Apiary: Easy-to-Use Desktop Application Fault Containment on Commodity Operating Systems

Shaya Potter and Jason Nieh

June 23, 2010
USENIX ATC

IBM Research
Research performed at Columbia University

Desktop Applications are Buggy!

- Desktop applications are prone to being exploited
  - Adobe Acrobat – multiples times in 2009-2010
    - PDF has dethroned MS Word documents as most common malware vector [F-Secure]
- But why should this even be possible?
  - I want to view the PDF as a “read-only” item

Approaches to Application Security

- Access Control Systems
  - Ex: Janus, Systrace, SELinux…
- Rewrite/Recompile Applications
  - Ex: Java, Google's Native Client
- Isolating Applications in Virtual Machines
  - Ex: VMware Unity

Isolated VMs for each Application?

Pros

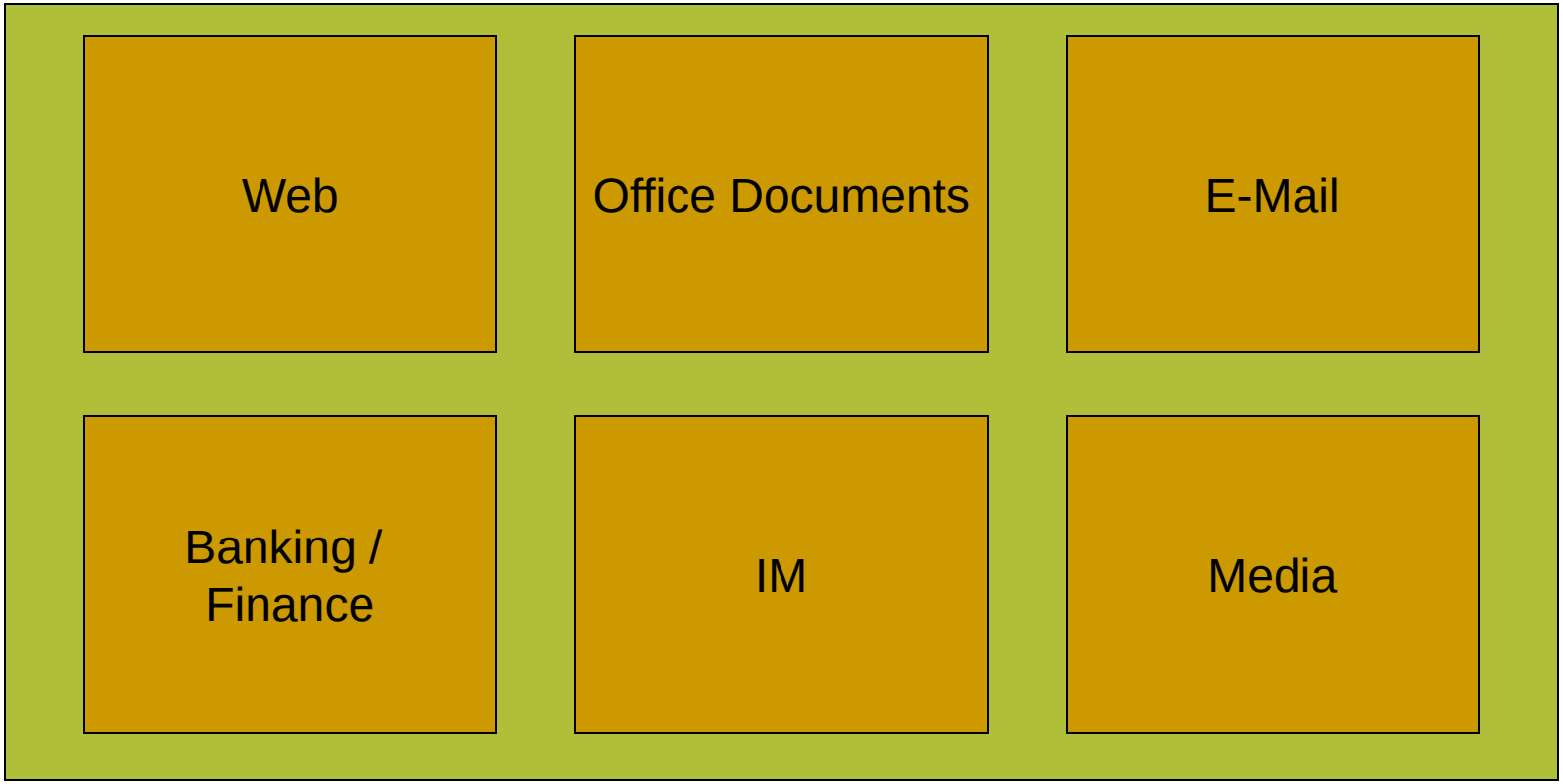
- No need to make complex rules
- Exploited applications are isolated
- Works with existing applications

Cons

- Exploited applications remain exploited
- Significant runtime overhead
- Lose integrated desktop feel
- Increase management burden

