

Dockerで構築したコンテナの 通信性能評価

平成30年2月16日

岡山大学 工学部 情報系学科

橋本 鉄平

Dockerとは

アプリケーション(AP)の実行環境構築にDockerを用いる手法が普及

<Dockerとは>

(1) コンテナと呼ばれる仮想化環境の管理ソフトウェア

リソースやファイルシステム等を制限し、
プロセスを隔離することにより実現

(2) コンテナで動作するAPやライブラリ(イメージ)の管理

(3) イメージをコンテナ内で実行



AP開発の効率化を支援

Dockerの利用により発生する問題

< Dockerによるソフトウェアサービス環境構築 >

- (1) サービスの機能単位でコンテナを構築
- (2) 複数のコンテナを連携


1台の計算機上にブリッジネットワークを構築して実現

メリット：機能拡張の容易性, サービス構成の柔軟性

< 問題 >

コンテナ間の通信量増大によるサービス処理性能低下

∴ コンテナ間の通信オーバーヘッド発生

 「複数コンテナによるサービス構成の柔軟性」
「コンテナ間の通信オーバーヘッド」 } **トレードオフ
の関係**

問題解決のための ネットワーク性能の定量化

<問題解決>

以下の2つの要素について適切に検討

- (1) **粒度** : 1つのコンテナにもたせる機能の細かさ
- (2) **配置** : 複数のサーバに対する各コンテナの配置場所



粒度と配置を検討するための材料の1つとして
ブリッジネットワークの性能を定量化

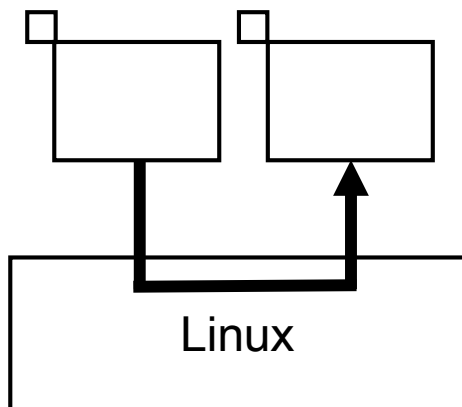
- (測定1) ホスト環境, 同一コンテナ, および異なるコンテナに属するプロセス間の通信スループット
- (測定2) 同一ブリッジネットワークに接続するコンテナの数を増加させた場合における異なるコンテナに属するプロセス間の通信スループット

