Empowered by Innovation NEC

A Consideration of Memory Saving by Efficient Mapping of Shared Libraries

Masahiko Takahashi

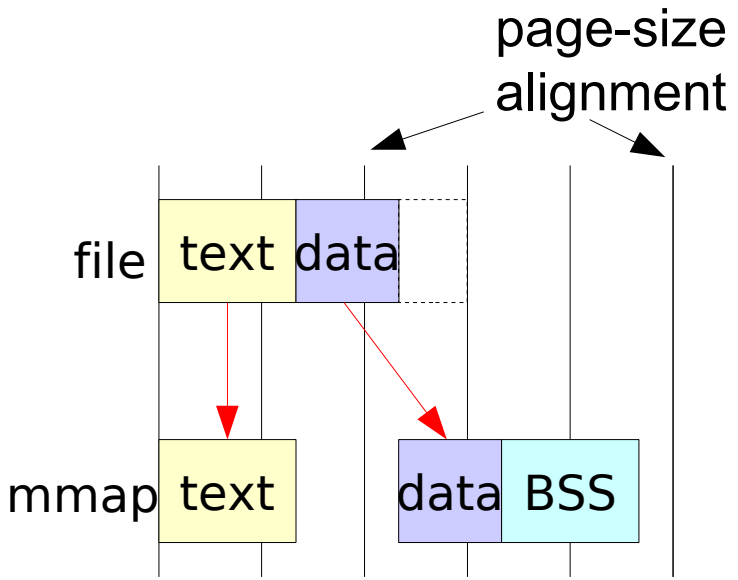
System Platforms Research Laboratories
NEC Corporation

April 12, 2010

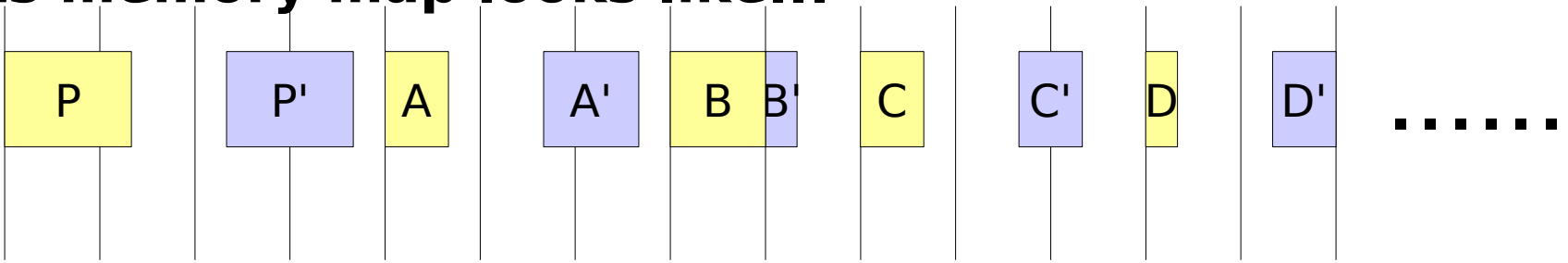
Embedded Linux Conference 2010

Introduction: How shared library is used

- Shared libraries are mapped to memory by using mmap
- Position independent code (PIC) uses PC-relative memory access



If process "P" links libA.so, libB.so, libC.so, and libD.so, its memory map looks like...

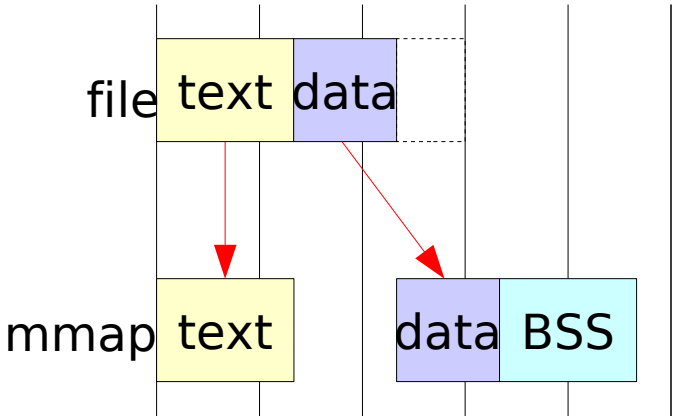


Text area (read/exec)

