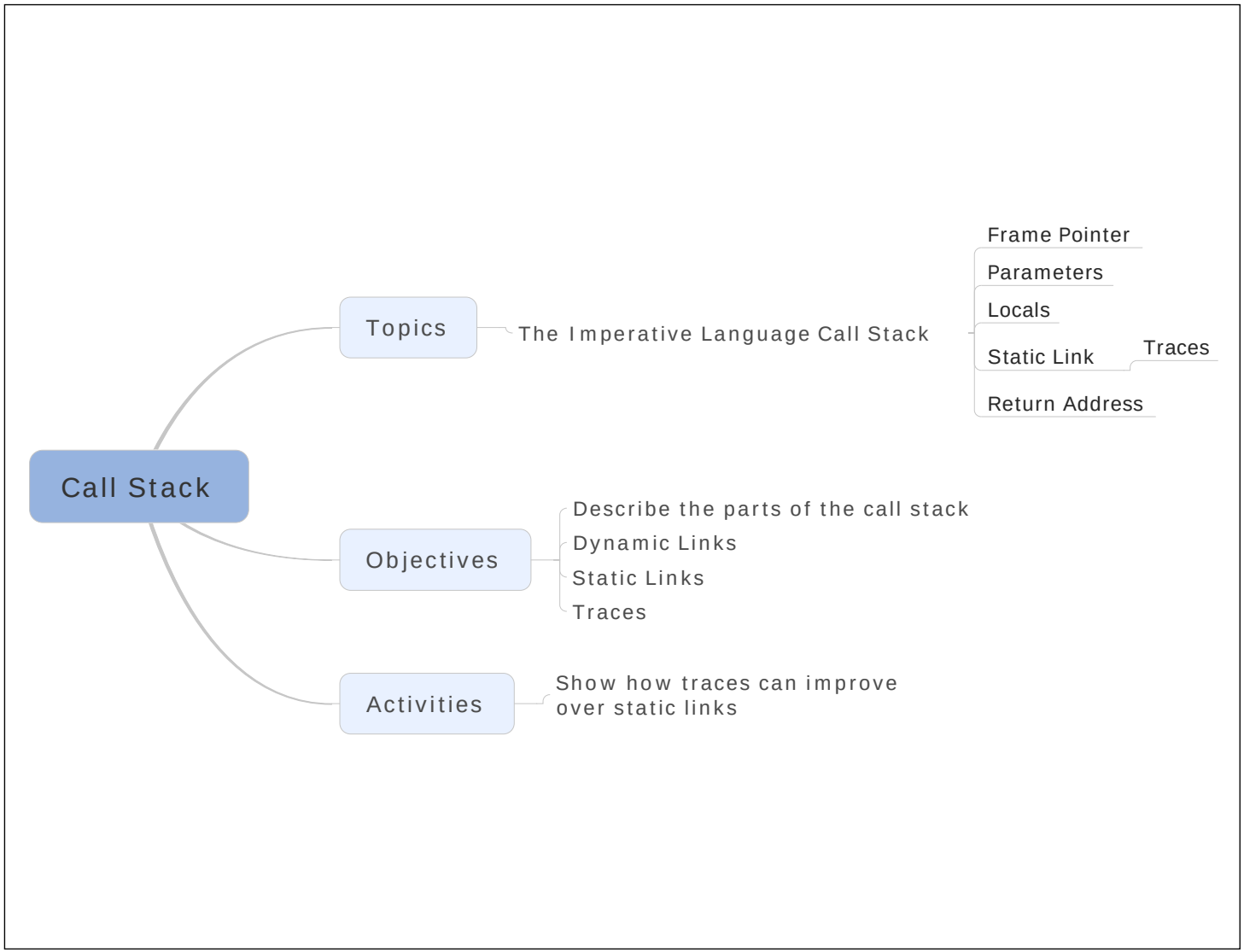
Call Stack

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Outline


Call Stack
Topics
The Imperative Language Call Stack
Frame Pointer
Parameters
Locals
Static Link
Traces
Return Address
Objectives
Describe the parts of the call stack
Dynamic Links
Static Links
Traces
Activities
Show how traces can improve over static links

