Lecture 5

CIS 4521/5521: COMPILERS

Announcements

- HW2: X86lite
  - Due: Weds. Feb. 12th at 10:00pm
  - Pair-programming
  - Sign up for teams via github classroom

  - **Please get started!** (I can see who has cloned the git repo!)

- Note: clone the project with `` `--recurse-submodules` `` flag
  - There is a shared, public git submodule to which you will need to push test cases.
  - We may need to adjust permissions on github to make this work, so please accept the invitation to join the `upenn-cis5521` organization.

Plan

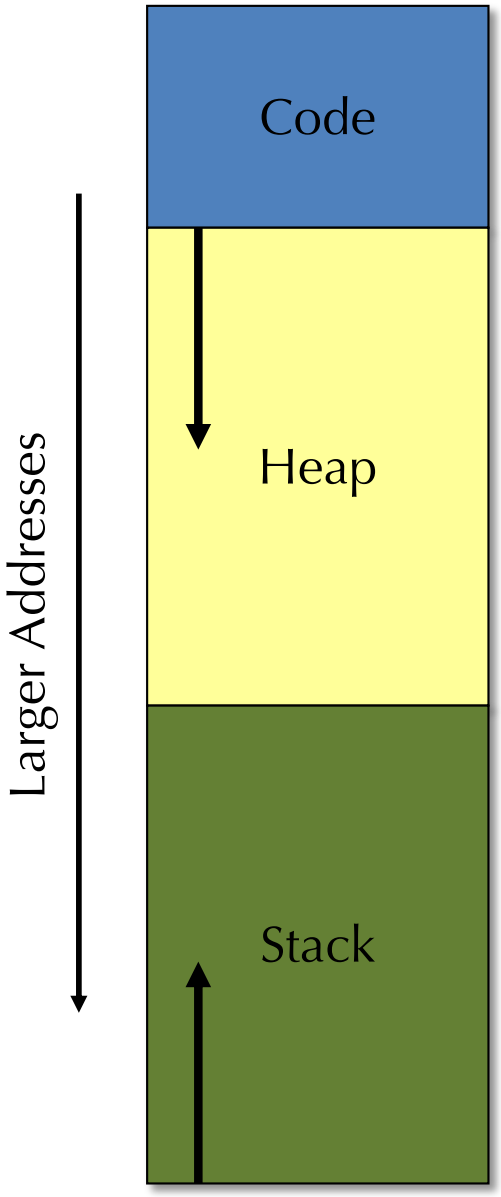
- x86 Programming
  - x86 calling conventions
- Generating x86_64 assembly by direct translation
  - Strategy 1: use the stack
  - Strategy 2: use a stack-based IR
- Along the way:
  - simple static analysis
  - translation invariants
  - more about language runtimes
- Intermediate Representations

PROGRAMMING IN X86LITE

3 parts of the C memory model

- The code & data (or "text") segment
  - contains compiled code, constant strings, etc.
- The Heap
  - Stores dynamically allocated objects
  - Allocated via "`malloc`"
  - Deallocated via "`free`"
  - managed by C runtime system
- The Stack
  - Stores local variables
  - Stores the return address of a function

- In practice, most languages use this model.



Local/Temporary Variable Storage

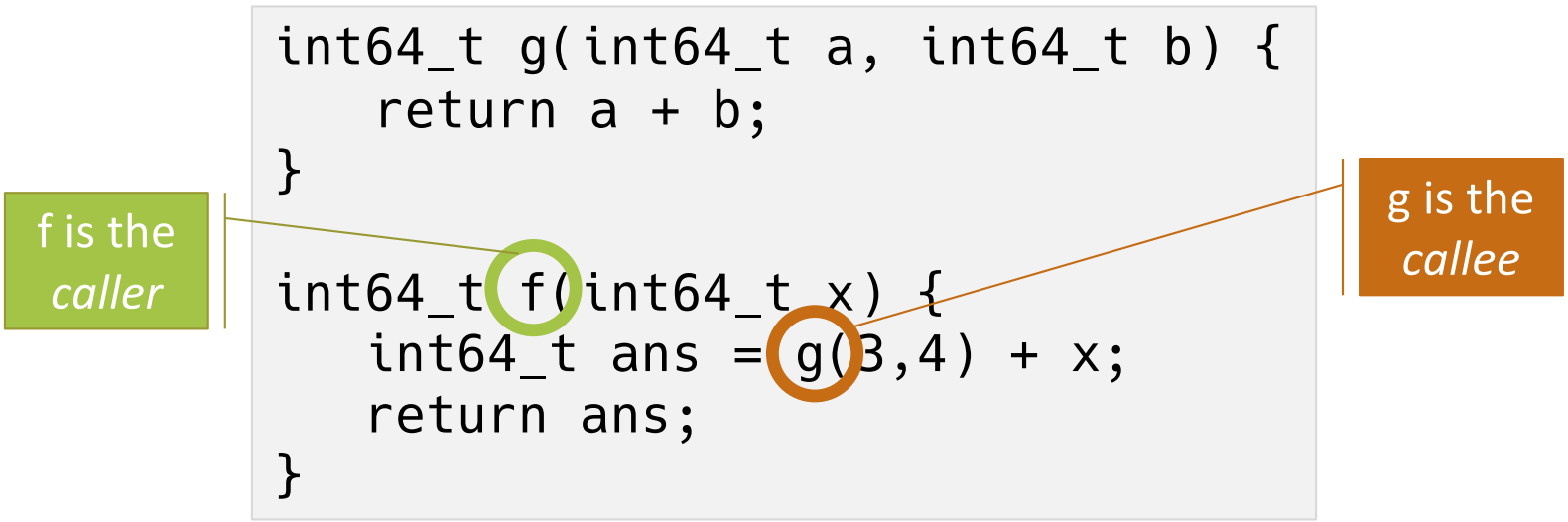
- Need space to store:
  - Global variables
  - Values passed as arguments to procedures
  - Local variables (either defined in the source program or introduced by the compiler)

- Processors provide two options
  - Registers: fast, small size (64 bits), very limited number
  - Memory: slow, very large amount of space (2 GB)
    - caching important

- In practice on X86:
  - Registers are limited (and have restrictions)
  - Divide memory into regions including the *stack* and the *heap*

