# Real-Time Linux and Open MPI -- An Introduction and Case Study

Nivedita Singhvi

April 15, 2010 : Linux Foundation Collab Summit

- Introduction to Real-Time Linux
- Running an Open MPI benchmark on Real-Time Linux
  - (ok, not so much on this....)

# Real-Time and HPC have a lot in common....

- Somewhat poorly defined terms!
  - mean different things to different people
- Include a  broad range of solution space
- Known for being areas of challenging requirements
  - complex problems to solve
  - often tends not to be default path, require custom solutions
  - want low latency, reduced OS jitter
  - have critical applications that need a lot of processor time

So if not a marriage made in heaven....
we should at least be dating, right?

# Well, while not quite the Montagues and the Capulets...

- Opposite ends of the size spectrum, typically
  - HPC typically massive installations, larger systems
  - Real-Time at the smaller end typically, embedded
    - still in its infancy in terms of scaling...
- Complex programming models vs simpler, more general needs
- Real-Time is coming out of the embedded and/or custom solution space into general-purpose systems solving problems that would not have been called real-time even a decade ago

# So what is this Real-Time Linux?

- Enterprise Real-Time is “softer” real-time
  - The world is not going to explode should you miss a deadline or a latency guarantee
- Typically want “as fast as possible” but as/more importantly:
  - “deterministically”, “predictably”, “consistently”

# Normal Non-Real-Time Compute Model

- **Excellent throughput but with wide variability**
- **Average times may vary**
- **No prioritisation**
- **Favors raw throughput, latencies uncontrolled**
- **Acceptable non-real-time behavior:**
  - **100,000 transactions**
  - **99,999 at 5ms each and 1 at 2 *seconds***
  - **Unacceptable for real-time systems**
  - **(courtesy Paul McKenney)**

# What to expect from Real Time Linux?

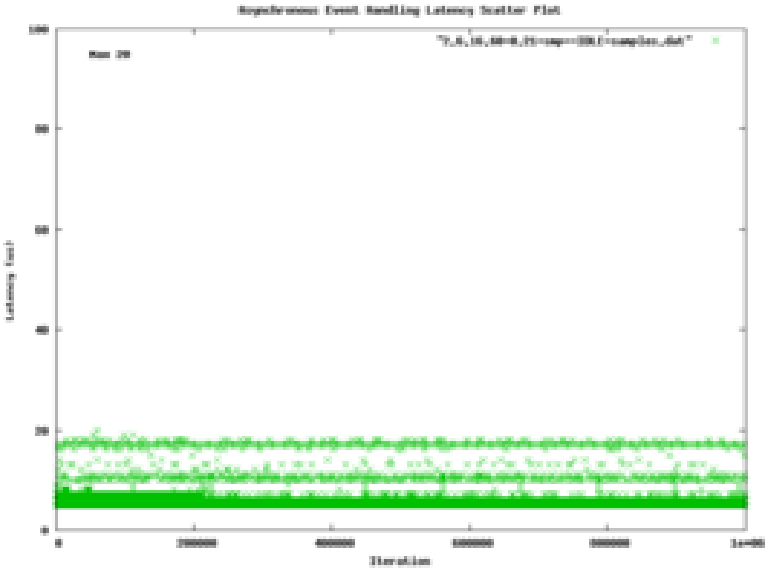
- Real Time Linux is not necessarily Real Fast Linux!!
- Determinism
  - sacrifice throughput to cap max latencies
- Highest priority real time tasks take precedence
  - they get system resources (*priority-driven environment*)
- The ability to shoot yourself in the foot!
  - “with great power comes great responsibility”