Data+BSS area (read/write)

mmap – pros

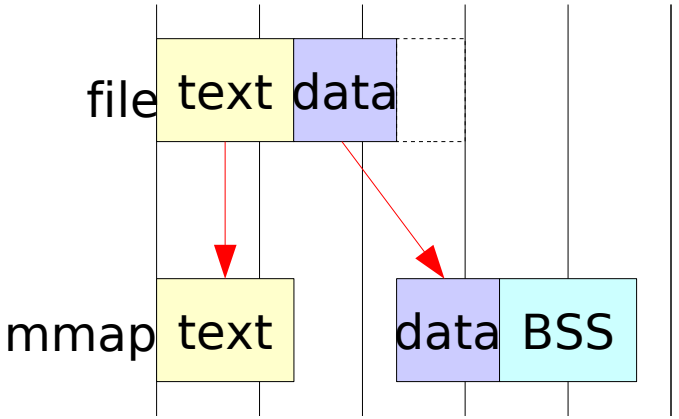
- Mmap provides demand-paging
- Pros
  - A physical page is not assigned until the first access to the page



mmap – cons

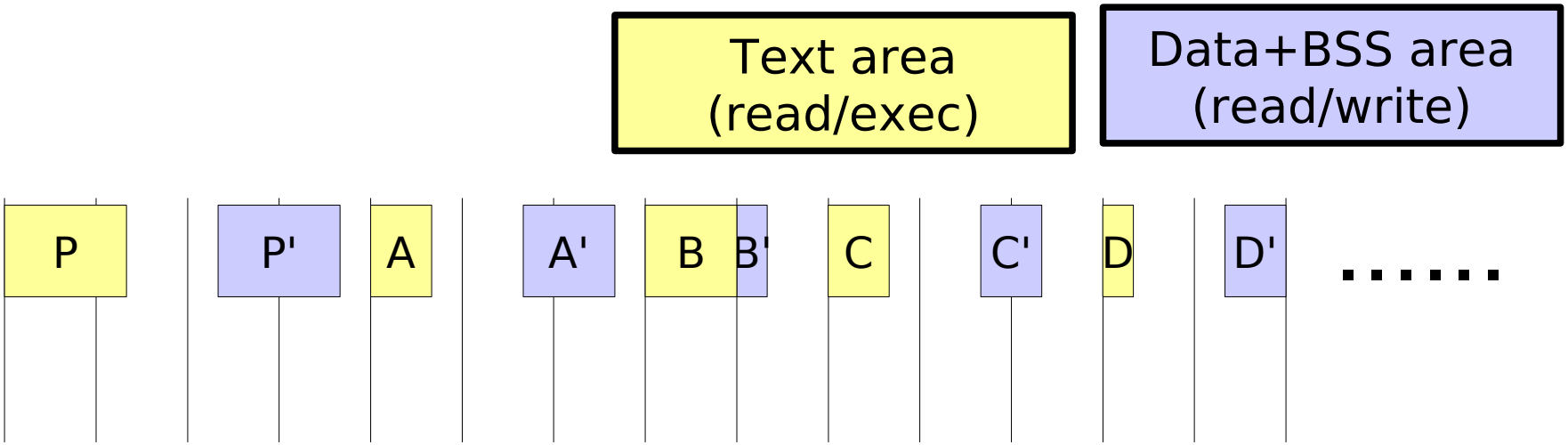
Cons

- Page-fault overhead
- Page-size alignment for each text and data area of a library
  → (internal) memory fragmentation



