z/TPF Detailed Summary
Program Management

Dennis Fallon
z/TPF Development

Program Management

- What's on the system ?
  - How does it get there ?
    - Images
    - Loaders
- Application Programs
  - Linkage
  - Commands
- File System and Program Management
  - Program Base Unique Files
  - Common Deployment

Program Management
Components of a z/TPF Image

The image pointer record (CTKX)

IPL programs (IPLA and IPLB)

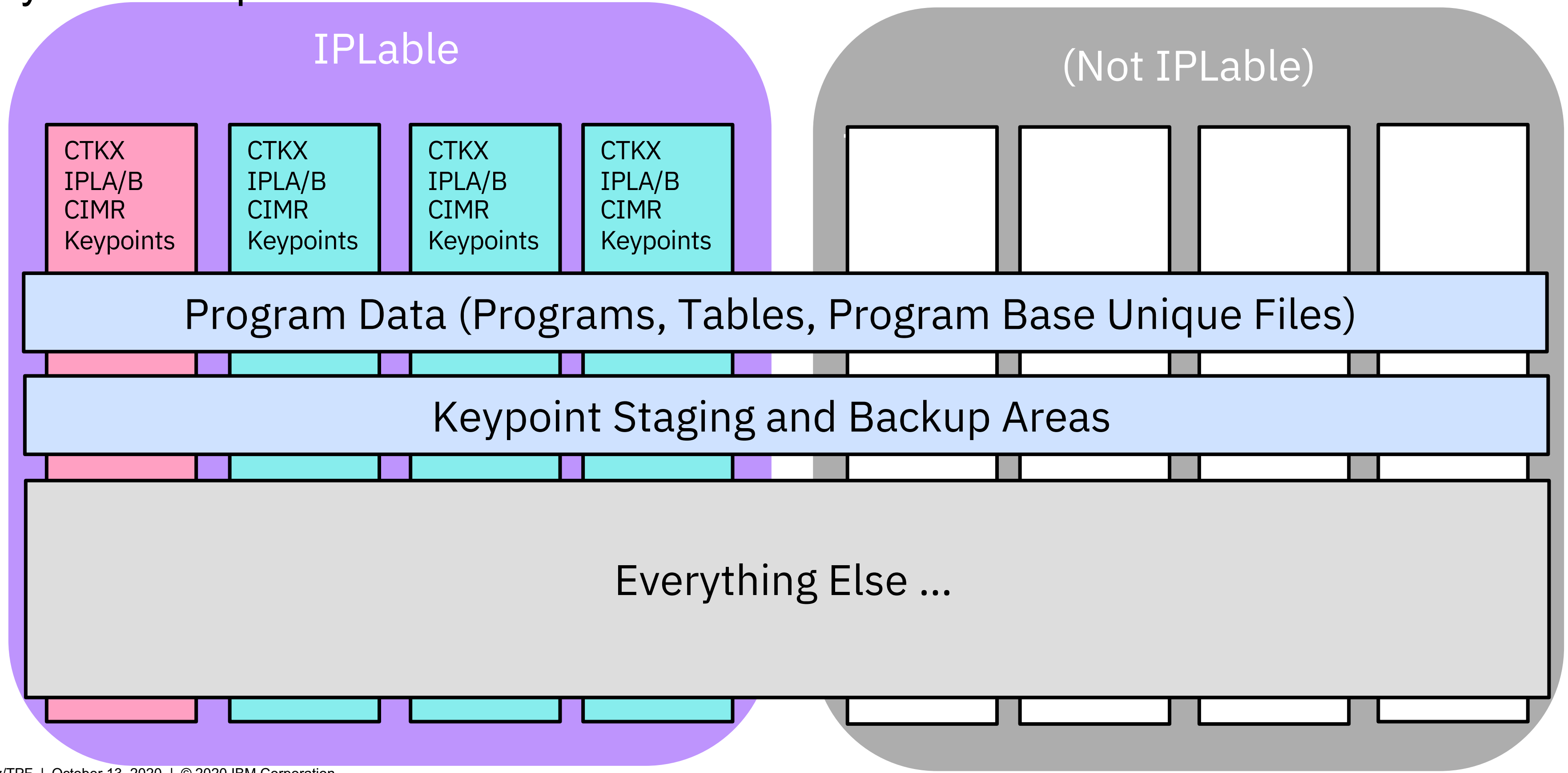
Core image restart (CIMR) area components
- Control program (CP)
- In-core dump formatter (ICDF) program
- Online component of the general file loader (ACPL)
- Global synchronization table (SIGT)
- Record ID attribute table (RIAT)
- File address compute program table (FCTB)
- User defined component (USR1)
- User defined component (USR2)

Keypoints
- Keypoint working area
- Keypoint staging area
- Keypoint backup area

Application program base components
- Program attribute table (IPAT)
  - Loaded under @CIMR section in input file
- E-type programs and program base unique files

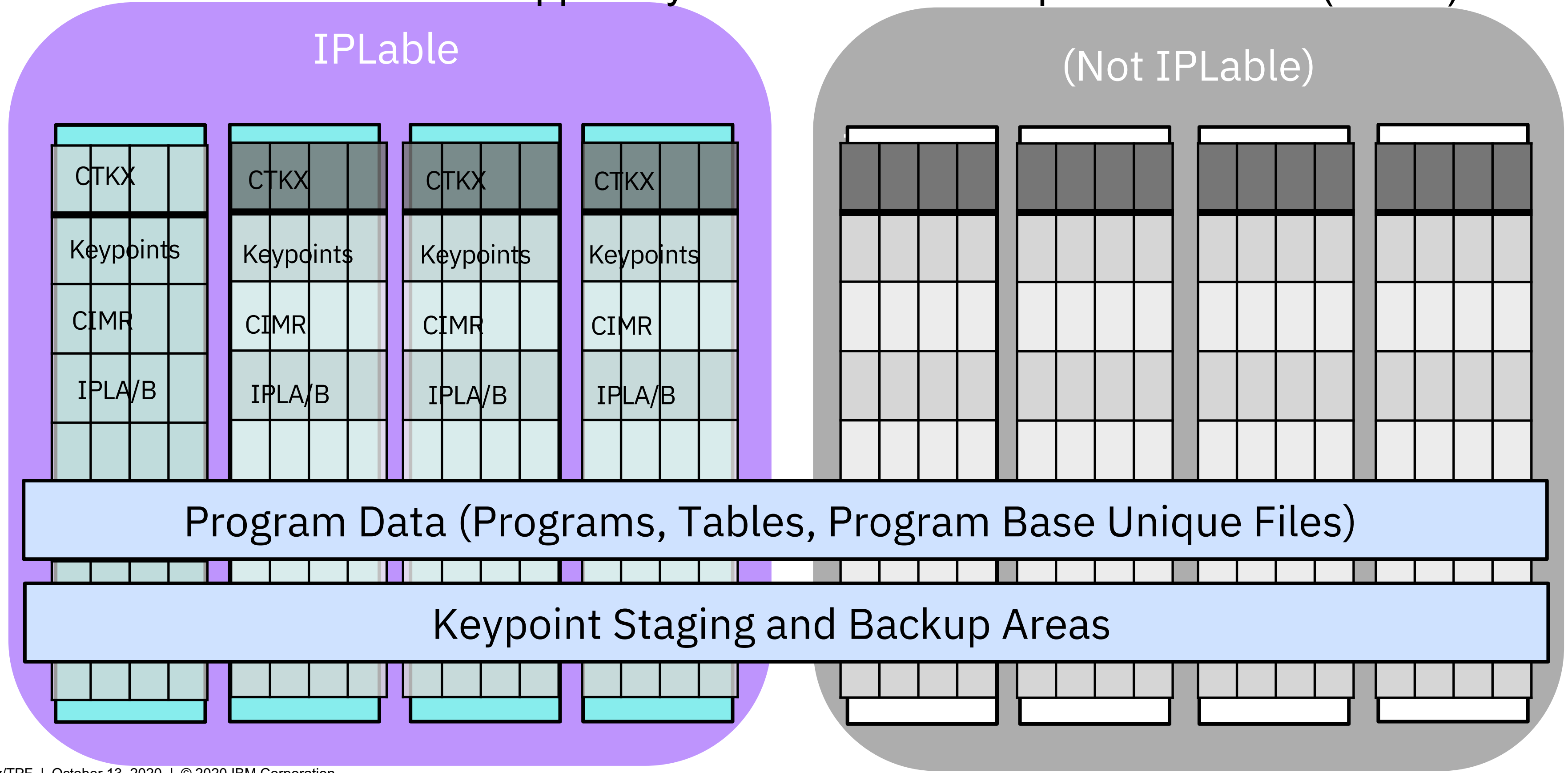
Program Management

System Components on DASD



Program Management

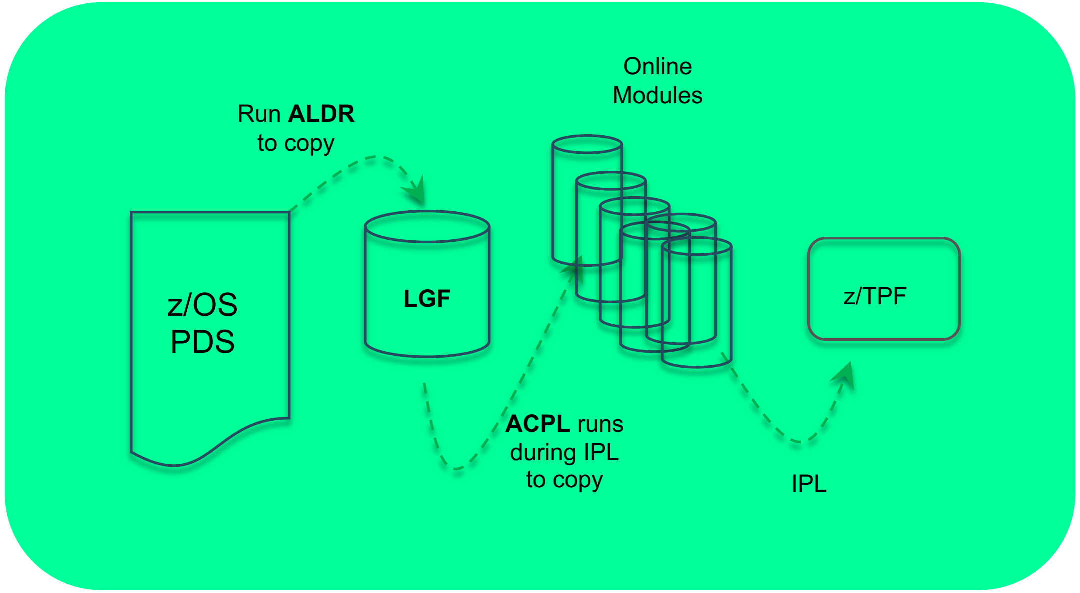
Data Resides in Records Mapped by File Address Computation Table (FCTB)



Program Management

General File Loader (The “chicken and the egg” loader)

The problem:
System components and programs are built “offline” (on z/OS or z/Linux) and need to be written to unique locations across multiple DASD using database mapping that only z/TPF understands !



