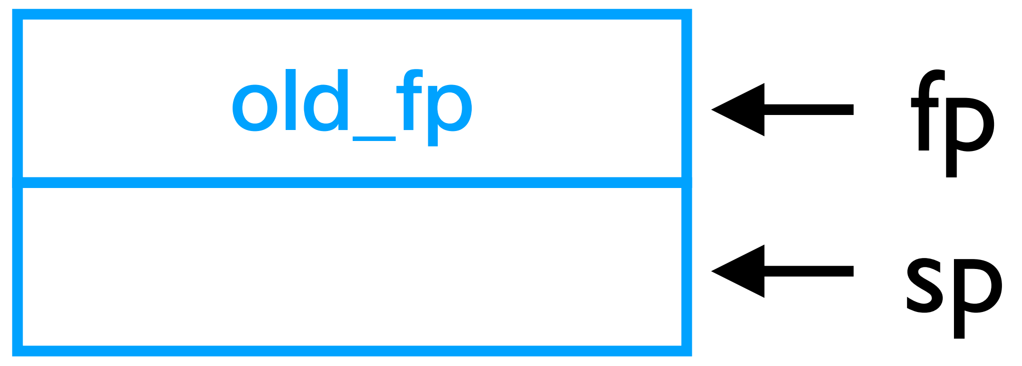
Code generation for functions

What should happen when foo calls bar?

1. Put Arguments on stack
2. Allocate space for return value
3. Save old return address (return address of foo) on stack
4. Jump to bar (using JR instructor)
5. Save old frame pointer (foo's frame pointer) on stack
6. Set frame pointer to point to top of stack
7. Allocate space for local variables of bar

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



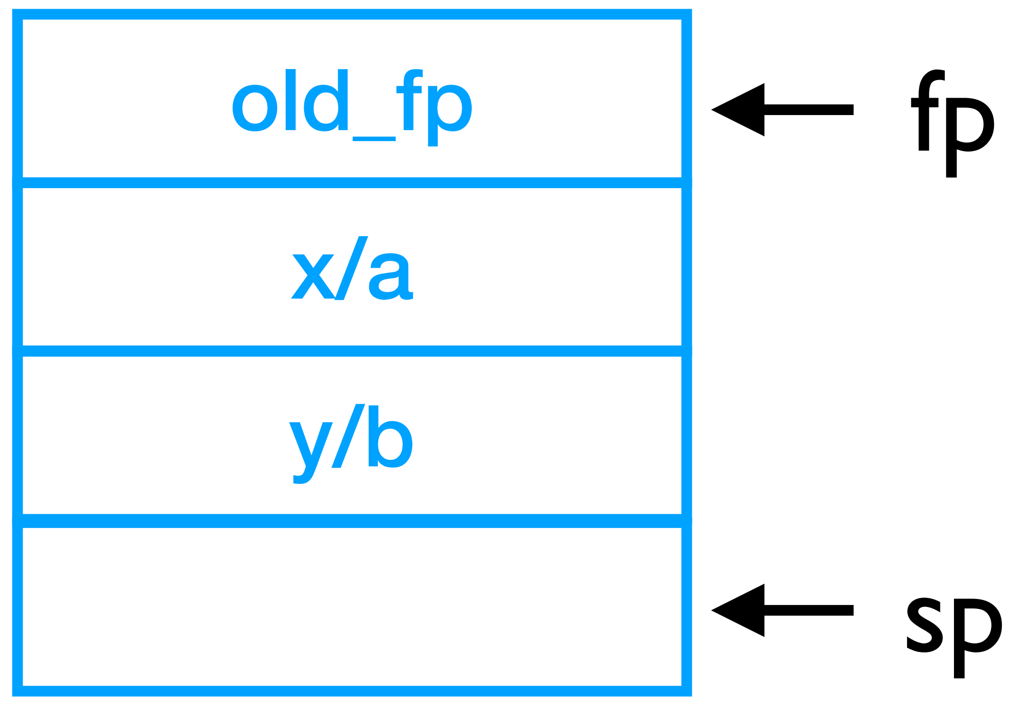
What should happen when foo calls bar?

1. Put Arguments on stack
2. Allocate space for return value
3. Save old return address (return address of foo) on stack
4. Jump to bar (using JR instructor)
5. Save old frame pointer (foo's frame pointer) on stack
6. Set frame pointer to point to top of stack
7. Allocate space for local variables of bar

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
```

```
int foo() {
   ...
   z = bar(x, y);
}

int bar(int a, int b) {
   int c;
}
```



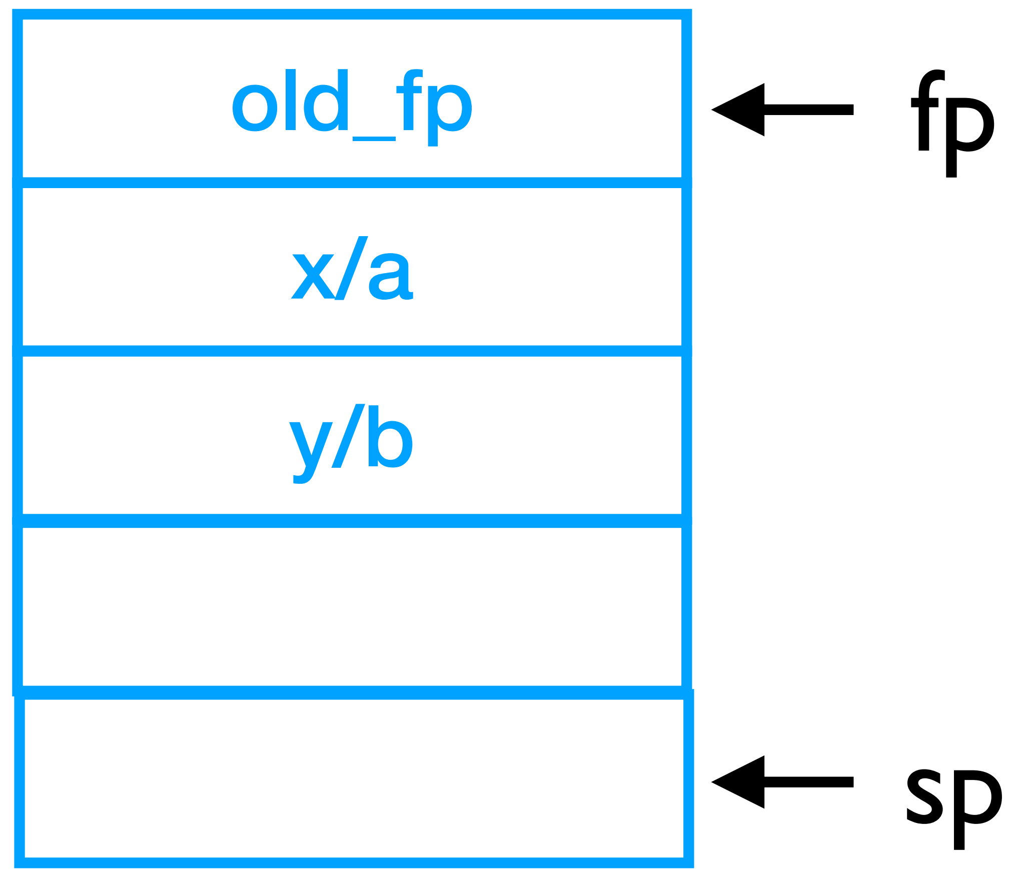
What should happen when foo calls bar?

