Mintオペレーティングシステムにおけるコア管理機能の実現

池田 騰　乃村 能成　谷口 秀夫
岡山大学大学院自然科学研究科

目次

目次

はじめに

計算機性能の向上にともない，

1台の計算機上に複数のOSを走行させる研究が活発

＜1台の計算機上で複数のOSを走行させる方式＞

(1) 仮想計算機方式

問題点：仮想化によるオーバヘッド，OS間の依存性

(2) Mint

仮想化によらず複数のLinuxを独立に走行させる方式

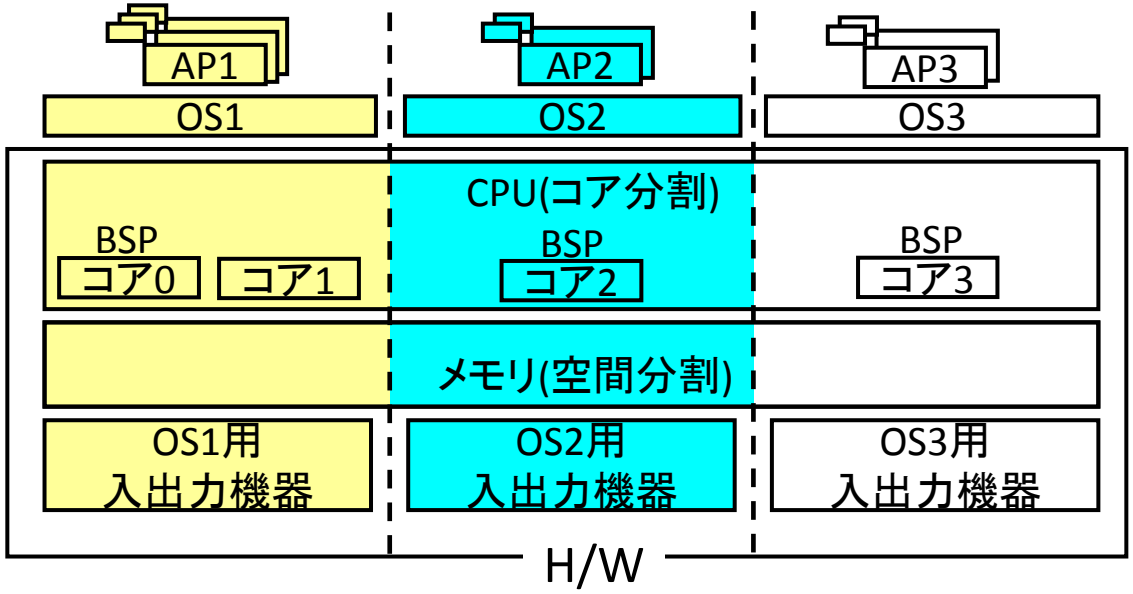
⇨ 仮想化によるオーバヘッド，OS間の依存性の問題を解決

⇩

OS間でのコアの移譲を可能とするコア管理機能を実現

目次

Mintの特徴



＜Mint＞

(1) 仮想化によらず1台の計算機上で複数のOSを動作

(2) 起動時にCPUをコア単位で分割し，OS毎に割り当てる

最初に起動するOS1のBoot Strap Processor(以下，BSP)はコア0

後から起動するOS2，OS3のBSPは起動時に指定したコア

(3)全OSは互いに独立したフルセットのLinuxとして動作

⇨ 既存のアプリケーションをそのまま使用可能

コア管理機能の目的

Mintでは，コアの分配を起動時に静的に設定

⇨ 負荷に応じたコアの動的な分配が不可能

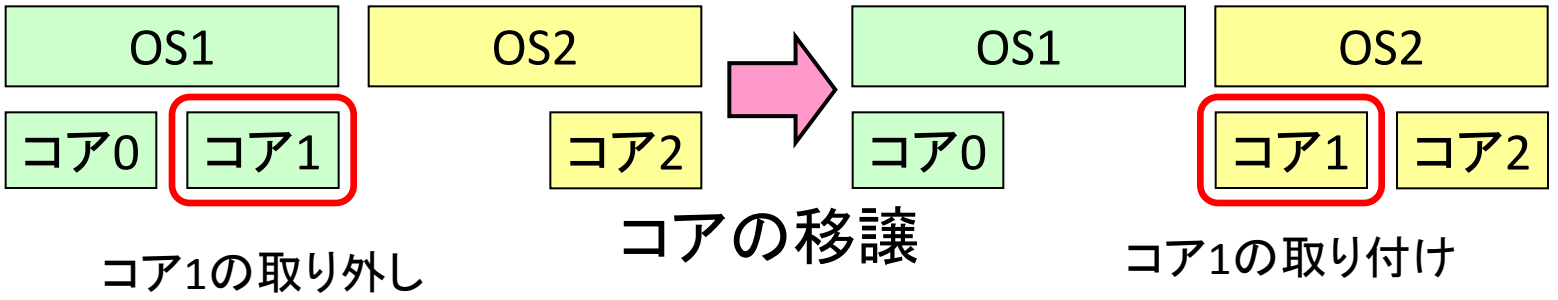
＜要求＞

起動後のコアの再分配を可能にしたい

CPUホットプラグを用いて，OS間でのコアの移譲を実現

＜CPUホットプラグ＞

Linuxが持つ，動的にCPUコアを着脱する機能



