Microkernel Construction III
Virtual Memory

Cristan Szmajda
cls@cse.unsw.edu.au

23 October 2003

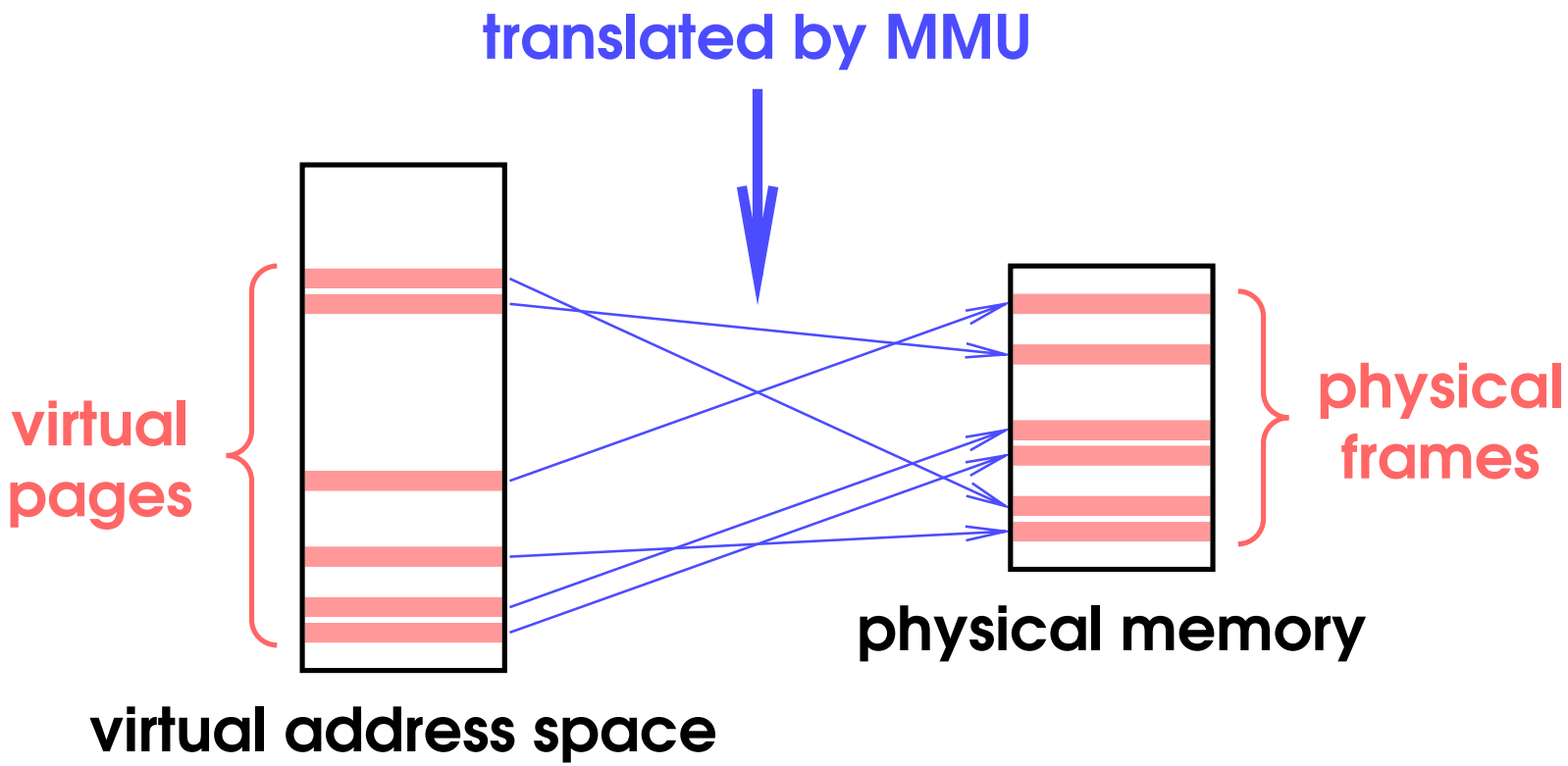
THE UNIVERSITY OF
NEW SOUTH WALES

Outline of lecture

- brief **introduction** to virtual memory and page tables
- **page table structures** and the **page table problem**
- the **guarded page table (GPT)**
  - → theory, implementation, and performance
- the **variable radix page table (VRPT)**
  - → theory, implementation, and performance
- advanced hardware VM features
  - → **superpages**
  - → support for **shared memory**
- virtual memory implementation in the **L4 microkernel**

Virtual memory (VM)

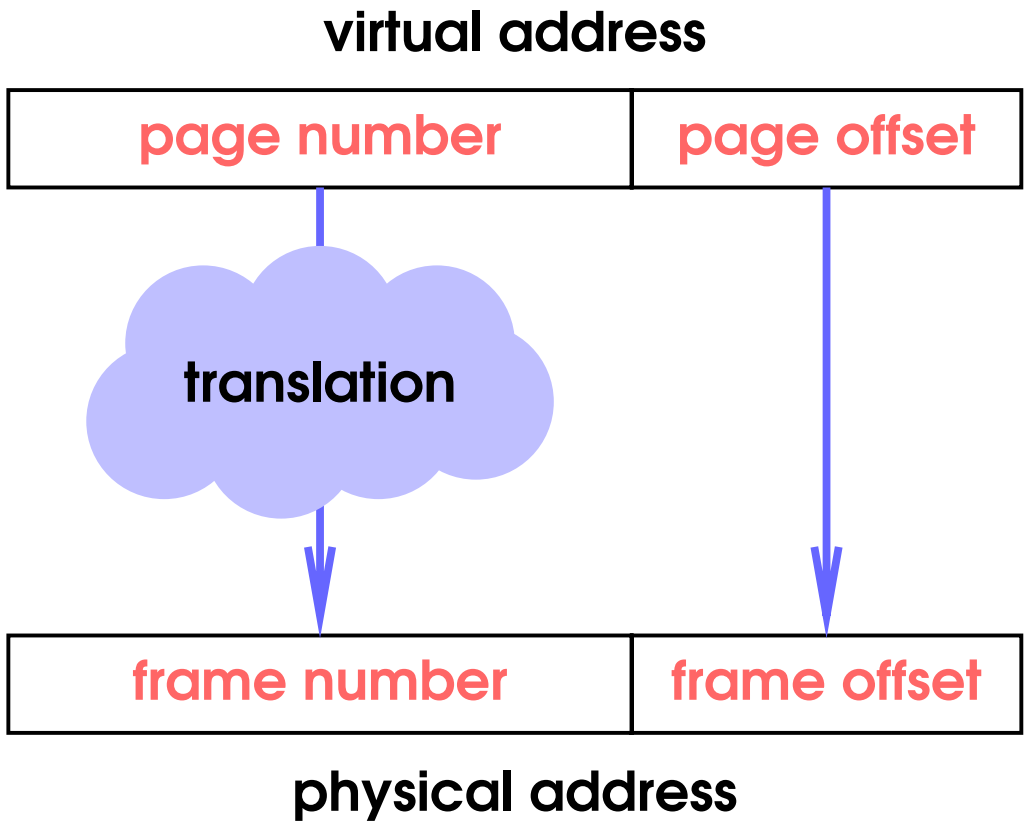
VM allows the OS to manage memory and swap to disk.



A memory management unit (MMU) translates virtual to physical.

Paging

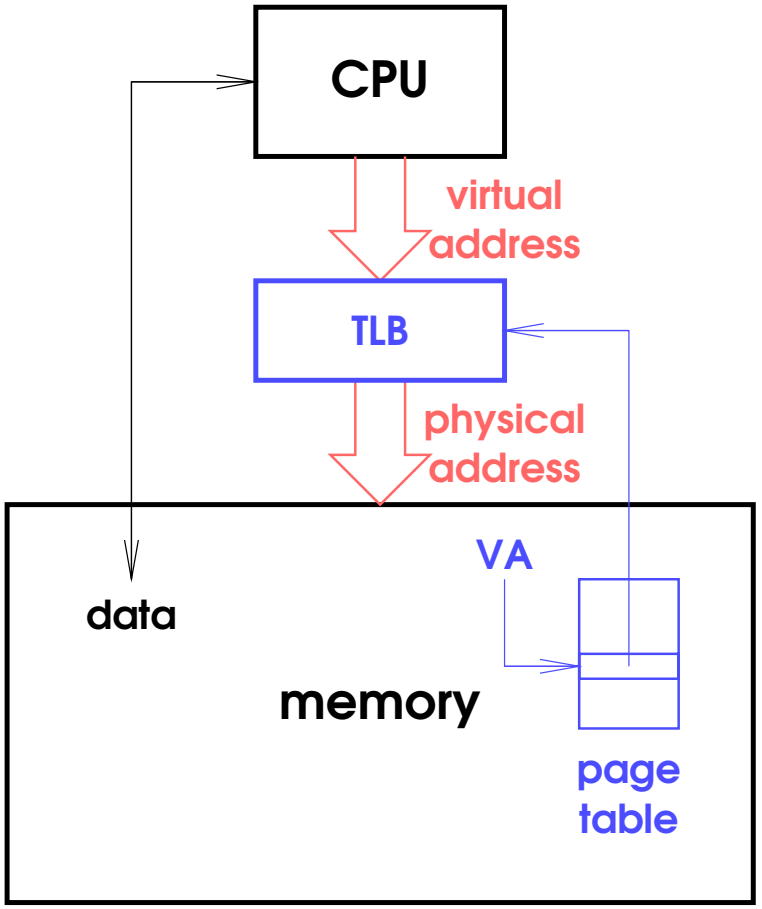
Translation: virtual page number → physical frame number



The page table (PT)

- table of translations
- stored in main memory
- cached in translation lookaside buffer (TLB)
- consulted on every TLB miss

  (hardware OR software)
- uses memory bandwidth
- comes in a variety of different formats



Page table structures

VM performance is directly limited by page table performance.

Classical page table structures were designed for

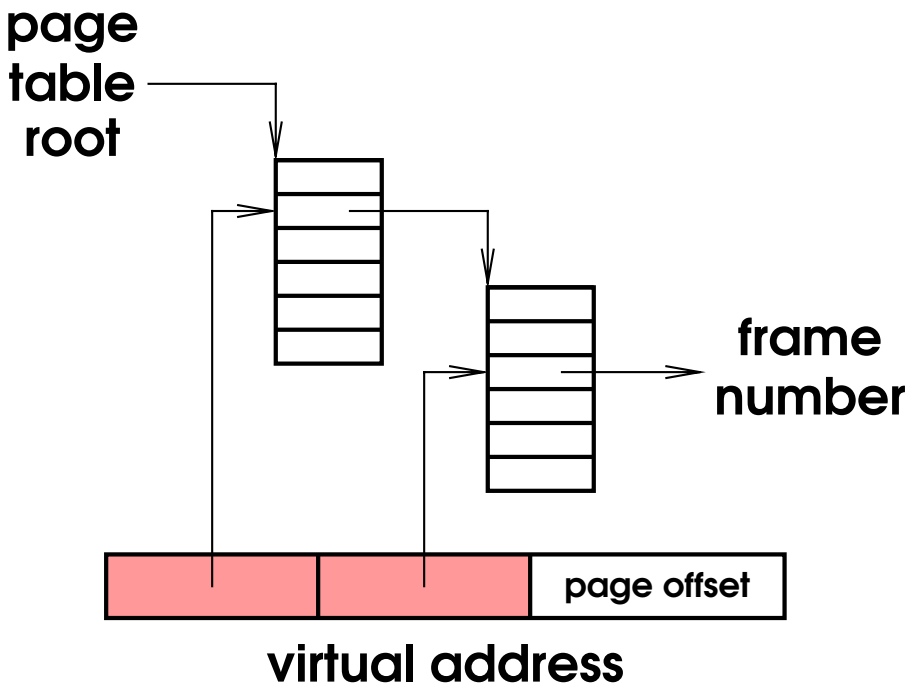
- 32-bit address spaces, and
- the Unix two-segment model.

How well do they perform in:

- large (64-bit) address spaces?
- sparse address-space distributions?
- micro-kernel system structures?

Multi-level page table (MLPT)

Traditional MLPT is used by many 32-bit processors.

