Kexecを利用した Mintオペレーティングシステムの 起動方式

中原 大貴　千崎 良太　牛尾 裕　片岡 哲也
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

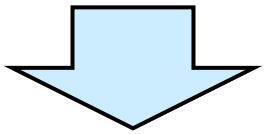
＜Mint＞

(1) マルチコアプロセッサにおいて，複数OS同時走行機能を実現

(2) 先行OSから後続OSを順次起動

＜現状の起動方式の問題点＞

走行するOSごとに異なるカーネルイメージが必要



カーネルイメージの単一化のため，Kexecを利用して後続OSを起動

＜Kexec＞

(1) Linuxカーネルの高速な再起動方式

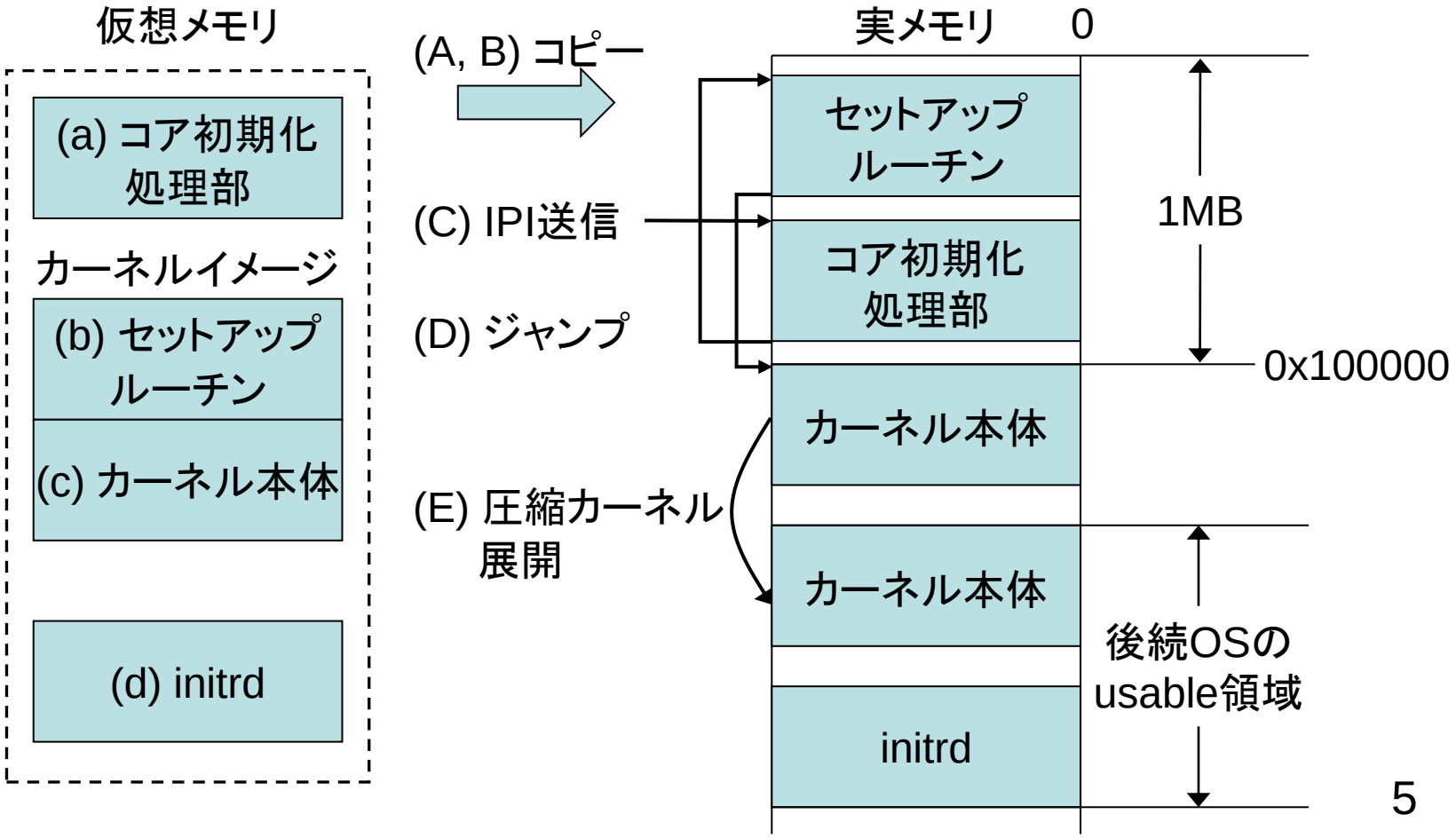
(2) Linuxに標準搭載

目次

目次

Mintの現状の起動方式

(1) 実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先行OSを起動
(3) 先行OSから後続OSを起動



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＞

カーネル間の実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

Kexecを利用した後続OSの起動方式

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の再起動方式には，多くの類似点

⇒ Kexecを改変し，再起動処理の代わりに後続OSの起動処理を実行

＜期待される効果＞

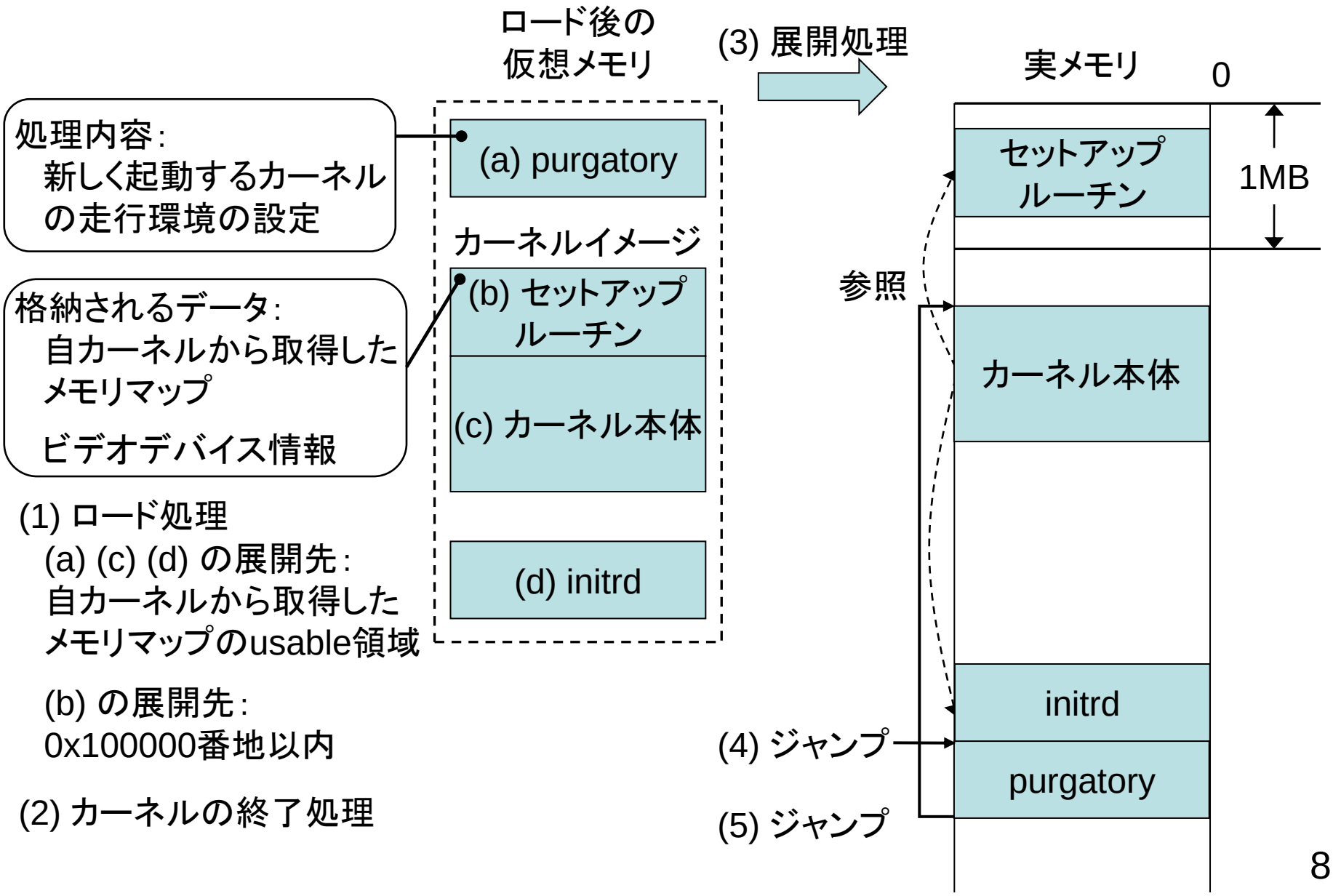
(1) 起動機能をLinuxカーネルに元々存在する機能で置き換え可能

⇒ (問題1)今後のLinuxのバージョンアップへの追従が困難に対処

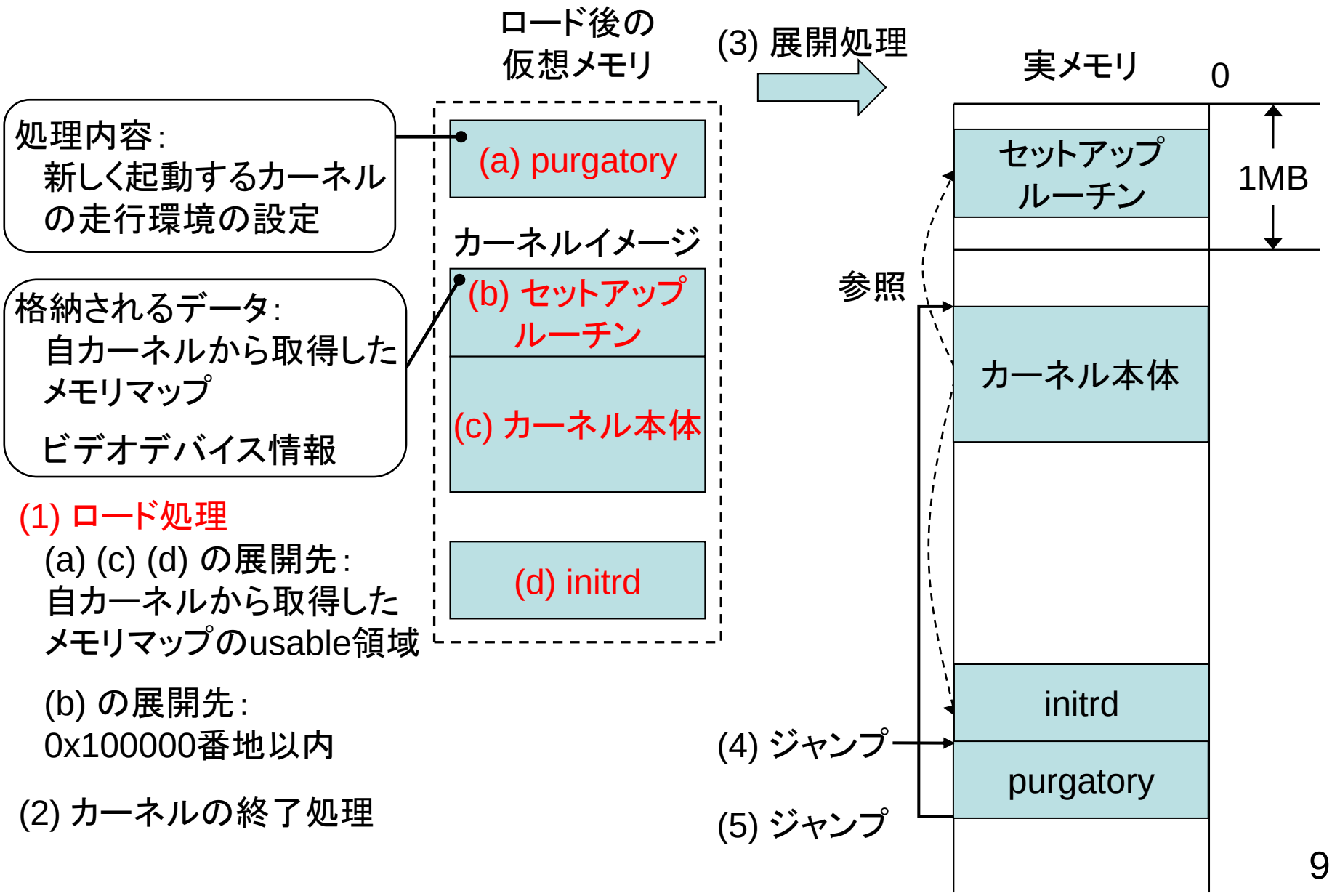
(2) 各カーネルの利用メモリ領域を起動時に任意に指定可能

⇒ (問題2)カーネル間の実メモリ配分の変更にカーネルの再構築が必要 に対処

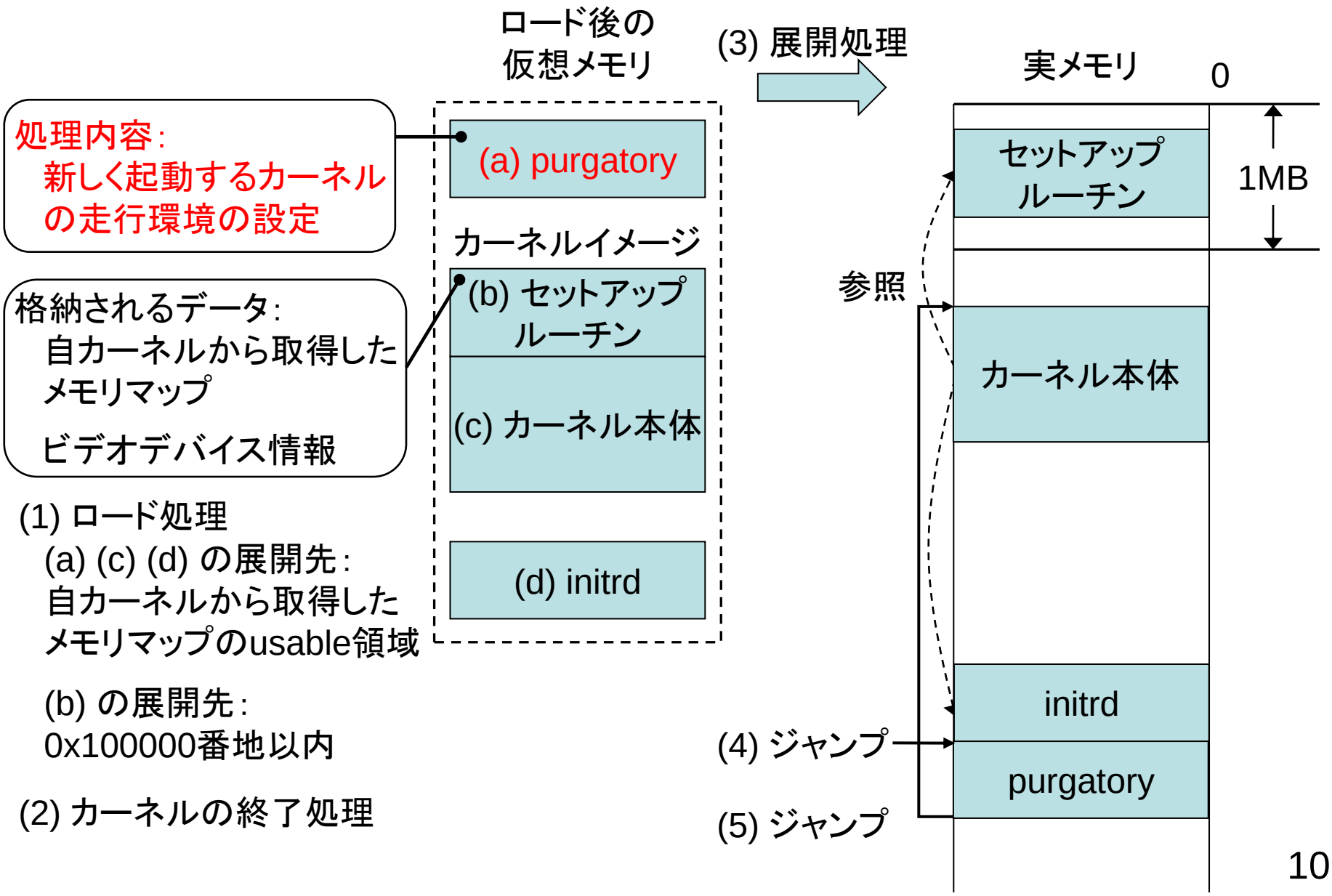
Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

