System Initialization

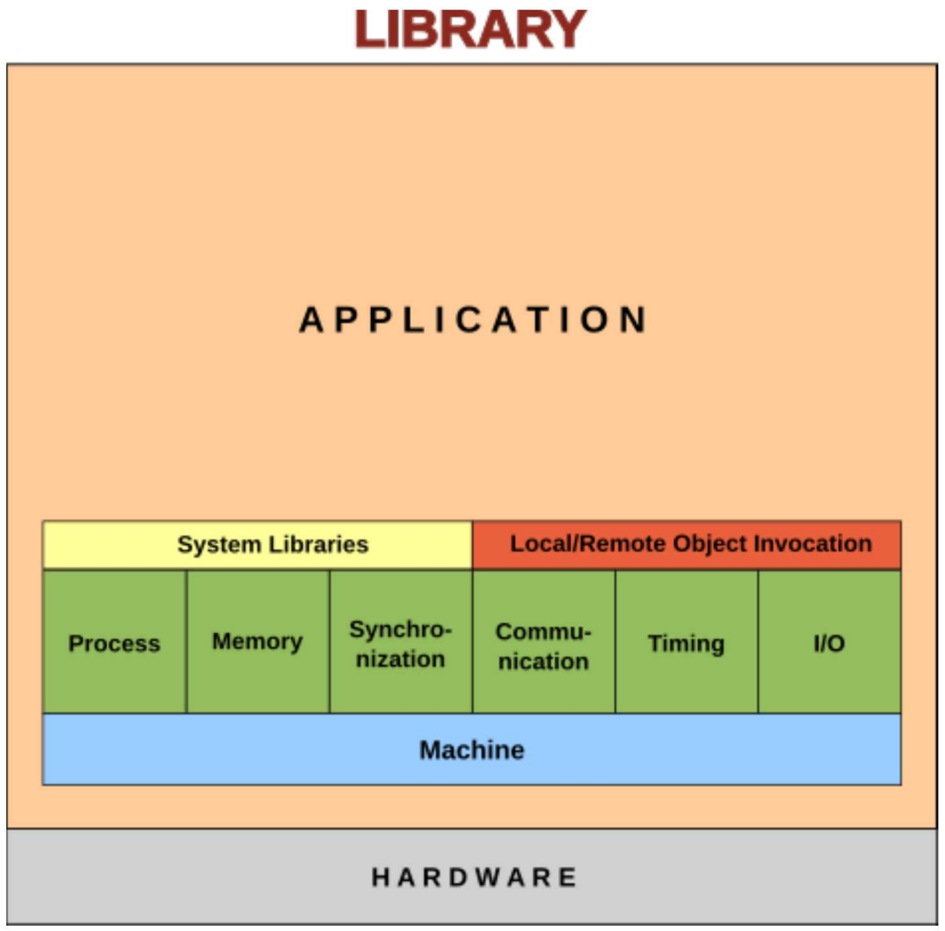
System software architectures

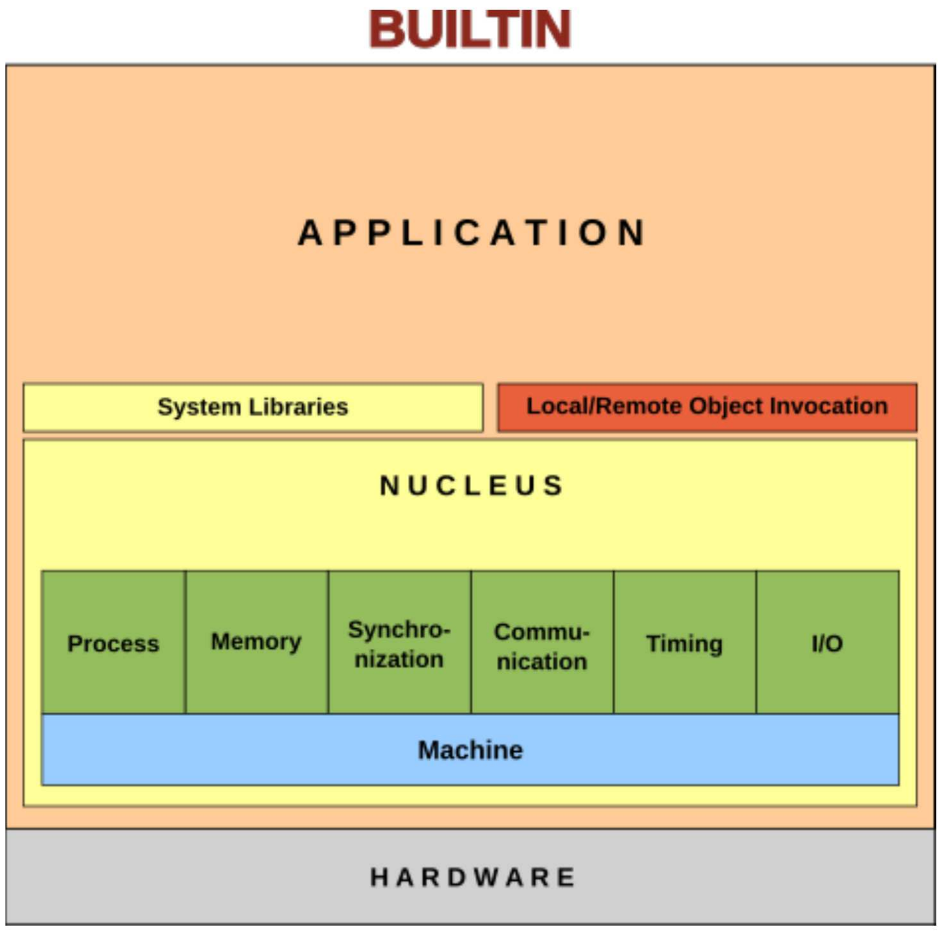
Bootstrapping the OS

Initializing the OS

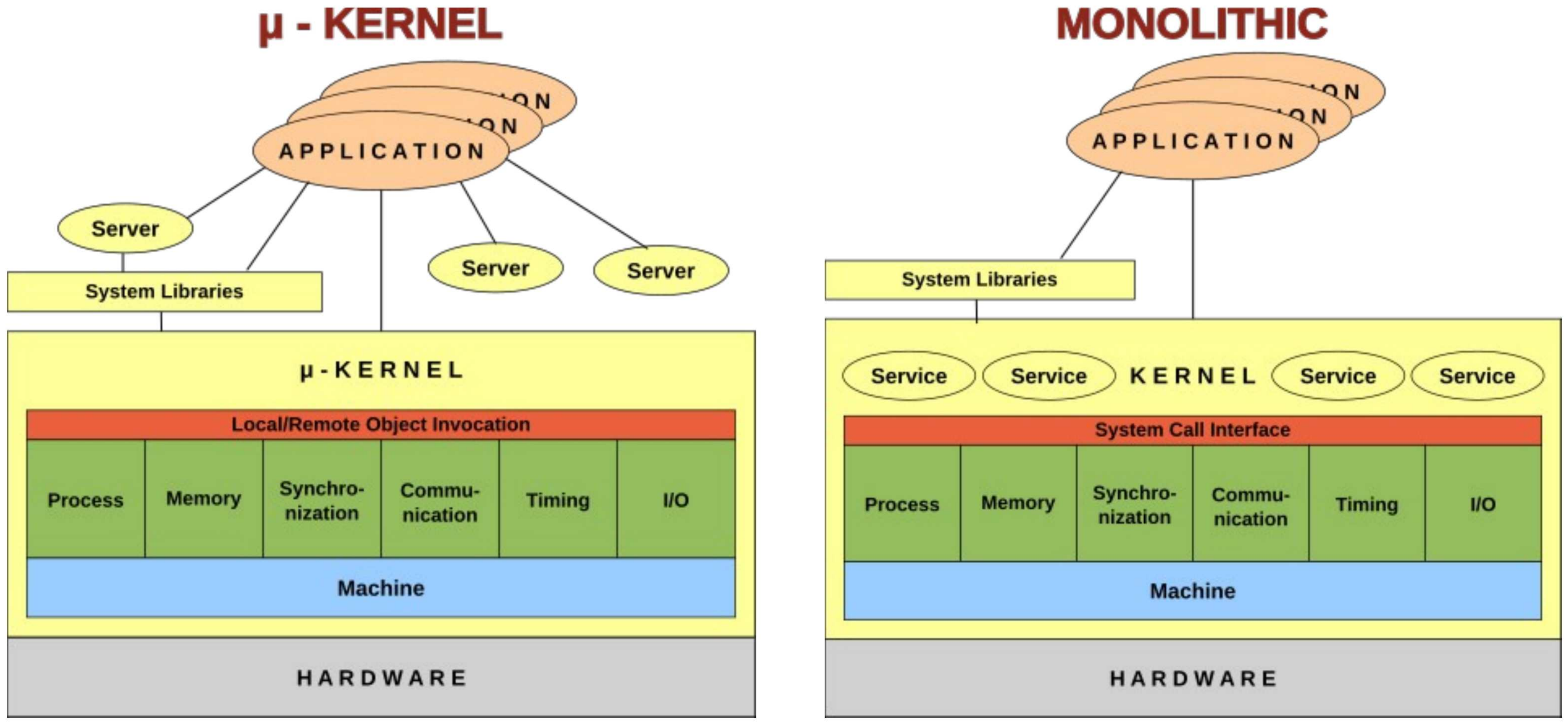
Defining an initial memory model

