



THE LINUX FOUNDATION
OPEN SOURCE SUMMIT

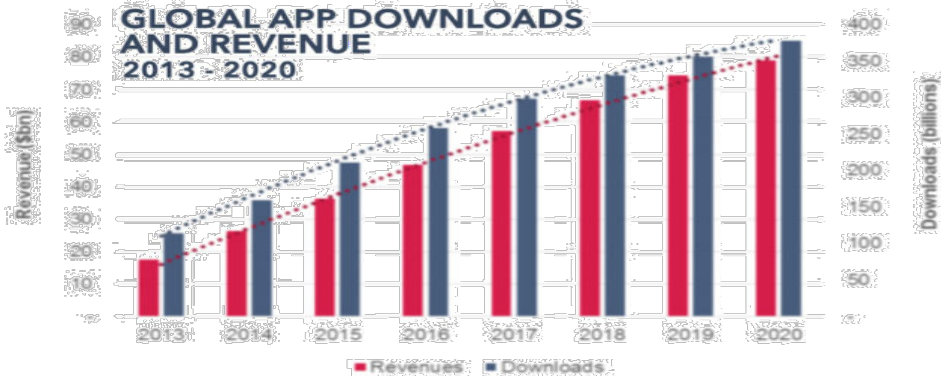
App Streaming Using Android Containers and ARM SOCs

Tzi-cker Chiueh

**Information and Communications Labs
Industry Technology Research Institute**

Everybody Wants to Make APPs

- Smartphone apps provide more **fluent** interaction experiences than browser-based web pages because they are able to access native system resources and advanced smartphone I/O capabilities.



Top 15 Smartphone Apps

January 2016

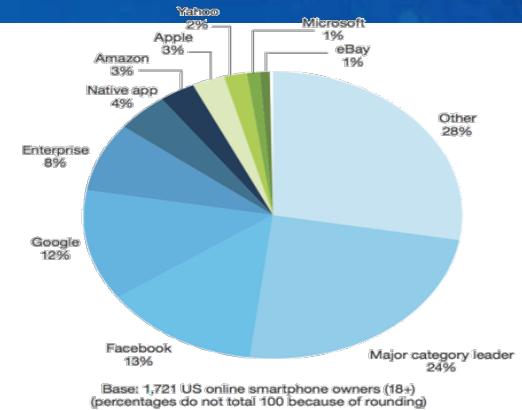
Total U.S. Smartphone Mobile Media Users, Age 18+ (iOS and Android Platforms)

Source: [com Score Mobile Metrix](#)

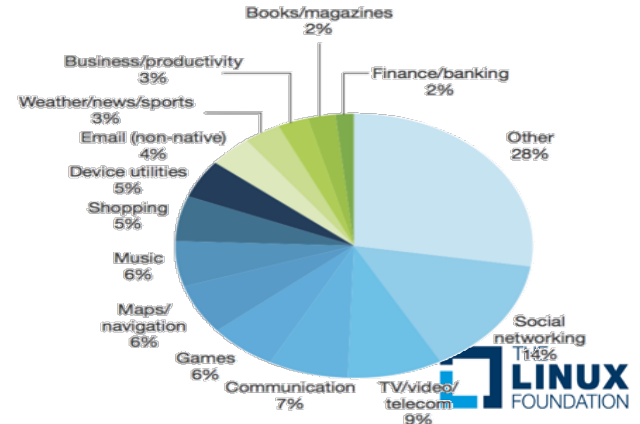
	Top 15 Apps	% Reach
1	Facebook	78.4%
2	Facebook Messenger	64.1%
3	YouTube	61.1%
4	Google Play	51.0%
5	Google Search	50.2%
6	Google Maps	48.7%
7	Gmail	44.8%
8	Pandora Radio	43.3%
9	Instagram	39.0%
10	Amazon Mobile	33.4%
11	Apple Music	31.5%
12	Yahoo Stocks	30.1%
13	Apple Maps	28.1%
14	Google Drive	26.8%
15	Twitter	25.2%

But APP Fatigue Is Apparent

- American mobile user behavior
 - Spent more than 85% time on APPs while using a smartphone, but only **five** APPs are frequently used
 - 80% of time on the first **3** most used APPs: 50% to 1st APP, 18% 2nd APP, 10% 3rd APP
 - More than 70% downloaded APPs retain less than 1 day in the smartphone
- Threat to APP innovation: Convincing a smartphone user to download and continue to use a new APP is increasingly difficult.
- Solutions: Go back to browser or use APP streaming



Source: Forrester's US Consumer Technographics® Behavioral Study, October 2014 to December 2014



APP Testing and Security

- Making an app work on a wide range of platforms is painful and very expensive
- **BYOD security**
 - Recent Russian attack on an US electric grid
 1. Infect contractor's computer
 2. Steal credential
 3. Implant malware on operator's computer



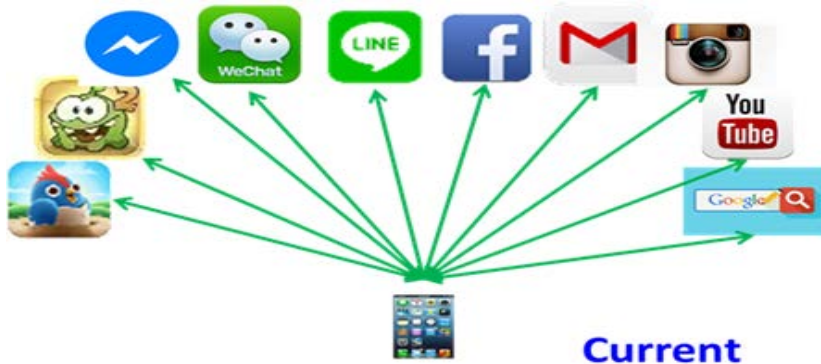
AWS Device Cloud



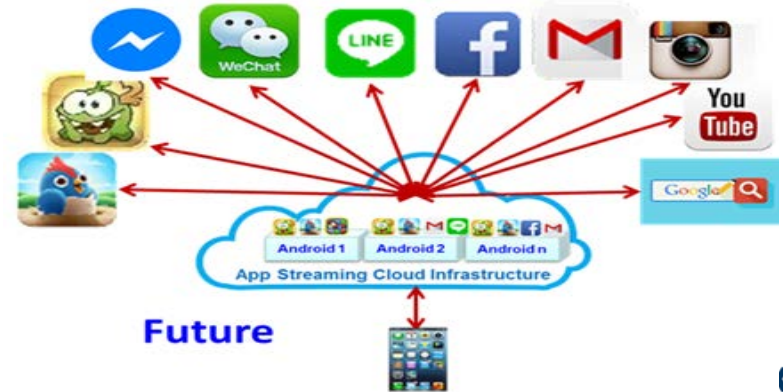
Smartphone APP Streaming

APP Streaming

- Vision: **One APP for all (Android) APPs**
 - APPs run in the cloud, experience all sensors in a user's smartphone, and stream their outputs to the smartphone's audio/video devices.
 - Technical objective: **Attain the same interactivity and usability as local APP execution with unmodified APPs**



Current



Future

Tightened BYOD Security

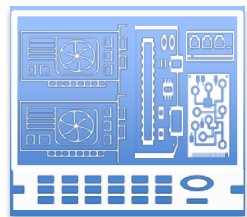


APPs run in
the cloud

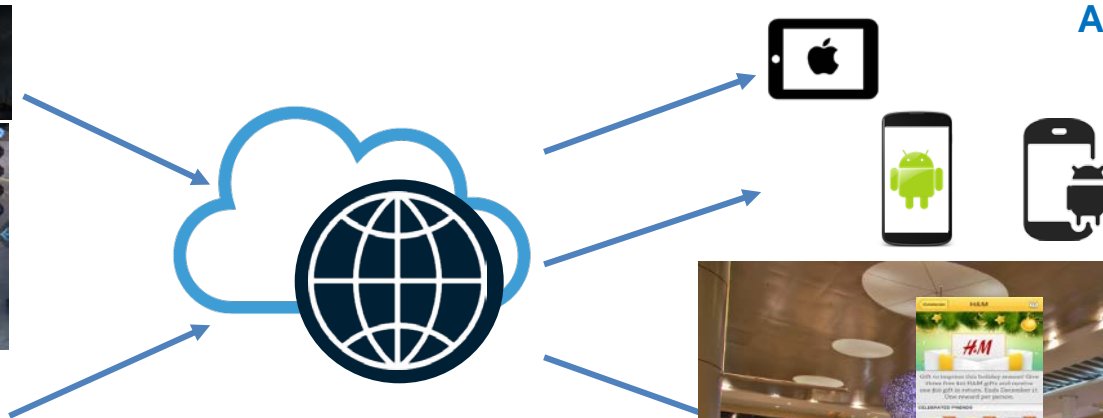
- Centralized control and management of which APPs are allowed in an enterprise
- Reduced data leakage risk even when device is compromised
- Minimal attack surface for malware on compromises device to infiltrate enterprise

Any Android or
iOS device

Run an APP without Downloading it



APPS run on
Android cloud



Any Android or
iOS devices

- Effortless invocation of long-tail APPs
- Trying out a game APP without downloading or installing it
- **A new type of APP market: APP storage and execution**



Case Study 1

Hatch Entertainment in Finland



- A subsidiary of Angry Birds maker Rovio
- Providing streaming access to mobile games in the same way as Netflix does for movies or Spotify for music
- The beta version of Hatch's cloud gaming app is already available in 16 countries
- Infrastructure strategy
 - Run Android software stack in containers on Qualcomm Centriq 2400, a 48-core ARM processor
 - Provide a large number of game instances per server and keep per-instance server cost down



