TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute of Systems Architecture, Operating Systems Group

DEBUGGING

Julian Stecklina (jsteckli@os.inf.tu-dresden.de)

Dresden, 2013/1/16

00 Goals

Give you an overview about:

- debugging *in general*,
- debugging operating systems.

We will *not* discuss:

- why programs contain faults,
- how to operate your debugger of choice.

00 Outline

01 Outline

01 Debugging

to debug To *detect*, *locate*, and *correct* errors in a computer program. Techniques include use of breakpoints, desk checking, dumps, inspection, reversible execution, single-step operation, and traces. (IEEE 610.12-1990: Standard Glossary of Software Engineering)

Debuggers are useful beyond bug-finding. They can be used to:

01 Debugging

to debug To *detect*, *locate*, and *correct* errors in a computer program. Techniques include use of breakpoints, desk checking, dumps, inspection, reversible execution, single-step operation, and traces. (IEEE 610.12-1990: Standard Glossary of Software Engineering)

Debuggers are useful beyond bug-finding. They can be used to:

- *understand* a program by exploring its execution (reverse engineering),
- *manipulating* its execution,
- ...

01 Terminology

The term *bug* is imprecise, better use:

fault	A defect or abnormal condition of a component, e.g. A (missing) action in a program's code that makes it misbehave.
error	An incorrect or inconsistent state because of a fault.
failure	User-visible misbehavior of a system (due to an error).

Colloquially, the term *bug* is somewhere in between error and fault.
[1]

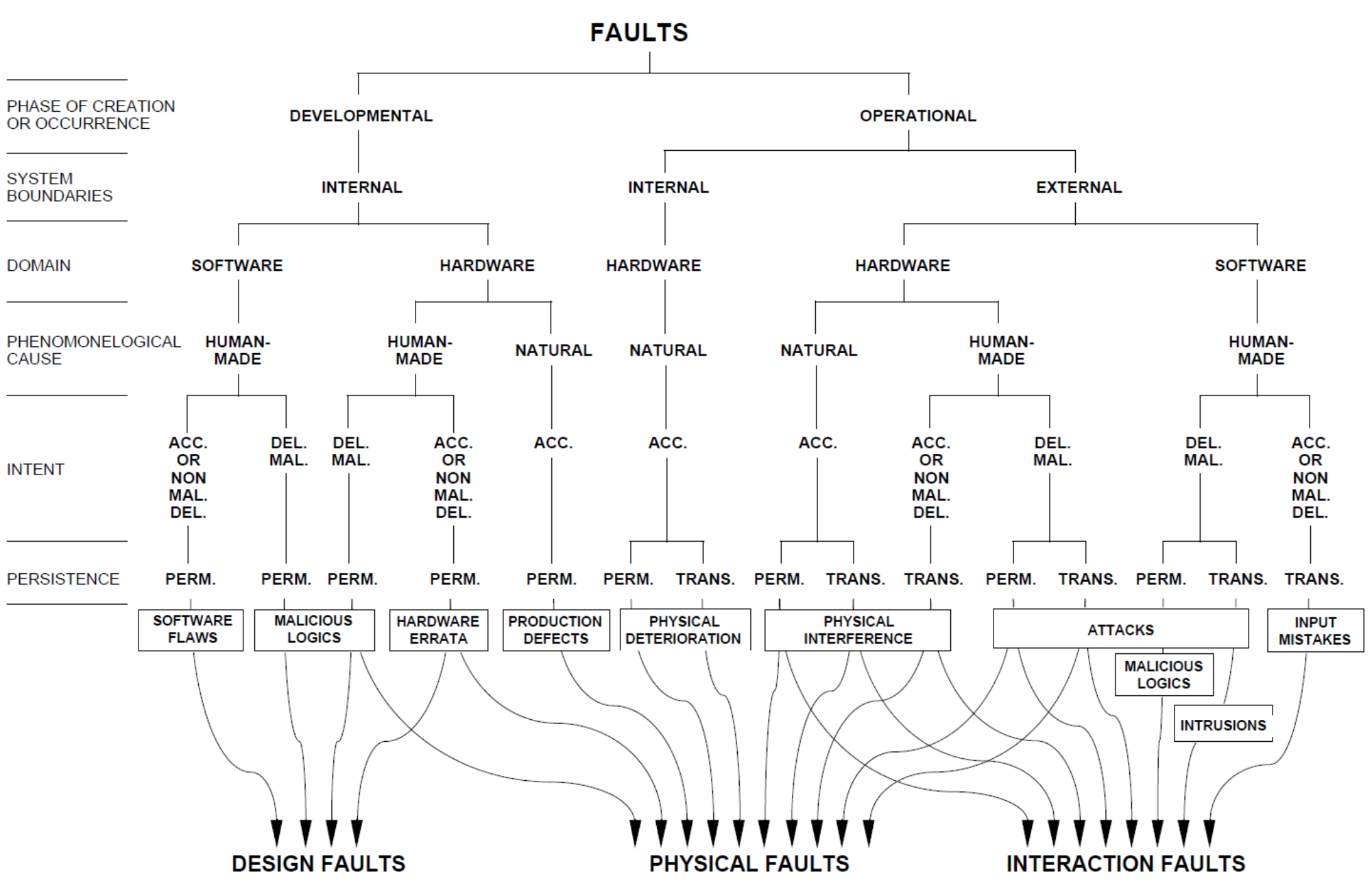

FAULTS
PHASE OF CREATION OR OCCURRENCE
SYSTEM BOUNDARIES
DOMAIN
PHENOMONELOGICAL CAUSE
INTENT
PERSISTENCE
DEVELOPMENTAL
OPERATIONAL
INTERNAL
INTERNAL
EXTERNAL
SOFTWARE
HARDWARE
HARDWARE
HARDWARE
SOFTWARE
HUMAN-MADE
HUMAN-MADE
NATURAL
NATURAL
NATURAL
HUMAN-MADE
HUMAN-MADE
ACC. OR NON MAL. DEL.
DEL. MAL.
DEL. MAL.
ACC. OR NON MAL. DEL.
ACC.
ACC.
ACC.
ACC. OR NON MAL. DEL.
DEL. MAL.
DEL. MAL.
ACC. OR NON MAL. DEL.
PERM.
PERM.
PERM.
PERM.
PERM.
PERM.
TRANS.
PERM.
TRANS.
TRANS.
PERM.
TRANS.
PERM.
TRANS.
TRANS.
SOFTWARE FLAWS
MALICIOUS LOGICS
HARDWARE ERRATA
PRODUCTION DEFECTS
PHYSICAL DETERIORATION
PHYSICAL INTERFERENCE
ATTACKS
MALICIOUS LOGICS
INTRUSIONS
INPUT MISTAKES
DESIGN FAULTS
PHYSICAL FAULTS
INTERACTION FAULTS

Image from [2]

01 Example: Ping of Death (Simplified)

The Internet Protocol (IP) specifies a way to fragment packets for narrow paths. Packets must be smaller than 2^{16} bytes. Fragments have 13 bit offset into original packet measured in 8-byte-units.

```
uint8_t  packet[65536];     // Will hold complete packet
uint13_t fragment_offset;   // from fragment IP header
uint16_t payload_length;    // from fragment IP header

if (payload_length > maximum) // usually 1492
  goto error_handling;

// Put fragment into correct place in original packet
memcpy(packet + fragment_offset*8, fragment_payload,
       payload_length);
```

01 Example: Ping of Death (Cont'd)

For fragment_offset $= 2^{13} - 1$ and payload_length > 7, the reassembly algorithm will overflow its buffer.

Windows 95 panics when it receives such a packet fragment.

Important data structures are overwritten by the last fragment.

The code doesn't check array bounds properly.

01 Example: Ping of Death (Cont'd)

For fragment_offset $= 2^{13} - 1$ and payload_length > 7, the reassembly algorithm will overflow its buffer.

Windows 95 panics when it receives such a packet fragment. *Failure*

Important data structures are overwritten by the last fragment.

The code doesn't check array bounds properly.

01 Example: Ping of Death (Cont'd)

For fragment_offset $= 2^{13} - 1$ and payload_length > 7, the reassembly algorithm will overflow its buffer.

Windows 95 panics when it receives such a packet fragment. *Failure*

Important data structures are overwritten by the last fragment. *Error*

The code doesn't check array bounds properly.

01 Example: Ping of Death (Cont'd)

For fragment_offset $= 2^{13} - 1$ and payload_length > 7, the reassembly algorithm will overflow its buffer.

Windows 95 panics when it receives such a packet fragment. *Failure*

Important data structures are overwritten by the last fragment. *Error*

The code doesn't check array bounds properly. *Fault*

Fault → Error → Failure

01 The Zune 30 Meltdown

Try to calculate the current year given the days passed since 1980:

```
int year = 1980;
while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  } else {
    days -= 365;
    year += 1;
  }
}
```

01 The Zune 30 Meltdown

Try to calculate the current year given the days passed since 1980:

```
int year = 1980;
while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  } else {
    days -= 365;
    year += 1;
  }
}
```

Failure

On December 31st, 2008 all Zune 30s drained their battery.

01 The Zune 30 Meltdown

Try to calculate the current year given the days passed since 1980:

```
int year = 1980;
while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  } else {
    days -= 365;
    year += 1;
  }
}
```

Error

The while loop does not terminate.

01 The Zune 30 Meltdown

Try to calculate the current year given the days passed since 1980:

```
int year = 1980;
while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  } else {
    days -= 365;
    year += 1;
  }
}
```

Fault

Code does not handle IsLeapYear(year) && (days == 366).

02 Outline

02 Entering the Debugger

User Interrupt

The debugger interrupts the running program.

Software Breakpoints

Debugger inserts special opcodes that cause a trap.

Exceptions

The program executed an illegal operation.

Hardware Breakpoints

The CPU can be programmed to cause a trap at certain points using special debugging registers.

```
Jan  4 04:52:24 mb-aw1n-bsd ntpd_initres[1208]: couldn't resolve '2.freebsd.pool
.ntp.org', giving up on it
Starting bsdstats.
wlan0:

Fatal trap 12: page fault while in kernel mode
cpuid = 0; apic id = 00
fault virtual address   = 0xc3aa7194
fault code              = supervisor read, page not present
instruction pointer     = 0x20:0xc09578d6
stack pointer           = 0x28:0xc2faeb74
frame pointer           = 0x28:0xc2faeb74
code segment            = base 0x0, limit 0xfffff, type 0x1b
                        = DPL 0, pres 1, def32 1, gran 1
processor eflags        = interrupt enabled, resume, IOPL = 0
current process         = 0 (iwi0 taskq)
trap number             = 12
panic: page fault
cpuid = 0
Uptime: 28s
Cannot dump. Device not defined or unavailable.
Automatic reboot in 15 seconds - press a key on the console to abort
--> Press a key on the console to reboot,
--> or switch off the system now.
```

02 What do we want to know?

Given a failure (for example a program crash because of an illegal memory read/write), we usually *get* from the hardware/operating system:

- the current register set (including the instruction pointer) and
- the application's raw, unstructured data.

