Trinity: High-Performance Mobile Emulation through Graphics Projection

Di Gao, **Hao Lin**, Zhenhua Li, Chengen Huang, Yunhao Liu

Feng Qian, Liangyi Gong, Tianyin Xu

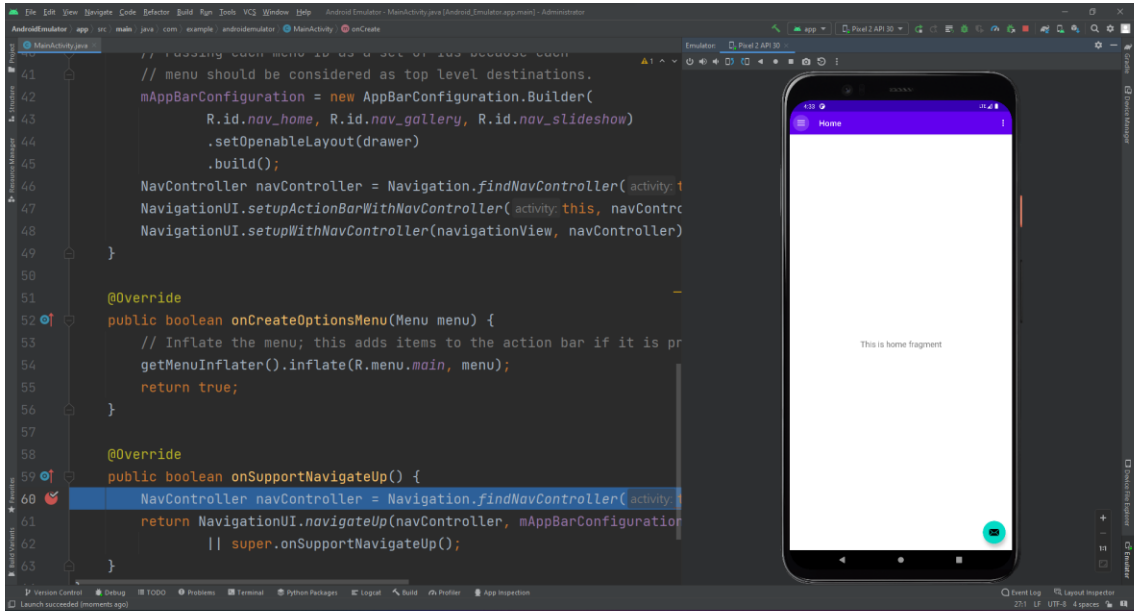
清華大學 Tsinghua University

University of Minnesota

CNIC

University of Illinois Urbana-Champaign

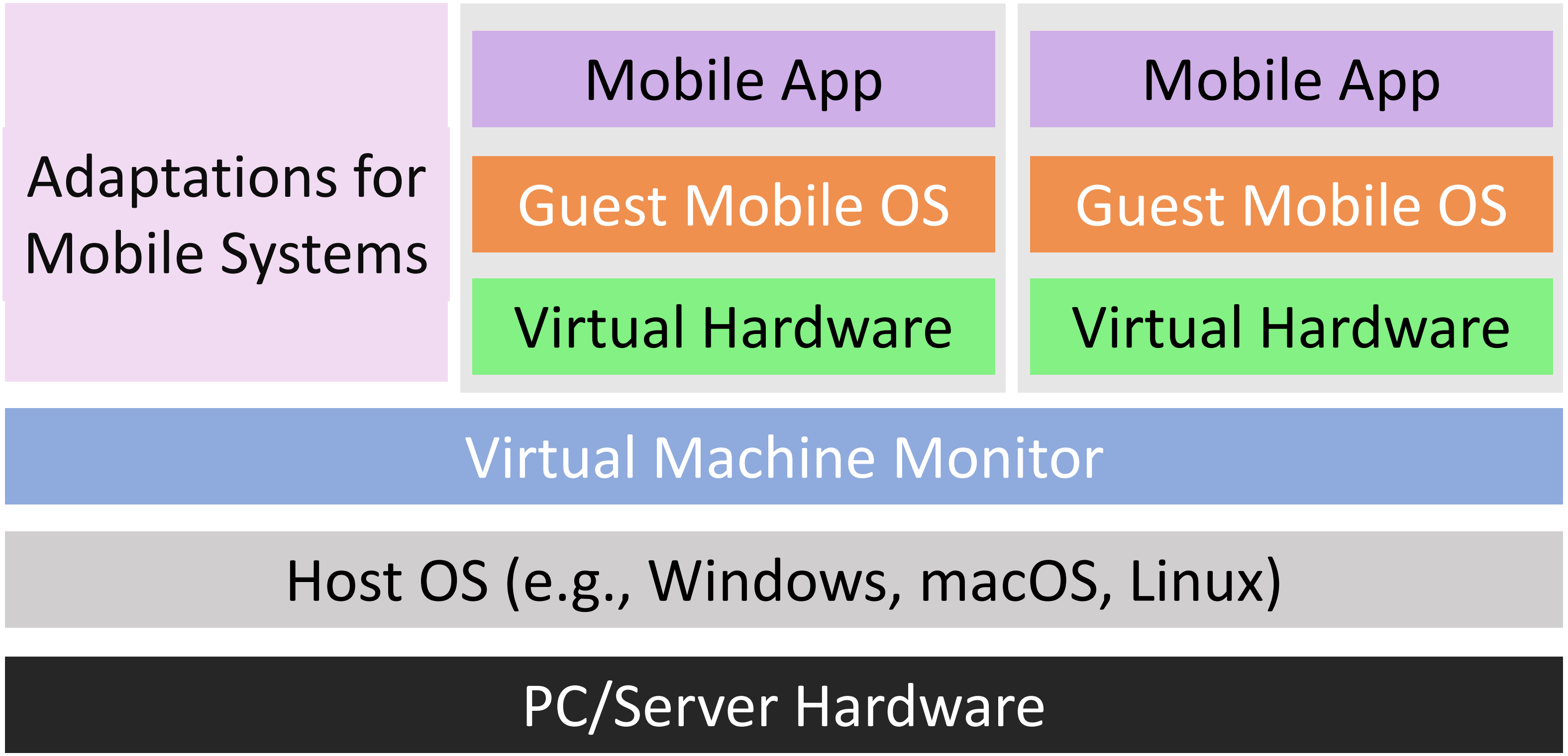
Mobile emulator: phone on your PC/server



- App debugging w/o hardware phones
- PC-based mobile gaming
- Malware detection, cloud/edge gaming …

What is a mobile emulator?

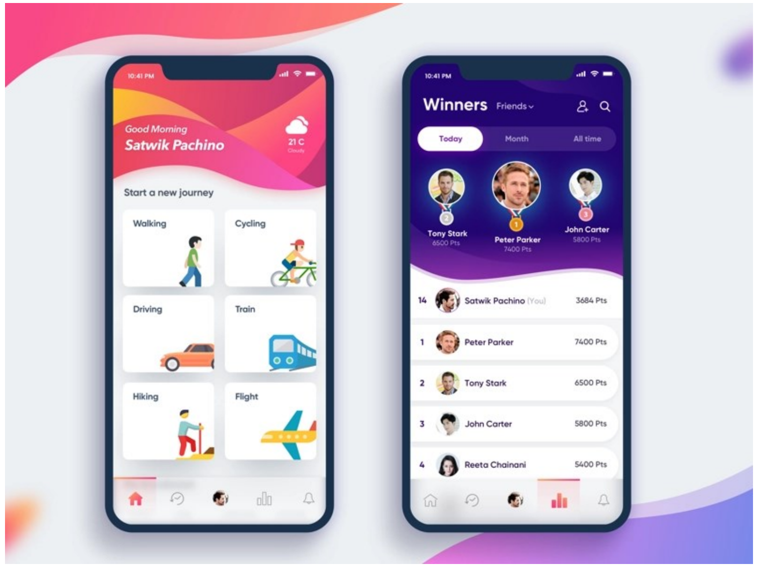
- A mobile emulator is a virtual machine

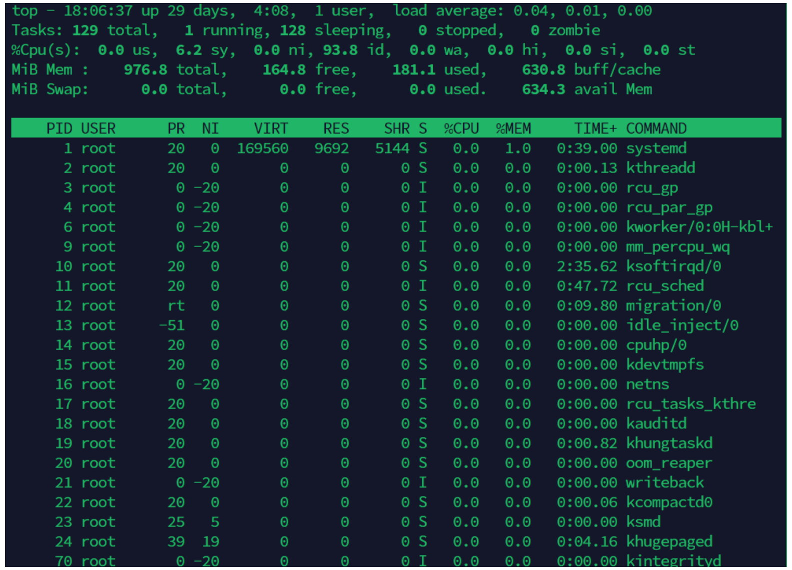


What is a mobile emulator?

