# KERNEL LEVEL THREADS IN NONSTOP

Lars Plum. NonStop Architect.

April 2024

# FORWARD-LOOKING STATEMENTS

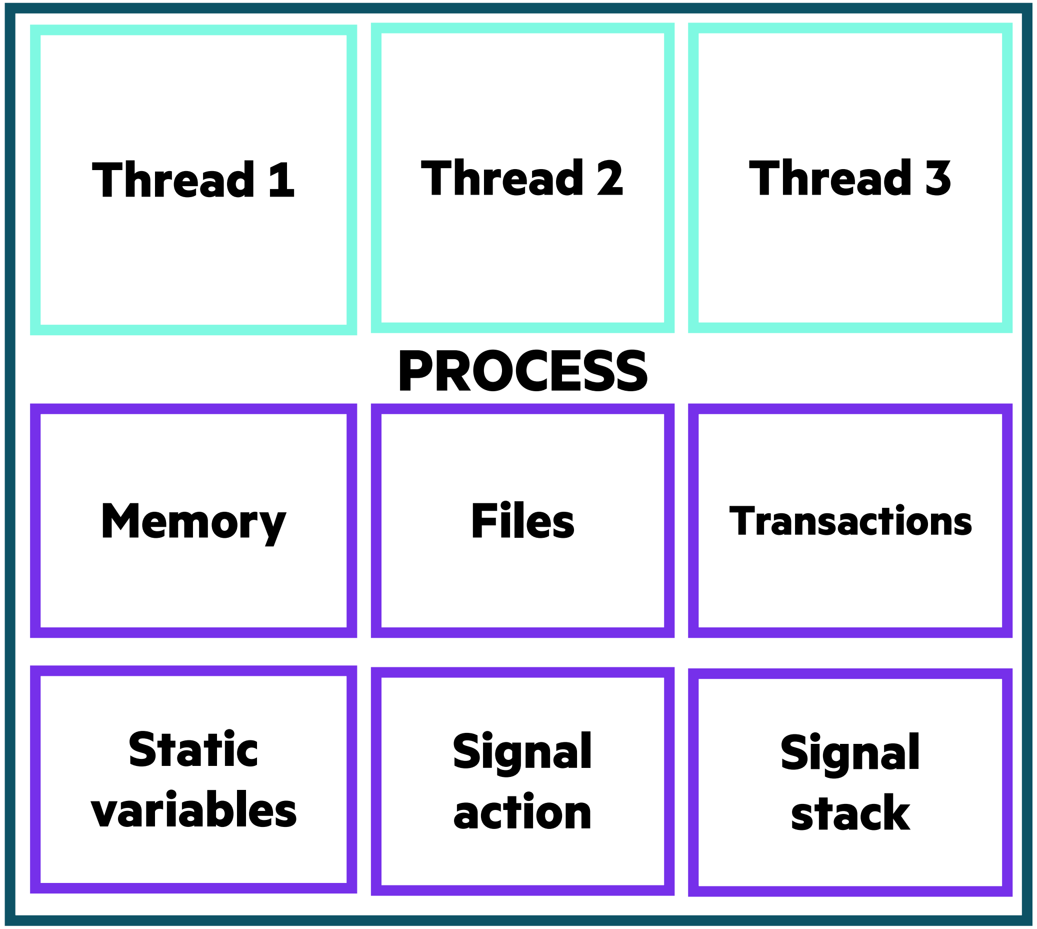
This document contains forward looking statements regarding future operations, product development, product capabilities and availability dates. This information is subject to substantial uncertainties and is subject to change at any time without prior notification. Statements contained in this document concerning these matters only reflect Hewlett Packard Enterprise's predictions and / or expectations as of the date of this document and actual results and future plans of Hewlett Packard Enterprise may differ significantly as a result of, among other things, changes in product strategy resulting from technological, internal corporate, market and other changes. This is not a commitment to deliver any material, code or functionality and should not be relied upon in making purchasing decisions.

# WHAT IS “KERNEL LEVEL THREADS”?

- Kernel Level Threads is an OS project underway in NonStop Engineering. This functionality is not available yet, but we want to make people aware of the project and what it can enable. KLT will be released in a future RVU and will be announced when it’s generally available.

