TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute for System Architecture, Operating Systems Group

PAGE TABLE STRUCTURES FOR FINE-GRAIN VIRTUAL MEMORY

JOCHEN LIEDTKE

CARSTEN WEINHOLD

MOTIVATION

- Context: object oriented systems
- Objects are pieces of memory
- Access control, based on:
  - ~~Segments~~ *Not really there, huge tables*
  - ~~Pages~~ *Hierarchies, sharing, coarse-grained*
- Goal: purely page-based architecture that efficiently supports fine granularity

REQUIREMENTS

- Fine-grained address spaces need:
  - Small pages (e.g., 16 byte)
  - Various page sizes (powers of 2)
  - Different page sizes can be mixed freely
  - Efficient operations on hierarchy of small and large regions

MULTI-LEVEL PTs

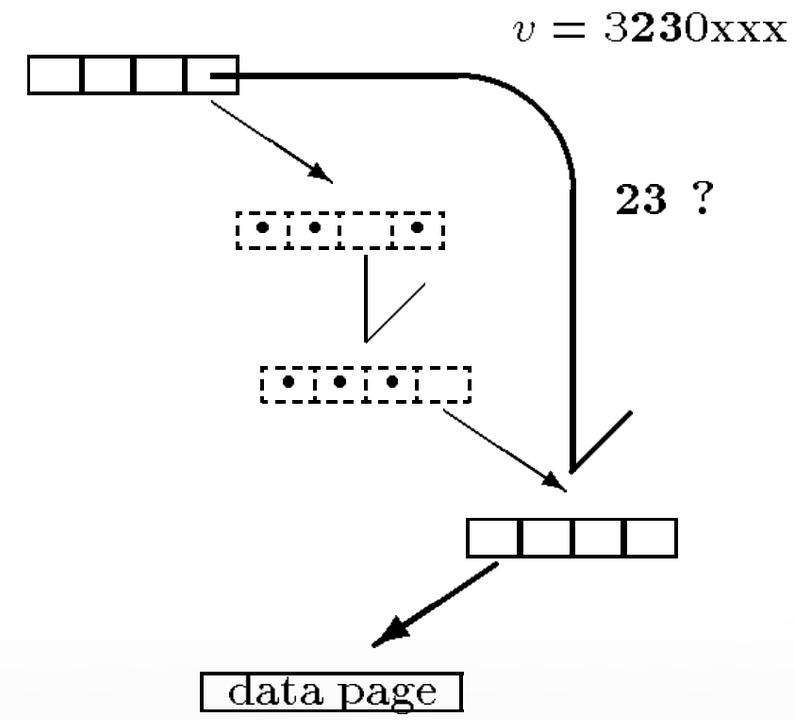
- Conventional page tables:
  - Multi-level
  - Few fixed page sizes (4K, 2M, 4M, ...)
  - Large tables per level (512, 1024 entries)
  - Coarse-grained
  - Wasted storage for sparse address spaces
  - „Small“ address spaces are expensive

INVERTED PTs

- Inverted page tables:
  - One entry per page
  - Sparse address spaces are efficient
- Hashed page tables:
  - IPT entries with physical address ⇒ aliasing
- Problems:
  - No hierarchy, changing attributes expensive
  - Fixed page sizes, high load on TLB

