CPU power saving methods for real-time workloads

Ramesh Thomas
Open Source Technology Center, Intel

Background

- Part of research on best configurations, methods and tools to help Real-Time Application development in RT Linux
- This talk is about CPU idle power management (C states) within real-time workloads
- Linux Foundation Wiki https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/cpuidle

Introduction

- Deeper C states are generally avoided in real-time configurations
  - Introduces jitter impacting determinism requirements

- Methods discussed here would allow real-time applications to take advantage of C state power savings without impacting determinism

Why do we need this?

- CPU idle saves power that can be used by active cores, reduced cooling needs and a lot more…

- Different C states (C0, C1, C6…) vary in degree of power savings and latency impact
  - Should be able to choose acceptable ones

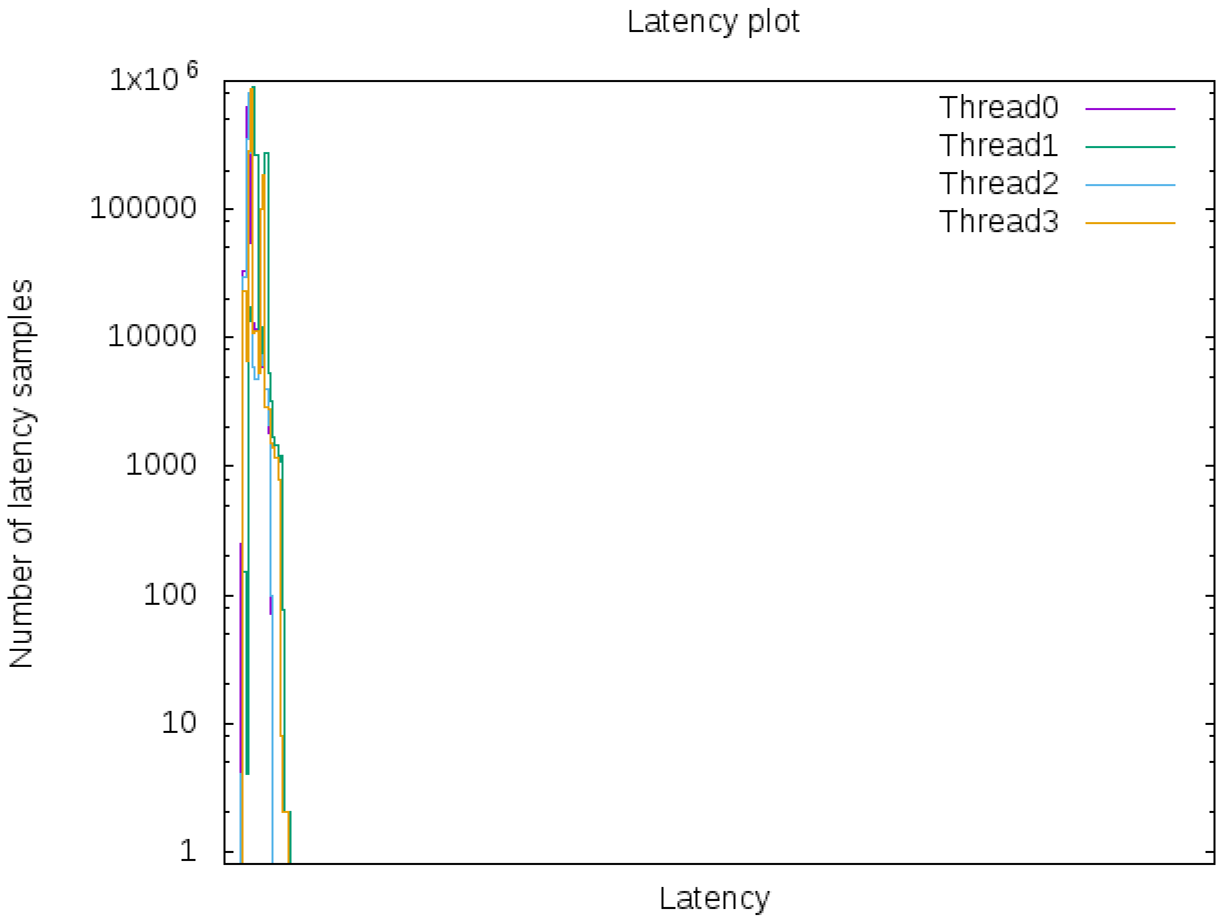
- Different cores may have different RT requirements
  - Should be able to manage C states in each core separately

Focus on Determinism

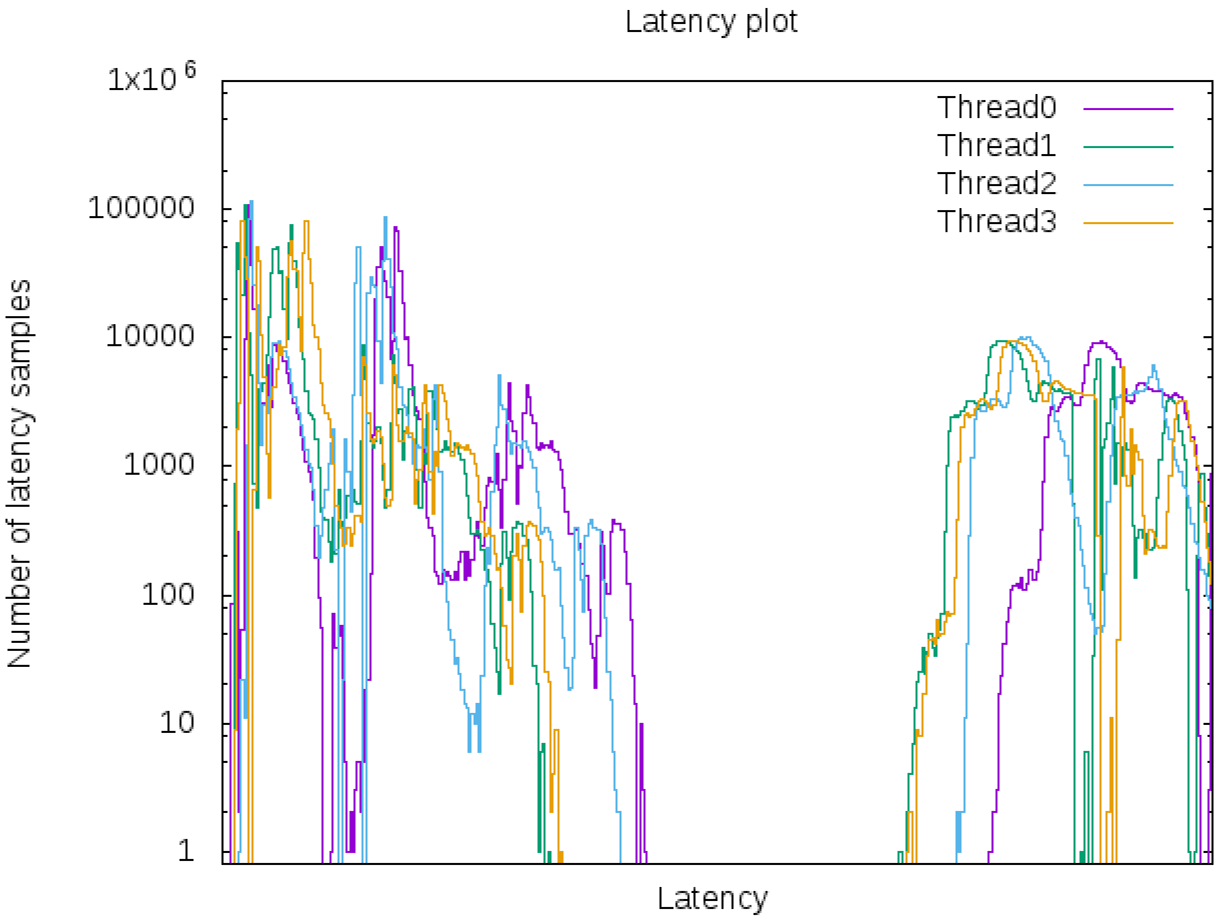
- Solutions tailored to a specific requirement

