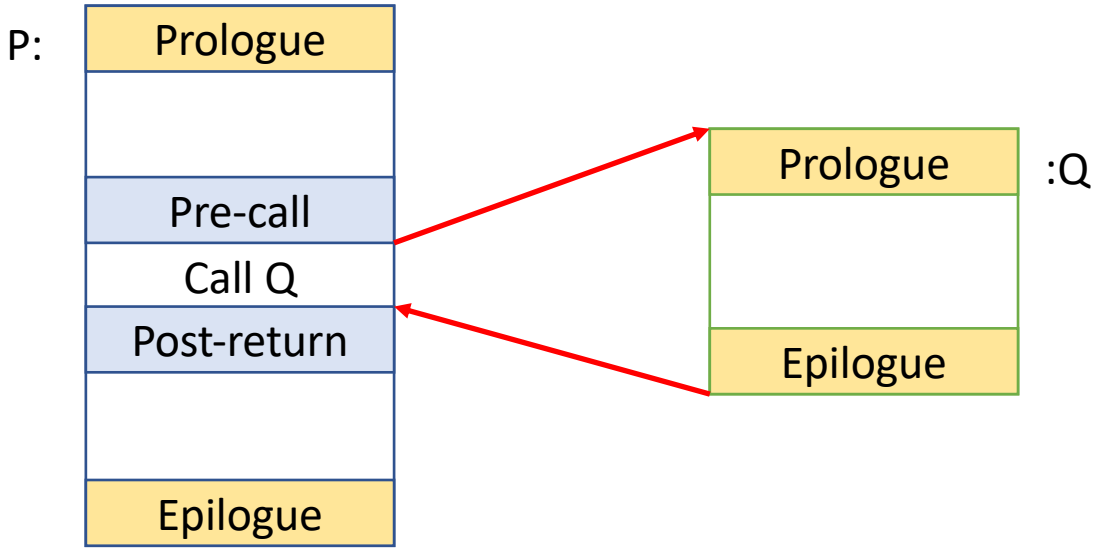
Procedure Linkage

- A procedure linkage is a contract between the compiler, the operating system, and the target machine that clearly divides responsibility for naming, allocation of resources, addressability, and protection.
- Standard procedure linkage
  - Pre-call and post-return sequences at each call site.
  - Prologue and epilogue sequences in each procedure.



Procedure Linkage on x86

- On real machines, the procedure linkage conventions of the ABI are complicated, and are often a function of the processor, OS, programming language, and (possibly) compiler.
  - This can lead to potential lack of interoperability.
- Also, the machine-level data widths and alignment requirements of different data types has to be taken into account.

- On IA-32, there were at least six different conventions for C/C++, differing in parameter passing details, stack cleanup responsibilities, and handling of struct/class arguments and results.
- On x86-64, there are two major conventions: the Microsoft x64 calling convention and the System V AMD64 ABI.
  - `gcc` and `clang` support both, depending on operating system.

What The Architecture Says (1 of 4)

[Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual, §6.2.4]

- Procedure linking information
  - The *stack-frame base pointer*: `%RBP` identifies a fixed reference point within the stack frame for the called procedure.
  - The *return instruction pointer*: Prior to branching to the first instruction of the called procedure, the `CALL` instruction pushes the address in the `%RIP` register onto the current stack. This address is then called the return-instruction pointer and it points to the instruction where execution of the calling procedure should resume following a return from the called procedure. Upon returning from a called procedure, the `RET` instruction pops the return-instruction pointer from the stack back into the `%RIP` register. Execution of the calling procedure then resumes.

What The Architecture Says (2 of 4)

[Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual, §6.4.1]

- Procedure calls are supported with `CALL` and `RET` instructions.
- When executing `CALL`, the processor does the following:
  - Pushes the current value of `%RIP` on the stack.
  - Loads the offset of the called procedure in `%RIP`.
  - Begins execution of the called procedure.
- When executing `RET`, the processor performs these actions:
  - Pops the TOS value (the return instruction pointer) into `%RIP`.
  - If the `RET` has an optional argument *n*, increments `%RIP` by *n*.
  - Resumes execution of the calling procedure.

What The Architecture Says (3 of 4)

[Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual, §6.4.3]

- Parameters (or results) can be passed between procedures in any of three ways: through GPRs, in an argument list, or on the stack.
  - The processor does not save the state of the GPRs on procedure calls. A calling procedure can thus pass up to six parameters to the called procedure by copying the parameters into any of these registers (except `%RSP` and `%RBP`) prior to executing the CALL instruction.
  - To pass a large number of parameters to the called procedure, the parameters can be placed on the stack, in the stack frame for the calling procedure. Here, it is useful to use the stack-frame base pointer (`%RBP`) to make a frame boundary for easy access to the parameters.
  - An alternate method of passing a larger number of parameters (or a data structure) to the called procedure is to place the parameters in an argument list in memory. A pointer to the argument list can then be passed to the called procedure through a GPR or the stack.

What The Architecture Says (4 of 4)

[Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual, §6.4.4]

- The processor does not save the contents of the GPRs or the `RFLAGS` register on a procedure call. A calling procedure should explicitly save the values in any of the GPRs that it will need when it resumes execution after a return. These values can be saved on the stack or in memory.
- The `PUSHA` and `POPA` instructions facilitate saving and restoring the contents of all of the GPRs.
- If a calling procedure needs to maintain the state of the `RFLAGS` register, it can save and restore all or part of the register using the `PUSHF`/`PUSHFD` and `POPF`/`POPFD` instructions.