Microsoftと
ライブラリOSの最前線

黒米 祐馬
慶應義塾大学

2015.07.28
iijlabセミナー

自己紹介

- 黒米 祐馬(@ntddk)
- 慶應義塾大学 環境情報学部 3年
- セキュリティ・キャンプ2015全国大会講師
- IIJ-IIアルバイト

Windowsをアプリとして動作させる新仮想化技術「ライブラリOS」

仮想化技術は数年前からIT分野で旬な話題
ンシューマですら4コアや6コアがあたり
プロセッサはメニーコアの時代に入ると見
すます重要度を増すことになると見られる

一口に仮想化技術といっても、考え方も実
イヤも実にさまざまだ。ハイパーバイザ
ルでハードウェアのフル仮想化、OS内部
供、サンドボックスやコンパートメント
化、挙げていけばきりがない。

CNET Japan 特集 All About Microsoft

マイクロソフト、クラウド上のアプリを隔離する新技術「Haven」を開発

リオ） 2014/10/07 12:50

する！
か？まとめ記事を読む！

ースに、クラウドの安全性に対する信頼
。

Intrusted Cloud with Haven」と題する
州ブルームフィールドで今週行われてい
tems Design and Implementationで紹
ーザーの「WalkingCat」氏に感謝）。

［速報］マイクロソフトがDockerと提携、次期Windows ServerでDockerを採用と発表。Microsoft AzureではDocker Hubとの統合も

2014年10月16日

米マイクロソフトはDocker社と提携し、次期Windows ServerでDockerをサポート。Microsoft AzureでもDockerをサポートするとともに、Docker Hubとの統合も行うと発表しました。

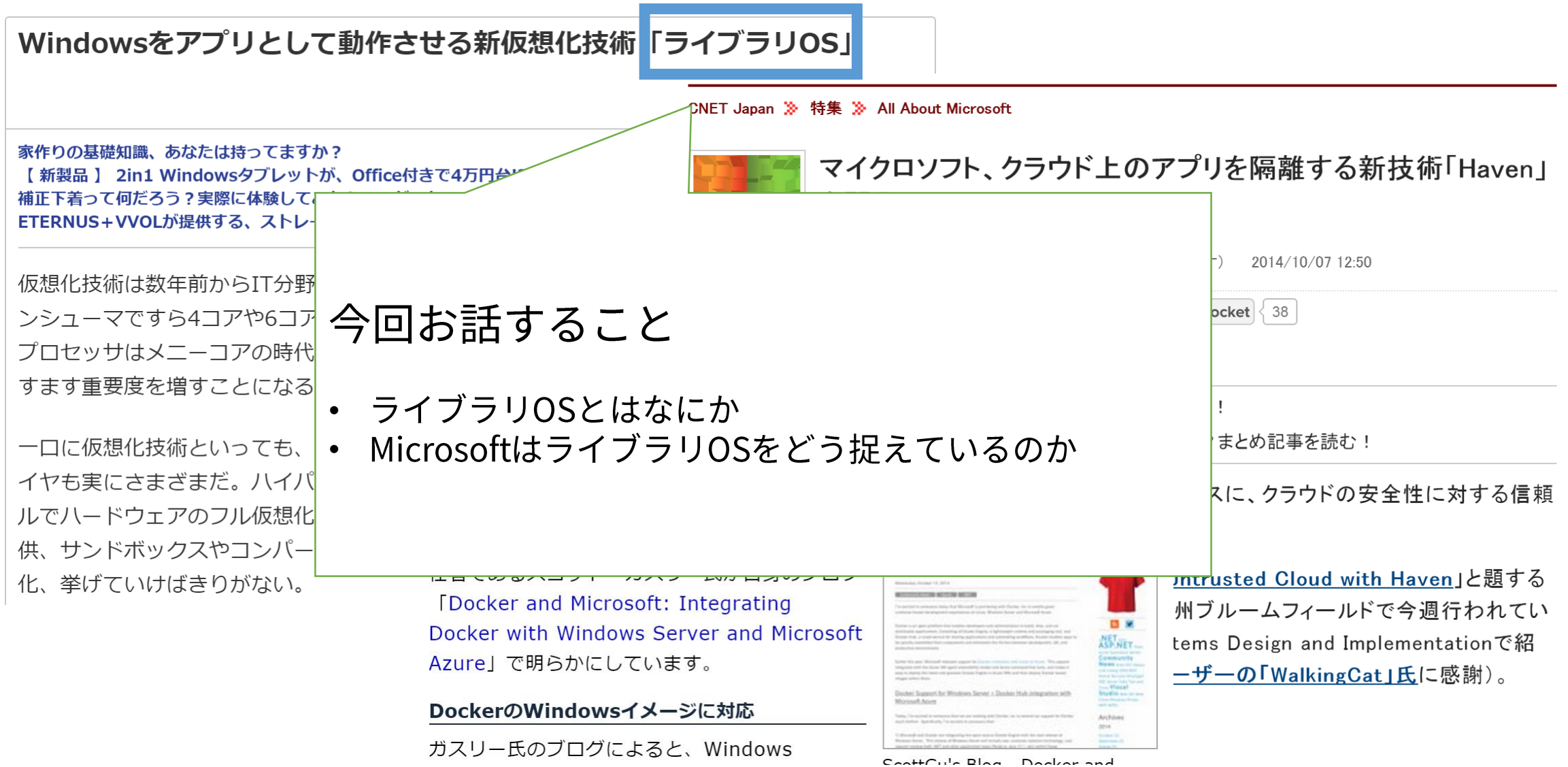
同社のエンタープライズおよびクラウド部門の責任者であるスコット・ガスリー氏が自身のブログ「Docker and Microsoft: Integrating Docker with Windows Server and Microsoft Azure」で明らかにしています。

ScottGu's Blog - Docker and

DockerのWindowsイメージに対応

ガスリー氏のブログによると、Windows

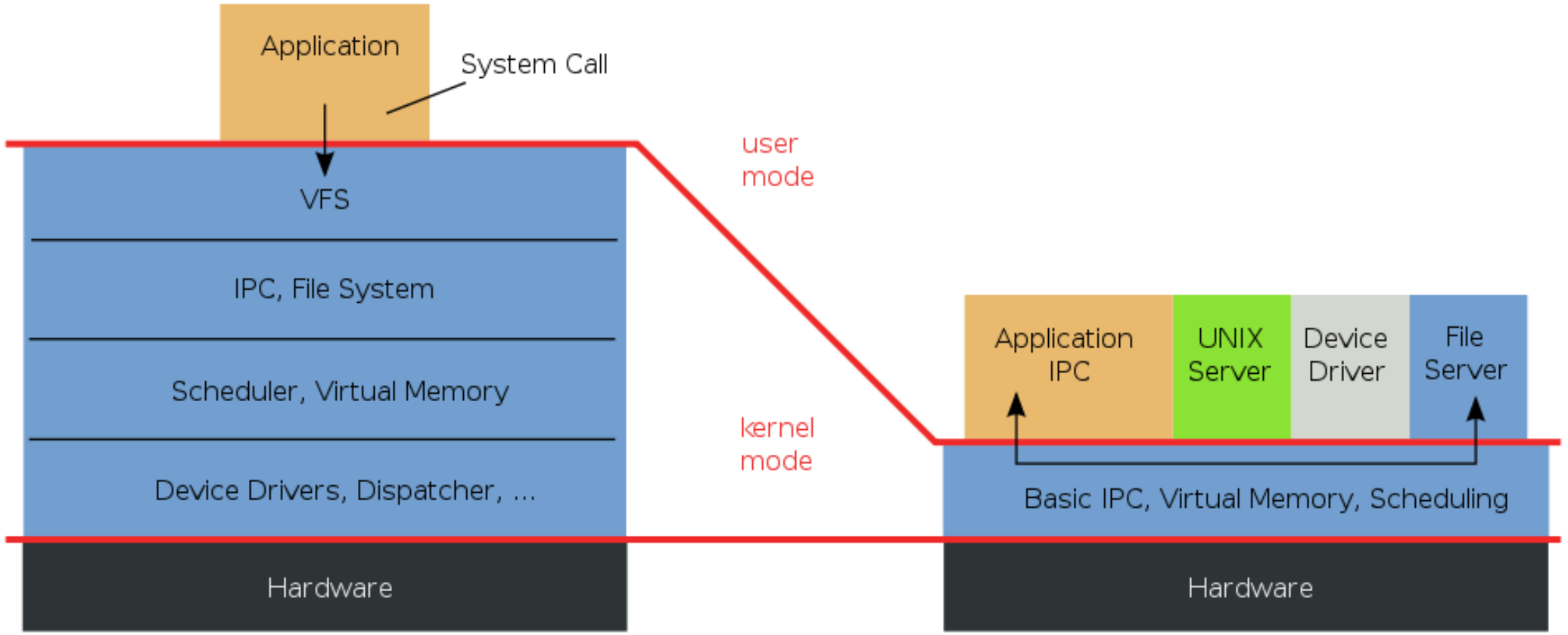
http://news.mynavi.jp/news/2011/10/28/022/
http://japan.cnet.com/sp/allaboutms/35054783/
http://www.publickey1.jp/blog/14/dockerwindows_serverdockermicrosoft_azuredocker_hub.html



http://news.mynavi.jp/news/2011/10/28/022/
http://japan.cnet.com/sp/allaboutms/35054783/
http://www.publickey1.jp/blog/14/dockerwindows_serverdockermicrosoft_azuredocker_hub.html

