Chapter 2

PROGRAM

Table of Contents:

2.1 MONOLITIC PROGRAMS
2.1.1 Static Linking 2-3
2.1.2 Loading 2-4
2.1.3 Execution Environment of a Program 2-6

2.2 MODULAR PROGRAMS
2.2.1 Dynamic Linking (general) 2-7
2.2.2 Dynamic Linking (at load time) 2-8
2.2.3 Dynamic Linking (at run time) 2-10
2.2.4 Shared Libraries 2-11
2.2.5 Memory Mapping of Address Spaces with DLLs 2-12
2.2.6 An Example of Dynamic Linking 2-13
2.2.7 Traps: Division of the Address Space 2-18
2.2.8 Execution Environment of a Program (Revisited) 2-20
2.2.9 Layered Design of an Operating System 2-21

Copyright © 1998-2002 by Marko Vuskovic

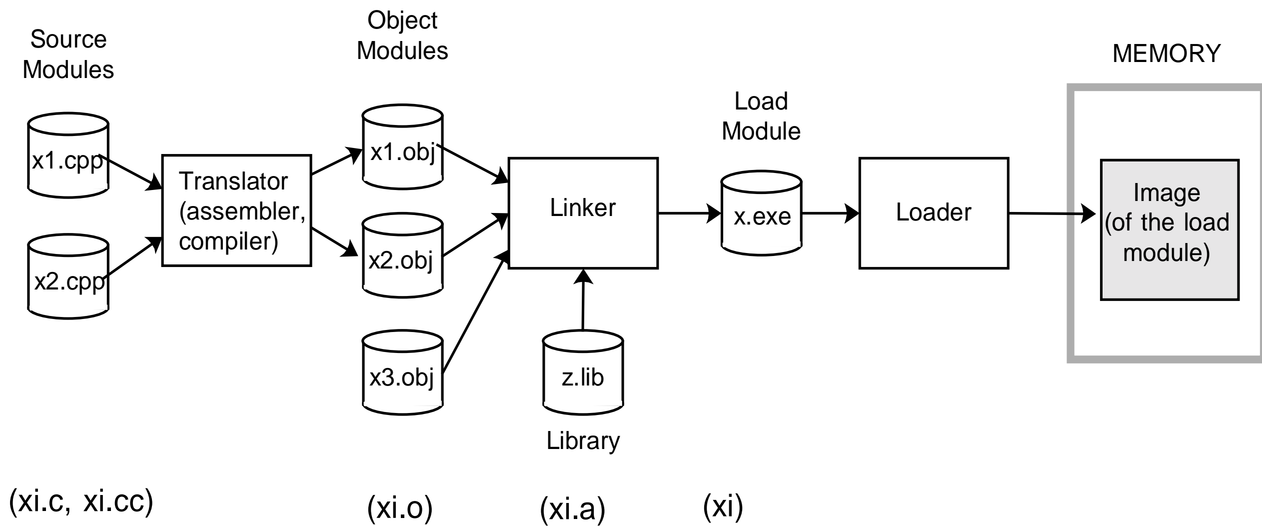
Modern operating systems are not monolithic programs, with a single monolithic executable image. They rather consist of many (hundreds and thousands) of images that are located in different areas of the memory and have different execution privileges. There are two basic mechanisms that support this modularization:

- **Dynamic linking**
- **Software traps**

These mechanisms will be discussed in this chapter. However the static linking and the execution environment of monolithic images will be discussed first.

Copyright © 1998-2002 by Marko Vuskovic

STATIC LINKING
(Generation of a Monolitic Image)



(xi.c, xi.cc) (xi.o) (xi.a) (xi)

Solaris:

```
ar xxx.a *.o
cc -o p main.c xxx.a
```

(xxx = library name)

Windows:

```
lib -out:xxx.lib  x.obj y.obj ...
cl -c main.cpp
link -out: p.exe main.obj  xxx.lib
```

Copyright © 1998-2002 by Marko Vuskovic

LOADING

Source Code

```
org $2000
start:
  move A,d1
  add B,d1
  jmp exit
  .......
  .......
exit:
  move d1, R
  .......
A dc.l 9
B dc.l 13
R ds.1
end
```

Absolute Load Module

```
2000  move 3000,d1
      add 3004,d1
      jmp 2200
      .........
      .........
2200  move d1,3008
      .........
      .........
3000  0009
3004  000D
3008  0000
```

Relocatable Load Module

Program Segment

```
0000  move 0000{d},d1
      add 0004{d},d1
      jmp 0200{p}
      .........
      .........
0200  move d1,0008{d}
      .........
      .........
```

Data Segment

```
0000  0009
0004  000D
0008  0000
```

Relocatable load modules can be placed at any location in memory.

Relocation bits:
- {0} - do not relocate
- {p} - relocate wrt program origin
- {d} - relocate wrt data origin

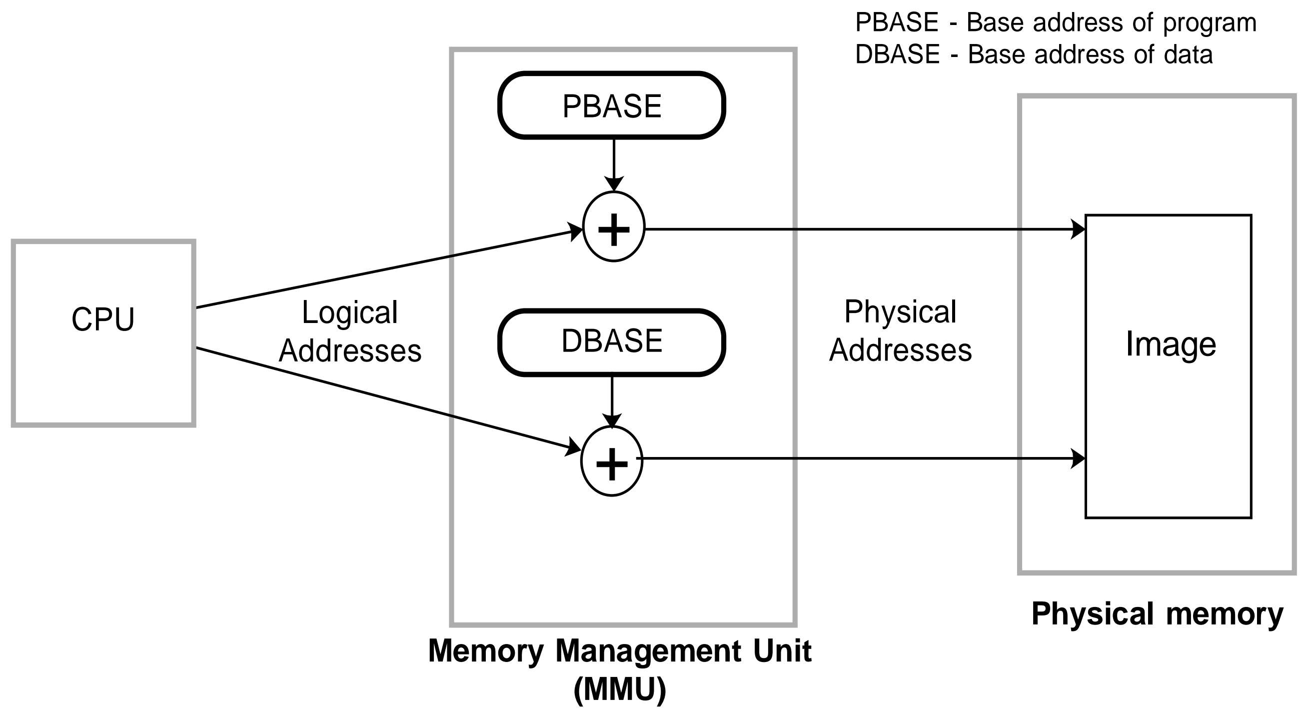
(Relocation bits can be placed with instructions, or in one place: relocation directory)