A Word on System Software Architectures



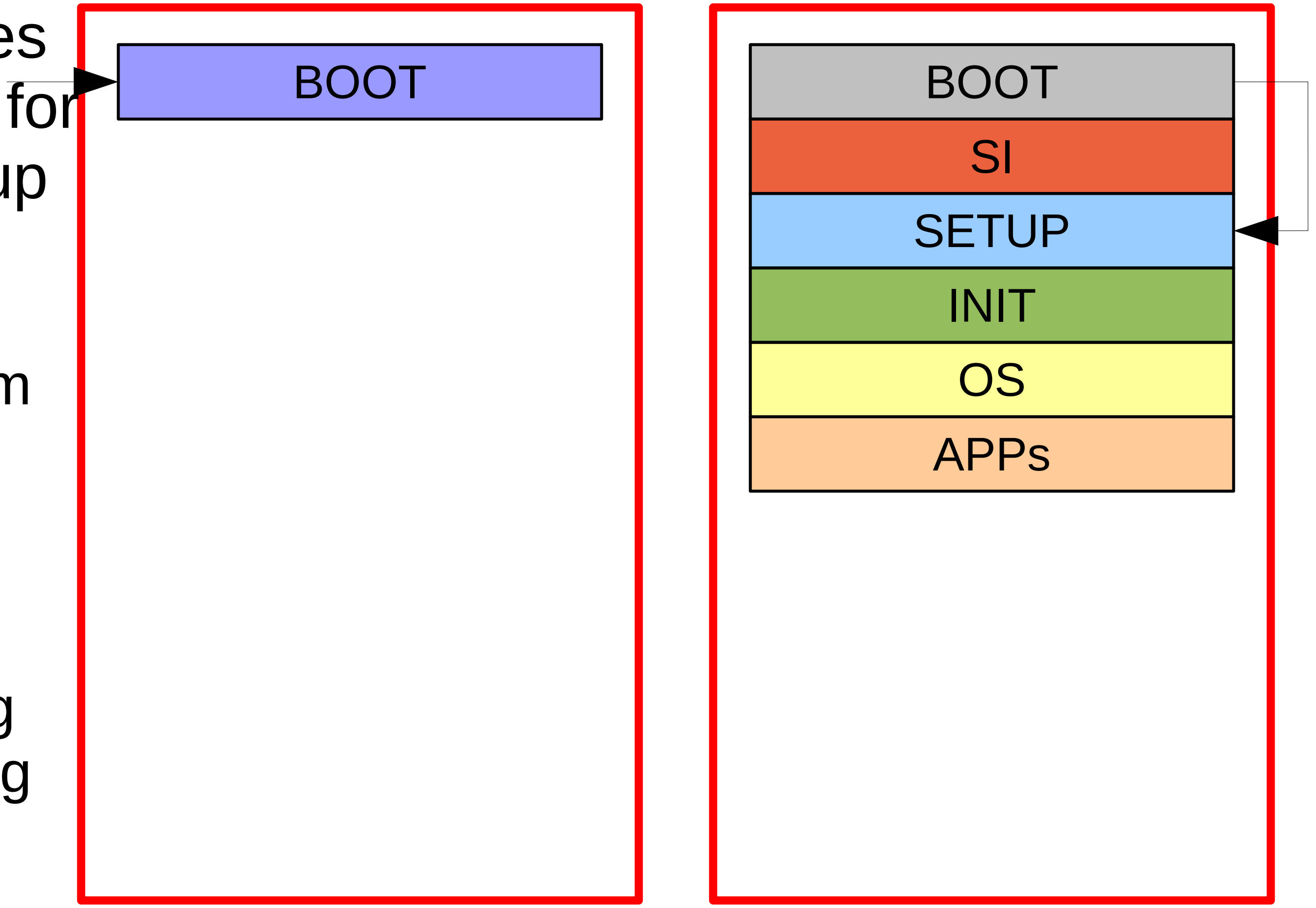


A Word on System Software Architectures



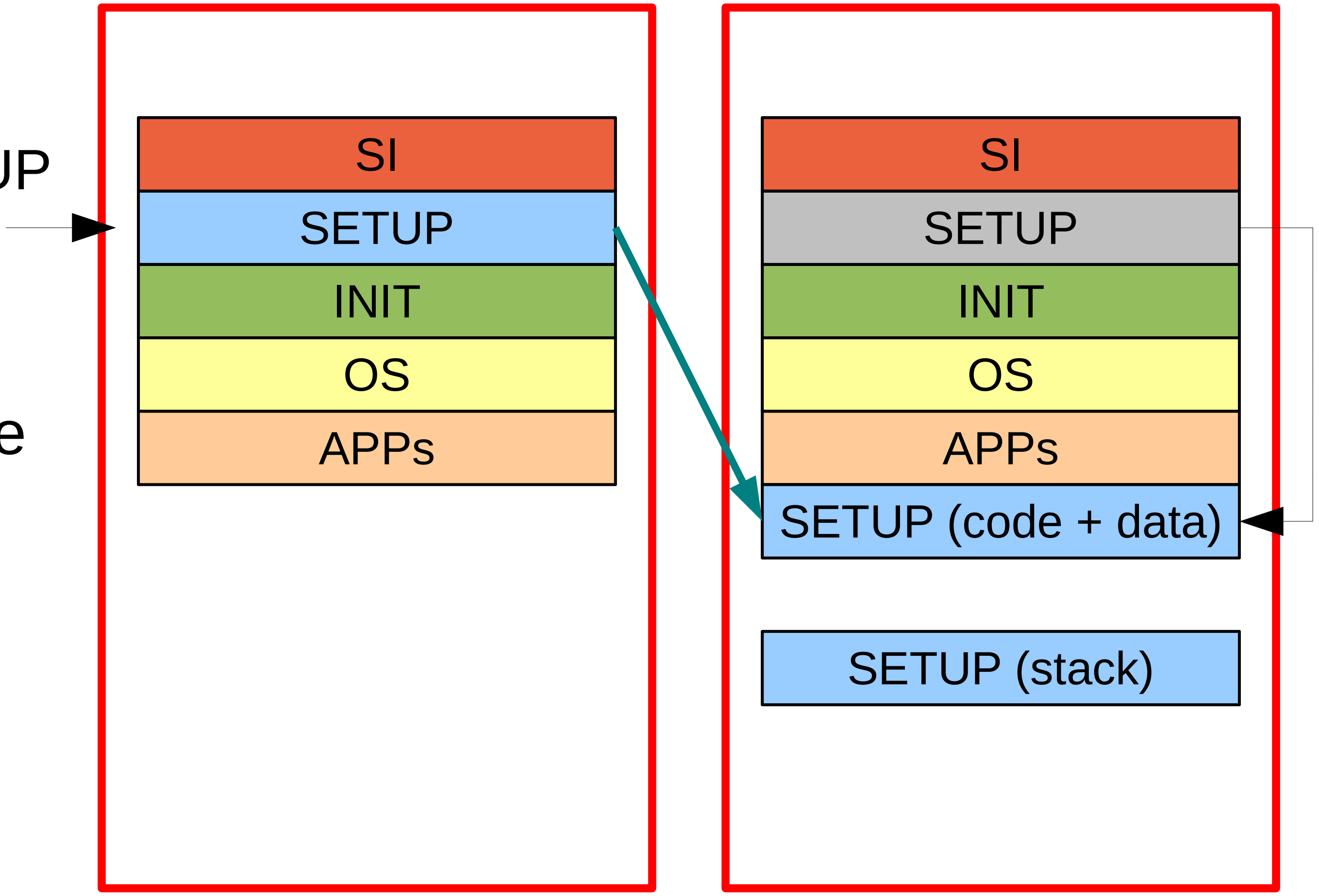
Initialization: Bootstrap I

- Only present in machines that boot from media or for architectures that start up badly
- Duties
  - Load the boot image from media (using the BIOS)
  - Disable interrupts
  - Enter GCC-compatible mode (if applicable)
  - Jump to SETUP skipping the ELF header assuming PC-relative addressing