Apiary

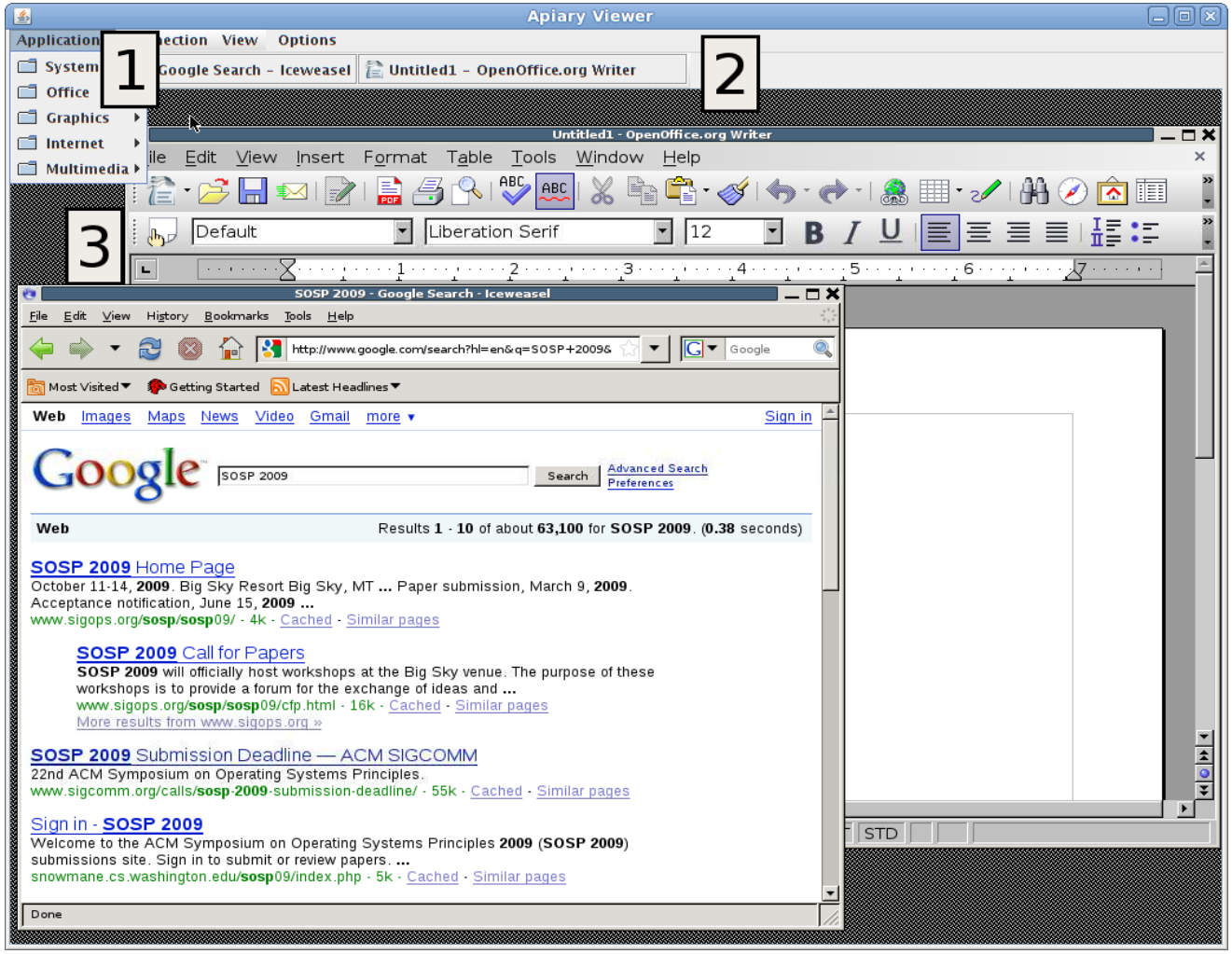
Desktop Applications are Isolated



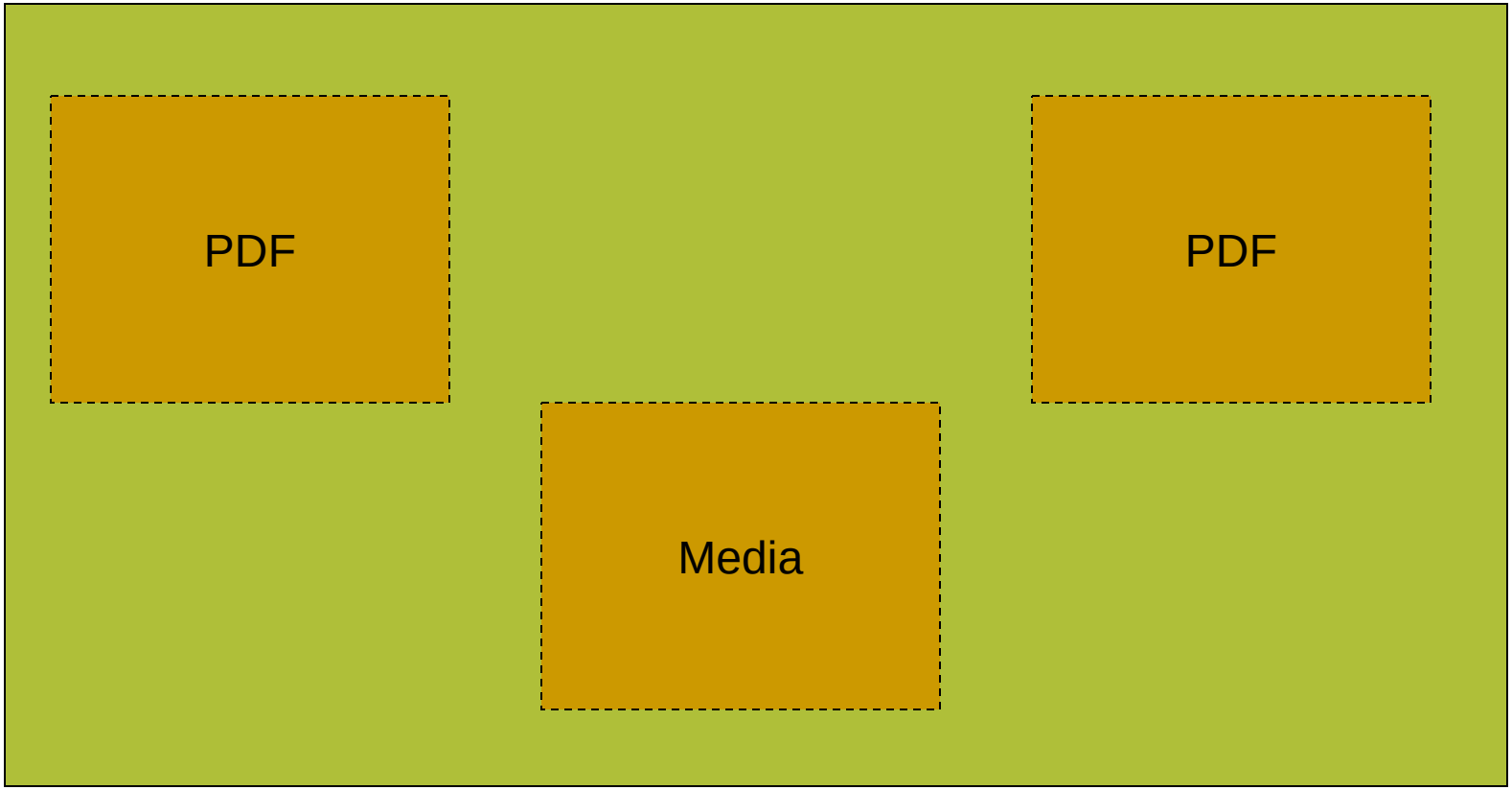
Persistent Application Containers

- Changes persist between application execution
- Needed for persistent data
  - Quicken
  - Research Papers
- But persistent data still needs to be isolated
  - Office documents have no need to access financial data in Quicken