- Main requirement for real-time applications is **determinism**

Good



Bad



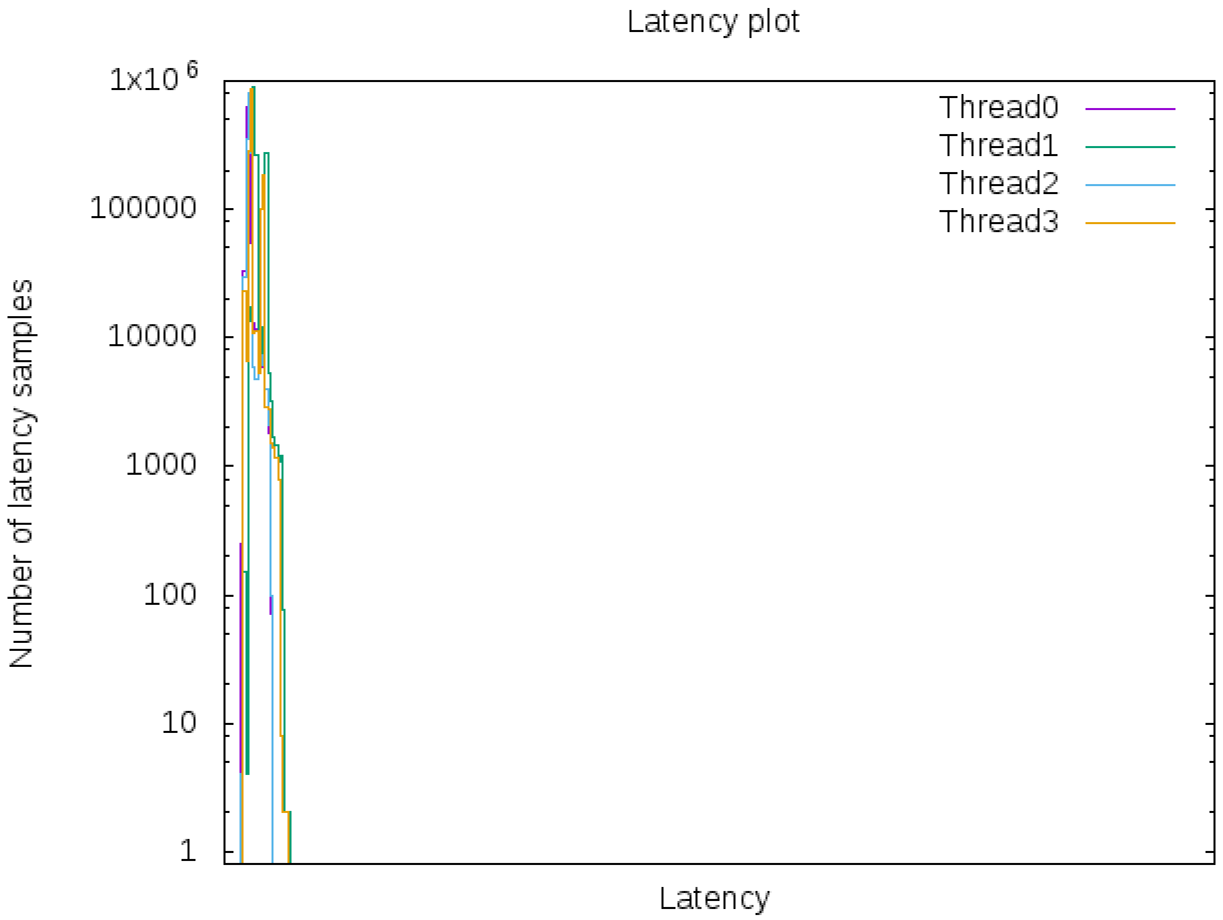
Example uses Intel® NUC kit with Intel® Celeron® Processor J3455.
Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Performance varies depending on system configuration.

Where does the jitter come from?

