The popular troubleshooting seminar, updated with the newest Db2 11.1 problem determination techniques and LIVE DEMOS! Learn how to identify and resolve database hangs, crashes, performance problems, and data corruption issues. Get familiar with diagnostic tools. Learn how and when to use these troubleshooting tools for a quick problem resolution. Topics covered: Performance; FODC; Hangs; Latching; Crashes/Traps; Data Corruption. Attendees will be able to characterize a problem in order to expedite resolution, learn how and what data to collect during intermittent hangs, slow-downs, and other frequent problems, and understand Db2 Support's methodology in troubleshooting Db2 and even non-Db2 hiccups. The seminar contains live demos during which the audience will be given an opportunity to perform an interactive investigation of selected problems.

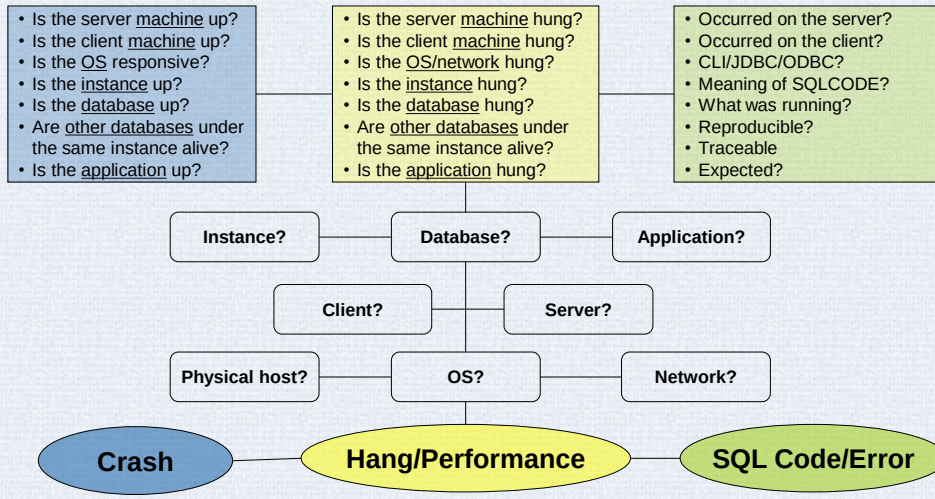

PREFACE: ESSENTIAL THEORY

A Bit of History: Db2 Releases

- 9.1 – Db2 Viper
- 9.5 – Db2 Viper II
- 9.7 – Db2 Cobra
- 9.8 – Db2 pureScale® Feature
- 10.1 – Db2 Galileo
- 10.5 – Db2 Kepler
- 11.1 – Db2 11.1 with BLU Acceleration

Problem Determination Decision Tree

- What happened? Where?



4

Most problems can be categorized into three main groups:

1. Crash
2. Hang/Performance
3. SQL Code/Error (unexpected results)

- To try and minimize the amount of data that needs to be collected and analyzed, it is beneficial to narrow down the scope of the problem.

- For example, are all users receiving a particular error or is it just one user on a client machine? If its just one or a few users or a particular application, start by examining the diagnostic information on the client side.

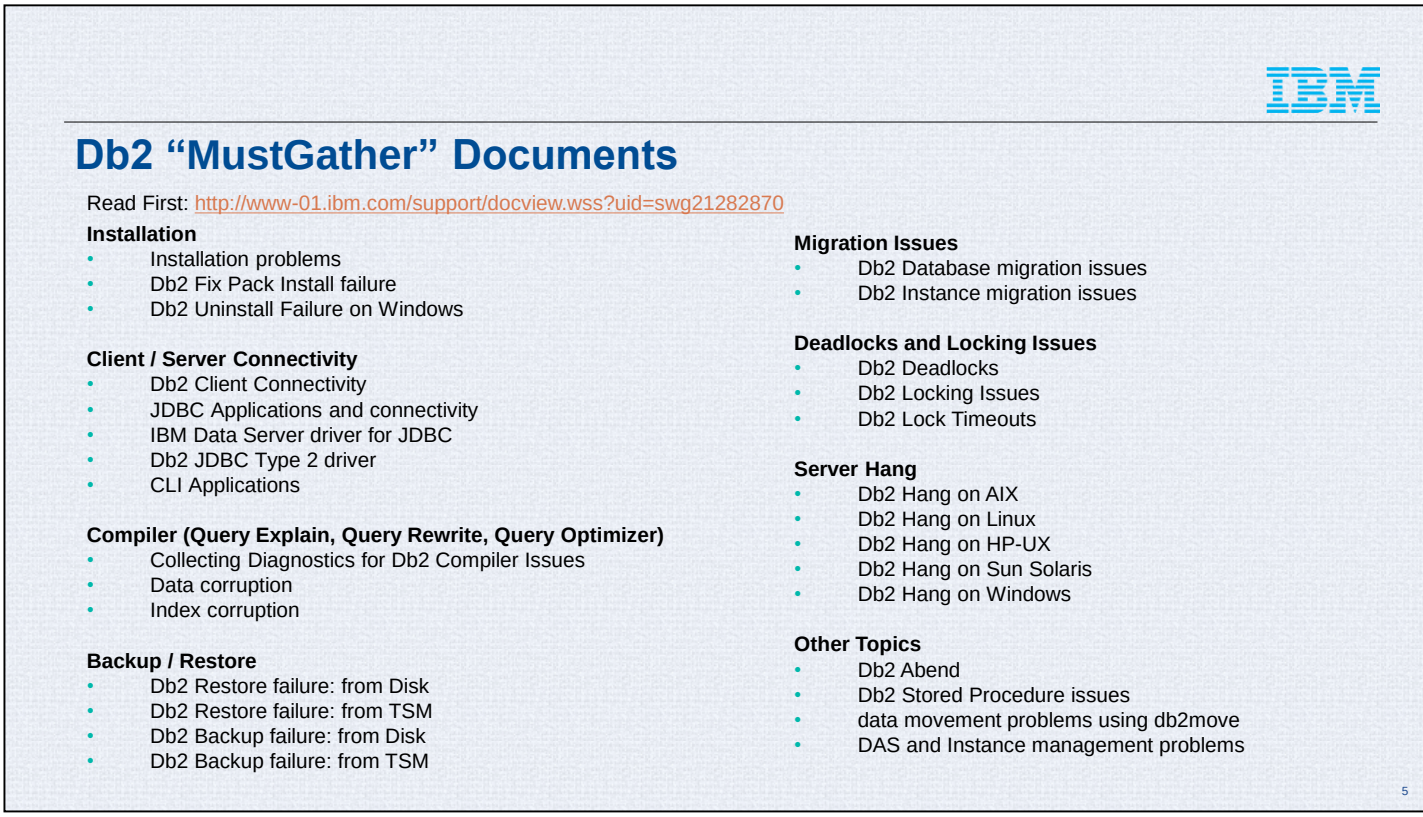
- If all the users/applications are reporting a problem, examine the diagnostic information on the database server.

- Note: **"I am having trouble connecting to Db2"** is NOT an error message we return. Rather **"When a user issues a connect to the db from the CLP as 'db2 connect to sample' they receive the message ----> sql1042 - unexpected system error".**

- Good practice is to include a brief description of the error message as it is defined. If the error message contains additional details such as that for sql0444n - user defined function "<function-name>" ... include all the additional info or secondary codes.

- Generally, secondary return codes are just SQL error codes i.e. sql2038 rc=1062 is really sql1062 - undefined db path. Read the **SQL error codes** as they contain useful information and actions to take that may potentially resolve the problem. Also provide the **SQLSTATE.**

- Can the problem be reproduced at will?



Db2 “Must Gather” Documents

For every problem, collecting data can aid in problem determination and save time resolving Problem Management Records (PMRs).

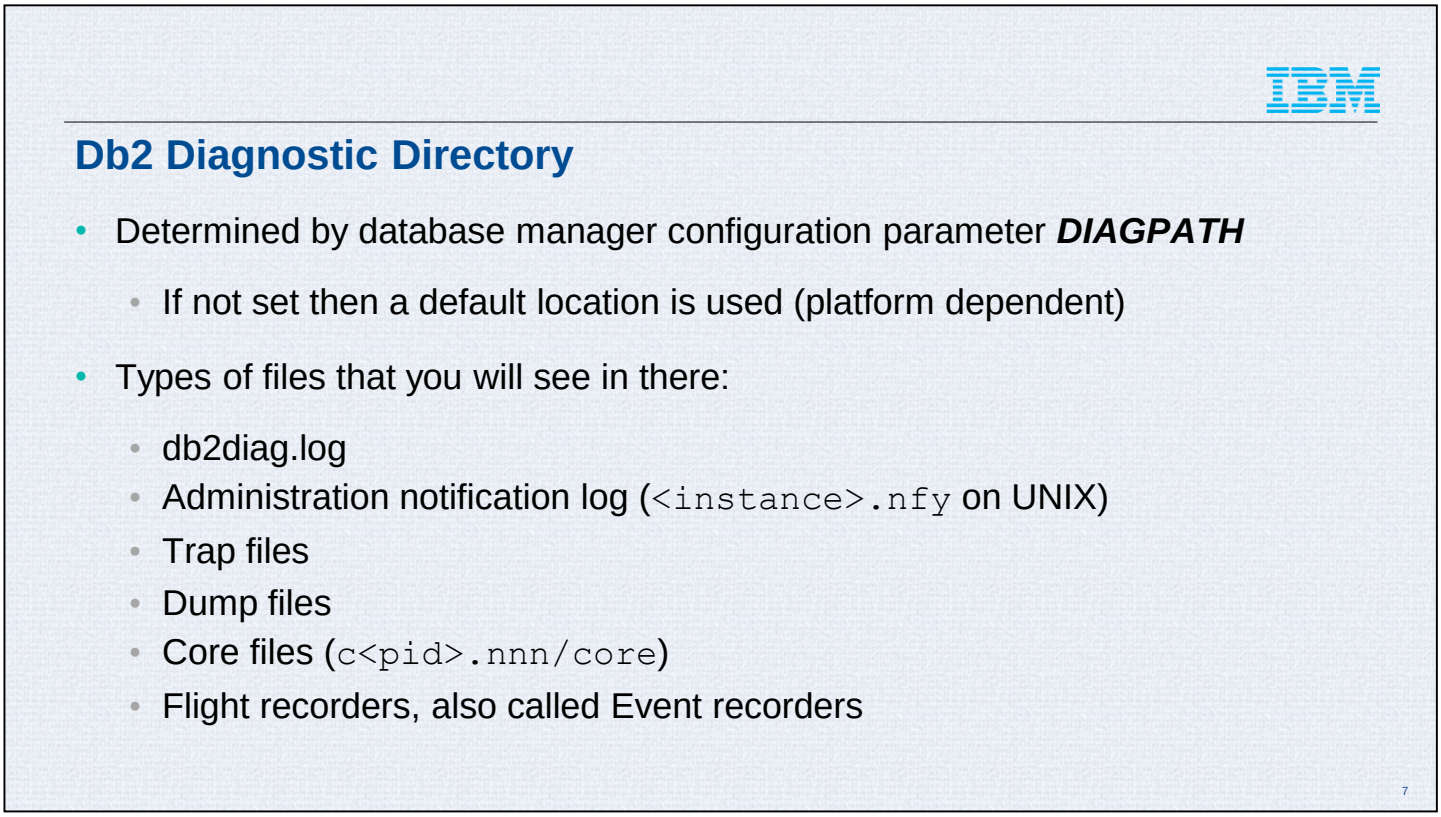
Gathering this data before calling IBM Db2 Support will help you understand the problem and save time analyzing the data. These documents will help answer the question which information should I collect?

Hexadecimal Dumps and Byte Ordering

- Some machines address bytes starting with the high-order byte (big endian). Others start with the low-order byte (little endian). For example, AIX and Solaris are using the big-endian model whereas INTEL, in particular Windows and Linux, use the little-endian model. The following shows how different sized variables would look like when dumped:

Variable Size	Big-endian	Little-endian
32 bit	0x12345678	0x78563412
16 bit	0x1234	0x3412
8 bit	0x12	0x12

- This reversal of bytes will not only affect the reason codes you may see in the Db2 diagnostic log file, but also will affect any other object hexadecimal dumps you will find throughout the Db2 diagnostic log file.



•DB2DIAG.LOG file

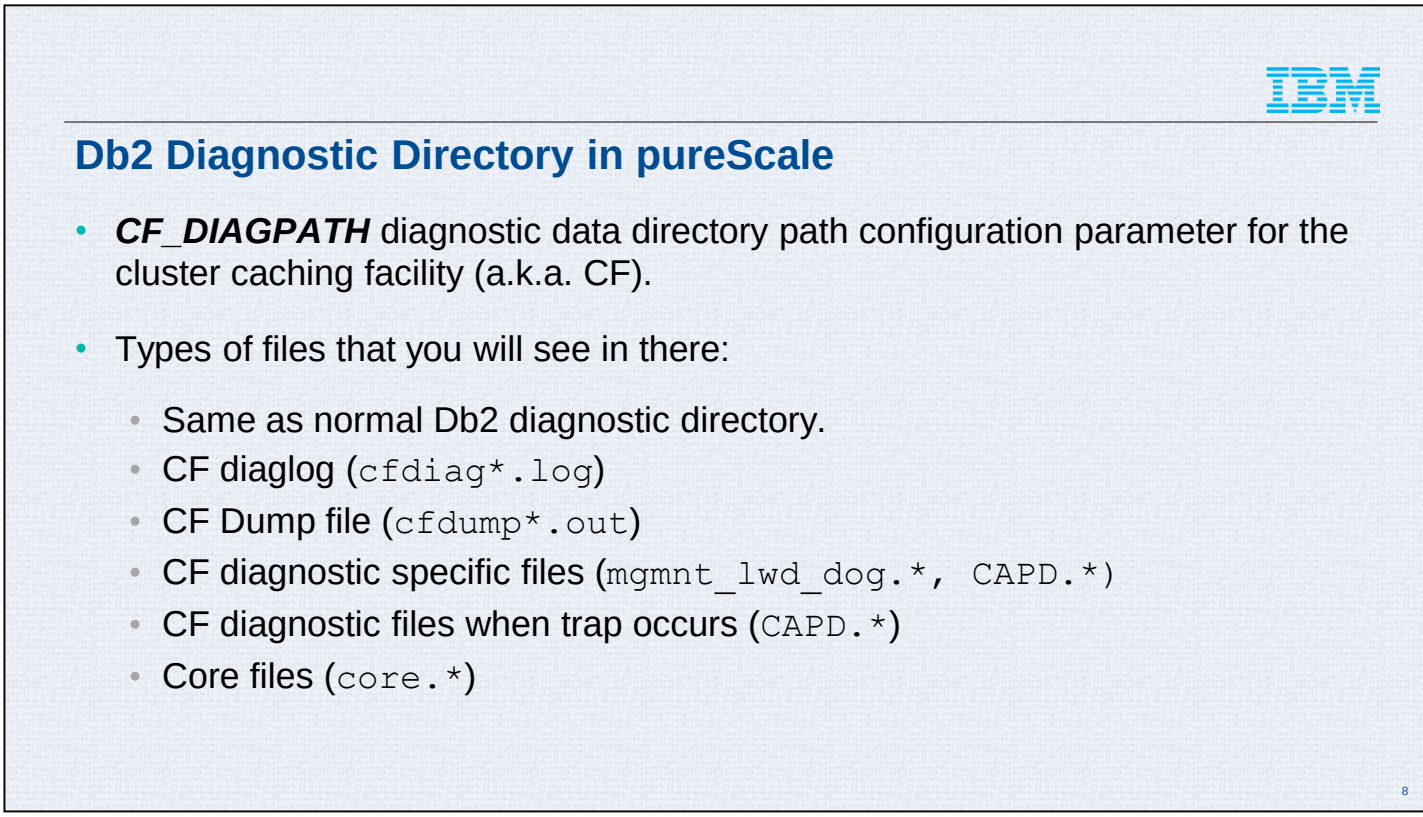
- •5 levels of instance level diagnostic settings – recommended is level 3
- •Default location, but configurable

•Dump files are binary files that contain various pieces of diagnostic information (miscellaneous components within Db2 will dump internal data structures, context, etc. when an error occurs). Only the Db2 Support team is able to format these files.

•The core files only exist on UNIX. Limited debugging can be done but the Db2 source code is needed to make a good use of them.

•The notification log only exists in UNIX. In Windows, notification log entries are written to the Windows Event Log and can be viewed using the Event Viewer.

•Flight recorder is a feature available in 9.8 and newer. It is a kind of a permanently active lightweight trace with a wrapping buffer which records recent events for a given component. As of now, the flight recorder output is considered internal and the output files cannot be formatted (used) by end customers.



Possible scenario for CF_DIAGPATH setting :

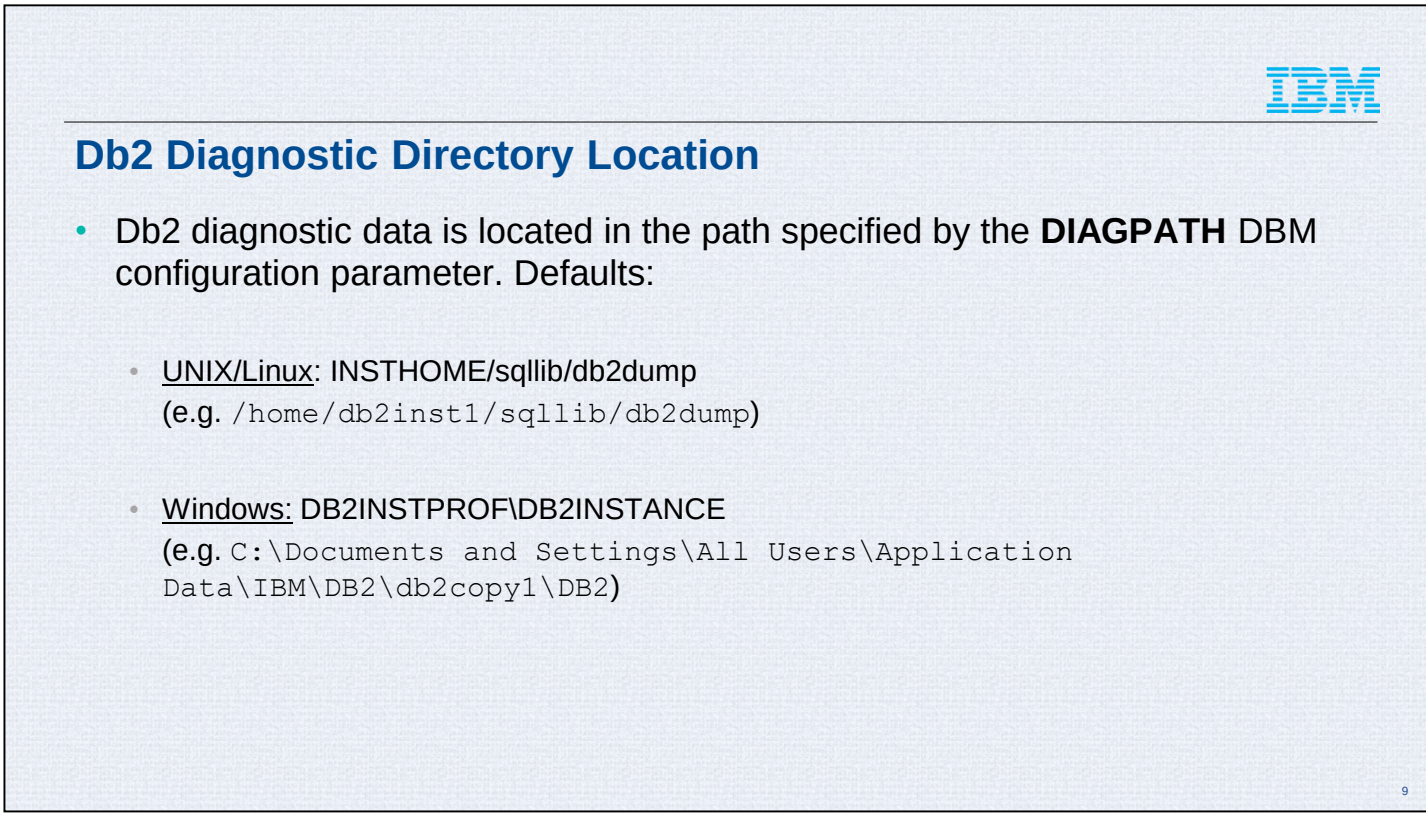
When CF_DIAGPATH is not set (default), it will be same as DIAGPATH, then DIAG0128 & DIAG0129 contains diagnostic dump from both Db2

(e.g. db2diag.log, admin log, etc) and CF (cfdiag*.log, cfdump*.out, etc)

When CF_DIAGPATH is set differently from DIAGPATH, then CF_DIAGPATH will have its own DIAG0128 & DIAG0129 containing diagnostic dump from CF.

E.g. cfdiag*.log, cfdump*.out, etc

DIAGPATH will have its own DIAG0128 & DIAG0129 too, containing diagnostic dump from Db2. E.g. db2diag.log, admin log, etc



In Windows environments: The default location of user data files, for example, files under instance directories, varies from edition to edition of the Windows family of operating systems. Use the DB2SET DB2INSTPROF command to get the location of the instance directory. The file is in the instance subdirectory of the directory specified by the DB2INSTPROF registry variable.

Db2 Diagnostic Directory Location in pureScale

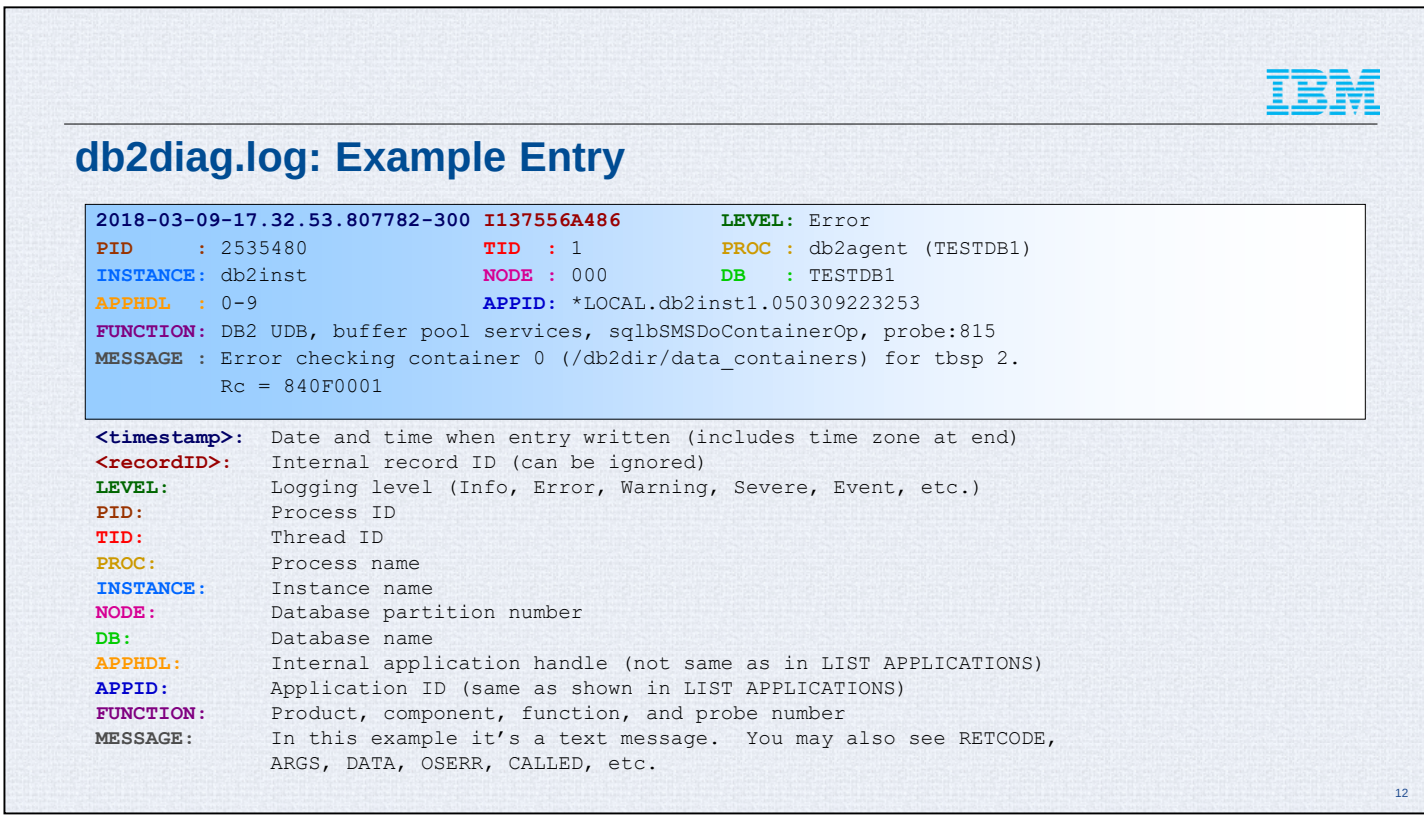
- Db2 diagnostic data located in the path specified by the ***DIAGPATH*** DBM configuration parameter while ***CF_DIAGPATH*** set to default:
  - UNIX/Linux: INSTHOME/sqllib/db2dump
    (e.g. `/home/db2inst1/sqllib/db2dump/<DIAG0000, DIAG0001, … DIAG0128, DIAG0129>`)
- If ***CF_DIAGPATH*** is set:
  - DIAG0128 and DIAG0129 created on the specified ***CF_DIAGPATH***
  - ***DIAGPATH*** will also have its own DIAG0128 and DIAG0129
  - Db2 will write into both locations

10

DIAG0128 and DIAG0129 are the diagnostic directories for the cluster caching facilities, a.k.a. CFs

Db2 Diagnostic Log – db2diag.log

- Diagnostic information is recorded to this file
- Mainly intended for Db2 Support
  - All messages are written in English
  - Many entries have no meaning without access to the Db2 source code
  - However, still valuable to examine when trying to diagnose a problem yourself
- Diagnostic entries captured are determined by database manager configuration parameter ***DIAGLEVEL***:
  - 0: No diagnostic data captured
  - 1: Severe errors only
  - 2: All errors
  - 3: All errors and warnings (*default*)
  - 4: All errors, warnings and informational messages



As an example of solving a problem yourself using the db2diag.log file, consider the case where a table space has been placed OFFLINE but you don't know why that has happened. Provided that you are running with a high enough DIAGLEVEL (the default will definitely capture this) you will see entries like this:

```
...
MESSAGE : Error checking container 0
          (/home/db2inst1/db2inst1/NODE0000/SQL00001/SQLT0002.0) for tbsp 2.
          Rc = 840F0001
...
RETCODE : ZRC=0x8402001E=-2080243682=SQLB_CONTAINER_NOT_ACCESSIBLE
          "Container not accessible"
...
MESSAGE : ADM6023I  The table space "USERSPACE1" (ID "2") is in state 0x"0".
          The table space cannot be accessed.  Refer to the documentation for
          SQLCODE -290.
...
```

Looking at this, you can see that the table space has been placed offline because one of its containers is inaccessible ("`/home/db2inst1/db2inst1/NODE0000/SQL00001/SQLT0002.0`"). If you fix the accessibility issue you can then bring the table space back online.

