# Preparing Linux Real-Time Kernel and Tuning Robotics Platform with a Modern ARM64 SoC

Krzysztof Kozlowski
Qualcomm Landing Team, Linaro
krzysztof.kozlowski@linaro.org

Linaro
Developer Services

# Introduction

- Krzysztof Kozlowski
- I work for Linaro in Qualcomm Landing Team / Linaro Developer Services
  - Upstreaming Qualcomm ARM/ARM64 SoCs
- I maintain few Linux kernel pieces (DT bindings, Samsung SoC, NFC and more)

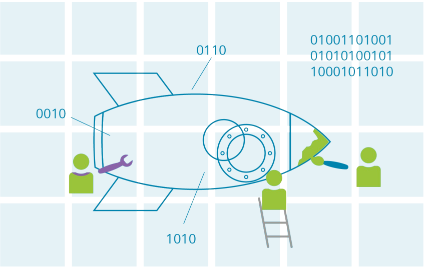
- What this talk will not be about
  - What is Real-Time and RTOS
  - PREEMPT_RT patchset

- What this talk will be about
  - Building and configuring a Real-Time Linux kernel
  - What to expect during testing and debugging
  - Basics of tuning the system for Real-Time
  - Evaluation and stress testing on embedded ARM64 robotics platform

# Linaro Developer Services

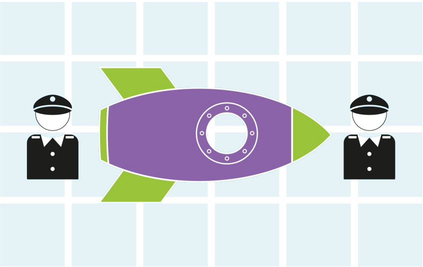
Linaro Developer Services helps companies build, deploy and maintain products on Arm

## Arm Software expertise



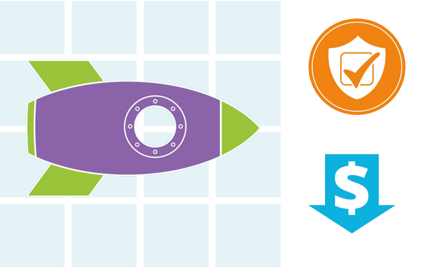

As part of Linaro, Developer Services has some of the world's **leading Arm Software experts**.

## Specialists in TEE on Arm



We specialize in security and Trusted Execution Environment (TEE) on Arm.

## Continuous Integration through LAVA



We offer continuous integration (CI) and automated validation through LAVA (Linaro's Automation & Validation Architecture)

## Build, Test and deploy faster

We support every aspect of product delivery, from building secure board support packages (BSPs), product validation and long-term maintenance.

For more information go to: [https://www.linaro.org/services/](https://www.linaro.org/services/)

Linaro
Developer Services

# Test platform - RB5

- The work I am describing was done on v6.1, but everything applies also to current v6.3
- [Qualcomm RB5 Robotics platform](https://developer.qualcomm.com/qualcomm-robotics-rb5-kit)
    - ARM64, 8-core SoC QRB5165 (SM8250)
    - 8 GB LPDDR 5 RAM
    - 128 GB UFS storage
    - WiFi, Bluetooth, and so on
    - Compliant with the 96Board

*Image source: https://developer.qualcomm.com/qualcomm-robotics-rb5-kit*

# First steps

- PREEMPT_RT is a patchset aiming to improve Real-Time aspects of the Linux kernel
- Most of it was already merged into mainline, but there are still some tasks to do
  - Still ~80 patches in PREEMPT_RT patchset
  - One can get the PREEMPT_RT from Git repo or as patchset for git-am
    - Remember to get Sebastian Andrzej Siewior’s key from kernel.org keyring
    - pgpkeys/keys/7B96E8162A8CF5D1.asc
- See [https://wiki.linuxfoundation.org/realtime/](https://wiki.linuxfoundation.org/realtime/) for details

