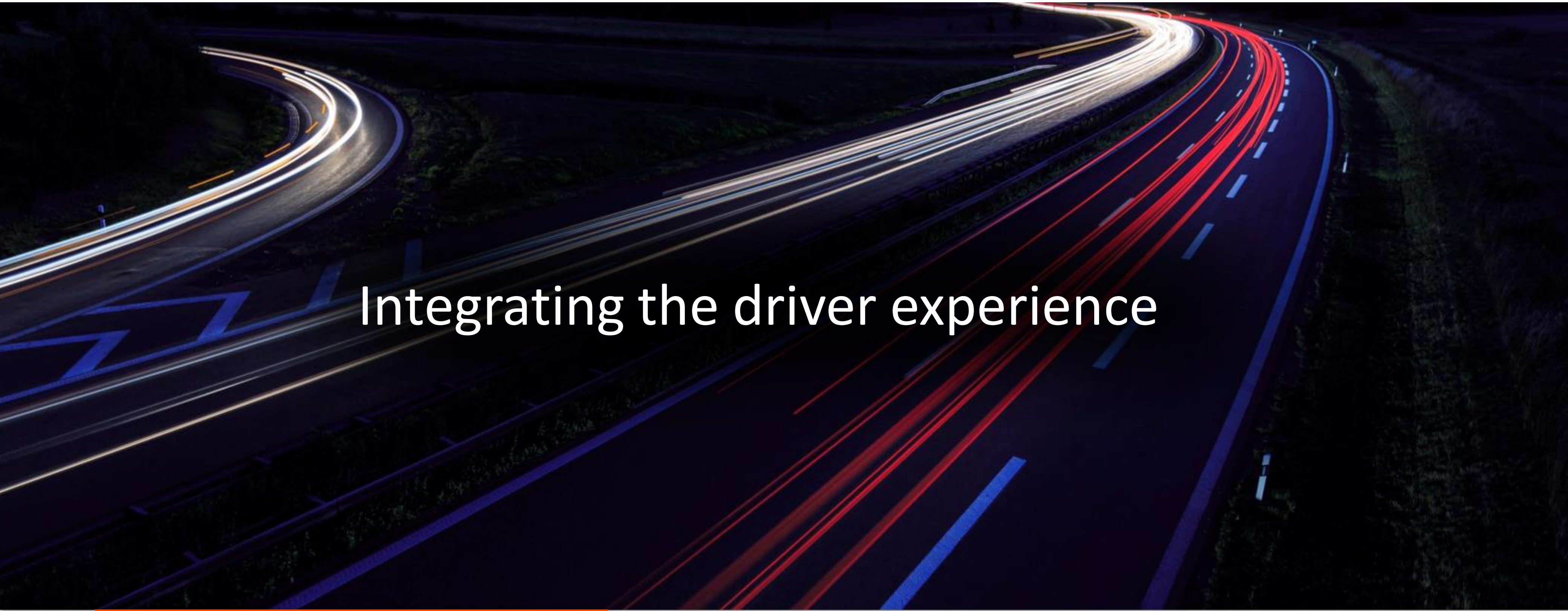
OPENSYNERGY



Hypervisor Return of experience Session

Genivi AMM Munich 2019

public

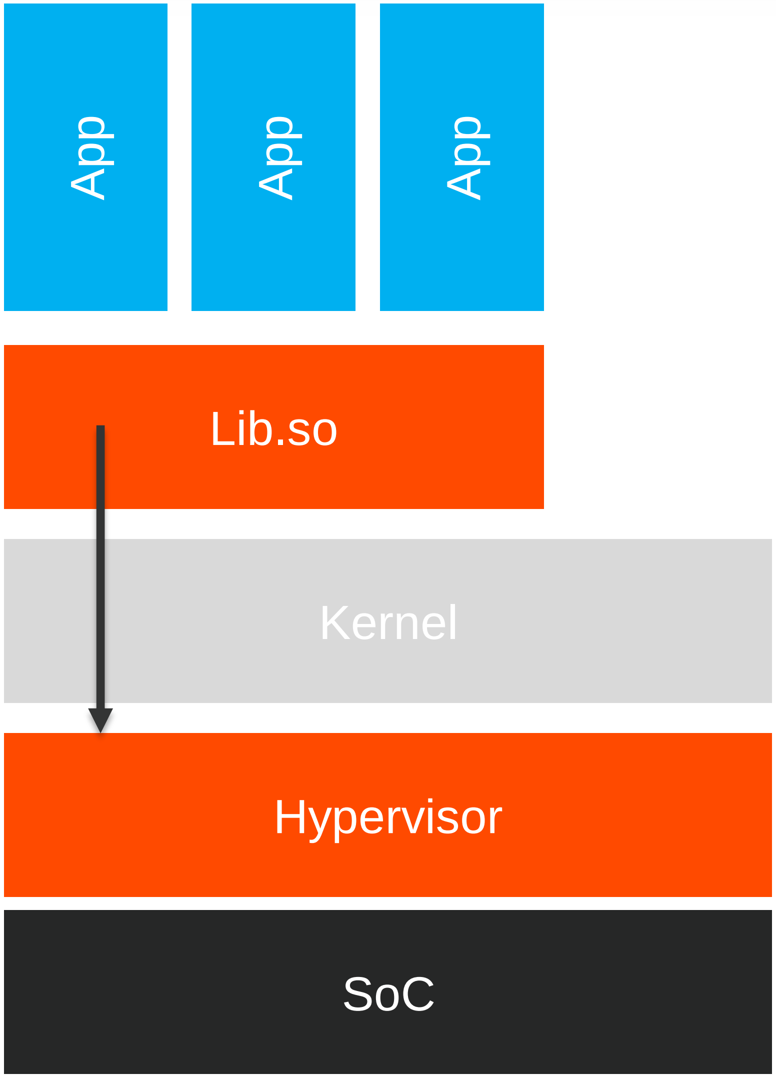
How **Not** to ~~Share~~ Virtualize Devices