- A mobile emulator is **more than** a traditional virtual machine

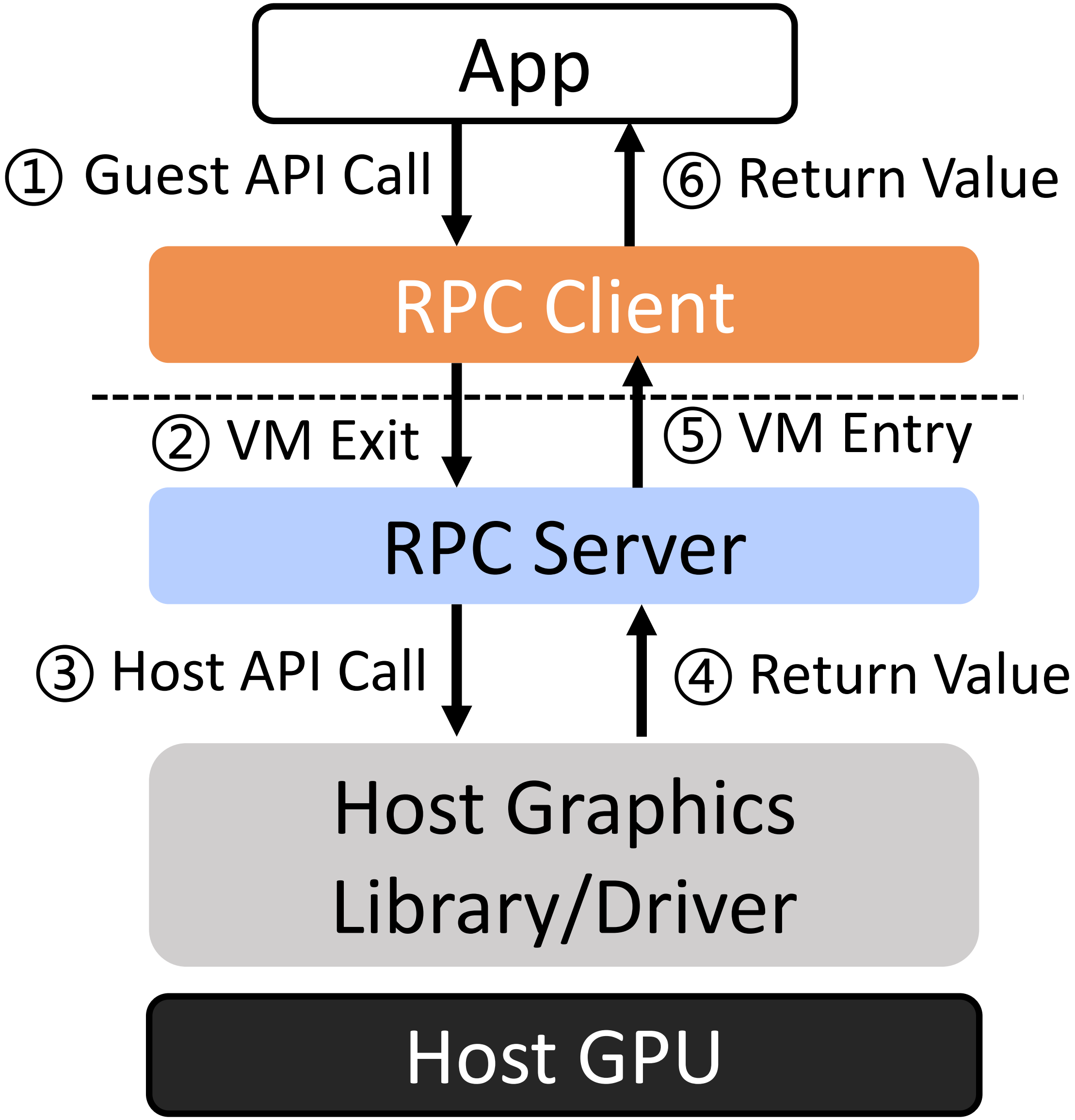
UI-centric mobile OSes ***vs.*** Headless server OSes





Graphics processing capability is key to the performance of mobile emulators

2002: API remoting : the idea of RPC



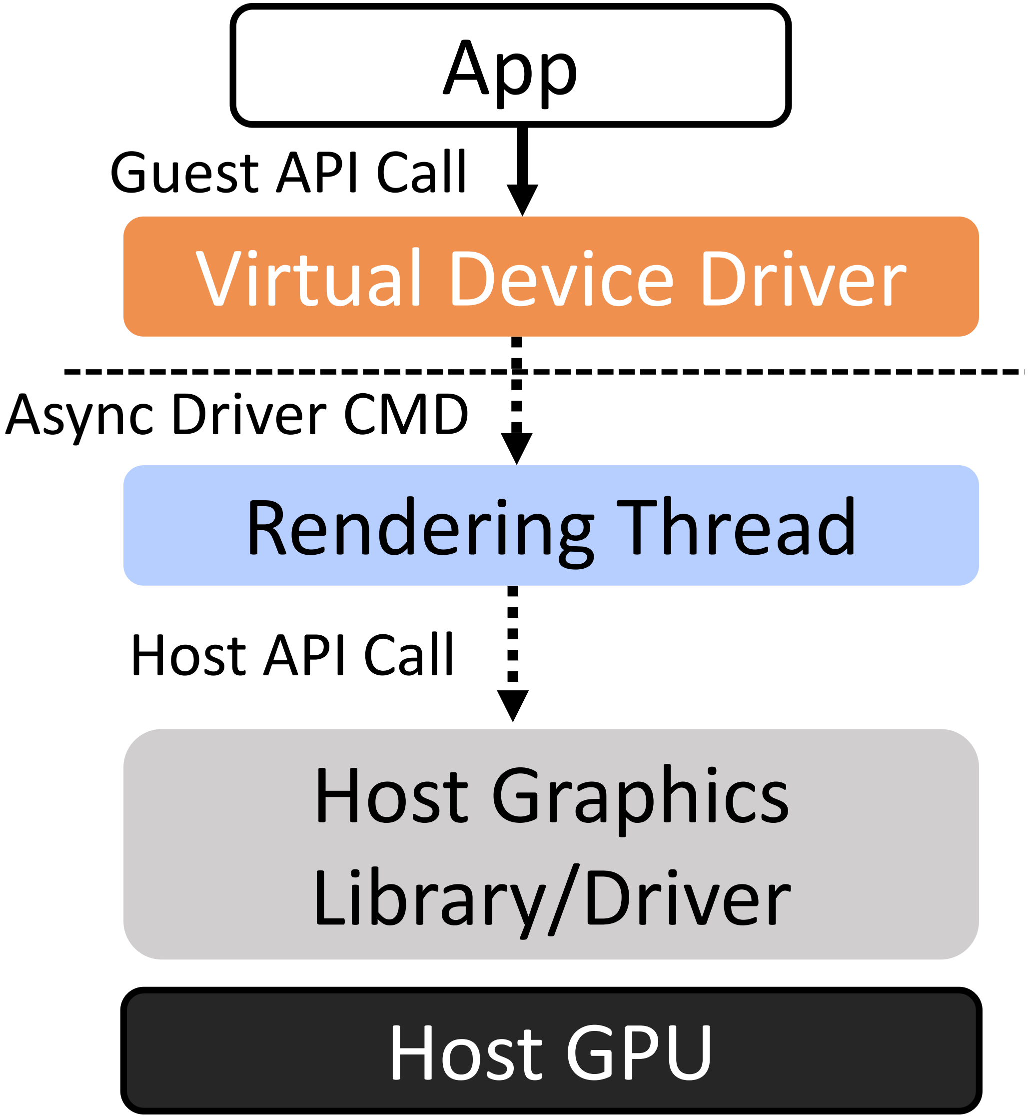
+ Straightforward implementation

- Frequent VM Exits stop the guest

- Cannot smoothly run common apps

E.g., Google Android Emulator (GAE)

