ACCESS Linnaeus Centre

ACCESS the future

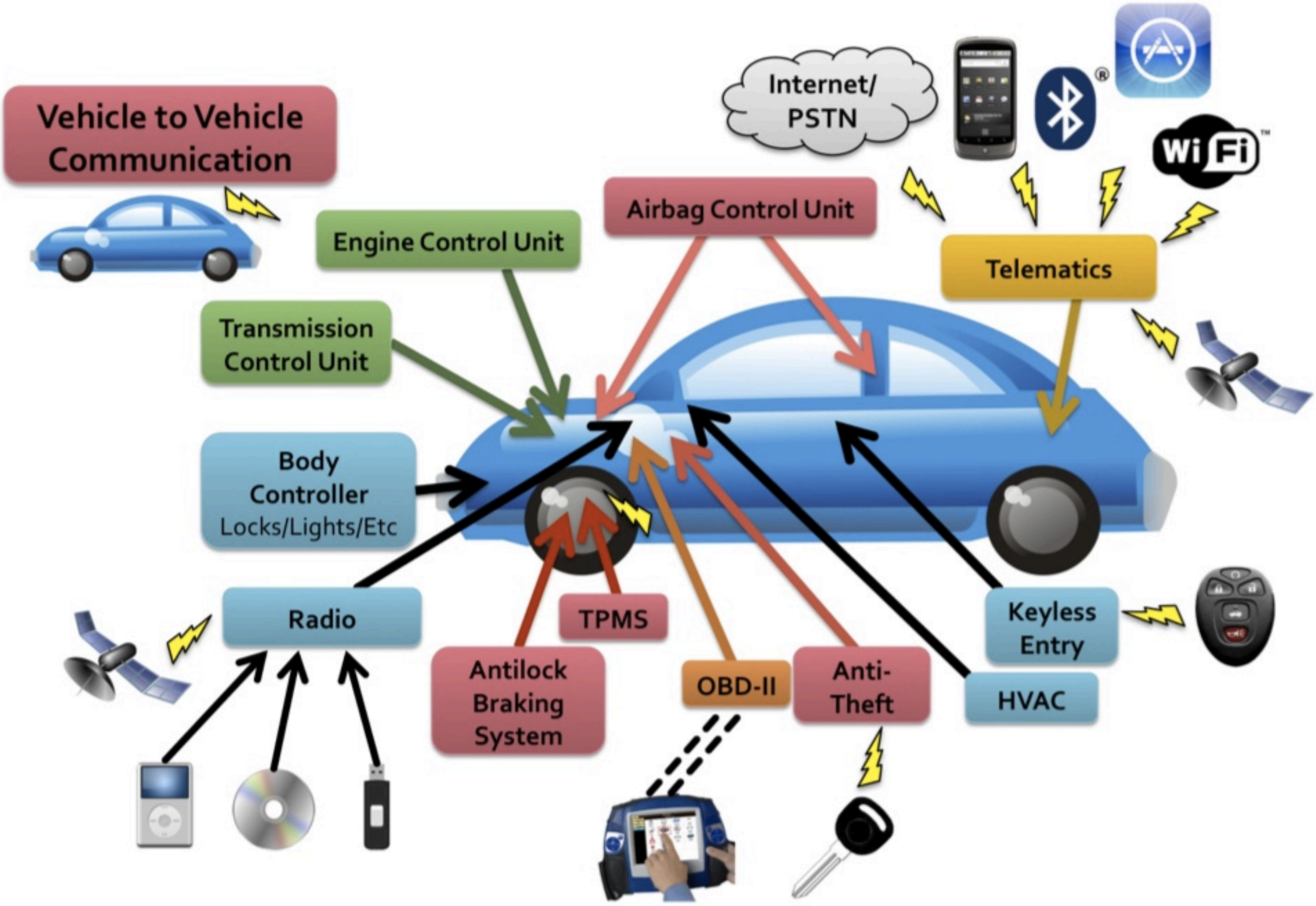
Provably Secure Execution Platforms for Embedded Systems

