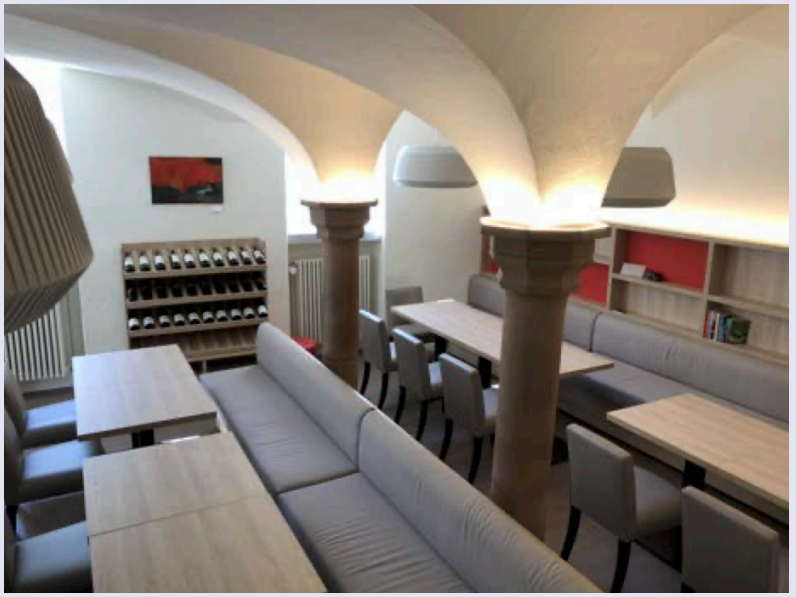
Universität Bamberg

Challenges Implementing Software-Defined Virtual Memory

Michael Engel (michael.engel@uni-bamberg.de)
Lehrstuhl für Praktische Informatik, insb. Systemnahe Programmierung
https://www.uni-bamberg.de/sysnap

Motivation



- Dagstuhl seminar "Power and Energy-aware Computing on Heterogeneous Systems" (PEACHES)

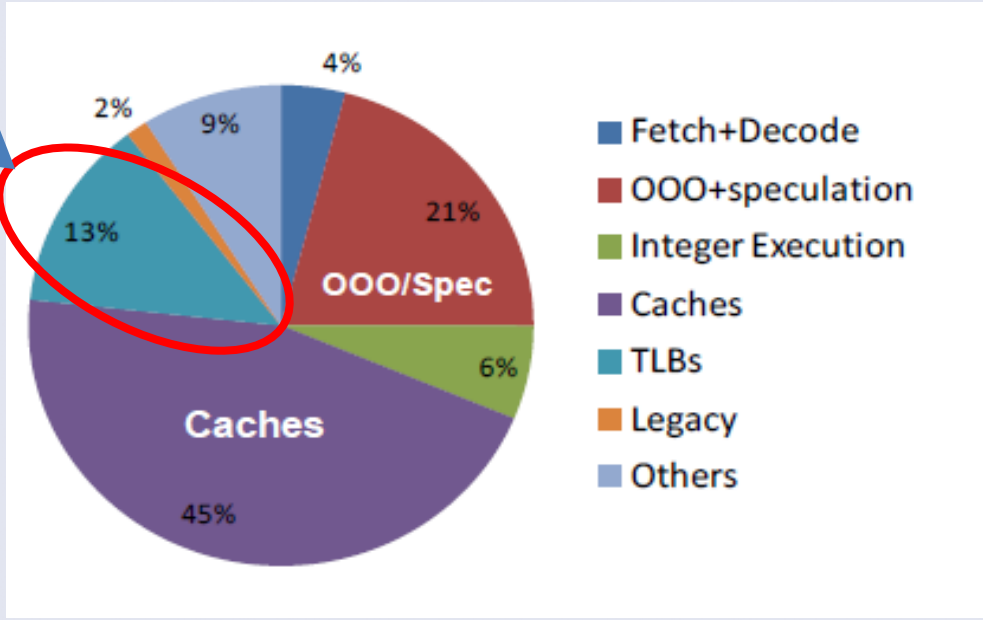
  https://www.dagstuhl.de/22341

- **Caution:** this is somewhere between a *WACI* (wild and crazy idea) and a *WIP* (work in progress)…

- Discussion:
  Which components contribute significantly to a computer's power and/or energy consumption?