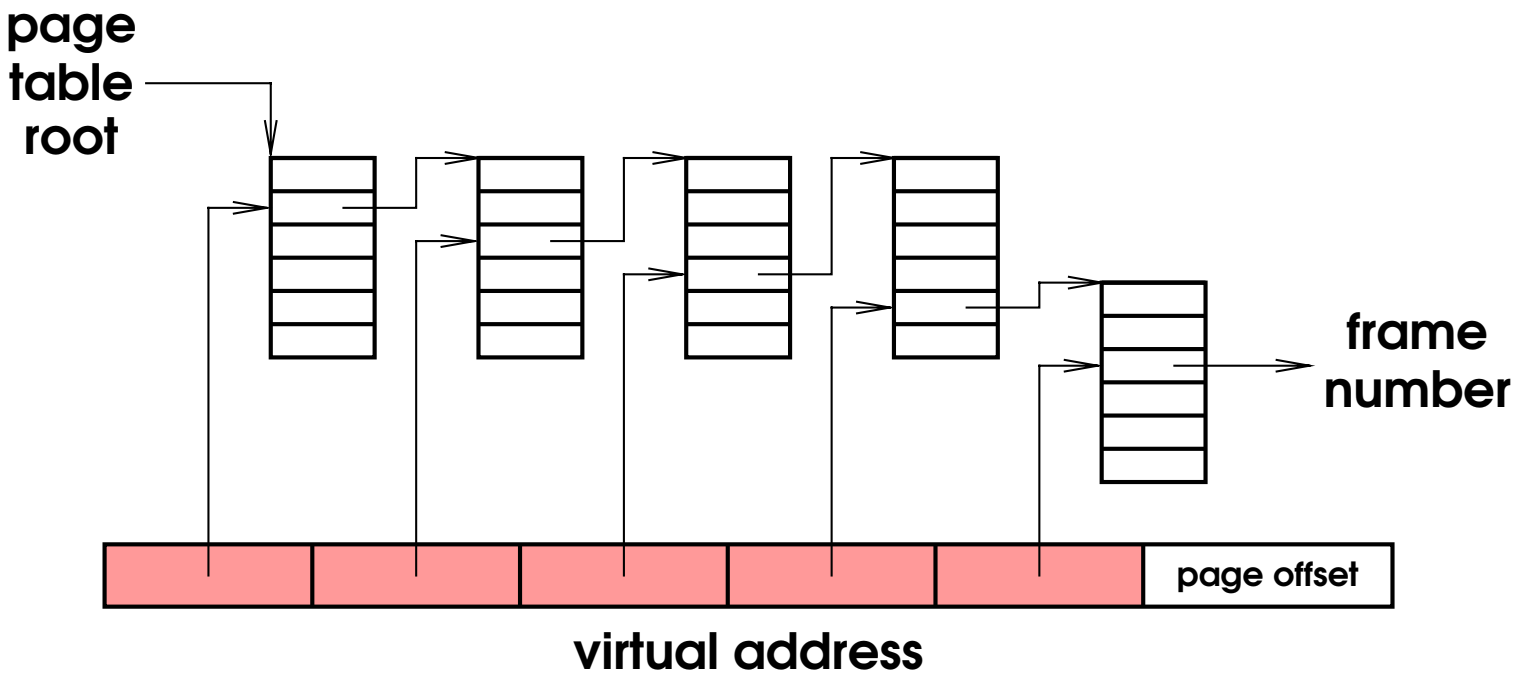


- Saves space when most of address space is unused.
- more levels $\rightarrow$ saves space, costs time

Multi-level page table (MLPT)

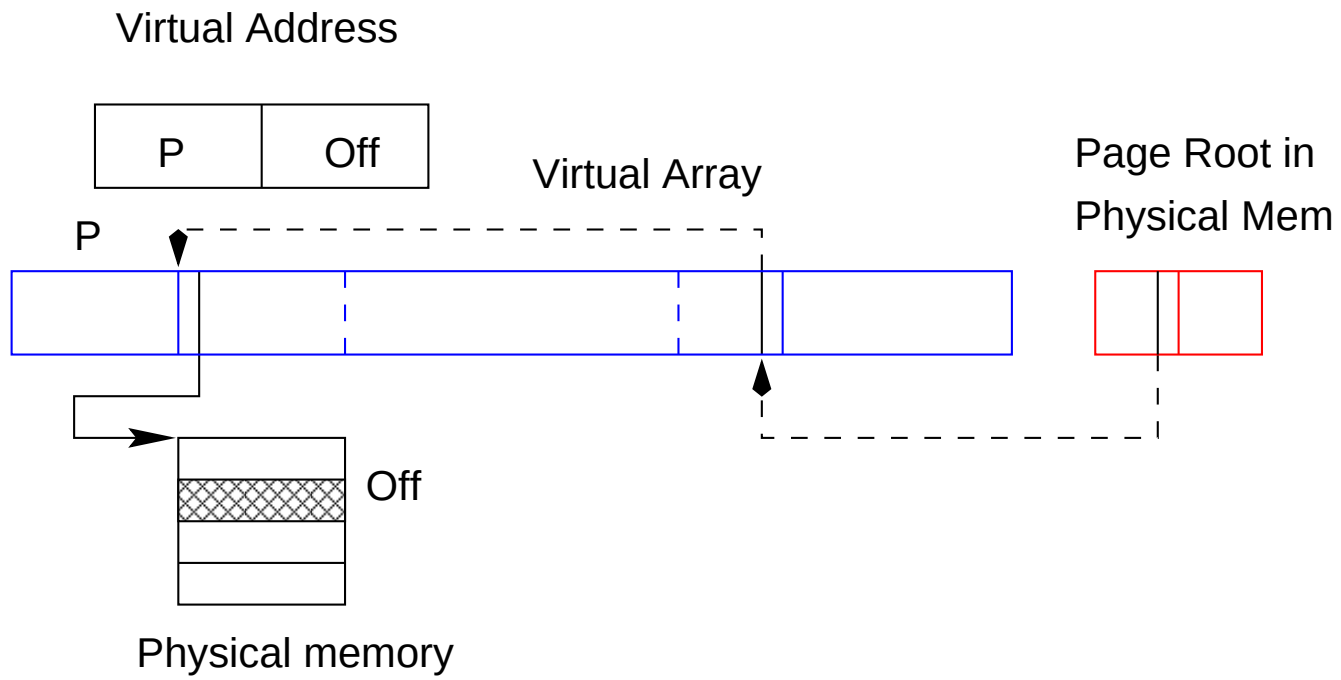
Disadvantages:

- Extension to 64-bit requires 4 to 5 levels.



- Sparse address spaces waste a lot of space.

Virtual linear page table (VLPT)

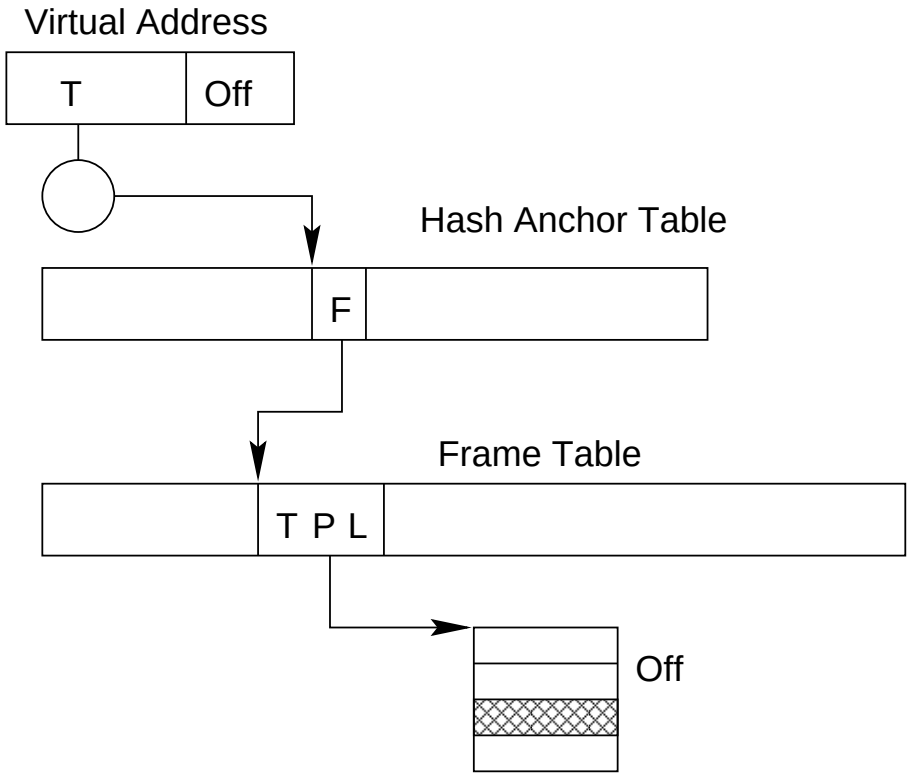


Equivalent to MPT, but:

- better best-case lookup time
- steals TLB entries from application
- requires nested exception handling

Inverted page table

- indexed by physical (not virtual address)
- hash anchor table (HAT) is for lookup by virtual address



Inverted page table

Advantages:

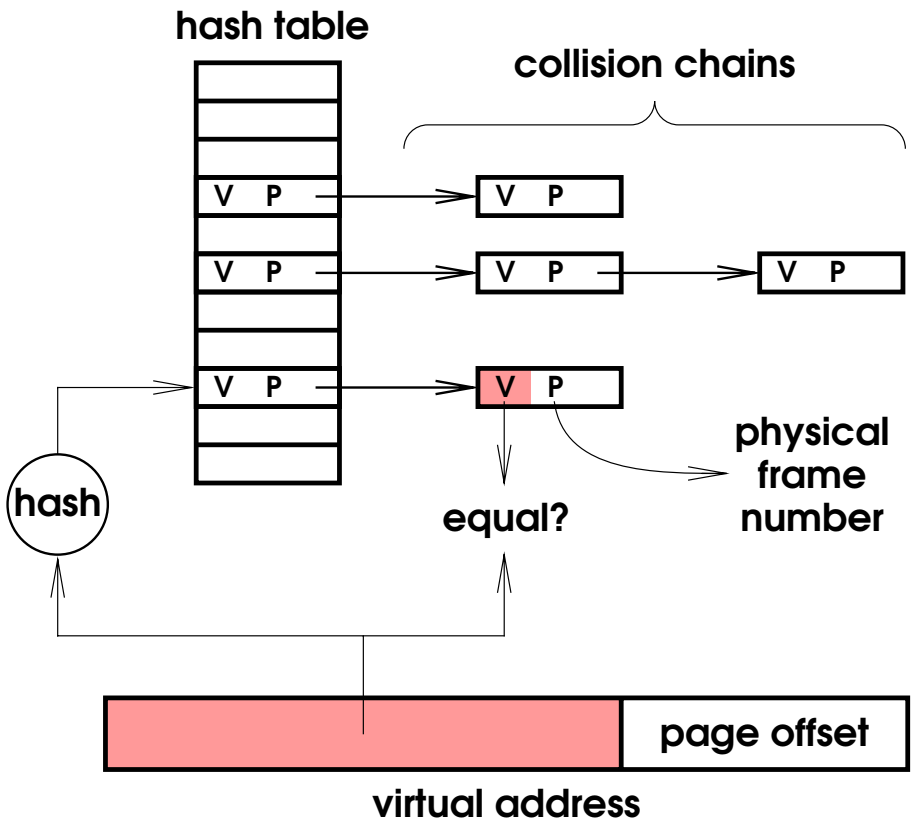
- scales with physical, not virtual, memory size
- no problem with virtual sparsity
- one IPT per system, not per address space
- PTEs bigger as need to store tag
- system needs a frame table anyway

Disadvantages:

- newer machines have sparse physical address space
- difficult to support super-pages
- sharing is hard

Hashed page table (HPT)

HPT merges the IPT and HAT into a single table.



Each HPT entry contains both virtual and physical page numbers.

Hashed page table (HPT)

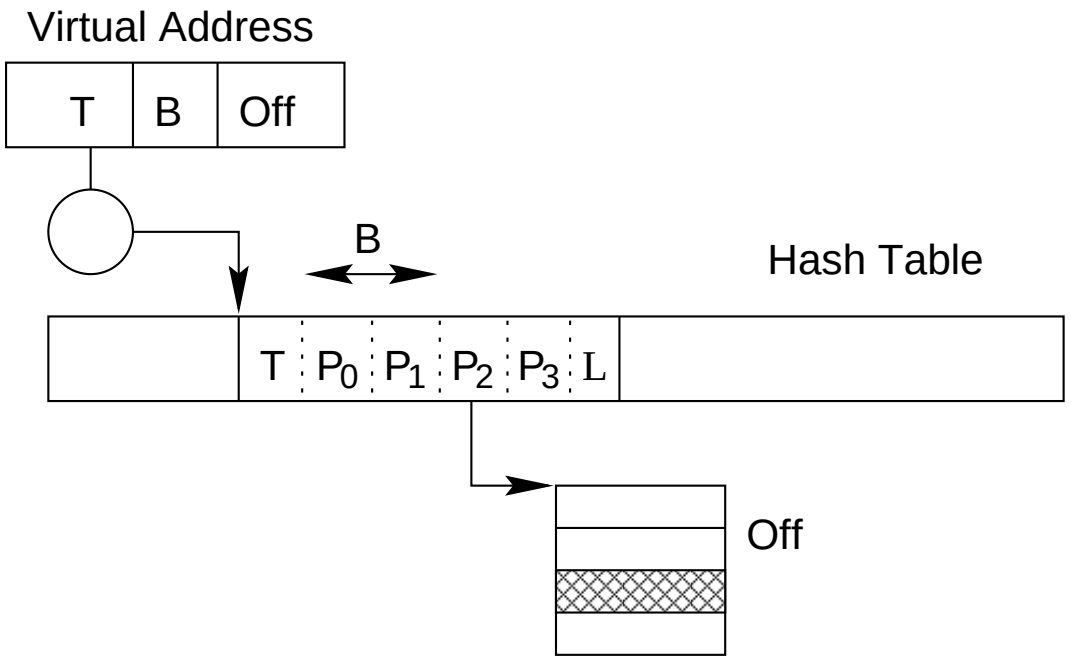
Advantages:

- best-case lookup: one memory reference
- sparsity no problem
- hash function independent of address size

Disadvantages:

- collisions
- sharing
- creation & deletion costs
- traversal
- assumes a single page size

