REAL-TIME CONTROL IN REDHAWK AND ROS 2.0

大島龍博 足立一希
プロフェッショナルサービス部
コンカレント日本株式会社 Concurrent Real-Time Inc of Japan

プロフィール

Concurrent Real-Time社

- 米国フロリダ本社は1966年創立
- 世界のリアルタイムコンピュータ技術をリードしてきた会社。
- 航空/宇宙/防衛/自動車/ロボットなどのリアルタイム分野にRedHawk RealTime Linux を提供。

https://www.concurrent-rt.com
tatuhiro.oshima@concurrent-rt.com
https://www.concurrent-rt.co.jp
oshima@concurrent-rt.co.jp

- コンカレント日本株式会社
  - 1986年創立Concurrent Real-Time 日本支社
- プロフェッショナルサービス部　部長
  - 前職 九工大･情報工学教室 文部技官
  - 情報処理学会正会員
  - 入社以来29年間リアルタイム畑
  - 専門
    - 並列計算機とリアルタイムOS
    - ドライバ開発と移植
    - リアルタイムよろず相談
- ROSとの関わり
  - RedHawkがARM64/Nvidia Jetsonをサポートしてから

アジェンダ

- 目的
  - RTOSがROS開発者に対してどのような利益が得られるかを具体的に提示
- 入手可能な以下のホワイトペーパーの概要と追試
  - “Real-time control in ROS and ROS 2.0”
  - “Threaded IRQs on Linux PREEMPT-RT”
  - “Towards a distributed and real-time framework for robots”
  - “Using ROS with RedHawk Linux on the NVIDIA Jetson TX2”
- 入手可能な以下のUbuntuベースOSの比較
  - X86_64(注) , ARM64(Nvidia Jetson AGX Xavier/Tx2)
  - “ピュアUbuntu18.04” vs “PREEMPT_RT” vs “RedHawk” vs “Linux for Tegra”

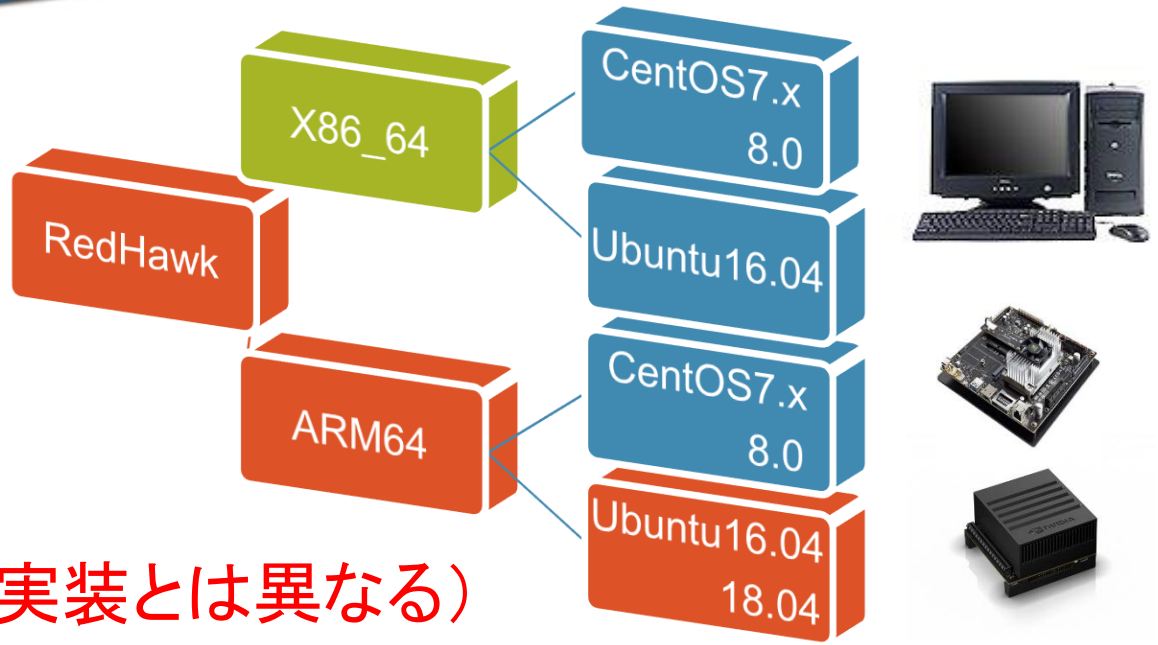
注：Jetson Tegraとほぼ同じ性能のCPU(Windows7世代)を使っています。

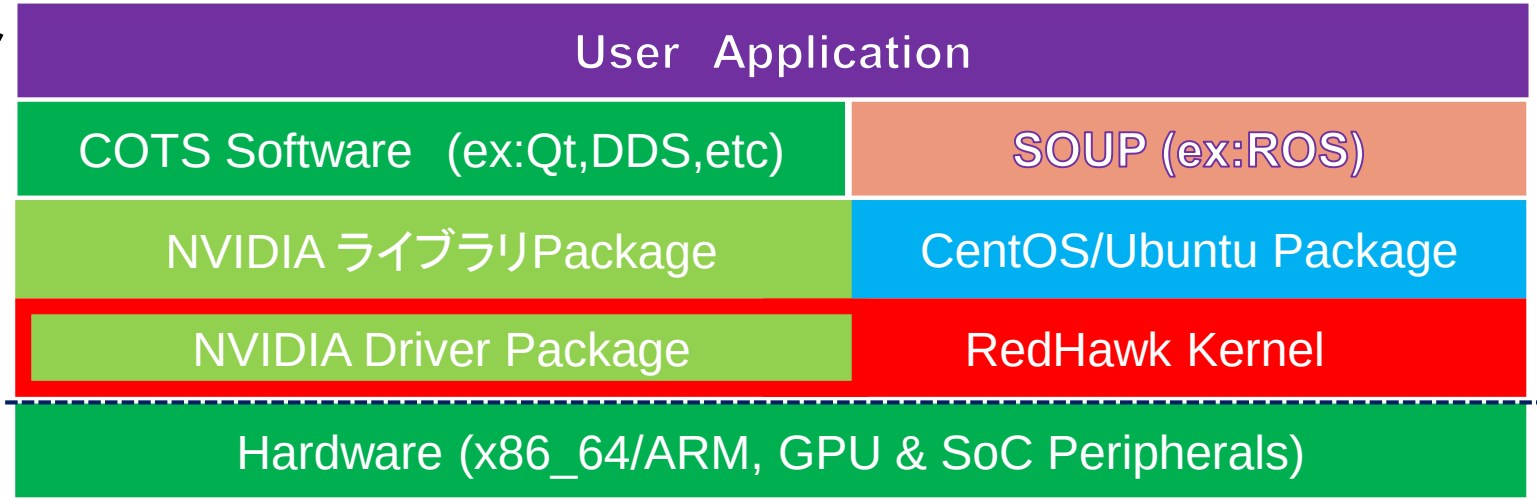
ROS2をサポートする有償RTOS

- **RedHawk**
  - https://www.concurrent-rt.com/wp-content/uploads/2016/09/Using-ROS-with-RedHawk-on-Jetson-TX2.pdf
- QNX
  - http://blackberry.qnx.com/en/articles/what-adas-market-needs-now
- eSOL
  - https://www.esol.com/embedded/ros.html
- Vxworks
  - https://github.com/Wind-River/vxworks7-ros2-build
- Green Hills Software
  - https://www.ghs.com/news/20190408_japanIT_ROS_support.html

RedHawkとは

- 商用のハードリアルタイムLinuxOS
  - サーバとエンベデッド
- X86_64とARM64をサポート
- CentOSまたはUbuntuとバイナリ互換
- POSIX-REALTIME-APIでプログラム
- オープンソースのシングルカーネル(Xenomi実装とは異なる)
- Nvidiaボード/ドライバをサポート
- Shieldedテクノロジー
  - SkyLake PDL 5μ秒
  - Jetson PDL 100μ秒
- apt installでROSが動作





リアルタイムFAQ

Q.ハードウェアが早くなれば、リアルタイム？

A.いいえ
　リアルタイムシステムは、要求される時間内に必ず応答を返さなくてはなりません。
　そのためにリアルタイムOSは、カーネルに必要な改修を行っています。
　Linuxでは、kernel.orgの標準カーネルに対して、SMPパッチ、PREEMPTパッチ、PRREMT_RTパッチなどを適用することで、Linuxのリアルタイム化を行います。

Q.PDLとはなんですか？

A.Process　Dispatch Latency とよばれ、割り込み待ちのユーザプロセスが実行開始されるまでの時間です。

Q.ハードリアルタイムとソフトリアルタイムは何が違う？

A.ハードリアルタイムは、応答性の保証があり、ソフトリアルタイムは、応答性の保証がありません。
その保証は、OSベンダーが行います。
したがってハードリアルタイムOSベンダーは、OSの設計値を示しているのではなく、ハードウェアを含めた性能試験を行い、保証値を公表しています。

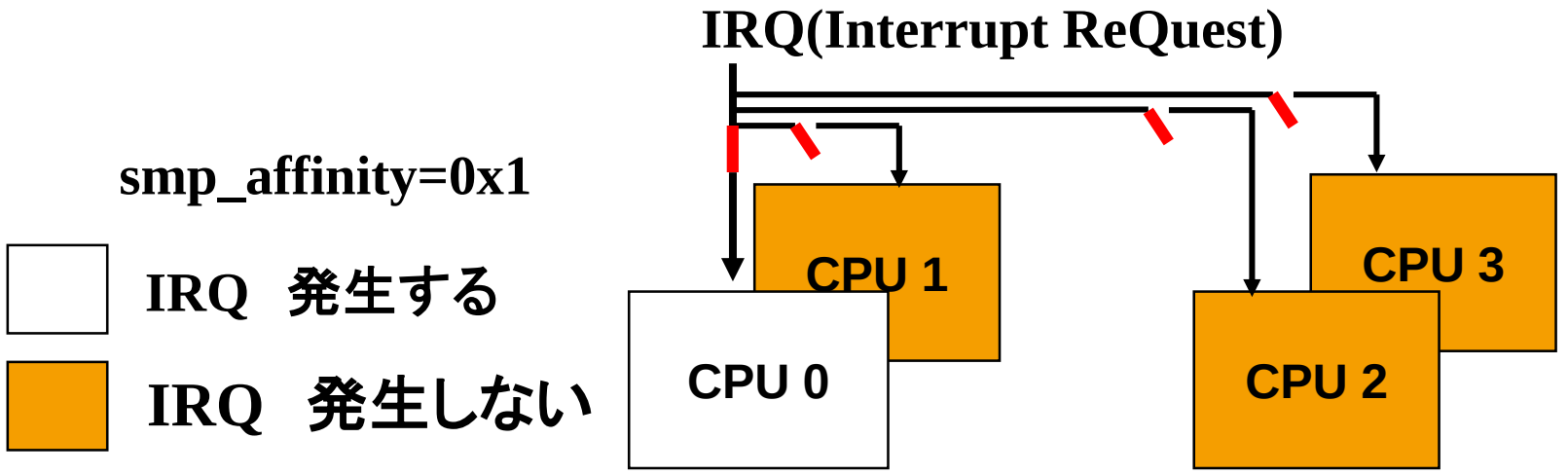
→ Board Support Package

システムのトータルジッターは、
保証値×システムコールの回数
になるため保証値は重要です。

POSIX Real Time API

- Semaphores
- Real-Time Signals
- Shared memory
- Memory locking
- Synchronous I/O
- Asynchronous I/O
- Execution scheduling
- Clocks and timers
- Message Queue

- POSIX-APIで定義されていないもの
  - 割り込みのコア制御
    - /proc/irq/*/smp_affinity
  - プロセスのコア割り当て
  - プロセス優先度
  - スケジュールポリシー



ROS1 / ROS2 ユースケース

ROS1

- 単一のロボット
- ワークステーションクラスのコンピュータ
- リアルタイム制御は求めない
  - ros_controlの特別な仕組みで別途対応
- 優れたネットワーク接続性
  - 有線または近接の高帯域幅ワイヤレス
- 主に学問分野の研究応用

ROS2

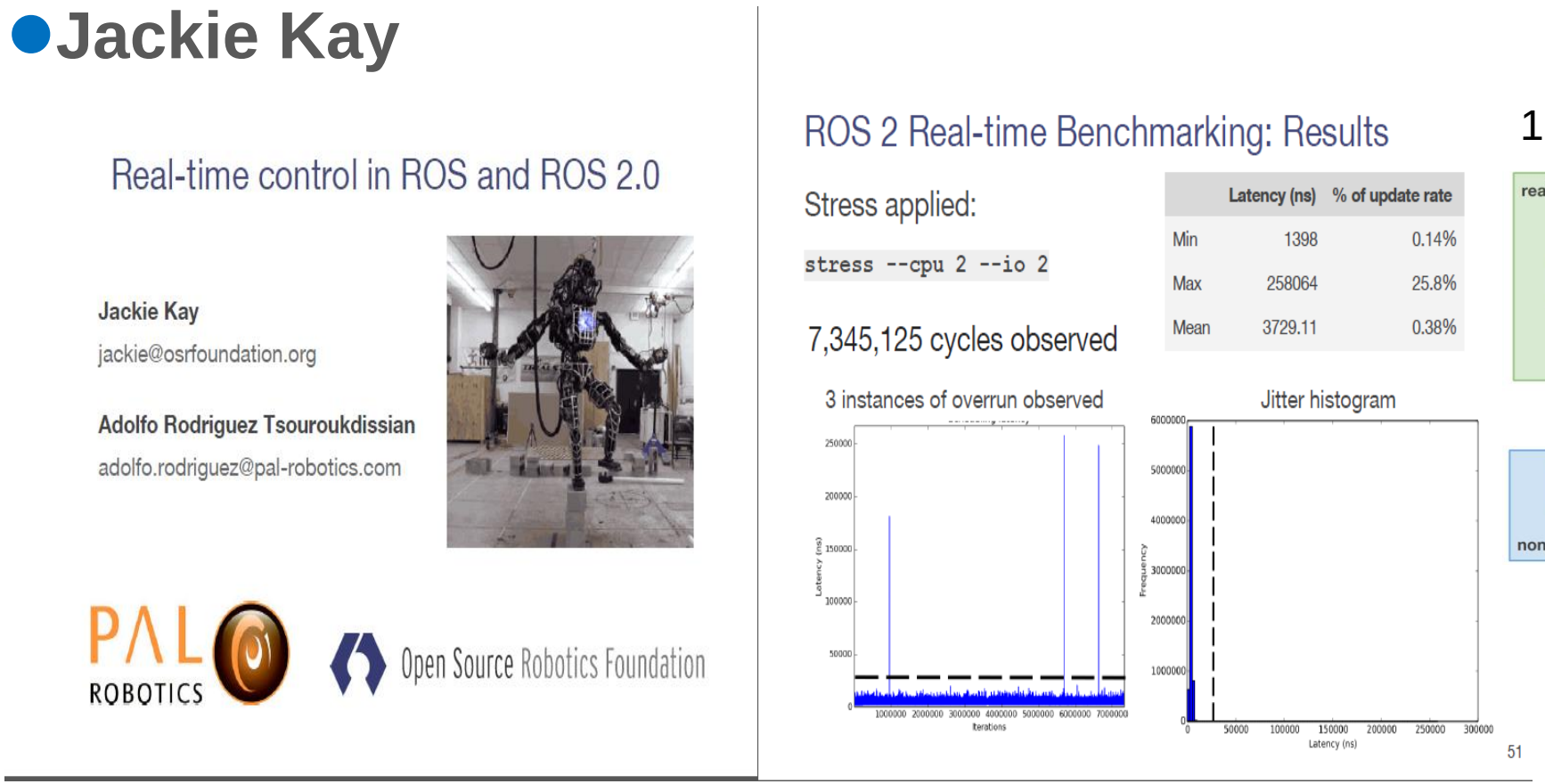
- 複数ロボットのチーム
- 小型の組み込みプラットフォーム
- リアルタイムシステム
  - プロセス間やマシン間の通信を含む、ROSで直接リアルタイム制御をサポート
- 非理想的なネットワーク
  - 低品質のWiFiや地上間通信リンクで、損失や遅延のためにネットワーク接続が劣化した場合でも、ROSを動作させる
- プロダクション環境
  - ROSが研究室で選択され続けるプラットフォームであり続けることは不可欠だが、ROSベースのラボプロトタイプが現実のアプリケーションでの使用に適したROSベースの製品に進化できる。
- プロジェクト範囲の縮小
  - 積極的に外部のオープンソースやCOTSプロダクトを取り入れる

