

Kexecによる コア別Linuxカーネルの起動方式

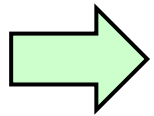
中原 大貴 乃村 能成 谷口 秀夫

岡山大学 大学院自然科学研究科

はじめに

<コアごとにLinuxカーネルを同時走行させる方式>

分割した物理メモリ領域を利用し、独立してカーネルを走行



仮想化によらずに複数のOSが同時に走行

<課題> 現状の起動方式の改善

<Kexec>

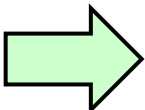
(1) Linuxカーネルの高速な再起動方式

(2) Linuxに標準搭載

Kexecの再起動機能は、課題とする起動方式に類似

<目的> Kexecを起動方式に応用したい

現状の起動方式

- (1) 物理メモリ領域を制限したカーネルイメージを**必要数作成**
- (2) ブートローダを用いて1番目のカーネルを起動
- (3) 2番目以降のカーネルについて
 - (A) 0x100000(1MB)番地以内に以下を配置:
 - (a) コア初期化処理部
 - (b) セットアップルーチン(リアルモードにおける初期化ルーチン)
 - (B) **起動対象のOSに割り当てた物理メモリ領域に**以下を配置:
 - (a) カーネル本体
 - (b) initrd
 - (C) 起動対象(未使用)のコアに**IPI(プロセッサ間割り込み)**送信
 IPIを受信したコアは(A)-(a)の領域から実行を開始