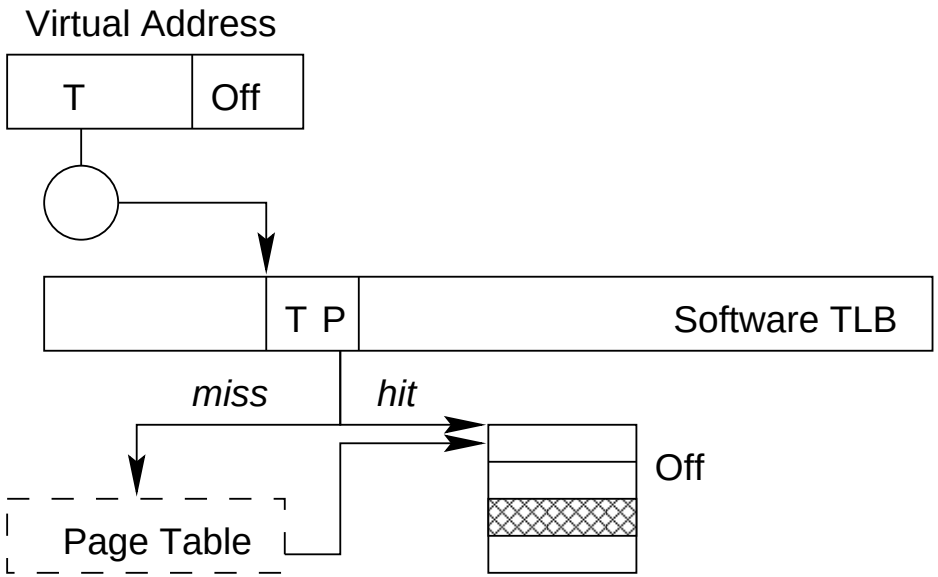
Clustered page table (CPT)



- an HPT with multiple pages per PTE
- can load multiple pages into the TLB per miss
- improves performance in presence of spatial locality
- clustering also used in MIPS R4000 hardware TLB entry

Software TLB

- a direct-mapped cache of TLB entries in main memory
- fast lookup; can achieve >95% hit rate
- also called *TLB cache* or *level 2 TLB*



- also simple enough for hardware implementation
- difficult to support super-pages

The page table problem

What's a computer architect to do?

- Provide PT walker in hardware.
  - → complexity (e.g., microcode, cache coherency)
  - → PT format difficult to change
- Provide a hardware HPT walker, but trap to software on every collision.
  - → could be used as hardware-assisted HPT, or
  - → could be used to cache some other PT format
- Trap to software on every TLB miss.
  - → PT format completely at discretion of software
  - → exceptions can be expensive

TLB performance

TLB overhead is getting worse in modern architectures.

- Clark and Emer (1985) report **5.4 – 8.7%** (VAX-11/780).

TLB performance

TLB overhead is getting worse in modern architectures.

- Clark and Emer (1985) report **5.4 – 8.7%** (VAX-11/780).
- Chen, Borg, and Jouppi (1992) report **2.9% – 47%** (R2000).

TLB performance

TLB overhead is getting worse in modern architectures.

- Clark and Emer (1985) report **5.4 – 8.7%** (VAX-11/780).
- Chen, Borg, and Jouppi (1992) report **2.9% – 47%** (R2000).
- Romer et al. (1995) report **5.5 – 71%** (Alpha 21064).

TLB performance

TLB overhead is getting worse in modern architectures.

- Clark and Emer (1985) report **5.4 – 8.7%** (VAX-11/780).
- Chen, Borg, and Jouppi (1992) report **2.9% – 47%** (R2000).
- Romer et al. (1995) report **5.5 – 71%** (Alpha 21064).
- Navarro et al. (2002) report **0 – 83%** (Alpha 21264).

  Extreme case: *matrix* benchmark: **650%**

TLB performance

TLB overhead is getting worse in modern architectures.

- Clark and Emer (1985) report **5.4 – 8.7%** (VAX-11/780).
- Chen, Borg, and Jouppi (1992) report **2.9% – 47%** (R2000).
- Romer et al. (1995) report **5.5 – 71%** (Alpha 21064).
- Navarro et al. (2002) report **0 – 83%** (Alpha 21264).

  Extreme case: *matrix* benchmark: **650%**

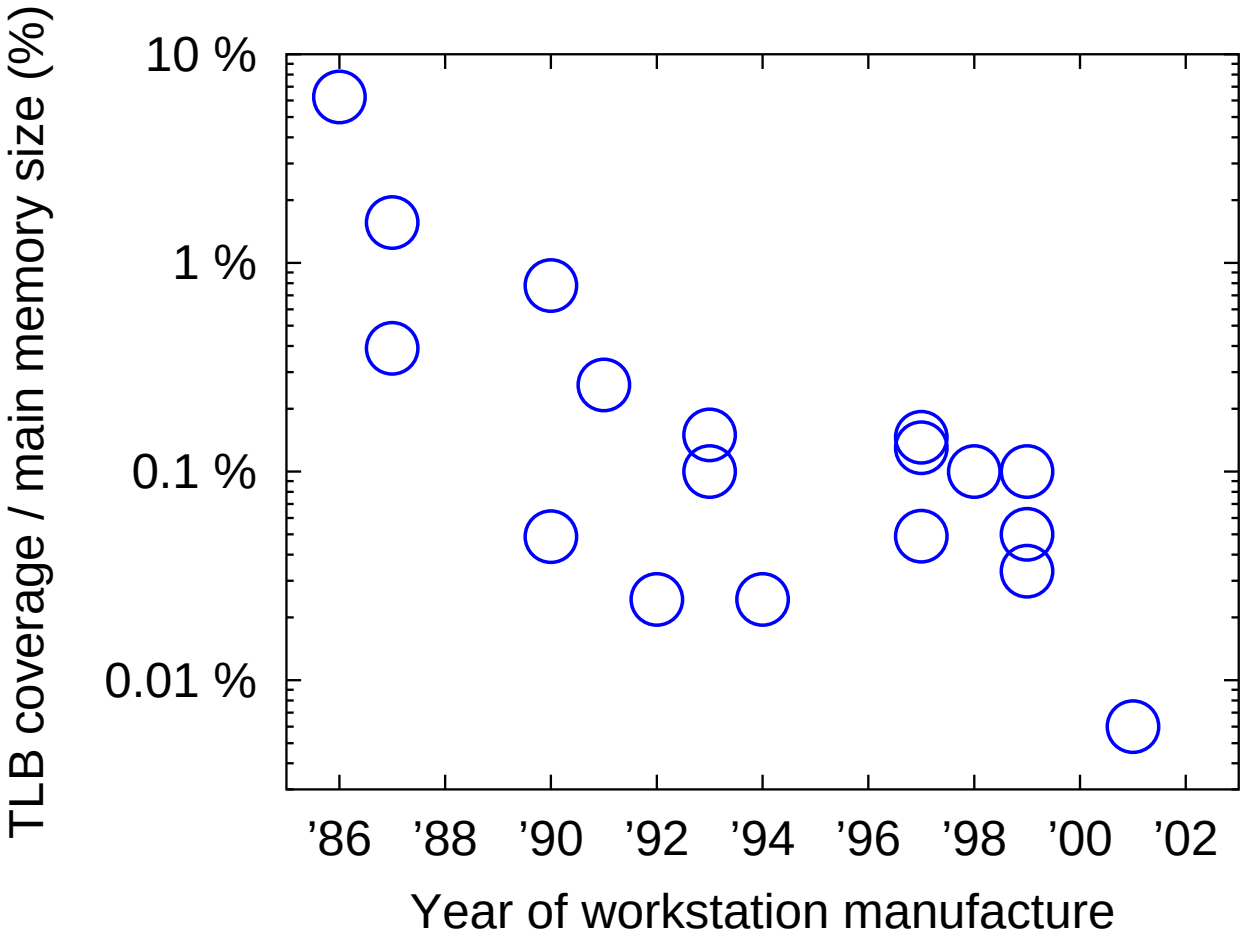
Average and **worst-case** overheads are trending up. **Why?**

Why has TLB overhead increased?

- Software expands to fill available space.
- Widening gap between processor and memory speed.
  - → lots of attention on cache hierarchies
  - → not so much attention on TLBs and page table
  - → TLB miss penalty is high
- 64-bit addresses
  - → widen CAM in TLB (which is virtually-tagged)
  - → complicate the page table
- Wide sharing → improves memory (but not TLB) coverage

TLB performance is improving slower than memory performance.

TLB capacity



Source: Navarro et al. (2002)

TLB miss penalty

Increasing due to more complex and pipelined processors.

- i80386: 9 to 13 cycles
- VAX-11/780: 17 to 23 cycles
- Pentium 4: 31 to 48 cycles (assuming L2 cache hits)
- PowerPC 604: 120 cycles (assuming software TLB hit)

Why not just make bigger and faster TLBs?

- large CAMs are slow and hot
- often flushed (context switch, address space teardown, protection change, etc.)
- MHz, MBytes, and caches sell computers, not TLBs

Why not just increase the page size?

- fragmentation
- I/O latency
- inertia

The solutions provided by computer architects are different…

Smart TLBs in modern architectures

How have computer architects addressed the problem?

- superpages

  → more TLB coverage with same number of entries

- shared tags (e.g., IA-64 protection keys)

  → addresses the effect of sharing on TLB coverage

- hardware assistance for TLB refill

  → faster exceptions

  → partial or full hardware PT walker

TLBs are becoming **smarter,** not **larger** and **faster.**

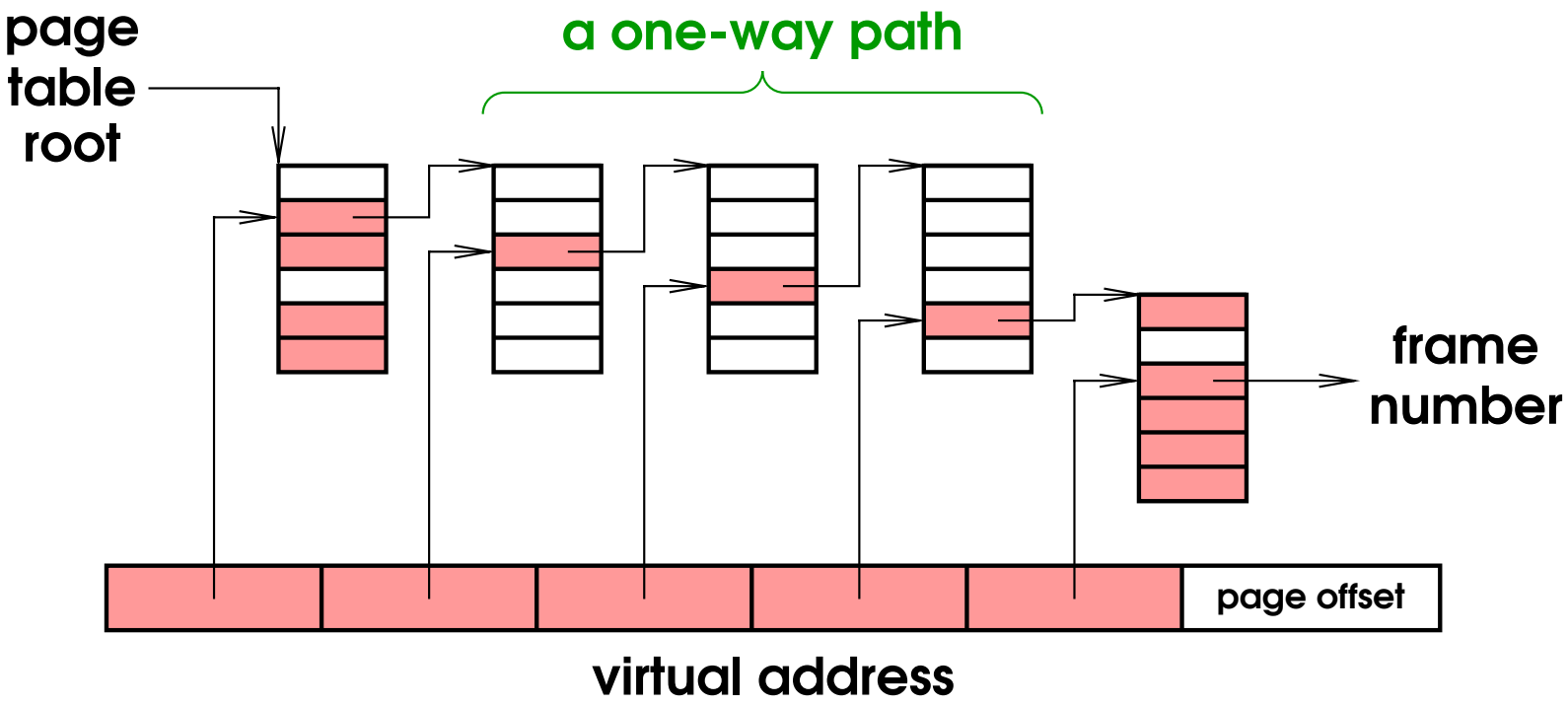
A smart page table

Need a page table that:

- minimizes the number of references to slow memory
- works well with 64-bit addresses
- works well with sparse address spaces
- is fast to create, destroy, and traverse
- supports mixed page sizes
- supports sharing explicitly

