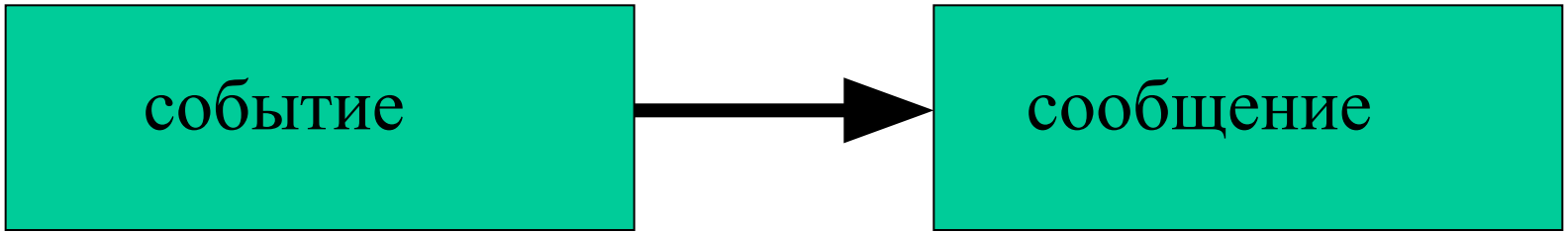
Сообщения Windows

Возникновение сообщений

- Сообщения являются реакцией системы Windows на различные происходящие в системе события:
  - движении мыши,
  - нажатие клавиши,
  - срабатывание таймера и т.д.



- Отличительным признаком сообщения является его код - 1 до Ox3FF (для системных сообщений).
- Каждому коду соответствует своя символическая константа, имя которой достаточно ясно говорит об источнике сообщения.

- события
  - *аппаратные*
  - *программные*
    - системы
    - прикладной программы

Пример:

Аппаратные события

- WM_MOUSEMOVE (код 0x200) - движение мыши
- WM_LBUTTONDOWN (код 0x201), нажатие на левую клавишу мыши
- WM_TIMER (код 0x113). срабатывание таймера

Программные события

по ходу создания и вывода на экран главного окна Windows последовательно посылает в приложение целую группу сообщений, сигнализирующих об этапах этого процесса:

- WM_GETMINMAXINFO для уточнения размеров окна,
- WM_ERASEBKGND при заполнении окна цветом фона,
- WM_SIZE при оценке размеров рабочей области окна,
- WM_PAINT для получения от программы информации о содержимом окна

Некоторые из этих сообщений Windows обрабатывает сама; другие обязана обработать прикладная программа.

- Может быть и обратная ситуация, когда сообщение создается в прикладной программе по воле программиста и посылается в Windows для того, чтобы система выполнила требуемые действия