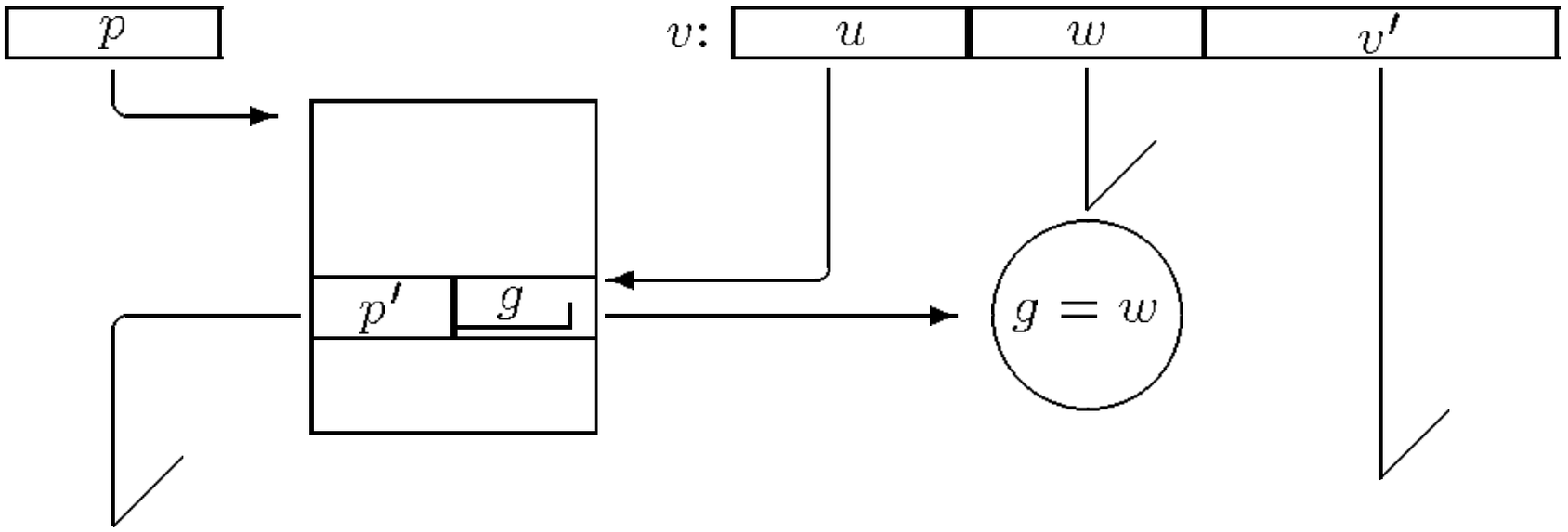
GUARDED PTs

- Sparse address space:
  - Only one valid entry in 2nd and 3rd PT level
  - PTs effectively not needed, wasting memory
- Idea:
  - Remove unneeded PTs
  - Add extra info to PT entries to indicate shortcut



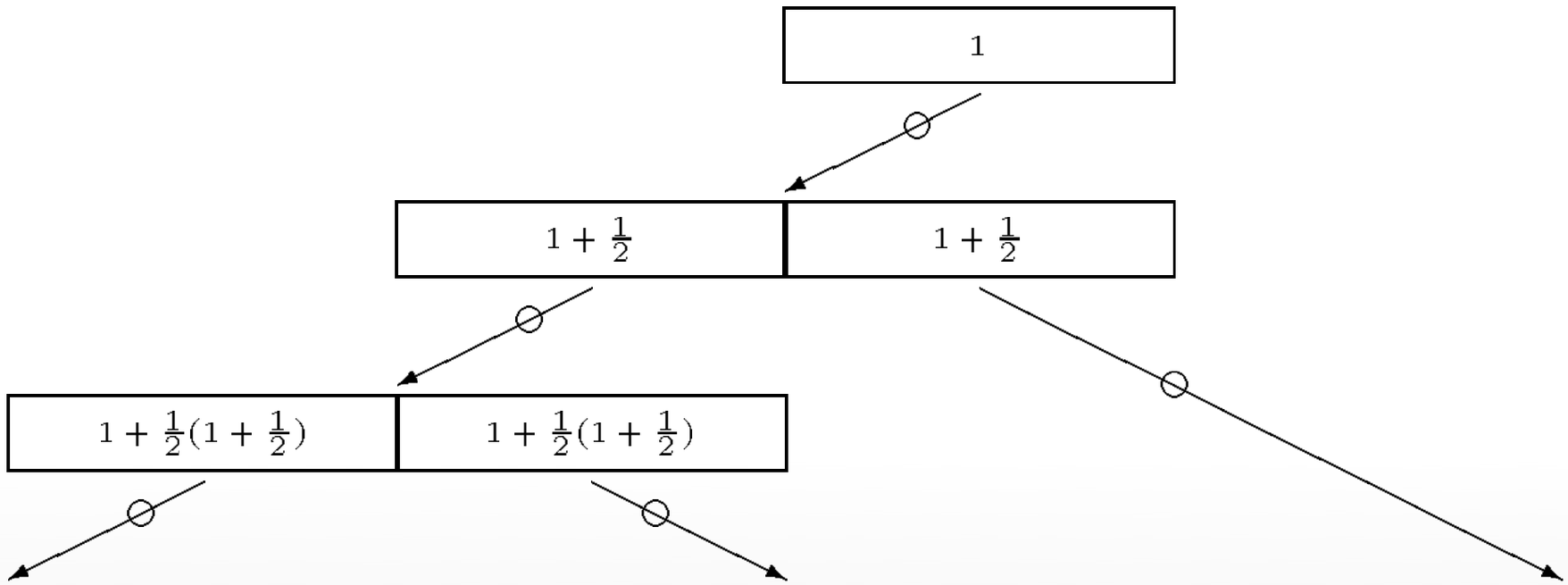
- PT entries augmentated:
  - Bit string *g:* „guard“
  - Size of next PT / data page
    - Length of *g*, size of next level can vary
- Translation works like for conventional PTs
- Conventional PTs are special case: $g=\varnothing$