ALDR: The offline portion of the general file loader. Runs on z/OS.
LGF: Loader General File. A single DASD mod, IPLable, mini-z/TPF system designed to run ...
ACPL: The online portion of the general file loader. Runs with a LGF is IPLed.

Program Management
General File Loader, cont'd

Used at system generation time, or for emergency load when no fallback image is available.

When building a new system, need to format both the Loader General File (LGF) and Online DASD Modules.

ALDR is the program that places the binaries onto the LGF

Intention is for ALDR to be a boot strap loader
- Cannot load (file system) files using ALDR
- Not enough space on LGF to hold entire program base
- Load IBM programs and essential applications

When the LGF is IPLed, the binaries are copied to the Online DASD modules.
- The general file loader always overwrites **image 1**
- Must be defined to use **program area 1 and IPL area 1**

What's an image ? (Or a program area or an IPL area ?)

Food for thought … If ACPL runs FROM the LGF, why is it loaded TO the CIMR area on the online DASD ?

Program Management
z/TPF Images

An image is a complete set of system components that can be IPLed
- Image selection is done at very beginning of IPL
- Each processor in a loosely coupled complex can be running on a different image

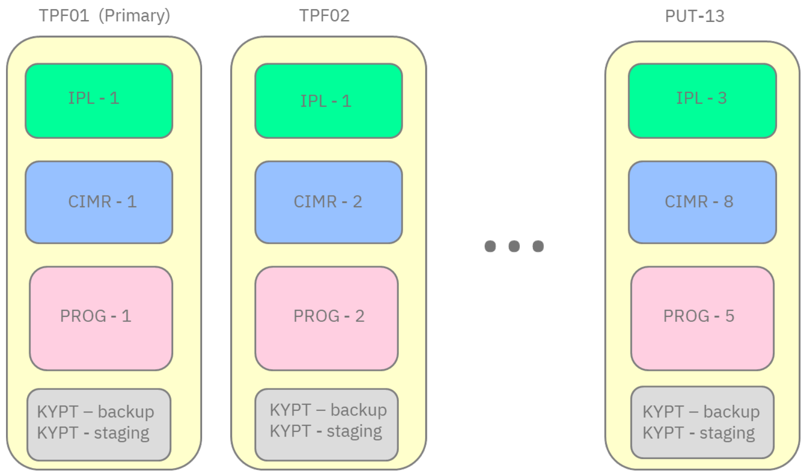
Up to 8 different images can be defined in the system - one image is marked as Primary
- The IPL area of the primary image is used on a hardware IPL

Each image has a unique Core Image Restart (CIMR) area

IPL and program base areas can be shared between images

There is only <u>one</u> working copy of Keypoints

ALDR (General File Loader) always loads to TPF01



Program Management

Why ? And additional info

Fallback:
- New code can be loaded to an image and then IPLed in order to begin using the new code
- If a problem is found with the code, the system can be re-IPLed using the previous image or another fallback image

Test Systems:
- Can maintain multiple builds for users to choose from

Coexistence/Migration on loosely coupled systems
- Can IPL one processor on a new image while other processors handle traffic on existing image and migrate over time

TPF ZIMAG command allows you to define the contents of each image, change the primary image, and enable and disable an image

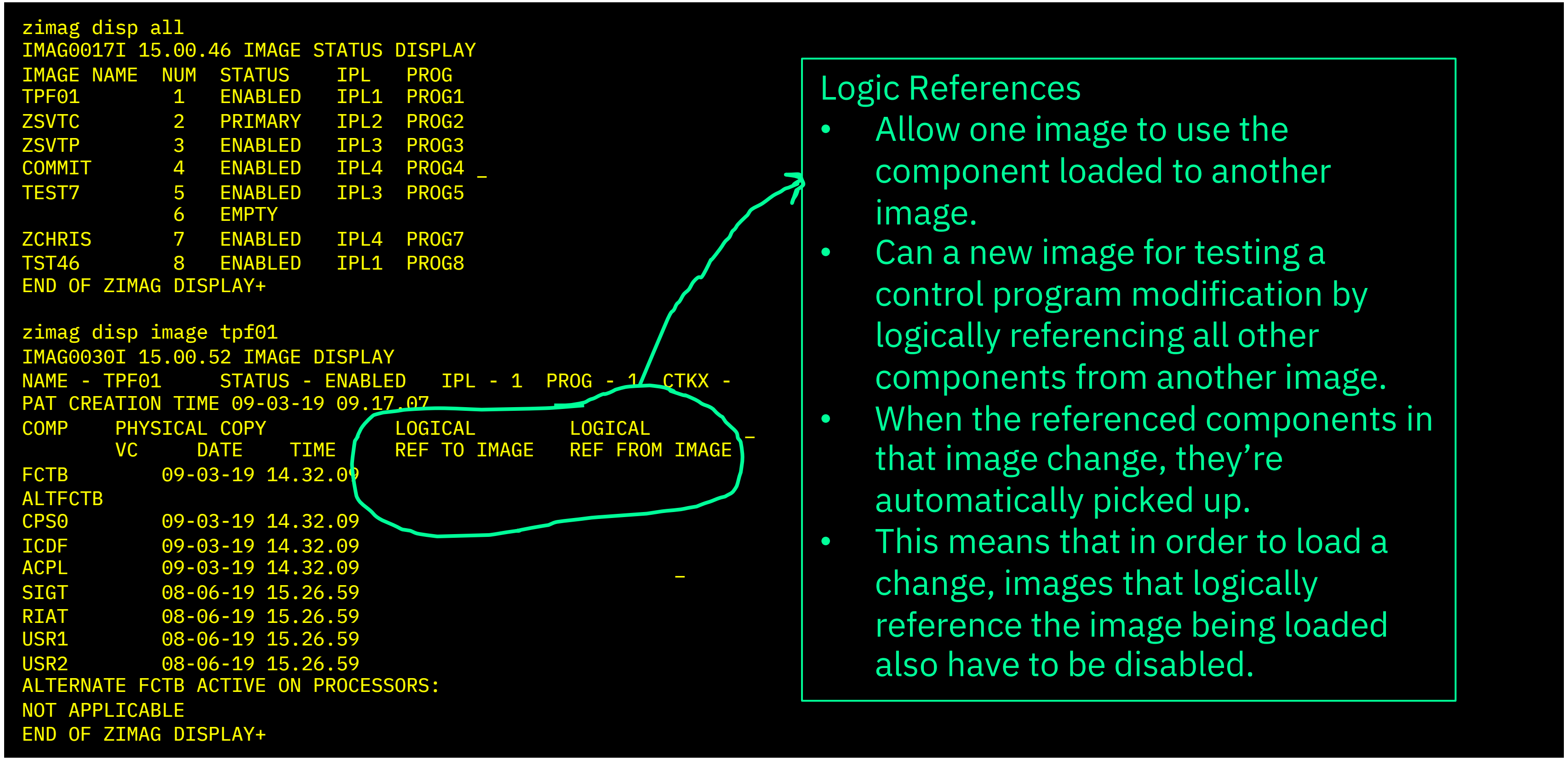
Only an enabled image can be selected during IPL

An image must be disabled in order to modify its contents

CIMR components can be copied from one image to another
- Can be either a physical copy or a logical copy (reference)

Program Management

ZIMAG Command



```
zimag disp all
IMAG0017I 15.00.46 IMAGE STATUS DISPLAY
IMAGE NAME  NUM  STATUS    IPL   PROG
TPF01         1  ENABLED   IPL1  PROG1
ZSVTC         2  PRIMARY   IPL2  PROG2
ZSVTP         3  ENABLED   IPL3  PROG3
COMMIT        4  ENABLED   IPL4  PROG4 _
TEST7         5  ENABLED   IPL3  PROG5
              6  EMPTY
ZCHRIS        7  ENABLED   IPL4  PROG7
TST46         8  ENABLED   IPL1  PROG8
END OF ZIMAG DISPLAY+

zimag disp image tpf01
IMAG0030I 15.00.52 IMAGE DISPLAY
NAME - TPF01     STATUS - ENABLED   IPL - 1  PROG - 1  CTKX -
PAT CREATION TIME 09-03-19 09.17.07
COMP     PHYSICAL COPY            LOGICAL        LOGICAL
         VC      DATE     TIME    REF TO IMAGE   REF FROM IMAGE _
FCTB         09-03-19 14.32.09
ALTFCTB
CPS0         09-03-19 14.32.09
ICDF         09-03-19 14.32.09
ACPL         09-03-19 14.32.09                              _
SIGT         08-06-19 15.26.59
RIAT         08-06-19 15.26.59
USR1         08-06-19 15.26.59
USR2         08-06-19 15.26.59
ALTERNATE FCTB ACTIVE ON PROCESSORS:
NOT APPLICABLE
END OF ZIMAG DISPLAY+
```

Logic References

- Allow one image to use the component loaded to another image.
- Can a new image for testing a control program modification by logically referencing all other components from another image.
- When the referenced components in that image change, they're automatically picked up.
- This means that in order to load a change, images that logically reference the image being loaded also have to be disabled.

Program Management
ZIMAG Commands

DEFINE
Allows you to define (or redefine) as many as 8 images.

ENABLE
Allows you to enable an image for an IPL.

PRIMARY
Defines an enabled image as the primary image. The primary image is used during a hard IPL and is valid only on the basic subsystem (BSS).

DISABLE
Disables an enabled image so that it cannot be IPLed.

CLEAR
Deletes a disabled image.

COPY
Allows you to copy core image restart area (CIMR) components from one image to another by reference (logically) or physically.

