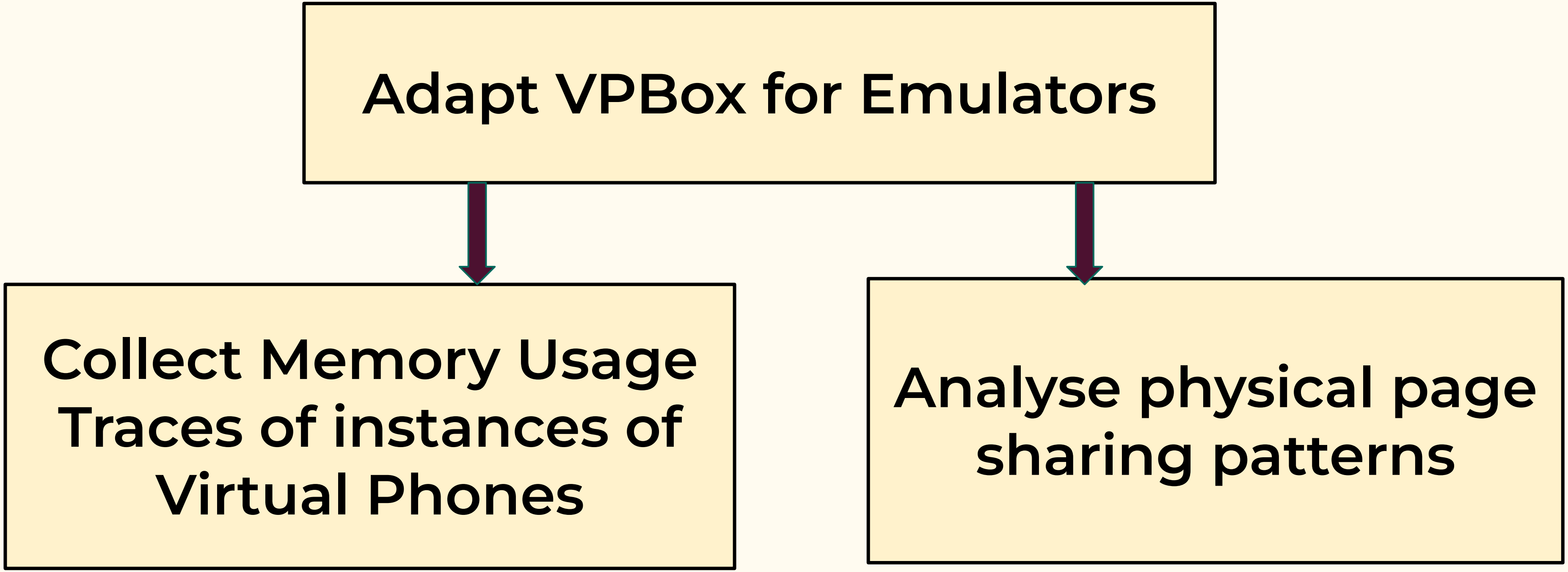
Memory Overhead Analysis of Container-based Android sandboxes

