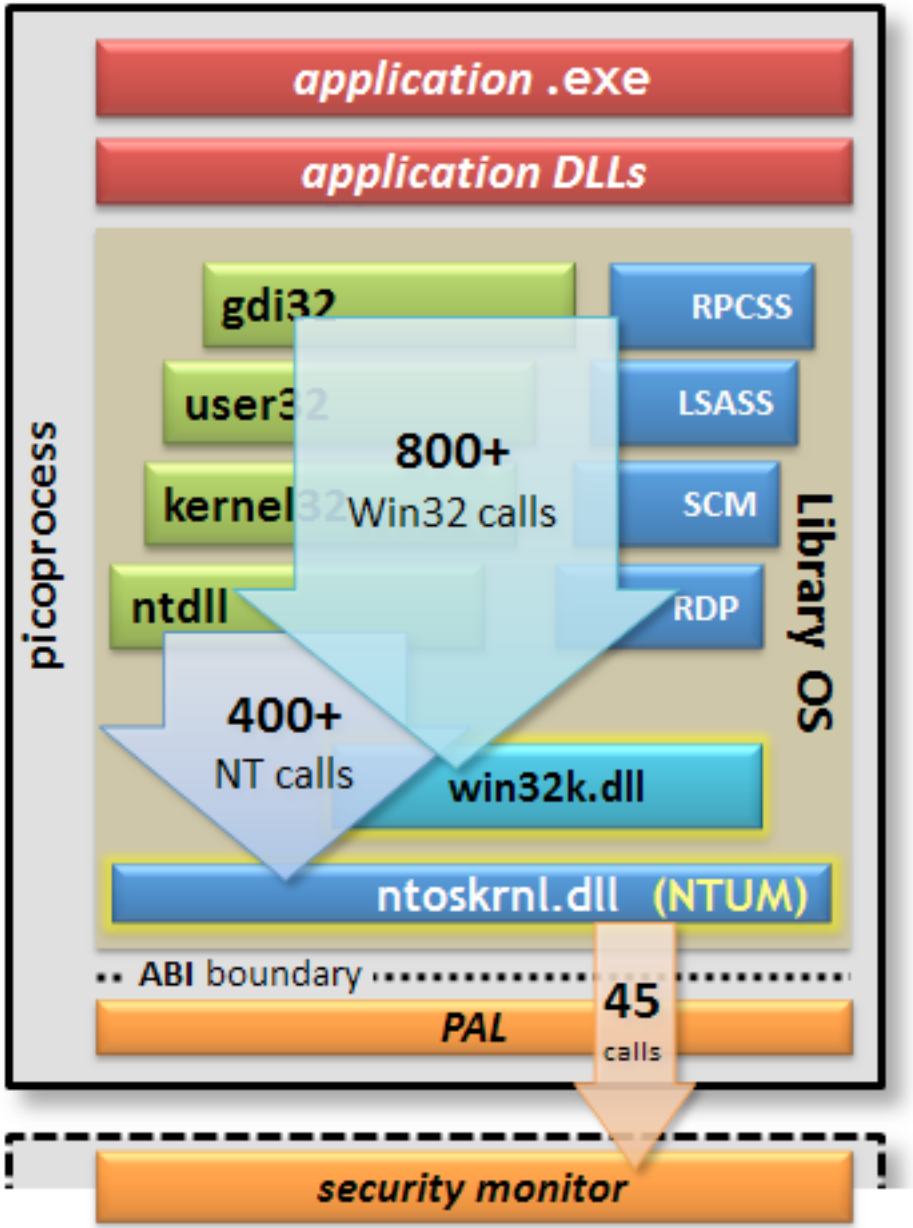
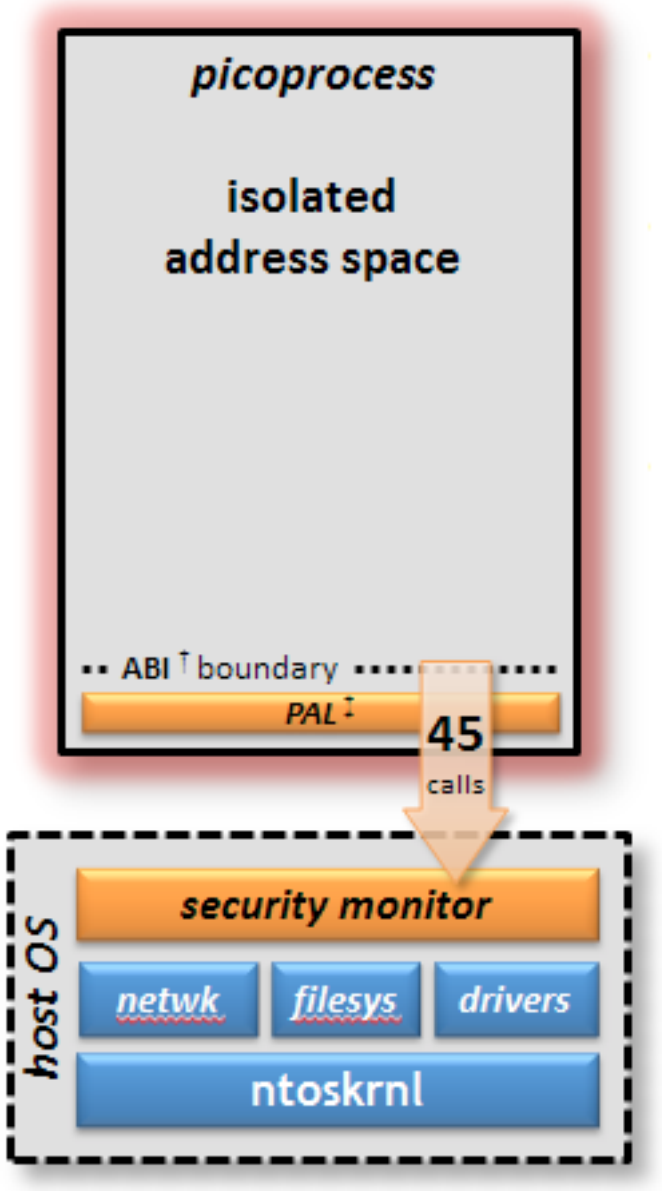
Slides for Drawbridge

