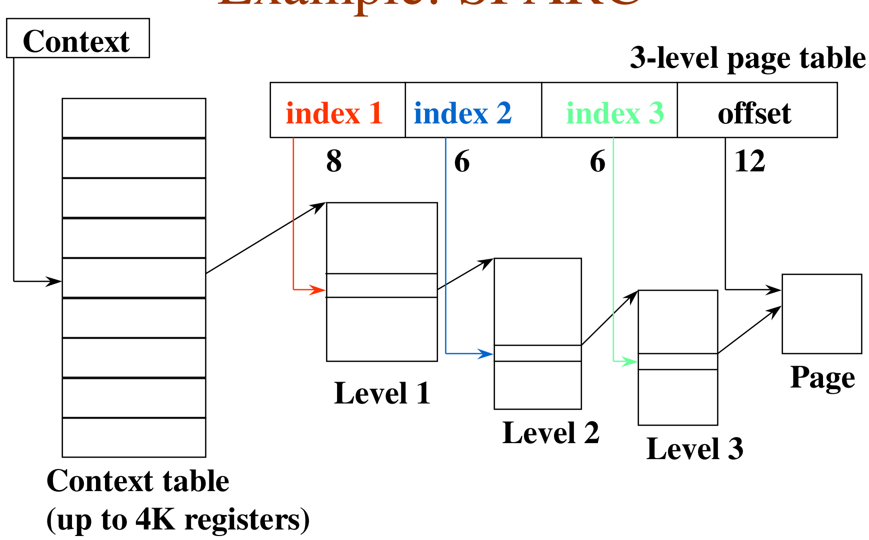
Solution1: Multi-Level Page Tables

- Use two or three levels page tables
- All entries in the topmost level must be there
- Entries in lower levels are there only if needed
- Store page tables in memory
- Have one CPU register contain address of top-most level table

Example: SPARC



SPARC: Cont'd

- Only level 1 table need be there entirely
  - 256 entries * 4 bytes/entry = 1K /process
- Context switching is not affected
  - Just save and restore the context register/process
- Second and third level tables are there only if necessary

Translation Lookaside Buffer

- A small associative memory in processor
- Contains recent mapping results
- Typically 8 to 32 entries
- If access is localized, works very well
- Must be flushed on context switching
- If TLB misses, then must resolve through page tables in main memory (slow)

Other Varieties

- 2-level page table in VAX systems
- 4-level page table in the 68030/68040
- Organize the cache memory by virtual addresses (instead of physical addresses)
  - Remove the TLB from critical path
  - Combine cache misses with address translations
  - e.g. MIPS 3000/4000

Solution 2: Inverted Page Tables

Rationale:

- Conventional per-process page tables depend on virtual memory size
- Virtual address space is getting larger (e.g. 64 bits)
- Size of physical memory projected is less than virtual memory in foreseeable future

Inverted Page Table

Main Idea:

- Global page table indexed by frame number
- Each entry contains virtual address & *pid*
- Use TLB to reduce the need to access PT
- On a TLB miss:
  - Search page table for the <virtual address, pid>
  - Physical address is obtained from the index of the entry (frame number)

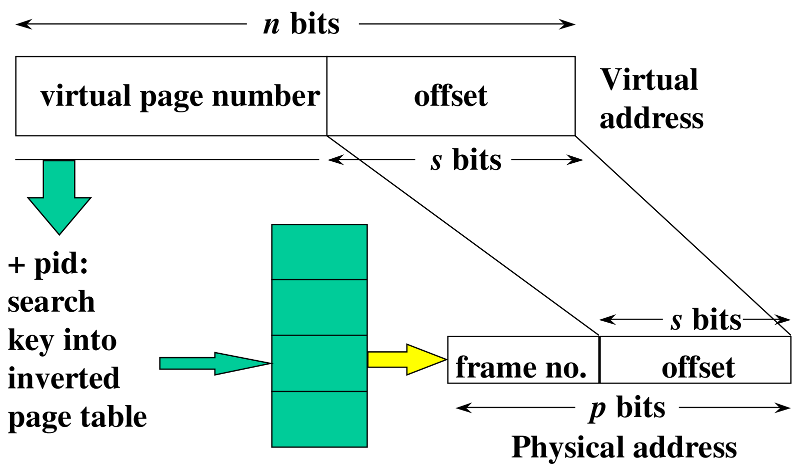
Inverted Page Table Structure

- Indexed by frame number
- Entry contains virtual address and pid using the frame number (if any)
- Contains protection & reference

pid	**Virtual addr**	**v**	**w**	**r**	**x**	**f**	**m**
pid	**Virtual addr**	**v**	**w**	**r**	**x**	**f**	**m**
pid	**Virtual addr**	**v**	**w**	**r**	**x**	**f**	**m**

v: valid bit
w: write bit
r: read bit
x: execute bit (rare)
f: reference bit
m: modified bit

Mapping Virtual to Real Addresses

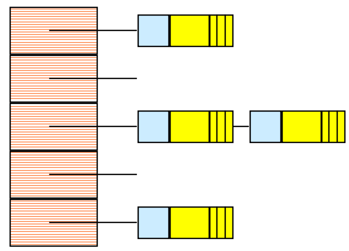


Properties & Problems

- Table size is independent of virtual address size, only function of physical memory size
- TLB misses are expensive
  - We don't know where to look
  - May require searching entire table (very bad)
- Virtual memory more expensive
- Sharing becomes very difficult

A Solution

- Organize Inverted Page Table as a hash table
  - Search key <pid, vaddr>
  - Search in hardware or software
- Examples: IBM 38, RS/6000, HP PA-RISC systems



Sharing & Inverted Page Tables

- Conceivably possible with a general hashing function
- Requires an additional field in page table entry
- Size no longer limited, so no system implements it

Frame no.	**pid**	**Virtual addr**	**v**	**w**	**r**	**x**	**f**	**m**
Frame no.	**pid**	**Virtual addr**	**v**	**w**	**r**	**x**	**f**	**m**
Frame no.	**pid**	**Virtual addr**	**v**	**w**	**r**	**x**	**f**	**m**