# WHAT IS A THREAD?

- Single execution path in a process which shares process-level information (e.g. memory, files) with other threads
- Divides program flow into parallel tasks



Thread 1 | Thread 2 | Thread 3

PROCESS

Memory | Files | Transactions

Static variables | Signal action | Signal stack

# HISTORY OF NONSTOP THREAD MODELS

MULTITHREADED SERVERS

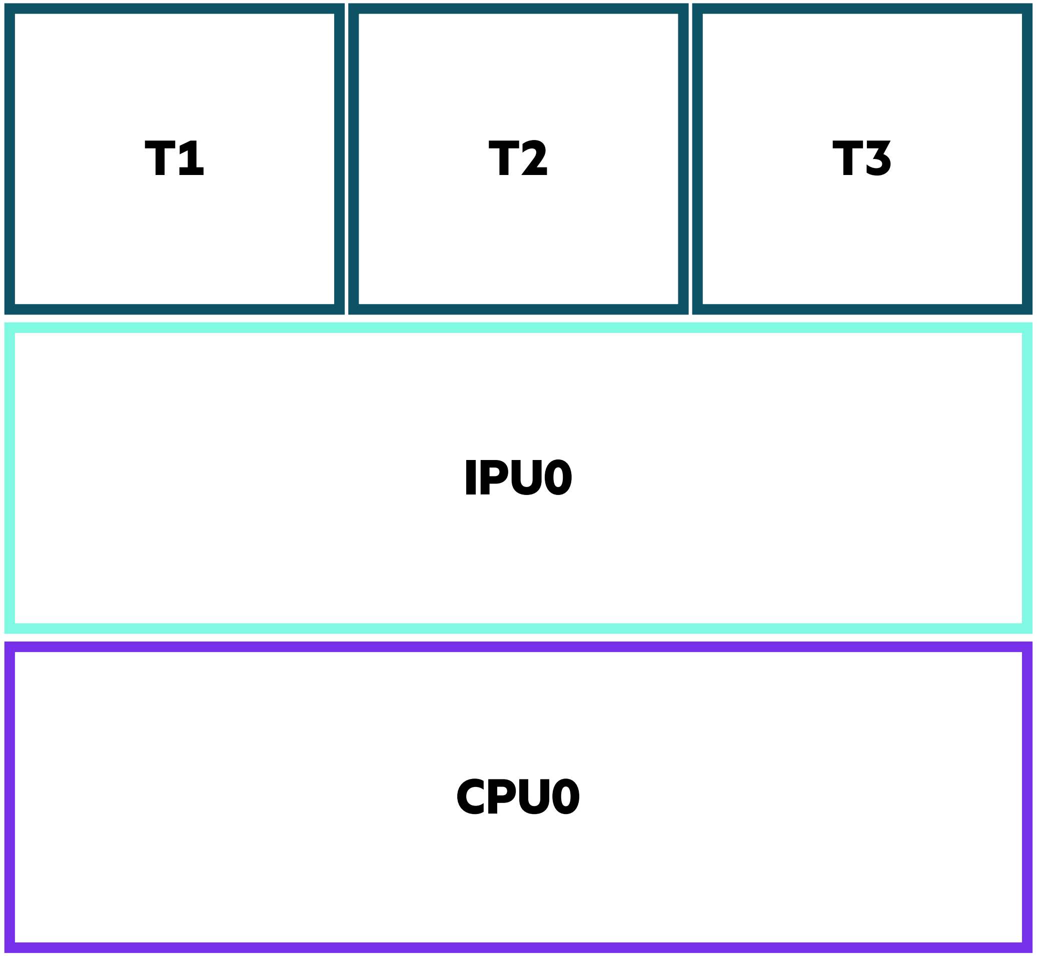
GTHREAD LIBRARY

STANDARD POSIX THREADS (SPT) LIBRARY

POSIX USER THREAD MODEL (PUT) LIBRARY

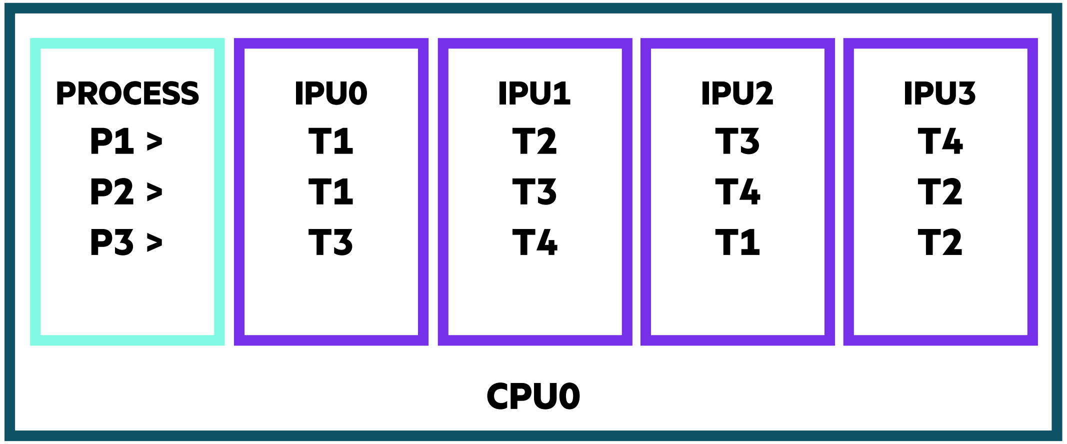
# USER LEVEL THREADS

- OSS user thread model executes in single process on single IPU (core) (Mx1)
- Thread library controls thread creation, scheduling and management
- NonStop OS not aware of threads
- Cooperative execution



# WHAT ARE KERNEL LEVEL THREADS?

- NonStop OS provides native kernel support for threads (1x1)
- Shared process-level information
- OS schedules threads
- Preemptive execution
- Thread library provides synchronization features
- POSIX (IEEE Std 1003.1 2004) compliant (Pthreads) API



