Session 17090
SLIP Trap Bootcamp
Part 1

z/OS Systems Programming & zNextGen – March 3rd, 2015

Patty Little plittle@us.ibm.com
IBM Poughkeepsie

SHARE
Technology · Connections · Results

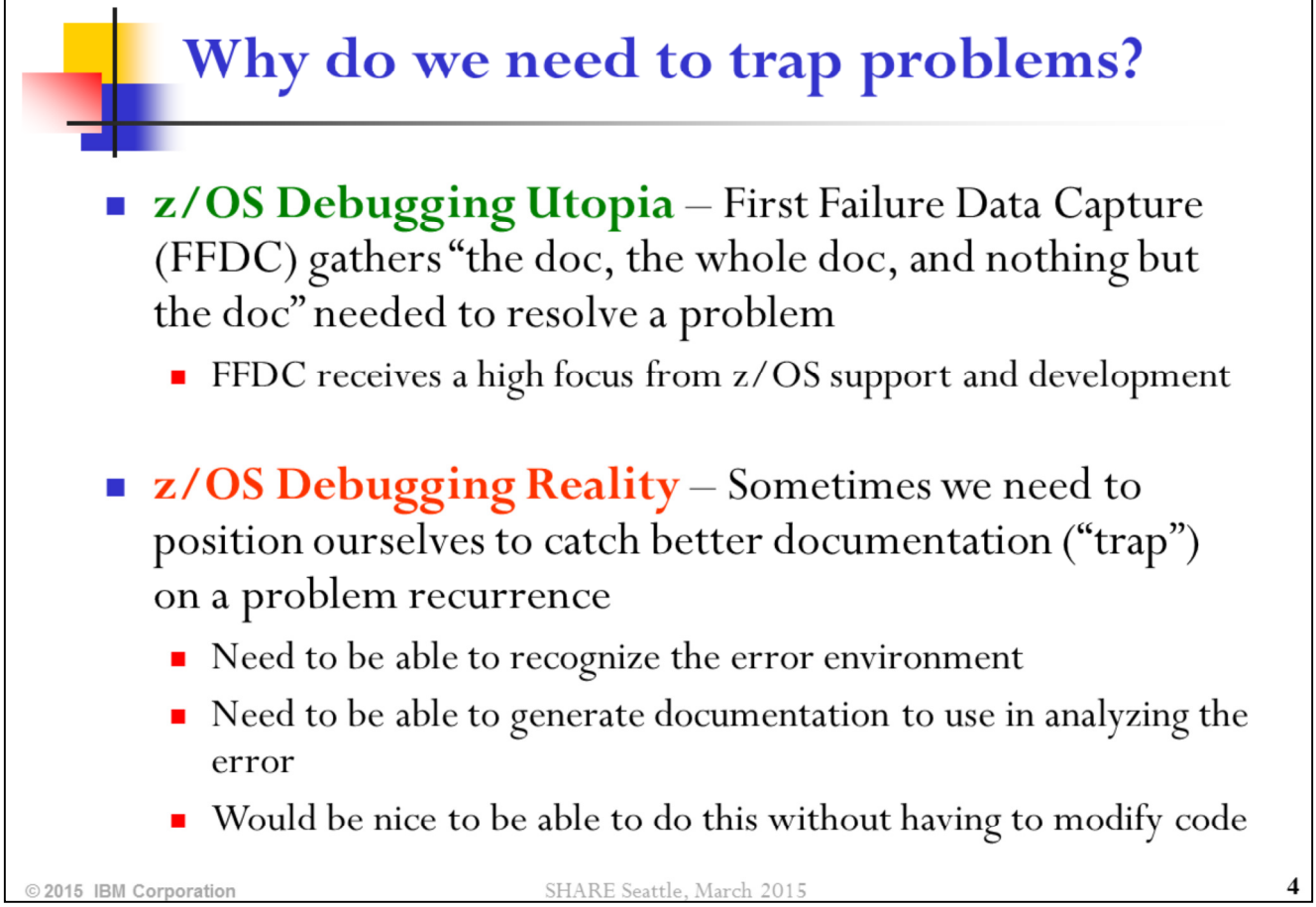
Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

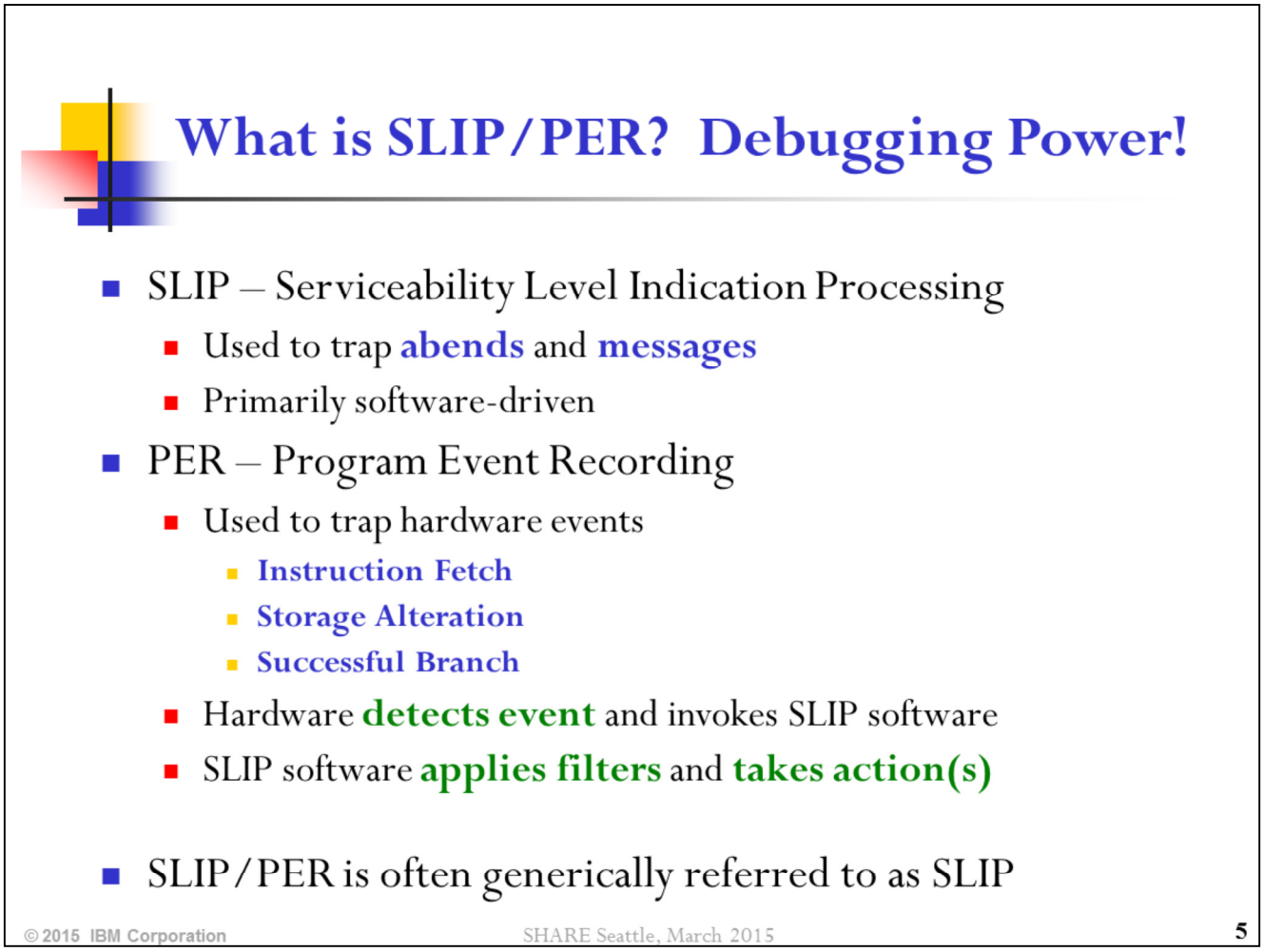
- MVS
- OS/390®
- z/Architecture®
- z/OS®

* Registered trademarks of IBM Corporation

Table of Contents



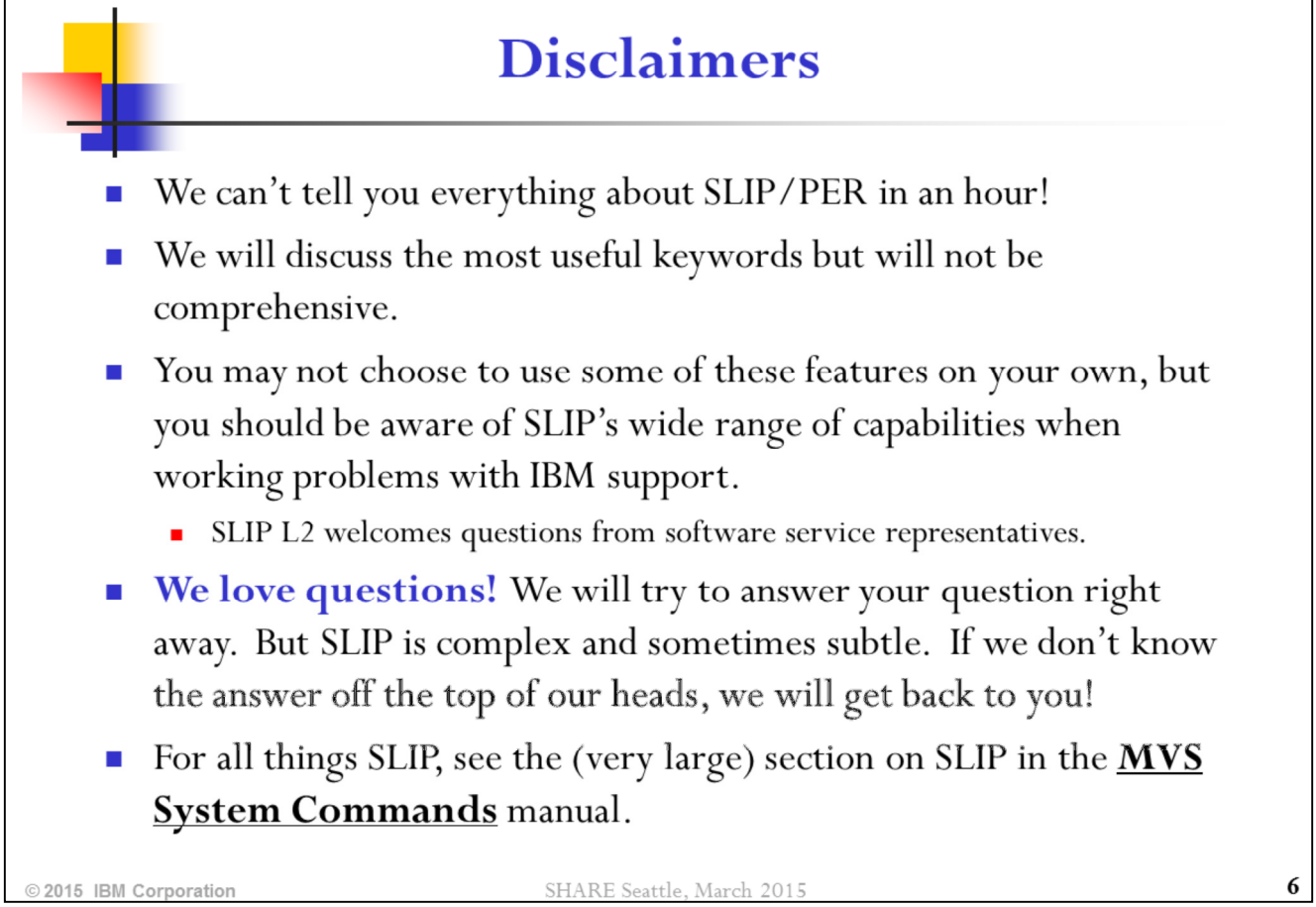
In an ideal debugging world, we wouldn't need SLIP because the system would always generate exactly the documentation needed for problem diagnosis. Reality is that sometimes a problem cannot be resolved from the documentation generated by the first occurrence. In this case, it is sometimes necessary to wait for a problem recurrence for an opportunity to gather better documentation. When this is necessary, SLIP/PER (SLIP for short) can often help. SLIP allows the debugger to very precisely define the error environment for which more documentation is needed. It also allows the debugger to specify various types of documentation to gather when this environment is detected, and allows for tailoring of this documentation to ensure that the most relevant data is included in what is produced. On top of all this, we will see that SLIP processing offers the ability to monitor for a chain of conditions, or even to repair corrupted storage.