2009: device emulation: async driver commands



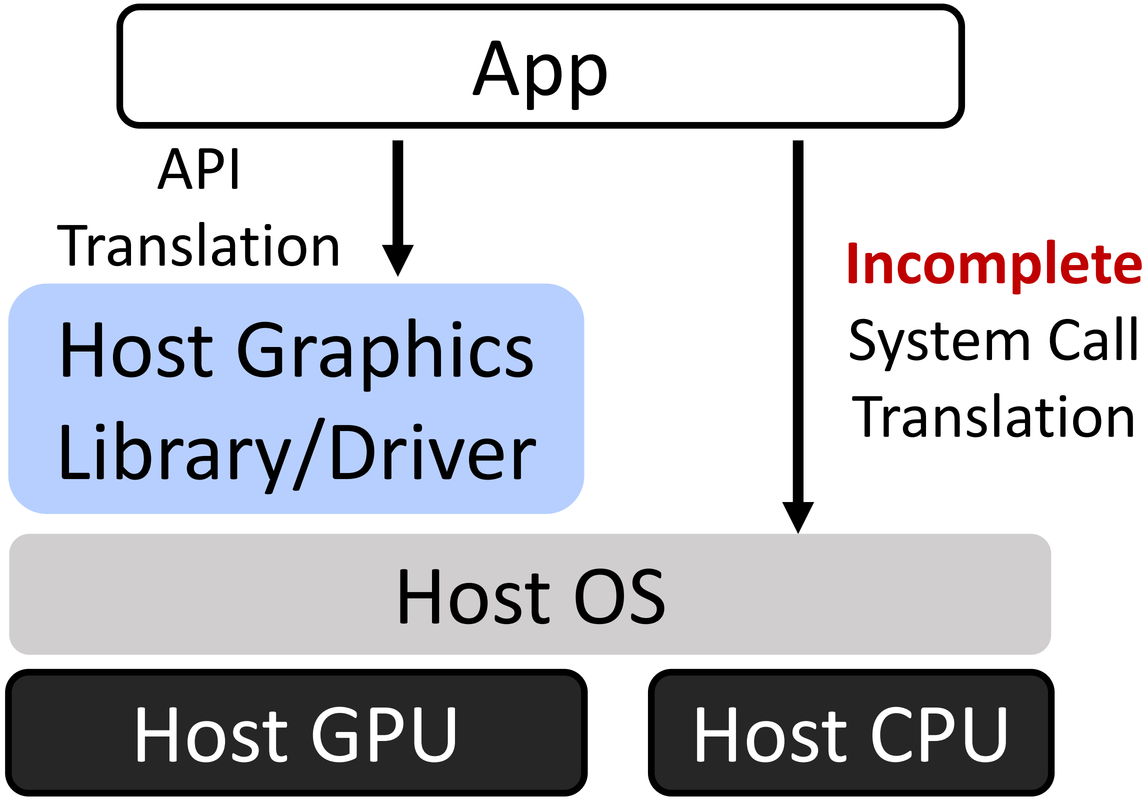
+ Reduced idle waiting at the guest

- Single-threaded rendering due to the loss of high-level information

- Cannot smoothly run heavy 3D apps

E.g., QEMU-KVM with virtio-gpu

2018: direct emulation: breaking virtualization



+ Satisfactory efficiency
- Guest-host Isolation (security) is damaged
- Compatibility is sacrificed

E.g., DAOW (Tencent Gameloop)

Our goal

A mobile emulator that can achieve
high efficiency and **compatibility**

In retrospect…

Virtualization-based mobile emulators
do well in compatibility (and security) but poorly in **efficiency**

⬇

Frequent VM Exits for **synchronous** host-side executions of API calls

⬇

Our wish: let the host asynchronously process synchronous API calls

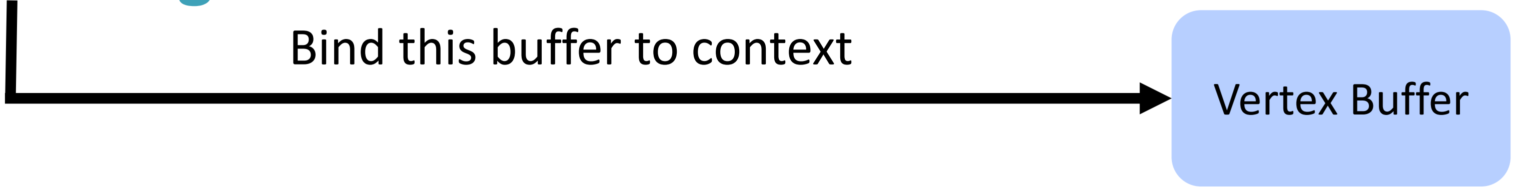
Contributions

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

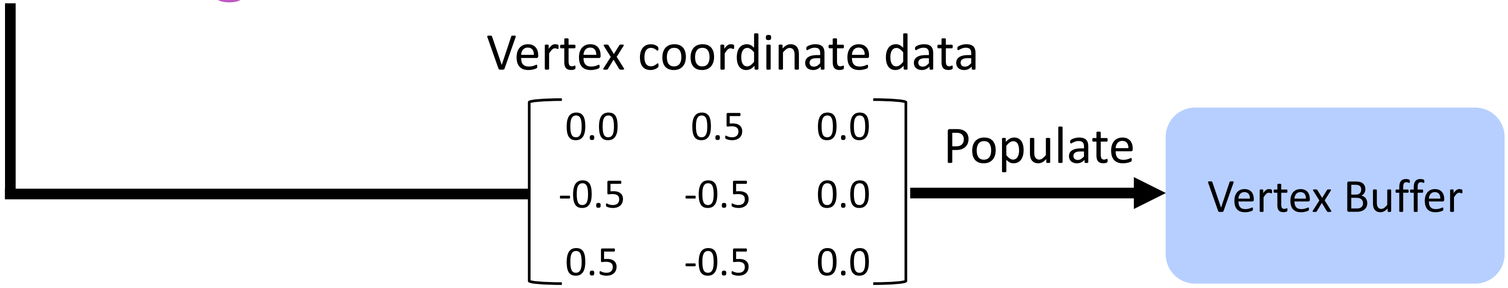
- A novel graphics virtualization method called **graphics projection**
  - **Decoupling** guest and host graphics processing
  - **Elastic flow control** for coordinating the decoupled control flows
  - **Adaptive data teleporting** for fast data flow delivery
- Trinity: the first and the only mobile emulator that can achieve **native efficiency** without loosing compatibility or security
  - Evaluation using standard benchmarks and real apps
  - Adoption by **Huawei DevEco Studio**, an Android IDE with millions of developers, to replace its originally used Google Android Emulator

"Hello, Triangle!"

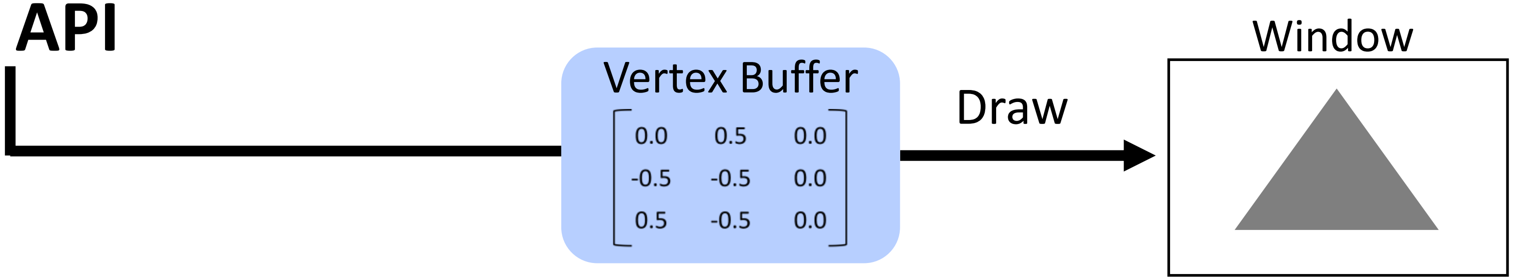
- Draw a triangle with Android's graphics framework OpenGL ES
  - 1. Context setting API



