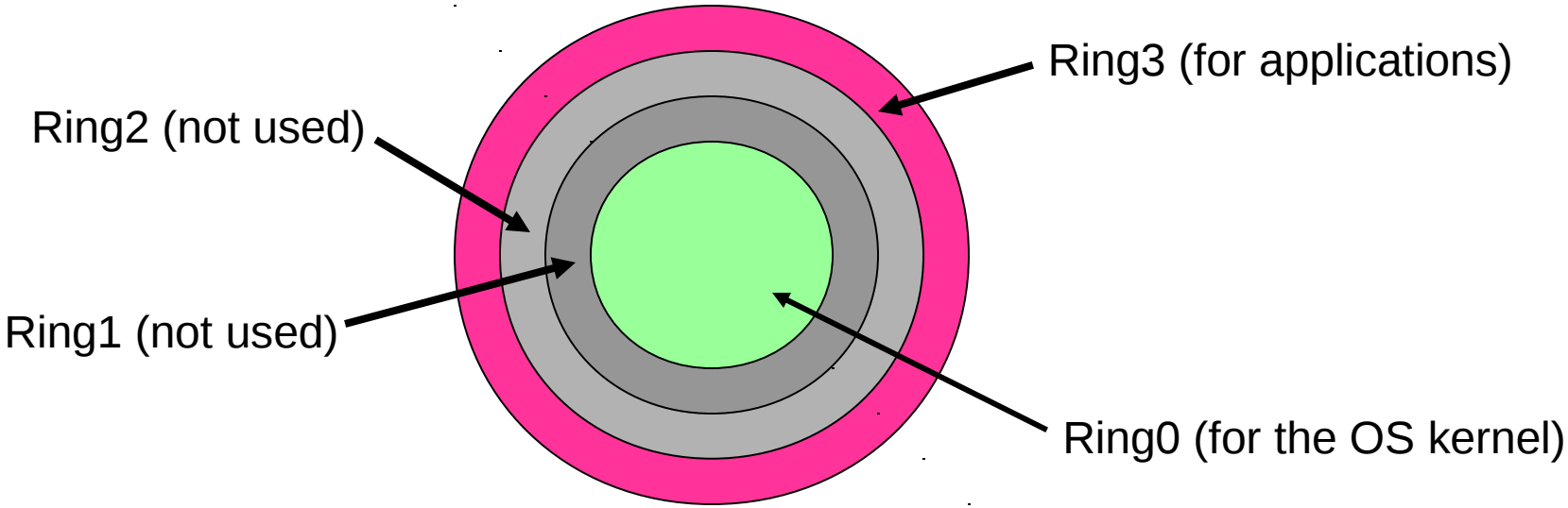
EM64T ‘fast’ system-calls

A look at the requirements for using Intel’s ‘syscall’ and ‘sysret’ instructions in 64-bit mode

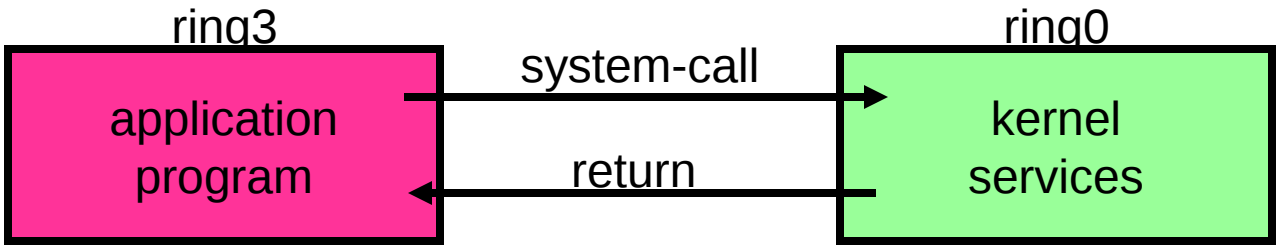
Privilege-levels

- Although the x86 processor supports four distinct privilege-levels in protected-mode, only two are actually used in the popular Windows and Linux operating-systems



Opportunity for optimization

- Just as the suppression of ‘segmentation’ in 64-bit memory-addressing has offered extra execution-speed and programming simplicity, there is opportunity for faster privilege-level transitions by eliminating references to system-tables in memory