現状の起動方式の問題点

<問題1>

今後のLinuxのバージョンアップへの追従が困難

∴ 各カーネルは起動順序や自他の配置アドレスを意識した構造

 Linux自身が持つ起動や再起動機能との整合性の維持が困難

<問題2>

カーネル間の物理メモリ配分の変更にカーネルの再構築が必要

∴ カーネル構築時に利用メモリ領域を固定

目的

Kexecを後続のカーネルの起動に利用

Kexecとは, **Linuxカーネルの高速な再起動方式**

現状の起動方式とKexecの再起動機能には, 多くの類似点

➡ Kexecを改変し, **再起動処理の代わりに後続のカーネルの起動処理を実行**

<期待される効果>

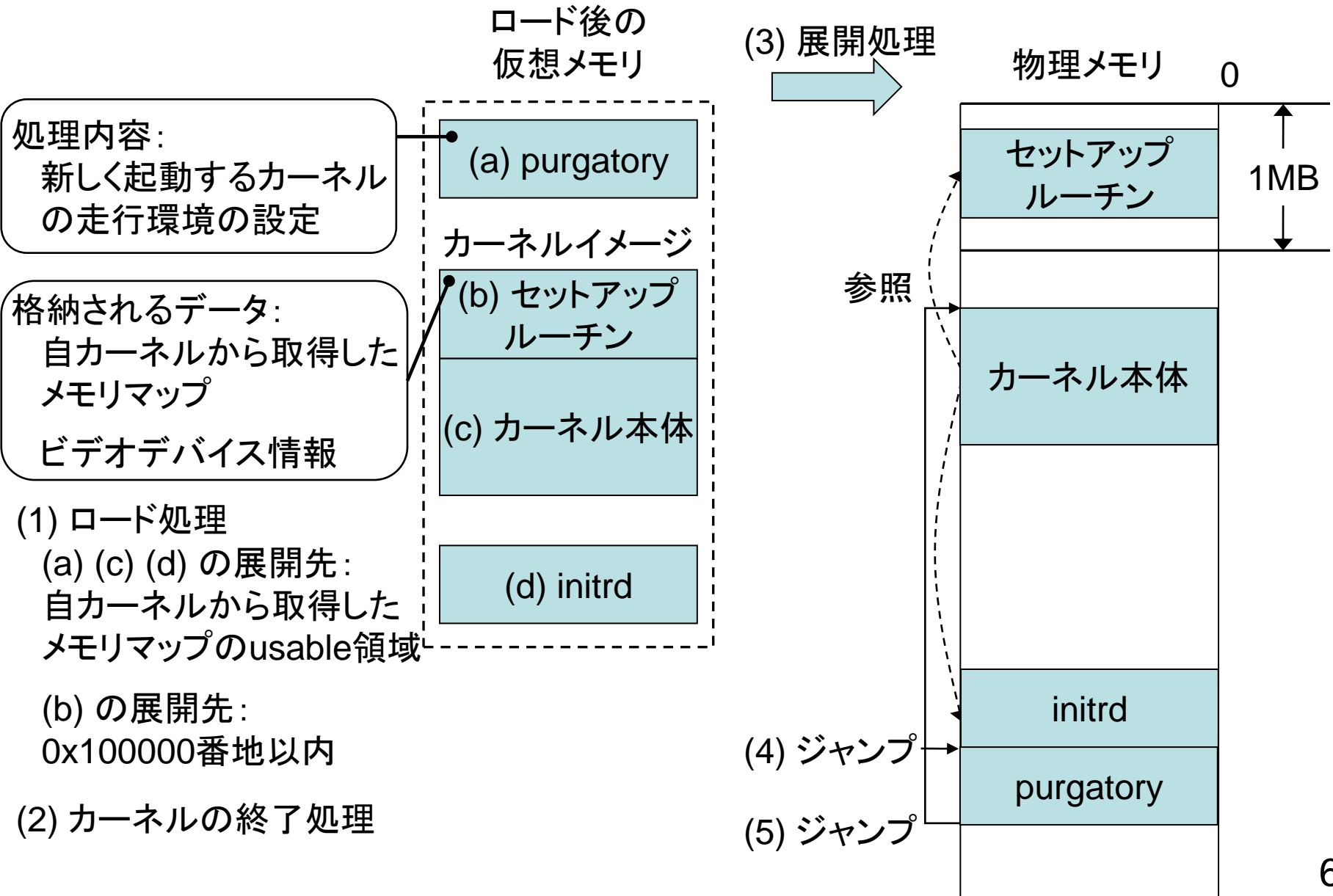
(1) 起動機能をLinuxカーネルに元々存在する機能で置き換え可能

➡ **問題1に対処可能**

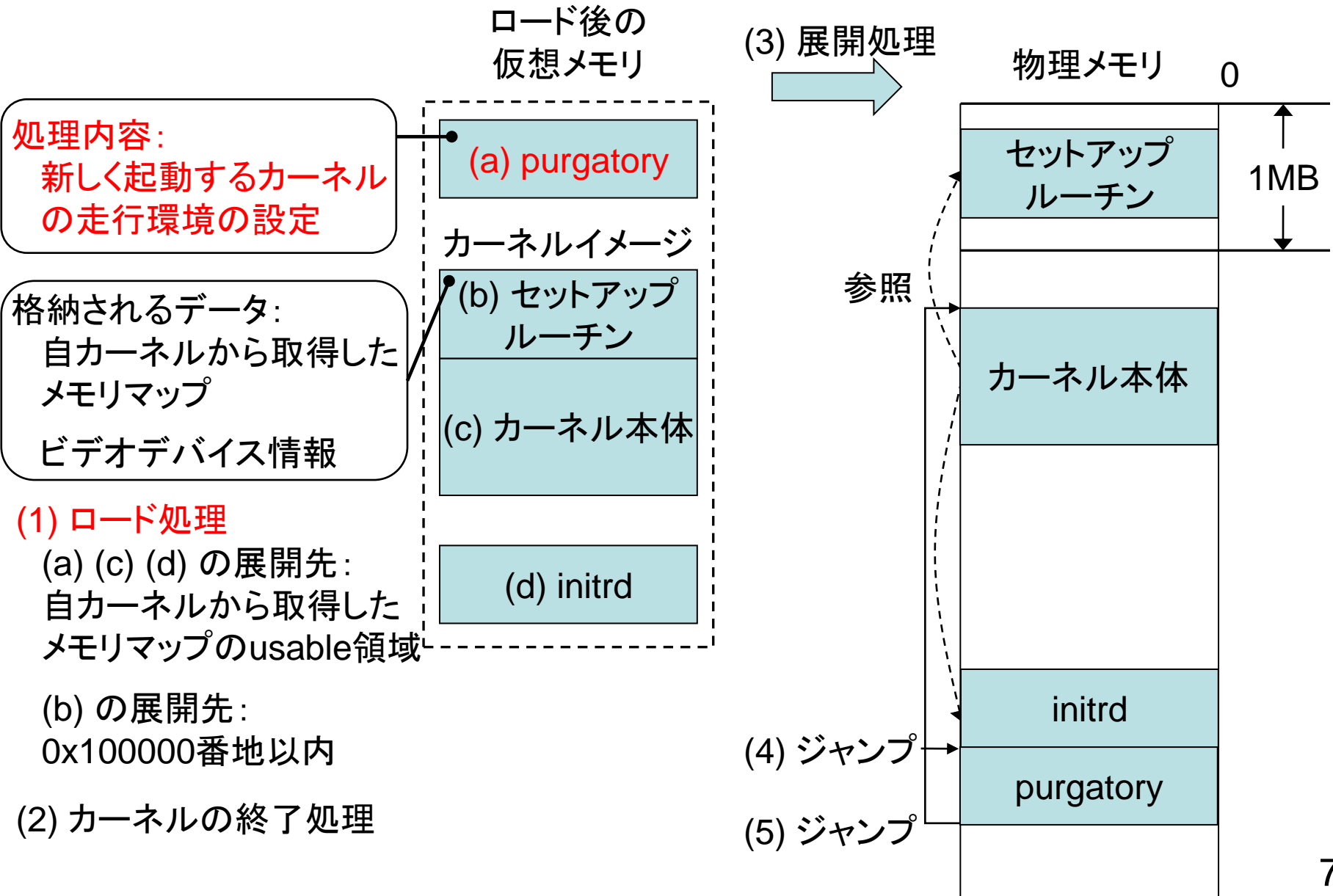
(2) 各カーネルの利用メモリ領域を起動時に任意に指定可能

➡ **問題2に対処可能**

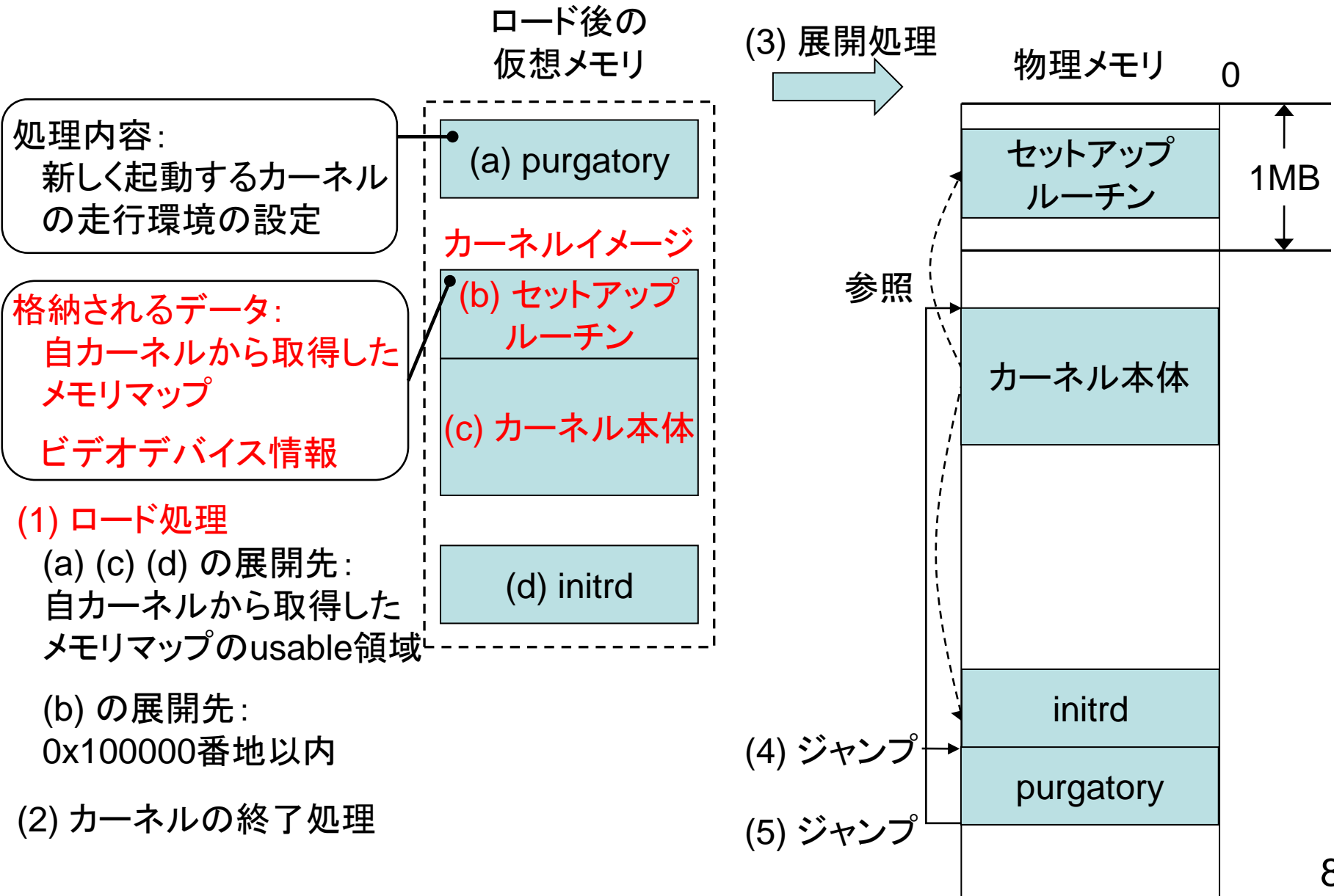
Kexecの処理流れ



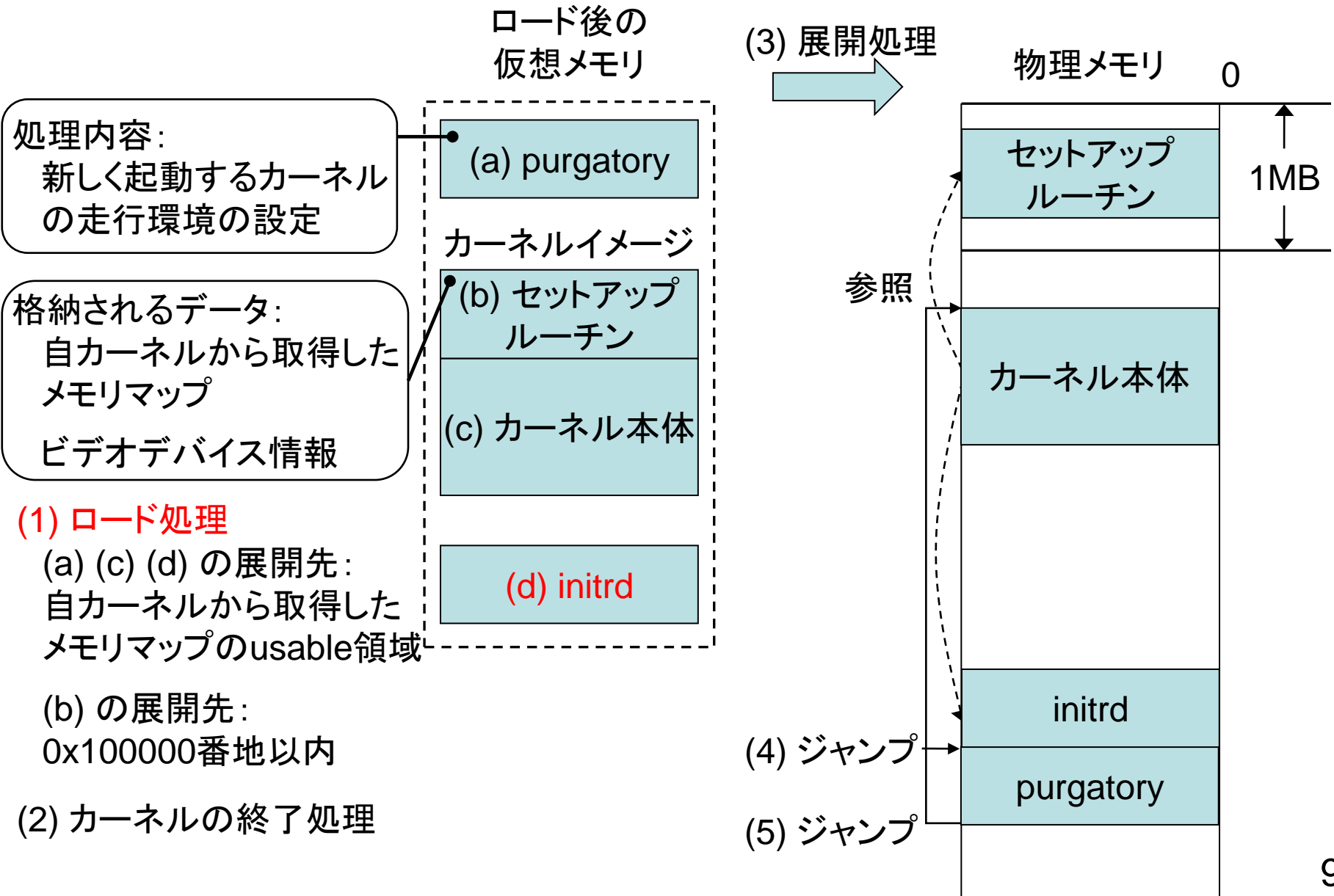
Kexecの処理流れ



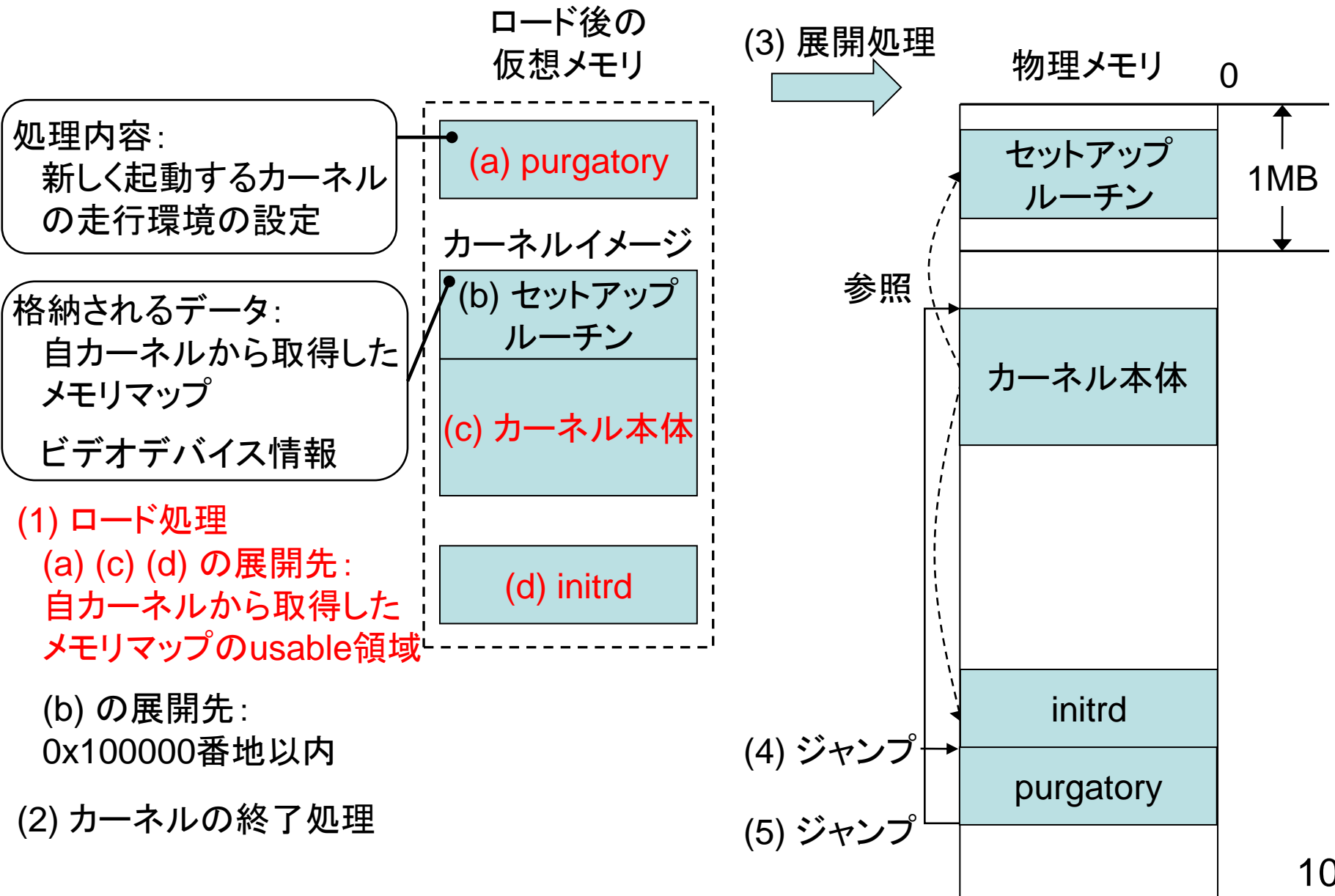
Kexecの処理流れ



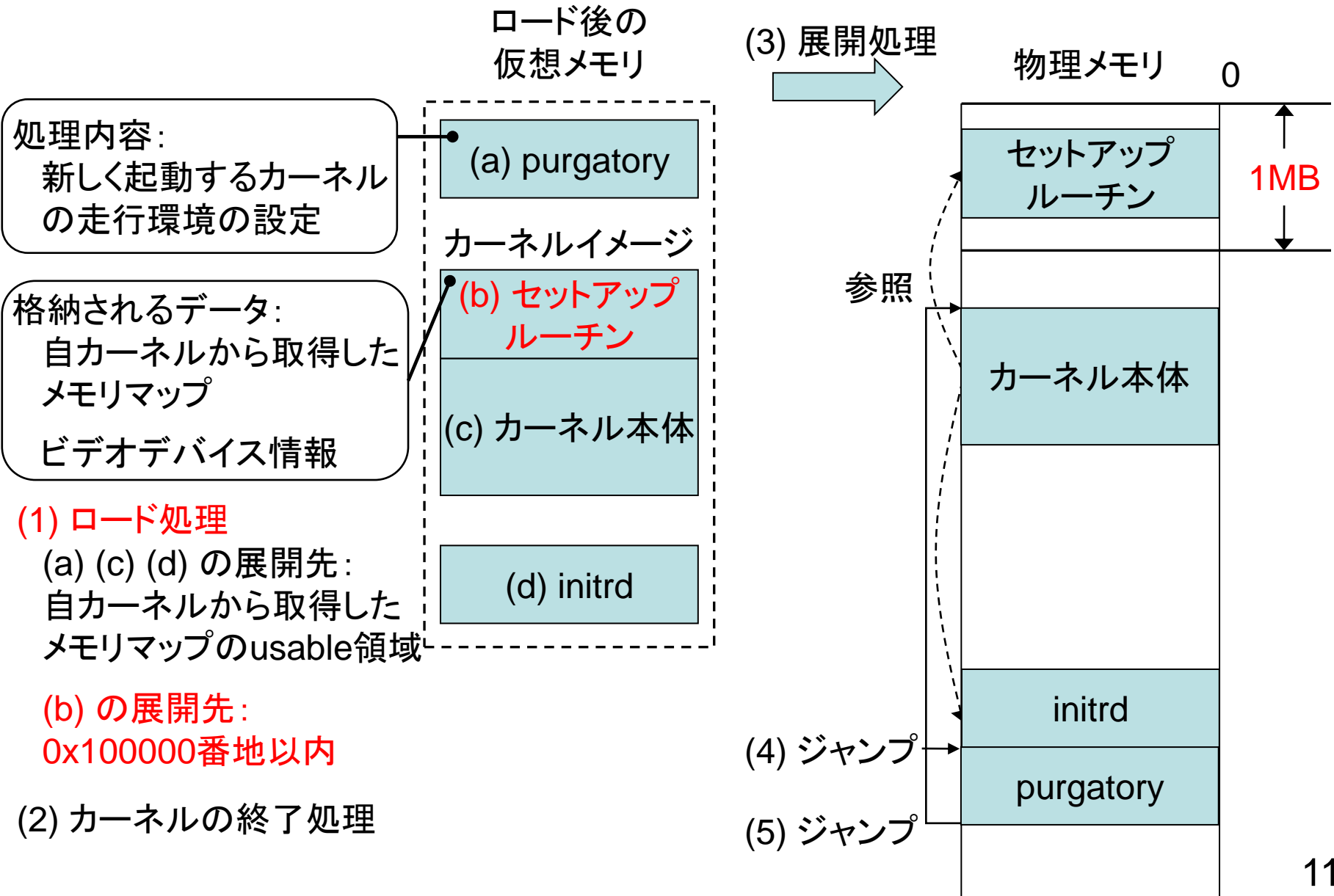
Kexecの処理流れ



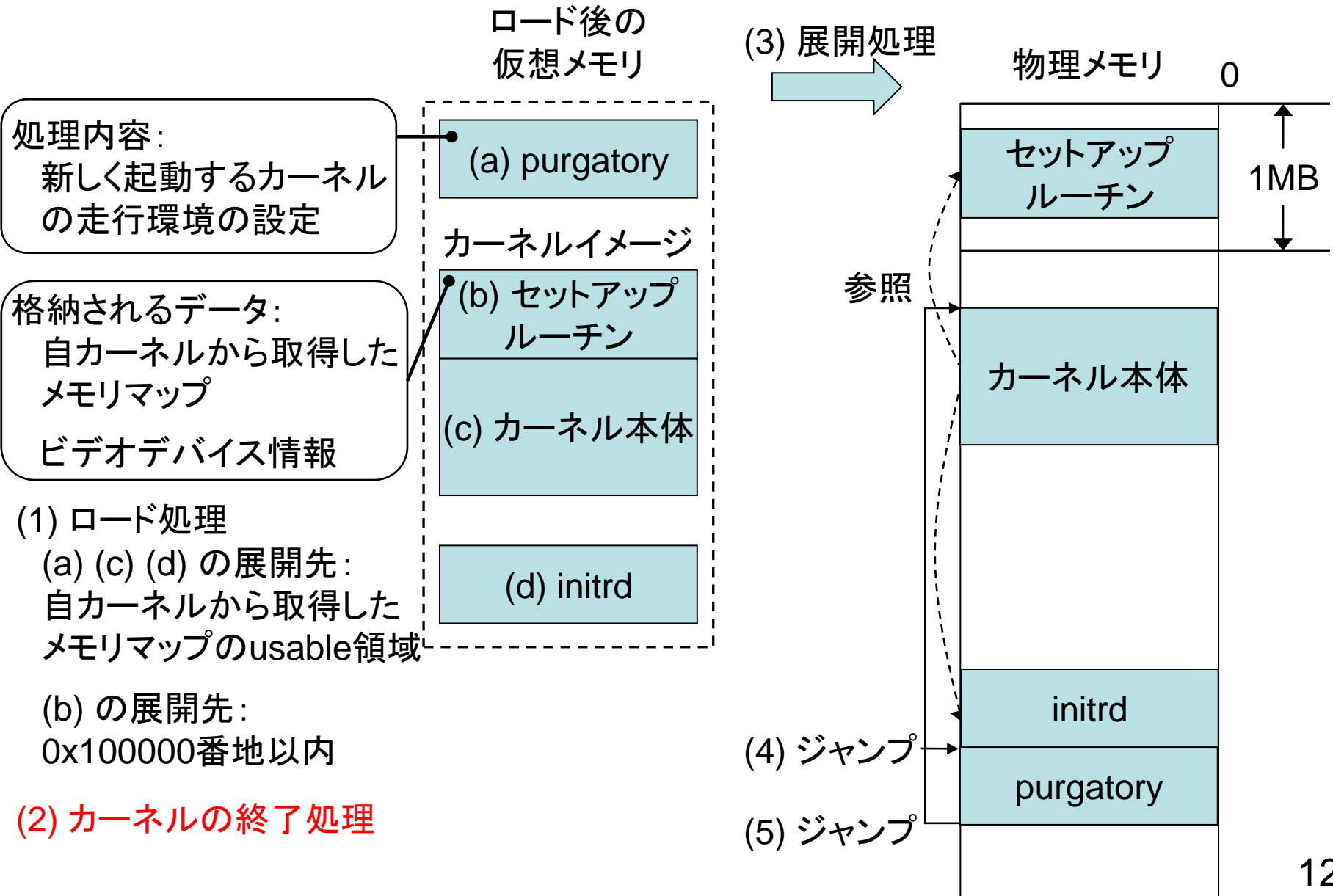
Kexecの処理流れ



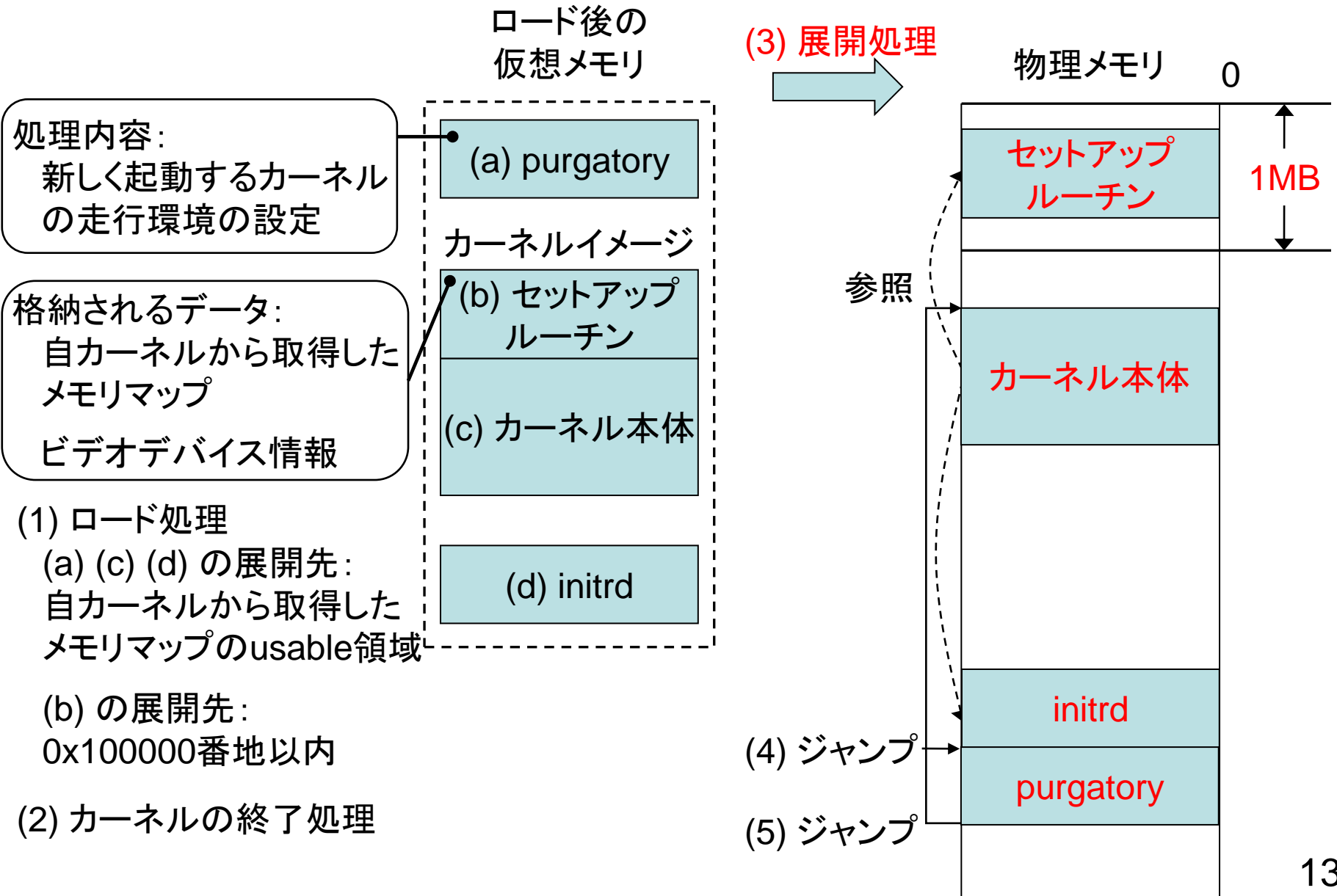
Kexecの処理流れ



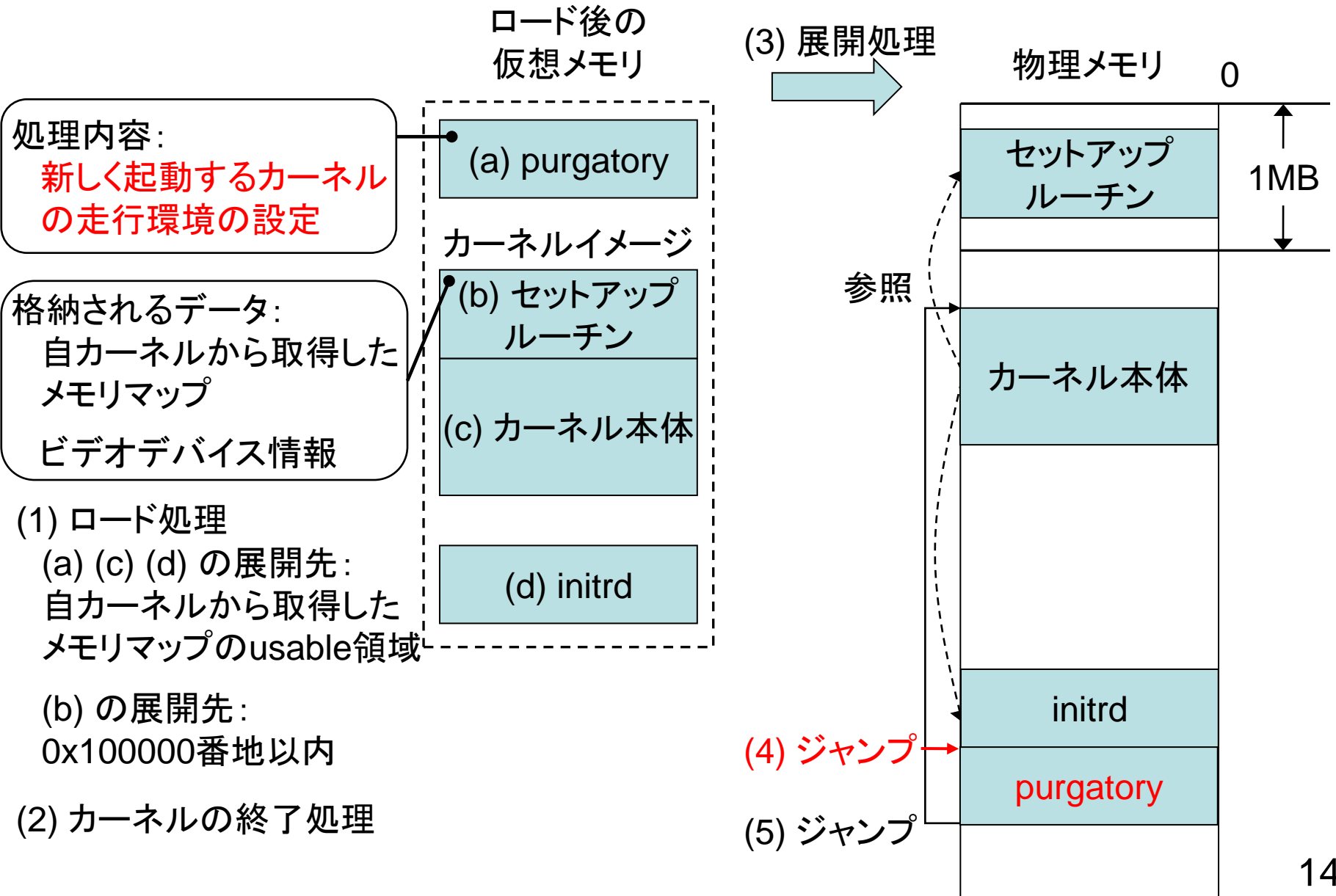
