How System SSL, Communication Server, RACF, ICSF, CLiC and PKI Services play together in the crypto area

Wai Choi, CISSP
IBM/System SSL Design and Development

Session ON

GSE UK Conference
2024 Charities

Every year, speakers and delegates at the GSE UK Conference donate to the charities we help. This year is no exception and we aim to collect donations for two worthy causes:
Air Ambulance UK and Muscular Dystrophy UK.

Please consider showing your appreciation by kindly donating a small sum to our charities this year!!

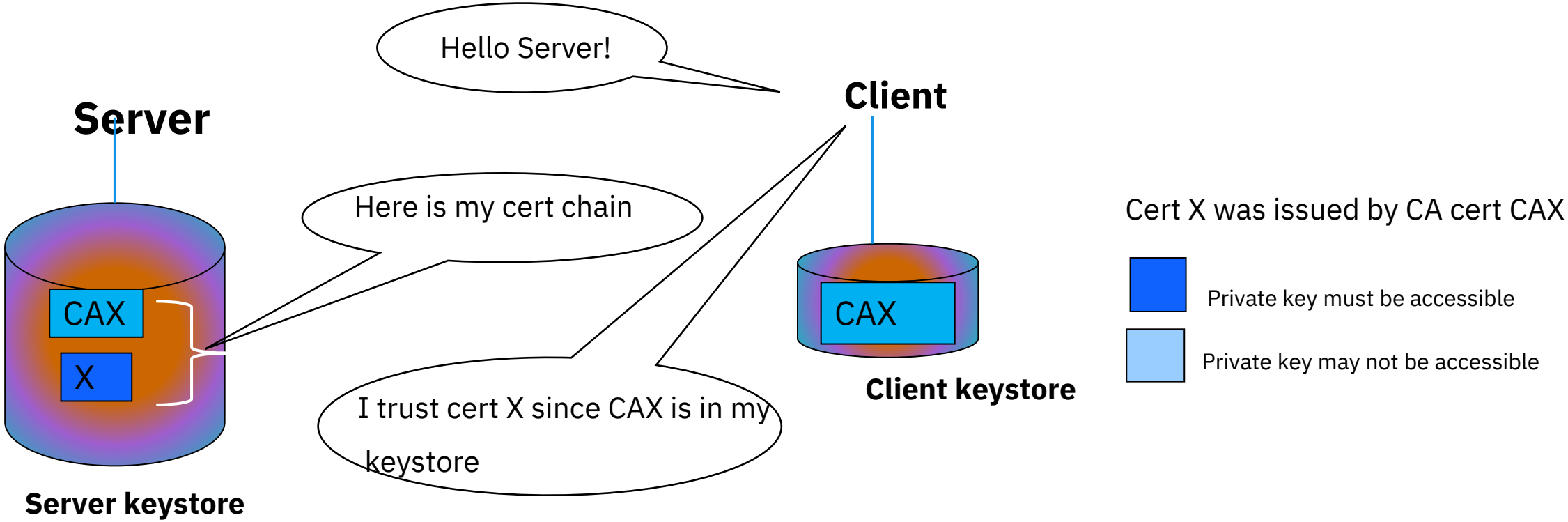
AIR AMBULANCES UK
SUPPORTING AIR AMBULANCE CHARITIES

MUSCULAR DYSTROPHY UK | OUR MUSCLES MATTER

Agenda

- A typical workflow to set up for securing communication through TLS from the keystore perspective
- How this workflow involves multiple components on z/OS
  - RACF, ICSF and CLiC, System SSL, Communication Server AT-TLS, JSSE and PKI Services
- Highlight the importance in each component

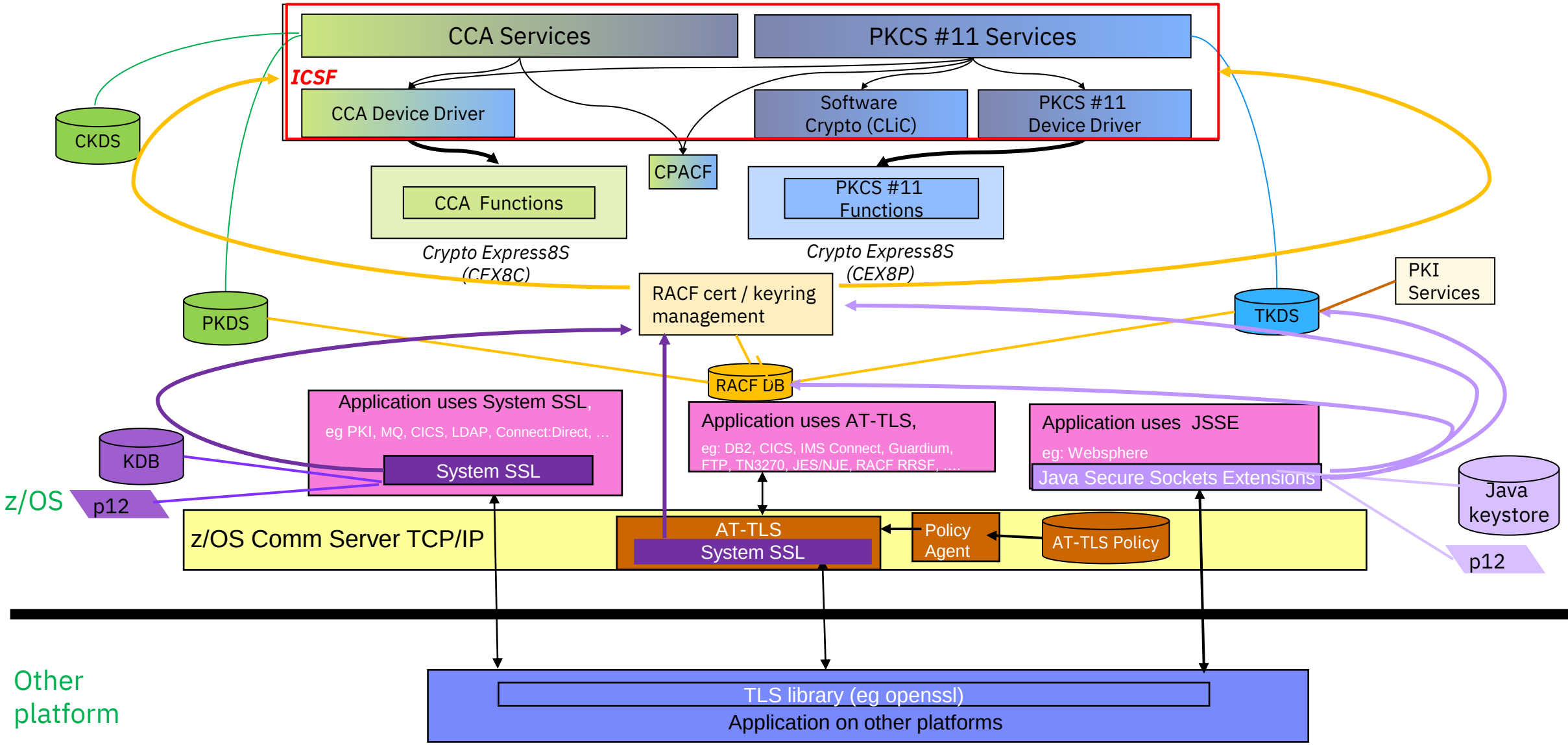
Keystores are the basic set up needed for TLS handshake



Different terms are used as keystores:

- Keyring (RACF), key database (System SSL), keystore+truststore (Java) PKCS#12 file (System SSL, Java), Token (ICSF)…
- The role is the same – a collection of certificate(s) and sometimes private key