Jeff Chase

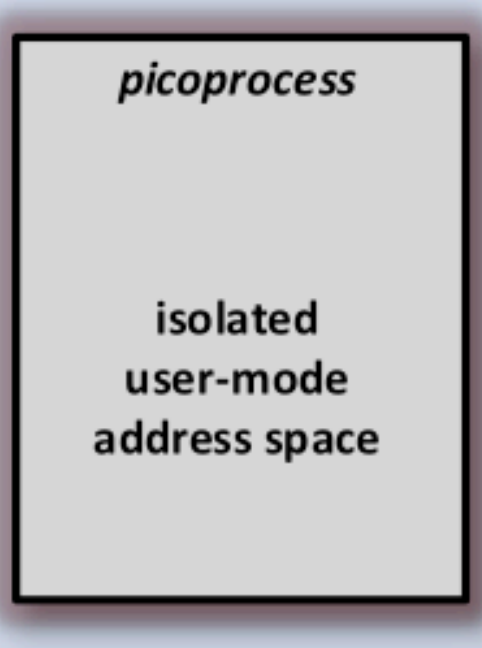
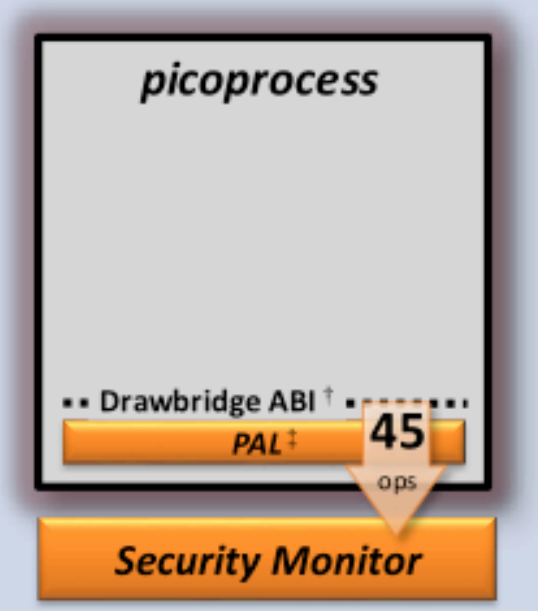
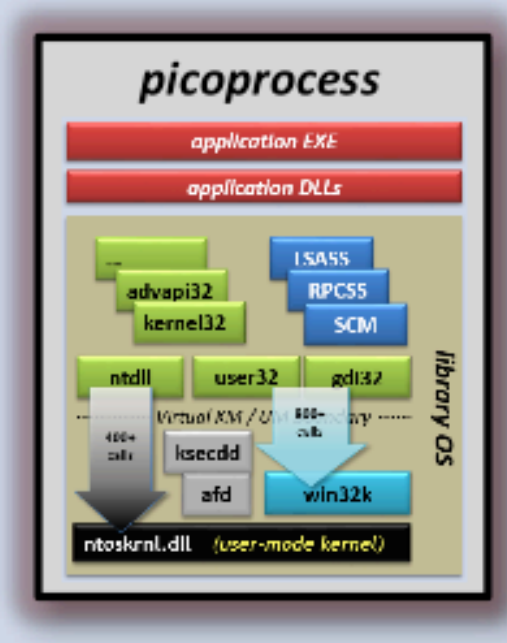
Drawbridge


application .exe
application DLLs
gdi32
user32
kernel32
ntdll
RPCSS
LSASS
SCM
RDP
800+
Win32 calls
400+
NT calls
win32k.dll
ntoskrnl.dll (NTUM)
ABI boundary
PAL
45
calls
security monitor
picoprocess
Library OS

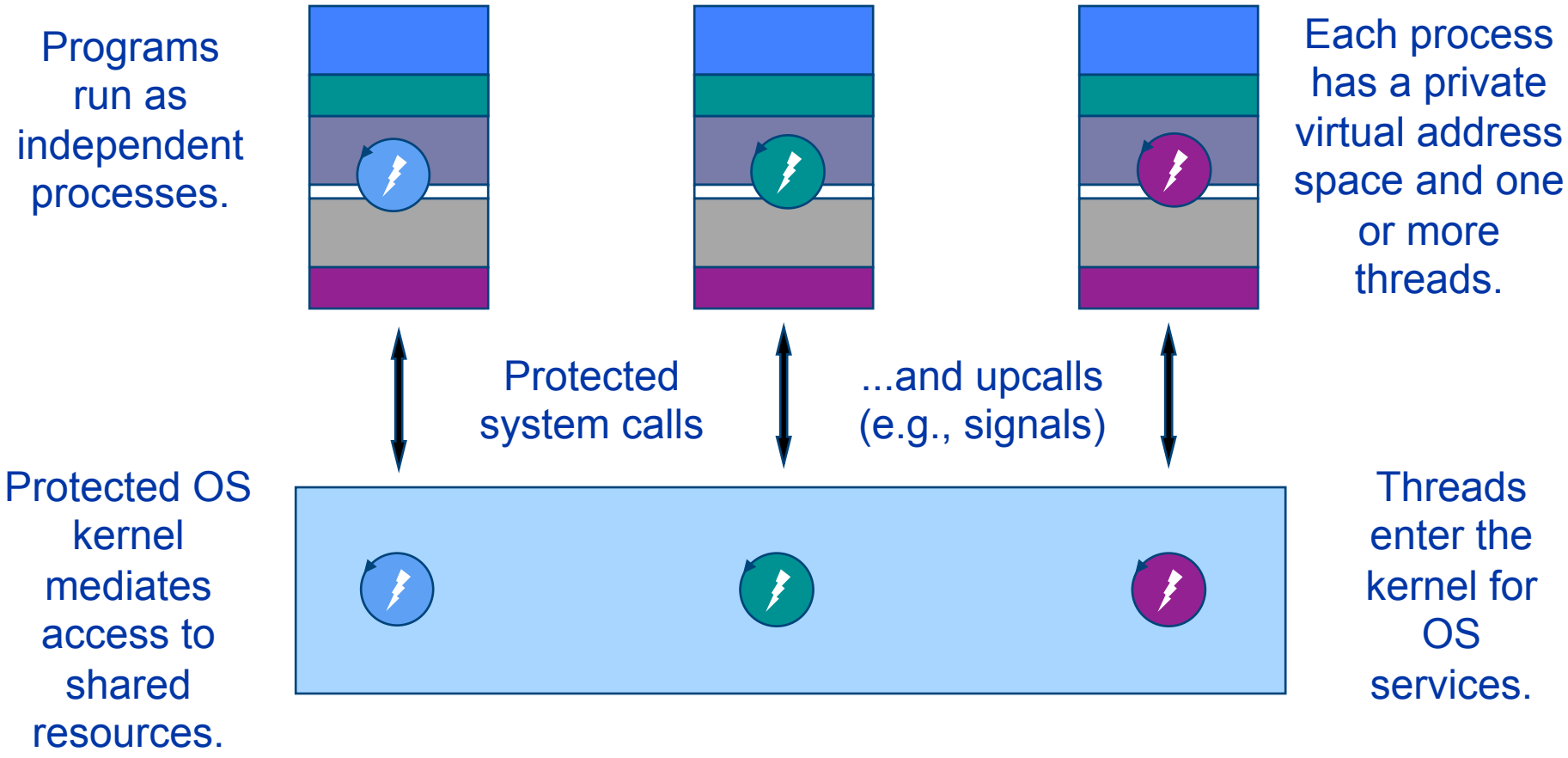

picoprocess
isolated
address space
ABI boundary
PAL
45
calls
security monitor
netwk
filesys
drivers
ntoskrnl
host OS