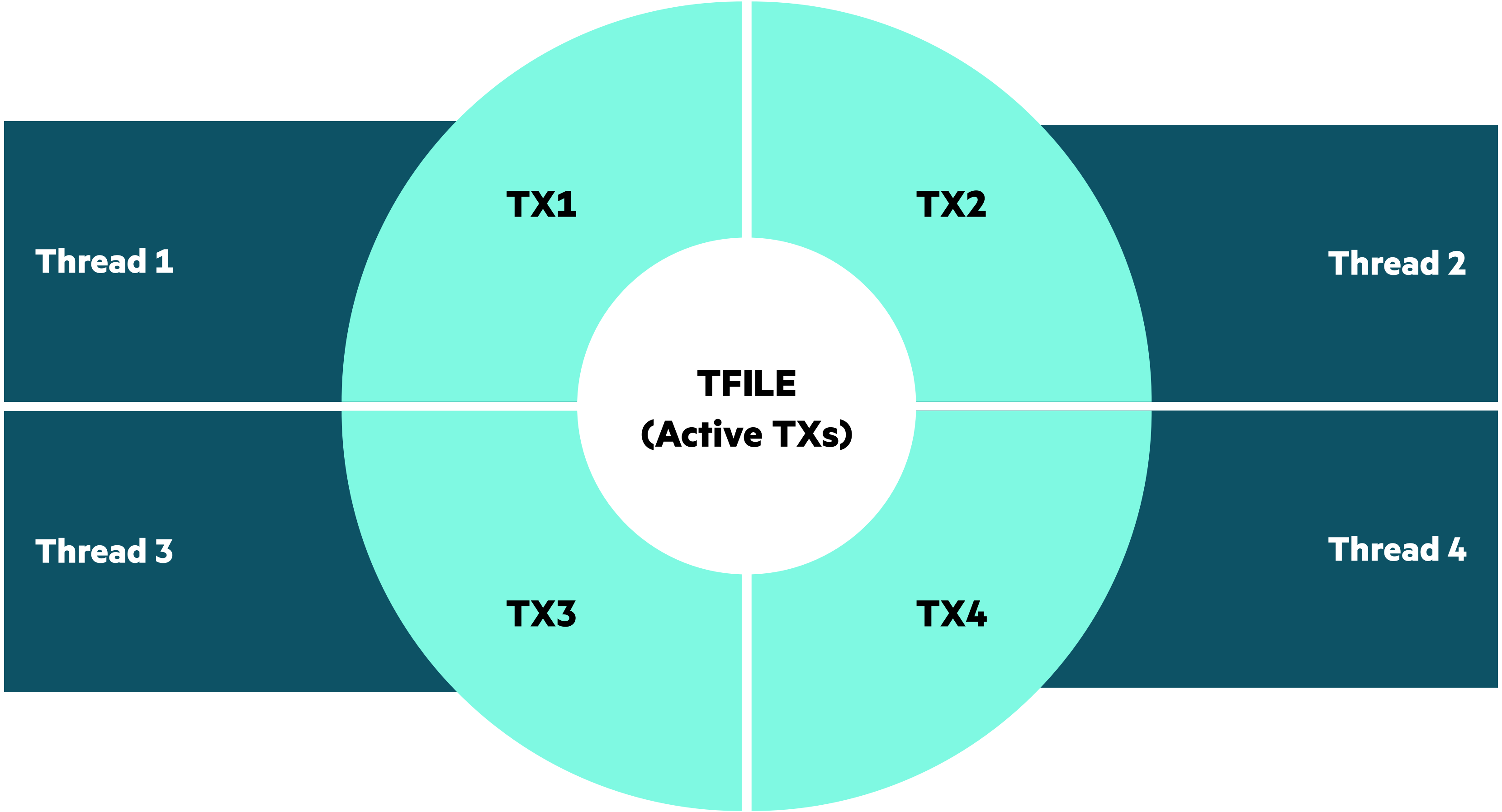
| PROCESS | IPU0 | IPU1 | IPU2 | IPU3 |
|---|---|---|---|---|
| P1 > | T1 | T2 | T3 | T4 |
| P2 > | T1 | T3 | T4 | T2 |
| P3 > | T3 | T4 | T1 | T2 |

# SHARED AND PER-THREAD INFORMATION

- Shared by all threads
  - Thread Group ID (main thread PID)
  - Memory (data, heap, segments)
  - File system (open files, file descriptors, file state, file locks)
  - Static variables
  - SQL cursors
  - Environment variables
  - Active transactions
  - Signal stack
  - Signal action

- Per-thread information
  - Thread ID
  - Program counter
  - Register set
  - Stack
  - errno
  - Current transaction
  - Thread local storage (TLS)
  - Priority (managed by NonStop OS)
  - Signal mask

# TMF FOR KLT PROCESSES



# WHAT DEFINES A KERNEL LEVEL THREADS APPLICATION?

- Compiled with –Wklt_model (_KLT_MODEL_ macro), linked with -lklt
- Process statically linked with KLT Model library
- Process identified as main thread
- Minor threads launched on the same CPU
- KLT threads are fundamentally OSS processes
- Execute side-by-side
- Process/thread views

# KLT PROGRAMMING CONSIDERATIONS

- Thread-safe and async-signal-safe functions
- Cancellation points
- Synchronizers
- Signal/assertion will lead to process termination
- Debugging: Native Inspect and NSDEE
- Use flat segments
- Use native APIs instead of PUT jacket APIs

# WHAT IS THE SCOPE OF KLT ON NONSTOP?

TNS/X (32/64-bit)

| | | | | |
|---|---|---|---|---|
| NonStop OS | OSS File System | Standard Millicode | Compilers and runtime | Native Inspect and NSDEE |
| KLT DLL | TMF | Security | Measure | CIP |
| SQL/MP | SQL/MX | Java/JVM | TS/MP | JI |
| JDBC | ODBC/MX | Apache | IMC | Pathsockets |

