Binary Translation

Jeffrey Sheldon

Classic Virtualization

- Ring Compression
- Run VM directly on hardware with reduced privileges
- Trap and emulate privileged instructions

Classic Virtualization

- Execute application code at cpl 3
- Lift kernel code from cpl 0 to cpl 1
- Point exception vectors into VMM
- When kernel performs privileged operation, trap into VMM to run interpreter.

push %cs

- Pushes current code segment selector
- Guest can tell if it has been lifted from cpl 0 to cpl 1.
- Essense: certain state cannot be hidden from guest.

popf

- EFLAGS register mixes
  - ALU flags (e.g., ZF, CF)
  - System flags (e.g., IF, VM)
- POPF sets EFLAGS to top of stack world
  - At CPL 0, all flags can be set
  - At CPL 1-3: cannot modify system flags
  - No trap, so cpl-lifting will mess up guest EFLAGS
- Essence: change in semantics, but no trap

Hidden Segment State

- CPU caches “hidden state” from in memory Descriptor on segment load
- Modify descriptor in memory
- Segment register is cached; no transparent reload
- VMM cannot context switch away from guest.

Possible Solutions

- Need instruction level control over guest
  - Code patching
    But the guest can inspect its own code
  - Prescanning pages
    Guest may jump to middle of instruction
  - Interpretation
    Very slow

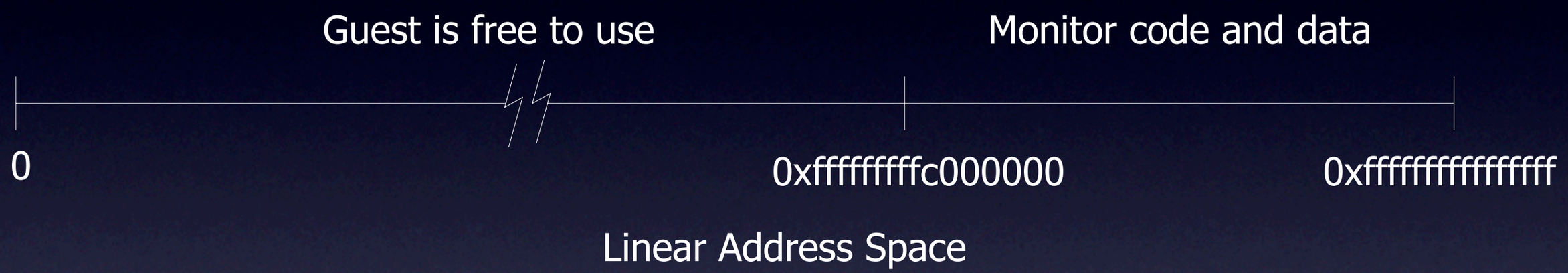
Binary Translation

- Translator input: full x86 ISA
- Translator output: subset of x86 ISA
  - Remove non-virtualizable instructions
  - Replace with VMM runtime call.

Binary Translation and Direct Exec

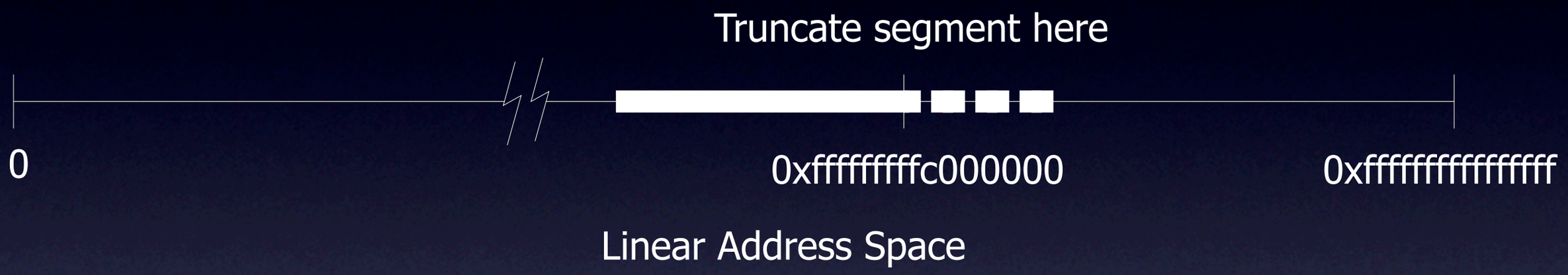
- Using BT doesn't prevent using DE as well
- Must use BT for:
  - CPL 0, 1 and 2
  - Real mode
  - Cached Segments
  - CPL 3 with IOPL 3
- Sometimes BT is desirable for performance (too many faults)

Monitor Location



- Guest can use [0, 0xfffffffffc000000) freely
- Remainder used by VMM

Protect VMM with Segment Truncation



- Cut guest segments 64MB before end of linear address space
- Fast switching, no address space switch
- Allows translator per-instruction control of guest or vmm access

Translator Properties

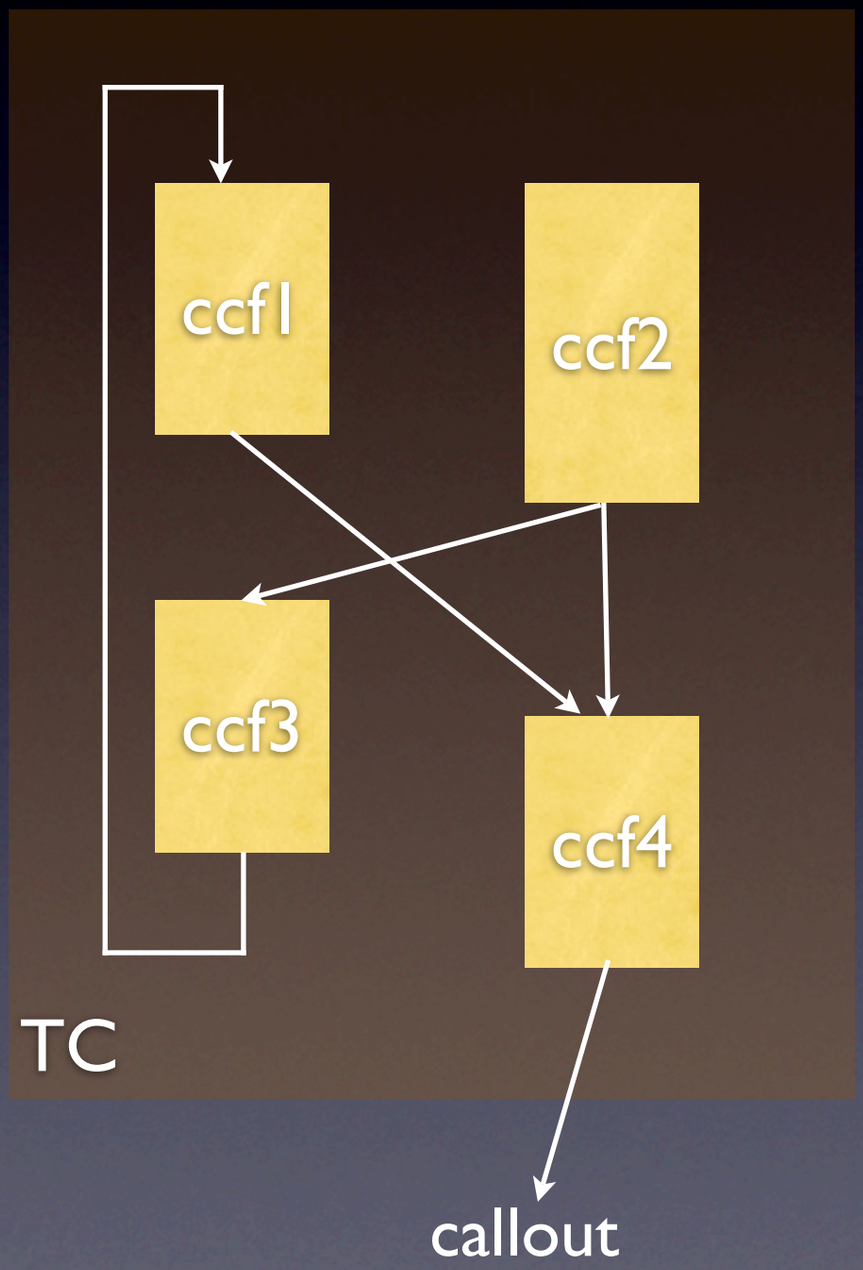
- Binary
  - Input is x86 “hex”, not source
- Dynamic
  - Interleave translation and execution
- On-demand
  - Translate only what we are about to execute
- System-level
  - Make no assumptions about guest code
  - Full virtual state recovery on fault/irq
- Adaptive