注意：
ROS2はリアルタイムOSではありません！

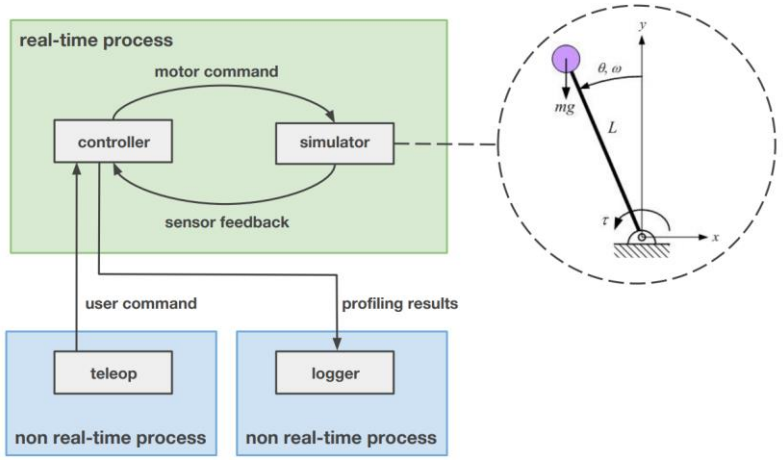
⇒ROS2では、制御と通信のリアルタイム性が重要

Real-time control in ROS and ROS 2.0

- **Jackie Kay**



12スレッド

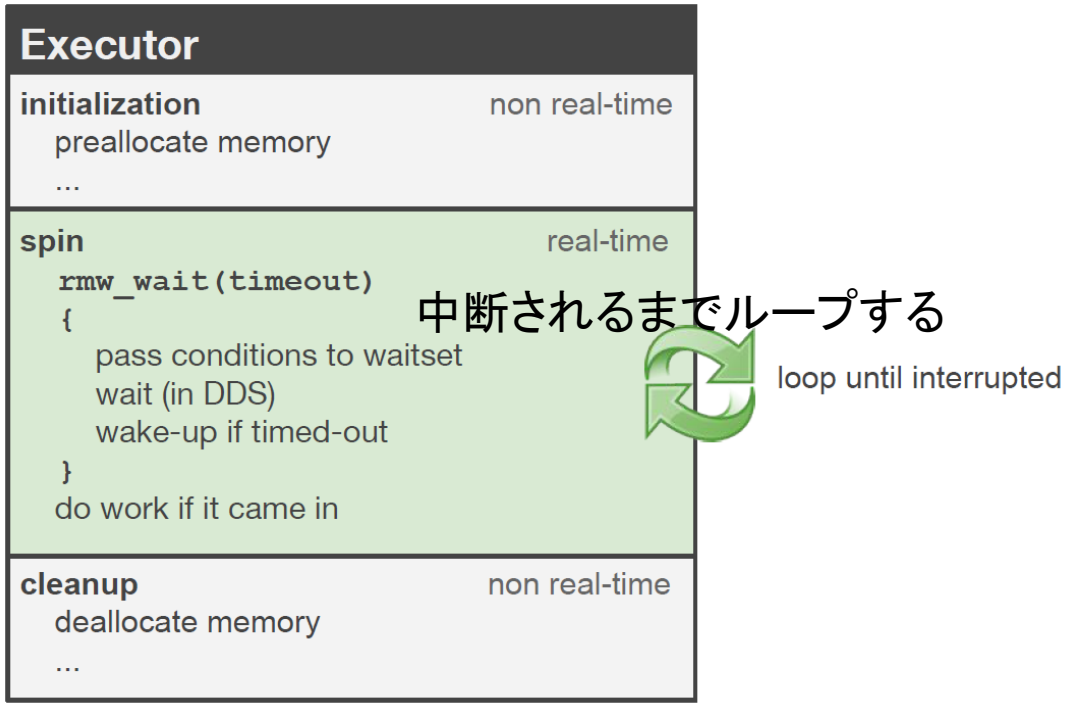


目標
1 kHz更新ループ（1 ms周期）
3%未満のジッター（30μs）

https://roscon.ros.org/2015/presentations/RealtimeROS2.pdf

ROS 2のリアルタイムプログラミング

- 公式ドキュメントに記述されているけれど！
- 1は、OS技術に依存
- 2は、適用するユースケース(応答時間)に依存
- 3は、プログラム技術に依存



1. リアルタイムのコンピューターシステムを作成するには、リアルタイムループを定期的に更新して期限に間に合わせる必要があります。
2. これらの締め切りのわずかな誤差（最大許容ジッタ）しか許容できません。
3. これを行うには、ページフォールトイベント、動的メモリの割り当て/割り当て解除、無期限にブロックする同期プリミティブなど、実行パスでの非決定的な操作を回避する必要があります。
4. リアルタイムコンピューティングで一般的に解決される制御問題の典型的な例は、倒立振子のバランス調整です。
5. コントローラーが予想外に長い時間ブロックされた場合、振り子が落ちるか不安定になります。ただし、振り子を制御するモーターが動作できる速度よりも速い速度でコントローラーが確実に更新されると、振り子はセンサーデータに反応して振り子のバランスを取ることに成功します。

- https://index.ros.org//doc/ros2/Tutorials/Real-Time-Programming/

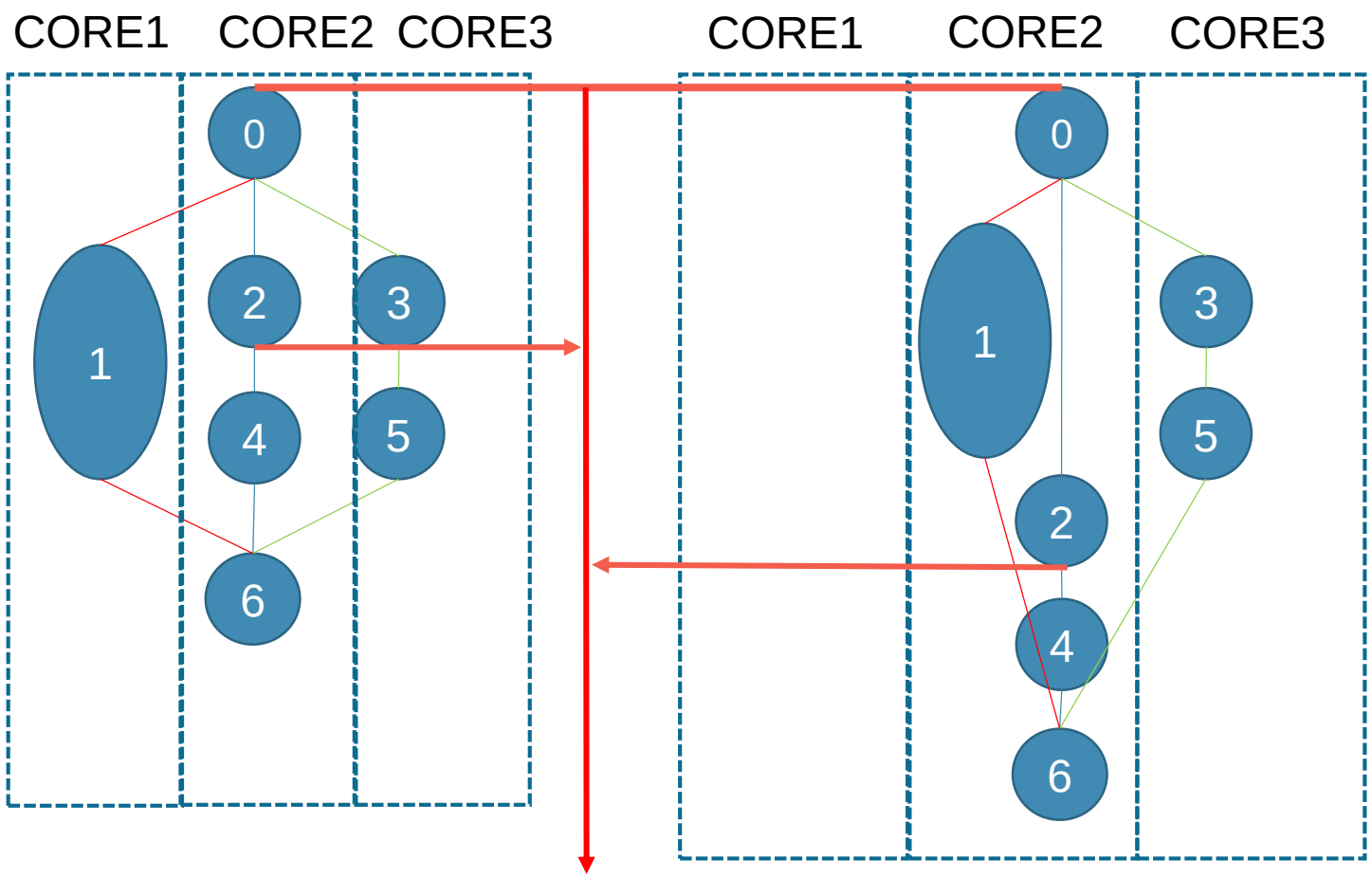
公式ドキュメントの意味

- ページフォールトイベント
- 動的メモリの割り当て/割り当て解除
- 無期限にブロックする同期プリミティブなど、
- 実行パスでの非決定的な操作を回避
  - いつでも同じ速度で実行されるのが最善ですが、最悪値で設計するのがセオリー

データミニステック

- メモリロックしなさい
- C++では、関数呼び出しごとに発生
- デッドロックの回避
  - プロセスの優先度とセマフォ
- 優先度設定(SCHED_RR,SCHED_FIFO)
- タイムアウト付き関数を利用
  - エラーになったらどうする？
- ノードの粒度（プログラムの大きさ）をそろえてSCHED_RR
- SCHED_RR はnice値で、クォンタムタイムが変化

スケジューリングとタイムクオンタム



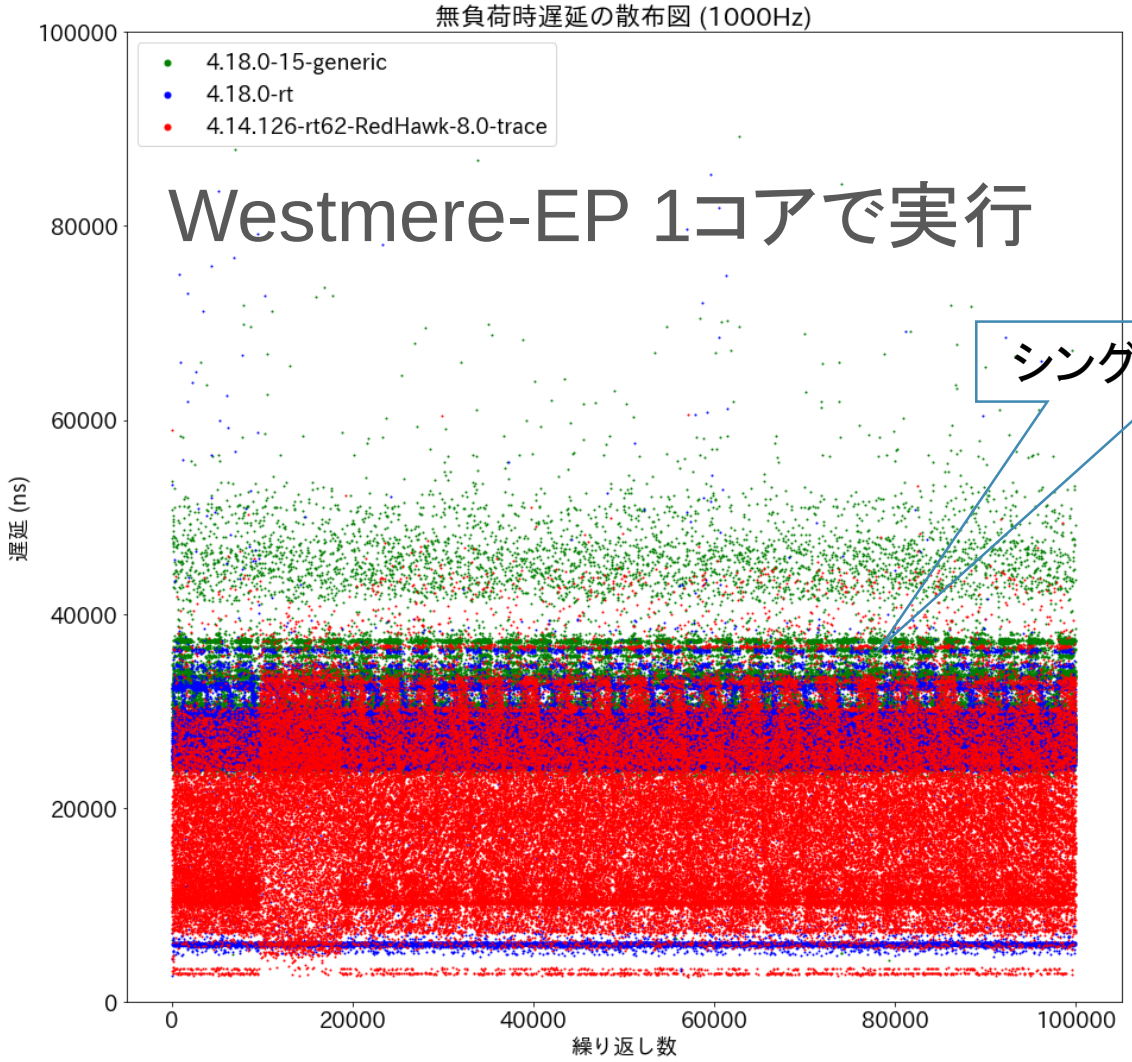
Nice値=>タイムクオンタム
-20 => 800ms
-19 => 780ms
-18 => 760ms
-17 => 740ms
-16 => 720ms
-15 => 700ms
-14 => 680ms
-13 => 660ms
-12 => 640ms
-11 => 620ms
-10 => 600ms
-9 => 580ms
-8 => 560ms
-7 => 540ms
-6 => 520ms
-5 => 500ms
-4 => 480ms
-3 => 460ms
-2 => 440ms
-1 => 420ms
0 => 100ms

Nice値=>タイムクオンタム		
	TX2	Xavier
0 =>	100ms	
1 =>	90ms	92ms
2 =>	90ms	88ms
3 =>	80ms	84ms
4 =>	80ms	
5 =>	70ms	72ms
6 =>	70ms	68ms
7 =>	60ms	64ms
8 =>	60ms	60ms
9 =>	50ms	52ms
10 =>	50ms	48ms
11 =>	40ms	44ms
12 =>	40ms	40ms
13 =>	30ms	32ms
14 =>	30ms	28ms
15 =>	20ms	24ms
16 =>	20ms	
17 =>	10ms	12ms
18 =>	10ms	8ms
19 =>	10ms	4ms

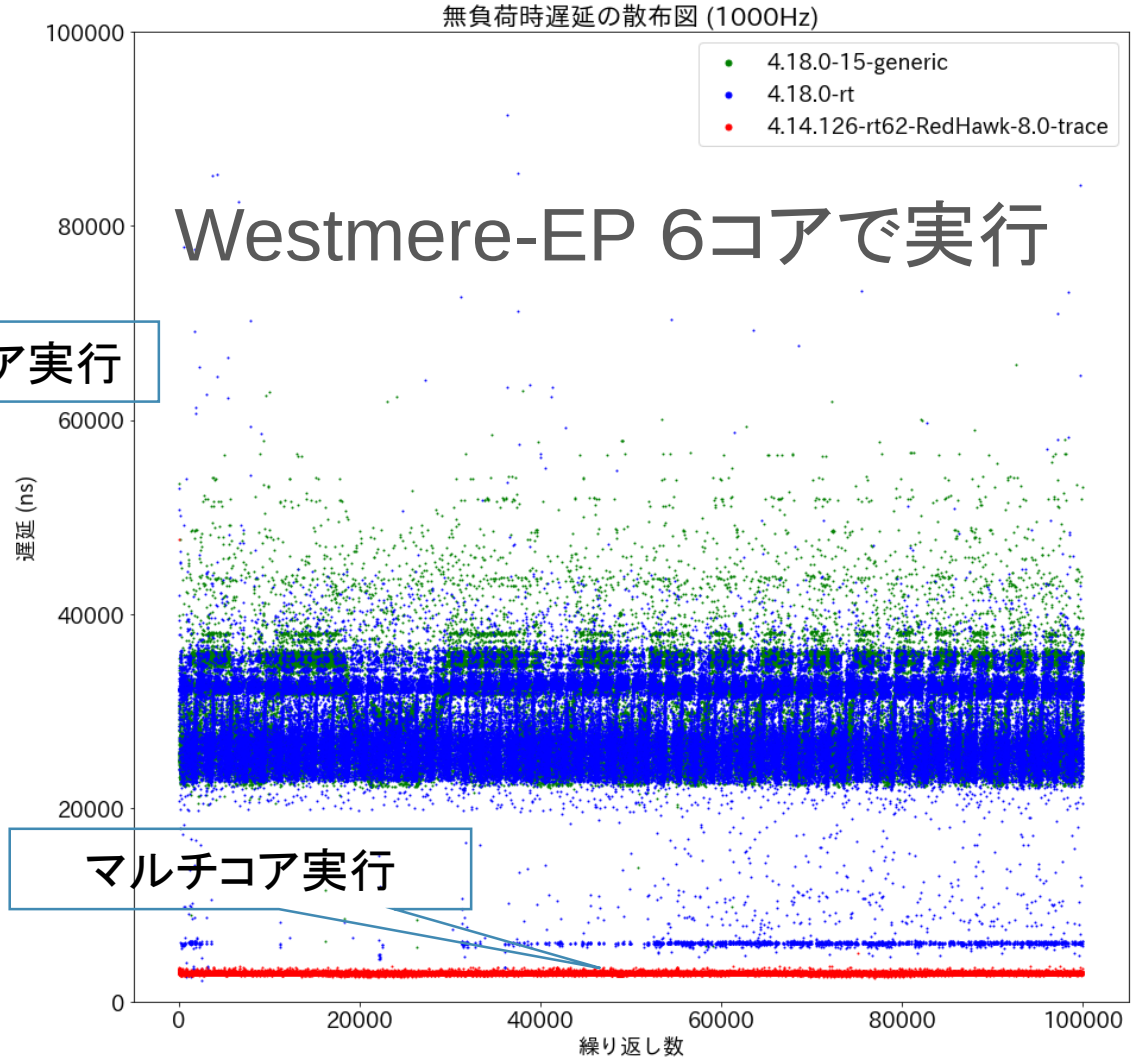
1，2，3が同時にアクティブになっても
1のCPU割り当てで、2の応答時間が変化する

x86ジッター評価(無負荷:トレンド)

