I/O帯域制御とIA64-Xenの現状

山幡 為佐久 <yamahata@valinux.co.jp>
VA Linux Systems Japan K.K.

Agenda

- IO帯域制御
- Xen/IA64の現状

IO帯域制御

IO帯域制御

- 稲越　宏弥(富士通)、高橋浩和(弊社)及び鶴田亮(弊社)らによる仕事

本研究・開発の一部は，経済産業省の委託を受けた技術研究組合 超先端電子技術開発機構(ASET) のセキュア・プラットフォームプロジェクトの成果です.

背景

- 仮想化環境下でディスクIOはあまり良く隔離されていない。
  - あるゲストがディスクIOを大量に行なうと他ゲストのディスクIOに悪影響がある。
  - 特に同一ディスクを使用しているものでは悪影響が顕著
  - ハイエンドストレージであっても別LUに分けてあっても悪影響がある場合も
    - LU毎に帯域を設定できたりするものもあるが、高価。
  - ioniceでは不十分
- => ディスクIO制限を実装しよう

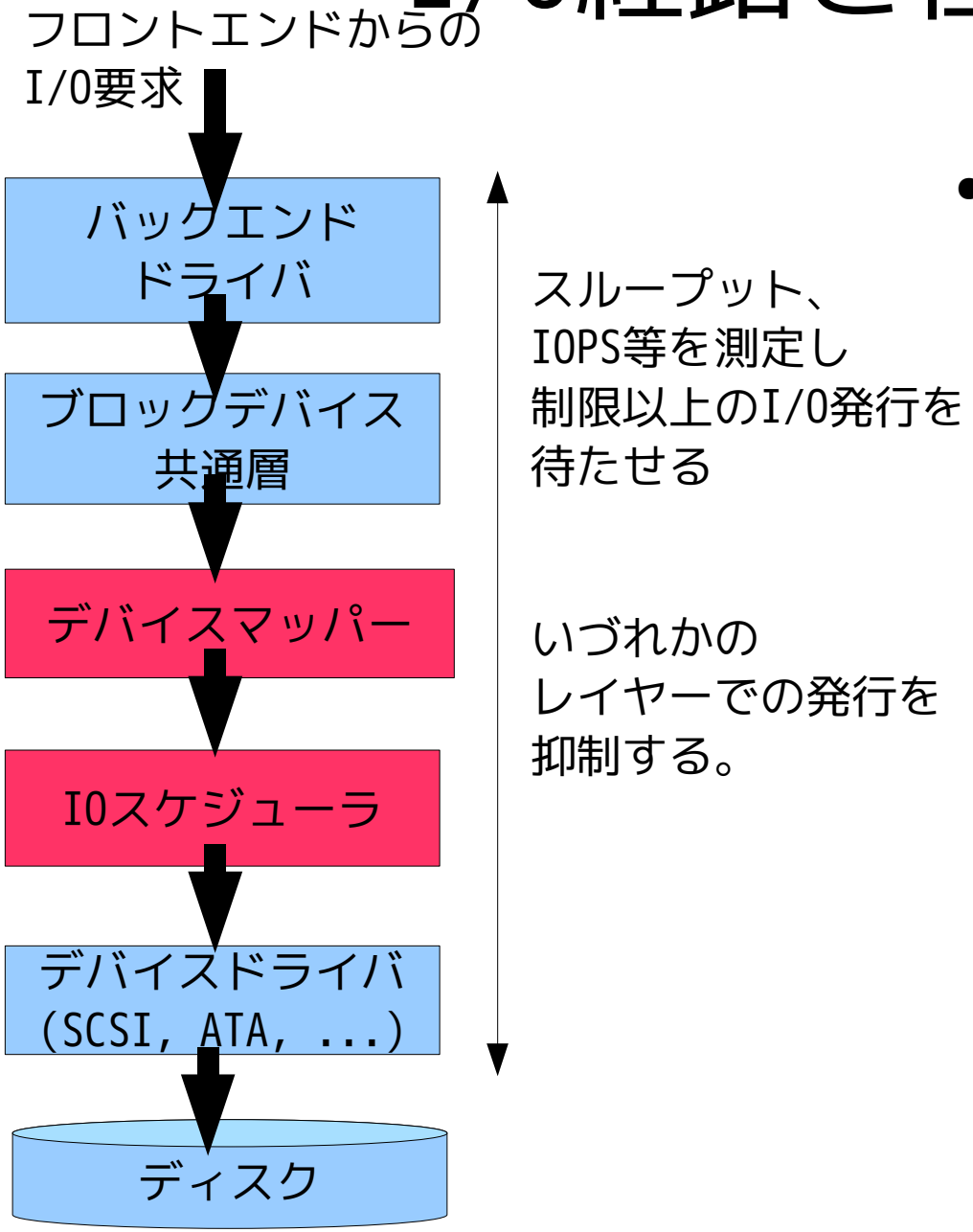
実装方針

- 準仮想化／完全仮想化両方のドメインに適用可能
  - 適用範囲が広くなるよう、なるべく汎用的に
- SAN環境での使用も想定
  - ハイエンドストレージでLUが異なっていても適用可能
  - 複数ブロックデバイスのグループに対しても適用可能であるような柔軟性が必要

実装方針(cont.)

- 特定のIOスケジューラに依存しない。
  - 現在CFQが主流だけど、別のがでてくるだろう。
    - 実際、BFQが提案されている。
  - ハイエンドストレージには noopを使う。
- Direct IO/write backの両方に適用可能
  - Cgroupを利用
    - =>Guest VMはcgroupにマップ
  - Cgroup memory controlerを利用
- IO制限方式はポリシーとして分離

I/O経路と性能制限



- どのレイヤー/サブシステム？
  - PV backend
  - Specific IO scheduler
  - デバイスドライバ
  - IO scheduler framework
  - Device Mapper

I/O経路と性能制限(cont.)

- 汎用性が不十分
  - PV backend
  - Specific IO scheduler
  - デバイスドライバ

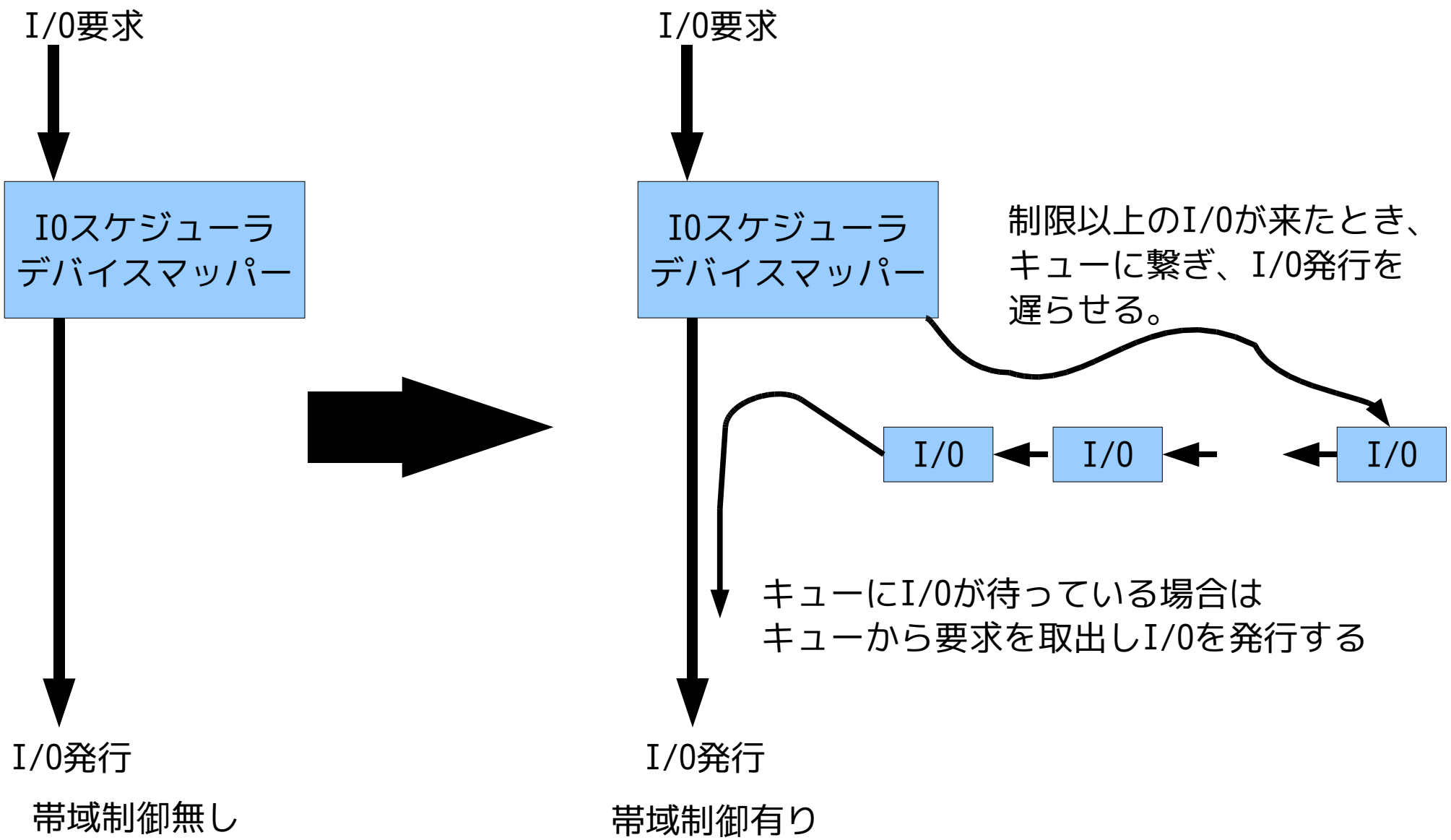
I/O経路と性能制限(cont.)