1. Put Arguments on stack
2. Allocate space for return value
3. Save old return address (return address of foo) on stack
4. Jump to bar (using JR instructor)
5. Save old frame pointer (foo's frame pointer) on stack
6. Set frame pointer to point to top of stack
7. Allocate space for local variables of bar

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



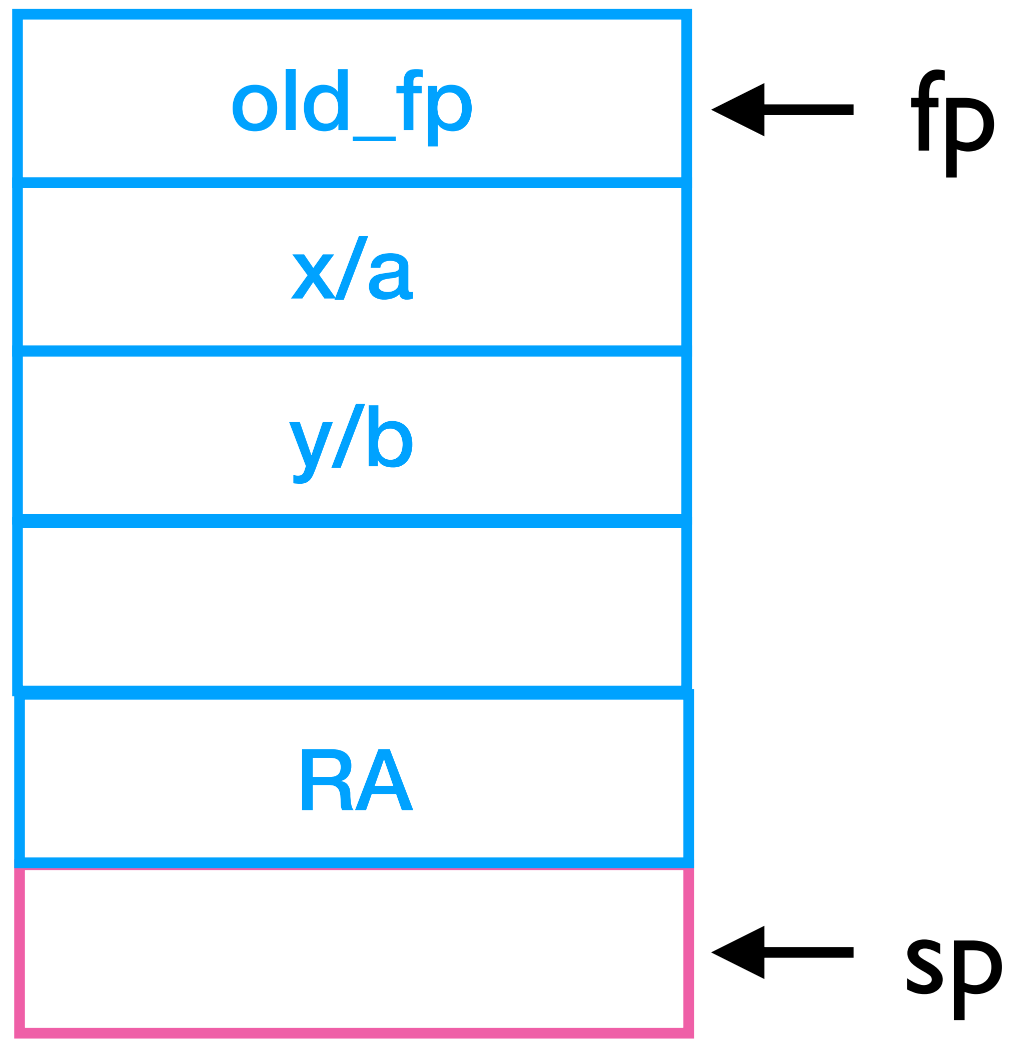
What should happen when foo calls bar?

1. Put Arguments on stack
2. Allocate space for return value
3. Save old return address (return address of foo) on stack
4. Jump to bar (using JR instructor)
5. Save old frame pointer (foo's frame pointer) on stack
6. Set frame pointer to point to top of stack
7. Allocate space for local variables of bar

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



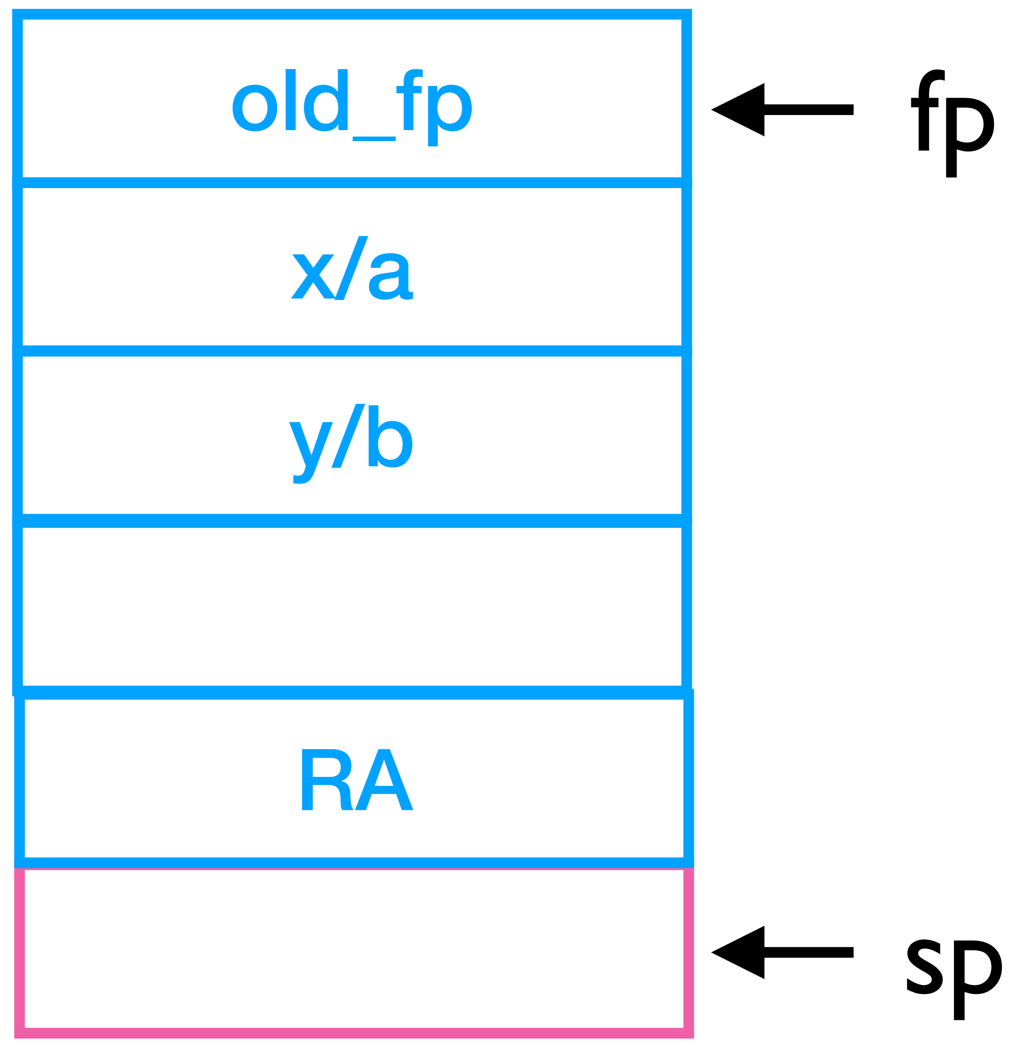
What should happen when foo calls bar?

1. Put Arguments on stack
2. Allocate space for return value
3. Save old return address (return address of foo) on stack
4. Jump to bar (using JR instructor)
5. Save old frame pointer (foo's frame pointer) on stack
6. Set frame pointer to point to top of stack
7. Allocate space for local variables of bar

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



What should happen when foo calls bar?

