メモリホットプラグの現状と今後

OSDN セミナー

Linux Kernel Conference

2003年10月15日

VA Linux Systems Japan K.K.

高橋浩和

taka@valinux.co.jp

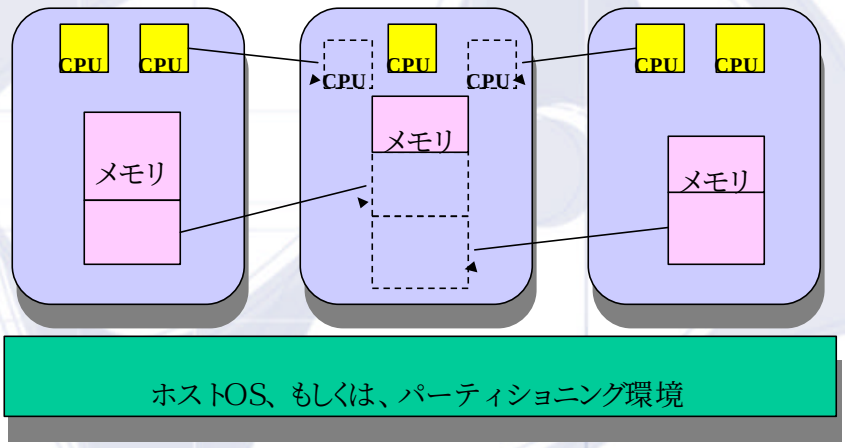
目次

メモリホットプラグの目的と効果

- メモリモジュールの活性保守交換
- 仮想マシン環境、パーティショニング環境における ゲストOS間でのメモリの移動
- NUMA環境におけるプロセスマイグレーション
- 組み込み機器における省電力処理との連携
- 物理メモリのデフラグメント処理

VA LINUX SYSTEMS JAPAN

OS間でのメモリとCPUの融通



VA LINUX SYSTEMS JAPAN

メモリホットプラグ実装の現状（1）

- **メモリマイグレーション機能の基本実装ができた**
  - ページキャッシュとプロセスメモリが対象
- **ノードホットプラグとメモリセクションホットプラグ**
  - IBMと富士通のチームが、自らのハードウェアに合わせて、仕様策定中
  - IBM： LPAR（論理区画）環境用の、小さな単位でのメモリホットプラグ
  - 富士通： システムパーティショニング環境用の、ノード単位でのホットプラグ

メモリホットプラグ実装の現状（2）

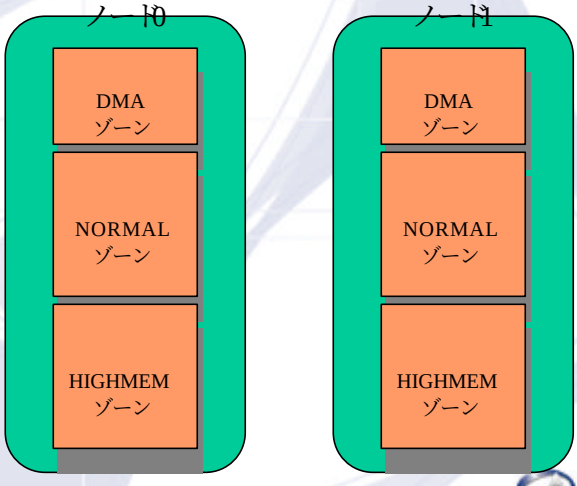
- **sysfsインターフェイスとホットプラグイベント**
  - 通常のホットプラグデバイスとして表現
- **hugetlbpageのホットプラグ**
  - 試作完了
- **物理ページのデフラグメント処理**
  - Marcelo Tosattiが汎用機能を設計開始

Copyright© 2004, VA Linux Systems Japan K.K. All rights reserved.

