HOCHSCHULE COBURG

Beyond Microkernels – Hardware Abstraction and Virtualization for Specific Use Cases

Prof. Dr. Michael Engel
Hochschule Coburg
Fakultät Elektrotechnik und Informatik
Labor für Mikrocomputer und Digitale Signalverarbeitung

Motivation

Memory is a crucial component of computer systems

- Increasing sizes required due to application demands
- Large DRAMs required even in small systems
  - Image and audio processing, streaming data, …

New non-functional criteria relevant in addition to performance

- Power/energy consumption, fault tolerance, security, …
- Multi-criterial optimizations required

No longer “as good as possible”

- Rather try to be as good as possible under given constraints

Lost in Abstractions...

Memory abstractions are lossy

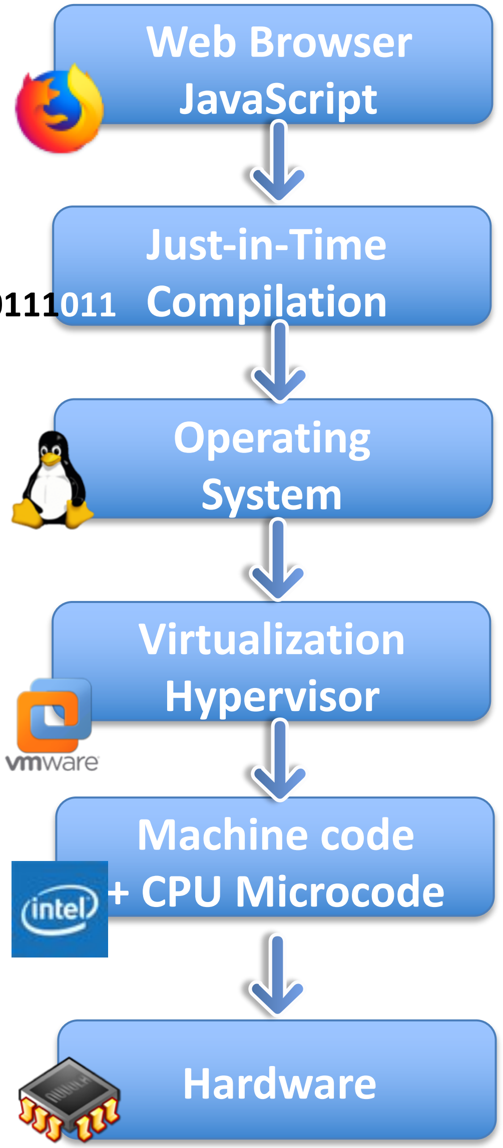
- For C, memory is just an array of bytes!

Memory allocation is a distributed task

- Global data – linker
- Local (stack) data – compiler/OS (stack init)
- Heap data – runtime/OS

Can we give programmers more control over memory allocation?

