Technology Consulting Company
Research, Development &
Global Standard

BitVisorのUEFI対応

榮樂 英樹
株式会社イーゲル

2013-12-06 BitVisor Summit 2

igel

UEFI概要

BIOSとは? UEFIとは?

igel

- BIOS: Basic Input/Output System
  - 1980年代に登場したIBM PCと、互換性があるファームウェア
  - ハードウェアの進化に伴って様々な拡張がなされてきた
  - 古い
- UEFI: Unified Extensible Firmware Interface
  - OSとファームウェアの間のインターフェイスを決める仕様
  - 元になったExtensible Firmware Interfaceは2000年前後に登場、当初はIA-64 (Itanium) ワークステーション・サーバーが採用
  - IA-64, IA-32, AMD64のほか、ARMでの採用例もある
  - 新しい

パーソナルコンピューターにおけるBIOSとUEFIの比較

igel

	BIOS環境	UEFI環境
CPU動作モード	リアルアドレスモード/仮想8086モード	プロテクトモード(32bit/64bit)
呼び出し方法	ソフトウェア割り込み(int命令)	関数呼び出し(call命令)
メモリ管理	メモリ容量/メモリマップの提供	メモリマップの提供/ページ単位の確保・解放
不揮発性メモリアクセス(起動デバイス変更等)	APIなし (ユーザー向けのBIOS Setupのみ)	APIあり
ファイルシステム	なし	あり (FAT)
ファームウェアが読み込んで実行するプログラム	ストレージのMBR(CD-ROM起動時を除く)	ファイルシステム上の**UEFIアプリケーション**

UEFIアプリケーションとは?

igel

- UEFI環境で実行可能なプログラムで、拡張子はEFI
- 通常使われるものはOSを読み込むだけだが、シェルなども存在する
- ファイルフォーマットはPortable Executable (PE)
  - フォーマットはMicrosoft Windowsで使われるものと同一だが、サブシステムが異なり0x0A (EFI application) となっている
  - 物理アドレスと仮想アドレスが一致する環境で実行されるため、任意のアドレスに読み込めるように、再配置可能にされている
  - 動的リンクライブラリ等は使用されない
- 呼び出し規約はMicrosoft Windowsと同一
  - 64ビット環境では、GNU/Linuxで一般的なSystem V AMD64 ABIと異なる点に注意

UEFIアプリケーションプログラミング

igel

- エントリポイント
  - typedef
    EFI_STATUS
    (EFIAPI *EFI_IMAGE_ENTRY_POINT) (
    IN EFI_HANDLE ImageHandle,
    IN EFI_SYSTEM_TABLE *SystemTable
    );
  - EFI_SYSTEM_TABLEには、関数ポインターを含む構造体へのポインターなどが含まれる
- Hello, worldの例

```
int _start (EFI_HANDLE i, EFI_SYSTEM_TABLE *s) {
    s->ConOut->OutputString (s->ConOut, L“Hello, world.¥r¥n”);
}
```

ServicesとProtocols

igel

- Services
  - Boot services: OSが起動するまで使用可能なサービス。画面出力、キーボード入力、ストレージ入出力など。
  - Runtime services: OS起動後も使用可能なサービス。リアルタイムクロック、不揮発性メモリアクセスなど。
- Protocols
  - ストレージ、ファイルシステム、ネットワークなどのインターフェイスはprotocolとして定義されていて、拡張可能になっている。

Protocols使用例: 自分自身の実行ファイルを読み込む

igel

※動作未確認

```
int _start (EFI_HANDLE i, EFI_SYSTEM_TABLE *s) {
    EFI_LOADED_IMAGE_PROTOCOL *l;
    EFI_SIMPLE_FILE_SYSTEM_PROTOCOL *f;
    EFI_FILE_HANDLE v, a;
    UINTN len = 100;
    char buf[100];

    s->BootServices->HandleProtocol (i,
    &LoadedImageProtocol, &l);
    s->BootServices->HandleProtocol (l->DeviceHandle,
    &FileSystemProtocol, &f);
    f->OpenVolume (f, &v);
    v->Open (v, &a, l->FilePath, EFI_FILE_MODE_READ, 0);
    a->Read (a, &len, buf);
```

定義済みGUID

UEFIアプリケーションの開発環境

igel

- gnu-efi – GNU toolchainを使用
- EDK – Microsoftのコンパイラーを使用
- EDK II – クロスプラットフォーム

gnu-efi (1/2)

igel

- GNU toolchainを使用するためGNU/Linux環境で手軽に使用可能
- rEFItで使用されている
- 以下のようにして使うらしい:

```
gcc -c -nostdinc -I/usr/include/efi/ -I/usr/include/efi/ia32/ -fpic
    -fshort-wchar -fno-strict-aliasing -fno-merge-constants a.c
gcc -nostdlib -Wl,-T,/usr/lib/elf_ia32_efi.lds -Wl,-Bsymbolic
    -shared /usr/lib/crt0-efi-ia32.o a.o -lefi -lgnuefi -lgcc -o a.so
objcopy -j .text -j .sdata -j .data -j .dynamic -j .dynsym -j .rel
    -j .rela -j .reloc --target=efi-app-ia32 a.so a.efi
```

gnu-efi (2/2)

