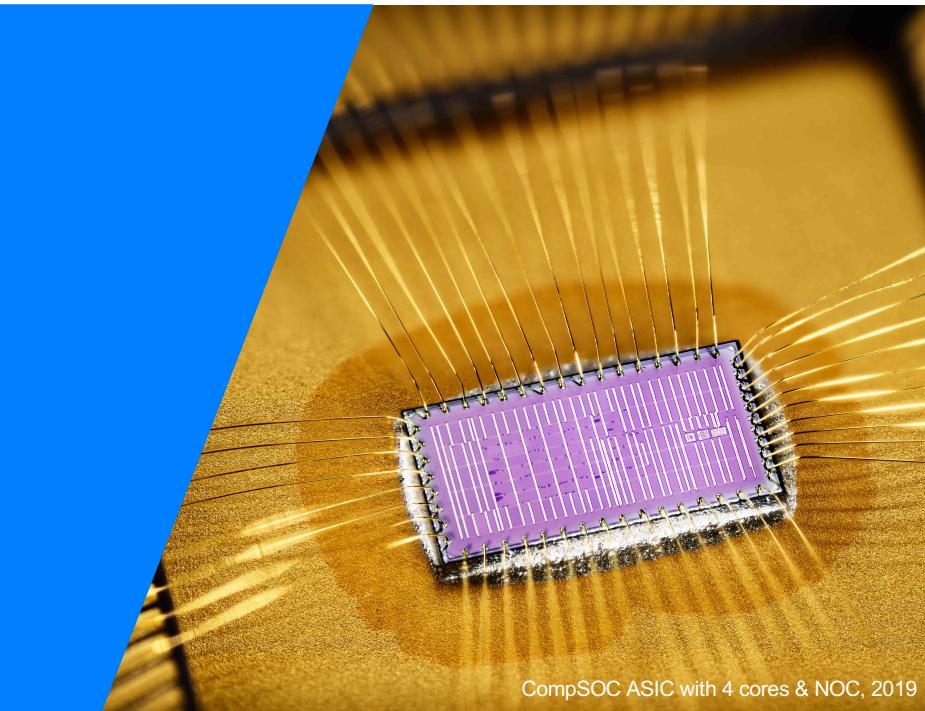


Introduction to the 5LIB0 CompSOC platform

Kees Goossens

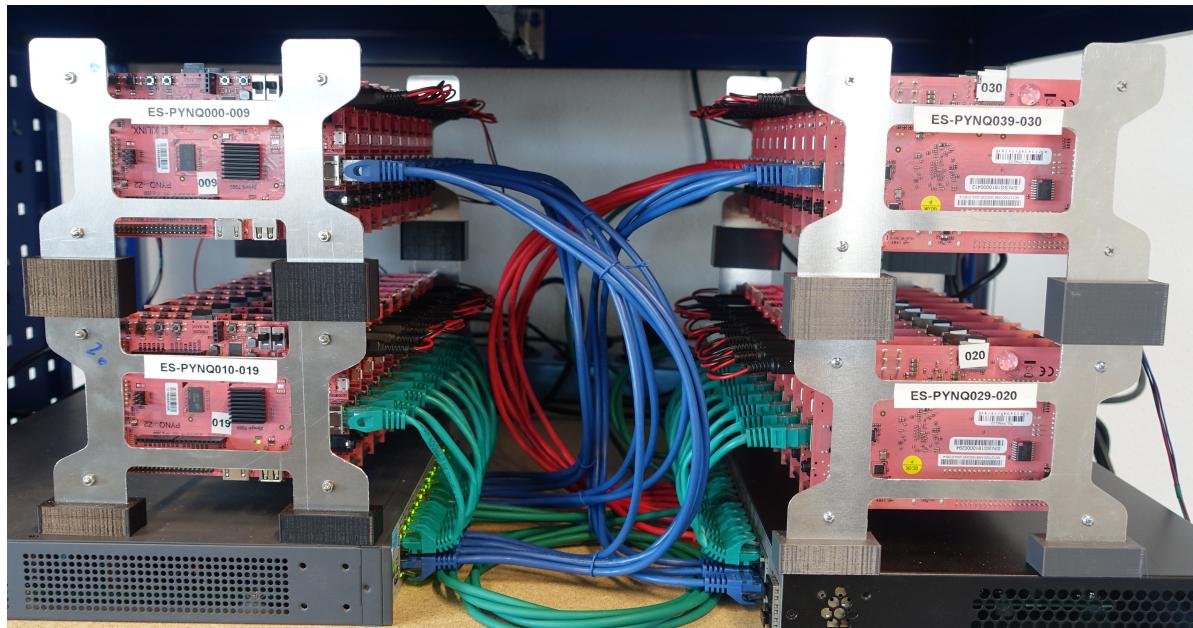


CompSOC ASIC with 4 cores & NOC, 2019

PYNQ Z-2 cloud farm

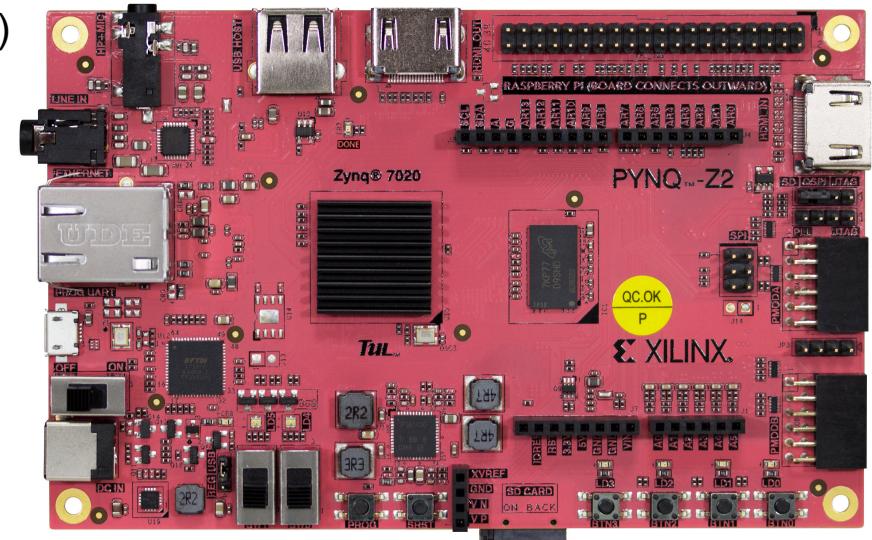
2

- all work will be on a PYNQ board (Xilinx Z7020 FPGA SOC)
- 80 boards “in the cloud” (= in our lab)
- accessible via TUE VPN
- every group has their own PYNQ board



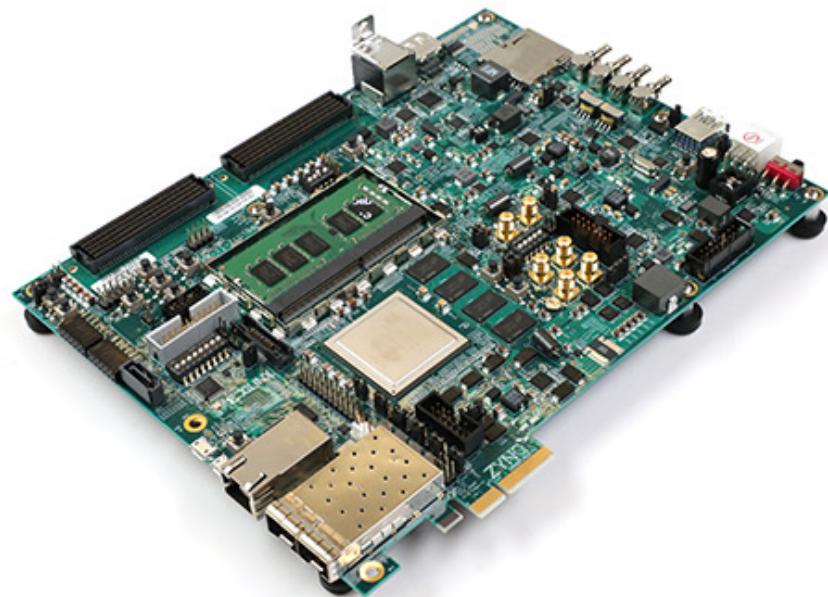
© Kees Goossens
Electronic Systems

5LIB0
2020/21



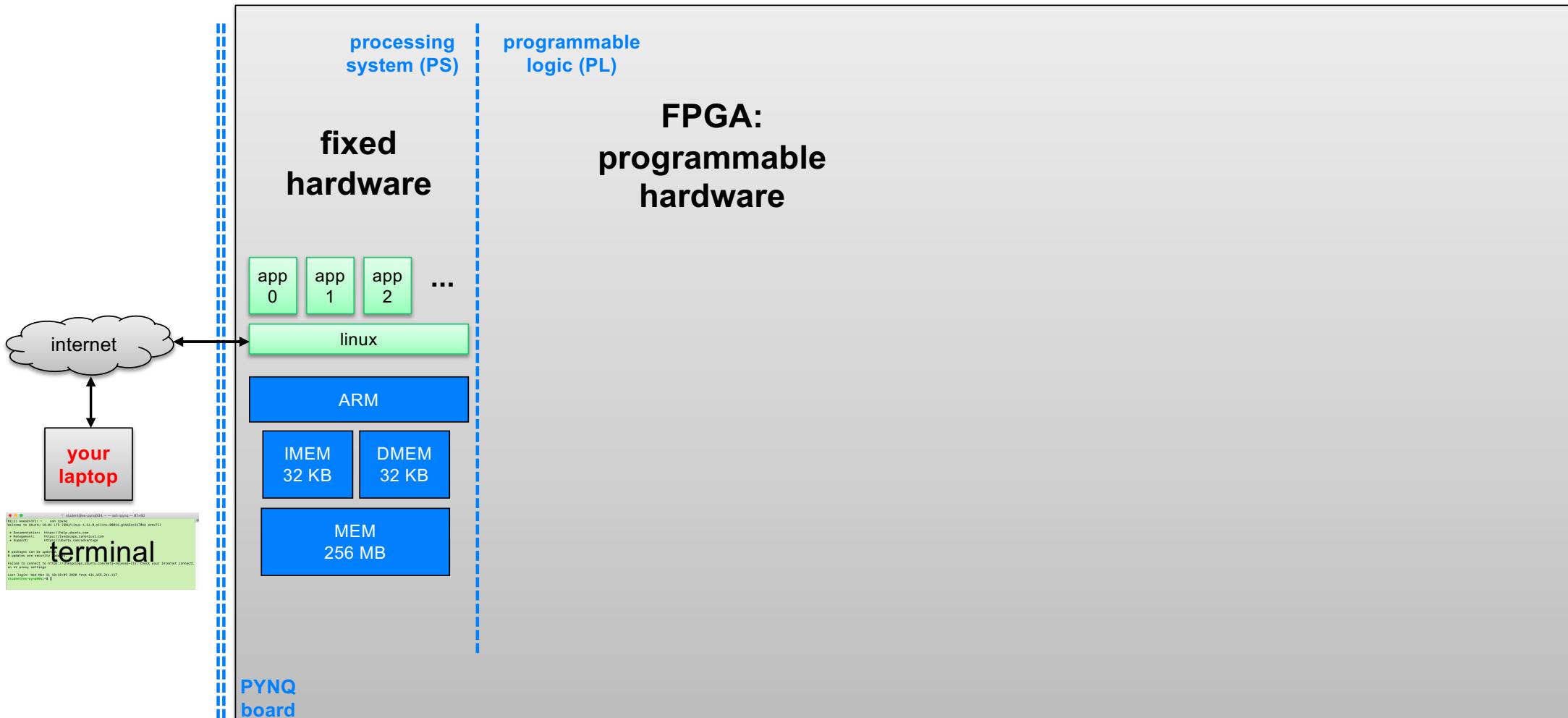
TU/e

- the CompSOC platform can also be mapped on much larger FPGAs, such as Ultrascale
- we can then either increase the number of tiles to e.g. 10
- or the memories can be much larger (e.g. 1 MB per tile, for 3 tiles)