Linuxメモリ管理の概要

VA LINUX SYSTEMS JAPAN

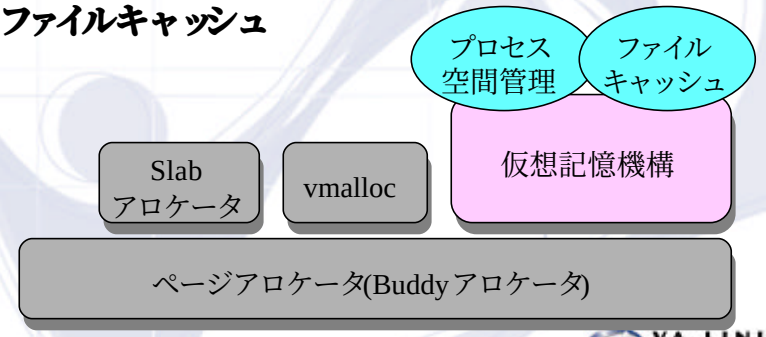
ノード、ゾーン、ページ



VA LINUX SYSTEMS JAPAN

メモリの種類

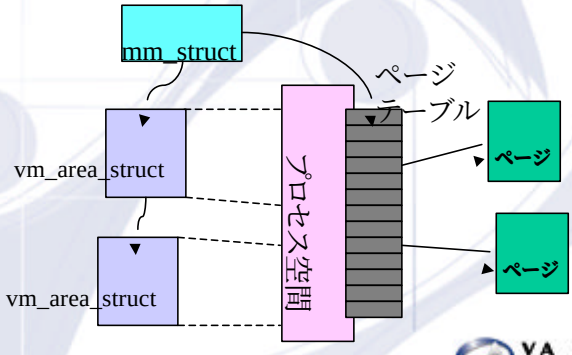
- **カーネル内部データ**
- **プロセスメモリ**
- **ファイルキャッシュ**



VA LINUX SYSTEMS JAPAN

プロセス空間(1)

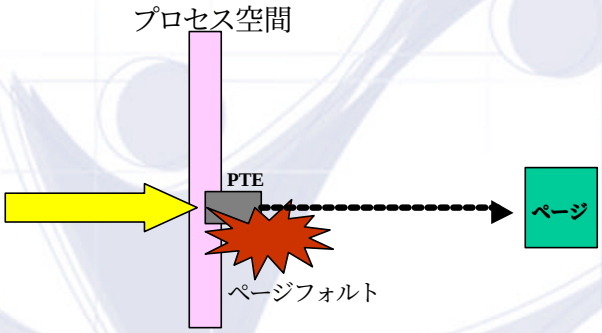
- **一つの空間全体を管理するmm_struct**
- **有効区間と属性を管理するvm_area_struct**



VA LINUX SYSTEMS JAPAN

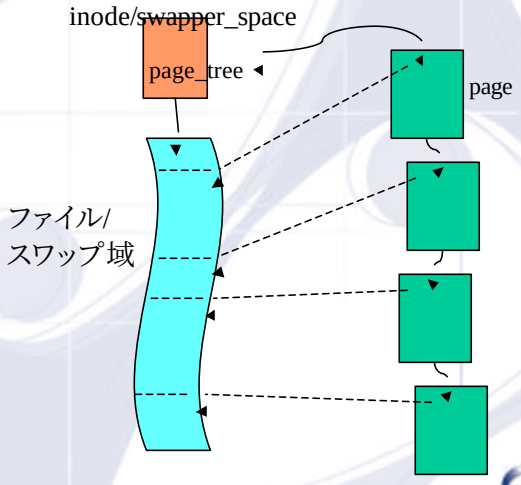
プロセス空間(2)

- **デマンドページングとコピーオンライト**
  - 実際に必要になるときまで、ページ割り当てを遅延



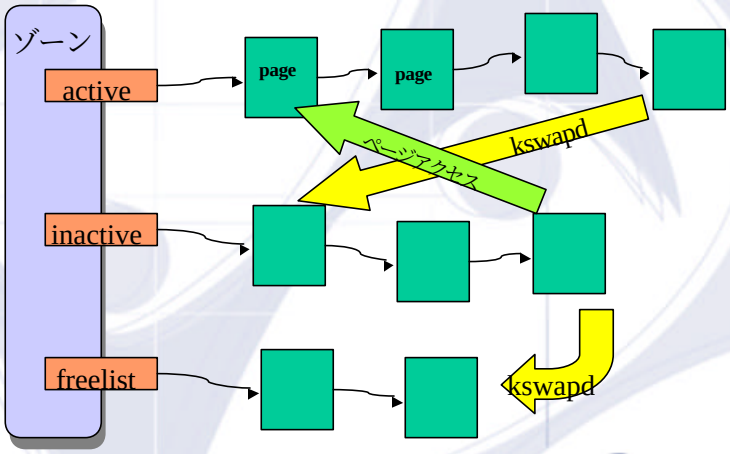
VA LINUX SYSTEMS JAPAN

ページキャッシュとスワップキャッシュ



VA LINUX SYSTEMS JAPAN

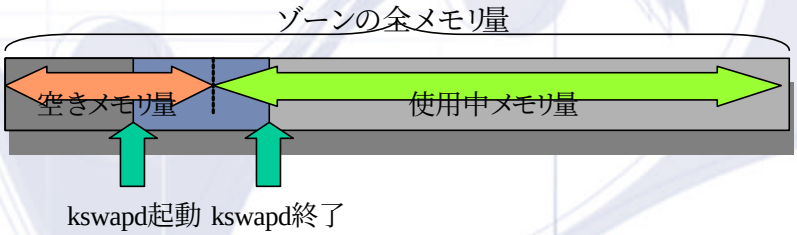
スワップアウトによるページの回収(1)