Apiary Retains Desktop Look and Feel



Introduces Ephemeral Containers



Ephemeral Application Containers

- Compromises cannot persist
- Protects from concurrent compromises
- Protects privacy

- Enables untrusted data to be viewed safely

Problems to Solve

- ~~Exploited applications remain exploited~~
- Significant overhead
- Lose integrated desktop feel
- Increase management burden

Apiary's Architecture

- 3 Components

1. OS Containers
2. Display Virtualization
3. Virtual Layered File System (VLFS)

OS Containers

- OS Containers are prevalent on commodity OSs
  - Solaris Zones, Linux Containers/VServer
- Low overhead
- Quick to instantiate
- Lower isolation than hardware VMs
  - Apiary can be used with hardware VMs if threat model requires it

Problems to Solve

- ~~Exploited applications remain exploited~~
- ~~Significant overhead~~
- Lose integrated desktop feel
- Increase management burden

Containers Integrated at Multiple Points

1. Display
2. Inter-Application Execution
3. File System

Integrated Display

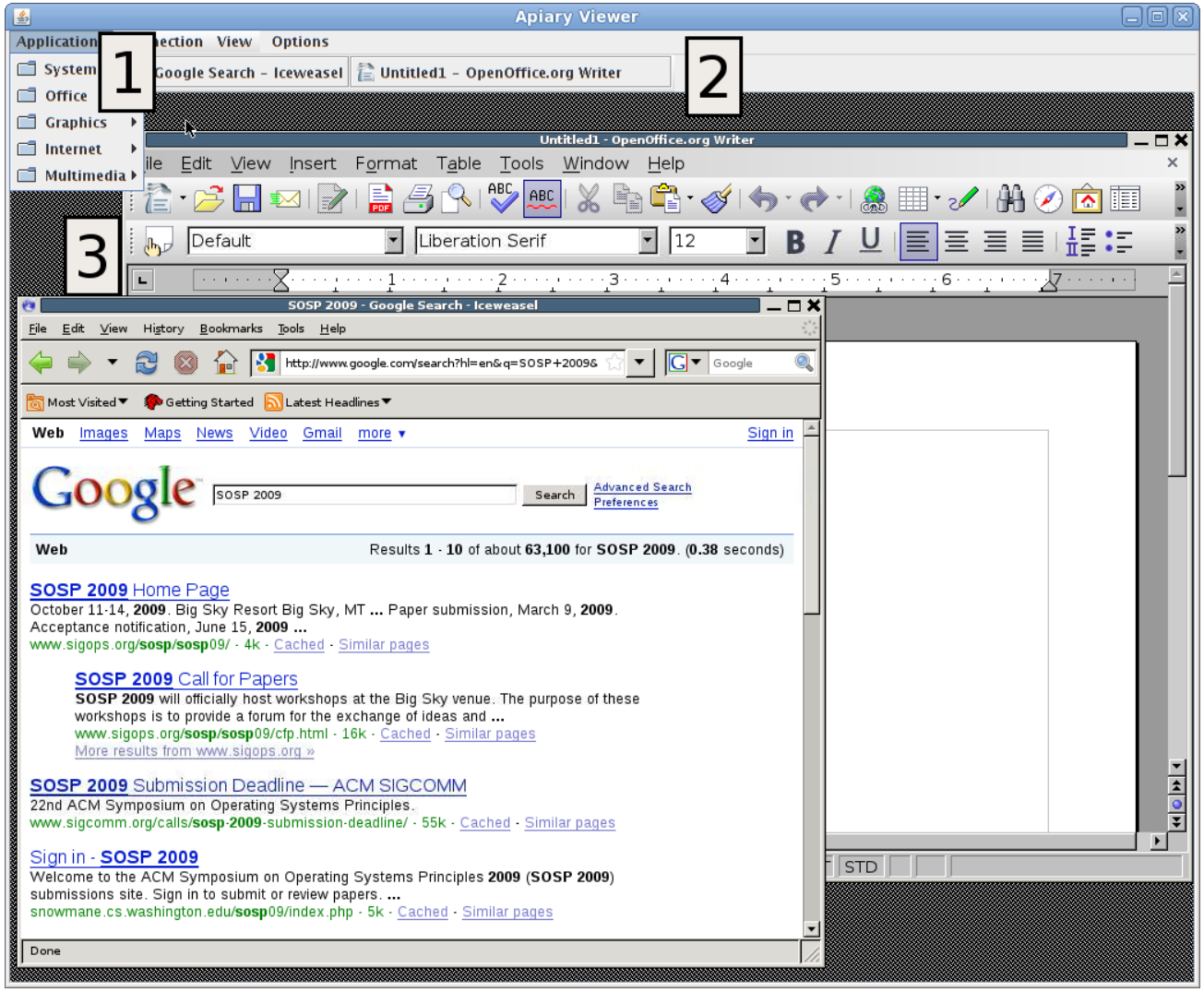
Problem

- Each container must have isolated displays
  - `XSendEvent()` / `W32SendMessage()` are vectors to exploit other running applications
- But, need a single desktop environment

Solution

- Provide each container with its own virtual display server
- Viewer composes together containers' displays
  - Single display, menu, task bar

