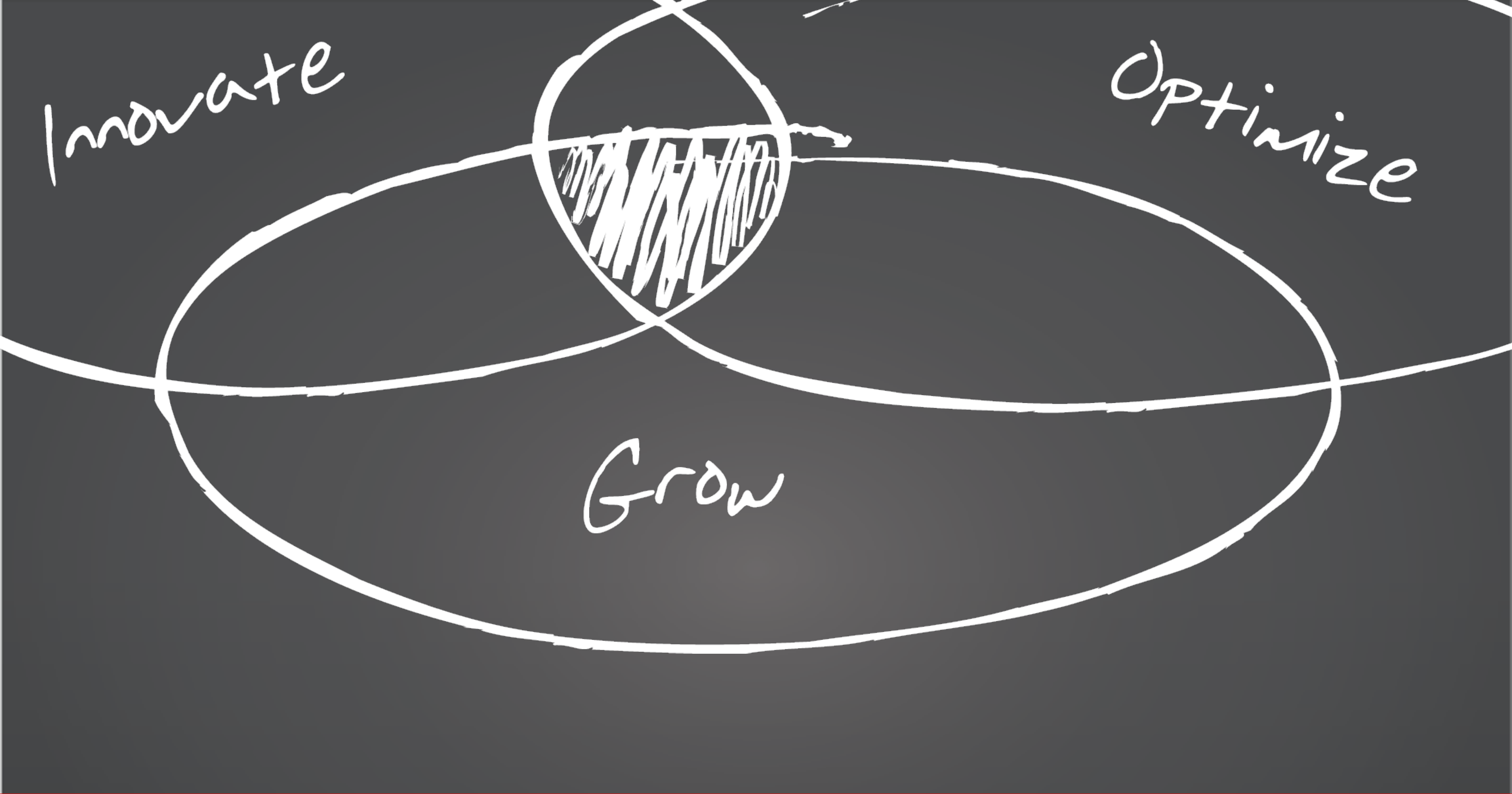
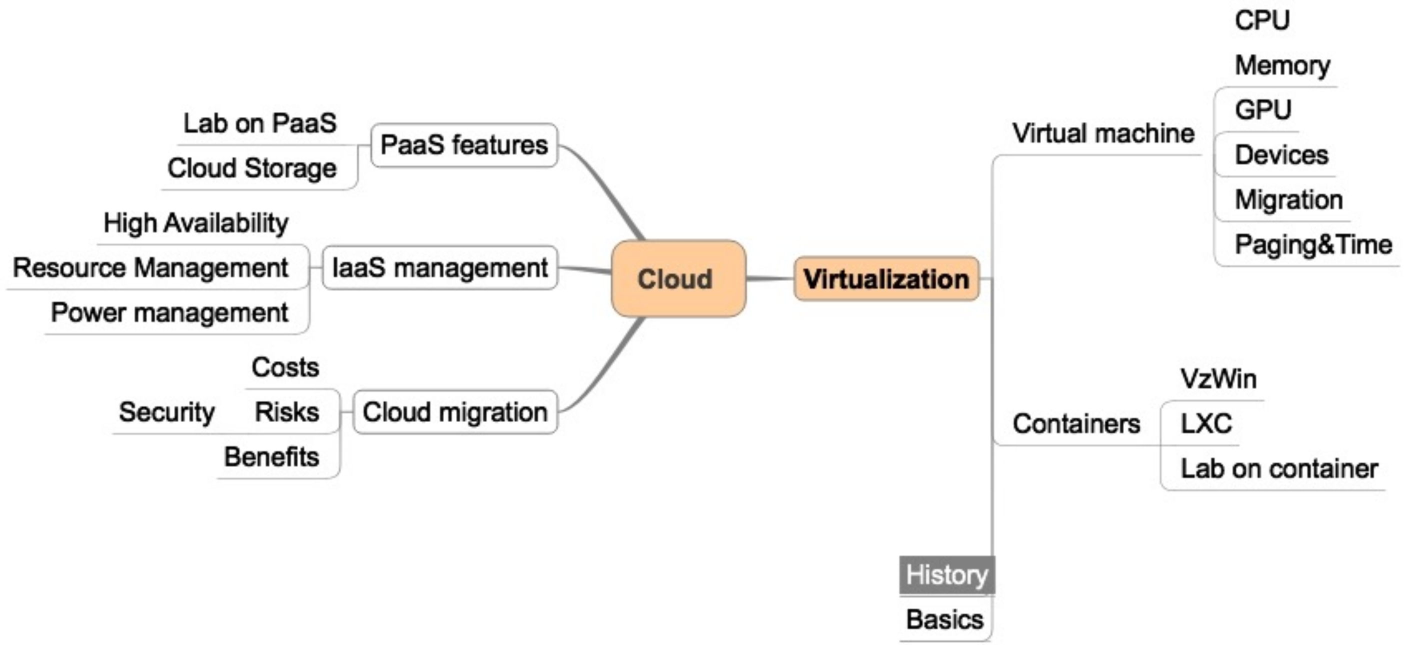