Essential Tools: db2level

Identifies the current level of your Db2 instance

```
$ db2level
DB21085I  This instance or install (instance name, where applicable: "db2inst1")
uses "64" bits and DB2 code release "SQL11011" with level identifier "0202010F".
Informational tokens are "DB2 v11.1.1.1", "s1612051900", "DYN1612051900AMD64", and Fix Pack "1".
Product is installed at "/opt/IBM/db2/V11.1".
```

db2inst1	Name of the current instance
64	Bitness of the instance (32 or 64 bits)
SQL11011	Product signature
0202010F	Internal release number
DB2 v11.1.1.1	Fix Pack/Mod signature
s1612051900	Internal build level/date
DYN1612051900AMD64	Platform/PTF identifier (platform dependent)
1	Fix Pack number (platform independent)
/opt/IBM/db2/V11.1	Install path

Essential Tools: `db2support`

- Collects environment data about either a client or server machine and places the files containing system data into a compressed file archive.
- The following syntax variations cover most problems:

  1) Collect data while the database is responsive
  ```
  db2support <outputdir> –d <dbname> -c –s
  ```

  2) Collect data while a database hang is suspected
  ```
  db2support <outputdir> –d <dbname> –s
  ```

  3) Collect data for a reproducible query performance problem:
  ```
  db2support <outputdir> -d <dbname> -sf <sqlfile> -cl 1
  ```

- *Note: The "-c" switch allows `db2support` to connect to the database. This is not desirable during suspected database hangs.*

14

-d *database_name* | -database *database_name*

Specifies the name of the database for which data is being collected.

-c | -connect

Specifies that an attempt be made to connect to the specified database.

-s | -system_detail

Specifies that detailed hardware and operating system information is to be gathered.

-sf *SQL file* | -sqlfile *SQL file*

Specifies the file path containing the SQL statement for which data is being collected.

-cl | -collect

Specifies the value of the level of performance information to be returned. Valid values are:

0 = collect only catalogs, db2look, dbcfg, dbmcfg, db2set

1 = collect 0 plus db2exfmt, db2caem (if -aem or -actevm, -appid, -uowid, -actid are specified)

2 = collect 1 plus .db2service (this is the default)

3 = collect 2 plus db2batch

db2diag -filter db=TESTDB -node 0 -time 2010-03-08-17.42.35.164191: Shows all db2diag.log entries that are associated with database TESTDB on database partition (node) 0 that were written since the given timestamp.

db2diag -gi “level=severe” -H 1d: Shows all db2diag.log entries that are listed as “Severe” (-gi is a case-insensitive search) that were added in the last day.

APARs

- Authorized Program Analysis Reports
  - Bugs in code or documentation that require a fix
  - Fixes for APARs are provided through Db2 Fix Packs
- Search on known APARs from Db2 Support site

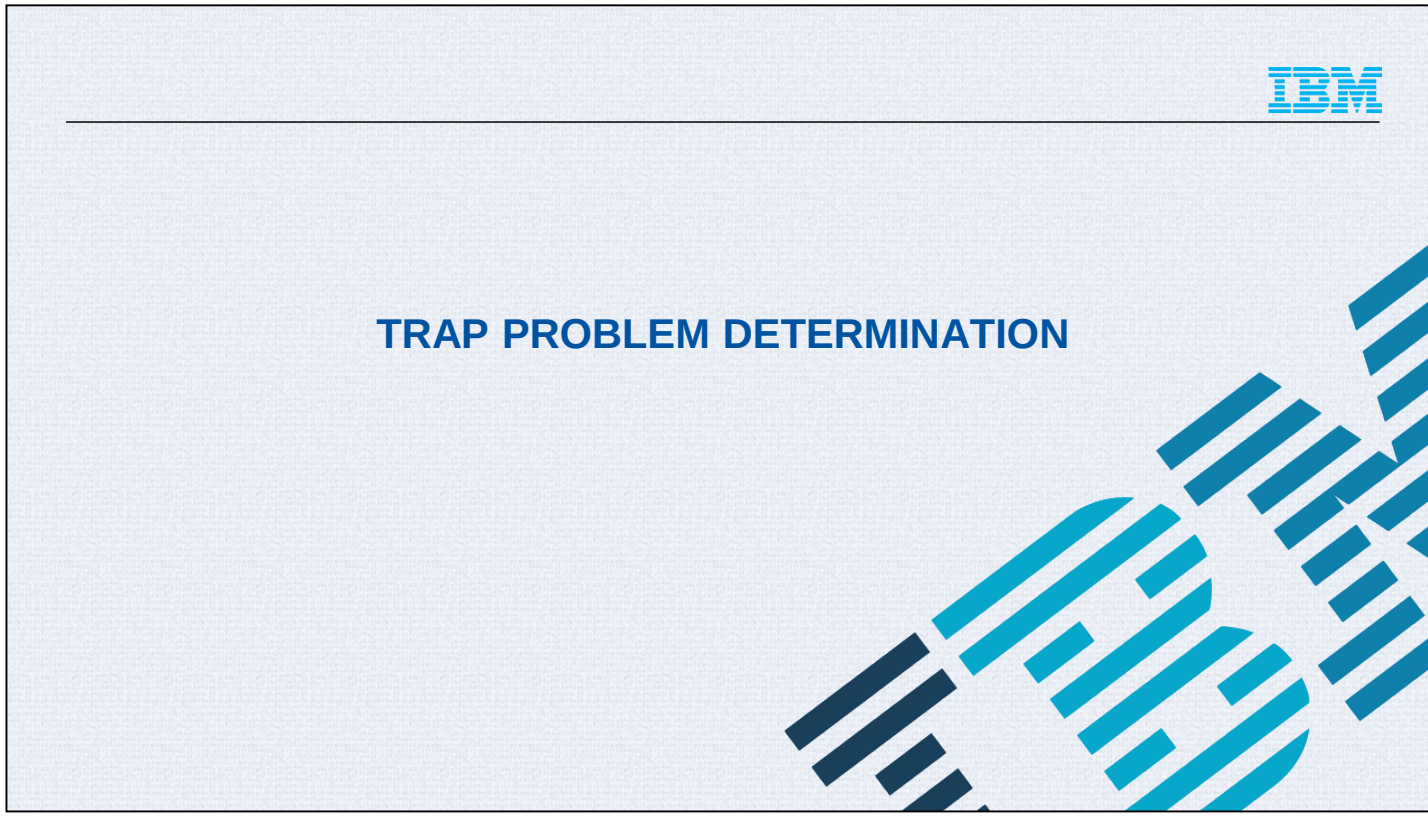
- Tip: Search for keywords related to your problem, including:
  - Db2 source code function names from:
    - db2diag.log
    - notification log
    - trap files (functions near the top of the stack)
  - SQLCODEs – e.g. SQL1042C/SQL1042/-1042
  - Operation/utility/workload being performed at time of problem

- E.g. “SQL1042C and db2start”

16

•Note that for trap files, the top couple of functions may be common error handling routines within Db2. Therefore, many different problems may show the same set of functions at the top of the stack.

•For these cases, what’s really important may be what’s “below” it. Therefore, when searching on the function names make sure that you try different ones as part of the search.


TRAP PROBLEM DETERMINATION

Trap Definition

- A **crash/abend** is a very generic term to describe any time Db2 comes down when it should not, i.e. an abnormal end

- A **trap** occurs when a process or thread receives a signal or raises an exception as the result of an instruction which cannot be executed. A trap is a very specific term and is not to be confused with “panic”, “shutdown”, "stop", or the more generic term of "crash". When discussing a scenario where you are sure Db2 trapped, use that terminology, i.e. "Db2 trapped" or "the instance trapped“.

Scope of Outage: Terminology

- "Instance abend" is an abend of the entire database manager (DBM). All connections to all databases will be terminated, processing will halt completely, and Db2 engine processes will disappear.

- "Database abend" means a shutdown of a specific database only. All connections to that specific database will be terminated, but the Db2 engine will remain up and running, and other databases will function normally.

TIP: if `db2start` needs to be run, it is an instance abend.

Multiprocessed Architecture

- UNIX/Linux, Db2 V9.1 and older – multiprocessed:

```
$ db2nps 0
Node 0
     UID        PID       PPID     C     STIME     TTY     TIME CMD
  psustr    1175724    3109032     0       Sep      29        - 0:03 db2sysc
  psustr     856180    1175724     0       Sep      29        - 0:02 db2ipccm
  psustr    1302588    1175724     0       Sep      29        - 0:00 db2gds
  psustr    1323260    1175724     0       Sep      29        - 0:00 db2resync
  nobody    1335386    1175724     0       Oct      01        - 0:00 db2fmp (C)
  psustr    1355860    1175724     0       Sep      29        - 5:32 db2hmon
    root    1753288    1175724     0       Sep      29        - 0:00 db2ckpwd
    root    1847322    1175724     0       Sep      29        - 0:00 db2ckpwd
    root    2846734    1175724     0       Sep      29        - 0:00 db2ckpwd
  psustr    1720496     856180     0       Sep      29        - 0:02 db2agent (idle)
  psustr    2019378     856180     0       Oct      01        - 0:08 db2agent (idle)
  psustr    2306128     856180     0       Sep      29        - 0:19 db2agent (idle)
  psustr    3047456     856180     0       Oct      01        - 0:08 db2agent (idle)
  psustr     909438    1302588     0       Sep      29        - 0:00 db2srvlst
```

Every engine dispatchable unit (EDU) implemented via a separate UNIX process having a unique process ID.

Multithreaded + Multiprocessed Architecture

- UNIX/Linux 9.7+ – multiprocessed and multithreaded:

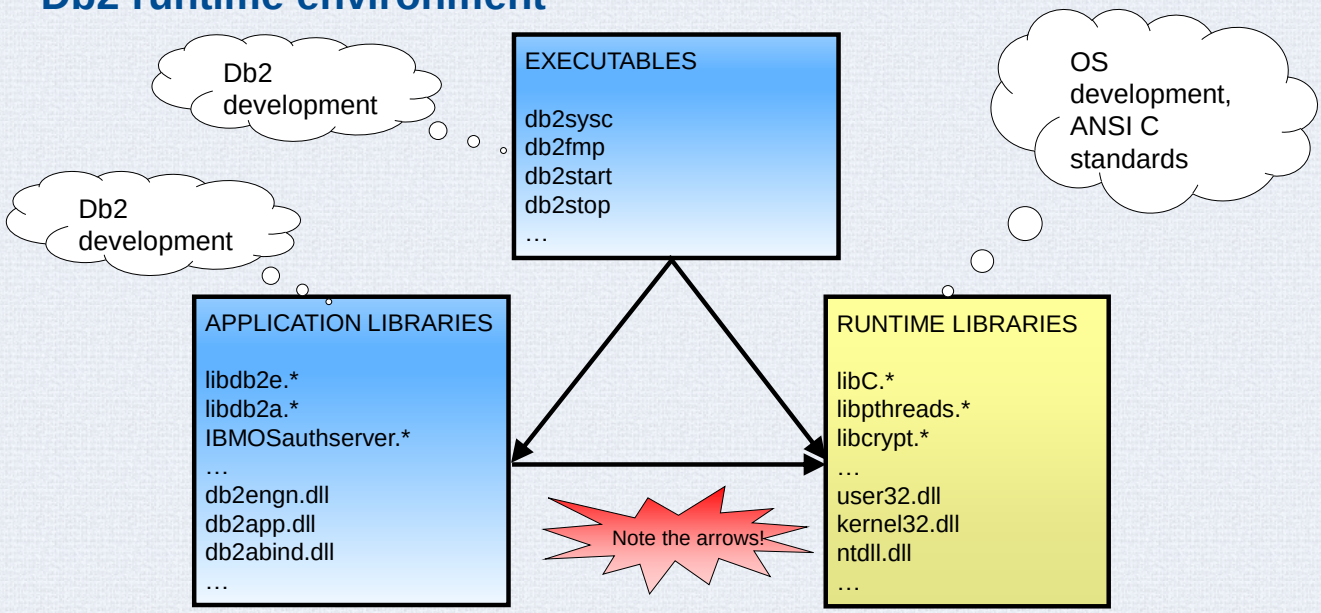
```
$ db2nps 0
Node 0
     UID        PID       PPID    C     STIME     TTY     TIME CMD
  psustr    3219534    3100840    0  16:21:33       -     0:00 db2sysc 0
    root    2269266    3219534    0  16:21:34       -     0:00 db2ckpwd 0
    root    2871470    3219534    0  16:21:34       -     0:00 db2ckpwd 0
    root    3039246    3219534    0  16:21:34       -     0:00 db2ckpwd 0
```

All EDUs now implemented as threads inside a single db2sysc. A new `db2pd -edus` command can be used to view the threads:

```
$ db2pd -edus
<...>
EDU ID    TID            Kernel TID     EDU Name          USR          SYS
============================================================================
2828      2828           5619907        db2resync 0       0.001298     0.000077
2571      2571           5652681        db2ipccm 0        0.000673     0.000318
2314      2314           5242967        db2licc 0         0.000828     0.000671
2057      2057           5460085        db2pdbc 0         0.000723     0.000072
1800      1800           5648583        db2extev 0        0.000861     0.000068
<...>
```

- Windows: always multiprocessed and multithreaded

Db2 runtime environment



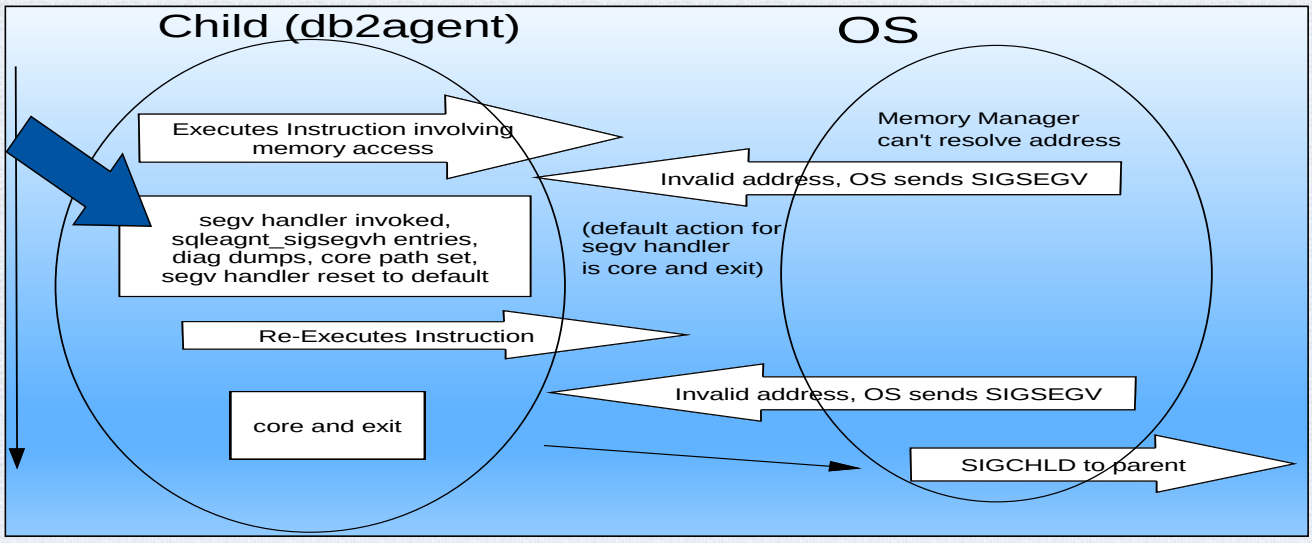
Db2 runtime environment

- Db2, just like most other programs, consists of three main component areas:
  - Program executables: usually contain some very basic functionality. Most of the important code is located in application libraries (see below).
  - Application libraries: the majority of code. Db2 libraries are *roughly* organized according to their use. For example, most engine code is located in libdb2e.* (‘e’ standing for ‘engine’) on UNIX or DB2ENGN.DLL on Windows.
  - Runtime libraries: helper libraries allowing the developer to utilize for example the standard C/C++ library routines, cryptographic features, etc. This code is not owned or created by Db2 development. If you see a crash in a runtime library, chances are that the problem is in the caller (Db2).
- Each component (executable, library) lives in its own address space. The address is assigned by the operating system.

Failing Line of Code

- Determining the line of code is the same on all platforms. You will need to know:
  1. Name of the program/library that you are executing in
  2. Address range where this executable/program has been loaded
  3. Offset at which you are executing – relative to the beginning of the library

Db2 Trap Signal Handling on UNIX/Linux



Inside the sqloEDUCodeTrapHandler function:

•A request is made to access some information in memory.

•The OS sends a signal to the process indicating the its an invalid address in memory.

•Db2's signal handler receives this signal and dumps information to the db2diag.log and sets up the path to the CORE file. The signal handler then resets itself.

•Db2 then requests the same information from memory again.

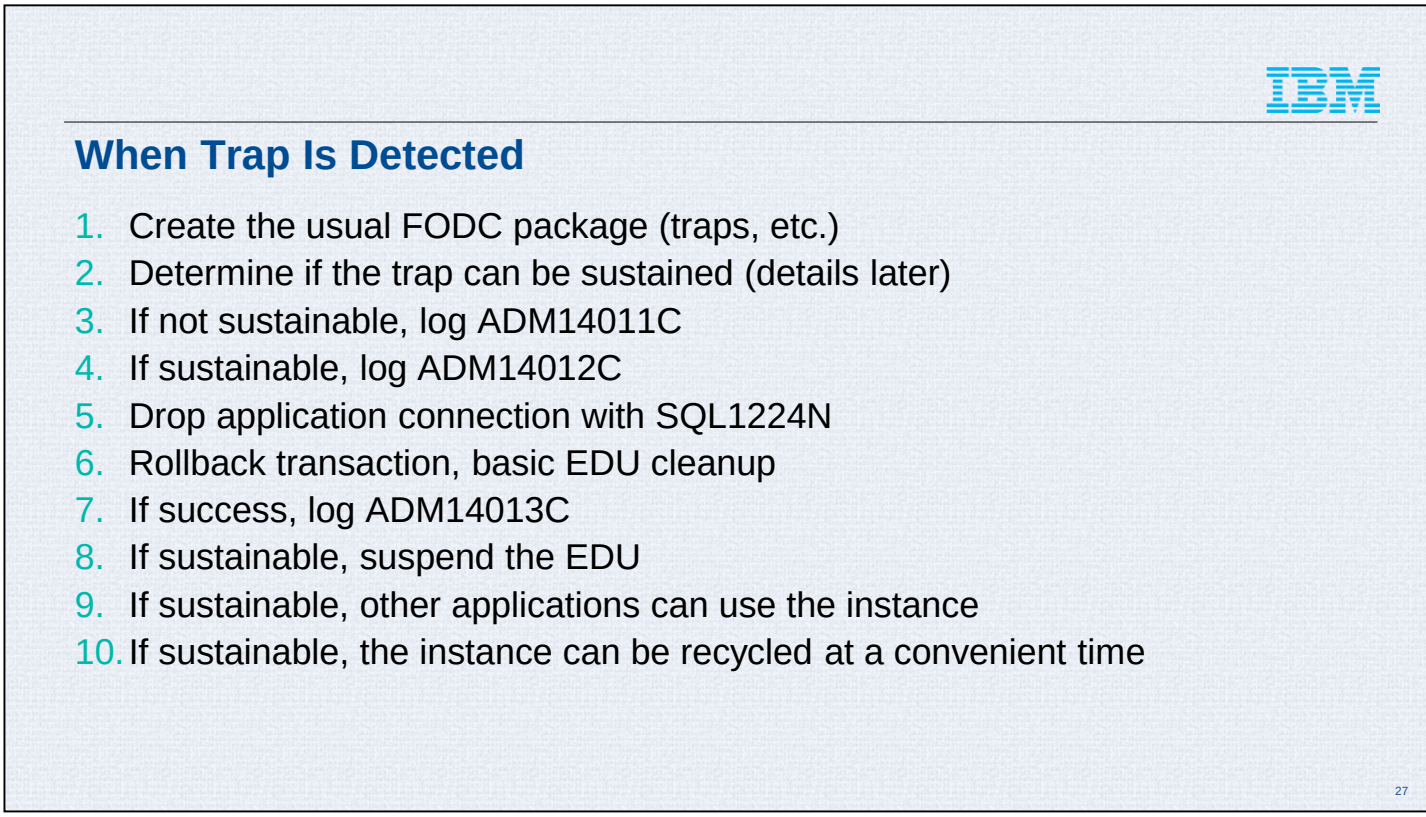
•The OS sends a signal to the process indicating the its an invalid address in memory.

•Db2's signal handler receives the signal from the OS and produces a CORE and exits.

•This is done to give us the opportunity to dump diagnostic information on receiving the first signal from the OS.

Db2 Exception Handling on Windows

- Db2 crash handling on Windows is implemented via exception handlers. When an exception occurs, a trap file that captures the exception context is written.
- The basic principles of this concept are similar to the previous picture showing UNIX/Linux crash signal handling, except that on Windows there are no signals but exceptions.



•Customers may experience unexpected Db2 outages

•These outages cause loss of business and inconvenience

•The goal is to reduce unexpected outages and increase data availability

Trap Resilience

A trap is an interrupt caused by an exceptional situation in Db2

Customers experience unexpected traps causing losses of business

The feature will keep the instance up during this kind of service interrupts

Trap Resilience: Sustainability

- The following list contains examples of when a trap cannot be sustained. The actual implementation is subject to change without any notice, even between Fix Pack:
  - Outage type is NOT a trap (e.g. bad page, index error)
  - EDU already in the process of sustaining a trap
  - EDU is running a utility (backup, restore, load, inspect, ...)
  - EDU is issuing a DDL (only DML supported at the moment)
  - EDU is executing a Db2 kernel operation
  - **DUMPCORE** is enabled in the **DB2FODC** registry variable
- Trap resilience can be disabled by the **DB2RESILIENCE** registry variable. Default: ON

Trap Files

- A **trap file** is a snapshot for the state of a Db2 process
- The state reflects the situation at the time when the trap file was generated, i.e. not much historical data
- Generated automatically if processing cannot continue because of an exception, or for serviceability reasons by Db2
- Contains the function sequence that was running when the error occurred
- Also contains information about the state of the process when the exception was caught, e.g. contents of registers, disassembly of code around the failing line, etc.

Trap Files – Naming Convention

Platform	Name
UNIX/Linux	<pid>.<eduid>.<node>.**trap.txt**
Windows	<pid>.<tid>.**trap.bin**

- On UNIX/Linux, the file is text-based
- On Windows, the file is binary. In order to translate the binary file into text, a formatting utility (`db2xprt`) and debug symbol files are required, both of which are shipped with the product.

