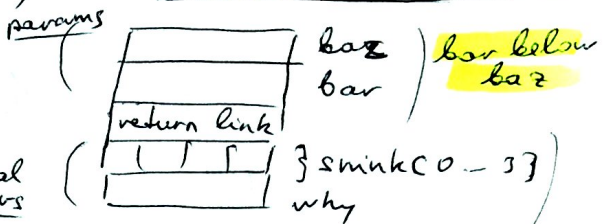
CS 107 lecture 10 Assembly

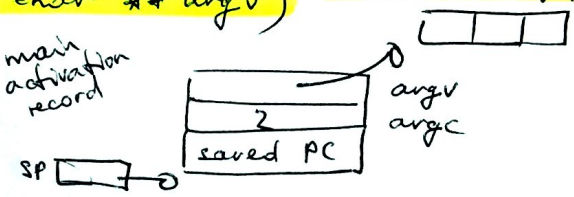
```
void foo (int bar, int *baz)
{
    char smink[4];
    short *why;
    why = (short *)(smink+2);
    *why = 50;
}

int main (int argc, char **argv)
{
    int i = 4;
    foo (i, &i);
    return 0;
}
```

8 Bytes

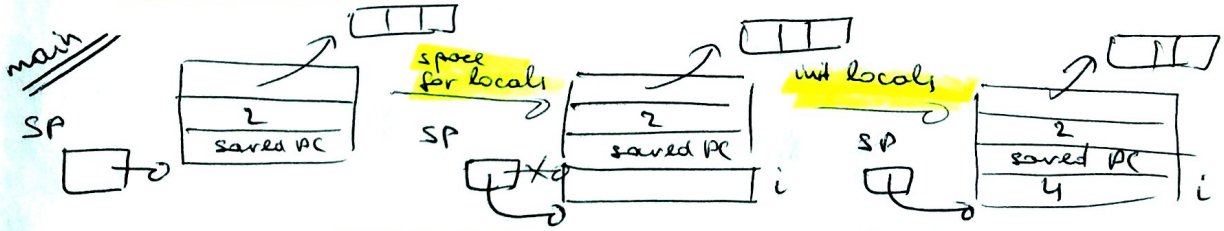
Activation record of foo



return link: what PC value would have been, if main was not interrupted by foo.



SP (stack pointer) points to the lowest address on stack that is relevant to execution. (R1 in previous examples).



SP = SP - 4;

M[SP] = 4;

foo

next:
- set a partial activation record for foo
- look at parameters of foo

space for params

```
SP = SP - 8;
R1 = M[SP+8];
R2 = SP + 8;
M[SP] = R1;
M[SP+4] = R2;
```

initialize params

transfer control to foo

a JMP instruction to jump to the first instruction associated with foo function, execute, jump back

```
CALL (foo)
SP = SP + 8;
```

SP

passed SP by CALL instruction; then SP = SP + 4

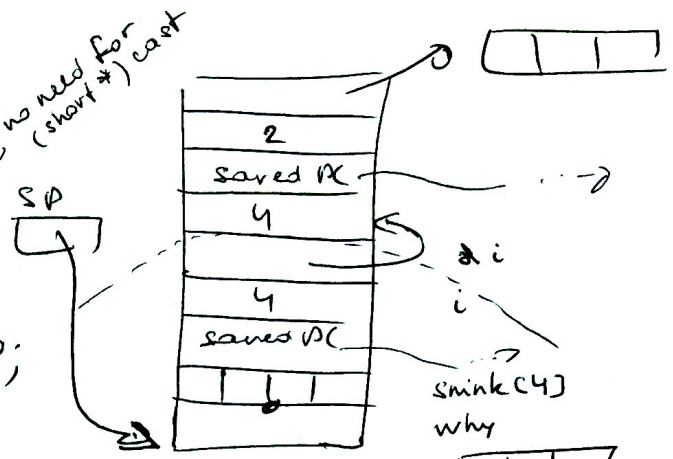
foo AR set by main

CS107 lecture 10

```
<foo>: SP = SP - 8;
       R1 = SP + 6;
       M[SP] = R1;
       R1 = M[SP];
       M[R1] = .2 50;
       SP = SP + 8;
       RET;
```

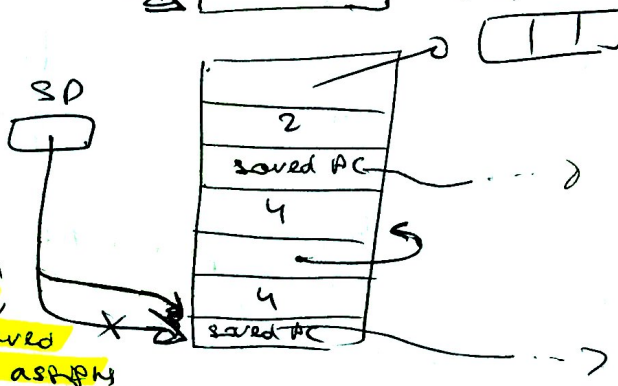
for control

no need for (short *) cast

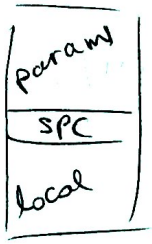


RV not used in foo but in main

dedicated to communicating return values



RET gets saved PC and assigns the true PC to it; then sets SP = SP + 4



allocated and init'ed by caller

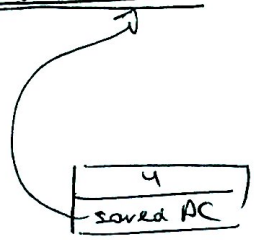
allocated and init'ed callee

add to main:

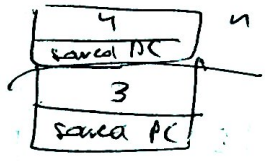
```
RV = 0;
RET;
```

```
int fact (int n)
{
  if (n == 0)
    return 1;
  return n * fact(n-1);
}
```

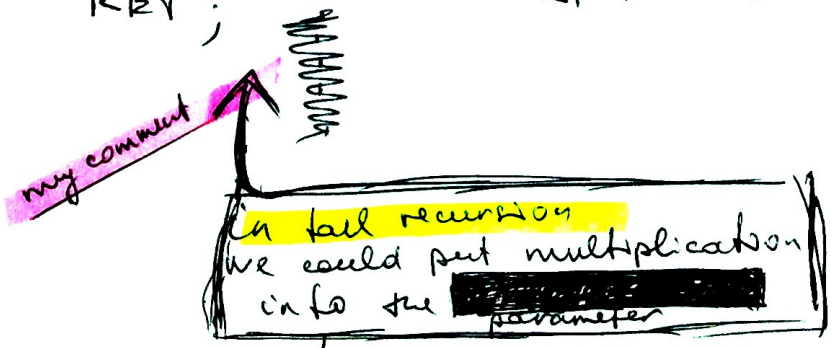
Recursive function




```
<fact>: R1 = M[SP + 4]; // past the saved PC, load n
        BNE R1, 0, PC + 12
        RV = 1;          } base case
        RET;
        R1 = M[SP + 4]; // n
        R1 = R1 - 1;    // n-1
        SP = SP - 4;     } partial AR
        M[SP] = R1;      } for recursive call
        CALL <fact>;
        SP = SP + 4;     } clean up local of call
        R1 = M[SP + 4];  } had to reload because
        RV = RV * R1;    } <fact> could reuse
        RET;             } R1 in another way
```



activation records continue to be allocated through recursive calls: rec. and then get deallocated.



↳ loop instead of recursion instructions by compiler

→ [redacted] prevent stack growth