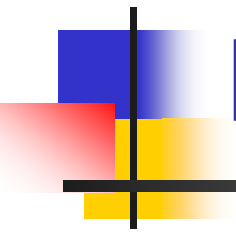
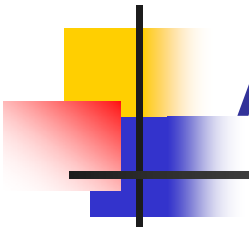


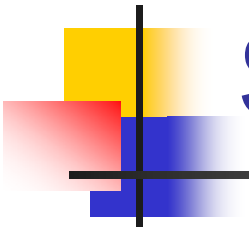
Stacks are Simple, Heaps are fun!

John Monti
IBM Poughkeepsie
jmonti@us.ibm.com



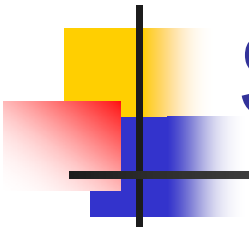
Agenda

- Stacks are Simple, an overview of LE Stacks
  - Data
  - Layout
  - Processing
- Heaps are fun, an overview of LE Heaps
  - Data
  - Layout
  - Processing
- Debugging Heap Damage (the fun part!)
- Sources of Additional Information



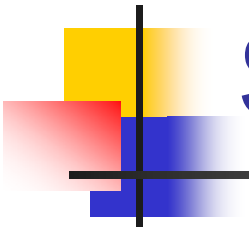
Stacks are simple

- Language Environment Storage Management
  - Stacks
    - Last In, First Out structures
    - Allow programs to be reentrant
    - Enclave level structures
      - “Main” programs have separate stacks
      - “Linked” programs have separate stacks
      - Pthreads have separate stacks



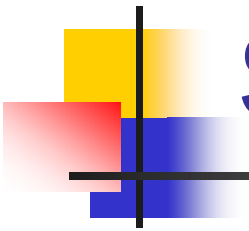
Stacks are simple...

- LE supports 2 independent stacks
  - User stack – (poorly named)
    - Used by user programs and LE
  - Library stack
    - Used “rarely” by LE
    - Always below the 16M line



Stacks are simple...

- Run-time options dealing with stacks
  - STACK(init,inc,ANY|BELOW,KEEP|FREE,dsInit,dsInc)
    - Init - Initial size of storage “chunk” allocated and managed by LE for user stack
    - Inc - When init is full, size of next storage “chunk” (increment)
    - ANY|BELOW - Location of storage
      - ANY Anywhere in 2G virtual storage
      - Below Always below 16M line
        - Required when all31(OFF)
    - KEEP|FREE - What to do when done with inc
      - KEEP Do not freemain the storage
      - FREE Freemain the storage
    - DsInit - Initial size of storage “chunk” (XPLINK)
    - DsInc - When initial full, size of next “chunk” (XPLINK)



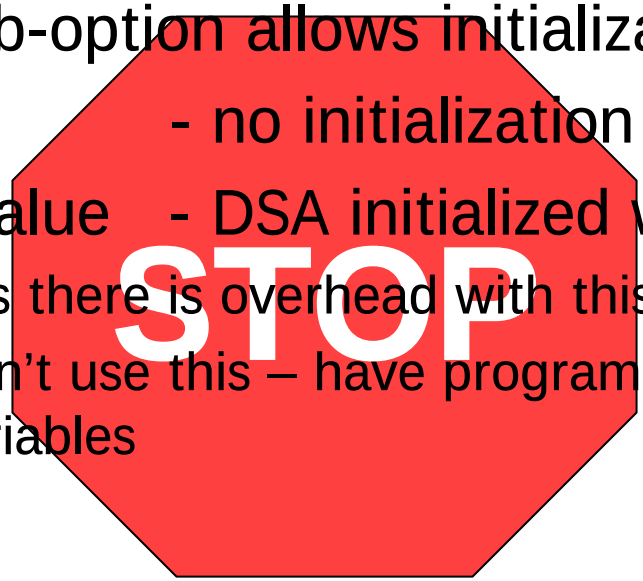
Stacks are simple...

- Run-time options dealing with stacks
  - LIBSTACK(init,inc,KEEP|FREE)
    - Init - Initial size of storage “chunk” allocated and managed by LE for library stack
    - Inc - When init is full, size of next storage “chunk” (increment)
    - KEEP|FREE - What to do when done with inc
      - KEEP Do not freemain the storage
      - FREE Freemain the storage

    NOTE: No ANY|BELOW, LIBSTACK always below the 16M line

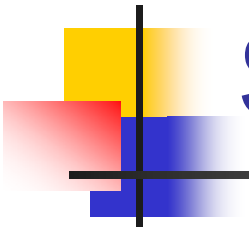
Stacks are simple...

- Run-time options dealing with stacks
  - STORAGE(...,...,dsa_alloc,...)
    - 3rd sub-option allows initialization of DSA
    - NONE - no initialization
    - Hex value - DSA initialized with single byte
      - Yes there is overhead with this initialization
      - Don't use this – have program initialize its own variables



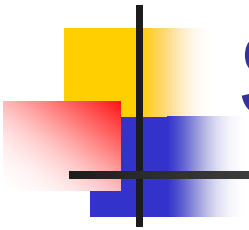
Stacks are simple...

- Run-time options dealing with stacks
  - THREADSTACK(ON|OFF,init,inc,ANY|BELOW,KEEP|FREE, dsInit,dsInc)
    - ON|OFF – Whether or not to use THREADSTACK for pthreads
    - Init - Initial size of storage “chunk” (like STACK)
    - Inc - Increment size of storage “chunk” (like STACK)
    - ANY|BELOW - Location of storage
      - ANY Anywhere in 2G virtual storage
      - Below Always below 16M line
        - Required when all31(OFF)
    - KEEP|FREE - What to do when done with inc
      - KEEP Do not freemain the storage
      - FREE Freemain the storage
    - DsInit, Dsinc – XPLINK “chunk” sizes



Stacks are simple...

- Run-time options dealing with stacks
  - RPTSTG(ON|OFF)
    - Produces a storage tuning report when ON
      - Generates suggested initial sizes for all LE managed storage
        - STACKs and HEAPs
        - Large overhead when ON – use for tuning only – not production

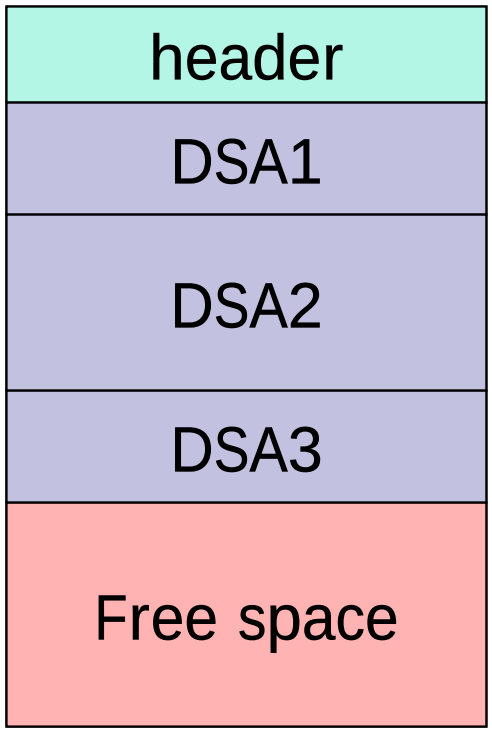


Stacks are simple...

- DATA
  - “Chunks” are called stack segments
    - Made up of 1 or more DSAs
  - DSA – Dynamic Save Area
    - Also called a “stack frame”
  - DSAs contain
    - Register Save Area (RSA)
    - NAB – Next Available Byte
    - Automatic (local) variables
      - C – int i;
      - PL/I – declare i fixed;
      - NOT COBOL working storage
        - COBOL LOCAL storage in stack

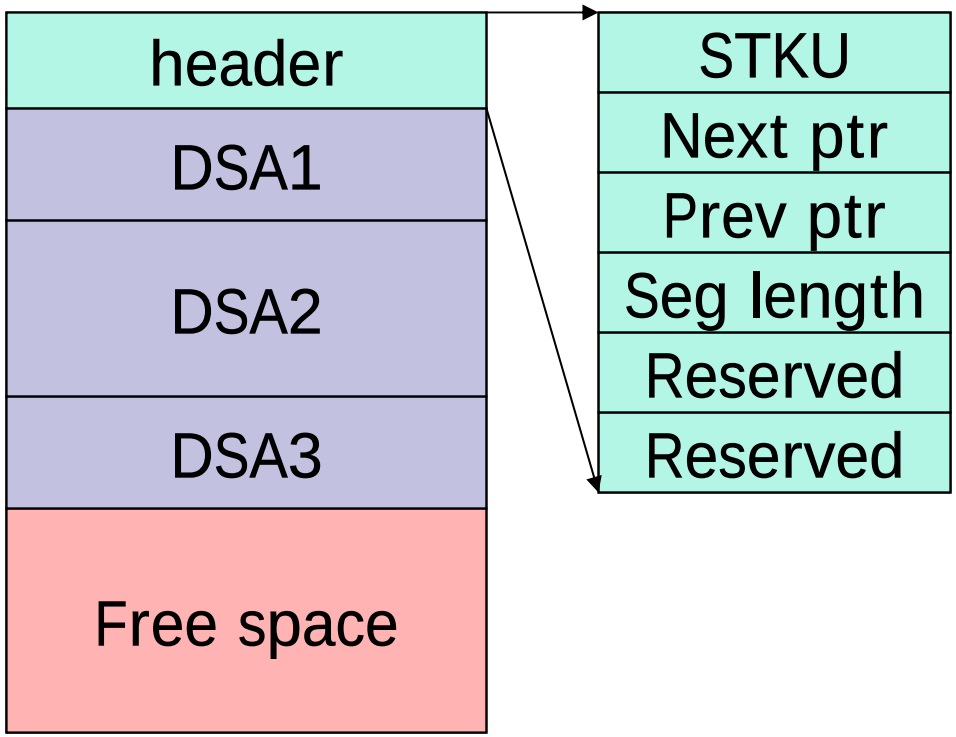
Stacks are simple...

- Layout – simple stack segment



Stacks are simple...