測定1

ホスト環境, 同一コンテナ, および異なるコンテナに属するプロセス間の通信スループット

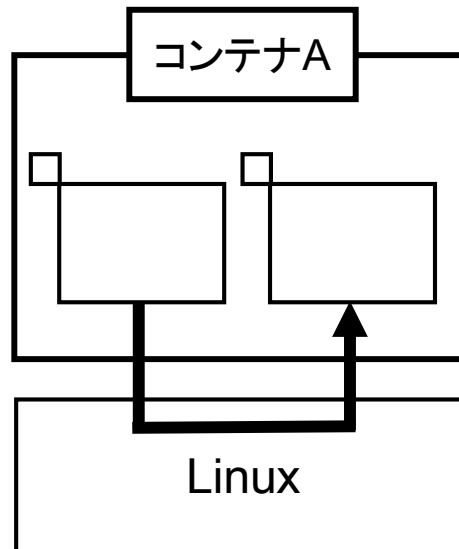
Dockerデーモン **なし**

ホスト環境

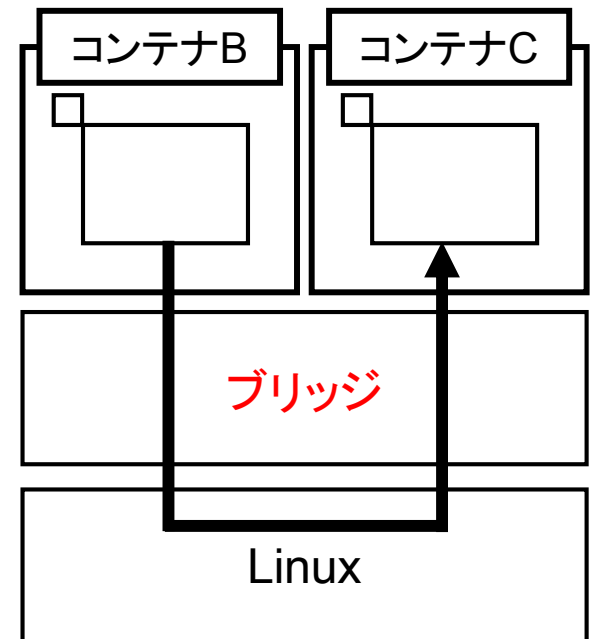


Dockerデーモン **あり**

同一コンテナ



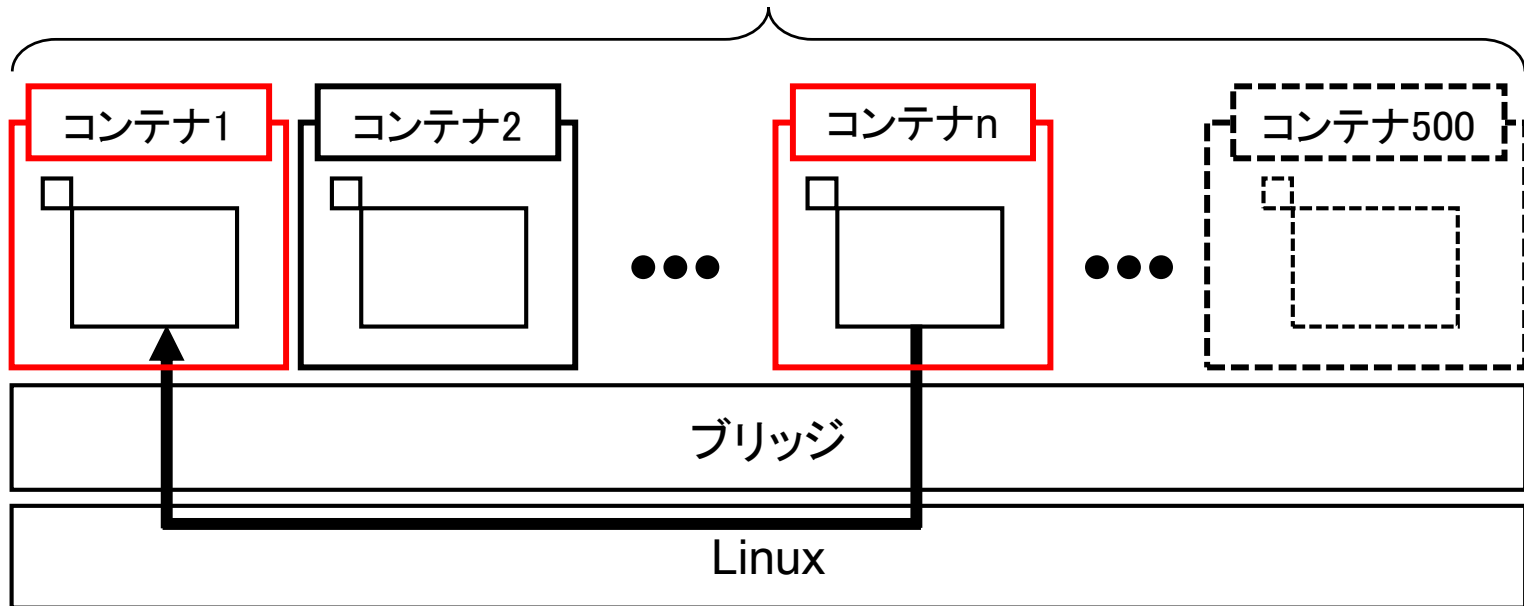
異なるコンテナ



測定2

同一のブリッジネットワークに接続するコンテナの数を増加させた場合における異なるコンテナに属するプロセス間の通信スループット

コンテナ数 n を2から500まで変化



- (1) ARPテーブルのエントリを n 個埋めた状態
- (2) 特定の2コンテナ間で通信を行い, その他のコンテナは通信を行わない状態

測定環境

OS	Debian 8.8
カーネル	Linux kernel 3.16.0-4-amd64
CPU	Intel Core i7-2600(3.40 GHz, 4コア)
メモリ	8.0 GB
Docker	Version 17.03.1-ce
プロトコル	TCP/IP