1. Do Not Replace User Space Libraries

OPENSYNERGY

- Don’t Replace an existing user space library with a hypervisor specific one
- Typical attempts
  - libdrm.so
  - libteec.so (trustzone access)
- The kernel is bypassed, therefore
  - User space resources need to be managed in hypervisor
  - App lifecycle management leaks into hypervisor

Better: Implement drivers using driver frameworks provided by operating system

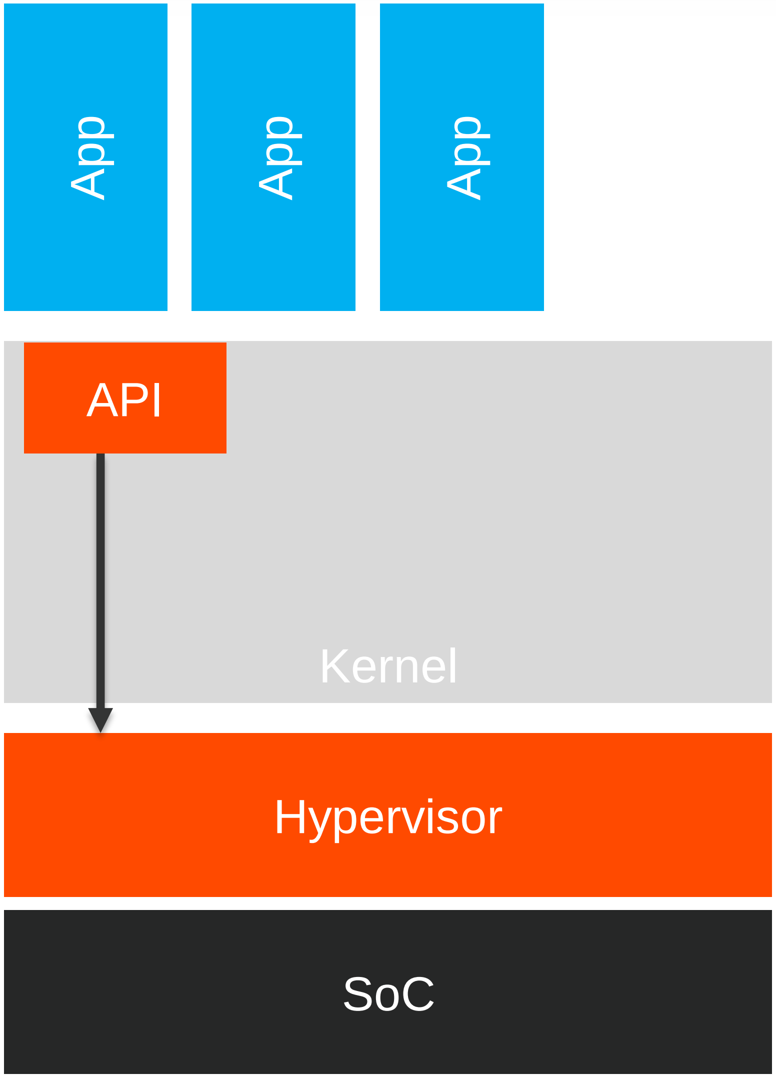


2. Do Not Replicate Kernel Subsystem APIs

OPENSYNERGY

- Don’t Re-implement an operating system API
  - Same operating system handle reference as original device
  - IOCTLs
  - Read/Write Functions
- Typical approach
  - Drm/kms
  - Fbdev
  - GPIOfs
- Complete implementations of “shim” layer end up being really bad re-implementations of existing subsystems