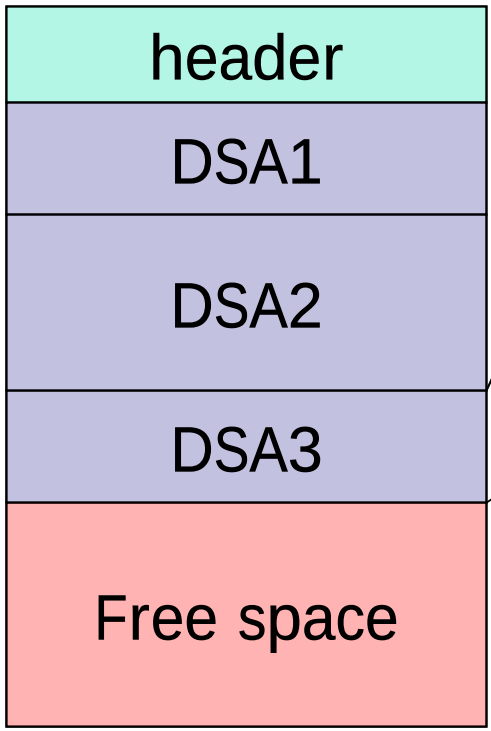
- Layout – header information



- x'18' bytes long
- Eyecatcher
  - STKU (user)
  - STKL (library)
- Next and Prev pointer
  - circular chain to SMCB
- Segment length includes header and free space
- Last 2 words reserved
  - normally 0

Stacks are simple...

- Layout – Dynamic Save Area

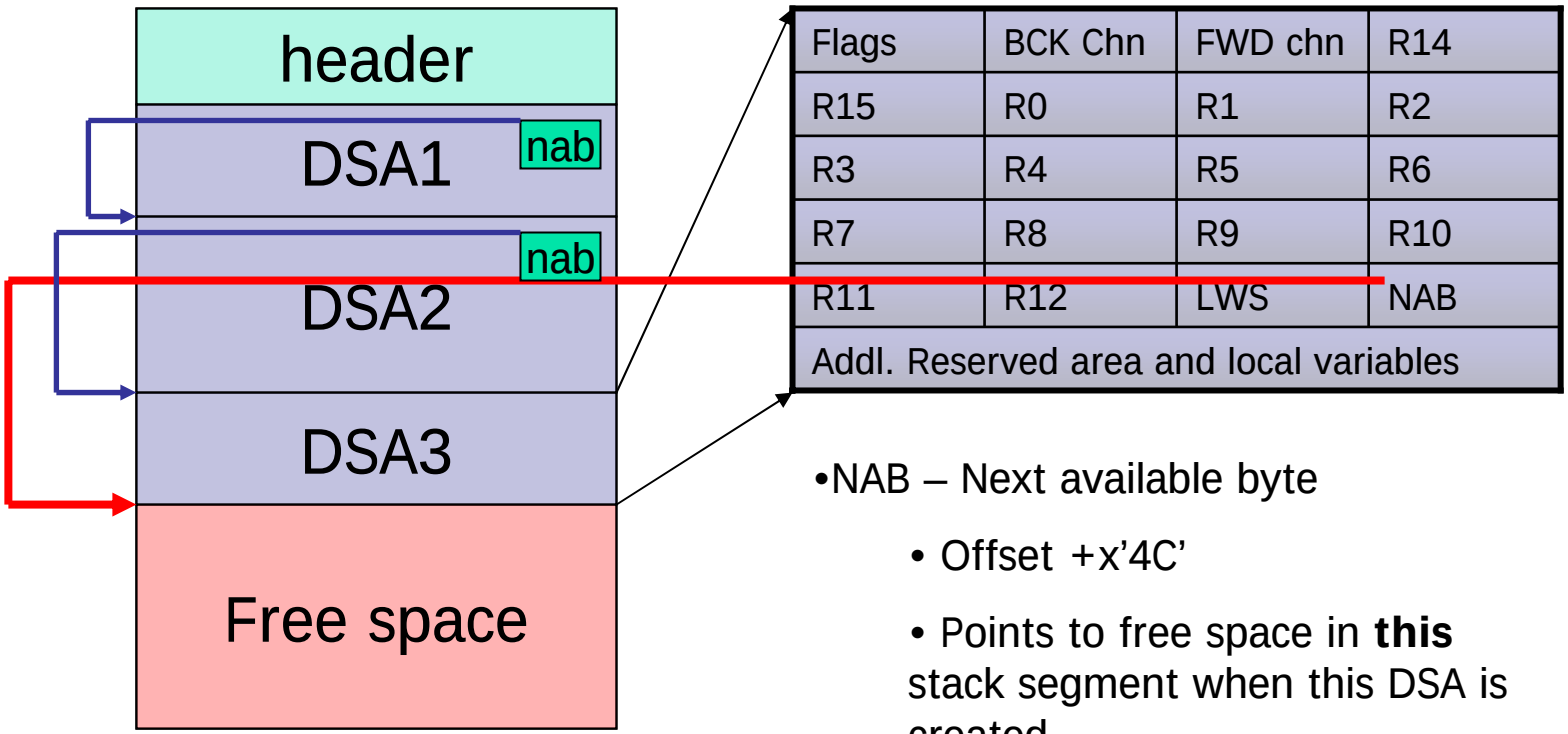


Flags	BCK Chn	FWD chn	R14
R15	R0	R1	R2
R3	R4	R5	R6
R7	R8	R9	R10
R11	R12	LWS	NAB
Addl. Reserved area and local variables			

- Starts with standard save area
- Possibly no forward chain pointer
- Possibly not all regs saved

Stacks are simple...

- Layout – Dynamic Save Area

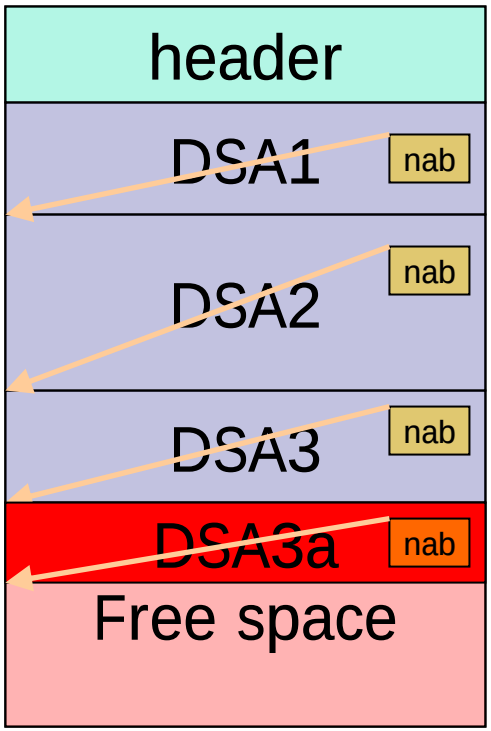


•NAB – Next available byte

- Offset +x'4C'
- Points to free space in **this** stack segment when this DSA is created.

Stacks are simple...

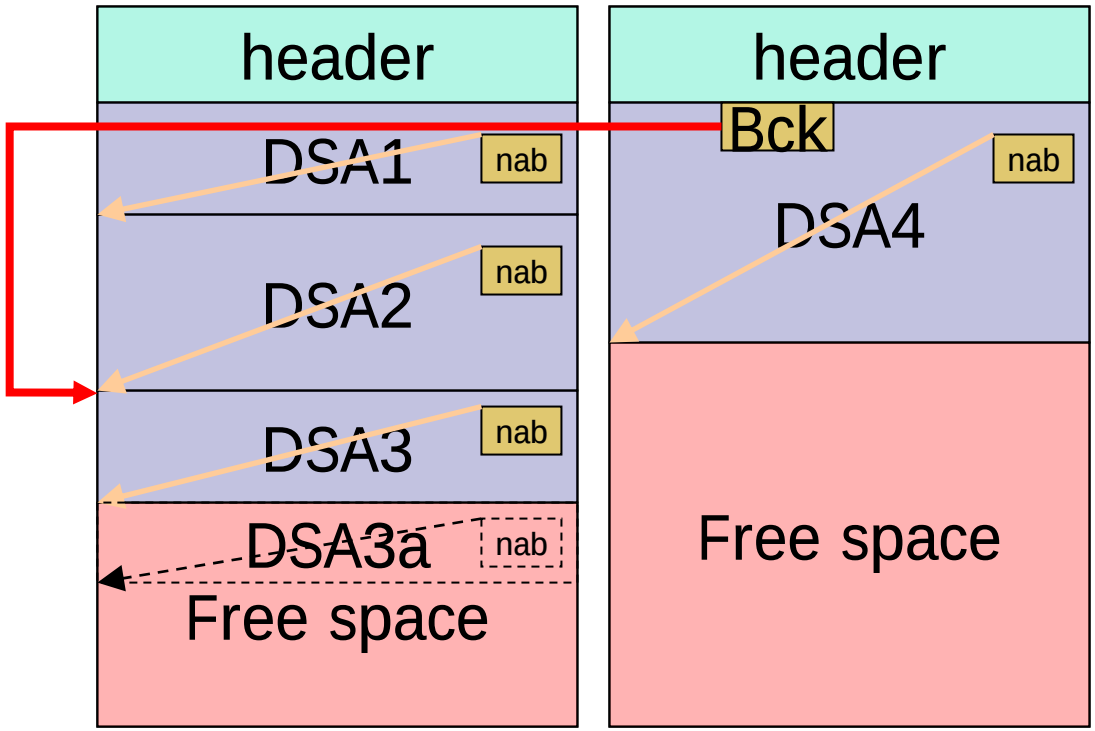
- Layout – NAB is not forward chain ptr



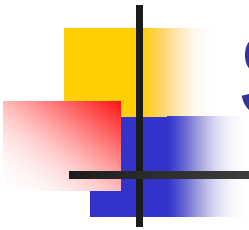
- Program 3 calls Program 3a
- Program 3a returns

Stacks are simple...

- Layout – NAB is not forward chain ptr

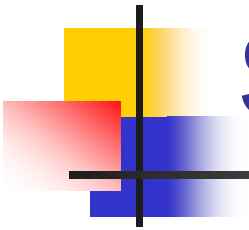


- Program 3 calls Program 3a
- Program 3a returns
  - DSA3a is residual data
- Program 3 calls Program 4
  - Not enough room
- NAB points to free space in the 2nd stack segment
- Backchain points into 1st segment



Stacks are simple...

- Processing - Simple case
  - Program (or function) A calls Program B
    - R13 must point to Prog A's DSA
  - "Prologue Code" in Program B executes
    - Saves Regs in Prog A's DSA
    - Determines size of DSA Prog B requires
    - Checks NAB in Prog A's DSA to determine if stack segment can contain Prog B's DSA
    - Uses area pointed to by NAB as Prog B's DSA
    - Stores new NAB in new DSA
    - Updates Backchain in new DSA



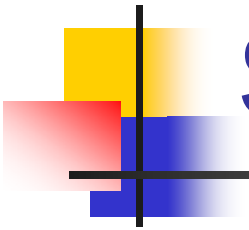
Stacks are simple...