We *want* to know:

02 What do we want to know?

Given a failure (for example a program crash because of an illegal memory read/write), we usually *get* from the hardware/operating system:

- the current register set (including the instruction pointer) and
- the application's raw, unstructured data.

We *want* to know:

- where we are (position in *source* file),
- how we got there (backtrace), and
- what the state of the program is (values of variables).

How does a debugger translate between those worlds?

02 Detour: Stack Layout

The stack (together with global variables) describes the current state of execution.
IA-32 (x86) ABI uses two CPU registers to manage the stack:

ESP A pointer to the top element of the stack.

EBP A pointer to the *frame* of the current function.

Frames form a linked list from the *initial function* to the *current one*!

```
int inner(int f)
{
return f;
}
```

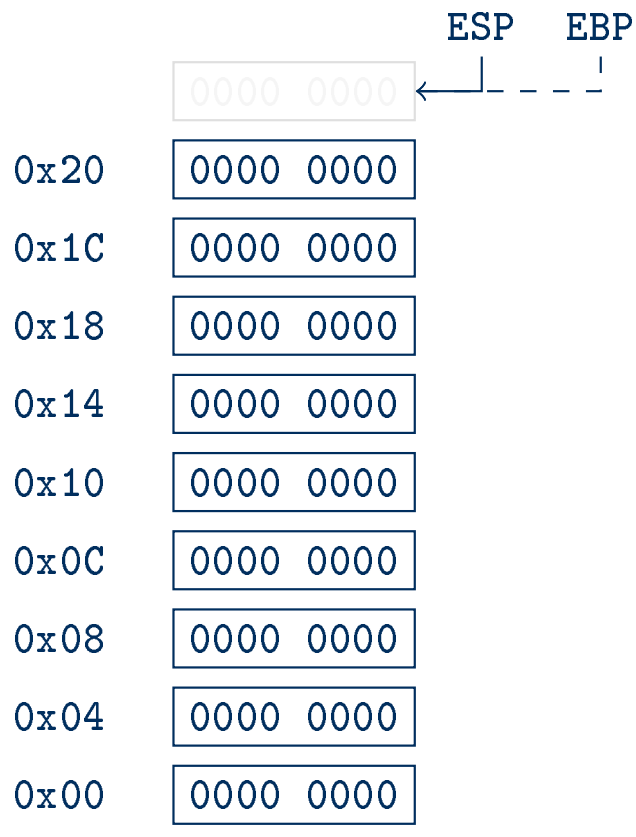
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

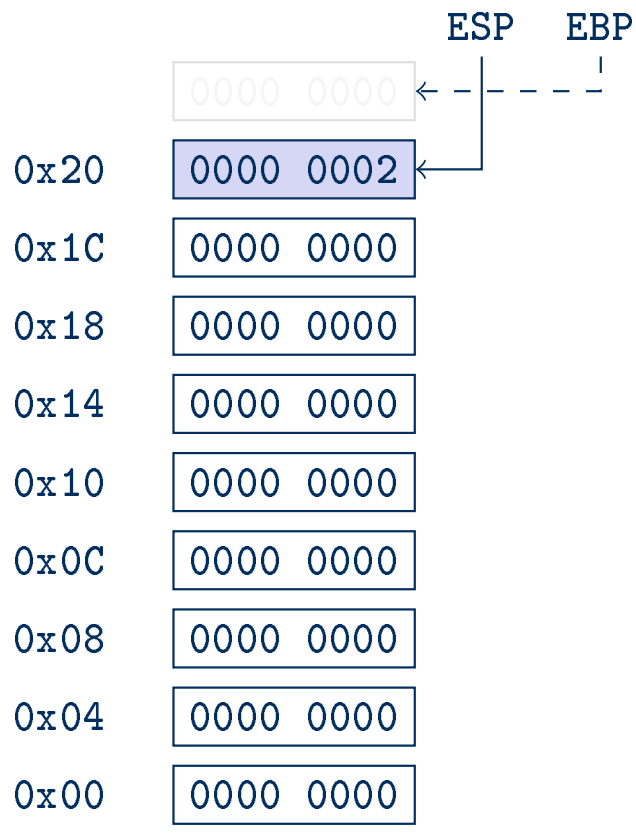
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

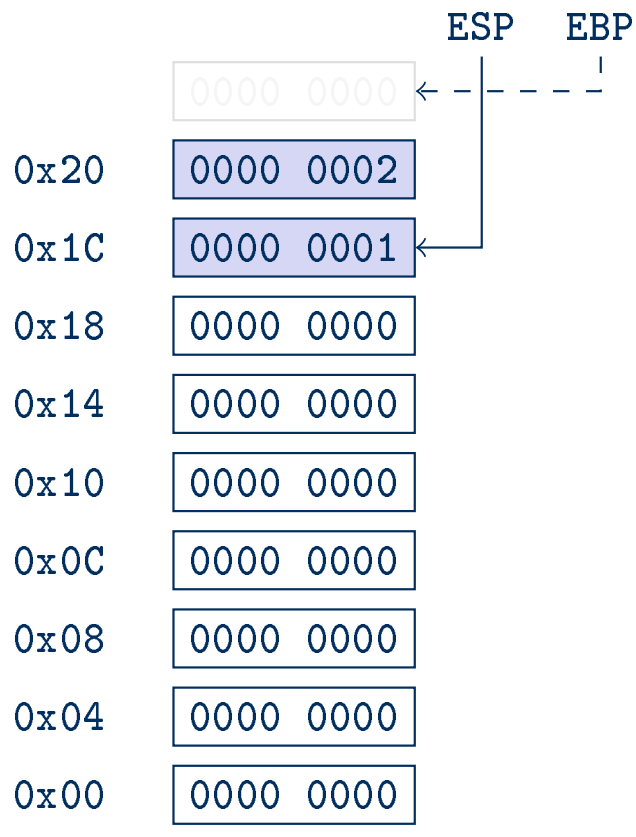
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

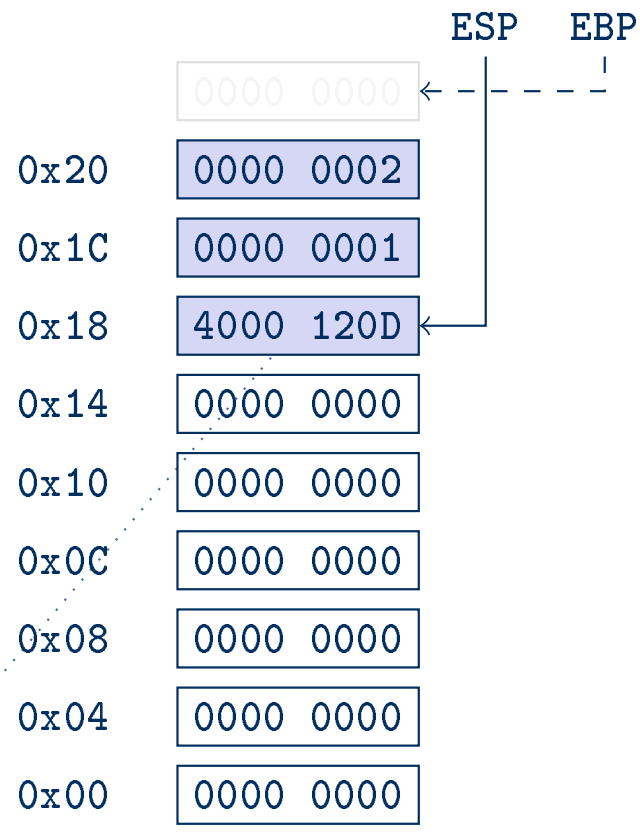
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

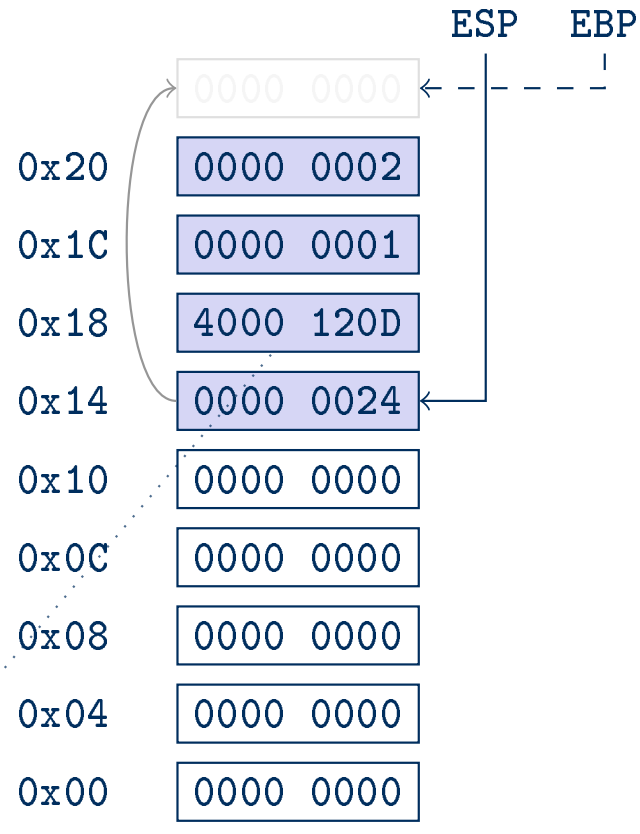
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

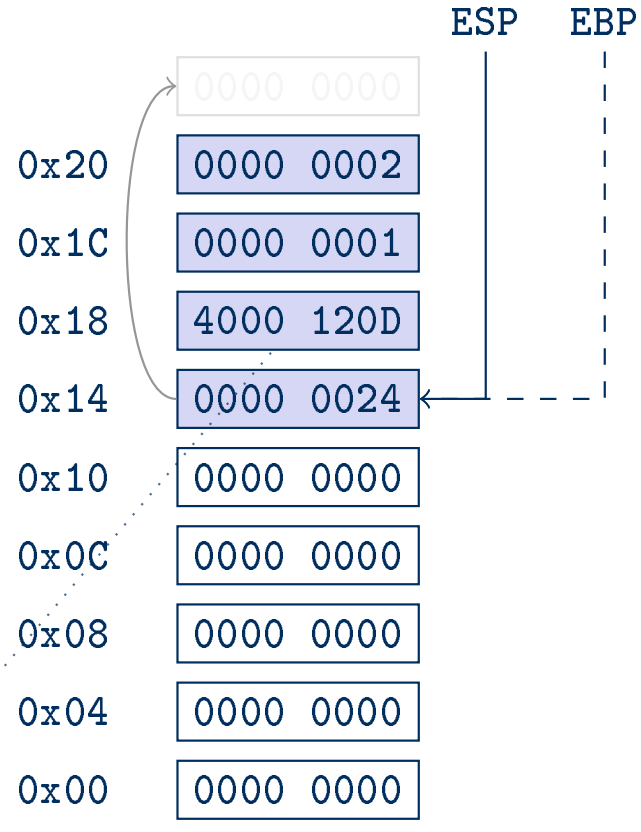
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

