高速な I/O デバイスへ対応するための システムソフトウェア研究

第 159 回 システムソフトウェアとオペレーティング・システム研究会

2023 年 5 月 16 日

安形 憲一

IIJ 技術研究所

発表内容

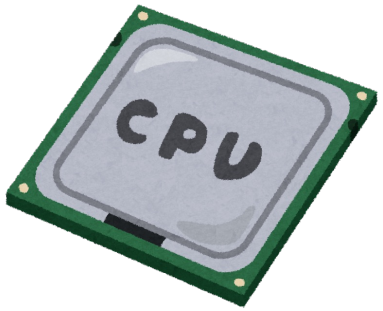
1. 研究の背景

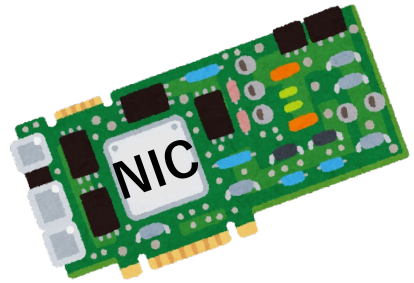
2. 仮想マシンについて
   - https://dl.acm.org/doi/10.1145/3582016.3582042
   - https://github.com/yasukata/ELISA

3. OS について
   - https://www.usenix.org/conference/atc23/presentation/yasukata
   - https://github.com/yasukata/zpoline

研究の背景

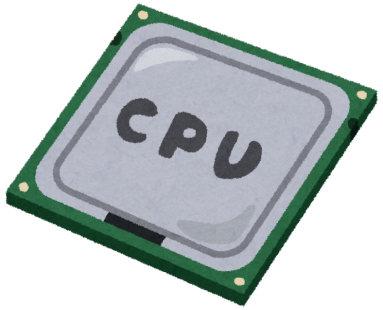
- CPU 性能と比較して I/O デバイスの高速化が著しい

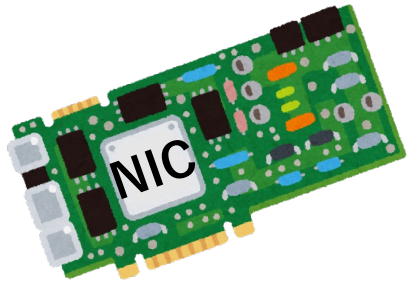




研究の背景

- CPU 性能と比較して I/O デバイスの高速化が著しい

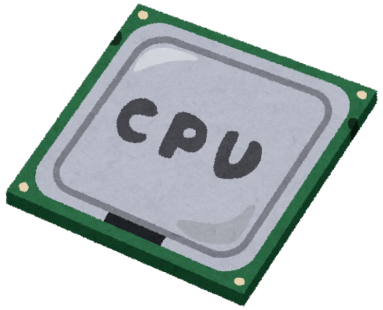


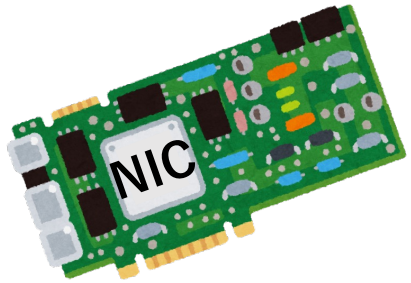


2012 年	Intel Core i7 4960X (6-core) **3.6** (base) / **4.0** (boost) GHz	Intel X540 **10** Gbps

研究の背景

- CPU 性能と比較して I/O デバイスの高速化が著しい

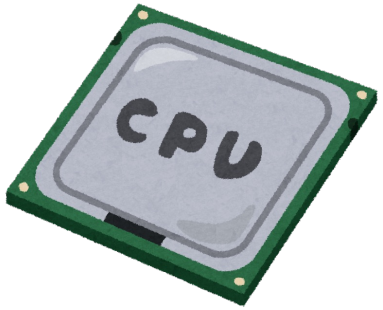


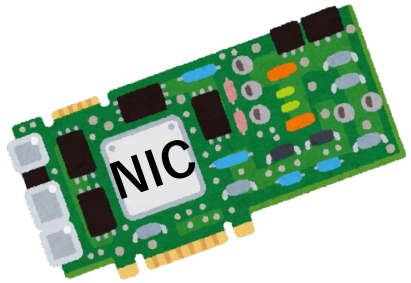


2014 年	Intel Core i7 6850K (6-core) **3.8** (base) / **4.0** (boost) GHz	Intel XL710 **40** Gbps
2012 年	Intel Core i7 4960X (6-core) **3.6** (base) / **4.0** (boost) GHz	Intel X540 **10** Gbps

研究の背景

- CPU 性能と比較して I/O デバイスの高速化が著しい

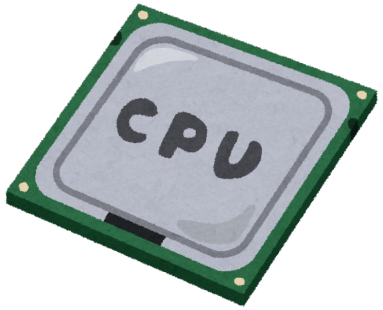


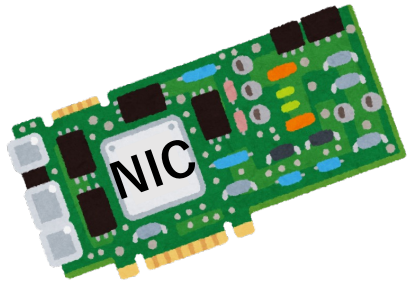


2022 年	Intel Core i9 13900KS (8P-/16E-core) **3.2** (base) / **6.0** (boost) GHz (P-core)	Intel E810 **100** Gbps
2014 年	Intel Core i7 6850K (6-core) **3.8** (base) / **4.0** (boost) GHz	Intel XL710 **40** Gbps
2012 年	Intel Core i7 4960X (6-core) **3.6** (base) / **4.0** (boost) GHz	Intel X540 **10** Gbps

研究の背景

- CPU 性能と比較して I/O デバイスの高速化が著しい

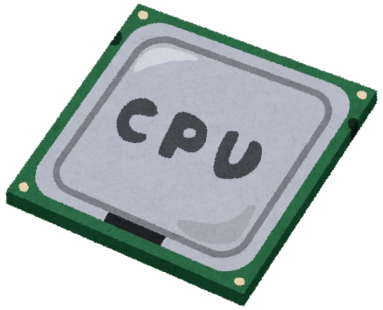




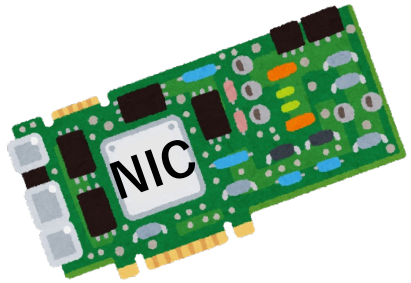
2022 年	Intel Core i9 13900KS (8P-/16E-core) **3.2** (base) / **6.0** (boost) GHz (P-core)	Intel E810 **100** Gbps
2014 年	Intel Core i7 6850K (6-core) **3.8** (base) / **4.0** (boost) GHz	Intel XL710 **40** Gbps
2012 年	Intel Core i7 4960X (6-core) **3.6** (base) / **4.0** (boost) GHz	Intel X540 **10** Gbps

研究の背景

- CPU 性能と比較して I/O デバイスの高速化が著しい



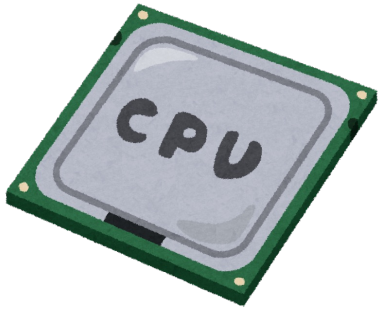
10 年間で
速度が 10 倍



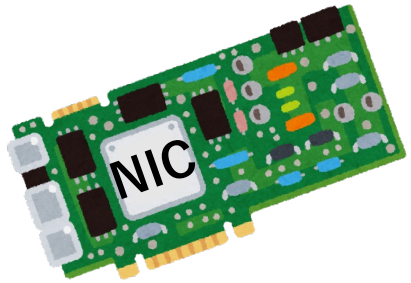
2022 年	Intel Core i9 13900KS (8P-/16E-core) **3.2** (base) / **6.0** (boost) GHz (P-core)	Intel E810 **100** Gbps
2014 年	Intel Core i7 6850K (6-core) **3.8** (base) / **4.0** (boost) GHz	Intel XL710 **40** Gbps
2012 年	Intel Core i7 4960X (6-core) **3.6** (base) / **4.0** (boost) GHz	Intel X540 **10** Gbps

研究の背景

- CPU 性能と比較して I/O デバイスの高速化が著しい



10 年間で
速度が 10 倍

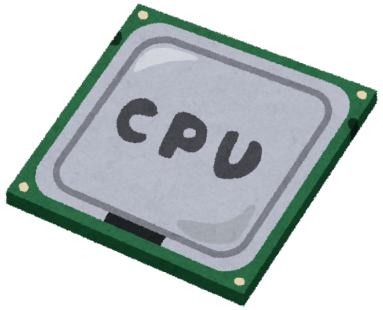


2022 年	Intel Core i9 13900KS (8P-/16E-core) **3.2** (base) / **6.0** (boost) GHz (P-core)	Intel E810 **100** Gbps
2014 年	Intel Core i7 6850K (6-core) **3.8** (base) / **4.0** (boost) GHz	Intel XL710 **40** Gbps
2012 年	Intel Core i7 4960X (6-core) **3.6** (base) / **4.0** (boost) GHz	Intel X540 **10** Gbps

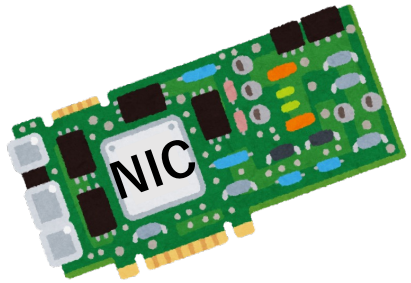
研究の背景

- CPU 性能と比較して I/O デバイスの高速化が著しい

ブースト時性能は 1.5 倍
ベース周波数は据え置き



10 年間で
速度が 10 倍

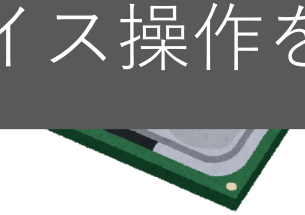


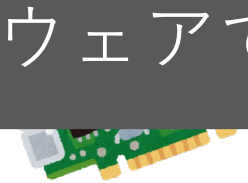
2022 年	Intel Core i9 13900KS (8P-/16E-core) **3.2** (base) / **6.0** (boost) GHz (P-core)	Intel E810 **100** Gbps
2014 年	Intel Core i7 6850K (6-core) **3.8** (base) / **4.0** (boost) GHz	Intel XL710 **40** Gbps
2012 年	Intel Core i7 4960X (6-core) **3.6** (base) / **4.0** (boost) GHz	Intel X540 **10** Gbps

研究の背景

CPU と比べ I/O デバイス性能の向上の方が速いことから
ソフトウェアの CPU 利用効率への要求が厳しくなっている
特に実際にデバイス操作を行うシステムソフトウェアで顕著

ブー
ベー

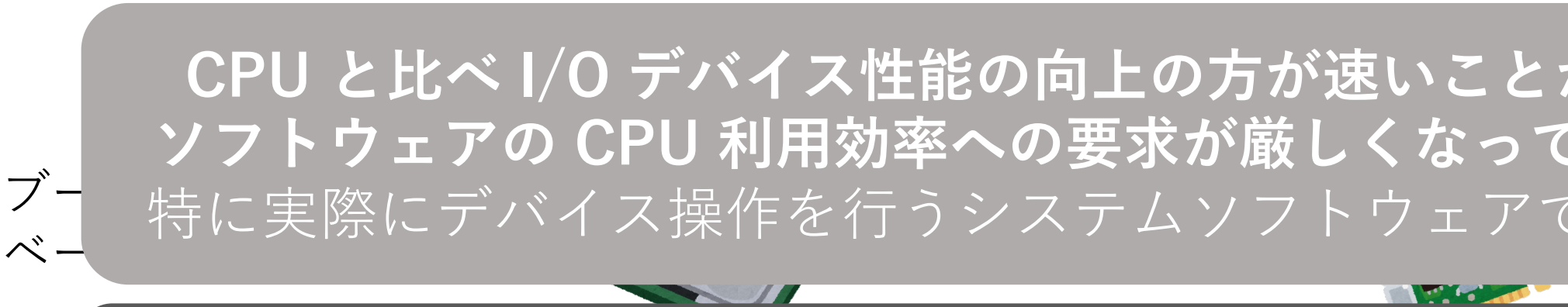




2022 年	Intel Core i9 13900KS (8P-/16E-core) **3.2** (base) / **6.0** (boost) GHz (P-core)	Intel E810 **100** Gbps
2014 年	Intel Core i7 6850K (6-core) **3.8** (base) / **4.0** (boost) GHz	Intel XL710 **40** Gbps
2012 年	Intel Core i7 4960X (6-core) **3.6** (base) / **4.0** (boost) GHz	Intel X540 **10** Gbps

研究の背景

CPU と比べ I/O デバイス性能の向上の方が速いことから
ソフトウェアの CPU 利用効率への要求が厳しくなっている
特に実際にデバイス操作を行うシステムソフトウェアで顕著



機能やコードパスを削ると高速化自体は可能なものの
なるべく重要な機能・特性を損いたくない

2014 年	Intel Core i7 6850K (6-core) **3.8** (base) / **4.0** (boost) GHz	Intel XL710 **40** Gbps
2012 年	Intel Core i7 4960X (6-core) **3.6** (base) / **4.0** (boost) GHz	Intel X540 **10** Gbps

研究の背景

CPU と比べ I/O デバイス性能の向上の方が速いことから
ソフトウェアの CPU 利用効率への要求が厳しくなっている
特に実際にデバイス操作を行うシステムソフトウェアで顕著

機能やコードパスを削ると高速化自体は可能なものの
なるべく重要な機能・特性を損いたくない

機能・特性を維持しつつ CPU 利用効率の向上を目指すと
解決されていない課題が結構ある
例：リソースの分離・共有、実装の拡張性・互換性・再利用性

研究の背景

CPU と比べ I/O デバイス性能の向上の方が速いことから
ソフトウェアの CPU 利用効率への要求が厳しくなっている
特に実際にデバイス操作を行うシステムソフトウェアで顕著

仮想マシンについて

と高速化自体は可能なものの
な機能・特性を損いたくない

機能・特性を維持しつつ CPU 利用効率の向上を目指すと
解決されていない課題が結構ある
例：**リソースの分離・共有**、実装の拡張性・互換性・再利用性

研究の背景

CPU と比べ I/O デバイス性能の向上の方が速いことから
ソフトウェアの CPU 利用効率への要求が厳しくなっている
特に実際にデバイス操作を行うシステムソフトウェアで顕著

仮想マシンについて

OS について

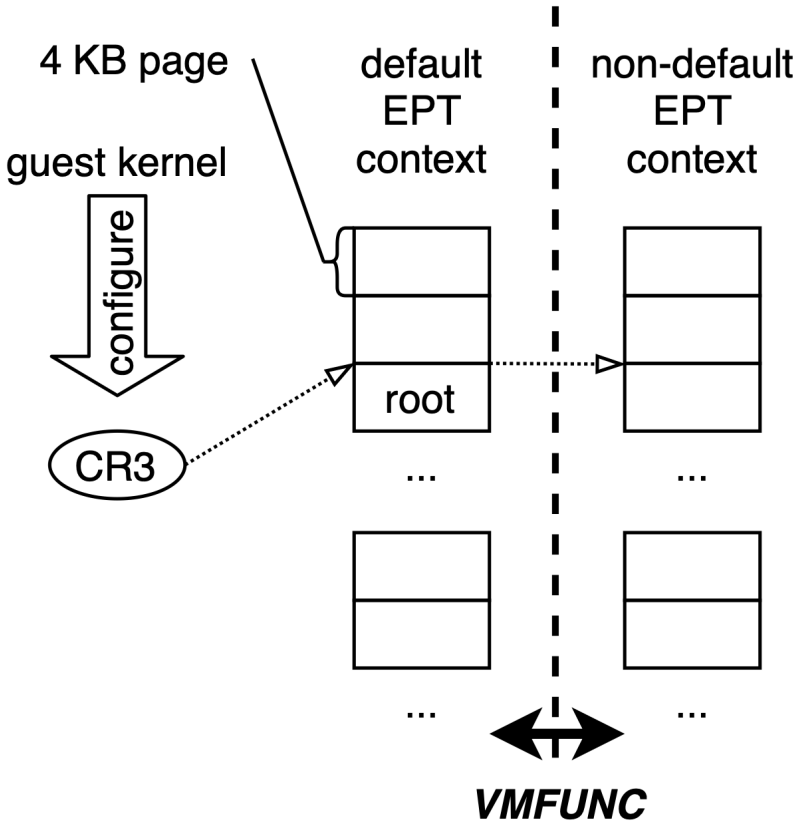
機能・特性を維持しつつ CPU 利用効率の向上を目指すと
解決されていない課題が結構ある
例：リソースの分離・共有、実装の拡張性・**互換性**・再利用性

仮想マシンについての研究

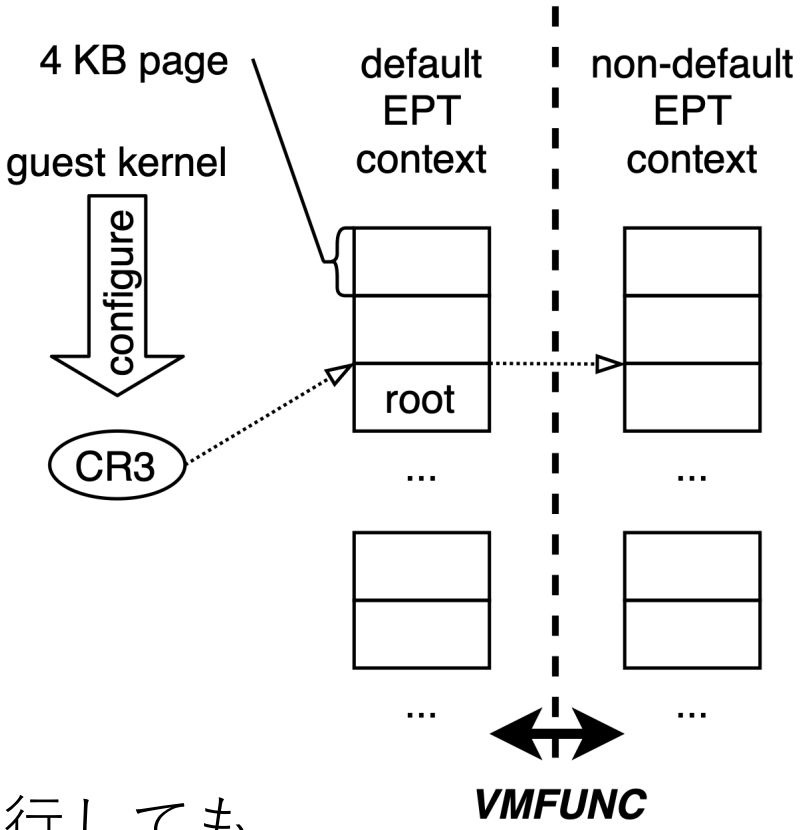
スライド

https://yasukata.github.io/presentation/2023/03/asplos2023/asplosc23main-p814-slides.pdf

