ENEA

Real-time Linux – the LWRT Concept

LWRT - a “Light-weight RunTime” environment for multicore devices that improves Linux real-time performance and provides better real-time behavioral characteristics for “SOME” networking and communications applications

Michael Christofferson

Director Marketing, CTO Office

Korea Linux Forum 2012

Enea - Powering Communications

- Increasing data traffic in communication devices require new and innovative software solutions to handle bandwidth, performance and power requirements.
- Enea software is heavily used in wireless Infrastructure (Macro, small cell), gateway, terminal, military, auto, etc.
- More than 250M of the 325M LTE population coverage is powered by Enea Solutions
- Enea Solutions run in more than 50% of the world's 8.2M radio base stations.
- Enea has recently released its first commercial Linux distribution, built by Yocto, and specially tailored for networking and communications
- Global presence, global development, and headquartered in Stockholm, Sweden

FOUNDED
1968

REVENUE
67M
USD

TEN OFFICES IN NORTH AMERICA, EUROPE AND ASIA

NO. OF EMPLOYEES
426

Numbers for 2011

What Does “Real-time” Mean?

- Real-time systems
  - Have “operational deadlines from event to system response”
  - Must guarantee the response to external events within strict time constraints
- Non-real-time systems
  - Cannot guarantee response time in any situation
  - Are often optimized for best-effort, high throughput performance
- “Real-time response” means deterministic response
  - Can mean seconds, milliseconds, microseconds.
  - I.e. not necessarily short times, but usually this is the case
- Real-time system classifications:
  - **Hard**: missing a deadline means total system failure
  - **Firm**: infrequent misses are tolerable, but result is useless. QoS degrades quickly
  - **Soft**: infrequent misses are tolerable, increased frequency degrades QoS more slowly

=> Real-time IS NOT the same as high-performance computing!

Examples of real-time systems

- **Hard real-time applications:**
  - Automotive: anti-lock brakes, car engine control
  - Medical: heart pacemakers
  - Industrial: process controllers, robot control
- **Firm real-time applications:**
  - 3G/4G baseband processing/signaling in base stations and radio network controllers
  - 3G/4G baseband processing/signaling in wireless modems (phones, tablets)
  - Many other examples in the networking space – RRU, optical transport, backhaul, too numerous to list
- **Soft real-time applications:**
  - IP network control signaling, network servers
  - Live audio-video systems

Enea LWRT is about Firm Real-Time Applications

So How about Firm Real-time and Linux?

ENEA

Three ways to address real-time in Linux:

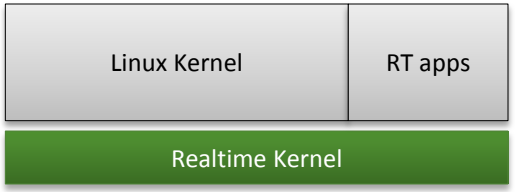
The PREEMPT_RT patch

Rework the internals of Linux:

- PREEMPT_RT, a patch to the mainstream Linux
- Taking 3.0.27 as an example, PREEMPT_RT patches 500+ locations in the kernel, with 11,500+ new lines of code in total.
- Decreases throughput, but offers full POSIX
- Suitable for low to moderate real-time requirements

"Thin-kernel" virtualization

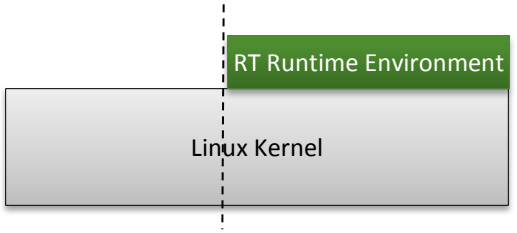
Add a thin real-time kernel underneath Linux:



- Virtualizes Linux
- Examples includes hypervisor, Xenomai, RTLinux etc
- Provides a highly deterministic RTOS environment for RT apps
- Cannot completely utilize the Linux eco-system (e.g. tools) in the realtime domain.
- Suitable for very high real-time requirements, inherited from classic RTOS domains

Vertical Partitioning + User mode RT Runtime

Vertically partition Linux in two domains:

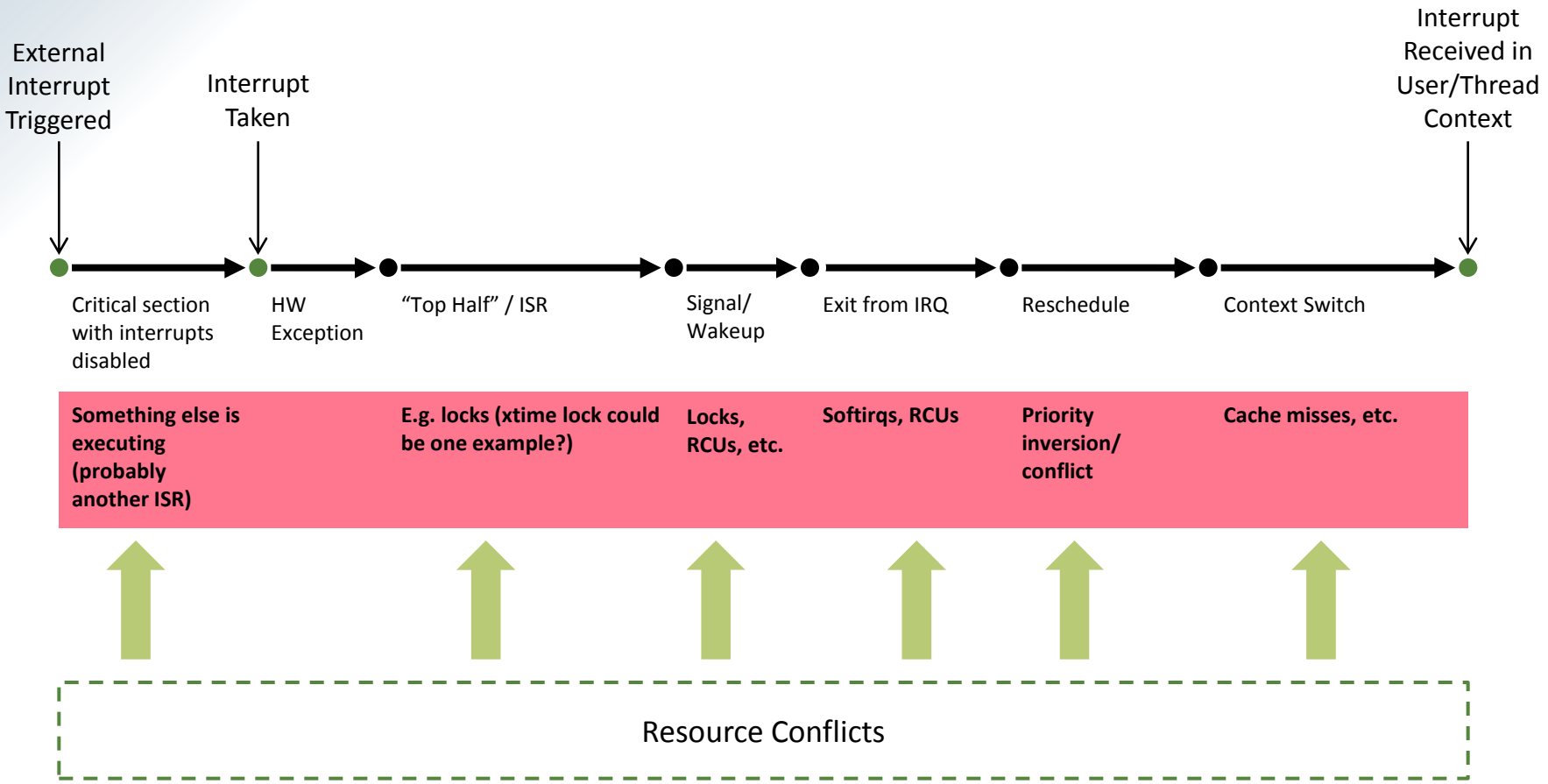


- Only on multicore
- "Shield" a set of cores from regular Linux OS scheduling
- A user mode light-weight runtime environment for real-time applications that avoid calling the kernel and thus resource conflicts
- An "ALL LINUX" solution, but with additional APIs

LWRT!!

ENEA

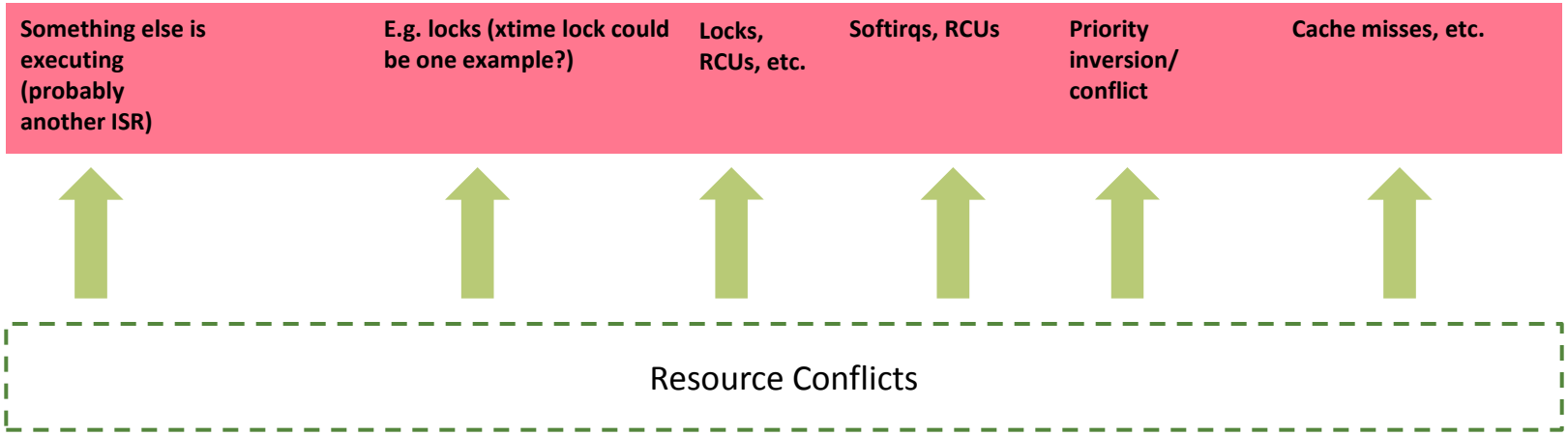
Linux Interrupt Processing
Delays from external event to response



How can we tackle resource conflicts?

Try to mitigate the effect of resource conflicts
(e.g. make ISRs preemptable, add priority inheritance, etc.).

Try to avoid resource conflicts
(e.g. partition the system, avoid shared resources in kernel etc.).



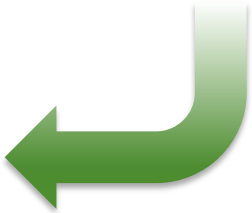
Resource conflicts “2”

Try to mitigate the effect of resource conflicts
(e.g. make ISRs preemptable, add priority inheritance, etc.).

Try to avoid resource conflicts
(e.g. partition the system, avoid shared resources in kernel etc.).

PREEMPT_RT

Vertical Partitioning



Vertical Partitioning + PREEMT_RT

The CONFIG_PREEMPT_RT patch set

- Developed by a team led by Ingo Molnar
  - Primarily before multicore evolution; unicore optimized technology
- Can be seen as a kind of added virtualization on HAL level
- Replaces most kernel spinlocks with mutexes with priority inheritance
- Move all interrupt handling to kernel threads
  - This means many drivers must be modified
- Taking 3.0.27 as an example, PREEMPT_RT patches 500+ locations in the kernel, with 11,500+ new lines of code in total.
- Good when the worst case response latency requirement is low to moderate, around 50-100 us.

How to avoid kernel resource conflicts?

Multiple interrupt sources competing for privilege

(Adding delay to the interrupt path)

Partition the system
– handle interrupts with core affinity

Multiple Posix threads/processes competing for CPU time

(Might be preempted in a non-deterministic way)

Partition the system
– only one thread per core

Multiple cores competing for memory

(For example, what if we hit a mutex or a RCU write-side lock?)

Avoid mechanisms that rely on shared memory (e.g. the Linux scheduler)

The Vertical Partitioning Concept

- Partitioning of the system into separate realtime critical (shielded cores) an non-critical domains.
- It is often the Linux kernel itself that introduces realtime problems.
- Realtime partition does not need full POSIX/Linux API
- A combination of partitioning, combined with a user-mode environment that allow us to avoid using the kernel can improve performance and realtime characteristics compared to a standard Linux.