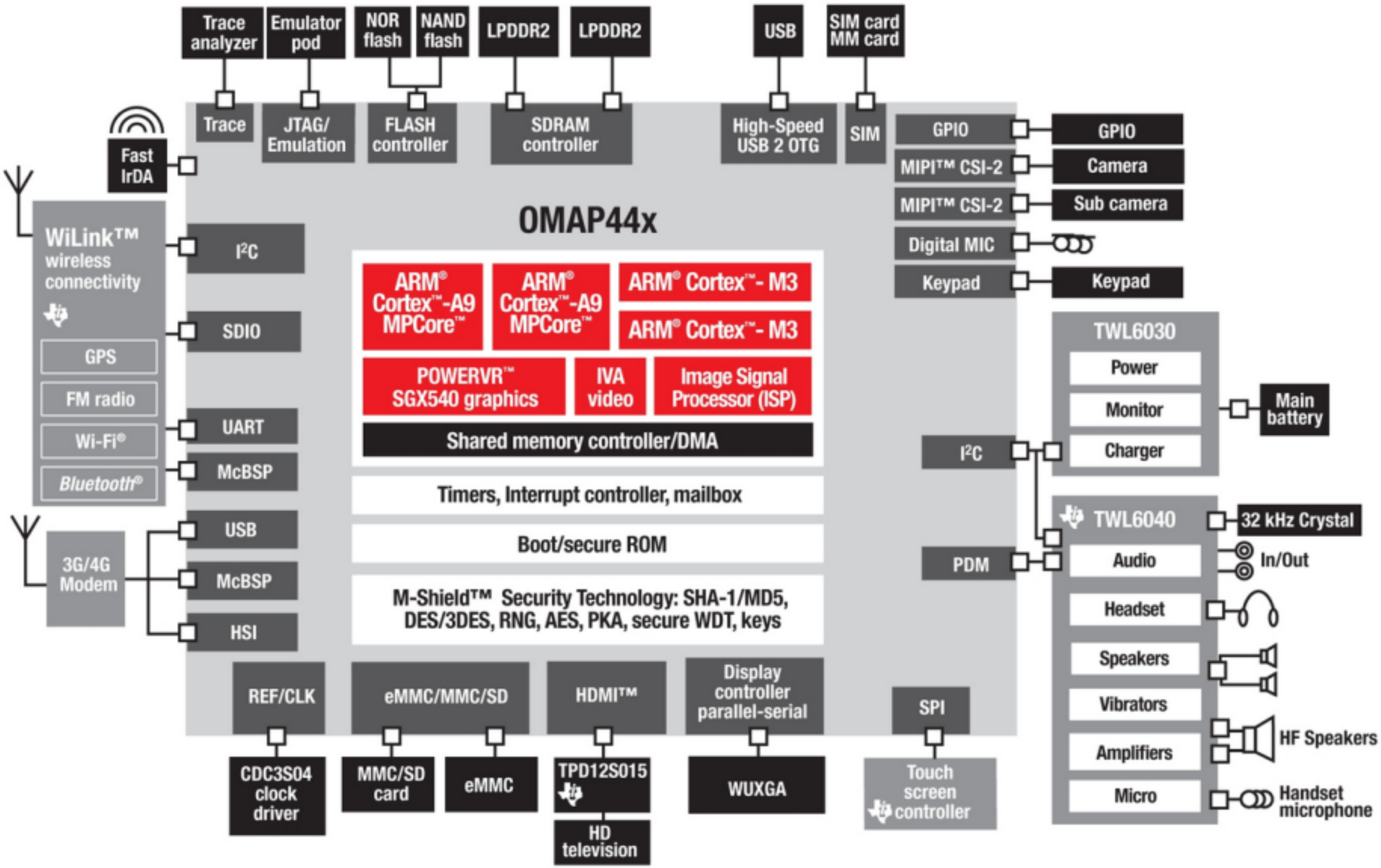
Mads Dam
TCS
School of Computer Science and Communication