ゲスト用ページテーブルの設定

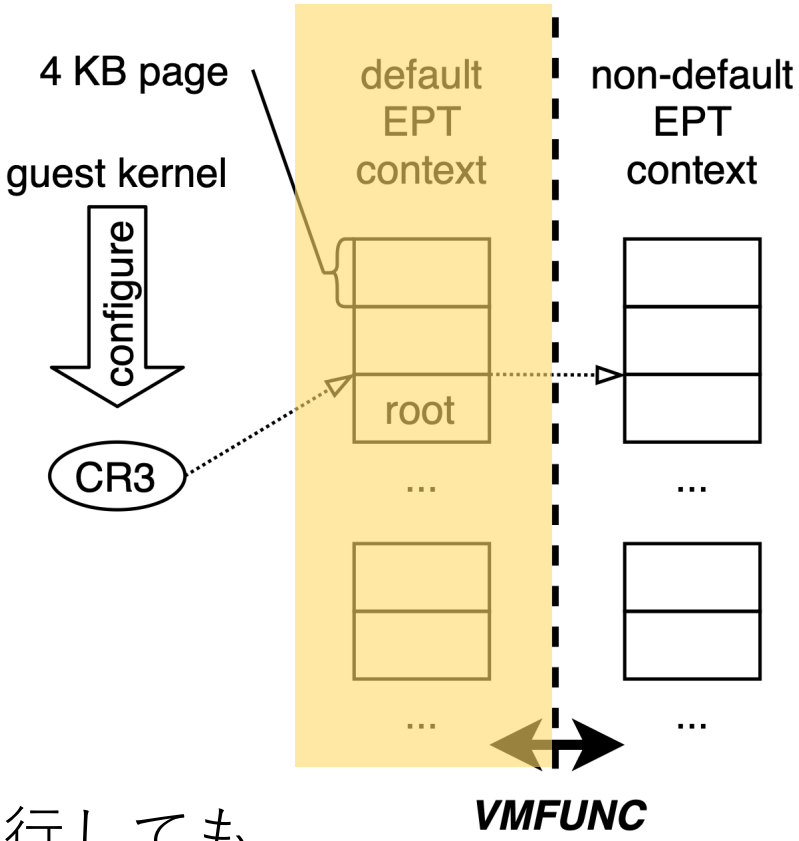


ゲスト用ページテーブルの設定



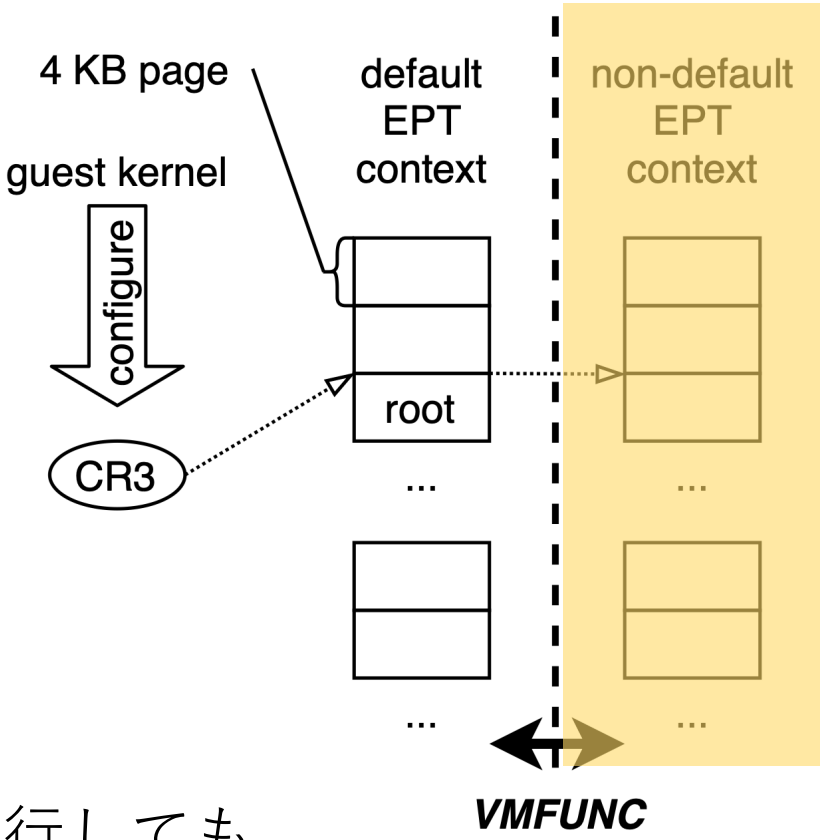
VMFUNC を実行しても
CR3 レジスタの値は変更されない

ゲスト用ページテーブルの設定



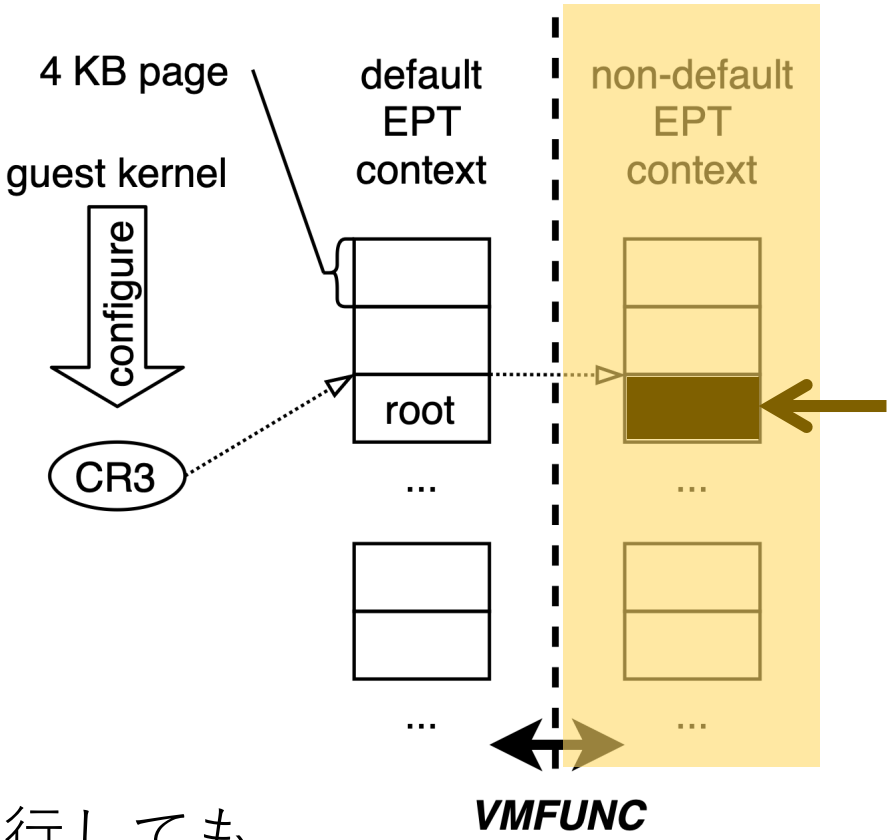
VMFUNC を実行しても
CR3 レジスタの値は変更されない

ゲスト用ページテーブルの設定



VMFUNC を実行しても
CR3 レジスタの値は変更されない

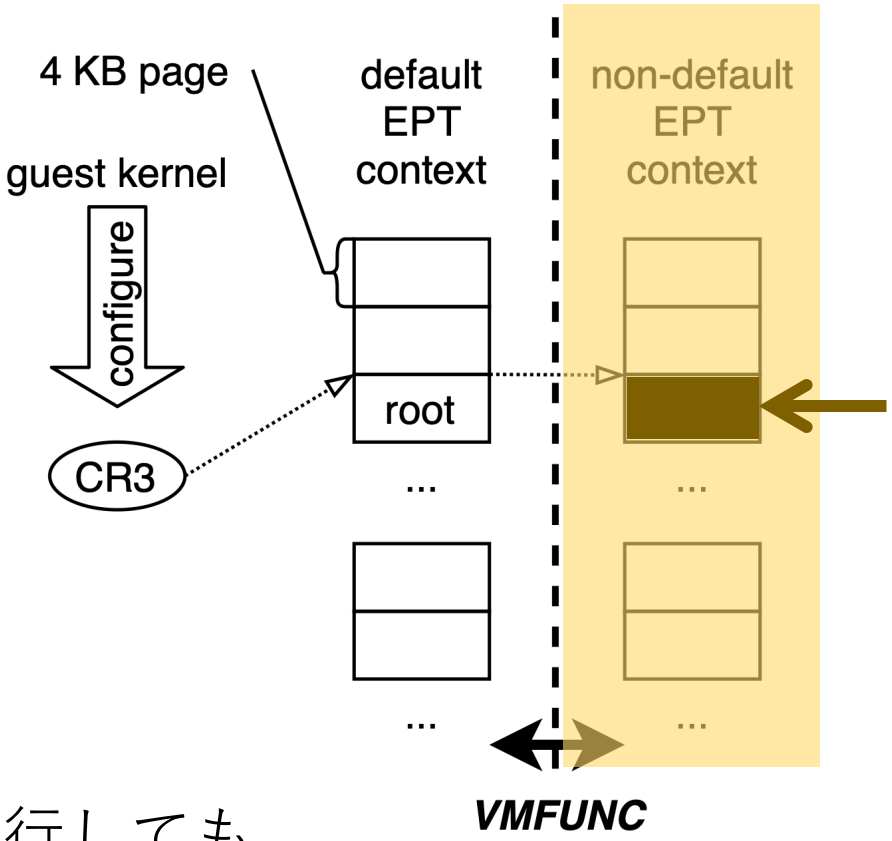
ゲスト用ページテーブルの設定



VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

VMFUNC を実行しても
CR3 レジスタの値は変更されない

ゲスト用ページテーブルの設定

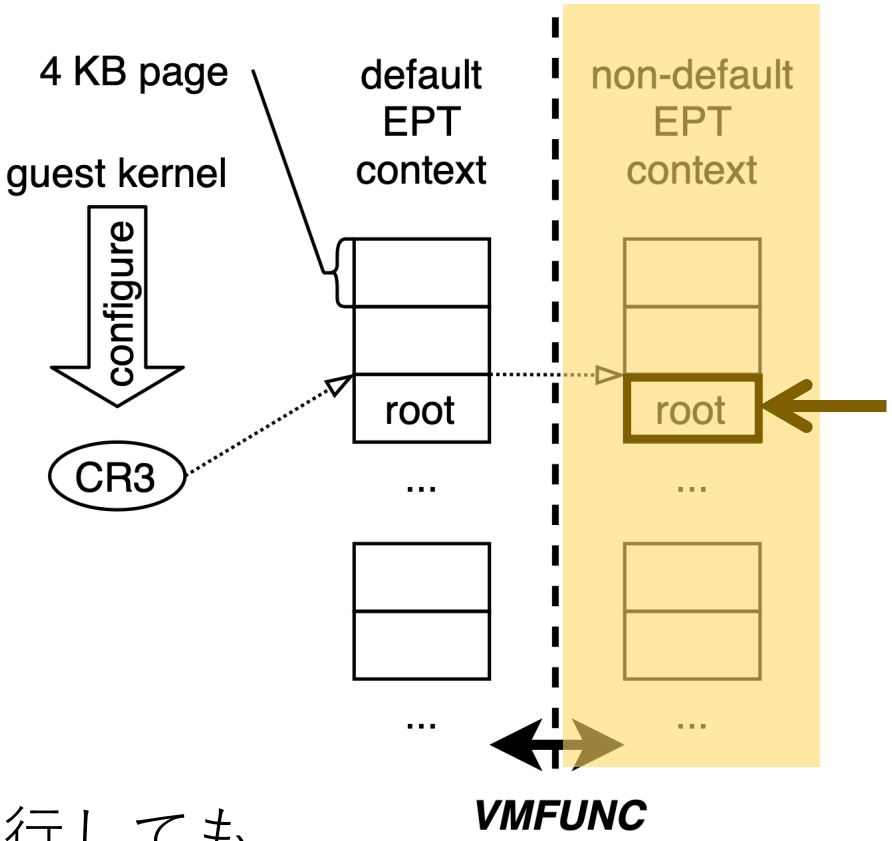


VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

VMFUNC を実行しても
CR3 レジスタの値は変更されない

ゲスト用ページテーブルの設定

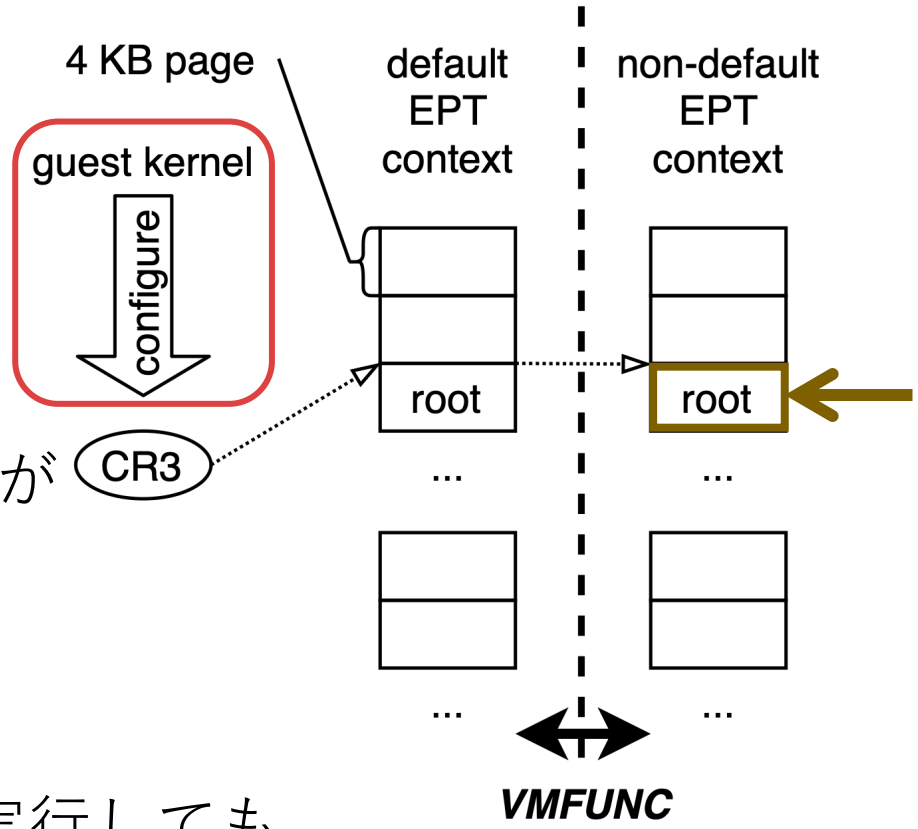


VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

VMFUNC を実行しても
CR3 レジスタの値は変更されない

ゲスト用ページテーブルの設定



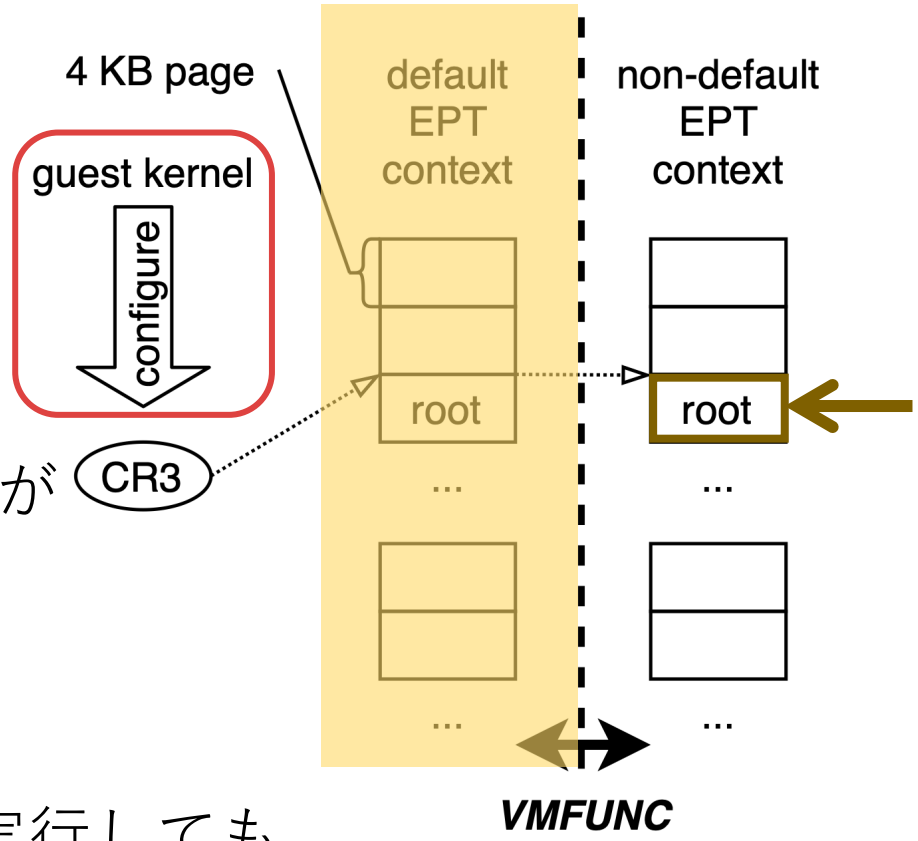
CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

VMFUNC を実行しても
CR3 レジスタの値は変更されない

ゲスト用ページテーブルの設定



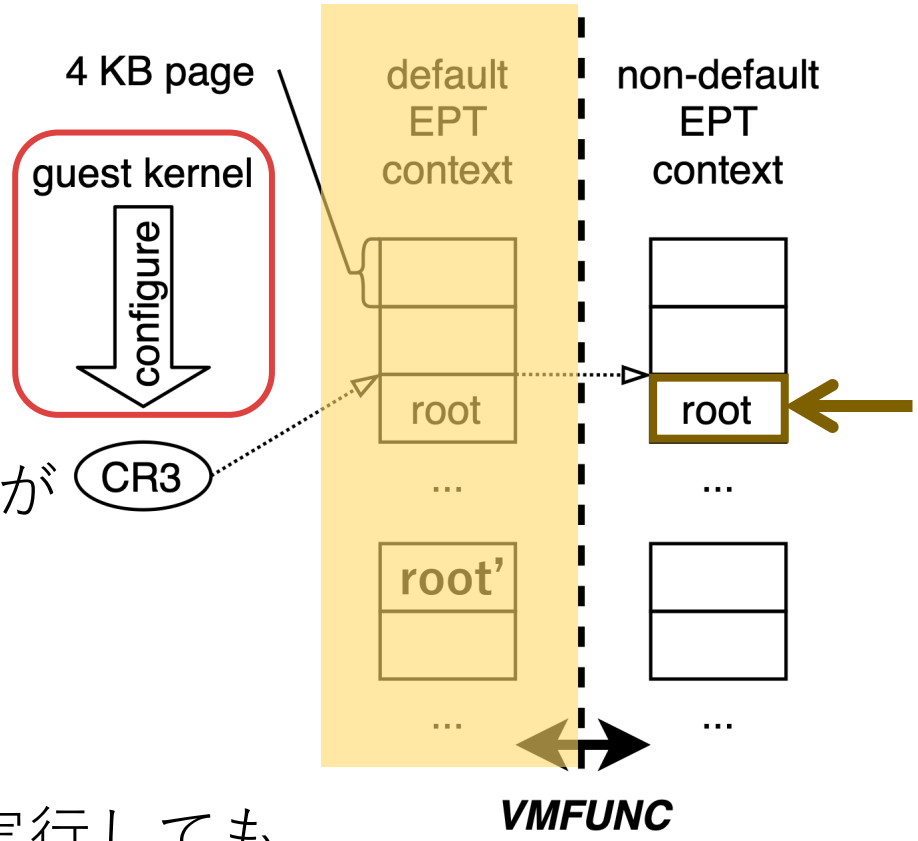
CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行しても
CR3 レジスタの値は変更されない

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

ゲスト用ページテーブルの設定



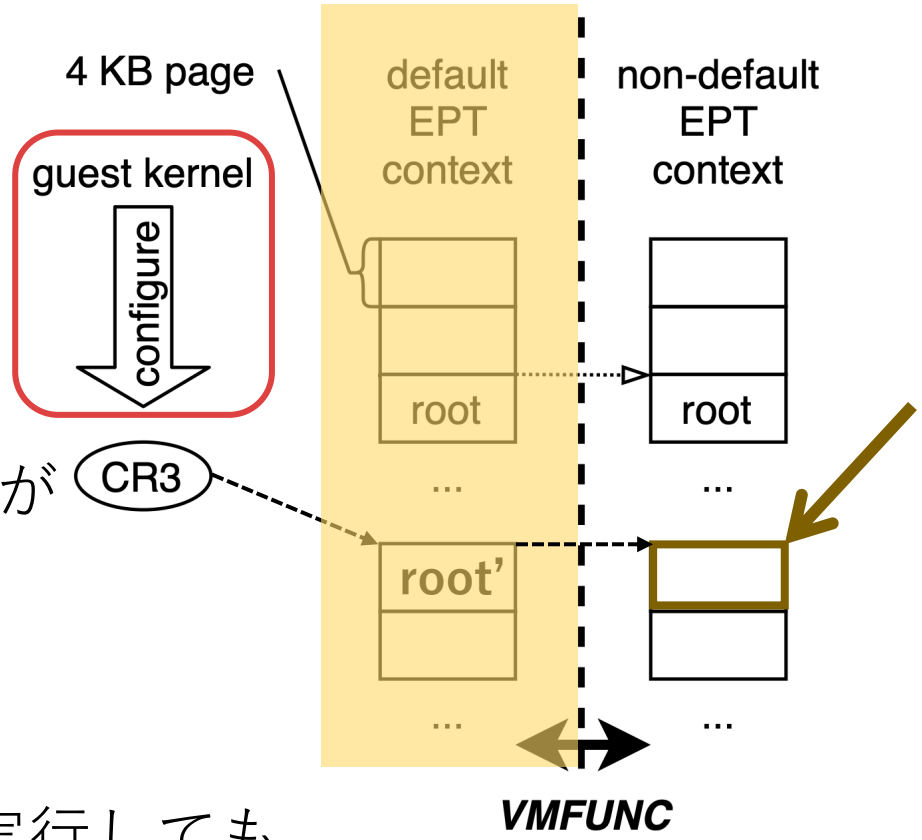
CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

VMFUNC を実行しても
CR3 レジスタの値は変更されない

ゲスト用ページテーブルの設定



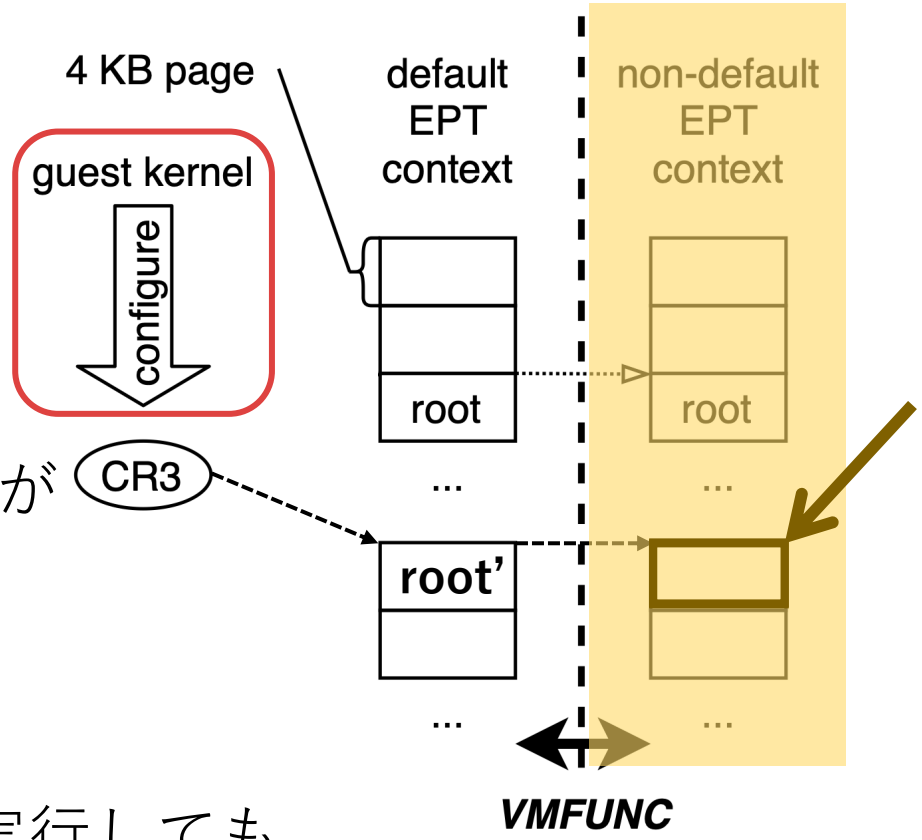
CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行しても
CR3 レジスタの値は変更されない

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

ゲスト用ページテーブルの設定



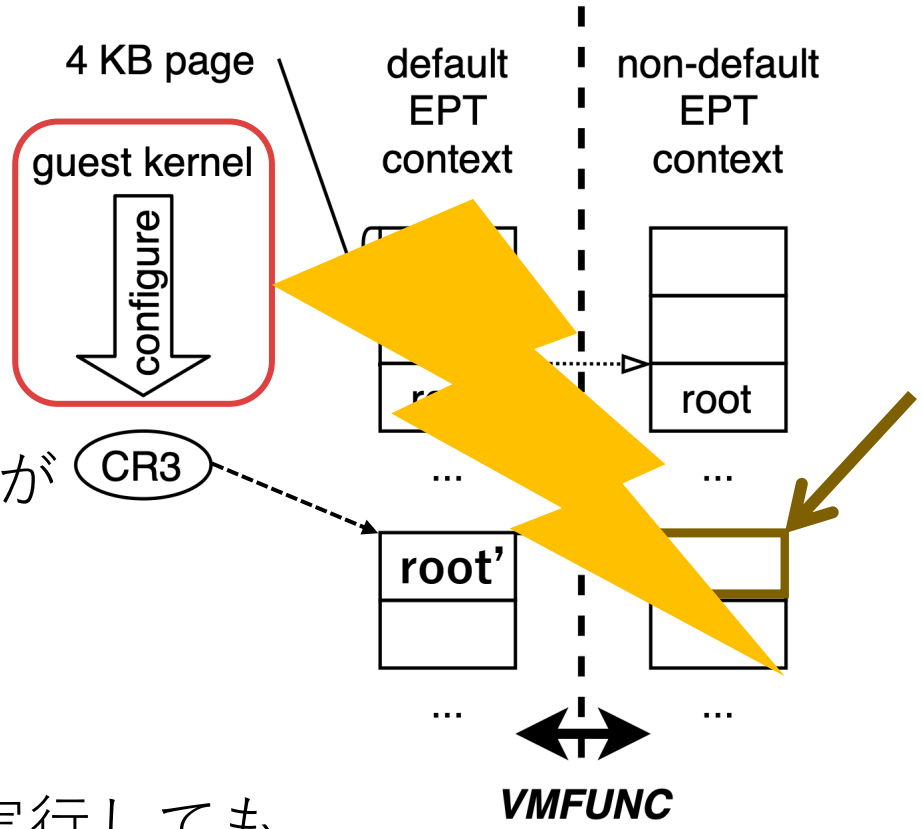
CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行しても
CR3 レジスタの値は変更されない