"Improve performance and realtime characteristics under Linux by partitioning the system into logical domains, and by avoiding usage of the Linux kernel and its resources more than necessary"

The Vertical Partitioning Concept (2)

- Configure processes and interrupts to run with core affinity
- Make minor modifications to the kernel to avoid running kernel threads/timers on real-time cores
- Avoid using/calling the kernel, and rely on a user-mode execution runtime environment
- When targeting interrupt latency at a 3-10 us average and 15-30 us worst case requirements

ENEA

LWRT
Linux User Space Runtime Environment

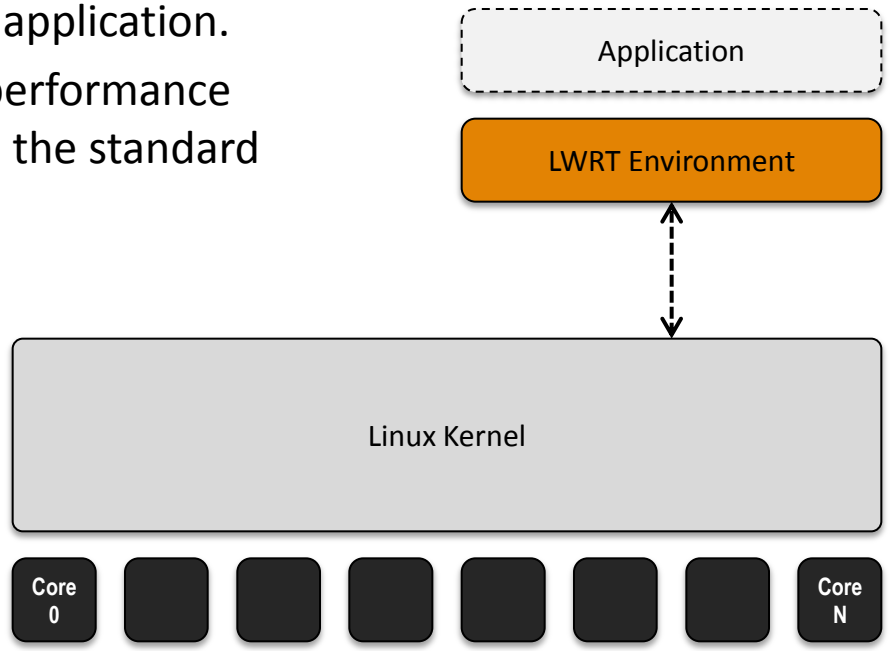
For complex networking and communications systems that *demand* firm real-time performance and behavior

LWRT == Light-weight Runtime environment

ENEA

What is LWRT?

- LWRT is a “Light-Weight Runtime (Environment)” built upon Linux.
- LWRT runs in most part in user-space,
  and is a essentially a library linked to the application.
- The LWRT environment provides better performance
  and realtime characteristics compared to the standard
  POSIX/Linux environment.



ENEA

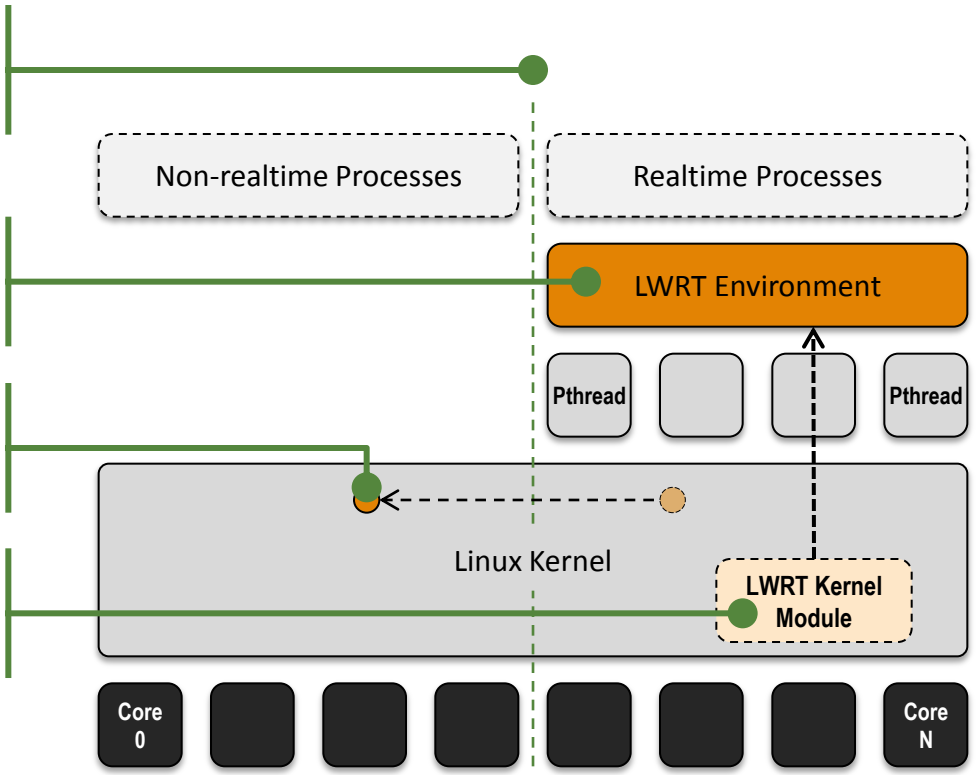
How does LWRT work?

LWRT partitions the system into one realtime domain and one non-realtime domain.

LWRT adds a user-mode runtime environment, including an optimized user-mode scheduler.

LWRT migrates some specific kernel functionality (e.g. timers) away from the realtime domain.

LWRT adds a kernel module to catch and forward interrupts to the user-mode environment.



What are the benefits of LWRT?