Aditi Goyal - 190057
Yatharth Goswami - 191178
Mentor: Prof Debadatta Mishra

Objective



Adapt VPBox for Emulators

Collect Memory Usage Traces of instances of Virtual Phones

Analyse physical page sharing patterns

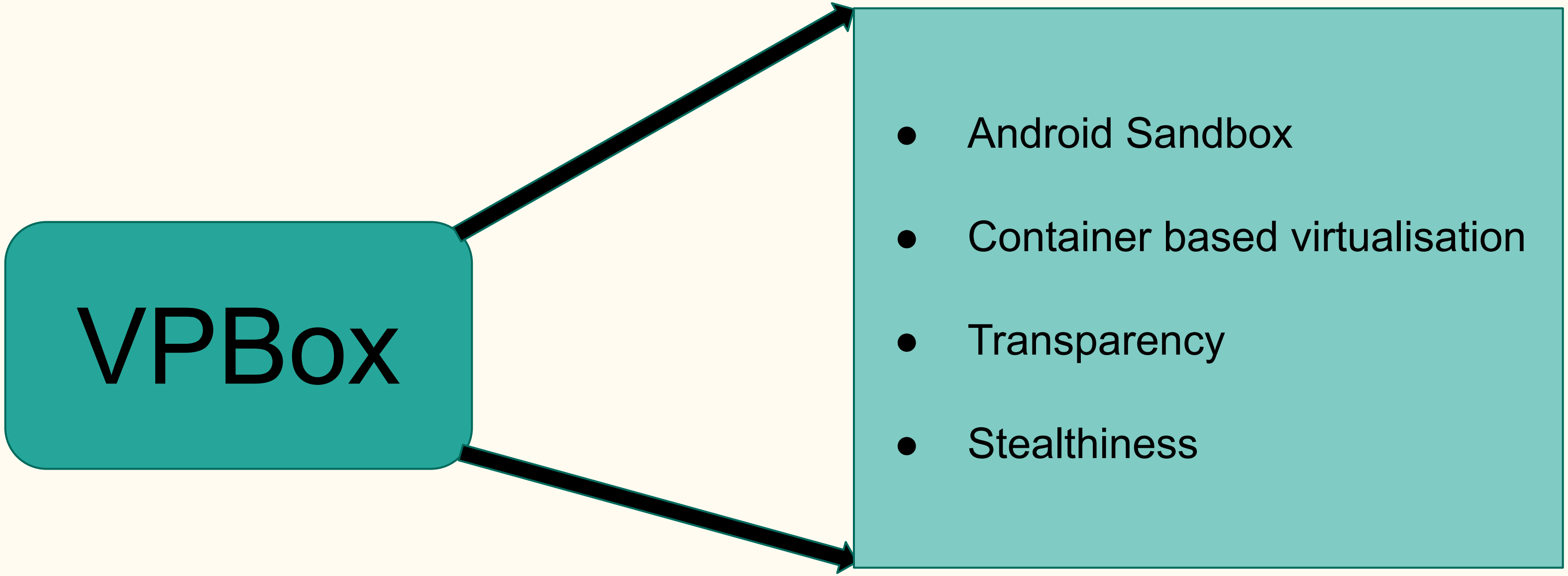
Container-Based Virtualization

Hardware resource multiplexing for all devices

Incomplete device virtualization

Lack of transparency & stealthiness

Introduction

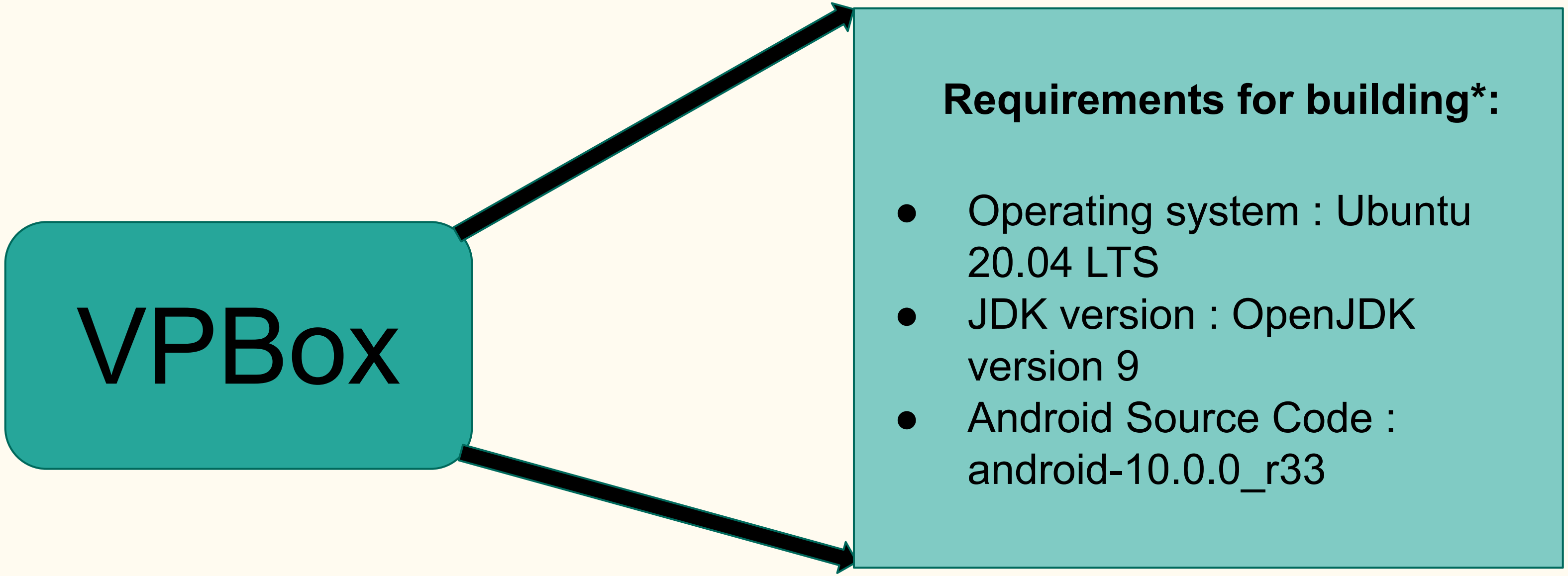


VPBox

- Android Sandbox
- Container based virtualisation
- Transparency
- Stealthiness

VPBox Original Paper: https://dl.acm.org/doi/pdf/10.1145/3460120.3484544

Introduction



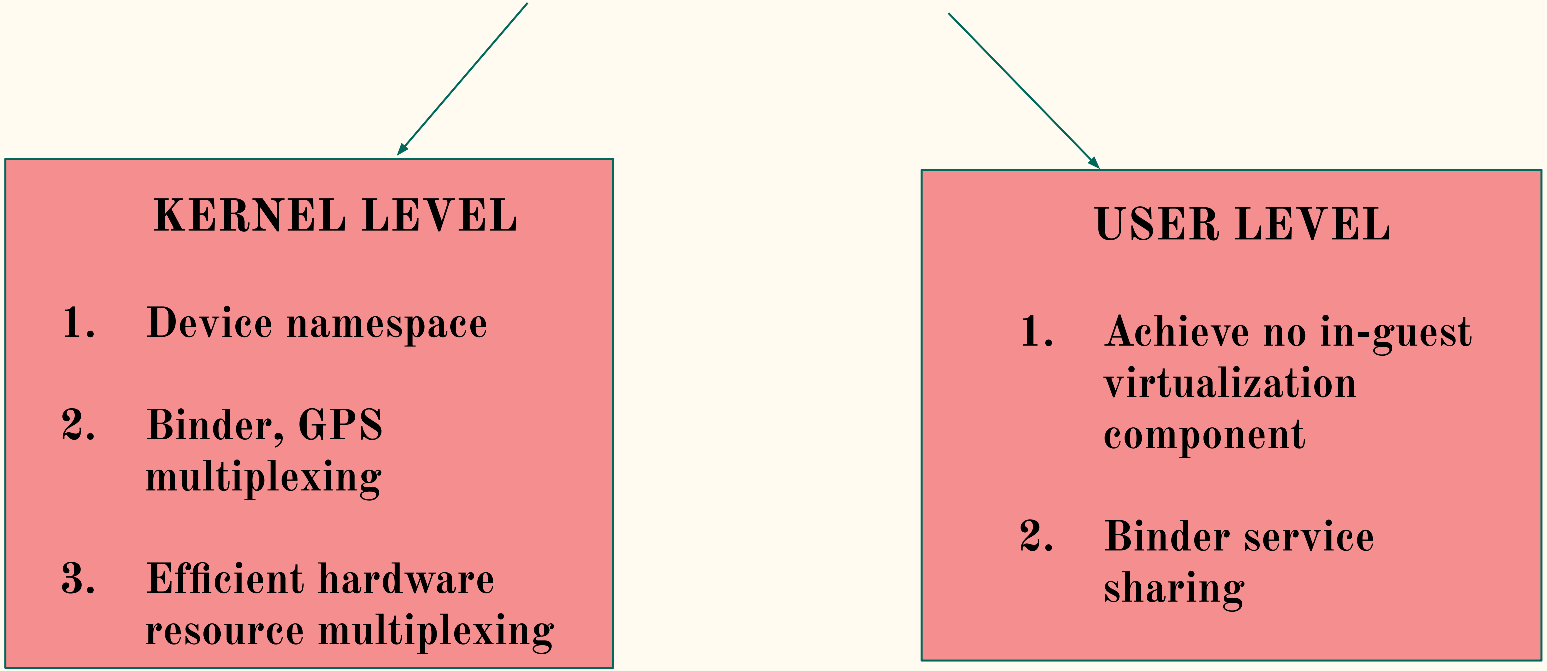
VPBox

Requirements for building*:

- Operating system : Ubuntu 20.04 LTS
- JDK version : OpenJDK version 9
- Android Source Code : android-10.0.0_r33

*Source Code for building for Pixel 3A XL physical device: https://github.com/VPBox/Dev