- IO scheduler framework
  - Pros
    - 設定が楽
  - Cons
    - 影響範囲大
- Device Mapper
  - Pros
    - 影響範囲小
  - Cons
    - 余計にdeviceをstackする必要がある(root deviceの時、特に問題)

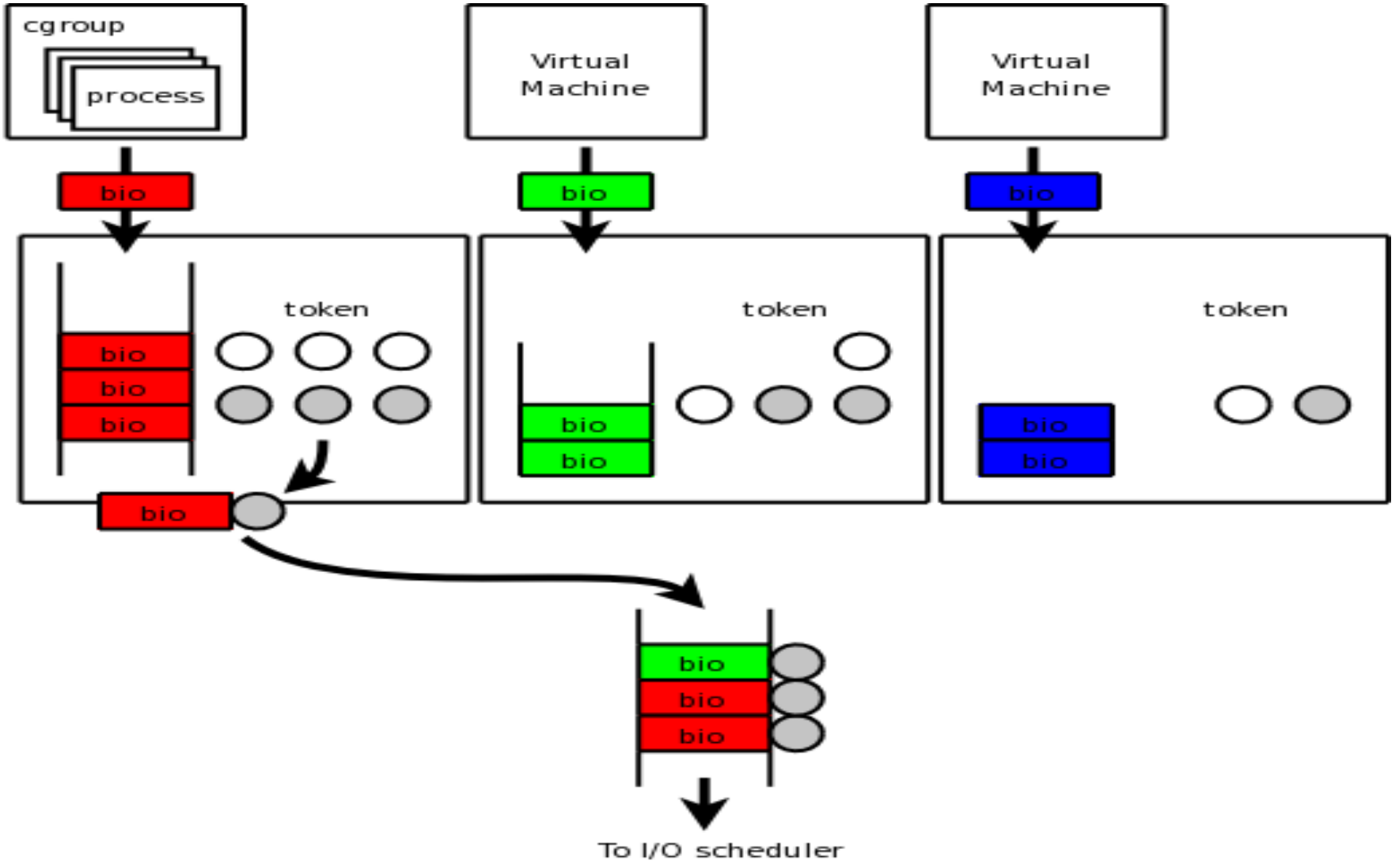
I/O経路と性能制限(cont.)

- => Device Mapperに
- 複数デバイスを管理できる柔軟性あり
  - IO scheduler frameworkでは難しい。
- 手っ取り早く実装できそう
  - に思えたのですが、、、
  - いろいろ助言をいただいたが、、、
  - 現在、IO controlerは乱立状態に。まだ、合意は見込めず。

性能制限方式


I/O要求
IOスケジューラ
デバイスマッパー
I/O発行
帯域制御無し
I/O要求
IOスケジューラ
デバイスマッパー
制限以上のI/Oが来たとき、
キューに繋ぎ、I/O発行を
遅らせる。
I/O
I/O
I/O
キューにI/Oが待っている場合は
キューから要求を取出しI/Oを発行する
I/O発行
帯域制御有り