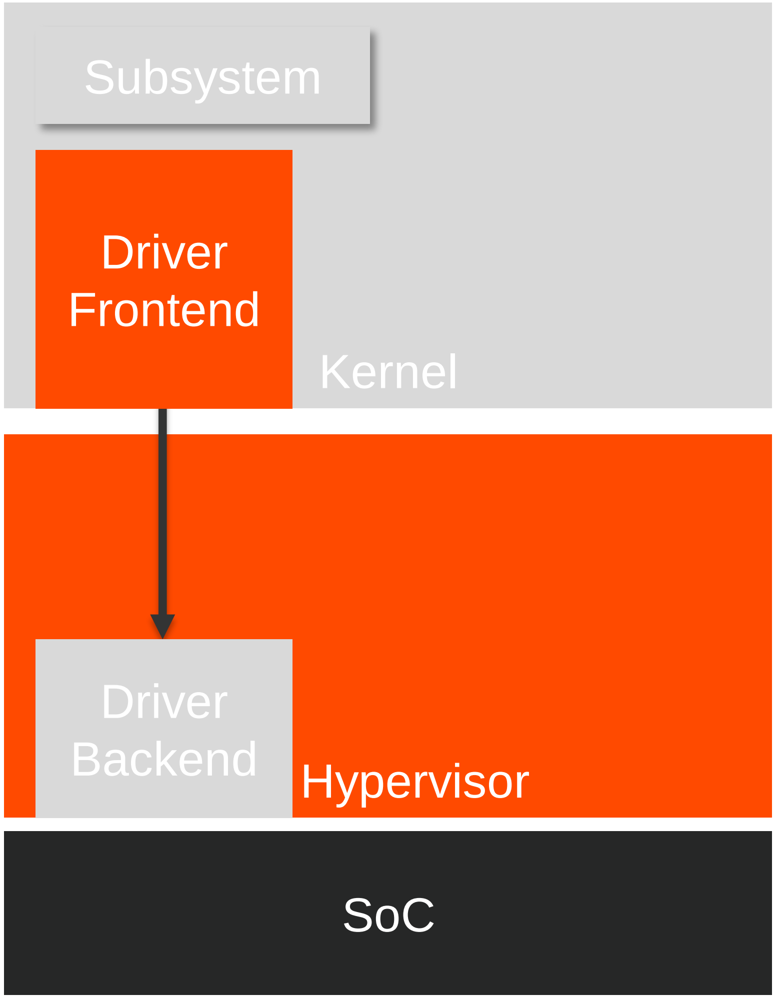
Better: Implement drivers using driver frameworks provided by operating system



3. Do Not Split a device driver

- Don’t Take **existing** device driver and cut it in half
  - Leave upper half in guest kernel
  - Move lower half into hypervisor
- Can be very efficient
  - Possibly low overhead
  - Looks very simple
  - Many drivers have mid-layers where a cut is easy
- Typical approach
  - GPU drivers
- Most mid-layers are very leaky
- Modification of existing driver cuts off update paths
- Very hardware specific
- The devil in the details, usually takes multiple times longer than planned