VA LINUX SYSTEMS JAPAN

スワップアウトによるページの回収(2)

- **ゾーン毎のウォーターマークによりkswapd動作**
- **もしくはメモリ確保処理の延長でページを解放**



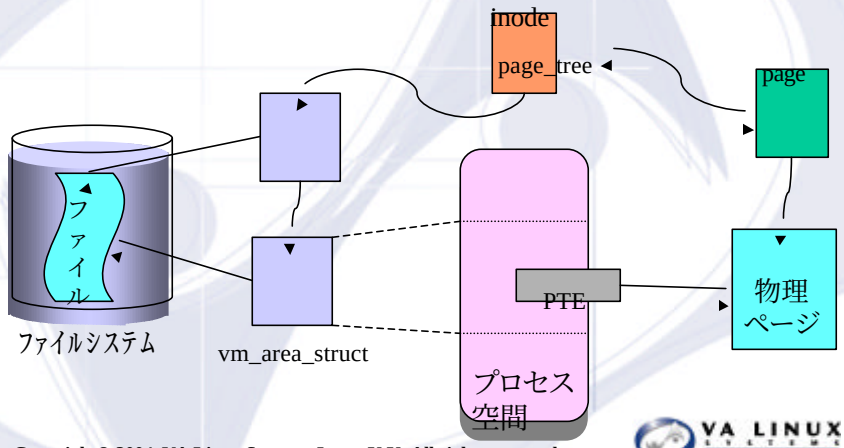
VA LINUX SYSTEMS JAPAN

スワップアウト処理

- **プロセス空間からのマップを解除**
- **ファイルマップ域**
  - ページがDirtyであれば、ファイルへ書き戻す
  - ファイルシステム固有のページ依存データを解放
  - ページを破棄
- **プロセスメモリ**
  - スワップキャッシュ上に移動。swap entryを割り当て
  - swapデバイスにデータを退避
  - PTEにswap entryを書き込む