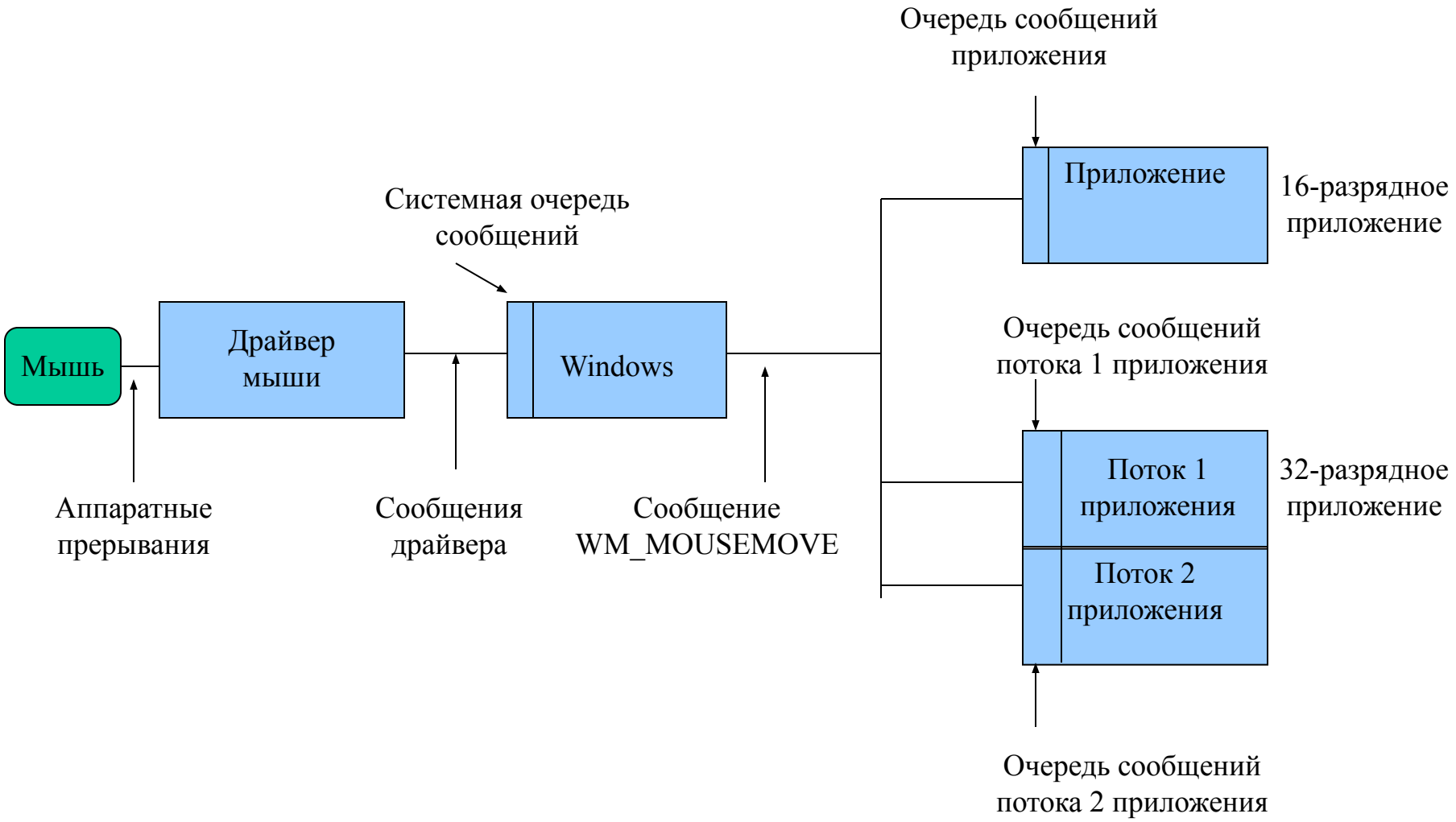
например

  - заполнила конкретной информацией окно со списком
  - сообщила о состоянии некоторого элемента управления.

- Сообщения такого рода тоже стандартизованы и имеют определенные номера, превышающие Ox3FF.

- Программист может предусмотреть собственные сообщения и направлять их в различные окна приложения для оповещения о тех или иных ситуациях.

Процедура создания и пересылки сообщения



Процедура пересылки и состав аппаратного сообщения

(на примере сообщения WM_MOUSEMOVE о движении мыши)

- Это сообщение возникает всякий раз, когда в результате движения мыши по столу зубец зубчатого колесика, связанного с катящимся по столу резиновым шариком, пересекает луч света от светодиода.
- Пересечение луча света в механизме мыши возбуждает сигнал аппаратного прерывания, который, поступив в компьютер,
- активизирует драйвер мыши, входящий в состав Windows.
- драйвер мыши формирует пакет данных и пересылает его в форме сообщения в системную очередь сообщений Windows.

- Дальнейшая судьба сообщения выглядит по-разному в 16- и 32-разрядных приложениях.