Display Integration



Integrated Applications

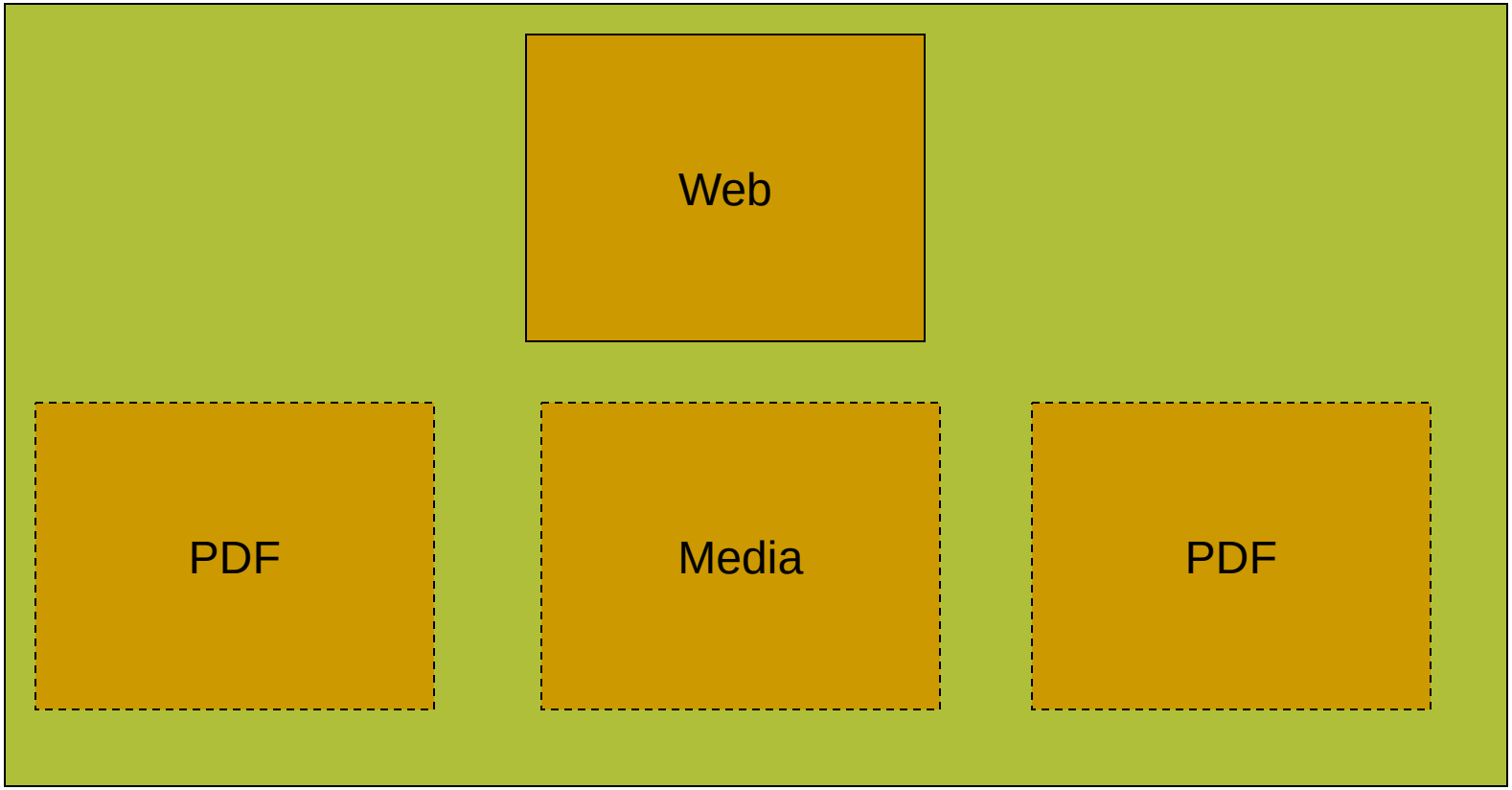
Problem

- Applications in different containers depend on each other
  - Firefox wants to run a PDF viewer or OpenOffice to view documents

Solution

- Applications can execute each other in an ephemeral helper mode

Integrated Applications



Integrated File System

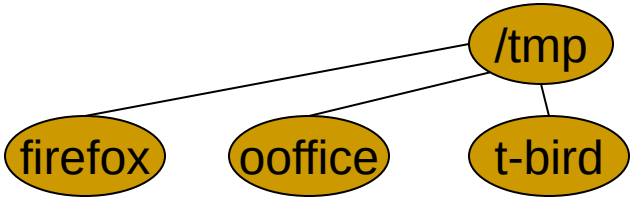
Problem:

- Ephemeral helper applications are useless if data can't be shared
  - How does Firefox pass the PDF file to the PDF viewer?

Solution

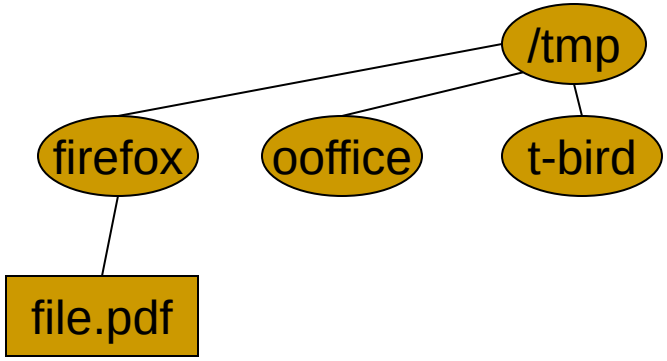
- Limited File System Integration
  - Protected/Shared "`/tmp`" for inter-application execution

Integrated File System – `/tmp`



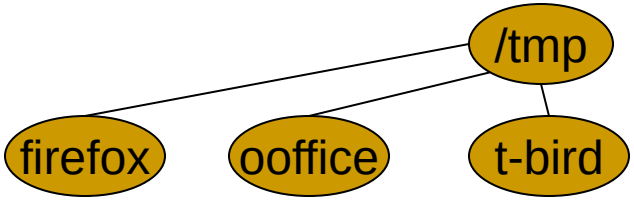
- Each container has its own directory under `/tmp`

