Лекция 22

30 апреля

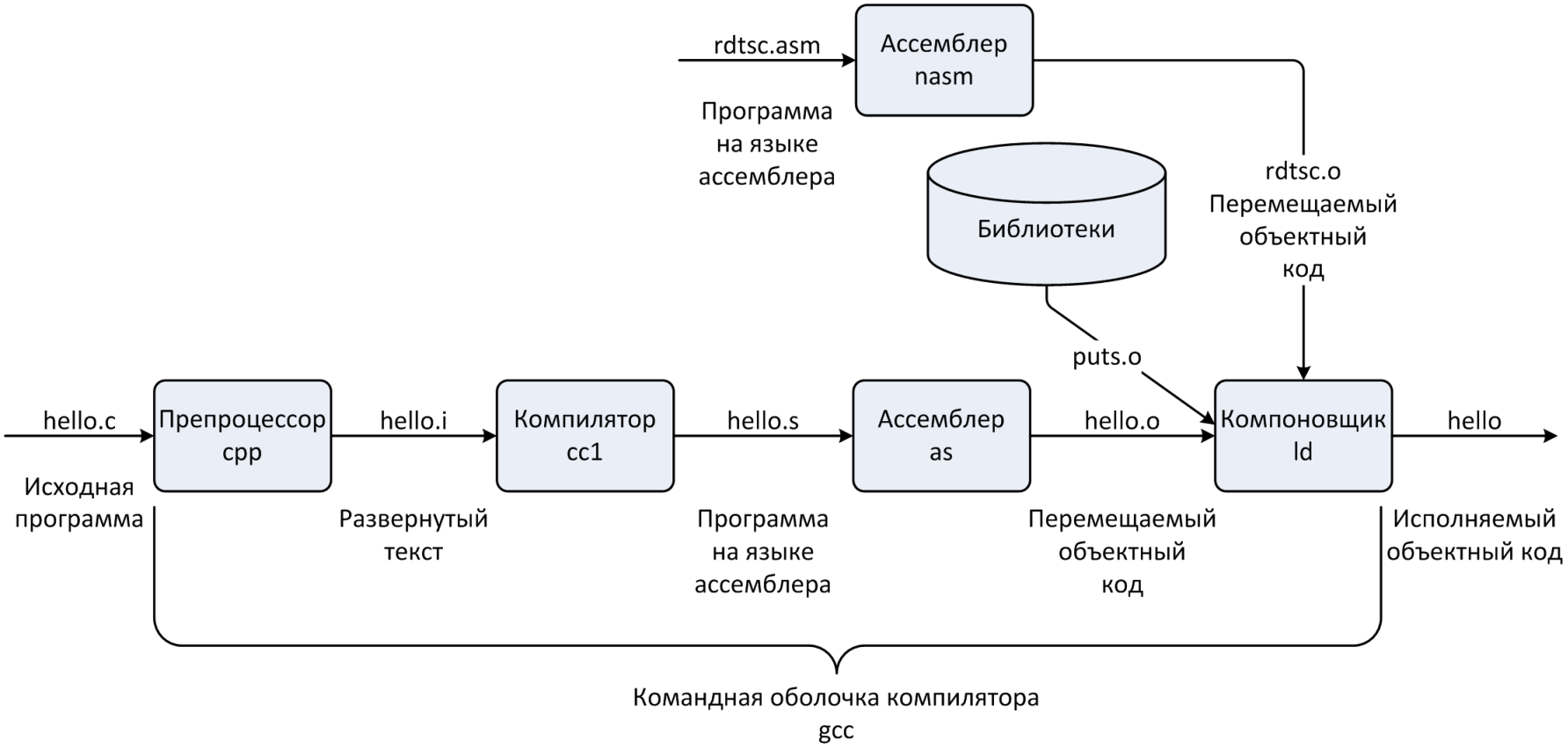
Язык программирования

- Цели, задачи, реализация языка
- Семантика
  - Операционная
- Компилируемые и интерпретируемые языки
  - Язык Java
    исходный код → байт-код → машинные команды
- Процессор – интерпретатор машинных команд
  - Эффективность написания программы в машинных командах крайне низкая
    - Кодирование команд
    - Организация программы
    - Организация вычислений