Internal memory fragmentation

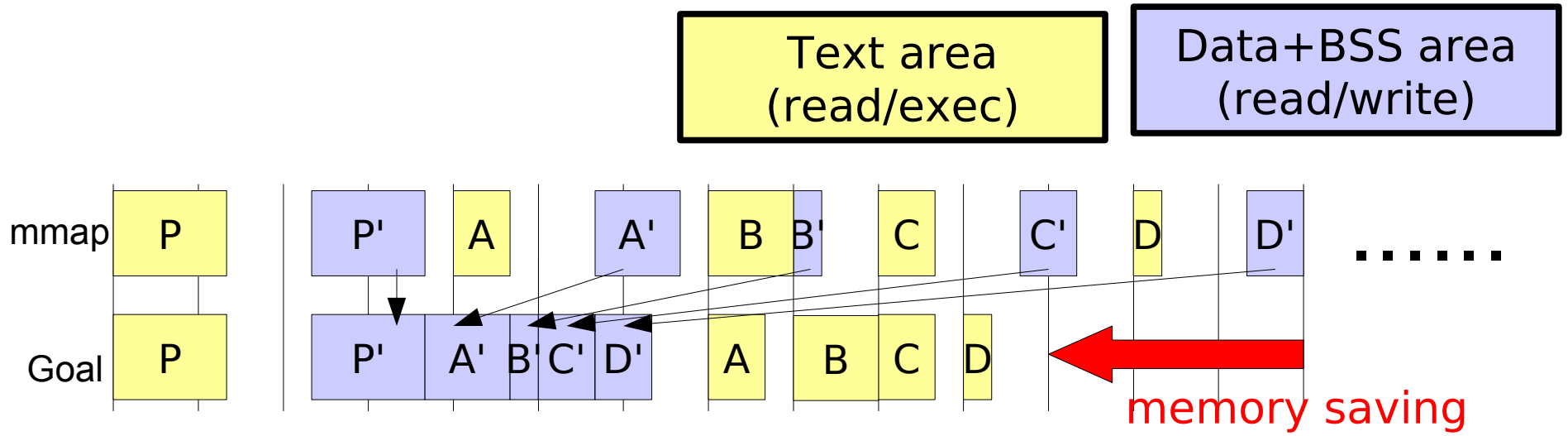
- When mmaped, each shared library is mapped in "text(non-writable), then, data(writable)" order,
- which means there are internal memory fragmentation between areas.



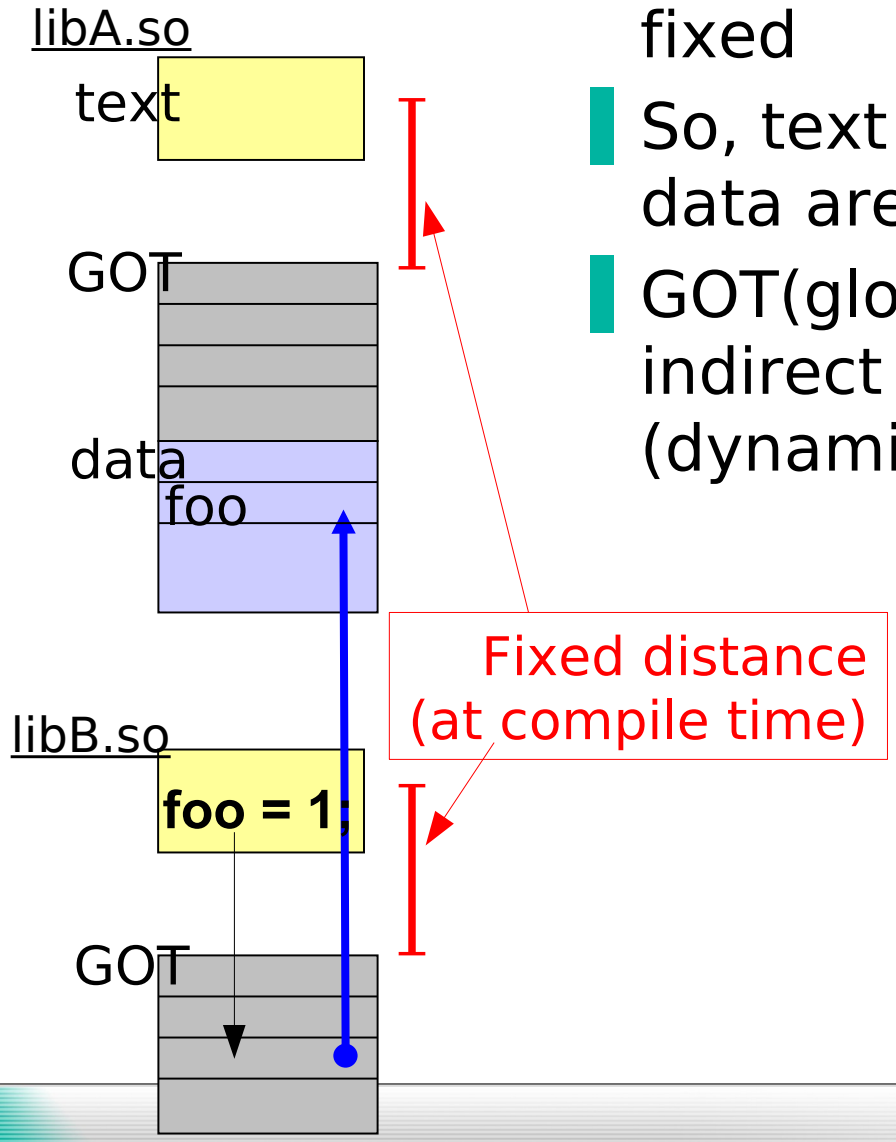
Goal: Eliminating the fragmentation for memory saving

