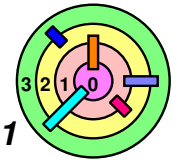
Ch 3: Basic Concepts

Bill Cheng

http://merlot.usc.edu/william/usc/



What's Next?

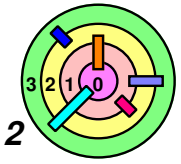
- So far, we have talked about abstractions
  - processes, files, threads
    - stuff at the user level
- We are not ready to talk about the OS yet
- Next step is something in between

Abstractions
(processes, files, threads)

- context for execution
- I/O architecture
- dynamic storage allocation
- linking & loading
- booting

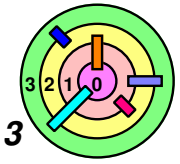
User

OS



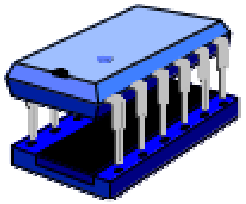
3.1 Context Switching

- Procedures
- Threads & Coroutines
- Systems Calls
- Interrupts



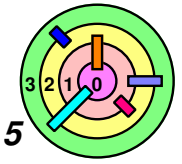
Context Switching

- The magic of OS
  - to provide the *illusion* that applications run *concurrently* and each application thinks it's the only application running on the processor
- The OS switches the processor from one application to another
  - switching happens *transparently* to the applications
- What is the OS doing when an application is running?



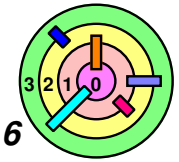
Context

- What's the execution context of a thread?
  - if we are going to talk about context switching, we need to know what we are switching and how to get back
- The *execution context* of a thread is the *current state* of our thread
  - what does the execution context include?
    - CPU registers, including the *instruction pointer*, *stack pointer*, *base/frame pointer*, etc.
    - stack
    - open files
    - etc.
    - i.e., everything that may affect the execution of the thread
  - turns out the stack is complicated
    - in reality, it's just the *current stack frame* of the current thread
    - what's *below* the current stack frame (and the rest of the address space) is also part of the thread's state/context



3.1 Context Switching

- *Procedures*
- Threads & Coroutines
- Systems Calls
- Interrupts

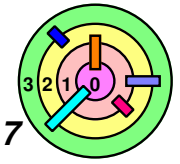


Subroutines

```
int main() {
  int i;
  int a;
  ...
  i = sub(a, 1);
  ...
  return(0);
}
```

```
int sub(int x, int y) {
  // computes x^y
  int i;
  int result = 1;
  for (i=0; i<y; i++)
    result *= x;
  return(result);
}
```

- You are in `main()` and are ready to call `sub()`
  - how do you make sure that `sub()` has the right context to execute the code in `sub()`?
    - you need to *prepare/setup the context* for `sub()`
  - how do you make sure that you can return from `sub()` and restore the `main()` context and continue to execute properly?
    - you need to first *save* the context of `main()`
    - after return from `sub()`, you need to *restore* the context of `main()` so `main()` can *resume* execution

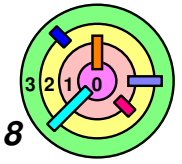


Subroutines

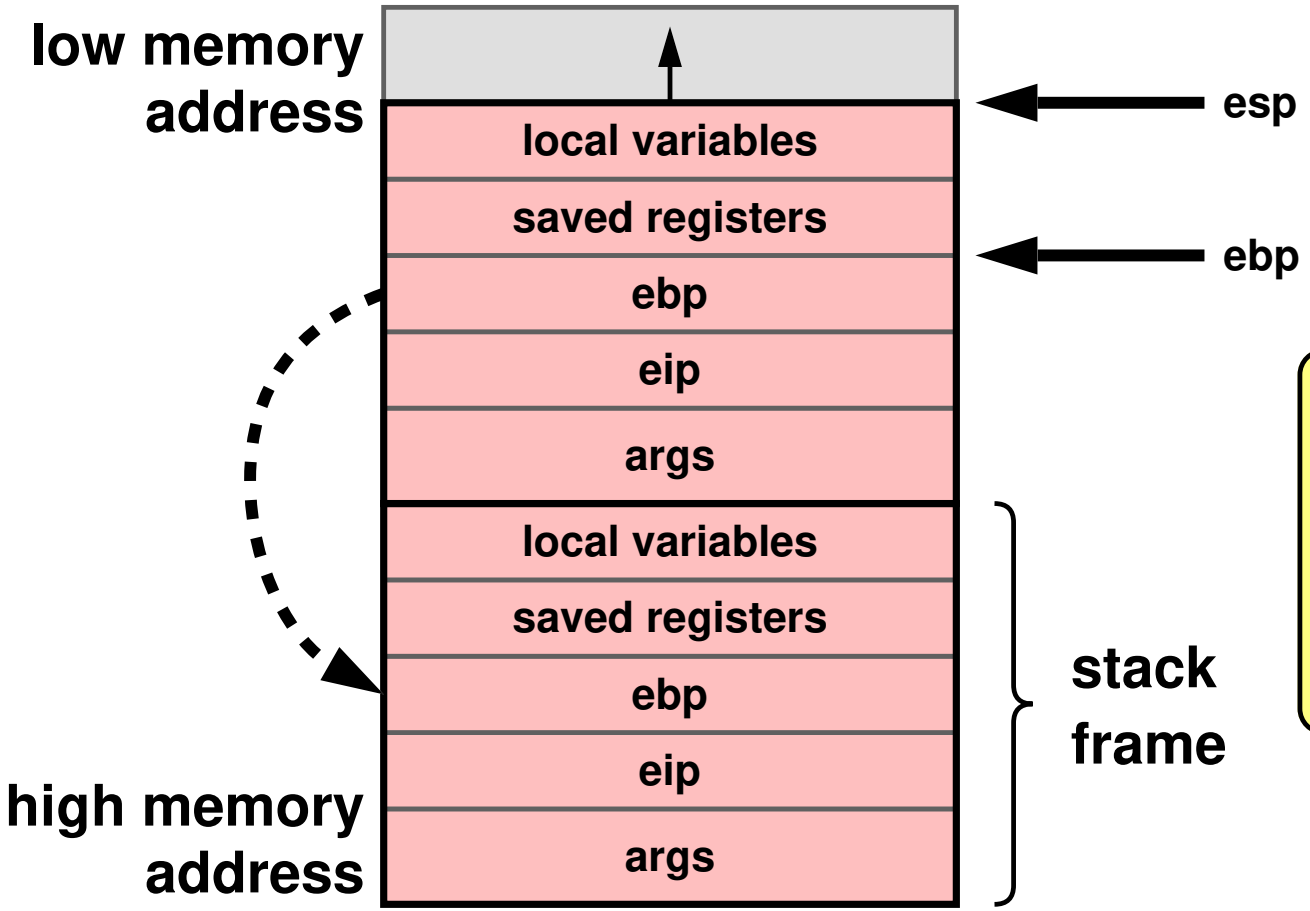
```
int main() {
  int i;
  int a;
  ...
  i = sub(a, 1);
  ...
  return(0);
}
```

```
int sub(int x, int y) {
  // computes x^y
  int i;
  int result = 1;
  for (i=0; i<y; i++)
    result *= x;
  return(result);
}
```