2 Levels of Virtualization



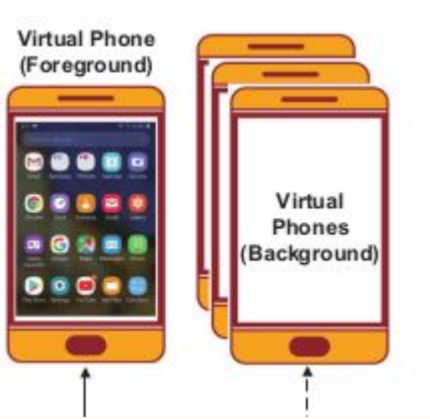
KERNEL LEVEL

1. Device namespace
2. Binder, GPS multiplexing
3. Efficient hardware resource multiplexing

USER LEVEL

1. Achieve no in-guest virtualization component
2. Binder service sharing

Memory Usage Bottleneck : Proposed Solutions



Extend Android

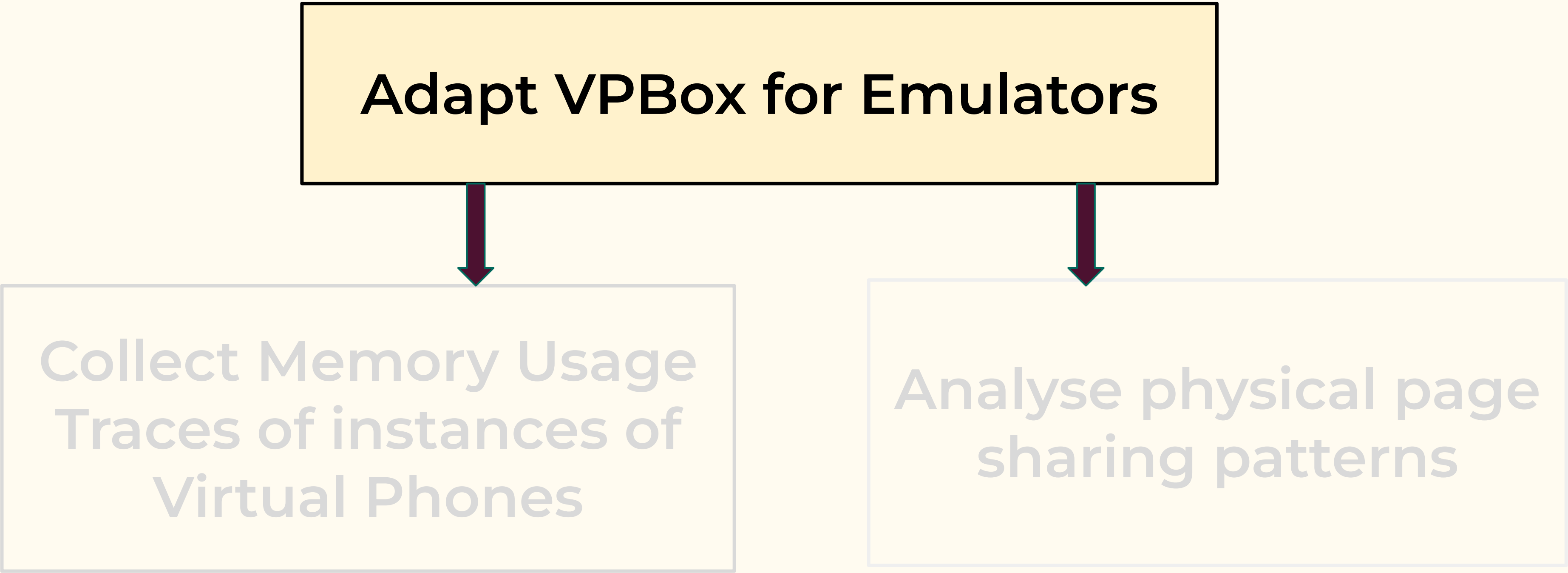
1. KSM
2. Low memory killer

Extra Features

1. AUFS
2. Screen off

Picture Credits : https://dl.acm.org/doi/pdf/10.1145/3460120.3484544

Objective



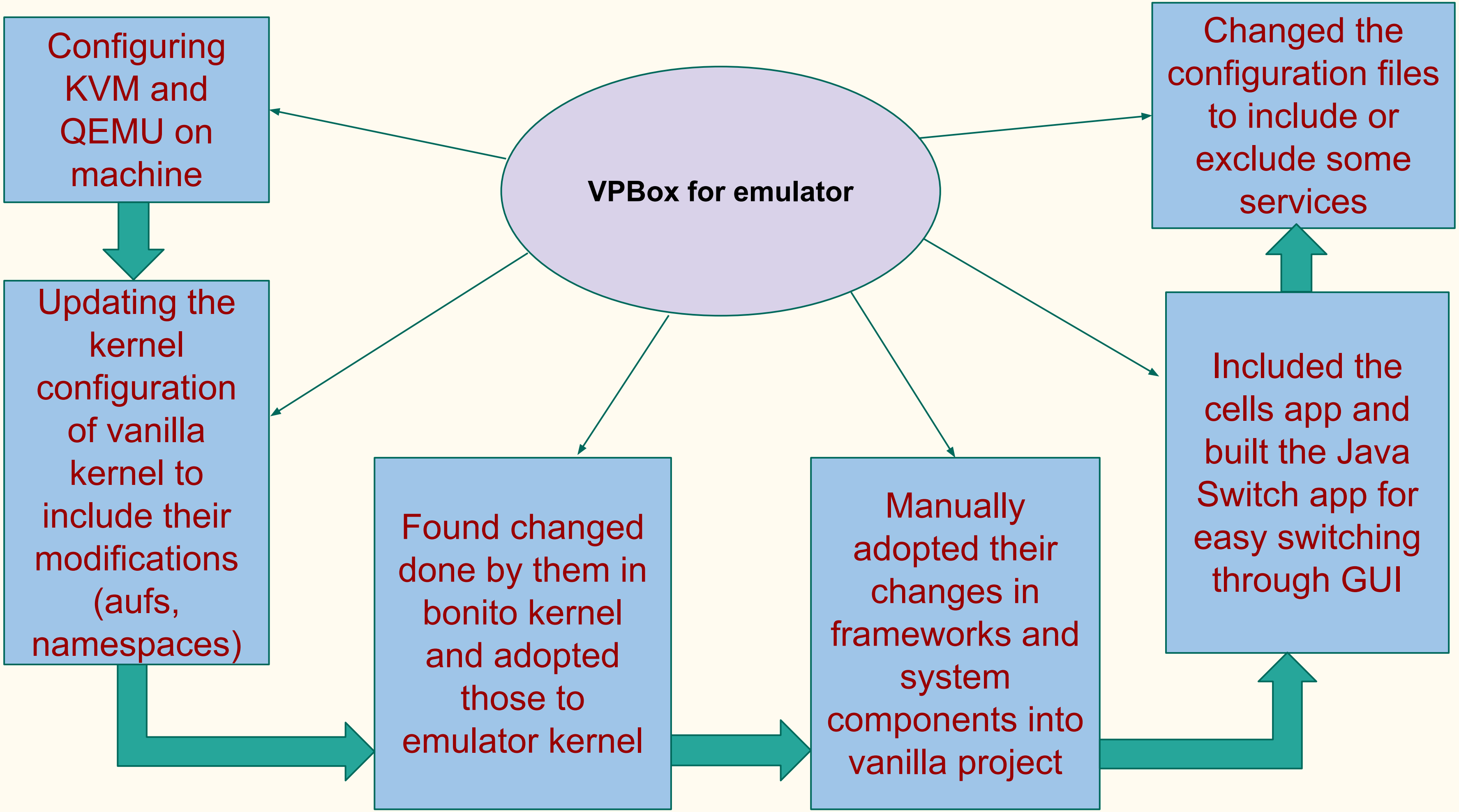
Adapt VPBox for Emulators