X86_64



シングルコア実行



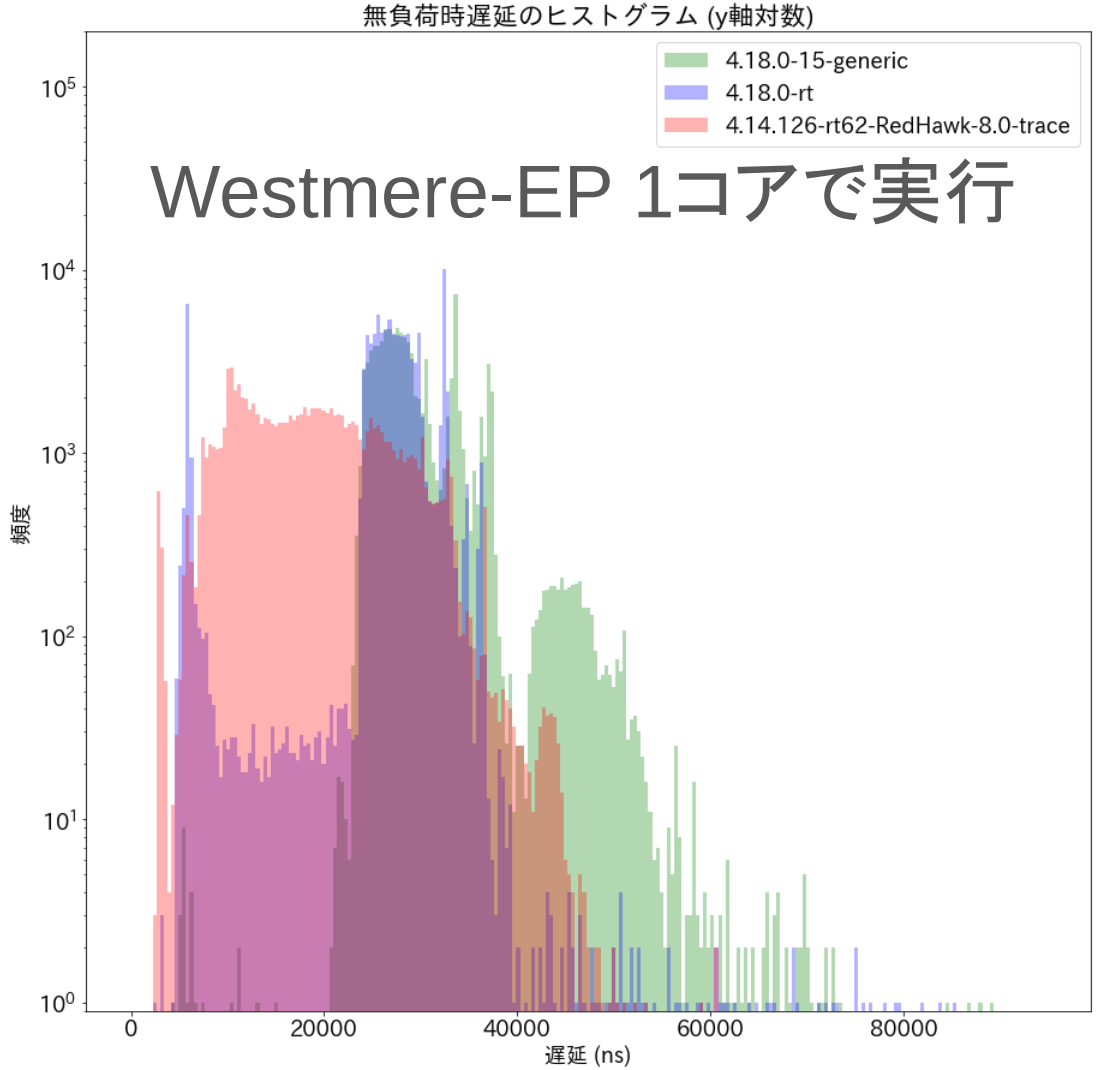
マルチコア実行

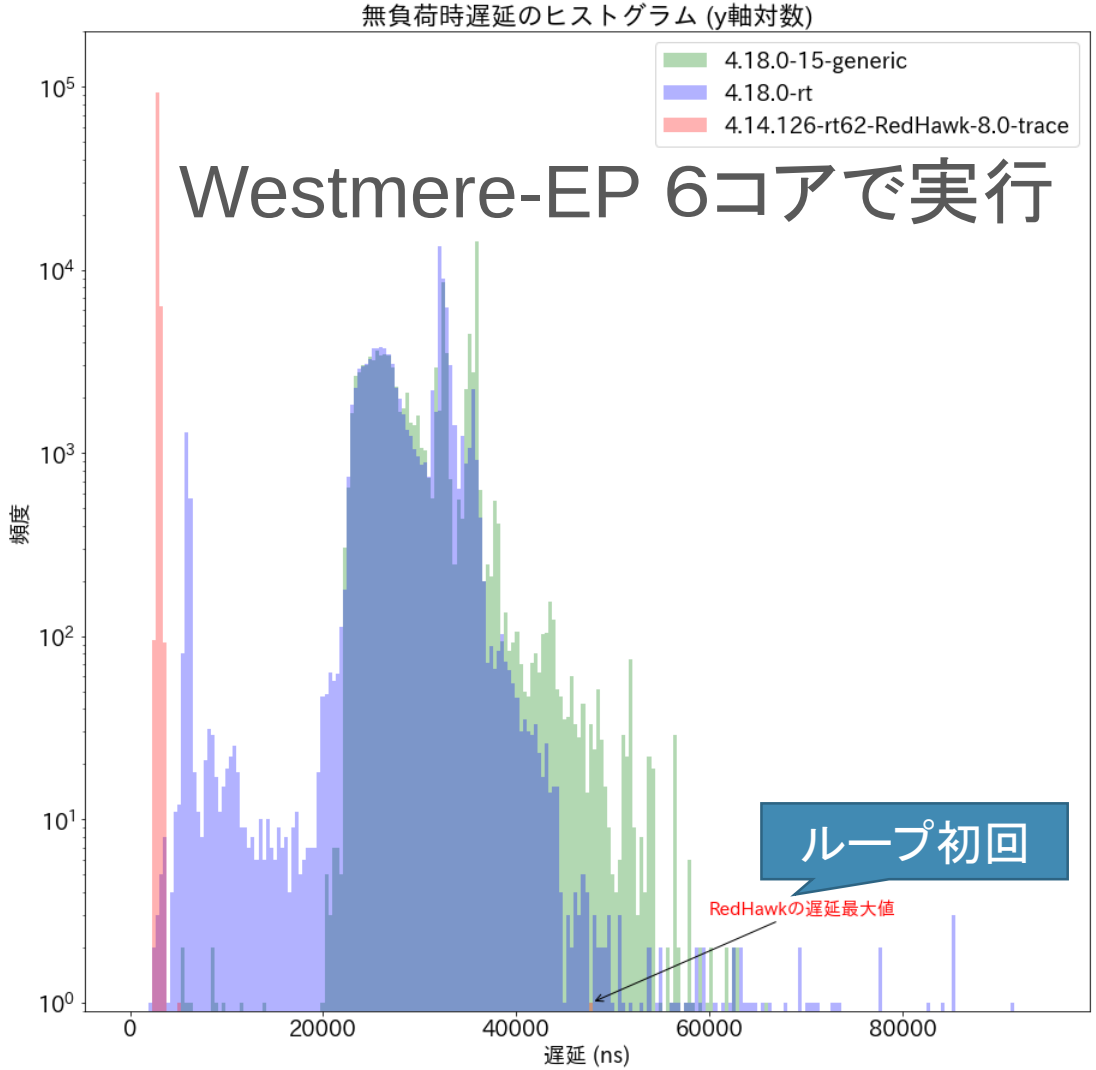
RedHawk

concurrent REAL-TIME

x86ジッター評価(無負荷:ヒストグラム)

X86_64





リアルタイム設定（RedHawk)

```
$ sudo -s
# shield -a 6-11
# run -a | grep kworker | awk '{print "run -s rr -P 98 -n", $9}' | sh
# run -b0 -u0
# run -b0 -g0
# source /usr/local/cnc/ros/dashing/setup.bash
```

※負荷テスト時には以下の1行を実行

```
# for i in {0..5}; do run -b ${i} stress --cpu 1 --io 1 --hdd 1 & done
# run -b 6-11 -s fifo -P 90 pendulum_demo -i 100000 -f pendulum_demo.log
```

12コア中6コアをリアルタイム用に予約

kworkerの優先度を上昇

移動可能なデーモンをコア0に移動

注：RedHawkのshieldは、システムクロック割り込みも停止します。

リアルタイム設定(Ubuntu)

※/etc/default/grubのGRUB_CMDLINE_LINUX_DEFAULTに"isolcpus=6-11"を追加

```
$ sudo update-grub
$ reboot
$ sudo -s
# for i in `ls /proc/irq/*/smp_affinity | grep -v "/0/" | grep -v "/2/"`
# do echo ffffffff,fffff03f > $i ; done
# ps -eo pid,comm | grep kworker | awk '{print "chrt -r -p 98 "$1}' | sh
# source /usr/local/cnc/ros/dashing/setup.bash
```

12コア中6コアをリアルタイム用に予約

移動可能なデーモンをコア0に移動

kworkerの優先度を上昇

※負荷テスト時には以下の1行を実行

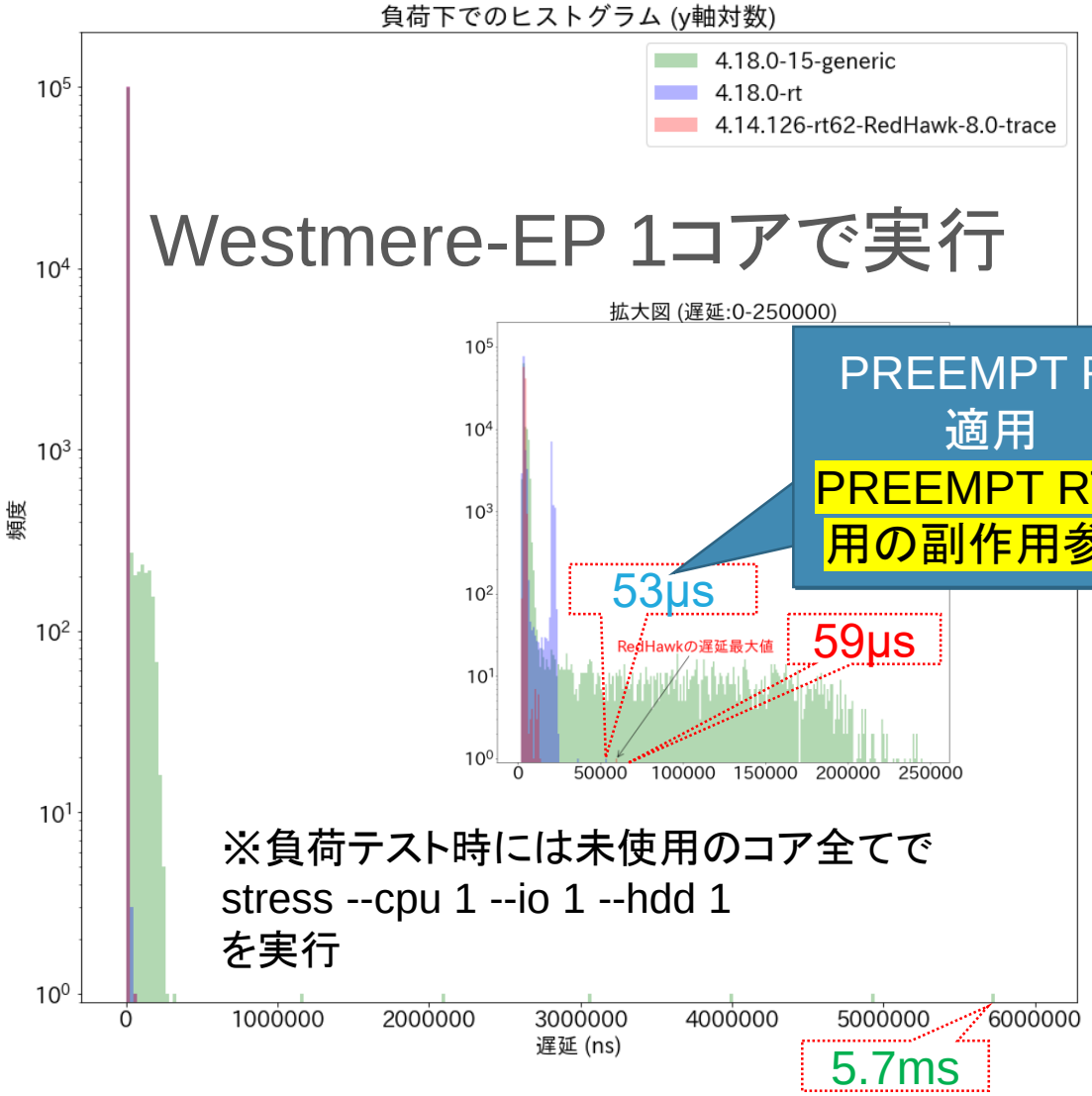
```
# for i in {0..5}; do taskset -c ${i} stress --cpu 1 --io 1 --hdd 1 & done
# taskset -c 6,7,8,9,10,11 pendulum_demo -i 100000 -f pendulum_demo.log & chrt -f -p 90 $!
```

注:Linuxカーネルのisolcpusは、システムクロック割り込みは停止できません。

⇒ 動的ティックレスの動作を特定のコアで有効にするにはカーネルコマンドで nohz_full パラメーターを使ってそのコアを指定します。(要カーネル再構築)

x86ジッター評価(有負荷:ヒストグラム)