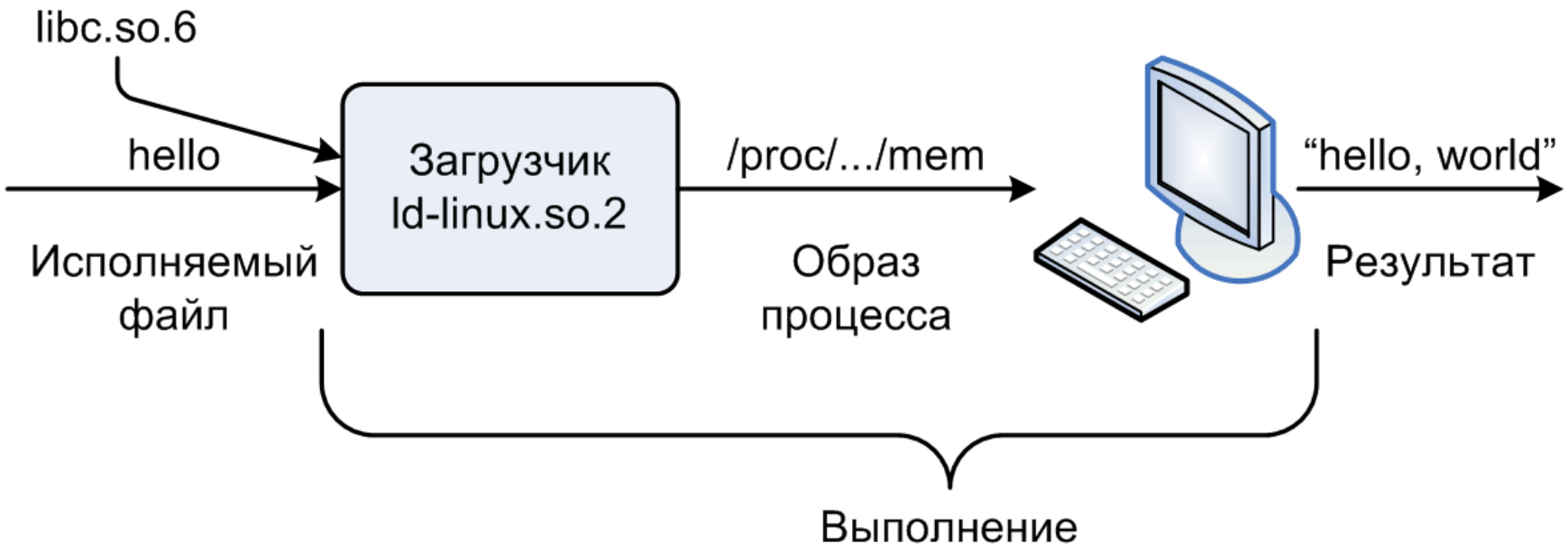
Система программирования

- Системные/прикладные программы
  - Операционная система
  - Программные средства разработки
- Система программирования – комплекс средств
  - Язык программирования
    - Информационные ресурсы
    - Программные инструменты
    - Библиотеки
- Этапы жизненного цикла программы
  - Проектирование
    - Сбор и анализ требований к программе
    - Разработка
  - Реализация
    - Кодирование
    - Отладка
  - Сопровождение