目次

問題点

＜コア管理機能を実現する際の問題点＞

(問題1) ローカルタイマ割り当ての重複

(問題2) 各種識別ID の重複

(問題3) OS 間にまたがるコア管理機構の欠如

問題点

＜コア管理機能を実現する際の問題点＞

(問題1) ローカルタイマ割り当ての重複

(問題2) 各種識別ID の重複

(問題3) OS 間にまたがるコア管理機構の欠如

ローカルタイマ割り当て

Linuxでは，各コアのタイマ割り込みの供給源として，
High Precision Event Timer(以下，HPET)を利用

＜Linux におけるHPET 利用方法＞

(1) HPET 0，1 をグローバルタイマ，イベントタイマとして利用

(2) HPET 2 以降を各コア用のローカルタイマとして分配

(3) 起動時に利用可能なタイマを全て検出し，HPET管理表に登録

(4) コア初期化時に管理表を参照し，未使用のHPETを割り当てる

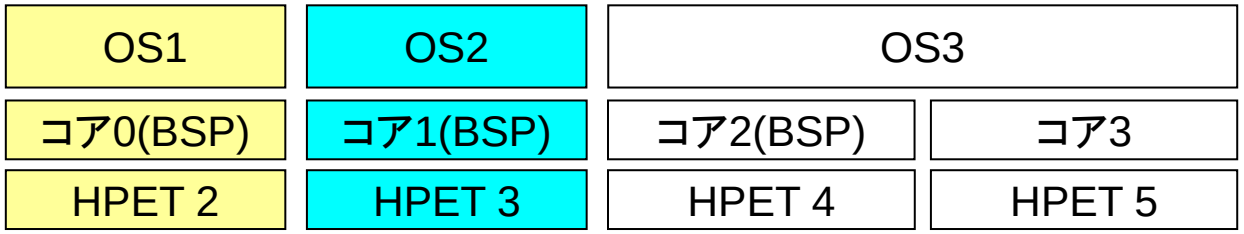
Mintにおいて，各OSはそれぞれ独自のHPET管理表を持つ

⇨ 他OSが占有するコアが持つHPETを別のOSが奪う危険性

⇩

HPET管理表について整合性を保つ必要

ローカルタイマ割り当ての重複



offset　　1　　2

HPET	2	3	4	5	・・・
OS1のHPET管理表	利用中	利用可能	利用可能	利用可能	・・・
OS2のHPET管理表	検索対象外	利用中	利用可能	利用可能	・・・
OS3のHPET管理表	検索対象外		利用中	利用中	・・・