Idea:

- Put all data areas into one area
- Use “read” systemcall, not “mmap”
  - Can't share a page between libraries when using mmap



Why “text-data” order is needed?



- The Load address of a library isn't fixed
- So, text area needs to access to data area by the fixed distance
- GOT(global offset table) is used for indirect access to data area (dynamic symbol resolution)

```
setFoo:
        load    r3, .GOT - .LPIC
        add     r3, pc, r3
.LPIC:
        load    r3, [r3, #8]
        store   r0, [r3]
        ret
.GOT:
        word ...
        word ...
```

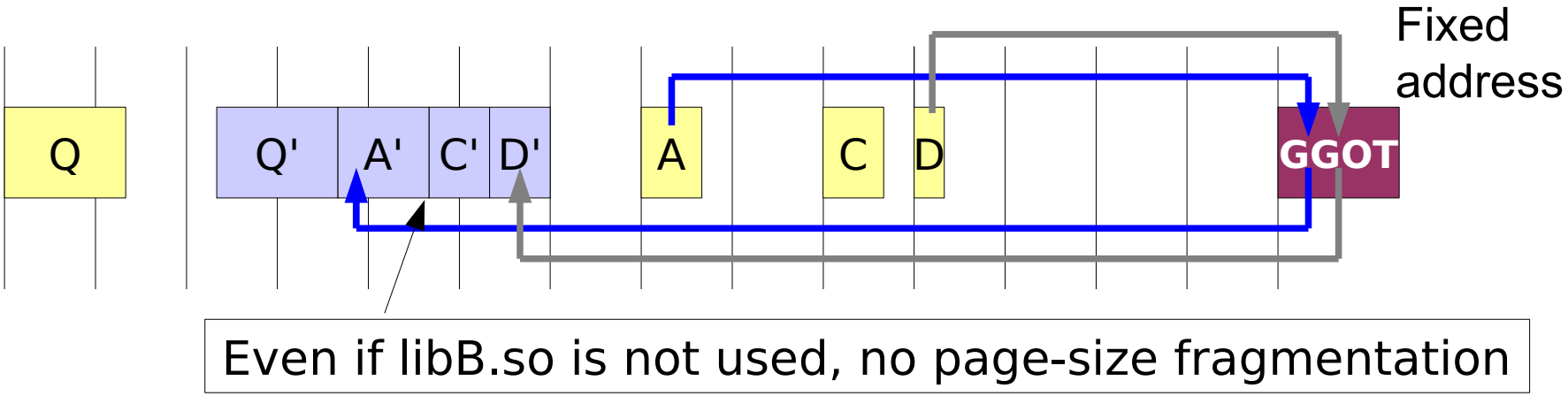
More generally … ***fix*** something

- How to access to data area from text area
  - Dynamic link(PIC): it needs the distance between text and data area to be fixed
    - Access with PC-relative address
    - Issue: internal memory fragmentation; due to page-size alignment
  - Static link: it fixes the address of data area by absolute address
    - Access with absolute address
    - Issue: text areas are also linked statically and cannot be shared between processes

Proposal: introducing fixed GGOT area

- Introduce a “Global GOT area”, on the fixed address, which is for indirect access to libraries' GOT areas
  - Access with absolute address
  - Assigned to each process independently