Joint work with colleagues from SICS and KTH

Access Industry Workshop, 24 Jan 2013

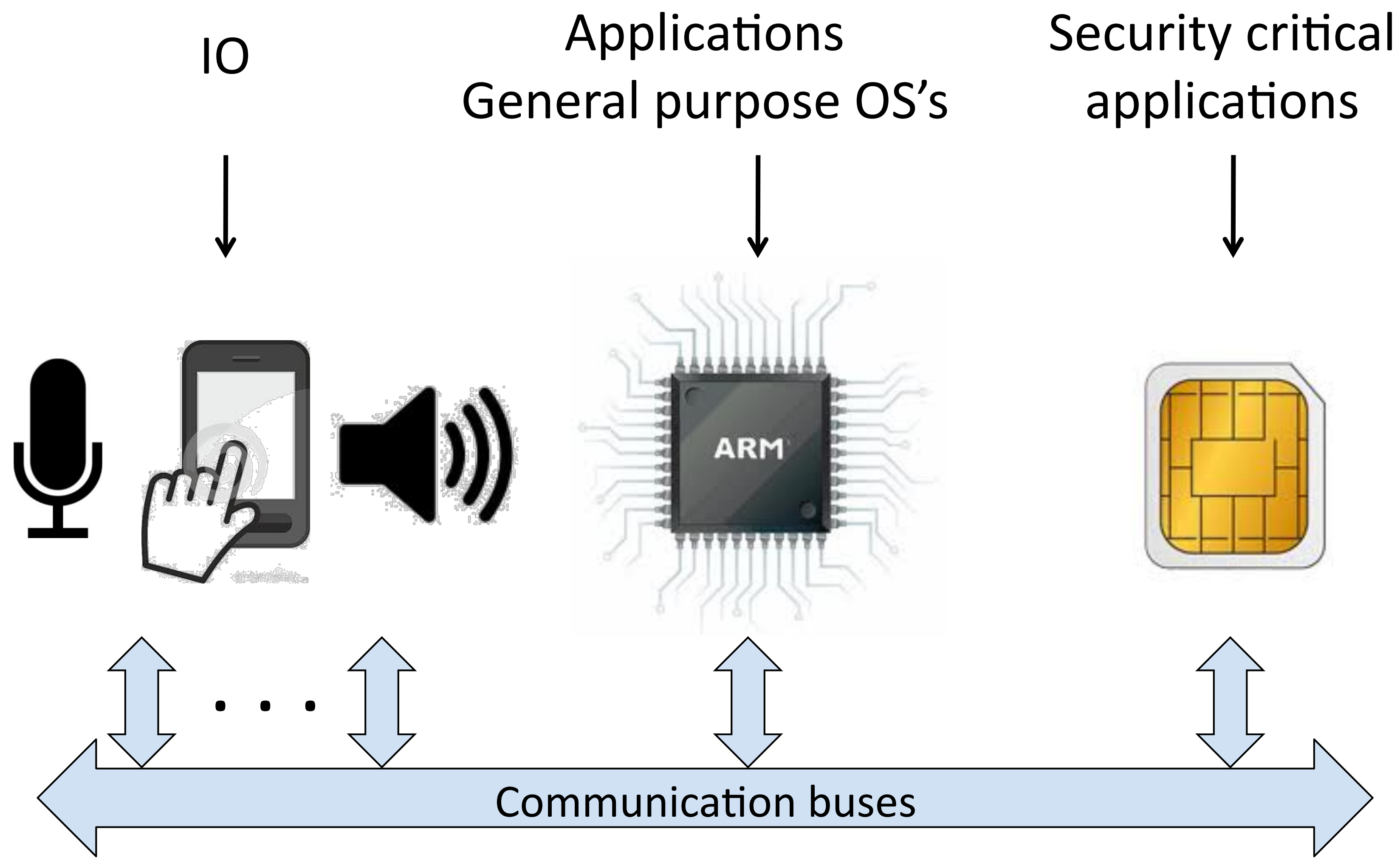
PROSPER


Vehicle to Vehicle Communication
Internet/ PSTN
Engine Control Unit
Airbag Control Unit
Telematics
Transmission Control Unit
Body Controller
Locks/Lights/Etc
Radio
TPMS
Antilock Braking System
OBD-II
Anti-Theft
Keyless Entry
HVAC