BT Framework

- Each translator invocation:
  - Consumes one translation unit (TU)
  - Produce one compiled code fragment (CCF)
- Store CCF in translation cache (TC)
  - Amortize translation costs

CCF Management

- Can "chain" CCFs together
- Can jump indirectly (via lookup table)
- May "call out" to helper functions
- May finish in a callout to translator



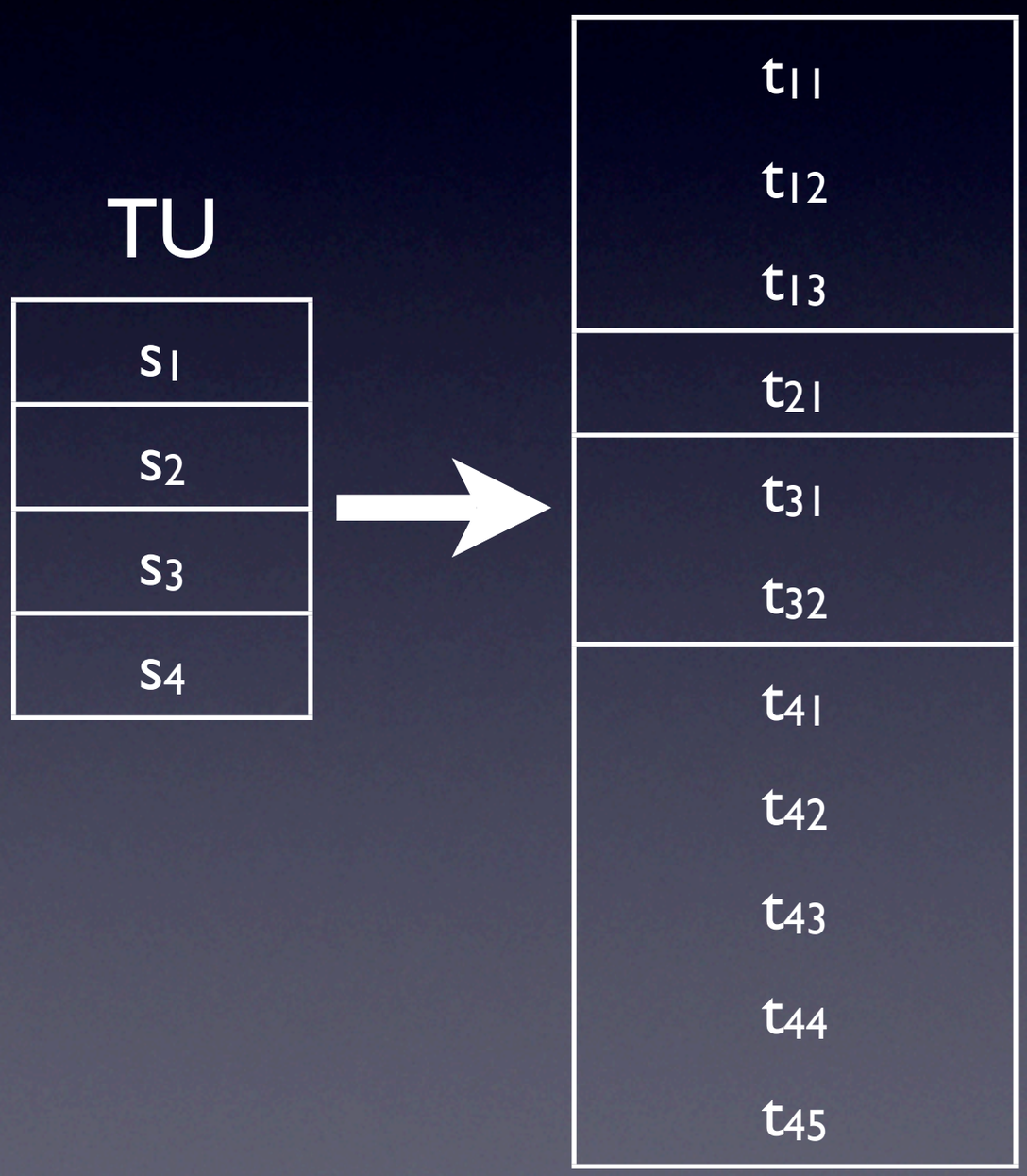
CCF Entry and Exit

- Can only enter a CCF at the start
  - Applies to both chaining and entering the TC
- Exception is returning to a CCF that called a service routine

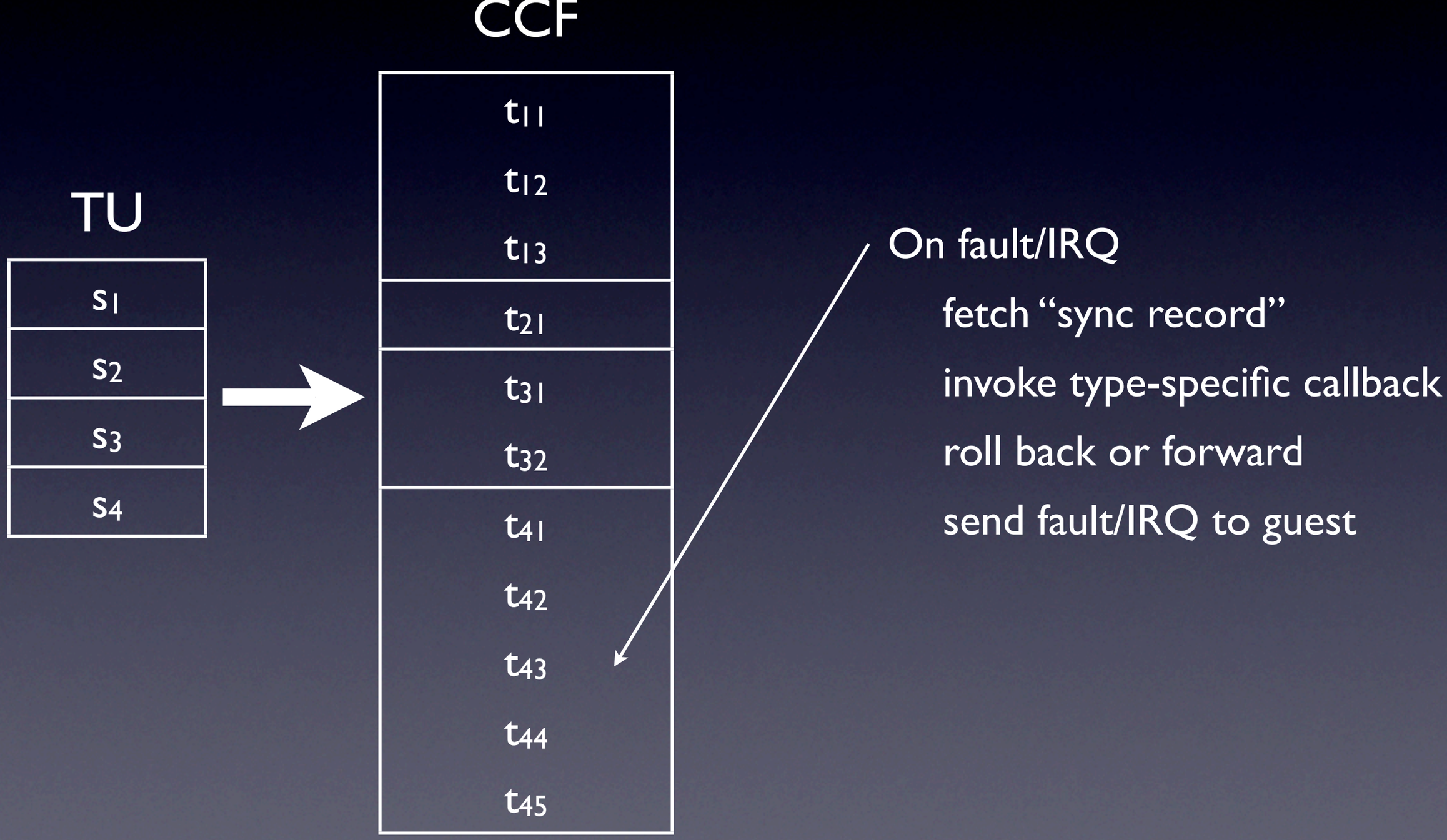
Precise Faults and IRQs

- x86 instructions are atomic — a fault or IRQ cannot occur mid-instruction
- The translator often requires many instructions to emulate the behavior of a single guest instruction
- We must have a mechanism for handling IRQs and faults that occur within a virtual instruction

Sync Regions



Sync Regions



Start with Guest Code

- VCPU program counter: 0xfffffadff8101047
- Access guest memory at this location

```
41 0f b6 c8 b8 ab aa aa 2a f7 e9 8b c2 c1
e8 1f 03 d0 8d 04 52 03 c0 3b c8 45 0f 44
e2 45 3a ca 0f 85 fe 1b fe ff ...
```

- Decoder converts to DecodeInfo (IR)
  - Opcode, operands, length, immediate, disp, etc.