Let's take a group picture

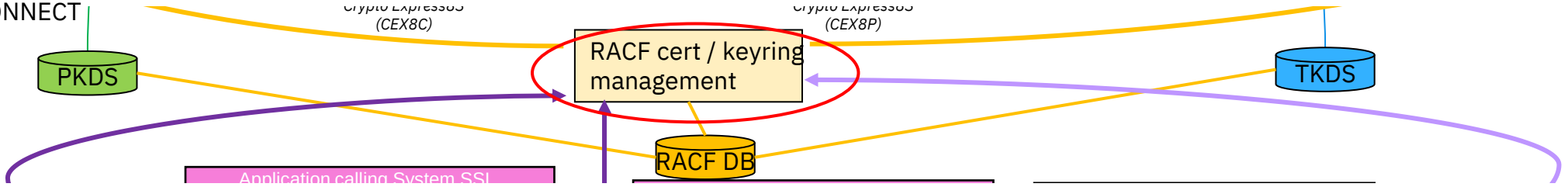


RACF

The big picture – 1st stop – RACF keyring

RACF RACDCERT command – populating the key ring

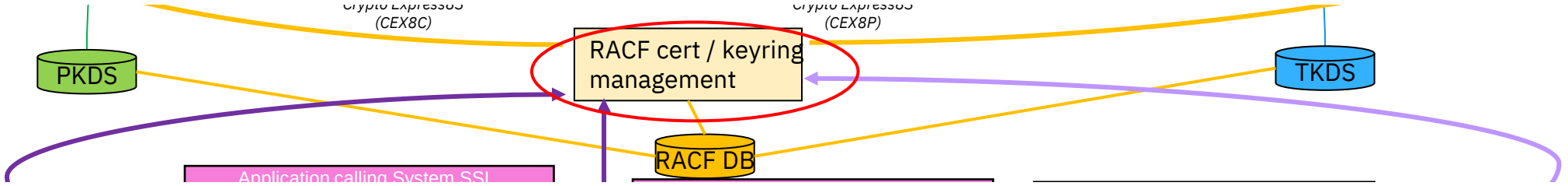
- There are 26 functions operating on certificate and the related objects including certificate request, keyring, token, mapping filter
- The most used functions are
  - GENCERT
  - GENREQ
  - ADD
  - ADDRING
  - CONNECT



- These functions get the certificate and the associated objects into the RACF DB
- The private key associated with the certificate can also be stored in the RACF DB, or in ICSF's PKDS or TKDS

Populate the keyring

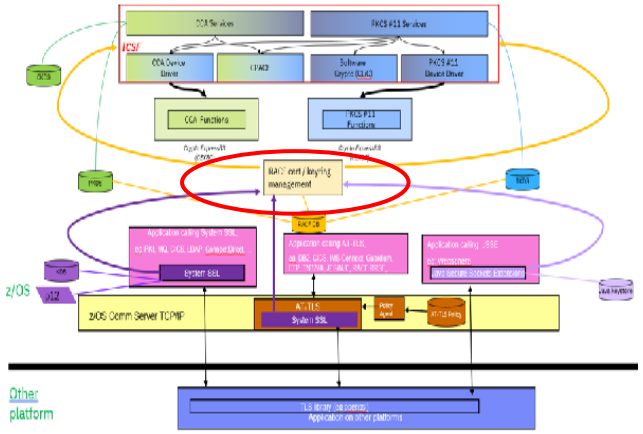
- Series of RACDCERT functions is the first preparation for TLS communication using RACF keyring
  - RACDCERT ADDRING(...)
  - RACDCERT GENCERT /ADD (...) (multiple times)
  - RACDCERT CONNECT(<cert>... <keyring>) (multiple times)



- RACDCERT LISTRING(<keyring>)

```
Ring:
     >myRing<
Certificate Label Name                 Cert Owner     USAGE      DEFAULT
------------------------------------   ------------   --------   -------
myCA                                   CERTAUTH       CERTAUTH     NO
myIntCA                                CERTAUTH       CERTAUTH     NO
myServerCert                           ID(ftpd)       PERSONAL     YES
```

RACF certificate/key retrieval

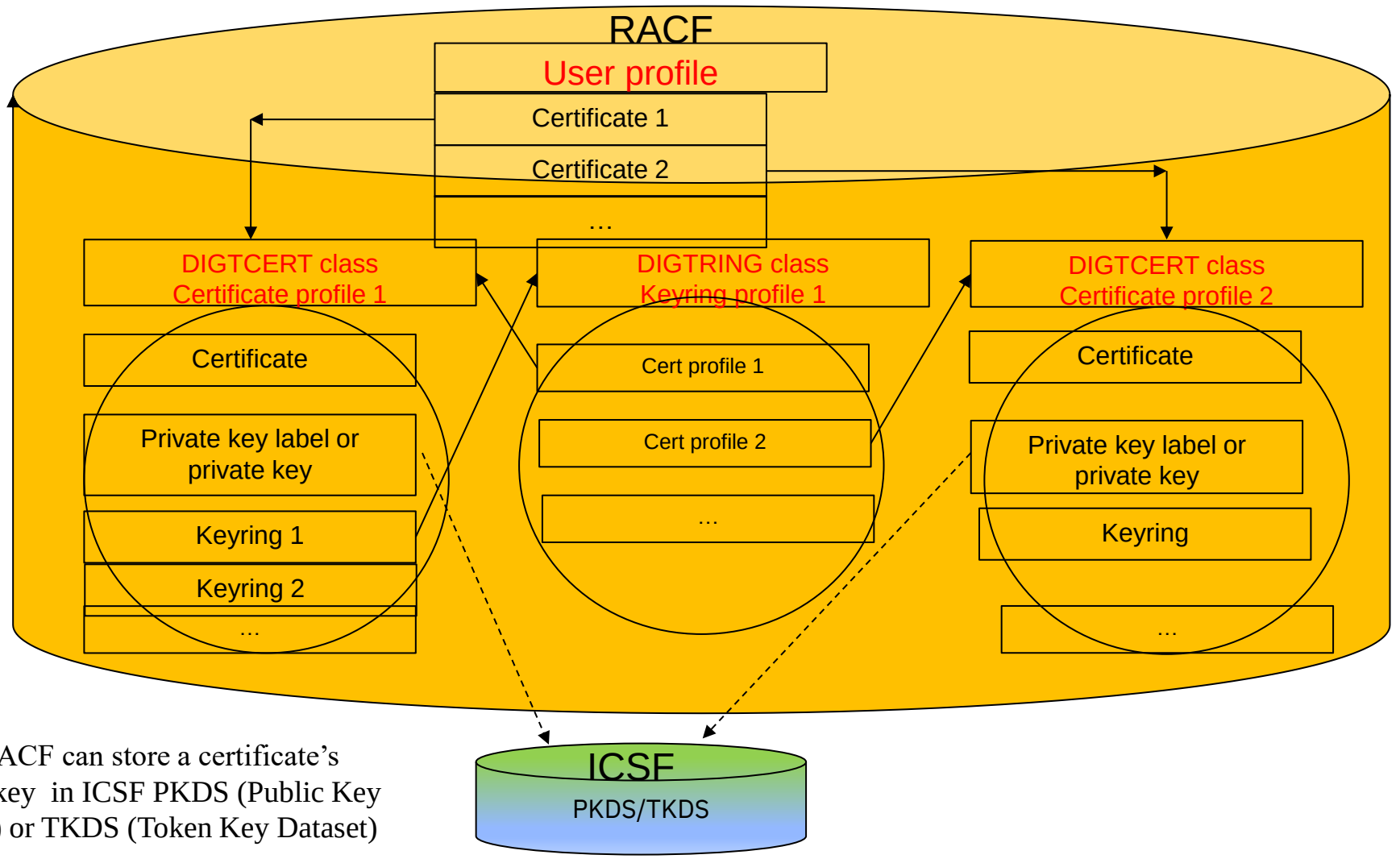


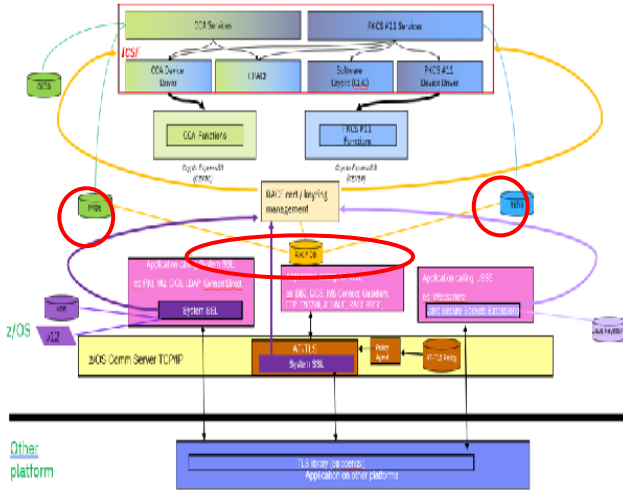
RACF Callable Services – R_DATALIB, initACEE