Relocatable loading is not convenient for multiprogramming environments, because the programs are swapped-in and out all the time.
Therefore the run-time loading is solution (see next page).

Copyright © 1998-2002 by Marko Vuskovic

LOADING (Cont.)

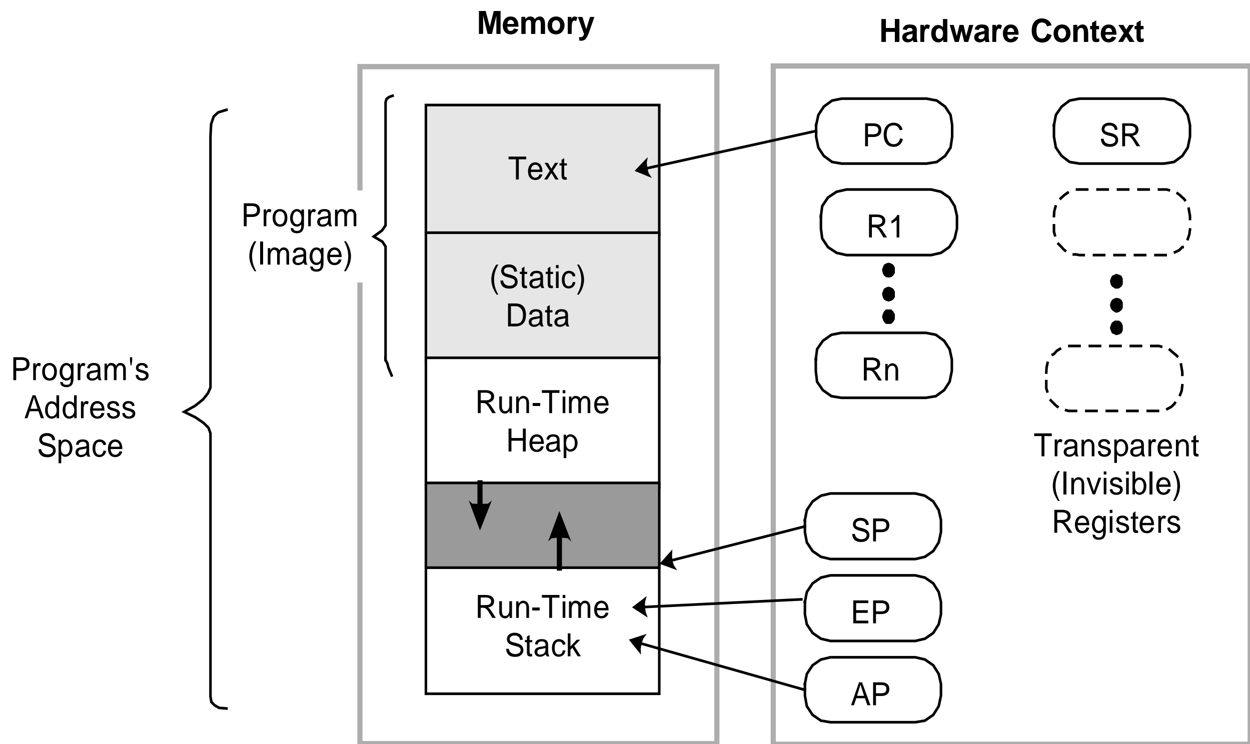
Dynamic (Run-Time) Loading



Absolute address is calculated at instruction execution time

Copyright © 1998-2002 by Marko Vuskovic

EXECUTION ENVIRONMENT OF A PROGRAM



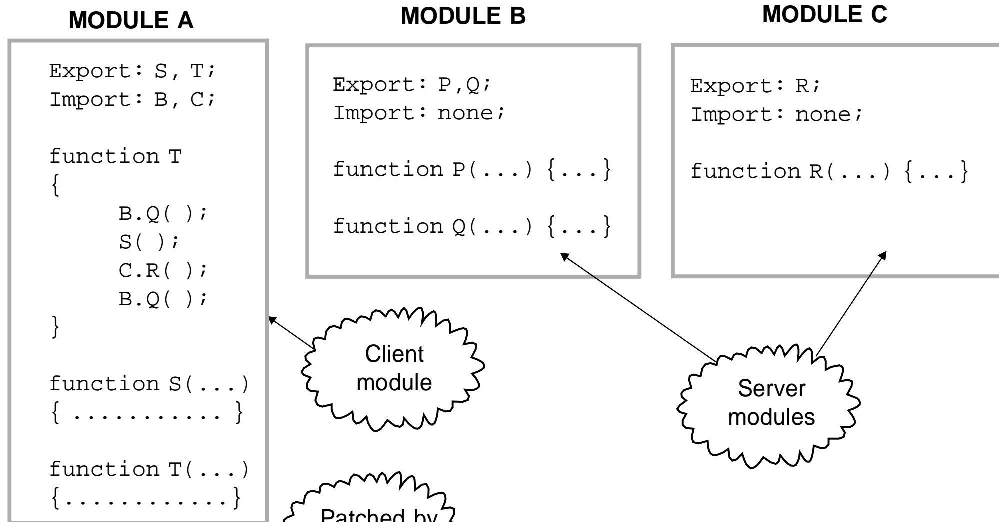
Program's address space (AS) =
Range of addresses that program can access