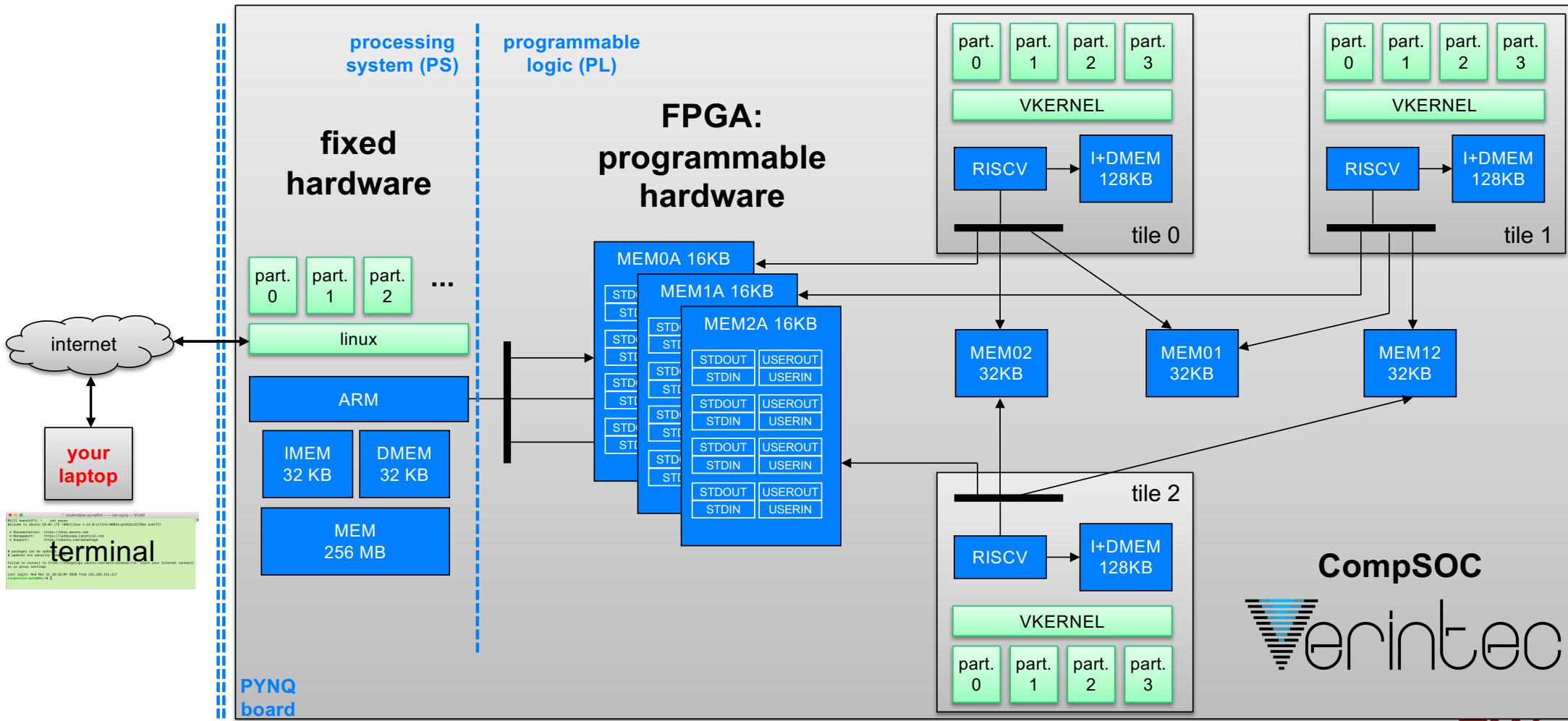
PYNQ architecture

4



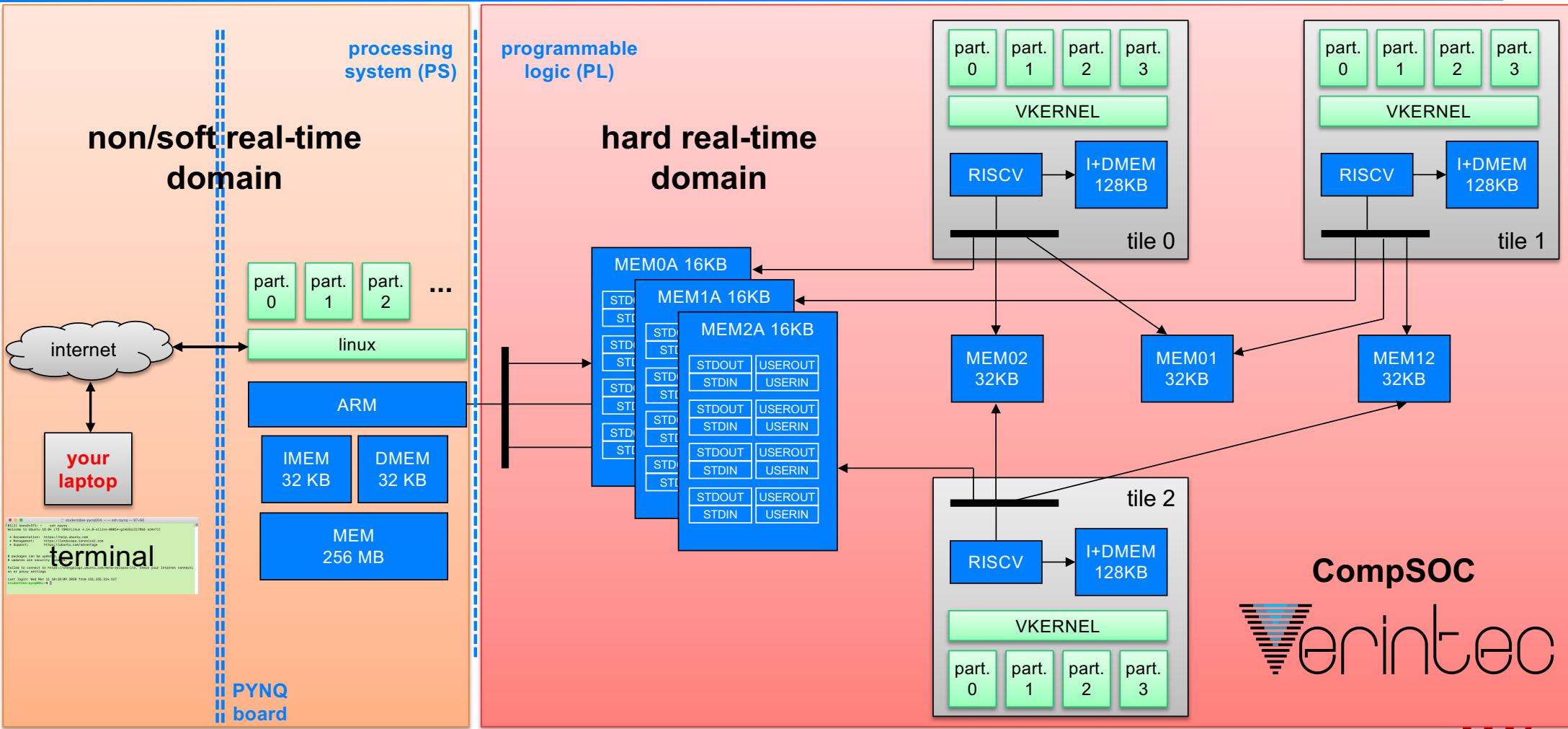
PYNQ architecture

5



(non) real-time domains

6



© Kees Goossens
Electronic Systems

5LIB0
2020/21

CompSOC
erintec
TU/e

system partitioning

7

- (very) soft real-time domain
 - optimised for ease of application development
 - dual-core ARM Cortex-A9
 - complex multi-level caching and interconnect
 - Linux (Ubuntu) operating system
 - (TCP/IP) connections with the rest of the world
- hard real-time domain
 - optimised for predictability & composability
 - 3 RISC-V tiles each with a 32-bit RISC-V core
 - simple scratchpad & shared memories, interconnect
 - VKERNEL: real-time microkernel
- communication between soft & hard real-time domains



a commercial platform based on
CompSOC principles

non/soft real-time domain

8

- dual-core ARM Cortex-A9
 - 650 MHz
 - 32 KB instruction and 32 KB data L1 caches per core
 - 512 KB of shareable L2 cache
 - 256 KB of on-chip SRAM (OCM)
 - 512 MB DRAM
- complex memory & interconnect hierarchy

PYNQ (Zynq Z7020) overview

9

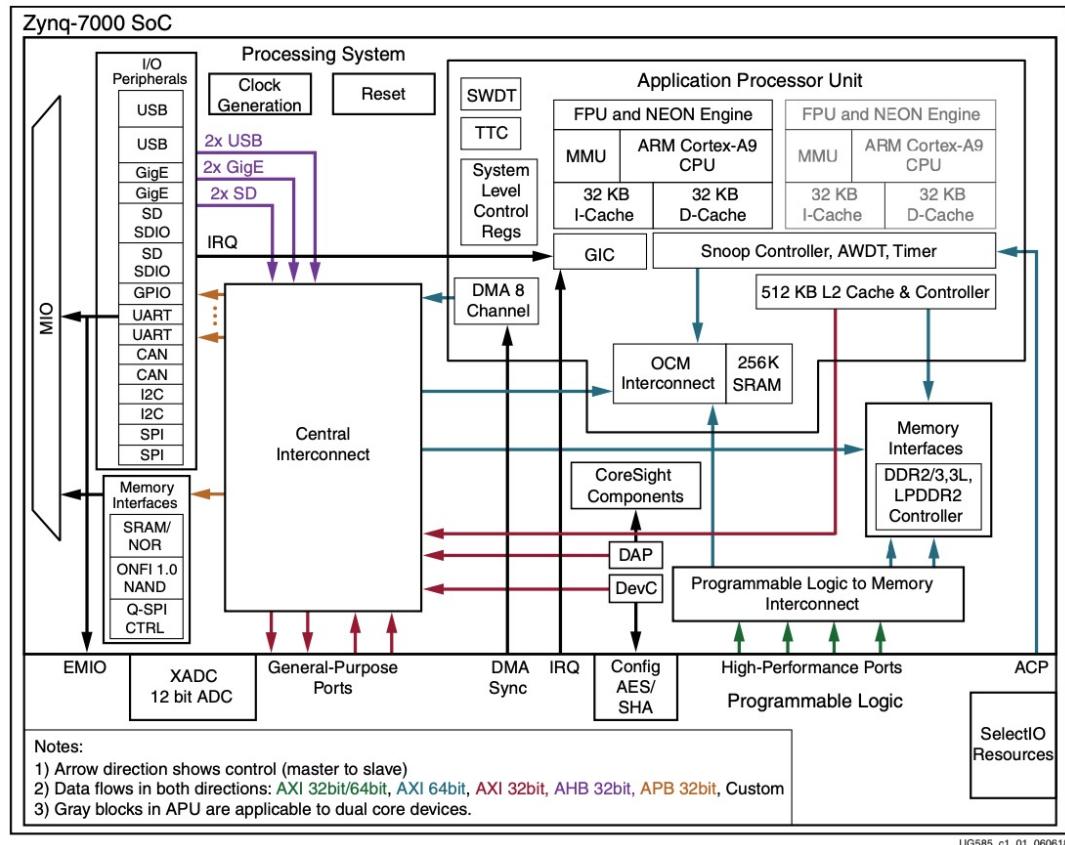
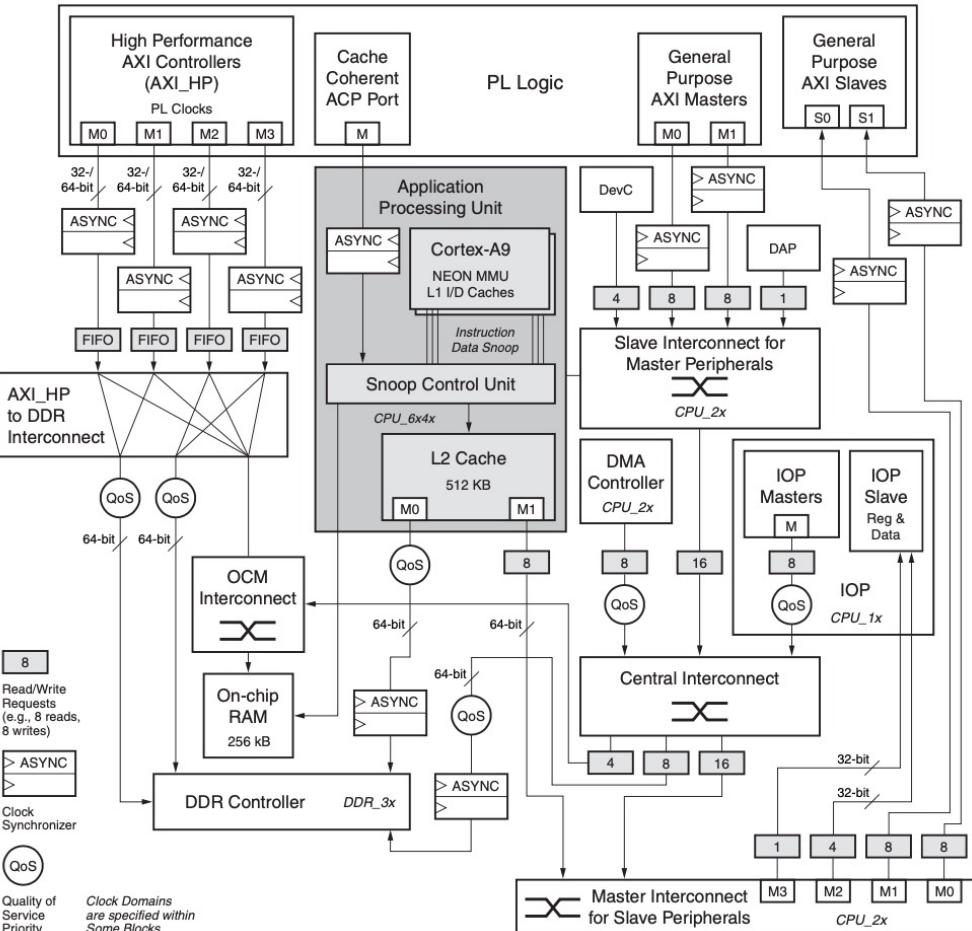


Figure 1-1: Zynq-7000 SoC Overview

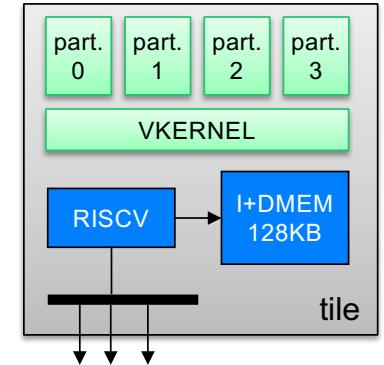


- the CompSOC / Verintec architecture ensures
 - **predictability**: for a real-time performance
 - **composability**: for independent application development
- with
 - RISC-V & memory hierarchy
 - VKERNEL to manage multiple applications
- that are tailored for real-time performance & analysability
- more details in later lectures



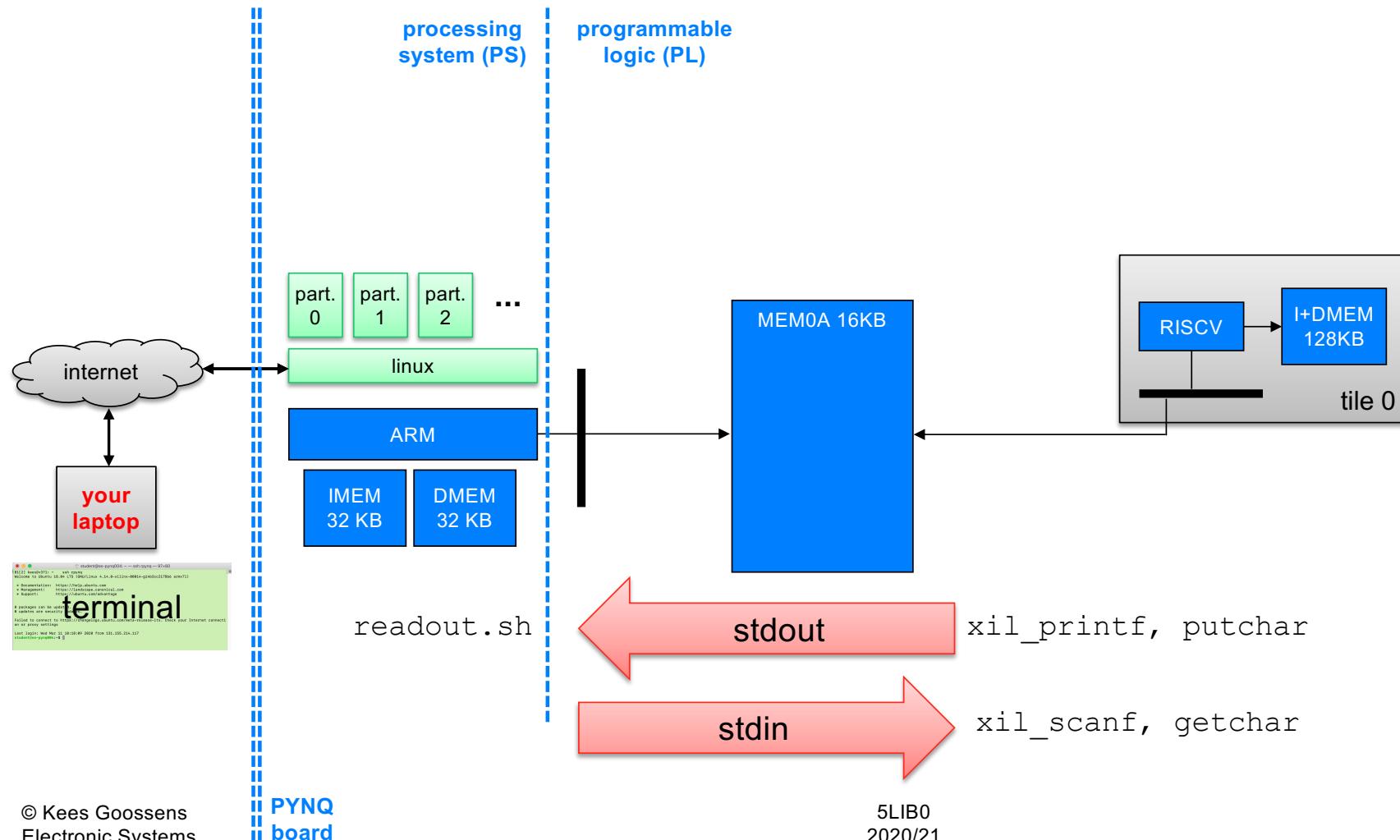
a commercial platform based on
CompSOC principles