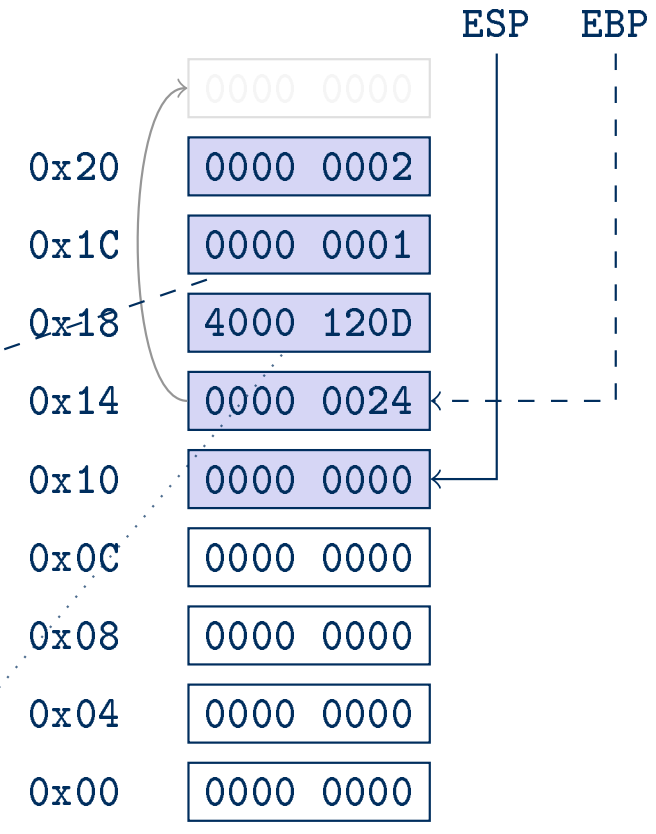
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

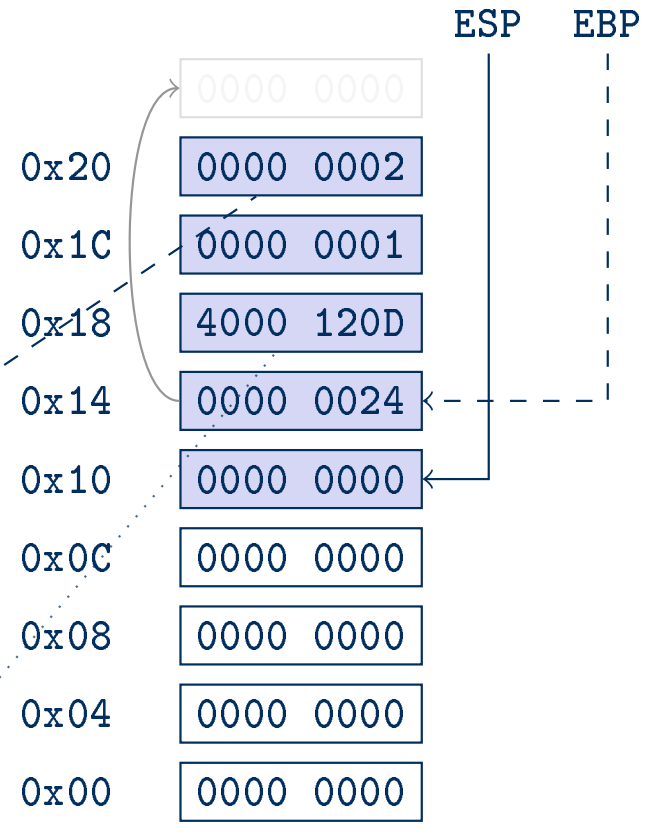
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

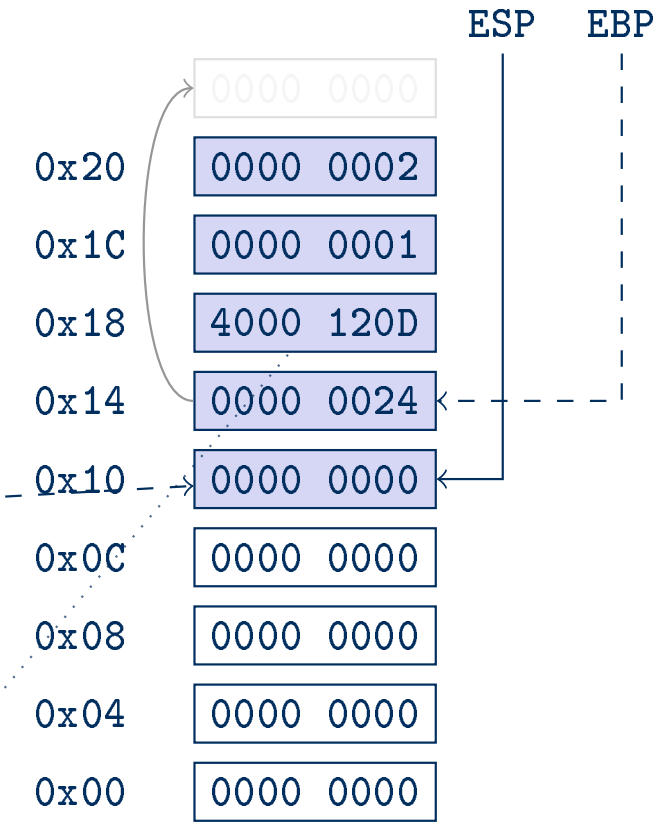
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

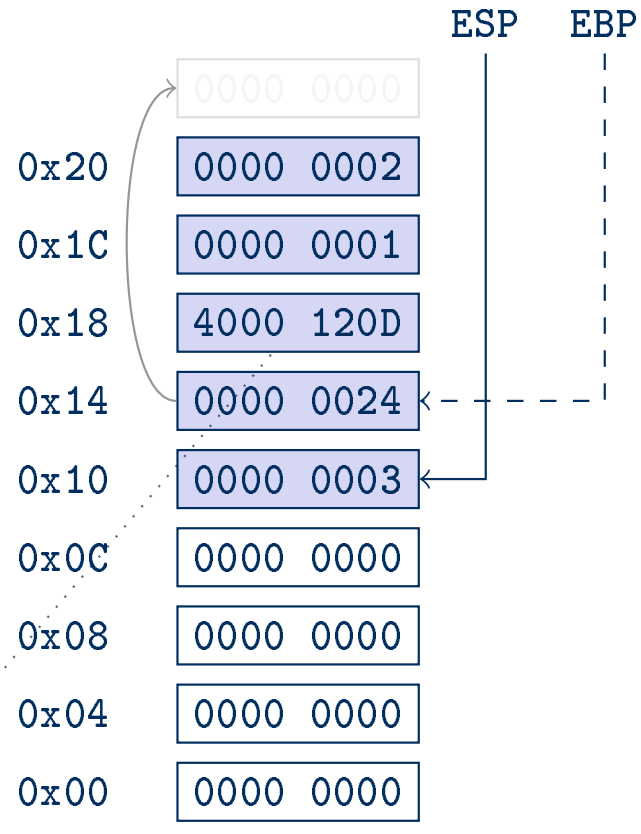
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

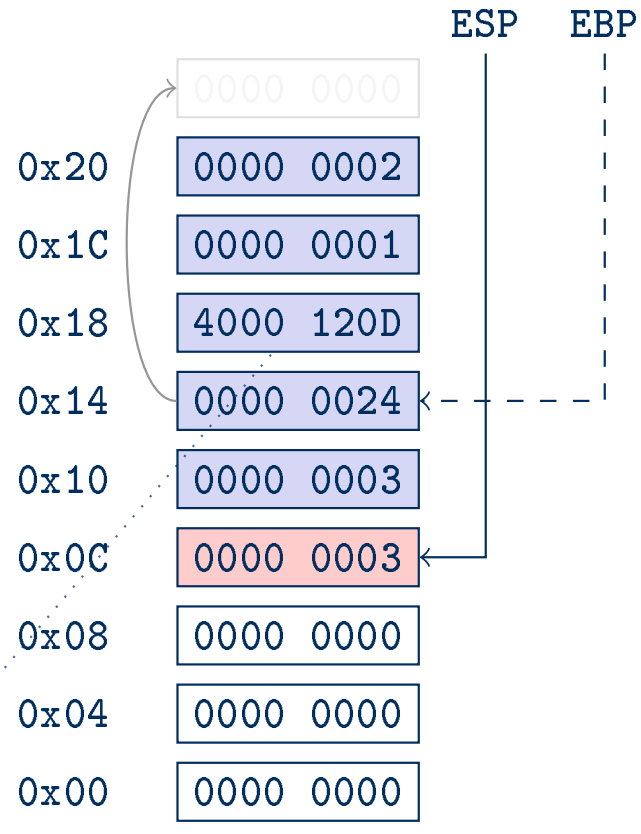
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

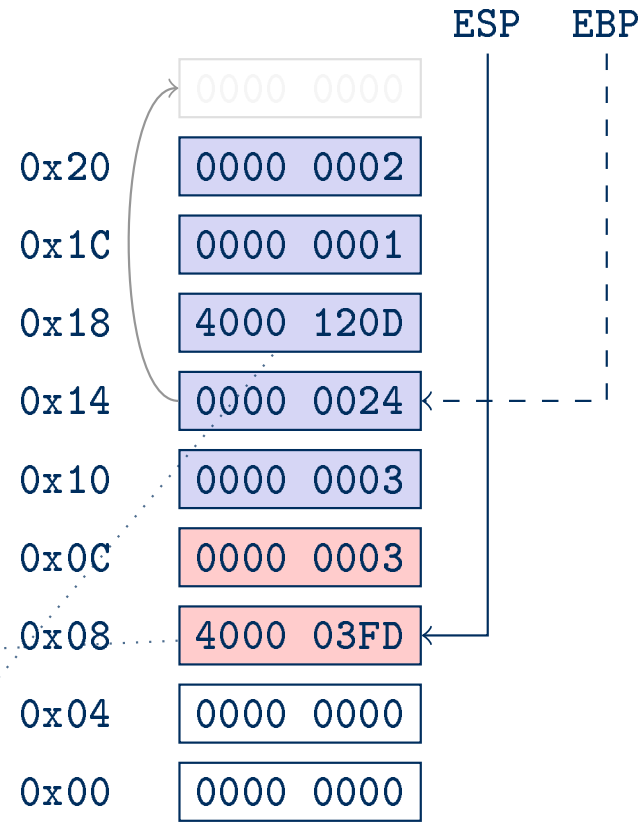
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

