Stack organization

how to lay out the program stack

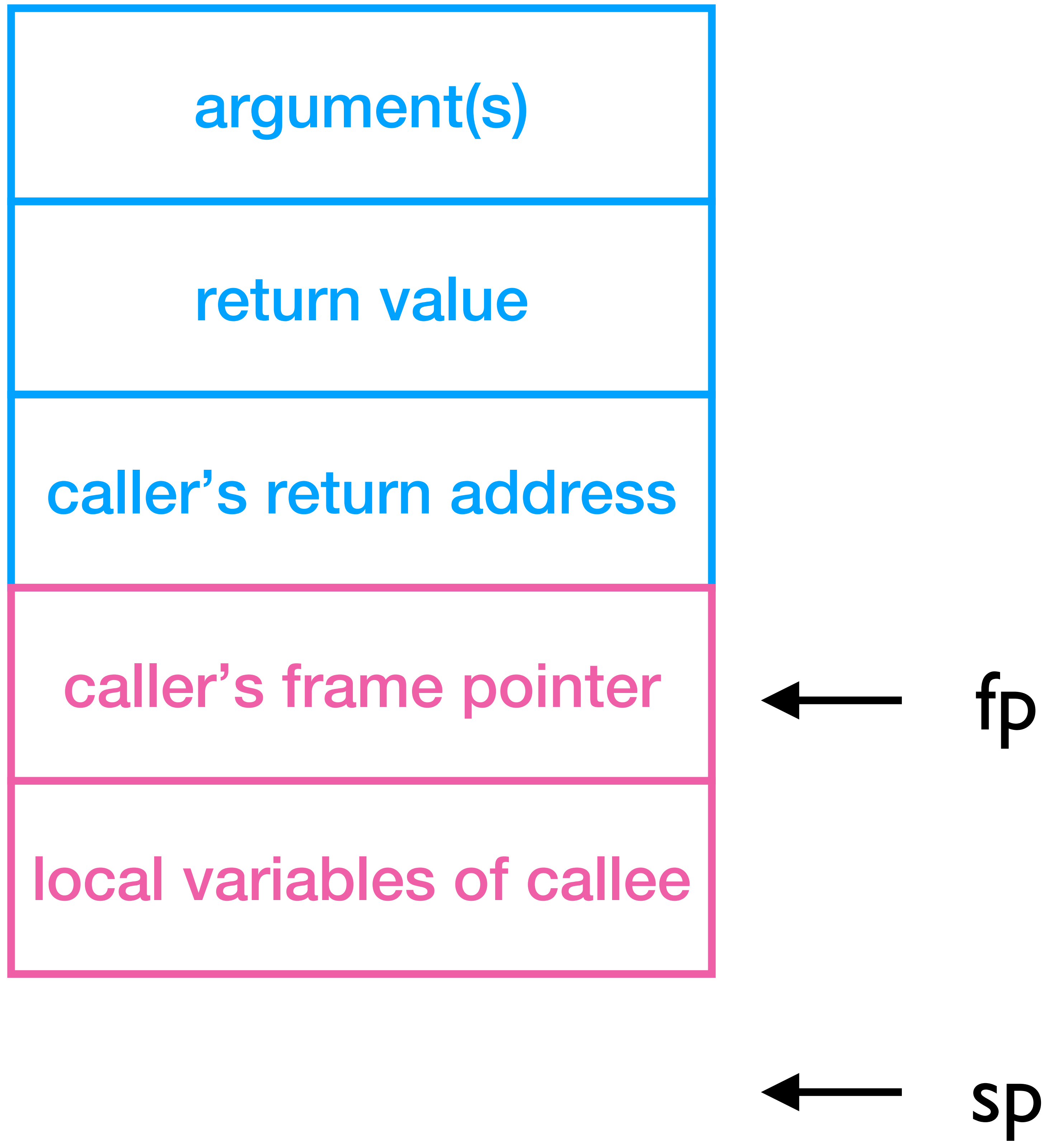
- The **stack** is the primary memory area for managing the interaction between functions
  - Arguments passed from caller to callee (though this may happen in registers as an optimization)
  - Return values passed from callee to caller (though this may happen in registers as an optimization)
  - Local variables for each function
  - Saved registers for each function (caller saves registers callee might use, and callee saves registers caller might need)
  - “Spilled” registers

key mechanisms for program stack

- Reserved area of memory
  - Different area than program text, globals, heap
  - In Risc-V, stack grows **down**: pushing an element onto the stack puts it at a lower address than the current top of the stack
- Two pointers
  - **Stack pointer** (**sp**): points to the top of the stack
    - In our approach, sp will point to the next *open* spot on the stack
    - Pushing on the stack: store to sp, decrement sp by appropriate amount
  - **Frame pointer** (**fp**): points to the base of the activation record
    - Locations of other parts of the stack are relative to fp
    - Optimization: can eliminate fp, but makes code generation more complicated (how? why?)

Activation record

- What does an activation record look like for a function?
- Caller places arguments and return value on stack
- Caller places its return address (where it should return to) on stack
  - Why? Register holding this address will be overwritten when invoking callee
- Callee saves old frame pointer on stack, then moves frame pointer to point to the base of its record
- Callee creates space for its local variables on stack


argument(s)
return value
caller’s return address
caller’s frame pointer
local variables of callee
fp
sp