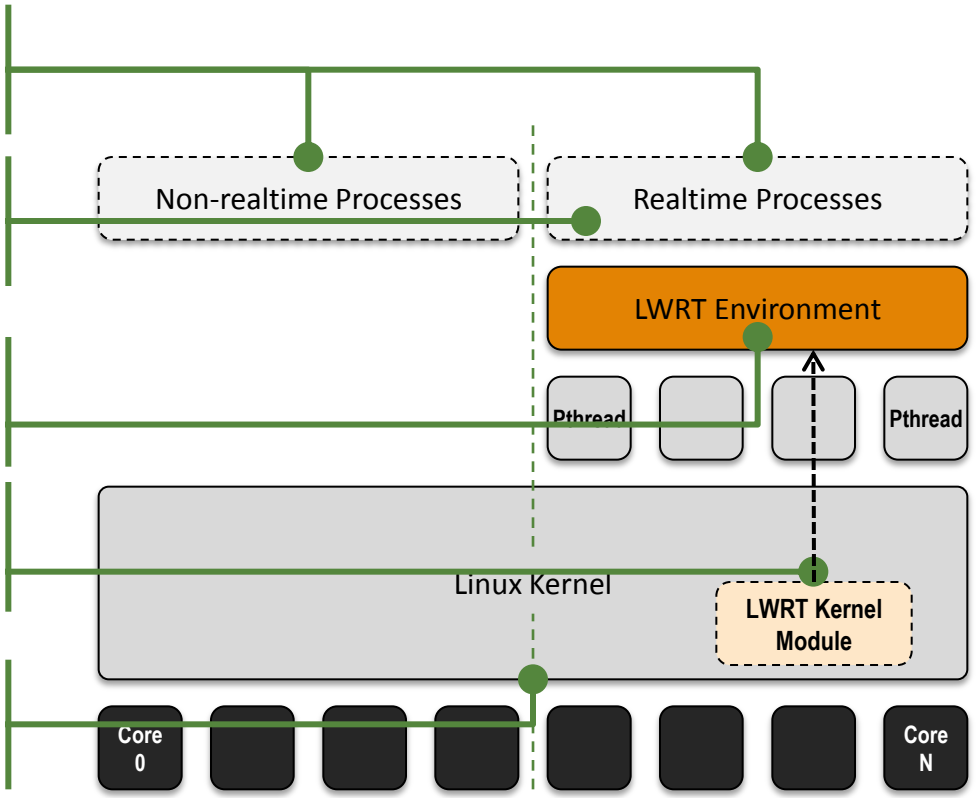
LWRT provides low latency and high throughput. LWRT does not depend on the PREEMPT_RT patch, and does not affect throughput negatively.

LWRT provides a solution that is unencumbered by GPL, even for interrupt driven code which can be placed in user-space without any major penalty.

LWRT provides optimized APIs for realtime applications, and allows the same application to use the POSIX/Linux APIs when realtime doesn't matter.

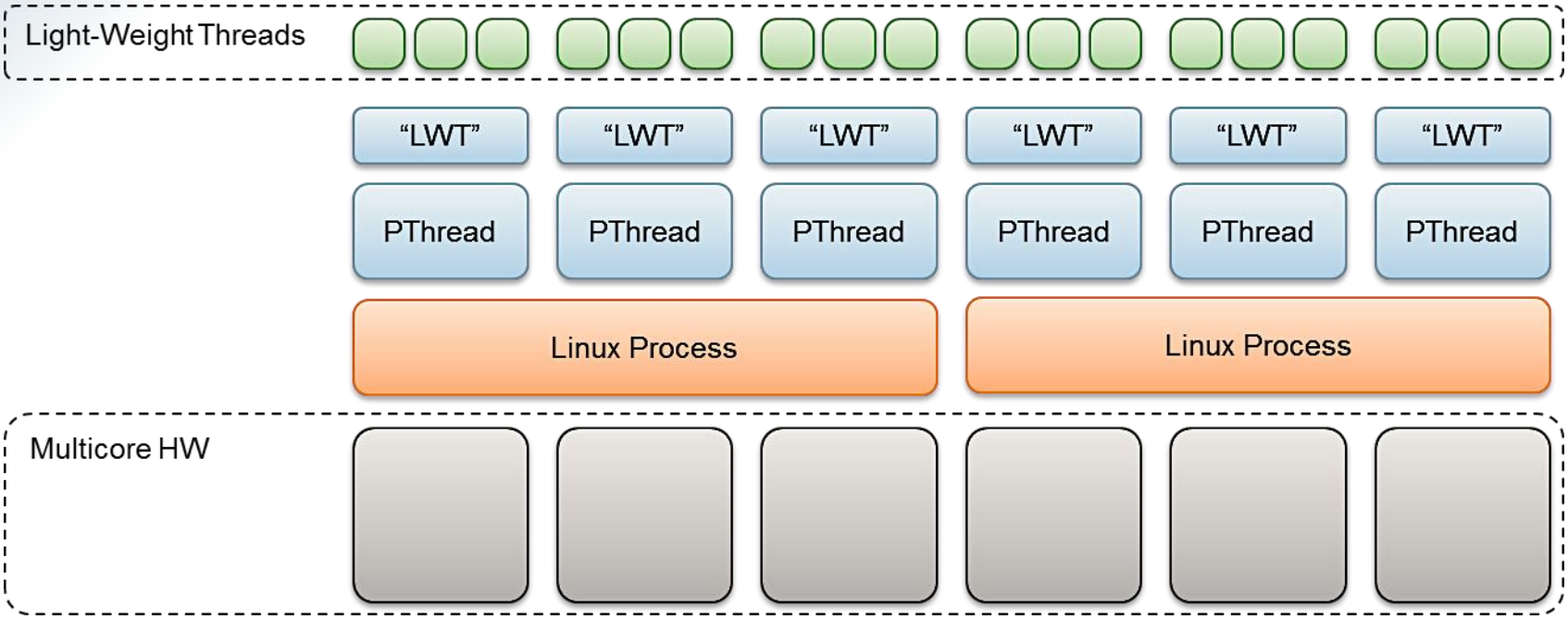
LWRT provides very good (i.e. low-latency) interrupt response time, all the way up to user-mode.

LWRT is an "all-Linux" solution, based on a single Linux Kernel. Thus, almost all tools from the existing Linux ecosystem will be available.