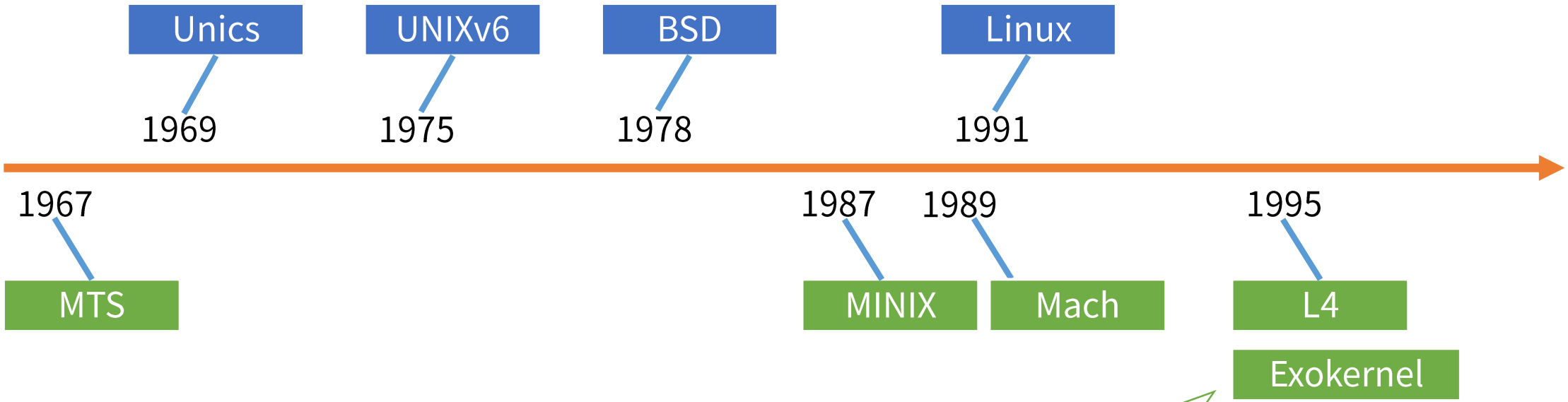
Monolithic Kernel based Operating System

Microkernel based Operating System



http://upload.wikimedia.org/wikipedia/commons/thumb/6/67/OS-structure.svg/1000px-OS-structure.svg.png

カーネルの歴史

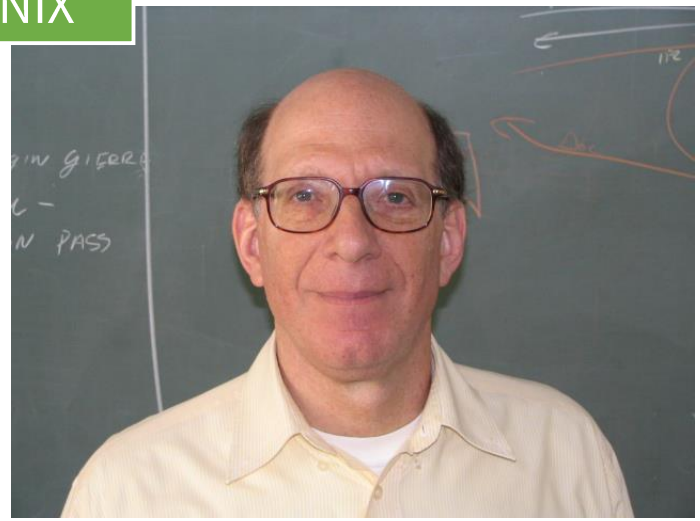

Unics
UNIXv6
BSD
Linux
1969
1975
1978
1991
1967
1987
1989
1995
MTS
MINIX
Mach
L4
Exokernel
ライブラリOSの起源

ニュースグループでの論争（1992年）

Linux

マイクロカーネルはプロセス間通信の
オーバーヘッドが大きすぎる

MINIX



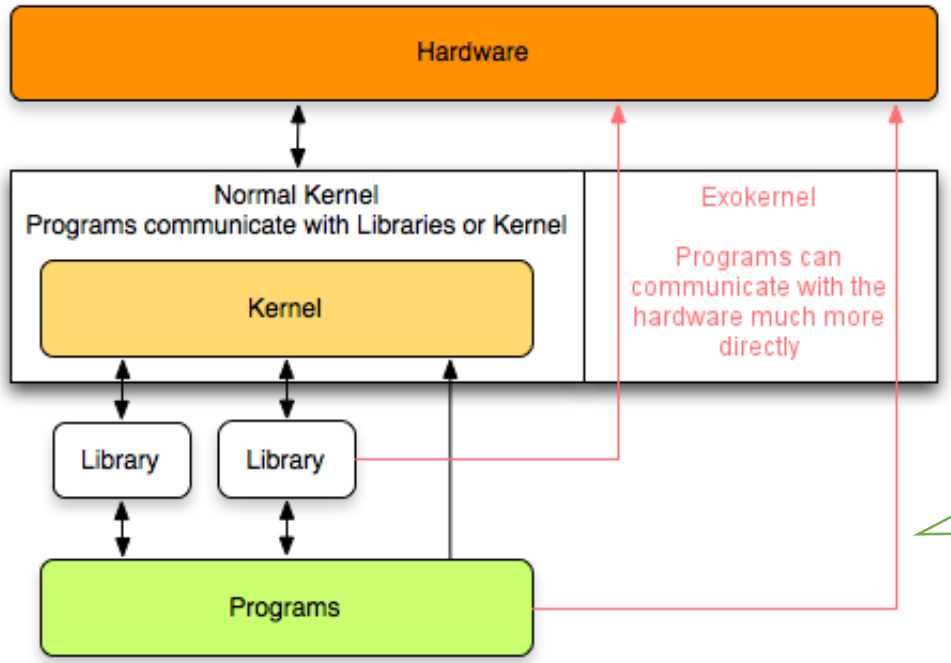
モノリシックカーネルは70年代の発想
Linuxはx86に依存しすぎている

http://i.gzn.jp/img/2012/06/18/linus-nvidia-f-word/linus09_m.jpg
http://upload.wikimedia.org/wikipedia/commons/c/c3/AndrewTanenbaum.JPG

エクソカーネル（1995年）

https://en.wikipedia.org/wiki/File:Exokernel_revised(english).png

ハードウェアの排他制御のみを提供する
軽量な第二世代マイクロカーネル



ライブラリOS

単一のプロセスのみサポート

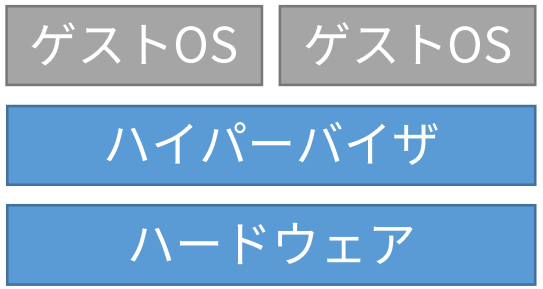
OSの機能はユーザーモードで動作するライブラリとしてカーネルから除去

ハイパーバイザ

計算機の内部で仮想化された計算機を動かす技術

- Type-1 ハイパーバイザ
  - Xen
  - Hyper-V

  など



- Type-2 ハイパーバイザ
  - QEMU
  - KVM

  など

エクソカーネルとの収斂進化

PopekとGoldbergの仮想化要件

仮想化を実現するためには特権命令と
センシティブ命令をトラップしなければならない

- 特権命令
  - 特権モードでなければ実行できない命令
- センシティブ命令
  - 制御センシティブ命令
    - システムを構成する資源の状態に影響を与える命令
  - 動作センシティブ命令
    - システムを構成する資源の状態に動作が依存する命令

PopekとGoldbergの仮想化要件

仮想化を実現するためには特権命令と
センシティブ命令をトラップしなければならない

- ¬特権命令∧センシティブ命令
  - 複数のOSが一つのCPUのレジスタを同時に操作するわけにはいかない
  - mov
  - jmp
  - call
  - ret
  - push
  - pop