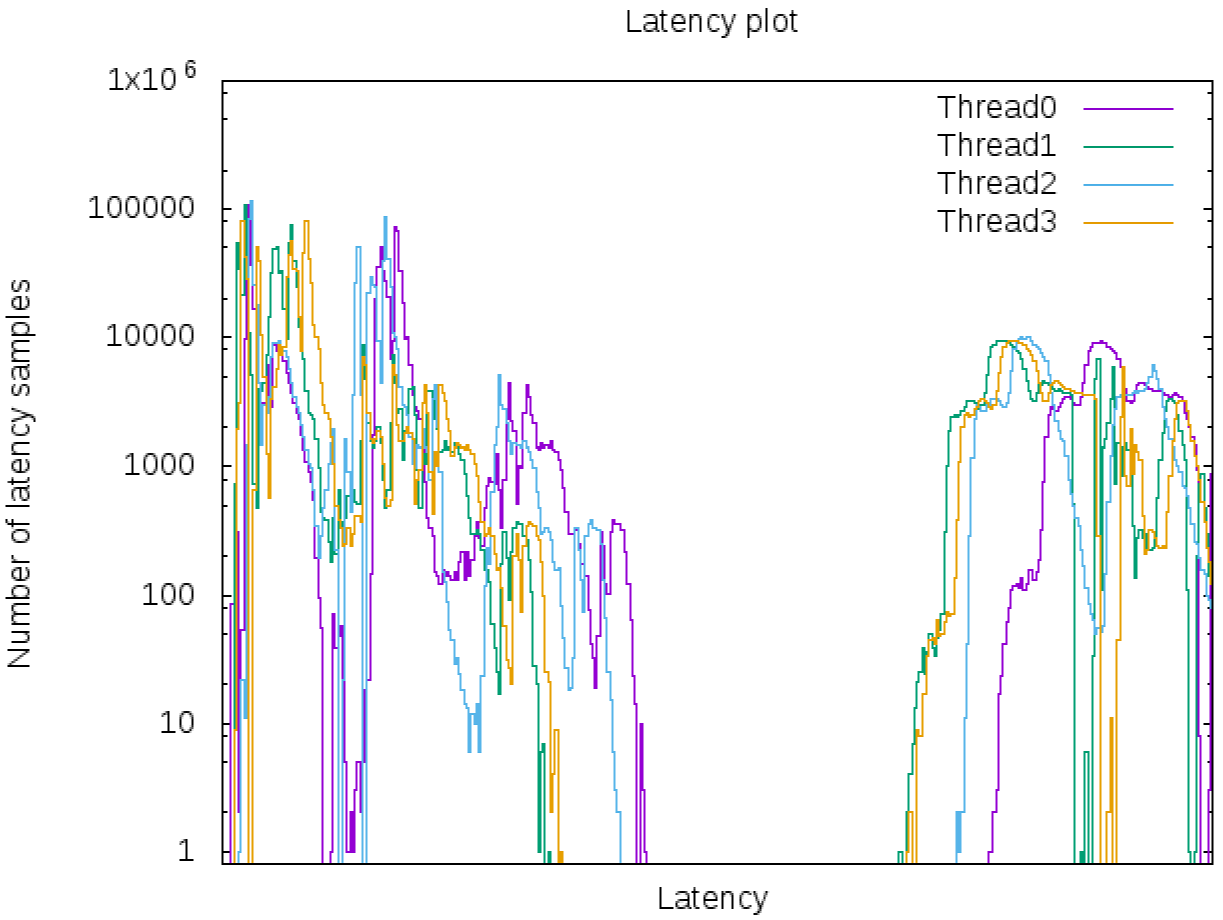
- Deeper the C state, more things get turned off, more state is lost
  - C1, C2 saves power while retaining most of CPU state
  - C3, C6 cache and TLB get flushed
  - C6 also power gated
  - Cache, TLB repopulation time would depend on their state
  - Synchronization activities in kernel add additional variability

- **It would be ok if the latency was consistent. But it is not.**

Good



Bad



Example uses Intel® NUC kit with Intel® Celeron® Processor J3455.
Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Performance varies depending on system configuration.

How to control C state selection?

- 2 attributes of C states can be used to control them

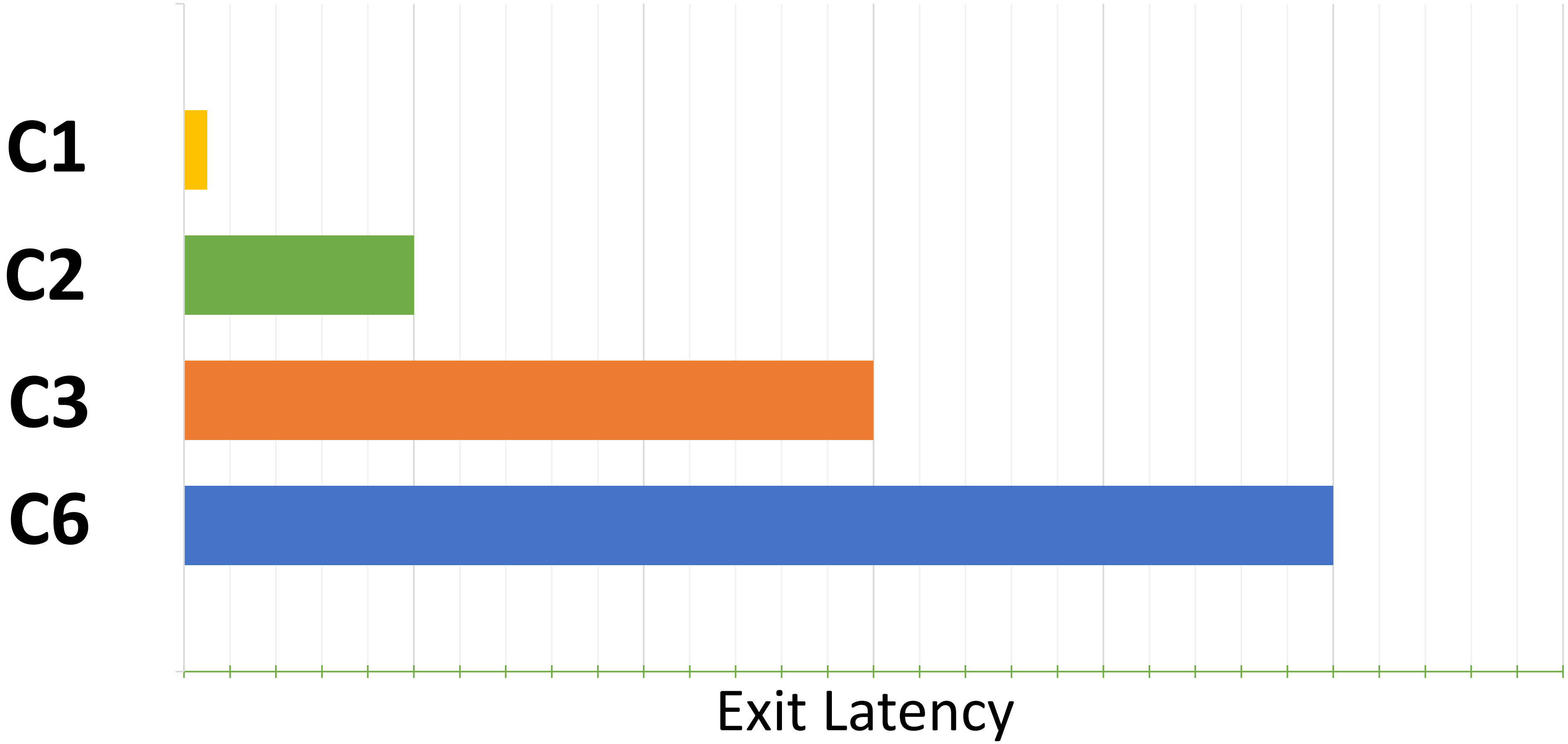
- C state **exit latency**
  - Deeper C states have higher exit latency
- C state **target residency**
  - Deeper C states need to be idle longer to compensate for the energy spent entering and exiting
- Kernel policy (governor) selects C states based on these attributes

