NetBSDのネットワーク機能を近代化する

尾崎亮太
中原健吾

IIJ Technical WEEK 2015
2015年11月13日

発表概要

- IIJとNetBSD
- NetBSDのネットワーク機能のMP化
  - NetBSDのネットワーク機能の概説
  - レイヤ2転送のMP化
  - 割り込み処理のMP化
  - 性能評価結果
- 今後の取り組みとまとめ

SEIL

- ルータ(サービスアダプタ)
  - SMF sx/ SACMによる集中管理
- SEILシリーズ
  - neu, Turbo
  - B1
  - X1/X2
  - x86 Fuji
- SEILアプライアンスシリーズ
  - BPV4
- その他のサービスアダプタ
  - SA-W1
    - レシピフレームワーク
    - mruby

SEILとNetBSD

- SEIL、SEILアプライアンスシリーズ
  - NetBSD 3ベース
    - 2009年にEOL
  - 必要な機能やセキュリティフィックスをバックポート
- SA-W1
  - NetBSD 6ベース
    - まだ現役
  - 定期的に本家と同期
- MP化作業
  - NetBSD-currentベース
    - 最新開発版

NetBSDとは？

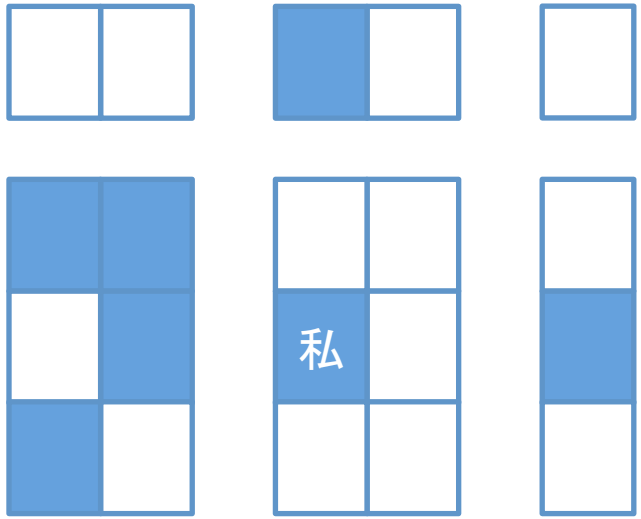
- BSD Unixの直系の子孫
  - 1993年ぐらいに4.3BSDからfork
  - 詳しい系譜はbsd-family-treeというファイルに書かれている
    - FreeBSDと兄弟ぐらいの関係
    - OpenBSDのfork元
- 多くのプラットフォーム・アーキテクチャをサポートしている
  - クロス開発がしやすい
  - 古いコンピュータもサポート
    - DEC VAX, Commodore Amiga, SHARP X68030, Apple (old) Macintosh, OMRON LUNA, SEGA Dreamcast
- BSDライセンス

NetBSDとIIJ

- NetBSDを独自に改良し利用
- できる範囲で本家へフィードバック
  - sh3
  - ドライババグフィックス
- 最近はより積極的にコミットしている(後述)

NetBSD developers in IIJ

- NetBSD developerとは、NetBSDのソースコードレポジトリにコミット権を持つ開発者
  - 一般的にはコミッタと呼ばれる人
- IIJにはNetBSD developerが多い



私の座席周辺のNetBSD developer

IIJの最近の貢献

- Cavium Octeon (MIPS)
- IRQ affinity, intrctl(8)
- MSI/MSI-Xサポート(x86)＆ドライバ対応
  - wm(4), iwm(4), vioif(4)
- RXマルチキュー
  - wm(4)
- ネットワークドライバMP化(カーネルロックなしで動く)
  - wm(4), vioif(4), vmx(4)
- PCI Extended Configuration Space
- DTrace for ARM
  - BEAGLEBONE, SHEEVAPLUG
- bridge(4) MP化
- ATFテスト (ネットワークスタック向け)
- cvs2git : NetBSDソースコードのgit変換ツール＆gitミラーサイト
  - https://github.com/IIJ-NetBSD/netbsd-src
- その他ネットワーク周りのコード整理・改善

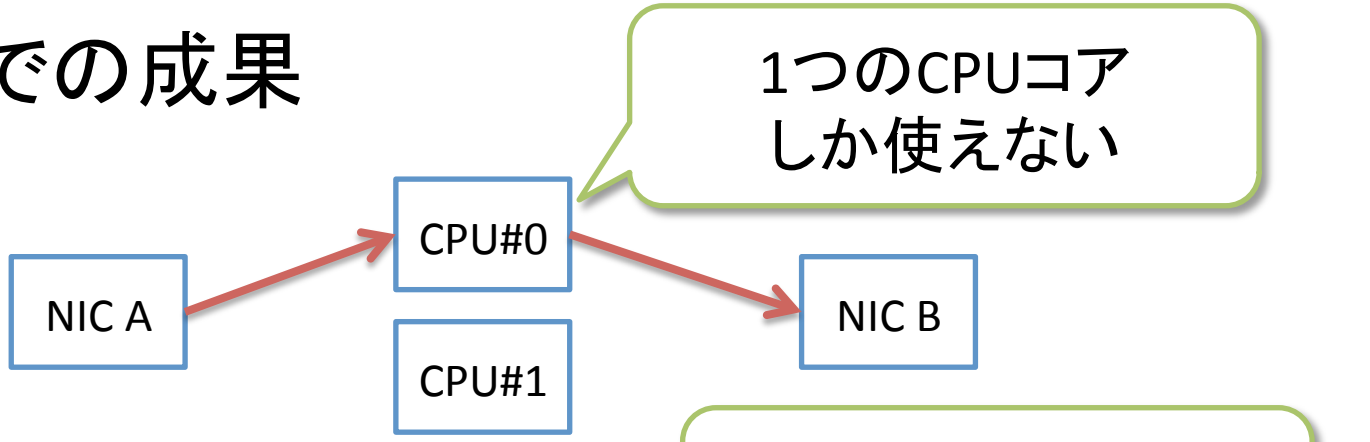
IIJの現在の取り組み

- NetBSDのネットワーク機能をMP化(*)し、パケット処理性能を向上する
  - (*) マルチプロセッサ化：プロトコルスタックやデバイスドライバが、複数のCPUコア上で並列に動作できるように、各機能を改良すること
  - ターゲットはパケット転送

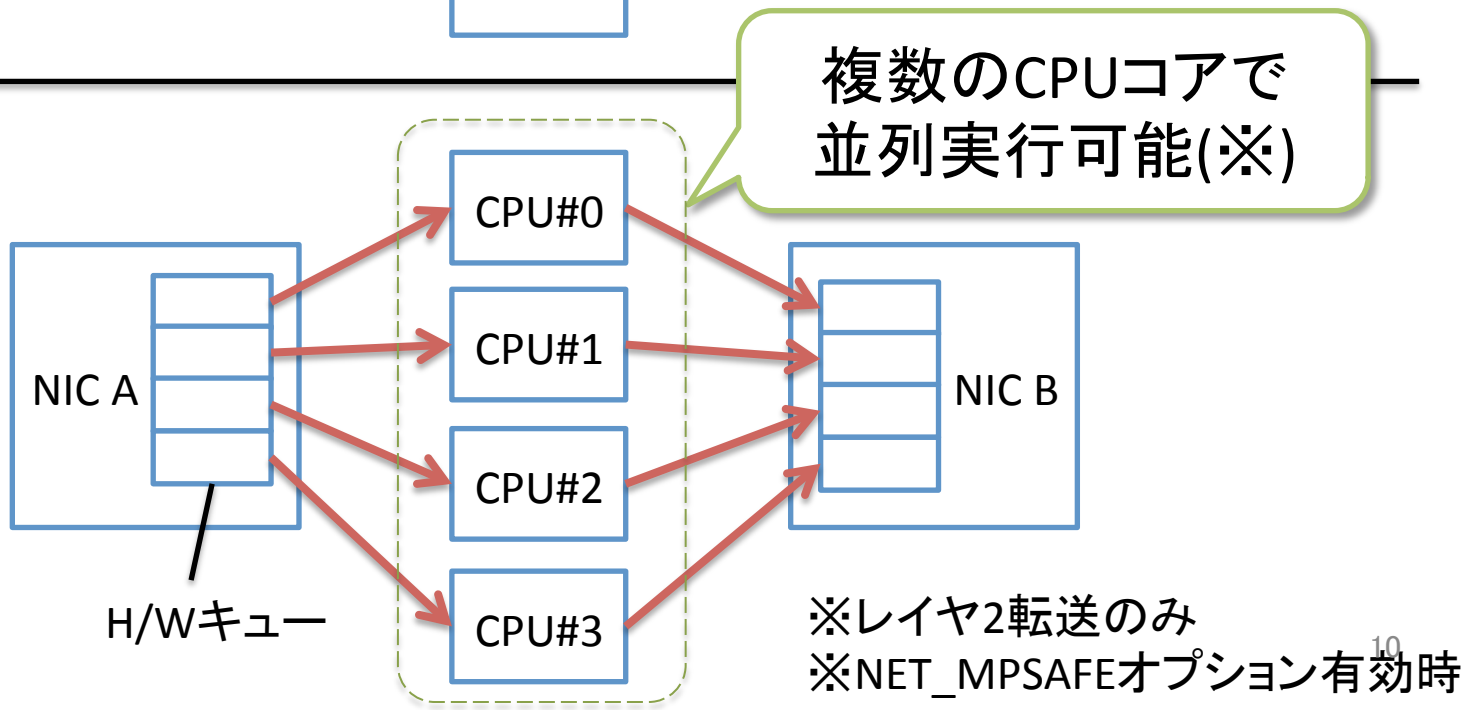
IIJの現在の取り組み

- 現在までの成果

Before



After



※レイヤ2転送のみ
※NET_MPSAFEオプション有効時

発表概要

- IIJとNetBSD
- NetBSDのネットワーク機能のMP化
  - NetBSDのネットワーク機能の概説
  - レイヤ2転送のMP化
  - 割り込み処理のMP化
  - 性能評価結果
- 今後の取り組みとまとめ