Case Study 2

- Redfinger in China

REDFINGER

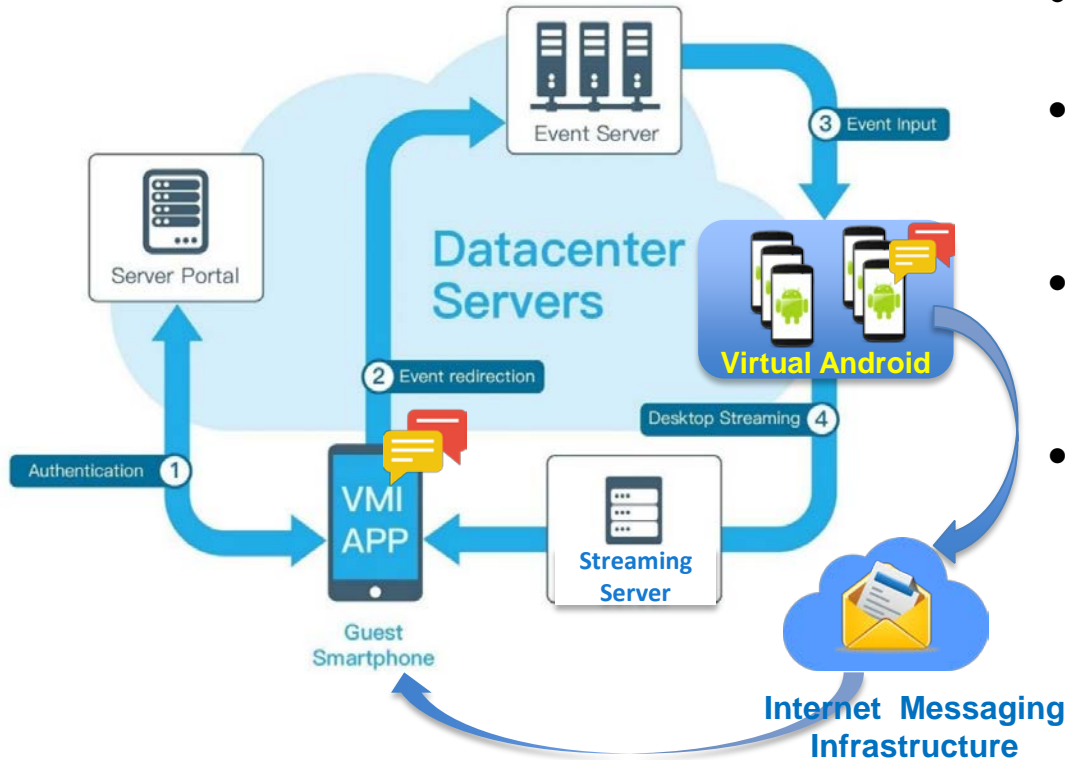
- A Baidu-invested startup
- Provide cloud virtual smartphone services
 - Cloud gaming (bot) - online 24/7
- 5 million subscribers in 2016
- Service Charge
 - \$9.95 USD/per month for 8 GB Storage, 4 GB RAM
- Infrastructure strategy
 - Hyperspace Cloud Android Smartphone platform: running Android emulators in the cloud



Cloud Gaming - Online 24/7



How App Streaming Works



- Bi-directional interactions between device and cloud
- The **Server Portal** that handles user authentication, mediation and virtual machine deployment
- Through **Event Server**, touch and other sensor data event are streamed up from client device
- Display and audio are sent via **Streaming Server** from cloud-based Android instance to client
 - Relay of intent & notification from cloud to device

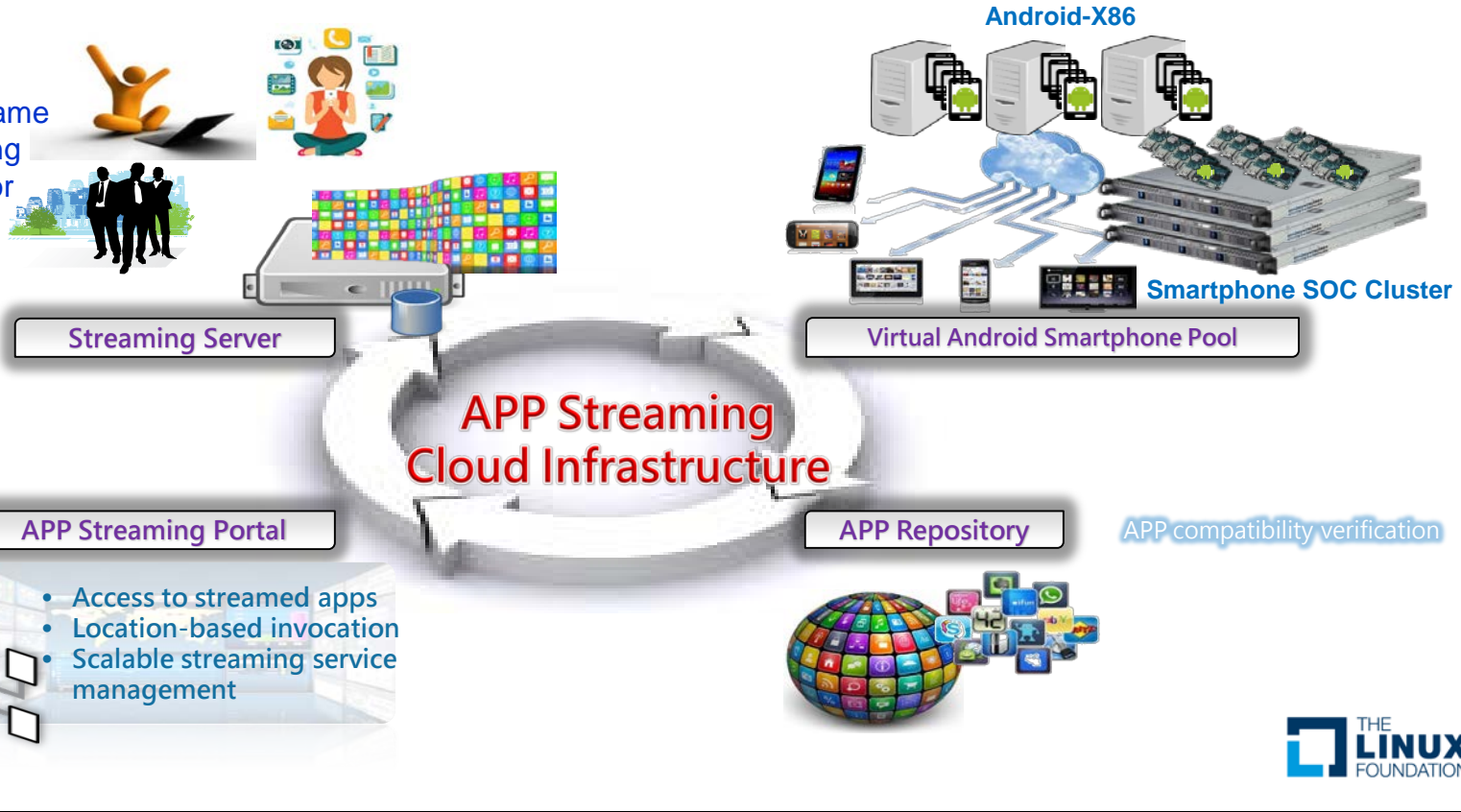
Requirements for App Streaming Infrastructure

- Virtualization of Android (framework + Linux)
 - Hypervisor-based virtual machine
 - OS-based container
 - Function as a service, e.g., Lambda
- Thin client on smartphone
 - Input: frame buffer, audio, intent and notification
 - Output: sensor data and touch events
- **App binary compatibility**
- Low-latency streaming



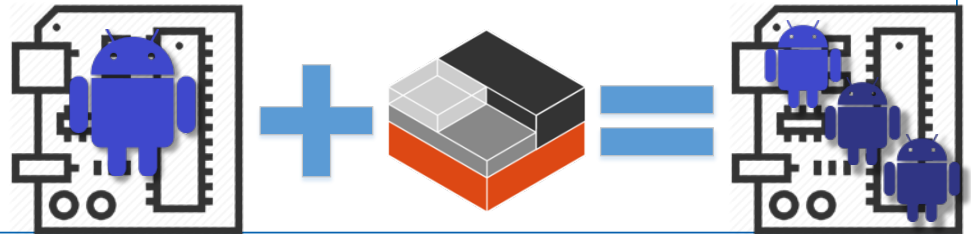
ITRI's Cloud-based APP Streaming Service

- Low-latency frame buffer streaming
- Physical sensor redirection



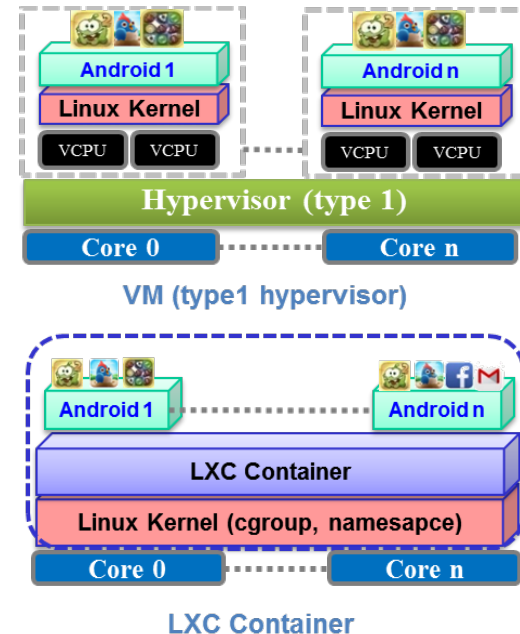
Android Containerization

- When LXC Meets Android



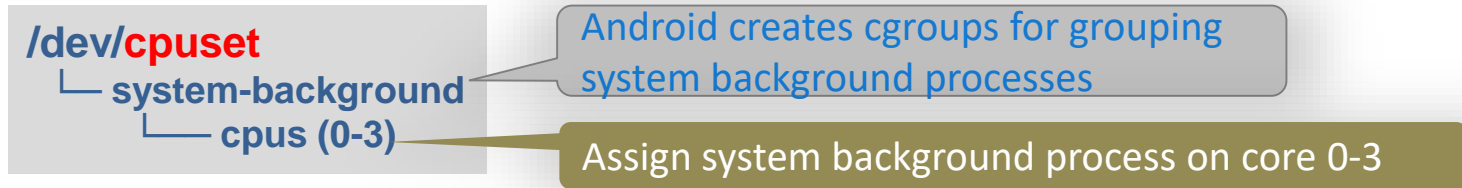
Linux Container (LXC)

- An operating system-level virtualization method for running multiple isolated Linux instances (containers) on top of a single Linux kernel
 - **Namespace**: objects
 - **Cgroup**: resource usage
 - **Union filesystem**: file sharing between host and containers
- Containers are more lightweight compared with traditional VMs: 6 to 8 times as many containers as VMs on the same hardware



Cgroup and Namespace

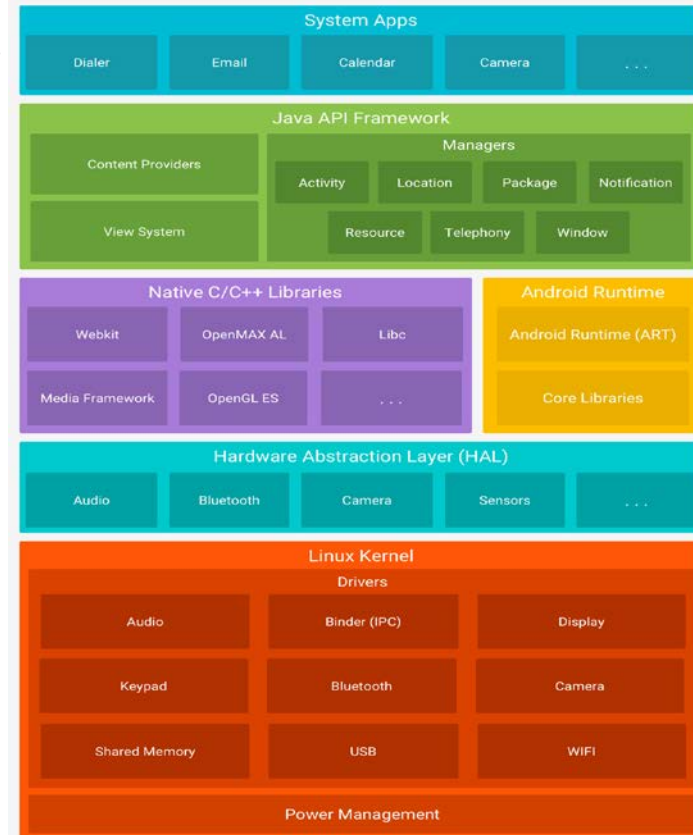
- **cgroup**: enables processes to form groups so as to limit their visibility and resource usage
 - Provides inter-container isolation in terms of resource usage



- **namespace**: limits the resources that a group can see
 - Provides inter-container isolation in terms of resource visibility/accessibility
 - LXC creates a set of namespaces for a container before it starts

Layered Architecture of Android

- A modified version of the Linux kernel as the base operating system
- Android run-time environment and libraries
 - Android Runtime (ART) and Dalvik virtual machine
- Java API Framework
 - Provides the basic functions of Android device
- Hardware Abstraction Layer (HAL)
 - Provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework.
- Mobile key applications
 - Dialer, contact, browser, ...