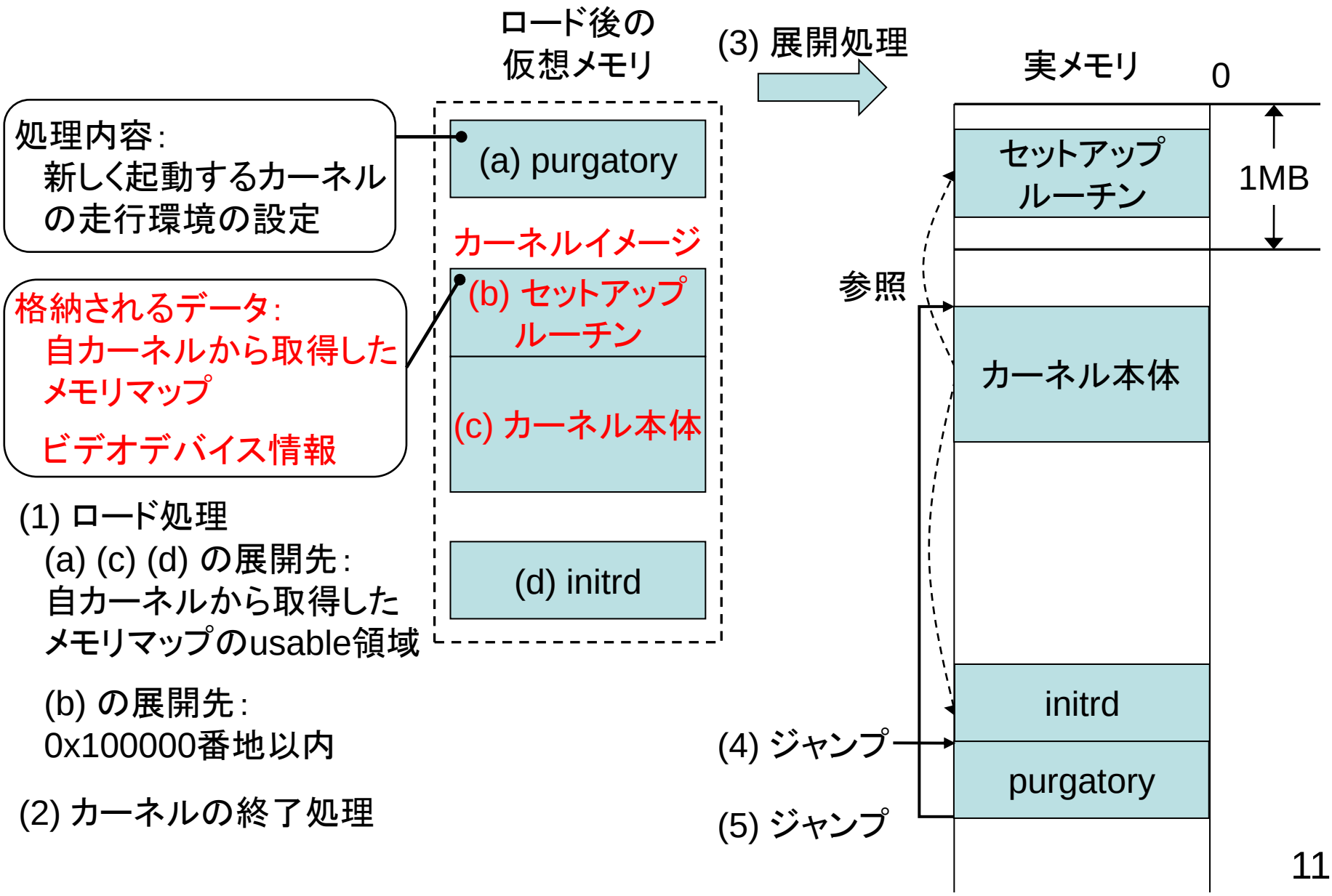
Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

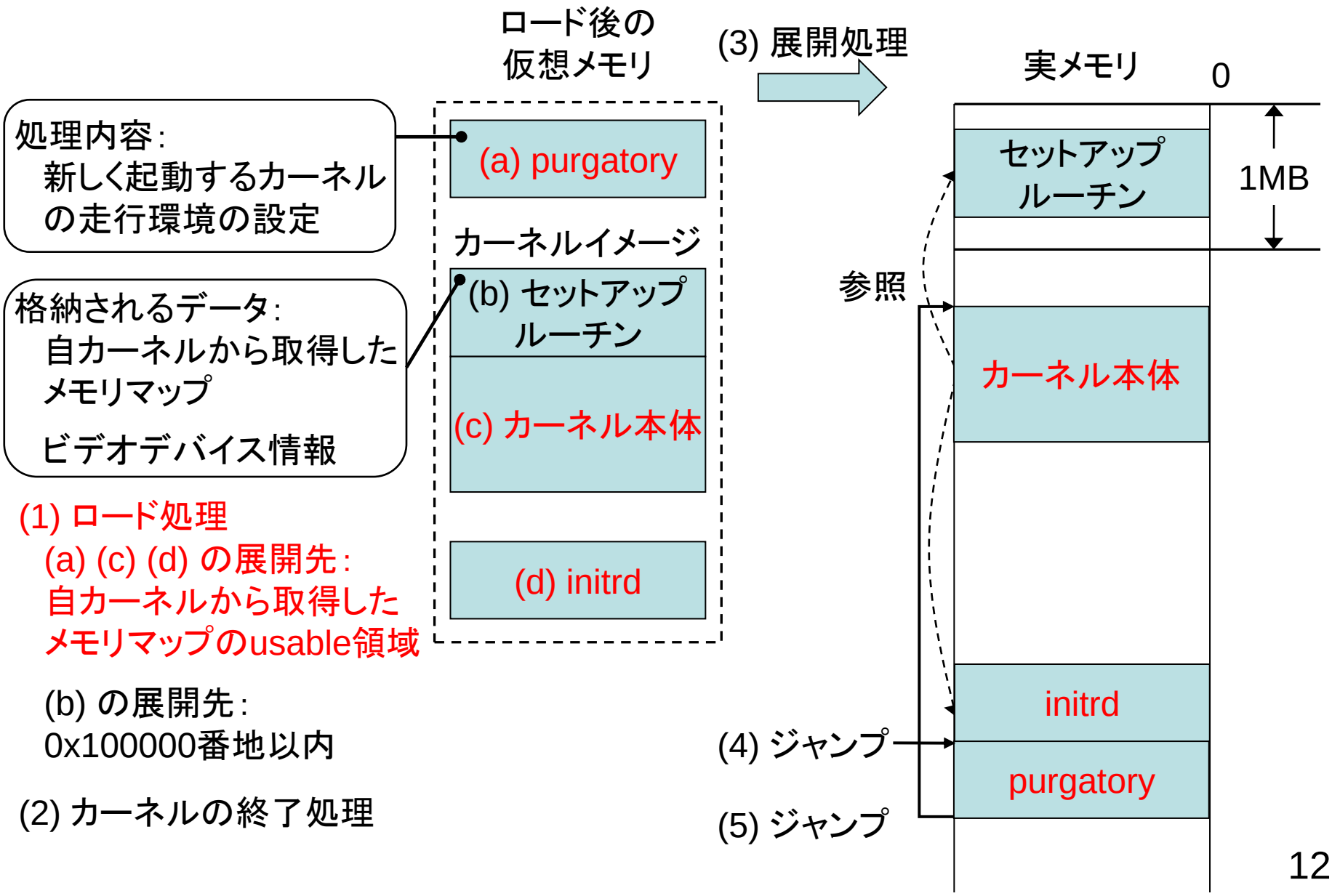
Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

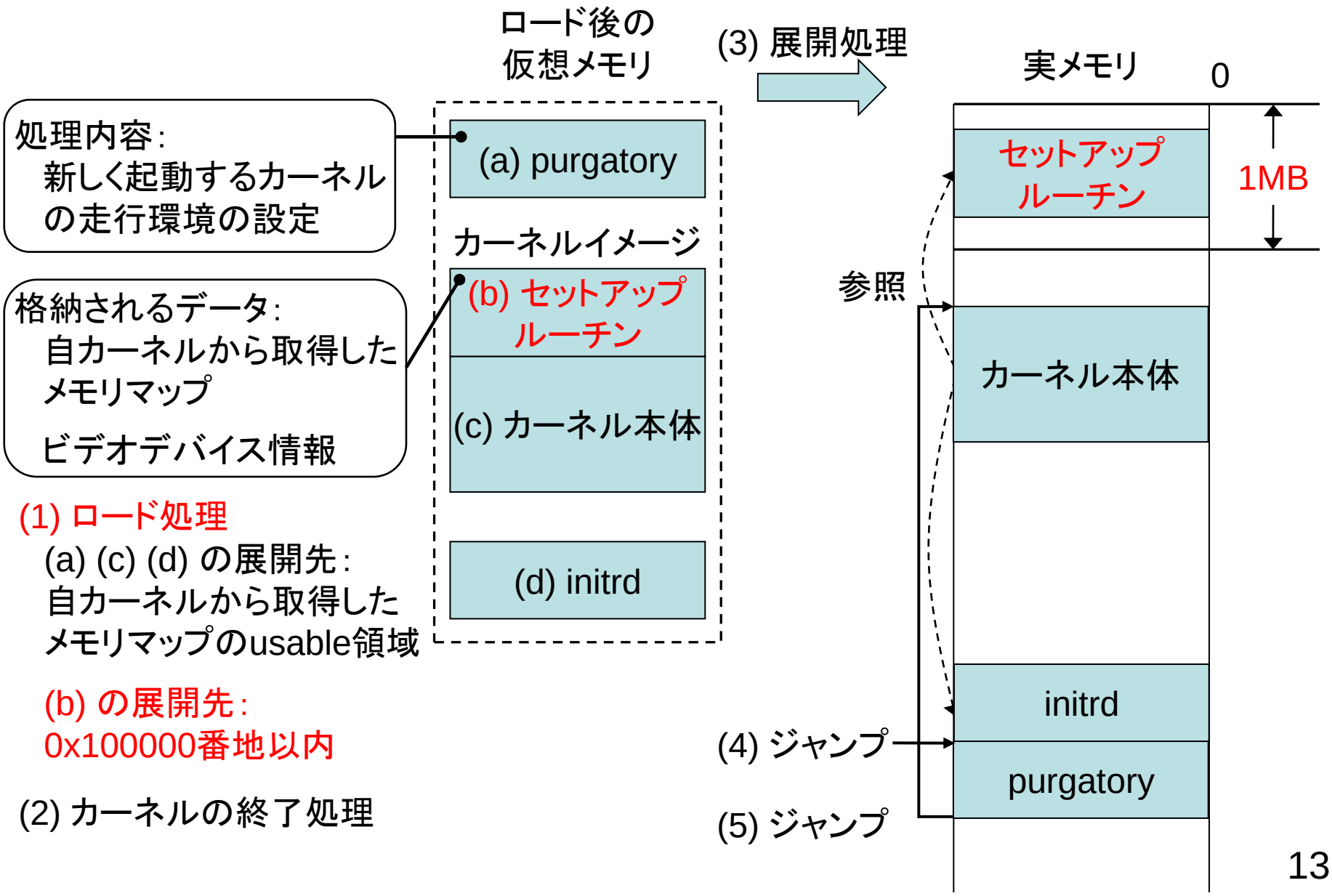
Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
カーネル本体
(c) カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