# Kernel build configuration

- CONFIG_PREEMPT_RT=y
  - Fully Preemptible Kernel (Real-Time)
  - $ cat /sys/kernel/realtime
- CONFIG_NO_HZ_FULL=y
  - Which will behave as NO_HZ_IDLE by default
- CONFIG_HZ_1000=y
- CONFIG_CPUSETS=y
  - For isolating CPUs for Real-Time workloads
- CONFIG_BLK_CGROUP_IOLATENCY=y

Most likely you will also want for evaluation and debugging latency issues:

- CONFIG_LATENCYTOP=y
- CONFIG_SCHED_TRACER=y
- CONFIG_TIMERLAT_TRACER=y
- CONFIG_HWLAT_TRACER=y

# I boot therefore I am (correct)

- That was easy, right? Kernel boots so job is done!
- Nope
- PREEMPT_RT will likely exercise a bit different driver paths in regard of concurrency
- Thus new race conditions are possible due to:
  - Missing synchronization
  - Different code-flow, e.g. order of driver callbacks between devices
  - Issues might not be visible during most of system boots
- Build a test kernel with:
  - CONFIG_KASAN=y
  - CONFIG_DEBUG_SHIRQ=y
  - CONFIG_SOFTLOCKUP_DETECTOR=y
  - CONFIG_DETECT_HUNG_TASK=y
  - CONFIG_WQ_WATCHDOG=y
  - CONFIG_DEBUG_PREEMPT=y
  - CONFIG_DEBUG_IRQFLAGS=y

# Checking locking correctness

- PREEMPT_RT change semantics of few kernel locks
- Build a test kernel with LOCKDEP:
  - CONFIG_PROVE_LOCKING=y
    - Lock debugging: prove locking correctness
  - CONFIG_PROVE_RAW_LOCK_NESTING=y
    - Enable raw_spinlock - spinlock nesting checks
  - CONFIG_DEBUG_ATOMIC_SLEEP=y
    - Sleep inside atomic section checking

```
BUG: sleeping function called from invalid context at kernel/locking/spinlock_rt.c:46
in_atomic(): 0, irqs_disabled(): 128, non_block: 0, pid: 298, name: systemd-udevd
preempt_count: 0, expected: 0
```

```
BUG: sleeping function called from invalid context at kernel/locking/spinlock_rt.c:46
in_atomic(): 1, irqs_disabled(): 0, non_block: 0, pid: 291, name: systemd-udevd
preempt_count: 1, expected: 0
```

# Checking locking correctness

- This is quite expected problem and it is a direct result of PREEMPT_RT: spinlock and few more locks are now sleeping primitives
- For example the spinlock should not be used within atomic sections:
    - Disabled interrupts
    - Disabled preemption
    - Instead one could use raw_spinlock
    - It is even trickier with local_lock(), but that's not a typical case, so out of scope

# What can go wrong - disabled IRQs

- Look for:
  - `BUG: sleeping function called from invalid context at kernel/locking/spinlock_rt.c:46 in_atomic(): 0, irqs_disabled(): 128, non_block: 0, pid: 298, name: systemd-udevd preempt_count: 0, expected: 0`
- Non-PREEMPT_RT correct but PREEMPT_RT incorrect:

```
local_irq_disable();
...
  spin_lock_irqsave(&l, flags);
  ...
  spin_unlock_irqrestore(&l, flags);
...
local_irq_enable();
```

Both correct (example approach):

```
local_irq_disable();
...
  raw_spin_lock_irqsave(&l, flags);
  ...
  raw_spin_unlock_irqrestore(&l, flags);
...
local_irq_enable();
```

Linaro
Developer Services

# What can go wrong - disabled preemption

- Look for:
  - `BUG: sleeping function called from invalid context at kernel/locking/spinlock_rt.c:46 in_atomic(): 1, irqs_disabled(): 0, non_block: 0, pid: 291, name: systemd-udevd preempt_count: 1, expected: 0`
- Non-PREEMPT_RT correct but PREEMPT_RT incorrect:

```
preempt_disable();
...
  spin_lock_irqsave(&l, flags);
  ...
  spin_ublock_irqrestore(&l, flags);
...
preempt_enable();
```

Both correct:

```
preempt_disable();
...
  raw_spin_lock_irqsave(&l, flags);
  ...
  raw_spin_unlock_irqrestore(&l, flags);
...
preempt_enable();
```

- These are simple cases. Much more complex is runtime PM which uses spinlock. Most of the drivers using pm_runtime_get_sync() is not expecting it to sleep.

