

z/OS Basics: ABEND and Recovery

(All You Need to Know to Write Your First ESTAE)

Vit Gottwald
CA Technologies

August 3, 2010
Session Number 8017



SHARE in Boston

Agenda

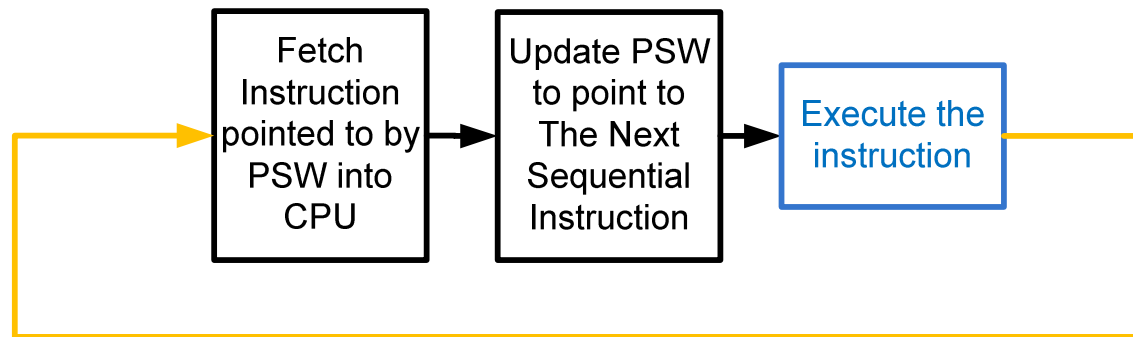
- Introduction
 - Basic Hardware Terms
 - Instruction Execution Loop
 - Interrupts
- Recovery
 - Program Error
 - Recovery/Termination Manager
 - ESTAE
 - z/OS Control Blocks
 - Special Considerations
- References

Basic terms

- Storage
 - Programs
 - Data
 - *Low Core* (first 8K of storage)
- CPU
 - 16 General Purpose Registers
 - Program Status Word (instruction pointer)
- Instruction
 - Operation code
 - Operands
 - Length

Instruction Execution Loop

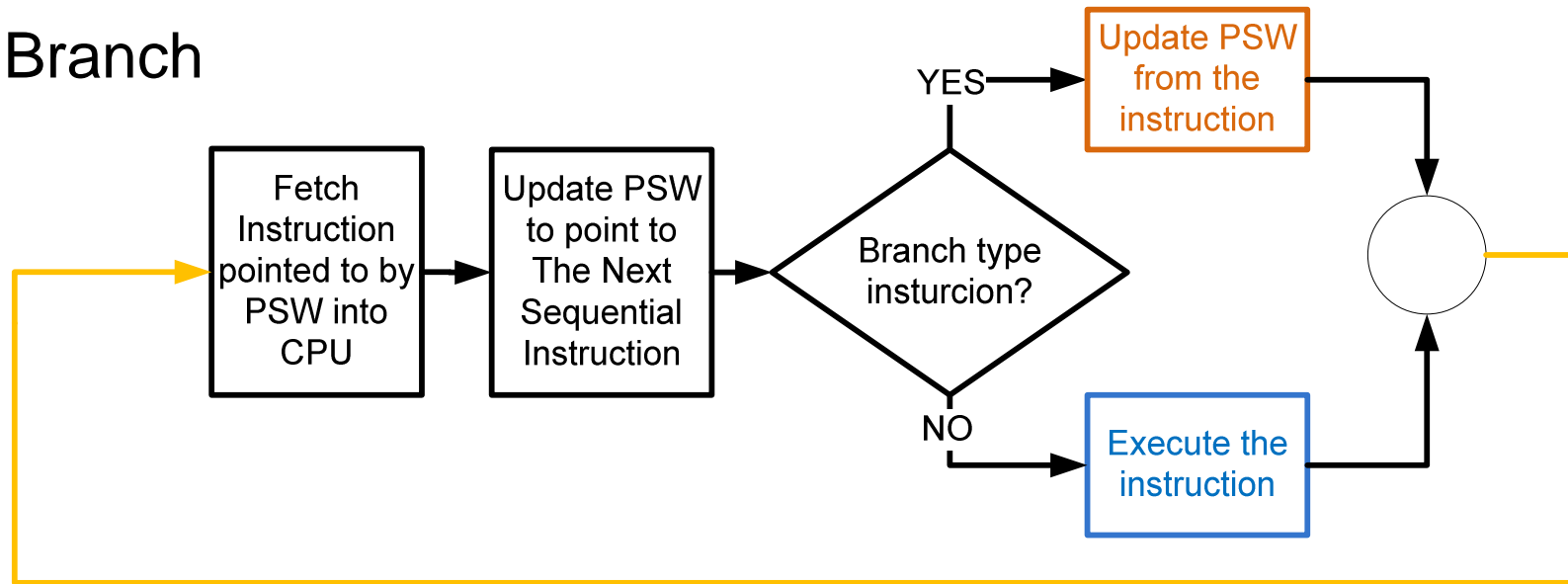
- Sequential



- How does the CPU know the instruction length?
 - First two bits of operation code
 - 00 – instruction is 2 bytes long
 - 01 or 10 – instruction is 4 bytes long
 - 11 – instruction is 6 bytes long

