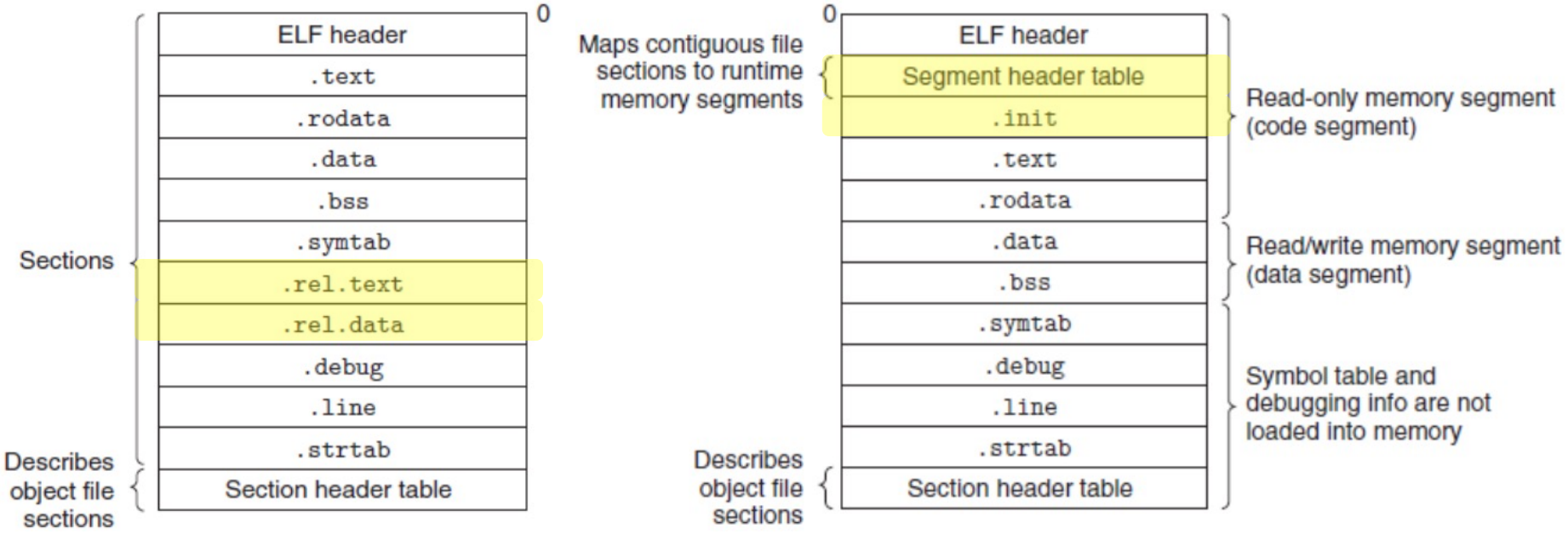
# CSE320 System Fundamentals II Linking 2

YOUNGMIN KWON

# Executable Object Files




ELF relocatable object file

ELF executable object file

# ELF Executable Object Files

.init section

- Defines a function _init that will be called by the program's initialization code

No .rel.text and .rel.data sections

- The file is already relocated

Program header table

- Makes it easy to load the file into memory

```
    Read-only code segment
1     LOAD off    0x00000000 vaddr 0x08048000 paddr 0x08048000 align 2**12
2          filesz 0x00000448 memsz 0x00000448 flags r-x

    Read/write data segment
3     LOAD off    0x00000448 vaddr 0x08049448 paddr 0x08049448 align 2**12
4          filesz 0x000000e8 memsz 0x00000104 flags rw-
```

# Dynamic Linking with Shared Libraries

Shared libraries

- An object module that can be loaded at an arbitrary memory address and linked with a program at run-time or load-time
- This process is called dynamic linking