  - 2. Resource management API



  - 3. Drawing API



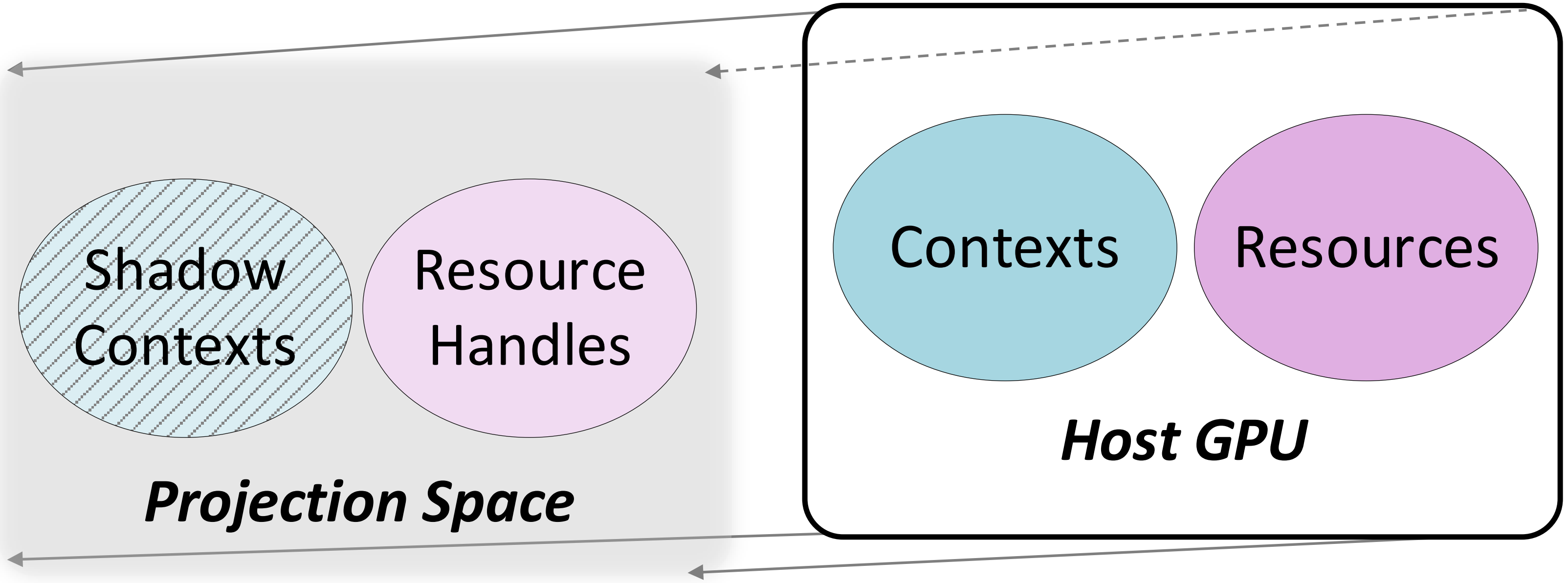
Characteristics of graphics APIs

- Many sync APIs do not immediately involve GPU
  - Context and Resource calls take effect upon actual drawing
  - Context and Resource calls account for 94% of all API calls
- Such APIs are fully handle-based (a handle is a small integer)

```
uint handle;
glGenBuffers(1, &handle);                // Resource call
glBindBuffer(GL_ARRAY_BUFFER, handle); // Context call
```

Key idea: graphics projection

- Project host-side contexts/resources onto the guest address space



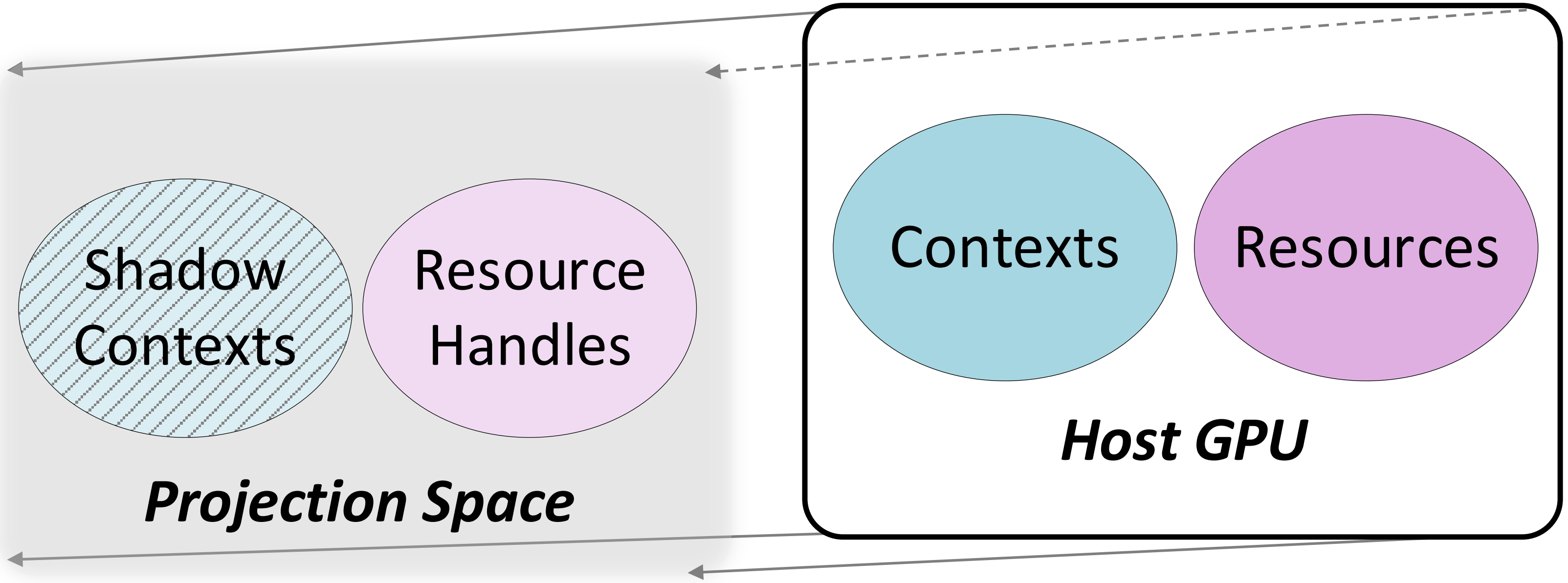
Shadow context: currently bound handle

Resource (graphics buffer) handle

```
uint handle;
glGenBuffers(1, &handle);                  // Resource call
glBindBuffer(GL_ARRAY_BUFFER, handle); // Context call
```

Key idea: graphics projection

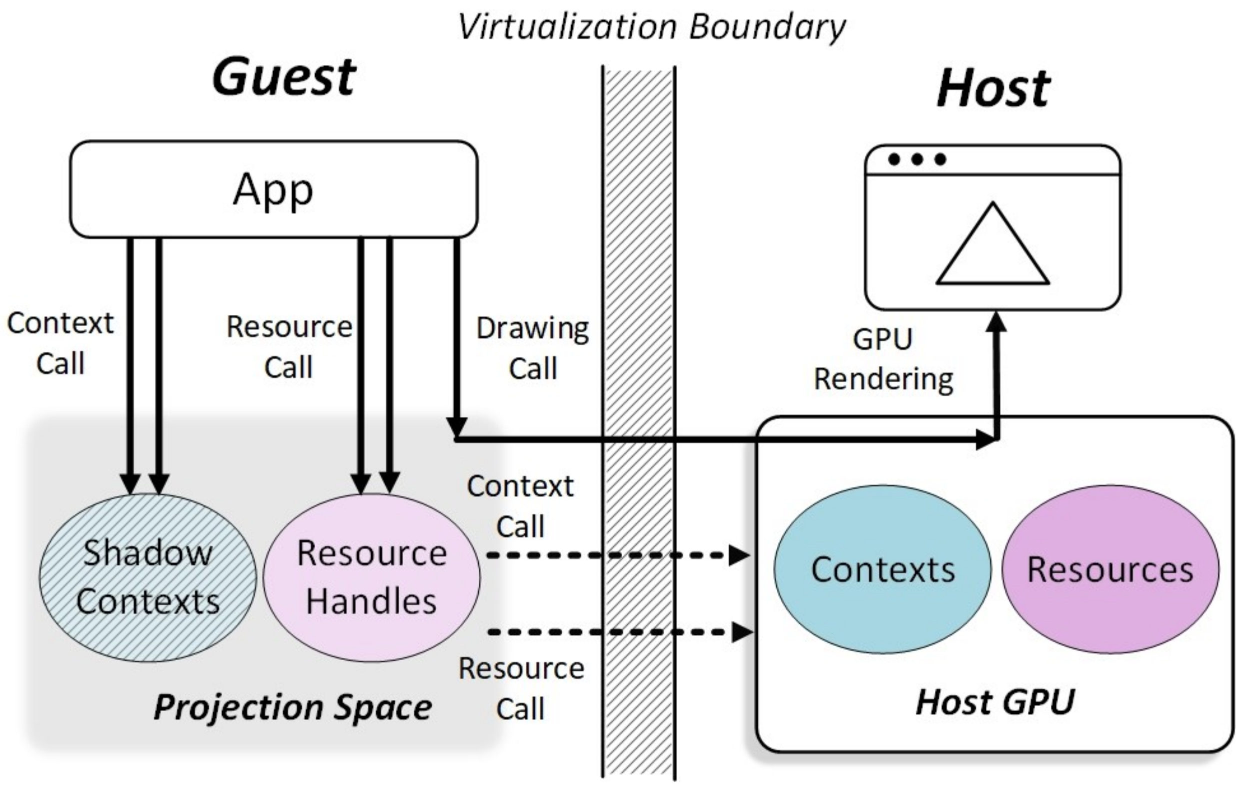
- Project host-side contexts/resources onto the guest address space



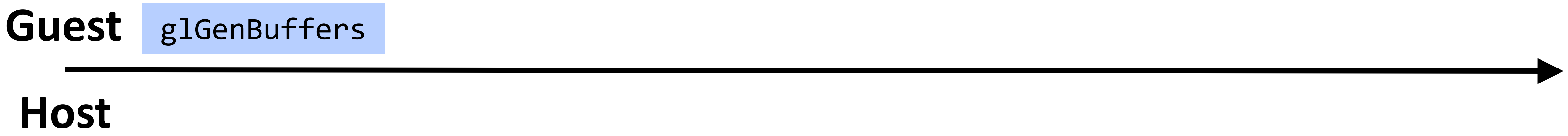
Shadow contexts and resource handles
"cache" the effect of Context and Resource calls