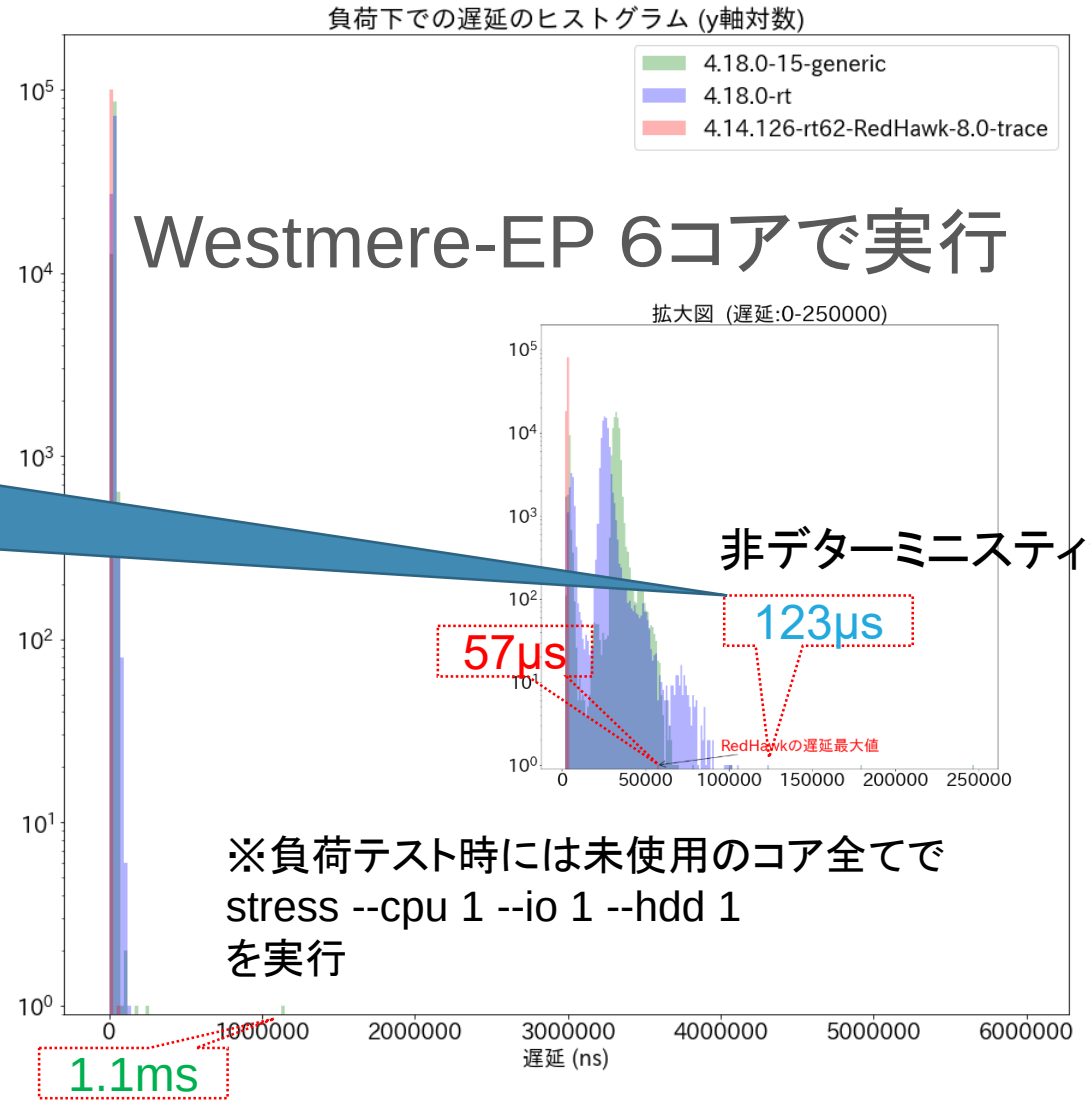
X86_64



PREEMPT RT 適用

PREEMPT RT適用の副作用参照

※負荷テスト時には未使用のコア全てで
stress --cpu 1 --io 1 --hdd 1
を実行



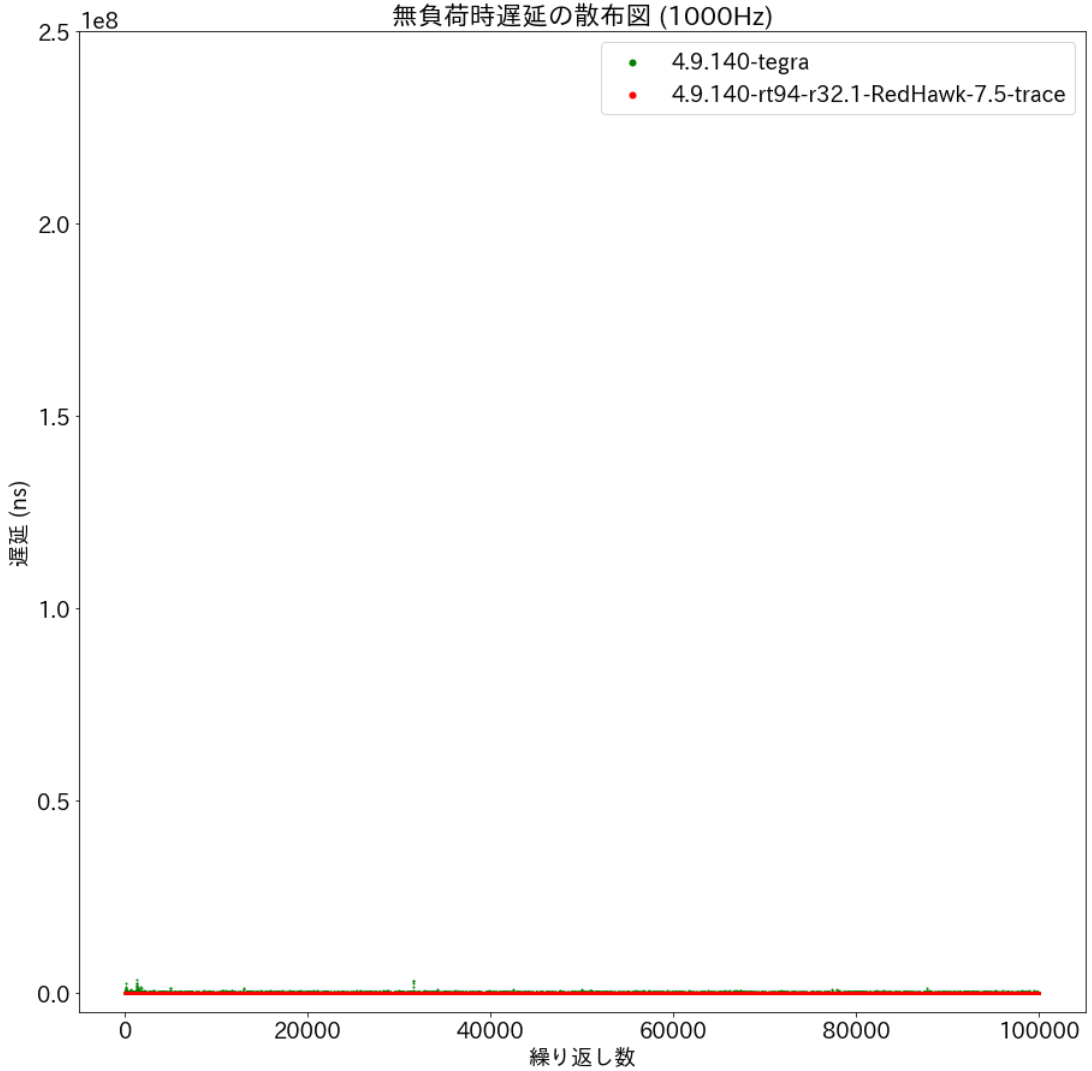
※負荷テスト時には未使用のコア全てで
stress --cpu 1 --io 1 --hdd 1
を実行

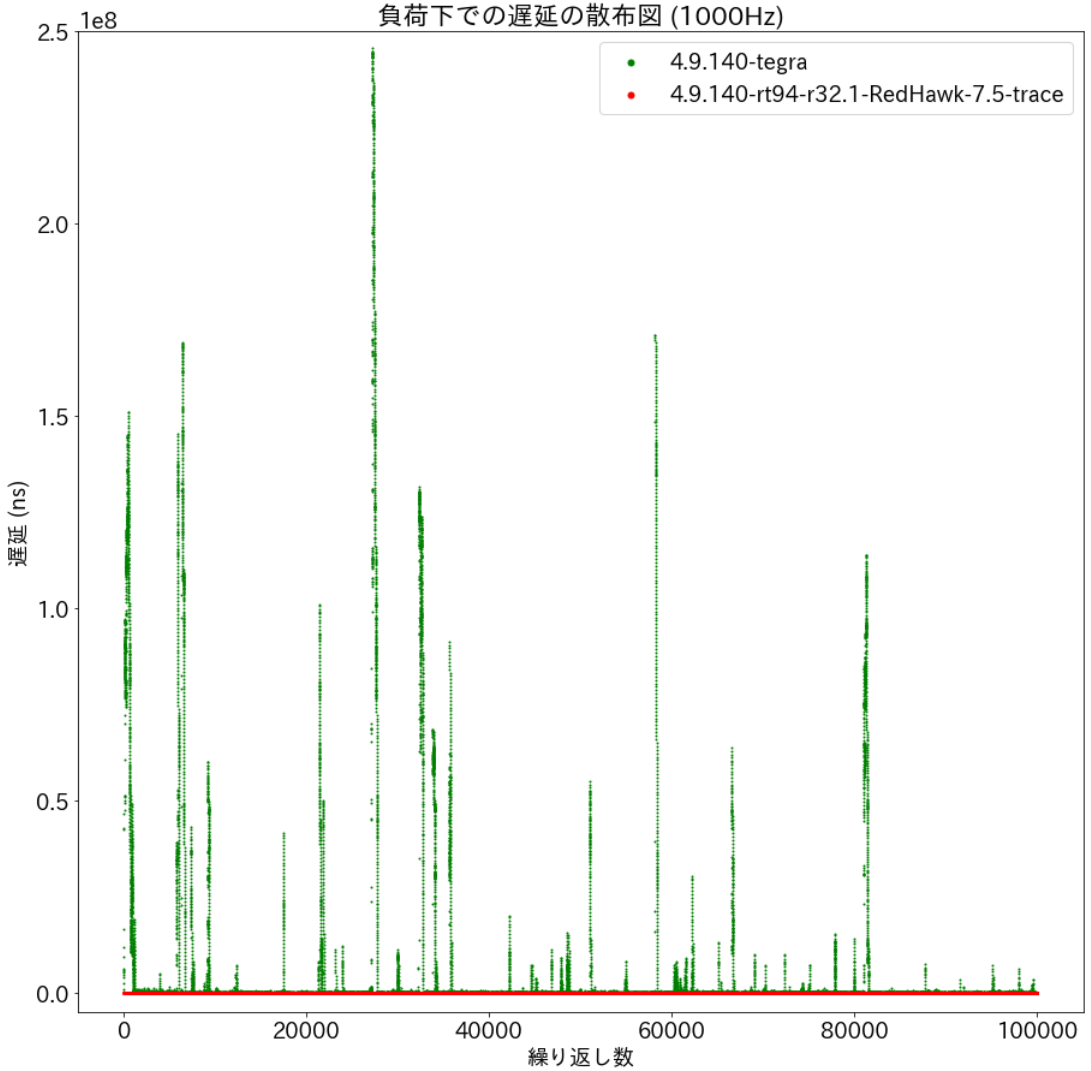
REDHAWK

concurrent REAL-TIME

ARMv8ジッター評価(トレンド)

Jetson AGX Xavier



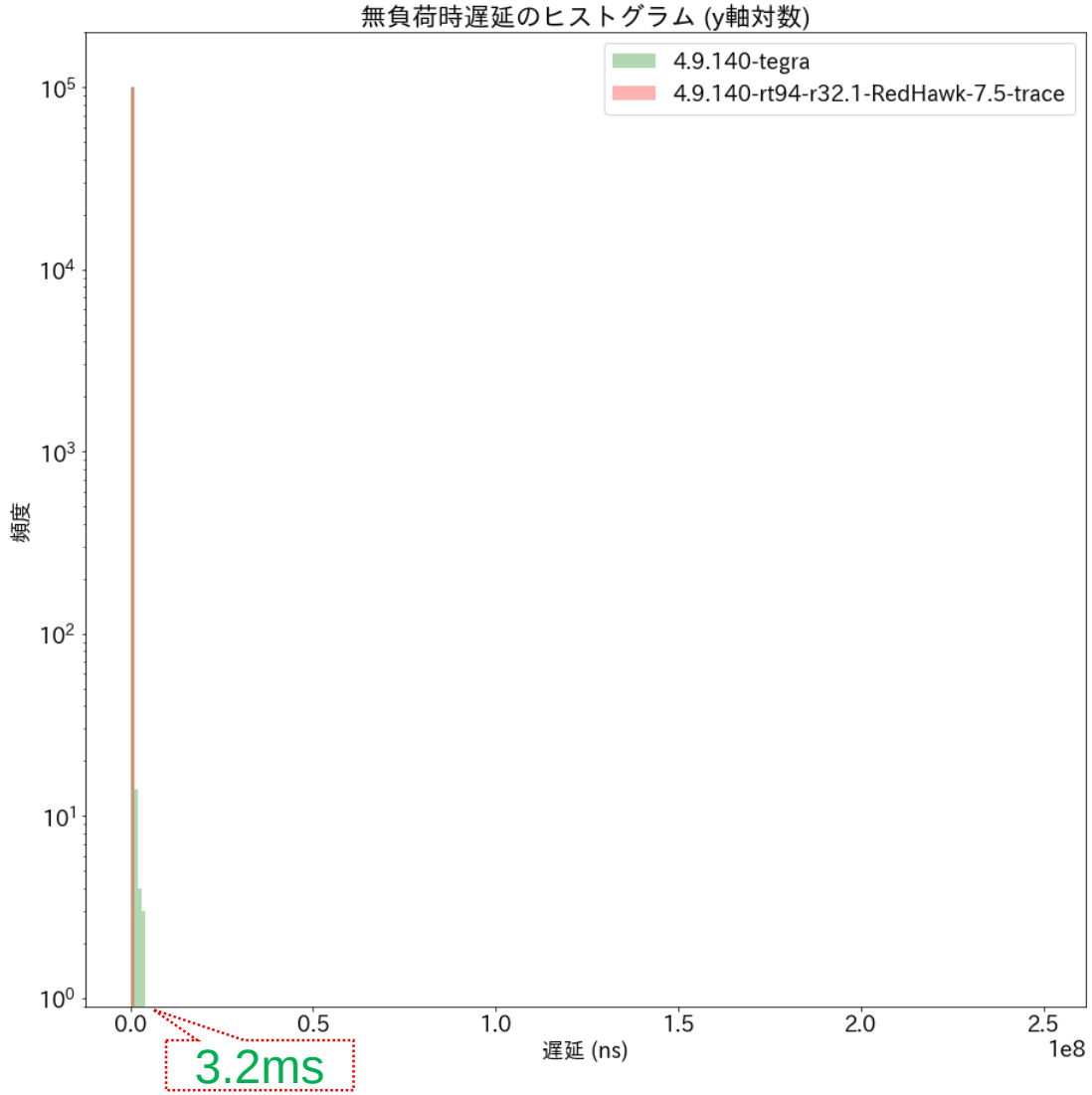


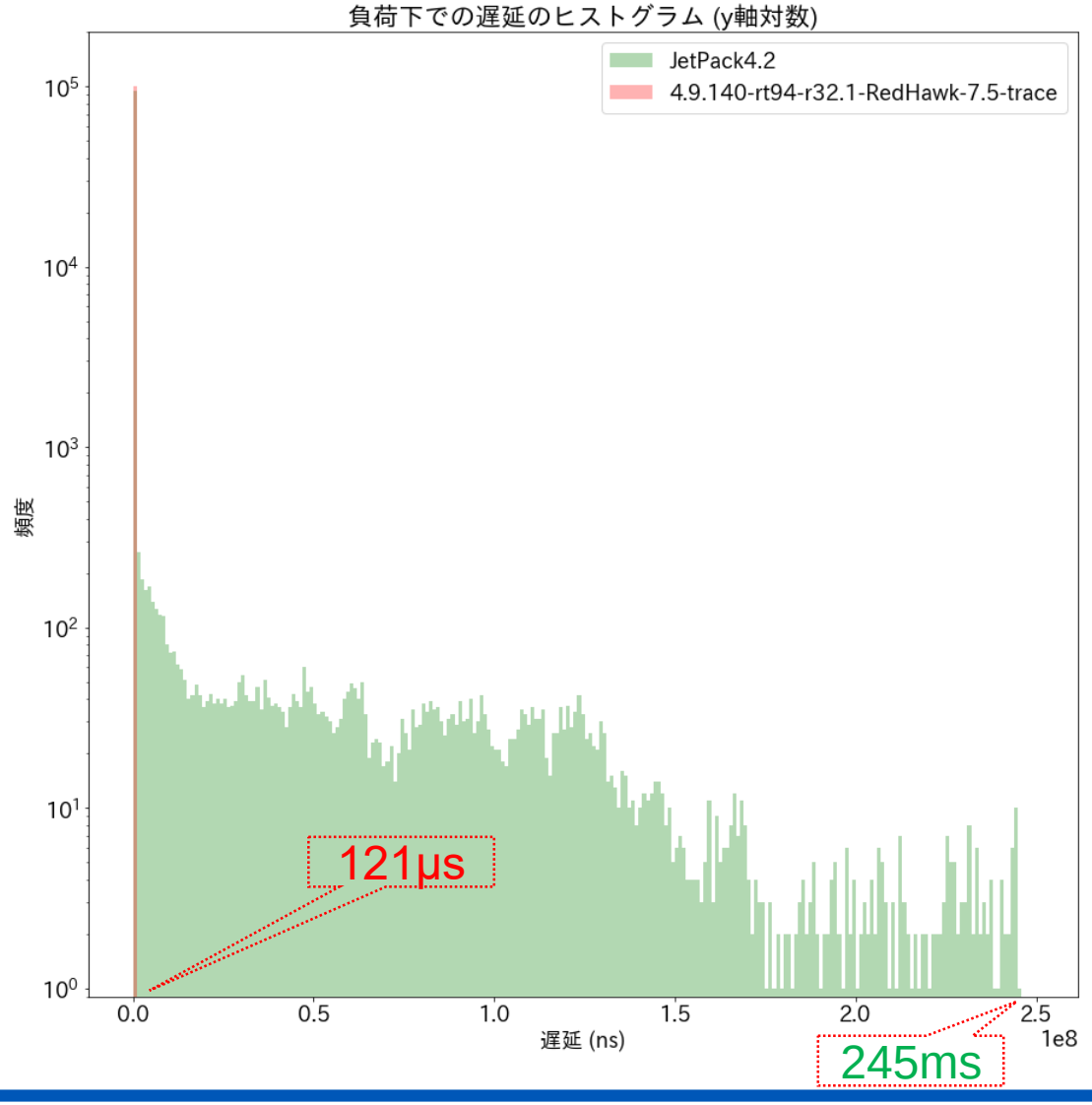
RedHawk

concurrent REAL-TIME

ARMv8ジッター評価(ヒストグラム)

Jetson AGX Xavier





RedHawk

concurrent REAL-TIME

まとめ：ジッター評価負荷あり(100000回)

Westmere-EP 6コア	オリジナルペーパー Xenomai	Ubuntu 18.04 4.18.0-15-generic	Ubuntu 18.04 PRT 4.18.0-rt1	Ubuntu 18.04 RedHawk8 4.14.126-rt62
最小(nsec)	1398.0	2375.0	2124.0	2601.0
最大(nsec)	258064.0	1136690.0	123114.0	57473.0
平均(nsec)	3729.1	29369.0	23770.7	3063.1
初回除外最大(nsec)		1136690.0	123114.0	3795.0
PendulumMotor received		92821	98137	103580
PendulumController received		90945	74702	102519
メッセージロスト率(%)		2.02	23.88	1.02

