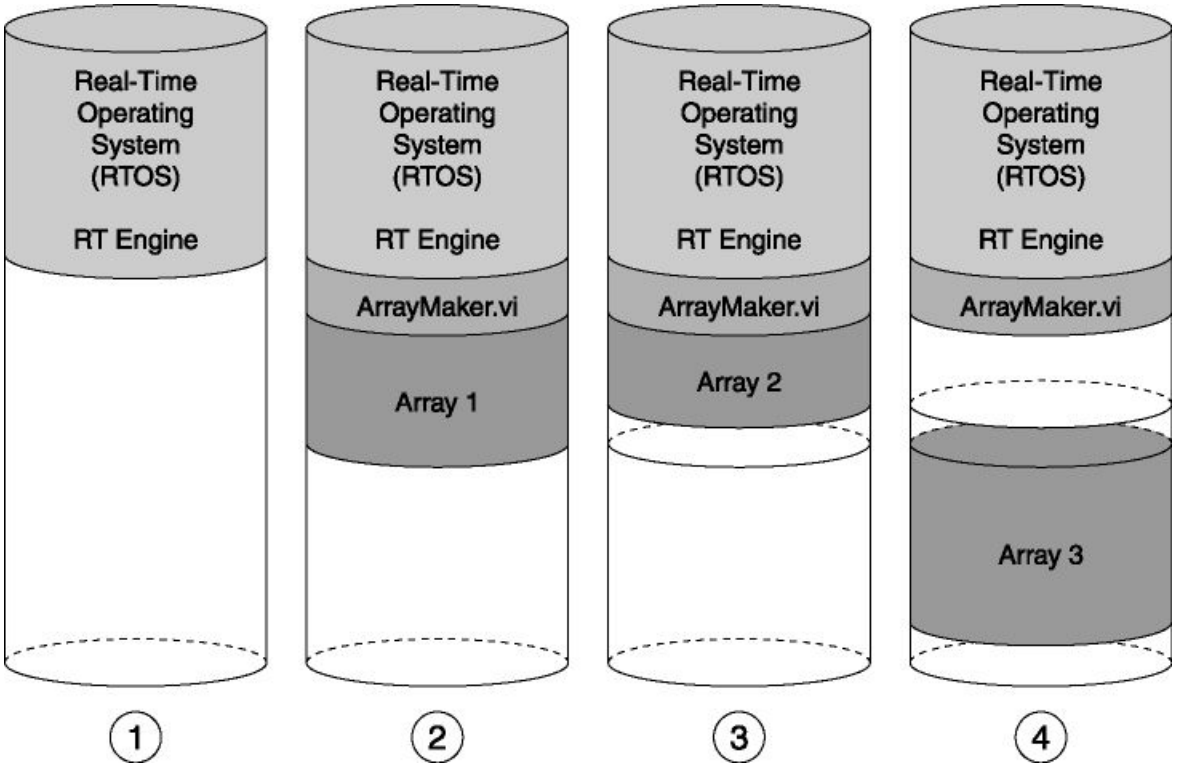
Ресурсы с общим доступом – управление памятью

Ключевые моменты по управлению памятью в системах реального времени:

- Менеджер памяти LabVIEW Real-Time является ресурсом с общим доступом
- Все операции по выделению памяти должны проводиться вне цикла с критическим приоритетом.
- Резервирование памяти под массивы должно выполняться вне цикла с критическим приоритетом
- Данные должны быть приведены нужному типу
- По возможности используйте метод замещения (Replace) для повторного использования памяти буферов
- Старайтесь как можно меньше использовать глобальные переменные

Разделяемые ресурсы, do not delete – used for notes

Избегайте конфликтов памяти



Одна и та же область памяти используется для массивов меньших по размеру чем Массив1 (Array1)

Так как Массив3 больше Массива1, должен быть найден другой непрерывный участок памяти be found

Резервируемый под массивы участок памяти должен быть по размеру не меньше размера самого большого ожидаемого массива.

Избегайте конфликтов памяти, cont.