- The context of `main()` includes CPU registers, any global variables (none here) and its local variables, `i` and `a`
- The context of `sub()` includes
  - any global variables, none here
  - its local variables, `i` and `result`
  - its arguments, `x` and `y`
- Global variables are in fixed location in the address space
- *Local variables* and *arguments* are in *current stack frame*

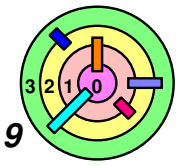


Intel x86 (32-Bit): Subroutine Linkage

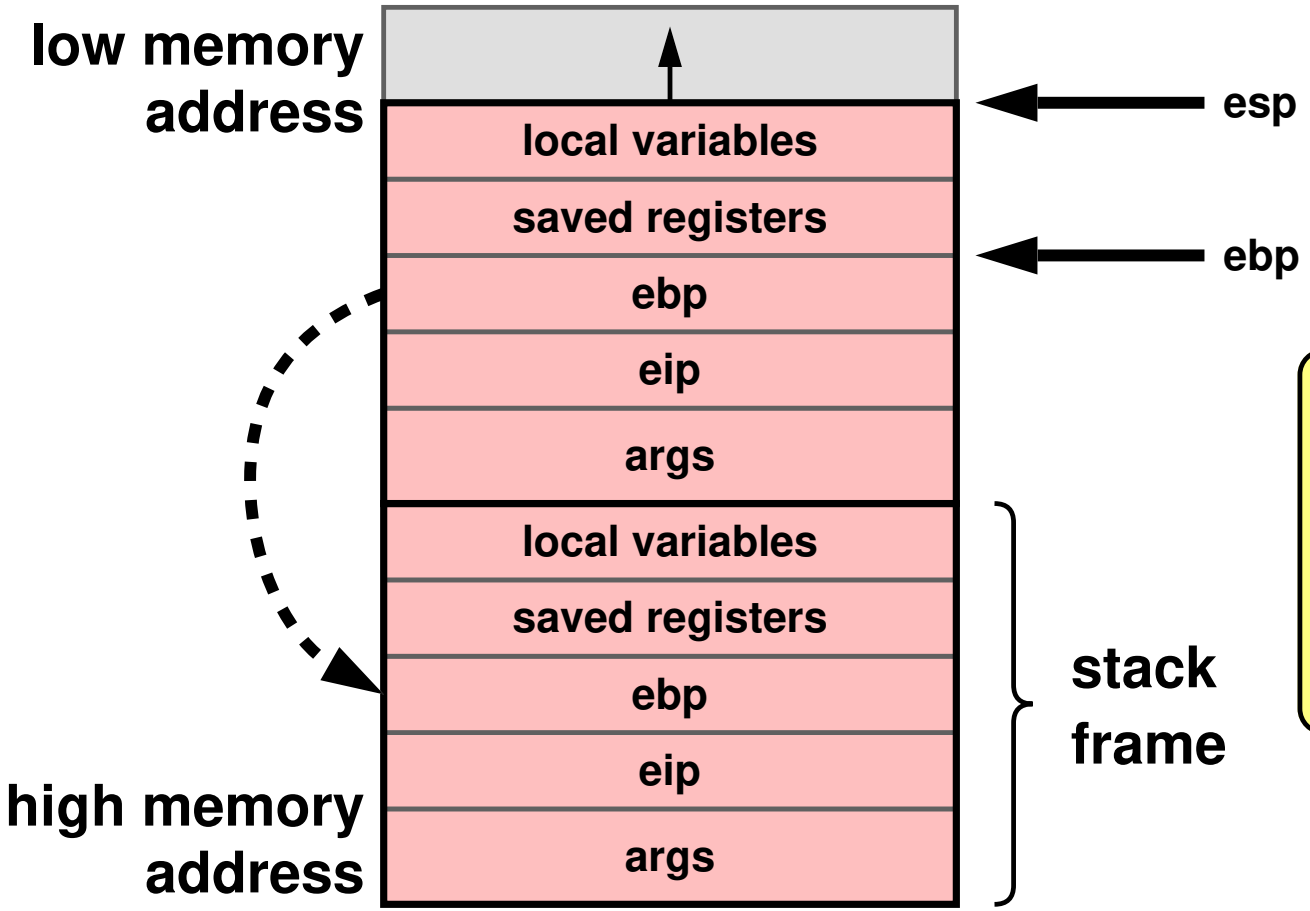


- please be reminded that our address space is up-side-down (comparing against the textbook)

- *esp* points to the end/top of the current stack frame
  - it is used to prepare the next stack frame
- *eip* contains the caller's *instruction pointer* register
  - in the stack frame, this is the *return address!*

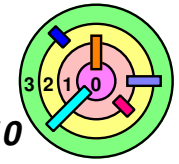


Intel x86 (32-Bit): Subroutine Linkage



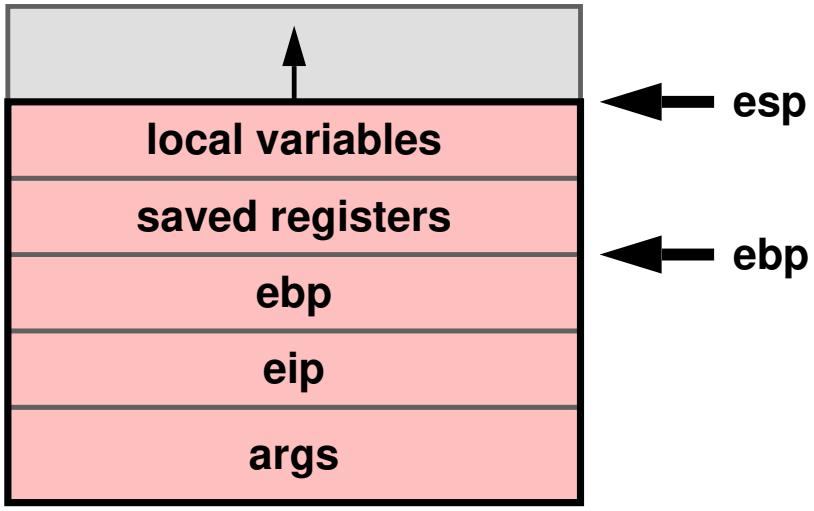
- please be reminded that our address space is up-side-down (comparing against the textbook)