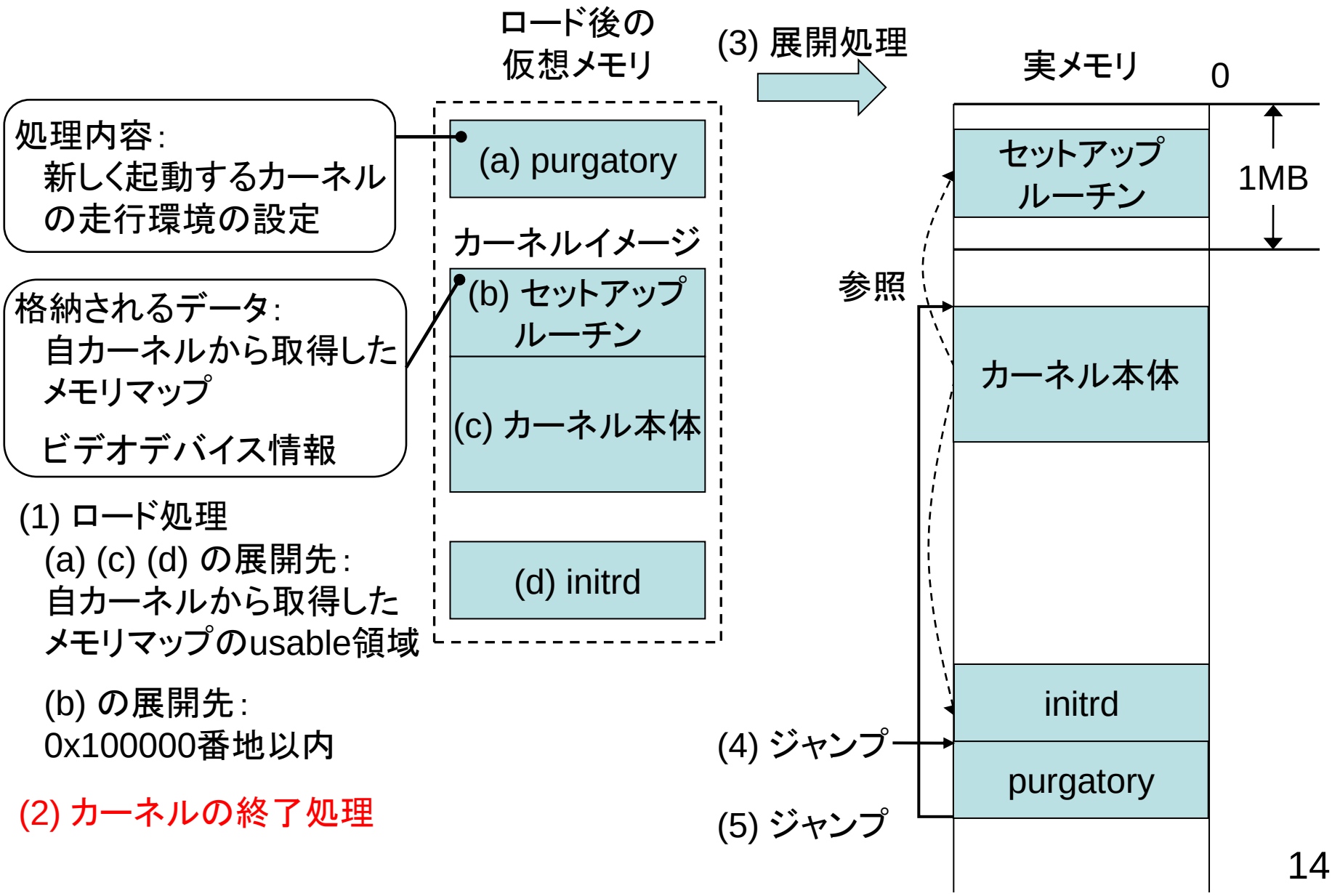
Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

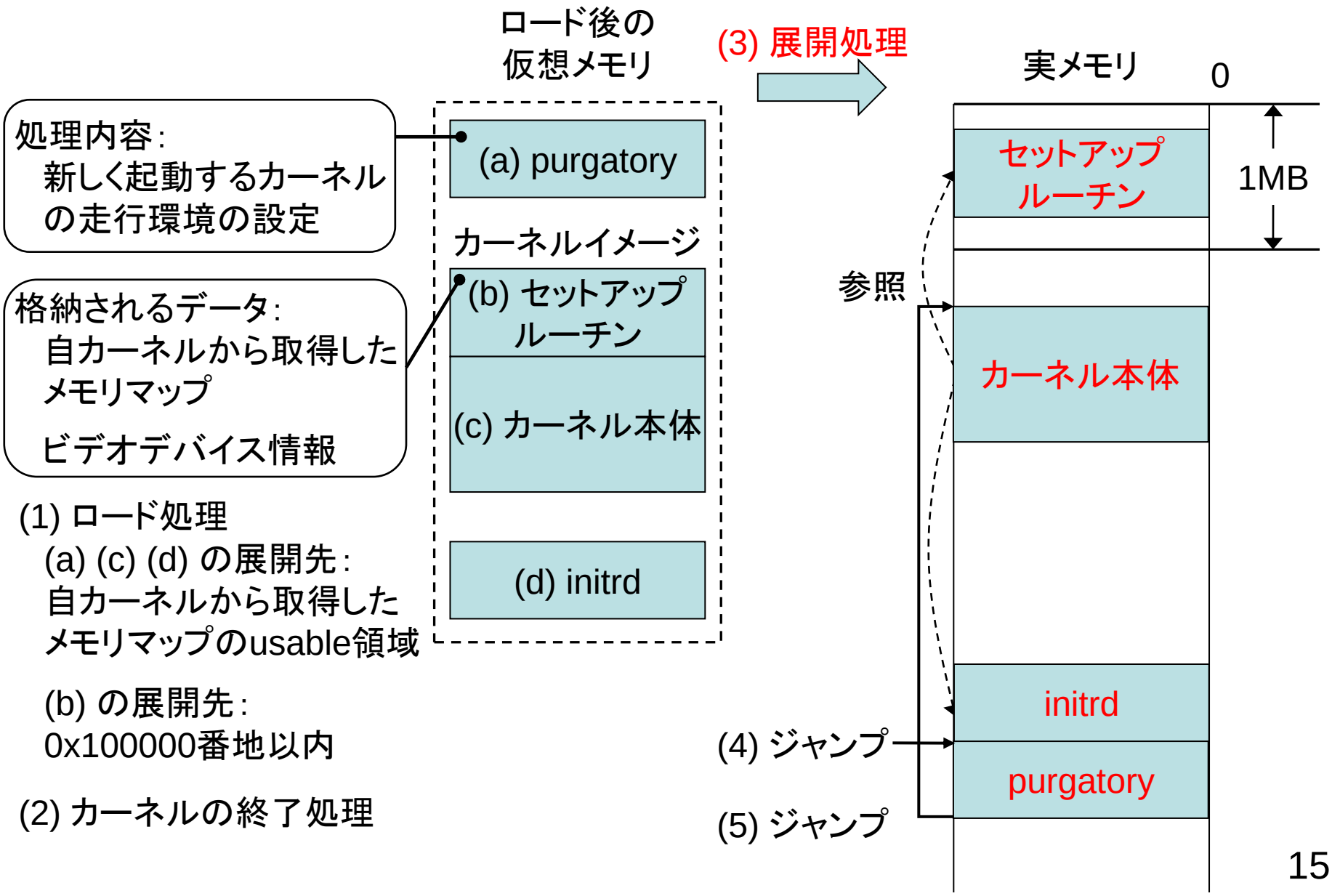
Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
カーネル本体
(c) カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

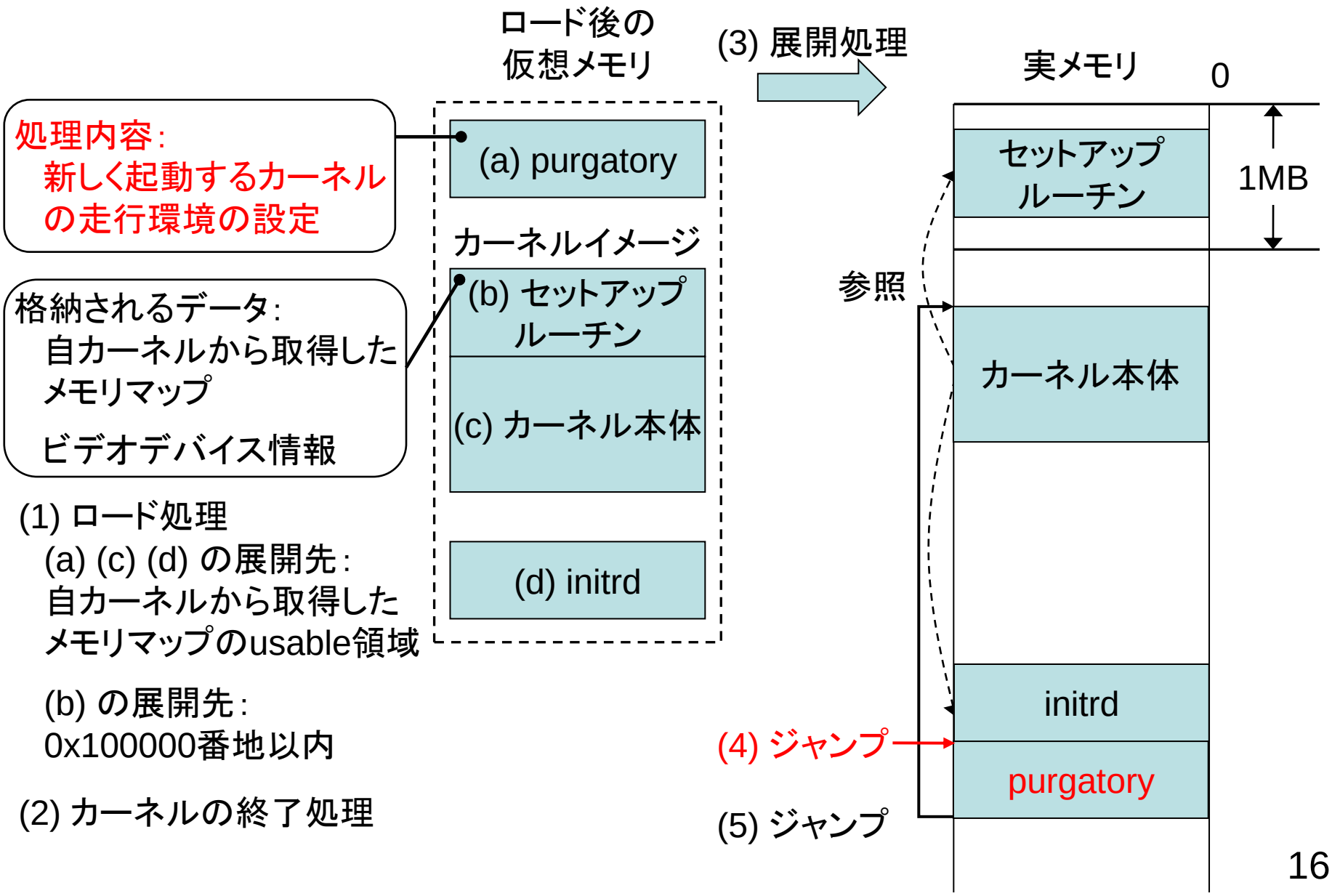
Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

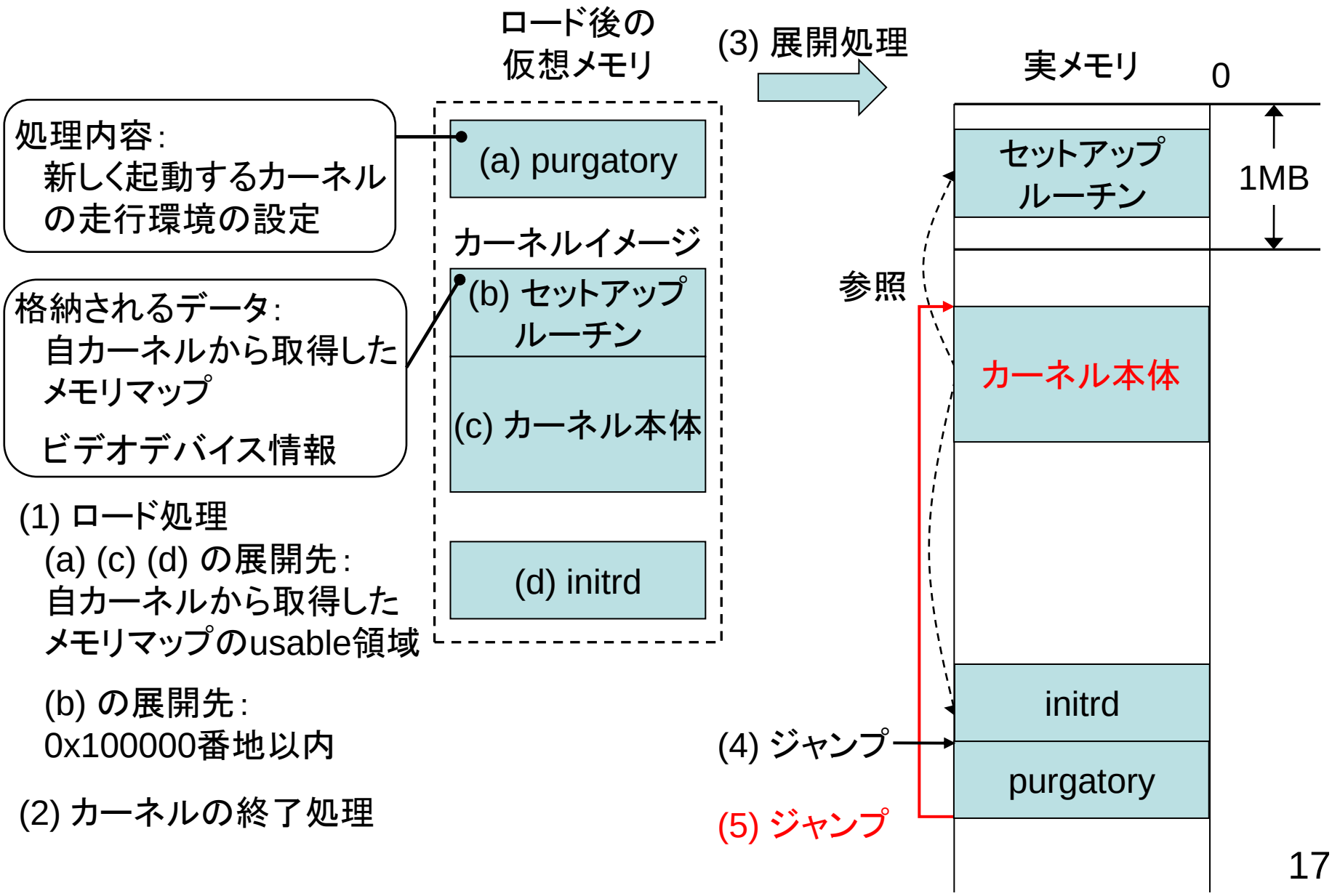
Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

Kexecの処理流れ


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
カーネル本体
(c) カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