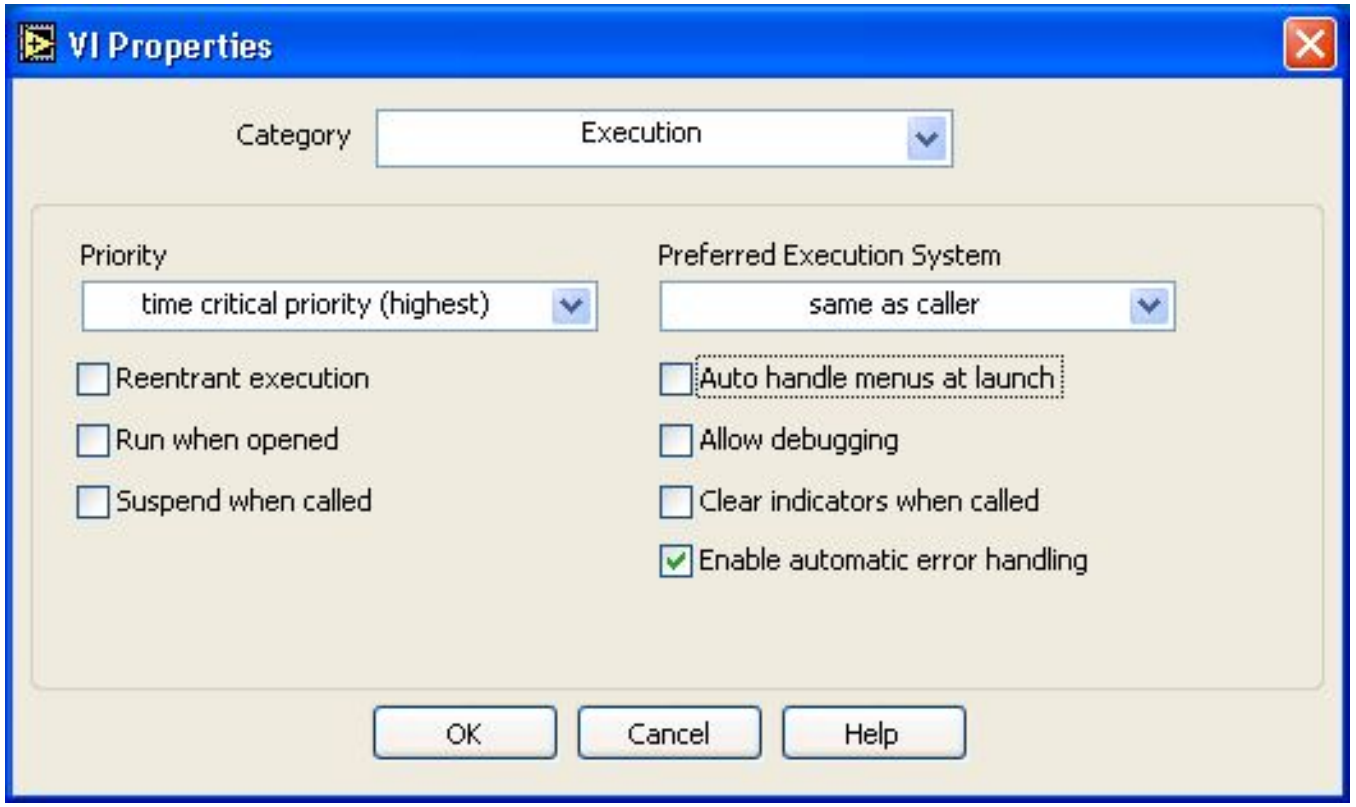
- Do not delete this slide; it is used for the manual notes.

Избегайте вызовов subVI в цикле

- Каждый вызов subVI приводит к определенным затратам на выполнение служебных операций
- затраты могут быть значительными при вызове subVI из тела цикла
- Вместо этого, если возможно, поместите тело цикла внутрь подпрограммы subVI

Отключите ненужные опции

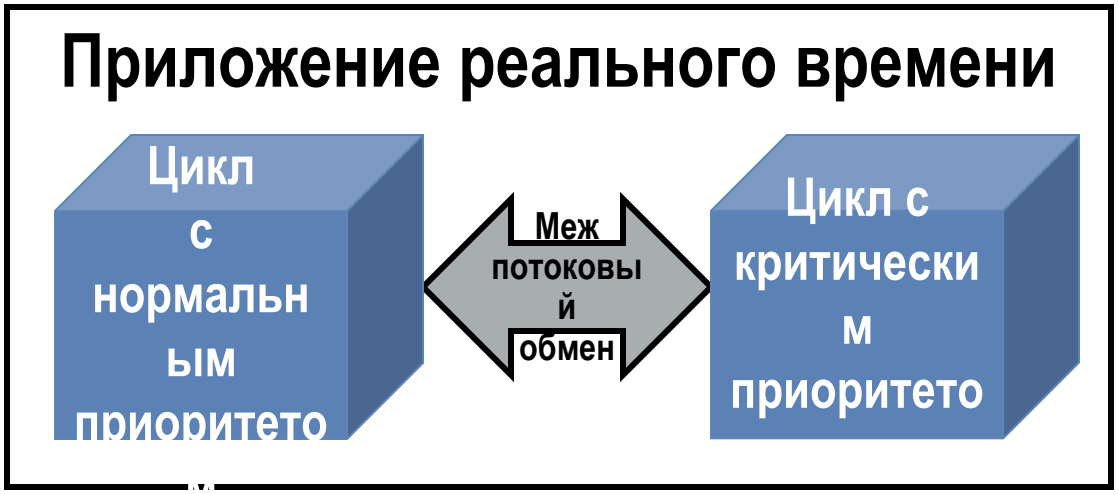
- Разрешить отладку (Allow debugging)
- Авто обработка меню при запуске (Auto handle menus at launch)



Избегайте использования Express VI

- Express VI в LabVIEW просты в использовании и ускоряют разработку
- Требуют дополнительных ресурсов для вспомогательных операций во время выполнения