など

x86では極めてありきたりな命令

<u>OS間のコンテキストスイッチが必要</u>

Xen 準仮想化（2003年）

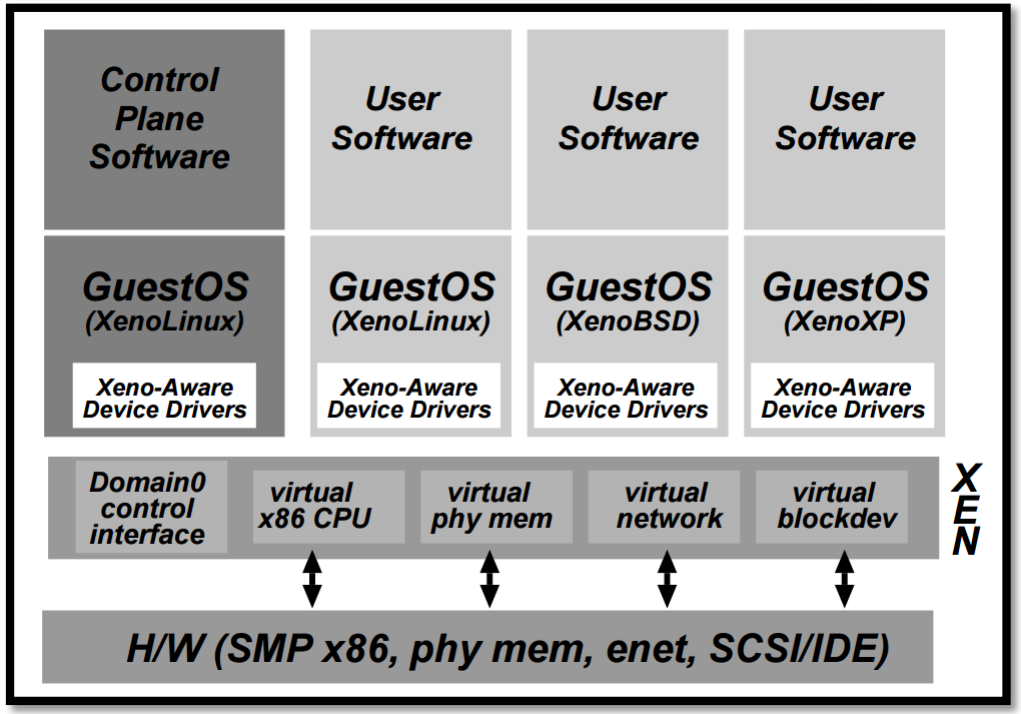
エクソカーネルの考え方を用いて
ハイパーバイザを実現

汎用OS

センシティブ命令の実行をハイパーバイザに通知するための準仮想化ドライバ

※Intel VT登場以前

ハードウェアの排他制御を提供する
ハイパーバイザ



汎用OSと準仮想化ドライバによるライブラリOSライクな仮想化

エクソカーネル再考

**エクソカーネルではアプリケーション
開発者がドライバを書かなければならない**

- メリット
  - 軽量
  - ハードウェアの特性を活かしたアプリケーションの開発が可能
- デメリット
  - ドライバ開発のコスト
  - ヘテロジニアスマルチコアアーキテクチャの時代にそぐわない

ハイパーバイザの上でライブラリOSを動かせばよいのではないか？

ハイパーバイザ再考

ハイパーバイザと汎用OSで
保護機能が重複している

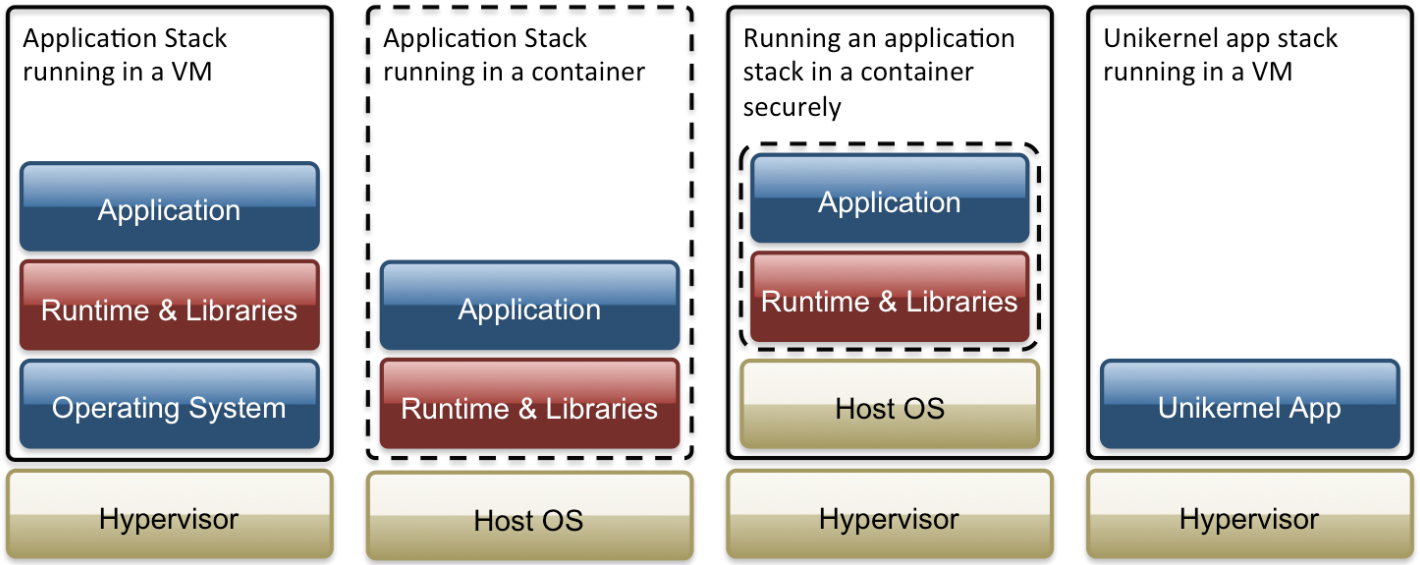
（モノリシックな）汎用OSはハイパーバイザによって構成されたクラウド環境を想定していない

ハイパーバイザの上でライブラリOSを動かせばよいのではないか？

ユニカーネル（2013年～）

ハイパーバイザの上で動作する
クラウド環境を前提としたライブラリOS

https://www.linux.com/images/stories/41373/unikernel-illustration.png



OSv, Click OS, Mirage OS, Erlang on Xen, Rump Kernels, NUSEなど

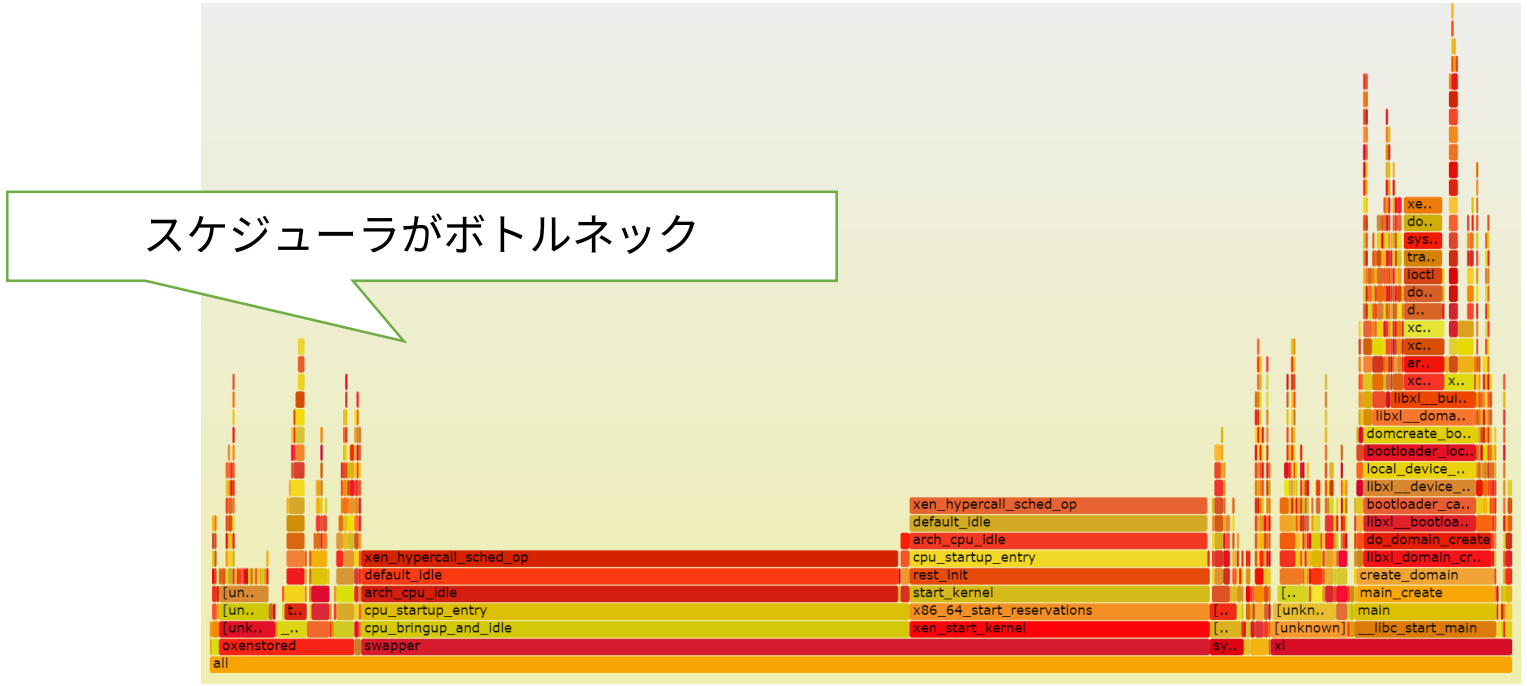
ライブラリOSからドライバを除いたもの

エクソカーネルにドライバを加えたもの

ユニカーネルはハイパーバイザの上で単一のプロセスのみをサポートする

ユニカーネル（2013年～）

ハイパーバイザの上で動作する
クラウド環境を前提としたライブラリOS



ライブラリOSを前提としたハイパーバイザの必要性？

Drawbridge（2011年）

コンテナ技術として
ライブラリOSを捉え直す

- ASPLOS’11
  - Rethinking and Protecting Operating Systemsセッション
- アプリケーションに変更を加えずサンドボックス化したい