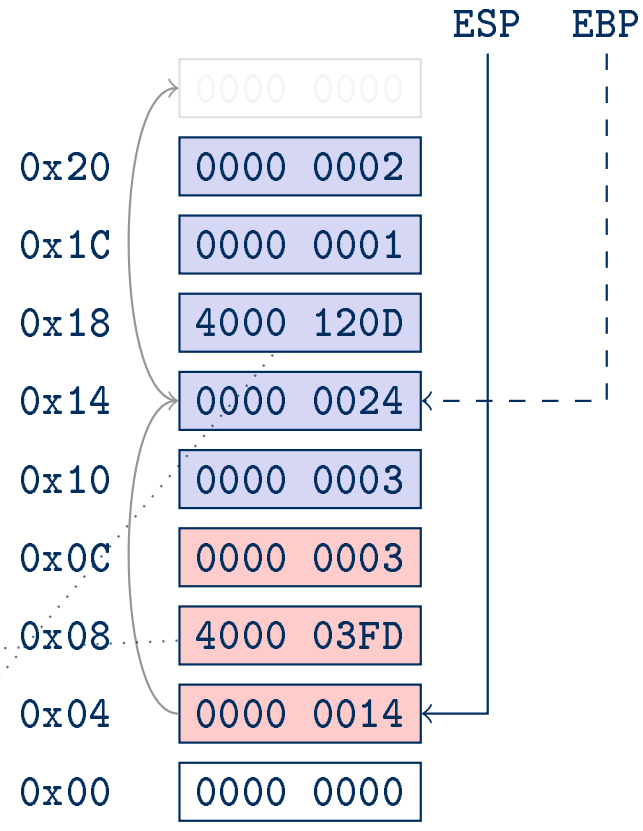
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

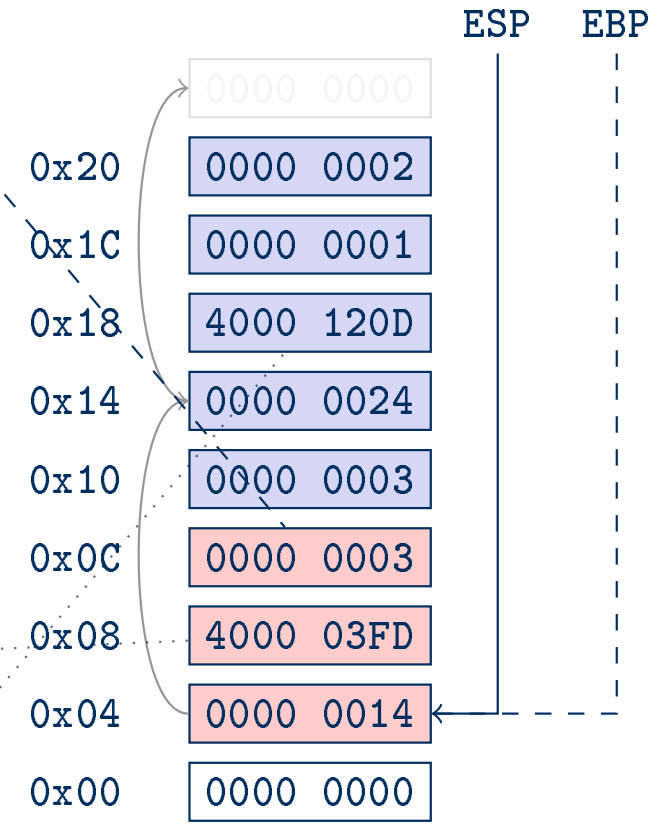
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

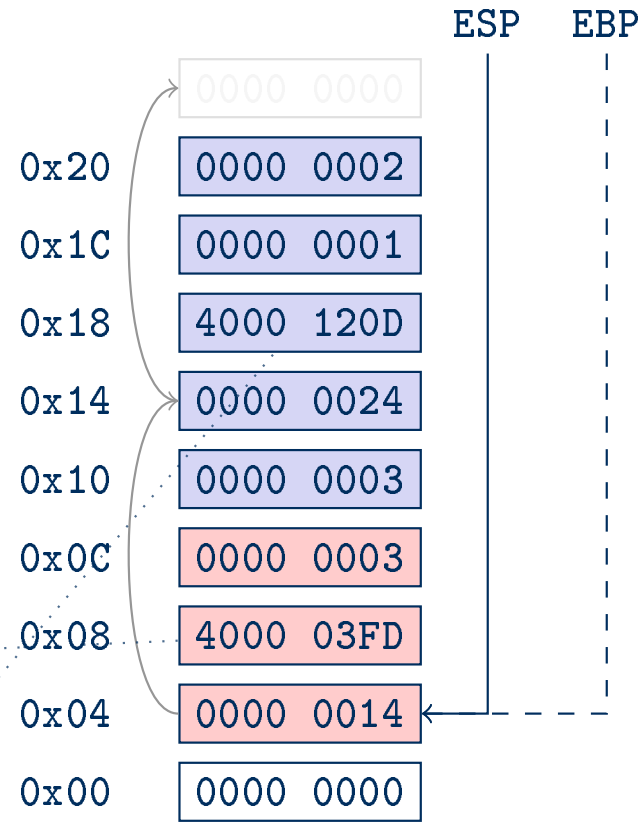
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

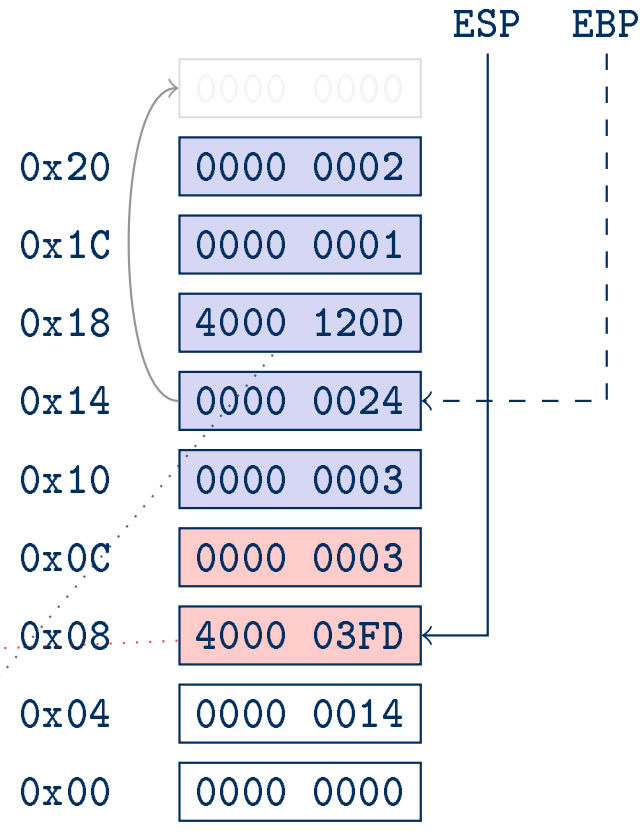
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

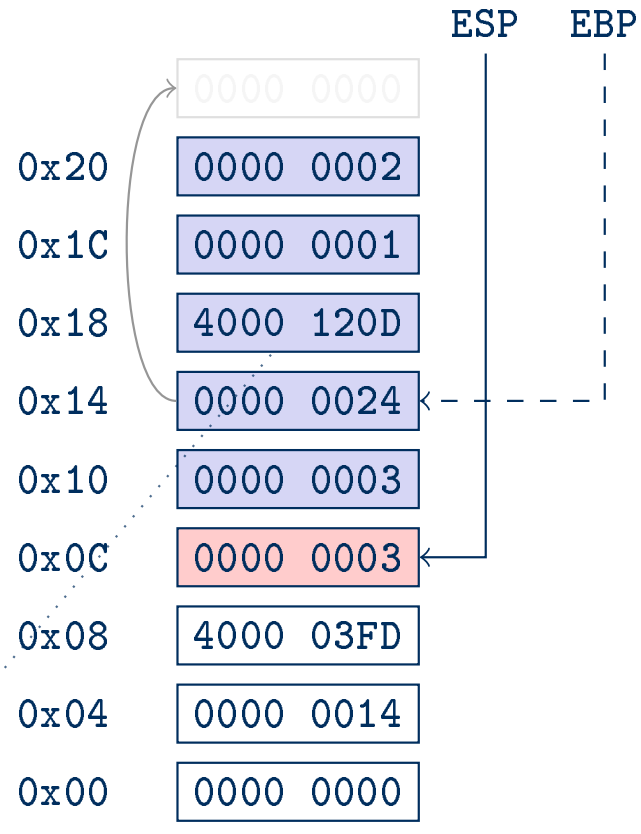
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

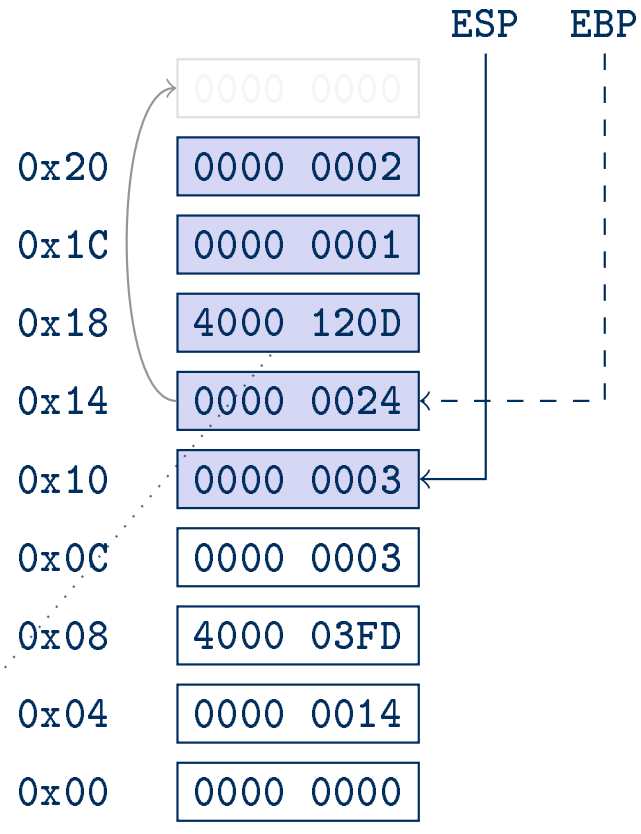
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