C-Like Languages

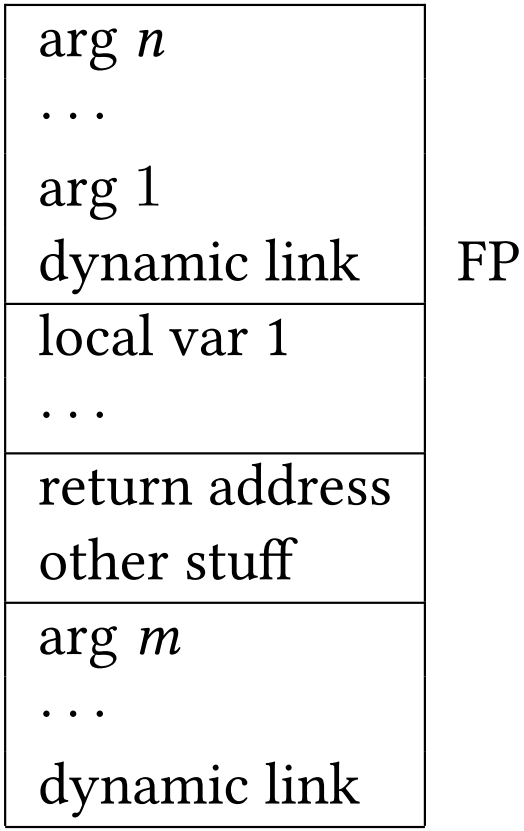
C-like languages have functions with these properties:

- Two layers of scope (local and global).
- Multiple, possibly variable number of parameters.
- Functions not first-class.

In order to call a function in a C-like language, we need space on the stack for several things.

- Return address
- Arguments
- Pointer to previous stack frame
- Local variables

Stack Frame Diagram for C



- The return address points to the machine code of the calling function.
- The dynamic link points to the stack frame of the calling function.
  - Don't confuse them!!
- Registers, temporary values, etc. get put in the "other stuff" section.

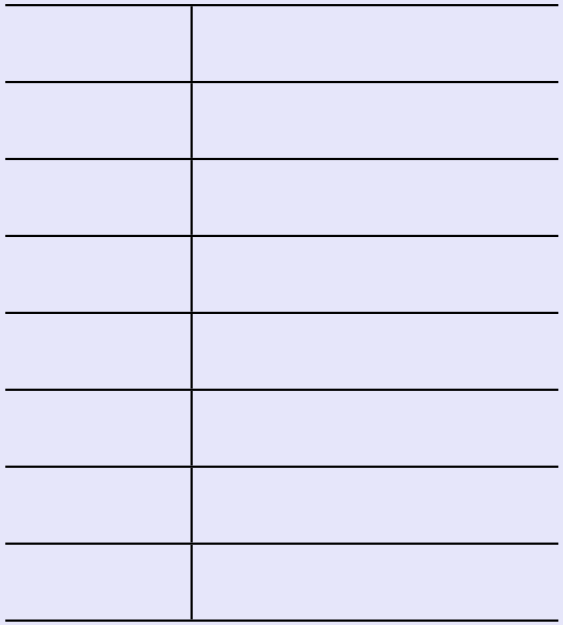
Example Function Call

- Call `foo` with 1,2,3
- Old stack frame at `0x01c8`
- Program counter at `0xff80`

C Code

```
1 int foo(int a, int b, int c) {
2     int d = 10;
3     int e = 20;
4     return a + e + d;
5 }
```

Stack Frame



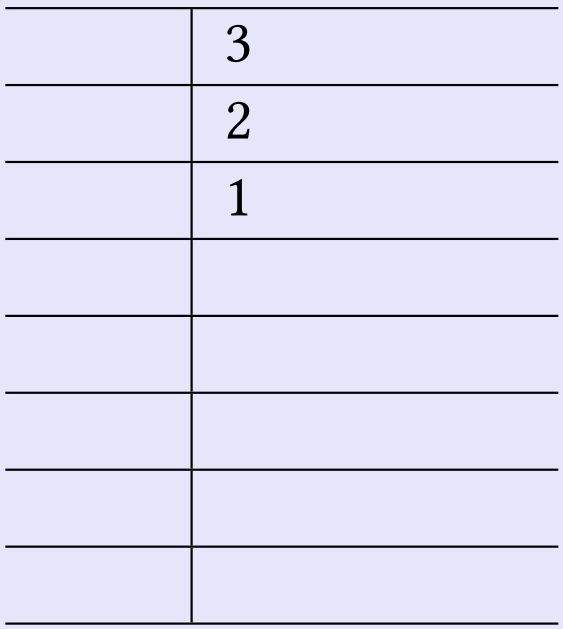
Example Function Call

- Call `foo` with 1,2,3
- Old stack frame at `0x01c8`
- Program counter at `0xff80`

C Code

```
1 int foo(int a, int b, int c) {
2     int d = 10;
3     int e = 20;
4     return a + e + d;
5 }
```

Stack Frame



Example Function Call

- Call `foo` with 1,2,3
- Old stack frame at `0x01c8`
- Program counter at `0xff80`

C Code

```
1 int foo(int a, int b, int c) {
2     int d = 10;
3     int e = 20;
4     return a + e + d;
5 }
```