Translation Unit

- TU is standard basic block (BB)
  - Typically ends in control flow
  - Maximum of 12 instructions
  - Restrictions when crossing page
  - Average length of ~5 instructions

```
41 0f b6 c8 b8 ab
aa aa 2a f7 e9 8b
c2 c1 e8 1f 03 d0
8d 04 52 03 c0 3b
c8 45 0f 44 e2 45
3a ca 0f 85 fe 1b
      fe ff
```

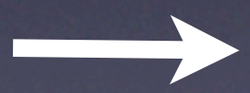
```
0xfffffadff8101047 <REXB> MOVZX %ecx,%r8b
0xfffffadff810104b  MOV   %eax,$0x2aaaaaab
0xfffffadff8101050  IMUL  %edxeax,%eax,%ecx
0xfffffadff8101052  MOV   %eax,%edx
0xfffffadff8101054  SHR   %eax,$0x1f
0xfffffadff8101057  ADD   %edx,%eax
0xfffffadff8101059  LEA   %eax,(rdx + 2*rdx)
0xfffffadff810105c  ADD   %eax,%eax
0xfffffadff810105e  CMP   %ecx,%eax
0xfffffadff8101060  CMOVZ %r12d,%r10d
0xfffffadff8101064 <REXRB> CMP   %r9b,%r10b
0xfffffadff8101067  JNZ   -0x1e402 ;0xfffffadff80e2c6b
```

IDENT case

- Most instructions require no change
- Translate by “memcopy”

TU

```
<REXB> MOVZX %ecx,%r8b
 MOV    %eax,$0x2aaaaaab
 IMUL   %edxeax,%eax,%ecx
 MOV    %eax,%edx
 SHR    %eax,$0x1f
 ADD    %edx,%eax
 LEA    %eax,(rdx + 2*rdx)
 ADD    %eax,%eax
 CMP    %ecx,%eax
 CMOVZ  %r12d,%r10d
<REXRB> CMP    %r9b,%r10b
 JNZ    -0x1e402
```



CCF

```
0xfffffffffe491ef0 <REXB> MOVZX %ecx,%r8b
0xfffffffffe491ef4  MOV    %eax,$0x2aaaaaab
0xfffffffffe491ef9  IMUL   %edxeax,%eax,%ecx
0xfffffffffe491efb  MOV    %eax,%edx
0xfffffffffe491efd  SHR    %eax,$0x1f
0xfffffffffe491f00  ADD    %edx,%eax
0xfffffffffe491f02  LEA    %eax,(rdx + 2*rdx)
0xfffffffffe491f05  ADD    %eax,%eax
0xfffffffffe491f07  CMP    %ecx,%eax
0xfffffffffe491f09  CMOVZ  %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP    %r9b,%r10b
0xfffffffffe491f10  JNZ Translation
```

Non-IDENT translations

- All control flow — as translator relocates
- in/out device access
- Privileged instructions (popf, cli, movcr)
- Semi-privileged instructions (sgdt, rdtsc)
- Segment operations
- Access to special memory/addresses

JNZ translation

```
                        ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JNZ ChainHelper0
0xfffffffffe491f16   JMP ChainHelper1
```

ChainHelper0	`int3`
	`0x0e` *(CHAINJCC_TAKEN)*
	`0xfffffadff80e2c6b` *(target)*
	`0xfffffffffe491f10` *(TCA)*
ChainHelper1	`int3`
	`0x0d` *(CHAINJCC)*
	`0xfffffffffe491f1b` *(target)*
	`0xfffffffffe491f16` *(TCA)*
	`...`

JNZ translation

```
                    ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JNZ  ChainHelper0
0xfffffffffe491f16   JMP  ChainHelper1
```

```
ChainHelper0        int3
                    0x0e  (CHAINJCC_TAKEN)
                    0xfffffadff80e2c6b  (target)
                    0xfffffffffe491f10  (TCA)
ChainHelper1        int3
                    0x0d  (CHAINJCC)
                    0xfffffffffe491f1b  (target)
                    0xfffffffffe491f16  (TCA)
                    ...
```

JNZ translation

```
                    ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JNZ  ChainHelper0
0xfffffffffe491f16   JMP  ChainHelper1
```

```
ChainHelper0        int3
                    0x0e  (CHAINJCC_TAKEN)
                    0xfffffadff80e2c6b  (target)
                    0xfffffffffe491f10  (TCA)
ChainHelper1        int3
                    0x0d  (CHAINJCC)
                    0xfffffffffe491f1b  (target)
                    0xfffffffffe491f16  (TCA)
                    ...
```

Invoke Translator

JNZ translation

```
                    ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JNZ  ChainHelper0
0xfffffffffe491f16   JMP  ChainHelper1
```

```
ChainHelper0        int3
                    0x0e  (CHAINJCC_TAKEN)
                    0xfffffadff80e2c6b  (target)
                    0xfffffffffe491f10  (TCA)
ChainHelper1        int3
                    0x0d  (CHAINJCC)
                    0xfffffffffe491f1b  (target)
                    0xfffffffffe491f16  (TCA)
                    ...
```

Invoke Translator

Assume no translation exists

JNZ translation

```
                        ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JNZ  ChainHelper0
0xfffffffffe491f16   JMP  ChainHelper1
```

← Swap condition

```
ChainHelper0        int3
                    0x0e  (CHAINJCC_TAKEN)
                    0xfffffadff80e2c6b  (target)
                    0xfffffffffe491f10  (TCA)
ChainHelper1        int3
                    0x0d  (CHAINJCC)
                    0xfffffffffe491f1b  (target)
                    0xfffffffffe491f16  (TCA)
                    ...
```

JNZ translation

```
                        ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JMP ChainHelper1
0xfffffffffe491f16   JNZ ChainHelper0
```

← Swap condition

```
ChainHelper0        int3
                    0x0e (CHAINJCC_TAKEN)
                    0xfffffadff80e2c6b (target)
                    0xfffffffffe491f10 (TCA)
ChainHelper1        int3
                    0x0d (CHAINJCC)
                    0xfffffffffe491f1b (target)
                    0xfffffffffe491f16 (TCA)
                    ...
```

JNZ translation

```
                        ...
0xffffffffe491f09   CMOVZ %r12d,%r10d
0xffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xffffffffe491f10   JZ   ChainHelper1
0xffffffffe491f16   JMP  ChainHelper0
```

← Swap condition

```
ChainHelper0        int3
                    0x0e  (CHAINJCC_TAKEN)
                    0xfffffadff80e2c6b  (target)
                    0xffffffffe491f10  (TCA)
ChainHelper1        int3
                    0x0d  (CHAINJCC)
                    0xffffffffe491f1b  (target)
                    0xffffffffe491f16  (TCA)
                    ...
```

JNZ translation

```
                        ...
0xffffffffe491f09   CMOVZ %r12d,%r10d
0xffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xffffffffe491f10   JZ   ChainHelper1
0xffffffffe491f16   JMP  ChainHelper0
```

← "Trim" the CCF

```
ChainHelper0        int3
                    0x0e  (CHAINJCC_TAKEN)
                    0xfffffadff80e2c6b  (target)
                    0xffffffffe491f10  (TCA)
ChainHelper1        int3
                    0x0d  (CHAINJCC)
                    0xffffffffe491f1b  (target)
                    0xffffffffe491f16  (TCA)
                    ...
```

JNZ translation

```
                        ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JZ   ChainHelper1
0xfffffffffe491f16   JMP  ChainHelper0
```

← "Trim" the CCF

```
ChainHelper0          int3
                      0x0e  (CHAINJCC_TAKEN)
                      0xfffffadff80e2c6b  (target)
                      0xfffffffffe491f10  (TCA)
ChainHelper1          int3
                      0x0d  (CHAINJCC)
                      0xfffffffffe491f1b  (target)
                      0xfffffffffe491f16  (TCA)
                      ...
```

JNZ translation

```
                    ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JZ   ChainHelper1
```

ChainHelper0	*Free for reuse*
ChainHelper1	**`int3`** **`0x0d`** *(CHAINJCC)* **`0xfffffffffe491f1b`** *(target)* **`0xfffffffffe491f16`** *(TCA)* **`...`**

JNZ translation

	...
0xfffffffffe491f09	CMOVZ %r12d,%r10d
0xfffffffffe491f0d	<REXRB> CMP %r9b,%r10b
0xfffffffffe491f10	JZ *ChainHelper1*
0xfffffffffe491f16	*Next CCF for 0xfffffadff80e2c6b*
	...