＜従来のMintの対処＞

OS間で利用しないHPETを静的に決定

⇨動的なコア分配の際に整合性を保てない

⇩

新たなHPETの割り当て方式が必要

問題点

＜コア管理機能を実現する際の問題点＞

(問題1) ローカルタイマ割り当ての重複

(問題2) 各種識別ID の重複

(問題3) OS 間にまたがるコア管理機構の欠如

各種識別ID

(1) Local APIC ID(以下，LAPIC ID)
(2) コアID
(3) 論理APIC ID

} OS間での重複が問題

LAPIC ID

		OS1		OS2	OS3
		コア0(BSP)	コア1	コア2(BSP)	コア3(BSP)
	LAPIC ID	LAPIC ID: 0	LAPIC ID: 2	LAPIC ID: 4	LAPIC ID: 6
OS1	コアID	0	1	2	3
	論理APIC ID	1	2	4	8
OS2	コアID	1	2	0	3
	論理APIC ID	2	4	1	8
OS3	コアID	1	2	3	0
	論理APIC ID	2	4	8	1

(1) OSの起動前に設定される固定のID

(2) OS間で一意の値を持つ

コアID

		OS1		OS2	OS3
		コア0(BSP)	コア1	コア2(BSP)	コア3(BSP)
	LAPIC ID	LAPIC ID: 0	LAPIC ID: 2	LAPIC ID: 4	LAPIC ID: 6
OS1	コアID	0	1	2	3
	論理APIC ID	1	2	4	8
OS2	コアID	1	2	0	3
	論理APIC ID	2	4	1	8
OS3	コアID	1	2	3	0
	論理APIC ID	2	4	8	1

(1) Linuxが自身の占有するコアの識別のために広く使用するID

(2) BSPを基準に設定されるため，OS毎に異なるID

BSPをコアID 0 とし，以降はコアの検出順にIDを割り当てる

(3) 通常はOS内でのみ使用するため，OS間の重複は問題ない

⇨ OS間でコアのやり取りをする場合に問題

論理APIC ID

		OS1		OS2	OS3
		コア0(BSP)	コア1	コア2(BSP)	コア3(BSP)
	LAPIC ID	LAPIC ID: 0	LAPIC ID: 2	LAPIC ID: 4	LAPIC ID: 6
OS1	コアID	0	1	2	3
	論理APIC ID	1	2	4	8
OS2	コアID	1	2	0	3
	論理APIC ID	2	4	1	8
OS3	コアID	1	2	3	0
	論理APIC ID	2	4	8	1

(1) 割り込みの通知先の指定に用いられるID

(2) コアIDを基準に生成されるため，OS間で異なるID

論理APIC ID = 1<<コアID

各OSが占有するコアの論理APIC IDが重複

割り込みが正しく通知されない問題が発生

各種識別IDの重複

	OS1		OS2	OS3
	コア0(BSP)	コア1	コア2(BSP)	コア3(BSP)
			offset:2	offset:3
OS1の論理APIC ID	1	2	4	8
OS2の論理APIC ID	2	4	4(2^offset)	8
OS3の論理APIC ID	2	4	8	8(2^offset)