- 40 MHz
- 5-stage pipeline
- no branch prediction
- no instruction/data caches
- hardware accelerators: integer divider & multiplier
- combined instruction & data memory with single-cycle access (AXI)
- AXI peripheral local bus (PLB) to access timers, shared memories, etc.
 - takes multiple cycles
- VKERNEL microkernel



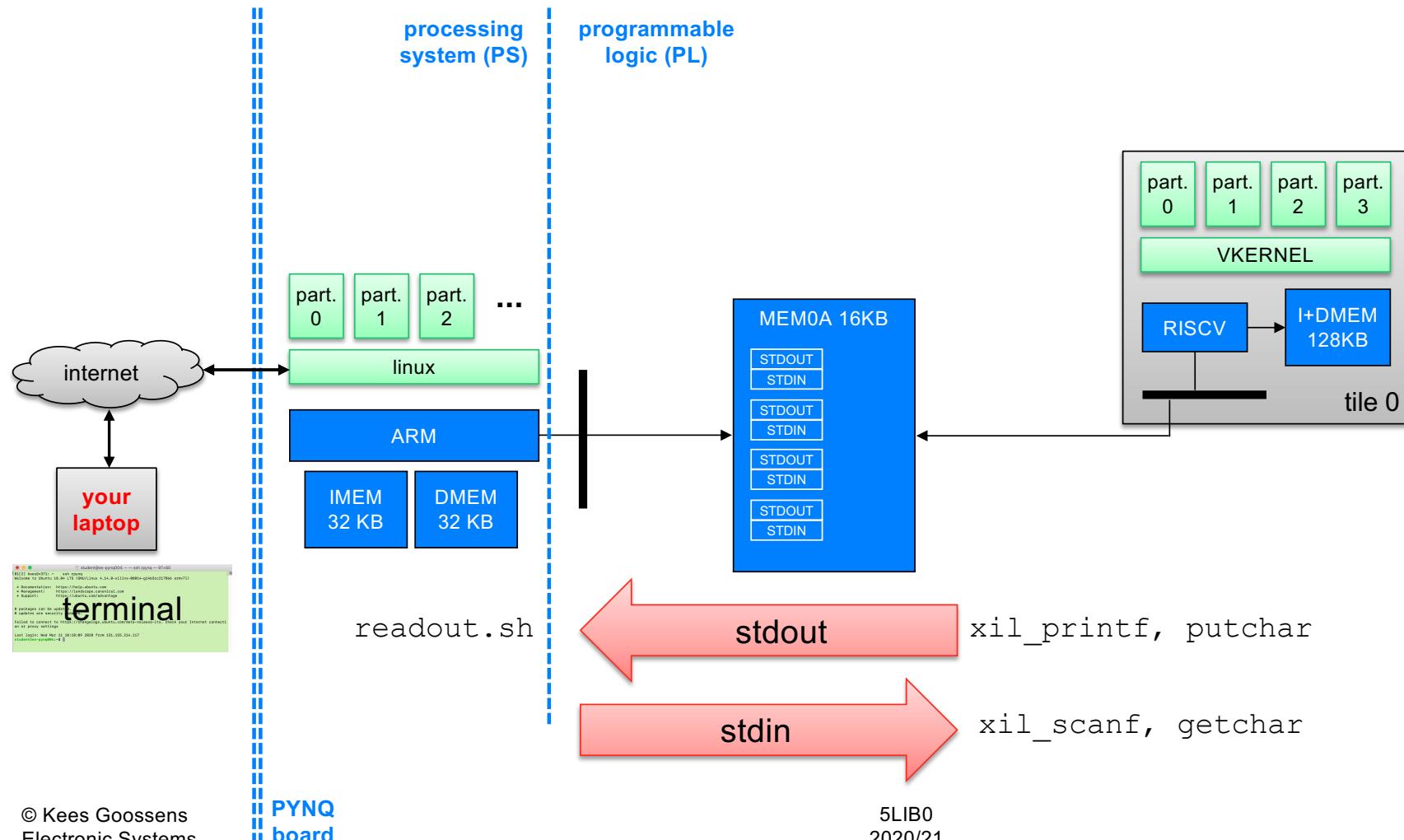
ARM – RISC-V communication

12



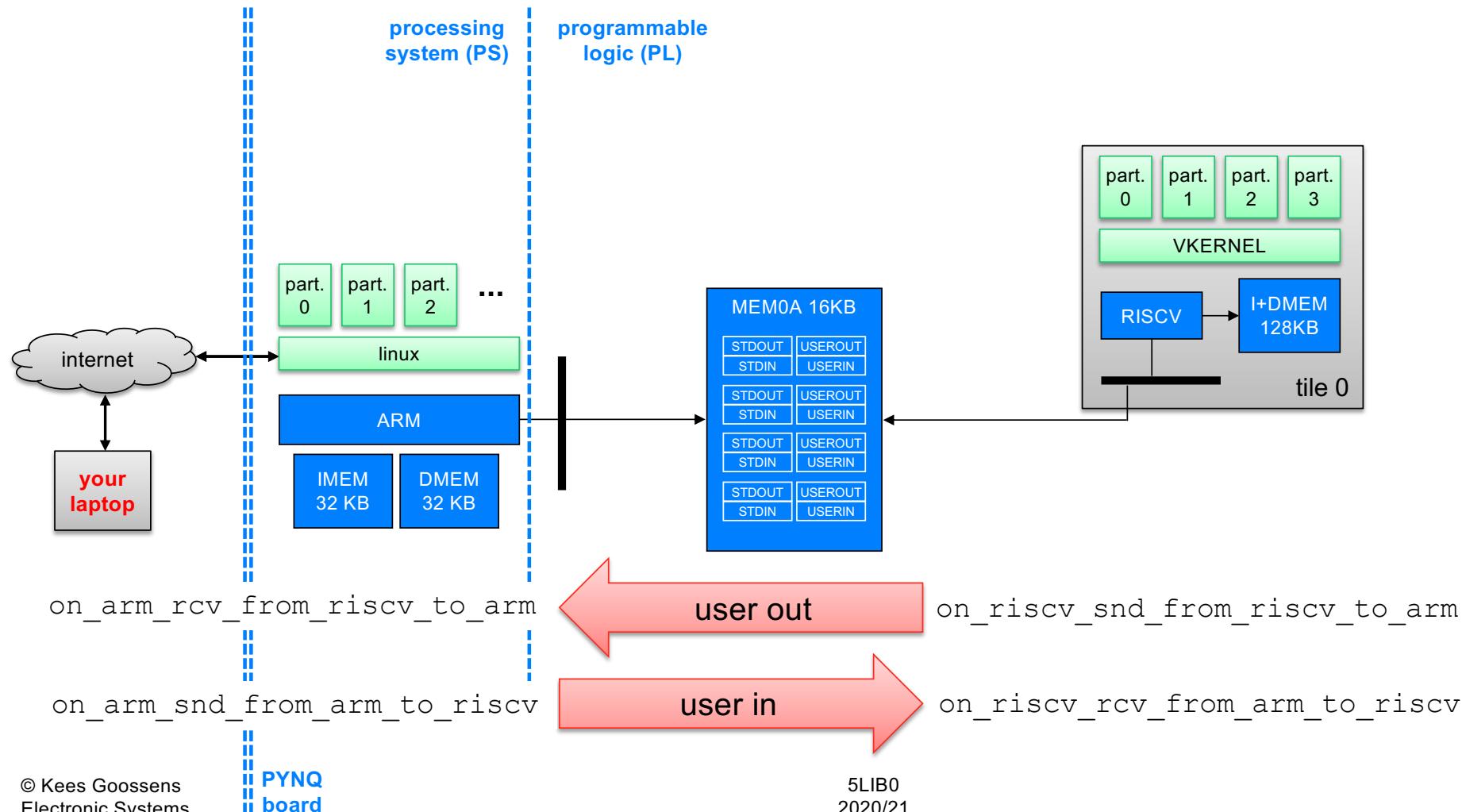
every partition has its own stdin & stdout (FIFOs)

13



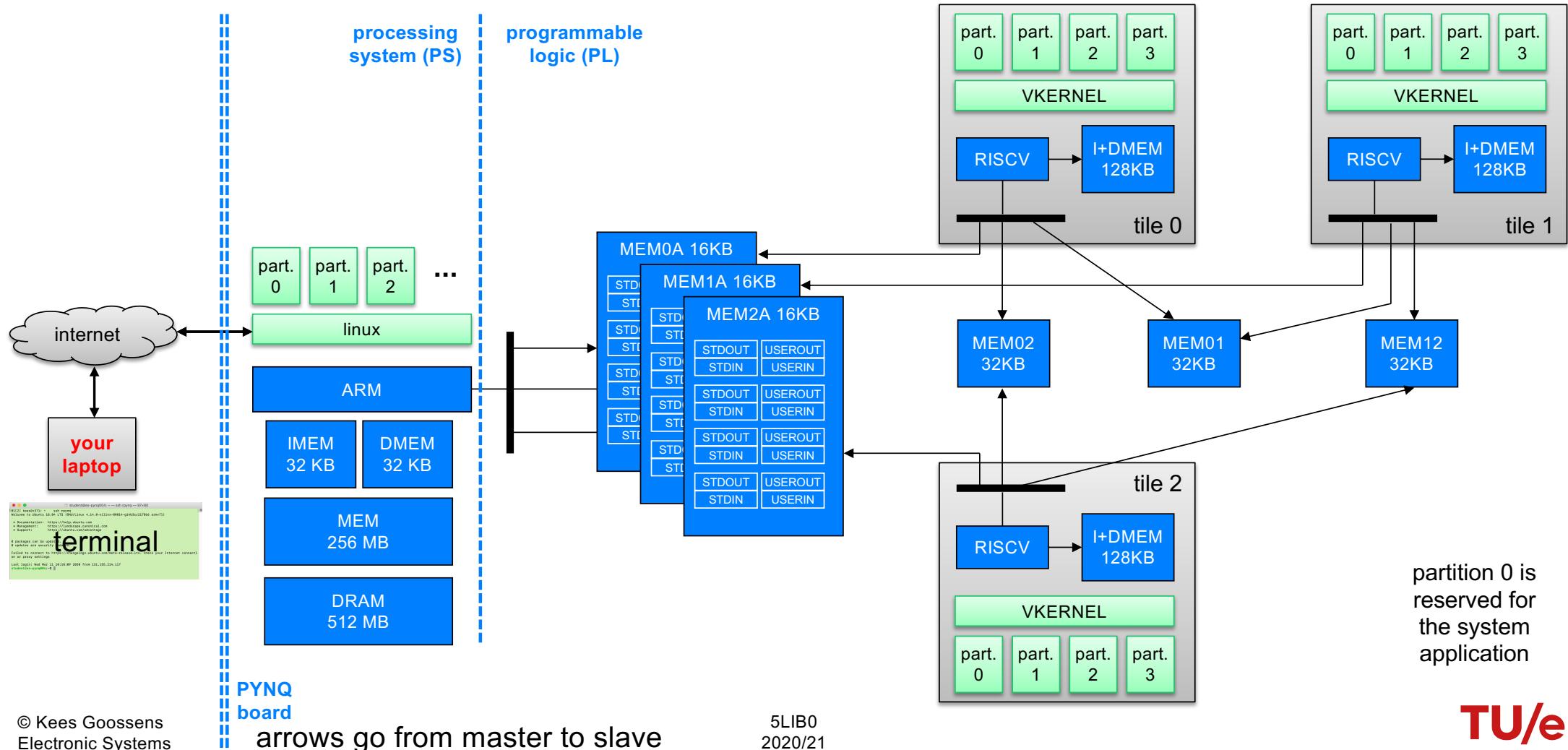
every partition also has independent pair of user-in & user-out FIFOs