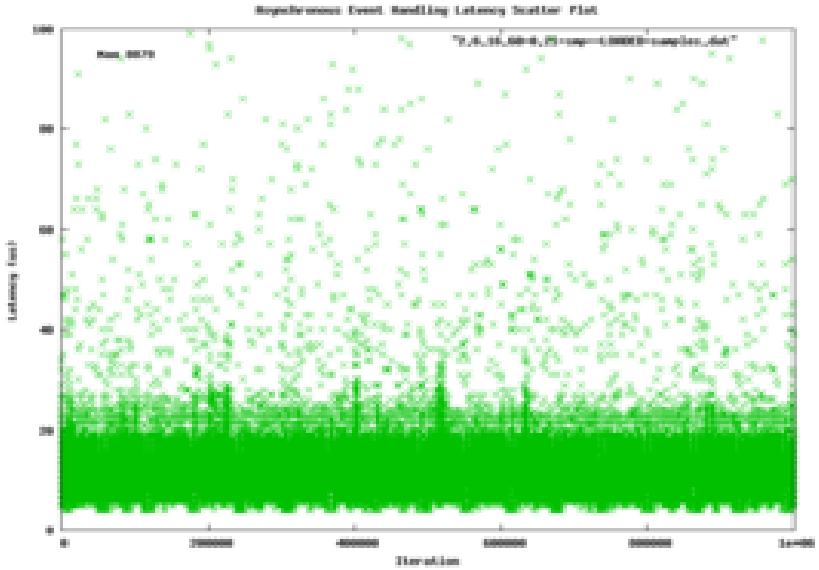
# Real-Time Kernel Goals - Determistic

- Deterministic
  - maximum latency bound
  - want to guarantee operations will take “at most 30 seconds”
  - no matter what happens (all paths)
  - Better to have 300 operations that take 20ms rather than 299 operations that take 15ms and 1 operation that takes 60ms.
  - ***Real-Time applications need bounded latencies***
  - Sacrifice throughput, performance for determinism
  - Trade-off benefits high priority tasks at the expense of low priority tasks