30

Traps, if existing, will always reside in the Db2 diagnostic directory (DIAGPATH).

Trap File Contents

- Build date
- Version Number
- Time of the dump
- **Signal or Exception which generated the dump**
- **Process & thread ID**
- Loaded libraries (common referred to as the “map”)
- Address of signal handlers
- **Registry dumps**
- **Call Stack – a detailed call stack**
- A dump of the OSS Memory sets
- Latch information for the EDU
- Locks being waited on
- Assembly code dump, …

Trap File Header

```
DB2 build information: DB2 v11.1.1.1 s1612051900 SQL11011        1
timestamp: 2018-07-05-16.51.29.801489                            2
instance name: db2inst1.001                                      3
EDU name     : db2agntp (SAMPLE) 1 [-]                           4
EDU ID       : 128                                               5
Signal #11                                                       6
uname: S:Linux R:3.0.101-63-default ... M:x86_64 N:demobox       7
process id: 25676                                                8
parent process id: 25672                                         9
thread id : 140201049450240 (0x7F8319BFE700)                    10
kthread id : 26587                                              11
```

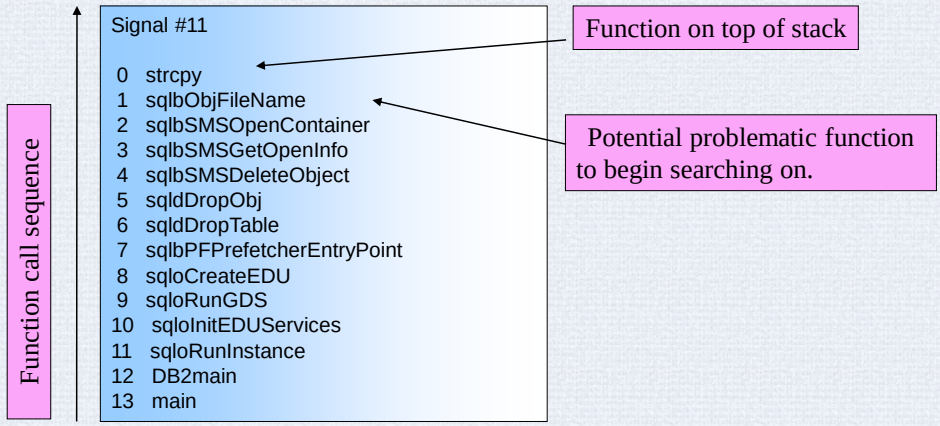
Refer to Speaker's Notes.

32

1. Db2 version information.
2. Timestamp when the trap file was generated.
3. Instance name and node number.
4. Name of the EDU and the database the EDU is connected to.
5. EDU ID, visible for example in "db2pd –edus".
6. Signal that generated the trap file.
7. OS version, machine number and machine name. "man uname" on a UNIX box for more information.
8. Process ID.
9. Parent process ID.
10. Thread ID.
11. Internal Kernel thread ID.

Call Stack

How to read a call stack:



33

NOTE: In general, you would want to search for the function on the top of the stack.

•In this example, the function strcpy() is a C library function and is less likely to by the culprit (not impossible).

•The most likely culprit is the caller to the strcpy() function. The strcpy() function may just be the victim.

Trap Signals/Exceptions

UNIX/Linux Signal ID	Description
SIGILL(4), SIGFPE(8), SIGTRAP(5), SIGBUS(10, Linux: 7), SIGSEGV(11), SIGKILL(9)	Instance trap. Bad programming, HW errors, invalid memory access, stack and heap collisions, problems with vendor libraries, OS problems. The instance shuts down.

Windows Exception		Description
ACCESS_VIOLATION	(0xC0000005)	Instance trap. Bad programming, HW errors, invalid memory access, stack overflows, problems with vendor libraries, OS problems. The instance shuts down.
ILLEGAL_INSTRUCTION	(0xC000001D)	
INTEGER_DIVIDE_BY_ZERO	(0xC0000094)	
PRIVILEGED_INSTRUCTION	(0xC0000096)	
STACK_OVERFLOW	(0xC00000FD)	

34

•On UNIX, a signal can be sent to a Db2 process by issuing a "kill - <signal #>. Signals are defined in the "signals.h" header file.

•For example, on AIX 5.3, the signal.h header file is located in /usr/include.sys/signal.h

•An extract of the signal.h header file is as follows:

```
#define  SIGHUP     1  /* hangup, generated when terminal disconnects */
#define  SIGINT     2  /* interrupt, generated from terminal special char */
#define  SIGQUIT    3  /* (*) quit, generated from terminal special char */
#define  SIGILL     4  /* (*) illegal instruction (not reset when caught)*/
#define  SIGTRAP    5  /* (*) trace trap (not reset when caught) */
#define  SIGABRT    6   /* (*) abort process */
#define SIGEMT    7  /* EMT intruction */
#define  SIGFPE     8  /* (*) floating point exception */
#define  SIGKILL    9  /* kill (cannot be caught or ignored) */
#define  SIGBUS     10  /* (*) bus error (specification exception) */
….
….
```

• To send an abort signal (SIGABRT) to a process, issue a "kill -6 <pid>".

• On Windows, use db2pd –stack to send "signals" to db2 processes/threads.

• WARNING: DO NOT randomly issue signals to a Db2 process unless directed to by Db2 Service. Sending inappropriate signals can lead to database problems.

Abend Symptoms

Applications or users receive an error indicating the death of the database manager when a request is submitted to the database. Common errors are:

```
SQL1224N The database manager is not able to accept new requests, has terminated
all requests in progress, or has terminated your  particular request due to a
problem with your request.
```

OR

```
SQL1032N  No start database manager command was issued.  SQLSTATE=57019
```

Workshop Environment

The exercises assume the existence a single partitioned Db2 instance running Db2 9.7 GA. Real life scenarios are used whenever possible, otherwise a customized Db2 library or other simulated tricks may be used ☺. The GA level was specifically chosen to allow us to reproduce known problems.

```
$ db2level
DB21085I  Instance "db2inst1" uses "64" bits and DB2 code release "SQL09070"
with level identifier "08010107".
Informational tokens are "DB2 v9.7.0.0", "s090521", "LINUXAMD6497", and Fix
Pack "0".
Product is installed at "/opt/ibm/db2/V9.7".

$ uname -a
Linux demobox 3.0.13-0.27-default #1 SMP Wed Feb 15 13:33:49 UTC 2012 (d73692b) x86_64
x86_64 x86_64 GNU/Linux
```

36

Before we begin with the hands-on, a few words about the environment. If you meet these conditions, you can apply the customized libraries to your environment so you can try this at home, too ☺

Scenario

```
$ db2 "connect to sample"

   Database Connection Information

 Database server        = DB2/LINUXX8664 9.7.0
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE

$ db2 "alter tablespace IBMDB2SAMPLEREL managed by automatic storage"
DB20000I  The SQL command completed successfully.

$ db2 "alter tablespace IBMDB2SAMPLEREL lower high water mark"
DB20000I  The SQL command completed successfully.

$ db2 "list tables"
SQL1224N  The database manager is not able to accept new requests, has
terminated all requests in progress, or has terminated the specified request
because of an error or a forced interrupt.  SQLSTATE=55032
```

Scope of Outage: Applied

For example, the following commands can be used to narrow down the scope of the outage:

```
$ db2 list active databases
SQL1032N  No start database manager command was issued.  SQLSTATE=57019

$ db2nps 0
Node 0
     UID        PID       PPID    C     STIME     TTY     TIME CMD

$ db2pd -edus
Unable to attach to database manager on partition 0.
Please ensure the following are true:
   - db2start has been run for the partition.
```

Conclusion: The entire database manager is down. This is an instance abend (not a database one). Will need to run `db2start` eventually.

38

-**db2nps <nodenum>** shows the process list for the given node, similar to a **ps** output on UNIX

-**db2pd –edus** shows the running threads/processes for the current instance

Diagnostics Produced by Trap

- Errors written to the db2diag.log
- Message written to the notify log (*.nfy)
- Errors written to the operating system logs
- Trap and dump files created in the **DIAGPATH**
- Core dumps for Db2 processes
- FODC package created in the **DIAGPATH**

Diagnostics: Applied

Always remember to check the DIAGPATH first:

```
$ db2 get dbm cfg | grep DIAGPATH
 Diagnostic data directory path                  (DIAGPATH) = /home/db2inst1/sqllib/db2dump

$ ls -l /home/db2inst1/sqllib/db2dump
total 72
-rw-rw-rw- 1 db2inst1 db2iadm1 53465 Mar 26 11:30 db2diag.log
-rw-rw-rw- 1 db2inst1 db2iadm1  1372 Mar 26 11:30 db2inst1.nfy
drwxr-x--- 4 db2inst1 db2iadm1  4096 Mar 26 11:30 FODC_Trap_2018-03-26-11.30.45.417849
drwxrwxr-t 2 db2inst1 db2iadm1  4096 Mar 26 11:30 stmmlog
```

Conclusion: The presence of the “FODC_Trap” directory tells us right away that we are dealing with an instance-wide trap.

Time of Outage: Step 1

Examine db2diag.log, find the first timestamp *pertinent to the outage*:

```
2018-03-26-11.30.45.695289-240 I9822E563          LEVEL: Error
PID     : 26689                TID  : 140093448775424PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000          DB   : SAMPLE
APPHDL  : 0-14                 APPID: *LOCAL.DB2.120326153046
AUTHID  : DB2INST1
EDUID   : 31                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, base sys utilities, sqleagnt_sigsegvh, probe:1
MESSAGE : Error in agent servicing application with coor_node:
DATA #1 : Hexdump, 2 bytes
```

Good strings to search for:

- *"Error in agent"*
- *"pdEDUIsInDB2KernelOperation"*
- *"pdResilienceIsSafeToSustain"*
- *"FODC_Trap…"* (name/type of the FODC directory)

Note the trapping EDU is db2agent with the EDUID of 31.

Time of Outage: Step 2

Continue to search db2diag.log backwards, find the first *unrelated timestamp*:

```
2018-03-26-11.30.40.646008-240 E8240E535           LEVEL: Event
PID     : 26689                TID  : 140093469746944PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000          DB   : SAMPLE
APPHDL  : 0-8                  APPID: *LOCAL.DB2.120326153040
AUTHID  : DB2INST1
EDUID   : 26                   EDUNAME: db2stmm (SAMPLE) 0
FUNCTION: DB2 UDB, Self tuning memory manager, stmmLogGetFileStats, probe:565
DATA #1 : <preformatted>
New STMM log file (/home/db2inst1/sqllib/db2dump/stmmlog/stmm.0.log) created automatically.

2018-03-26-11.30.45.279811-240 E8776E487           LEVEL: Info
PID     : 26689                TID  : 140093448775424PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000          DB   : SAMPLE
APPHDL  : 0-14                 APPID: *LOCAL.DB2.120326153046
AUTHID  : DB2INST1
EDUID   : 31                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, buffer pool services, sqlbExtentMovementEntryPoint, probe:4829
DATA #1 : <preformatted>
Extent Movement started on table space 3
```

The ***sqlbExtentMovementEntryPoint*** message still has the timestamp of the outage. The preceding one is 5 s away. Also note the EDUID for the preceding message is 26 – different from the trap (31).

What Happened: Step 1

Examine the *"FODC_Trap"* path, locate the trap file for EDU 31

```
$ ls -l FODC_Trap_2018-03-26-11.30.45.417849
total 8784
-rw-r--r-- 1 db2inst1 db2iadm1   17811 Mar 26 11:30 03915407.000.locklist.txt
drwxr-x--- 2 db2inst1 db2iadm1    4096 Mar 26 11:30 26689.000.core
-rw-r--r-- 1 db2inst1 db2iadm1  223795 Mar 26 11:30 26689.31.000.apm.bin
-rw-r----- 1 db2inst1 db2iadm1    1193 Mar 26 11:30 26689.31.000.cos.txt
-rw-r--r-- 1 db2inst1 db2iadm1  209197 Mar 26 11:30 26689.31.000.db2pd.SAMPLE.txt
-rw-r--r-- 1 db2inst1 db2iadm1 1894480 Mar 26 11:30 26689.31.000.dump.bin
-rw-r--r-- 1 db2inst1 db2iadm1  164078 Mar 26 11:30 26689.31.000.rawstack.txt
-rw-r--r-- 1 db2inst1 db2iadm1   29693 Mar 26 11:30 26689.31.000.trap.txt
-rw-r--r-- 1 db2inst1 db2iadm1    2206 Mar 26 11:30 chkconfig.txt
-rw-r----- 1 db2inst1 db2iadm1 6291312 Mar 26 11:30 db2eventlog.000.crash
-rw-r--r-- 1 db2inst1 db2iadm1    3724 Mar 26 11:30 db2pd.bufferpools.SAMPLE.txt
-rw-r--r-- 1 db2inst1 db2iadm1    9883 Mar 26 11:30 db2pd.dbcfg.SAMPLE.txt
-rw-r--r-- 1 db2inst1 db2iadm1    8716 Mar 26 11:30 db2pd.dbmcfg.txt
-rw-r--r-- 1 db2inst1 db2iadm1     743 Mar 26 11:30 db2pd.dbptnmem.txt
-rw-r--r-- 1 db2inst1 db2iadm1    7213 Mar 26 11:30 db2pd.memory.SAMPLE.txt
drwxr-xr-x 2 db2inst1 db2iadm1    4096 Mar 26 11:30 OSSNAPS
-rw-r--r-- 1 db2inst1 db2iadm1   21376 Mar 26 11:30 procmaps.txt
```

What Happened: Step 2

Examine the contents of the trap file, especially the signal/calling stack.
Optionally, use "`c++filt`" to demangle the names on the stack.

```
$ cat 26689.31.000.trap.txt | c++filt > 26689.31.000.trap.txt.filtered

$ vi 26689.31.000.trap.txt.filtered
DB2 build information: DB2 v9.7.0.0 s090521 SQL09070
timestamp: 2018-03-26-11.30.45.445398
instance name: db2inst1.000
EDU name     : db2agent (SAMPLE) 0
EDU ID       : 31
Signal #11
…
<StackTrace>
---FUNC-ADDR---- ------FUNCTION + OFFSET------
00007F6A1804D109 ossDumpStackTraceEx + 0x01e5
00007F6A18047F2A OSSTrapFile::dumpEx(unsigned long, int, siginfo*, void*, unsigned long) + 0x00cc
00007F6A1AA6A2A9 sqlo_trce + 0x02eb
00007F6A1AAAB9B1 sqloEDUCodeTrapHandler + 0x0167
…
00007F6A19D3FCF6 sqlbAlterPoolAct(unsigned short, SQLP_LSN8*, SQLB_GLOBALS*) + 0x03f8
…
00007F6A19E0D5BB sqldmpnd(sqeAgent*, int, char*, SQLP_LSN8*, SQLD_RECOV_INFO*) + 0x01cb
00007F6A1AADD1A9 sqlptppl(sqeAgent*) + 0x02f7
00007F6A1969EE2A sqlpxcm1(sqeAgent*, SQLXA_CALL_INFO*, int) + 0x05ae
00007F6A1970EE5D sqlrrcom(sqlrr_cb*, int, int) + 0x0467
00007F6A19D16AC5 sqlbEMReduceContainers(SQLB_POOL_CB*, unsigned int, sqeBsuEdu*) + 0x0341
00007F6A19D15672 sqlbLockAndMoveExtents(SQLB_POOL_CB*, bool, unsigned int, sqeBsuEdu*) + 0x04be
00007F6A19D1B4C5 sqlbExtentMovementEntryPoint(sqeBsuEdu*, void*) + 0x00bf
00007F6A192708B3 sqleIndCoordProcessRequest(sqeAgent*) + 0x062b
```

44

The **"c++filt"** utility can be used to demangle stack symbols. During compilation, the C compiler always replaces routine names chosen by the developer with their "mangled" equivalents chosen and generated by the compiler. The "mangled" routine name looks somewhat similar to the original. Example:

Original: sqlbAlterPoolAct(unsigned short, SQLP_LSN8*, SQLB_GLOBALS*)

Mangled: _Z16sqlbAlterPoolActttP9SQLP_LSN8P12SQLB_GLOBALS

The purpose of "mangling" is to ensure that all routine names in the scope of the program being executed are **unique**. The disadvantage is that this process affects the readability of symbols. The "**c++filt**" utility can be used to revert to the original human readable routine names, i.e. demangle the routine names. This utility is a standard part of the C compiler (i.e. not part of Db2).

What Happened: Step 3

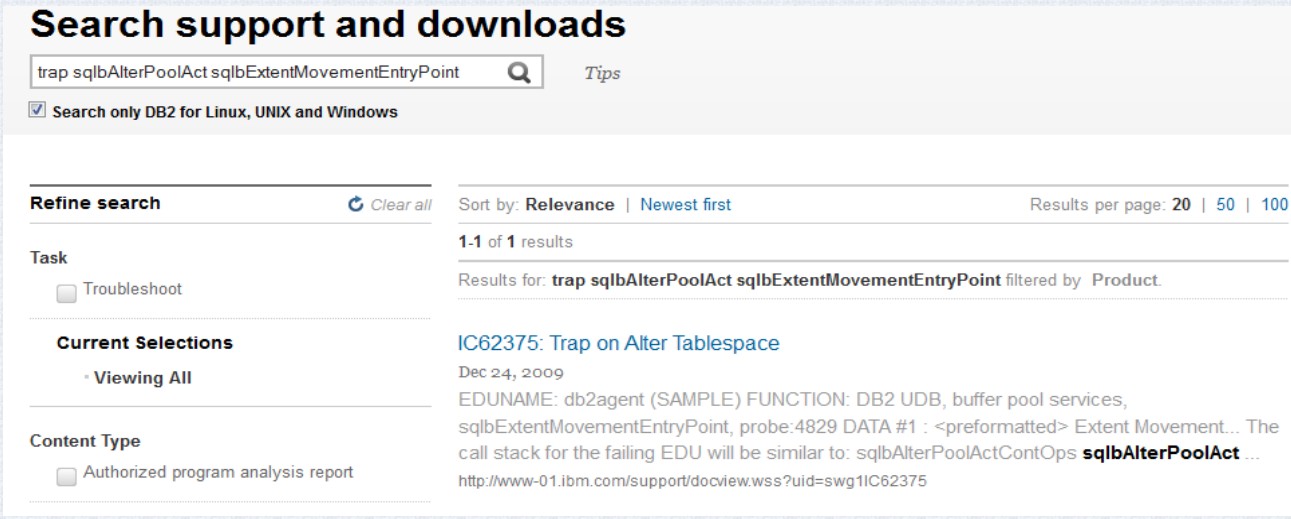
Time to summarize:

1. Trapped right after ALTER TABLESPACE – *possibly reproducible*?
2. An instance-wide trap.
3. Signal #11 (SIGSEGV) indicates an unexpected error.
4. The most recent routine on the stack is **sqlbAlterPoolAct** – *perhaps something to do with a pool (tablespace) alter?*
5. The first db2diag.log message is from routine **sqlbExtentMovementEntryPoint** – *perhaps something to do with Extent Movement?*
6. The EDU name is “db2agent”.

PRETTY GOOD DESCRIPTION => search for existing APARs!

What Happened: Step 4

Search for Db2 APARs at IBM Support Portal



https://www.ibm.com/support/home/

Kewords used: “trap sqlbAlterPoolAct sqlbExtentMovementEntryPoint”

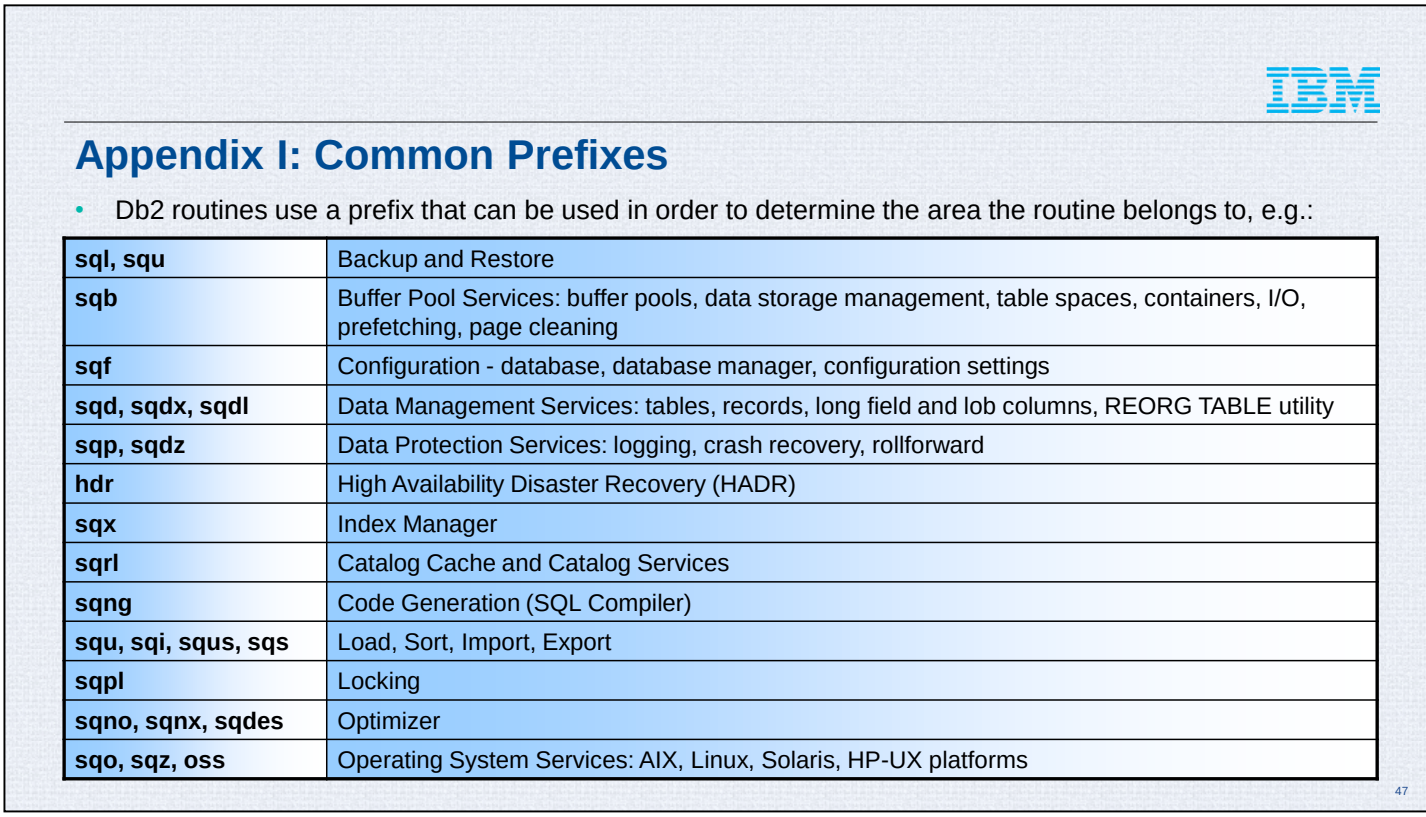
IC62375: Trap on Alter Tablespace

Error description
Issuing "ALTER TABLESPACE <name> MANAGED BY AUTOMATIC STORAGE"
on a table space with is already automatic-storage-enabled may
result in a panic on a subsequent ALTER TABLESPACE statement.
All processes associated with the instance will be terminated,
errors will be logged to db2diag.log, and a FODC_Trap
subdirectory will be created in the db2dump directory.

The db2diag.log will contain a message similar to:
2009-08-05-17.18.35.379272-240 E10400E482 LEVEL: Info
PID : 23098 TID : 47582189447488PROC :
db2sysc
INSTANCE: tomhart NODE : 000 DB : SAMPLE
APPHDL : 0-36 APPID: *LOCAL.DB2.090805211845
AUTHID : TOMHART
EDUID : 72 EDUNAME: db2agent (SAMPLE)
FUNCTION: DB2 UDB, buffer pool services,
sqlbExtentMovementEntryPoint, probe:4829
DATA #1 : <preformatted>
Extent Movement started on table space 3

The call stack for the failing EDU will be similar to:
sqlbAlterPoolActContOps
sqlbAlterPoolAct

storMgrAction
sqldmpnd



Note the symbolic names use add an extra ‘l’. For example, sqlbAlterPoolAct from the previous example has the prefix of ‘sqlb’, which translates to component ‘sqb’ – buffer pool services.