NetBSDのネットワーク機能のMP化
何をやらないといけないのか？

- 割り込み処理のMP化
  - すべての割り込みはCPU#0に上がっていた
  - ハードウェアで割り込みを分散する機能が欲しい
    - MSI/MSI-X, RSS (Receive Side Scaling)
- デバイスドライバのMP化
  - 送受信処理
  - ハードウェアマルチキューの活用
- ネットワークスタックのMP化
  - レイヤ2 (bridge, vlan, bpfなど)
  - レイヤ3 (IP転送、ルーティングテーブルなど)
- ネットワーク向けテストを充実
  - 既存の動作を壊すことなく修正するには必須
  - 次項参照

ATFテスト

- ATF: Auto Test Framework
- NetBSD に統合されているテストフレームワーク
- ユーザプロセスとして、NetBSDカーネルを動作させることが可能
  - rump kernelの仕組みを使う
  - 複数カーネルを動かしてネットワークを構築し、ネットワーク機能のテストを実行可能
- 定期的に実行して結果をWebに公開
  - http://releng.netbsd.org/test-results.html

我々が追加したATFテスト

- 45テストケース
- 例
  - レイヤ2転送
  - IPv4/IPv6転送
  - ARP, NDP
  - ICMP/ICMPv6リダイレクト
  - ifconfig(8)コマンドのオプション
  - gif(4): generic tunnel interface
  - ルーティングフラグ

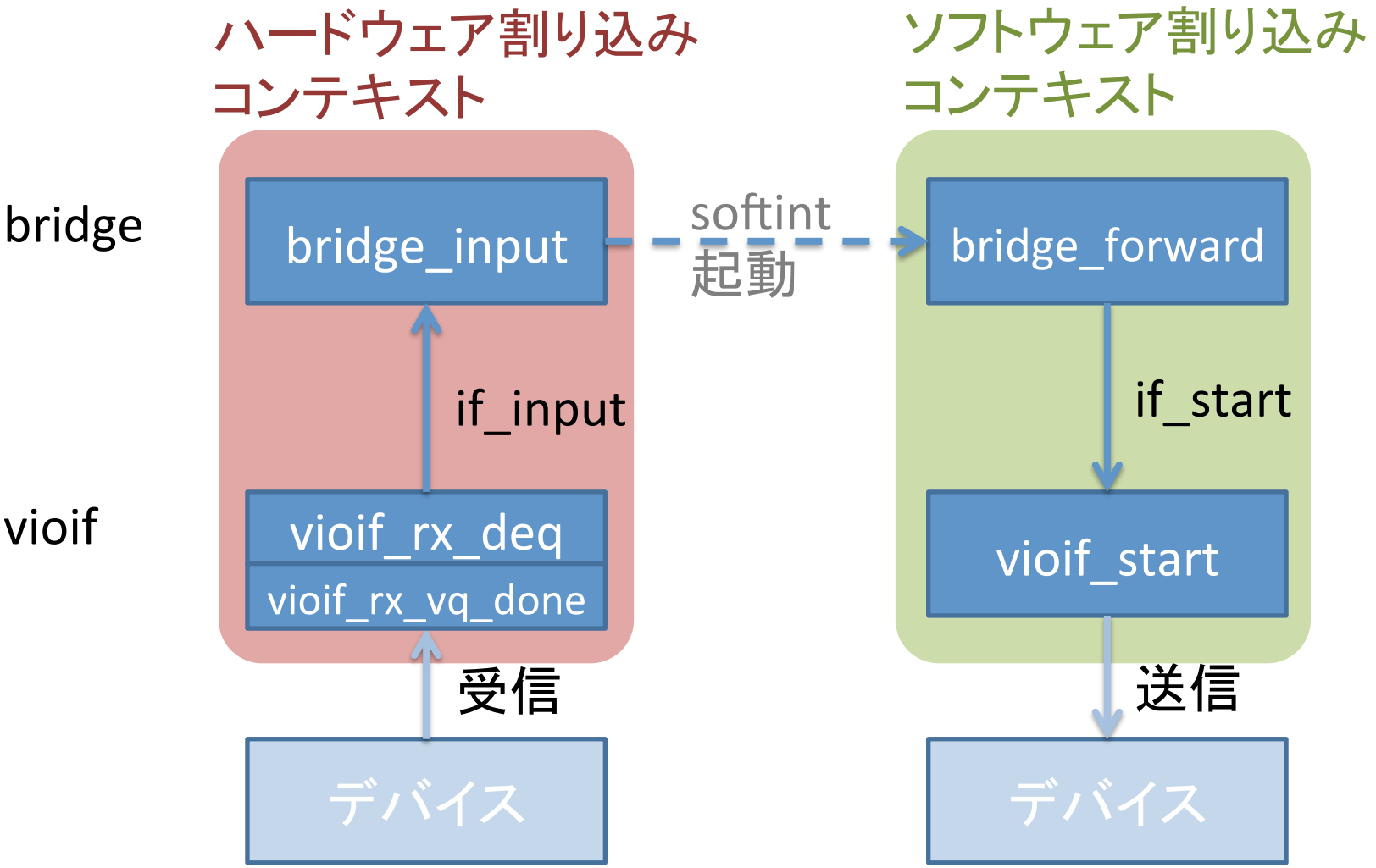
NetBSDのネットワーク機能の概説

- レイヤ2転送の動作
  - 元々の実装の話
- 旧来からの排他機構

レイヤ2転送の動作(1/2)

- bridge(4)
  - ブリッジ擬似デバイス
  - 接続したインタフェース間で、イーサフレーム(等)を転送する

レイヤ2転送の動作(2/2)



ソフトウェア割り込み(softint)

- 概要
  - ハードウェア割り込みでやるには長くかかる・優先度が低い処理を実行するための仕組み
- 特徴
  - スリープ/ブロック可能
    - 可能だが長い期間かかるものは非推奨
  - メモリ確保，開放可能
    - kmem(9) APIはsoftintでの動作を許可していない
    - 現状では確保，開放可能となっているが非推奨
      - kmem_intr_{alloc,free} APIは使える
  - 処理中に実行CPUが移動することはない
    - bridgeのsoftintも常にCPU#0で動く

旧来からの排他機構

- カーネルロック(KERNEL_LOCK)
- IPL, SPL
  - spl(9)

カーネルロック

- いわゆるビッグカーネルロック
  - もしくはジャイアントロック
- CPU間で排他制御
  - あるCPUでカーネルロックを取っている時には、他のCPUではそのロックが取れない
- どこでも利用できる
  - ハードウェア割り込みコンテキストでも使用可能、スリープ可能、他の排他機構と併用可能、再入可能
- 割り込みは禁止しない
- デフォルトではネットワークデバイスドライバの受信処理はこのロックを取ったまま動作する
  - システム内である瞬間には高々一つの割り込みハンドラしか実行されない

