

# Automatically Analyze HANA Issues with SAP HANA dump analyzer

Jan 2019

# Example: HANA Runtime Dump

A **SAP HANA runtime dump** is a **text file** that provides various information about the current state of the SAP HANA database. Runtime dumps are essential for SAP support to understand the SAP HANA behavior in a problem situation.

A runtime dump can be created manually or automatically and per default it is created in the SAP HANA trace directory /usr/sap/<sid>/HDB<inst>/<host>/trace.

**Standard runtime dump** that can be created in different situations:

- Manually via "runtimedump dump" in hdbccons (SAP Note [2222218](#))
- Manually via SQL (MANAGEMENT\_CONSOLE\_PROC)
- Manually via SAP HANA Studio -> "Administration" -> "Diagnosis Files" -> "Diagnosis Information" -> "Collect"
- Implicitly as part of a full system info dump (SAP Note [1732157](#))
- Automatically via SAP HANASitter (SAP Note [2399979](#))

The runtime dumps of following **name convention** will be created

- <service>\_<host>.<port>.rtdump.<timestamp>.trc
- <service>\_<host>\_<port>\_runtimedump\_<timestamp>.trc

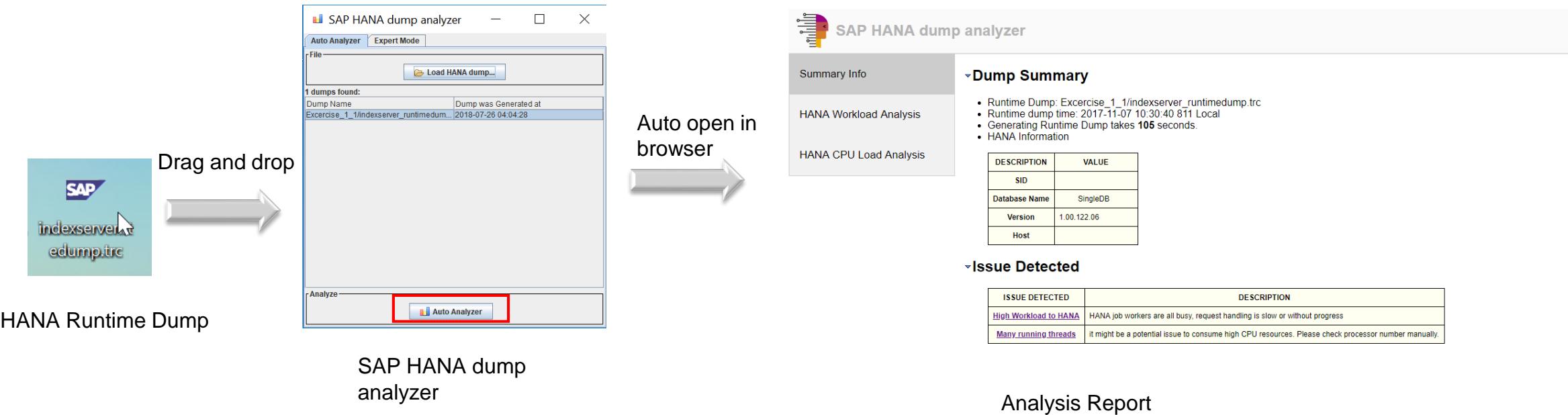
There are also Memory/Savepoint/Page corruption runtime dumps, more details can be found in SAP Note [2400007](#) - FAQ: SAP HANA Runtime Dumps

# Example: HANA Runtime Dump

## One runtime dump looks like this...

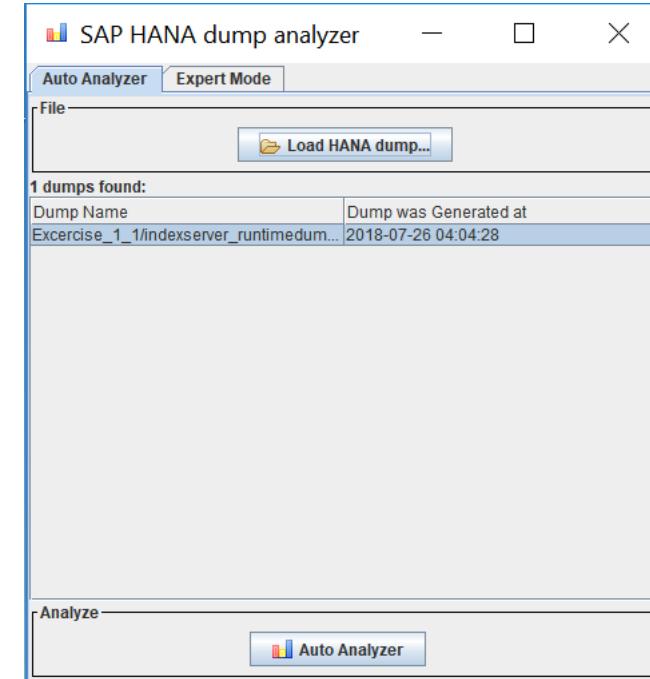
# Use SAP HANA dump analyzer to Analyze a HANA issue

- SAP HANA dump analyzer has a easy to use interface: just drag and drop your runtime dump and click Auto Analyzer Button!



# Environment Setup

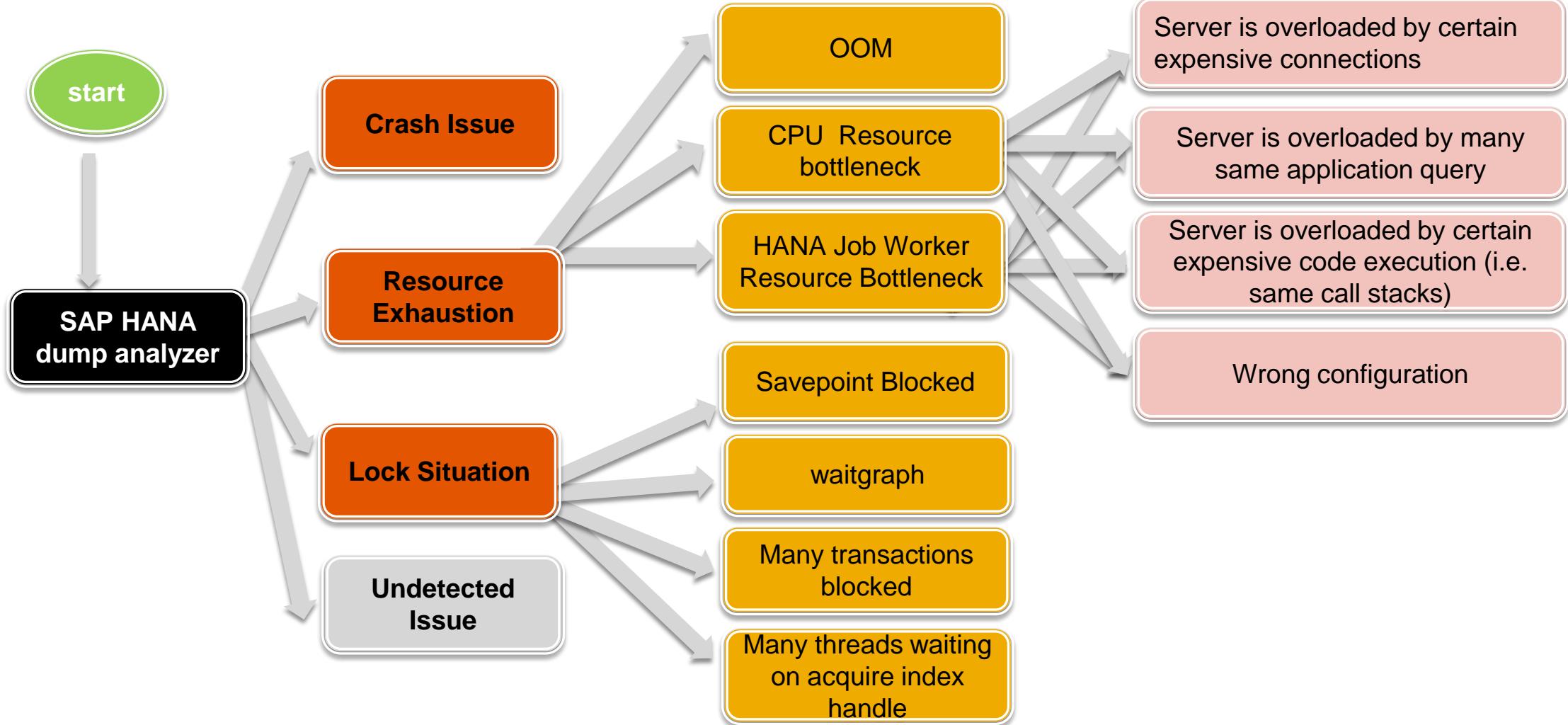
- JDK 8.0 is required
- Double click on HANADumpAnalyzer.jar
- Super Easy Interface: Drag drop and click!
- Supported platform: Windows, Linux and MacOS.



## Further Info

- The tool is introduced via SAP **Knowledge Based Article 2498739** (How-To: Analyzing Runtime Dumps with SAP HANA dump analyzer ).

# A Glance at HANA Auto Analyzer Decision Tree



# SAP HANA dump analyzer – OOM Analyzer

Global Allocation Limit & Inter process analysis including:

- IPMM information, e.g. HANA configured global memory allocation limit, memory consumption by different processes, memory consumption situation

The screenshot shows the SAP HANA dump analyzer interface. On the left, there's a sidebar with a logo and the text "SAP HANA dump analyzer". Below it, a button labeled "OOM Analysis" is highlighted with a red box. The main content area has a title "Dump Summary" with a red border. Inside, a list of runtime dump details is shown:

- Runtime Dump: indexserver.redump.oom.trc
- Runtime dump time: 2017-06-06 06:38:55 346 Local
- Generating Runtime Dump takes 23 seconds.
- Rundirectory: /usr/sap/HPO/HDB00/hostname
- Version : 1.00.102.05.1455873859 (fa/newdb100\_maint\_rel)

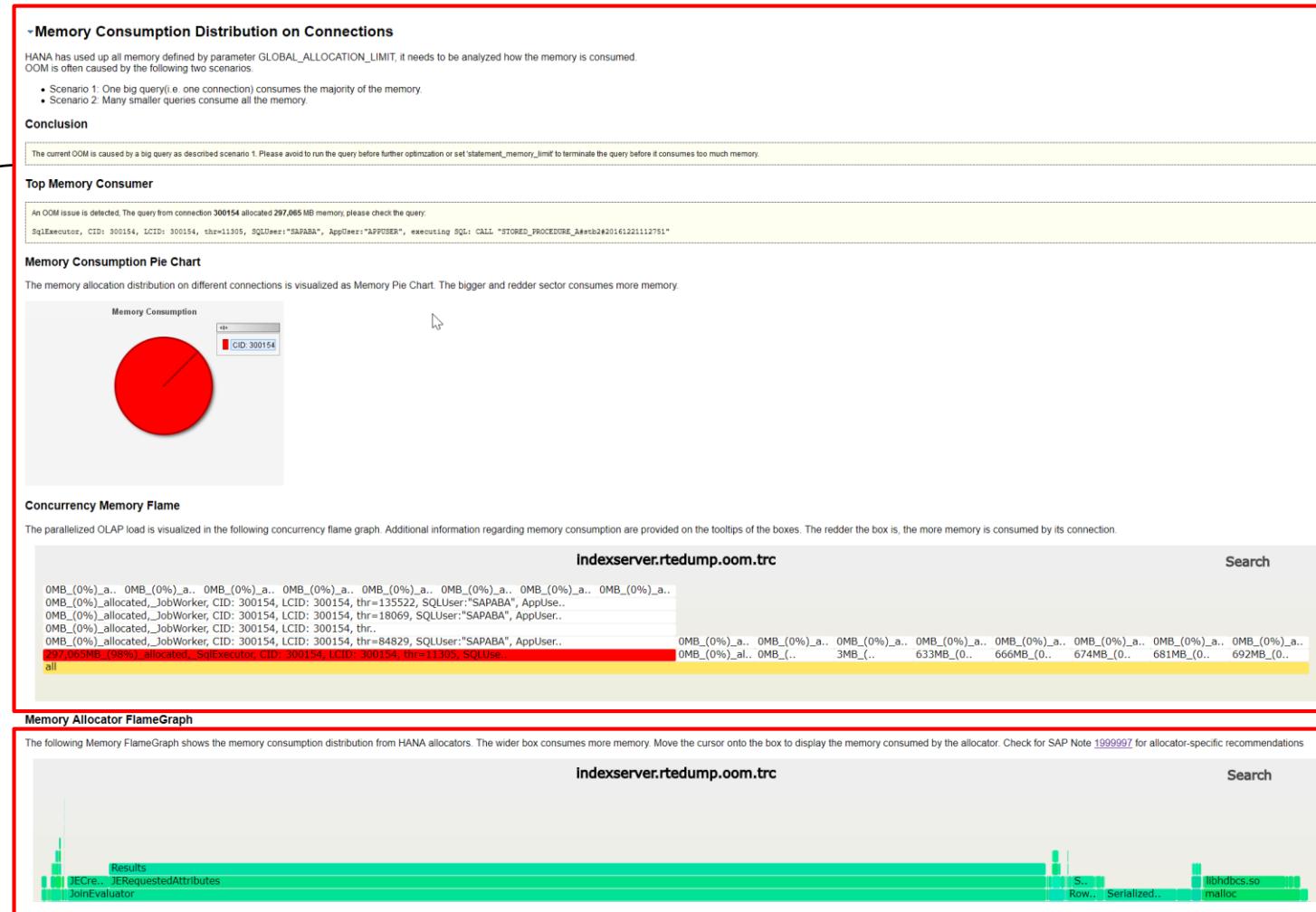
Below this is another section titled "Global Allocation Limit & Inter Process Memory Management (i.e. IPMM)" with a red border. It contains a heading "IPMM Short Info" and a large block of text detailing memory usage statistics for various SAP components like hbnameserver, hdbcomplexserver, hdbpreprocessor, hdtxsengine, and hdbindexserver. The text includes memory allocation (PAL), shared memory (SHARED\_MEMORY), code size (CODE\_SIZE), and other parameters for each component.

Runtime dump summary including:

- Dump file name and exception time
- HANA DB information

# SAP HANA dump analyzer – OOM Analyzer

- Memory consumption analysis including:
- Point out if issue is cause by single big query
  - Memory consumption distribution on connections visualized with pie chart
  - Thread concurrency flame graph with memory consumption visualization and information



- Memory allocator flamegraph to show the memory distribution via different HANA allocator

# SAP HANA dump analyzer – Crash Analyzer

The screenshot shows the SAP HANA dump analyzer interface. At the top, there's a navigation bar with icons for Home, Help, and Logout. Below it, a sidebar on the left has 'Crash Analysis' selected. The main area is titled 'Dump Summary' and contains a bulleted list of runtime dump details. A red box highlights the 'Dump Summary' section. Another red box highlights the 'Crash Analysis' section, which contains a detailed stack trace of a crash. A large red arrow points from the 'Crash Analysis' section towards the right side of the slide, where a summary of the runtime dump is listed.