The total virtualization

Reinvent the virtual machine

Lecture #2

Course overview



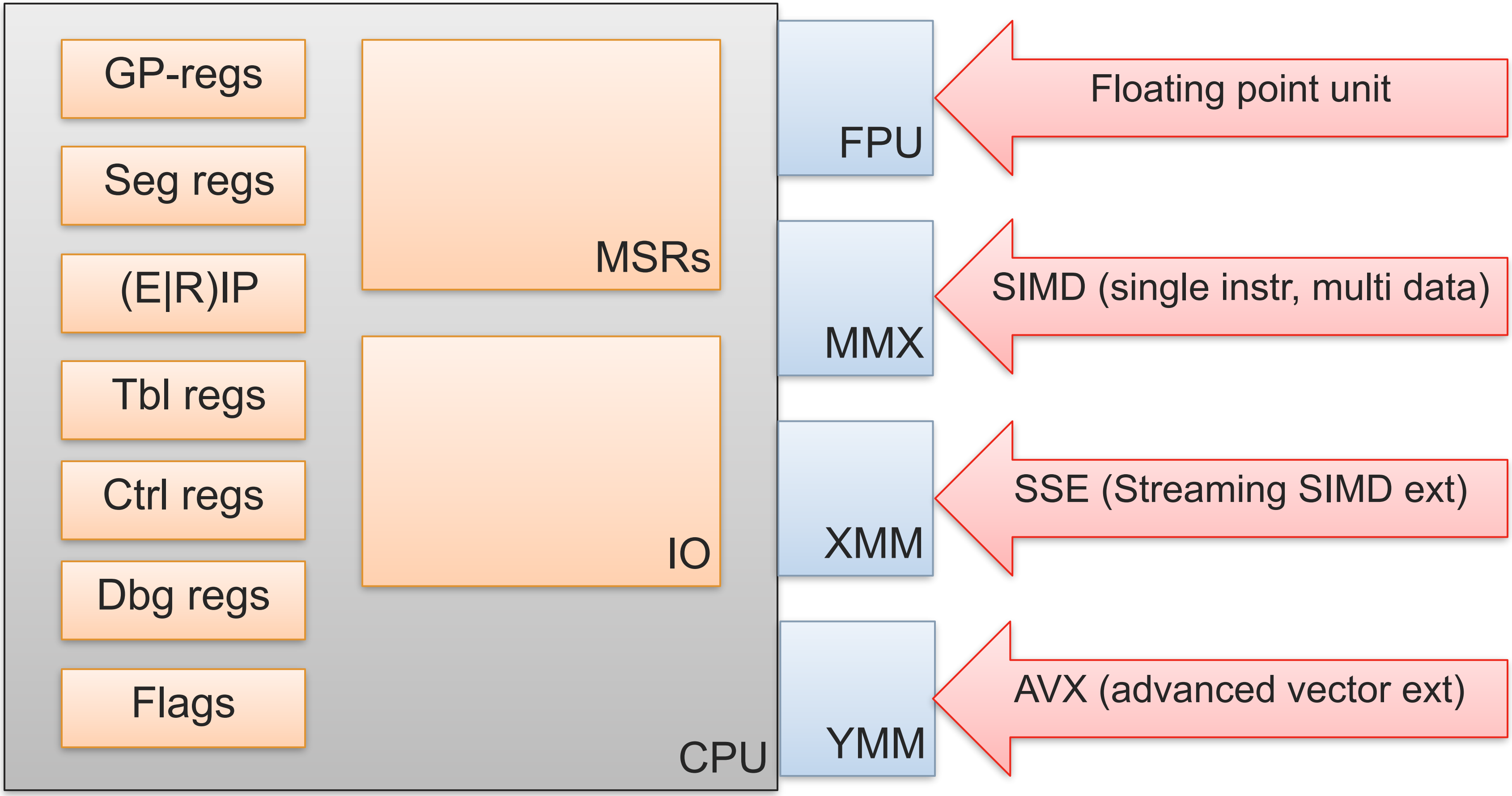
Content

- Basics of Intel x86
- Reinvent the VM
- Difficulties of Intel x86 virtualization

In order to understand the solution, one needs to understand the problem.