Stack Frame

	3
	2
	1
FP	0x01c8

Example Function Call

- Call `foo` with 1,2,3
- Old stack frame at `0x01c8`
- Program counter at `0xff80`

C Code

```
1 int foo(int a, int b, int c) {
2     int d = 10;
3     int e = 20;
4     return a + e + d;
5 }
```

Stack Frame

	3
	2
	1
FP	0x01c8
	10
	20

Example Function Call

- Call `foo` with 1,2,3
- Old stack frame at `0x01c8`
- Program counter at `0xff80`

C Code

```
1 int foo(int a, int b, int c) {
2     int d = 10;
3     int e = 20;
4     return a + e + d;
5 }
```

Stack Frame

	3
	2
	1
FP	0x01c8
	10
	20
	0xff80

Example Function Call

- Call `foo` with 1,2,3
- Old stack frame at `0x01c8`
- Program counter at `0xff80`

C Code

```
1 int foo(int a, int b, int c) {
2     int d = 10;
3     int e = 20;
4     return a + e + d;
5 }
```

Stack Frame

	3
	2
	1
FP	0x01c8
	10
	20
	0xff80
	tmp stuff

Example Function Call

- Call `foo` with 1,2,3
- Old stack frame at `0x01c8`
- Program counter at `0xff80`

C Code

```
1 int foo(int a, int b, int c) {
2     int d = 10;
3     int e = 20;
4     return a + e + d;
5 }
```

Stack Frame

	31
	2
	1
FP	0x01c8
	10
	20
	0xff80
	tmp stuff

Pascal Like Languages

- Many languages have *nested scope*... functions can be defined within functions.
- Pascal is the most famous.
- OCaml (and most functional languages) let you do this too.

```
let rec foo a =
  let t = 10 + a in
  let bar b =
    let u = 20 in a + u + b + t + foo 9
  let baz c =
    let v = 30 in a + v + c + t + bar 5
  in baz 4
```

- We now need a pointer to the *parent scope's* stack frame.

Example

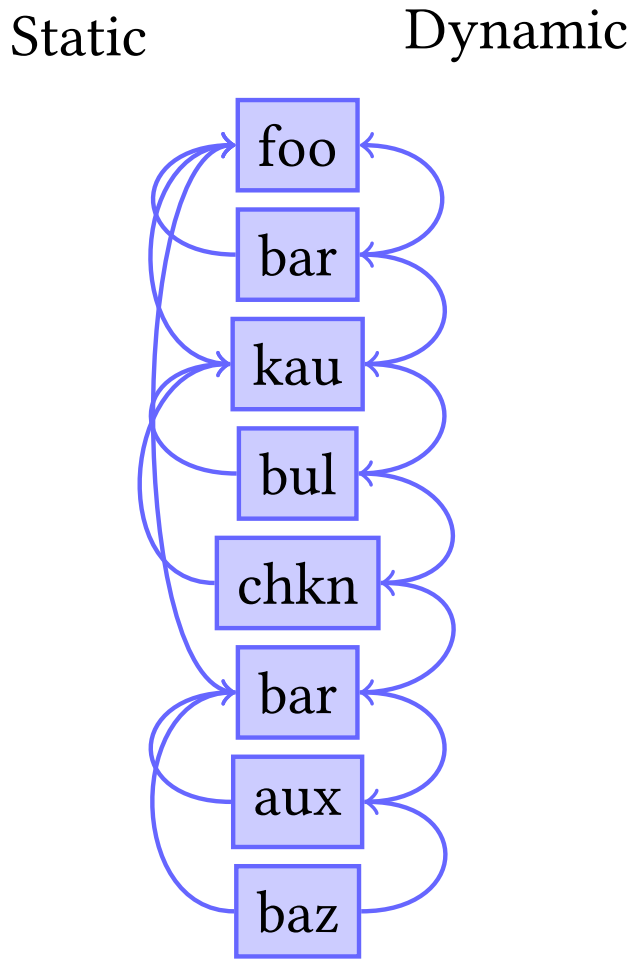
- Consider this code, where indentation denotes scope:

```
1 foo x = ..
2     bar y = ...
3         baz z = ...
4         aux x = ...
5     kau y = ...
6         bul z = ...
7         chkn w = ...
```

Suppose we have the call sequence
$foo \rightarrow bar \rightarrow kau \rightarrow bul \rightarrow chnk \rightarrow bar \rightarrow aux \rightarrow baz$

Example

```
1 foo x = ..
2     bar y = ...
3         baz z = ...
4         aux x = ...
5     kau y = ...
6         bul z = ...
7         chkn w = ...
```



Suppose we have the call sequence

$foo \rightarrow bar \rightarrow kau \rightarrow bul \rightarrow chnk \rightarrow bar \rightarrow aux \rightarrow baz$