Instruction Execution Loop

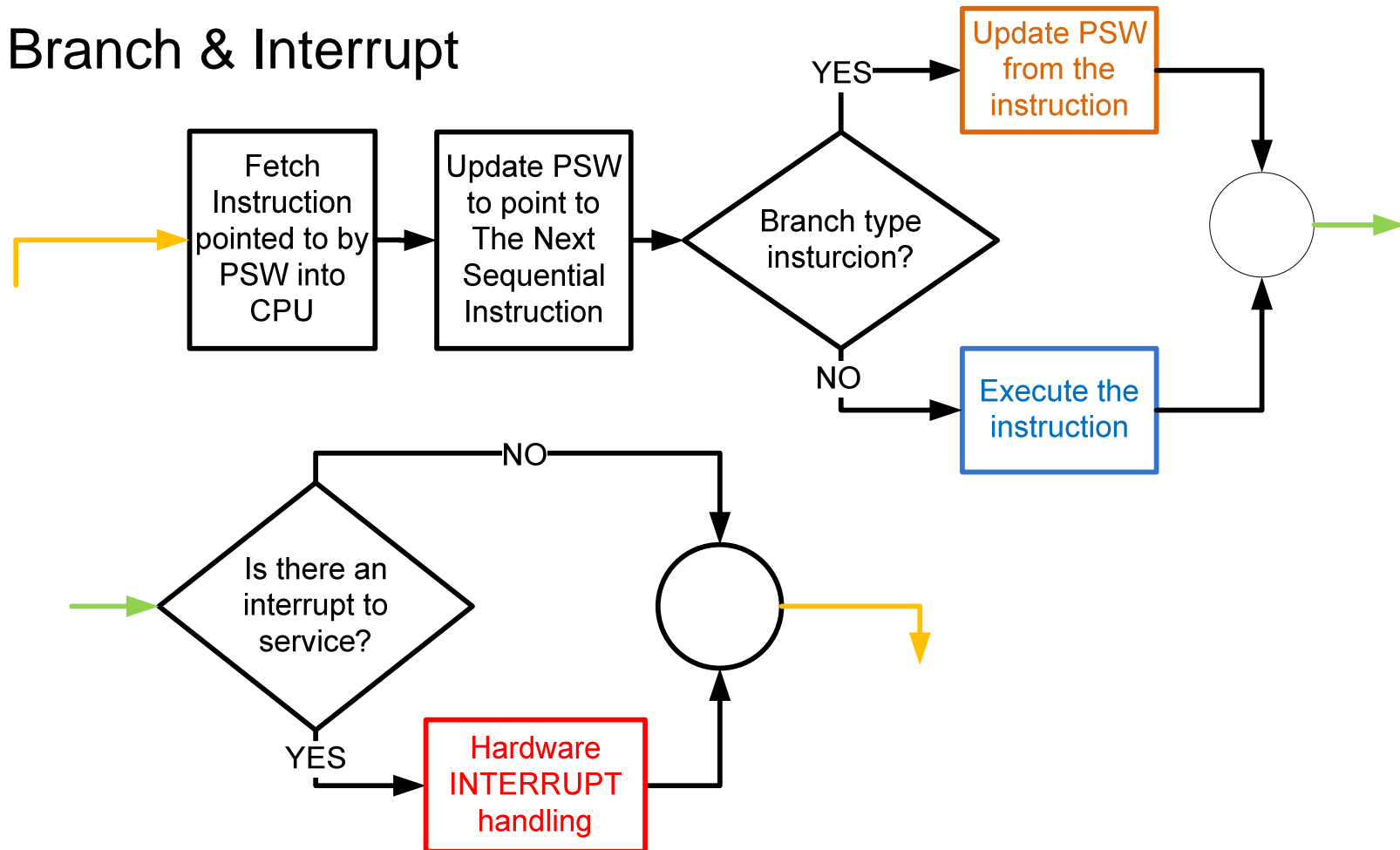
- Branch



- Branch type instructions replace the instruction address in PSW

Instruction Execution Loop

- Branch & Interrupt



What does the hardware do?

- Save into Low Core
 - Current PSW
 - PSW extension
 - Interrupt code
 - Instruction Length Code (ILC)
 - TEA
 - BEAR } discussed later
- Load from Low Core
 - New PSW assigned to the type of interrupt that occurred

Hardware
INTERRUPT
handling

Interrupts

- Each interrupt type has its own fields in Low Core
 - old-PSW
 - new-PSW
- First Level Interrupt Handler (FLIH)
 - Routine pointed to by instruction address in new-PSW
- Interrupt types
 - Restart, External, Machine Check, I/O
 - SVC
 - Program Check
 - CPU recognized problem in execution of an instruction
 - Categorized by Program Interruption Code (PIC)

Program Interruption Code (PIC)

	PIC	Reason	Type of instruction ending
S0C1	0001	Operation	suppressed
	0002	Privileged operation	suppressed
	0003	Execute	suppressed
S0C4	0004	Protection	suppressed or terminated
	0005	Addressing	suppressed or terminated
	0006	Specification	suppressed or completed
	0007	Data	suppressed, terminated or completed
	0008	Fixed-point overflow	completed
	0009	Fixed-point divide	suppressed or completed
	000A	Decimal overflow	completed
	000B	Decimal divide	suppressed
	000C	HFP exp. overflow	completed
	000D	HFP exp. underflow	completed
000E	HFP significance	completed	
	000F	HFP divide	suppressed
	0010	Segment translation	nullified
	0011	Page translation	nullified

For more PICs see, SA22-7832-07 ,
Chapter 6, Figure 6-1 Interruption Action

Program error

- Hardware detected (Program Check)
 - FLIH receives control and decides whether the program check is an error (e.g. PIC 11 - page fault - is not always a program error)
 - If the P.C. is considered an error, FLIH passes control to *RTM*
 - Results in 0Cx ABENDs
- Software detected
 - Either a z/OS component or a user program discover a problem and decide to terminate abnormally (call ABEND macro)
 - The call of ABEND macro is an entry to *RTM*
 - Typically the ABEND code is in the form xNN
 - NN - SVC hex number of the z/OS service detecting the problem
 - e.g. x13 is a group of ABENDs related to open processing

Recovery/Termination manager (RTM)

- Receives control early after the discovery of a *program error* (or when a program ends normally)
- Passes control to appropriate *recovery routine* (if present)
- If recovery not successful and either of
 - **//SYSUDUMP**, **//SYSABEND**, or **//SYSMDUMP** DD present, requests documentation of the error by calling z/OS dump services (SNAP macro)
- Handles the final *termination* of the program
 - Closing any open datasets
 - Freeing memory
 - Releasing ENQs

Recovery routine

- Responsible for
 - Fixing the error and giving the failing program another chance (retry)
 - Documenting the error, cleaning up resources, and continuing with *termination* process (percolate)
- Two basic types
 - ESPIE – to handle Program Checks with PIC 1-F hex
 - ESTAE-like – to handle ABENDs (Program Checks are special case)

Extended Specify Task Abnormal Exit (ESTAE)

- Established through ESTAE macro
- At entry receives pointers to
 - Parameter specified by the user at ESTAE macro call
 - System Diagnostic Work Area (SDWA)
 - Contains the ABEND information
 - May not be available, check if R0 equals 0C hex
- Communicates with RTM via SDWA
 - Read information directly from SDWA
 - SDWAABCC, SDWACRC, SDWAEC1, SDWAILC1, SDWAINC1, SDWAGRSV, SDWAFLGS, SDWATRAN, SDWABEA, ...
 - Write information directly to SDWA
 - SDWASR00 – SDWASR15, ...

Extended Specify Task Abnormal Exit (ESTAE), cont'd

- Communication with RTM via RTM, cont'd
 - SETRP macro
 - Whether to *retry* (RC=4) or *percolate* (RC=0)
 - Specify the *retry address* (RETADDR=)
 - Restore retry registers from SDWA (RETREGS=YES)
 - ... all options described in [3]
 - SDWA 64 bit extension
 - provided only when SDWALOC31=YES in ESTAE macro call
- Detailed usage in [1], Chapter named “Providing recovery”
 - Not easy to digest at first time (following sample should help)

Translation Exception Address (TEA)

- Location 168-175 in Low Core
- Filled in when page or segment translation occurs (PIC 10 and 11)
- Bits 0-51 contain address of the page we tried to access
- Bits 52-63 are unpredictable
- Provided in SDWA
 - 32 bit portion in **SDWATRAN**
 - Full 64 bit in SDWA 64 bit extension (**SDWATRNE**)

Breaking Event Address Register (BEAR)

- 8 bytes long CPU register
- When a branch type instruction is executed, it's address is placed in the breaking-event-address register
- When a program interruption occurs, the current contents of the BEAR is placed into Low Core location 110-118
- Provided in 64 bit SDWA extension (**SDWABEA**)
- Priceless for debugging
“wild branches”



Very Simple Example

1. Establish an ESTAE
2. Cause a Program Check by branching to `FFFFFFFFFE` hex
3. Recovery routine gets control and sets retry registers:
 - Clear R0 Translation Exception Address to R3
 - ABEND code into R1 Breaking Event Address to R4
 - Reason code into R2
4. Retry
5. Disable the ESTAE
6. Cause an S0C1 ABEND by `DC H'0'`
 - ESTAE no longer defined -> proceed with *termination*
 - Register content displayed in the 'diagnostic dump' in file 1