Virtual memory today

- Virtual memory only feasible due to the use of a translation lookaside buffer (TLB)
  - cache for page table entries
- TLBs contribute significantly to power and energy consumption:
  - **~3-13% of "core" power** due to TLB [Sodani '11]
  - TLB contributes **6.6–13% on-chip memory's dynamic energy** [Basu '12]
  - TLB shows up as **hotspot** [Puttaswamy '06]

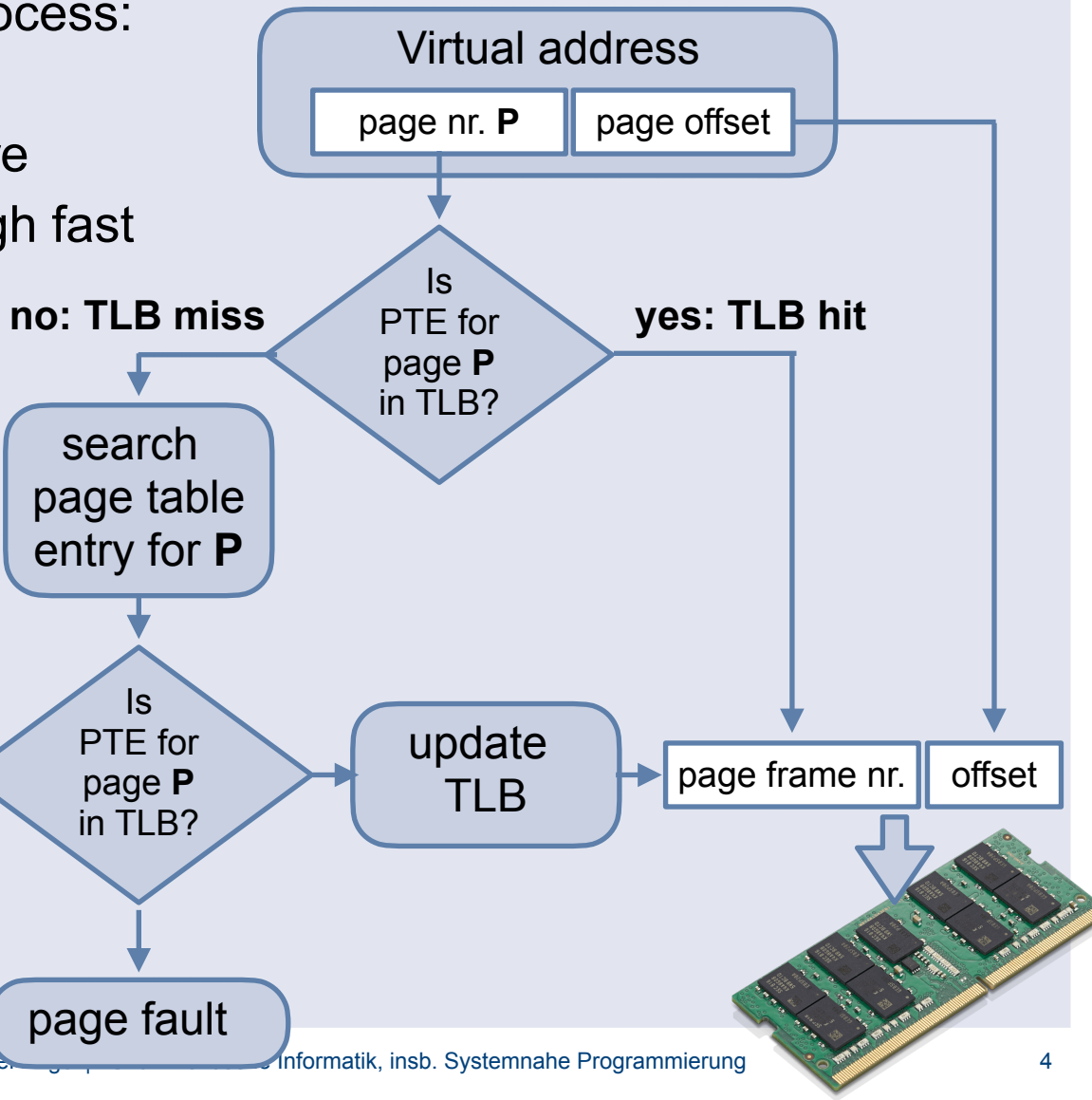


* From Sodani's /Intel's MICRO 2011 Keynote

Reminder: Virtual address translation

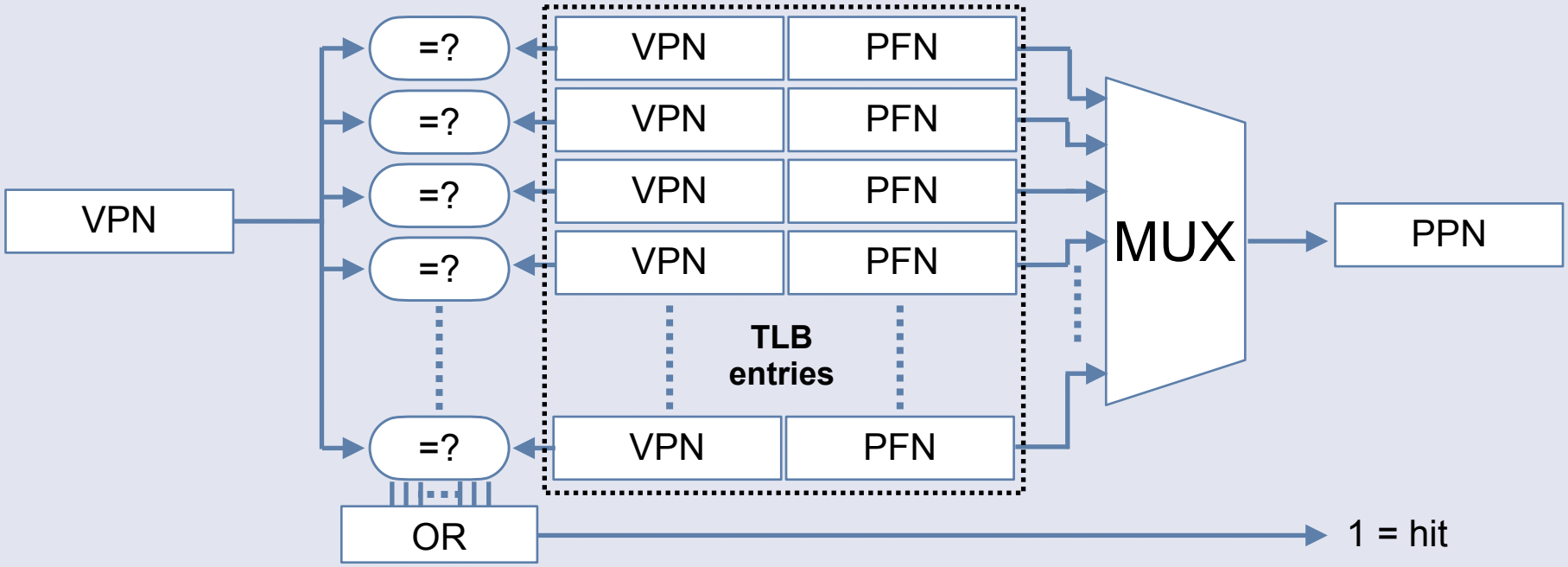
Virtual address translation process:

- Fast path: **TLB hit**
  - *critical path* in hardware
  - avoid slowdown through fast hit/miss detection
  - further latency hiding through cache (virt. addressed, phys. indexed)
- Slow path: **TLB miss**
  - page table entry in main memory
  - usually 2–5 levels deep page table
- Very slow path: **page fault**



Why are TLBs energy hungry?

- Fast path: **TLB hit:** critical path in hardware
  - avoid slowdown through **fast hit/miss detection**
- Typical number of TLB entries: 32–64
  - Required: determine hit/miss in single clock cycle
  - Solution: compare each TLB entry's page number to requested page number **in parallel**



Slow path: Software TLB miss handling

- Common solution: hardware page table walker as state machine
- Alternative: Software solution for the **slow path**
  - MIPS, Alpha, AMD29k, PA-RISC, optional for PPC, RISC-V [Nagle ’93], used e.g. in L4/MIPS [Heiser ’01]

- TLB miss raises exception
  - Hand-optimized fast code path to load TLB entry from page table