2 Methods corresponding to Attributes

- Block C states with higher exit latencies

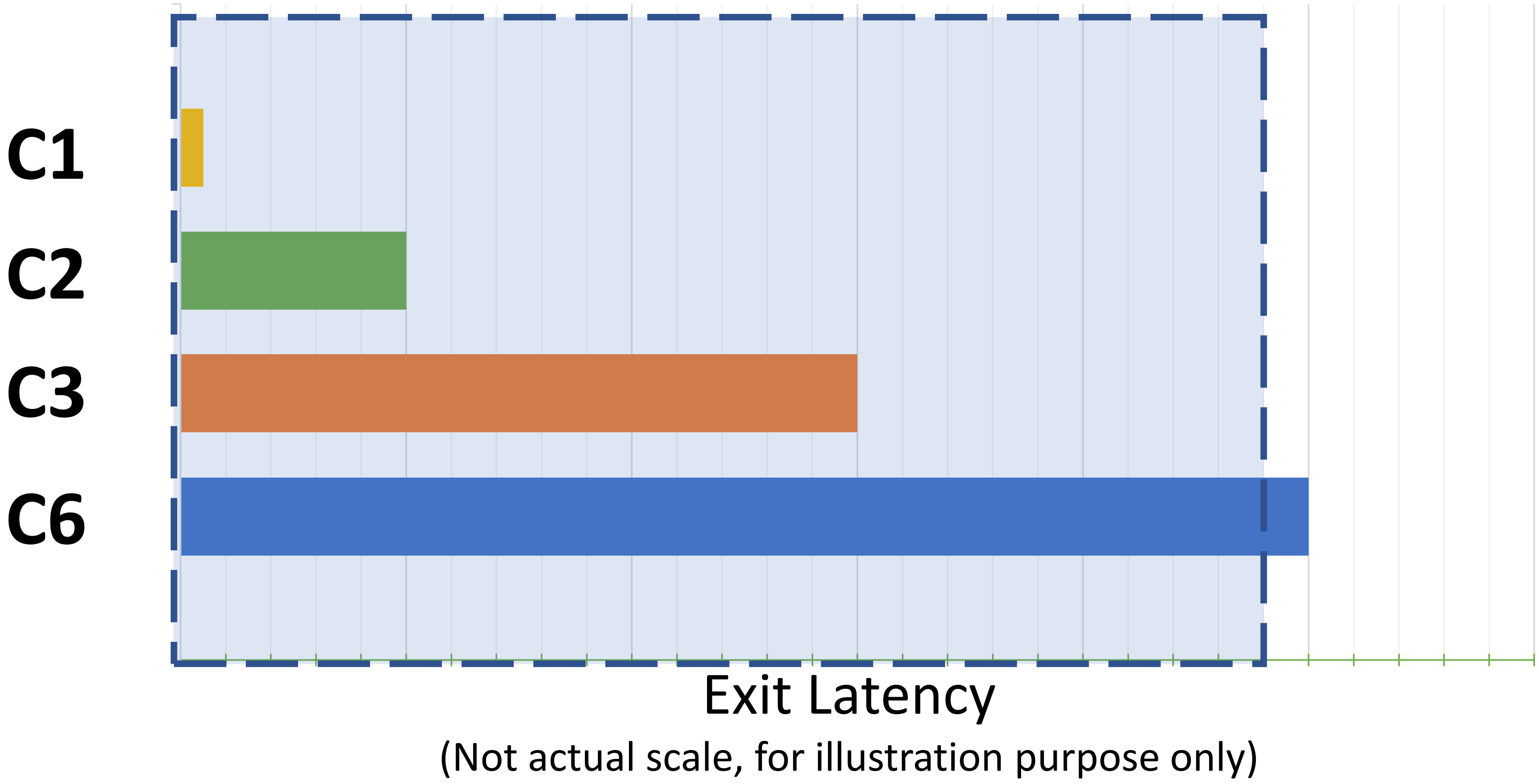
- Block C states with higher target residencies