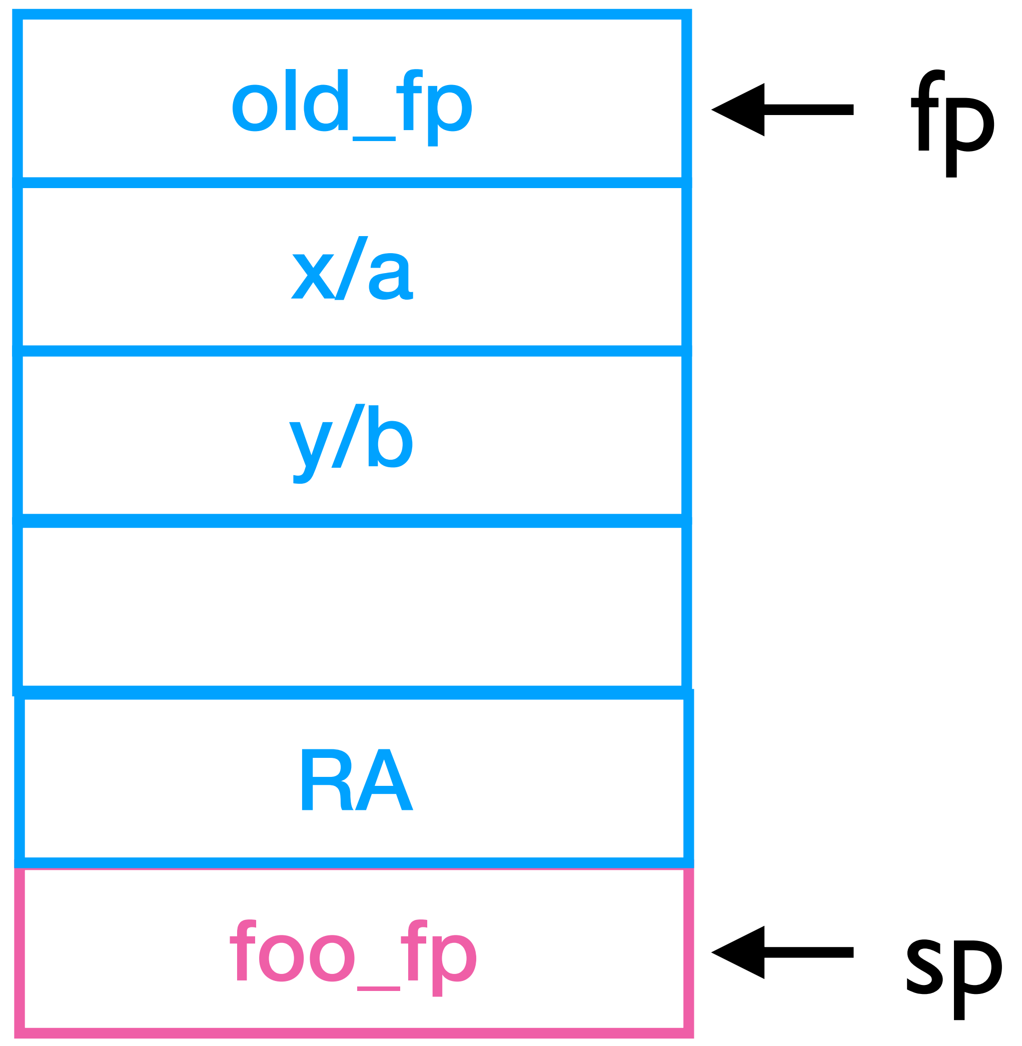
1. Put Arguments on stack
2. Allocate space for return value
3. Save old return address (return address of foo) on stack
4. Jump to bar (using JR instructor)
5. Save old frame pointer (foo's frame pointer) on stack
6. Set frame pointer to point to top of stack
7. Allocate space for local variables of bar

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

bar:
SW FP, 0(SP)
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



What should happen when foo calls bar?

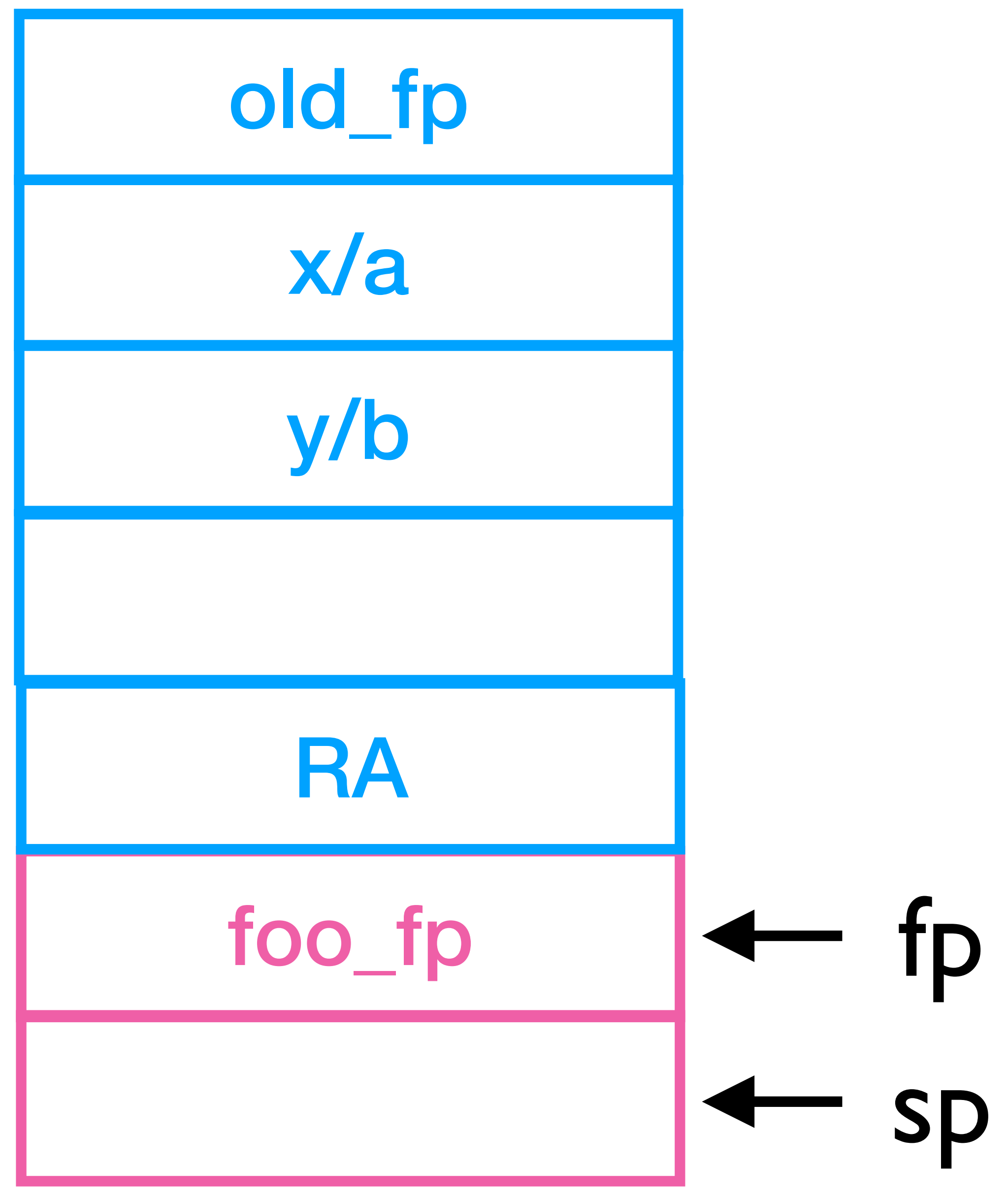
1. Put Arguments on stack
2. Allocate space for return value
3. Save old return address (return address of foo) on stack
4. Jump to bar (using JR instructor)
5. Save old frame pointer (foo's frame pointer) on stack
6. Set frame pointer to point to top of stack
7. Allocate space for local variables of bar

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

bar:
SW FP, 0(SP)
MV FP, SP
SUBI SP, SP, 4
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



What should happen when foo calls bar?

