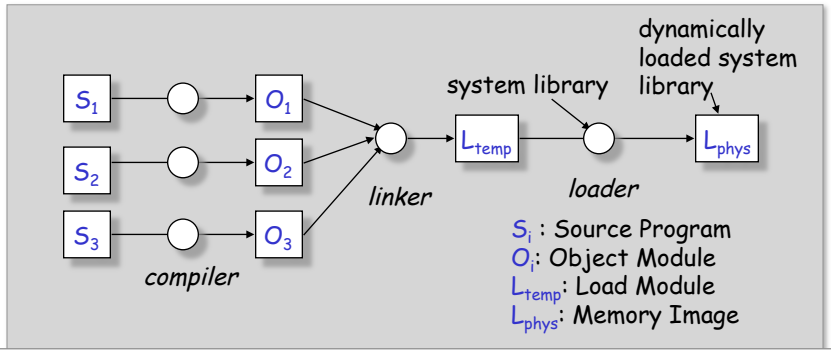
Linking and Loading

- Preparing Program for Execution
- Relocation
- Address binding
  - Linking, loading

- *Reading: Doeppner 3.4*

Preparing a Program for Execution



- compiler: translates symbolic instructions, operands, and addresses into numerical values.
- linker: resolves external references; i.e. operands or branch addresses referring to data or instructions within some other module
- loader: brings program into main memory.

Steps Involved

1. Name /Symbol Resolution
2. Relocation

Linking
linker: `ld` (GNU Linux)

3. Program Loading

Loading
loader: `execve` (GNU Linux)

Name/Symbol resolution

File *subr.c* :

```
int X;

void subr(int y) {
  int x = y;
}
```

This file can stand on its own

File *main.c* :

```
extern int X;
int *aX = &X;

int main() {
    void subr(int);
    int y = X;
    subr(y);
    return 0;
}
```

Cannot stand on its own
Where is X defined?

Where do we jump when we need to execute subr() ?

Symbol Resolution

- Compile source code into object files
- Object files?
  - Contains compiled source
  - Symbol tables
  - Relocation data

(Can view contents of Object files using `objdump` (in linux))

- Two types of Object files
  - Relocatable Object files
  - Shared Object files

Symbol resolution

```
int X;          //subr.c :

void subr(int y) {
  int x = y;
}
```

f_1

```
extern int X;          //main.c
int *aX = &X;
int main() {
    void subr(int);
    int y = X;
    subr(y);
    return 0;
}
```

f_2

$O = f_1 \cup f_2 \ldots$

$O \leftarrow O \cup \{ f_i . O \}$

D1

U1

D2

U2

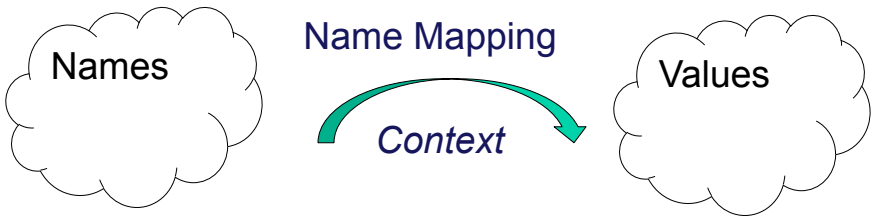
$D \leftarrow D \cup \{D_i\}$

$U \leftarrow (U \cup U_i) - (U \cap D)$

Symbol resolution with a library

- Libraries?
  - libc / libm
  - Combination of many object (.o) files
- Linker goes looking at libraries if the set U is NOT NULL

Generalization for Name Resolution



1. Table lookup
   - Single Object file (Symbol table)
2. Path name resolution
   - /home/user/suneil …
3. Search through contexts
   - Linker

Linking: Relocation

```
int main (int argc, char *[]) {
    return argc;
}
```

```
main:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %eax
        movl    %ebp, %esp
        popl    %ebp
        ret
```

This code is inherently relocatable.

Linking: Relocation (cont)

```
int X=6;
int *aX = &X;

int main () {
    void subr(int);
    int y = X;
    subr(y);
    return 0;
}

void subr(int i) {
    int x = i;
}
```

```
.globl _X
        .data
_X:
        .long   6
.globl _aX
_aX:
        .quad   _X
        .text
.globl _main
_main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movl    _X(%rip), %eax
        movl    %eax, -4(%rbp)
        movl    -4(%rbp), %edi
        call    _subr
        movl    $0, %eax
        leave
        ret
.globl _subr
_subr:
        pushq   %rbp
```

This code is not freely relocatable!
e.g.:
- Variable **aX** needs to know location of **X**.
- Call to **subr()** needs to know location.

Linking: Relocation (cont)

File *main.c* :

```
extern int X;
int *aX = &X;

int main() {
    void subr(int);
    int y = X;
    subr(y);
    return 0;
}
```

```
$> gcc main.c
Undefined symbols:
  "_subr", referenced from:
      _main in ccLXS9NR.o
  "_X", referenced from:
      _main in ccLXS9NR.o
      _aX in ccLXS9NR.o
ld: symbol(s) not found
collect2: ld returned 1 exit status
$>
```

```
gcc -o prog main.c subr.c
```

File *subr.c* :

```
int X;

void subr(int y) {
  int x = y;
}
```

This code is **not** freely relocatable!
e.g.:
- Variable **aX** needs to know location of **X**.
- Call to **subr()** needs to know location of **subr()**.