- ...while requiring as little detail knowledge about the hardware as possible

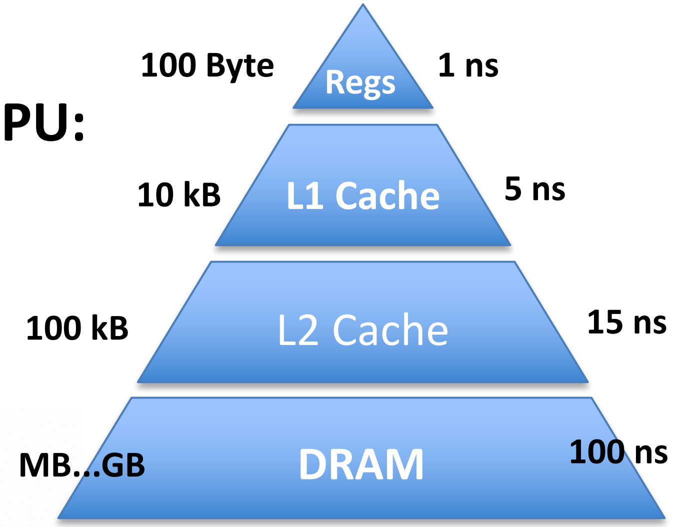


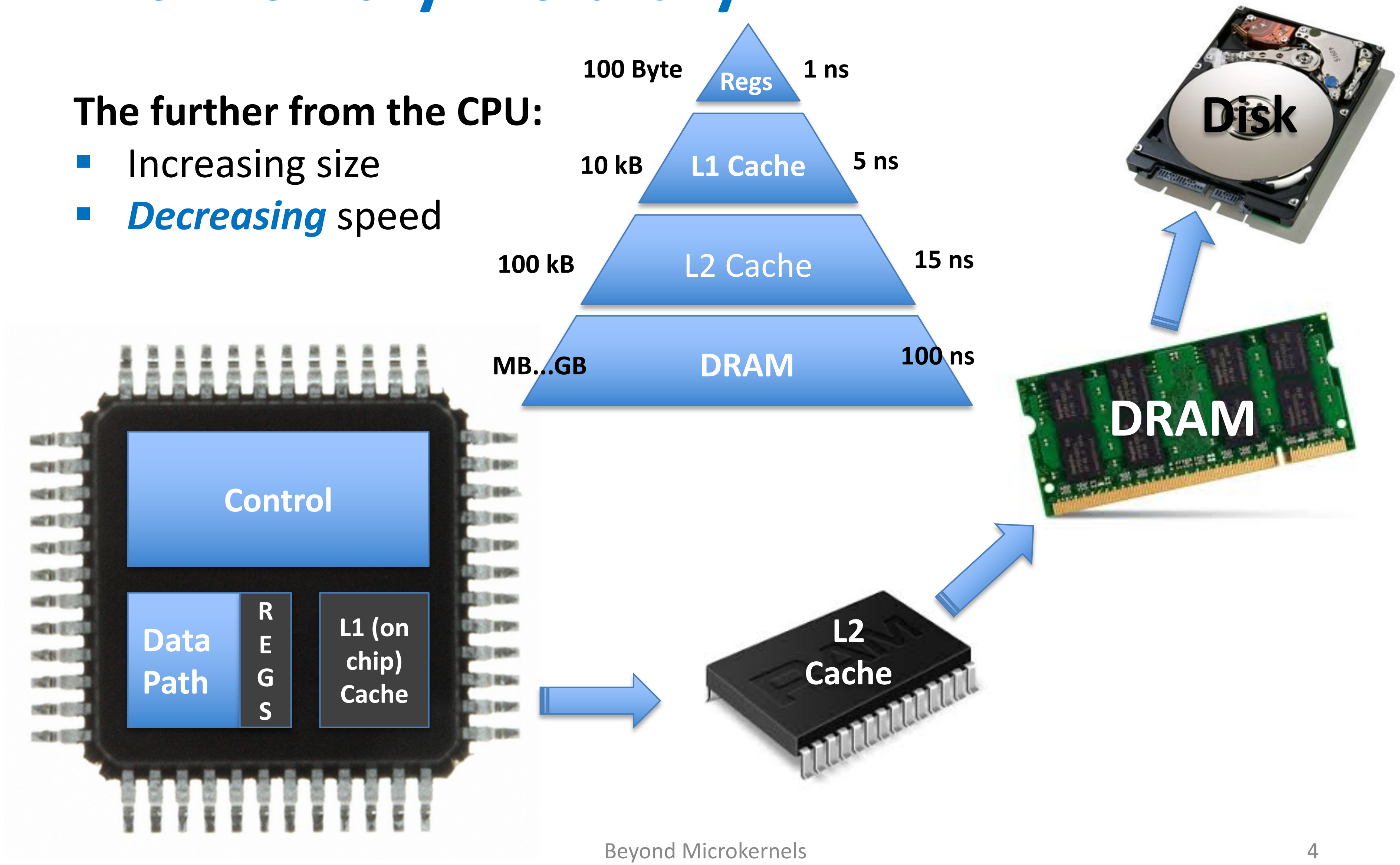
HOCHSCHULE COBURG

The Memory Hierarchy

The further from the CPU:

- Increasing size
- ***Decreasing*** speed





Non-functional properties of memories (1)

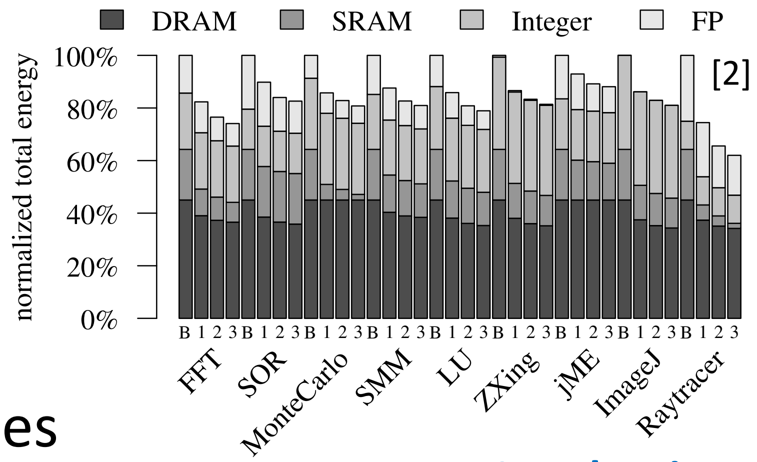
HOCHSCHULE COBURG

Memory has a large influence on non-functional properties of a system

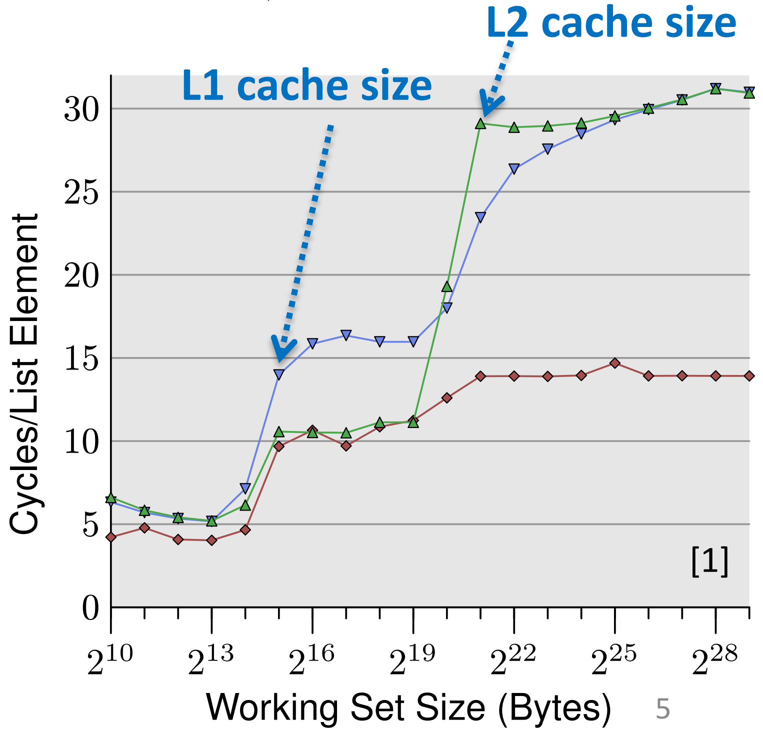
- Average, best, and worst case performance, throughput and latencies
- Power and energy consumption
- Reliability and security

Non-functional properties depend on many parameters of memory, e.g.