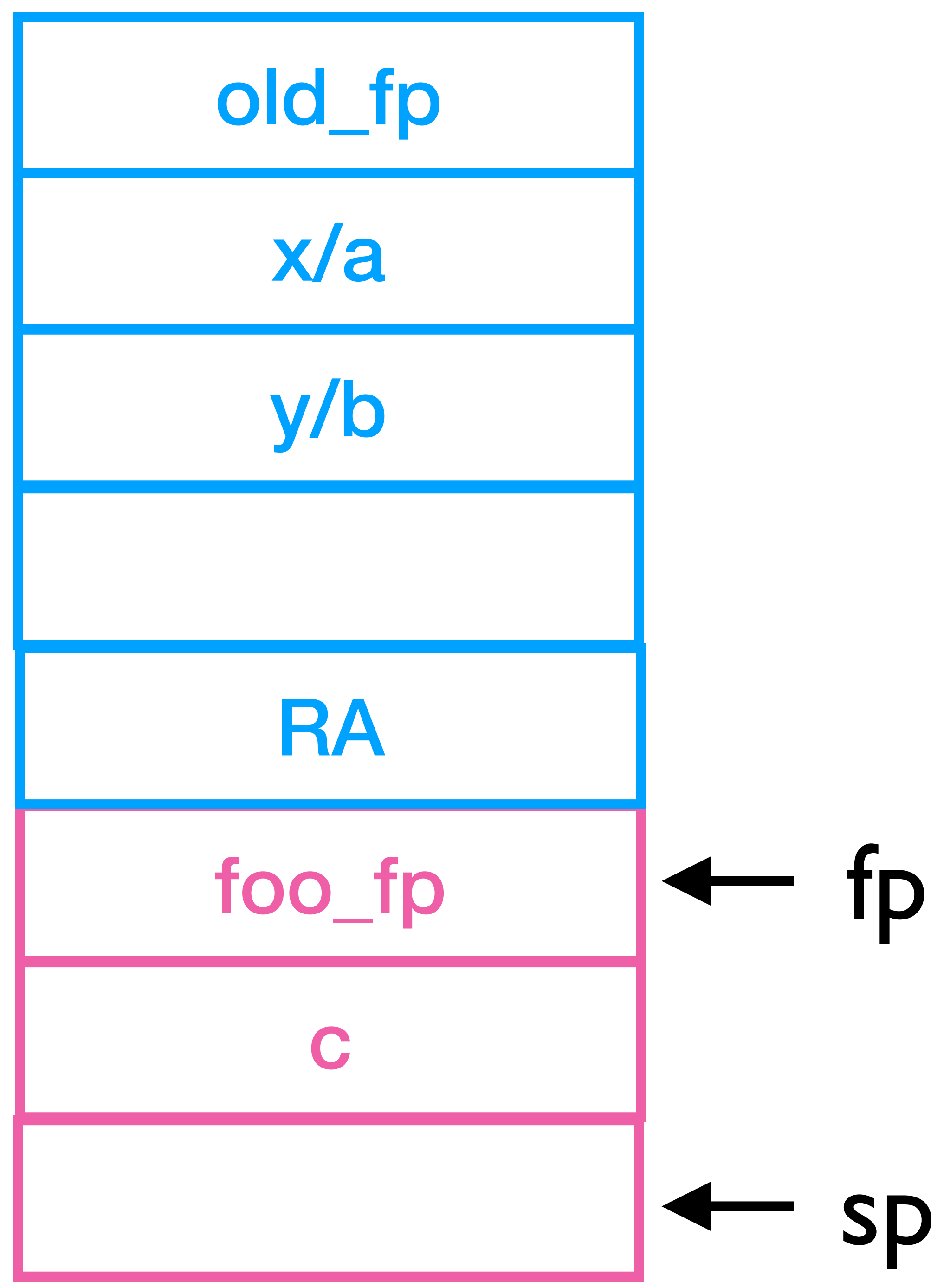
1. Put Arguments on stack
2. Allocate space for return value
3. Save old return address (return address of foo) on stack
4. Jump to bar (using JR instructor)
5. Save old frame pointer (foo's frame pointer) on stack
6. Set frame pointer to point to top of stack
7. Allocate space for local variables of bar

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

bar:
SW FP, 0(SP)
MV FP, SP
SUBI SP, SP, 4
SUBI SP, SP, 4
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



What about returning from bar?

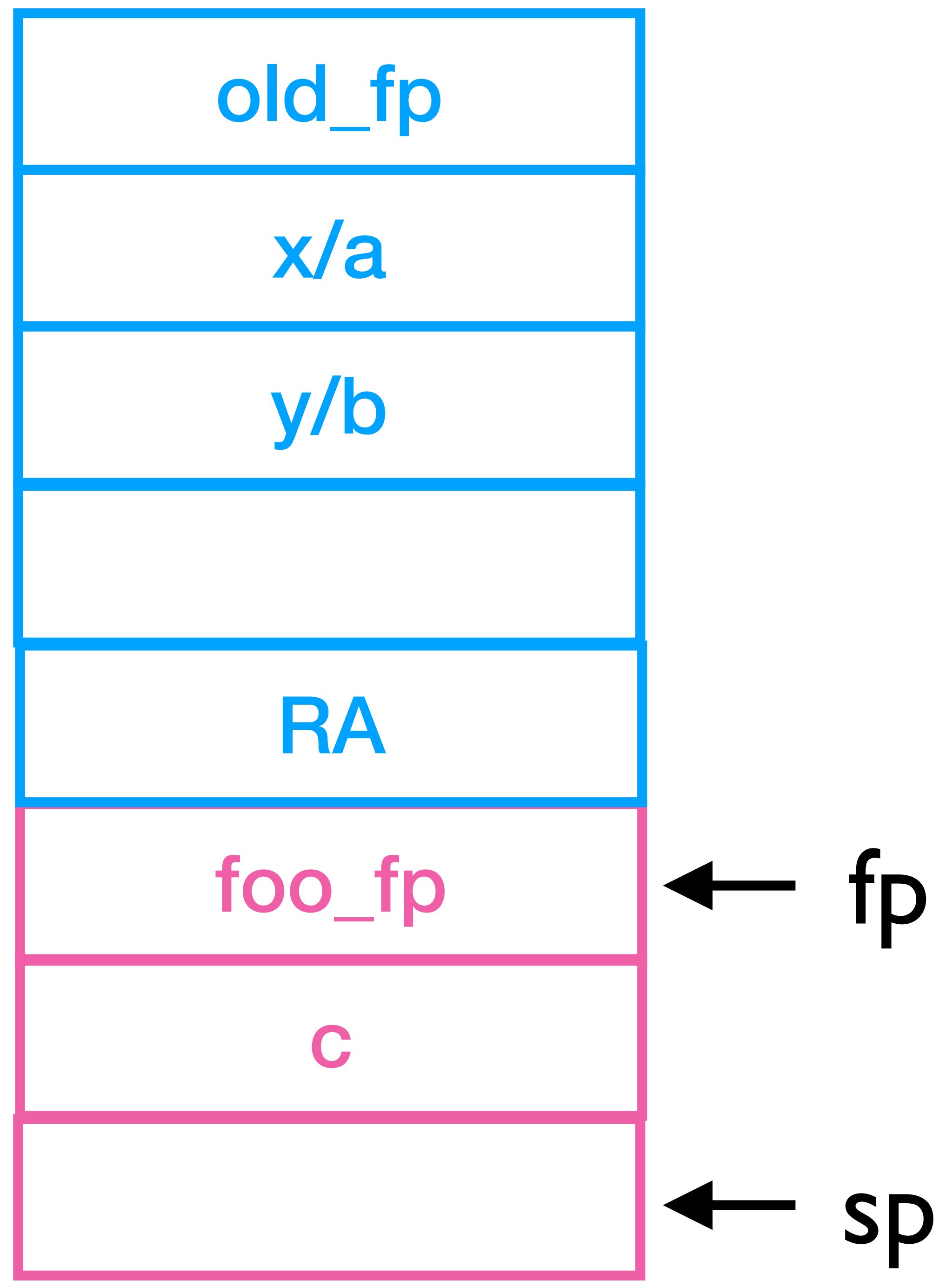
1. Put return value in appropriate location
2. Reset stack pointer to top of foo's activation record
3. Reset frame pointer back to old location
4. Return to foo

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

bar:
SW FP, 0(SP)
MV FP, SP
SUBI SP, SP, 4
SUBI SP, SP, 4
...