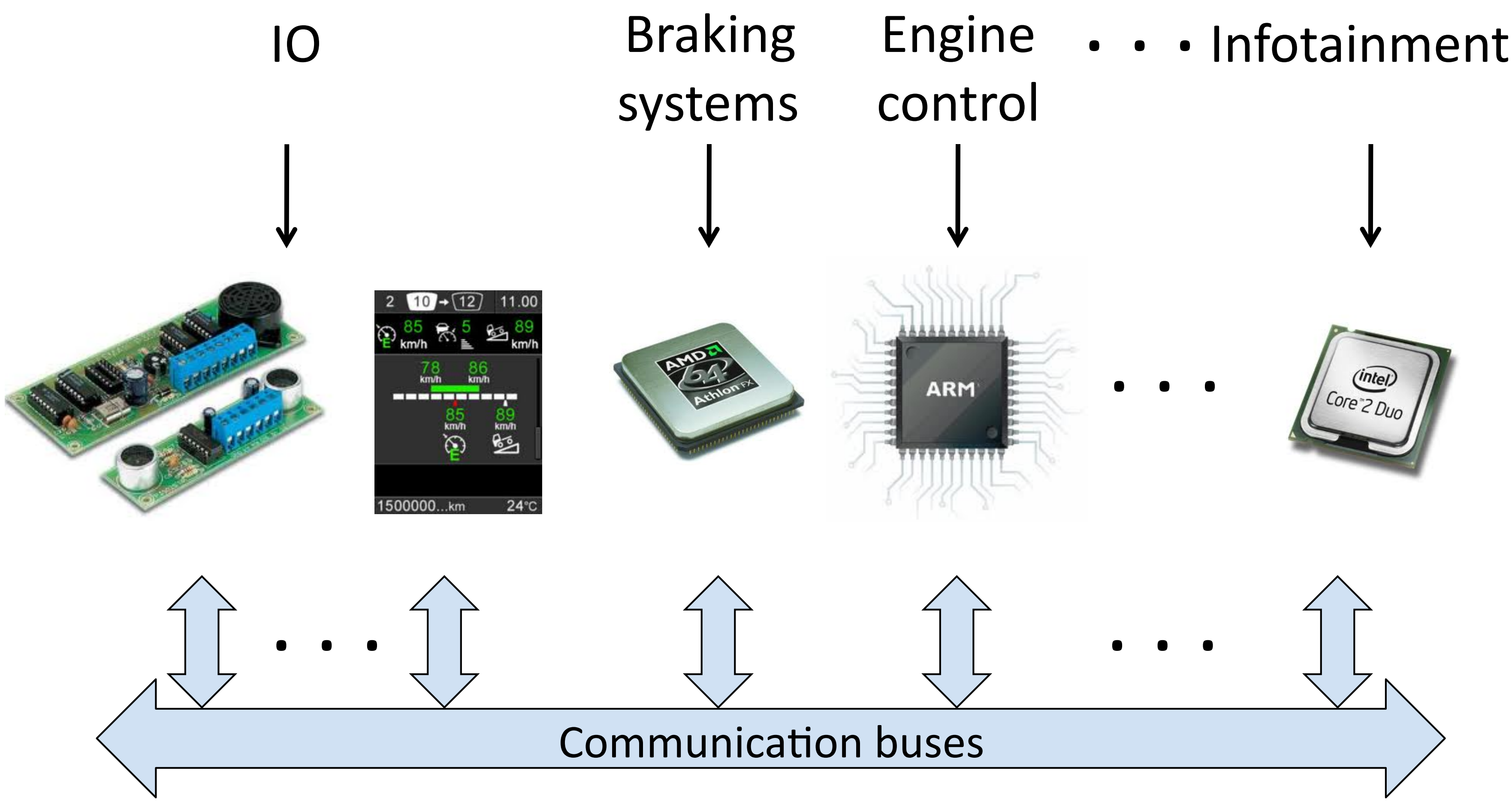


Trace analyzer
Emulator pod
NOR flash
NAND flash
LPDDR2
LPDDR2
USB
SIM card MM card
Trace
JTAG/ Emulation
FLASH controller
SDRAM controller
High-Speed USB 2 OTG
SIM
Fast IrDA
GPIO
GPIO
MIPI™ CSI-2
Camera
MIPI™ CSI-2
Sub camera
Digital MIC
Keypad
Keypad
OMAP44x
WiLink™ wireless connectivity
GPS
FM radio
Wi-Fi®
Bluetooth®
I²C
SDIO
UART
McBSP
USB
McBSP
HSI
3G/4G Modem
ARM® Cortex™-A9 MPCore™
ARM® Cortex™-A9 MPCore™
ARM® Cortex™- M3
ARM® Cortex™- M3
POWERVR™ SGX540 graphics
IVA video
Image Signal Processor (ISP)
Shared memory controller/DMA
Timers, Interrupt controller, mailbox
Boot/secure ROM
M-Shield™ Security Technology: SHA-1/MD5, DES/3DES, RNG, AES, PKA, secure WDT, keys
TWL6030
Power
Monitor
Charger
Main battery
I²C
TWL6040
32 kHz Crystal
PDM
Audio
In/Out
Headset
Speakers
Vibrators
Amplifiers
HF Speakers
Micro
Handset microphone
REF/CLK
eMMC/MMC/SD
HDMI™
Display controller parallel-serial
SPI
CDC3S04 clock driver
MMC/SD card
eMMC
TPD12S015
HD television
WUXGA
Touch screen controller

A Little More Abstractly …



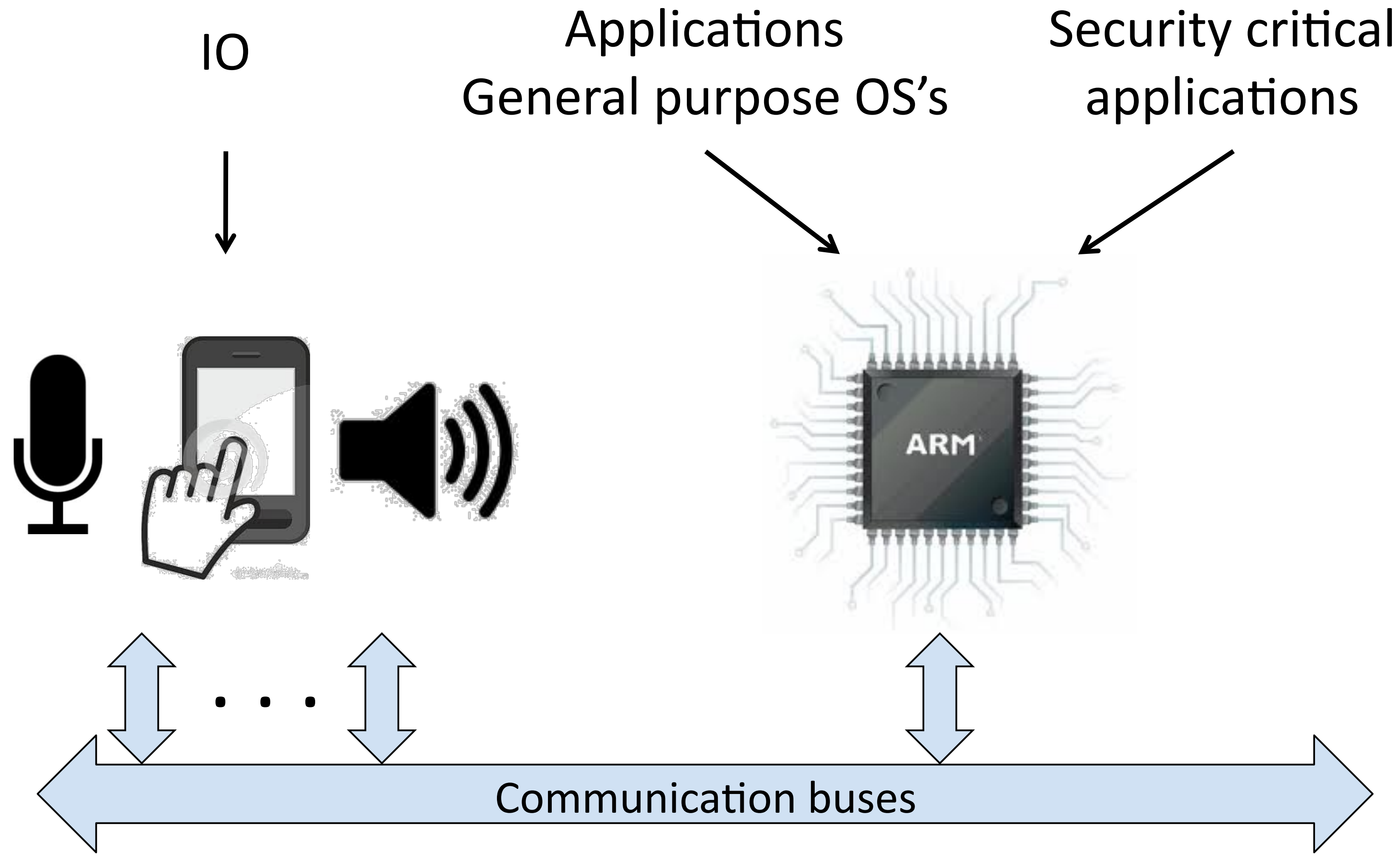
Or in Automotive …



The State of Affairs

- 1 security domain = 1 (or more) dedicated processors
  - Sharing the communication medium is bad enough
  - But we have techniques for that
- OS's are not to be trusted
  - So sharing the processors is not possible
  - At least for high robustness/high reliability applications
- But this is wasteful
  - Processor cycles, energy and materials consumption
  - Complexity, over-engineering, operation, maintenance

Secure Virtualization



Requirements

- Processor is *partitioned* into different *guest systems*

Applications General purpose OS's	Security critical applications

- Critical to avoid fault propagation and information leakage
- Isolation:
  - Each guest system executes as if in sole control of the processor
  - Communication not tampered with by the processor
- This is called a *separation kernel* !