Kexecの処理流れ



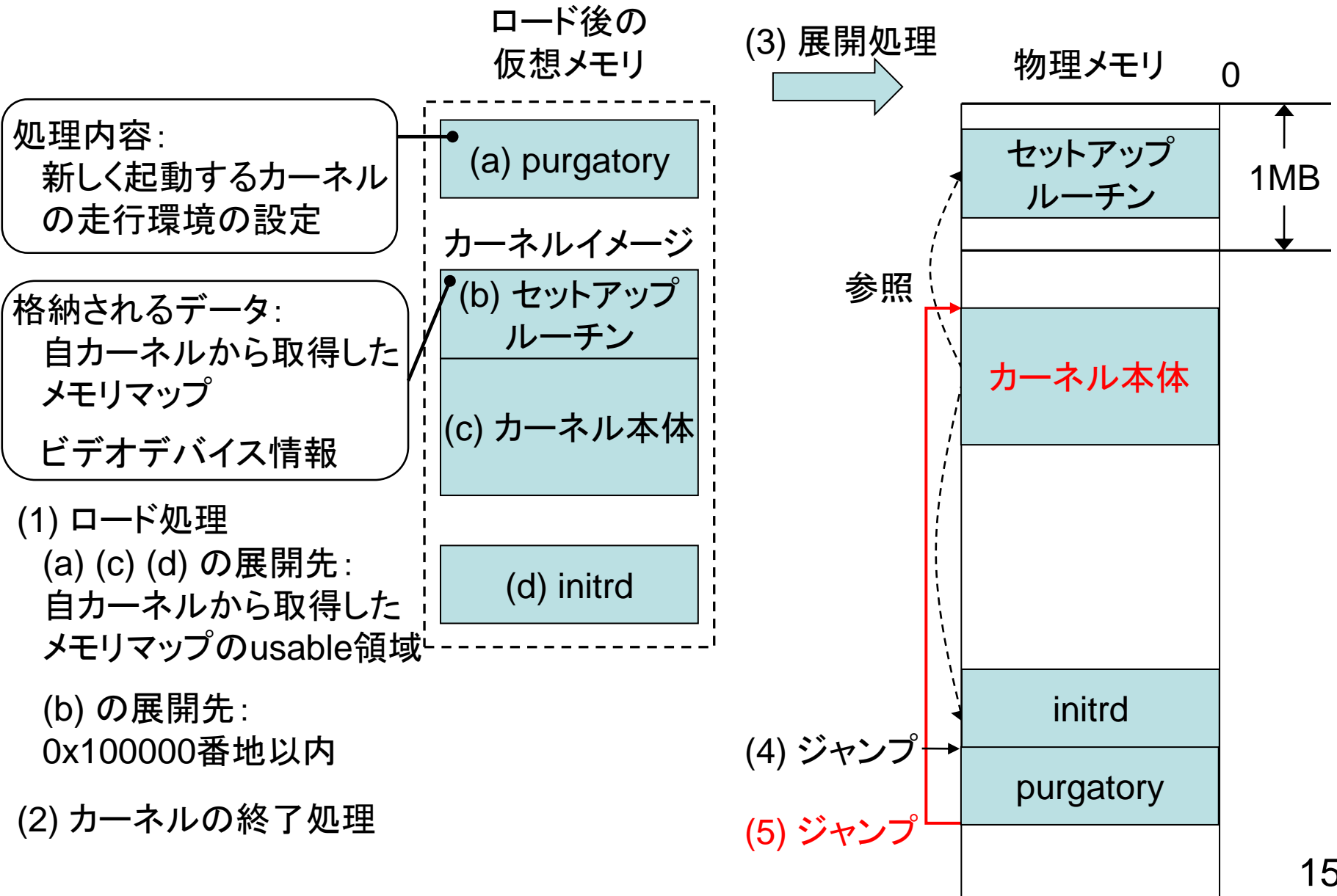
Kexecの処理流れ



Kexecの処理流れ



Kexecの処理流れ



課題

Kexecは、カーネルを再起動する機能

➡ Kexecを後続のカーネルの起動用に改変する必要

(課題1) コアの起動

再起動ではなく、別のコアを起動させる必要

(課題2) プロテクトモードへの切り替え

起動後のコアをリアルモードから切り替える必要

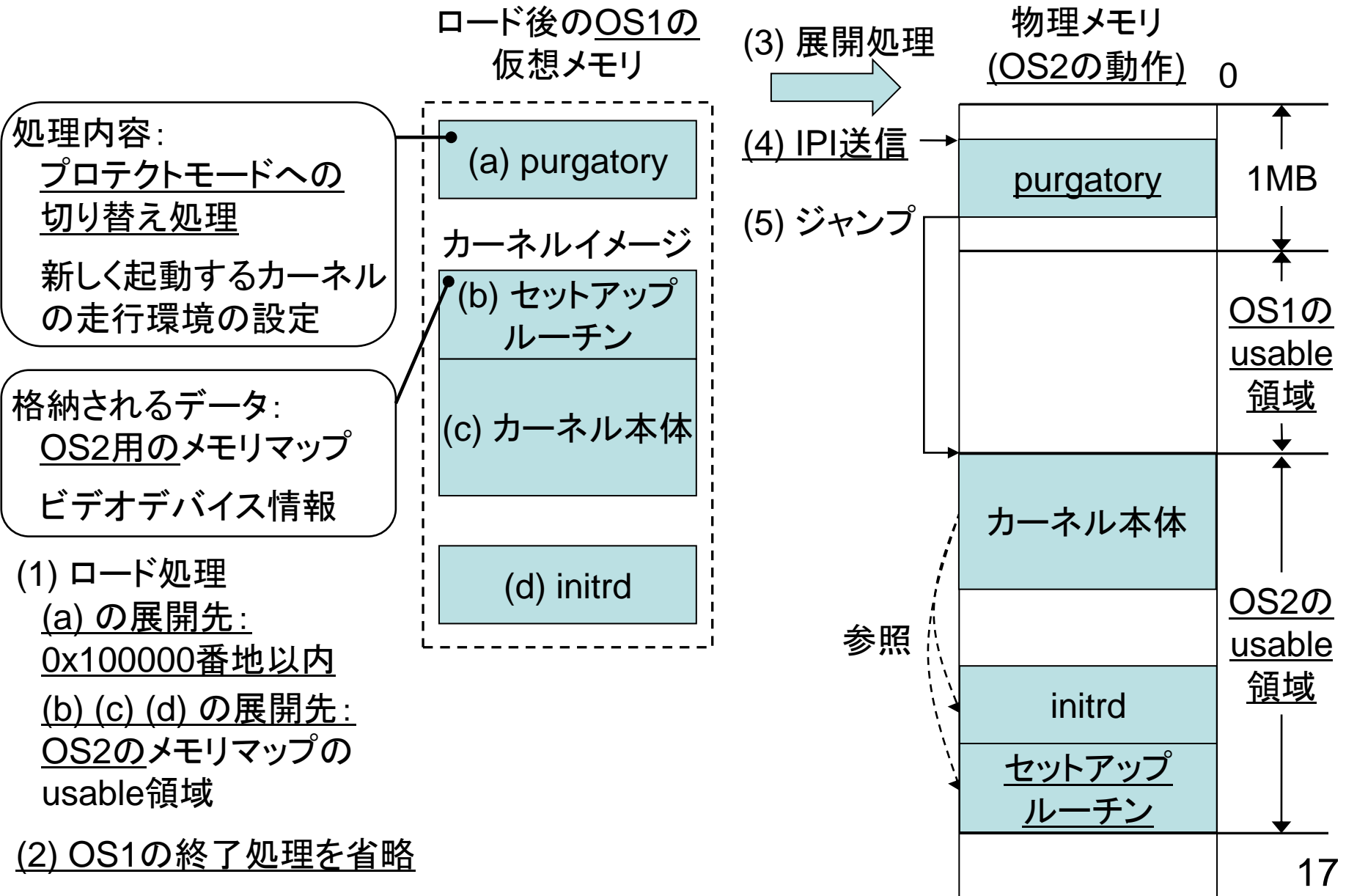
(課題3) 後続カーネル用のメモリマップの用意

後続カーネルにあわせた個別のメモリマップが必要

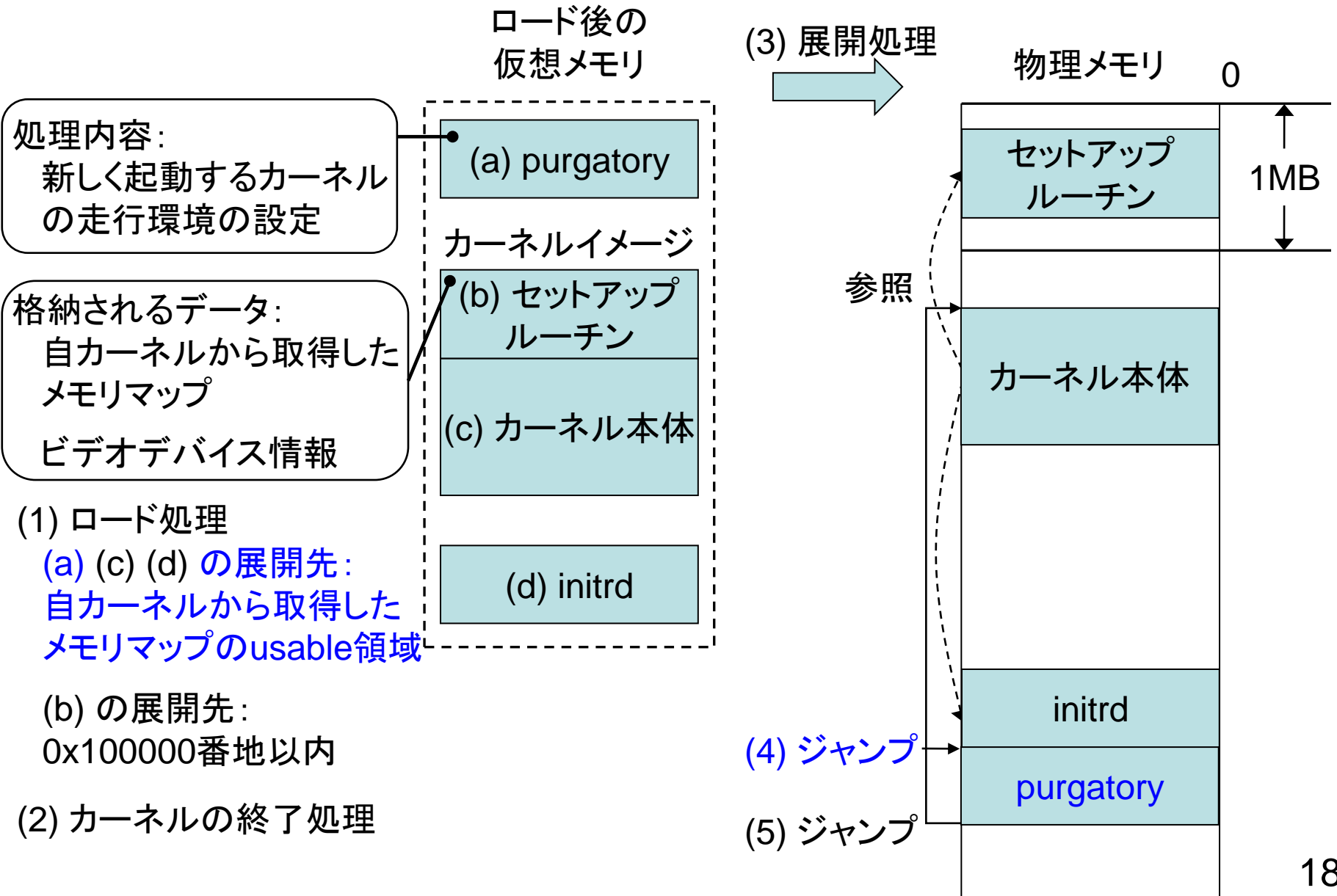
(課題4) 先行するカーネルの走行環境の保護

カーネルの終了処理を省略する必要

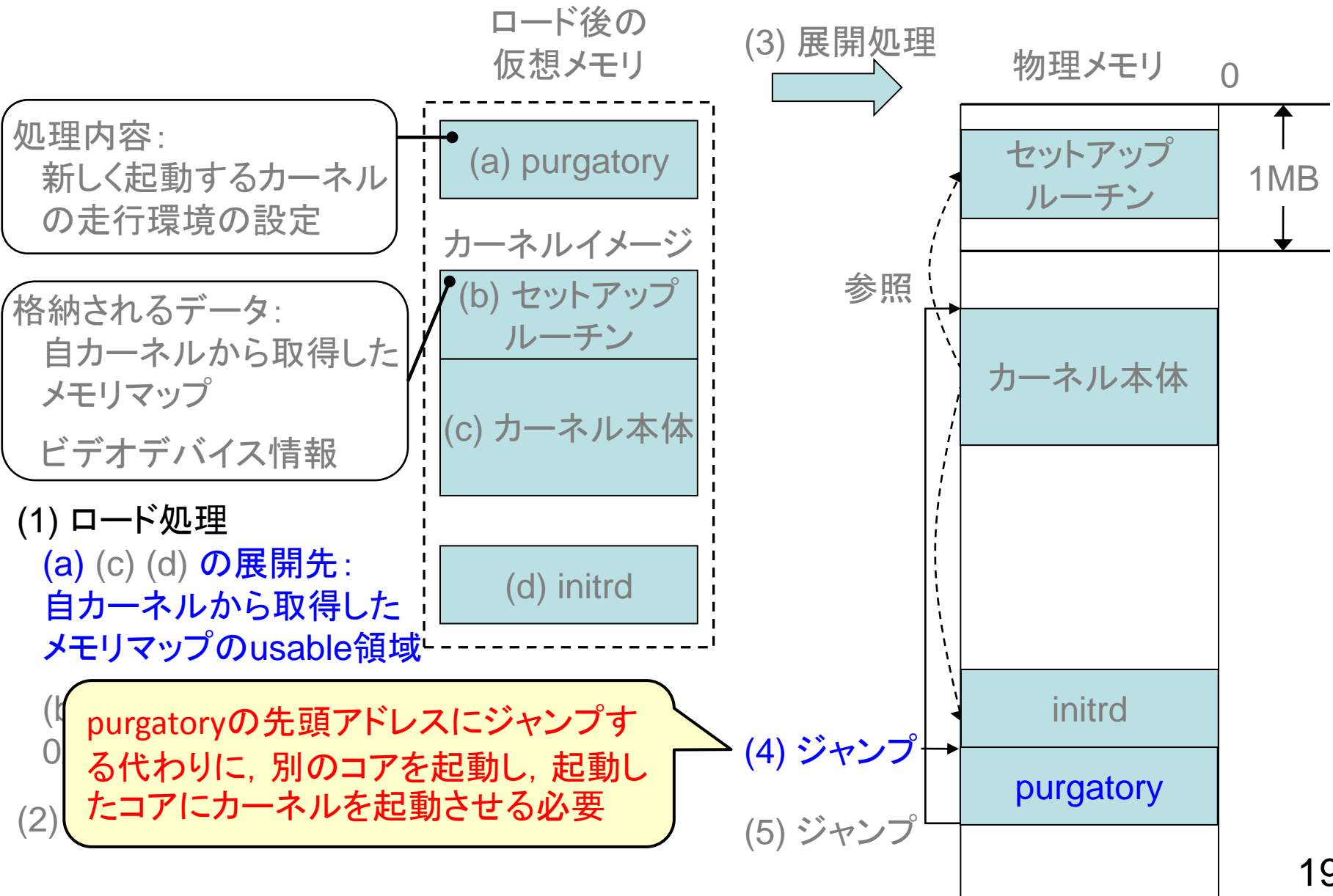
対処後の処理流れ



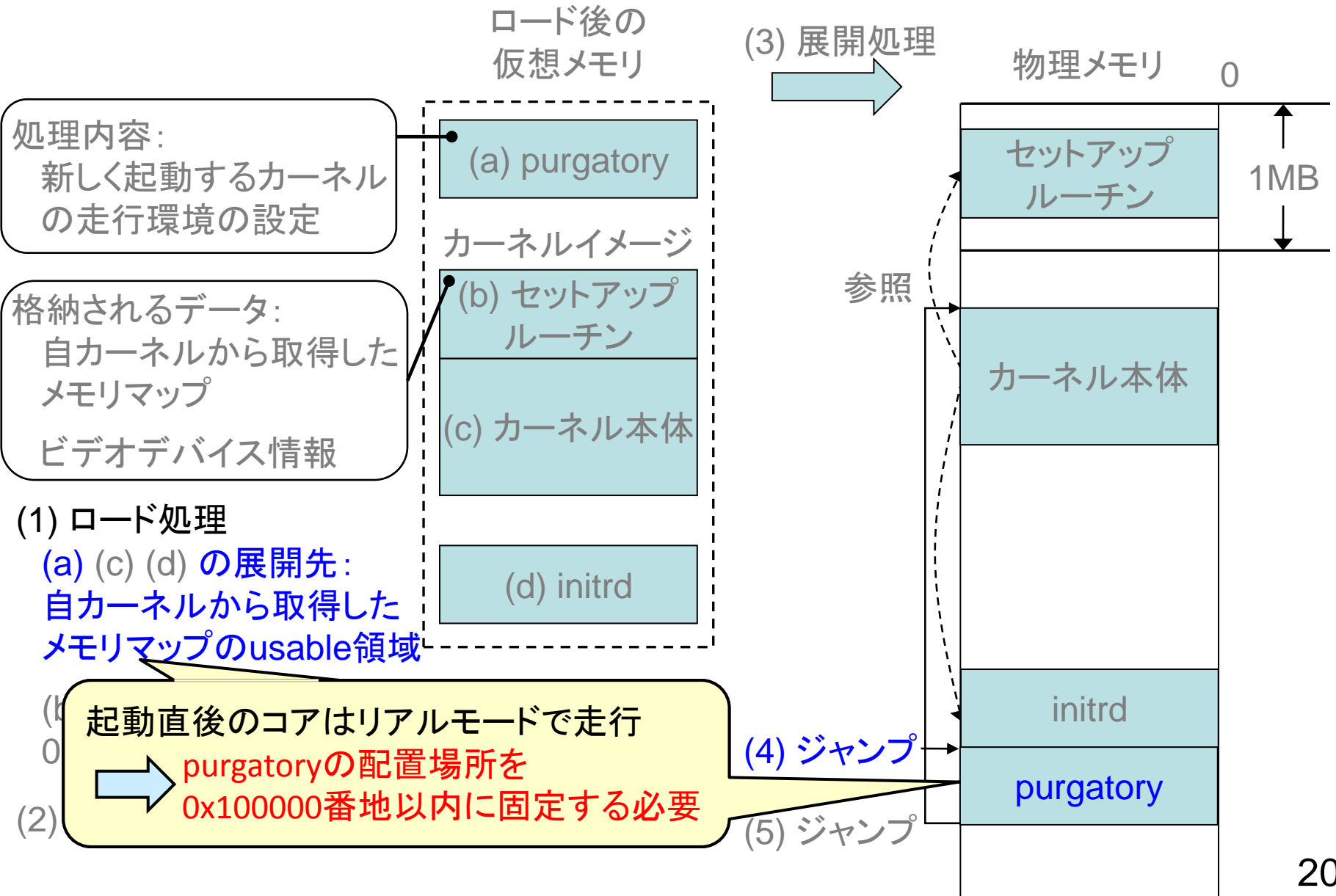
(課題1) コアの起動



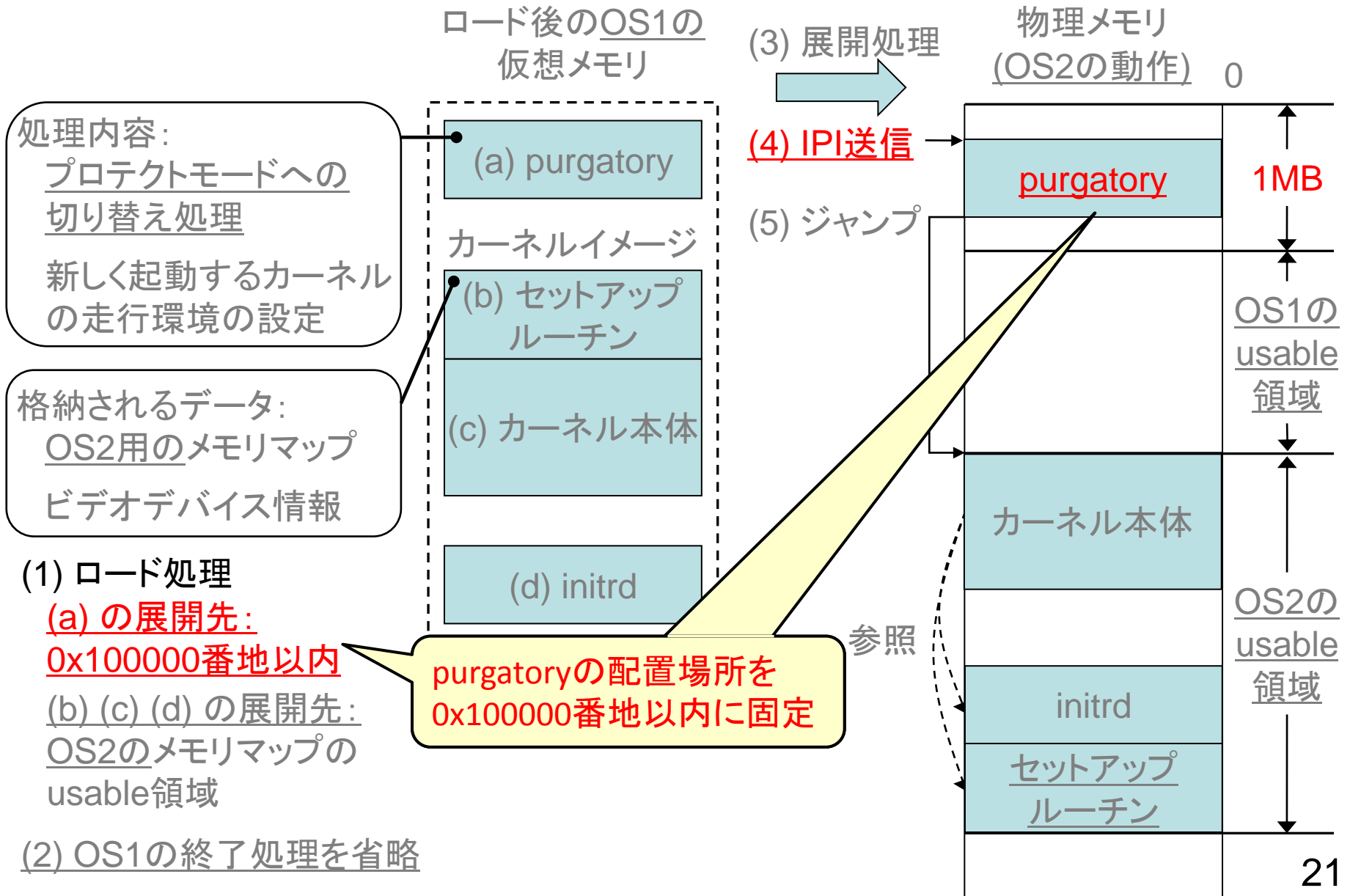
(課題1) コアの起動



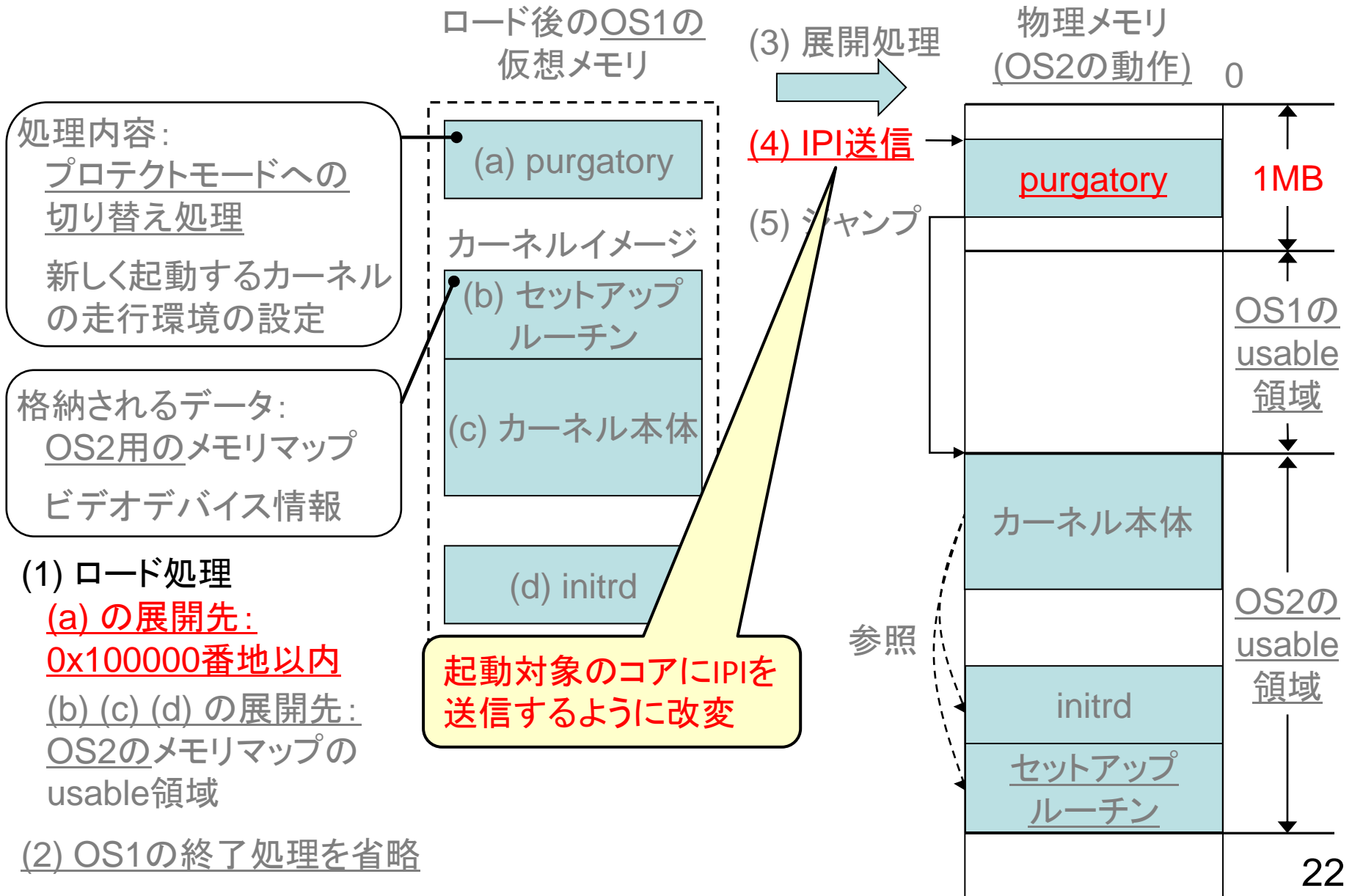
(課題1) コアの起動



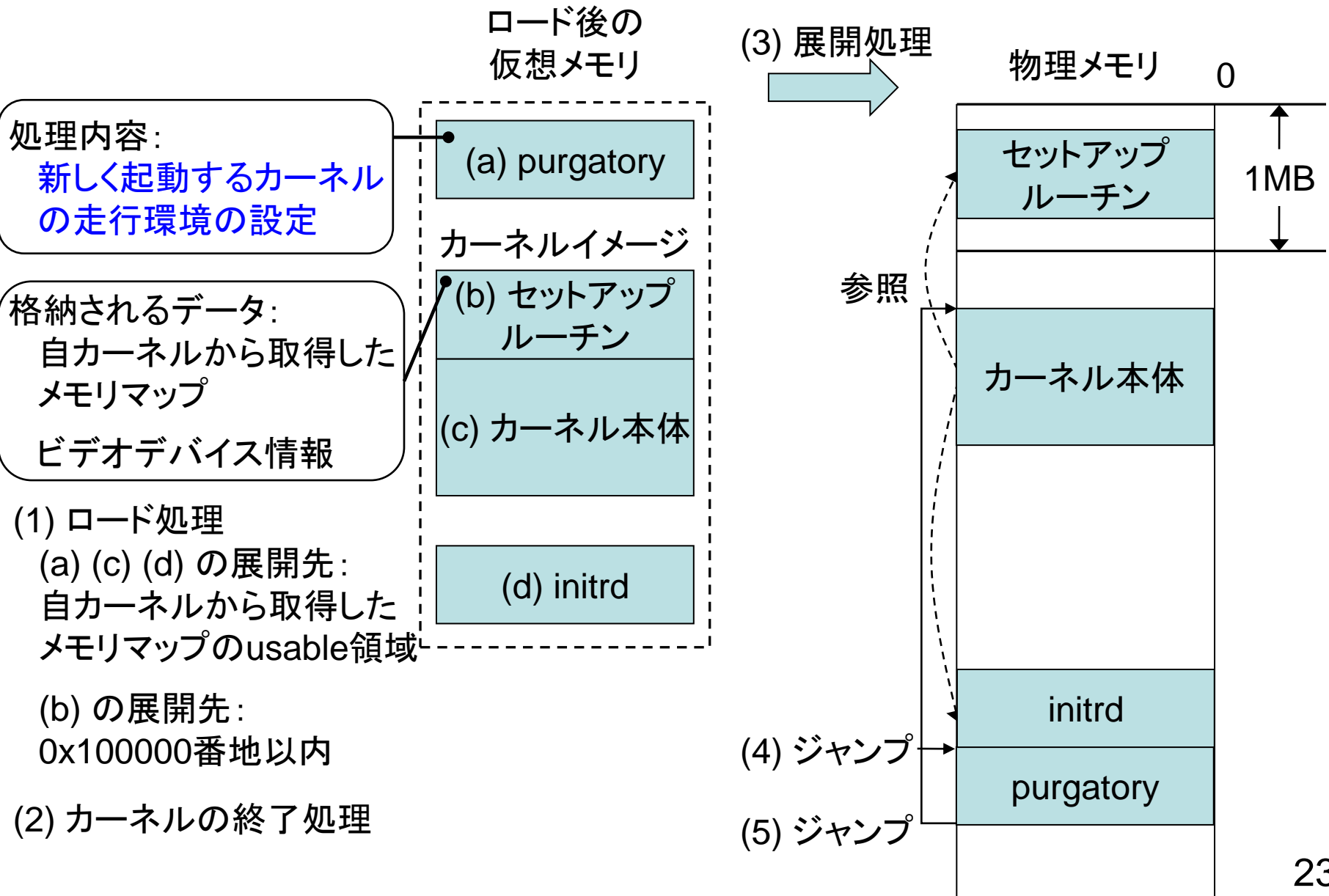
(対処1) IPIの送信によるコアの起動



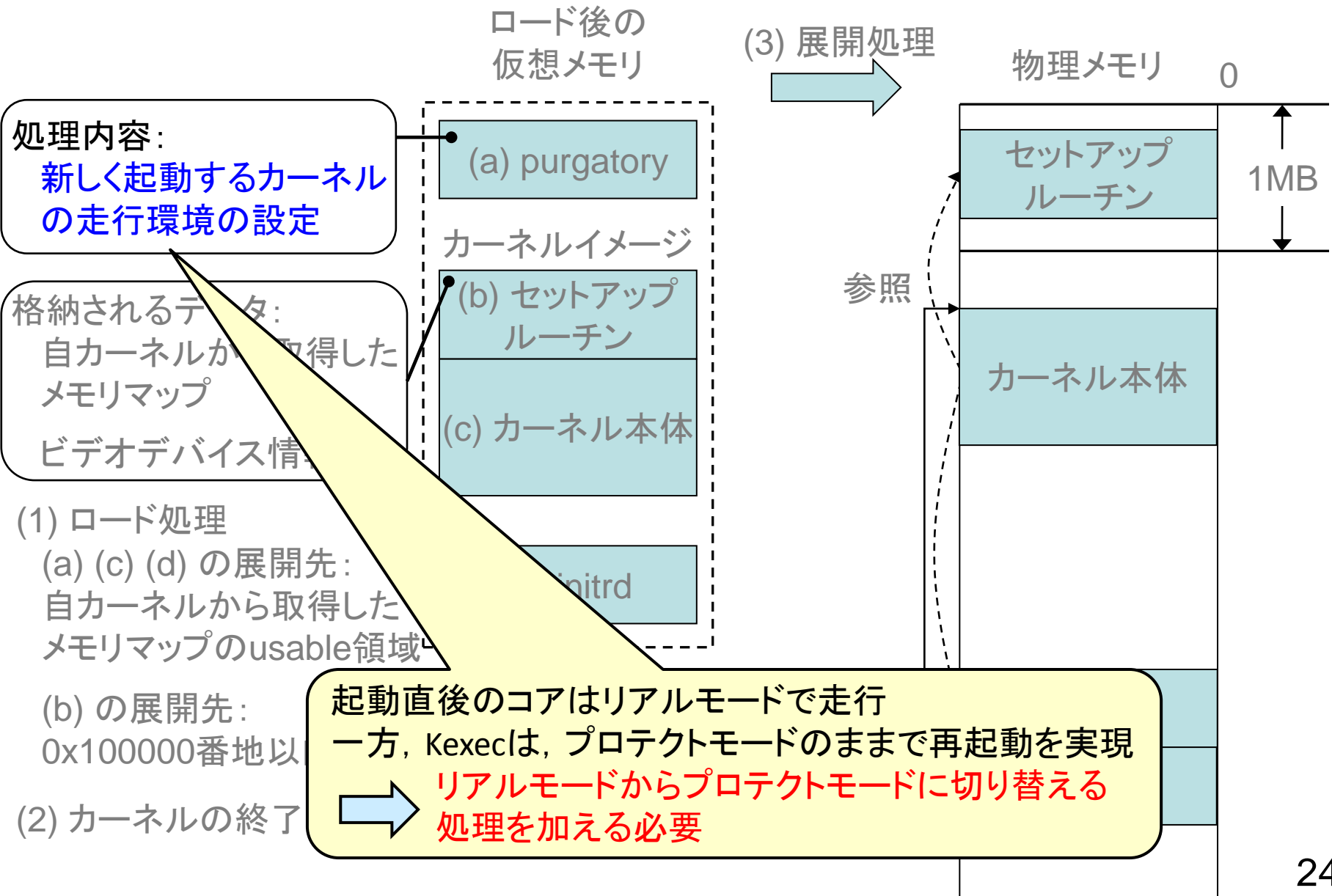
(対処1) IPIの送信によるコアの起動



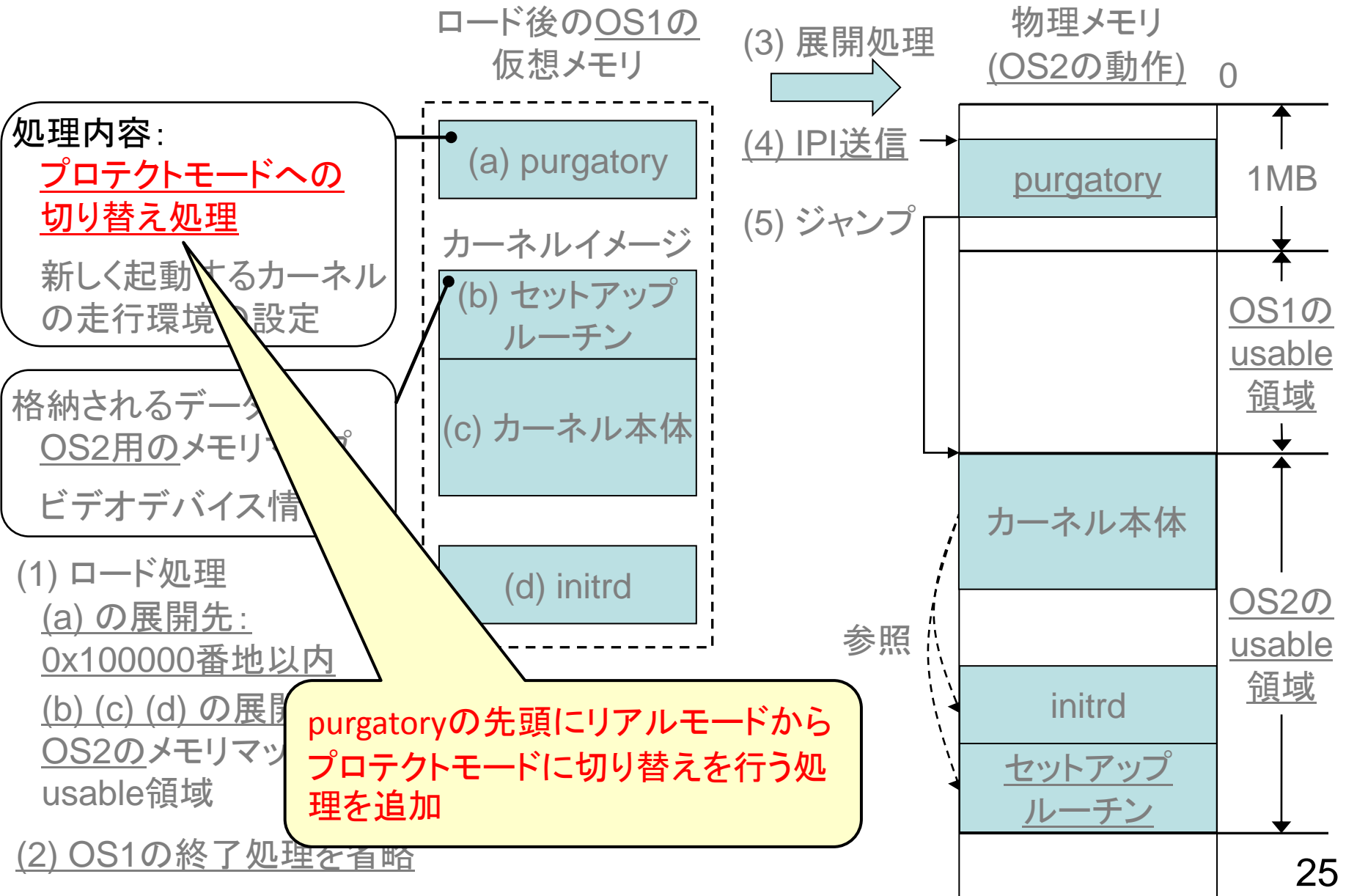
(課題2) プロテクトモードへの切り替え



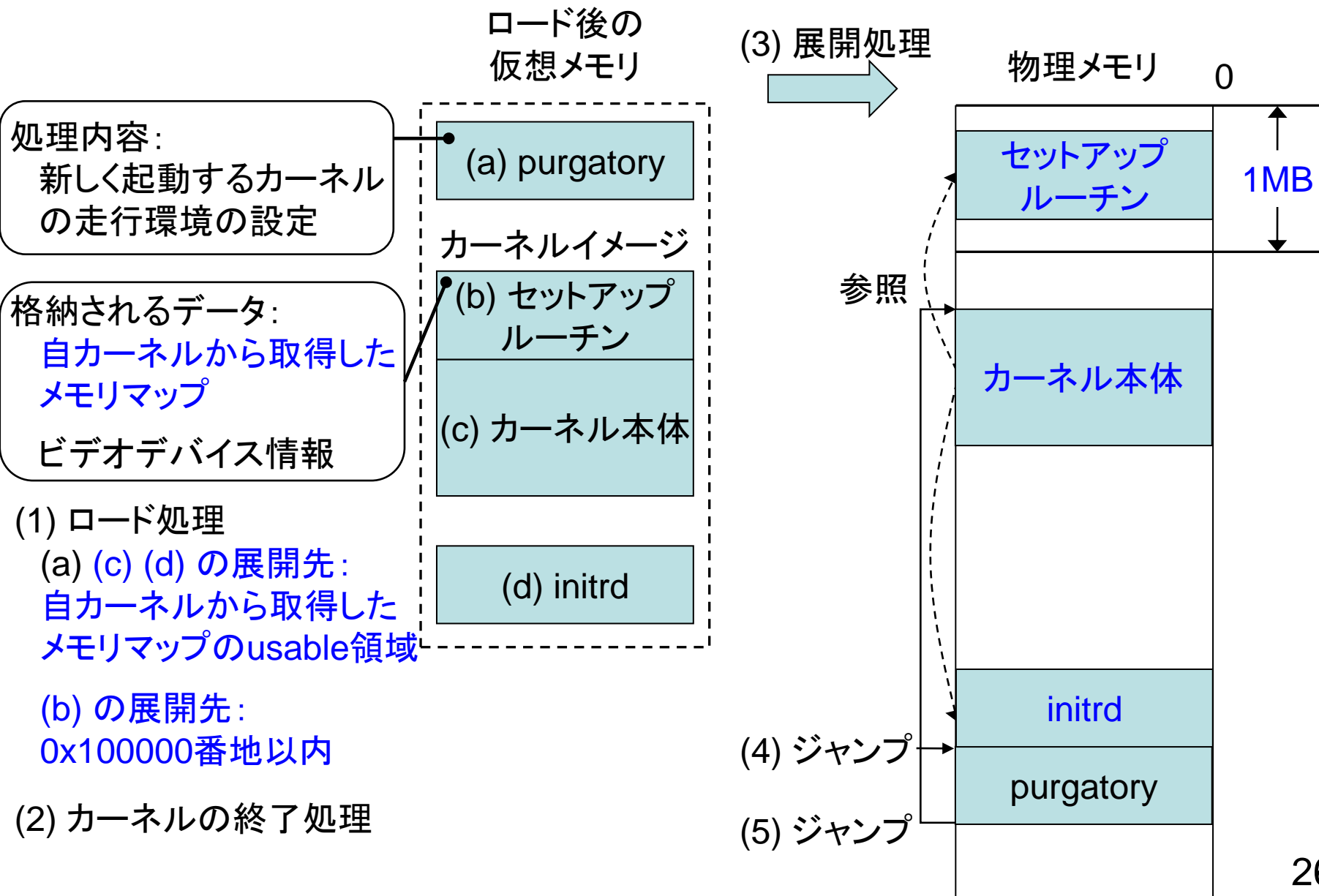