Appendix II: Core Files

- For UNIX-based systems, when Db2 terminates abnormally, a core file is generated by the operating system.
- Among other things, the core image will include most or all of Db2's memory allocations, which may be required for problem analysis.
- By default, Db2 core files are located in the path `$HOME/sqllib/db2dump/<core_directory>`, where `<core_directory>` is the core path directory name.
- If the corefile `ulimit` is set to unlimited, Db2 will override this with a smaller number unless instructed otherwise (DB2FODC). This will prevent filling up the file system if an outage happens and a core needs to be generated.
- A Windows equivalent of a UNIX-based core file is a process (mini)dump. Process dumps can be configured at the OS level or by using advanced debug techniques (`ADPlus`, `WinDbg`, `Userdump`).

48

There is one directory for each process. Directory names start with the letter "c", followed by the process identifier (pid) number of the affected process. A name extension provides the database partition number.

For example:

$HOME/sqllib/db2dump/c56772.010 is a directory containing a core file for the process with pid 56772 on partition 10.

Default core size on linux is zero, default on AIX is 1G, etc.
Cores generated through our trap handler are truncated to 2G if they are bigger;
Cores created outside of our trap handler can be bigger.

Appendix III – Sleeping Beauty

- For reproducible problems, sometimes it is useful to “freeze” the instance while the problem is happening, e.g. in order to attach a debugger
- An internal registry variable, **DB2SLEEP**, achieves just that

```
db2set DB2SLEEP=ON
```

- When enabled, **DB2SLEEP** suspends the instance after creating the FODC package, meaning the problematic process/EDU will still exist

- The sleeping instance can be resumed by

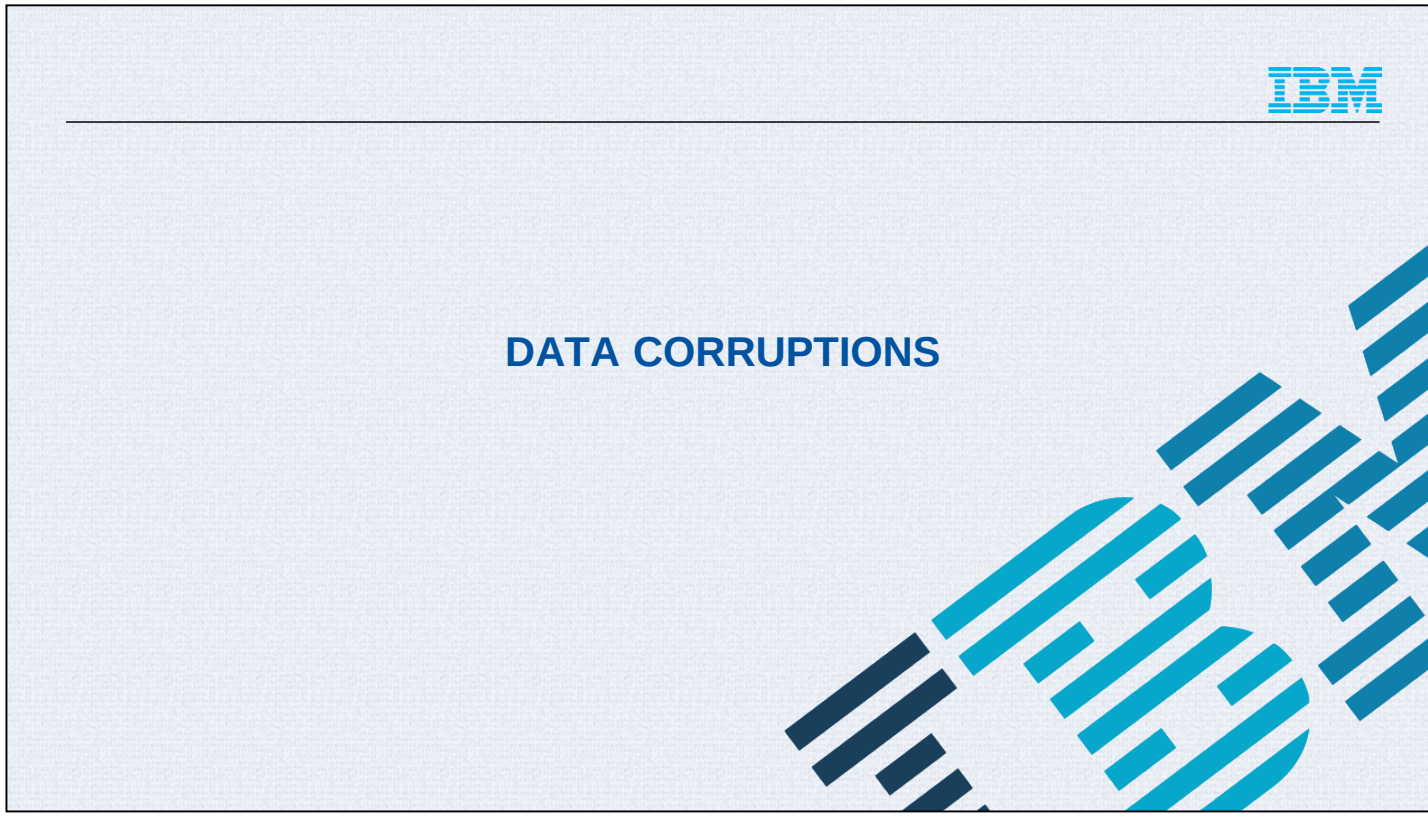
```
db2pdcfg -wakeupinstance
```

Use with caution!

Appendix IV – Debuggers

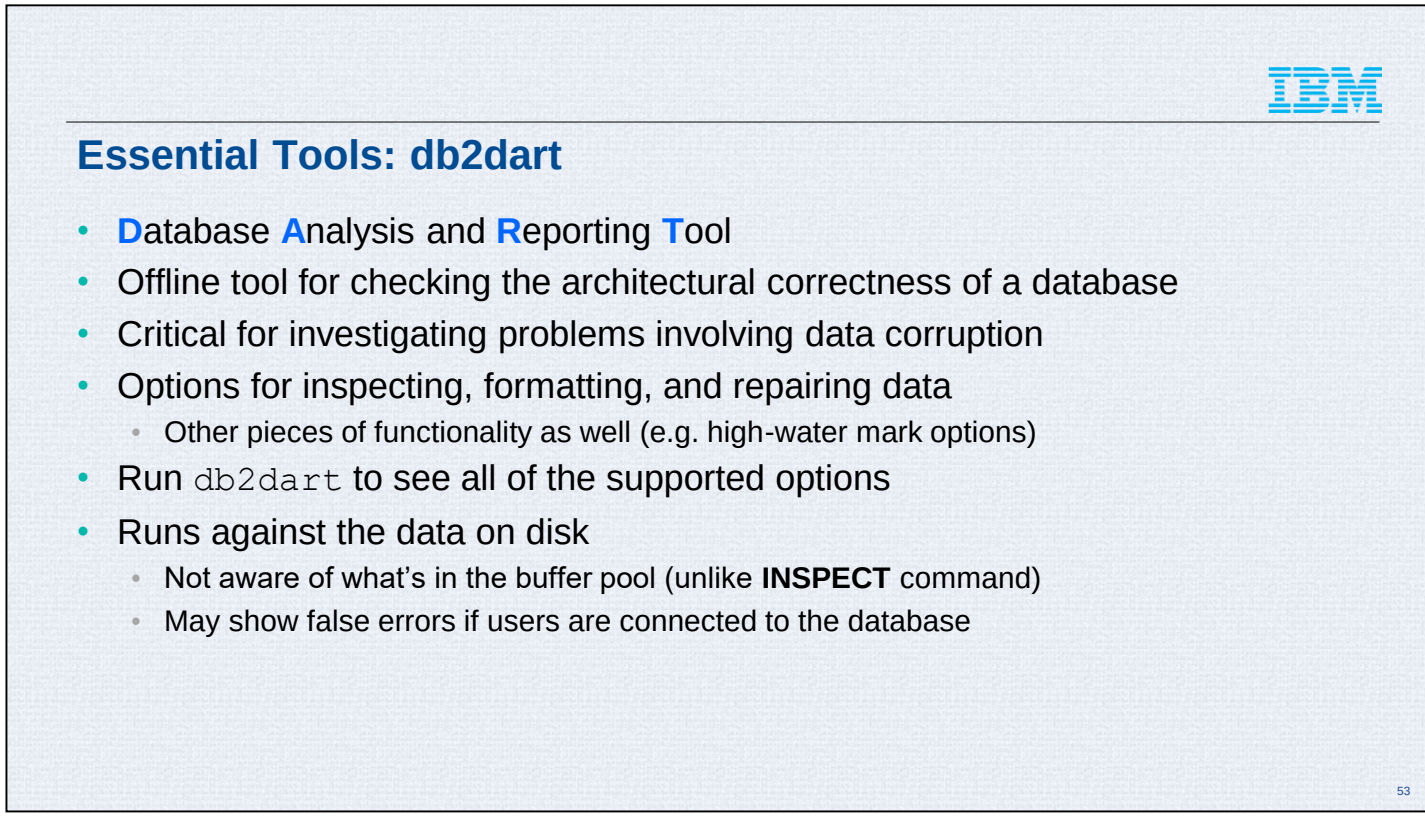
- Wiki: A debugger or debugging tool is a computer program that is used to test and debug other programs (the "target" program).
- Frequently used commands:

Action	dbx	gdb	Windbg
Attach to process	dbx [-a pid] prog [core]	gdb [prog[core|procID]]	windbg [-p pid | -z core | prog]
Call stack	where	bt, where	kb, kp, kd
Registers	registers	info registers	r
Loaded libraries	map	info sharedlibraries	lm
Running threads	thread	info threads	~
Switch thread	thread <tid>	thread <tid>	~ <tid>
Switch frame	frame <id>	frame <id>	.frame <id>
Examine memory	x <addr>/<fmt>	x/<fmt> <addr>	dw, db, dc <addr>
Disassemble	listi <addr>	disas <addr>	u <addr>
Print expression	print <exp>	print <exp>	? <exp>


DATA CORRUPTIONS

Abort/Panic Definition

- A **crash/abend** is a very generic term to describe any time Db2 comes down when it should not, i.e. an abnormal end
- A **panic** is a self-induced crash. Typically a panic occurs in error paths where there is no reasonable way to handle the error and continue operation. In the engine, panics typically end up invoking `sqle_panic()`. For example, we typically panic after reading a page from disk and discovering that its checksum is bad. Panics typically mark the database as bad.



db2dart should not be run while the database is up and running. Because it deals with data directly on disk, it is not aware of changes that may exist to pages in the buffer pool. As a result, it may give false errors. Similarly, if run on a database that is inconsistent (the database was brought down abnormally and requires crash recovery to be performed) then you may also see false errors in that case too.

The repair options of db2dart should only be done under the supervision of a Db2 Support analyst.

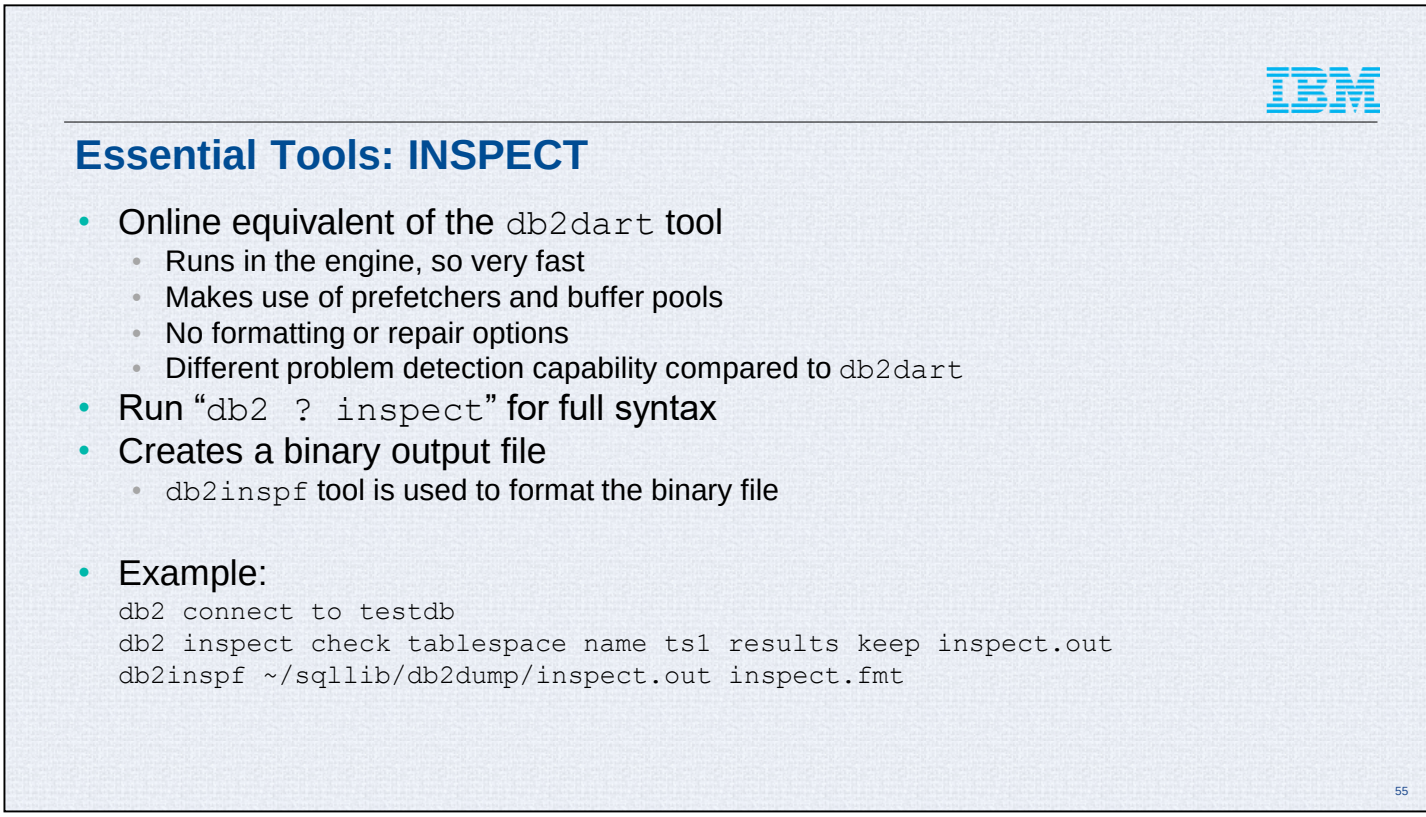
When targeting specific table spaces or tables, you have the choice to specify object identifiers on the command line, or you can wait to be prompted for them.

Essential Tools: `db2dart` Common Options

- Inspect at the database (/DB), table space (/TS), or table (/T) level
  - Default `db2dart` operation is a full database inspection
  - Checks validity of meta-data structures, data page/row headers, etc.
  - Does not verify the logical correctness of the data!
- Dump formatted table data in delimited ASCII format (/DDEL)
  - LOBS are excluded
- Mark index object as invalid (/MI)
  - Handy for damaged indexes that need to be marked for rebuilding

- Examples:

```
db2dart testdb
db2dart proddb /ts /tsi 4
db2dart sample /t /tsi 2 /oi 5
```



Comparison of INSPECT and db2dart

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.trb.doc/doc/c0020763.html

Essential Tools: `db2ckbkp`

- Primary purpose is to test the integrity of a backup image
  - Use when a backup image appears to be corrupt
  - May also run proactively whenever a backup is taken
  - Can also be used to display meta-data stored within the backup image, e.g. can show the storage paths associated with an automatic storage database
- Run "`db2ckbkp`" to see the various options
- Example:

```
db2ckbkp MYDB.0.db2inst.NODE0000.CATN0000.20180221223624.001
```

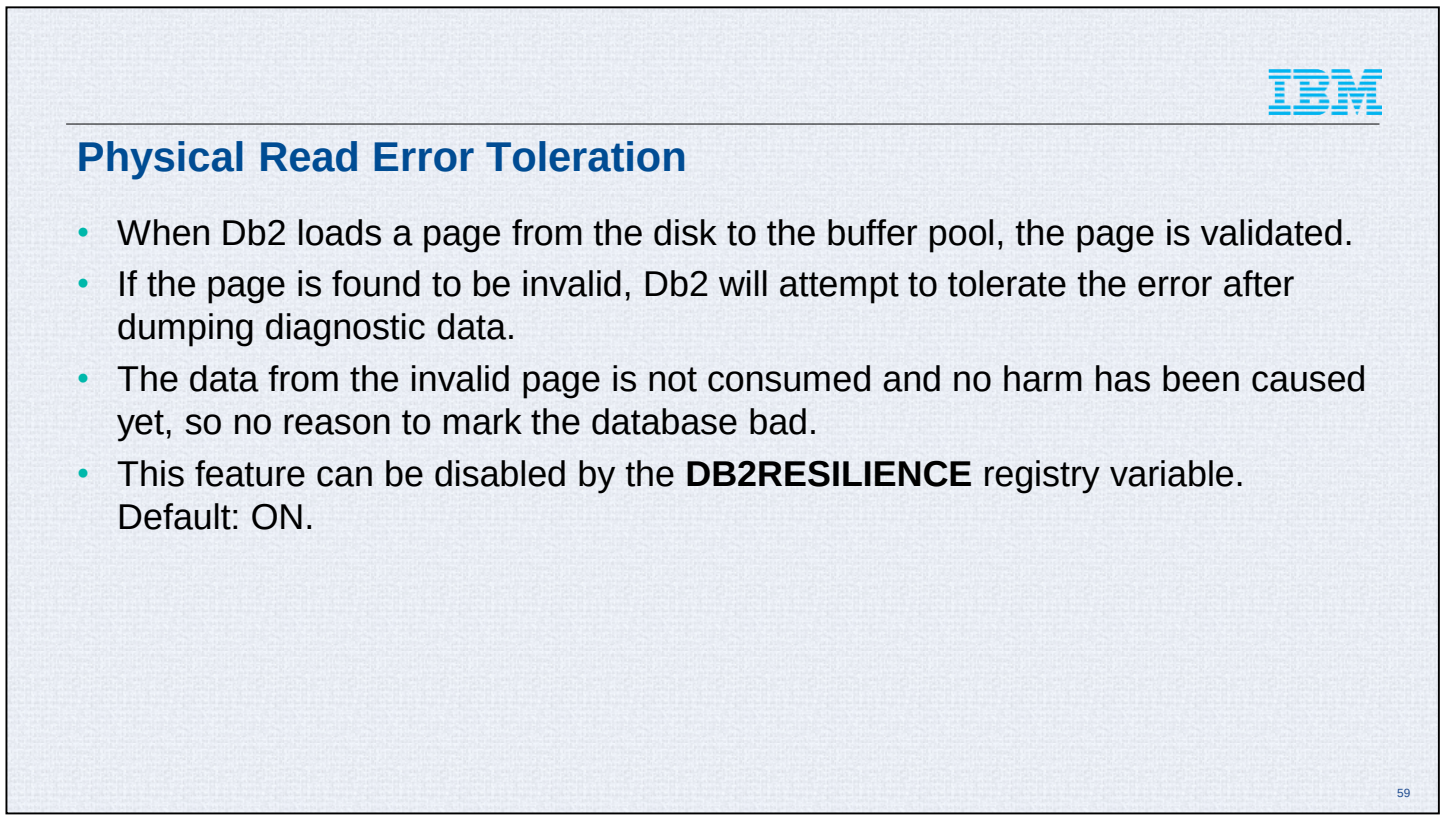
- For TSM backup images, use the VERIFY option of `db2adutl`

Physical Corruption

- Data is physically damaged. For example, a database page contains nothing but zeroes *(which is not allowed – a header and some other metadata are required for every database page).*
- Can be detected easily by running tools such as `db2dart`, **INSPECT**, or even by a visual inspection *(if it is known where the corruption may be located).*
- A frequent symptom is page verification errors (CBIT or others) reported in db2diag.log while reading the page from the disk.
- The root cause can vary: Db2 bugs **(almost never for CBIT problems though!)**, file system bugs, OS problems, HW issues, …

Logical Corruption

- Data is physically correct. However, there is a mismatch between what Db2 “thinks” the page contains (metadata) and the actual contents of the page.
- Typical examples: index page pointing to an incorrect root, an incorrect number of data slots on a data page, etc.
- The failure only happens during runtime. No problems found during `db2dart` or other inspections.
- The root cause is usually a Db2 bug. These kinds of problems usually cannot be attributed to problems outside of Db2 *(however, exceptions have been noted ☺)*.



•Customers may experience unexpected Db2 outages

•These outages cause loss of business and inconvenience

•The goal is to reduce unexpected outages and increase data availability

Logical and Physical Read Error Toleration

Avoid business outages due to file container corruptions

Avoid business outages due to unexpected database read errors

Upon Detecting Physical Page Error

1. Create the usual FODC package (call stacks, etc...).
2. Terminate the application with SQL1655 (new SQL code). The current operation fails, but the application can continue to use the same connection.
3. Log ADM6006E (admin log) with the page details.
4. In order to prevent diagnostic log flooding, limit the logging of the same error (currently to once every five minutes).
5. Database remains accessible.

Logical Read Error Toleration

- For logical reads, consumers of buffer pool pages perform validation on various internal structures that are stored in the page.
- If the page is found to be invalid, Db2 will attempt to tolerate the error after dumping diagnostic data.
- If the page is not modified yet (not “dirty”), the page will be marked bad. In this case the page in unloaded (“unfixed”) from the buffer pool and, if required, reloaded again.
- This feature can be disabled by the **DB2RESILIENCE** registry variable. Default: ON.

Upon Detecting Bad In-memory Page

1. Create the usual FODC package (call stacks, etc...).
2. Terminate the application with SQL1656 (another new SQL code, different from the physical read error code). The current operation fails, but the application can continue to use the same connection.
3. Log ADM6007E (admin log; different from the physical read error ADM code) with the page details.
4. In order to prevent diagnostic log flooding, limit the logging of the same error (currently to once every five minutes).
5. Database remains accessible.

Abort Signals/Exceptions

UNIX/Linux Signal IDs	Description
most UNIX's: SIGABRT(6) **HP-UX: SIGIOT(6)**	Instance panic. Self induced by Db2 due to unrecoverable problems. Typically associated with data (disk) corruption. The instance shuts down.

Windows Exception	Description
User Defined Exception (0xE0000002)	Diagnostic info signal. Dumps diagnostic info for the failing EDU. The instance shuts down during subsequent processing.

63

•On UNIX, a signal can be sent to a Db2 process by issuing a "kill - <signal #>. Signals are defined in the "signals.h" header file.

•For example, on AIX 5.3, the signal.h header file is located in /usr/include.sys/signal.h

•An extract of the signal.h header file is as follows:

```
#define  SIGHUP     1  /* hangup, generated when terminal disconnects */
#define  SIGINT     2  /* interrupt, generated from terminal special char */
#define  SIGQUIT    3  /* (*) quit, generated from terminal special char */
#define  SIGILL     4  /* (*) illegal instruction (not reset when caught)*/
#define  SIGTRAP    5  /* (*) trace trap (not reset when caught) */
#define  SIGABRT    6   /* (*) abort process */
#define SIGEMT    7  /* EMT intruction */
#define  SIGFPE     8  /* (*) floating point exception */
#define  SIGKILL    9  /* kill (cannot be caught or ignored) */
#define  SIGBUS     10  /* (*) bus error (specification exception) */
….
….
```

• To send an abort signal (SIGABRT) to a process, issue a "kill -6 <pid>".

• On Windows, use db2pd –stack to send "signals" to db2 processes/threads.

• WARNING: DO NOT randomly issue signals to a Db2 process unless directed to by Db2 Service. Sending inappropriate signals can lead to database problems.


CORRUPTION EXERCISE

Scenario

```
$ db2 "connect to sample"

   Database Connection Information

 Database server        = DB2/LINUXX8664 9.7.0
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE

$ db2 "create table t1 (i1 int, c2 char(250), c3 char(250), c4 char(250), c5 char(250))"
DB20000I  The SQL command completed successfully.

$ db2 "import from 'dataFile01.del' of del messages /dev/null insert into t1"
Number of rows read         = 99
Number of rows skipped      = 0
Number of rows inserted     = 99
Number of rows updated      = 0
Number of rows rejected     = 0
Number of rows committed    = 99

$ db2 "create index i1 on t1 (i1,c2,c3,c4,c5)"
DB20000I  The SQL command completed successfully.

$ db2 +c "create index i2 on t1 (i1,c3,c2,c4,c5)"
DB20000I  The SQL command completed successfully.
```

Scenario (cont'd)

```
$ db2 +c "insert into t1 values"\
>       "(1,'1','1','1','1'),"\
…
>       "(25,'25','25','25','25')"
DB20000I  The SQL command completed successfully.

$ db2 "rollback work"
DB20000I  The SQL command completed successfully.

$ db2 "terminate"; db2stop; db2start
SQL1063N  DB2START processing was successful.

$ db2 "connect to sample"

 Database server        = DB2/LINUXX8664 9.7.0
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE

$ db2 "delete from t1 where i1 >= 1000"
DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:
SQL1034C  The database is damaged.  All applications processing the database
have been stopped.  SQLSTATE=58031
```

Scope of Outage: Step 1

We can use the same commands as in the Abend exercise:

```
$ db2 list active databases
SQL1032N  No start database manager command was issued.  SQLSTATE=57019

$ db2nps 0
Node 0
     UID        PID       PPID     C     STIME     TTY      TIME CMD

$ db2pd -edus
Unable to attach to database manager on partition 0.
Please ensure the following are true:
   - db2start has been run for the partition.
```

Conclusion: The entire database manager is down. This is an instance abend (not a database one). Will need to run `db2start` eventually.