Token based I/O throttling



課題

- IO制御
  - Heuristicに依存するのでいろいろなベンチマークを行なって検証が必要。

  => 頑張ってベンンチマーク
- 遅延書き込み
  - ページに書き込んだ人を覚えておく必要がある。

  => memcgを利用

  => bio-cgroupを実装して追跡

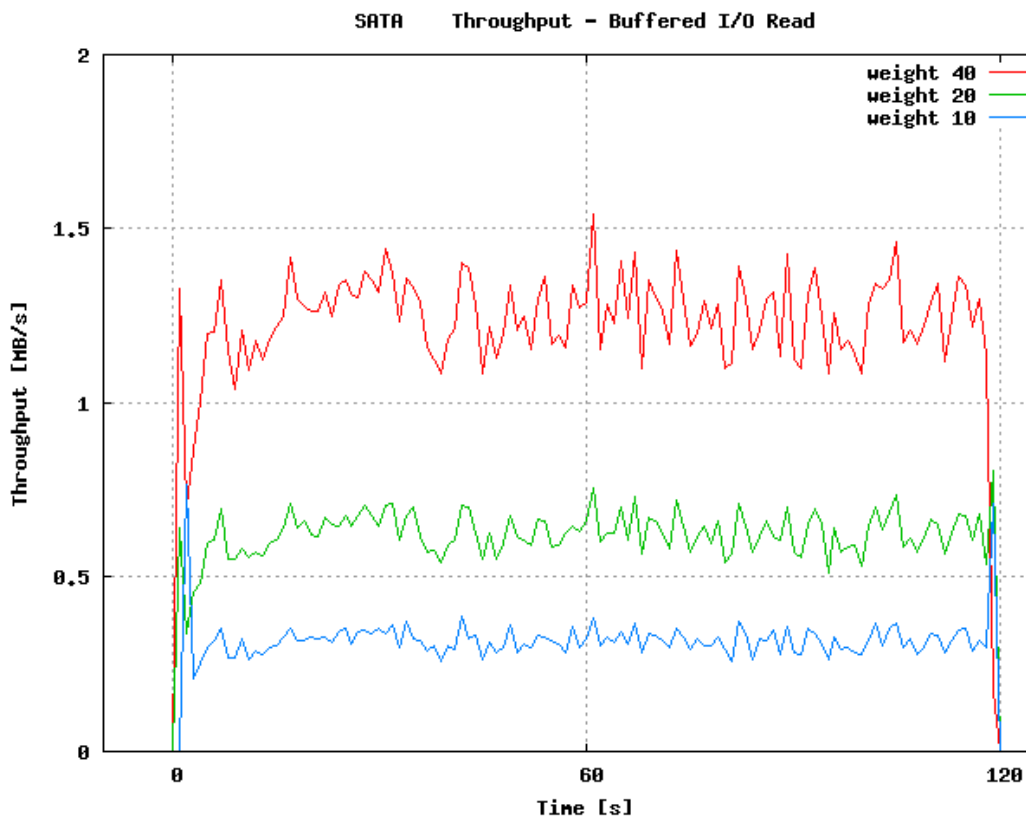
現在の実装状況

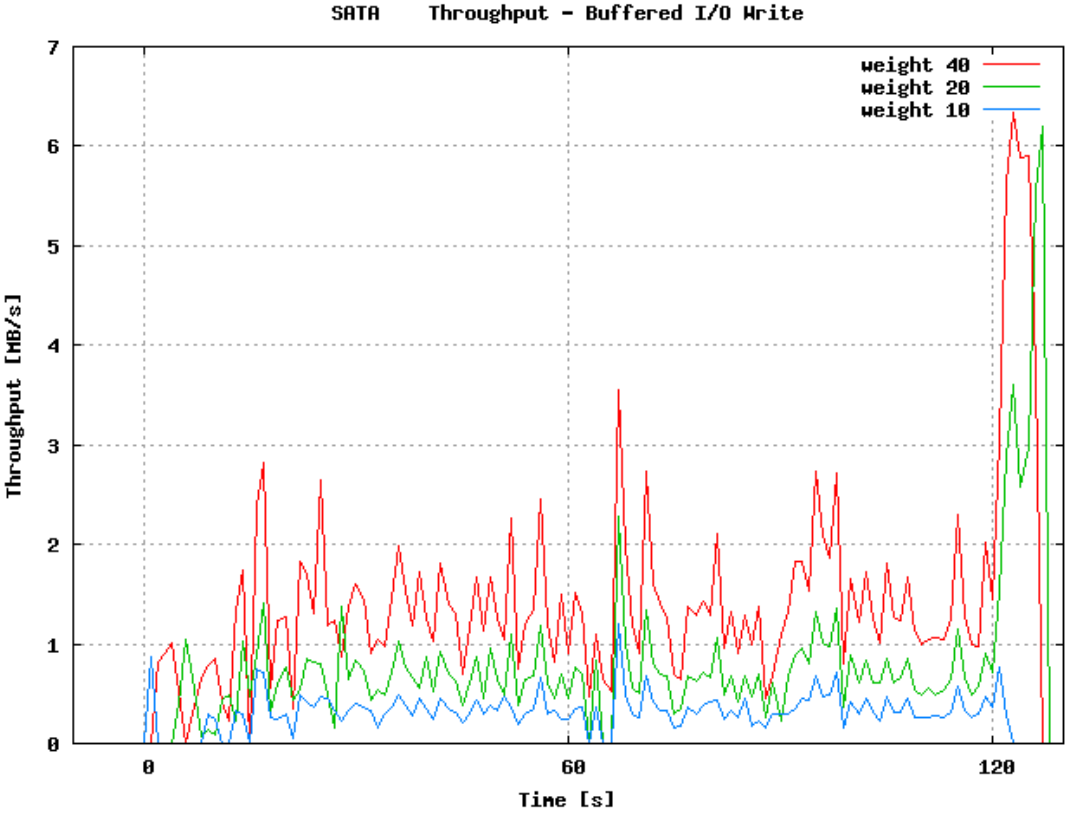
- Dm-ioband
  - 実装は安定
  - スループット劣化は殆どなし
  - マージできる品質
- Bio-cgroup
  - memcg再実装に合わせて、bio-cgroup再実装中。I/O追跡機能のオーバヘッド削減

ベンチマーク

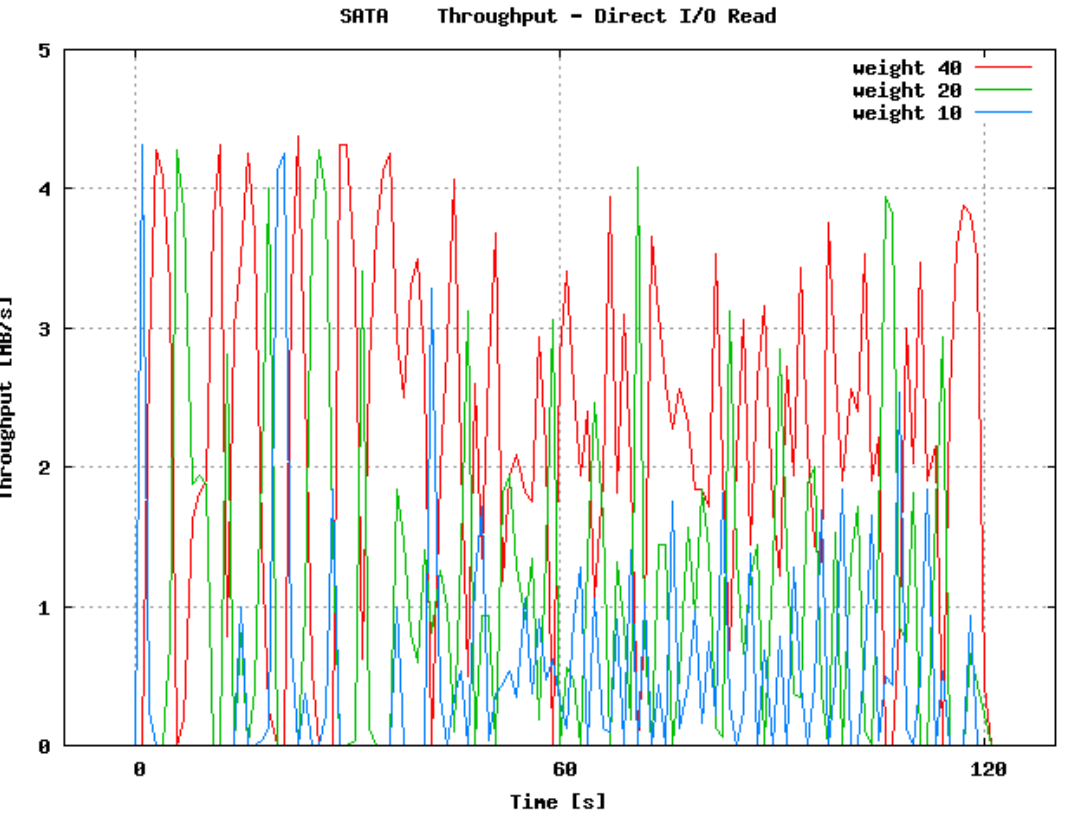
- いろいろなものでベンチマークを行なった
  - SATA
    - Western Digial WD2500JS SATA2 2700rpm 8MB buffer
    - PC とは 1.5Gbps で接続
  - High-end storage
    - ミッドレンジ　FC-SAN(4GB cache, 4GBps Fiber Channel)
    - データは全てキャッシュに載った状態で測定
  - SSD
    - スループット公称値 Read 700MB/s、Write 600MB/s