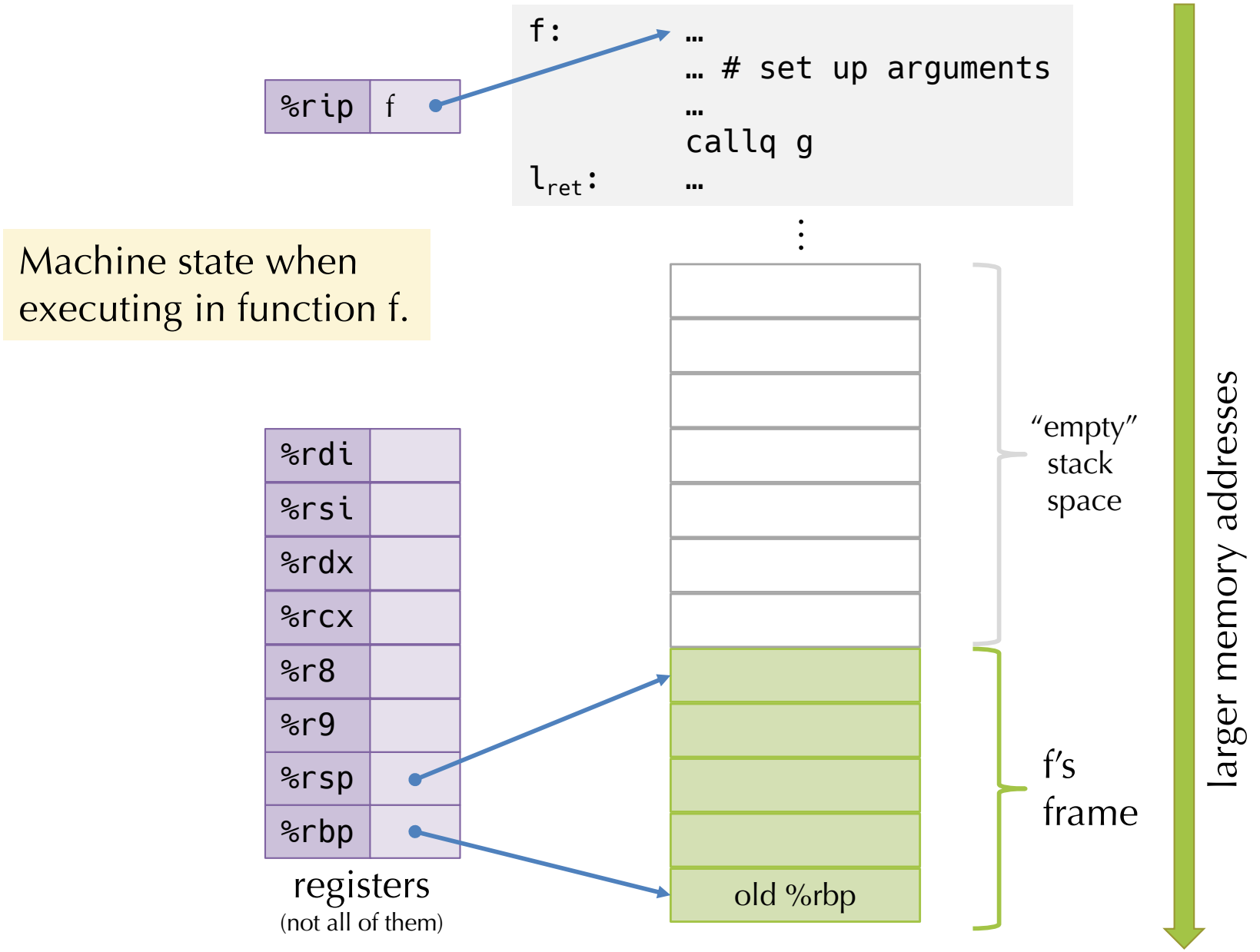
Calling Conventions

- Specify the locations (*e.g.*, register or stack) of arguments passed to a function and returned by the function



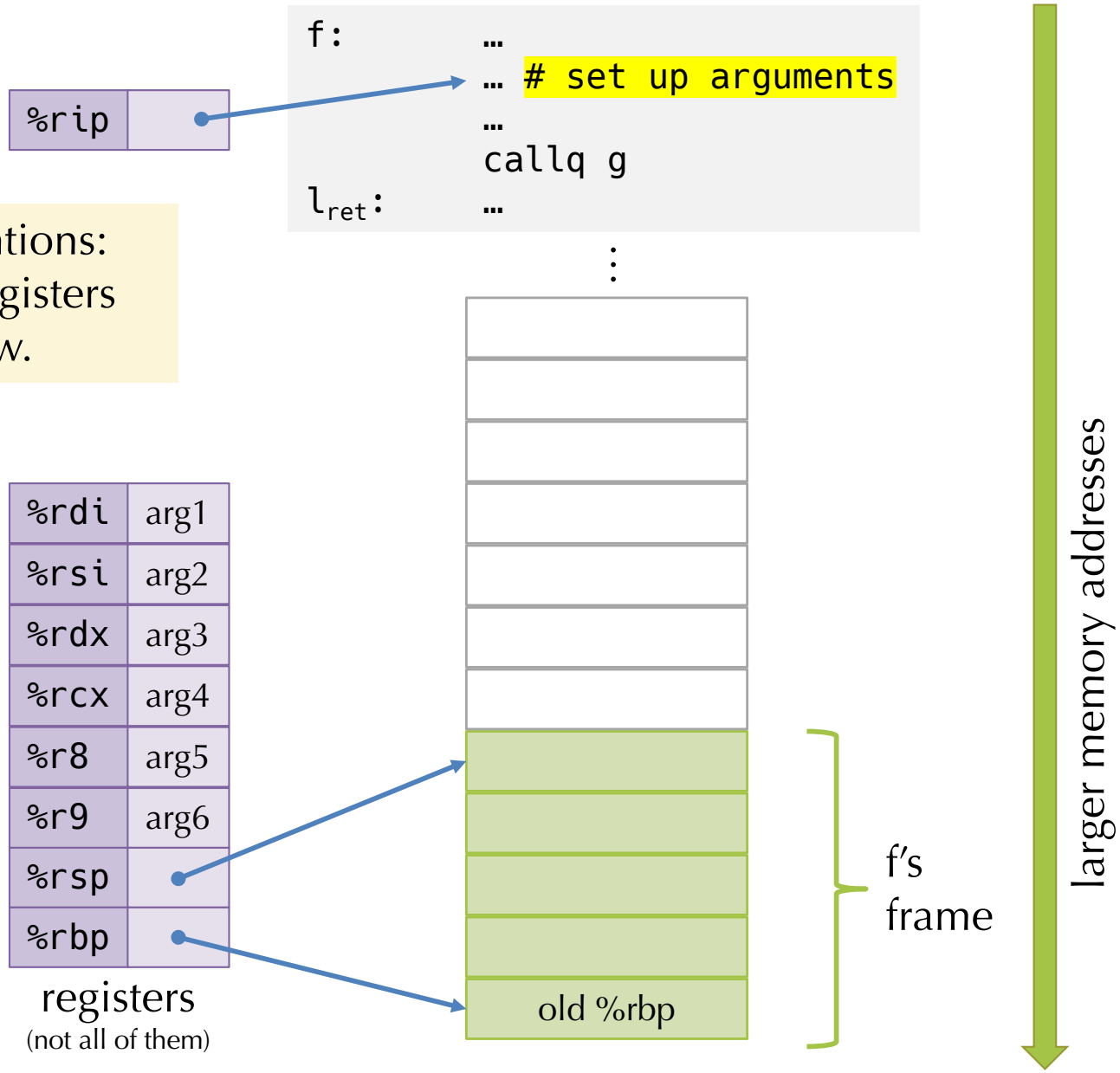
- Designate registers either:
  - Caller Save – *e.g.*, freely usable by the called code
  - Callee Save – *e.g.*, must be restored by the called code
- Define the protocol for deallocating stack-allocated arguments
  - Caller cleans up
  - Callee cleans up (makes supporting variable number of arguments harder)

x64 Calling Conventions: Caller Protocol



x64 Calling Conventions: Caller Protocol

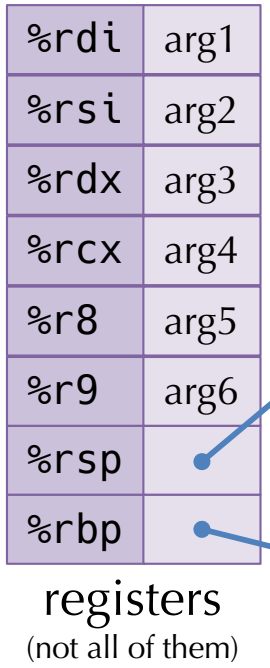
Calling conventions:
args 1…6 in registers
as shown below.

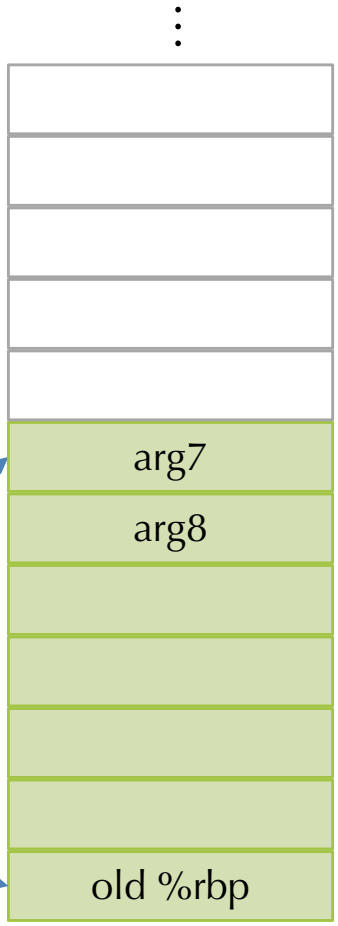


x64 Calling Conventions: Caller Protocol

```
f:      …
        … # set up arguments
        …
        callq g
l_ret:  …
```