- Cache architecture
- Memory type
- Alignment and aliasing of data



[2]



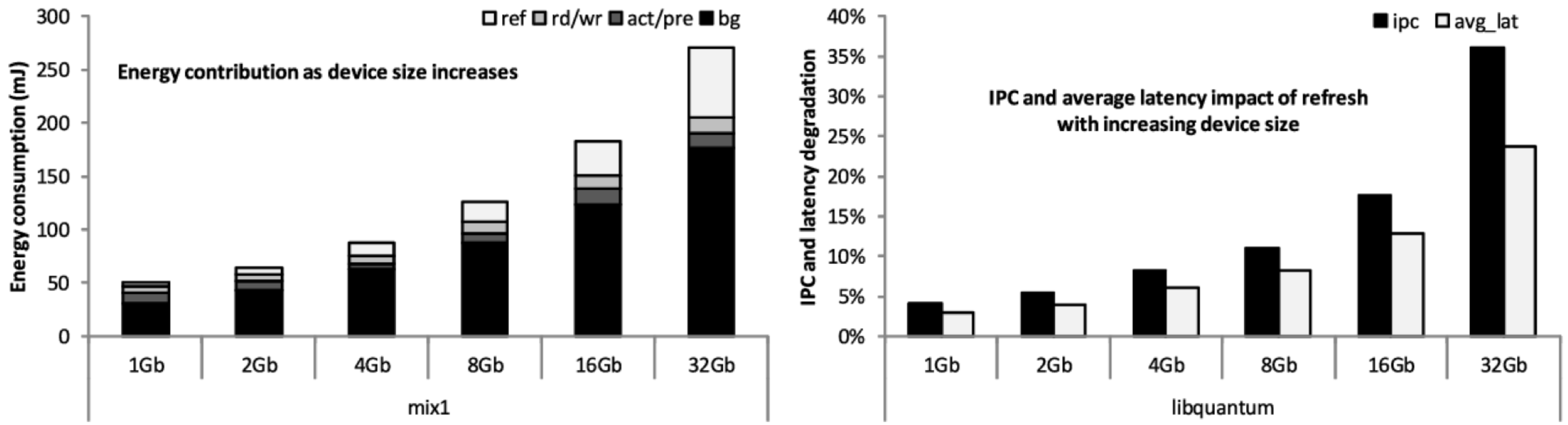
[1]

Non-functional properties of memories (2)

Impact of memory size and refresh on energy consumption

- Growing share of energy consumption and latency due to requirements of DRAM refresh

DRAM device trends. Both speed and size increase with each DDR generation.



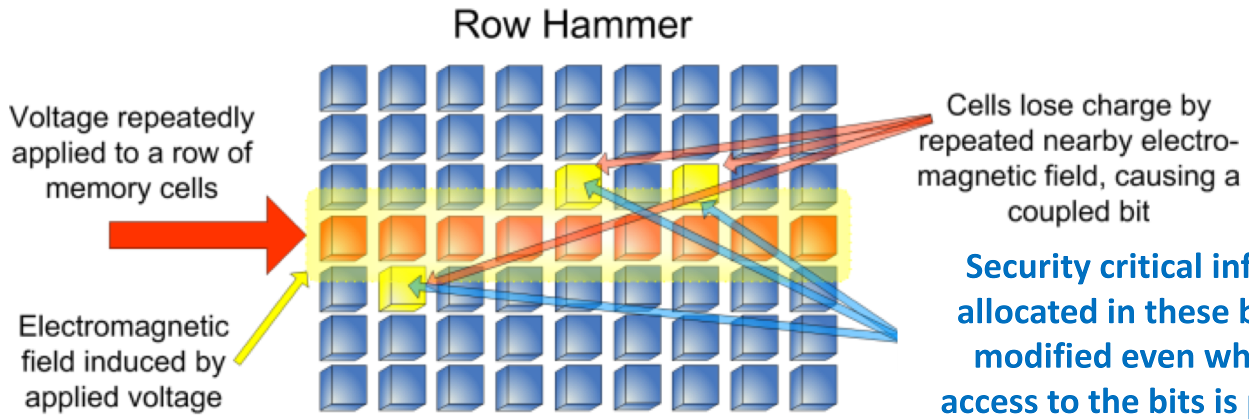
DRAM evolution and non-functional properties [3]

HOCHSCHULE COBURG

Non-functional properties of memories (3)

Impact of memory technology on system security

- Rowhammer security attack: unintended side effect in dynamic random-access memory (DRAM) [12]
- Causes memory cells to leak their charges and interact electrically between themselves
  - possibly leaking the contents of nearby memory rows that were not addressed in the original memory access

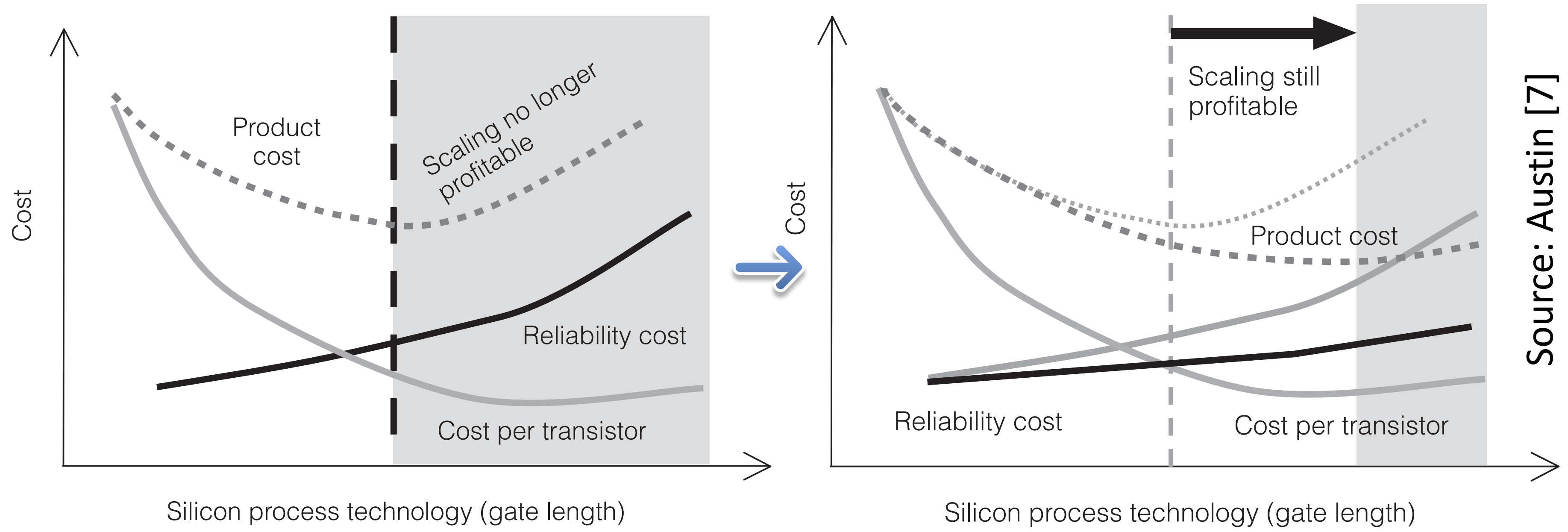

Row Hammer
Voltage repeatedly applied to a row of memory cells
Electromagnetic field induced by applied voltage
Cells lose charge by repeated nearby electro-magnetic field, causing a coupled bit

Security critical information allocated in these bits can be modified even when direct access to the bits is prohibited!