- Processing - Simple case - continued
  - "Epilogue Code" executes to return to Prog A.
    - R13 updated from backchain pointer
    - Registers loaded from Prog A's DSA (R13)
    - Control returned to Prog A via R14
  - NOTES:
    - Prog B's save area not cleared
      - May be useful for debug purposes.
    - No NAB processing takes place
      - NAB in PROG A's DSA again valid

Stacks are simple...

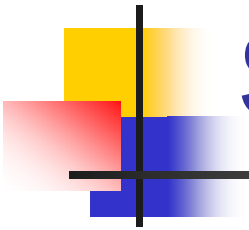
- Processing - Simple case - continued
  - Sample prolog code

```
00028E  90E6  D00C  STM   r14,r6,12(r13)   Save registers we will be using in caller's DSA
000292  58E0  D04C  L     r14,76(,r13)     Get the NAB from callers save area
000296  4100  E0A8  LA    r0,168(,r14)     Add the size we need (x'A8')
00029A  5500  C314  CL    r0,788(,r12)     Is this still within this segment
00029E  4720  F014  BH    20(,r15)         If not, go get another segment
0002A2  58F0  C280  L     r15,640(,r12)    Return to here. R14=new DSA, R0=new NAB
0002A6  90F0  E048  STM   r15,r0,72(r14)   Update new NAB
0002AA  9210  E000  MVI   0(r14),16        Update flags in DSA
0002AE  50D0  E004  ST    r13,4(,r14)      Update backchain pointer
0002B2  18DE        LR    r13,r14          Set R13 to be the current DSA
0002B4  0530        BALR  r3,r0            Set addressability and go...
0002B6        End of Prolog
```



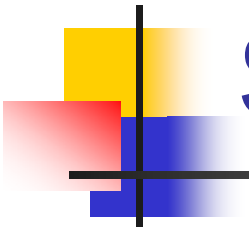
Stacks are simple...

- Processing - Less Simple case
  - Program (or function) A calls Program B
    - R13 must point to Prog A's DSA
  - "Prologue Code" in Program B executes
    - Saves Regs in Prog A's DSA
    - Determines size of DSA Prog B requires
    - Checks NAB in Prog A's DSA to determine if stack segment can contain Prog B's DSA
    - NOT ENOUGH ROOM!
      - One of LE's stack overflow routines gets called
      - A new stack segment is created
      - Extra DSA may be inserted into new segment



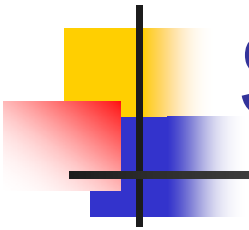
Stacks are simple...

- Processing - Less Simple case (continued...)
  - "Prologue Code" in Program B execution...
    - Inserted DSA will be for module CEEVSSFR
      - Used to update SMCB information
      - Used to FREEMAIN stack with FREE option
    - Uses address returned from stack overflow routine as Prog B's DSA
    - Stores new NAB in new DSA residing in new segment
    - Updates Backchain in new DSA



Stacks are simple...

- Processing - Less Simple case - continued
  - "Epilogue Code" executes to return to Prog A.
    - R13 updated from backchain pointer
    - Regs loaded from previous save area (R13)
    - Control is returned via R14
    - Return to CEEVSSFR, segment is collapsed.
    - R13 updated from backchain pointer
    - Registers loaded from Prog A's DSA (R13)
    - Control returned to Prog A via R14
  - NOTES:
    - Prog B's save area not cleared but possibly freed
    - No NAB processing takes place



Stacks are simple...

- Simple application
  - COBOL program dynamically calls an amode 24 assembler program (non-LE enabled).
  - Assembler program then calls other assembler which calls a second COBOL program.
- Result
  - U4088 RC=63 ABEND

Stacks are simple...

- SYSMDUMP - Use IPCS Verbexit
  - VERBX LEDATA ‘STACK’
    - Storage Management Control Block (SMCB)

```
Stack Storage Control Blocks

    SMCB:  00017178
  +000000  EYE_CATCHER:SMCB  US_EYE_CATCHER:USTK     USFIRST:00027000
  +00000C  USLAST:00027000   USBOS:00027000    USEOS:00047000
  +000018  USNAB:F0F00000    USINITSZ:00020000       USINCRSZ:00020000
  +000024  USANYBELOW:80000000     USKEEPFREE:00000000     USPOOL:80000002
  +000030  USPREALLOC:00000001     US_BYTES_ALLOC:00000000
  +000038  US_CURR_ALLOC:00000000        US_GETMAINS:00000000
  +000040  US_FREEMAINS:00000000   US_OPLINK:00      LS_THIS_IS:LSTK
  +00004C  LSFIRST:00026000        LSLAST:00026000   LSBOS:00026000
  +000058  LSEOS:00027000    LSNAB:00026018    LSINITSZ:00001000
  +000064  LSINCRSZ:00001000       LSANYBELOW:80000000
  +00006C  LSKEEPFREE:00000001     LSPOOL:80000001   LSPREALLOC:00000001
  +000078  LS_BYTES_ALLOC:00000000       LS_CURR_ALLOC:00000000
  +000080  LS_GETMAINS:00000000    LS_FREEMAINS:00000000   LS_OPLINK:00
```

Stacks are simple...

- SYSMDUMP - Use IPCS Verbexit
  - User stack begins at 00027000
  - Ends at 00047000 – (length 00020000)

```
Stack Storage Control Blocks

     SMCB:  00017178
   +000000  EYE_CATCHER:SMCB  US_EYE_CATCHER:USTK     USFIRST:00027000
   +00000C  USLAST:00027000   USBOS:00027000    USEOS:00047000
   +000018  USNAB:F0F00000    USINITSZ:00020000       USINCRSZ:00020000
   +000024  USANYBELOW:80000000     USKEEPFREE:00000000     USPOOL:80000002
   +000030  USPREALLOC:00000001     US_BYTES_ALLOC:00000000
   +000038  US_CURR_ALLOC:00000000        US_GETMAINS:00000000
   +000040  US_FREEMAINS:00000000   US_OPLINK:00      LS_THIS_IS:LSTK
   +00004C  LSFIRST:00026000        LSLAST:00026000   LSBOS:00026000
   +000058  LSEOS:00027000    LSNAB:00026018    LSINITSZ:00001000
   +000064  LSINCRSZ:00001000       LSANYBELOW:80000000
   +00006C  LSKEEPFREE:00000001     LSPOOL:80000001   LSPREALLOC:00000001
   +000078  LS_BYTES_ALLOC:00000000       LS_CURR_ALLOC:00000000
   +000080  LS_GETMAINS:00000000    LS_FREEMAINS:00000000   LS_OPLINK:00
```

Stacks are simple...

- SYSMDUMP - Use IPCS Verbexit
  - Library stack begins at 00026000
  - Ends at 00027000 – (length 00001000)

```
Stack Storage Control Blocks

    SMCB:  00017178
  +000000  EYE_CATCHER:SMCB  US_EYE_CATCHER:USTK     USFIRST:00027000
  +00000C  USLAST:00027000   USBOS:00027000    USEOS:00047000
  +000018  USNAB:F0F00000    USINITSZ:00020000       USINCRSZ:00020000
  +000024  USANYBELOW:80000000     USKEEPFREE:00000000     USPOOL:80000002
  +000030  USPREALLOC:00000001     US_BYTES_ALLOC:00000000
  +000038  US_CURR_ALLOC:00000000        US_GETMAINS:00000000
  +000040  US_FREEMAINS:00000000   US_OPLINK:00       LS_THIS_IS:LSTK
  +00004C  LSFIRST:00026000        LSLAST:00026000   LSBOS:00026000
  +000058  LSEOS:00027000    LSNAB:00026018    LSINITSZ:00001000
  +000064  LSINCRSZ:00001000       LSANYBELOW:80000000
  +00006C  LSKEEPFREE:00000001     LSPOOL:80000001   LSPREALLOC:00000001
  +000078  LS_BYTES_ALLOC:00000000       LS_CURR_ALLOC:00000000
  +000080  LS_GETMAINS:00000000    LS_FREEMAINS:00000000   LS_OPLINK:00
```

Stacks are simple...

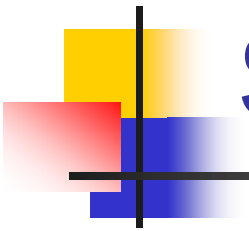
- SYSMDUMP – Next are the DSAs
  - DSA is not in an LE stack segment
    - Not in 00026000-00047000 (this is ok)

```
DSA backchain

   DSA:  00007834
+000000  FLAGS:0000  MEMD:0000   BKC:A00078A4      FWC:00000000
+00000C  R14:00007828      R15:9F935858      R0:9F935858
+000018  R1:000004F5       R2:000187FC       R3:1F8009AE
+000024  R4:000270B8       R5:00000000       R6:00047370
+000030  R7:00000000       R8:8001B3F0       R9:00007800
+00003C  R10:00047038      R11:8001B128      R12:00016A88
+000048  LWS:00000000      NAB:00000000      PNAB:00000000
+000064  RENT:90ECD00C     CILC:189F1BFE     MODE:80012DEA
+000078  RMR:00000000
```

Stacks are simple...

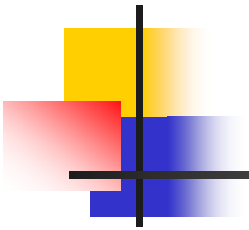
- SYSMDUMP – Next are the DSAs
  - Backchain is not a valid 31 bit address
    - ‘Dirty’ high order byte
    - Results in U4088 RC=63

```
DSA backchain

   DSA:  00007834
+000000  FLAGS:0000  MEMD:0000   BKC:A00078A4      FWC:00000000
+00000C  R14:00007828      R15:9F935858      R0:9F935858
+000018  R1:000004F5       R2:000187FC       R3:1F8009AE
+000024  R4:000270B8       R5:00000000       R6:00047370
+000030  R7:00000000       R8:8001B3F0       R9:00007800
+00003C  R10:00047038      R11:8001B128      R12:00016A88
+000048  LWS:00000000      NAB:00000000      PNAB:00000000
+000064  RENT:90ECD00C     CILC:189F1BFE     MODE:80012DEA
+000078  RMR:00000000
```



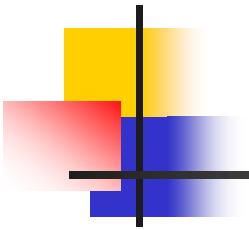
Stacks are simple...

- Summary of stacks
  - LE Stacks used for save areas and local variables
  - You can no longer count on forward chain pointer
  - NAB is not really a forward chain pointer
  - All regs may not be saved (especially in C and C++)
  - Compilers (not run-time) drive the management of stacks.



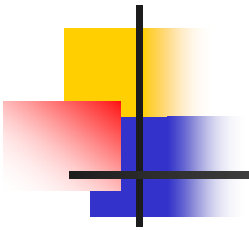
...Heaps are fun!