UNREF
Deletes the logical references of CIMR components from a disabled image.

MAKEPHYS
Allows you to make all of your CIMR component references physical copies.

DISPLAY IMAGE
Displays the image name, status, associated IPL and program areas, CTKX version code (if physically loaded), and CIMR component.

DISPLAY PROG
Displays all of the program areas defined in the z/TPF system and identifies which (if any) images they are associated with.

DISPLAY IPL
Displays all the IPL areas that were loaded and identifies which (if any) images they are associated with. This option also displays IPLA and IPLB information for each area.

DISPLAY PROCESSOR
Displays the image associated with each processor in the complex, as well as the status of the processor.

Program Management

Loading to an Image (or “the loader formerly knows as Tape Loader”)

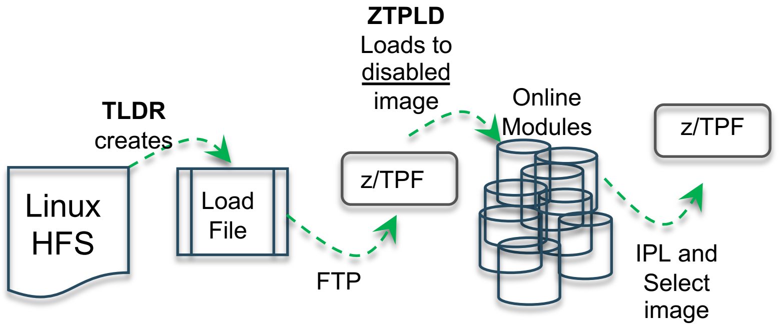
First, the offline component of the “Image Loader” (TLDR) creates a loadfile and then an online component (the ZTPLD command) processes the loadfile

There are multiple means (media) to transfer a loadfile to the online z/TPF system
- General Data Set (DASD shared between z/TPF and z/OS)
- Tape
- VRDR (when running under z/VM)
- Binary file (via FTP)

Based on the media, a data definition name is defined to z/TPF to identify the input
- ZDSMG DEF LOAD1 PATH='/usr/user1/load1' SAVE

Then the command to process the loadfile
- ZTPLD TPF02 LOAD1

Once the load completes, need to enable the image before IPLing



Program Management
Loading Keypoints

Control program keypoints are data records used to maintain system operations. These records reflect the current status of the system and are essential to the startup/restart process. They contain information about hardware (tape, DASD) and are shared between images.

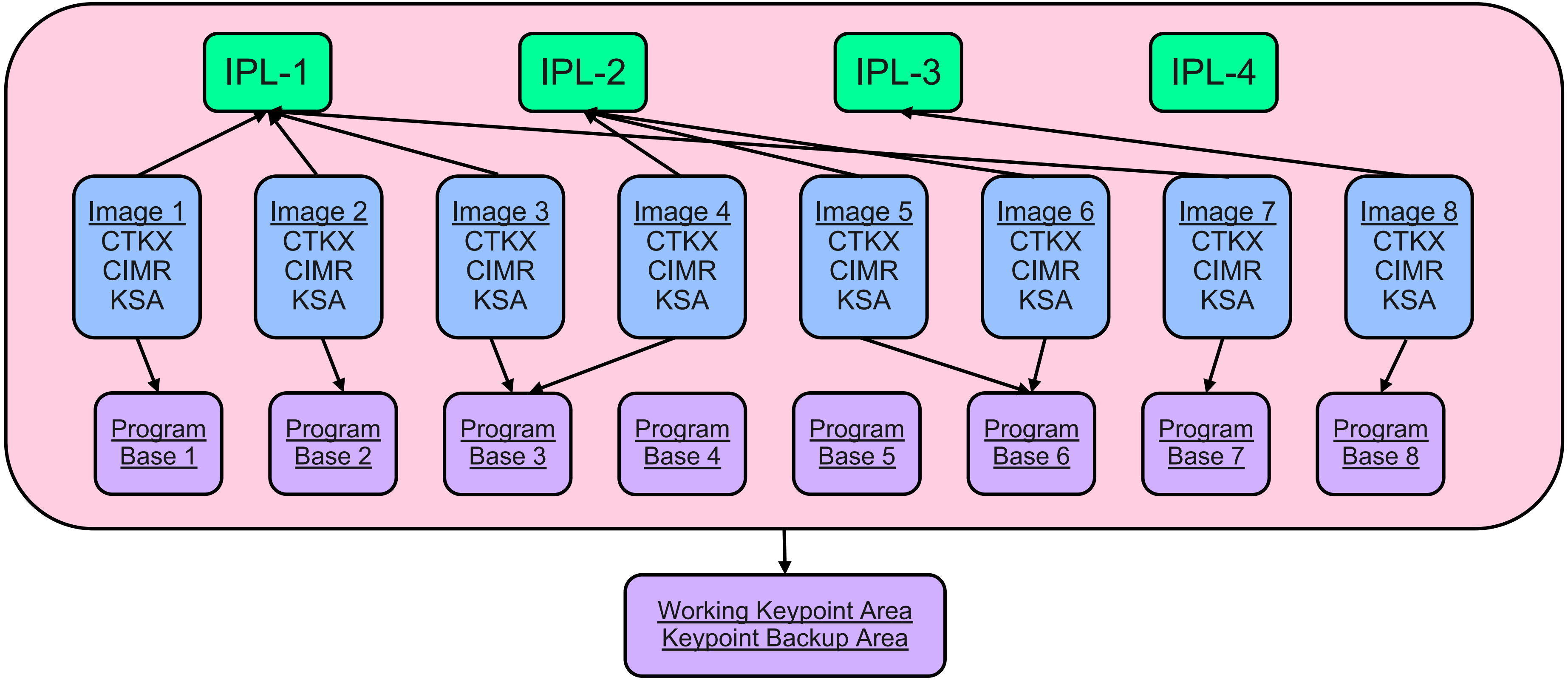
When keypoints are loaded to an image, they are put into a staging area made up of image unique records
- #KSAx where x is the image number.

Once the load completes, and the image is enabled, the keypoints can be moved to the working area
- #KEYPT
- All processors impacted by the move must be deactivated before the move completes in order to avoid overwriting the newly moved keypoint with incorrect data

When moving a keypoint to the working area, the existing keypoint is copied to the backup area first
- #KBA
- The backup area needs to be empty for a move to complete successfully
- Can restore keypoints from the backup area

Program Management

Updated view of z/TPF Images



Program Management

Example ALDR Loader Input File

```
@DEFINE
  SYSID=BSS
  IMGCLEAR=YES
  CWD = /tpf/z11/build/bss/load/
  &APPATH = /tpf/z11/build/opensource/stdload/:
          /tpf/z11/build/base/stdload/:
          /tpf/z11/build/bss/load/

@CTKX
  CTKX.kpt

@CIMR
  CPS0.so, ACPL.so, ICDF.so, SIGT.so, RIAT.so,
  IPAT.so, USR1.so, USR2.so, FCTB.so

@IPL
  IPLA.so, IPLB.so

@GFKEYPOINT
  CTKAGF.kpt, CTKVGF.kpt
  @@CTKV 8 000E  /* patch x'000E' into keypoint at offset 8 */

  @INCLUDE /tpf/z11/build/bss_gfk.loadlist
```

```
@KEYPOINT
  CTKA.kpt  %B, CTKA.kpt  %C, CTKA.kpt  %D,
  CTKB.kpt  %O, CTKC.kpt,  CTKD.kpt,
  CTKEPB.kpt  %B, CTKEPC.kpt  %C, CTKI.kpt

@INCLUDE /tpf/z11/build/bss_kpt.loadlist

@APPLICATION  &APPATH
  CISO.so, CLBM.so, CTIS.so, CTAL.so, COSY.so,
  /tpf/z11/build/base/xml4c/load/CXML.so, CFIN.so

 @INCLUDE /tpf/z11/build/bss_app.loadlist
```

General File Keypoints:

- Loader General File is built from scratch with every ALDR load
- Must contain ALL components necessary to IPL
- ACPL loads everything on the LGF to the online system
- LGF needs all keypoints, but not all keypoints should be reloaded
- @GFKEYPOINT used to put keypoints on LGF that are not to be loaded online

Program Management

Example TLDR Loader Input File

```
@DEFINE
   OVERLAY_IPAT=YES
   SYSID=BSS
   PROGCLEAR=YES
   ELDRCLEAR=YES
   DEBUGFILES=YES
   CWD = /tpf/z11/build/bss/load/
   &APPATH =/tpf/z11/build/opensource/stdload/:
            /tpf/z11/build/base/stdload/:
            /tpf/z11/build/bss/load/
@CTKX
   CTKX.kpt

@CIMR
   CPS0.so, ACPL.so, ICDF.so,
   SIGT.so, RIAT.so, IPAT.so,
   USR1.so, USR2.so, FCTB.so

@IPL
   IPLA.so, IPLB.so
```

```
@KEYPOINT
   CTKA.kpt  %B, CTKA.kpt  %C, CTKA.kpt  %D,
   CTKB.kpt  %O, CTKC.kpt,  CTKD.kpt,
   CTKEPB.kpt  %B, CTKEPC.kpt  %C,
   CTKI.kpt  @@CTKEPB %B 1E 11FF
/* patch x'11FF' into keypoint at offset x'1E' */

@APPLICATION  &APPATH
   CISO.so, CLBM.so, CTIS.so, CTAL.so,COSY.so,
   /tpf/z11/build/base/xml4c/load/CXML.so,
   CFIN.so

@INCLUDE /tpf/z11/build/bss_app.loadlist

@FILE
   /tpf/files/data/datafile.txt
/sys/tpf_pbfiles/data/datafile.txt -p 755 -o
tpfdfltu

@VERIFY
   IPAT.so
```

Program Management

Application Program Base

Each of 8 program bases consists of:

- Programs
  Called “E-type” or “ECB-controlled” or “realtime” to distinguish them from the control program, these are customer applications and utilities and IBM-supplied programs.

- Program Attribute Table (PAT or IPAT)
  Built from “control files” and loaded using the general file loader or image loader, contains attributes for each program and is used for program linkage and many other functions and commands

- Entry Point Linkage Table (EPLT)
  Built and maintained by program management, contains information used to link to multiple entry points in assembly language programs

- Program Base Unique Files
  Unique copies of the same file name. Allows an application to use a file and both the application and file contents can be different across images.