Summary

- Drawbridge is a light-weight VMs alternative for secure application hosting.
- Drawbridge consists of three pieces:

Picoprocess	Security Monitor	Library OS
		
• Secure container for code. • **No** access to the host OS.	• Provides primitive OS operations to picoprocs. (threads, VM, i/o streams)	• Subset of Windows w/ picoproc. enlightenments. ~16MB mem. overhead

Operating Systems: The Classical View

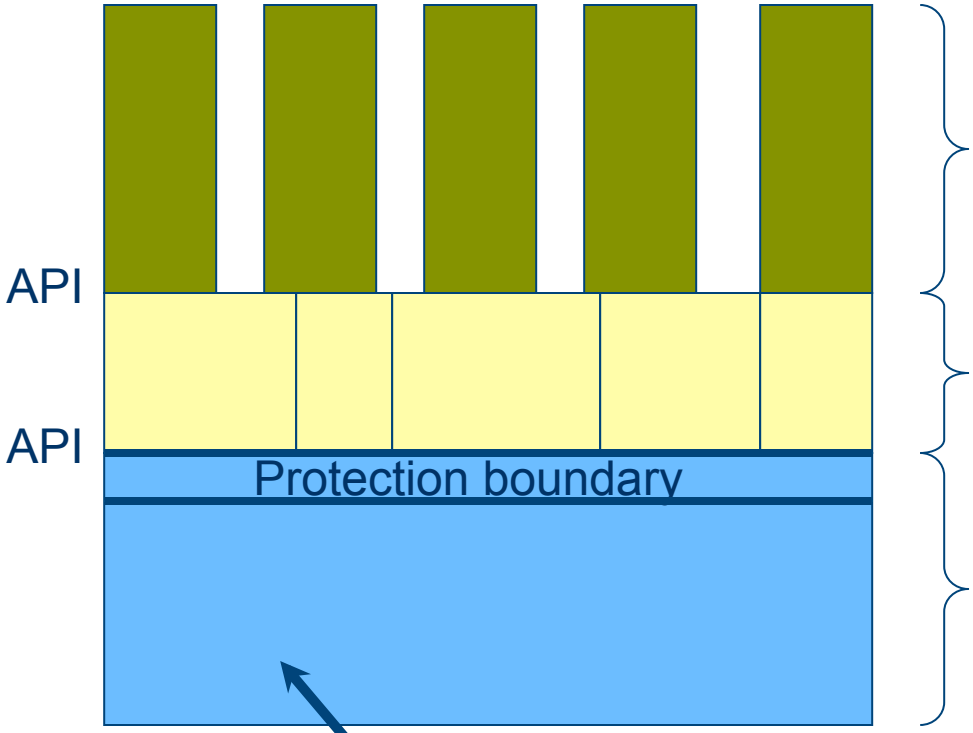


The kernel code and data are protected from untrusted processes.

Some questions

- What functions can/should be in the kernel?
- What functions can/should be in a library?
- What are the tradeoffs?
- What about sharing? Resource management?
  - From Drawbridge: registry, COM, files, display…
- What are the costs/benefits of a “minimal” kernel ABI?
  - Security? Portability? Transportability? (Migration)
- Why is Microsoft interested in Drawbridge?
- Why now?
- How does it differ from earlier microkernels, e.g., Mach?

OS Platform: A Model



Applications/services. May interact and serve one another.

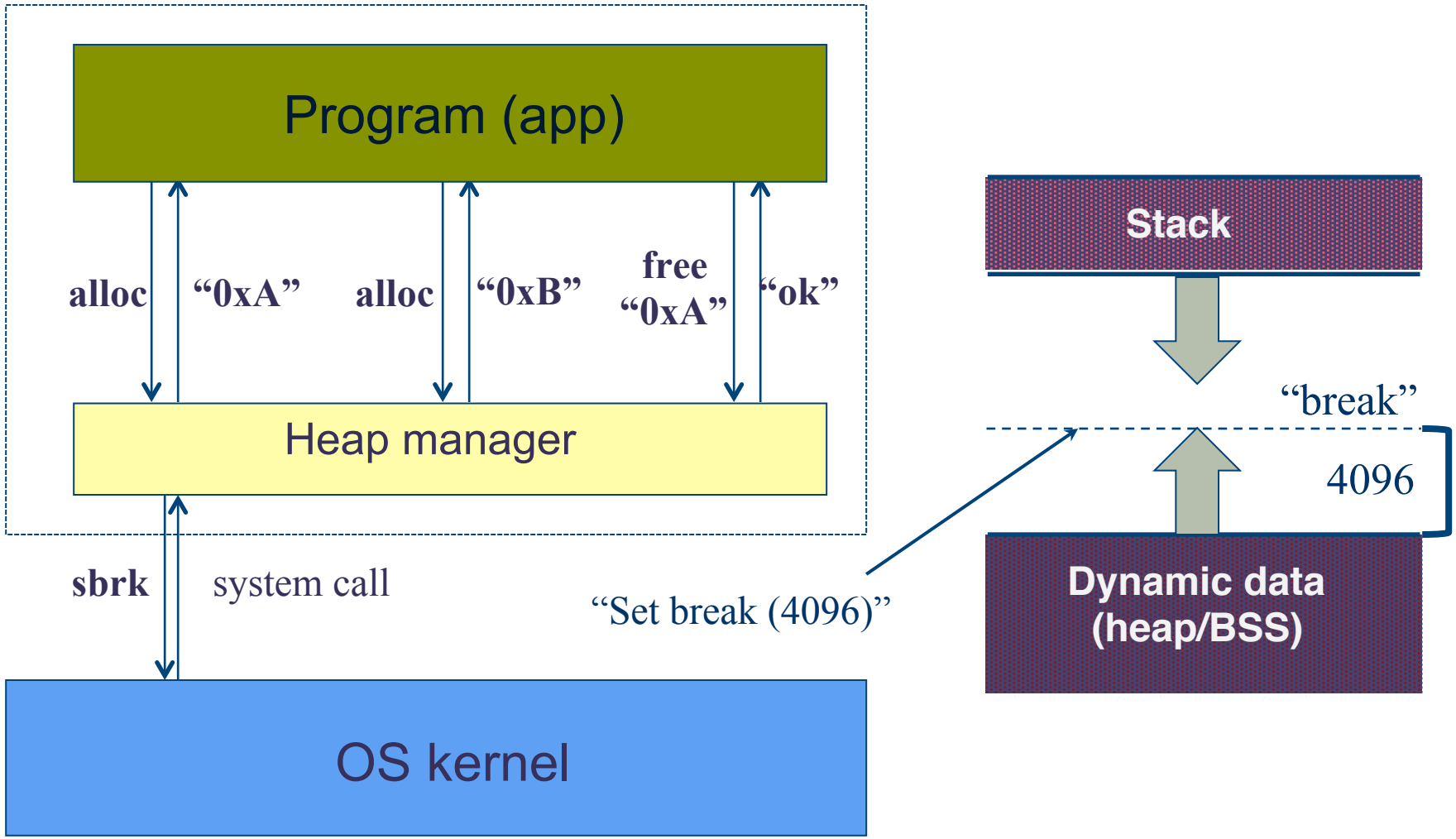
Libraries/frameworks: packaged code used by multiple applications

OS platform: same for all applications on a system E,g,, classical OS **kernel**

OS mediates access to shared resources. That requires protection and isolation.