Scope of Outage: Step 2 - DIAGPATH

```
$ db2 get dbm cfg | grep DIAGPATH
 Diagnostic data directory path               (DIAGPATH) = /home/db2inst1/sqllib/db2dump

$ ls -l /home/db2inst1/sqllib/db2dump
total 16580
-rw-r---r-- 1 db2inst1 db2iadm1   31672 Mar 27 12:51 35989563.000.locklist.txt
-rw-r---r-- 1 db2inst1 db2iadm1   31672 Mar 27 12:51 55676162.000.locklist.txt
-rw-r---r-- 1 db2inst1 db2iadm1   31672 Mar 27 12:51 85463469.000.locklist.txt
-rw-r---r-- 1 db2inst1 db2iadm1   31672 Mar 27 12:51 92542329.000.locklist.txt
-rw-r---r-- 1 db2inst1 db2iadm1   22830 Mar 27 12:51 9321.17.000.apm.bin
-rw-r---r-- 1 db2inst1 db2iadm1 8849752 Mar 27 12:51 9321.17.000.dump.bin
-rw-r---r-- 1 db2inst1 db2iadm1  969013 Mar 27 12:51 9321.17.000.stack.txt
-rw-r---r-- 1 db2inst1 db2iadm1   61208 Mar 27 12:51 9321.19.000.dump.bin
-rw-r---r-- 1 db2inst1 db2iadm1   58992 Mar 27 12:51 9321.22.000.dump.bin
-rw-r---r-- 1 db2inst1 db2iadm1   27839 Mar 27 12:51 9321.22.000.stack.txt
-rw-rw-rw- 1 db2inst1 db2iadm1  482720 Mar 27 12:51 db2diag.log
-rw-r----- 1 db2inst1 db2iadm1 6291312 Mar 27 12:50 db2eventlog.000
-rw-rw-rw- 1 db2inst1 db2iadm1    7229 Mar 27 12:51 db2inst1.nfy
drwxrwxr-t 2 db2inst1 db2iadm1    4096 Mar 27 11:46 events
drwxr-x--- 2 db2inst1 db2iadm1    4096 Mar 27 12:51 FODC_DBMarkedBad_2018-03-27-12.51.22.987481
drwxr-x--- 3 db2inst1 db2iadm1    4096 Mar 27 12:51 FODC_IndexError_2018-03-27-12.51.21.074640_9321_17_000
drwxr-x--- 4 db2inst1 db2iadm1    4096 Mar 27 12:51 FODC_Panic_2018-03-27-12.51.23.883825
drwxrwxr-t 2 db2inst1 db2iadm1    4096 Mar 27 11:41 stmmlog
```

Conclusion: The presence of “FODC_IndexError” is indicative of an index issue. The other two “FODC” directories are a bit newer, and they were likely created as the consequence of the index error.

Time of Outage

Use the same technique as in the Abend exercise:

```
2018-03-27-12.51.04.267088-240 E23093E469          LEVEL: Event
PID     : 9321                 TID  : 139769593980672PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000          DB   : SAMPLE
APPHDL  : 0-8                  APPID: *LOCAL.DB2.120327165104
AUTHID  : DB2INST1
EDUID   : 26                   EDUNAME: db2stmm (SAMPLE) 0
FUNCTION: DB2 UDB, Self tuning memory manager, stmmLog, probe:1008
DATA #1 : <preformatted>
Starting STMM log from file number 0

2018-03-27-12.51.20.996688-240 I23563E543          LEVEL: Severe
PID     : 9321                 TID  : 139769631729408PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000          DB   : SAMPLE
APPHDL  : 0-7                  APPID: *LOCAL.db2inst1.120327165102
AUTHID  : DB2INST1
EDUID   : 17                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, index manager, sqliCleanupEmptyLeaf, probe:1213
RETCODE : ZRC=0x87090054=-2029453228=SQLI_PRG_ERR "Program error"
          DIA8575C An index manager programming error occurred.
```

It appears that the first relevant message is the *sqliCleanupEmptyLeaf* error reported by a db2agent with the EDUID of 17.

What Happened: Step 1

Examine the *"FODC_IndexError"* path, locate the trap file for EDU 17:

```
$ ls -l FODC_IndexError_2018-03-27-12.51.21.074640_9321_17_000
total 2292
-rw-r--r-- 1 db2inst1 db2iadm1   31672 Mar 27 12:51 72532410.000.locklist.txt
-rw-r--r-- 1 db2inst1 db2iadm1  227682 Mar 27 12:51 9321.17.000.apm.bin
-rw-r--r-- 1 db2inst1 db2iadm1 2028000 Mar 27 12:51 9321.17.000.dump.bin
-rw-r--r-- 1 db2inst1 db2iadm1   29541 Mar 27 12:51 9321.17.000.stack.txt
drwxr-xr-x 2 db2inst1 db2iadm1    4096 Mar 27 12:51 DART0000
-rwxr-xr-x 1 db2inst1 db2iadm1     151 Mar 27 12:51 db2cos_indexerror_long
-rwxr-xr-x 1 db2inst1 db2iadm1    1408 Mar 27 12:51 db2cos_indexerror_short
```

Also note the presence of the *"DART0000"* directory. This path contains automatic dumps of pages and structures pertinent to the outage. This information is extremely valuable to users with a deep understanding of Db2's data layout.

What Happened: Step 2

Examine the contents of the trap file, especially the signal/calling stack.
Optionally, use "`c++filt`" to demangle the names on the stack.

```
$ cat 9321.17.000.stack.txt | c++filt > 9321.17.000.stack.txt.filtered

$ vi 9321.17.000.stack.txt.filtered
DB2 build information: DB2 v9.7.0.0 s090521 SQL09070
timestamp: 2018-03-27-12.51.21.221030
instance name: db2inst1.000
EDU name     : db2agent (SAMPLE) 0
EDU ID       : 17
Signal #12
…
<StackTrace>
---FUNC-ADDR---- ------FUNCTION + OFFSET------
00007F1EAFAE6109 ossDumpStackTraceEx + 0x01e5
00007F1EAFAE0F2A OSSTrapFile::dumpEx(unsigned long, int, siginfo*, void*, unsigned long) + 0x00cc
00007F1EB25032A9 sqlo_trce + 0x02eb
00007F1EB254473A sqloDumpDiagInfoHandler + 0x00e0
…
00007F1EB4F38936 pthread_kill + 0x0036
00007F1EB25442A9 sqloDumpEDU + 0x0045
00007F1EB186FD99 sqldDumpContext(sqeBsuEdu*, int, int, int, int, int, char const*, …) + 0x068f
00007F1EB134DB69 sqlidelk(sqeAgent*, SQLD_IXCB*, SQLI_IXPCR*, SQLD_KEY*, SQLZ_RID, …) + 0x09b3
00007F1EB0CB235C sqldKeyDelete(SQLD_DFM_WORK*, SQLD_TCB*, SQLD_TDATAREC*, SQLZ_RID, …) + 0x0478
00007F1EB0CB1237 sqldRowDelete(sqeAgent*, SQLD_CCB*, unsigned long) + 0x039f
00007F1EB125091E sqlridel(sqlrr_cb*) + 0x00c6
```

What Happened: Step 3

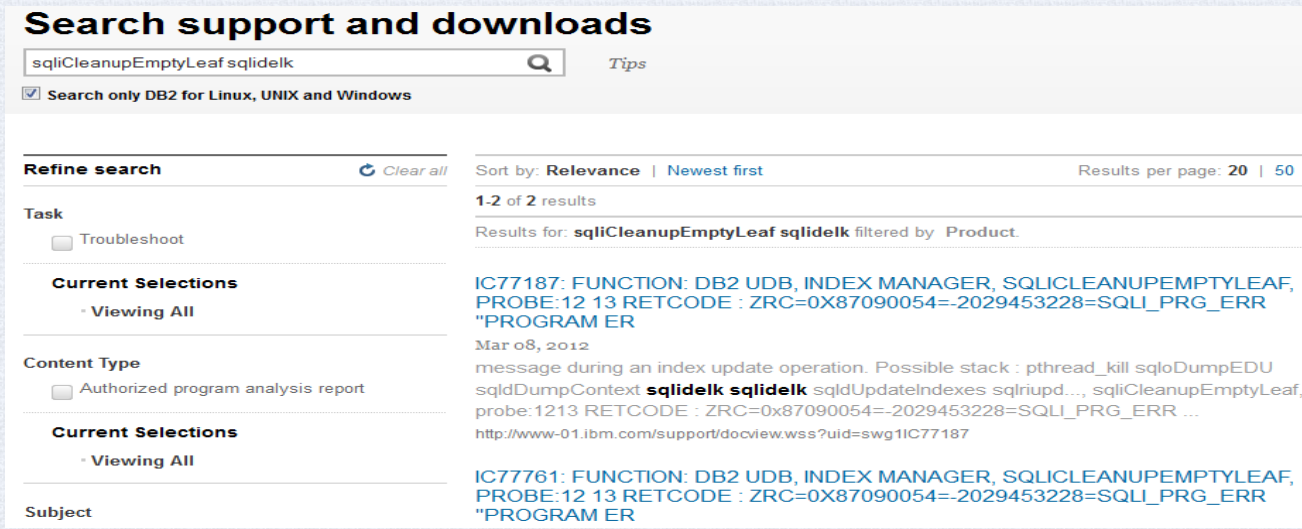
So far the investigation has been quite similar to Abend:

1. Trapped when deleting from table T1 – *possibly reproducible*?
2. An instance-wide trap.
3. Signal #12 (SIGUSR2) means a self-inflicted death.
4. The most recent routine on the stack is **sqlidelk** – *perhaps something to do with deleting an index key?*
5. The first db2diag.log message is from routine **sqliCleanupEmptyLeaf** – *definitely an index key delete!*
6. The EDU name is “db2agent”.

PRETTY GOOD DESCRIPTION => search for existing APARs!

What Happened: Step 4

Search for Db2 APARs at IBM Support Portal



Kewords used: “sqliCleanupEmptyLeaf sqlidelk”

IC77761: FUNCTION: DB2 UDB, INDEX MANAGER, SQLICLEANUPEMPTYLEAF, PROBE:12 13 RETCODE : ZRC=0X87090054=-2029453228=SQLI_PRG_ERR "PROGRAM ER

Error description
a Db2 instance can encounter a programming error message during an index update operation.

Possible stack :
pthread_kill
sqloDumpEDU
sqldDumpContext
sqlidelk
sqlidelk
sqldUpdateIndexes
sqlriupd

db2diag.log:
2011-06-15-14.12.56.240728+120 I103233A541 LEVEL: Severe
PID : 11075798 TID : 22950 PROC : db2sysc 0
INSTANCE: db2inst1 NODE : 000 DB : SAMPLE
APPHDL : 0-6656 APPID: *LOCAL.SAMPLE.110615080414
AUTHID : db2inst1
EDUID : 22950 EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, index manager, sqliCleanupEmptyLeaf, probe:1213
RETCODE : ZRC=0x87090054=-2029453228=SQLI_PRG_ERR "Program

Local fix
To avoid hitting problem, the customer should COMMIT any CREATE INDEX statements immediately after the CREATE INDEX statement has completed, instead of including the CREATE INDEX as a part

of a larger transaction.

BUT WAIT! LIMITED TIME OFFER!

This error is reproducible. An attempt to access the T1 table may result in another outage of the same kind. In many index or data corruption cases, we have an option to repair the object!

The BIG ONE: `db2dart /MI`

```
$ db2dart /H

 db2dart - Database Analysis Tool

    The db2dart command analyses databases, table spaces and tables.
    The primary function of this command is to examine databases for
    architectural correctness, and to report any encountered errors.

  Syntax:
    db2dart <database name> [action] [options ...]

 Repair actions:
  Make sure the database is offline for these actions.
    /MI    Marks index object as invalid.
           (Database must be offline. See notes 1, 5)
```

Repair: What?

Time to go back to db2diag.log. We need to find the object ID and table space ID of the damaged object (a good search string is “obj”):

```
2018-03-27-12.51.21.188119-240 I46737E568          LEVEL: Severe
PID     : 9321                 TID  : 139769631729408PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000          DB   : SAMPLE
APPHDL  : 0-7                  APPID: *LOCAL.db2inst1.120327165102
AUTHID  : DB2INST1
EDUID   : 17                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, trace services, sqlt_logerr_string (secondary logging fu, probe:0
DATA #2 : String, 108 bytes
Index object = {TBSPACEID=<3>; OBJECTID=<5>; OBJECTTYPE=<INX>} Parent object = {TBSPACEID=<3>; OBJECTID=<5>}
…
2018-03-27-12.51.23.784098-240 I465004E2315        LEVEL: Severe
PID     : 9321                 TID  : 139769610757888PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000
EDUID   : 22                   EDUNAME: db2pclnr (SAMPLE) 0
FUNCTION: DB2 UDB, buffer pool services, sqlbClnrAsyncWriteSetup, probe:300
DATA #1 : Buffer page descriptor, PD_TYPE_SQLB_BPD, 152 bytes
  Pagekey: {pool:3;obj:5;type:1} PPNum:3
```

It appears that we are dealing with object 5 in table space 3.

Repair: Anything Else?

In the case of HW failures, it is always good to check if there are any more corruptions present. A full `db2dart` scan is the best option, but this is time consuming. Alternatively, /TS or /T switch can be used to limit the db2dart scope to a table space or table, respectively.

```
$ db2dart sample
…
 Table inspection start: DB2INST1.T1
         Data inspection phase start. Data obj: 5  In pool: 3
         Data inspection phase end.
         Index inspection phase start. Index obj: 5  In pool: 3
         Index inspection phase start. Index obj: 5  In pool: 3
         Error: Bad Index Token (22)
…
 DB2DART Processing completed with error!
```

Conclusion: In our case, index 5 in pool 3 is the only corrupted object.

Repair: JUST DO IT!

```
$ db2dart sample /MI /OI 5 /TSI 3
…
Modification for page (obj rel 0, pool rel 704) of pool ID (3) obj ID (5), written out to disk
   successfully.

$ db2start; db2 "connect to sample"; db2 "select count(*) from T1"
…
1
-----------
         99

  1 record(s) selected.

$ vi db2diag.log
…
2018-03-27-14.46.37.890316-240 E496451E488         LEVEL: Warning
PID     : 14147                TID  : 139799037994752PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000          DB   : SAMPLE
APPHDL  : 0-7                  APPID: *LOCAL.db2inst1.120327184632
AUTHID  : DB2INST1
EDUID   : 17                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, data management, sqldEndIndexCreate, probe:1
MESSAGE : ADM5542W  Indexes on table "DB2INST1.T1" are successfully rebuilt.
```


HANG/PERFORMANCE PROBLEMS

What Constitutes a Hang?

- A **hang** is a situation in which the database stops responding to incoming requests, or stops processing existing requests
- Typically happens due to a shared resource (lock, latch, … – see explanation on the subsequent slides) being held by somebody/something other than the requestor
- **Temporary**: A small "glitch" causing processing to slow down temporarily, often indistinguishable from a performance problem
- **Permanent**: Multiple execution threads competing about the same set of resources, a kind of a *"reversed standoff ☺"*

Performance Problem?

- Execution slower than usual, expected, or *previously established*
- Investigating performance problems is mostly identical to investigating hangs

Q: What is the main difference between a hang and a performance problem?

A: If experiencing a performance issue, things will eventually get done. Slowly but surely ☺

80

•A frequent mistake is to “feel” that execution should be faster. Rather than guessing, it is important to establish performance baselines and targets. Careful documentation and planning of system and configuration changes never hurts, either. A great idea is to determine and document query execution plans for the baseline.

All About Latching

- What is a **latch**?
  - When multiple processes or threads are trying to access the same shared information at the same time, it is necessary to control access to that information
  - This is done via an internal mechanism called a **latch**. The way it works is that one EDU can acquire a latch on a particular resource, and if another EDU also wants to access that same information, the other EDU must wait until the latch is freed by the first EDU before access is allowed.
  - In IBM Db2 pureScale ®, a cross-member latch which requires both the global lock portion and the local latch portion is called **lotch** (abbreviated from *"lock or latch"*)
- What is a **deadlatch**?
  - A type of a hang (usually defect-related) caused by resource contention involving latches. An EDU is asking for a latch that is already owned, and the owner is either unable to make progress, or is waiting for a resource owned by the current EDU. Remember the standoff?

81

Think of a latch as a low level lock within Db2.

Diagnostic Data

- The difficultly with hang/performance problems is that it is critical to generate diagnostic information *while the problem is happening*
- Looking at a hang/performance issue in the post-mortem sense without collecting information ***during the problem's occurrence*** will almost always lead to a dead end
- To make matters more complicated, sometimes a *"Db2 problem"* is in fact a problem outside of Db2. It could be:
  - Application issues
  - Network problems
  - Operating system/hardware glitches
  - Etc...

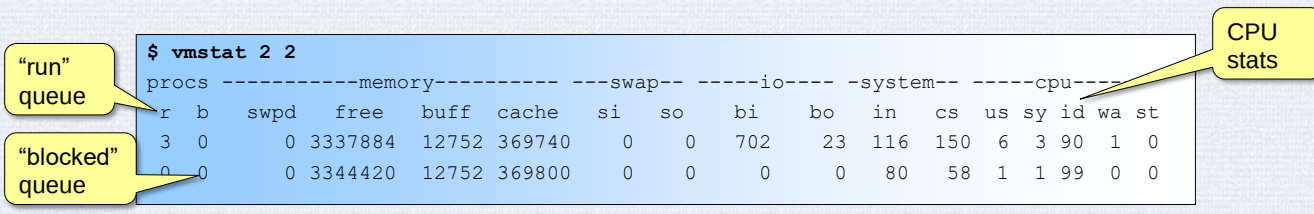
Top-Down Approach

- Arguably the most critical part is to narrow down the scope of occurrence. One of the many possible approaches is:

1. Examine the operating system first. Look for excessive I/O, network traffic, errors in the system error logs, …
2. Look at the Db2 instance scope. Try to determine if the problem is affecting the entire instance. Are instance level commands working well?
3. Examine the Db2 database scope. Problems with all databases? Or some?
4. EDU or application scope. All applications affected? Or some? What about non-agent EDUs such as prefetchers or page cleaners?

Operating System Scope

Command	What does this tell you?
Log into the system via `ssh`, `telnet`, etc…, and pick any OS command such as `ls`, `ps`, …	If user login and/or OS commands take a long time to run, the problem is likely system-wide, i.e. not a Db2 problem.
`vmstat 2 10`	Any system issues, e.g. high CPU or paging?
`iostat 2 10`	Any busy storage devices? Note: `iostat` might not be pre-installed with the OS.



84

r: The number of processes waiting for run time.

b: The number of processes in uninterruptible sleep.

CPU

These are percentages of total CPU time.

us: Time spent running non-kernel code. (user time, including nice time)

sy: Time spent running kernel code. (system time)

id: Time spent idle. Prior to Linux 2.5.41, this includes IO-wait time.

wa: Time spent waiting for IO. Prior to Linux 2.5.41, included in idle.

st: Time stolen from a virtual machine. Prior to Linux 2.6.11, unknown.

Instance Scope

Command	What does this tell you?
`db2 get dbm cfg` `db2 get db cfg for <dbname>`	These two commands are simple and acquire very few (if any) latches. Useful in narrowing down the scope.
`db2 list active databases`	A quick way to tell if the instance is hanging.
`db2 list applications show detail`	Any application activity reported?
`db2pd -edus`	Especially check the CPU stats to see if any EDUs show unusually large counters.

Q: Why are we suggesting the “old” snapshot interface here?
A: Unlike the new and more capable MON_GET.* table functions, the snapshot interface uses an infrastructure which runs “somewhat” outside of the Db2 kernel. The advantage is that no database connection is needed for these instance-level commands.

Database Scope

Command	What does this tell you?
`db2 connect to <dbname>`	Combined with `db2_all` as necessary, this is the simplest way to confirm database connectivity.
`MON_GET_DATABASE_DETAILS`	A more comprehensive way to look up database-level statistics.
`db2 get snapshot for database on <dbname>`	The simpler, “snapshot-way”, of achieving the same.
Having connected to your database, verify that you can execute queries from the command line.	Sometimes the problem may only be specific to an application. By verifying if things are working from the CLP you can narrow down if the problem is with an application or with the database.

Application/EDU Scope

Command	What does this tell you?
Do Db2 snapshots complete? For example: `db2 get snapshot for applications on <dbname>`	If a snapshot hangs, it likely means the problem is an instance-wide hang, possibly due to latching issues. If a snapshot completes, then the problem may be more specific to a certain database or application connection and is less likely to be latching related.
Are snapshots showing “movement”? Get more than one snapshot with some time in between, say one minute apart. Look for changes (deltas) between the snapshots.	If read/written or other signs of activity by comparing two snapshots, then you may be dealing with a performance problem instead of a real hang issue.

The Big One – Internals

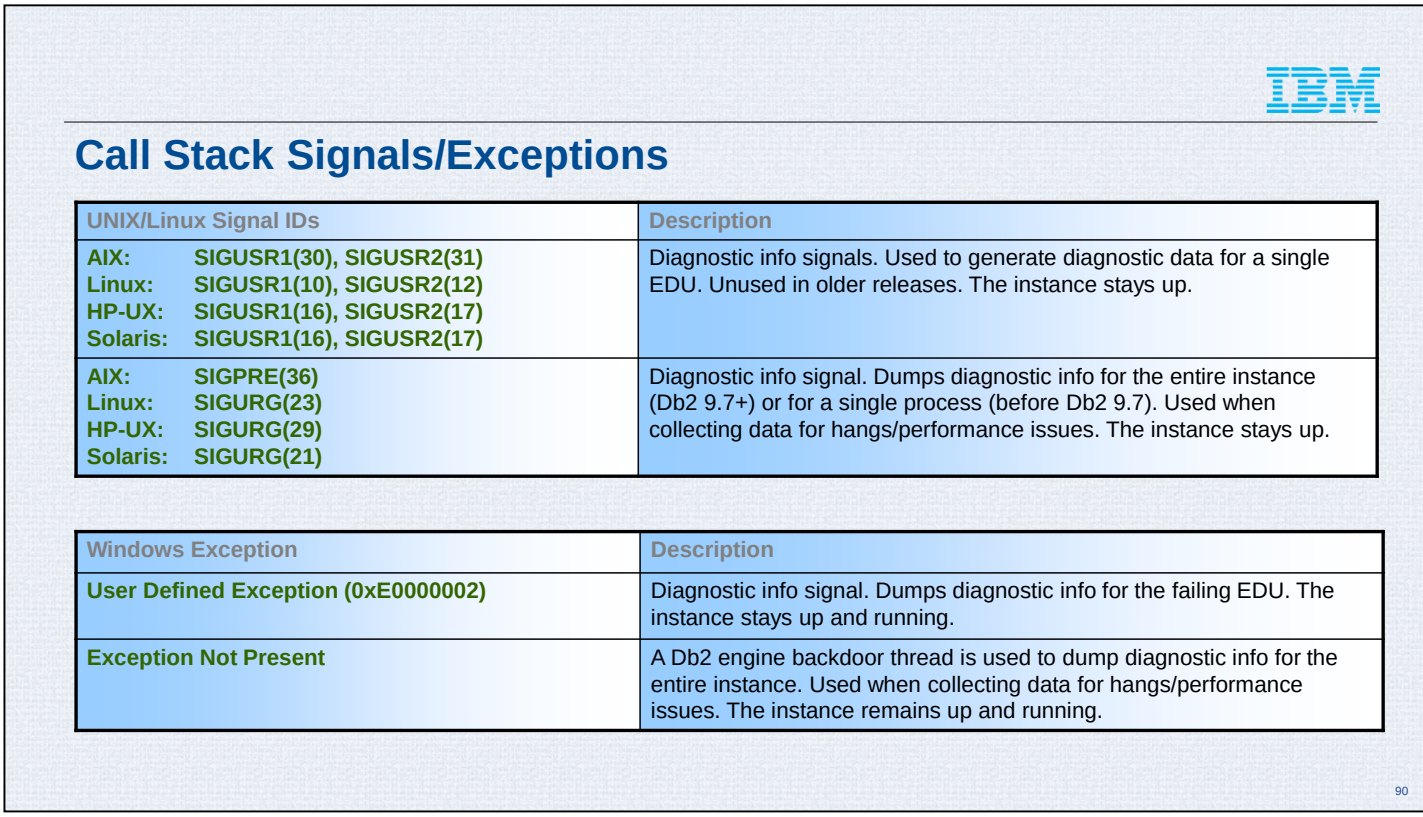
- After we have established that this is not an OS hang, a network outage, i.e. we are most likely dealing with a Db2 or an application issue, it is time to collect the most important pieces:

1. Call stacks
2. Data for active EDUs
3. Traces, including performance traces and/or profiling information
4. Lock information*
5. Access plans*

** out of scope for this presentation*

Call Stacks

- A **call stack** is a snapshot for the state of a Db2 process at the time for when the stack dump was captured. Call stack files are also known as “trap files” *(although in a slightly different context)*, "stack trace backs“, or “stacks”.
- Generated automatically if processing cannot continue because of an exception, or for serviceability reasons by Db2. Can also be generated manually by the user (e.g. hangs, performance issues).
- Contains the function sequence that was running when the error occurred.
- Also contains information about the state of the process when the signal or exception was caught, e.g. contents of registers, disassembly of code around the failing line.