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

ゲスト用ページテーブルの設定



CR3 の値は
ゲストカーネルが
自由に変更する

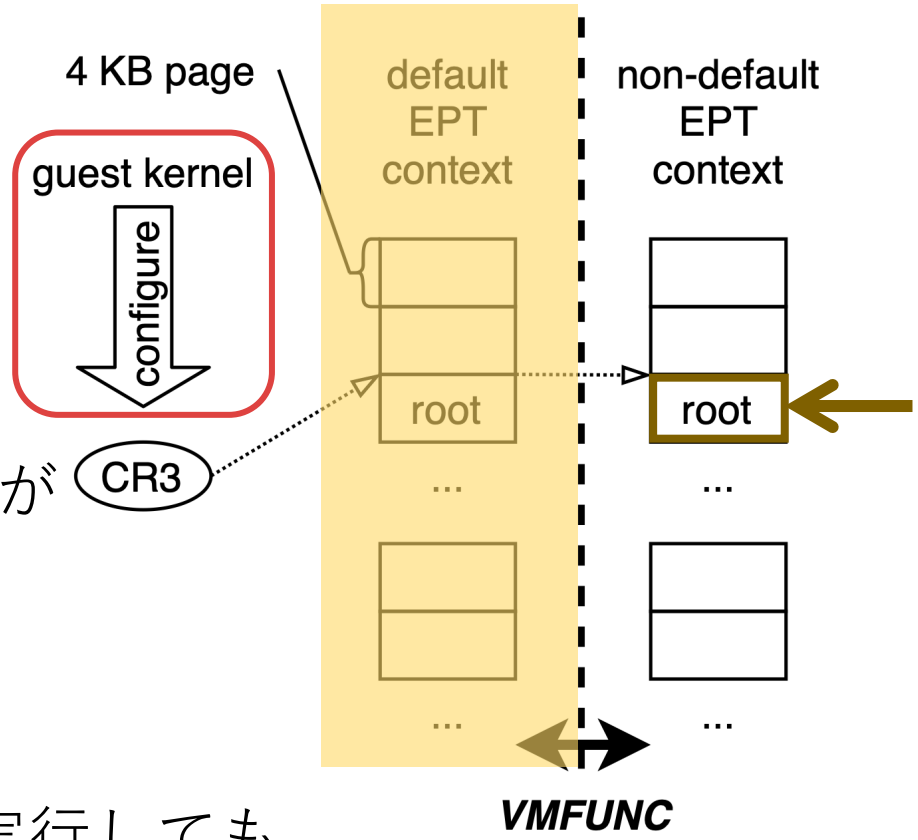
VMFUNC を実行しても
CR3 レジスタの値は変更されない

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

多くの場合、準備できていないと仮想マシンがクラッシュする

ゲスト用ページテーブルの設定



CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行しても
CR3 レジスタの値は変更されない

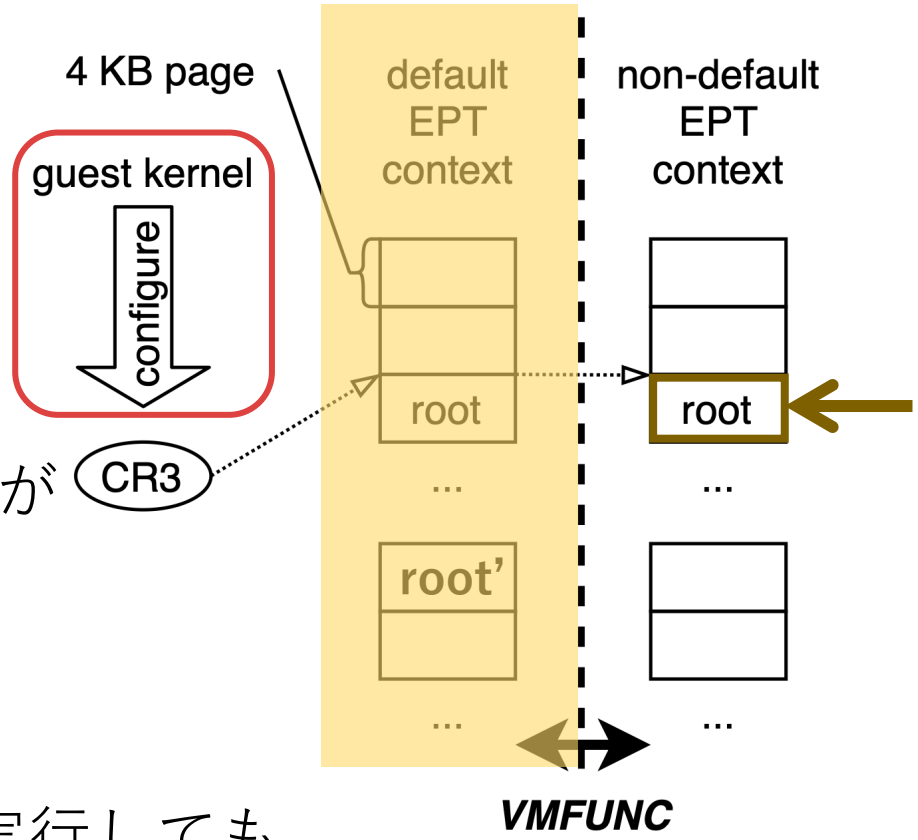
<u>よくある解決策</u>

CR3 アクセスをトラップ

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

ゲスト用ページテーブルの設定



CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行しても
CR3 レジスタの値は変更されない

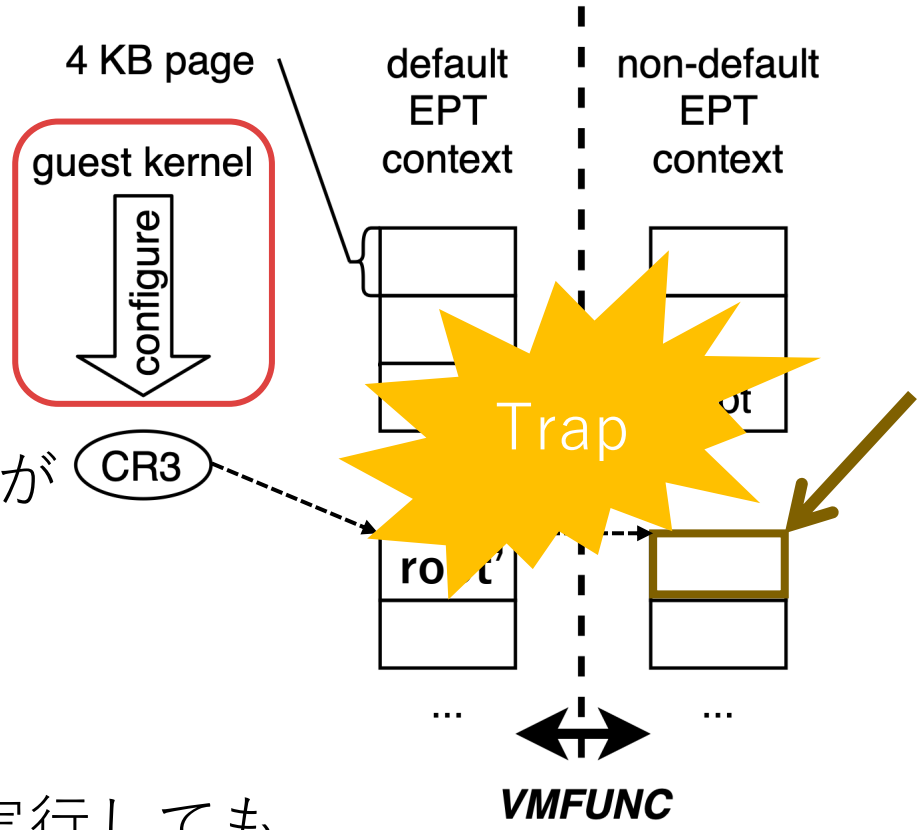
よくある解決策

CR3 アクセスをトラップ

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

ゲスト用ページテーブルの設定



CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行しても
CR3 レジスタの値は変更されない

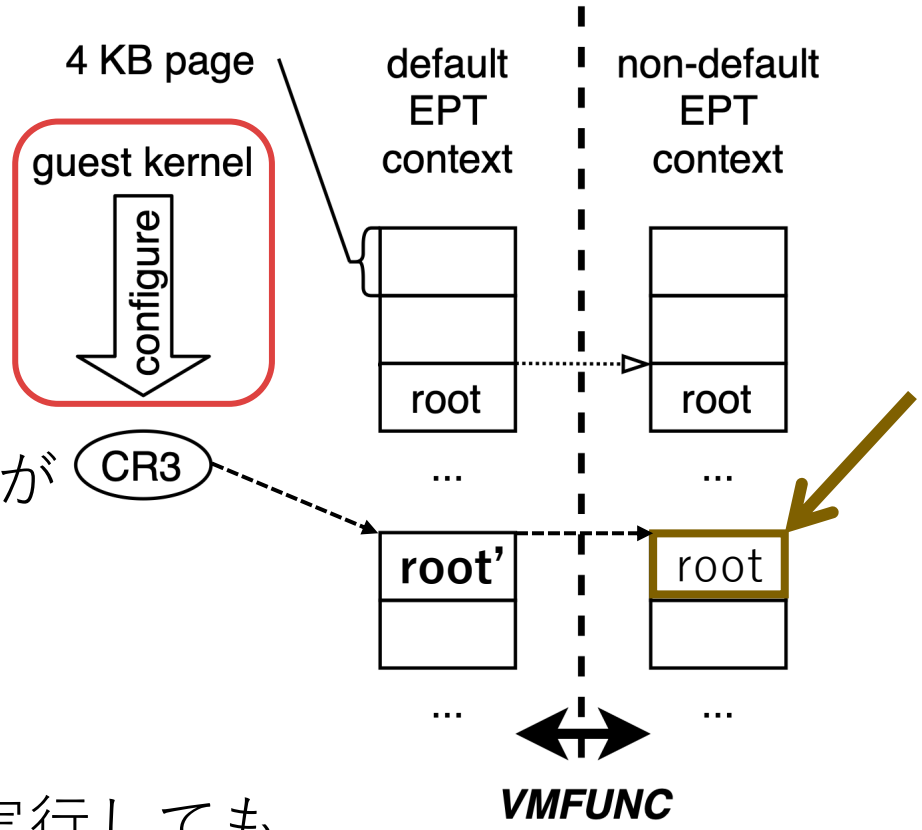
よくある解決策

CR3 アクセスをトラップ

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

ゲスト用ページテーブルの設定



CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行しても
CR3 レジスタの値は変更されない

<u>よくある解決策</u>

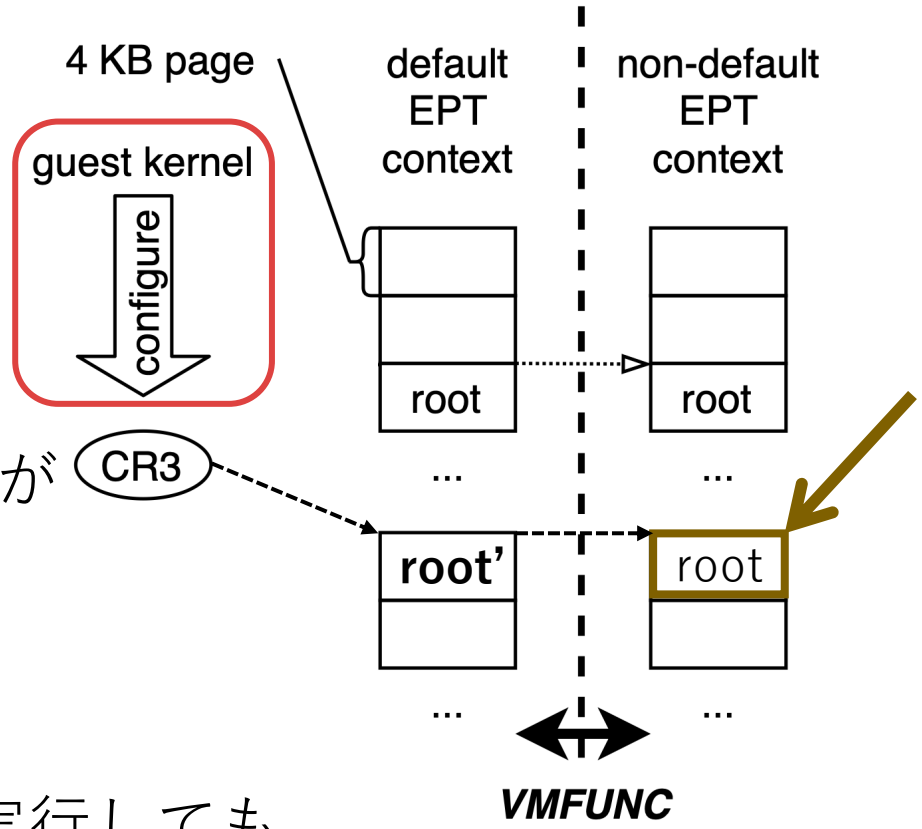
CR3 アクセスをトラップ

新しい参照先に root を用意

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

ホストはここに
ページテーブルのルートを
用意しておく必要がある

ゲスト用ページテーブルの設定



CR3 の値は
ゲストカーネルが
自由に変更する

VMFUNC を実行しても
CR3 レジスタの値は変更されない

よくある解決策

CR3 アクセスを**トラップ**

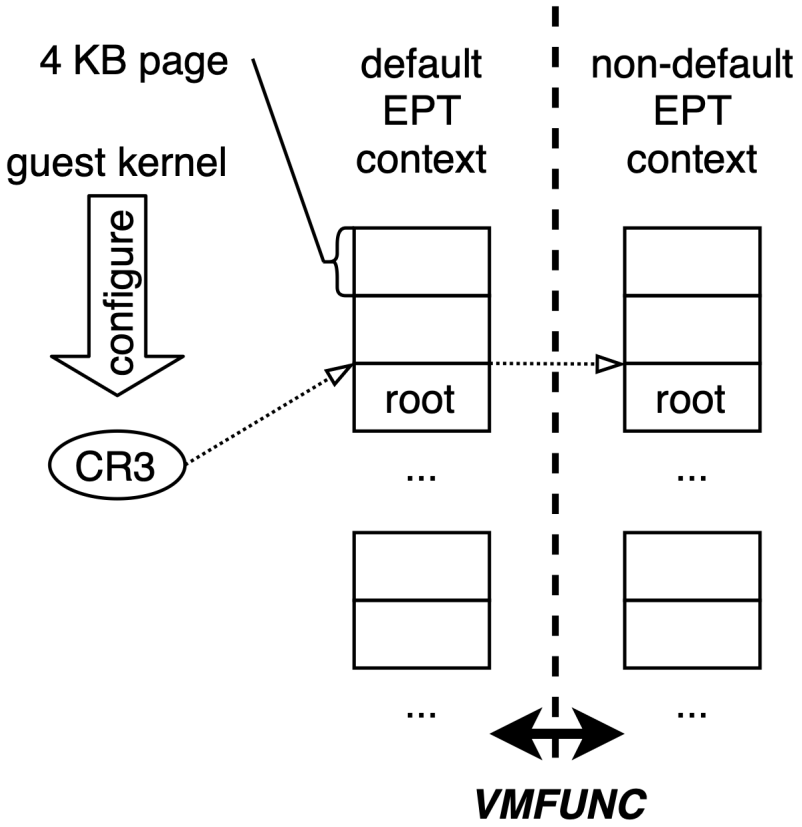
新しい参照先に root を用意

VMFUNC を実行後は
同じ Guest Physical Address 上の
ページがページテーブルの
ルートとして参照される

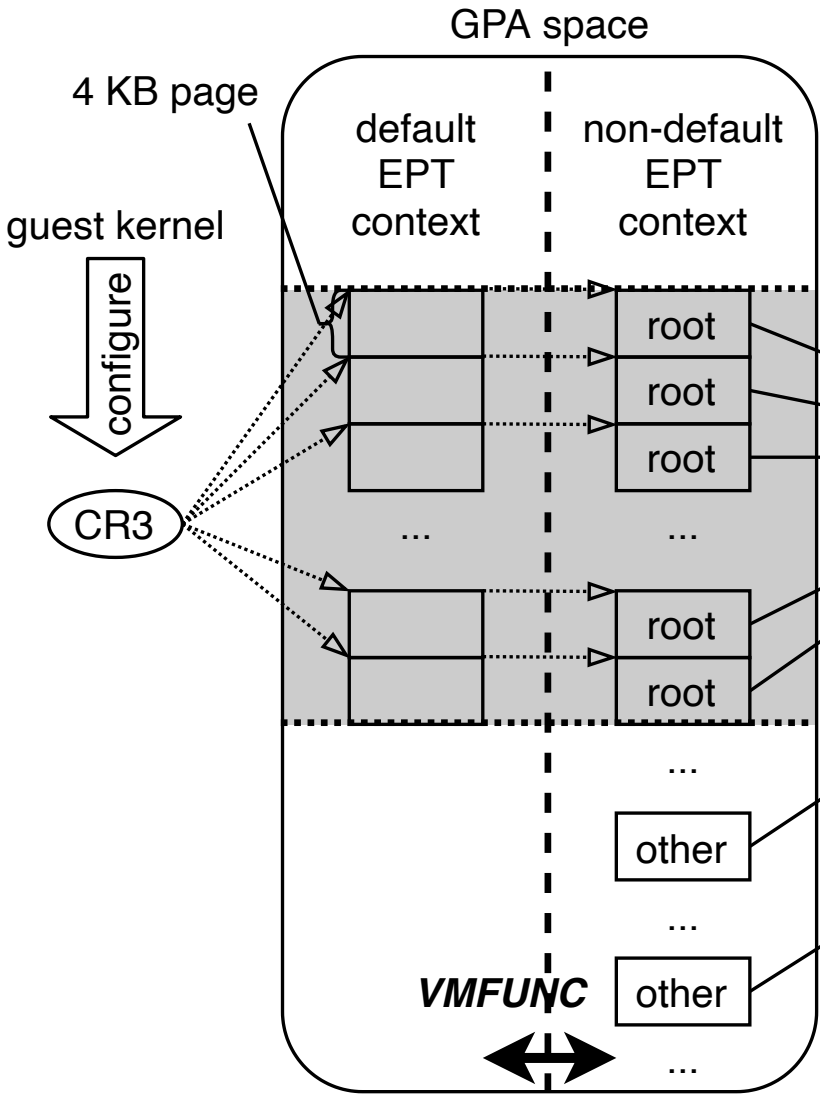
ホストはここに
ページテーブルのルートを
用意しておく必要がある

課題：トラップは VM-exit なのでコストが高い

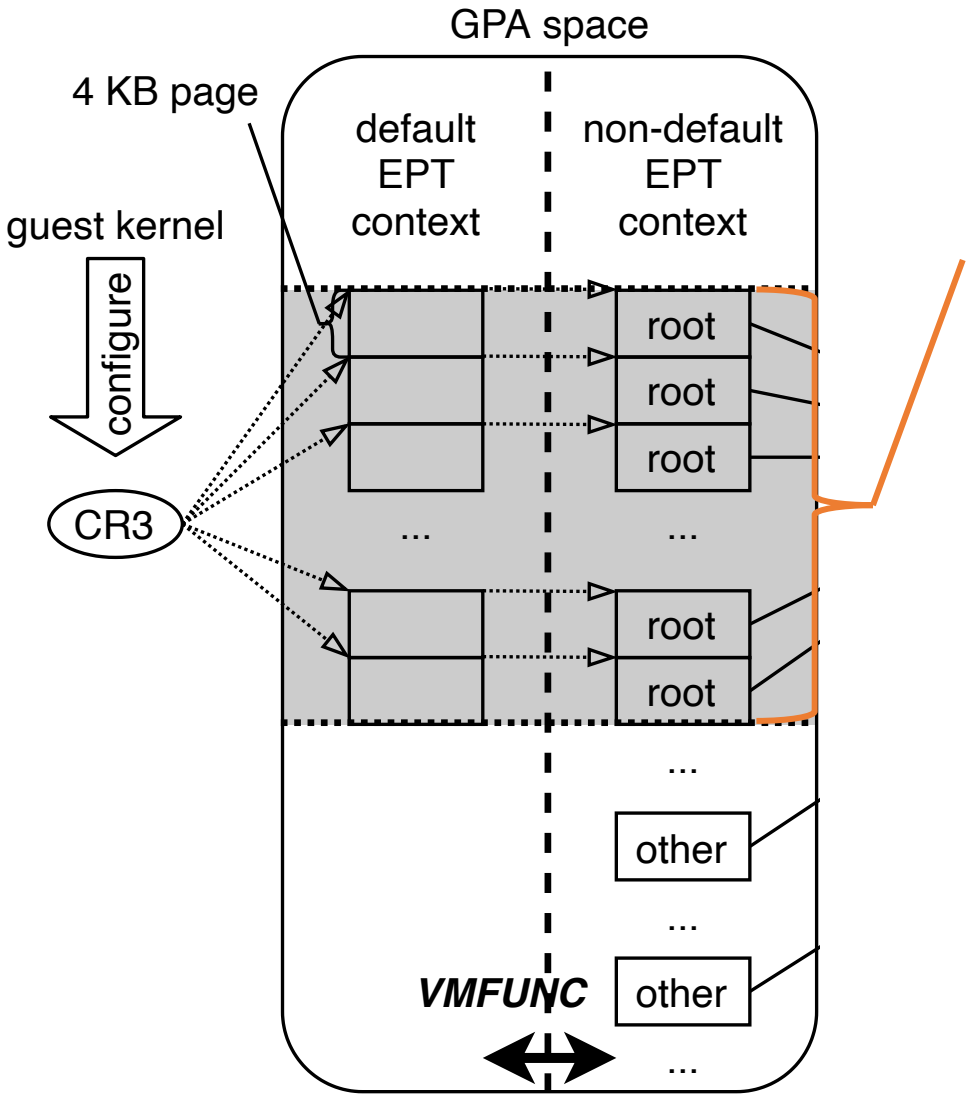
提案手法：Anywhere Page Table (APT)



提案手法：Anywhere Page Table (APT)



提案手法：Anywhere Page Table (APT)



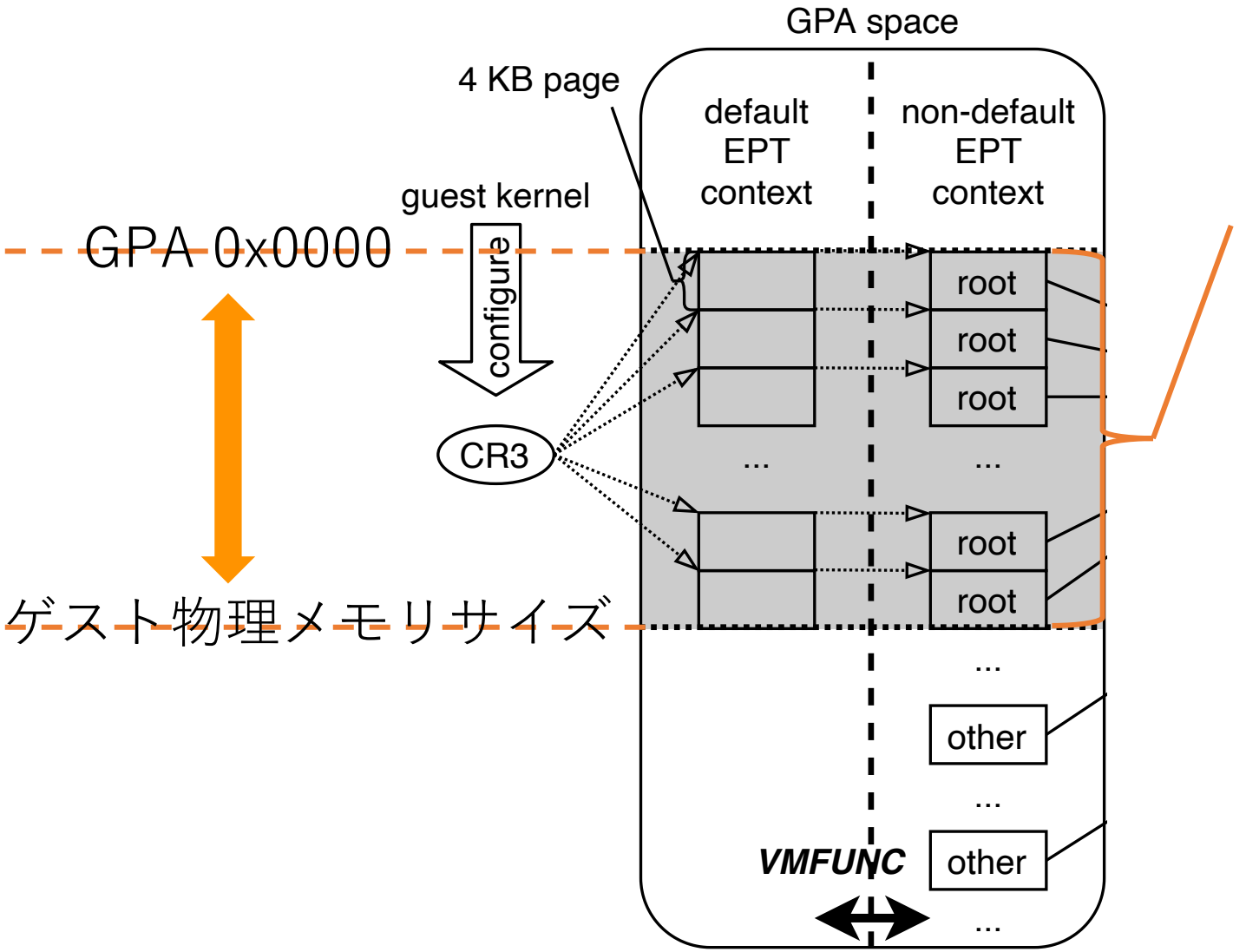
アイデア

ゲストカーネルが

参照可能な GPA 領域全てに

root を配置する

提案手法：Anywhere Page Table (APT)



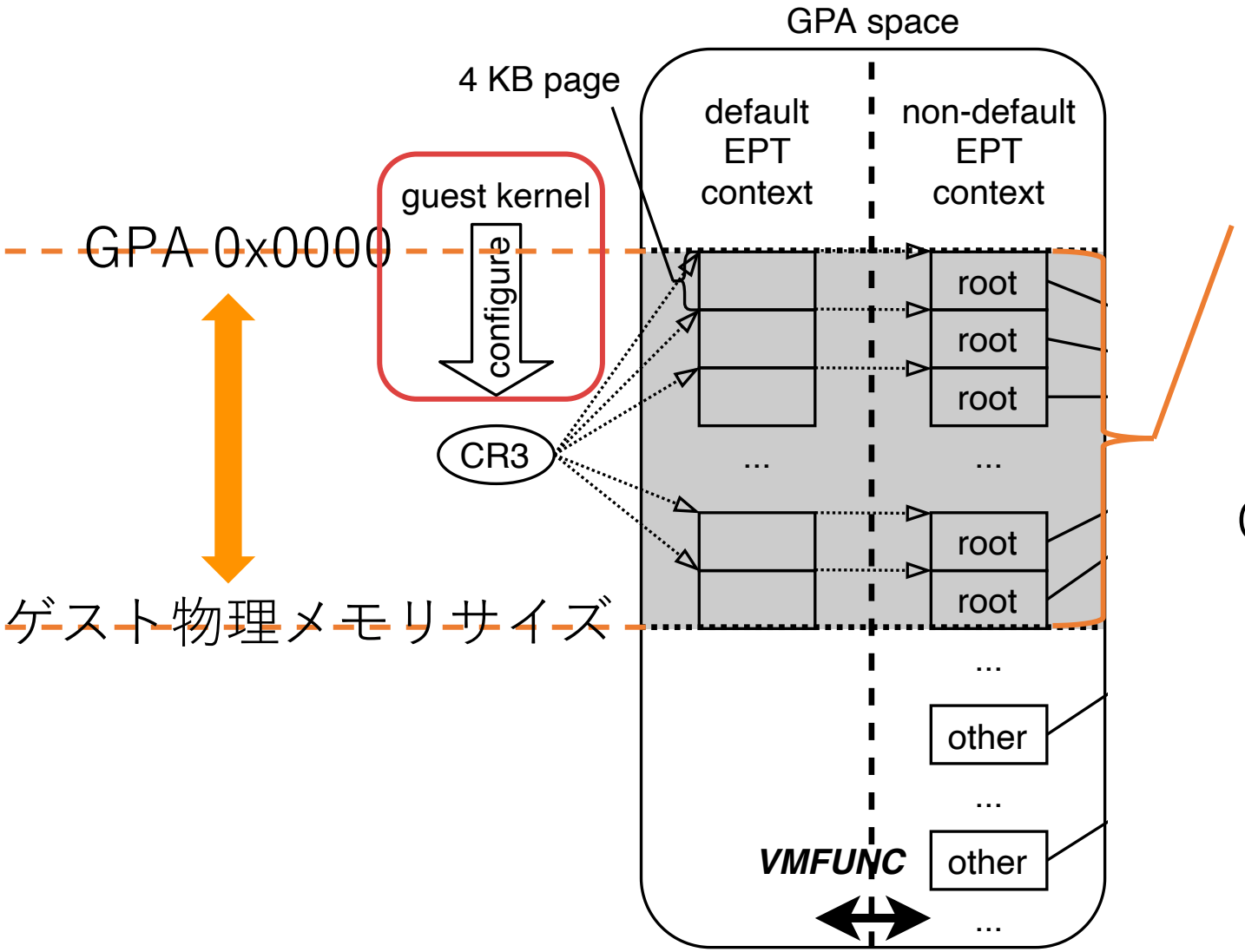
アイデア

ゲストカーネルが

参照可能な GPA 領域全てに

root を配置する

提案手法：Anywhere Page Table (APT)

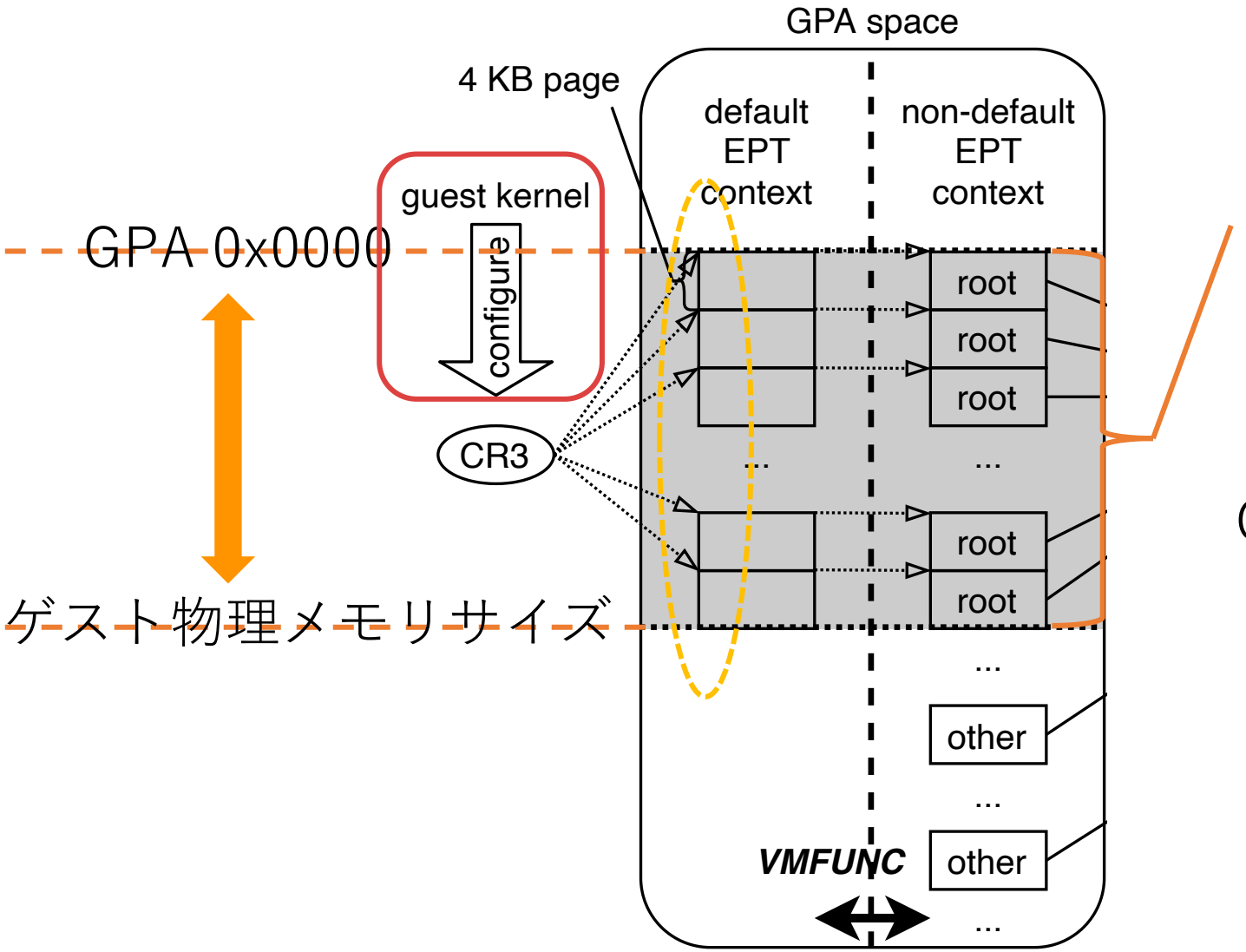


アイデア

ゲストカーネルが
参照可能な GPA 領域全てに
root を配置する

ゲストカーネルが
CR3 がどこを参照するように
設定したとしても
切り替え後に必ず
同じ位置に root がある

提案手法：Anywhere Page Table (APT)

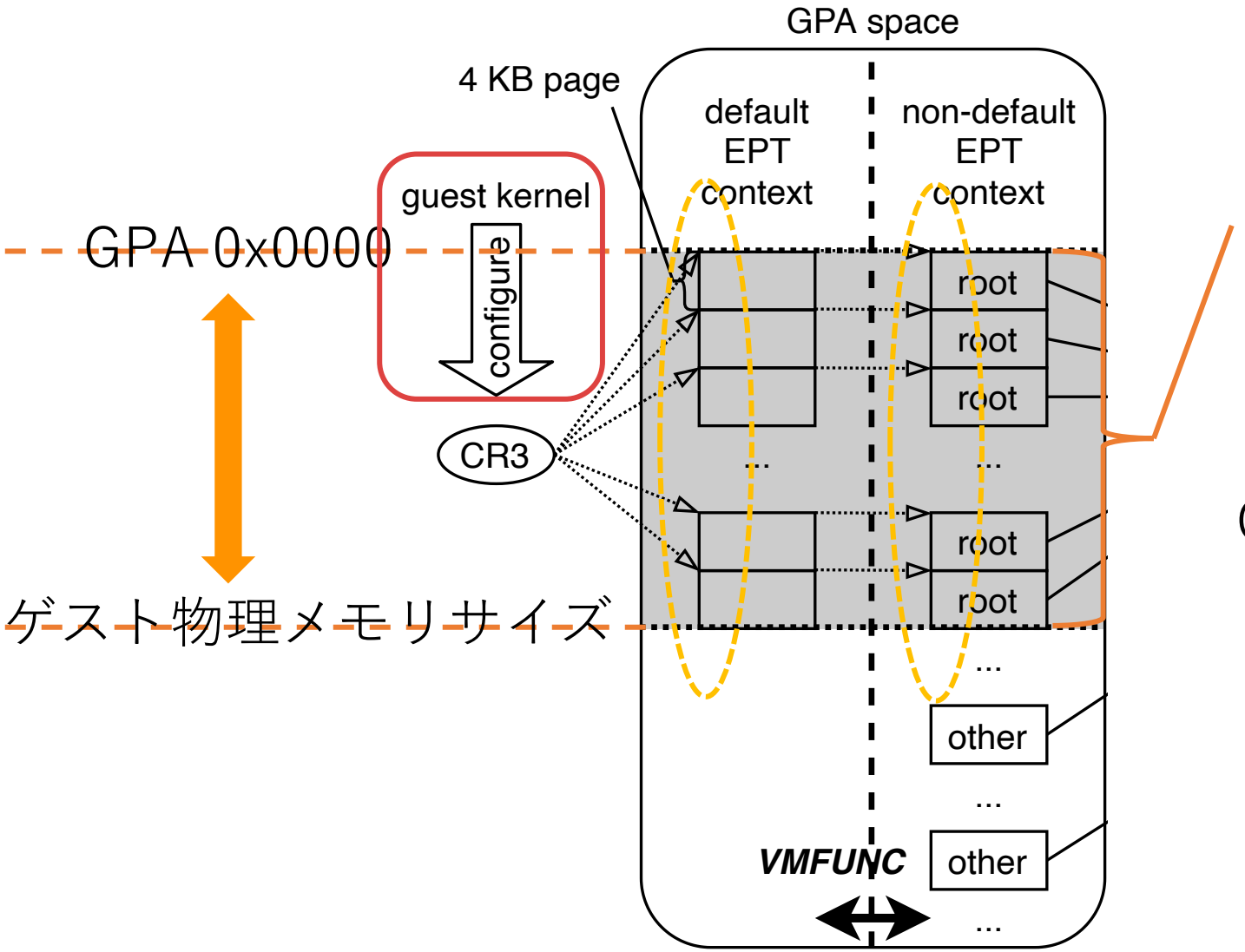


アイデア

ゲストカーネルが
参照可能な GPA 領域全てに
root を配置する

ゲストカーネルが
CR3 がどこを参照するように
設定したとしても
切り替え後に必ず
同じ位置に root がある

提案手法：Anywhere Page Table (APT)

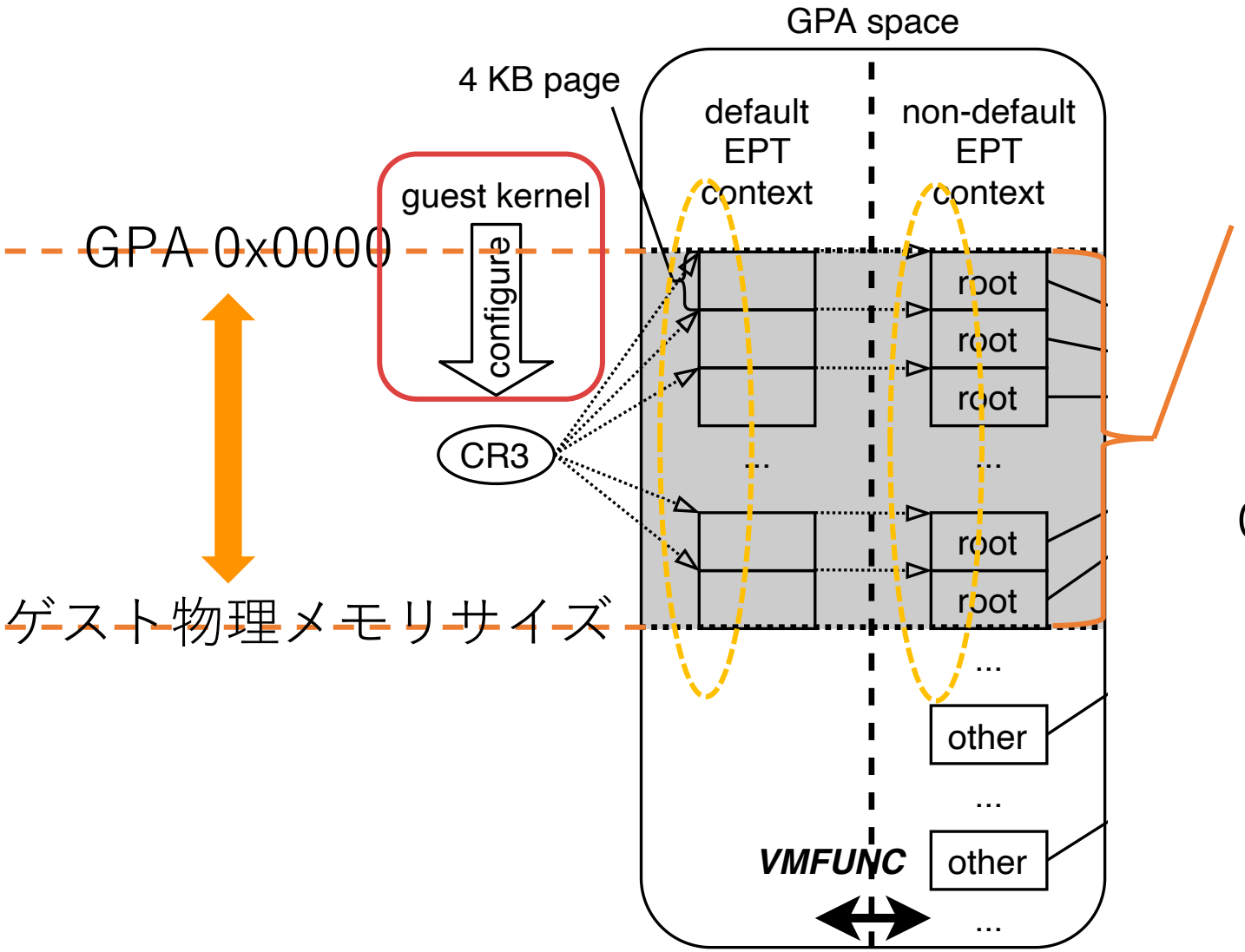


アイデア

ゲストカーネルが
参照可能な GPA 領域全てに
root を配置する

ゲストカーネルが
CR3 がどこを参照するように
設定したとしても
切り替え後に必ず
同じ位置に root がある

提案手法：Anywhere Page Table (APT)



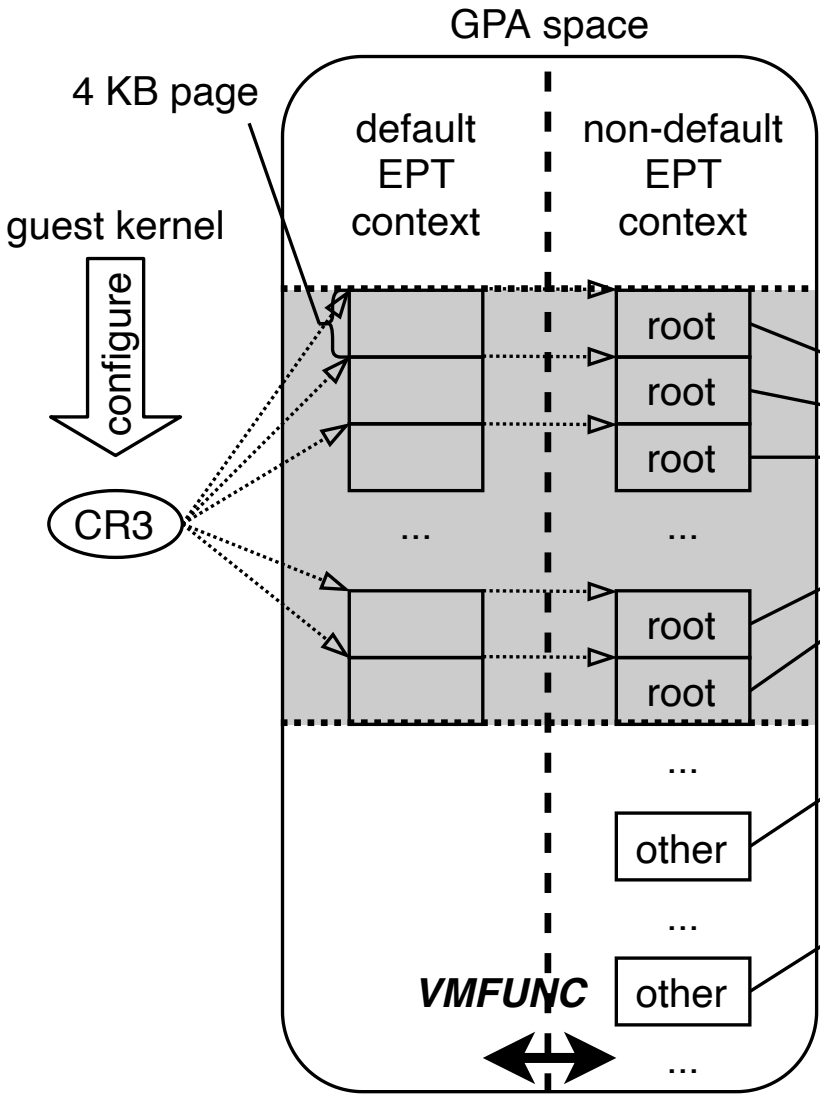
アイデア

ゲストカーネルが
参照可能な GPA 領域全てに
root を配置する

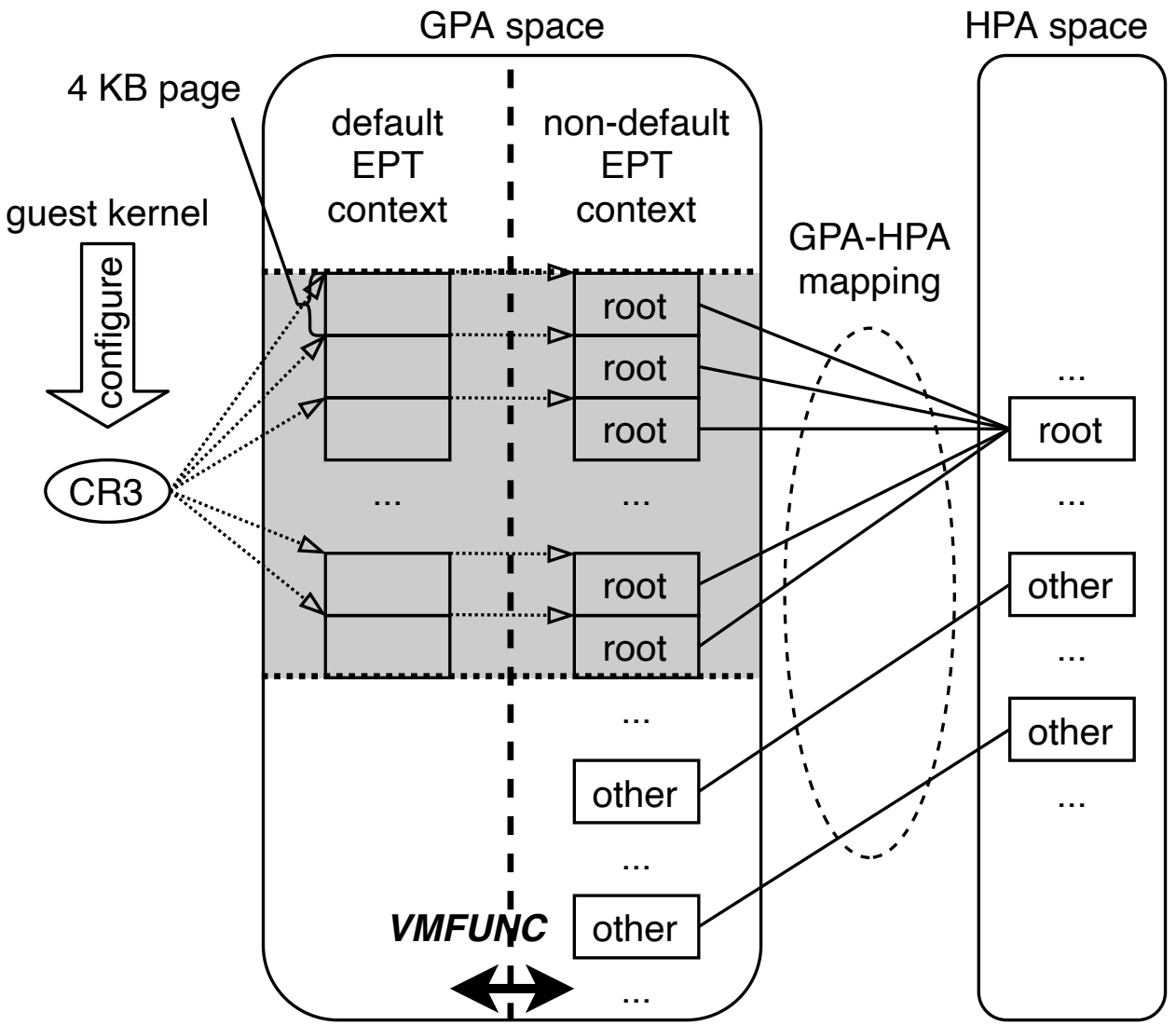
ゲストカーネルが
CR3 がどこを参照するように
設定したとしても
切り替え後に必ず
同じ位置に root がある