＜従来のMintの対処＞

(1) コアIDの重複は未対処

(2) 各OSが占有するコアの論理APIC IDのみ重複を回避

➡ OS間でコアの移譲を行う場合に問題が発生

(A) OS間でコアのやり取りを行うため，コアIDの重複が問題

(B) 新たに占有したコアに関して論理APIC IDが重複

OS間でコアを一意に判別する方式が必要

問題点

＜コア管理機能を実現する際の問題点＞

(問題1) ローカルタイマ割り当ての重複

(問題2) 各種識別ID の重複

(問題3) OS 間にまたがるコア管理機構の欠如

OS間にまたがるコア管理機構の欠如

OS間でのコアの移譲には他のOSが占有するコアの情報が必要

＜Linuxにおけるコアの管理＞

(1) コアIDを用いて各コアを識別，管理

(2) CPUホットプラグの際にもコアIDを使用

⇨OS間でコアをやり取りをすることを当然想定していない

＜Mintにおけるコアの管理＞

(1) コアの割り当ては起動時に静的に決定

(2) 各OSは自身の占有するコアのみコアIDで管理

⇨OS 間でコアの利用状況を統一的に管理していない

⇩

OS間にまたがるコアの利用状況管理の枠組みが必要

目次

(1) はじめに
(2) Mintの特徴と目的
(3) 問題点
(4) 課題
    (A) HPETの割り当て方式の変更
    (B) OS間でのコアの識別方法
    (C) コア利用状況管理
(5) 実現
(6) 評価
(7) おわりに