**Dump Summary**

- Runtime Dump: Excercise\_6\_2/indexserver.30044.crashdump.20140408-200432.041349.trc
- Exception time: 2014-04-08 20:04:32 000 Local
- Generating Runtime Dump takes 4 seconds
- Rundirectory: /usr/sap/HWD/HDB00/ushdcb569
- Version: 1.00.69.03.388114 (NewDB100\_REL)

**Crash Analysis**

A crash issue is detected, please find the crash info below.

```
[CRASH_SHORTINFO] exception short info: (2014-04-08 20:04:32 739 Local)
SIGNAL 11 (SIGSEGV) caught, thread: 674850nr=41551 Request addr: 0x0000000000000038 time: 2014-04-08 20:04:32 000 Local
Instance HWD/00, OS Linux ushdcb569 2.6.32-24.0-2-default #1 SMP 2010-10-29 16:39:49 +0200 x86_64

[STACK_SHORT] Stacktrace of crash:
0: ptme::SparsePagePool::Partition::remove(ptime::d_Ref) + 0xad
1: ptme::SparsePagePool::remove(ptime::d_Ref) + 0x6f
2: ptme::Transaction::delete_container(int, bool, bool, bool) + 0x57a
3: ptme::QueryExecutor::post_drop_in_memory_table(ptime::Transactions, ptime::DropTableInfo&, bool) + 0x5c
4: NewDb::DistDDLRequestHandler::executePostDropTable(TrexNet& Requests) + 0x3d5
5: NewDb::DistDDLRequestHandler::handle(TrexNet& Requests) + 0x446
6: TRexAPI::TREXIndexServer::handle(TrexNet& Requests, TrexService::HandlerContexts) + 0x3ae0
7: WorkerThread::run(void*) + 0x98
8: TRexThread::PoolThread::run() + 0xa7
9: TRexThread::PoolThread::run(void*) + 0x18
10: Execution::Thread::staticMain((void**) + 0x5d8
11: Execution::Thread::staticMain(void*) + 0xd3

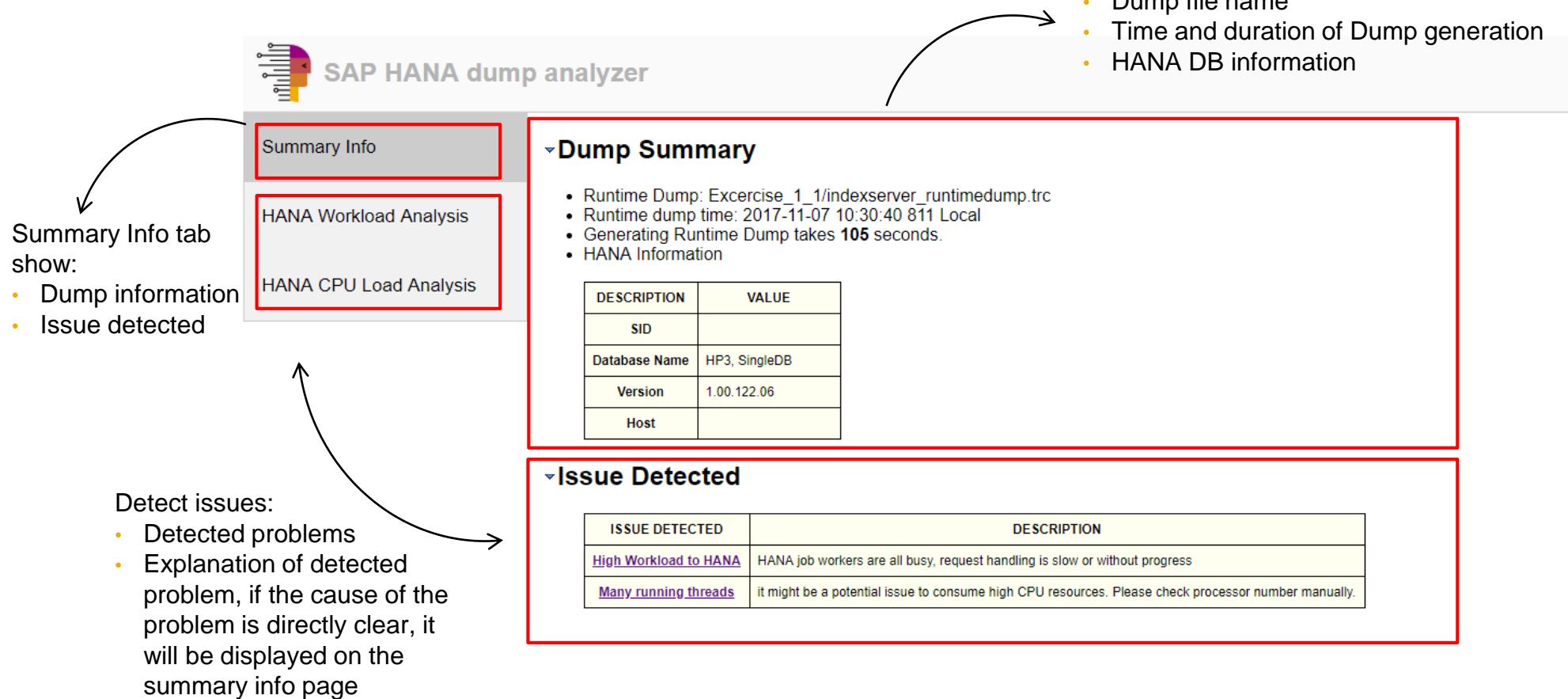
[CRASH_STACK] stacktrace of crash: (2014-04-08 20:04:32 741 Local)
--> Symbolic stack backtrace <--
0: ptme::SparsePagePool::Partition::remove(ptime::d_Ref<ptime::PageHeader>, int, ptme::SparsePagePool::Partition::PAGE_TYPE) + 0xad
Symbol: _ZN5ptme14SparsePagePoolremoveENS_5_d_RefINS_10PageHeaderEEE
SFrame: IP: 0x000074bed5b60d(0x000074bed5b560+0x3d) FP: 0x00007495db8ba80 SP: 0x00007495db8ba20 RP: 0x000074bed5a0cf
Params: rax=0x0, rbx=0x7f4a69f4a100, rcx=0x0, rdx=0x0, rsi=0xffffffff, rdi=0x7f495db8ba30, rbp=0x7f495db8ba70, rsp=0x7f4920000000, r9=0x3f4000, r10=0x4f68d766ff94, r11=0x3b9acacf1a, r12=0x7f4a69f4a0f0, r13=0x7f495db8ba40, r14=0x229002fc
r15=0x7f495db8bc00
Source: mmn_pageheader.h:119
Module: /usr/sap/HWD/HDB00/exe/libhbdrskernel.so

1: ptme::SparsePagePool::remove(ptime::d_Ref<ptime::PageHeader>) + 0x6f
Symbol: _ZN5ptme14SparsePagePoolremoveENS_5_d_RefINS_10PageHeaderEEE
SFrame: IP: 0x000074bed5a0cf(0x000074bed5a000+0x0) FP: 0x00007495db8ba80 SP: 0x00007495db8ba0 RP: 0x000074bedf8683a
Params: ?, ?, 0x23f4000
Regs: rbx=0x7f4a69f4a100, rdx=0x23f4000, rbp=0x7f495db8baa0, rsp=0x7f495db8ba80, r12=0x40a, r13=0x7f495db8ba80, r14=0x229002fc000, r15=0x7f495db8bc0
Source: mmn_sparsespapepool.cc:381
Module: /usr/sap/HWD/HDB00/exe/libhbdrskernel.so

2: ptme::Transaction::delete_container(int, bool, bool, bool) + 0x57a
Symbol: _ZN5ptme11Transaction16delete_containerEbb0
SFrame: IP: 0x000074bed8683a(0x000074bed862c0+0x57a) FP: 0x00007495db8bd00 SP: 0x00007495db8bab0 RP: 0x000074bed1365cc
Params: ?, ?, 0x7f4923f4000
Regs: rbx=0x7f4a69f4a100, rdx=0x7f4923f4000, rbp=0x7f495db8bc0, rsp=0x7f495db8bab0, r12=0x7f495db8bc80, r13=0x7f4923f4000, r14=0x229002fc000, r15=0x7f495db8bc60
Source: transact.cc:1895
Module: /usr/sap/HWD/HDB00/exe/libhbdrskernel.so

3: ptme::QueryExecutor::post_drop_in_memory_table(ptime::Transactions, ptme::DropTableInfo&, bool) + 0x5c
Symbol: _ZN5ptme13QueryExecutor25post_drop_in_memory_tableERNS_11TransactionERNS_13DropTableInfoEb
SFrame: IP: 0x000074bed1365cc(0x000074bed136570+0x5c) FP: 0x00007495db8bf60 SP: 0x00007495db8bd00 RP: 0x000074bed329745
```

# SAP HANA dump analyzer – Summary Info



# SAP HANA dump analyzer – CPU Load Analyzer

SAP HANA dump analyzer

Summary Info

HANA Workload Analysis

**HANA CPU Load Analysis**

IO Analysis

**CPU Load Statistics**

There are many running threads (i.e. threads which are not waiting on e.g. synchronization), this is expected to utilize CPU resource intensively.

PROCESSOR_NUM	288
THREADS_COUNT (threads which have a call stack)	337
RUNNING_THREADS_COUNT (threads are not waiting on e.g. synchronization)	230

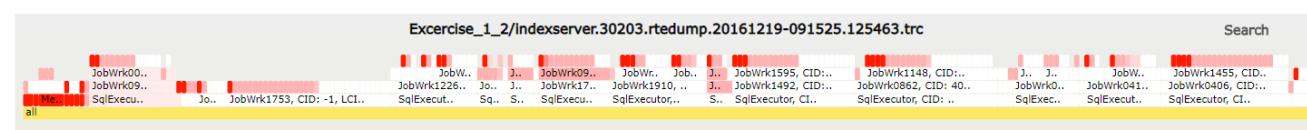
**Detailed Information**

**Concurrency FlameGraph**

As system is under high load, please analyze the parallelized OLAP load from the concurrency flame graph:

- One OLAP request spawns multiple threads for execution via one connection, it's visualized as one column.
- If there are many columns on the bottom level, it means there are many requests executed at the same time.
- If there are large columns (i.e. all boxes from one connection occupies a large area), it means there are queries with huge parallelization. You can get to the bottom of the column (i.e. the sqlexecutor) to get the query information
- The runtime of the thread is visualized in color, the redder connection runs for longer period.

Exercise\_1\_2/indexserver.30203.rtedump.20161219-091525.125463.trc



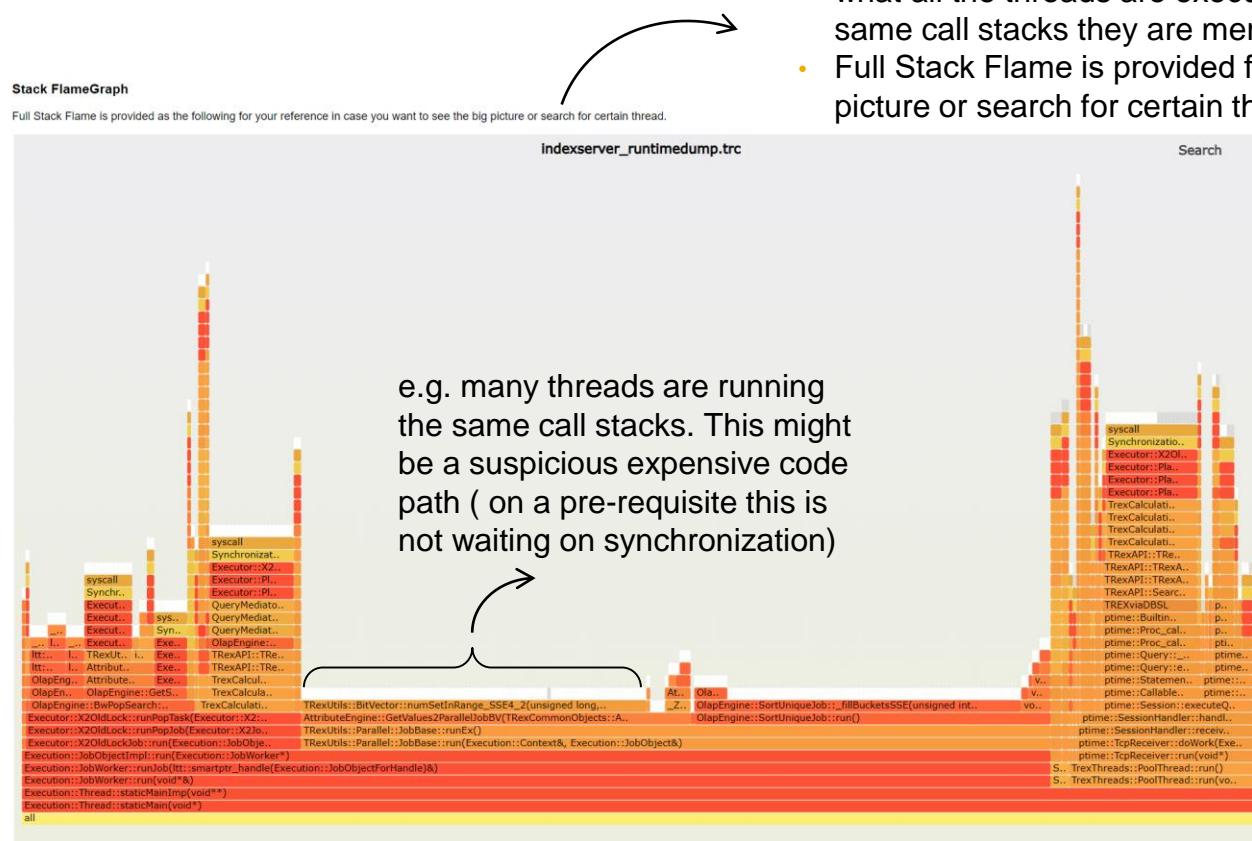
Detailed information including:

- Concurrency flame graph visualize how (mainly) OLAP load are using threads resources

CPU load statistics including:

- CPU logical processor number (as recorded in CPUINFO in runtimedump)
- Thread count for threads which have a call stack executed currently as recorded in STACK\_SHORT section
- Running thread count for threads which are not waiting, e.g. on synchronization

# SAP HANA dump analyzer – CPU Load Analyzer



Detailed information including:

- Running threads stack which may lead to high CPU utilization. This shows what all the threads are executing. In case many threads are executing the same call stacks they are merged to the same big box
- Full Stack Flame is provided for reference in case it's needed see the big picture or search for certain thread

# SAP HANA dump analyzer – HANA Workload Analyzer

Detailed information including:

- OLAP workload concurrency flame graph
- Pie chart visualization of connection number per Application & statement.

SAP HANA dump analyzer

Summary Info

HANA Workload Analysis

HANA CPU Load Analysis

**-JobWorker Statistics**

HANA is busy and request handling is slow or without progress based on system view M\_JOBEXECUTORS\_.

- All job workers are busy, the number of busy job workers (i.e. BUSY\_WORKER\_COUNT) reaches to the limit of MAX\_CONCURRENCY.
- New job workers could not be started any further (i.e. FREE\_WORKER\_COUNT = 0)
- Jobs waiting to be executed are queued up. (i.e. QUEUED\_JOBS number of jobs are queued up)
- No processor number can be retrieved from [CPUINFO] section in runtime dump, please check processor number manually to see whether the configuration makes sense or not.

DESCRIPTION	VALUE
MAX_CONCURRENCY	440
MAX_CONCURRENCY_HINT	16
BUSY_WORKER_COUNT	447
QUEUED_JOBS	159
FREE_WORKER_COUNT	0
TOTAL_WORKER_COUNT	1346
PARKED_WORKER_COUNT	606
SYSBWAIT_WORKER_COUNT	1
JOBWAIT_WORKER_COUNT	291
YIELDWAIT_WORKER_COUNT	0

Jobworker statistics based on system view M\_JOBEXECUTORS\_. In case MAX\_CONCURRENCY is configured too small or too big, it will directly be detected

**Detailed Information**

**OLAP Workload Concurrency FlameGraph**

Job workers are threads, which are responsible to process parallelized OLAP load and internal activities like savepoints or garbage collection. As all job workers are busy, please analyze the parallelized OLAP load from the concurrency flame graph:

- One OLAP request spawns multiple threads for execution via one connection, it's visualized as one column.
- If there are many columns on the bottom level, it means there are many requests executed at the same time.
- If there are large columns (i.e. all boxes from one connection occupies a large area), it means there are big requests.
- The runtime of the thread is visualized in color, the redder connection runs for longer period.