Linaro
Developer Services

# What can go wrong - memory allocation

- Memory allocator is now fully preemptible, also for GFP_ATOMIC
- Look for:
  - `BUG: sleeping function called from invalid context`
- Non-PREEMPT_RT correct but PREEMPT_RT incorrect:

```
raw_spin_lock(&l);
p = kmalloc(sizeof(*p), GFP_ATOMIC);
...
raw_spin_unlock(&l);
```

Both correct:

```
spin_lock(&l);
p = kmalloc(sizeof(*p), GFP_ATOMIC);
...
spin_unlock(&l);
```

- ... or move the allocation out of critical section

Linaro Developer Services

# System Evaluation and Tuning

# Evaluation of the system

- $ cat /sys/kernel/realtime returns 1, so are we done?
- Let's check how the system behaves
- Real-Time use case requires application to respond to event within some deadline
- Time between event and actual response => latency
- For your workload, real or simulated, you might need to know what is the **maximum** experienced latency
- Why maximum matters?
    - Consider time between hitting brakes pedal in the car and reaction of the brakes
    - Or between critical pressure in some pipe in industrial setup and system reaction
    - It does not matter that on average brakes or system reacts within microseconds
    - It matters that it never reacts too late - over some threshold, defined by your system requirements

# Evaluation of the system - tools

- The typical tools for this are cyclictest and stress-ng
  - cyclictest - application measuring latencies in real-time systems caused by the hardware, the firmware, and the operating system.
  - stress-ng - stressor of various parts of system, includes also cyclic functionality
  - rtla timerlat - cyclictest on steroids, using kernel tracers
- E.g. make your RT CPUs busy at 60% and measure latencies with cyclictest

```
cgexec -g cpuset:rt stress-ng --cpu 6 --cpu-load 60
cgexec -g cpuset:rt cyclictest -m --affinity 7 --threads 1 -p 95 -h 150 \
    --mainaffinity=2 --policy fifo
```

# Evaluation of the system

- Qualcomm RB5 Robotics platform example latencies
    - ARM64, 8-core SoC QRB5165 (SM8250)
    - Three clusters
        - 4x Cortex-A55
        - 3x Cortex-A77
        - 1x Cortex-A77 (Prime)
- Kernels compared:
    - Vanilla: v6.1.7 stable kernel
    - RT: v6.1.7-rt5, Qualcomm Landing Team kernel
        - v6.1 kernel with PREEMPT_RT patches
        - With some hardware enablement patches being upstreamed
        - With Real-Time fixes developed during entire process
            - Already upstreamed or in process
            - Issue found using tools described at the end of the talk
        - Should be without differences against current mainline (-PREEMPT_RT)

# Measurements - try 1 - idle

- No load, idle system, cyclictest on CPU0-7

| | Min latency [us] | | | Average lat. [us] | | | Max latency [us] | | |
|---|---|---|---|---|---|---|---|---|---|
| Cluster | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 |
| CPU | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 |
| Van-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 18, 17, 15, 18 | 6, 6, 5 | 5 | 729, 861, 167, 353 | 92, 100, 97 | 94 |
| RT-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 20, 20, 17, 18 | 6, 7, 7 | 6 | 164, 169, 230, 612 | 51, 317, 67 | 73 |

- On average system behaves nice…
- But maximum latencies are in both cases very high

# Measurements - try 1 - busy 60%

- System busy with ~60% load

| | Min latency [us] | | | Average lat. [us] | | | Max latency [us] | | |
|---|---|---|---|---|---|---|---|---|---|
| Cluster | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 |
| CPU | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 |
| Van-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 16, 16, 16, 18 | 14, 4, 6 | 4 | 307, 343, 558, 210 | 21, 98, 60 | 28 |
| RT-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 21, 20, 17, 19 | 8, 6, 6 | 7 | 212, 547, 921, 653 | 61, 69, 72 | 43 |