The critical component for end-to-end system security

Our Goal

- Build a
  - formally specified
  - *fully verified* at *machine code level*
  - separation kernel
    - (or: Secure hypervisor)
  - for a *commodity smartphone processor*/SoC
    - ARMv7, ARM CortexA8
  - capable of supporting
    - commodity os
    - sim application
  - with guaranteed *strong isolation* properties

Related Work

seL4:

- Microkernel
- Verification as Haskell level
- Weak isolation guarantees

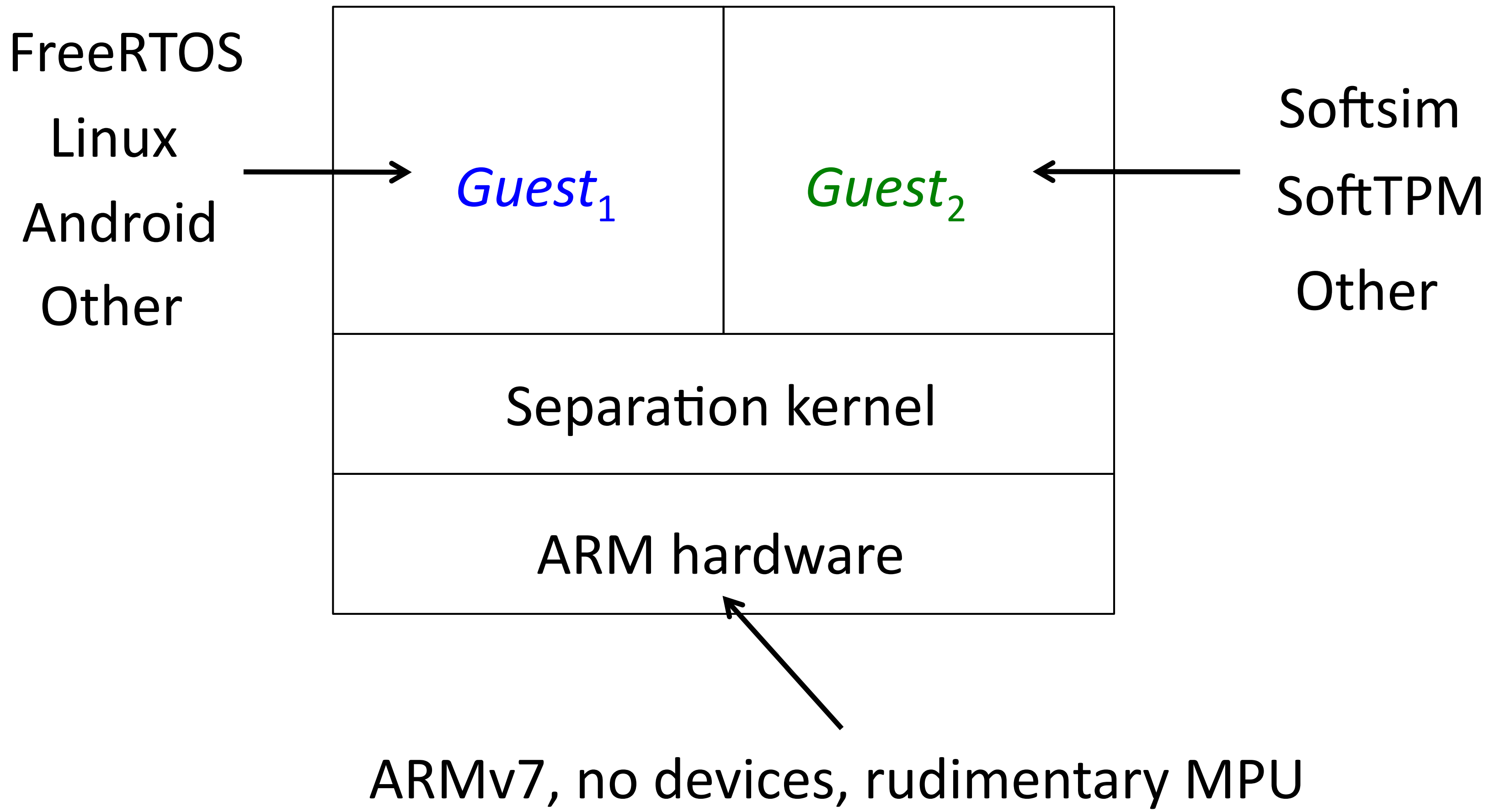
Microsoft Hyper-V + Saarbrucken:

- Weak isolation guarantees, C -> machine code

NSA + clients:

- Several experiments
- Formally verified separation kernel
- Limited model, few public details available
- Green Hills CC certified separation kernel
- Less weak isolation properties