HOCHSCHULE COBURG

Trends in Memory Reliability

- Shrinking structure sizes and reduced supply voltages
  ⇒ Increased memory error rates, new error types (multiple bit errors)
- Traditional HW-based FT approaches
  ⇒ more hardware for error detection and correction required (e.g., ECC)
- Profitability ends if *cost(additional HW)* > *gain(new technology)*



Source: Austin [7]

Research Projects Related to Memory

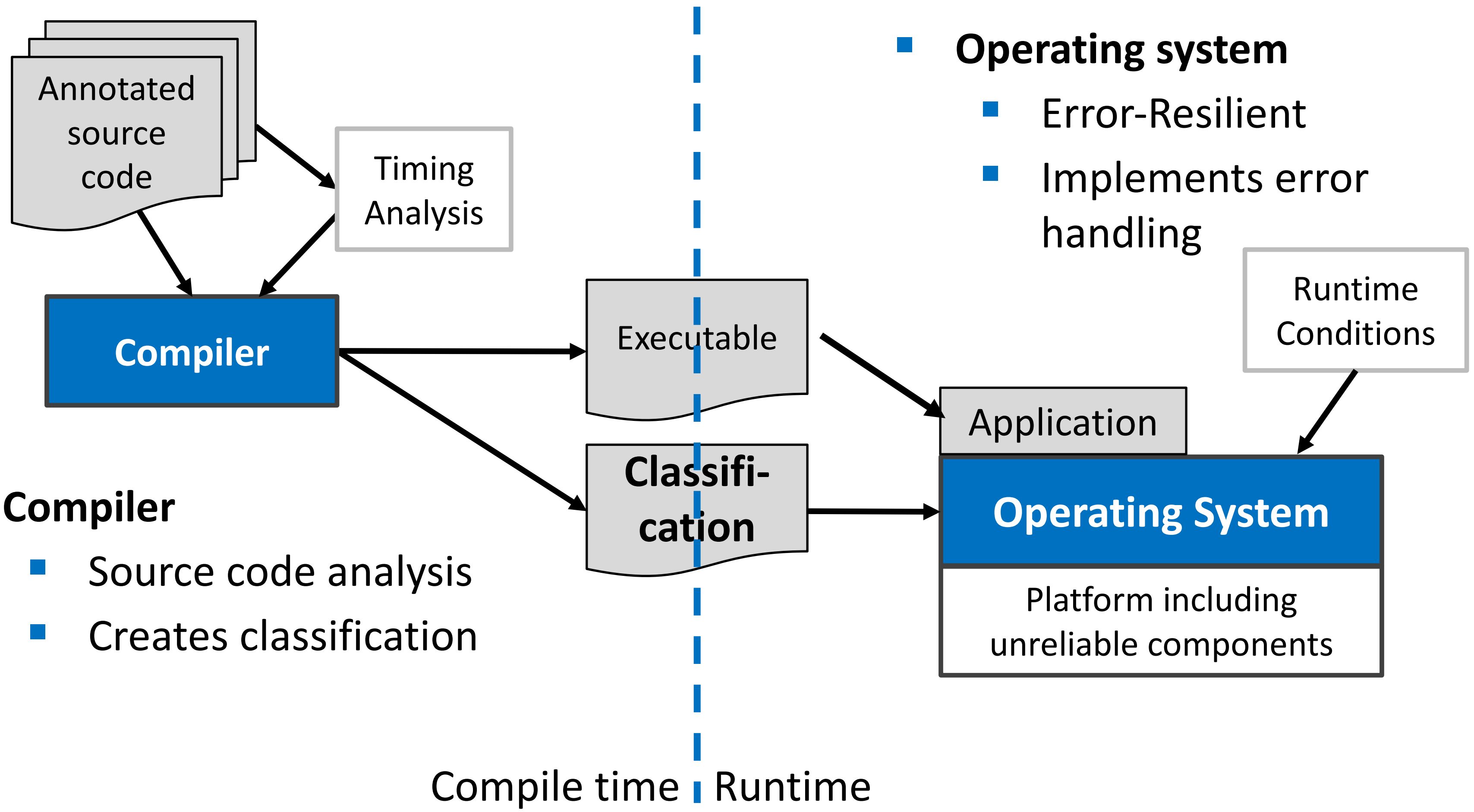
FEHLER (2010–2016) [8,9,10]

- Introduce flexible memory fault tolerance to embedded systems
- Statically classify relevance of data objects on application level
- Only correct fatal errors, handle errors with impact on QoS (Silent Data Corruption, SDC) on a best-effort basis to conserve runtime, energy, etc.
- Joint work with Andreas Heinig, Florian Schmoll and Peter Marwedel

RAMpage (2011–2013) [5,6]

- Automatic detection of permanent memory errors at runtime on Linux
- Live remapping of affected memory pages, handling of affected processes
- Increase system life- and uptime of systems
- Ecological impact: continue to use devices with soldered RAM
- Joint work with Horst Schirmeier, Ingo Korb and Jens Neuhalfen

FEHLER High-Level View



- **Operating system**
  - Error-Resilient
  - Implements error handling

- **Compiler**
  - Source code analysis
  - Creates classification

FEHLER Compiler-OS Interaction

Application knowledge provided by annotations

- Impact of errors
- Urgency of error correction
- Feasible correction methods

```
unreliable int j;
reliable   int control;
```