args > 6 pushed onto the stack (from right to left)
Note: %rsp changes

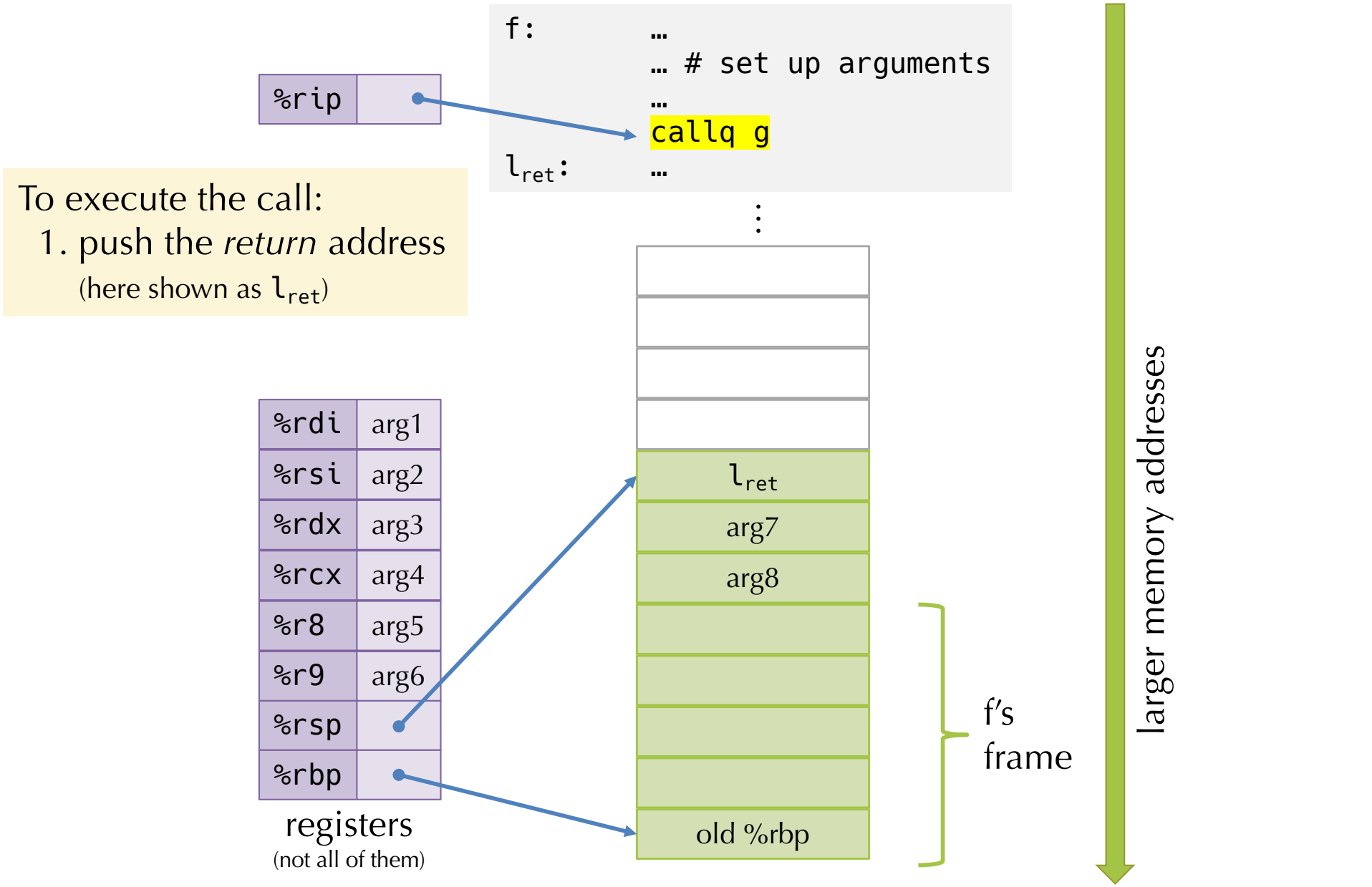




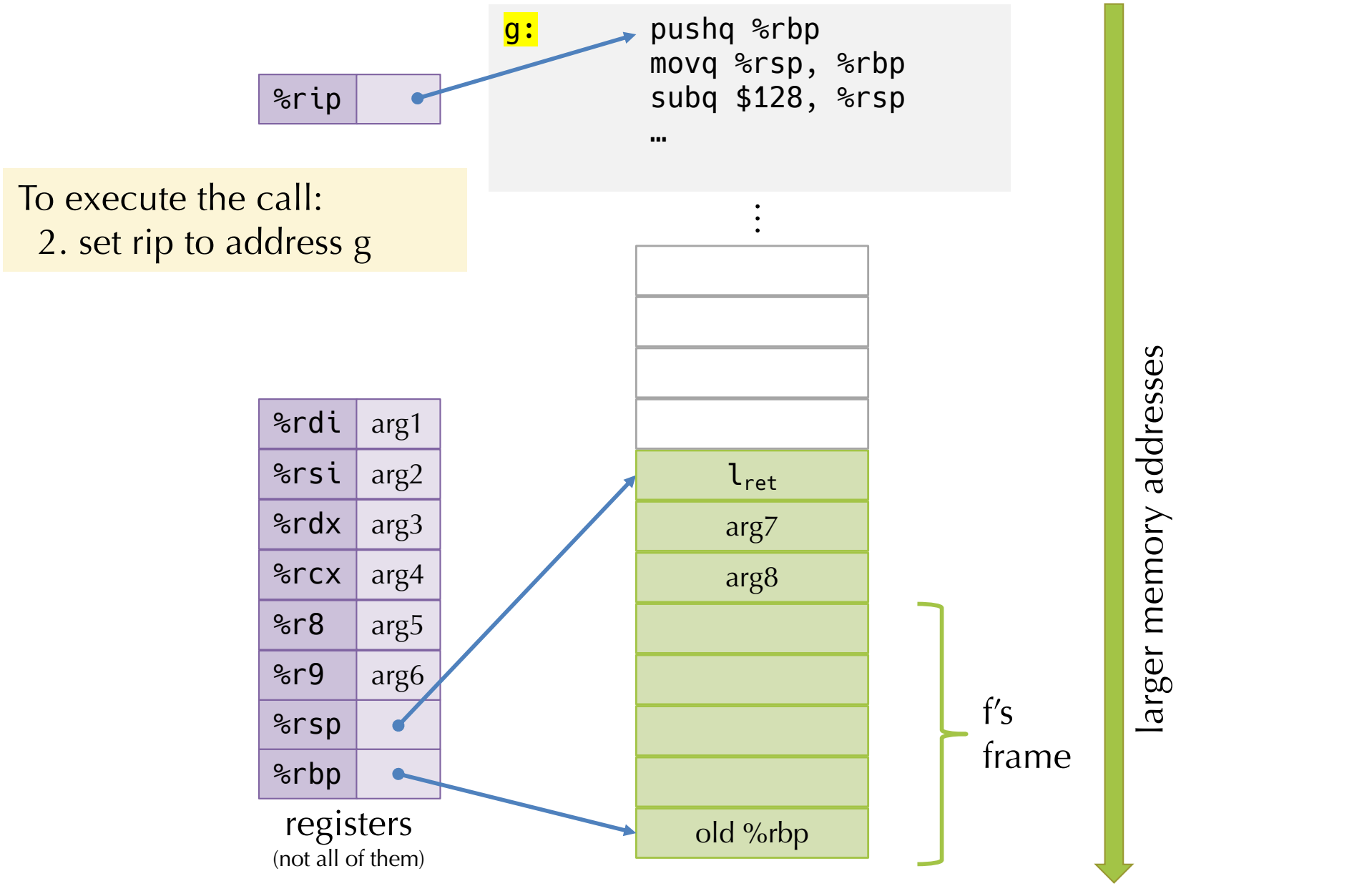
f's frame