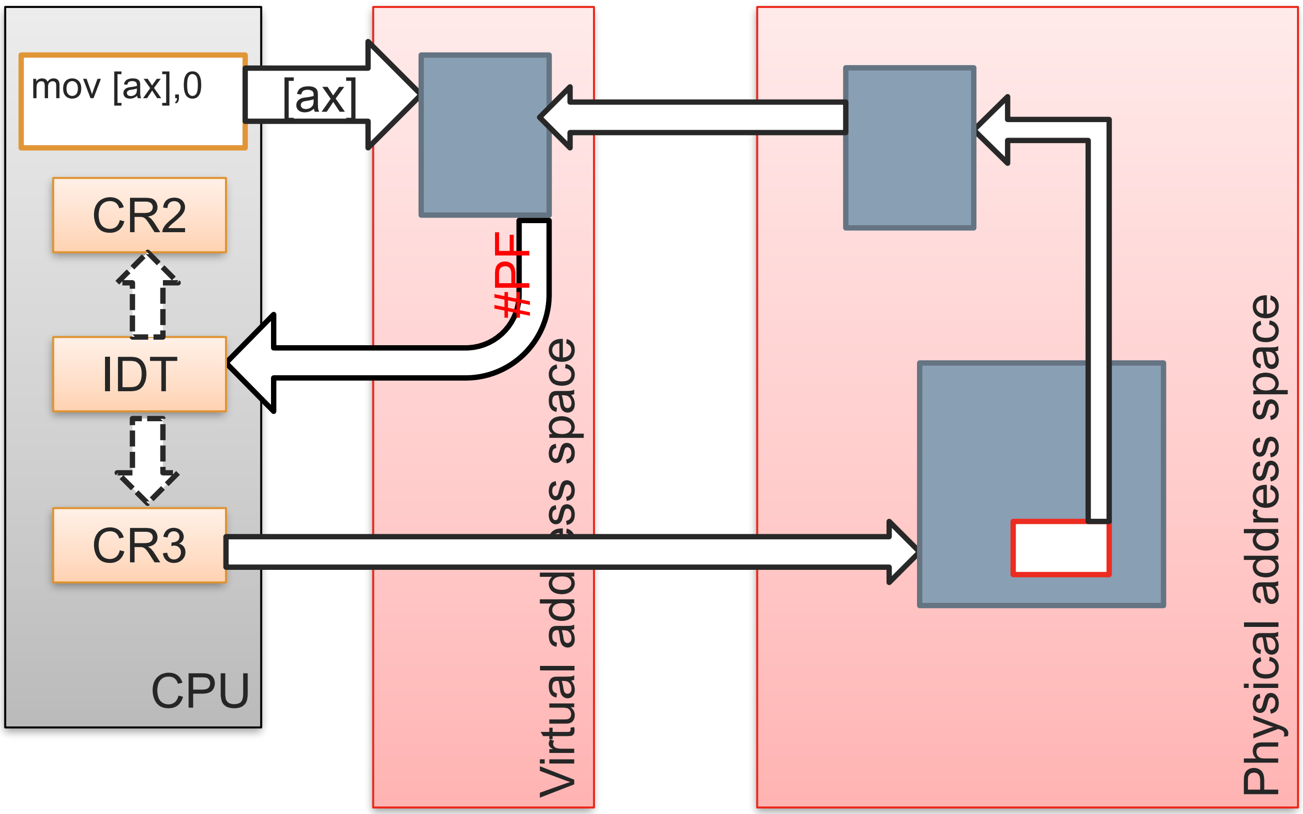
Intel x86 -> CPU

Intel x86 -> CPU



Intel x86 -> Paging

Intel x86 -> Paging



Intel x86 -> Paging