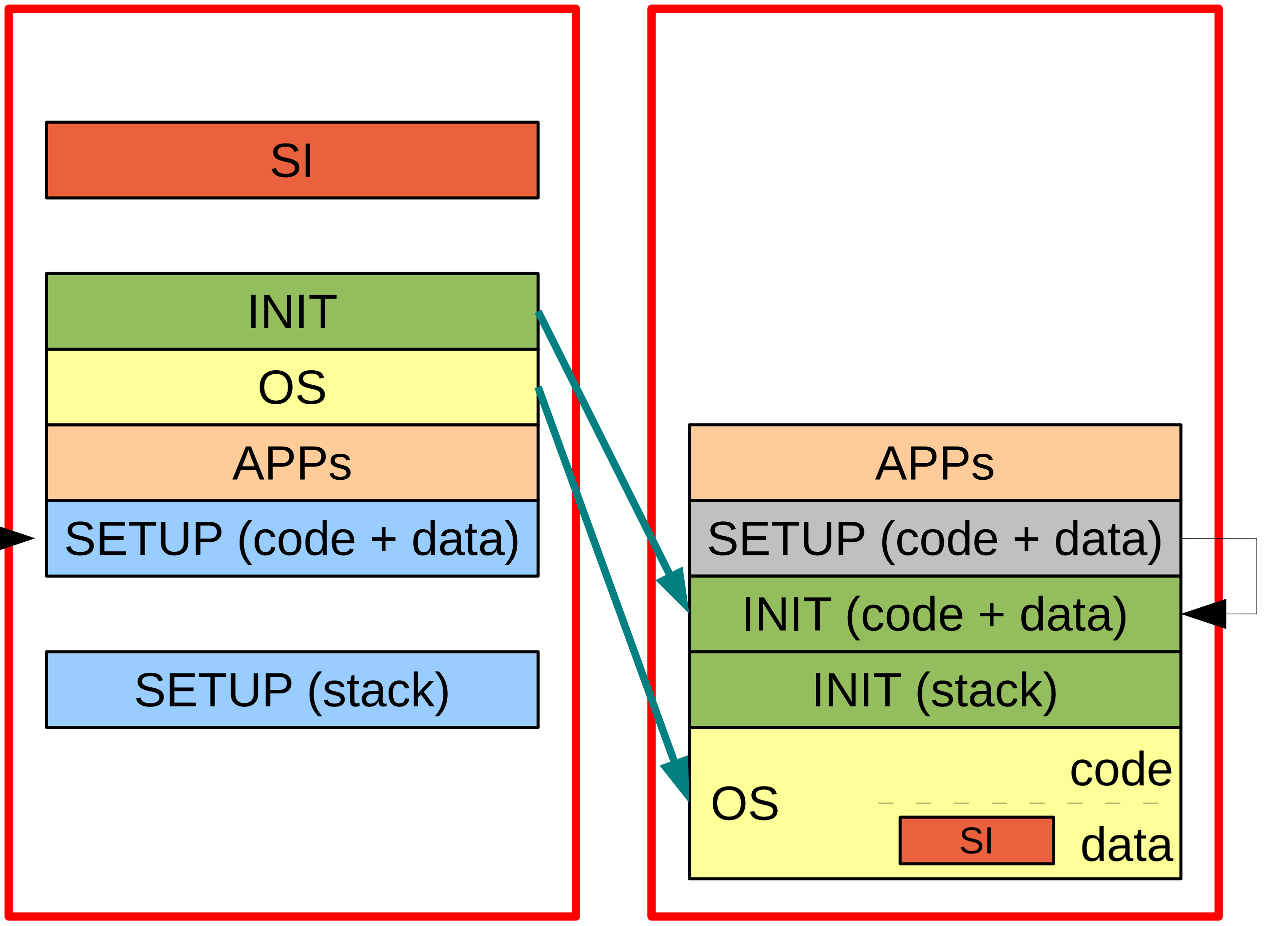
Initialization: Bootstrap II

- Duties
  - Relocate SETUP
  - Define a stack for SETUP
  - Jump to the relocated SETUP
- Can be merged with the first bootstrap stage