TRANSLATION



SMALL TREES

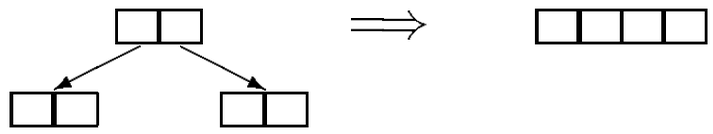
- Less than 2 GPT entries needed per page
- Regardless of address-space / page sizes



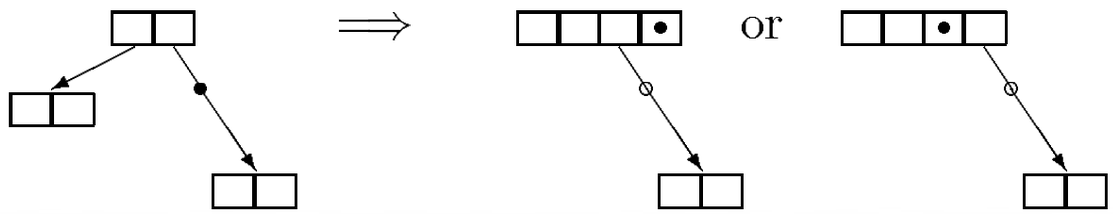
$$1 + ½(1 + ½(1 + \ldots)) = 1 + ½ + ¼ + \ldots < 2$$

TREE DEPTH

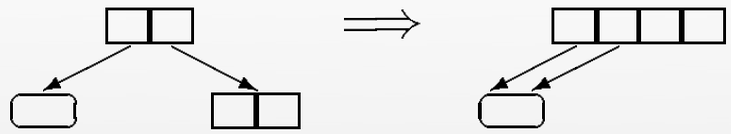
- Translation steps determine performance
- *n* address bits can be translated in ⌈ *n*/2 ⌉ steps