Integrated File System – `/tmp`



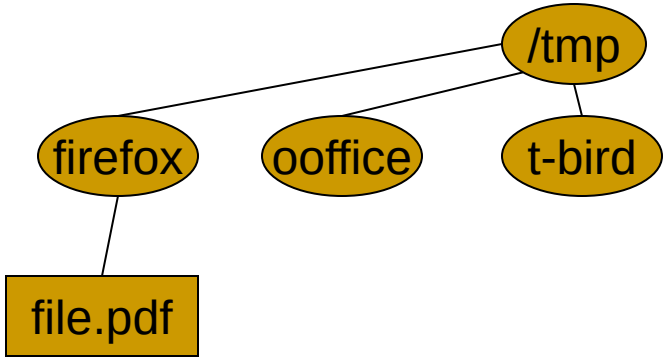
- Each container uses that directory as its own temp directory
  - Firefox will save all temporary files to `/tmp/firefox`

Integrated File System – `/tmp`



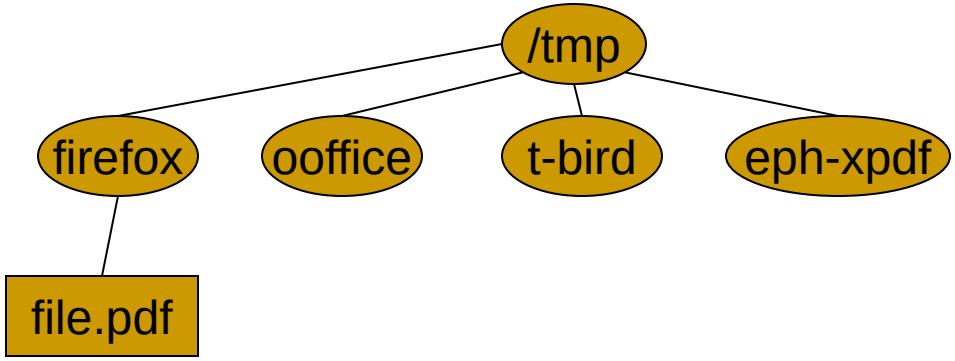
- But files are invisible to other containers

Integrated File System – `/tmp`



- Firefox will launch `xpdf /tmp/firefox/file.pdf`

Integrated File System – `/tmp`



- Creates a new ephemeral container for Xpdf
- Allows `/tmp/firefox/file.pdf` to be visible in the new ephemeral Xpdf container
- Ephemeral Xpdf container executes program as called

Integrated File System – Global View

Problem

- Files might need to be shared between isolated containers.

Solution

- File System Manager Container
  - Provides a global namespace view to move files between containers

Problems to Solve

- ~~Exploited applications remain exploited~~
- ~~Significant overhead~~
- ~~Lose integrated desktop feel~~
- Increase management burden

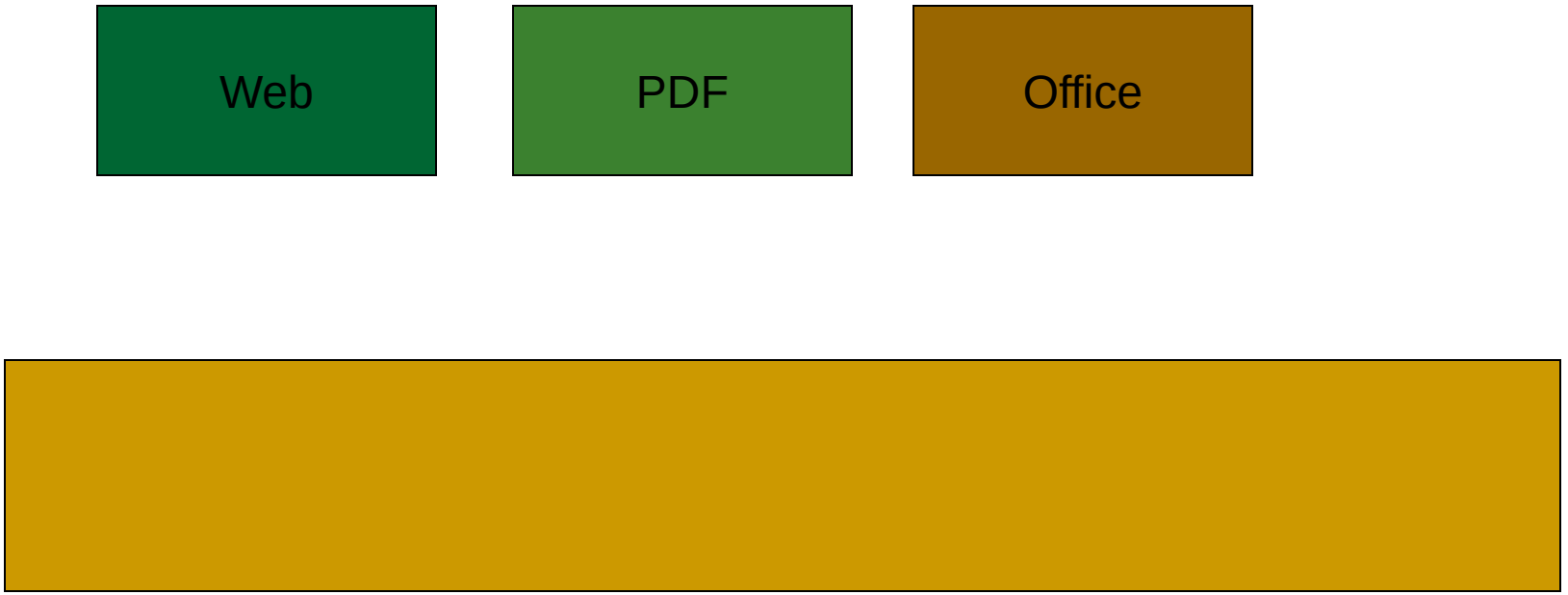
Container Management Problems

- How do we efficiently provision them?
- How do we efficiently store them?
- How do we efficiently get updates applied?