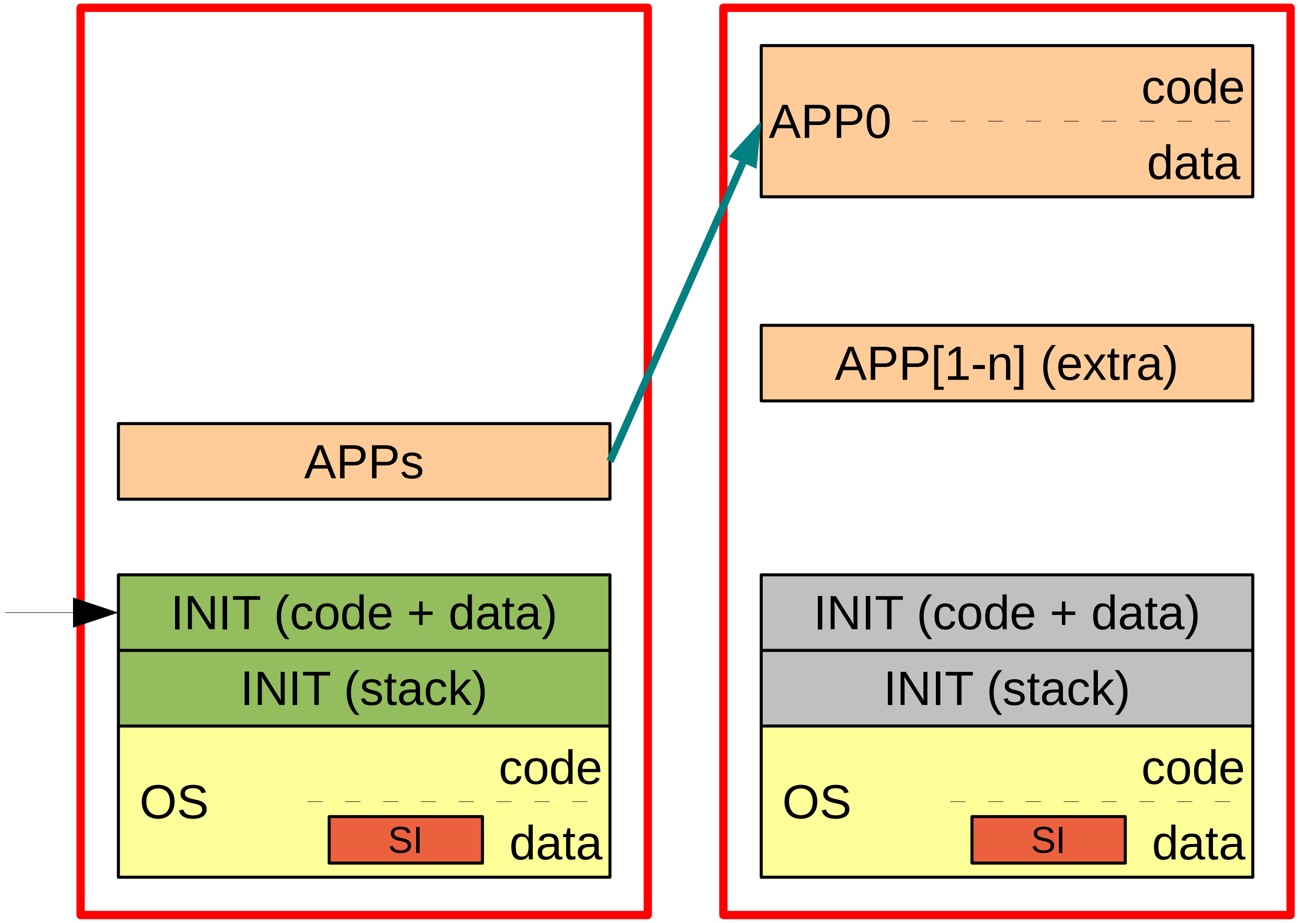
Initialization: SETUP

- Initialization code that is too exotic to fit in the OS
- Duties
  - Initialize hardware components
  - Setup an initial address space
  - Load INIT an EPOS
  - Allocate a stack for INIT
  - Jump into INIT



Initialization: INIT

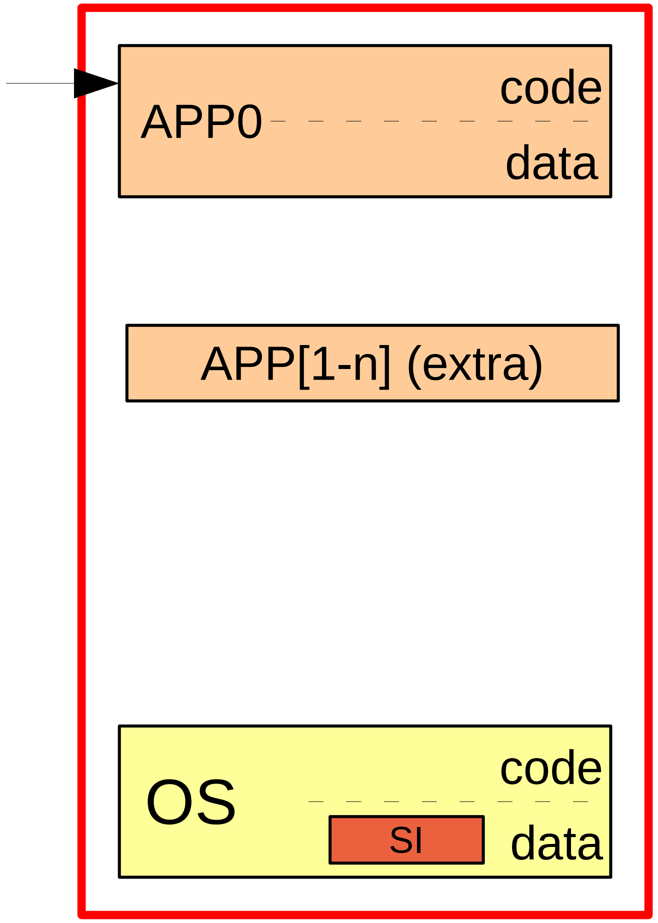
- Duties
  - Initialize EPOS components
  - Load the first APP creating the first process
    - An application loader in multitasking configs
  - For μ-kernels, install a system call interface
  - Pass what remains from the boot image to the first APP in an extra segment