Kexec改変時の課題

Kexecは，カーネルを再起動する機能

⇒ Kexecを後続OSの起動用に改変する必要

(課題1) コアの起動

再起動するのではなく，別のコアを起動する必要

(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

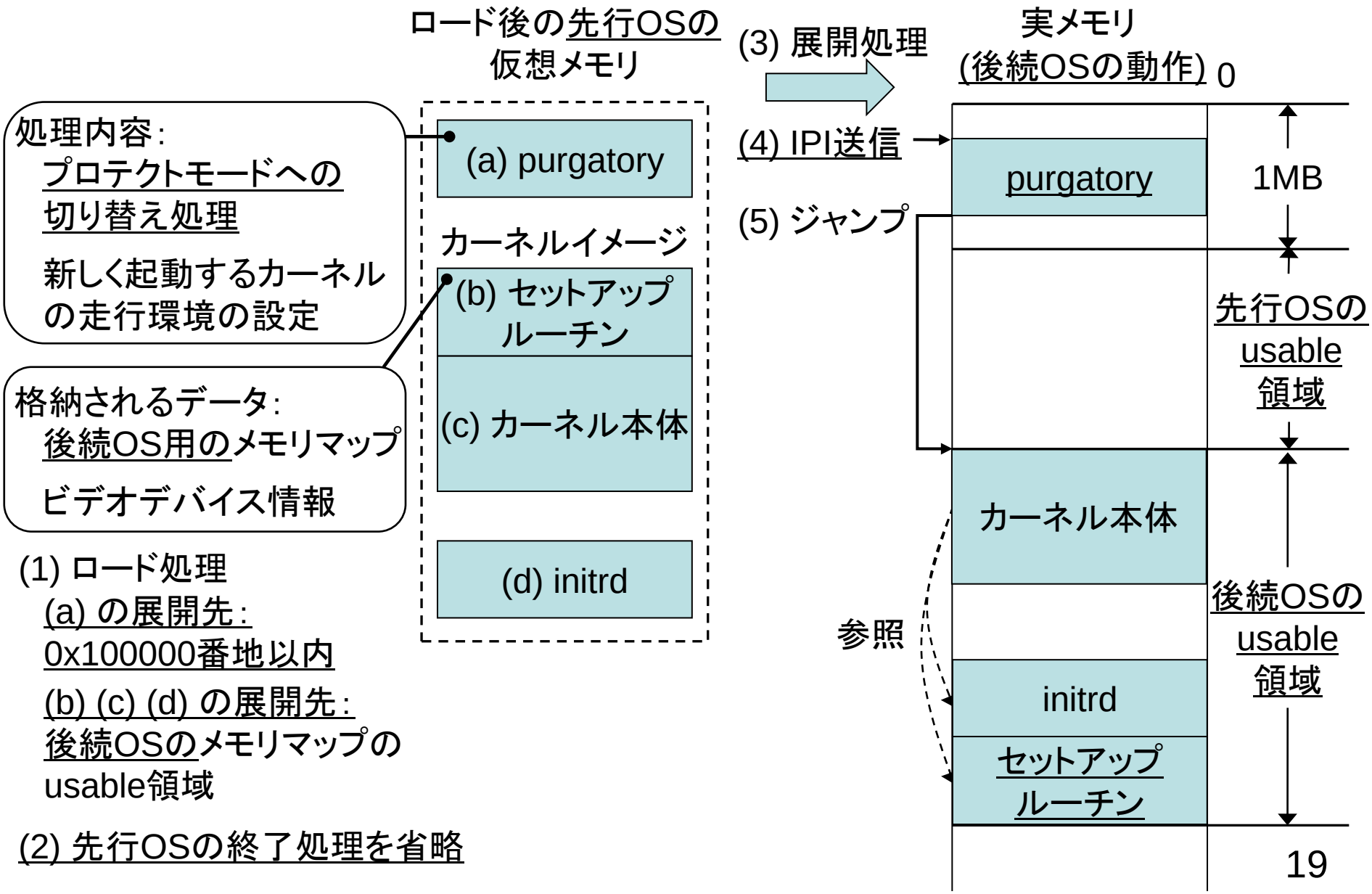
(課題3) 後続OS用のメモリマップの用意

後続OSにあわせた個別のメモリマップが必要

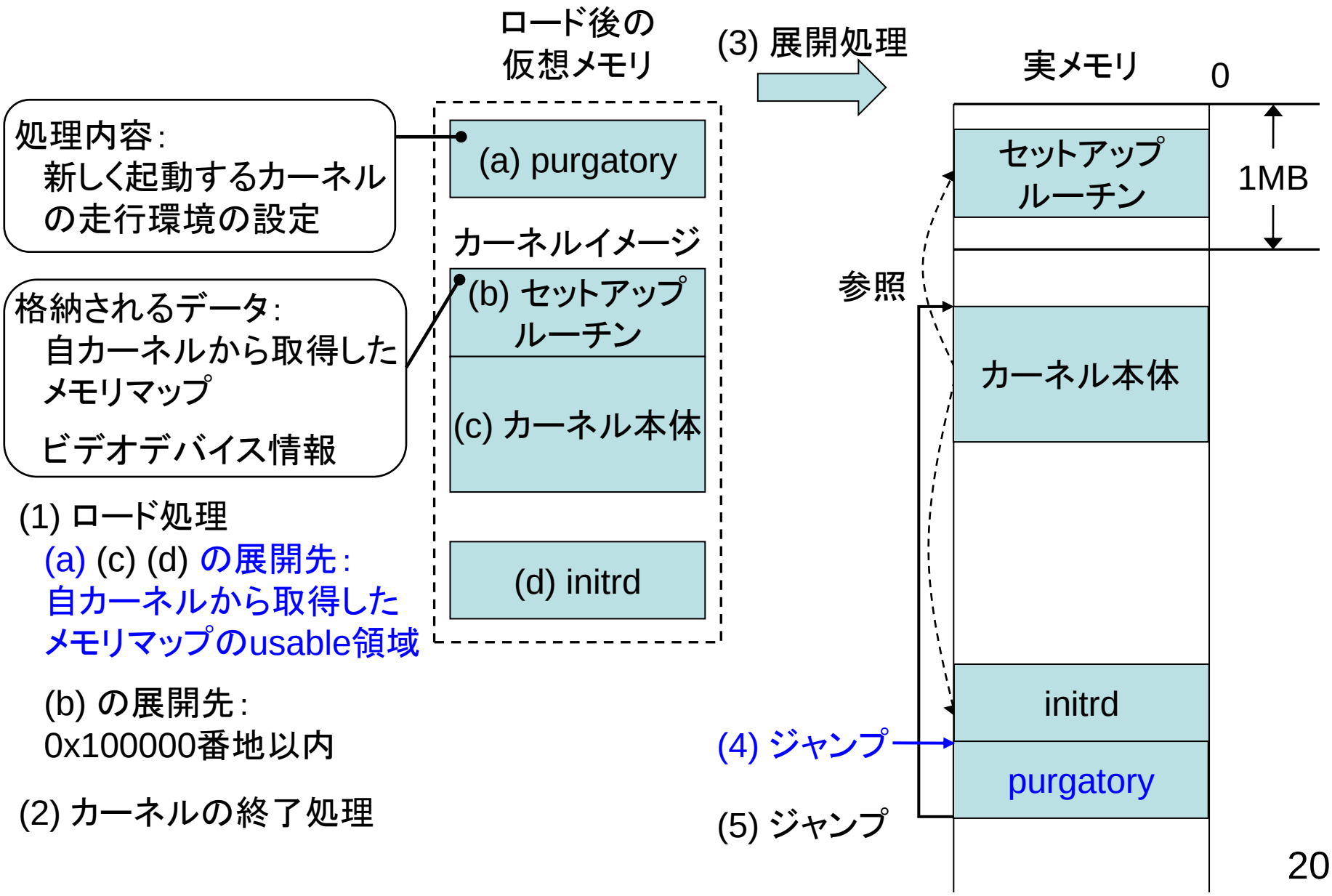
(課題4) 先行するカーネルの走行環境の保護

カーネルの終了処理を省略する必要

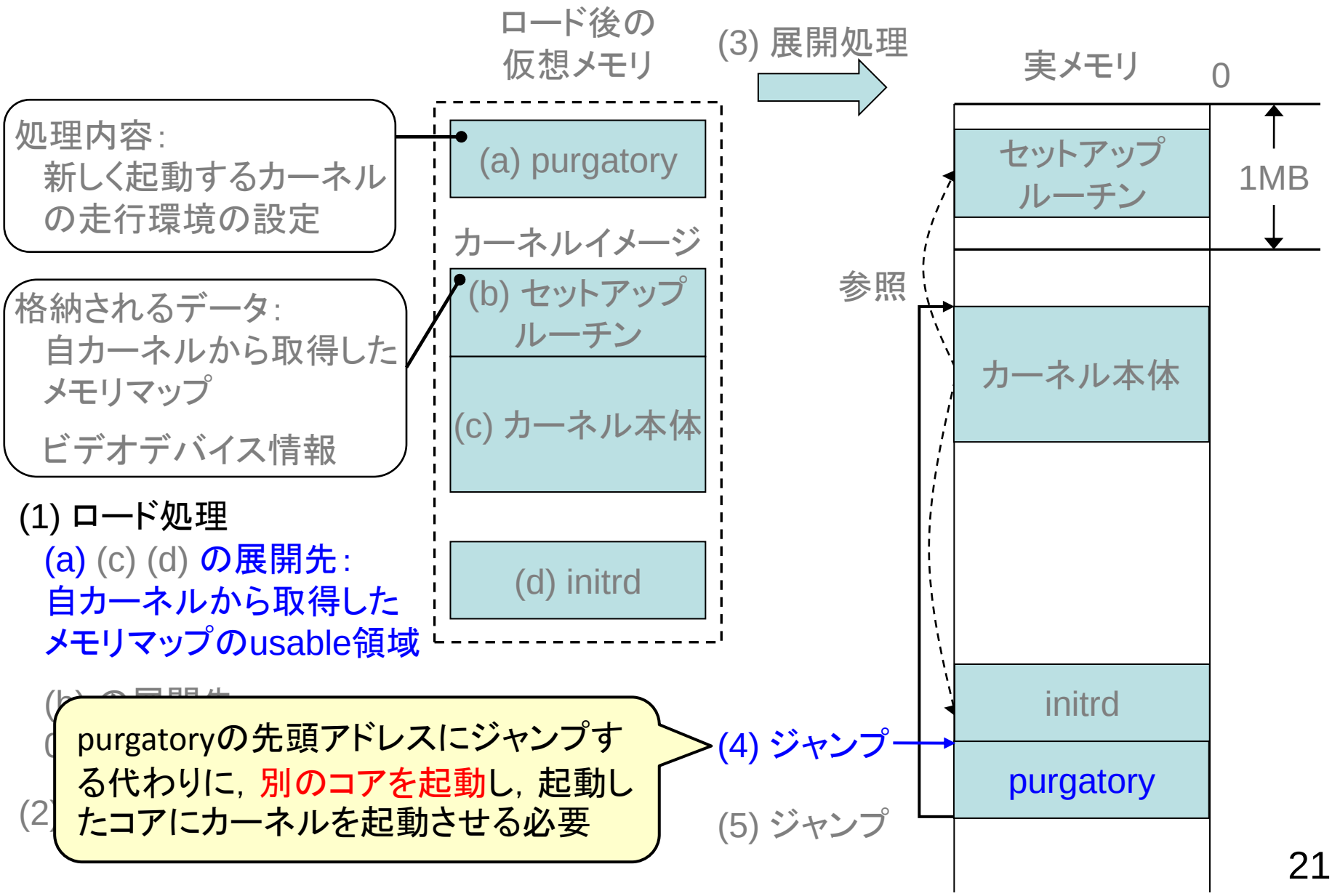
Kexecを利用した後続OS起動方式の処理流れ



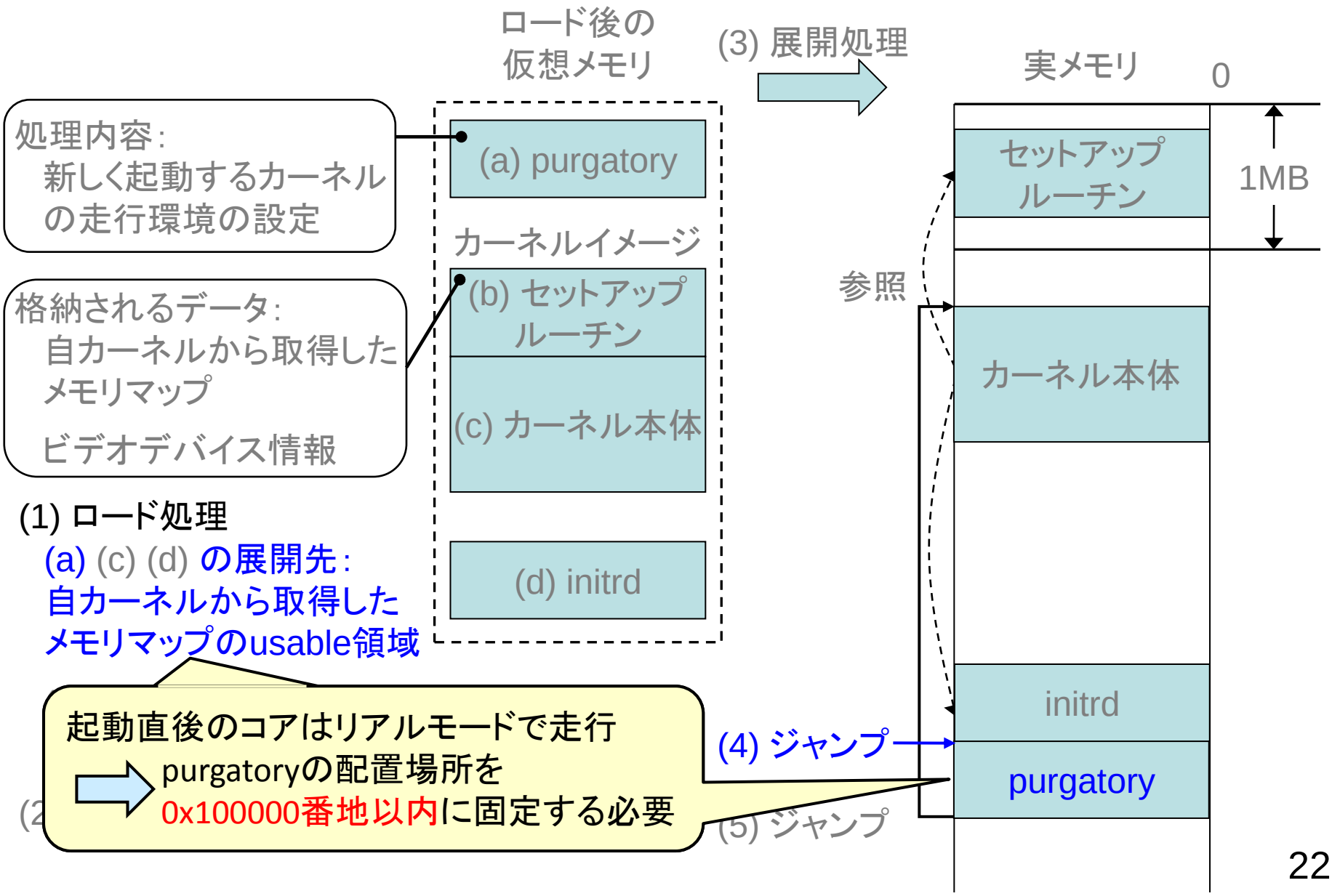
(課題1) コアの起動


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
(b) の展開先:
0x100000番地以内
(5) ジャンプ
(2) カーネルの終了処理