Win 16

- В единицей работы компьютера является выполняемое приложение, **называемое задачей.**
- Каждая задача имеет свою очередь сообщений ограниченного размера.
- Все сообщения, предназначенные данной задаче,
  - поступают в ее входную очередь,
  - извлекаются функцией GetMessage(), входящей в цикл обработки сообщений.

Win32

- В Win32 единицей работы компьютера **считается поток выполнения.**
- Каждое приложение создает по меньшей мере один, первичный поток, однако может создать и много потоков.
- Концепция потоков позволяет организовать в рамках одного приложения параллельное выполнение нескольких фрагментов программы.
- Для каждого потока в 32-разрядном приложении создается своя очередь сообщений. Эти очереди, в отличие от очередей Win 16, не имеют фиксированного размера, а могут неограниченно расширяться.
- Сообщения из системной очереди передаются не в приложение в целом, а распределяются по его потокам.
- Сообщения от мыши обычно (хотя не всегда) адресованы тому окну, над которым находится ее курсор.
  - щелкая по пункту меню или по кнопке в некотором окне, мы хотим вызвать действие именно для этого окна.
  - Окна создаются с помощью функции CreateWindow() тем или иным потоком приложения.
  - Сообщения от мыши направляются в очередь того потока, который создал данное окно.
- Рассмотрим теперь, из чего состоит каждое сообщение.

Состав сообщения

- В начале главной функции приложения WinMain объявлена структурная переменная Msg.
- Это важнейшая переменная, с помощью которой в программу передается содержимое сообщений Windows.
- Каждое сообщение представляет собой пакет из шести данных, описанных в файле WINUSER.H с помощью структуры типа MSG:

```
typedef struct tagMSG {
HWND        hwnd;     //Дескриптор окна, которому адресовано сообщение
UINT        message; //Код данного сообщения :
WPARAM      wParam;  //Дополнительная информация
LPARAM      lParam;   //Дополнительная информация
DWORD       time;     //Время отправления сообщения
POINT       pt;   //Поз. курсора мыши на момент отправления сообщения
} MSG;              //Новое имя для типа tagMSG
```

Структура Msg заполняется следующей информацией: (для сообщения WM_MOUSEMOVE)

Msg.hwnd - дескриптор окна под курсором мыши;

Msg.message - код сообщения WM_MOUSEMOVE=Ox200;

Msg.wParam - комбинация битовых флагов, индицирующих состояние клавиш мыши (нажаты/не нажаты), а также клавиш Ctrl и Shift;

Msg.lParam - позиция курсора мыши относительно рабочей области окна;

Msg.time - время отправления сообщения;

Msg.pt - позиция курсора мыши относительно границ экрана.

Вызывая оконную функцию, функция DispatchMessage()

- передает ей первые 4 параметра;
- если программе для организации правильной реакции на пришедшее сообщение требуются оставшиеся два параметра, их можно извлечь непосредственно из переменной MSG.

Манипуляции с мышью могут порождать и другие сообщения

- нажатие левой клавиши возбуждает сообщение WM_LBUTTONDOWN (код 0x201),
- отпускание левой клавиши - сообщение WM_LBUTTONUP (код 0x202),
- нажатие правой клавиши - сообщение WM_RBUTTONDOWN (код 0x204).
- двойной щелчок левой клавиши порождает целых 4 сообщения:
  - WM_LBUTTONDOWN,
  - WM_LBUTTONUP,
  - WM_LBUTTONDBLCLK (код 0x203) и снова
  - WM_LBUTTONUP.
- Программист может обрабатывать
  - все эти сообщения,
  - только сообщения о двойном нажатии, не обращая внимания на остальные.
- Механизм образования всех этих сообщений в точности такой же, как и для сообщения WM_MOUSEMOVE
  1. аппаратное прерывание,
  2. формирование драйвером мыши пакета данных,
  3. установка сообщения в системную очередь,
  4. пересылка сообщения в очередь приложения,
  5. вызов оконной функции.
- Даже пакеты данных для этих сообщений не различаются.