Better: Find good functional abstractions and implemented SoC agnostic drivers



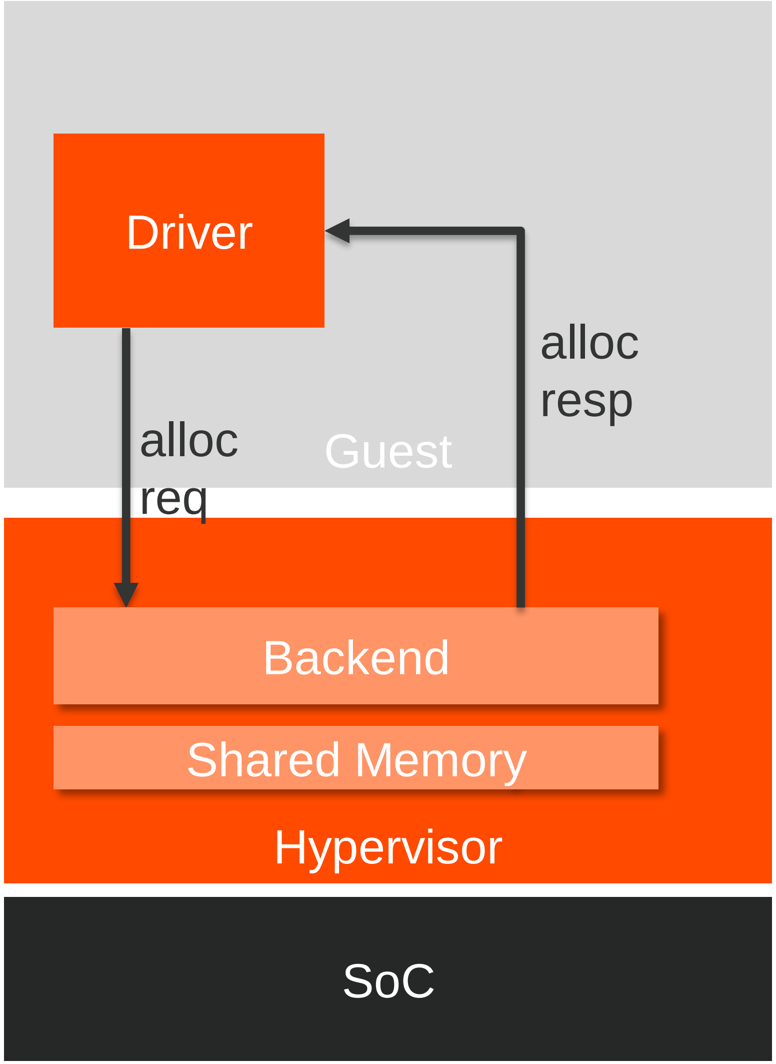
4. Avoid Host side allocations

Problem

- Guest needs to allocate “special” memory
  - Cache coherent
  - Device accessible (<4gig)

Naïve solution

- Remote memory allocation



- Bookkeeping is distributed
- Memory owner might have different lifecycle than Memory manager
- To avoid memory starvation
  - Pooled allocations -> memory waste

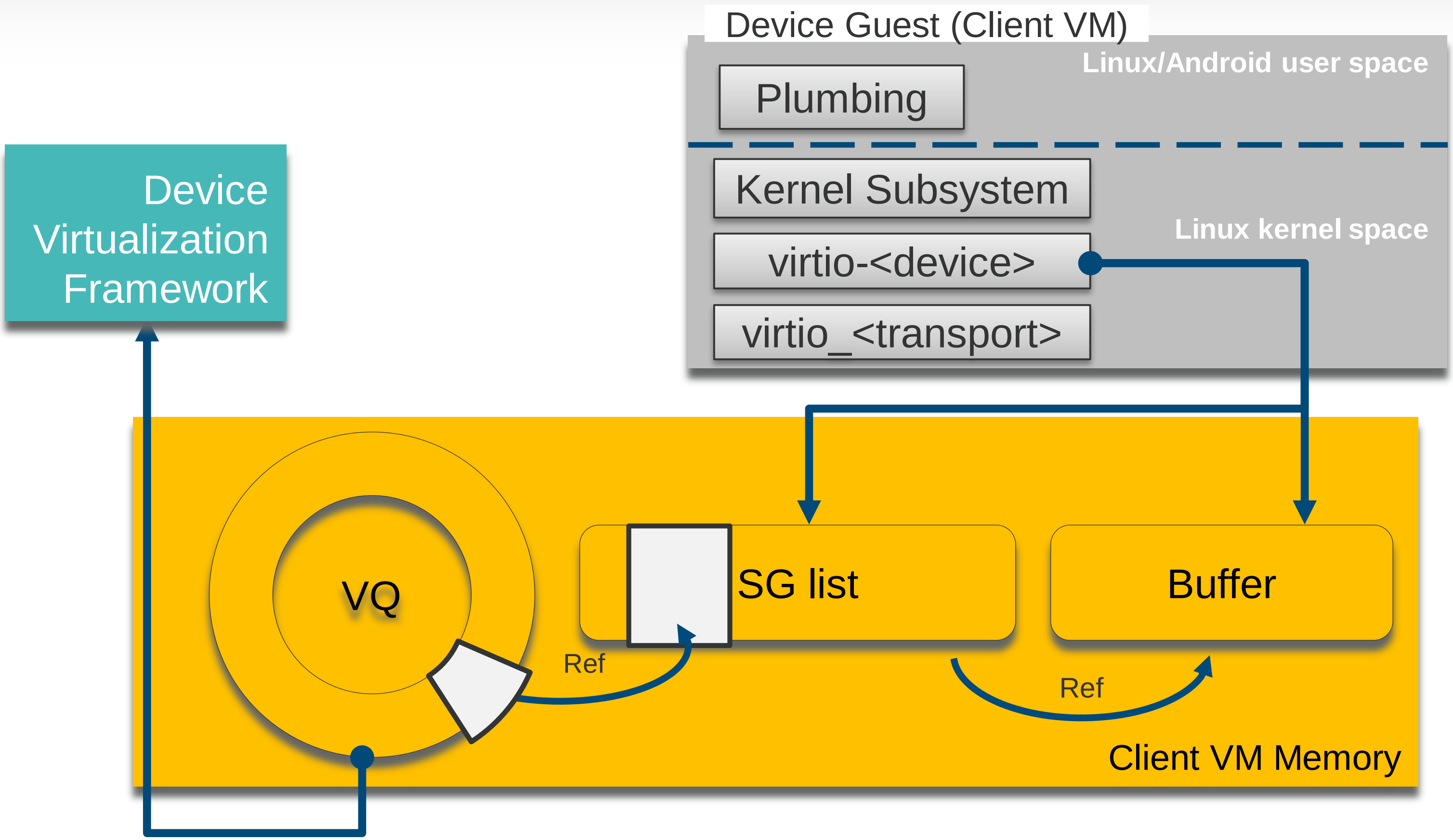
Better: Import guest buffers, teach guest to manage buffers himself, use iommus

OPENSYNERGY

Virtualize device functions – not devices!