# Mainline idle/loaded - async-handler

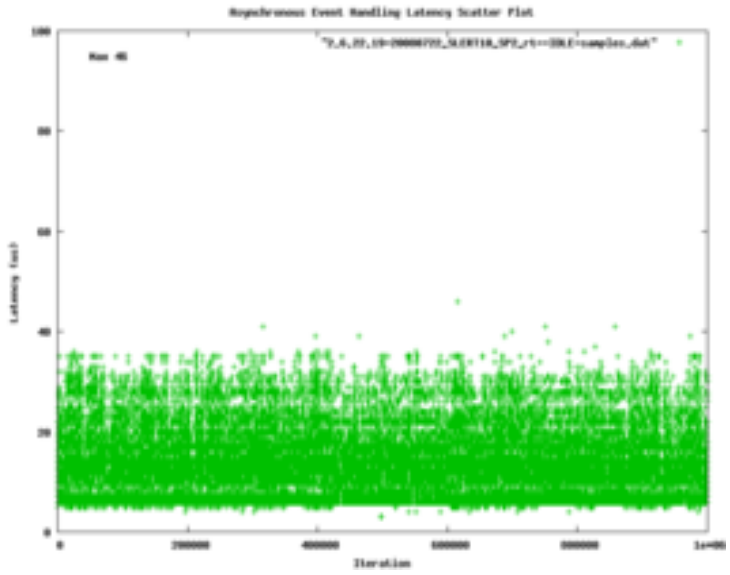
- Scatterplot

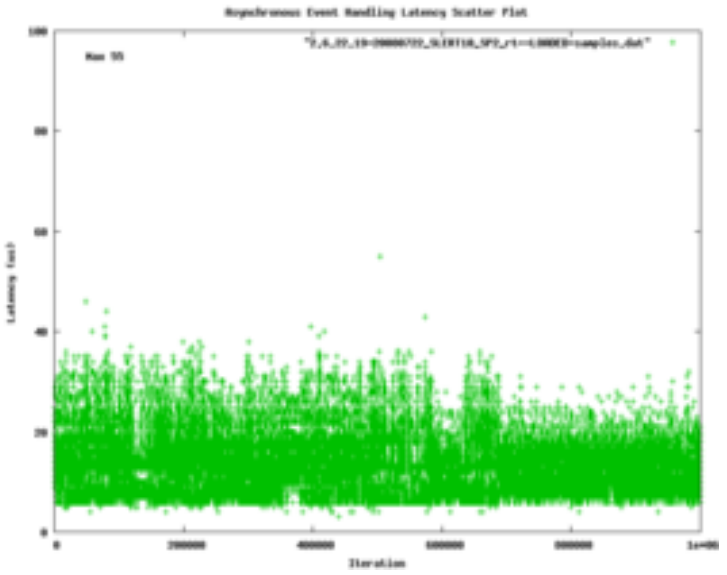




# RT idle/loaded - async-handler

- Scatterplot



# Real-Time Kernel Characteristics - Priority Driven

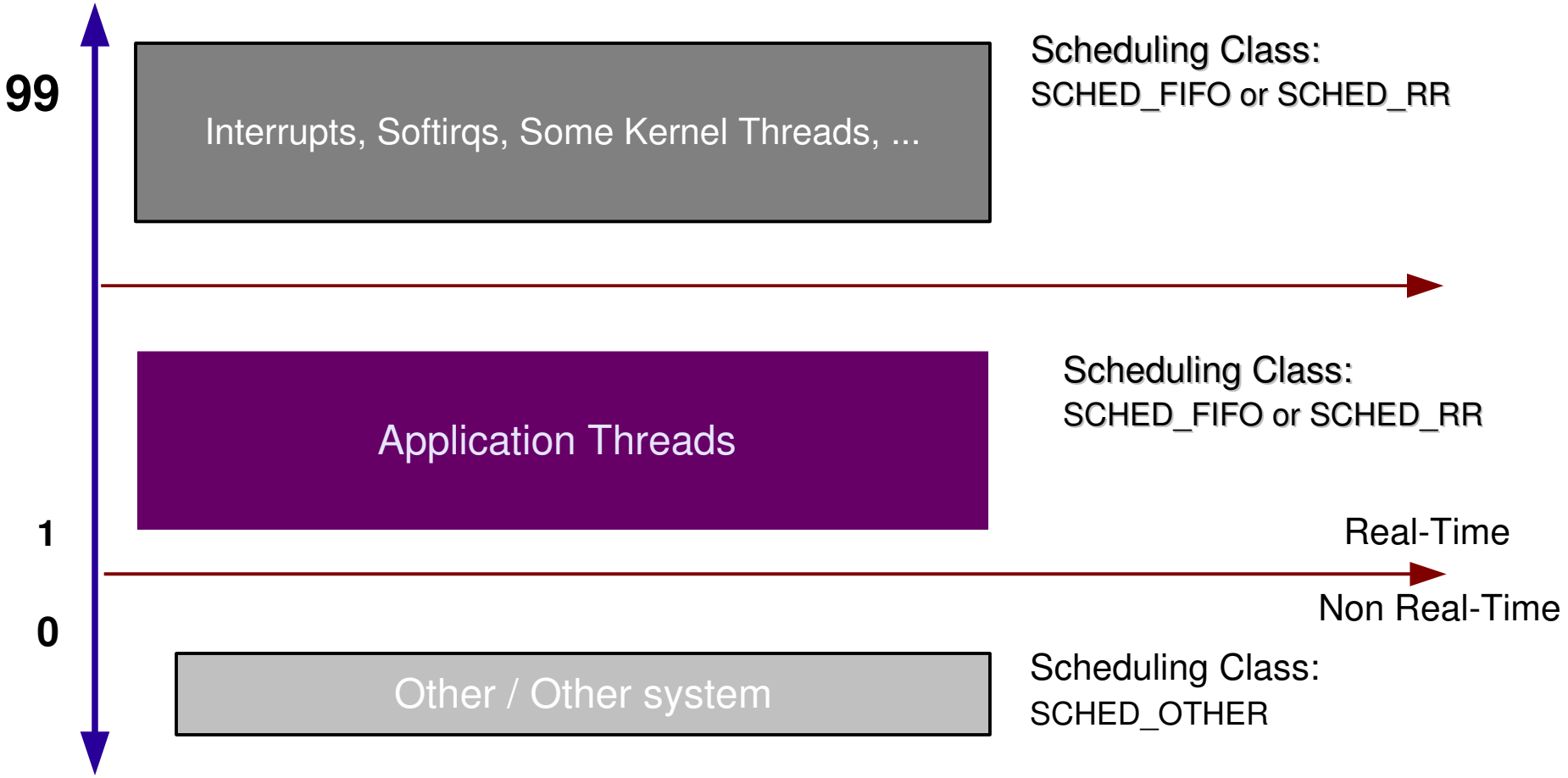
- Priority Driven
  - Traditional OSs are optimized for performance, overall throughput
  - BUT on RT we:
  - ***Always execute the N highest priority threads on an N-CPU box***
    - ***Latency guarantees can only be met if adequate resources!***
  - SCHED_FIFO not limited by timeslices (will not relinquish the CPU!)
  - if a higher priority thread becomes runnable, lower priority thread is migrated or suspended
    - this can cause some cache thrashing
    - inefficient overall (might yield almost immediately, ...)
    - potential deadlock, need care while programming, administering
  - Sacrifice system throughput for real-time priority ordered hierarchy
  - Trade-off benefits high priority tasks at the expense of low priority tasks

- Why do you need a Real-Time Linux kernel?
  - isn't mainline (POSIX 4) mostly there? (scheduler, timers, async I/O, ...)
- Mainline Linux
  - a real-time scheduler (CFS), high-resolution timers, tools - chrt, ftrace, perf, ...,
- Real-Time Linux (CONFIG_PREEMPT_RT)
  - more preemptive kernel (user space has greater control over kernel)
  - interrupt handling is different (irqs, softirqs are kernel threads)
  - locking is modified (rt mutexes replace spinlocks)
    - priority inheritance, preemptible
  - whole lot of other underlying infrastructure to support this
    - time, timer infrastructure; preemptive RCU, ....