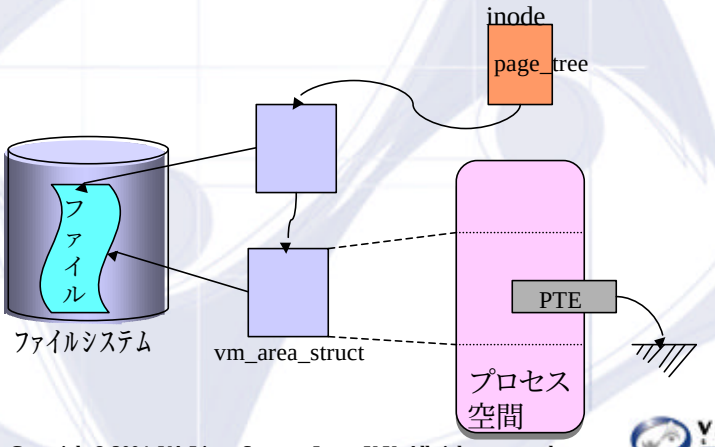
VA LINUX SYSTEMS JAPAN

objrmap – ファイルマッピング域 -

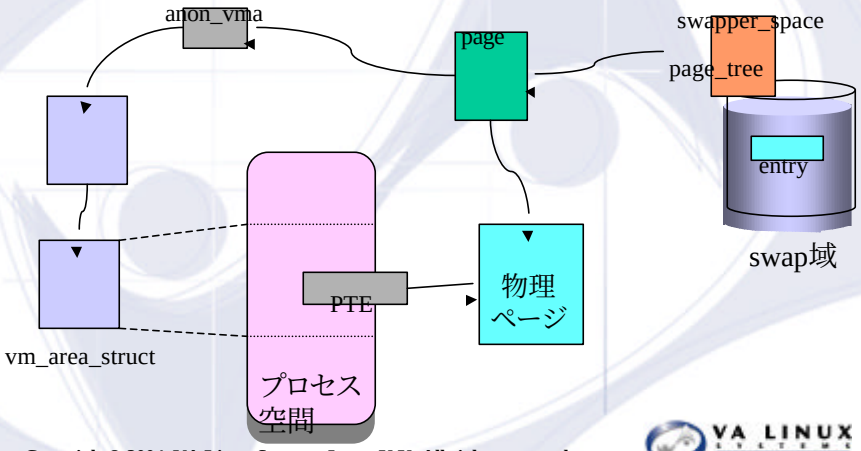


VA LINUX SYSTEMS JAPAN

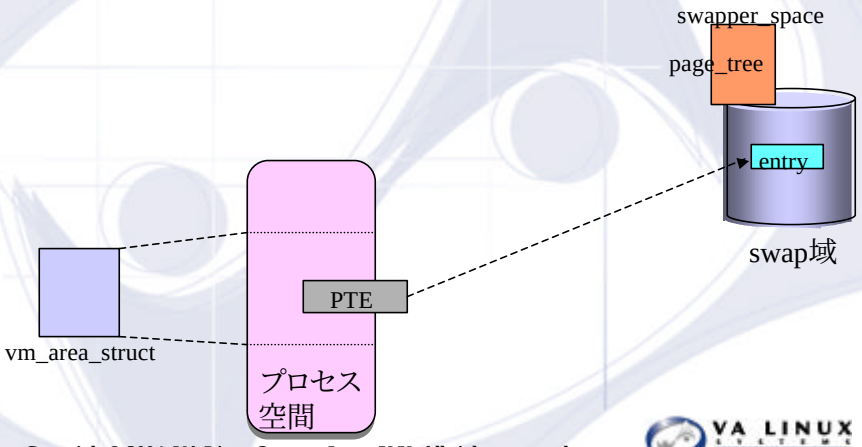
ページの解放 - ファイルマッピング域 -



objrmap – 無名マッピング域 -



ページの解放 – 無名マッピング域 -



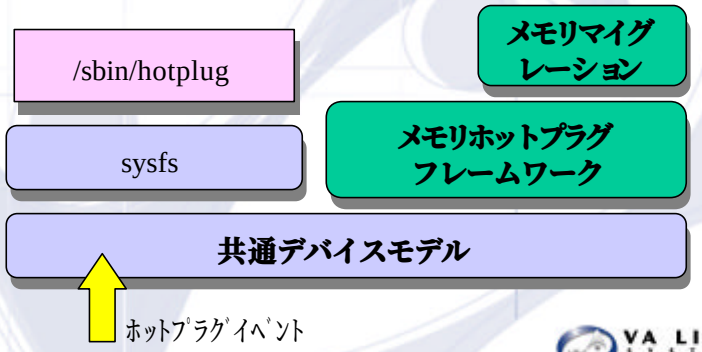
VA LINUX SYSTEMS JAPAN

メモリホットプラグの実装

VA LINUX SYSTEMS JAPAN

構造

- **ホットプラグデバイスの一種として実装**
  - CPUも同様の実装

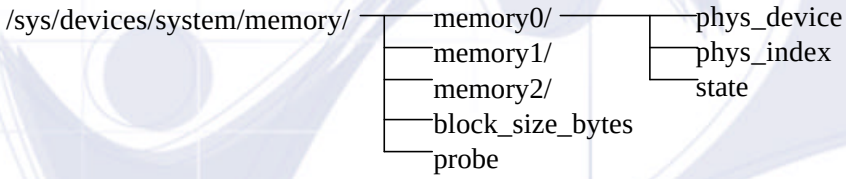


メモリの追加

- **ホットプラグイベントの発生**
  - sysfsへ生成
  - hotplugスクリプト起動。メモリ属性の決定と管理対象への組み込み支持
- **メモリ削除を考慮した管理データ配置に注意**
  - ノード管理構造(pgdat)、ページ管理構造(mem_map)をどこに確保すべきか？

sysfs

- **小さな単位でのホットプラグへの対応（案）**
  - 各メモリ単位の状態表示と制御



VA LINUX SYSTEMS JAPAN

メモリの削除

- **対象領域からのメモリ確保を禁止**
- **対象領域からのページの強制移動**
  - メモリ上に展開されているデータの退避
- **メモリ管理データ構造の解放**
  - ページ管理構造(mem_map)の解放
  - ノード毎の削除ならノード管理構造の解放。CPU、各種デバイスのデータ構造解放順序制御
- **sysfsからの対応エントリの削除**