課題

＜コア管理機能を実現する際の問題点＞

(問題1) ローカルタイマ割り当ての重複

(問題2) 各種識別ID の重複

(問題3) OS 間にまたがるコア管理機構の欠如

＜コア管理機能を実現する際の課題＞

(課題1) HPET の割り当て方式の変更

(課題2) OS 間でのコア識別方式の導入

(課題3) コア利用状況管理機能の導入

課題

＜コア管理機能を実現する際の問題点＞

(問題1) ローカルタイマ割り当ての重複

(問題2) 各種識別ID の重複

(問題3) OS 間にまたがるコア管理機構の欠如

＜コア管理機能を実現する際の課題＞

(課題1) HPET の割り当て方式の変更

(課題2) OS 間でのコア識別方式の導入

(課題3) コア利用状況管理機能の導入

HPETの割り当て方式の変更

＜(問題1)の対処案＞

各OSがOS間通信を利用してHPETの利用状況を他のOSに伝搬

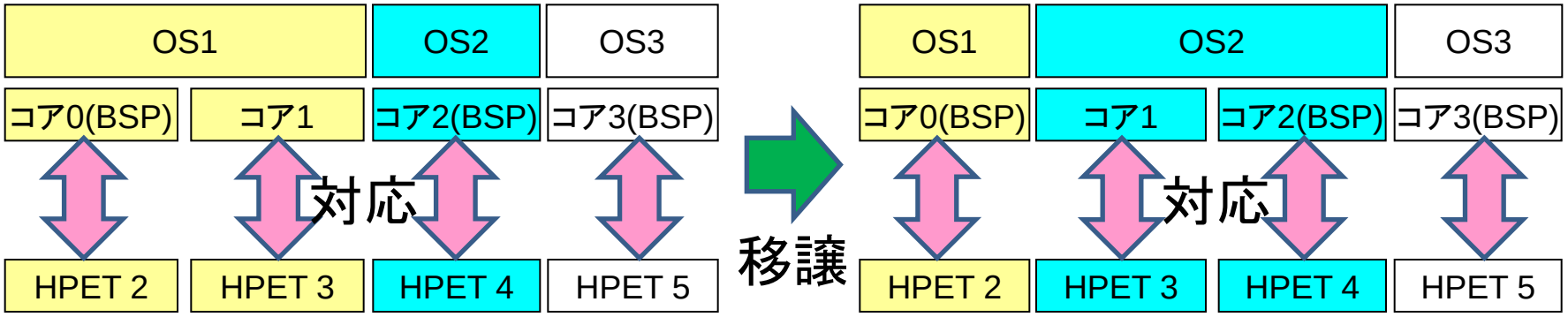
➡ OS間の独立性が低下

各コアにHPETを割り当てるルールを統一

＜新規HPET割り当て方式＞

(1) LAPIC ID とHPET を1 対1 で対応付け

(2) 各コアは自身の持つLAPIC ID に対応するHPETを利用



課題

＜コア管理機能を実現する際の問題点＞

(問題1) ローカルタイマ割り当ての重複

(問題2) 各種識別ID の重複

(問題3) OS 間にまたがるコア管理機構の欠如