[RAD Lab]

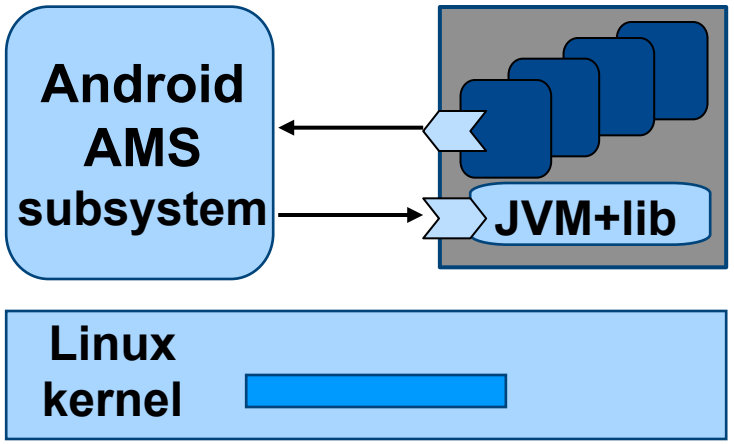
Example: heap manager



“Subsystems”

- A server process may provide trusted system functions to other processes, outside of the kernel.
  - E.g., this code is trusted, but like other processes it cannot manipulate the hardware state except by invoking the kernel.

- Example: Android **Activity Manager** subsystem provides many functions of Android, e.g., component launch and brokering of component interactions.
- With no special kernel support! It uses same syscalls as anyone else.
- AMS controls app contexts by forking them with a trusted lib, and issuing RPC commands to that lib.

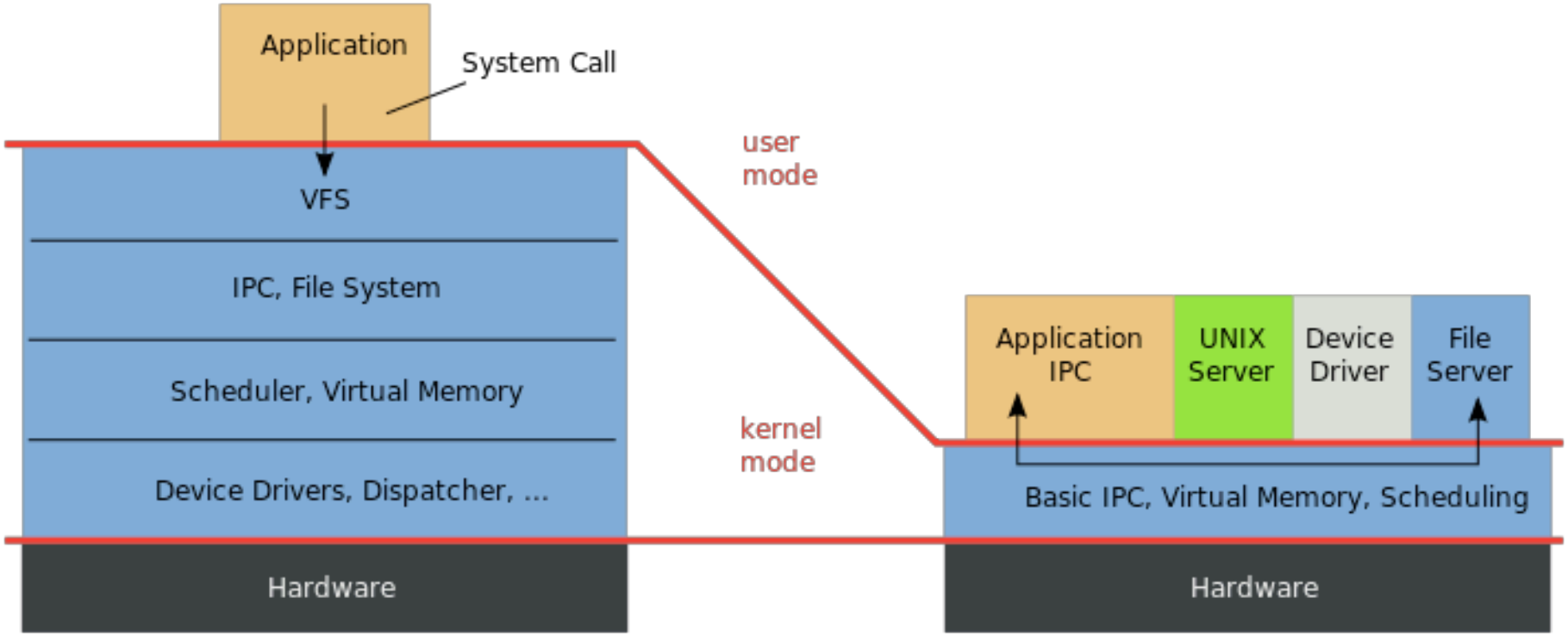


“binder” message driver in kernel

“OS as a service”

Monolithic Kernel
based Operating System

Microkernel
based Operating System



Point of “OS as a Service”

Kernel support for fast cross-domain call (“local RPC) enables OS services to be provided as user programs, outside the kernel, over a low-level “microkernel” syscall interface. This low-level syscall interface is not an API: it is hidden from applications, which are built to use the higher-level OS service APIs.