Very Simple Sample, cont'd

	COPY ASMMSP	ENABLE STRUCTURED PROGRAMMING MACROS
	SYSSTATE ARCHLVL=2	USE Z/ARCHITECTURE INSTRUCTIONS
	ASMMREL ON	USE RELATIVE BRANCHING
SAUTH	CSECT	
SAUTH	AMODE 31	ABOVE THE LINE TO GET BEAR
SAUTH	RMODE ANY	
	STM 14,12,12(13)	
	LARL 8,RECOVERY	RECOVERY ROUTINE ADDRESS
	LARL 9,RETRY	RECOVERY ROUTINE PARAMETER ADDRESS
	ESTAE (8),CT,PARAM=(9),SDWALOC31=YES	ESTABLISH ESTAE
	LHI 15,-2	MAX EVEN 31 BIT ADDRESS -> S0C4-11 X SEE BOTH TEA AND BEAR
	BR 15	BRANCH TO HELL (PSW USELESS)
RETRY	DS 0H	
	ESTAE 0	REMOVE THE ESTAE
	DC H'0'	INVALID OPERATION CODE -> S0C1-1

Very Simple Sample, cont'd

```

RECOVERY DS      0H
                IF      CHI,0,EQ,X'0C'      Q.SDWA MISSING
*                WTO     'SDWA MISSING'     may change registers 0,1,14,15
                SR      15,15              PERCOLATE
                BR      14                  RETURN TO RTM=PERCOLATE BY DEFAULT
                ENDIF
                STM     14,12,12(13)        SAVE REGISTERS
                LR      3,1                  SAVE POINTER TO SDWA
                USING   SDWA,3              MAP SYSTEM DIAGNOSTIC SAVE AREA
                ...                          see next slide and include here
                SETRP   RC=4,RETADDR=(2),WKAREA=(3),RETREGS=YES
                DROP    3
                LM      14,12,12(13)        LOAD REGISTERS
                BR      14                  RETURN TO RTM
*-----*
                IHASDWA                      GENERATE SDWA DSECT
                END     SAUTH                  END ASSEMBLY

```

Very Simple Sample, cont'd

```
*-----*
SR      0,0
ST      0,SDWASR00
MVC     SDWASR01,SDWAABCC   SAVE ABEND CODE IN R1
MVC     SDWASR03,SDWATRAN   SAVE TRANSLATION EXCEPTION ADDRESS
L       4,SDWAXPAD         ADDRESS OF SDWA EXTENSION POINTERS
USING   SDWAPTRS,4
      L      5,SDWASRVP         RECORDABLE EXTENSION
USING   SDWARC1,5
      MVC    SDWASR02,SDWACRC   SAVE REASON CODE
DROP   5
      L      6,SDWAXEME         64-BIT EXTENSION
USING   SDWARC4,6
      MVC    SDWASR04,SDWABEA+4  SAVE BREAKING EVENT ADDRESS-31
DROP   6
DROP   4
```

```

> APPLID(A11IRO53)    USER(Y8V,GOTVI01)
> JOB(Y8VSMPL,1893)  SCRL CSR  COLS 00009 00088                F 01 P 0001
.1.....+.....2.....+.....3.....+.....4.....+.....5.....+.....6.....+.....7.....+.....8.....+...
JOB01893  TSS7001I Count=07696 Mode=Fail Locktime=None Name=GOTTWALD, VIT
JOB01893  $HASP373 Y8VSMPL  STARTED - WLM INIT - SRVCLASS BATSTWLM - SYS CA31
JOB01893  IEF403I Y8VSMPL - STARTED - TIME=04.24.30
JOB01893  IEA995I SYMPTOM DUMP OUTPUT 745
          SYSTEM COMPLETION CODE=0C4 REASON CODE=00000011
          TIME=04.24.31 SEQ=27556 CPU=0000 ASID=023E
          PSW AT TIME OF ERROR 078D0000 FFFFFFFE ILC 2 INTC 11
          NO ACTIVE MODULE FOUND
          NAME=UNKNOWN
          DATA AT PSW 7FFFFFF8 - ***** ***** 000A0000
          GR 0: 00000001_00000000 1: 00000000_3AD00F5C
             2: 00000000_00000040 3: 00000000_007DAD84
             4: 00000000_007DAD60 5: 00000000_007FF370
             6: 00000000_007BBFE0 7: 00000000_FD000000
             8: 00000000_3AD00F94 9: 00000000_3AD00F8C
             A: 00000000_00000000 B: 00000000_007FF370
             C: 00000000_80C979D2 D: 00000000_00006F60
             E: 00000000_80FDD4B8 F: 00000000_FFFFFFFE
          END OF SYMPTOM DUMP
JOB01893  IEA995I SYMPTOM DUMP OUTPUT 746
          SYSTEM COMPLETION CODE=0C1 REASON CODE=00000001
          TIME=04.24.31 SEQ=27557 CPU=0000 ASID=023E
          PSW AT TIME OF ERROR 078D0000 BAD00F94 ILC 2 INTC 01
          ACTIVE LOAD MODULE ADDRESS=3AD00F48 OFFSET=0000004C
          NAME=GO
          DATA AT PSW 3AD00F8E - 00840A3C 0000A70E 000CA774
          GR 0: 00000001_00000000 1: 00000000_840C4000
             2: 00000000_00000011 3: 00000000_7FFFF000
             4: 00000000_3AD00F8A 5: 00000000_007FF370
             6: 00000000_007BBFE0 7: 00000000_FD000000
             8: 00000000_3AD00F94 9: 00000000_3AD00F8C
             A: 00000000_00000000 B: 00000000_007FF370
             C: 00000000_80C979D2 D: 00000000_00006F60
             E: 00000000_80FDD4B8 F: 00000000_00000000
          END OF SYMPTOM DUMP
JOB01893  IEF450I Y8VSMPL RUNPGM - ABEND=S0C1 U0000 REASON=00000001 747
          TIME=04.24.31
JOB01893  IEF404I Y8VSMPL - ENDED - TIME=04.24.31
JOB01893  $HASP395 Y8VSMPL ENDED

```

```

> APPLID(A11IRO33)      USER(Y8V,GOTVI01)
> JOB(Y8VSMPL,1893)    SCRL CSR  COLS 00001 00080          F 04 P 0003
<.....1.....2.....3.....4.....5.....6.....7.....8
000000          00000 000B4 1611 SAUTH CSECT
                                1612 SAUTH AMODE 31          ABOVE
                                1613 SAUTH RMODE ANY
000000 90EC D00C          0000C 1614 STM 14,12,12(13)
000004 C080 0000 0024    0004C 1615 LARL 8,RECOVERY          RECOV
00000A C090 0000 001D    00044 1616 LARL 9,RETRY          RECOV
                                1617 ESTAE (8),CT,PARAM=(9),SDWALOC3
                                1619+* MACDATE 10/01/94
000010          00000 1621+ CNOP 0,4          E
000010 A715 000E          0002C 1622+ BRAS 1,*+28          L
000014 16          1623+ DC AL1(22)          F
                                +          A
000015 000000          1624+ DC AL3(0)          F
000018 000000000          1625+ DC A(0)          S
00001C 000000000          1626+ DC A(0)          S
000020 80          1627+ DC AL1(128)          F
000021 01          1628+ DC AL1(1)          T
000022 0000          1629+ DC AL2(0)          R
000024 000000000          1630+ DC A(0)          S
000028 000000000          1631+ DC AL4(0)          S
00002C 5081 0014          00014 1632+ ST 8,20(1,0)          P
000030 5091 0004          00004 1633+ ST 9,4(1,0)          P
000034 4100 0100          00100 1634+ LA 0,256(0,0)          C
000038 4110 1000          00000 1635+ LA 1,0(0,1)          M
00003C 0A3C          1636+ SVC 60          I
00003E A7F8 FFFE          FFFFFE 1637 LHI 15,-2          MAX E
                                SEE B
000042 07FF          1638 BR 15          BRANC
000044          1639 RETRY DS 0H
                                1640 ESTAE 0          REMOV
                                1642+* MACDATE 10/01/94
000044          1644+ DS 0H
000044 4100 0084          00084 1645+ LA 0,132(0,0)          I
000048 0A3C          1646+ SVC 60          I
00004A 0000          1647 DC H'0'          INVAL
00004C          1648 RECOVERY DS 0H
                                1649 IF CHI,0,EQ,X'0C'          Q.SDW
                                CHI 0,X'0C'
00004C A70E 000C          0000C 1660+
000050 A774 000F          0006E 1661+ BRC 7,#@LB1