- **R_DATALIB** functions provide majority programming interface on keyring
  - The dataGet functions is used to retrieve certificate and key from the RACF DB (vs RACDCERT EXPORT command)
  - Retrieval of key is controlled by the input attribute, the default flag or the label
  - The only interface for all z/OS applications that need to do the retrieval from RACF DB for **TLS communication**
  - There are other functions that can also add / delete certificates too

- **InitACEE's** register/deregister functions are used to add/delete a certificate; its query function is to query whether the RACF DB contains the certificate
  - It is usually called after the handshake process in the client authentication process. The certificate presented by the client would be used to build the Accessor Environment Element (ACEE) to identify and authorize the client to perform certain action

Certificates and keyrings in RACF DB

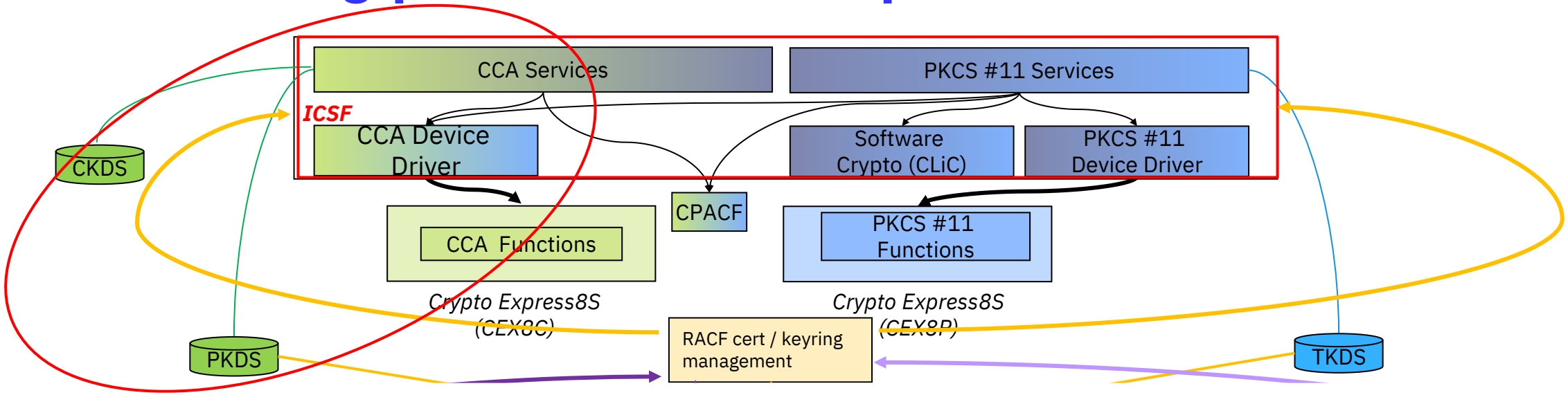




Note: RACF can store a certificate's private key in ICSF PKDS (Public Key Dataset) or TKDS (Token Key Dataset)

RACF, ICSF and PKI Services

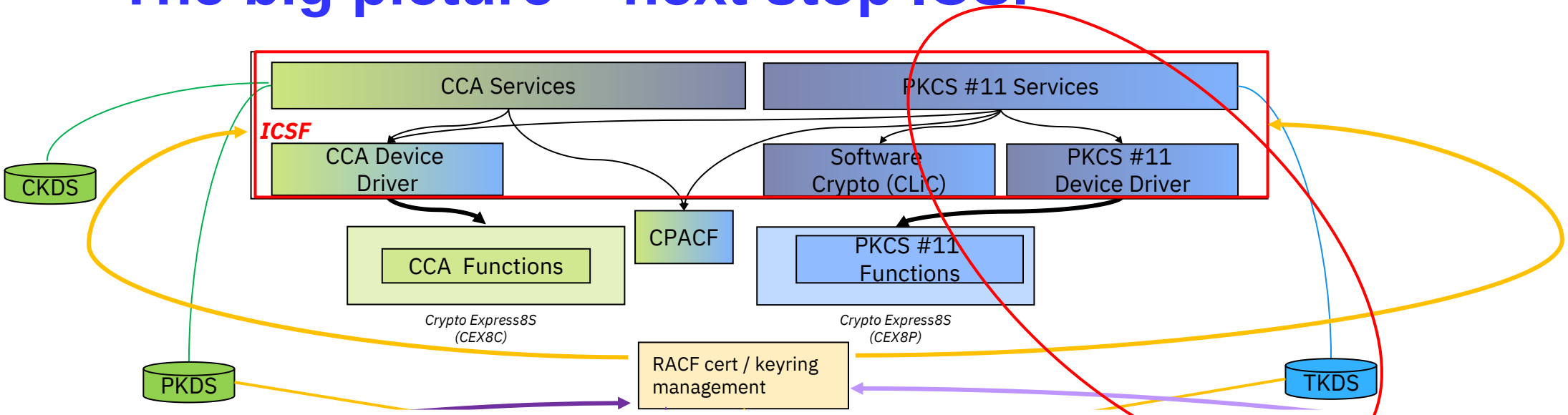
The big picture – next stop ICSF



Integrated Cryptographic Service Facility (ICSF) is the driver of almost all the cryptographic operations. It provides two cryptographic services

- Common Cryptographic Architecture (CCA) Services
  - Include CCA hardware and CP Assist for Cryptographic Functions (CPACF)
  - CCA manages two key datasets to store key materials
    - Cryptographic Key Data Set (CKDS) – for symmetric keys
    - Public Key Data Set (PKDS) – for asymmetric keys
  - All keys are secure keys protected under a master key

The big picture – next stop ICSF

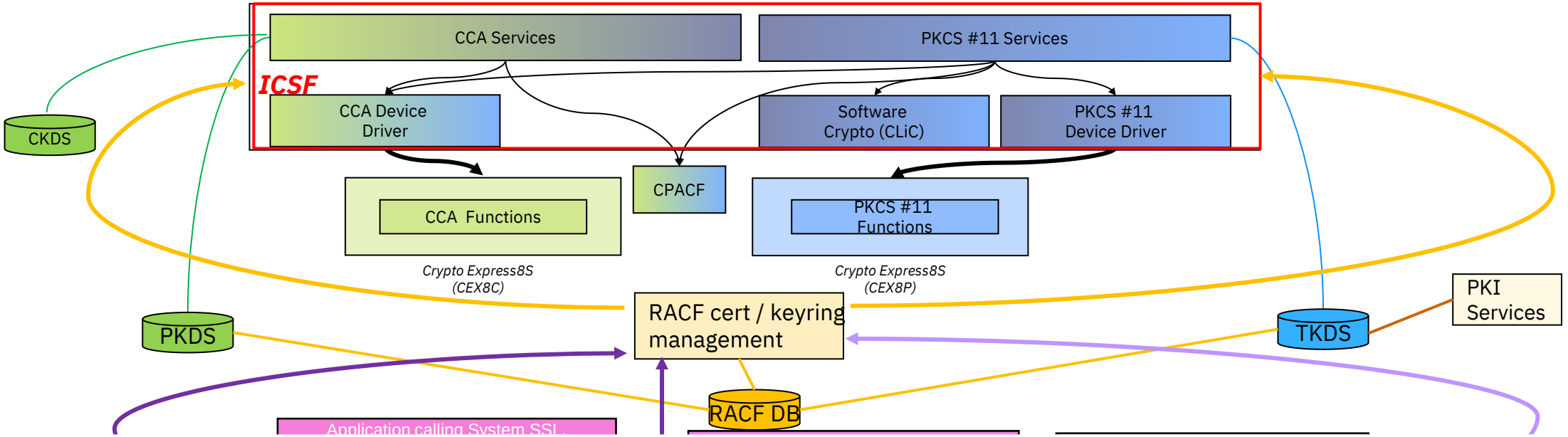


- Public Key Cryptography Standard #11 (PKCS#11) Services
  - Include PKCS#11 hardware and PKCS#11 software functions (through the Crypto Lite library in C (CLiC)
  - Manage one key dataset to store key materials
    - Token Key Dataset (TKDS) – for asymmetric keys
    - Objects managed by CLiC are clear
    - That's why TKDS can contain both secure and clear objects

Which KDS is/are related to certificates?