LWRT Architecture and Services

LWRT Architecture



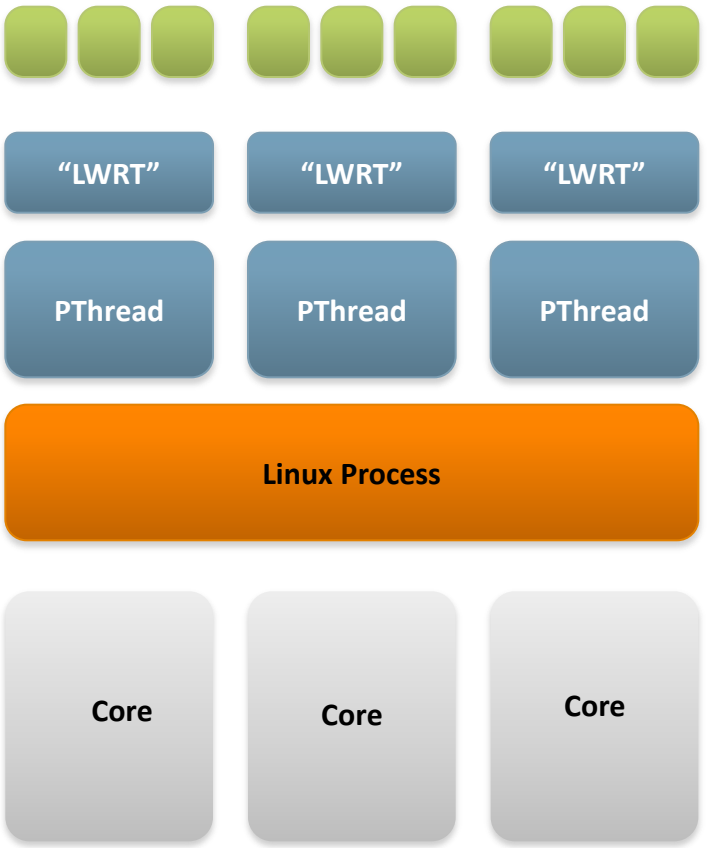
- Multiple threads per core
- Deterministic scheduling
- Low scheduling overhead

- Low inter-thread messaging overhead
- Cheap thread creation
- Low overhead buffer management
- Timer services

ENEA

So LWRT is a Different Multi-threading Model

- A Light-weight thread Environment implemented in user-space over a single pthread
- A user-mode scheduler in the scope of a Linux process
  - - No kernel thread context switch
  - - Preemptive, priority based scheduler
  - - 100% Linux, utilizes same Linux tools as rest of system
  - - Complete access to normal POSIX/libC
- Support for message-passing between threads (and processes). With simple buffer management (for messages), and interrupt handling



ENEA

LWRT Performance

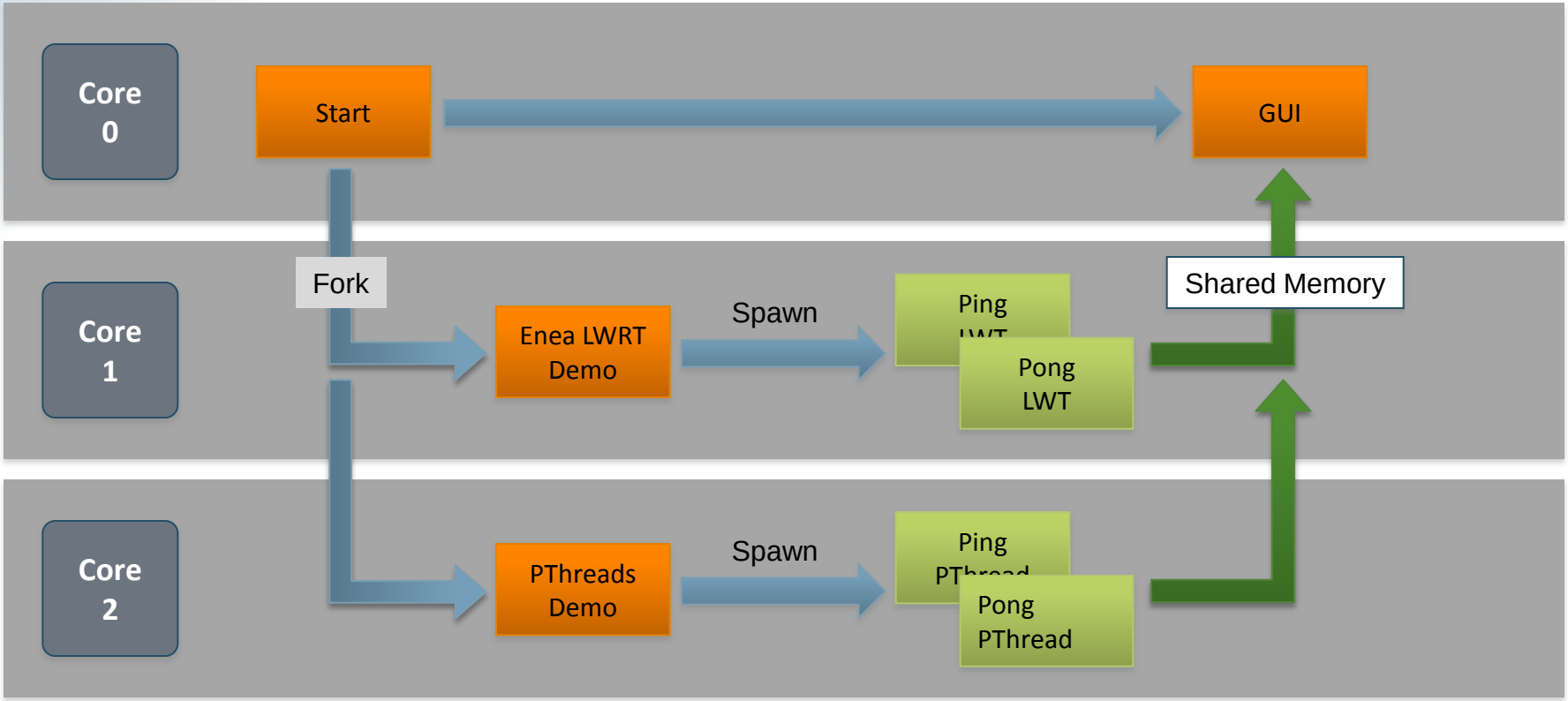
Preliminary Results

Some Hard Numbers

- WCDMA/LTE Base Station Performance Requirements:
  - Average interrupt latency 3-5 usec.
  - Maximum interrupt latency 20-30 usec.
  - Approx. ~30 context switches/TTI or LTE 2
  - WCDMA frame sync interrupt every 66.66… usec.
  - WCDMA TTI interrupt (every 2 msec for HSDPA).
  - LTE 2 interrupts per slot (every 0.5 msec)
  - Interrupt overhead < 10,000 cycles for A15 @ 1.5 GHz.