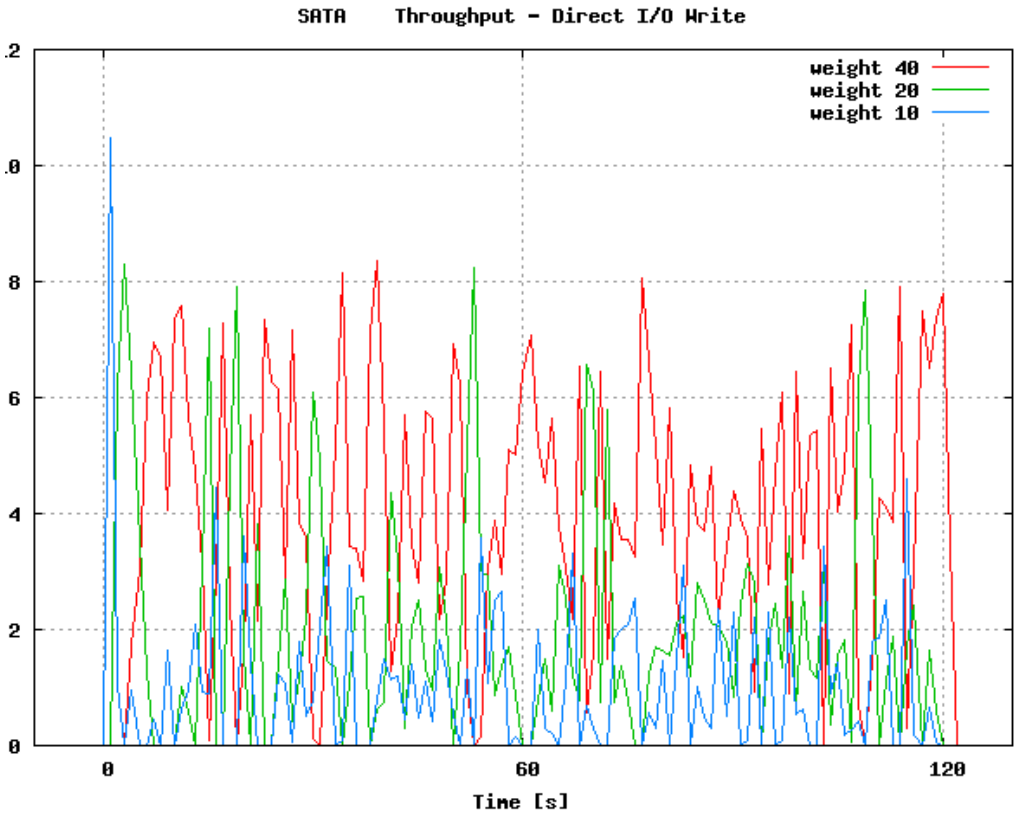
SATA Buffered I/O



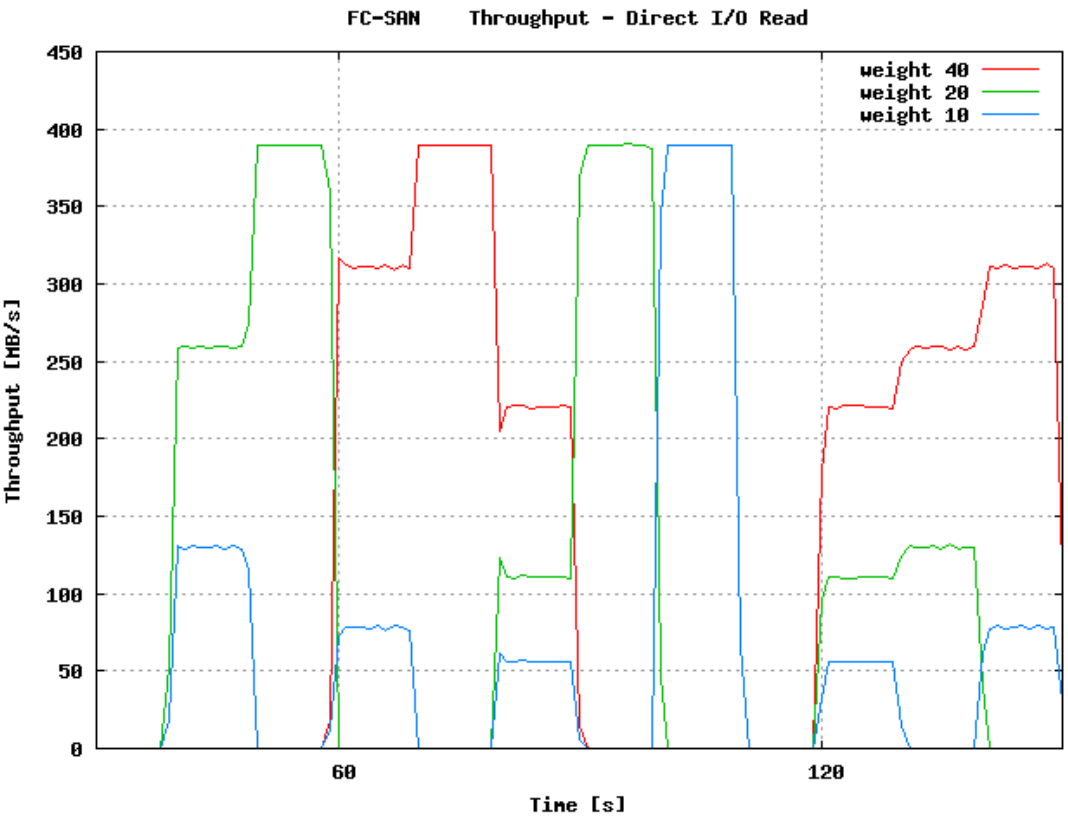


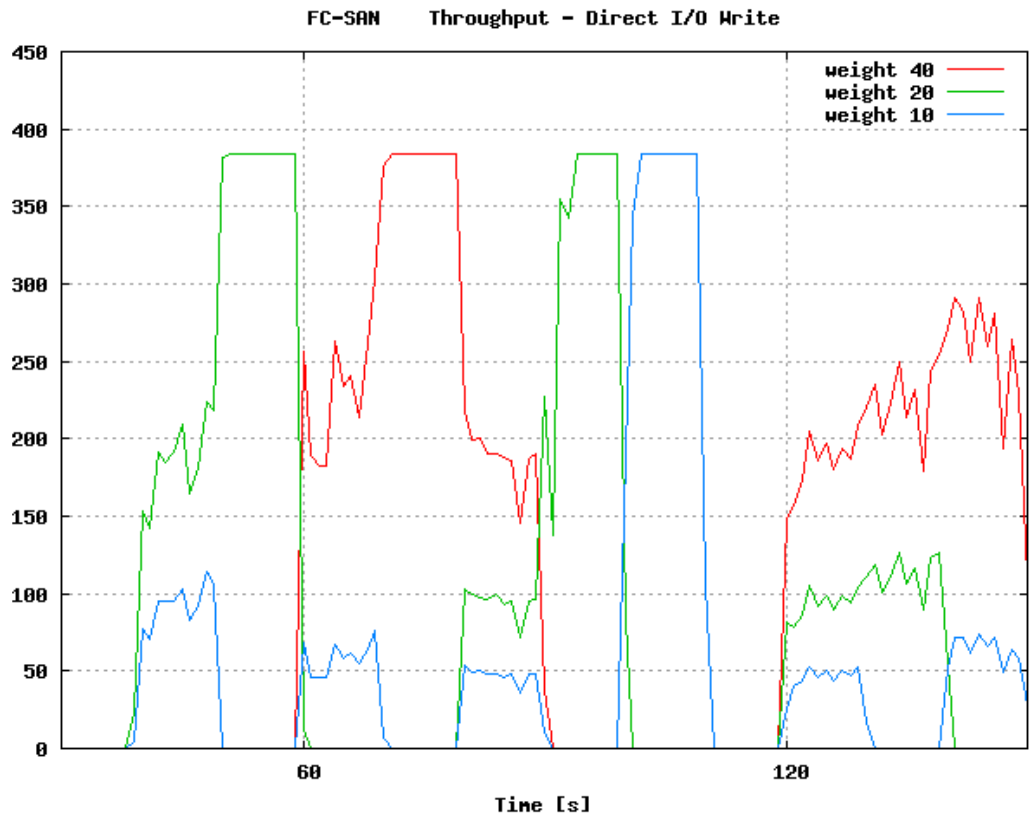
SATA Direct I/O



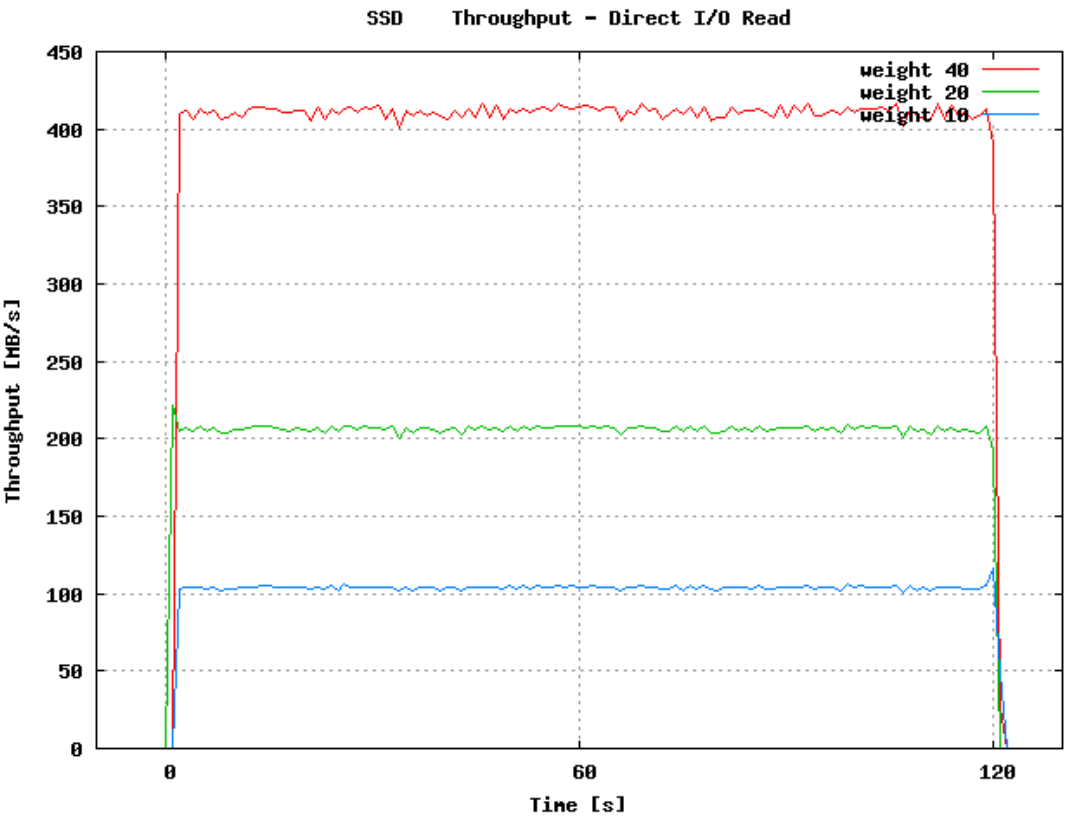


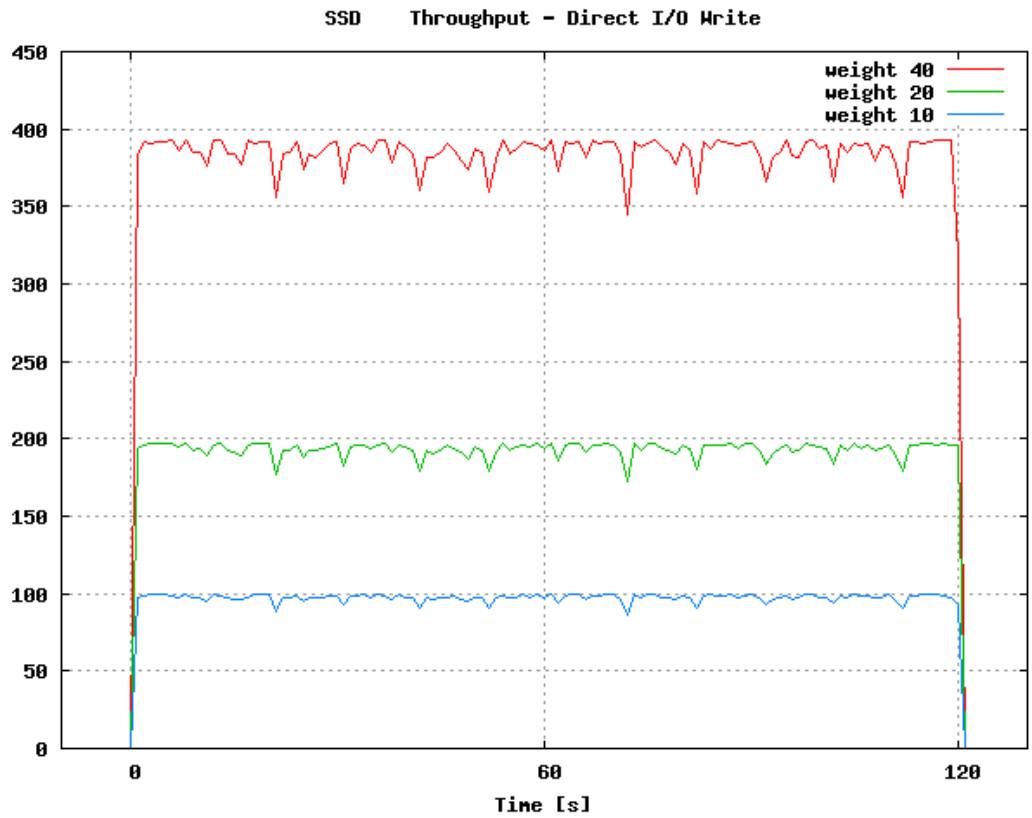
Highend Storage Direct I/O





SSD Direct I/O





FAQ

- 帯域制御がきいていないみたいだけど、、、
  - ウェイトでの帯域制御
    - デバイス A と B で帯域を共有している場合、A のみに I/O 負荷を与えると A が全ての帯域を使う。A と B 同時に負荷を与えると割り当てた帯域に従う。レートリミットする帯域制御ポリシを dm-ioband に追加することも可能。
  - I/O の並列度が一定以上にならないと帯域制御を始めない。
    - デフォルトでは並列度 4 以上で帯域制御を行う。動作確認には fio や xdd 等のマルチプロセス/マルチスレッドで I/O を行うベンチマークツールを使うと良い。