**Indexserver\_runtimedump.trc**

**Application & Statement WorkLoad Analysis**

The running connections are executing different queries and different applications. The workload distribution on different applications and different statements are visualized as the following. More threads are running with the same application/query on the bigger sector. You can move the cursor onto the sector to get more details.

**Conclusion**

System is overloaded by a single query (i.e. over 50% of the threads are running the same query) with SQL: [all Schema].STORED PROCEDURE\_A[7, 7, 7, 7, 7, 7, 7, 7]

**Application WorkLoad**

**Statement WorkLoad**

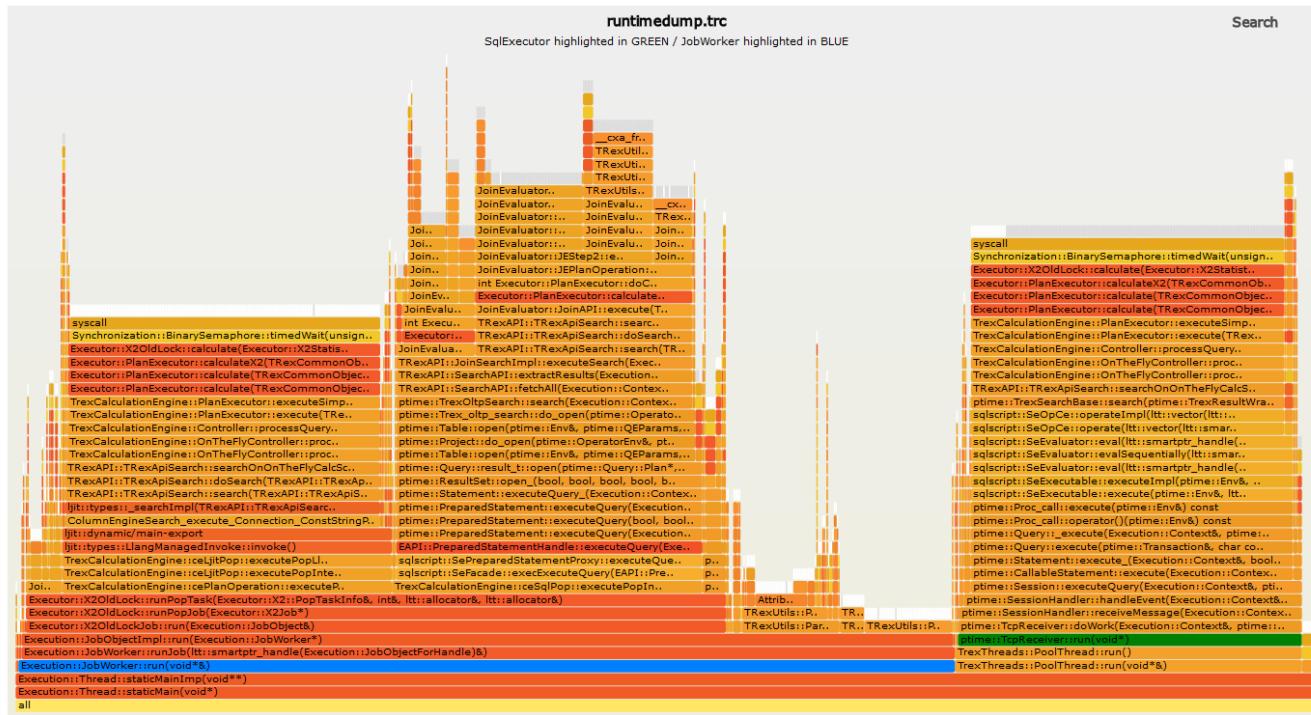
e.g. if many threads are executing the same statement, it will be directly highlighted in the reddest and biggest sector

# SAP HANA dump analyzer – HANA Workload Analyzer

## Threads Stack FlameGraph

As all job workers are busy, please check the big picture what HANA is busy from the threads flame graph.

- SQL executors are threads, which are responsible for normal SQL request processing, highlighted in green.
  - Job workers are threads, which are responsible to process parallelized OLAP load and internal activities like savepoints or garbage collection, highlighted in blue.



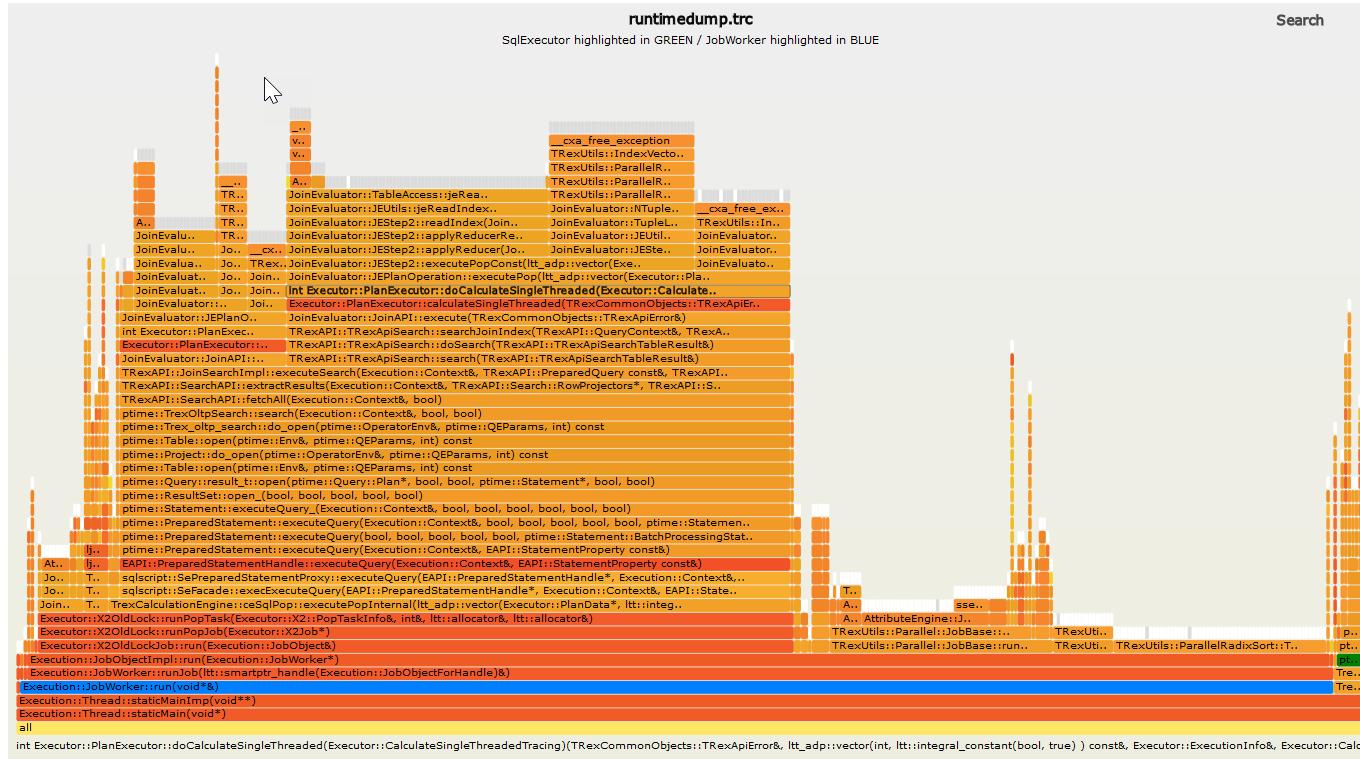
As all job workers are busy, please check the big picture what HANA is busy from the threads flame graph:

- SQL executors are threads, which are responsible for normal SQL request processing, highlighted in green.
  - Job workers are threads, which are responsible to process parallelized OLAP load and internal activities like savepoints or garbage collection, highlighted in blue.

# SAP HANA dump analyzer – HANA Workload Analyzer

## Running Threads Stack FlameGraph

- Usually if all job workers are occupied and busy, either there is a higher load to the system or HANA is not processing the requests fast enough.
- To understand how HANA is processing the request, please check the running threads (i.e. threads that are not waiting on e.g. synchronization) below:



- Usually if all job workers are occupied and busy, either there is a higher load to the system or HANA is not processing the requests fast enough.
- To understand how HANA is processing the request, the call stacks of running threads are visualized in the running thread flame graph

# SAP HANA dump analyzer – Savepoint Analyzer

If the savepoint is blocked over certain defined threshold, the savepoint analyzer will receive the thread blocks savepoint and print in the savepoint blocker section.



**SAP HANA dump analyzer**

Summary Info

Savepoint Blocked Analysis

WaitGraph Analysis

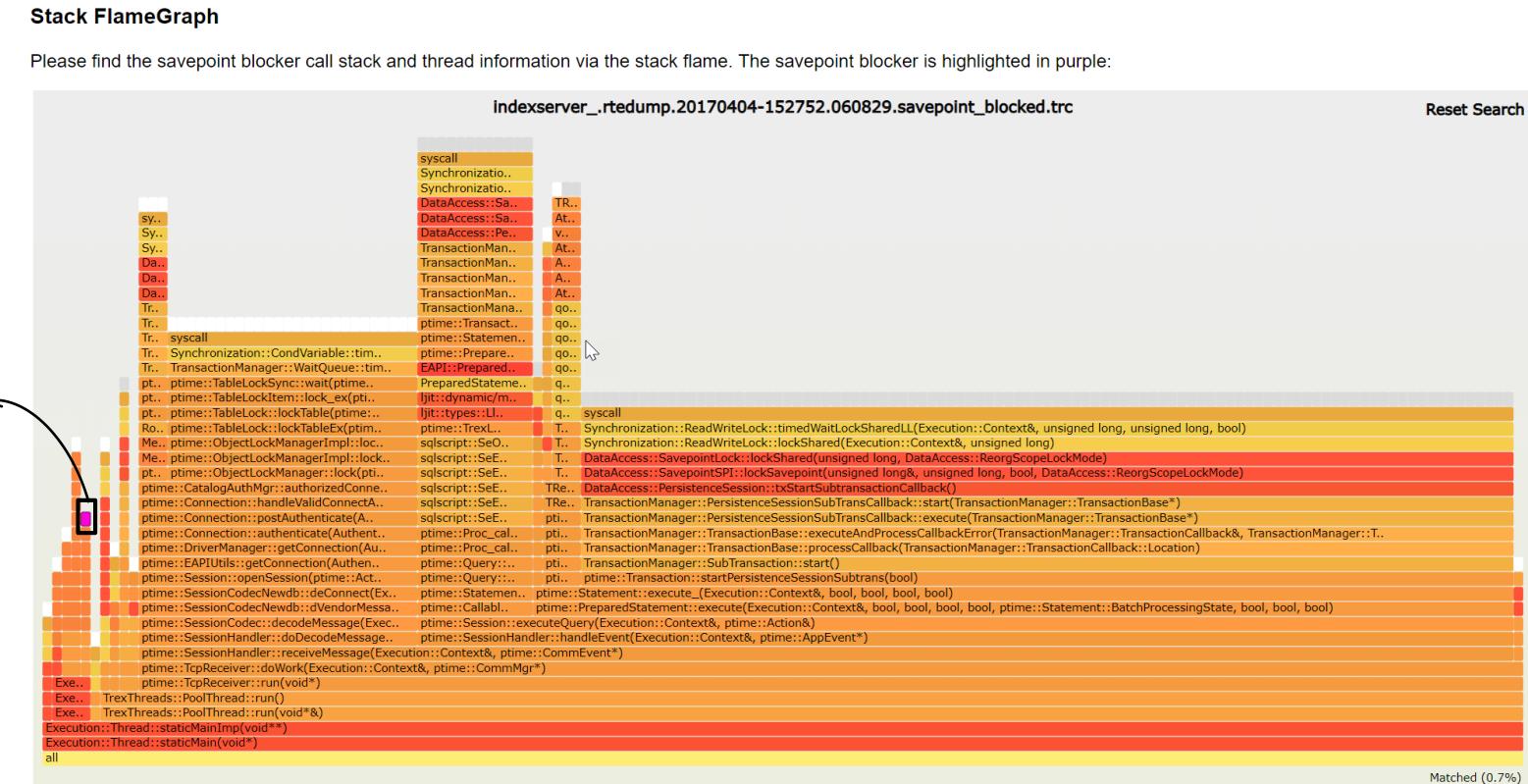
**Savepoint Blocker**

As Savepoint is blocked, The key question is to find out the one blocks the savepoint.  
The Savepoint has to wait for an exclusive ConsistentChangeLock while preparing to enter the critical phase. The ConsistentChangeLock is owned by the following thread(s):

```
67563[thr=63702]: JobWrk0019, TID: 8814, UTID: 7591174856, CID: 318948, LCID: 318948, parent: 64661, SQLUserName: "SAP_ARCHIVE", AppUserName: "CaITH", AppName: "HDBStudio", ConnCtx: 318948 (LDBID: 2, LCID: 318948), StmtCtx: (1) 1369875433359431 {Parent: 318948, MemoryLimit: 512, ObjHdl: 139075432672256, User: "SAP_ARCHIVE", Schema: "SAP_ARCHIVE", SesCtx: (ObjHdl: 139112808425472, CID: 318948, LCID: 318948, User: "SAP_ARCHIVE", Schema: "SAP_ARCHIVE", Ver: 8, CtxID: 18935)}, type: "JobWorker", method: "", detail: "", command: "" at 0x00007f21365c3acc in void UnifiedTable::impl::convertValueToString(char const&, unsigned long&, char const*, unsigned long, UnifiedTable::PersDataTypeDescriptor const*, UnifiedTable::ConvertBuffer&)+0xec at Convert.cpp:468 (libhbdbunifiedtypes.so)
```

# SAP HANA dump analyzer – Savepoint Analyzer

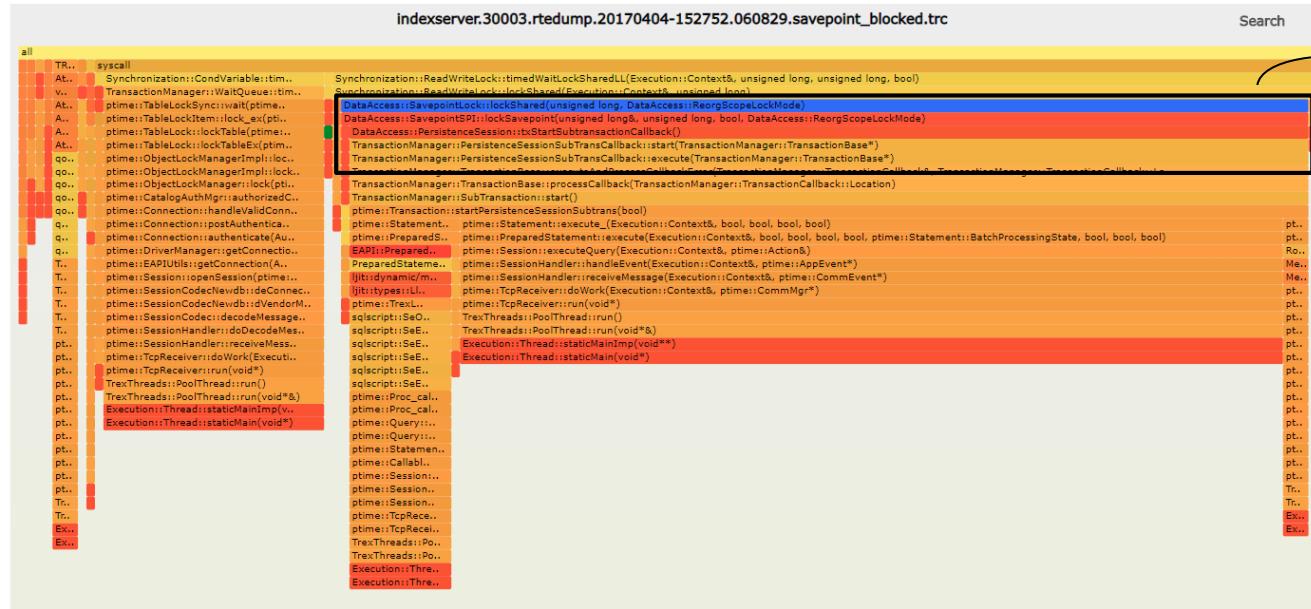
Since savepoint blocker is captured, it needs to be further understood for the behavior of the savepoint blocker, e.g. whether this is an expensive operation which running for long? There are many savepoint blockers and they need to take time till they are all finished? In order to show more information or provide the end user capability to search more information, stack flame graph is provided with savepoint blocker highlighted in purple by default.