# WHAT DOES KLT MEAN FOR YOUR APPLICATIONS AND TOOLS?

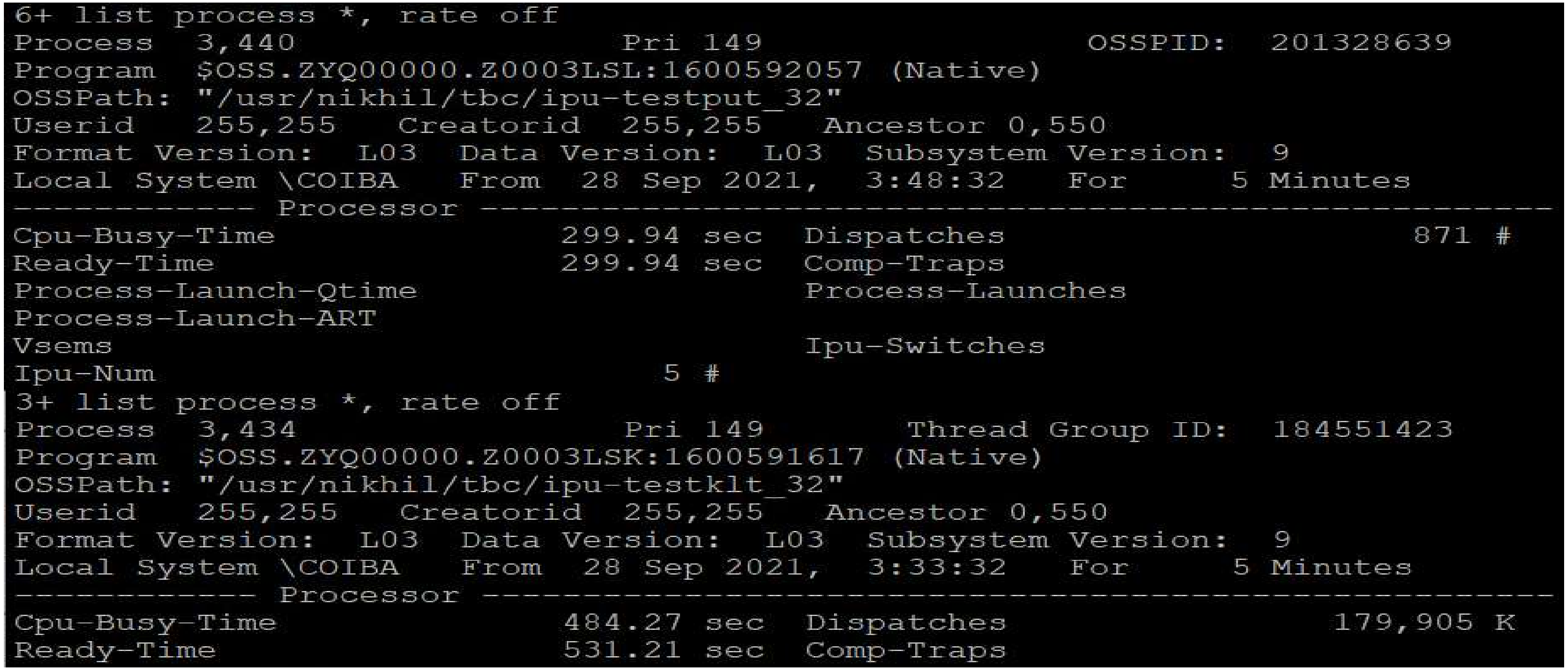
- Existing thread libraries remain available
- Porting application to NonStop OSS written to use threads: use KLT model
- Why change SPT or PUT-based application to KLT?
- Java virtual machine (JVM) and KLT
- Foundation for future parallel middleware and languages

# EXAMPLE: SEAMLESS MIGRATION FROM PUT TO KLT

- OpenSSL supported by HPE presently available with PUT threading model
- Easy to port OpenSSL from PUT Model to KLT Model
  - Replace _PUT_MODEL with _KLT_MODEL_ macro
  - Replace -lput with –lklt
- Result: OpenSSL tests with KLTDLL ran successfully

# EXAMPLE: MIGRATION FROM PUT TO KLT

- CPU-bound sample application with 5 minor threads using pthread mutex and condition variable built for PUT and KLT
- Measurement taken for 5 minutes; limited by CPU time
- Same application built for KLT: more CPU time available