- 64bitバイナリを生成するには
  - ABIの差異の吸収: #define EFIAPI __attribute__ ((ms_abi))
  - ライブラリは上のような属性を使わずuefi_call_wrapperという関数を通じて呼び出しを行うように実装されており、この関数を自分で用意する必要がある。これは例えば以下のように実装する:

```
pop %rsi
xchg %rcx,%rdx
push %r9
push %r8
push %rdx
push %rcx
call *%rdi
push %rsi
ret $32
```

EDK

igel

- Microsoftのコンパイラーを使用するEFI開発キット
- GCCで使用する場合:
  - ヘッダーファイルは一部修正が必要だがほぼそのまま使用可能
  - MinGWターゲットのクロスコンパイラーを使用するのが簡単(DebianやUbuntu環境では公式パッケージでインストール可能)
  - Windows用のバイナリを生成して、バイナリの先頭から0xDCバイト後ろにあるサブシステムを0x0A (EFI application) に変更すればよい
- BitVisorでは、EDKから一部ヘッダーファイルを取り出して使用

EDK II

igel

- クロスプラットフォームなUEFI開発環境
- GNU/Linux対応
  - ヘッダーファイルの修正が不要
  - GCCのMinGWクロスコンパイラーを構築するスクリプトなどが含まれる
- BitVisorでも、これを使ったほうがよかったかも

BITVISOR実装

BIOS環境におけるBitVisorの起動方法

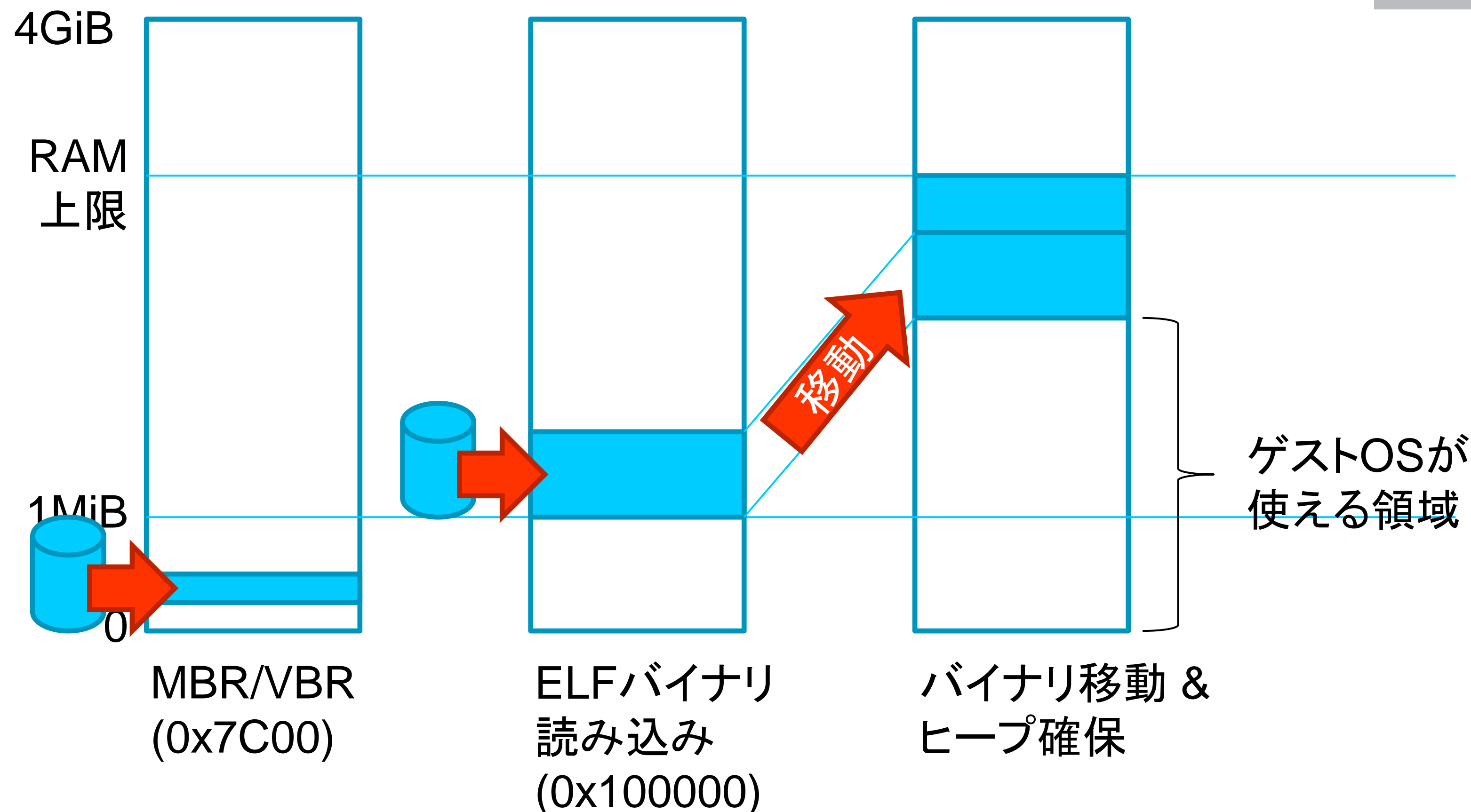
igel

- GNU GRUB
  - 有名な高機能ブートローダー
- VMMローダー
  - BitVisorソースツリーのboot/loaderディレクトリにある小さなプログラム
  - MBRなどに書き込んで使用可能

実行されるプログラムの順番:

- MBR→GNU GRUB→BitVisor→MBR/VBR→...
- MBR (bootloader)→BitVisor→VBR→...
- MBR (bootloaderusb)→BitVisor→MBR→...