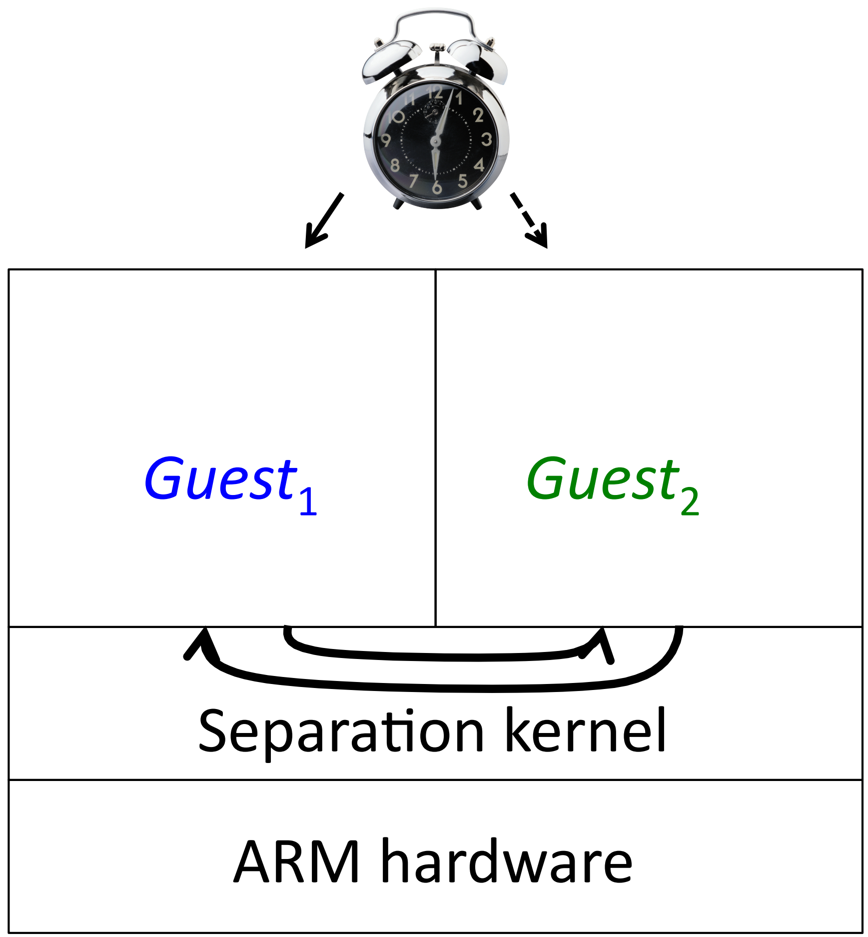
The Target System



The Prosper Kernel, v0

- Almost minimal non-trivial first step
  - No virtual memory
  - But explicit communication
- Two guest systems
- Context switching
  - Fixed scheduling
  - Static memory allocation
- Kernel routines for communication between guest systems
- Similar to SICS hypervisor, but for some details
- Design for verification

The Prosper Kernel, v0



How Functional Is This?

Not very functional at all

- No devices
  - Nae, a memory mapped device w/o dma would be ok
  - No hardware interrupts yet
  - But polling would work
- No memory management
- No kernel/user space guest system distinction

On the other hand:

- *Can* run two simple controllers
- that communicate using asynchronous message passing
- with some care

Properties

Isolation:

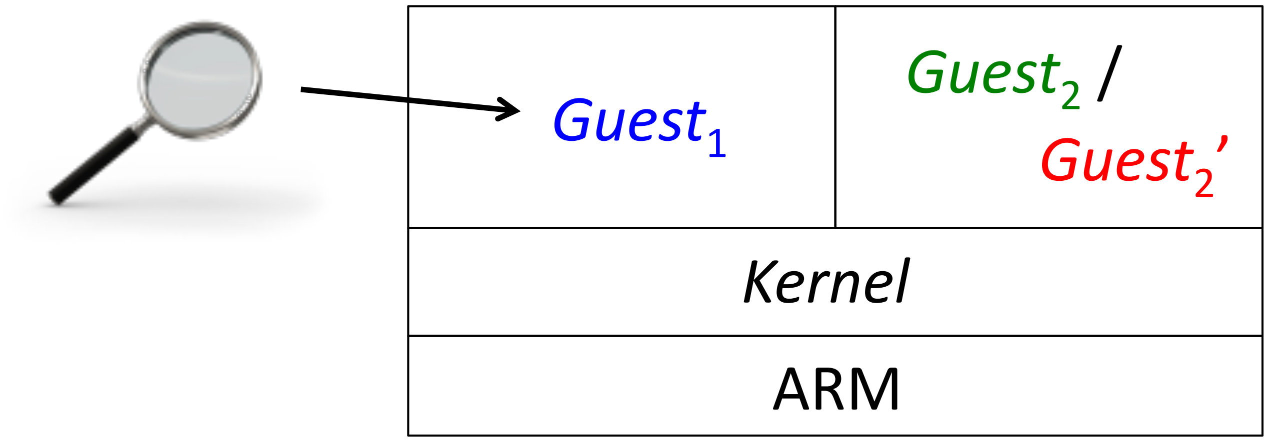
- Guests cannot unduly influence each other
- Allowed information flow only

This is the goal!