ENEA

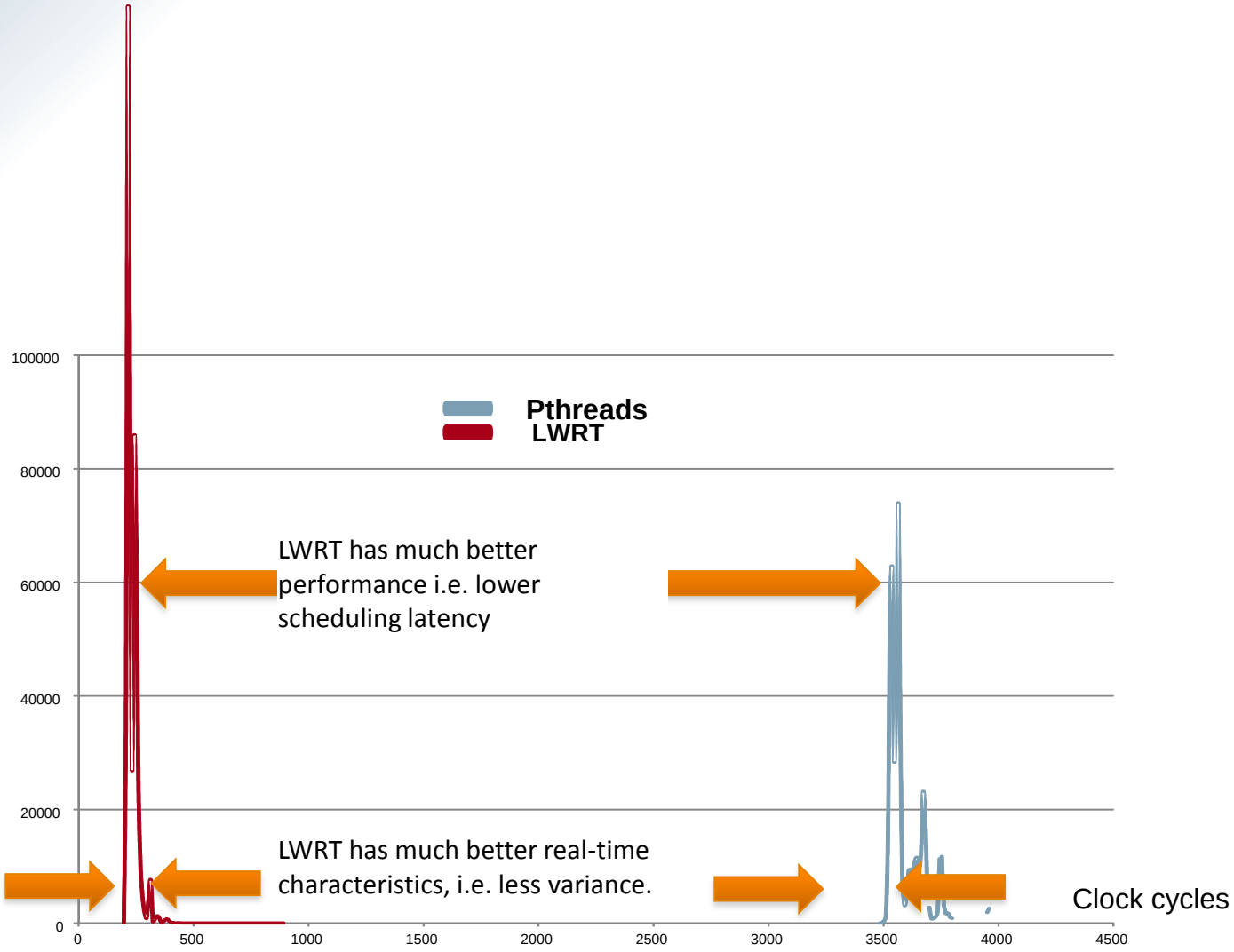
Context Switching and Inter-thread Communications LWRT vs Pthreads Benchmark Setup



- Measurements done on x86 using TSC (time stamp counter)
- Demo HW: AMD Phenom II N620 Dual-Core processor @ 2.8GHz.
- Linux: OpenSuSE 12.1, 32 bit