BIOS環境におけるBitVisor起動時のメモリの使い方



– BIOSはメモリマップを提供するだけなので、使用状況をそれぞれが気を付けながら使う
– ELFバイナリのアドレス0x100000はGNU GRUB対応のため

UEFI環境における起動方法検討

igel

- UEFI用のGNU GRUB
  - GNU GRUBはバイナリ実行前にUEFIのExitBootServices() を呼び出す (OSがハードウェアにアクセスできるように)
  - ExitBootServices() の後はboot servicesが一切使えなくなる
  - BitVisorは自身の起動後にOSを起動したいのに、boot servicesが使えないとOSを起動することはできない

UEFI環境におけるBitVisorのGNU GRUB対応は不可能

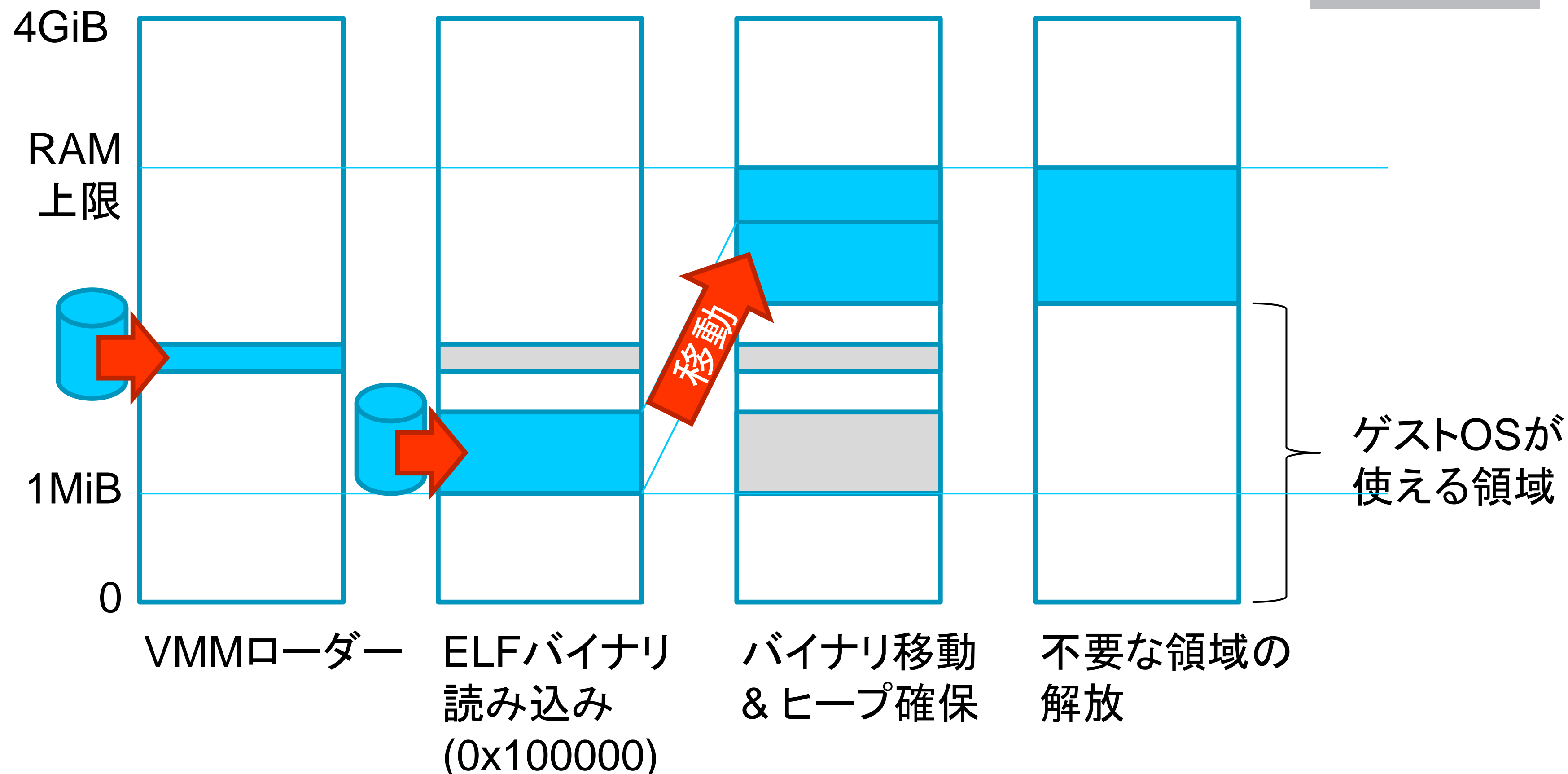
- UEFIアプリケーション化
  - 既存コードの大がかりな変更が必要 (ABI、メモリ管理、プロセスなど)
- UEFI用のVMMローダー開発