Сообщения от клавиатуры

- WM_KEYDOWN о нажатии любой "несистемной" клавиши (т. е. любой клавиши, не сопровождаемой нажатием клавиши Alt),
- WM_KEYUP об отпускании несистемной клавиши,
- WM_SYSKEYDOWN о нажатии "системной" клавиши (т. е. любой клавиши совместно с клавишей Alt) и др.

сообщениями нижнего уровня

- Рассмотренные сообщения относятся к **сообщениями нижнего уровня** - они оповещают об аппаратных событиях практически без всякой их обработки Windows.
- Некоторые аппаратные события предварительно обрабатываются Windows, и в приложение поступает уже результат этой обработки.

Сообщения верхнего уровня

1. При нажатии левой клавиши мыши над строкой меню
   - аппаратное прерывание поглощается системой Windows
   - вместо сообщения WM_LBUTTONDOWN формируется сообщение WM_COMMAND,
     - в число параметров WM_COMMAND входит идентификатор того пункта меню, над которым был курсор мыши.

   Это избавляет нас от необходимости анализа положения курсора мыши и выделения всех положений, соответствующих прямоугольной области данного пункта меню.

2. Сообщение WM_NCLBUTTONDOWN, формируется Windows, если пользователь нажал левую клавишу мыши в нерабочей области окна, т. е. на его заголовке (с целью, например, "перетащить" окно приложения на другое место экрана).

Рассмотренный механизм прохождения сообщений справедлив главным образом для аппаратных сообщений.