*You may use PKCS#11 services without the cryptographic coprocessors
*CPACF has a limited set of functions for the handshake process to improve the performance without hardware

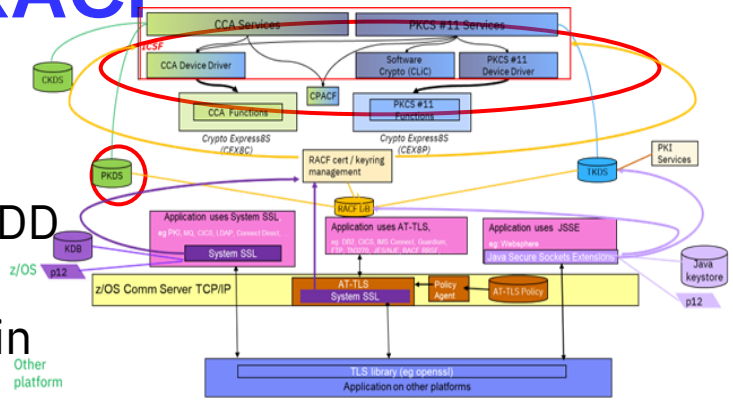
PKDS and TKDS managed by ICSF, RACF and PKI Services



- PKDS and TKDS are managed by ICSF
- But RACF can create PKDS/TKDS key that is associated with the certificate in RACF through the RACDCERT command
- PKI Services can create certificate with its associated key in TKDS too
- Important advice – don't manipulate the private key object in the KDS created through RACDCERT or PKI Services from ICSF!!!

ICSF PKDS secure key management through RACF

- Two key datasets are used by ICSF to store the asymmetric key materials
  - Public Key Dataset (PKDS)
    - When the keyword PKDS is specified in RACDCERT GENCERT or ADD command, the private key associated with certificate would be stored in the ICSF PKDS, referenced by a private key label stored in RACF DB
    - Eg, racdcert id(xxx) gencert … withlabel('pkdscert') rsa(pkds(*))
      (Note: pkds(*) indicates the certificate label would be used as key label too)

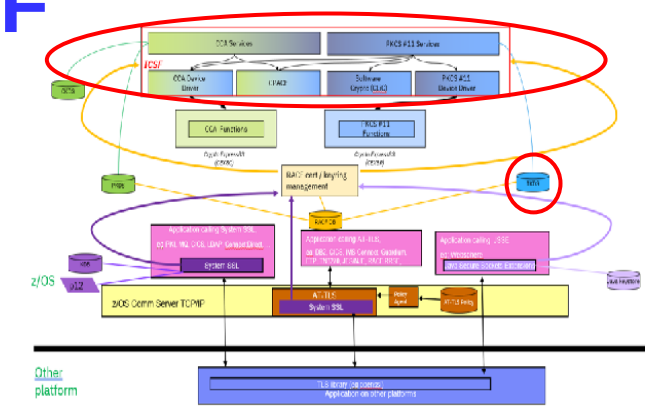



```
RACDCERT id(xxx)
list(label('pkdscert')):

Label: pkdscert
…
Key Type: RSA
Key Size: 2048
Private Key: YES
PKDS Label: PKDSCERT
Ring Associations:
*** No rings associated ***
```

```
ICSF Utility panel – option 6 – PKDS KEYS
ICSF - PKDS Key Attributes

Label: pkdscert
…
Key Usage:     KEYM SIGN NO-XLATE FR-NONE

Sections:      PRIVATE PUBLIC                Token Format:    CRT

Public Exponent:
   00010001
 Modulus:
   D65B3DF6195205AC34F669AA56D8CF4…
```

ICSF TKDS secure key management through RACF

- Token Key Dataset (TKDS) – use to store the secure private key
  - When the keyword TOKEN is specified in RACDCERT GENCERT command, the private key associated with certificate would be stored in the ICSF TKDS, referenced by a private key label stored in RACF DB
  - Eg, racdcert id(xxx) gencert … withlabel('tkdscert') rsa(token(mytoken))
    (Note: the token must exist , may be created through RACDCERT ADDTOKEN)



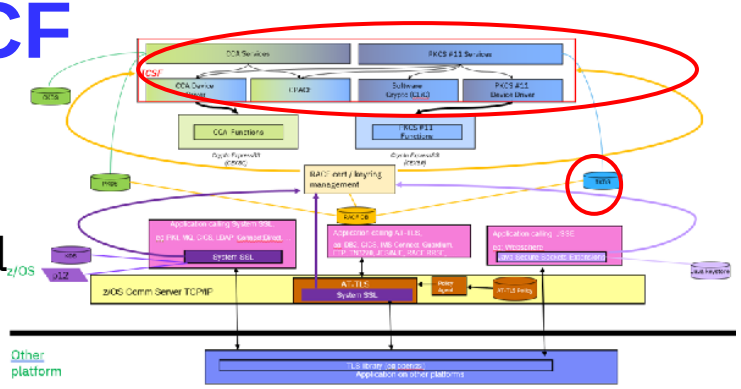

```
RACDCERT id(xxx)
list(label('tkdscert')):

Label: pkdscert
…
Key Type: RSA
Key Size: 2048
Private Key: YES
TKDS Token: MYTOKEN
TKDS ID: 00000004y
Ring Associations:
*** No rings associated ***
```

```
ICSF Utility panel - option 7 - PKCS11 TOKEN
ICSF Token Management - Token Details

Token name: MYTOKEN
…
_ Object 00000004Y PRIVATE KEY   PRIVATE: TRUE        MODIFIABLE: TRUE
                                 EXTRACTABLE: NEVER  SENSITIVE:ALWAYS
    LABEL:          IRRRSAPRIVKEY
...
```

Note: The object ends with 'Y' indicates secure object, 'T' indicates clear one. TKDS can contain both secure and clear objects

ICSF TKDS clear key management through RACF

- Token Key Dataset (TKDS) – use to store the clear private key
  - If you use the BIND command to bind a certificate with the private key in RACF to a TOKEN, copies of the certificate and private key will be stored in TKDS too, as clear objects
  - Eg, racdcert id(xxx) **gencert** … withlabel('tkdscert2')
    racdcert id(xxx) **bind**(token(mytoken) label('tkdscert2'))
    (Note: the token must exist , may be created through RACDCERT ADDTOKEN)
  - RACDCERT list won't show the TKDS information, but ICSF would show the objects

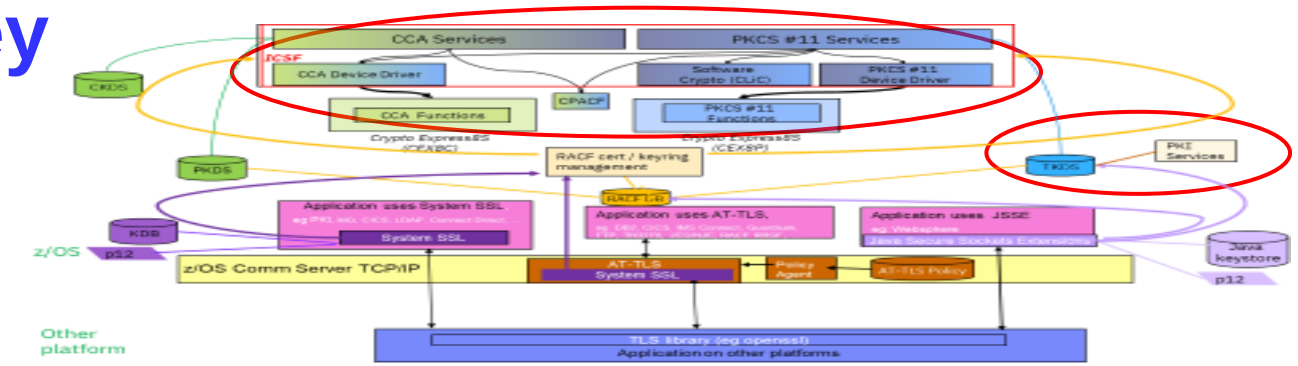



```
RACDCERT id(xxx)
list(label('tkdscert2):

Label: tkdscert2
…
Key Type: RSA
Key Size: 2048
Private Key: YES
Ring Associations:
*** No rings associated ***
```

```
ICSF Token Management – Token Details

Token name: MYTOKEN
…
 _ Object 00000007T PRIVATE KEY   PRIVATE: TRUE       MODIFIABLE: TRUE
                                  EXTRACTABLE: TRUE  SENSITIVE: FALSE
    LABEL:          tkdscert2
...
ICSF Utility panel – option 7 – PKCS11 TOKEN
```