(課題2) プロテクトモードへの切り替え



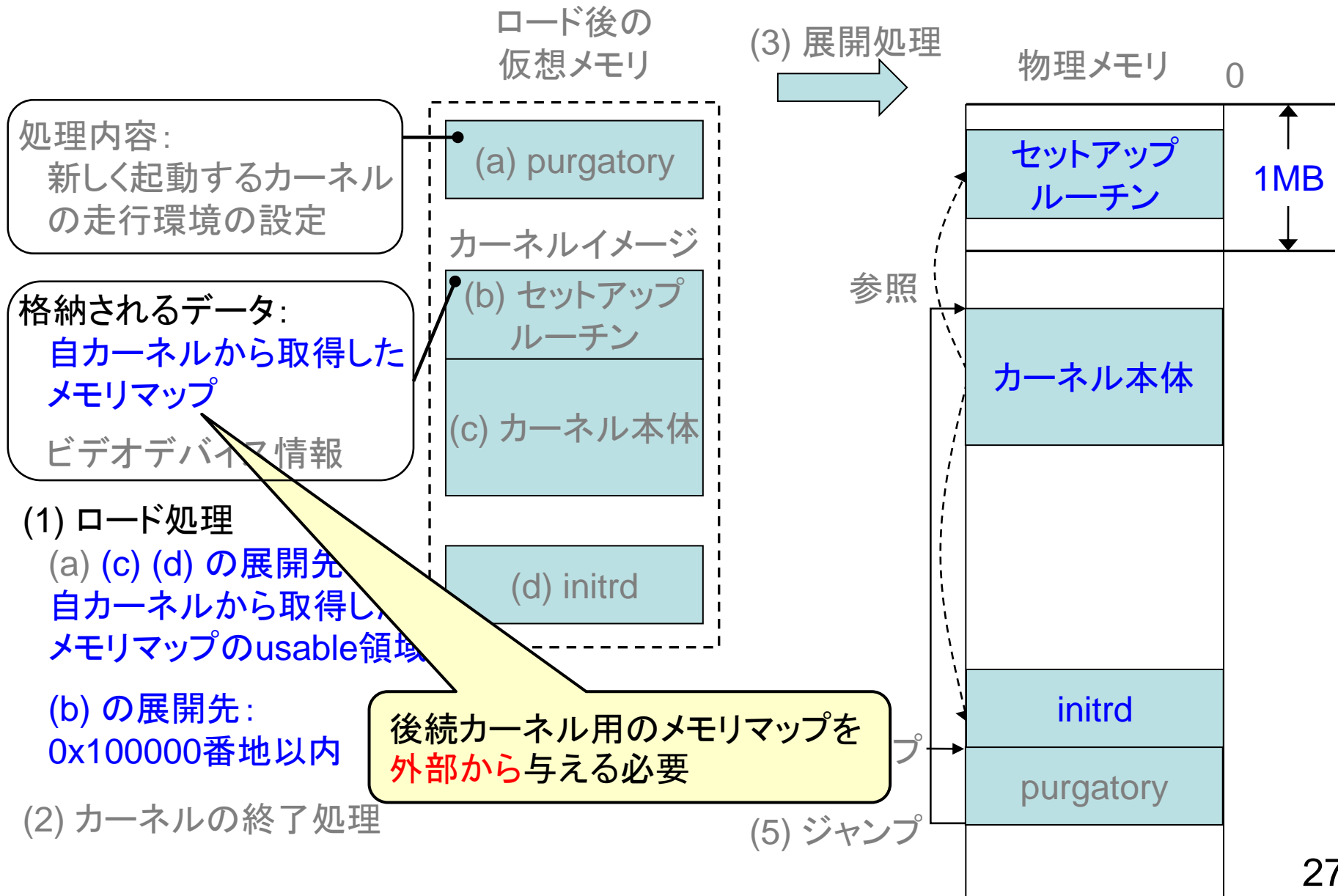
(対処2) プロテクトモードへの切り替え処理を追加



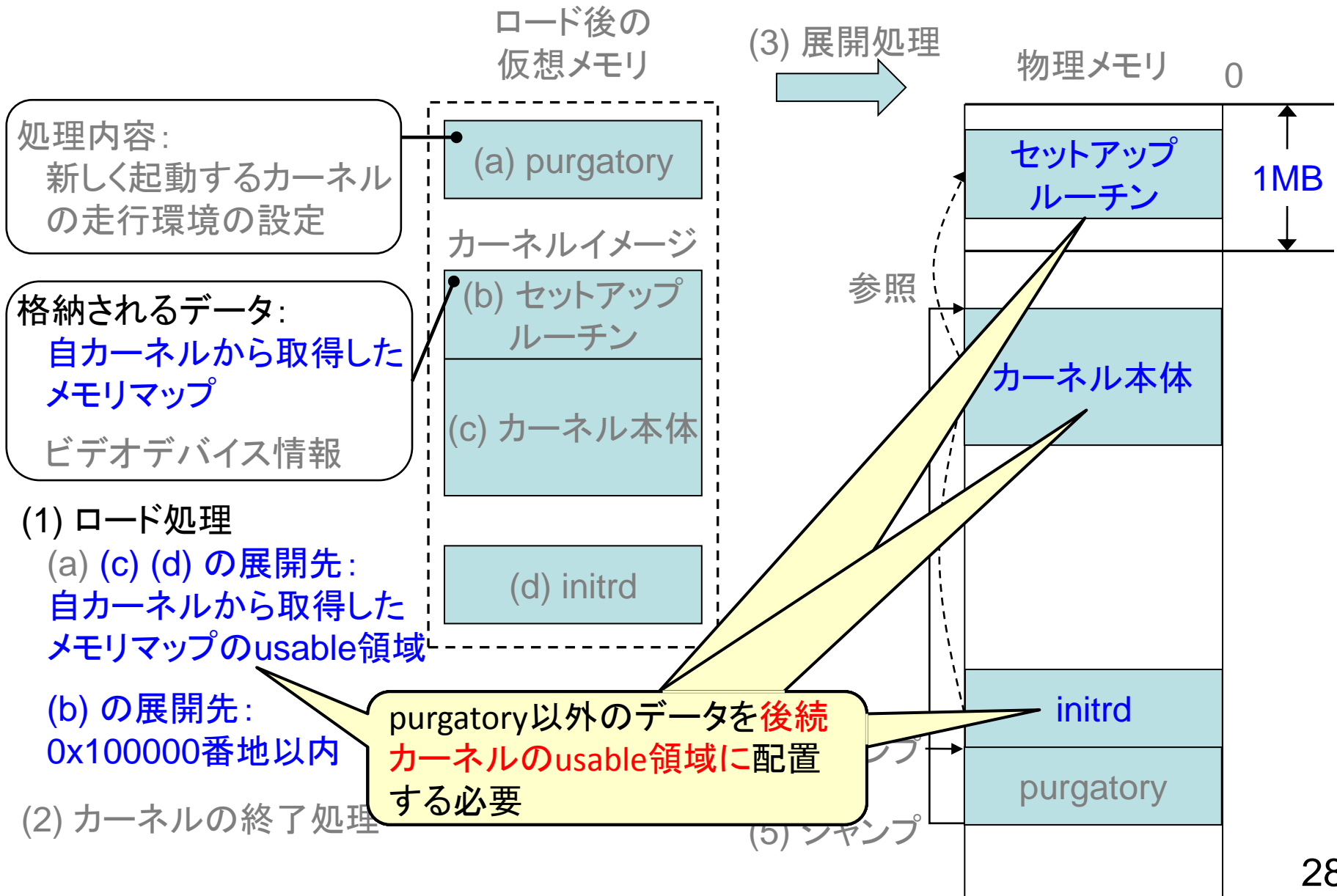
(課題3) 後続カーネル用のメモリマップの用意



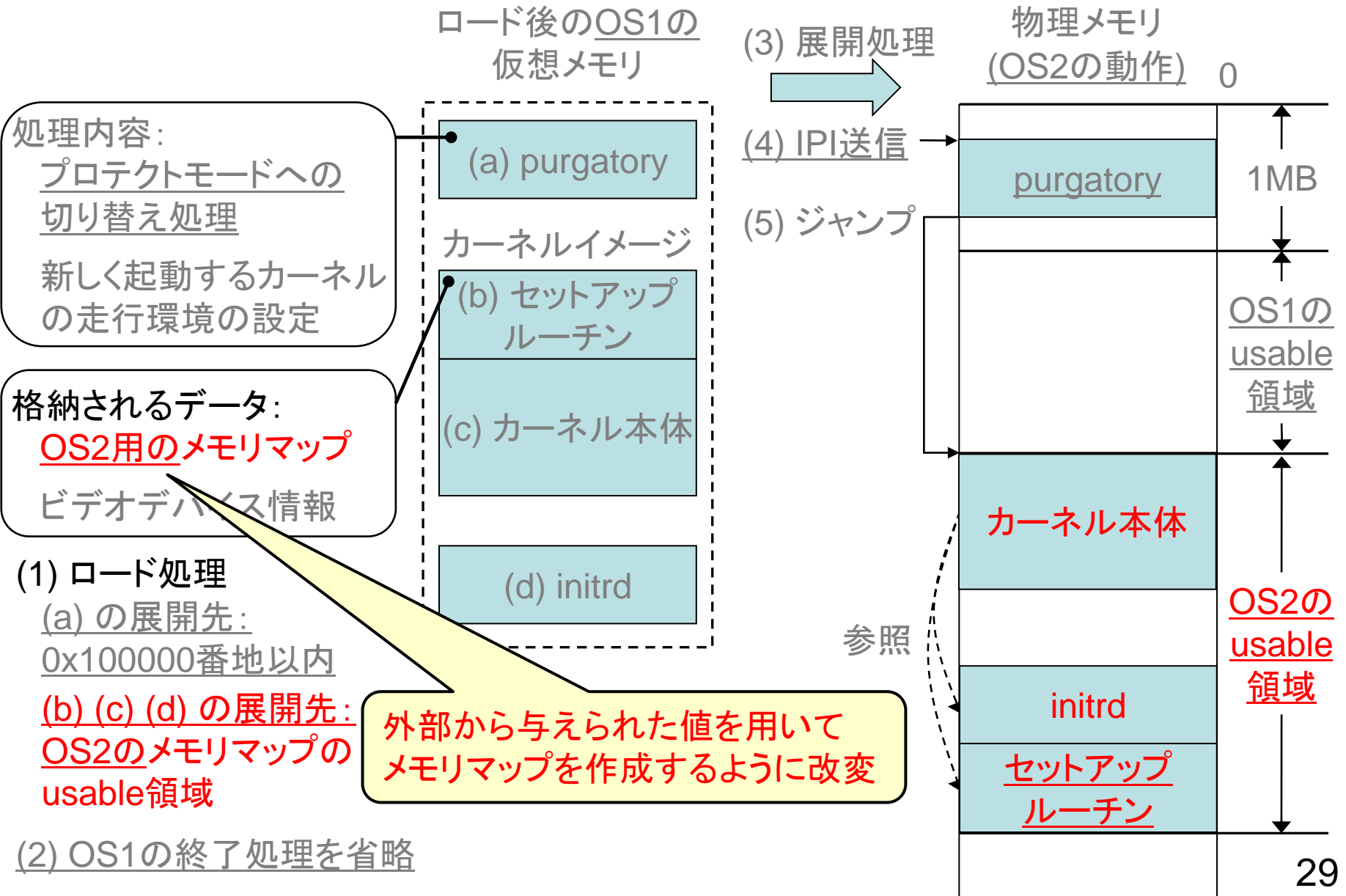
(課題3) 後続カーネル用のメモリマップの用意



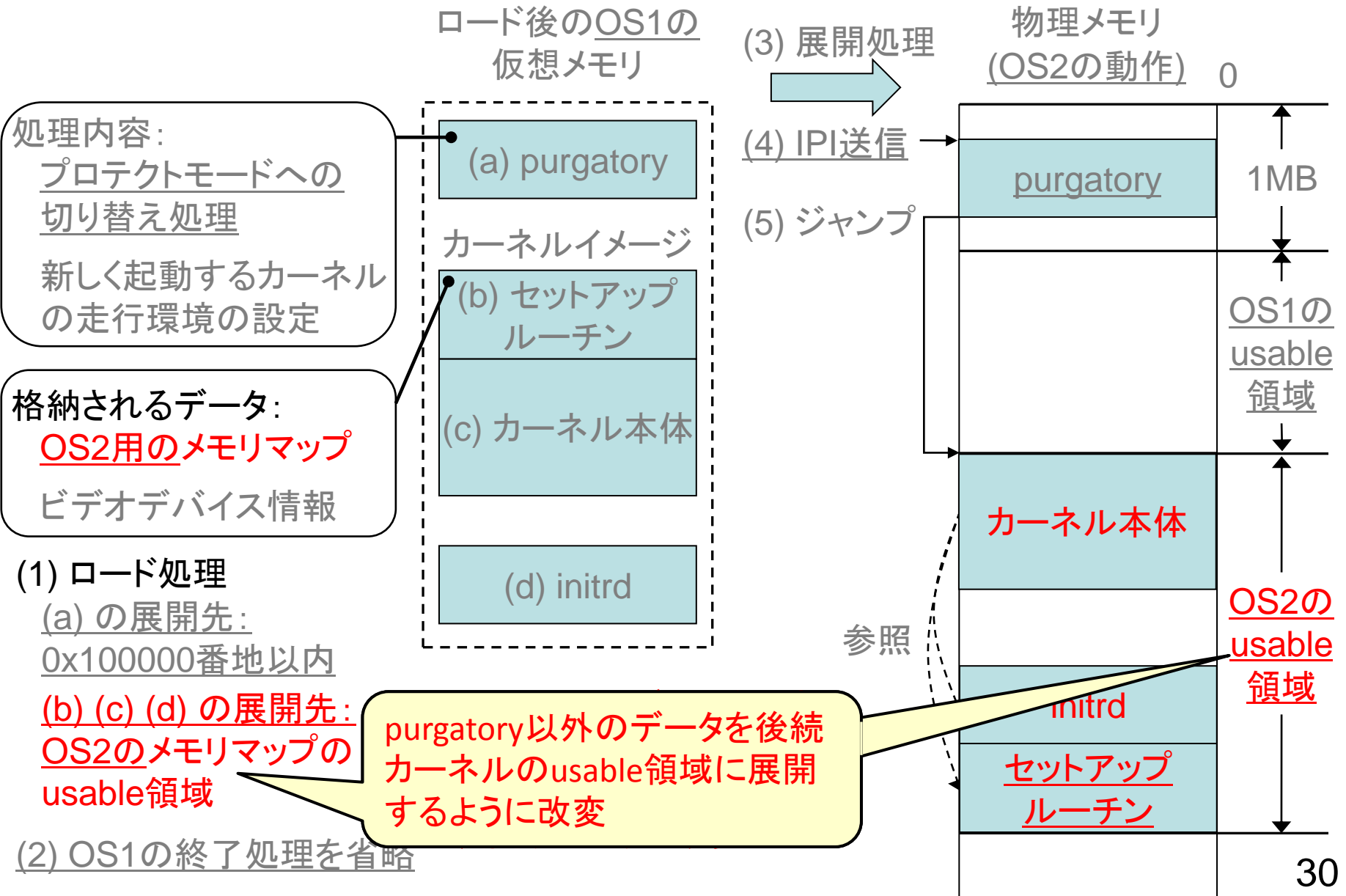
(課題3) 後続カーネル用のメモリマップの用意



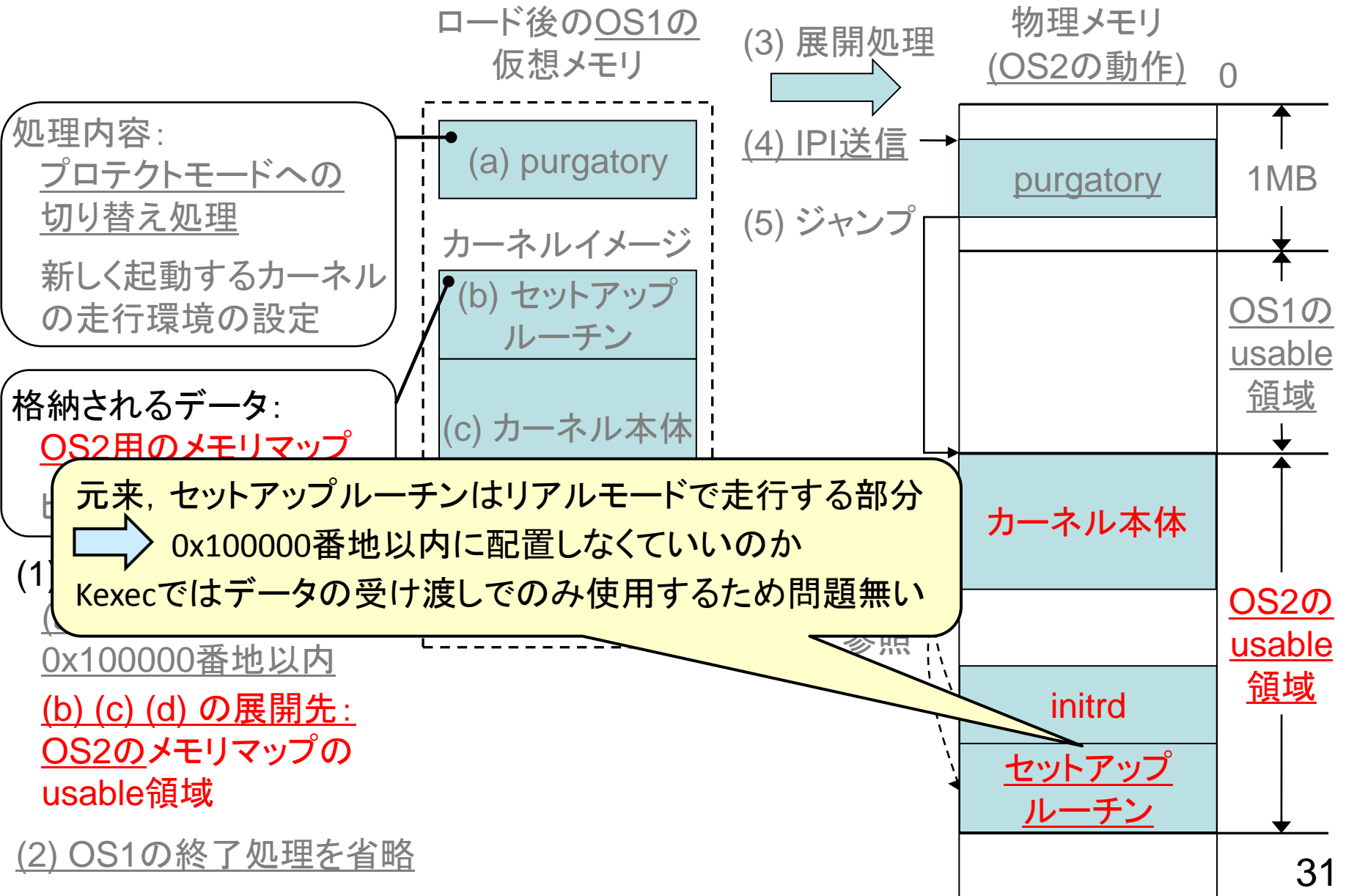
(対処3) メモリマップの改変



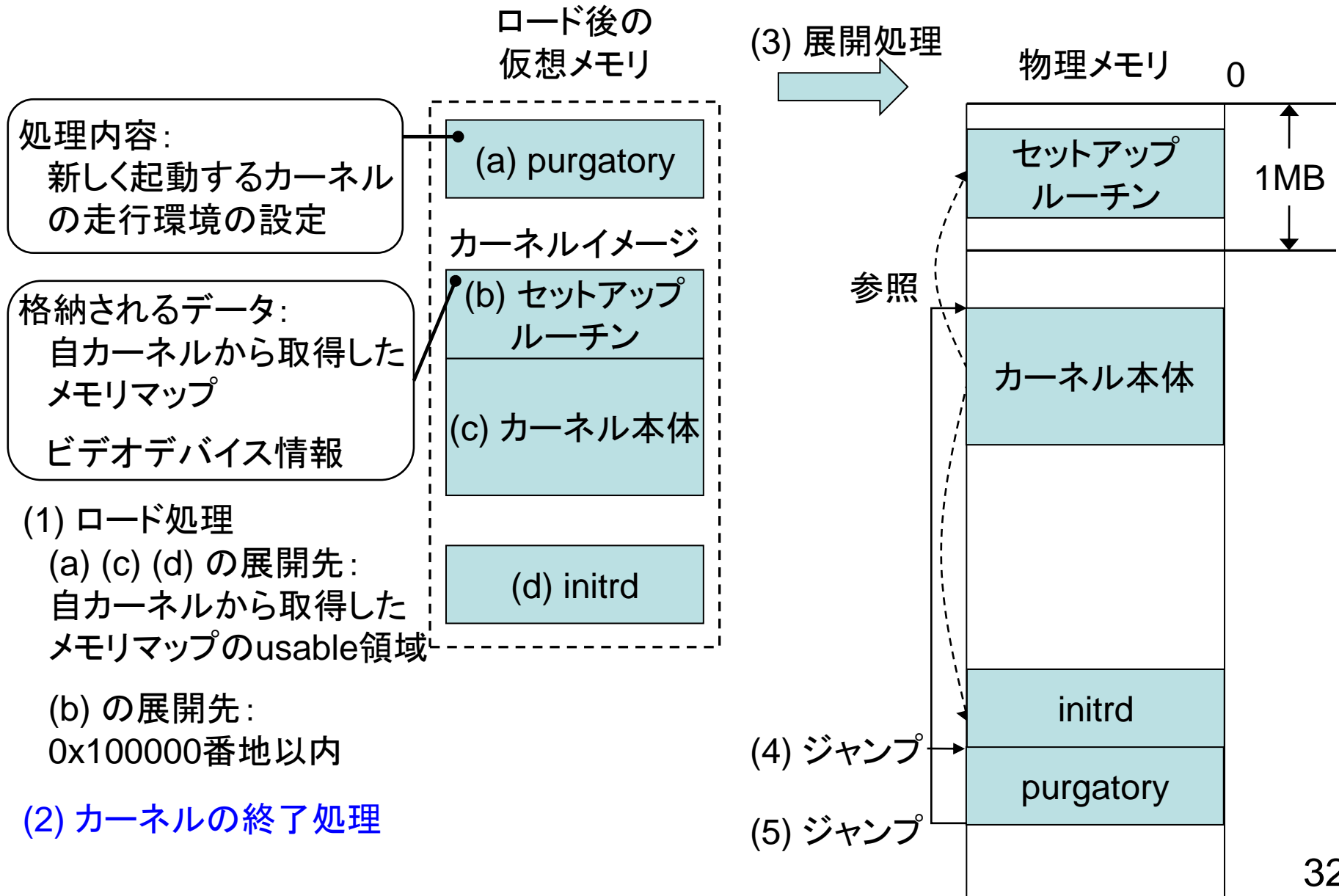
(対処3) メモリマップの改変



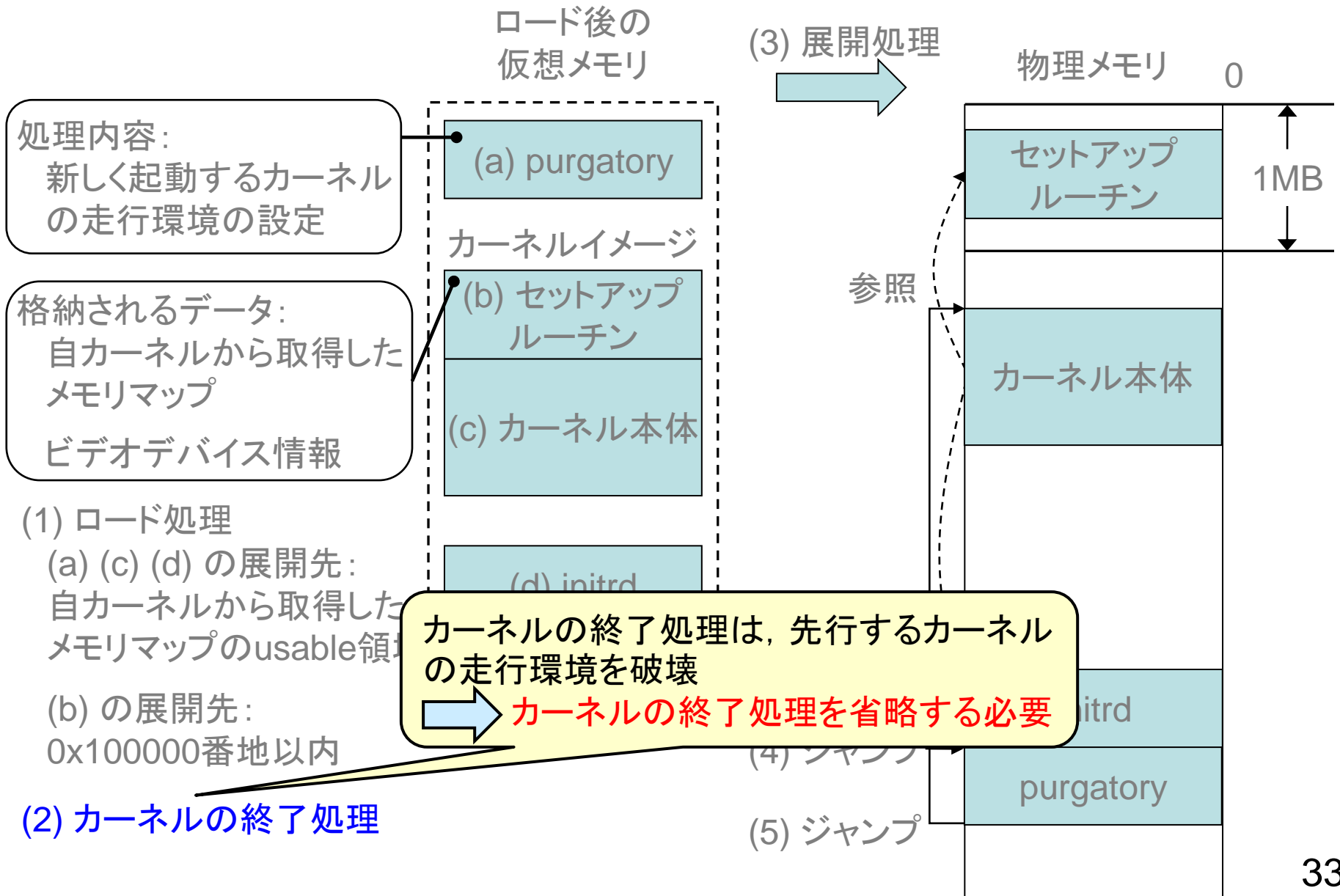
(対処3) メモリマップの改変



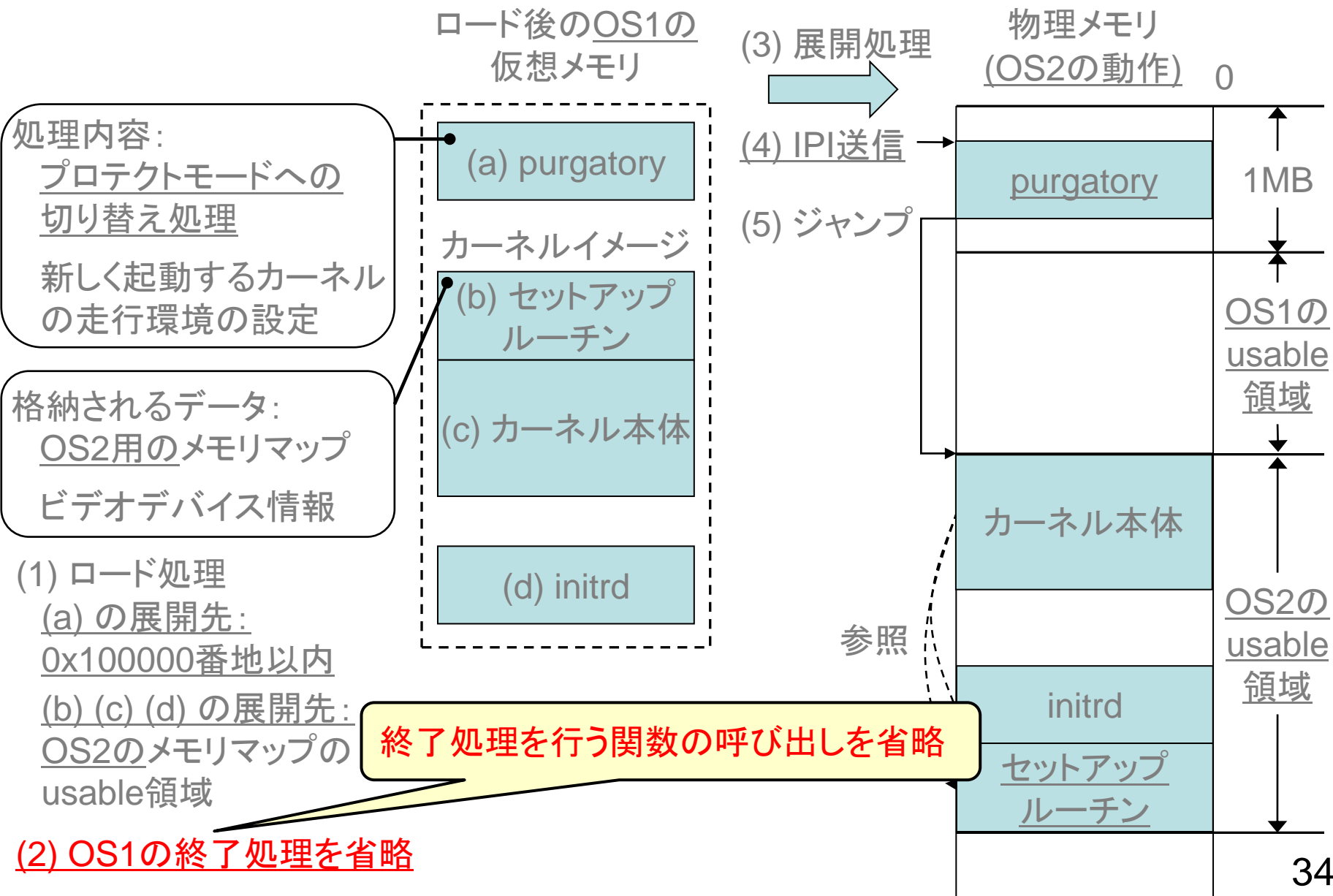
(課題4) 先行するカーネルの走行環境の保護



(課題4) 先行するカーネルの走行環境の保護



(対処4) 終了処理の省略



コード改修量

Kexecを用いた起動処理により, コード改修量を削減

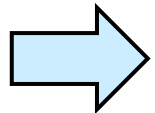
<カーネル内>

(1) 既存の起動方式によるコード改修量

全体:926行 起動処理:212行

(2) Kexecを用いた起動方式によるコード改修量

全体:779行 起動処理:65行

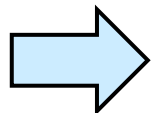


コード改修量を147行削減

<ブート用AP>