SLIP/PER is the premiere tool for trapping programming events in z/OS. It can trap software events such as abends or messages, or it can trap hardware events such as the execution (fetching) of an instruction or the alteration of an area of storage. Its power is in its flexibility both in filtering on event environments and in generating documentation.

The monitoring of hardware events is accomplished by the PER part of SLIP/PER processing. When a monitored hardware event occurs, hardware detects this and signals SLIP software. SLIP software handles the additional filtering and taking of action as appropriate.

SLIP software also provides filtering for the software-generated events, and will take action as appropriate.



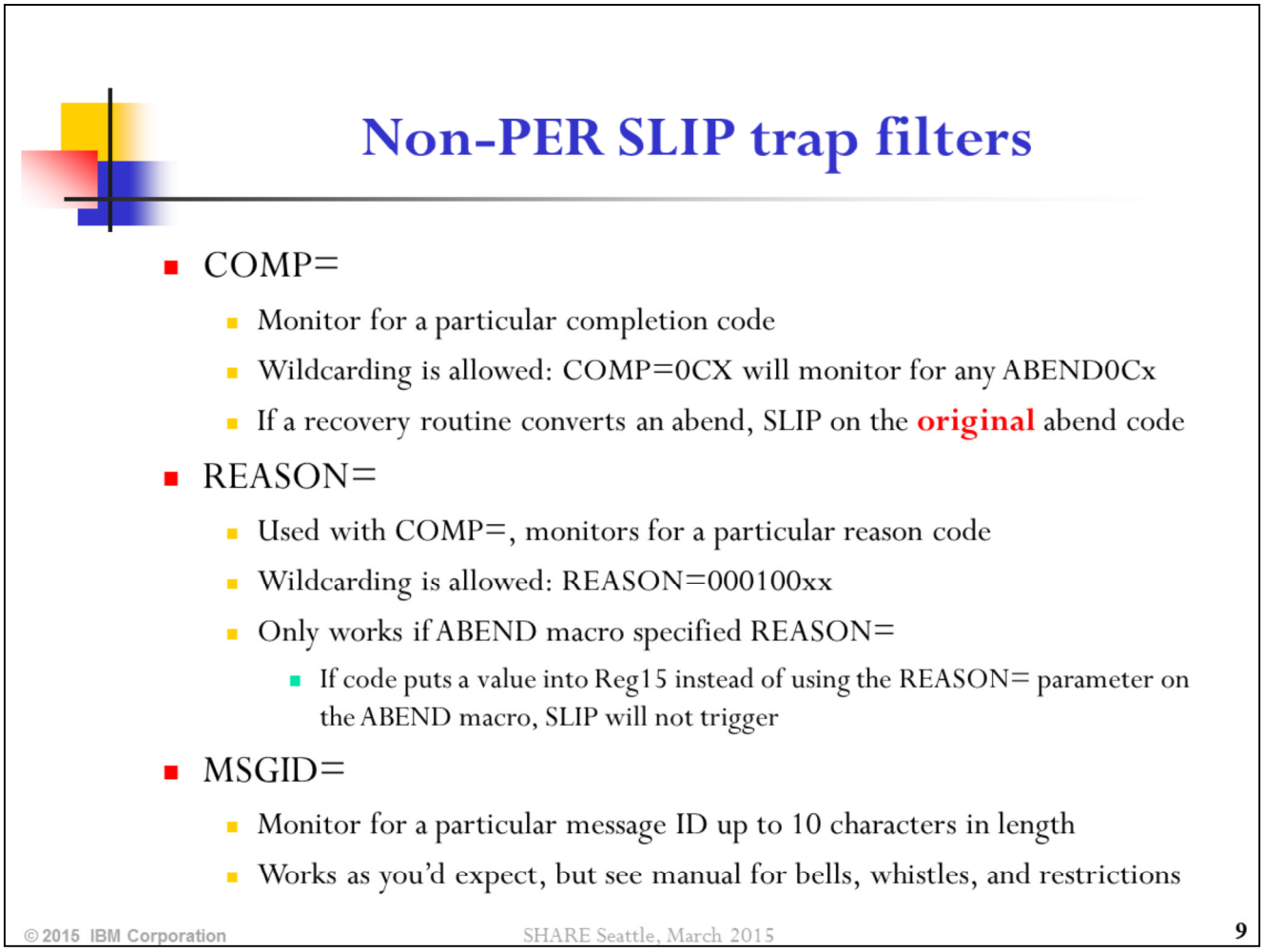
Due to bullet #1, we will not be covering the REMOTE parameter.

Non-PER SLIP traps

Non-PER SLIP traps

- Non-PER SLIP traps use only software to monitor for error events
- Can monitor for abend codes (aka 'completion codes') and messages
- Can have multiple non-PER SLIP traps active at the same time, allowing one to trap for many different software events simultaneously

Typically there are many completion code SLIPs active on a system at any given time. Some are in place to collect dumps. Others are actually in place to suppress unwanted dumps. There is no performance overhead associated with SLIP traps that monitor for abend codes and messages.

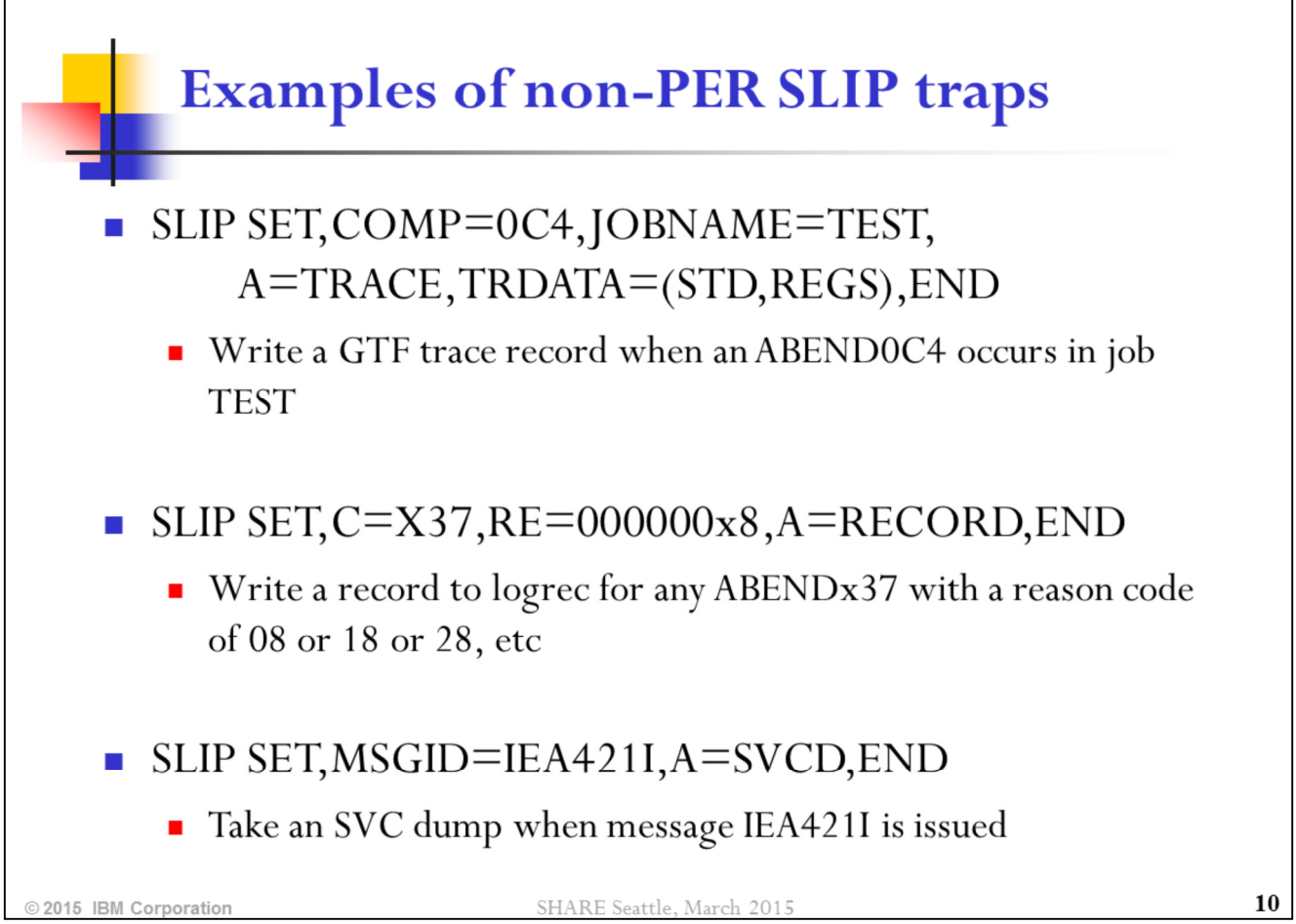


COMP can be shortened to "C=". REASON can be shortened to "RE=". The specified REASON code can be anywhere from 1 to 8 digits.

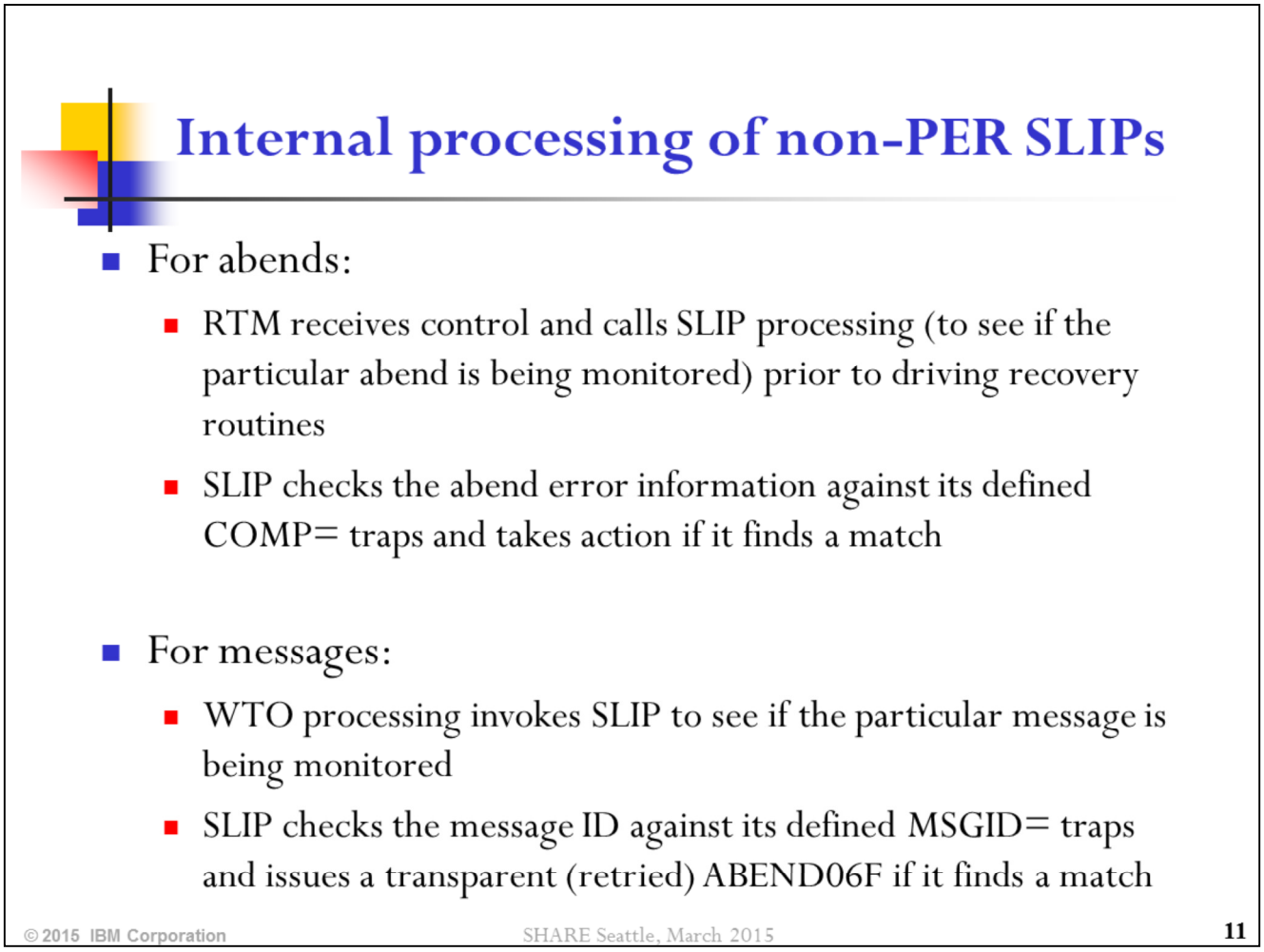
MSGID SLIPs don't work on certain types of messages, most notably branch entry WTOs with the NLCKS, LOADWAIT, or SYNCH=YES

parameter specified. MSGID only is applicable for WTO messages to SYSLOG, not messages to other logs. MSGID does accept message IDs that include blanks or special characters if you put single quotes around the message ID,
e.g. MSGID='%TEST MSG'.