- Language Environment Storage Management
  - Heaps
    - Completely random access
    - Allows storage to be dynamically allocated at runtime
  - Enclave level control structures
    - Each 'main' has a separate stack and heap
    - Each 'link' causes a separate stack and heap
    - pthreads share a single heap for all threads



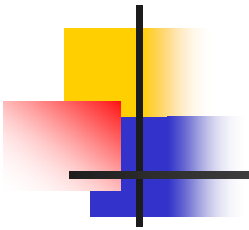
...Heaps are fun!

- Language Environment Storage Management
  - Heaps
  - Four independently maintained sets of heap segments all with similar layouts:
    - User Heap
      - COBOL W/S
      - C/C++ (malloc or operator new)
      - PL/1 dynamic storage (allocate)
    - LE Anywhere Heap
      - COBOL and LE above the line CBs
    - LE Below Heap
      - COBOL and LE below the line CBs
    - Additional Heap
      - Defined by the userall threads



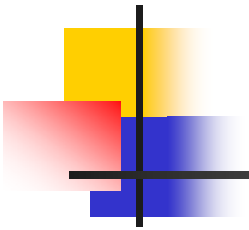
...Heaps are fun!

- Runtime options dealing with the heaps
  - HEAP(init,inc,ANY|BELOW,KEEP|FREE,int24,inc24)
    - User heap - mostly application use
    - init - Initial size of the "chunk" of storage obtained to be managed by LE for user heap
    - Inc - When initial "chunk" is full, size of next "chunk" (minimum)
    - ANY|BELOW - Location of "chunk“
      - Not sensitive to ALL31 setting
    - KEEP | FREE - What to do when done with the increment when empty
      - KEEP - Do not freemain the storage
      - FREE - Freemain the storage
    - int24 - Inital size of the "chunk" of storage obtained
      - (if ANY specified but BELOW requested (minimum))
    - inc24 - Size of next "chunk"
      - (if ANY specified but BELOW requested (minimum)



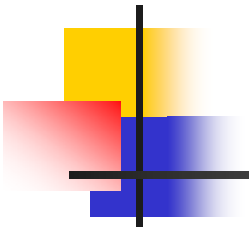
...Heaps are fun!

- Runtime options dealing with the heaps...
  - ANYHEAP(init,inc,ANY|BELOW,KEEP|FREE)
    - LE use - normally above the line
    - init - Same as HEAP.
    - inc - Same as HEAP. (minimum)
    - ANY | BELOW - Location of storage
    - KEEP | FREE - Same as HEAP
  - BELOWHEAP(init,inc,KEEP|FREE)
    - LE use - always below the line
    - init - Same as HEAP.
    - inc - Same as HEAP. (minimum)
    - KEEP | FREE - Same as HEAP



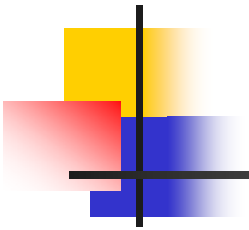
...Heaps are fun!

- Runtime options dealing with the heap...
  - STORAGE(getheap, freeheap,...)
    - Initialize heap storage.
    - getheap - NONE or one byte hex value to initialize storage when heap element obtained
      - 00 similar to WSCLEAR option
    - freeheap - NONE or one byte hex value to initialize storage when heap element freed
      - Useful for debug purposes or security
  - There is overhead for both getheap and freeheap when not NONE
    - Overhead can be significant with large heaps causing all pages to be accessed.
    - No STOP sign on this page



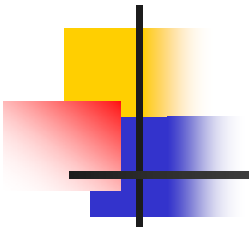
...Heaps are fun!

- Runtime options dealing with the heap…
  - RPTSTG(ON|OFF)
    - Produces report with tuning information.
    - Not to be used in production
  - HEAPCHK(ON|OFF,freq,delay,depth,pooldepth)
    - More later….



...Heaps are fun!

- Heap Layout
  - Much more complicated than Stack
  - Based on algorithm from IBM's Watson Research Center
    - FAST 1ST!!!, Storage Eff 2nd, Error checks 3rd
  - Elements of user requested length
    - User responsible for requesting storage be freed.

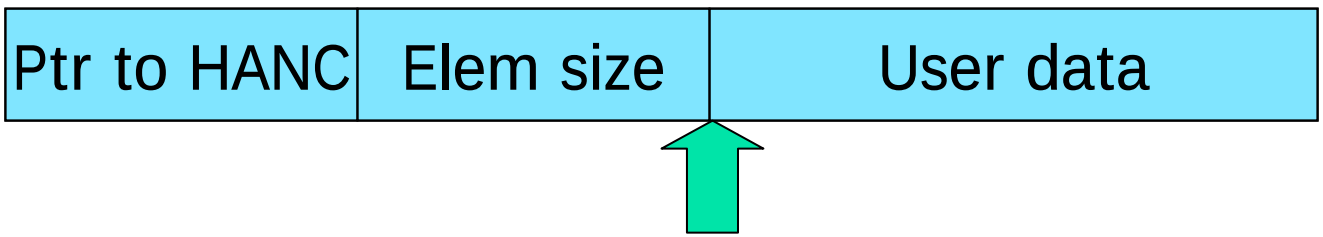


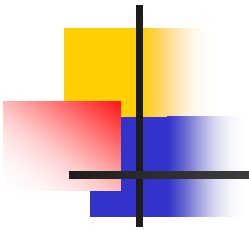
...Heaps are fun!

- Heap Layout continued...
  - Header
    - x'20' byte header (8 fullwords of data)
      - Eyecatcher 'HANC'
      - Pointer to next Heap segment or HPCB
      - Pointer to previous Heap segment or HPCB
      - Heapid (user heap = 0)
      - Pointer to beginning of segment
      - Root address (largest free element in segment)
      - Heap Segment Length
      - Root element length (size of largest free element)

...Heaps are fun!

- Heap Layout continued...
  - Allocated element
    - 8 byte header
      - Pointer to beginning of heap segment (HANC)
      - Size of element including header
    - User portion
      - Address returned to user



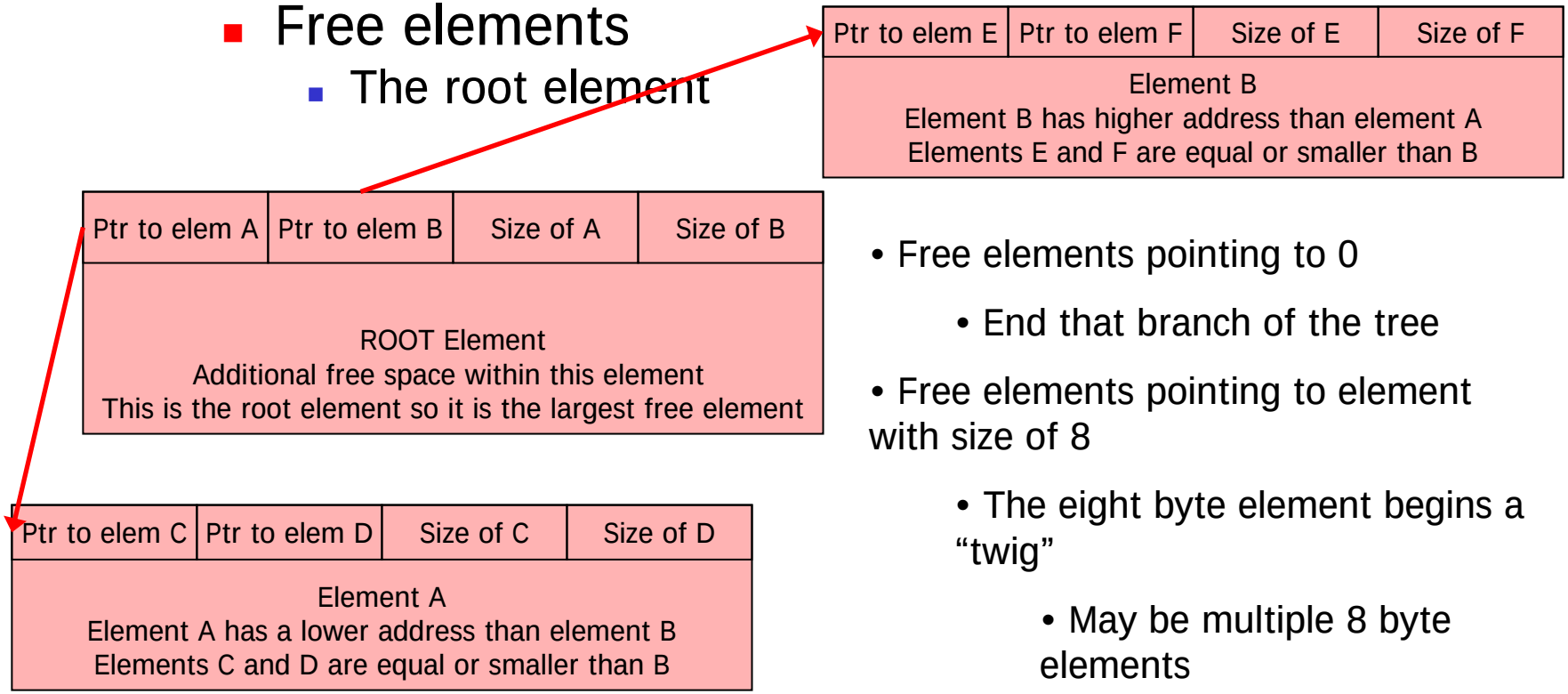


...Heaps are fun!

- Heap Layout continued...
  - Free elements
    - Maintained in a Cartesian Tree
      - Larger elements toward the root
      - Smaller elements toward the leaves
      - Lower addresses to the left
      - Higher addresses to the right
  - Each free area contains x'10' bytes of information about OTHER free elements.
    - Left node address
    - Right node address
    - Left node size
    - Right nonde size

...Heaps are fun!