Path compression

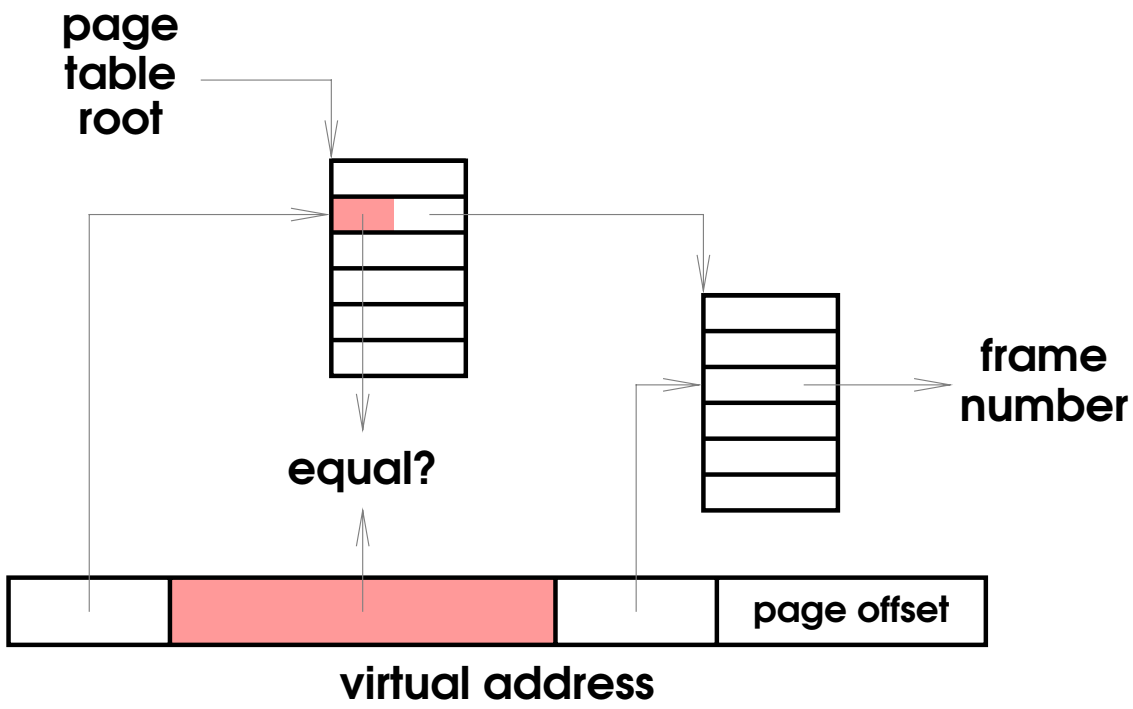
Sparsity often creates **one-way paths** of tables in MLPT.



Idea: bypass these paths.

Guarded page table (GPT)

A path-compresed MLPT, invented by Liedtke (1995).



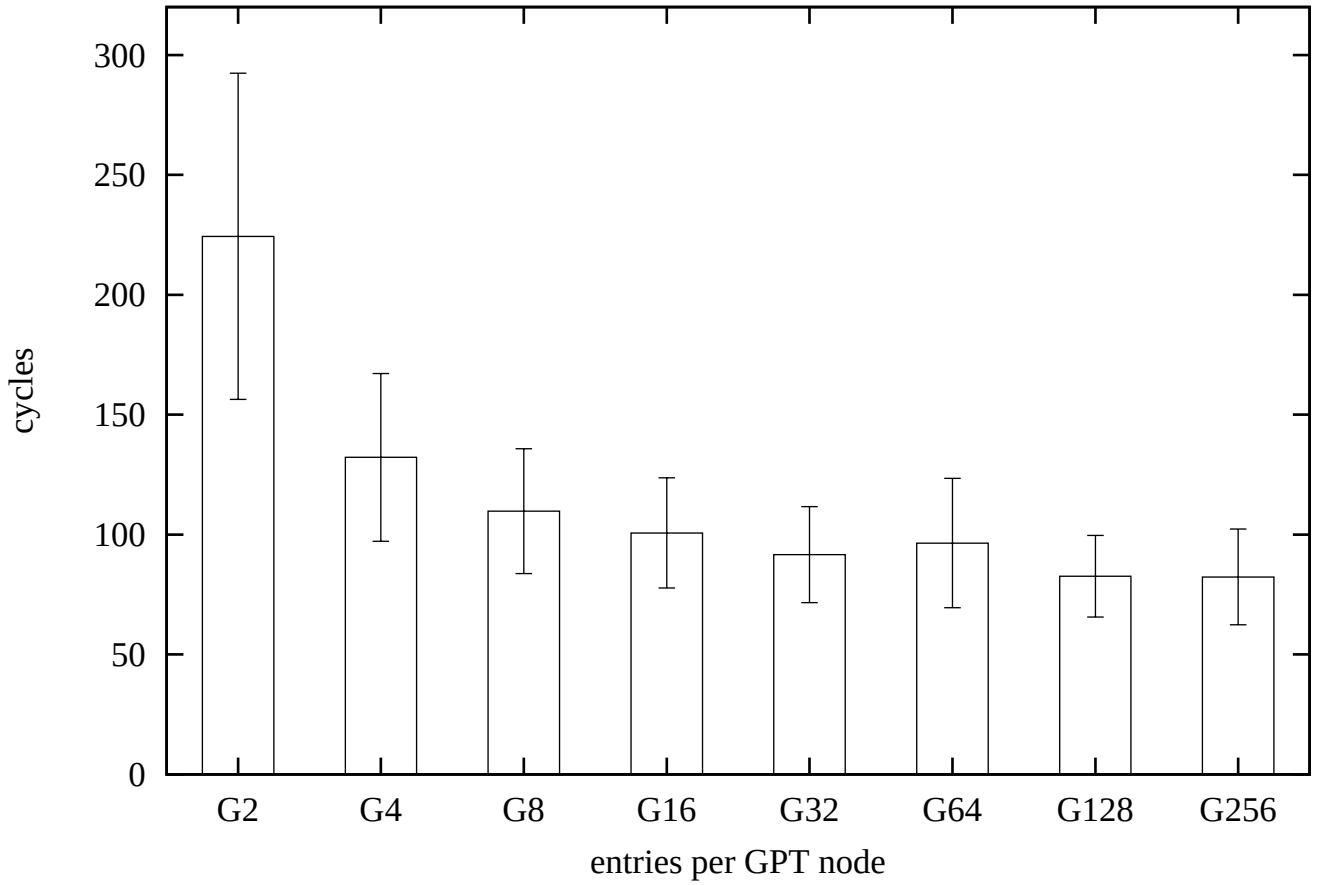
- check skipped address bits (`guards') during lookup
- takes advantage of sparsity in the address space

GPT performance

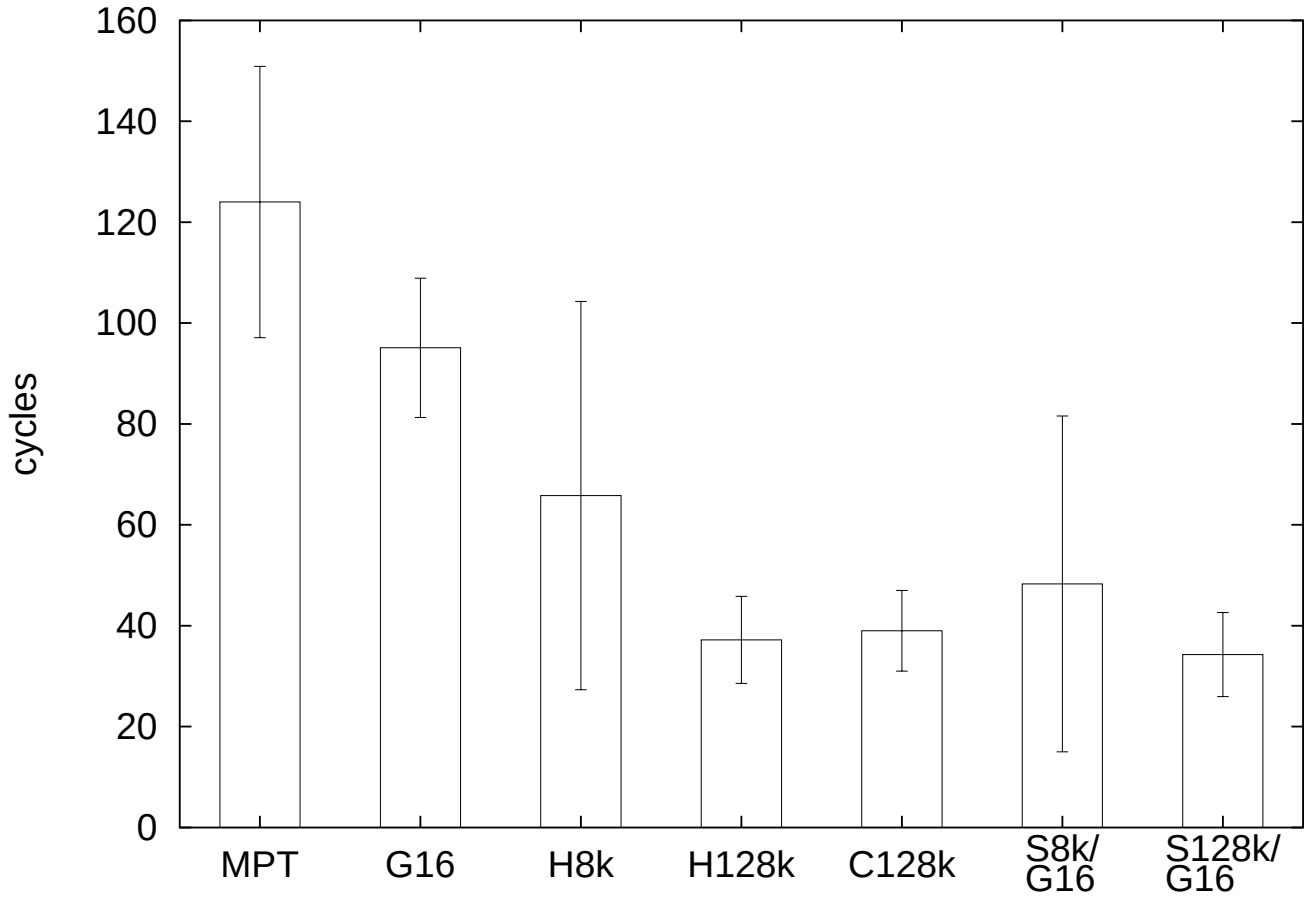
Elphinstone (1999) studied GPT and various other page tables, using L4/MIPS as a testbed.

source	name	size (M)	type	remarks
	go	0.8	I	game of go
	swim	14.2	F	PDE solver
SPEC	gcc	9.3	I	GNU C compiler
CPU95	compress	34.9	I	file (un)compression
	apsi	2.2	F	PDE solver
	wave5	40.4	F	PDE solver
	c4	5.1	I	game of connect four
	nsieve	4.9	I	prime number generator
Alburto	heapsort	4.0	I	sorting large arrays
	mm	7.7	F	matrix multiply
	tfftdp	4.0	F	fast fourier transform