- Similarly to idle case - maximum latencies are in both cases very high
- The results are not good - something is missing

# Tuning the system

- Kernel with PREEMPT_RT is not enough
- Several regular kernel activities (housekeeping tasks) can interrupt Real-Time application adding unexpected latencies
    - RCU callbacks
    - Periodic timer ticks
    - Interrupts
    - Workqueues
- Also Real-Time application should not fight with other processes for CPU time
- Usually some CPUs are assigned to housekeeping tasks and some to Real-Time
    - E.g. CPU 0-1 for housekeeping, rest (CPU 2-7) for RT

# Tuning the system - command line

- Offload RCU callbacks from RT CPUs:
  - rcu_nocbs=2-7 rcu_nocb_poll
- Default IRQ affinity to housekeeping CPUs:
  - irqaffinity=0-1
- Mitigate for xtime_lock contention:
  - skew_tick=1
- Disable lockup detectors:
  - nosoftlockup nowatchdog
- For specific workloads (one thread per CPU core) disable tick on RT CPUs:
  - nohz_full=2-7
  - Long latency penalty during context switches, thus it must match specific workload

# Tuning the system - runtime

- Keep IRQs on housekeeping CPUs:
  - systemctl disable irqbalance
  - Or use IRQBALANCE_BANNED_CPUS so they will be balanced between housekeeping CPUs (e.g. to still distribute busy UFS and USB/Ethernet interrupts among two CPUs)
- Move workqueues to housekeeping CPUs:
  - echo 03 > /sys/devices/virtual/workqueue/blkcg_punt_bio/cpumask
    echo 03 > /sys/devices/virtual/workqueue/scsi_tmf_0/cpumask
    echo 03 > /sys/devices/virtual/workqueue/writeback/cpumask
  - And possibly other…
- Disable CPU frequency scaling
  - cpupower frequency-set -g performance
- Disable deeper CPU idle states
  - cpupower idle-set -d 1
- Allowing RT application up to 100% of CPU time (optional)
  - /proc/sys/kernel/sched_rt_runtime_us
  - Other tasks can starve

# Measurements - try 2 - idle - basic tuning

- No load, idle system, cyclictest on CPU0-7

| | Min latency [us] | | | Average lat. [us] | | | Max latency [us] | | |
|---|---|---|---|---|---|---|---|---|---|
| Cluster | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 |
| CPU | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 |
| Van-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 18, 17, 15, 18 | 6, 6, 5 | 5 | 729, 861, 167, 353 | 92, 100, 97 | 94 |
| RT-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 20, 20, 17, 18 | 6, 7, 7 | 6 | 164, 169, 230, 612 | 51, 317, 67 | 73 |
| RT-#2 | 5, 5, 4, 5 | 1, 1, 2 | 1 | 6, 6, 5, 5 | 2, 2, 2 | 2 | **99, 80, 21, 44** | **86, 33, 15** | **84** |

- A bit better, specially for slower cluster, but still too high

# Tuning the system - cpusets

- Older kernels used “isolcpus” command line argument
- Since some time, cgroups/cpusets should be used
  - For instructions see: [https://docs.kernel.org/admin-guide/cgroup-v2.html#cpuset](https://docs.kernel.org/admin-guide/cgroup-v2.html#cpuset)
- All further tests will exclude housekeeping/bulk CPUs from measurement

```
cd /sys/fs/cgroup/
echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control

# Create housekeeping cpuset for CPU 0-1:
mkdir /sys/fs/cgroup/bulk
echo "+cpuset" >> bulk/cgroup.subtree_control
echo 0-1 >> bulk/cpuset.cpus
ps -eLo lwp | while read thread; do echo $thread > bulk/cgroup.procs ; done
```