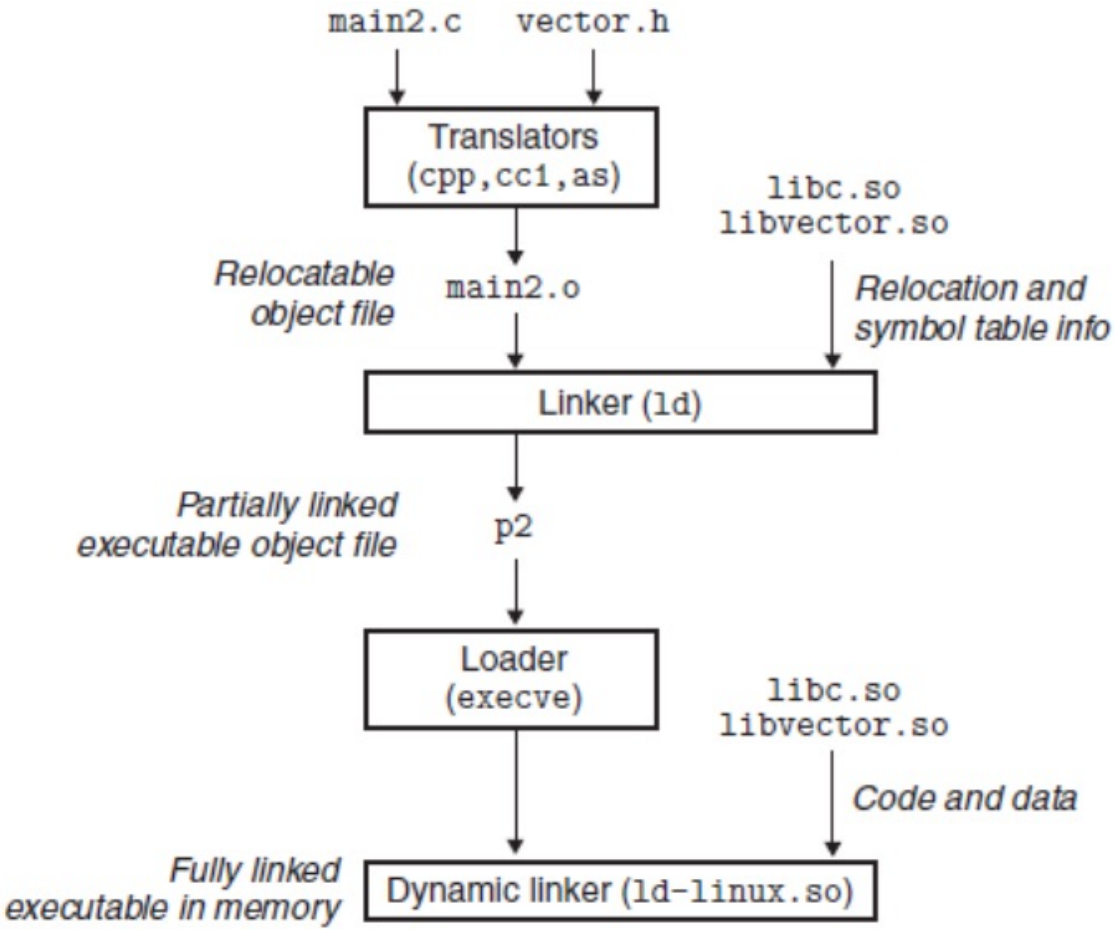
To build a shared library

- `gcc -shared -fpic -o libvector.so addvec.c mulvec.c`

To link shared objects at load-time

- `gcc main2.c ./libvector.so`

# Dynamic Linking with Shared Libraries



# Loading and Linking from Applications

Dynamic linking at run-time

- Applications can request the dynamic linker to load and link shared libraries at run-time

Real world usages

- Distributing software updates
- High-performance web servers
  - Instead of creating a child process to run CGI programs, load, link, and run the appropriate functions directly

# Loading and Linking from Applications

Related functions:

```
#include <dlfcn.h>

// Loads and links the shared library filename
// flag: RTLD_GLOBAL, RTLD_NOW, RTLD_LAZY, ...
// Returns a pointer to handle or NULL on error
void *dlopen(const char *filename, int flag);

// Returns the address of the symbol or NULL
void *dlsym(void *handle, char *symbol);

// Unloads the shared library
int dlclose(void *handle);

// Returns an error message if previous call to
// dlopen, dlsym, or dlclose failed
const char *dlerror(void);
```

# Loading and Linking from Applications

```c
/* main3.c */

#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#define CHECKNULL_EXIT(p)                       \
    if((p) == NULL) {                           \
        fprintf(stderr,                         \
            "%s in file %s @ line %d\n",        \
            dlerror(), __FILE__, __LINE__); \
        exit(1);                                \
    }

void (*_addvec)(int *x, int *y, int *z, int n);
void (*_mulvec)(int *x, int *y, int *z, int n);
int *_addcnt;
int *_mulcnt;

int x[2] = {1, 2};
int y[2] = {3, 4};
int z[2];
```