# SAP HANA dump analyzer – Savepoint Analyzer

## Reverse Stack FlameGraph

The waiting savepoint will block all subsequent shared and exclusive requests for ConsistentChangeLock and many threads can queue up. The following Reverse Stack Flame highlights the queued up threads blocked by savepoint in blue.



- The waiting savepoint will block all subsequent shared and exclusive requests for ConsistentChangeLock and many threads can queue up. The queued up threads blocked by savepoint are highlighted in blue in the reverse stack flame graph
- The savepoint thread is highlighted in green.

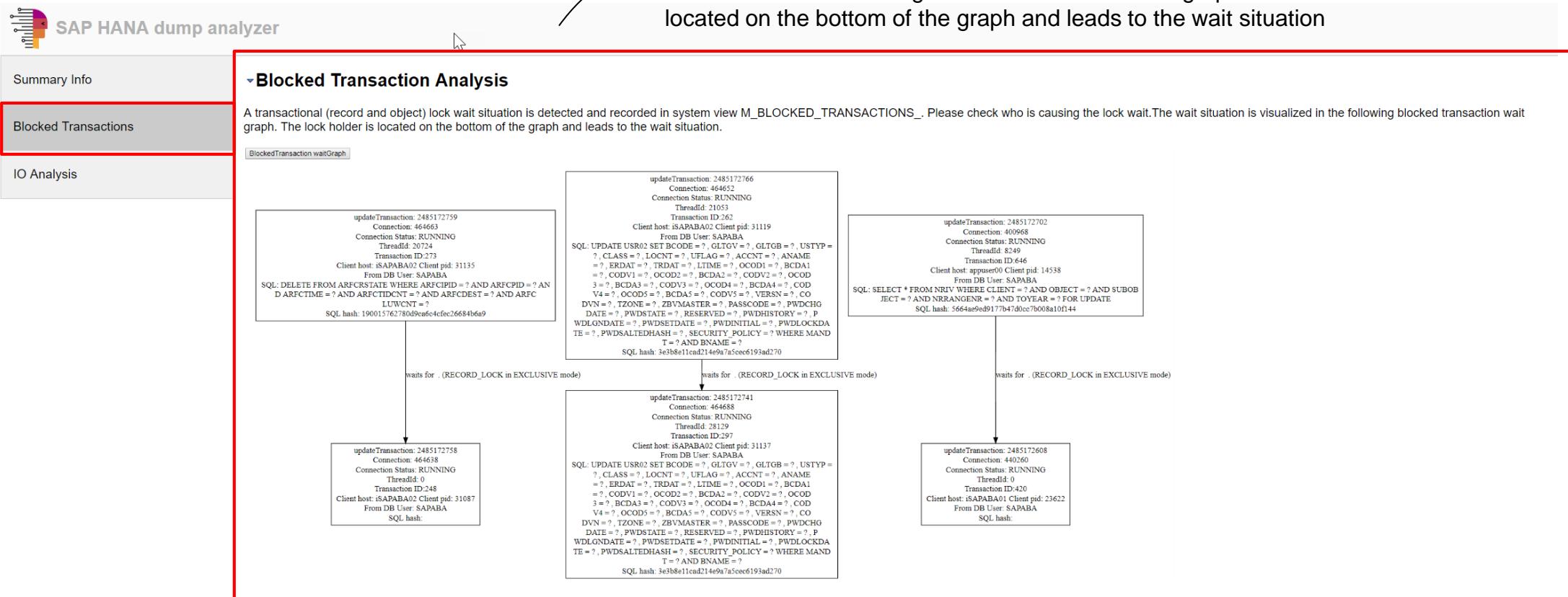
# SAP HANA dump analyzer – WaitGraph Analyzer

If the savepoint is blocked over certain defined threshold, the savepoint analyzer will receive the thread blocks savepoint and print in the savepoint blocker section.

The screenshot shows the SAP HANA dump analyzer interface. The left sidebar has three tabs: 'Summary Info', 'Savepoint Blocked Analysis', and 'WaitGraph Analysis'. The 'WaitGraph Analysis' tab is highlighted with a red box. The main content area has a heading 'WaitGraph Analysis' and a sub-section 'WaitGraph'. It contains a message: 'A wait situation has been detected, please check the WaitGraph. The blocker of the wait situation is located on the bottom of the WaitGraph'. Below this is a 'WaitGraph' diagram. The diagram consists of a large number of small boxes representing threads, connected by arrows indicating dependencies. A specific cluster of nodes at the bottom is highlighted in red, indicating they are the blockers. A legend on the right side of the graph provides a key for the colors used in the nodes.

# SAP HANA dump analyzer – Blocked Transaction Analyzer

If a transactional (record and object) lock wait situation is detected and recorded in system view `M_BLOCKED_TRANSACTIONS`, the blocked transactions will be automatically analyzed by SAP HANA dump analyzer. The wait situation is visualized in the following blocked transaction wait graph. The lock holder is located on the bottom of the graph and leads to the wait situation



# SAP HANA dump analyzer – IndexHandle State Analyzer

If a wait situation linked with indexHandle is detected and affects many thread waiting, this will be automatically analyzed by SAP HANA dump analyzer. The wait situation will be visualized to the waitgraph, the blocker of the wait situation is located on the bottom layer of the graph and leads to the wait situation.

**SAP HANA dump analyzer**

**Summary Info**

**IndexHandle States Analysis**

**IndexHandle Internal states**

A wait situation has been detected. The wait situation is linked to the index handle, a column store table level synchronization lock that is mainly used for delta merge synchronization. Please check the one causing the wait situation. Typical known scenarios are described in SAP Note [2057046](#). The blocker of the wait situation is located on the bottom layer of the graph and leads to the wait situation

The blocker(s) are:

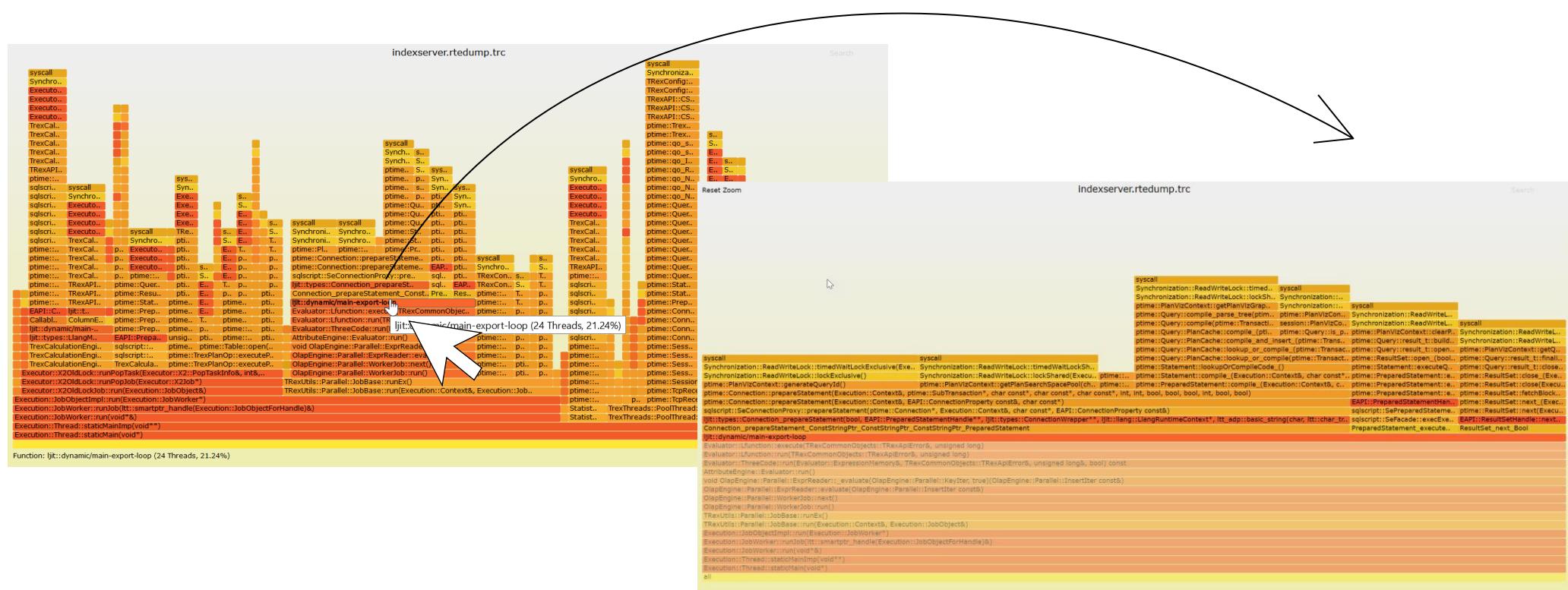
```
Blocker:  
thr=31867  
NextState: search_delta  
Synchronization: BinarySemaphore.timedWait  
Execution: JobBarrier::wait  
Execution: JobObjectImpl::wait  
Execution: JobContextImpl::wait  
Execution: JobContext::waitFinalize  
TRexAPI::TableLoad::loadLocal  
TRexAPI::TableLoad::handle  
TRexAPI::SpeculativePreloadThread::run  
TrexThreads::PoolThread::run  
TrexThreads::PoolThread::run  
Execution: Thread::staticMainImp  
Execution: Thread::staticMain  
SQL: null
```

**Open indexHandleStates**

The diagram shows a complex waitgraph with multiple nodes representing different states of the index handle. Nodes include 'thr=31867' (IndexHandle Internal state), 'thr=31867' (SearchDelta), 'thr=31867' (IndexHandle Internal state), 'thr=31867' (IndexHandle Internal state), 'thr=31867' (IndexHandle Internal state), 'thr=31867' (IndexHandle Internal state), and 'thr=31867' (IndexHandle Internal state). Edges represent transitions between these states, with some edges being highlighted in red, indicating the blocker of the wait situation.

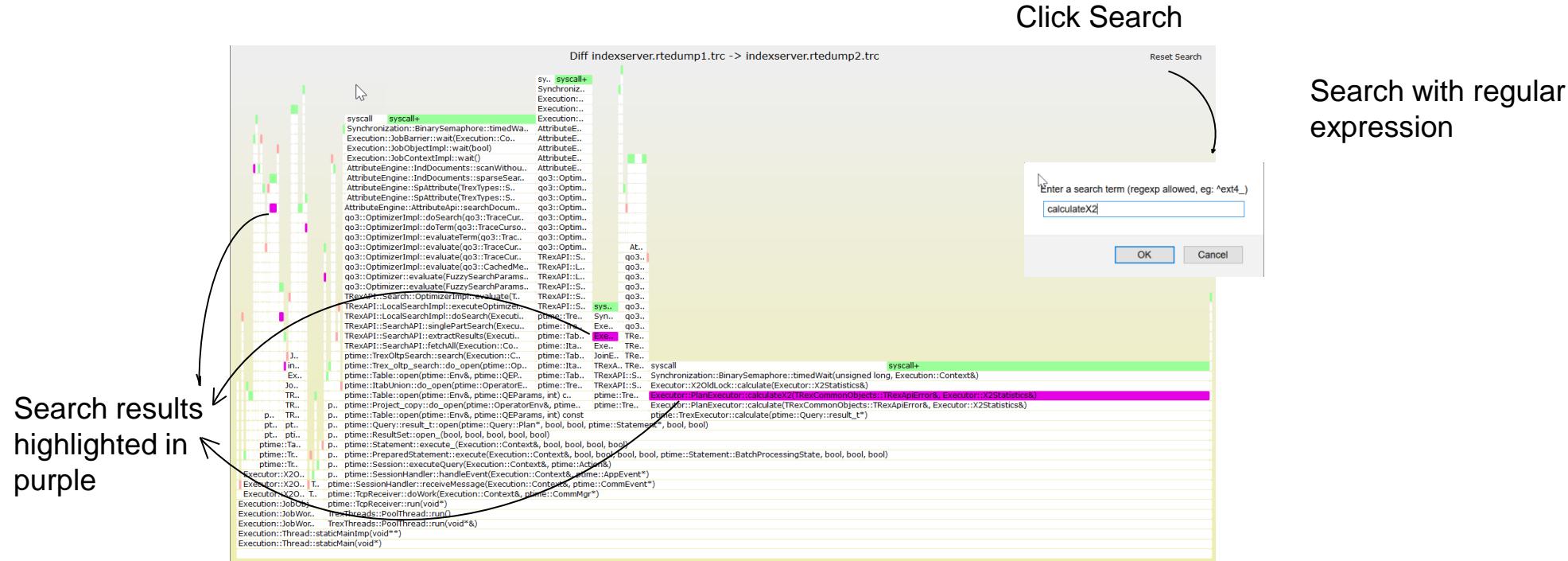
# SAP HANA dump analyzer Tips and Tricks

- Click on Flame to zoom in



# SAP HANA dump analyzer Tips and Tricks

- Zoom via browser means with CTRL+, CTRL-
- Flame Graph Search with regular expressions



# References & Acknowledgements

- Most ideas and generation of Flame Graph are inspired by Brendan Gregg on <http://www.brendangregg.com/flamegraphs.html>

# Thank you.

Contact information:

Nina Li      [nina.li01@sap.com](mailto:nina.li01@sap.com)  
SAP MCC Business Down Management Team