Система программирования языка Си



Выполнение программы



Разработка Си-программы

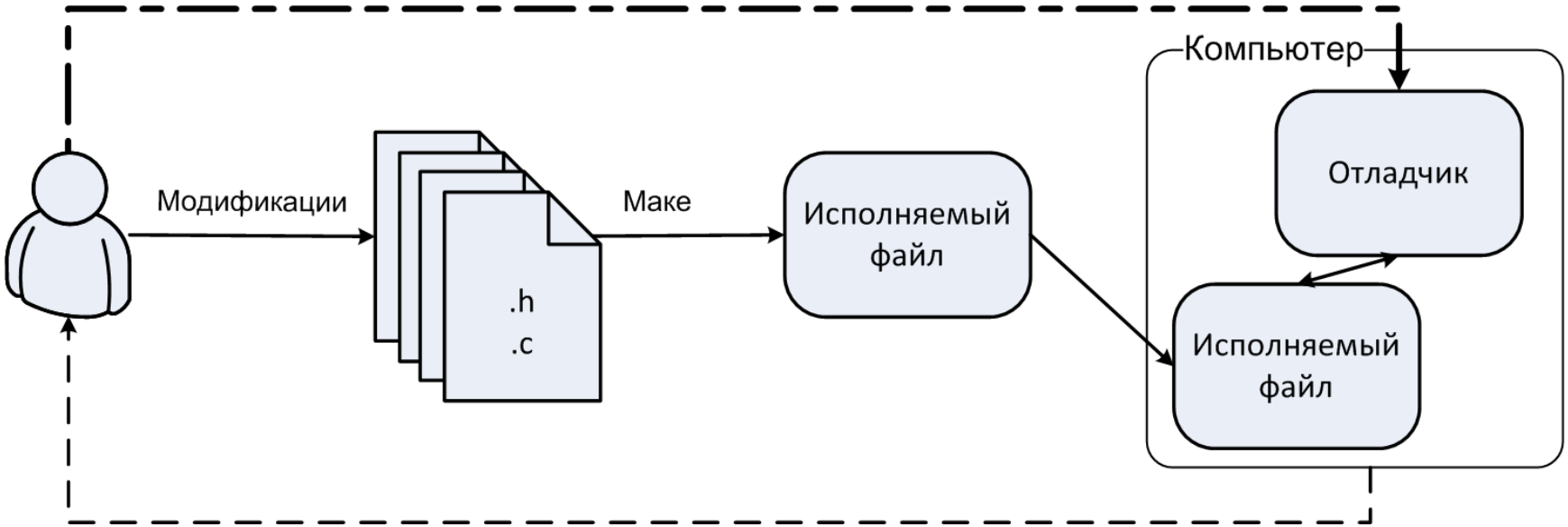


Схема работы ассемблера

- Проблема: опережающие ссылки
- Первый проход: составление таблицы символов
  - Символ
    - Метка
    - Значение, которому приписано имя
  - Таблица символов
    - Длина поля
    - Глобальный/локальный
    - Перераспределение
- Второй проход: построение объектного кода

Пример Си-программы

main.c

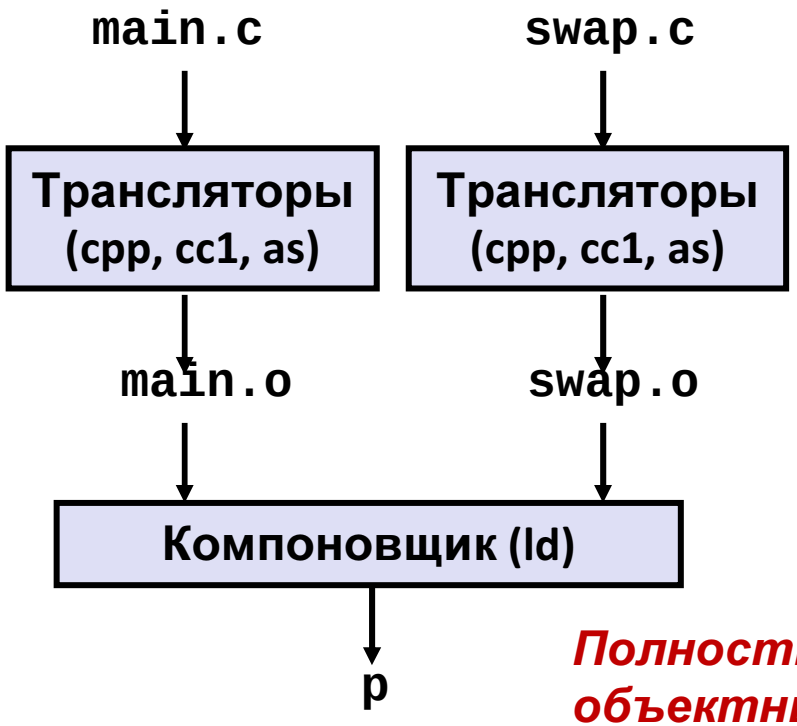
```
int buf[2] = {1, 2};

int main()
{
  swap();
  return 0;
}
```