Future work(1)

- Upstreamへのマージ
  - 手詰り気味?
  - Device mapperにマージされれば主導権はとれそうだけど
- dm-iobandアルゴリズムのブロックデバイス共通層への直接実装
  - 試作版作成中

Future work(2)

- IO controlerの合意をとりたい。
  - IO schedulerのレイヤーに移植する？
- bio-cgroup
  - dm-ioband以外からの利用
- I/O帯域制御ポリシーの追加
  - スループット保証ポリシー
  - 遅延保証ポリシー

Xen/IA64の現状

Xen/IA64

- X86以外のアーキテクチャで唯一実用レベルになっている
  - Xen/POWER
  - Xen/Arm
- 主な機能はほぼx86と同等

主なトピック

- HVM domain save/restore/live-migration
- SIOEmu: Self IO Emulation
  - ia64独自の機能
- メンテナー交替（2008年5月6日）
- kexec/kdump
  - EFI mapping
- 安定性の向上

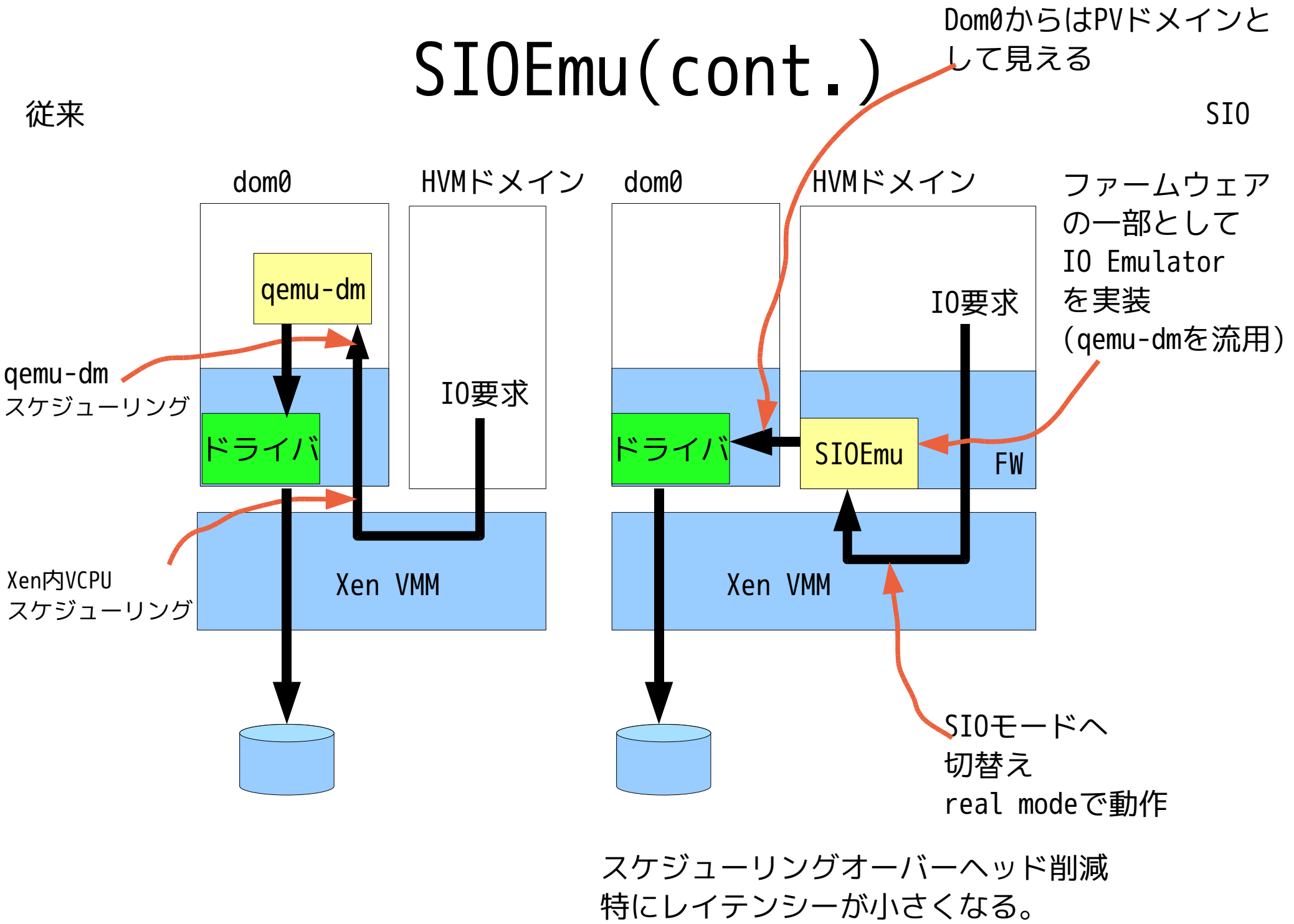
主なトピック(cont.)