- *ebp* contains the caller's *base (frame) pointer* register
  - this is a link to the caller's *stack frame*
- *eax* contains the return value of a function
- some fields are not alwasy present, compiler decides



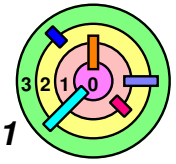
Intel x86 (32-Bit): Subroutine Linkage

- Who sets what?
  - *args* is explicitly setup by the *caller*
  - *eip* is copied into the stack frame by a "`call`" machine instruction in the *caller* function
  - *ebp* is copied explicitly by the *callee*
  - other registers are saved explicitly by the *callee* code in the "saved registers" region
    - as it turned out, for x86, some registers are designated to be saved by the callee code
  - space for local variables is *created* explicitly by the *callee* code
    - local variables are *not* initialized automatically



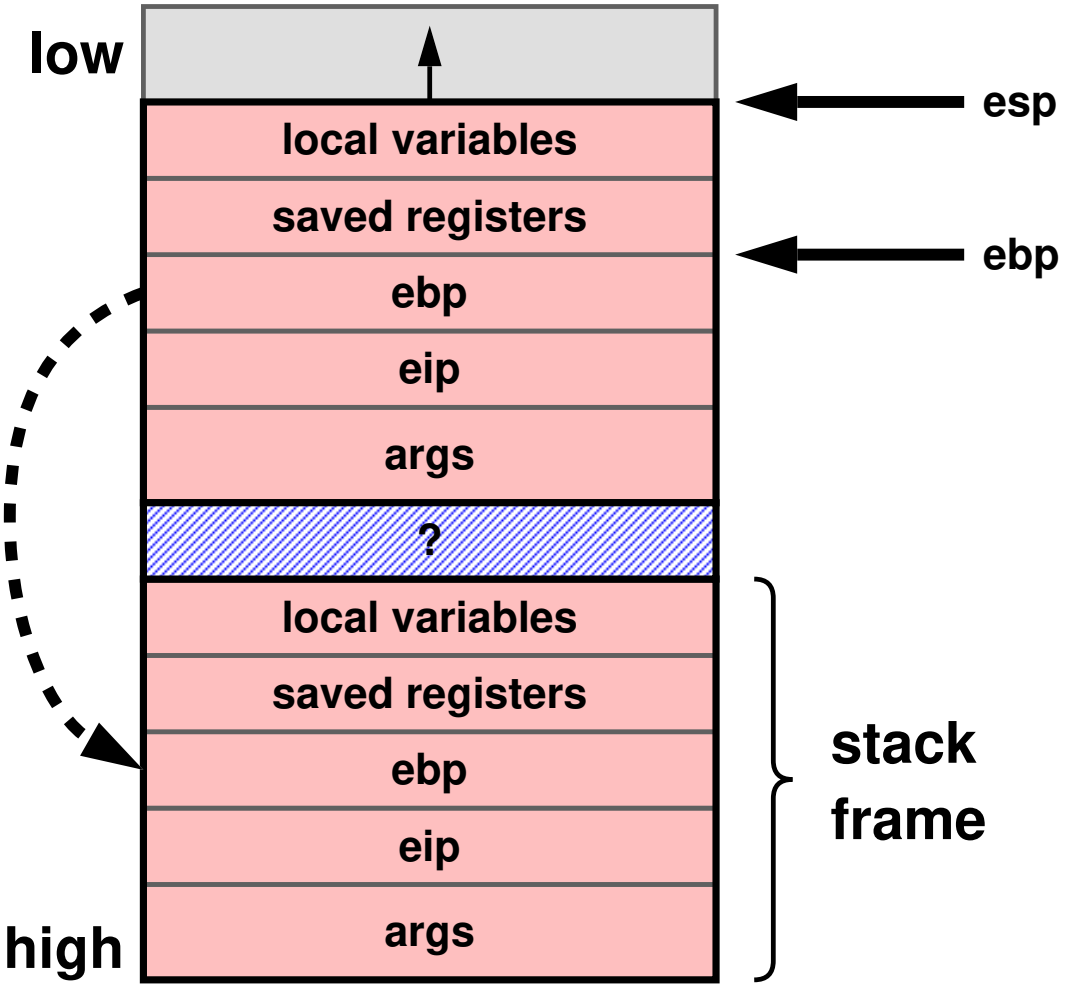
- What does the stack frame look like for the following function?

```
void func() { printf("I'm here.\n"); }
```



Intel x86 (32-Bit): Subroutine Linkage

- In reality, there can be stuff between stack frames
  - e.g., by convention, specific registers are saved and restored by the caller (this can depend on the compiler)



Intel x86: Subroutine Code (1)

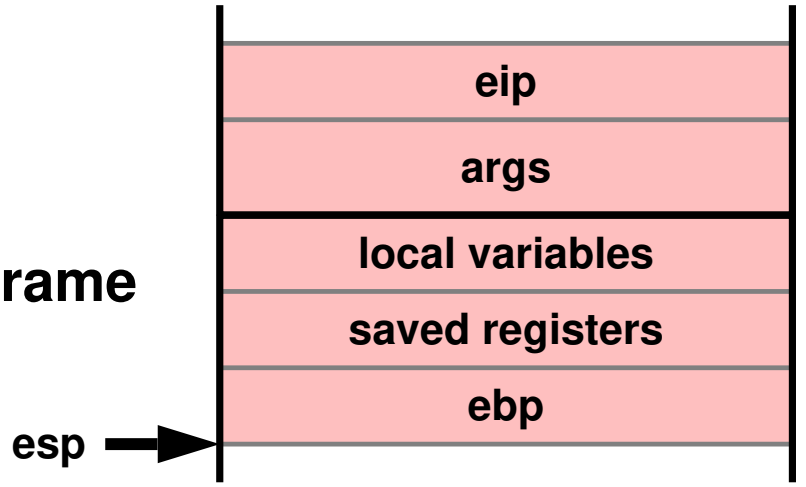
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