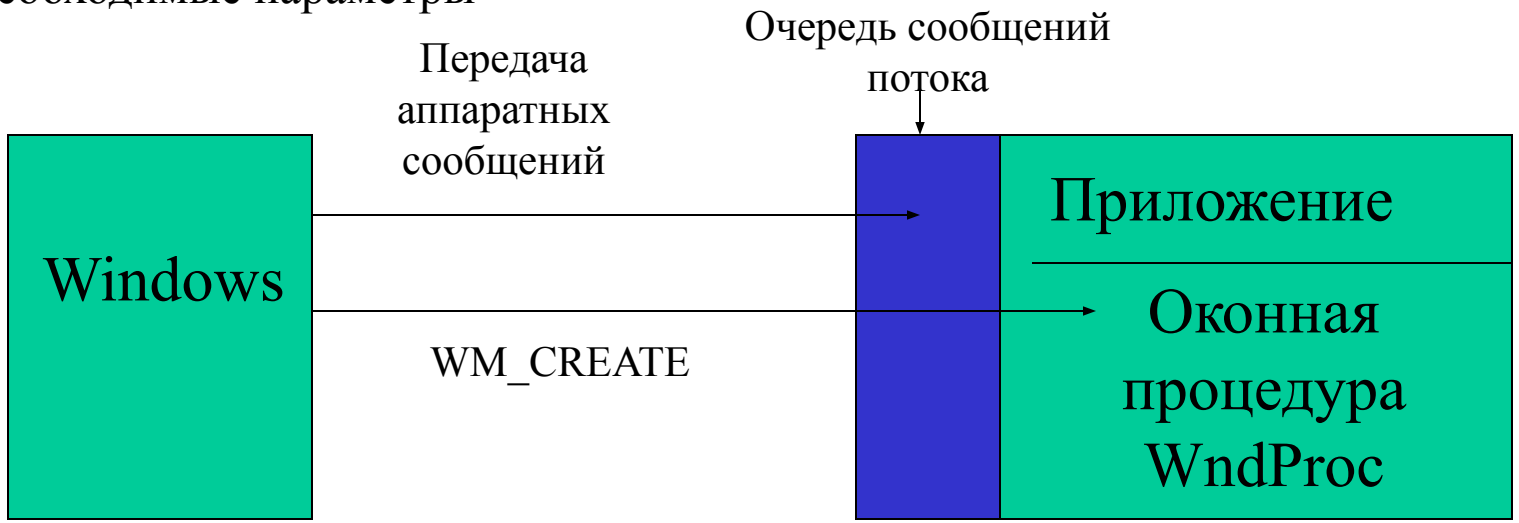
Программные сообщения

- Большая **часть программных сообщений,** т. е. сообщений, прямо не связанных с аппаратными событиями, а возникающими по ходу протекания программных процессов в приложениях или в самой Windows, обслуживаются системой иным образом.

- Рассмотрим сообщение WM_CREATE.

Сообщение WM_CREATE

- Сообщение WM_CREATE генерируется системой Windows в процессе создания окна, чтобы программист, перехватив это сообщение, мог выполнить необходимые инициализирующие действия:
  - установить системный таймер,
  - загрузить требуемые ресурсы (шрифты, растровые изображения),
  - открыть файлы с данными и т. д.
- Сообщение WM_CREATE не поступает в очередь сообщений приложения и, соответственно, не изымается оттуда функцией GetMessage().
- Windows непосредственно вызывает оконную функцию WndProc и передает ей необходимые параметры



Обработка сообщения WM_CREATE

С точки зрения программиста обычно не имеет особого значения, каким образом вызывается оконная функция –

- функцией DispatchMessage() или
- непосредственно программами Windows.

Однако

- при обработке сообщения WM_MOUSEMOVE все содержимое этого сообщения находится в структурной переменной Msg,
- при обработке WM_CREATE мы имеем дело только с параметрами, переданными Windows в оконную функцию.

  В переменной Msg в это время находится старое, уже обработанное сообщение, т. е."мусор".

В обход очереди сообщений приложения и структурной переменной Msg, обрабатываются сообщения:

- WM_INITDIALOG - инициализация диалога,
- WM_SYSCOMMAND - выбор пунктов системного меню,
- WM_DESTROY - уничтожение окна
- многие другие.

Обработка сообщений

Функция GetMessage() анализирует очередь сообщений приложения.

- Если в очереди обнаруживается сообщение, то GetMessage()
  1. извлекает сообщение из очереди
  2. передает в структуру Msg,
  3. после чего завершается с возвратом значения TRUE.

При отсутствии сообщений в очереди функция GetMessage() ведет себя по-разному в 16- и 32-разрядных приложениях.

Поведение GetMessage() в 16-разрядных приложениях при отсутствии сообщений в очереди

GetMessage() в 16-разрядных приложениях при отсутствии сообщений в очереди вызывает программный блок Windows,

- который передает управление циклам обработки сообщений других работающих приложений (других задач).
- После опроса остальных приложений управление возвращается в наше приложение в ту же точку анализа очереди сообщений.

Такой способ организации параллельного выполнения нескольких приложений носит название коллективной или невытесняющей многозадачности.

Характерной чертой этого механизма является неопределенность моментов передачи управления от задачи к задаче.

– Действительно, смены задачи не произойдет, пока приложение не закончит обработку текущего сообщения. Более того, если одно из приложений, начав обрабатывать какое-либо сообщение, войдет в бесконечный цикл, то ни другие приложения, ни сама система Windows никогда не получат управление - произойдет "зависание" системы.

Поведение GetMessage() в 32-разрядных приложениях при отсутствии сообщений в очереди

- Здесь единицей работы компьютера считается поток.
- Система распределяет процессорное время между потоками на регулярной основе, предоставляя каждому потоку по очереди определенный квант времени порядка 20 мс.
- Зацикливание одного из потоков нарушит нормальное выполнение конкретно этого потока, однако не отразится на работоспособности остальных потоков и всей системы.
- Такая организация вычислительного процесса получила название **вытесняющей многозадачности.**

- В действительности очередность переключения задач (или, точнее, потоков) оказывается более сложной, так как передавая управление от потока к потоку, система учитывает их приоритеты.