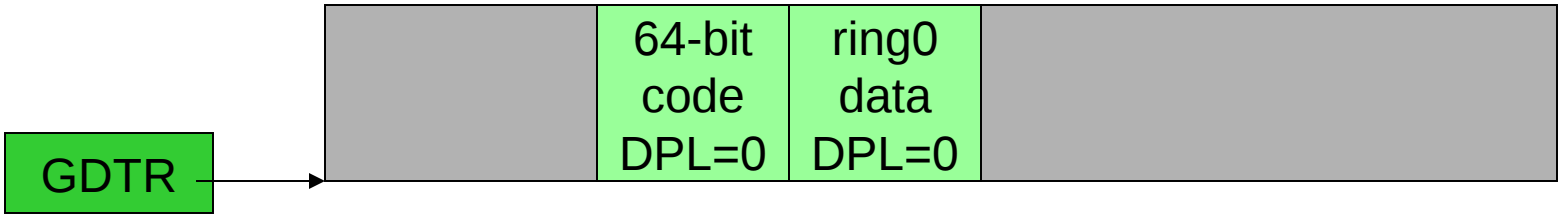


Sacrifice ‘flexibility’ for ‘speed’

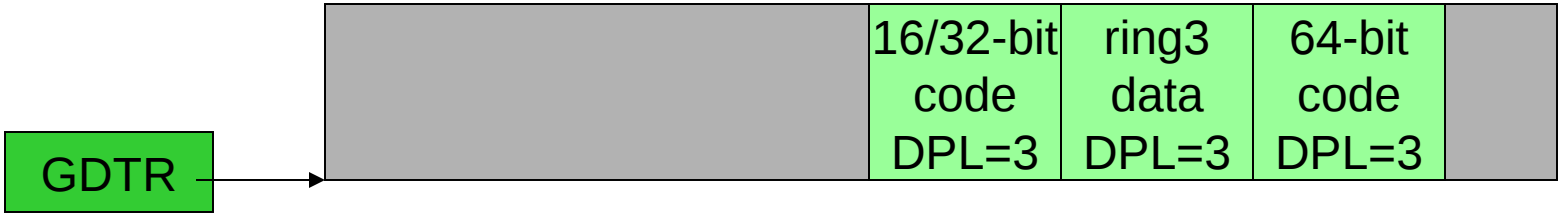
- The ‘syscall’ and ‘sysret’ instructions allow much faster ring-transitions during normal system-calls – by keeping all the required information in special CPU registers – but accepting some limitations:
  - Transitions are only between ring3 and ring0
  - Only one ‘entry-point’ to all kernel-services
  - Some formerly ‘general-purpose’ registers would have to acquire a dedicated function

Layout of GDT descriptors

- Use of ‘syscall’ requires a pair of global descriptors to be adjacently placed:



- Use of ‘sysret’ requires a triple of global descriptors to be adjacently placed:



Model-Specific Registers

- Some MSRs must be suitably initialized:

0xC0000080: IA32_MSR_EFER
0xC0000081: IA32_MSR_STAR
0xC0000082: IA32_MSR_LSTAR
0xC0000083: IA32_MSR_CSTAR
0xC0000084: IA32_MSR_FMASK

- The Intel processor must be executing in 64-bit mode to use ‘syscall’ and ‘sysret’

Extended Feature Enable Register

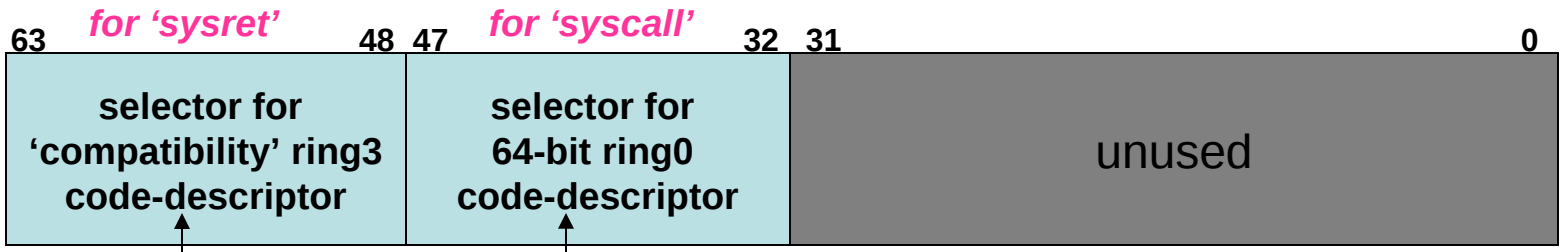
- This Model-Specific Register (MSR) was introduced in the AMD64 architecture and perpetuated by EM64T (for compatibility)