巨大なWindowsサブシステムをいかに切り詰めるか？

Drawbridgeのアーキテクチャ

コンテナ技術として
ライブラリOSを捉え直す

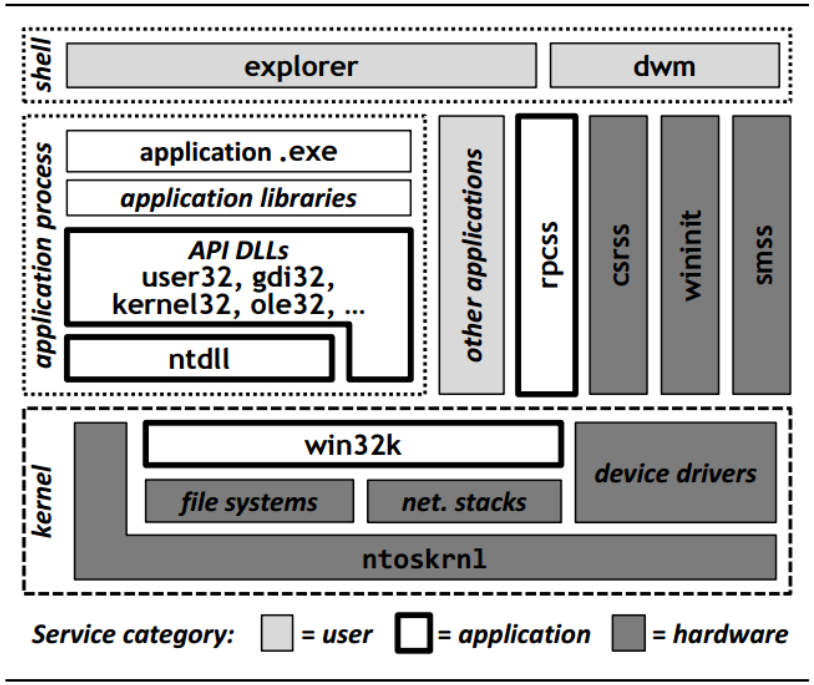


Figure 1. Windows 7 OS Architecture.

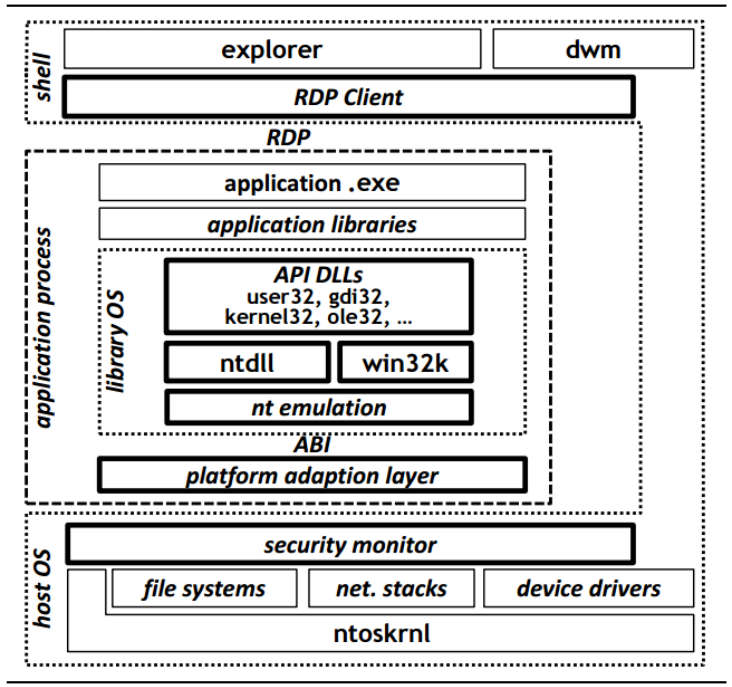


Figure 2. Drawbridge Architecture.

Windowsカーネルの一部をユーザーモードのライブラリとして再実装

ライブラリOS

コンテナ技術として
ライブラリOSを捉え直す

DLLとして実装されたWindowsカーネル

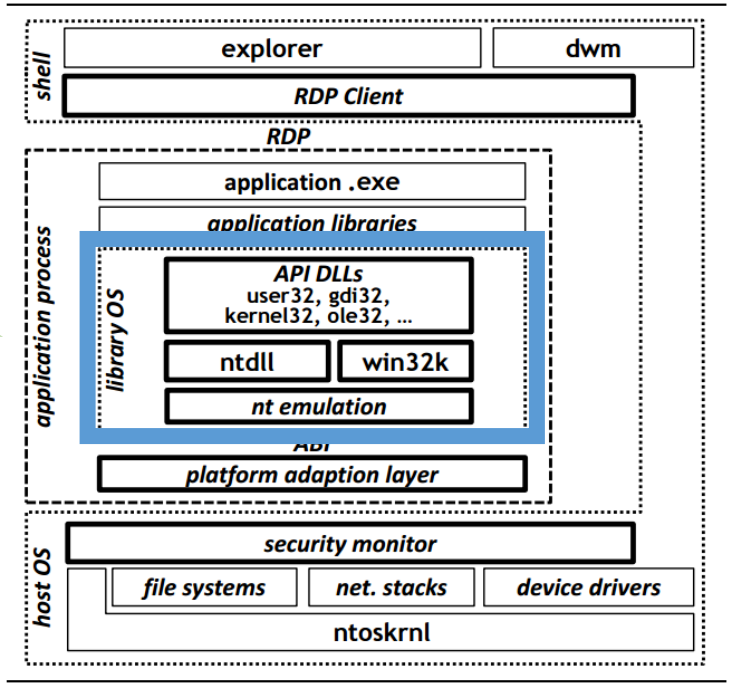


Figure 2. Drawbridge Architecture.

Windowsカーネルの一部をユーザーモードのライブラリとして再実装

Picoprocess

コンテナ技術として
ライブラリOSを捉え直す

プロセスベースのコンテナ

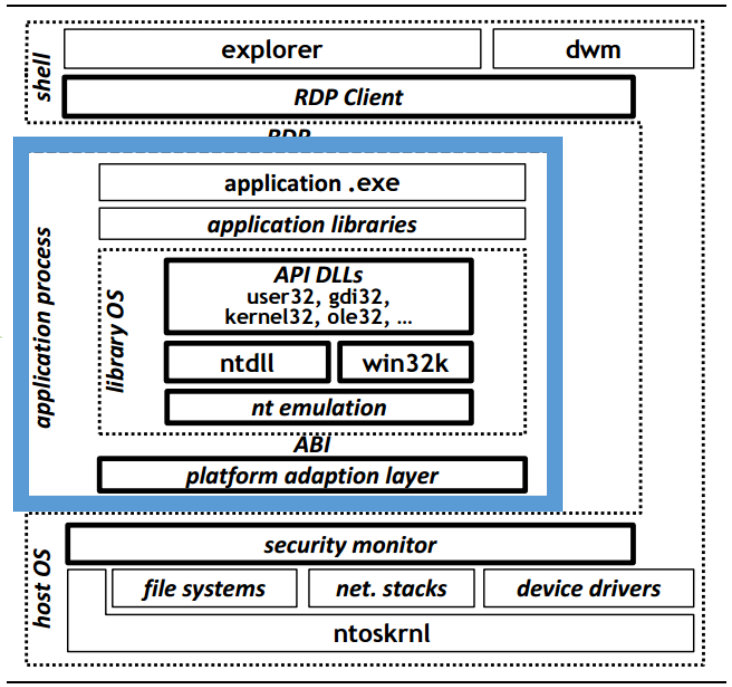


Figure 2. Drawbridge Architecture.