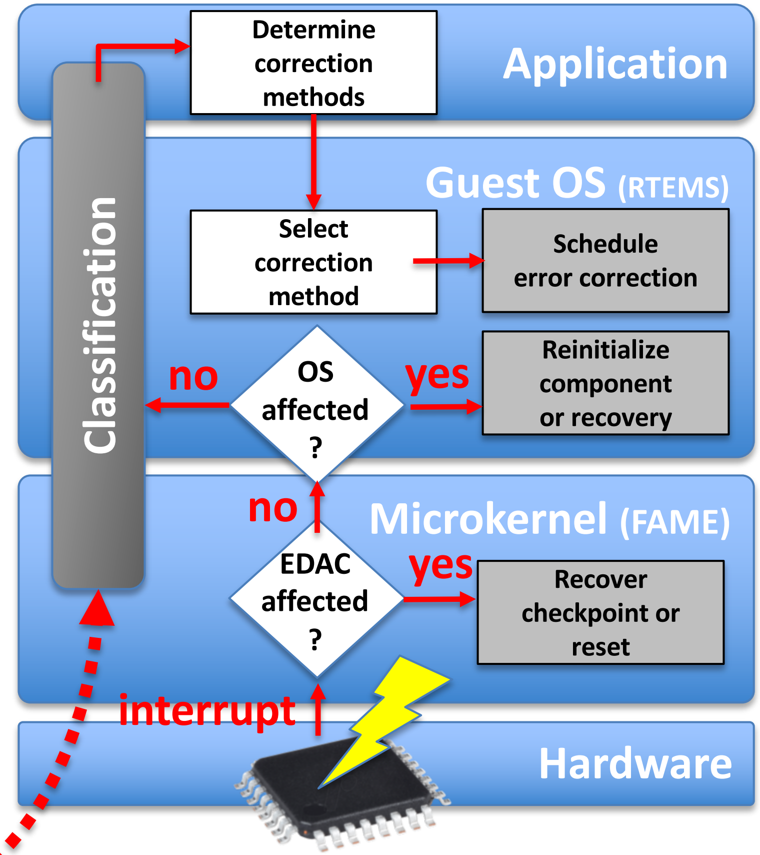
Propagation of annotations and inference of error classes

```
reliable int y = control;
```

Encoding of classificiation

01101101010110110110101101010101
10110110101101101010110101101011

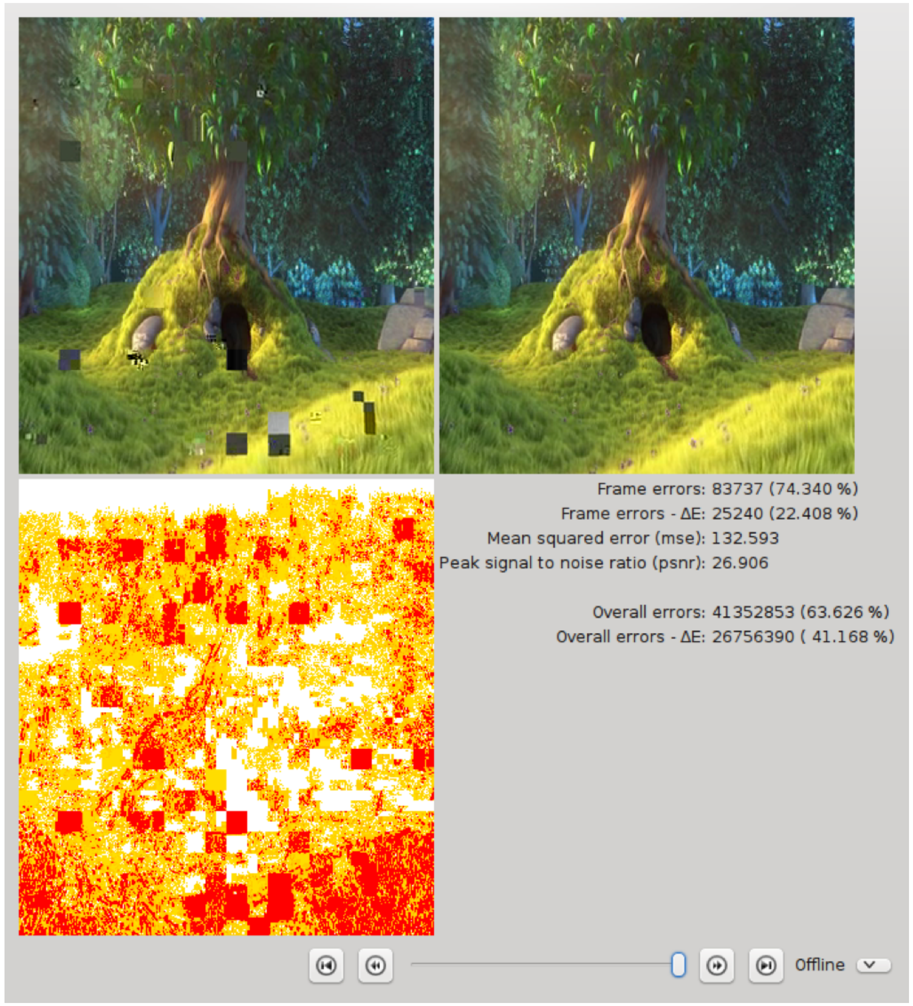
Compile time | Runtime



Paravirtualization-based microkernel environment ➛ keep EDAC running

FEHLER Use Cases

- H.264 video decoder
  - ca. 3500 LoC ANSI-C
- ARM926 simulation and real HW platform
- Assess error impact using QoS analysis tool
- Decoding with errors (upper left) and correctly decoded video frame (upper right)
- Compared using various metrics (lower half)
- Example: low error injection rate
- Few visible error impacts
- Metrics indicate many more that are not discernible