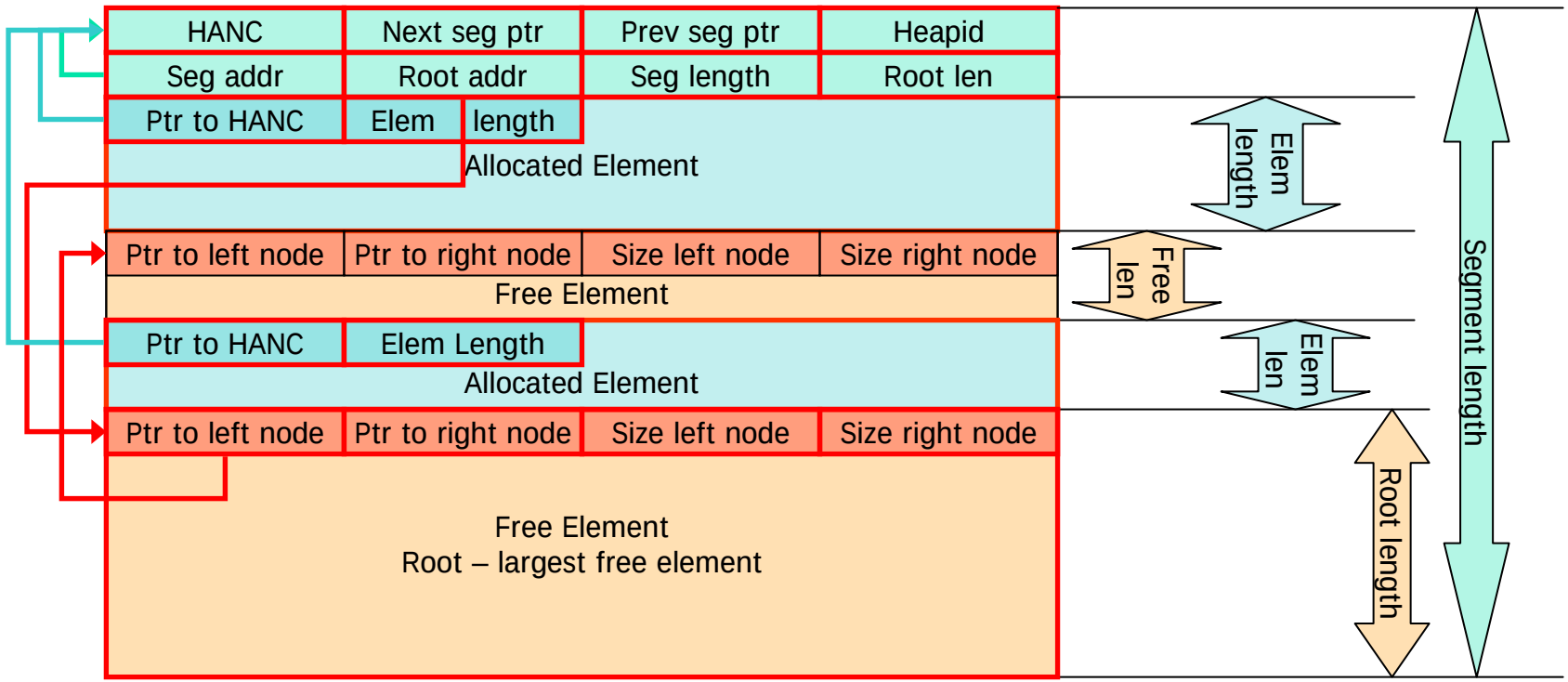
- Heap Layout continued...
  - Free elements
    - The root element

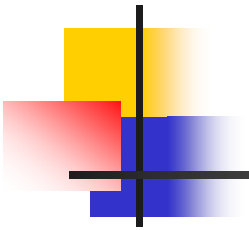


- Free elements pointing to 0
  - End that branch of the tree
- Free elements pointing to element with size of 8
  - The eight byte element begins a “twig”
    - May be multiple 8 byte elements

...Heaps are fun!

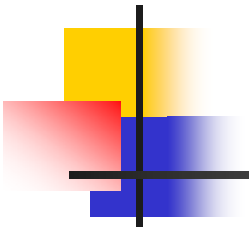
- Heap Layout continued... A “simple” example





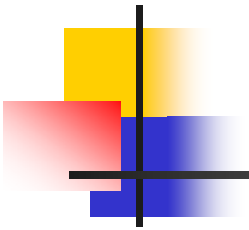
...Heaps are fun!

- Processing
  - Allocation
    - The Free Element tree of the latest heap segment is searched starting at the root for the smallest element which will satisfy the request.
    - If no free element found in that segment earlier segments are searched.
    - If no free element is large enough to satisfy the request, an additional heap segment is allocated via GETMAIN.
    - The needed storage from the free element is then allocated (8 byte header filled out).
      - If additional storage is left over in the element it is added to the free tree as a new free element.
    - A pointer to the user storage (after the 8 byte header) is returned to the user.



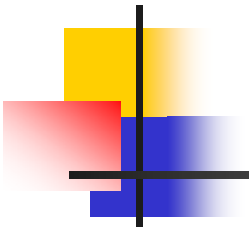
...Heaps are fun!

- Processing
  - Free
    - Size of element is determined from the 8 byte header.
    - Element (including header) is returned to the free tree
      - If free element is adjacent to an existing free element the elements are combined into a larger free element
      - The free tree will be restructured as necessary.
    - If the entire heap is now free the segment will be FREEMAINed if FREE is specified.



...Heaps are fun!

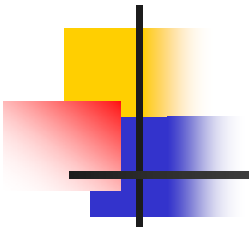
- Hands on Debug using a COBOL program



...Heaps are fun!

- Sample program to cause heap damage

```
000100       IDENTIFICATION DIVISION.
000200       PROGRAM-ID. SHAREHP.
000300
000400       DATA DIVISION.
000500       WORKING-STORAGE SECTION.
000600       01  WS-EYE              PIC X(4) VALUE "WS-B".
000700       01  HEAPID              PIC S9(9) BINARY VALUE 0.
000800       01  STORAGE-SIZE-24       PIC S9(9) BINARY VALUE 24.
000900       01  STORAGE-SIZE-16       PIC S9(9) BINARY VALUE 16.
001000       01  STORAGE-SIZE-8       PIC S9(9) BINARY VALUE 8.
001100       01  ADDRESS-0             POINTER.
001200       01  ADDRESS-1             POINTER.
001300       01  ADDRESS-2             POINTER.
001400       01  WS-DATA.
001500         03 WS-DATA1             PIC X(16) VALUE "1234567890123456".
001600         03 WS-DATA2             PIC S9(9) BINARY VALUE 0.
...
003100       LINKAGE SECTION.
003200       01 HEAP-DATA-AREA      PIC X(17).
003300       PROCEDURE DIVISION.
```



...Heaps are fun!

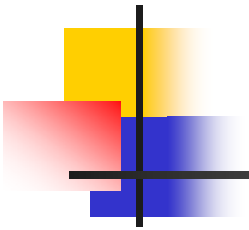
- Sample program to cause heap damage...

```
003400       MAIN-PROG.
003500           DISPLAY "STARTING SHAREHP..."
003600           CALL "CEEGTST" USING HEAPID, STORAGE-SIZE-16,
003700                                ADDRESS-0, FC.
003800
003900           CALL "CEEGTST" USING HEAPID, STORAGE-SIZE-16,
004000                                ADDRESS-1, FC.
004100
004200           CALL "CEEGTST" USING HEAPID, STORAGE-SIZE-16,
004300                                 ADDRESS-2, FC.
004400
004500           CALL "CEEFRST" USING ADDRESS-1, FC.
004600
004700           SET ADDRESS OF HEAP-DATA-AREA TO ADDRESS-2.
004800           MOVE WS-DATA TO HEAP-DATA-AREA.
004900
005000           CALL "CEEGTST" USING HEAPID, STORAGE-SIZE-8,
005100                                ADDRESS-1, FC.
005200           GOBACK.
```

...Heaps are fun!

- Sample program job log

```
08.51.19 JOB25721 ---- WEDNESDAY, 16 FEB 2005 ----
08.51.20 JOB25721  $HASP373 JMONTIGO STARTED - WLM INIT  - SRVCLASS WLMLONG  - SYS AQFT
08.51.20 JOB25721  IEF403I JMONTIGO - STARTED - TIME=08.51.20
08.51.23 JOB25721  IEA995I SYMPTOM DUMP OUTPUT  345
   345               USER COMPLETION CODE=4039 REASON CODE=00000000
   345              TIME=08.51.21  SEQ=09748  CPU=0000  ASID=00BA
   345              PSW AT TIME OF ERROR  078D1000   A01ECE96  ILC 2  INTC 0D
   345                ACTIVE LOAD MODULE           ADDRESS=201256C0  OFFSET=000C77D6
   345                NAME=CEEPLPKA
   345                DATA AT PSW  201ECE90 - 00181610  0A0D58D0  D00498EC
   345                AR/GR 0: 80AB5B3E/84000000   1: 00000000/84000FC7
   345                      2: 00000000/20381F88   3: 00000000/00000002
   …
   345                      E: 00000000/A01E0DDE   F: 00000000/00000000
   345              END OF SYMPTOM DUMP
08.51.23 JOB25721  IEA993I SYSMDUMP TAKEN TO POSIX.JMONTI.SHARE.HEAP.SYSMDUMP
08.51.23 JOB25721  IEF450I JMONTIGO GO - ABEND=S000 U4038 REASON=00000001  347
   347                     TIME=08.51.23
```

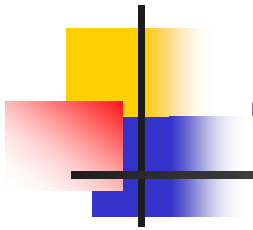


...Heaps are fun!

- Sample program job output

```
STARTING SHAREHP...
CEE3703I In HANC Control Block, the Eye Catcher is damaged.
CEE3704I Expected data at 203A1018 : HANC.
  203A0FF8: 00000000 00000000 00000000 00000000  00000000 00000000 00000000
00000000  |................................|
  203A1018: C8C1D5C3 0001D000 201230B8 00000000  A03A1018 203A1080 00008000
00007F98  |HANC.........................."q|
CEE0802C Heap storage control information was damaged.
         From compile unit SHAREHP at entry point SHAREHP at statement 896 at
compile unit offset +000005C2 at entry offset +000005C2 at address 201011B2.
```

- CEE37xxI messages are new
  - Not designed for this use.
  - Attempt to help with debug, but often not meaningful. LE is "confused".
  - We can make them meaningful (later...)
  - CEE0802C message indicates heap damage



...Heaps are fun!

- Sample program CEEDUMP

```
CEE3DMP V1 R6.0: Condition processing resulted in the unhandled condition.        02/16/05 8:51:23 AM                Page:  1

Information for enclave SHAREHP

  Information for thread 8000000000000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry        E Addr    E  Offset   Statement  Load Mod  Service  Status
    20381648  CEEHDSP       201DD170  +00003F58  CEEHDSP      201DD170  +00003F58              CEEPLPKA  UQ91316  Call
    203814B0  CEEHSGLT      201EF1B8  +0000005C  CEEHSGLT     201EF1B8  +0000005C              CEEPLPKA  HLE7709  Exception
    20381398  CEEV#GTS      202BBFF8  +00000698  CEEV#GTS     202BBFF8  +00000698              CEEPLPKA  HLE7709  Call
    203812E8  CEEVGTST      202CC720  +00000072  CEEVGTST     202CC720  +00000072              CEEPLPKA  HLE7709  Call
    20381100  IGZCFCC       2034AB78  +000002CA  IGZCFCC      2034AB78  +000002CA              IGZCPAC            Call
    20381030  SHAREHP       20100BF0  +000005C2  SHAREHP      20100BF0  +000005C2         896  SHAREHP            Call
```

- Not reported as a program check (software detected)
- CEEV#GTS reported CEE0802 condition to handler
- Tell us who "detected" the error, not who "caused" the error.