Decouple host and guest control/data flows

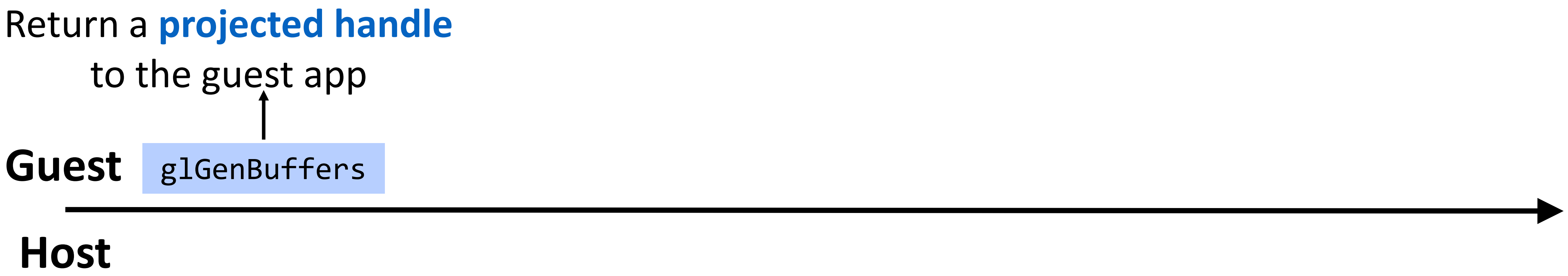
- Most Context and Resource calls are processed in the projection space
- Their effects are **asynchronously reproduced** by the host GPU
- Drawing calls are already async



Draw a triangle with graphics projection

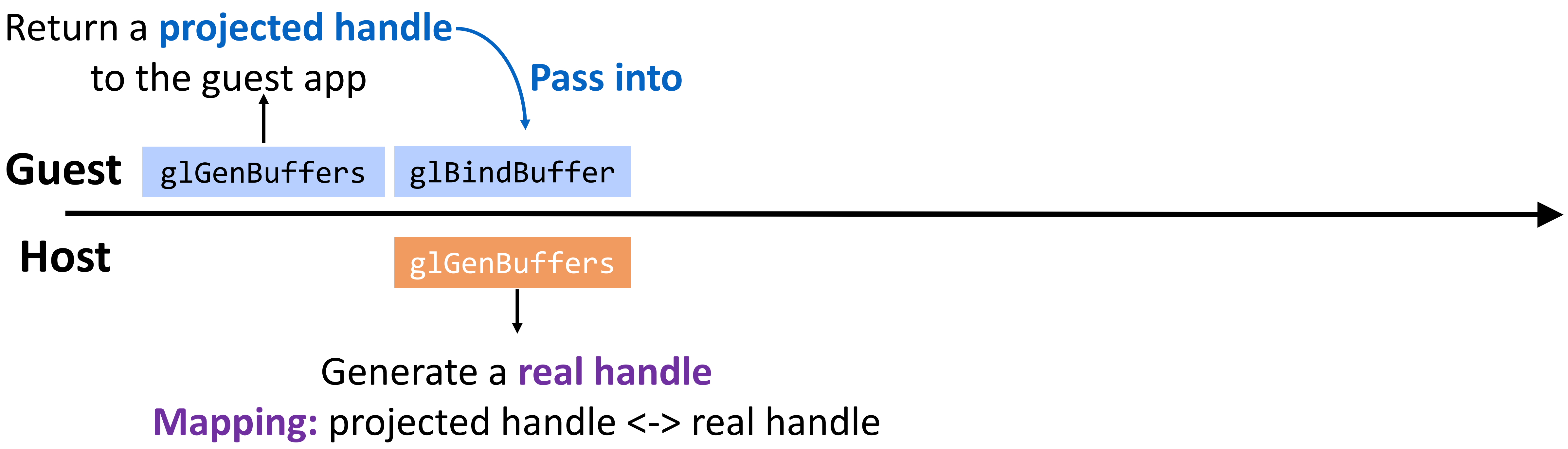


Draw a triangle with graphics projection

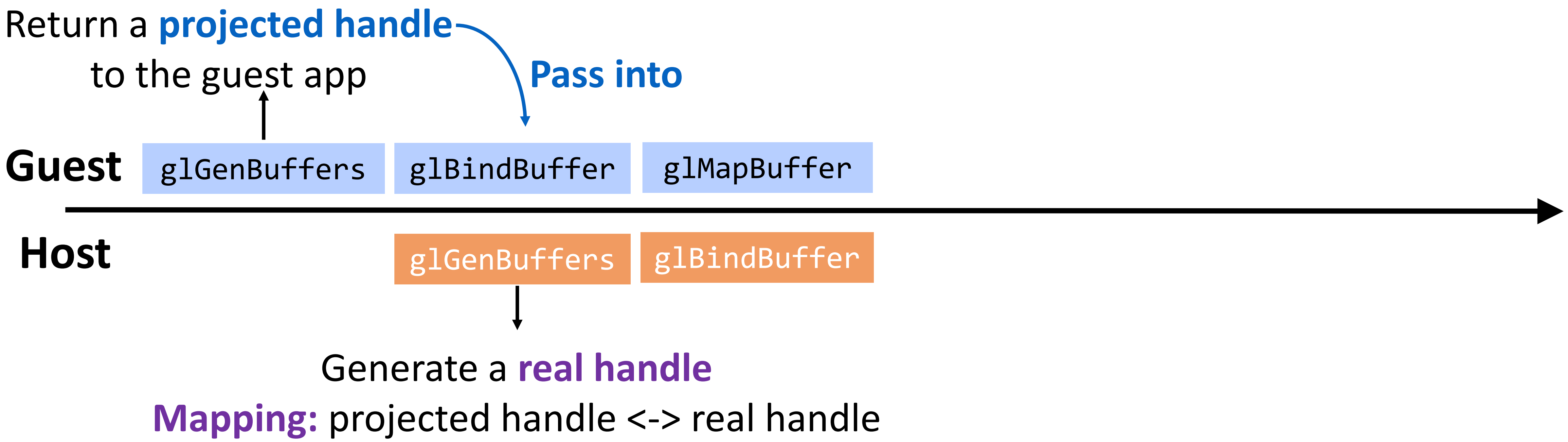


Draw a triangle with graphics projection

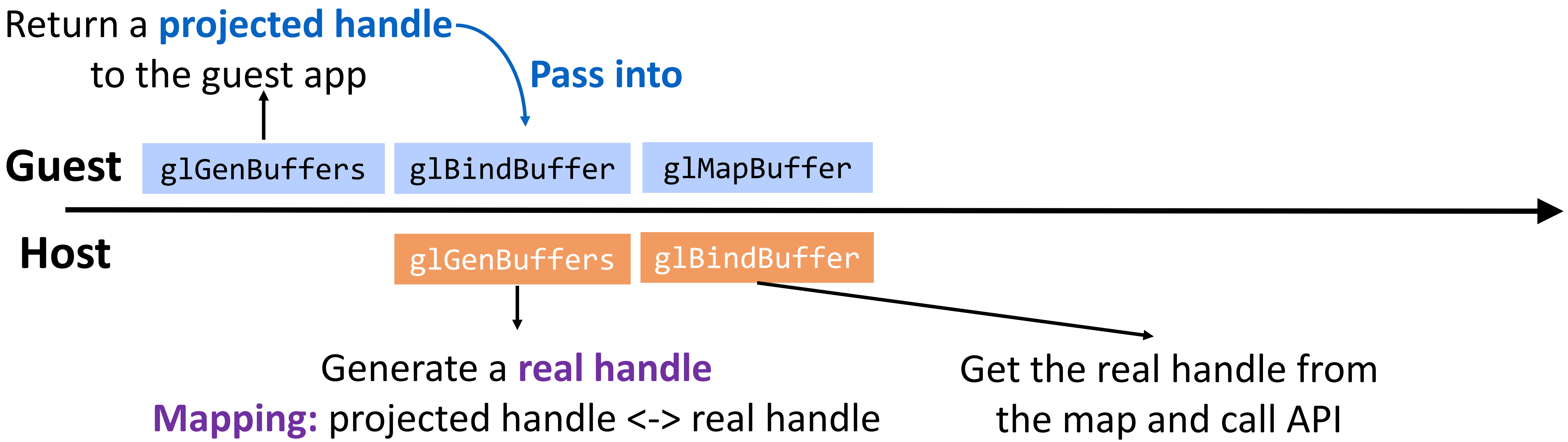
Draw a triangle with graphics projection