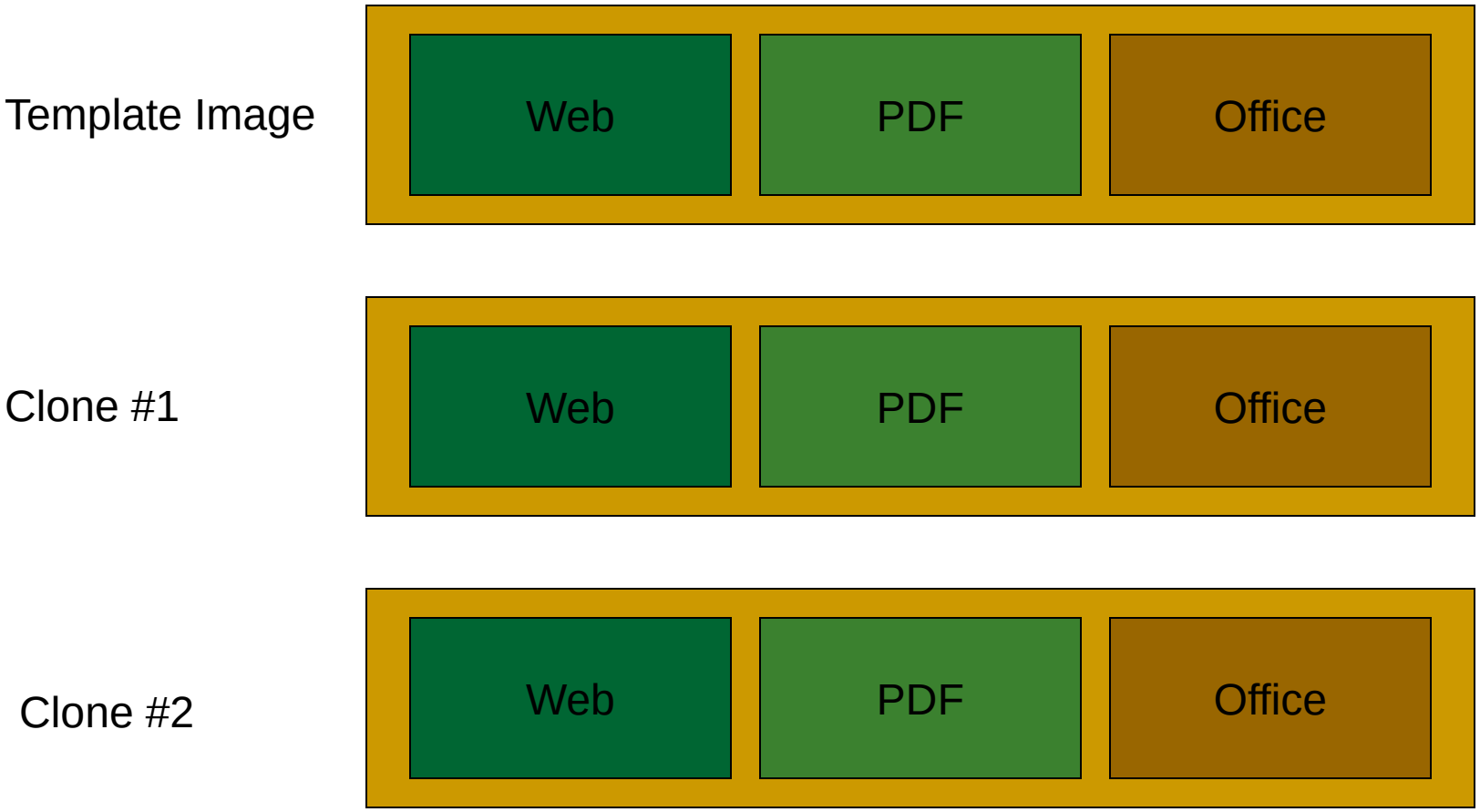
Possible Approaches?