Trap による設定が不要

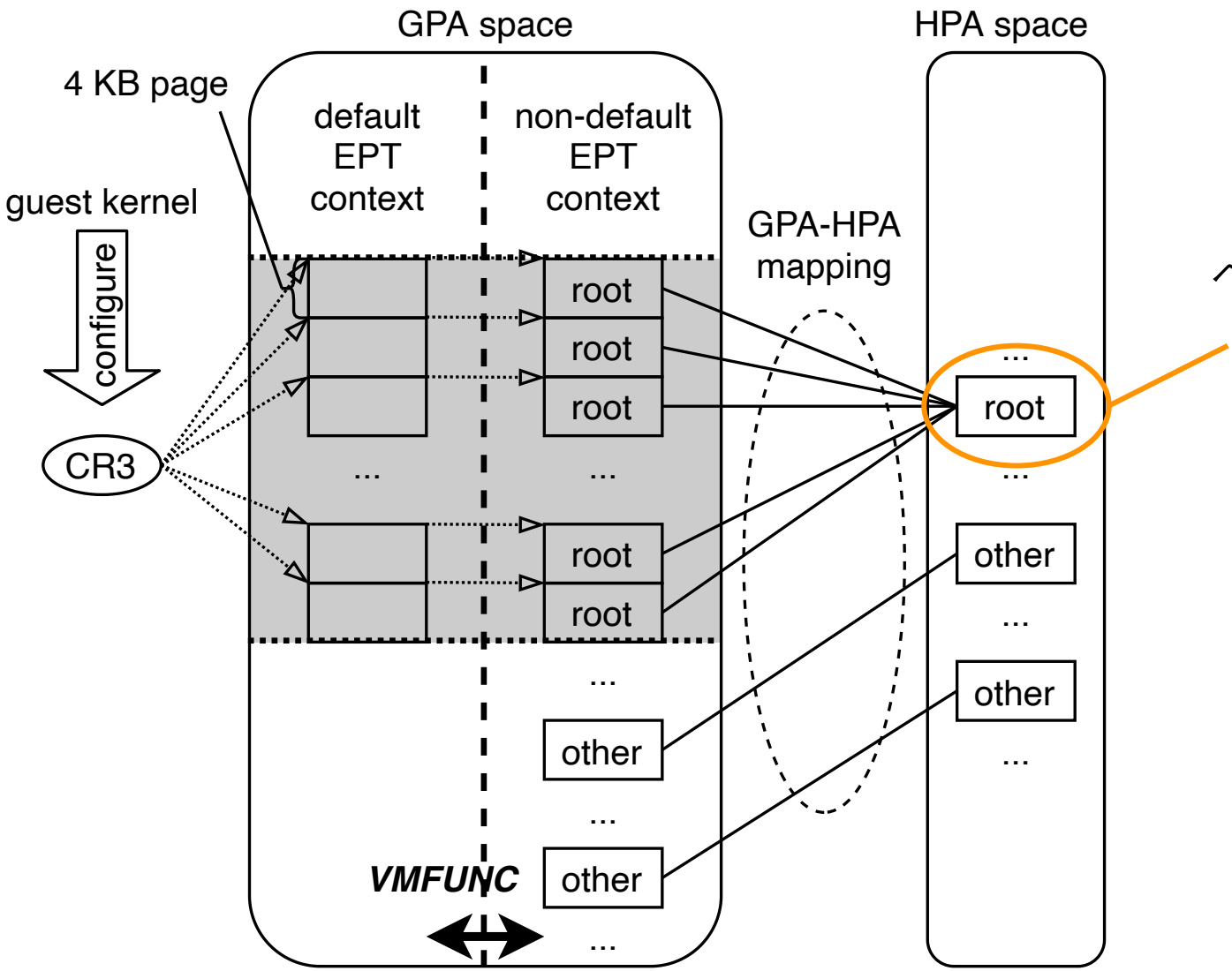
提案手法：Anywhere Page Table (APT)



提案手法：Anywhere Page Table (APT)

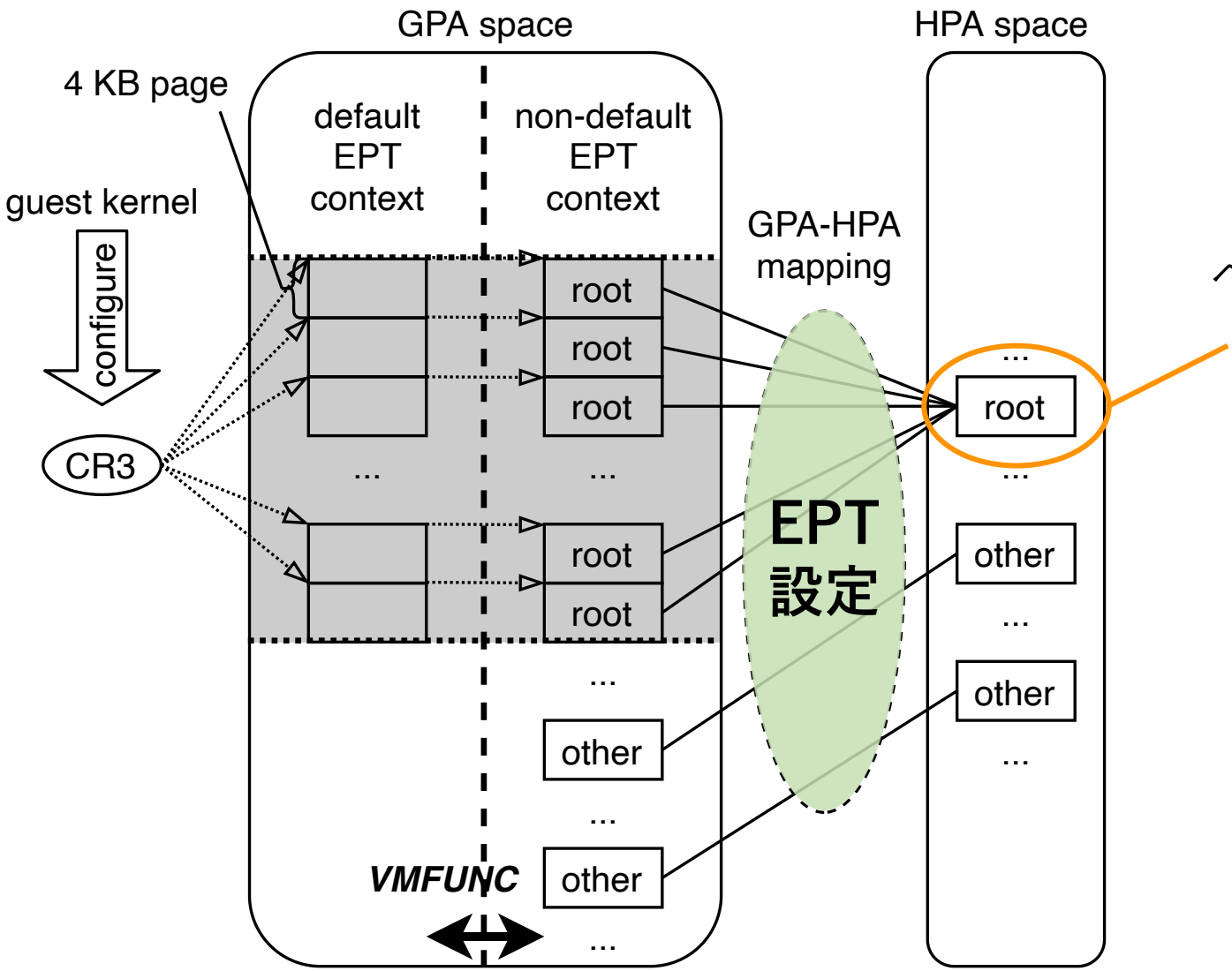


提案手法：Anywhere Page Table (APT)



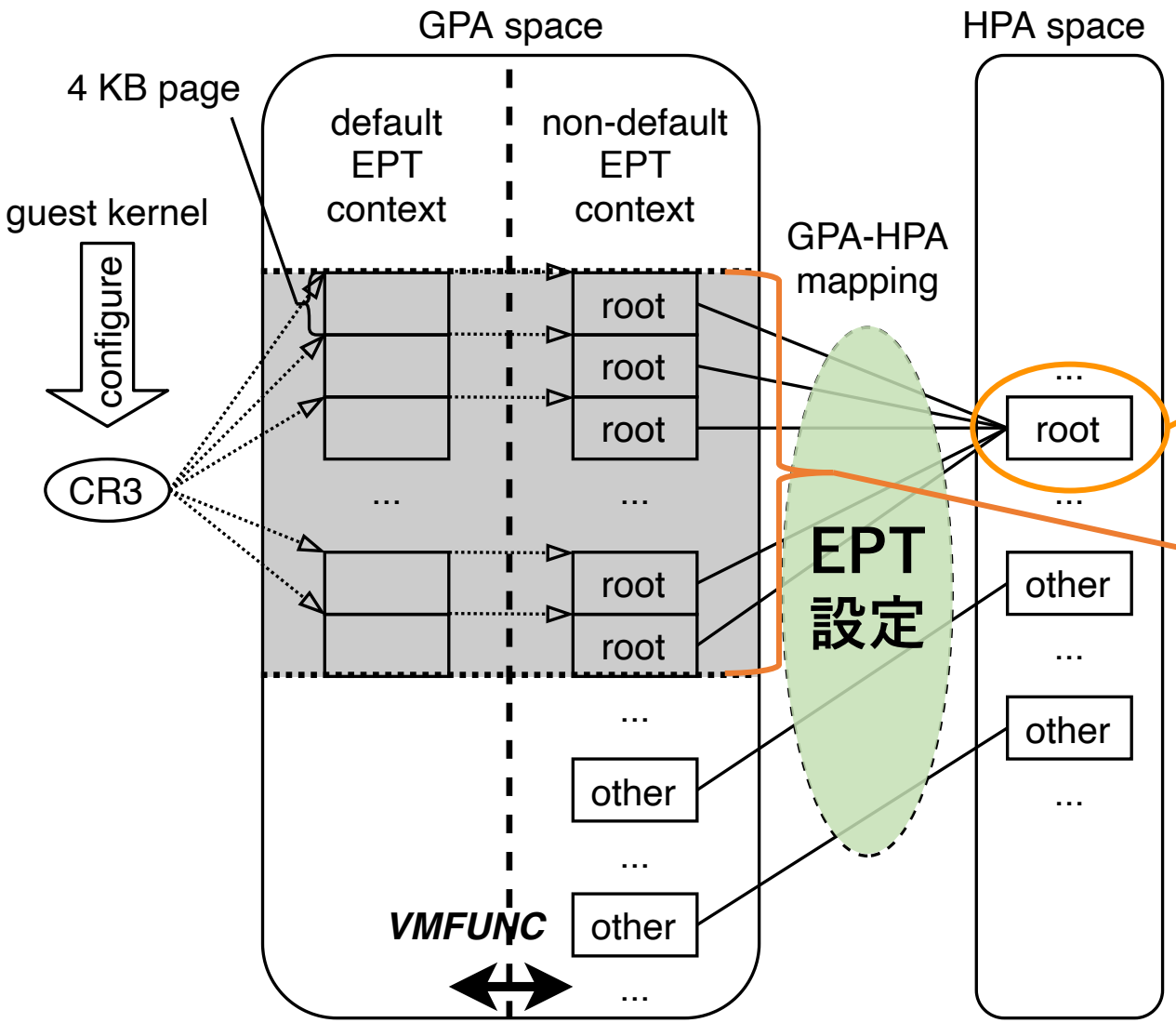
物理的には
ページテーブルは
１つで大丈夫

提案手法：Anywhere Page Table (APT)



物理的には
ページテーブルは
１つで大丈夫

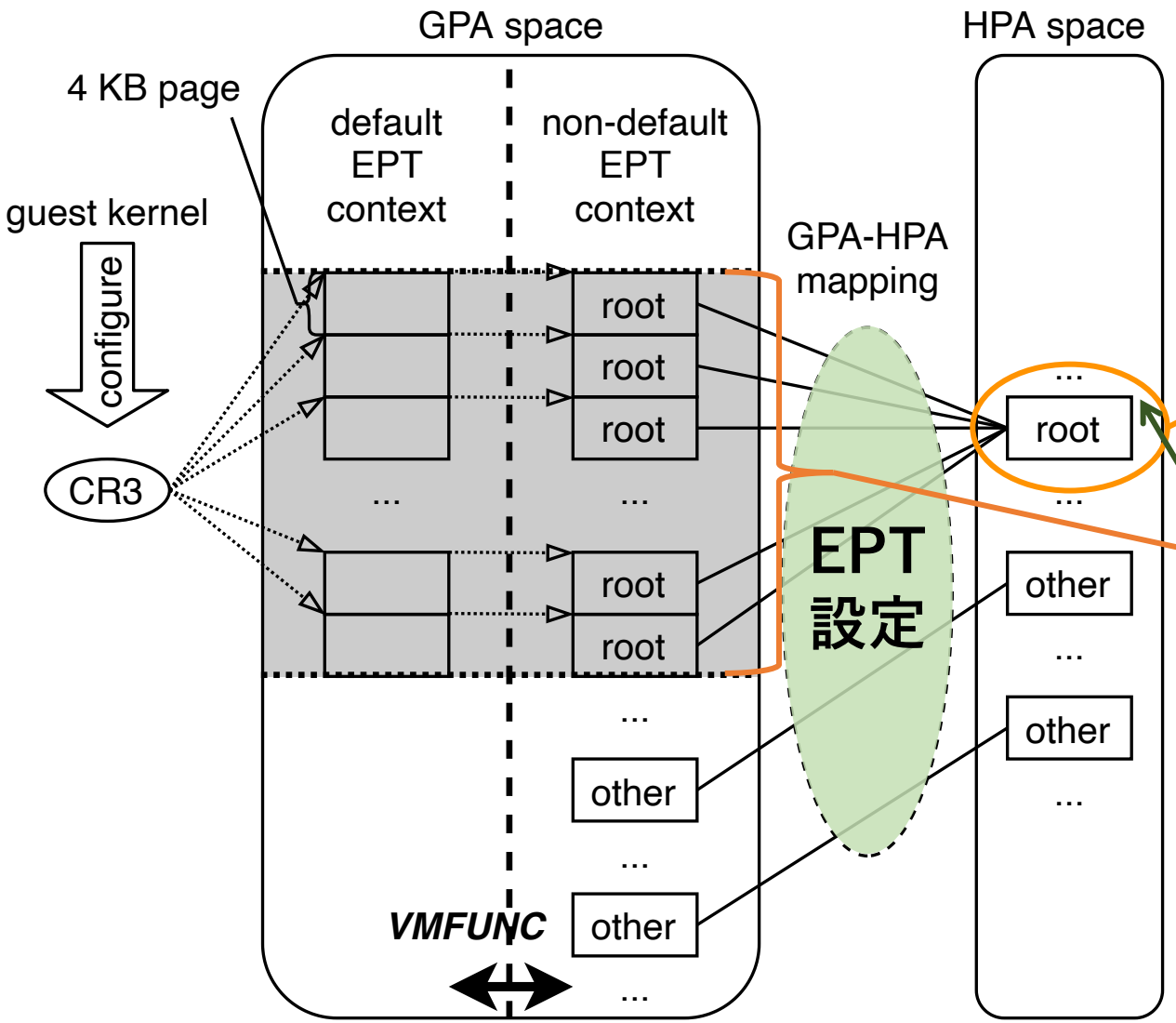
提案手法：Anywhere Page Table (APT)



物理的には
ページテーブルは
１つで大丈夫

EPT の設定で
この GPA 領域のページ
全てが、
root が配置されている
Host Physical Address
を 参照するように設定

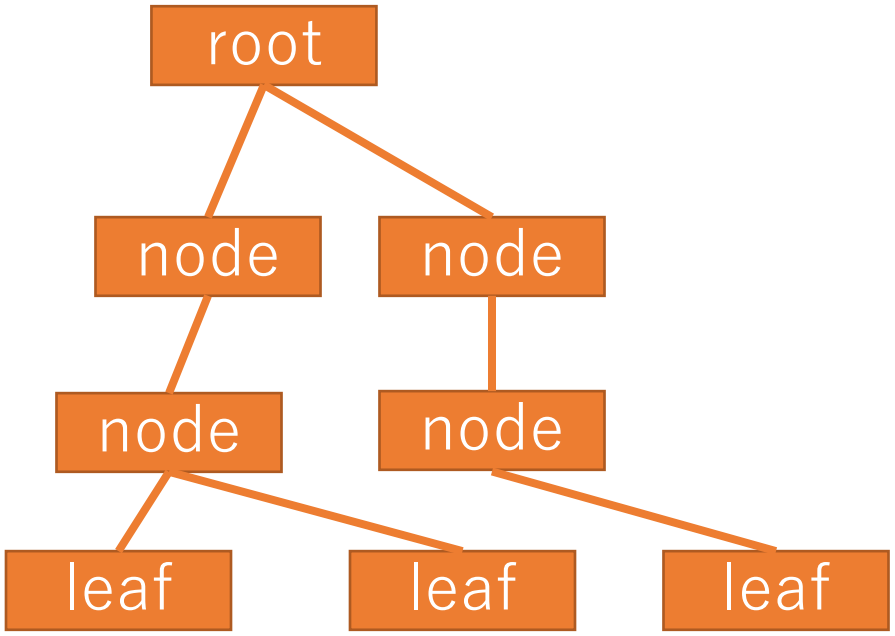
提案手法：Anywhere Page Table (APT)



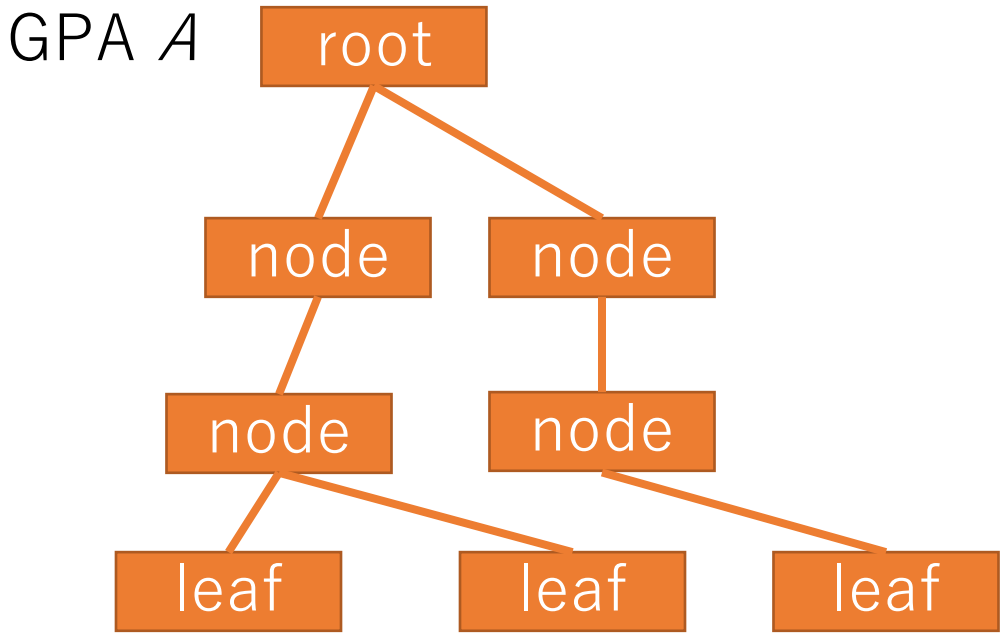
物理的には
ページテーブルは
１つで大丈夫

EPT の設定で
この GPA 領域のページ
全てが、
root が配置されている
Host Physical Address
を 参照するように設定

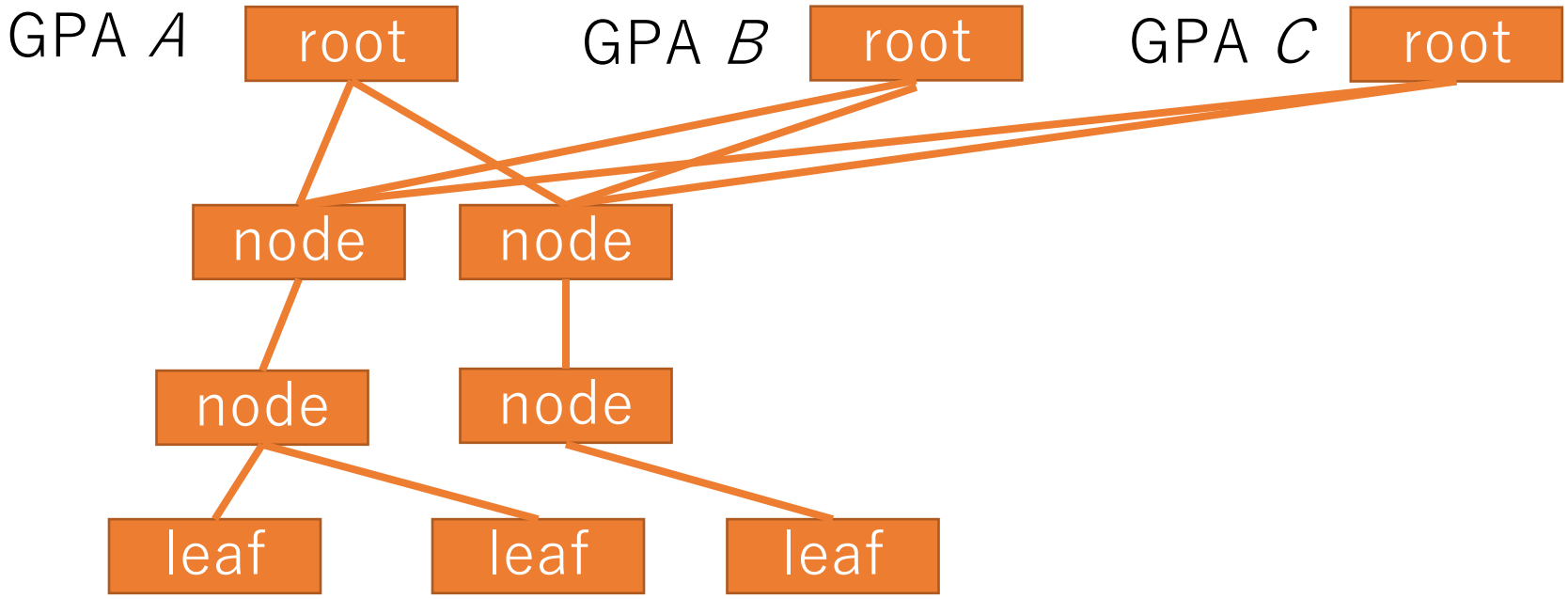
提案手法：Anywhere Page Table (APT)



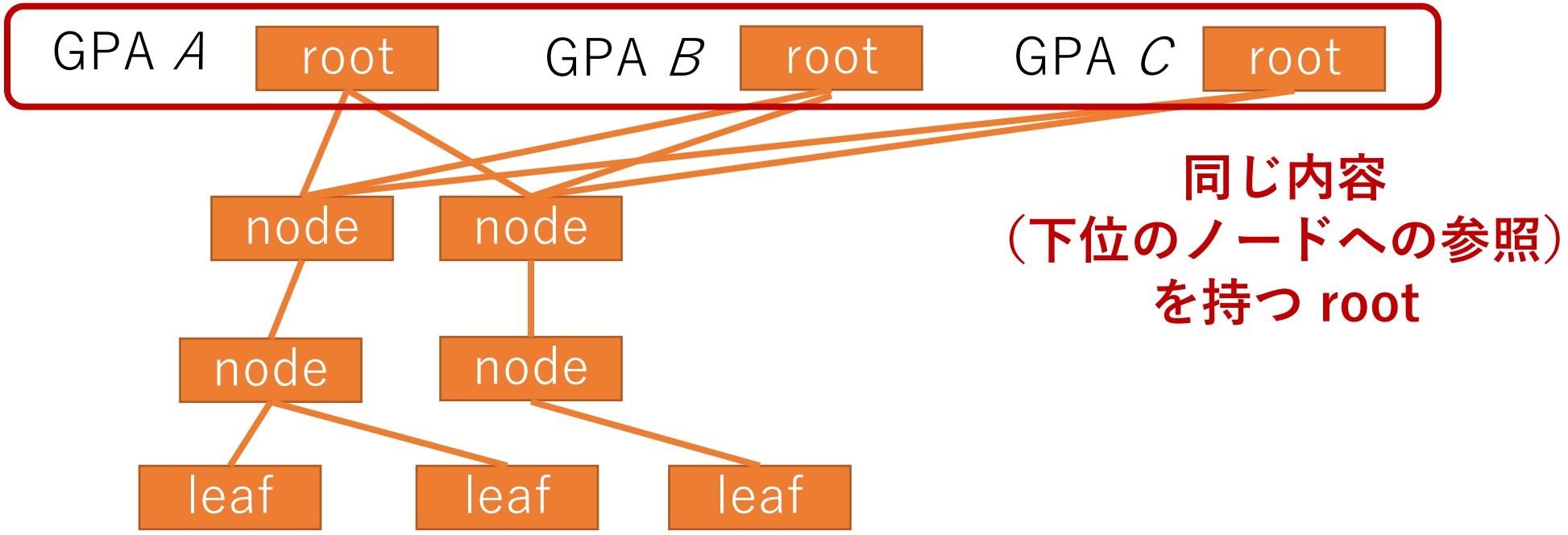
提案手法：Anywhere Page Table (APT)



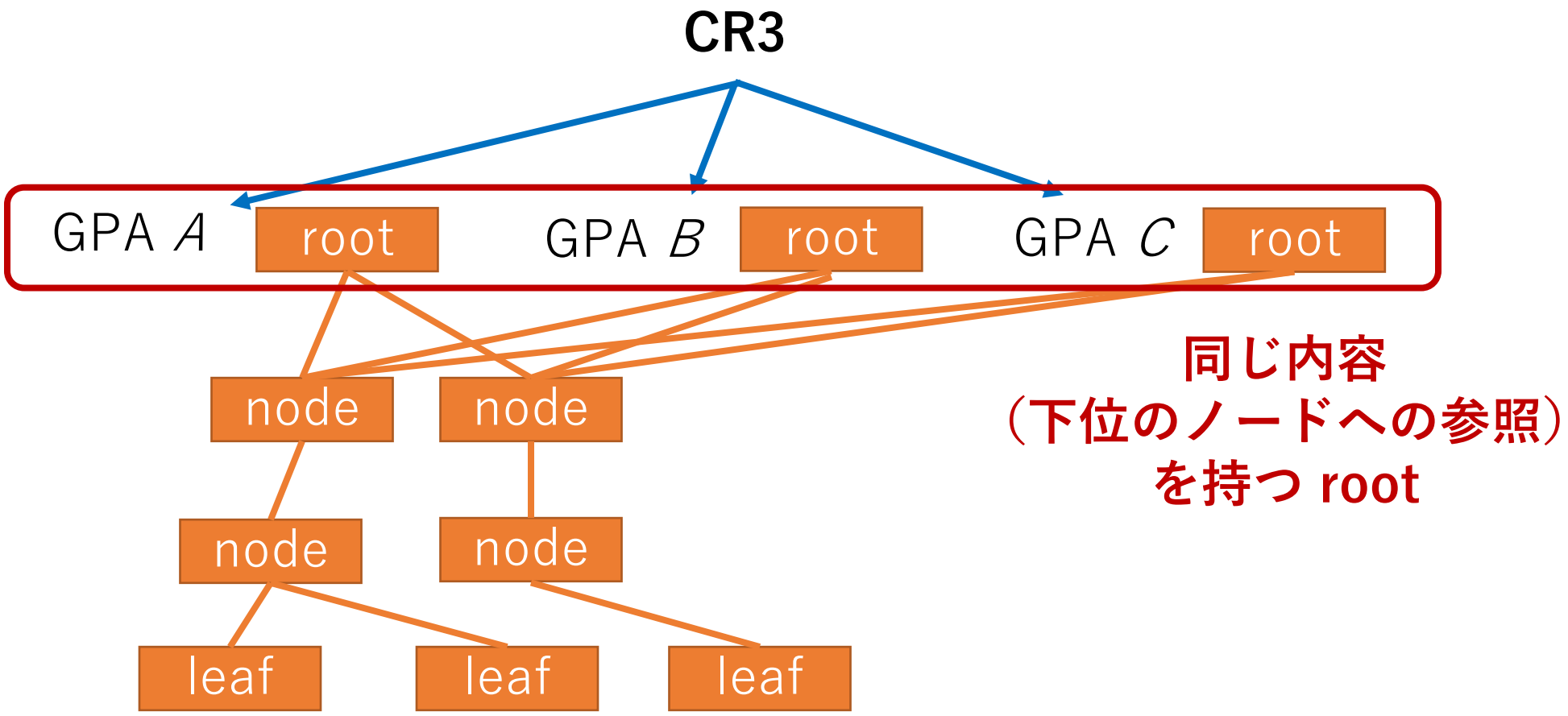
提案手法：Anywhere Page Table (APT)