UEFI用のVMMローダー開発

igel

- BitVisorを読み込み、実行して、仮想マシン上に切り替わった後、終了するUEFIアプリケーションを開発する
  - VMM本体は今まで通りELFバイナリを使用する
  - BootOrderの最初にVMMローダーを入れることで、BitVisor起動後にローダーが終了すると、次のBootOrderに書かれたOSが起動する
- UEFIによるメモリの確保が必要となる
  - AllocatePages() にAllocateMaxAddressを指定して呼び出すことで、ファームウェアが空き領域を見つけてくれる
  - メモリタイプを例えばEfiUnusableMemoryとしておくことで、ファームウェア、他のUEFIアプリケーションやOSがその領域を使用することはなくなる

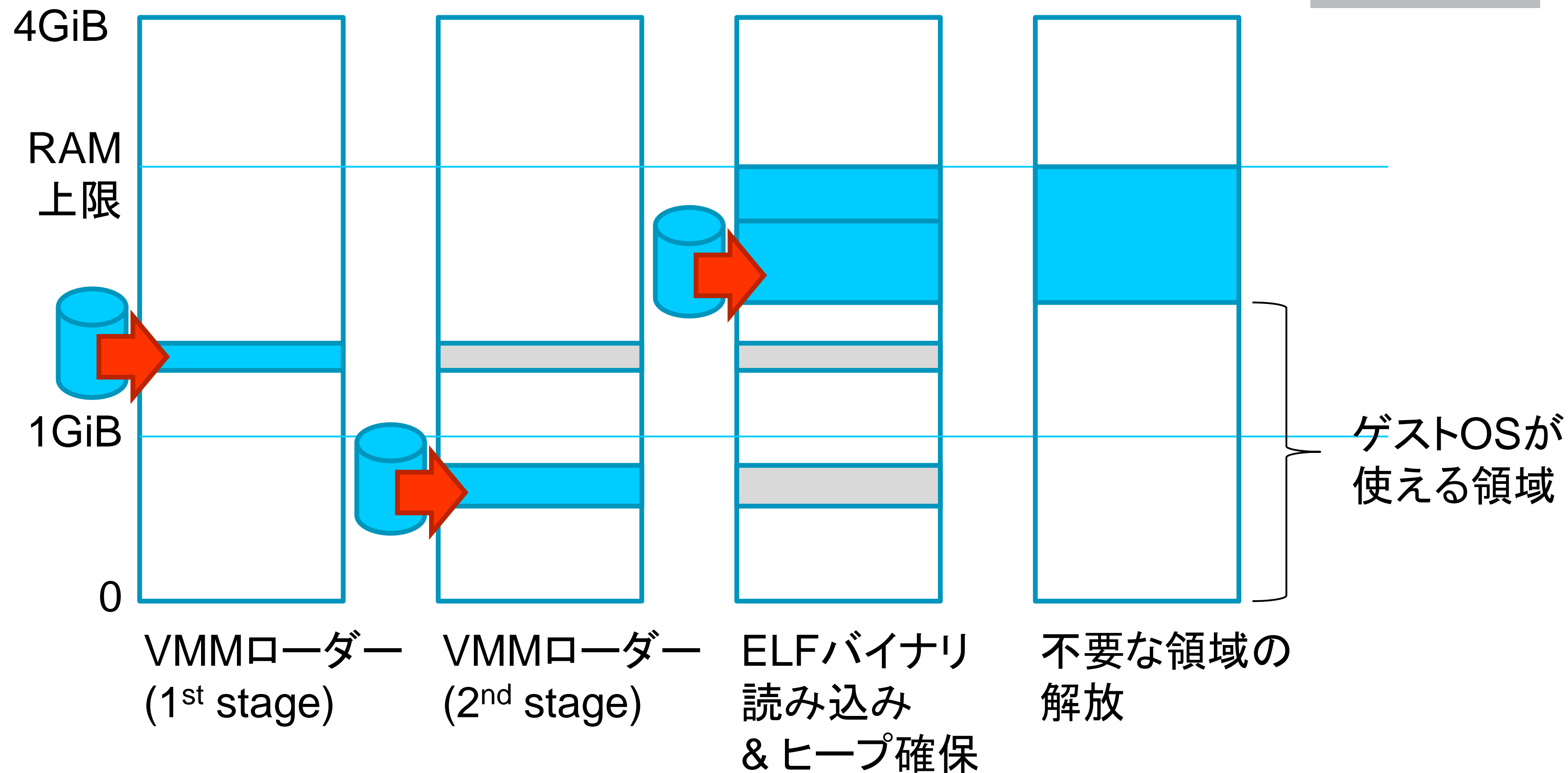
UEFI環境においてもBIOS環境と同じメモリの使い方をすると



- 1MiBから始まる領域をアドレス固定で確保してもいいのか
- GNU GRUB対応しないならば、ELFバイナリ全体を後から移動する必要はなく、目的のアドレスに直接読み込めばよいはず