# Real-Time Kernel Features - Scheduling

- CFS Scheduler - default scheduling algorithm
- **SCHEDULING POLICIES: SCHED_RR, SCHED_FIFO, SCHED_OTHER**
  - **SCHED_FIFO** not limited by timeslices
    - first highest priority task that's runnable gets CPU / keeps CPU
    - preempted only by higher priority task
    - real-time priority between 1-99
  - **SCHED_RR** has priority but is timesliced amongst same priority level
    - round robin : once timeslice completes, goes to back of runqueue
  - **SCHED_OTHER** does not have a real-time priority (time-sliced, priority 0)
- **SCHEDULING PRIORITIES:** SCHED_FIFO, SCHED_RR: 1 - 99

# Real Time Tuning of Applications



# Real-Time Kernel Features - Threaded Interrupts

- HW and SW Interrupts on RT are kernel threads
- Work deferred from HW interrupt to softirq (now a kernel thread)
  - except timer interrupt
- Their priorities fall somewhere between 30 - 99 (current default)
  - can be tricky if large number of high priority threads
  - manipulating these priorities is HIGH ART
  - need to ensure critical kernel activities not blocked
- Applications used to get interrupts which ran till they completed
  - caused a lot of variance in the time it took to run
  - now kernel activity competes for time with high priority real-time processes
  - allows high priority real-time tasks to run with fewer interrupts

- Would HPC benefit from using Real-Time?

- Would HPC benefit from using Real-Time?
  - Depends!

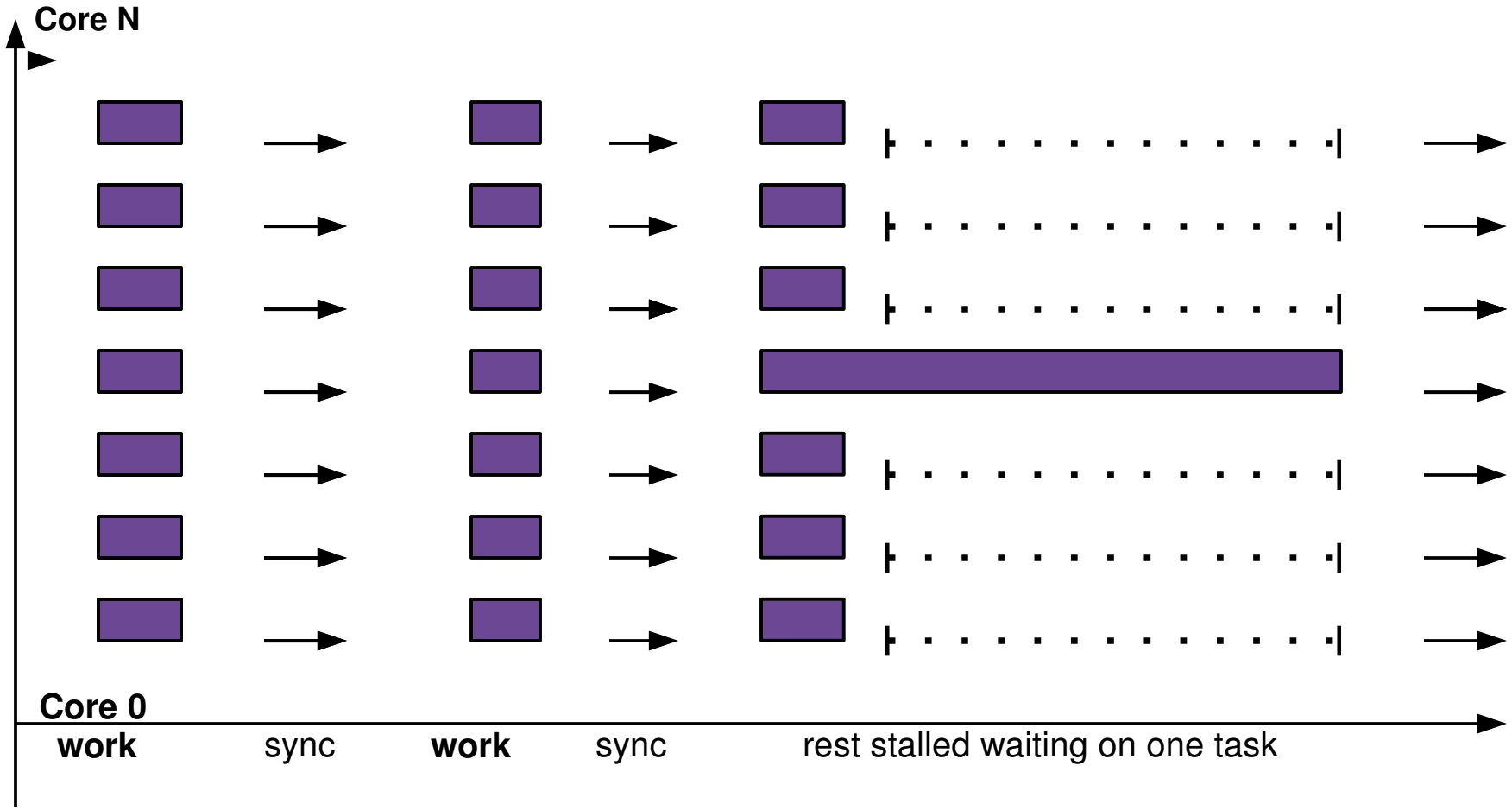
- Would HPC benefit from using Real-Time?
  - Depends!
- Should HPC be paying attention to Real-Time?