Program Management
Program Base Unique Files

Program base unique files reside under /sys/tpf_pbfiles

- /sys/tpf_pbfiles is a special target that gets translated to /tpf_pbfiles/n where ‘n’ is the program base number currently in use.
- Files can be loaded to /sys/tpf_pbfiles using the image loader and the E-type loader.
- A file, /sys/tpf_pbfiles/myApp/config.csv created in /sys/tpf_pbfiles on image IMAGE02 (running with program base 2) is actually created as /tpf_pbfiles/2/myApp/config.csv.
- The same file created on image IMAGE03 (running with program base 3) is created as /tpf_pbfiles/3/myApp/config.csv.
- An application running on IMAGE02 that does an fopen(“/sys/tpf_pbfiles/myApp/config.csv”) will see the program base 2 version of the file while the same application running on IMAGE03 will see the program base 3 version of the file.
- Allows you to update an application and a corresponding file and load both together, while another image runs backlevel code and sees a corresponding backlevel file.

Program Management

E-Type or Online Loader (OLDR)

The E-Type or online loader provides for activation and control of different program and file versions while the system is up and running.

To handle interface changes between programs, there is the concept of a loadset that acts as a unit where all items in the loadset are acted upon at the same time.

Primary loadset operations are the following:

– Load - writes the binaries to the z/TPF system
– Activate - allows transactions to use the new versions
– Accept – overlays base versions of programs with versions in the loadset
– Deactivate - stops transactions from using the new versions
– Delete - removes the binaries from the z/TPF system

Program Management
E-Type or Online Loader (OLDR)

- Both ALDR (general file loader) and TLDR (image loader) require IPLing the system in order to use new programs
- E-type loader (OLDR) designed to address the requirement for loading application programs without requiring an IPL
- Also allows fallback to prior versions without an IPL
- For realtime programs ONLY
  - Also limited support for files in TPF filesystem

Program Management
How Does It Work ?

- Several programs with interdependencies are loaded simultaneously in a “loadset”
- All programs in the loadset are written to fixed file records
- After the entire loadset has been written, the loadset can be “activated”
- Programs in the loadset are read into memory – this takes a small amount of time, but many ECBs will be created during this process
- ECBs created before and during the process of loadset activation will not use any programs in the loadset
- ECBs created after the last program is read in will be able to use the newly activated versions

Program Management
Some E-type Loader Facts

- ZOLDR command manages loadsets
- Loadsets do nothing until activated
- Activated loadsets can be deactivated, then reactivated or deleted
- Loadsets can be “accepted”
- Programs in an accepted loadset overwrite the base versions (that may have been loaded by ALDR or TLDR)
- Loadsets are associated with the program base of the active image on whatever processor the command is run
- Several loadsets can contain different versions of a single program
- Even after a loadset is deactivated or accepted, multiple versions of the program may persist until all ECBs that can potentially use the obsolete version finish processing

Program Management

E-Type or Online Loader (OLDR)

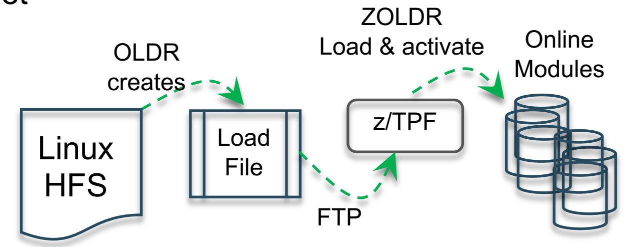
Similar to the Image Loader, the Online Loader has an offline component that creates a loadfile and then an online component processes the loadfile

There are multiple means (media) to transfer a loadfile to the online z/TPF system
- General Data Set (DASD shared between z/TPF and z/OS)
- Tape
- VRDR (when running under z/VM)
- Binary file (via FTP)

Based on the media, a data definition name is defined to z/TPF to identify the input
- ZDSMG DEF LOAD1 PATH='/usr/user1/load1' SAVE

Then the command to process the loadfile, with option to process a single loadset
- ZOLDR LOAD LOAD1

Once the load completes, need to activate the loadset



Program Management
E-type Loader ZOLDR Command

ZOLDR LOAD – move loadsets from OLDR output medium to fixed file records

ZOLDR ACTIVATE – activate a loadset on one or more processors

ZOLDR DEACTIVATE – deactivate a loadset on one or more processors
- ECBs currently using the loadset continue to use it until they exit
- EXIT and FORCE options are used to cause existing ECBs to stop using the loadset in case of problems

ZOLDR EXCLUDE – exclude a specific program from an active loadset

ZOLDR REINCLUDE – reinclude a previously excluded program in a loadset

Program Management
E-type Loader ZOLDR Command, cont’d

ZOLDR ACCEPT – Replace base versions of programs with versions in an active loadsets

ZOLDR DELETE – Delete a deactivated loadset from the system
- Delayed until all ECBs still using the deactivated loadset exit

ZOLDR DISPLAY – almost anything imaginable related to loadsets

Program Management
E-type Loader and Files

A loadset can contain program base unique files alongside programs.

- When the loadset is activated, new ECBs will start using the new programs and the new files.

A loadset can* also contain programs that reside outside of /sys/tpf_pbfiles.

- *A user exit, UELK, must be updated in order to permit such files.
- Even though those files are shared by all program bases, the new version is only visible to processors running on the same program base.
- Because those files are shared by all program bases, accepting the loadset will overwrite a file that can potentially impact another processor. To prevent problems, a loadset containing files that are not program base unique can be accepted only if all of the processors in the z/TPF complex are running on the same program base.

Program Management

Example OLDR Loader Input File

```
@DEFINE
SYSID=BSS
DEBUGFILES=YES
&APPATH = /tpf/z11/build/opensource/stdload/:
          /tpf/z11/build/base/stdload/:
          /tpf/z11/build/bss/load/

@LOADSET TEST1 &APPATH
CISO.so, CLBM.so, CTIS.so

@FILE /tpf/files/data/datafile.txt
      /sys/tpf_pbfiles/data/datafile.txt -p 755 -o tpfdfltu

@LOADSET TEST2 &APPATH
CFIN.so, CXML.so, CTAL.so

@LOADSET TEST#
XNEE.so, XXAA.so

@VERIFY /tpf/z11/build/bss/load/IPAT.so
```

@VERIFY:

- Specifies a Program Attribute Table that will be used to verify that all programs in the loadset are in that PAT table
- The E-type loader can load programs that are not in the table (see next slide) with limitations

Program Management

E-type Loader and “Unallocated Programs”

When a program is loaded by ZTPLD, there must be an entry in the PAT so ZTPLD knows where to write the program.

The E-type loader only overlays the base program location if/when the loadset is accepted. Before that, the program resides in temporary file storage.

A program that does not have an entry in the PAT can be loaded using the E-type loader. However, it cannot be accepted.

When z/TPF is IPLed, a fixed number of spare PAT slots are allocated and the ZAPAT ADD command can use those to add (that number of) programs to the PAT without requiring an IPL.

Once ZAPAT ADD is done, a loadset containing the program can be accepted.

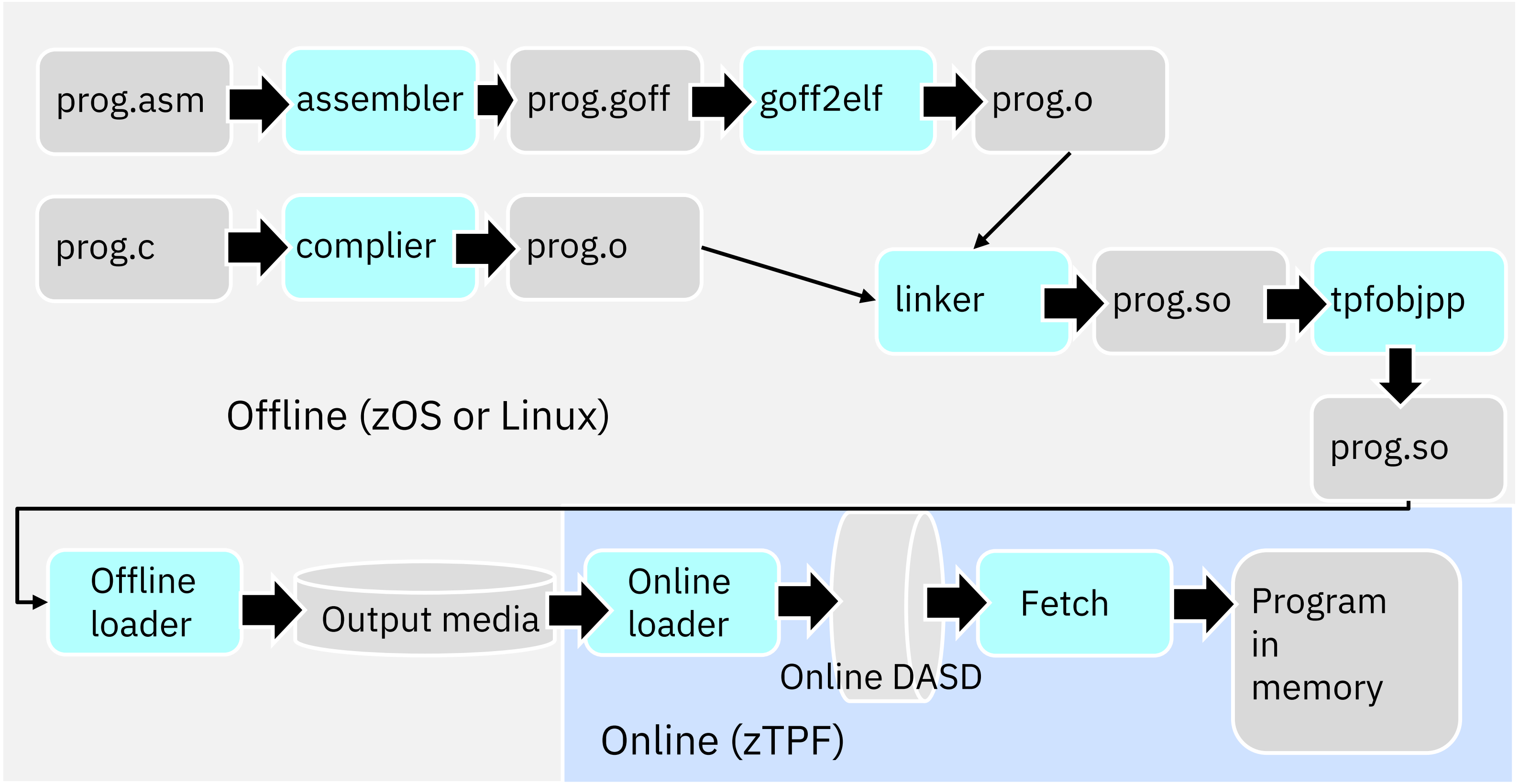
The PAT was originally the “Program Allocation Table”

- It was built by a program called the “allocator” and provided a “slot” for each program in the system.
- There was (and still is) a 1:1 correspondence between a PAT slot index and the ordinal number of the primary record for the program.
- When a program was no longer needed, it was marked “SPARE” to keep entries in order.
- The allocator tables were replaced by the control files with z/TPF and programs are listed in build order.
- To ensure the order of entries is maintained online, the loaders “merge” a new PAT with the existing PAT.