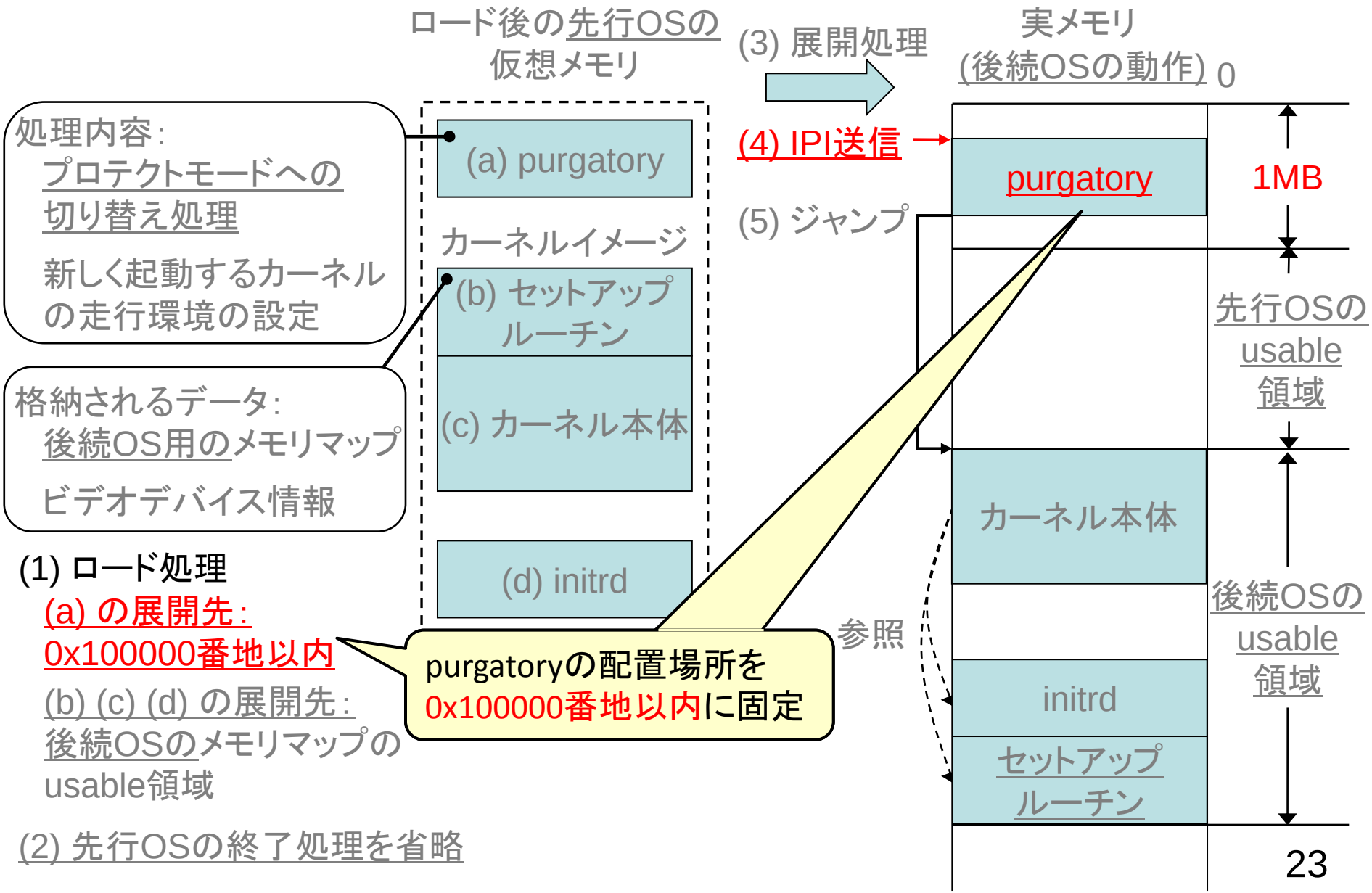
(課題1) コアの起動


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
purgatoryの先頭アドレスにジャンプする代わりに，別のコアを起動し，起動したコアにカーネルを起動させる必要
(4) ジャンプ
purgatory
(5) ジャンプ

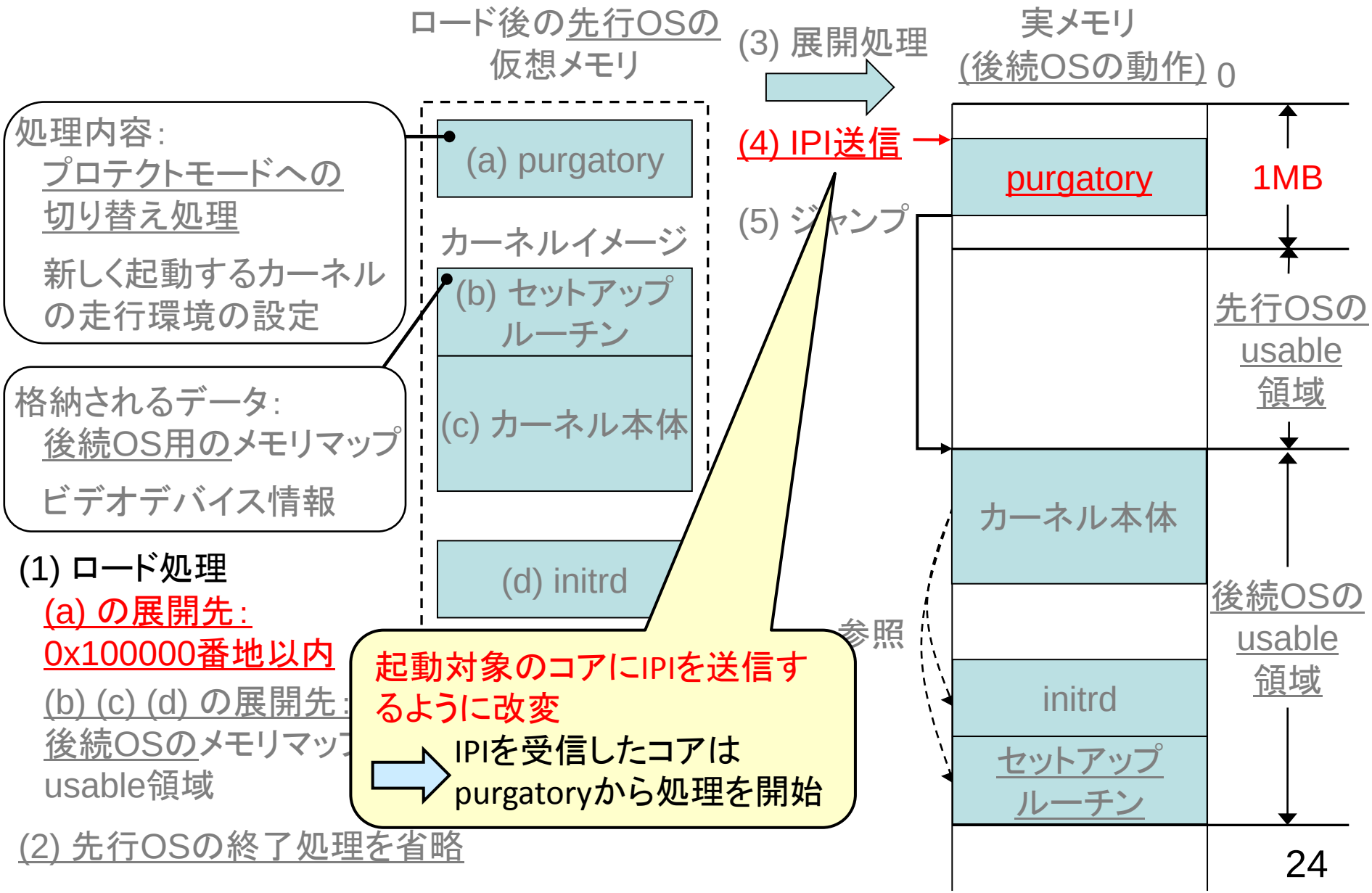
(課題1) コアの起動


ロード後の
仮想メモリ
(3) 展開処理
実メモリ
0
処理内容:
新しく起動するカーネル
の走行環境の設定
(a) purgatory
セットアップ
ルーチン
1MB
カーネルイメージ
参照
格納されるデータ:
自カーネルから取得した
メモリマップ
ビデオデバイス情報
(b) セットアップ
ルーチン
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) (c) (d) の展開先:
自カーネルから取得した
メモリマップのusable領域
(d) initrd
initrd
(4) ジャンプ
purgatory
起動直後のコアはリアルモードで走行
purgatoryの配置場所を
0x100000番地以内に固定する必要
(5) ジャンプ