ChainHelper0	*Free for reuse*
ChainHelper1	int3 0x0d *(CHAINJCC)* 0xfffffffffe491f1b *(target)* 0xfffffffffe491f16 *(TCA)* ...

JNZ translation

	`...`
`0xfffffffffe491f09`	`CMOVZ %r12d,%r10d`
`0xfffffffffe491f0d`	`<REXRB> CMP %r9b,%r10b`
`0xfffffffffe491f10`	`JZ` *ChainHelper1*
`0xfffffffffe491f16`	*Next CCF for 0xfffffadff80e2c6b*
	`...`

ChainHelper0	*Free for reuse*
ChainHelper1	`int3`
	`0x0d` *(CHAINJCC)*
	`0xfffffffffe491f1b` *(target)*
	`0xfffffffffe491f16` *(TCA)*
	`...`

JNZ translation

```
                          ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JZ   ChainHelper1
0xfffffffffe491f16  Next CCF for 0xfffffadff80e2c6b
                          ...
```

ChainHelper0	*Free for reuse*

ChainHelper1	`int3`
	`0x0d` *(CHAINJCC)*
	`0xfffffffffe491f1b` *(target)*
	`0xfffffffffe491f16` *(TCA)*
	`...`

Invoke Translator

JNZ translation

```
                          ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JZ   ChainHelper1
0xfffffffffe491f16  Next CCF for 0xfffffadff80e2c6b
                          ...
```

```
ChainHelper0        Free for reuse

ChainHelper1        int3
                    0x0d (CHAINJCC)
                    0xfffffffffe491f1b (target)
                    0xfffffffffe491f16 (TCA)
                    ...
```

Invoke Translator

But what if a translation already exists?

SrcKey

- Handle used to find CCFs in the TC
- SrcKey computed from current VCPU state
- Contains: RIP, CS base, CS limit, code size, mode, holdoff, BPN

JNZ translation

```
                          ...
0xffffffffe491f09   CMOVZ %r12d,%r10d
0xffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xffffffffe491f10   JZ    CCF for 0xfffffadff810106b
0xffffffffe491f16  Next CCF for 0xfffffadff80e2c6b
                          ...
```

JNZ translation

```
                        ...
0xfffffffffe491f09   CMOVZ %r12d,%r10d
0xfffffffffe491f0d <REXRB> CMP   %r9b,%r10b
0xfffffffffe491f10   JZ   CCF for 0xfffffadff8110106b
0xfffffffffe491f16  Next CCF for 0xfffffadff80e2c6b
                        ...
```

- In the optimal case (as above) Jcc gets translated to just a single Jcc instruction (sometimes with reverse polarity)

CCF Entry and Exit

- Requires a convention for guest state
- Guest ALU flags are live in hardware
- Guest registers are live in hardware registers
- Except %r11, the “reserved” register
  - Guest %r11 is stored in VCPU
  - Hardware %r11 is dead

%r11

- Many translations need a scratch register
- They can save and restore a register with guest state
- But having a free one helps keep things short and fast

Example: %rip-rel

```
 OR    (r10),%rax
<REXR> MOVZX %r10d,0x140da2(%rip) ;0xffffff8000118f7a1
<OP> NOP
 MOV   %rcx,%r9
 MOV   %rax,%rsi
 SUB   %rcx,0x127c84(%rip) ;0xffffff80001176698
 MUL   %rdxrax,%rax,%rcx
 MOVZX %ecx,0x1a(r9)
<OP> SHR   %cx,$0xc
```

Instructions access memory relative to the instructions' addresses.

Example: %rip-rel

```
 OR     (r10),%rax
<REXR> MOVZX %r10d,0x140da2(%rip) ;0xffffff8000118f7a1
<OP> NOP
 MOV    %rcx,%r9
 MOV    %rax,%rsi
 SUB    %rcx,0x127c84(%rip) ;0xffffff80001176698
 MUL    %rdxrax,%rax,%rcx
 MOVZX  %ecx,0x1a(r9)
<OP> SHR    %cx,$0xc
```

Instructions access memory relative to the instructions' addresses.

Load guest address into %r11 and use that as base. →

Do access relative to previous %r11 value →

```
IDENT              OR     (r10),%rax
REEMIT_RSVD_REG    MOV    %r11,$0xffffff8000118f7a1
                  <REXRB> MOVZX %r10d,(r11)
IDENT             <OP> NOP
                   MOV    %rcx,%r9
                   MOV    %rax,%rsi
REEMIT_RSVD_REG    SUB    %rcx,-0x19109(r11)
IDENT              MUL    %rdxrax,%rax,%rcx
                   MOVZX  %ecx,0x1a(r9)
                  <OP> SHR    %cx,$0xc
```

Example: Guest uses %r11

```
MOV    %r10,0x148579(%rip)
MOV    %r11d,%edx
MOV    %r9,%rcx
MOV    %rax,(r10)
MOV    %r8,%rax
SHR    %r8,$0xc
AND    %r8d,$0xffffffff
CMP    %r11,%r8
JBE    0x46
```

Example: Guest uses %r11

```
MOV    %r10,0x148579(%rip)

MOV    %r11d,%edx


MOV    %r9,%rcx
MOV    %rax,(r10)
MOV    %r8,%rax
SHR    %r8,$0xc
AND    %r8d,$0xffffffff
CMP    %r11,%r8
JBE    0x46
```

Example: Guest uses %r11

```
MOV    %r10,0x148579(%rip)

MOV    %r11d,%edx

MOV    %r9,%rcx
MOV    %rax,(r10)
MOV    %r8,%rax
SHR    %r8,$0xc
AND    %r8d,$0xffffffff
CMP    %r11,%r8
JBE    0x46
```

```
REEMIT_RSVD_REG   MOV    %r11,$0xffffff80001196ec0
                  MOV    %r10,(r11)
REEMIT_RSVD_REG   MOV    %r11d,%edx
                 <GS> MOV -0x23ef4d2(%rip),%r11 ; VR11
IDENT             MOV    %r9,%rcx
                  MOV    %rax,(r10)
                  MOV    %r8,%rax
                  SHR    %r8,$0xc
                  AND    %r8d,$0xffffffff
REEMIT_RSVD_REG   CMP    %r11,%r8
JCC               JBE    0x1306e5
                  JMP    0x13310d
```

Example: Guest uses %r11

Load %rip-rel address as before

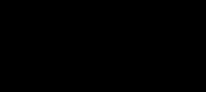
```
MOV    %r10,0x148579(%rip)

MOV    %r11d,%edx

MOV    %r9,%rcx
MOV    %rax,(r10)
MOV    %r8,%rax
SHR    %r8,$0xc
AND    %r8d,$0xffffffff
CMP    %r11,%r8
JBE    0x46
```

```
REEMIT_RSVD_REG   MOV    %r11,$0xffffff80001196ec0
                  MOV    %r10,(r11)
REEMIT_RSVD_REG   MOV    %r11d,%edx
                  <GS> MOV -0x23ef4d2(%rip),%r11 ; VR11
IDENT             MOV    %r9,%rcx
                  MOV    %rax,(r10)
                  MOV    %r8,%rax
                  SHR    %r8,$0xc
                  AND    %r8d,$0xffffffff
REEMIT_RSVD_REG   CMP    %r11,%r8
JCC               JBE    0x1306e5
                  JMP    0x13310d
```

Example: Guest uses %r11

Load guest %r11 into hardware %r11 and store into VCPU

```
MOV    %r10,0x148579(%rip)
MOV    %r11d,%edx
MOV    %r9,%rcx
MOV    %rax,(r10)
MOV    %r8,%rax
SHR    %r8,$0xc
AND    %r8d,$0xffffffff
CMP    %r11,%r8
JBE    0x46
```



```
REEMIT_RSVD_REG   MOV    %r11,$0xffffff80001196ec0
                  MOV    %r10,(r11)
REEMIT_RSVD_REG   MOV    %r11d,%edx
                  <GS> MOV -0x23ef4d2(%rip),%r11 ; VR11
IDENT             MOV    %r9,%rcx
                  MOV    %rax,(r10)
                  MOV    %r8,%rax
                  SHR    %r8,$0xc
                  AND    %r8d,$0xffffffff
REEMIT_RSVD_REG   CMP    %r11,%r8
JCC               JBE    0x1306e5
                  JMP    0x13310d
```