Text = Code part of the program image

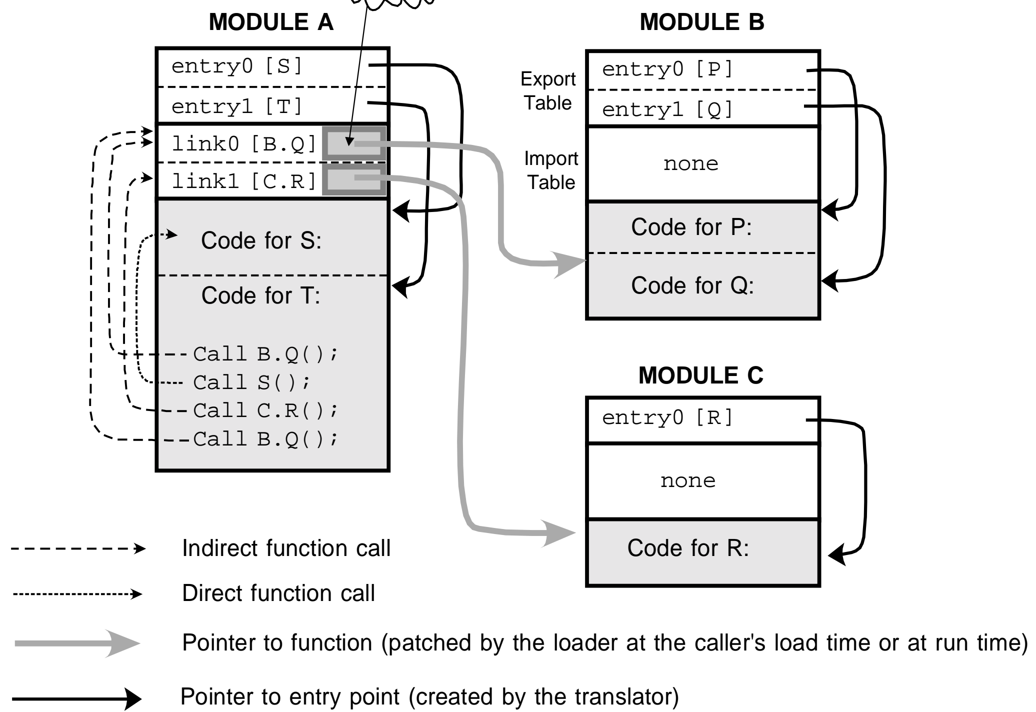
Copyright © 1998-2002 by Marko Vuskovic

DYNAMIC LINKING

Sources:

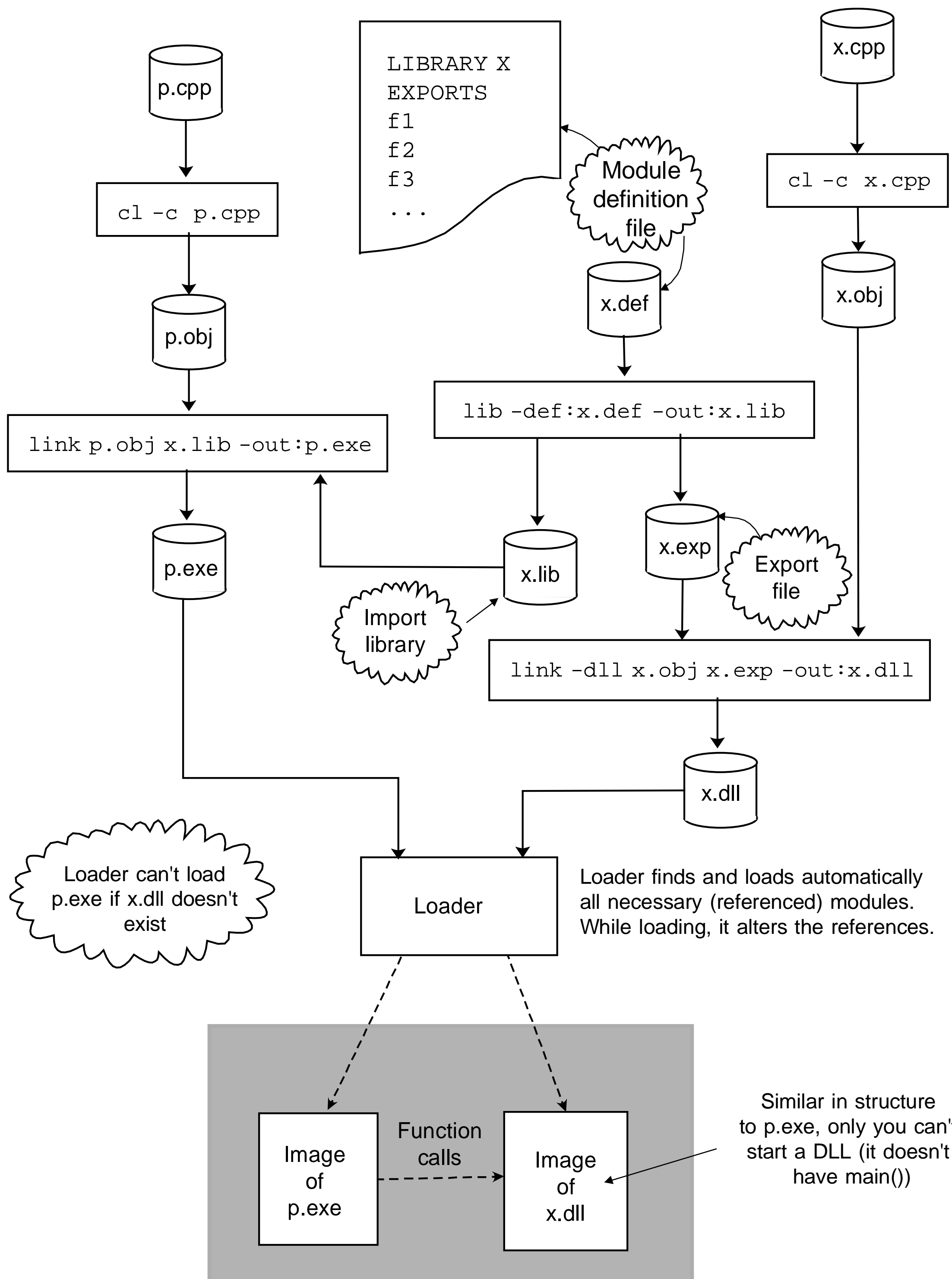


Object Modules:



Copyright © 1998-2002 by Marko Vuskovic

DYNAMIC LINKING (AT LOAD-TIME)



Copyright © 1998-2002 by Marko Vuskovic

Comment:

File `x.lib` created by the library manager using the command:

```
lib -out:x.lib x.obj
```

is a static library (in Common Object File Format, COFF), used in static linking as shown on the page 2-1.