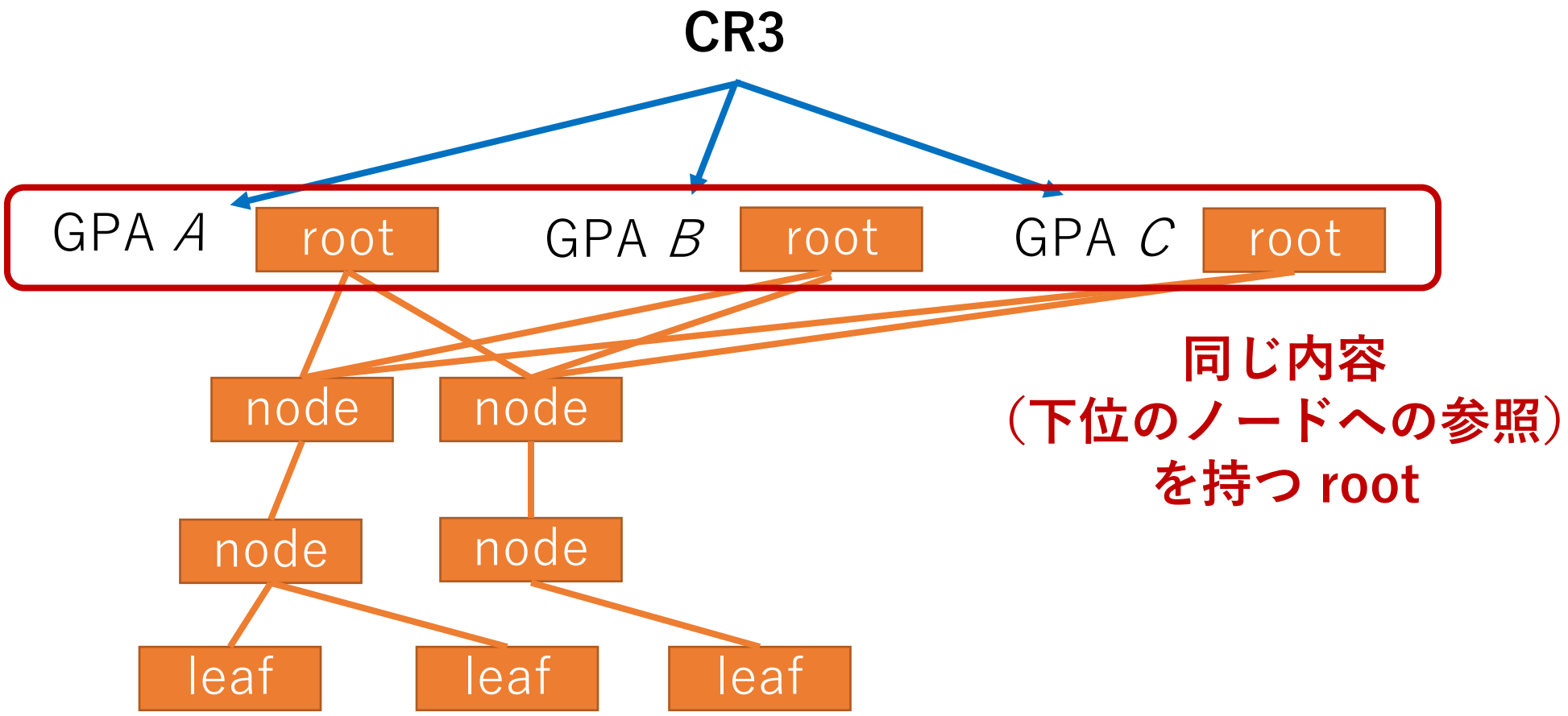
提案手法：Anywhere Page Table (APT)



提案手法：Anywhere Page Table (APT)

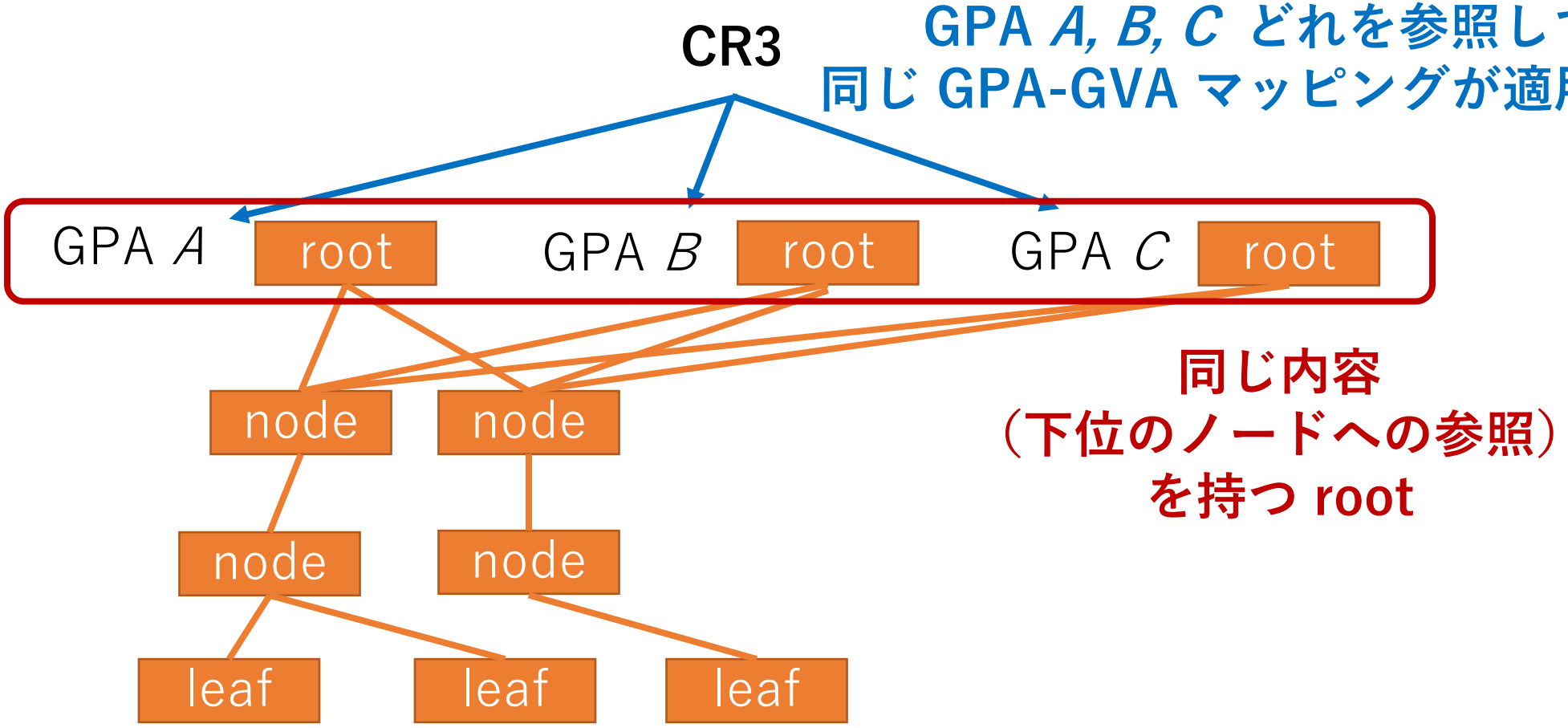


提案手法：Anywhere Page Table (APT)

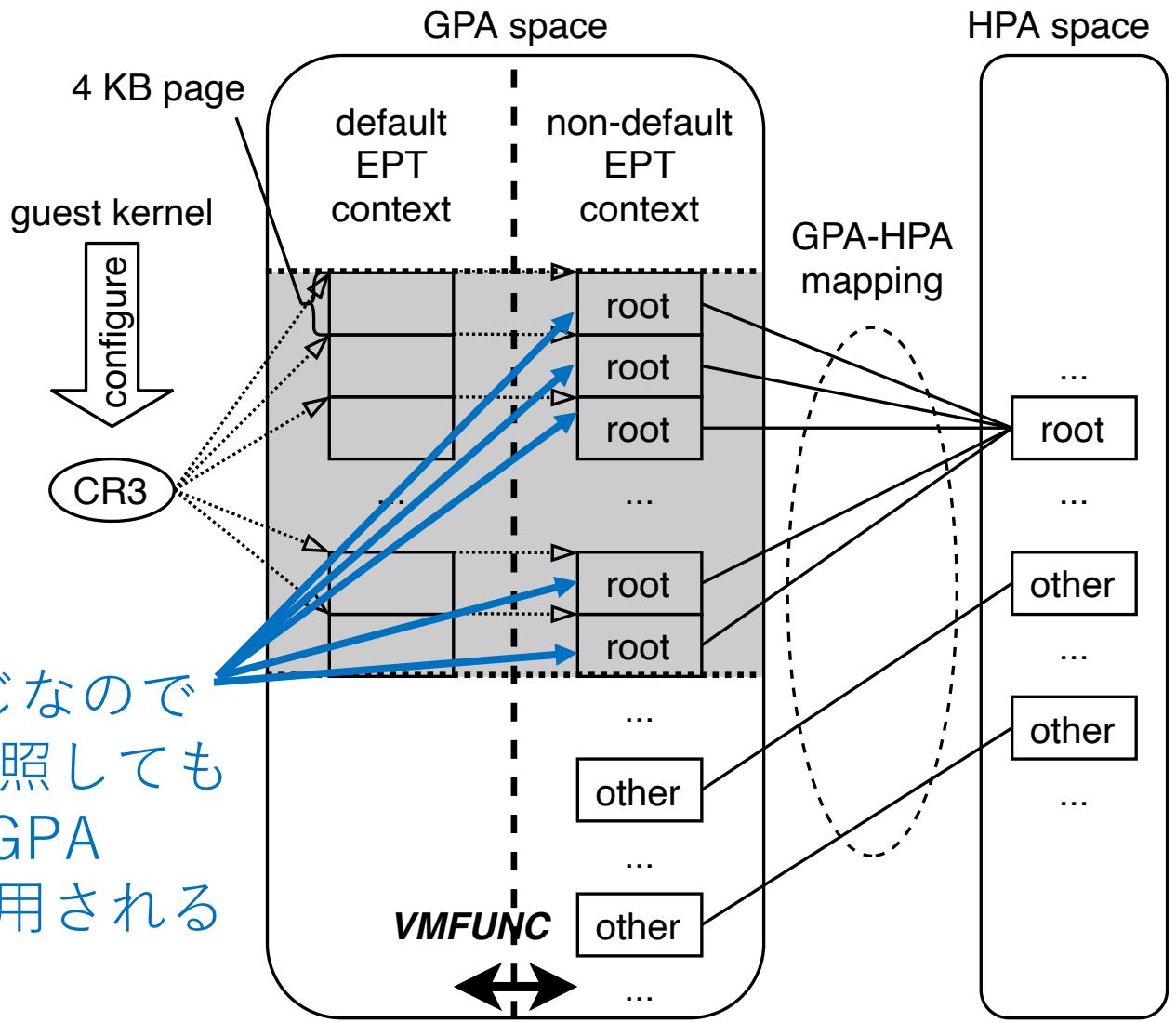


どの位置の root から辿っても見える leaf が同じ

提案手法：Anywhere Page Table (APT)

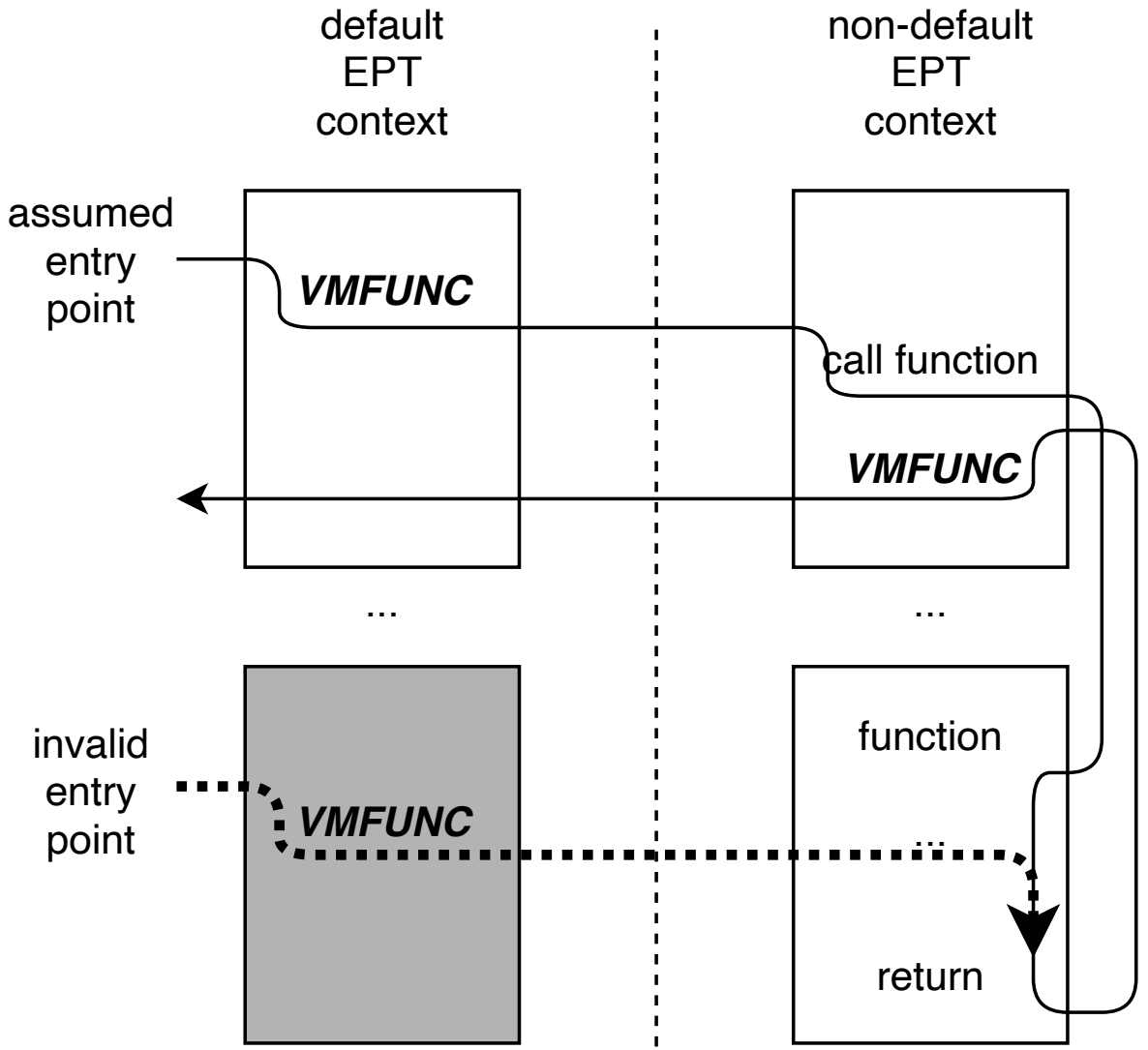


提案手法：Anywhere Page Table (APT)

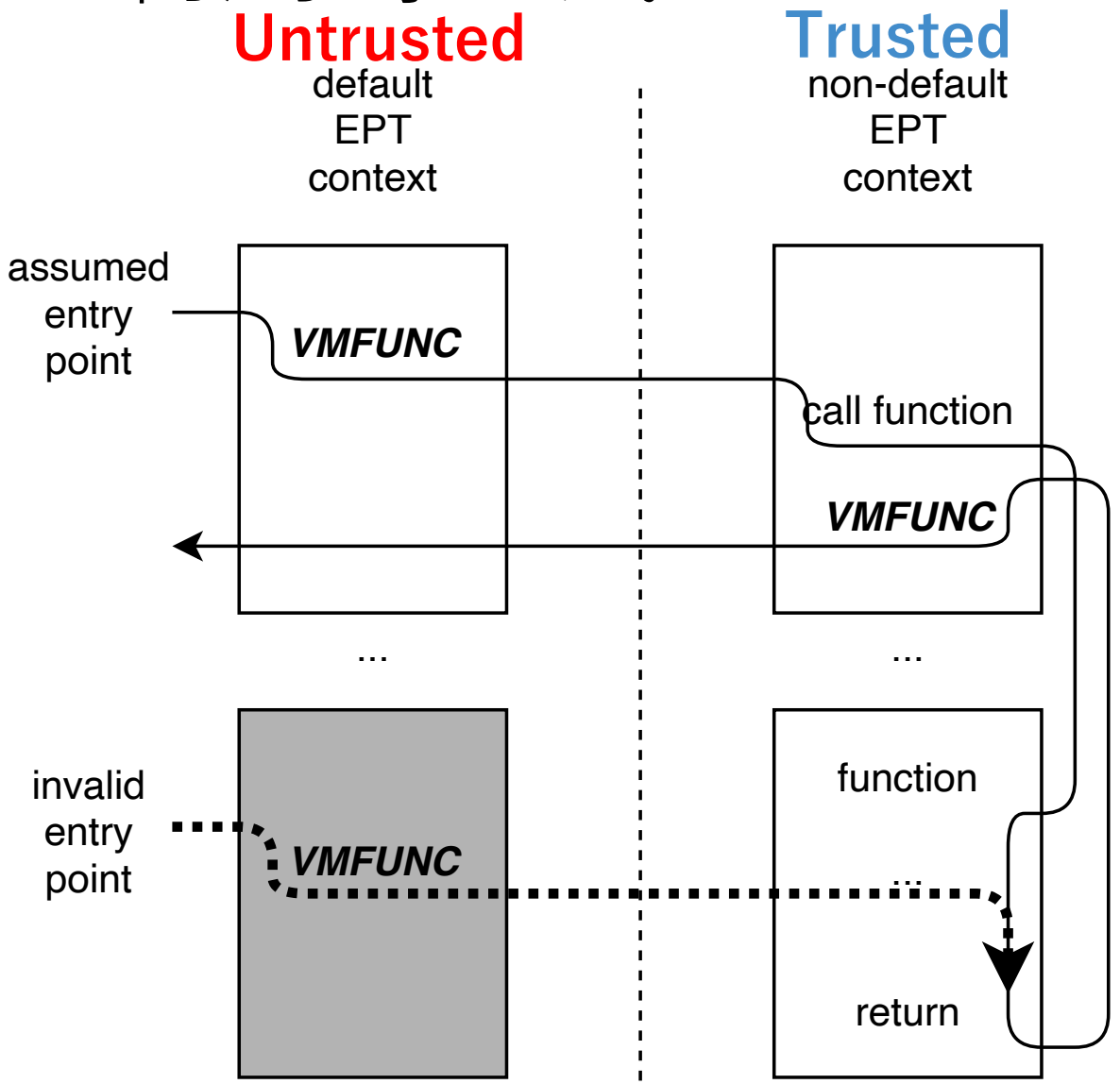


内容が全て同じなので
CR3 がどこを参照しても
同じ GVA-GPA
マッピングが適用される

VMFUNC 利用時に気をつけるべき攻撃

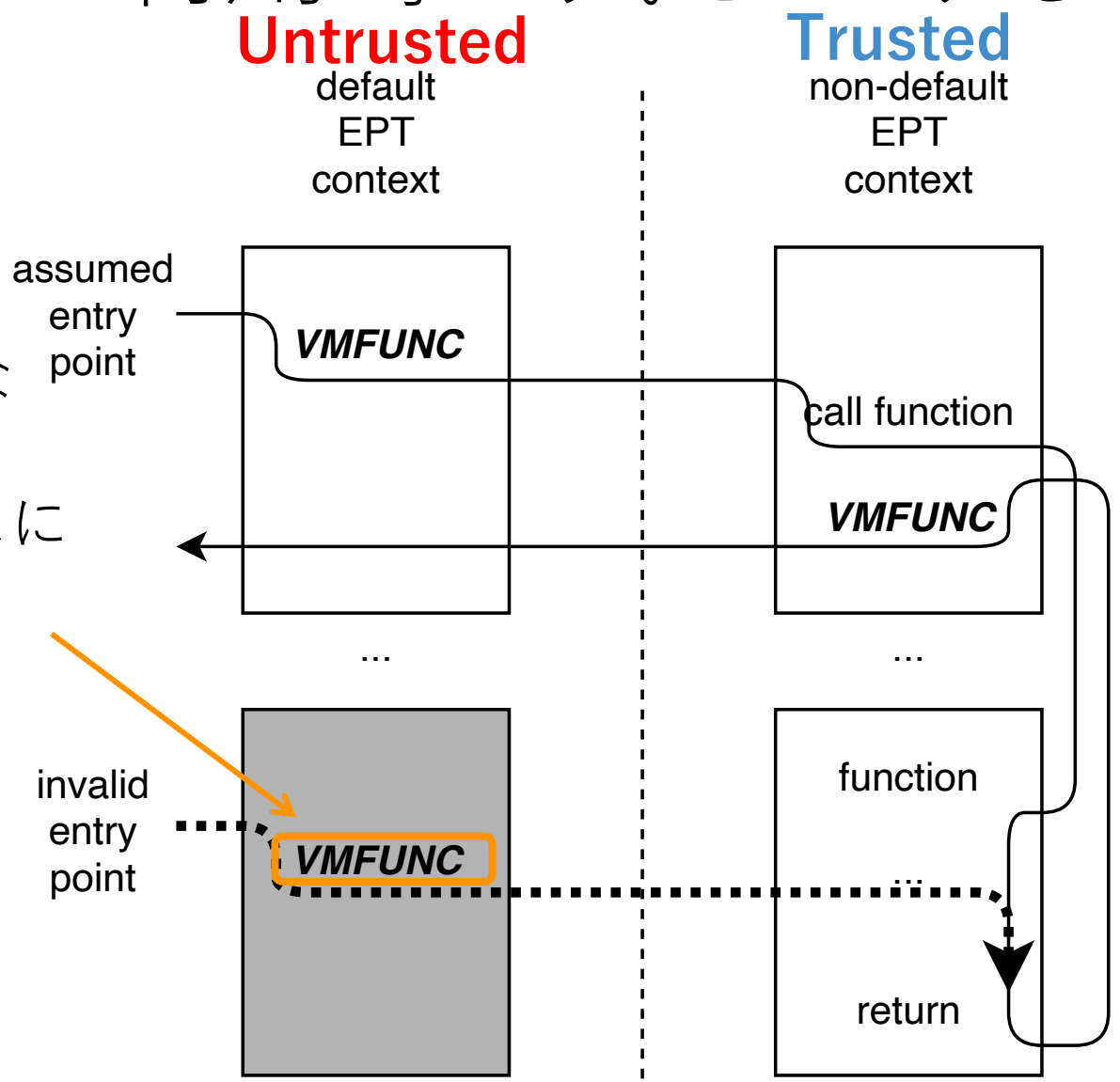


VMFUNC 利用時に気をつけるべき攻撃



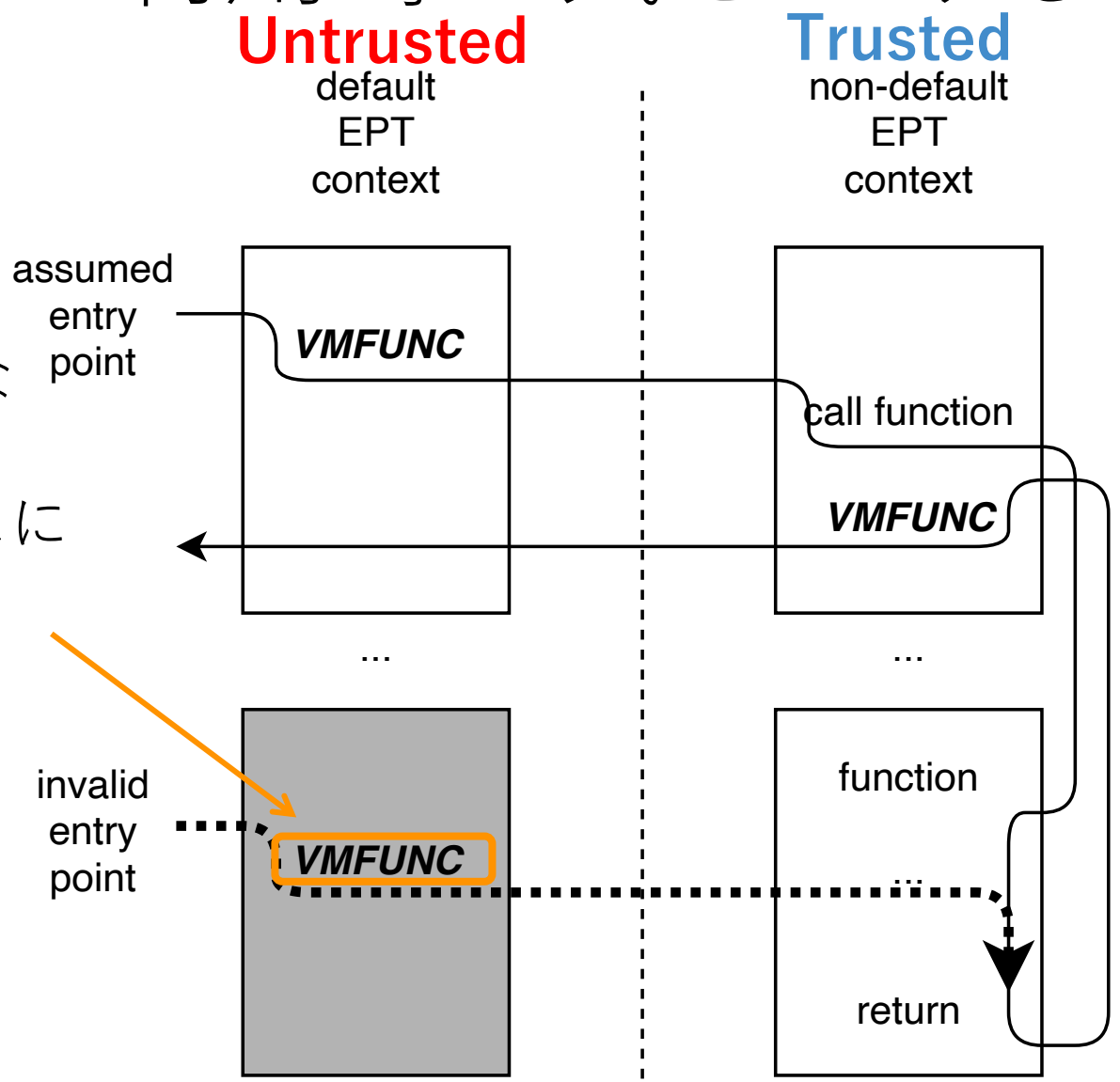
VMFUNC 利用時に気をつけるべき攻撃

CR3 を操作可能な
ゲストは好きな
仮想メモリアドレスに
VMFUNC 命令を
配置し実行できる



VMFUNC 利用時に気をつけるべき攻撃

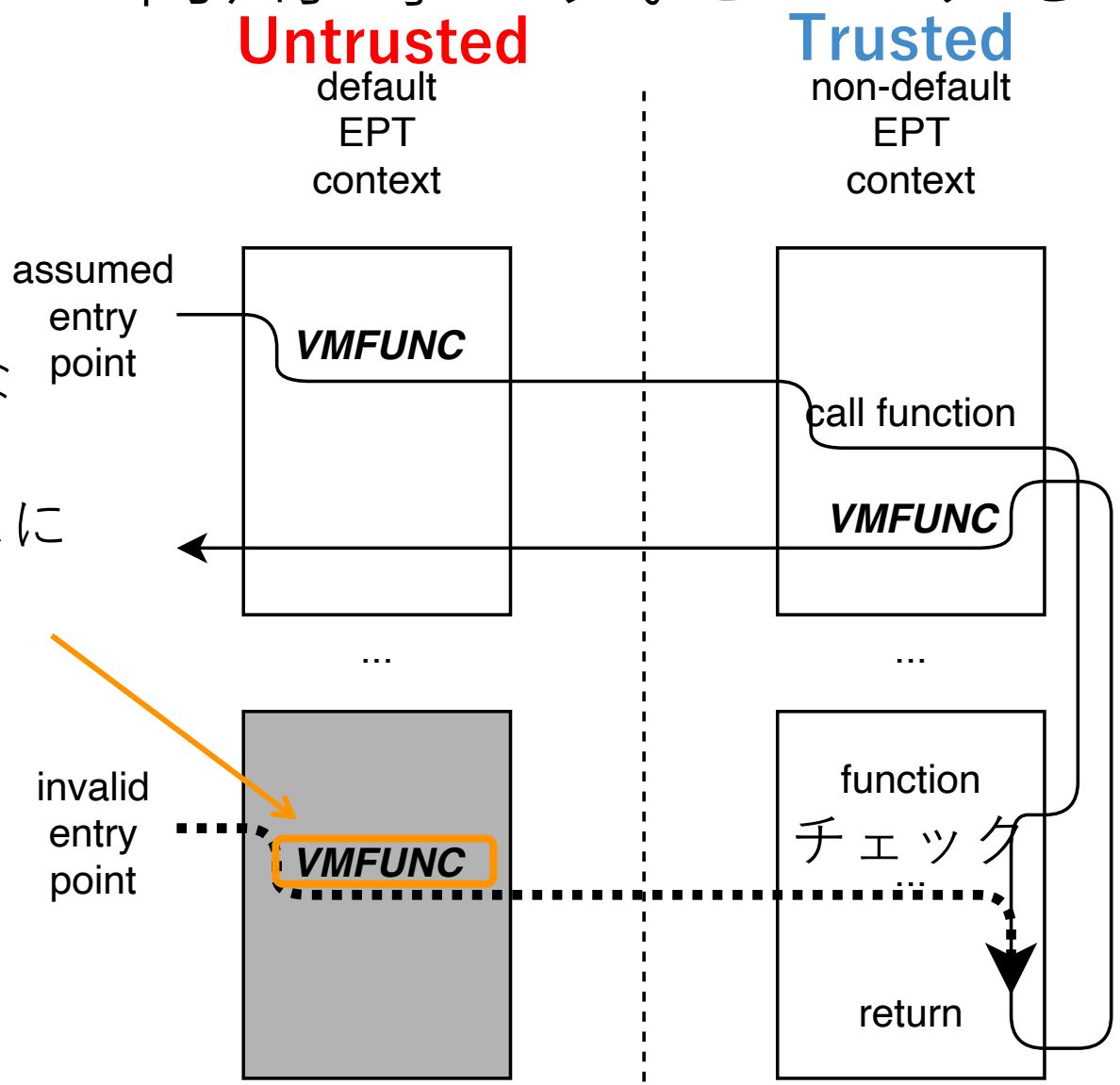
CR3 を操作可能なゲストは好きな仮想メモリアドレスに VMFUNC 命令を配置し実行できる