Очередность переключения задач с учетом их приоритетов

- системному потоку, отвечающему за ввод с клавиатуры или от мыши, система назначает более высокий приоритет, чтобы обеспечить быструю реакцию на ввод пользователем новых данных.
- Более высоким приоритетом также обладают потоки приложения переднего плана (т. е. приложения, окно которого расположено на Рабочем столе поверх всех остальных).
- При этом система динамически изменяет в некоторых пределах приоритеты выполняющихся потоков по определенному алгоритму, чтобы обеспечить их более эффективное выполнение.

Вытесняющая многозадачность

- При наличии вытесняющей многозадачности нет необходимости ожидать, пока приложение, захватившее время процессора, закончит обрабатывать очередное сообщение.
- Каждое выполняемое приложение периодически получает квант процессорного времени. Что именно делает это приложение, - опрашивает ли свою очередь сообщений или обрабатывает поступившее сообщение, - не имеет значения. Блок Windows, отвечающий за распределение процессорного времени, может в любой момент времени прервать активную задачу и передать управление следующей.
- Таким образом, в 32-разрядных приложениях функция GetMessage(), не обнаружив сообщений в очереди "своего" потока, может не передавать управление системе, а продолжить опрос очереди до истечения кванта времени или до обнаружения в очереди сообщения.
- Однако это повлекло бы нерациональную трату процессорного времени.