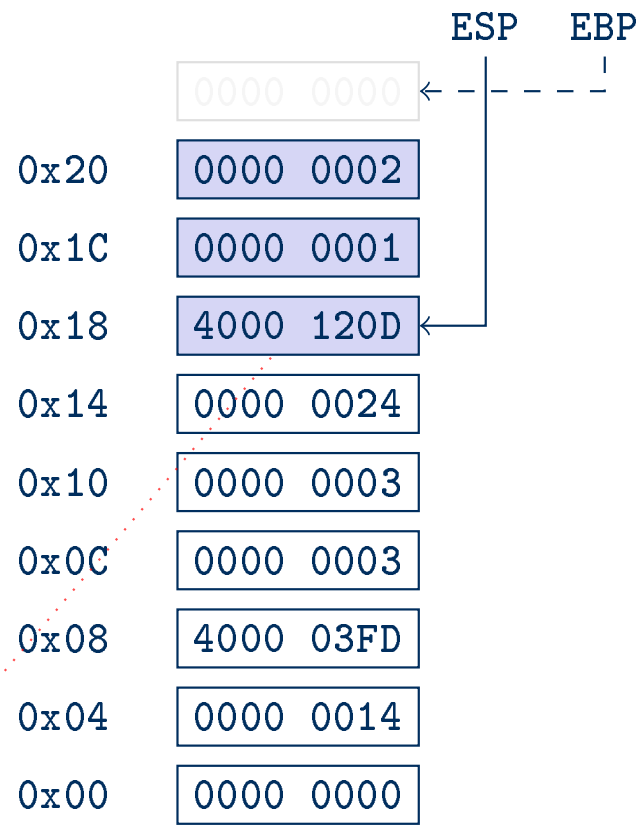
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

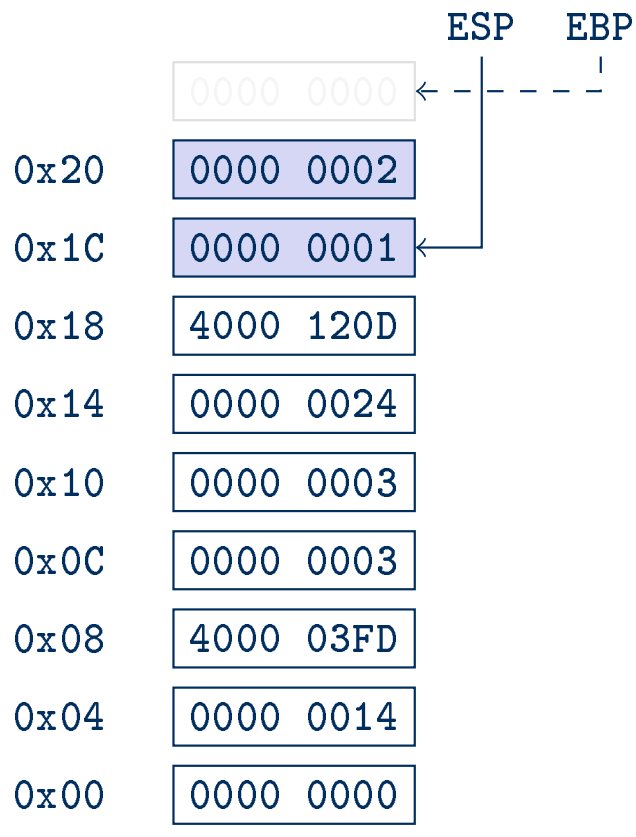
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



```
int inner(int f)
{
return f;
}
```

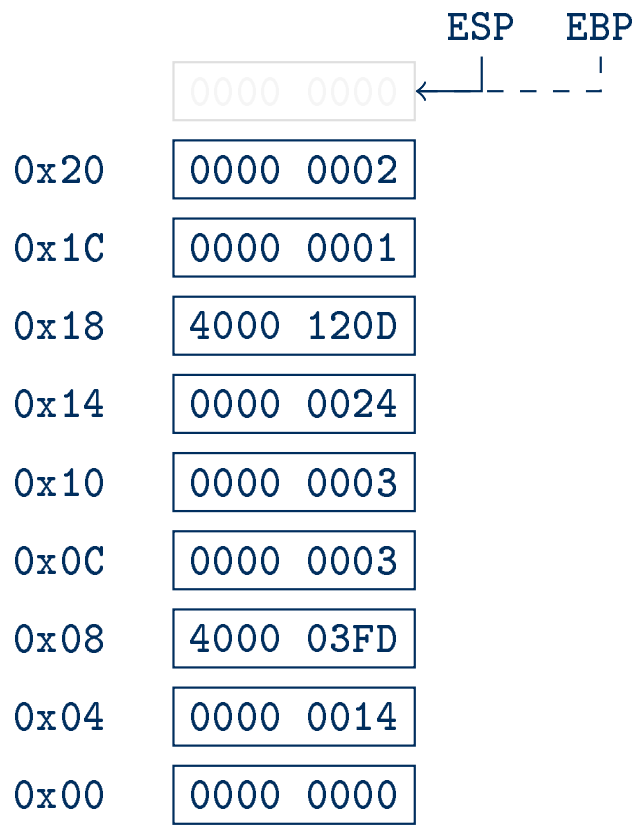
```
int outer(int a, int b)
{
int c = a + b;
return inner(c);
}
```

```
outer(1, 2);
```

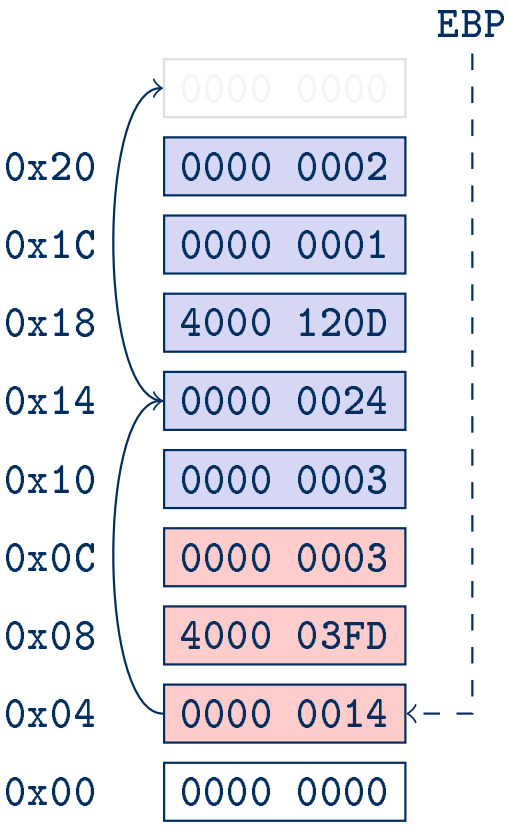
```
inner:
push %ebp
mov %esp, %ebp
mov 8(%ebp), %eax
leave
ret
```

```
outer:
push %ebp
mov %esp, %ebp
sub $4, %esp
mov 8(%ebp), %eax
add 0xC(%ebp),%eax
mov %eax, -4(%ebp)
pushd -4(%ebp)
call inner
add $4, %esp
leave
ret
```

```
push $2
push $1
call outer
add $8, %esp
```



02 Stack Layout (cont'd)



Call Frame (C/C++)

- `EBP` points to current frame.
- Arguments and Local variables are referenced relative to frame pointer.
- Next frame pointer is at `EBP`.
- Return address is `EBP+4`.

Backtrace

- Follow link of frame pointers.
- Use return address (or `EIP` for first frame) to resolve function name and arguments.

02 C Calling Convention Trivia

In the C calling convention the caller cleans up the stack adding 1 instruction per function *call*, while x86 has an instruction to do it in the `RET` instruction. Why?

Arguments are pushed right-to-left. Wouldn't left-to-right be more natural?

02 C Calling Convention Trivia

In the C calling convention the caller cleans up the stack adding 1 instruction per function *call*, while x86 has an instruction to do it in the `RET` instruction. Why?

Arguments are pushed right-to-left. Wouldn't left-to-right be more natural?

The callee does not need to know how many arguments are on the stack:

- printf
- va list, stdarg.h
- C did not require function prototypes for a long time

02 Watch out for Optimizations

`-fomit-frame-pointer` allows GCC to use `EBP` for other purposes when no frame pointer is needed.

```
int complex(int a)
{
    return a+5;
}
```

Tail Call Optimization turns calls into jumps when they are in *tail position*.

```
int simple(int a)
{
    return complex(a);
}
```

02 Optimizations vs. Debugging

Until now the debugger relied on *conventions* to

- find parameters and variables,
- unwind the stack,

but adhering to conventions obstructs compiler optimization, such as

02 Optimizations vs. Debugging

Until now the debugger relied on *conventions* to

- find parameters and variables,
- unwind the stack,

but adhering to conventions obstructs compiler optimization, such as

- keeping variables in registers,
- inlining functions,
- . . .

02 Understanding the Binary

Use debugging information. The UNIX world uses ELF and DWARF.

ELF

- *Executable and Linkable Format* (or *Extensible Linking Format*)
- the lingua franca of UNIX executable formats
- aims to be OS and CPU architecture neutral
- very versatile

DWARF [3]

- (more or less) language-independent debugging information embedded in special ELF sections
- call frame/data layout information
- line number information

02 Attaching Metainformation to Code

Naïve Solution

Store

- source file, line number,
- call frame information,
- location of variables

for every single instruction in the binary.