VMFUNC はプログラムカウンタの値を変更しない

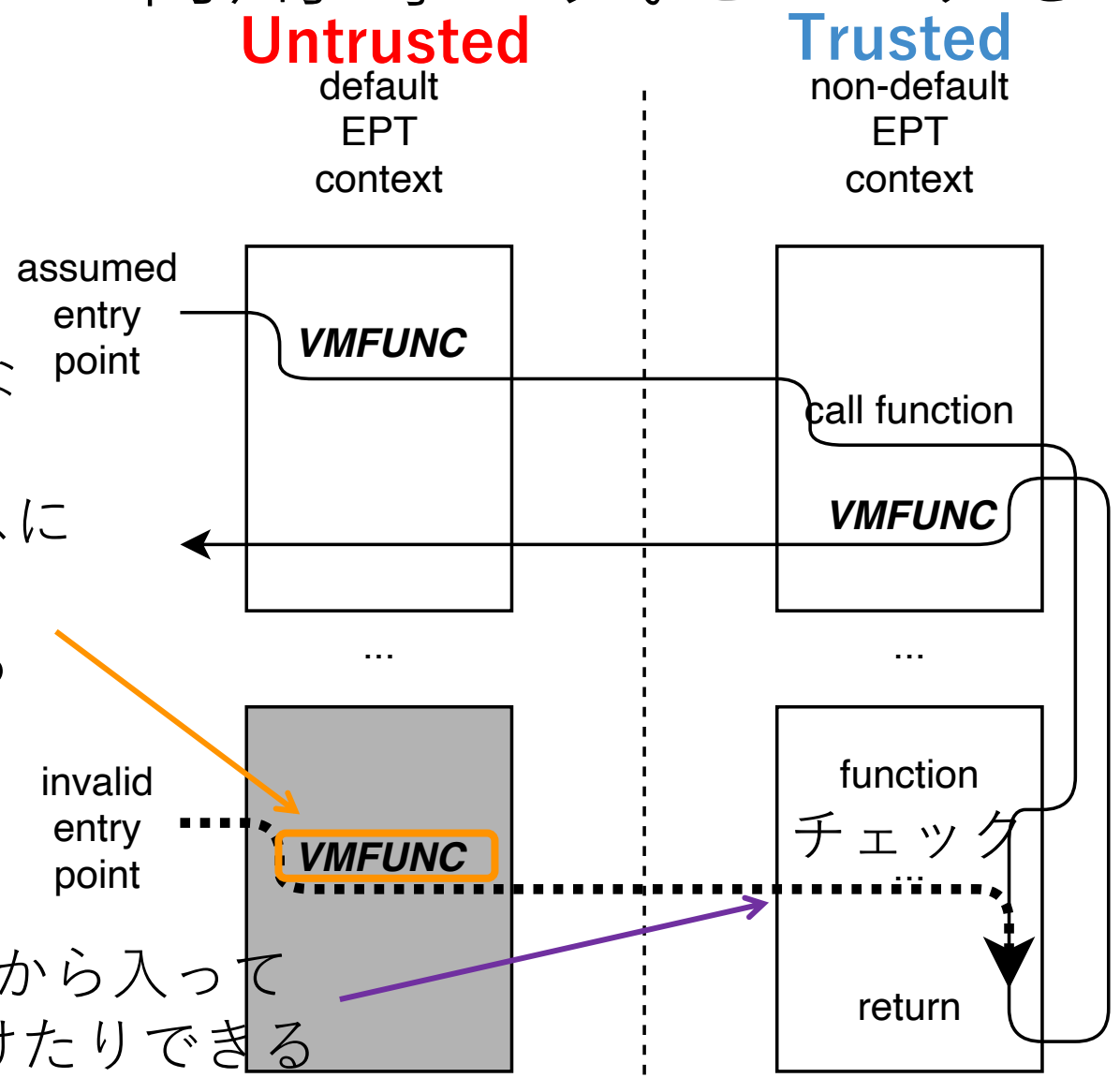
VMFUNC 利用時に気をつけるべき攻撃

CR3 を操作可能なゲストは好きな仮想メモリアドレスに VMFUNC 命令を配置し実行できる



VMFUNC はプログラムカウンタの値を変更しない

VMFUNC 利用時に気をつけるべき攻撃

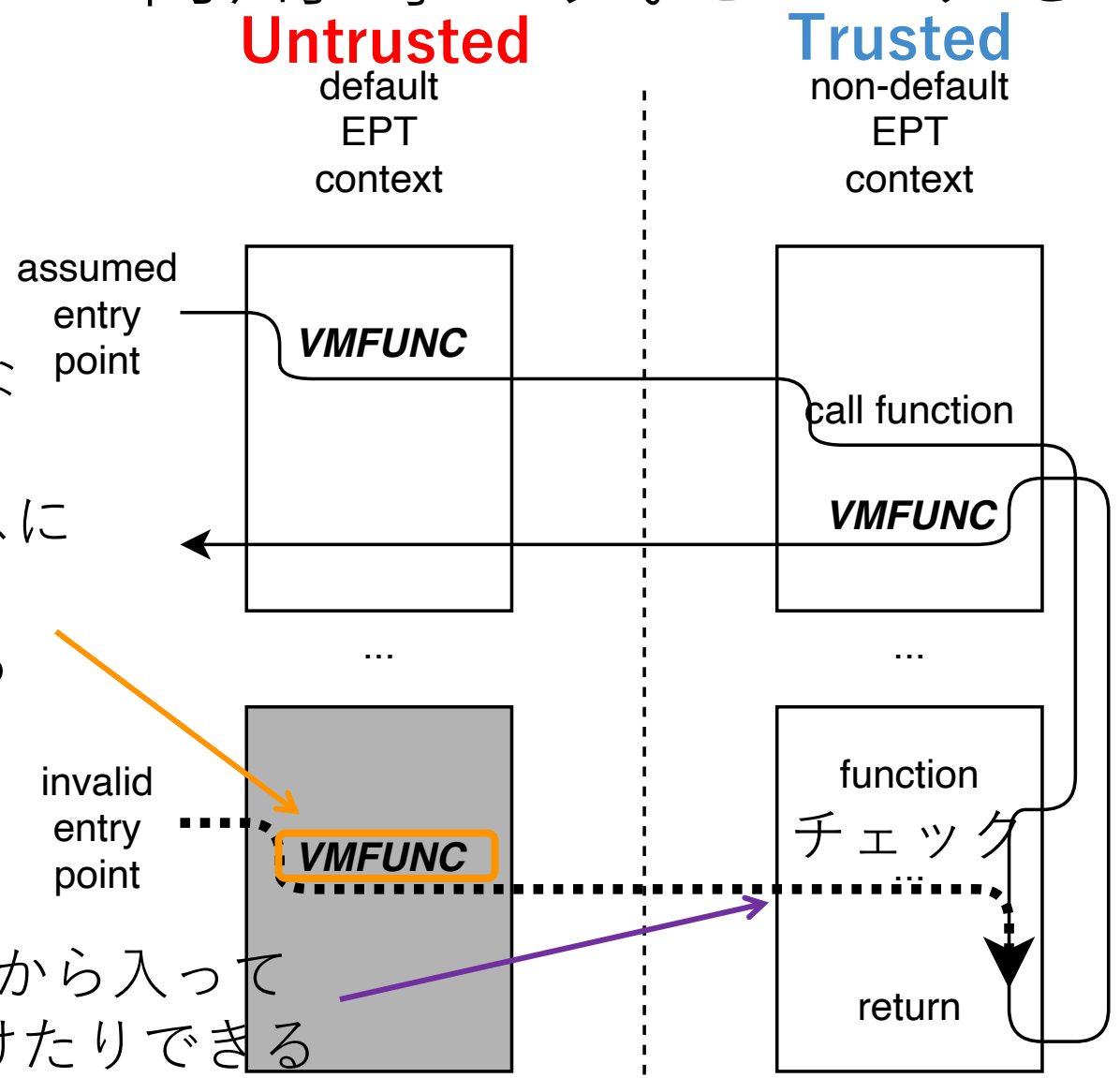


CR3 を操作可能な
ゲストは好きな
仮想メモリアドレスに
VMFUNC 命令を
配置し実行できる

VMFUNC は
プログラムカウンタの
値を変更しない

特定の入り口以外から入って
チェックを通り抜けたりできる

VMFUNC 利用時に気をつけるべき攻撃



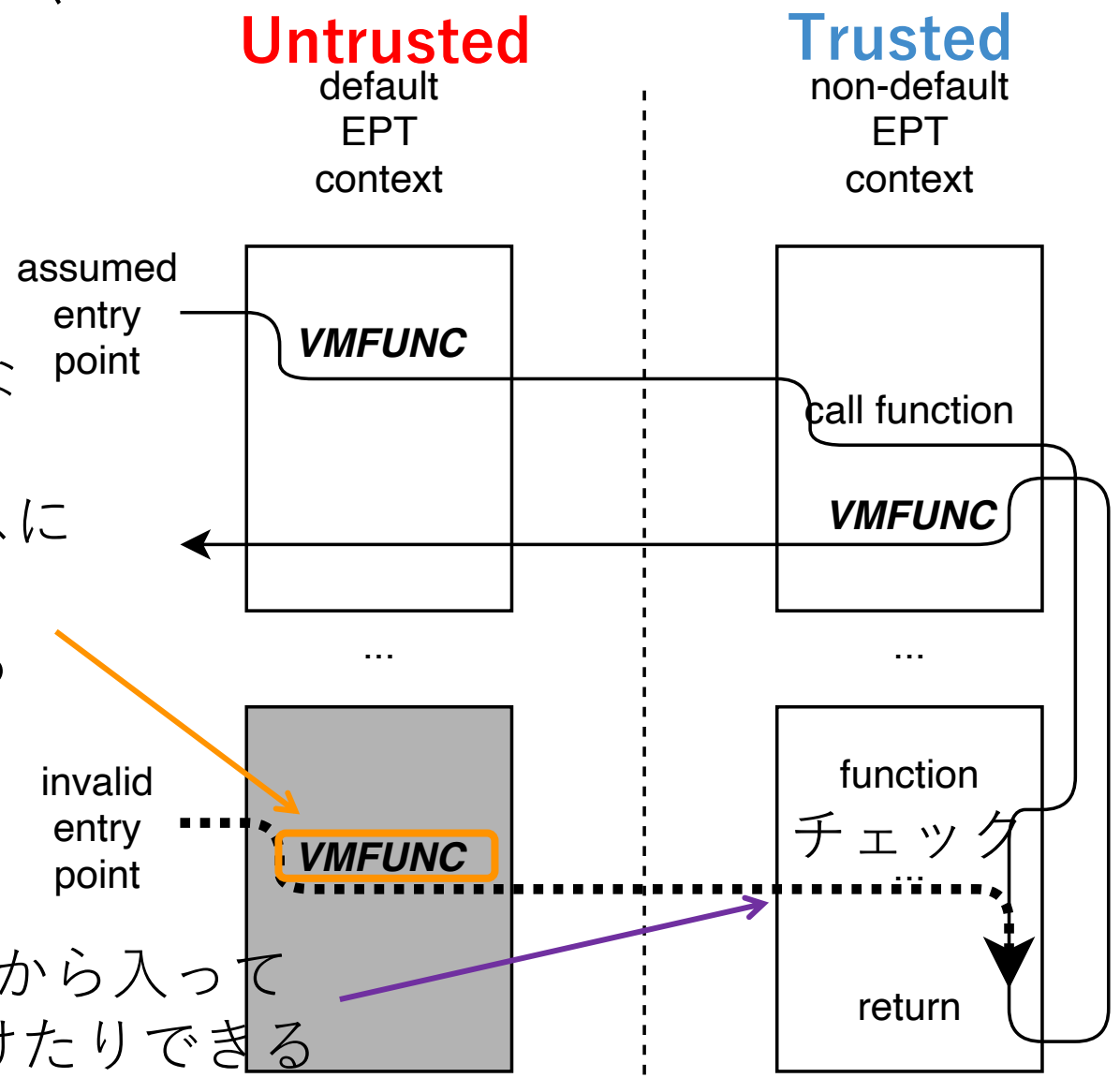
CR3 を操作可能な
ゲストは好きな
仮想メモリアドレスに
VMFUNC 命令を
配置し実行できる

特定の入り口以外から入って
チェックを通り抜けたりできる

VMFUNC は
プログラムカウンタの
値を変更しない

VMFUNC だけでは
コンテキスト
切り替え後の
着地位置を
強制できない

防御方法



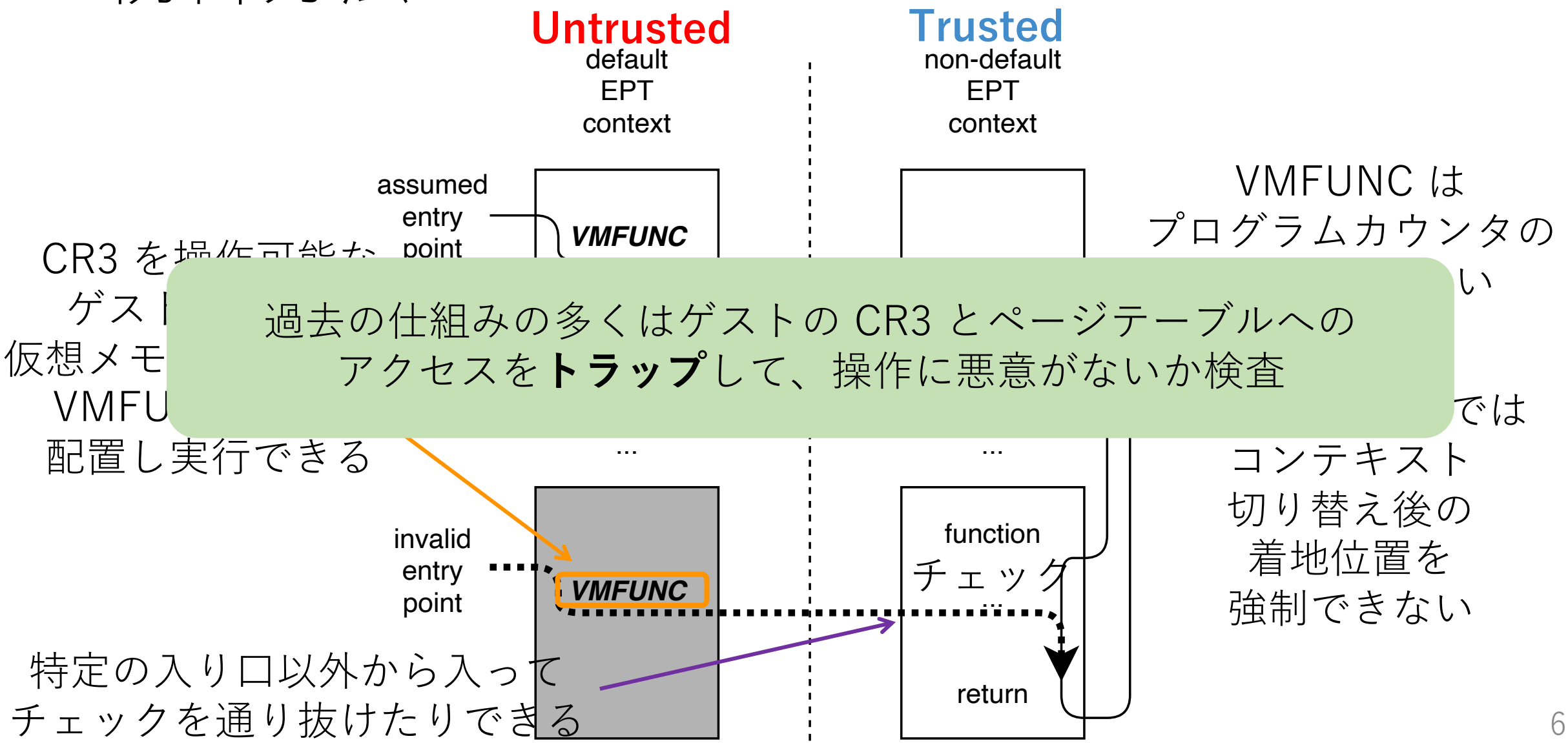
CR3 を操作可能な
ゲストは好きな
仮想メモリアドレスに
VMFUNC 命令を
配置し実行できる

特定の入り口以外から入って
チェックを通り抜けたりできる

VMFUNC は
プログラムカウンタの
値を変更しない

VMFUNC だけでは
コンテキスト
切り替え後の
着地位置を
強制できない

防御方法



過去の仕組みの多くはゲストの CR3 とページテーブルへの
アクセスを**トラップ**して、操作に悪意がないか検査

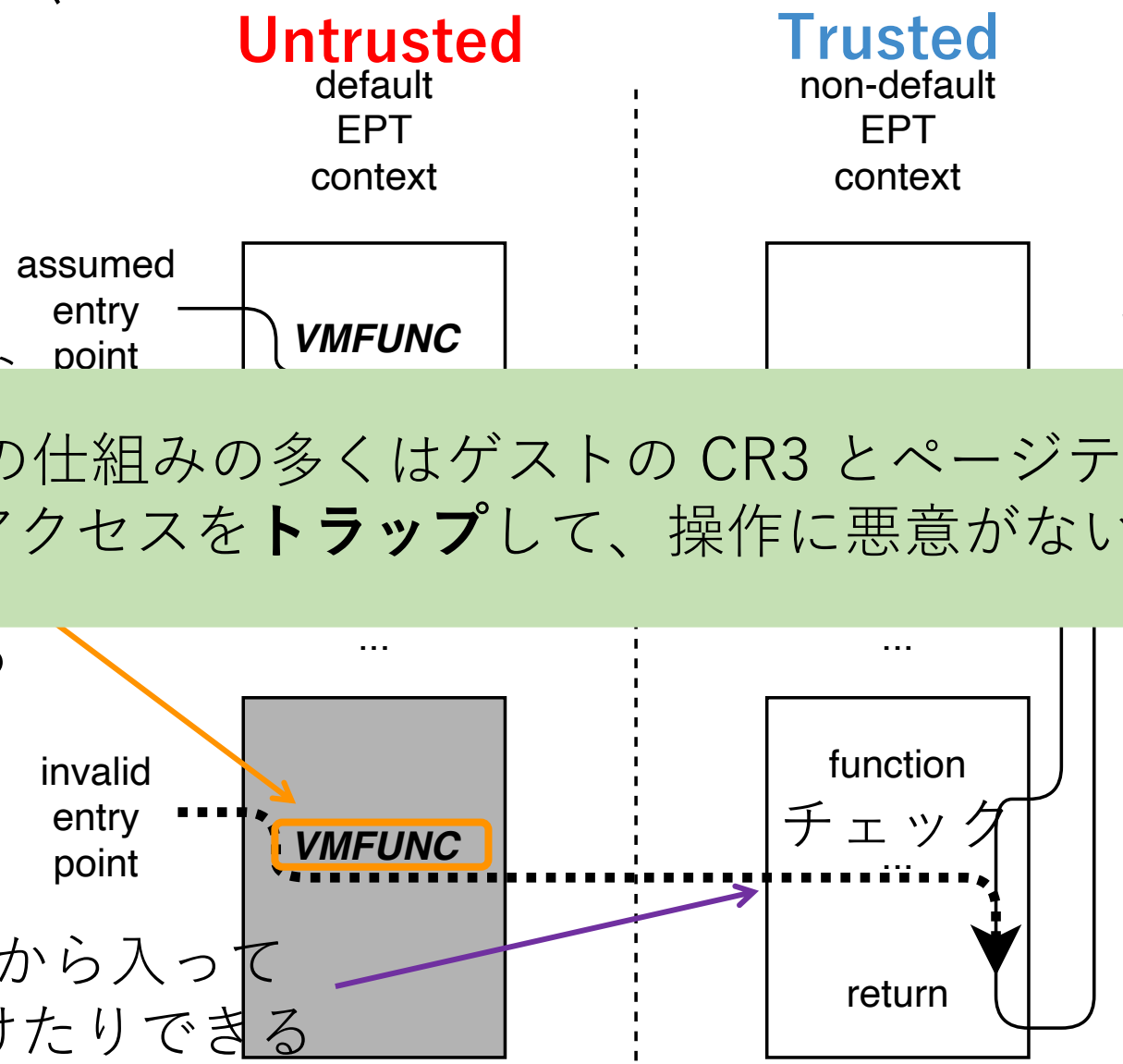
CR3 を操作可能な
ゲスト
仮想メモ
VMFU
配置し実行できる

特定の入り口以外から入って
チェックを通り抜けたりできる

VMFUNC は
プログラムカウンタの
い
では
コンテキスト
切り替え後の
着地位置を
強制できない

防御方法

できれば**トラップ**したくない



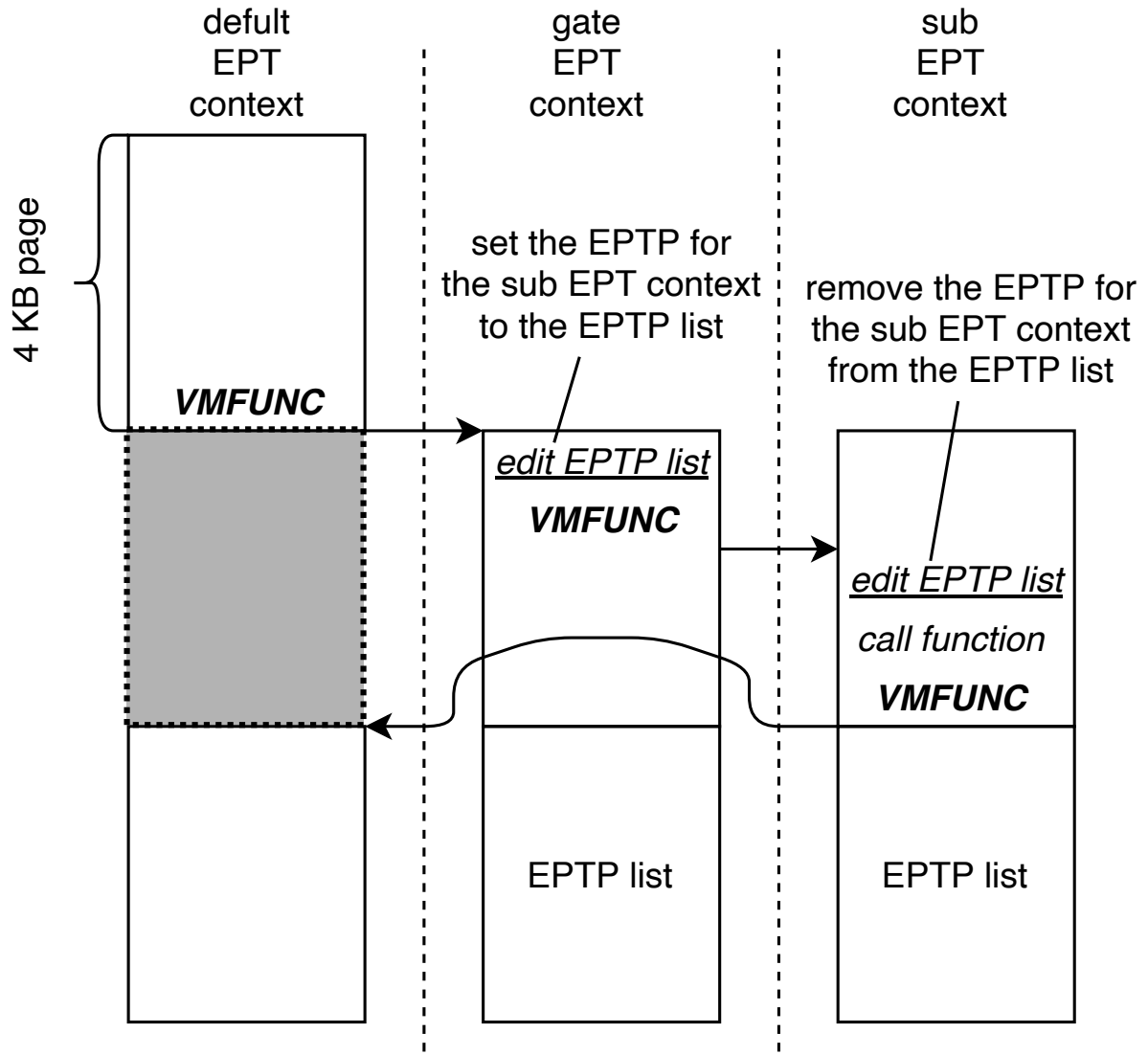
CR3 を操作可能な
ゲスト
仮想メモ
VMFU
配置し実行できる

過去の仕組みの多くはゲストの CR3 とページテーブルへの
アクセスを**トラップ**して、操作に悪意がないか検査

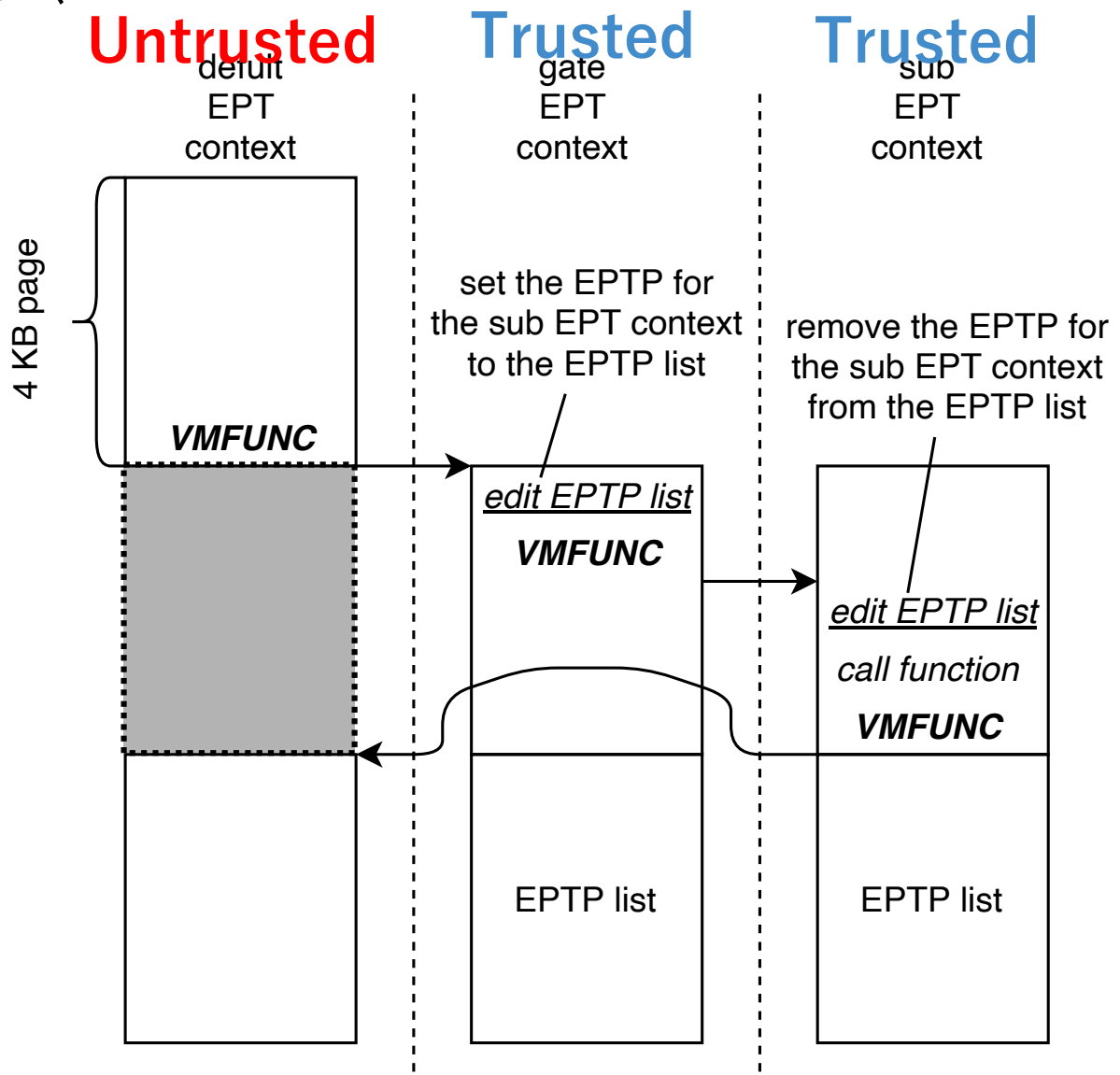
VMFUNC は
プログラムカウンタの
い
では
コンテキスト
切り替え後の
着地位置を
強制できない