```

Some more SDWA fields of interest

- SDWAXPAD – SDWA extension pointers (SDWAPTRS dsect)
- SDWASRVP – address of recordable extension (SDWARC1 dsect)
- SDWAXEME – address of 64-bit extension (SDWARC4 dsect)

- SDWAERRB, on, SDWAPERCB – a previous ESTAE percolated
- When your ESTAE gets control make **sure whether it is the first one or whether some other ESTAE already percolated !!!**

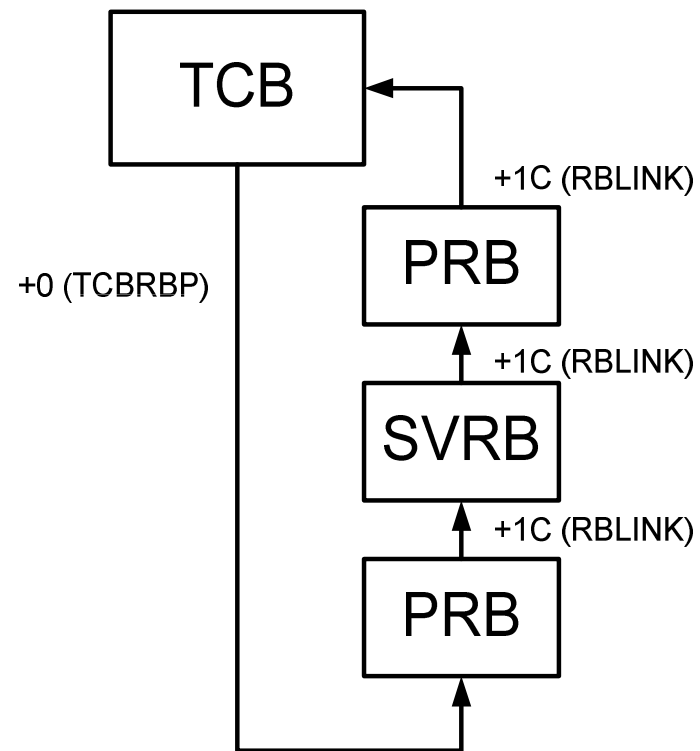
Make sure to

- Establish your recovery routine when your routine gets control from system, exit, or other app.
- Remove the recovery routine before returning to the caller
- Learn more
 - Read “Providing recovery”, especially section “Special considerations” in [1]
 - Learn about TCB and RB chains and how they relate to recovery routines
 - Be careful when dealing with Linkage Stack, see IEALSQRY macro

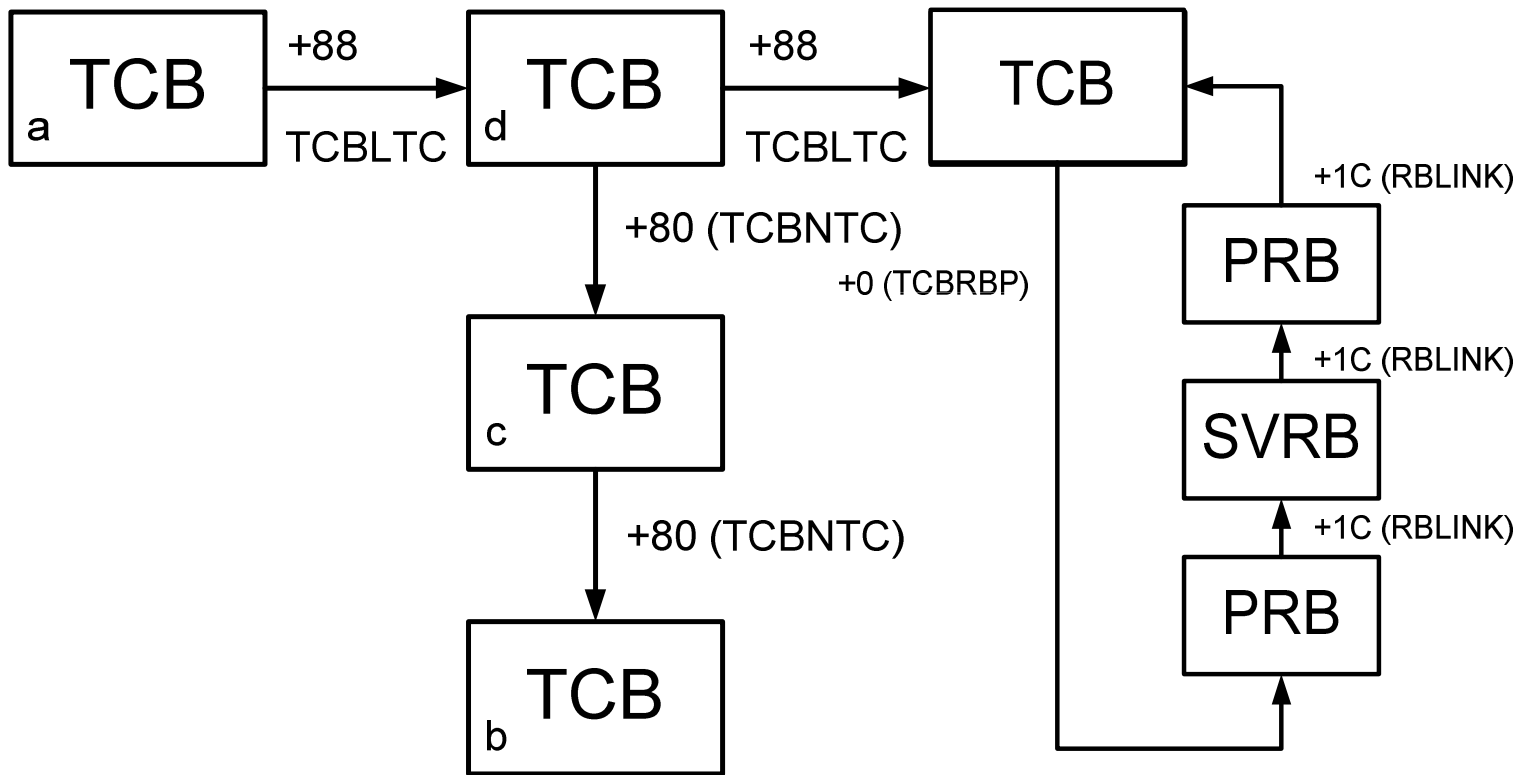
Multiple ESTAEs

- When your program establishes multiple ESTAEs
- And an ABEND occurs
 1. The most recently defined ESTAE routine gets control
 2. When it decides to percolate, previously defined ESTAE gets control
 3. Ditto
 4. ...
- ESTAE is represented by a STAE Control Block (SCB)
- SCBs form a stack (LIFO) with the newest SCB on the top
- When an ESTAE percolates its SCB is removed from the stack and control is passed to the next on the top

z/OS Dispatcher Control Blocks



z/OS Dispatcher Control Blocks



Other Recovery Routine Types

- ESTAI
 - Subtask recovery
 - Defined on ATTACH(X) macro with ESTAI= parameter
- Associated Recovery Routine (ARR)
 - Recovery for abends in PC routines
- Functional Recovery Routine (FRR)
 - Recovery in SRB routines
 - Defined through SETFRR macro

References

- [1] - *MVS Programming Assembler Services Guide (SA22-7605)*
- [2] - *MVS Programming Assembler Services Reference (SA22-7606)*
- [3] - *MVS Data Areas*
- [4] - *MVS Control Blocks, Hank Murphy, McGraw Hill 1995*
- [5] - *Principles of Operation (SA22-7832)*

References, cont'd

- [JB] - *Joachim von Buttlar, “System z Architecture”, [big, but worth reading, skip the IBM propaganda at the beginning],*
<http://public.dhe.ibm.com/software/dw/university/systemz/SystemzArchitectureCourse.pdf>
- [EJ] - *Ed Jaffe, Structured Assembler Language Programming Using HLASM,*
ftp://ftp.phoenixsoftware.com/pub/demo/Structured_Assembler.pdf

Please do not forget to fill in the
evaluation forms.

