concurrent

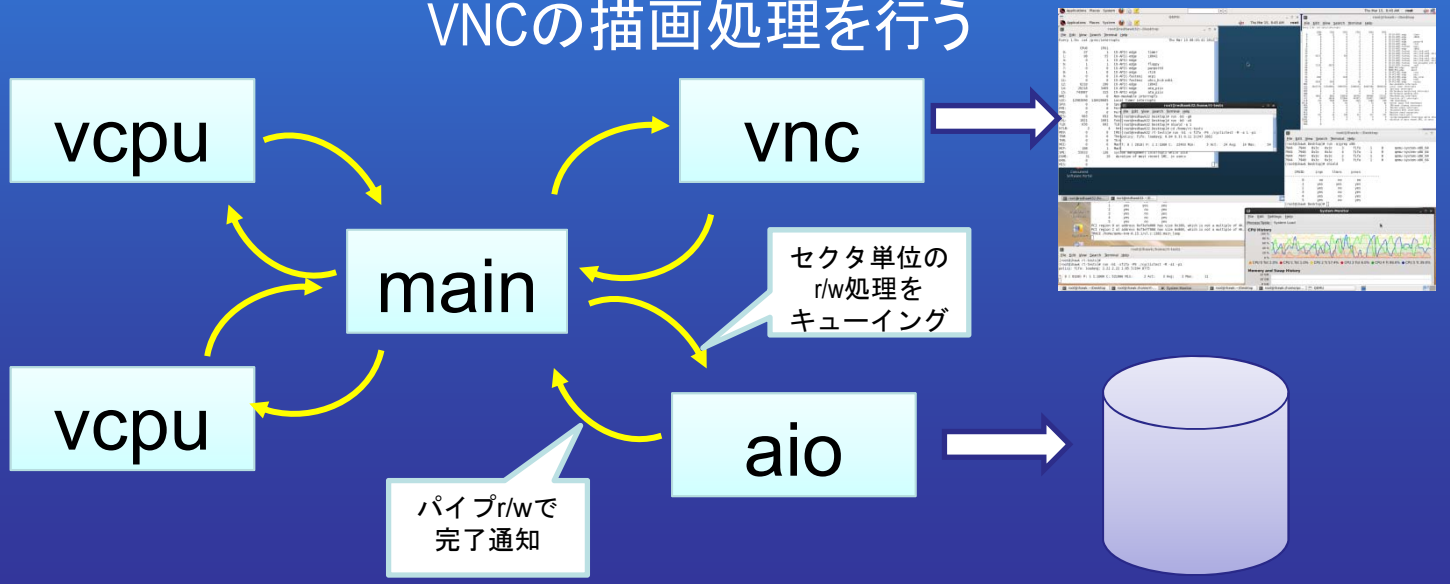
Real Time QEMU

Professional Service

Tatuhiro Oshima

QEMUの構造

- QEMUは、複数のサービススレッドを生成し並行処理。
  - main thread　メインスレッド　すべてのIOとイベントを管理
  - vcpu thread　ゲストOSを実行 smpで指定した数だけ生成
  - paio thread　ディスクIOを行うスレッド linuxのAIOを使用
  - vnc thread　configur時に指定すると生成する
    VNCの描画処理を行う



