# Linkers and Libraries

Advisor: Dr. Gwan-Hwan Hwang
Lecturer: Chi Wu-Lee

## Agenda

- Introduction
- Static-Linking Libraries: Build & Usage
- Dynamic-Linking Libraries: Build & Usage

## Libraries

- A library is a collection of subprograms used to develop software.
  - Allows code and data to be reused, shared and changed in a modular fashion.
  - Linking: A linker resolves the references between executables and libraries.

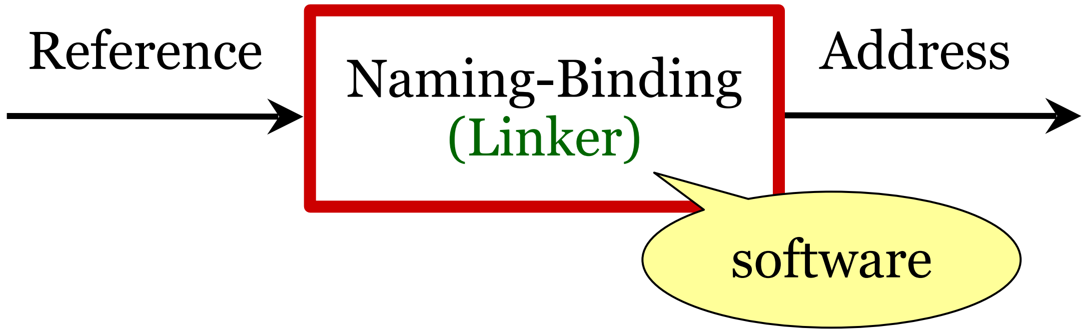
## Benefits of Using Libraries

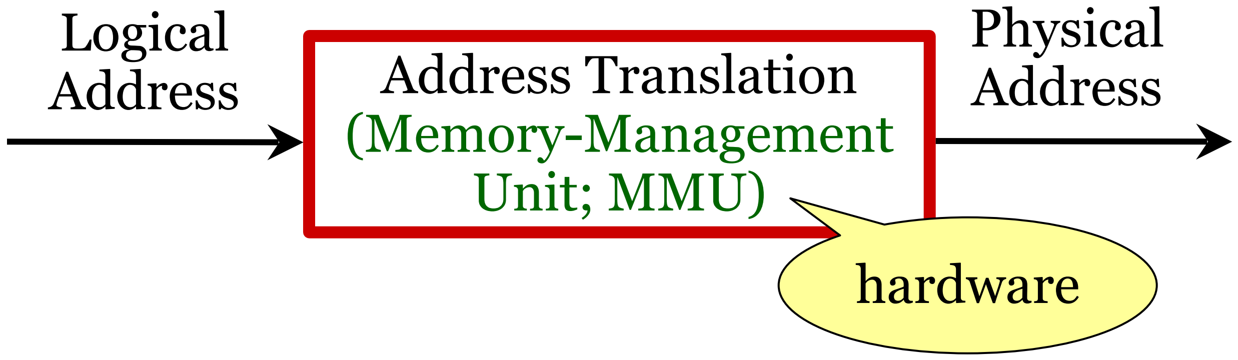
- Software Engineering Perspective:
  - Increasing the reusability of common routines.
  - Easy to upgrade by changing the libraries only.
- System Utilization Perspective:
  - The code segment can be sharing at runtime; decrease the consume of memory and disk space.