Windowsカーネルの一部をユーザーモードのライブラリとして再実装

Security monitor

コンテナ技術として
ライブラリOSを捉え直す

ABI
- スレッディング
- 仮想メモリ
- ファイルシステム
- ネットワーキング

カーネルモジュール

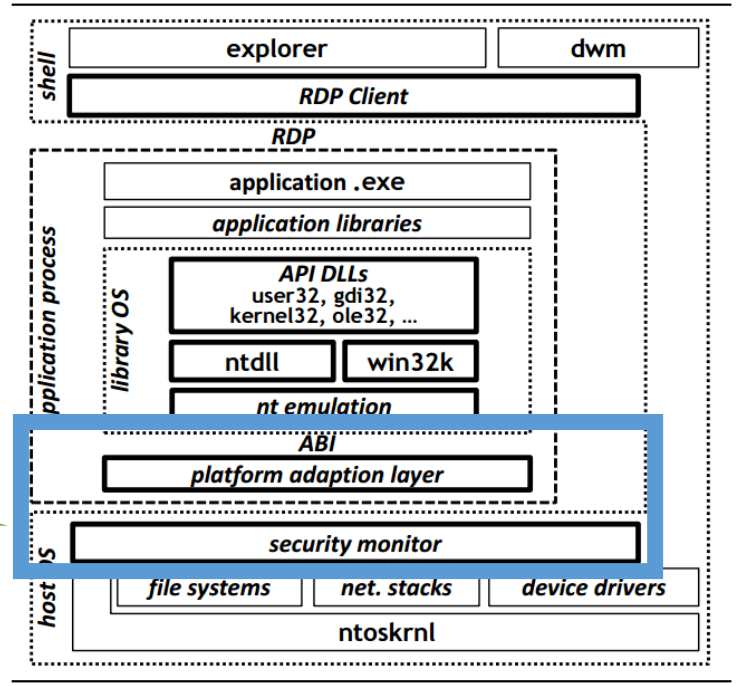


Figure 2. Drawbridge Architecture.

ABI boundaryがsecurity monitorよりも上にあるなら，
Windows以外のホストpicoprocessが動くようになるかも？

Windowsカーネルの一部をユーザーモードのライブラリとして再実装

Memory Management Primitives
- DkVirtualMemoryAlloc
- DkVirtualMemoryFree
- DkVirtualMemoryProtect

Thread Primitives
- DkThreadCreate
- DkThreadDelayExecution
- DkThreadYieldExecution
- DkThreadExit
- DkThreadGetParameter
- DkThreadRaiseException
- DkNotificationEventCreate
- DkSynchronizationEventCreate
- DkSemaphoreCreate
- DkSemaphoreRelease
- DkSemaphorePeek
- DkEventSet
- DkEventClear
- DkEventPeek
- DkObjectsWaitAny
- DkAbortEventRegister

Child Process Primitives
- DkProcessCreate
- DkProcessGetExitCode
- DkProcessExit

I/O Stream Primitives
- DkStreamOpen
- DkStreamRead
- DkStreamWrite
- DkStreamMap
- DkStreamMapPeBinary
- DkStreamUnmap
- DkStreamSetLength
- DkStreamFlush
- DkStreamDelete
- DkStreamGetEvent
- DkStreamRename
- DkStreamEnumerateChildren
- DkStreamAttributesQuery
- DkStreamAttributesQueryByHandle

Other Primitives
- DkSystemTimeQuery
- DkRandomBitsRead
- DkInstructionCacheFlush
- DkObjectReference
- DkObjectClose
- DkInputEventRead
- DkFrameBufferExport
- DkFrameBufferNotifyUpdate
- DkDebugStringPrint

Upcalls
- LibOsInitialize
- LibOsThreadStart
- LibOsExceptionDispatch

Files/Storage
- file:

Console Redirection
- null:
- stderr:
- stdin:
- stdout:

Named Pipes
- pipe.client:
- pipe.server:

TCP/IP Stack
- dns:
- tcp.client:
- tcp.server:
- tcp:

HTTP.SYS
- http.application:
- http.server:

Drawbridgeの性能

- ネイティブ環境と遜色のないフットプリント

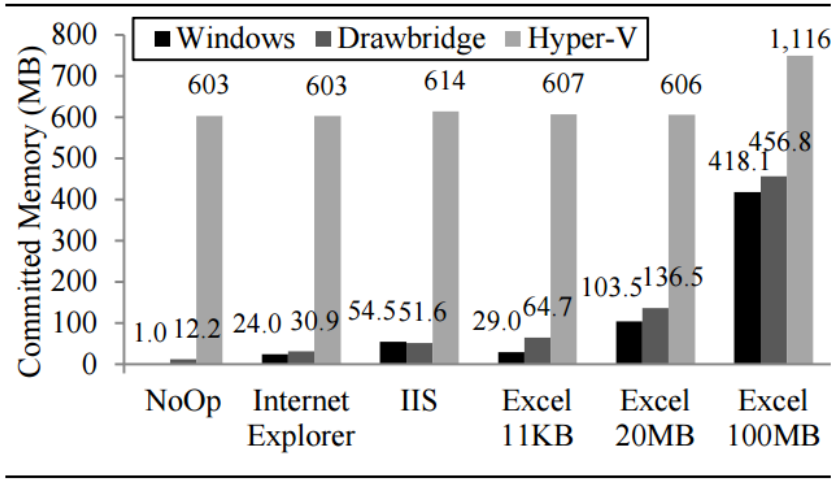


Figure 4. Memory per application (including memory used by library OS or guest OS).

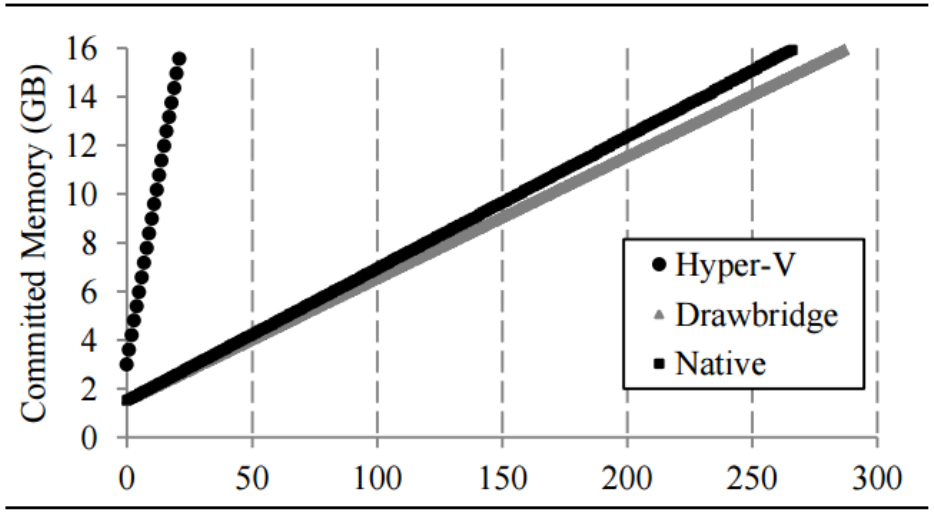
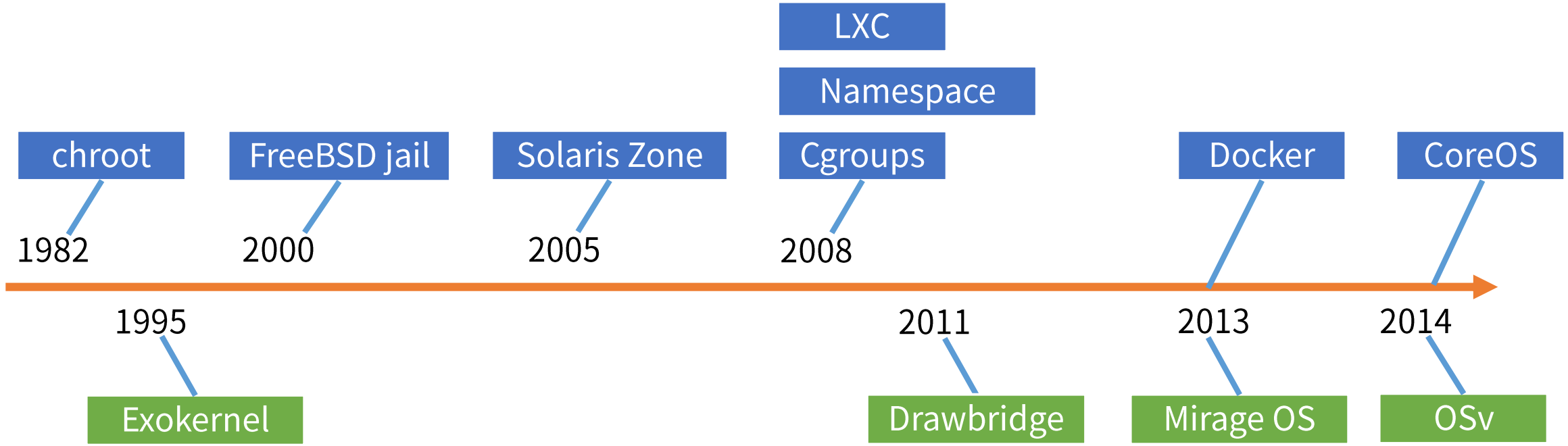


Figure 8. Memory committed for increasing copies of IIS.