14



memory hierarchy

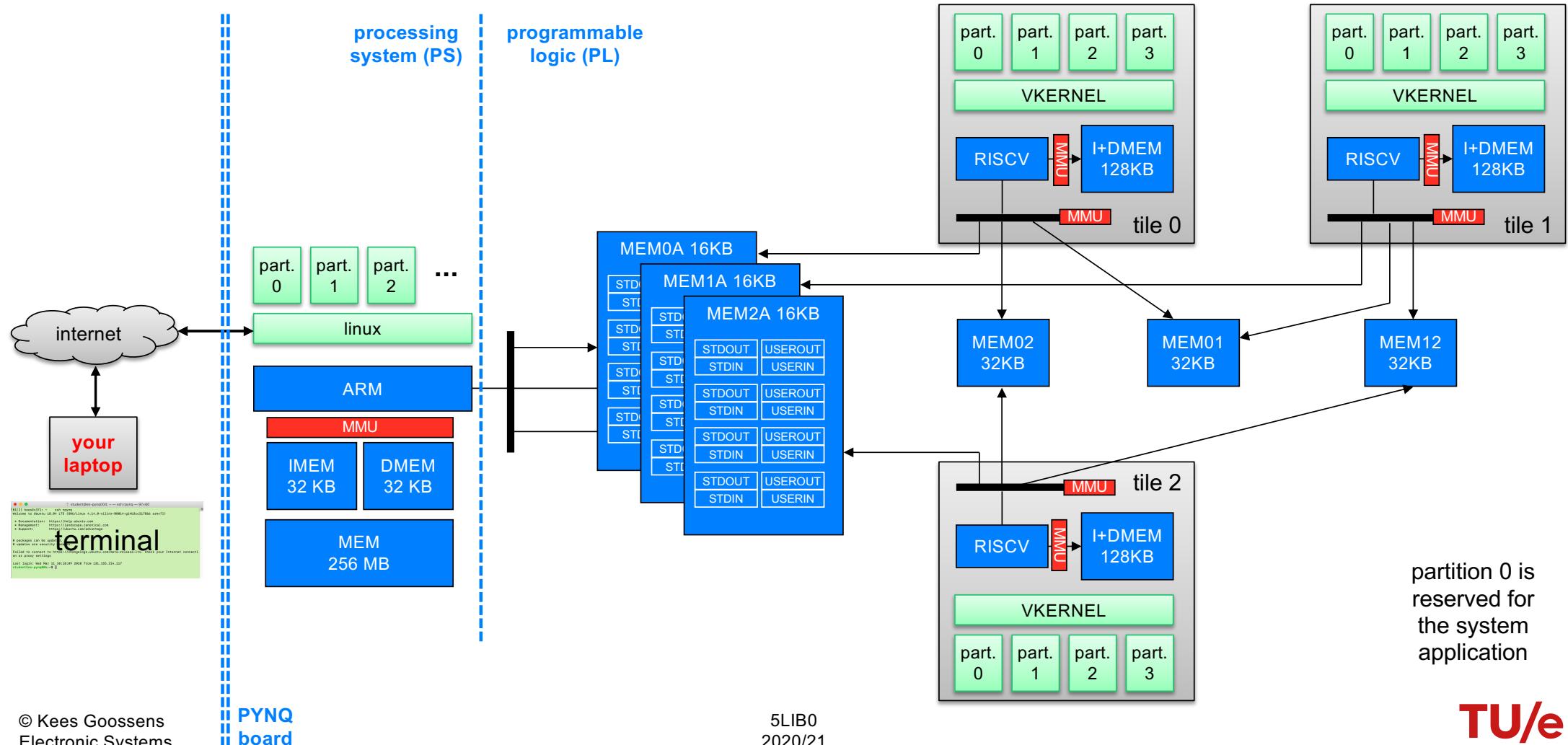
15



- ARM – RISC-V communication memories: **mem0a, mem1a, mem2a**
 - 16 KB = 0x4000 bytes = 4 Kword
- instruction & data memories: **mem0, mem1, mem2**
 - 128 KB = 0x20000 bytes = 32 Kword
- RISC-V – RISC-V communication memories: **mem01, mem02, mem12**
 - 32 KB = 0x8000 bytes = 8 Kword
- **memory location**
- **physical address**: the memory address that hardware sees / uses for a memory location
- **virtual (logical) address**: the memory address that software sees / uses for a memory location
- the translation from virtual to physical addresses is performed by a **memory management unit (MMU)**
- for ARM we always show virtual addresses
- we first discuss physical addresses

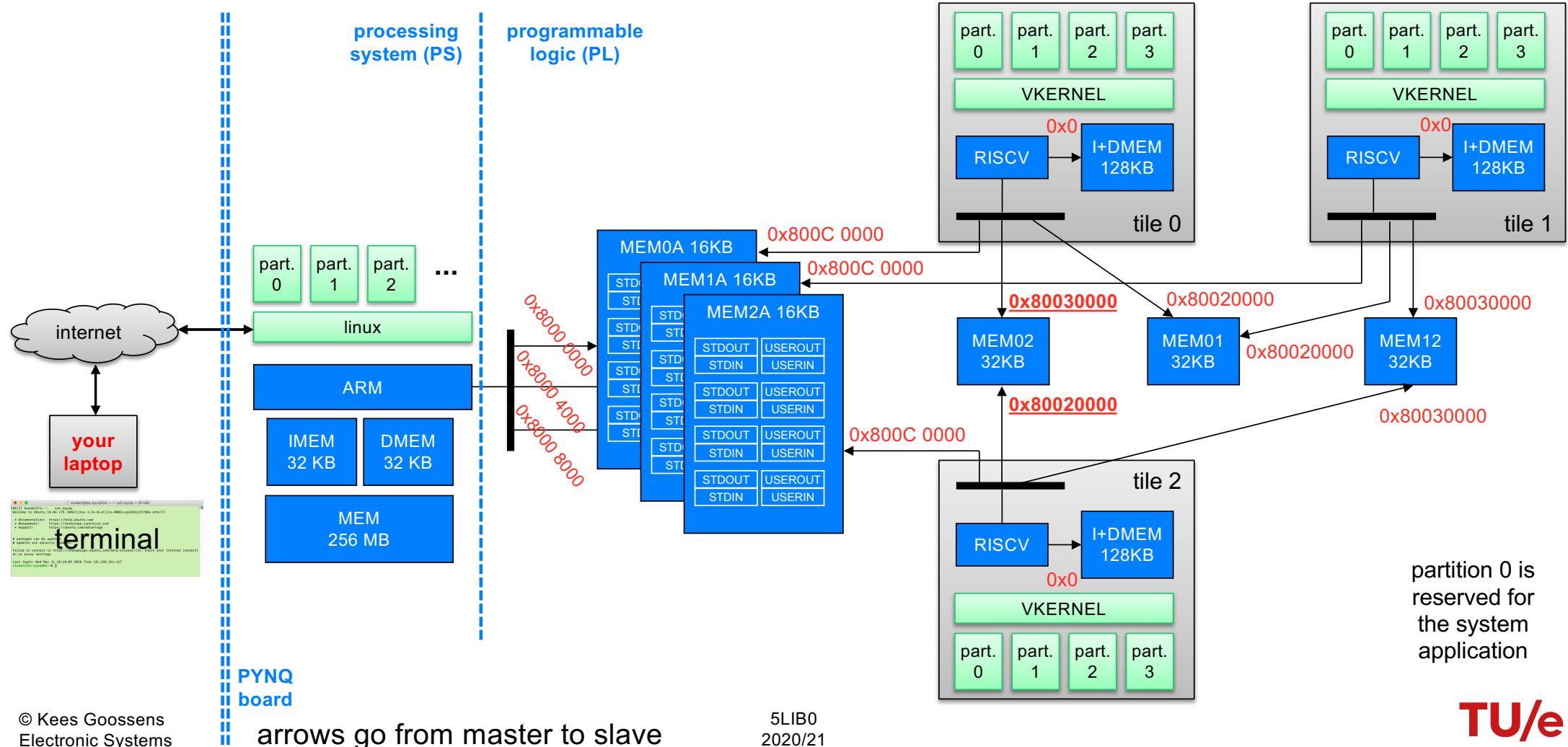
MMUs

17



starting physical addresses of memories

18



physical memory map

19

- ARM – RISC-V communication memories: **mem0a, mem1a, mem2a**
 - 16 KB = 0x4000 bytes = 4 Kword
 - same physical addresses 0x800C0000 up to 0x800C4000 on all tiles
- instruction & data memories: **mem0, mem1, mem2**
 - 128 KB = 0x8000 bytes = 32 Kword
 - same physical addresses 0x0 up to 0x20000 on all tiles
- RISC-V – RISC-V communication memories: **mem01, mem02, mem12**
 - 32 KB = 0x8000 bytes = 8 Kword
 - mem01 on both tile 0 (`TILE0_MEM01_START`) and tile 1 (`TILE0_MEM01_START`) physical addresses 0x80020000 up to 0x80028000
 - mem12 on both tile 1 (`TILE0_MEM12_START`) and tile 2 (`TILE2_MEM12_START`) physical addresses 0x80030000 up to 0x80038000
 - mem02 on tile 0 (`TILE0_MEM02_START`) physical addresses **0x80030000 up to 0x80038000** notice this difference!
 - mem02 on tile 2 (`TILE2_MEM02_START`) physical addresses **0x80020000 up to 0x80028000**

memory map for shared memories is defined in vep_0/libbsp/include/platform.h

20

- lists all local memories of all tiles along with base addresses and sizes
- use the symbolic name (e.g. TILE0_MEM01_START) instead of the absolute number

```
#define TILE0_MEM01_START 0x80020000      /* mem01 in core 0's mem map */
#define TILE0_MEM02_START 0x80030000      /* mem02 in core 0's mem map */
#define TILE1_MEM01_START 0x80020000      /* mem01 in core 1's mem map */
#define TILE1_MEM12_START 0x80030000      /* mem12 in core 1's mem map */
#define TILE2_MEM02_START 0x80020000      /* mem02 in core 2's mem map */
#define TILE2_MEM12_START 0x80030000      /* mem12 in core 2's mem map */

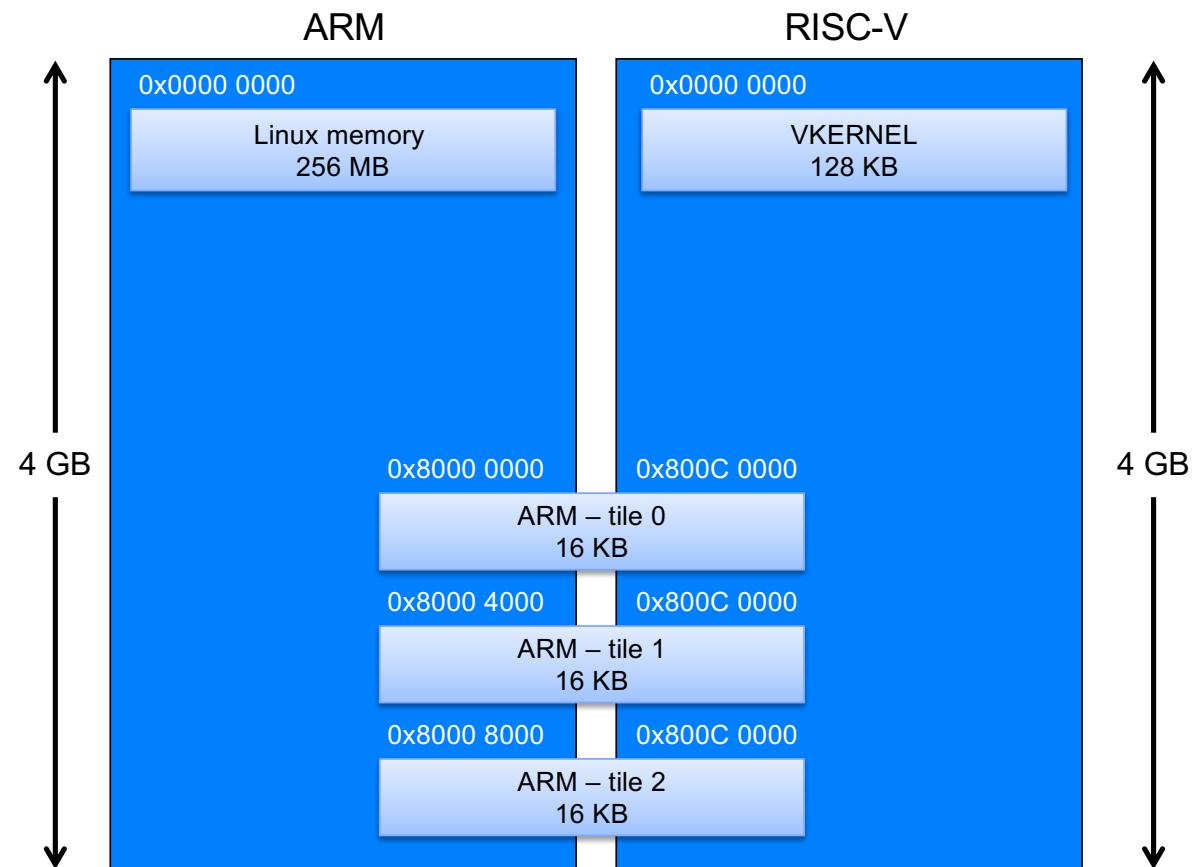
#define TILE0_MB_LOCAL_MEMORY_DLMB_BRAM_IF_CNTLRL_SLMB 0x00000000      /* mem0: core 0's instr. & data memory */
#define TILE0_AXI_BRAM_CTRL_0_S_AXI 0x0800C0000      /* mem0a between RISC-V core 0 and ARM */
#define TILE0_GLOBAL_TIMER 0x80FC0000      /* global timer of core 0 */
#define TILE0_PARTITION_TIMER 0x80FD0000      /* partition timer of core 0 */

/* and the same for the other two tiles TILE1* & TILE2* */
```

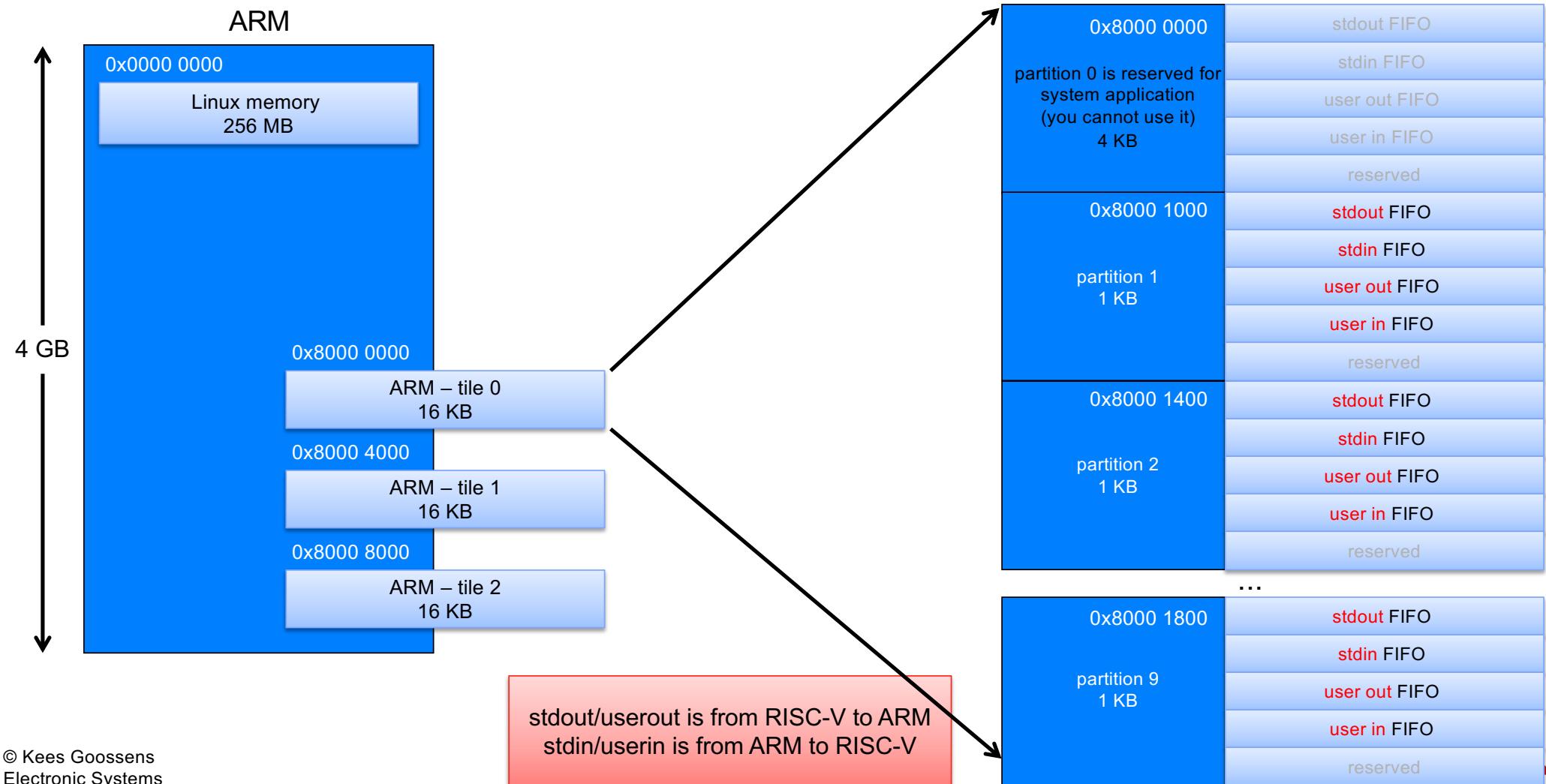
memory map ARM - RISC-V

21

- 32 bit address space (4 GB) on both cores
- mem0a, mem1a, mem2a have different addresses on the ARM
- mem0a, mem1a, mem2a have the same address on all RISC-V cores
- ARM: virtual addresses
- RISC-V: physical addresses