IPLとSPL

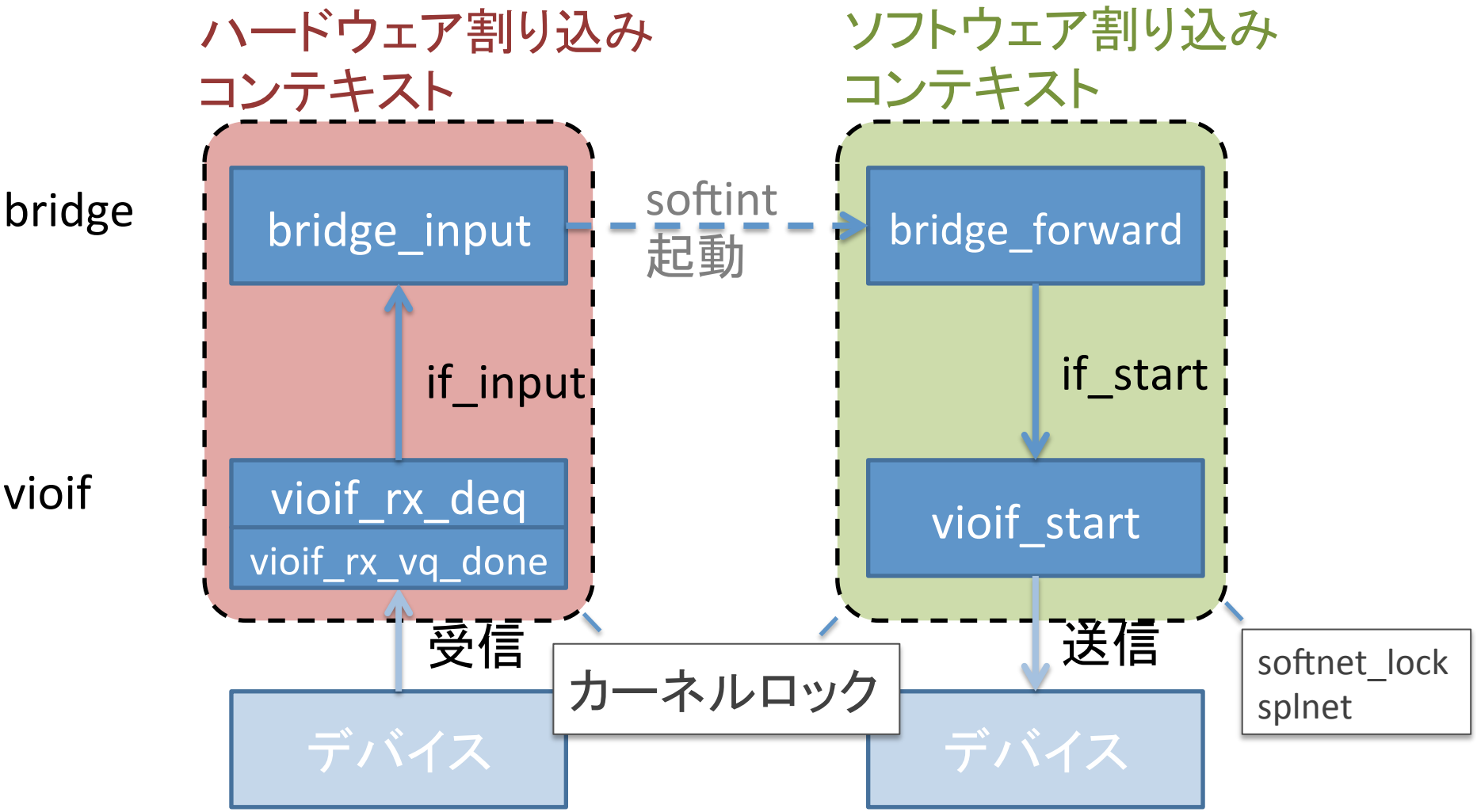
- IPL: Interrupt Priority Level
  - 割り込み(ハンドラ)の優先度
- SPL: System interrupt Priority Level
  - 現在のシステムの割り込み優先度
  - SPLより低いIPLの割り込みハンドラは実行が抑制される
- `spl(9)`
  - SPLを変えることができる
  - 割り込みハンドラと共有するデータを保護したい場合に使用
  - 例: splnetはSPLをIPL_NETまで上げて、IPL_NET以下のレベルの割り込みを禁止する
- 制限
  - 実行中のCPUにしか影響しない

IPL_*	HIGH, SCHED, VM/NET, SOFTSERIAL, SOFTNET, SOFTBIO, SOFTCLOCK, NONE

レイヤ2転送の排他制御(1/2)

- 排他制御
  - bridge_input: カーネルロック
  - bridge_forward: カーネルロック, splnet, softnet_lock (adaptive mutex)

レイヤ2転送の排他制御(2/2)



レイヤ2転送のMP化

- 排他制御機構
  - mutex(9)
  - pserialize(9)
- bridge(4)のMP化

mutex(9) (1/2)

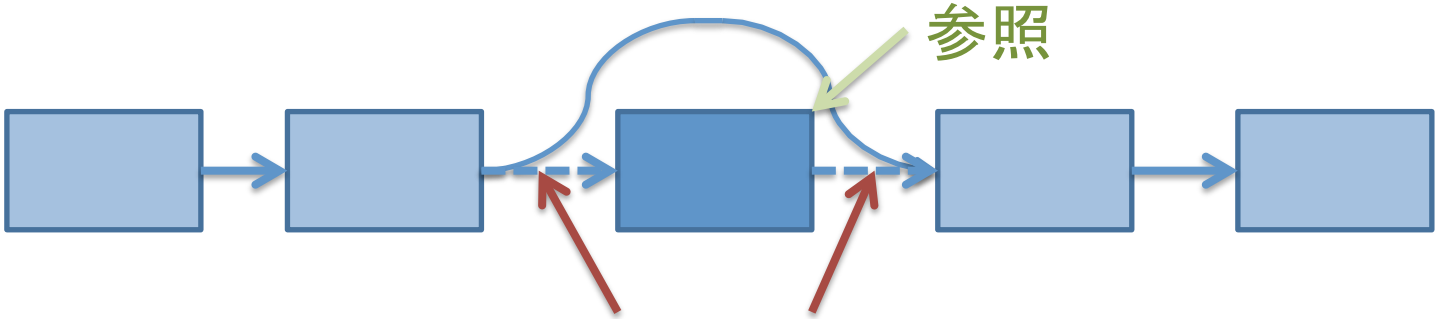
- 共有データの排他アクセス
  - mutex_enterからmutex_exitまでを排他
- 2種類のmutexが存在: spin, adaptive
  - IPLによりどちらを使うかが決まる
    - HIGH, SCHED, VM/NET => spin
    - SOFT* and NONE => adaptive
- Spin mutex
  - mutexが解放されるまでビジーウェイトする
  - ハードウェア割り込みコンテキストで使用可能
  - SPLを必要な IPLまで上げる
    - spl APIを置き換え可能
    - MP化に向けてspl APIはmutexに置き換えられるべき

mutex(9) (2/2)

- Adaptive mutex
  - 最初はビジーウェイトする
    - 別のCPUが対象となるmutexを保持していた場合
  - さらに待たないといけない場合にはスリープする
  - ハードウェア割り込みコンテキストでは使用不可
  - 優先度逆転問題に対応するための機能あり
    - turnstile
- 再入不可能
  - 同じmutexをもう一度取ろうとするpanic

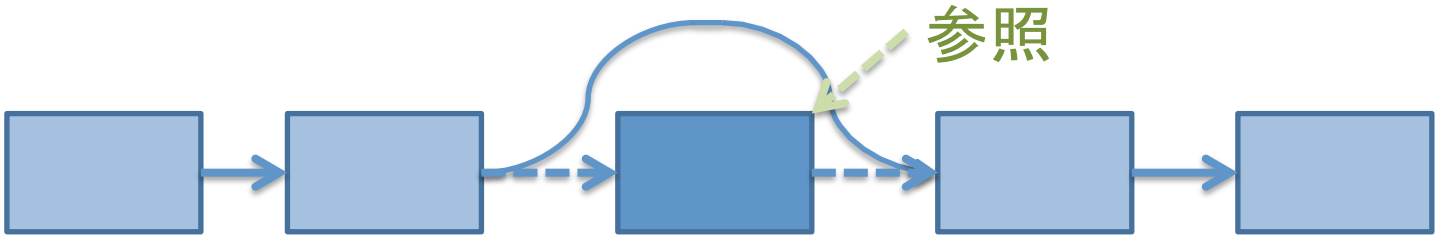
pserialize(9) (1/3)

- pserialize = passive serialization
- LinuxのRCU (read-copy-update)の類似機能
  - ただし、必要なAPIが揃っていない
- 動機
  - ほぼデータ参照しか行わないデータ構造に対して、参照側の高いスケーラビリティを実現する
- アプローチ
  - ロックによる排他処理をせず、参照者がオブジェクトを参照しなくなったことを保証する同期機構を提供する



pserialize(9) (2/3)

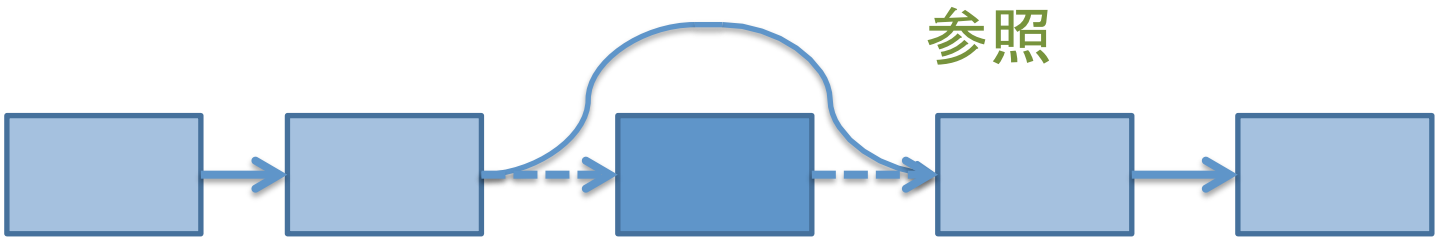
- 参照者からの参照が無くなったことをどう確認するか?
  - 仮定:参照者はクリティカルセクション(CS)内でブロック・スリープしない
    - コンテキストスイッチが起きない
  - CSを実行しているCPU上でコンテキストスイッチが起きたならば、参照者はCSを抜けているため対象オブジェクトへの参照を持たない
  - すべてのCPU上でコンテキストスイッチが起きたならば、すべての参照者が対象オブジェクトへの参照を持たない状態になっている
    - →更新者は安全にオブジェクトを解放できる