Collect Memory Usage Traces of instances of Virtual Phones

Analyse physical page sharing patterns

Device Specifications

- Emulated Physical Device: Pixel 3a XL
- Android version: Android 10.0.0_r33
- Linux kernel version: goldfish-kernel-4.14.112
- Android Emulator API Version 29
- Memory: 8GB RAM



VPBox for emulator

Configuring KVM and QEMU on machine

Updating the kernel configuration of vanilla kernel to include their modifications (aufs, namespaces)

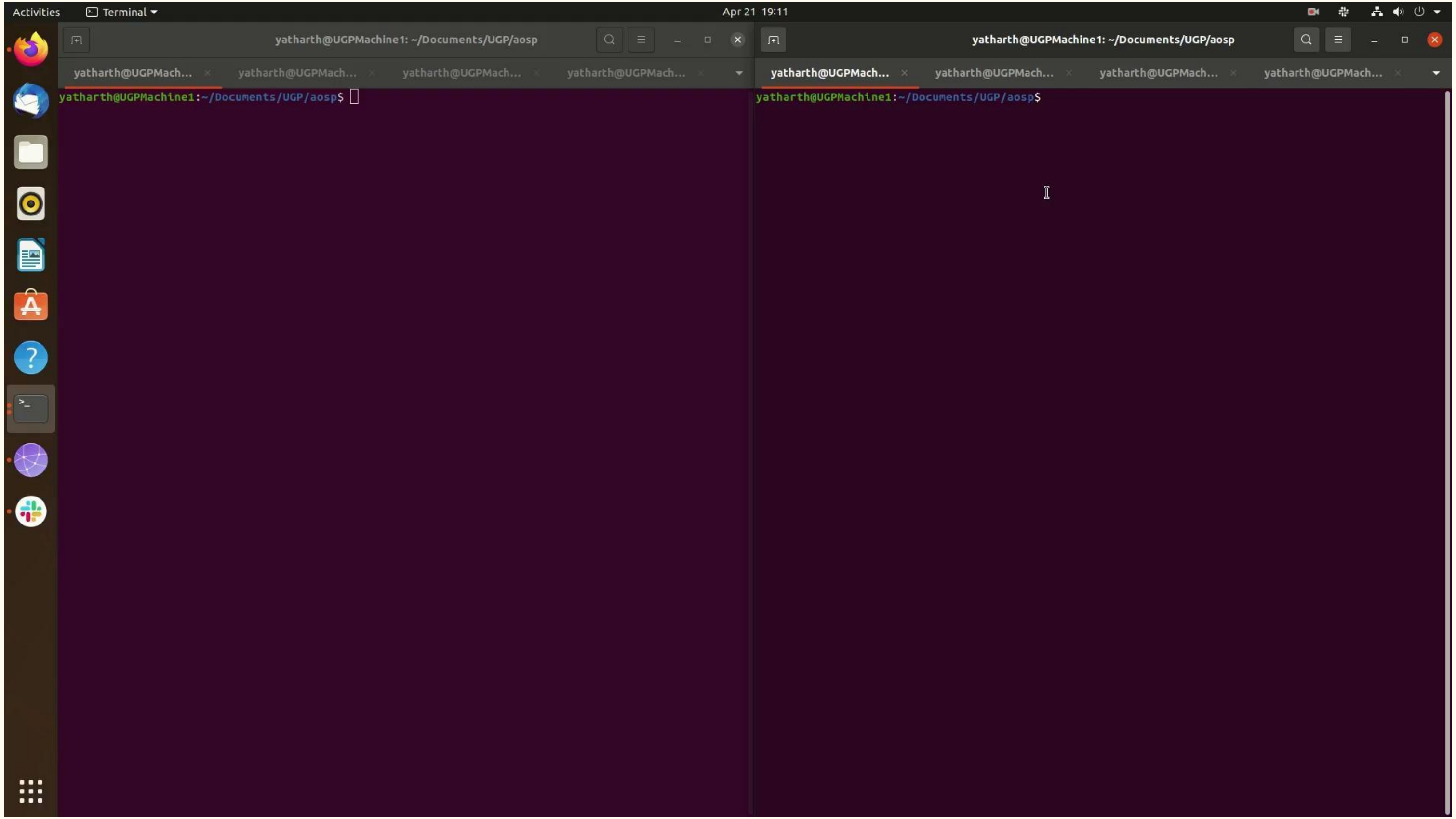
Found changed done by them in bonito kernel and adopted those to emulator kernel