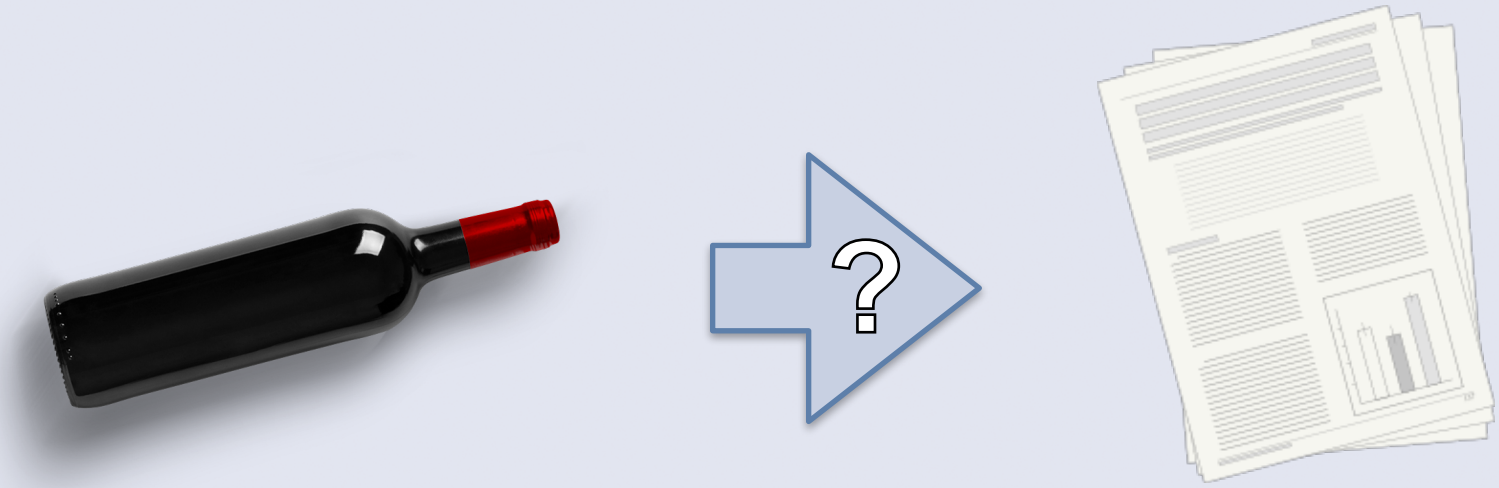
- Advantage: higher flexibility
  - e.g. used in Liedtke’s GPT implementation [Liedtke ’95]

```
 0          lui     k0, KERNEL_BASE          # kernel vars ptr
 1          sd      t0, K_TLB_T0_SAVE(k0)    # save t0
 2          lui     k1, TLB2_BASE            # TLB cache base adr
 3          dmfc0   k0, C0_ENTRYHI           # has VPM/2
 4          dsll    t0, k0, 38
 5          dsrl    t0, t0, 47               # STLB index
 6          daddu   k1, t0, k1
 7          ld      t0, (k1)                 # EntryHi
 8          bne     t0, k0, 2f               # check tag
 9          lwu     t0, 8(k1)                # EntryLo1
10          lwu     k0, 12(k1)               # EntryLo0
11          dmtc0   t0, C0_ENTRYLO0
12          dmtc0   k0, C0_ENTRYLO1
13          lui     k0, KERNEL_BASE
14          tlbwr                            # load TLB
15          ld      t0, K_TLB_T0_SAVE(k0)    # restore t0
16          eret
17 2:       j       tlb2_miss                # TLB cache miss
18          lui     t0, KERNEL_BASE
```

Listing 4.1: TLB refill handler `xtlb_refill`.

L4/MIPS TLB miss fast path handler [Heiser ’01]

Can we improve **fast path** power/energy?