更新　.oO(コンテキストスイッチを待つ)

pserialize(9) (3/3)

- pserialize_read_{enter,exit}
  - クリティカルセクション開始、終了に使用
  - プログラマは参照者がCS中でスリープ・ブロックしないことを保証しなければならない
- pserialize_perform
  - 全CPUが2回コンテキストスイッチするまで待つ



更新　.oO(解放できる！)

pserialize(9)使用例

参照

```
s = pserialize_read_enter();
/* リスト等にあるオブジェクトを参照する */
pserialize_read_exit(s);
```

更新

```
mutex_enter(&writer_lock);
/* オブジェクトをリストから削除する */

pserialize_perform(psz);
/* ここまでくると、参照者が誰もオブジェクトを参照していないことが保証される */

mutex_exit(&writer_lock);
/* これ以降、安全にオブジェクトを解放できる */
```

bridge(4)のMP化

- RX割り込みハンドラでカーネルロックを取らないように
  - vioif(4)のMP化
- レイヤ2転送のスケーラビリティ確保のためにpserialize(9)を導入
- 2つのリソースを保護
  - メンバリスト
    - ブリッジに接続されているインタフェース管理用のリスト
  - MACアドレステーブル
    - ブリッジを通過したフレームのMACアドレスのキャッシュを管理するハッシュテーブル(+リスト)

MACアドレステーブル(1/2)

- アクセス方法の特徴
  - キャッシュはパケット転送処理中に追加される
    - 削除はタイマハンドラ等で実施される
  - キャッシュを保持したままではスリープ・ブロックしない
- 参照
  - リストのアクセスにpserializeを使用
- 更新
  - リスト更新にはspin mutexを使用
    - キャッシュはハードウェア割り込みコンテキストで追加される
  - pserialize_perform用にadaptive mutexが必要

MACアドレステーブル(2/2)
(更新者側のコードを抜粋)

```
static void
bridge_rtdelete(struct bridge_softc *sc, struct ifnet *ifp)
{
    struct bridge_rtnode *brt, *nbrt;

    BRIDGE_RT_LOCK(sc);
    BRIDGE_RT_INTR_LOCK(sc);
    LIST_FOREACH_SAFE(brt, &sc->sc_rtlist, brt_list, nbrt) {
        if (brt->brt_ifp == ifp)
            break;
    }
    if (brt == NULL)
        // snip error handling
    bridge_rtnode_remove(sc, brt);
    BRIDGE_RT_INTR_UNLOCK(sc);
    BRIDGE_RT_PSZ_PERFORM(sc);
    BRIDGE_RT_UNLOCK(sc);

    bridge_rtnode_destroy(brt);
}
```

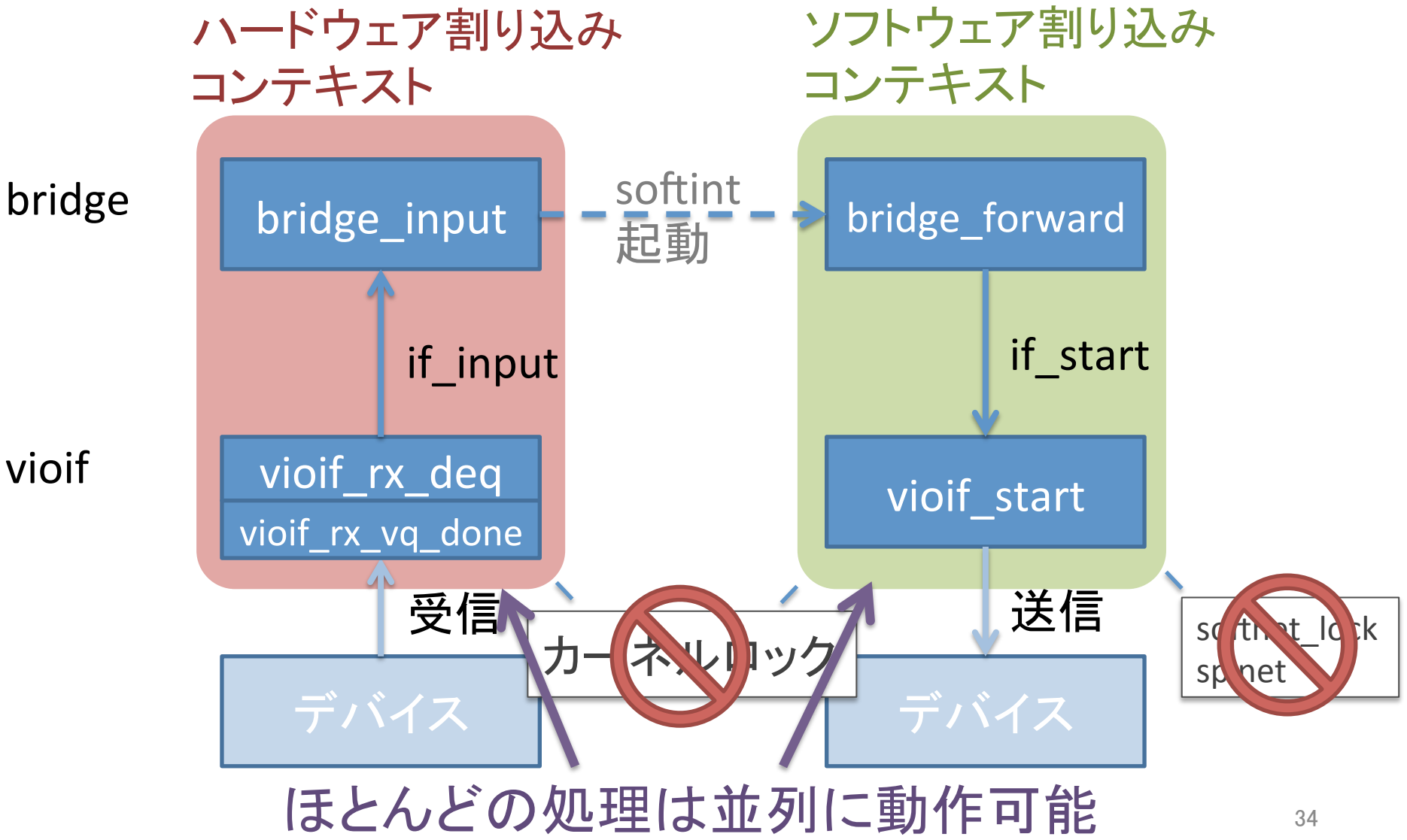
Spin mutex

Adaptive mutex

←リストから
オブジェクトを抜き取る

←オブジェクト解放

レイヤ2転送の動作


ハードウェア割り込み
コンテキスト
ソフトウェア割り込み
コンテキスト
bridge
vioif
bridge_input
softint
起動
bridge_forward
if_input
if_start
vioif_rx_deq
vioif_rx_vq_done
vioif_start
受信
送信
カーネルロック
デバイス
デバイス
ほとんどの処理は並列に動作可能

割り込み処理のMP化

- 割り込み分散(IRQ affinity)
- MSI/MSI-X
- Multi-queue
- Multi-queueとMSI-Xとの組み合わせ
- 実装概要(受信側)

割り込み分散のサポート

- デフォルトでは、全ての割り込みはCPU#0に割り付けられる
- 分散方法
  - デバイスドライバはカーネルAPI (intrrupt_distribute) により、割り込み先CPUを別CPUへの再割当てを行うことができる
  - システム管理者はユーザランドコマンド (intrctl(8)) により、割り込み先CPUを別のCPUへの再割当てを行うことができる

割り込み分散の例 (1)

- intrctl(8)
  - intrctl listサブコマンドにより割り込みカウントを表示

```
# intrctl list | grep -e CPU -e vmx -e wm
 interrupt name CPU#00(+)        CPU#01(+)
 ioapic0 pin 16        10*               0        wm0: legacy
 msix0 vec 0        49115*               0        vmx0: tx 0
 msix0 vec 1            0*               0        vmx0: tx 1
 msix0 vec 2       359017*               0        vmx0: rx 0
 msix0 vec 3          477*               0        vmx0: rx 1
 msix0 vec 4            0*               0        vmx0: link
 msix1 vec 0            0*               0        wm1: tx0
 msix1 vec 1            0*               0        wm1: tx1
 msix1 vec 2            0*               0        wm1: rx0
 msix1 vec 3            0*               0        wm1: rx1
 msix1 vec 4            0*               0        wm1: link
```

割り込み分散の例 (2)

- intrctl(8)
  - intrctl affinity –i “interrupt id” –c “cpuid” サブコマンドにより割り込みを別CPUに割り付け

```
# intrctl affinity -i 'msix0 vec 2' -c 1
# intrctl list | grep -e CPU -e vmx -e wm
 interrupt name CPU#00(+)        CPU#01(+)
 ioapic0 pin 16        43*               0          wm0: legacy
 msix0 vec 0        49154*               0          vmx0: tx 0
 msix0 vec 1            0*               0          vmx0: tx 1
 msix0 vec 2       359081              155*          vmx0: rx 0
 msix0 vec 3          574*               0          vmx0: rx 1
 msix0 vec 4            0*               0          vmx0: link
 msix1 vec 0            0*               0          wm1: tx0
 msix1 vec 1            0*               0          wm1: tx1
 msix1 vec 2            0*               0          wm1: rx0
 msix1 vec 3            0*               0          wm1: rx1
 msix1 vec 4            0*               0          wm1: link
```