(1) 独自に作成した起動APのコード量:240行

(2) kexec-toolsのコード改修量:228行



ブート用APのコード改修量は, ほぼ同等

カーネルイメージの単一化

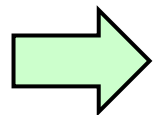
Kexecを用いた起動処理により, カーネルイメージを単一化

<起動の手順>

- (1) 先行するカーネルをブートローダから起動
- (2) 後続のカーネルをKexecを用いて起動

<各計算機資源の割り当て方法>

単一のカーネルイメージを用いて異なるパラメータを指定



それぞれのカーネルに異なる計算機資源を割り当て

- (1) CPU: ブートオプションとKexecのオプションで指定
- (2) メモリ: ブートオプションまたはKexecのオプションで指定
- (3) 入出力機器: 独自に作成したブートオプションで指定

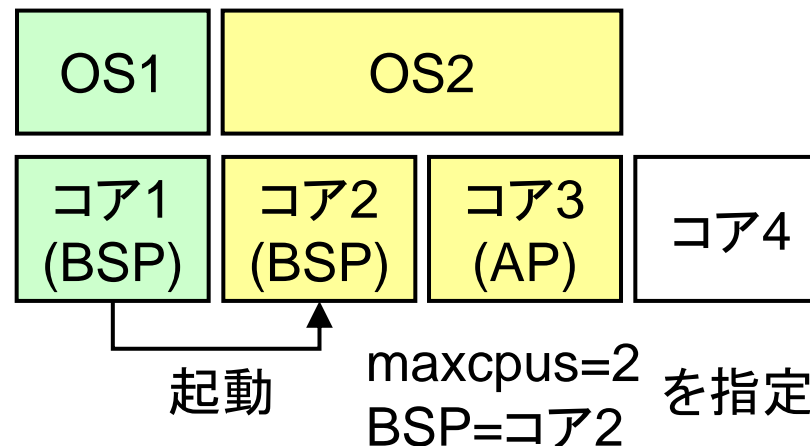
CPUの割り当て方法

各OSが占有するコアを指定する必要

<maxcpusブートオプション>

指定した値と同じ個数のコアを占有

Kexecに追加した実行オプションで起動対象(BSP)のコアを指定



BSP(Boot Strap processor):カーネル起動時に最初に起動するコア

メモリの割り当て方法

- (1) 各カーネルの展開先アドレスを指定する必要
- (2) 各OSが利用するメモリ領域を指定する必要

<先行するOSの場合>

既存の起動方式と同じ方法でメモリを割り当て

∴ 先行OSは、ブートローダを用いて起動

- (1) カーネルの展開先アドレスと利用する先頭アドレス

カーネルのコンフィグで指定することで**コンパイル時に確定**

- (2) 利用する終端アドレス

memブートオプションで指定

<後続のOSの場合>

Kexecのオプションを用いてメモリを割り当て

カーネルの展開先アドレスの指定方法