Session #8017

z/OS control blocks

- Piece of storage that has a meaning to z/OS
- Described in IBM manual “MVS Data Areas, Vol1. – Vol6.”
 - Not very verbose, useful if you know what you are looking for and are familiar z/OS (MVS) terminology

z/OS control blocks – PSA, CVT

- Prefix Save Area (PSA)
 - Prefix Area contains several fields that have hard wired addresses in the CPU for interrupt handling. The rest is used by FLIH and various other components of z/OS
 - In z/OS terminology Prefix Area is called Prefixed Save Area
 - Contains pointers to other control blocks
 - Task Control Block (TCB) at offset 21C
 - Address Space Control Block (ASCB) at offset 224
 - Communication Vector Table (CVT) at offset 10
- Communication Vector Table (CVT)
 - Anchor to most if not all z/OS control blocks!

z/OS control blocks – ASCB, TCB

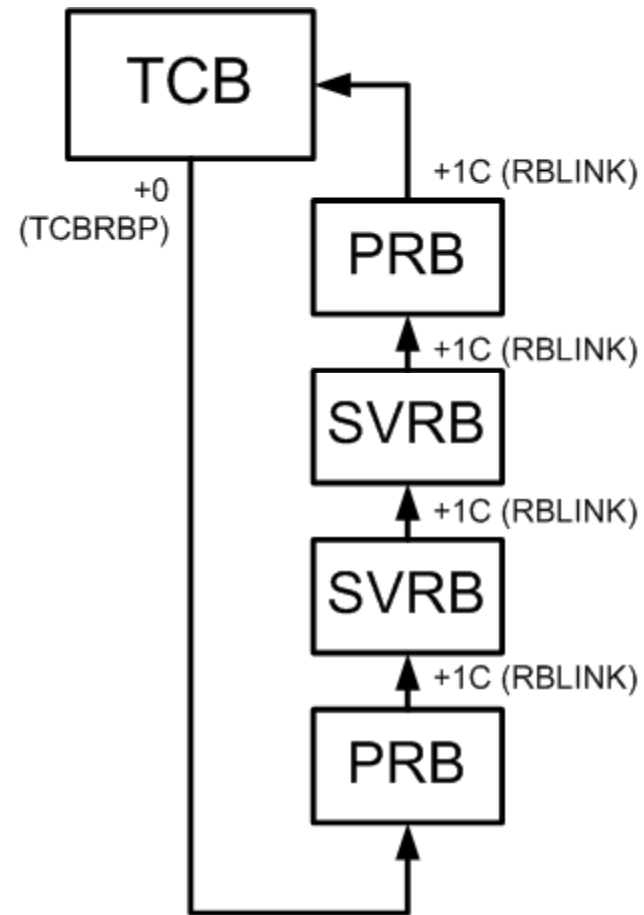
- Address Space Control Block (ASCB)
 - Represents single instance of virtual storage to z/OS (recall MVS = Multiple Virtual Storage)
 - Usually one ASCB per Job – XTCB
- Task Control Block (TCB)
 - Represents unit of work to z/OS (a *task*)
 - Think of a “task” being a “thread” in PC/UNIX terminology
 - It is an anchor to all resources z/OS allocated on behalf of the task, when TCB is removed, all resources for the task are deallocated

z/OS control blocks - PRB, SVRB

- Request Block (PRB, SVRB, IRB)
 - While TCB represents a unit of work to z/OS, RB represents a particular item we want z/OS to do on behalf of our task
 - When we request a particular program to be run, Program Request Block is created
 - When our program wants to use operating system services, it issues a suitable SVC and a Service Request Block is created
 - External interrupt may generate an asynchronous exit routine to be run (e.g. IRB created for STIMER exit routine)
 - The sequence of the Request Blocks is then called an RB chain, it is chained of a TCB in a reverse order than it was created

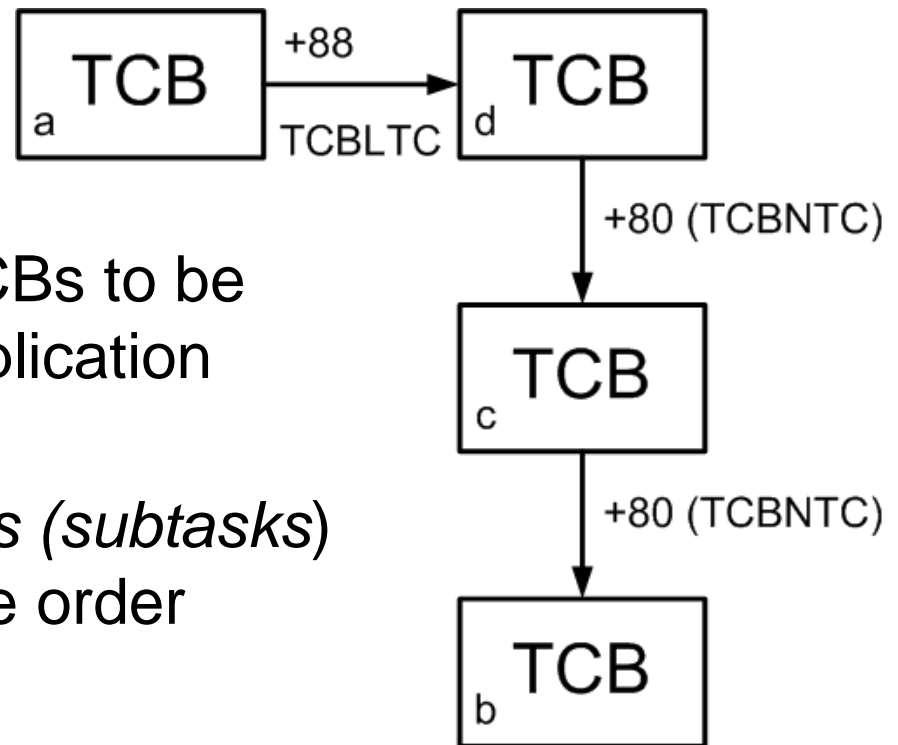
RB Chain

- TCB at offset 0 contains a fullword pointer to the most recently created RB
- Each RB points to the previously created RB
- Last RB in the chain (the first created) points back to the TCB



TCB chain

- TCB created by ATTACH macro, DETACH removes
- Program running under a TCB can request further TCBs to be created -> multi-threaded application
- Here the mother task *a*) attached three *daughter tasks (subtasks) b), c), and d)* in the respective order



TCBLTC (+88) field points to the subtask the current TCB attached last
 TCBOTC (+84) –not shown on picture- points to the parent task
 TCBLTC (+80) – points to the task attached previously by parent task

How does RTM receive control?