Обмен данных между потоками



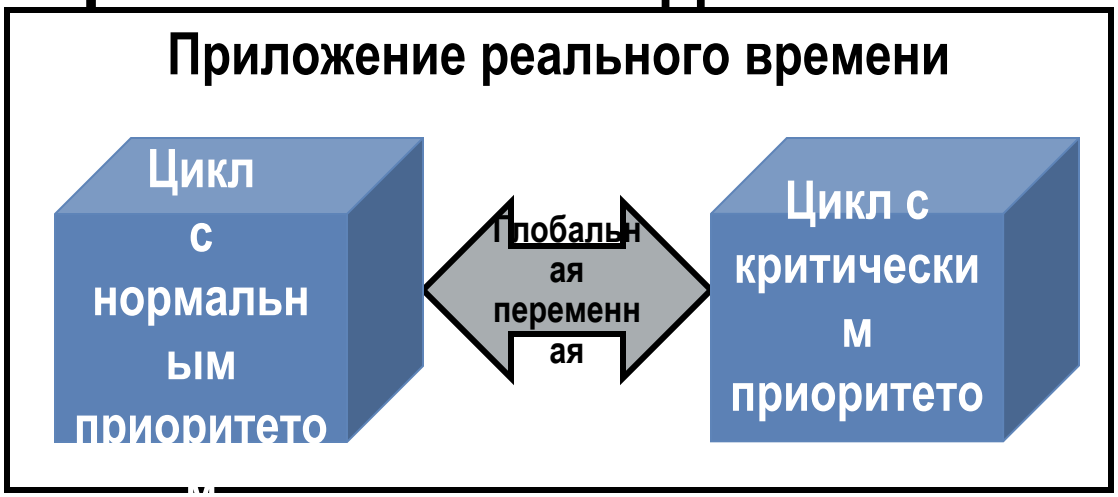
Методы обмена данными между потоками

Удовлетворительный: Глобальные переменные

Хороший: Функциональные глобальные переменные

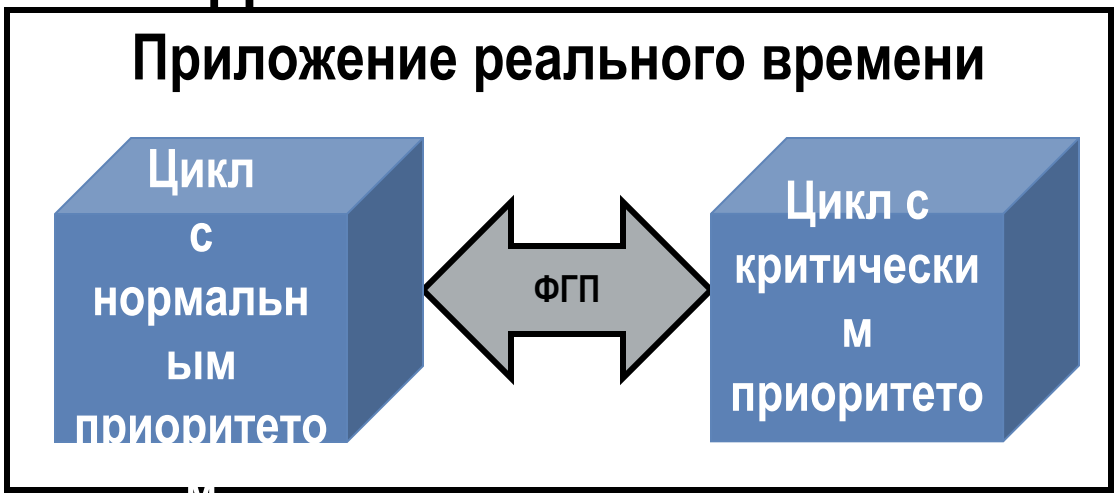
Отличный: буферы Real-Time (RT FIFO), буферы в переменных с общим доступом (Shared-Variable FIFOs)

Глобальные переменные (Global Variables) – Удовлетворительный метод



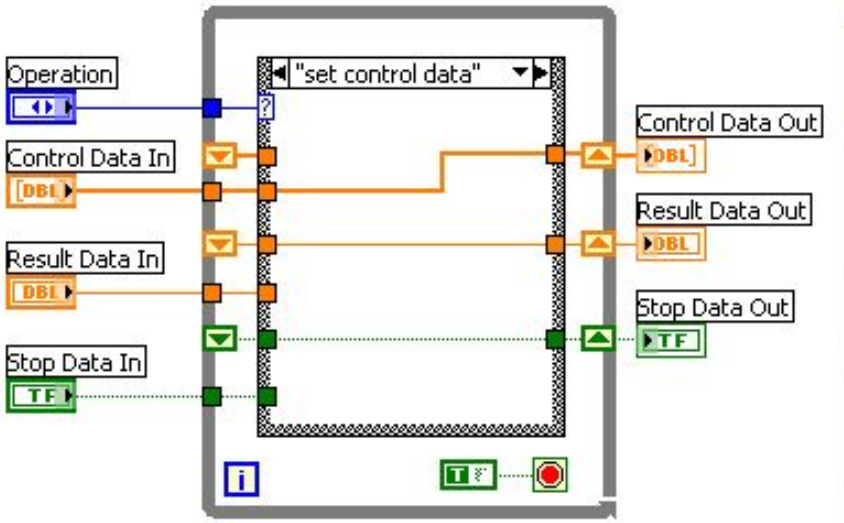
- Может приводить к возникновению джиттера, так как представляет собой разделяемый ресурс
- Может приводить к потере данных – запись в глобальную переменную может производиться несколько раз до того момента, когда произойдет чтение.
- Удовлетворительный метод для скалярных данных (<32 bits)