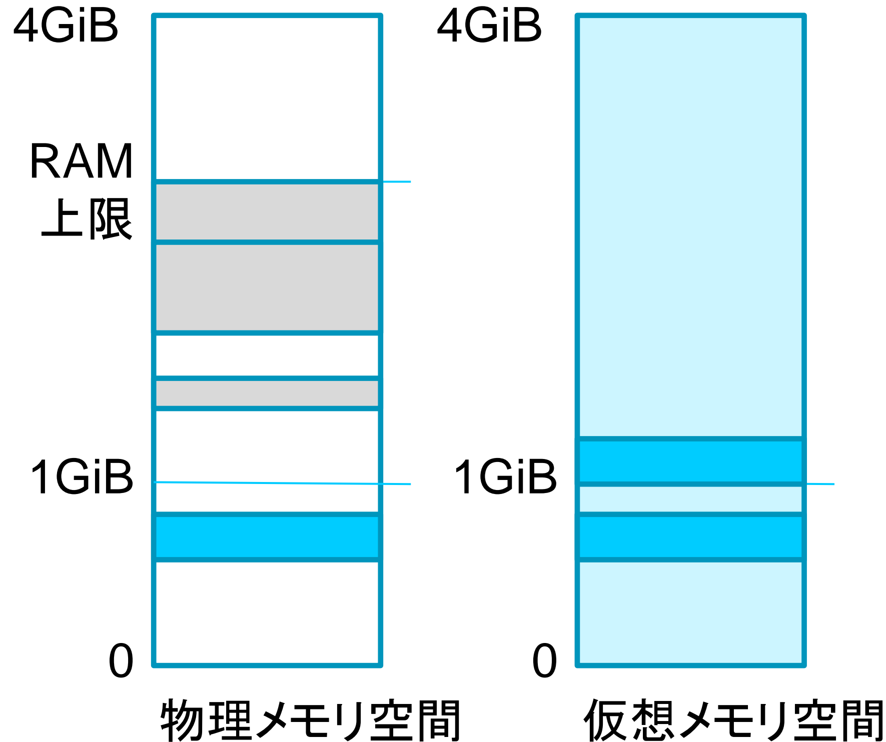
UEFI環境におけるBitVisor起動時のメモリの使い方

igel



- VMMローダー (1st stage) はUEFIアプリケーション
- VMMローダー (2nd stage) はELFバイナリの先頭64KiB

VMMローダー (2nd stage)について

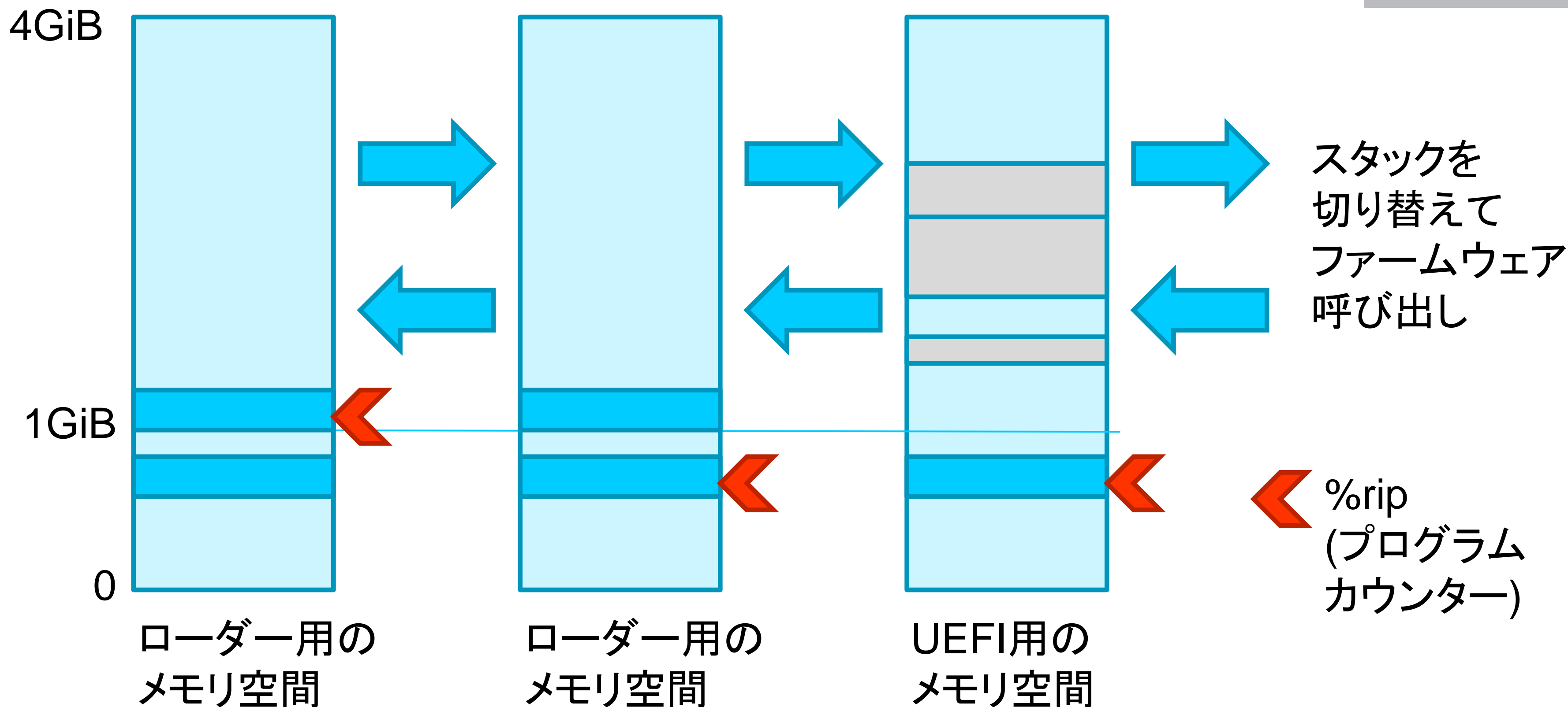