Using quotes also provides a loose form of wildcarding. See the SLIP section in the System Commands manual for more information.



We'll talk about the various SLIP filters and action keywords later.



The fact that RTM drives SLIP before it drives recovery is important in the case of converted abends.

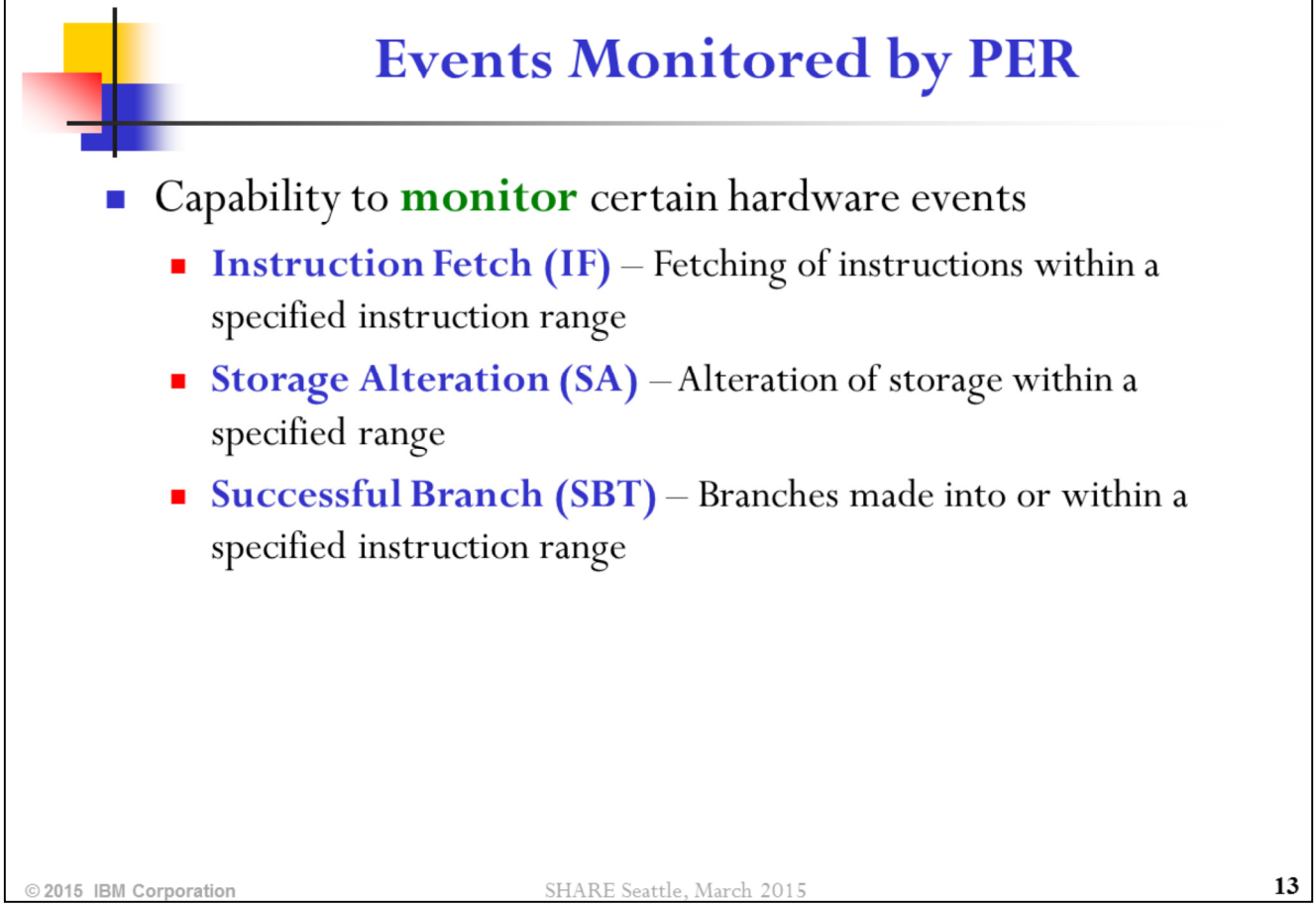
A converted abend is when a recovery routine takes the original abend code and converts it to one that is more meaningful to the component that suffered the error. For example, EXCP recovery will sometimes convert an ABEND0C4 into an ABENDA00. RTM saves the original abend completion code in a field in the SDWA: SDWASABC at +X'90' into the SDWARC1 (first recordable extension) which can be found in a logrec record of the abend.

This means that if you are SLIP-ing on an abend code that gets converted, you must SLIP on the original abend code, since SLIP does not know about the converted one.

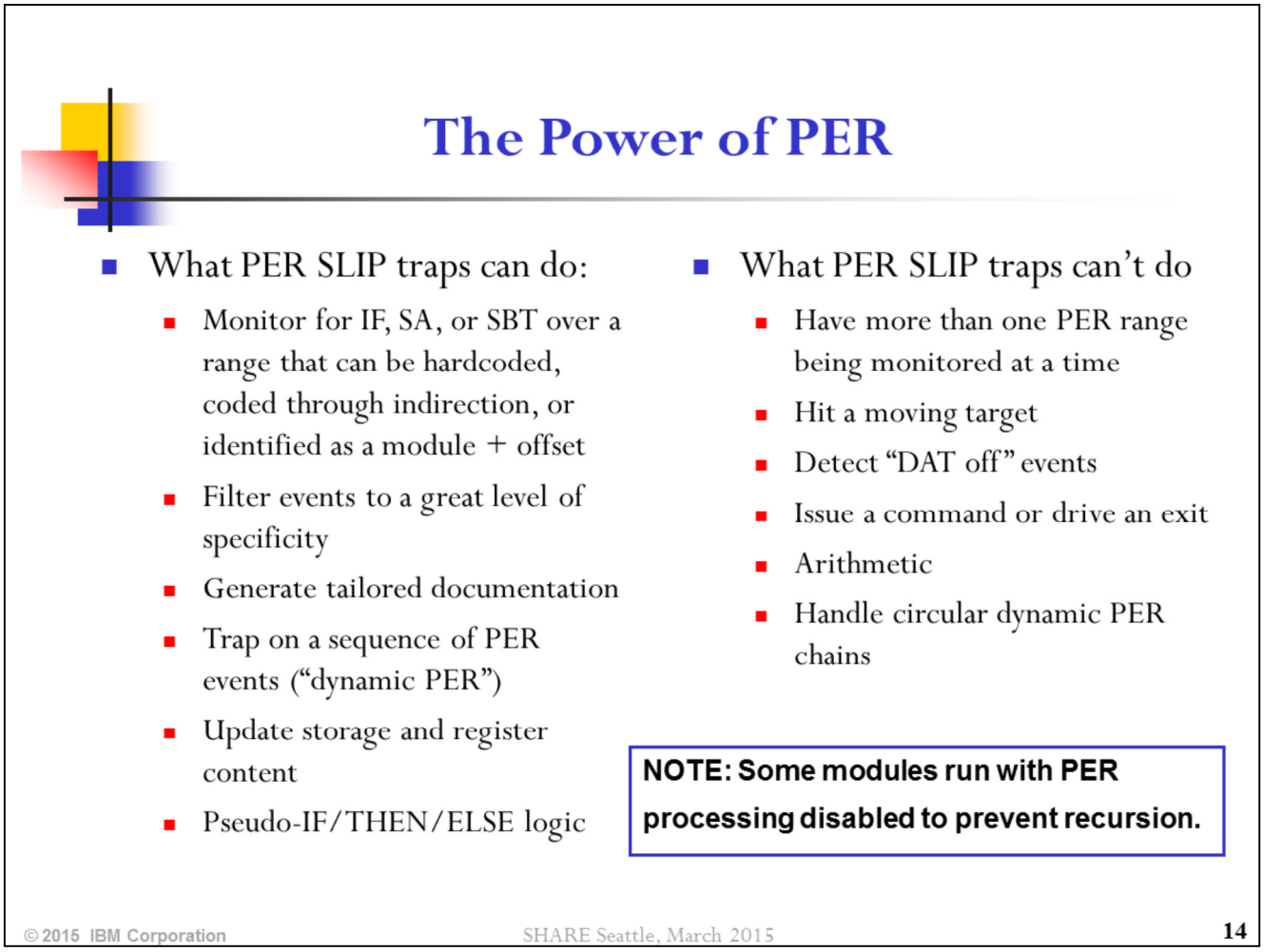
When SLIP finds a match for a message that it is monitoring, SLIP module IEAVTSMG will actually issue an ABEND06F "under the covers". This abend is transparent in that it gets retried without every being recorded or taking a dump. Its purpose is to force abend entry into RTM so that RTM's existing logic to handle completion code SLIPs can be used to

handle the MSGID case as well. Once SLIP processing completes, RTM will get control back and then drive a recovery routine that will retry the ABEND06F back into WTO processing who will return control to the caller.

PER SLIP traps



There is actually a 4th event that is monitored by PER as well: SAS catches stores done by the STURA instruction as well as other alterations.



SLIP/PER's robust filtering capabilities allow for the trapping of very specific problem scenarios.

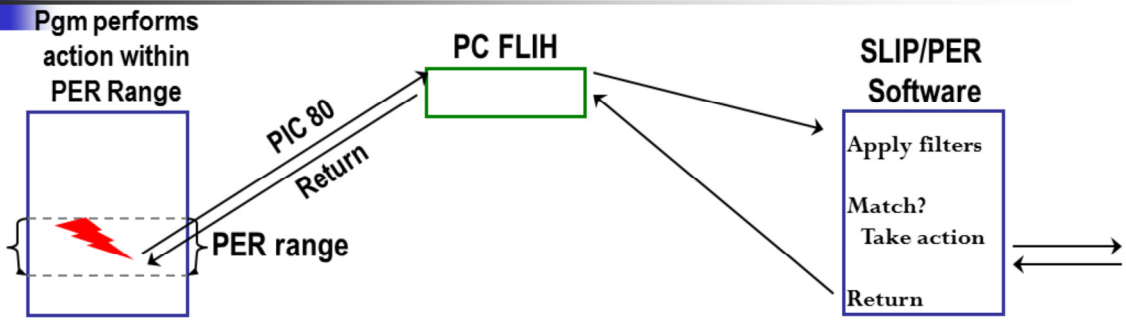
SLIP/PER provides a wide variety of actions that can be taken. SLIP provides parameters for tailoring of the documentation that it is asked to produce.

Dynamic PER is supported through the A=TARGETID parameter.

Updating of register and storage content is supported through the A=REFBEFOR/REFAFTER parameter.