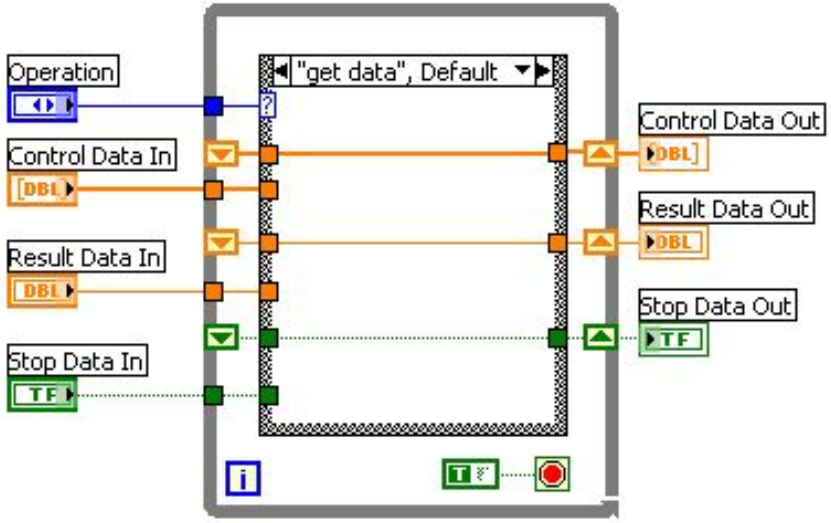
Функциональные глобальные переменные (ФГП) – Хороший метод



- Может иметь несколько входов и выходов
- Может быть *пропущена, если занята (skip if busy)*
- Может приводить к потере данных при передаче