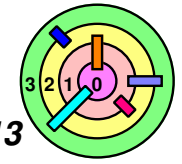
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

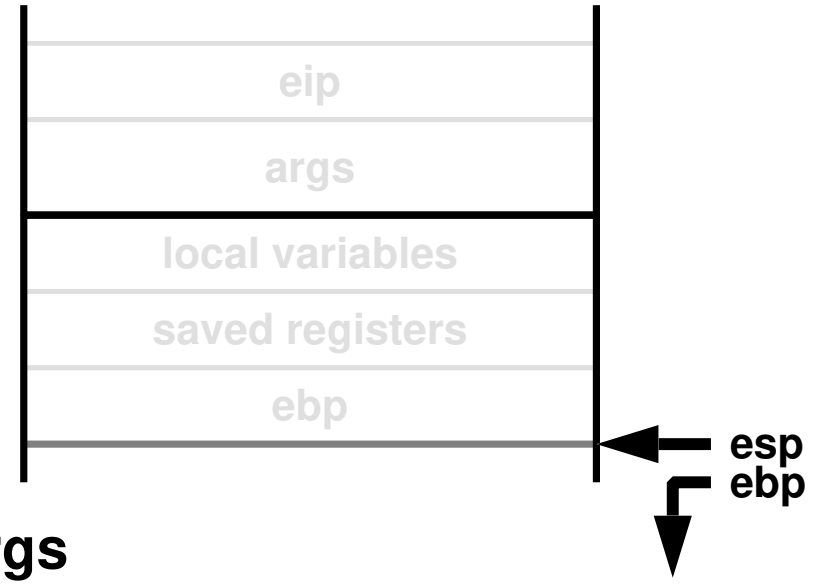
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

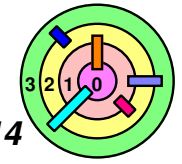
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

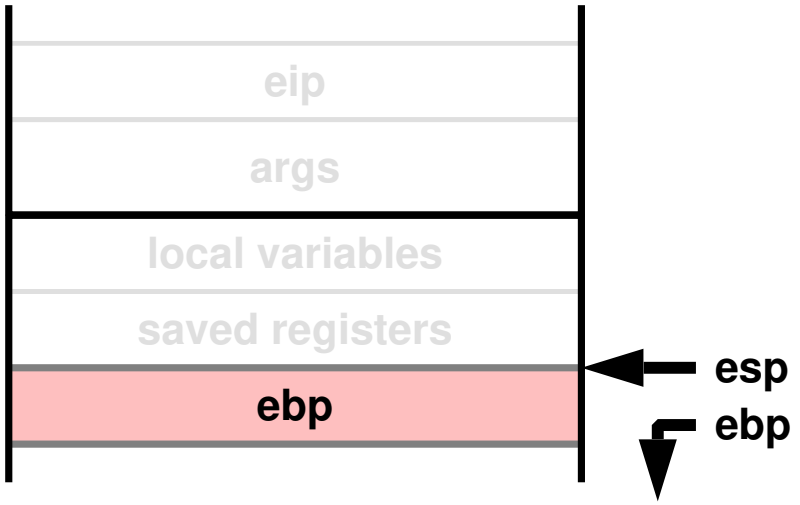
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

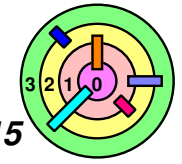
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

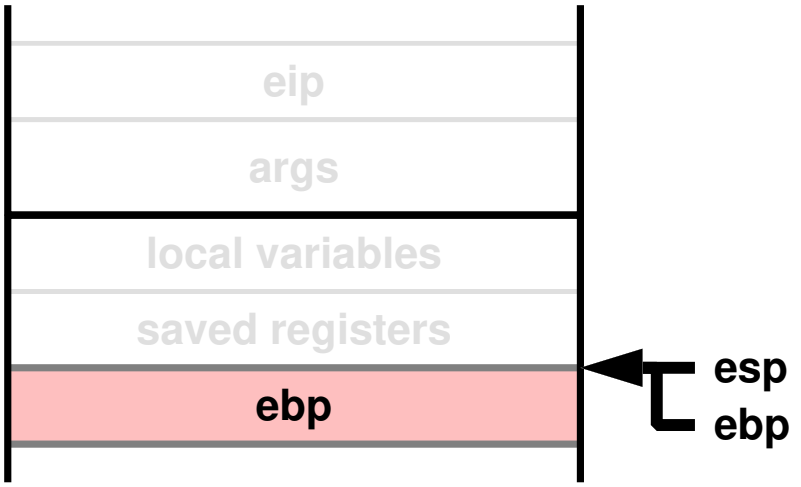
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

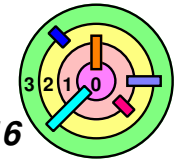
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

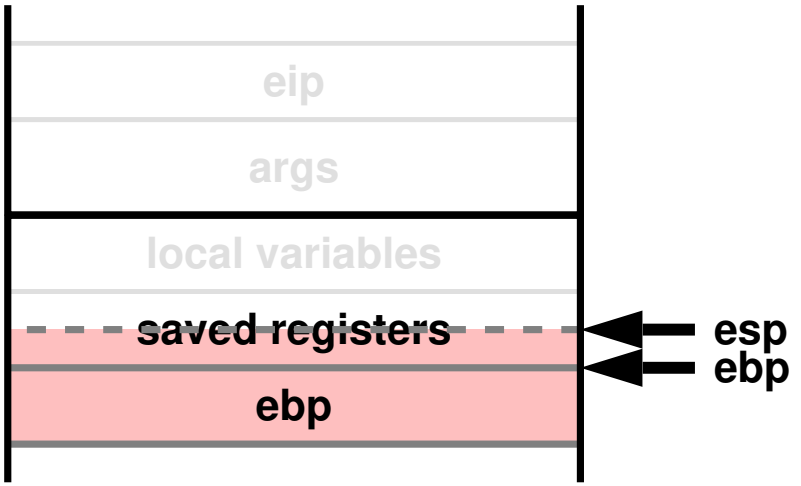
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