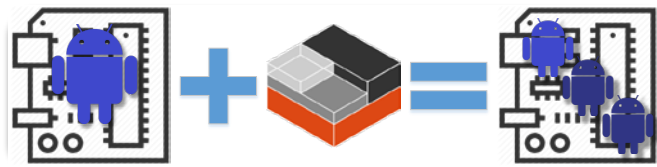


LXC Meets Android

- Fundamental issue: one vs. multiple Android instances

- Resource access and usage

- Android's resource usage assignment for process groups formed with cgroup
- Android's use of SELinux
- Android's permission settings for procs (/proc) and sysfs(/sys)



- Device virtualization

- Binding of virtual and **local/remote** physical peripheral devices
- Fair sharing of local physical peripheral devices among host and containers



SELinux & Android

- Security-Enhanced Linux
 - A Linux kernel security module supporting mandatory access controls (MAC)
- Android uses SELinux to enforce MAC over all processes
 - Android defines complicated rules/security contexts to enforce mandatory access control over all processes

```
# file_contexts
```

```
/system/bin/surfaceflinger u:object_r:surfaceflinger_exec:s0
```

```
/dev/mali[0-9] u:object_r:gpu_device:s0
```

labeling /system/bin/surfaceflinger

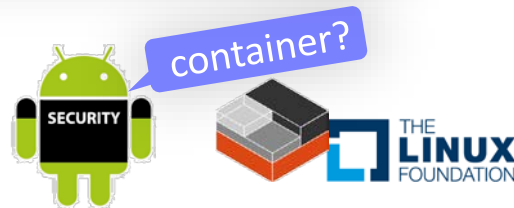
labeling gpu resources

```
# surfaceflinger.te
```

```
allow surfaceflinger gpu_device:chr_file { ioctl };
```

allow type:surfaceflinger to access gpu resource

- **TODO: Container-aware SELinux**
 - Allows each container to have its own security contexts and rules



Access Permission Setting

- Symptom
 - Kernel sysfs and procfs are “shared” between host and containers
 - When an Android container shuts down, “/proc/sysrq-trigger” is touched, resulting in all mount points becoming read-only
- Root cause: Default LXC permission setting is inadequate

LXC access control rules	/proc	/proc/sys	/proc/sysrq-trigger	note
proc:mixed	r/w	ro	ro	lxc essential
proc:rw	r/w	r/w	r/w	lxc essential
proc:android	r/w	r/w	ro	Android specific

- A patch to LXC
 - Add “proc:android” control rule for Android
- TODO
 - Need to more thoroughly examine Android’s accesses to kernel sysfs and procfs

Android's CPU Resource Assignment

- CPU share assignment after Android 5.1.1

cgroup	system service process	purpose
bg_non_interactive	system_server com.android.systemui	To keep UI smoothness at a certain satisfaction level even when system is busy, system assigns least 5% of CPU resources to this cgroup

- CPU core binding after Android 7.1.2 (for BIG * 2 + LITTLE * 4)
 - Associate specific CPU cores with a specific cgroup

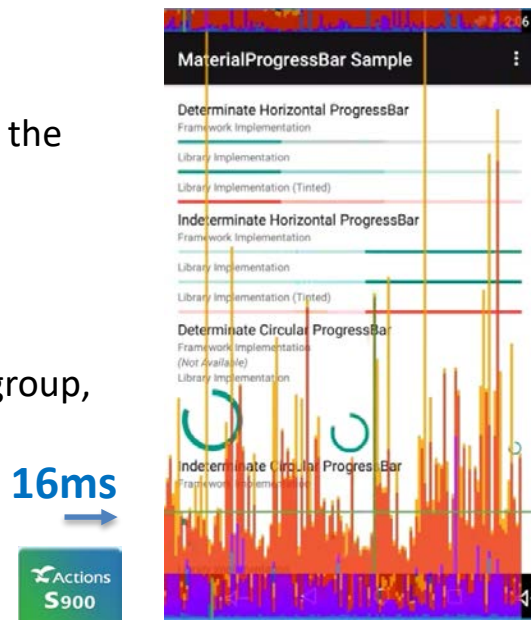
top-app (cpus 0-5)	foreground (cpus 0-5)	system-background (cpus 0-3)	background
Current active App	system_server com.android.systemui	surfaceflinger servicemanager	Non-focused Apps

Insufficient CPU Share for UI Services

- Purpose of **cpu.shares**
 - Allows assigning a percentage of CPU time to a cgroup
- Android 5.1.1 usage
 - Creating **bg_non_interactive** group for grouping system services
 - In order to maintain frame rate at a specific level, Android assigns 5% of the CPU resource to the **bg_non_interactive** group
- Problem with multiple Android containers
 - As the # of containers increases, App UI update freq. becomes lower
- Root cause
 - Containers also assign their system services to the **bg_non_interactive** group, whose CPU share remains at 5%.
 - Each container's cpu share for bg UI services is < 5%.

system service process in bg_non_interactive	cpu.shares
system_server, com.android.systemui	5%
system_server, com.android.systemui system_server, com.android.systemui,	5%

UI Update (ms/per frame) 2 containers



16ms



Cortex-A53 * 4
RAM: 3G

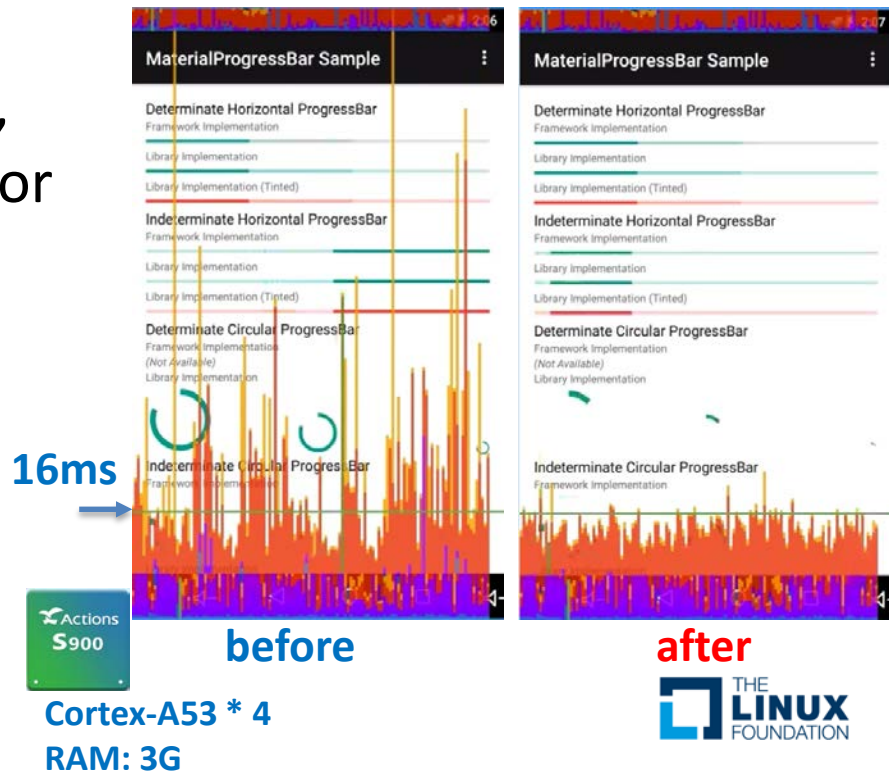


Enhancement to CPU Share Assignment

- Solution
 - As the # of containers increases, linearly increase the cpu share for bg_non_interactive
 - $5\% * (1 + \# \text{ of containers})$

	system service process	cpu.shares
Host only	system_server, com.android.systemui	5%
+ 1 container	system_server, com.android.systemui system_server, com.android.systemui	10%
+ 2 containers	system_server, com.android.systemui system_server, com.android.systemui system_server, com.android.systemui	15%

UI Update (ms/per frame), 2 containers



Smaller No. of CPU Cores Than Needed

- Purpose of **cpuset**
 - Assigns individual CPU cores to a specific cgroup
- Android 7.1.2 usage on Rockchip RK3399
 - Essential Android cgroups
 - top-app, foreground, system-background, background



Cortex-A72 * 2
Cortex-A53 * 4

top-app (cpus 0-5)	foreground (cpus 0-5)	system-background (cpus 0-3)	background
Current active App	system_server com.android.systemui	surfaceflinger servicemanager	Non-focused Apps

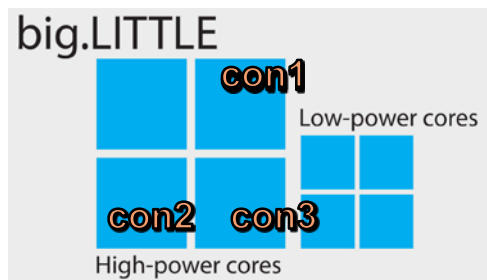
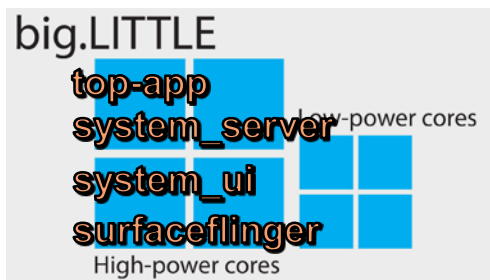
- Symptom
 - As # of containers increases, LITTLE cores are very busy, but BIG cores are not.