- Through an ABEND macro call (SVC 13 - 0A0D)
 - Terminates either current TCB or the job step TCB in the current address space
- Through a CALLRTM macro call
 - TYPE=ABTERM
 - a “super” version of ABEND
 - Allows to terminate a (TCB=) in current or other address space
 - TYPE=MEMTERM
 - Terminates an address space without giving control to task level recovery routines and resource managers

Recovery/Termination macros

- CALLRTM
 - TYPE=ABTERM is used by CANCEL operator command
 - TYPE=MEMTERM is used by the FORCE oper. command
 - You definitely want to stay away from it, supervisor state and key 0 is required to do a CALLRTM

Recovery/Termination macros

- ABEND
 - Generates an SVC 13 (0A0D)
 - Also has a branch entry
 - Allows to specify
 - ABEND code (12 bits) - separate values for System/User ABEND
 - Reason code (RETURN=, 32 bits) - passed to recovery routines
 - Dump options
 - *DUMP* – request a dump
 - *DUMPOPT* – parm. list for the SNAP macro
 - Scope of the ABEND
 - *STEP* – if specified, the job step TCB is terminated, if not specified, the default is to terminate the current TCB

RTM1 and RTM2

- RTM is composed of two parts
 - RTM1 aka “System Level RTM”
 - RTM2 aka “Task Level RTM”
- RTM1
 - Entered via CALLRTM (e.g. from FLIH for an erroneous P.C.)
 - Runs under the environment of the failing program
 - ESPIE registers with RTM1 – low overhead recovery routine
- RTM2
 - Entered via ABEND macro call either from RTM1 or directly
 - Runs as an z/OS subroutine (RB created – 0A0D)
 - ESTAE registers with RTM2 (another RB created when called)

ESTAE macro

- Assume you are writing your first ESTAE routine for your very simple program to recover from a B37 system ABEND
- You will use

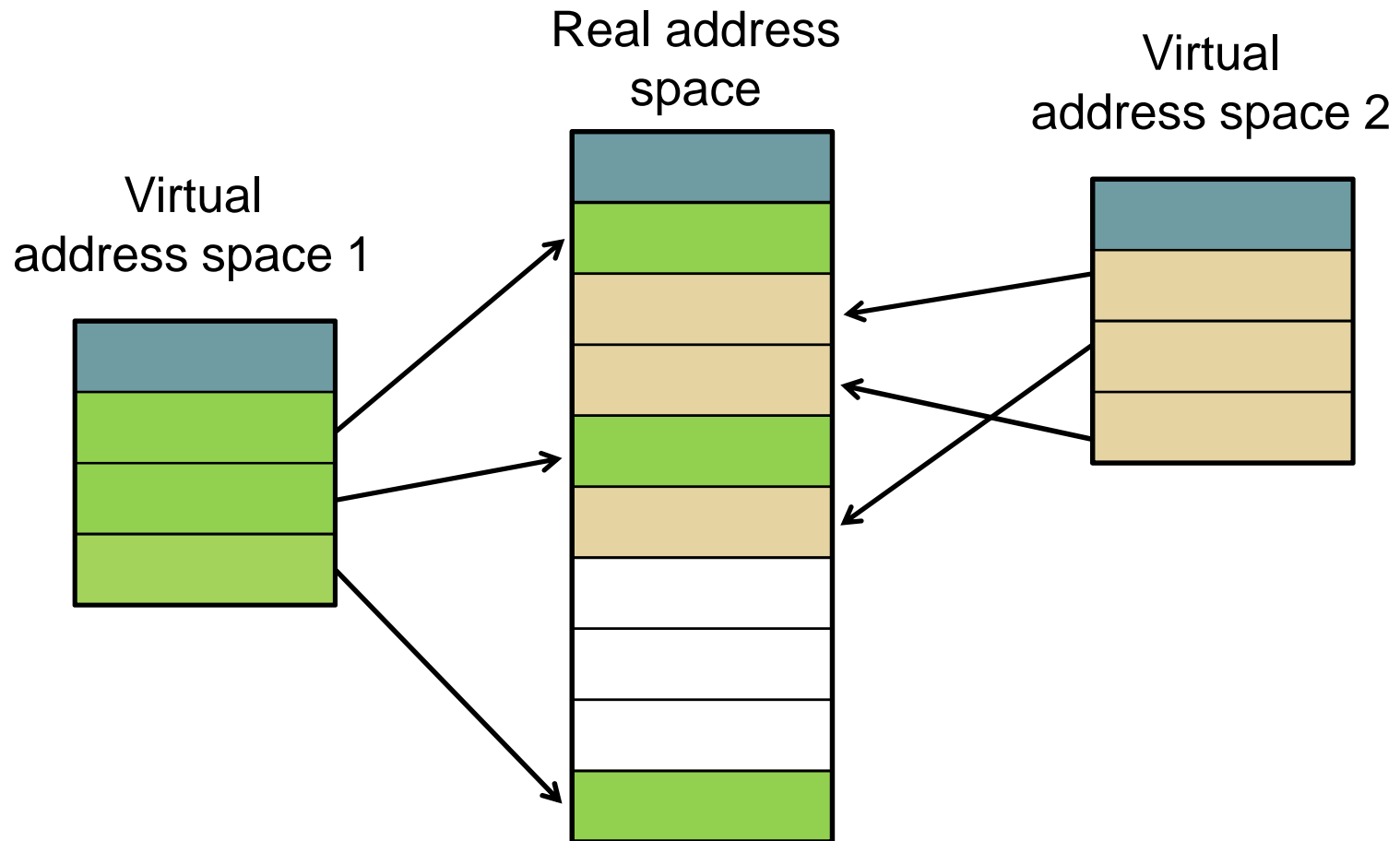
```
ESTAE EXIT_ADDR , CT , PARAM=PARAM_LIST
```

 - EXIT_ADDR – address of the recovery routine
 - PARAM_LIST - parameter list passed to the recovery routine when it is invoked by RTM
 - CT – create as opposed to OV - override an existing ESTAE