## Naming and Binding

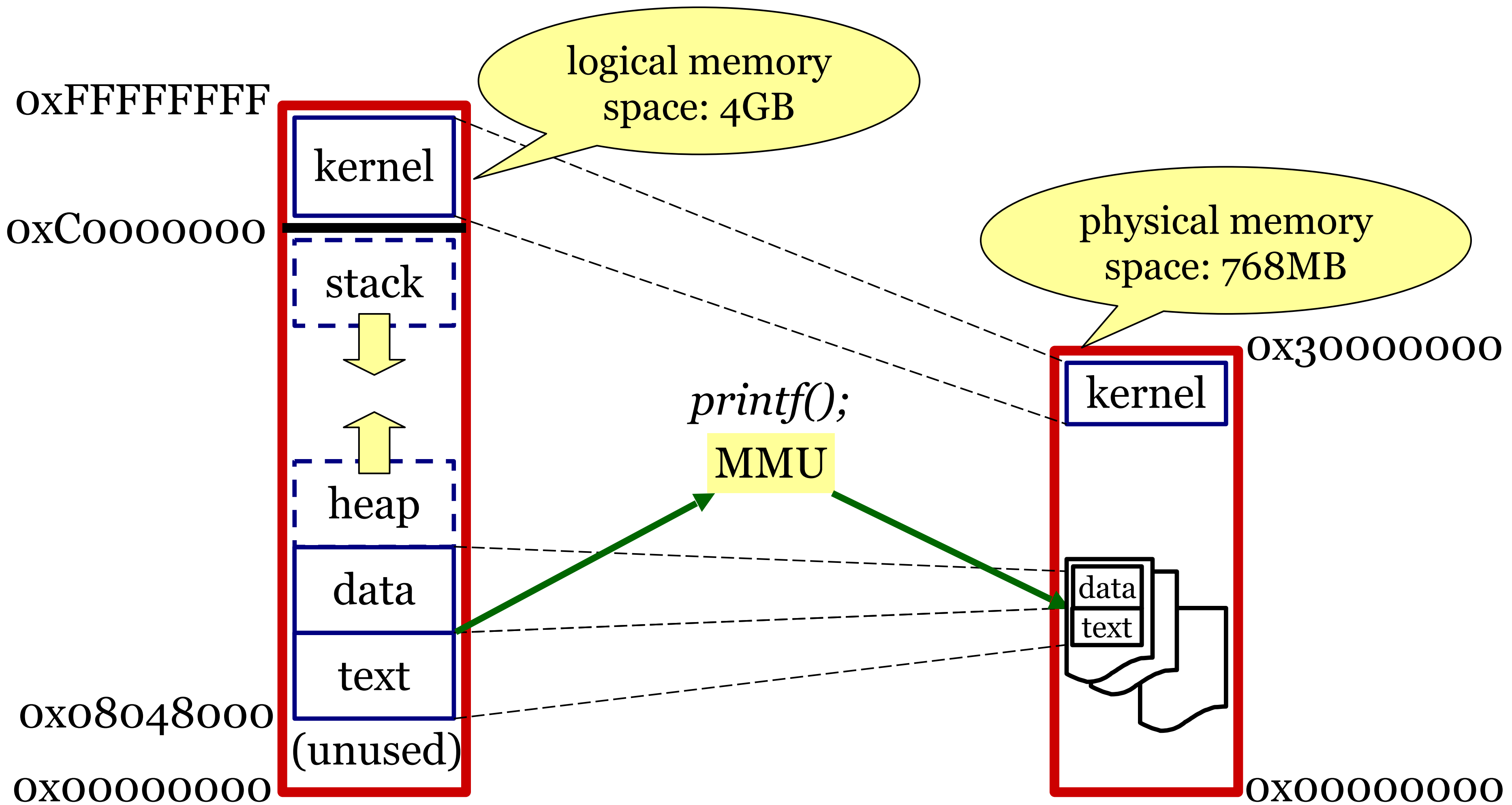
- Name Binding: The association of values with identifiers.
  - An identifier bound to a value is said to reference that value. (like DNS)
  - Reference Resolving: Retrieving the value/address by a reference.
  - Note: Different to the reference resolving, address translation is retrieving the memory address by another memory address.

# Naming-Binding & Address Translation





# Address Translation & MMU (Linux)



## Linkers

- Deal with modules.
- Find the library routines and determine the addresses at runtime.