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



What about returning from bar?

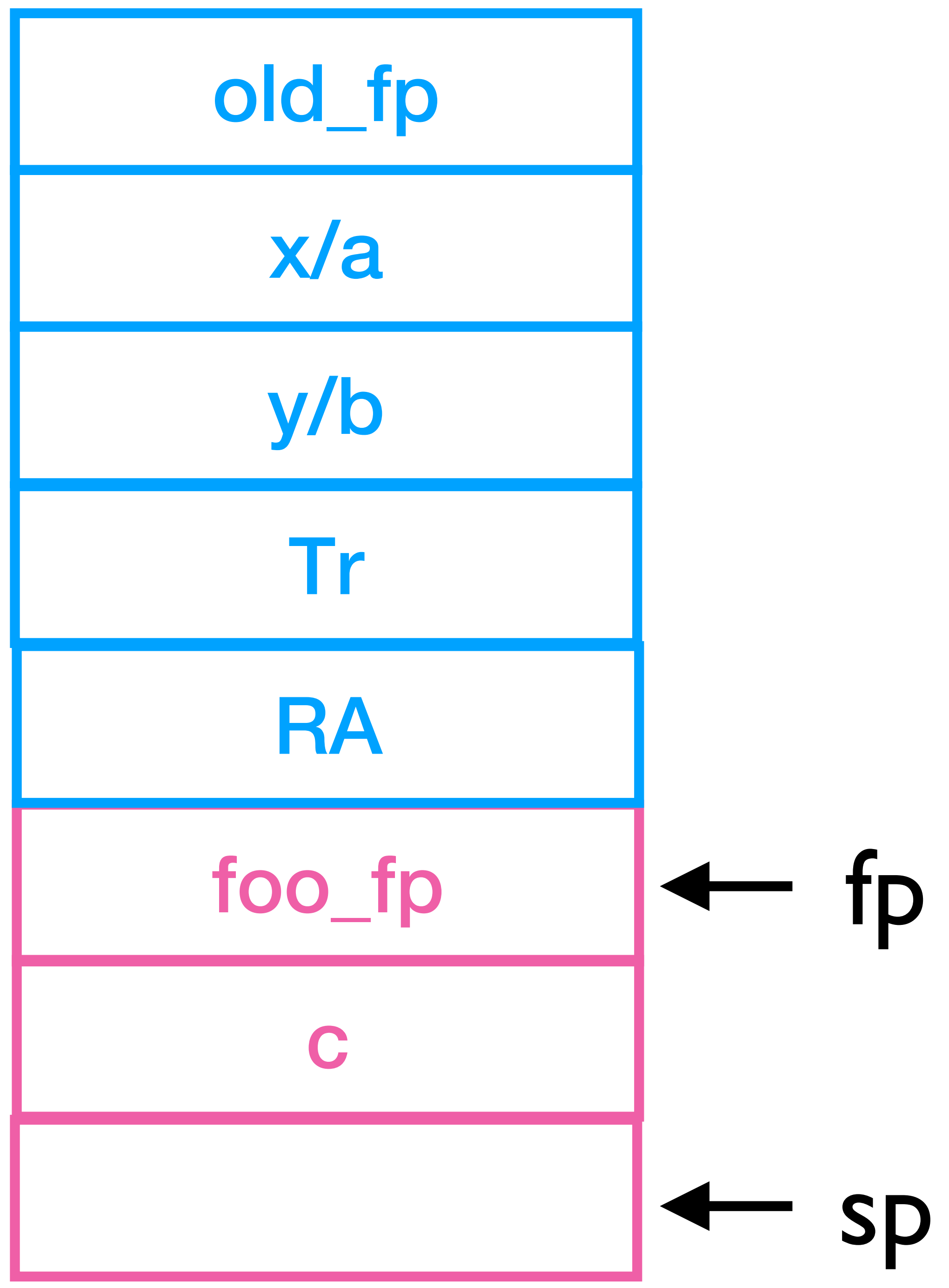
1. Put return value in appropriate location
2. Reset stack pointer to top of foo's activation record
3. Reset frame pointer back to old location
4. Return to foo

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

bar:
SW FP, 0(SP)
MV FP, SP
SUBI SP, SP, 4
SUBI SP, SP, 4
...
SW Tr, 8(FP)
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



What about returning from bar?