Virtualized device Architecture with VIRTIO

OPENSYNERGY



- Drivers in Kernel
- Uses existing subsystems
- Only abstract device functions
- All allocations in guest memory

Bulk data transport via DMA-like memory model

- Buffer **allocations** handled by „Driver“ part (client)
- **Direct** R/W access to allocated buffers in the „Device“ part (server)

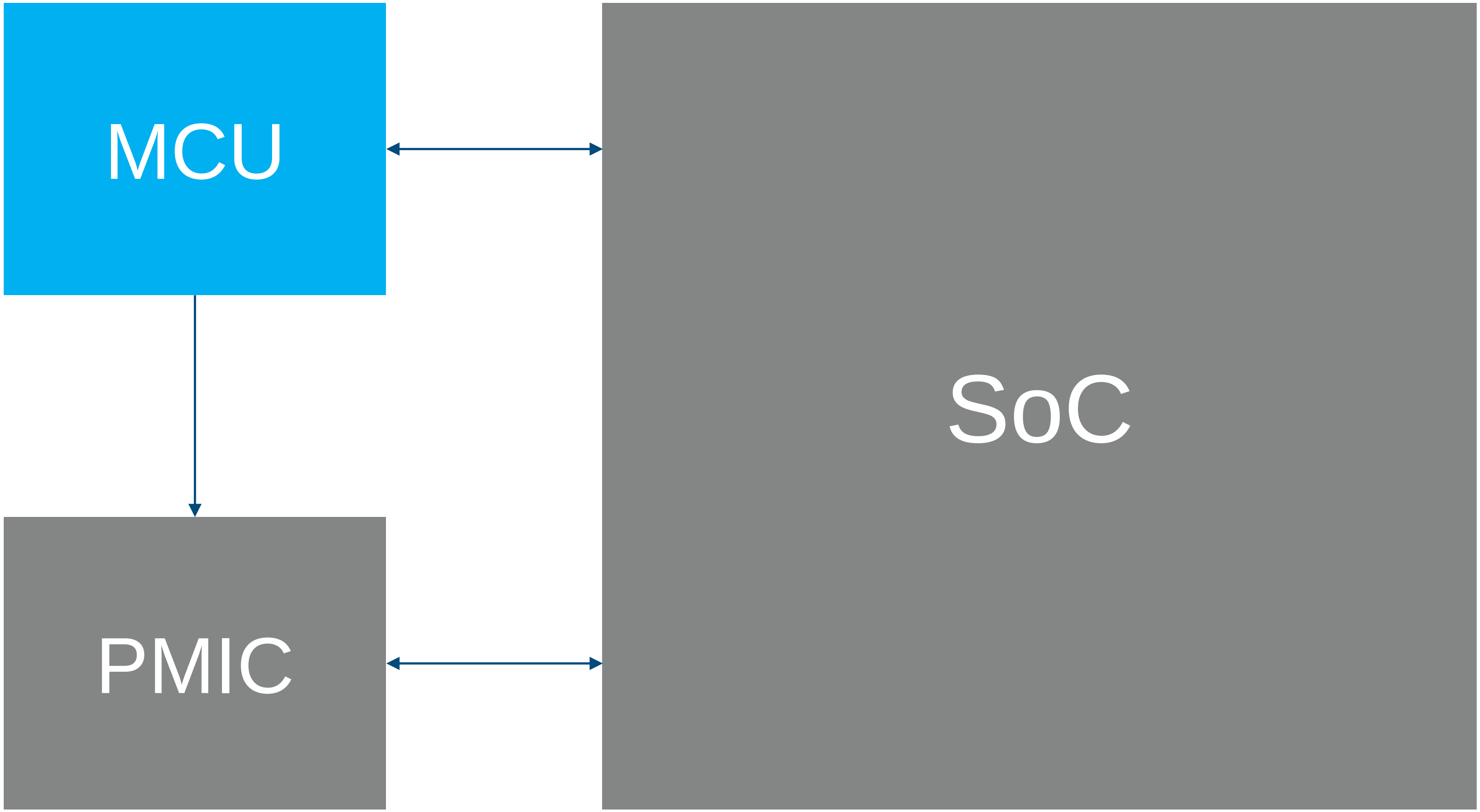
Metadata transport via virt-queues (ring buffers, asynchronous pipeline)