→ Not fixed load address, nor fixed distance for text/data



Pros and cons of the proposal

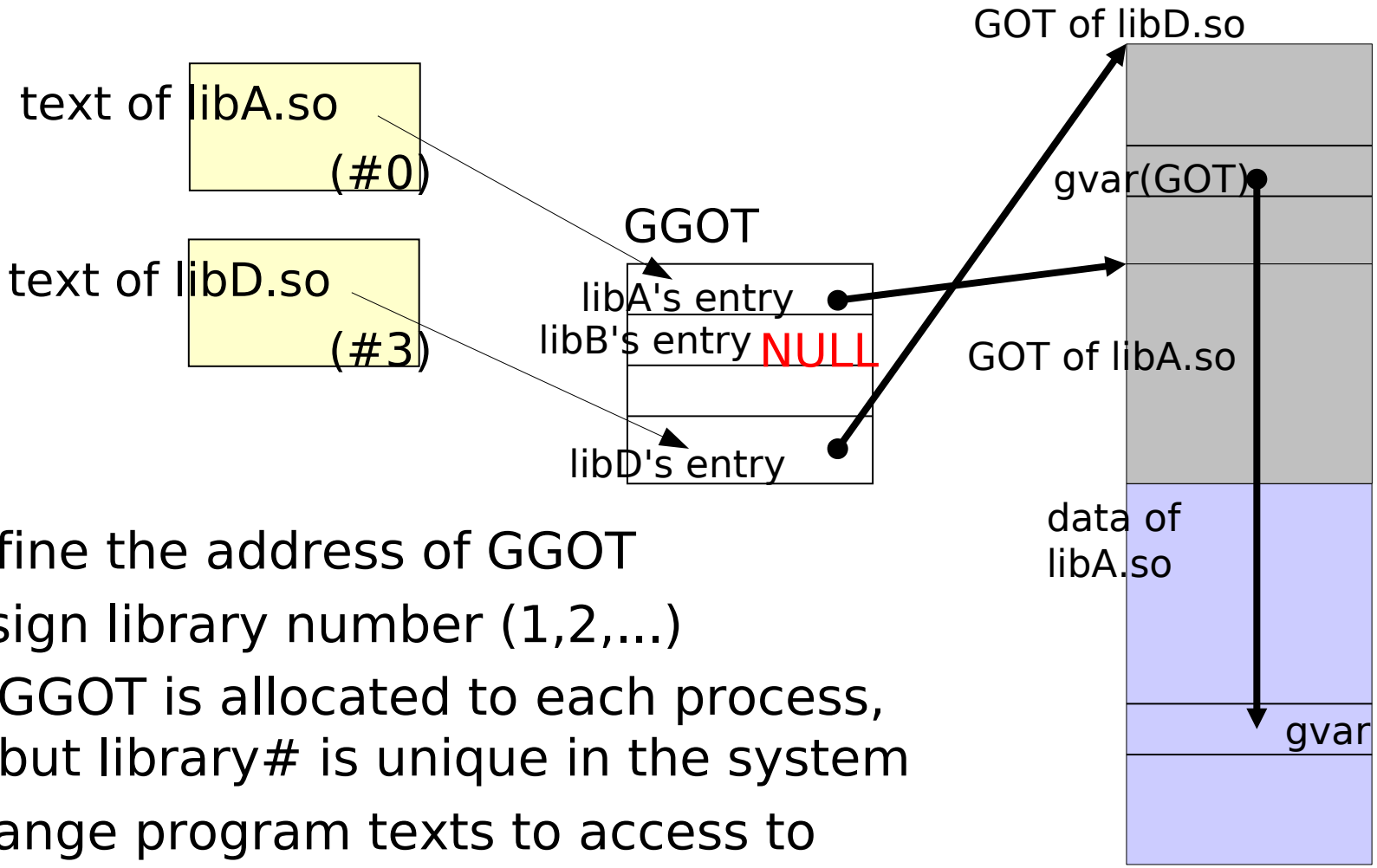
Pros

- Eliminating internal memory fragmentation

Cons

- Lack of demand-paging for data areas
  - Because of using “read” systemcall for whole data areas (instead of mmap)
- Tiny overhead in indirect access to GOT via GGOT

Implementation Design



- Define the address of GGOT
- Assign library number (1,2,...)
  - GGOT is allocated to each process, but library# is unique in the system
- Change program texts to access to "GGOT address + its library# offset"