concurrent

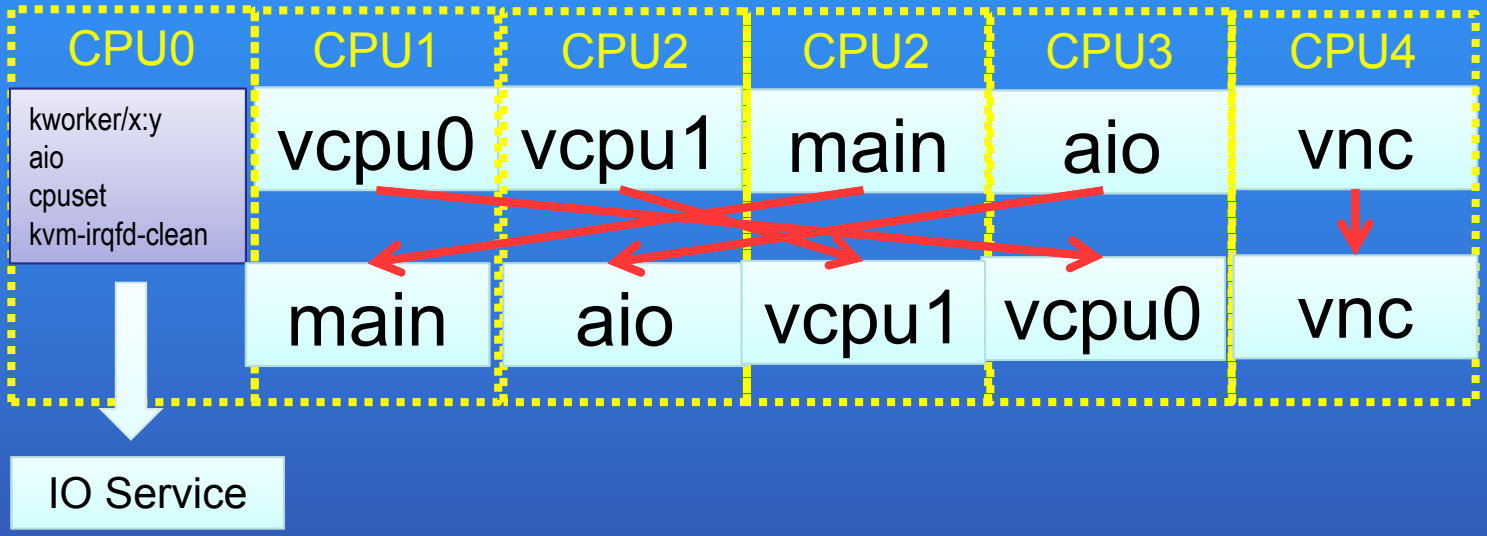
QEMUスレッド

- 総てはディフォルト固定で、無制御状態
  - スケジュール　　TS
  - 優先度　　0
  - CPUアフィニティ　　総てのスレッドは同じマスク

- 総てのリクエストは、mainスレッドが制御
- ゲストOSから発行されたディスクIOは、カーネルのAIOコールで処理される

QEMUプロセススケジュール

- 空いているCPUでQEMUスレッドは実行される
- プロセスに割り付けが可能でも実行ＣＰＵは移動



- カーネルスレッドのスケジュールも必要
- キャッシュがフラッシュするので、ジッター大
- mainスレッドがスケジュールされないとイベントリターンは配信されない

マルチQEMUプロセスのスケジューリング

- プロセス単位で、公平なスケジュールを期待しているが....

- スケジュールは、スレッド単位なので、要求の多いQEMUスレッド(AIO)がスケジュールされてしまう

- Cgroup制御を行っても、Guest2のIO要求の数は減らせない(Guest1はDISKIOを使用していない)

→プロセス単位の動的優先処理が必要

基本方針

- カーネル側はRedHawkの基本機能を利用
  →カーネルの変更はしない
  - RedHawkリアルタイムカーネル機能
    - Memory lock
    - Real time scheduling & priority
    - User code Preempt lock (Only RedHawk)
    - Shield CPU(Only RedHawk)
- QEMU側のみリアルタイム化する

- バージョンアップに備えて基本的な構造は変えない

モディファイ1

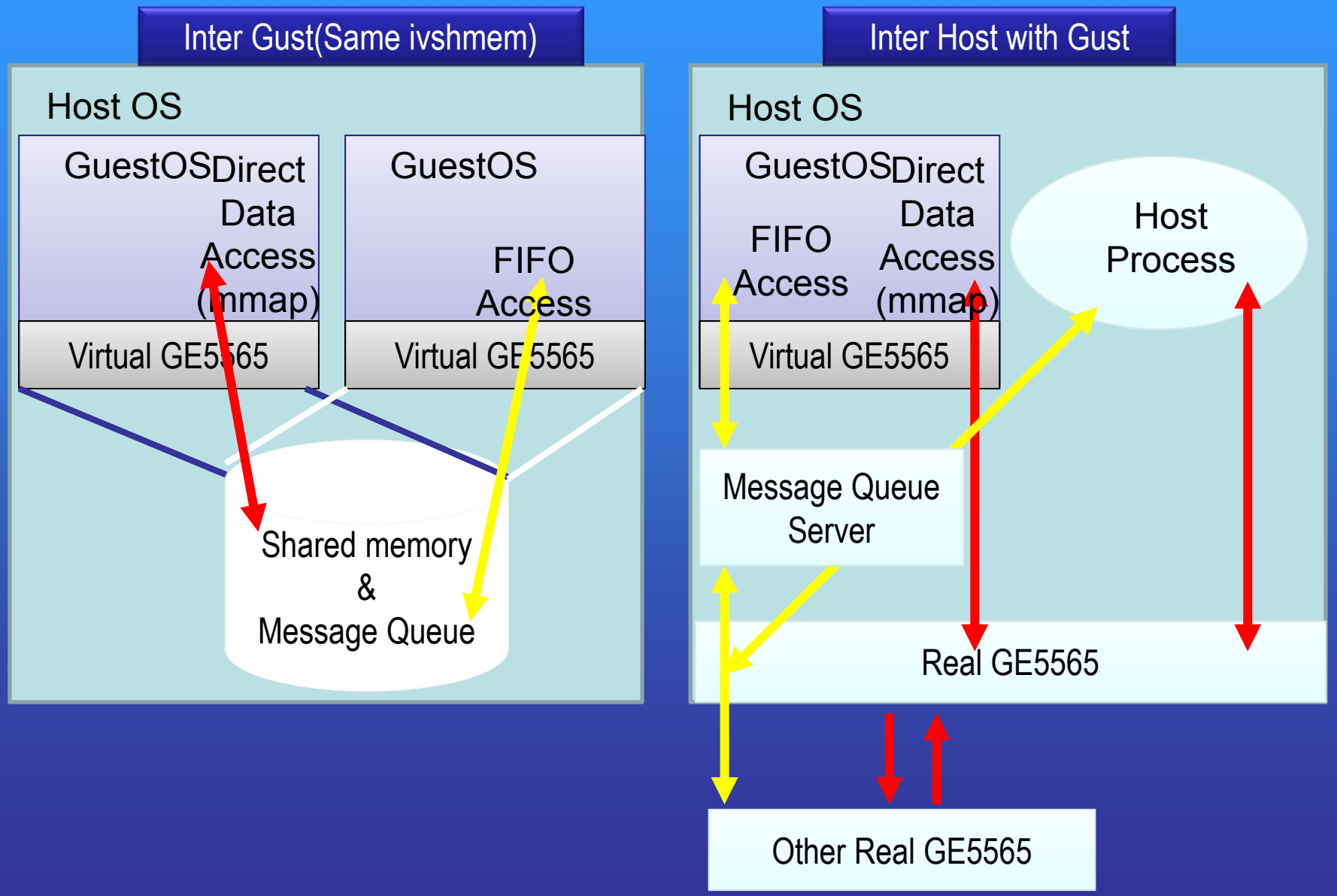
- リアルタイム化
  - QEMUスレッド別の制御データ
    - スケジュール
      SCHED_FIFO,SCHED_RR,SCHED_OTHER
    - 優先度　　0(TS) or 1～99(RR or FIFO)
    - Nice値　　-19～+20(TS or RR)
      値でタイムクオンタムが異なる
      → 連続実行可能なCPU時間
    - CPUアフィニティ
  - メモリロック
  - プリエンプション禁止（QEMU連続実行を保証）
    - ハードウェア模擬部分のレイテンシィ確保には必須