Manually adopted their changes in frameworks and system components into vanilla project

Included the cells app and built the Java Switch app for easy switching through GUI

Changed the configuration files to include or exclude some services

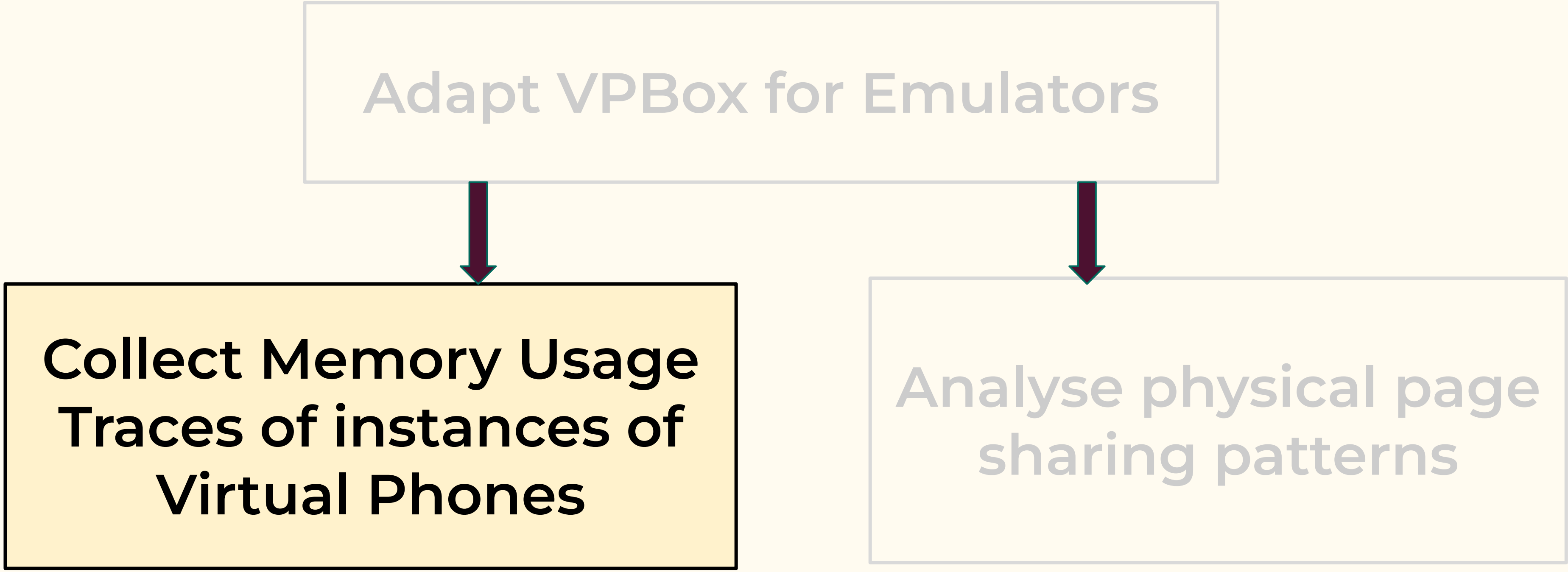
Short Demo..



```
yatharth@UGPMachine1:~/Documents/UGP/aosp$
```

```
yatharth@UGPMachine1:~/Documents/UGP/aosp$
```

Objective



Adapt VPBox for Emulators

Collect Memory Usage Traces of instances of Virtual Phones

Analyse physical page sharing patterns

Collecting memory usage data

1. Implemented a BFS inside kernel which on being given a start pid, walks over VM areas of all the processes in the subtree of the given process.

2. Identified physical mappings for the processes by following page table entries.

3. Collected the following information about each virtual address

   a. Physical mapping

   b. VM area permissions + PTE permissions

   c. File path

4. Used the pseudo sysfs filesystem in linux kernel to get the above information for init processes of all the virtual phones as well as host by setting up callbacks for the same inside kernel.