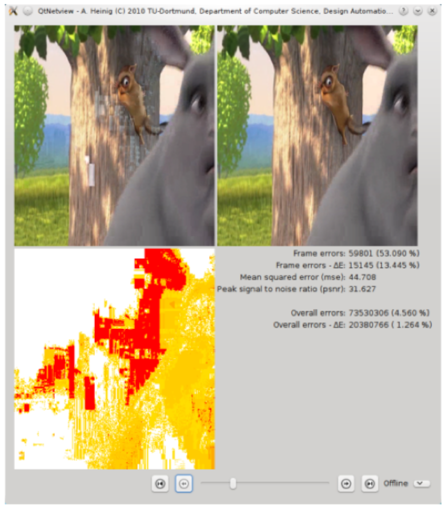
```
                Frame errors: 83737 (74.340 %)
          Frame errors - ΔE: 25240 (22.408 %)
         Mean squared error (mse): 132.593
Peak signal to noise ratio (psnr): 26.906

         Overall errors: 41352853 (63.626 %)
   Overall errors - ΔE: 26756390 ( 41.168 %)
```

FEHLER Results

- H.264 video decoder, different videos & resolutions
- **No application crashes due to hardware errors!**
- Significant amount of memory can remain unprotected:

Resolution	Memory size of reliable data	Memory size of unreliable data
176 x 144	90 kB (55%)	74 kB (45%)
352 x 288	223 kB (43%)	297 kB (57%)
1280 x 720	1 585 kB (37%)	2 700 kB (63%)


QtNetview - A. Heinig (C) 2010 TU-Dortmund, Department of Computer Science, Design Automatio...
Offline

Frame errors: 59801 (53.090 %)
Frame errors - ΔE: 15145 (13.445 %)
Mean squared error (mse): 44.708
Peak signal to noise ratio (psnr): 31.627

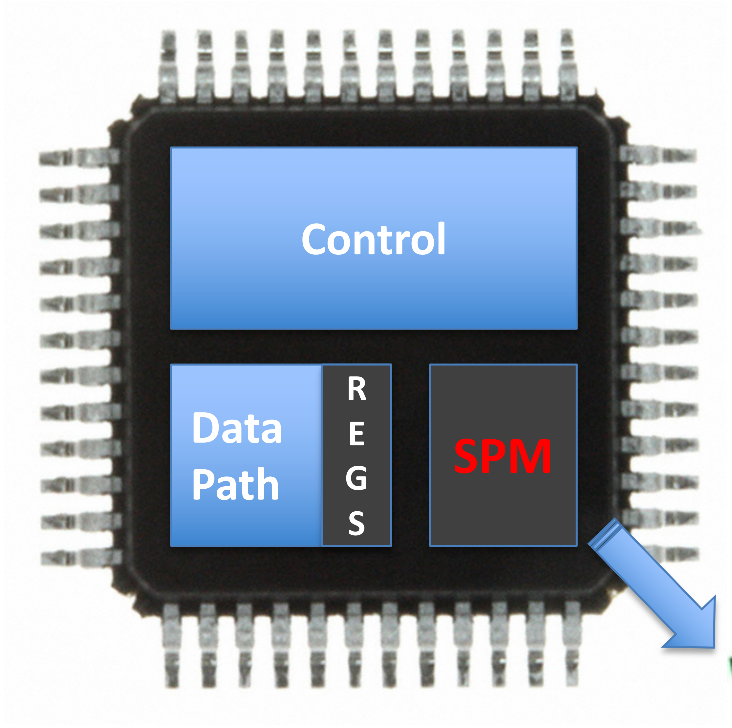
Overall errors: 73530306 (4.560 %)
Overall errors - ΔE: 20380766 (1.264 %)

- QoS impact of uncorrected errors:

Injection into unreliable memory	240 errors / s	80 errors / s
Average PSNR of frames with errors	40.89 dB	50.57 dB

Beyond FEHLER: Enable Software Control of Memory

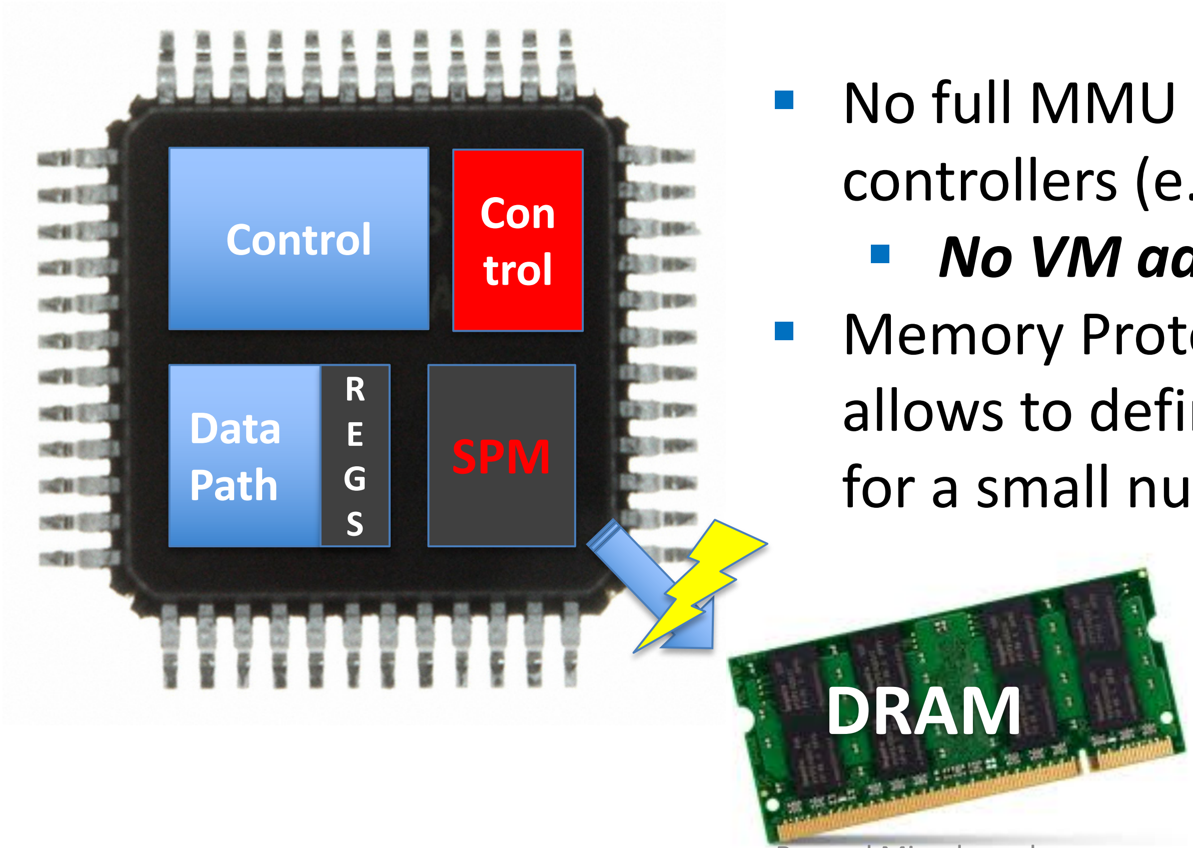
- **Can we build an architecture covering multiple use cases?**
- Idea: use on-chip scratchpad memories (SPM/TCM)