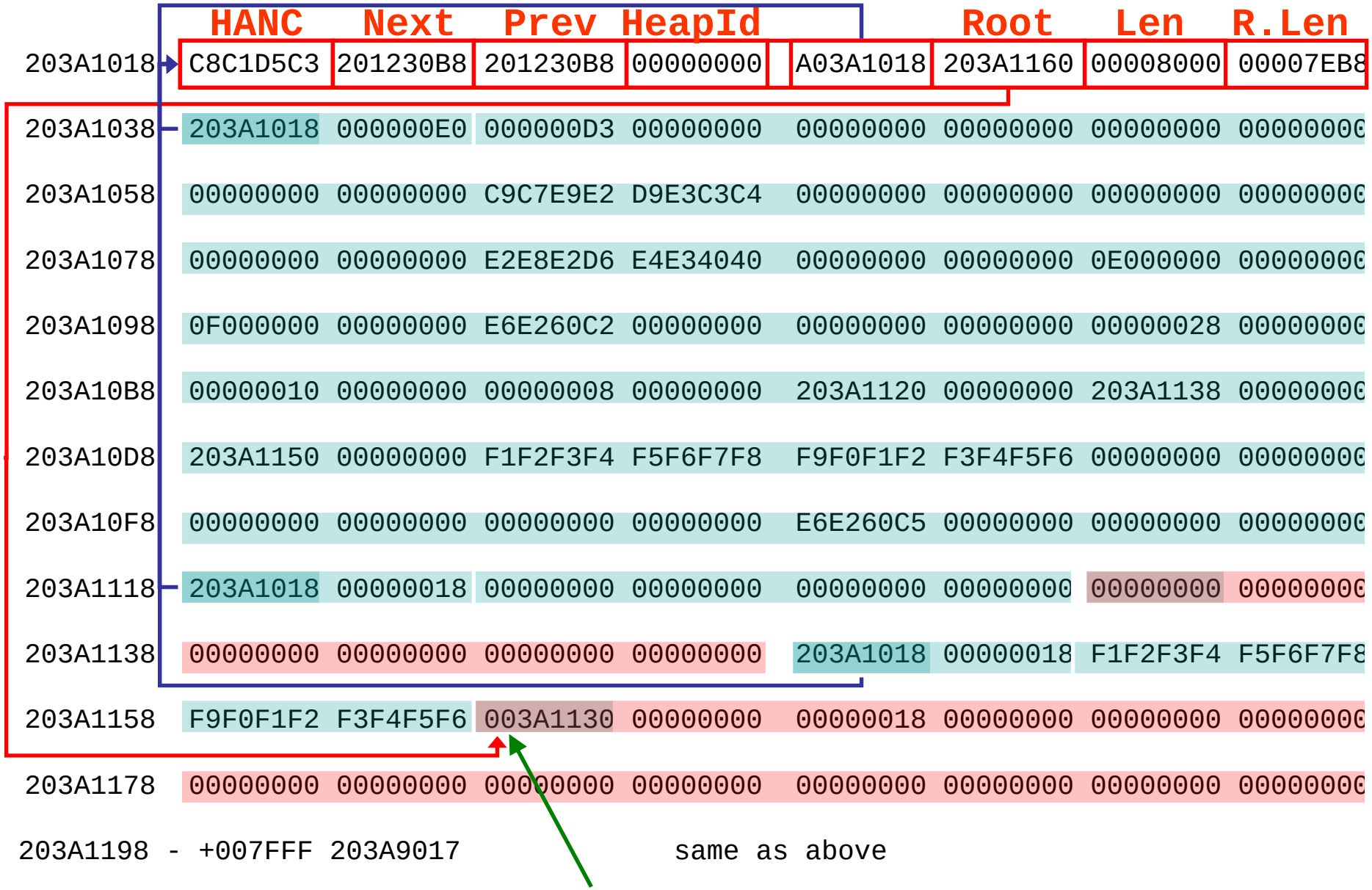
...Heaps are fun!

- Sample program CEEDUMP

```
Initial (User) Heap                                          : 203A1018
  +000000 203A1018  C8C1D5C3 201230B8 201230B8 00000000  A03A1018 203A1160 00008000 00007EB8 |HANC.....................-......=.|
  +000020 203A1038  203A1018 000000E0 000000D3 00000000  00000000 00000000 00000000 00000000 |...........L....................|
  +000040 203A1058  00000000 00000000 C9C7E9E2 D9E3C3C4  00000000 00000000 00000000 00000000 |........IGZSRTCD................|
  +000060 203A1078  00000000 00000000 E2E8E2D6 E4E34040  00000000 00000000 0E000000 00000000 |........SYSOUT  ................|
  +000080 203A1098  0F000000 00000000 E6E260C2 00000000  00000000 00000000 00000028 00000000 |........WS-B....................|
  +0000A0 203A10B8  00000010 00000000 00000008 00000000  203A1120 00000000 203A1138 00000000 |................................|
  +0000C0 203A10D8  203A1150 00000000 F1F2F3F4 F5F6F7F8  F9F0F1F2 F3F4F5F6 00000000 00000000 |...&....1234567890123456........|
  +0000E0 203A10F8  00000000 00000000 00000000 00000000  E6E260C5 00000000 00000000 00000000 |................WS-E............|
  +000100 203A1118  203A1018 00000018 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
  +000120 203A1138  00000000 00000000 00000000 00000000  203A1018 00000018 F1F2F3F4 F5F6F7F8 |........................12345678|
  +000140 203A1158  F9F0F1F2 F3F4F5F6 003A1130 00000000  00000018 00000000 00000000 00000000 |90123456........................|
  +000160 203A1178  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
  +000180 203A1198 - +007FFF 203A9017             same as above
```

- Damage normally occurs in User Heap so check there first

```
203A1018  C8C1D5C3 201230B8 201230B8 00000000  A03A1018 203A1160 00008000 00007EB8
203A1038  203A1018 000000E0 000000D3 00000000  00000000 00000000 00000000 00000000
203A1058  00000000 00000000 C9C7E9E2 D9E3C3C4  00000000 00000000 00000000 00000000
203A1078  00000000 00000000 E2E8E2D6 E4E34040  00000000 00000000 0E000000 00000000
203A1098  0F000000 00000000 E6E260C2 00000000  00000000 00000000 00000028 00000000
203A10B8  00000010 00000000 00000008 00000000  203A1120 00000000 203A1138 00000000
203A10D8  203A1150 00000000 F1F2F3F4 F5F6F7F8  F9F0F1F2 F3F4F5F6 00000000 00000000
203A10F8  00000000 00000000 00000000 00000000  E6E260C5 00000000 00000000 00000000
203A1118  203A1018 00000018 00000000 00000000  00000000 00000000 00000000 00000000
203A1138  00000000 00000000 00000000 00000000  203A1018 00000018 F1F2F3F4 F5F6F7F8
203A1158  F9F0F1F2 F3F4F5F6 003A1130 00000000  00000018 00000000 00000000 00000000
203A1178  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
203A1198 - +007FFF 203A9017                    same as above
```

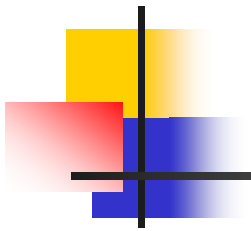


...Heaps are fun!

- Sample program CEEDUMP

```
Initial (User) Heap                                          : 203A1018
  +000000 203A1018  C8C1D5C3 201230B8 201230B8 00000000  A03A1018 203A1160 00008000 00007EB8  |HANC.....................-......=.|
  +000020 203A1038  203A1018 000000E0 000000D3 00000000  00000000 00000000 00000000 00000000  |...........L....................|
  +000040 203A1058  00000000 00000000 C9C7E9E2 D9E3C3C4  00000000 00000000 00000000 00000000  |........IGZSRTCD................|
  +000060 203A1078  00000000 00000000 E2E8E2D6 E4E34040  00000000 00000000 0E000000 00000000  |........SYSOUT  ................|
  +000080 203A1098  0F000000 00000000 E6E260C2 00000000  00000000 00000000 00000028 00000000  |........WS-B....................|
  +0000A0 203A10B8  00000010 00000000 00000008 00000000  203A1120 00000000 203A1138 00000000  |................................|
  +0000C0 203A10D8  203A1150 00000000 F1F2F3F4 F5F6F7F8  F9F0F1F2 F3F4F5F6 00000000 00000000  |...&....1234567890123456........|
  +0000E0 203A10F8  00000000 00000000 00000000 00000000  E6E260C5 00000000 00000000 00000000  |................WS-E............|
  +000100 203A1118  203A1018 00000018 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +000120 203A1138  00000000 00000000 00000000 00000000  203A1018 00000018 F1F2F3F4 F5F6F7F8  |........................12345678|
  +000140 203A1158  F9F0F1F2 F3F4F5F6 003A1130 00000000  00000018 00000000 00000000 00000000  |90123456.....................|
  +000160 203A1178  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +000180 203A1198 - +007FFF 203A9017             same as above
```

- The overlay resulted in the CEE0802C error.
- This is a simple case with a single small heap.
- Large heaps and/or multiple heaps are difficult to diagnose by hand.
- Problem 1!!!!



...Heaps are fun!

- Problem 2 – modify the program slightly

```
003600           CALL "CEEGTST" USING HEAPID, STORAGE-SIZE-16,
003700                                 ADDRESS-0, FC.
003800
003900           CALL "CEEGTST" USING HEAPID, STORAGE-SIZE-16,
004000                                 ADDRESS-1, FC.
004100
004200           CALL "CEEGTST" USING HEAPID, STORAGE-SIZE-16,
004300                                   ADDRESS-2, FC.
004400
004500           CALL "CEEFRST" USING ADDRESS-1, FC.
004600
004700           SET ADDRESS OF HEAP-DATA-AREA TO ADDRESS-2.
004800           MOVE WS-DATA TO HEAP-DATA-AREA.
004900
004910           CALL "CEEGTST" USING HEAPID, STORAGE-SIZE-24,
004920                                   ADDRESS-1, FC.
004930
005000           CALL "CEEGTST" USING HEAPID, STORAGE-SIZE-8,
005100                                   ADDRESS-1, FC.
005200           GOBACK.
```

...Heaps are fun!