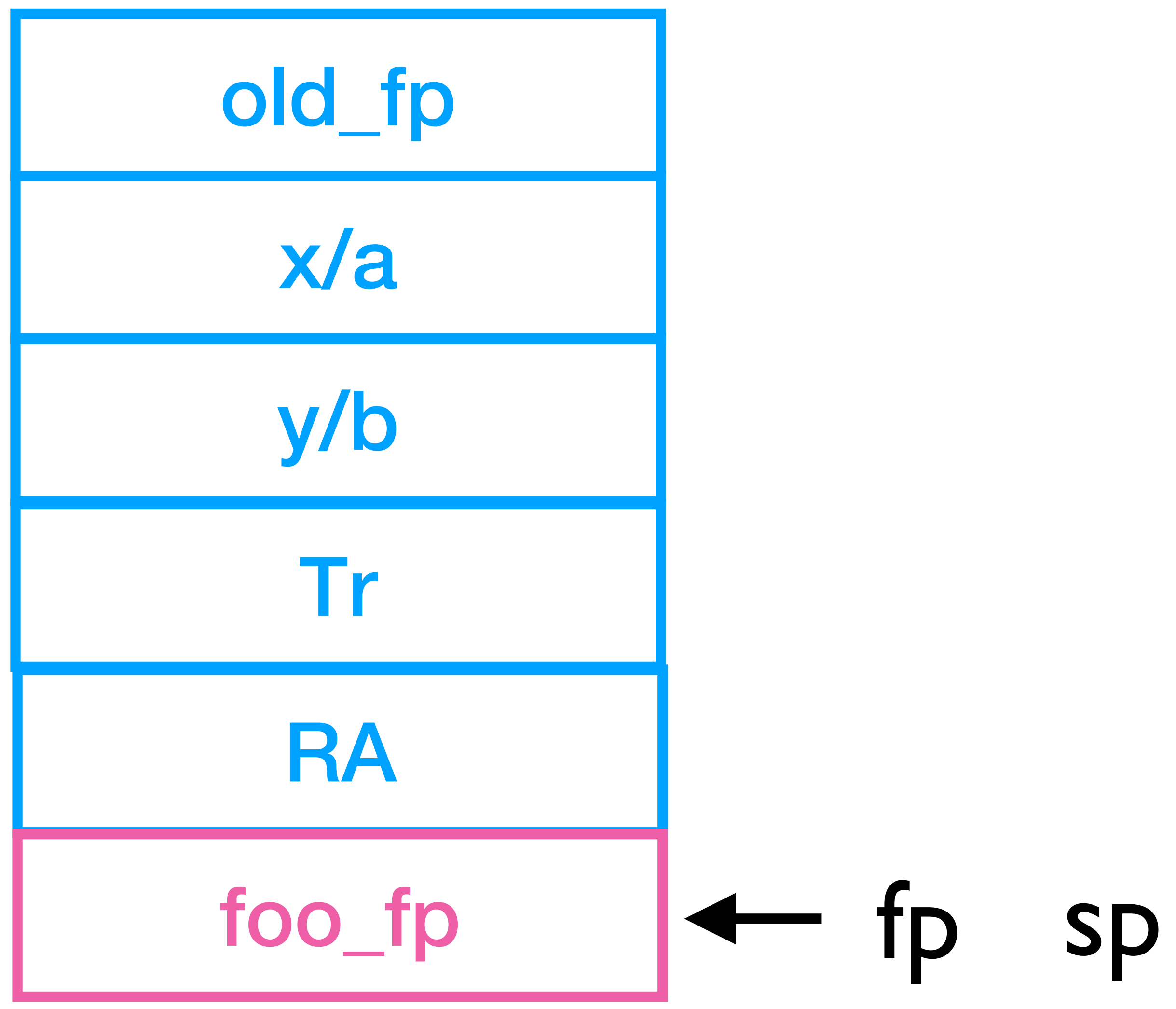
1. Put return value in appropriate location
2. Reset stack pointer to top of foo's activation record
3. Reset frame pointer back to old location
4. Return to foo

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

bar:
SW FP, 0(SP)
MV FP, SP
SUBI SP, SP, 4
SUBI SP, SP, 4
...
SW Tr, 8(FP)
MV SP, FP
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



What about returning from bar?

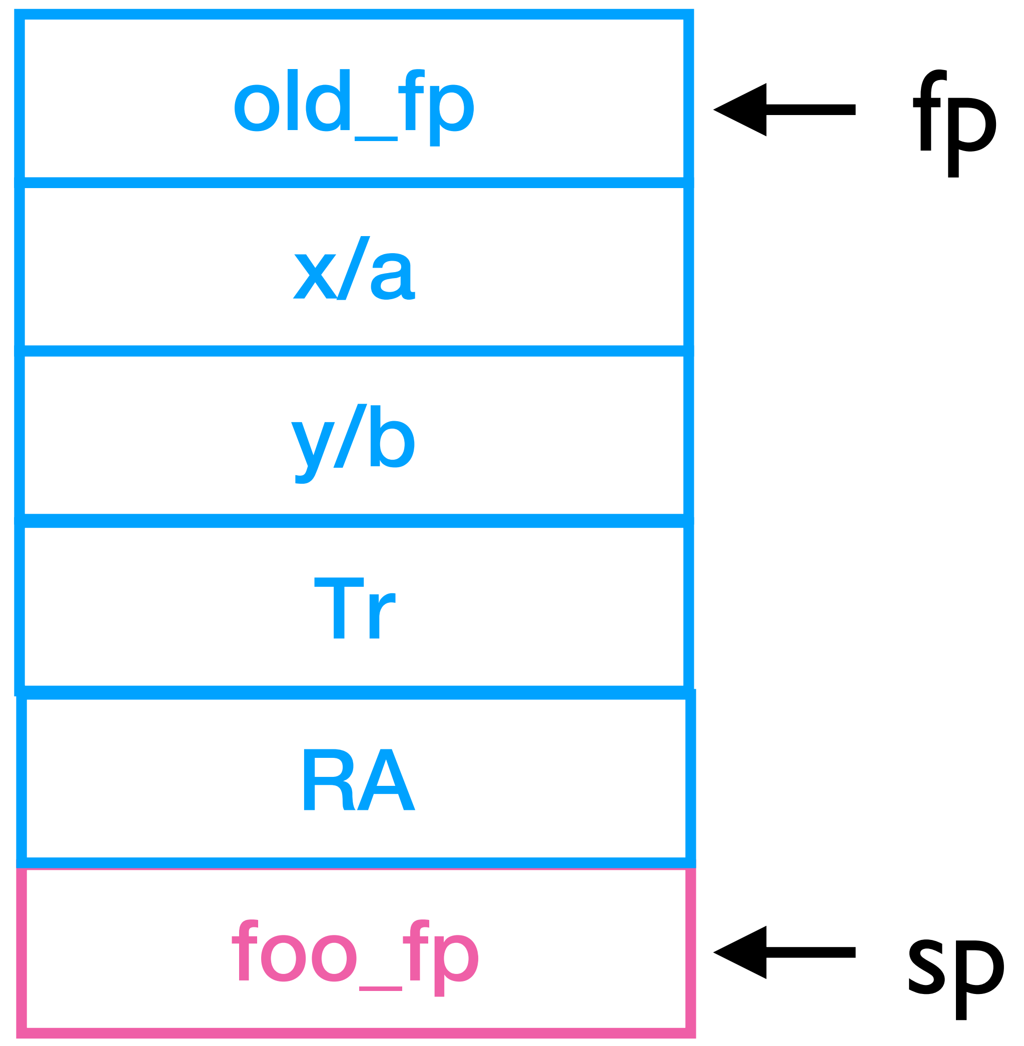
1. Put return value in appropriate location
2. Reset stack pointer to top of foo's activation record
3. Reset frame pointer back to old location
4. Return to foo

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

bar:
SW FP, 0(SP)
MV FP, SP
SUBI SP, SP, 4
SUBI SP, SP, 4
...
SW Tr, 8(FP)
MV SP, FP
LW FP, 0(FP)
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



What about returning from bar?