swap.c

```
extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;

void swap()
{
  int temp;

  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

Статическая компоновка

- Программа транслируется и компонуется драйвером компилятора:
  - `unix> gcc -O2 -g -o p main.c swap.c`
  - `unix> ./p`



Файлы с исходным кодом

Раздельно откомпилированные файлы с перемещаемым объектным кодом

Полностью скомпонованный исполняемый объектный файл (содержит код и данные всех функций определенные в `main.c` и `swap.c`)

Почему нужен компоновщик?

- Причина 1: Модульность программы
  - Программа может быть организована как набор небольших файлов с исходным кодом, а не один монолитный файл.
  - Есть возможность организовывать библиотеки функций, являющихся общими для разных программ
    - например, библиотека математических функций, стандартная библиотека языка Си

Почему нужен компоновщик?

- Причина 2: Эффективность
  - Время: Раздельная компиляция
    - Меняем код в одном файле, компилируем только его, повторяем компоновку
    - Нет необходимости повторять компиляцию остальных файлов с исходным кодом.
  - Место на диске: Библиотеки
    - Общие функции можно объединить в одном файле...
    - Исполняемые файлы и образ программы в памяти содержит только те функции, которые действительно используются.

Что делает компоновщик?

- Шаг 1. Разрешение символов
  - В программе определяют и используют *символы* (переменные и функции):
    - `void swap() {…}  /* определение символа swap */`
    - `swap();  /* ссылка на символ */`
    - `int *xp = &x;  /* определение символа xp, ссылка на x */`
  - Определения символов сохраняются в таблице символов.
    - Таблица символов – массив структур
    - Каждая запись содержит имя, размер, позицию символа.
  - Компоновщик устанавливает связь каждой ссылки на символ с единственным определением символа.

Что делает компоновщик?

- Шаг 2. Перемещение
  - Несколько объявлений секций кода и данных объединяются в единые секции
  - Символы перемещаются с их относительных позиций в .o-файлах на абсолютные адреса в исполняемом файле.
  - Обновляются все ссылки на символы, согласно их новым позициям.

Три типа объектных файлов (модулей)

- Перемещаемые объектные файлы (**.** o-файлы)
  - Содержит код и данные в форме, позволяющей проводить компоновку с другими перемещаемым объектными файлами.
    - Каждый **.** o-файл производится из **одного** файла с исходным кодом (**.** c-файла)
- Исполняемые объектные файлы (a.out-файлы)