The @VERIFY option in the E-type loader might be useful to ensure that programs have been added to the PAT before promoting them to production.

Program Management

Application Processing, Code to Executable



Program Management
Program Types

Early versions of TPF supported applications written in Basic Assembly Language (BAL) only.
These programs needed to fit in a single 4K file record when assembled.
As a result, legacy applications consist of hundreds of small programs and complex nested paths through them.

All programs are packaged as Shared Object (.so) libraries
- Consistent with z/TPF programming model where any program can call any other program
- Exception is Java - packaged as .jar files

BSO:
- BAL (Basic Assembly Language) Shared Object
- Made up of one or more assembly language source files written using legacy BEGIN/FINIS macros and ENTER/BACK linkage

- CSO:
  C Shared Object
- Made up of one or more source files written in C, C++ and/or assembly language that uses macros that are compatible with C linkage

Format of the Shared Object is Executable and Linkable Format (ELF)

Program Management

Program-to-program calls

There are two primary mechanisms for one program to get to another in z/TPF.

Entry Point

1. An entry point is a 4-character program name. It is invoked using assembly language ENTxC macros and from C using explicit functions (entrc()) or by simply coding the 4-character program name as a function call (ABCD()).
2. BSOs contain an entry point for each BEGIN macro (each BAL source file will contains a BEGIN macro)
3. CSOs can optionally specify an entry point. This is coded in the make file for the program (APP_ENTRY := main)

Entry points are maintained by Program Management through the Entry Point Linkage Table (EPLT).

While the 4K restriction on early BAL programs was restrictive in some cases, it was a waste of storage space in others.
As a result, multiple small functions might be coded in a single BAL source file using “transfer vectors” – multiple entry points on a single BEGIN macro call, with a branch table at the top of the code to transfter processing to the correct entry point.
These transfer vectors are still supported for legacy applications.

Program Management
Program-to-program calls, cont’d

Library Function

- CSOs can have externally visible functions (with or without an entry point).
- To call a library function, library (CSO) must appear on the “NEEDED list” of the calling program. This means the library must be listed when the caller is linked.
- Standard libraries are included by the z/TPF build tools automatically. Additional libraries (including customer application libraries) are specified in each caller’s make file:
  ```
  LIBS := CFVZ
  LIBS += CFVN
  ```

Program Management
Getting Clean Links

When a CSO is built using the z/TPF build tools, the linker will report any unresolved references. This occurs when a symbol (function or data item) cannot be found in the list of libraries given to the linker.

In the case of a library function, if the unresoved reference occurs because of a missing library, then the function call will fail on z/TPF because online linkage will not look in the needed library.

It is also possible that the link was performed offline against a copy of the library that did not contain the symbol. In this case, it is possible that the symbol will be found when the program is loaded online (if the online version of the library differs from the copy that the link was performed against).

Entry points are found using the EPLT online, so the program containing the entry point does not need to be listed in the make file. However, the linker needs to know what entry points are in the EPLT. A pair of dummy libraries, TPFSTUB.so (for IBM programs) and USRSTUB.so (for customer applications), contain symbols for every entry listed in the control files that specifies a STUB is needed. Maintain the STUB fields in the control files to ensure clean and accurate links.

Program Management

Executable and Linkable Format (ELF)

- Executable and Linking Format
  Describes the object files created by the compiler as well as the executables created by the linker

- Describes binary file format used by LINUX and documented as part of System V Application Binary Interface

- System V ABI consists of Generic ABI (gABI) that must be used in conjuunction with a processor-specific ABI supplement
  Linux for zSeries Supplement

Program Management

Types of ELF Files

- **Relocatable file**
  Holds code and data suitable for linking with other object files to create an executable or a shared object file.
  Output of compiler/assembler.

- **Executable file**
  Holds a program suitable for execution.
  Output of linker.
  Not used by TPF.

- **Shared object**
  Holds code and data suitable for linking with executable files.
  Used for all application programs by TPF.

- CIMR Components built as ELF Shared Objects
  Offline Loader strips out text as in 4.1

On Linux, a process loads an application program into a fixed (virtual) memory address. That application calls any number of libraries (shared objects) that can be loaded into memory at any address. Generally, all applications (executables) are loaded to the same memory address and this address is known at link time.

On z/TPF, all programs* reside in memory simultaneously and ECBs are able to enter any program anywhere in memory. Process startup cost is minimized because programs are loaded once per IPL.

Program Management
ELF Utilities (on Linux)

readelf
- Formatted display of ELF file contents with many options

objdump
- Semi-formatted display
- Lower level than readelf

nm
- Display symbols
- Can display long symbol names that are truncated by readelf

c++filt
- Demangle c++ function names

Program Management
ELF Segments

Readelf display using –l option to see program headers (segments)

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x000000000000accc 0x000000000000accc  R E    0x1000
  LOAD           0x000000000000b000 0x000000000000b000 0x000000000000b000
                 0x0000000000002200 0x0000000000002280  RW     0x1000
  DYNAMIC        0x000000000000d000 0x000000000000d000 0x000000000000d000
                 0x0000000000000200 0x0000000000000200  RW     0x8
  GNU_EH_FRAME   0x000000000000ac10 0x000000000000ac10 0x000000000000ac10
                 0x00000000000000bc 0x00000000000000bc  R      0x4

 Section to Segment mapping:
  Segment Sections...
   00     .hash .dynsym .dynstr .rela.dyn .rela.data .rela.got .rela.plt .init
.plt .text .fini .rodata .eh_frame_hdr
   01     .data .jcr .eh_frame .gcc_except_table .ctors .dtors .got .dynamic .bss
   02     .dynamic
   03     .eh_frame_hdr
```

Program Management

ELF Sections

Readelf display using –S option to see section headers

```
Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .hash             HASH             0000000000000120  00000120
       00000000000006b0  0000000000000008   A       2     0     8
  [ 2] .dynsym           DYNSYM           00000000000007d0  000007d0
       0000000000000ac8  0000000000000018   A       3    13     8
  [ 3] .dynstr           STRTAB           0000000000001298  00001298
       00000000000006e2  0000000000000000   A       0     0     1
. . .
  [ 6] .rela.got         RELA             0000000000001c20  00001c20
       0000000000000120  0000000000000018   A       2    20     8
  [ 7] .rela.plt         RELA             0000000000001d40  00001d40
       0000000000000708  0000000000000018   A       2     9     8
  [ 8] .init             PROGBITS         0000000000002448  00002448
       000000000000000c  0000000000000000  AX       0     0     1
  [ 9] .plt              PROGBITS         0000000000002454  00002454
       0000000000000980  0000000000000020  AX       0     0     4
  [10] .text             PROGBITS         0000000000003000  00003000
       00000000000071c4  0000000000000000  AX       0     0     4096
. . .
  [12] .rodata           PROGBITS         000000000000a1d0  0000a1d0
       0000000000000a40  0000000000000000   A       0     0     8
. . .
  [14] .data             PROGBITS         000000000000b000  0000b000
       0000000000000010  0000000000000000  WA       0     0     8
. . .
```

Program Management
ELF Header

```
ELF Header:
  Magic:   7f 45 4c 46 02 02 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           IBM S/390
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          424704 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         4
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 28
```

Program Management
ELF Needed List