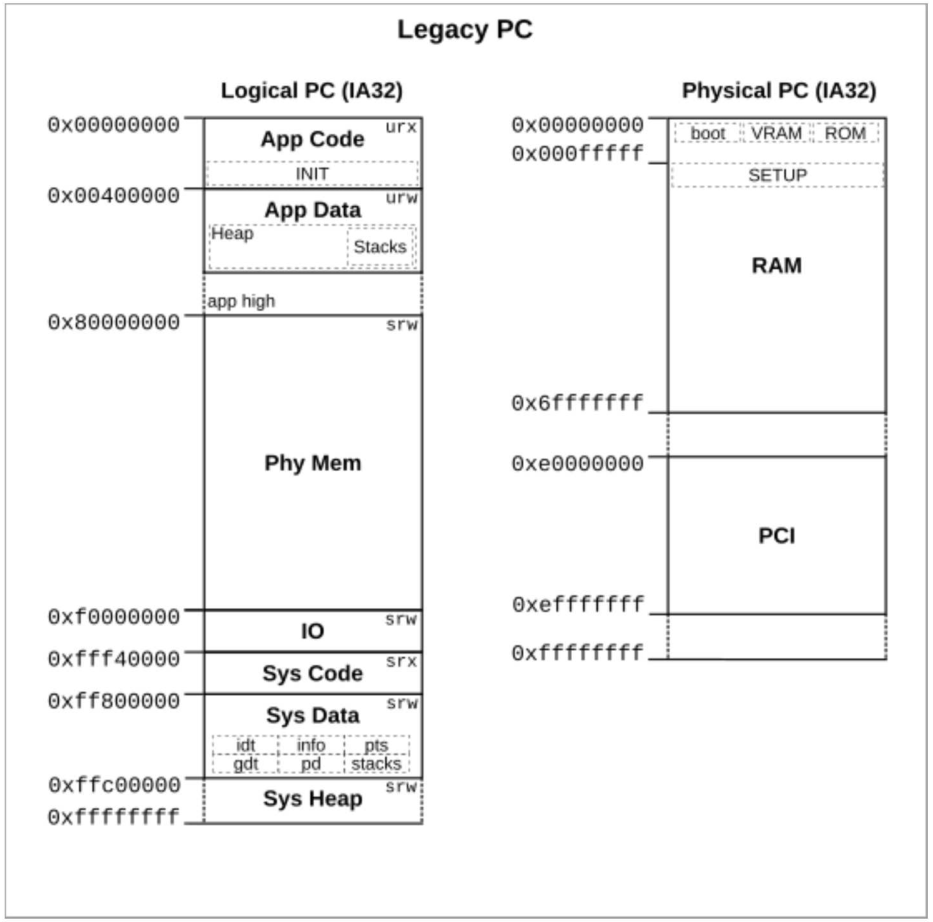
Initialization: First APP (loader)

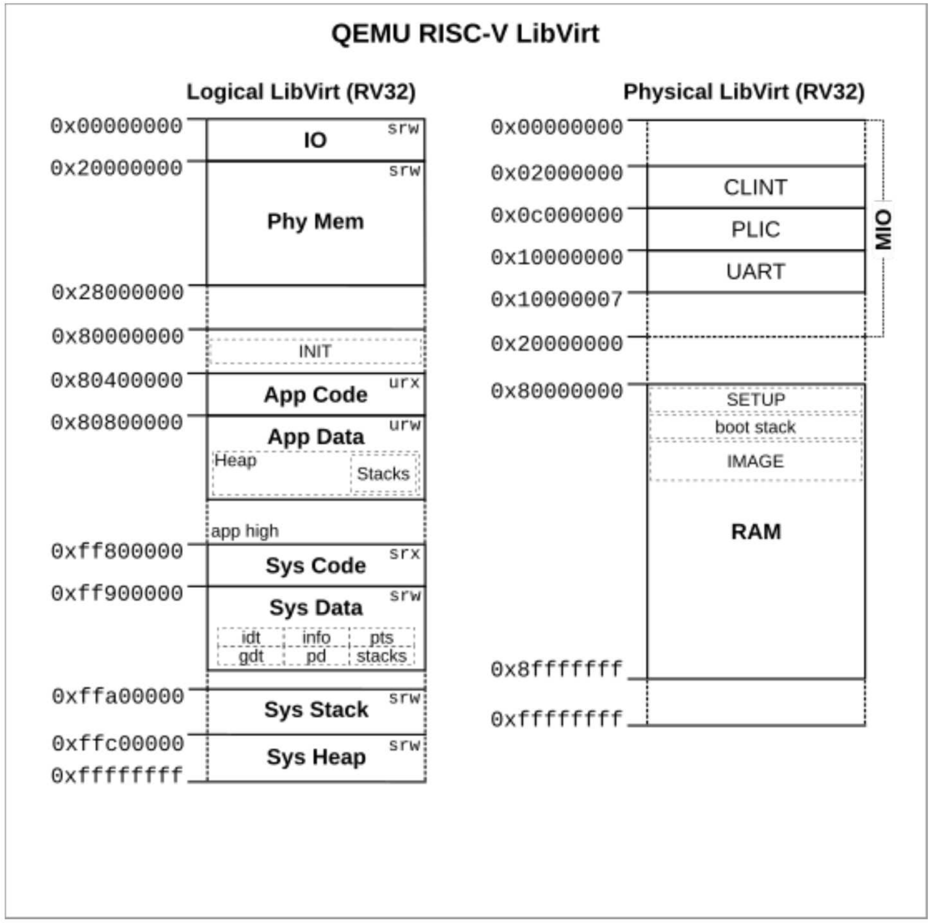
- The first APP possesses all the resources in the system
  - It is either
    - The single application
    - An application loader that loads subsequent applications