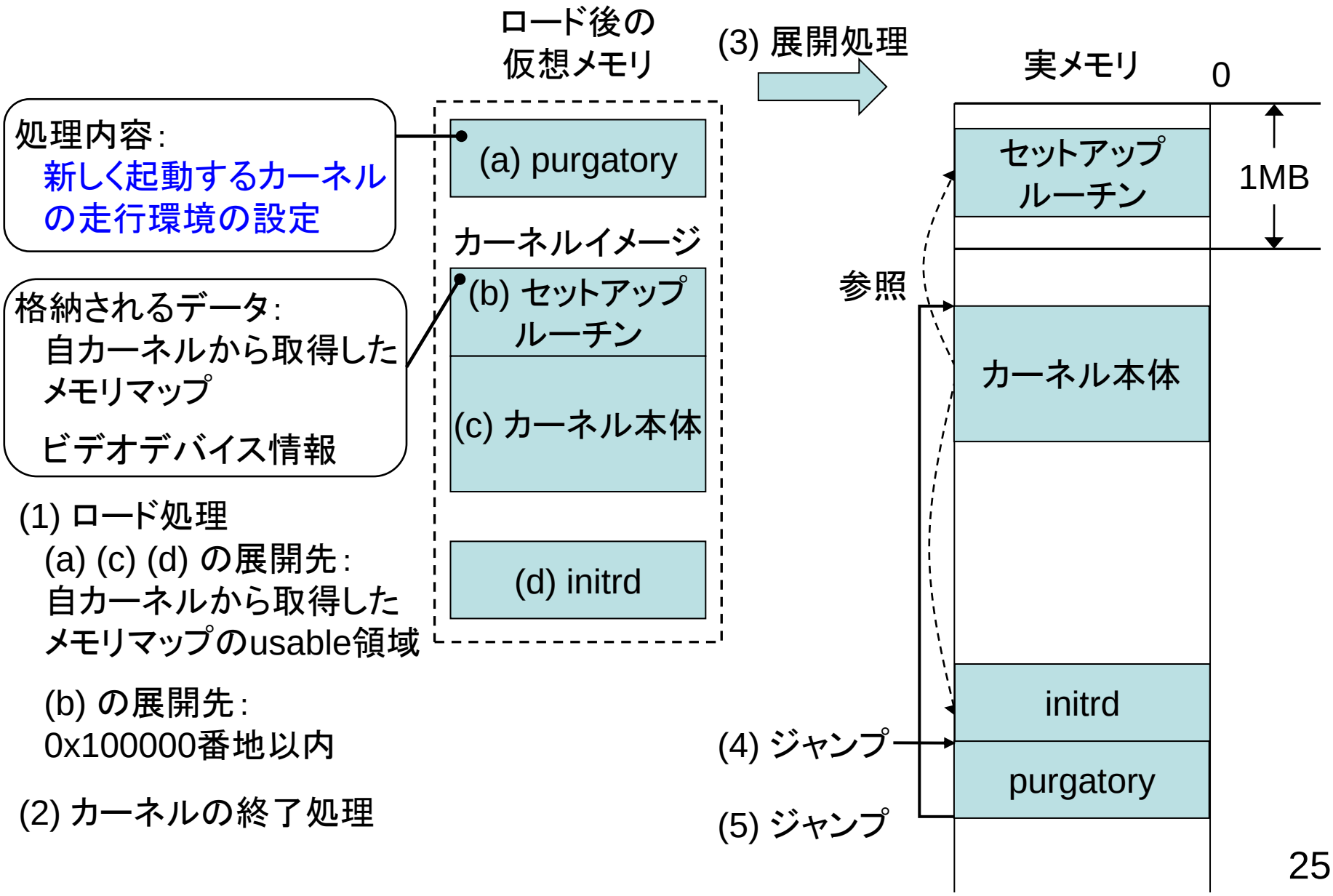
(対処1) IPIの送信によるコアの起動



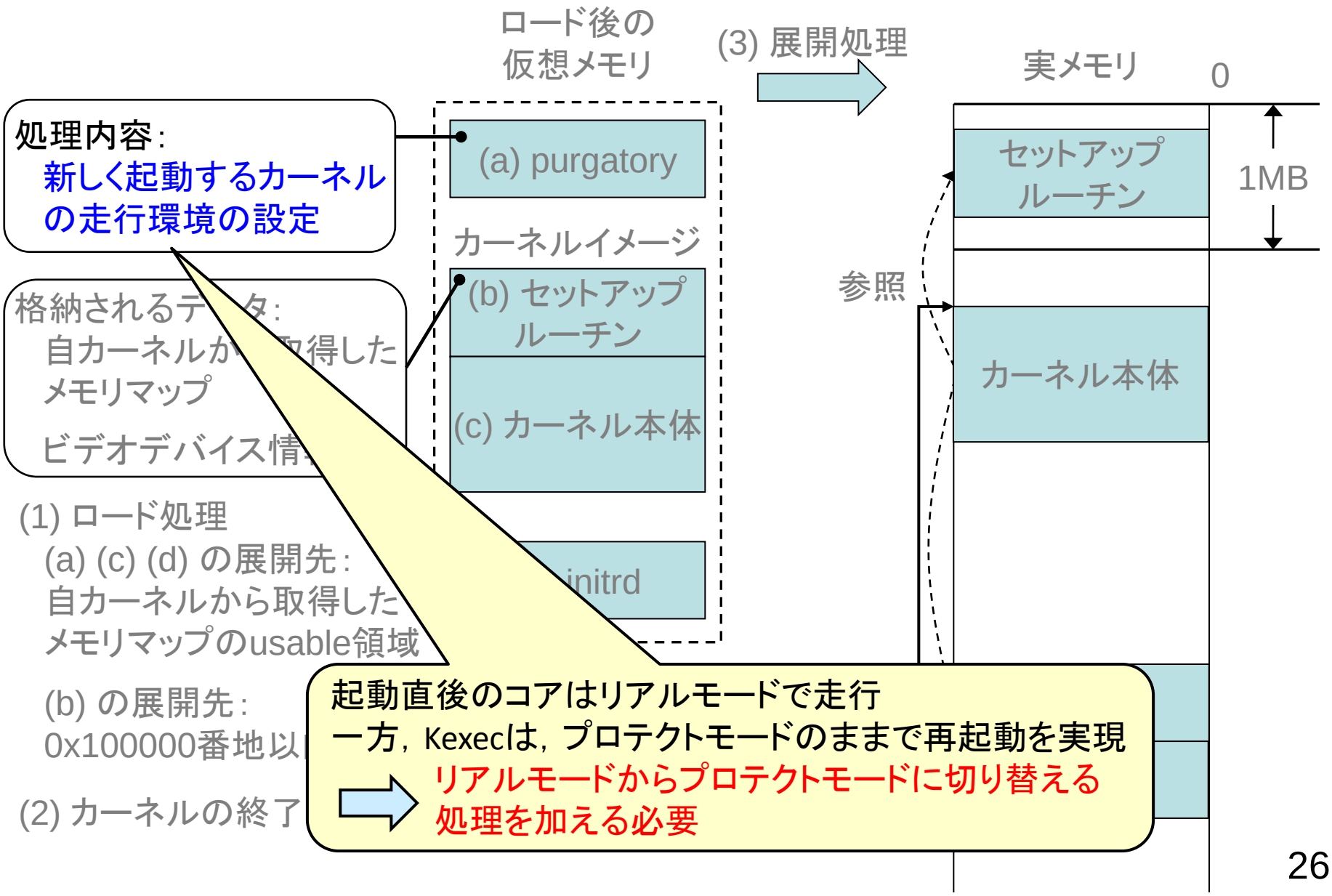
(対処1) IPIの送信によるコアの起動



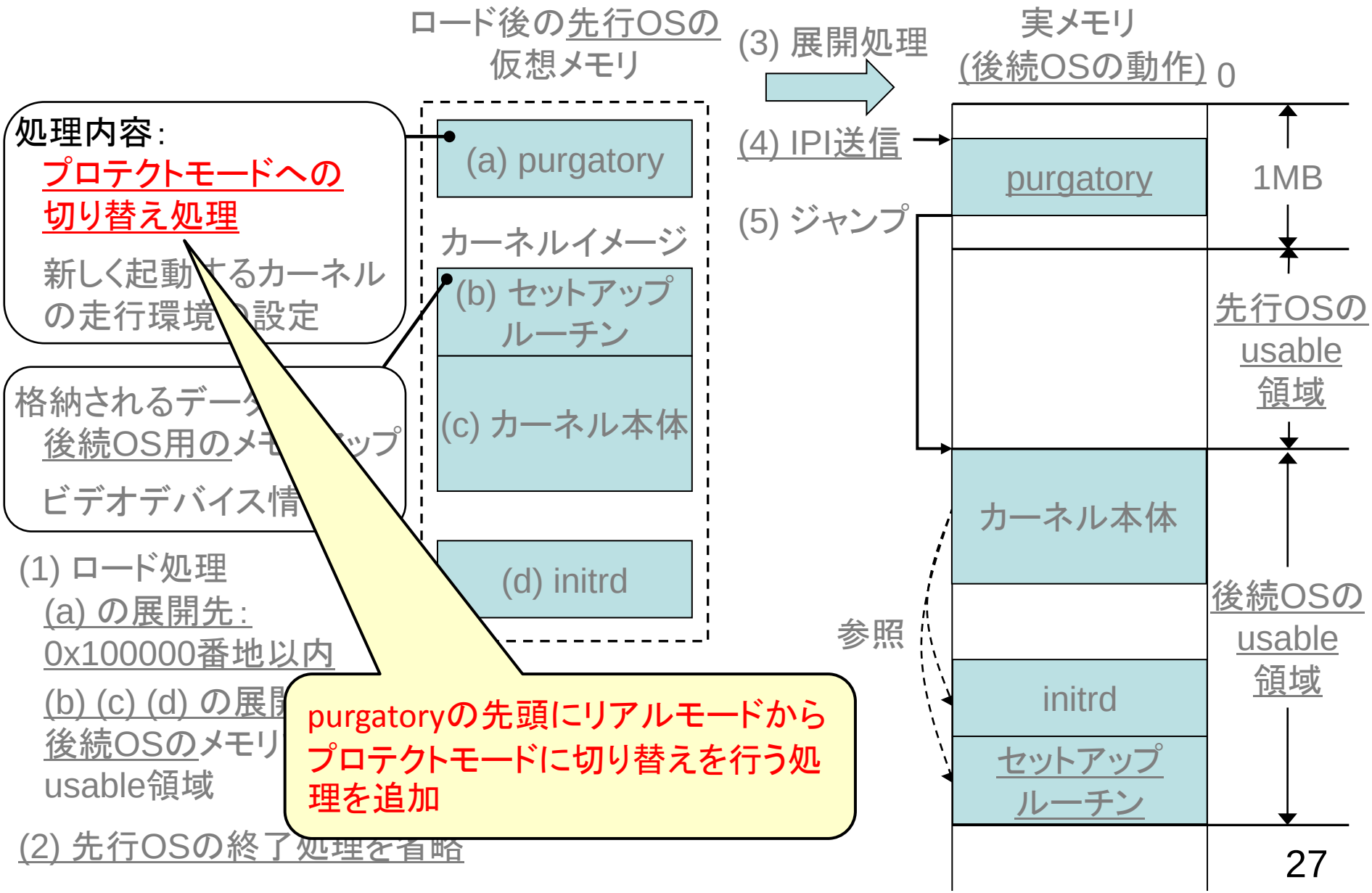
(課題2) プロテクトモードへの切り替え



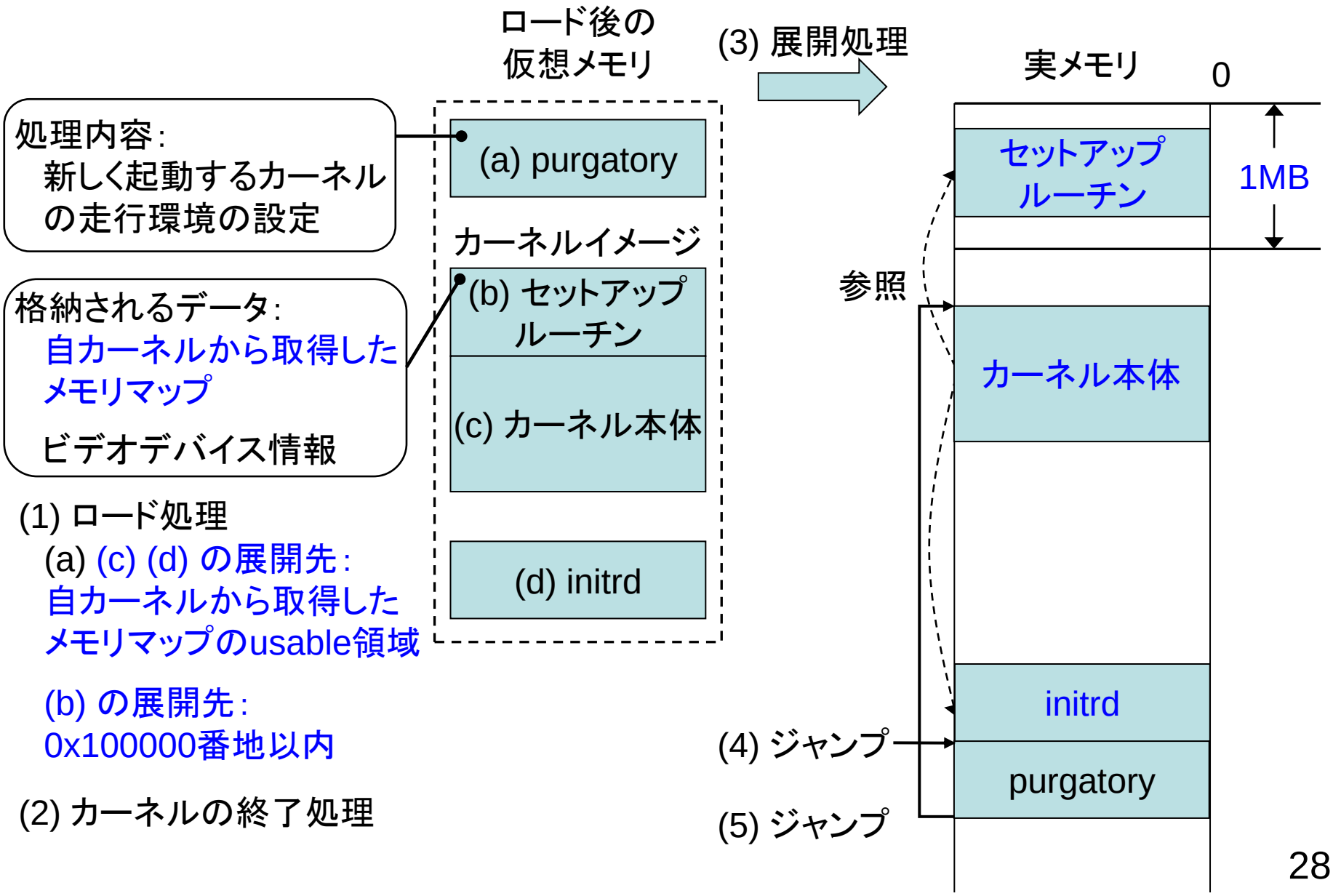
(課題2) プロテクトモードへの切り替え



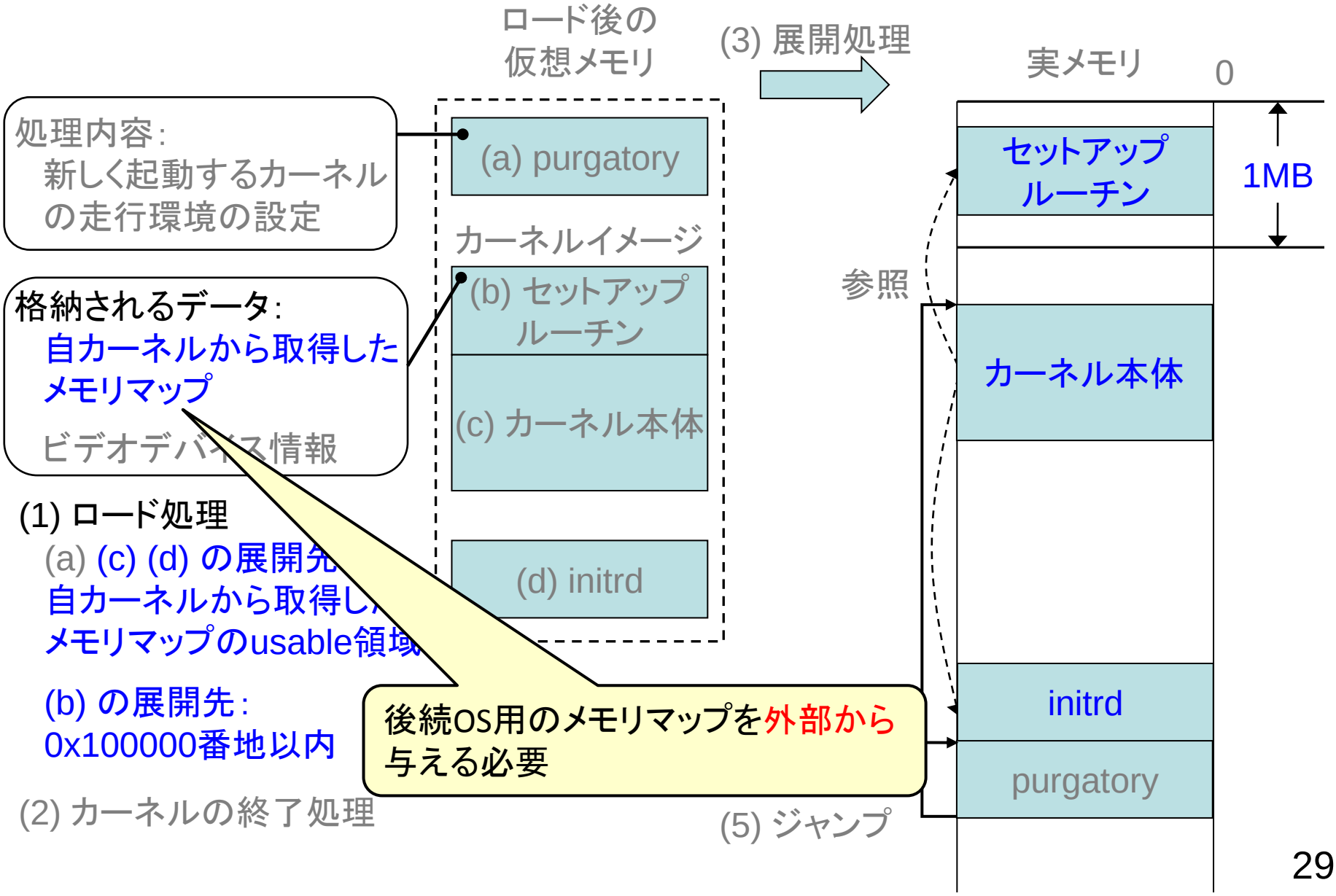
(対処2) プロテクトモードへの切り替え処理の追加



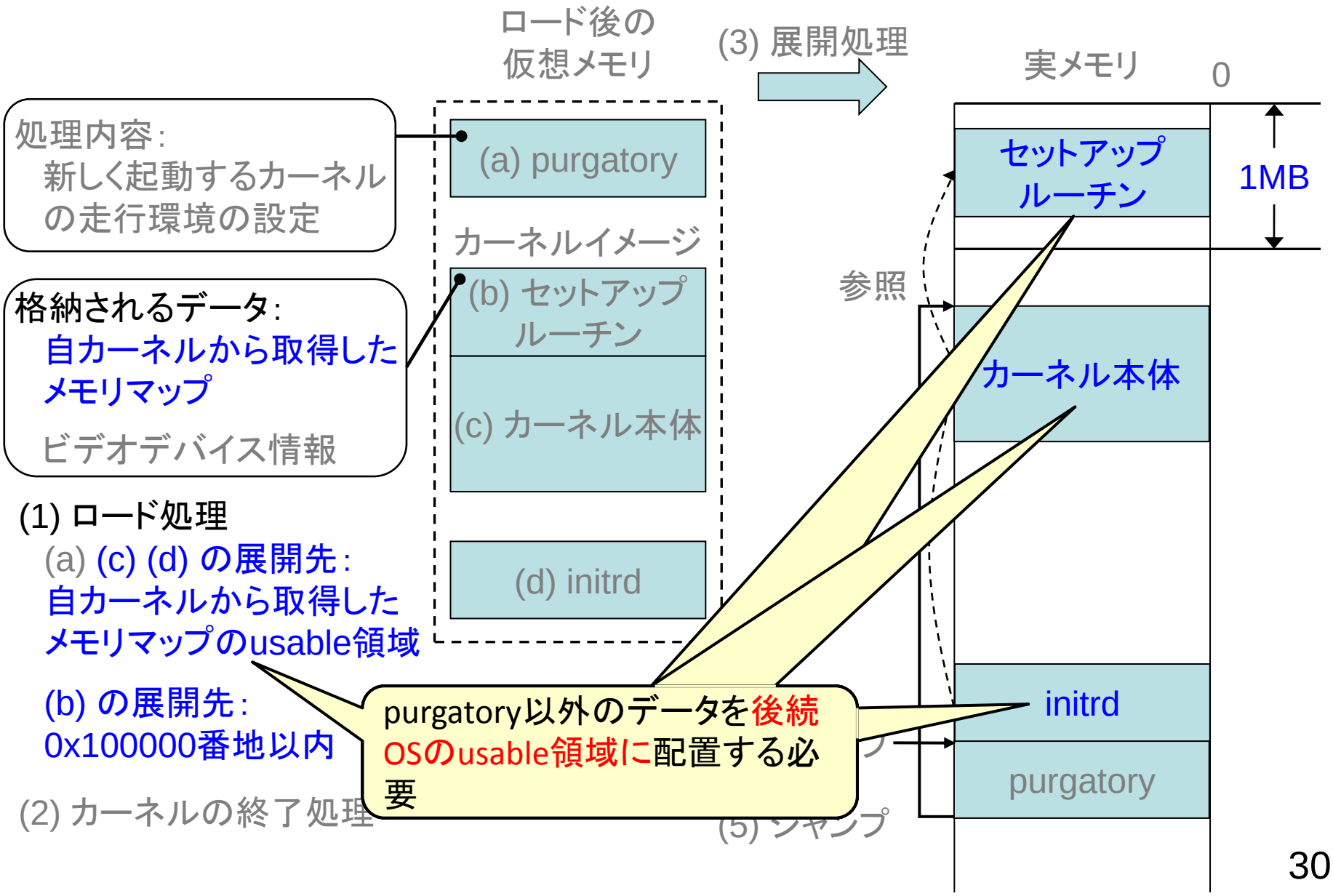
(課題3) 後続OS用のメモリマップの用意



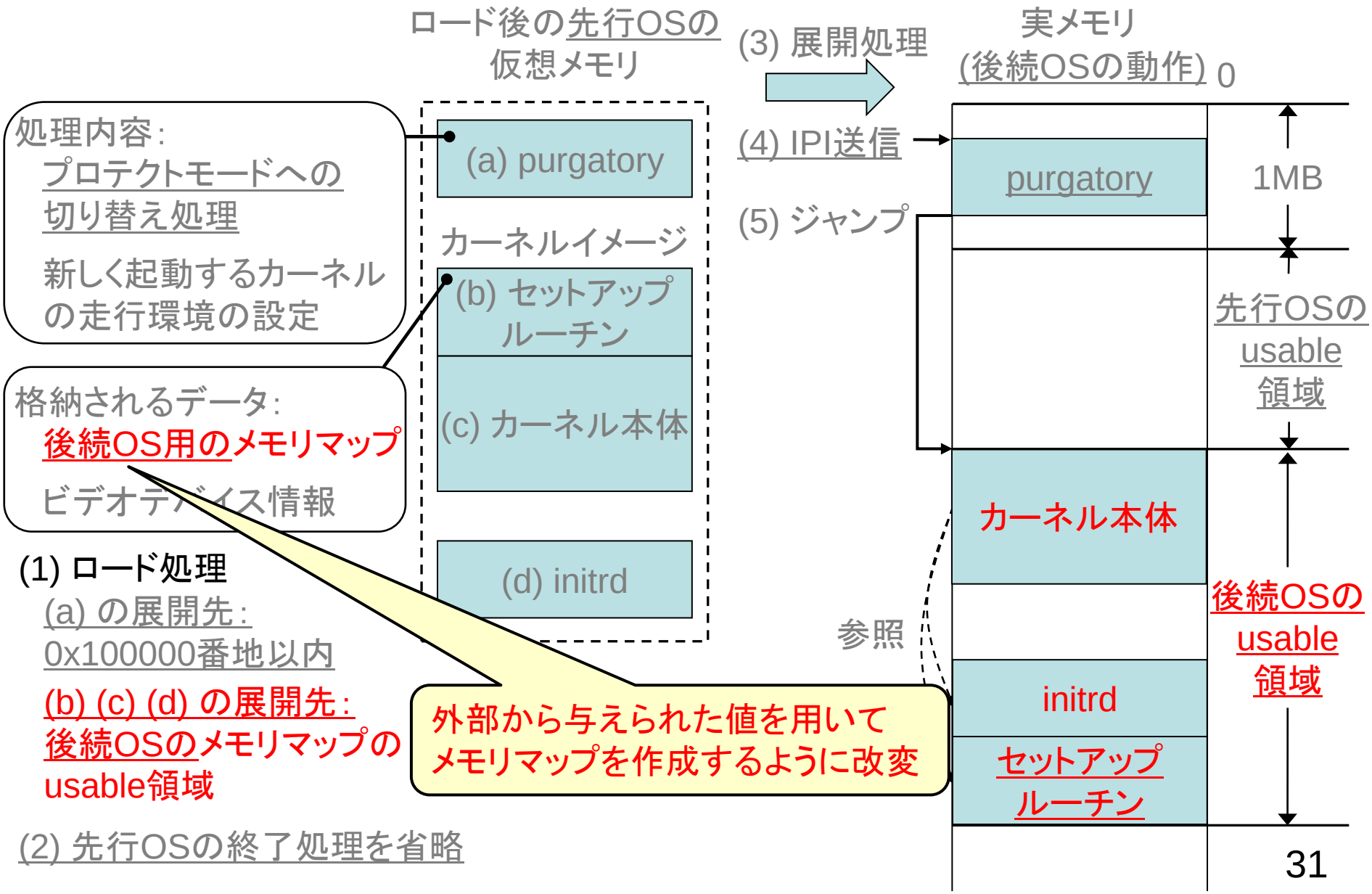
(課題3) 後続OS用のメモリマップの用意



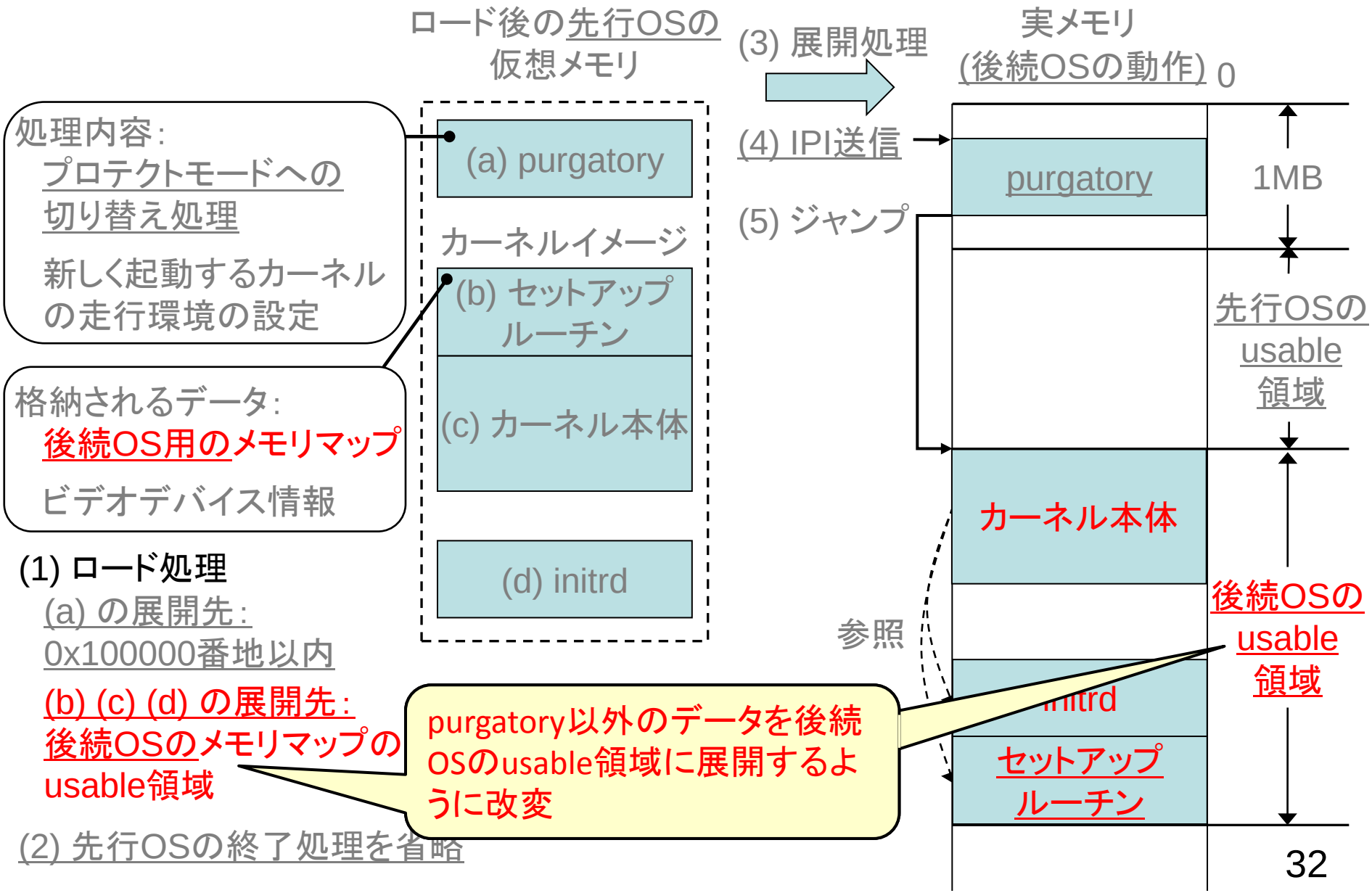
(課題3) 後続OS用のメモリマップの用意



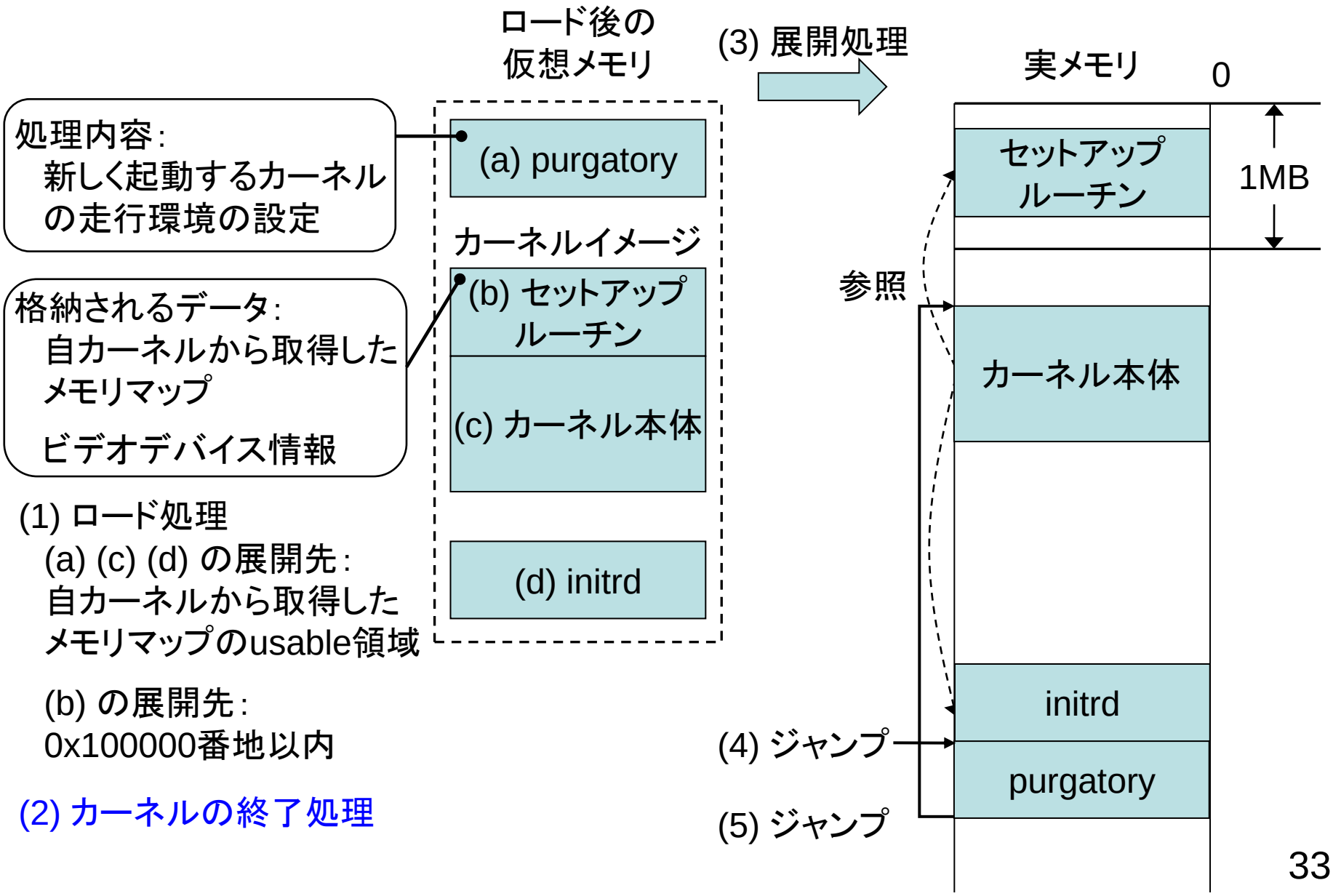
(対処3) メモリマップの改変



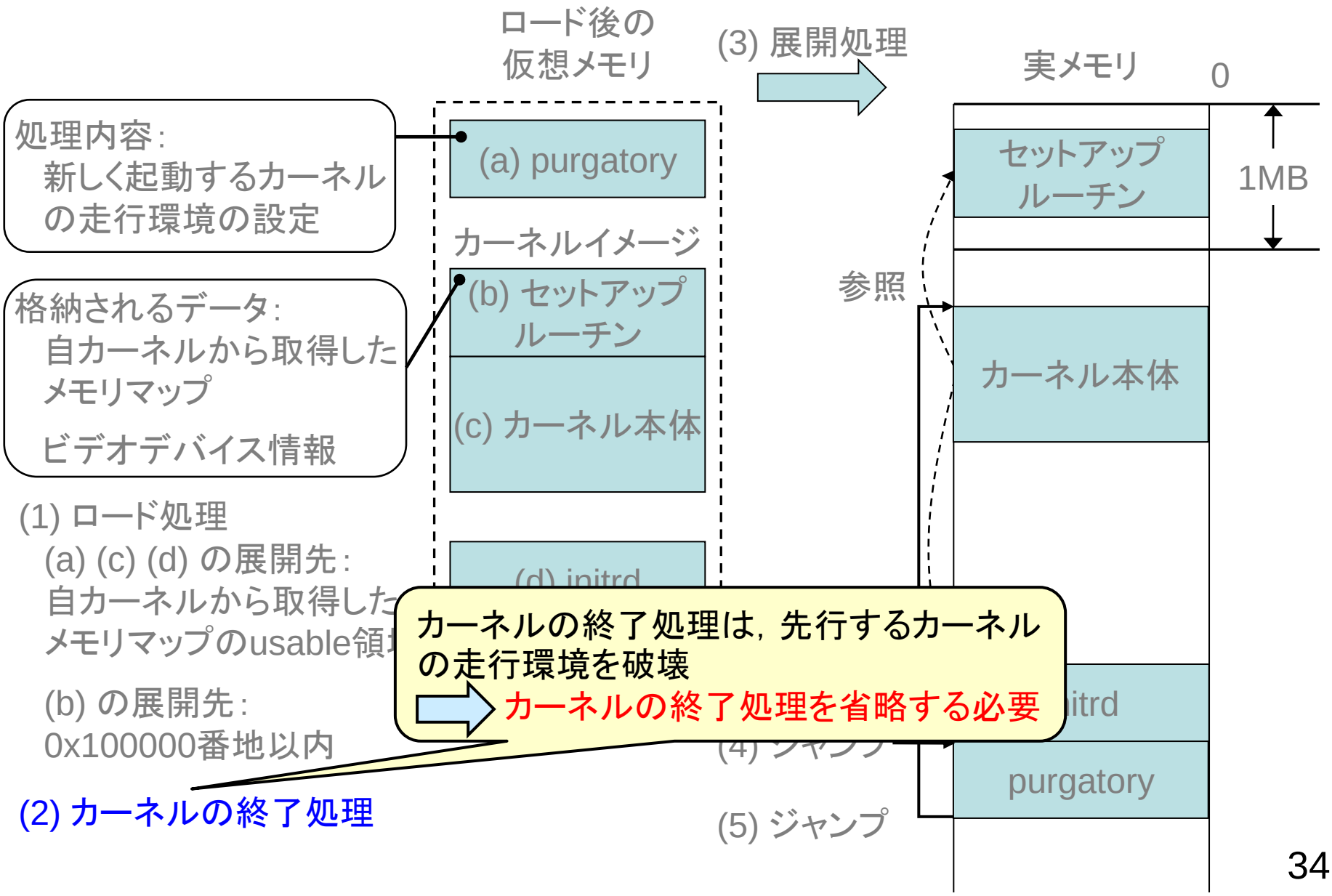
(対処3) メモリマップの改変



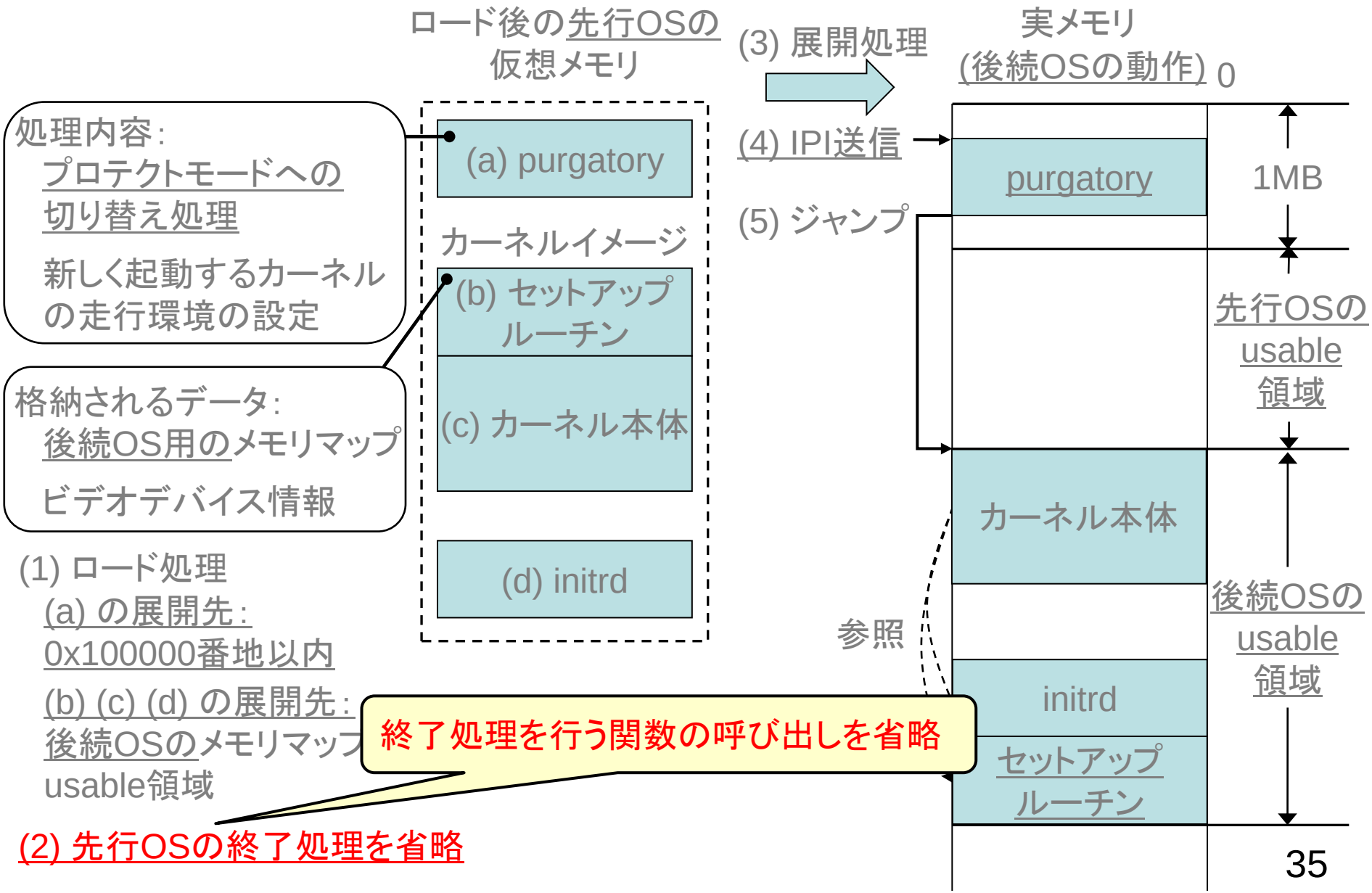
(課題4) 先行するカーネルの走行環境の保護



(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


ロード後の先行OSの
仮想メモリ
(3) 展開処理
実メモリ
(後続OSの動作)
0
処理内容:
プロテクトモードへの
切り替え処理
新しく起動するカーネル
の走行環境の設定
(a) purgatory
(4) IPI送信
purgatory
1MB
(5) ジャンプ
カーネルイメージ
(b) セットアップ
ルーチン
先行OSの
usable
領域
格納されるデータ:
後続OS用のメモリマップ
ビデオデバイス情報
(c) カーネル本体
カーネル本体
(1) ロード処理