Legend:

SCE = SysCall/sysret is Enabled (1=yes, 0=no)
LME = Long-Mode is Enabled (1=yes, 0=no)
LMA = Long-Mode is Active (1=yes, 0=no)
NXE = Non-eXecutable pages Enabled (1=yes, 0=no)

NOTE: The MSR address-index for EFER = 0xC0000080, and this register is accessed using RDMSR or WRMSR instructions

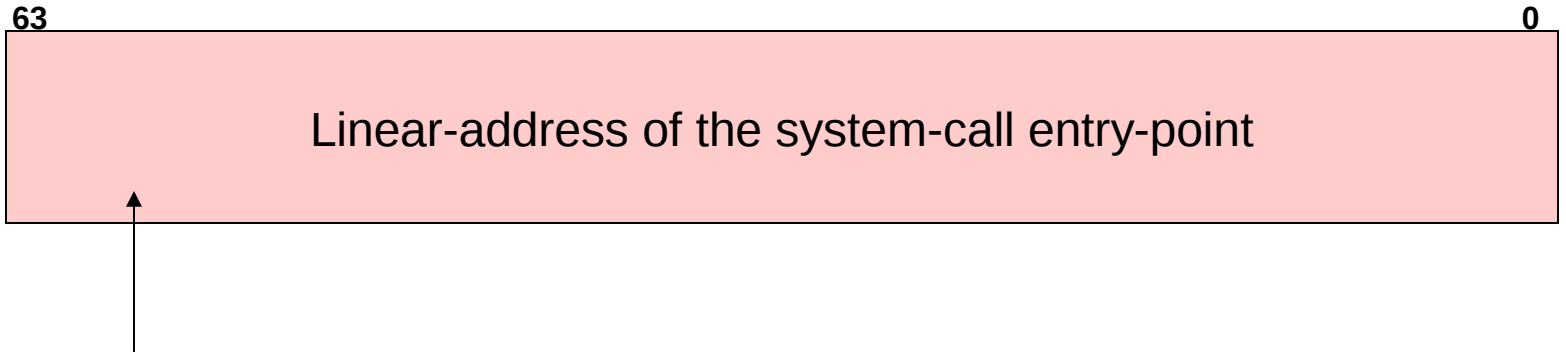
The MSR_STAR register



This selector is for the first in a pair of adjacently-placed GDT descriptors for the CS and SS registers, respectively, upon the transition from ring3 to ring0

This selector is for the first in a triple of adjacently-placed GDT descriptors for the CS and SS (or SS and CS) registers, respectively, upon the transition from ring0 to ring3 (depending on whether ‘sysret’ or ‘sysretq’ is used)

The MSR_LSTAR register

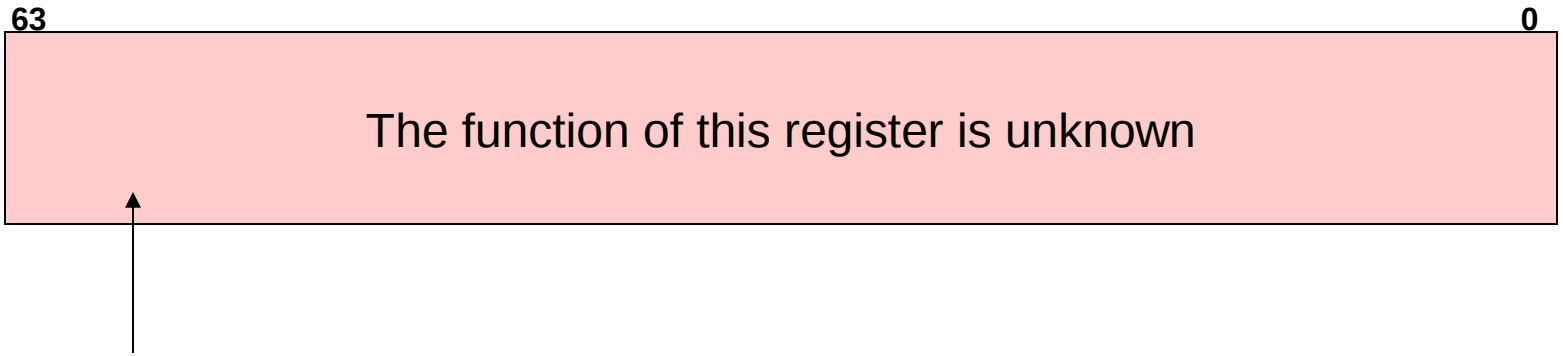


This is the 64-bit address which will go into the RIP register when the ‘syscall’ instruction is ececuted by ring3 code