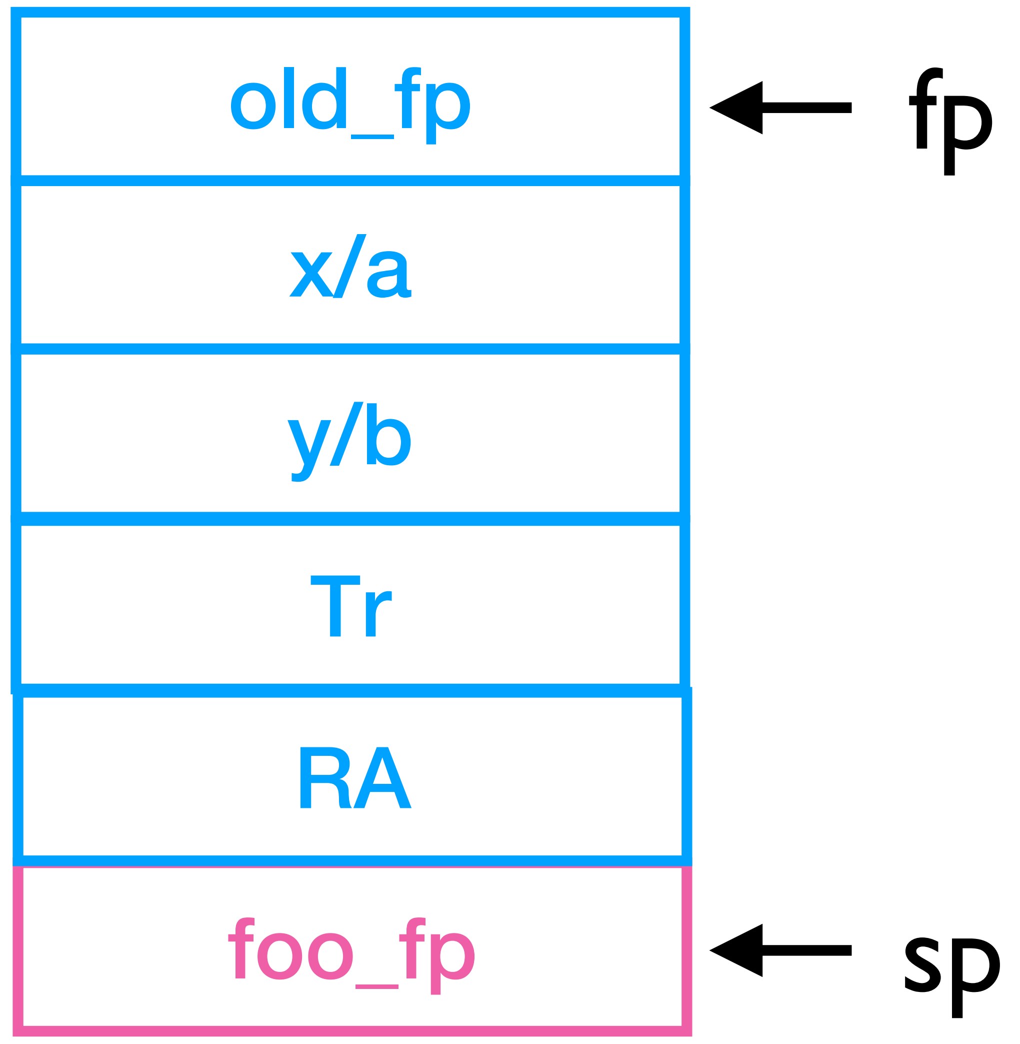
1. Put return value in appropriate location
2. Reset stack pointer to top of foo's activation record
3. Reset frame pointer back to old location
4. Return to foo

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

bar:
SW FP, 0(SP)
MV FP, SP
SUBI SP, SP, 4
SUBI SP, SP, 4
...
SW Tr, 8(FP)
MV SP, FP
LW FP, 0(FP)
RET
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



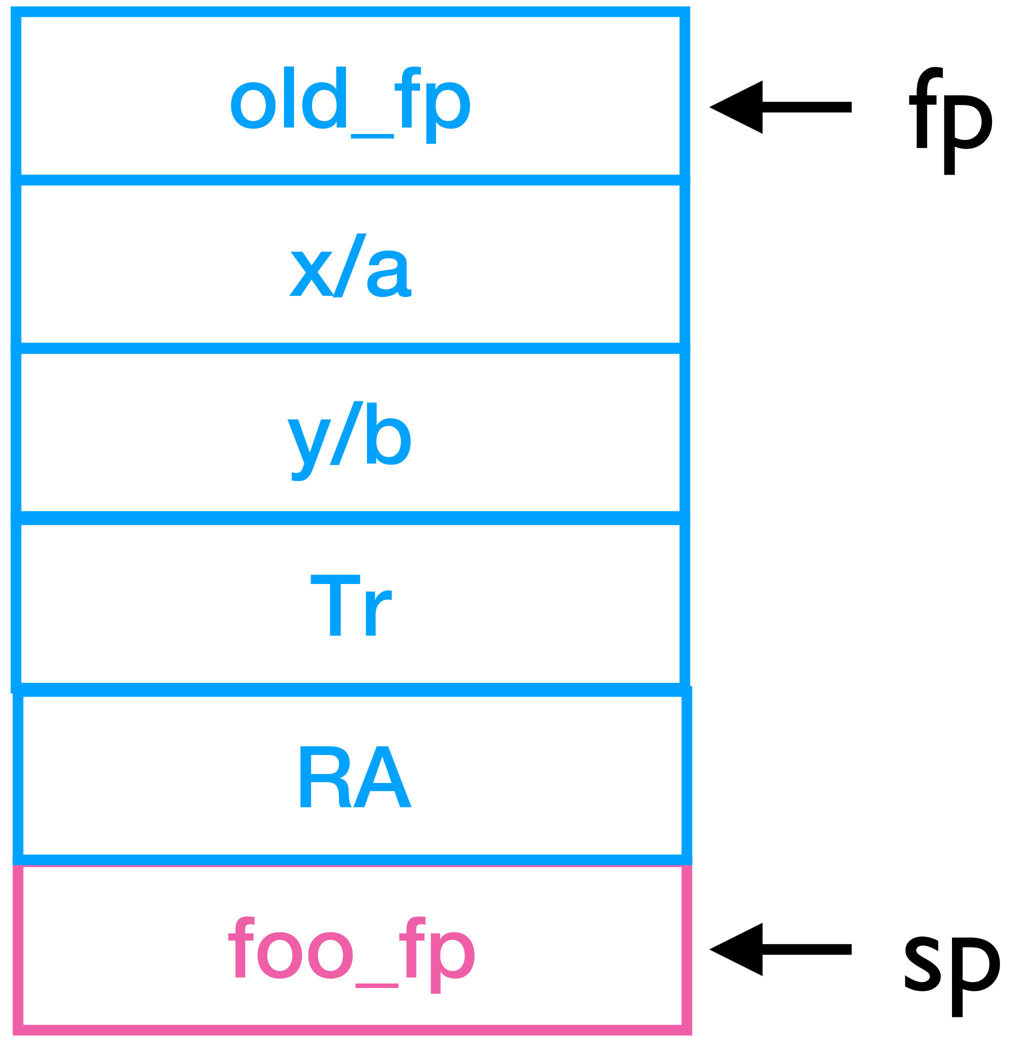
Now what does foo do?

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```

1. Restore old return address
2. Retrieve return value from stack
3. Remove arguments and return value from stack



Now what does foo do?