モディファイ2

- 模擬デバイスのモディファイ
  - ivshmem
  - Guest-Hostの通信(再起動通知、負荷率取得など)
    - ivshmemを利用して、guest側から再起動通知を行う。
    - RT-QEMU側では、再起動通知期間中、Guestの優先度を強制的にTSに設定する。
  - Guest間の通信
- 新規模擬デバイス(vrfmem)
  - リフレクティブメモリ(5565)模擬デバイス
  - 実デバイスとリンクし異なるホストにまたがるHost-Guest間通信を可能

concurrent

Inter virtual shared memory の構造



モディファイ3

- VNC部高速化
  - スレッド生成(標準搭載:configure vnc_thread=“yes”)
  - mainスレッドで実行される、画面差分処理の高速化
    - memcmp() → sse2_memcmp()
    - memcpy() → sse2_memcpy()
  - mainスレッドで実行される、画面差分処理の並列化
    - for (y=0;y<SIZE;y++)　を2分割し高速化

-rt オプション

- オプション -rtに続く最初の xxxx ４文字はスレッドを現し、
  その後の　_yyyyは機能を現す
  - vcpu　　仮想CPU
  - main　　管理スレッド
  - paio　　非同期IOスレッド
  - vncm　　vnc メイン側スレッド (注1)
  - vncs　　vnc スレーブ側スレッド(注２)
    - 注1：vnc高速化スレッドパラメータ、無指定の場合には、ループ分割しない
    - 注2：configureで指定されるオリジナルスレッドパラメータ