The former value from the RIP register (i.e., the ‘return-address’) will be saved in the RCX general-purpose register, to be used later by the ‘sysret’ instruction (so therefore it must be preserved)

The MSR_CSTAR register

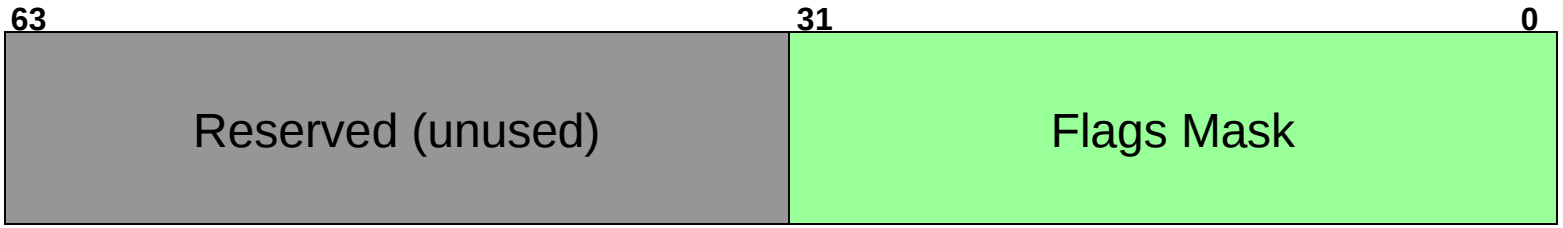
It's a mystery …



This register is observed to exist – Linux x86_64 writes a value into this register in fact – although current Intel documentation omits mention or explanation of this Model-Specific Register

(We did find an obsolete Intel document online which referred to this register, but did not make clear its past purpose or function)