- Would HPC benefit from using Real-Time?
  - Depends!
- Should HPC be paying attention to Real-Time?
  - Depends!

- Would HPC benefit from using Real-Time?
  - Depends!
- Should HPC be paying attention to Real-Time?
  - Depends!
- Anything HPC can learn from or share with Real-Time?

- Would HPC benefit from using Real-Time?
  - Depends!
- Should HPC be paying attention to Real-Time?
  - Depends!
- Anything HPC can learn from or share with Real-Time?
  - Depends!

# Jitter is bad for lock-step applications (common in HPC)




**Improved determinism (max latencies capped) can improve throughput**

# Sources of Jitter

- Jitter - variation in execution times (also OS overhead)
  - lack of determinism
- Sources of jitter for application threads:
  - Scheduling - CPU contention
  - Lock contention
  - Interrupts - Hardware, Softirq (periodic, asynchronous)
  - Variance in memory allocation times, etc...
  - Device/Network/external  factors
  - system mgt events, throttling, ...

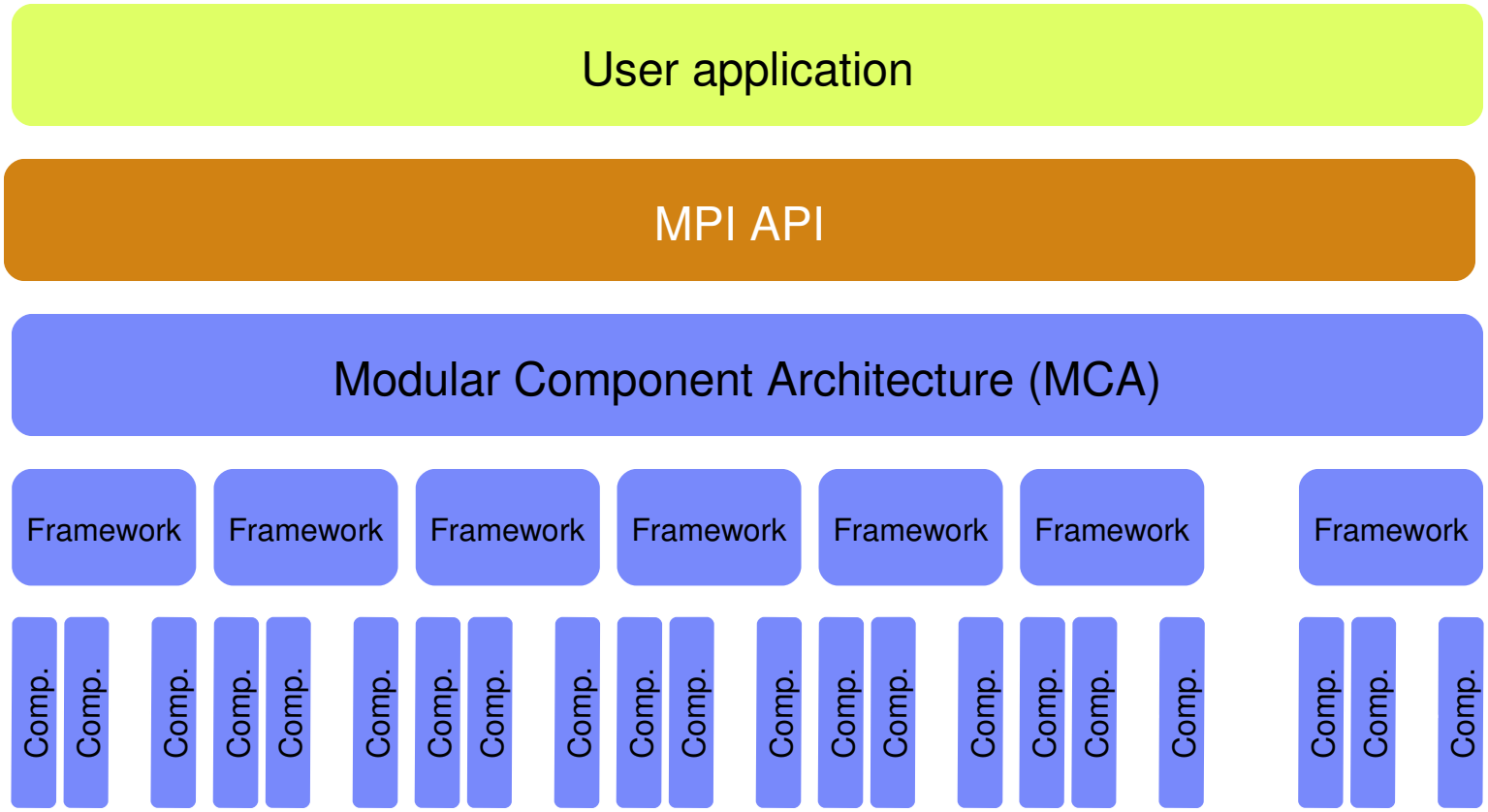
# What can users do?