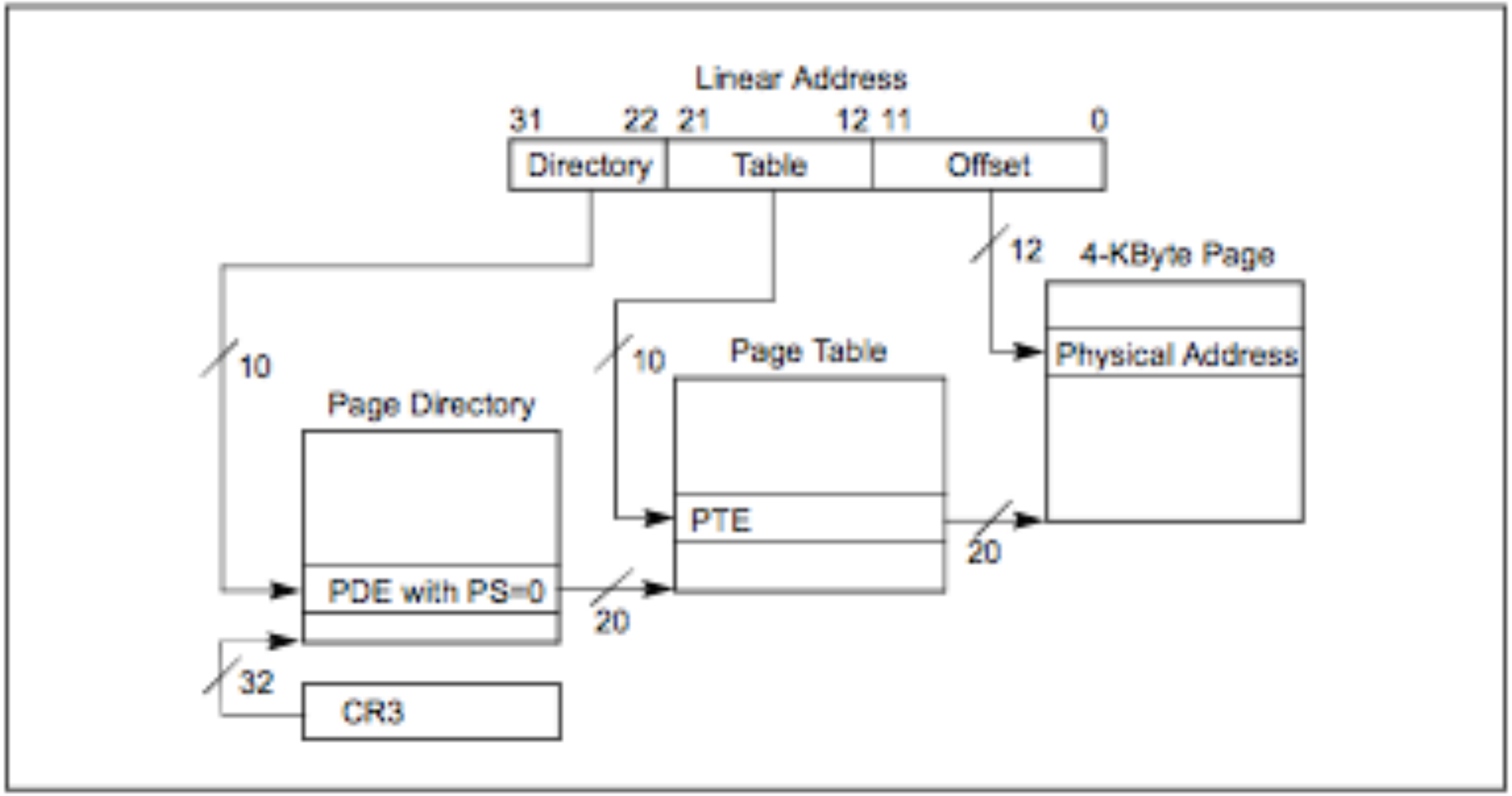


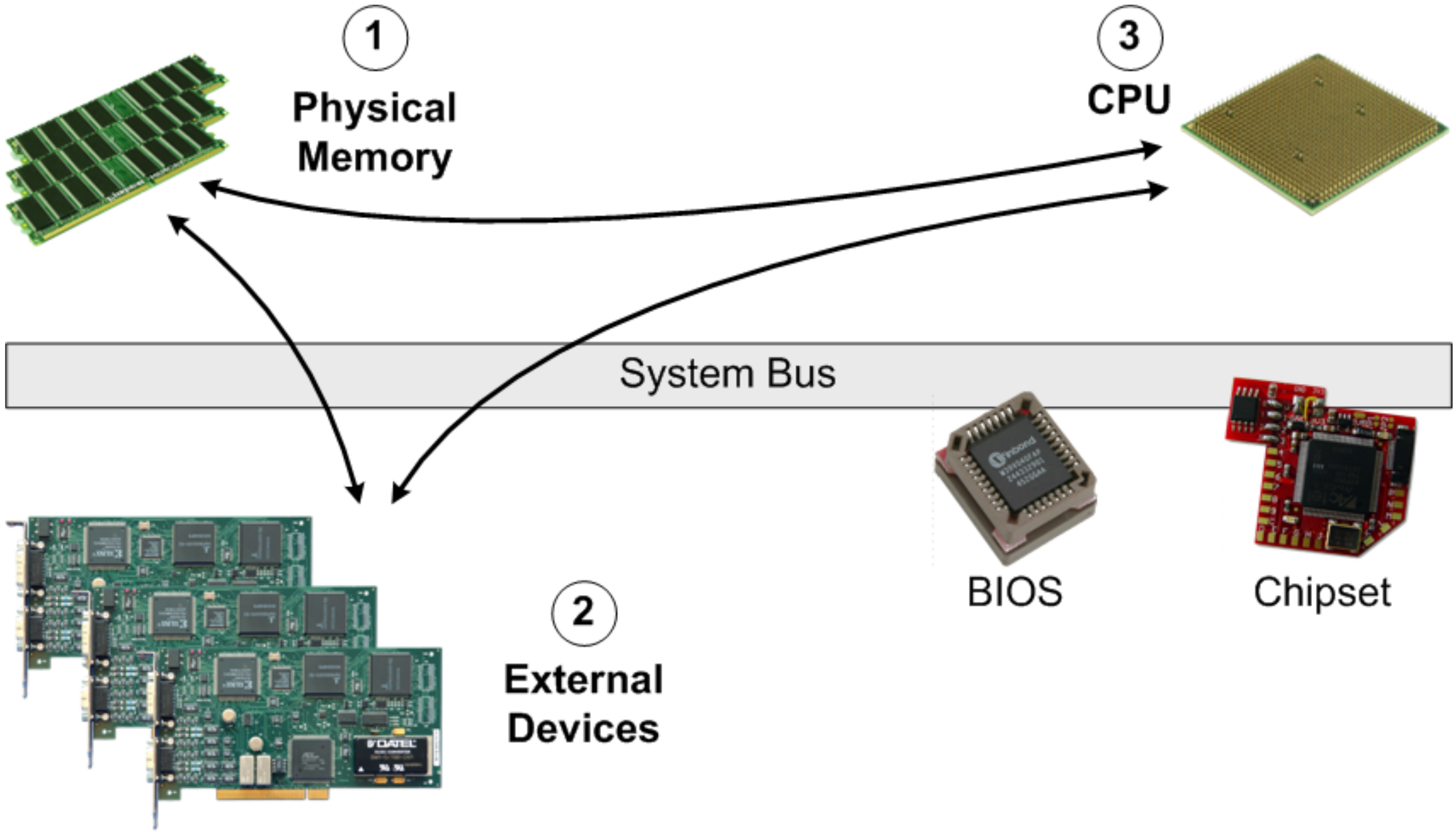
Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Intel x86 -> devices