ФГП

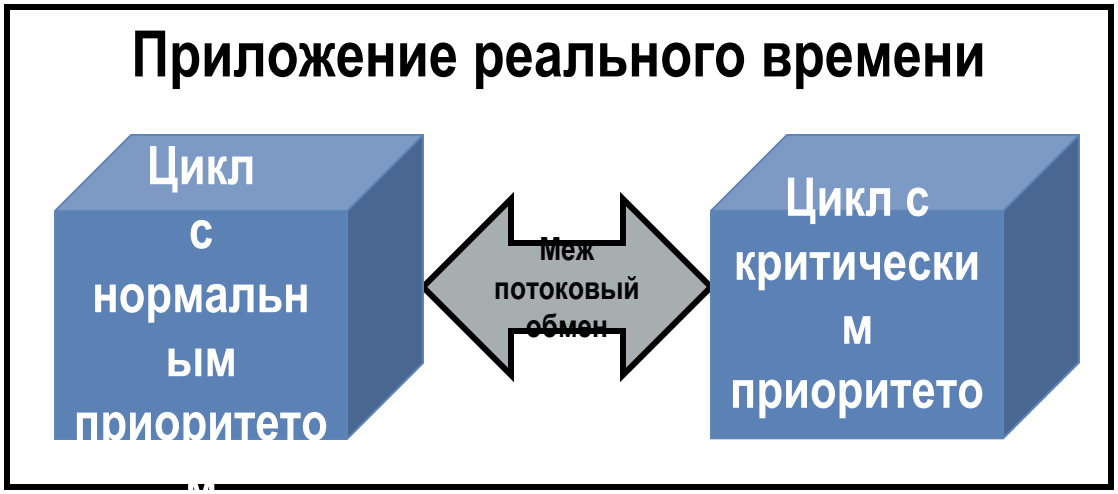




Пример: NI Example Finder

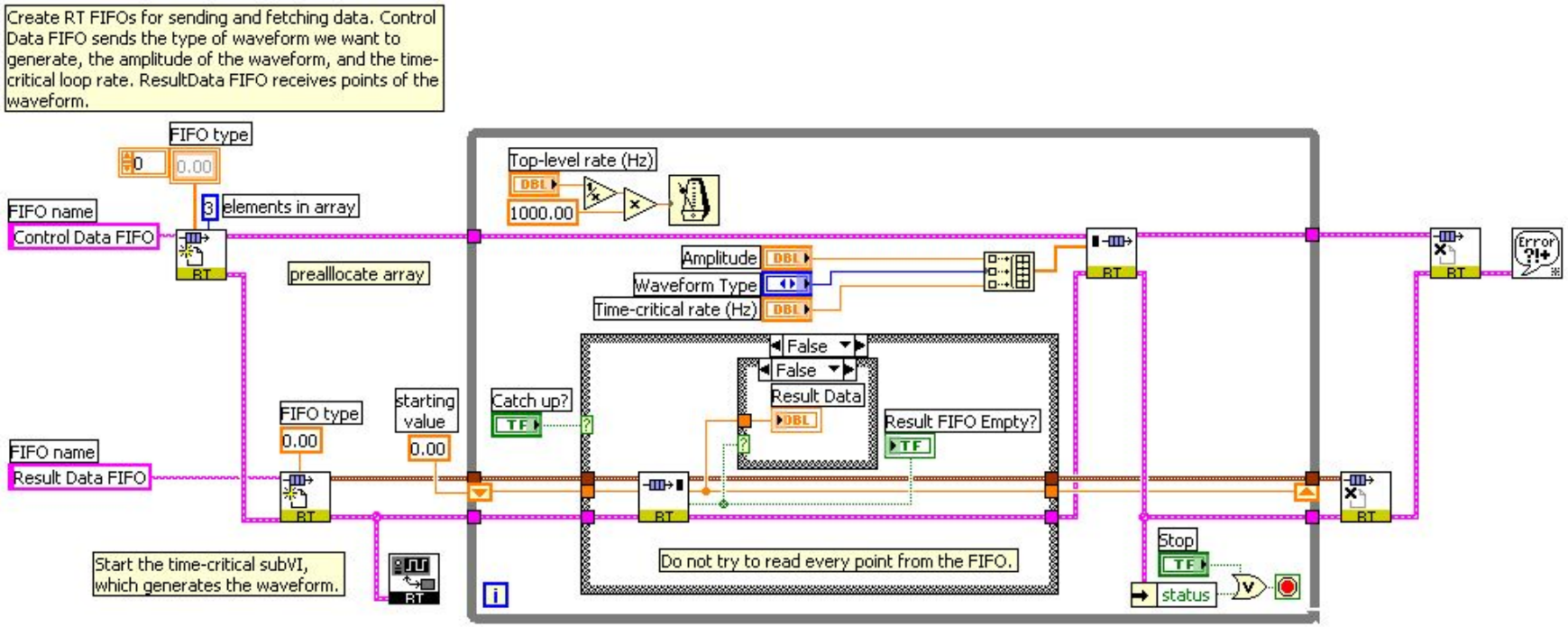
Toolkits and Modules»Real-Time»Multithreaded Communication»
Functional Global Communication

Буферы (Real-Time FIFO) — самый лучший метод



- Фиксированный размер буфера
- Предупреждения о потере данных
- Детерминированная передача данных

Буферы Real-Time FIFO



Пример: NI Example Finder

Toolkits and Modules»Real-Time»Communication»RT FIFO Communication

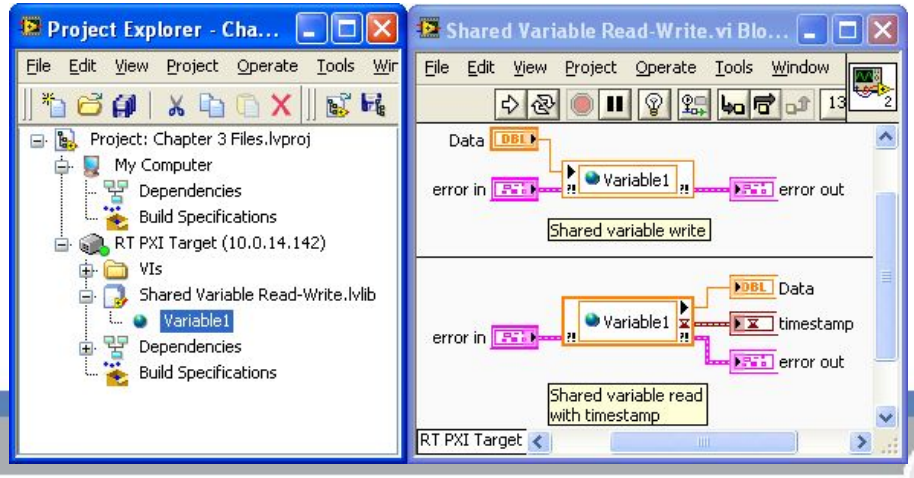
Переменные с общим доступом (shared variables)

Переменные с общим доступом являются универсальным средством передачи данных:

- Недетерминированная передача данных межу VI (глобальная переменная)
- Недетерминированная передача данных межу ведущим и целевым устройствами
- Недетерминированная передача данных между ведущими устройствами
- Детерминированная передача данных между циклами реального времени (Real-Time FIFO)
- Детерминированная передача данных между целевыми устройствами

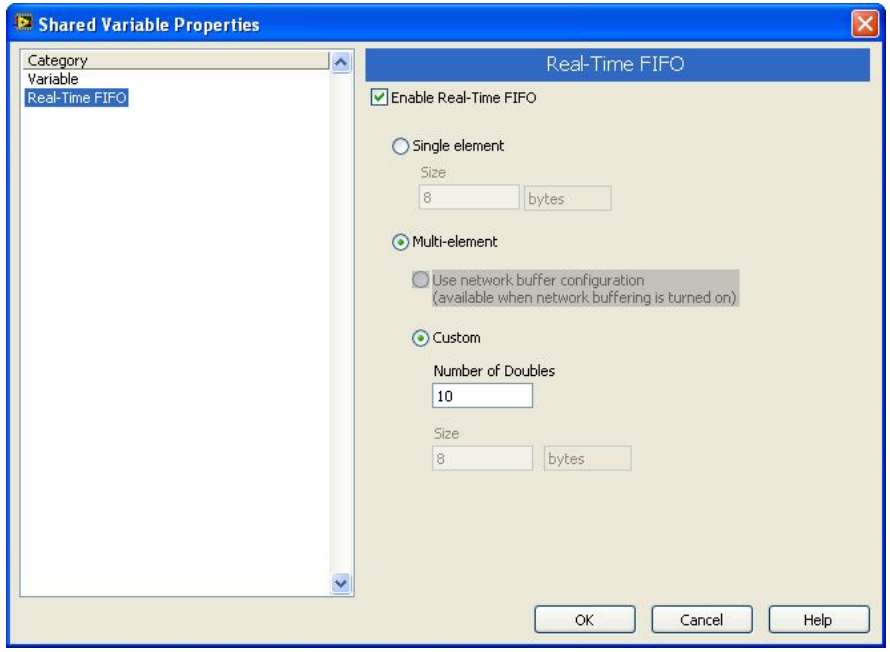
Создание и использование переменных с общим доступом