Relocatableな圧縮カーネルイメージとKexecをオプションを利用

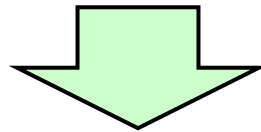
<Relocatableな圧縮カーネルイメージ>

- (1) 圧縮カーネルイメージが配置されたアドレスから展開
- (2) カーネル構築時にRelocatableに設定して作成

<Kexecのオプション>

- (1) --mem-min: データ展開先の先頭アドレスを指定
- (2) --mem-max: データ展開先の終端アドレスを指定

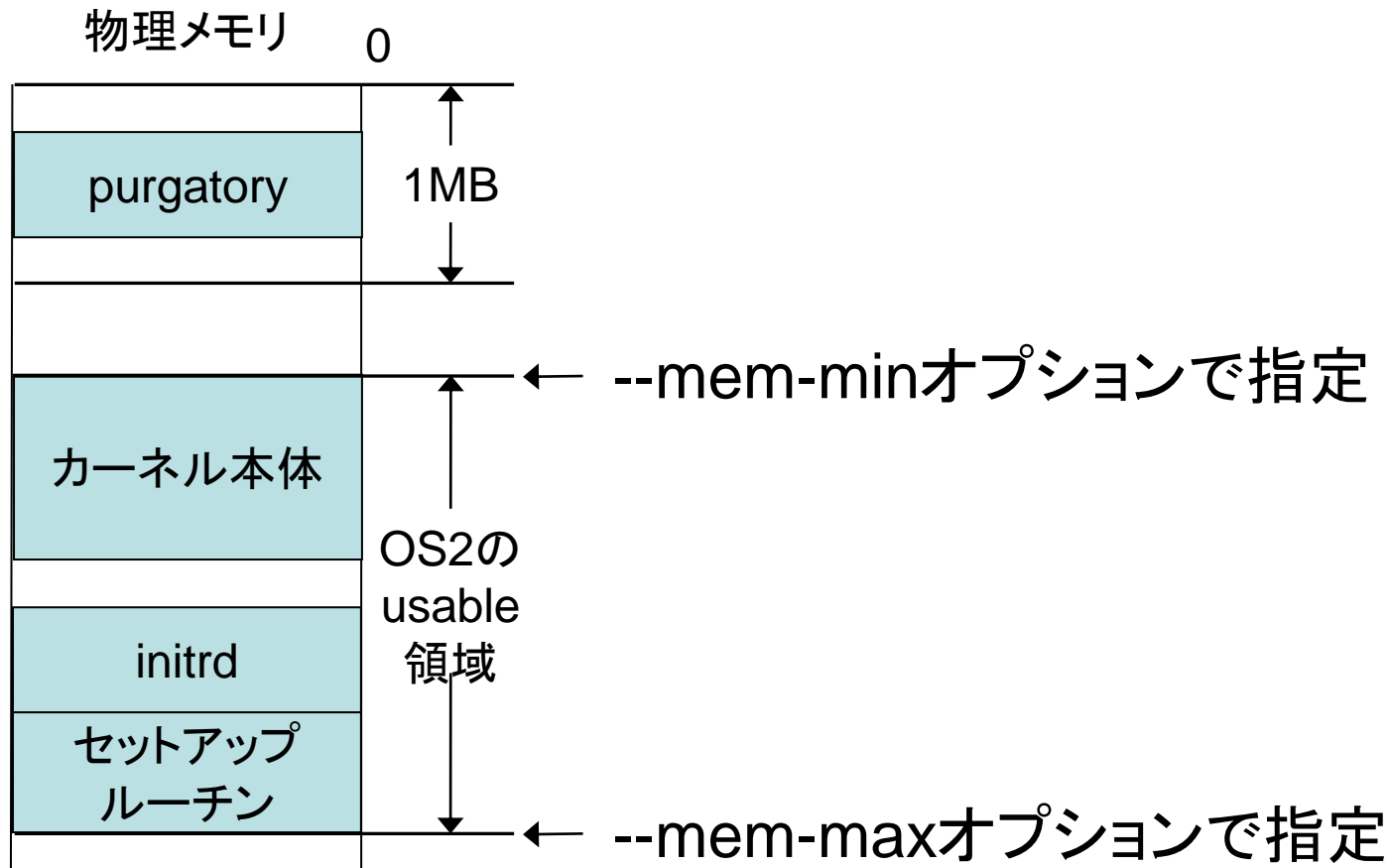
purgatory以外のデータは, 2つのオプションで指定したメモリ領域に展開



任意のアドレスにカーネルを展開可能

利用するメモリ領域の指定方法

usable領域の先頭と終端のアドレスを2つのオプションを用いて指定



入出力機器の割り当て方法

各OSが占有する入出力機器を指定する必要

各OSで占有するデバイスをあらかじめ確定

<例>

OS1: HDD1, VGA, キーボード

OS2: HDD2, シリアルポート

OS3: HDD3, NIC

占有対象のデバイスを指定するブートオプションを作成

OSごとにカーネル内の処理が異なる部分は、全てこのブートオプションの値によって処理を分岐

ブートローダからの起動時とKexecからの起動時の両方で使用可能

おわりに

<後続カーネル起動方式の改変>

Kexecは、カーネルを再起動する機能

➡ Kexecを後続のカーネルの起動用に改変

(対処1) IPIの送信によるコアの起動

(対処2) プロテクトモードへの切り替え処理を追加

(対処3) メモリマップの改変

(対処4) 終了処理の省略

<カーネルイメージの単一化>

単一のカーネルイメージを用いて異なるパラメータを指定