(a) の展開先:
0x100000番地以内
(d) initrd
参照
後続OSの
usable
領域
(b) (c) (d) の展開先:
後続OSのメモリマップ
usable領域
initrd
終了処理を行う関数の呼び出しを省略
セットアップ
ルーチン
(2) 先行OSの終了処理を省略

目次

カーネルイメージの単一化

＜課題＞

Mintの各OSの差分は，占有するハードウェア資源のみ

⇨ 各OS起動時に占有するハードウェア資源を指定する必要

＜起動時に使用可能なパラメータ＞

(1) 先行OSはブートローダから起動

⇨ ブートオプションを使用可能

(2) 後続OSは先行OSからKexecを利用して起動

⇨ Kexecの実行オプションとブートオプションを使用可能

＜指定が必要なハードウェア資源＞

CPU, メモリ, 入出力機器

(カーネルの展開先アドレス, 利用するメモリ領域)

CPUの指定方法

＜指定対象と指定方法＞

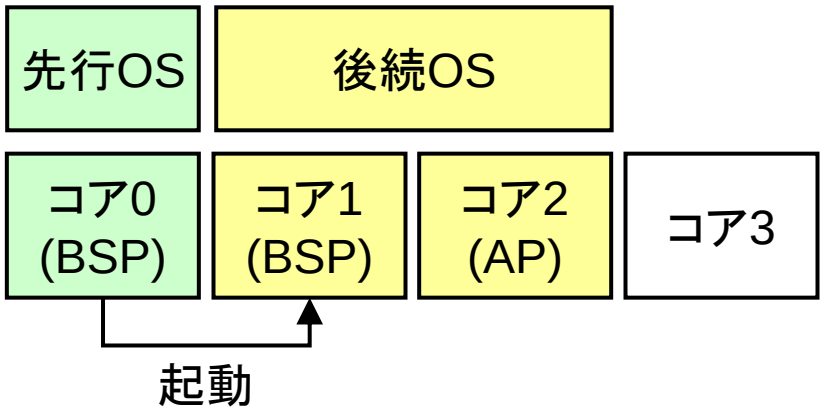
(A) BSP(起動対象のコア)

先行OS：自動的に先頭のコア(コア0)になるため，指定しない

後続OS：独自にKexecに追加したオプションを用いて指定

(B) 占有するコアの個数

Linuxの既存機能maxcpusブートオプションを用いて指定



先行OS

(A) BSP: 指定無し

(B) maxcpus: 1

後続OS

(A) BSP: 1

(B) maxcpus: 2

カーネルの展開先アドレスの指定方法

通常，カーネルの展開先アドレスは，カーネルのビルド時に確定

再配置可能な圧縮カーネルイメージを利用

圧縮カーネルイメージが配置されたアドレスから展開

＜圧縮カーネルイメージの配置方法＞

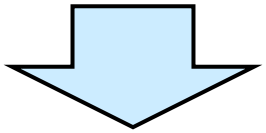
(1) 先行OS

ブートローダによってメモリの先頭(0x100000)付近に配置

(2) 後続OS

Kexecの既存機能mem-minオプションを使用

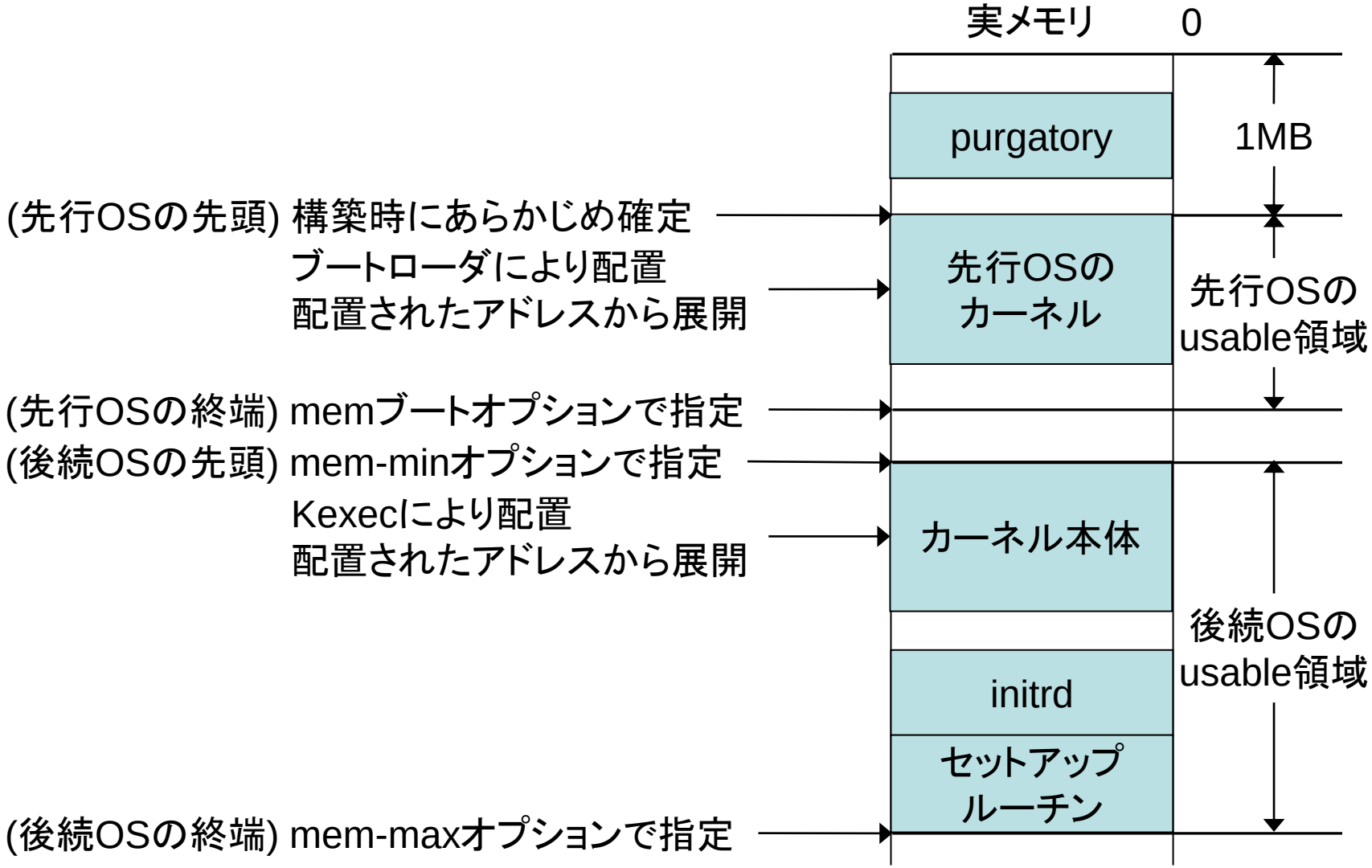
Kexecのデータ展開先の先頭アドレスを指定可能