File `x.lib` (same name!) created by the command:

```
lib -def:x.def -out:x.lib
```

is called import library and is not a COFF file. It is used to build `p.exe` which is meant to be dynamically linked with `x.dll`. This library doesn't contain the code of `x.cpp`, it rather contains the information necessary for dynamic linking.

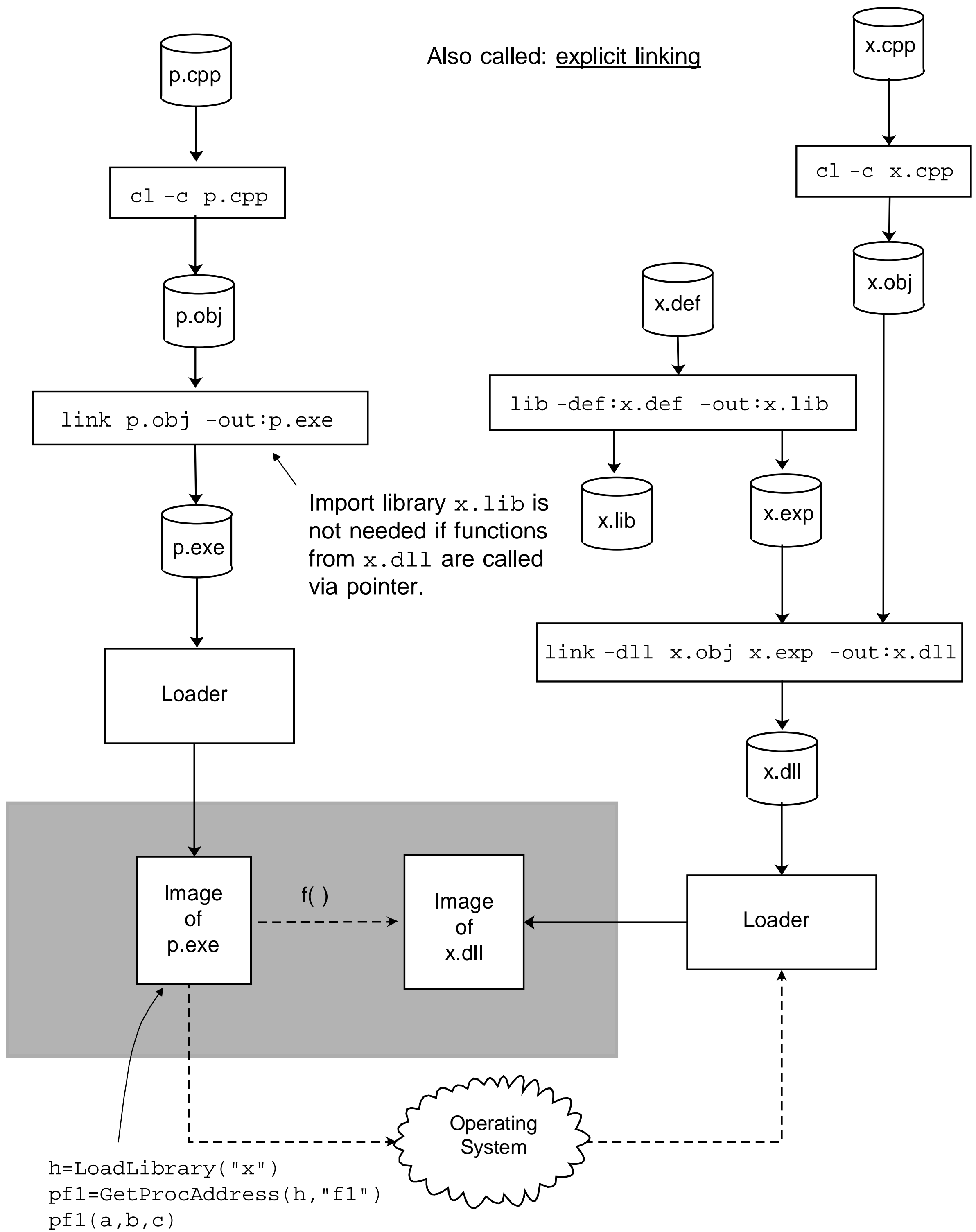
Benefits of dynamic linking:

1. Facilitates changes and upgrades (no need for relinking, sometimes the object files are not available.)

2. Code sharing (a single copy of DLL can be used by many applications, see pages 2-12, 13)

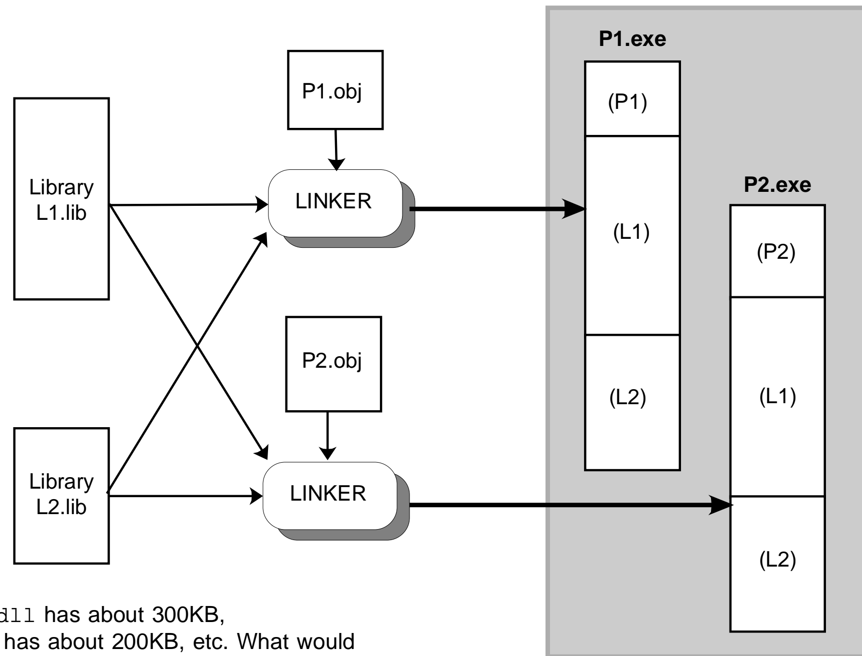
3. Extensibility (facilitates extending of the functionality of an application, or an operating system.)