VA LINUX SYSTEMS JAPAN

メモリのマイグレーション処理

- **swapout方式と、remap方式の併用**
- **swapout方式 利用頻度の低いページは、そのまま解放してしまう。**
  - この方式だけでは、対応できないページある
- **remap方式 利用頻度の高いページは、ページを交換する。**
  - 代替ページを確保、データをコピー、古いページの代わりに登録する
- **システムフリーズが不要**

swapout方式

- **ゾーン毎に受け持ちのkswapdがいる**
- **閾値の変更**
- **PTEのアクセスビット、PG_refernced フラグの無視**
- **扱えないページ**
  - mlockされたページ
  - 二次記憶を割り当てられないページ
  - アクセス頻度の高いページ。新しいアクセス要求をブロックできないため
- **I/Oが伴うため遅い、またシステム負荷を上げてしまう**

swapout方式コードイメージ

- **対象の領域でLRU上にあるpageを選択しリストに登録。リスト毎shrink_list()関数に渡す。**

```
for (削除領域内のpageに対し) {
        if (PageLRU(page) && steal_page_from_lru(zone, page))
                list_add(&page->lru, page_list);
}

shrink_list(&page_list, &sc);
```

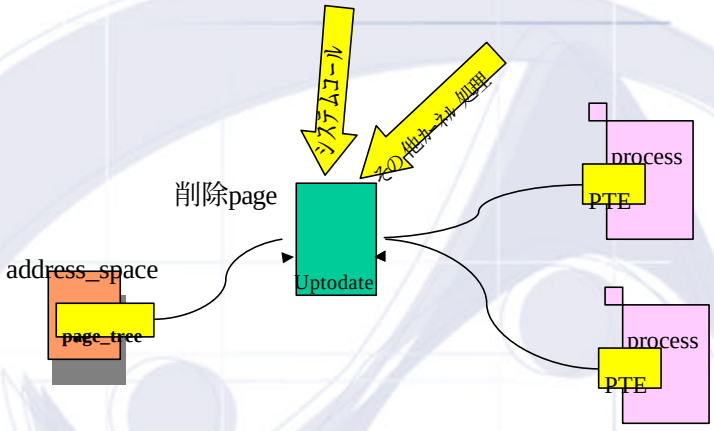

remap方式

- **ページ交換処理中に発生するアクセス要求のブロック**
  - 新しいアクセス要求を交換予定のページ(PG_lock, !PG_uptodate)へ転送
    - page_tree上のページを交換
    - ページをプロセス空間から剥がす
  - 古いページへのアクセスが無くなるのを待って、ページデータをコピー
  - ブロックされているプロセスを起床
- **理論的には、I/O発生なしに実行可能**