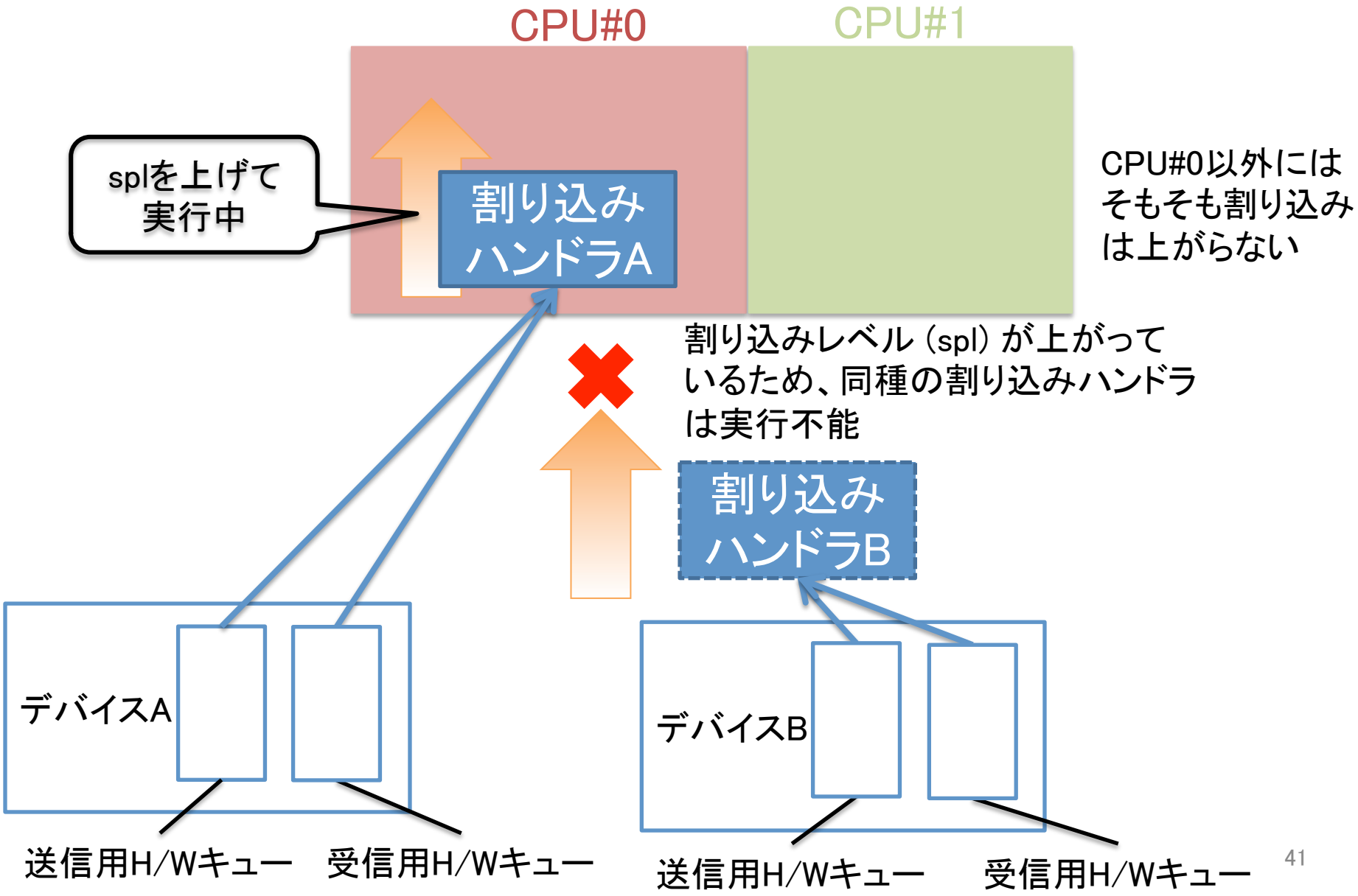
MSI/MSI-Xサポート (1/2)

- MSI: Message Signaled Interruptの略
  - PCI, PCI Expressバスの仕様として定義
    参照: https://pcisig.com/specifications ※ 有料
    - 割り込みをメモリ書き込みとして発生
    - 1つのデバイスが**複数**の割り込みを使用可能
      - 従来の割り込み (INTx) は1つのみ
  - MSI-XはMSIの拡張版
    - 1つのデバイスが使用できる割り込み数の上限増加
    - メモリ書き込み(割り込み)のアドレスとデータとをソフトウェアから**割り込み個別**に設定可能
  - **1つのデバイスから複数のCPUに同時に割り込みをあげることが可能**

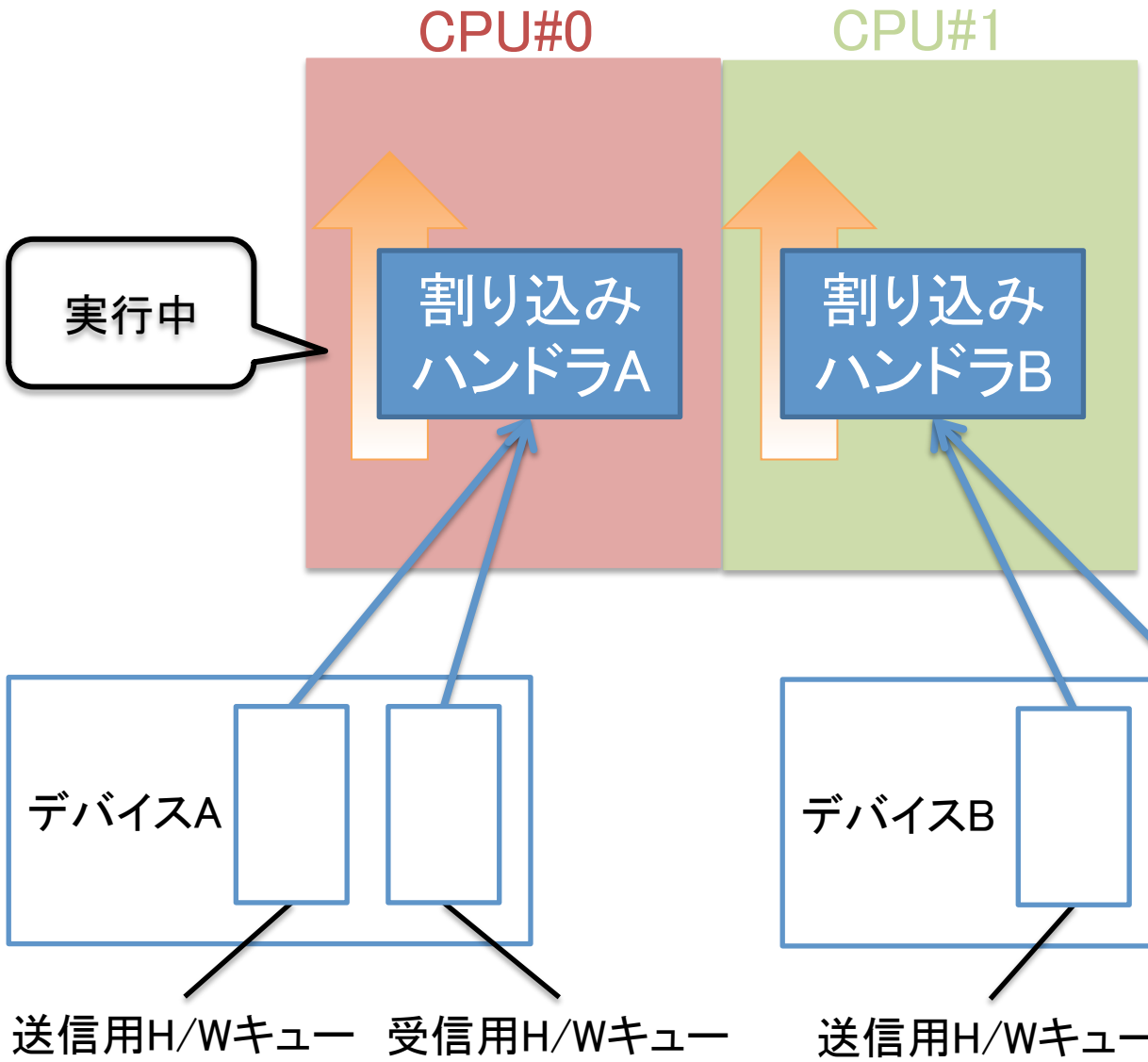
MSI/MSI-Xサポート (2/2)

- 開発開始時点の状況
  - NetBSD/ppcはMSIのみのサポートだった
  - NetBSD自体がMSI-Xを非サポートだった
- 開発内容
  - NetBSD/x86向けのMSI/MSI-Xサポートを実装
    - NetBSD/ppcのMSI-Xサポートも計画中