Copyright © 1998-2002 by Marko Vuskovic



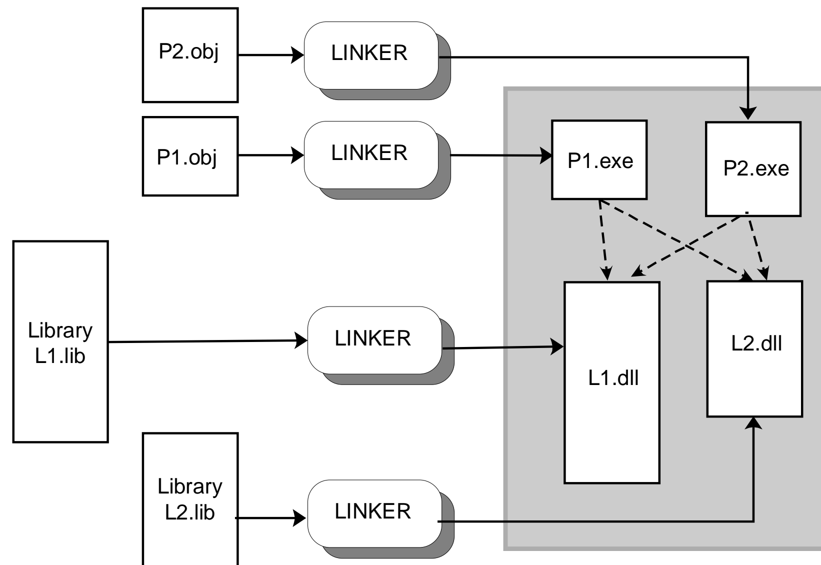
Copyright © 1998-2002 by Marko Vuskovic

NON-SHARED LIBRARIES



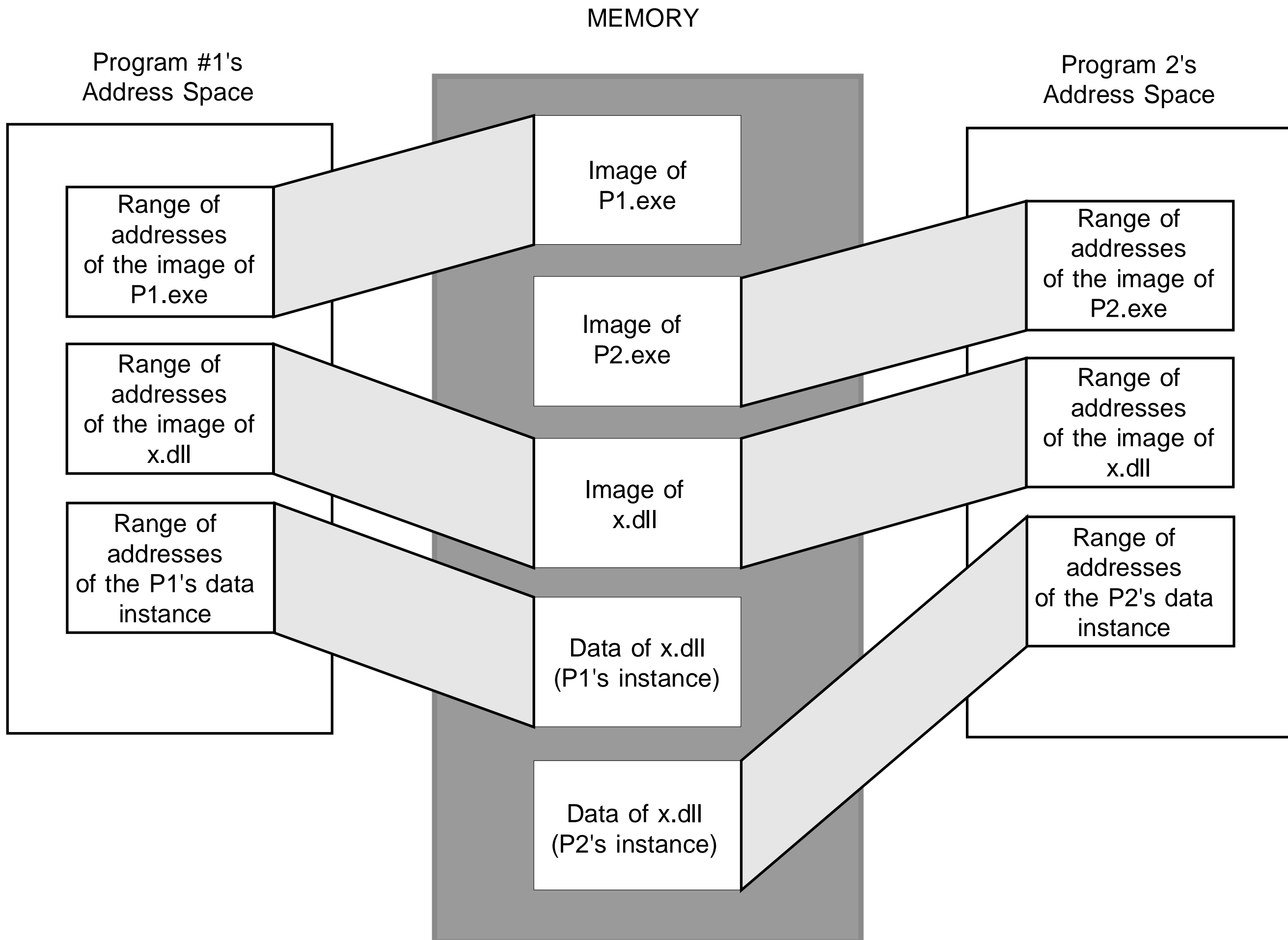
`kernel32.dll` has about 300KB, `gdi32.dll` has about 200KB, etc. What would happen if every application had a copy of these and many other system files?

SHARED LIBRARIES



Copyright © 1998-2002 by Marko Vuskovic

MEMORY MAPPING OF ADDRESS SPACES WITH DLLs



NOTICE: The terms "Program #1" and "Program #2" will be replaced by more adequate terms "Process #1" and "Process #2" in the following chapters.

All DLLs must be reentrant. Therefore each client has to have its own instance of DLL's data.