- Щелкните правой кнопкой мыши на проекте или библиотеке для создание переменной. Переменная должна принадлежать какой-либо библиотеке
- Диалог Shared Variable Properties позволяет задать конфигурацию переменной
  - Выберите Однопроцессорный (Single-process), Публикуемый в сети (Network-published), Синхронизируемый (Time Triggered)
  - Задайте дополнительные опции
- Переменная с общим доступом на блок диаграмме используется схожим образом с глобальной перемененной за следующими исключениями:
  - Переменная с общим доступом имеет терминалы состояния ошибки
  - Переменная с общим доступом может возвращать временную метку (timestamp)



Переменные и Real-Time FIFOs

Вы можете создать переменную, которая использует буфер Real-Time FIFOs для передачи данных. Для этого надо выбрать опцию Enable Real-Time FIFO.

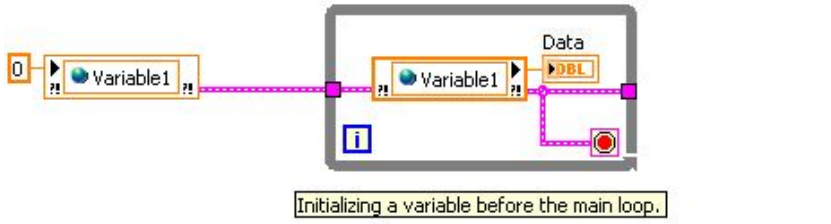


RT FIFO Enabled

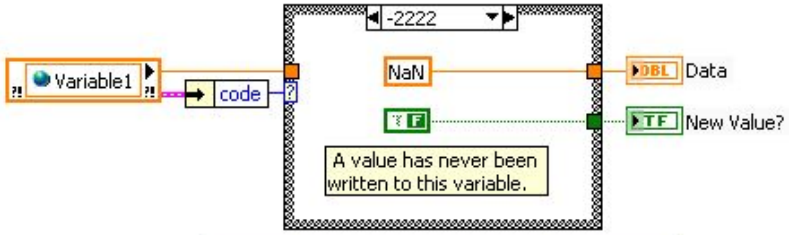
Техника программирования

Инициализация

- Буфер создается в момент первого чтения или записи в переменную
  - Создайте и инициализируйте буфер, записав значение в переменную перед основным циклом
- Попытка читать из буфера перед тем, как туда будет записана какая- либо величина, приведет к ошибке **–2222** и возврату значения по умолчанию
  - Проверяйте состояние ошибки для неинициализированных переменных.



Initializing a variable before the main loop.

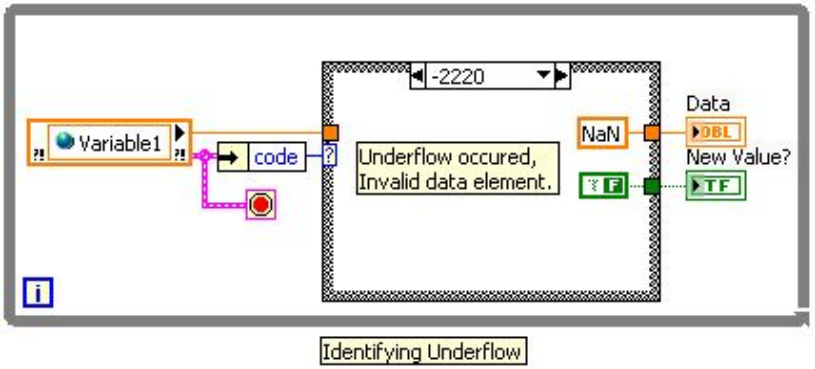


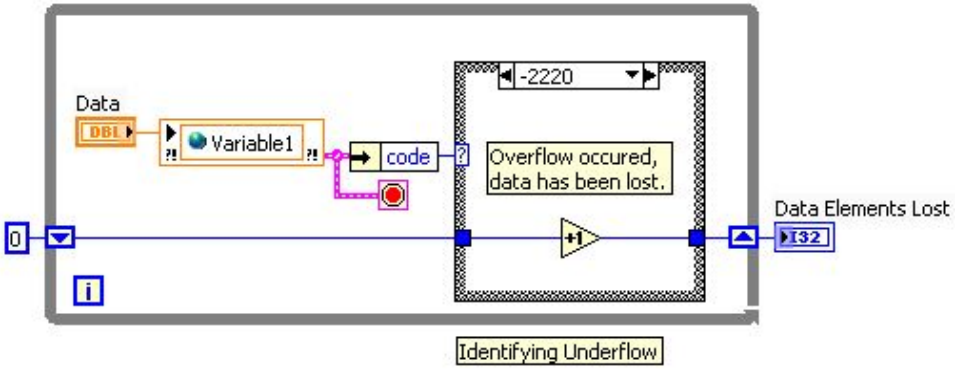
Checking a variable to see if it has ever been written.