C State Exit Latencies

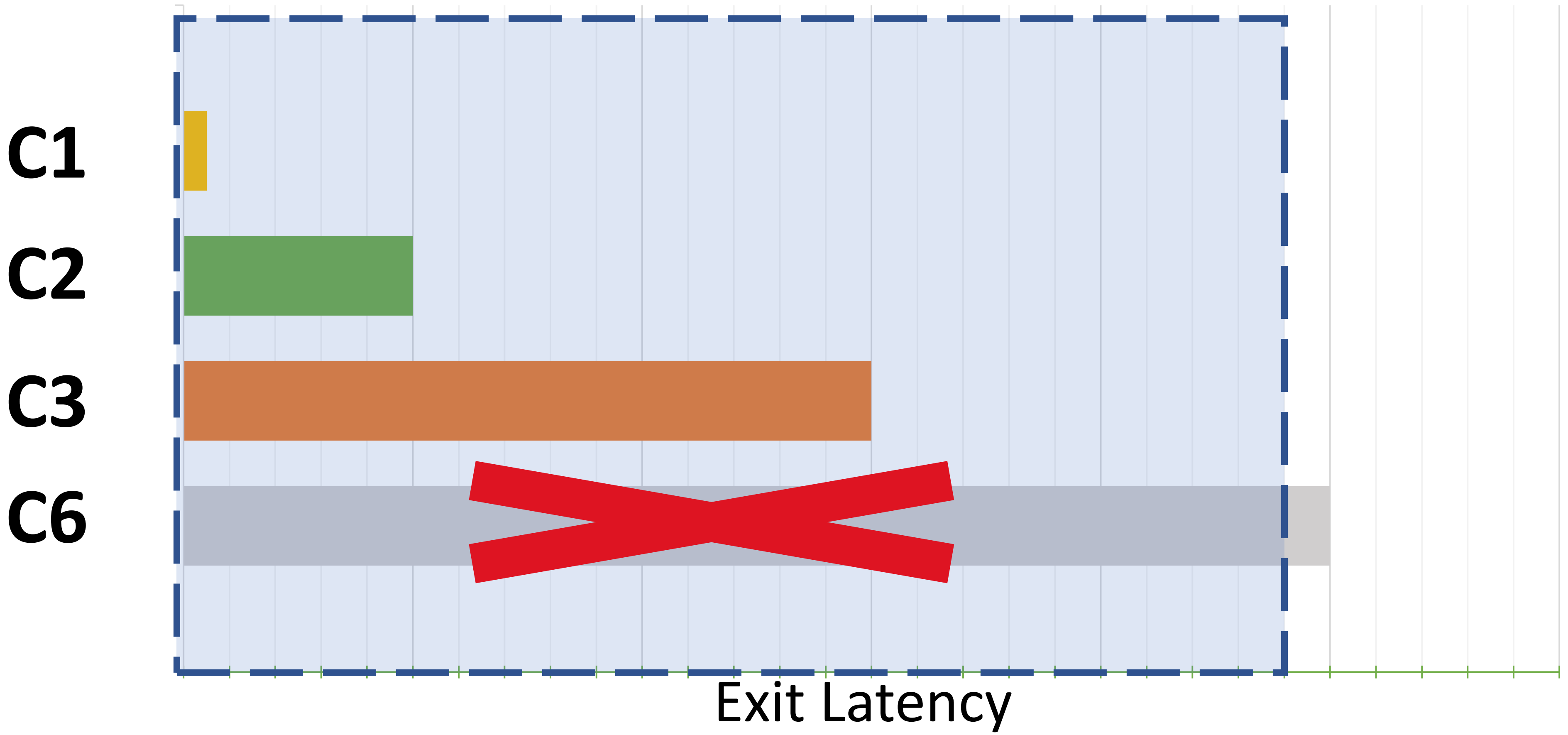


(Not actual scale, for illustration purpose only)

C State Exit Latency Constraint

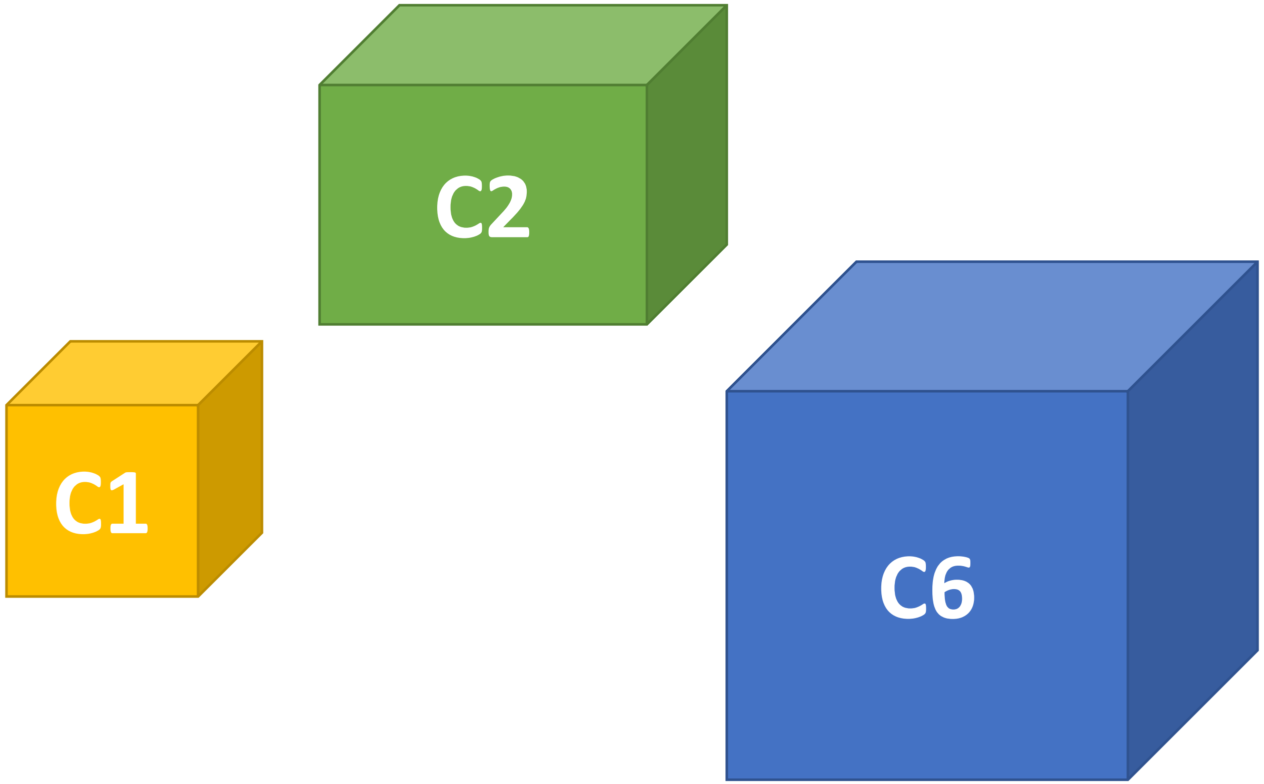


Filter C States by Exit Latencies



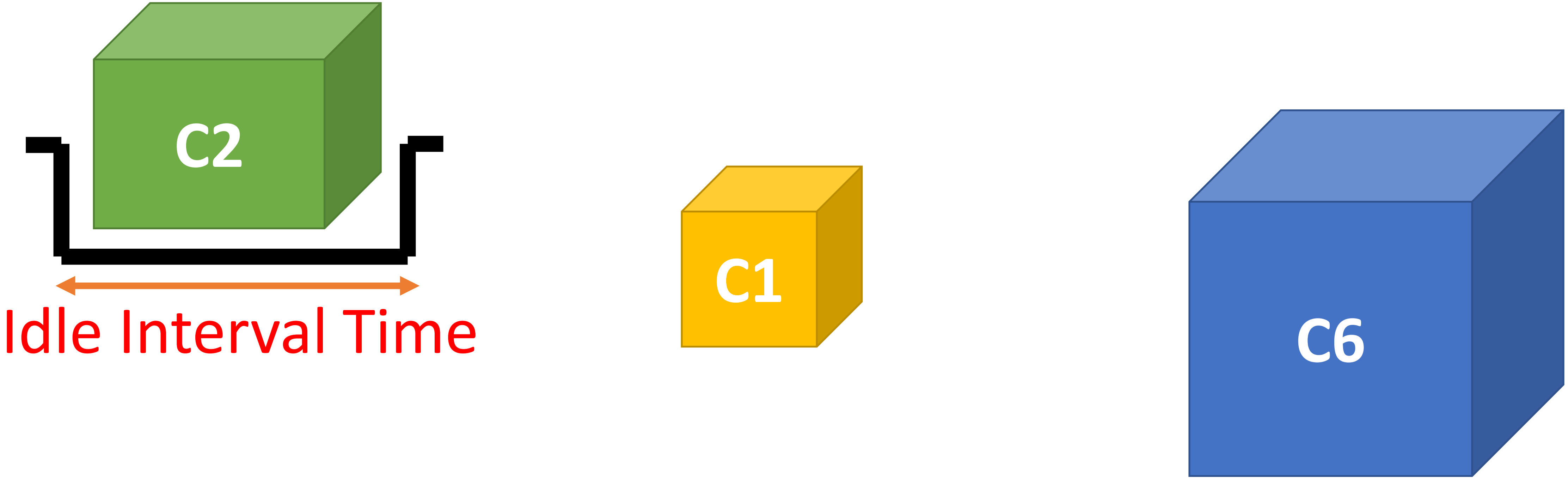
(Not actual scale, for illustration purpose only)