Copyright © 1998-2002 by Marko Vuskovic

AN EXAMPLE OF DYNAMIC LINKING

1. Implementation of libraries

```
/* x.cpp

  This library contains two mathematical functions written in C
  language. It can be used as statically or dynamically linked
  library. It can be called from both, C or C++.
*/

#define  LIB_VERSION   1
#include "x.h"

int version(void) {return(LIB_VERSION);};

int add(int x, int y) {return (x+y);};

int sub(int x, int y) {return (x-y);};
```

```
=================================================================
```

```
/* y.cpp

  This library contains two mathematical functions written in C
  language. It can be used as statically or dynamically linked
  library. It can be called from both, C or C++.

 */

#define  LIB_VERSION   1
#include "y.h"


int mul(int x, int y) {return (x*y);};

int divide(int x, int y) {return (x/y);};
```

Copyright © 1998-2002 by Marko Vuskovic

1. Implementation of libraries (Continued)

```
/* x.h

   This is header file for the mathematical library x.cpp.
   The prototypes are wrapped into extern "C" statement to tell
   the  C++ compiler to turn off its name mangling
 */

extern "C"
{
        int version(void);
        int add(int,int);
        int sub(int,int);
}
```

```
====================================================================
```

```
/* y.h

   This is header file for the mathematical library x.cpp.
   The prototypes are wrapped in extern "C" to tell the C++
   Compiler to turn off its name mangling
 */

extern "C"
{
        int mul(int,int);
        int divide(int,int);
}
```

Copyright © 1998-2002 by Marko Vuskovic

2. DLL Module Definition Files

```
; x.def

; This file defines all external symbols needed to create
; files x.lib and x.exp by the library manager.
; (x.exp is needed for creation of dynamic link library x.dll,
; x.lib is needed for load-time dynamic linking
; of the application p.exe with the library x.dll)

LIBRARY x

EXPORTS
 version
 add
 sub

; End of x.def




============================================================




; y.def

; This file defines all external symbols needed to create
; files y.lib and y.exp by the library manager.
; (y.exp is needed for creation of dynamic link library y.dll,
; y.lib is needed for load-time dynamic linking
; of the application p.exe with the library y.dll)

LIBRARY y

EXPORTS
 mul
 divide

; End of y.def
```

Copyright © 1998-2002 by Marko Vuskovic

3. Main Program: Dynamic Linking at Load Time

```
// p1.cpp
//
// This program does some calculations and uses
// library of mathematical functions x.cpp
// The library can be statically or dynamically linked

#include <iostream.h>
#include "x.h"     // Contains prototypes of functions add and sub
#include "y.h"      // Contains prototypes of functions mul and divide


void main()
{
        int a, b;
        cout << "Enter first number (a): ";
        cin  >> a;                  // Get the first operand
        cout << endl << "Enter second number (b): ";
        cin  >> b;                  // Get the second operand
        cout << endl << "Library version: " << version() << endl;
        cout << endl << "Here are some important results:" << endl;

        // Functions called via name:

        cout << "a + b = " << add(a,b) << endl;
        cout << "a - b = " << sub(a,b) << endl;
        cout << "a * b = " << mul(a,b) << endl;
        cout << "a / b = " << divide(a,b) << endl;
}
```