Техника программирования

Идентификация переполнения (Overflow) и опустошения (Underflow)

- Переполнение возникает, когда производится попытка записи в буфер, который уже полон. Непрочитанные данные сбрасываются для того чтобы освободить место для новых данных
  - Ошибка –**2221** возникает при переполнении очереди
- Опустошение возникает, когда производится попытка чтения из пустой очереди. Возвращается значение по умолчанию
  - Ошибка –**2220** возникает при опустошении очереди

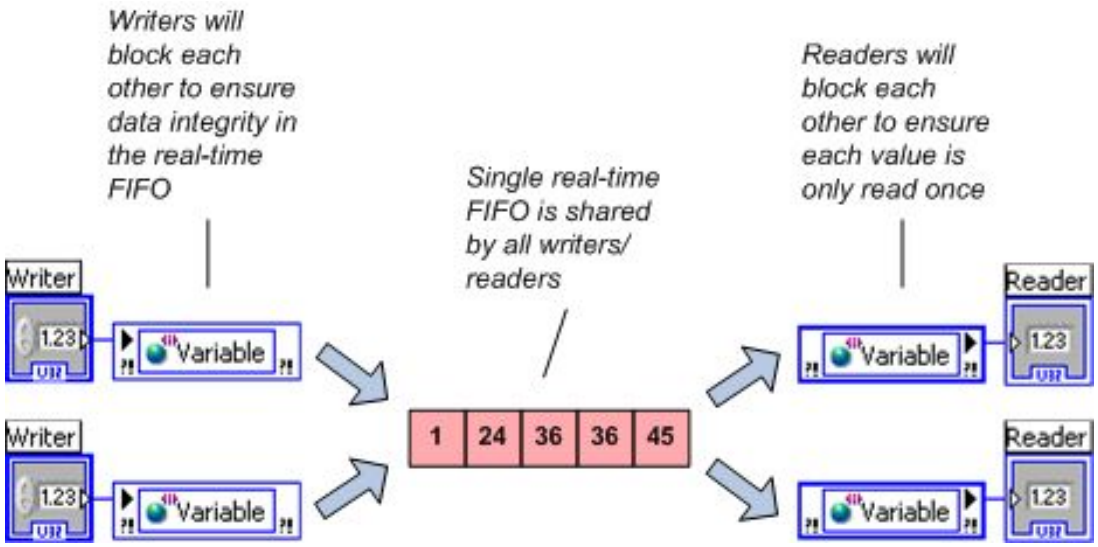




Техника программирования

Несколько операций считывания и записи

- Попытка проведения нескольких считываний или записей одновременно приводит к блокированию однотипных операций
- Попытка проведения нескольких считываний из буфера одновременно приводит к тому, что каждая операция будет считывать и удалять элементы попеременно, предотвращая возможность считывания всех данных в ходе одной операции.

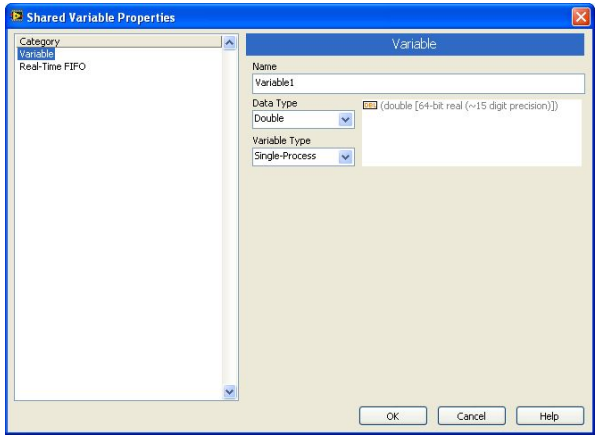


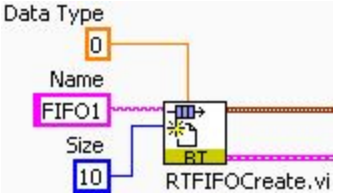
VI для работы с буферами и переменные с общим доступом

	Переменные	VI для работы с буферами
Конфигурирование	Статическое	Динамическое
Производительность	Операция записи должна хранить временную метку (timestamp)	Операция записи производится быстрее
Программирование	Проще конфигурируется и применяется в обычных случаях	Проще проводится динамическое изменение конфигурации
Дополнительные возможности	Доступна временная метка (Timestamp), которая может быть преобразована к другим типам данных	Совместимость с LV 7.*x* и более ранними версиями

Сравнение статического и динамического конфигурирования.