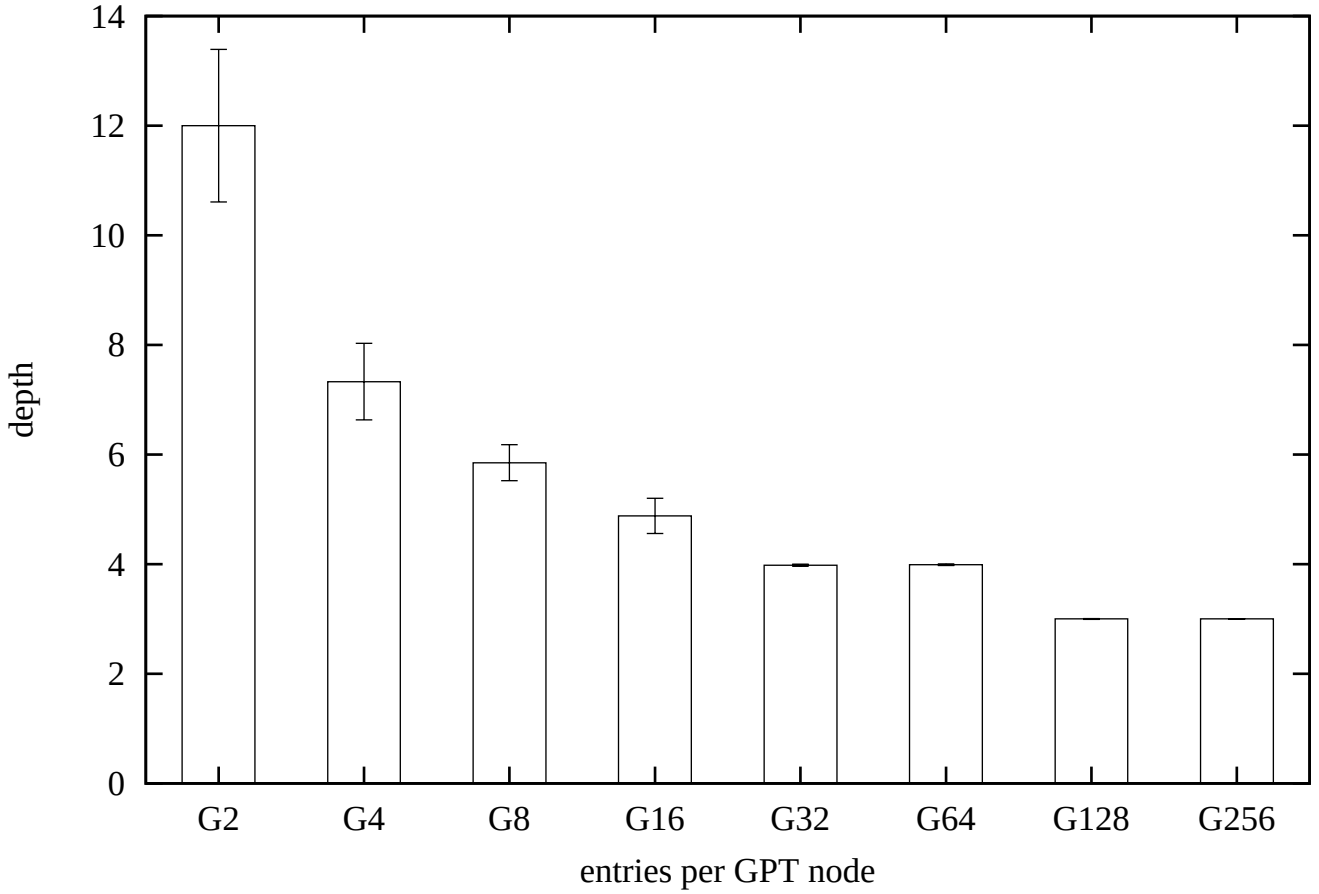
GPT refill time



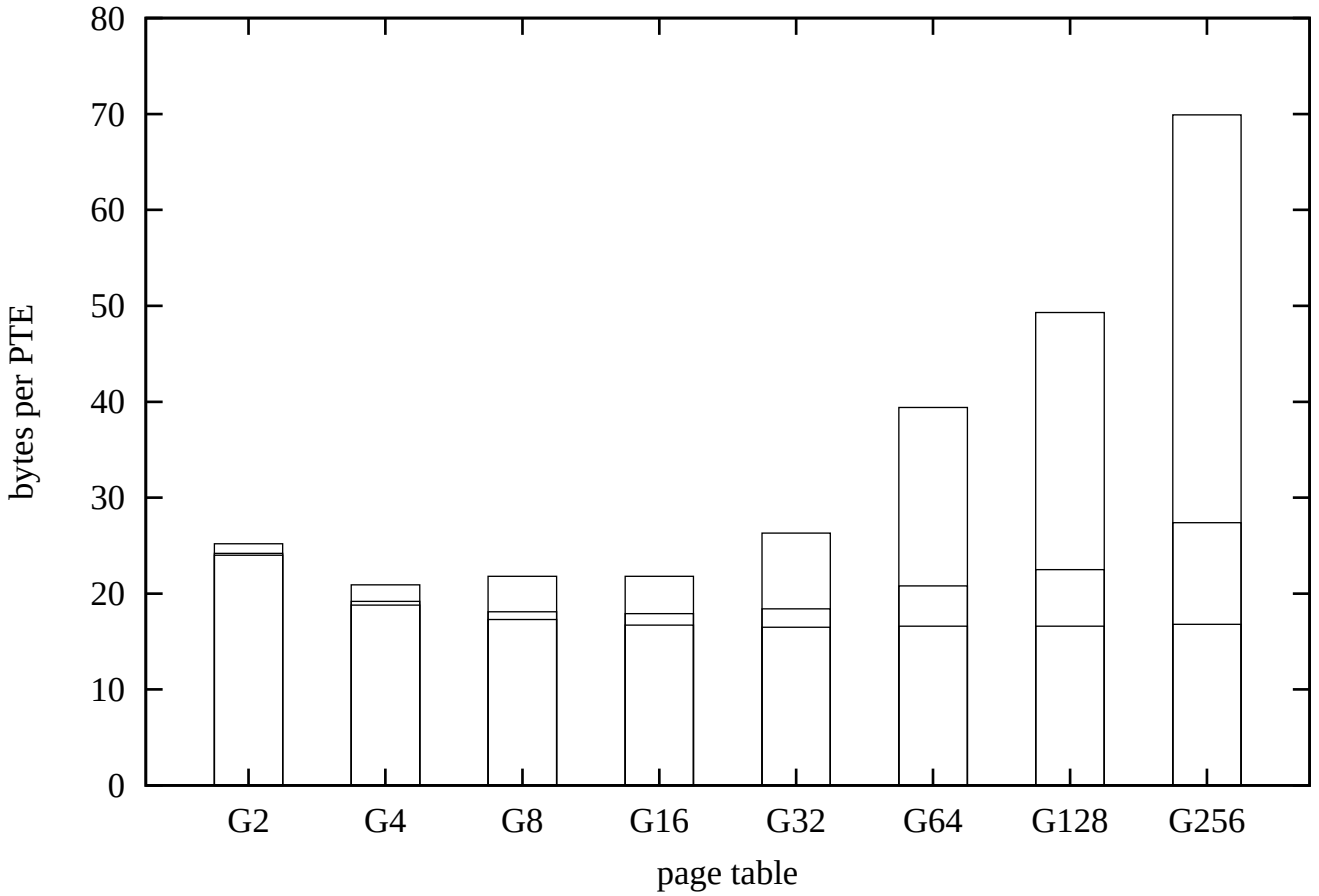
GPT versus other page tables



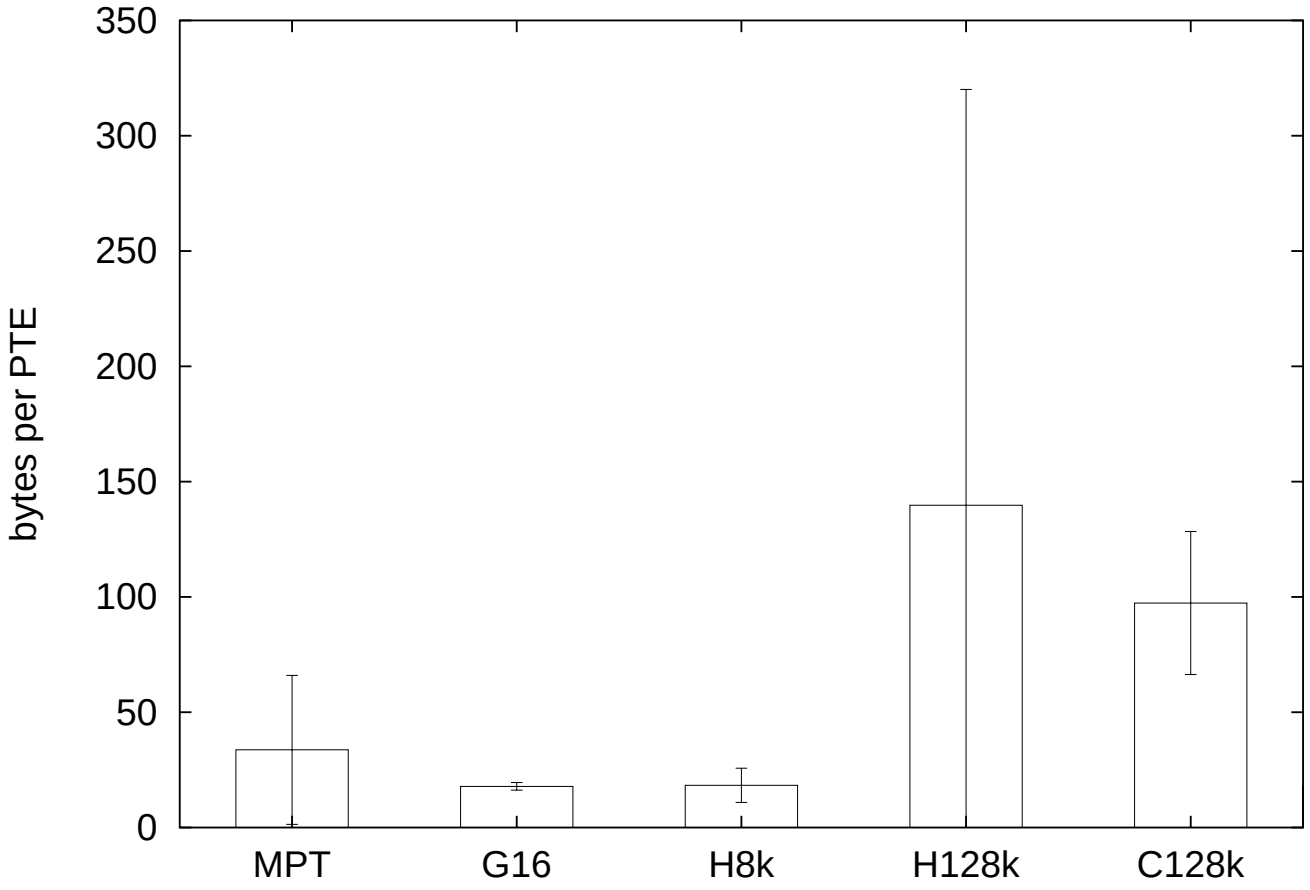
GPT depth


14
12
10
8
6
4
2
0
depth
G2
G4
G8
G16
G32
G64
G128
G256
entries per GPT node

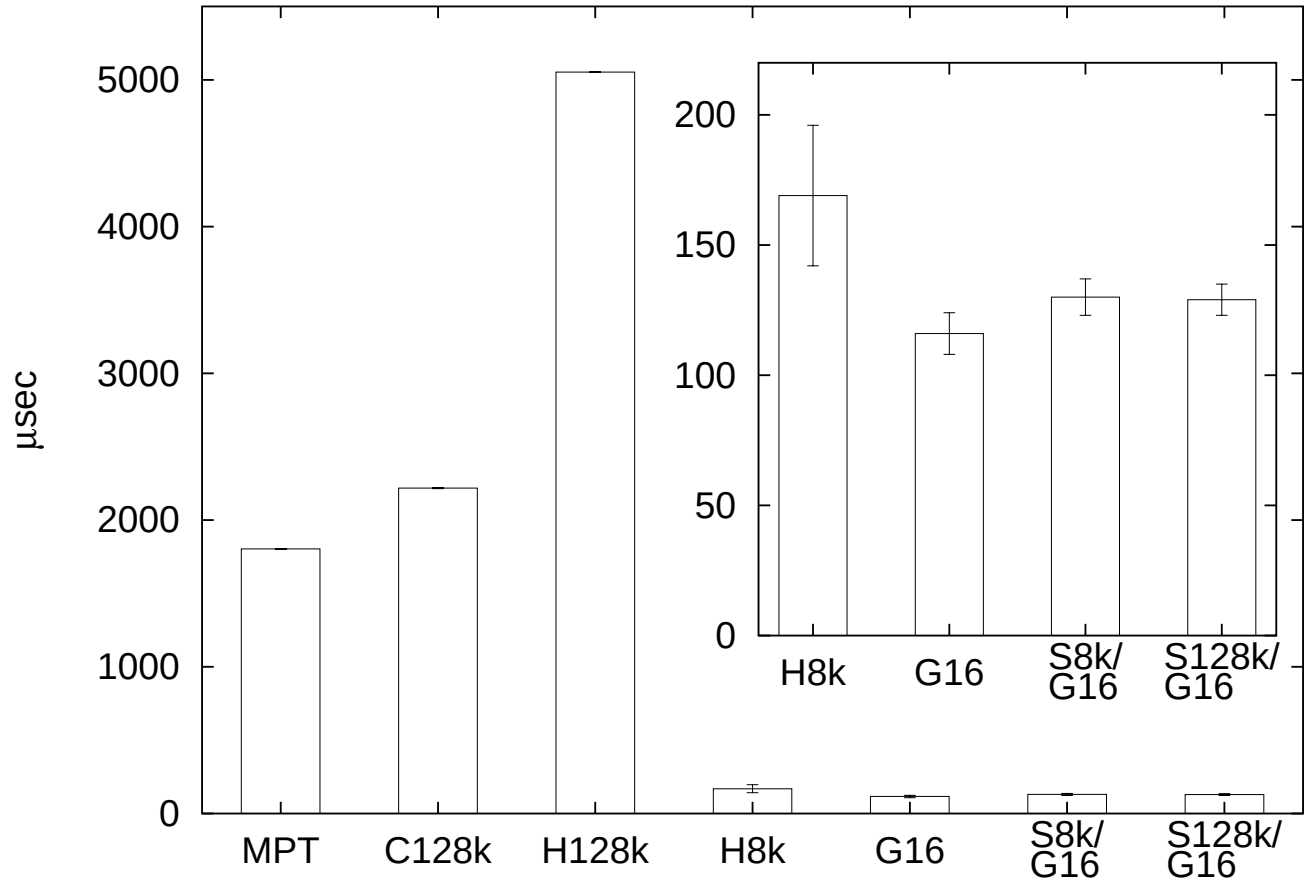
GPT space


80
70
60
50
40
30
20
10
0
bytes per PTE
G2
G4
G8
G16
G32
G64
G128
G256
page table