Other properties are relevant too:

- Functionality
- Extent of virtualization
- Performance

Isolation



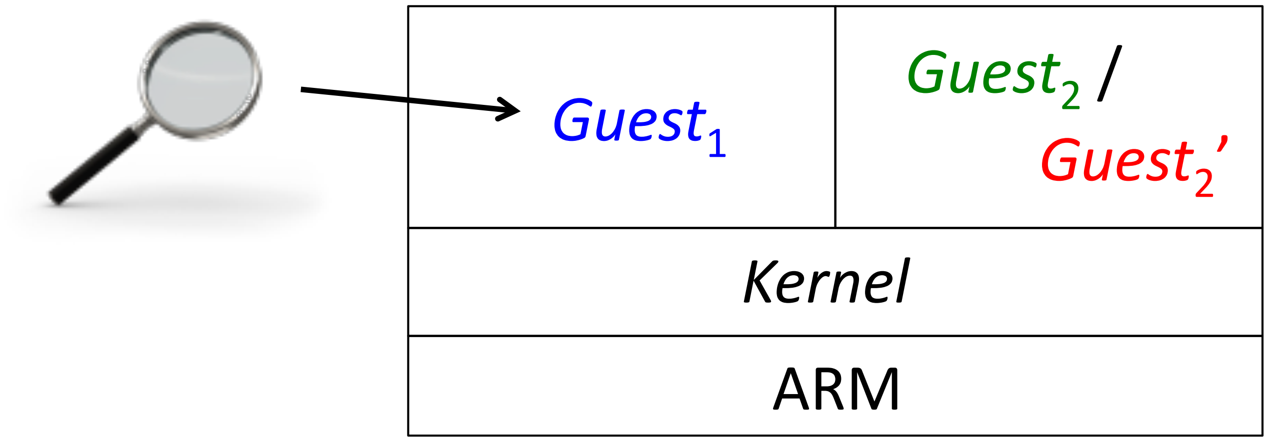
Vanilla noninterference:

- $Guest_1$, $Guest_2$ are parts of memory
- Observe $Guest_1$:s memory
- Pick $Guest_1$
- Pick $Guest_2$, $Guest_2'$

Isolation:

- $Guest_1$ + $Guest_2$ + *Kernel* + ARM cannot be distinguished from $Guest_1$ + $Guest_2'$ + *Kernel* + ARM

Isolation



Vanilla noninterference:

- $Guest_1$, $Guest_2$ are parts of memory
- Observe $Guest_1$:s memory
- Pick $Guest_1$
- Pick $Guest_2$, $Guest_2$'

Isolation:

- ~~$Guest_1$ + $Guest_2$ + *Kernel* + ARM cannot be distinguished from $Guest_1$ + $Guest_2$' + *Kernel* + ARM~~

Doesn't work, sorry: Guest1 and Guest2 are meant to communicate

Our Approach

Idea:

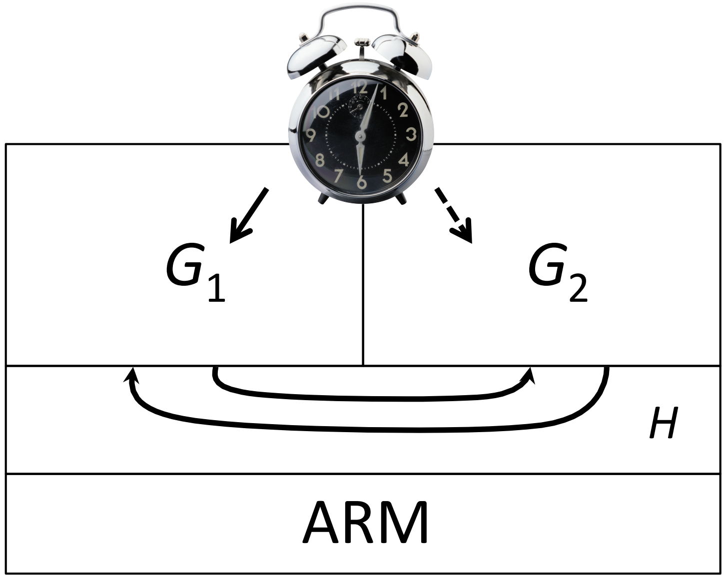
- Define *ideal model*
- Ideal model specifies *desired* behaviour
  - By extension also the *undesired* behaviour
- Correct by construction

Real model is a model of the implementation

Correctness proof:

- Show that ideal model $\cong$ real model
- $\cong$ is “indistinguishability”, or “equivalent behaviour”

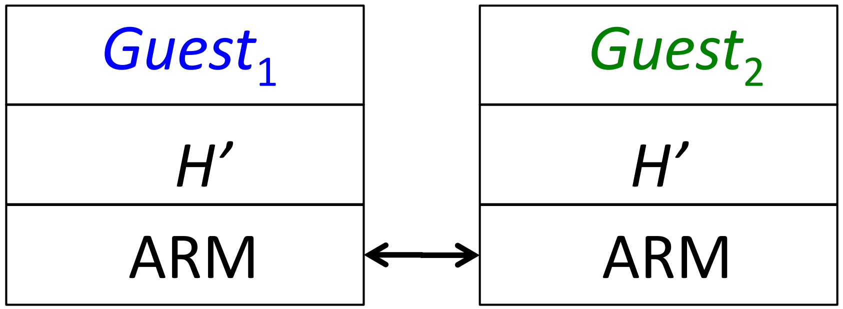
Real Model



We already have it

- Two guest systems sharing one ARM processor
- Message passing using kernel calls + context switching
- Ingredients:
  - Kernel handlers for transitions to privileged modes
  - Formal model of ARM hardware (Cambridge HOL4)