C States Target Residencies



Deeper C state = higher Target Residency

Filter C States by Target Residencies



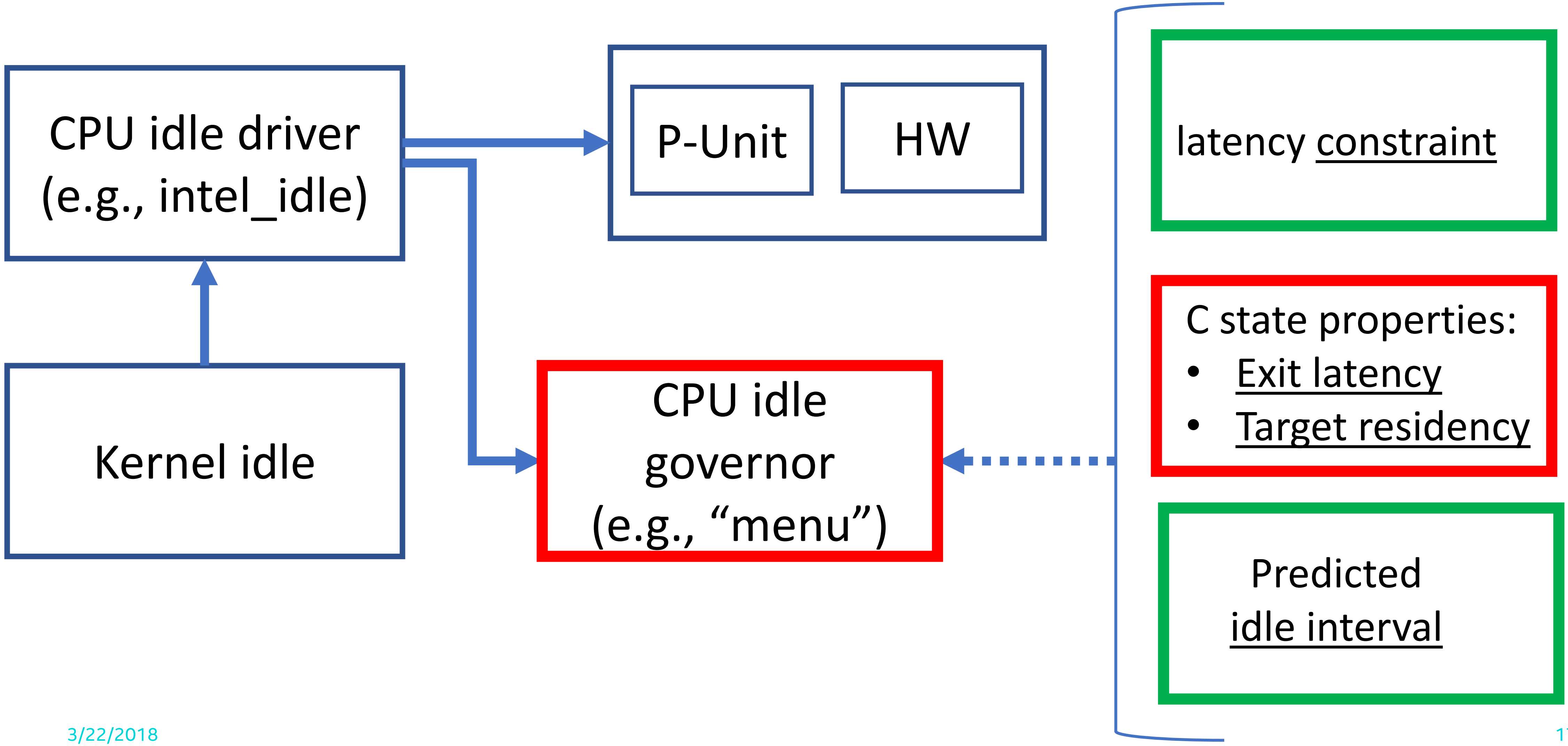
Pick deepest C state with TR that fits idle time

Name the 2 methods

- 1. **SAFE LATENCY CONSTRAINT**
  - Block C states with higher exit latencies

- 2. **SAFE IDLE INTERVAL**
  - Block C states with higher target residencies

C State Selection Policy in Kernel



PM QoS (Quality of Service) Framework

- Allows user to specify a resume latency constraint
- CPU idle governor limits C states with exit latencies lower than the constraint
- Application can change constraint at different phases
- C states can be controlled in each core independently

PM QoS (continued...)

- *Write constraint to /sys/devices/system/cpu/cpuN/power/pm_qos_resume_latency_us*
  - *e.g. $echo 30 > /sys/.../pm_qos_resume_latency_us*

- During critical phases write **"n/a"** to PM QoS disabling all C states.
- At non-critical phases, write **0** to remove all restrictions saving maximum power

- Pull following commits from 4.16 into current RT Linux (4.14)
  - 704d2ce, 0759e80 and c523c68