＜コア管理機能を実現する際の課題＞

(課題1) HPET の割り当て方式の変更

(課題2) OS 間でのコア識別方式の導入

(課題3) コア利用状況管理機能の導入

OS間でのコア識別

＜(対処案1) コアID統一方式＞

コアID 決定規則を変更し全OSで同じコアに同じコアIDを付加

(利点) Linuxの枠組み内でOS間でのコアの識別が可能

(欠点) BSPにコアID 0以外を割り当てる必要

コアID 0をBSPとする箇所はカーネル内に随所に存在

⇨ カーネルの改変箇所が多く，実装工数が高い

＜(対処案2) 論理APIC ID の算出規則変更方式＞

論理APIC ID への算出規則を変更し，論理APIC ID の重複を防ぐ

(利点) OSの改変箇所を局所化可能

(欠点) コアIDは未統一

各OSはそのままコアIDでコアを管理

⇨ OS間でのコアの識別には別の識別方法が必要

OS間でのコアの識別方式

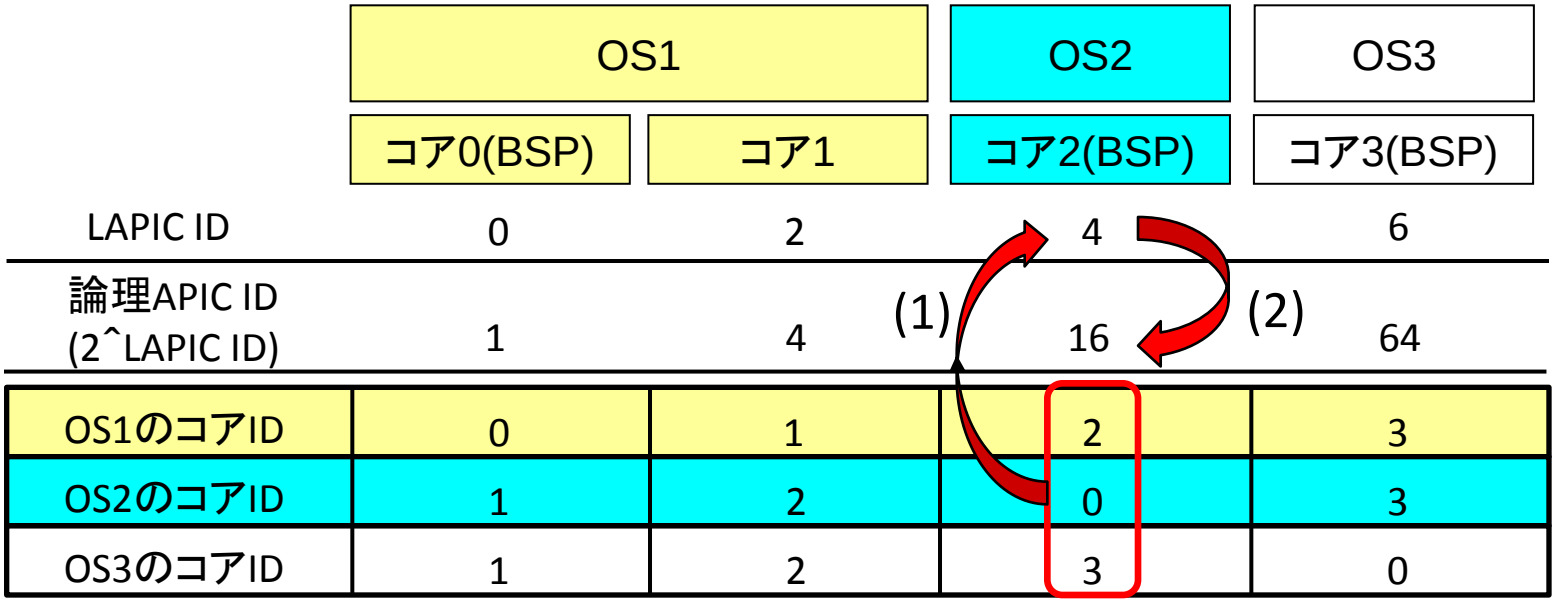
＜新規論理APIC ID算出規則＞

OS間で一意なLAPIC IDから論理APIC IDを算出するように変更

＜算出手順＞

(1) コアIDをLAPIC IDに変換

(2) LAPIC IDから論理APIC IDを算出



課題

＜コア管理機能を実現する際の問題点＞

(問題1) ローカルタイマ割り当ての重複

(問題2) 各種識別ID の重複

(問題3) OS 間にまたがるコア管理機構の欠如