- 当時はWindowsのメインラインにマージする予定はなかった
  - “At the time of writing, Microsoft has no plans to productize any of the concepts prototyped in Drawbridge. ”

コンテナ技術とライブラリOSの歴史

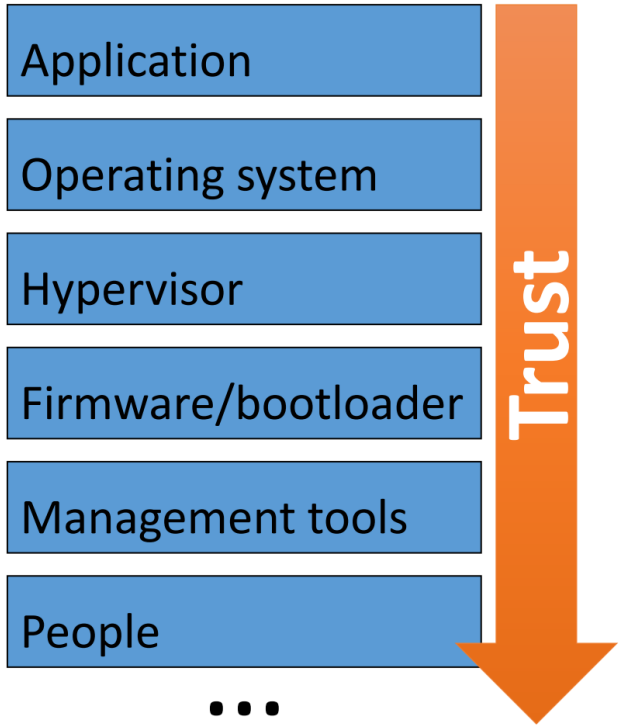


DrawbridgeはMirage OSやOSvといったフルスクラッチのライブラリOSよりもLinuxを切り詰めたCoreOSに近い

Haven（2014年）

Drawbridgeの応用

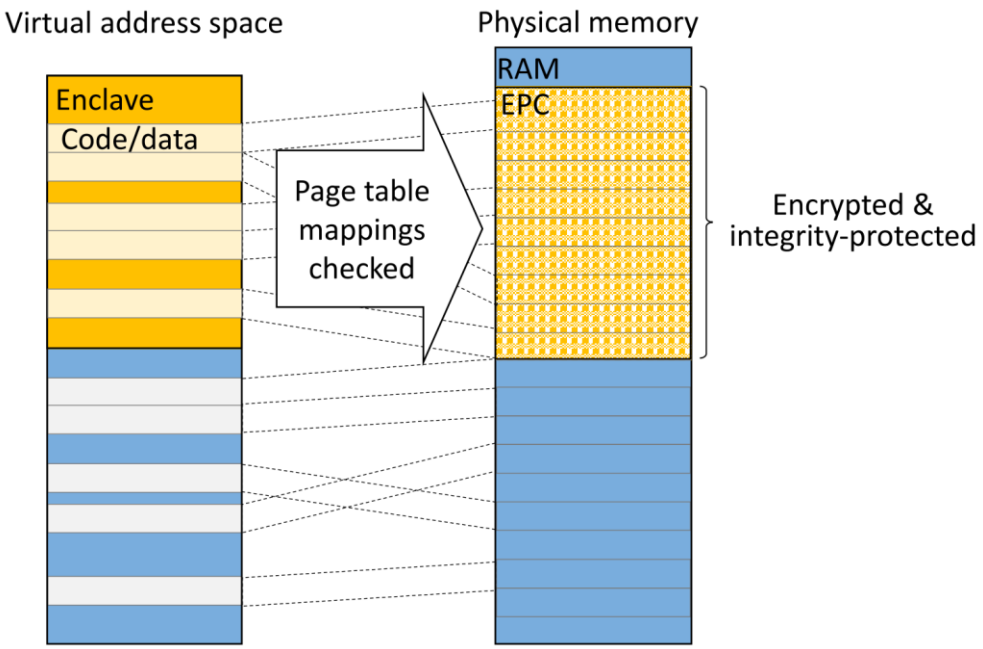
- OSDI’14 Best Paper
- 階層型のトラストモデルは不完全
  - アプリケーションはハイパーバイザを信頼しなければならない

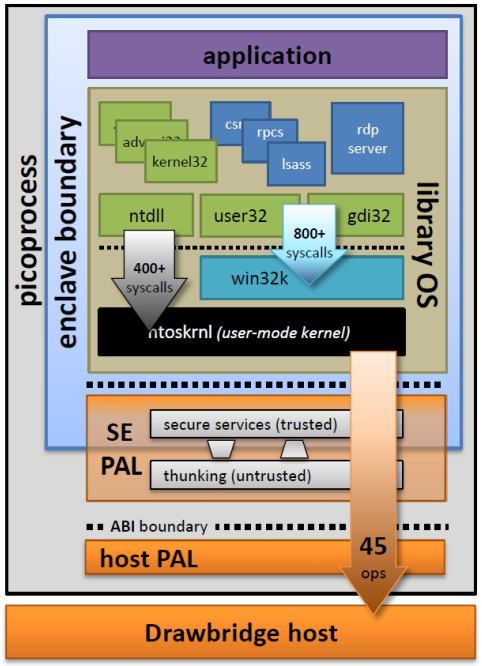


信頼できないクラウド上でアプリケーションを保護する

Haven（2014年）

Intel SGXによるpicoprocessの保護

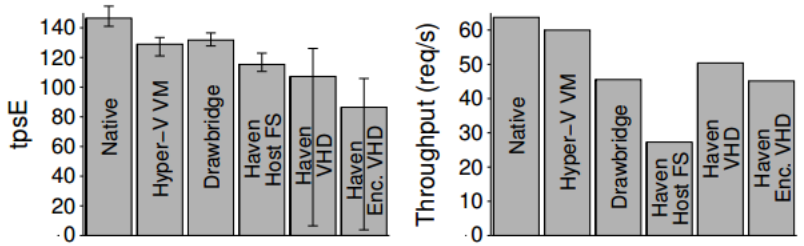




アクセス制御のためのページングの拡張

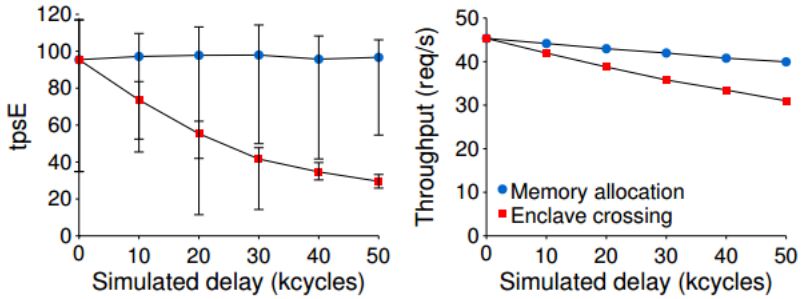
Havenの性能

- 捌けるトランザクション数の減少
- VMと比べて35%(Apache)～65%(SQL Server)の性能低下



(a) SQL Server, TPC-E (b) MediaWiki on Apache

Figure 4: Performance breakdown



(a) SQL Server, TPC-E (b) MediaWiki on Apache

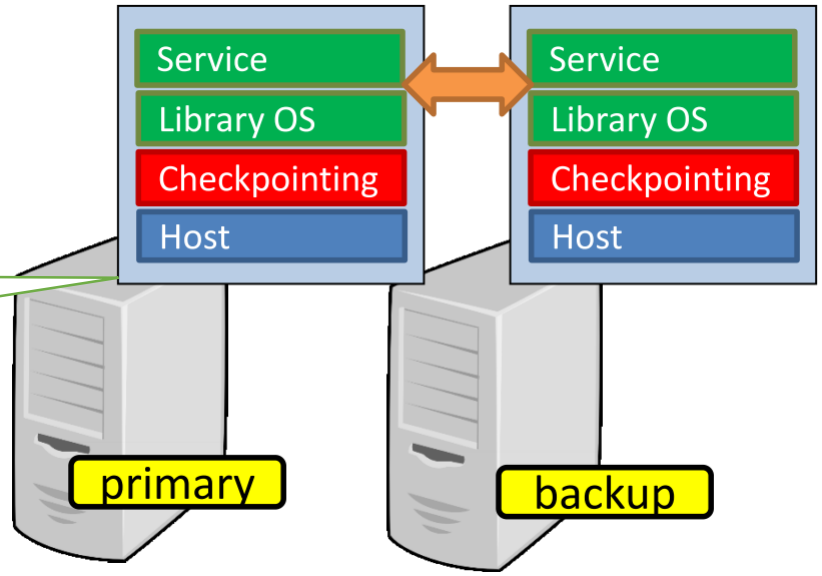
Figure 5: Sensitivity to SGX instruction overhead