DWARF Solution

Store *programs* that generate

- the Call Frame Information Table,
- Line Number Table, and
- compute variable locations.

```c
#include <stdio.h>

void hello(void)
{
    puts("Hello World!\n");
}
```

objdump -d:

```
0 <hello>:
0: push %ebp
1: mov %esp,%ebp
3: sub $0x14,%esp
6: push somepointer
b: call puts
10: add $0x10, %esp
13: leave
14: ret
```

objdump -W:

```
Extended opcode  2: set Address to 0x0
Special opcode   8: Address +=  0 (0x00)
                    Line    +=  3 (4)
Special opcode  90: Address +=  6 (0x06)
                    Line    +=  1 (5)
Special opcode 188: Address += 13 (0x13)
                    Line    +=  1 (6)
Advance PC by 2 to 0x15
Extended opcode  1: End of Sequence
```

```c
#include <stdio.h>

void hello(void)
{
    puts("Hello World!\n");
}
```

objdump -d:

```
0 <hello>:
0: 55
1: 89 e5
3: 83 ec 14
6: 68 00 00 00 00
b: e8 fc ff ff ff
10: 83 c4 10
13: c9
14: c3
```

objdump -W:

```
Extended opcode  2: set Address to 0x0
Special opcode   8: Address +=  0 (0x00)
                    Line    +=  3 (4)
Special opcode  90: Address +=  6 (0x06)
                    Line    +=  1 (5)
Special opcode 188: Address += 13 (0x13)
                    Line    +=  1 (6)
Advance PC by 2 to 0x15
Extended opcode  1: End of Sequence
```

```c
#include <stdio.h>

void hello(void)
{
    puts("Hello World!\n");
}
```

objdump -d:

```
0 <hello>:
0: 55← Line 4
1: 89 e5
3: 83 ec 14
6: 68 00 00 00 00
b: e8 fc ff ff ff
10: 83 c4 10
13: c9
14: c3
```

objdump -W:

```
Extended opcode  2: set Address to 0x0
Special opcode   8: Address +=  0 (0x00)
                    Line    +=  3 (4)
Special opcode  90: Address +=  6 (0x06)
                    Line    +=  1 (5)
Special opcode 188: Address += 13 (0x13)
                    Line    +=  1 (6)
Advance PC by 2 to 0x15
Extended opcode  1: End of Sequence
```

```c
1  #include <stdio.h>
2
3  void hello(void)
4  {
5      puts("Hello World!\n");
6  }
```

objdump -d:

```
0 <hello>:
0: 55← Line 4
1: 89 e5
3: 83 ec 14
6: 68 00 00 00 00← Line 5
b: e8 fc ff ff ff
10: 83 c4 10
13: c9
14: c3
```

objdump -W:

```
Extended opcode  2: set Address to 0x0
Special opcode   8: Address +=  0 (0x00)
                    Line    +=  3 (4)
Special opcode  90: Address +=  6 (0x06)
                    Line    +=  1 (5)
Special opcode 188: Address += 13 (0x13)
                    Line    +=  1 (6)
Advance PC by 2 to 0x15
Extended opcode  1: End of Sequence
```

```c
1  #include <stdio.h>
2
3  void hello(void)
4  {
5      puts("Hello World!\n");
6  }
```

objdump -d:

```
0 <hello>:
0: 55← Line 4
1: 89 e5
3: 83 ec 14
6: 68 00 00 00 00← Line 5
b: e8 fc ff ff ff
10: 83 c4 10
13: c9← Line 6
14: c3
```

objdump -W:

```
Extended opcode  2: set Address to 0x0
Special opcode   8: Address +=  0 (0x00)
                    Line    +=  3 (4)
Special opcode  90: Address +=  6 (0x06)
                    Line    +=  1 (5)
Special opcode 188: Address += 13 (0x13)
                    Line    +=  1 (6)
Advance PC by 2 to 0x15
Extended opcode  1: End of Sequence
```

```c
#include <stdio.h>

void hello(void)
{
    puts("Hello World!\n");
}
```

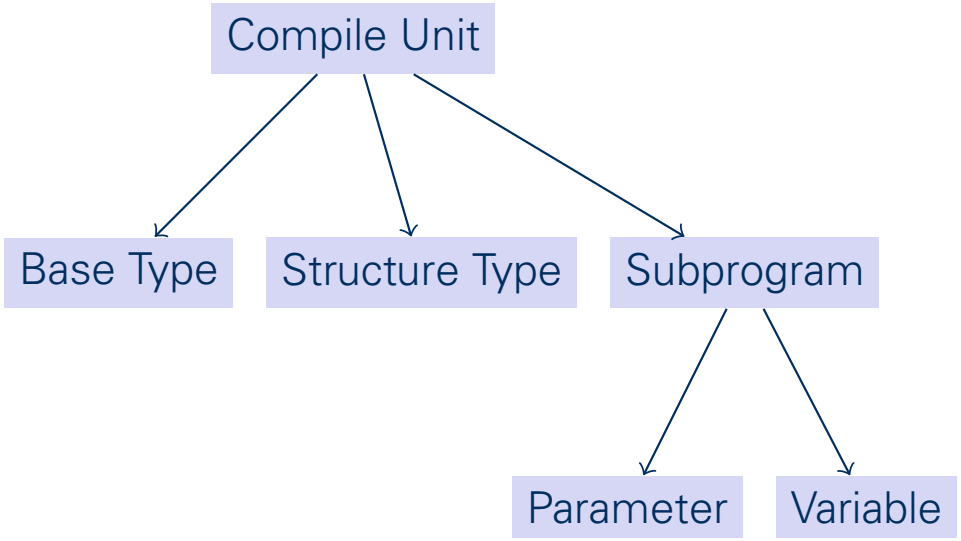
objdump -d:

```
0 <hello>:
0: 55← Line 4
1: 89 e5← Line 4
3: 83 ec 14← Line 4
6: 68 00 00 00 00← Line 5
b: e8 fc ff ff ff← Line 5
10: 83 c4 10← Line 5
13: c9← Line 6
14: c3← Line 6
```

objdump -W:

```
Extended opcode  2: set Address to 0x0
Special opcode   8: Address +=  0 (0x00)
                    Line    +=  3 (4)
Special opcode  90: Address +=  6 (0x06)
                    Line    +=  1 (5)
Special opcode 188: Address += 13 (0x13)
                    Line    +=  1 (6)
Advance PC by 2 to 0x15
Extended opcode  1: End of Sequence
```

02 Data Layout



Data is described in a graph of *records*.

Frame and structure offsets are computed by stack machine programs.

02 Data Layout (cont'd)

```
struct foo {
  int a;
  char b[10];
};
```

```
 Abbrev Number: 6 (DW_TAG_structure_type)
  DW_AT_name          : foo
  DW_AT_byte_size     : 16
[...]
 Abbrev Number: 7 (DW_TAG_member)
  DW_AT_name          : a
[...]
  DW_AT_type          : <0x4f>
  DW_AT_data_member_location: DW_OP_plus_uconst: 0
 Abbrev Number: 7 (DW_TAG_member)
  DW_AT_name          : b
[...]
  DW_AT_type          : <0xb0>
  DW_AT_data_member_location: DW_OP_plus_uconst: 4
```

02 Call Frame Information

```
Abbrev Number: 117 (DW_TAG_subprogram)
 DW_AT_name         : (indirect string, offset: 0x481d): strchr
 DW_AT_decl_file    : 1
 DW_AT_decl_line    : 136
 DW_AT_type         : <0x267bf>
 DW_AT_low_pc       : 0x4043d6
 DW_AT_high_pc      : 0x4043eb
 DW_AT_frame_base   : 0xc966        (location list)
 DW_AT_sibling      : <0x307dc>
Abbrev Number: 118 (DW_TAG_formal_parameter)
 DW_AT_name         : s
 DW_AT_decl_file    : 1
 DW_AT_decl_line    : 136
 DW_AT_type         : <0x267bf>
 DW_AT_location     : 0xc992        (location list)
```

02 Frame Pointer in Optimized Code

```
char *strchr(char *s, int c)
{
 do {
  if (c == *s)
   return s;
 } while (*s++);
 return 0;
}
```

```
4043d6: push   %ebx
4043d7: mov    (%eax),%cl
4043d9: movsbl %cl,  %ebx
4043dc: cmp    %ebx, %edx
4043de: je     4043e9
4043e0: test   %cl,  %cl
4043e2: je     4043e7
4043e4: inc    %eax
4043e5: jmp    4043d7
4043e7: xor    %eax, %eax
4043e9: pop    %ebx
4043ea: ret
```

```
0000c966 004043d6 004043d7 (DW_OP_breg4: 4)
0000c966 004043d7 004043ea (DW_OP_breg4: 8)
0000c966 004043ea 004043eb (DW_OP_breg4: 4)
```

02 Finding Variable Values in Optimized Code

DWARF includes a stack-based untyped Forth-like language to describe how to retrieve certain values from the program state.

*DW_OP_*	Stack (before)	Stack (after)
dup	*a*	*a a*
drop	*a*	
over	*a b*	*a b a*
plus_uconst:*x*	*a*	*a*+*x*
breg*n*:*x*		reg*n*+*x*
dref	*x*	value at address *x*
lt	*a b*	1 if $a < b$, 0 otherwise
bra:x	*a*	(skip x bytes of code, unless $a = 0$)
...		

02 Finding Variable Values in Optimized Code

```
unsigned
minshift(unsigned start,
         unsigned size,
         unsigned minshift)
{
 unsigned shift = bsf(start | (1u << 31));
 if (shift < minshift) minshift = shift;
 shift = bsr(size | 1);
 if (shift < minshift) return shift;
 return minshift;
}
```

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

```
401298: push %esi
401299: or $0x80000000,%eax
40129e: push %ebx
40129f: or $0x1,%edx
4012a2: bsf %eax,%ebx
4012a5: sub $0x4,%esp
4012a8: mov %edx,%eax
4012aa: mov %ecx,%esi
4012ac: call bsr
4012b1: cmp %esi,%eax
4012b3: cmovbe %eax,%esi
4012b6: mov %ebx,%eax
4012b8: cmp %ebx,%esi
4012ba: cmovbe %esi,%eax
4012bd: add $0x4,%esp
4012c0: pop %ebx
4012c1: pop %esi
4012c2: ret
```

Parameters in **EAX**, **EDX**, **ECX** on entry.

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
EBX

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
EBX

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
EBX+2^{31}

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
EBX+2^{31}

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
EBX+2^{31}
ESI

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
EBX+2^{31}
ESI

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI
EBX+2^{31}

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI
EBX+2^{31}

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI
EBX+2^{31}
ESI

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI
EBX+2^{31}
ESI

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI
EBX+2^{31}
ESI+2^{31}

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI
EBX+2^{31}
ESI+2^{31}

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI
0

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI
0

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

EBX
ESI

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

ESI
EBX

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

ESI
EBX

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

ESI

```
        (parameter minshift)
0000d09b 004012ac 004012b0
DW_OP_breg1: 0;
DW_OP_dup;
DW_OP_plus_uconst: 2147483648;
DW_OP_breg3: 0;
DW_OP_swap;
DW_OP_over;
DW_OP_plus_uconst: 2147483648;
DW_OP_lt;
DW_OP_bra: 1;
DW_OP_swap;
DW_OP_drop;
DW_OP_stack_value
```

ESI

02 Short Summary

We have seen how to:

- generate backtraces,
- locate corresponding source code,
- find values in optimized code, and
- interpret data.

But it was x86 only. What about ARM, AMD64, . . . ?

- DWARF is cross-platform, cross-architecture,
- principles apply to other architectures as well.

02 DWARF

. . . is intended to allow debugging even in highly optimized code, using

- a set of densely coded tables (Source code, CFI) and
- byte-code programs (compute values/locations).

Of course, there are problems:

02 DWARF

. . . is intended to allow debugging even in highly optimized code, using

- a set of densely coded tables (Source code, CFI) and
- byte-code programs (compute values/locations).

Of course, there are problems:

- complex (Turing-complete?)
- hard to implement on both sides (compiler/debugger)

03 Outline

03 Debugger: Where shall you go?

Application debuggers are usually applications themselves, with some general support from the kernel. But where to put the debugger, if you want to debug the kernel itself?