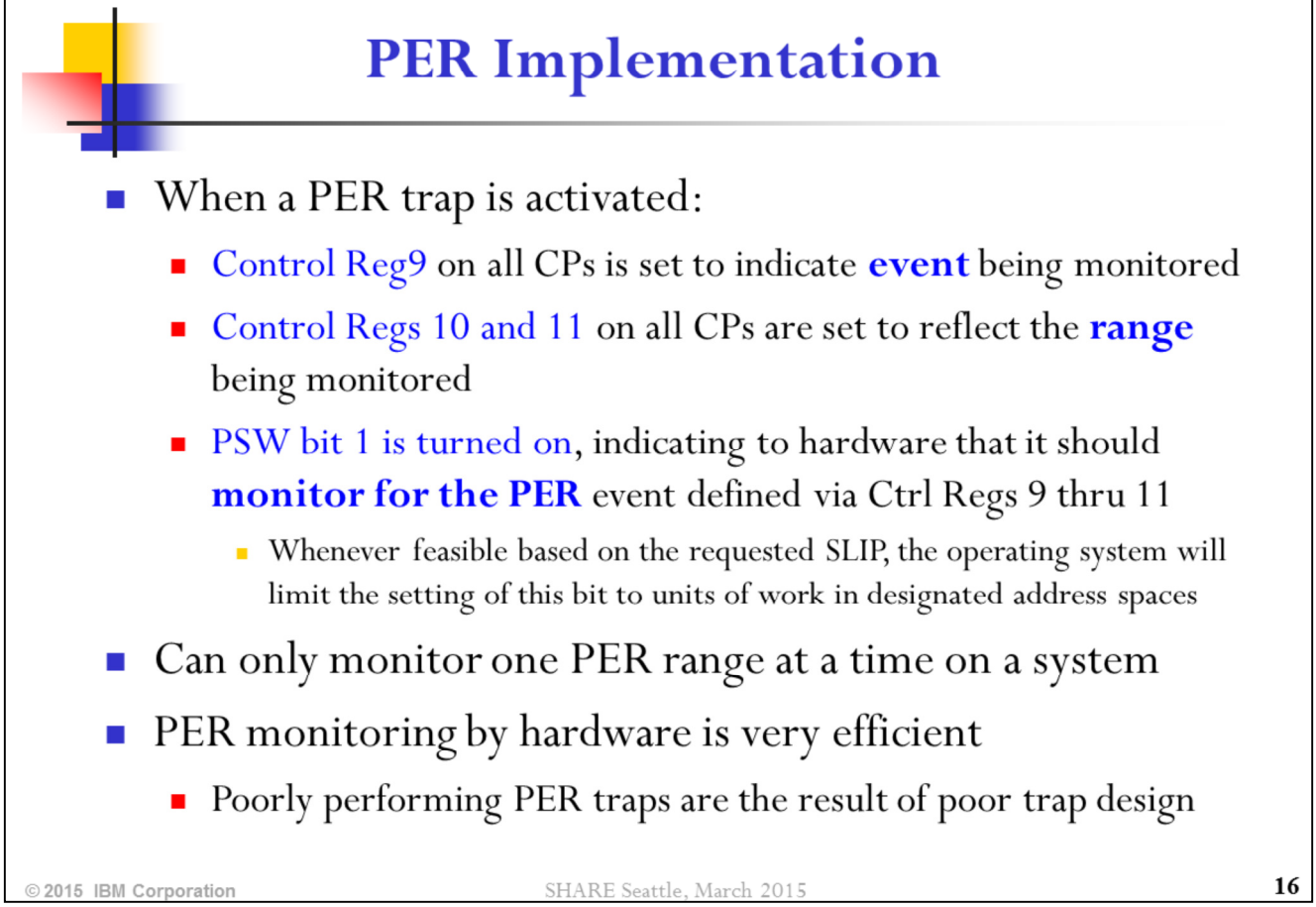
Pseudo-IF/THEN/ELSE logic is supported through the A=SUBTRAP parameter.

PER Processing

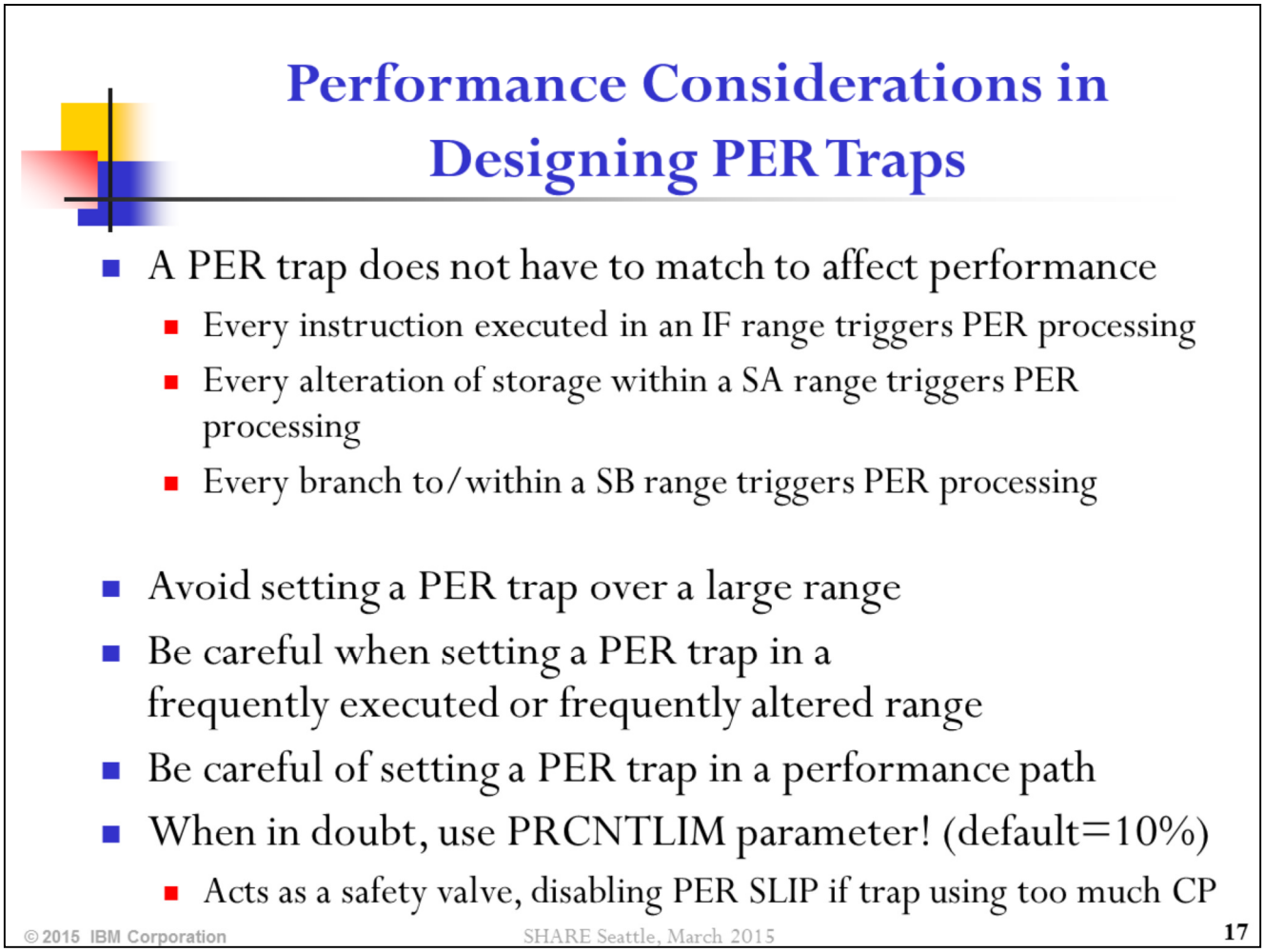


- Hardware detects a monitored event occurring and issues a program interrupt (PIC 80)
- The operating system's Program Check First Level Interrupt Handler (PC FLIH) receives control and routes control to SLIP/PER software
- SLIP/PER software checks filters of active PER trap to see if environment at the time of interrupt meets all criteria
  - YES – Takes requested action(s)
    - Usually operating system returns control to point of PER interrupt
  - NO – Operating system returns control to point of PER interrupt

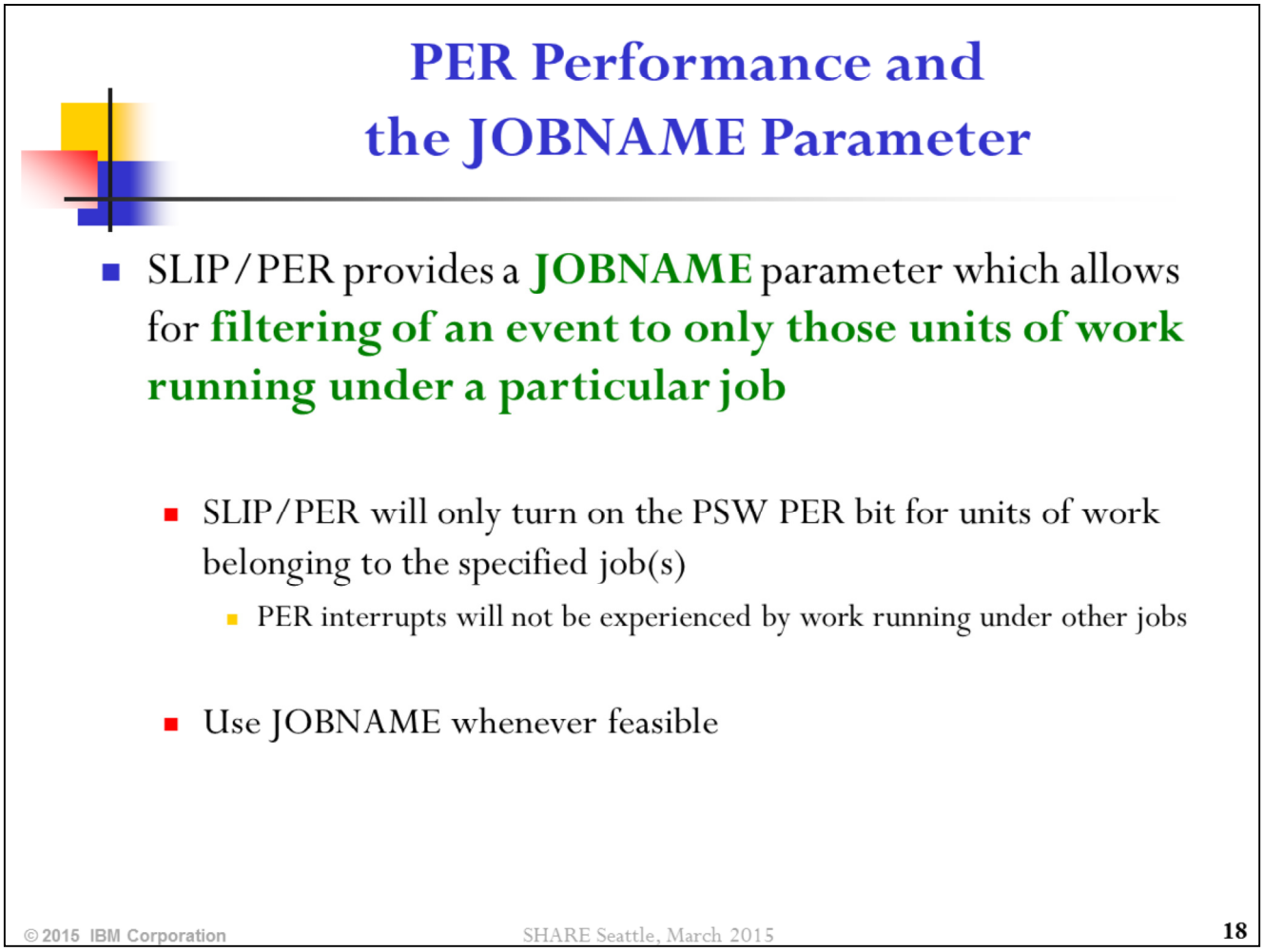
After most SLIP actions, control will be returned to the program that experienced the PER interrupt. An exception is A=RECOVERY which instead targets the interrupted unit of work with an ABEND06F.