- paravirt_ops/ia64(Work in progress)
  - Linux上流へのマージ
- Catching up x86 changes
  - Remote-qemu (qemu-xen)
  - VRAM allocation
  - Cpufreq
  - VT-d

SIOEmu

- Self IO EMUlation
  - Qemu-dmを対象ドメイン内で動作させる仕組み
  - Tristan Gingoldが開発
  - IA64 MMUのモード切替えを活かしている。
  - X86で同様の事をやろうとするとMMUの切替のオーバーヘッドが大きいと思われる。
    - => stub domain

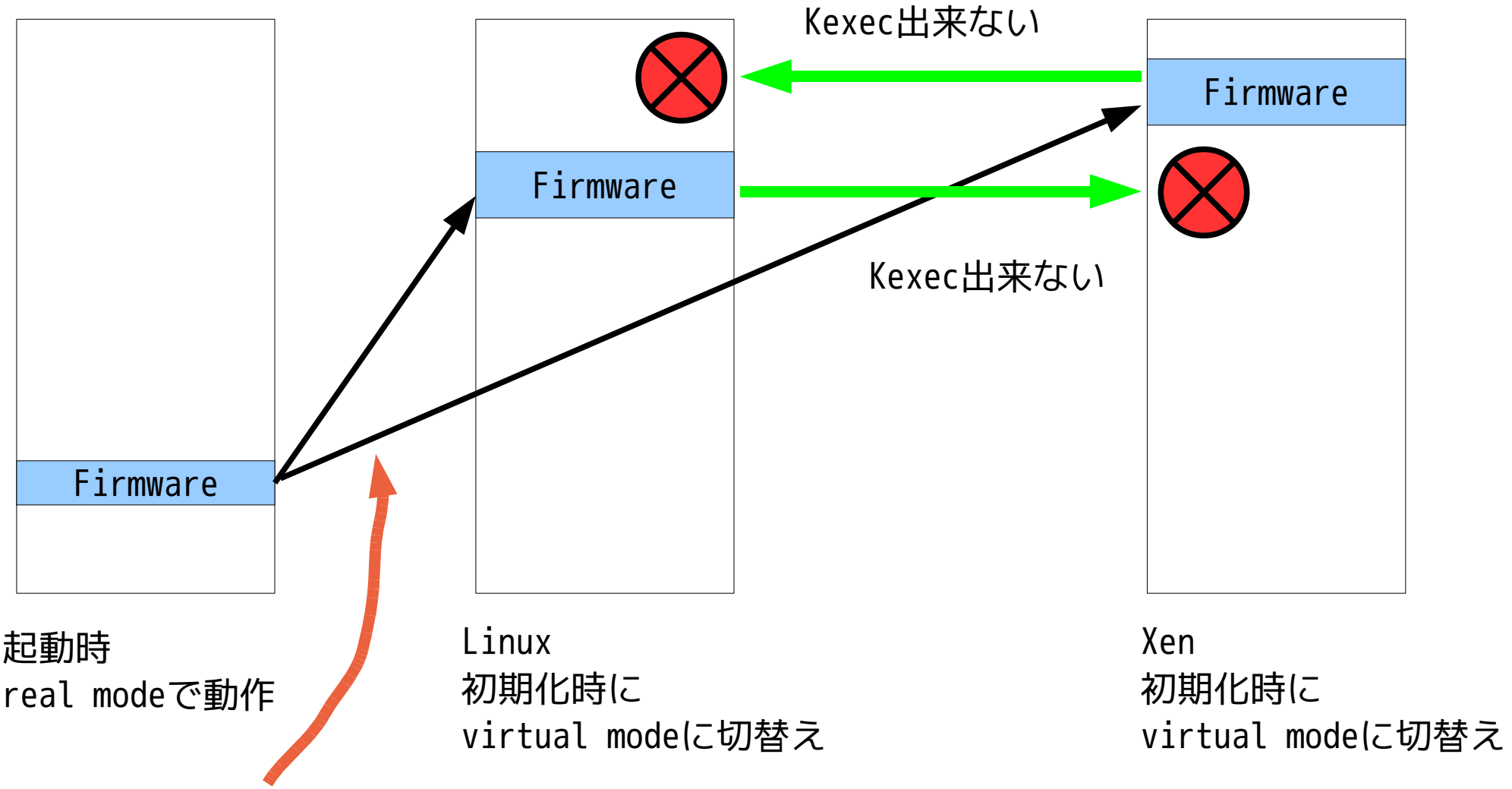
SIOEmu(cont.)


Dom0からはPVドメインとして見える
従来
SIO
dom0
HVMドメイン
dom0
HVMドメイン
ファームウェアの一部としてIO Emulatorを実装（qemu-dmを流用）
qemu-dm
qemu-dm
スケジューリング
IO要求
IO要求
ドライバ
ドライバ
SIOEmu
FW
Xen内VCPU
スケジューリング
Xen VMM
Xen VMM
SIOモードへ切替え
real modeで動作
スケジューリングオーバーヘッド削減
特にレイテンシーが小さくなる。