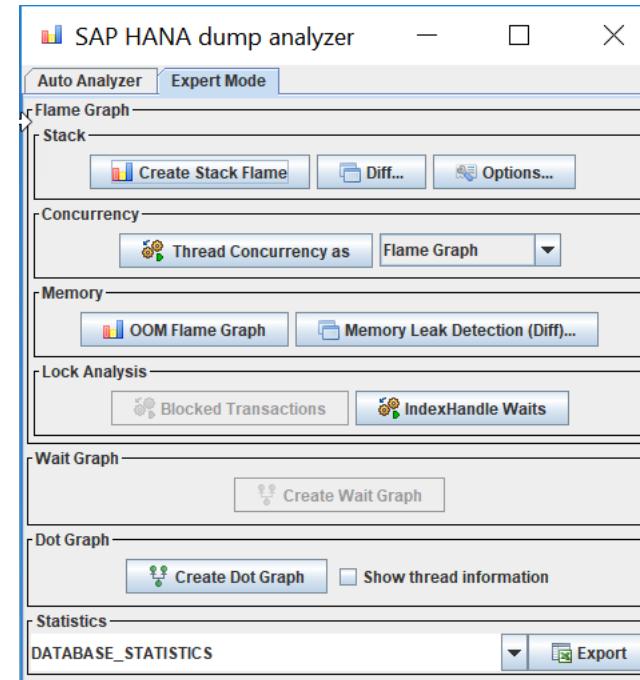
# **Appendix 1**

## **SAP HANA dump analyzer Expert Mode**



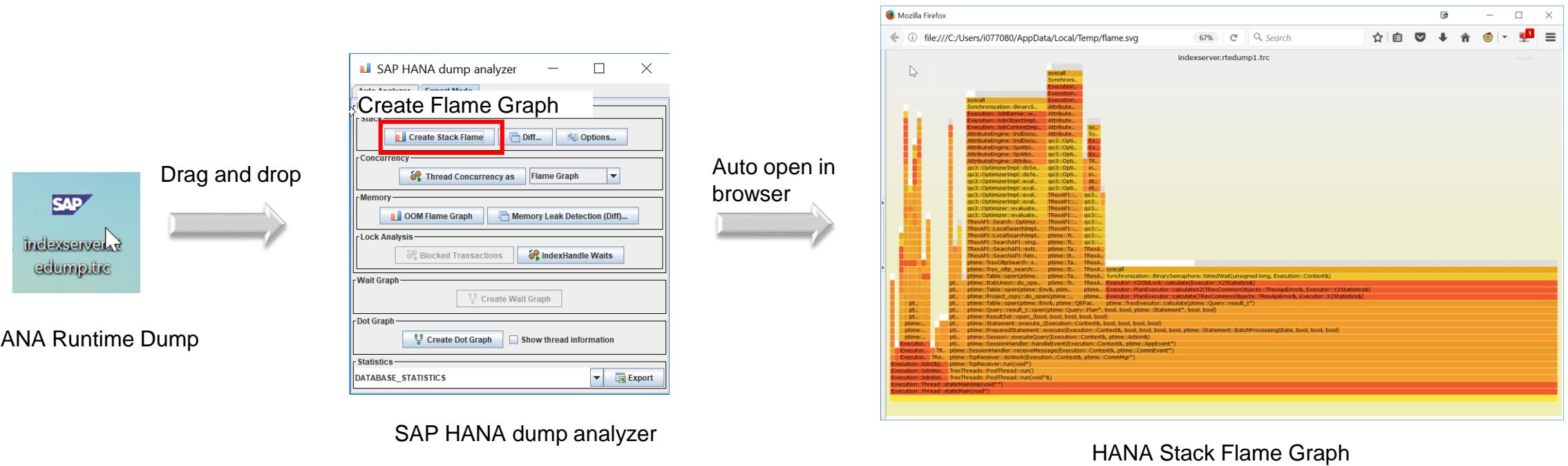
# SAP HANA dump analyzer – Expert Mode

In case Issue is not detected automatically. The end user will be redirect to the expert mode to analyze the issue manually. You could also change to the "export mode" tab in case you want to carry out some analysis manually



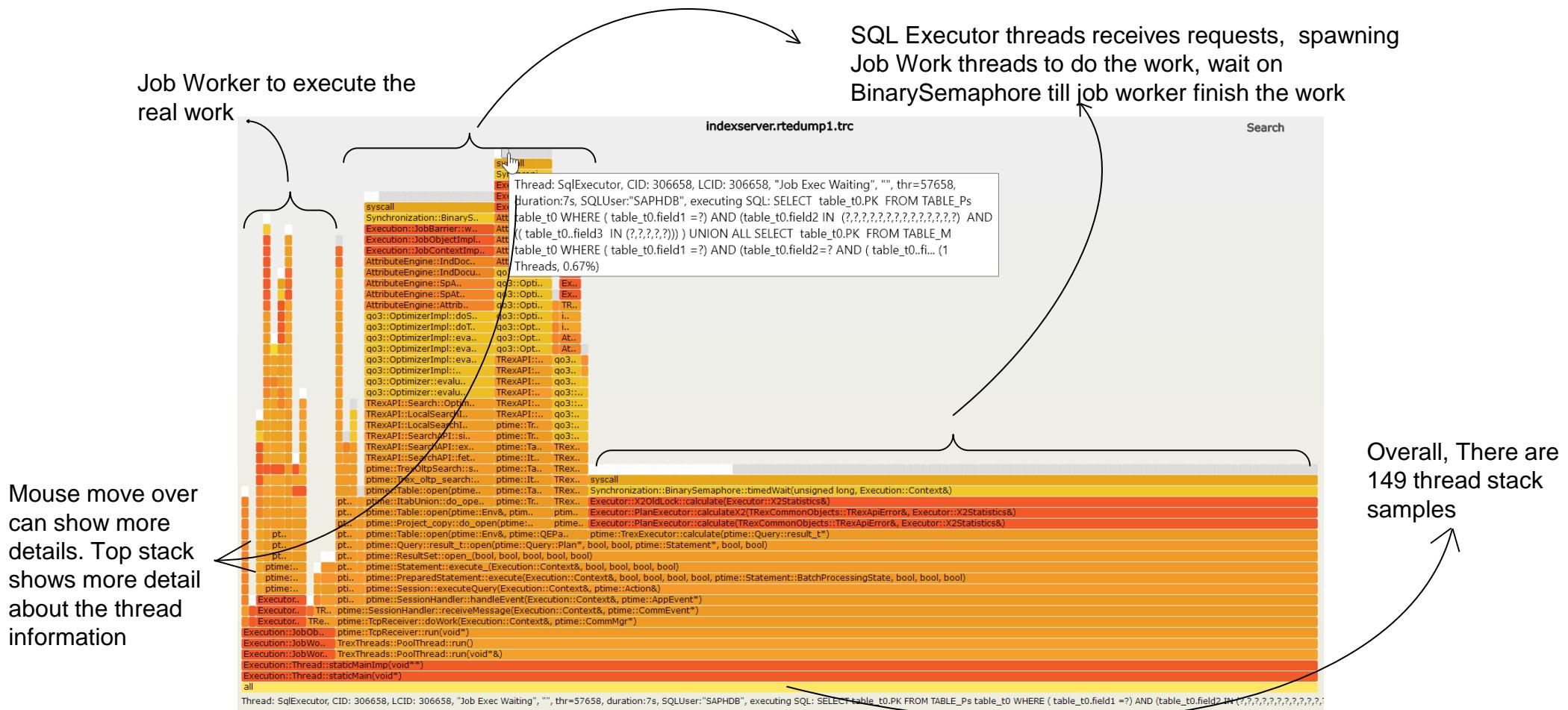
# HANA Stack Flame Graph: Use SAP HANA dump analyzer to generate Stack Flame Graph

- SAP HANA dump analyzer has a easy to use interface: just drag and drop your runtime dump!



# HANA Stack Flame Graph: Example 1

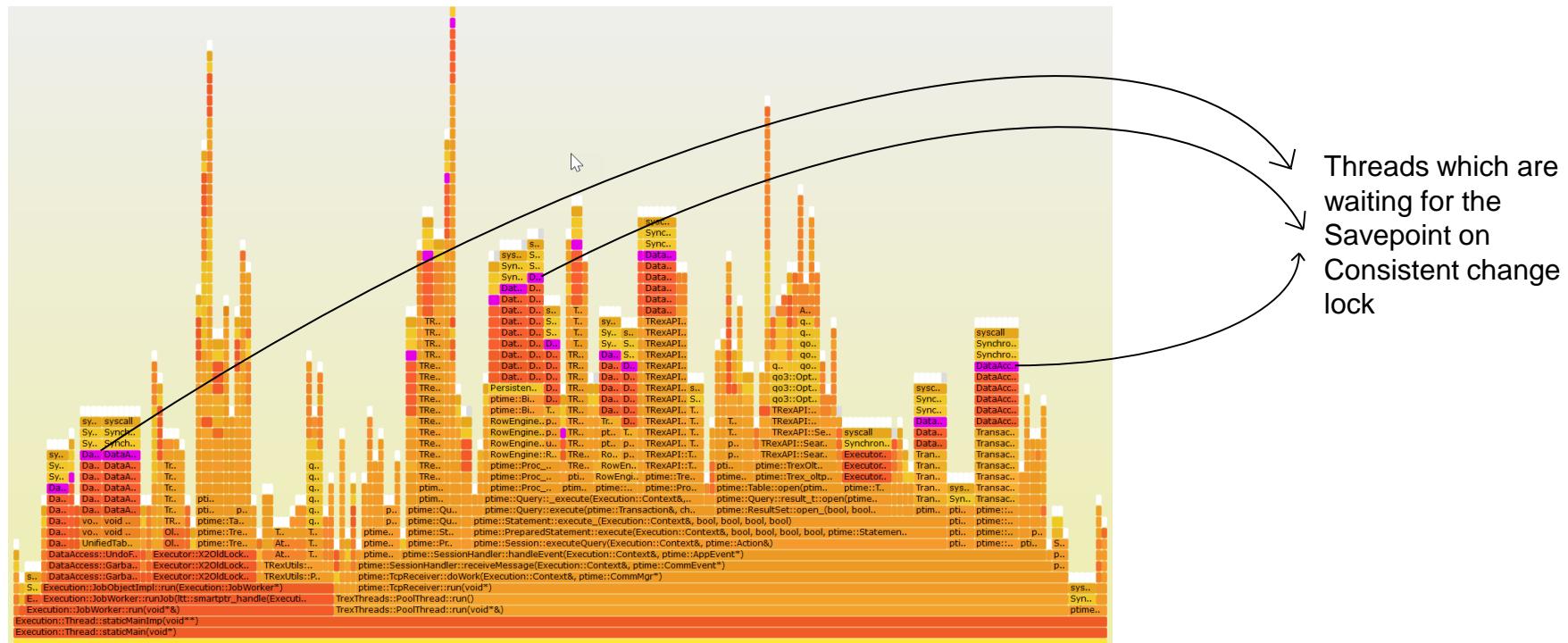
## Use SAP HANA dump analyzer to generate Stack Flame Graph



# HANA Stack Flame Graph: Example 2

## Use SAP HANA dump analyzer to generate Reverse Stack Flame Graph

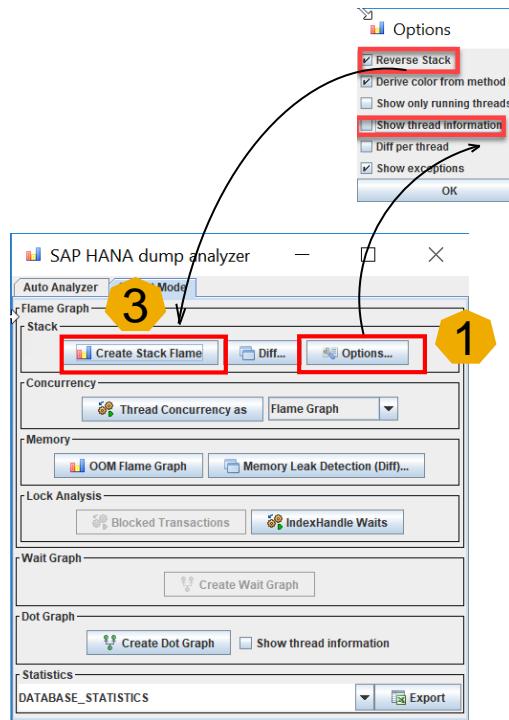
- If the majority of the threads are waiting for the same lock or resources (e.g. IO) however they are on the very different call stacks, it might be the stack flame graph does not show the problem at all: i.e. there is no obvious bigger block on the stack flame graph.



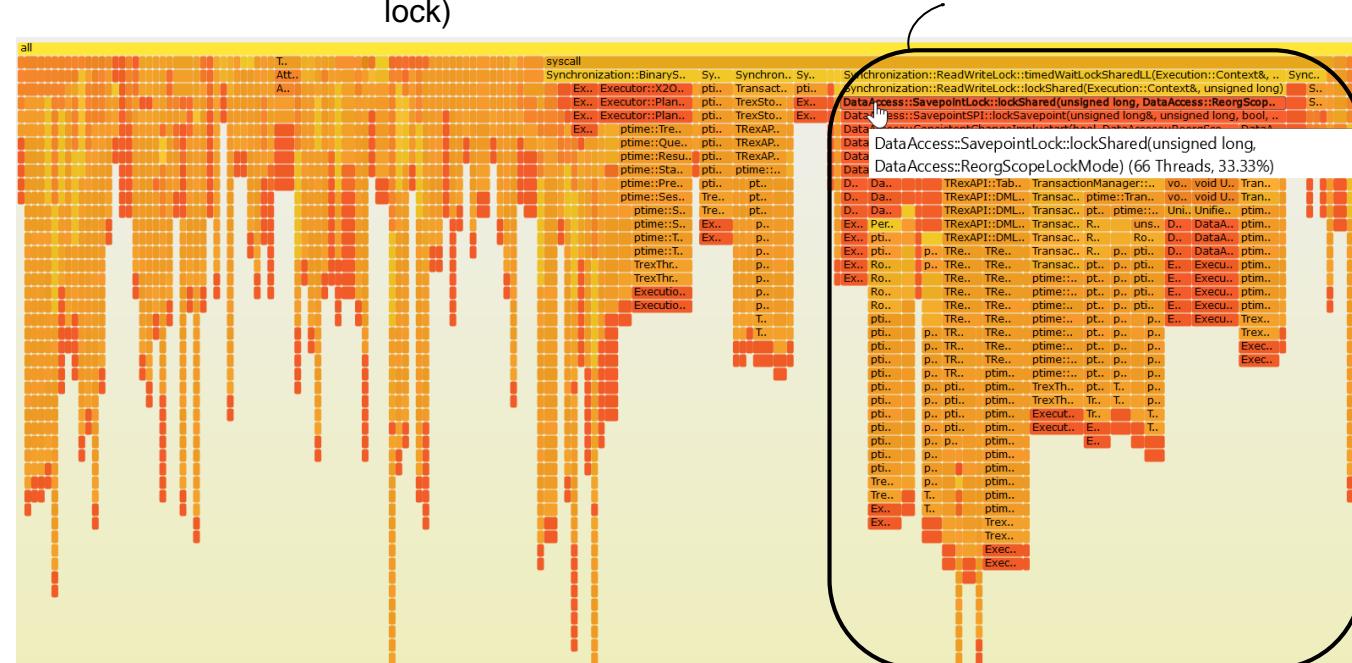
# HANA Stack Flame Graph: Example 2

## Use SAP HANA dump analyzer to generate Reverse Stack Flame Graph

- A reverse stack flame graph is available to align and sort the call stack samples on the top stack flame (i.e. standard flame graph aligns and sorts based on bottom stack frame). In case the different threads on different call stacks are waiting for same resources or locks, the issue would be more visible in reverse stack flame graph.