その後の文字で、以下を現す。

- xxxx_sched　　スケジュール、
- xxxx_prio　　優先度
- xxxx_nice　　nice値

-rt オプション例

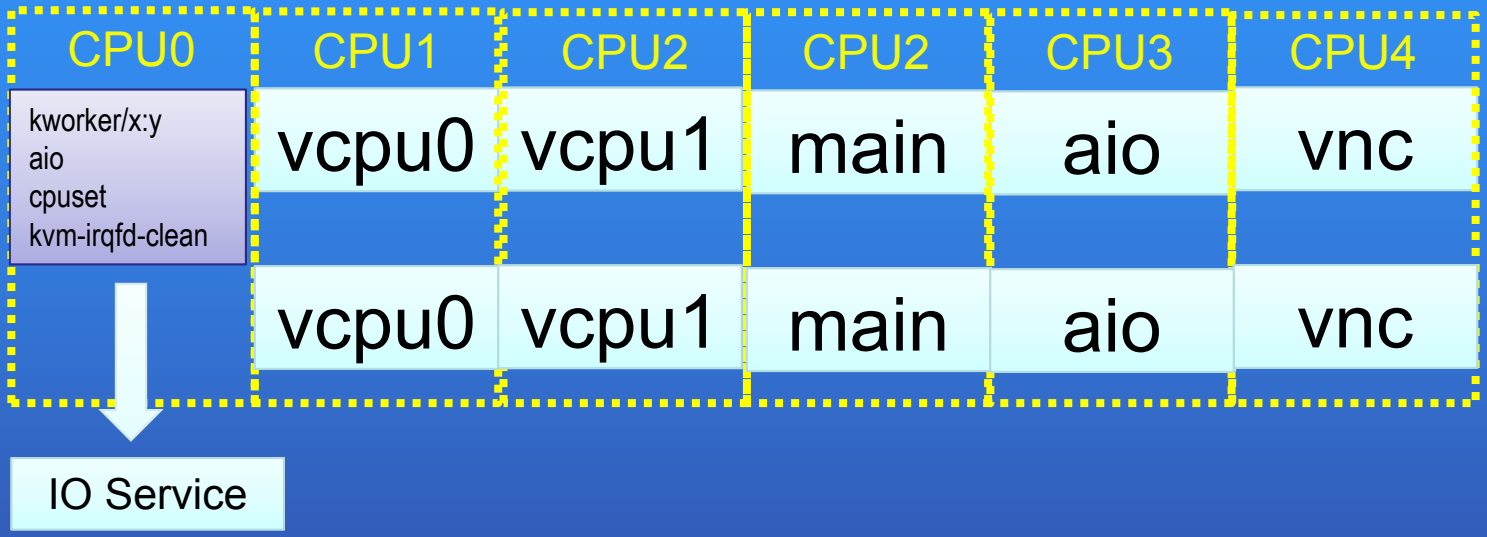
- -rt vcpu_bias=3,4 -rt vcpu_sched=fifo,fifo -rt vcpu_prio=1,1 -rt vcpu_nice=0,0
- -rt main_bias=5 -rt main_sched=fifo -rt main_prio=1 -rt main_nice=0
- -rt paio_bias=6 -rt paio_sched=rr -rt paio_prio=1 -rt paio_nice=19
- -rt vncs_bias=7 -rt vncs_sched=rr -rt vncs_prio=1 -rt vncs_nice=19
- -rt vncm_bias=7 -rt vncm_sched=rr -rt vncm_prio=1 -rt vncm_nice=19

-20	800ms	-10	600ms	0	100ms	10	48ms
-19	780ms	-9	580ms	1	92ms	11	44ms
-18	760ms	-8	560ms	2	88ms	12	40ms
-17	740ms	-7	540ms	3	84ms	13	32ms
-16	720ms	-6	520ms	4	80ms	14	28ms
-15	700ms	-5	500ms	5	72ms	15	24ms
-14	680ms	-4	480ms	6	68ms	16	20ms
-13	660ms	-3	460ms	7	64ms	17	12ms
-12	640ms	-2	440ms	8	60ms	18	8ms
-11	620ms	-1	420ms	9	52ms	19	4ms