- OS is mapped into all address spaces at the same position and in supervisor mode (and thus protected from applications)



Memory Map


Legacy PC
Logical PC (IA32)
0x00000000
App Code
urx
INIT
0x00400000
App Data
urw
Heap
Stacks
app high
0x80000000
srw
Phy Mem
0xf0000000
IO
srw
0xfff40000
Sys Code
srx
0xff800000
Sys Data
srw
idt
info
pts
gdt
pd
stacks
0xffc00000
Sys Heap
srw
0xffffffff
Physical PC (IA32)
0x00000000
0x000fffff
boot
VRAM
ROM
SETUP
RAM
0x6fffffff
0xe0000000
PCI
0xefffffff
0xffffffff


QEMU RISC-V LibVirt
Logical LibVirt (RV32)
0x00000000
IO
srw
0x20000000
srw
Phy Mem
0x28000000
0x80000000
INIT
0x80400000
App Code
urx
0x80800000
App Data
urw
Heap
Stacks
app high
0xff800000
Sys Code
srx
0xff900000
Sys Data
srw
idt
info
pts
gdt
pd
stacks
0xffa00000
Sys Stack
srw
0xffc00000
Sys Heap
srw
0xffffffff
Physical LibVirt (RV32)
0x00000000
0x02000000
CLINT
0x0c000000
PLIC
0x10000000
UART
0x10000007
0x20000000
MIO
0x80000000
SETUP
boot stack
IMAGE
RAM
0x8fffffff
0xffffffff

C++ Global Objects

- EPOS uses C++ global constructors to handle multiple architectures

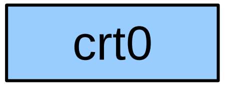
  crt0 calls *_init()* declared at crtend
    _init() uses the Init Array (IA) to call the constructors of all global objects
  at the end, *crt0* calls *_fini()*, declared at *crtbegin*
    _fini() uses the Fini Array (FA) to call the destructors of all global objects

  - IA and FA are collected by the linker in order of appearance

- For the library architecture

C++ Global Objects

- EPOS uses C++ global constructors to handle multiple architectures
  - *crt0* calls *_init()* declared at crtend
    - *_init()* uses the Init Array (IA) to call the constructors of all global objects
  - at the end, *crt0* calls *_fini()*, declared at *crtbegin*
    - *_fini()* uses the Fini Array (FA) to call the destructors of all global objects
  - IA and FA are collected by the linker in order of appearance
- For the library architecture



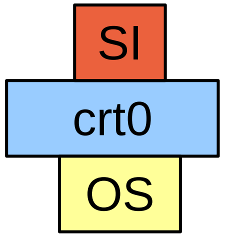
C++ Global Objects

- EPOS uses C++ global constructors to handle multiple architectures

  crt0 calls *_init()* declared at crtend
    _init() uses the Init Array (IA) to call the constructors of all global objects
  at the end, *crt0* calls *_fini()*, declared at *crtbegin*
    _fini() uses the Fini Array (FA) to call the destructors of all global objects

  - IA and FA are collected by the linker in order of appearance

- For the library architecture



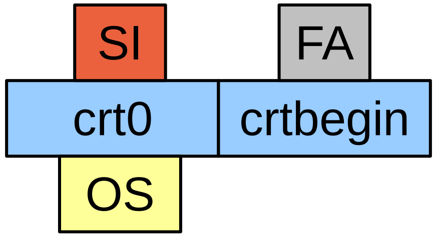
C++ Global Objects

- EPOS uses C++ global constructors to handle multiple architectures

  crt0 calls *_init()* declared at crtend
    _init() uses the Init Array (IA) to call the constructors of all global objects
  at the end, *crt0* calls *_fini()*, declared at *crtbegin*
    _fini() uses the Fini Array (FA) to call the destructors of all global objects

  - IA and FA are collected by the linker in order of appearance

- For the library architecture



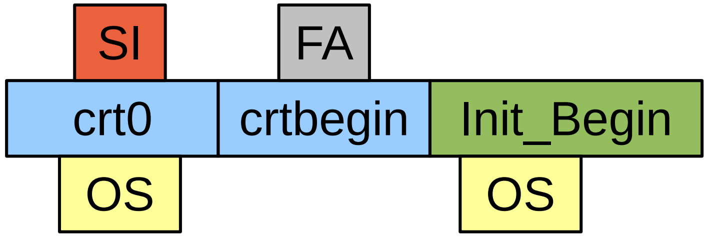
C++ Global Objects

- EPOS uses C++ global constructors to handle multiple architectures

  crt0 calls *_init()* declared at crtend
    _init() uses the Init Array (IA) to call the constructors of all global objects
  at the end, *crt0* calls *_fini()*, declared at *crtbegin*
    _fini() uses the Fini Array (FA) to call the destructors of all global objects