VQ=virt-queue
SG=Scatter Gather

System Management Architecture

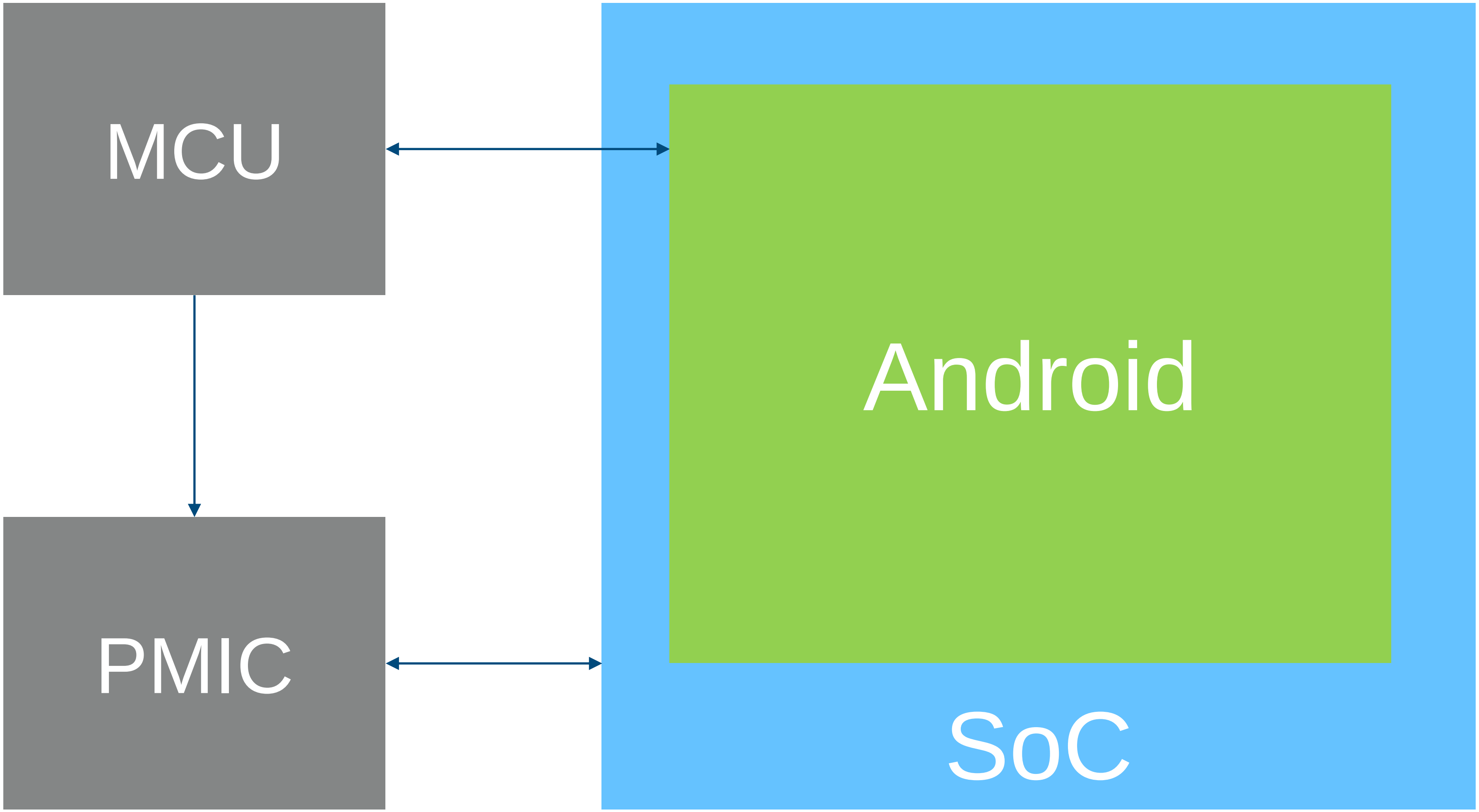
Classical Design (Example)