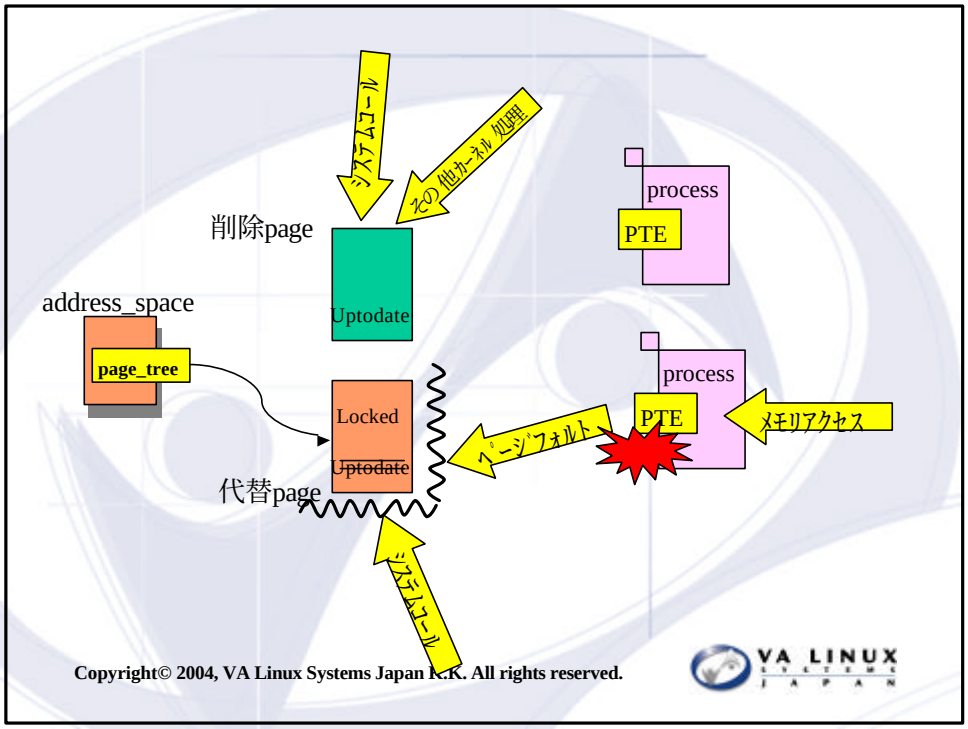
remap方式

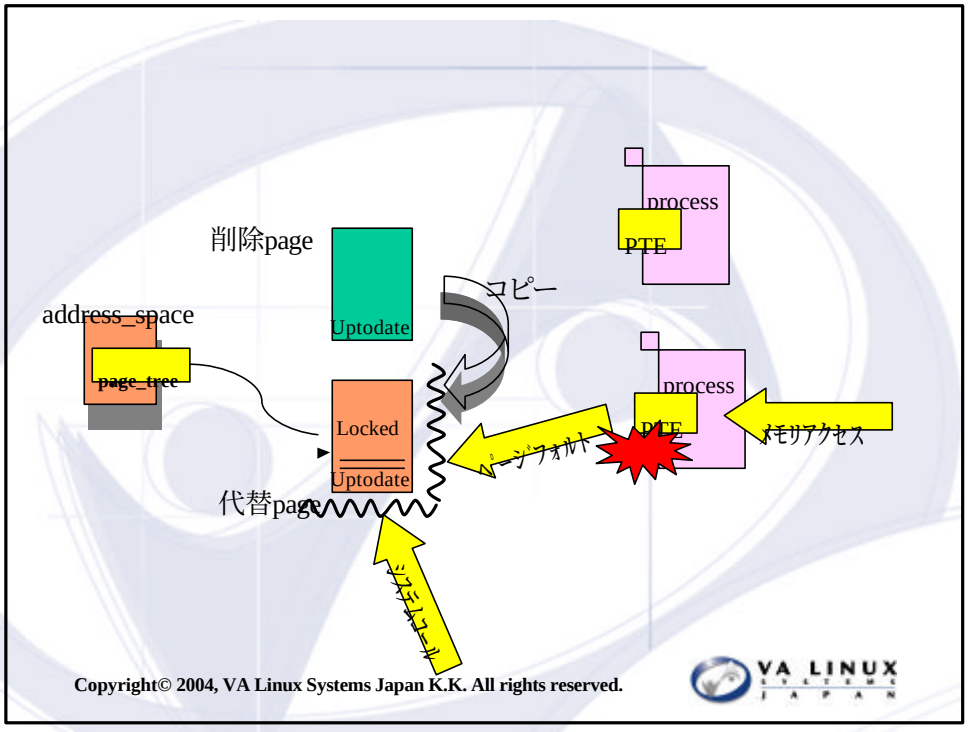
- **待ち合わせが必要なページ**
  - 参照があるページ
    - システムコール処理中
    - プロセス空間にマップされている
  - I/O中のページ
    - lock状態
    - writeback状態
  - dirtyなページ（PG_privateが立っているとき）

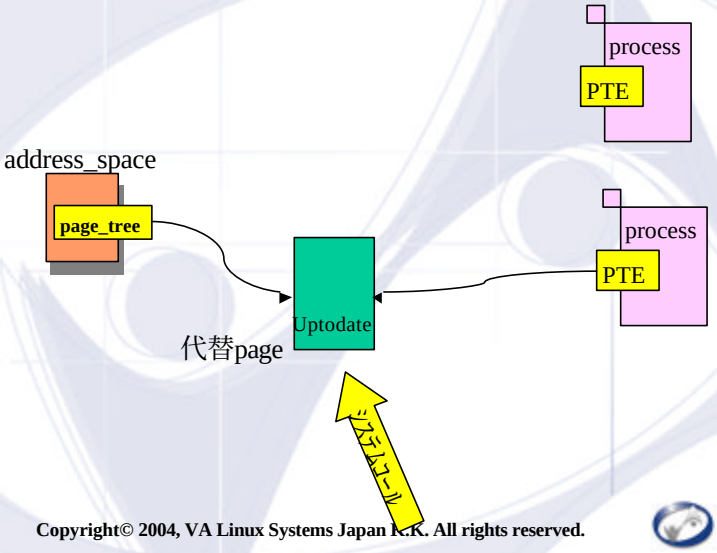
VA LINUX SYSTEMS JAPAN



VA LINUX SYSTEMS JAPAN







VA LINUX SYSTEMS JAPAN

二次記憶が割り当てられてないページ

- **プロセスメモリ**
  - swapエントリを割り当ててから、remap処理を開始する
- **RAMディスク、擬似ファイルシステムが利用するページ**
  - swapout方式は不可能、remap方式のみで対処する