- SPM = small, fast, energy-efficient on-chip static RAM (SRAM)
  - Hard(er) to extract information
  - Optional: reduced overhead for protection against bit flips
- Only store ***protected*** information in external DRAM
  - e.g. using software-based ECC & encryption

Software-Defined Memory

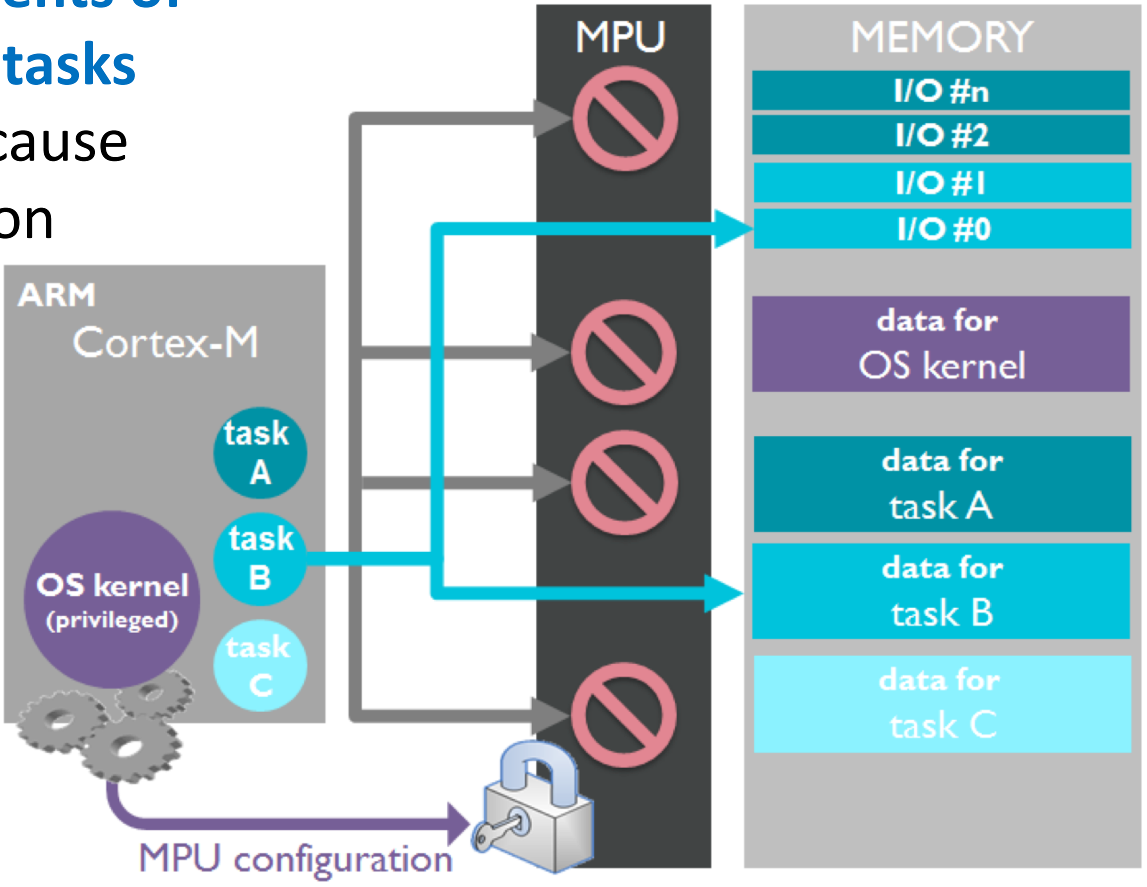
- **Applications can only access SPM RAM directly**
- All other memory accesses are intercepted by the microkernel



- No full MMU available on small controllers (e.g. Cortex-M)
  - ***No VM address translation!***
- Memory Protection Unit (MPU) only allows to define access permissions for a small number of segments

ARM Cortex-M MPU

- **MPU defines segments of RAM accessible to tasks**
- All other accesses cause a memory protection exception
- MPU configuration can be changed on the fly (e.g. during a task switch)