Intel x86 -> devices

- I/O ports
- Memory mapped I/O
- DMA
- Interrupts

Intel x86 – > common view



Virtualize x86

```
class CVirtualMachine{
   Vcpu* m_pVcpu;
```

Virtualize x86: VCPU

```
class Vcpu{
   struct{
         ULONG_PTR m_GpRegs[ 8 ];
         ULONG_PTR m_Eip;
         // ….
    }RegState;

  void Add( PINSTR );
  void Mov( PINSTR );
  // ….

   PINSTR Decode( struct Vcpu* pVcpu, ULONG_PTR Addr );
    void Exec( struct Vcpu* pVcpu, PINSTR );
}
```

Virtualize x86: VCPU

```
/**
 * @brief arg0 = arg0 + arg1
 *       flags affected
 */
void Vcpu::Add( PINSTR pInstr )
{
        ULONG_PTR uArg1 = GetArg( pInstr->arg[0] );
        ULONG_PTR uArg2 = GetArg( pInstr->arg[1] );

        uArg1 += uArg2;
        SetArg( pVcpu, pInstr->arg[0], uArg1 );
        UpdateFlags( uArg1, &this->RegState.Flags );

        this->RegState.uEip += pInstr->uLen;
}
```

Virtualize x86: vmem

```
ULONG_PTR Vcpu::GetArg( PARG_DESC* pArg )
{
    switch( pArg->uType )
    {
        case ARG_REG:
            return RegState.m_GpRegs[ pArg->uID ];
        case ARG_IMM:
            return pArg->uImm;
        case ARG_MEM:
         {
             ULONG_PTR uAddr = ExtractLinAddr( pArg->pInstr );
             return ReadMem( uAddr, pArg->uSz, this->PgModel);
        }
```

Virtualize x86: vmem

```
ULONG_PTR ReadMem( ULONG_PTR uAddr, UINT sz,
                   PagingModel* pPg, GuestPhyMem* pPhyMem )
{
    if( unlikely (pPg->uModelId == NO_PAGING) )
        PhyIdx = uAddr;
    else
     {
        PhyIdx = pPg->Map( uAddr, sz, &uErrCode );
        if( GEN_EXC == uErrCode )
            GenerateException( EXCEPTION_PF, … );
    }

  PUCHAR pGuestMem = pPhyMem->GetPage( PhyIdx, sz );
    return (PULONG_PTR)pGuestMem[ uAddr & (PAGE_SIZE-1) ];
}
```

Virtualize x86. update1

```
class CVirtualMachine{
   Vcpu*           m_pVcpu;
   GuestPhyMem* m_pPhyMem;
```

Virtualize x86: vmem

```
PCHAR GuestPhyMem::GetPhyPage( ULONG_PTR uPhyAddr,
                               UINT sz )
  {
      // read file representing VM phy mem at offset
     int fd = open(“./vm.mem”, O_RDWR);

     sz = (sz + PAGE_SIZE - 1) & (PAGE_SIZE – 1);

     return mmap( NULL, sz, PROT_WRITE, MAP_SHARED, fd,
    uPhyAddr & (PAGE_SIZE -1));
  }
```

Virtualize x86: vdev

```
void Vcpu::AssignIoMem( ULONG_PTR uAddr,
                        size_t uSize, void* pCallback );

void Vcpu::AssignIRQ( UINT uIrqNo, void* pCallback );

void Vcpu::AssignIoPort( ULONG_PTR uAddr, void* pCallback );
```

Virtualize x86. update2

```
class CVirtualMachine{
   Vcpu*              m_pVcpu;
   GuestPhyMem* m_pPhyMem;

   IDE*               m_pCDRom;
   SATA*              m_pHD;
   NetCard*           m_pNet;
   Usb*               m_pUsb;

   VKeyboard*         m_pKeyBoard;
   Vmouse             m_pMouse;
```

Virtualize x86. update3

```
class CVirtualMachine{
   Vcpu*              m_pVcpu;
   GuestPhyMem*  m_pPhyMem;

   IDE*               m_pCDRom;
   SATA*              m_pHD;
   NetCard*           m_pNet;
   USB*               m_pUsb;

   VKeyboard*         m_pKeyBoard;
   Vmouse             m_pMouse;

   Video*             m_pVideo;
```