# Tuning the system - cpusets (continued)

- Now the Real-Time group:

```
mkdir /sys/fs/cgroup/rt
# Consider "isolated" partition, but then tasks won't be balanced
# echo isolated > rt/cpuset.cpus.partition
echo root > rt/cpuset.cpus.partition
echo "+cpuset" >> rt/cgroup.subtree_control
echo "2-7" >> rt/cpuset.cpus

# Test if group has correct (not invalid) configuration
cat rt/cpuset.cpus.partition
-> expected: root

# Run your app with:
cgexec -g cpuset:rt ..........
```

# Measurements - try 3 - idle - full tuning

- No load, idle system, cyclictest on CPU2-7

| | Min latency [us] | | | Average lat. [us] | | | Max latency [us] | | |
|---|---|---|---|---|---|---|---|---|---|
| Cluster | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 |
| CPU | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 |
| Van-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 18, 17, 15, 18 | 6, 6, 5 | 5 | 729, 861, 167, 353 | 92, 100, 97 | 94 |
| RT-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 20, 20, 17, 18 | 6, 7, 7 | 6 | 164, 169, 230, 612 | 51, 317, 67 | 73 |
| RT-#2 | 5, 5, 4, 5 | 1, 1, 2 | 1 | 6, 6, 5, 5 | 2, 2, 2 | 2 | 99, 80, 21, 44 | 86, 33, 15 | 84 |
| Van-#3 | 3, 5 | 1, 1, 1 | 1 | 6, 5 | 2, 2, 2 | 2 | **13, 11** | **5, 5, 4** | **4** |
| RT-#3 | 4, 5 | 1, 2, 2 | 1 | 6, 6 | 2, 2, 2 | 2 | **19, 11** | **3, 5, 5** | **4** |

# Measurements - try 3 - busy 60% - full tuning

- System busy with ~60% load

| | Min latency [us] | | | Average lat. [us] | | | Max latency [us] | | |
|---|---|---|---|---|---|---|---|---|---|
| Cluster | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 |
| CPU | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 |
| Van-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 16, 16, 16, 18 | 14, 4, 6 | 4 | 307, 343, 558, 210 | 21, 98, 60 | 28 |
| RT-#1 | 5, 5, 5, 5 | 2, 2, 2 | 2 | 21, 20, 17, 19 | 8, 6, 6 | 7 | 212, 547, 921, 653 | 61, 69, 72 | 43 |
| Van-#3 | 4, 4 | 2, 2, 2 | 2 | 7, 7 | 3, 5, 5 | 5 | **19, 18** | **15, 14, 14** | **38** |
| RT-#3 | 5, 5 | 2, 2, 2 | 1 | 6, 6 | 2, 2, 2 | 2 | **14, 10** | **8, 4, 4** | **4** |

# Measurements - try 3 - busy 100% - full tuning

- System busy with ~100% load

| | Min latency [us] | | | Average lat. [us] | | | Max latency [us] | | |
|---|---|---|---|---|---|---|---|---|---|
| Cluster | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 | 4xA55 | 3xA77 | A77 |
| CPU | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 | 0, 1, 2, 3 | 5, 6, 7 | 7 |
| Van-#3 | 4, 4 | 3, 3, 3 | 2 | 5, 6 | 4, 4, 4 | 4 | **36, 18** | **9, 10, 11** | **36** |
| RT-#3 | 5, 5 | 3, 3, 3 | 2 | 6, 8 | 4, 5, 5 | 4 | **22, 18** | **7, 15, 10** | **8** |

# Results

- Heterogeneous systems will have different latency results on different cores
- With a properly tuned system, is the PREEMPT_RT even needed?
- The mainline kernel almost does not differ from PREEMPT_RT in results
  - The mainline kernel already introduces Real-Time scheduler: SCHED_FIFO and SCHED_RR
- Let’s just use mainline and ditch PREEMPT_RT?
- No, we can’t
  - Well, this was just a test executed for some minutes, not a real product running for days
  - Just because test does not hit some case with high latency, it’s not a proof it is not there waiting to bit you
  - Mainline does not guarantee these latencies
  - It does not come with mechanisms solving for example priority inversion problem in scheduling