•On UNIX, a signal can be sent to a Db2 process by issuing a "kill - <signal #>. Signals are defined in the "signals.h" header file.

•For example, on AIX 5.3, the signal.h header file is located in /usr/include.sys/signal.h

•An extract of the signal.h header file is as follows:

```
#define  SIGHUP     1  /* hangup, generated when terminal disconnects */
#define  SIGINT     2  /* interrupt, generated from terminal special char */
#define  SIGQUIT    3  /* (*) quit, generated from terminal special char */
#define  SIGILL     4  /* (*) illegal instruction (not reset when caught)*/
#define  SIGTRAP    5  /* (*) trace trap (not reset when caught) */
#define  SIGABRT    6   /* (*) abort process */
#define SIGEMT    7  /* EMT intruction */
#define  SIGFPE     8  /* (*) floating point exception */
#define  SIGKILL    9  /* kill (cannot be caught or ignored) */
#define  SIGBUS     10  /* (*) bus error (specification exception) */
….
….
```

• To send an abort signal (SIGABRT) to a process, issue a "kill -6 <pid>".

• On Windows, use db2pd –stack to send "signals" to db2 processes/threads.

• WARNING: DO NOT randomly issue signals to a Db2 process unless directed to by Db2 Service. Sending inappropriate signals can lead to database problems.

Call Stacks – Naming Convention

Type of File	Name
call stack (UNIX/Linux)	<pid>.<eduid>.<node>.**stack.txt**
call stack (Windows)	<pid>.<tid>.**stack.bin**

- On UNIX/Linux, the file is text-based
- On Windows, the file is binary. In order to translate the binary file into text, a formatting utility (`db2xprt`) and debug symbol files are required, both of which are shipped with the product.

91

Stacks, if existing, will always reside in the Db2 diagnostic directory (DIAGPATH).

Call Stack Contents

- Build date
- Version Number
- Time of the dump
- Signal or Exception which generated the dump
- Process & thread ID
- Loaded libraries (common referred to as the “map”)
- Address of signal handlers
- Registry dumps
- **Call Stack – a detailed call stack**
- A dump of the OSS Memory sets
- **Latch information for the EDU**
- **Locks being waited on**
- Assembly code dump, …

The following information is critical to solving most hangs, and useful for performance issues, too:

1. Stack Trace
2. Latch Information
3. Lock Waited On

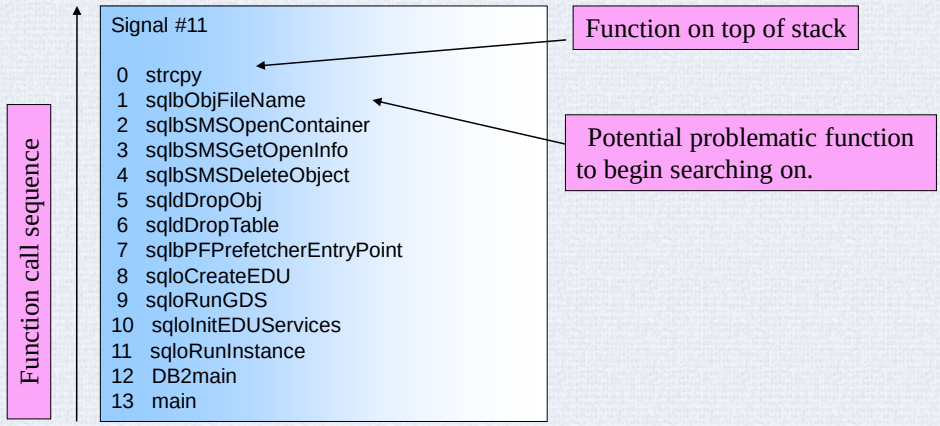
A good idea is to run **c++filt** (not a Db2 component, this is part of the C compiler) to demangle the stack symbols.

Example:

$ c++filt -p < 48347.106.001.stack.txt > 48347.106.001.stack.filtered.txt

Call Stack

How to read a call stack:



94

NOTE: In general, you would want to search for the function on the top of the stack.

•In this example, the function strcpy() is a C library function and is less likely to by the culprit (not impossible).

•The most likely culprit is the caller to the strcpy() function. The strcpy() function may just be the victim.

Generate Call Stacks – Db2 Ways

- All platforms:

```
db2pd -stack all                          (single partition)
rah db2pd -alldbp -stack all              (multiple)
```

- UNIX/Linux Alternatives (if `db2pd` is hanging):

```
AIX:                kill -PRE <db2sysc PID>
Solaris, Linux, HP: kill -URG <db2sysc PID>
```

- Windows Alternatives (if `db2pd` is hanging):

```
db2bddbg -d db2ntDumpTid "<path>" -1 <filename>
db2nstck
```

95

Note:

• On AIX for example, a kill -36 is equivalent to a kill – SIGPRE as defined in the signal.h file on AIX.

• Each UNIX platform will haves it own signals defined.

Generate Call Stacks – Native Ways

- The aforementioned *"Db2 way"* of generating call stacks is always preferred. For completeness, call stacks can be generated using other methods, too:
- **Debuggers**
  - `gdb`, `dbx` (both multi-platform)
  - `kdb` (AIX), `adb` (Solaris), `WinDbg` (Windows)
- **OS Commands**
  - Linux: `cat /proc/<pid>/stack`, cat `/proc/<pid>/task/<thread_id>/stack`
  - AIX: `procstack`
  - Solaris: `pstack`
  - Windows: `adplus`, `procstack`

96

The Db2 ways of obtaining call stacks are always preferred as these methods produce much more information than the native OS methods. However, sometimes it is useful to learn these approaches in case even the most fundamental Db2 commands such as "db2pd" become non-responsive.

Call Stack Analysis: `analyzestack`

- At this point we have multiple call stack files. Each file contains information pertinent to one EDU. It is a one time snapshot of what was happening in the EDU while the stack was generated.
- We need to take a look at each individual call stack. This could be a problem if there are thousands of EDUs running on the system.
- SOLUTION: `analyzestack`

```
$ ~/sqllib/pd/analyzestack -i .

<...>
Slurping 19 trap files ..
<...>

Please check the following files:
========================================
StackAnalysis.out

Analysis Complete ...
```

97

Note that **analyzestack** was first shipped in 9.7 Fix Pack 5. The tool is located under sqllib/pd. Frequently used options:

-i directory | file_list
Input can be a directory where trap files exist or list of trap files or a single trap files. You can use wildcard characters for the file list (but please remember to enclose the file list in quotes if you use them) or you can use the command as -i <file1> -i <file2> -i <file3> as well.

-l [output file]
Do latch analysis on the files. Output filename is optional. If not provided it defaults to "LatchAnalysis.out".

-m1 timestamp1 -m2 timestamp2
Analyze stacks for timestamps between <timestamp1> and <timestamp2> (both inclusive). The timestamps can be of format "yyyy-mm", "yyyy-mm-dd", "yyyy-mm-dd-hh" or "yyyy-mm-dd-hh.min" or "yyyy-mm-dd-hh.min.sec".

-d depth
Provide the number of functions to be compared in stacks (Current max is: 100 and default is 40).

StackAnalysis.out

- The output is a frequential analysis of individual stacks, grouped by stack patterns, e.g.:

```
Stack:
 ======================
  0x00007F0F6184721D _Z16sqlbClnrFindWorkP12SQLB_CLNR_CB + 0x2e22
  0x00007F0F6184BBF4 _Z18sqlbClnrEntryPointP12sqbPgClnrEdu + 0x026a
  0x00007F0F6185D8A0 _ZN12sqbPgClnrEdu6RunEDUEv + 0x00b8
  0x00007F0F7794E306 _ZN9sqzEDUObj9EDUDriverEv + 0x0232
  0x00007F0F7794E0C5 _Z10sqlzRunEDUPcj + 0x003c
  0x00007F0F74670336 sqloEDUEntry + 0x0c7d

 Summary:
 Found in 26 stacks of a total of 123 stacks in 123 files
 Found in:
   ./25228.100.001.stack.txt -- db2pclnr(TESTDB) -- 2018-08-21-11.49.21.966490(Signal #10)
   ./25228.75.001.stack.txt -- db2pclnr(TESTDB) -- 2018-08-21-11.49.21.970320(Signal #10)
   <...etc...>
```

98

Produced by **“analyzestack –i <dir_name>”**

LatchAnalysis.out

- A sorted overview of latch holders and waiters, e.g.:

```
========================================
******** LATCHWAIT DETECTED ( #1 )  ***********
Printing LatchWait information
========================================
<<<<   Holder Information (Address = 0x7f56a17f07e0)   >>>>
   File Information:
       Line: 359, File: sqlpgResSpace.C HoldCount: 1
----------------------------
   PID(s):
   233 (./63809.233.000.stack.txt) -- Line: 359, File: sqlpgResSpace.C HoldCount: 1 (SQLO_LT_SQLP_DBCB__add_logspace_sem)
   Agent Type:  db2agent (DTW)

*** Stack ***
Timestamp: 2018-03-28-10.57.34.408457
0x00007F5CBECDAE1F sqloWaitEDUWaitPost + 0x03bf
0x00007F5CB98EC1E3 _Z19sqlbCastoutForFlushP16SQLB_OBJECT_DESCjtP9SQLP_LSN8P22SQLB_OBJECT_PAGE_RANGEP12SQLB_GLOBALS + 0x0963
0x00007F5CB98EB012 _Z9sqlbFlushP9SQLP_LSN8jtP12SQLB_GLOBALS + 0x04c2
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<<<<   Waiter Information (Address = 0x7f56a17f07e0)  >>>>
   TOTAL WAITERS >> 71
    <...etc...>
```

99

Produced by "**analyzestack –i <dir_name> –l**" (that is, lower case L)

Data for Active EDUs

`db2pd -edus`	running EDUs
`db2pd -latches`	acquired latches
`db2pd -locks`	database locks (not latches!)
`db2pd -wlocks`	database locks being waited on (agent-level)
`db2pd -applications`	active applications
`db2pd -transactions`	running transactions
`db2pd -dynamic`	running SQL (Dynamic Cache)
`db2pd -dbcfg`	DB CFG (useful during hangs)
`db2pd -dbmcfg`	DBM CFG (likewise)
`db2level`	obviously ☺
`db2set -all`	never hurts

If a database hang is suspected, the goal is to avoid running commands that require a database connection. Depending on the type of a hang, such commands are likely to hang, too.

Db2 Trace Facility

- In problem determination it is sometimes very useful to have information on what was occurring on the system during the actual time of failure. A Db2 trace provides information on:
  - what functional calls were made (most recent at bottom of output),
  - the actual code path used,
  - and sometimes even the data being manipulated at each point within the function.

NOTE: We do not expect customers to analyze a trace. The following slides are for informational purposes only.

101

•Db2 traces are invoked by issuing the **db2trc** command from an operating system command prompt.

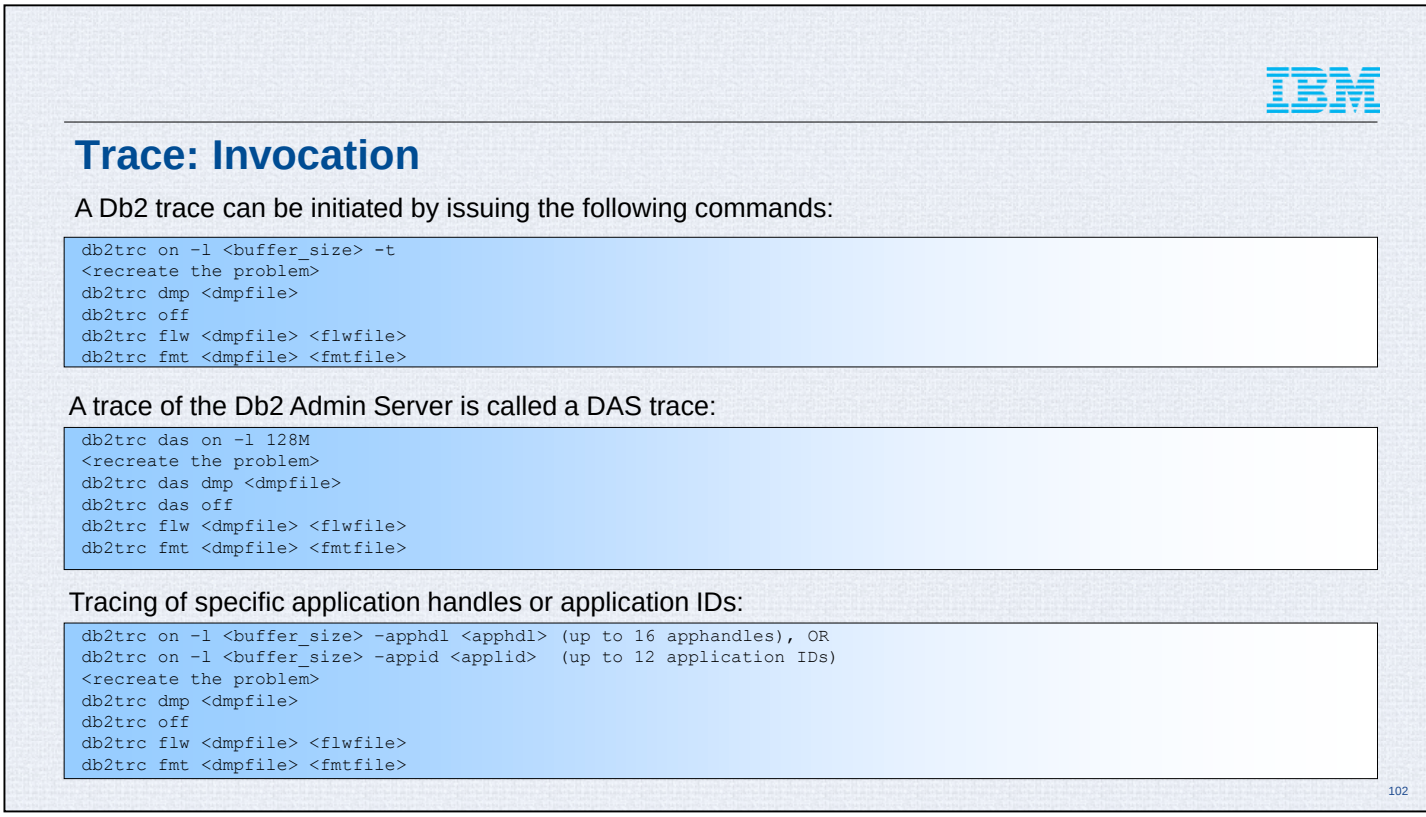
•When invoked, trace points within the Db2 source will 'fire' during runtime.

•The firing of each trace point causes information such as the location within the code, error codes, return codes, and certain variables to be written to a buffer. The size of the buffer is specified by the user that started the trace.

•The buffer is circular, meaning that once the trace utility has reached the bottom of the buffer it will wrap back up to the top. This option can be controlled of course.

•**db2trc** allows for administration of the facility and parsing and formatting of the trace dump files.

•You must reproduce the problem and it will affect performance.



-l [*bufferSize*]

This option specifies the size and behavior of the trace buffer. -l specifies that the last trace records are retained (that is, the first records are overwritten when the buffer is full). The buffer size can be specified in either bytes or megabytes. To specify the buffer size in megabytes, add the character M | m to the buffer size. For example, to start db2trc with a 4–megabyte buffer: db2trc on -l 4m The default and maximum trace buffer sizes vary by platform. The minimum buffer size is 1 MB. The buffer size must be a power of 2.

[-t]

Include timestamps.

Trace: Flow (FLW)

```
308986      sqleProcessSCoordRequest entry [eduid 37 eduname db2agent]
310069      | sqlpParallelRecovery entry [eduid 37 eduname db2agent]
              <...lots of other calls here...>
316955      | sqlpParallelRecovery exit [rc = SQLB_EMP_MAP_INFO_NOT_FOUND]
317046      sqleProcessSCoordRequest exit
```

- **Unique trace ID. Increasing order, trace always starts with 1.**
- **Db2 function called. Name chosen by Db2 developers, often self-explanatory.**
- **Specific place in function. Could be "entry", "exit", "probe number", "marker", …**
- **Db2 "thread" (EDU) ID and name. Matches the EDU ID and name in db2diag.log.**
- **Return code. A good string to search for in Db2 APARs.**

FLW provides a visual representation of which Db2 routines were called and by whom, their return code, markers, and probe points. The trace IDs are not sequential (i.e. contain "holes") because of context switching, i.e. EDU "A" may own entries 1 and 3, but EDU "B" running in parallel will own 2 and 4.

EDU is a Db2 term for "thread". Stands for "Engine Dispatchable Unit".

Trace: Format (FMT)

```
316955   exit DB2 UDB recovery manager sqlpParallelRecovery fnc (2.3.94.48.0) pid 14925 tid
   46912874998080 cpid 14546 node 0 rc = 0x8402001B =
         -2080243685 = SQLB_EMP_MAP_INFO_NOT_FOUND

316956   entry DB2 UDB base sys utilities sqleSubCoordTerm fnc (1.3.5.1051.0) pid 14925 tid
   46912874998080 cpid 14546 node 0 eduid 37 eduname
         db2agent
```

- **Unique trace ID. Matches the ID in FLW.**
- **Specific place in function. Could be "entry", "exit", "probe number", "marker", …**
- **Db2 area, component, and function called. Note the unique "IP address".**
- **Process/thread/EDU/Node ID, EDU name. Also could contain timestamp, etc…**
- **Return code. Same as in FLW.**

FMT provides additional detail about individual trace entries. Unlike FLW, the entries are perfectly sequential and ordered by time. When timestamps are present (`db2trc -t`), these entries could be used for performance measurements. Because of the aforementioned context switching, extra attention needs to be paid to EDU which owns the trace entry of interest.

Trace: Print Call Stack

```
pid = 14925 tid = 46912874998080 node = 0

308986 sqleProcessSCoordRequest entry [eduid 37 eduname db2agent]
310069 | sqlpParallelRecovery entry [eduid 37 eduname db2agent]
314023 | | sqlpPRecReadLog data [probe 1250]
314027 | | | sqlprProcDPSrec data [probe 430]
314028 | | | | sqlpRecDbRedo entry [eduid 37 eduname db2agent]
314030 | | | | | sqldmrdo data [probe 0]
314031 | | | | | | sqldomRedo entry [eduid 37 eduname db2agent]
314032 | | | | | | | sqldRedoFastTruncTable entry [eduid 37 eduname db2agent]
```

If you take a trace and only consider the initial entry for each routine, you will get a "call stack" – the perfectly ordered sequence of internal Db2 calls (try `db2trc print -stack <traceid> <flwfile>`).

Performance Trace

```
$ db2trc on -perfcount -t

<…run your scenario…>

$ db2trc dmp trace.dmp
$ db2trc off
$ db2trc perffmt trace.dmp trace.perfmt
$ sort -k2nr trace.perfmt > trace.perfmt.sorted
$ vi trace.perfmt.sorted
         1         15.725198000 sqlrr_execimmd
         1         15.725046000 sqlrr_execute_immed
         1         15.702721000 sqlriSectInvoke
    524288         11.086911000 sqlrinsr
    524288         10.367470000 sqldRowInsert
    262145          8.946307000 sqlriisr
```

A great way to “profile” what is happening in Db2. The first column denotes the number of executions, the second column is the total time (in seconds) spent in the routine.

Trace: General Techniques

- Search for **error return codes.** For example, if we are hitting SQL1032, try searching for -1032.
- Search for **pdLog** and or **sqlt_logerr_zrc**, as this points to where information was written to the db2diag.log file and looking above this point might help find the error.
- Try searching bottom up. The trace captures information in the order it occurs, and if the error happens after some execution time, that means the error will be at the bottom (or at least close to).
- Look for trace points where the trace shows hung processes or threads. These can be identified by no return codes or obvious function exits.

Trace Empty?

- When collecting trace to investigate a hang on UNIX/Linux, it is possible that a process may not be logging information into the trace even though trace is on!
- This usually occurs because the hanging EDU has never reached a point where Db2 Trace gets initialized (usually at the entry/exit point of most Db2 routines). In other words, the EDU is stuck inside one routine.
- To force the process to go through the trace initialization, you can send a signal to the process while the trace is turned on:

AIX: SIGGRANT(60) **Linux: SIGPROF(27)** **HP: SIGPROF(21)** **Solaris: SIGPROF(29)**	Initialize flags for the trace facility. The instance stays up.

- The signal handler will force the process to go through the trace init routines and now the process will be tracing.
- Another trick for hang problems and `db2trc` is that if new commands are hanging, try `db2trc` with the `-i` option, in which case the trace buffer will not wrap.

108

•If the process is currently in a codepath (such as a loop) that does not trip over the trace initialization function then the process will not be logging trace entries.

•Sometimes for hang cases, other database activity ends up filling up the trace buffer and wrapping past the “good stuff” that you want to see. By using –i option on the trace, it will trace all the activity until the trace buffer is full and then it will NOT wrap. This is okay for hang cases because you are only interested in the initial codepath that leads into a hang.

Catch-All: `db2fodc`

- All of the aforementioned techniques can be summarized by one word: `db2fodc`
- `db2fodc` is an executable shipped with Db2 which will perform the required data collection automatically
- It is advisable to use `db2fodc` whenever possible:
  - IBM personnel is familiar with the output data format produced by this tool
  - Another reason is that it is possible to automate parsing/analysis of the output

Note: `db2fodc` cannot be used if an non-Db2 problem is suspected

FODC: First Occurrence Data Collection

- **Automatic FODC** – FODC performed by Db2 automatically when an outage or error condition is detected.
- **Manual FODC** – First Occurrence Data Collection invoked manually by the end user due to a particular symptom.
- **FODC package** – a set of diagnostic information collected during the manual or automatic FODC invocation.

110

When an outage occurs and automatic first occurrence data capture (FODC) is enabled, data is collected based on symptoms. The data collected is specific to what is needed to diagnose the outage.

For cases that cannot be determined automatically, e.g. hang or performance issue, data can be collected manually by the user while the outage is happening.

FODC: Components

Item	File Name	Release	Auto/Manual
db2fodc executable	db2fodc	sqllib/bin	M
hang script	db2cos_hang	sqllib/bin	M
performance script	db2cos_perf	sqllib/bin	M
index error script	db2cos_indexerror_short db2cos_indexerror_long	sqllib/bin	both
bad page script	db2cos_datacorruption	sqllib/bin	A
trap script	db2cos_trap	sqllib/bin	A
threshold script	db2cos_threshold	sqllib/bin	M

FODC: Customization

- The behaviour of the data collection is controlled via arguments passed to both `db2fodc` and the data collection scripts, and can be customized. The callout scripts can be modified, too.
- On UNIX, `sqllib/bin/db2cos_hang` to `sqllib/adm/db2cos_hang`, then modify the copy. On Windows, modify the default script in `sqllib/bin/db2cos_hang.bat`.
- In other words, on UNIX, `db2fodc` first tries to execute `sqllib/adm/db2cos_hang`; if not found, `sqllib/bin/db2cos_hang` is used. On Windows, `sqllib/bin/db2cos_hang.bat` is always launched.
- A useful trick is to modify `db2cos_hang` by enabling `no_wait="ON"`. This will remove sleeps between iterations, making the script finish faster. Great mostly for hang situations, not so much for performance scenarios.

Manual FODC: Syntax

-hang

Collects FODC data related to a potential hang situation or a performance issue.

-perf

Collects data related to a performance issue.

db2fodc –hang is currently the preferred method for collecting information for both performance and hang issues, although your mileage may vary. The **–perf** option has less impact on a running workload. The disadvantage is that **–perf** collects less information than **–hang**.

Manual FODC: Basic vs. Full

- Each of the **-hang**/**-perf** options accepts another parameter specifying how much data we want to collect:

basic

The basic collection mode will be run, without user interaction.

full

The full collection mode will be executed, without user interaction.

- If neither **basic** nor **full** is specified, the tool will be interactive. This will be a problem in partitioned environments! Always use **basic** or **full** when executing against a remote partition.