```
Dynamic section at offset 0x12000 contains 20 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [CTIS]
 0x0000000000000001 (NEEDED)             Shared library: [CISO]
 0x0000000000000001 (NEEDED)             Shared library: [CTAL]
 0x0000000000000001 (NEEDED)             Shared library: [CFVS]
 0x0000000000000001 (NEEDED)             Shared library: [COMX]
 0x0000000000000001 (NEEDED)             Shared library: [COMS]
 0x000000000000000e (SONAME)             Library soname: [CMQI]
 0x0000000000000004 (HASH)               0x120
 0x0000000000000005 (STRTAB)             0xf48
 0x0000000000000006 (SYMTAB)             0x648
 0x000000000000000a (STRSZ)              784 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000003 (PLTGOT)             0x11000
 0x0000000000000002 (PLTRELSZ)           1440 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x1a20
 0x0000000000000007 (RELA)               0x1258
 0x0000000000000008 (RELASZ)             1992 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x0000000000000000 (NULL)               0x0
```

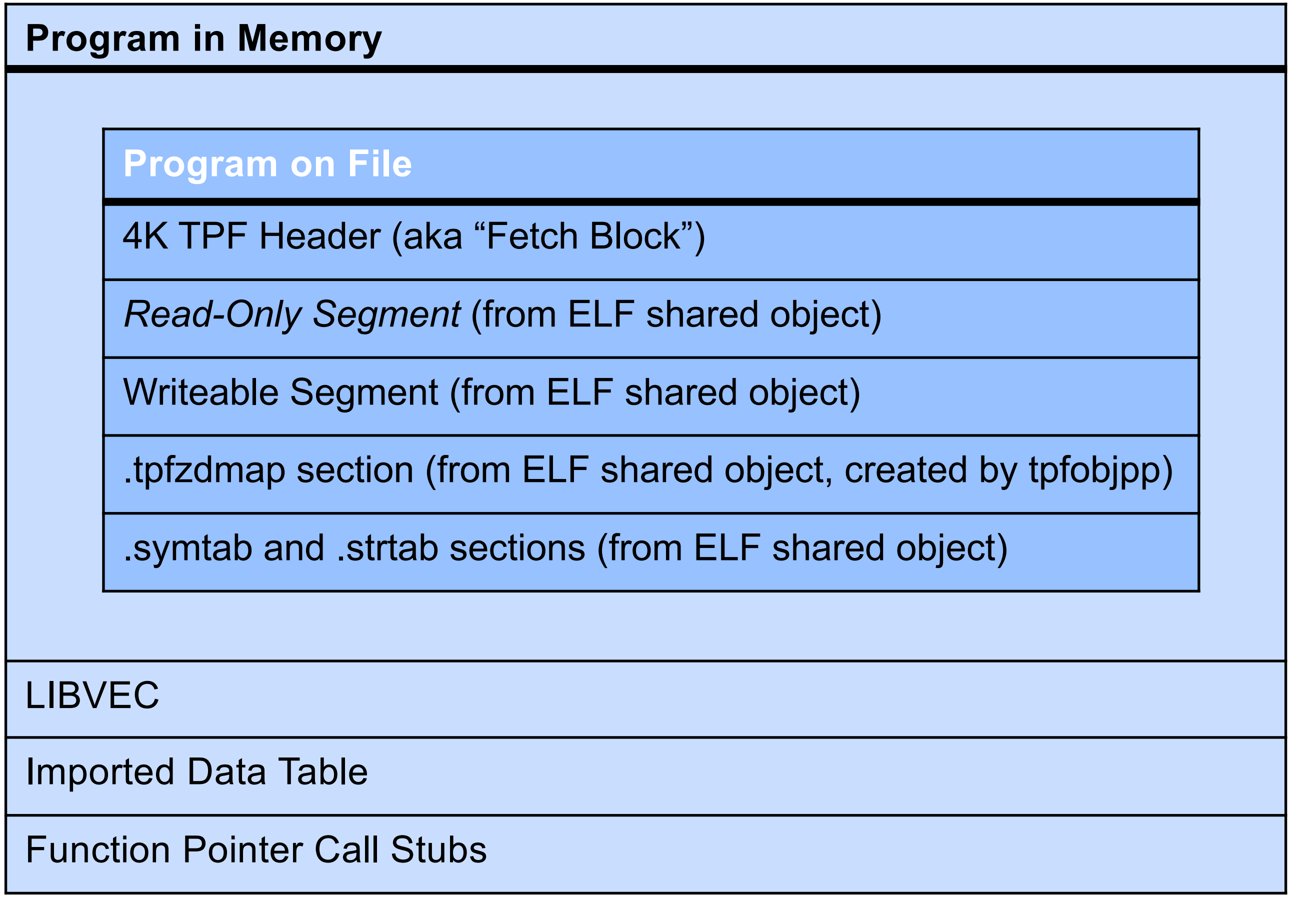
Program Management
ELF Symbol Table and Relocation Section

```
Symbol table '.symtab' contains 168 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
. . .
   126: 0000000000000000   214 FUNC    GLOBAL DEFAULT  UND seteuid
   127: 00000000000043d8    92 FUNC    GLOBAL DEFAULT   10 cngh_create_proc_line
   128: 00000000000047c0   876 FUNC    GLOBAL DEFAULT   10 createDirectories
   129: 0000000000000000   106 FUNC    GLOBAL DEFAULT  UND defrc
   130: 0000000000000000   126 FUNC    GLOBAL DEFAULT  UND sprintf
   131: 00000000000073d4  1156 FUNC    GLOBAL DEFAULT   10 _Z19tpf_cfg_process_cfghj
   132: 0000000000005fc4  3894 FUNC    GLOBAL DEFAULT   10 _Z17tpf_cfg_parse_cfghjP1
. . .

Relocation section '.rela.plt' at offset 0x1d40 contains 75 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
. . .
00000000c188  004b0000000b R_390_JMP_SLOT    00000000000047c0 createDirectories + 0
. . .

>c++filt _Z19tpf_cfg_process_cfghj
tpf_cfg_process_cfg(unsigned char, unsigned int)
```

Program Management


Program in Memory
Program on File
4K TPF Header (aka “Fetch Block”)
Read-Only Segment (from ELF shared object)
Writeable Segment (from ELF shared object)
.tpfzdmap section (from ELF shared object, created by tpfobjpp)
.symtab and .strtab sections (from ELF shared object)
LIBVEC
Imported Data Table
Function Pointer Call Stubs

Program Management
Core Resident Program Areas

64-bit Standard CRPA (some C programs)
Program 1
Program 2
Program 3
…

64-bit Copy On Write (COW) CRPA (some C programs)
Program 1
Program 2
Program 3
…

Copy-on-Write

If a program in COW CRPA writes into its own memory, the 4K page containing the modified storage is copied and translation tables are modified so only that the ECB that modified the storage sees the new copy. This is expensive and should be avoided when possible.

If a program in STD CRPA writes into its own memory, a system error (dump) occurs.

31-bit Standard CRPA (assembly language programs)
Program 1
Program 2
Program 3
…

31-bit Copy On Write (COW) CRPA (few C programs)
Program 1
Program 2
Program 3
…

Program Management
Fetch Processing

Fetch (phase 1 of 2) loads program into memory and performs local relocations
- At this point, the program is not ready to run, but other programs can resolve their references to it.

Dynamic Linkage (phase 2) resolves external references.
- At this point, the program is ready to run.

Program attribute defines when fetch processing is performed
- Preload – fetches program serially early in system restart
- On demand – fetches program on first call
- Default – on demand on a test system or in parallel during system restart

CRPA Sweeper
- Can be used to periodically remove infrequently used programs from memory

Program Management
Function Linkage – Linux Model

The Linux model supports “lazy binding”. The address of each called function is not resolved until the first time it is called. This is efficient for the process model, where executables are read into memory on every process startup, and many function calls may not be performed during the life of any given process.

function();

.text

linkage setup

```
brasl   %r14,function@PLT
```

.plt

Piece of stub code that looks up function address in .got section and branches to it

.got

8-byte address of function() initially points to code to resolve function address

.rela.plt

```
R_390_JMP_SLOT    0000000000000000 function + 0
```

.dynamic

```
(NEEDED)     Shared library: [CNG0]
(NEEDED)     Shared library: [CUDA]
(NEEDED)     Shared library: [CABC]
:
```

Program Management

Function Linkage – z/TPF Model

z/TPF has no need for lazy binding. The cost of resolving relocations for all function calls is trivial when done once per program per IPL.
The linkage code generated by the compiler and linker provides z/TPF with a mechanism for intercepting function calls and allowing z/TPF to redirect them to the correct version when multiple versions of a program have been loaded using the E-type loader.

function();

.text

linkage setup