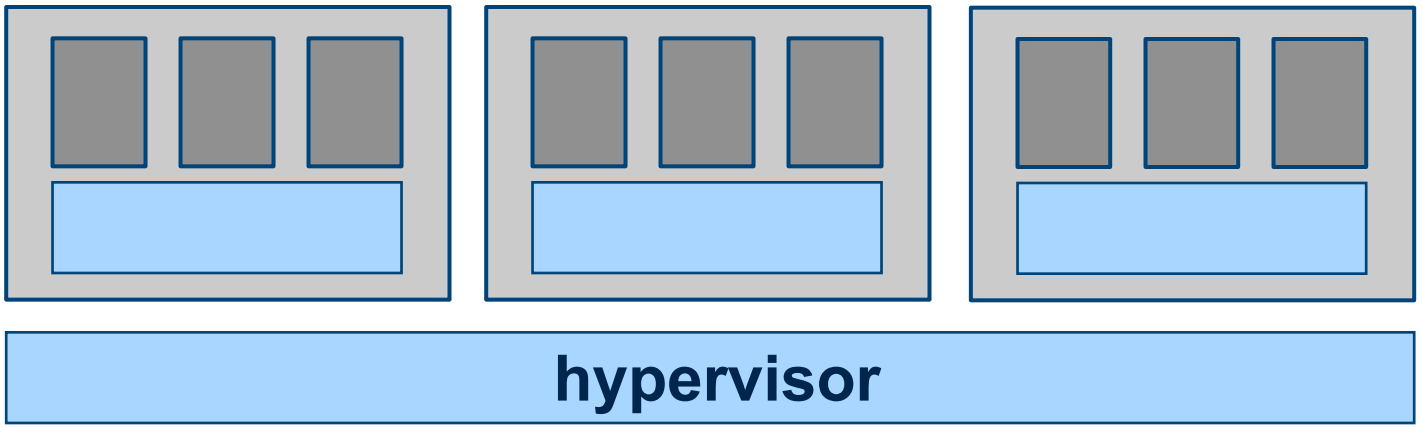
Many systems use this structure. Android uses it. Android is a collection of libraries and services over a “standard” Linux kernel, with binder supported added to the kernel as a plug-in module (a special device driver).

This structure originated with research “microkernel” systems in the 1980s, most notably the Mach project at CMU. The kernel code base for MacOSX derives substantially from Mach.

Windows uses this structure to some extent. Microsoft’s first modern OS was Windows NT (released in 1993). NT was strongly influenced by the research work in microkernels.

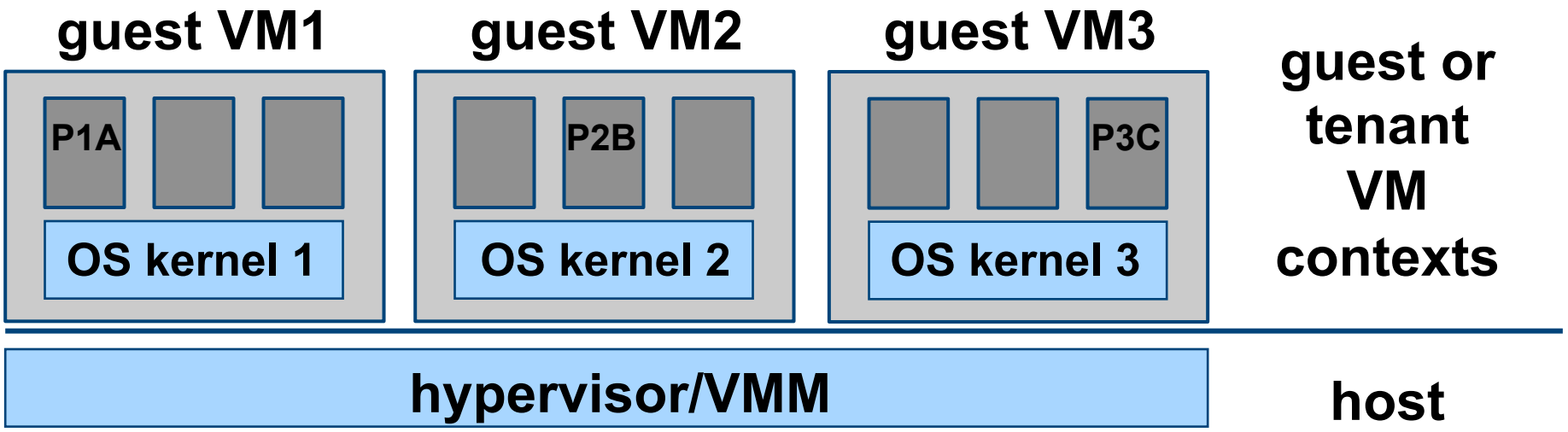
Native virtual machines (VMs)

- Slide a **hypervisor** underneath the kernel.
  - New OS layer: also called **virtual machine monitor (VMM)**.
- Kernel and processes run in a **virtual machine (VM)**.
  - The VM "looks the same" to the OS as a physical machine.
  - The VM is a sandboxed/isolated context for an entire OS.
- Can run multiple VM **instances** on a shared computer.




vmware


Xen


guest VM1
guest VM2
guest VM3
guest or tenant VM contexts
P1A
P2B
P3C
OS kernel 1
OS kernel 2
OS kernel 3
hypervisor/VMM
host