測定1の結果と評価結果

環境	通信スループット
ホスト環境	62.09 Gbps
同一コンテナ	60.32 Gbps
異なるコンテナ	39.16 Gbps

(1) (2)

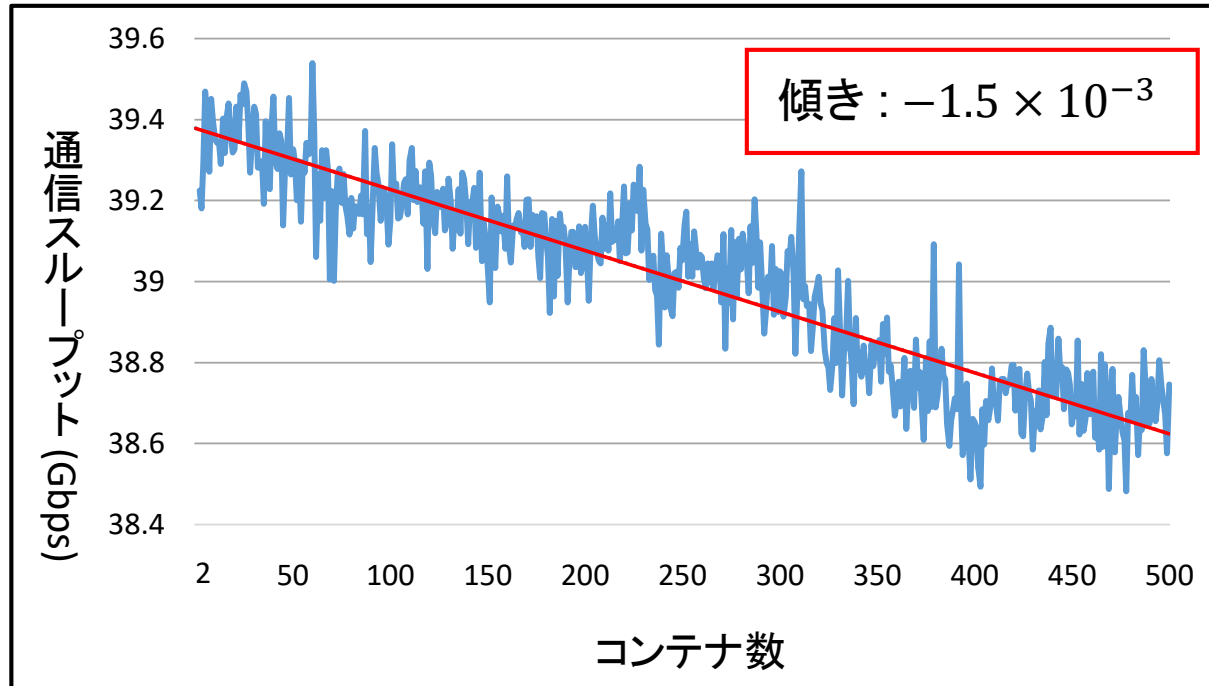
(1) デーモンプロセス起動とコンテナ使用によるオーバーヘッド

 **約3%低下**

(2) コンテナ間通信によるオーバーヘッド

 **約35%低下**

測定2の結果と評価結果



(1) 同一のブリッジネットワークデバイスへ接続するコンテナ数の増加によるオーバヘッド



最大値に対して約0.004% (1.5Mbps) ずつ低下

まとめ

<実績>

Dockerによるサービス環境構築のオーバヘッドの明確化

(1) デーモンプロセス起動とコンテナ使用によるオーバヘッド

 約3%低下