- as application?
- as part of the kernel?
- as application on a remote system?
- integrated into a Virtual Machine Monitor?

03 Debugger: Where shall you go?

Application debuggers are usually applications themselves, with some general support from the kernel. But where to put the debugger, if you want to debug the kernel itself?

- ~~as application?~~
- as part of the kernel?
- as application on a remote system?
- integrated into a Virtual Machine Monitor?

03 Example: FreeBSD's DDB

. . . is a complete debugger inside the FreeBSD kernel with support for:

- online debugging:
  - breakpoints
  - stacktraces
  - examining/manipulating kernel data
- deadlock/lock-order-reversal detection
- writing the OS state to disk (*kernel dump*)
- . . .

03 Example: FreeBSD's DDB (cont'd)

Input via USB

- USB keyboards are increasingly common
- new USB stack in FreeBSD 8 did not support polling mode at first
- USB keyboards unusable in DDB

(Fixed before FreeBSD 8.0)

Kernel Dump

- needs block device subsystem to work (IDE, USB, . . .)
- needs (part of) swap subsystem to work

03 Example: Fiasco's JDB

. . . is a fully-fledged debugger implemented in the microkernel. JDB is about one third of Fiasco's source code.

03 Example: Fiasco's JDB

. . . is a fully-fledged debugger implemented in the microkernel. JDB is about one third of Fiasco's source code.

> *A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e., permitting competing implementations, would prevent the implementation of the system's required functionality.*
>
> Jochen Liedtke

03 As Part of the Kernel?

The debugger is a subsystem in the kernel.

Pro

- highly integrated
- specialized tools possible

Contra

- subsystems used by the debugger are out of scope
- potentially very large
- blurry distinction between kernel and debugger
- *micro*kernel?

03 As application on a remote system?

The debugger is an application on another host. The kernel contains low-level debugging and communication infrastructure in a small *stub*.

Pro

- reuse existing application debuggers (GDB)
- GUI frontends
- less development effort

Contra

- application debuggers ill-suited
- bad integration into OS
- communication?

03 Communicating with a Remote Stub

RS-232

- not available out-of-the-box anymore
- serial line driven by two known I/O ports
- trivial to send data from anywhere in your code
- slow (typically max 115200 baud)

EHCI Debug Port

- not always supported by EHCI controller
- RS-232 replacement for *legacy-free* systems
- no complete USB stack needed, but PCI...
- need special USB *Debug Device* (90$)
- speed?
- never seen in the wild...

03 As application using a core dump?

The debugger is an application on another host. The kernel is able to *dump* all its state to permanent storage.

Pro

- reuse existing application debuggers (GDB)
- GUI frontends
- less development effort
- no need to communicate

Contra

- application debuggers ill-suited
- OS needs to be able to write to permanent storage
- debugged system is dead

03 Integrated into a VMM?

Special case of remote debugging. The VMM implements stub functionality directly.

03 Integrated into a VMM?

Special case of remote debugging. The VMM implements stub functionality directly.

Pro

- able to debug *everything*
- no test box needed
- no OS-specific stub needed

Contra

- limited to a small set of *virtual* hardware
- even less integration into OS

03 Detour: A Small Stub?

The typical debugging stub (about 1000–2000 LOC) for GDB contains:

- communication over a serial line,
- the increasingly complex GDB protocol implementation

in order to:

03 Detour: A Small Stub?

The typical debugging stub (about 1000–2000 LOC) for GDB contains:

- communication over a serial line,
- the increasingly complex GDB protocol implementation

in order to:

- access and manipulate memory and CPU state (registers),
- control execution.

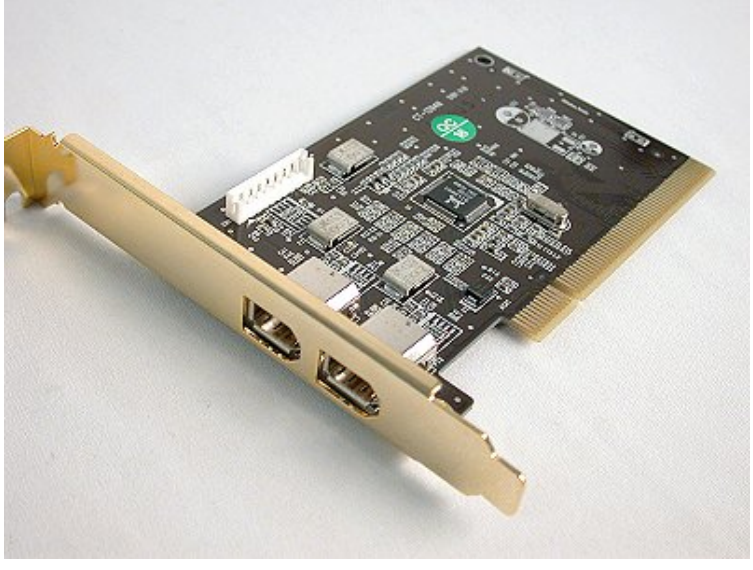
03 Reducing Stub Size . . .

With off-the-shelf hardware (Firewire) we have *direct remote* access to:

- physical memory (Remote DMA) and
- basic execution control by sending interrupts

from a *remote computer*. We could also use a custom:

- Thunderbolt,
- PCI/PCMCIA or PCIe/ExpressCard device.



03 Reducing Stub Size . . . (cont'd)

The stub still needs to:

03 Reducing Stub Size . . . (cont'd)

The stub still needs to:

- handle these interrupts,
- expose CPU state.

The remote host takes over:

- resolving physical to virtual addresses,
- setting breakpoints.

03 . . . to Zero.

If we can read memory without any help, why not *inject the needed stub at runtime*?

Pro

- Stub can be developed as separate project.
- One stub for multiple target kernels.

Contra

- slightly fragile
- ?

There is an experimental implementation for the NOVA hypervisor.

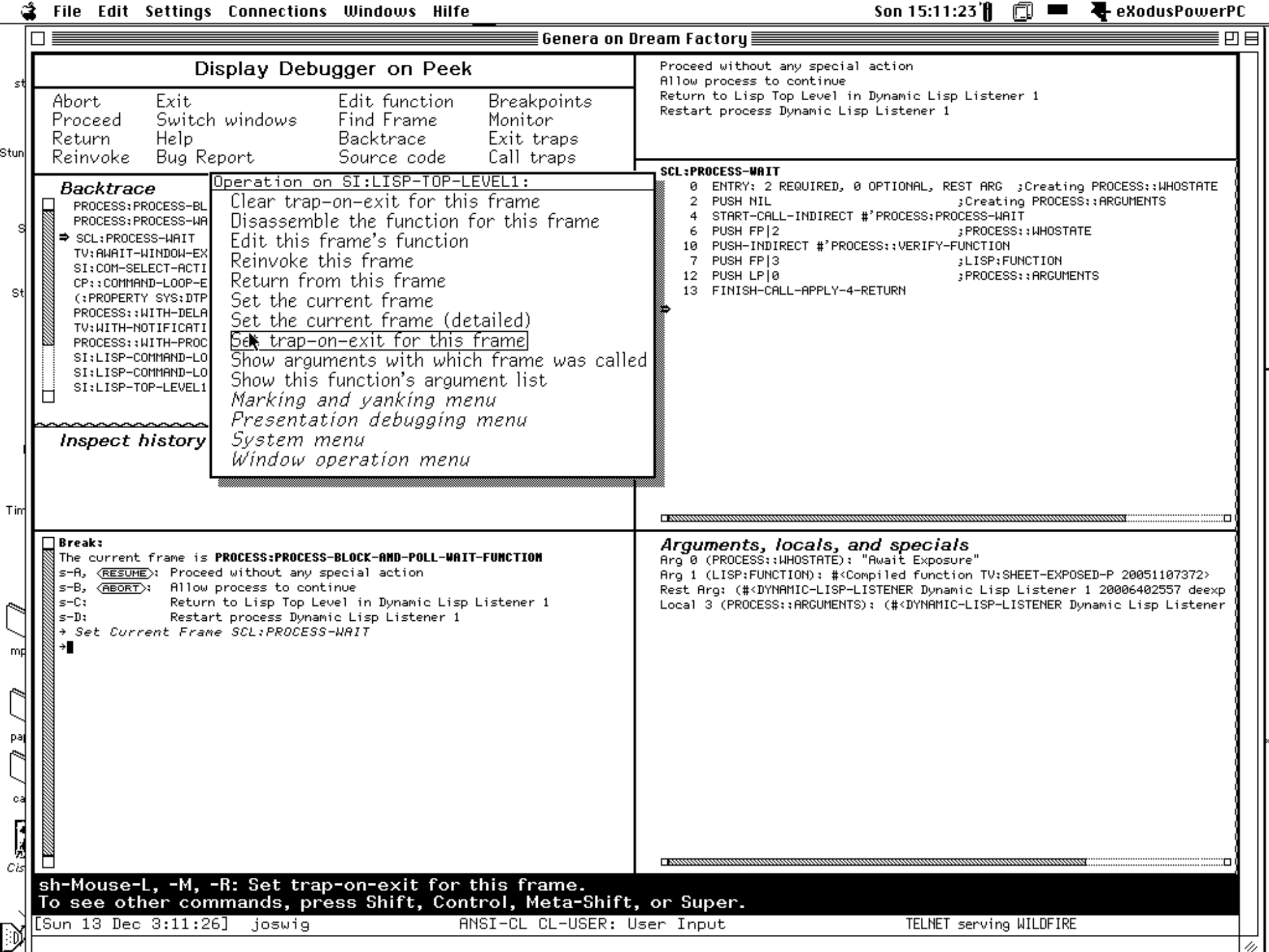
04 Outline

04 The Glorious Past

Genera is . . .

- built in the early 80s,
- a single-user OS
- a single-address-space OS without a clear kernel/application distinction,
- a language runtime

written almost completely in ZetaLisp and some assembler.