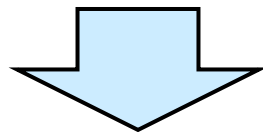
➡ それぞれのカーネルに異なる計算機資源を割り当て

參考資料

プロテクトモードへの切り替え

起動直後のコアはリアルモードで走行

一方, Kexecは, プロテクトモードのままで再起動を実現



リアルモードからプロテクトモードに切り替える処理を加える必要

(対処2) プロテクトモードへの切り替え処理を追加

(1) purgatoryの先頭にリアルモードからプロテクトモードに切り替えを行う処理を追加

後続カーネル用のメモリマップの用意

Kexecは、再起動のためにメモリマップを起動中のカーネルから取得
後続カーネル用のメモリマップを外部から与える必要

(対処3) メモリマップの改変

- (1) 外部から与えられた値を用いてメモリマップを作成するように改変
- (2) purgatory以外のデータを(1)で作成したメモリマップのusable領域に展開するように改変

<セットアップルーチンの配置位置>

セットアップルーチンはリアルモードで走行する部分

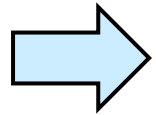
➡ 物理メモリの0x100000番地以内に配置しなければならない

しかし、Kexecではデータの受け渡し以外の目的では使われない

➡ 0x100000番地以上のusable領域に配置するように改変

先行するカーネルの走行環境の保護

カーネルの終了処理は、先行するカーネルの走行環境を破壊



カーネルの終了処理を省略する必要

(対処4) 終了処理の省略

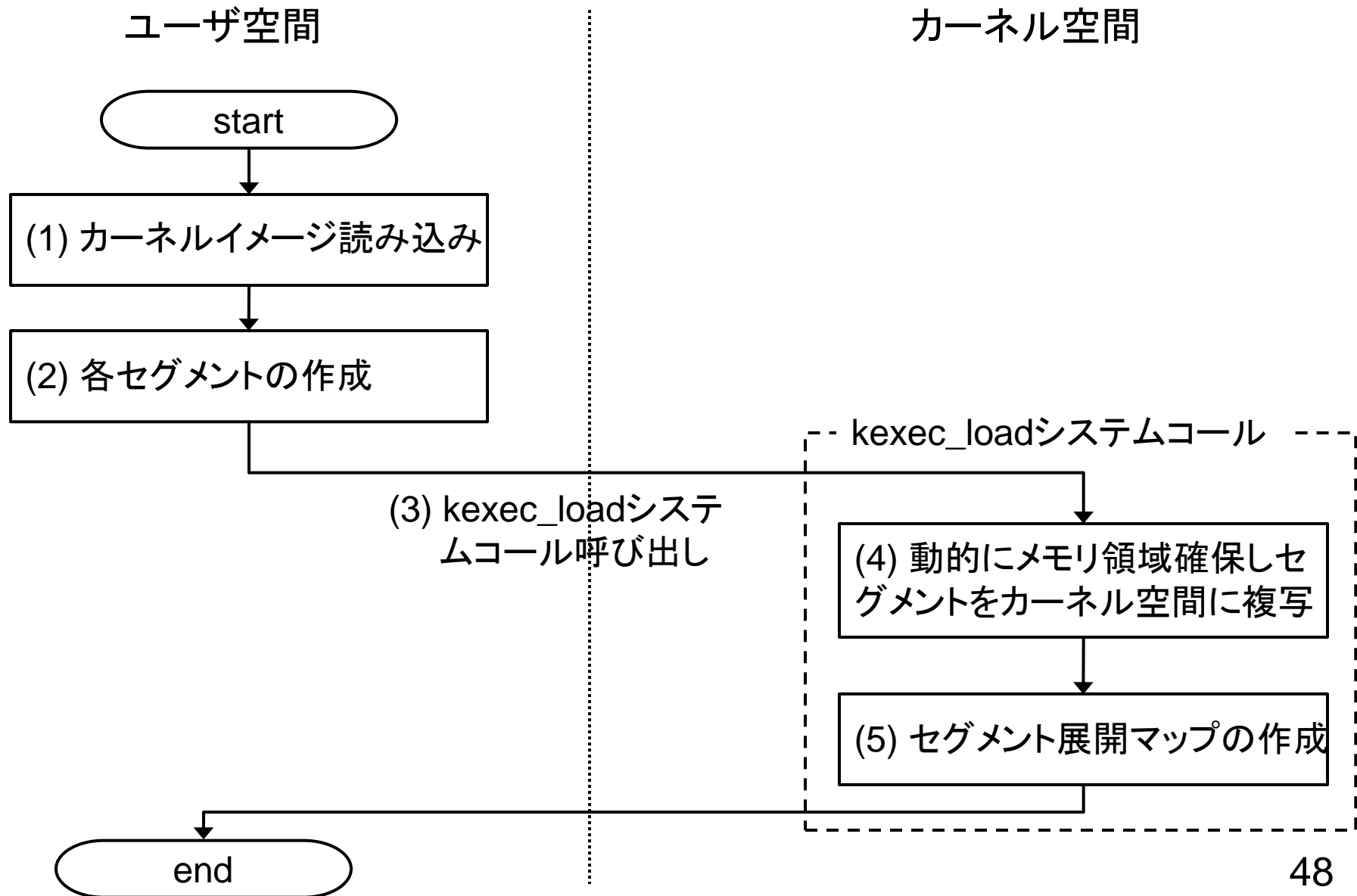
以下の処理を行う関数の呼び出しを省略

- (1) 終了処理
- (2) Local IRQの無効化
- (3) 各種セグメントレジスタの初期化
- (4) GDTの初期化
- (5) IDTの初期化
- (6) 各種汎用レジスタの初期化