GPT versus other page tables

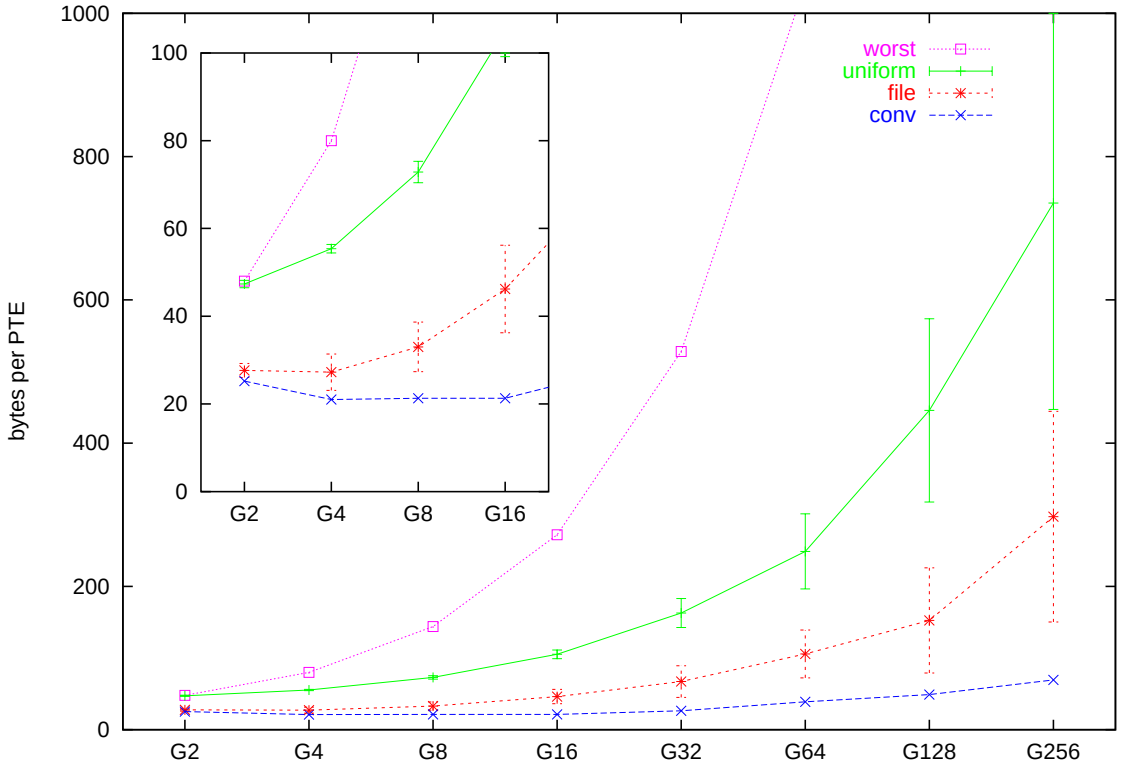


Address space establishment/teardown cost



Other benchmarks

- sparse benchmark: uniformly distributed pages
- file benchmark: uniformly distributed objects



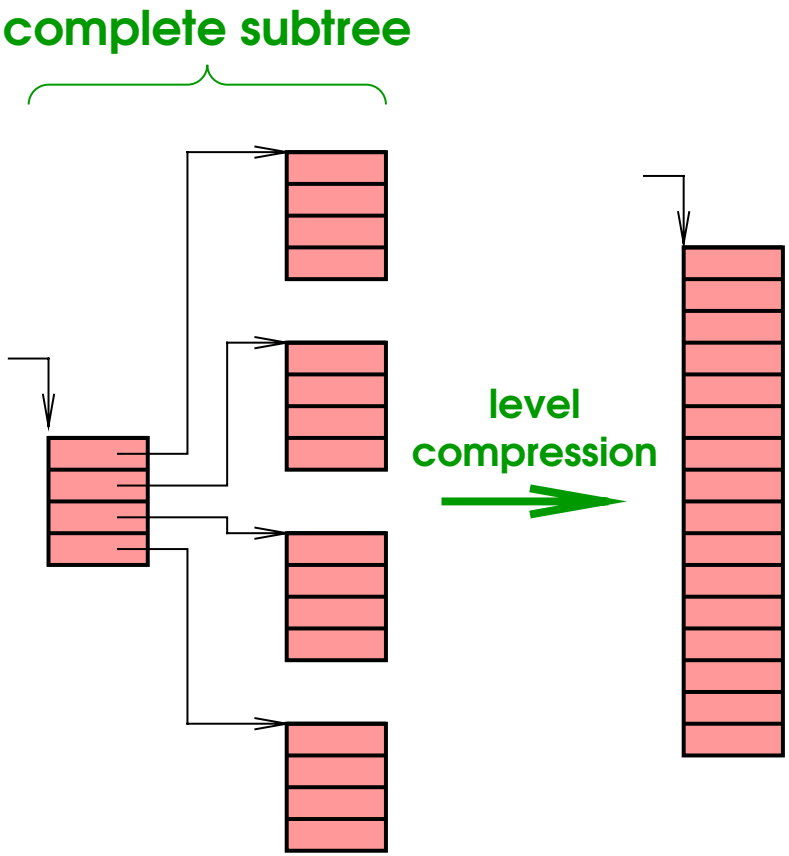
GPT conclusions

- low establishment/teardown cost
- small GPT node size saves space, especially for sparse distributions
- tree depth can become a problem, especially for dense distributions

L4/MIPS solution: use GPT with a software TLB.

Level compression

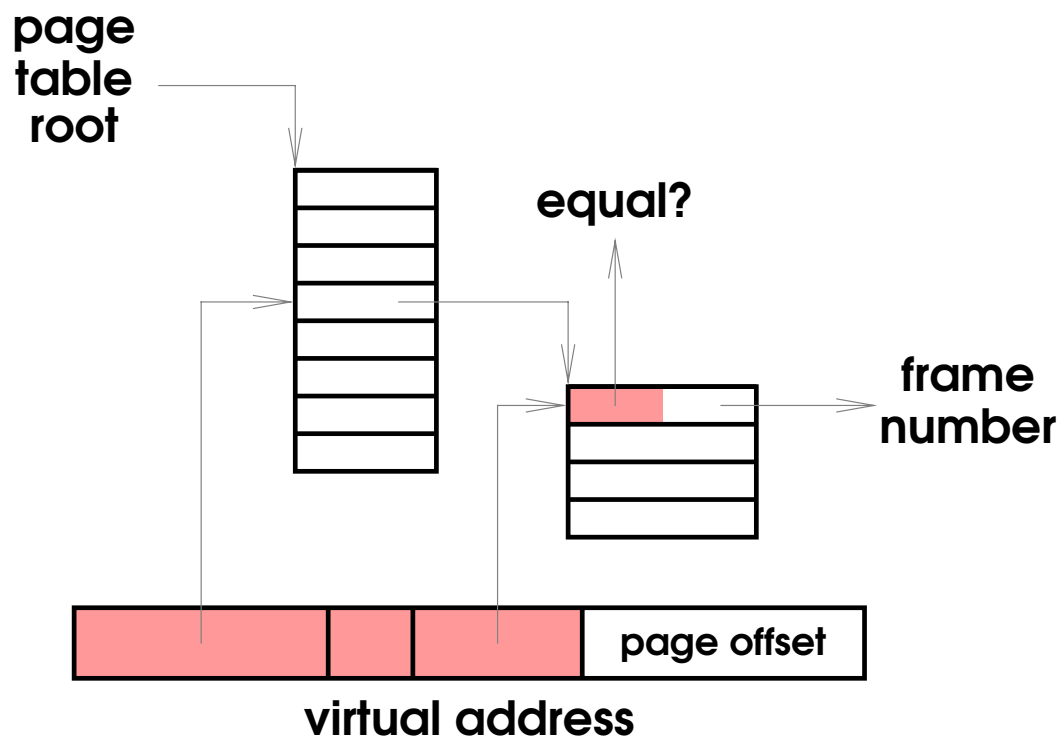
Density creates a lot of **complete** subtrees in MLPT and GPT.



Idea: compress complete subtrees.

Variable radix page table (VRPT)

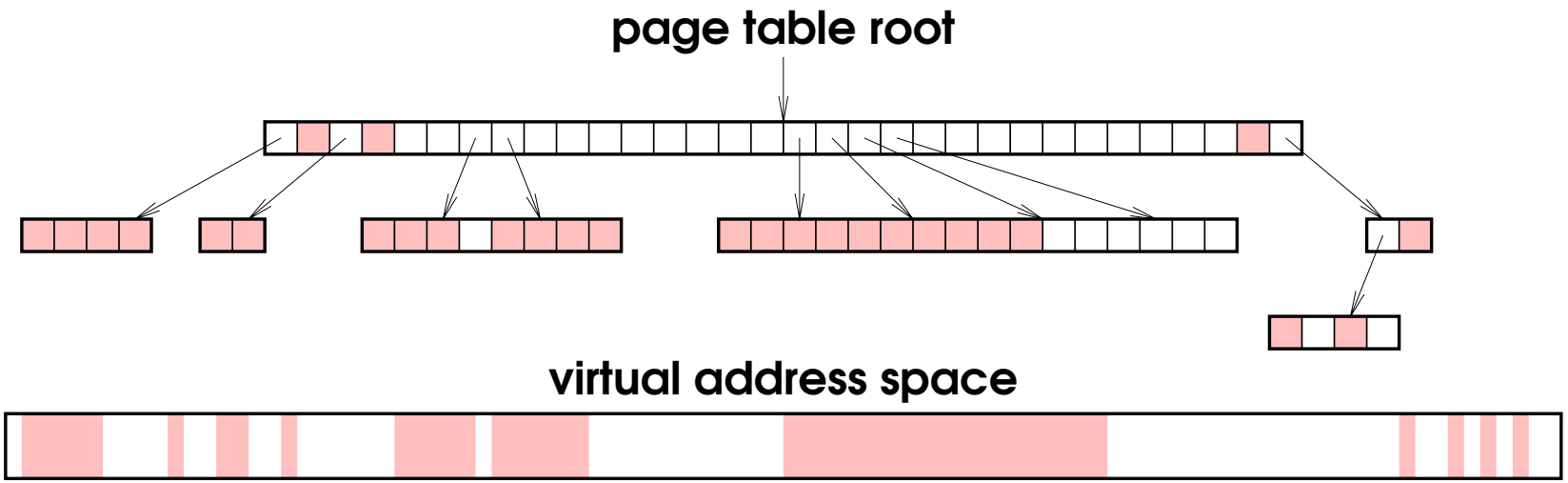
Page table applying **both** level compression and path compression.



All guard comparisons deferred until a leaf is reached.

VRPT structure

- page table size may be any power of two
- best of both worlds: **shallow** and **space-efficient**
- cheap to create, delete, and traverse

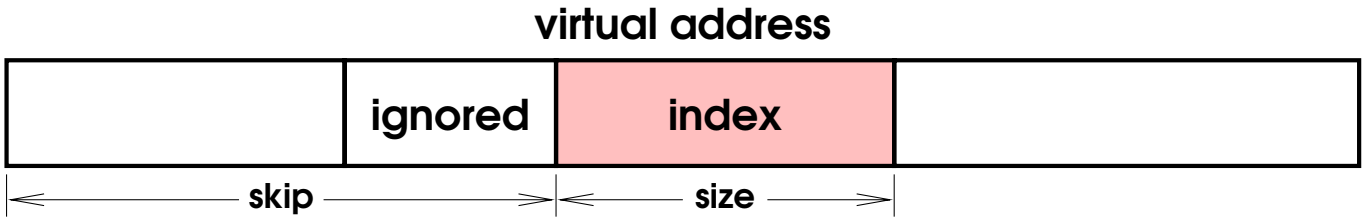


Leaves may occur at any level of the page table.

VRPT implementation

Each node:

- skips a variable number of bits (called **skip**)
- indexes with a variable number of bits (called **size**)



In software, each indexing step takes two SHIFTs and an OR.

Hardware could use a bitfield extraction unit.

VRPT vs. GPT

Elphinstone (1999) derived the following GPT algorithm.

repeat {

$u = v \gg (v_{len} - s)$

$g = (p + 32u) \rightarrow guard$

$g_{len} = (p + 32u) \rightarrow guard_{len}$

if $g == (v \gg (v_{len} - s - g_{len}))$ **and** $(2^{g_{len}} - 1)$ {

$v'_{len} = v_{len} - s - g_{len}$

$v' = v$ **and** $(2^{g_{len}})$

$s' = (p + 32u) \rightarrow size'$

$p' = (p + 32u) \rightarrow table'$

} **else**

page fault

} **until** p is a leaf

VRPT vs. GPT

After common subexpression elimination, the GPT loop has 17 arithmetic and load operations.

VRPT is much simpler.

repeat {
    $p = \&p \rightarrow ptr[v \ll p \rightarrow skip \gg p \rightarrow size]$
} **until** p is a leaf

if $p \rightarrow virt \neq v$
    page fault

All guard checking is deferred until the end.

The inner loop on the MIPS R4000 requires only 7 instructions.

VRPT implementation

Problem: how to choose radix at each level?

$\rightarrow$ Can complex restructuring algorithms be avoided?

VRPT implementation

Problem: how to choose radix at each level?

$\rightarrow$ Can complex restructuring algorithms be avoided?

In VRPT:

- greedy strategy allocates large tables
- unused space may be reclaimed at any time, and the returned to the memory manager
- power of two regions are managed by a buddy system allocator

Page table structure converges to best time–space tradeoff.

VRPT performance

Benchmarks: selected from SPEC CPU95 and CPU2000.