Prototype Implementation

- The prototype is on x86 Linux with glibc-2.7
- The following 2 modifications are needed
  - Modify ld.so (ELF loader)
  - Binary rewrite of shared libraries' text

Modification of ld.so

- Changes are approximately 60 lines (3 parts)
  1. Allocate memory for GGOT area
  2. Write the load address of GOT area of each shared library to its corresponding GGOT entry
  3. Address recalculation in symbol relocation, e.g.,
     - R_386_GLOB_DAT
     - R_386_JMP_SLOT
     - DT_PLTGOT
     - DT_FINI_ARRAY/DT_INIT_AFFAY?

Binary rewrite of shared libraries

- Just for prototyping
- Mainlining to gcc/gld is the right way

- We notice that at least the following three rewrites are needed
  1. Change GOT access to GGOT access in normal function
  2. Change GOT access to GGOT access in _init() and _fini()
  3. Change offset value in accessing RODATA area

1. Change GOT access to GGOT in normal functions

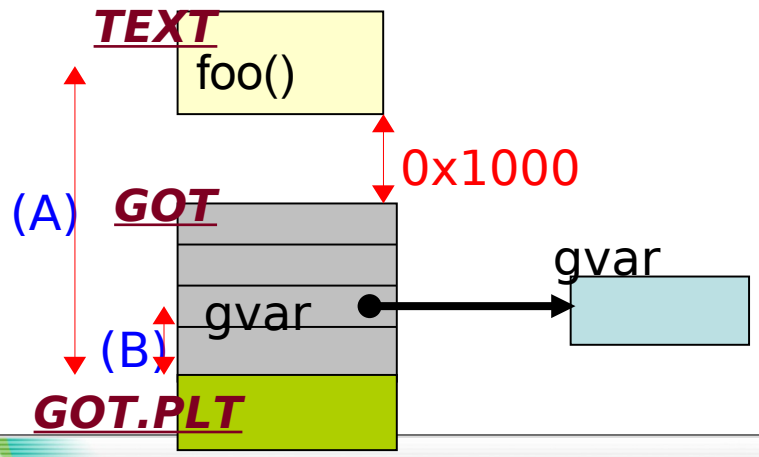
PIC

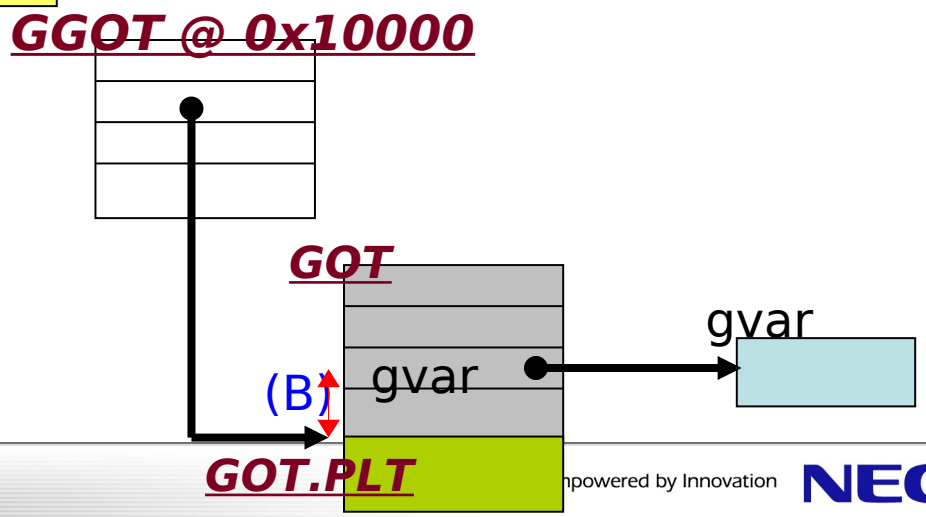
```
foo:
    call    get_pc_thunk.bx
    add     $0x141a,%ebx      …(A)
    mov     -0x18(%ebx),%ecx…(B)
    mov     %eax,(%ecx)
    ret

get_pc_thunk.bx:
    mov     (%esp),%ebx
    ret
    nop
```