```
brasl   %r14,function@PLT
```

.plt

compiler generated code modified by TPF

```
Load register with LIBVEC offset from
LBV3
Load register with address of PAT
stub from .got
LBV3     DS    F
```

.got

8-byte address of PAT stub

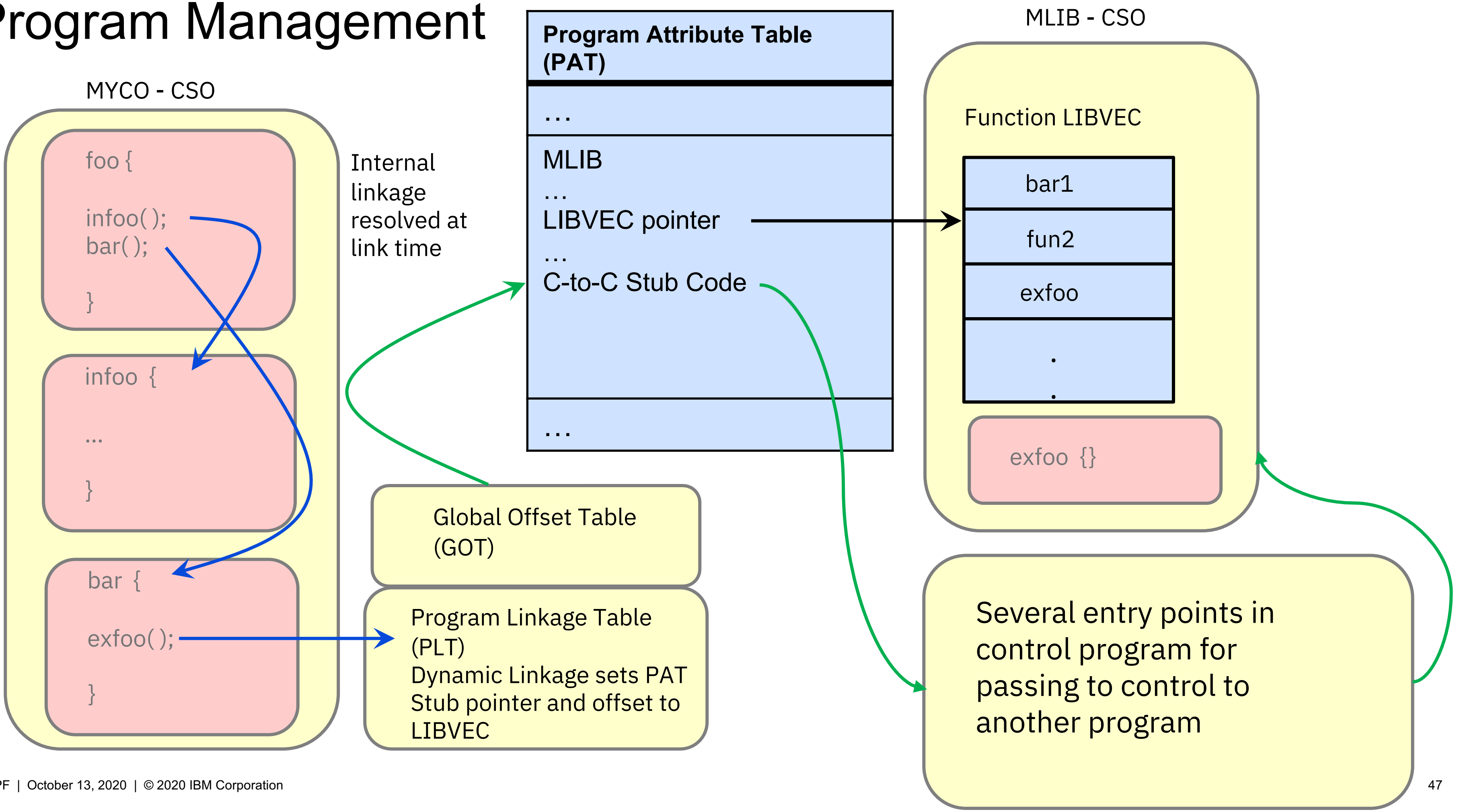
.rela.plt

R_390_JMP_SLOT   0000000000000000 function + 0

.dynamic

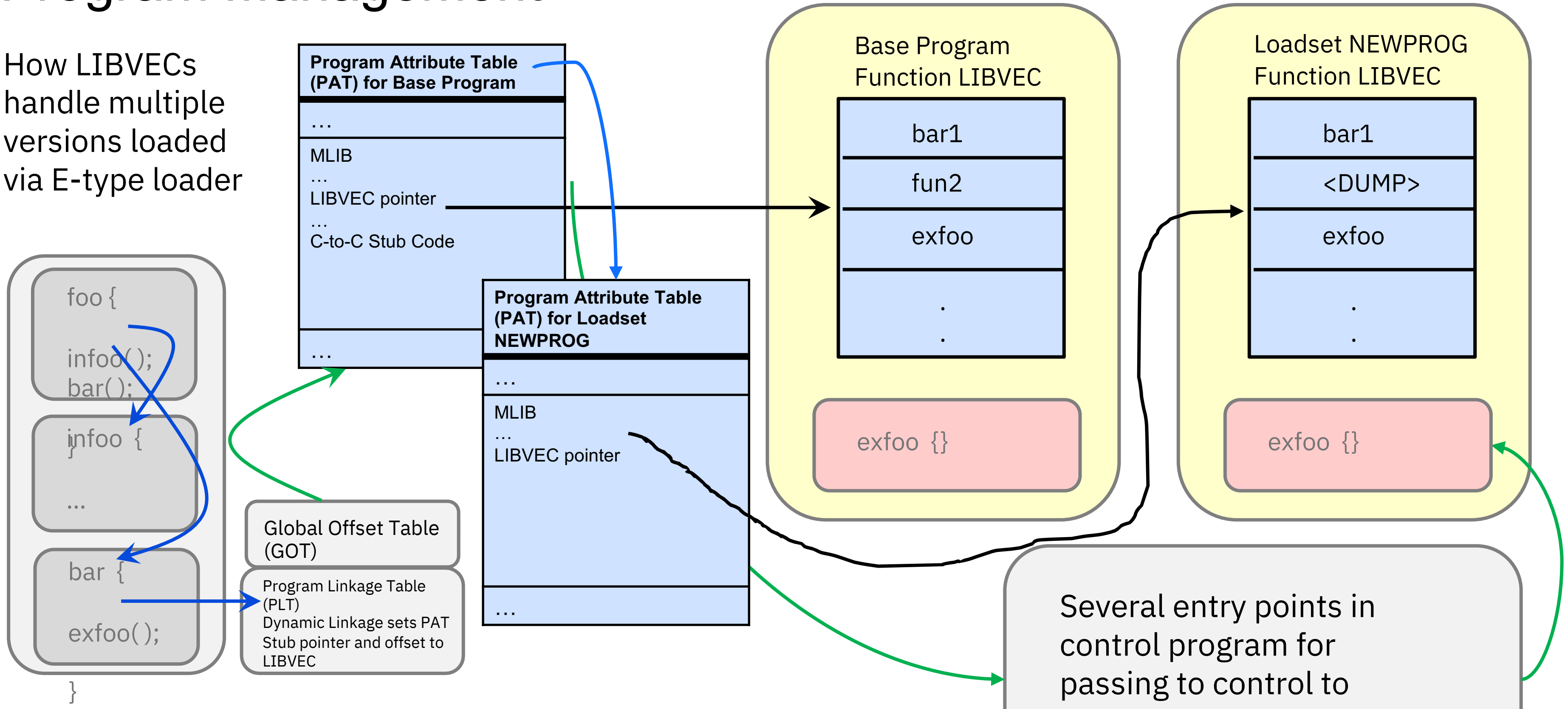
```
(NEEDED)     Shared library: [CNG0]
(NEEDED)     Shared library: [CUDA]
(NEEDED)     Shared library: [CABC]
:
```

Program Management


MYCO - CSO
foo {
infoo();
bar();
}
infoo {
...
}
bar {
exfoo();
}
Internal linkage resolved at link time
Program Attribute Table (PAT)
...
MLIB
...
LIBVEC pointer
...
C-to-C Stub Code
...
MLIB - CSO
Function LIBVEC
bar1
fun2
exfoo
.
.
exfoo {}
Global Offset Table (GOT)
Program Linkage Table (PLT)
Dynamic Linkage sets PAT Stub pointer and offset to LIBVEC
Several entry points in control program for passing to control to another program

Program Management

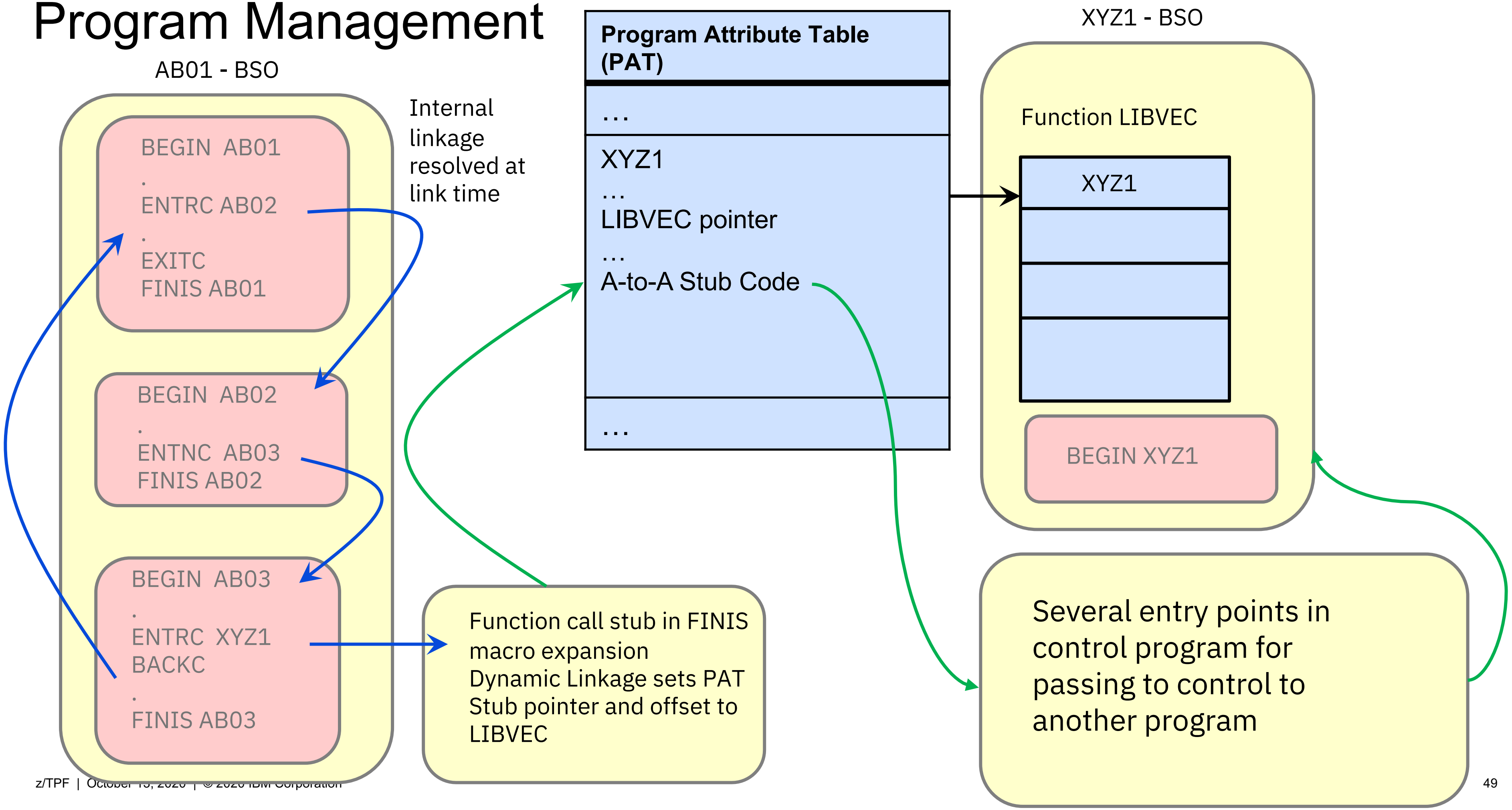
How LIBVECs handle multiple versions loaded via E-type loader



foo {
info();
bar();
}
infoo {
...
bar {
exfoo();
}
Global Offset Table (GOT)
Program Linkage Table (PLT)
Dynamic Linkage sets PAT Stub pointer and offset to LIBVEC
Program Attribute Table (PAT) for Base Program
…
MLIB
…
LIBVEC pointer
…
C-to-C Stub Code
…
Program Attribute Table (PAT) for Loadset NEWPROG
…
MLIB
…
LIBVEC pointer
…
Base Program Function LIBVEC
bar1
fun2
exfoo
.
.
exfoo {}
Loadset NEWPROG Function LIBVEC
bar1
<DUMP>
exfoo
.
.
exfoo {}

Several entry points in control program for passing to control to another program

Program Management



Internal linkage resolved at link time

Function call stub in FINIS macro expansion
Dynamic Linkage sets PAT Stub pointer and offset to LIBVEC

Several entry points in control program for passing to control to another program

Program Management
Enter/Back Assembler Macros

z/TPF programs are invoked through one of the create macros or through one of the following enter macros:

- ENTNC
  - Enter with no return. The calling program does not expect a return of control.
- ENTRC
  - Encountered during entry processing, BACKC returns control to the last program that issued an ENTRC.
- ENTDC
  - Enter and drop previous programs. An ECB-controlled program is called and the Enter-Back macro control information that was saved is reinitialized to remove linkages to all previous programs.
- SWISC TYPE=ENTER
  - Transfer the ECB to another I-stream and drop previous programs. This macro performs the function of ENTDC while transferring the ECB to another I-stream.

C language APIs for each of these macros exist.

Program Management
Commands

ZDMAP
– displays the link map information given a program name or an address in memory

ZDPAT / ZAPAT
– displays or alters the attributes of a given program with option for in memory, on file, or both

ZDPGM / ZAPGM
– displays or alters the contents of a given program with option for in memory, on file, or both

ZDPLT
– display linkage type of a given program, or displays names of programs given linkage type

Program Management
Active Program Detection

Records what programs are actively called / used on a production system

Collected on each processor by CPUID

Some programs have special characteristics that prevent detection
- Shown as “other”