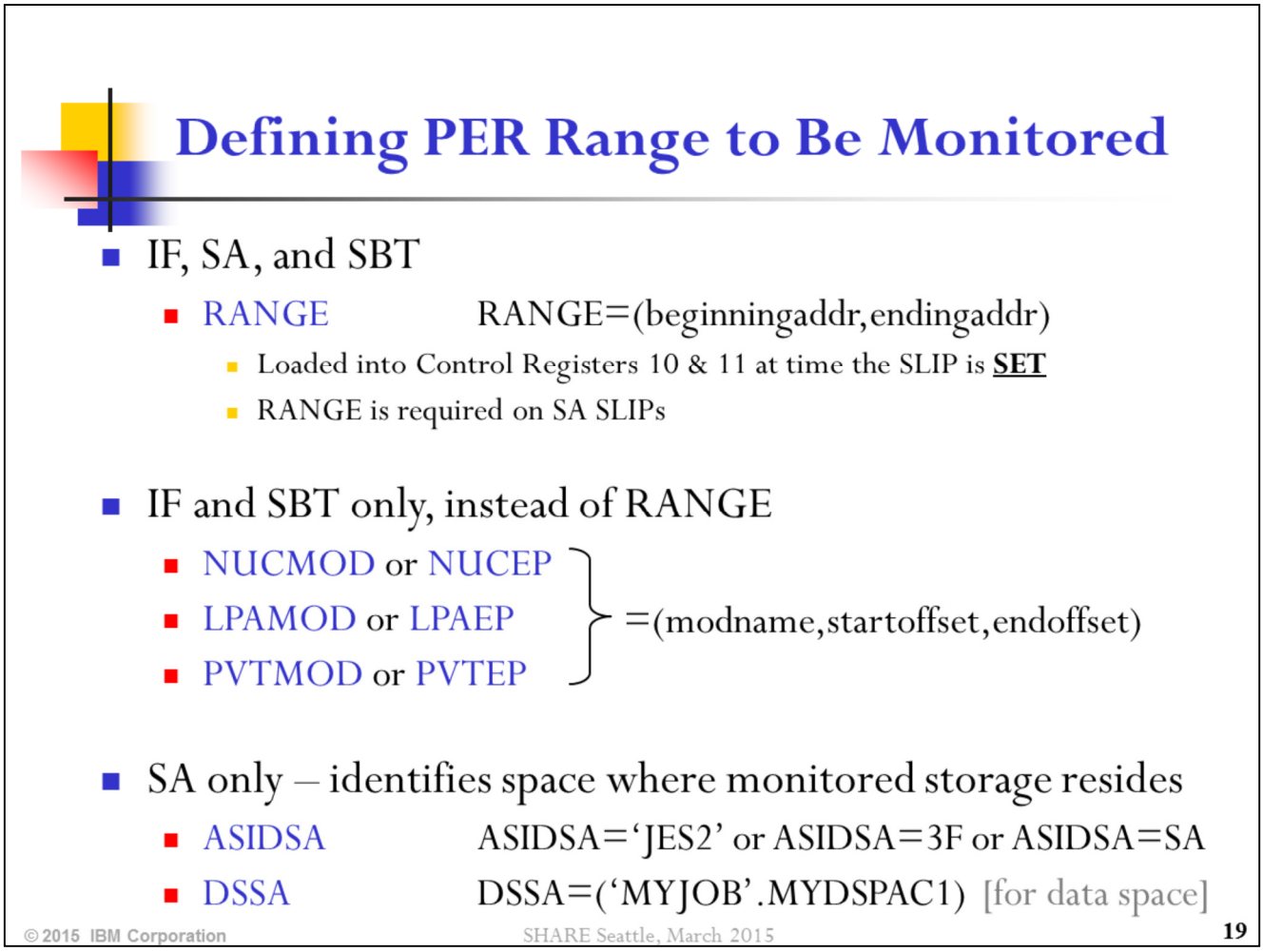
PER processing can be enabled and disabled under a running unit of work simply by turning on and off the PER bit in the PSW. Some operating system code turns off this bit while it is running in order to prevent recursion scenarios. However, most operating system code runs enabled for PER.



The PRCNTLIM (Percent Limit) parameter limits how much CPU a PER trap is allowed to consume on the system. If SLIP/PER CPU usage exceeds the indicated percentage, the offending PER trap is disabled.



When using PVTMOD, it is advisable to use JOBNAME=, and also MODE=HOME whenever feasible. If you are trying to catch an event that is occurring under a job in a cross memory environment, MODE=HOME may not be feasible. However, be sensitive to possible system performance issues that could result. Code a low PRCNTLIM (1-2%) to be on the safe side.



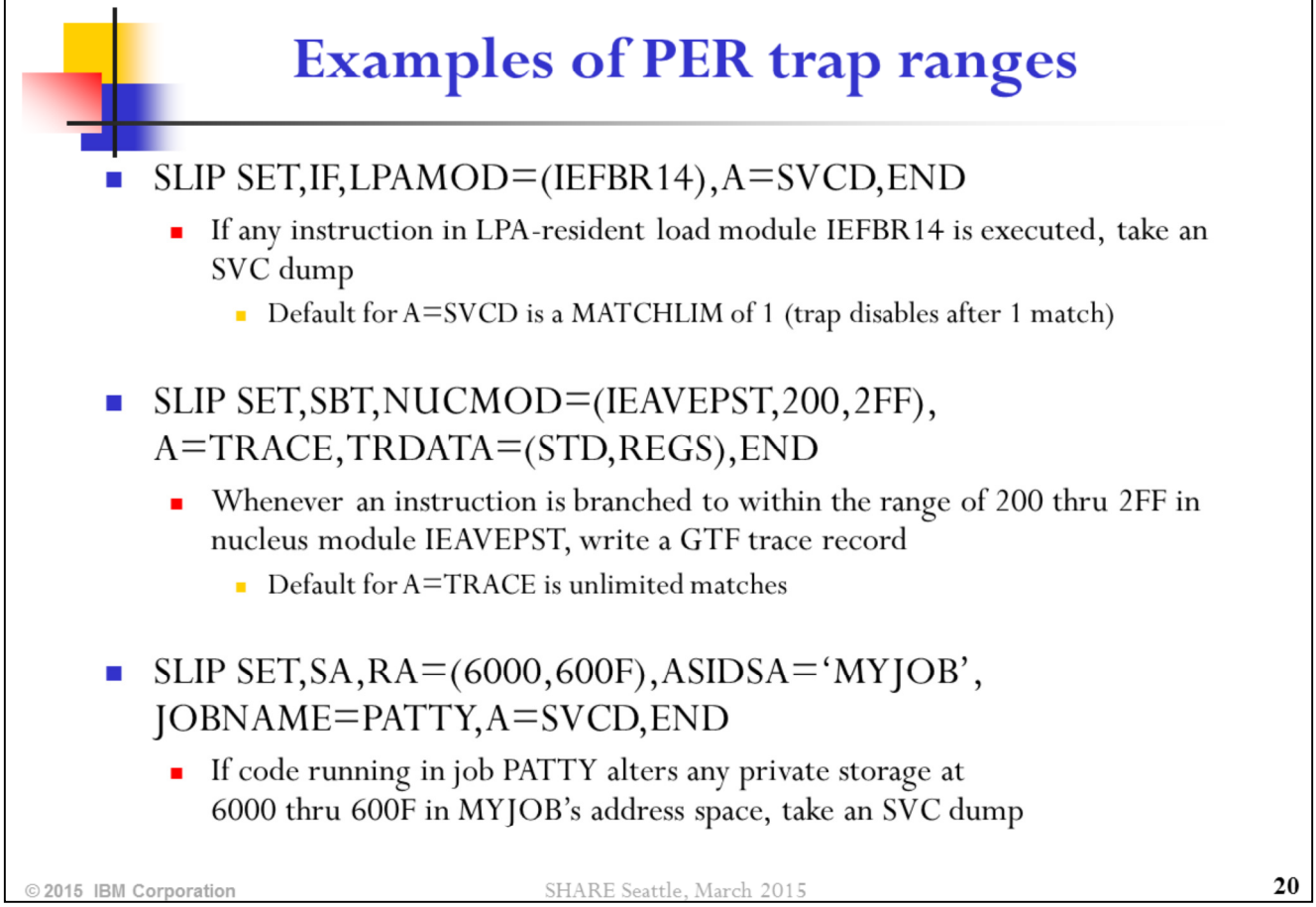
RANGE can be used to define the range to be monitored for SB, IF, or SA. SLIP permits indirection to be specified in the RANGE parameter. This is resolved at the time the SLIP is set, and the resulting address is placed into the control registers used by SLIP. Note that the ending address can be omitted, in which case only the single byte indicated by the beginning address will be monitored.

NUCMOD & NUCEP, LPAMOD & LPAEP, PVTMOD & PVTEP can be used in a SBT or IF PER trap instead of RANGE. For example, one can specify:

NUCMOD=(IEAVEPST,40,4F)

on an IF PER trap to monitor the fetching of instructions in the POST code between offset +40 and offset +4F. This is much more flexible than having to hardcode the addresses in the RANGE parameter. Note that specification of a starting offset and ending offset are optional. If neither is specified, then the entire module is monitored. If ending offset is omitted, then only the instruction at the starting offset is monitored.

Use ASIDSA or DSSA on every SA PER trap. If monitoring a private storage range in an address space for alteration, specify the jobname or ASID owning that storage on ASIDSA. If monitoring global storage, specifiy ASIDSA=SA. If monitoring data space storage, use DSSA.

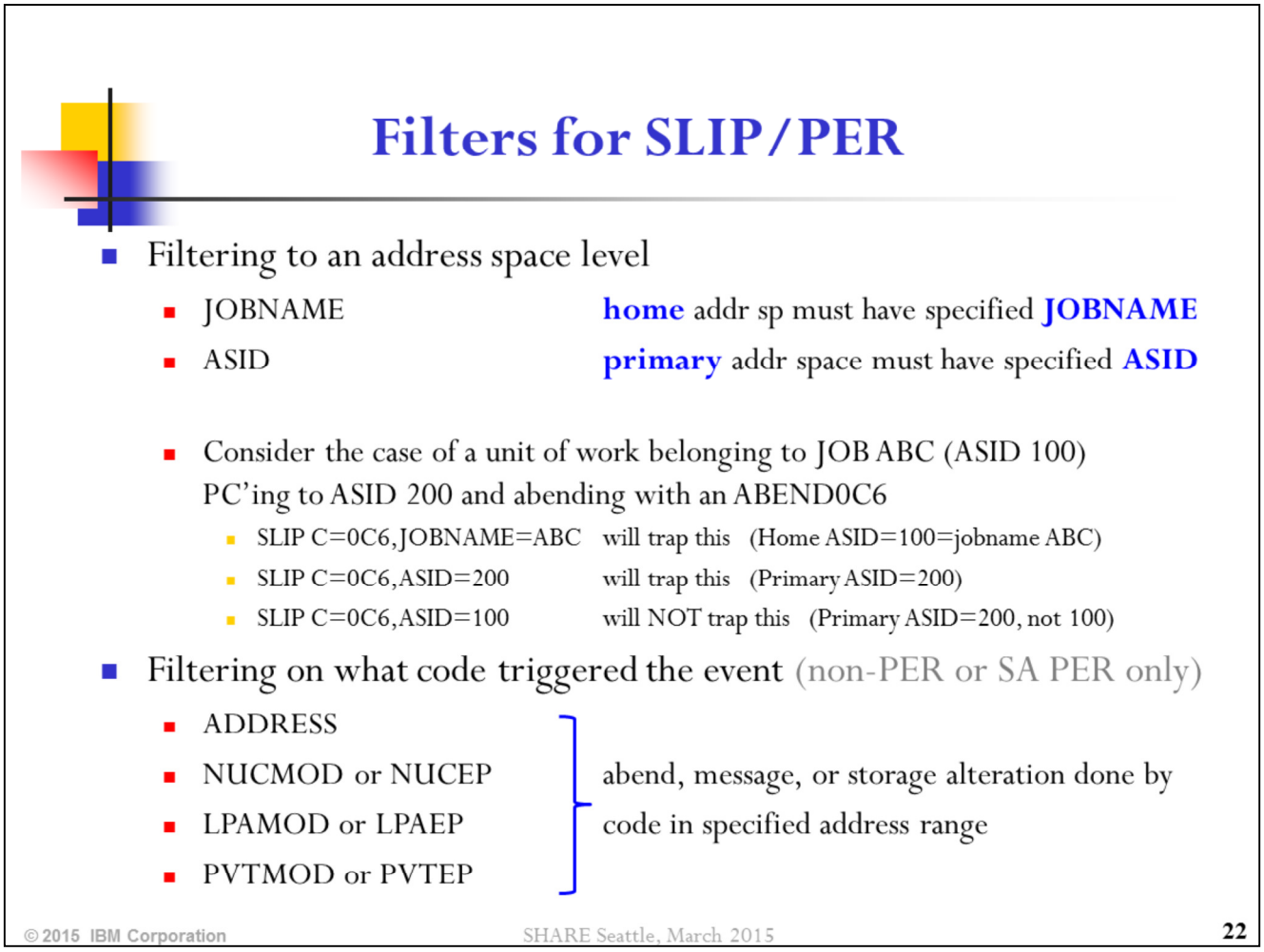


Whenever possible, use NUCMOD/NUCEP, LPAMOD/LPAEP, or PVTMOD/PVTEP to define the range for an IF or SBT PER trap. This makes the trap easier and more versatile. Sometimes modules will change locations in storage, especially across an IPL, and a hardcoded RANGE value may become incorrect.

The RANGE parameter must be used for SA SLIPs to define the range of storage to be monitored for alteration. If NUCMOD/NUCEP, LPAMOD/LPAEP, or PVTMOD/PVTEP is used on a SA SLIP trap, this does not define the range to be altered, but rather it acts as a filter to define who SLIP is wanting to catch altering the specified range.

Note that the ASIDSA parameter defines the space where that range of storage lives, and JOBNAME acts as a filter for who SLIP wants to catch altering the storage.