特定の入り口以外から入って
チェックを通り抜けたりできる

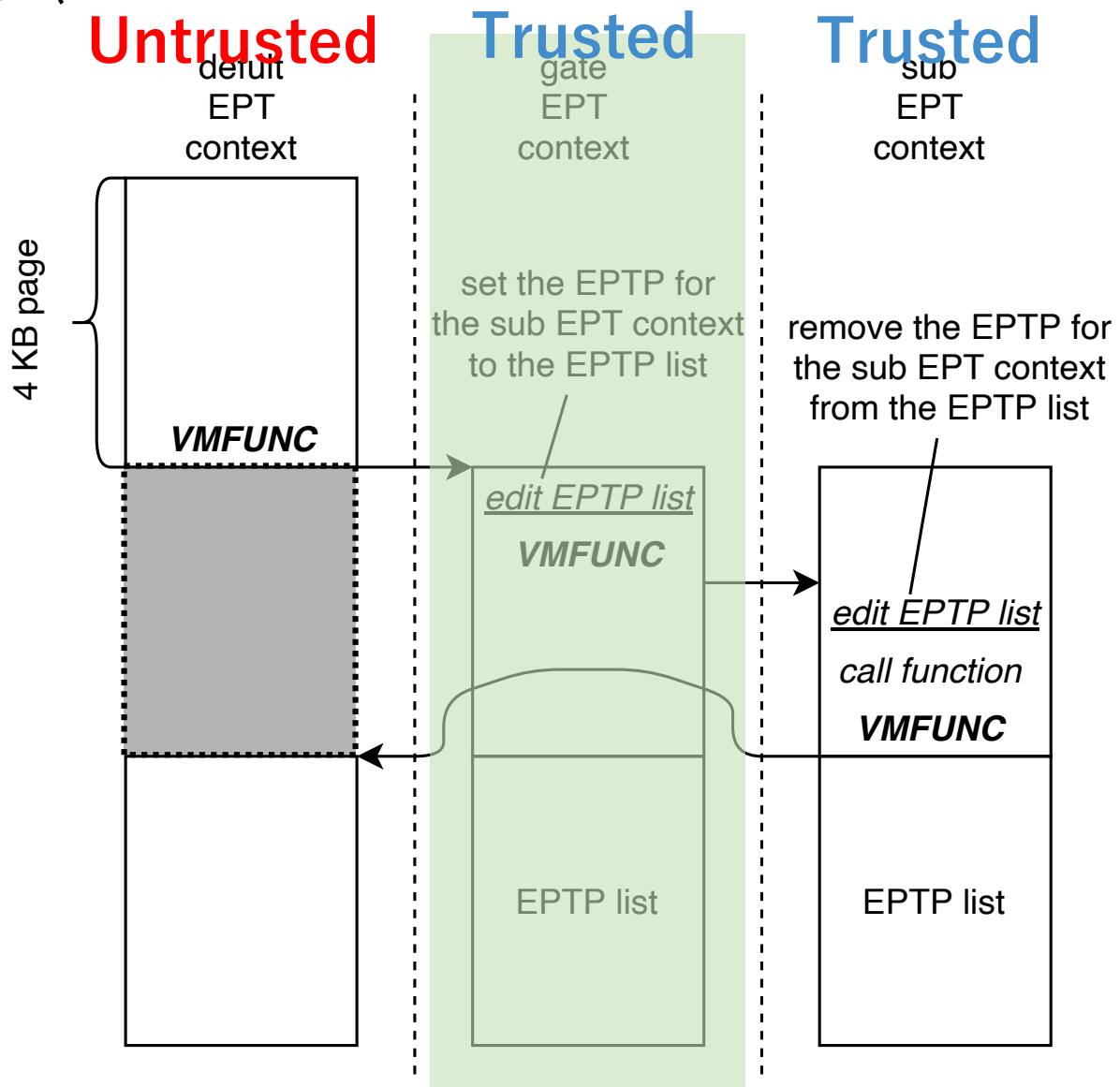
防御方法：Gate EPT Context



防御方法：Gate EPT Context

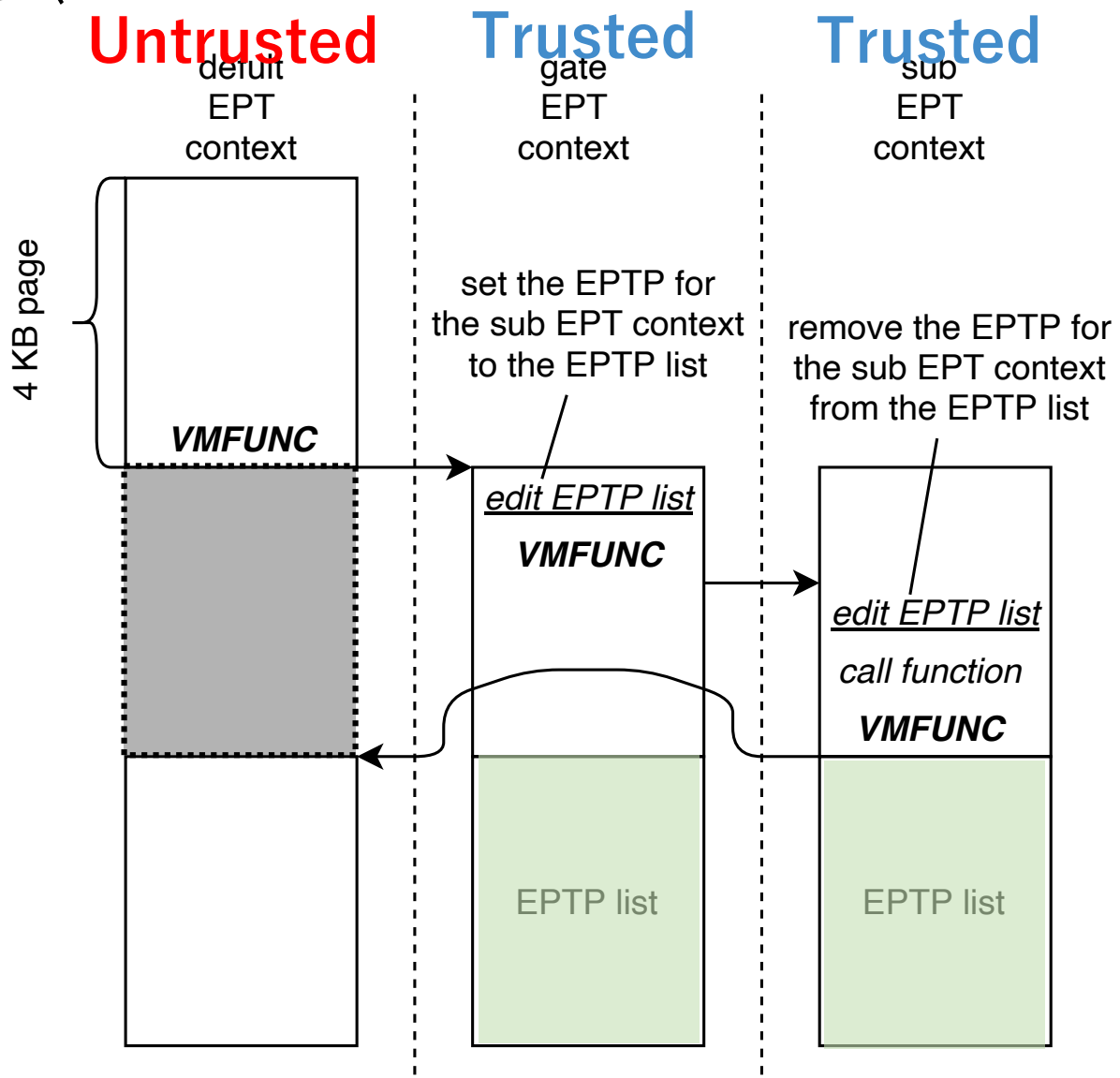


防御方法：Gate EPT Context



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する

防御方法：Gate EPT Context



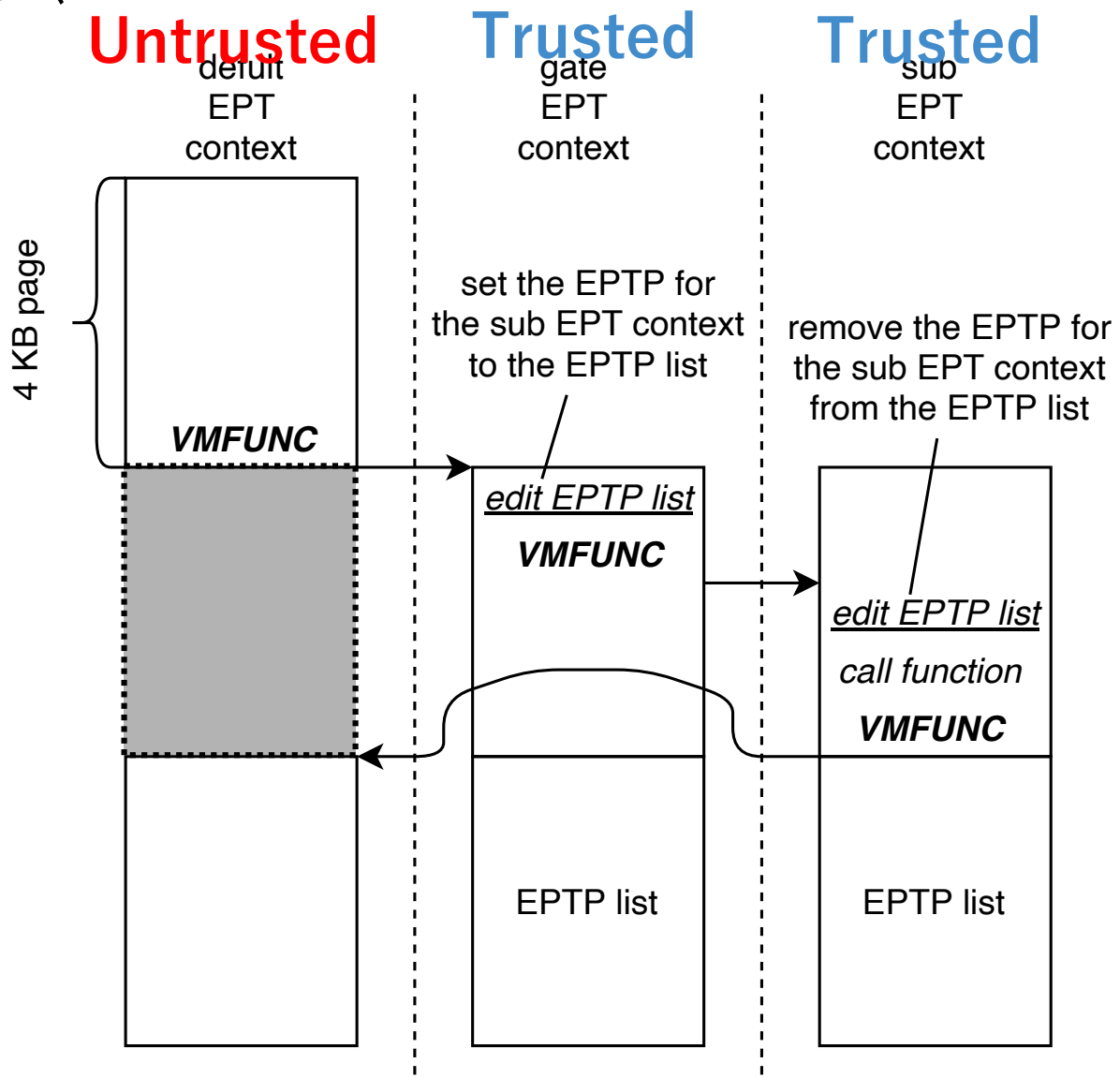
アイデア

コンテキスト移行を
仲介するコンテキストを
追加する

＋

EPTP list を trusted な
EPT コンテキストに
貼り付け

防御方法：Gate EPT Context



アイデア

コンテキスト移行を仲介するコンテキストを追加する

＋

EPTP list を trusted な EPT コンテキストに貼り付け

＋

コンテキスト移行中に EPTP list を編集する

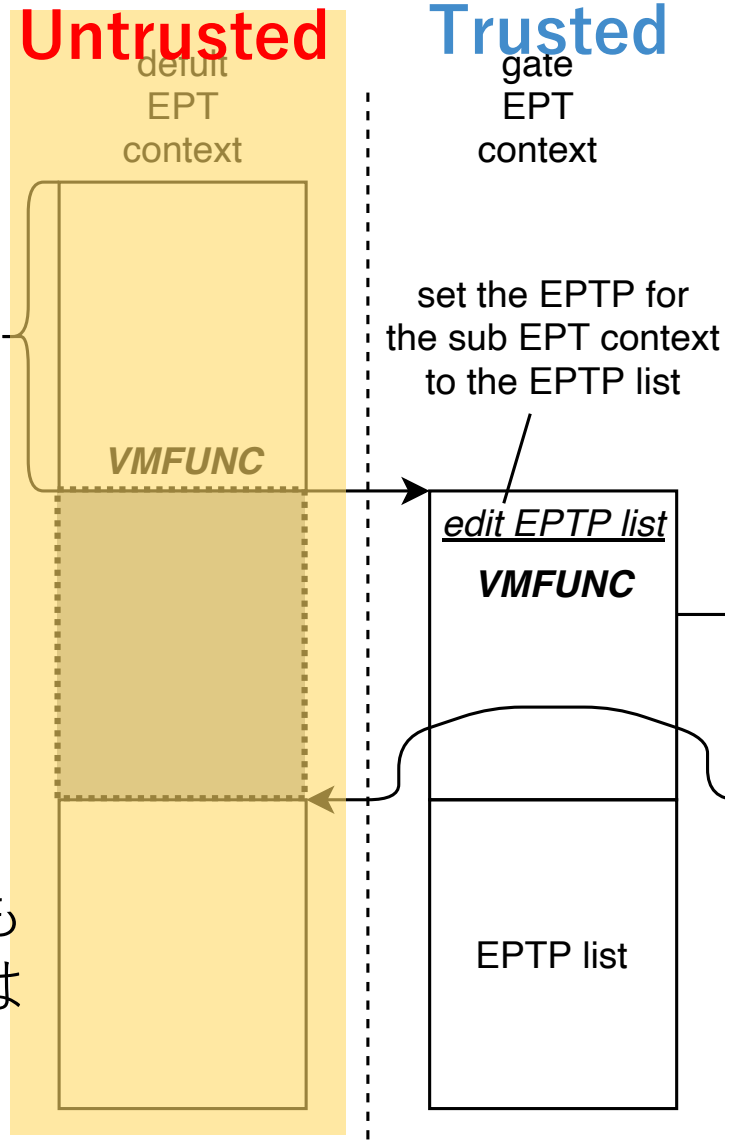
防御方法：Gate EPT Context

default コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[511]: gate EPT

EPTP list に
エントリがないため
ゲストはどうやっても
sub コンテキストへは
入れない



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

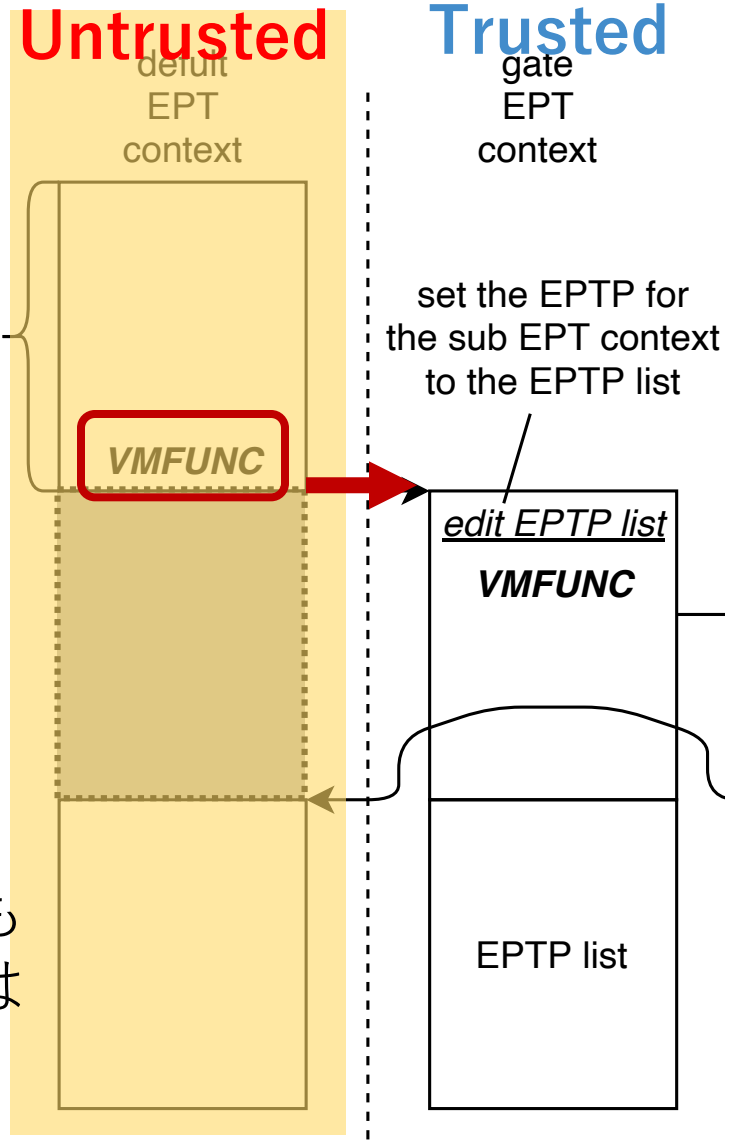
防御方法：Gate EPT Context

default コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[511]: gate EPT

EPTP list に
エントリがないため
ゲストはどうやっても
sub コンテキストへは
入れない



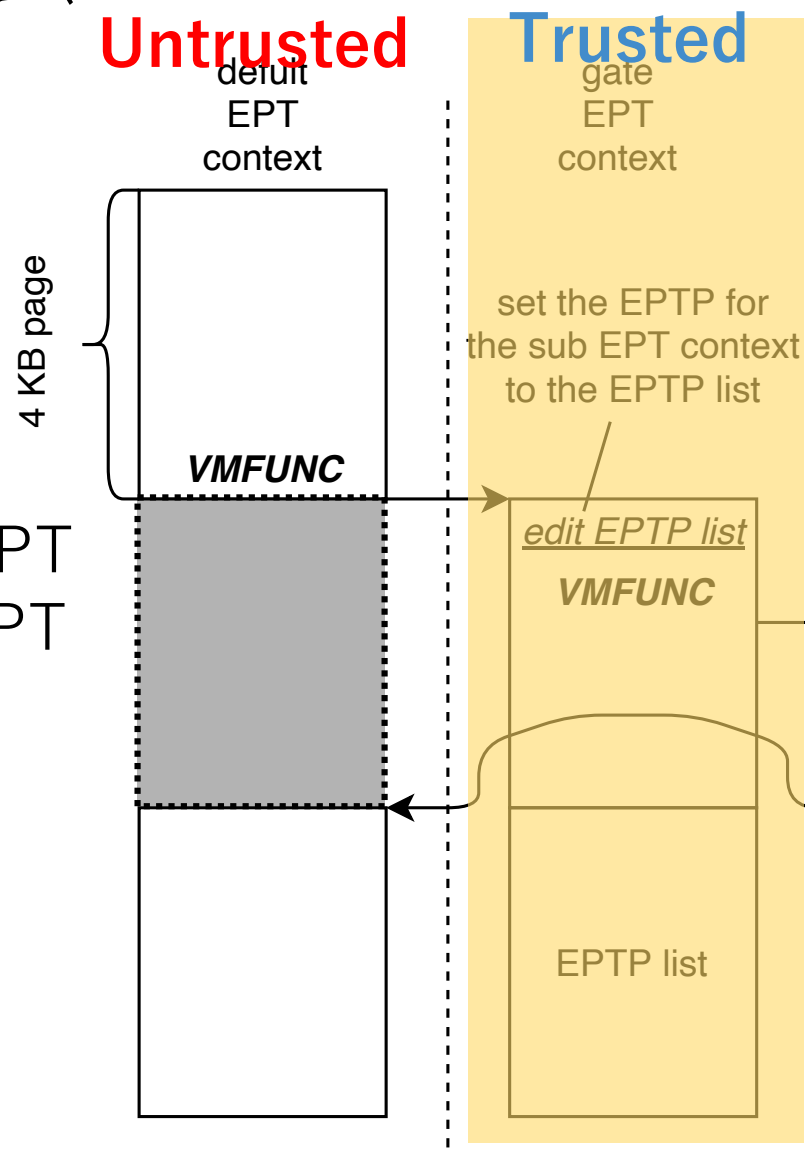
アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

防御方法：Gate EPT Context

gate コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[511]: gate EPT



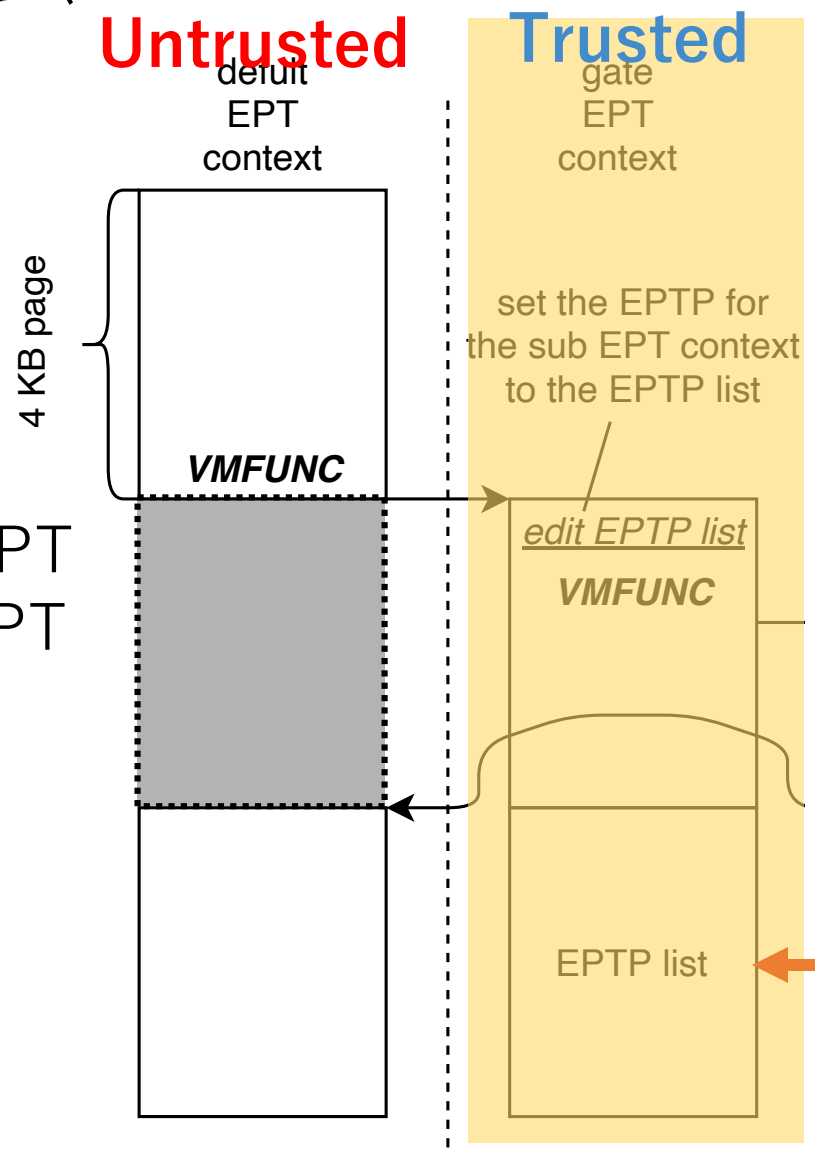
アイデア
コンテキスト移行を
仲介するコンテキストを
追加する

防御方法：Gate EPT Context

gate コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[511]: gate EPT



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

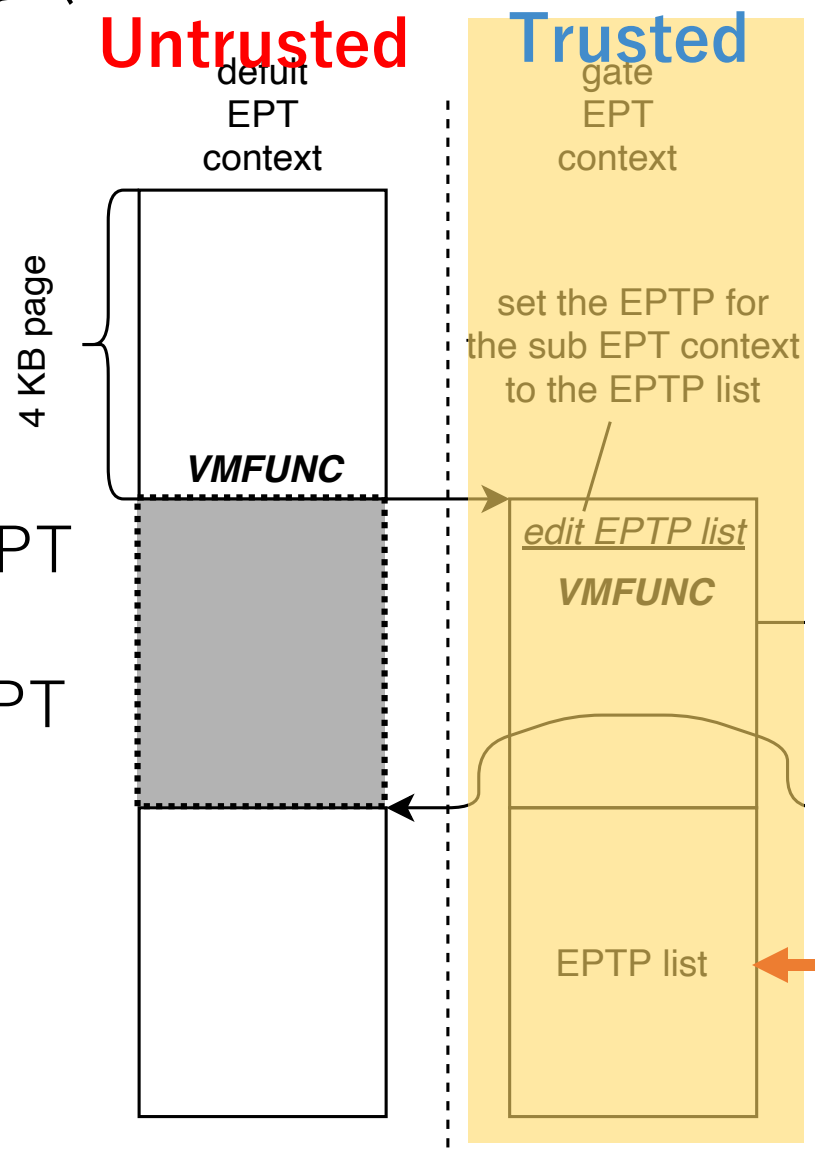
gate コンテキスト内で
EPTP list を編集：
sub EPT を追加

防御方法：Gate EPT Context

gate コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[1]: sub EPT
EPTP[511]: gate EPT



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

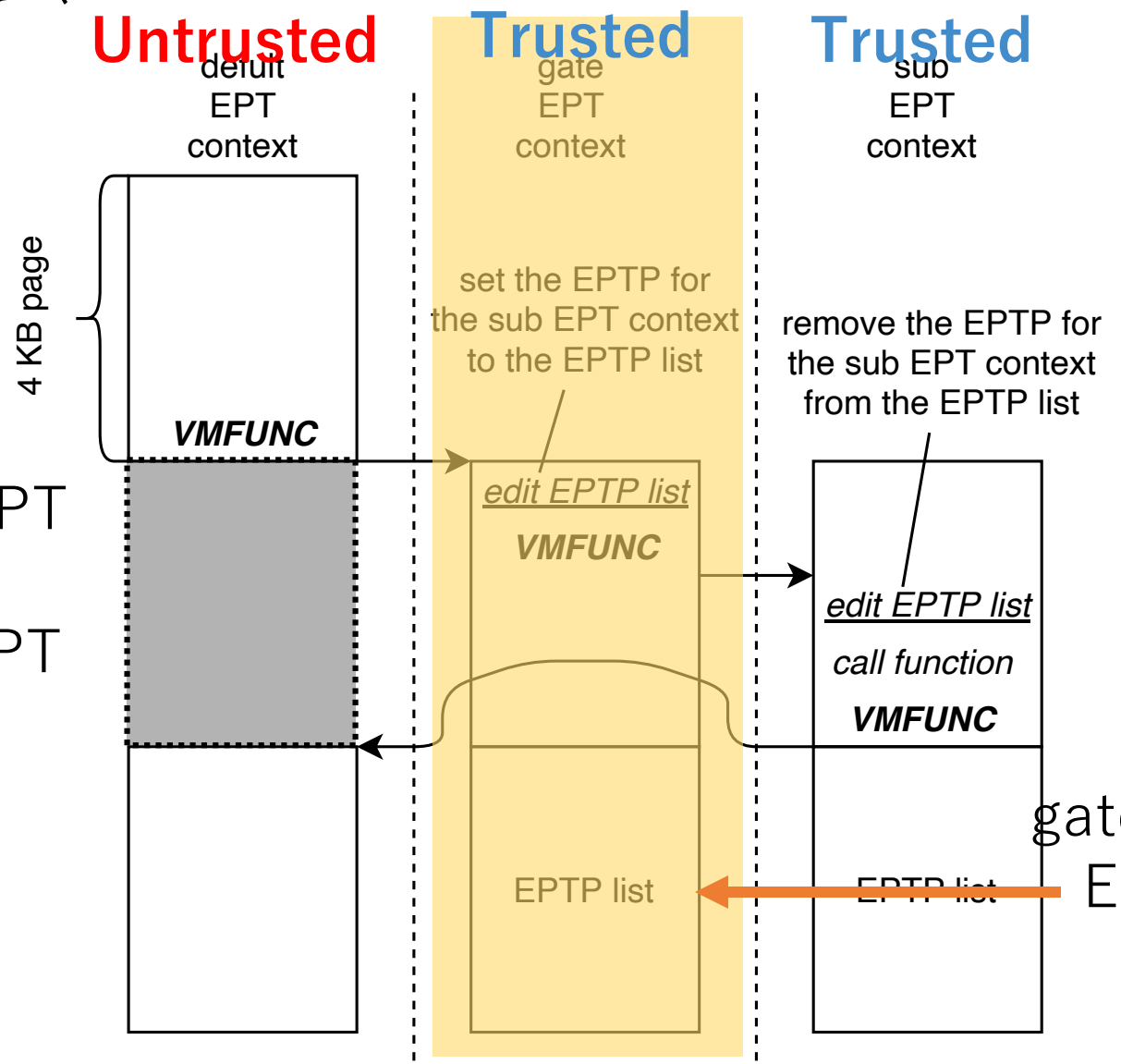
gate コンテキスト内で
EPTP list を編集：
sub EPT を追加

防御方法：Gate EPT Context

gate コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[1]: sub EPT
EPTP[511]: gate EPT