KVM

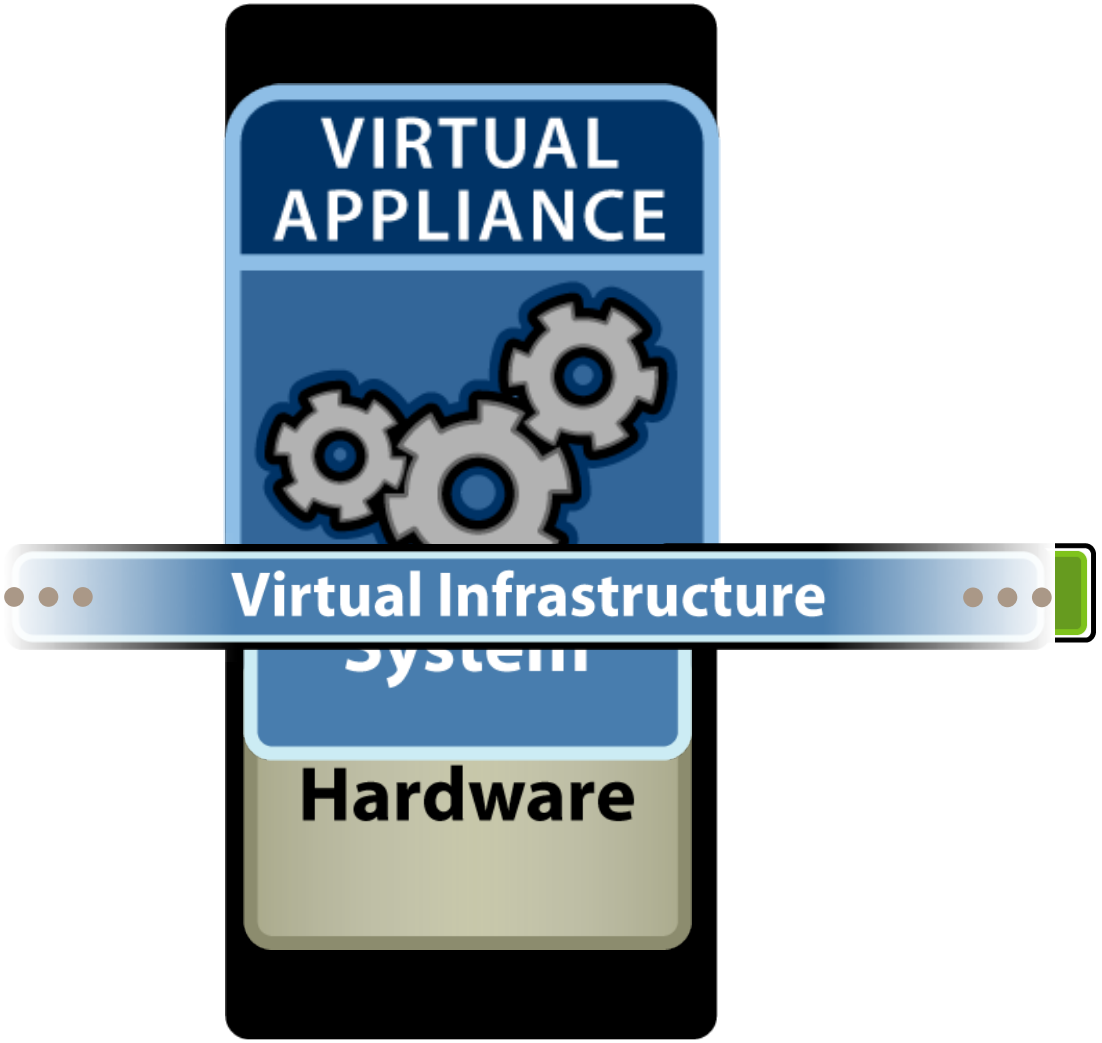

Sun
VirtualBox


Parallels

Image/Template/Virtual Appliance

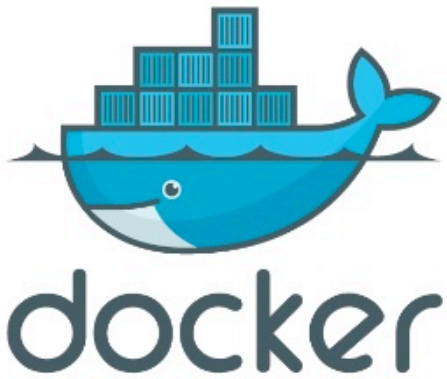
- A **virtual appliance** is a program for a virtual machine.
  - Sometimes called a **VM image** or **template**
- The image has everything needed to run a virtual server:
  - OS kernel program
  - file system
  - application programs
- The image can be instantiated as a VM on a cloud.
  - Not unlike running a program to instantiate it as a process

Thank you, VMware



Containers

- Note: lightweight container technologies offer a similar abstraction, but the VMs share a common kernel.
  - E.g., Docker



Docker, Containers, and the Future of Application Delivery

Drawbridge thread ABI/API

The ABI supports multithreading through five calls to create, sleep, yield the scheduler quantum for, resume execution of, and terminate threads, as well as seven calls to create, signal, and block on synchronization objects:

```
DKHANDLE DkThreadCreate(Addr, Param, Flags);
DkThreadDelayExecution(Duration);
DkThreadYieldExecution();
DkThreadResume(ThreadHandle);
DkThreadExit();
DKHANDLE DkSemaphoreCreate(InitialCount, MaxCount);
DKHANDLE DkNotificationEventCreate(InitialState);
DKHANDLE DkSynchronizationEventCreate(InitialState);
DkSemaphoreRelease(SemaphoreHandle, ReleaseCount);
BOOL DkEventSet(EventHandle);
DkEventClear(EventHandle);
ULONG DkObjectsWaitAny(Count, HandleArray, Timeout);
```

Bascule thread ABI (refines Drawbridge)

```
SemaphoreCreate(InitCount, MaxCount) -> SemaphoreHandle
SemaphoreRelease(SemaphoreHandle, ReleaseCount)
SemaphorePeek(SemaphoreHandle) -> Count
NotificationEventCreate(InitialState) -> EventHandle
SynchronizationEventCreate(InitialState) -> EventHandle
EventSet(EventHandle)
EventClear(EventHandle)
EventPeek(EventHandle) -> State
ObjectReference(Handle)
ObjectClose(Handle)
ObjectsWaitAny(Count, Handles, Timeout) -> Index
ThreadCreate(Routine, Arg, Stack, Params) -> ThreadHandle
ThreadExit()
ThreadYieldExecution()
ThreadRaiseException(ThreadHandle, OpaquePointer)
```

Bascule/Drawbridge thread ABI

```
ThreadCreate(Routine, Arg, Stack, Params) -> ThreadHandle
ThreadExit()
ThreadYieldExecution()
ThreadRaiseException(ThreadHandle, OpaquePointer)
```

Bascule/Drawbridge semaphore ABI

```
SemaphoreCreate(InitCount, MaxCount) -> SemaphoreHandle
SemaphoreRelease(SemaphoreHandle, ReleaseCount)
SemaphorePeek(SemaphoreHandle) -> Count

ObjectsWaitAny(Count, Handles, Timeout) -> Index
```

Bascule/Drawbridge event ABI

```
NotificationEventCreate(InitialState) -> EventHandle
SynchronizationEventCreate(InitialState) -> EventHandle
EventSet(EventHandle)
EventClear(EventHandle)
EventPeek(EventHandle) -> State

ObjectsWaitAny(Count, Handles, Timeout) -> Index
```

Drawbridge I/O: streams

The primary I/O mechanism in Drawbridge is an I/O stream. I/O streams are byte streams that may be memory-mapped or sequentially accessed.

Streams are named by URIs…Supported URI schemes include file:, pipe:, http:, https:, tcp:, udp:, pipe.srv:, http.srv, tcp.srv:, and udp.srv:. The latter four schemes are used to open inbound I/O streams for server applications:

```
DKHANDLE DkStreamOpen(URI, AccessMode, ShareFlags,
       CreateFlags, Options);
ULONG DkStreamRead(StreamHandle, Offset, Size, Buffer);
ULONG DkStreamWrite(StreamHandle, Offset, Size, Buffer);
DkStreamMap(StreamHandle, Addr, ProtFlags, Offset, Size);
DkStreamUnmap(Addr);
DkStreamSetLength(StreamHandle, Length);
DkStreamFlush(StreamHandle);
DkStreamDelete(StreamHandle);
DkStreamWaitForClient(StreamHandle);
DkStreamGetName(StreamHandle, Flags, Buffer, Size);
DkStreamAttributesQuery(URI, DK_STREAM_ATTRIBUTES *Attr);
DkStreamAttributesQueryByHandle(StreamHandle,
       DK_STREAM_ATTRIBUTES *Attr);
```