# Useful tools

# Latency spikes - hwlatdetect

- What if the average latency is low, but the maximum is high?
- Check latencies introduced by hardware or firmware with **hwlatdetect**
    - On RT/isolated CPUs

```
hwlatdetect  --duration=600s --cpu-list=2-7 --threshold=5
   parameters:
        CPU list:          2-7
        Latency threshold: 5us
        Sample window:     1000000us
        Sample width:      500000us
     Non-sampling period:  500000us
        Output File:       None
Max Latency: Below threshold
Samples recorded: 0
Samples exceeding threshold: 0
```

Linaro
Developer Services

# Latency spikes - tracing

- Cyclictest can help trace the cause of the latency
  - First set up your tracing
  - Then cyclictest with “-b XX --tracemark” argument

```
cd /sys/kernel/tracing/
echo function > current_tracer
echo 1 > tracing_on
cgexec -g cpuset:rt cyclictest -m --affinity 7 --threads 1 -p 95 -h 150 \
    --mainaffinity=2 --policy fifo -b 25 --tracemark

less trace # look for tracing_mark_write
```

# Latency spikes - rtla osnoise

- Look for OS noise with rtla
  - apt-get install rtla
  - Or build it from linux/tools/tracing/rtla
- rtla osnoise gives answers about noise caused by the system
- How much of time is taken from RT application, e.g. by IRQs or preemption?
- Look for noise on isolated CPUs
- Refer to RTLA: Real-time Linux Analysis Toolset - Daniel Bristot De Oliveira, Red Hat for tutorial/howto (or Daniel's session also today)

```
$ rtla osnoise top --stop 10 --threshold 5 --cpus 2-7 --trace
CPU Period       Runtime        Noise  % CPU Aval   Max Noise   Max Single
  2 #4           4000000         6664    99.83340        2075           67
  3 #4           4000000          472    99.98820         263           19
  4 #4           4000000            0   100.00000           0            0
  5 #4           4000000         6542    99.83645        2170          147
  6 #4           4000000          155    99.99612          54           54
  7 #4           4000000           15    99.99962          15           15
```

# Latency spikes - rtla timerlat

- rtla timerlat is a cyclictest on steroids
  - Refer to RTLA: Real-time Linux Analysis Toolset or Daniel's session also today

```
rtla timerlat top --cpus 2-7 --auto 25
## CPU 2 hit stop tracing, analyzing it ##
  IRQ handler delay:                                  1.23 us (4.85 %)
  IRQ latency:                                        5.24 us
  Timerlat IRQ duration:                             10.47 us (41.31 %)
  Blocking thread:                                    6.62 us (26.10 %)
                   swapper/2:0                             6.62 us
    Blocking thread stack trace
           -> timerlat_irq
           -> __hrtimer_run_queues
           -> hrtimer_interrupt
           -> arch_timer_handler_virt
           -> handle_percpu_devid_irq
```

Linaro
Developer Services

# Resources and references

- cylictest
- Optimizing RHEL 8 for Real Time for low latency operation
- RTLA: Real-time Linux Analysis Toolset - Daniel Bristot De Oliveira, Red Hat

# Introducing Linaro

**Linaro collaborates with businesses and open source communities to:**

- Consolidate the Arm code base & develop common, low-level functionality
- Create open source reference implementations & standards
- Upstream products and platforms on Arm

**Why do we do this?**

- To make it easier for businesses to build and deploy high quality and secure Arm-based products
- To make it easier for engineers to develop on Arm

**Two ways to collaborate with Linaro:**

1. Join as a member and work with Linaro and collaborate with other industry leaders
2. Work with Linaro Developer Services on a one-to-one basis on a project

For more information go to: [www.linaro.org](www.linaro.org)

# Linaro membership collaboration

HISILICON

life.augmented

SAMSUNG

arm

FUJITSU

oppo

Google

Microsoft

Red Hat

Qualcomm
Innovation Center

AMD
XILINX

socionext™

NXP

# Thank you

Linaro
Developer Services