  - Содержит код и данные в такой форме, что их можно напрямую копировать в память и запускать выполнение программы.

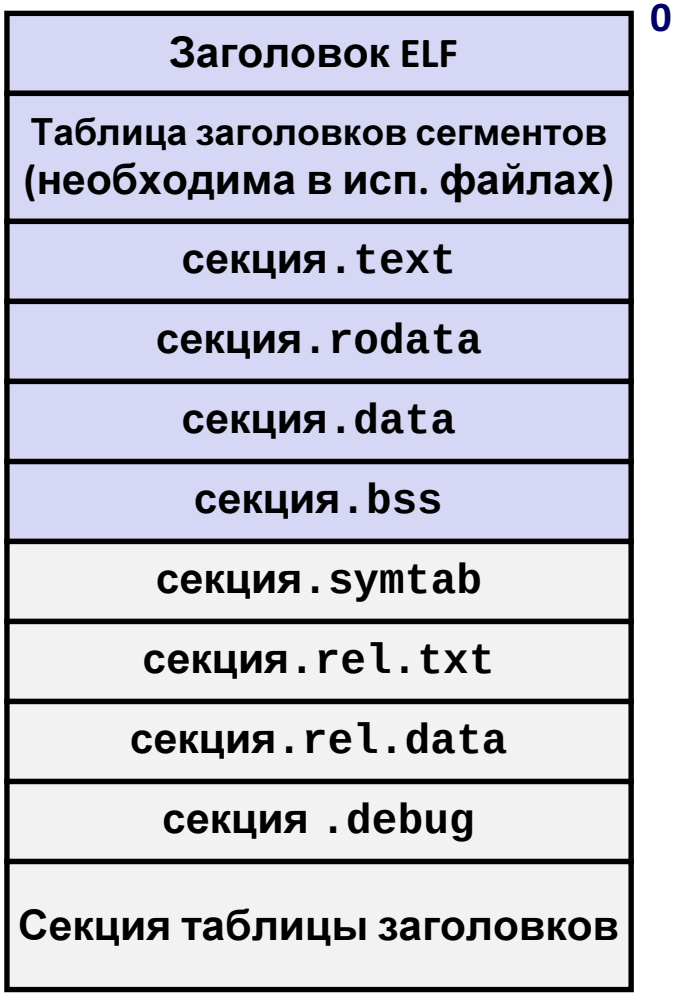
– Разделяемые объектные файлы (.so-файлы)
  - Особый вид перемещаемого объектного файла, который может быть загружен в память и скомпонован с программой динамически, во время ее загрузки и во время работы.
  - Windows - Dynamic Link Libraries (DLL)

Executable and Linkable Format (ELF)

- Стандартный бинарный формат объектных файлов
- Был предложен в AT&T System V Unix
  - Позже был поддержан в BSD и Linux
- Единый формат для
  - Перемещаемых объектных файлов (`.o`),
  - Исполняемых объектных файлов (`a.out`)
  - Разделяемых объектных файлов (`.so`)

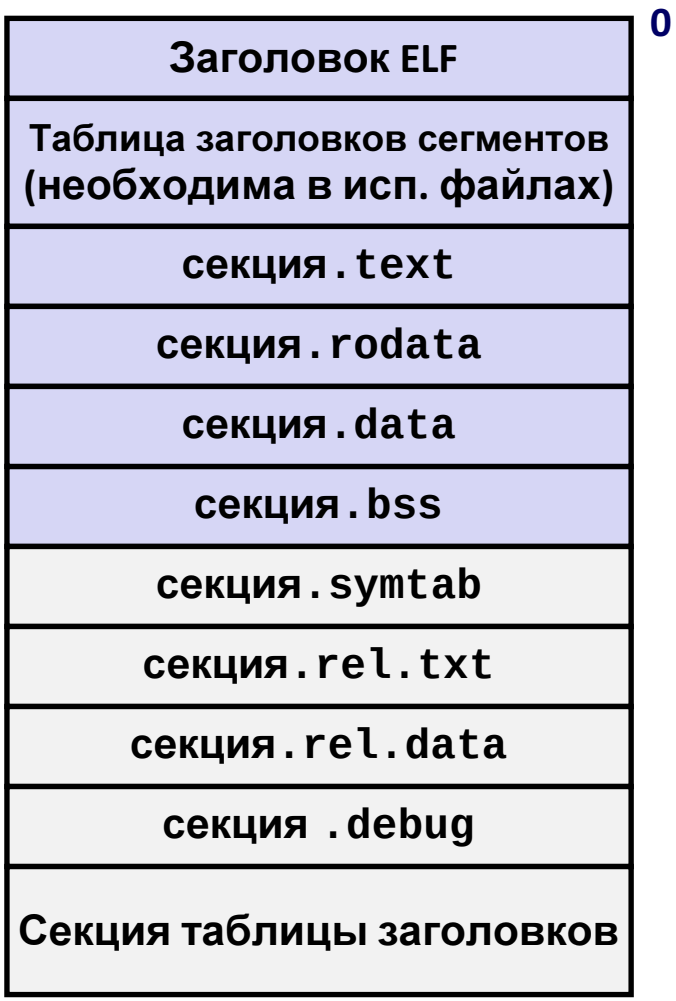
Формат ELF файла

- Заголовок Elf
  - Размер машинного слова, порядок байт, тип файла (.o, исп., .so), и др.
- Таблица заголовков сегментов
  - Размер страницы, сегменты виртуальной памяти, размеры сегментов.
- Секция `.text`
  - код
- Секция `.rodata`
  - Данные, доступные только на чтение: таблицы переходов, константы
- Секция `.data`
  - Инициализированные глобальные переменные
- Секция `.bss`
  - Неинициализированные глобальные переменные
  - У секции есть заголовок, на сама секция не занимает места



Формат ELF файла (продолжение)

- Секция `.symtab`