ARM v8 4コア	オリジナルペーパー Xenomai	Ubuntu 18.04 JetPack4.2 4.9.140-tegra	Ubuntu 18.04 JetPack4.2 RedHawk7.5 4.9.140-rt94-r32.1	Ubuntu 16.04 JetPack3.2 RedHawk7.3.1 4.4.38-rt49-r28.2 (TegraTX2) 参考値
最小(nsec)	1398.0	18486.0	4840.0	6555.0
最大(nsec)	258064.0	245677239.0	120873.0	47488.0
平均(nsec)	3729.1	3775425.6	8467.7	8021.6
初回除外最大(nsec)		245677239.0	87162.0	
PendulumMotor received		83567	99232	
PendulumController received		74190	99039	
メッセージロスト率(%)		11.22	0.19	

x86は、平均速度からCPUはほぼ同等。
しかし、応答の最大値はXenomaiに比して約5倍安定している。

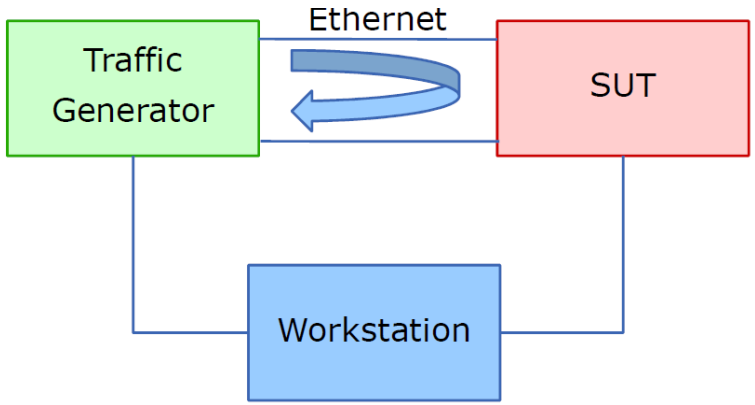
PREEMPT RT適用
PREEMPT RT適用の副作用参照

ARMは、平均速度からCPUが遅い事が読み取れる
しかし、応答の最大値はXenomaiに比して約5倍安定している。

PREEMPT RT未適用

PREEMPT RT適用の副作用

- PREEMPT RTのIRQスレッドが高い通信トラフィック上で実行される場合に**パケットが失われ、無視出来ない遅延が発生する。**
- “Threaded IRQs on Linux PREEMPT-RT”
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.163.7898&rep=rep1&type=pdf#page=23
  - http://www.artist-embedded.org/docs/Events/2009/OSPERT/OSPERT09-Henriques.pdf



Tx Rate (μsecs)	Tx/Rx Frames			
	Tx	Rx	Lost	Lost (%)
10	56,920,922	56,920,922	0	0.00
9	62,747,678	62,664,039	83,639	0.13
8	70,389,695	70,389,695	0	0.00
7	79,638,318	79,637,232	1,086	0.00
6	91,039,343	89,535,226	1,504,117	1.65
5	108,608,754	98,436,714	10,172,040	9.37

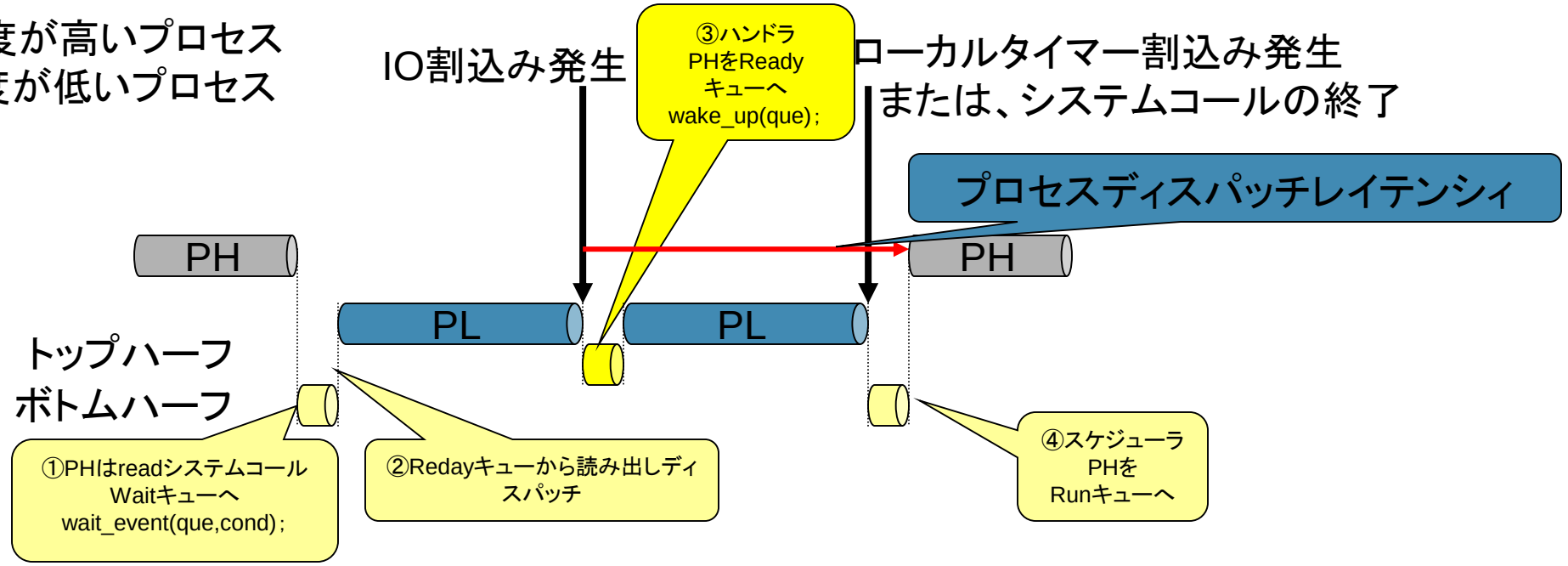
送受信の割り込み回数が異なる

Threaded IRQs kernel, cyclictest priority HIGH

プロセスディスパッチ1

標準カーネル

PH:優先度が高いプロセス
PL:優先度が低いプロセス

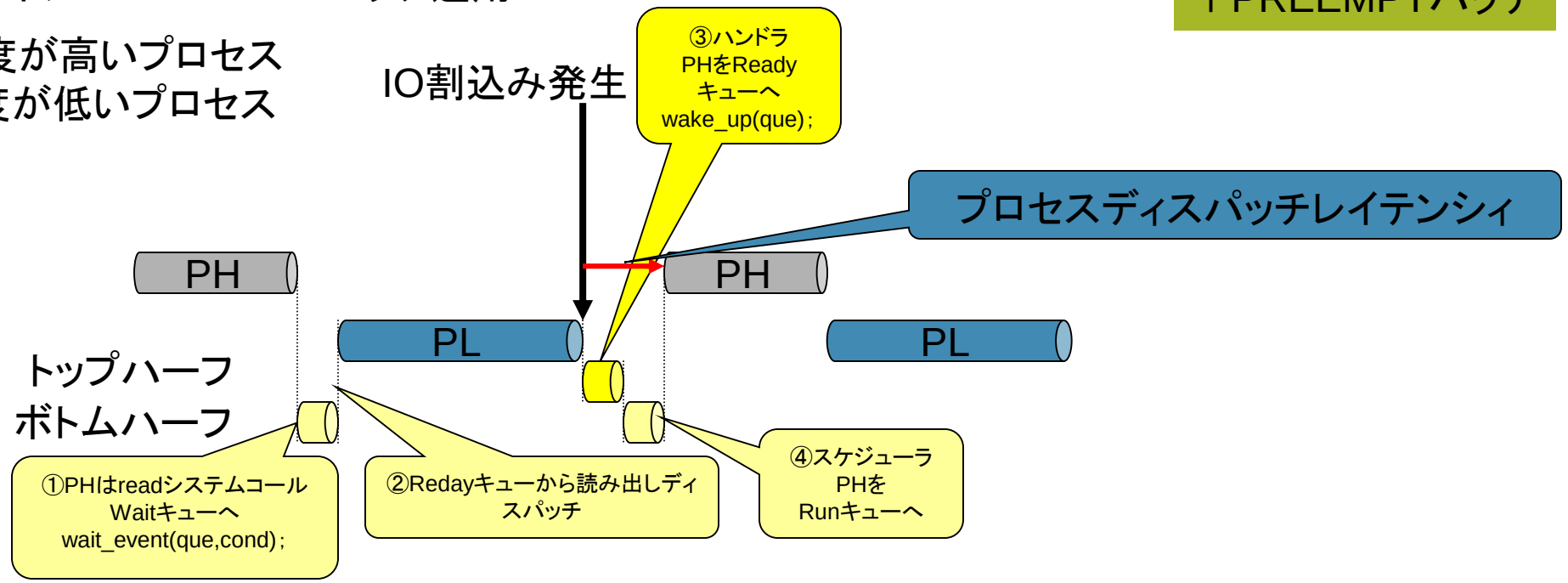


プロセスディスパッチ2

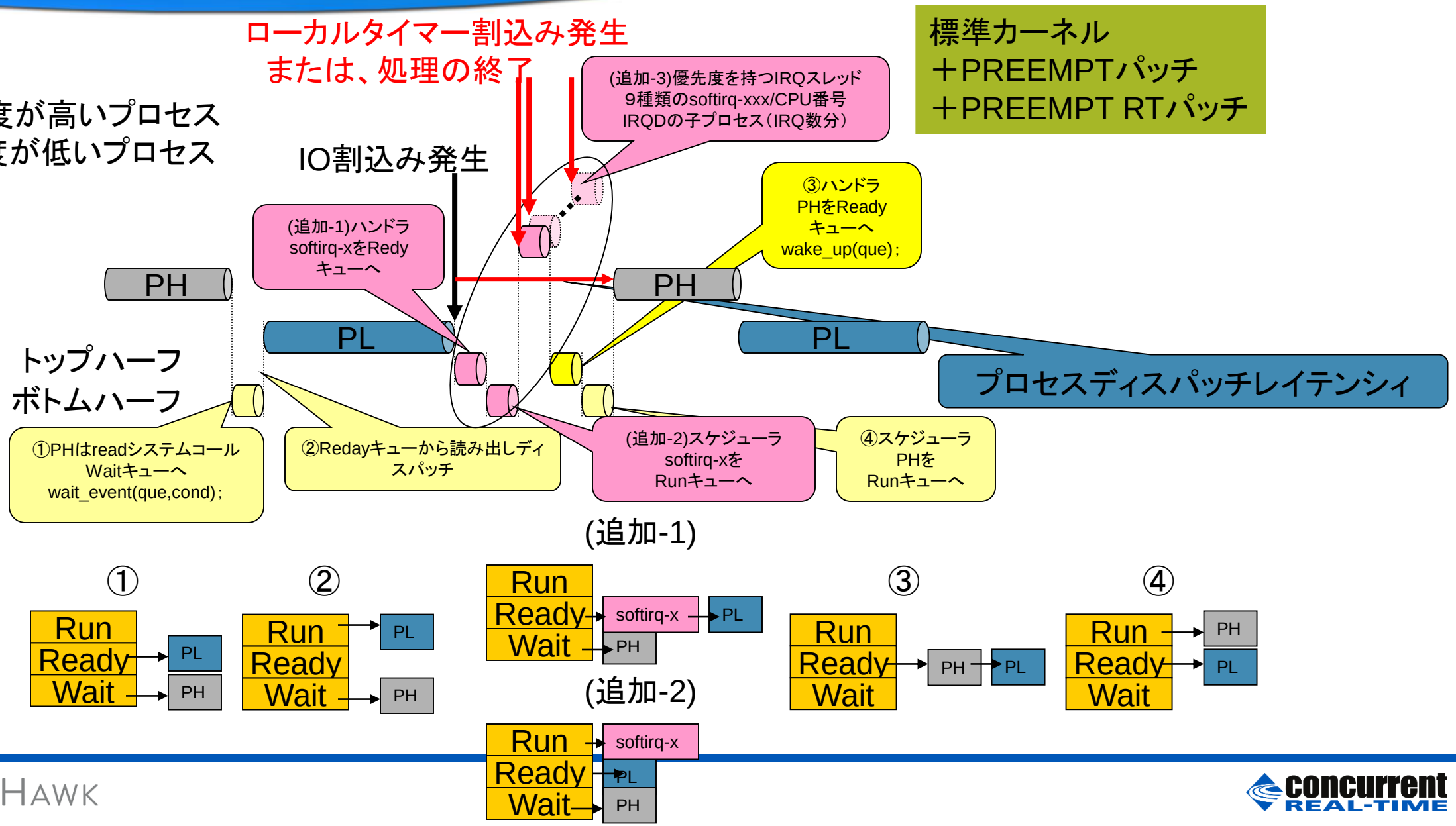
標準カーネル
＋PREEMPTパッチ

標準カーネル＋PREEMPTパッチ適用

PH:優先度が高いプロセス
PL:優先度が低いプロセス



プロセスディスパッチ3



ロボット用の分散リアルタイムフレームワークに向けて

- ROS2のリアルタイムパフォーマンスは、”DDS受信の応答”によって決まります。

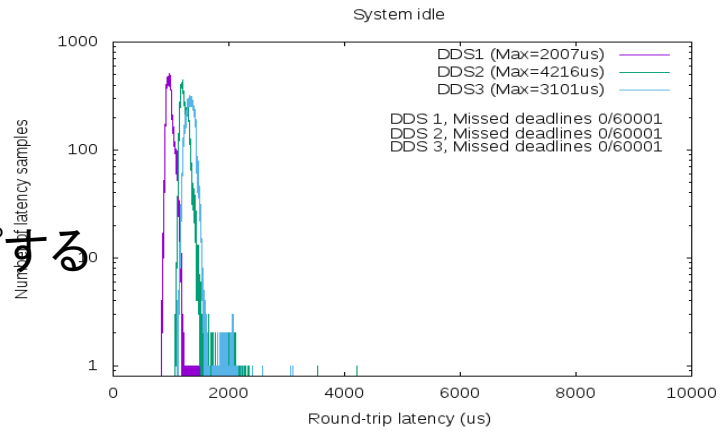
Executor

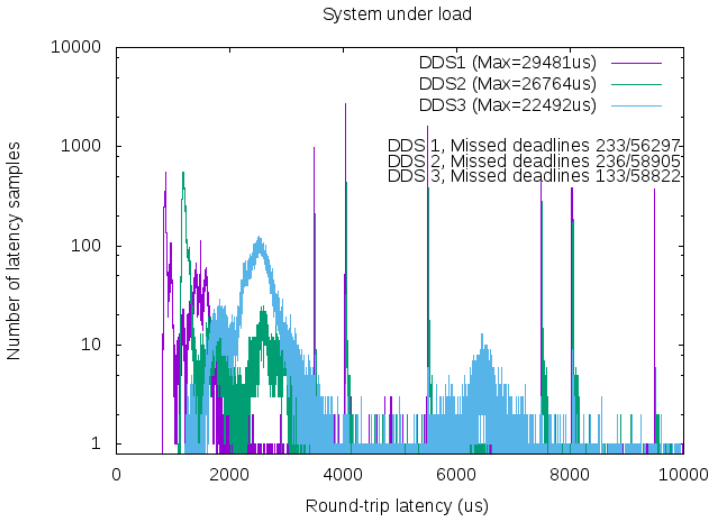
initialization — non real-time
- preallocate memory
- …

spin — real-time
```
rmw_wait(timeout)
{
    pass conditions to waitset
    wait (in DDS)
    wake-up if timed-out
}
do work if it came in
```

中断されるまでループする

cleanup — non real-time
- deallocate memory
- …



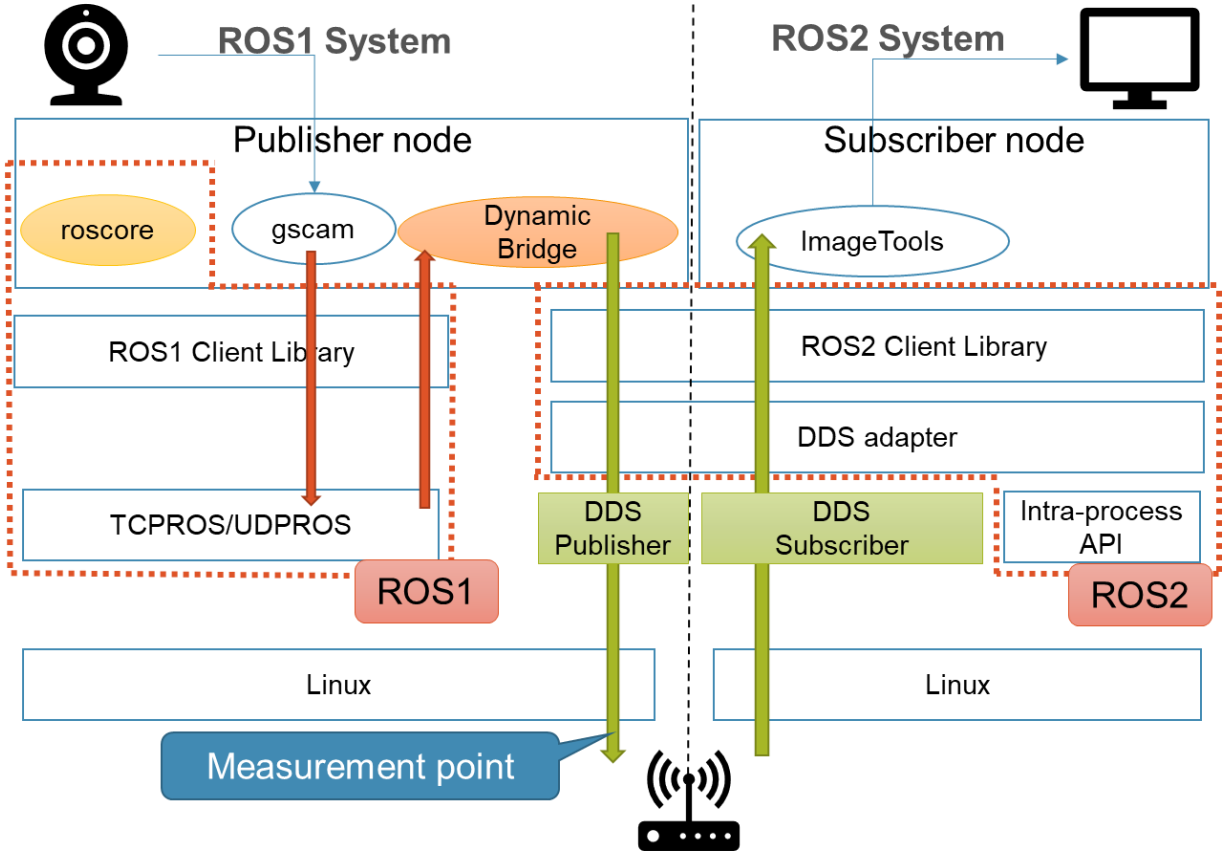


参照　Towards a distributed and real-time framework for robots
https://hackernoon.com/towards-a-distributed-and-real-time-framework-for-robots-469ba77d6c42
https://arxiv.org/pdf/1809.02595.pdf