  - IA and FA are collected by the linker in order of appearance

- For the library architecture



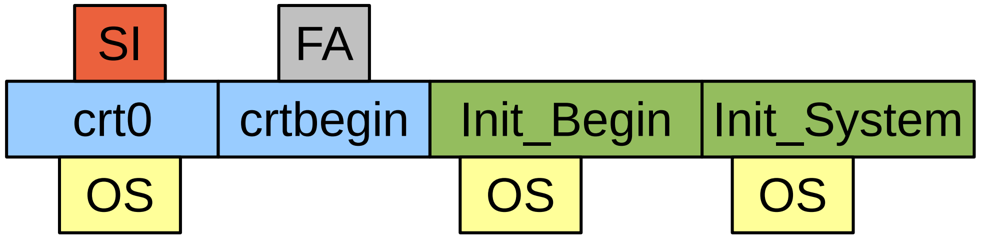
C++ Global Objects

- EPOS uses C++ global constructors to handle multiple architectures

  crt0 calls *_init()* declared at crtend
    _init() uses the Init Array (IA) to call the constructors of all global objects
  at the end, *crt0* calls *_fini()*, declared at *crtbegin*
    _fini() uses the Fini Array (FA) to call the destructors of all global objects

  - IA and FA are collected by the linker in order of appearance

- For the library architecture



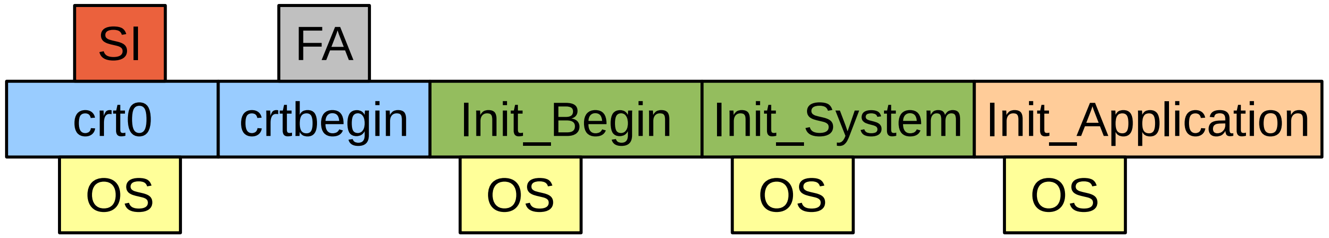
C++ Global Objects

- EPOS uses C++ global constructors to handle multiple architectures

  crt0 calls *_init()* declared at crtend
    _init() uses the Init Array (IA) to call the constructors of all global objects
  at the end, *crt0* calls *_fini()*, declared at *crtbegin*
    _fini() uses the Fini Array (FA) to call the destructors of all global objects

  - IA and FA are collected by the linker in order of appearance

- For the library architecture



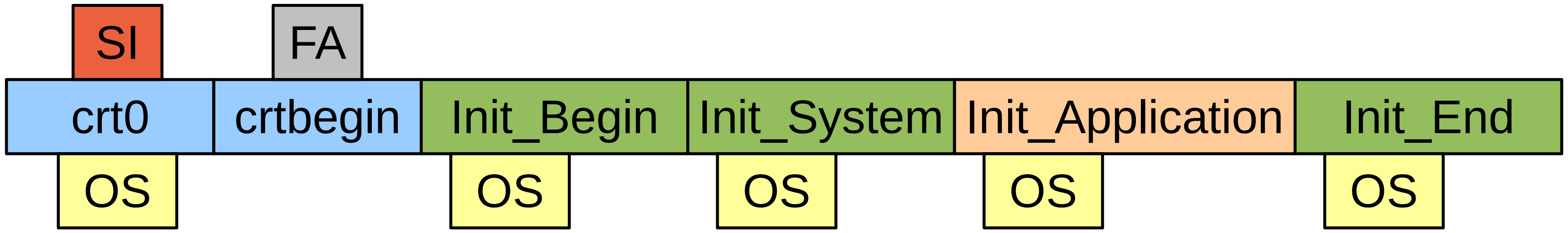
C++ Global Objects

- EPOS uses C++ global constructors to handle multiple architectures

  crt0 calls *_init()* declared at crtend
    _init() uses the Init Array (IA) to call the constructors of all global objects
  at the end, *crt0* calls *_fini()*, declared at *crtbegin*
    _fini() uses the Fini Array (FA) to call the destructors of all global objects

  - IA and FA are collected by the linker in order of appearance

- For the library architecture



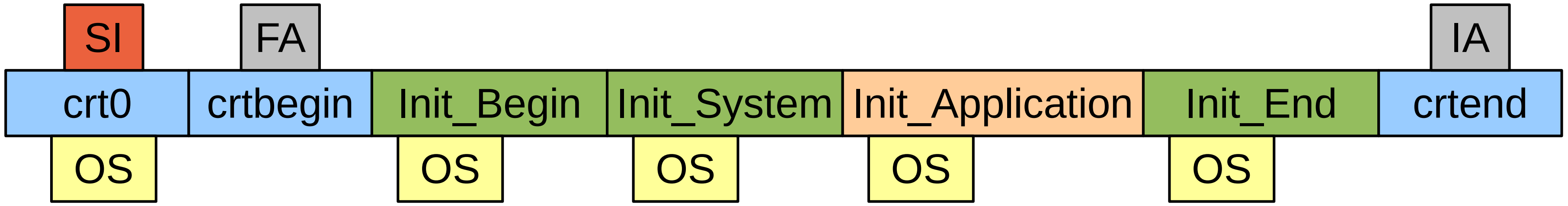
C++ Global Objects

- EPOS uses C++ global constructors to handle multiple architectures

  crt0 calls *_init()* declared at crtend
    _init() uses the Init Array (IA) to call the constructors of all global objects
  at the end, *crt0* calls *_fini()*, declared at *crtbegin*
    _fini() uses the Fini Array (FA) to call the destructors of all global objects

  - IA and FA are collected by the linker in order of appearance

- For the library architecture


SI
FA
IA
crt0
crtbegin
Init_Begin
Init_System
Init_Application
Init_End
crtend
OS
OS
OS
OS
OS