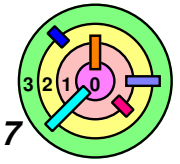
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

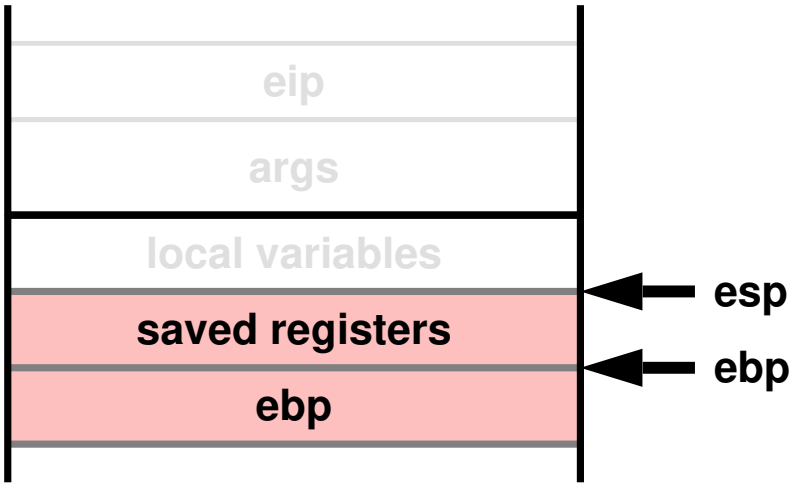
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

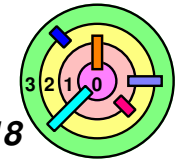
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

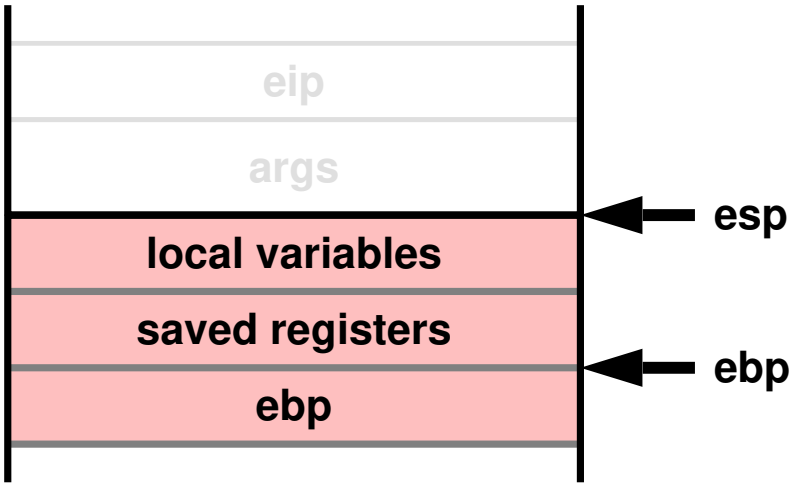
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

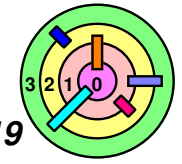
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl 8(%ebp)
   movl -12(%ebp), %esi
   pushl %esi
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

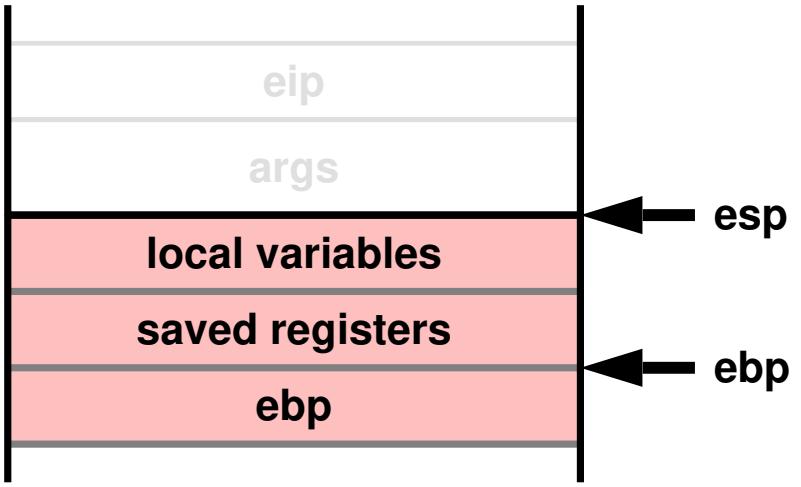
set up stack frame

- now you are ready to execute C code
- in gdb, if you break in `main()`, the breakpoint is set here

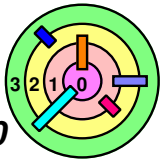
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

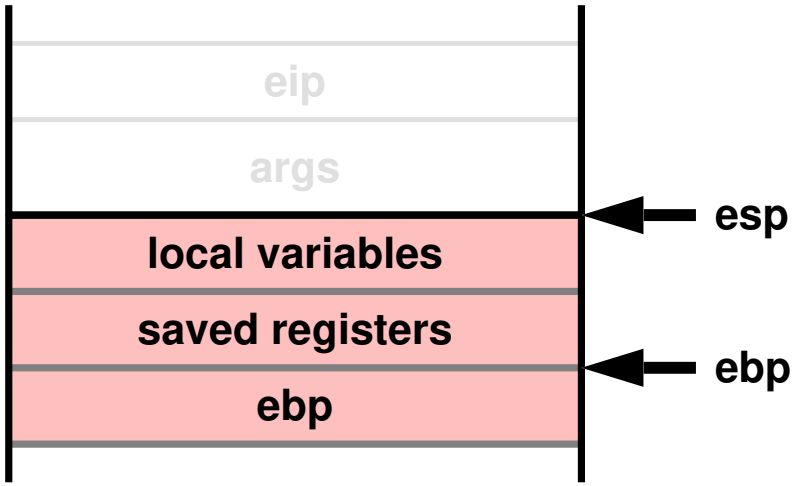
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

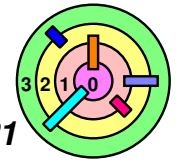
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