larger memory addresses

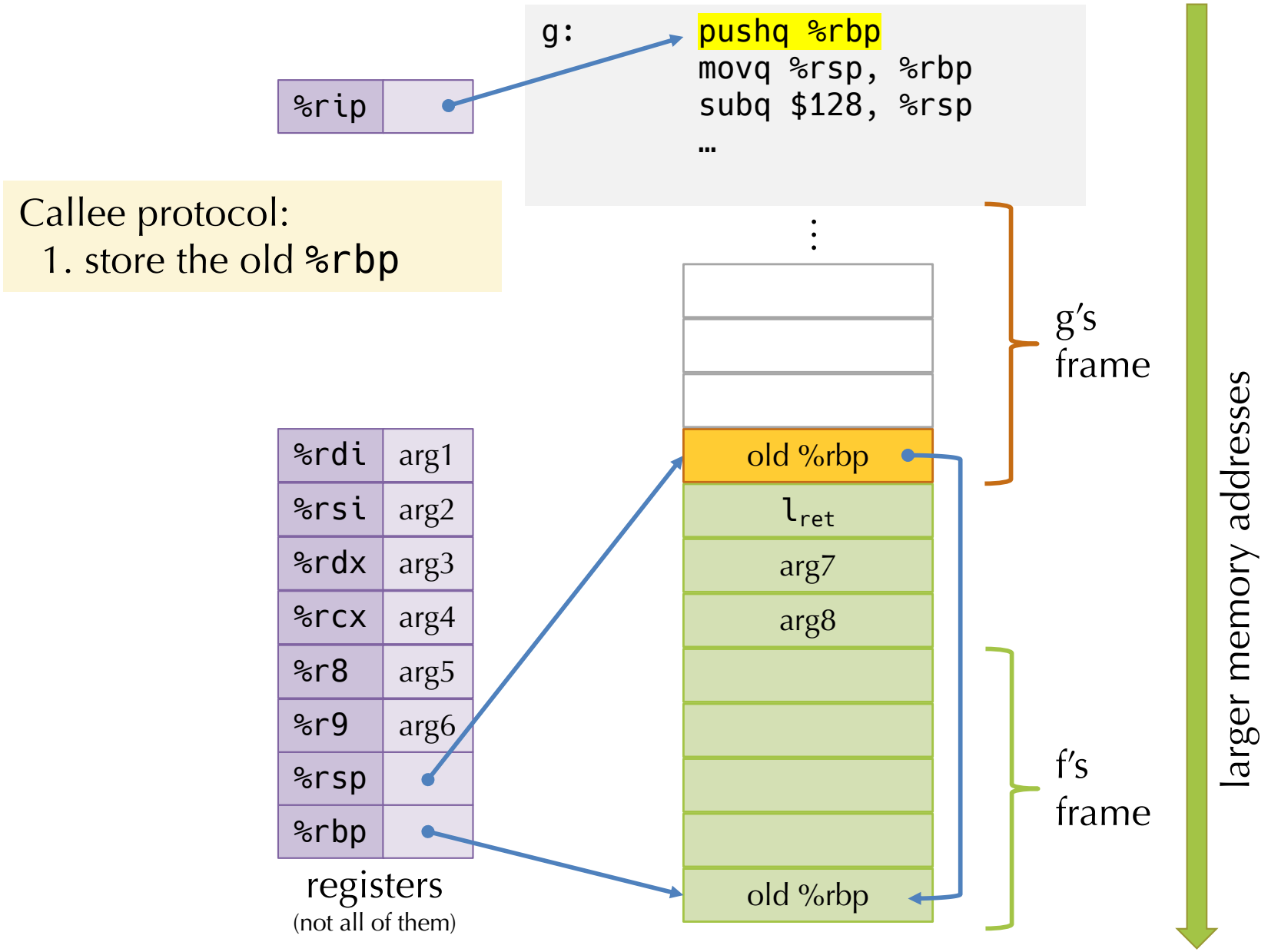
call instruction



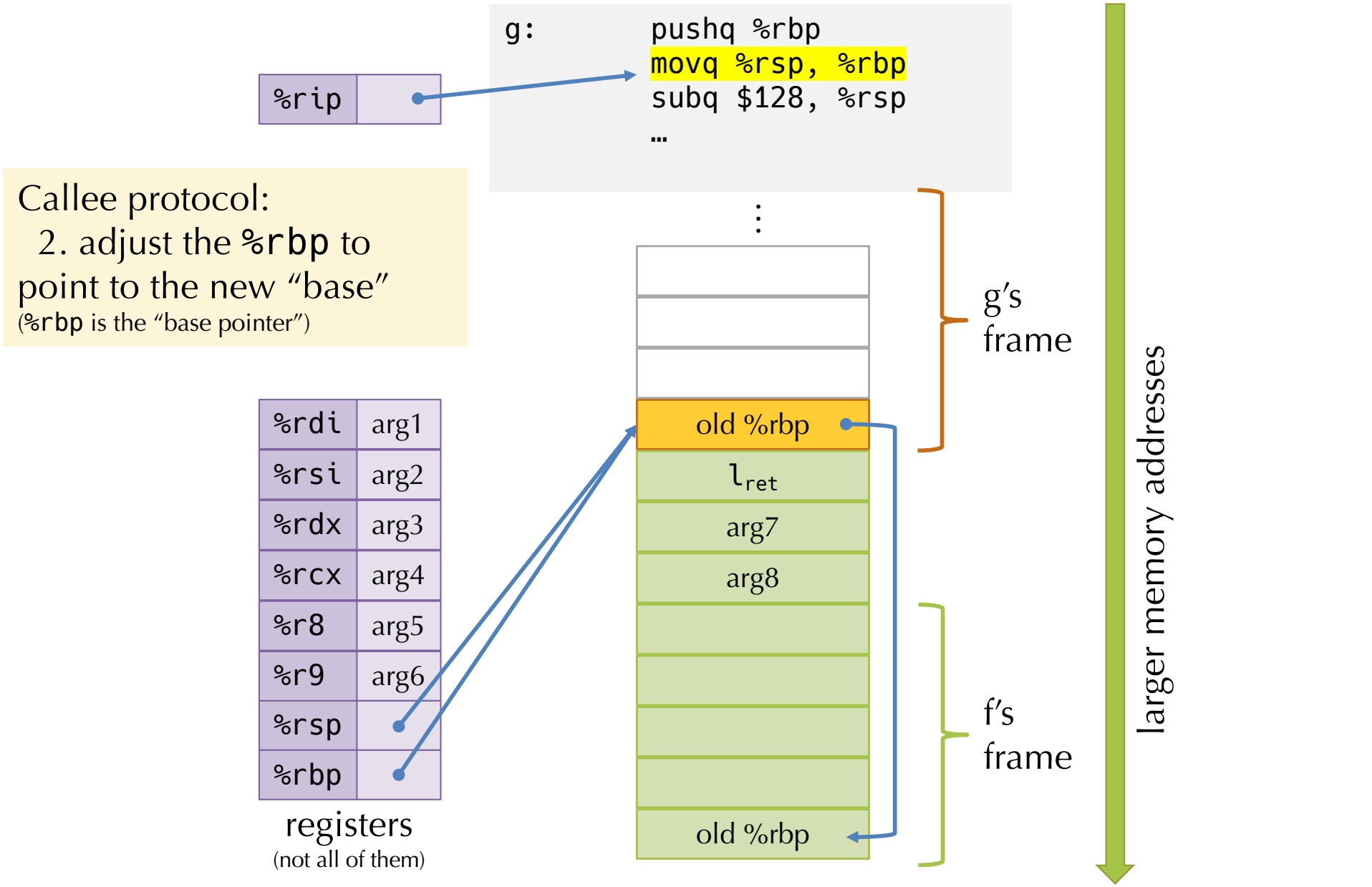
call instruction



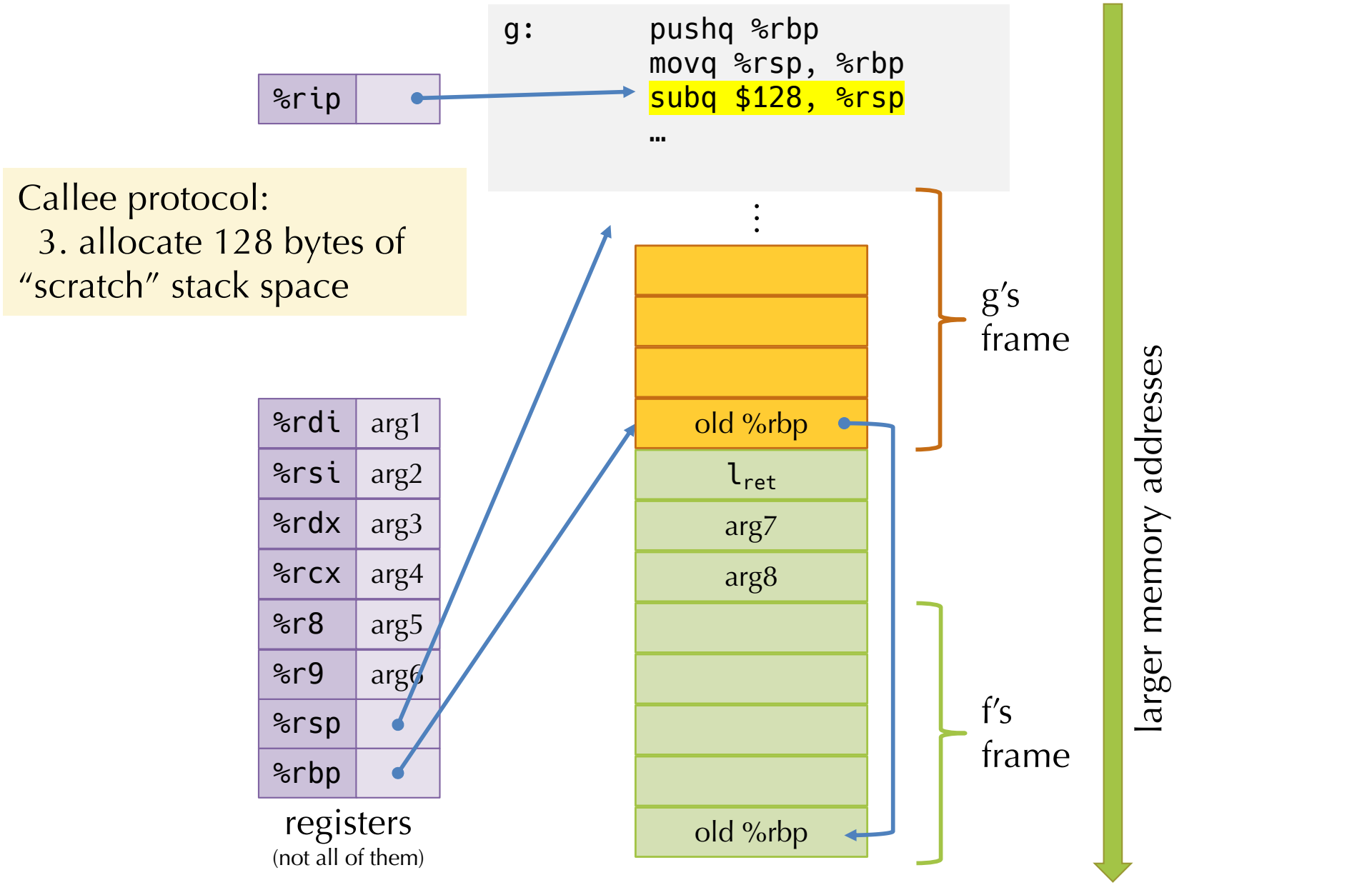