## Print Shared Library Dependencies

- UNIX Platform

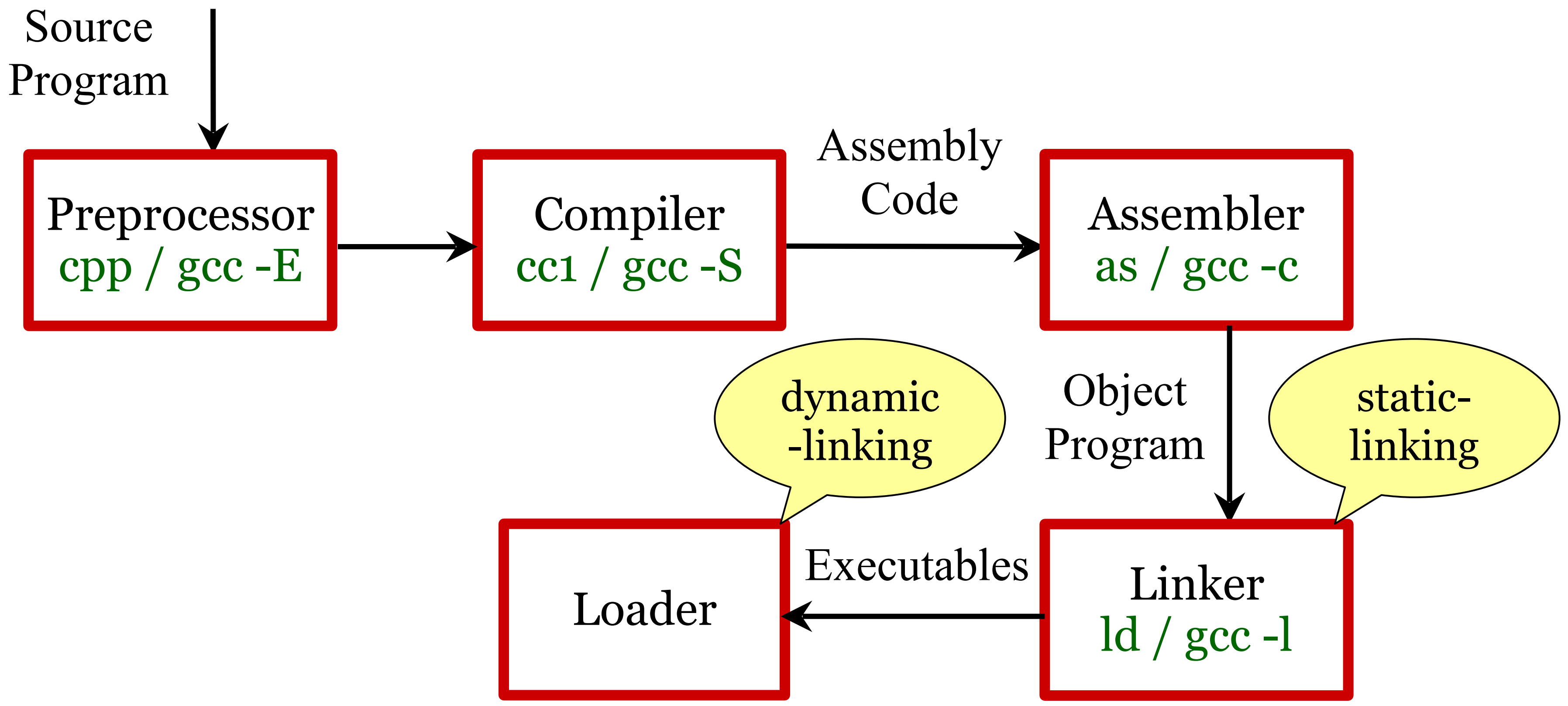
```
> ldd /bin/bash
    linux-gate.so.1 =>  (0xffffe000)
    libncurses.so.5 => /lib/libncurses.so.5 (0xb7f1d000)
    libdl.so.2 => /lib/libdl.so.2 (0xb7f19000)
    libc.so.6 => /lib/libc.so.6 (0xb7dfb000)
    /lib/ld-linux.so.2 (0xb7f61000)
```