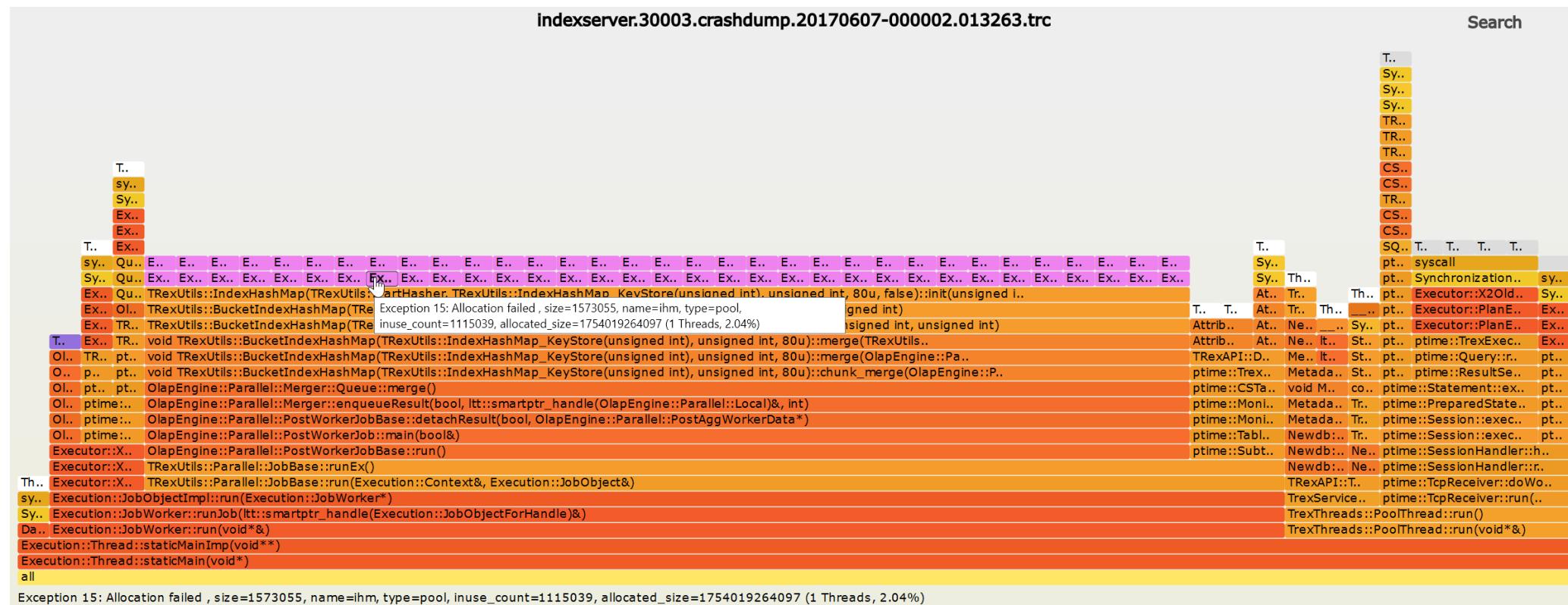
Auto open in browser  
→



# HANA Stack Flame Graph: Example 3

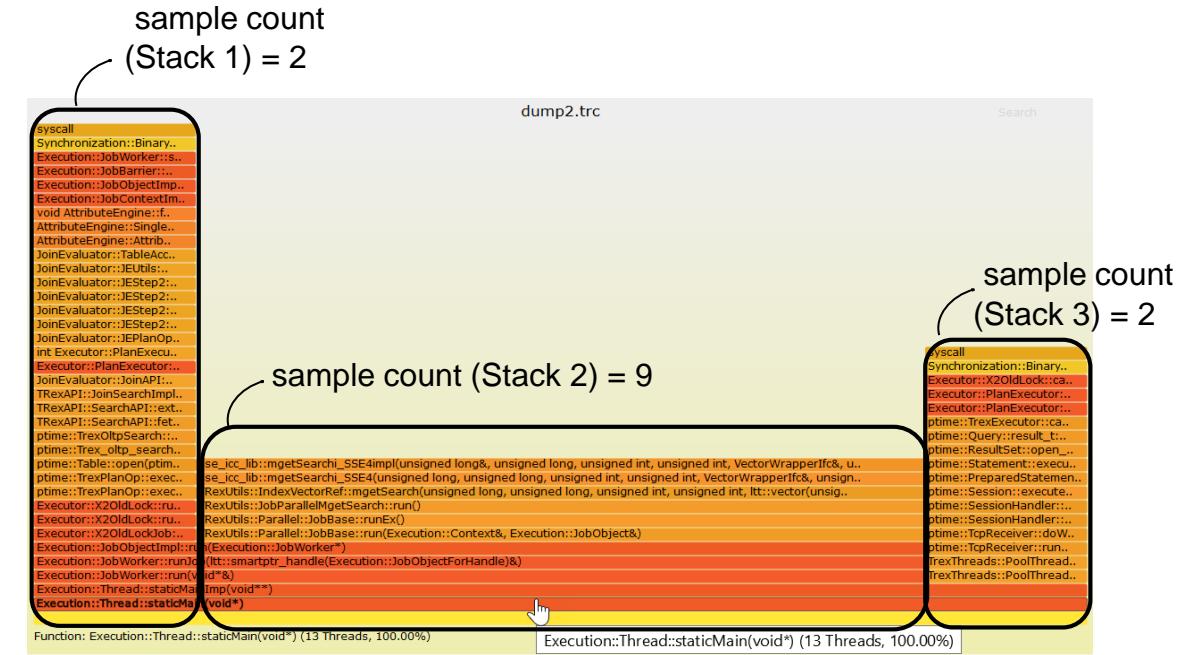
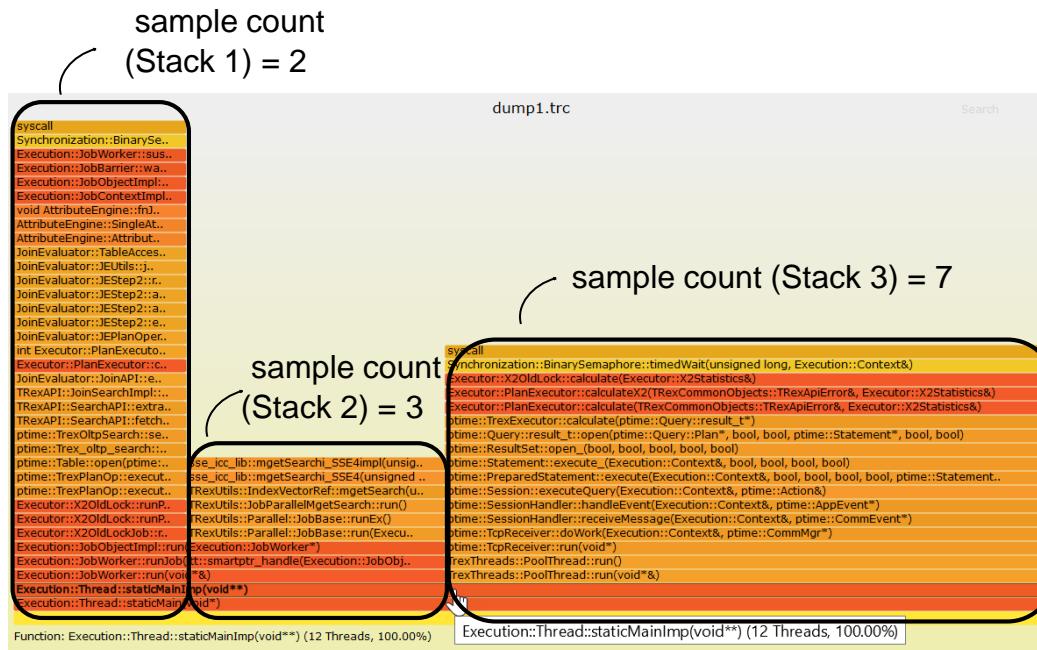
## Use SAP HANA dump analyzer to generate Stack Flame Graph

- If the exceptions are shown in certain stack frames, they are by default highlighted in purple.



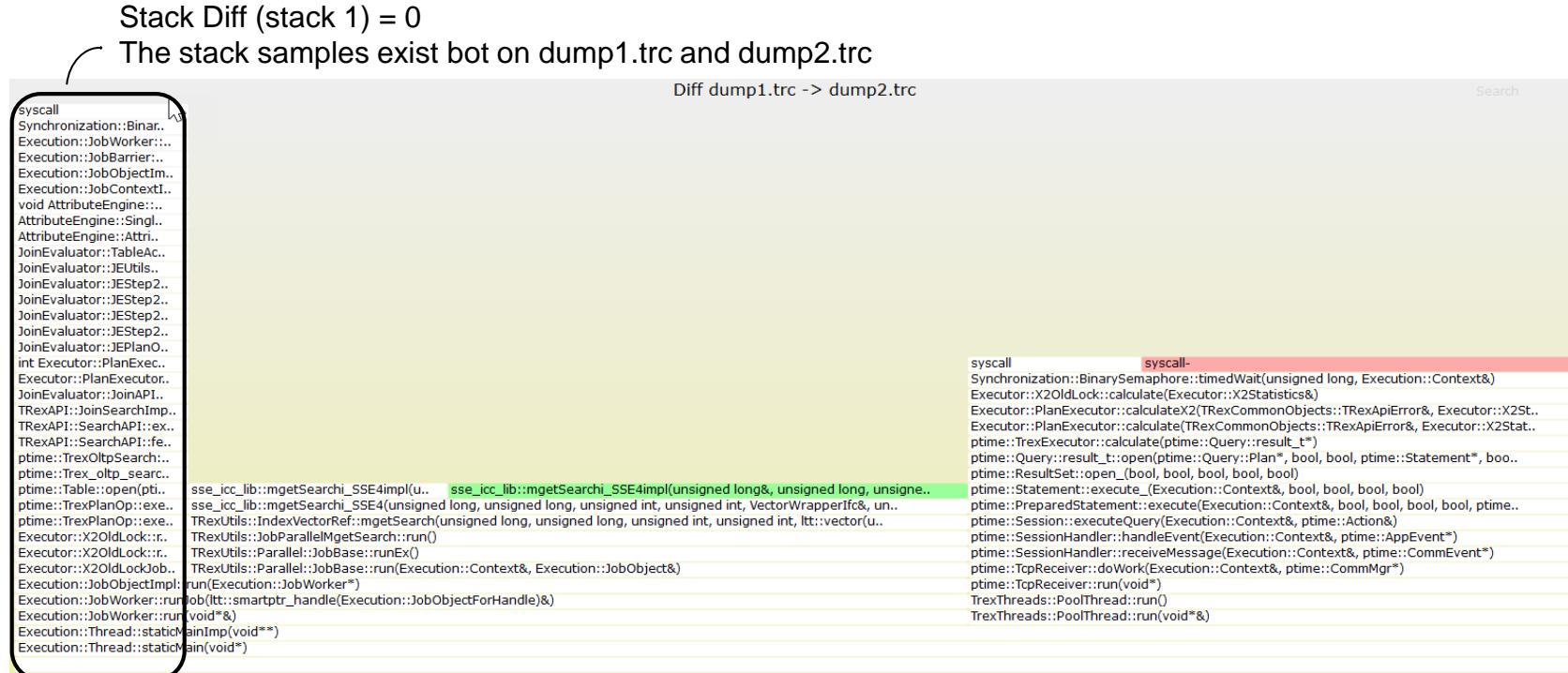
# HANA Stack Diff Flame Graph

- To explain the Stack Diff, a small example is shown in the following.



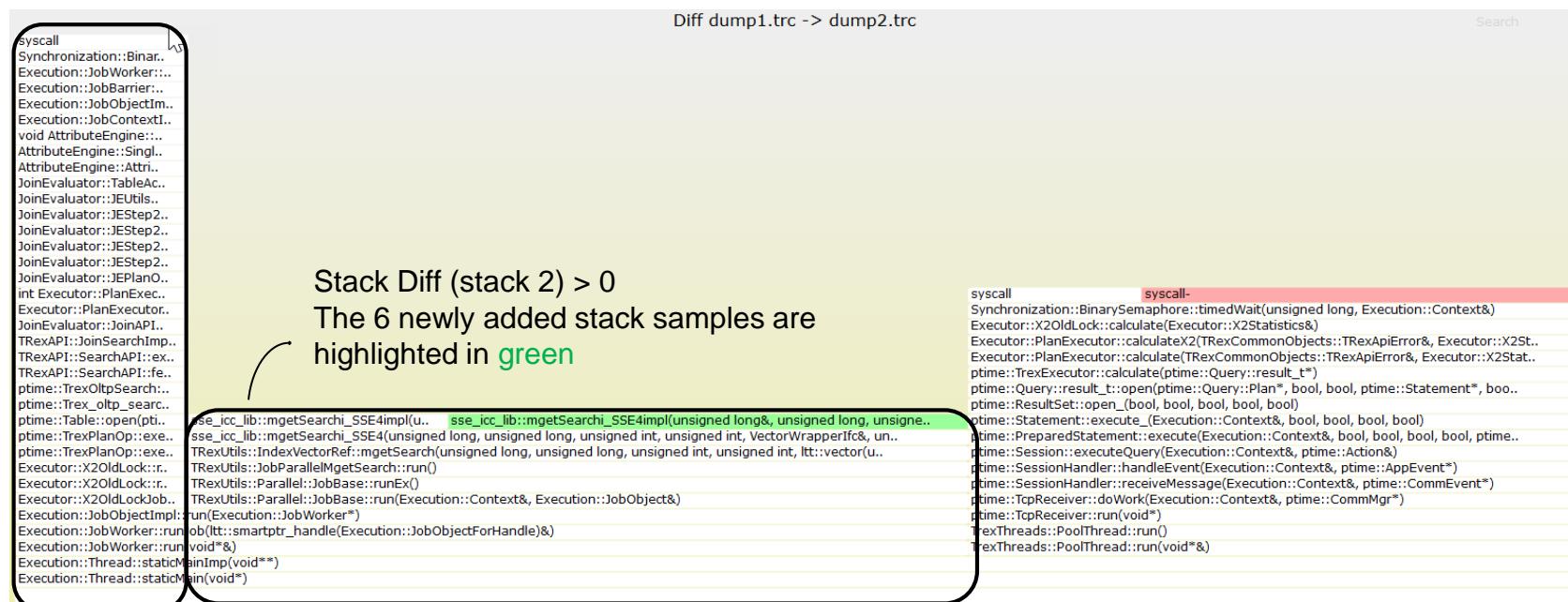
# HANA Stack Diff Flame Graph

- Stack Diff = For each stack (stack sample count in dump2.trc – stack sample count in dump1.trc)
- If Stack Diff = 0: The stack samples exist both on dump1.trc and dump2.trc, the stack samples are shown in stack diff graph, no highlight.



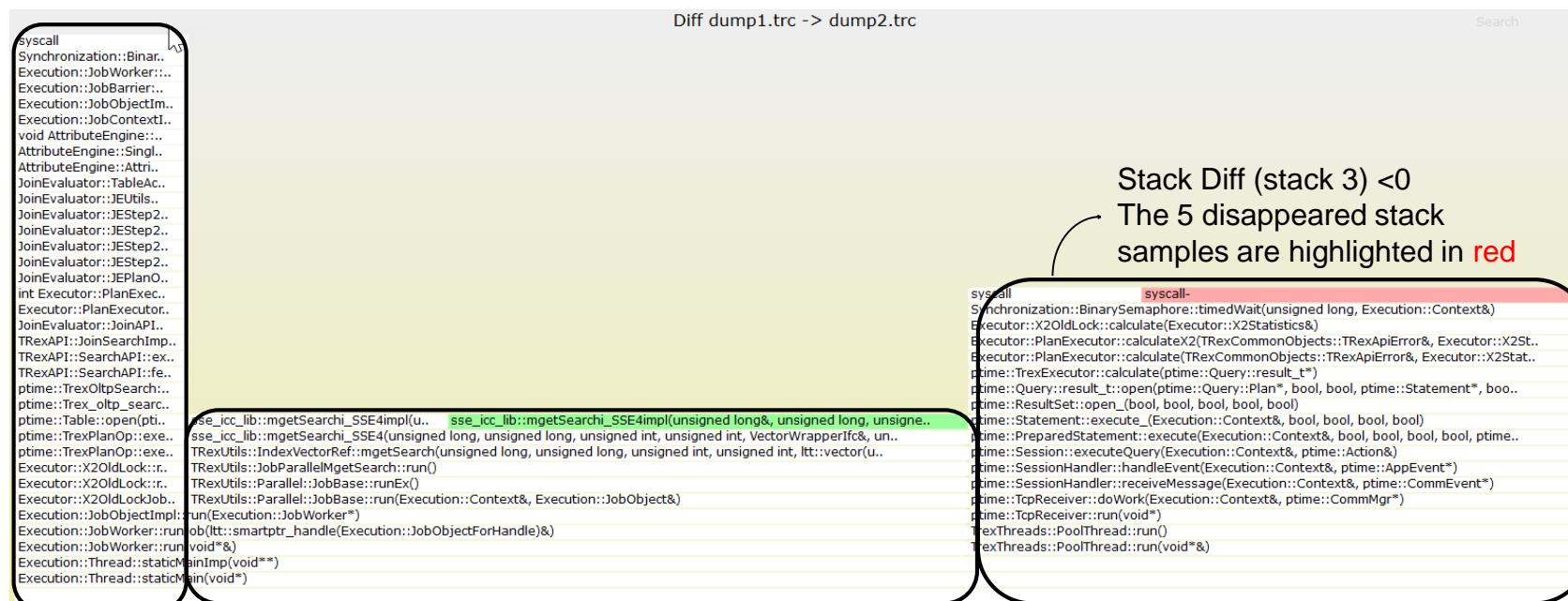
# HANA Stack Diff Flame Graph

- Stack Diff = For each stack (stack sample count in dump2.trc – stack sample count in dump1.trc)
  - If Stack Diff = 0: The stack samples exist both on dump1.trc and dump2.trc, the stack samples are shown in stack diff graph, no highlight.
  - Else if Stack Diff > 0: The stack samples are more added in dump2.trc than dump1.trc. The newly created stack samples are highlighted in green