VA LINUX SYSTEMS JAPAN

解放を抑制されているページの扱い

- **mlockされたページ**
  - 強制交換の仕組みの実装
- **その他解放を抑制されているページ**
  - 操作の完了を待ち合わせる
    - I/O処理中のページ。ページキャッシュ上のページ、Direct I/O中のプロセス空間に割り当てたページ
    - futex　mutex変数のアクセス処理
  - ホットプラグ可能なメモリ域に割り当てない
    - AIOのイベント通知用バッファ域

VA LINUX SYSTEMS JAPAN

自由度の高いマイグレーション処理(1)

- **ファイルシステム固有のマイグレーション処理を定義可能とする。**
  - address_space_operationsとして、migrate_pageメソッドを追加
  - ファイルシステム固有の管理下にあるページの移動
    - page->privateメンバを利用しているページ
    - ファイルシステムの判断でページ参照数を上げて、解放を抑制しているページ

VA LINUX SYSTEMS JAPAN

自由度の高いマイグレーション処理(2)

- **ファイルシステムがmigrate_pageメソッドを未定義**
  - PG_privateなページはwritebackした後、ファイルシステム固有データ(buffer以外もある)を解放し、移動する
- **ファイルシステム固有データの移動処理を定義**
  - PG_privateなページは、ファイルシステムが用意した固有データのマイグレーション処理に依頼する
- **マイグレーション処理全体を定義**
  - ファイルシステムが管理するページ全てのマイグレーション処理の責任を持つ

VA LINUX SYSTEMS JAPAN

remap処理コードイメージ

- **swap entryの割り当て**
- **ファイルシステム固有メソッドがあればそれを呼び出し、なければデフォルト処理を呼び出す**

```
lock_page(page);
if (PageAnon(page) && !PageSwapCache(page))
        add_to_swap(page, GFP_KERNEL);
newpage = page_cache_alloc(mapping);
if (mapping->a_ops->migrate_page)
        mapping->a_ops->migrate_page(page, newpage);
else
        generic_migrate_page(page, newpage, migrate_page_common);
page_cache_release(page);
```


VA LINUX SYSTEMS JAPAN

remap処理コードイメージ

- **page_tree上でのページの交換**
- **プロセス空間からのunmap**
- **待ち合わせと、ファイルシステム固有データの移動**

```
replace_pages(page, newpage);
if (page_mapped(page))
        try_to_unmap(page, &vlist);
migrate_fn(page, newpage, &vlist);
unlock_page(newpage);
```


マイグレーション処理の性能改善

- **I/O発生を抑制する**
  - buffer付きのページでも、writebackなしにページ交換。
    ファイルシステム固有の交換処理にて対応
- **他のカーネル処理で利用中のページは後回しにする**

注意点 :マイグレーション処理の要求

- **同じページを二度page_treeから求めてはならない**

```
page = find_get_page(mapping->page_tree, index);
                    :
```

○ page_cache_get(page);

× page = find_get_page(mapping->page_tree, index);