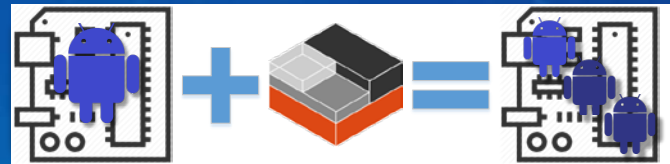
Enhancement to CPU Core Assignment



- Root cause
 - Android's cpuset usage and scheduler are designed for a single Android instance
- When multiple Android instances run on a machine
 - Allow system service processes to use BIG cores, or
 - Assign each container a big core for its system service and normal processes



Summary



item	(kernel) function	lxc usage	Android usage	conflict
cgroup cpu.shares	Specifying a relative share of CPU time available to the tasks in a cgroup	Grouping by different containers	Grouping in Android defined cgroups	yes
cgroup cpuset	Assigning individual CPUs and memory nodes to cgroups	Grouping by different containers	Grouping in Android defined cgroups	yes
SELinux	Supporting access control security policies for mandatory access controls (MAC).	<ul style="list-style-type: none">• SELinux container-awareness is missing• Android defines complicated rules/security context to enforce mandatory access control over all processes• Currently, each container share the same security rules		
Kernel procs (/proc) and sysfs (/sys)	A pseudo file system provided by the kernel that exports information about various kernel subsystems or process state	lxc defines 2 types access control <ul style="list-style-type: none">• mixed• rw	Current lxc access control is not enough for multi-Android-instance usage	yes

Overhead of Android Containerization

- Platform: Google Nexus 7
 - Qualcomm® Snapdragon™ S4 Pro 8064 Quad-Core, 1.5 GHz



Android 6.x.x
GPU: Adreno 320
RAM: 2G

- Minimal container interception overhead
 - GPU Benchmark 3D on Google Nexus 7



	on host only	on container only	host + container (simultaneously)	
			Host	Container
Score	11227	11024	5976	5356
FPS (avg)	41.24	40.68	21.17	19.39

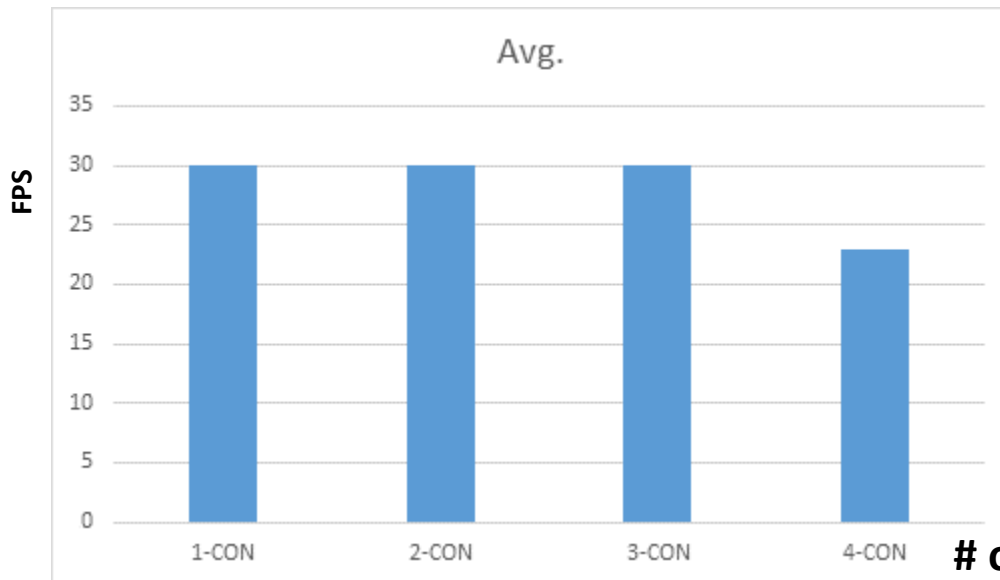
- 3DRating for OpenGL ES 2.0



	on host only	on container only	Host + Container (simultaneously)	
			Host	Container
Score	2560	2527	1588	1579

Scalability of Android Containerization

- Platform: Actions S900 (ARM SOC)
- Workload – Google YouTube



Android 5.1.1

CPU: Cortex-A53 * 4 @ 1.8G

GPU: Imagination Power VR G6230

RAM: 3G

Scalability of Android Containerization

- Platform: Intel® NUC
 - Intel Core i7-6770HQ processor
- Workload – Seascape, an ocean renderer using OpenGL ES 2.0 and specific GLSL shader features

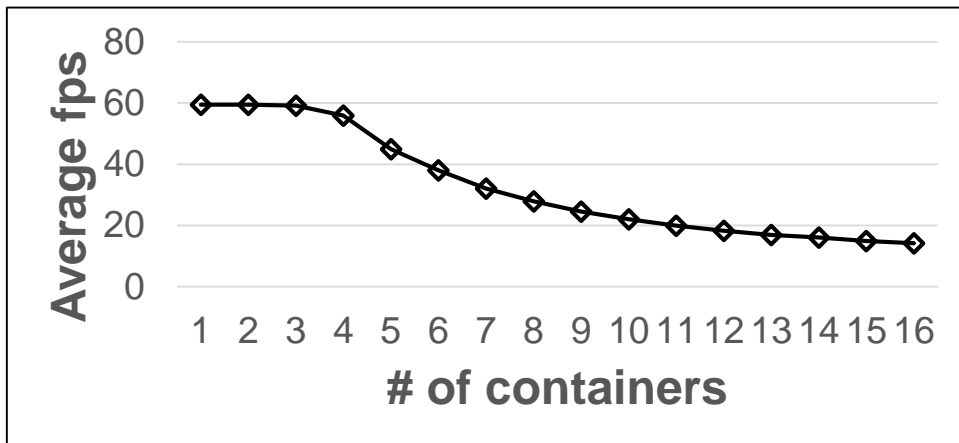


Android 7.1.2

CPU: QUAD CORE @ 2.6 ~ 3.5G

GPU: Intel Iris Pro Graphics 580

RAM: 32G



Containerization vs. Virtualization

- Android x86 on KVM + Qemu
- Workload – Seascape, an ocean render using OpenGL ES 2.0 and specific GLSL shader features

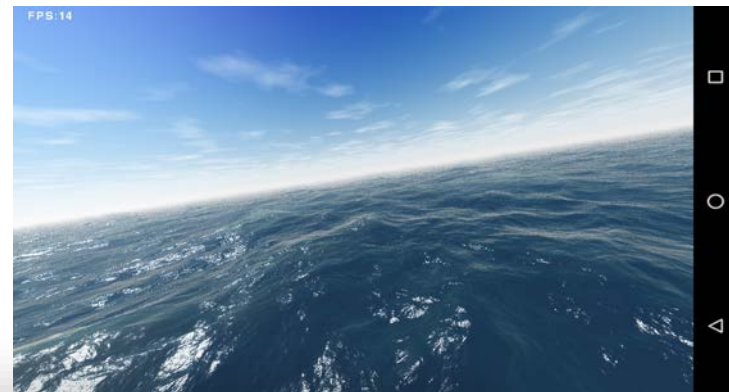
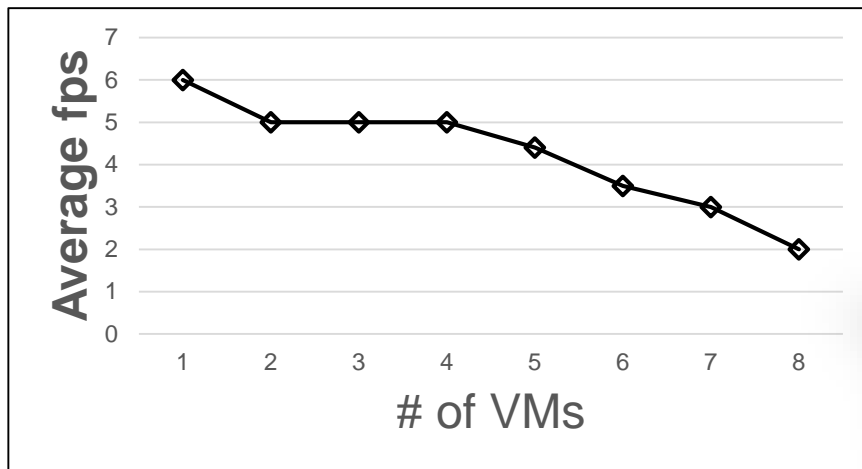