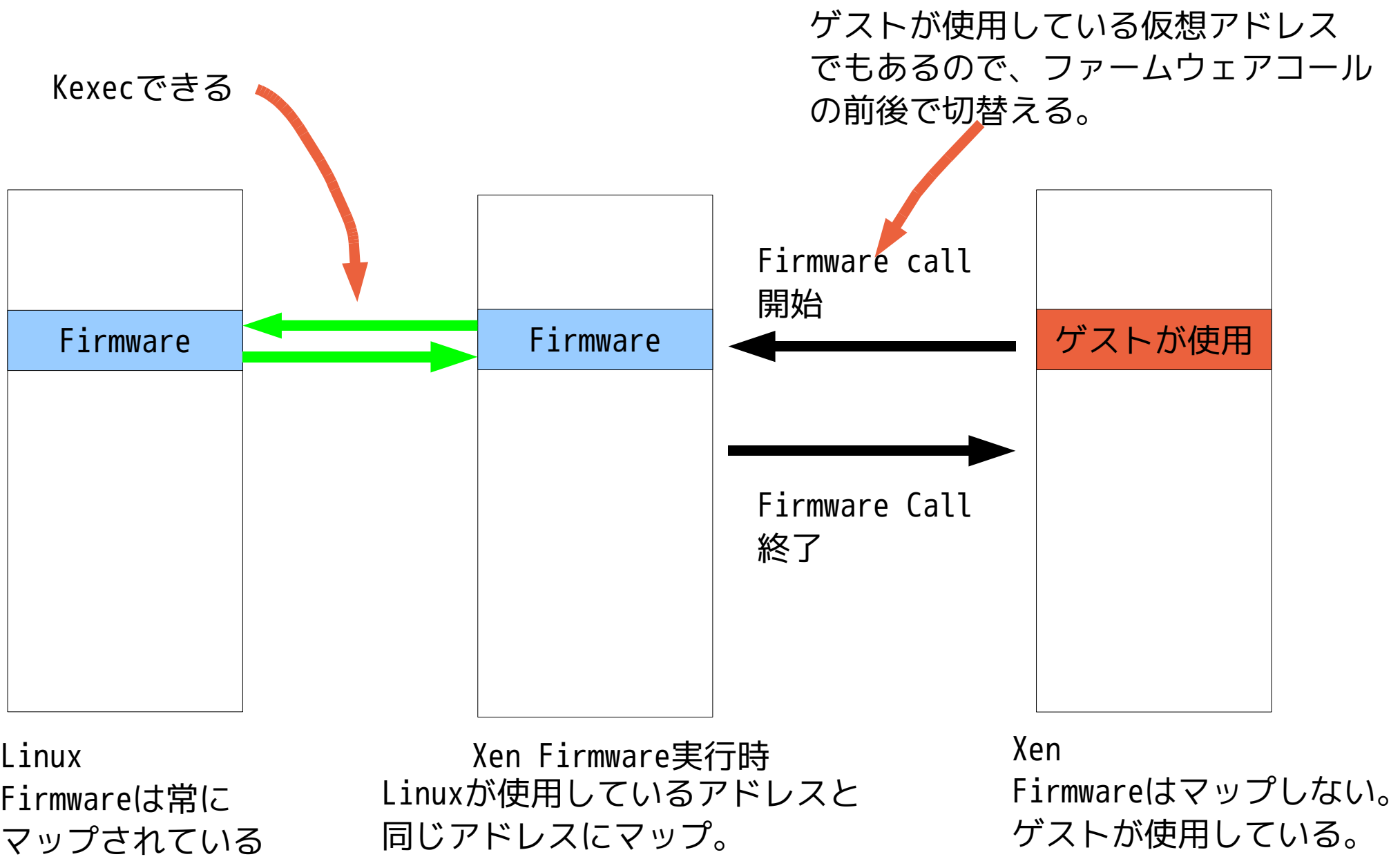
Kexec/Kdump

- EFI mapping
  - EFI
    - 起動時real modeで動作、一度だけvirtual modeに変更可能
    - LinuxとXenではマップするアドレスが違っているのでXen->Linux, Linux->Xenのkexecが出来ない。
    - Linuxと同じアドレスにマップ
    - ファームウェアコールの前後で空間切替え
  - X86でもUEFIをサポートすれば問題になるはず。

Kexec/Kdump(cont.)

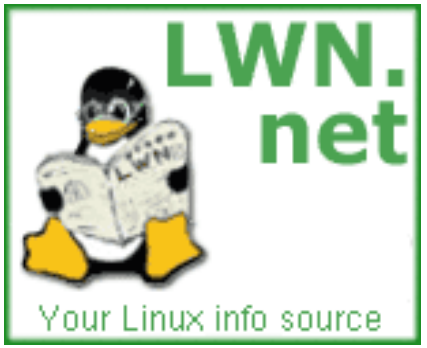


Kexec/Kdump(cont.)



paravirt_ops/ia64

- http://wiki.xensource.com/xenwiki/XenIA64/UpstreamMerge
- 2008年1月頃から議論が始まる
- 2008/3月/28日 ia64/pv_ops コミット
  - Minimal pv_ops
  - 2.6.27
- 2008/10月/17日 ia64/xen コミット
  - Minimal DomU
  - 2.6.28



The 2.6.28 merge window closes

[Kernel] Posted Oct 24, 2008 8:40 UTC (Fri) by corbet

Linus has released 2.6.28-rc1 and closed the merge window for this development cycle. Changes merged since the previous summary include IA64 Xen support, ultrawideband radio (UWB) support with wireless USB support on top of it, range timers, and some extensive changes to the block driver API. Also, this kernel has been named "Killer Bat of Doom."

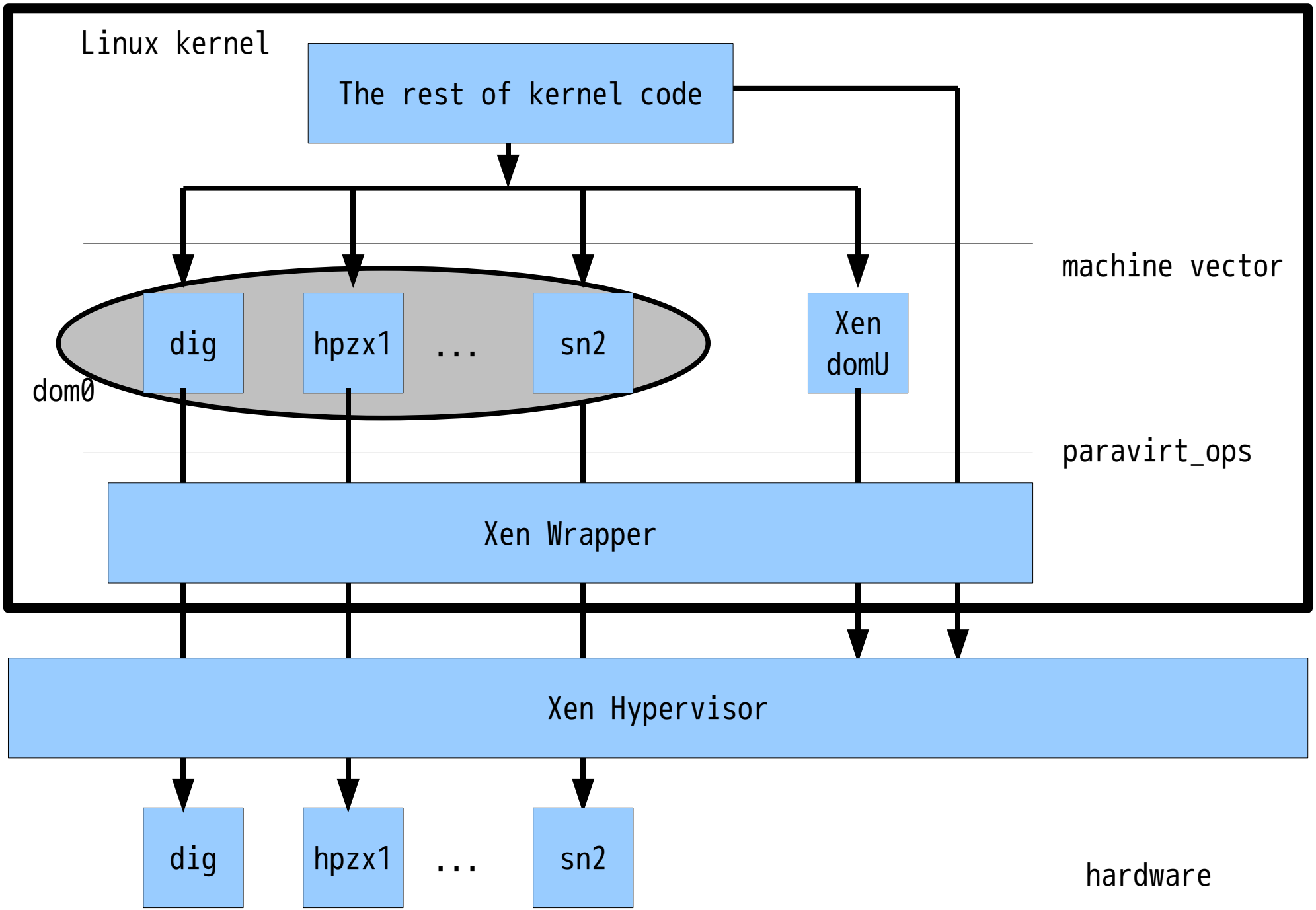
Comments (22 posted)

課題

- IA64 machine vector
- Privileged instruction (para)virtualization
- Binary patching
- Hand-written assembly code
  - Multi compile
  - PV checker

IA64 machine vector

- IA64は既にmachine vectorと呼ばれる層があった。
- DomUの場合はxen machine vector + paravirt_opsに
- Dom0の場合はmachine vectorを若干書き換える必要がありそう。


Linux kernel
The rest of kernel code
machine vector
dig
hpzx1
...
sn2
dom0
Xen
domU
paravirt_ops
Xen Wrapper
Xen Hypervisor
dig
hpzx1
...
sn2
hardware