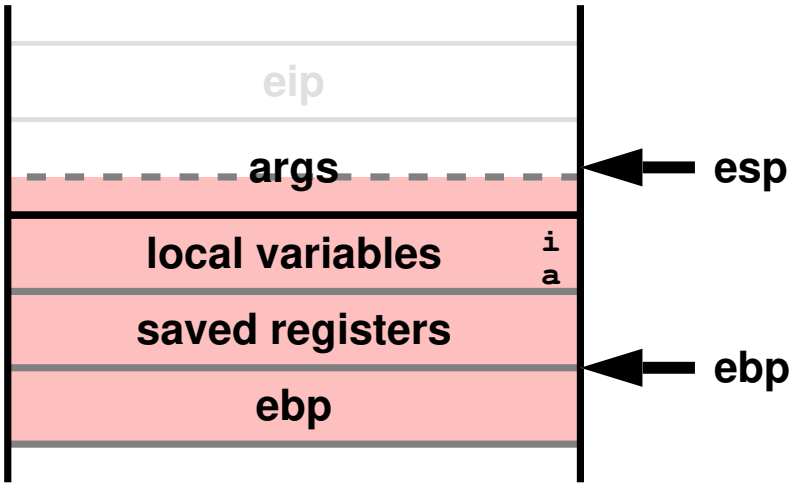
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

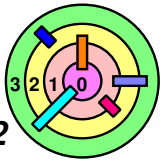
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

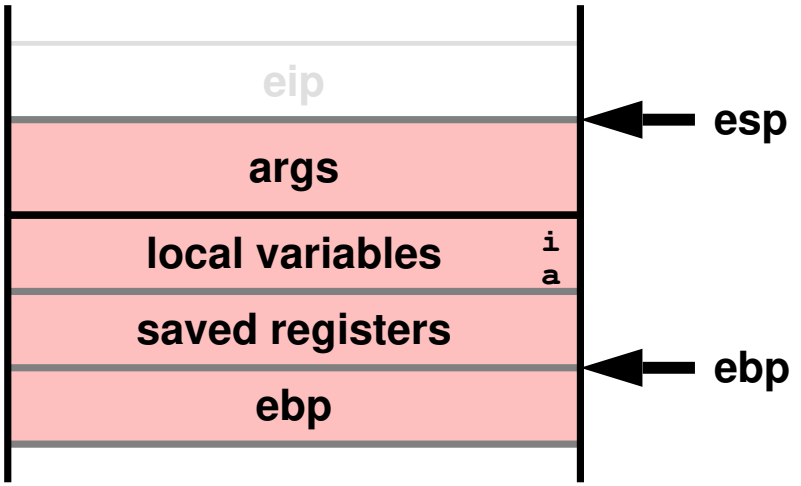
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

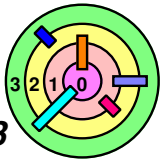
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

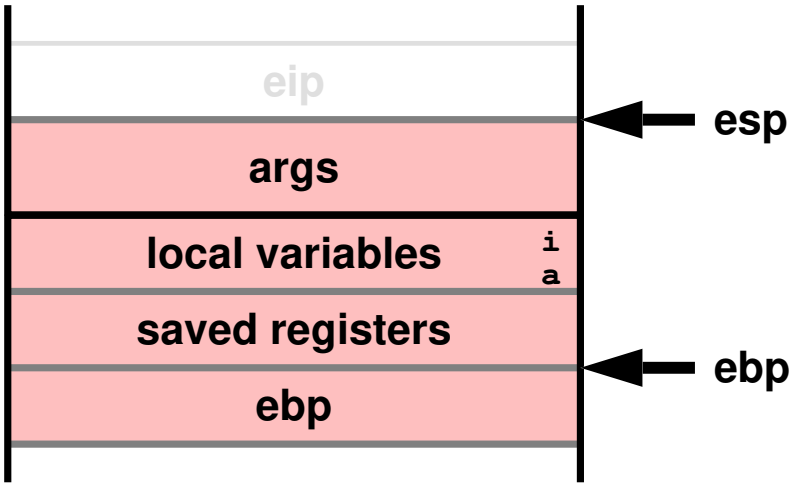
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

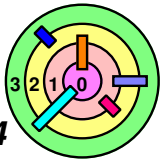
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

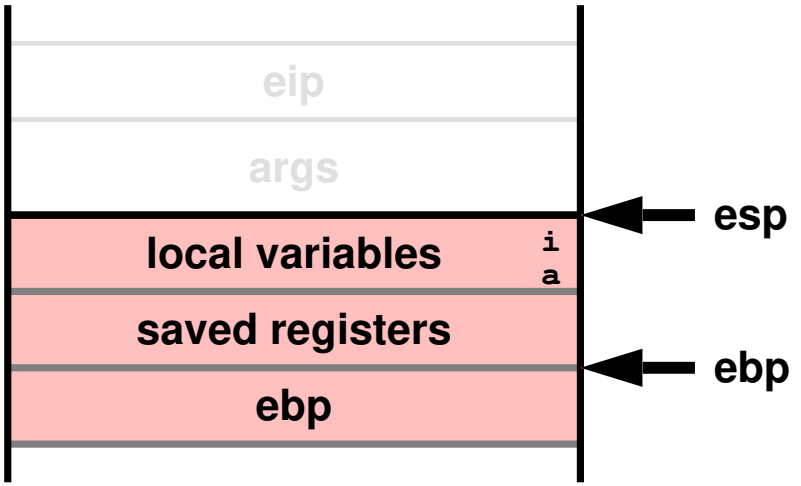
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

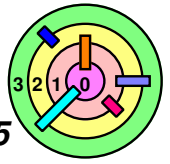
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

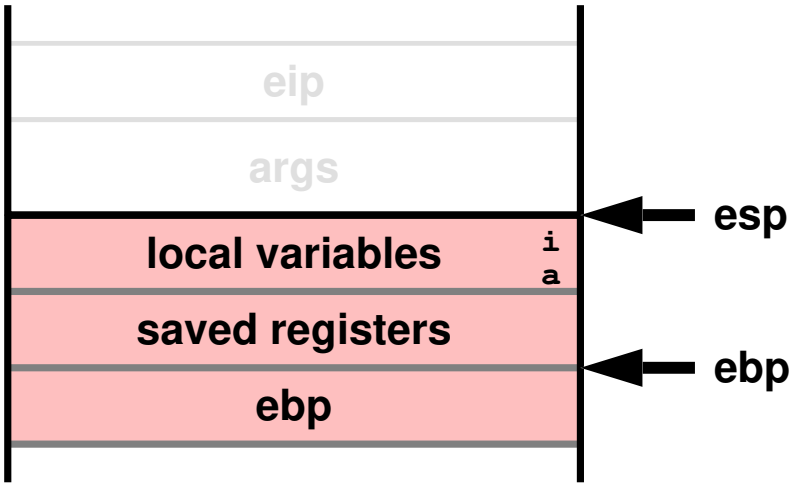
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