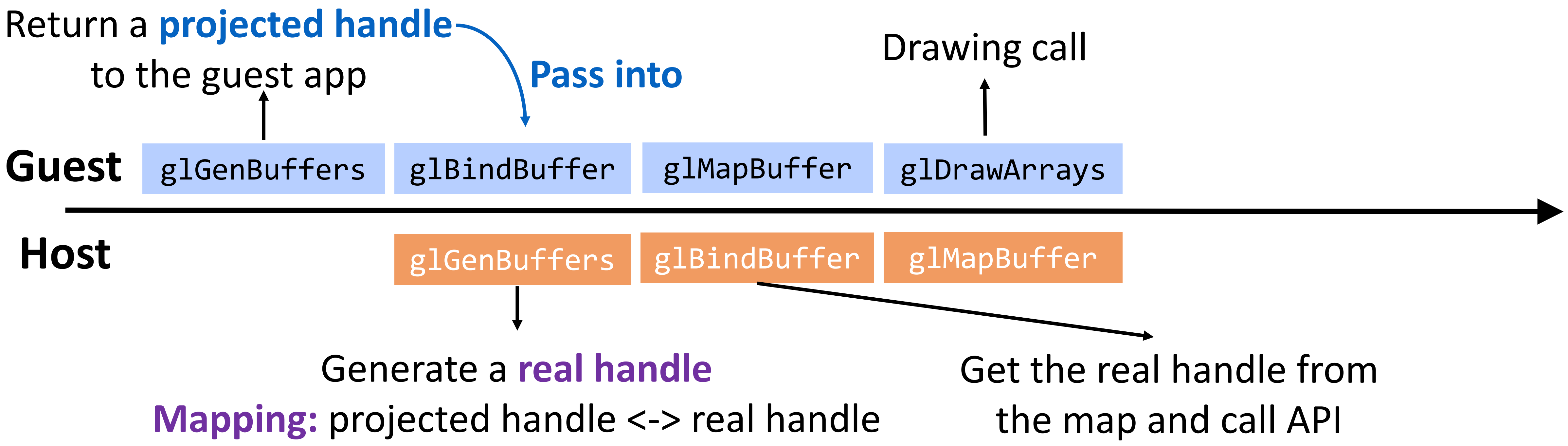
Draw a triangle with graphics projection



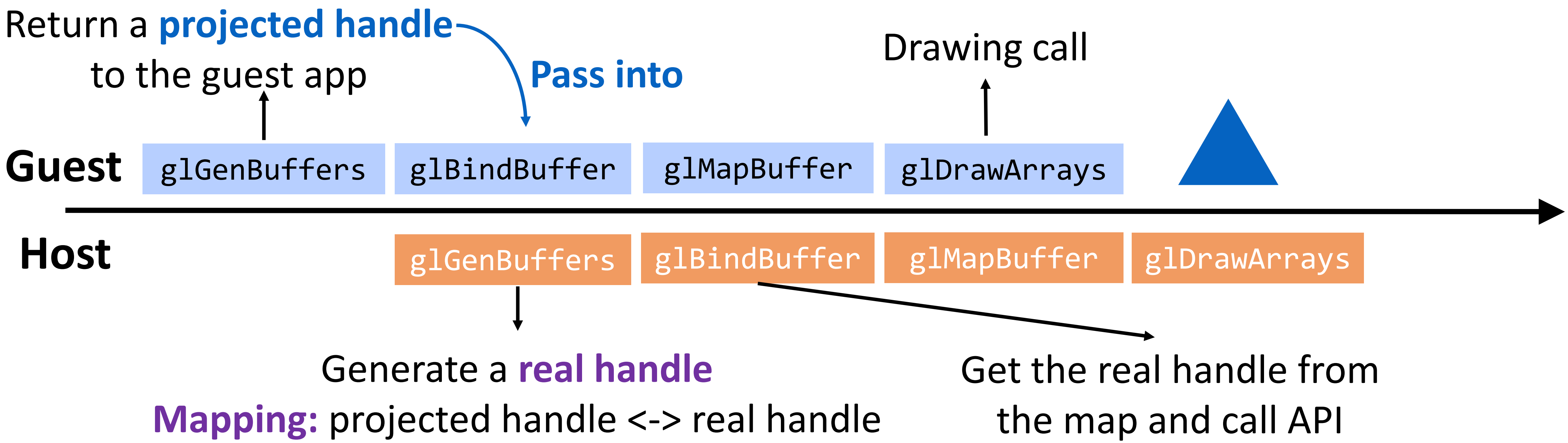
Draw a triangle with graphics projection



Draw a triangle with graphics projection

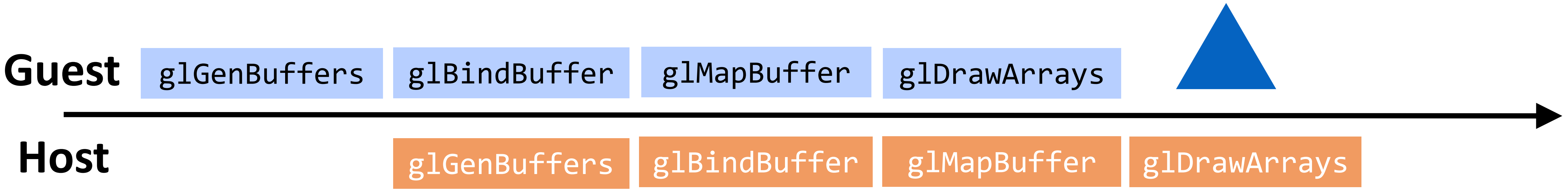


Draw a triangle with graphics projection

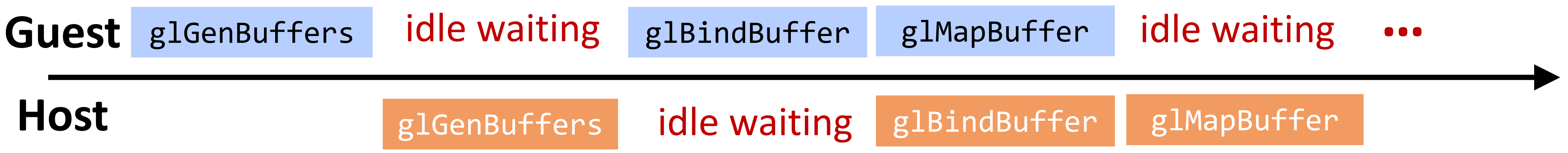


Draw a triangle with graphics projection

Graphics projection’s timeline



API remoting’s timeline



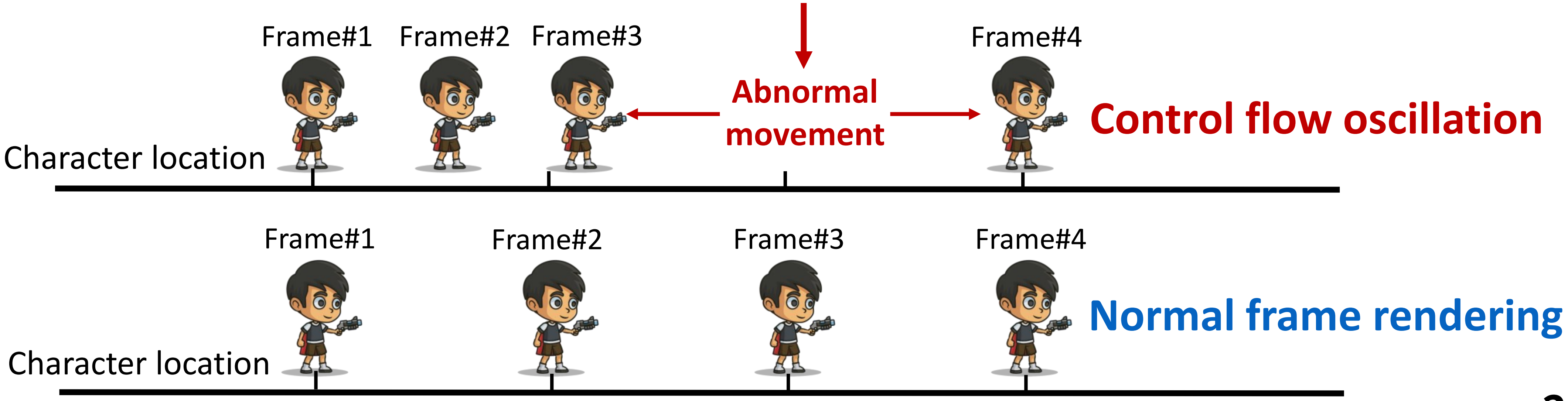
Effectiveness of projection space

- **99.93% API calls** do not need sync host-side executions
  - Only 41.4% do not need sync execution in API remoting
- **26% API calls** are **directly resolved** at the projection space
  - Mostly context/resource read APIs
- **<1 MB memory cost** for even a graphics-intensive app’s projection