Example: Guest uses %r11

Use hardware %r11 directly

```
MOV    %r10,0x148579(%rip)

MOV    %r11d,%edx

MOV    %r9,%rcx
MOV    %rax,(r10)
MOV    %r8,%rax
SHR    %r8,$0xc
AND    %r8d,$0xffffffff
CMP    %r11,%r8
JBE    0x46
```

```
REEMIT_RSVD_REG   MOV    %r11,$0xffffff80001196ec0
                  MOV    %r10,(r11)
REEMIT_RSVD_REG   MOV    %r11d,%edx
                  <GS> MOV -0x23ef4d2(%rip),%r11 ; VR11
IDENT             MOV    %r9,%rcx
                  MOV    %rax,(r10)
                  MOV    %r8,%rax
                  SHR    %r8,$0xc
                  AND    %r8d,$0xffffffff
REEMIT_RSVD_REG   CMP    %r11,%r8
JCC               JBE    0x1306e5
                  JMP    0x13310d
```

Example: CLI

- Can’t clear hardware interrupt flag
- Need to clear software flag in VCPU
- Can’t corrupt flags
- Can’t access VCPU using %ds, must use %gs override

Example: CLI

CLI

```
PRIV     MOV    %r11,%rax
         LAHF   %ah
         SETO   %al
        <GS> AND    -0x21c5c79(%rip),$-0x201 ; VC->_flags._gp
         CMP    %al,$-0x7f
         SAHF   %ah
         MOV    %rax,%r11
CHAIN0   BPDISP 13 BPDISPATCH_CHAIN
```

Example: CLI

```
<OP> NOP
 CLI
 MOV    %rax,0x68(rcx)
 TEST   %rax,%rax
 MOV    (rdx),%rax
 JZ     0x1c
```

```
IDENT            <OP> NOP
PRIV_SCR_OPTIM    MOV    %r11,%rax
                  LAHF   %ah
                  SETO   %al
                 <GS> AND    -0x2254d93(%rip),$-0x201
                                        ; VC->_flags._gp
IDENT_I_RSVD_F    MOV    %rax,0x68(rcx)
                  TEST   %rax,%rax
IDENT             MOV    (rdx),%rax
JCC               JZ     0x2b702e
                  JMP    0x2bd5eb
```

Trace Handling

- Most translations run "at speed"
- Key exception: writes to traced memory
  - #PF
  - Decode and interpret instruction
  - Fire trace callbacks
  - Resume execution
  - Can take 1000s of cycles in a VM vs. ~1 cycle natively

Adaptive BT

- Detect and track offending guest instructions
- Replace existing (and future) IDENT translations with special ones
  - Avoid #PF cost
  - No run-time redecoding
  - Fast resumption of guest
- Order of magnitude trace cost reduction

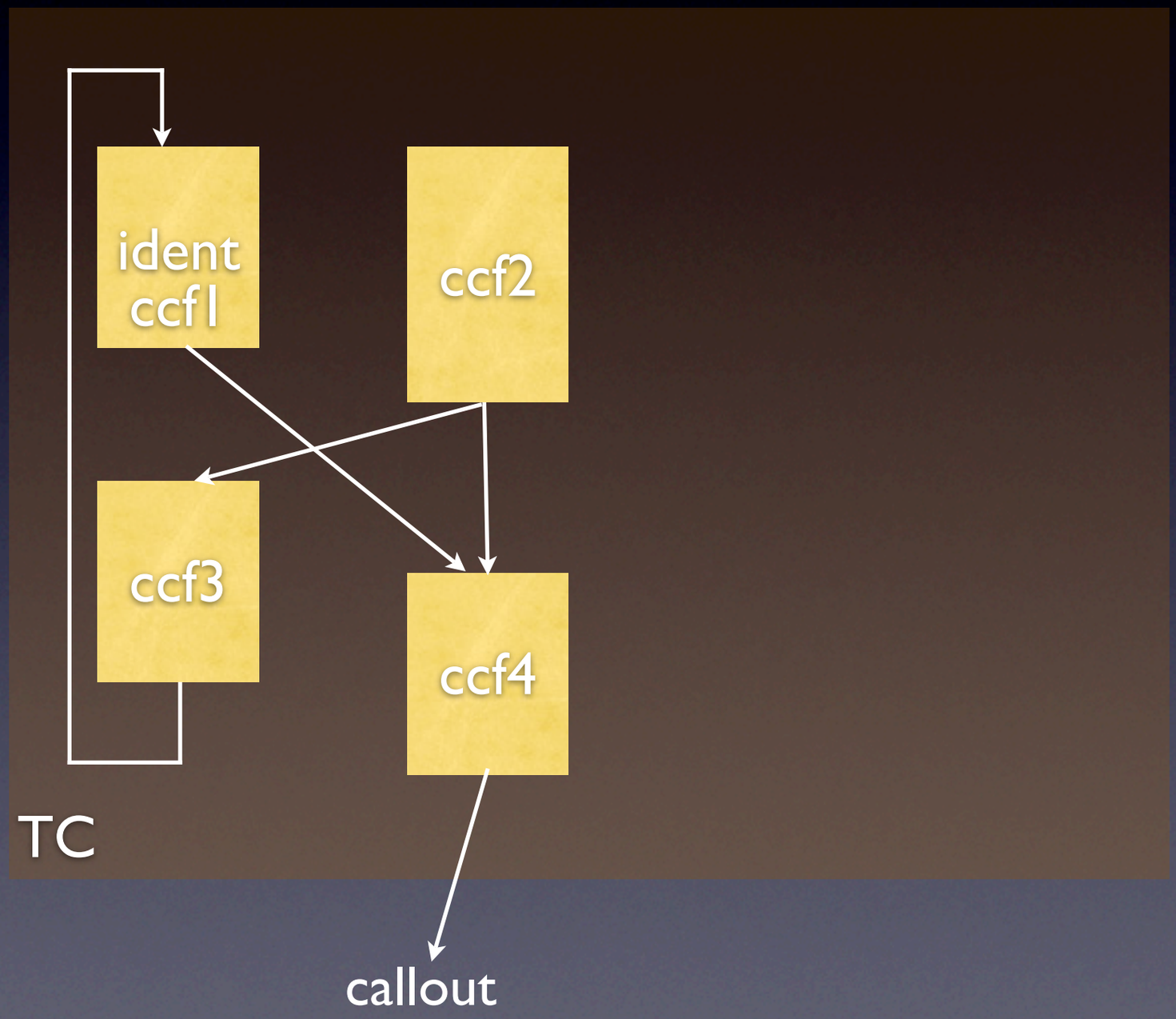
Translation Types

- Specialized trace types (a further 2-10x speedup): general traces, page table, APIC, DT
- Device accesses: MMIO, I/O space (port prediction), patterns (VGA)