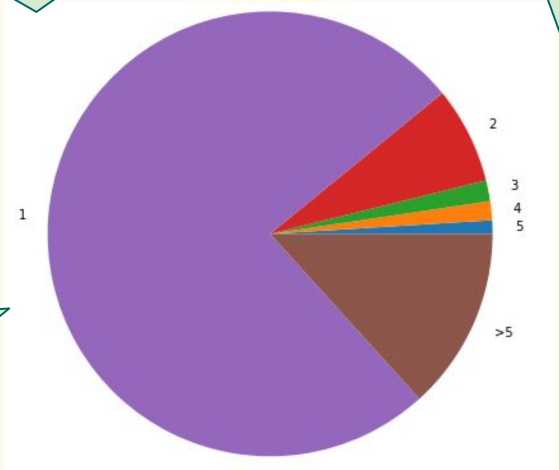
5. We have removed the physical memory mappings corresponding to device /goldfish_pipe (related to emulator) so as to compare the results with the paper.

Memory Overhead Evaluation

Only Host Active:

Number of pages vs Sharing count

There is page sharing across processes!



Total usage : 952 MB

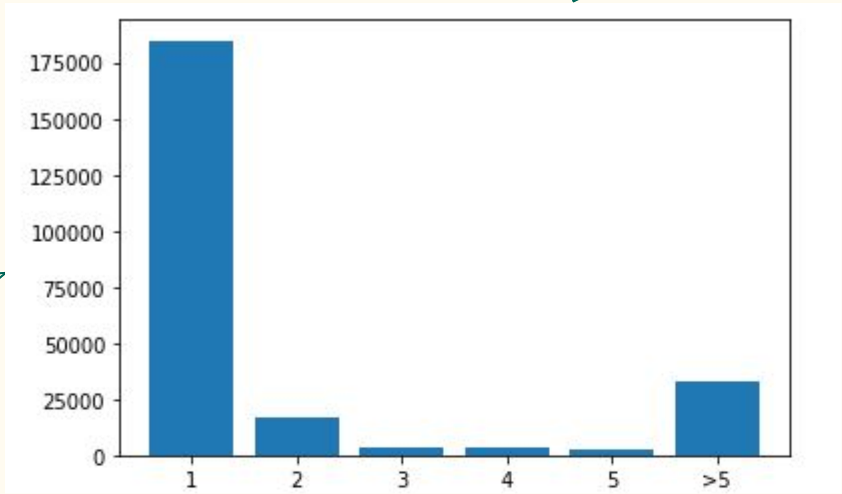
Note: Data is taken without running any applications

Memory Overhead Evaluation

Only Host Active

Number of pages vs Sharing count

There is page sharing across processes!



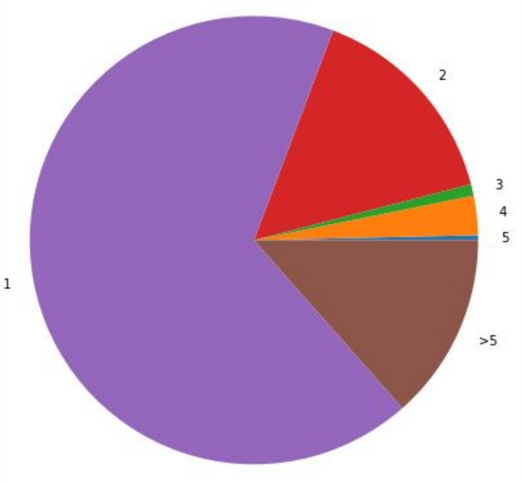
Total usage : 952 MB

Memory Overhead Evaluation

Host + VP1:

Number of pages vs Sharing count

There is page sharing across host and VP1!



Total usage : 1388 MB

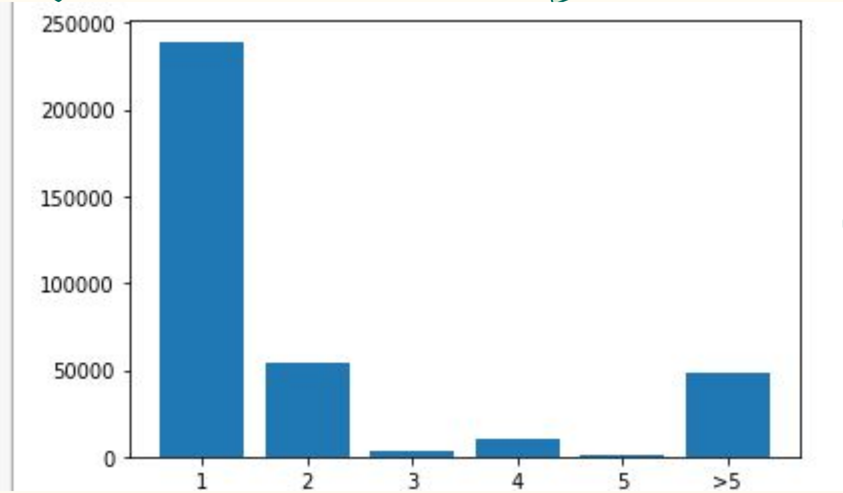
Note: Data is taken without running any applications

Memory Overhead Evaluation

Host + VP1:

Number of pages vs Sharing count

Amount of sharing between host and VP1 : 363 MB



Total usage : 1388 MB

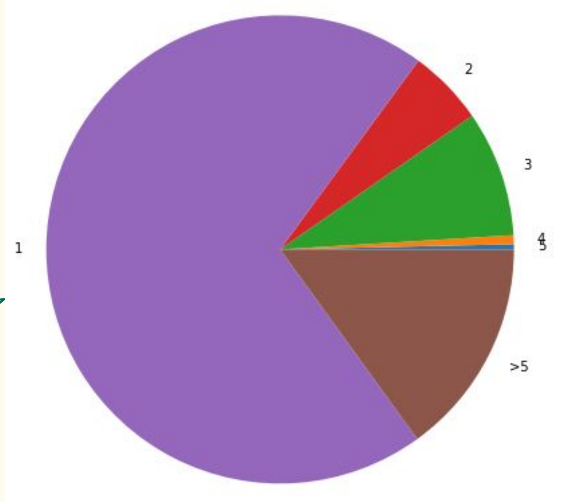
Proportion of pages shared 2 times increased!