Tardigrade（2015年）

Drawbridgeの応用

- NSDI’15

- LVMによるチェックポイント
- 例外ハンドラによるスレッドコンテキストの保存
- セマフォ



ライブラリOSの非同期レプリケーションによるFault-tolerantなシステム

Tardigradeの性能

- Bascule（Drawbridgeの拡張）上に実装

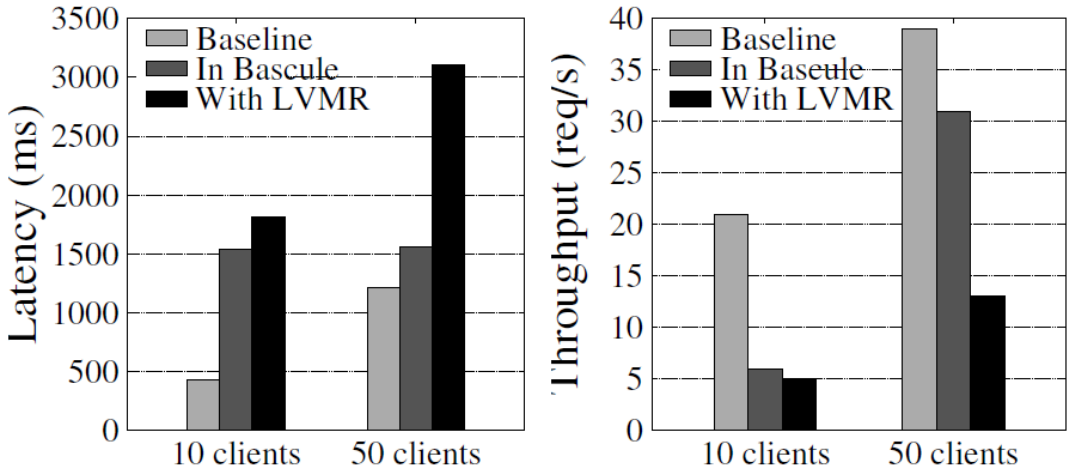


Figure 11: Performance of MediaWiki in Apache with different numbers of client threads

runC

Dockerと互換性のあるコンテナランタイム

- “Native support of Windows 10 containers is being contributed directly by Microsoft engineers”
  - Drawbridgeのことではないか？

```
// TODO Windows: This can be factored out in the future as Cgroups are not
// supported on the Windows platform.
```

DockerとDrawbridge

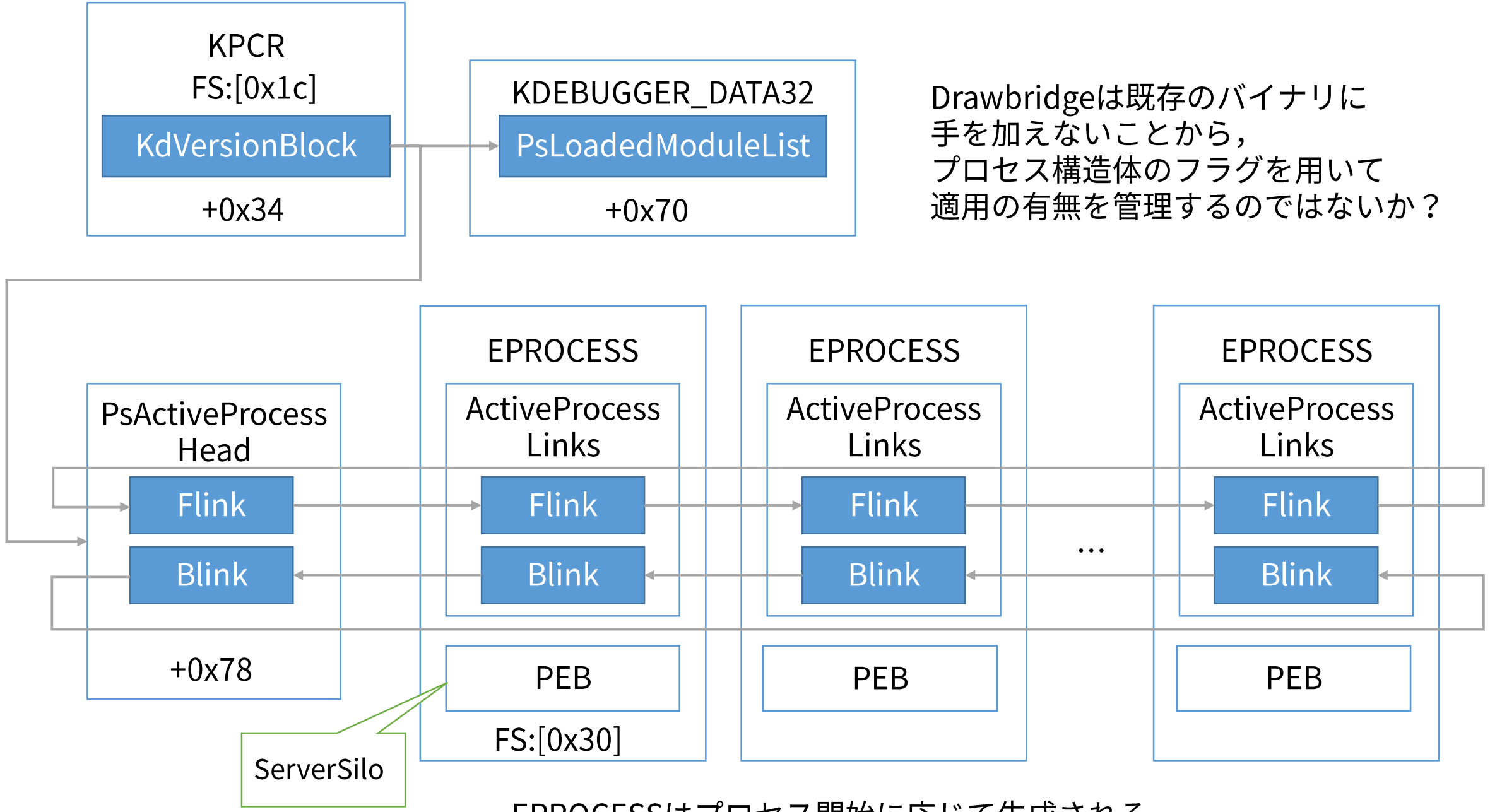
- AzureチームのMadhan Ramakrishnan曰く，

Regarding Drawbridge, as you pointed out it is an internal research project that we have been innovating on, and that has helped us gain valuable experience with containers.

Much of what we announced today was born from the experience that we had with Drawbridge and we are excited to bring container technologies to Windows Server and the Docker ecosystem along with Linux.

We think the combination of our own hypervisor for container virtualization and Docker containers for creating a unified deployment and management experience is a compelling scenario for our customers.

https://news.ycombinator.com/item?id=8461111


KPCR
FS:[0x1c]
KdVersionBlock
+0x34
KDEBUGGER_DATA32
PsLoadedModuleList
+0x70
Drawbridgeは既存のバイナリに
手を加えないことから，
プロセス構造体のフラグを用いて
適用の有無を管理するのではないか？
PsActiveProcess
Head
Flink
Blink
+0x78
EPROCESS
ActiveProcess
Links
Flink
Blink
PEB
FS:[0x30]
EPROCESS
ActiveProcess
Links
Flink
Blink
PEB
…
EPROCESS
ActiveProcess
Links
Flink
Blink
PEB
ServerSilo
EPROCESSはプロセス開始に応じて生成される

Picoprocess(Build 10159)

- Windows 8.1

```
_ETHREAD+0x770 PicoContext : Ptr64 Void
_EPROCESS+0x6a8 PicoContext : Ptr64 Void
```

- Windows 10 Build 10159

```
_EPROCESS + 0x6f0 PicoContext: Ptr64 Void
_ETHREAD + 0x788 PicoContext : Ptr64 Void
```

- ただし未だにPspPicoRegistrationDisabledとなっている
  - サーバーのみ対応？

Picoprocess(Build 10159)

C:¥Program Files (x86)¥Windows Kits¥10¥Include¥10.0.10075.0¥um¥minwin¥ntosp.h

```
typedef struct _PS_PICO_THREAD_ATTRIBUTES {
    HANDLE Process;
    ULONG_PTR UserStack;
    ULONG_PTR StartRoutine;
    ULONG_PTR StartParameter1;
    ULONG_PTR StartParameter2;
…
    ULONG UserFsBase;
    ULONG UserGsBase;
…
    USHORT UserFsSeg;
    USHORT UserGsSeg;

    ULONG_PTR Eax;
…
    PVOID Context;

} PS_PICO_THREAD_ATTRIBUTES, *PPS_PICO_THREAD_ATTRIBUTES;
```

- レジスタの退避
- FS/GSレジスタにはKPCR構造体が入る

Picoprocess(Build 10159)