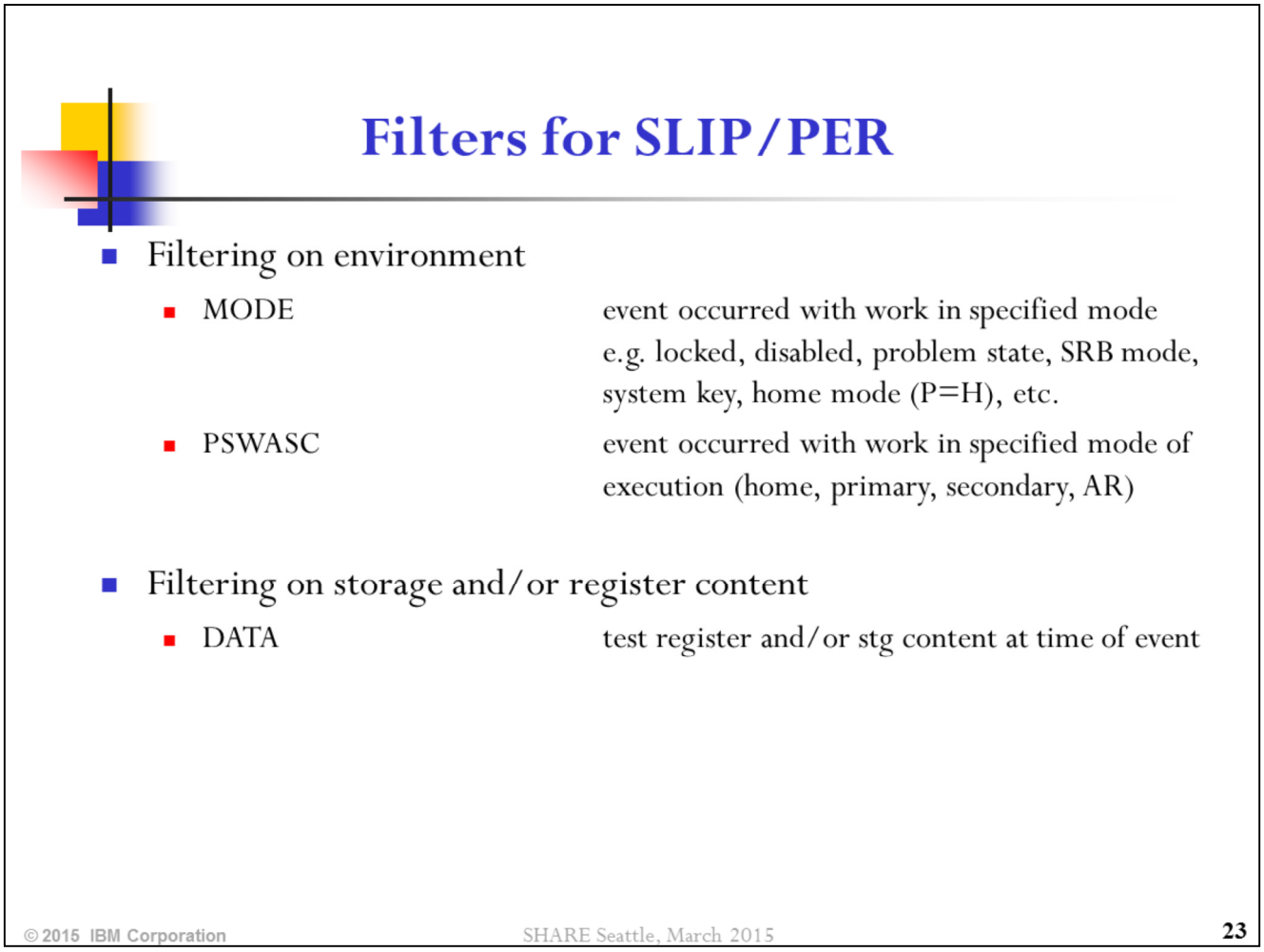
Filters for SLIP traps



Filtering is done primarily by software, although some JOBNAME filtering occurs at the hardware level. Software gets control as a result of the PER interrupt and examines the defined traps in LIFO order to determine what filters to apply.

By definition, the HOME address space is the space where a unit of work lives, where it starts off life. PRIMARY address space is the space where a unit of work is presently executing, fetching instructions. When a program does a space-switching PC, this causes the unit of work's primary address space to change. However, its home address space never changes.

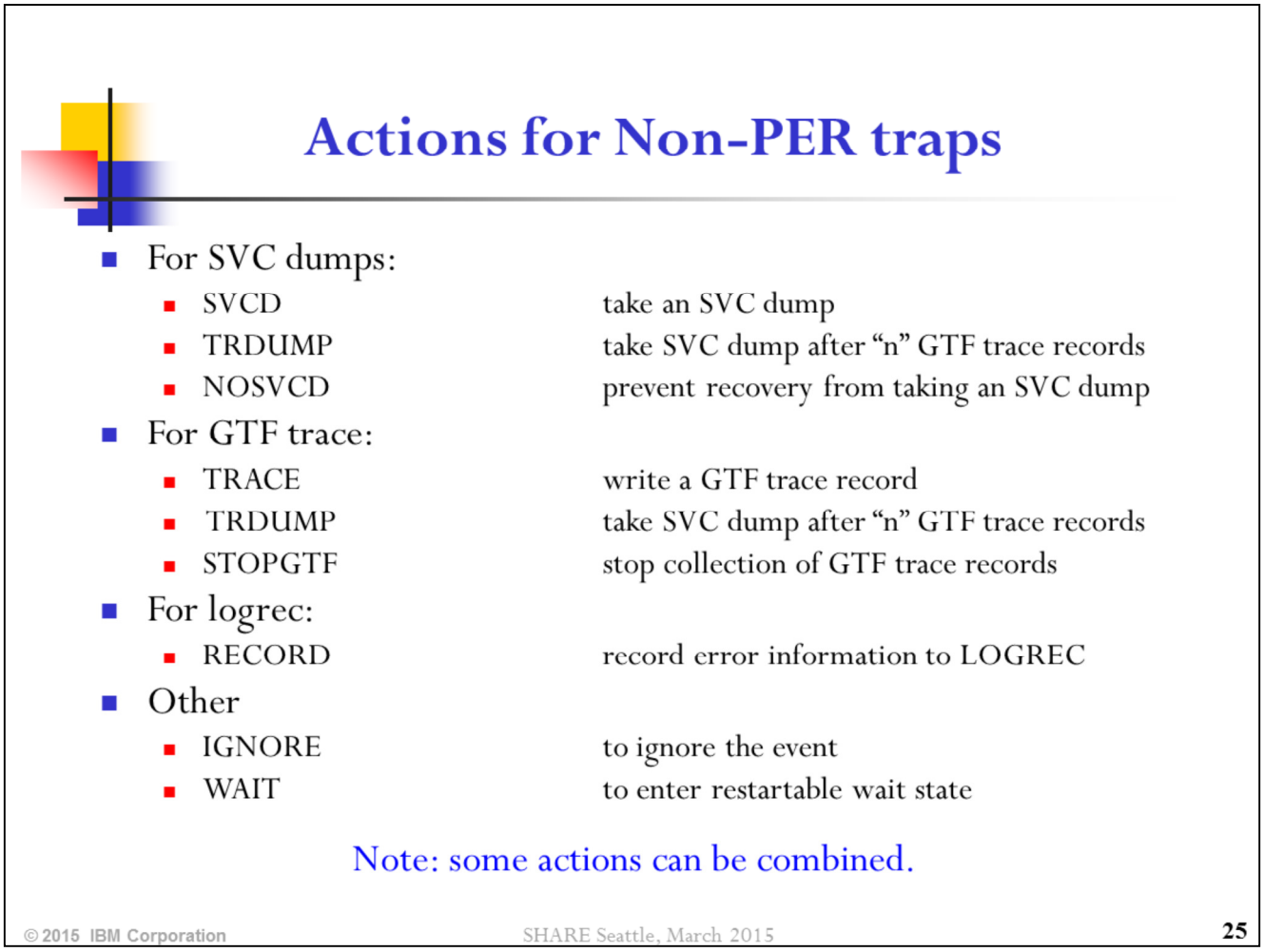
Note that for non-PER traps and for SA PER traps, the NUCMOD/NUCEP, LPAMOD/LPAEP, and PVTMOD/PVTEP parameters act as filters whereas for IF and SBT PER traps, these parameters define the range to be monitored by hardware.



After JOBNAME/ASID, the DATA parameter is probably the most commonly used FILTER. It provides a large degree of filtering capability, including the ability to perform bit checks and/or byte checks in registers or in storage. Multiple checks can be made as part of the DATA filtering process.

The MODE and PSWASC are not heavily used (with the exception of MODE=HOME on PVTMOD/PVTEP SLIPs). However, they offer a variety of options for filtering the event being trapped to certain environmental conditions.

SLIP Actions

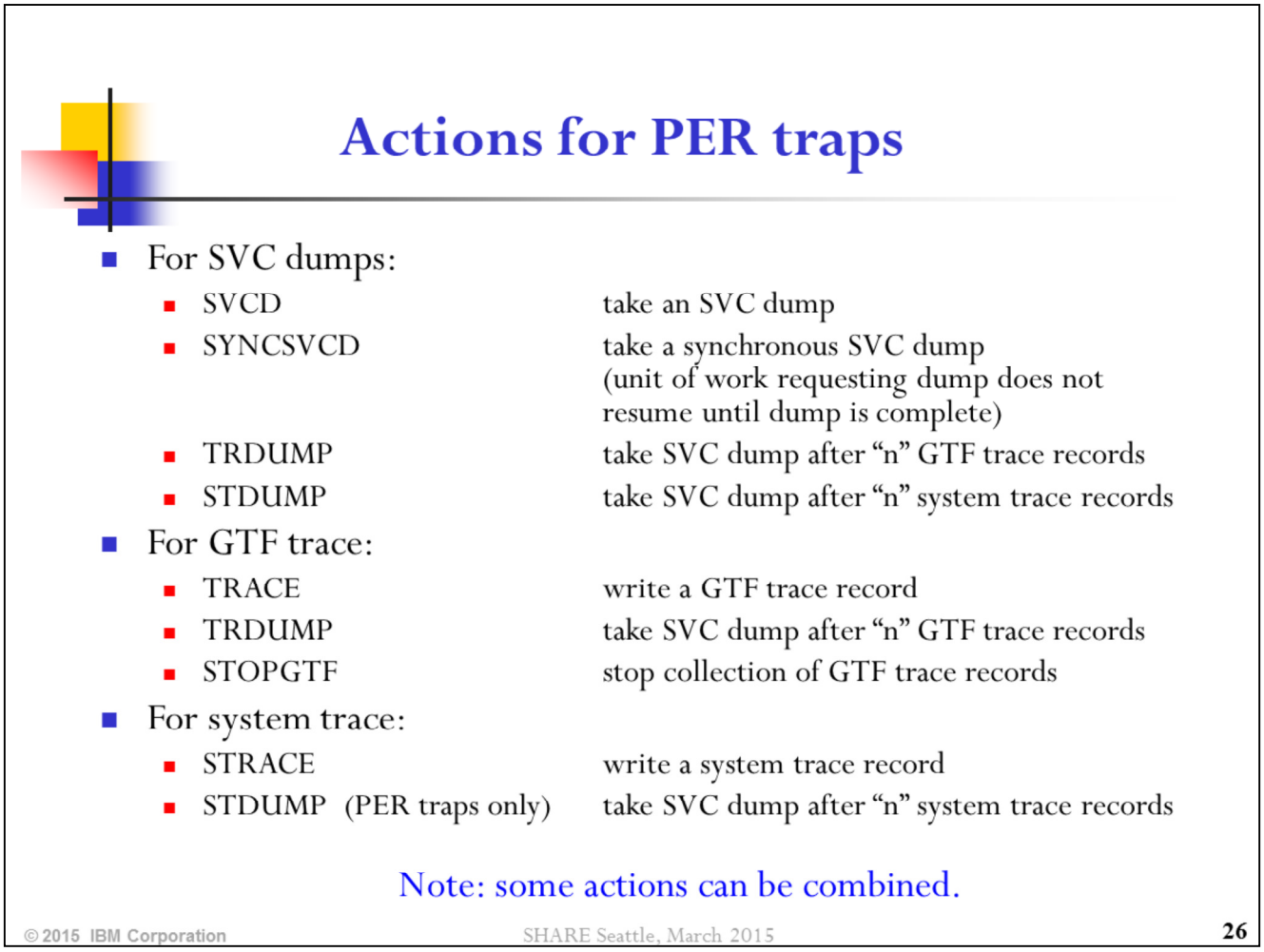


SVCD takes an asynchronous (scheduled) SVC dump.