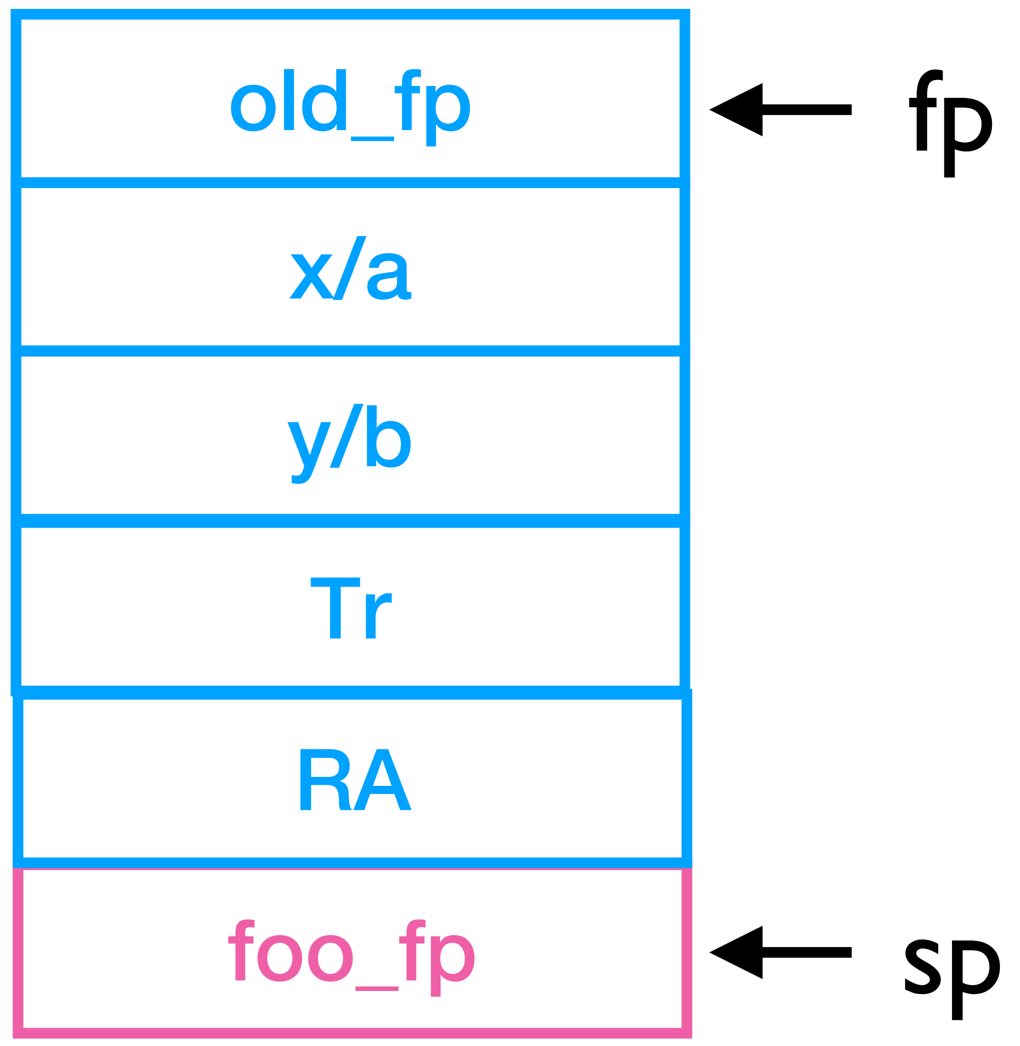
1. Restore old return address
2. Retrieve return value from stack
3. Remove arguments and return value from stack

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

LW RA, 4(SP)
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



Now what does foo do?

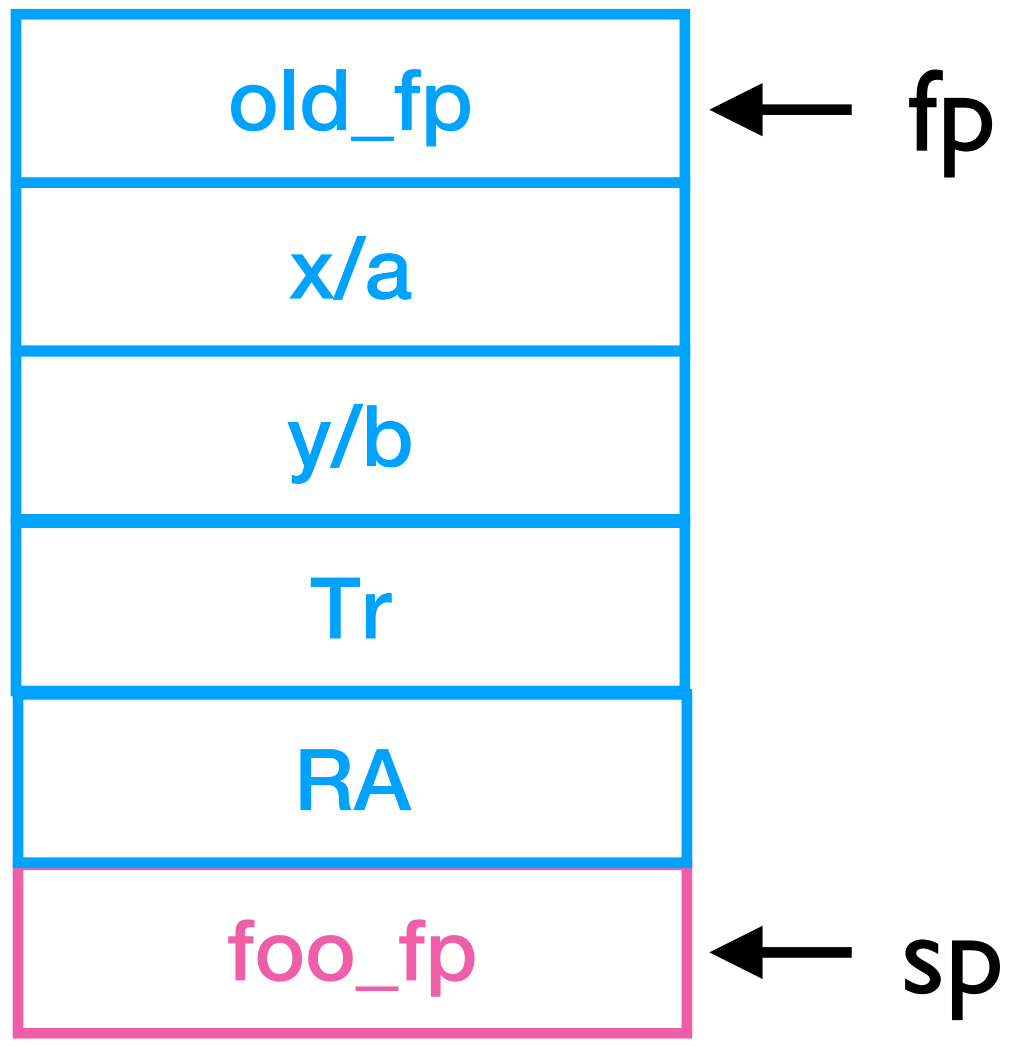
1. Restore old return address
2. Retrieve return value from stack
3. Remove arguments and return value from stack

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

LW RA, 4(SP)
LW Tr, 8(SP)
```

```
int foo() {
   ...
   z = bar(x, y);
}

int bar(int a, int b) {
   int c;
}
```



Now what does foo do?

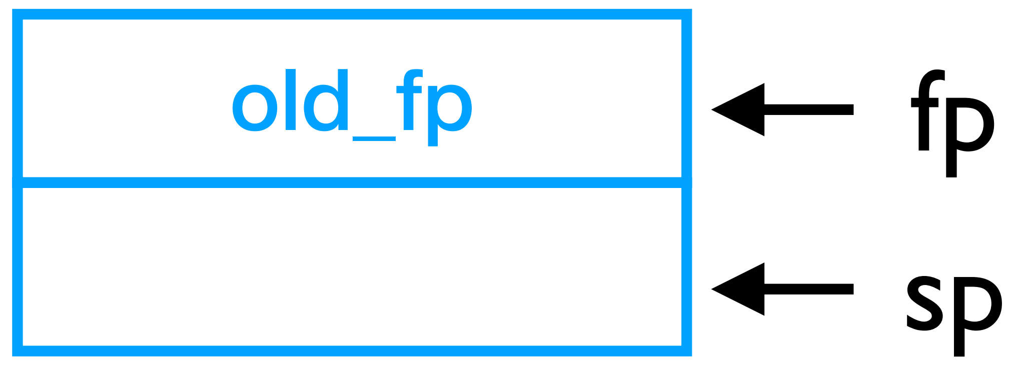
1. Restore old return address
2. Retrieve return value from stack
3. Remove arguments and return value from stack

```
SW Tx, 0(SP)
SW Ty, -4(SP)
SUBI SP, SP, 8
SUBI SP, SP, 4
SW RA, 0(SP)
SUBI SP, SP, 4
JR bar

LW RA, 4(SP)
LW Tr, 8(SP)
ADDI SP, SP, 16
```

```
int foo() {
  ...
  z = bar(x, y);
}

int bar(int a, int b) {
  int c;
}
```



next: what if foo and bar reuse registers?