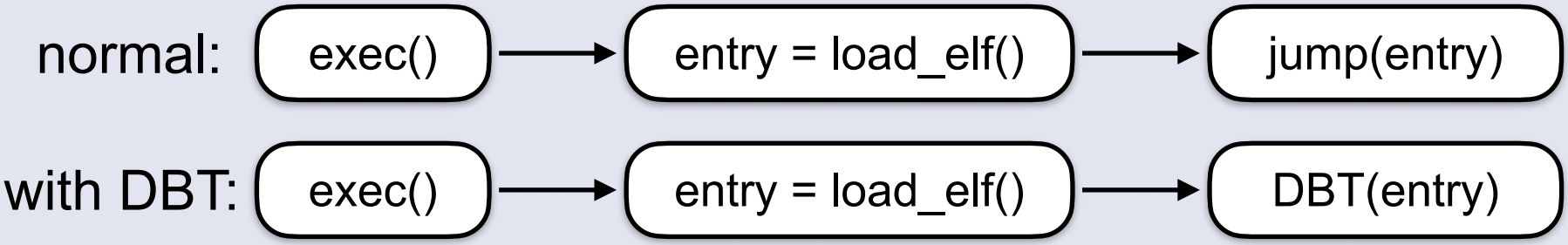


- **Is a software solution for the fast path feasible?**
  - Idea: replicate MMU functionality in software

- What does a MMU do?
  - *translate* virtual to physical addresses for code and data accesses
  - *control* access to pages (RWX)
  - implicitly: *deny/handle* access to non-mapped memory (page fault)

Idea: Translate addresses in software

- **Dynamic Binary Translation** (DBT)
  - A form of just-in-time compilation (JIT) with identical input and output language: machine instructions of a given architecture
- Use DBT to **rewrite binary code during execution**
  - Replace addresses generated by the code by physical ones
  - Dynamic adaptation at runtime
- Translation of user processes:



Example code

Example code for TLB lookup and miss handling [Sieh '17]

```
uint8_t virt_load(uint32_t vaddr) {
  uint32_t paddr;

again:
  found = tlb_lookup(vaddr & ~0xfff, &paddr);
  if (! found) {
    tlb_fill(vaddr & ~0xfff);
    goto again;
  }

  return phys_load(paddr | (vaddr & 0xfff));
}
```

```
struct {
  uint32_t vaddr, paddr;
} tlb[SIZE];

int tlb_lookup(uint32_t vaddr, uint32_t *paddrp) {
  unsigned int hash = (vaddr >> 12) % SIZE;

  if (vaddr == tlb[hash].vaddr) {
    *paddrp = tlb[hash].paddr;
    return 1;
  } else {
    return 0;
  }
}
```

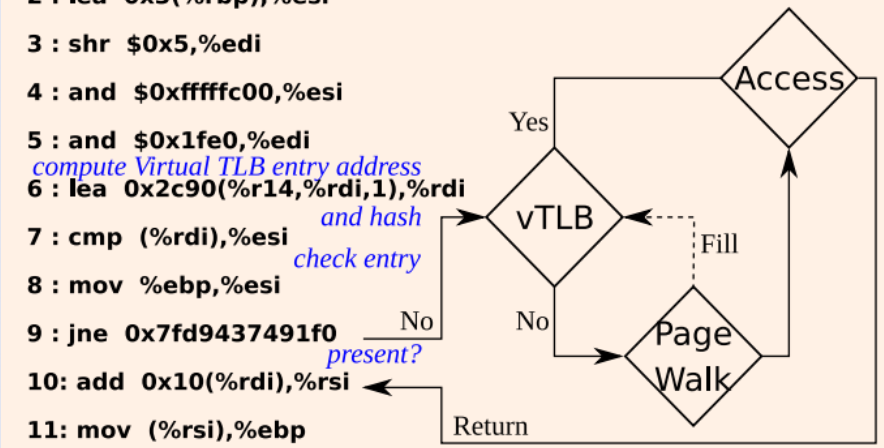
Translation of basic block from MIPS to x86 code in qemu-softmmu (with MMU emulation):

Target source code

ldr sp, [pc, #4] ; @ pc = 0x1000c

Generated host code

```
0 : mov  $0x1000c,%ebp
                        get target address
1 : mov  %ebp,%edi
2 : lea  0x3(%rbp),%esi
3 : shr  $0x5,%edi
4 : and  $0xfffffc00,%esi
5 : and  $0x1fe0,%edi
compute Virtual TLB entry address
6 : lea  0x2c90(%r14,%rdi,1),%rdi
                        and hash
7 : cmp  (%rdi),%esi
                        check entry
8 : mov  %ebp,%esi
9 : jne  0x7fd9437491f0
                        present?
10: add  0x10(%rdi),%rsi
11: mov  (%rsi),%ebp
```



[Cota '19]

Why could this be promising?