割り込み分散無しの場合



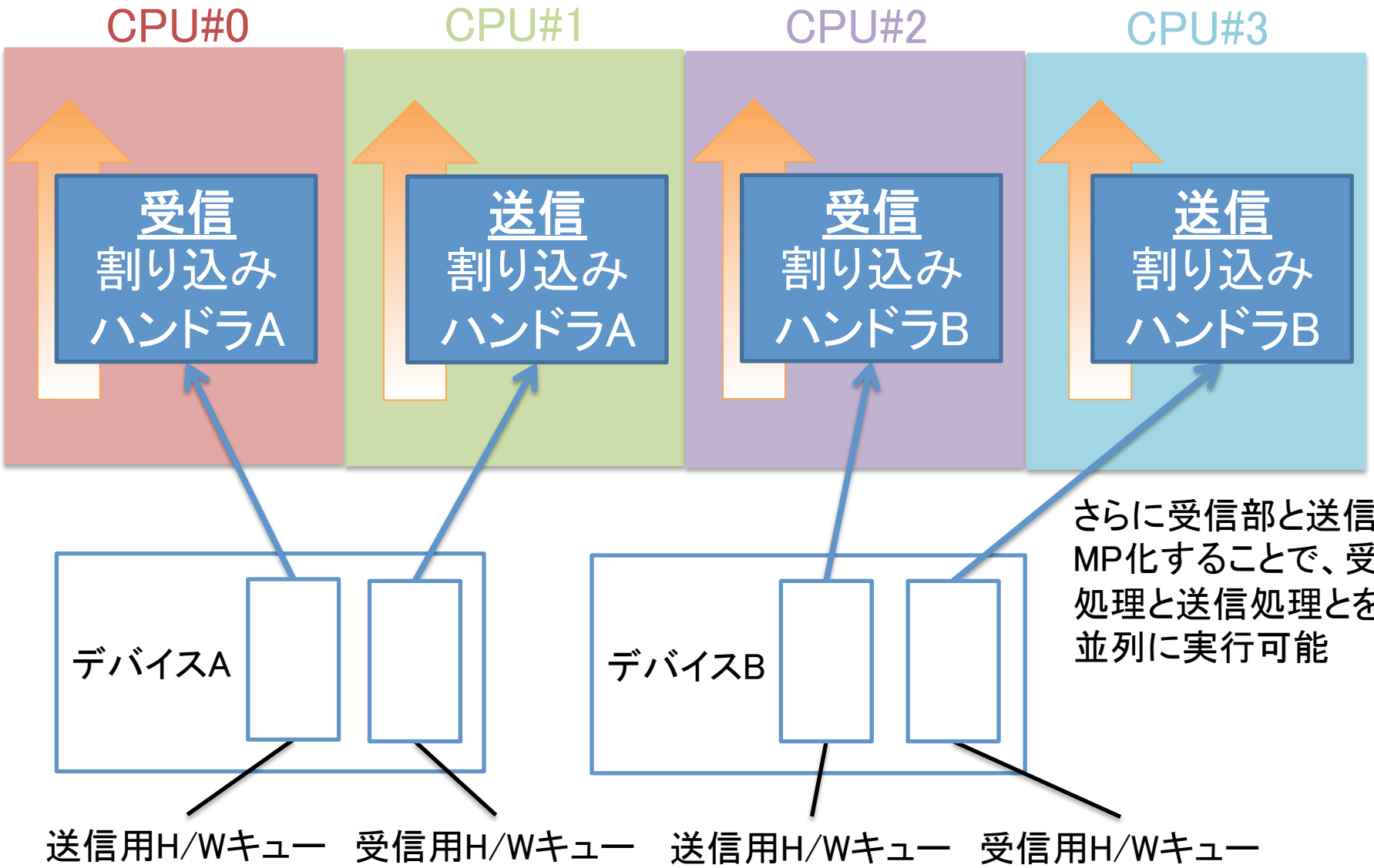
割り込み分散ありの場合



別CPUの割り込みレベルの
影響は受けない
割り込みハンドラ全体を
MP化することで、別CPUで
同種の割り込みハンドラを
並列に実行可能

※ NET_MPSAFE有効時のみ
無効時は KERNEL_LOCK
を取るため実行不能

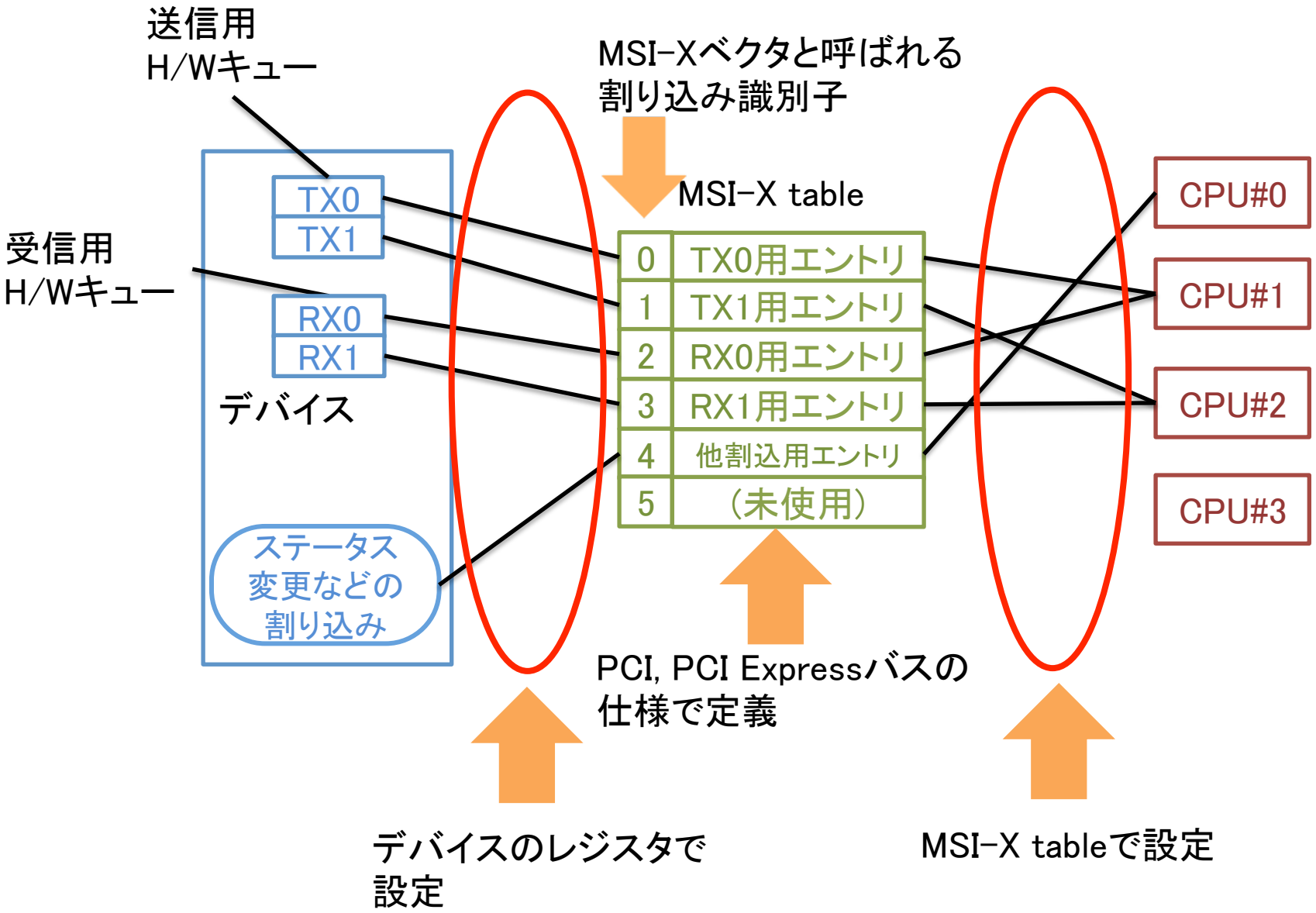
割り込み分散 + MSI-Xの場合



Multi-queueに関して

- 最近のGigabit以上のEthernet controllerは複数の送信受信キューを持っている
  - このことを指して「Multi-queue」と呼ぶ
- (多くの場合)各ハードウェアキューに対して、処理を行うCPUを個別に割り当て可能
  - 別CPUに割り当てられたハードウェアキューからのパケット処理は並列に処理可能