Control flow oscillation

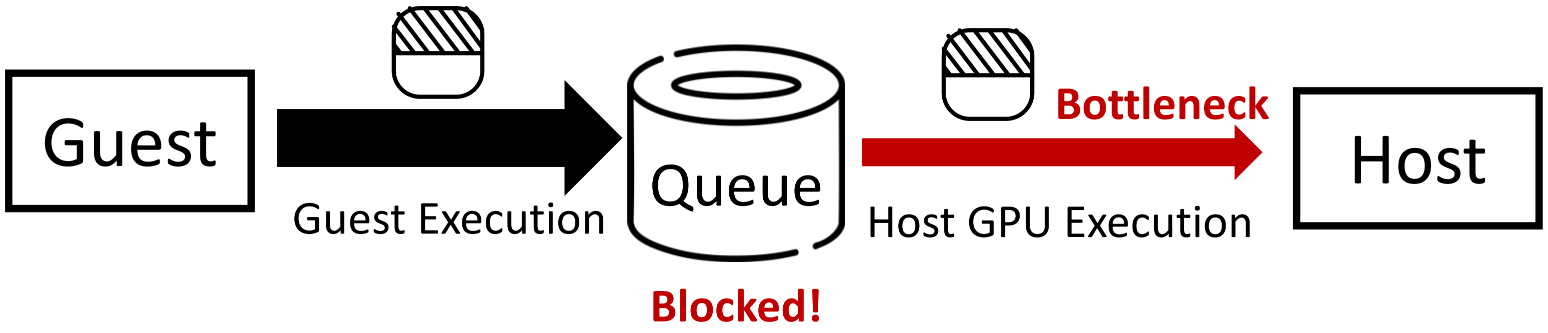
- Lightweight guest processing is **fast** at first, and then is **blocked**
- **Frame rendering time** is **short** at first, but then becomes **very long**

Character movement = **frame rendering time** × moving speed



Elastic flow control

- Key insight: guest control flow blocking can be modeled as network congestion



- Idea: adapt the **multiplicative**-increase/multiplicative-decrease (MIMD) congestion control algorithm of networking

- **Multiplicatively** adjust guest sleep time after a frame is rendered

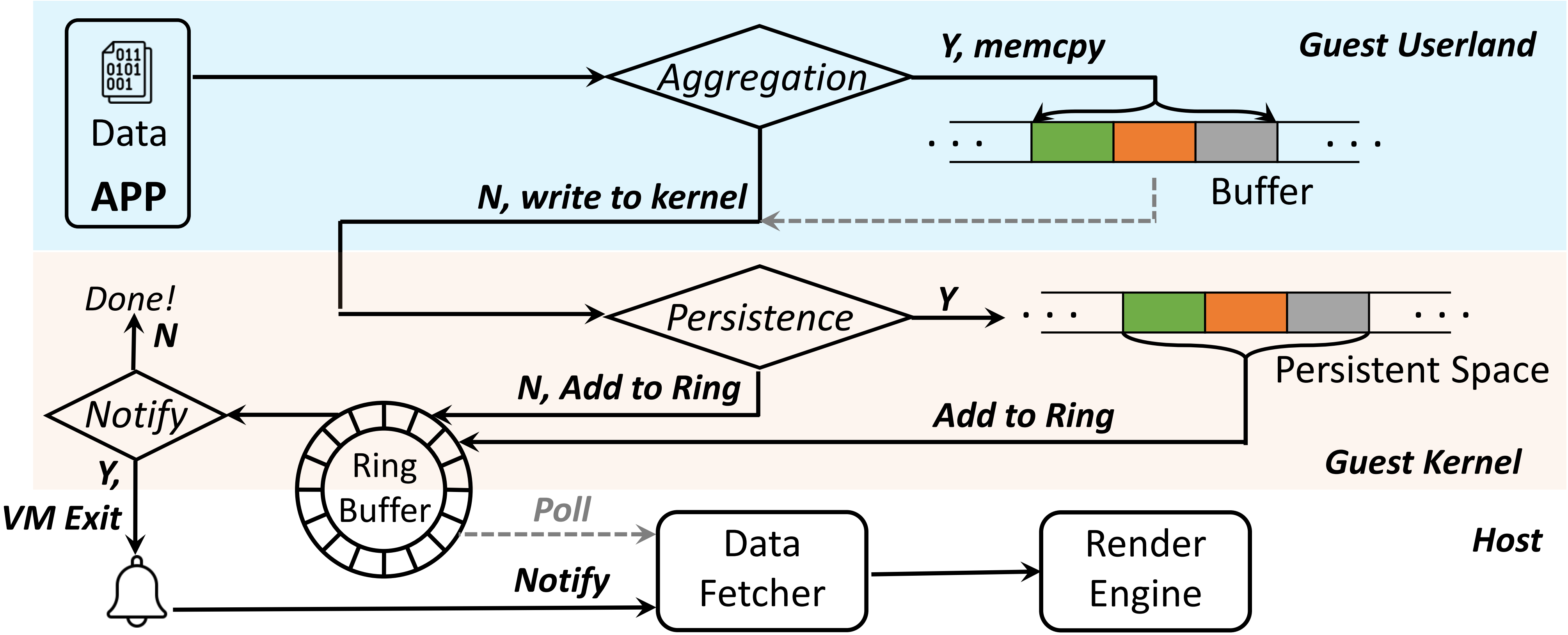
Unsmooth data flow delivery

- Delivering data under **highly dynamic situations** is challenging
  - **System dynamics**: CPU usage and available memory bandwidth
  - **Data dynamics**: a popular 3D app can generate up to **1 GB** graphics data per second, but the data generation rate is **<1 MB/s** in most cases

- No single strategy fits all dynamic situations!
  - E.g., a memcpy incurs copy delay, but is useful in batching calls
  - Copy delay ≈ data size / memory bandwidth

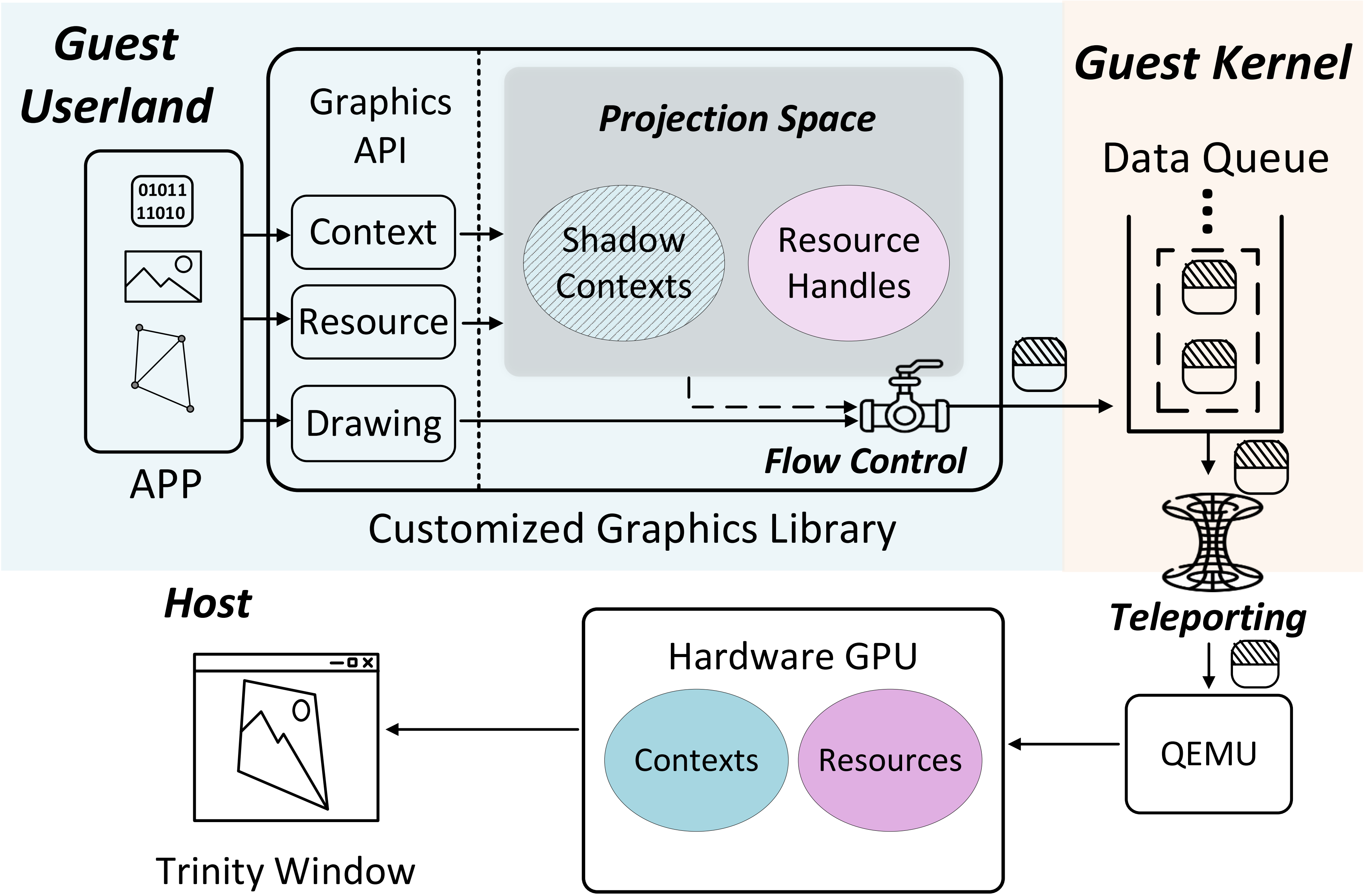
Adaptive data teleporting

- Decompose data delivery into three stages
- Estimate every strategy's delay using in-situ system and data status