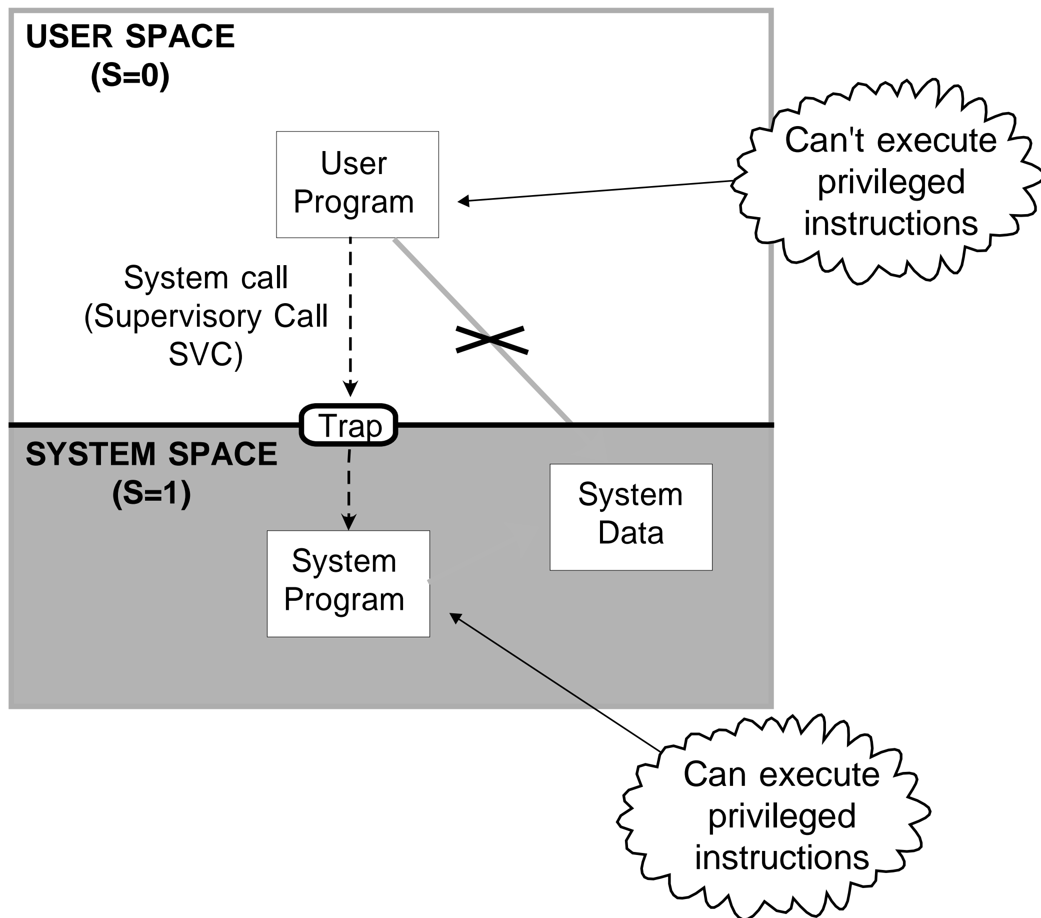
Copyright © 1998-2002 by Marko Vuskovic

4. Main Program: Dynamic Linking at Run Time

```
// p2.cpp

// This program uses some math operations (mul and divide),
// which are defined in dynamic link library y.dll.
// The program is meant to be dynamically linked with y.dll at
// run time. It calls functions from DLLs via pointers instead
// via name. This makes the program p2.cpp totally independent
// from y.lib (as opposed to p1.cpp)

#include <windows.h>
#include <stdlib.h>
#include <iostream.h>

HINSTANCE h;                          // Handle to a DLL
int (*mul)(int,int);                  // Pointer to library function
int (*divide)(int,int);               //    "           "        "
// The pointers can have the same name as functions

int main (int argc, char *argv[])
{
      int a, b;
      a = atoi(argv[1]);              // Get input numbers
      b = atoi(argv[2]);
      . . . . . . . . . . . . . . . . . . . . . . . . .
      // After some time, we need to use the functions from y.dll
      // Therefore, we have to load y.dll explicitly:

      h = LoadLibraryEx("y",NULL,0);   // System function
      if (h == NULL){
                                      // Handle the error:
            cout << "p2: Library y.dll doesn't exist."
                 << " (Error code = "<< GetLastError()<< ")" << endl;
           return(-1);                // Or do something else}
      else{             // The library is successfully loaded:
            mul = (int (*) (int, int))GetProcAddress(h, "mul");
             if (mul == NULL){          // Handle the error:
                   FreeLibrary(h);
                   cout << "p2: Couldn't get the function address"
                     << " (Error code = "<< GetLastError()<< ")"<< endl;
                 return (-1);           // Or do something else
            }
            divide = (int (*) (int, int))GetProcAddress(h, "divide");
           . . . . . . . .check success as above. . . . . . . . .

           // Functions called via pointers:
            cout << "a*b = "<< mul(a,b) << endl;
            cout << "a/b = "<< divide(a,b) << endl;
      }
}
```

Copyright © 1998-2002 by Marko Vuskovic

DIVISION OF THE ADDRESS SPACE

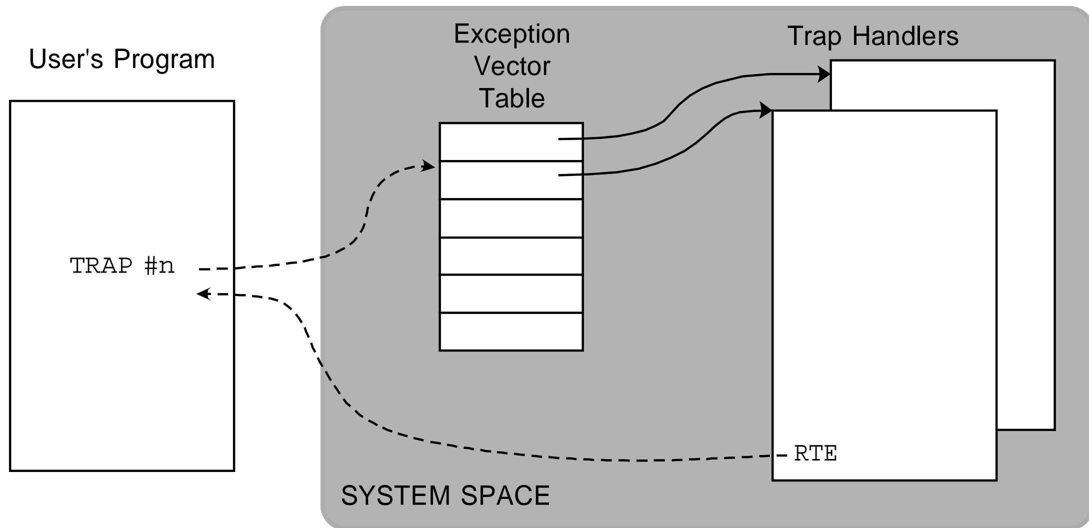


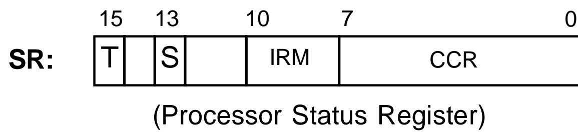
Some operating systems use even finer division.

For example VAX has four modes:

<u>Kernel</u> mode (MM, i/r, i/o)
<u>Executive</u> mode (OS service routines, file management
<u>Supervisor</u> mode (some OS service routines, user interface)
<u>User</u> mode (user programs, utilities)

Copyright © 1998-2002 by Marko Vuskovic





(Processor Status Register)

T - Trace bit
S - Status bit (S=0 -user mode,
S=1 -supervisory mode)
IRM - Interrupt mask
CCR - Condition Code Register

TRAP #n:

SSP := SSP-4; (SSP) := PC;	Push PC onto system stack
SSP:=SSP-2; (SSP): = SR;	Push SR onto system stack
Set bit "S" in SR;	Switch to supervisory mode
PC := EVT[n];	Load PC with the address of the trap handler

RTE:

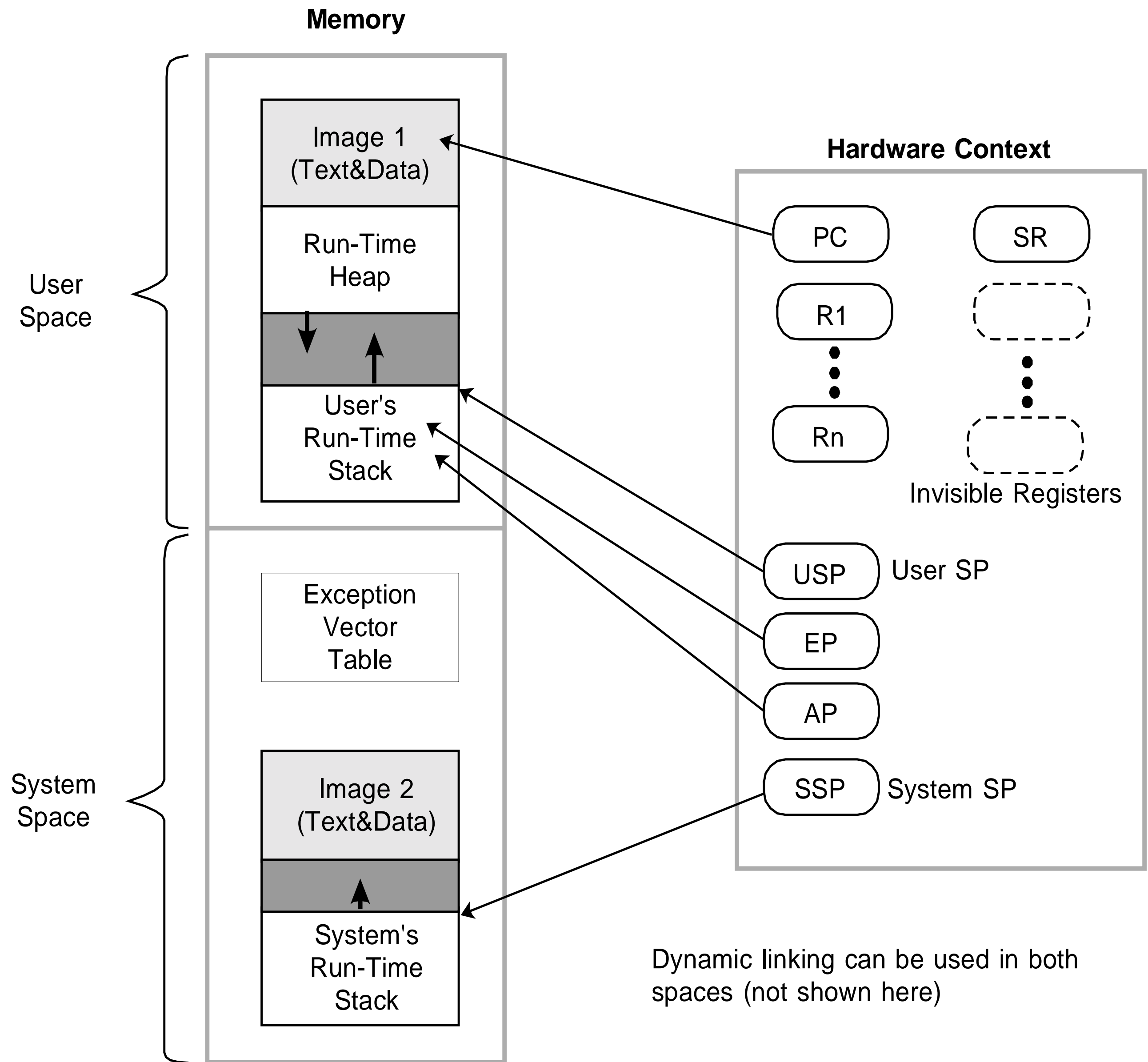
SR := (SSP); SSP := SSP+2;	Pop the SR from the system stack
PC := (SSP); SSP := SSP+4;	Pop the PC from the system stack (S-bit is not reset to 0, it is automatically set to whatever it was in SR before the TRAP call)