The MSR_FMASK register

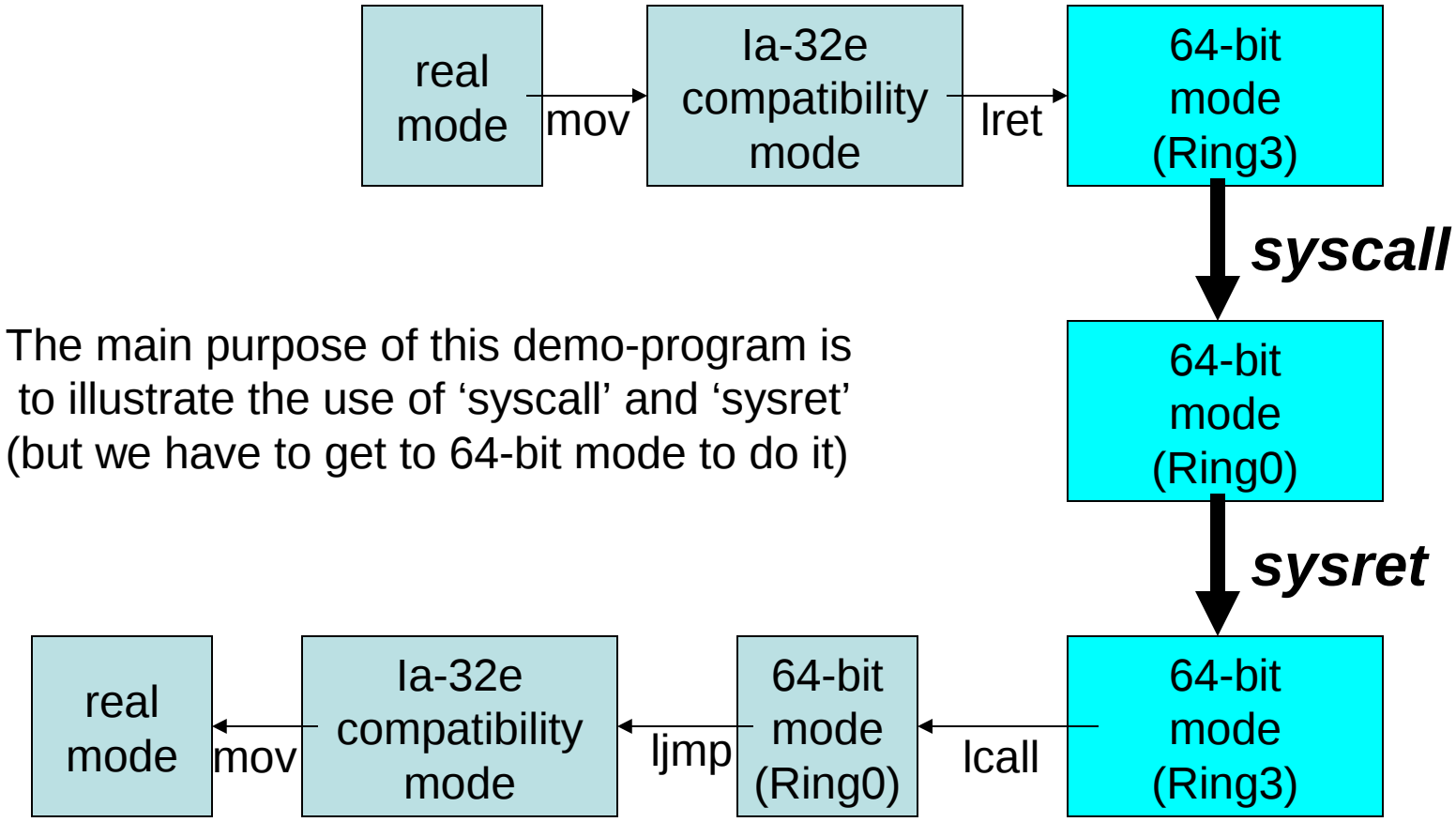


This register can be programmed by an Operating System with a bitmask that will be used by the processor to automatically ‘clear’ a specified selection of bits in the RFLAGS register when ‘syscall’ is executed (the former value of RFLAGS is saved in the general-purpose R11 register)

‘fastcall.s’

- We created a demo-program that shows the use of ‘syscall’ and ‘sysret’, indicating what setup-steps are needed:
  - Page-mapping tables (user-accessible frames)
  - Global Descriptor Table layout
  - Task-State Segment (needs ESP0 value)
  - EFER (needs LME=1 and SCE=1)
  - CR4 needs PAE=1,
  - CR3 needs physical address of page-map level4
  - CR0 needs PE=1 and PG=1

Transitions in ‘fastcall.s’

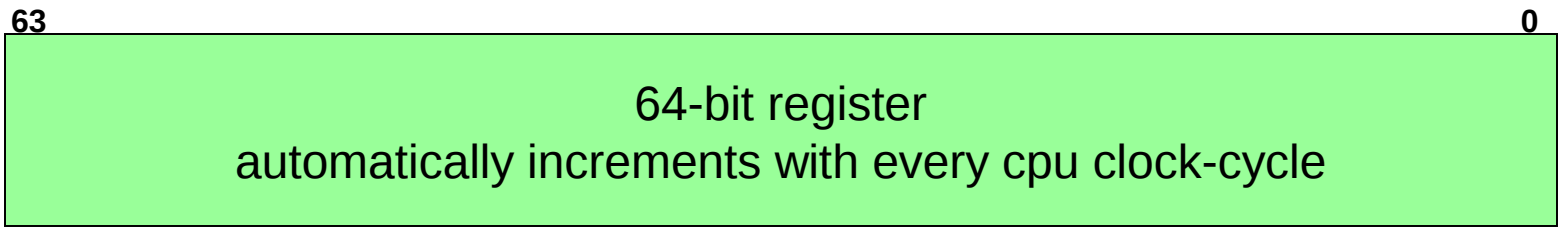


The main purpose of this demo-program is to illustrate the use of ‘syscall’ and ‘sysret’ (but we have to get to 64-bit mode to do it)

In-class exercise

- In our ‘fastcall.s’ demo-program there are two transitions from ring3 to ring0 (in one case via ‘syscall’ and in the other via ‘lcall’ through a call-gate
- Can you measure which of these is faster? (for example, by using the processor’s TimeStamp Counter, accessible with the ‘rdtsc’ instruction)

TimeStamp Counter



The ‘rdtsc’ instruction returns this register’s current value:

```
rdtsc

#  EAX = least-significant 32-bits from TSC
#  EDX = most-significant 32-bits from TSC
```

(This 64-bit register is initialized to zero at system-startup)