Note: The object ends with 'Y' indicates secure object, 'T' indicates clear one. TKDS can contain both secure and clear objects

ICSF TKDS secure and clear key management through PKI Services



Token Key Dataset (TKDS) – PKI Services uses to store the secure or clear private key and certificate related objects

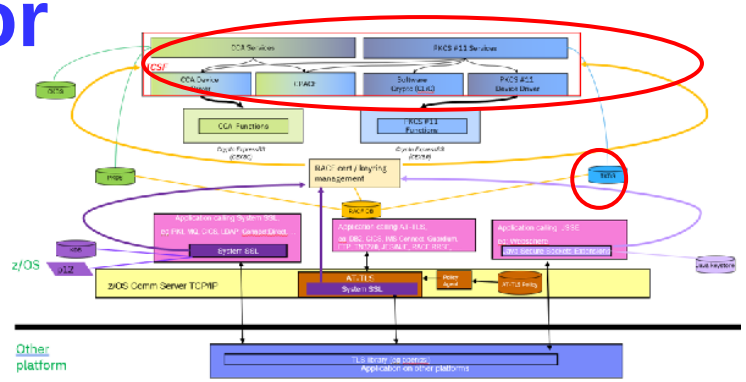
- Depends how you configure PKI to create secure key or clear key for the certificate in pkiserv.conf

  `TokenName=PKISRVD.PKIToken`

  `SecureKey=T (default is F, in case there is no EP11 processor)`

  Note: RACDCERT GENCERT only support secure key in TKDS
- Objects created in the token:
  - Public Key Object
  - Private Key Object
  - Certificate Object
  - PKCS12 package

ICSF TKDS token can be used as a keystore for handshake



- Token Key Dataset (TKDS) – use to store certificate chain as in a keystore
  - You can use the BIND command to bind certificates without the private key, copies of the certificates stored in TKDS can be used to build up the chain for handshake
  - Eg, racdcert **certauth** bind(token(mytoken) label('myCA'))
    racdcert **certauth** bind(token(mytoken) label('myIntCA'))
    (Note: bind to the same token which contains the certificate and the private key bound previously)
  - After all the certificates and private key set up in the token, you may use it as input to System SSL to perform handshake, just like a keyring

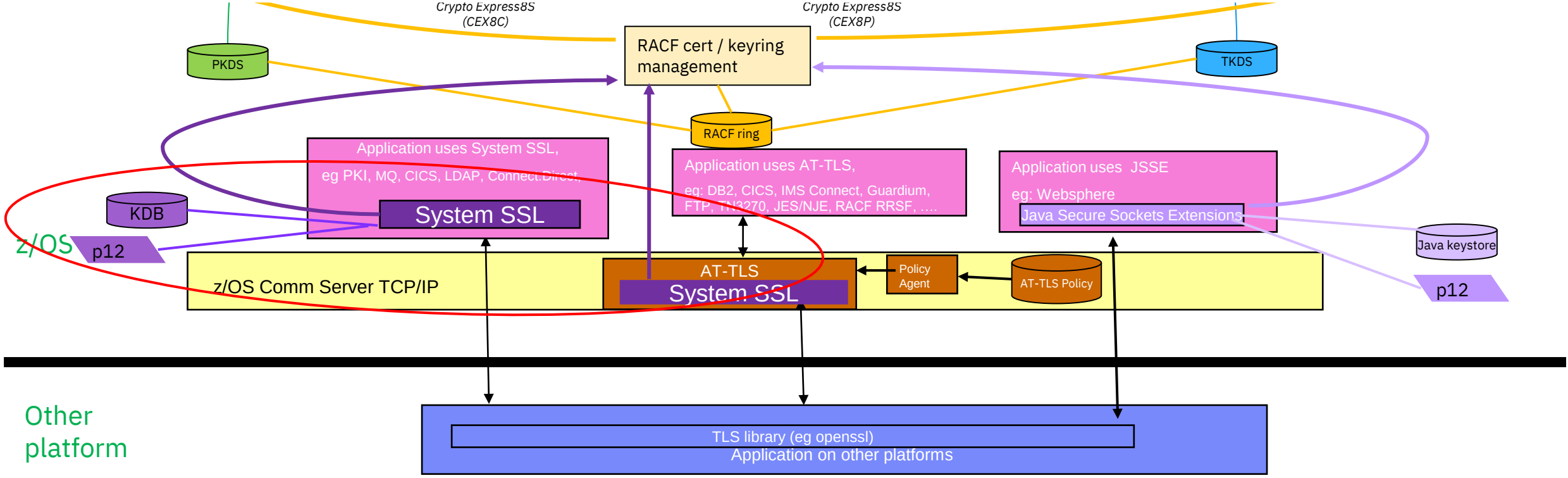
E.g. RACDCERT LISTTOKEN(mytoken)

```
Seq Num     Attributes                                Labels
--------    ------------------------------------      -------------
0000000B Default: NO     Priv Key: NONE    TKDS: myCA
         Usage: CERTAUTH Pub Key: YES     RACF: myCA
         Owner: CERTAUTH
0000000D Default: NO     Priv Key: NONE    TKDS: myIntCA
         Usage: CERTAUTH Pub Key: YES     RACF: myIntCA
         Owner: CERTAUTH
0000000F Default: YES    Priv Key: CLEAR TKDS: tkdscert2
         Usage: PERSONAL Pub Key: YES    RACF: tkdscert2
         Owner: ID(xxx)
```

System SSL

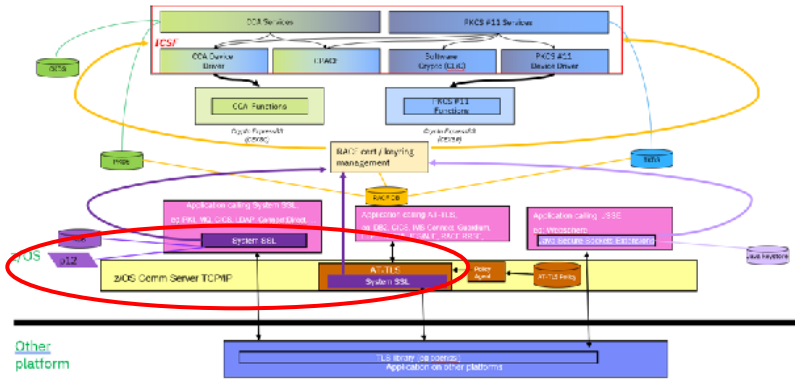
The big picture – next stop System SSL

- System SSL provides APIs for the Secure Socket Layer (SSL)/ Transport Layer Security(TLS) protocol
- The protocol requires each of the communicating parties to have a certificate/key store
- System SSL supports the following certificate/key stores
  - its native gskkyman KDB
  - SAF key ring.
  - PKCS#11 token
  - PKCS#12 file
  - GSKIT CMS V4 database (seldomly used)



Different keystores input to System SSL

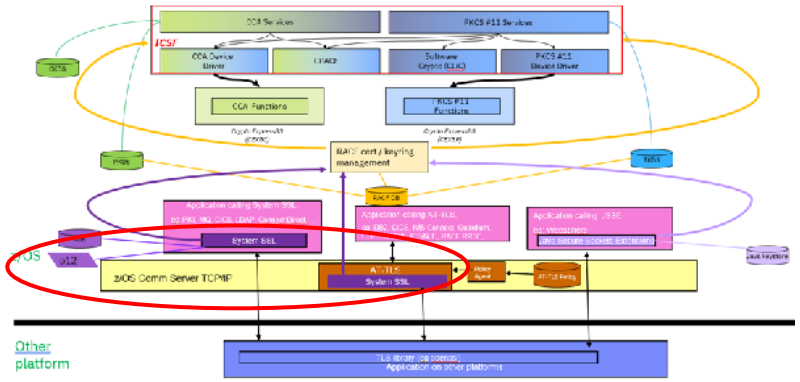
- The support of gskkyman KDB and PKCS#12 is through System SSL's own implementation. They are both files on zFS.
  - Use <KDB name> and <KDB password or a stash file> for gskkyman kdb (usually it uses a suffix .kdb, but not required)
  - Use <PKCS#12 file> and <PKCS#12 password>
  - System SSL can differentiate between these two formats not judging from the file extension – myfile.p12 can be a KDB, myp12.kdb can be a PKCS#12 file