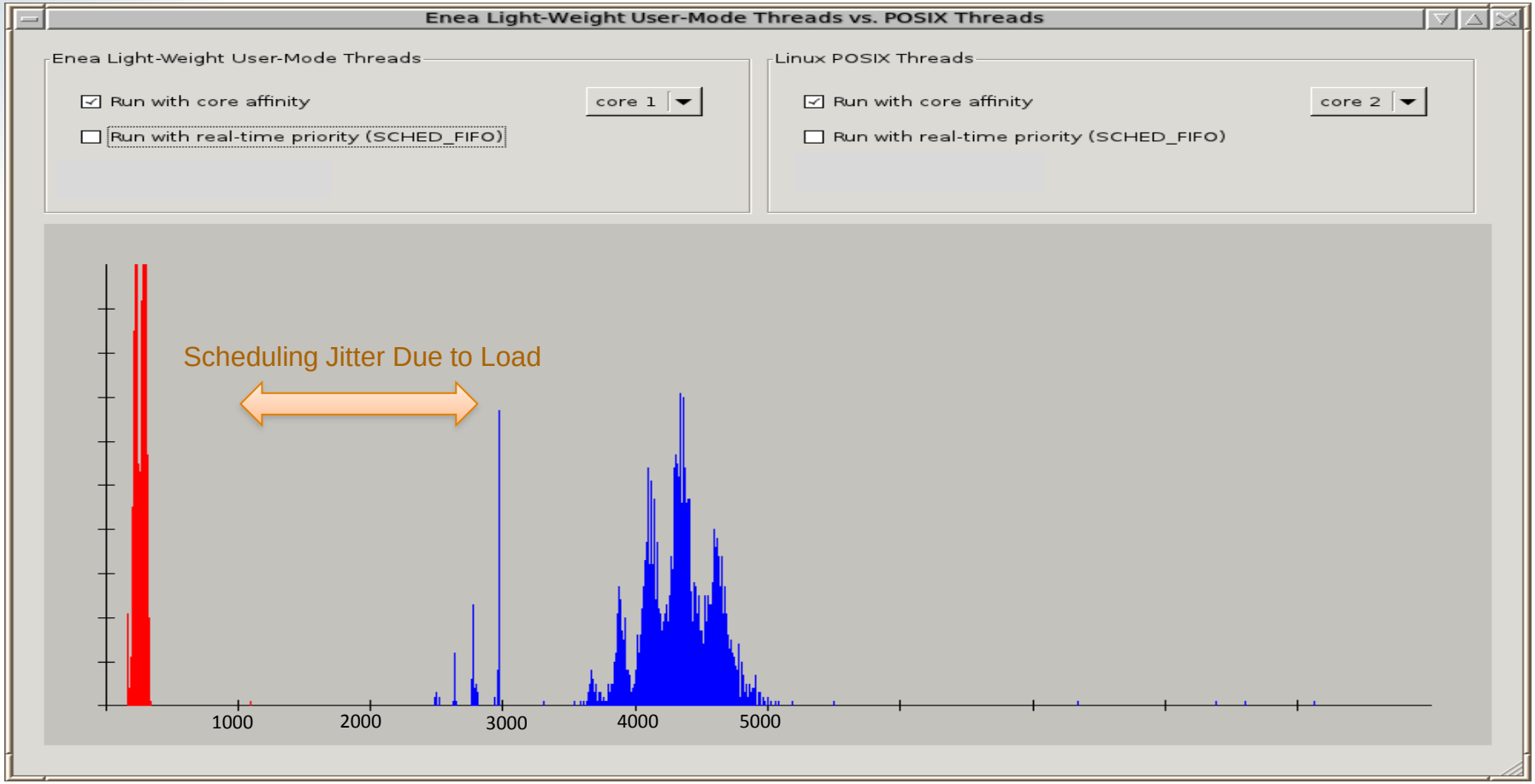
ENEA

LWRT vs Pthreads - Context Switch Overhead



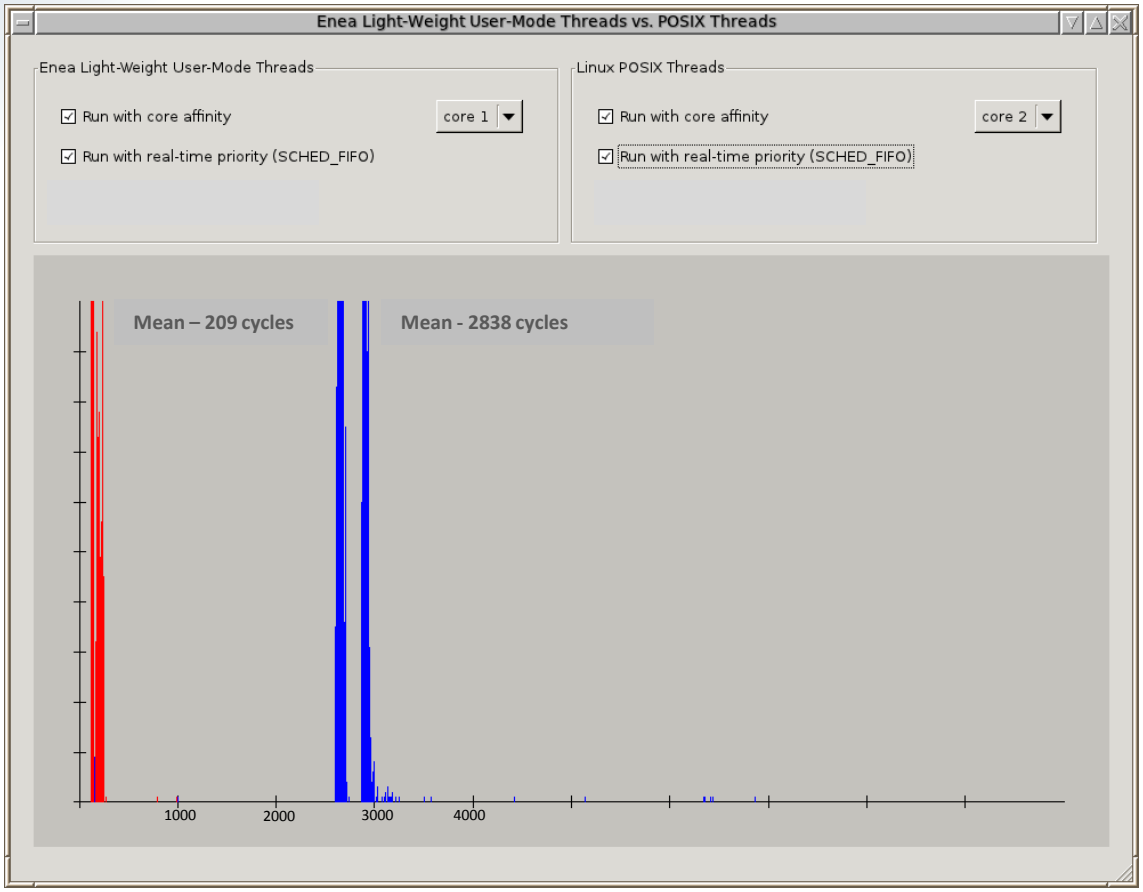
LWRT vs Pthreads

Scheduling Behavior under System Load



LWRT vs Pthreads

Inter-thread Communications



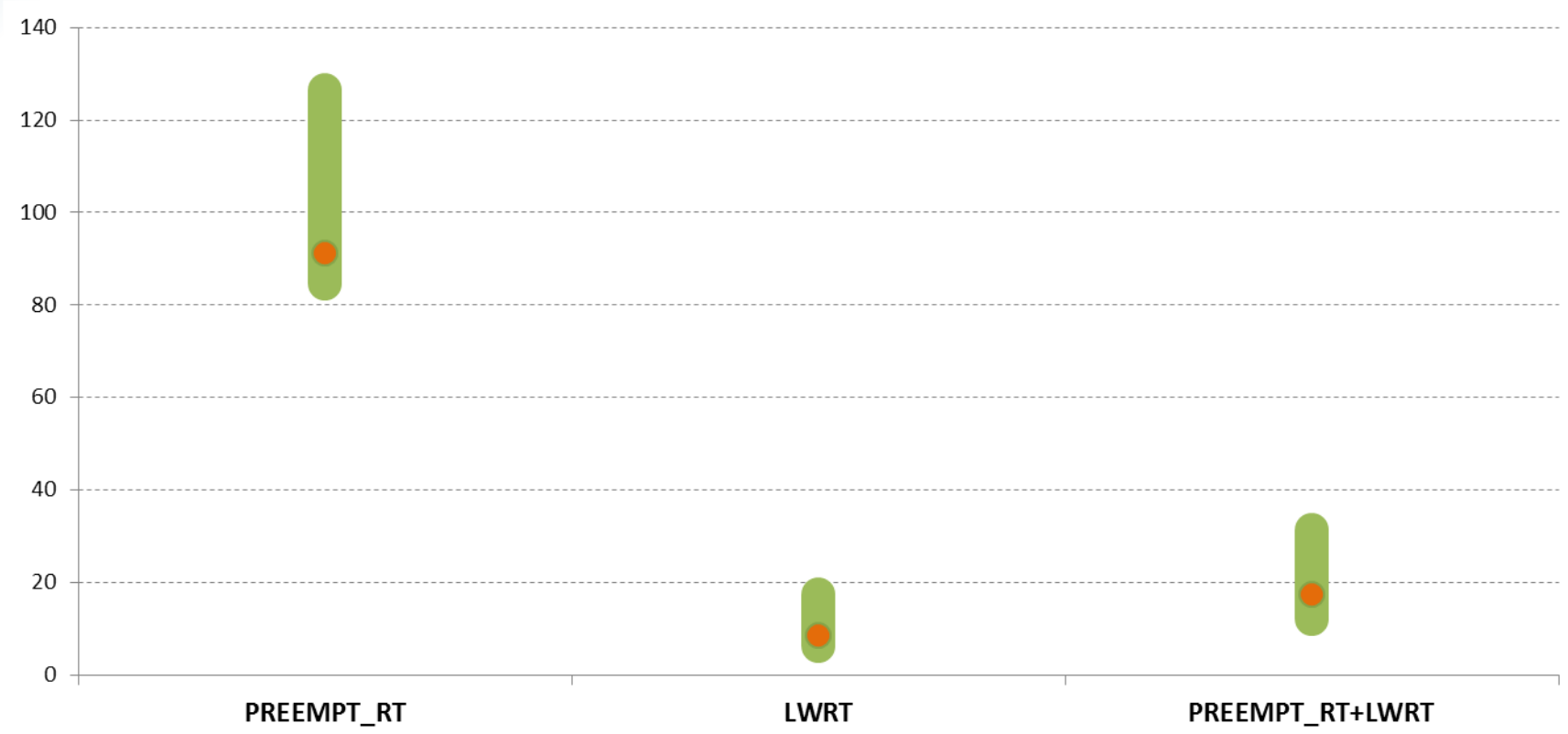
- Ping-Pong benchmark measuring scheduling time/latency
- Histogram show one-trip latency LWT example based on signaling
- Pthread example based on semaphores
- In the demo, the user can interactively play with core affinity and scheduling policy

ENEA

LWRT and PREEMPT_RT

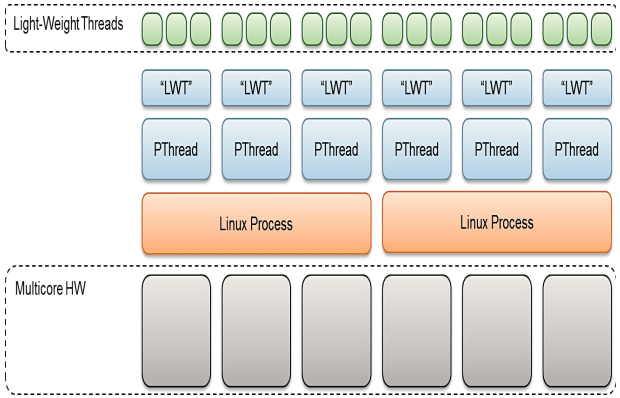
Interrupt Latency under System Load

Interrupt latency measured for Linux 3.0.28 on an Intel Core 2 T5500 @ 1.66GHz.
Min/max/average in µ-seconds
Using ”Stress” open source for Load
“End to End” Measure – from HW interrupt raised to user application start



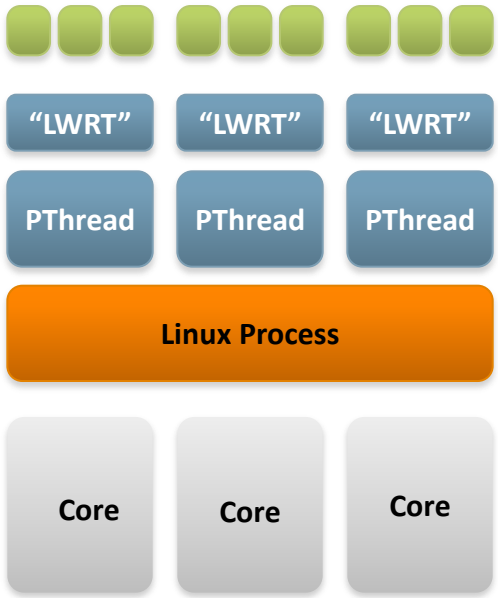
Comparing LWRT and PREEMPT_RT

- PREEMPT_RT applies to the entire system, whereas LWRT partitions the system.
- PREEMPT_RT improves latency at the expense of throughput. LWRT on the other hand is built upon the idea that one part of the system might be optimized for latency, and one part might be more focused on throughput.
- PREEMPT_RT makes significant modifications to the Linux kernel (e.g. 500+ locations, 11,500+ lines). LWRT makes minimal modifications, and tries to use as much of “standard” kernel as possible.
- PREEMPT_RT assumes that all device drivers have been adapted to the patch. LWRT puts no special requirements on device driver.
- LWRT achieves same interrupt latency as PREEMP_RT, with better end-to-end performance (time)



Current Status of LWRT

- LWRT is a “PROTOTYPE”
  - No real build environment
  - Manual configuration for vertical partitioning
  - Still exist some non-RT parts (timeouts, timers, …)
  - API’s are not standard
  - Limited ways to use POSIX API “safely”
  - Interrupt handling is but one example
  - Thin documentation
- Moving forward
  - Add build and configuration capability
  - POSIX API “wrapper” for LWRT functions
  - Move non-RT critical parts to non-RT domain
  - Safe usage of full POSIX
  - Much more work on interrupt handling



Thank you for attending

Questions?