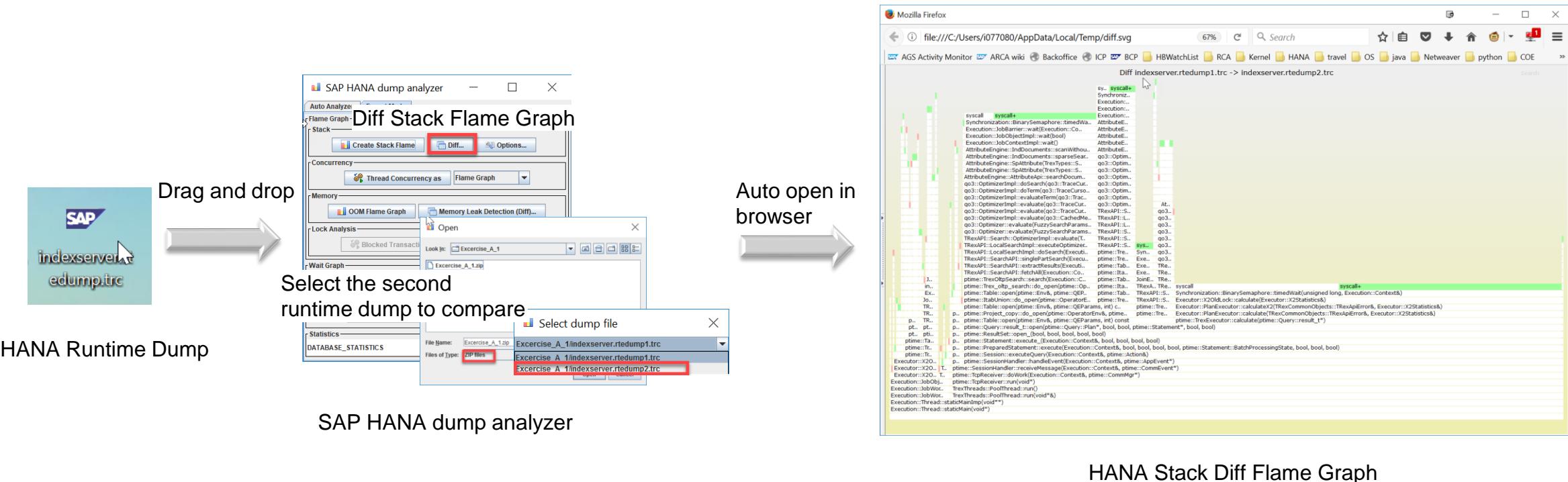
# HANA Stack Diff Flame Graph

- Stack Diff = For each stack (stack sample count in dump2.trc – stack sample count in dump1.trc)
- If Stack Diff = 0: The stack samples exist both on dump1.trc and dump2.trc, the stack samples are shown in stack diff graph, no highlight.
- Else if Stack Diff > 0: The stack samples are more added in dump2.trc than dump1.trc. The newly created stack samples are highlighted in green
- Else Stack Diff < 0: The stack samples are (all/partly) disappeared in dump2.trc compared to dump1.trc. The disappeared stack samples are highlighted in red



# HANA Stack Diff Flame Graph: Use SAP HANA dump analyzer to generate Stack Diff Flame Graph

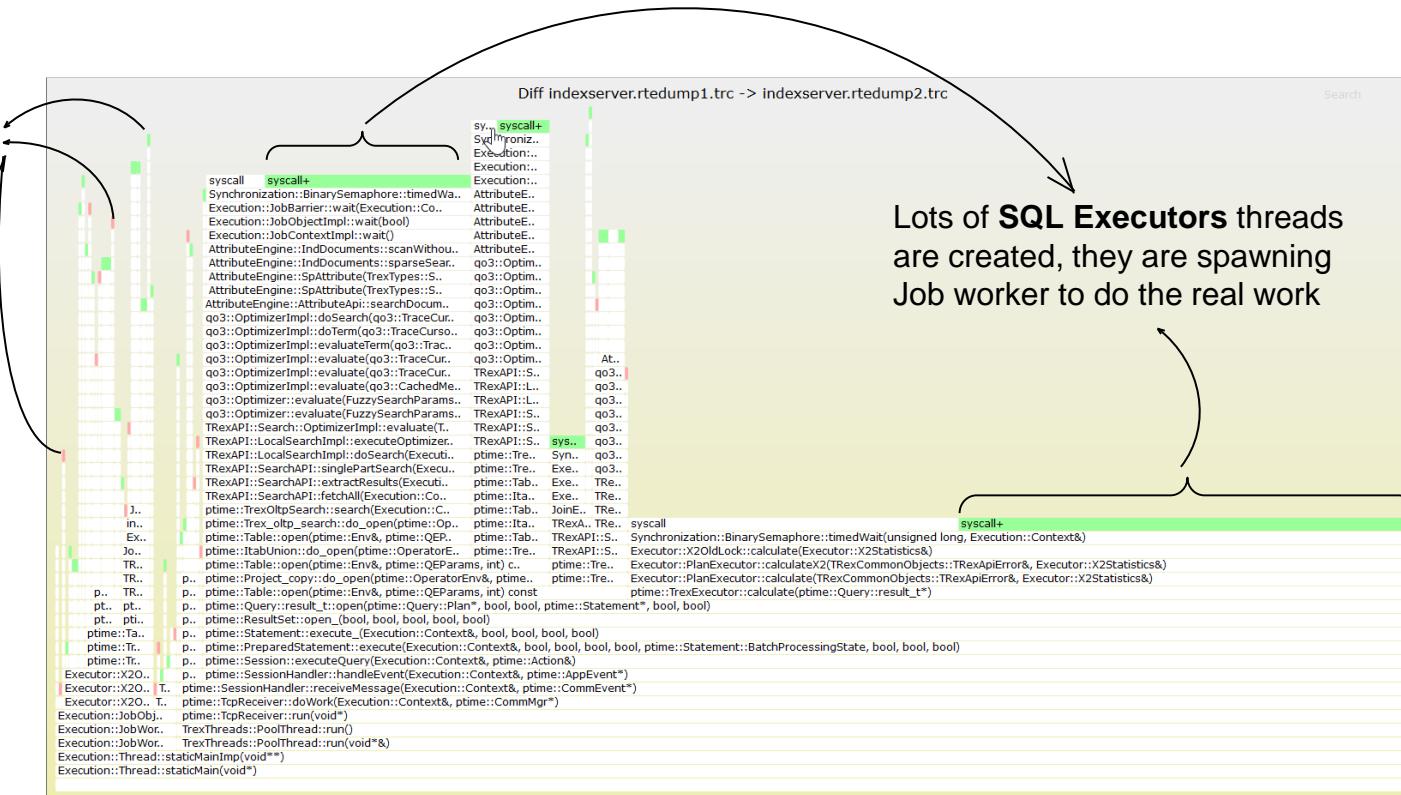
- To conclude the progress HANA is making between two runtime dump, the stack diff can be visualized using HANA Dump analyzer



# HANA Stack Diff Flame Graph: Example 4

## Use SAP HANA dump analyzer to Generate Stack Diff Flame Graph

**Job Workers** are not much created: some Job workers sample stacks are vanished, some Job worker sample stacks are newly created, overall job worker number is not changed



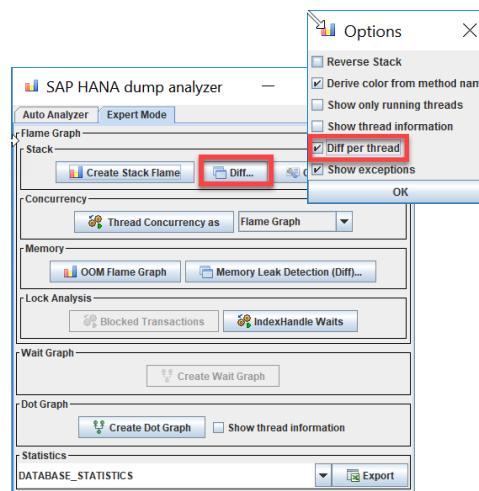
The Stack Diff Flame Graph shows the following

- The SQL Executors are continuing receiving requests.
- The SQL Executors spawns Job workers to finish the work.
- However Job workers are not processing fast enough.
- The requests are queueing up.
- The HANA index server stands still.

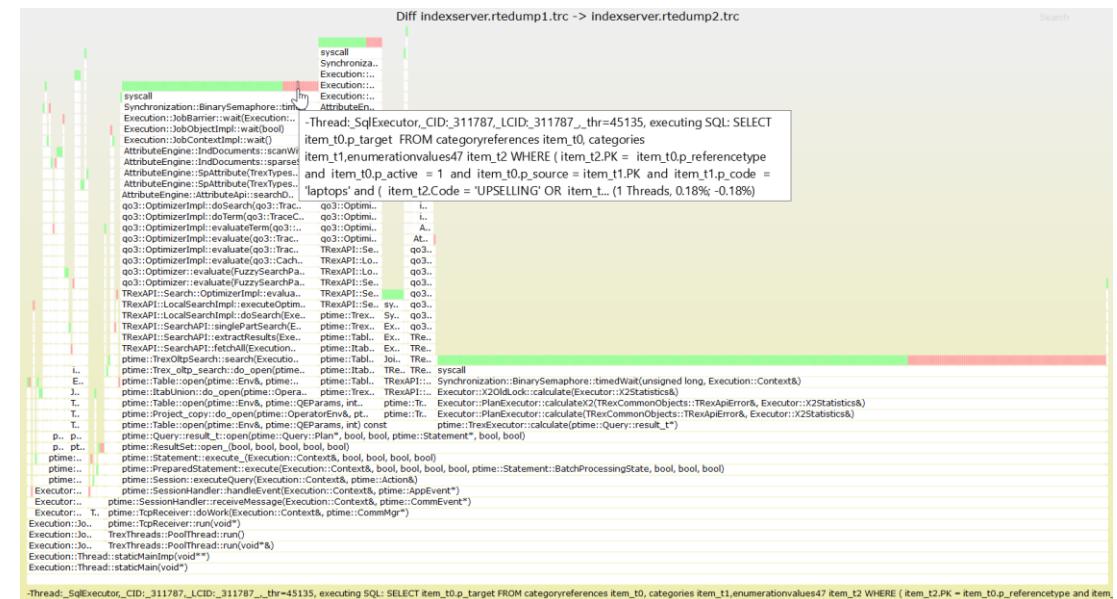
# HANA Stack Diff Flame Graph: Example 4

## Use SAP HANA dump analyzer to Generate Stack Diff Flame Graph

- By default, stack diff is not considering each stack sample as a different one. i.e. in most of the case, whether the same call stacks are coming from the same threads does not make a difference for seeing the diff of the call stacks.
- In some of the cases, a stack diff view considering each stack sample as a different one might be important, e.g. to conclude whether the threads are making progress or not. Thus there is an option available “Diff per thread” to achieve this.
- If option “Diff per thread” option is checked, a stack diff flame graph with non/ very little thread stack samples highlighted in green or red means HANA makes no/ little progress.