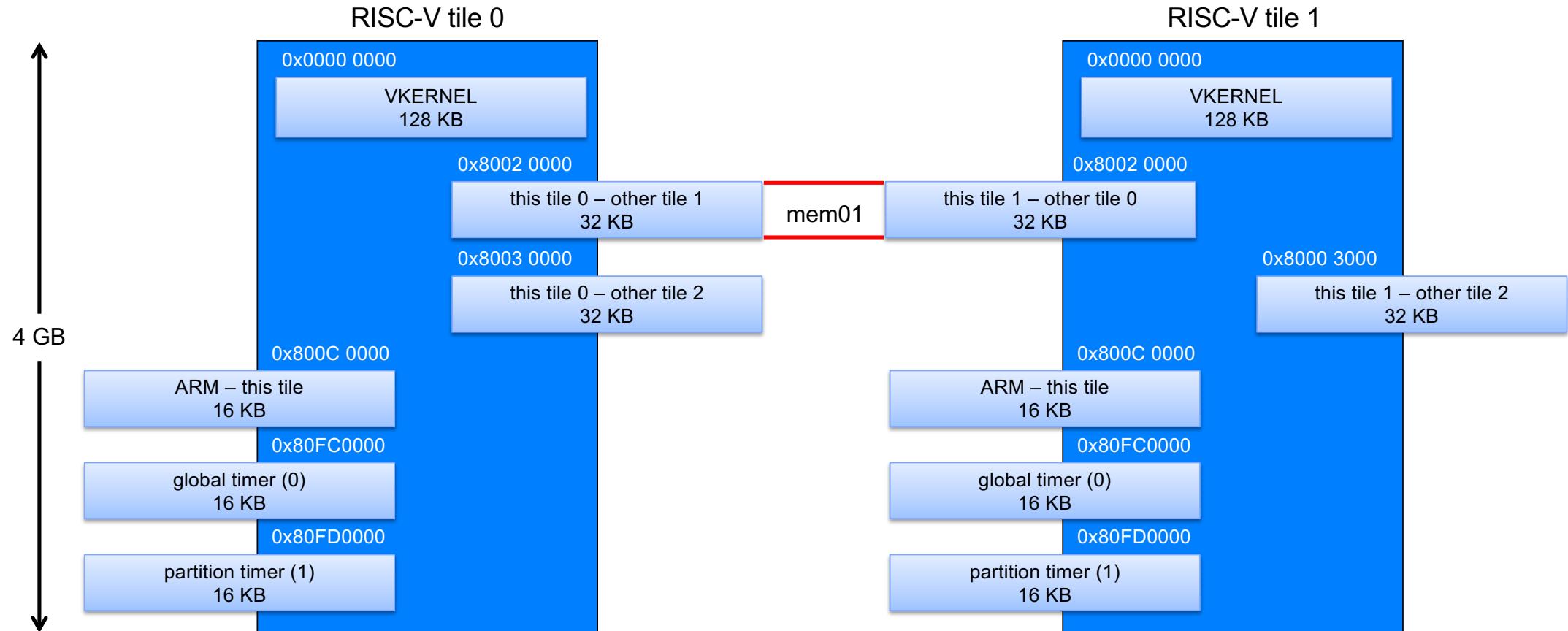


predefined C-HEAP FIFOs in mem0a, mem1a, mem2a as seen from the ARM



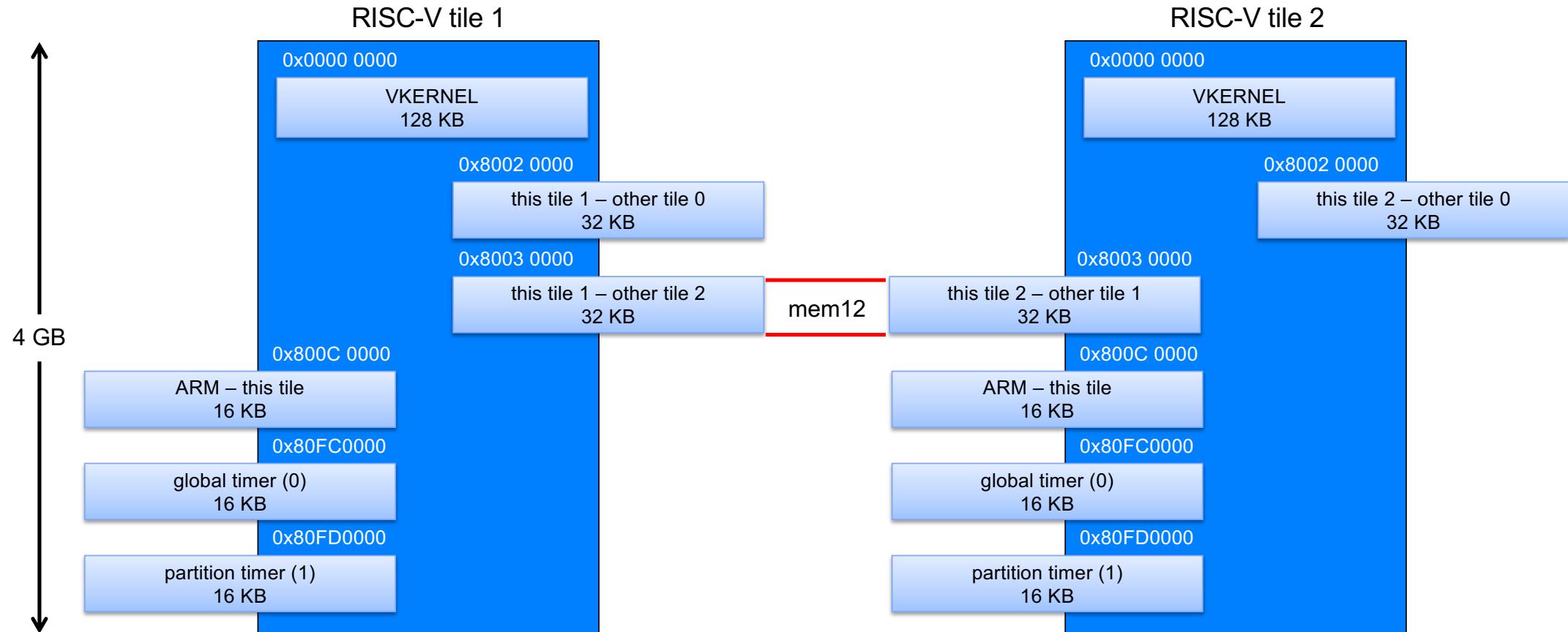
RISC-V shared memories (mem01)

23



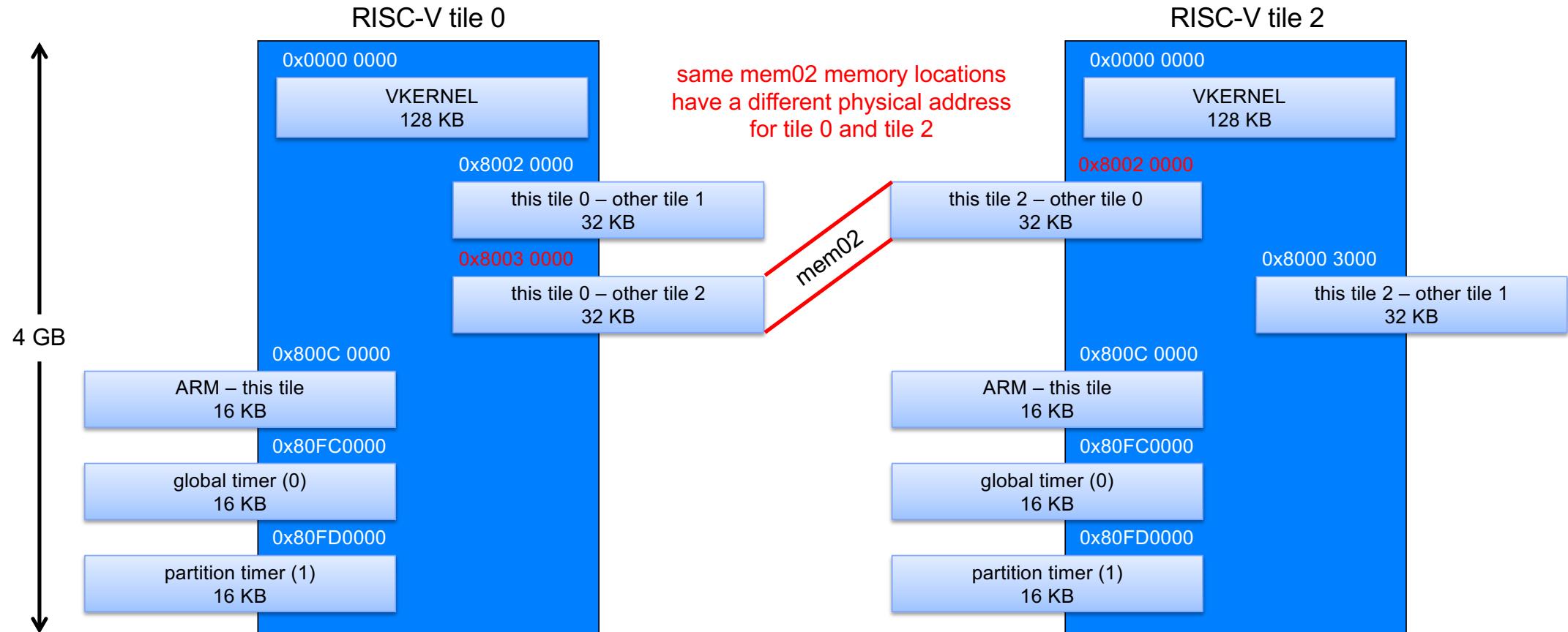
RISC-V shared memories (mem12)

24



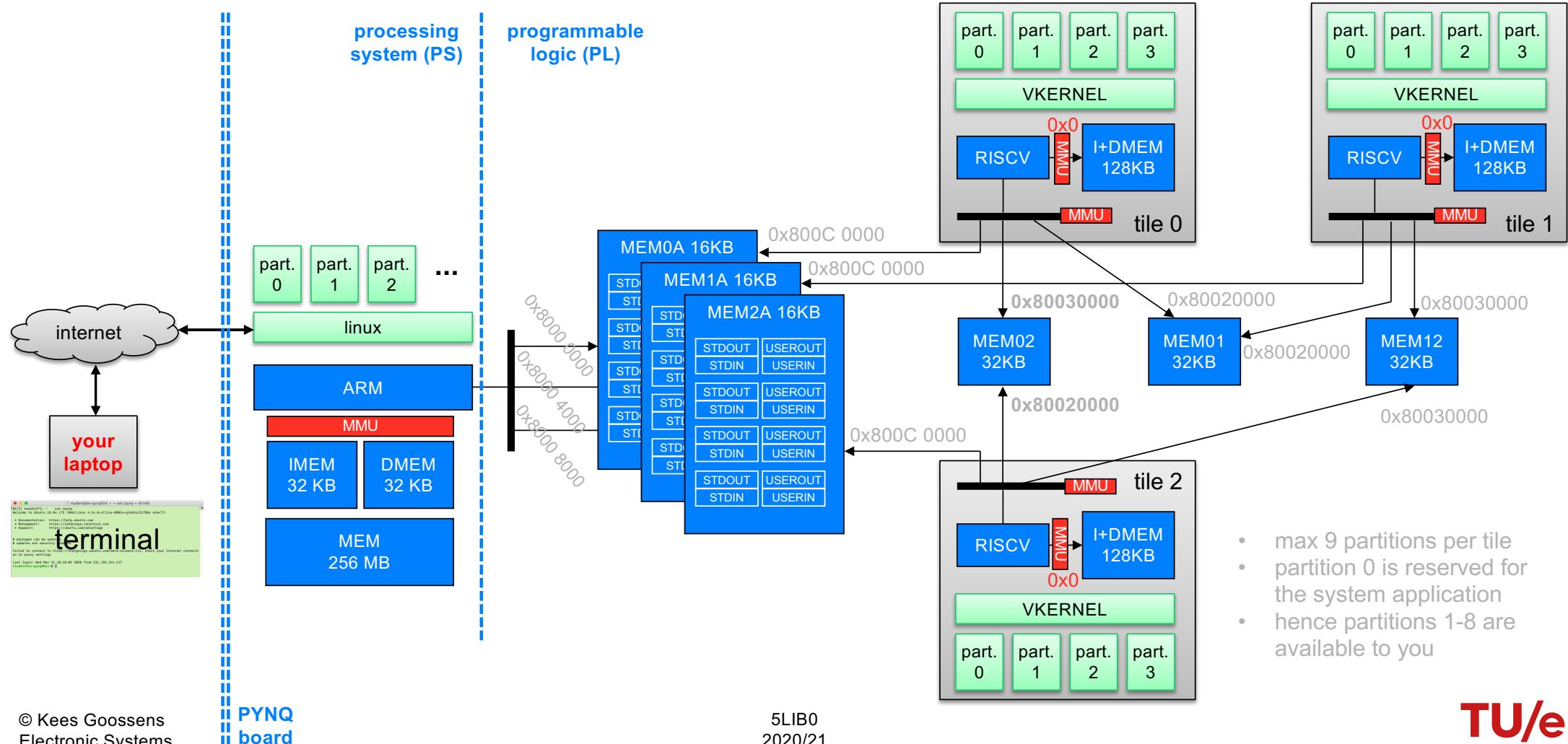
RISC-V shared memories (mem02)