callee function prologue



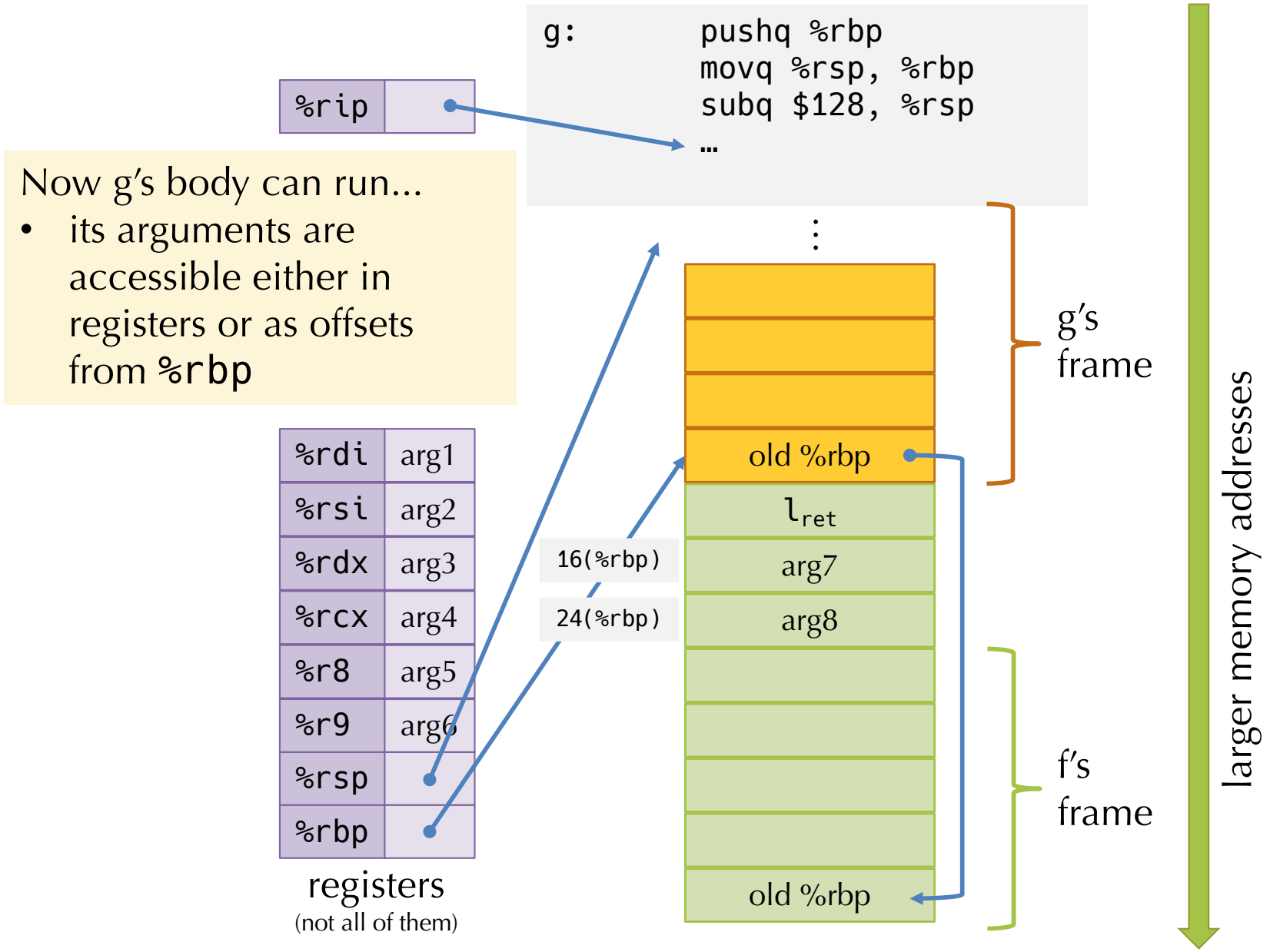
callee function prologue



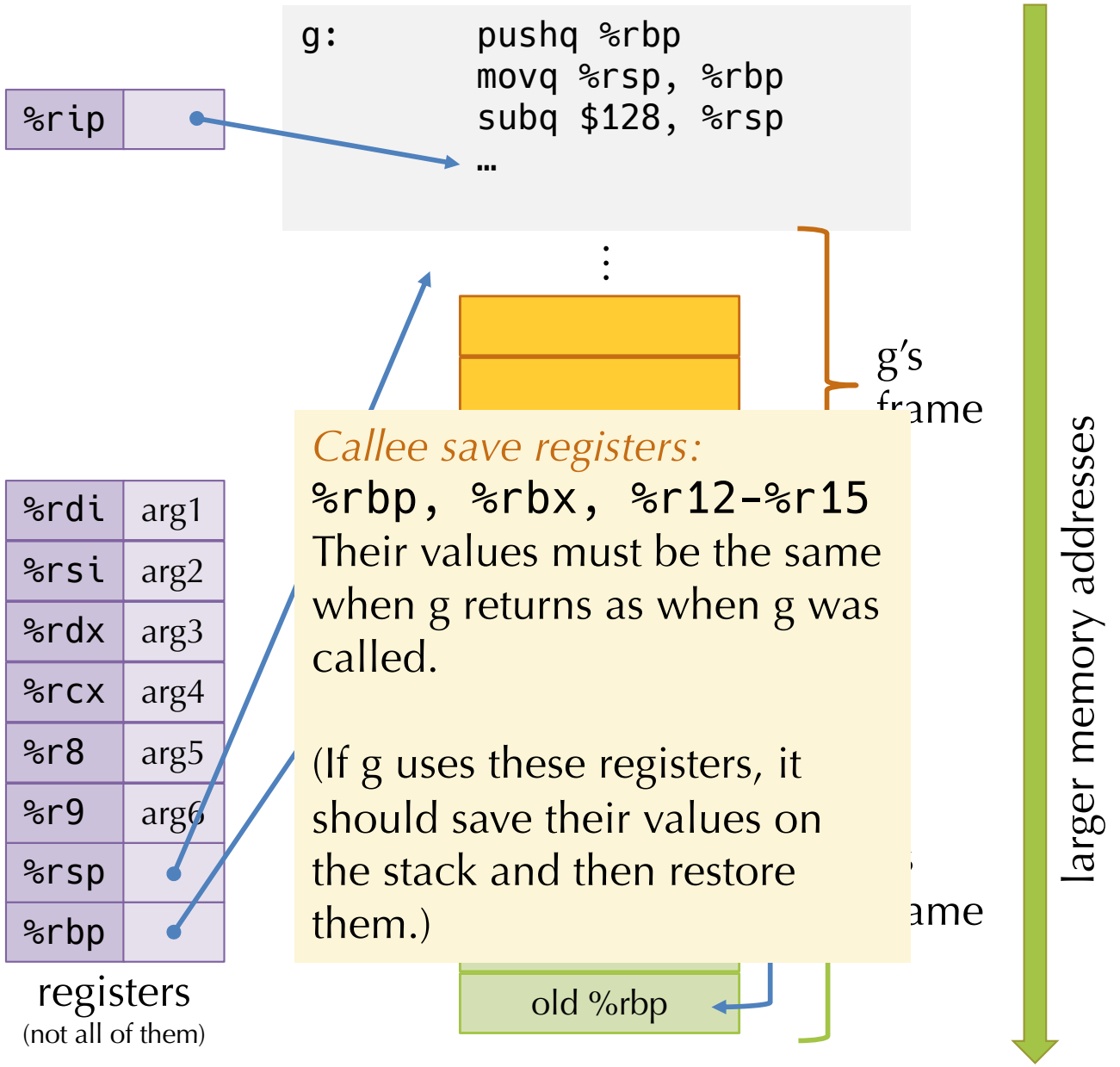
callee function prologue