Android 7.1.2

CPU: QUAD CORE @ 2.6 ~ 3.5G

GPU: Intel Iris Pro Graphics 580

RAM: 32G

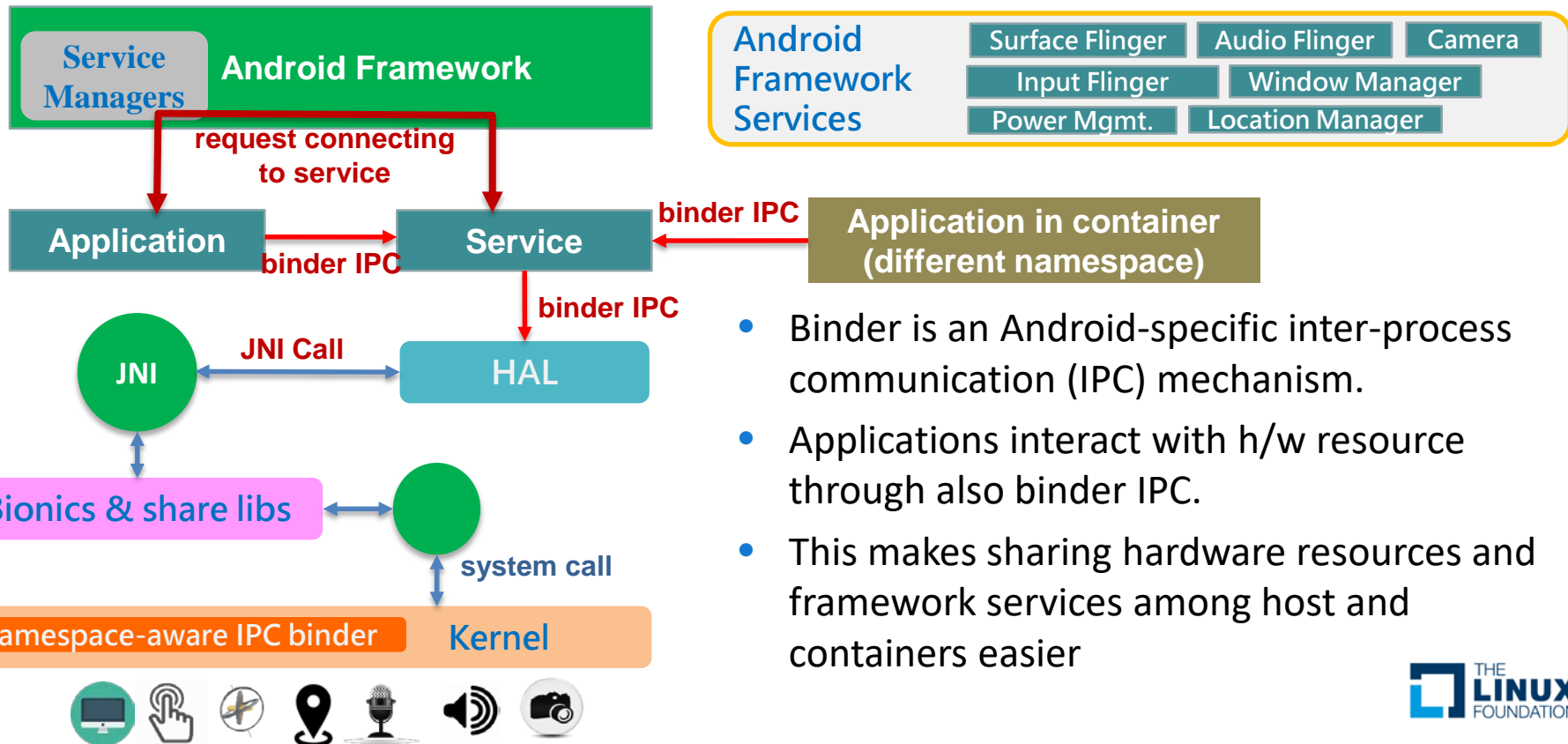


Pure software rendering,
as Android x86 on KVM is
not able to access GPU



Device Virtualization

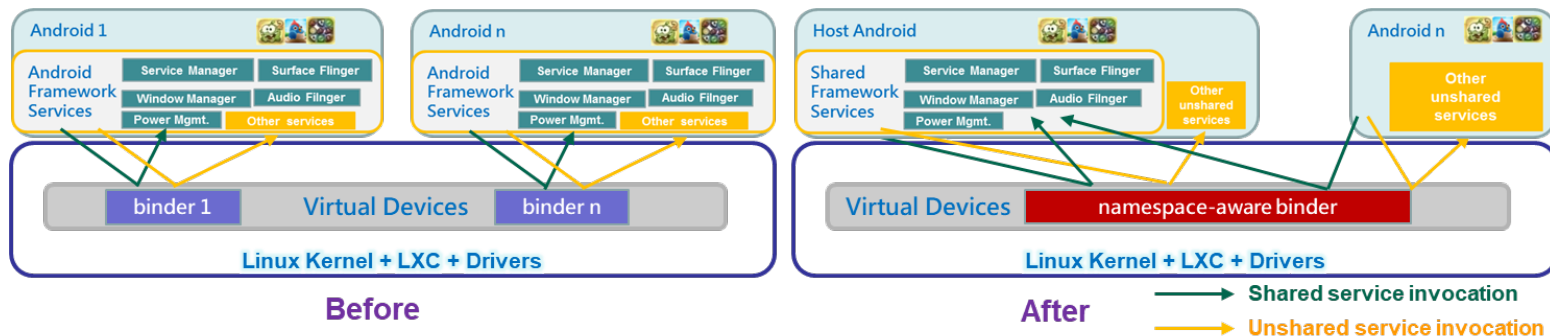
Android's Device Control Model



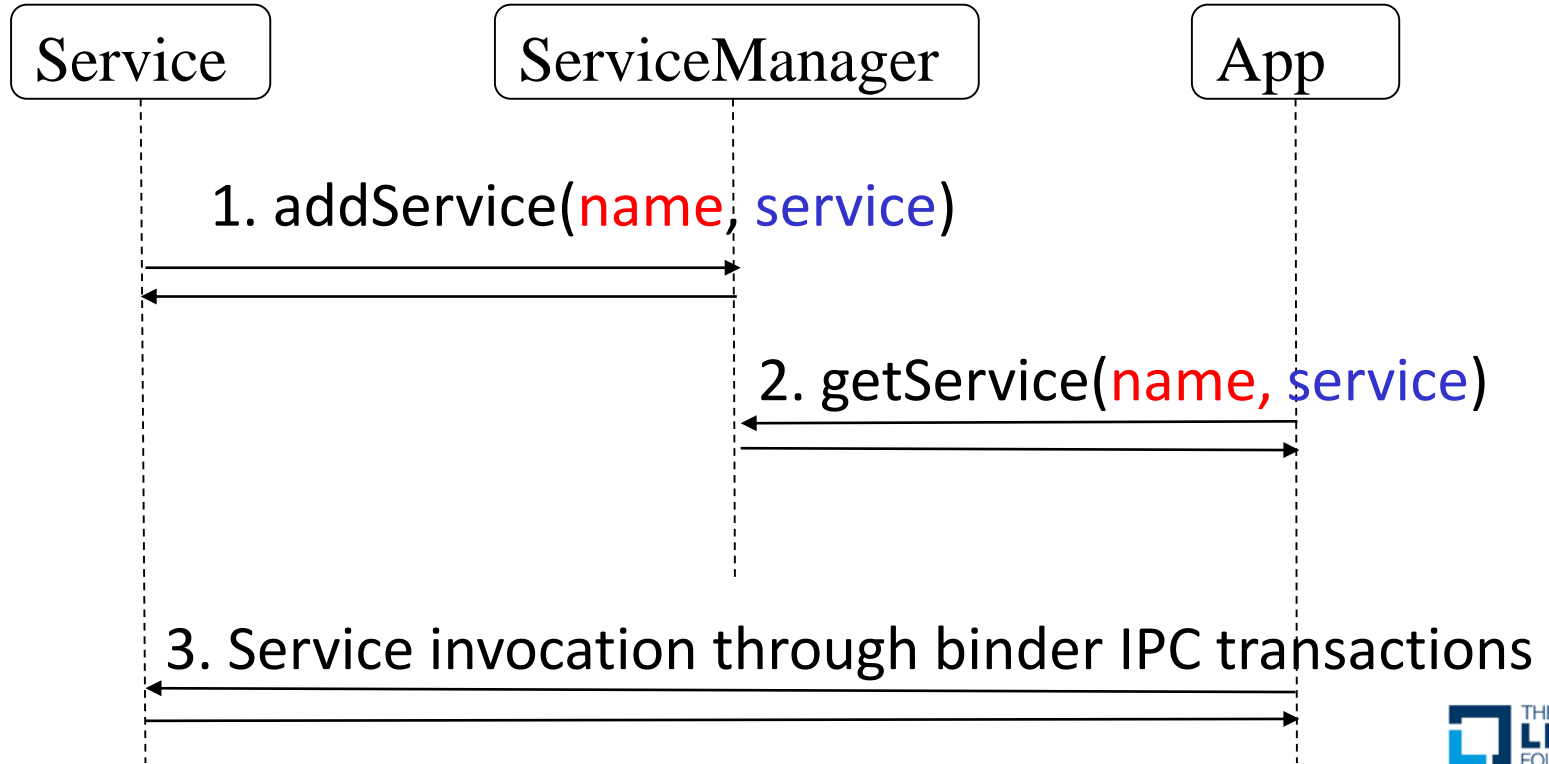
- Binder is an Android-specific inter-process communication (IPC) mechanism.
- Applications interact with h/w resource through also binder IPC.
- This makes sharing hardware resources and framework services among host and containers easier

Selective Sharing of System Services

- How to selectively share system services among Android instances
 - Namespace-aware IPC binder vs. Per-container binder
 - Inspired by the Container Virtualization Adapted to Android project at the Architecture Laboratory of Zhejiang University
 - Which system services are sharable is configurable.
 - Caller is aware of the container ID of callee.