```
Active Program Detection Report Version 1.0
Creation Date 08/14/2017

Active Program Detection Start Times
CPUID, Date
B, 04/27/2017
C, 06/30/2017
D, 08/12/2017

Active Program Detection Status
Program Name, Type, Any, B,   C,   D
ABCD,         BSO,  YES, YES, NO,  NO
ABCE,         BSO,  YES, YES, YES, YES
ABCF,         BSO,  YES, YES, YES, NO
ABCG,         BSO,  NO,  NO,  NO,  NO
ABD0,         DATA, OTHER, OTHER, OTHER
.
.
.
```

Program Management
Common Deployment

Common Deployment - make Deployment Descriptors available for use.

Deployment Descriptor - an XML file that describes the capabilities and options for a specific function or component.
This XML file must be deployed on the z/TPF system before it can be used. When a deployment descriptor is deployed, the XML file is parsed and the results are placed in memory. This process reduces processor usage for subsequent uses of this information.

Examples: REST, DFDL, MongoDB, ADBI

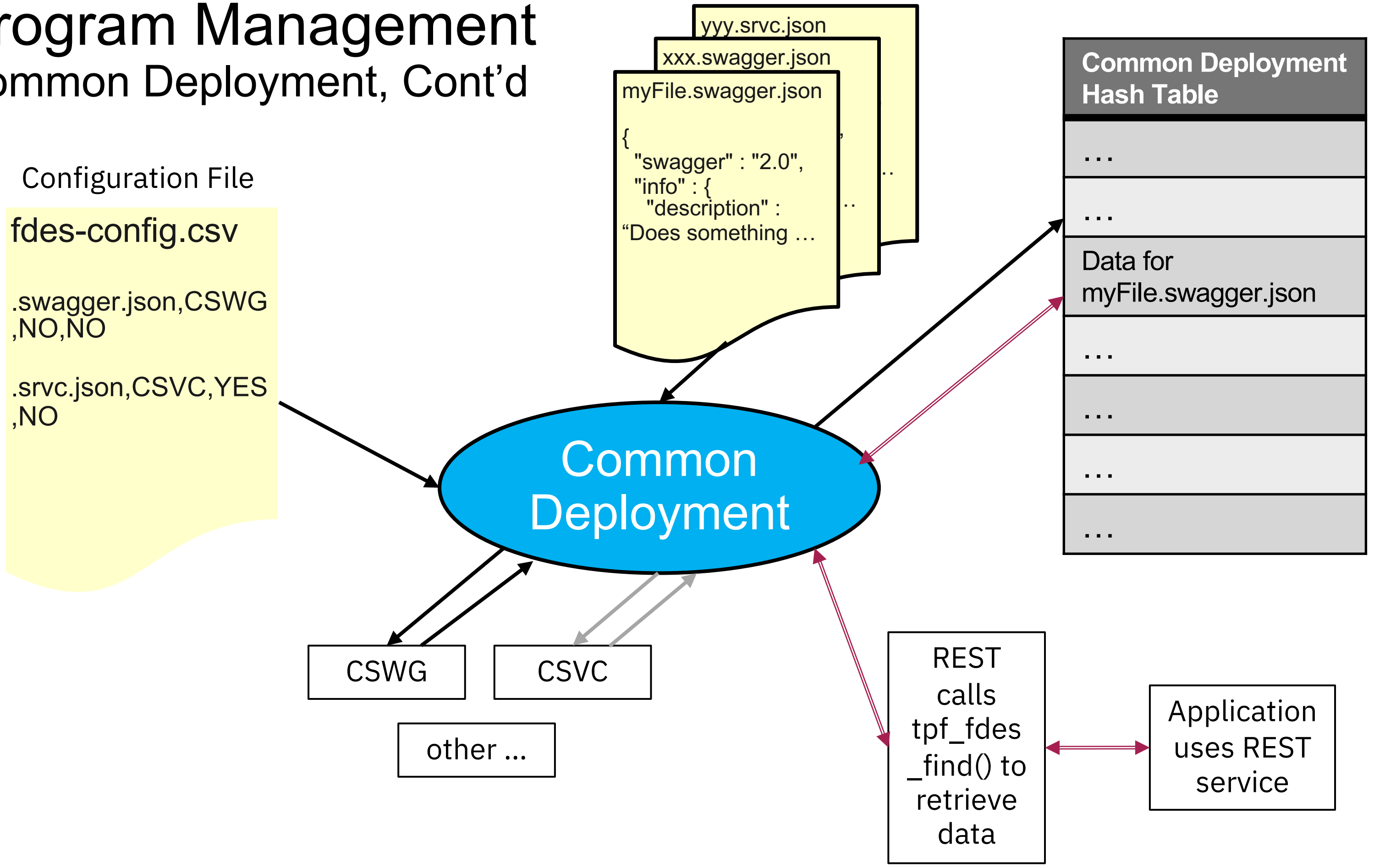
Parsing the descriptor file and creating the memory structure are tasks that are unique to each function.

Common Deployment performs common tasks:

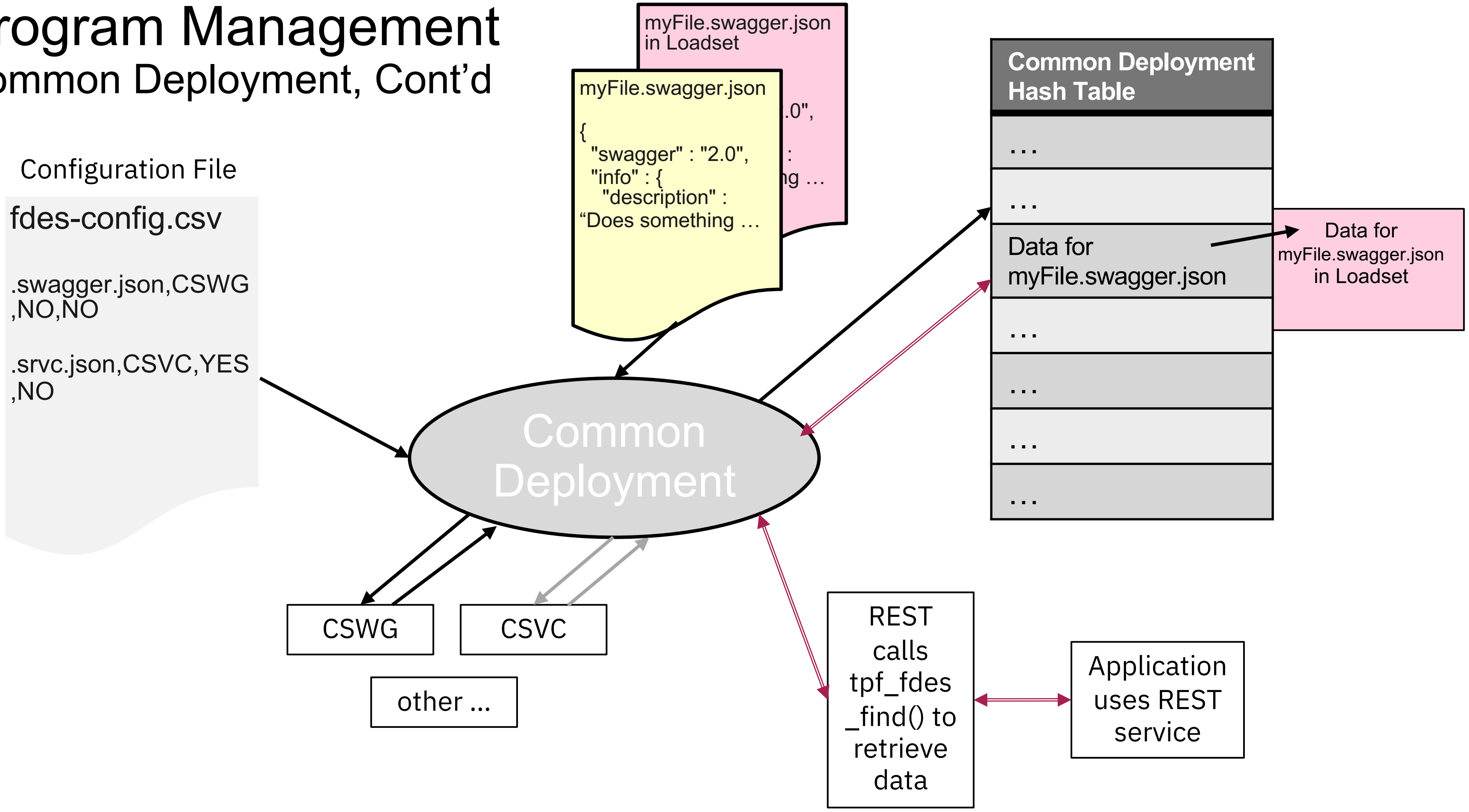
- Maintains the status of which files are deployed
- Deploys the file again after an IPL
- Provides a mechanism to find the in-memory structure
- Handles any changes to the file; that is, handles a file in a loadset that was activated or deactivated.
- During restart, common deployment initializes the memory structures and calls function-unique processing for each deployment descriptor. Additionally, if a file is deployed, the in-memory structure is marked as available for use.

Program Management

Common Deployment, Cont'd



Program Management
Common Deployment, Cont'd



Program Management

Common Deployment – Deleting a descriptor file

Files under /sys/tpf_pbfiles (including common deployment descriptor files) should be deleted through the loaders.

– If you delete a descriptor file using ‘zfile rm’ the file is deleted, but memory structures remain in the common deployment hash table.
– It can also prevent you from loading another file that defines some of the same configuration data.

Instead, add an @@DELETE statement to your loader input file.

– You can delete a file as part of a loadset.
– The file will appear as missing if a new ECB tries to access it, while old versions of the file continue to be available to ECBs created before the loadset was activated.
– Common deployment structures will also appear as missing to new ECBs, while they continue to be available to older ECBs.

Thank You!

Questions or Comments?

BACKUP

Trademarks