Page tables:

3LPT	three-level page table (43-bit only)
HPT	hashed page table
CPT	hashed page table with clustered entries
GPT	guarded page table
CACHE	guarded page table with HPT-style cache
VRPT	variable radix page table

Platforms:

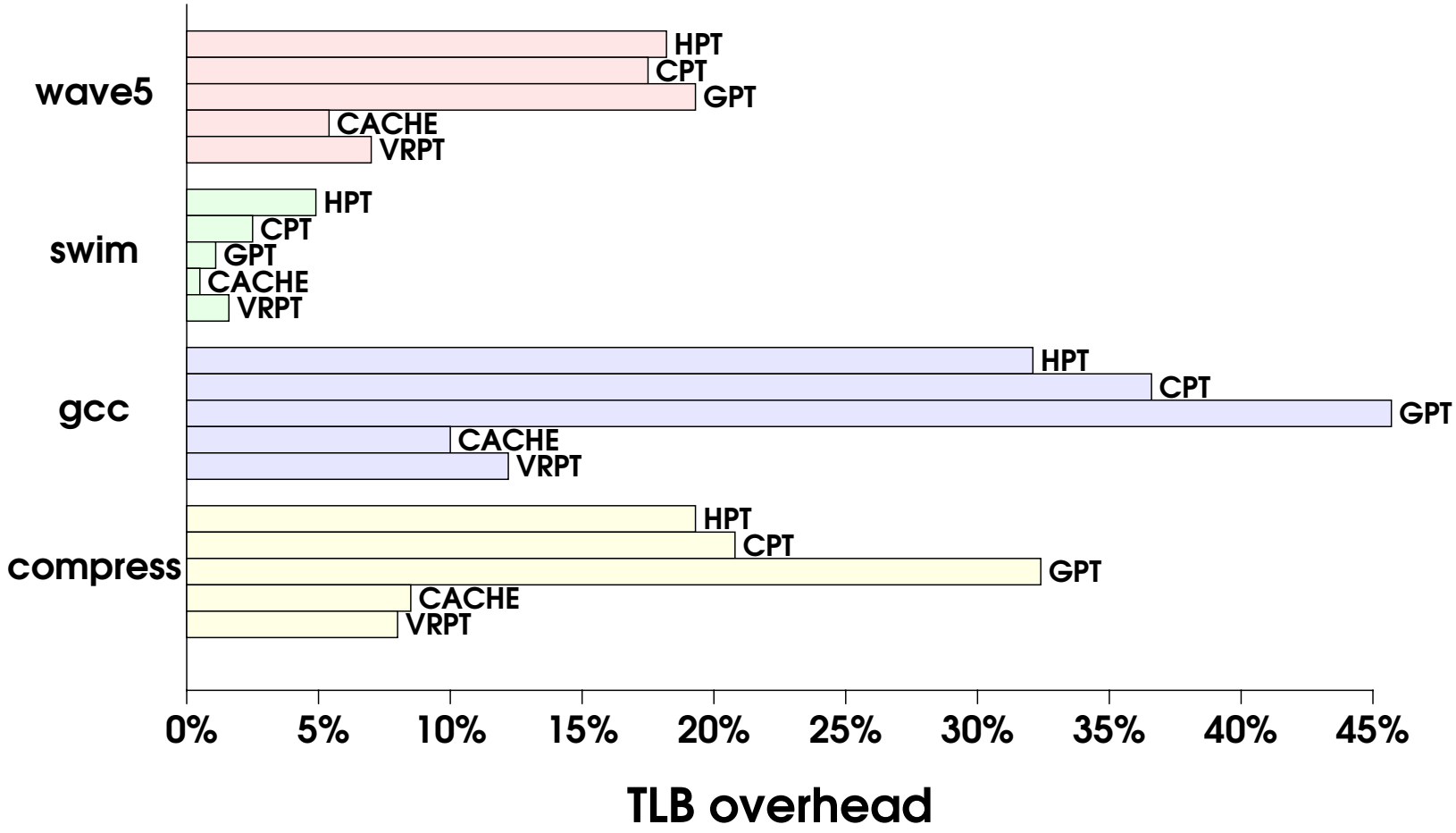
IA-64	IDT MIPS R4700 running L4/MIPS
MIPS64	Intel Itanium running a custom kernel

Methodology

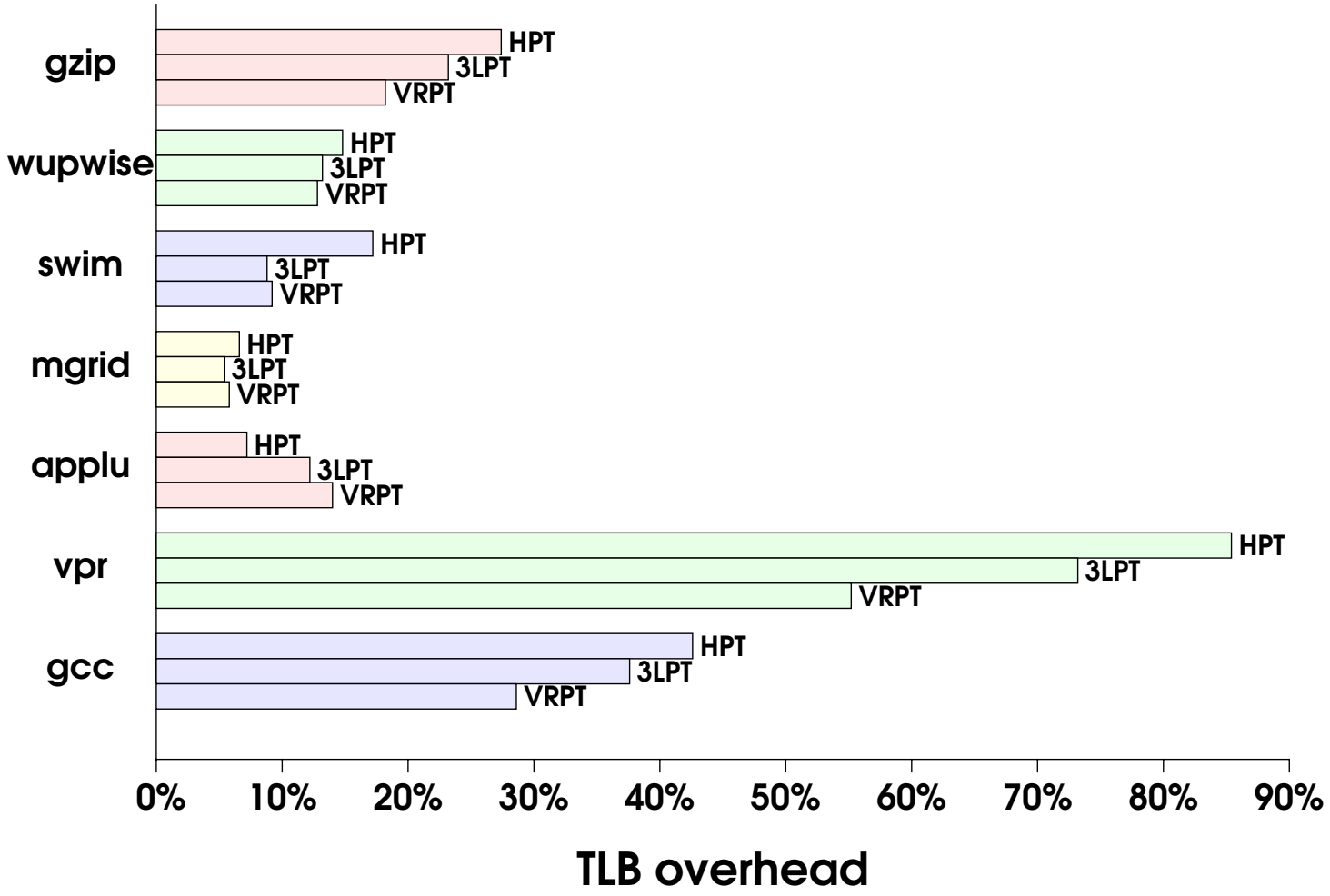
Compare execution times with paging on and off.

- counts direct and indirect costs of VM
  - → cost of exceptions
  - → TLB refill handler
  - → cache pollution
- expressed as per cent overhead

VRPT performance (MIPS64)



VRPT performance (IA-64)



Superpages

Supported by most modern machines.

machine	ITLB	DTLB	page sizes
StrongARM	32	32	4k, 64k, 1M
Pentium III	32	64	4k, 4M
Itanium	64	96	4k, 8k, 16k, 64k, 256k, 1M, 4M, 16M, 64M, 256M, 4G
Alpha 21264	128	128	8k, 64k, 512k, 4M
UltraSPARC	64		8k, 64k, 512k, 4M
MIPS R4000	96		4k, 16k, 64k, 256k, 1M, 4M, 16M
PowerPC 601	256		4k

Superpages

OS may use superpages in a variety of ways.

- globally increase page size (easiest)
- page size per process or segment
- arbitrary page size mixtures (hardest)

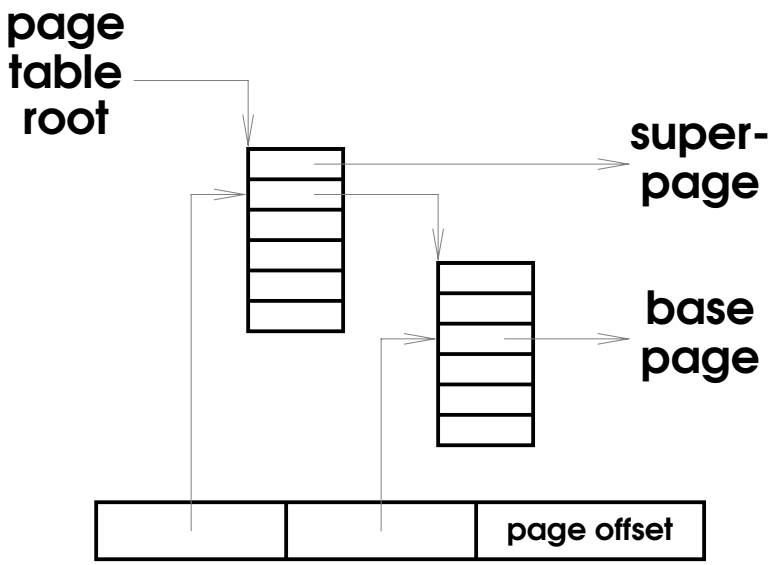
Navarro et al. (2002) achieved significant speedup with mixed page sizes.

$\rightarrow$ "performance benefit often exceeds 30%"

Superpage representation in the page table

In many PTs it is difficult to accommodate mixed page sizes.

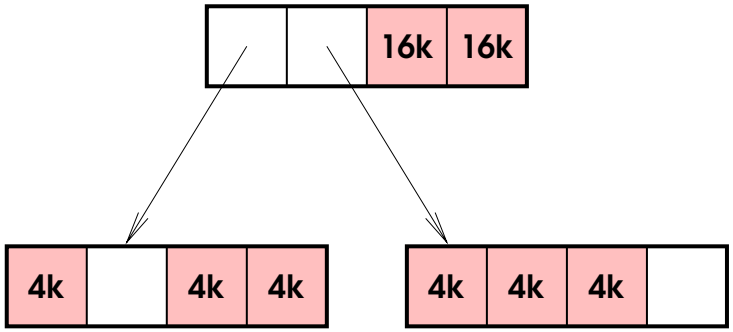
- HPT cannot support superpages: PTEs must be replicated
- MLPT supports a limited number of superpages



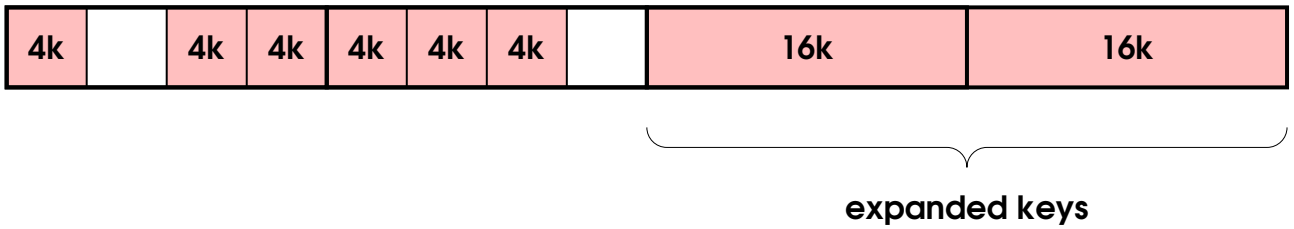
Other page sizes must be replicated.

Superpage factorization in VRPT

VRPT accommodates mixed page sizes using **PTE factorization**.