Linking: Relocation (cont)

File *main.c* :

```
extern int X;
int *aX = &X;

int main() {
    void subr(int);
    int y = X;
    subr(y);
    return 0;
}
```

```
.globl _aX
        .data
_aX:
        .quad   _X
        .text
.globl _main
_main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movq    _X@GOTPCREL(%rip), %rax
        movl    (%rax), %eax
        movl    %eax, -4(%rbp)
        movl
        movl
        call
        movl
        leave
        ret
```

File *subr.c* :

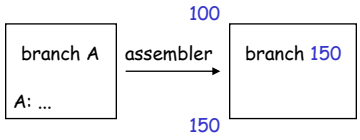
```
int X;

void subr(int y) {
  int x = y;
}
```

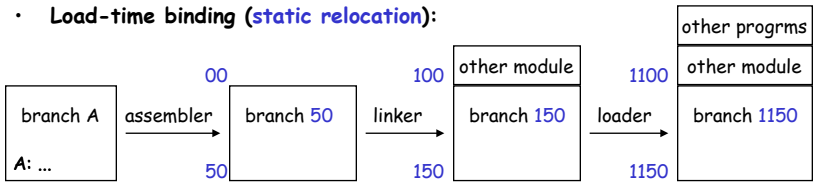
```
 .text
.globl _subr
_subr:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -20(%rbp)
        movl    -20(%rbp), %eax
        movl    %eax, -4(%rbp)
        leave
        ret
.comm _X,4,2
```

Address Binding

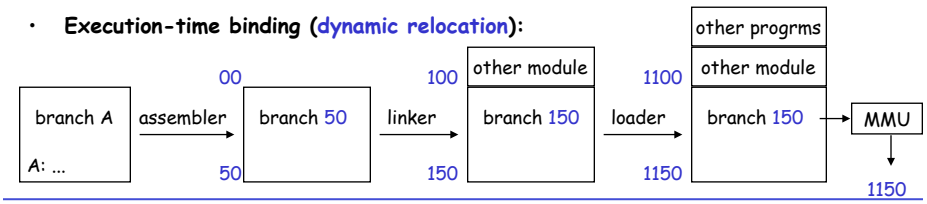
- **Compile-time binding:**



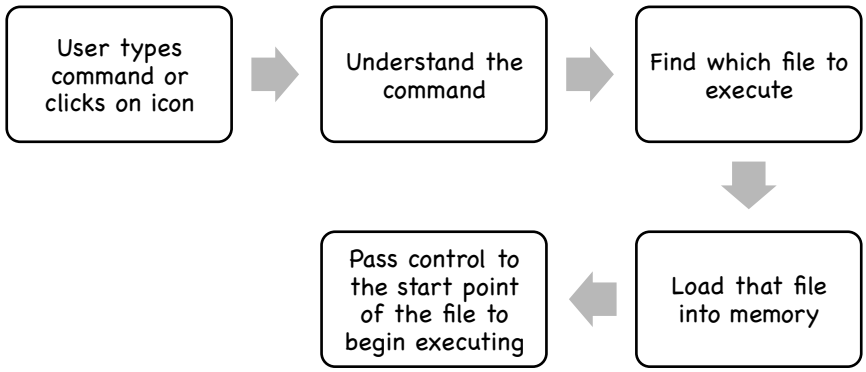
- **Load-time binding (static relocation):**



- **Execution-time binding (dynamic relocation):**



Loading



Shared Objects

- Do not embed common routines in every program
  - Have one copy of the module and share it between multiple programs
- Instructions associated with memory locations.
  - Different programs may call the shared module with different address

~~Jump to location 150~~

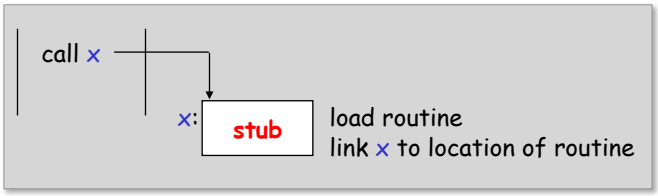
Solution: Position independent code :

All addresses are relative (to program counter)

Jump to offset X from here.

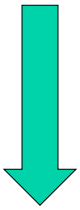
Dynamic Loading, Dynamic Linking

- Dynamic Loading:
  - load routine into memory only when it is needed
  - routines kept on disk in relocatable format
- Load-time Linking:
  - postpone linking until load time.
- Dynamic Linking:
  - postpone linking until execution time.
  - Problem: Need help from OS!



Example

```
#include <stdio.h>
int main (int argc, char *[]) {
       printf("Hello World");
       return 0;
}
```



```
gcc hello.c -o helloworld
./helloworld
```

Symbol Resolution/ Link/ Load

- Figuring out where all the symbol definitions are
  - printf() … ?

> printf is complicated…
> *Takes at least 2 arguments, can be more.*
>
> `printf ("%d",&IntegerVariableName);`
>
> (in C/C++ Is there a limit to the number of arguments?)

- Look in other related files, object files, libraries (static and dynamic), environment variables.
- Load and run the program