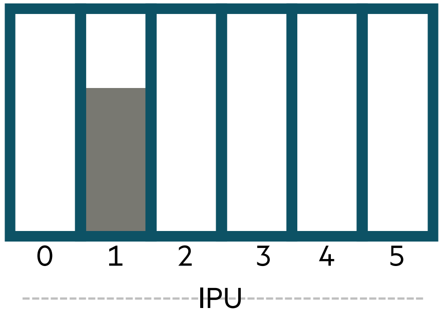
```
6+ list process *, rate off
Process  3,440               Pri 149                OSSPID:  201328639
Program  $OSS.ZYQ00000.Z0003LSL:1600592057 (Native)
OSSPath: "/usr/nikhil/tbc/ipu-testput_32"
Userid   255,255   Creatorid  255,255   Ancestor 0,550
Format Version:  L03  Data Version:  L03  Subsystem Version:  9
Local System \COIBA   From  28 Sep 2021,  3:48:33   For     5 Minutes
------------ Processor ------------------------------------------------------------
Cpu-Busy-Time                 299.94 sec  Dispatches                     871 #
Ready-Time                    299.94 sec  Comp-Traps
Process-Launch-Qtime                      Process-Launches
Process-Launch-ART
Vsems                                     Ipu-Switches
Ipu-Num                             5 #
3+ list process *, rate off
Process  3,434               Pri 149      Thread Group ID:  184551423
Program  $OSS.ZYQ00000.Z0003LSK:1600591617 (Native)
OSSPath: "/usr/nikhil/tbc/ipu-testklt_32"
Userid   255,255   Creatorid  255,255   Ancestor 0,550
Format Version:  L03  Data Version:  L03  Subsystem Version:  9
Local System \COIBA   From  28 Sep 2021,  3:33:32   For     5 Minutes
------------ Processor ------------------------------------------------------------
Cpu-Busy-Time                 484.27 sec  Dispatches                 179,905 K
Ready-Time                    531.21 sec  Comp-Traps
```

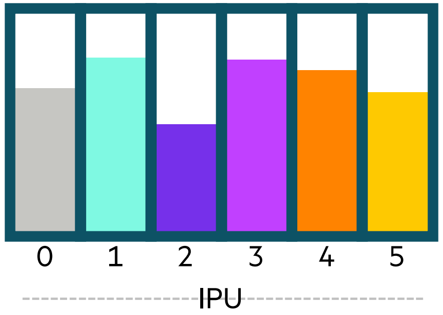
# EXAMPLE: MIGRATION FROM PUT TO KLT

## KLT threads make more efficient use of IPUs

| PIN | OSSPID | Program File | Thread Group ID | CPU Busy Time (sec) | Ready Time (sec) | IPU Num |
|---|---|---|---|---|---|---|
| 434 | 184551423 | OSSPath: /usr/nikhil/tbc/ipu-testklt_32 | 184551423 | 72.217199 | 80.449998 | 5 |
| 435 | 1962936303 | OSSPath: /usr/nikhil/tbc/ipu-testklt_32 | 184551423 | 122.655593 | 131.691824 | 0 |
| 436 | 2046822387 | OSSPath: /usr/nikhil/tbc/ipu-testklt_32 | 184551423 | 72.506165 | 78.964639 | 1 |
| 437 | 520095733 | OSSPath: /usr/nikhil/tbc/ipu-testklt_32 | 184551423 | 72.172332 | 79.277389 | 2 |
| 438 | 318769137 | OSSPath: /usr/nikhil/tbc/ipu-testklt_32 | 184551423 | 72.874681 | 81.065527 | 3 |
| 439 | 1694500848 | OSSPath: /usr/nikhil/tbc/ipu-testklt_32 | 184551423 | 71.843710 | 79.762221 | 4 |

- Exploit parallelism
- Use native APIs
- Use common code base for NSK and *NIX version of your application
- Improved responsiveness
- Easier to bring new solutions and products to NonStop
- Existing solutions run more efficiently
- Scale-up+out





# THANK YOU

lars.plum@hpe.com