Manual FODC: Partitioned Environment

-dbpartitionnum (or **–dbp**) ***dbpartition_number***

- Collects FODC data related to all the specified database partition numbers. Only a local partition can be specified, i.e. does not work for partitions located on remote physical machines. By default, only information from the current partition number is collected.

-alldbpartitionnum (or -**alldbp**)

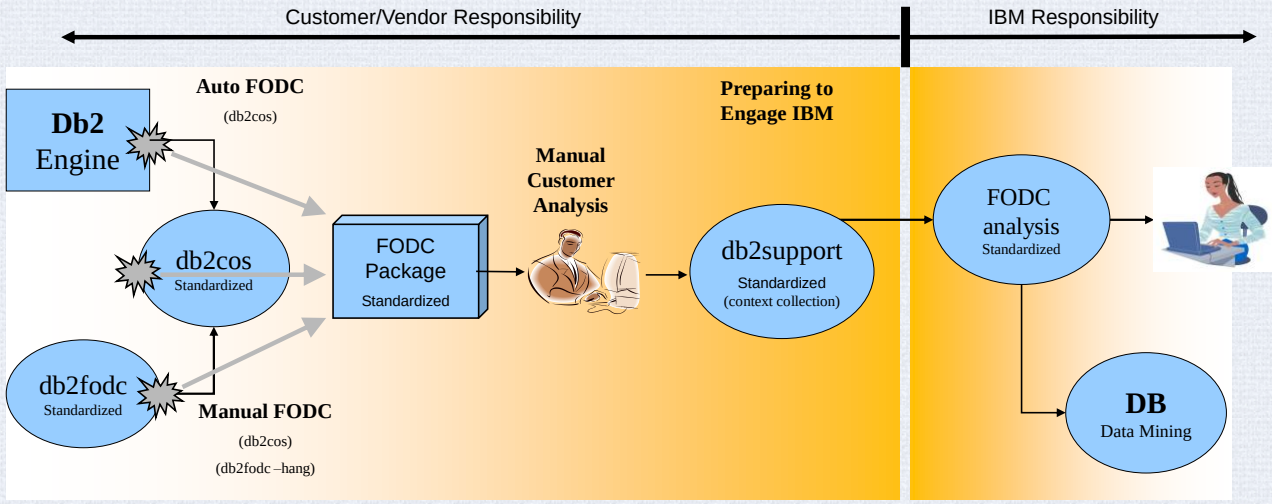
- Specifies that this command is to run on all active database partition servers in the instance. `db2fodc` will report information from database partition servers on the same physical machine that `db2fodc` is being run on.
- In a multi-partitioned environment with multiple physical nodes invoke `db2fodc` on all the nodes in a single invocation in the following way:

```
rah "; db2fodc -hang <options> -alldbp"
```

FODC: Output

- The output data is located in an **FODC_<symptom>_<timestamp>** subdirectory in the default diagnostic path, where **symptom** is the outage symptom (hang, etc..), and **timestamp** is the start time of the invocation. A db2diag.log diagnostic message is logged to inform the user about the directory name used.
- `db2fodc` uses a log file, **db2fodc_*.log**, placed inside the FODC directory. Inside this file `db2fodc` will also store status information and meta-data describing the FODC package. This file will contain information on the type of FODC, the start/stop timestamp of data collection, and other information useful for the analysis of the FODC package.
- `db2support` can be used to collect any FODC packages found under DIAGPATH

Standardized Investigation for Outages



FODC: Most Common Packages

NAME	AUTO/MANUAL	DESCRIPTION
FODC_AppErr	A	Unexpected application termination (e.g. SQL0901N)
FODC_BadPage	A	Bad page (e.g. page checksum problem)
FODC_DBMarkedBad	A	Database marked bad (e.g. transaction logging issue)
FODC_Hang	M	Hang-related information
FODC_IndexError	Both	Index problems (e.g. index corruption)
FODC_Panic	A	Self-inflicted Db2 death, panic (e.g. severe problem)
FODC_Perf	M	Performance-related data
FODC_Trap	A	Trap (e.g. programming error, memory corruption)
FODC_Member	A	pureScale member issue

118

FODC_Member

* New FODC collection for Db2 pureScale feature.

* Supported only on Linux and AIX in Db2 V9.8 and V10

* Collection is done by db2roam process for a member when:

  - Member fails to respond to a whitelisting event from CF. For example, the member is not responding to a reconstruct notification from CF which will delay all in-flight transactions to continue running.

  - Member initiate a suicide, i.e. member has lost all contact with the CFs and decides that the quickest way out of it is to commit suicide and be restarted.

* FODC calls are placed within db2rocm process so that diagnostic data are collected before kill signal is sent to a member.

* FODC call invokes PD callout scripts db2cos_member in <DB2DIR>/bin directory

* All diagnostic information are direct to a new FODC directory FODC_Member_<timestamp> created in DIAGPATH.

* Timeout for the PD data collection is 20 seconds

FODC: DB2FODC Variable

SUBOPTION	DESCRIPTION	DEFAULT
DUMPCORE: ON | OFF	Core file generation	ON
DUMPDIR: path to directory	Specifies absolute pathname of the directory where core file or shared memory dump will be created.	Default diagnostic directory
CORELIMIT:size	The maximum size of core files created.	Not applicable
DUMPSHMEM: ON | OFF	Shared memory dump during outage	OFF
MEMSCAN: ON | OFF	Memory scan on outage	OFF

To change the FODC parameters:

```
db2pdcfg -fodc DUMPCORE=ON CORELIMIT=2000
```

To make these changes permanent:

```
db2set DB2FODC="DUMPCORE=ON CORELIMIT=2000"
```

Essential Tools: `db2pd/db2pdcfg`

- "Monitor and Troubleshoot Db2" command
  - Retrieves various statistics, internal meta-data, and snapshot information from a running Db2 instance
  - Similar to the "`onstat`" utility for Informix
  - Run "`db2pd -help`" for options
- Completely non-intrusive, doesn't acquire latches
  - Very fast retrieval
  - Doesn't impact the engine in any way (can be run even if system is hung)
  - Data may not always be completely accurate
- Read-only operations are possible in `db2pd`. Operations modifying Db2's behaviour have been moved to `db2pdcfg`.

Essential Tools: `dsmtop`

- What is `dsmtop`?
  - Monitoring tool providing a dynamic real-time view of a running Db2 system
  - Can be used to monitor Db2 10.1 and above
  - As of Db2 11.1, shipped with the product
  - Replacement of the now-deprecated `db2top`
- What can `dsmtop` do for you?
  - It calls Db2 monitor APIs repeatedly in background, and displays the result in a “semi-graphic” console interface
  - Help to calculate several common matrices, such as bufferpool hit ratio
  - Provide some basic performance analysis

Essential Tools: dsmtop Example

```
[/]18:25:12 Delta sp17.8 secs, Next r-1.608 secs(3.052) DB Overv Linux,member[1db2inst2sample
File Options View Display Help

                                    ActSessions 100% ==================
                                    LogUsed       0%

        Start       Start Database       System  Instance Mem Database Mem  BufferPool  Shar
         Date        Time   Status Physical Mem      Committed     Committed Memory Used  Memo
   2018-05-22    18:24:45   ACTIVE         3832          58432        153216       15360

                    Select      UID   Activities    Activities   Activities    Pct Time    Th
 Transactions/s     Stmts/s  Stmts/s  Completed/s    Aborted/s    Queued/s      Queued Viola
          0.05        0.05     0.00         0.05         0.00         0.00        0.00

                    Active    Locks         Lock          Lock                      Log
    Connections Connections     Held        Waits Escalations/m  Deadlocks/m    Reads/s     W
              9           9        2            0         0.00         0.00        0.00

       Logical    Physical      Hit        Async                      Async      Avg P
      Reads/s     Reads/s    Ratio      Reads/s      Writes/s     Writes/s  Read Time    Re
        49.62       13.71   70.62%         0.00         0.00         0.00        0.04

                      Sort    Hash    Hash Join         Hash   Hash Grpby      Avg P
       Sorts/m Overflows/m  Joins/m  Overflows/m    Grpbys/m  Overflows/m Write Time   Wri
          0.00        0.00     0.00         0.00         0.00         0.00        0.00
                                localhost                                Regex:N Export:N Trace:N
```

Bring System Back Online

- If we are completely hung, all commands are stuck, and the usual ways to stop the instance (`db2stop force`) are hanging too, we will have to kill the instance. This will forcefully bring down the instance in an unfriendly way:

`db2_kill`	(UNIX/Linux)
`db2nkill.exe`	(Windows)

- Once the instance has been shut down, some time should be spent to ensure there are no rogue/stray Db2 processes or resources:

`ps -elf/ipcs -a`	(UNIX/Linux)
`Task Manager`	(Windows)

Cannot Kill?

- If a process cannot be killed even using the most privileged OS commands such as SIGKILL (`kill -9`) on UNIX/Linux or `taskkill` on Windows, then the process is likely stuck in the OS kernel.
- This is an operating system problem and likely the hang itself is also an operating system problem. Need to contact OS support and/or collect OS dumps.
- Typically, the only way to get out of this state is to reboot the machine once the necessary OS data has been collected.

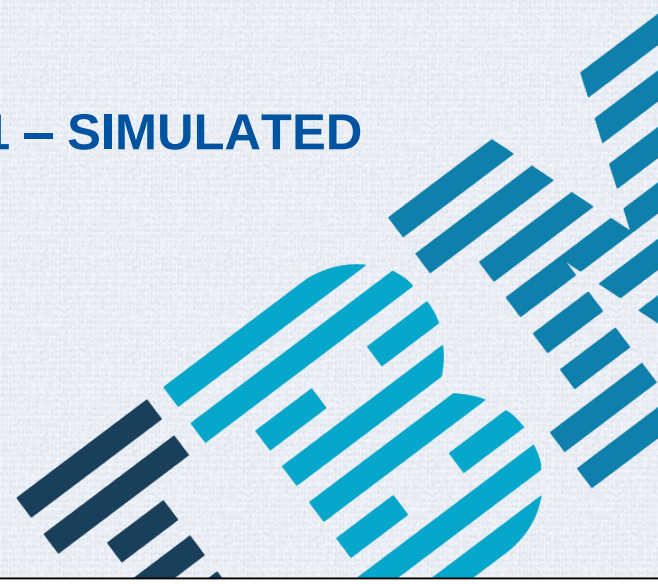
124

•It is not programmatically possible to avoid a SIGKILL (9). If a process is not responding to SIGKILL, it means it's stuck in OS kernel code.

•It's likely that crash recovery will be required after a kill. If the first connect gives an error or does not complete, then there may be a crash recovery problem.

•Unresponsive crash recovery is not *usually* a real hang. Use **`db2pd -db <dbname> -recovery`** to monitor crash recovery progress.

HANG EXERCISE 1 – SIMULATED



Exercise 1

```
SESSION 1:
$ db2 "connect to sample"
<...hanging...>

SESSION 2:
$ db2 "connect to sample"
<...hanging...>

SESSION 3:
$ db2 "list active databases"
<...hanging...>
```

Two ways to collect data:

1. Collect everything – recommended, `db2fodc -hang` can be helpful
2. Iterative – iteratively get to the root cause; for intermediate/advanced users

Collect Everything: `db2fodc -hang`

```
$ db2fodc -hang basic
"db2fodc": List of active databases: "SAMPLE"

**********************WARNING**********************
*     This tool should be run with caution.        *
*     It can cause significant performance         *
*  degradation, especially on busy systems with a  *
*        high number of active connections         *
*                                                  *
*   All times specified below are estimates based  *
*    on the test runs and your individual times    *
*       may vary depending on hardware and         *
*     OS configurations and current workload       *
***************************************************

You have 10 seconds to cancel this script with Ctrl-C
You may interrupt execution at any time by issuing Ctrl-C
The script will then dump any active db2 trace, and inform where to find
the output files

<...several minutes later...>

Output directory is /home/db2inst1/sqllib/db2dump/FODC_Hang_2018-04-17-14.35.53.447540
Open db2fodc_hang.log in that directory for details of collected data
```

127

Note that since we are unable to connect to the database, we are using the **–basic** option of **db2fodc** which does not need a database connect. The **–full** option collects more data, e.g. DB snapshots, but unfortunately this is a no-go in this case.

Diagnostics: Call Stacks

```
$ ls -l FODC_Hang_2018-04-17-14.35.53.447540
-rw-r--r-- 1 db2inst1 db2iadm1 55718 Apr 17 14:40 6258.1.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 55004 Apr 17 14:40 6258.11.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 54878 Apr 17 14:40 6258.12.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 27747 Apr 17 14:38 6258.13.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 55334 Apr 17 14:40 6258.14.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 54688 Apr 17 14:40 6258.15.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 55664 Apr 17 14:40 6258.16.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 59742 Apr 17 14:40 6258.17.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 54862 Apr 17 14:40 6258.18.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 54864 Apr 17 14:40 6258.19.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 54864 Apr 17 14:40 6258.20.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 54860 Apr 17 14:40 6258.21.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 55698 Apr 17 14:40 6258.22.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 55834 Apr 17 14:40 6258.23.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 55836 Apr 17 14:40 6258.24.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 55836 Apr 17 14:40 6258.25.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 59440 Apr 17 14:40 6258.26.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 59354 Apr 17 14:40 6258.27.000.stack.txt
-rw-r--r-- 1 db2inst1 db2iadm1 59354 Apr 17 14:40 6258.28.000.stack.txt
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 17 14:42 DB2CONFIG
-rw-r--r-- 1 db2inst1 db2iadm1  3978 Apr 17 14:42 db2fodc_hang.log
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 17 14:42 DB2PD
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 17 14:36 DB2SNAPS
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 17 14:42 DB2TRACE
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 17 14:36 OSCONFIG
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 17 14:37 OSSNAPS
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 17 14:37 OSTRACE
```

128

The stacks are the most critical piece of information for hangs.

Waiters: Method 1

```
$ vi StackAnalysis.out

 00007FC71126F86E sqloSpinLockConflict + 0x0240
  00007FC710E895C7 _ZN13sqeDBIterator13lockDbMgrArgsEPKci + 0x00b7
  00007FC710FDD4C7 _Z17sqm_get_next_dbcbbPP13dbcb_use_listPP16sqeLocalDatabaseb + 0x00a5
  00007FC710FA8656 _Z12sqlmonssagntj13sqm_entity_idP6sqlmaijPvP14sqlm_collectedttP5sqlca + 0x02e6
  00007FC710FD9188 _Z15sqlmonssbackendP12SQLE_DB2RA_T + 0x0500

Found in:
        ./6258.27.000.stack.txt -- db2agent(instance) -- 2018-04-17-14.38.04.895773(Signal #10)
        ./6258.27.000.stack.txt -- db2agent(instance) -- 2018-04-17-14.40.05.637437(Signal #10)
        ./6258.28.000.stack.txt -- db2agent(instance) -- 2018-04-17-14.38.04.894481(Signal #10)
        ./6258.28.000.stack.txt -- db2agent(instance) -- 2018-04-17-14.40.05.637776(Signal #10)

 00007FC71126F86E sqloSpinLockConflict + 0x0240
  00007FC710EC1FFC _ZN8sqeDBMgr23StartUsingLocalDatabaseEP8SQLE_BWAP8sqeAgentRccP8sqlo_gmt + 0x1074
  00007FC710EABAE1 _ZN14sqeApplication13AppStartUsingEP8SQLE_BWAP8sqeAgentccP5sqlcaPc + 0x0209
  00007FC710EA6D1E _ZN14sqeApplication13AppLocalStartEP14db2UCinterface + 0x01b8

 Found in:
        ./6258.26.000.stack.txt -- db2agent(SAMPLE) -- 2018-04-17-14.38.04.898404(Signal #10)
        ./6258.26.000.stack.txt -- db2agent(SAMPLE) -- 2018-04-17-14.40.05.637623(Signal #10)
```

Three unique Db2 EDUs *(note the different timestamps)* are stuck in **lockDbMgr**() -> **sqloSpinLockConflict**(). The presence of **sqloSpinLockConflict**() indicates that these agents are waiting for a latch.

129

The “analyzestack” tool was used (see earlier slides) to generate StackAnalysis.out.

Concluding that **sqloSpinLockConflict**() means that the EDU is waiting for a latch takes some experience. However, the backup slides will try to list the most frequent call stack routines to make this decision easier.

Holders: Method 1

```
$ vi StackAnalysis.out

 00007FC70FC58765 ossSleep + 0x003b
  00007FC711B6A5DB _ZN16sqeLocalDatabase12FirstConnectEP8SQLE_BWARcP8sqeAgentP8sqlo_gmtii + 0x0307
  00007FC710EC1BE8 _ZN8sqeDBMgr23StartUsingLocalDatabaseEP8SQLE_BWAP8sqeAgentRccP8sqlo_gmt + 0x0c60
  00007FC710EABAE1 _ZN14sqeApplication13AppStartUsingEP8SQLE_BWAP8sqeAgentccP5sqlcaPc + 0x0209
  00007FC710EA6D1E _ZN14sqeApplication13AppLocalStartEP14db2UCinterface + 0x01b8

 Found in:
        ./6258.17.000.stack.txt -- db2agent(SAMPLE) -- 2018-04-17-14.38.04.897566(Signal #10)
        ./6258.17.000.stack.txt -- db2agent(SAMPLE) -- 2018-04-17-14.40.05.636082(Signal #10)
```

- Unlike in the case of Waiters, this EDU is not immediately visible to an inexperienced eye. However, a closer examination reveals that all the waiters are waiting for a local database start.
- Having reviewed "**StackAnalysis.out**", EDU 17 seems to be the only logical candidate here. This Db2 agent is the only thread other than the Waiters that has anything to do with a database start.
- At this point it would be nice to have access to Db2 source code *(check with your local Torrent supplier ☺)*.

Waiters: Method 2

Remember that a call stack contains latch information (if the latch is trackable).

```
$ grep  -i "latch type" *.txt | sort

6258.17.000.stack.txt:Holding Latch type: (SQLO_LT_sqeDBMgr__dbMgrLatch) - Address: (0x2007301e8), Line: 4731, File:
    sqle_database.C HoldCount: 1
6258.17.000.stack.txt:Holding Latch type: (SQLO_LT_sqeDBMgr__dbMgrLatch) - Address: (0x2007301e8), Line: 4731, File:
    sqle_database.C HoldCount: 1
6258.26.000.stack.txt:Waiting on latch type: (SQLO_LT_sqeDBMgr__dbMgrLatch) - Address: (0x2007301e8), Line: 681, File:
    sqle_database_services.C
6258.26.000.stack.txt:Waiting on latch type: (SQLO_LT_sqeDBMgr__dbMgrLatch) - Address: (0x2007301e8), Line: 681, File:
    sqle_database_services.C
6258.27.000.stack.txt:Waiting on latch type: (SQLO_LT_sqeDBMgr__dbMgrLatch) - Address: (0x2007301e8), Line: 280, File:
    sqlmutil.C
6258.27.000.stack.txt:Waiting on latch type: (SQLO_LT_sqeDBMgr__dbMgrLatch) - Address: (0x2007301e8), Line: 280, File:
    sqlmutil.C
6258.28.000.stack.txt:Waiting on latch type: (SQLO_LT_sqeDBMgr__dbMgrLatch) - Address: (0x2007301e8), Line: 280, File:
    sqlmutil.C
6258.28.000.stack.txt:Waiting on latch type: (SQLO_LT_sqeDBMgr__dbMgrLatch) - Address: (0x2007301e8), Line: 280, File:
    sqlmutil.C
```

Pay close attention to the address! The address is always unique for a latch. Two different latches residing at two different addresses can share the same symbolic latch type (name)!

131

Some latches in Db2 are not trackable. Typically, those are “hot” latches, i.e. latches that are very frequently accessed. Tracking those would imply a significant performance degradation. For this reason it is good to learn the previous method as well.

Unlike in the previous method, this picture gives an obvious explanation on the deadlatch.

Conclusion

- **EDU 27 and 28**
  - `waiting for SQLO_LT_sqeDBMgr__dbMgrLatch at 0x2007301e8`
  - `stuck in` **`lockDbMgrArgs`**`()`
- **EDU 26**
  - `waiting for SQLO_LT_sqeDBMgr__dbMgrLatch at 0x2007301e8`
  - `stuck in` **`StartUsingLocalDatabase`**`()`
- **EDU 17**
  - `holding SQLO_LT_sqeDBMgr__dbMgrLatch at 0x2007301e8`
  - `stuck in` **`FirstConnect`**`() -> ` **`ossSleep`**`() - doing nothing`

- The problem is with EDU 17. Using application snapshots or `db2pd`, we could map this EDU to a running application. However, do not be searching for known defects. Remember, a custom made library was used, and a sleep command was injected into FirstConnect() ☺

- Admittedly, this has been somewhat too textbook-like example. Now that we have a general idea, the next example will be harder. But first, let's see how this approach compares with investigating performance problems!


PERFORMANCE EXERCISE – SIMULATED

Exercise

1. Baseline test

```
Starting a performance test...
$ time db2 "select * from staff" > /dev/null

real    0m5.90s
user    0m0.08s
sys     0m2.45s
```

2. After some secret modifications ☺

```
Starting a performance test...
$ time db2 "select * from staff" > /dev/null

real    13m16.26s
user    0m0.18s
sys     0m3.95s
```

134

The first test reflects the normal processing times. The second test indicates a problem. We are assuming that we do not know what has changed, and there is no way to revert any potential changes until we know what may have triggered the failing behaviour.

Collect Everything I: db2fodc –hang full

```
Starting a performance test...
$ time db2 "select * from staff" > /dev/null

<...open another session or send the session to the background...>
<...issue the following command while the problem is present...>

$ db2fodc -hang full
...
Collecting more DB2 CONFIG info (started at 04:28:11 PM)
Estimated time to completion is 5 minutes (Ctrl-C to interrupt)
Note: next steps may hang on busy or non-responsive systems
      since they require connection to database.
......Finished at 04:28:14 PM

.Collecting DB2 monitoring info (started at 04:28:15 PM)
Estimated time to completion is 10 minutes (Ctrl-C to interrupt)
..........Waiting 1 minutes and 0 secs before starting the next iteration...
..........Finished at 04:29:34 PM

.
Output directory is /home/db2inst1/sqllib/db2dump/FODC_Hang_2018-04-24-16.16.35.819757
Open db2fodc_hang.log in that directory for details of collected data
```

135

•In this example, we are collecting all the data at once. Experienced users may try to collect data interactively on the fly.

•Similar to investigating hangs, diagnostic data for performance problems must be collected while the performance problem is present. Usually, post-mortem is not possible – unless the problem has some easily visible symptoms in db2diag.log or other static diagnostic logs.

•We can run the **"full"** option of **"db2fodc –hang"** since we are able to connect to the database just fine when the problem is present, i.e. this is not a blocking issue.

•Note that there is also the **"db2fodc –perf"** option. There are advantages and disadvantages to using the **"-perf"** switch. The advantage is that the output is arguably easier to look at as less "types" of data is being gathered. The disadvantage is that this information alone is often insufficient to determine the root cause; **"-hang"** is better in this sense.