# Loading and Linking from Applications

```
int main() {
    void *handle;
    CHECKNULL_EXIT(handle = dlopen("./libvector.so", RTLD_LAZY|RTLD_GLOBAL));
    CHECKNULL_EXIT(_addvec = dlsym(handle, "addvec"));
    CHECKNULL_EXIT(_addcnt = dlsym(handle, "addcnt"));
    CHECKNULL_EXIT(_mulvec = dlsym(handle, "mulvec"));
    CHECKNULL_EXIT(_mulcnt = dlsym(handle, "mulcnt"));

    _addvec(x, y, z, 2);
    printf("z = [%d, %d]\n", z[0], z[1]);
    printf("addcnt = %d\n", *_addcnt);

    _mulvec(x, y, z, 2);
    printf("z = [%d, %d]\n", z[0], z[1]);
    printf("mulcnt = %d\n", *_mulcnt);

    dlclose(handle);
    return 0;
}
```

To compile: `gcc -rdynamic main3.c -ldl`

# Position-Independent Code (PIC)

Position-Independent Code

- Referencing symbols in the same executable object module → PC-relative addressing.
- Referencing external symbols → GOT (Global Offset Table)
- The code segments of shared modules can be loaded anywhere in memory without being modified by the linker
- Each process will get its own copy of the data segment

# PIC Data Reference

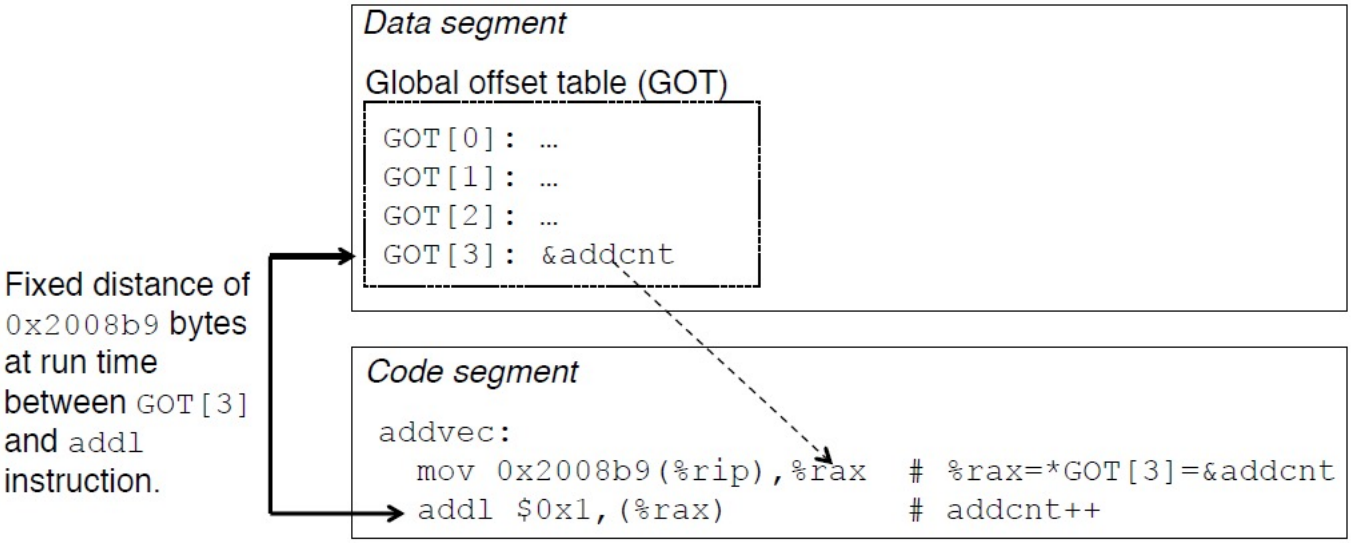
Global Offset Table (GOT)

- For each referenced global data objects (function, variable), a pointer entry is prepared that will be replaced by the absolute address of the object at the load-time

PIC reference

- The data segment is always at the same distance from the code segment
- Place GOT at the beginning of the data segment
- Each object module has its own GOT

# PIC Data Reference




`addvec` loads the address of `addcnt` indirectly from GOT[3]

The distance from `%rip` to GOT[3] is a constant (0x2008b9)