Virtualize x86: video

```
void Video::Init()
{
    AssignIoPort( 0x3BA, VideoMemIoHandler );

    AssignIoMem( 0xA0000, 32*PAGE_SIZE, VideoMemBuffer );
    …
}

Void Video::UpdateVideo()
{
   m_pVga = m_PhyMem->GetPage( 0xA0000, 32*PAGE_SIZE );
   ….
}
```

Virtualize x86. Main cycle

```
CVirtualMachine::Exec()
{
   for( ;; )
   {
       PINSTR pInstr = m_pVcpu->Decode( m_pVcpu->RegState.uEip );
       UINT    uExc = m_pVcpu->Exec( pInstr );
       if( uExc != 0xff )
           m_pVcpu->GenerateException( uExc );

  }
```

Virtualize x86: BIOS

```
char* pBios = ReadBiosFile("bios.bin");
memcpy( m_pPhyMem->GetPage(0x7c00), pBios, sz );
m_pVcpu->RegState.uEip = 0x7c00;
```

The sources of BIOS:

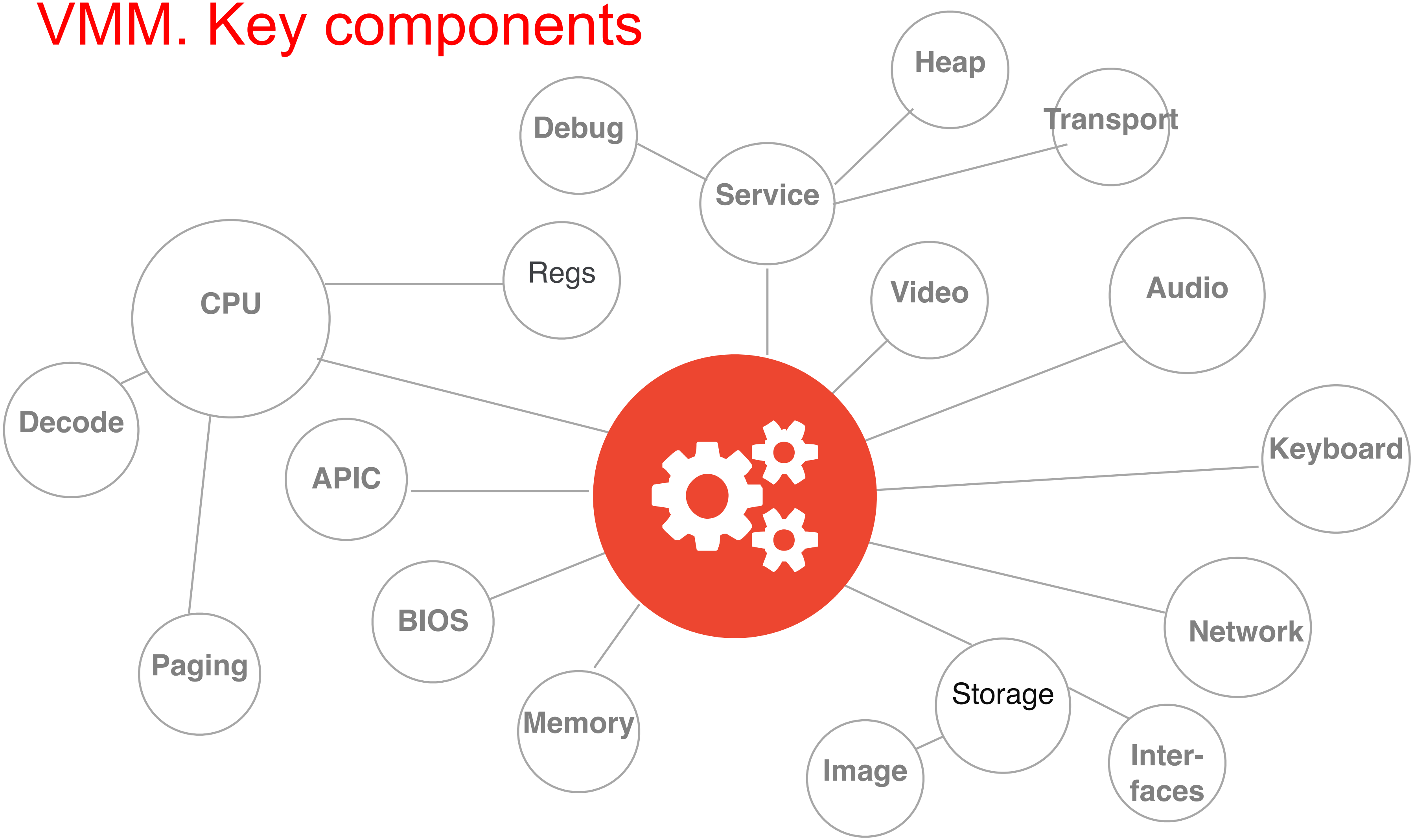
✓ Buy
✓ Write your own BIOS
✓ Use Open-source project (tinybios)

EFI!