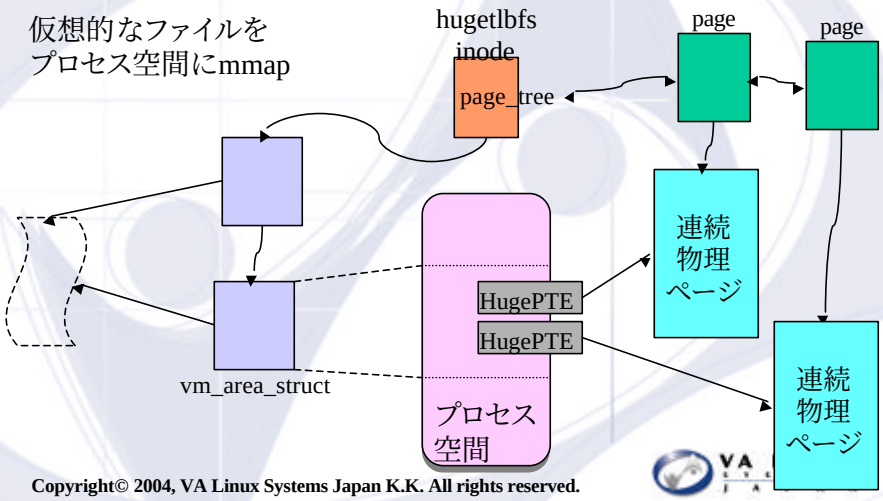
VA LINUX SYSTEMS JAPAN

HugeTlbPageのホットプラグ（1）

- **汎用メモリマイグレーション処理の利用**
  - hugetlbpageは、hugetlbfs上擬似ファイルのページキャッシュ(のようなもの)として実現
- **実装上の課題**
  - 汎用メモリマイグレーション処理は、ページングに依存
    - ページフォルト処理がない（Intel技術者が実装中)、ページの動的確保処理がない
    - ページアウトの対象にならない。LRU上にない、rmap構造を持たない(構造が少し足らない)

VA LINUX SYSTEMS JAPAN

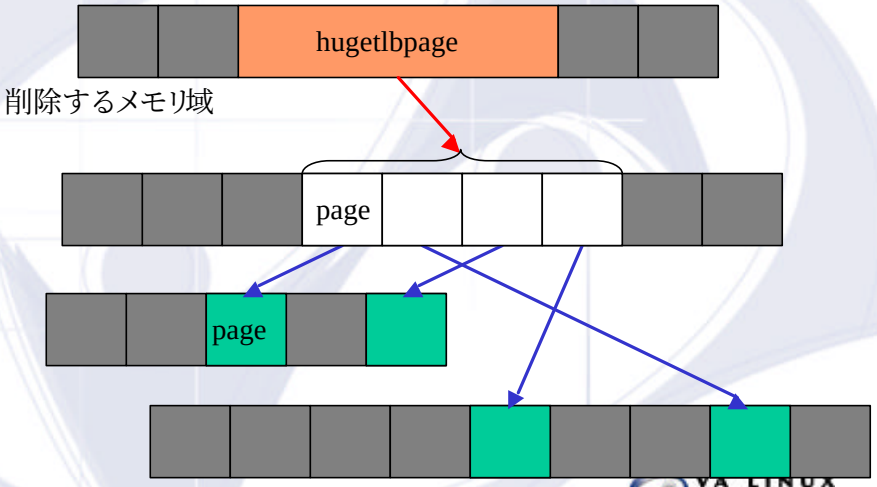
hugetlbfsとHugeTlbPage



HugeTlbPageのホットプラグ(2)

- **二段階のメモリマイグレーション**
  - 動的HugeTlbPage確保。最終的には、汎用デフラグ処理(Marcelo検討開始)を採用予定
  - HugeTlbPage自体の移動
- **ページングの基本機能の実装**
  - ページフォルト処理。最終的にはIntel技術者が開発中のものを採用予定
  - objrmap構造の導入と、プロセス空間からのunmap処理の実装

二段階のメモリマイグレーション



課題

- **動的削除可能なメモリ領域の指定方法の決定**
  - ゾーン単位？それともももっと小さな別の単位？
- **NUMAの考慮**
- **nonlinearマッピング(MAP_POPULATE指定のmmap)対応objrmapの実装**
- **メモリマイグレーション速度の制御**
- **ドキュメント整備**

今後の目標

- **カーネル空間のメモリ交換**
- **RDMA対応**
- **ネットワーク障害の扱い**
- **仮想マシンへの(実験的な)適用**
- **その他**

VA LINUX SYSTEMS JAPAN

カーネル空間のメモリ交換

- **DMAアドレスの交換**
  - DMAバッファ。デバイスドライバとの連携、強制停止と再開
- **物理アドレスの交換**
  - IA32であれば、PTEやGDTなど
- **論理アドレスの交換**
  - 大部分のカーネルデータはこれで十分
  - メモリ大喰らいslabへの対応、inode/dentryなど。

VA LINUX SYSTEMS JAPAN

協力者募集

- **開発参加者、スポンサーはいつでもwelcome**
  - lhms-devel@lists.sourceforge.net
- **利用技術を開発する者が現れると面白い**

VA LINUX SYSTEMS JAPAN