Instruction (Para)Virtualization

- IA64の特権命令の準仮想化方式の検討
  - Pre-Virtualization and after-burning
  - Hybrid Virtualization. ハードウェアサポート(VT-i)を前提にする。
  - Privify. (vBlades方式)
  - paravirt_ops.
- => paravirt_ops方式に

Binary Patch

- 特権命令準仮想化方式にparavirt_opsを使用する為必要となる。
  - 関数呼出オーバーヘッド削減
    - 特にnativeの場合の性能低下を防ぐ為に必要
    - nativeの場合は(ほぼ)1命令に対応
  - IA64の場合、gccインラインアセンブリでは関数呼出が記述できない
  - =>関数呼出オーバーヘッドが問題になる特権命令準仮想化(pv_cpu_ops)のみ対応させる。
  - =>呼び出される関数はアセンブリで記述

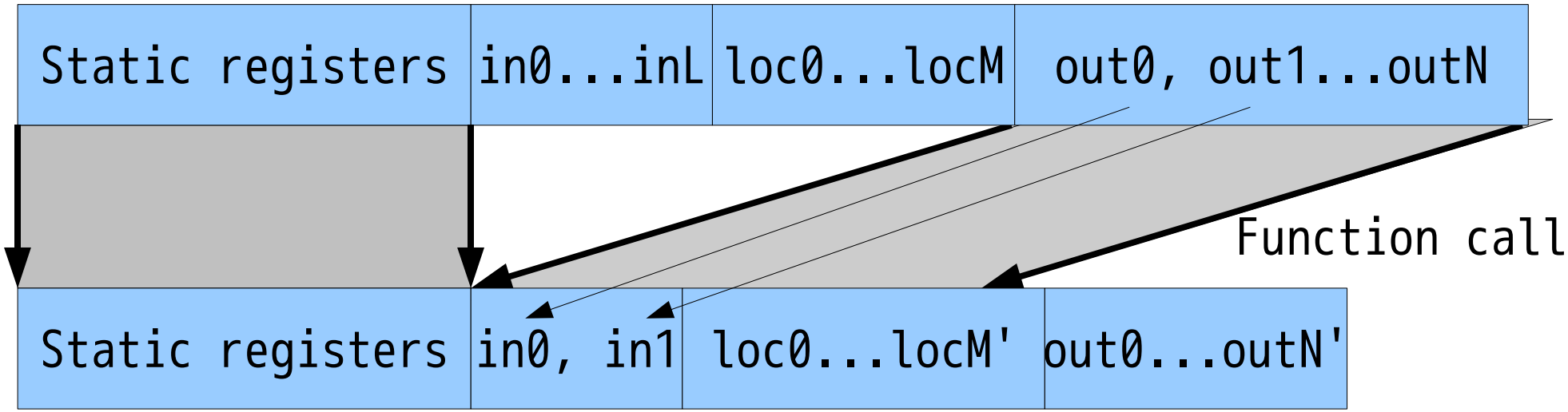
```
caller_func(in0, ... inL)
    loc0, ... locM
    out0, ... outN

    bp_func(out0, out1)
    other_func(out0, out1, ... outN)
```

asm(“...”::: “out0”, “out1”)では破壊されるレジスタが全て記述できない。

func(in0,..., inL)

r0-r31　　M, N are determined by GCC

Clobbered registers



r0-r31　　M', N' are determined by GCC

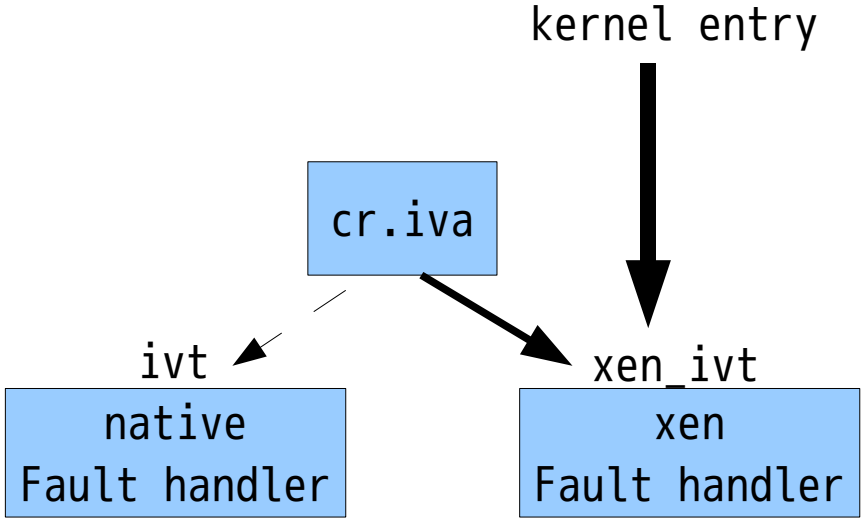
bp_func(in0, in1)

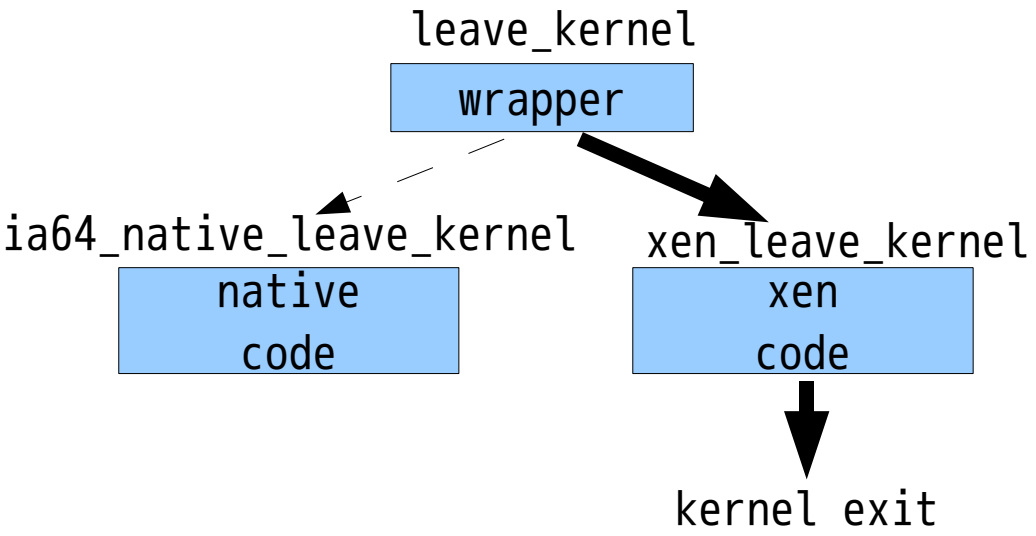
Hand-Written Assembly Code

- .Sファイル
- カーネルの入口と出口及びコンテキストスイッチ
- スタックは使えない場合も多い
- 極度に最適化されているので命令順序すら触るのは難しい
- => マルチコンパイル方式に

実行コードの切替え

- Fault handlerはcr.ivaレジスタで切替え.
- その他のコードは間接ジャンプを追加して切替え





マルチコンパイル

- 一つのファイルをCPP定義を変えて何度もコンパイルする。
- ほぼ1命令に1マクロが対応

例

Patch

```
-(p8) mov cr.itir=r25
+     MOV_TO_ITIR(p8, r25, r24)
```

Native

```
#define MOV_TO_ITIR(pred, reg, clob)    \
(pred)  mov cr.itir = reg               \
        CLOBBER(clob)
```

Xen

```
#define MOV_TO_ITIR(pred, reg, clob)    \
(pred)  movl clob = XSI_ITIR;           \
        ;;                              \
(pred)  st8 [clob] = reg
```

余分に使うレジスタはコードを注意深く読んで見つける。

PVチェッカー

- .Sファイル準仮想化は壊れやすい
  - マクロを使う必要がある所に直接命令を書いてしまうなど。
- 簡単なチェッカを実装
  - 全ての場合を検出しないけが、実用上十分と思われる。
  - 実際CONFIG_SMP=nの時、検出してLinux/ia64メンテナに修正要求をされた。

例: cover命令

```
#define COVER   nop 0
#define cover   .error "cover should not be used directly."
```

paravirt_ops/ia64 future work

- domUの場合の最適化
  - More .S file paravirtualization
  - save/restore
  - Binary patch
  - Balloon expantion
    - Memory hot add
  - Spin lock(?)

paravirt_ops/ia64 future work (cont.)

- Dom0
  - Ioremap
  - DMA
  - pci
  - kexec/kdump
  - xenoprof
  - mca

質疑応答

ご静聴ありがとうございました。