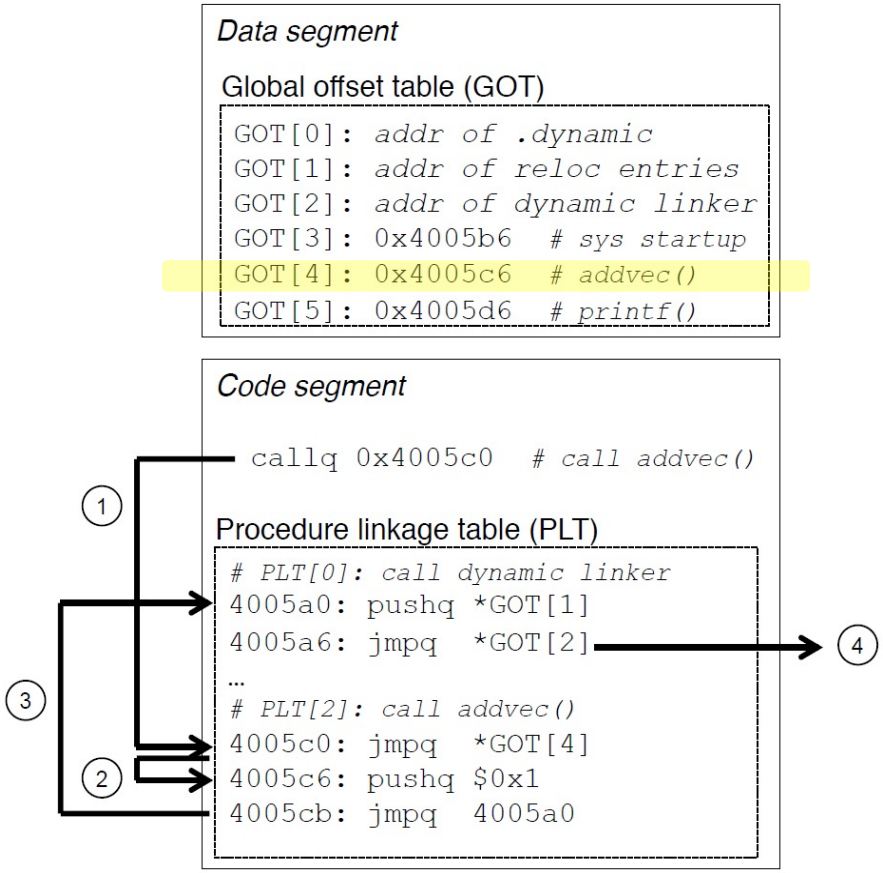
# PIC Function Call

PIC function call

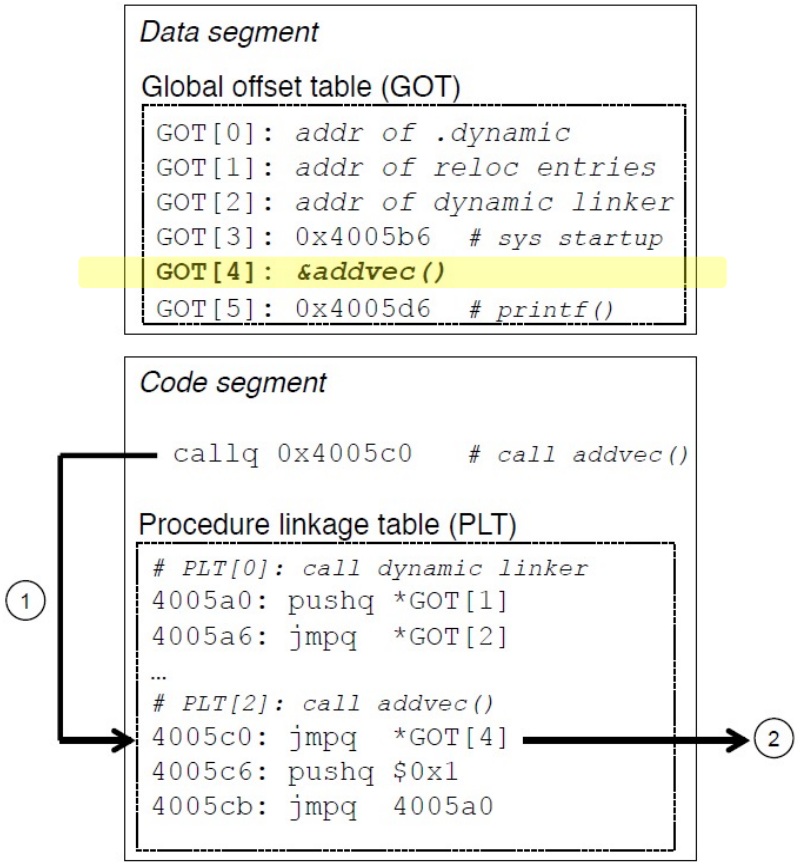
- Uses GOT and Procedure Linkage Table (PLT)
  - GOT is part of data segment
  - PLT is part of code segment

- Lazy binding: defers the binding of each procedure address until the first time the procedure is called

# PIC Function Call




First invocation of addvec




Subsequent invocations of addvec

# Questions?

Thank you for your attention during the semester!

Any questions or comments?