Operating System: Disk/CPU/Memory

OSSNAPS/vmstat.1/2:

```
$ cat FODC_Hang*/OSSNAPS/vmstat.1
 16:17pm  up   2:27,  1 user,  load average: 1.05, 0.41, 0.71
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  2  14304  66568   5848 458032    0    5 23605   130  302  804  3  7 63 27  0
 1  2  14304  66308   5856 458016    0    0 58880   236  999 1521  4 14  0 82  0
 1  2  14304  78476   5664 446148    0    0 50304     0  878 1352 12 24  0 64  0
 0  2  14304  78612   5664 446148    0    0 55680     0  927 1435  1 15  0 84  0
 2  2  14304  78620   5664 446140    0    0 59776     0  979 1500  1 12  0 87  0
 0  2  14304  78620   5664 446140    0    0 58240   168  999 1494  1 12  0 87  0
 0  2  14304  78620   5664 446116    0    0 54656     0  905 1392  1 14  0 85  0
 0  2  14304  78744   5664 446116    0    0 41344     0  716 1096  2 13  0 85  0
 0  2  14304  78760   5664 446108    0    0     0     0   49  121  1  0  0 99  0
 1  2  14304  78760   5664 446108    0    0 41472     0  696 1093  0 11  0 89  0
```

- Blocked queue (“b”) constantly above zero => processes doing I/O
- I/O wait time (“wa”) very high => waiting for I/O operations to complete

Conclusion: The system is I/O bound

Applications

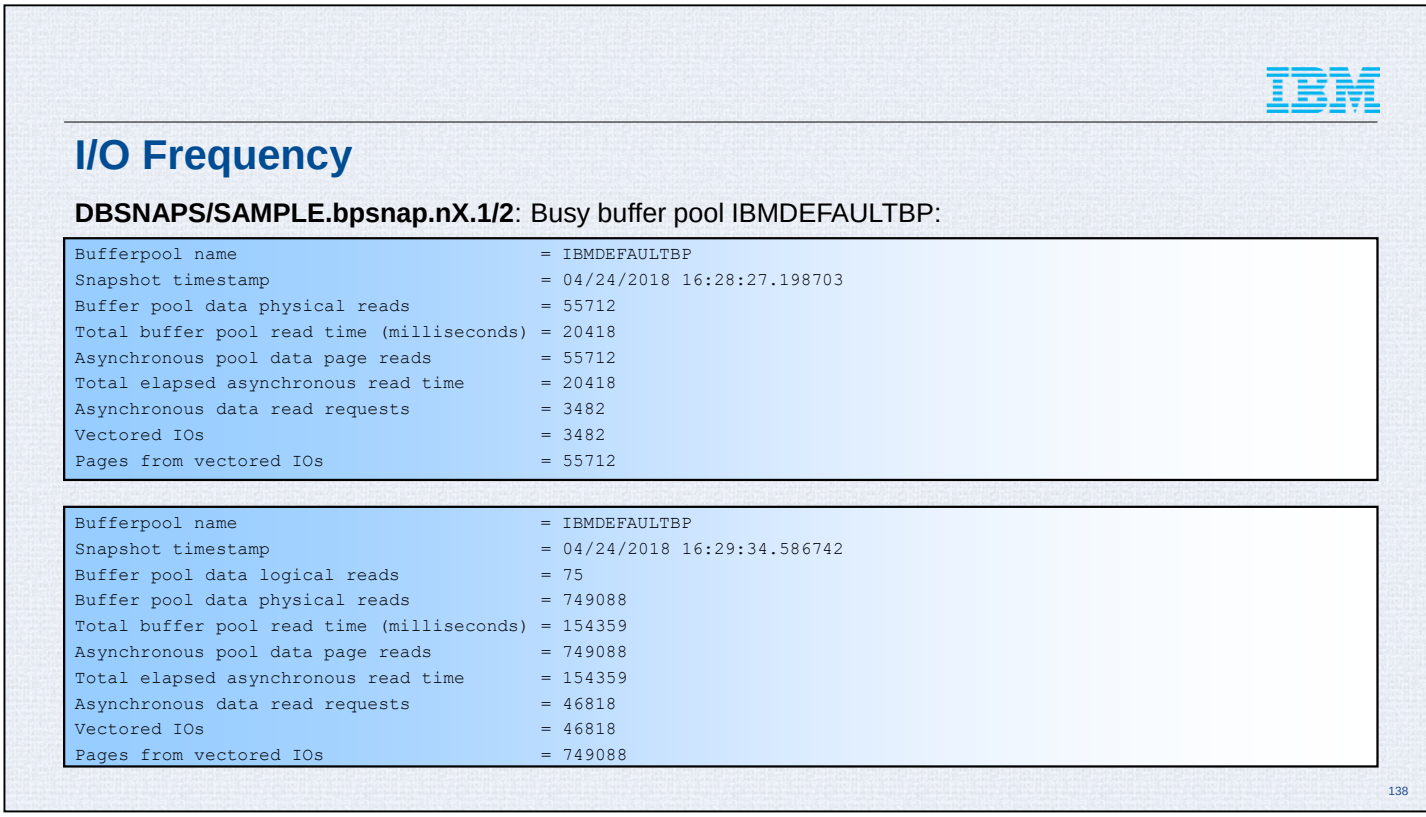
DBSNAPS/SAMPLE.appsnap.nX.1/2: The only active application is 72:

```
Application handle                         = 72
Application status                         = UOW Executing
Application name                           = db2bp
Snapshot timestamp                         = 04/24/2018 16:28:26.734603
Time waited for prefetch (ms)              = 0
Rows selected                              = 117286
Rows read                                  = 117467
Rows fetched                               = 0
Buffer pool data logical reads             = 0
```

```
Application handle                         = 72
Application status                         = UOW Executing
Application name                           = db2bp
Snapshot timestamp                         = 04/24/2018 16:29:34.182203
Time waited for prefetch (ms)              = 27228
Rows selected                              = 128731
Rows read                                  = 128912
Rows fetched                               = 10877
Buffer pool data logical reads             = 68
```

137

Compare the following snapshots taken one minute apart. There is a slight increase of every counter. We appear to be reading data from the disk.



Compare the following snapshots taken one minute apart. There is a significant increase of every counter. The symptoms are indicative of an increased prefetching activity.

Db2 EDUs (Engine Dispatchable Units)

DB2PD/db2pd.edus.txt: Busy prefetchers, EDUs 53 and 54

```
EDU ID  EDU Name                              USR (s)       SYS (s)
============================================================
59      db2agntdp (SAMPLE  ) 0                0.010000      0.000000
58      db2agntdp (SAMPLE  ) 0                0.040000      0.010000
57      db2agntdp (SAMPLE  ) 0                0.170000      0.090000
56      db2evmgi (DB2DETAILDEADLOCK) 0        0.200000      0.030000
55      db2pfchr (SAMPLE) 0                   0.010000      0.000000
54      db2pfchr (SAMPLE) 0                   0.200000     14.520000
53      db2pfchr (SAMPLE) 0                   0.190000     21.980000
52      db2pclnr (SAMPLE) 0                   0.000000      0.000000
51      db2dlock (SAMPLE) 0                   0.000000      0.030000
50      db2lfr (SAMPLE) 0                     0.000000      0.000000
49      db2loggw (SAMPLE) 0                   0.030000      0.020000
48      db2loggr (SAMPLE) 0                   0.200000      0.200000
47      db2wlmd (SAMPLE) 0                    0.120000      0.100000
46      db2taskd (SAMPLE) 0                   0.160000      0.070000
45      db2fw0 (SAMPLE) 0                     0.180000      0.080000
44      db2agent (SAMPLE) 0                   0.350000      0.420000
31      db2stmm (SAMPLE) 0                    0.400000      0.390000
16      db2resync 0                           0.010000      0.000000
15      db2tcpcm 0                            0.000000      0.000000
14      db2ipccm 0                            0.060000      0.040000
12      db2thcln 0                            0.000000      0.010000
11      db2alarm 0                            0.440000      0.270000
1       db2sysc 0                             0.970000      1.200000
```

Conclusion: This matches our previous suspicion – prefetching gone wrong?

Call Stacks

```
$ vi StackAnalysis.out
 Stack:
 ======================
 00007F10565C5FB0 readv + 0x00e0
 00007F10587FFFCC sqloReadV + 0x0236
 00007F1058E5225B sqlbReadAndReleaseBuffers + 0x04e7
 00007F1058E5080F sqlbProcessRange + 0x0649
 00007F1058E5010E sqlbServiceRangeRequest + 0x00c8
 00007F1058E4D9DD sqlbPFPrefetcherEntryPoint + 0x0977
 00007F1058E4D030 sqbPrefetcherEdu6RunEDUEv + 0x003a

 Summary:
 Found in 6 stacks of a total of 68 stacks in 24 files
 Found in:
 ./10587.53.000.stack.txt -- db2pfchr(SAMPLE) -- 2018-04-24-16.18.58.091218(Signal #10)
 ./10587.53.000.stack.txt -- db2pfchr(SAMPLE) -- 2018-04-24-16.21.03.416486(Signal #10)
 ./10587.53.000.stack.txt -- db2pfchr(SAMPLE) -- 2018-04-24-16.23.10.484986(Signal #12)
 ./10587.54.000.stack.txt -- db2pfchr(SAMPLE) -- 2018-04-24-16.18.58.096031(Signal #10)
 ./10587.54.000.stack.txt -- db2pfchr(SAMPLE) -- 2018-04-24-16.21.03.416319(Signal #10)
 ./10587.54.000.stack.txt -- db2pfchr(SAMPLE) -- 2018-04-24-16.23.10.477532(Signal #12)
```

Conclusion: Indeed, prefetchers 53 and 54 very busy reading data from the disk!

140

We already know how to use the "`analyzestack`" `tool to create StackAnalysis.out`

Collect Everything II: `db2fodc -perf full`:

- Arguably the most useful data from `db2fodc -perf full` is the performance traces and call stacks ***(StackTrace.*)***.
- The stack traces are really a subset of those gathered by `db2fodc -hang`, and we have already reviewed those. Only the "hottest" stacks have been gathered.
- However, the performance traces are something new:

```
$ ls -l FODC_Perf*
-rw-r--r-- 1 db2inst1 db2iadm1  2245 Apr 24 16:50 db2fodc.log
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 24 16:37 db2perfcount.0305
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 24 16:46 db2perfcount.0837
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 24 16:45 db2trc.0761
-rw-r--r-- 1 db2inst1 db2iadm1     0 Apr 24 16:32 iostat.out
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 24 16:49 snapshots
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 24 16:33 StackTrace.0077
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 24 16:36 StackTrace.0229
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 24 16:40 StackTrace.0457
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 24 16:42 StackTrace.0609
drwxr-xr-x 2 db2inst1 db2iadm1  4096 Apr 24 16:47 StackTrace.0913
-rw-r--r-- 1 db2inst1 db2iadm1 17862 Apr 24 16:49 vmstat.out
```

141

Let us interrupt the investigation for a bit. The user may also decide to gather **"db2fodc –perf full"** information. The advantage/disadvantages have already been discussed on the previous slides. This page shows how this data may be useful in our investigation.

Formatting Performance Traces

```
$ db2trc perffmt perfcount_dmp perfcount_dmp.fmt

$ sort -k 2.2b,3rn -k 4,5rn perfcount_dmp.fmt > perfcount_dmp.fmt.sorted

$ head perfcount_dmp.fmt.sorted
318     (53 sec, 491624000 nanosec)      sqloWaitEDUWaitPost
7607    (19 sec, 807799000 nanosec)      sqloReadV
7600    (19 sec, 767058000 nanosec)      sqloReadVLow
3       (5 sec, 602000 nanosec)          sqlorque2
3       (5 sec, 404000 nanosec)          sqlorqueInternal
1       (0 sec, 13718000 nanosec)        sqlfReadDb2nodes
1       (0 sec, 13660000 nanosec)        sqloReadDb2nodes
14      (0 sec, 7237000 nanosec)         sqloGetEnvUnCached
42      (0 sec, 6992000 nanosec)         EnvPrfOpen
2       (0 sec, 3913000 nanosec)         sqlfcsys
```

- A performance trace shows the most frequently executed routines
- 1st column: <number of executions>, 2nd column: <total time>, 3rd: <routine>

Conclusion: A lot of time spent executing **sqloReadV()** – disk readv() routine

142

The conclusion matches our previous slides. There is a process/processes that is/are performing a lot of disk reads. From the call stacks (see previous slides) we know that it is the prefetchers.

Trace Facility

- **DB2TRACE/db2trace1/2.flw** too short (no data for the prefetchers 53 and 54)
- If the issue is reproducible, we can take custom traces:

```
$ db2trc on -t -p 10587,13970798804352024592 -f trace.dmp
Trace is turned on

$ db2trc flw -t trace.dmp trace.flw

$ vi trace.flw
pid = 10587 tid = 139707992237824 node = 0

188            0.002303000   sqloReadV entry [eduid 80 eduname db2pfchr]
189            0.002305000   | sqloReadVLow entry [eduid 80 eduname db2pfchr]
192            0.004709000   | sqloReadVLow exit
193            0.004722000   sqloReadV exit
194            0.004723000   sqloReadV entry [eduid 80 eduname db2pfchr]
195            0.004724000   | sqloReadVLow entry [eduid 80 eduname db2pfchr]
200            0.007196000   | sqloReadVLow exit
201            0.007213000   sqloReadV exit
202            0.007214000   sqloReadV entry [eduid 80 eduname db2pfchr]
203            0.007215000   | sqloReadVLow entry [eduid 80 eduname db2pfchr]
209            0.009680000   | sqloReadVLow exit
210            0.009692000   sqloReadV exit
```

Conclusion: Could the prefetcher be looping?

143

[-t]

Include timestamps.

[-p <pid>[.<tid>][,<pid>[.<tid>]]]

Trace or format only these process/thread combinations.

[-l [<bufferSize>] | -i [<bufferSize>] | -f <filename>]

Trace into shared memory (-l, -i) or to a file (-f).

Conclusion

1. System I/O bound
2. Application running a SELECT issuing a lot of prefetching requests
3. Prefetchers spending a lot of time reading data from disk
4. Prefetchers possibly looping

- This alone is usually sufficient to focus on the offending application (e.g. gather explains), or search for existing APARs

- <u>The real root cause:</u>
  - A modified Db2 library was used
  - The library had new injection points in the prefetching code path
  - Instead of doing one prefetch, the prefetcher was issuing 10,000 identical read requests in a loop ☺
  - This had resulted in an enormous I/O strain


HANG EXERCISE 2 – REAL WORLD

Exercise 2: Workshop Environment

The exercise assumes the existence a single partitioned Db2 instance running Db2 9.7 GA. This is a real life scenario reported from the field. The GA level was specifically chosen to allow us to reproduce the known problem.

```
$ db2level
DB21085I  Instance "db2inst1" uses "64" bits and DB2 code release "SQL09070"
with level identifier "08010107".
Informational tokens are "DB2 v9.7.0.0", "s090521", "LINUXAMD6497", and Fix
Pack "0".
Product is installed at "/opt/ibm/db2/V9.7".

$ uname -a
Linux demobox 3.0.13-0.27-default #1 SMP Wed Feb 15 13:33:49 UTC 2018 (d73692b) x86_64 x86_64 x86_64
GNU/Linux
```

Exercise 2

```
$ db2 "connect to sample"

   Database Connection Information

 Database server        = DB2/LINUXX8664 9.7.0
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE

$ db2 "create table t1 (i1 int, c2 char(250) generated always as (i1))"
DB20000I  The SQL command completed successfully.

$ db2 "import from 'dataFile01.del' of del messages /dev/null insert into t1 (i1)"
Number of rows read         = 50
Number of rows skipped      = 0
Number of rows inserted     = 50
Number of rows updated      = 0
Number of rows rejected     = 0
Number of rows committed    = 50

$ db2 "create index i1 on t1 (i1,c2) pctfree 0"
DB20000I  The SQL command completed successfully.

$ db2 "create table t2 (i1 int, c2 char(250) generated always as (i1), c3 char(250) generated always as (i1), c4
    char(250) generated always as (i1), c5 char(250) generated always as (i1))"
DB20000I  The SQL command completed successfully.
```

Exercise 2 (cont'd)

```
$ db2 "create table t3 (i1 int, c2 char(1) generated always as (i1))"
DB20000I  The SQL command completed successfully.

$ db2 "connect reset"
DB20000I  The SQL command completed successfully.

$ db2 "update db cfg for sample using LOGBUFSZ 4 LOGFILSIZ 4 LOGPRIMARY 2 LOGSECOND 0 SOFTMAX 200"
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.

$ db2stop; db2start; db2 "connect to sample"
<...skipping...>

$ db2 "insert into t2 (i1) values 1"
DB20000I  The SQL command completed successfully.
<...repeat 16 times...>

$ db2 "insert into t3 (i1) values 1"
DB20000I  The SQL command completed successfully.
<...repeat 8 times...>

$ db2 "insert into t1 (i1) values 1,3,5,7,9"
<...HANGS...>
```

Putting It All Together

```
$ db2pd -stack all
Attempting to produce all stack traces for database paritition.
See current DIAGPATH for trapfiles.

$ db2pd -latches
Database Partition 0 -- Active -- Up 0 days 01:06:43

Latches:
Address            Holder     Waiter     Filename             LOC
No latch holders.
No latch waiters.
```

NO LATCHES? HUH? This is weird! Time to take a look at the stacks:

```
$ ~/sqllib/pd/analyzestack -i .
<...skipping...>
Analysis Complete ...
```

Stack Analysis

```
$ vi StackAnalysis.out

  00007F78BD8D1576 sqloltch_notrack + 0x0072
  00007F78BDFE66A8 sqlilidx + 0x01dc
  00007F78BE01BD0F sqliundo + 0x1007
  ...
  00007F78BD9689EB sqldmund + 0x0193
  00007F78BDDE947C sqlptudo + 0x0214
  00007F78BDDE8C6F sqlptud1 + 0x023f
  00007F78BF21689E sqlpSpRb + 0x02c2
  00007F78BD95F1C4 sqldRowInsert + 0x0b38

Found in:
        ./16524.17.000.stack.txt -- db2agent(SAMPLE)
```

1. We seem to be dealing with an agent that is trying to insert a row.
2. The agent is in the middle of an UNDO (rollback) operation.
3. We are trying to acquire a latch (again, source code would be useful ☺), but there is no visible holder for this latch.

ANYONE DARE GUESS? (there is a hint close to the top of this page!)

What Happened: Step 1

1. The presence of `sqloltch_notrack()` tells us that we are trying to acquire a latch that is not tracked (“hot”). This is why the latch holder is not visible.

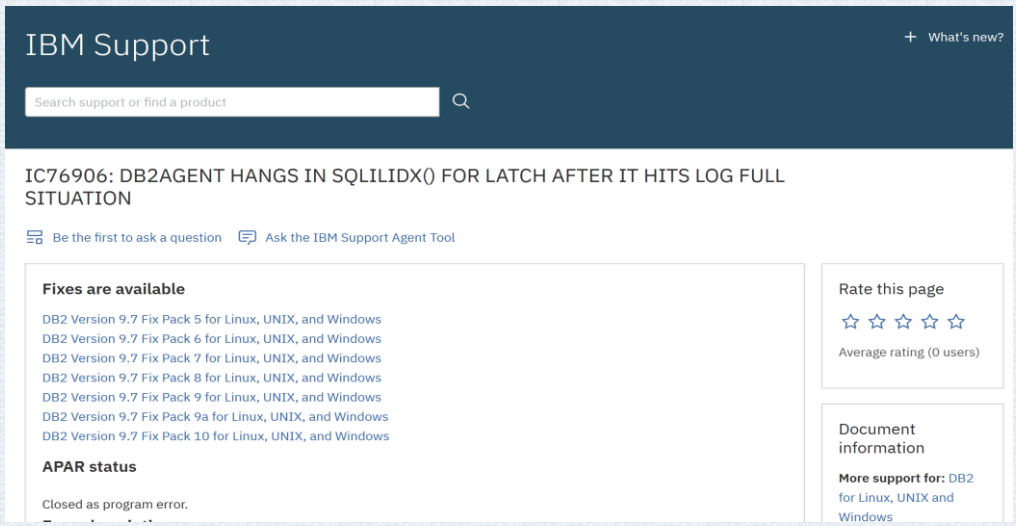
2. The fact that “**StackAnalysis.out**” contains no other candidates and we have definitely gathered the call stacks for all running EDUs means that the owner of the latch must be the same EDU!

3. In other words, EDU 17 is self-deadlatching itself. The agent is trying to acquire a latch that the agent already owns.

PRETTY GOOD DESCRIPTION => search for existing APARs!

What Happened: Step 2

Search for Db2 APARs at IBM Support Portal



152

Keyword used: “sqlilidx hang”

IC76906: DB2AGENT HANGS IN SQLILIDX() FOR LATCH AFTER IT HITS LOG FULL SITUATION

Error description

After Db2 hits a log full situation, in rare case some db2agent EDUs performing insert/update/delete may hang, and there is at least one hung EDU with these functions on the stack: sqlilidx sqliundo ixmUndo waiting for latch.

Local fix

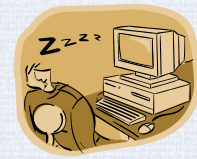
Avoid log full situation. Adjusting max_log and num_log_span can help to reduce risk of hitting log full situation

Hangs: Important Routines

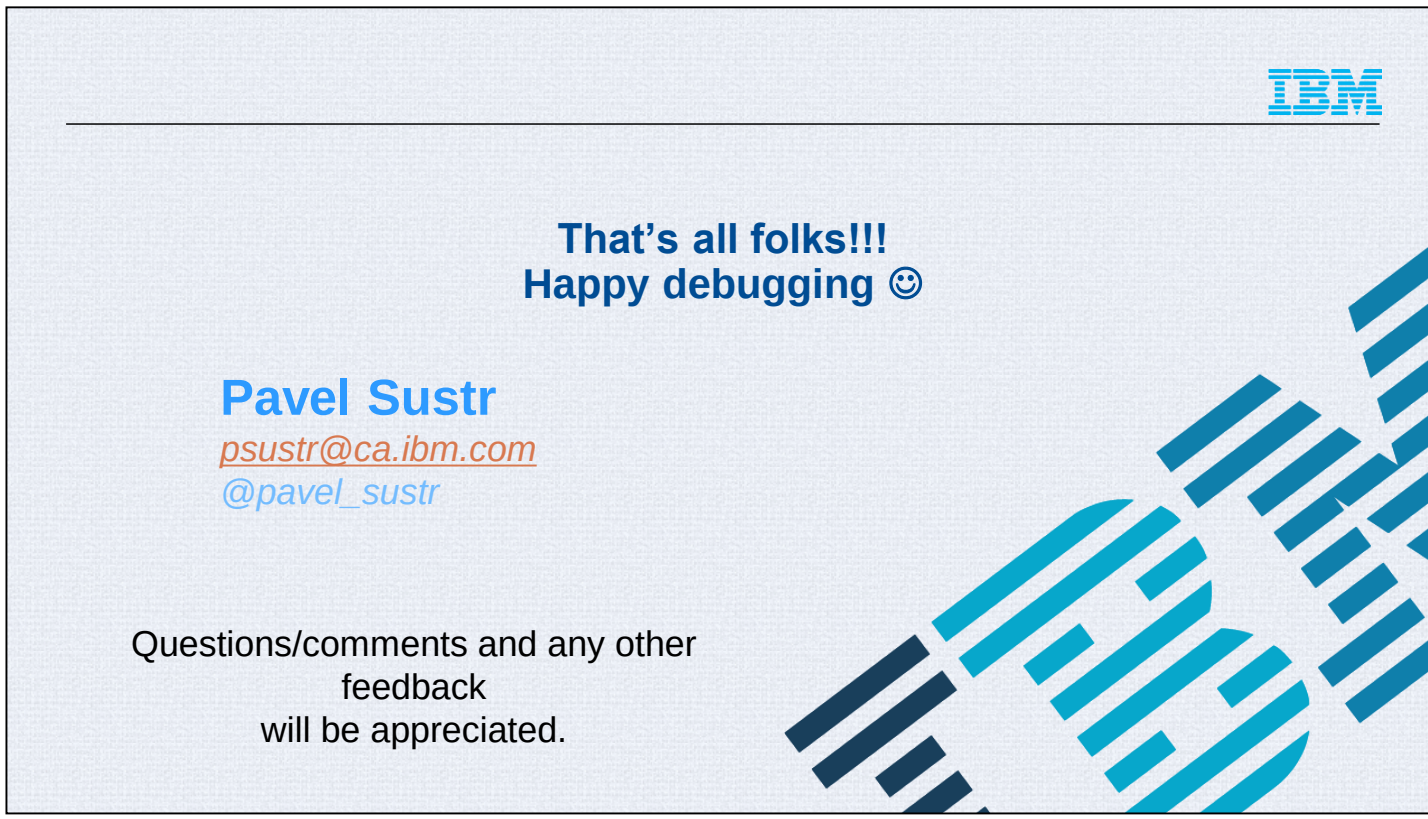
- The following routines serve as the first eyecatcher. An EDU executing these routines is always waiting for a latch, and this EDU should be closely examined:
  - `getConflictComplex`
  - `sqloltch`
  - `sqloltch_notrack`
  - `sqloSpinLockConflict`

Hangs: Less Important Routines

- The presence of the following routines usually (but not always ☺) indicates that the owning EDU is legitimately idle (e.g. sleeping, waiting for work), and the problem is elsewhere:
  - `msgrcv`
  - `ossSleep`
  - `semtimedop`
  - `sqleIntrptWait`
  - `sqloCSemP`
  - `sqloWaitEDUWaitPost`
  - `sqlorest`
  - `sqlorqueInternal`



- Also, if an application state is “UOW Waiting”, this application is NOT executing inside the Db2 kernel. Instead, the application is waiting for a remote request (usually outside of Db2) => not a Db2 issue.



Pavel's Bio

Senior Manager and Senior Software Engineer with IBM Db2 LUW development, responsible for multiple core Db2 kernel components. Always thrilled to work on hard-to-crack puzzles. Expertise in Db2 LUW kernel architecture, configuration and administration, advanced problem determination, memory architecture, memory leak troubleshooting, and assembly language. Hands-on development experience with buffer pool management, storage, prefetching, page cleaning, transaction logging, recovery, monitoring, and problem determination. As a member of the Db2 team, Pavel spent years in Db2 L2/L3 advanced support (over 1,500 resolved cases), then transitioned to Db2 LUW kernel development. In his past life Pavel was an application developer mostly using C++, SQL, .NET, Oracle, MS-SQL, and Informix on Windows, Linux, Solaris, and HP-UX.