✓ **Data delivery throughput is 5.3x larger than Google Android Emulator**

Trinity: high-performance Android emulator

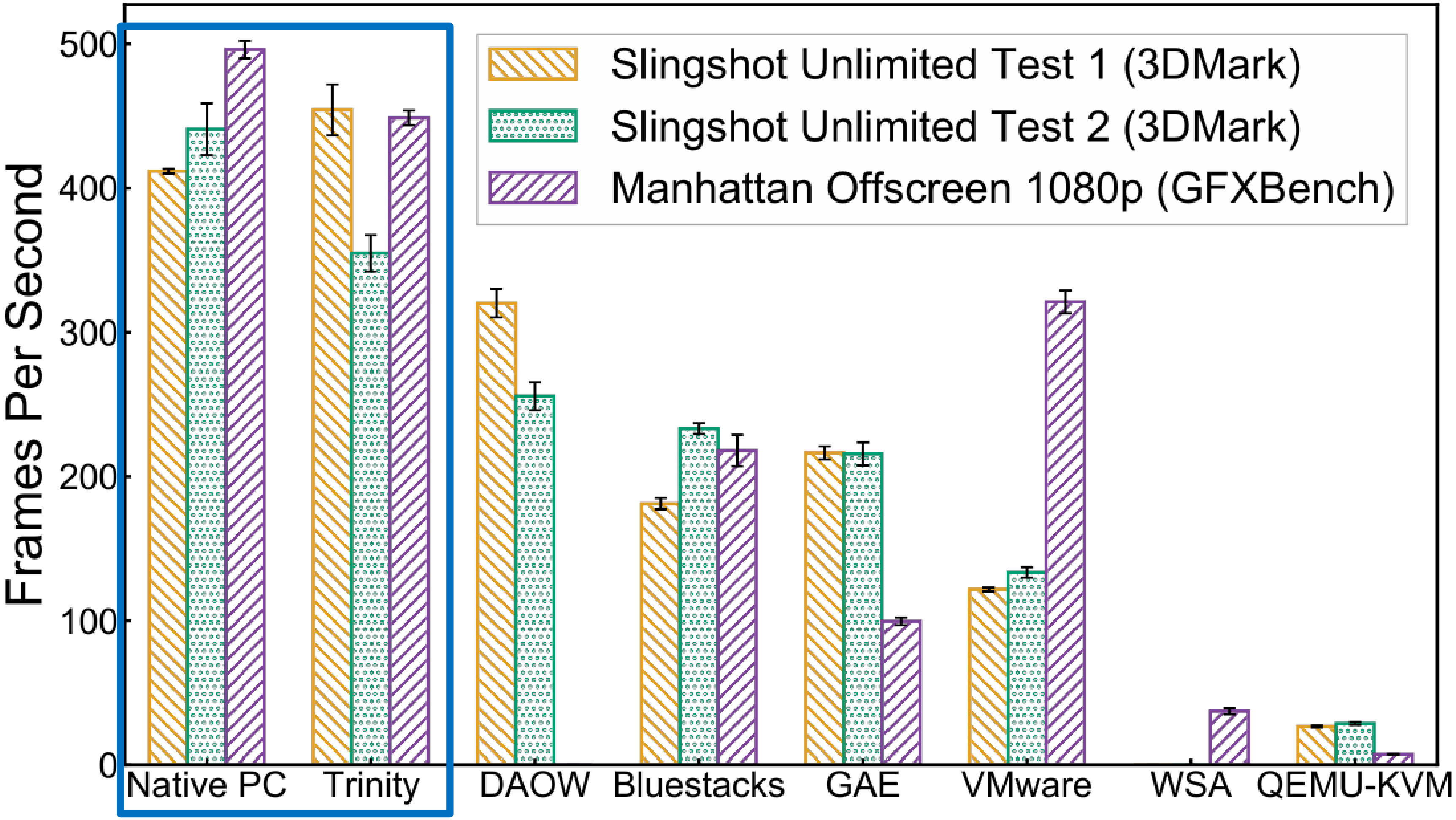


Evaluation

- Evaluate the extreme efficiency using **standard benchmarks**
- Evaluate the efficiency of running **top-100 apps** from Google play
- Evaluate compatibility with **random 10K apps** from Google Play

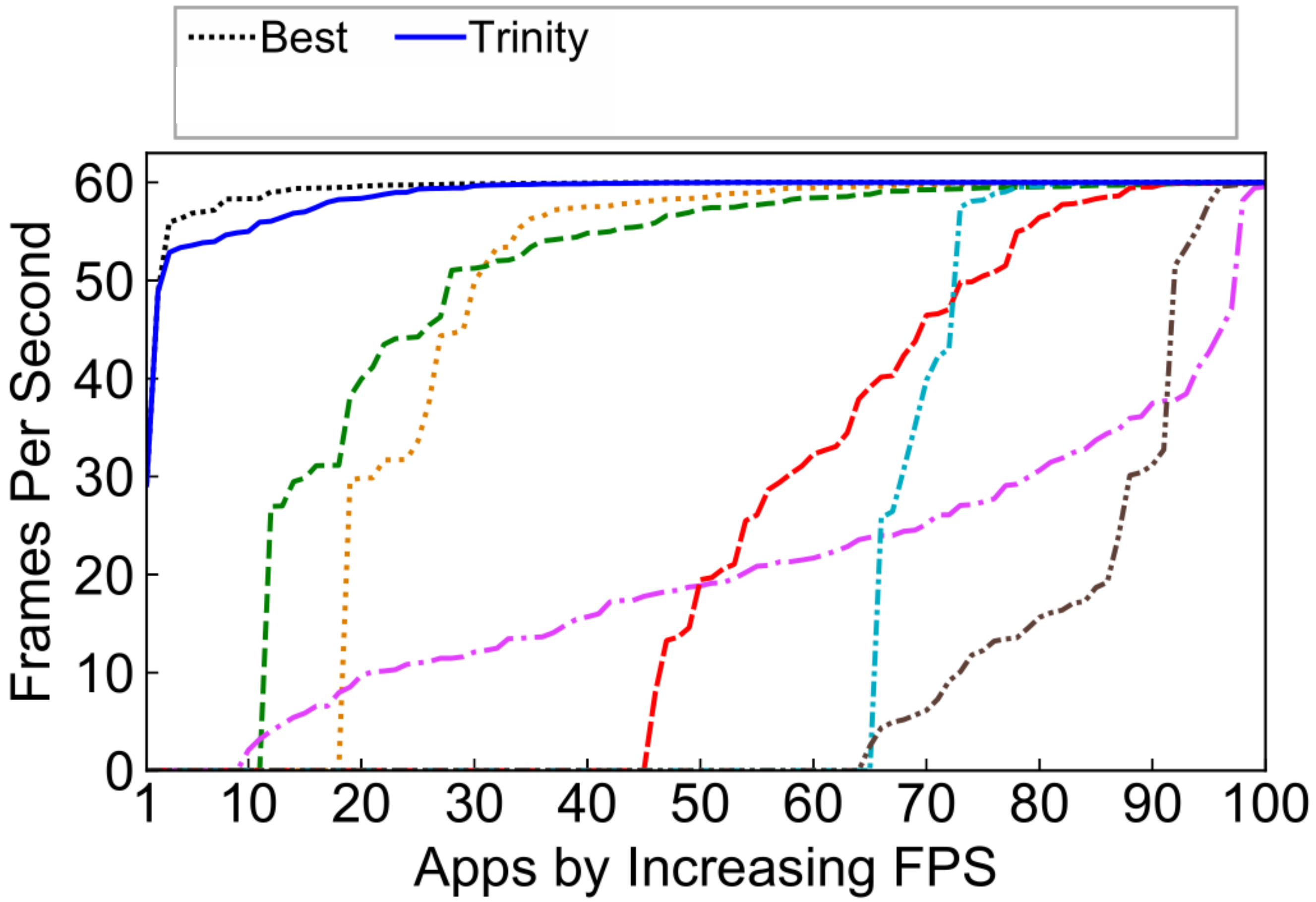
Standard graphics benchmarks

Trinity achieves an average of **93.3% (up to 110%) native hardware performance**



Top-100 3D apps

- Highest efficiency in **76 apps**
- For the other 24, there is **no perceivable (<6 FPS) difference** between Trinity and the emulator yielding the highest FPS
- Can smoothly run all apps



Random 10K apps

- Compatible with **97.2%** of the apps (no crash with random input)
    - 0.07% actively evade emulators
    - 0.43% require special hardware
    - 2.3% even crash on real devices

Conclusion

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

- A highly-efficient graphics virtualization method called **graphics projection**
- Elastic **flow control** and adaptive **data teleporting** mechanisms for matching the decoupled guest/host graphics processing rates
- The first mobile emulator that can smoothly run heavy 3D apps without losing compatibility or security
- https://TrinityEmulator.github.io/