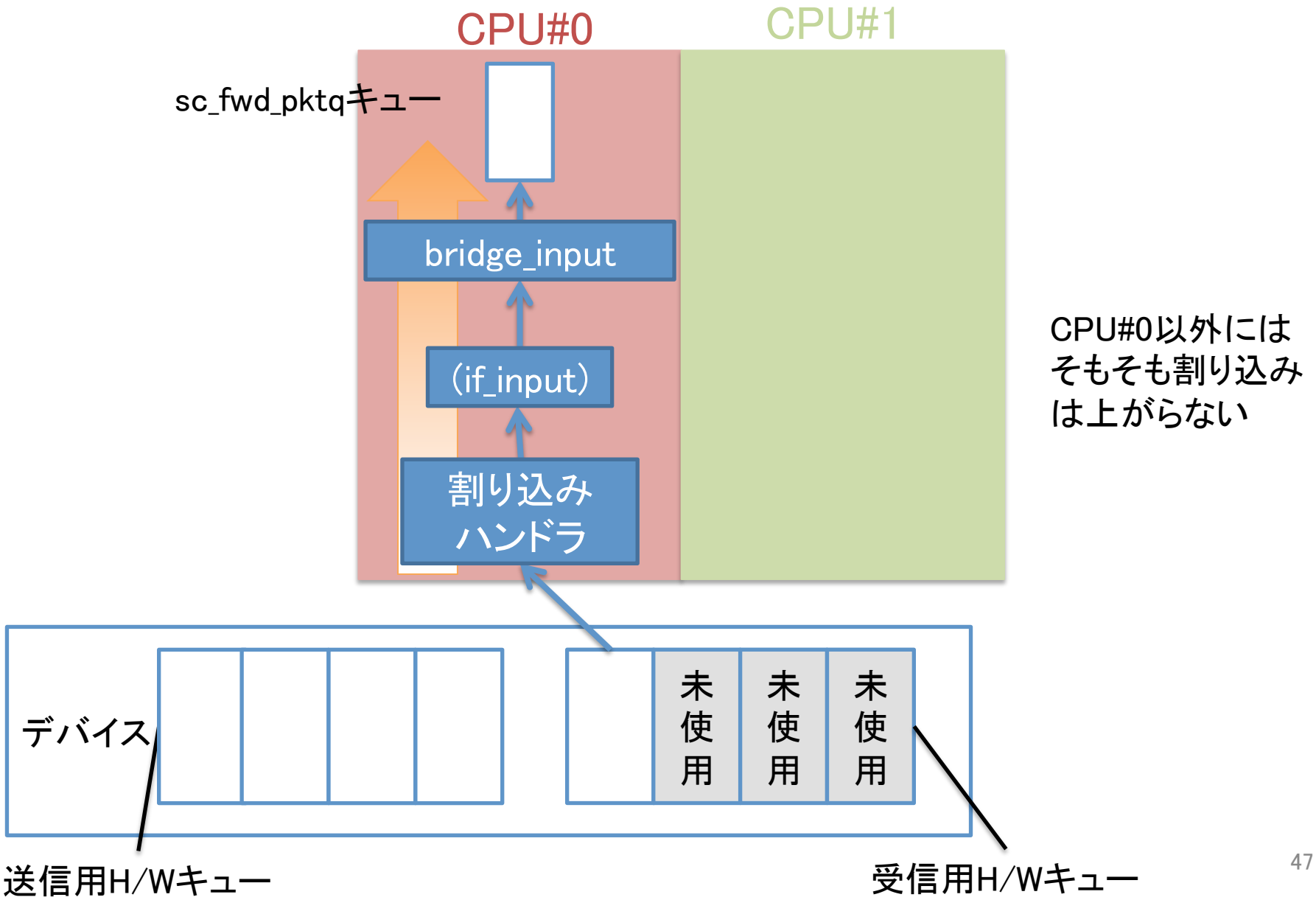
Multi-queueとMSI-Xとの関係例



実装概要 – 受信部

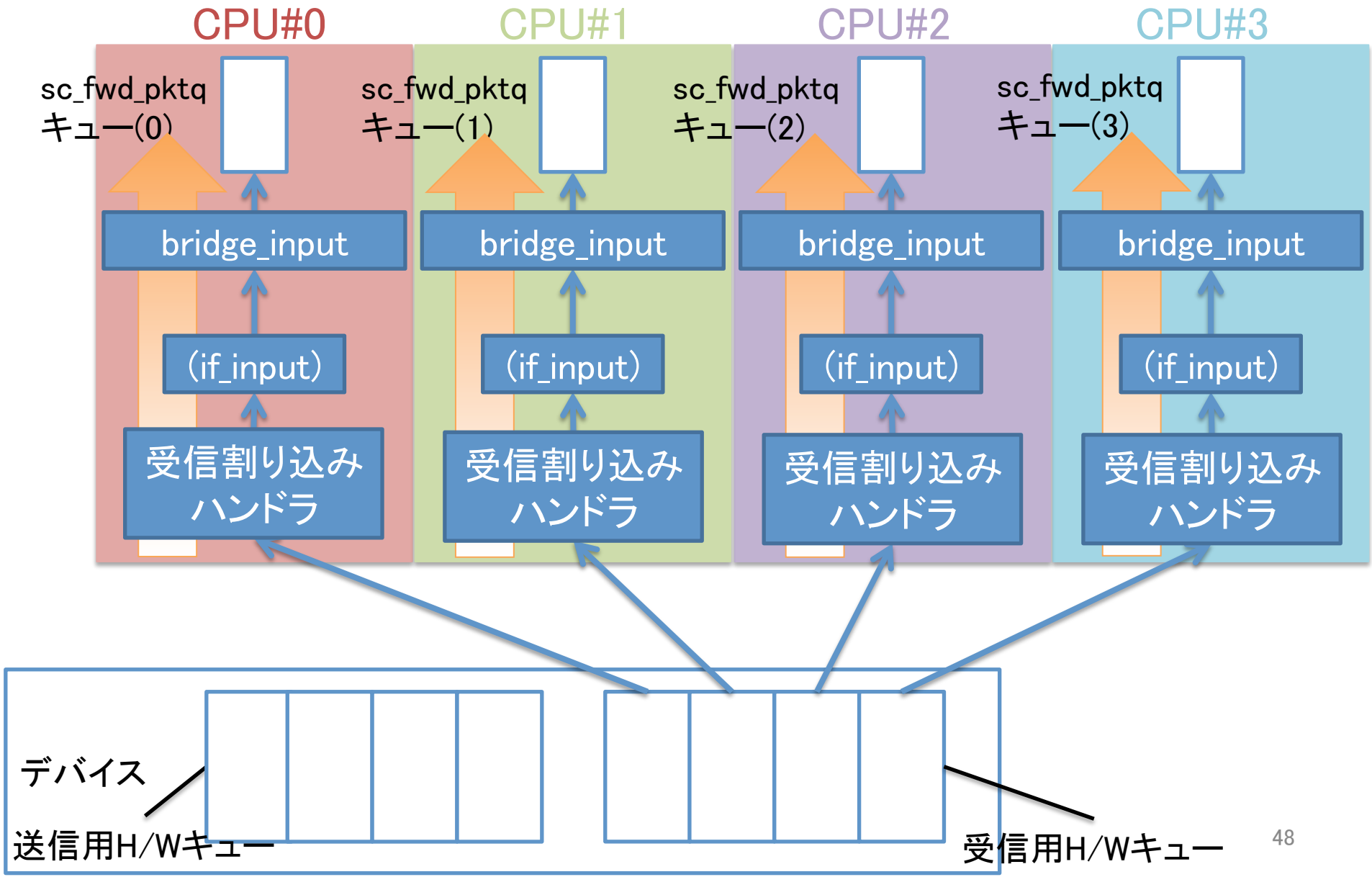
- if_wm
  - 複数のハードウェアキューが使えるように準備
    - ハードウェアキュー管理用構造体を複数作成可能に
    - Multi-queue関連レジスタの設定
  - 複数のハードウェアキューを使ってスケーリング向上
    - 各受信割り込みハンドラを別々のCPUに割り当て
    - 受信処理全体のロックから受信キュー毎のロックに細分化
- if_bridge
  - 各CPU用に個別のキューを使用
    - このキューはbridge_inputとbridge_forwardとの間で使用
  - 他はbridge(4) MP化時に対応済み

実装結果 – 受信部（修正前）



CPU#0以外には
そもそも割り込み
は上がらない

実装結果 – 受信部 (修正後)


CPU#0
CPU#1
CPU#2
CPU#3
sc_fwd_pktq
キュー(0)
sc_fwd_pktq
キュー(1)
sc_fwd_pktq
キュー(2)
sc_fwd_pktq
キュー(3)
bridge_input
bridge_input
bridge_input
bridge_input
(if_input)
(if_input)
(if_input)
(if_input)
受信割り込み
ハンドラ
受信割り込み
ハンドラ
受信割り込み
ハンドラ
受信割り込み
ハンドラ
デバイス
送信用H/Wキュー
受信用H/Wキュー