Translation Types

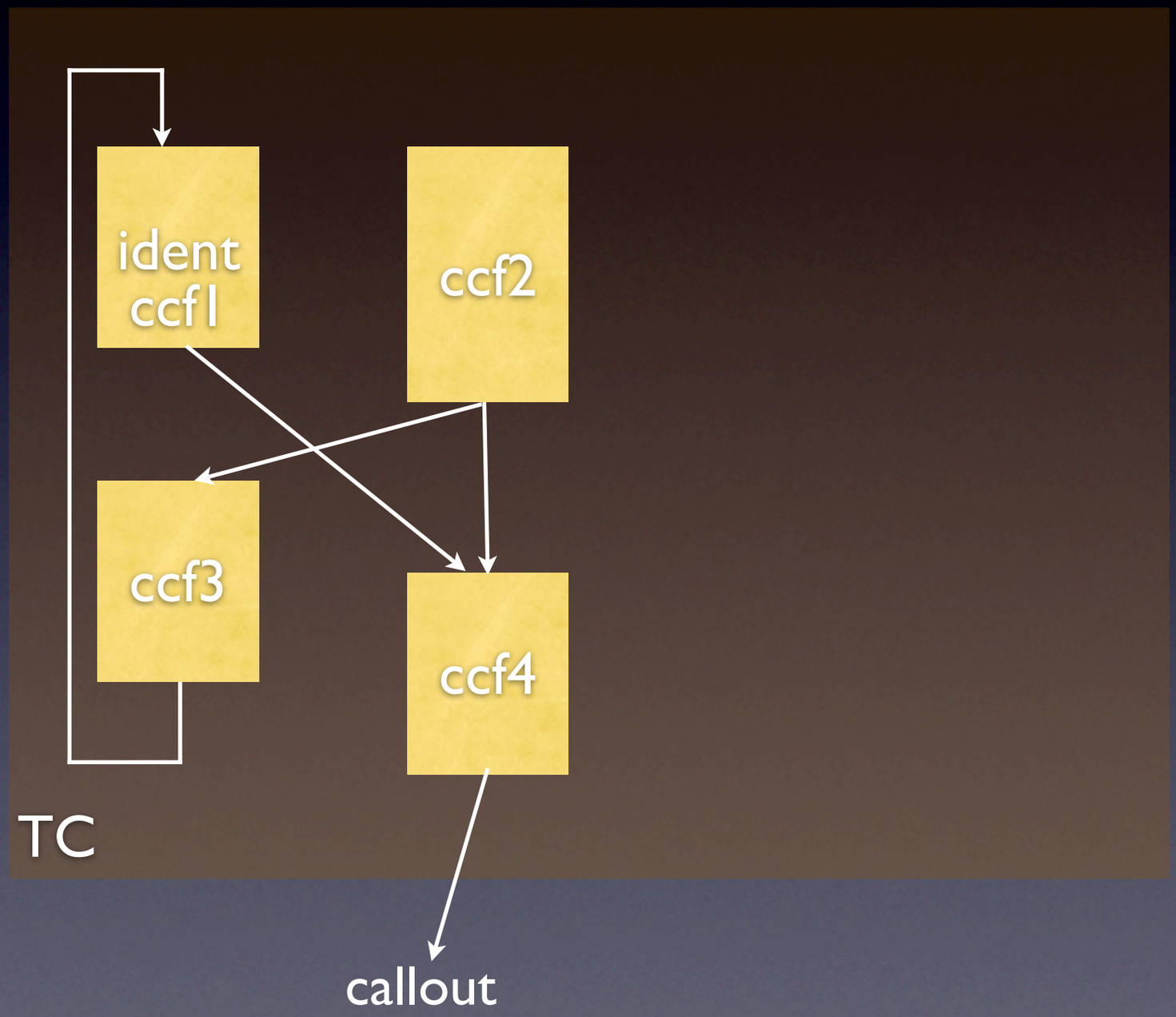
- Identifying which specialized translation to use can be difficult
- Often useful to be able to switch between different translations dynamically
- Don’t want to pay cost (time or space) of retranslating