- VMMの仮想アドレスにマップして動作する
  - 1GiB未満を使用するのは簡略化のため

- VMMの一部なので、VMMが必要とするメモリサイズやアラインメントの情報を持っていて、適切なメモリを確保できる

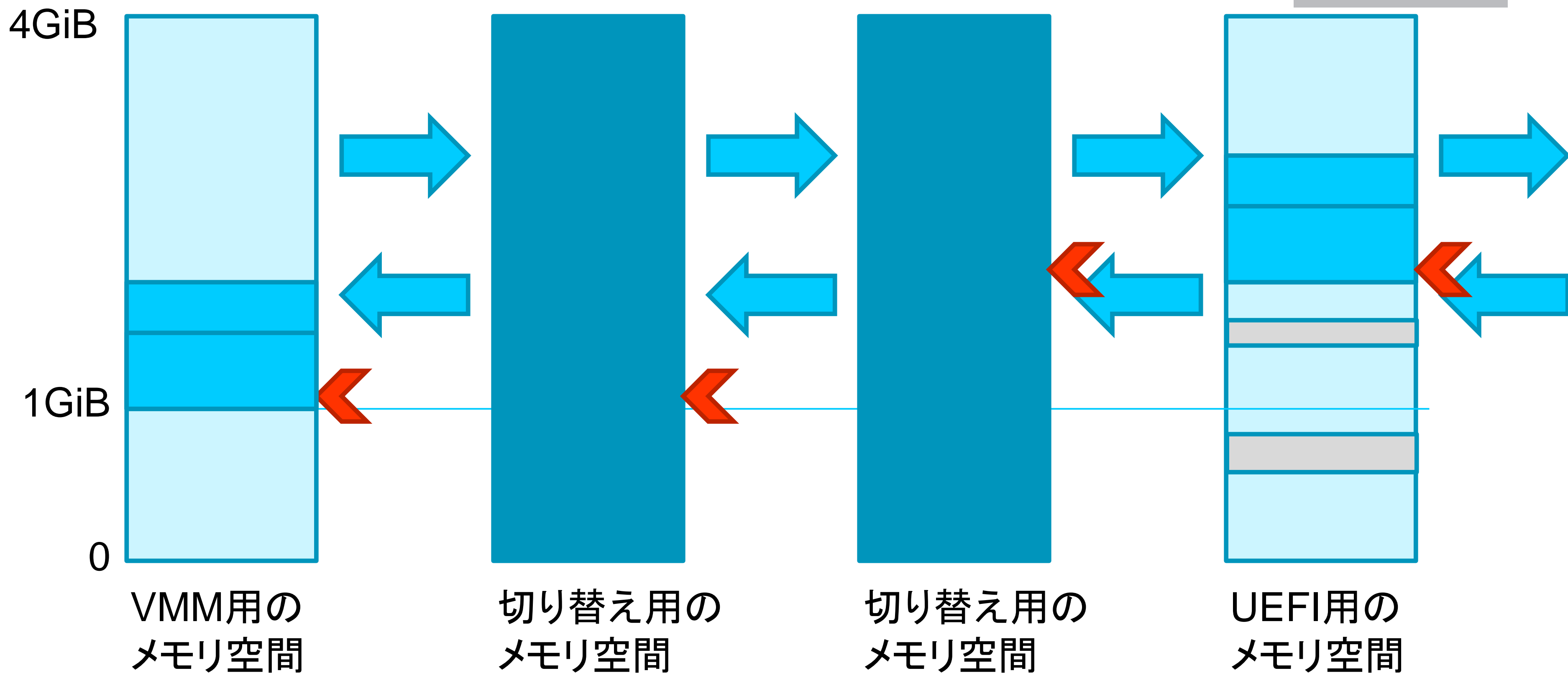
- UEFIファームウェア呼び出しは、物理アドレスと仮想アドレスが一致する状態に移行して行う