Butler W. Lampson

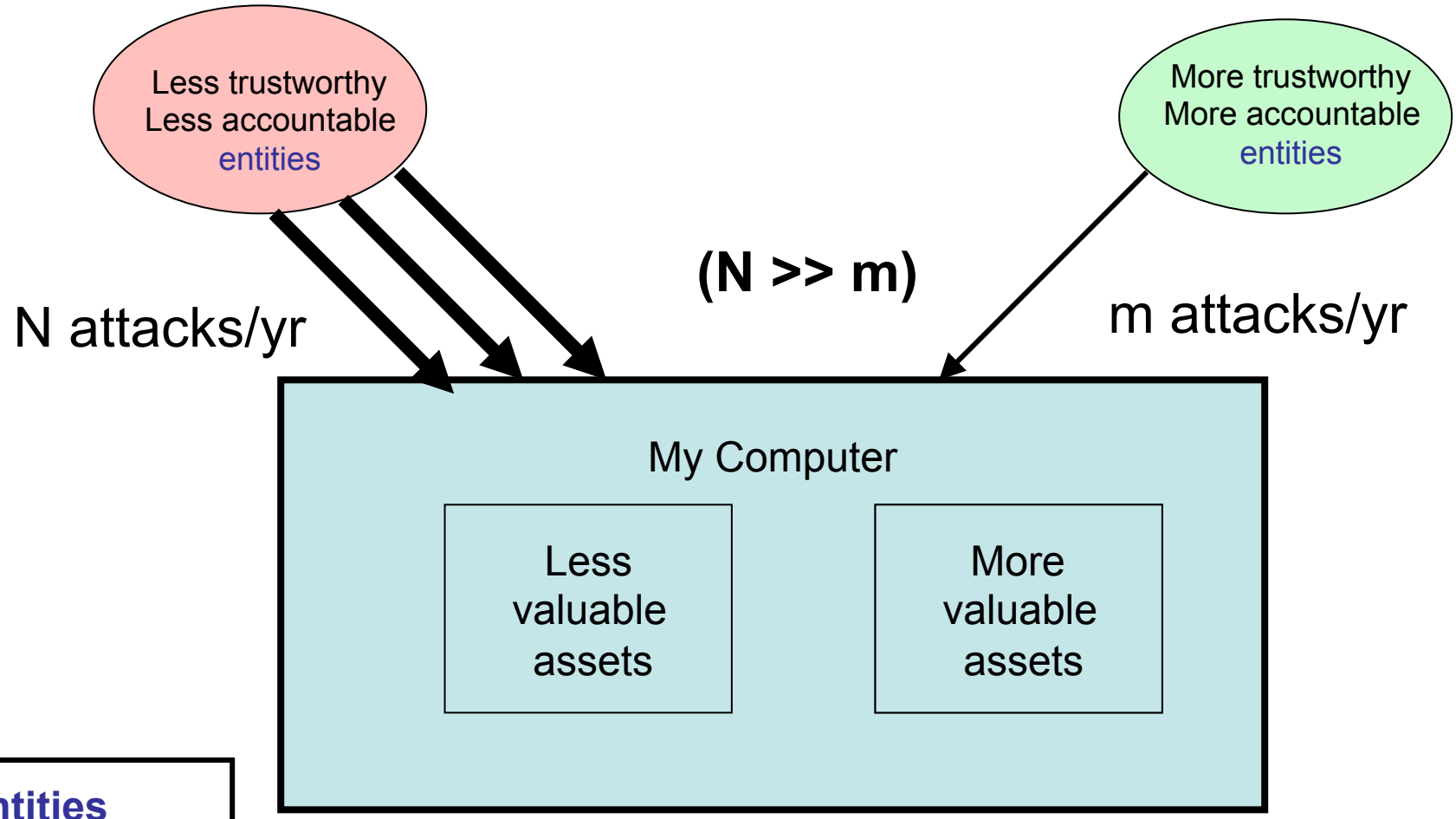
http://research.microsoft.com/en-us/um/people/blampson

Butler Lampson is a Technical Fellow at Microsoft Corporation and an Adjunct Professor at MIT…..He was one of the designers of the SDS 940 time-sharing system, the Alto personal distributed computing system, the Xerox 9700 laser printer, two-phase commit protocols, the Autonet LAN, the SPKI system for network security, the Microsoft Tablet PC software, the Microsoft Palladium high-assurance stack, and several programming languages. He received the ACM Software Systems Award in 1984 for his work on the Alto, the IEEE Computer Pioneer award in 1996 and von Neumann Medal in 2001, the Turing Award in 1992, and the NAE's Draper Prize in 2004.

Accountability vs. Freedom

- Partition world into two parts:
  - Green Safer/accountable
  - Red Less safe/unaccountable
- Two aspects, mostly orthogonal
  - User Experience
  - Isolation mechanism
    - Separate hardware with air gap
    - VM
    - Process isolation