  - Таблица символов
  - Имена функций и статических переменных
  - Имена секций
- Секция `.rel.text`
  - Данные для перемещения секции **`.text`**
  - Адреса инструкций которые должны быть обновлены
- Секция `.rel.data`
  - Данные для перемещения секции **`.data`**
  - Адреса глобальных переменных, инициализированных ссылками на внешние функции или глобальные переменные
- Секция `.debug`
  - Данные для символьного отладчика (**`gcc -g`**)
- Секция таблицы заголовков
  - Смещения и размеры каждой секции



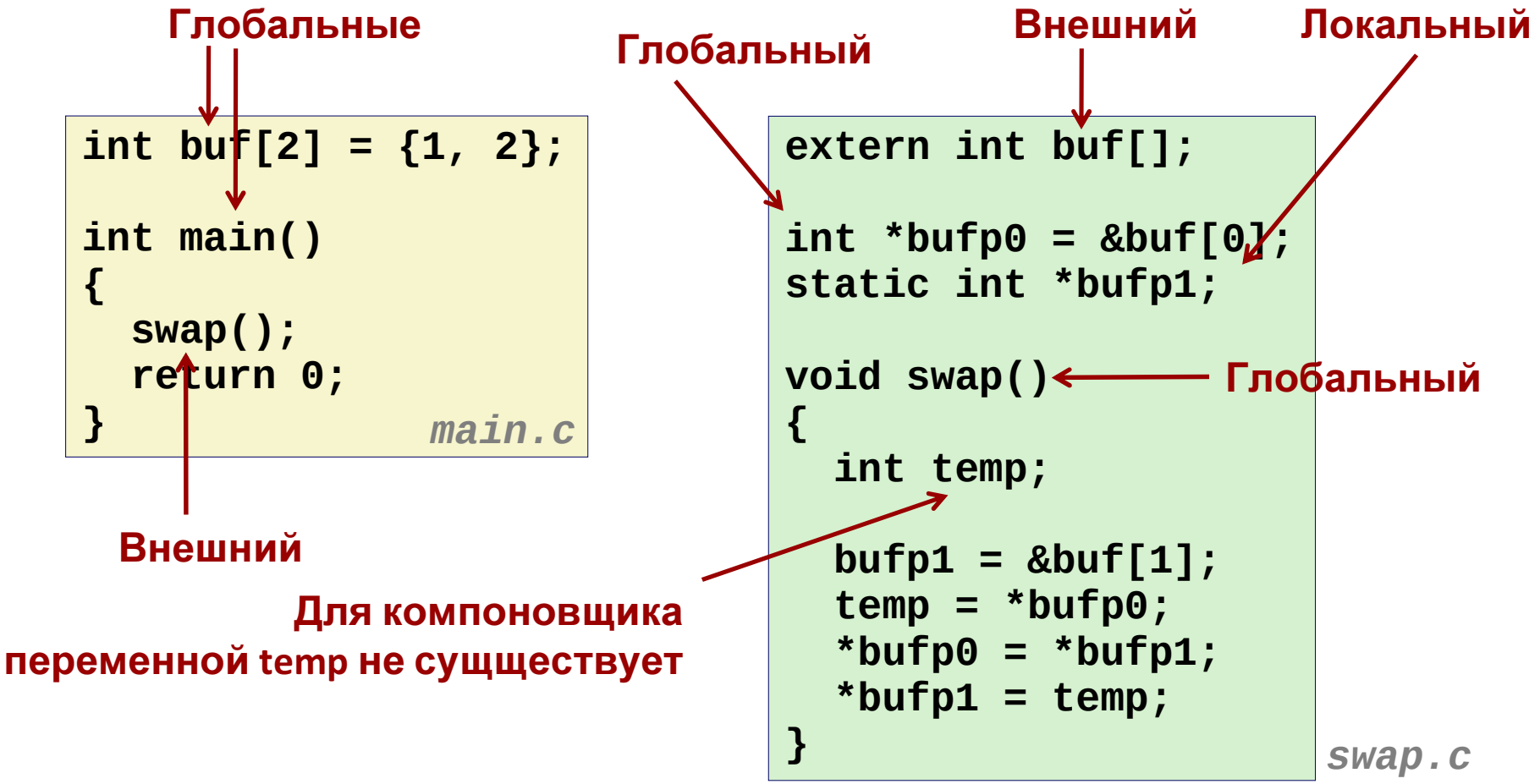
Символы в процессе компоновки

- Глобальные символы
  - Символы определенные в одном модуле таким образом, что их можно использовать в других модулях.
  - Например: не-**`static`** Си-функции и не-**`static`** глобальные переменные.

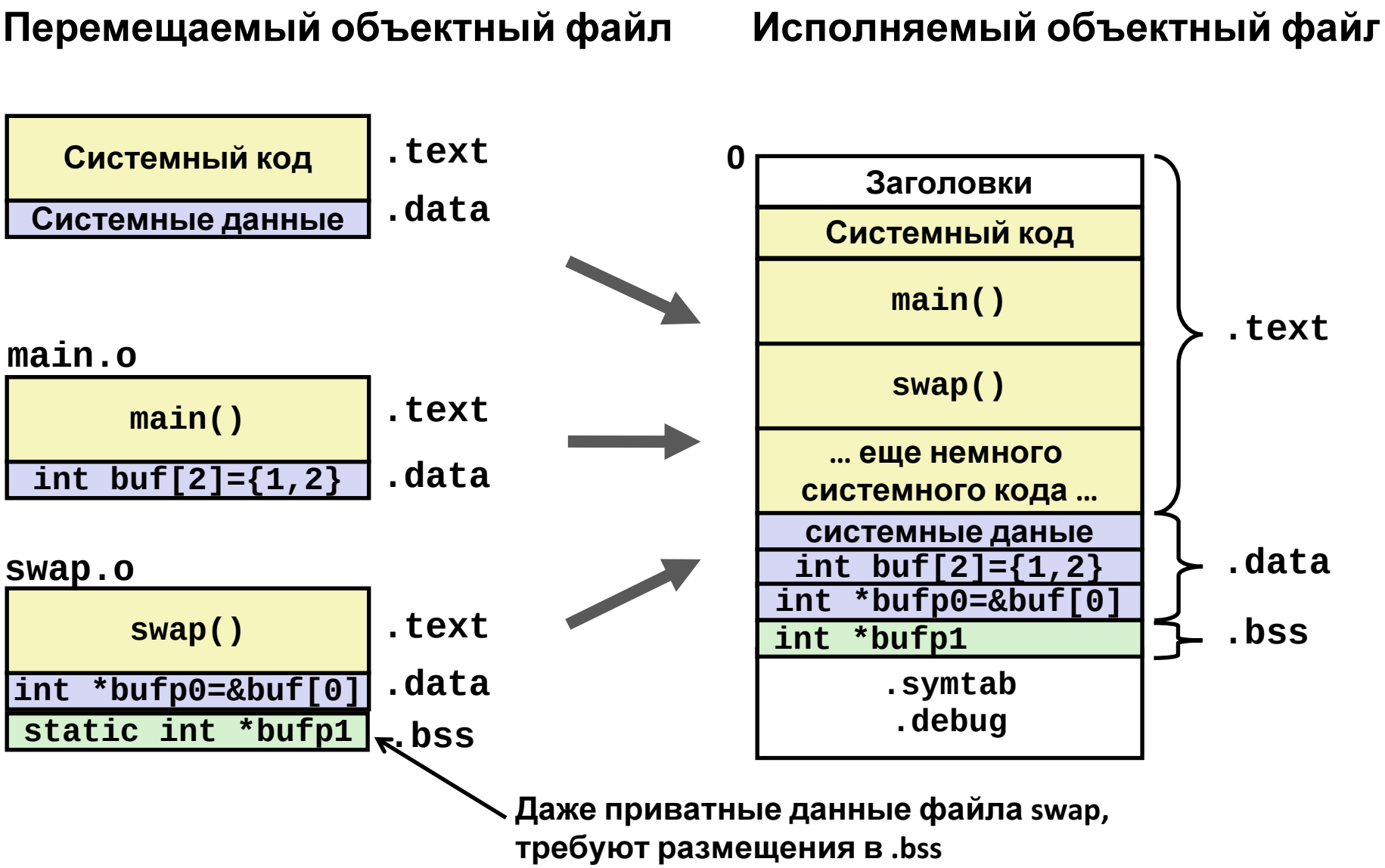
- Внешние символы
  - Глобальные символы, которые используются в модуле, но определены в каком-то другом модуле.

- Локальные символы
  - Символы определены и используются исключительно в одном модуле.
  - Например: Си-функции и переменные, определенные с модификатором static.
  - **Локальные символы *не являются* локальными переменными Си-программы**

Разрешение символов



Перемещение кода и данных



Данные для перемещения (main)

main.c