nice値テーブル

QEMUスレッドスケジュール

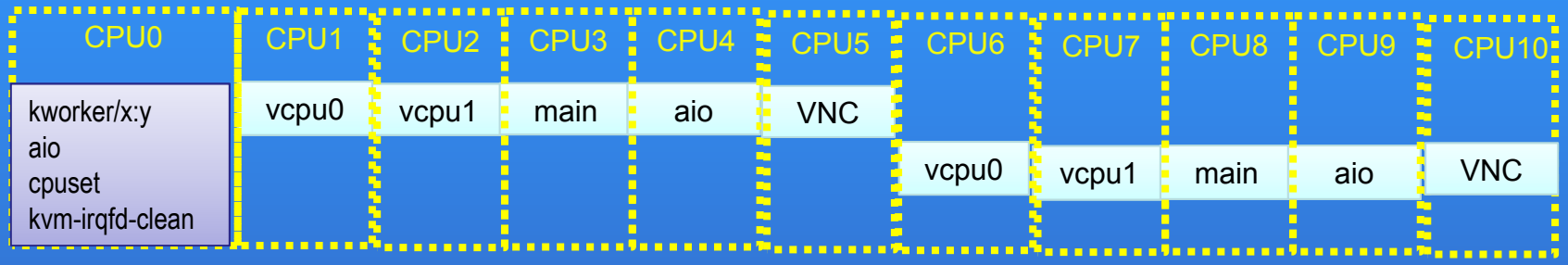
- スレッド別に割り付けが可能



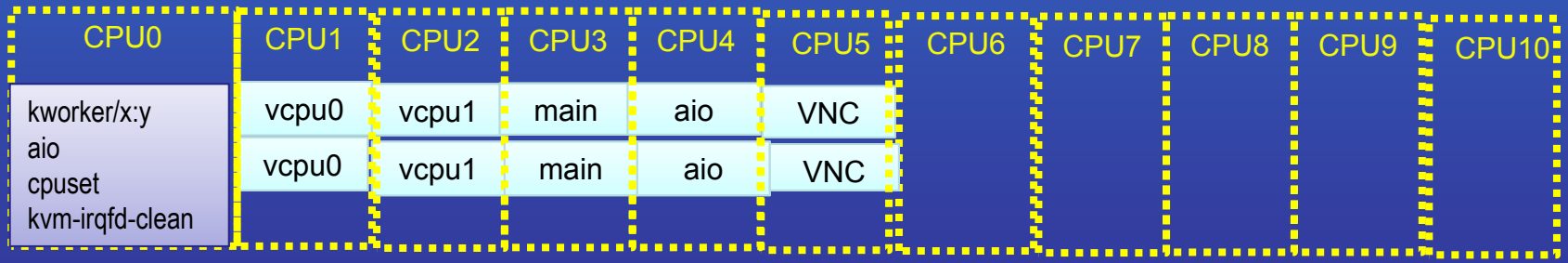
- スレッド別にスケジュール＆優先度を設定
- スレッド別にnice値を設定可能(タイムクオンタム)