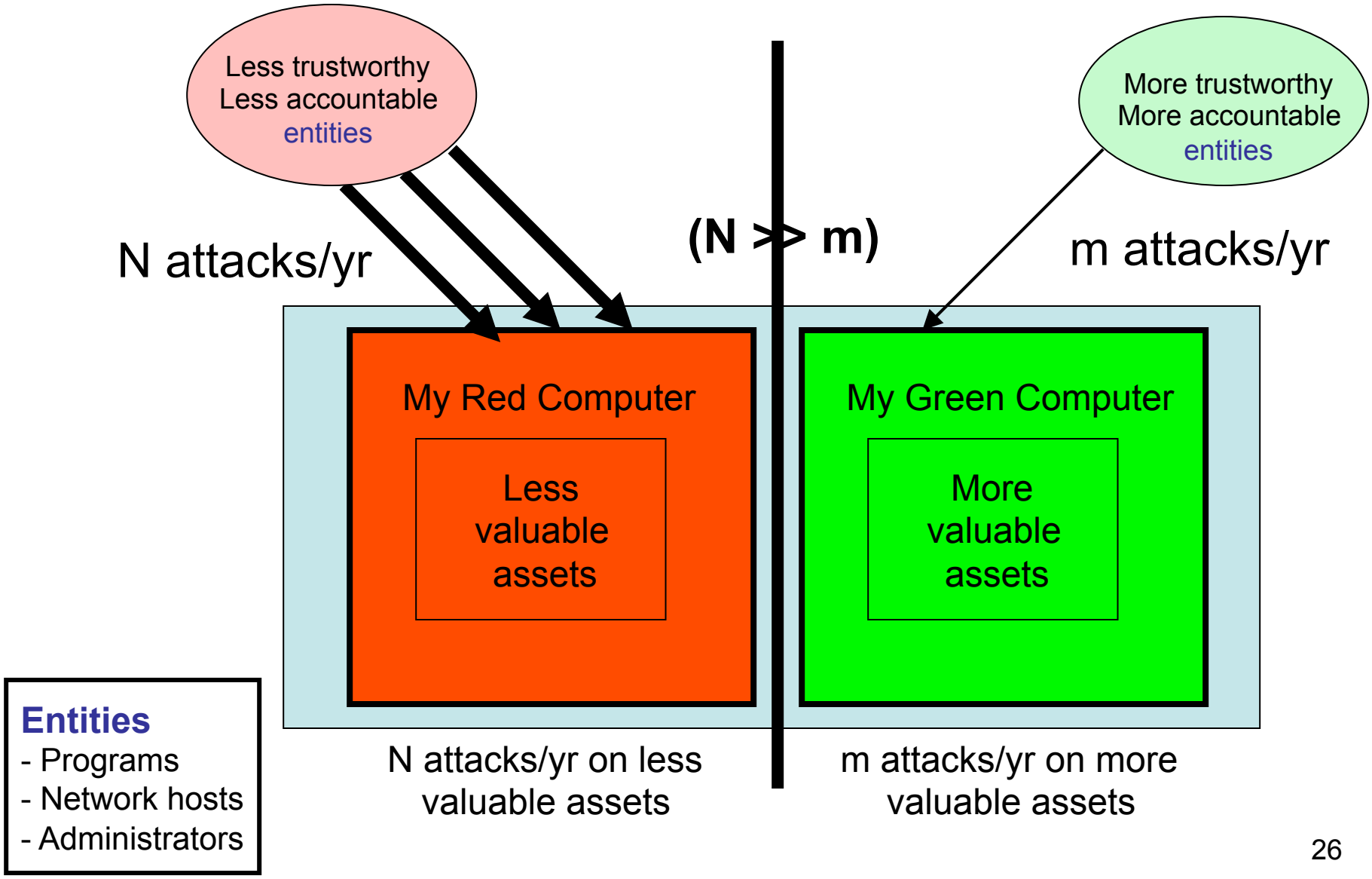
Without R|G: Today



Entities
- Programs
- Network hosts
- Administrators

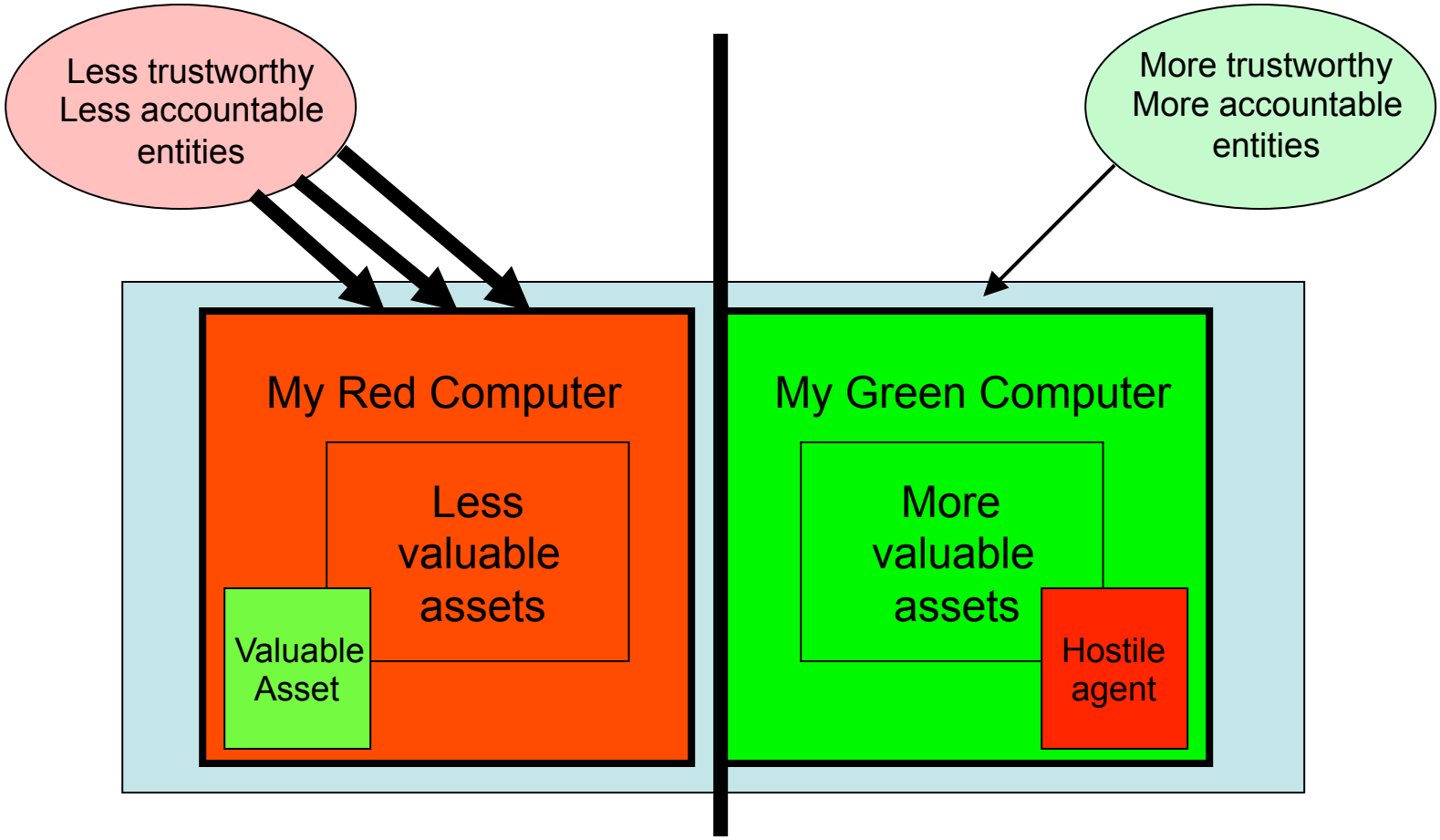
Total: N+m attacks/yr on all assets

With R|G


Less trustworthy
Less accountable
entities
More trustworthy
More accountable
entities
N attacks/yr
(N >> m)
m attacks/yr
My Red Computer
Less valuable assets
My Green Computer
More valuable assets
N attacks/yr on less valuable assets
m attacks/yr on more valuable assets
Entities
- Programs
- Network hosts
- Administrators

Must Get Configuration Right

- Keep valuable stuff out of red
- Keep hostile agents out of green



Why R|G?

- Problems:
  - Any OS will always be exploitable
    - The richer the OS, the more bugs
  - Need internet access to get work done, have fun
    - The internet is full of bad guys
- Solution: Isolated work environments:
  - Green: important assets, only talk to good guys
    - Don't tickle the bugs, by restricting inputs
  - Red: less important assets, talk to anybody
    - Blow away broken systems
- Good guys: more trustworthy / accountable
  - Bad guys: less trustworthy or less accountable