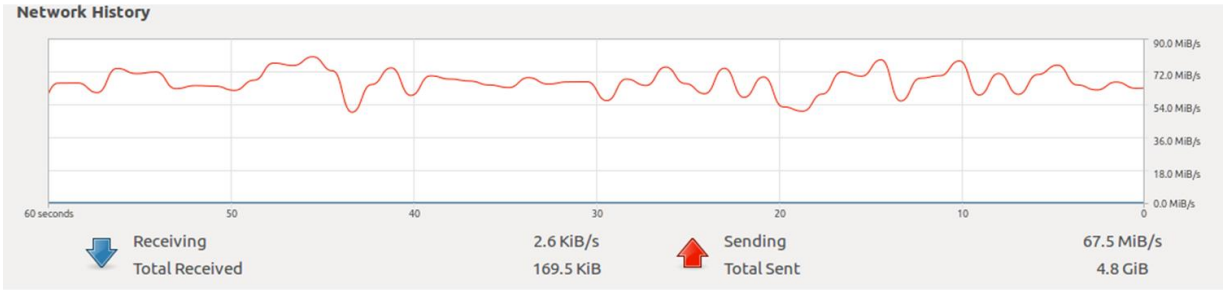
RedHawk
concurrent REAL-TIME

DDSネットワークトラフィック

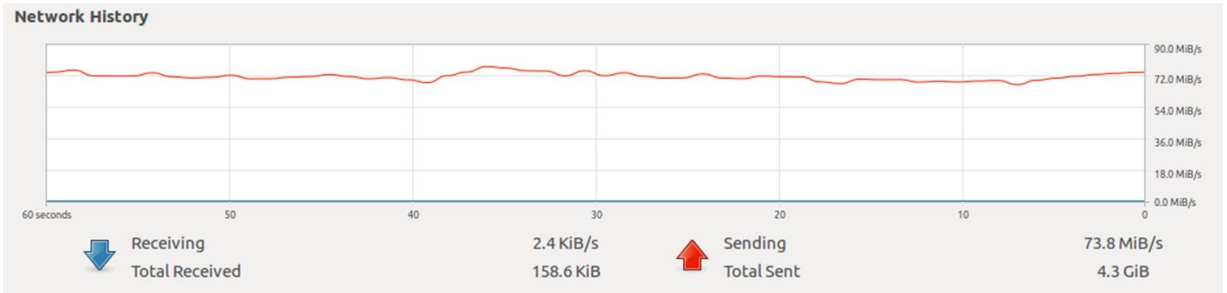
Jetson TX2



Nvidia kernel



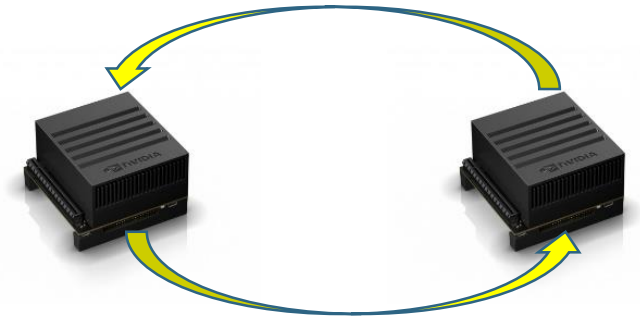
RedHawkKernel



ROS2のserviceを用いて遅延評価

Jetson AGX Xavier

- 2つのXavier間をetherで直結し、クライアントはclock_gettimeしたtimespec構造を2つのint64型として送り、サーバはその値をそのまま返信する。
- Geoffrey Biggs氏によるROS Japanユーザグループ講習会資料のソースを参考にし、リアルタイム性の向上にあたり右記の3点の処理を追加した。



- QoSのデフォルトからの変更(ROS1におけるUDPROSに準じる)
- プロセス優先度を変更
- メモリロックを追加

サーバ側:

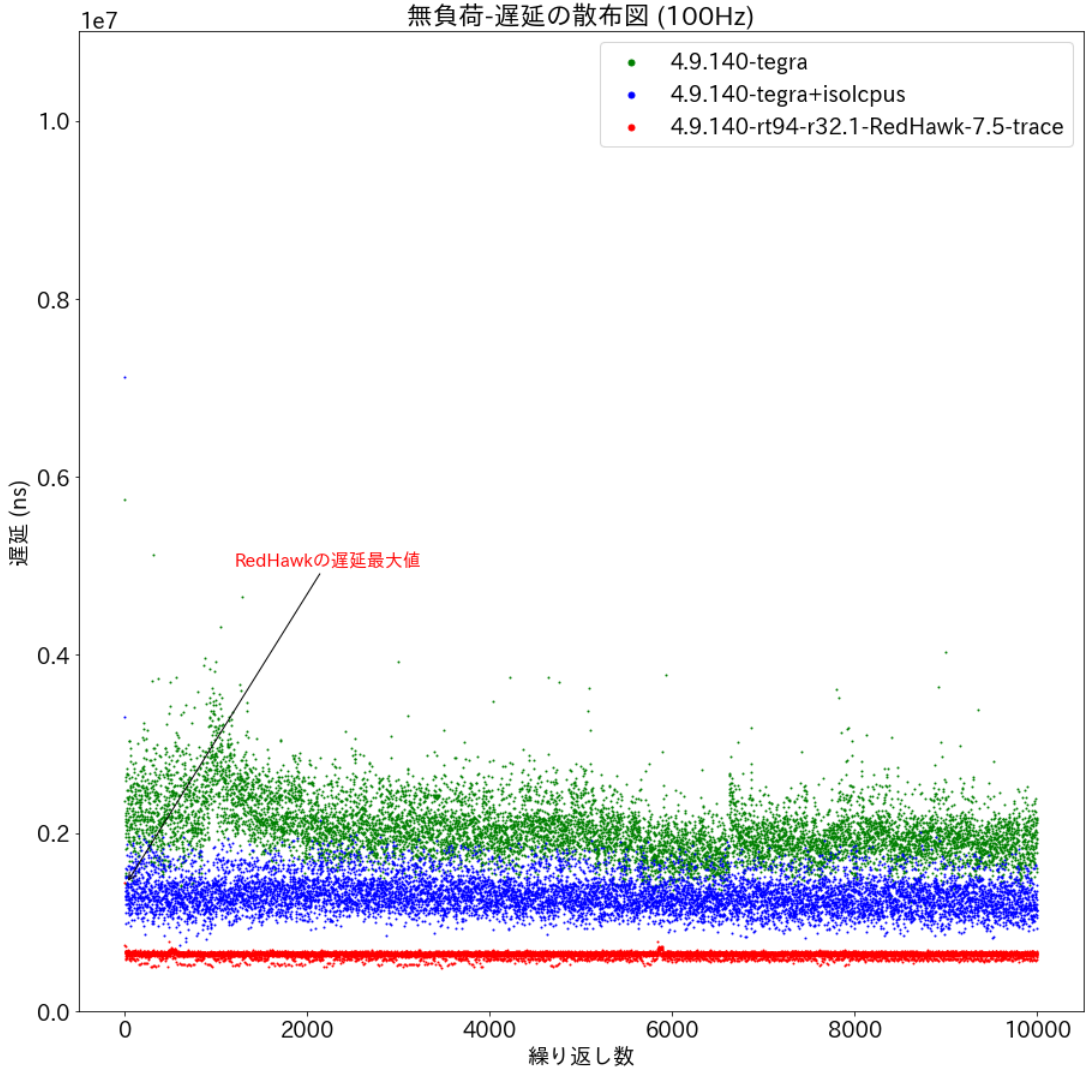
```
# run -b 4-5 ros2 run timer_server timer_server
```

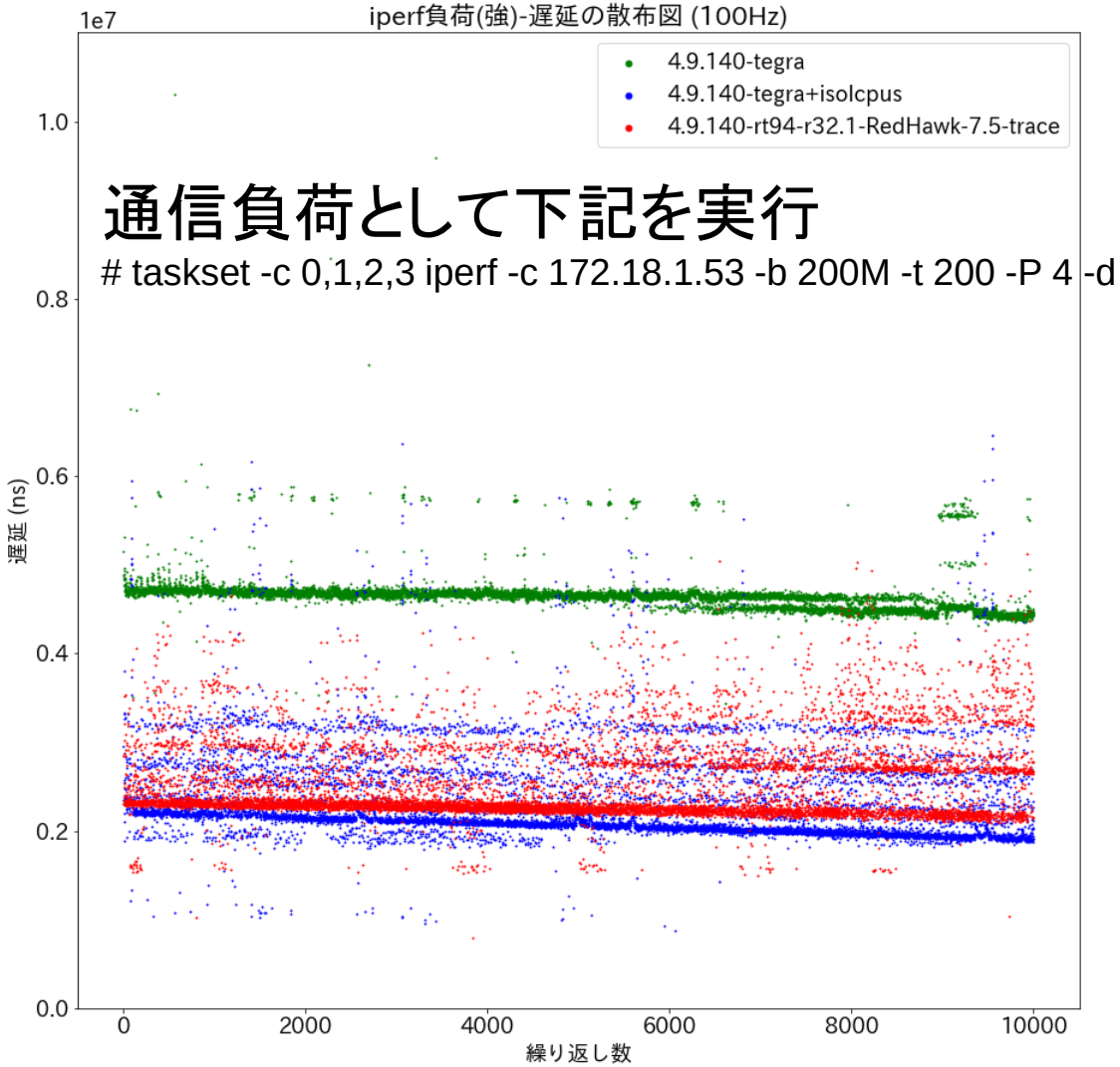
クライアント側:

```
# run -b 4-5 ros2 run timer_client timer_client
```

通信遅延評価1(トレンド)