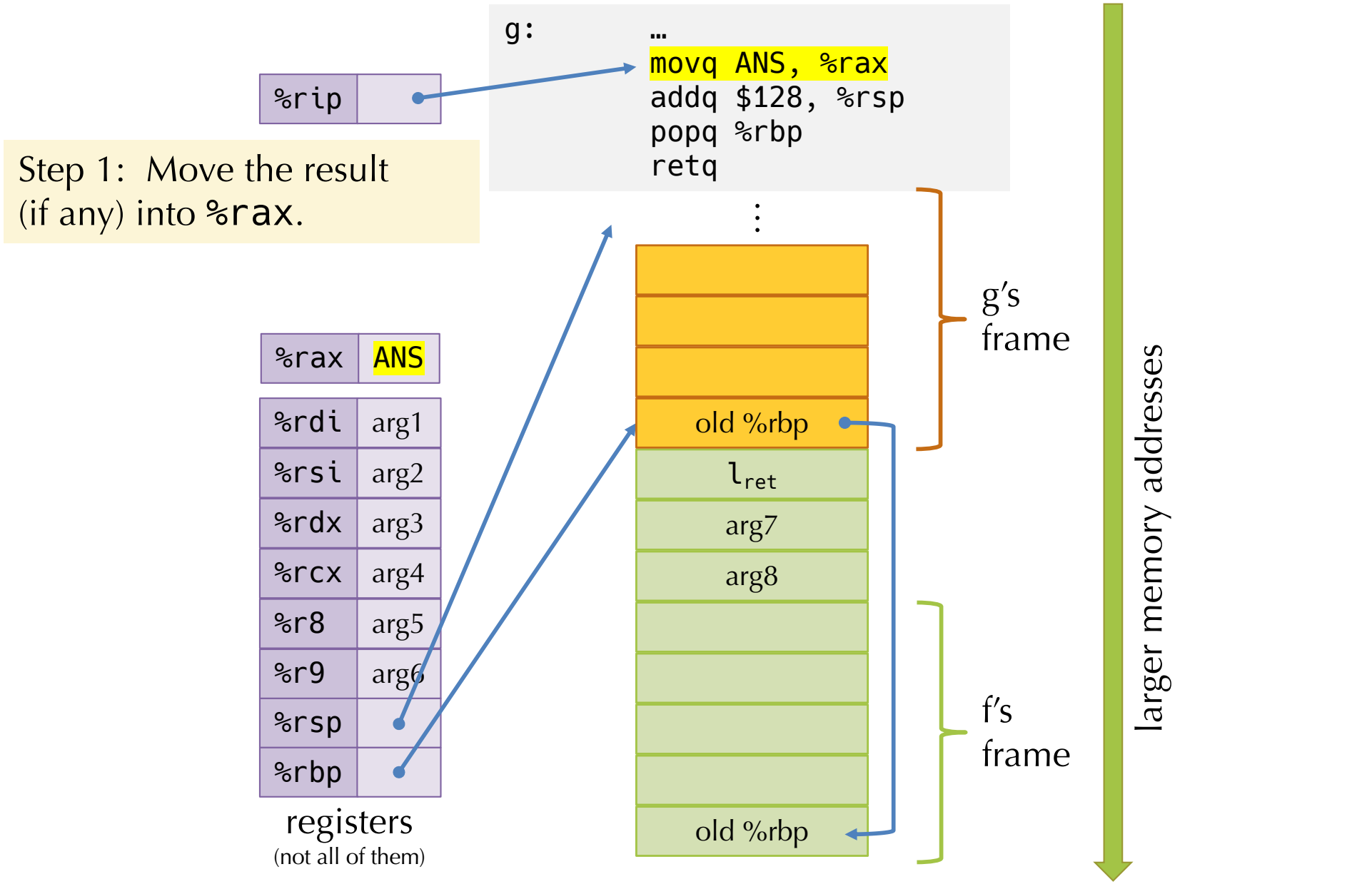
callee invariants: function arguments



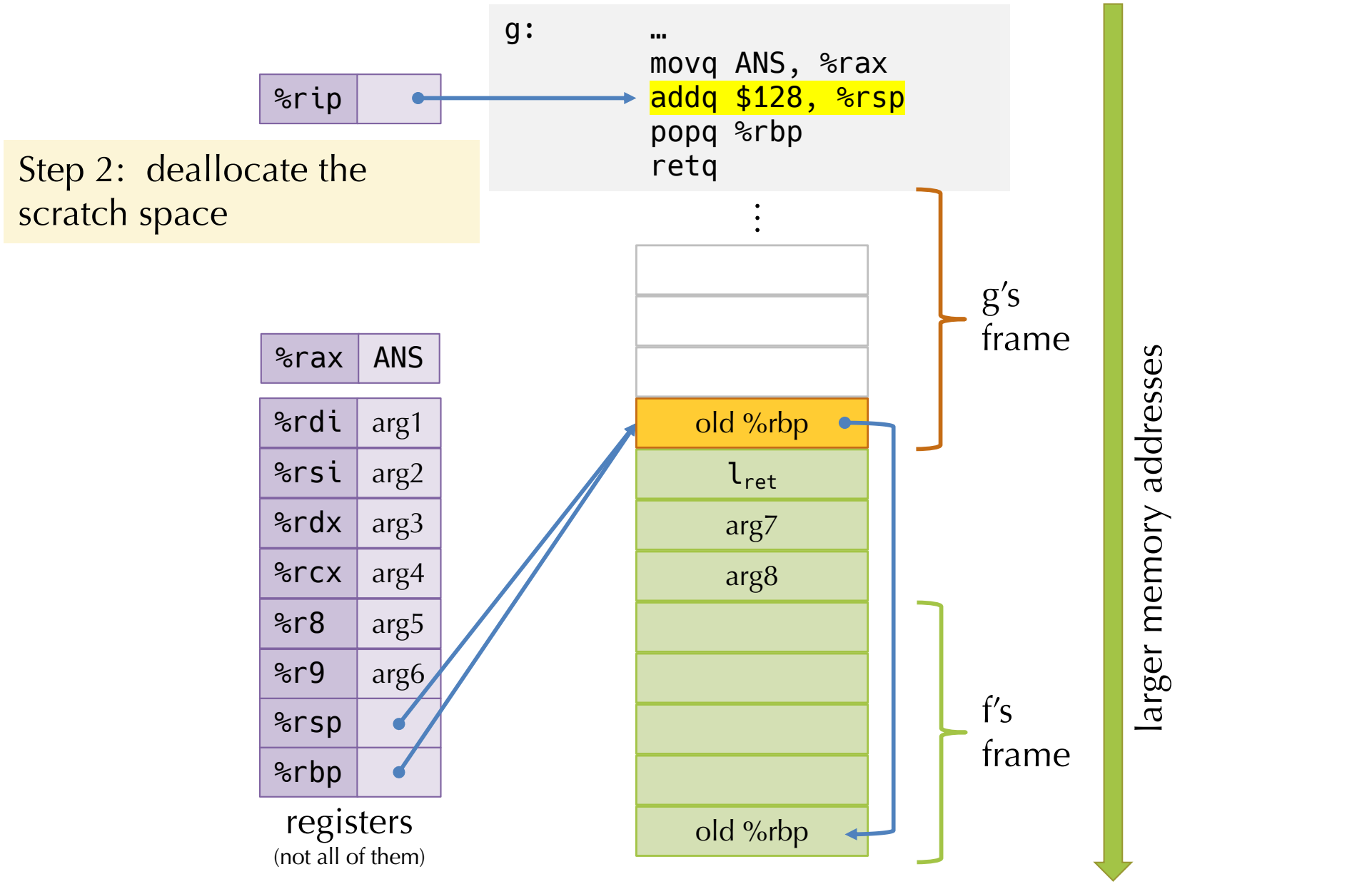
callee invariants: callee save registers