- All nodes that decode only one bit can be transformed

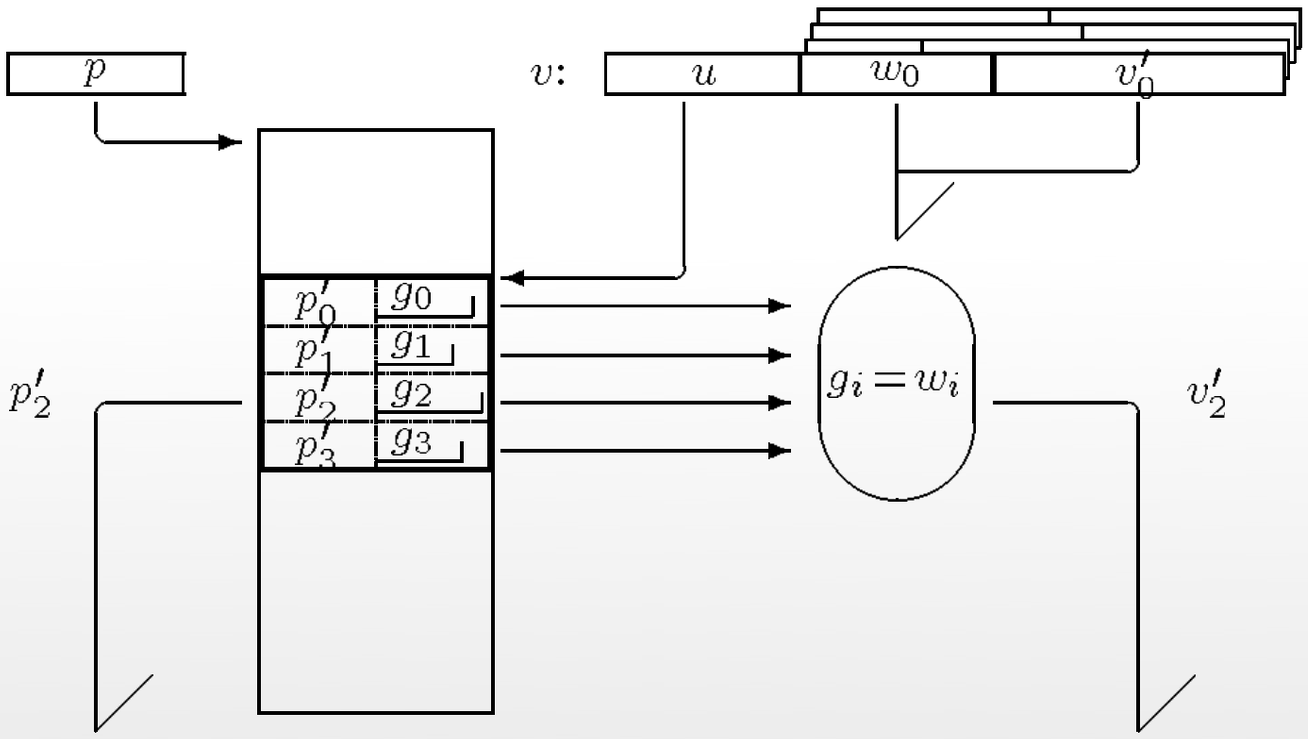


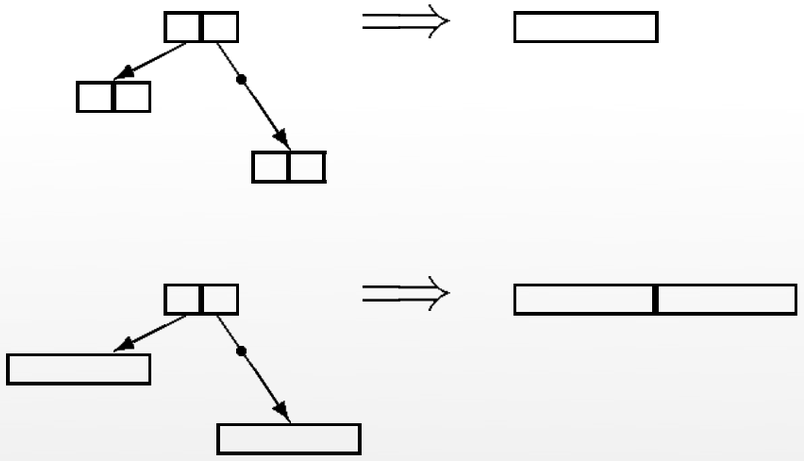
- Transformed tree remains small



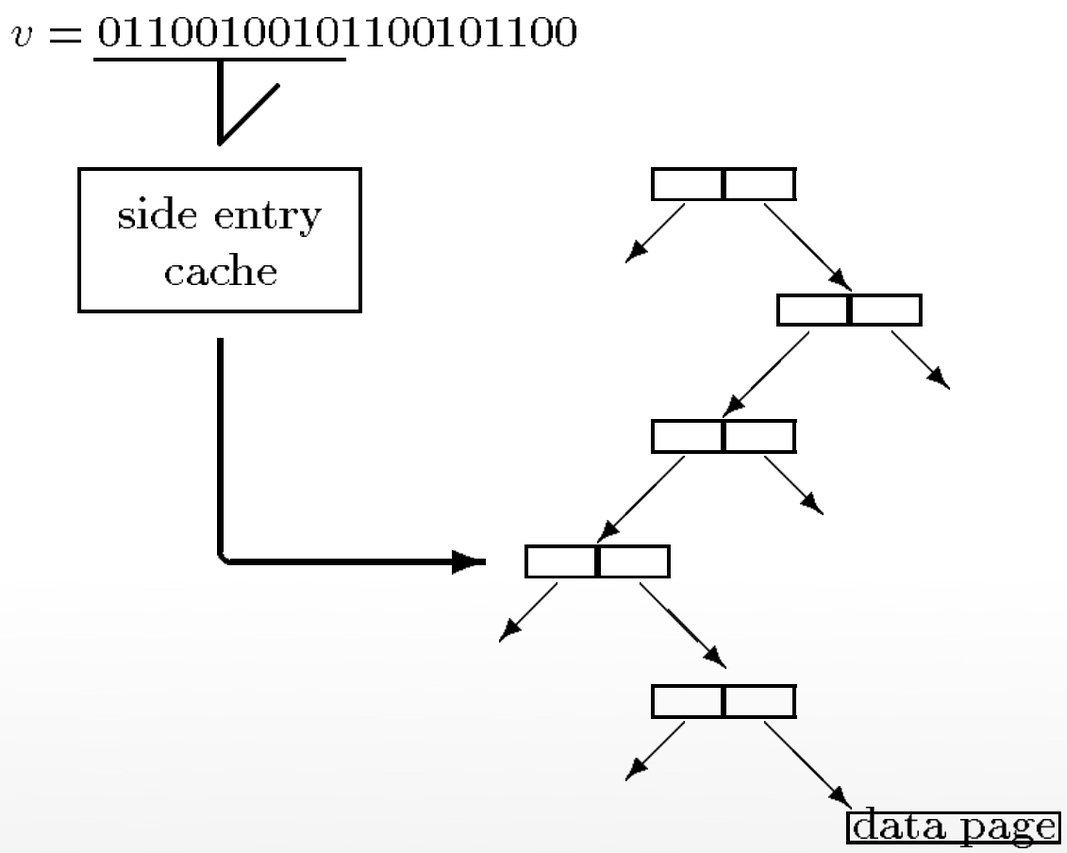
TREE DEPTH

- Translation in *n*/2 steps still takes long
- *k*-associative translation:
  - Perform *k* translations in parallel
  - Allows for larger nodes