Recap

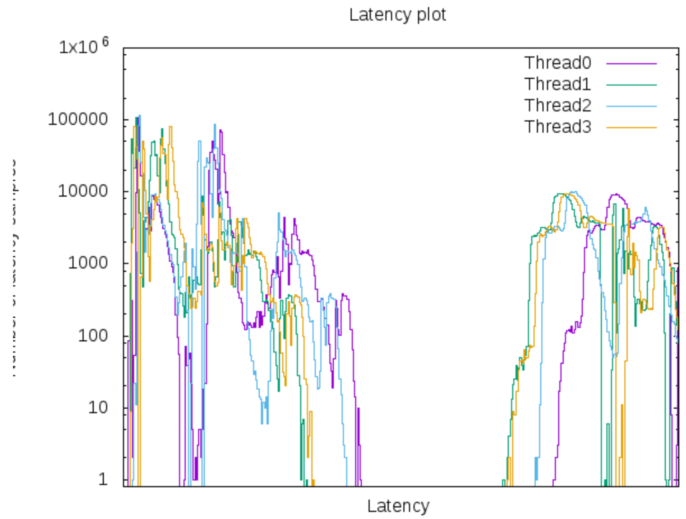
C State Atttributes | **User Controls**

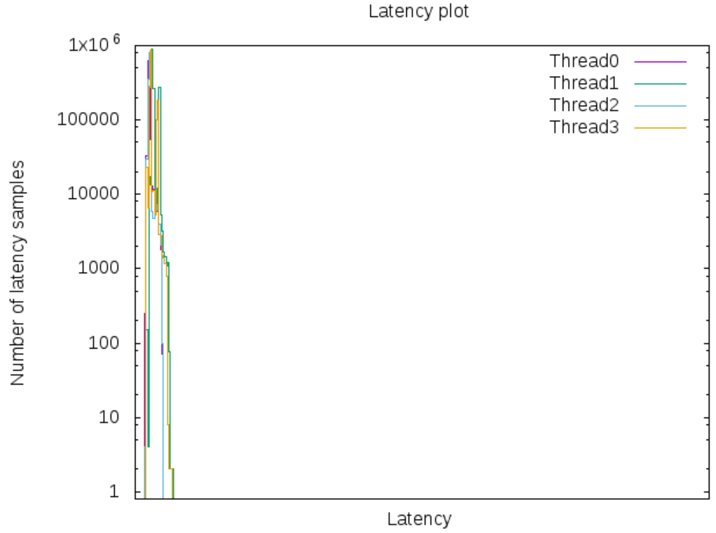
Exit Latency ⇔ Latency Constraint

Target Residency ⇔ Limit Idle Interval

Calibration

1. Find worst-case latency
2. Find Safe Latency Constraint
3. Find Safe Idle Interval





Example uses Intel® NUC kit with Intel® Celeron® Processor J3455.
Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Performance varies depending on system configuration.

Calibrate Worst-case latency

- Use cyclictest with histogram option
  - $cyclictest -a3 -n -q -H1000 -t4 -p80 **-i200** -m –D24h –laptop
- PM QoS constraint set to “n/a” = “no restriction”
- Cyclictest “interval” (-i) option set to high value

Find Safe Latency Constraint

- Use cyclictest with histogram option
  - $cyclictest -a3 -n -q -H1000 -t4 -p80 -i200 -m –D24h –laptop

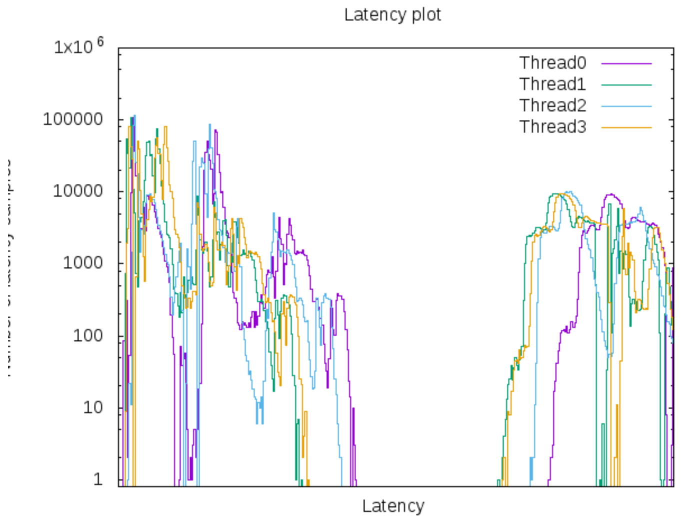
- Calibrate PM QoS constraints until desired latency behavior is achieved

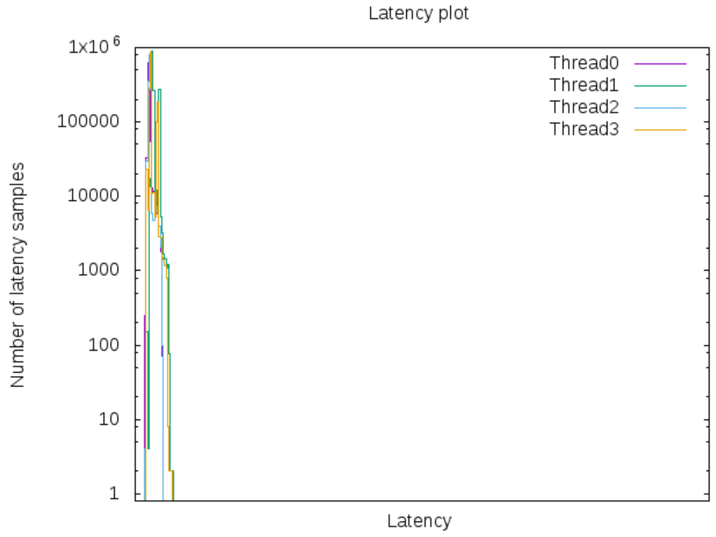
Find Safe Idle Interval

- Calibrate idle interval of cyclictest until desired latency behavior is achieved
  - $cyclictest -a3 -n -q -H1000 -t4 -p80 **–i100** -m –D24h –laptop
- (Set PM QoS to “no restriction” for this calibration)

Example Calibration
(hypothetical numbers)