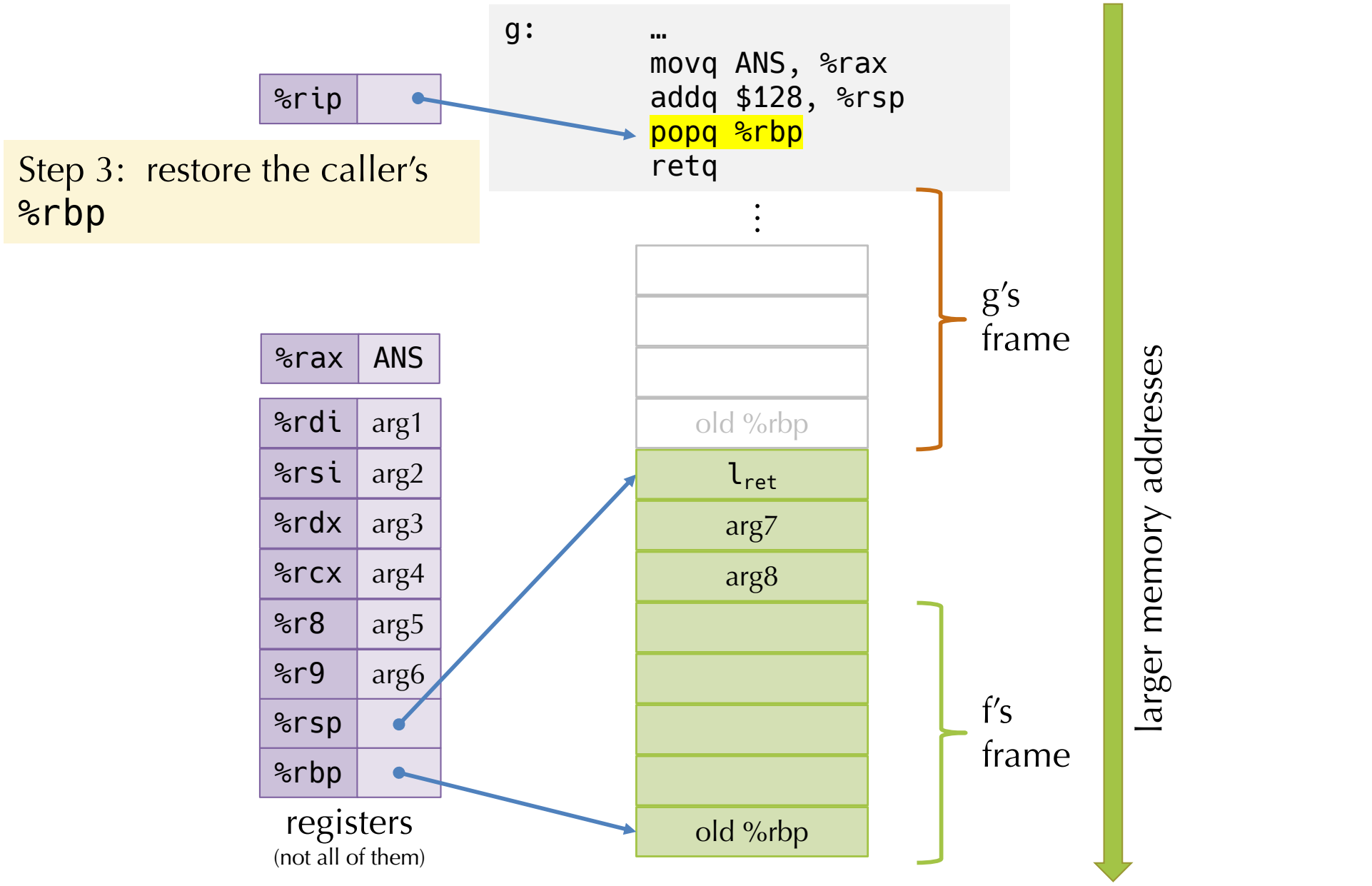
callee epilogue (return protocol)



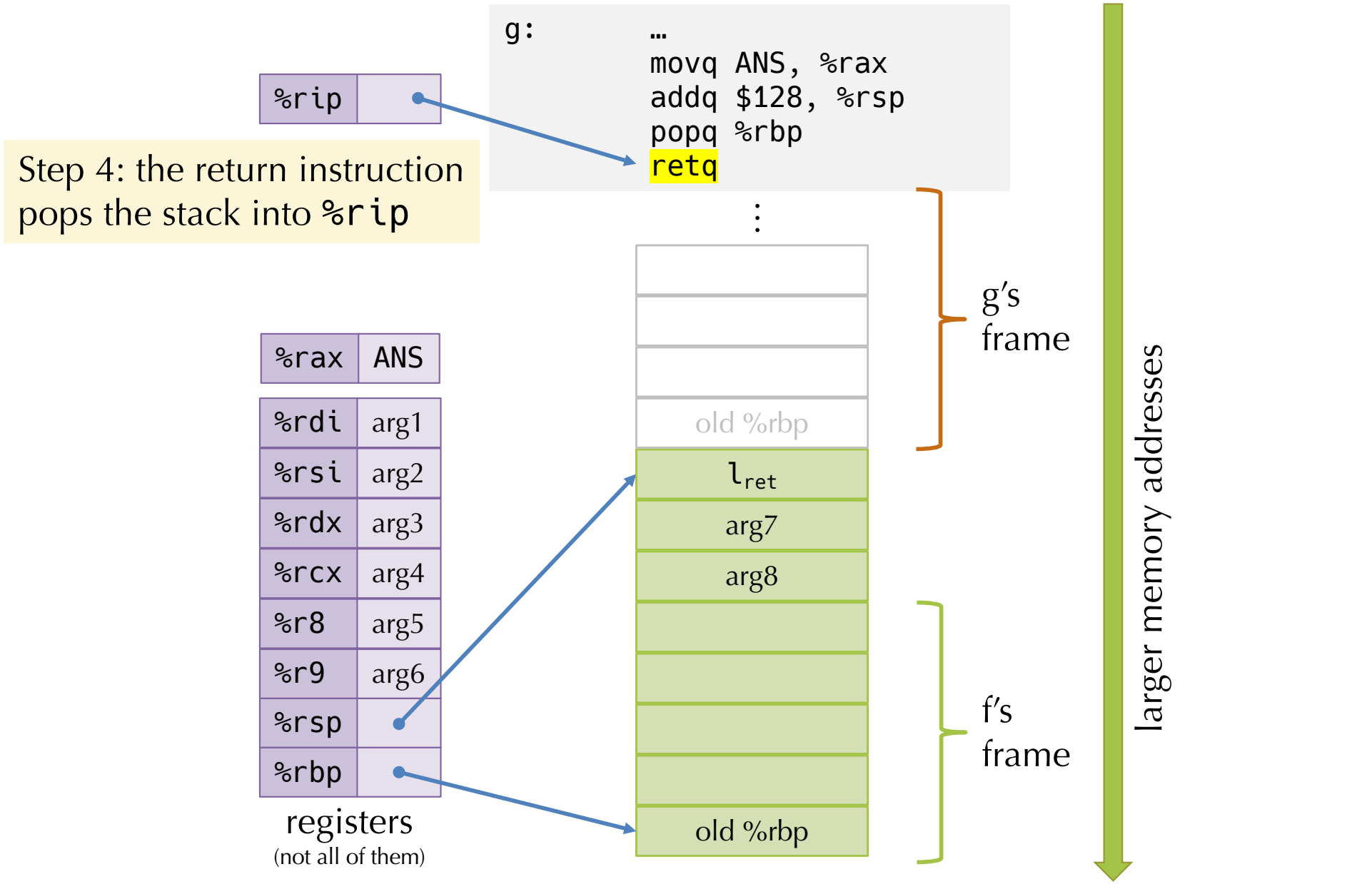
callee epilogue (return protocol)