25



MMUs

26



- max 9 partitions per tile
 - partition 0 is reserved for the system application
 - hence partitions 1-8 are available to you

virtualised memory – communication memories

27

- ARM – RISC-V communication memories: **mem0a, mem1a, mem2a**
 - 16 KB = 0x4000 bytes = 4 Kword
 - **equally divided between the system partition (0) and up to 8 user partitions (1-8, 1 KB each)**
 - **each partition sees virtual addresses starting at 0x800C 0000**
 - ARM communication memories are managed by the system

virtualised memory – instruction & data memories

28

- instruction & data memories: **mem0, mem1, mem2**
 - 128 KB = 0x20000 bytes = 32 Kword
 - same physical addresses 0x0 up to 0x20000 on all tiles
 - divided between up to 9 partitions (4-64 KB per partition)
 - each partition sees **virtual addresses** 0x0 up to 0x1000 (4 KB) or 0x10000 (64KB)
 - for each partition declares a **memory region** in the instruction/data memory
 - 0K-32K is reserved for the partition 0 (system application)
 - partitions can use 4, 8, 16, 32, 64 KB memory, which must be aligned ($128K \% \text{ size} == 0$) in memory
- e.g. declaring in the vep-config.txt file
 - on tile 0 partition 1 has 14K stack in 64K memory starting at 64K
 - on tile 1 partition 1 has 4K stack in 32K memory starting at 96K

virtualised memory – shared memories

29

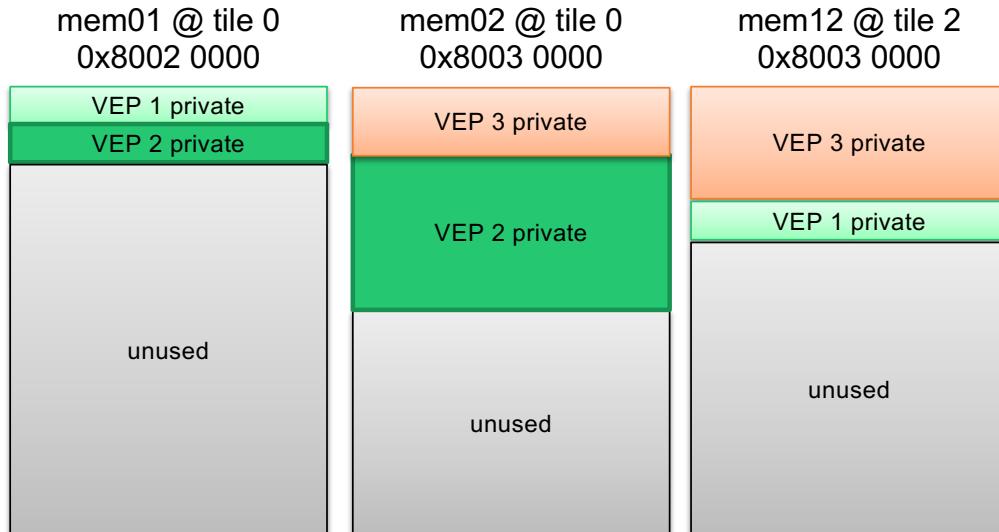
- RISC-V – RISC-V communication memories: mem01, mem02, mem12
 - 32 KB = 0x8000 bytes = 8 Kword
 - each VEP can declare a **private memory region** in each shared memory that only it can access
 - you have to manage the data layout in the private memory regions
 - **each partition sees virtual addresses from at the start address of the shared memory (i.e. 0x800[23]0000)**
 - thus (in your C code) all partitions of the VEP see their private region at the start of each shared memory
- e.g. declaring in the vep-config.txt file
 - in mem01 have 1K private memory starting at 0K
 - in mem02 have 3K private memory starting at 13K
- means that
 - on tile 0 this VEP accesses its 1K in mem01 starting at VEP_PRIVATE_REGION_TILE0_MEM01_START, i.e. 0x80020000
 - on tile 0 this VEP accesses its 1K in mem02 starting at VEP_PRIVATE_REGION_TILE0_MEM02_START, i.e. 0x80030000
 - on tile 1 this VEP accesses its 1K in mem01 starting at VEP_PRIVATE_REGION_TILE1_MEM01_START, i.e. 0x80020000
 - on tile 2 this VEP accesses its 1K in mem02 starting at VEP_PRIVATE_REGION_TILE2_MEM02_START, i.e. 0x80020000
 - since no private memory has been declared in mem12 it cannot be accessed from any tile by this VEP

virtualised memory – private memory regions

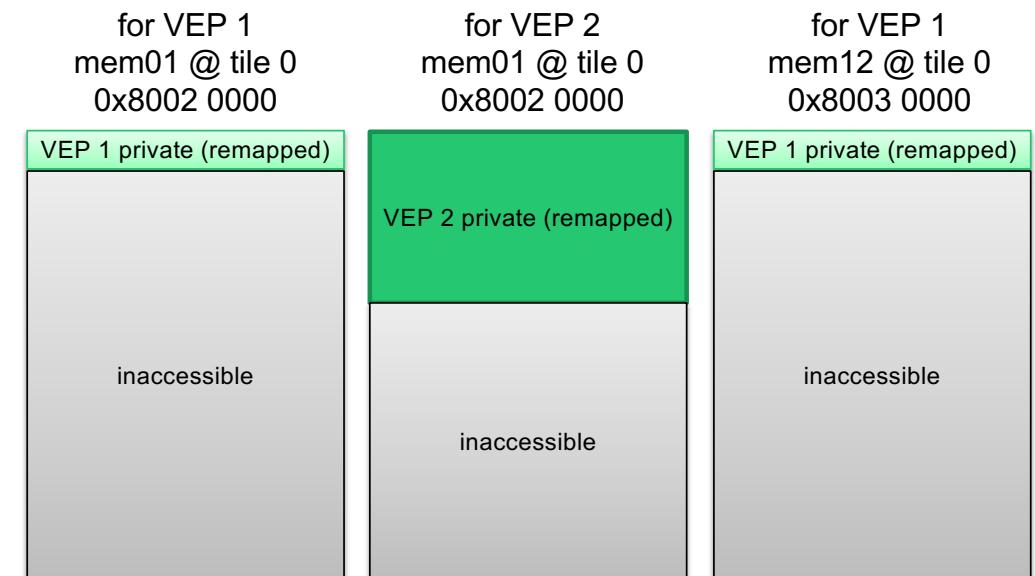
30

- VEP 1 runs on tiles 0 & 2, has private memory on mem01/02
- VEP 2 runs on tile 1, has private memory on mem02
- VEP 3 runs on tiles 1 & 2, has private memory on mem01/12

physical memory maps



logical memory maps



virtualised memory – private memory regions

31

- **typedef the data layout of the shared memory regions in `shared_memories/vep_private_memory.h`**

```
typedef volatile struct {
    uint32_t r0;                                // the data fields in this shared region
    uint32_t r1;
    char vheap[1000];
} vep_private_mem01_t;                         // the name of the shared region
#define USE_VEP_PRIVATE_MEM01 /* set this to automatically declare all variables */
```

- **the variable `vep_private_mem01` is automatically declared**
- **use the shared region in your code e.g. `partition_0_1/main.c`**

```
#include "vep_private_memory.h"
#include "vep_public_memory.h"
int main (void)
{
    xil_printf("r0 is mapped in private memory at address 0x%08X with value %d\n",
              &(vep_private_mem01->r0), vep_private_mem01->r0);
    vep_private_mem01->r0 = 1;
    ...
}
```

virtualised memory – private memory regions

32

- declare the data layout of the shared memory regions in `shared_memories/vep_private_memory.h`

```
typedef volatile struct {  
    uint32_t r0;  
    uint32_t r1;  
    char vheap[1000];  
} vep_private_mem01_t;  
#define USE_VEP_PRIVATE_MEM01  
  
typedef volatile struct {  
    uint32_t r0;  
    uint32_t r2;  
} vep_private_mem02_t;  
#define USE_VEP_PRIVATE_MEM02  
  
typedef volatile struct {  
    uint32_t r1;  
    uint32_t r2;  
} vep_private_mem12_t;  
#define USE_VEP_PRIVATE_MEM12  
  
// IMPORTANT: to use a private memory region you must:  
// 1- typedef the struct containing all data to be placed in the region  
// 2- uncomment the relevant #define USE_VEP_PRIVATE_MEM* below  
//     this will declare & initialise the vep_private_memXY external variables  
// 3- declare the memory region in the vep-config.txt file  
//     without this, the region will not be declared in the memory map  
//  
// all fields are set to 0 when the vep is loaded  
// (it may therefore be used to have an 'initialized' as first field in the struct,  
// this allows other veps can check if this vep is running and initialized or not)  
//  
// the private data of this vep on tile T in memory M is mapped to  
// VEP_PRIVATE_REGION_TILE<T>_MEM<M>_START
```

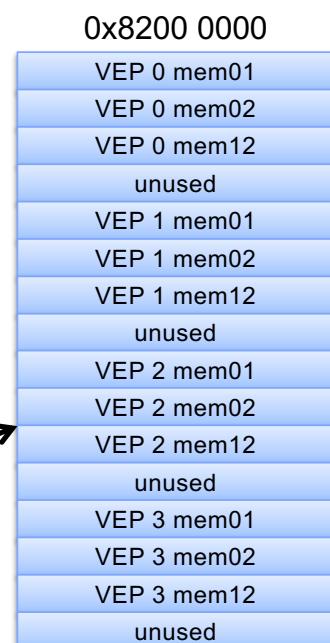
virtualised memory – public memory regions

advanced, can be skipped

33

- RISC-V – RISC-V communication memories: **mem01, mem02, mem12**
 - 32 KB = 0x8000 bytes = 8 Kword
 - each VEP can declare a **public memory region** in each shared memory that **all VEPs can access**, e.g.
 - in mem01 have 1K public memory starting at 1K
 - in mem02 have 2K public memory starting at 11K
 - you have to manage the data layout in the public memory regions
 - the maximum size of a public region is 8K
 - the MMU remaps the private regions of VEPs to a global memory region
 - 8 KB per region, first order by memory index, then by VEP number:
 - a VEP can access its own public region in memory (01, 02, 12) starting at address **VEP_PUBLIC_REGION_MEM<M>_START**
 - to access the public region of any VEP v in memory m

```
#define VEP_PUBLIC_REGION_START(vep,memindex) ...
#define SHARED_MEMORY_MEM01_INDEX 0
#define SHARED_MEMORY_MEM02_INDEX 1
#define SHARED_MEMORY_MEM12_INDEX 2
– e.g. VEP_PUBLIC_REGION_START(VEP_ID/*2*/, SHARED_MEMORY_MEM12_INDEX)
```



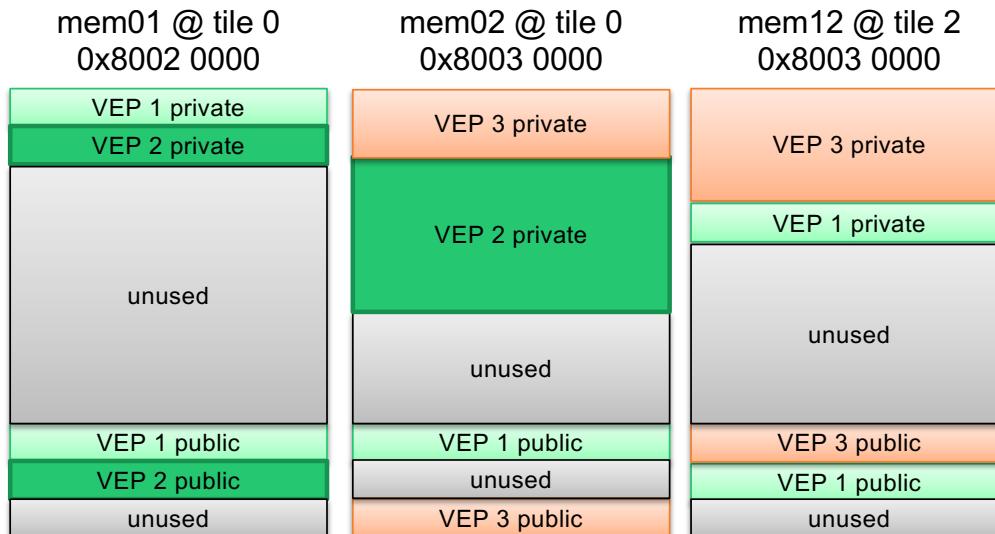
virtualised memory – private & public memory regions

advanced, can be skipped

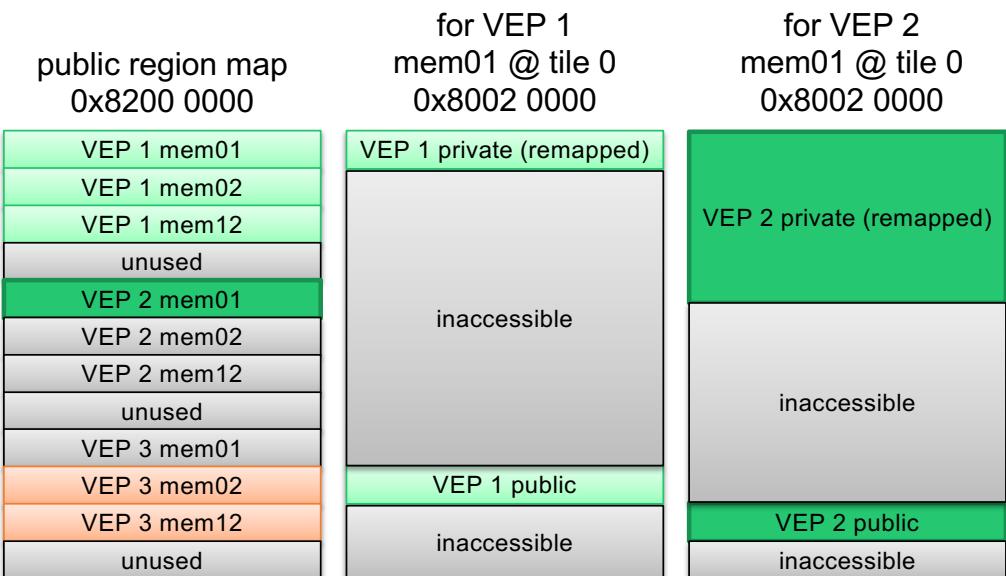
34

- VEP 1 runs on tiles 0 & 2, has private memory on mem01/02, and public memory on mem01/02/12
- VEP 2 runs on tile 1, has private memory on mem02, and public memory on mem01
- VEP 3 runs on tiles 1 & 2, has private memory on mem01/12, and public memory on mem02/12

physical memory maps



logical memory maps



shared memory regions: make mmp

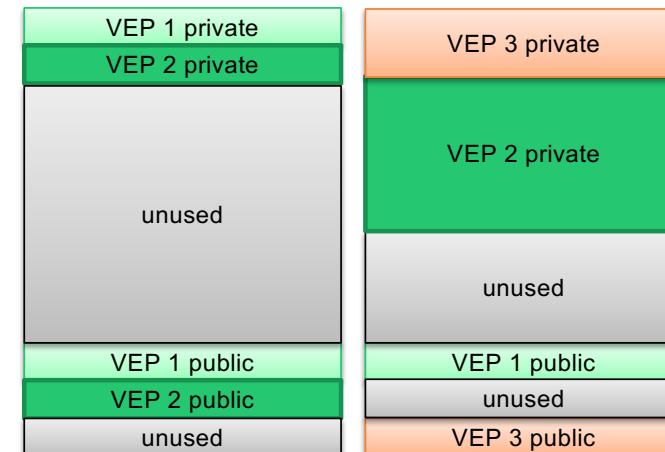
35

- in the top-level directory `make mmp / memorymap-physical`
displays the **physical** region layout in the shared memories, i.e. without remapping by the MMU

```
mem01 from 0K range 1K private for vep 1
mem01 from 1K range 2K private for vep 2
mem01 from 10K range 1K public for vep 5
mem01 from 12K range 1K public for vep 1
mem01 from 13K range 2K public for vep 2
mem01 from 15K range 1K public for vep 4
mem01 from 5K range 1K private for vep 4
mem02 from 0K range 1K private for vep 1
mem02 from 1K range 2K private for vep 2
mem02 from 10K range 1K public for vep 5
mem02 from 15K range 1K public for vep 4
mem02 from 5K range 1K private for vep 4
mem12 from 0K range 1K private for vep 1
mem12 from 1K range 1K private for vep 2
mem12 from 10K range 1K public for vep 5
mem12 from 12K range 4K public for vep 4
mem12 from 5K range 1K private for vep 4
```

i.e. the textual form of something like this
(NB the text and figure don't correspond)

mem01 @ tile 0 mem02 @ tile 0
0x8002 0000 0x8003 0000



shared memory regions: make mm

36

- in the top-level directory `make mm / memorymap` displays the **logical public & private memory regions**

memory mapping from shared memories into per-partition memory map:

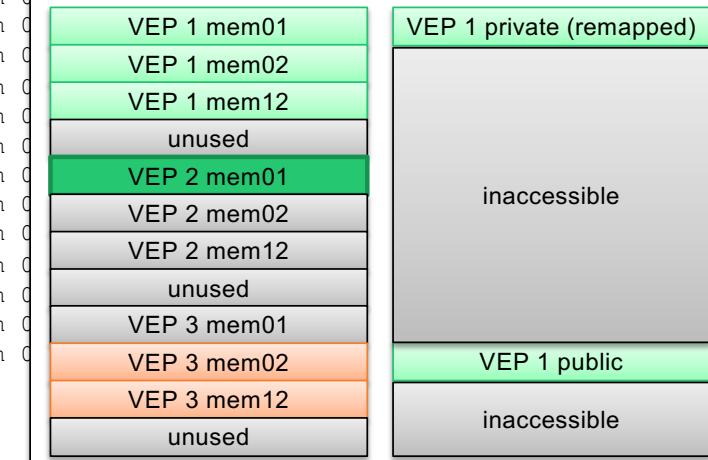
vep v's public region in memory m is mapped starting at $0x82000000 + (v * 4 + (\text{index of memory } m)) * 8K$

shared memory indexes starting at 0: [mem01,mem02,mem12]

logical	physical
on tile 0 partition 1 entry 3 map private region of vep 1 in mem01 to 0x80020000	from 0x80020000 (+ 0K) with mask 0xFFFFFC00 (1K)
on tile 0 partition 1 entry 4 map private region of vep 1 in mem02 to 0x80030000	from 0x80030000 (+ 0K) with mask 0xFFFFFC00 (1K)
on tile 0 partition 1 entry 5 map public region of vep 1 in mem01 to 0x82008000 (+ 24K)	from 0x80023000 (+ 12K) with mask 0xFFFFE000 (8K)
on tile 0 partition 1 entry 6 map public region of vep 2 in mem01 to 0x82010000 (+ 48K)	from 0x80023400 (+ 13K) with mask 0xFFFFE000 (8K)
on tile 0 partition 1 entry 7 map public region of vep 4 in mem01 to 0x82020000 (+ 96K)	from 0x80023C00 (+ 15K) with mask 0xFFFFE000 (8K)
on tile 0 partition 1 entry 8 map public region of vep 4 in mem02 to 0x82022000 (+104K)	from 0x80023D00 (+ 16K) with mask 0xFFFFE000 (8K)
on tile 0 partition 2 entry 3 map private region of vep 2 in mem01 to 0x80020000	from 0x80020000 (+ 0K) with mask 0xFFFFFC00 (1K)
on tile 0 partition 2 entry 4 map private region of vep 2 in mem02 to 0x80030000	from 0x80030000 (+ 0K) with mask 0xFFFFFC00 (1K)
on tile 0 partition 2 entry 5 map public region of vep 1 in mem01 to 0x82008000 (+ 24K)	from 0x80023000 (+ 12K) with mask 0xFFFFE000 (8K)
on tile 0 partition 2 entry 6 map public region of vep 2 in mem01 to 0x82010000 (+ 48K)	from 0x80023400 (+ 13K) with mask 0xFFFFE000 (8K)
on tile 0 partition 2 entry 7 map public region of vep 4 in mem01 to 0x82020000 (+ 96K)	from 0x80023C00 (+ 15K) with mask 0xFFFFE000 (8K)
on tile 0 partition 2 entry 8 map public region of vep 4 in mem02 to 0x82022000 (+104K)	from 0x80023D00 (+ 16K) with mask 0xFFFFE000 (8K)
on tile 0 partition 3 entry 3 map private region of vep 4 in mem01 to 0x80020000	from 0x80020000 (+ 0K) with mask 0xFFFFFC00 (1K)
on tile 0 partition 3 entry 4 map private region of vep 4 in mem02 to 0x80030000	from 0x80030000 (+ 0K) with mask 0xFFFFFC00 (1K)
on tile 0 partition 3 entry 5 map public region of vep 1 in mem01 to 0x82008000 (+ 24K)	from 0x80023000 (+ 12K) with mask 0xFFFFE000 (8K)
on tile 0 partition 3 entry 6 map public region of vep 2 in mem01 to 0x82010000 (+ 48K)	from 0x80023400 (+ 13K) with mask 0xFFFFE000 (8K)
on tile 0 partition 3 entry 7 map public region of vep 4 in mem01 to 0x82020000 (+ 96K)	from 0x80023C00 (+ 15K) with mask 0xFFFFE000 (8K)
on tile 0 partition 3 entry 8 map public region of vep 4 in mem02 to 0x82022000 (+104K)	from 0x80023D00 (+ 16K) with mask 0xFFFFE000 (8K)
on tile 1 partition 1 entry 3 map private region of vep 1 in mem01 to 0x80020000	from 0x80020000 (+ 0K) with mask 0xFFFFFC00 (1K)
on tile 1 partition 1 entry 4 map private region of vep 1 in mem12 to 0x80030000	from 0x80030000 (+ 0K) with mask 0xFFFFFC00 (1K)
on tile 1 partition 1 entry 5 map public region of vep 1 in mem01 to 0x82008000 (+ 24K)	from 0x80023000 (+ 12K) with mask 0xFFFFE000 (8K)
on tile 1 partition 1 entry 6 map public region of vep 2 in mem01 to 0x82010000 (+ 48K)	from 0x80023400 (+ 13K) with mask 0xFFFFE000 (8K)
on tile 1 partition 1 entry 7 map public region of vep 4 in mem01 to 0x82020000 (+ 96K)	from 0x80023C00 (+ 15K) with mask 0xFFFFE000 (8K)
on tile 1 partition 1 entry 8 map public region of vep 4 in mem12 to 0x82024000 (+112K)	from 0x80023D00 (+ 16K) with mask 0xFFFFE000 (8K)
on tile 1 partition 2 entry 3 map private region of vep 2 in mem01 to 0x80020000	from 0x80020000 (+ 0K) with mask 0xFFFFFC00 (1K)
...	

i.e. the textual form of something like this
(NB the text and figure don't correspond)

public region map
0x8200 0000



shared memory regions: make mmp

37

- in the partition directories `make mm / memorymap` displays the region layout for **that partition of that VEP**
- i.e. its own private regions, and public regions of all VEPs

mem01	from 1K range 2K private for vep 2
mem01	from 12K range 1K public for vep 1
mem01	from 13K range 2K public for vep 2
mem01	from 15K range 1K public for vep 4
mem02	from 1K range 2K private for vep 2
mem02	from 15K range 1K public for vep 4

- `make mm` in the VEP directory shows the memory map for each of the partitions

virtualised versus virtual memory

38

- virtualised memory = address translation
 - translate virtual to physical addresses
- virtual memory
 - a memory management technique that provides an "idealized abstraction of the storage resources that are actually available on a given machine" which "creates the illusion to users of a very large (main) memory"
 - requires translation of virtual addresses to physical address
 - but must do much more, see e.g. https://en.wikipedia.org/wiki/Virtual_memory
- currently CompSOC does not offer virtual memory

ARM – RISC-V communication memories: mem0a, mem1a, mem2a

39

- reserved for ARM - RISC-V FIFO communication
- stdin & stdout of a partition are automatically synchronised for you
 - on ARM: read out with `readout.sh` on the terminal
 - on RISC-V: use `stdin` & `stdout`
- user-out & user-in FIFO channels
 - simple FIFO protocol to use these
 - built on top of C-HEAP
 - introduced in detail later
 - `on_arm_rcv_from_riscv_to_arm`
 - `on_arm_snd_from_arm_to_riscv`
 - `on_riscv_snd_from_riscv_to_arm`
 - `on_riscv_rcv_from_arm_to_riscv`

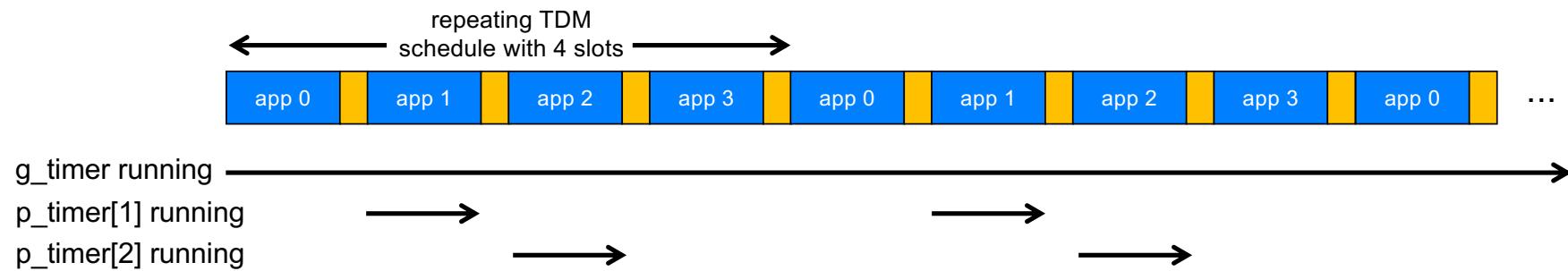
partition 0 is reserved for system application (you cannot use it) 4 KB	stdout FIFO
	stdin FIFO
	user out FIFO
	user in FIFO
	user data 892 B
	reserved
	reserved
	reserved
partition 1 1 KB	stdout FIFO
	stdin FIFO
	user out FIFO
	user in FIFO
	reserved
	reserved
	reserved
	reserved
partition 2 1 KB	stdout FIFO
	stdin FIFO
	user out FIFO
	user in FIFO
	reserved
	reserved
	reserved
	reserved
partition 9 1 KB	stdout FIFO
	stdin FIFO
	user out FIFO
	user in FIFO
	reserved
	reserved
	reserved
	reserved

ARM physical addresses are for tile 0;
add 0x4000 for tile 1; add 0x8000 for tile 2;
note that there are user 8 partitions available on each tile

each RISC-V tile has 2 timers

40

- **global timer**
 - `volatile uint64_t *g_timer = (uint64_t*) TILE2_GLOBAL_TIMER;`
 - always running
 - can be used to compute the precise **elapsed time** (from start to finish) – i.e. the **response time** (RT)
 - wall timers on all tiles are identical (it is as if it is a single timer for all tiles)
- **partition timer** (virtualised per partition)
 - `volatile uint64_t *p_timer = (uint64_t*) TILE2_PARTITION_TIMER;`
 - runs only when the partition is running
 - i.e. in all TDM slots allocated to this partition on this core
 - the precise number of clock cycles that the partition has executed – i.e. the **execution time** (ET)



atomicity (we'll revisit this in the FIFO lecture)

41

- simultaneous access of the same memory location in a shared memory from two processors
- read/write to an aligned word of 32 bits is atomic, even on the same address on a dual-port memory
 - write + read in same cycle → you get always the value before writing
 - write + write in same cycle → undefined behaviour, avoid it!
- this also holds for read/write of different or same byte in the same word
- non-word-aligned reads/writes of >1 byte are split up into multiple reads/writes and are NOT atomic
- see also the C lecture on structure packing

in the CompSOC platform,
not true in general

terminology

42

- RISC-V core
- dual-core ARM processor
- tile: RISC-V core + local bus + local peripherals
- microkernel / VKERNEL
- system application
- partition: one executable on one core
- virtual execution platform (VEP): one or more partitions (on one or more tiles)
- one application per VEP