- Переменные конфигурируются статически, при помощи окон диалога
  - Простое программирование и подсоединение на блок диаграмме
  - Экономия места на блок диаграмме
- VI для работы с буферами позволяют задавать конфигурацию динамически при помощи кода на блок диаграмме
  - Упрощает чтение кода на блок диаграмме
  - Упрощает конфигурирование во время исполнения программы
  - Улучшает управляемость процессом создания и уничтожения буфера





Упражнение 3-2

Обмен данными между потоками

Время на выполнение: 1 час

OBJECTIVE

Использование ГП, ФГП и буферов Real-time FIFO для обмена данными между потоками. Сравнение перечисленных методов.

Упражнение в классе 3-3

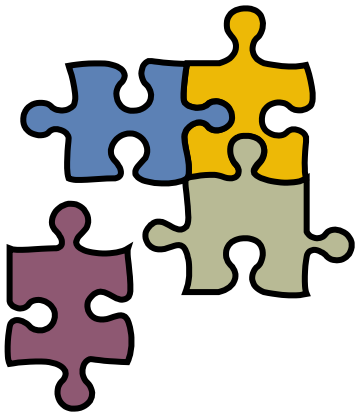


Схема проекта

Время на выполнение: 40 мин.

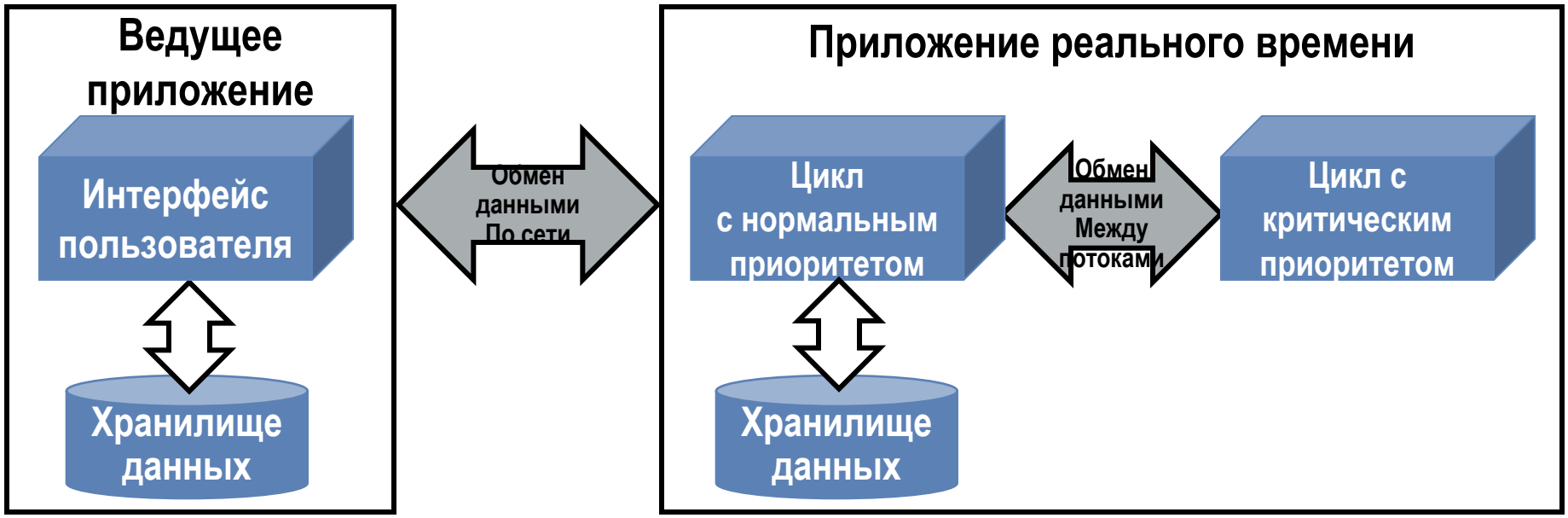
OBJECTIVE

Дается документ с требованиями, требуется создать схему проекта.

Резюме

1. Какие существуют методы для улучшения детерминизма?
2. Какой метод обмена данными между потоками является лучшим с точки зрения детерминизма?
3. Какой метод обмена данными межу потоками самый простой при программировании?
4. Многозадачность запрещена в критическом по времени потоке; как это влияет на выбор методов программирования?

Резюме — архитектура приложения



Резюме — многопоточность

- Многопоточность позволяет отделить критические по времени задачи от некритических
- LabVIEW RT использует комбинированное расписание: циклическое и по приоритетному прерыванию. Потоки с одинаковым приоритетом подчиняются циклическому расписанию
- Уровни допустимых приоритетов (по возрастанию): фоновый (background), нормальный (normal), выше нормального (above normal), высокий (high), критический (time-critical)
- Перерыв в режиме ожидания приводит к приостановке выполнения VI или потока
- Если любой VI в критическом по времени потоке останавливается на перерыв в режиме ожидания, весь поток останавливается на перерыв. Поэтому, рекомендуется использовать только один VI или цикл критический по времени

Резюме

- Методы обмена данными межу потоками:
  - Удовлетворительный: Глобальные Переменные
  - Хороший: Функциональные Глобальные Переменные
  - Отличный: буферы Real-Time FIFO или переменные с общим доступом (с буферами Real-Time FIFO)
- Для улучшения детерминизма:
  - Избегайте использование разделяемых ресурсов
  - Избегайте перевыделения памяти Избегайте вызовов subVI в цикле
  - Отключите ненужные опции