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

利用するメモリ(usable)領域の先頭アドレスと終端アドレスを指定



入出力機器の指定方法

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

独自に追加したブートオプションを用いて占有するデバイスを指定

デバイスドライバの登録処理を行う部分は，全てこのブートオプションの値によって処理を分岐

目次

(評価1) コード改変量

＜カーネルの改変量＞

	全体	起動処理	ファイル数	hunk数
従来方式	334行	146行	11	15
Kexec方式	257行	69行	6	14
差(削減割合)	77行(23.1%)	77行(52.7%)	5	1

＜ブート用APの改変量＞

従来方式	167行(新規作成)
Kexec方式	117行(全体:10047行)
差(削減割合)	50行(29.9%)

(1) Mint全体のコード改変量を約23.1%削減

(2) ブート用APのコード改変量を約29.9%削減

(3) カーネルのコード改変量は，ブート用APよりも少ない

(評価2) 起動処理に要する時間

＜評価方法＞

(A) Intel Core i7(2.8GHz)を搭載した計算機を利用

(B) TSCレジスタの値を取得して時間を測定

(C) 電源投入もしくはAP実行からkernel_initスレッド実行までを測定

＜評価結果＞

(1) 未改変のLinuxの起動時間	24.37秒
(2) 未改変のLinuxのKexecによる再起動時間	2.17秒
(3) 改変版KexecによるMintの後続OS起動時間	0.60秒

(A) (1)と(2)の差は，BIOS，ブートローダ，およびセットアップルーチンを走行しないことによる

(B) (2)と(3)の差は，カーネル終了処理を行わないことによる

おわりに

＜Kexecを利用した後続OS起動方式＞

Kexecは，カーネルを再起動する機能

⇨ Kexecを後続OSの起動用に改変

＜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

＜評価＞

(1) コード改変量を削減できることを確認

(2) 起動に要する時間を短縮できることを確認