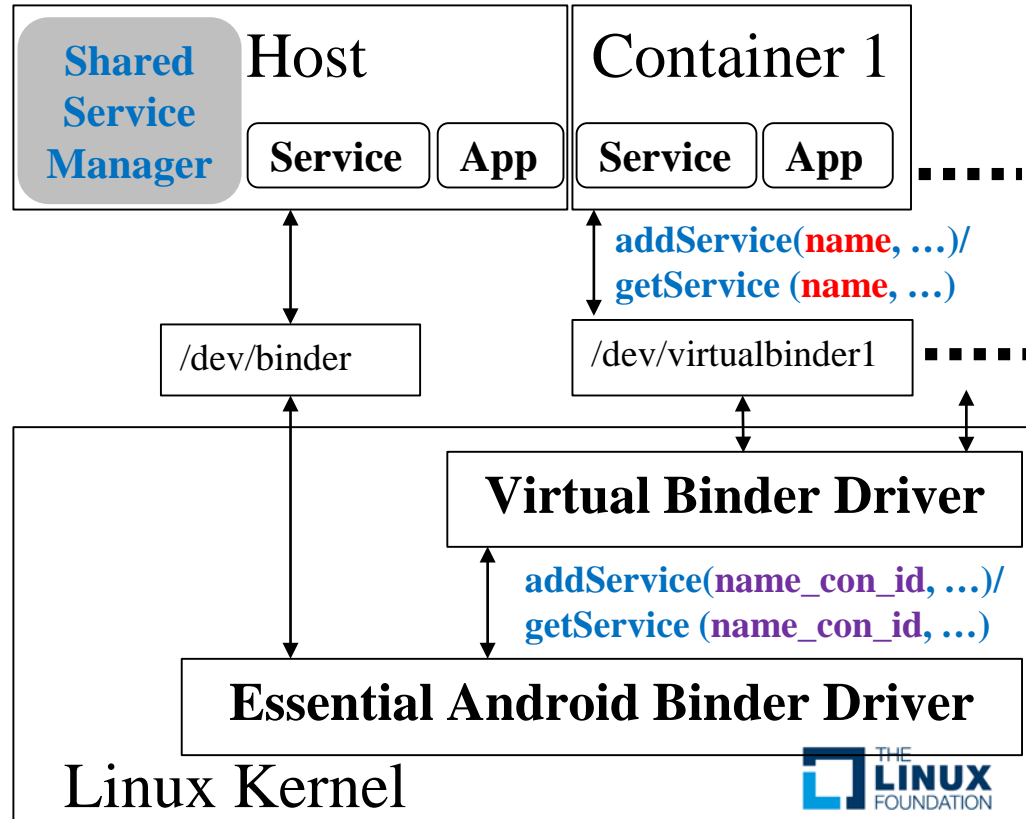


Android's Service Registration/Invocation

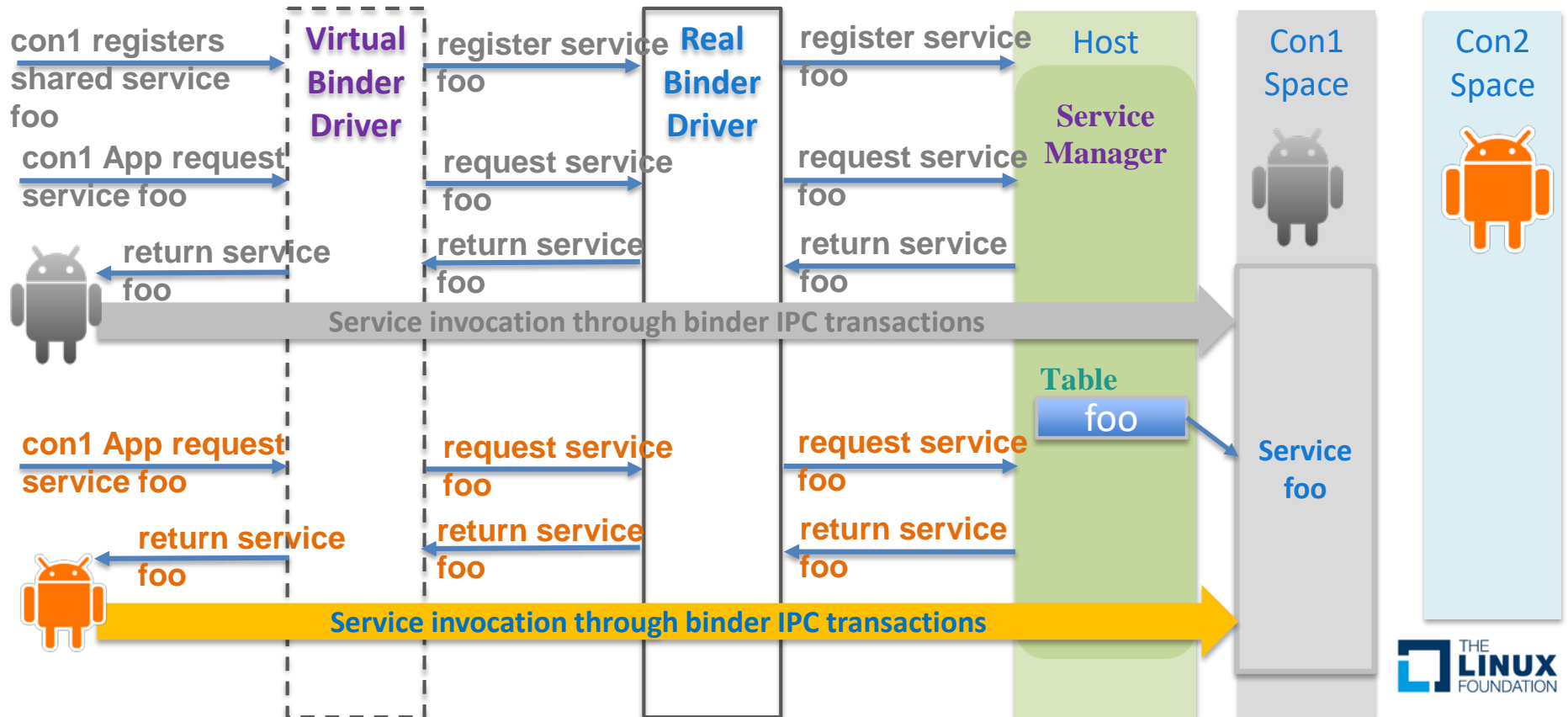


Virtual Binder Architecture

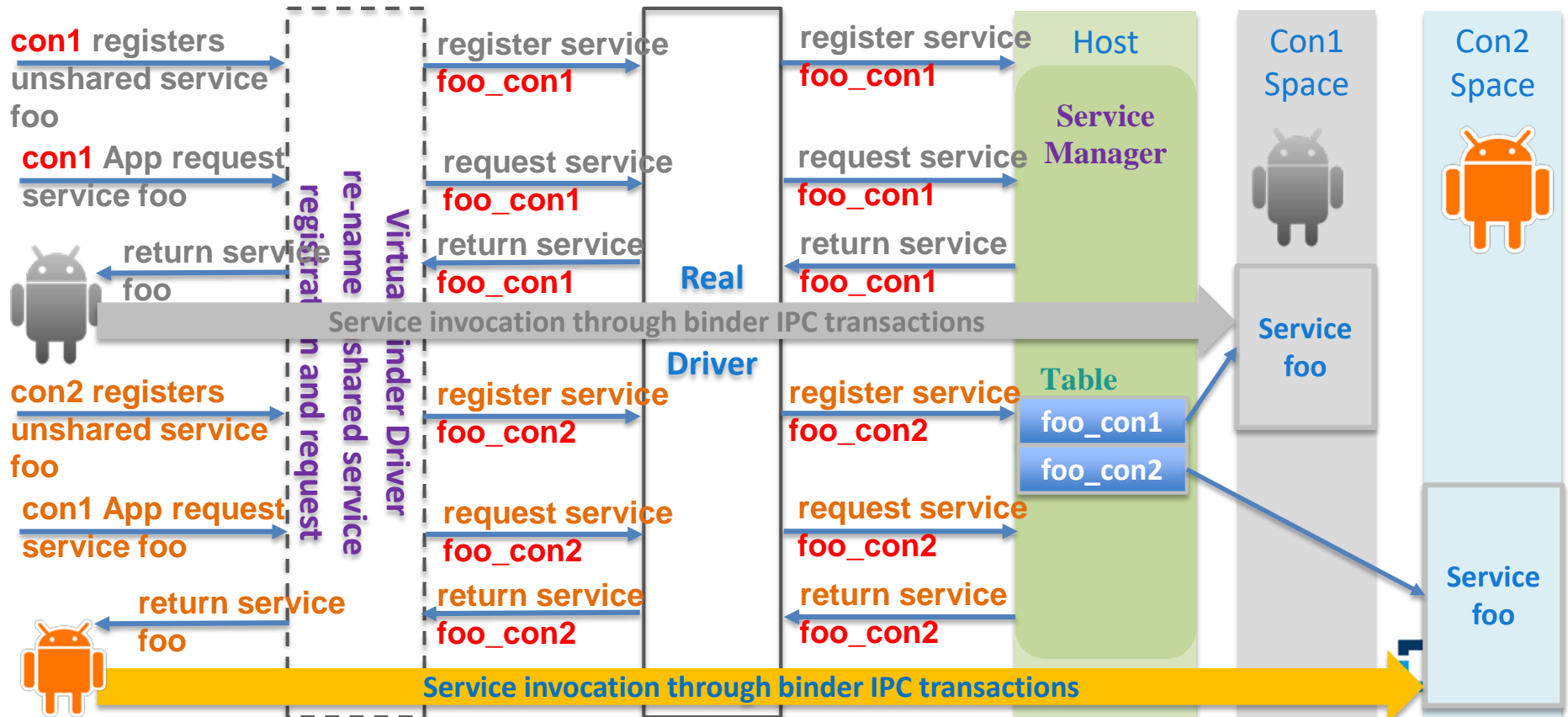
- Virtual binder (driver) as the wrapper of essential Android binder
- It bridges service registration and query from containers
- It re-names the service name for only un-shared services



Working of Virtual Binder – Shared Service

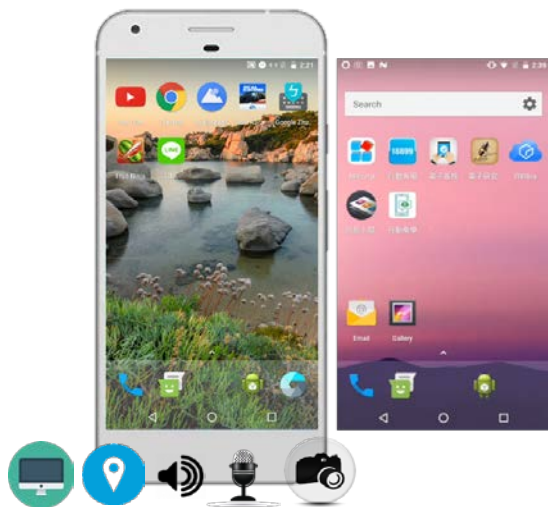


Working of Virtual Binder – Unshared Service



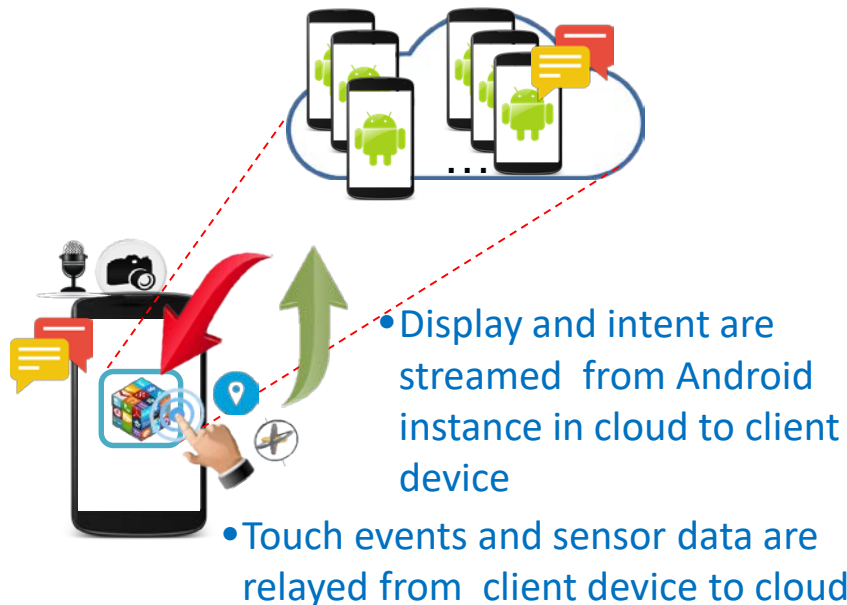
Two Use Cases of Android Containerization

Smartphone Virtualization



- Local h/w peripheral on host and shared among host and containers
- Essential mobile user experience: One active container at a time

Virtual Smartphone in the Cloud



- Display and intent are streamed from Android instance in cloud to client device
- Touch events and sensor data are relayed from client device to cloud

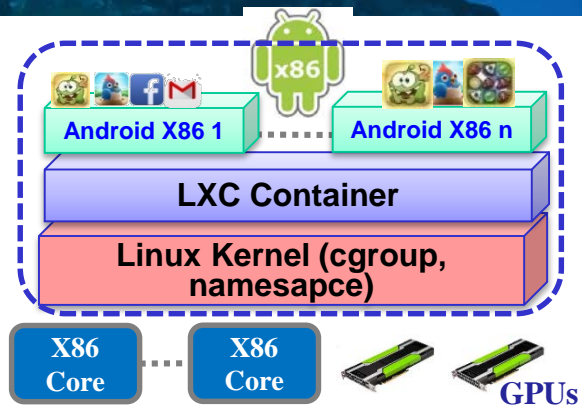
HW Platforms for Android Containers



- **Smartphone**

- Android
- Google Pixel 2

- BYOD security application

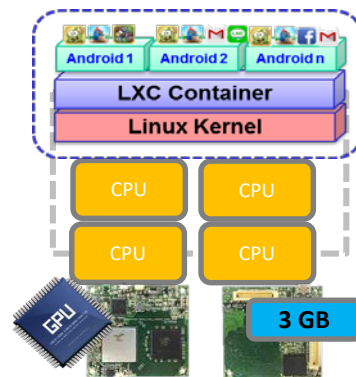


- **X86 Server Machine**

- Android X86
- Powerful GPU

- Run smartphone Apps with ARM native code result-in

- Binary translation overhead
- Compatibility issues



- **Smartphone SOC Cluster**





- Run smartphone Apps natively (Native ARM ISA)
- Dedicated GPU, Audio/Video h/w Encoder/Decoder