- The support of Keyring and PKCS#11 are based on the RACF R_datalib dataGet functions
  - Use **<ring owner id>/<ring name>** if keystore is a keyring
  - Use ***TOKEN*/<token name>** if keystore is a token in TKDS

System SSL gskkyman

- Other than APIs, System SSL also provides a gskkyman utility through menu driven and command line interfaces to add, delete, create … certificate and key in the gskkyman kdb and PKCS#11 token




```
       Database Menu

   1  - Create new database
   1b - Create new empty database
   2  - Open database
   3  - Change database password
   4  - Change database record length
   5  - Delete database
   6  - Create key parameter file
   7  - Display certificate file (Binary or Base64 ASN.1 DER)

  11  - Create new token
  12  - Delete token
  13  - Manage token
  14  - Manage token from list of tokens

   0  - Exit program

Enter option number:
```

Command examples:

Import into the gskkyman kdb a certificate and its associated private key from a PKCS#12 file

gskkyman –i –k <kdb name> -kpw <kdb password> –l <label for the input cert> -p <pkcs#12 file name> -p12pw <p12 password>

Display the content of a file containing a certificate

gskkyman –dcv <file name>

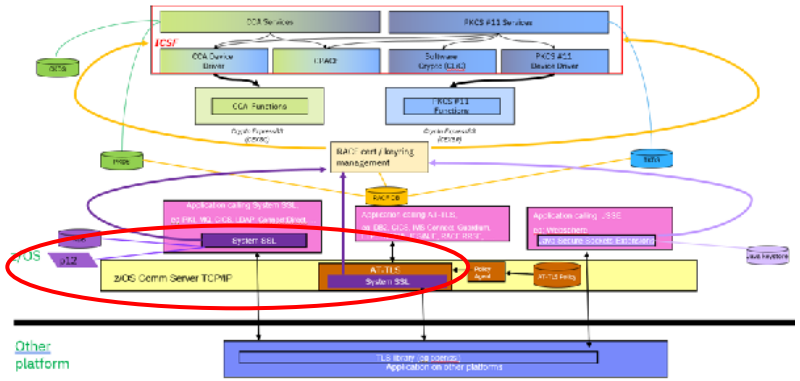
Important advice – don’t manipulate the private key object created through gskkyman from ICSF!!!

System SSL environment variables

- The basic set of information needed for handshake
  - TLS version
  - Cipher suites
  - Name of the keystore

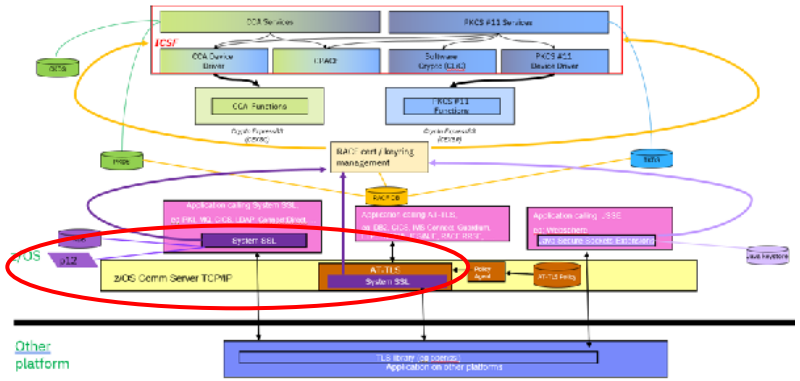
- The first set of System SSL function calls, gsk_environment_open() picks up the environment variables for the handshake, like
  - GSK_PROTOCOL_SSLV/TLSV<version #>
  - GSK_V3_CIPHERS
  - GSK_KEYRING_FILE
  - ....
- These environment variable values can be overwritten by the corresponding input parameters during the API call, for example
  - Statement CEEPRMxx: CEEDOPT(..., ENVAR(GSK_PROTOCOL_**TLS1_3=OFF**)...
  - Application: gsk_attribute_set_enum(.... GSK_PROTOCOL_TLSV1_3, GSK_PROTOCOL_**TLSV1_3_ON**)

  Effective protocol is TLSv1.3



System SSL exploiters

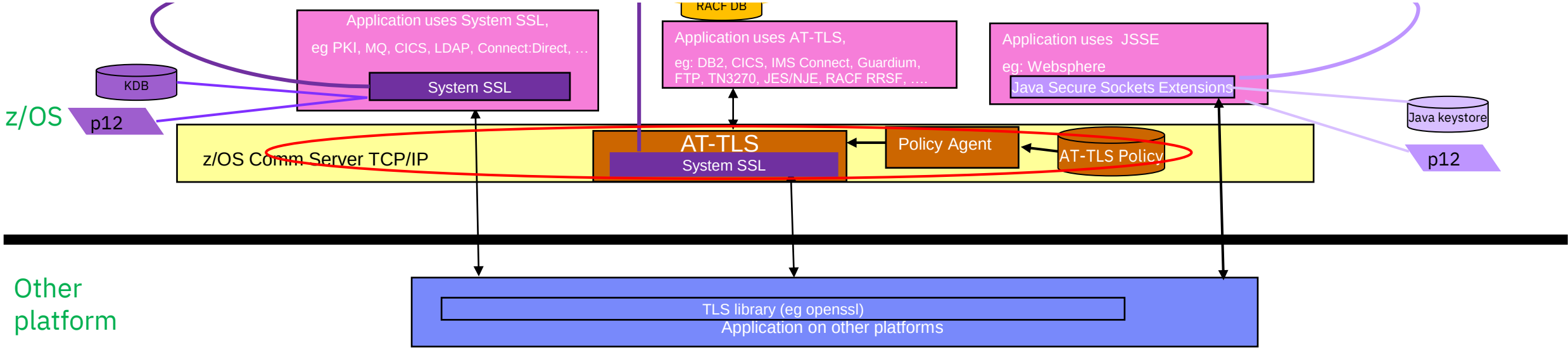
- Some IBM exploiters
  - HTTP Server
  - MQ
  - LDAP
  - PKI Services
  - Connect:Direct



Communication Server
AT-TLS

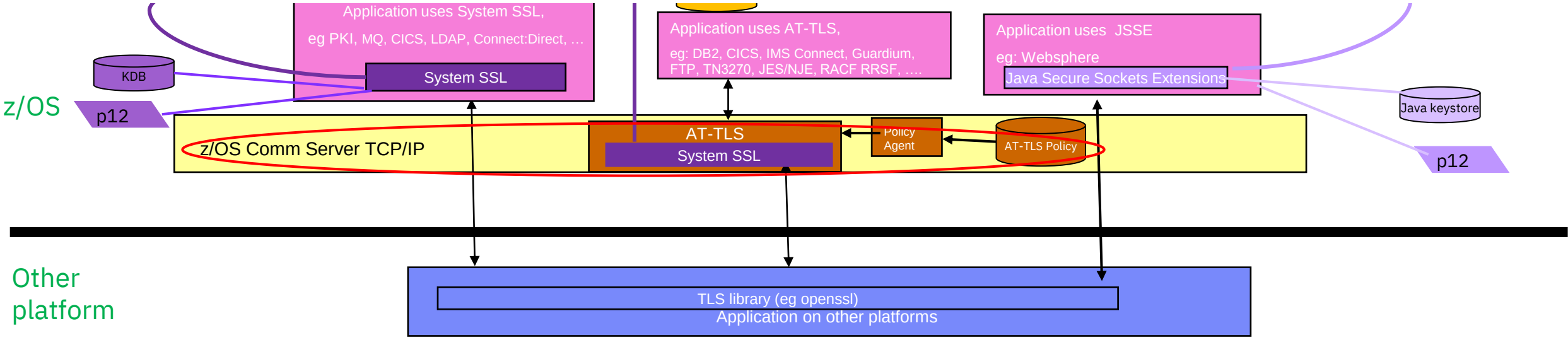
The big picture – next stop AT-TLS

- Communication Server **Application Transparent TLS** (AT-TLS) is the internal SSL application that z/OS provides for the other z/OS components and customer applications
- It makes System SSL calls on behalf of the application. It handles handshake based on the policy rules in the **policy file** read by the **policy agent**
- AT-TLS creates and manages the LE environments under which System SSL executes. These environments are organized as TTLSGroup objects in the policy file.
- Since System SSL functions are executed under a certain TTLSGroup, most of its environment variables do not take effect under AT-TLS