Jetson AGX Xavier



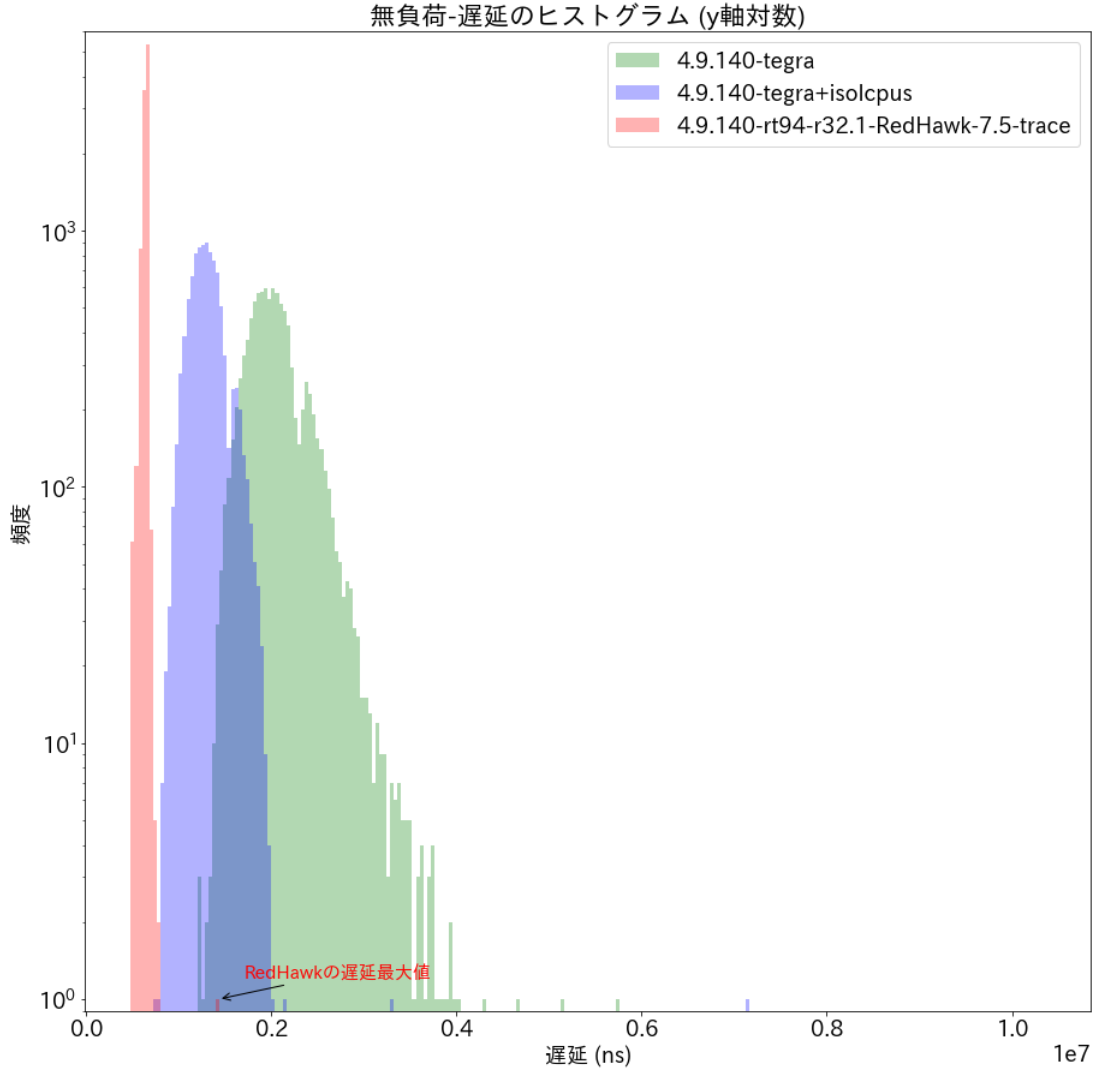


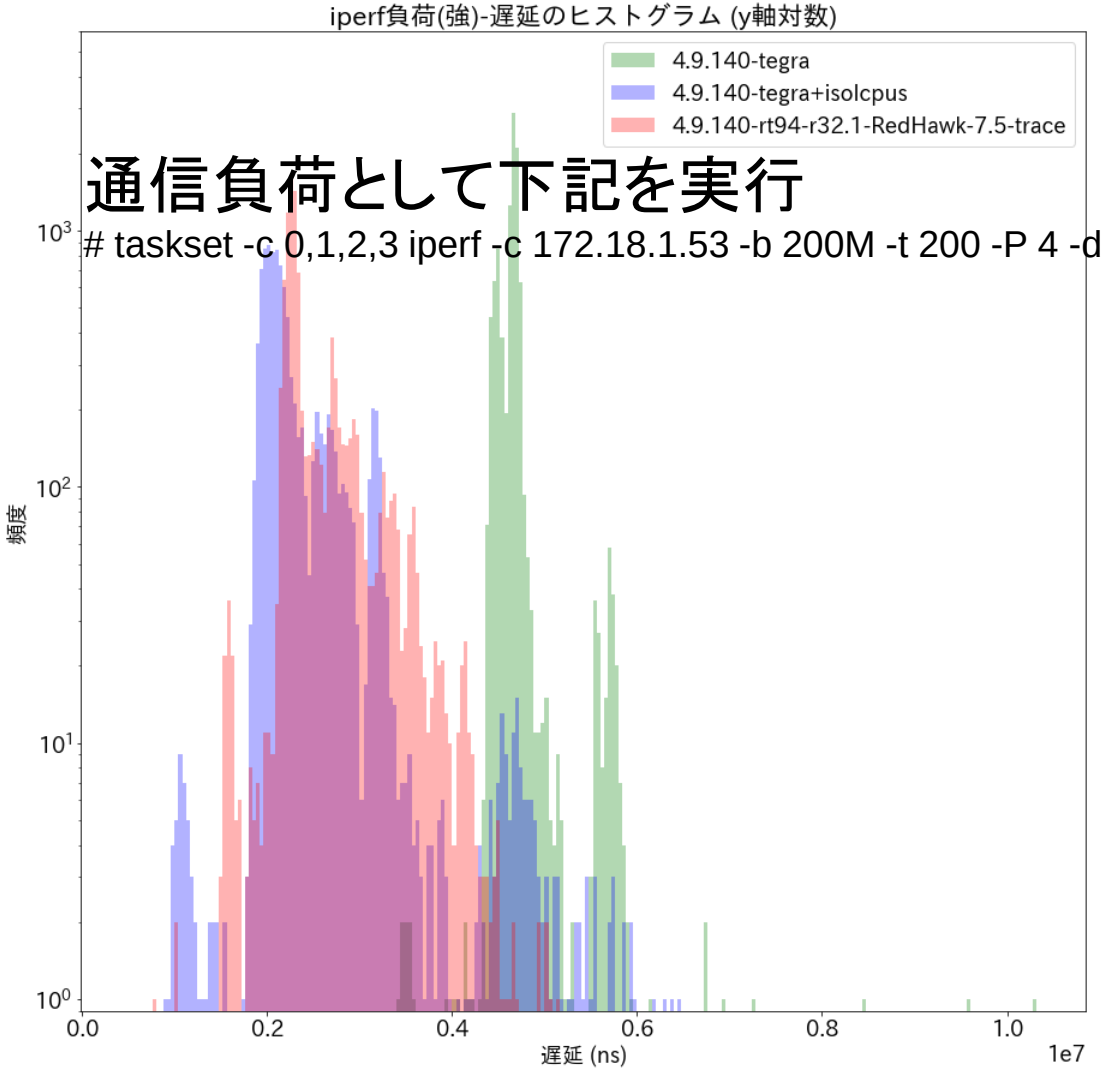
RedHawk

concurrent REAL-TIME

通信遅延評価1(ヒストグラム)

Jetson AGX Xavier



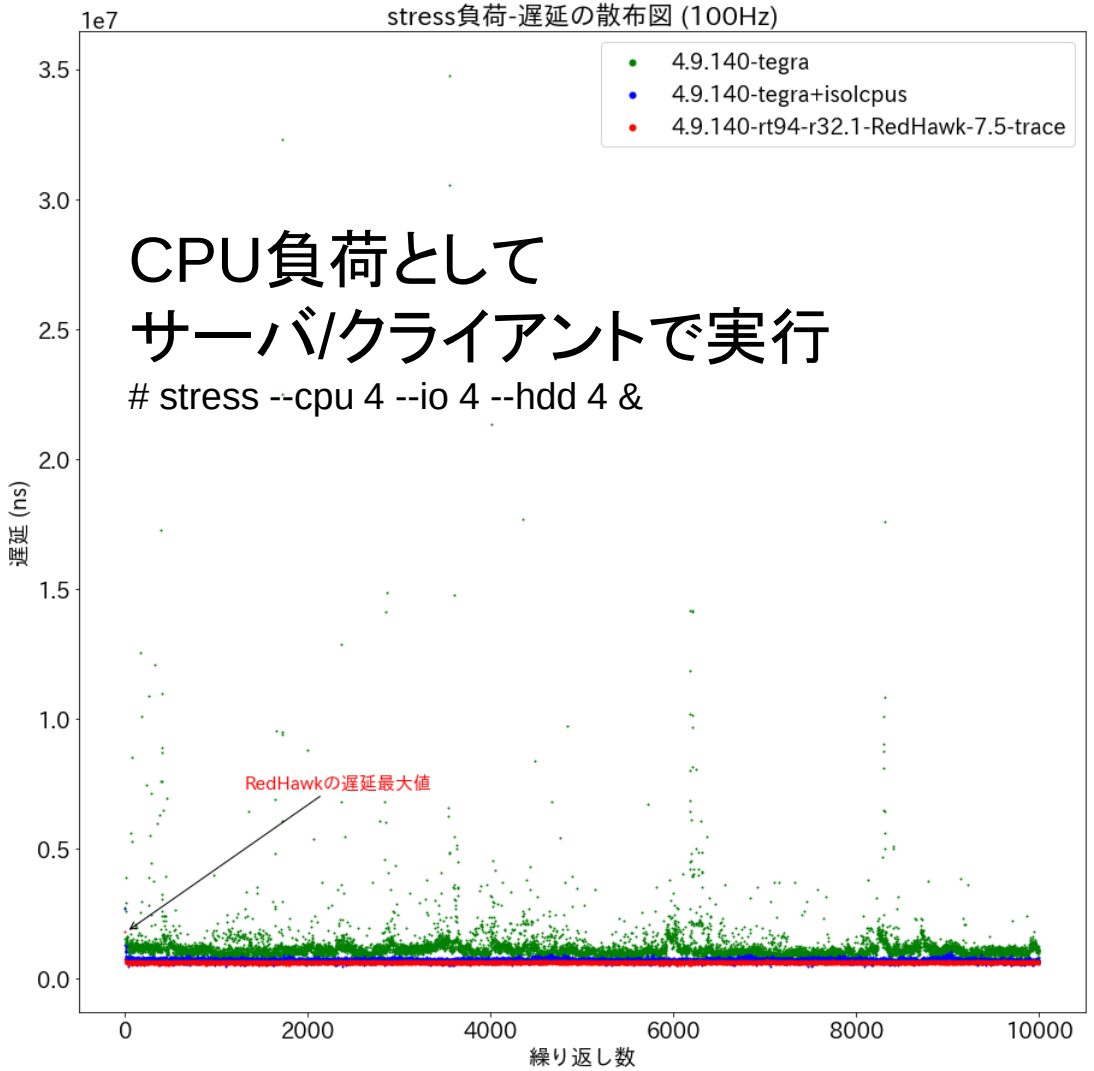


RedHawk

concurrent REAL-TIME

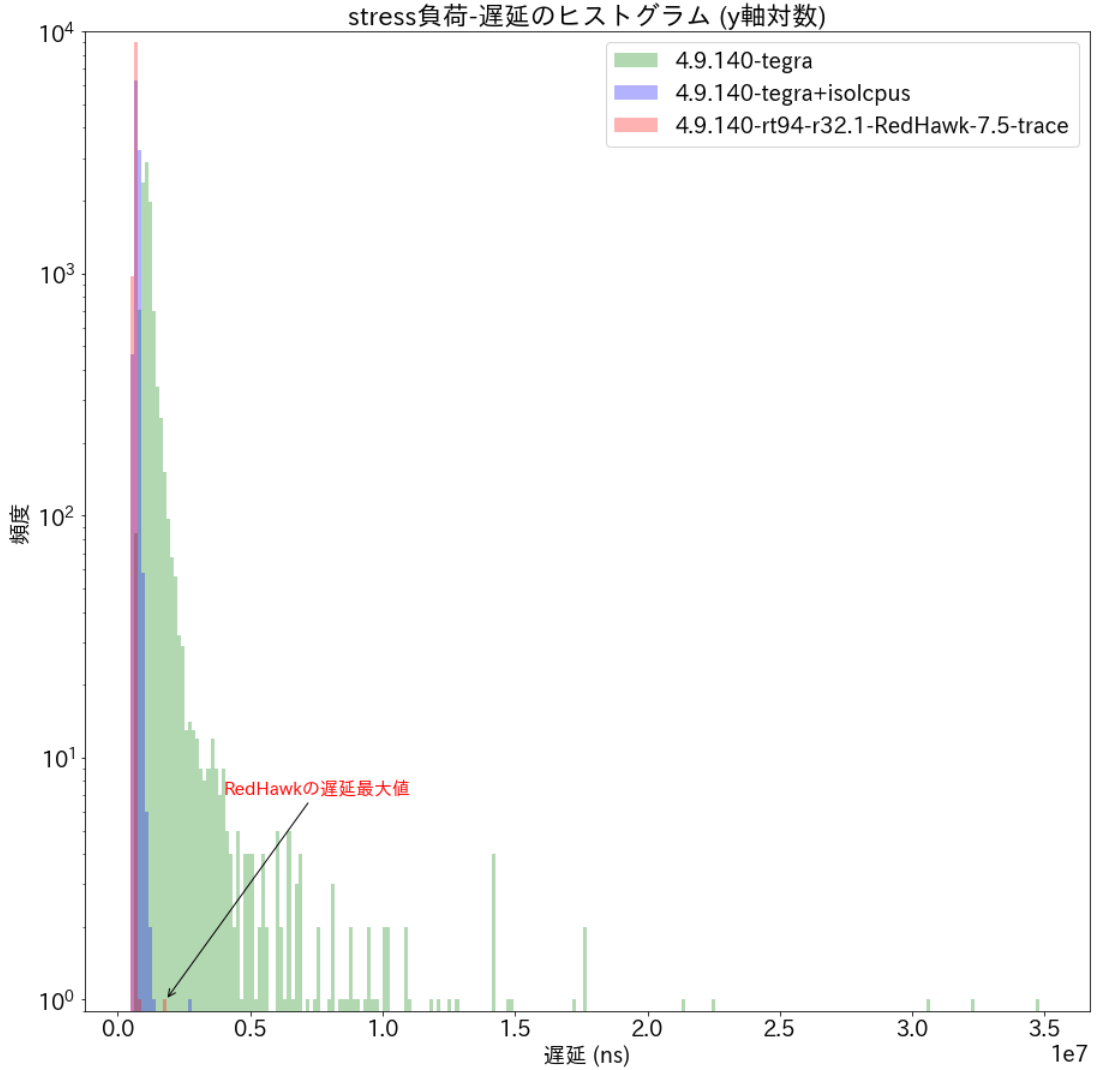
通信遅延評価2(トレンド,ヒストグラム)

Jetson AGX Xavier



CPU負荷として
サーバ/クライアントで実行

stress --cpu 4 --io 4 --hdd 4 &



RedHawk

concurrent REAL-TIME

まとめ:通信遅延評価

ARM v8 無負荷	Ubuntu 18.04 JetPack4.2 4.9.140-tegra	Ubuntu 18.04 JetPack4.2 4.9.140-tegra +isolcpus	Ubuntu 18.04 JetPack4.2 RedHawk7.5 4.9.140-rt94-r32.1
最小(nsec)	1200032.0	743904.0	487872.0
最大(nsec)	5745792.0	7124608.0	1435904.0
平均(nsec)	2054524.3	1310220.6	636011.4
初回除外最大(nsec)	5129760.0	3307616.0	777600.0
標準偏差	332489.7	198846.1	27981.3

ARM v8 通信負荷	Ubuntu 18.04 JetPack4.2 4.9.140-tegra	Ubuntu 18.04 JetPack4.2 4.9.140-tegra +isolcpus	Ubuntu 18.04 JetPack4.2 RedHawk7.5 4.9.140-rt94-r32.1
最小(nsec)	3434144.0	880864.0	792480.0
最大(nsec)	10308320.0	6454336.0	5121792.0
平均(nsec)	4651612.3	2319364.0	2522169.2
初回除外最大(nsec)	10308320.0	6454336.0	5121792.0
標準偏差	210379.3	505813.2	445846.1

ARM v8 CPU負荷	Ubuntu 18.04 JetPack4.2 4.9.140-tegra	Ubuntu 18.04 JetPack4.2 4.9.140-tegra +isolcpus	Ubuntu 18.04 JetPack4.2 RedHawk7.5 4.9.140-rt94-r32.1
最小(nsec)	614144.0	460992.0	502208.0
最大(nsec)	34724736.0	2672640.0	1822400.0
平均(nsec)	1241344.2	708169.7	639464.5
初回除外最大(nsec)	34724736.0	1321504.0	769056.0
標準偏差	1004689.7	68382.5	28725.8

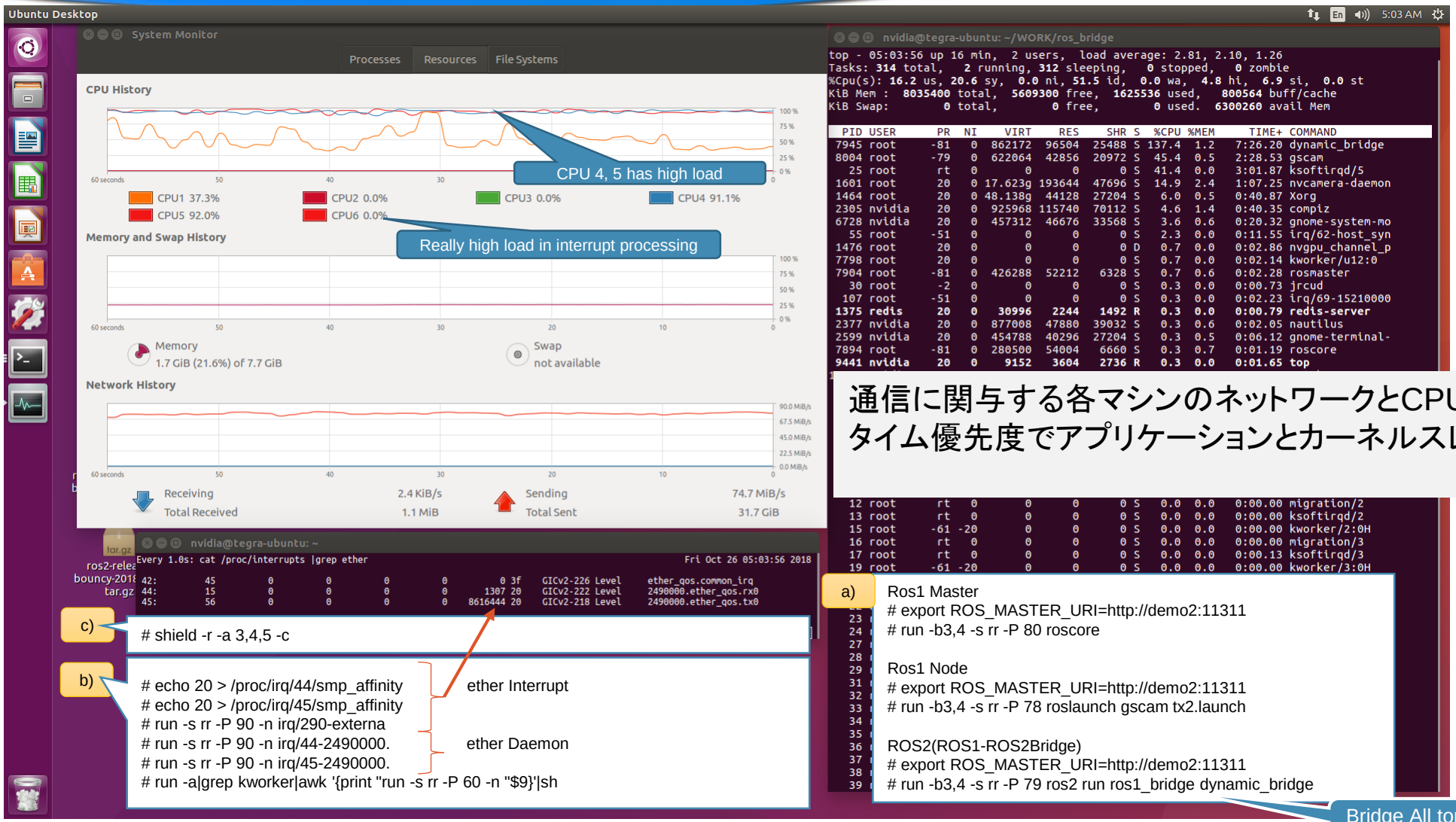
Using ROS2 with RedHawk™

Jetson TX2

- ROSはRTOSではありませんが…
  - ROS 2.0は、リアルタイムに対応し、基礎となるRTOSと緊密に協力するように設計されています
- ROS 2.0はRedHawk OSを非常に効果的に使用できます
- 下記、URLのホワイトペーパーにROS2インストールを含む詳細を記述しています。
  - “Using ROS with RedHawk Linux on the NVIDIA Jetson TX2”
  - ROS 2.0には、倒立振子のサンプルデモが含まれています
  - TX2上の最大PDLを15m秒から50μ秒まで改善しました

https://www.concurrent-rt.com/wp-content/uploads/2016/09/Using-ROS-with-RedHawk-on-Jetson-TX2.pdf