Memory Overhead Evaluation

Host + VP1+VP2:

Number of pages vs Sharing count

There is page sharing across host, VP1 and VP2!



Total usage : 1697 MB

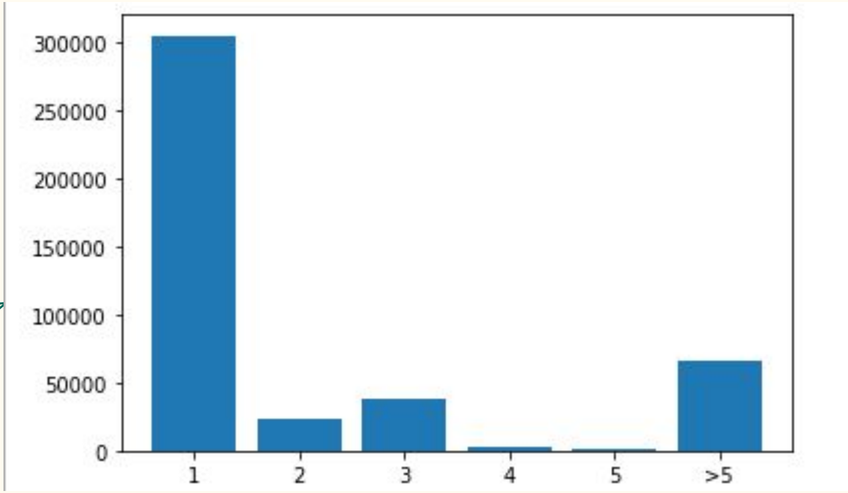
Note: Data is taken without running any applications

Memory Overhead Evaluation

Host + VP1+VP2:

Number of pages vs Sharing count

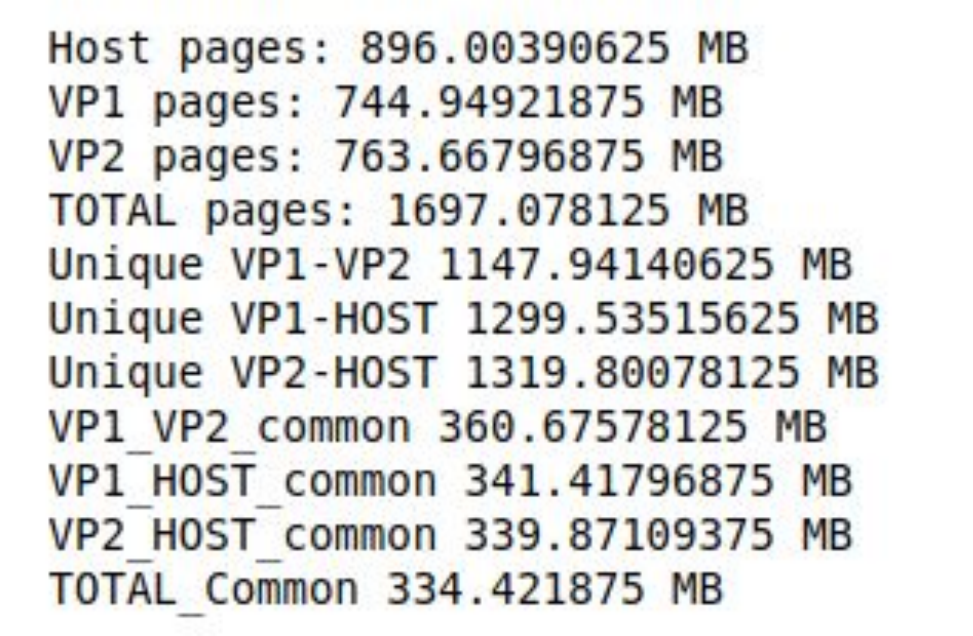
Amount of sharing among all three: 334 MB



Proportion of pages shared 3 times increased!