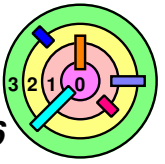
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $
   movl $
   popl %
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

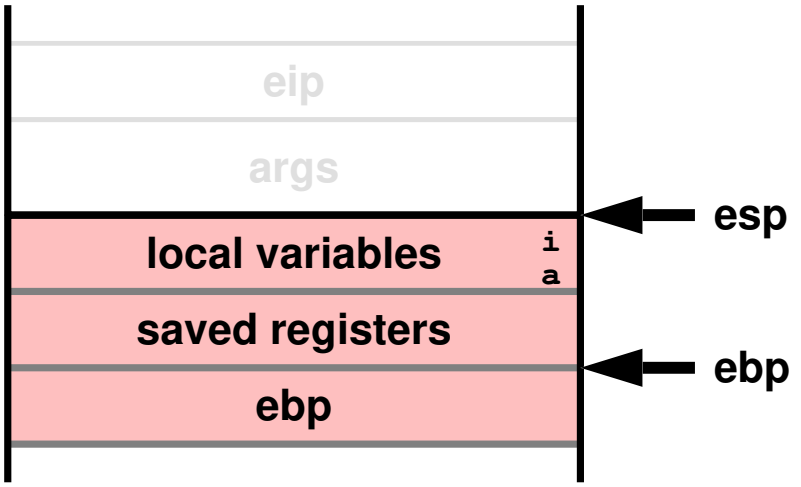
set up stack frame

push args

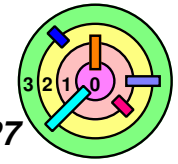
pop args; result

value and restore frame

- what happens when you make a subroutine call now?




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

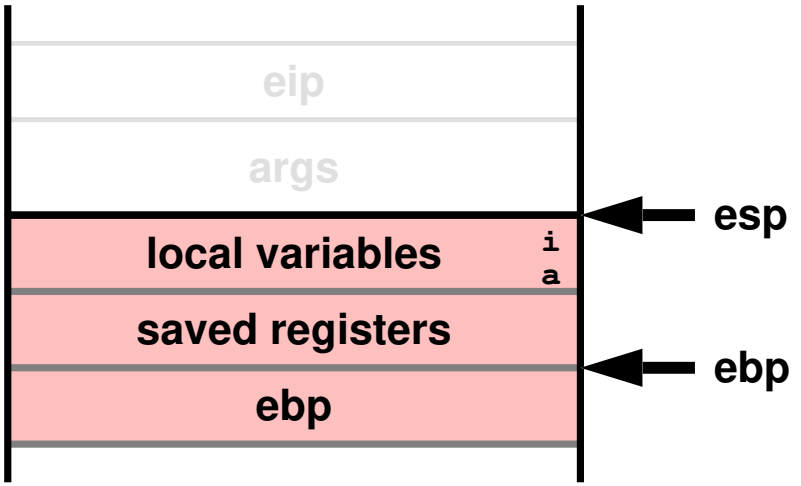
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

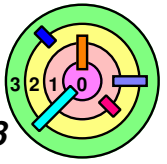
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

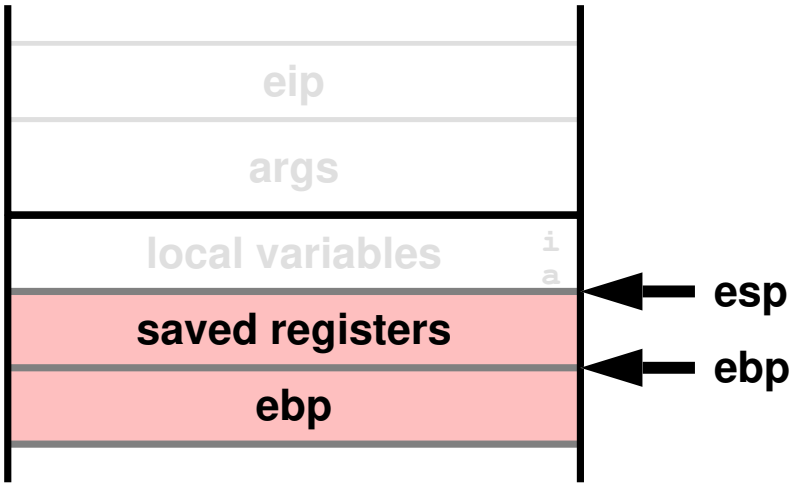
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

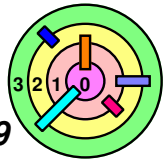
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

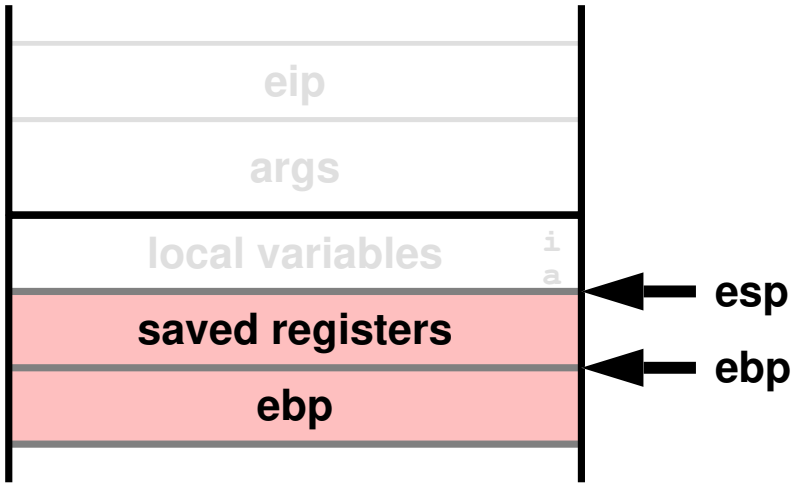
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

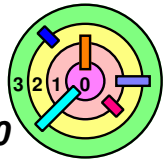
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