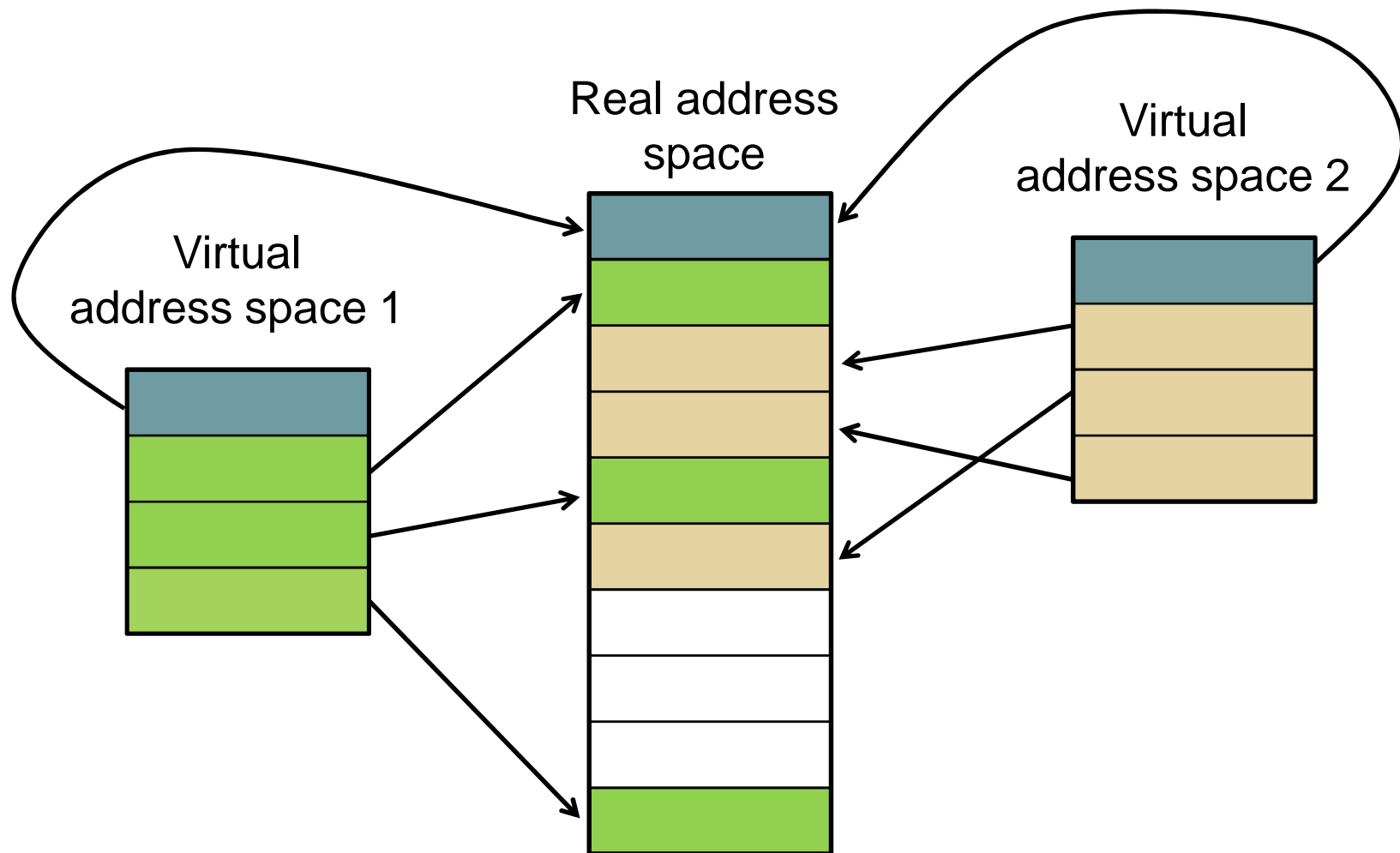
Virtual Storage

- Virtual storage
 - Introduced in S/370 in early 1970's
 - Each “application” (*address space*) can use the full range of addresses available on the architecture independently of all other applications
 - Implemented in hardware via Dynamic Address Translation
 - *VIRTUAL ADDRESSES* translated into *REAL ADDRESSES*