QEMUスレッドスケジュール

- 完全分離型　FIFO (最も高速に応答)
  - サービスの応答性を重視

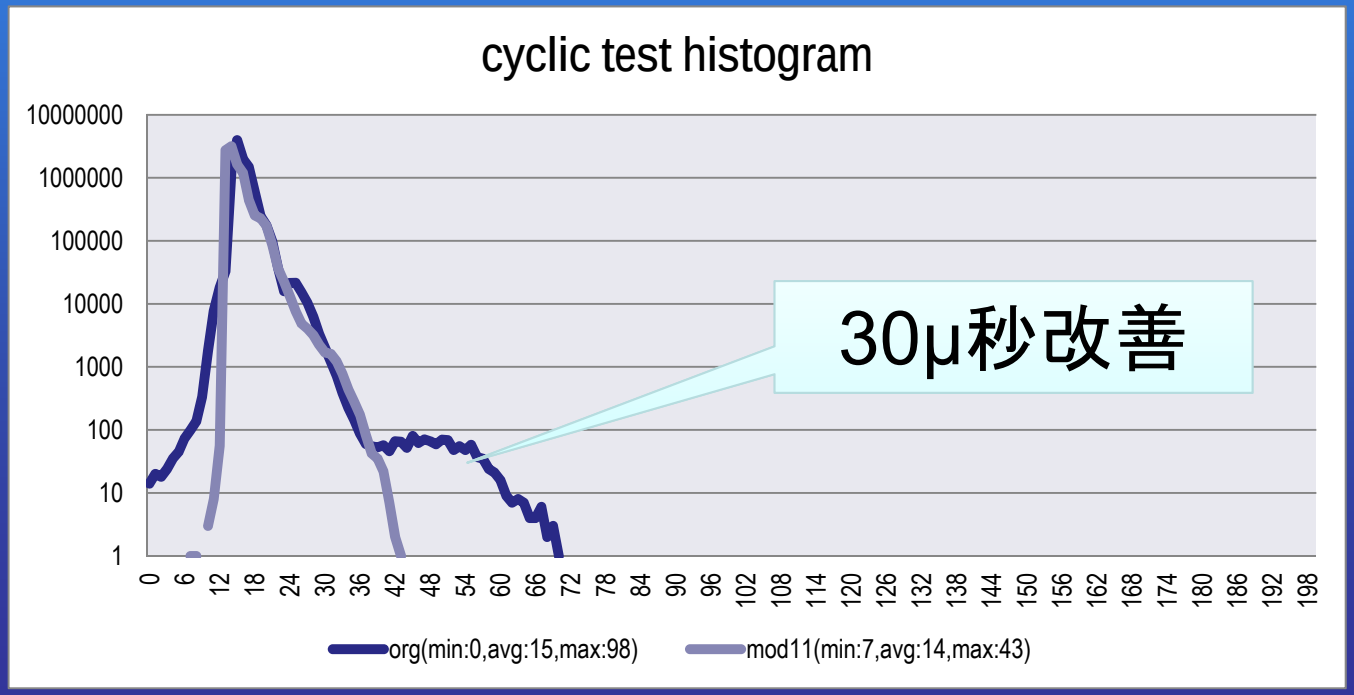


- 完全共有型　RR (Nice値でサービス時間を制御)
  - サービスの公平性を重視



ベンチマークテスト１

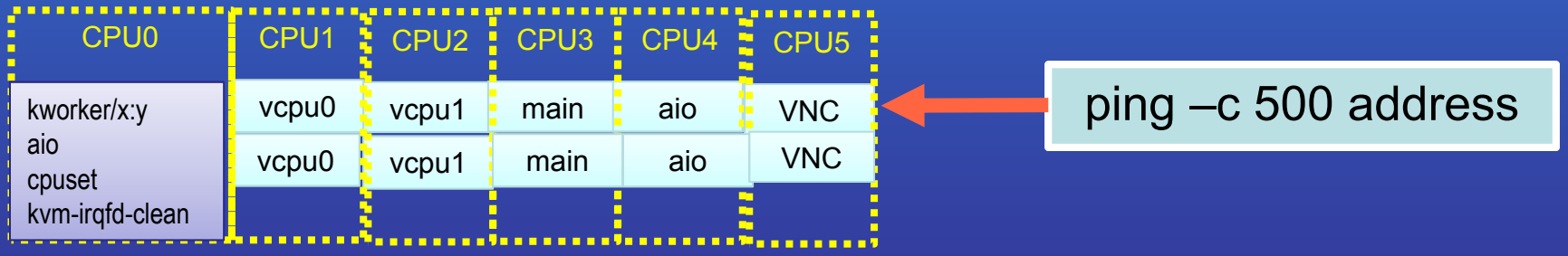
- スレッド制御の効果を試験
  ホスト側で同一CPUに負荷を与え、guest側でcyclic testを行う(stress --cpu 8 --io 4 --vm 2 --vm-bytes 128)
  - org:スレッドアフィニティなし優先度あり
  - mod:スレッドアフィニティあり優先度あり



ベンチマークテスト２

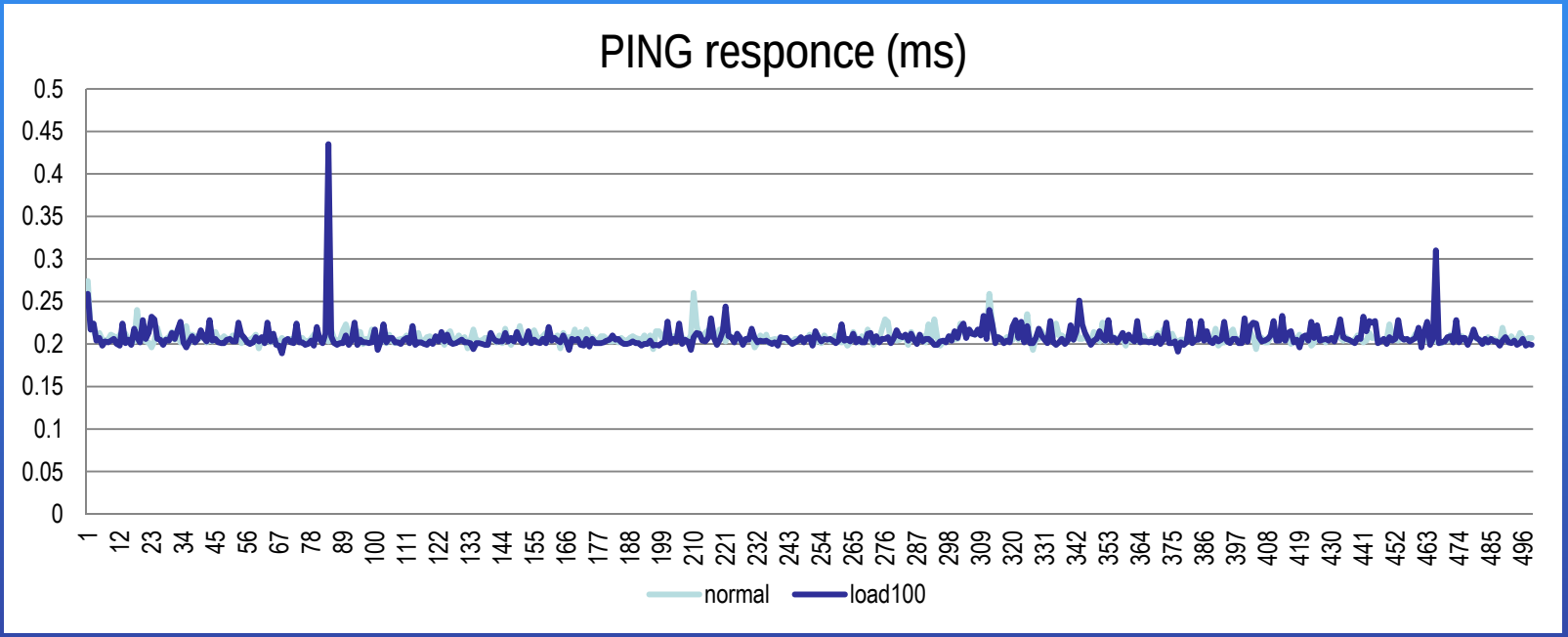
- 完全共有型の試験
  同一コアにGuest1とGuest2を公平に割り付け
  外部システムからGuest1にPINGを500回行う
  - Normal test: Guest1,Guest2とも無負荷
  - Load test　: stress –cpu2 をGuest2で実行し、Guest2のCPU負荷100%×2にする