- Problem 2 – CEEDUMP now looks like:

```
Initial (User) Heap                                                  : 203A1018
  +000000 203A1018  C8C1D5C3 1FF230B8 1FF230B8 00000000  A03A1018 203A1180 00008000 00007E98
  +000020 203A1038  203A1018 000000E0 000000D3 00000000  00000000 00000000 00000000 00000000
  +000040 203A1058  00000000 00000000 C9C7E9E2 D9E3C3C4  00000000 00000000 00000000 00000000
  +000060 203A1078  00000000 00000000 E2E8E2D6 E4E34040  00000000 00000000 0E000000 00000000
  +000080 203A1098  0F000000 00000000 E6E260C2 00000000  00000000 00000000 00000018 00000000
  +0000A0 203A10B8  00000010 00000000 00000008 00000000  203A1120 00000000 203A1168 00000000
  +0000C0 203A10D8  203A1150 00000000 F1F2F3F4 F5F6F7F8  F9F0F1F2 F3F4F5F6 00000000 00000000
  +0000E0 203A10F8  00000000 00000000 00000000 00000000  E6E260C5 00000000 00000000 00000000
  +000100 203A1118  203A1018 00000018 00000000 00000000  00000000 00000000 00000000 00000000
  +000120 203A1138  00000000 00000000 00000000 00000000  203A1018 00000018 F1F2F3F4 F5F6F7F8
  +000140 203A1158  F9F0F1F2 F3F4F5F6 203A1018 00000020  00000018 00000000 00000000 00000000
  +000160 203A1178  00000000 00000000 003A1130 00000000  00000018 00000000 00000000 00000000
  +000180 203A1198  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
  +0001A0 203A11B8 - +007FFF 201A9017                    same as above
```

  - Same overlay occurs with element at 203A1160
  - However damage now moved to 203A1180
  - Why? Bigger problem!

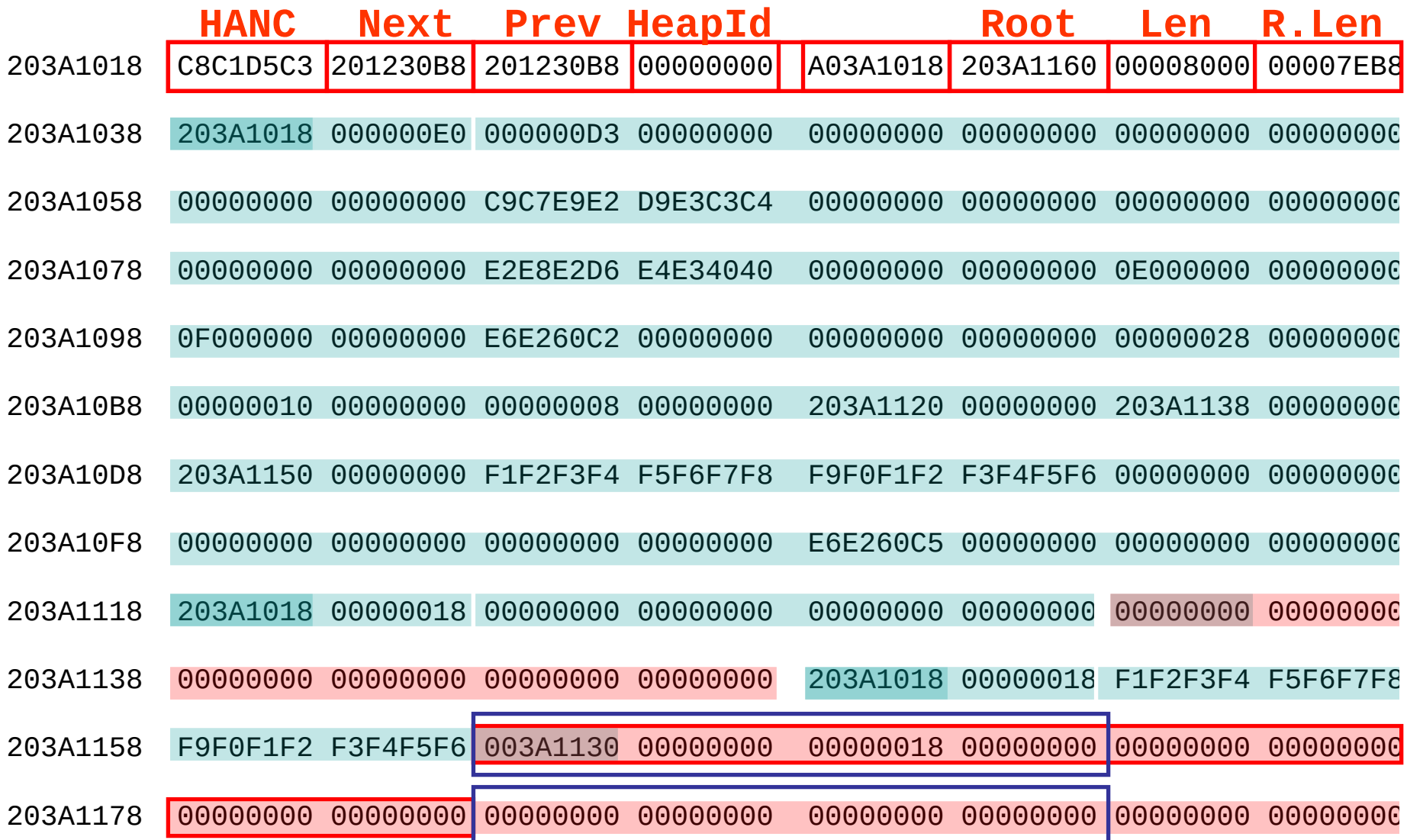
	HANC	Next	Prev	HeapId		Root	Len	R.Len
203A1018	C8C1D5C3	201230B8	201230B8	00000000	A03A1018	203A1160	00008000	00007EB8
203A1038	203A1018	000000E0	000000D3	00000000	00000000	00000000	00000000	00000000
203A1058	00000000	00000000	C9C7E9E2	D9E3C3C4	00000000	00000000	00000000	00000000
203A1078	00000000	00000000	E2E8E2D6	E4E34040	00000000	00000000	0E000000	00000000
203A1098	0F000000	00000000	E6E260C2	00000000	00000000	00000000	00000028	00000000
203A10B8	00000010	00000000	00000008	00000000	203A1120	00000000	203A1138	00000000
203A10D8	203A1150	00000000	F1F2F3F4	F5F6F7F8	F9F0F1F2	F3F4F5F6	00000000	00000000
203A10F8	00000000	00000000	00000000	00000000	E6E260C5	00000000	00000000	00000000
203A1118	203A1018	00000018	00000000	00000000	00000000	00000000	00000000	00000000
203A1138	00000000	00000000	00000000	00000000	203A1018	00000018	F1F2F3F4	F5F6F7F8
203A1158	F9F0F1F2	F3F4F5F6	003A1130	00000000	00000018	00000000	00000000	00000000
203A1178	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

Request for 32 (x'20') bytes comes in – 24 + 8 byte header

Follow root to 203A1160 – non zero pointer – but size only x'18 – no room

	HANC	Next	Prev	HeapId		Root	Len	R.Len
203A1018	C8C1D5C3	201230B8	201230B8	00000000	A03A1018	203A1160	00008000	00007EB8
203A1038	203A1018	000000E0	000000D3	00000000	00000000	00000000	00000000	00000000
203A1058	00000000	00000000	C9C7E9E2	D9E3C3C4	00000000	00000000	00000000	00000000
203A1078	00000000	00000000	E2E8E2D6	E4E34040	00000000	00000000	0E000000	00000000
203A1098	0F000000	00000000	E6E260C2	00000000	00000000	00000000	00000028	00000000
203A10B8	00000010	00000000	00000008	00000000	203A1120	00000000	203A1138	00000000
203A10D8	203A1150	00000000	F1F2F3F4	F5F6F7F8	F9F0F1F2	F3F4F5F6	00000000	00000000
203A10F8	00000000	00000000	00000000	00000000	E6E260C5	00000000	00000000	00000000
203A1118	203A1018	00000018	00000000	00000000	00000000	00000000	00000000	00000000
203A1138	00000000	00000000	00000000	00000000	203A1018	00000018	F1F2F3F4	F5F6F7F8
203A1158	F9F0F1F2	F3F4F5F6	003A1130	00000000	00000018	00000000	00000000	00000000
203A1178	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

So use the first x'20' bytes for a new allocated element.



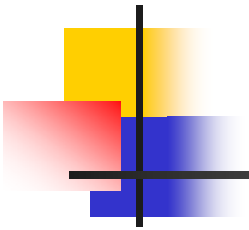
Remainder of free element must be returned to free tree.
So copy the information from existing free element.

```
                HANC     Next     Prev   HeapId             Root     Len   R.Len
203A1018  C8C1D5C3 201230B8 201230B8 00000000  A03A1018 203A1160 00008000 00007EB8
203A1038  203A1018 000000E0 000000D3 00000000  00000000 00000000 00000000 00000000
203A1058  00000000 00000000 C9C7E9E2 D9E3C3C4  00000000 00000000 00000000 00000000
203A1078  00000000 00000000 E2E8E2D6 E4E34040  00000000 00000000 0E000000 00000000
203A1098  0F000000 00000000 E6E260C2 00000000  00000000 00000000 00000028 00000000
203A10B8  00000010 00000000 00000008 00000000  203A1120 00000000 203A1138 00000000
203A10D8  203A1150 00000000 F1F2F3F4 F5F6F7F8  F9F0F1F2 F3F4F5F6 00000000 00000000
203A10F8  00000000 00000000 00000000 00000000  E6E260C5 00000000 00000000 00000000
203A1118  203A1018 00000018 00000000 00000000  00000000 00000000 00000000 00000000
203A1138  00000000 00000000 00000000 00000000  203A1018 00000018 F1F2F3F4 F5F6F7F8
203A1158  F9F0F1F2 F3F4F5F6 003A1130 00000000  00000018 00000000 00000000 00000000
203A1178  00000000 00000000 003A1130 00000000  00000018 00000000 00000000 00000000
```

Remainder of free element must be returned to free tree.
So copy the information from existing free element.