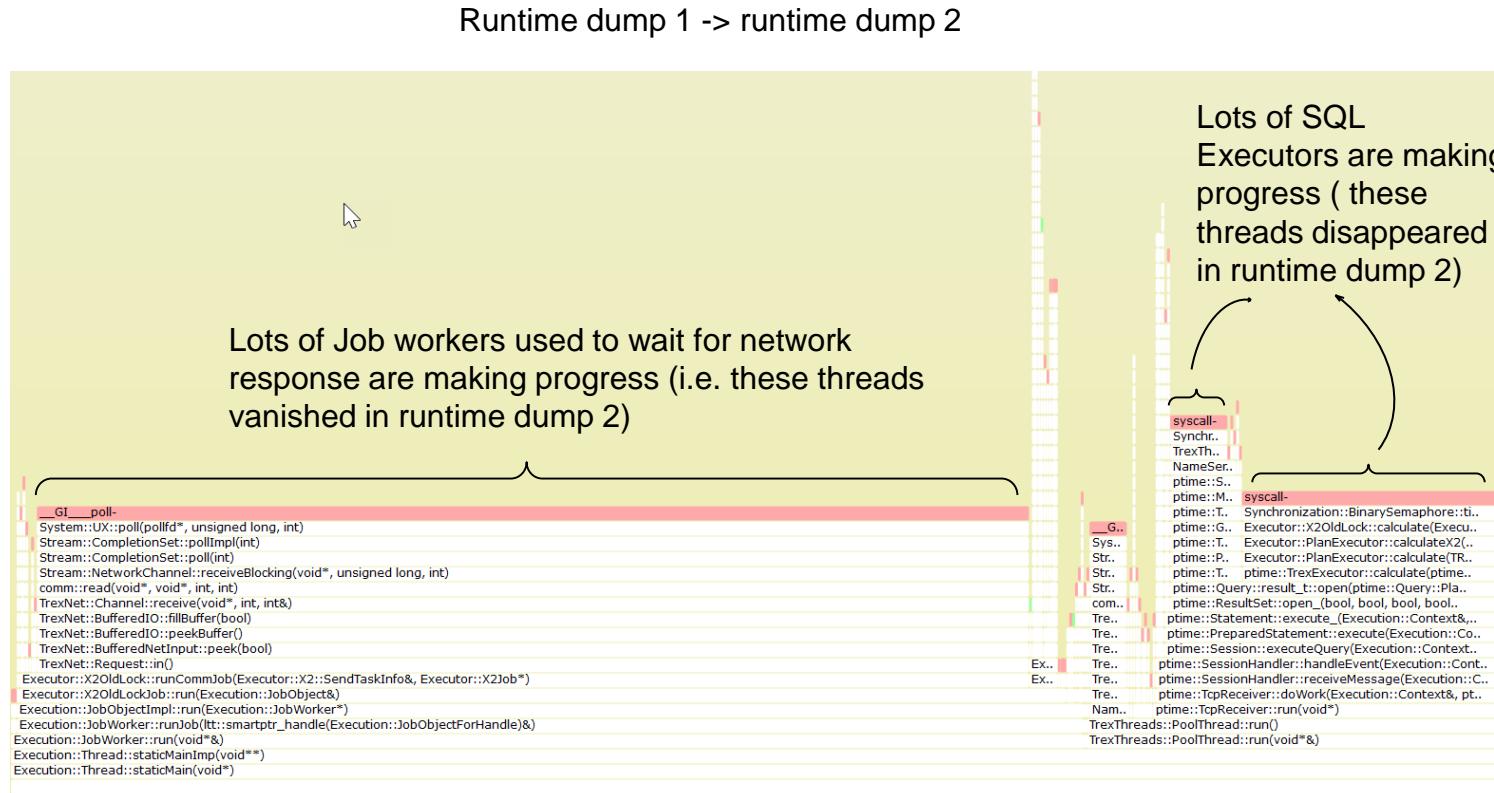


Diff per thread



# HANA Stack Diff Flame Graph: Example 5

## Use SAP HANA dump analyzer to Generate Stack Diff Flame Graph

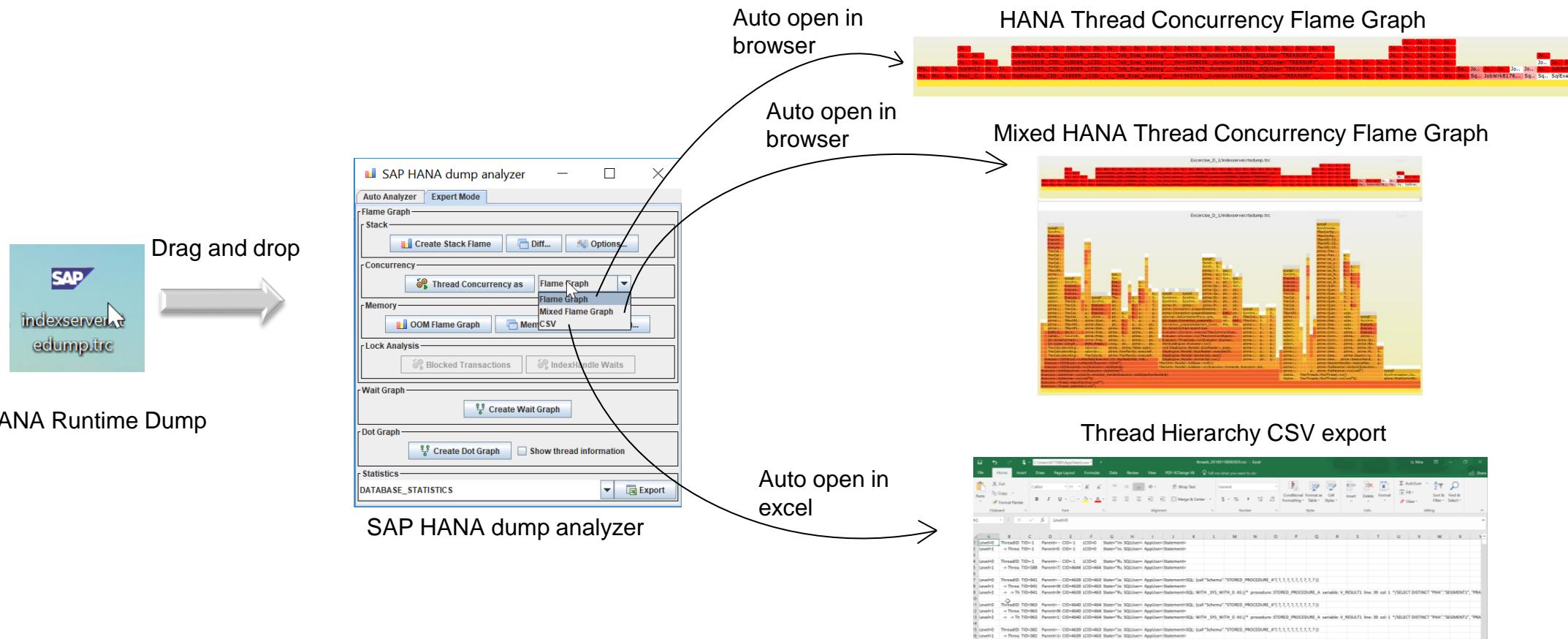


The Stack Diff Flame Graph shows the following

- The Job workers are making progress on runtime dump 2 compared to runtime dump 1: they are not all waiting for network response any more.
- The SQL Executors are also making progress on runtime dump 2 compared to runtime dump 1. The SQL Executors which were all waiting for Job worker to finish the work in runtime dump 1 are making progress on runtime dump 2: most likely because Job workers are not blocked on network response anymore.
- A blocking situation in runtime dump 1 is resolved in runtime dump 2.

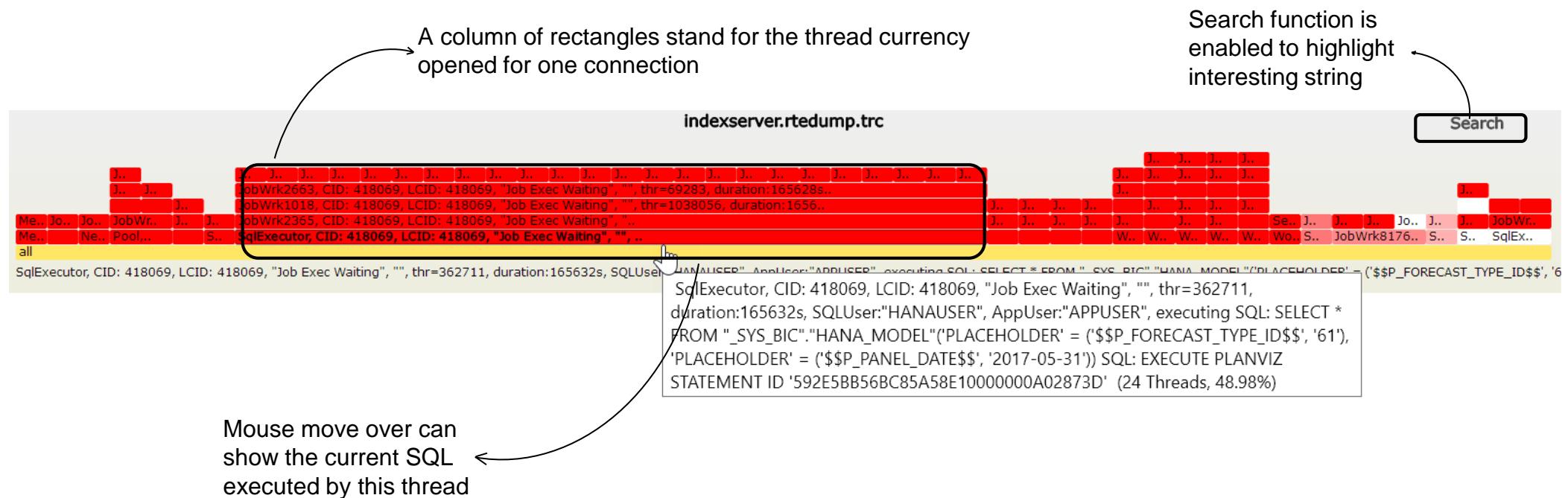
# HANA Thread Concurrency Flame Graph: Use SAP HANA dump analyzer to Analyze Thread Concurrency

- SAP HANA dump analyzer has a easy to use interface: just drag and drop your runtime dump!



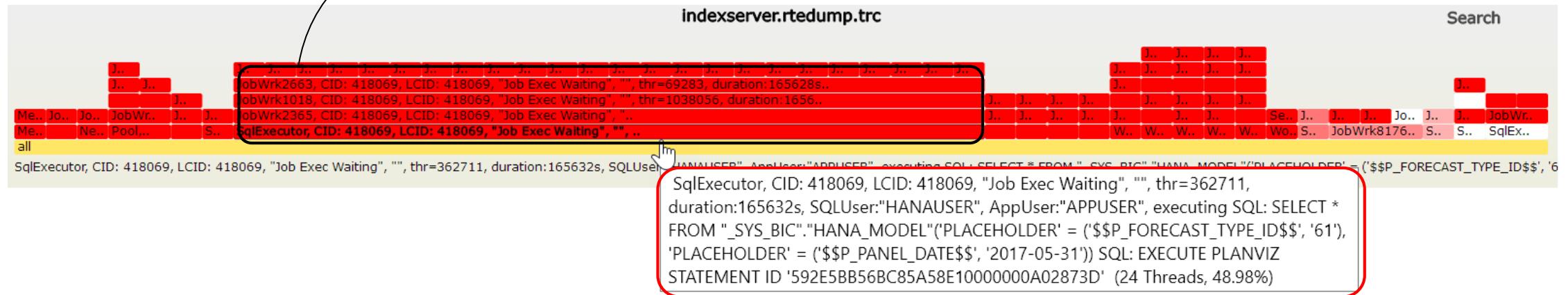
# HANA Thread Concurrency Flame Graph

- HANA Thread Concurrency Flame Graph visualizes the thread concurrency on thread tree.
  - In case a single connection spawns lots of threads for execution, a block of concurrent threads will be visualized.
  - Concurrency > 1 is visualized. (i.e. if one thread is executing one query, it's not shown in the Thread Concurrency Flame Graph)

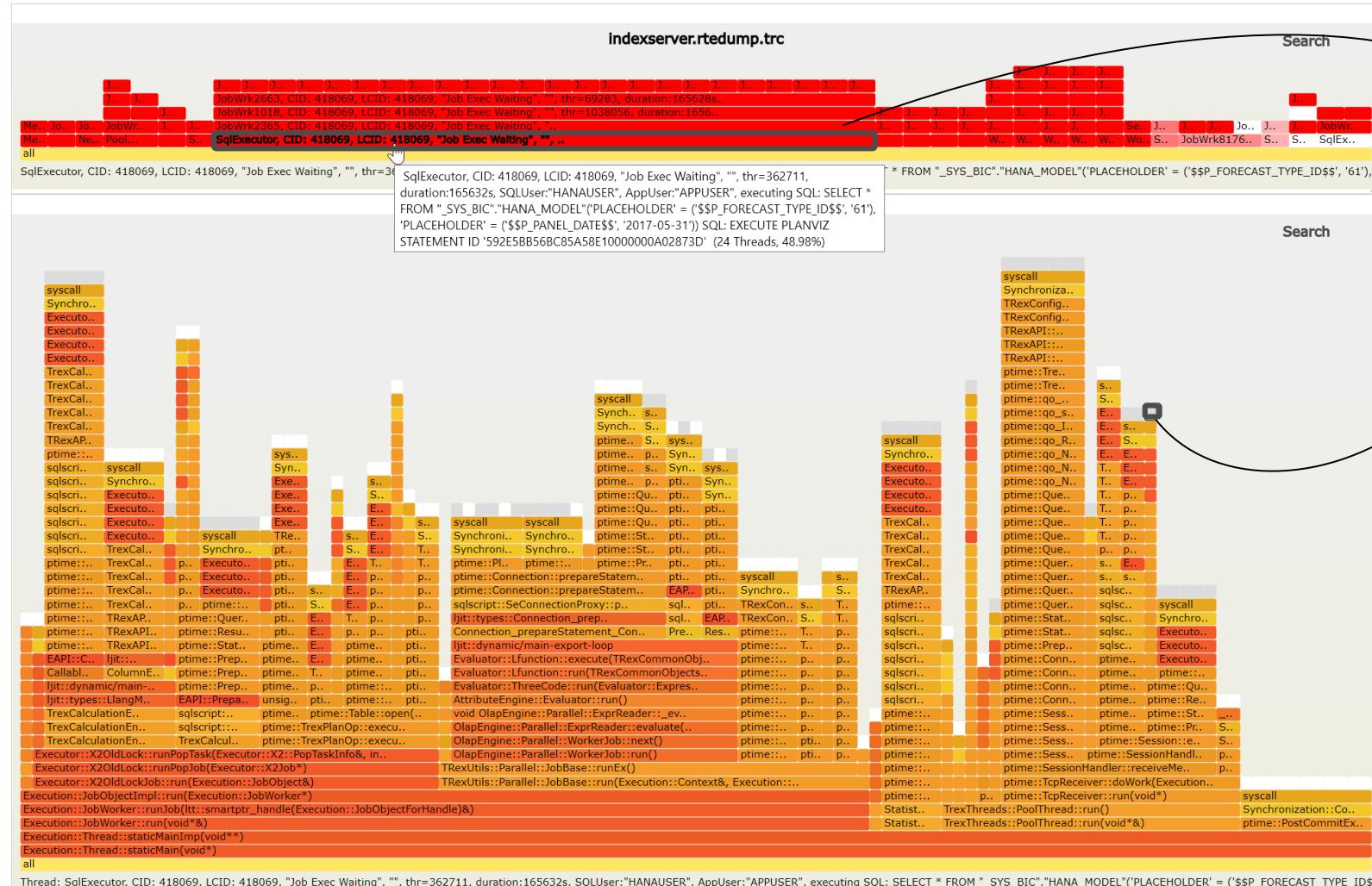


# HANA Thread Concurrency Flame Graph

A connection with CID 418069 is spawning lots of threads executing on stored procedure on a calculation view, the SqlExecutor on connection 418069 has a duration of 165632s already.



# HANA Mixed Concurrency Flame Graph

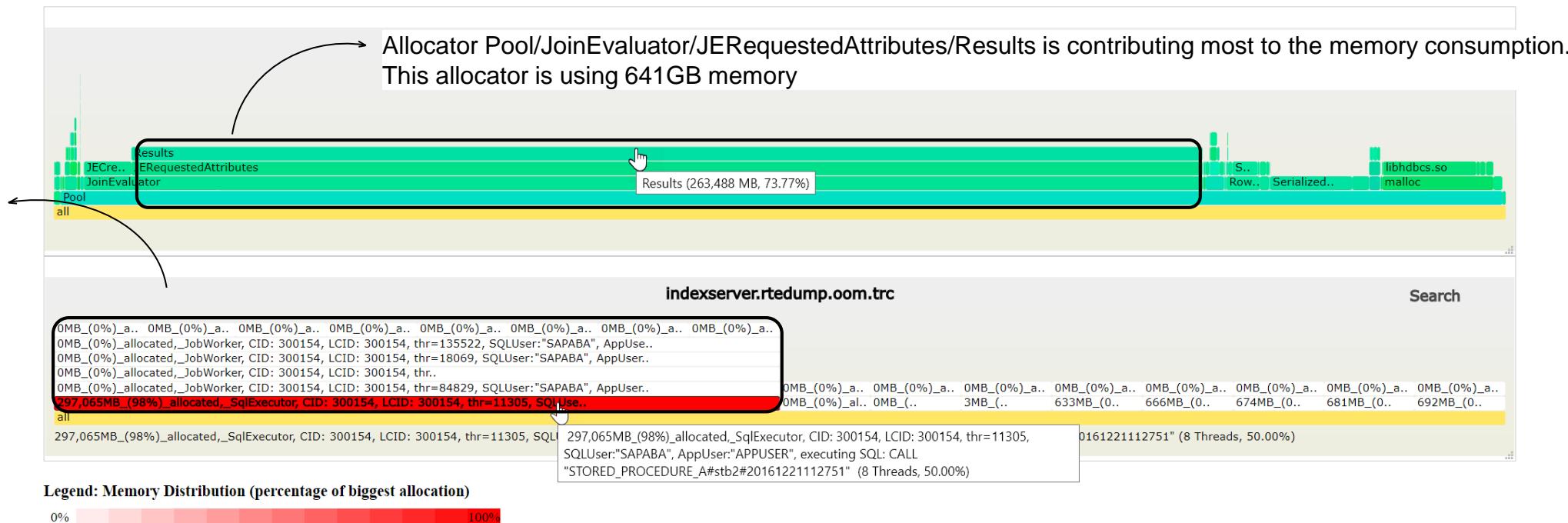


Mouse move over on concurrency stack: this directly shows the thread call stack from the same thread in stack flame graph. In case certain connection is stuck and related thread's call stack needs to be checked, this can be done within one step.

# HANA OOM Flame Graph

- HANA Memory Flame Graph visualizes the memory consumption from HANA allocators
- In case M\_ACTIVE\_STATEMENTS is available, memory consumption from the connection is visualized via red color on Concurrency Flame Graph. The **transparency of the red color** visualized on concurrency flame boxes stands for the memory distribution per thread
- Memory Diff Flame Graph is also available in SAP HANA dump analyzer

One big query on connection 300154 allocated 99% memory, i.e. around 297,065 MB. This is the query causing HANA indexserver OOM



# © 2019 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

See <http://global.sap.com/corporate-en/legal/copyright/index.epx> for additional trademark information and notices.