→総てのスレッドは、固定優先度1のクオンタム4ms(RR,1,NICE 19)



ベンチマークテスト２(続き)

- ネットワークIOレイテンシィ （500回のpingの応答時間)

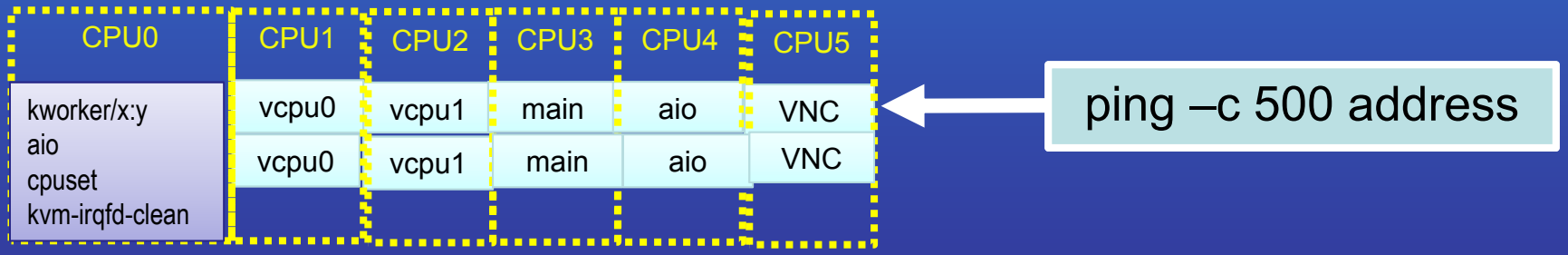


- Pingの応答時間に大きな変化は無い。

ベンチマークテスト３

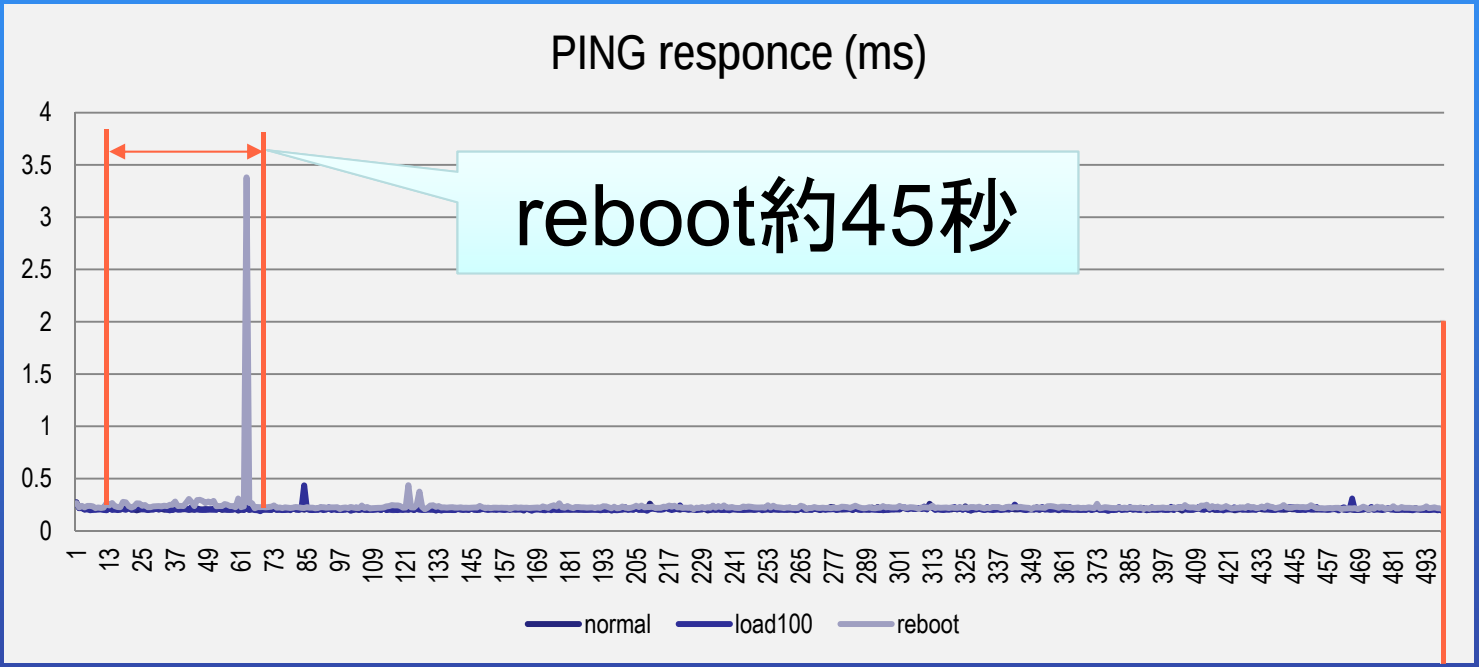
- 完全共有型の試験
  同一コアにGuest1とGuest2を公平に割り付け
  外部システムからGuest1にPINGを500回行う
  - Normal test: Guest1,Guest2とも無負荷
  - Load test　: Guest2をrebootする