- Package Management
- COW Disks/File Systems

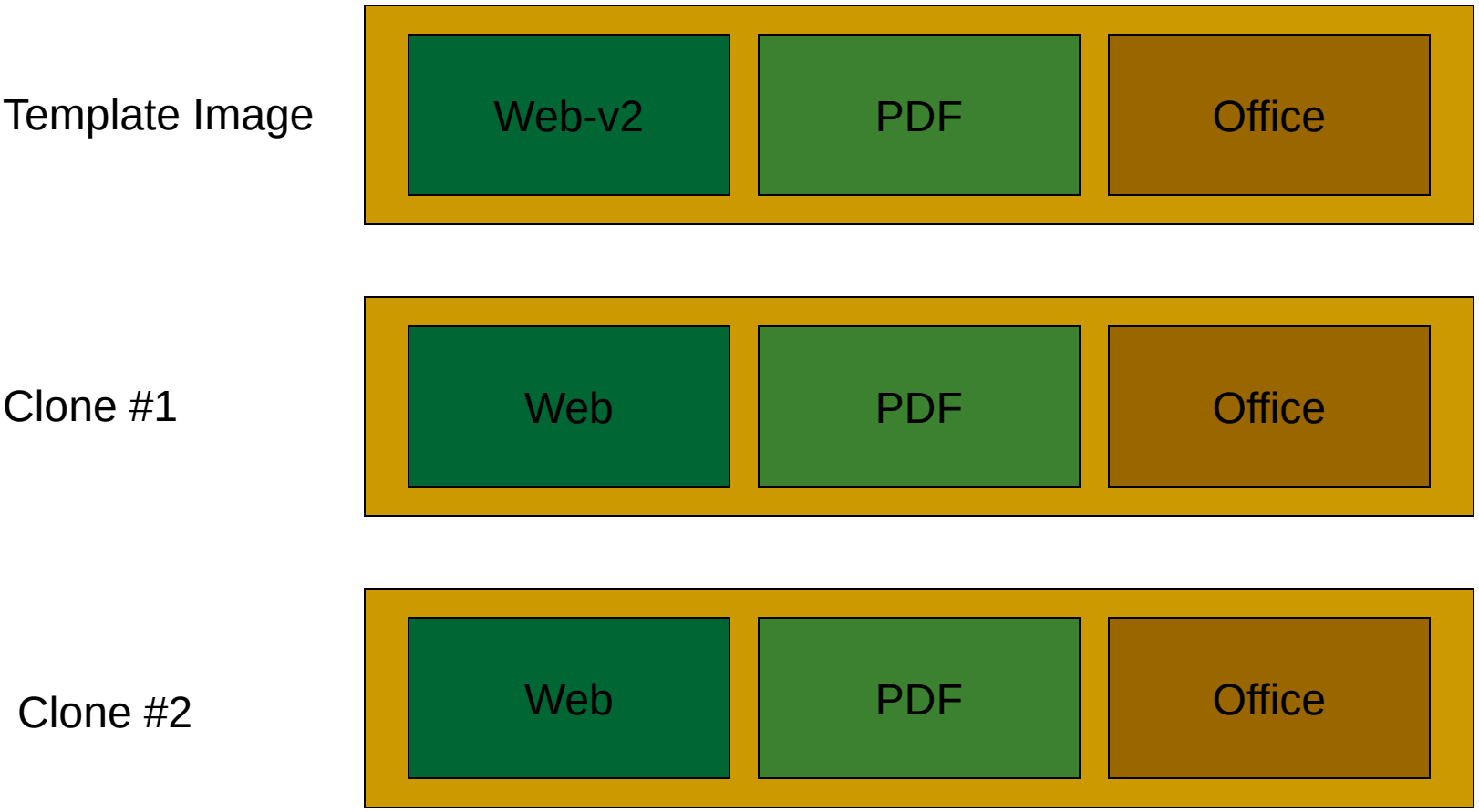
Package Management



COW Disks/File Systems



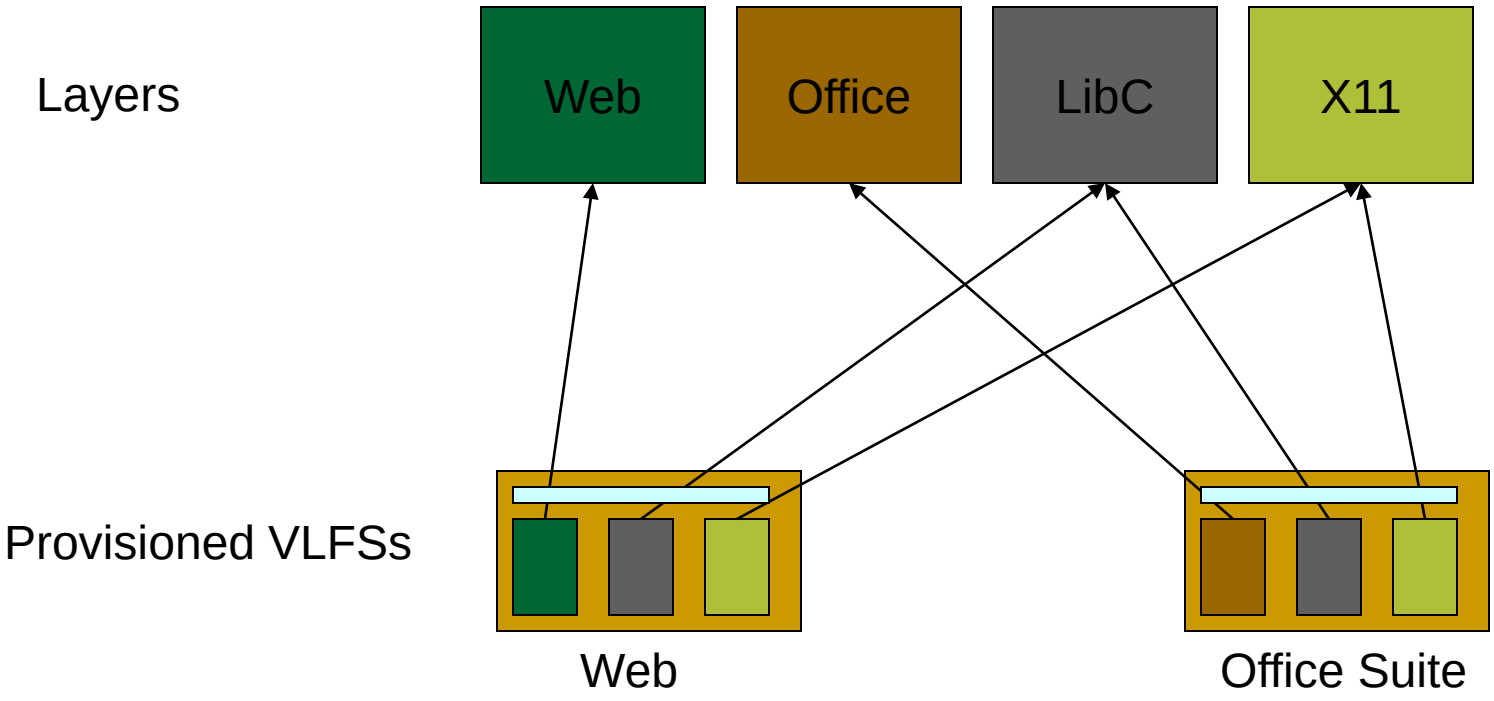
COW Disks/File Systems



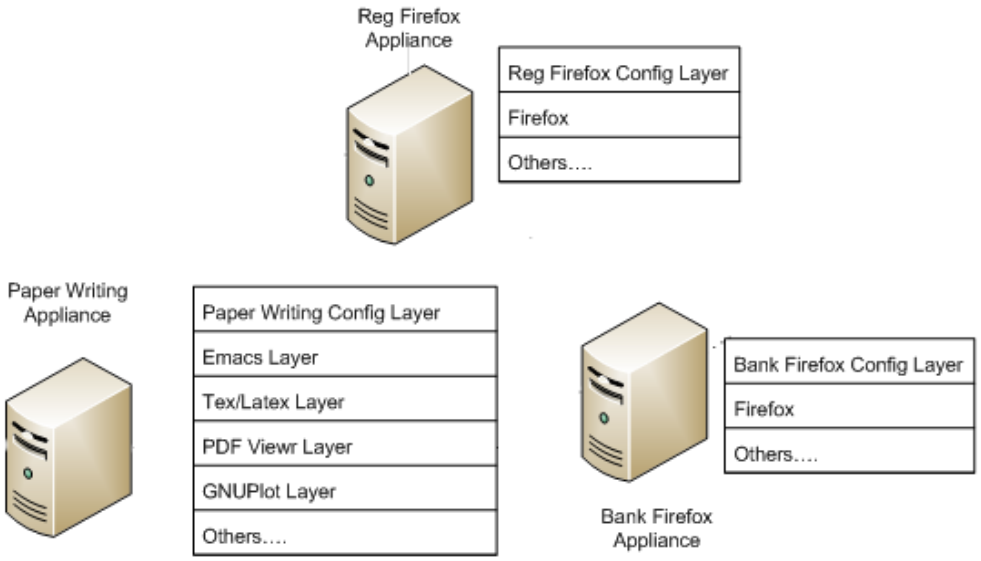
The Virtual Layered File System

- Makes the FS a full partner with the package manager
  - Packages are transformed into a set of shared layers
  - Combine Unioning File System concepts with package management