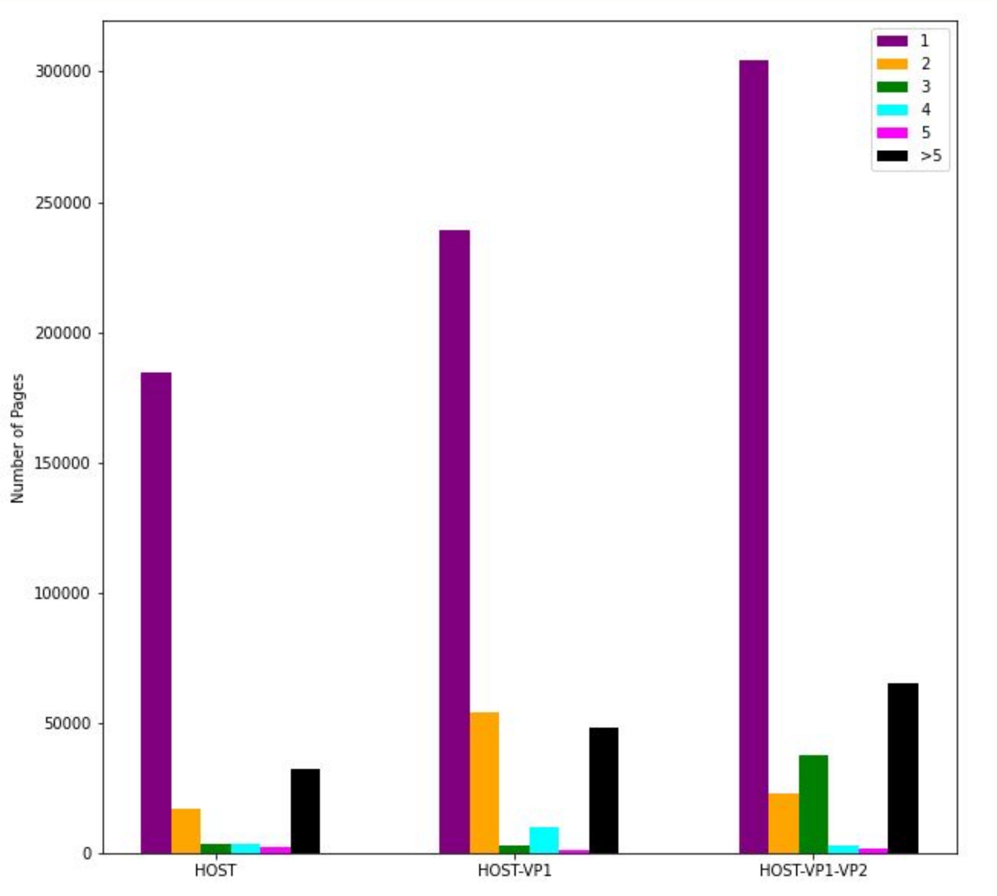
Total usage : 1697 MB

More Information

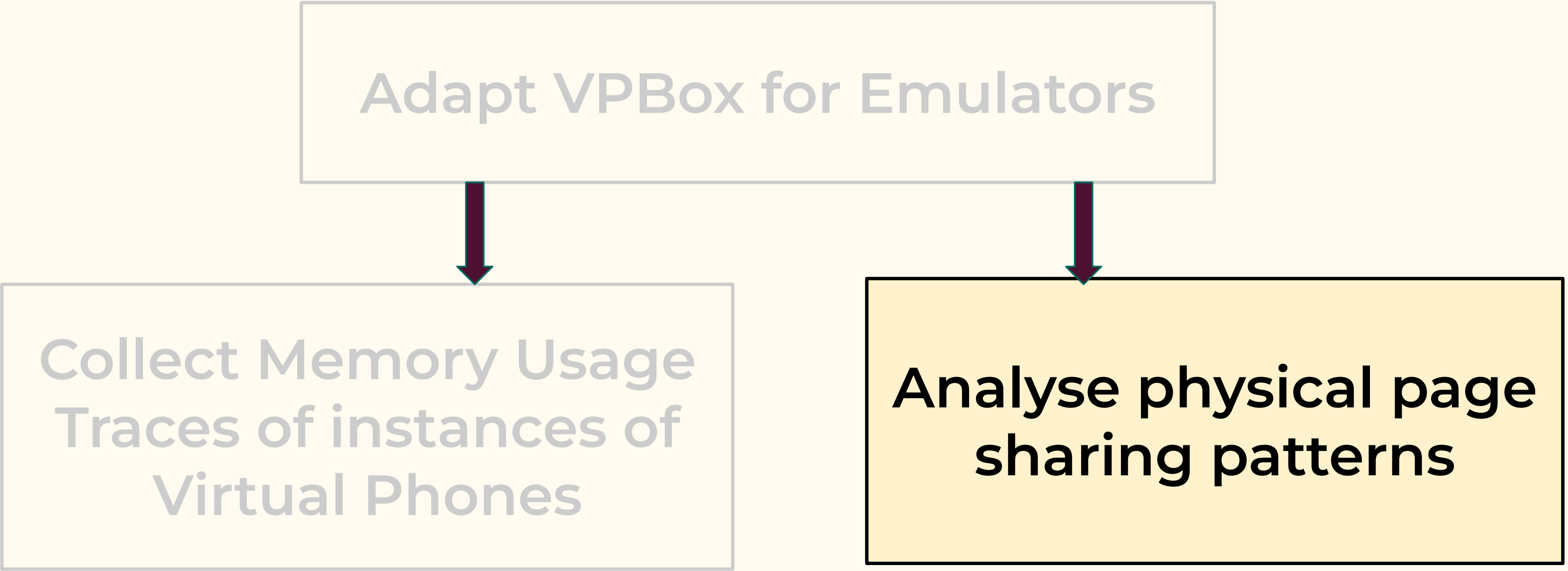


```
Host pages: 896.00390625 MB
VP1 pages: 744.94921875 MB
VP2 pages: 763.66796875 MB
TOTAL pages: 1697.078125 MB
Unique VP1-VP2 1147.94140625 MB
Unique VP1-HOST 1299.53515625 MB
Unique VP2-HOST 1319.80078125 MB
VP1_VP2_common 360.67578125 MB
VP1_HOST_common 341.41796875 MB
VP2_HOST_common 339.87109375 MB
TOTAL_Common 334.421875 MB
```

Summarizing it..



Objective



Adapt VPBox for Emulators

Collect Memory Usage Traces of instances of Virtual Phones

Analyse physical page sharing patterns

Observations

Ineffectiveness in sharing shared object files (libraries):

- We found the top VMAs which have unshared physical pages and their corresponding files.
- Analysed the library files(.so) present in the above list to see their association with namespaces.
- Found that the unshared pages corresponding to these files are equally distributed among HOST, VP1 and VP2.
- Example: ‘/system/lib64/libcrypto.so’ has 399 physical pages unique to host and 304 pages unique to both VP1 and VP2

Observations

Memory overhead of Virtual Phone

- We observe that overhead of each of the virtual phone is around 350-400 MB. This is somewhat consistent with what are results shown in the paper.

- The amount of pages which are unique across VP phones increase by around 250-300 MB, which is also consistent with what is shown in the paper.

Observations

KSM analysis

- We took the pte flags as well while translating a virtual address.
- Next, we found the addresses which have the VM area permissions as ‘rw’ but pte_flags as only ‘r’.
- These might be the pages that are being handled by KSM.
- Most of these pages corresponded to ANONYMOUS mappings or same namespace.

Future Work

Root Cause of the memory overheads

Aggressiveness of KSM

Security related analysis and network overheads

AUFS => OverlayFS?

QUESTIONS?

THANK YOU!