VMMローダー (2nd stage) からのUEFIファームウェア呼び出し



- 1GiB未満の領域を利用してメモリ空間を切り替える
- VMMからのBIOS呼び出しでも同じような手法を使っている

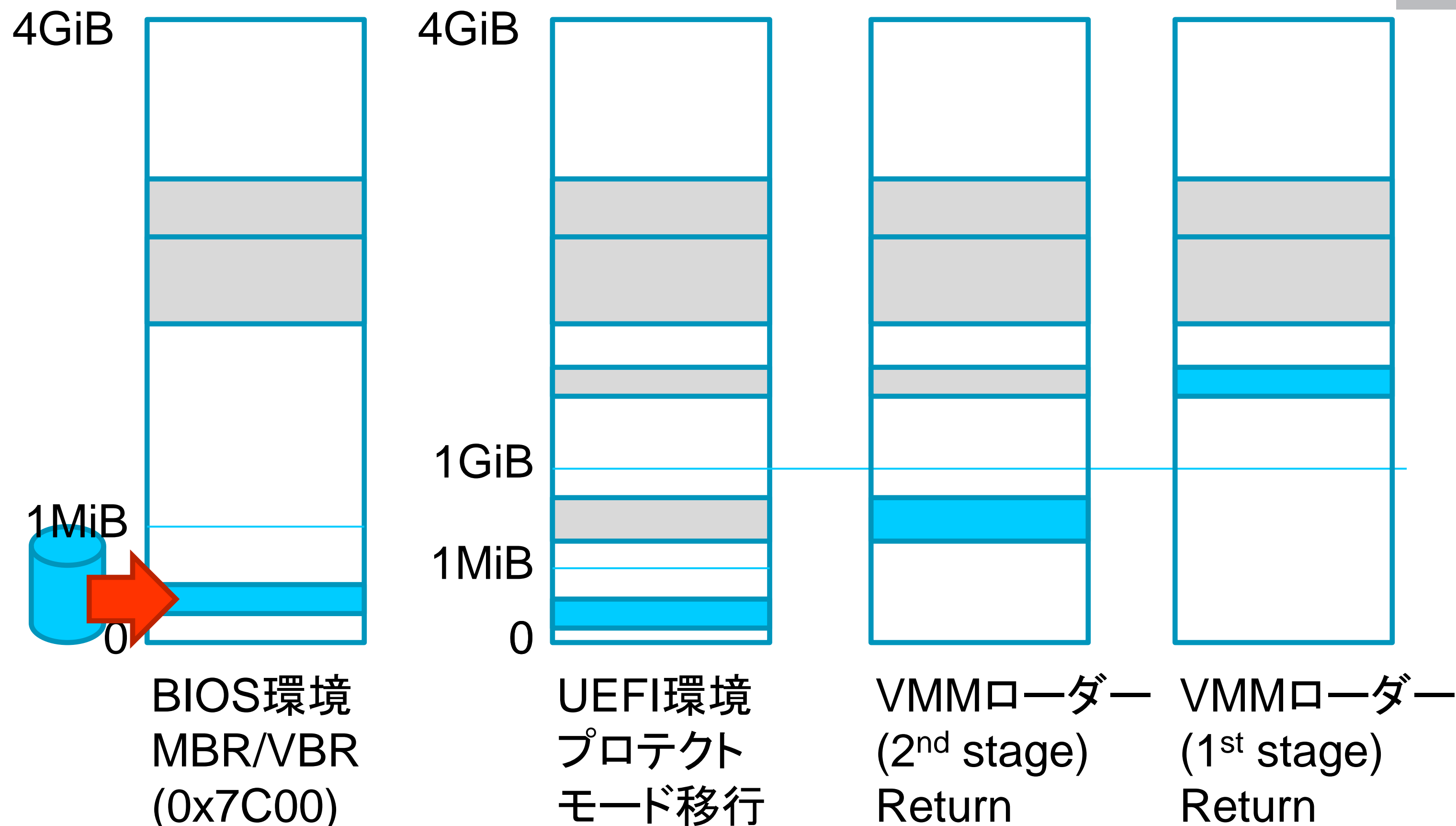
VMMからのUEFIファームウェア呼び出し



- 切り替え用のメモリ空間 (全体が同じページを指す) を利用してメモリ空間を切り替える

%rip

ゲストOSの実行



– UEFI環境では、リアルアドレスモードで実行開始する実装はそのままで、プロテクトモードへ移行しVMMローダーへ戻る小さなコードを、1MiB未満の領域に書き込んで実行する

ログ出力

igel

- テキストVRAMへの直接書き込み
  - UEFI環境では最初からハードウェア的にグラフィックスモードで起動するため実質使用できない
- グラフィックVRAMへの直接書き込み
  - グラフィックスアダプターが限定される
  - UEFI環境ではVGA BIOSフォントは使用できない (たぶん)
- 内蔵シリアルポート
  - 搭載機種が非常に限定される
- Ethernet
  - UEFIドライバーがある場合、アクセスが衝突する可能性がある

ログ出力のUEFI対応

igel

- UEFIのSimple Text Output Protocolを使用した出力
  - BSP (起動時に使われるCPUコア) から、boot servicesが使える間だけ使用可能
  - VMM起動直後からVMMローダーに戻るまでの間のログ出力用として使用するようにした
  - 仮想化支援機能が無効になっている時のように、起動中に致命的な状況を確認するのに有用である
- グラフィックVRAMへの直接書き込み
  - UEFI環境では内蔵フォントを使用するようにした
  - 容量削減のため、VGA BIOSフォントの8x16に対し、内蔵フォントは6x13とした

BitVisorのBIOS/GNU GRUB依存部分の例

igel

- UEFIでは不要な部分
  - callrealmode
  - install_int0x15_hook
  - loadbootsector
  - sync_cursor_pos
  - struct multiboot_info
  - move_vmm
  - etc.
- UEFIに適合するように変更が必要な部分
  - entry
  - get_ebda_address (ACPI)
  - alloc_realmodemem
  - getallsysmemmap
  - etc.