VLFS Example



The VLFS/Software Appliance



- VLFS defines Software Appliance

How Apiary Uses the VLFS

- Users install application appliances instead of individual applications
  - Predefined sets of layers
  - Able to be created by various organizations
    - Banks
    - ISVs
- Appliances leverage global set of layers
  - Don’t need to manage systems from scratch

How Does it Solve the Problems?

- How do we efficiently provision them?
  - Shared Layers means no copying
  - Instantly able to create file systems for ephemeral execution
- How do we efficiently store them?
  - Each common layer is only stored once, like a regular system
- How do we efficiently get updates applied?
  - Update layer once in repository, able to be used by all application containers that depend on that layer

Other VLFS Advantages

- How do we make sure they are secure?
  - Dividing into layers isolates changes, makes malicious changes visible
- Avoids “DLL Hell”
  - Each application container has its own independent set of shared libraries
  - Allows incompatible applications to be installed in same machine

Problems to Solve

- ~~Exploited applications remain exploited~~
- ~~Significant overhead~~
- ~~Lose integrated desktop feel~~
- ~~Increase management burden~~

Experimental Results

Case Study #1 – Malicious PDF File

- Traditional Desktop
  - Can destroy entire computer
- Always viewed in ephemeral container
  - Attack succeeds
  - Doesn’t affect user

Case Study #2 – Malicious Plugins

- Traditional Computer – Persistent, invisible
- Ephemeral Container
  - Doesn't impact user beyond current ephemeral instance
- Persistent Container – Worse
  - Does damage
  - Can have multiple Persistent Containers for similar programs
    - Similar to Red/Green Isolation
  - Can see if system programs were modified by looking at private layer

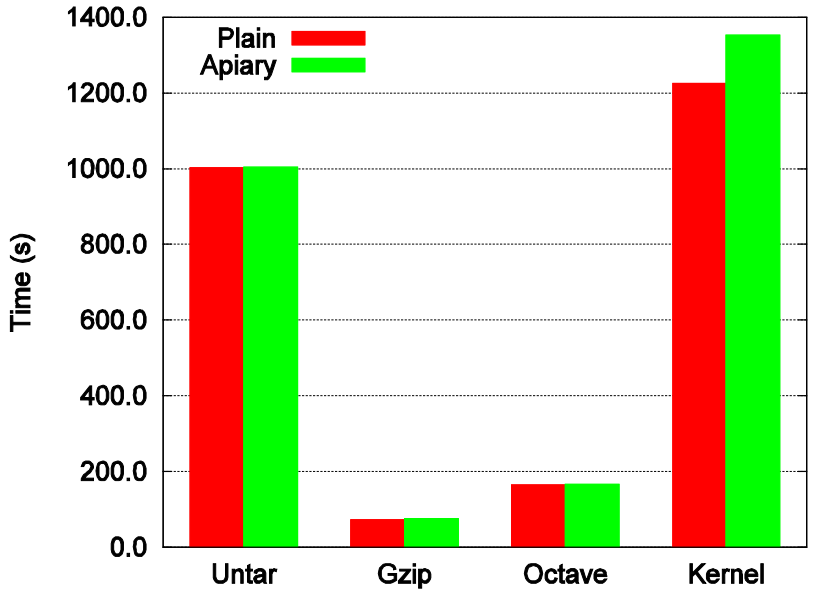
Usage Study

- 24 Users performed tasks including:
  - E-mail
  - IMing
  - Web Browsing
  - Document editing
- Three environments – Plain Linux, No Ephemeral Containers, Ephemeral Containers

Usage Study

- Task completion time was about the same in all containers
- Users didn’t notice overhead of instantiating ephemeral containers
- Users found environment easy to use

Overhead as Containers Scale



- 25 parallel instances/containers running each test
- Overhead generally minimal, even kernel build is only about 10%

Quick Instantiation

	Firefox	T-Bird	OOffice	Xpdf	Mplayer
Apiary	.005s	.005s	.005s	.005s	.005s
Create	276s	294s	365s	291s	294s
Tar Extract	86s	87s	150s	81s	81s
FS-Snap	.016s	.016s	.016s	.016s	.016s

- Why not use an FS with a snapshot/branching semantic (ZFS/Btrfs?)
  - Provisions basically as quick!
  - But, each FS once branched is independent
    - Has to be managed independently!

Efficient Disk Usage

	Firefox	T-Bird	OOffice	Xpdf	Mplayer
Size	353MB	367MB	645MB	339MB	355MB
# Layer	129	125	186	130	162
Shared	330MB	335MB	329MB	330MB	326MB
Unique	23MB	32MB	316MB	9MB	29MB

	Single FS	Multiple FS	VLFSs
Size	**743MB**	2.1GB	**743MB**

Fast File System Updates

	Traditional	VLFS
Time	18s	0.12s

- Time is just for actual file system update
  - For machine maintenance in Apiary, machines can be offline which can add significant time to the traditional updates

Conclusions

- Apiary introduces a new compartmentalized application paradigm
  - Works with existing applications, without changes or recompilation
  - Introduces Ephemeral Containers to prevent compromises from persisting
- VLFS enables simple container management
- Low Overhead and Easy to Use

Questions?

- For more information

http://www.ncl.cs.columbia.edu/

spotter@cs.columbia.edu
spotter@us.ibm.com