→総てのスレッドは、固定優先度1のクオンタム4ms(RR,1,NICE 19)



ベンチマークテスト４

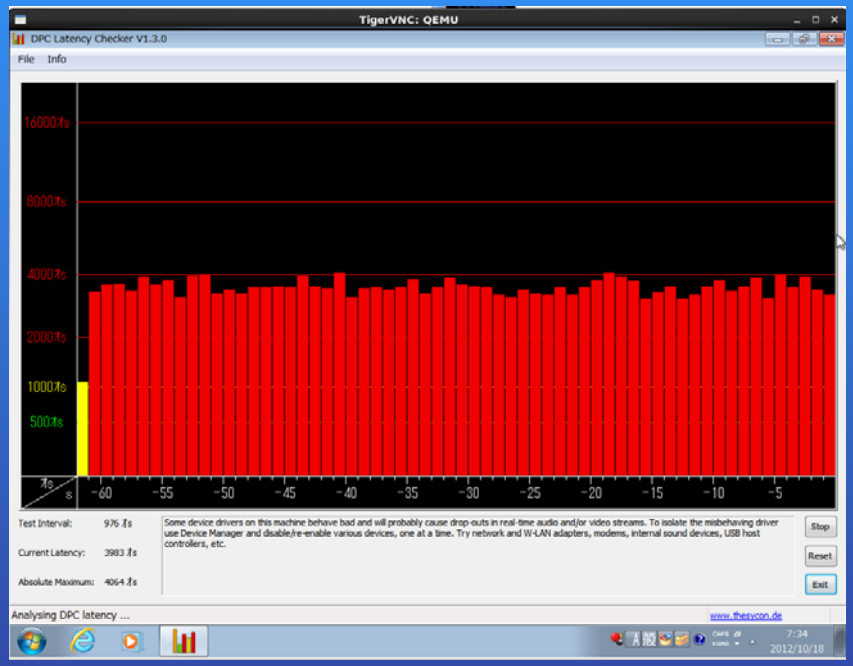
- ネットワークIOレイテンシィ （500回のpingの応答時間)

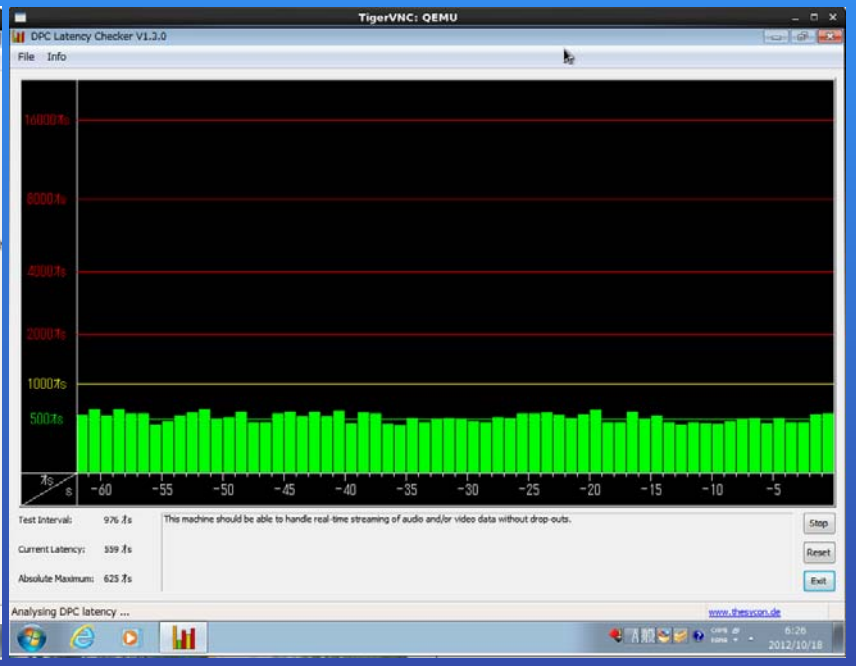


- Pingの応答時間は、4ms以下(nice19の設定)
- Pingの応答時間が大きく遅い箇所は、Guestネットワークの構成時１回だけ

ベンチマークテスト５

- Windows7 & DPC Latency Checker





オリジナル版　　RealTime QEMU

まとめ

- RT-QEMUの特徴
  - RedHawk Realtime Linuxカーネルの機能を利用し、QEMUのみのモディファイでリアルタイム化。
  - カーネルのバージョンアップに迅速に対応可能。
  - Guest側カーネルの変更も無い
  - 十分な応答性を確保
  - 用途に応じて、細かなチューニングが可能