File Edit Settings Connections Windows Hilfe
Son 15:11:23
eXodusPowerPC
Genera on Dream Factory
Display Debugger on Peek
Abort Exit Edit function Breakpoints
Proceed Switch windows Find Frame Monitor
Return Help Backtrace Exit traps
Reinvoke Bug Report Source code Call traps
Proceed without any special action
Allow process to continue
Return to Lisp Top Level in Dynamic Lisp Listener 1
Restart process Dynamic Lisp Listener 1
Backtrace
PROCESS:PROCESS-BL
PROCESS:PROCESS-WA
⇒ SCL:PROCESS-WAIT
TV:AWAIT-WINDOW-EX
SI:COM-SELECT-ACTI
CP::COMMAND-LOOP-E
(:PROPERTY SYS:DTP
PROCESS::WITH-DELA
TV:WITH-NOTIFICATI
PROCESS::WITH-PROC
SI:LISP-COMMAND-LO
SI:LISP-COMMAND-LO
SI:LISP-TOP-LEVEL1
Inspect history
Operation on SI:LISP-TOP-LEVEL1:
Clear trap-on-exit for this frame
Disassemble the function for this frame
Edit this frame's function
Reinvoke this frame
Return from this frame
Set the current frame
Set the current frame (detailed)
Set trap-on-exit for this frame
Show arguments with which frame was called
Show this function's argument list
Marking and yanking menu
Presentation debugging menu
System menu
Window operation menu
SCL:PROCESS-WAIT
0 ENTRY: 2 REQUIRED, 0 OPTIONAL, REST ARG ;Creating PROCESS::WHOSTATE
2 PUSH NIL ;Creating PROCESS::ARGUMENTS
4 START-CALL-INDIRECT #'PROCESS:PROCESS-WAIT
6 PUSH FP|2 ;PROCESS::WHOSTATE
10 PUSH-INDIRECT #'PROCESS::VERIFY-FUNCTION
7 PUSH FP|3 ;LISP:FUNCTION
12 PUSH LP|0 ;PROCESS::ARGUMENTS
13 FINISH-CALL-APPLY-4-RETURN
Break:
The current frame is PROCESS:PROCESS-BLOCK-AND-POLL-WAIT-FUNCTION
s-A, (RESUME): Proceed without any special action
s-B, (ABORT): Allow process to continue
s-C: Return to Lisp Top Level in Dynamic Lisp Listener 1
s-D: Restart process Dynamic Lisp Listener 1
→ Set Current Frame SCL:PROCESS-WAIT
→
Arguments, locals, and specials
Arg 0 (PROCESS::WHOSTATE): "Await Exposure"
Arg 1 (LISP:FUNCTION): #<Compiled function TV:SHEET-EXPOSED-P 20051107372>
Rest Arg: (#<DYNAMIC-LISP-LISTENER Dynamic Lisp Listener 1 20006402557 deexp
Local 3 (PROCESS::ARGUMENTS): (#<DYNAMIC-LISP-LISTENER Dynamic Lisp Listener
sh-Mouse-L, -M, -R: Set trap-on-exit for this frame.
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Sun 13 Dec 3:11:26] joswig ANSI-CL CL-USER: User Input TELNET serving WILDFIRE

Top of History

(#<HTTP:COMMON-FILE-LOG Common-File-Log-80 (port: 80) 1071466350>)
#<PROCESS:PROCESS HTTP-Common-File-Log-80-Daemon (2 0) 1071456435>
(#<STANDARD-GENERIC-FUNCTION TQ::TASK-QUEUE-MAIN-LOOP 21013003534> #<HTTP:COMMON-FILE-LOG Common-File-Log-80 (port: 80) 1071466350>)
#<STANDARD-GENERIC-FUNCTION TQ::TASK-QUEUE-MAIN-LOOP 21013003534>
#<HTTP:COMMON-FILE-LOG Common-File-Log-80 (port: 80) 1071466350>

Bottom of History

Exit
Return
Modify
DeCache
Clear
Set \
Source

Top of object

a list
(#<STANDARD-GENERIC-FUNCTION TQ::TASK-QUEUE-MAIN-LOOP 21013003534> #<HTTP:COMMON-FILE-LOG Common-File-Log-80 (port: 80) 1071466350>)

Bottom of object

Top of object

#<STANDARD-GENERIC-FUNCTION TQ::TASK-QUEUE-MAIN-LOOP 21013003534>

Bottom of object

Top of object

#<HTTP:COMMON-FILE-LOG Common-File-Log-80 (port: 80) 1071466350>
An instance of HTTP:COMMON-FILE-LOG. #<Message handler for HTTP:COMMON-FILE-LOG>

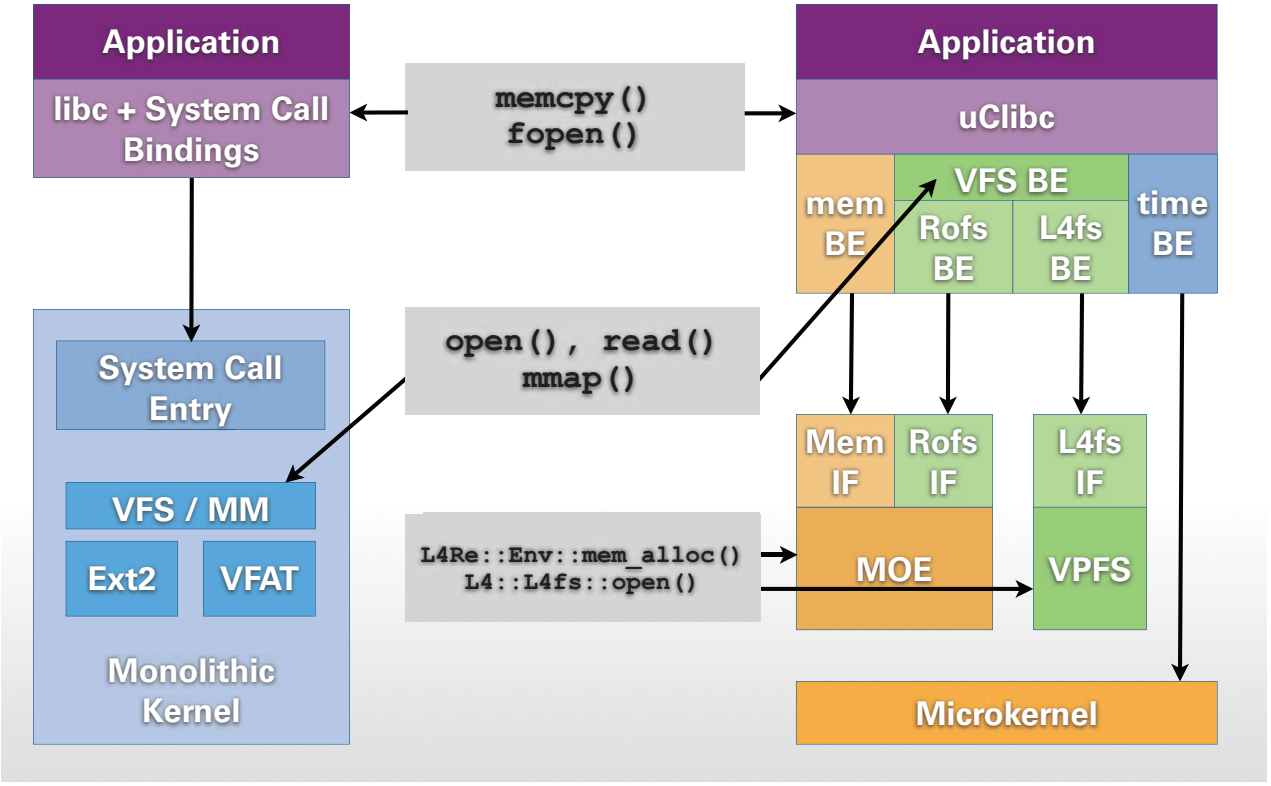
HTTP::LOCK:	#<PROCESS::SIMPLE-NONRECURSIVE-NORMAL-LOCK HTTP Log Lock 1071507201>
HTTP::LOG-FILE-NAME:	"Common-Log"
HTTP::FILE-LOGGING:	T
FILE-STREAM:	NIL
HTTP::FILENAME:	#P"DF:>CL-HTTP>log>Common-Log-80.text"
HTTP::LOG-TIMES-IN-GMT-P:	T
TQ::QUEUE:	NIL
TQ::POINTER:	NIL
TQ::N-PENDING-TASKS:	0
TQ::RUN-P:	T
TQ::LOCK:	#<PROCESS::SIMPLE-NONRECURSIVE-NORMAL-LOCK Task Queue 1071507210>
TQ::PROCESS-PRIORITY:	0
TQ::PROCESS:	#<PROCESS:PROCESS HTTP-Common-File-Log-80-Daemon (2 0) 1071456435>
TQ::PROCESS-NAME:	"HTTP-Common-File-Log-80-Daemon"
TQ::WAIT-WHOSTATE:	"Log Wait"
HTTP::NAME:	"Common-File-Log-80"
URL:PORT:	80
HTTP::CREATION-TIME:	3122046510
HTTP::NOTIFICATION:	NIL
HTTP::N-ACCESS-DENIALS:	109

More below

Choose a value by pointing at the value. Right finds function definition.

[Sun 13 Dec 3:10:03] joswig ANSI-CL CL-USER: User Input TELNET serving WILDFIRE

04 Problems for a Multi-Server OS



04 Problems for a Multi-Server OS

While Genera is amazing, its approach does not apply to modern OSes. Writing a systems debugger for a microkernel OS is an open field of research.
Application state is potentially scattered among several servers.

The monolithic kernel sees *processes, pipes, sockets, files,* ....

The microkernel sees *tasks, memory mappings, IPC gates,* ....

A *kernel* debugger is not enough!

04 Making it Worse

GDB is used almost exclusively for remote debugging, yet has no concept of:

- multiple address spaces (or virtual memory in general),
- multiple CPUs,
- special registers, such as CR3 or MSRs.

04 Making it Worse

GDB is used almost exclusively for remote debugging, yet has no concept of:

- multiple address spaces (or virtual memory in general),
- multiple CPUs,
- special registers, such as CR3 or MSRs.

It is an application debugger after all.

05 Outline

05 Noteworthy Alternatives to Debugging

Use the right tool for the job at hand:

Valgrind a toolkit to build sophisticated analysis programs using binary translation. Ported to L4!

- memory/heap analysis
- thread-safety analysis
- cache analysis

dtrace an advanced tracing framework for Solaris/FreeBSD/. . . . Can be used to find non-functional/performance bugs.

- gathers information from an abundance of probe points in the kernel
- small overhead
- uses an AWK-like language for scripting

05 Better not debug at all

Writing bug-free code is *not that hard*:

- *Version Control!*
- *Document your code!*
- Make assumptions explicit!
- Write test cases!
- Use a proper language (when you can)!
  - bounds checking
  - garbage collection
  - expressive type system
  - generic programming/macros

05 Next Week

Next week's lecture will be about the *Genode Operating System Framework* by Norman Feske, OS group alumni and co-founder of Genode Labs.

The last lecture will be on *Resilience* on January 29th.

05 References I


Victor P. Nelson, 1990.
Fault-tolerant Computing: Fundamental Concepts
`http://www.win.tue.nl/space4u/documents/papers/VPNeslon90.pdf`


Algirdas Avizienis et al, 2001
Fundamental Concepts of Dependability
`http://ce.aut.ac.ir/~dehghan/Course/SFT/FundamentalConceptsofDependability.pdf`


DWARF Debugging Information Format
`http://dwarfstd.org/doc/Dwarf3.pdf`