- On mainline Linux (non-RT)
  - real-time priority scheduling
  - chrt to set scheduling class and priority (careful!)
  - taskset to pin interrupts, threads
- Run on RT  (tuning, priorities)
- Run on RT with application code changes
  - e.g. mlockall, etc to pin memory

# Typical....

# (don't do that)

# Plugin High-Level View



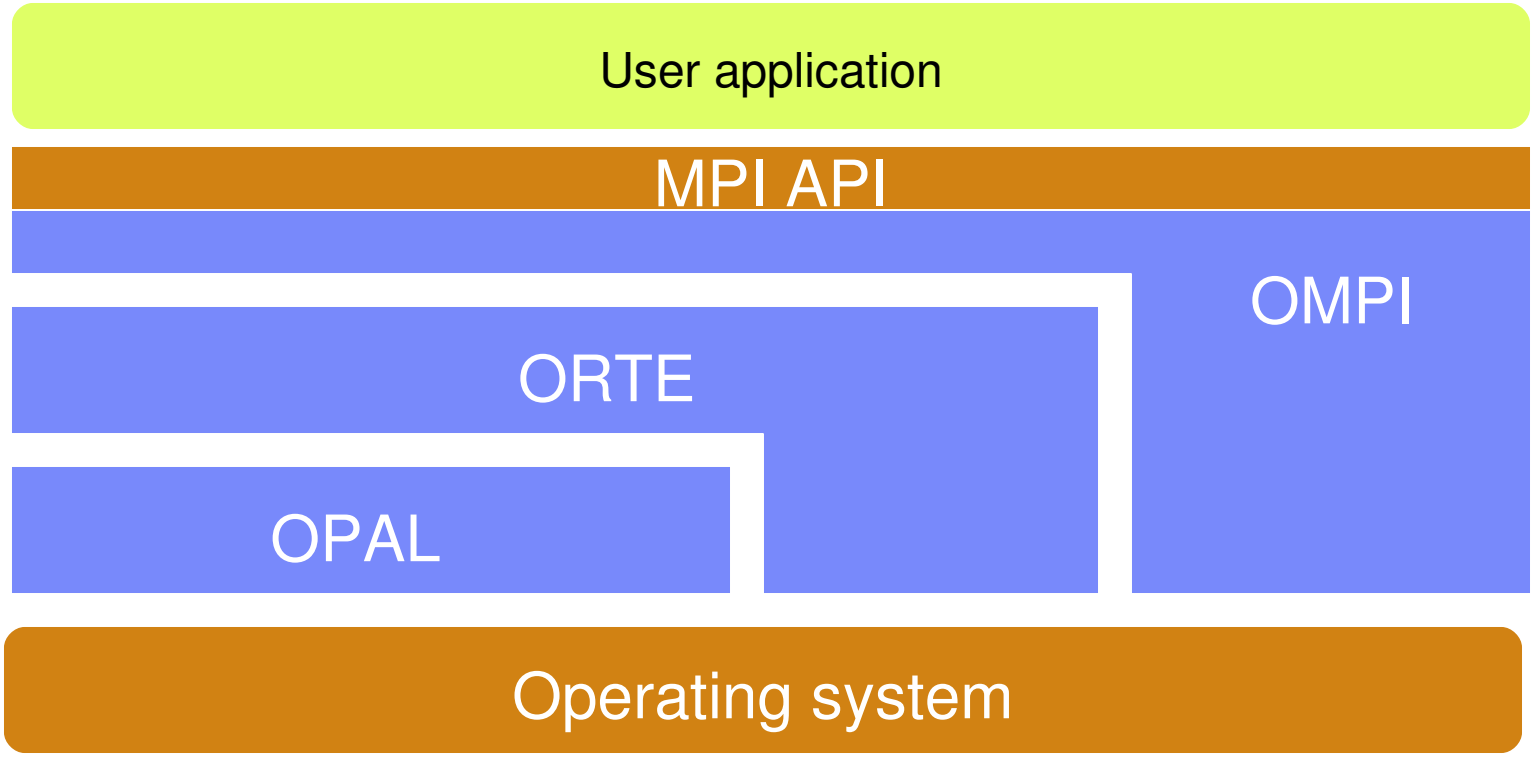

Courtesy: Jeff Squyres "What is MPI" 2008

# Three Main Code Sections

- Open MPI layer (OMPI)
  - Top-level MPI API and supporting logic
- Open Run-Time Environment (ORTE)
  - Interface to back-end run-time system
- Open Portability Access Layer (OPAL)
  - OS / utility code (lists, reference counting, etc.)
- Dependencies - not layers
  - OMPI → ORTE → OPAL