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

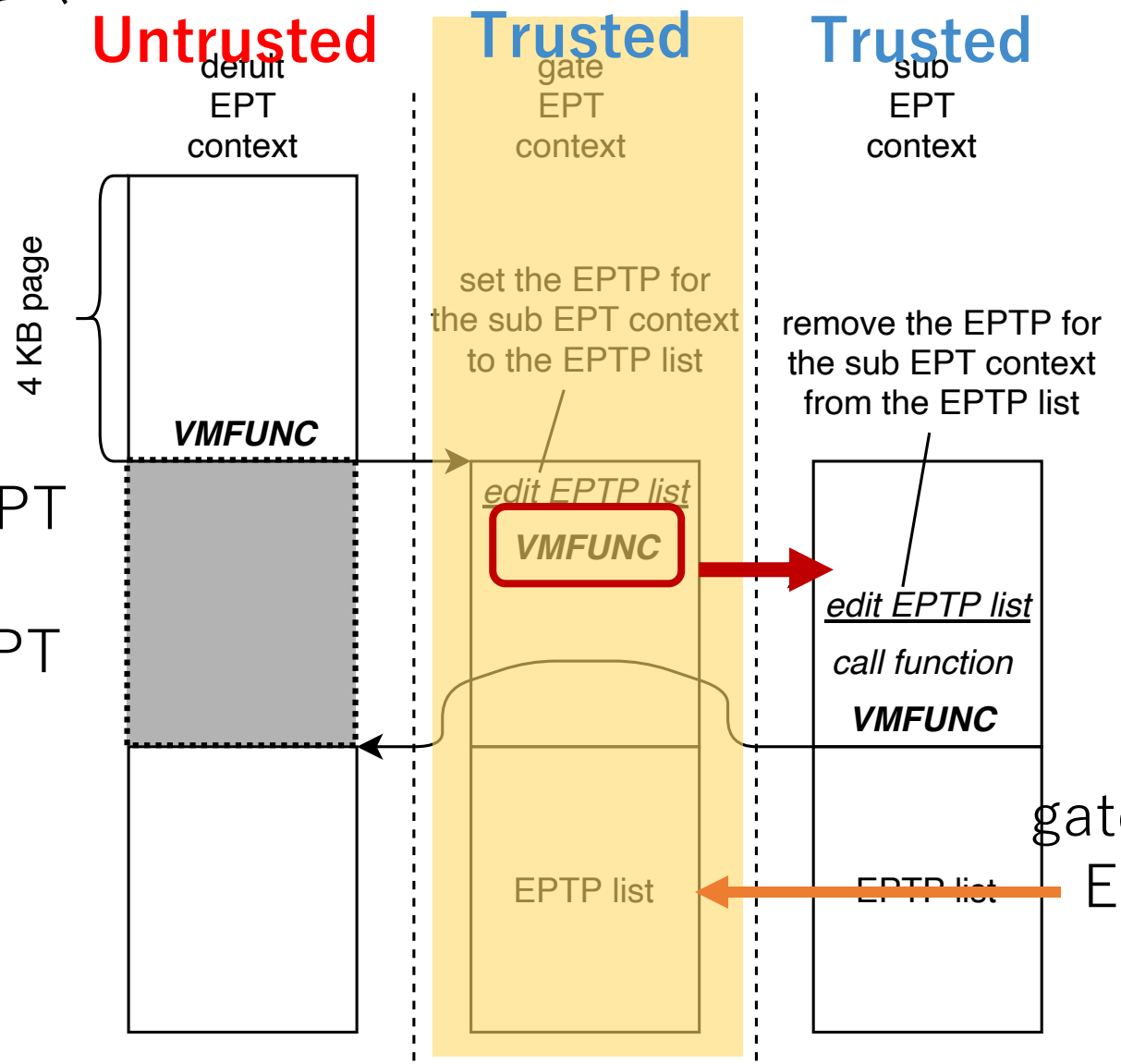
gate コンテキスト内で
EPTP list を編集：
sub EPT を追加

防御方法：Gate EPT Context

gate コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[1]: sub EPT
EPTP[511]: gate EPT



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

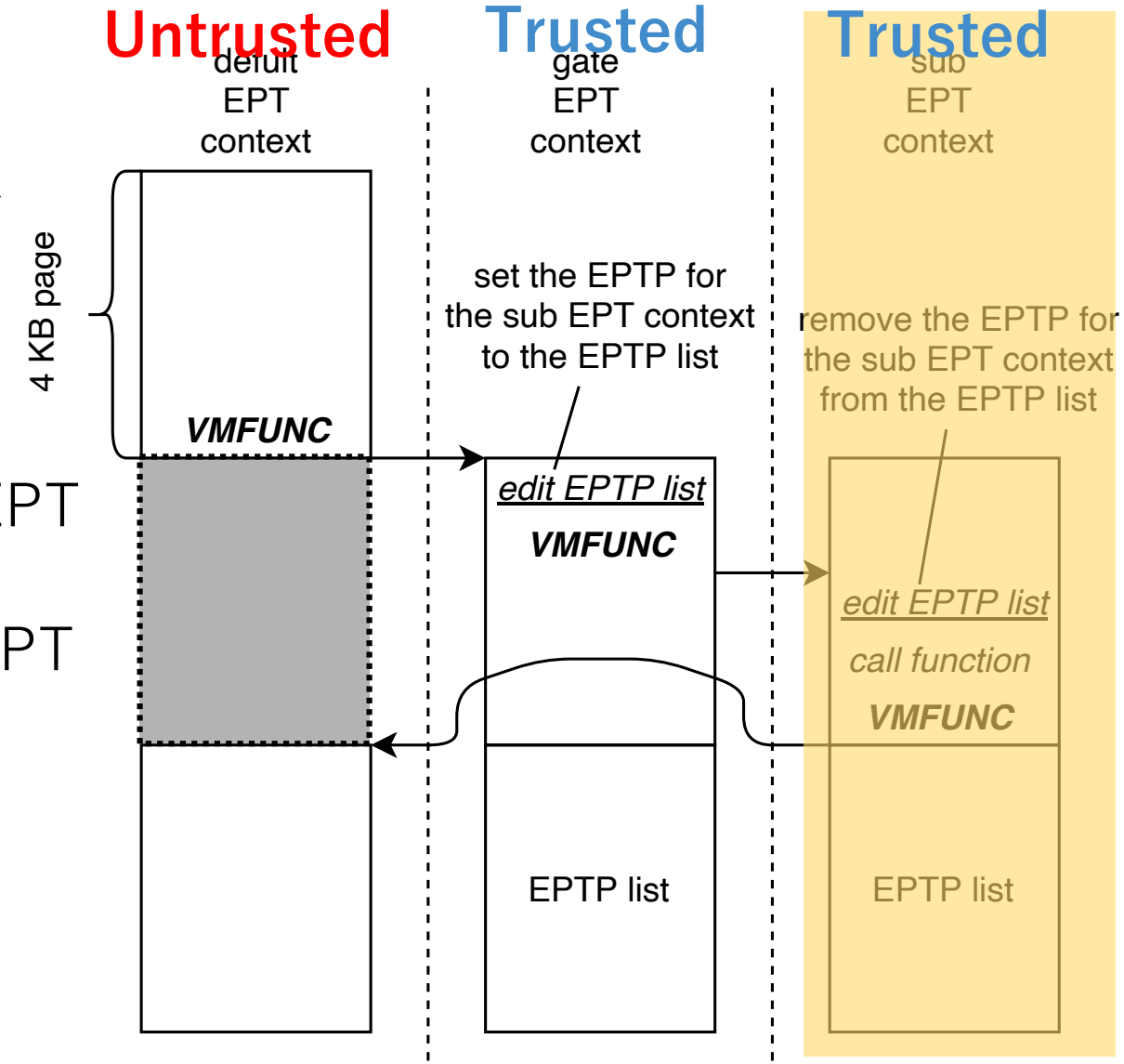
gate コンテキスト内で
EPTP list を編集：
sub EPT を追加

防御方法：Gate EPT Context

sub コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[1]: sub EPT
EPTP[511]: gate EPT



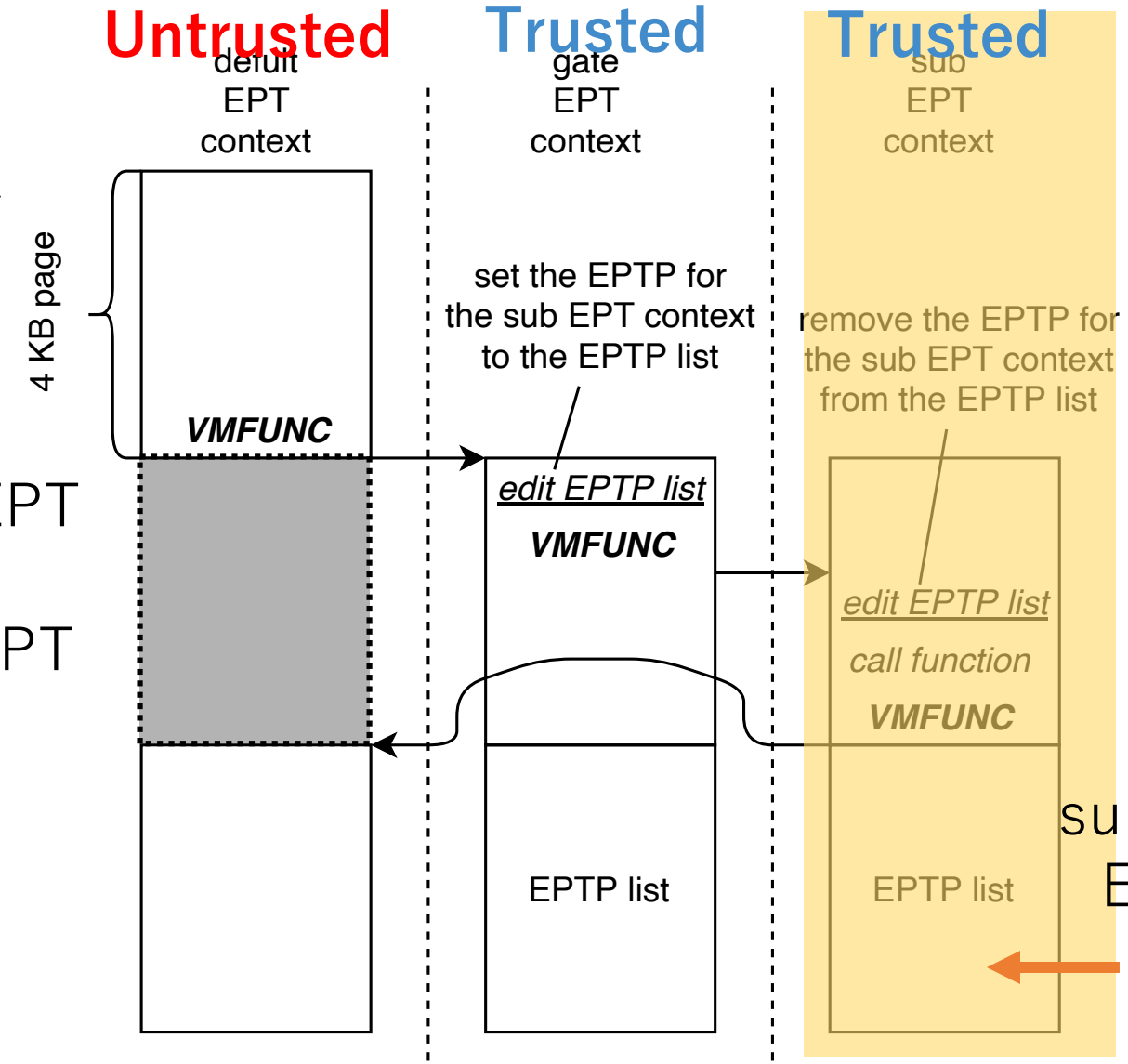
アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

防御方法：Gate EPT Context

sub コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[1]: sub EPT
EPTP[511]: gate EPT



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

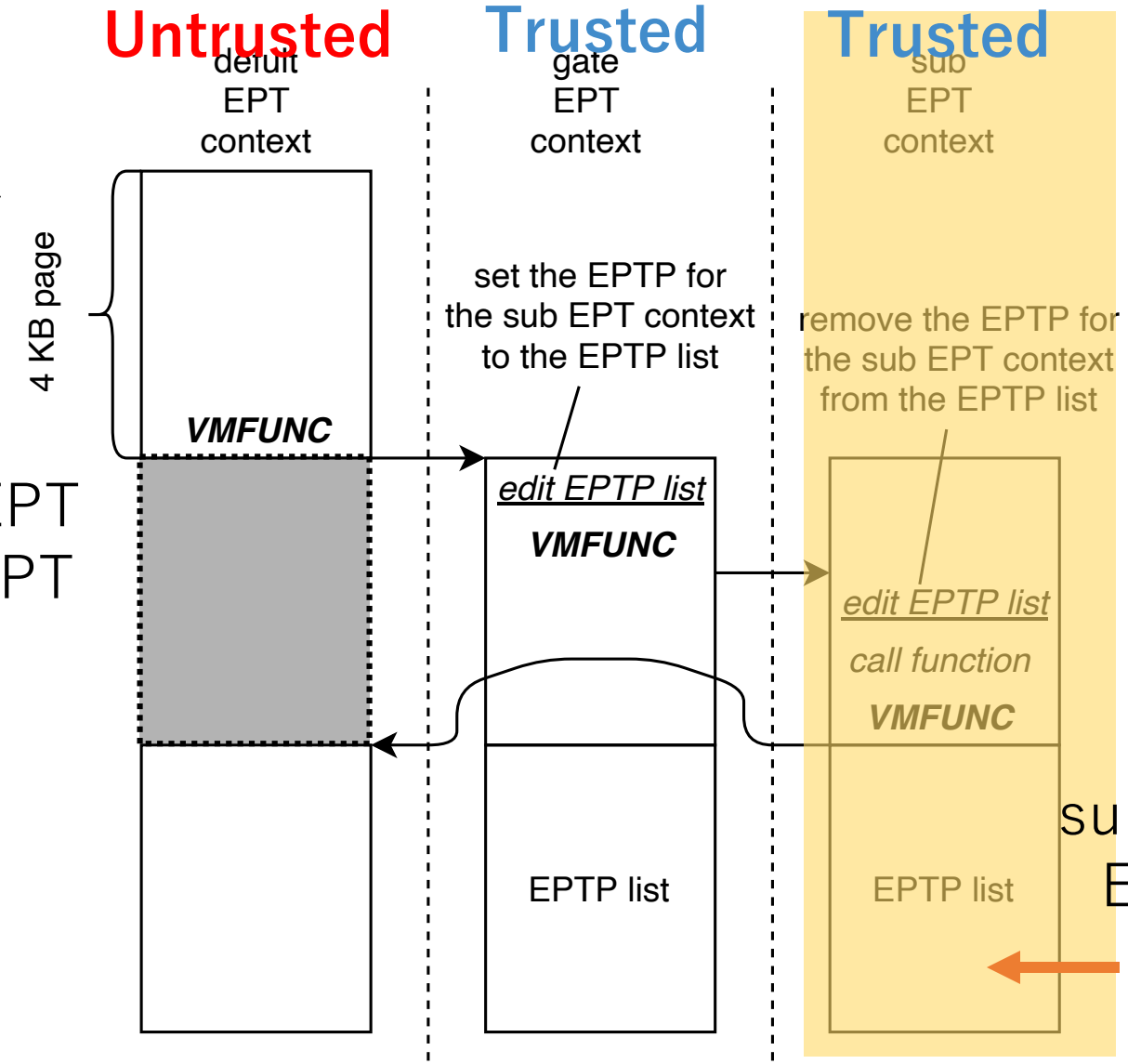
sub コンテキスト内で
EPTP list を編集：
sub EPT を削除

防御方法：Gate EPT Context

sub コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[511]: gate EPT



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

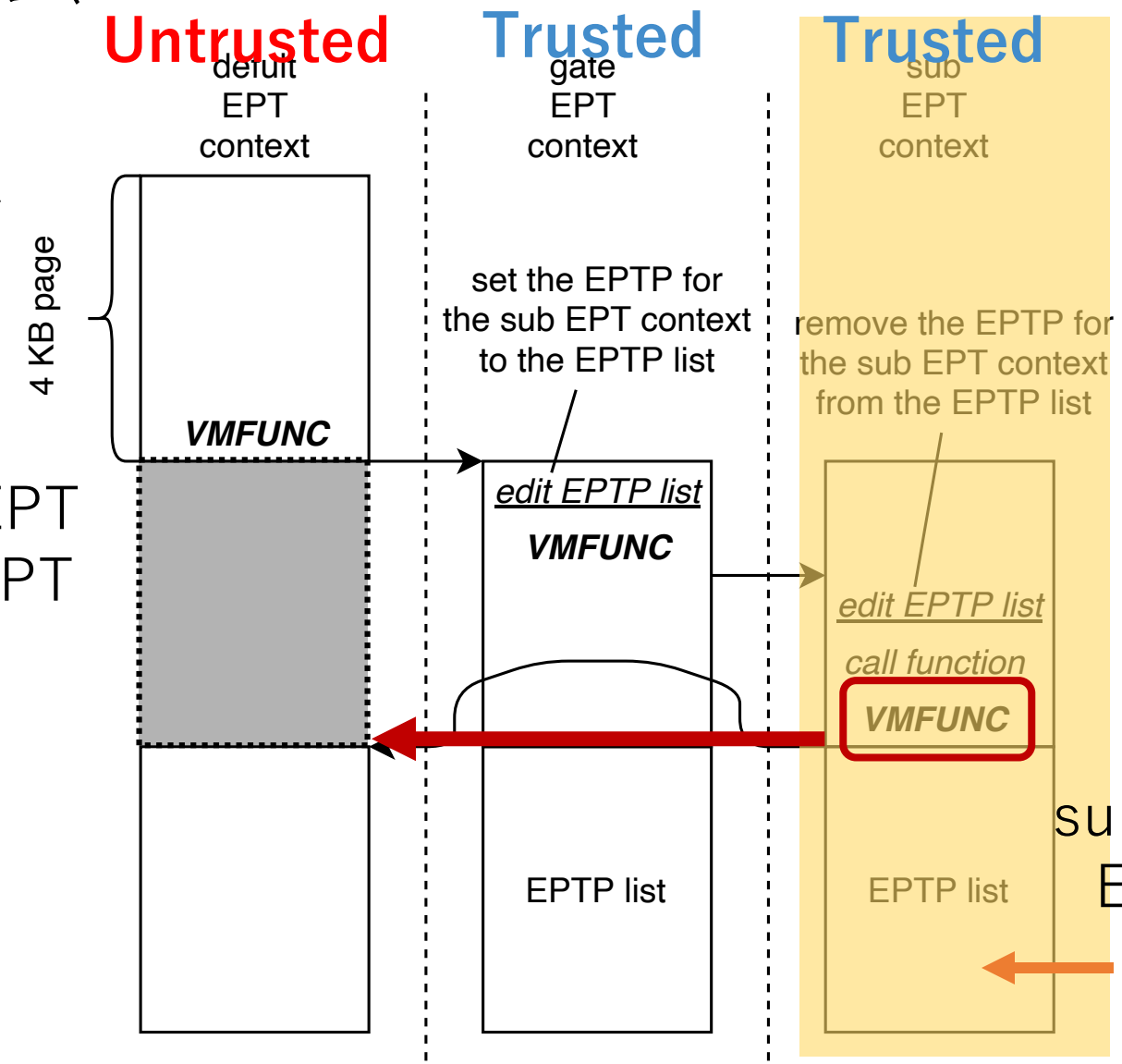
sub コンテキスト内で
EPTP list を編集：
sub EPT を削除

防御方法：Gate EPT Context

sub コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[511]: gate EPT



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

sub コンテキスト内で
EPTP list を編集：
sub EPT を削除

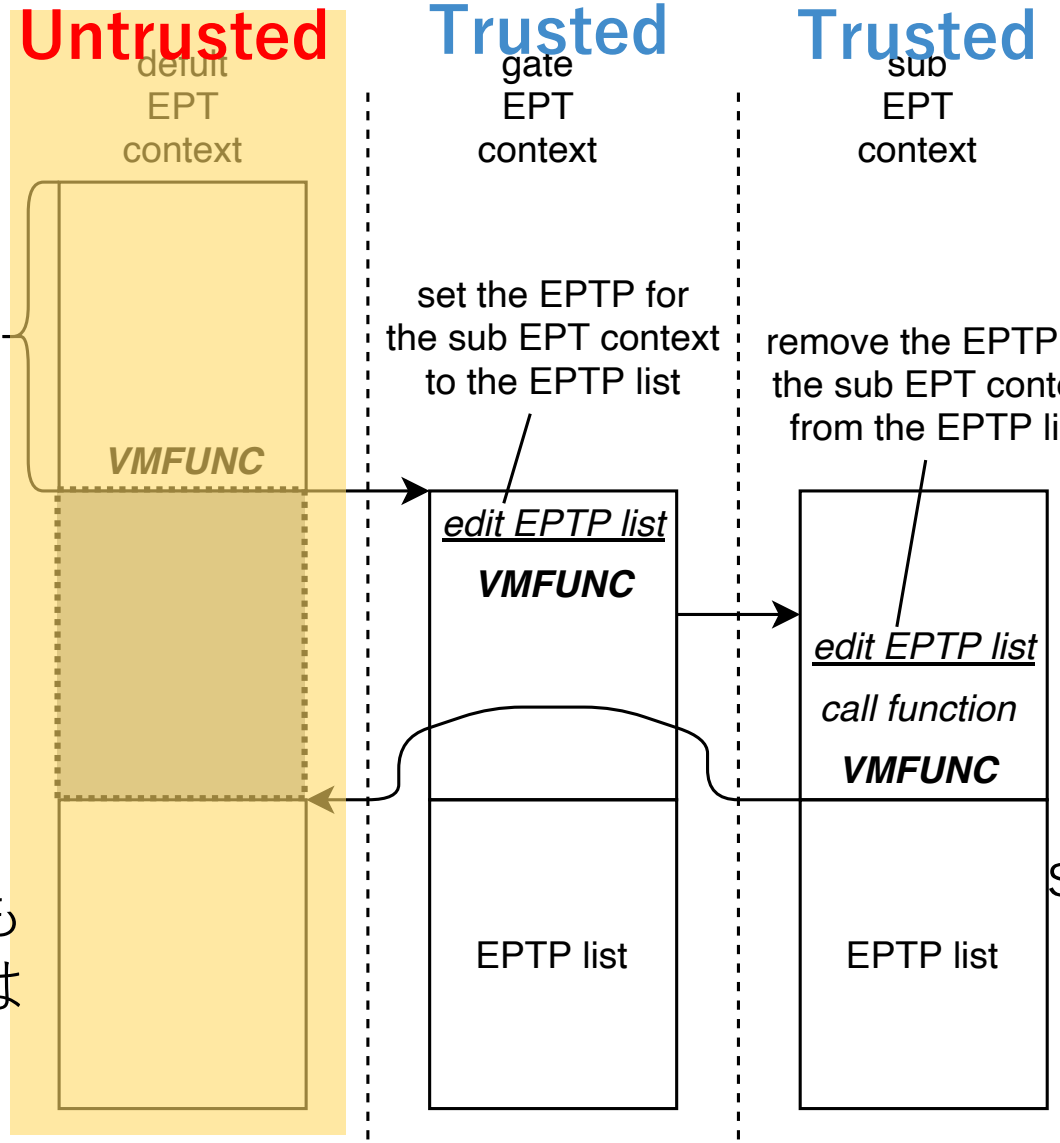
防御方法：Gate EPT Context

default コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[511]: gate EPT

EPTP list に
エントリがないため
ゲストはどうやっても
sub コンテキストへは
入れない



アイデア
コンテキスト移行を
仲介するコンテキストを
追加する
＋
EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

sub コンテキスト内で
EPTP list を編集：
sub EPT を削除

防御方法：Gate EPT Context

アイデア
コンテキスト移行を

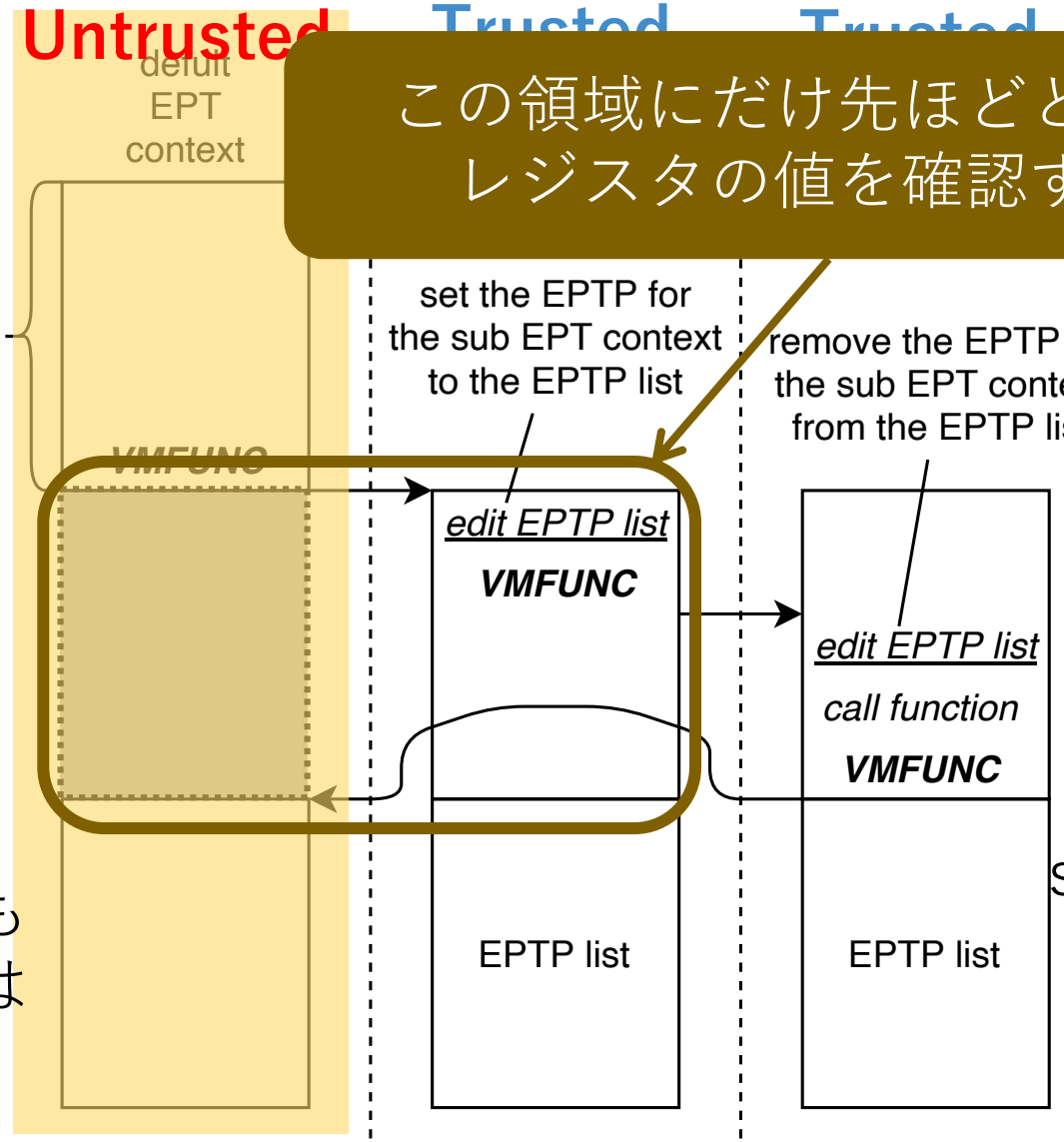
この領域にだけ先ほどと同じ攻撃ができるので
レジスタの値を確認するように気を付ける

default コンテキスト
にいる時

EPTP list

EPTP[0]: default EPT
EPTP[511]: gate EPT

EPTP list に
エントリがないため
ゲストはどうやっても
sub コンテキストへは
入れない



EPTP list を trusted な
EPT コンテキストに
貼り付け
＋
コンテキスト移行中に
EPTP list を編集する

sub コンテキスト内で
EPTP list を編集：
sub EPT を削除

実装

- KVM (120 行くらい追加)
  - 最小限のハイパーコール
    - https://github.com/yasukata/kvm-elisa
      - https://github.com/yasukata/ELISA#setup
- ユーザー空間プログラム (1200 行くらい)
  - PT/EPT 設定、アプリ開発用 API
    - https://github.com/yasukata/libelisa
      - https://github.com/yasukata/ELISA#section-52--libelisa
- QEMU とゲストカーネルは変更なし

https://github.com/yasukata/ELISA#list-of-the-elisa-prototype-repositories

実装の解説

https://github.com/yasukata/ELISA

README.md

The default EPT context

A guest VM only needs to load a 12 KB code block (4 KB x 3 : top, middle, and bottom) in its program to make an entry point.

The top 4 KB page has the entry point to the gate EPT context, named `elisa_gate_entry`, at the end of it.

The middle 4 KB page is empty. (The shaded part in Figure 5.)

When the execution returns from the sub EPT context, it lands at the top of the bottom 4 KB page.

The gate EPT context

The guest VM enters the gate EPT context by executing VMFUNC at the end of the top 4 KB page, and lands at the top

実装の解説

https://github.com/yasukata/ELISA

README.md

The default EPT context

A guest VM only needs to load a 12 KB code block (4 KB x 3 : top, middle, and bottom) in its program to make an entry point.

The top 4 KB page has the entry point to the gate EPT context, named `elisa_gate_entry`, at the end of it.

Click

The middle 4 KB page is empty. (The shaded part in Figure 5.)

When the execution returns from the sub EPT context, it lands at the top of the bottom 4 KB page.

The gate EPT context

The guest VM enters the gate EPT context by executing VMFUNC at the end of the top 4 KB page, and lands at the top

```
19  void ____asm_impl_client(void)
20  {
21          asm volatile (
22          /* -- 4K align -- */
23          ".align 0x1000 \n\t"
24          ".globl gate_entry_page \n\t"
25          "gate_entry_page: \n\t"
26
27          ".space 0x1000 - (gate_entry_end - elisa_gate_entry) \n\t"
28
29          ".globl elisa_gate_entry \n\t"
30          "elisa_gate_entry: \n\t"
31
32          "endbr64 \n\t"
33          "push %rbp \n\t"
34          "movq %rsp, %rbp \n\t"
35          "push %rbx \n\t"
36          "movq $0x0, %rax \n\t"
37          "vmfunc \n\t"
38          "gate_entry_end: \n\t"
39
40          /* -- 4K align -- */
41
42          /* this 4K page, will map gate_ctx_page in gate context */
43          ".space 0x1000 \n\t"
44
45          /* -- 4K align -- */
```

背景知識の解説

- https://yasukata.hatenablog.com/entry/2023/04/10/085714
  - メモリの分離・共有
  - ページテーブル
  - 共有メモリ
  - ...

OS についての研究

スライド

https://yasukata.github.io/presentation/2023/05/ipsj-sigos159/ipsj-sigos159-os.pdf

解説記事

- https://yasukata.hatenablog.com/entry/2021/10/14/145642
- https://yasukata.hatenablog.com/entry/2021/11/19/144708

まとめ

まとめ

CPU と比べ I/O デバイス性能の向上の方が速いことから
ソフトウェアの CPU 利用効率への要求が厳しくなっている

まとめ

CPU と比べ I/O デバイス性能の向上の方が速いことから
ソフトウェアの CPU 利用効率への要求が厳しくなっている

機能・特性を維持しつつ CPU 利用効率の向上を目指すと
解決されていない課題が結構ある

まとめ

CPU と比べ I/O デバイス性能の向上の方が速いことから
ソフトウェアの CPU 利用効率への要求が厳しくなっている

機能・特性を維持しつつ CPU 利用効率の向上を目指すと
解決されていない課題が結構ある

<u>取り組んでいる研究</u>

- 仮想マシン：分離と共有の両立のためのコスト低減
- OS：ユーザー空間 OS 機能の互換性向上