- global timer
- partition timer

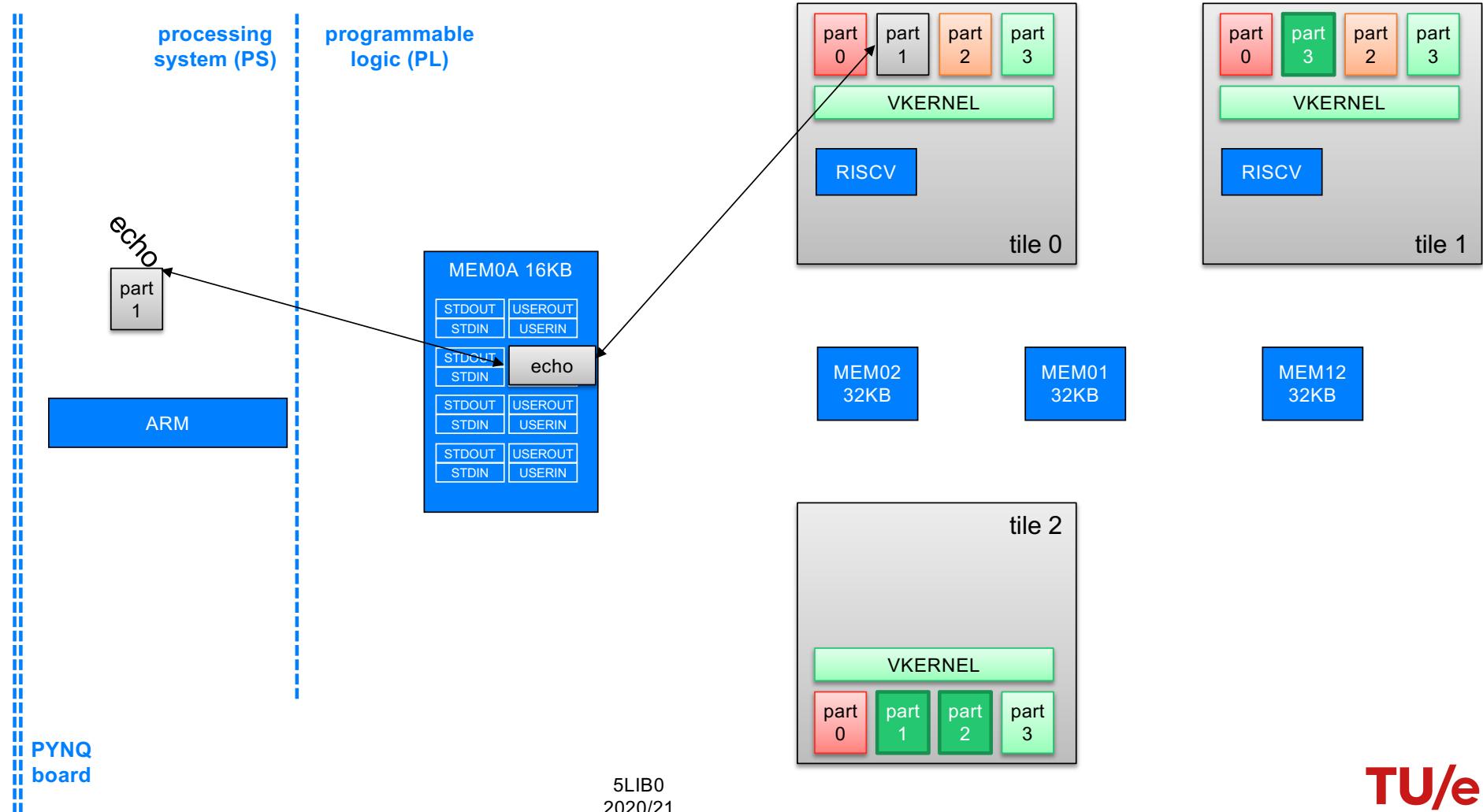
terminology

43

- RISC-V core
- dual-core ARM processor
- tile: RISC-V core + instruction/data memory + peripheral bus + peripherals
- partition: one executable on one core, with resources:
 - one region in instruction/data memory
 - zero or more TDM slots
 - stdin/out & user-in/out FIFOs in the ARM-RISC-V memory (arm0/1/2a)
- virtual execution platform (VEP) consists of one or more partitions on one or more tiles, with resources:
 - at most one private region per shared memory (mem01, 02, 12)
 - at most one public region per shared memory (mem01, 02, 12)
- one application per VEP
 - an application could consist of multiple VEPs, but we won't consider that possibility here
- system application
- microkernel / VKERNEL
- global timer
- partition timer

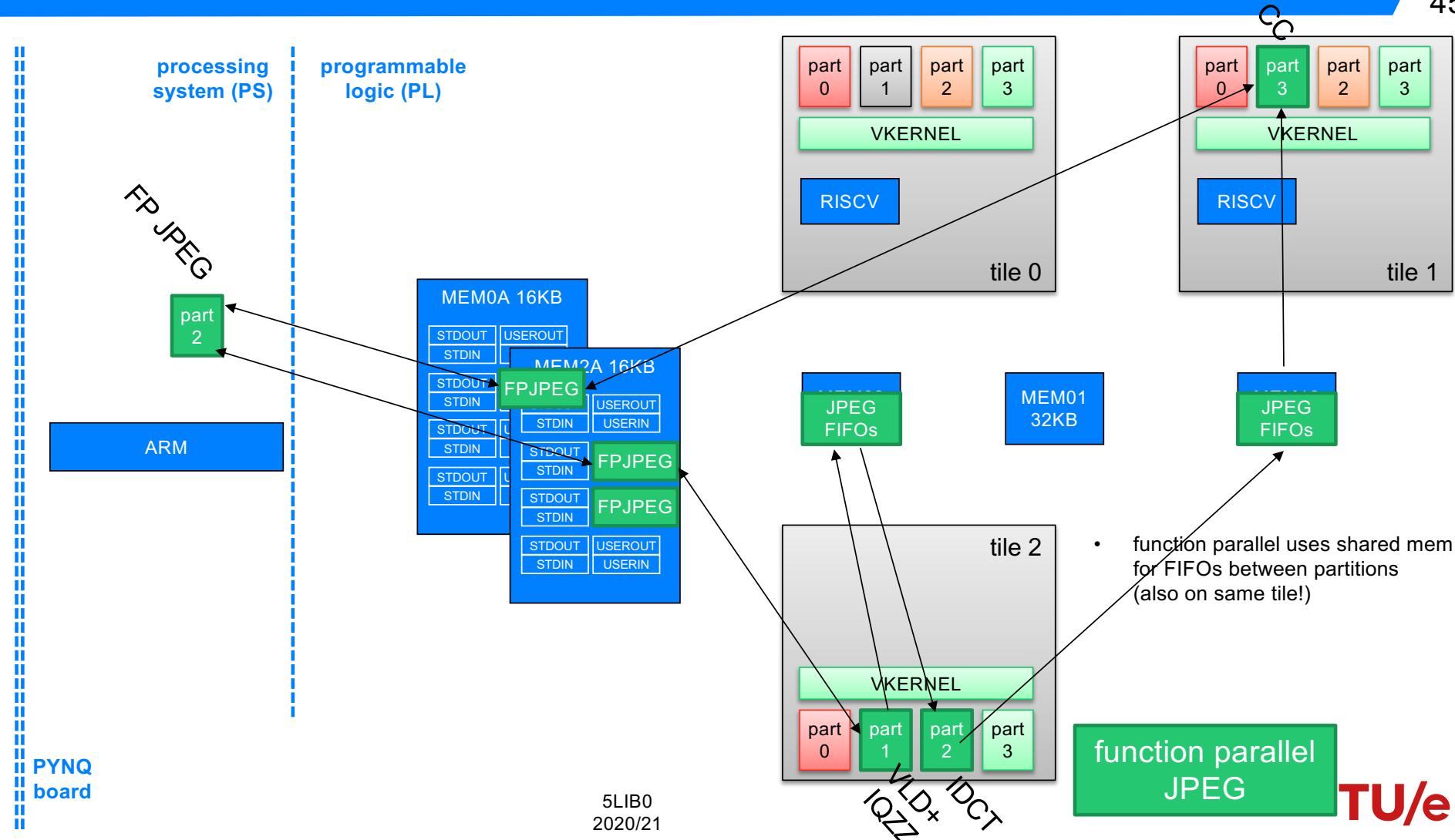
VEP, partition, tile

44



VEP, partition, tile

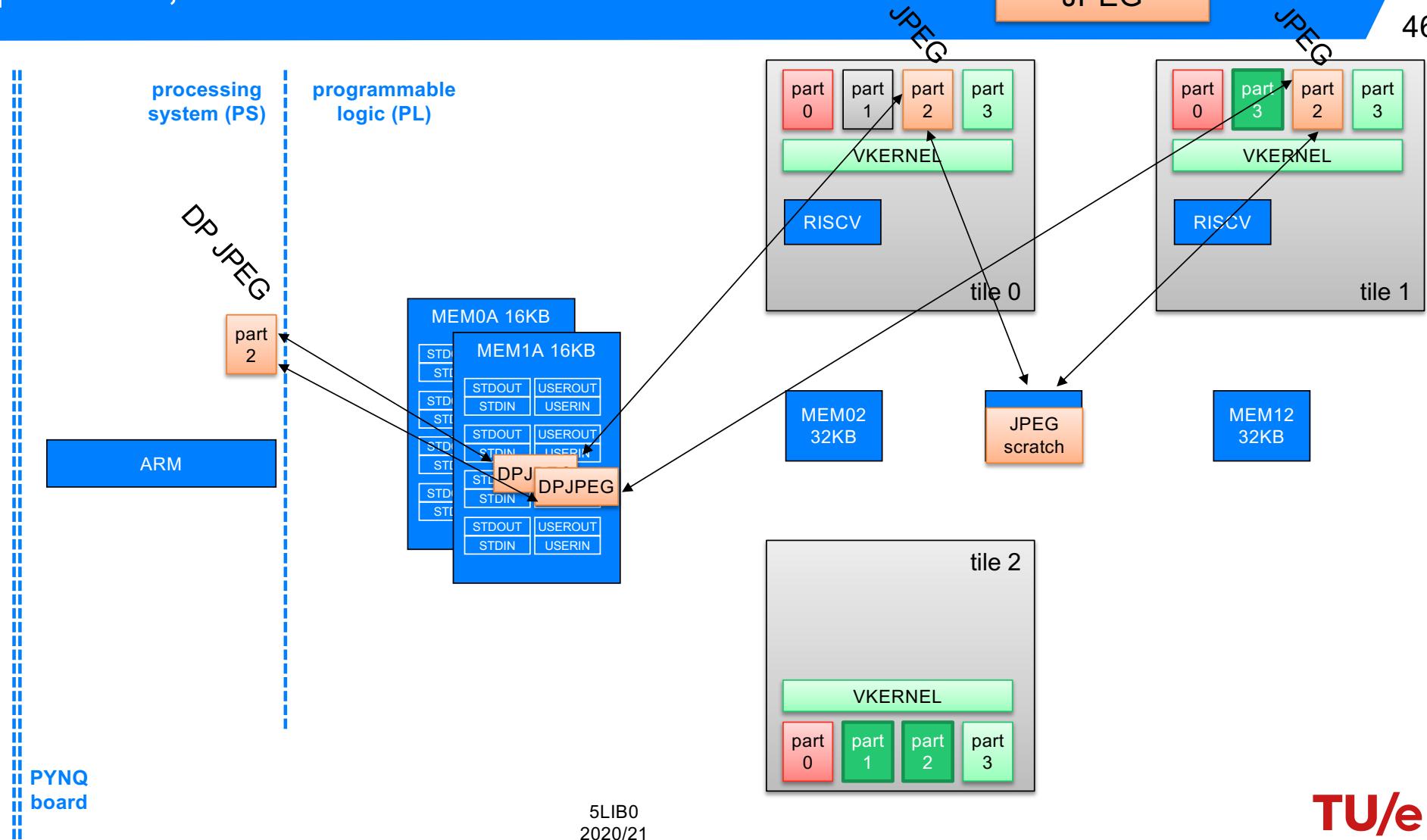
45



VEP, partition, tile

data parallel
JPEG

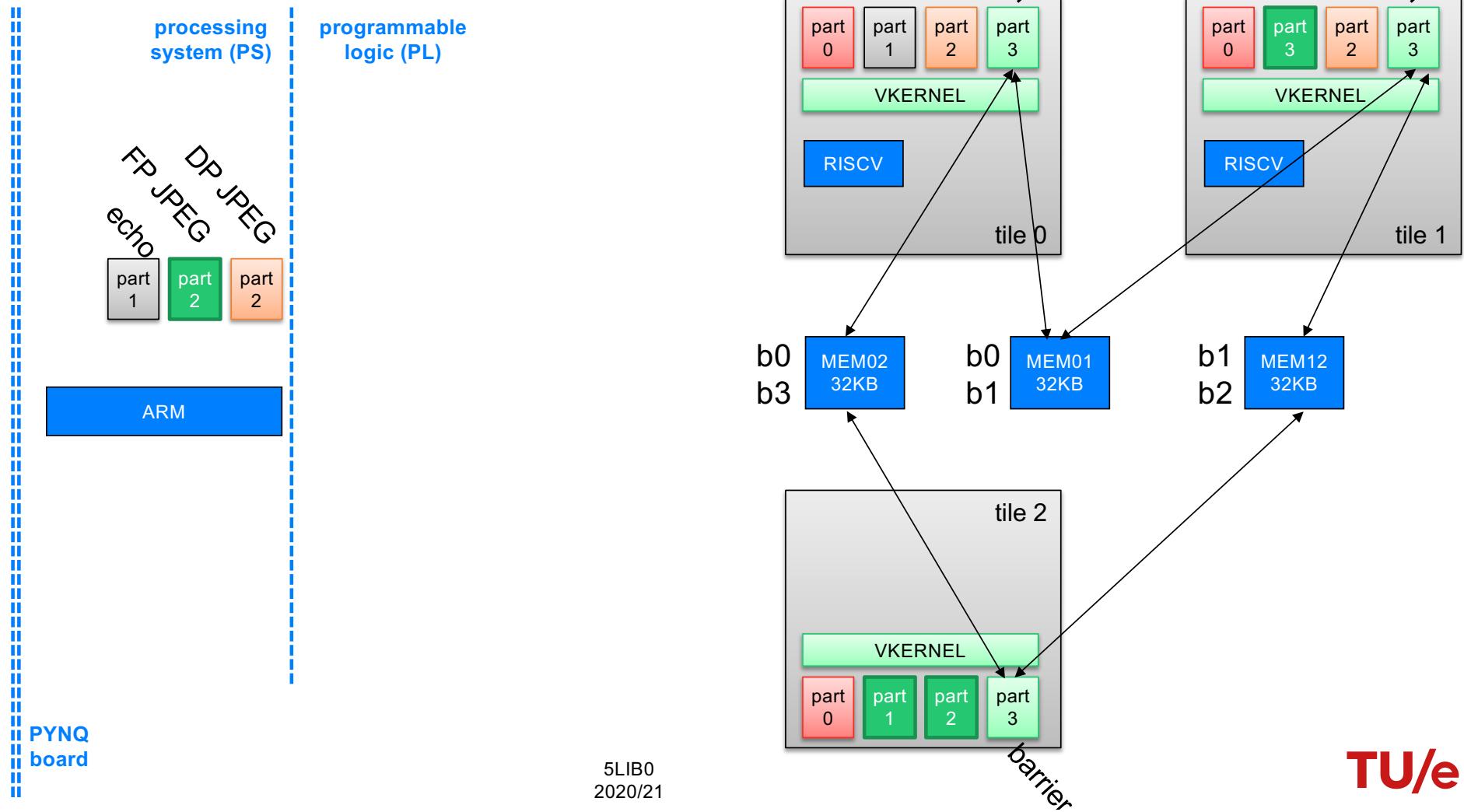
46



VEP, partition, tile

barrier

barrier
47



VEP, partition, tile