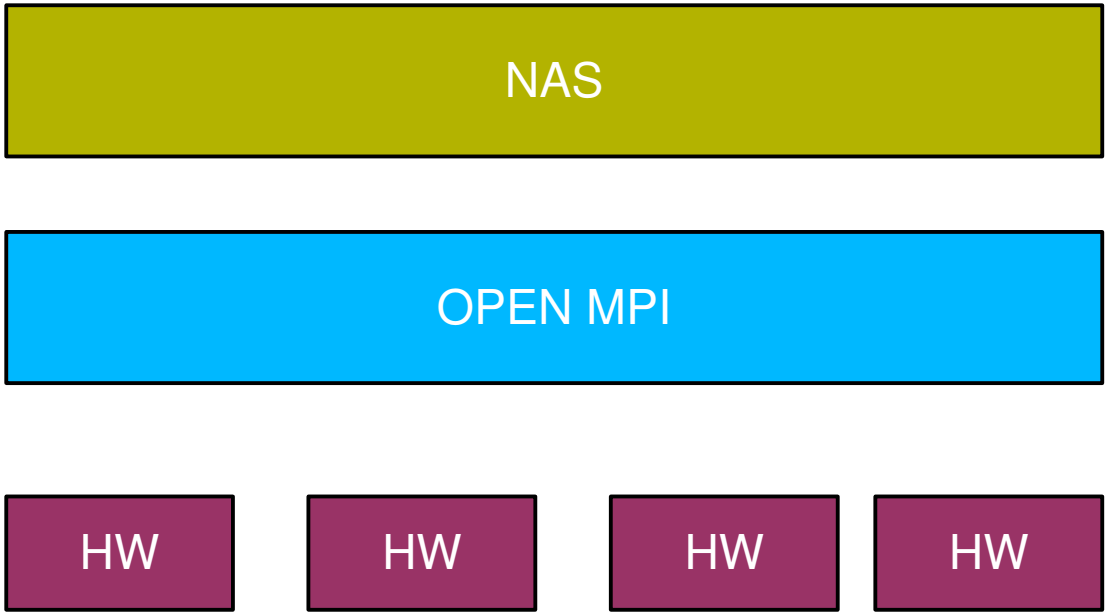
Courtesy Brad Benton

# Three Main Code Sections



## NASA's NAS Parallel Benchmark (NPB)

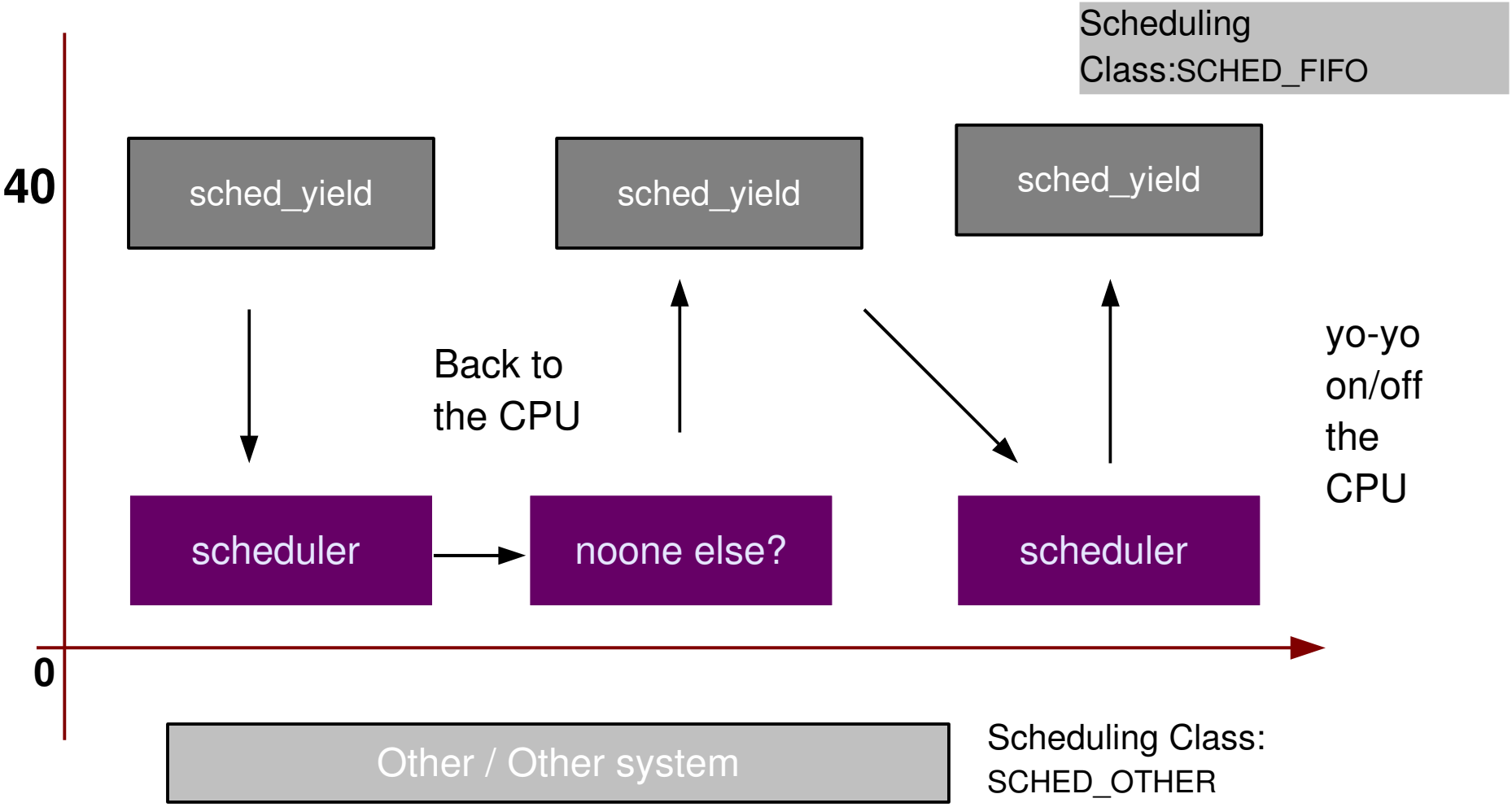
| | |
|---|---|
| BT | Block Tridiagonal |
| CG | Conjugate Gradient |
| EP | Embarrassingly Parallel |
| FT | Fast Fourier Transform |
| IS | Integer Sort |
| LU | Lower-Upper symmetric Gauss-Seidel |
| MG | MultiGrid |
| SP | Scalar Pentadiagonal |



# Shooting oneself in the foot...

- Immediate livelock
  - sched_yield() use
  - application believes it's yielding the CPU
    - just wants a delay
    - needs other activity to occur on the system for forward progress
    - depencies
    - asynchronous I/O from other nodes, ...
  - well, guess what...

# sched_yield() doesn't do what you think it does...



# So if there's one lesson - Real-Time Needs...

- Top down architecture design (**)
- Identified  dependencies
  - correctly prioritize in some sort of hierarchical sorting
  - ensure forward progress of your application

# Setting real-time attributes

- **rtctl** utility to manage threads, processes
  - /etc/rtgroups file
  - set system default scheduling priority/class for processes
  - note, rtctl utility default differs from kernel default
    - "ps -eLo pid,rtprio,policy,comm"
- command line using **chrt**

- Lots of jitter measurement and other tools available
  - resources on the web
  - cyclictest
  - ftrace, perf, kernelshark, trace-cmd, ...
- Lots of tuning help available
  - https://rt.wiki.kernel.org/index.php/Main_Page
  - https://www.ibm.com/developerworks/wikis/display/LinuxP/OS+Jitter+Mitigation+Techniques (IBM-specific)

## Legal Statements

- Copyright International Business Machines Corporation 2010.
- Permission to redistribute in accordance with Linux Foundation Collaboration Summit submission guidelines is granted; all other rights reserved.
- This work represents the view of the authors and does not necessarily represent the view of IBM or Intel.
- IBM, IBM logo, ibm.com are trademarks of International Business Machines Corporation in the United States, other countries, or both.
- Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, and service names may be trademarks or service marks of others.
- References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

## Legal Statements

- INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.