For a SLIP defined to disable after N matches (via the MATCHLIM parameter), TRDUMP writes a GTF trace record for all N matches, and also takes an SVC dump on the Nth match. STOPGTF will stop GTF trace.

Using A=RECORD is way to force an error to be recorded to LOGREC. (By default, errors handled by FRRs are recorded and errors handled by ESTAE-type recovery are not. These defaults can be overridden by the recovery routine.) However, the logrec will only be written if there is some recovery routine established at the time the error occurs.

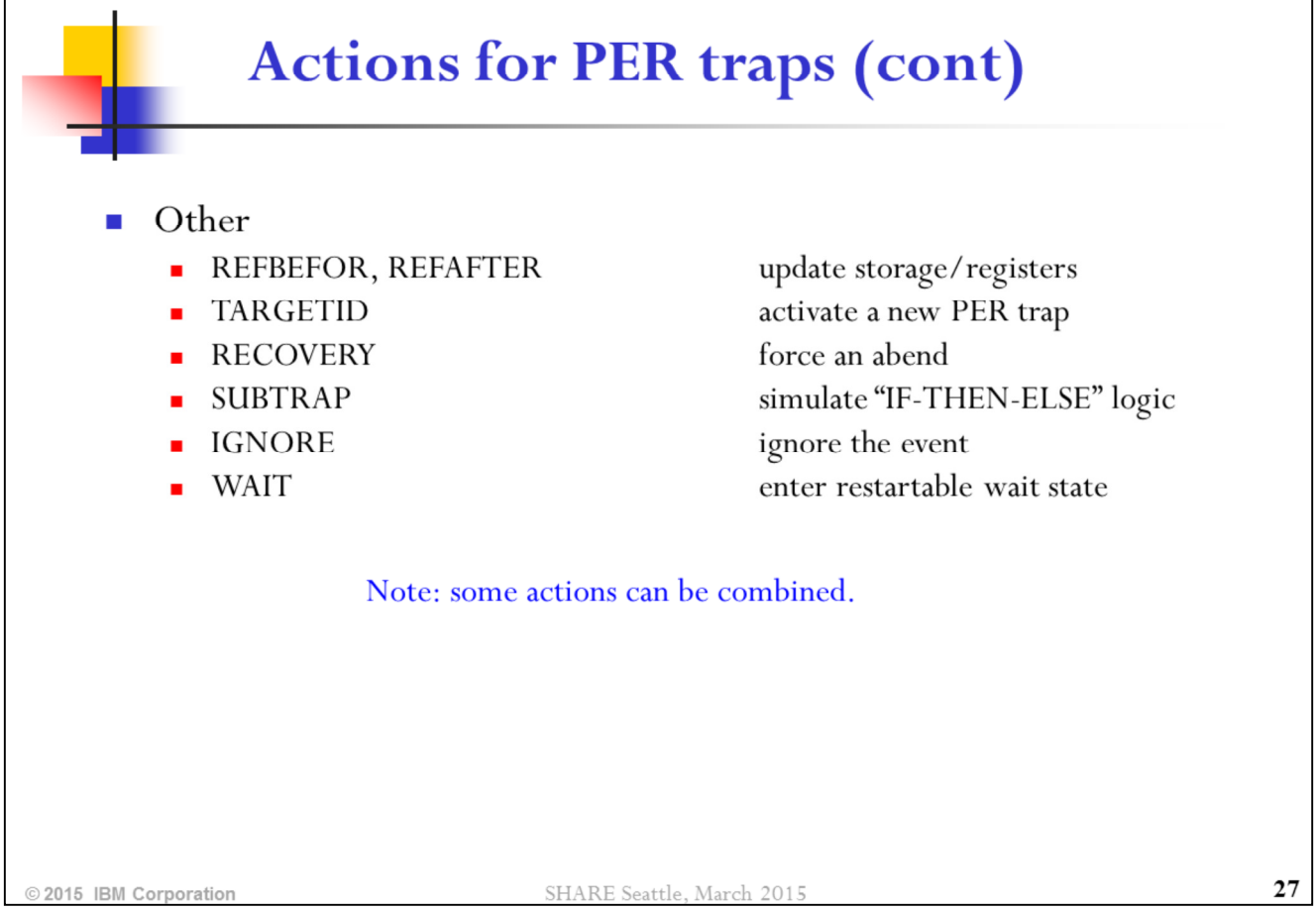
A=IGNORE says "do no further SLIP processing." It does NOT mean "don't take an SVC dump." If you want to suppress a nuisance SVC dump, use A=NOSVCD. There are also SLIP ACTION keywords to suppress SYSABEND, SYSUDUMP, and SYSMDUMPs. NODUMP may be used to suppress all dumps being generated for a particular error condition.)



SVCD takes an asynchronous (scheduled) SVC dump. The unit of work requesting the dump gets control back and is allowed to run after the dump is kicked off (although dump processing will later set all the TCBs in the address space non-dispatchable). In comparison, SYNCSVCD takes a synchronous SVC dump, which means that the unit of work requesting the dump does not continue running until the dump capture completes. A synchronous SVC dump can only be taken if the PER event occurs in an Enabled Unlocked Task (EUT) environment. There are other restrictions as well. Note that if A=SYNCSVCD is requested on a PER trap and the environment at the time the trap matches is invalid for a synchronous SVC dump, then a regular (scheduled) SVC dump will be taken instead.

For a SLIP defined to disable after N matches (via the MATCHLIM parameter), TRDUMP writes a GTF trace record for all N matches, and also takes an SVC dump on the Nth match.

For a SLIP defined to disable after N matches (via the MATCHLIM parameter), STDUMP writes a system trace record for all N matches, and also takes an SVC dump on the Nth match.



A=REFBEFOR,REFBEFOR=(target,operator,source) and A=REFAFTER,REFAFTER=(target,operator,source) are very powerful SLIP options that will update register content and storage locations when a SLIP matches. Use these with great care! The only difference between REFBEFOR and REFAFTER is whether the REFRESH action is done BEFORE or AFTER the other actions, such as taking a dump. If the damage to be repaired by the refresh option will not impact the gathering of documentation, then use REFAFTER so that the damage is viewable in the dump. If the damage will impair the ability to gather documentation, then use REFBEFOR to repair it first; however, you might want to use REFAFTER to save the corrupted data somewhere before using it to correct the corruption. Multiple actions can be taken on the REFBEFOR/REFAFTER specification.

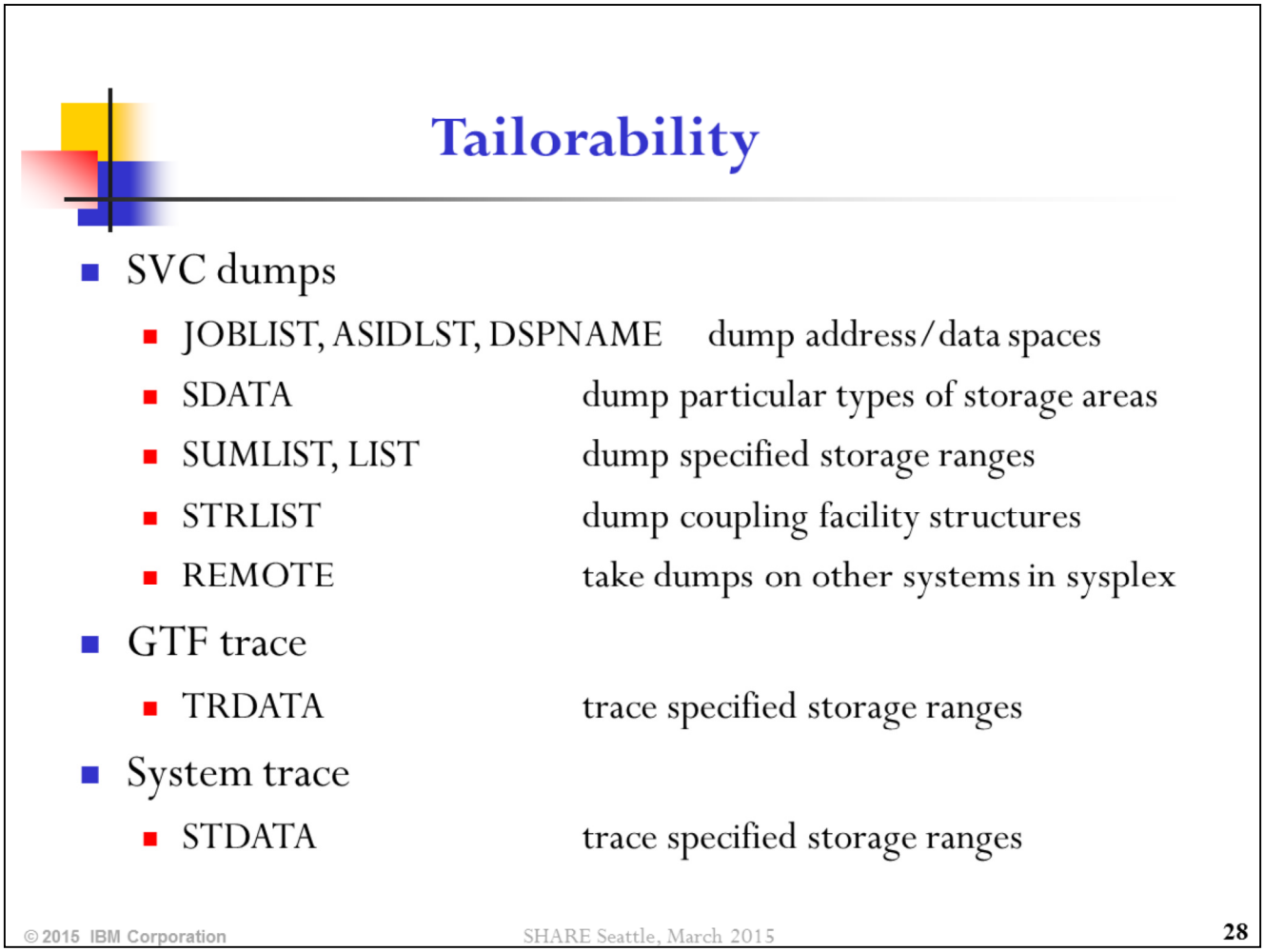
Dynamic PER is a capability that lets you trigger a new PER trap when an active PER trap matches. The most common use for dynamic PER is to trigger a SA SLIP on a particular area of storage after an IF SLIP detects the GETMAIN of that area of storage.

A=RECOVERY injects an ABEND06F (same as SLIP processing uses behind the scenes

for MSGID processing) to force active recovery to get control. A practical use of this could be if code is looping, or if code is known to be running with bad data that could lead to corruption.

SUBTRAP is a powerful but complex SLIP feature that is relatively new. It provides a sort of IF-THEN-ELSE capability with SLIP traps.

Both dynamic PERs and SUBTRAP PERs will be explored in SLIP Bootcamp Part 2.