UEFIに適合するように変更が必要な部分

igel

- entry
  - 起動処理
  - 初期ページテーブル作成
- get_ebda_address (ACPI)
  - EFI_SYSTEM_TABLEのConfigurationTableから取得
- alloc_realmodemem
  - AllocatePages() を使用してメモリを確保
- getallsysmemmap
  - GetMemoryMap() を使用してメモリマップを取得
  - 不要かもしれない

32bit対応について

igel

現在の実装では32bitのUEFI環境には対応していない

理由:

- 32bitのUEFIファームウェアを搭載し仮想化支援機能に対応したPCは非常に少ない
  - Early 2006からMid 2007までのMac mini
  - 他には?
- Microsoft Windowsは64bitのみUEFIに対応している
- Intel CPUではVMX有効のままページングをオフにできないため対応が面倒

機種依存の問題

igel

機種依存の問題とは

igel

- UEFIで決められていない部分については、ファームウェアの実装が機種によって異なる
- 特に、ファームウェアがゲストOSとして実行されるというのは特殊な状況である

- 実際に遭遇した問題
  - ExitBootServices() でフリーズする (1)
  - ExitBootServices() でフリーズする (2)
  - ExitBootServices() の後で、640KiB RAMにアクセスできない
  - その他

ExitBootServices() でフリーズする (1) (昨年発表済み)

igel

機種: MacBook Air Mid 2009

- マルチプロセッサ/マルチコアの対応の問題
  - 2つ目のコアで、スピンロックを使って何かやっている
  - スピンロックがロック状態のままBitVisorがマルチプロセッサの初期化をして止めてしまうと、後でスピンロックをロックしようとしても解除するプログラムがないのでフリーズしてしまう
  - OSは、ExitBootServices() の後でマルチプロセッサの初期化を行うので、問題ないようだ。BitVisorはマルチプロセッサの初期化をした後でまたUEFIの機能を使用したいので、問題となる。

プロセッサ0	UEFI (boot loader)	BitVisor	UEFI
プロセッサ1	Firmware (lock/unlock)	BitVisor	
スピンロック			

ExitBootServices() でフリーズする (2)

igel

機種: Galleria FSH-X4 (AMD FX-4100搭載PC)

- マルチプロセッサ/マルチコアの対応の問題
  - マルチプロセッサの初期化をして2つ目以降のコアを止めてしまうと、ExitBootServices() でフリーズし先に進まなくなる
  - 以下の2つの簡単なUEFIアプリケーションを作成して実験したところ、やはり100%再現したため、VMMは関係ない
    - Local APICにアクセスしてINITをすべてのプロセッサに送る
    - ExitBootServices() を呼び出した後、直ちにResetSystem() を呼び出してシステムをリセットさせる

マルチプロセッサ初期化処理の変更が必要

ExitBootServices() でフリーズする問題の解決方法

igel

- 最初はLocal APICのアクセスをフックするようにしておき、マルチプロセッサの開始処理を行わない
  - フックする処理自体はAMD CPU対応のために実装済み
  - UEFI環境ではIntel CPUであっても最初はフックする
- ゲストOSがLocal APICにアクセスし、他のプロセッサにINITやStart-up IPIを送ろうとしたときに、マルチプロセッサ開始処理を行う
  - Intel CPUであれば、この時点でフックを登録解除する
- BitVisorの初期化処理の中に、全プロセッサで同期を取る部分があり、その部分には工夫が必要だった
- 実装はしたが一部環境でまだ問題が残っている

ExitBootServices() の後で、640KiB RAMにアクセスできない

igel

機種: Galleria FSH-X4 (AMD FX-4100搭載PC)

- ExitBootServices() の後、RAMのアドレス0～0x9FFFFの内容がすべて0xFFになっている
  - ExitBootServices() とResetSystem() を呼び出すテストプログラムは正常に動作するので、調査に手間取った
  - 原因: AMD CPU専用のキャッシュ設定 (Fixed-Range MTRRの拡張) がExitBootServices() の実行中に変化し (BitVisorが検知できない方法によって、SMMにより?)、整合性が保てなくなり、不適切な設定になっていた (memory-mapped I/Oでなければならないところがsystem memoryとなっていた)

AMD CPU専用のキャッシュ設定をpassthroughとして解決

その他の機種依存問題 (UEFIと関係あるかどうか不明)

igel

- BitVisor上ではOSが起動しない環境の例
  - Mac OS X 10.6.7 on MacBookAir1,1
    - cpus=1を付けると動作する
  - OS X 10.8.5 on MacBookPro10,2 / MacPro4,1
- BitVisor上ではSuspend-to-RAMが動作しない環境の例
  - Windows 8.1 Preview / Linux on Galleria FSH-X4 (AMD FX-4100搭載PC)
    - Linuxはmaxcpus=1を付けると動作する
  - OS X 10.8.2 on Macmini6,1
  - Mac OS X 10.7.5 on iMac12,1
  - Mac OS X 10.6.3 on MacPro3,1

BitVisorのUEFI対応 まとめ

igel

- UEFI用のVMMローダー開発
  - メモリ確保、ゲストOSの実行方法の変更
- UEFIファームウェア呼び出し実装
  - メモリ空間切り替え
- BIOS/GNU GRUB依存部分の変更
- 機種依存の問題への対応