OPENSYNERGY



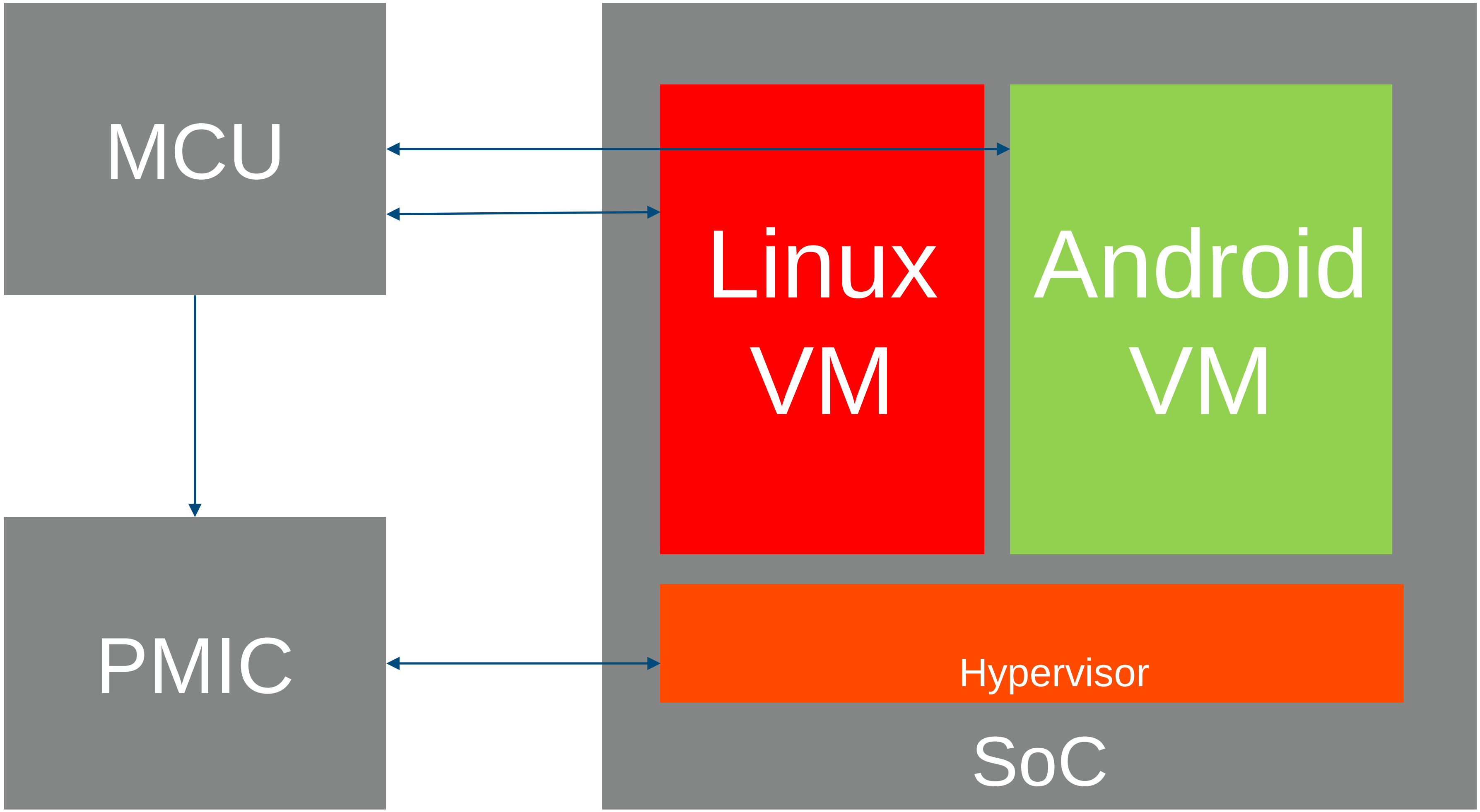
The MCU controls the systems state, using its own state machine

Android Design Assumption (example)



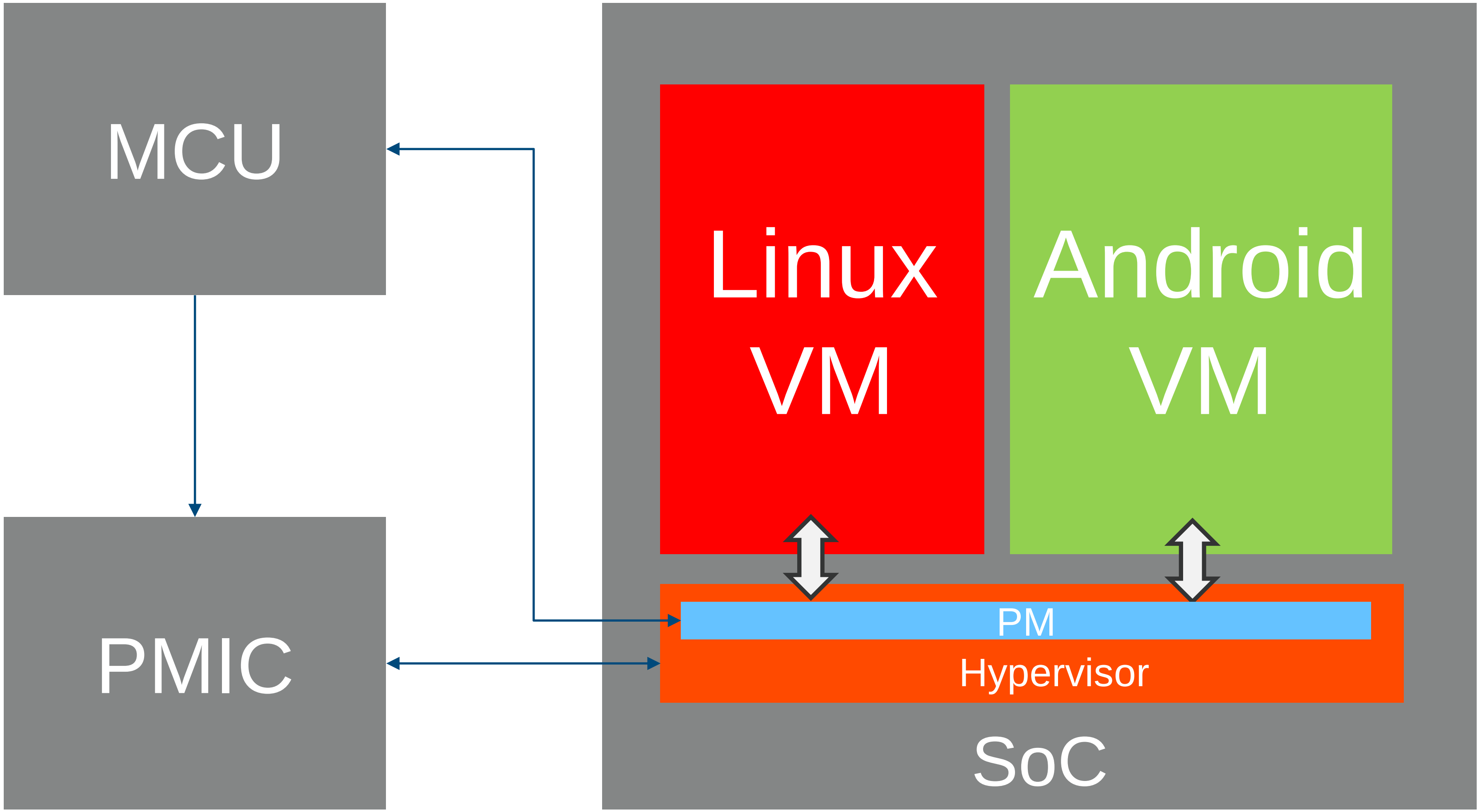
The MCU is demoted, Android is the new master of the state