Virtualize x86. update4

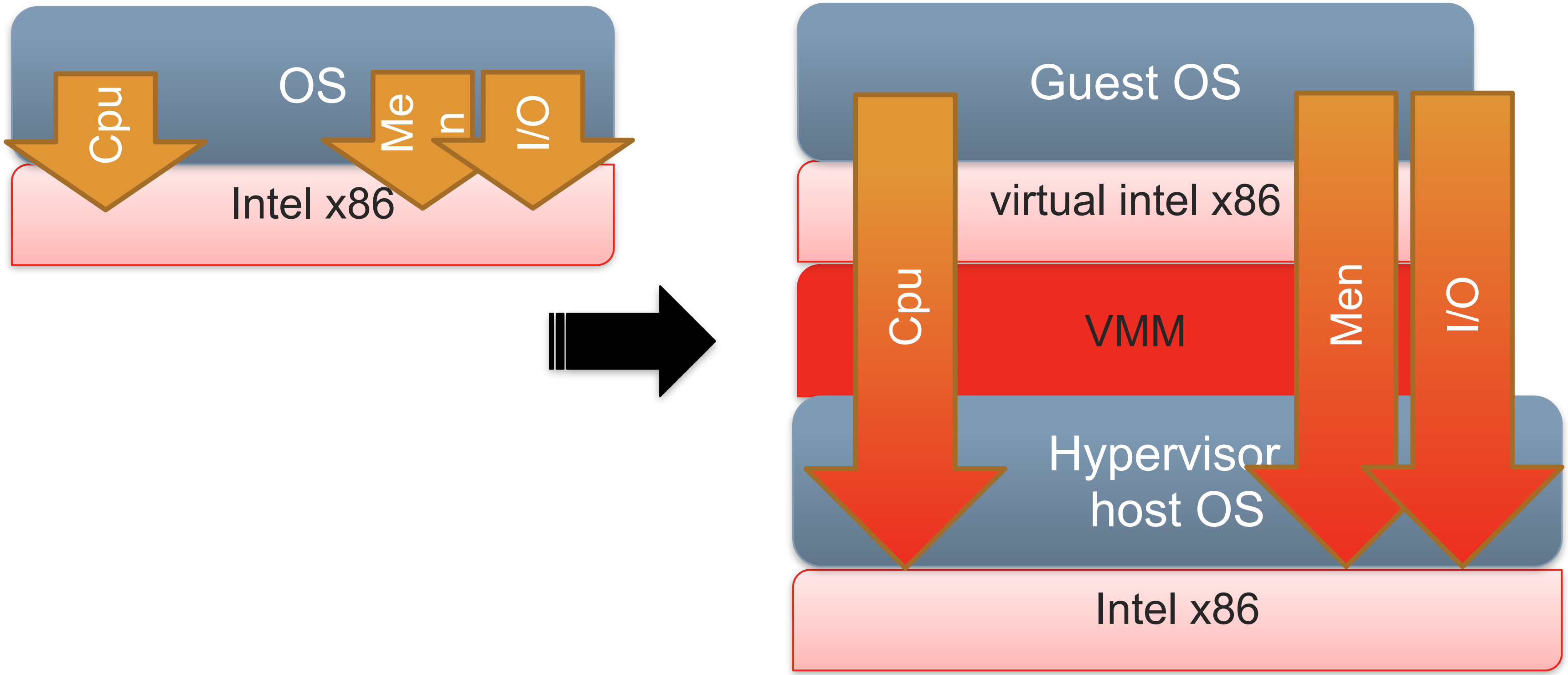
```
class CVirtualMachine{
   Vcpu*                  m_pVcpu;
   GuestPhyMem*  m_pPhyMem;

   IDE*                    m_pCDRom;
   IDE*                    m_pHD;
   NetCard*             m_pNet;
   Usb*                    m_pUsb;

   VKeyboard*        m_pKeyBoard;
   Vmouse              m_pMouse;

   Video*                m_pVideo;
   char*                  m_pBios;
}
```