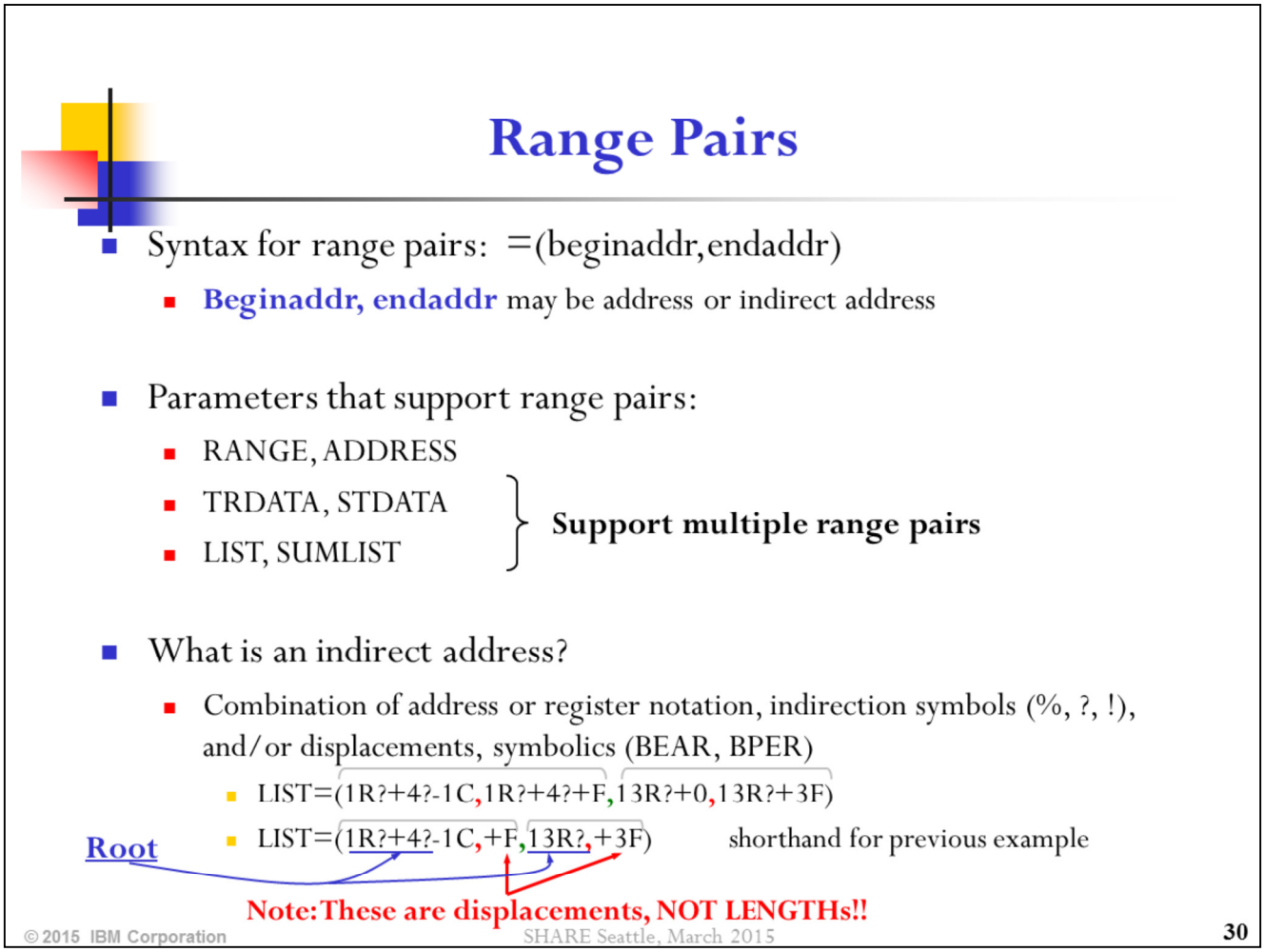
Multiple jobs, ASIDs, and data spaces may be requested. Note that JOBLIST and ASIDLST are not mutually exclusive.

SUMLIST gathers the specified data ranges during summary dump processing (very early in dump processing, and therefore very timely data). LIST gathers the specified data ranges during later dump processing.

Indirection may be specified in the LIST, SUMLIST, TRDATA, and STDATA parameters.

Wildcarding may be used in JOBLIST and DSPNAME.

Using Indirection in SLIPs



% - interpret address as 24-bit

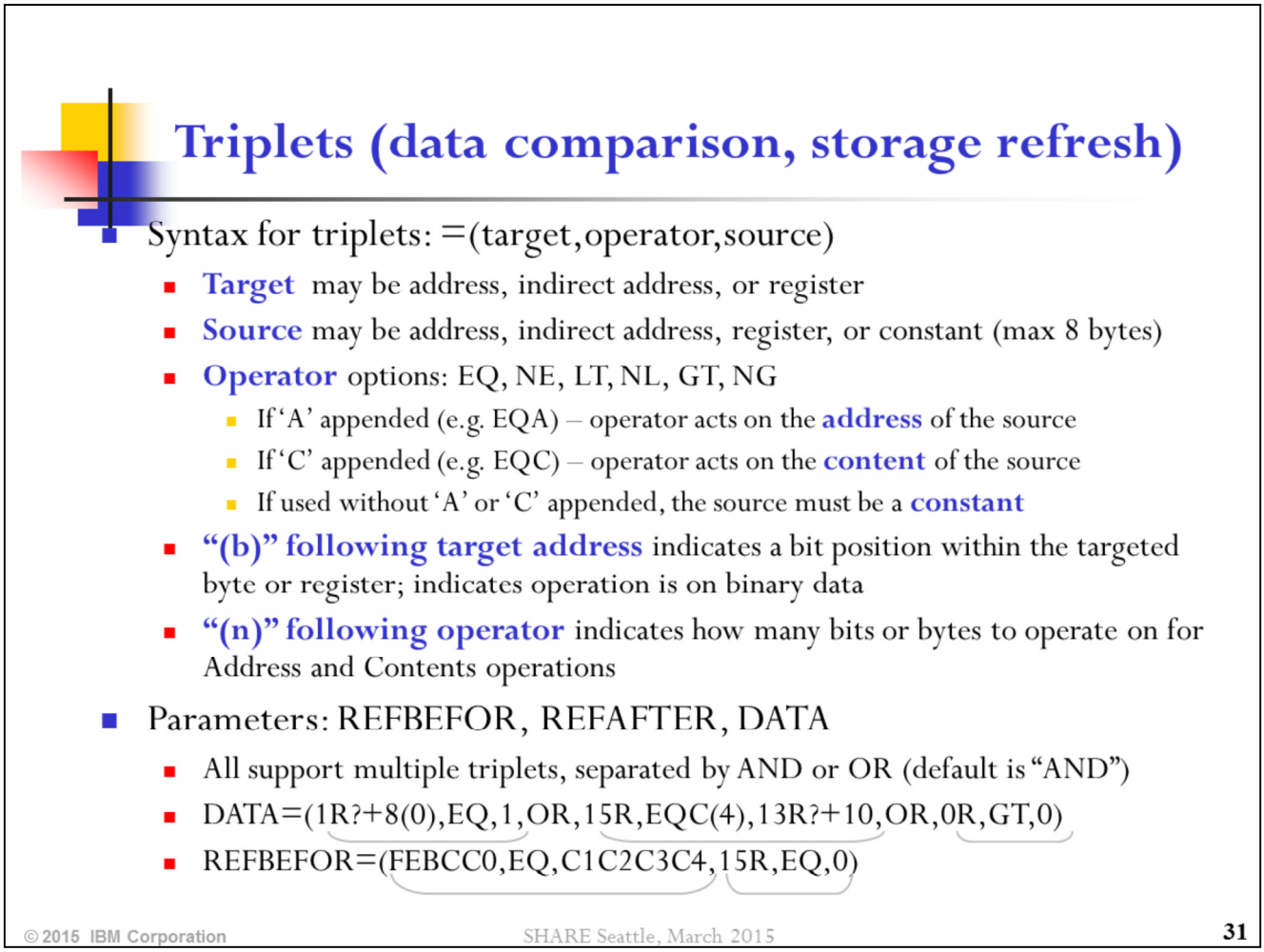
? – interpret address as 31-bit

! – interpret address as 64 bit

Use R to represent a 31-bit general purpose register and G to represent a 64-bit general purpose register.

The symbolic BEAR refers to the address where the last flow-altering instruction occurred. This includes branches, PC, PR, LPSW, etc.

The symbolic BPER refers to the beginning address in a PER range. It is helpful for DATA comparisons.



In addition to the operators listed above, there is also NE, NL, NG, etc.

A SLIP is operating on binary data if and only if there is a "(b)" value specified after the target. Based from 0, this indicates the beginning bit position that is to be altered or compared within the target register or address. If, in addition, a "(n)" is specified after the operator, this indicates how many bits are to be moved or compared. If no "(n)" is specified when performance a "C" (Contents) or "A" (Address) type of operation, the default is 1 bit.

If there is no "(b)" after the target, and the operator is followed by "(n)", this indicates how many bytes are to be altered or compared. This syntax should be used when performing a "C" (Contents) or "A" (Address) type of operation. The default is 4 bytes.

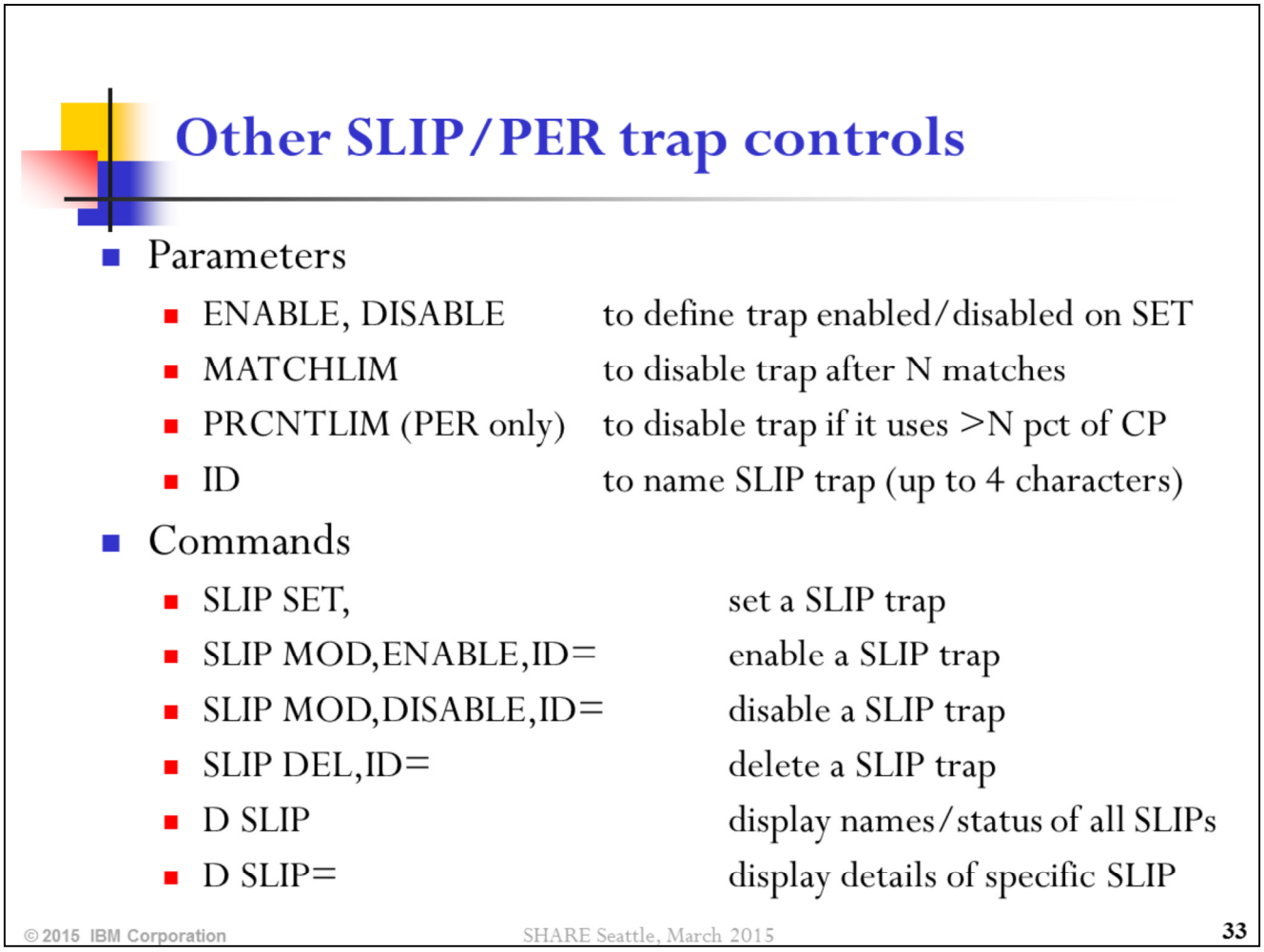
When SLIP does a contents or an address refresh (alter), it refreshes the first *n* bytes of storage and the last *n* bytes of a register.

If doing an EQ, GT, or LT, the number of bytes to be altered or compared is determined by the number of bytes in the constant supplied as the source.

The first example above (DATA=) will match if bit 0 at the storage location pointed to by Reg1+8 (perhaps the start of the 3rd word in a parameter list?) is on, or if the content of Reg15 is the same as the data found at address 13R?+10, or if Reg0 is greater than 0.

The second example above (REFBEFOR=) will update the storage at FEBCC0 to contain eyecatcher 'ABCD', and will set Reg15 to contain 0. This is done BEFORE any other requested actions, most likely because the damage that we are repairing would otherwise prevent successful completion of other actions.

Controlling SLIP traps



A SLIP trap can be set as disabled, then enabled at some later point. When entering multiple SLIP traps with interdependencies, it is advisable to enter them all disabled with a similar SLIP ID, then to enable them simultaneously through the SLIP MOD,ENABLE,ID= command, using wildcarding in the ID to target the multiple SLIPs.

A SLIP must be disabled before it can be deleted.

D SLIP will show the names of all SLIP traps defined to the system, along with an indication of whether the SLIP is enabled or disabled. To see the parameters of a specific SLIP, use: D SLIP=xxxx .

Any Questions?