- Perfect match for app streaming workload



Comparison

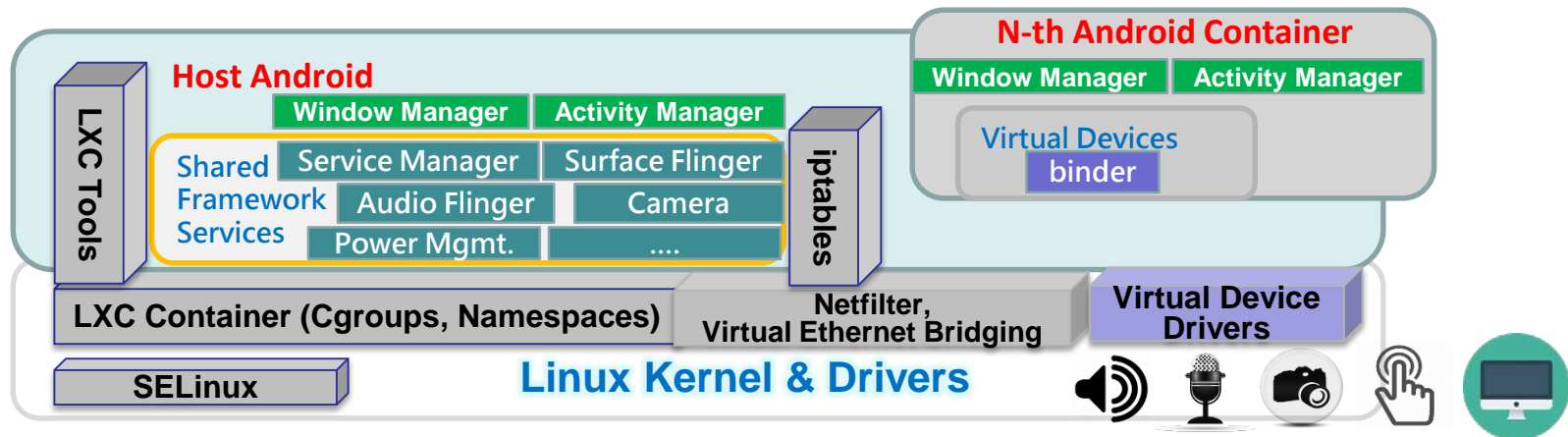
Sharing of peripheral devices is mostly achieved by sharing controlling system services

Resource	Framework Services	Smartphone	Cloud Deployment
	Window Manager	isolated	isolated, individual virtual display for each container
	Surface Flinger	shared	
	Input Flinger (Input Reader)	isolated, only active container consumes input	isolated, injecting touch and other sensor data streamed from user device
	Audio Flinger	shared	isolated, providing fake audio device for each container, which streams voice over the Internet
	Camera Service ("media.camera")	shared	isolated, redirect from client to cloud
Intent/ Notification	Intent/Notification service	isolated	Isolated, private channel between client and cloud Android through Internet cloud messaging services



Android Containerization for Smartphone Virtualization

Containerizing Android for Phone

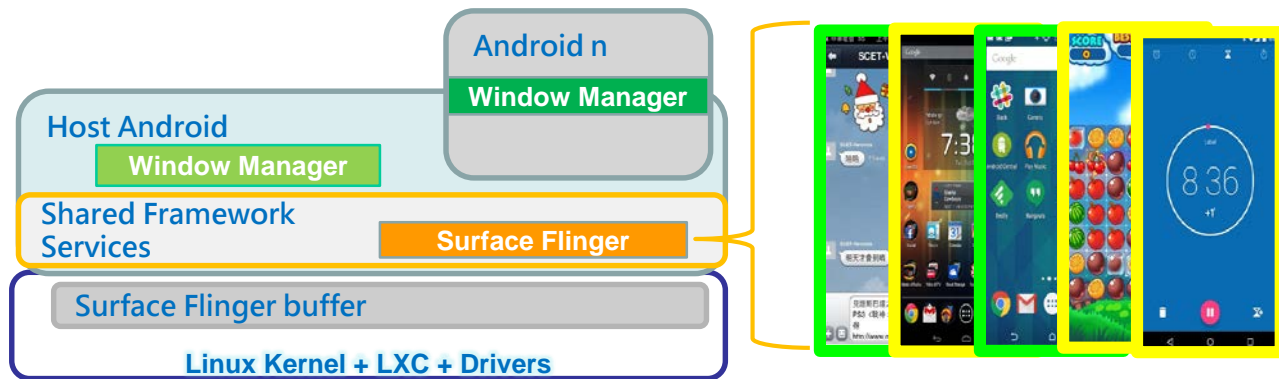


- Android as the host OS
- Peripherals and framework services are shared among host and Containers
 - Primary display is shared between and host and containers, switching between host and active container

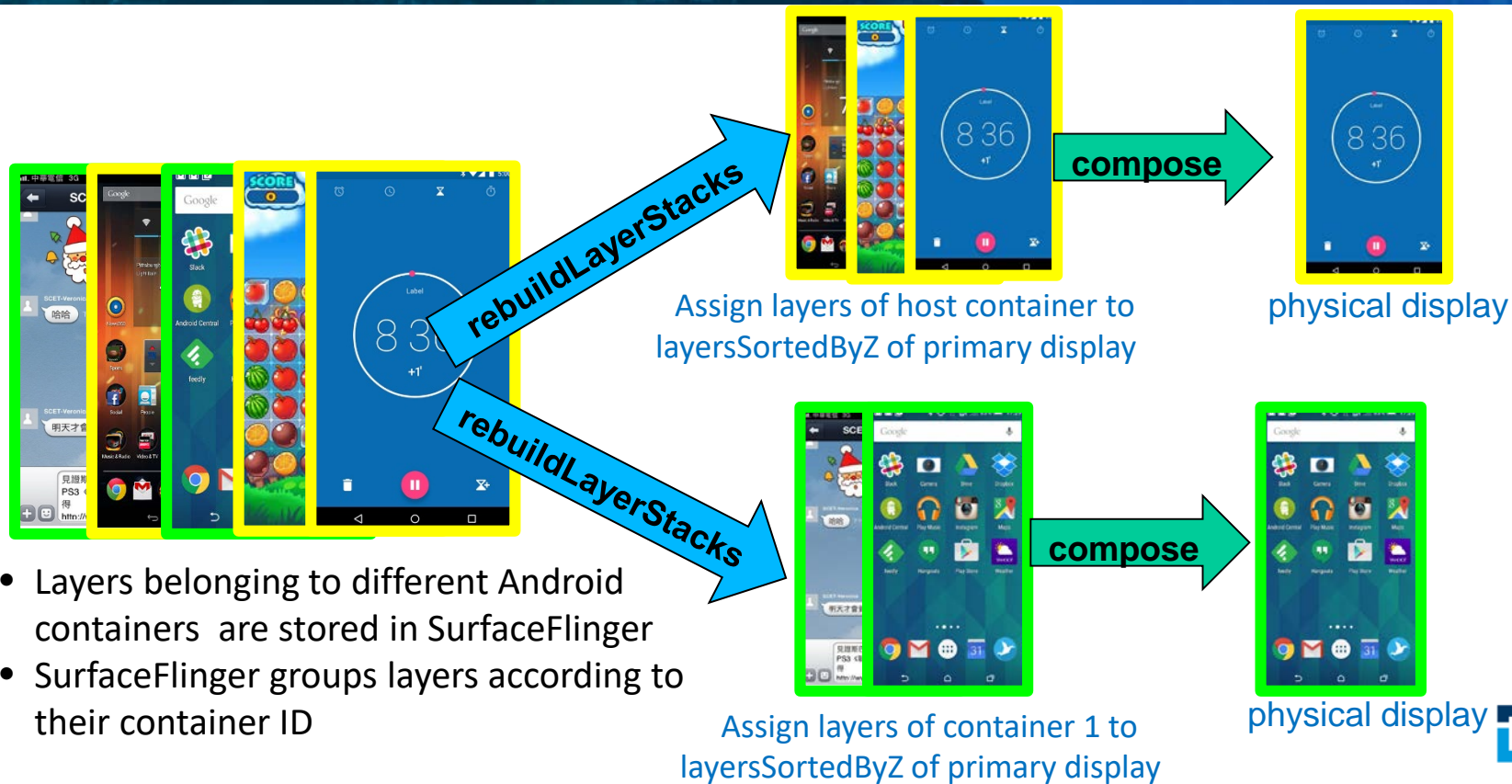
Shared SurfaceFlinger



- Sharing of SurfaceFlinger decreases resource usage
- Each container has its own window manager
- A single SurfaceFlinger instance runs on the host
 - Maintains a separate list of layers for each Android container by grouping layers according to their container ID



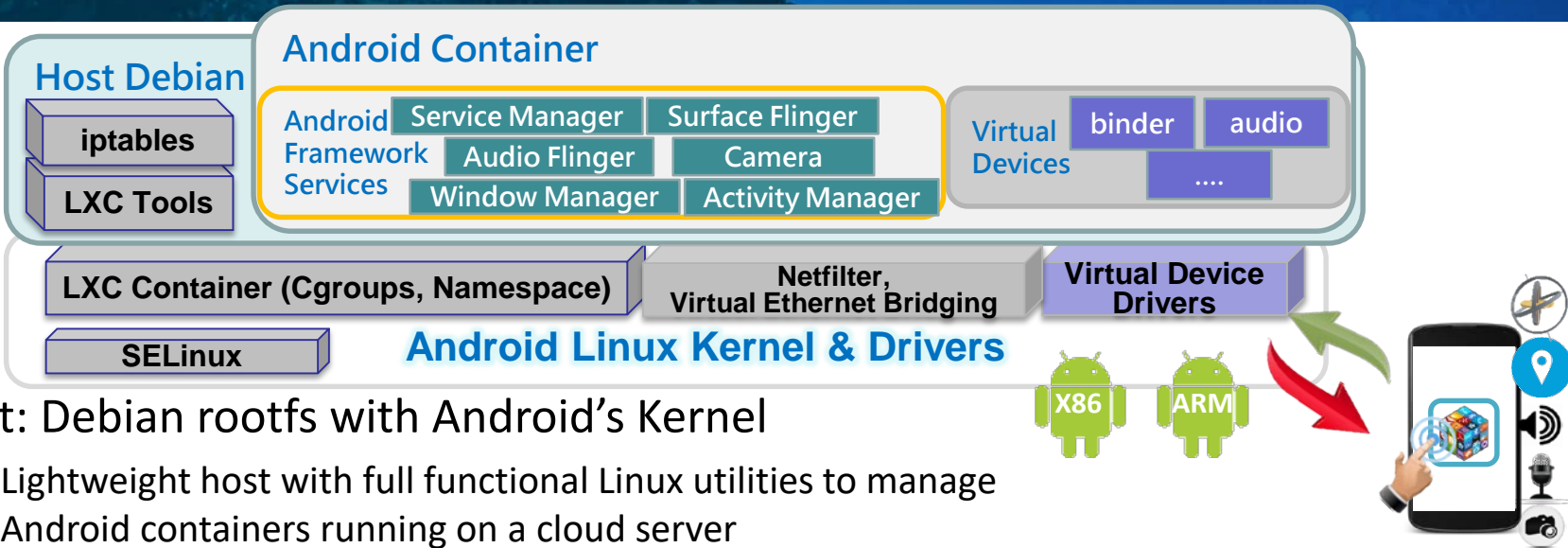
Switching Display between Containers



- Layers belonging to different Android containers are stored in SurfaceFlinger
- SurfaceFlinger groups layers according to their container ID

Android Containerization for Virtual Smartphone in the Cloud

Containerizing Android for Cloud Server



- Host: Debian rootfs with Android's Kernel
 - Lightweight host with full functional Linux utilities to manage Android containers running on a cloud server
- Each Android container has its own Android framework and services
- Configurable virtual display for each container and its remote client
- Virtual device remoting (sensors, audio, camera, location)