Side Entry-Point Caching



USER MAPPINGS

- Paper proposes user-level mapping
- PT entries augmented with type *t*:
  - *t* = alias
    - Address in PT entry is virtual
    - Aliasing of virtual memory
  - *t* = call-on-reference
    - Address in PT entry is function pointer
    - Executes user-defined code on access

ALIASING

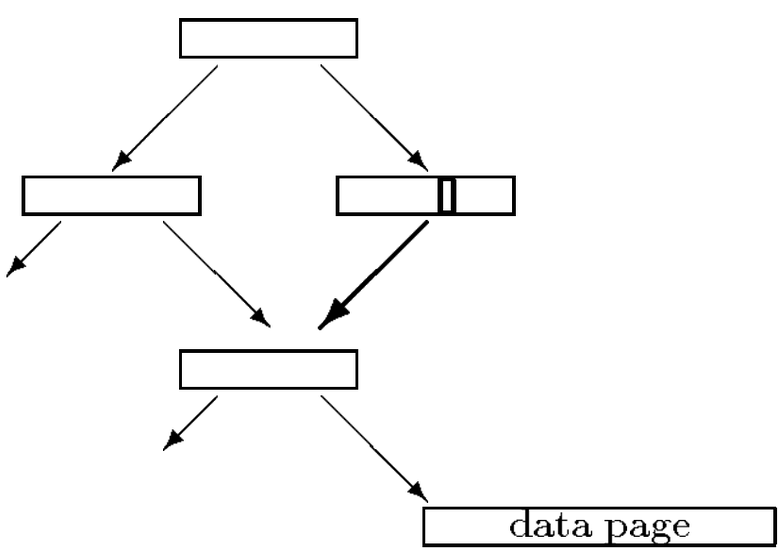


Figure 11: *Physical Aliasing.*

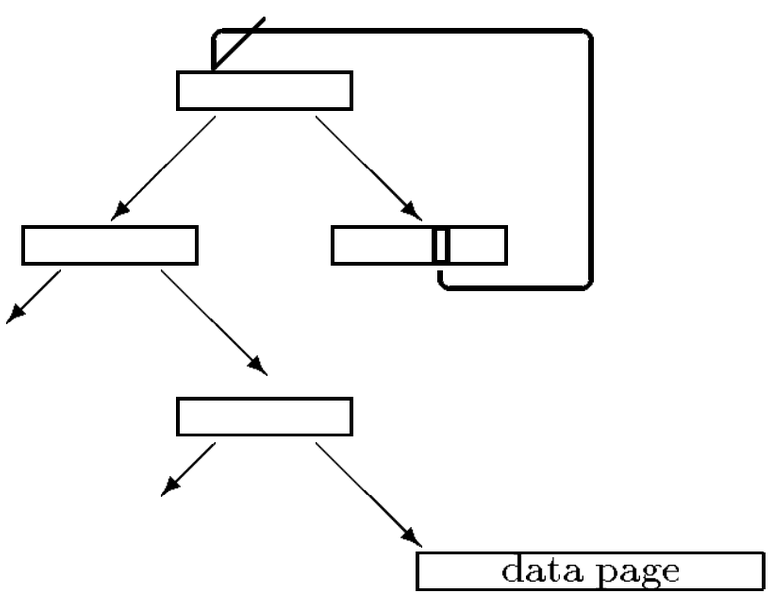


Figure 10: *Virtual Aliasing.*

MAP INSTRUCTION

- Unprivileged **map** instruction:
  - Allows app to modify its own PT entries
  - No kernel entry / exit needed
  - Only works if PT entry type t matches
  - Virtual addresses only, no isolation breach

FINAL SLIDE

- GPTs allow for a wide range of page sizes
- Very efficient for sparse address spaces

- Discussion points:
  - What about PT allocation?
  - Different page sizes vs. complexity?
  - Usecases?
  - Practical performance? MIPS prototype?