＜コア管理機能を実現する際の課題＞

(課題1) HPET の割り当て方式の変更

(課題2) OS 間でのコア識別方式の導入

(課題3) コア利用状況管理機能の導入

コア利用状況管理

統一のコア識別子を用いてコアの利用状況を管理する機能を導入

＜コアの管理方式＞

(方式1)特定のOS(以下，管理OS)によるコアの集中管理

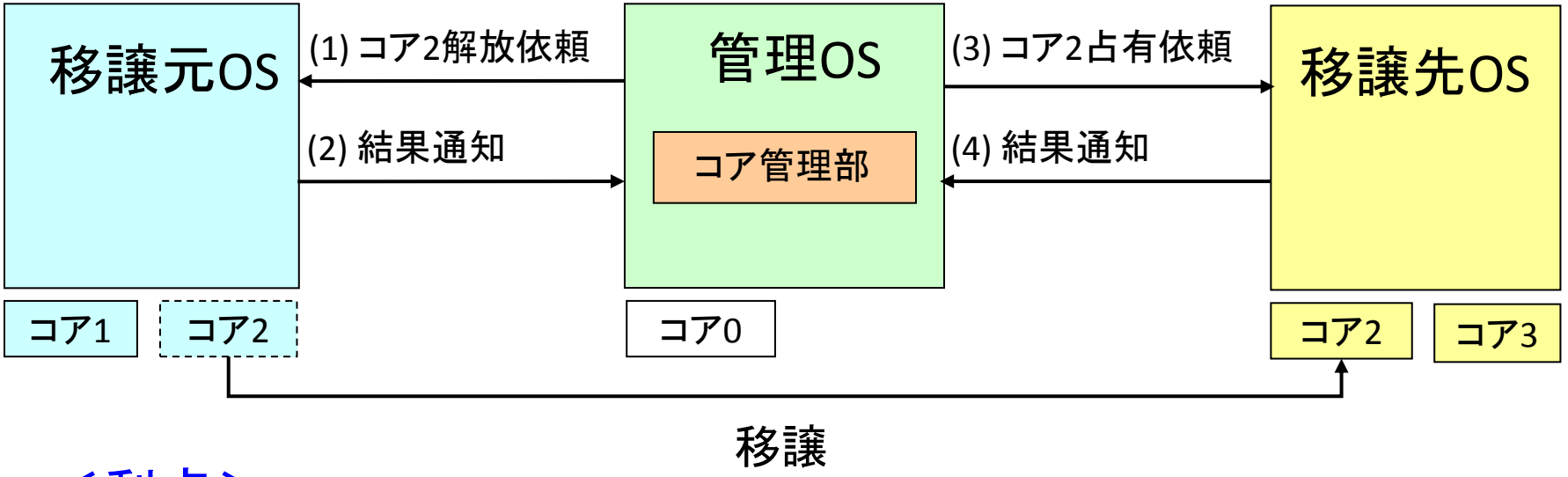
(1) 管理OSのみがコアの占有状況を管理

(方式2)各OSによるコアの分散管理

(1) 各OSから操作できる共有メモリ領域を作成

(2) 各OSが共有メモリ領域を用いてコア占有状況を管理

特定のOSによるコアの集中管理



＜利点＞

管理OS以外のOSはコアの把握が不要

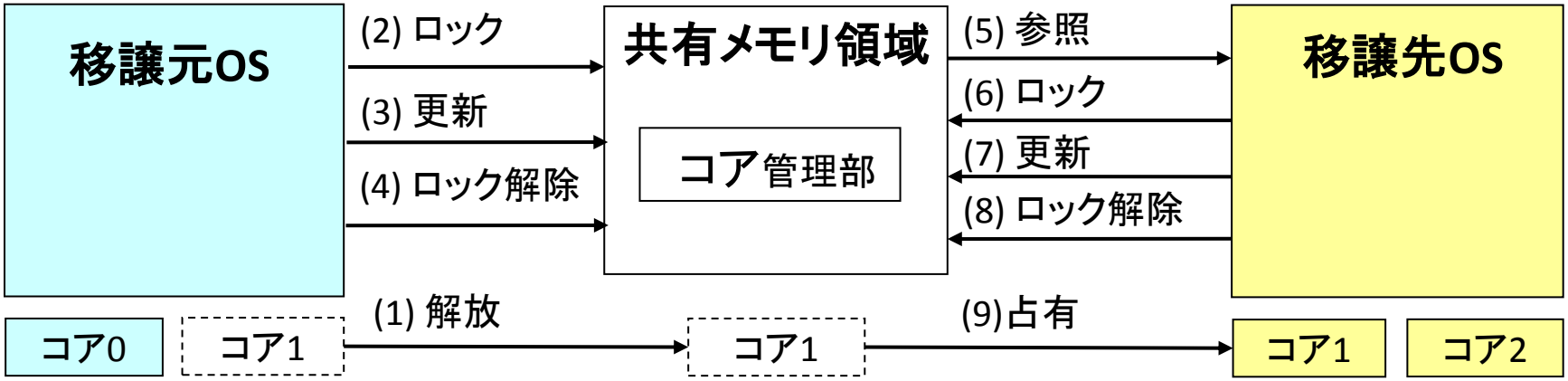
⇨ 共有メモリ領域の作成，OS間での排他制御が不要

＜欠点＞

OS間に依存関係が存在

⇨ 単一障害点が存在

各OSによるコアの分散管理



＜利点＞

OS間に依存関係が存在しない

⇨単一障害点が存在しない

＜欠点＞

OS間での排他制御が必要

コア管理方法の比較

管理方式	利点	欠点
管理OSによるコアの集中管理	(1)管理OS以外のOSはコアの把握が不要	(1) OS間に依存関係が存在
各OSによるコアの分散管理	(1)他のOSに依存しない	(1) 全OSでコアの把握が必要 (2) OS間で排他制御が必要