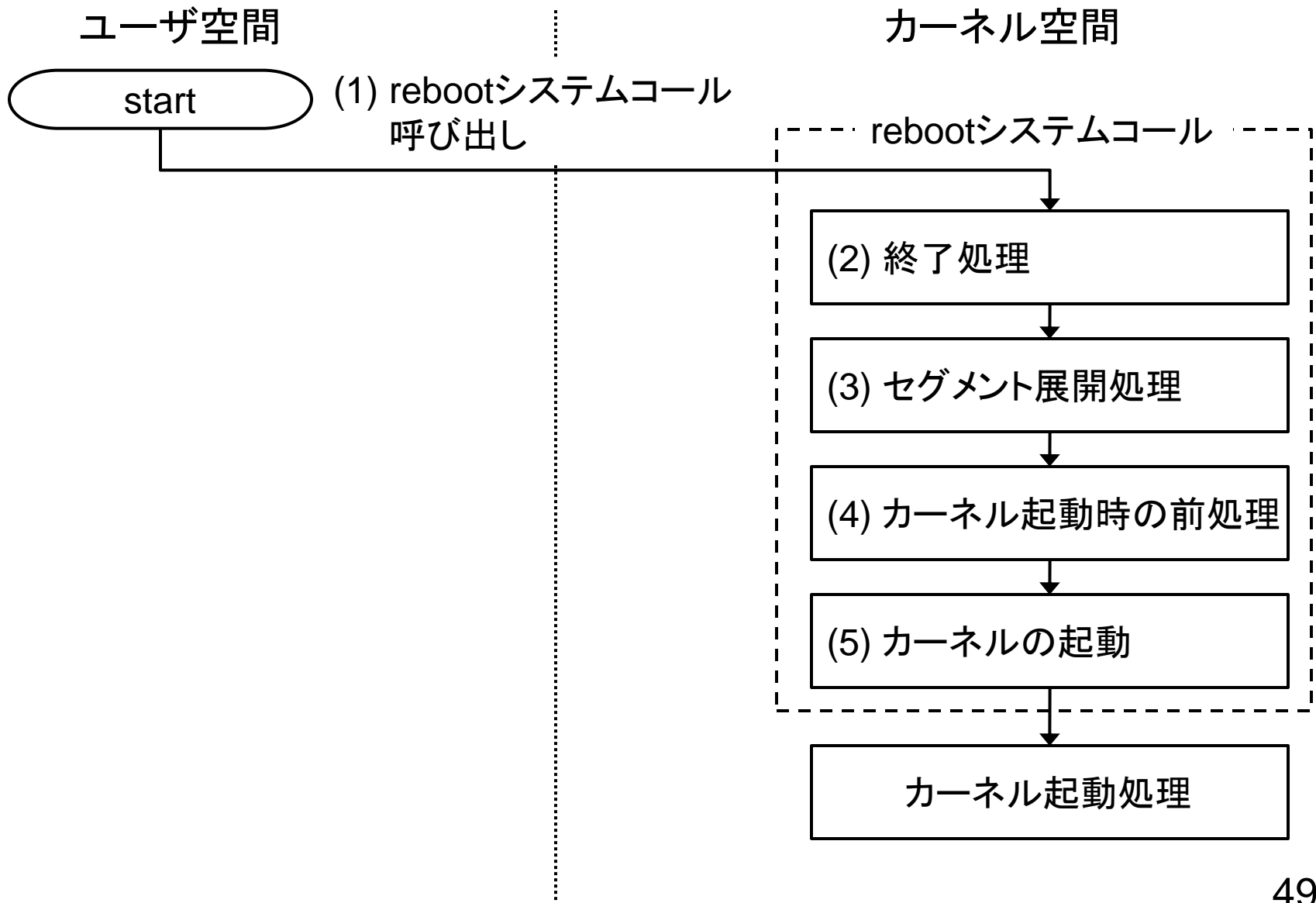
Kexecによるカーネル終了処理

	ファイル	関数	対象
(1)	drivers/base/core.c	device_shutdown	デバイス
(2)	drivers/base/sys.c	sysdev_shutdown	システムデバイス
(3)	arch/x86/include/asm/smp.h	smp_send_stop	SMP(コア)
(4)	arch/x86/kernel/apic.c	lapic_shutdown	APIC
(5)	arch/x86/kernel/hpet.c	hpet_disable	HPET

カーネル読み込み機能の処理流れ



カーネル起動機能の処理流れ



おわりに

Kexecを用いて、コアごとにLinuxカーネルを同時走行させる方式について述べた

(課題1) コアの起動

(課題2) プロテクトモードへの切り替え

(課題3) 後続カーネル用のメモリマップの用意

(課題4) 先行するカーネルの走行環境の保護

(対処1) IPIの送信によるコアの起動

(対処2) プロテクトモードへの切り替え処理を追加

(対処3) メモリマップの改変

(対処4) 終了処理の省略

はじめに

1台の計算機上で複数のOSを走行させる研究が活発

<先行研究>

(1) SHIMOS[1][2]

(2) TwinOS[3]

(A) 各OSは実計算機上で直接動作

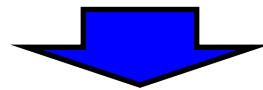
(B) ハードウェアの分割方法

(i) メモリ: 空間分割

(ii) CPU: タイマ割り込みによる時分割

(シングルコアプロセッサ)

(iii) 入出力機器: OSごとに占有



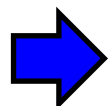
1台の計算機上で複数のLinuxを独立に走行させる方式を設計

設計目標

- (1) 1台の計算機上で2つ以上のLinuxを動作
- (2) 各OSは、複数あるコアのうち1つ以上のコアを占有
- (3) 入出力機器について
 - (A) デバイス単位で分割
 - (B) 各OSが仮想化によらず直接占有制御

<特徴>

- (1) 各OS間の影響が最小になるようにそれぞれのOSが独立
- (2) 各OSが単独で動作する場合に近い環境と性能で走行



各OSは**独自に終了と再起動**を実行可能

提案方式の構成

(1) CPU

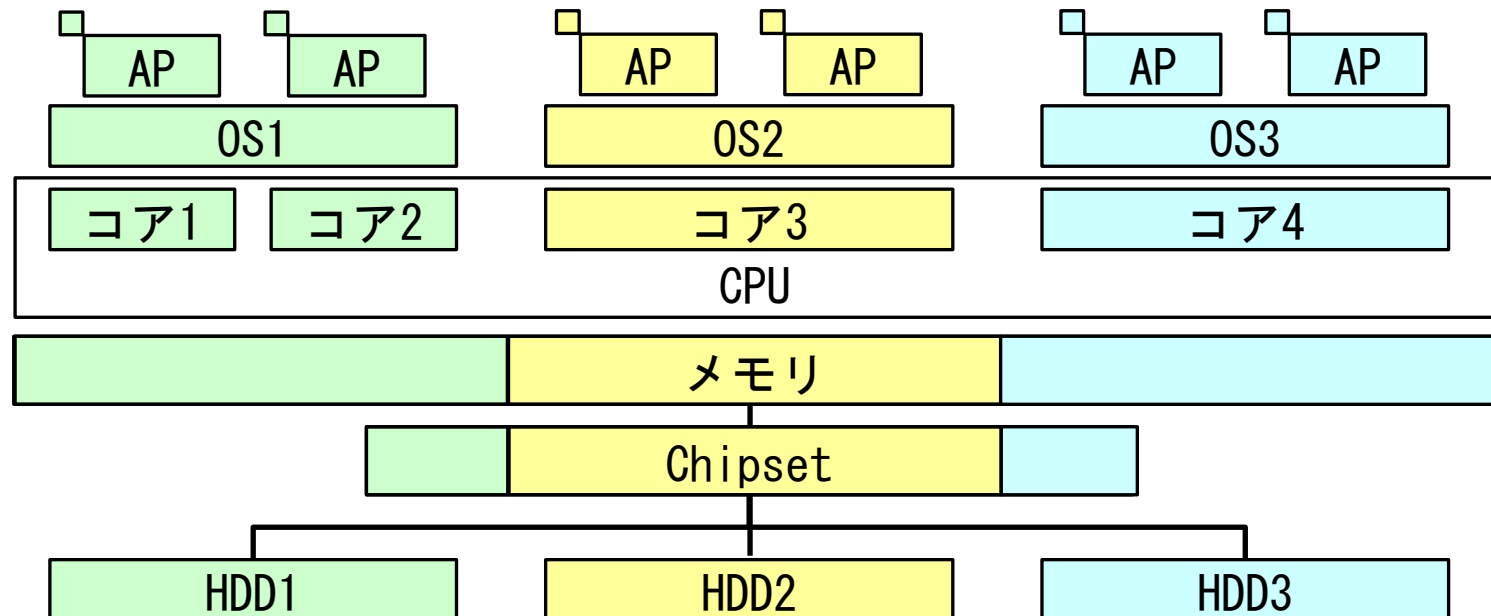
- (A) マルチコアプロセッサを使用
- (B) コアを分割し各OSに分配

(2) 実メモリ

- (A) OSの起動時に領域を分割占有

(3) 入出力機器

- (A) OS毎に指定された入出力機器のみを占有制御



提案方式の実現における課題

- (1) メモリの分割と占有
- (2) 各OSの起動
- (3) コアの分割と占有
- (4) 入出力機器の分割と占有
- (5) 割り込みの制御
- (6) 終了処理と再起動処理

以降では、課題の内容と設計について説明

メモリ分割と占有

<課題>

各カーネルが使用する物理メモリの範囲を制限

➡ 占有する領域の終端/先頭アドレスの指定が必要

(1) 終端アドレス

Linuxの既存機能によりブートパラメータで指定可能

(2) 先頭アドレス

Linuxの既存機能はないため対処が必要

<設計>

BIOSが取得したメモリマップをOSが占有するメモリ領域のみが
利用可能な領域であるよう書き換える

➡ 先頭アドレスの指定が可能

各OSの起動

<課題>

- (1) 1番目のOSは、通常のLinuxに近い初期化処理で起動可能
- (2) 2番目以降のOSはハードウェア制御を制限
 - ➡ 制限に合わせてカーネル起動処理を変更

<2番目以降のOSが受ける制限>

- (1) ブートローダを介さない
- (2) 占有するコアの初期化を行えない

<設計>

1番目のOSから他のOSを順次起動

- (1) カーネルイメージをメモリ上に展開
- (2) コアの初期化処理

他のOSが占有するコアを1番目のOSから初期化

- (3) セットアップコードの実行

CPUコアの分割と占有

<課題>

Linuxの既存機能: 先頭コアからの利用コア数は指定可能

➡ 何番目のコアから何個占有させるかを指定できる必要

<設計>

(1) 占有ルールの設定

x86 SMPの各コアは, Local APICのID(LAPIC ID)により識別可能

(A) 未使用のコアの内, LAPIC IDが最小のものをBSPに指定

(B) LAPIC IDの小さいコアから順番に利用 (Boot Strap Processor)

(C) 自身のBSPよりも小さいLAPIC IDを持つコアは不使用

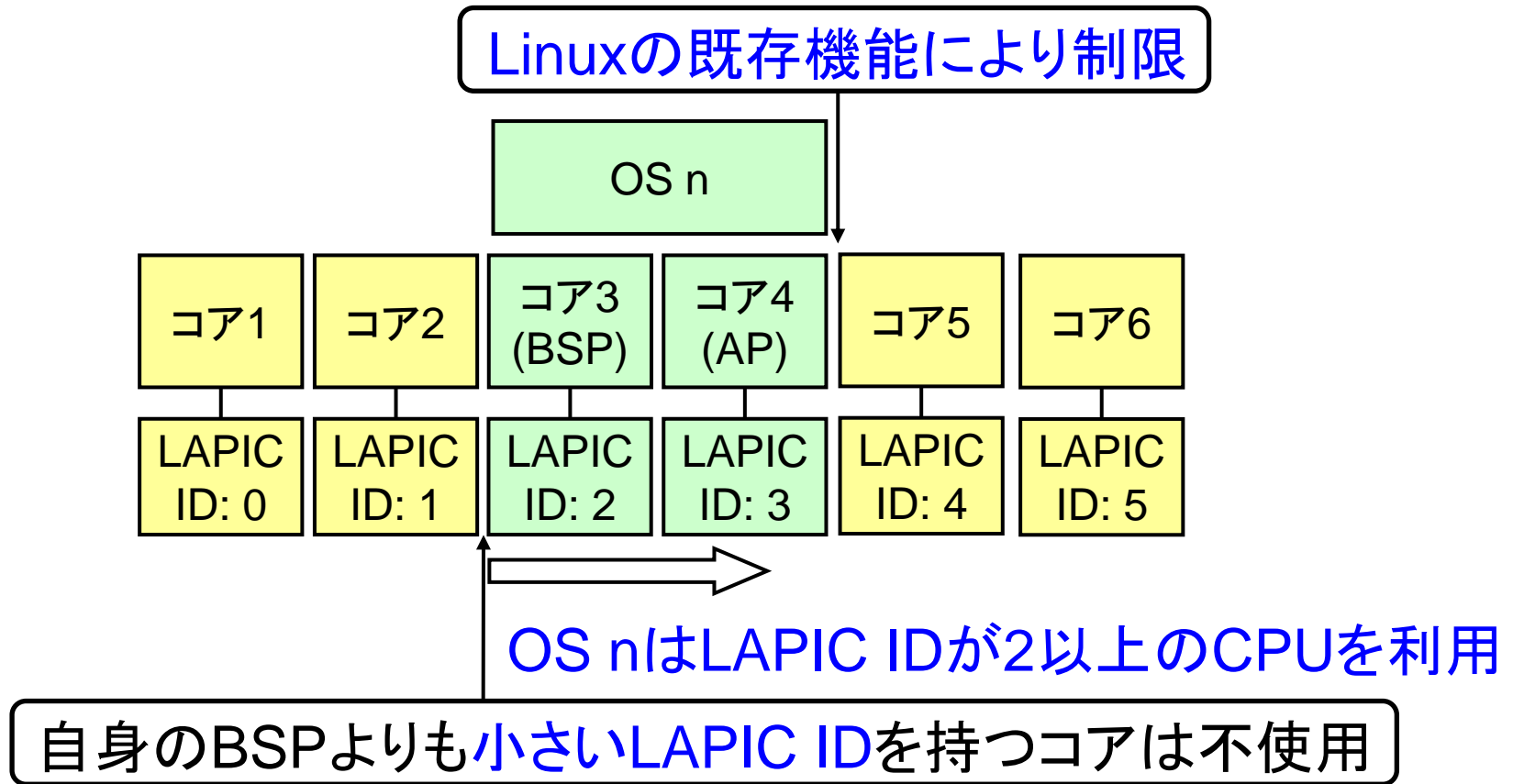
➡ 指定したコアから指定個数を占有可能

(2) 論理APIC IDの重複防止

各コアのLAPICは, 論理APIC IDにより割り込み先を決定

➡ 論理APIC IDの決定規則を操作し, 重複しないように変更 57

CPUコアの占有ルール



BSP: Boot Strap Processor

AP : Application Processor

入出力機器の分割と占有

<課題>

通常のLinuxは、利用可能なデバイスをすべて占有する

➡ 占有するデバイスと占有しないデバイスの仕分けが必要

<設計>

カーネルコードを変更することで、**デバイスタイプ別に**分割と占有

(1) **PCI等のデバイス**

デバイスドライバを使用

➡ **デバイスドライバの利用の可否により分割・占有**

(2) **レガシーデバイス** (PS/2キーボード, シリアルポート, VGA)

デバイスドライバを未使用

➡ **カーネルコードを直接変更することで分割・占有**

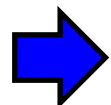
割り込みの制御

<課題>

通常のLinuxを複数走行させた場合の割り込みの設定

(A) 先に起動しているカーネルの設定を上書き

(B) 最後に起動したカーネルが割り込みを設定

 初期化処理と設定の調整が各OS間で必要

<設計>

(1) I/O APICに制御される割り込み

他のカーネルのコアに関する設定内容进行操作せずに
割り込みを設定

(2) MSI(Message Signaled Interrupt)を用いる割り込み

デバイス単位で割り込みを設定可能

 特別な処理は不必要

終了処理

<課題>

他のOSに影響を与える処理には対処が必要

<設計>

(1) デバイス終了処理

(A) デバイスの終了処理を除外

(2) タイマの無効化

(A) タイマの無効化処理を除外

(3) I/O APICの無効化

(A) I/O APICの無効化処理を除外

(B) 終了するOSの割り込み設定をI/O APICから除外

(4) 計算機のシャットダウン

(A) 計算機をシャットダウンせず, コアをHALT状態に設定

再起動処理

<課題>

終了処理後, 独自にカーネルを展開し起動処理を行う必要

<設計>

独自にカーネルを展開し起動を行う処理を追加

(1) 終了処理

(2) initrdとカーネルイメージの配置

(A) initrdとカーネルイメージを指定のメモリ位置に展開する

(3) 圧縮カーネルの展開ルーチンにジャンプ

(A) BIOS, ブートローダ, セットアップルーチンの処理を行わない

(B) 圧縮カーネルの展開ルーチンから処理を開始する

おわりに

(1) まとめ

1台の計算機上で複数のLinuxを独立に走行させる方式の設計

<特徴>

- (1) 各OS間の影響が最小になるようにそれぞれのOSが独立
- (2) 各OSが単独で動作する場合に近い環境と性能で走行

 各OSは**独自に終了と再起動**を実行可能

(2) 残された課題

提案方式の実装と評価