- Windows Platform
  - Anywhere PE Viewer
  - Microsoft Process Explorer

## Categories of Libraries (by linking time)

- Static linking libraries
- Dynamic linking libraries
  - Run-Time Environment libraries
- Programming Language libraries
  - Interface (ex: .h)

# From Source to Execution (1/2)



# From Source to Execution (2/2)

| | |
|---|---|
| **Preprocessor** | cpp ./hello.c > main.c |
| **Compiler** | /usr/lib/i386-linux-gnu/gcc/i686-linux-gnu/4.5/cc1 main.c |
| **Assembler** | as -o main.o main.s |
| **Linker** | ld -o main.out /usr/lib/crt1.o /usr/lib/crti.o /usr/lib/i386-linux-gnu/gcc/i686-linux-gnu/4.5/crtbegin.o ./main.o -lc -lgcc -lgcc_s /usr/lib/i386-linux-gnu/gcc/i686-linux-gnu/4.5/crtend.o /usr/lib/crtn.o -L /usr/lib/i386-linux-gnu/gcc/i686-linux-gnu/4.5 -L /usr/i686-pc-linux-gnu/lib -L /usr/lib/ -dynamic-linker /lib/ld-linux.so.2 |

## Static Linking Libraries

- The code segments will be copy to each executables.

- Pros:
  - Easy to use; no dependency problem after compilation.
- Cons:
  - The executable size will be larger.
  - Require re-linking when libraries changed.

## Dynamic Linking Libraries (1/2)

- Allow multiple processes to share the same code segment.
- Pros:
  - Greater flexibility
  - Possible support for plug-ins.
- Cons:
  - Slow application at start time.
  - Dependent on the libraries when execution.

## Dynamic Linking Libraries (2/2)

- The references can be resolved either at:
  - Load-time
  - Run-time
- UNIX Platform
  - “shared-object”: lib*.so
- Windows Platform
  - “dynamic-linking library”: *.dll

## Location of Libraries

- UNIX Platform
  - /lib: runtime environment libraries
  - /usr/lib: for program development
- Windows Platform
  - C:\WINDOWS\system32\
  - The libraries for program development will be accompanies with compiler, like: Visual C++.

## Linking with C Runtime Libraries

- Static Linking
  - gcc -static -o hello-s hello.c /usr/lib/i386-linux-gnu/libc.a
  - hello-s size: 632K
- Dynamic Linking
  - gcc -o hello-d hello.c
  - hello-d size: 7.0K

- The CRT libraries consume 625K

## Static-Linking Libraries

- Build
  - gcc -c sayhello.c => create sayhello.o
  - ar rcs libfoo.a sayhello.o
- Usage
  - gcc -static -o hello-s main.c -L. libfoo.a

## GNU Binary Utilities

- strings: display all printable characters.
- ar: create static-linking libraries.
- size: list section sizes and total sizes.
- objdump: de-assemble the specified section from object files.

# Dynamic Linking

- Static Shared Libraries
- Dynamic Shared Libraries

# Dynamic Linking Library

- Build
  - gcc -fPIC -c sayhello.c
    - =>create position-independent code
  - gcc -shared sayhello.o -o libmylib.so
    - =>create shared object file
- Usage
  - gcc -o hello-d main.c libmylib.so
- Runtime Environment Variable
  - export LD_LIBRARY_PATH=$LD_LIBRARY_PATH: path (.)

# GCC – GNU Compiler Collection

- cpp:preprocess macros(preprocess)
- cc1: perform semantic routines and translate into assembly language(compiler)
- as: assemble to relocatable object files(assembler)
- ld: linking(linker)
- To view the commands executed to run the stages of compilation.
  - gcc -v