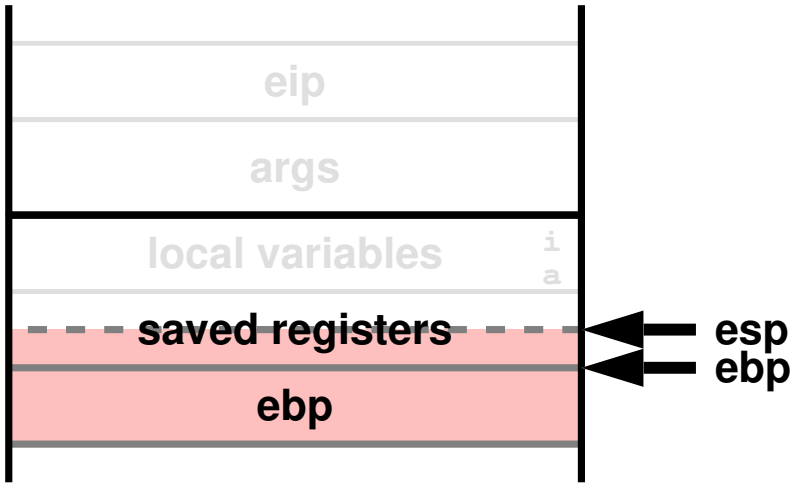
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

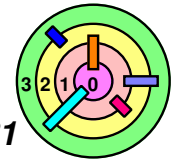
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

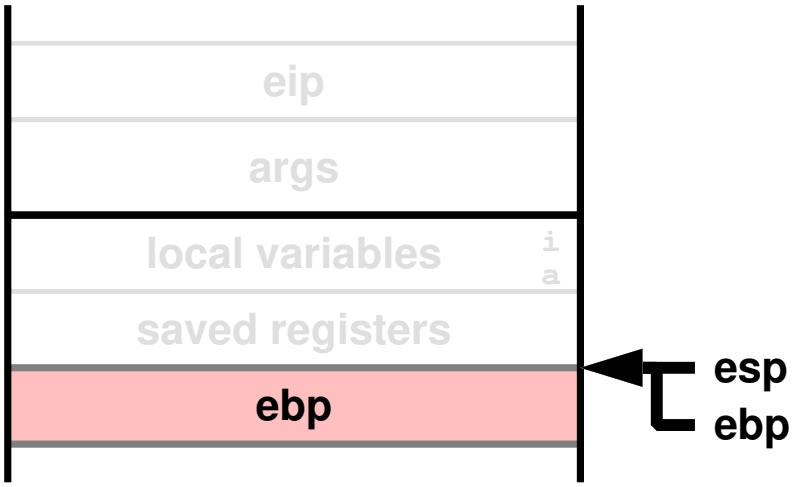
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

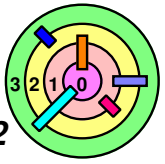
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

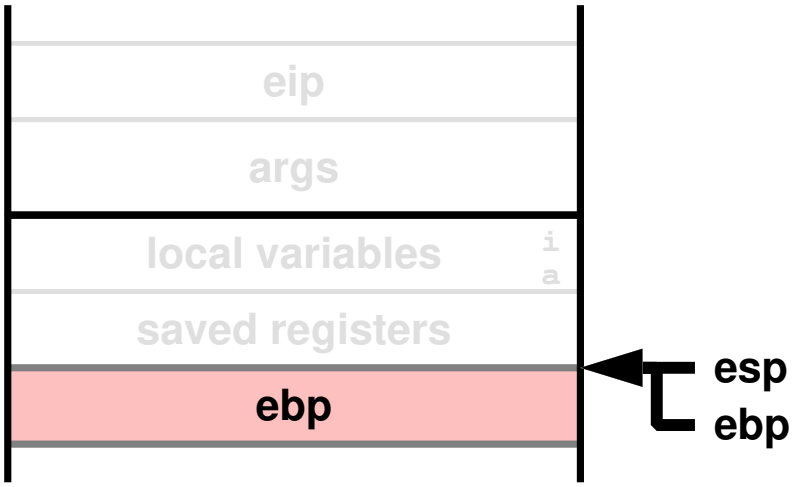
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

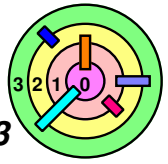
push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```



Intel x86: Subroutine Code (1)

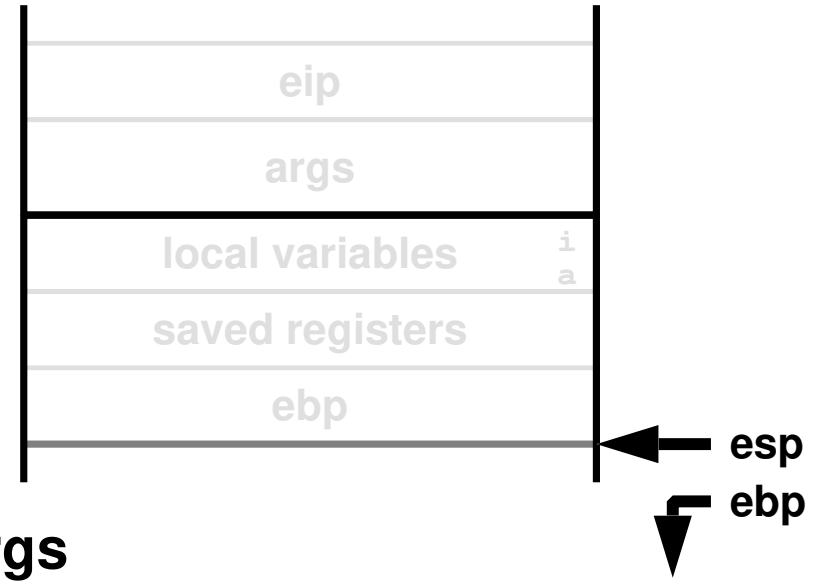
```
main:
   pushl %ebp
   movl %esp, %ebp
   pushl %esi
   pushl %edi
   subl $8, %esp
   ...
   pushl $1
   movl -12(%ebp), %eax
   pushl %eax
   call sub
   addl $8, %esp
   movl %eax, -16(%ebp)
   ...
   addl $8, %esp
   movl $0, %eax
   popl %edi
   popl %esi
   movl %ebp, %esp
   popl %ebp
   ret
```

set up stack frame

push args

pop args; get result

set return value and restore frame




```
int main() {
   int i;
   int a;
   ...
   i = sub(a, 1);
   ...
   return(0);
}
```