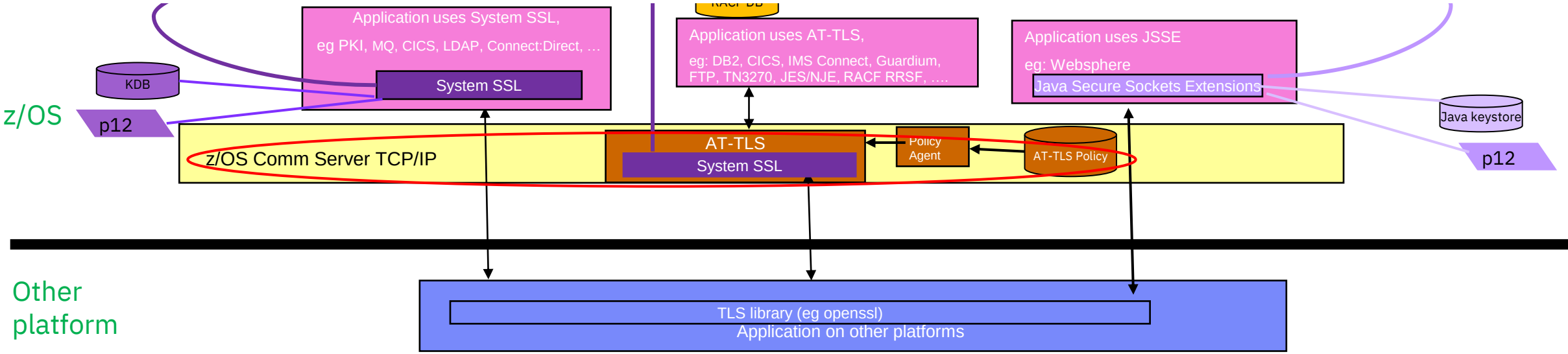
AT-TLS exploiters

- Some IBM exploiters:
  - FTP
  - CSSMTP
  - z/OS LDAP
  - DB2
  - CICS
  - IMS
  - RACF Remote Resource Sharing Facility (RRSF)
  - zSecure
  - z/OSMF



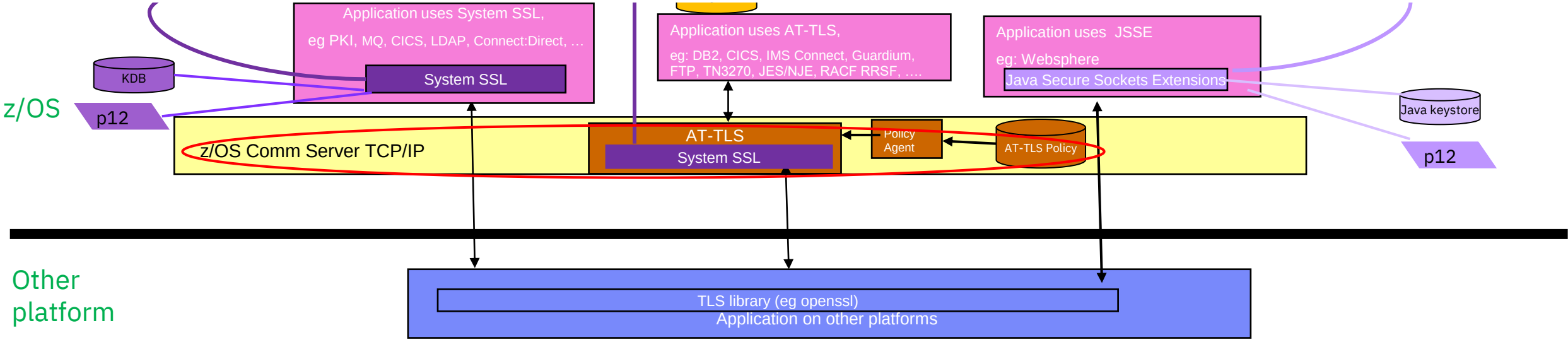
AT-TLS vs SSL setting

- Policy agent specifies majority of System SSL settings according to the policy statements, choose its own if it is not specified and pass it to SSL API
- Policy agent can also specify a parameter value when it invokes SSL API
- Either way the values from the policy agent will trump the value set from the SSL environment variables
- Some corresponding variables (System SSL in purple, AT-TLS in brown)
  - GSK_PROTOCOL_TLS<version> - TLS<version>
  - GSK-TLS_SIG_ALG_PAIRS – SignaturePairs
  - GSK_OCSP_RESPONSE_SIGALG_PAIRS – OCSPResponseSigAlgPairs



AT-TLS Configuration

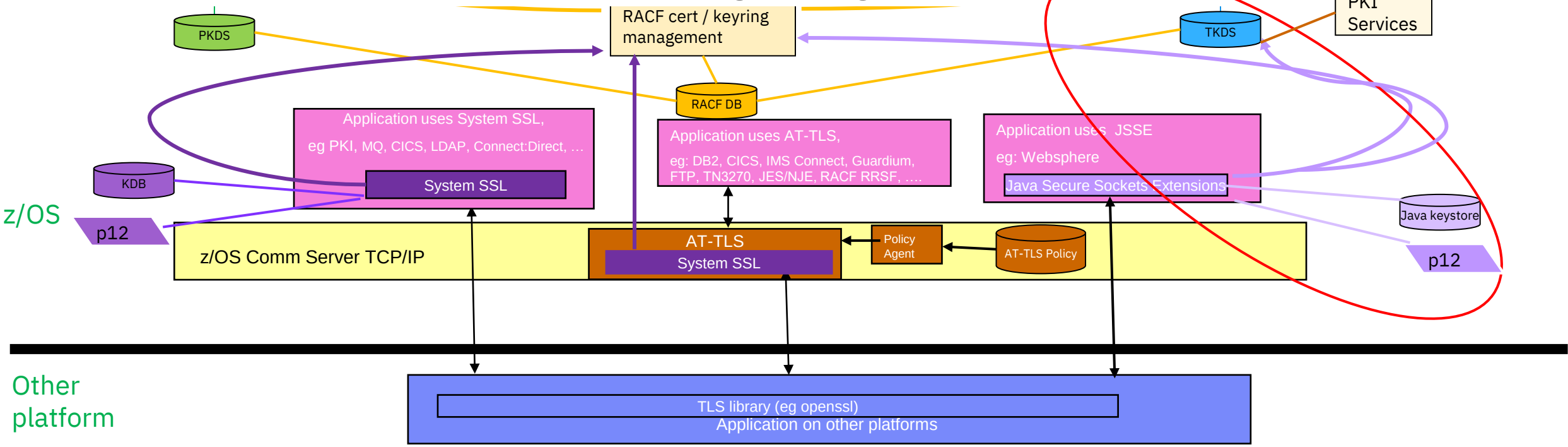
- AT-TLS policy statements set up
  - coded by hand
  - z/OSMF plugin IBM Network Configuration Assistant (NCA)