- We can run kernel and user mode tasks in physical memory
  - The kernel is untranslated and runs in physical memory (e.g. used in L4 [Nourai ’05])
- MMU emulation requires DBT translation of code/data addresses
  - TLB translation for every instruction fetch
  - and for each load/store instruction
- Most code accesses do not need translation
  - Except control flow – ends basic block, jumps back to the DBT
- Data accesses require translation
  - 20–40% of instructions are load and store on typical RISC
- Are translations cacheable along with a DBTed code block?

Can we improve **fast path** power/energy?

- A simple question – let's see how deep the rabbit hole is…

Rabbit holes are deep and complex…

DBT/JIT approaches exist in a number of related areas…

- Runtime optimization
  - Synthesis (of course!), DynamoRIO
- Running high-level language VMs
  - Inferno (Dis), Java OSes, Singularity
- System emulation
  - qemu, FAUmachine, …
- Hardware co-designed solutions
  - Translation in cache only
- Compiler/runtime cooperation [Suchy '20]

- …but none of the related papers mention runtime translation of binaries!

Did we find an undiscovered opportunity?

- **So… did we find the (a?) pot of gold?**
  - Time for some reflection
    – the wheel has been reinvented many times in computer science…
- **Or did we hit the problem that negative results are never published**
  - **…and dozens of PhD students have tried this approach in vain?**

What could be the challenges realizing this idea?

- Can we actually save energy?
  - Parallelism in hardware replaced by serial execution in software
  - Can optimizations for regular JIT engines (e.g. trace scheduling) help here?
- Unix semantics based on virtual memory
  - How does this approach work for shared libraries, shared memory, mmap?
- Can we achieve the same level of security as with a hardware MMU?
  - JIT compiler attacks, Thompson's "Reflections on Trusting Trust"
- Memory overhead of DBT on small machines
- What about x86?

Interim conclusion and outlook

DBT (and JIT translation in general) is an interesting approach

- Implementation and evaluation for VM use case ongoing
  - Might turn out to require too much overhead
- Other use cases – optimization, more?

Opportunities

- More flexible memory management possible
  - e.g., Liedtke's GPTs
  - Software implementation of capabilities?
- "Design in software first approach"
  - Develop OS functionality first
  - Derive possible required optimizations in hardware later

References

[Nourai ’05] Abi Nourai, *A Physically-Addressed L4 Kernel*, B.Sc. Thesis, UNSW 2005

[Heiser ’99] Gernot Heiser, *Inside L4/MIPS*, UNSW 2001

[Liedtke ’95] Jochen Liedtke, *Guarded Page Tables*, PhD thesis, GMD 1996

[Sodani] Avinash Sodani, *Race to Exascale: Challenges and Opportunities*, MICRO’11 Keynote

[Basu ’12] Arkaprava Basu, Mark D. Hill, Michael M. Swift, *Reducing memory reference energy with opportunistic virtual caching*, Proc. ISCA ’12

[Puttaswamy '06] Kiran Puttaswamy, Gabriel H. Loh, *Thermal analysis of a 3D die-stacked high-performance microprocessor*, Proc. VLSI ’06

[Sieh ’04] Volkmar Sieh and Martin Weitz, *Advanced virtualization techniques for FAUmachine,* 11th International Linux System Technology Conference 2004

[Nagle '93] David Nagle et al., *Design Tradeoffs for Software-Managed TLBs*, ACM SIGARCH Computer Architecture News May 1993

[Rajagopalan] Mohan Rajagopalan et al., *Binary Rewriting of an Operating System Kernel*, https://www2.cs.arizona.edu/~debray/Publications/wbia-plto.pdf

[Chapman '07] Matthew Chapman, *MagiXen: Combining binary translation and virtualization*, Technical Report, HP Labs 2007

[Cota '19] Emilio Cota, Scalable Emulation of Heterogeneous Systems, PhD Defense, Columbia University 2019

[Suchy '20] Brian Suchy et al., *CARAT: A Case for Virtual Memory through Compiler- and Runtime-Based Address Translation*, Proc. PLDI ’20

[Sieh ’17] Volkmar Sieh, *Virtual Machines: Emulation*, https://www4.cs.fau.de/Lehre/WS17/V_VM/Vorlesung/emulation.pdf

Wiss. MitarbeiterIn (m/w/d) TV-L 13 mit Gelegenheit zur Promotion

michael.engel@uni-bamberg.de