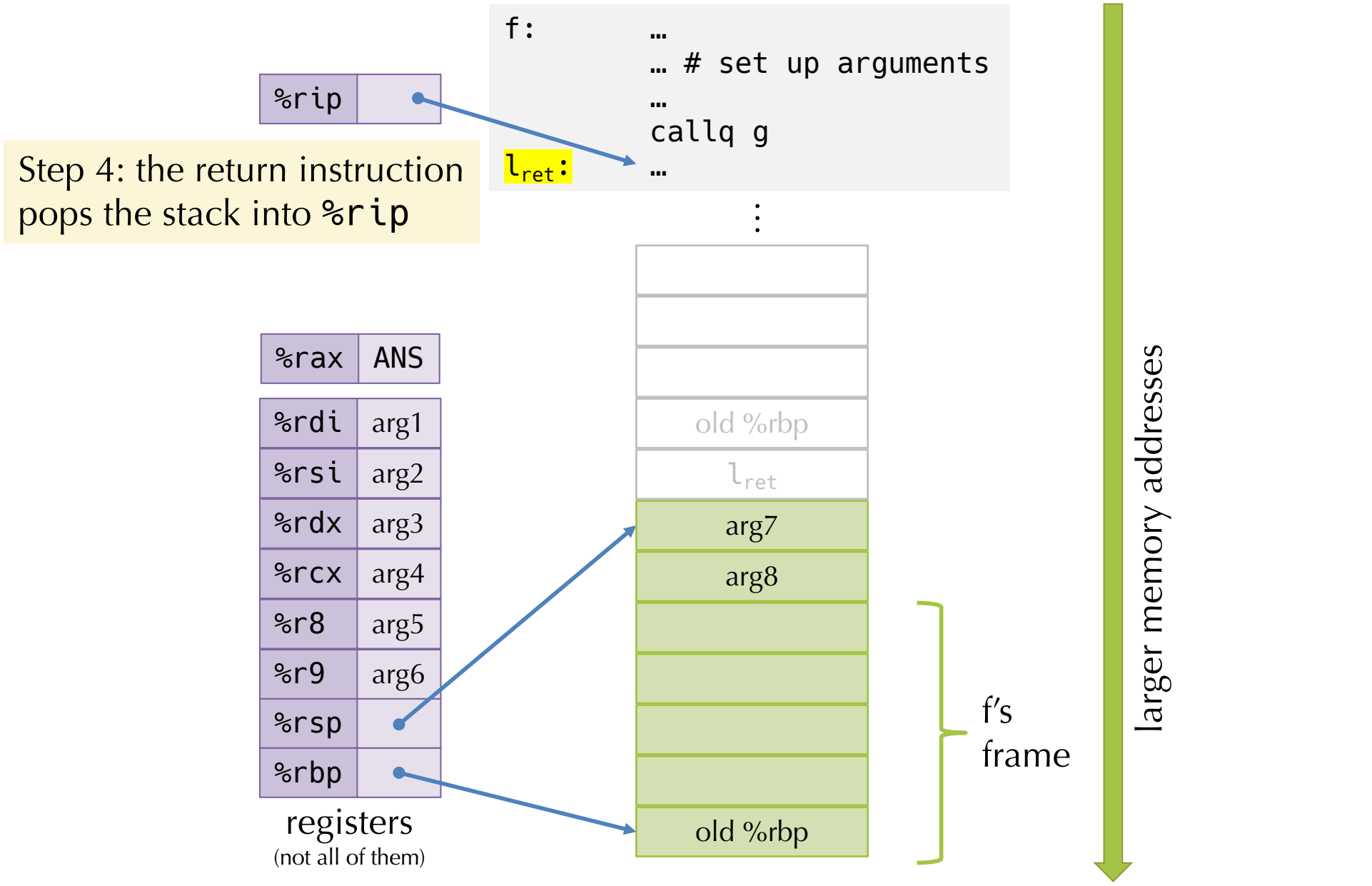
callee epilogue (return protocol)



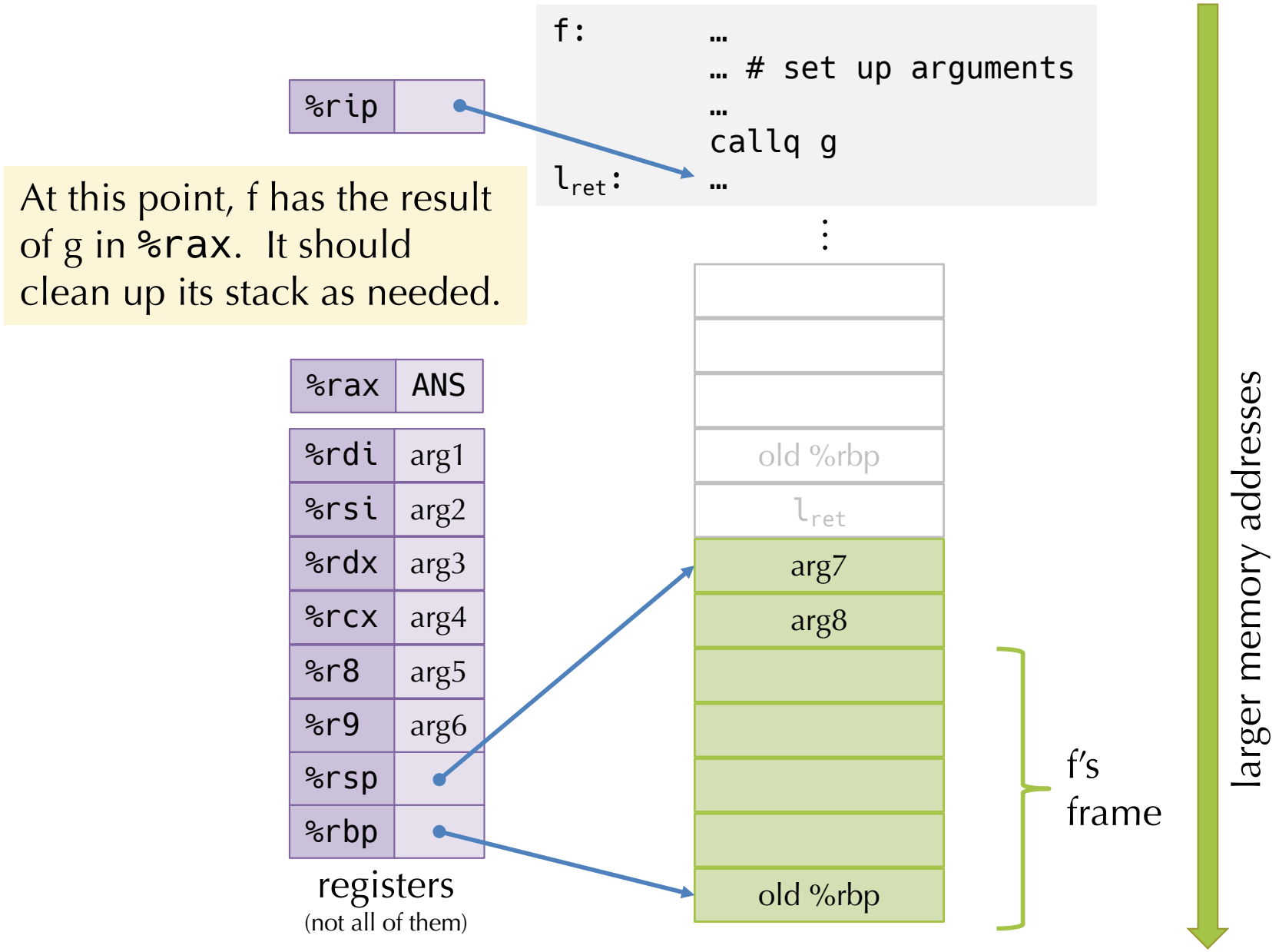
callee epilogue (return protocol)



callee epilogue (return protocol)



back in f


f:
…
… # set up arguments
…
callq g
l_ret:
…
%rip
At this point, f has the result of g in %rax. It should clean up its stack as needed.
%rax
ANS
%rdi
arg1
%rsi
arg2
%rdx
arg3
%rcx
arg4
%r8
arg5
%r9
arg6
%rsp
%rbp
registers
(not all of them)
old %rbp
l_ret
arg7
arg8
old %rbp
f's frame
larger memory addresses

X86-64 SYSTEM V AMD 64 ABI

- Modern variant of C calling conventions
  - used on Linux, Solaris, BSD, OS X

- Callee save: `%rbp, %rbx, %r12-%r15`
- Caller save: all others

- Parameters 1 .. 6 go in: `%rdi, %rsi, %rdx, %rcx, %r8, %r9`
- Parameters 7+ go on the stack (in right-to-left order)
  - so: for n > 6, the n^{th} argument is located at (((n-7)+2)*8)(`%rbp`)
  - e.g.: argument 7 is at `16(%rbp)` and argument 8 is at `24(%rbp)`

- Return value: in `%rax`

- 128 byte "red zone" – scratch pad for the callee's data
  - typical of C compilers, not required
  - can be optimized away

32-bit cdecl calling conventions

- Still "Standard" on X86 for many C-based operating systems
  - Still some wrinkles about return values
    (e.g., some compilers use `EAX` and `EDX` to return small values)
  - 64 bit allows for packing multiple values in one register
- All arguments are passed on the stack in right-to-left order
- Return value is passed in `EAX`
- Registers `EAX`, `ECX`, `EDX` are caller save
- Other registers are callee save
  - Ignoring these conventions will cause havoc (bus errors or seg faults)