VMM. Key components

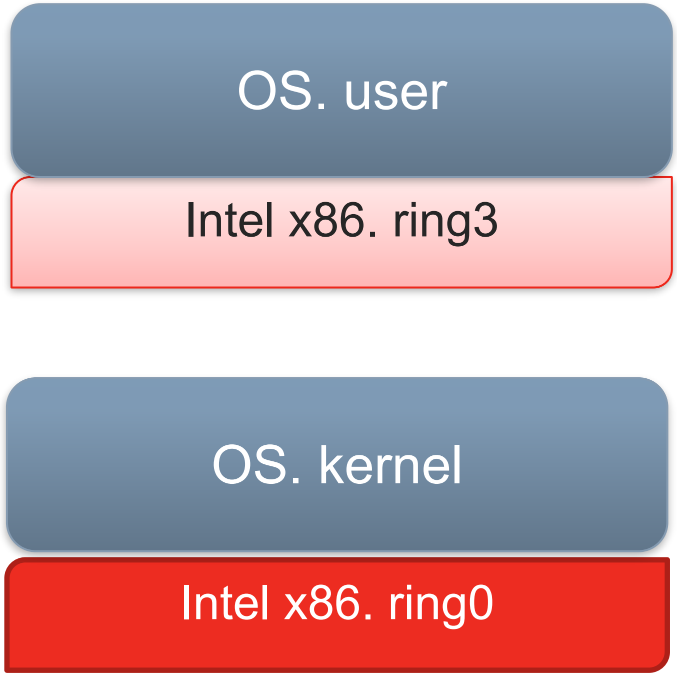


Congrats! We've just reinvented ~~the weel~~ bosch

Intel x86 virtualization



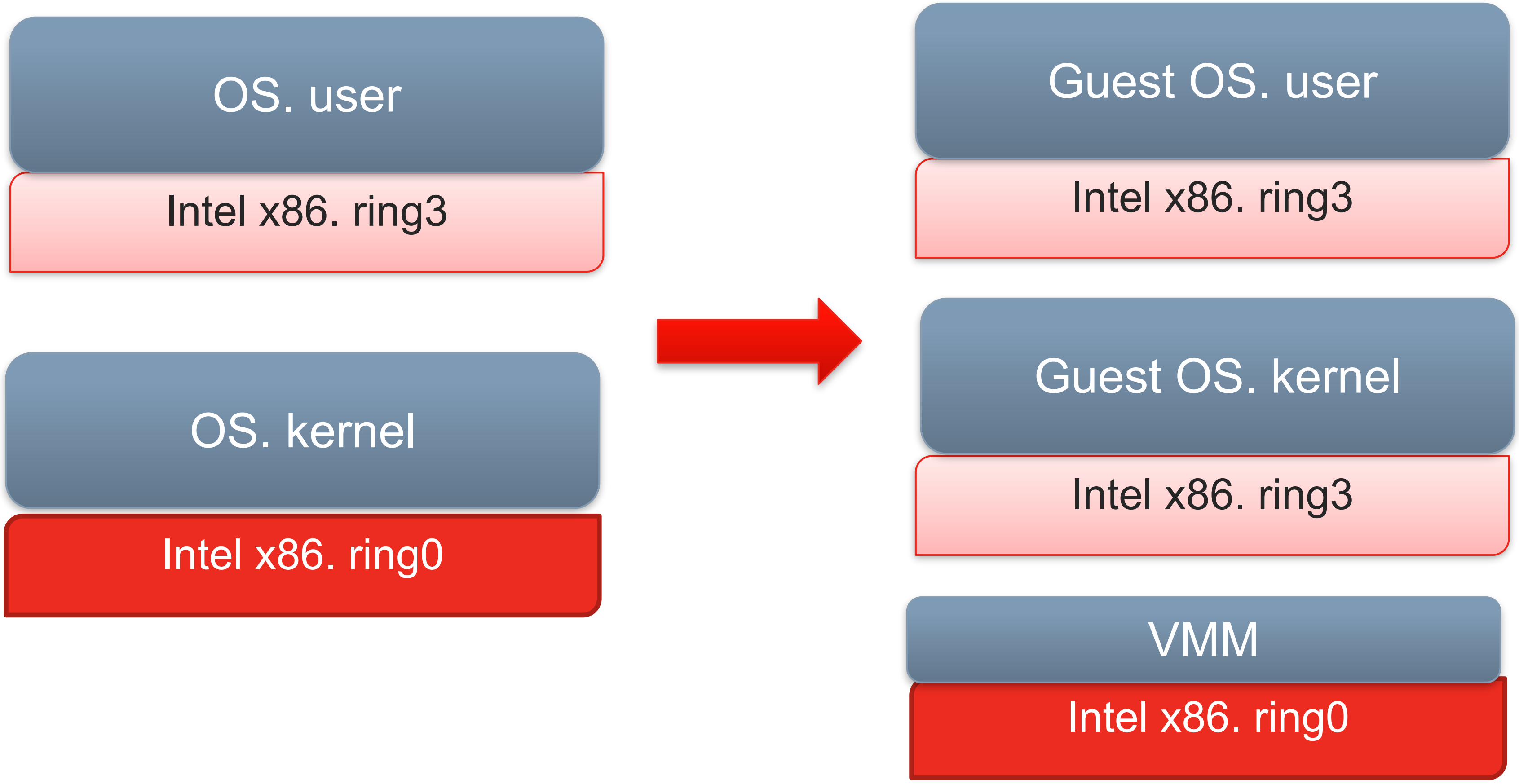
Intel x86 virtualization: de-privilege



All privilege instructions are executed in OS.kernel on ring0

Intel x86 virtualization: de-privilege

All privilege instructions causes traps to VMM on ring0



De-privilege doesn't work in x86. Why?

Intel x86 virtualization: Goldberg&Popek rules

Requirement 1 The method of executing non- privileged instructions must be roughly equivalent in both privileged and user mode.

Requirement 2 There must be a method to isolate a running VM.

Requirement 3 There must be a way to automatically signal the VMM when guest OS attempts to execute a sensitive instruction. It must also be possible for the VMM to simulate the effect of the instruction.

Sensitive instructions:

- Mode/state of machine
- Sensitive registers and/or memory locations
- Virtual memory protection
- I/O

Intel x86 virtualization

"Analysis of the Intel Pentium's ability to support a secure virtual machine monitor", J.S. Robin, C.E.Irvine, 2000

The analysis of Section 3 shows that the Intel processor is not virtualizable according to Goldberg's hardware rules.

- ✓ SGDT, SIDT, SLDT, STR
- ✓ SMSW
- ✓ PUSHF/POPF
- ✓ MOV CS/SS, reg; PUSH CS/SS;
- ✓ CALL, JMP, RET, IRET, INT N for call gates

Conclusions

The complexity is the fortune of all long-living large systems that maintain back compatibility as Intel x86 platform does. Its virtualization was impossible in theory

Questions?