Proposal Lib# offset

GGOT address

```
foo:
    mov     $0x10000,%ebx
    mov     0x4(%ebx),%ebx
    mov     -0x18(%ebx),%ecx…(B)
    mov     %eax,(%ecx)
    ret
```





2. Change GOT access to GGOT in _init()/_fini()

PIC

```
_init:
       call LP0
 LP0:  pop  %ebx
       add  $0x15c0,%ebx
       mov  -0x10(%ebx),%edx
       test %edx,%edx
       je   LP1
       call __gmon_start__@plt
 LP1:  call frame_dummy
       call __do_global_ctors_aux
```

Proposal

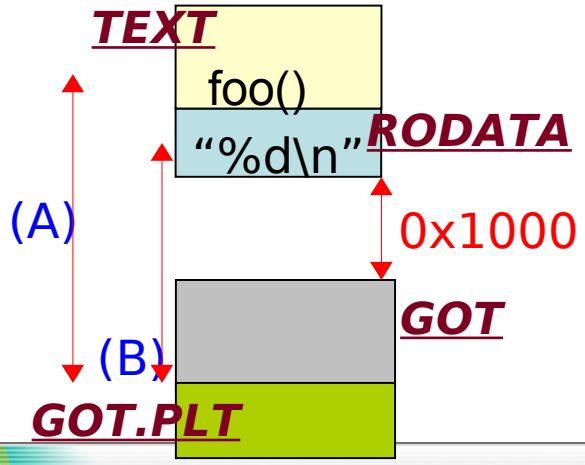

```
_init:
       mov  $0x10000,%ebx
       mov  0x4(%ebx),%ebx
       nop
       mov  -0x10(%ebx),%edx
       test %edx,%edx
       je   LP1
       call __gmon_start__@plt
 LP1:  call frame_dummy
       call __do_global_ctors_aux
```

3. Change offset value in accessing RODATA area

PIC

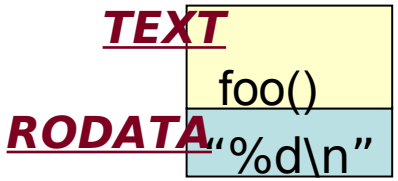
printf(“%d\n”,i);

```
foo:
    call get_pc_thunk.bx
    add  $0x141a,%ebx          …(A)

    lea  -0x1174(%ebx),%eax…(B)
    mov  %eax,(%esp)
    call printf@plt
     :
    ret
```



Proposal

```
foo:
     mov   $0x10000,%ebx
     mov   0x4(%ebx),%ebx
     call LP0
LP0:pop   %ebx
     lea   xxxxxx(%ebx),%eax
     mov   %eax,(%esp)
     call  printf@plt
      :
     ret
```



Unfortunately, 6bytes are not sufficient to rewrite instructions in x86 architecture.

3. Change offset value in accessing RODATA area (cont.)

printf("%d\n",i);

PIC

```
foo:
    call get_pc_thunk.bx
    add  $0x141a,%ebx          …(A)
    lea  -0x1174(%ebx),%eax…(B)
    mov  %eax,(%esp)
    call printf@plt
     :
    ret
```

Proposal

```
foo:
    mov   $0x10000,%ebx
    mov   0x4(%ebx),%ebx
    call  bar1
    nop
    mov   %eax,(%esp)
    call  printf@plt
     :
    ret

bar1:
    mov    (%esp),%eax
    lea    0x1a9(%eax),%eax
    ret
```

Unfortunately, 6bytes are not sufficient to rewrite instructions. So, a function named "bar?" is added to set the offset. All *bar* s Should be put together into one area.

Future Work

- Verification and evaluation
- Mainlining (ld.so and gcc/gld)
- Selective use of GGOT or mmap for libraries

Conclusion

- For memory saving, efficient memory mapping of shared libraries is proposed
- The prototype on x86 Linux required two modifications, but implemented in mainlins (ld.so and gcc/gld) is the right way

- I need your help;
  - Basic idea, implementation, and verification,
  - to push into mainline,
  - and other things ...

Thank you.

Questions ?

人と地球にやさしい情報社会を
イノベーションで実現する
グローバルリーディングカンパニー

To be a leading global company
leveraging the power of innovation
to realize an information society
friendly to humans and the earth

NECグループビジョン2017

NEC Group Vision 2017

Empowered by Innovation