Privileged Instructions:

MOVE SR,<ea>	Read info from the status register
MOVE <ea>,SR	Modify status register
ANDI #N,SR	Modify Status register
EORI #N,SR	Modify Status register
ORI #N,SR	Modify Status register
MOVE USP,An	Read user stack pointer
MOVE An,USP	Modify user stack pointer
MOVEC Rc,Rn	Read the specified control register
MOVEC Rn,Rc	Modify the specified control register
MOVES Rn,<ea>	Modify a location in specified address space (user/system, program/data)
MOVES <ea>,Rn	Read a location from the specified address space
RTE	Return from exception

Copyright © 1998-2002 by Marko Vuskovic

EXECUTION ENVIRONMENT OF A PROGRAM (Revisited)

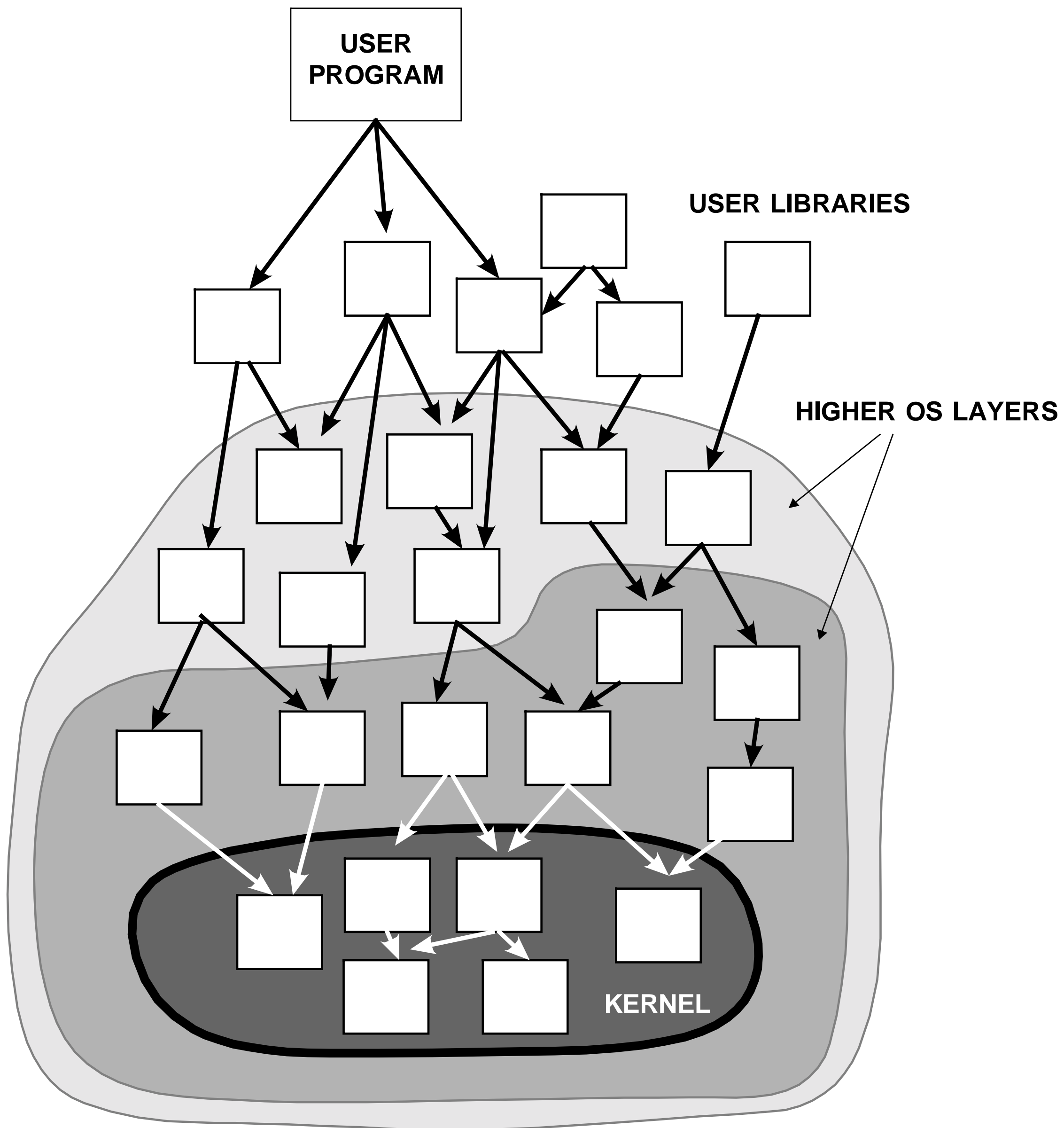


Dynamic linking can be used in both spaces (not shown here)

No linkage exists between Image 1 and Image 2

Copyright © 1998-2002 by Marko Vuskovic

LAYERED DESIGN OF AN OPERATING SYSTEM



Advantage of the layered design: changes affect limited sections of the code.

Copyright © 1998-2002 by Marko Vuskovic