(2) コンテナ間通信によるオーバヘッド

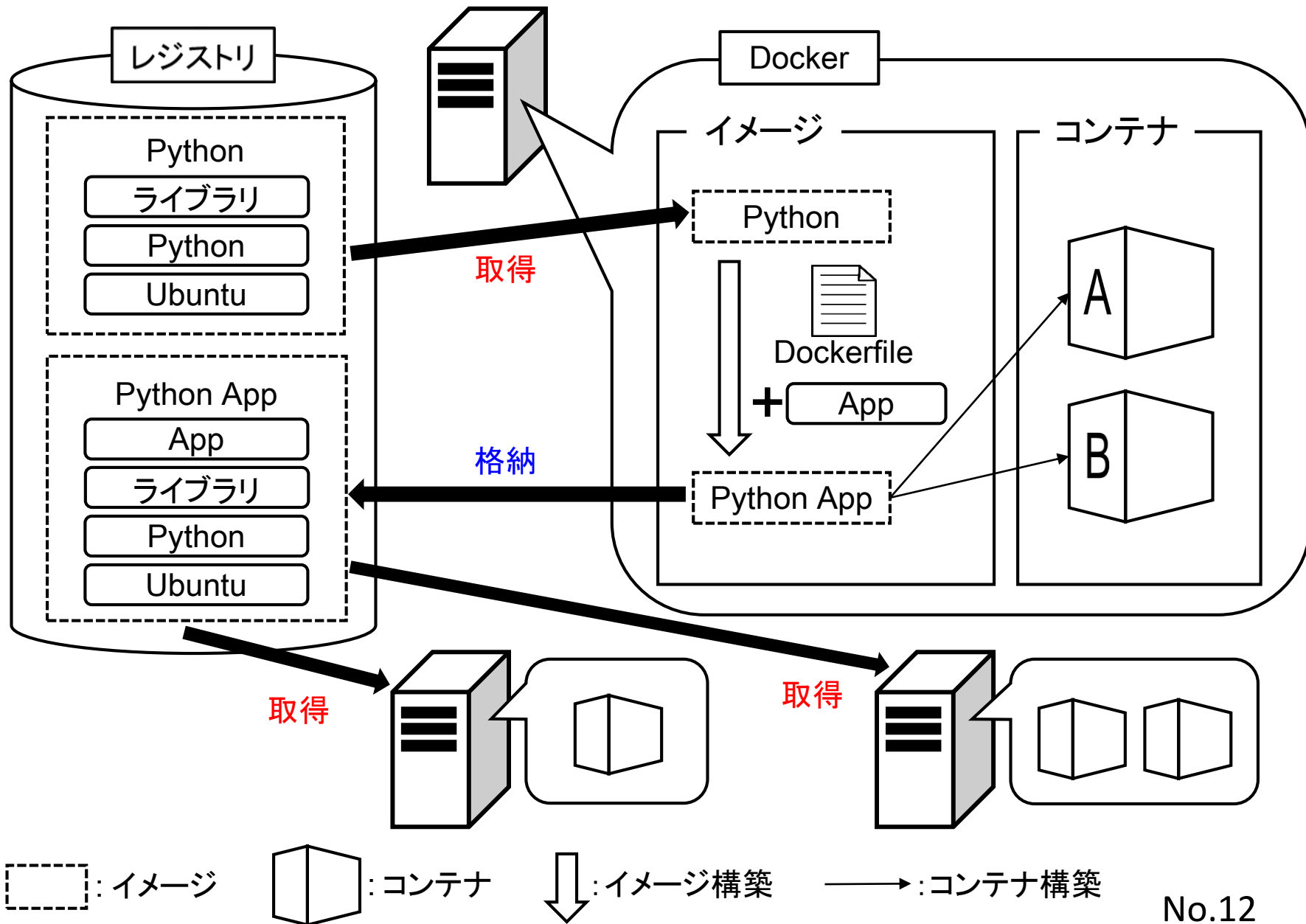
 約35%低下

(3) 同一のブリッジネットワークデバイスへ接続するコンテナ数の増加によるオーバヘッド

 最大値に対して約0.004% (1.5Mbps) ずつ低下

予備スライド

Dockerの利用例



コンテナの実現

＜プロセスをコンテナに隔離する3つの要素＞

(1) リソースの分離

namespacesを利用

右表のリソースを分離

名前空間	分離するリソース
IPC	共有メモリ, セマフォ, メッセージキュー
Mount	ファイルシステムのマウント
Network	ネットワークデバイス, プロトコルスタック
PID	プロセステーブル
User	ユーザID, グループID
UTS	ホスト名, ドメイン名