(方式1)はOS間で依存関係が存在

⇨ 1つのOSの不具合が全体に影響を与える

(方式2)はOS間での排他制御が必要

⇨ 排他制御のオーバヘッドは大きくならないと想定

∵コア管理部の排他制御の頻度は低い

⇩

(方式2)を採用

コア識別子変換機能

各OSはOS間で異なるコアIDを用いてコアを管理

⇨ OS間でのコアの管理にはコアIDの統一か別の識別方法が必要

(対処案1)全てのOSで同じコアIDを用いるように改変

➡ 実装工数が問題

⇩

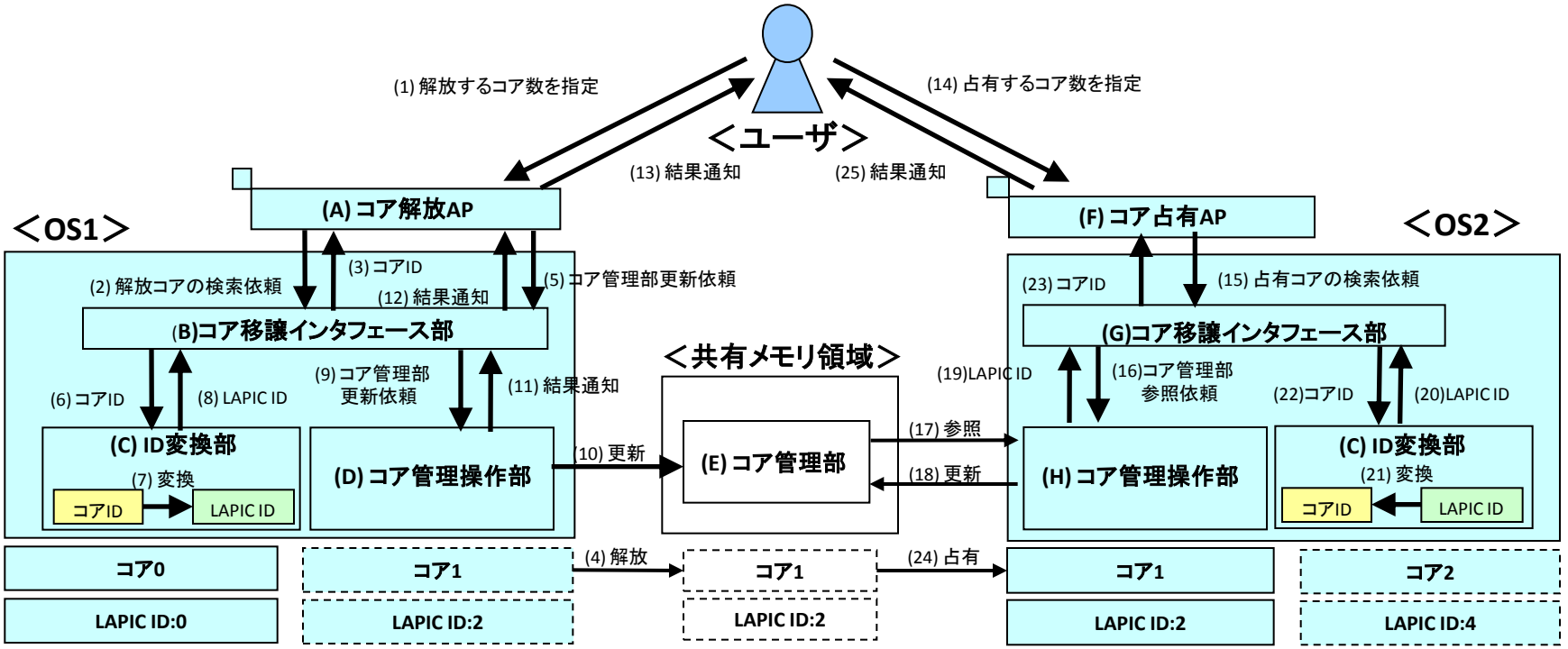
(対処案2)全てのOSで一意に設定されるコア識別子を用意し，
コアIDの代わりにコア移譲の際のみ使用

⇩

コア識別子としてLAPIC IDを利用

目次

実現



＜ユーザによるコア移譲手順＞

(1) 移譲元OS(OS1)でコア解放APを実行

(2) 移譲先OS(OS2)でコア占有APを実行

⇨ 移譲元OSと移譲先OSはそれぞれコアの解放と占有を実行

目次

評価

コア管理機能の実行時間について評価

＜評価方法＞

コア管理機能によるコアの移譲にかかる時間を測定

(1) コアの解放と占有について処理時間を10回測定

(2) 解放処理と占有処理の平均処理時間と合計処理時間を算出

評価環境

Mint	OS	Fedora14(Linux Kernel 2.6.39)
	CPU	Intel(R) Core(TM) i7 CPU 870 @ 2.93GHz
	メモリ	1GB×2

コア管理機能の実行時間

コアの解放処理時間:18.23ms
コアの占有処理時間:112.90ms
合計処理時間:131.13ms
} CPUホットプラグ実行時間と同程度

＜コア移譲の用途＞

コア移譲の実行時間は131.13ms程度

プロセスのスケジューリングを契機としてコアの移譲を行い，
新たに占有したコアにプロセスを割り当てることは難しい

おわりに

コア管理機能を実現

⇨ OS間でのコアの移譲を実現

＜評価＞

(1)コアの移譲にかかる時間は131.13ms

⇨プロセスのスケジューリングを契機としたコア移譲は難しい

＜残された課題＞

(1) コア移譲の際の割り込み通知先の変更処理の実現

(2) CPUの使用率により，自動でコアの解放と占有を行う方式の検討