Example system (Android+Linux)

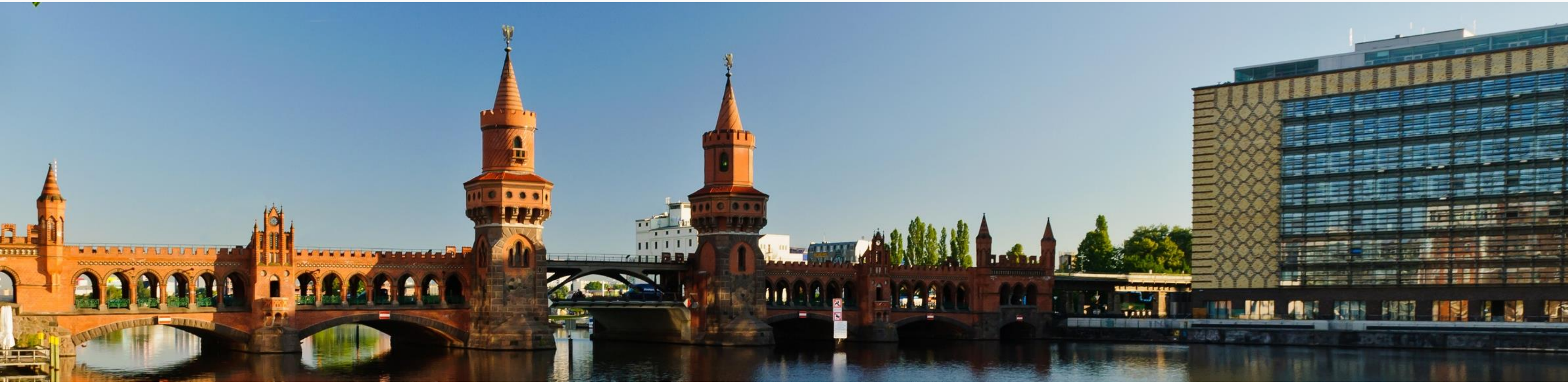


Who is the new master?

Example system (Android+Linux)



Central decision point in hypervisor, but guests “believe” they are in charge



Headquarter

Further Locations

Berlin

OpenSynergy GmbH

Rotherstraße 20
D-10245 Berlin
Germany
Phone +49 30 / 6098 5400

Munich

OpenSynergy GmbH

Starnberger Str. 22
D-82131 Gauting / Munich
Germany
Phone: +49 89 / 2153 9073

Utah

OpenSynergy, Inc. (USA)

765 East 340 South
Suite 106
American Fork, Utah 84003
USA

California

OpenSynergy, Inc. (USA)

501 W. Broadway, Suite
832
San Diego, California 92101
USA
Phone +1 619 962 1725

Shanghai

OpenSynergy GmbH

2608, Enterprise Square
228 Mei Yuan Road
Shanghai
P.R. of China
Phone: +86 132 6277 7738

E-Mail info@opensynergy.com
Web www.opensynergy.com