C:¥Program Files (x86)¥Windows Kits¥10¥Include¥10.0.10075.0¥um¥minwin¥ntosp.h

```
#if (NTDDI_VERSION >= NTDDI_THRESHOLD)
_IRQL_requires_max_(PASSIVE_LEVEL)
NTKERNELAPI
NTSTATUS
PsRegisterPicoProvider (
    _In_ PPS_PICO_PROVIDER_ROUTINES ProviderRoutines,
    _Out_ PPS_PICO_ROUTINES PicoRoutines
    );
#endif

typedef struct _PS_PICO_PROVIDER_ROUTINES {
    PPS_PICO_PROVIDER_SYSTEM_CALL_DISPATCH DispatchSystemCall;
    PPS_PICO_PROVIDER_THREAD_EXIT ExitThread;
    PPS_PICO_PROVIDER_PROCESS_EXIT ExitProcess;
    PPS_PICO_PROVIDER_DISPATCH_EXCEPTION DispatchException;
} PS_PICO_PROVIDER_ROUTINES, *PPS_PICO_PROVIDER_ROUTINES;
```

picoprocess化したいプロセスの登録？

Picoprocess(Build 10159)

C:¥Program Files (x86)¥Windows Kits¥10¥Include¥10.0.10075.0¥um¥minwin¥ntosp.h

```
typedef struct _PS_PICO_ROUTINES {
    PPS_PICO_CREATE_PROCESS CreateProcess;
    PPS_PICO_CREATE_THREAD CreateThread;
    PPS_PICO_GET_PROCESS_CONTEXT GetProcessContext;
    PPS_PICO_GET_THREAD_CONTEXT GetThreadContext;
    PPS_GET_CONTEXT_THREAD_INTERNAL GetContextThreadInternal;
    PPS_SET_CONTEXT_THREAD_INTERNAL SetContextThreadInternal;
    PPS_TERMINATE_THREAD TerminateThread;
    PPS_RESUME_THREAD ResumeThread;
    PPS_PICO_SET_THREAD_DESCRIPTOR_BASE SetThreadDescriptorBase;
    PPS_SUSPEND_THREAD SuspendThread;
} PS_PICO_ROUTINES, *PPS_PICO_ROUTINES;
```

管理対象のプロセス，スレッド

Picoprocess(Build 10159)

C:¥Program Files (x86)¥Windows Kits¥10¥Include¥10.0.10075.0¥um¥minwin¥ntosp.h

```
typedef
NTSTATUS
PS_PICO_CREATE_PROCESS (
    _In_ PPS_PICO_PROCESS_ATTRIBUTES ProcessAttributes,
    _Outptr_ PHANDLE ProcessHandle
    );

typedef PS_PICO_CREATE_PROCESS *PPS_PICO_CREATE_PROCESS;

typedef
VOID
PS_PICO_PROVIDER_SYSTEM_CALL_DISPATCH (
    _In_ PPS_PICO_SYSTEM_CALL_INFORMATION SystemCall
    );

typedef PS_PICO_PROVIDER_SYSTEM_CALL_DISPATCH *PPS_PICO_PROVIDER_SYSTEM_CALL_DISPATCH;
```

Picoprocess(Build 10159)

- PspCreatePicoProcess
- PspCreatePicoThread

Server Silo Functions (Build 10159)

C:¥Program Files (x86)¥Windows Kits¥10¥Include¥10.0.10075.0¥um¥minwin¥ntosp.h

```
typedef struct _CONTAINER_ID_INFO {
    GUID ContainerId;
    ULONG Flags;
} CONTAINER_ID_INFO, *PCONTAINER_ID_INFO;

typedef enum _CONTAINER_TYPE {
    ContainerTypeCpu,
    ContainerTypeDiskIo,
    ContainerTypeNetIo,
//    ContainerTypeWorkingSet,
    ContainerTypeHeap,
    ContainerTypeImmediate,
    ContainerTypeMaximumList
} CONTAINER_TYPE, *PCONTAINER_TYPE;
```

Server Silo Functions (Build 10159)

C:¥Program Files (x86)¥Windows Kits¥10¥Include¥10.0.10075.0¥um¥minwin¥ntosp.h

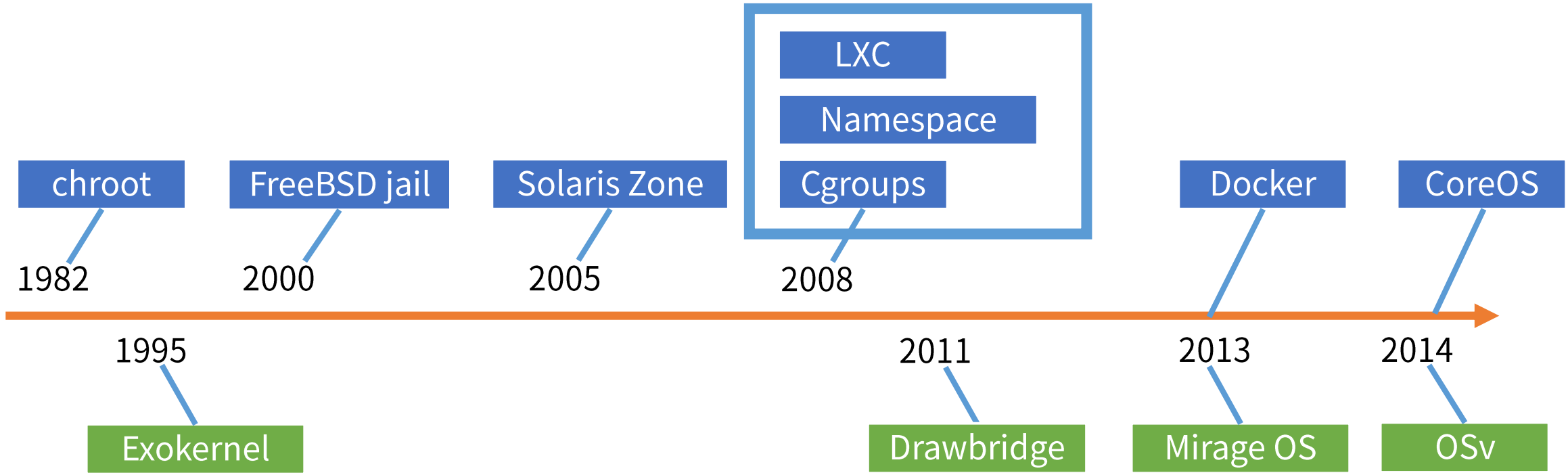
```
#if (NTDDI_VERSION >= NTDDI_WIN10)

_IRQL_requires_max_(DISPATCH_LEVEL)
NTKERNELAPI
NTSTATUS
PsGetEffectiveContainerId(
    _In_ CONTAINER_TYPE ContainerType,
    _In_ PETHREAD Thread,
    _Out_ PCONTAINER_ID_INFO ContainerIdInfo
    );

#endif
```

Server Silo Functions (Build 10159)

- NtQueryInformationSiloObject
- NtSetInformationSiloObject
- SmpStartServerSilo
- CreatePrivateNameSpace（Windows 8.1より）

Siloに指定したプロセスはOpenなどの対象外となる

Server Silo Functions(Build 10159)



SiloはWindowsにおける名前空間分離？

PicoprocessとServer Silo Functions

- 10.0.10075.0まではntosp.hに定義が存在していたが10.10.0.10158.0では無くなっている

参考文献

- D. R. Engler, M. F. Kaashoek and J. O'Toole, Jr., "**Exokernel: An Operating System Architecture for Application-Level Resource Management**," SOSP'95 Proceedings of the 15th ACM symposium on Operating systems principles, pp. 251-266, 1995.
- Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt and Andrew Warfield, "**Xen and the Art of Virtualization**," SOSP'03 Proceedings of the 19th ACM symposium on Operating systems principles, pp. 164-177, 2003.
- Donald E. Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky and Galen C. Hunt, "**Rethinking the Library OS from the Top Down**," ASPLOS'11 Proceedings of the 16th international conference on Architectural support for programming languages and operating systems, pp. 291-304, 2011.
- Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand and Jon Crowcroft, "**Unikernels: Library Operating Systems for the Cloud**," ASPLOS'13 Proceedings of the 18th international conference on Architectural support for programming languages and operating systems, pp. 461-472, 2013.

参考文献

- Andrew Baumann, Dongyoon Lee, Pedro Fonseca, Lisa Glendenning, Jacob R. Lorch, Barry Bond, Reuben Olinsky and Galen C. Hunt, "**Composing OS Extensions Safely and Efficiently with Bascule**," EuroSys'13 Proceedings of the 8th ACM European Conference on Computer Systems, pp. 239-252, 2013.
- Andrew Baumann, Marcus Peinado and Galen Hunt, "**Shielding Applications from an Untrusted Cloud with Haven**," OSDI'14 Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation, pp. 267-283, 2014.
- Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti and Vlad Zolotarov, "**OSv: Optimizing the Operating System for Virtual Machines**," USENIX ATC'14 Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference, pp. 61-72, 2014.
- Jacob R. Lorch, Andrew Baumann, Lisa Glendenning, Dutch Meyer and Andrew Warfield, "**Tardigrade: Leveraging Lightweight Virtual Machines to Easily and Efficiently Construct Fault-Tolerant Services**," NSDI'15 Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation, pp. 574-588, 2015.