HOCHSCHULE COBURG

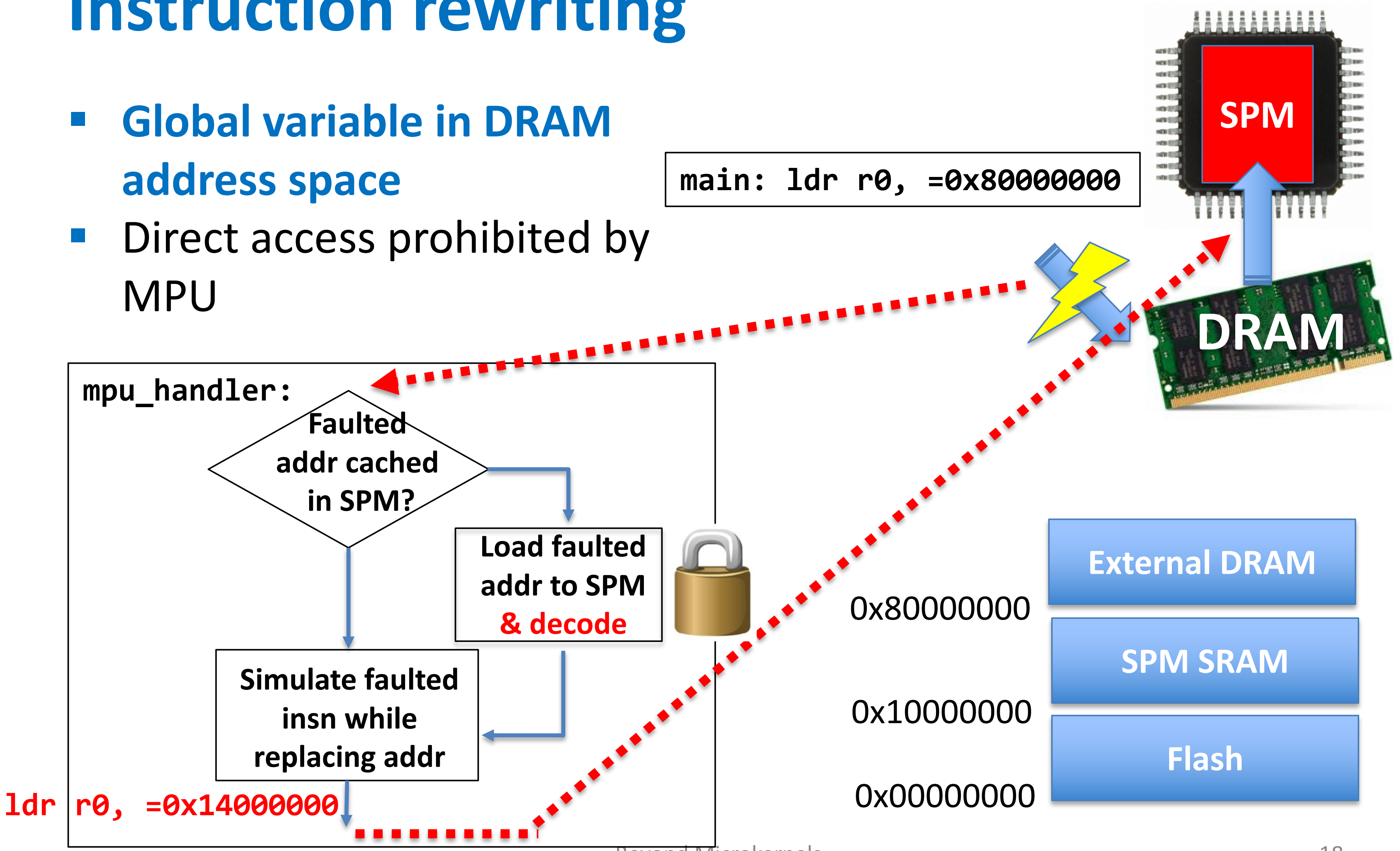
Software-Defined Memory: Protection Problems

- **Applications can only access SPM RAM directly**
- SPM is treated like a (software-controlled) cache

- **Problem:**
  - Applications are not expected to handle SPM contents directly **and** require more RAM than available in SPM
    - ***„Real“ DRAM memory addresses used by compiler***
  - However, the MPU does not perform address ***translation***
- **Two solution approaches:**
  - Rewrite addresses on the fly
  - Use additional level of indirection

Software-Defined Memory: Instruction rewriting

- **Global variable in DRAM address space**
- Direct access prohibited by MPU



```
main: ldr r0, =0x80000000
```

```
mpu_handler:
```

```
ldr r0, =0x14000000
```

SPM
DRAM
Faulted addr cached in SPM?
Load faulted addr to SPM & decode
Simulate faulted insn while replacing addr
External DRAM
SPM SRAM
Flash
0x80000000
0x10000000
0x00000000

Challenges of Software-Defined Memory

- **Rewriting or indirection?**
  - **Indirection** uses pointers to pointers
    => Easy to adapt accesses, no exception once „fixed“,
    - Runtime overhead for every load/store instruction
    - Compiler backend modifications required
  - **Instruction rewriting** faults all load/store accesses to DRAM
    - Cost of exception handling + rewriting
- **Memory management service in microkernel**
  - Requires efficient control of SPM contents
    - Relation to (embedded) garbage collectors?
  - Performs encoding/encryption & decoding/decryption in software => efficient implementation?

Conclusion

- **Memory properties used for allocation and data flow**
  - Tight control of memory behavior helps to reduce hardware overhead and improve reliability
  - Similar handling of additional non-functional properties
    - E.g., refresh [3], allocation of rows, power-save modes

- **Basic design principle**
  - Perform as much analysis work as possible at compile time
  - Pass relevant meta data to runtime components
  - Optimize at runtime while considering additional constraints

References

[1] U. Drepper, What Every Programmer Should Know About Memory, RedHat Inc., 2007

[2] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, P. Marwedel, Scratchpad memory: design alternative for cache on-chip memory in embedded systems, Proceedings of the tenth international symposium on hardware/software codesign, 2002

[3] I. Bhati, M.-T. Chang, Z. Chishti, S.-L. Lu, B. Jacob, DRAM Refresh Mechanisms, Penalties, and Trade-Offs, IEEE Transactions on Computers Vol. 65, pp. 108-121, 2016

[5] H. Schirmeier, J. Neuhalfen, I. Korb, O. Spinczyk, M. Engel, Rampage: Graceful degradation management for memory errors in commodity linux servers, PRDC 2011

[6] H. Schirmeier, I. Korb, O. Spinczyk, M. Engel, Efficient online memory error assessment and circumvention for Linux with RAMpage, International Journal of Critical Computer-Based Systems 17 4 (3), 227-247

[7] D. Austin et al., Reliable Systems on Unreliable Fabrics, IEEE Design&Test 2008

[8] F. Schmoll et al., Improving the Fault Resilience of an H.264 Decoder using Static Analysis Methods, ACM TECS, 2013

[9] A. Heinig et al., Classification-based Improvement of Application Robustness and QoS in Probabilistic Computer Systems, ARCS'12 **Best Paper Award**

[10] A. Heinig et al., Using Application Knowledge to Improve Embedded Systems Dependability, HotDep 2011

[11] D. Cordes, M. Engel, O. Neugebauer, P. Marwedel, Automatic extraction of pipeline parallelism for embedded heterogeneous multi-core platforms, CASES 2013

[12] Y. Kim et al., Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors, ISCA'14