z/Architecture Virtual Storage



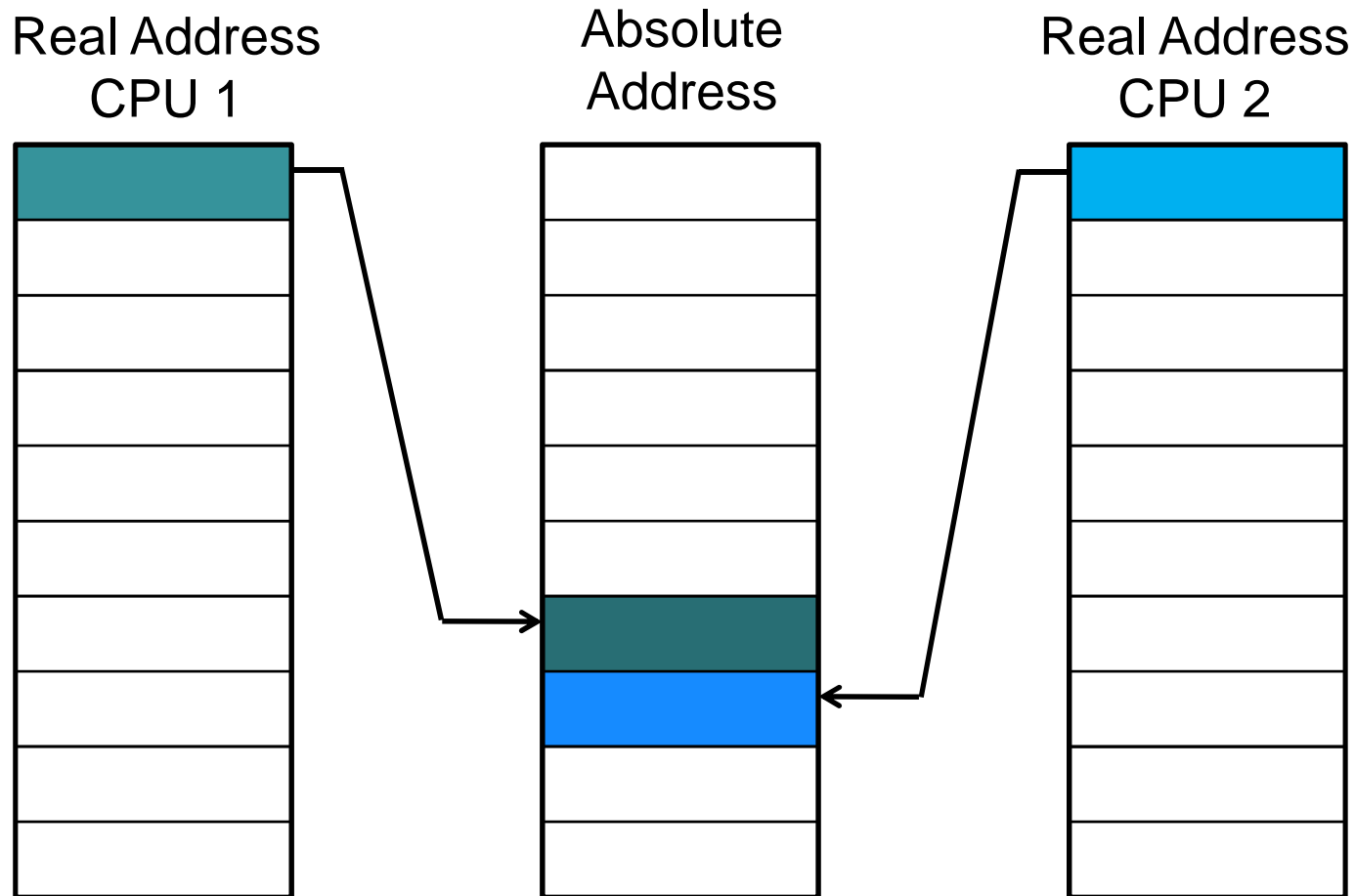
z/Architecture Virtual Storage



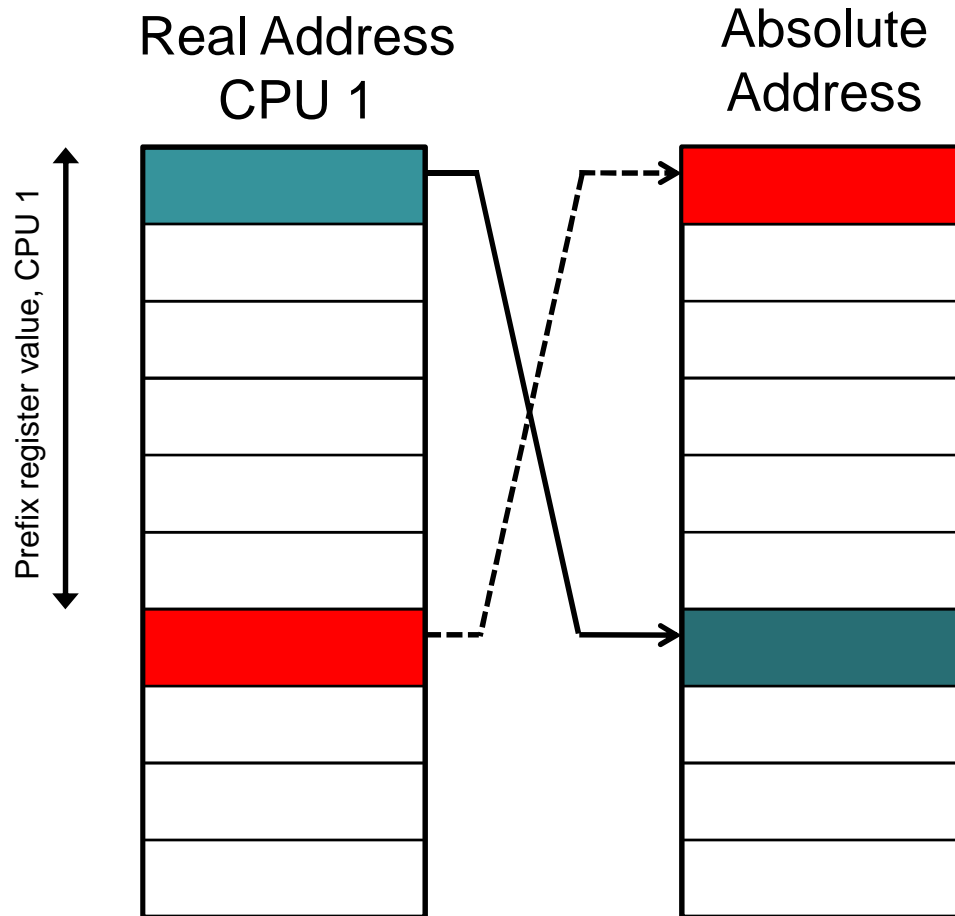
z/Architecture Virtual, Real, Absolute

- How to handle this with multiple CPUs?
- Prefix register
 - 64 bits, bits 0-32 are always 0
 - Used for assigning a range of real addresses 0-1FFF to a different block in *absolute* storage for each CPU
 - The mechanism is called *Prefixing*, the storage *Prefix Area*

z/Architecture Prefixing



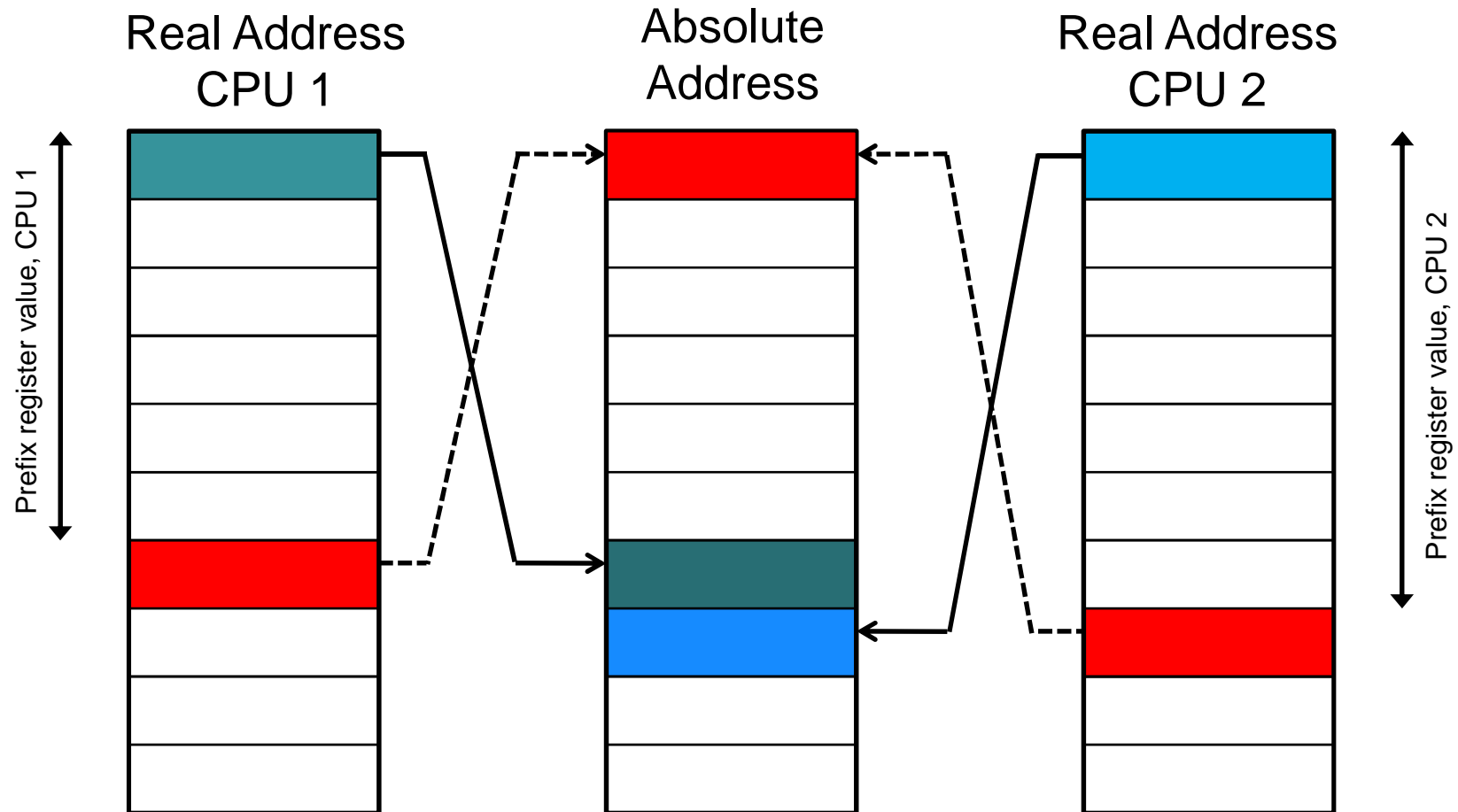
z/Architecture Prefixing



Say Prefix register value in CPU1 is 6000, then

- Real Addresses 1-1FFF are translated to Absolute Addresses 6000-7FFF
- Real Addresses 6000-7FFF are translated to Absolute Addresses 1-1FFF

z/Architecture Prefixing



Types of Instruction Ending

- Completion
 - Successful completion or partial completion (for interruptible instructions at a unit of work boundary – CC=3)
 - PSW points to the next sequential instruction
- Suppression
 - As if the instruction just executed was a no-operation (NOP)
 - contents of any result fields, including condition code are not changed
 - PSW points to next sequential instruction

Types of Instruction Ending, cont'd

- Nullification
 - Same as Suppression but
 - PSW points to the instruction just executed
- Termination¹⁾
 - causes the contents of any fields due to be changed by the instruction to be unpredictable (some may change, other not)
 - The operation may replace all, part, or none of the contents of the designated result fields and may change the condition code
 - PSW points to the next sequential instruction

1) For detailed description see SA22-7832-07, Chapter 5, Type Of Instruction Ending

Termination

- Releasing all resources acquired by the task being terminated
- RTM calls *Resource Managers* to do the actual cleanup
 - Closing any open datasets
 - Freeing memory
 - Releasing ENQs
 - ...
- Performed for both normal and abnormal program end