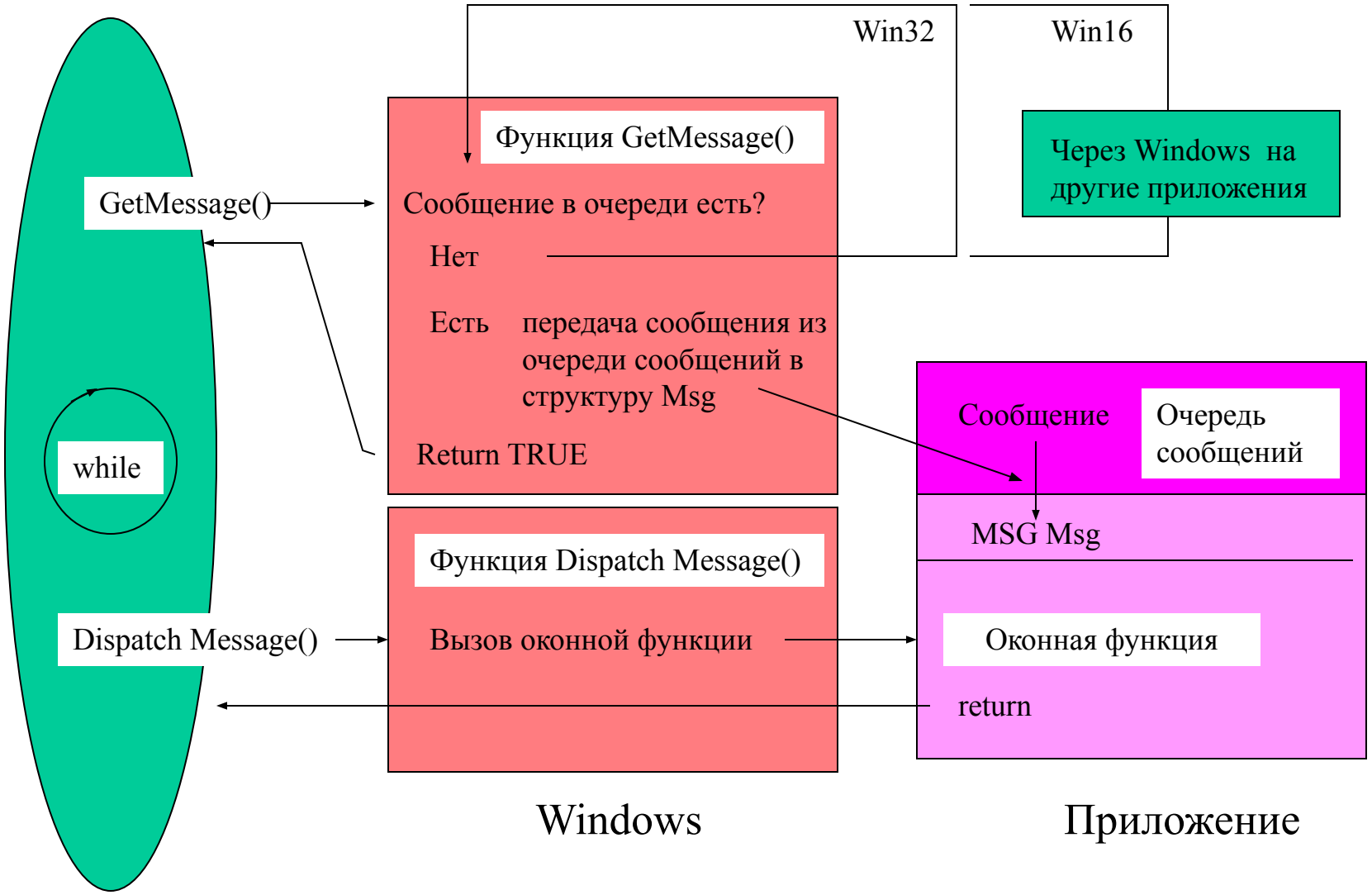
Спящий поток

- В действительности, если при выполнении функции GetMessage() оказывается, что очередь сообщений пуста, система останавливает выполнение данного потока, переводя его в "спящее" состояние.
- "Спящий" поток не потребляет процессорного времени и не тормозит работу системы.
- Поток возобновит свою работу, как только в очереди появится сообщение. Это событие снова вызывает к жизни функцию GetMessage(), которая выполняет предназначенную ей работу - перенос сообщения из очереди сообщений в структурную переменную Msg.

Завершение обработки сообщения

- В любом случае функция GetMessage() завершится (с возвратом значения TRUE) лишь после того, как очередное сообщение попадет в переменную Msg.
- Далее в цикле while вызывается функция DispatchMessage(). Ее назначение - вызов оконной функции для того окна, которому предназначено очередное сообщение. После того как оконная функция обработает сообщение, возврат из нее приводит к возврату из функции DispatchMessage() на продолжение цикла while.

Цикл обработки сообщений



Функция GetMessage()

- Функция GetMessage() требует 4 параметра.
  - адрес структуры Msg, в которую GetMessage() должна передать изъятое из очереди сообщение.
  - HWND позволяет определить окно, чьи сообщения будут извлекаться функцией GetMessage().
    - Если этот параметр равен NULL, GetMessage() работает со всеми сообщениями данного приложения.
  - Два последних параметра определяют диапазон сообщений, которые анализируются функцией GetMessage().

например

- при параметрах WM_KEYFIRST и WM_KEYLAST, GetMessage будет забирать из очереди только сообщения, относящиеся к клавиатуре
- константы WM_MOUSEFIRST и WM_MOUSELAST позволят работать только с сообщениями от мыши.

Чаще всего надо анализировать все сообщения. Чтобы исключить фильтрацию сообщений, оба параметра должны быть равны нулю.

Сообщение WM_QUIT

- Особая ситуация возникает, если функция GetMessage() обнаруживает в очереди сообщение WM_QUIT с кодом 0x12.
- В этом случае GetMessage() сразу же завершается с возвратом значения FALSE.
- Однако цикл while выполняется, лишь если GetMessage() возвращает TRUE.
- Возврат FALSE приводит к завершению цикла и переходу на предложение return 0; т. е. к завершению функции WinMain() и всего приложения.
- Таким образом, условием завершения приложения является появление сообщения WM_QUIT.