Ideal Model



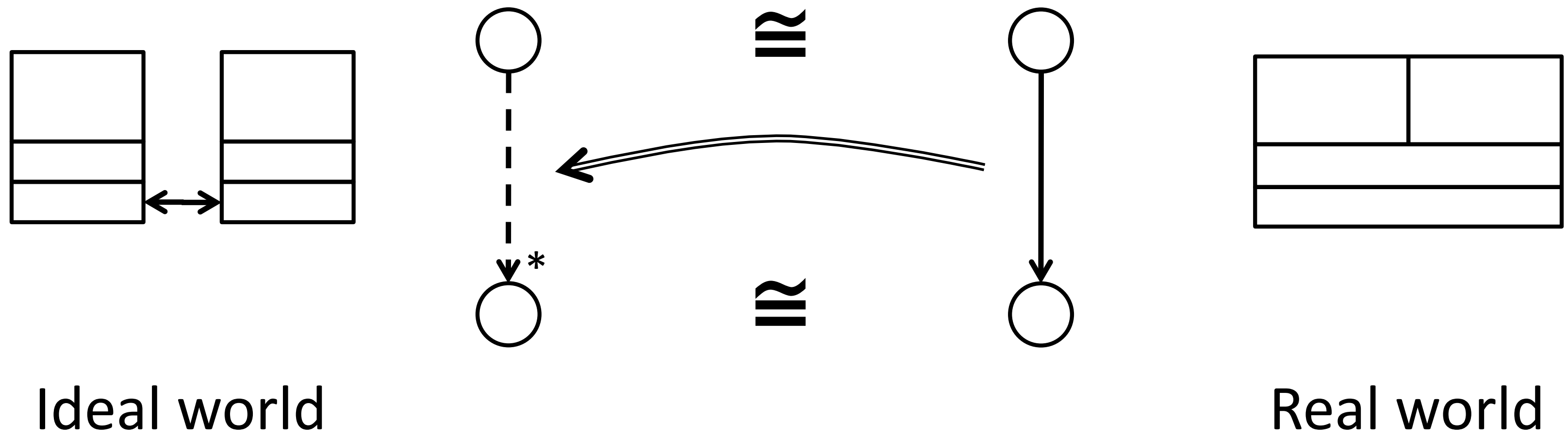
Ideal model

- $Guest_1$ and $Guest_2$ execute "as is" on physically separate ARM processors
- User mode execution only
- Communication, context switching, error handling, by "magic"
- Key part of the proof - *not trivial stuff*

Simulation

Need to:

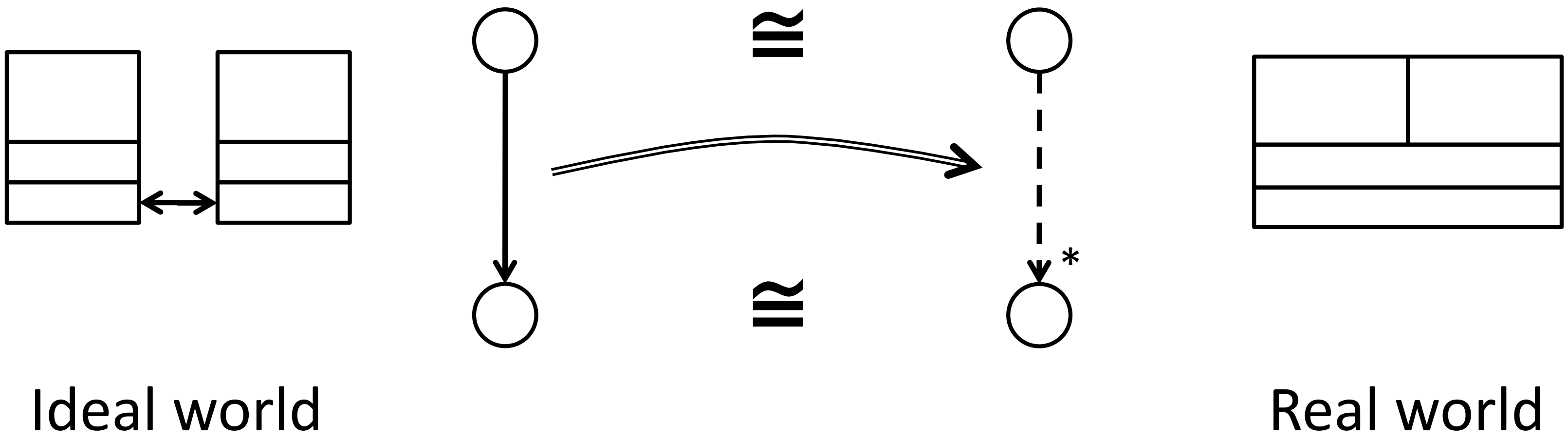
- Establish a correspondence between computation states
- Show that correspondence preserved under computation



… and the Other Direction Too

Need to:

- Establish a correspondence between computation states
- Show that correspondence preserved under computation



- This direction is not shown in seL4 + Hyper-V exercises
- Important for information flow control

What Is Involved?

- Real model (in HOL4)
- Ideal model (in HOL4)
- "Top level theorem" (in HOL4)
- Handler specifications (in HOL4)
- ARM security lemma (in HOL4)
  - Instructions are well-behaved re. mpu policy
  - Project in itself
- Handler specs implies top level theorem (in HOL4)
- Handler correctness (in BAP)
- Boot code correctness (in BAP)
- Various helper tools
- So far: > 3 manyears in total

Next Steps and Challenges

- Many: Fine grained timing, memory management, IO, multicore, tools
  - We are doing this
- Does formal verification give absolute security guarantees?
  - Sorry, no
- Complexity?
  - Yes this is an issue
- Does this scale?
  - We think so
  - Product line approach should be feasible
  - But what about device and (processor) platform proliferation?

Thank You!