```
int buf[2] =
  {1,2};

int main()
{
  swap();
  return 0;
}
```

main.o

```
0000000 <main>:
   0:   8d 4c 24 04      lea    ecx, [0x4 + esp]
   4:   83 e4 f0         and    esp, 0xfffffff0
   7:   ff 71 fc         push   dword [0xfffffffc + ecx]
   a:   55               push   ebp
   b:   89 e5            mov    ebp, esp
   d:   51               push   ecx
   e:   83 ec 04         sub    esp, 0x4
  11:   e8 fc ff ff ff   call   12 <main+0x12>
                        12: R_386_PC32 swap
  16:   83 c4 04         add    esp, 0x4
  19:   31 c0            xor    eax, eax
  1b:   59               pop    ecx
  1c:   5d               pop    ebp
  1d:   8d 61 fc         lea    esp, [0xfffffffc + ecx]
  20:   c3               ret
```

```
objdump –r –d –M intel
```

```
Дизассемблирование секции .data:

 00000000 <buf>:
   0:   01 00 00 00 02 00 00 00
```

Данные для перемещения (swap, .data)

swap.c

```
extern int buf[];

int *bufp0 =
            &buf[0];
static int *bufp1;

void swap()
{
  int temp;

  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

```
Дизассемблированная секция .data:

00000000 <bufp0>:
   0:   00 00 00 00

        0: R_386_32 buf
```

Данные для перемещения (swap, .text)

swap.c

```
extern int buf[];

int
  *bufp0 = &buf[0];

static int *bufp1;

void swap()
{
  int temp;

  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

swap.o

Дизассемблированная секция .text:

```
00000000 <swap>:
   0:   8b 15 00 00 00 00       mov    edx, 0x0
                        2: R_386_32    buf
   6:   a1 04 00 00 00          mov    eax, 0x4
                        7: R_386_32    buf
   b:   55                      push   ebp
   c:   89 e5                   mov    ebp, esp
   e:   c7 05 00 00 00 00 04    mov    [0x0], 0x4
  15:   00 00 00
                        10: R_386_32   .bss
                        14: R_386_32   buf
  18:   8b 08                   mov    ecx, [eax]
  1a:   89 10                   mov    [eax], edx
  1c:   5d                      pop    ebp
  1d:   89 0d 04 00 00 00       mov    0x4, ecx
                        1f: R_386_32   buf
  23:   c3                      ret
```

Исполняемый файл до/после перемещения (.text)

```
0000000 <main>:
   . . .
   e:   83 ec 04          sub    esp, 0x4
  11:   e8 fc ff ff ff    call   12 <main+0x12>
                        12: R_386_PC32 swap
  16:   83 c4 04          add    esp, 0x4
  . . .
```

0x8048396 + 0x1a
= 0x80483b0

```
08048380 <main>:
 8048380:      8d 4c 24 04             lea    ecx, [0x4+esp]
 8048384:      83 e4 f0                and    esp, 0xfffffff0
 8048387:      ff 71 fc                push   dword [0xfffffffc+ecx]
 804838a:      55                      push   ebp
 804838b:      89 e5                   mov    ebp, esp
 804838d:      51                      push   ecx
 804838e:      83 ec 04                sub    esp, 0x4
 8048391:      e8 1a 00 00 00          call   80483b0 <swap>
 8048396:      83 c4 04                add    esp, 0x4
 8048399:      31 c0                   xor    eax, eax
 804839b:      59                      pop    ecx
 804839c:      5d                      pop    ebp
 804839d:      8d 61 fc                lea    esp, [0xfffffffc+ecx]
 80483a0:      c3                      ret
```

```
   0:   8b 15 00 00 00 00       mov    edx, 0x0
                        2: R_386_32    buf
   6:   a1 04 00 00 00          mov    eax,0x4
                        7: R_386_32    buf
   ...
   e:   c7 05 00 00 00 00 04    movl   0x0, 0x4
  15:   00 00 00
                        10: R_386_32   .bss
                        14: R_386_32   buf
   . . .
  1d:   89 0d 04 00 00 00       mov    0x4, ecx
                        1f: R_386_32   buf
  23:   c3                      ret
```

```
080483b0 <swap>:
 80483b0:       8b 15 20 96 04 08      mov    edx, 0x8049620
 80483b6:       a1 24 96 04 08         mov    eax, 0x8049624
 80483bb:       55                     push   ebp
 80483bc:       89 e5                  mov    ebp, esp
 80483be:       c7 05 30 96 04 08 24   mov    0x8049630, 0x8049624
 80483c5:       96 04 08
 80483c8:       8b 08                  mov    ecx, dword [eax]
 80483ca:       89 10                  mov    dword [eax], edx
 80483cc:       5d                     pop    ebp
 80483cd:       89 0d 24 96 04 08      mov    0x8049624, ecx
 80483d3:       c3                     ret
```

Исполняемый файл после перемещения (.data)

```
Дизассемблированная секция .data:

08049620 <buf>:
 8049620:       01 00 00 00 02 00 00 00

08049628 <bufp0>:
 8049628:       20 96 04 08
```