性能評価結果

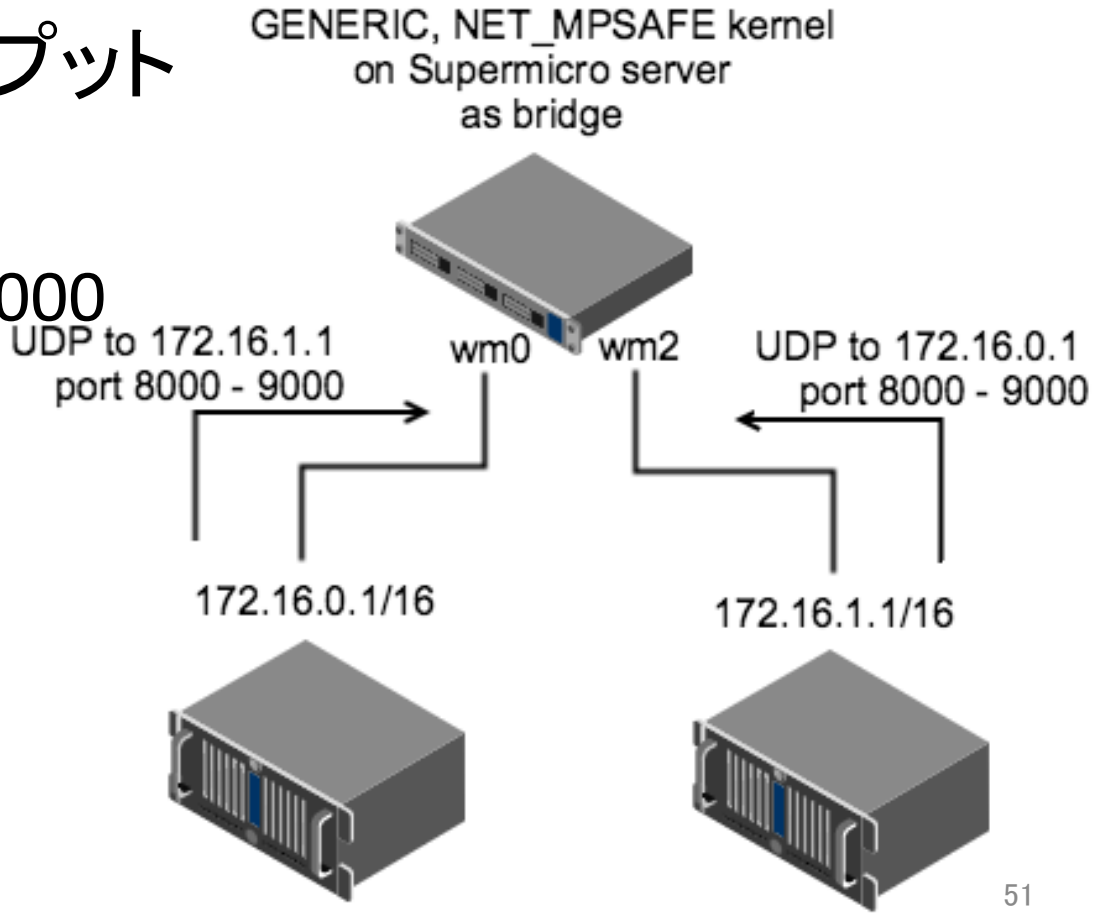
- 測定環境
- 測定結果

測定環境
DUT (Device Under Test: 測定対象)

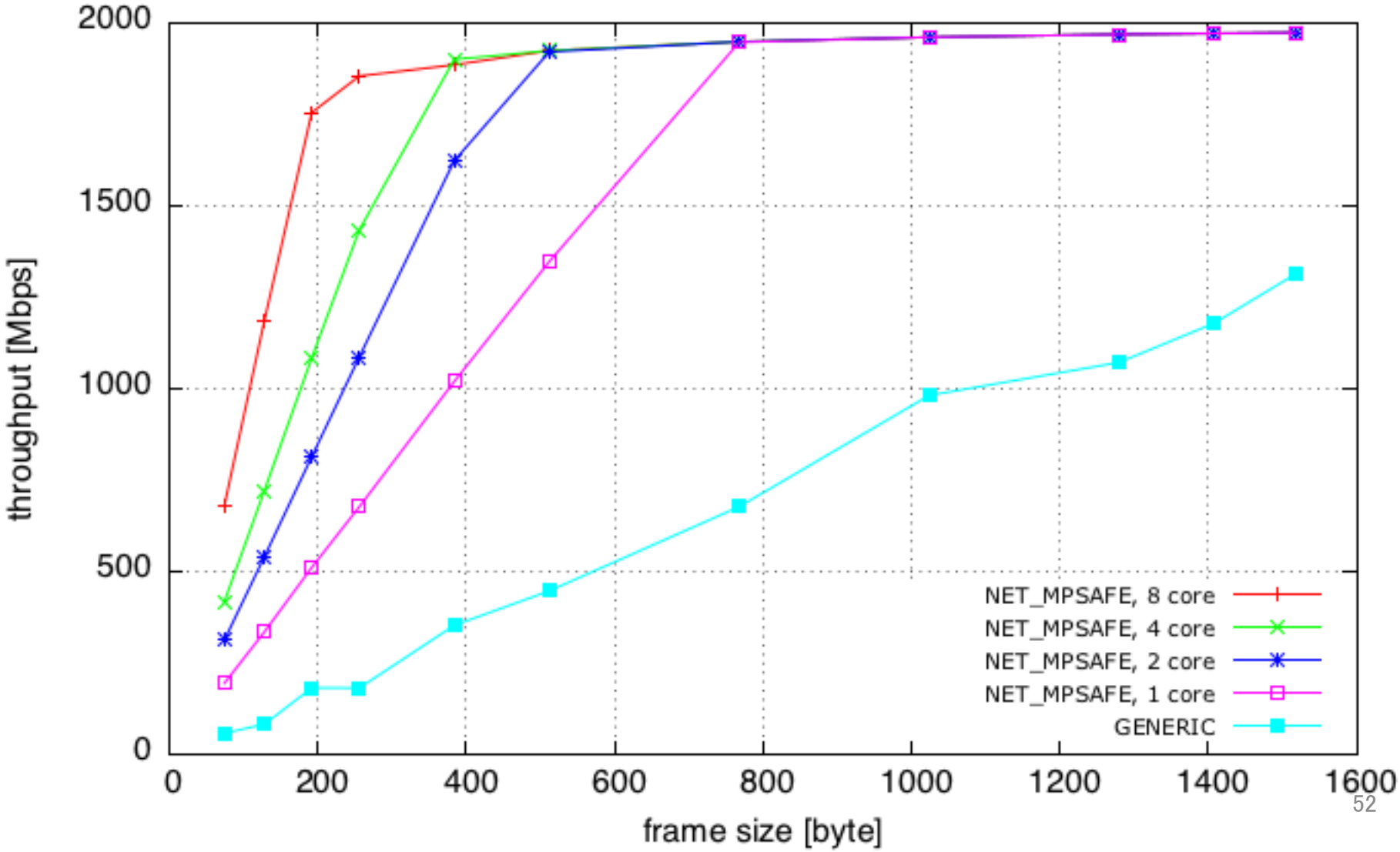
- ハードウェア
  - Supermicro A1SRi-2758F
    - **8 core** Atom C2758 SoC
    - 4 port I354 Ethernet アダプタ(送受信共に**8キュー**)
- カーネル
  - GENERIC (比較用の修正前カーネル)
    - 2015-01-07時点のNetBSD-currentを使用
    - GENERICコンフィグでビルド
  - NET_MPSAFE (修正後のカーネル)
    - GENERICカーネルに MP化実装を追加
    - GENERICコンフィグに加えてNET_MPSAFEを有効にしたコンフィグでビルド

測定環境
全体図

- DUTをブリッジとして使用
  - 2つのポートを使用
- RFC 2544スループット
- UDP
  - ポート8000 ～ 9000
  - 双方向



測定結果
フレームサイズ vs. Mbps



発表概要

- IIJとNetBSD
- NetBSDのネットワーク機能のMP化
  - NetBSDのネットワーク機能の概説
  - レイヤ2転送のMP化
  - 割り込み処理のMP化
  - 性能評価結果
- 今後の取り組みとまとめ

今後の取り組み

- TXマルチキュー
- レイヤ3 (パケット転送)のMP化
  - 我々の本命
  - 次ページ参照
- その他機能のMP化
  - bpf / vlan
  - gif (汎用トンネルインタフェース)
  - ipsec / opencrypto

レイヤ3転送のMP化

- ルーティングテーブルやネクストホップ(ARP/NDP)キャッシュにロックを導入する
  - 現在はカーネルロック、softnet_lock依存
- 現在までの成果
  - ルーティングテーブル探索場所の整理
    - レイヤ2でやっていた
  - ネクストホップキャッシュのデータ構造の見直し
    - FreeBSDからデータ構造を移植(lltable/llentry)
    - ARPに適用
- 今後の予定
  - lltable/llentryをNDPに適用
  - ルーティングテーブルからネクストホップキャッシュを取り除く
    - ロックを取りやすくするため
  - ルーティングテーブルにロック導入

今後やるかもしれないこと

- 受信処理のソフトウェア割り込み化
  - レイヤ2のコードがハードウェア割り込みコンテキストで実行される
    - 制約が多いので辛い
    - すべてソフトウェア割り込みで動いてくれると楽
- ルーティングテーブル置き換え
  - 現実装はかなり古く並列化しにくい
  - 何で置き換えるか？
    - rttree(9) by Dennis Ferguson, ART (OpenBSD), etc.
  - Poptrie (SIGCOMM 2015)の導入？

まとめ

- IIJはNetBSDにたくさん貢献しています
- NetBSDのネットワーク機能の一部が並列に動作するようになりました
  - bridge(4)
  - MSI/MSI-X
  - ハードウェアマルチキュー
- レイヤ3パケット転送処理の並列化にはもう少しかかりそうです