1. Worst-case latency = 400 us
2. Safe Latency Constraint = 30 us
3. Safe Idle Interval = 100 us





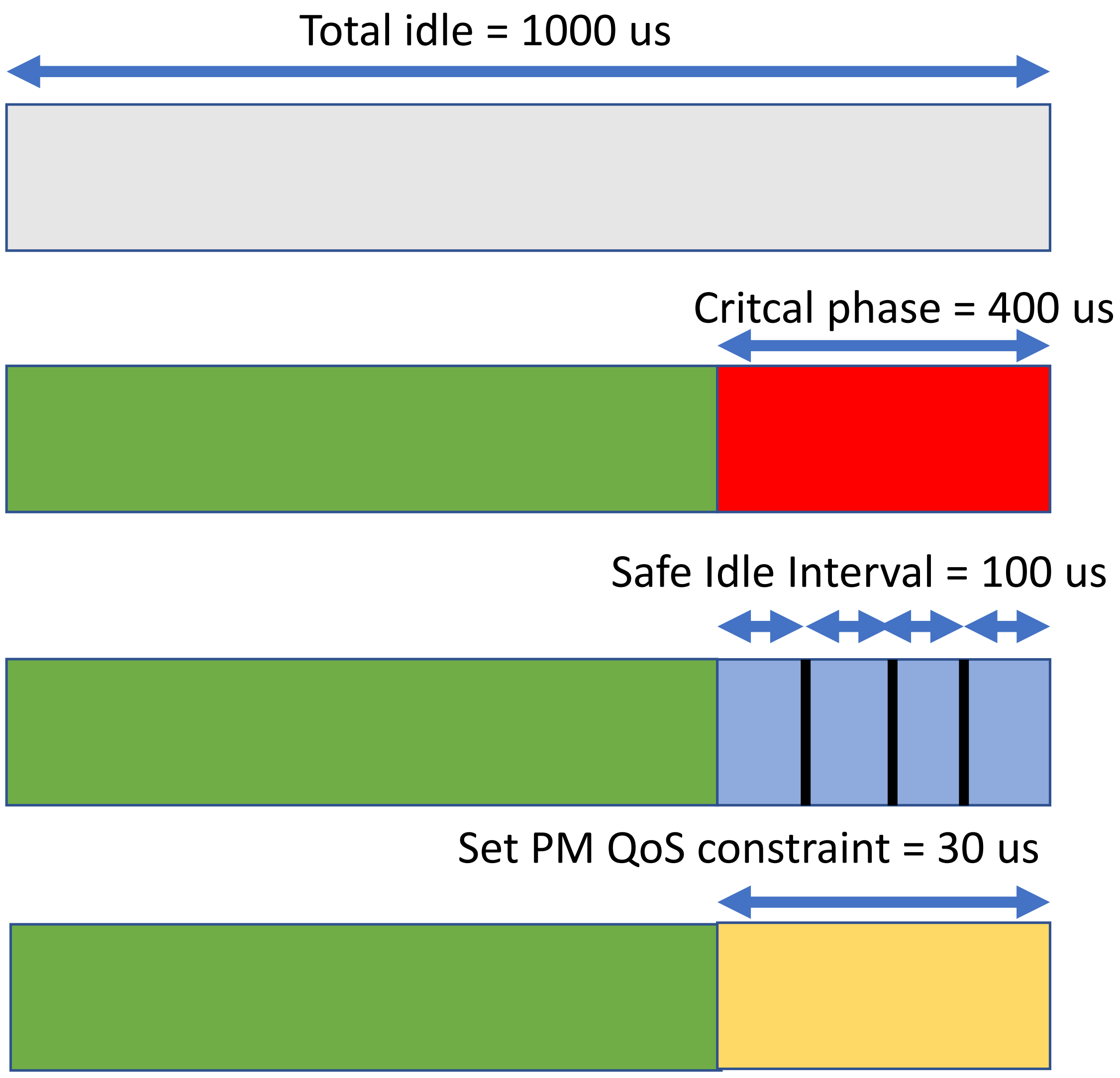
Example uses Intel® NUC kit with Intel® Celeron® Processor J3455.
Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Performance varies depending on system configuration.

Example Tuning

(hypothetical numbers)
Worst-case latency = 400 us
Safe Latency Constraint = 30 us
Safe Idle Interval = 100 us

Deadline = 1000 us

1. Wake 400 us before deadline
2. Prime cache
3. In critical phase use
   - Safe Idle Interval method or
   - Safe Latency Constraint method



Additional Strategies

- CPU topology awareness helps
  - Depth of C state depends on state of units in group
    - Logical processors in core, cores in package
  - Group threads that can go idle at same time
- Prime cache after waking from deeper C states before reaching critical phase
  - Execute code and access data few times
- Fine tune kernel configurations
  - isolcpus, irqaffinity, nohz_full, rcu_nocb, etc.
- Refer wiki for more details https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/cpuidle

Key Takeaways

- Methods do not compromise Real-time constraints
- Provides flexibility, variety and degree of options
- Uses available tools and infrastructure
- Scalable and can be easily included early in application design

References

- Linux Foundation Wiki https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/cpuidle
- Kernel parameters https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt
- Kernel scheduling ticks https://www.kernel.org/doc/Documentation/timers/NO_HZ.txt
- PM QoS https://www.kernel.org/doc/Documentation/power/pm_qos_interface.txt
- Cyclictest https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest
- Reducing OS jitter https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/kernel-per-CPU-kthreads.txt?h=v4.14-rc2
- Good reference for C states https://books.google.com/books?id=DFAnCgAAQBAJ&pg=PA177&lpg=PA177&dq=c+state+latency+MSR&source=bl&ots=NLTLrtN4JJ&sig=1ReyBgj1Ej0_m6r6O8wShEtK4FU&hl=en&sa=X&ved=0ahUKEwifn4yI08vZAhUFwVQKHW1nDgIQ6AEIZzAH#v=onepage&q=c%20state%20latency%20MSR&f=false

Thank you.

Optimal Kernel Boot Parameters

- isolcpus= cpu list. Give the list of critical cores. Isolates the critical cores at user level.
- irqaffinity=cpu list. Give list of non-critical cores. This will protect the critical cores from IRQs.
- rcu_nocbs=cpu list. Give the list of critical cores. This stops RCU callbacks from getting called into the critical cores.
- nohz=off. The kernel's "dynamic ticks" mode of managing scheduling-clock ticks is known to impact latencies while exiting CPU idle states. This option turns that mode off. Refer to https://www.kernel.org/doc/Documentation/timers/NO_HZ.txt for more information about this setting.
- nohz_full=cpu list. Give the list of critical cores. This will enable "adaptive ticks" mode of managing scheduling-clock ticks. The cores in the list will not get scheduling-clock ticks if there is only a single task running or if the core is idle. The kernel should be built with either the CONFIG_NO_HZ_FULL_ALL or CONFIG_NO_HZ_FULL options enabled.

PM QoS New Commits

- If working on RT 4.14, pull in following commits from 4.16
  - 704d2ce, 0759e80 and c523c68
-
- $git format-patch -1 <commit>
- $git apply --reject <patch>
- Apply in order. May need to resolve some rejects

Priming Cache

- Priming cache refers to forcing population of CPU cache with code and data that needs consistent access times

- Before reaching the point where jitter is to be avoided, in preparation, execute/access critical code/data causing them to get loaded in the cache.

$turbostat --debug

Useful tools

- cyclictest
  - Measure latencies. Sleeps for a specified interval and compares that interval with actual time spent in sleep. The difference is the latency. Generates histogram which can be used with a plotting tool.
  - Run with –laptop option. By default it disables C states using PM QoS
- turbostat
  - Gives C state utilization by cores.
  - $turbostat –debug
- powertop
  - Shows power consumption details
- gnuplot
  - Plotting tool.

Disclaimers