ROS 2通信を最適化するための最良の戦略



通信に関与する各マシンのネットワークとCPUの使用を制限し、リアルタイム優先度でアプリケーションとカーネルスレッドを設定します。

RedHawk

concurrent REAL-TIME

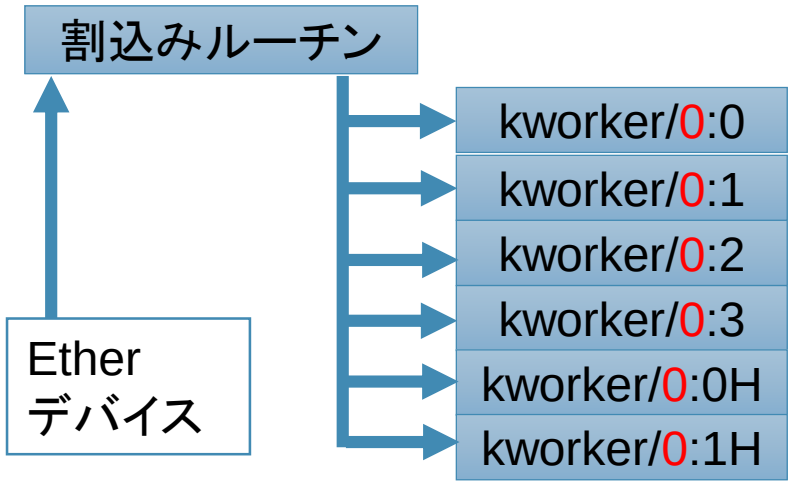
割込みの副作用

- Etherケーブルを挿抜するとジッターが発生する。
- CPUnで割込み処理を行うkworkerの優先度を上げる
- kworkerの名称
  - kworker/%u:%d%s (cpu, id, priority)
    kworker/u13:0 のようなuスレッドは現在CPUバインドされていないことを意味する。
    負のnice値(-20)を持つkworkerは、名前に「H」が付いている。
- 例　優先度0:スケジューリング Other

```
$ sudo run -s fifo -P 90 -n kworker/0:0
$ sudo run -s fifo -P 90 -n kworker/0:1
$ sudo run -s fifo -P 90 -n kworker/0:2
$ sudo run -s fifo -P 90 -n kworker/0:3
$ sudo run -s fifo -P 90 -n kworker/0:0H
$ sudo run -s fifo -P 90 -n kworker/0:1H
```

まとめて

```
$ sudo run -a|grep kworker|awk '{print "run -s fifo -P 90 -n "$9}'|sh
```



カーネルスレッドの優先度

Jetson TX2

Pid	Tid	Bias	Actual	CPU	Policy	Pri	Nice	Name
7	7	0x01	0x01	0	fifo	100	0	migration/0
8	8	0x00	0x02	1	fifo	100	0	migration/1
12	12	0x00	0x04	2	fifo	100	0	migration/2
16	16	0x08	0x08	3	fifo	100	0	migration/3
20	20	0x10	0x10	4	fifo	100	0	migration/4
24	24	0x20	0x20	5	fifo	100	0	migration/5
32	32	0x39	0x31	0	fifo	100	0	ltmrd
34	34	0x39	0x31	0	fifo	50	0	irq/55-mc_statu
43	43	0x39	0x31	4	fifo	50	0	irq/424-30c0000
44	44	0x39	0x31	0	fifo	50	0	irq/431-max7762
56	56	0x39	0x31	4	fifo	50	0	irq/62-host_syn
57	57	0x39	0x31	0	fifo	50	0	irq/63-host_sta
101	101	0x39	0x31	3	fifo	1	0	tegradc.0/a
102	102	0x39	0x31	0	fifo	1	0	tegradc.0/b
103	103	0x39	0x31	0	fifo	1	0	tegradc.0/c
104	104	0x39	0x31	0	fifo	1	0	tegradc.0/d
105	105	0x39	0x31	3	fifo	1	0	tegradc.0/e
106	106	0x39	0x31	3	fifo	1	0	tegradc.0/f
107	107	0x39	0x31	3	fifo	1	0	tegradc.0/sl
108	108	0x39	0x31	4	fifo	50	0	irq/69-15210000
112	112	0x39	0x31	5	fifo	50	0	irq/73-gk20a_st
115	115	0x39	0x31	0	fifo	50	0	irq/252-1521000
143	143	0x39	0x31	0	fifo	50	0	irq/54-tegra_rt
150	150	0x39	0x31	0	fifo	50	0	irq/22-3460000.
151	151	0x39	0x31	0	fifo	50	0	irq/77-b150000.
153	153	0x39	0x31	3	fifo	50	0	irq/78-b150000.
155	155	0x39	0x31	4	fifo	50	0	irq/78-b150000.
156	156	0x39	0x31	0	fifo	50	0	irq/52-b000000.
157	157	0x39	0x31	0	fifo	50	0	irq/81-c150000.
158	158	0x39	0x31	4	fifo	50	0	irq/53-d230000.
171	171	0x39	0x31	0	fifo	50	0	irq/92-gpcdma.0
172	172	0x39	0x31	0	fifo	50	0	irq/93-gpcdma.1
173	173	0x39	0x31	0	fifo	50	0	irq/94-gpcdma.2
174	174	0x39	0x31	0	fifo	50	0	irq/95-gpcdma.3
175	175	0x39	0x31	0	fifo	50	0	irq/96-gpcdma.4
176	176	0x39	0x31	0	fifo	50	0	irq/97-gpcdma.5
177	177	0x39	0x31	0	fifo	50	0	irq/98-gpcdma.6
178	178	0x39	0x31	0	fifo	50	0	irq/99-gpcdma.7
179	179	0x39	0x31	0	fifo	50	0	irq/100-gpcdma.
180	180	0x39	0x31	0	fifo	50	0	irq/101-gpcdma.
181	181	0x39	0x31	0	fifo	50	0	irq/102-gpcdma.
182	182	0x39	0x31	0	fifo	50	0	irq/103-gpcdma.
183	183	0x39	0x31	0	fifo	50	0	irq/104-gpcdma.
184	184	0x39	0x31	0	fifo	50	0	irq/105-gpcdma.
185	185	0x39	0x31	0	fifo	50	0	irq/106-gpcdma.
186	186	0x39	0x31	0	fifo	50	0	irq/107-gpcdma.
187	187	0x39	0x31	0	fifo	50	0	irq/108-gpcdma.
188	188	0x39	0x31	0	fifo	50	0	irq/109-gpcdma.

51と98の間
6と49の間
2と 4の間

189	189	0x39	0x31	0	fifo	50	0	irq/110-gpcdma.
190	190	0x39	0x31	0	fifo	50	0	irq/111-gpcdma.
191	191	0x39	0x31	0	fifo	50	0	irq/112-gpcdma.
192	192	0x39	0x31	0	fifo	50	0	irq/113-gpcdma.
193	193	0x39	0x31	0	fifo	50	0	irq/114-gpcdma.
194	194	0x39	0x31	0	fifo	50	0	irq/115-gpcdma.
195	195	0x39	0x31	0	fifo	50	0	irq/116-gpcdma.
196	196	0x39	0x31	0	fifo	50	0	irq/117-gpcdma.
197	197	0x39	0x31	0	fifo	50	0	irq/118-gpcdma.
198	198	0x39	0x31	0	fifo	50	0	irq/119-gpcdma.
199	199	0x39	0x31	0	fifo	50	0	irq/120-gpcdma.
200	200	0x39	0x31	0	fifo	50	0	irq/121-gpcdma.
201	201	0x39	0x31	0	fifo	50	0	irq/122-gpcdma.
202	202	0x39	0x31	0	fifo	50	0	irq/123-gpcdma.
203	203	0x39	0x31	0	fifo	50	0	irq/23-3440000.
207	207	0x39	0x31	0	fifo	50	0	irq/24-3400000.
209	209	0x39	0x31	4	fifo	1	0	mmcqd/0
211	211	0x39	0x31	3	fifo	1	0	mmcqd/0boot0
213	213	0x39	0x31	4	fifo	1	0	mmcqd/0boot1
215	215	0x39	0x31	0	fifo	1	0	mmcqd/0rpmb
216	216	0x39	0x31	0	fifo	50	0	irq/256-3400000
219	219	0x39	0x31	0	fifo	50	0	irq/64-150c0000
221	221	0x39	0x31	0	fifo	50	0	irq/60-3530000.
222	222	0x39	0x31	0	fifo	50	0	irq/34-3210000.
224	224	0x39	0x31	0	fifo	50	0	irq/35-c260000.
226	226	0x39	0x31	0	fifo	50	0	irq/36-3240000.
229	229	0x39	0x31	0	fifo	50	0	irq/60-xotg
234	234	0x39	0x31	0	fifo	50	0	irq/65-15700000
640	640	0x01	0x01	0	fifo	50	0	dhd_dpc
641	641	0x39	0x31	0	fifo	50	0	dhd_rxf
3	3	0x01	0x01	0	rr	99	0	ksoftirqd/0
9	9	0x00	0x02	1	rr	99	0	ksoftirqd/1
13	13	0x00	0x04	2	rr	99	0	ksoftirqd/2
17	17	0x08	0x08	3	rr	99	0	ksoftirqd/3
21	21	0x10	0x10	4	rr	99	0	ksoftirqd/4
25	25	0x20	0x20	5	rr	99	0	ksoftirqd/5
30	30	0x39	0x31	0	rr	1	0	jrcud
2063	2094	0x39	0x31	5	rr	5	0	pulseaudio
2063	2107	0x39	0x31	0	rr	5	0	pulseaudio
2063	2108	0x39	0x31	3	rr	5	0	pulseaudio
2068	2079	0x39	0x31	0	rr	99	1	rtkit-daemon

カーネルスレッド

Jetson TX2

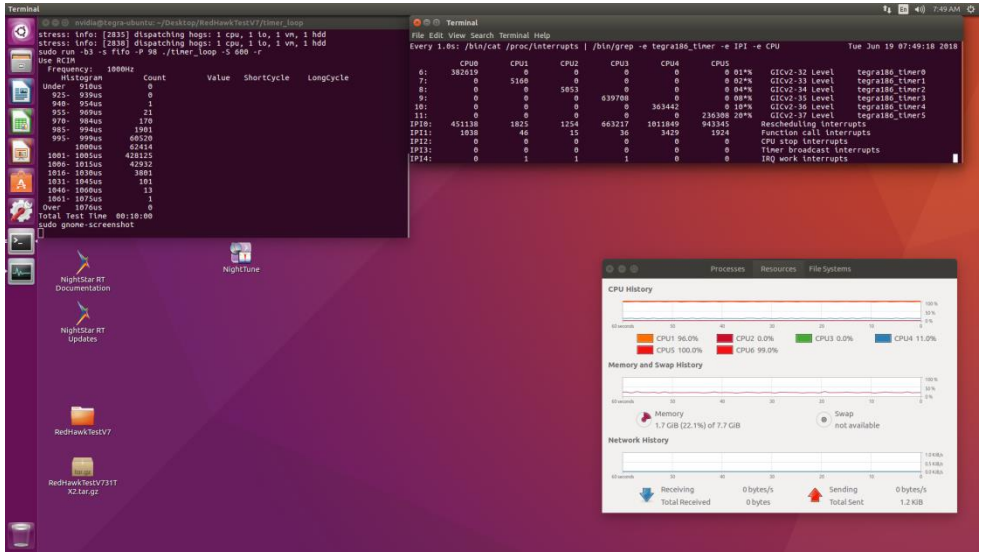
名称(赤字部分なし)	
tegradc.0/*	tegra-iommu
irq/*-gpcdma.*	tegra186-gpc-dma
irq/60-xotg	XUSB OTG Controller
irq/60-3530000.xhci	Xhci USB Controller
irq/22-3460000.sdhi irq/23-3440000.sdhi irq/24-3400000.sdhi irq/256-3400000.sdhi	Tegra Secure Digital Host Controller
irq/64-150c0000.nvcsi irq/65-15700000.vi	Sensor Controller
dhd_dpc dhd_rxf	Broadcom Wireless Driver
irq/55-mc_status	Memory (error) Controller
irq/424-30c0000.watchdog	Watchdog Controller
irq/431-max7762.top	Max77620 Controller
irq/62-host_syncpt irq/63-host_staus	Tegra Graphics Host Syncpoint Integration
kworker/%u:%d%s (cpu, id, priority)	割り込み、タイマー、I / Oなどがある場合に、カーネルの実際の処理の大部分を実行するカーネルワーカースレッド。

GPU副作用の軽減

Jetson TX2

- Xorgとnvgpu_channel_pollの優先度を上げると、安定する
- nvgpu_channel_poll:GPUとの通信路(FIFO)を管理

```
$ sudo run -b0 -s rr -P 51 -n nvgpu_channel_p
$ sudo run -b0 -s rr -P 51 -n Xorg -L a
$ run -a|grep -e Xorg -e gpu
1048    1048    0x01    0x01      0      rr         51     0       Xorg
1064    1064    0x01    0x01      0      rr         51     0       nvgpu_channel_p
```



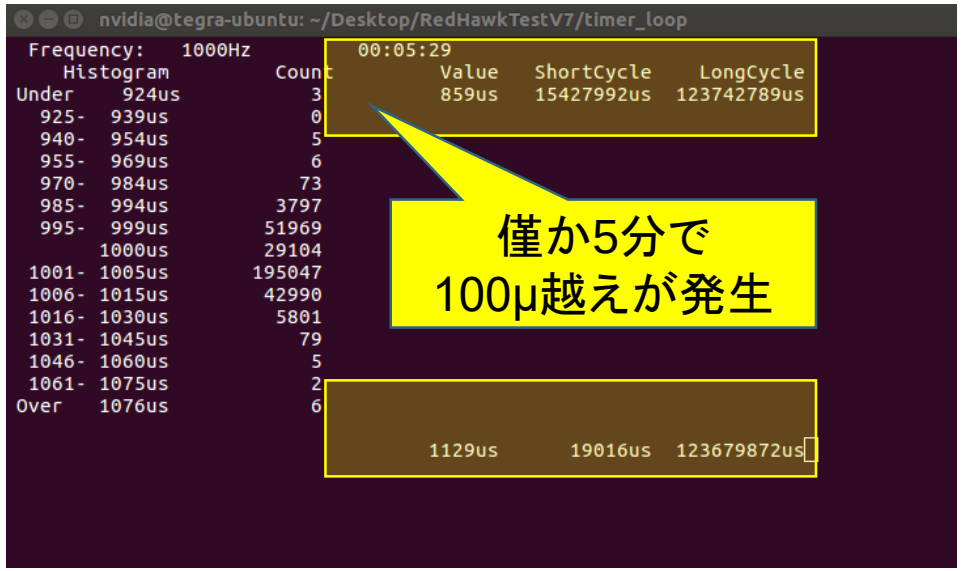
RedHawk

concurrent REAL-TIME

POSIX タイマー性能

Jetson TX2

NVIDIAカーネル

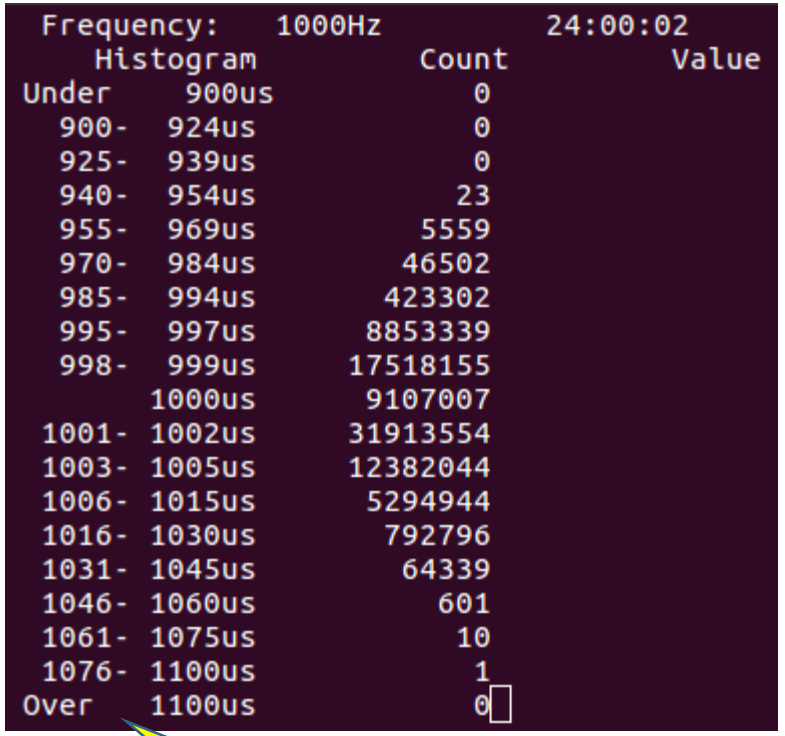


僅か5分で
100μ越えが発生

CORE	SHIELDED	Programs
0	No	Interrupt,Daemon stress --cpu1 --io 1 --vm 1 --hdd 1
1	No	DOWN
2	No	DOWN
3	No	stress --cpu1 --io 1 --vm 1 --hdd 1
4	No	stress --cpu1 --io 1 --vm 1 --hdd 1
5	Yes	ベンチマークテストプログラム

ベンチマーク以外の
CPUには、100%負荷

RedHawkカーネル



24H試験でも
100μ秒以下

RedHawk

concurrent rt REAL-TIME

THANK YOU