Translation Types

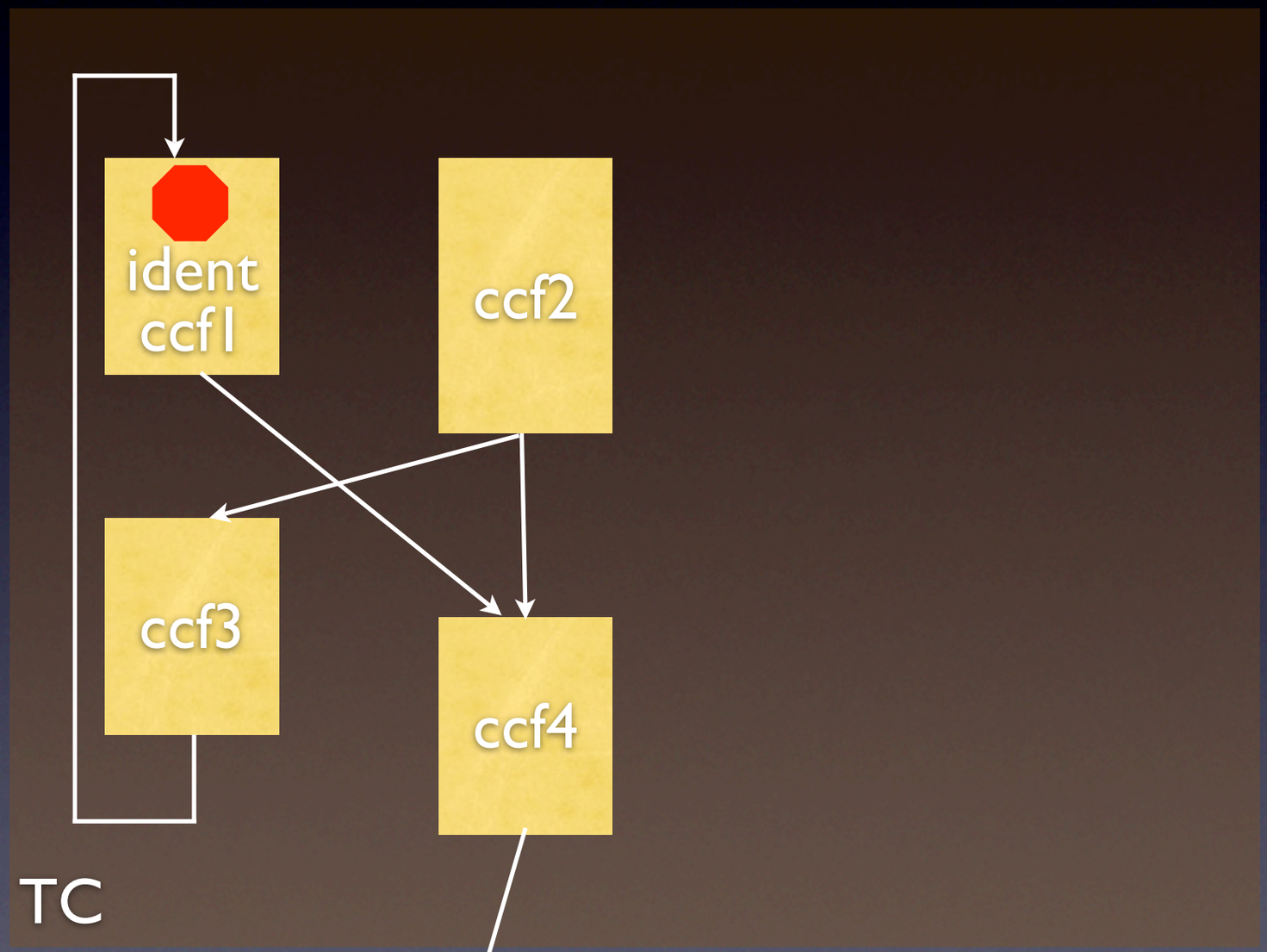


Translation Types

Initially IDENT



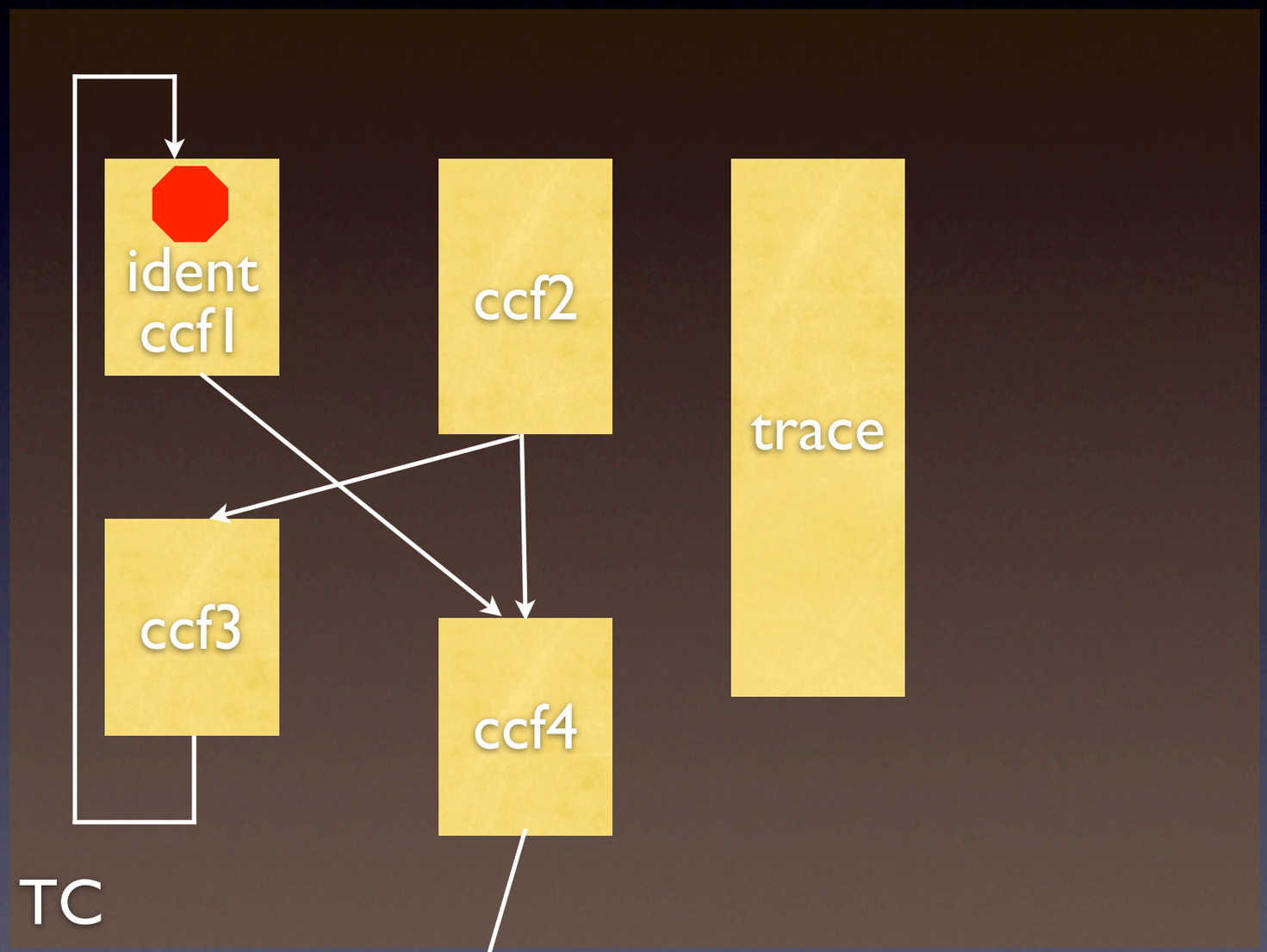
Translation Types



Initially IDENT

Trace fault

Translation Types

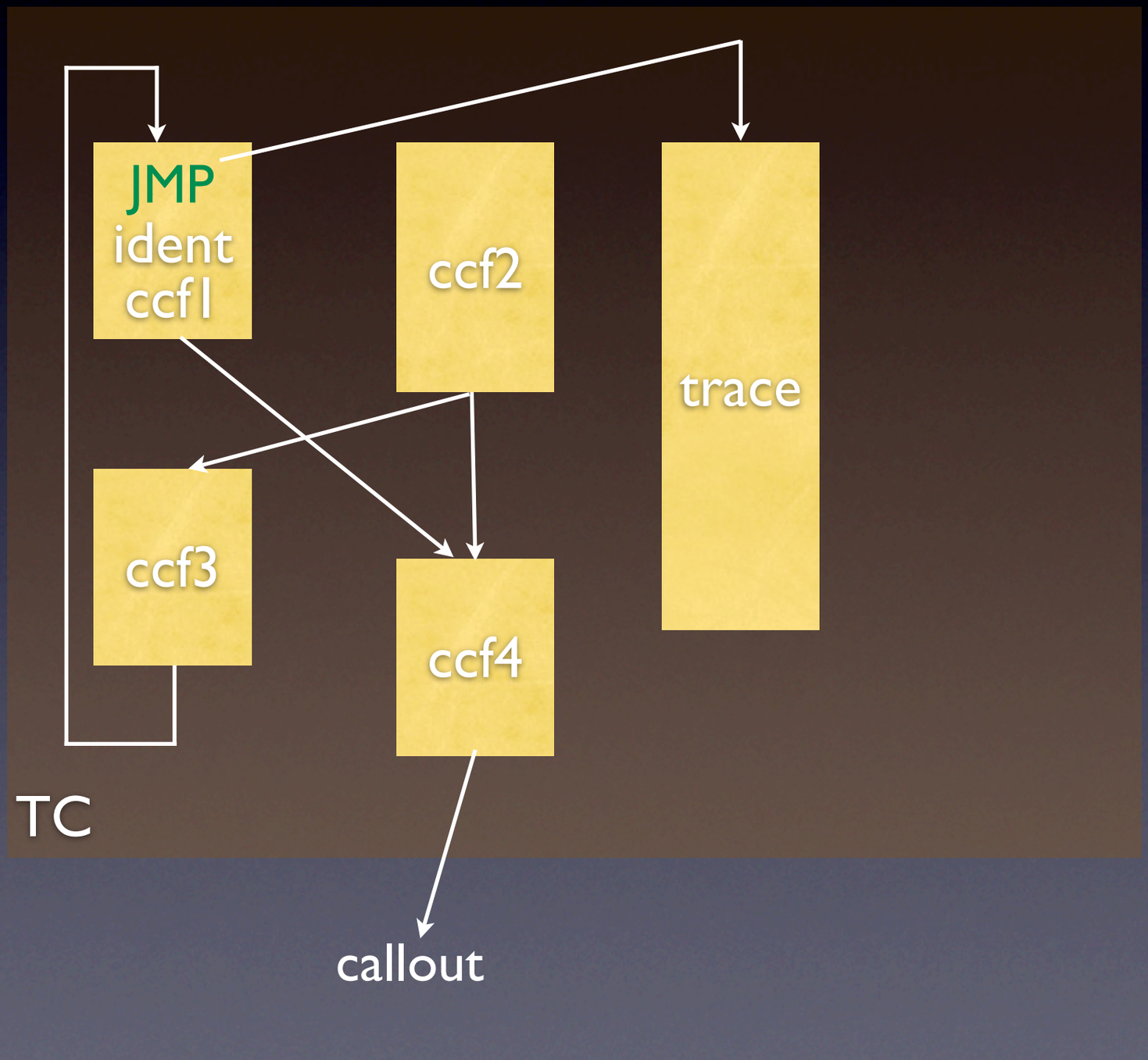


Initially IDENT

Trace fault

New translation for handling trace simulate)

Translation Types



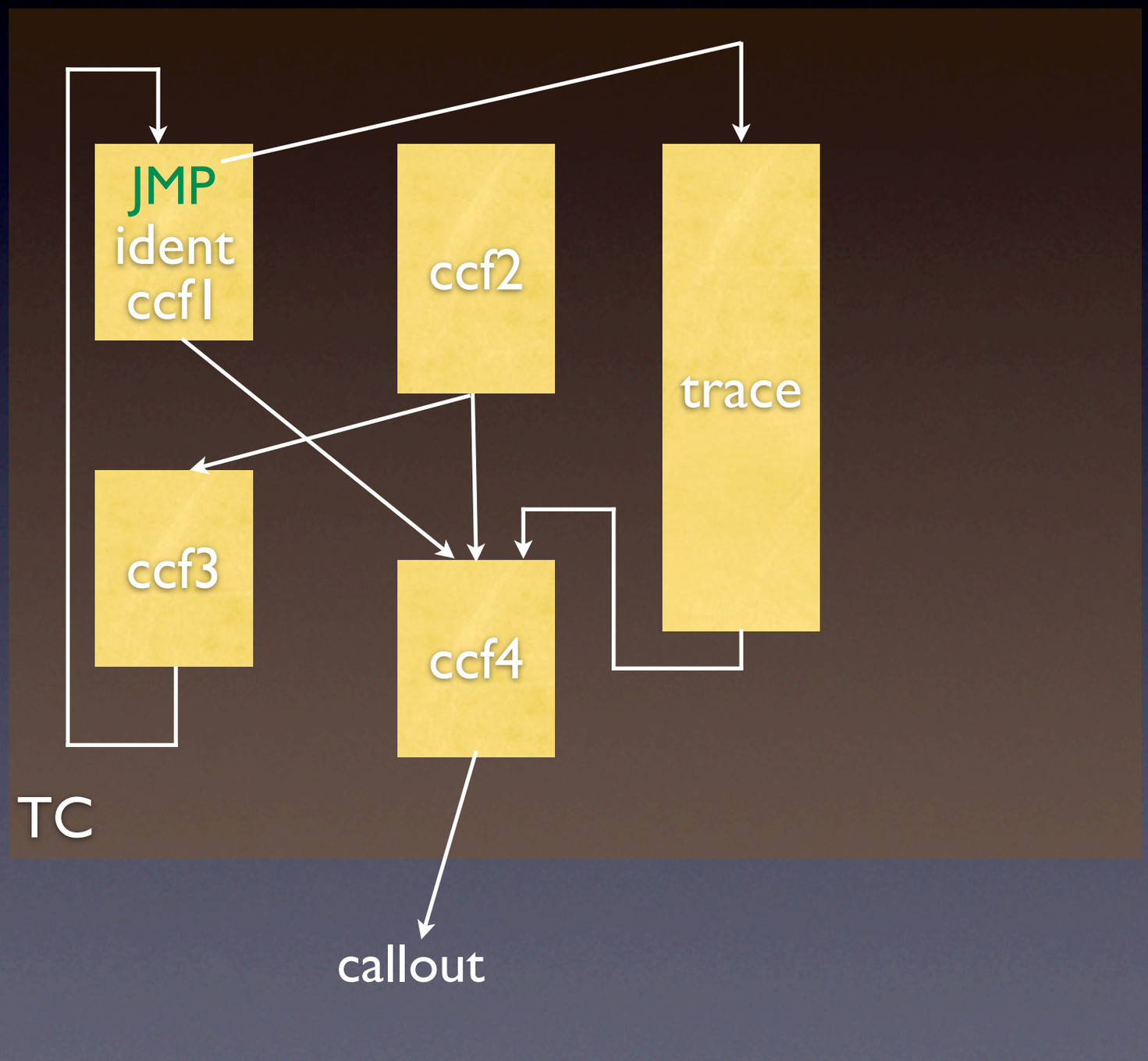
Initially IDENT

Trace fault

New translation for handling trace simulate)