ITRI ARM SOC Cluster

Carrier Board*5, 7 SOM per CB
(4 on top side, 3 on back)

ARM node*35
(Actions S900 SOM)

LED Indicator light*2
power & network

LAN Switch
(1G*8 + 10G*2)

PSU
500W 1U

BMC
for SOM management

Backplane Board
(power supply, Ethernet port, & MCU)

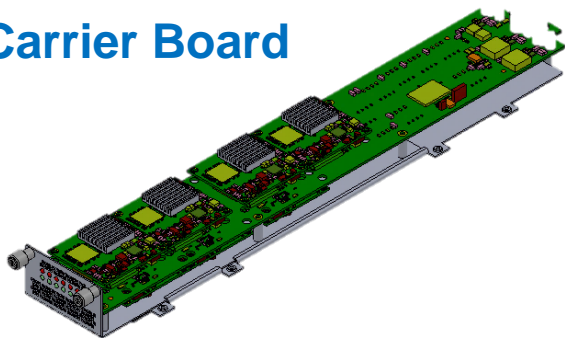


- 1U chassis system
- Total power consumption is **< 160W**



Carrier Board and System on a Module

Carrier Board



- 7 node per carrier board (4 on top side, 3 on back)
- 1G USB-Ethernet RT8153 * 7
- 8-port 1GbE switch RTL8370N
- Connector for SOMs; Goldfingers to Backplane board

SOM



- Actions S900
 - Quad-core 64-bit Cortex-A53 @1.8GHz
- Video Encode
 - H.264 baseline profile, up to 1080p@60fps
- 3G LPDDR3; 8GB eMMC
- GPU: Imagination Power VR G6230
- USB 3.0 for external 1GbE
- Connect to Carrier Board

Networking in ITRI ARM SOC Cluster

LAN Switch
(1G*8 + 10G*2)

BMC

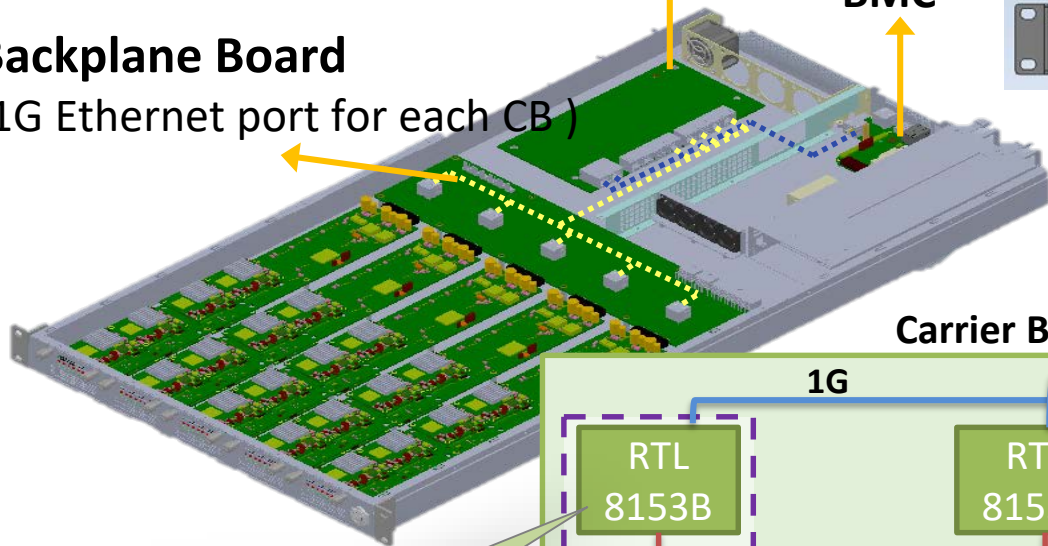
REAR VIEW



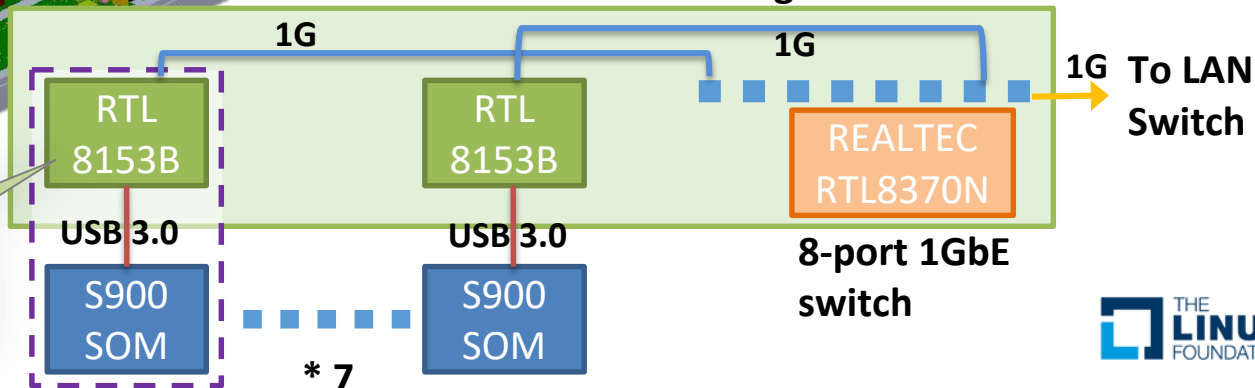
External net 10G * 2

1G * 1 for BMC

Backplane Board
(1G Ethernet port for each CB)



Carrier Board Block Diagram

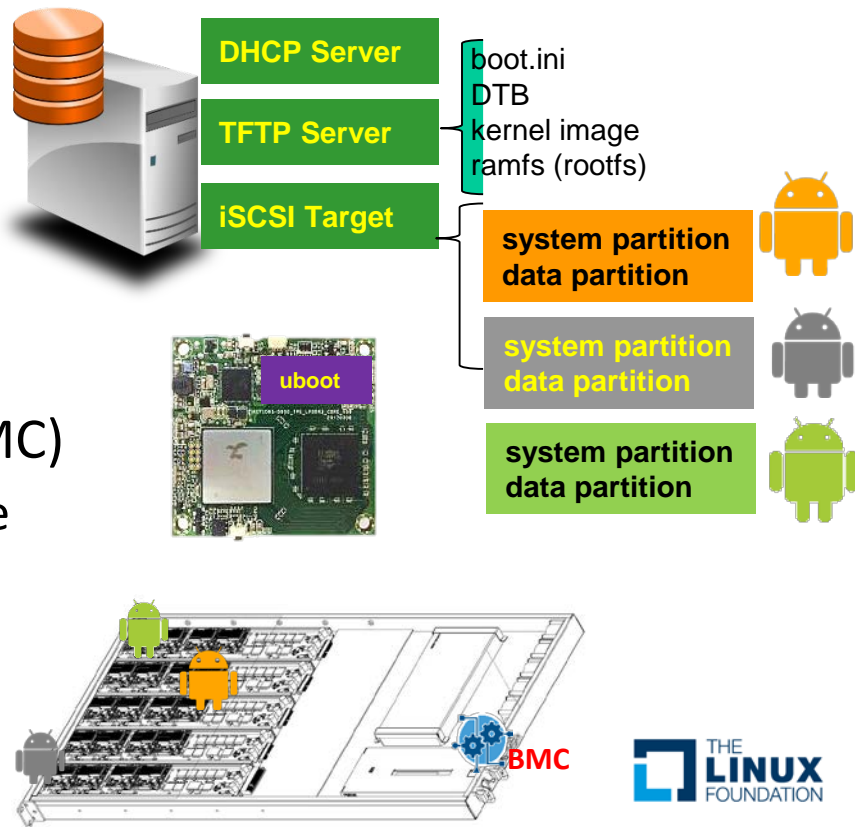


USB 3.0 to Ethernet Gigabit



Manageability for ITRI ARM SOC Cluster

- Diskless boot for host and containers
- Container lifecycle management
 - A container can be selectively launched and shut-down on any node
- Configurable container display
- Baseboard management controller (BMC)
 - Monitor power status and temperature
 - Remote power control
 - Power policy setting



Comparison of APP Streaming Platforms

	APPs Compatibilities	Per Android Instance Cost	Per Android Instance Power Consumption
X86 + Android Virtual Machine	~ 60%	~\$370 USD	< 30W (average)
X86 + Android Container	~ 70%	~\$185 USD	< 15W (average)
ARM SOC Cluster + Android Virtual Machine	> 90%	~\$56USD	< 5W
ARM SOC Cluster + Android Container	> 90%	~\$28 USD	< 2.5W

- ARM: ARM Cortex-A53, 4 cores @ 1.7GHz, RAM: 3GB, 3 containers per node
- X86: Intel Xeon, 8 cores @ 3.5GHz RAM: 64GB, > 40 containers



Conclusions

Conclusions

- Ability to run Android applications in cloud has many applications.
 - App streaming and Virtual Mobility Infrastructure (VMI).
- Android containerization reduces the per-Android-instance cost and enables GPU sharing.
- Using ARM SOC for smartphones to build an Android application cluster solves the binary compatibility problem and significantly improves the cost of scalability.
- Future work
 - Installable Android Container to non-Android Linux distributions
 - Serverless computing model for running Android applications in the cloud

Android Containerization on github

- <https://github.com/clondroid>
- In January, 2018, CBA team released “CLONDRROID”
- Enabling multiple tailored Android containers to run on a Google Pixel XL 2 phone



Android Containerization

CBA (Container-based Android)

📍 ICL/ITRI @ Hsinchu, Taiwan



Android Container on
Google Pixel 2



CBA Team Members



Tzi-cker Chiueh,
General Director of ICL/ITRI



Sting Cheng



Victor Hsu



Tian-Jian Wu



Te-Yu Tsai



Ian Tsai



I-Fan Wang





Thank  & You!



Appendix

Comparisons – Design Considerations (2)

Its all about h/w resource sharing and isolation

resource	phone	cloud deployment
	<ul style="list-style-type: none">• shared between and host and containers• switch between host and containers	<ul style="list-style-type: none">• individual virtual display for each container• stream display content over the Internet
	<ul style="list-style-type: none">• shared between and host and containers• input goes to active one	<ul style="list-style-type: none">• touch and other sensor data is streamed up from user device to containers
	<ul style="list-style-type: none">• shared between and host and containers• exclusive or non-exclusive	<ul style="list-style-type: none">• individual virtual device for each containers• stream voice content over the Internet
	<ul style="list-style-type: none">• shared between and host and containers• exclusive	<ul style="list-style-type: none">• Camera re-direction
Intent/ Notification	<ul style="list-style-type: none">• isolated in each container space	<ul style="list-style-type: none">• Intent/Notification re-direction

BMC

- Commercial BMC chip in x86 world does NOT fit in ARM Cluster
- S/W
 - Facebook OpenBMC for Yosemite
 - Yosemite consists of modular chassis for x86 microservers, which is architecturally similar to ARM cluster
- H/W
 - BBG controller board
- Features
 - Web API for power control and power policy setting
 - Sensors
 - FAN Control
 - Watchdog