(2) ファイルシステムの制限

chrootを利用

プロセスの認識するルートディレクトリを変更

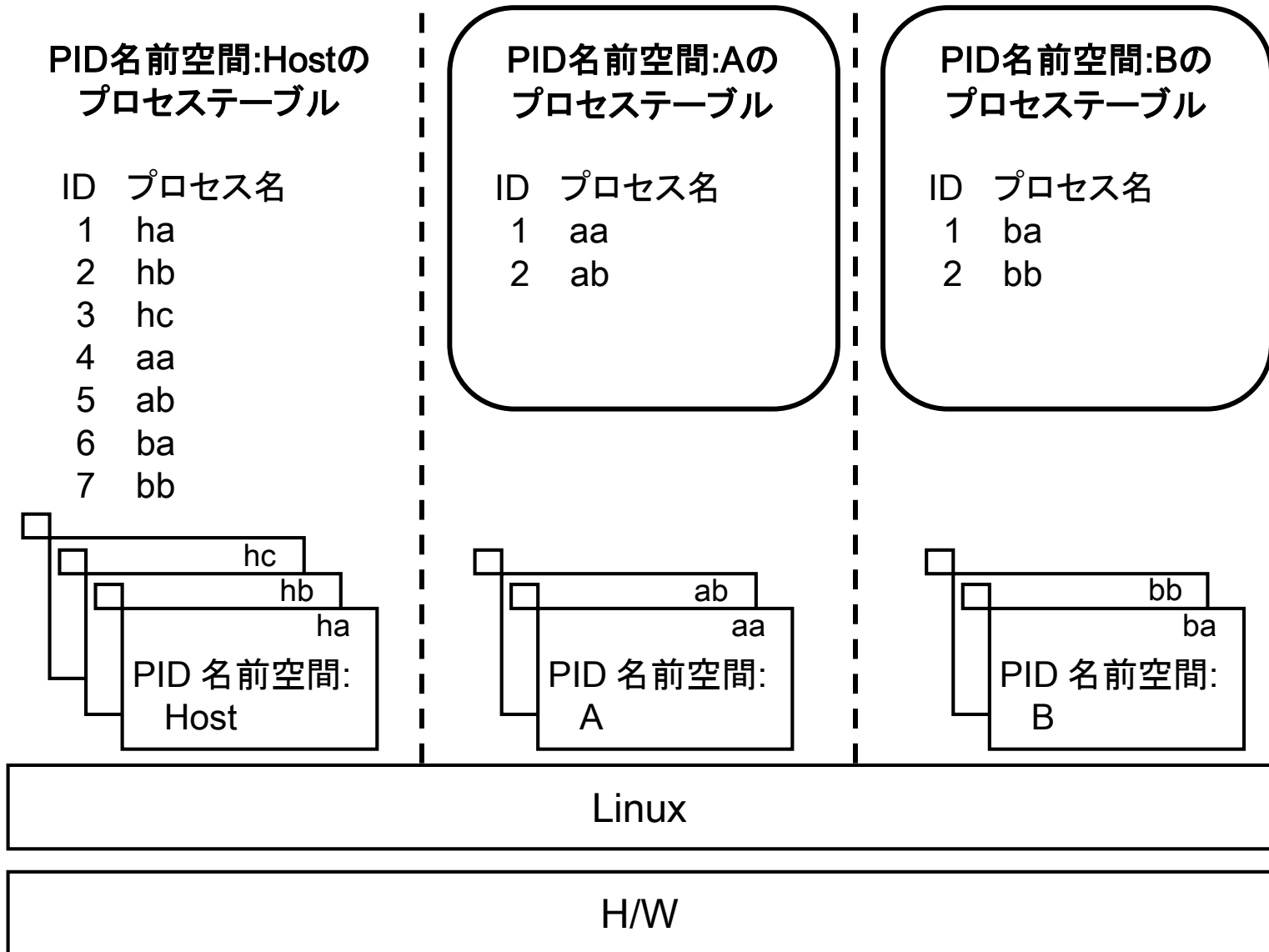
(3) CPUとメモリの使用量制限