	HANC	Next	Prev	HeapId		Root	Len	R.Len
203A1018	C8C1D5C3	201230B8	201230B8	00000000	A03A1018	203A1160	00008000	00007EB8
203A1038	203A1018	000000E0	000000D3	00000000	00000000	00000000	00000000	00000000
203A1058	00000000	00000000	C9C7E9E2	D9E3C3C4	00000000	00000000	00000000	00000000
203A1078	00000000	00000000	E2E8E2D6	E4E34040	00000000	00000000	0E000000	00000000
203A1098	0F000000	00000000	E6E260C2	00000000	00000000	00000000	00000028	00000000
203A10B8	00000010	00000000	00000008	00000000	203A1120	00000000	203A1138	00000000
203A10D8	203A1150	00000000	F1F2F3F4	F5F6F7F8	F9F0F1F2	F3F4F5F6	00000000	00000000
203A10F8	00000000	00000000	00000000	00000000	E6E260C5	00000000	00000000	00000000
203A1118	203A1018	00000018	00000000	00000000	00000000	00000000	00000000	00000000
203A1138	00000000	00000000	00000000	00000000	203A1018	00000018	F1F2F3F4	F5F6F7F8
203A1158	F9F0F1F2	F3F4F5F6	201A1018	00000020	00000018	00000000	00000000	00000000
203A1178	00000000	00000000	003A1130	00000000	00000018	00000000	00000000	00000000

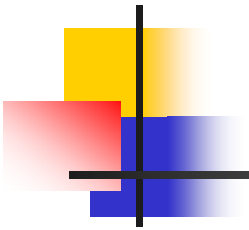
Now create the allocated element
Problem 2! Damage has moved



...Heaps are fun!

- SYSMDUMP – IPCS
  - Use VERBX LEDATA ‘HEAP’
    - First ENSM – Enclave Storage Management CB

```
Language Environment Product 04 V01 R06.00

Heap Storage Control Blocks

     ENSM:  201230A0
   +000000  EYE_CATCHER:ENSM  ST_HEAP_ALLOC_FLAG:00000000
   +000008  ST_HEAP_ALLOC_VAL:00000000    ST_HEAP_FREE_FLAG:00000000
   +000010  ST_HEAP_FREE_VAL:00000000     REPORT_STORAGE:00000000
   +000018  UHEAP:C8D7C3C2 203A1018 203A1018 00008000 00008000 00002000 00001000 00000000 00
   +000048  AHEAP:C8D7C3C2 2037D000 2037D000 00004000 00002000 00002000 00001000 00000000 00
   +000078  BHEAP:C8D7C3C2 20123118 20123118 00002000 00001000 00002000 00001000 80000000 00
   +0000A8  ENSM_ADDL_HEAPS:F0F00000
```



...Heaps are fun!

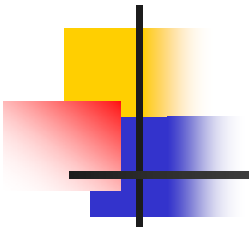
- SYSMDUMP – IPCS
  - Then HPCB and header for user heap

```
User Heap Control Blocks

     HPCB:  201230B8
   +000000  EYE_CATCHER:HPCB  FIRST:203A1018    LAST:203A1018

     HANC:  203A1018
   +000000  EYE_CATCHER:HANC  NEXT:201230B8     PREV:201230B8
   +00000C  HEAPID:00000000   SEG_ADDR:A03A1018       ROOT_ADDR:203A1160
   +000018  SEG_LEN:00008000        ROOT_LEN:00007EB8

   This is the last heap segment in the current heap.
```

...Heaps are fun!

- SYSMDUMP – IPCS
  - Next is the free tree information
    - NOTE: *ERROR* indicates an error was found

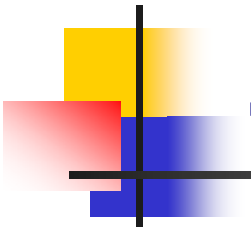
```
Free Storage Tree for Heap Segment 203A1018

          Node       Node      Parent      Left       Right      Left      Right
 Depth  Address    Length      Node       Node       Node      Length    Length
     0  203A1160  00007EB8  00000000  003A1130  00000000  00000018  00000000
*ERROR*  The left node address does not fall within the current heap segment
```



...Heaps are fun!

- SYSMDUMP – IPCS
  - Next map all elements in heap segment

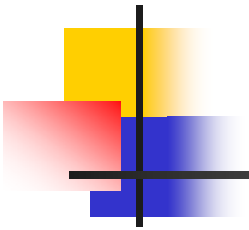
```
 Map of Heap Segment 203A1018
 To display entire segment: IP LIST 203A1018 LEN(X'00008000') ASID(X'0181')

 203A1038: Allocated storage element, length=000000E0. To display: IP LIST
203A1038 LEN(X'000000E0') ASID(X'0181')
 203A1040: 000000D3 00000000 00000000 00000000  00000000 00000000 00000000
00000000 |...L............................|

 203A1118: Allocated storage element, length=00000018. To display: IP LIST
203A1118 LEN(X'00000018') ASID(X'0181')
 203A1120: 00000000 00000000 00000000 00000000
|................                |
```



...Heaps are fun!

- SYSMDUMP – IPCS
  - Showing error as we go...

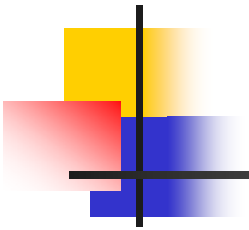
```
 203A1130: Allocated storage element, length=00000000. To display: IP LIST
203A1130 LEN(X'00000008') ASID(X'0181')
 203A1138: 00000000 00000000
|........                          |
*ERROR*  The heap segment address in the allocated storage element header is not
valid
WARNING  This storage element may be a free storage node not found during free
storage tree validation
*ERROR*  The length of this storage element is zero
WARNING  Attempting to identify a resume location after encountering a storage
element validation error
```



...Heaps are fun!

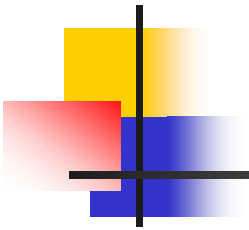
- SYSMDUMP – IPCS
  - Finally Summary information
    - Amount of allocated and free storage
    - Number of elements
    - Identifies if there were errors in this heap segment
    - Problem 1 solved!

```
Summary of analysis for Heap Segment 203A1018:
  Amounts of identified storage:  Free:00007EB8  Allocated:00000110  Total:00007FC8
  Number of identified areas   :  Free:       1  Allocated:       4  Total:       5
  00000018 bytes of storage were not accounted for.
  Errors were found while processing this heap segment.
  This is the last heap segment in the current heap.
```



...Heaps are fun!

- HEAPCHK runtime option
  - Runtime debug tool to help diagnose a heap damage problem
  - In normal cases any heap damage may not be noticed until significant time has passed (problem 2)
  - The HEAPCHK runtime option forces all the heap segments to be validated on a regular basis.
  - Gets a dump closer to the real causer.
  - Generates a U4042 ABEND
    - Use System dump, if needed, to debug.



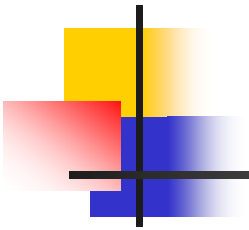
...Heaps are fun!

- HEAPCHK runtime option ...
  - HEAPCHK(ON|OFF,freq,delay,depth,pooldepth)
    - ON - turns HEAPCHK on (performance dog)
    - OFF - normal processing
  - freq
    - Defaults to 1, indicates every call to a heap routine (get or free) validates the heap
    - Other values for less frequent checks
  - delay
    - Allows some number of calls to occur prior to 'freq' being used.
  - depth
    - Depth of traceback for storage leak
  - pooldepth
    - Depth of heappools trace

...Heaps are fun!

- Sample program job log (with HEAPCHK(ON))

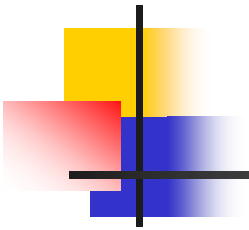
```
17.12.44 JOB22865 ---- WEDNESDAY, 16 FEB 2005 ----
17.12.45 JOB22865  $HASP373 JMONTIGO STARTED - WLM INIT  - SRVCLASS WLMLONG  - SYS AQFT
17.12.45 JOB22865  IEF403I JMONTIGO - STARTED - TIME=17.12.45
17.12.48 JOB22865  IEA995I SYMPTOM DUMP OUTPUT  937
   937               USER COMPLETION CODE=4042 REASON CODE=00000000
   937              TIME=17.12.45  SEQ=11443  CPU=0000  ASID=018E
   937              PSW AT TIME OF ERROR  078D1000   A01AB242  ILC 2  INTC 0D
   937                ACTIVE LOAD MODULE          ADDRESS=201256C0  OFFSET=00085B82
   937                NAME=CEEPLPKA
   937                DATA AT PSW  201AB23C - 00181610  0A0D47F0  B10A1811
   937                AR/GR 0: 80AB5B3E/84000000   1: 00000000/84000FCA
   937                      2: 00000000/00000000   3: 00000000/00000001
  ...
   937                      E: 00000000/00000000   F: 00000000/00000000
   937               END OF SYMPTOM DUMP
17.12.48 JOB22865  IEA993I SYSMDUMP TAKEN TO POSIX.JMONTI.SHARE.HEAP.SYSMDUMP
17.12.48 JOB22865  IEF450I JMONTIGO GO - ABEND=S000 U4042 REASON=00000000  939
   939                     TIME=17.12.48
```



...Heaps are fun!

- Sample program output (with HEAPCHK(ON))
  - In this case CEE37xx messages a very meaningful
  - The damage has not yet moved
  - Full debug may still need to be done with SYSMDUMP and IPCS
  - Use a storage alteration (SA) SLIP if needed (non-CICS)
  - Problem 2 Solved! ?

```
STARTING SHAREHP...
CEE3701W Heap damage found by HEAPCHK run-time option.
CEE3707I Left  pointer is bad in the free tree at 203A1160 in the heap
segment beginning at 203A1018.
  203A1140: 00000000 00000000 203A1018 00000018  F1F2F3F4 F5F6F7F8 F9F0F1F2
F3F4F5F6  |................1234567890123456|
  203A1160: 003A1130 00000000 00000018 00000000  00000000 00000000 00000000
00000000  |................................|
CEE3702S Program terminating due to heap damage.
```



...Heaps are fun!

- Summary for heaps
  - Heap used for dynamic storage
  - CEEDUMPs contain information on heap errors but they are difficult to find
  - SYSTEM DUMPs using LEDATA 'HEAP' make debug much simpler
  - Use HEAPCHK runtime option to debug
    - Big performance hit

Sources of Additional Info

- All Language Environment documentation available on DISK 1 of zOS CD collection, the DVD collection and on the Language Environment website
  - Language Environment Debug Guide
  - Language Environment Runtime Messages
  - Language Environment Programming Reference
  - Language Environment Programming Guide
  - Language Environment Customization
  - Language Environment Migration Guide
  - Language Environment Writing ILC Applications
- Language Environment Web site
  - http://www-03.ibm.com/systems/z/os/zos/features/lang_environment/