Java Secure Socket Extension

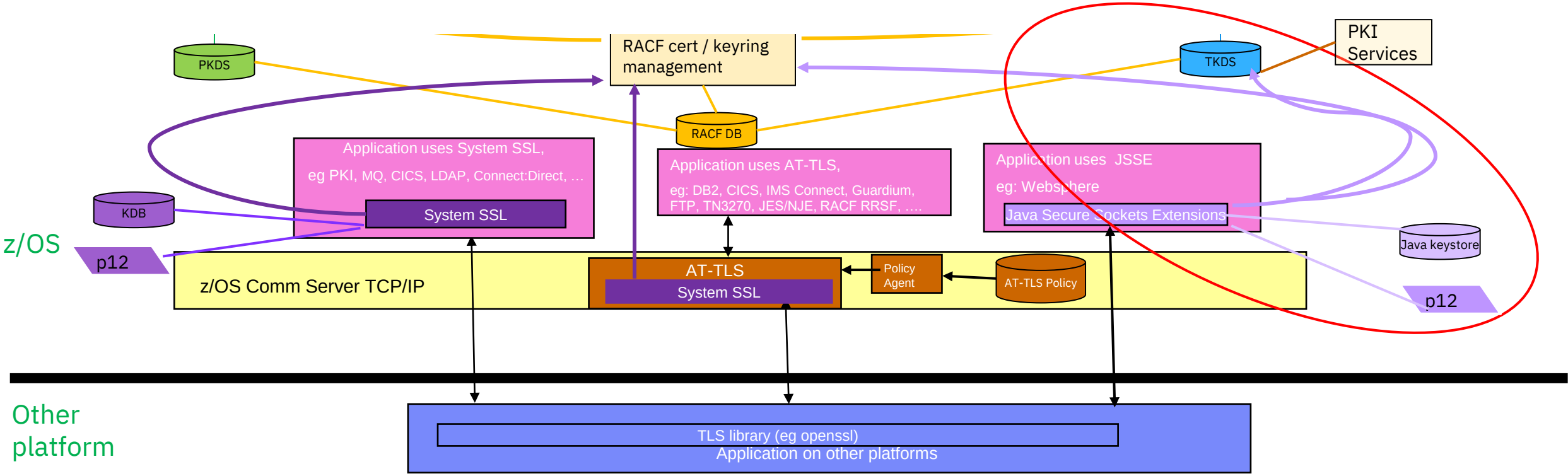
The big picture – next stop Java Secure Socket Extension

- Java provides TLS implementation via Java Secure Sockets Extension (JSSE), similar role as System SSL
- JSSE invokes Java Cryptographic Extension (JCS) providers (eg, IBMJCECCA, SunPKCS11), for cryptographic operations
- Other than its own keystore, truststore and PKCS#12 file, it also supports SAF keyrings, and z/OS token
- Java also calls the same R_datalib service to manage keyring, which can contain key in RACF or PKDS



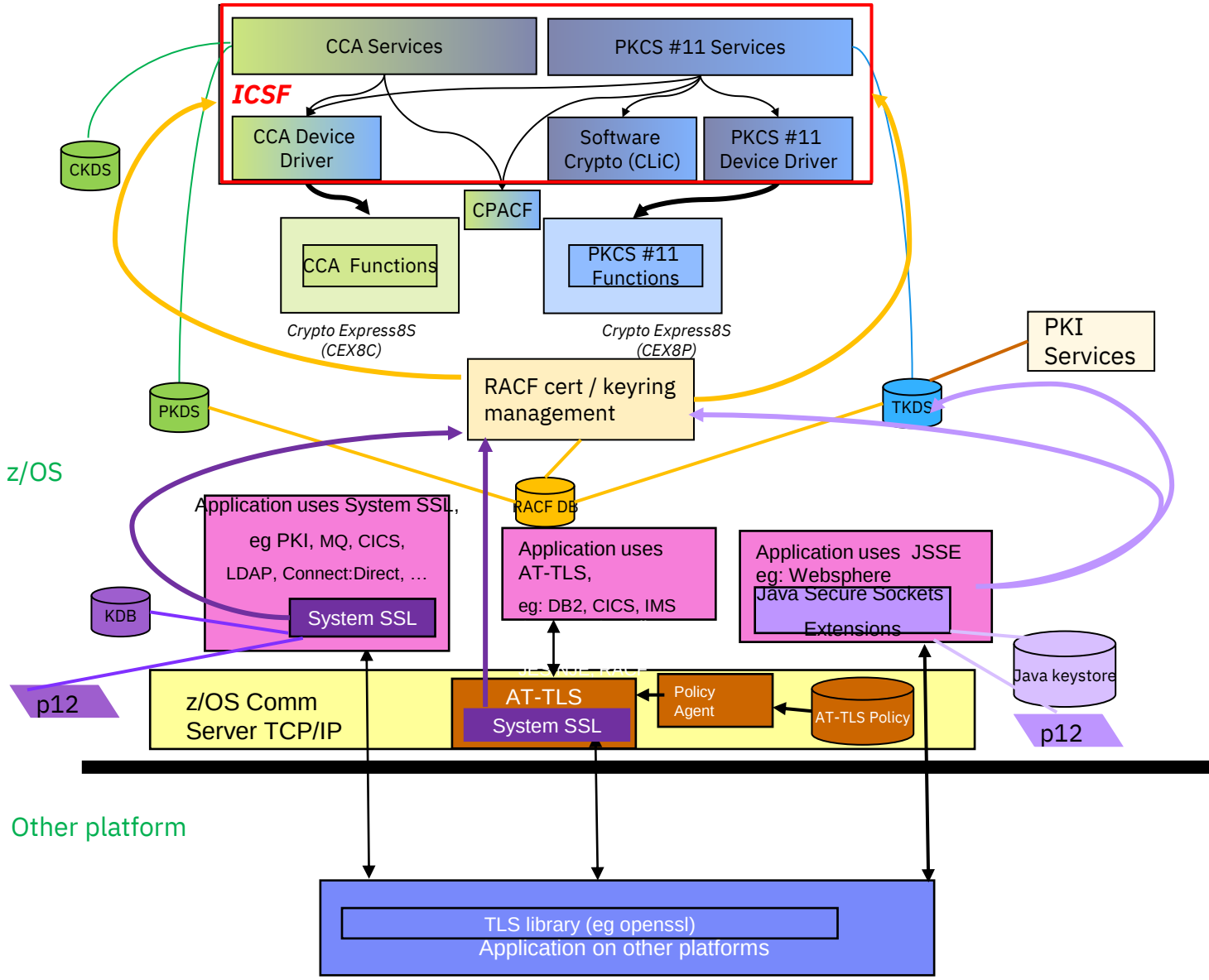
JCEE exploiters

- WebSphere and Liberty are the main exploiters of JCEE
- Notice that Java token support is not through RACF like System SSL does. It uses the SunPKCS11 provider



Summary

- ICSF provides crypto functions through hardware or software. Software is supported by CLiC
- RACF set up certificate/key stores with keys in RACF keyring, PKDS or TKDS
- System SSL and JSSE can set up certificates/key stores with keys in their own DBs
- During handshake, System SSL or JSSE retrieves the certificate and key from the certificate/key stores. If the store is a RACF keyring or a PKCS#11 token, the RACF callable service R_datalib is used
- The retrieval of key is based on the input attribute when R_datalib is called
- An application can call System SSL directly or use Communication Server AT-TLS to implement the TLS protocol



References

- **Articles Controlling TLS settings on z/OS**

  https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/flora-gui1/2023/08/14/zos-tlsssl-config-info-hub?CommunityKey=406e5630-08ab-45a7-8592-d1c960f86311

- **RACF Command Language Reference (The RACDCERT chapter)**

  https://www.ibm.com/docs/en/SSLTBW_3.1.0/pdf/icha400_v3r1.pdf

- **RACF Callable Services (R_datalib, initACEE)**

  https://www.ibm.com/docs/en/SSLTBW_3.1.0/pdf/ichd100_v3r1.pdf

- **RACF Security Administrator's Guide (Chapter on managing certificates scenarios)**

  https://www.ibm.com/docs/en/SSLTBW_3.1.0/pdf/icha700_v3r1.pdf

- **ICSF Application Programmer's Guide**

  https://www.ibm.com/docs/en/SSLTBW_3.1.0/pdf/csfb400_icsf_apg_hcr77e0.pdf

- **System SSL Programming**

  https://www.ibm.com/docs/en/SSLTBW_3.1.0/pdf/gska100_v3r1.pdf

- **Communication Server IP System Administrator's Commands**

  https://www.ibm.com/docs/en/SSLTBW_3.1.0/pdf/halu101_v3r1.pdf

- **IBM Semeru Runtime Certified Edition for z/OS 17**

  https://www.ibm.com/docs/en/semeru-runtime-ce-z/17

Please submit your session feedback!

- All done via the Whova App
- QR Code to the right to download the Whova App
- This session is ON