Forward to active translation

Translation Types



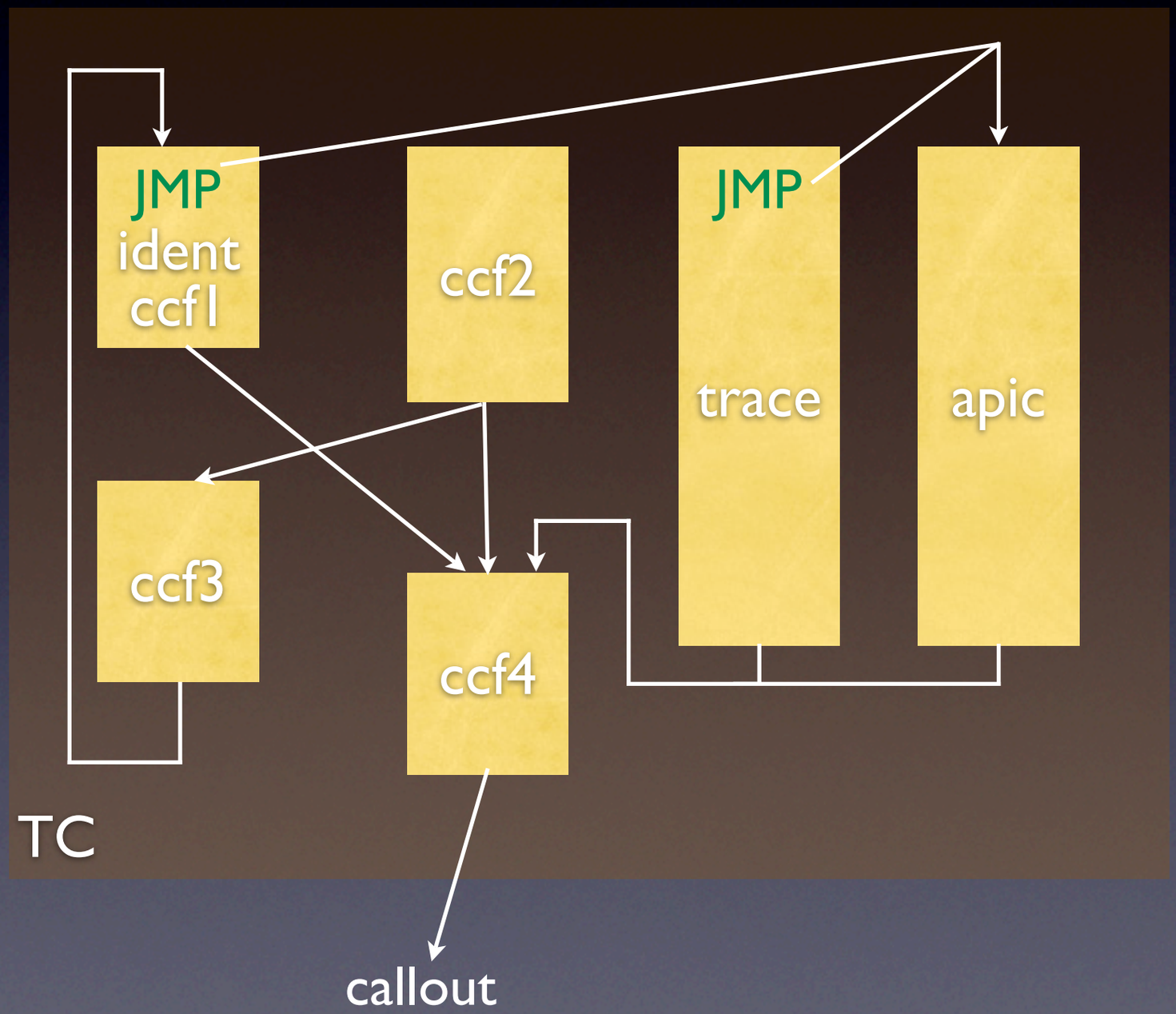
Initially IDENT

Trace fault

New translation for handling trace simulate)

Forward to active translation

Translation Types



Initially IDENT

Trace fault

New translation for handling trace simulate)

Forward to active translation

Adapt to guest behavior by moving JMPs around to select active translation

Widening BT

- Translator supports real, v8086, 16/32/64-bit protected mode input
- Always generates 64-bit protected mode output
- When converting from non-64-bit source, we call it “widening”

Widening BT

- Operand sizes can be converted by appropriately adding or removing prefixes
- Some instructions don’t exist in 64-bit mode
- 64-bit mode doesn’t provide segmentation support, requiring all segmentation to be done in software

Widening BT example

```
MOVZX %edx,-0x7fb806a8
DEC   %edx
MOVZX %ecx,%cx
DEC   %edx
CMP   %ecx,%edx
JNZ   -0x385 ;0x8044dfa6
```

Widening BT example

```
MOVZX %edx,-0x7fb806a8
DEC   %edx
MOVZX %ecx,%cx
DEC   %edx
CMP   %ecx,%edx
JNZ   -0x385 ;0x8044dfa6
```

Memory accessing instruction

Single byte dec not available in 64 bit mode

Widening BT example

```
MOVZX %edx,-0x7fb806a8
DEC   %edx
MOVZX %ecx,%cx
DEC   %edx
CMP   %ecx,%edx
JNZ   -0x385 ;0x8044dfa6
```

```
WIDEN_MEM       MOV    %r11,%rax
                LAHF   %ah
                SETO   %al
                MOV    %r10d,$-0x7fb806a8
                MOV    %r15d,%r10d
                SUB    %r15,-0x209d28c(%rip)
                           ; BT->widenSeg[SEG_DS].loAddr
                CMP    %r15,-0x209d28b(%rip)
                           ; BT->widenSeg[SEG_DS].span32R
                JC     0x2
                BPDISP 18 BPDISPATCH_WIDENSEGMENTCHK
                ADD    %r10d,-0x209d2ae(%rip)
                               ; BT->widenSeg[SEG_DS].base
                CMP    %al,$-0x7f
                SAHF   %ah
                MOV    %rax,%r11
                MOVZX  %edx,(r10)
NEARLY_IDENT    DEC    %edx
                MOVZX  %ecx,%cx
                DEC    %edx
                CMP    %ecx,%edx
JCC             JNZ    0x4a504c
                JMP    0x49f824
```

Widening BT example

```
MOVZX %edx,-0x7fb806a8
DEC   %edx
MOVZX %ecx,%cx
DEC   %edx
CMP   %ecx,%edx
JNZ   -0x385 ;0x8044dfa6
```

```
NEARLY_IDENT <A> MOVZX %edx,-0x7fb806a8
               DEC   %edx
               MOVZX %ecx,%cx
               DEC   %edx
               CMP   %ecx,%edx
JCC            JNZ   0x505dc4
               JMP   0x500909
```

Code Modification

- If guest writes to source code, translated code can no longer be used
- First line of defense:
  - Apply trace to source page (write protect)
  - Notified when write occurs
  - Invalidate the translated code

Code Modification

- Invalidation is costly — reduce need for it
  - Track individual source bytes
  - No invalidation when code & data share page
- Traps are costly
  - Use self-guarding (“checkcode”) translations
  - Works well for false sharing
  - Executing guard code is costly itself
- Adaptive algorithm — approach optimal mix