cgroupsを利用

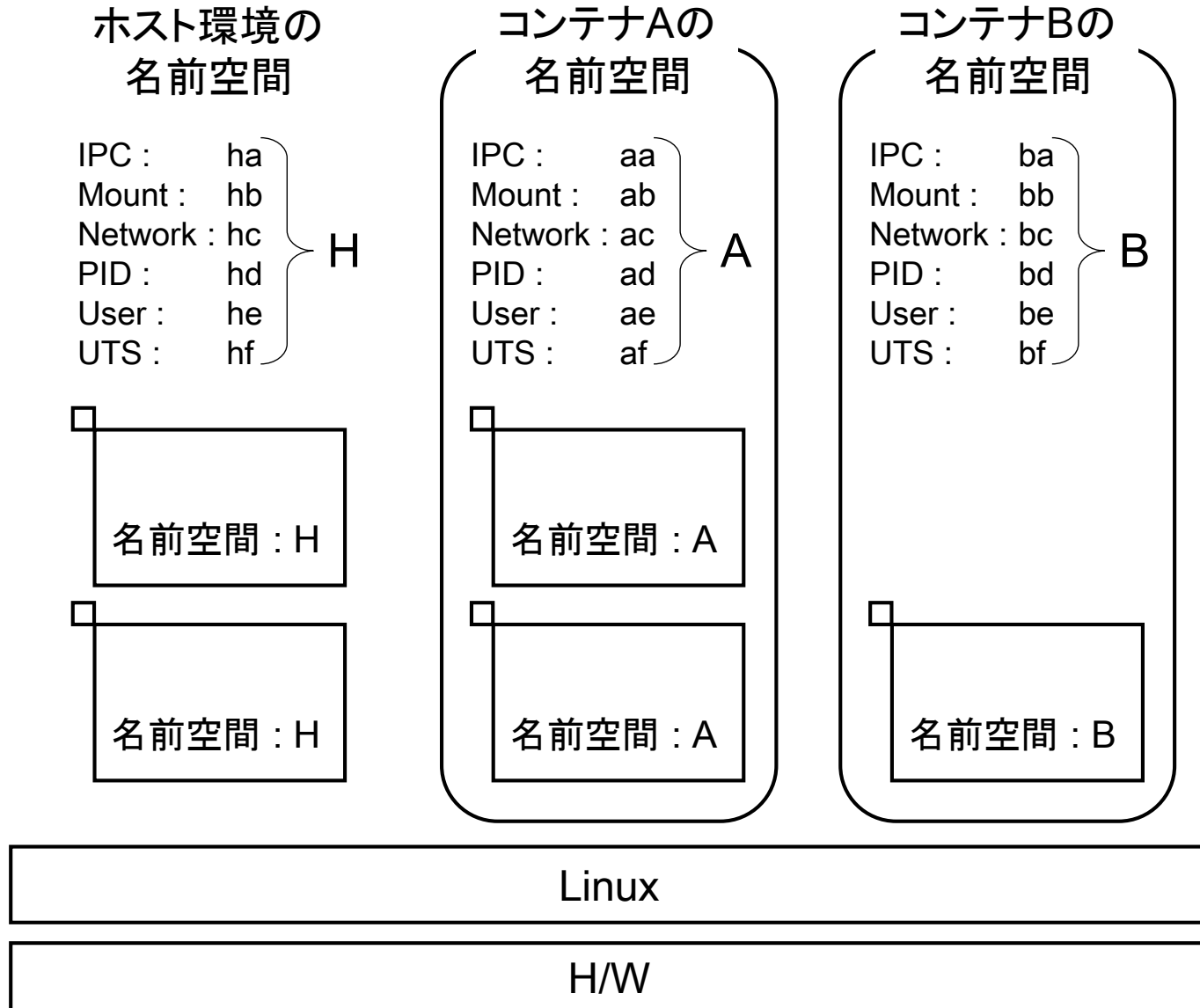
プロセスのグループ化

グループ単位でCPUとメモリの使用量を制限

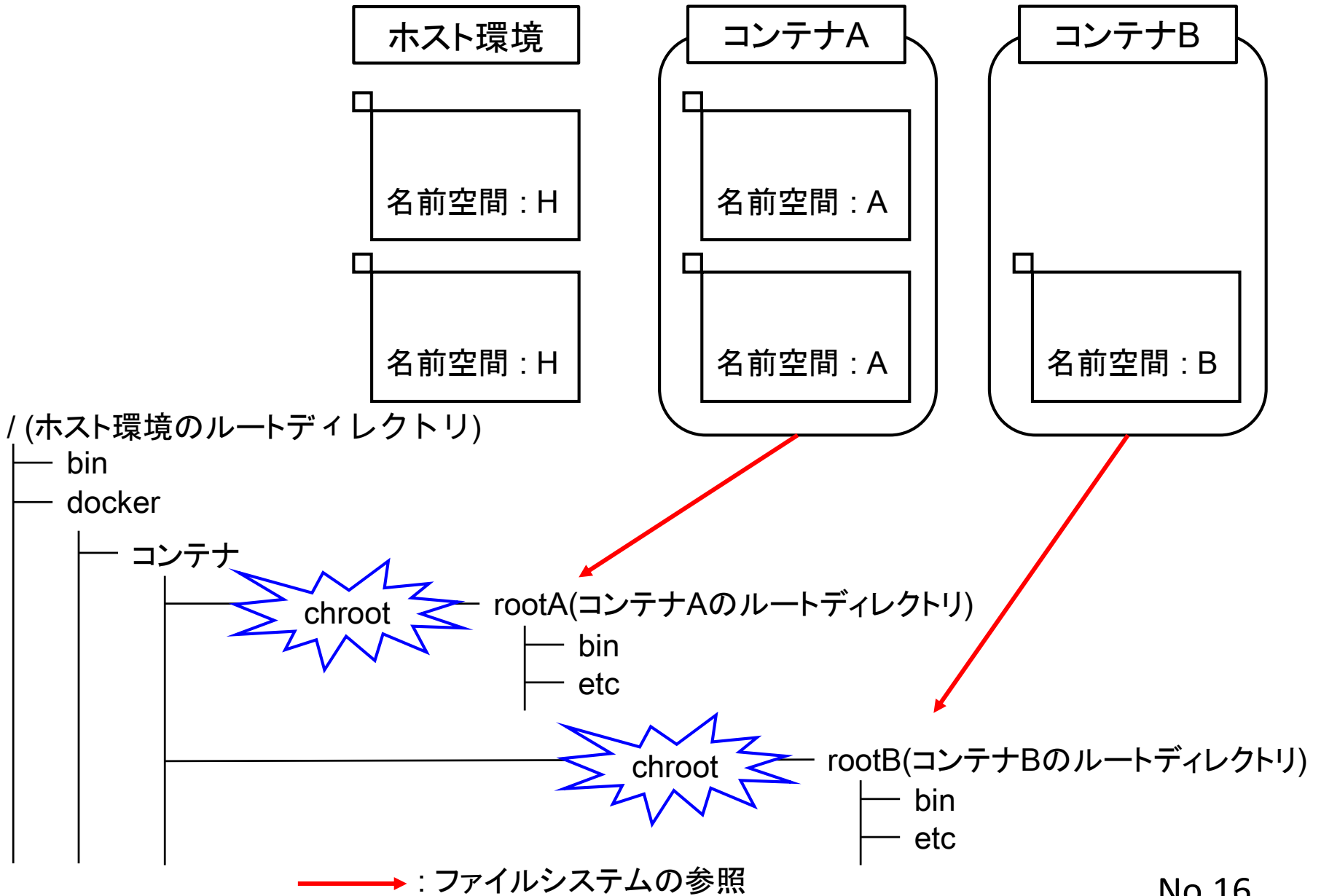
プロセステーブルの分離



リソース分離によるプロセスの隔離



ファイルシステムの制限



ブリッジネットワークの実現

vethデバイスとLinux Bridgeを利用してブリッジネットワークの構築