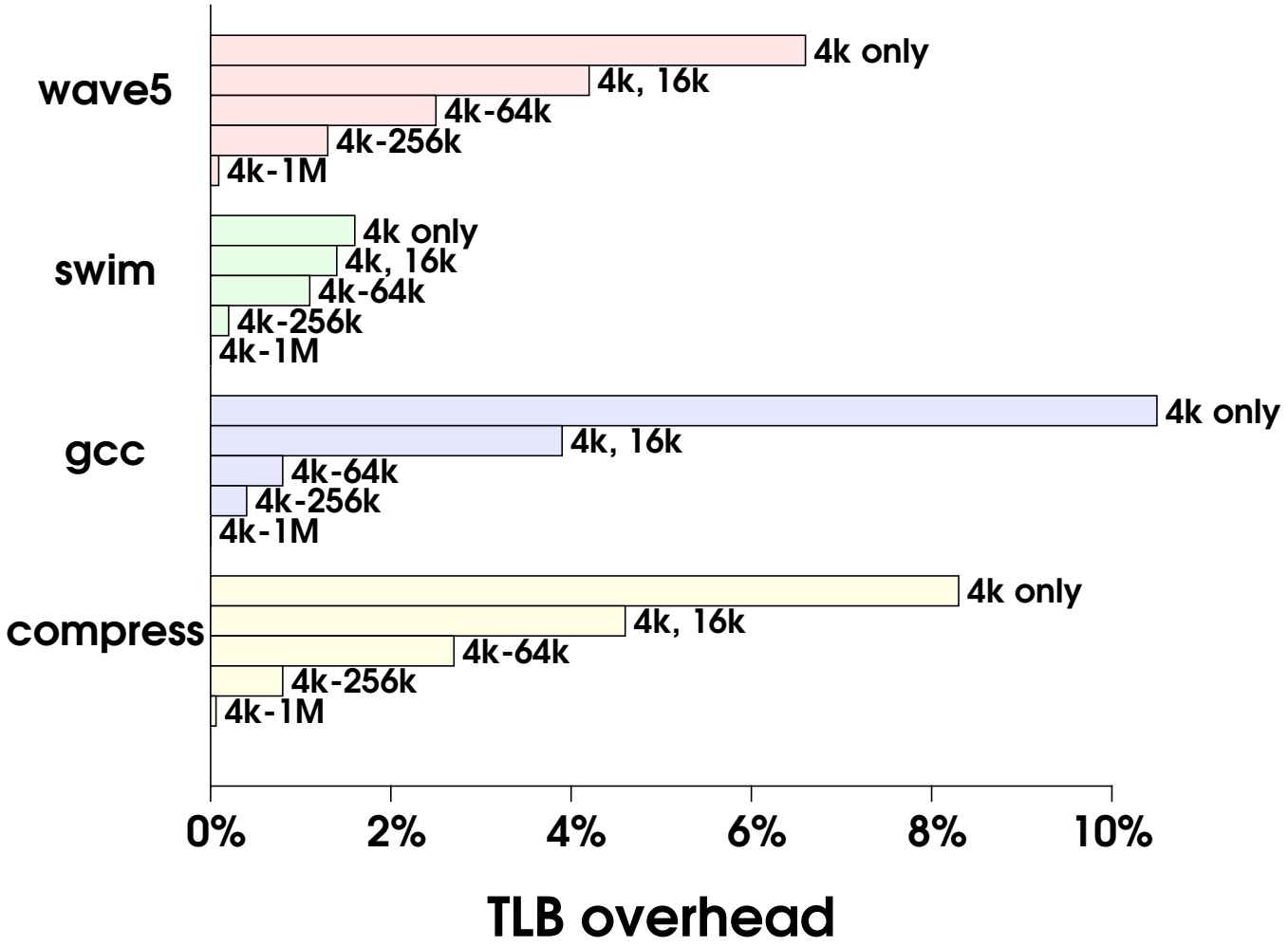


Problem: fragmentation if too many sizes. **Solution:** key expansion.

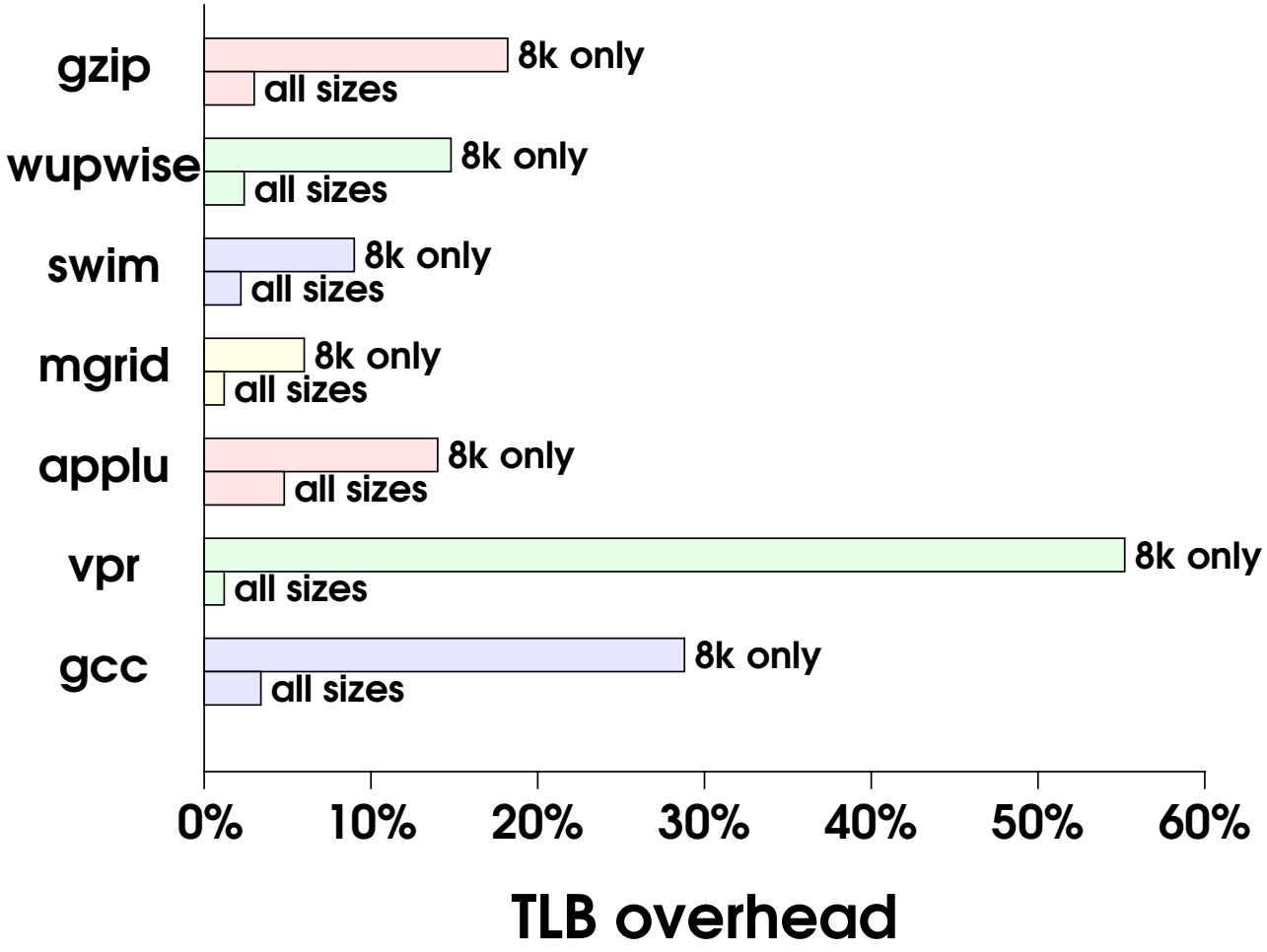


Expansion and **factorization** can be applied flexibly as desired.

Superpage performance (MIPS64)

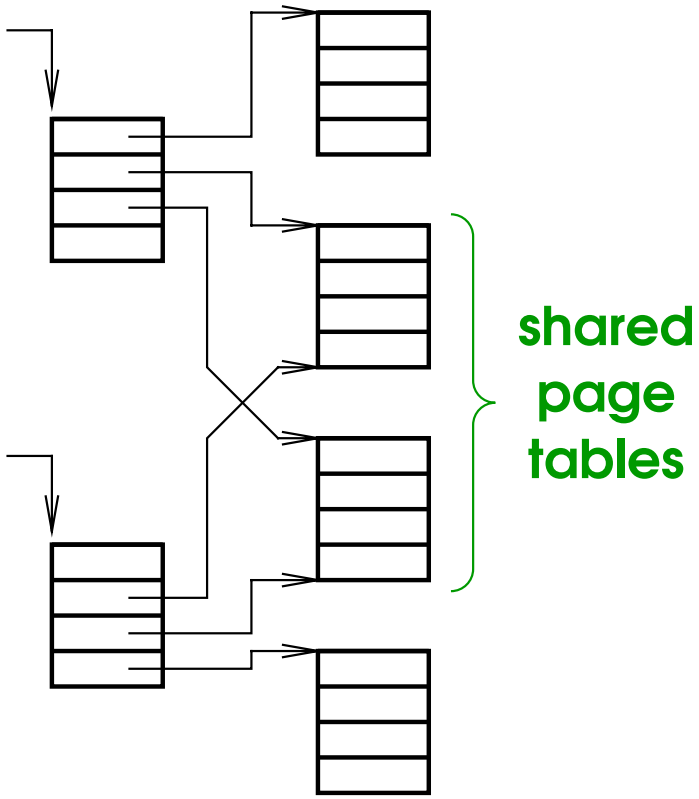


Superpage performance (IA-64)



Sharing in VRPT

VRPT can use **cross-linked** page tables to share segments.



Technique is also available with MLPT, but less flexible.

Sharing in VRPT

Advantages:

- Updating a shared segment is easier.
- Sharing is explicit: easy to apply shared TLB tags.
- Each page doesn't have to be mapped to each space individually.

  $\rightarrow$ reduce **minor page faults**

- A shared segment can have different protection attributes in different address spaces.

Performance evaluation: **future work**.

VRPT conclusions

- VRPT is competitive with other page tables.
- VRPT is competitive with a memory-based TLB cache.
- VRPT may be suitable for hardware implementation.
- Superpages show dramatic performance improvement.

VRPT is currently being implemented in the Pistachio L4 kernel in a VM subsystem called Calypso.

Microkernels and SASOSes

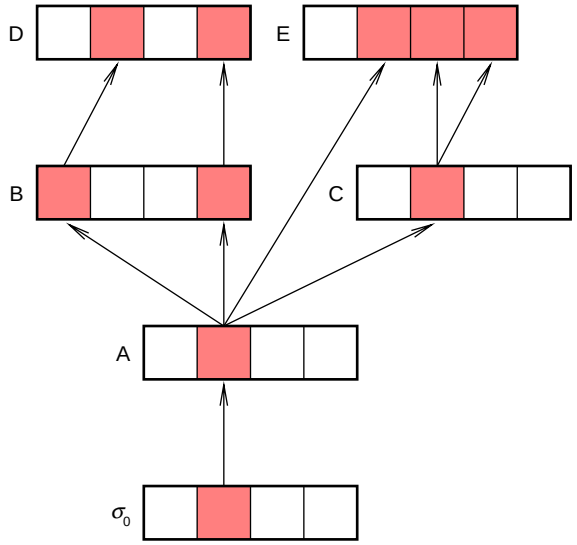
Some microkernel and SASOS features exacerbate VM load.

- service decomposition
  - → frequent address space switches
  - → reduces locality
  - → kernel consumption of TLB entries
- sparse address space layout
- shared memory
  - → increases fragmentation
  - → reduces locality
  - → requires more TLB entries to cover physical memory

VM performance is critical to the success of microkernel and SASOS systems.

Implementation in L4

L4 provides three operations: **map**, **grant**, and **unmap**.

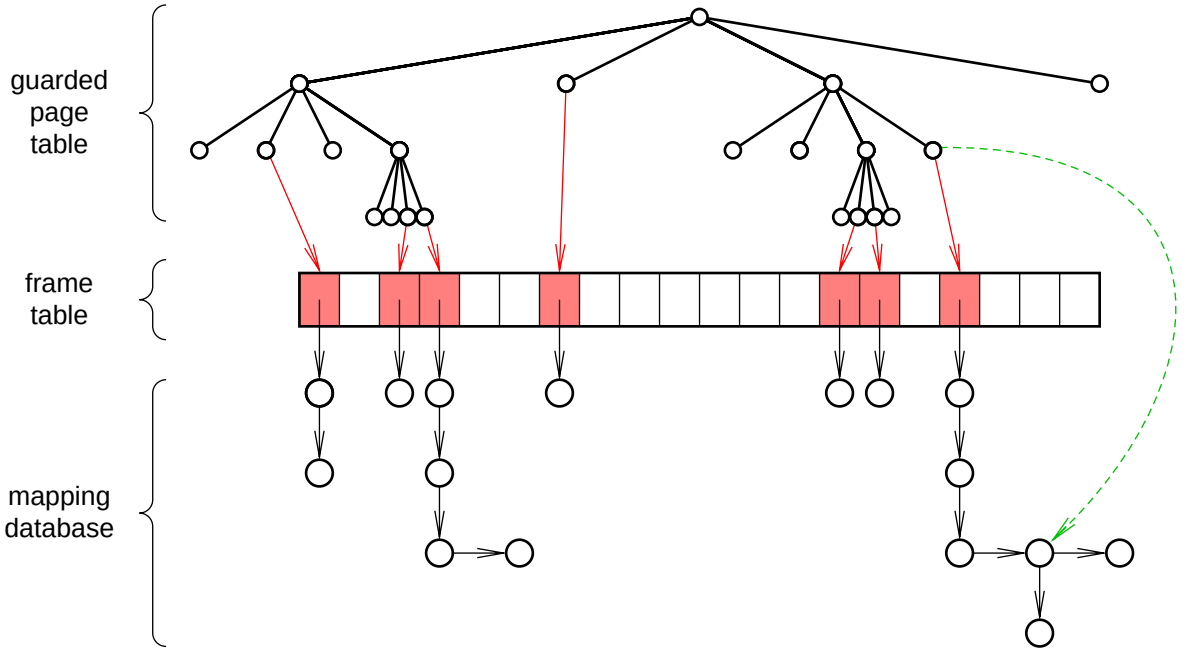


L4 must remember the history of **map** operations in the **mapping database**, to allow future undo with **unmap**.

Memory management is the responsibility of *user-level pagers*.

L4 data structures

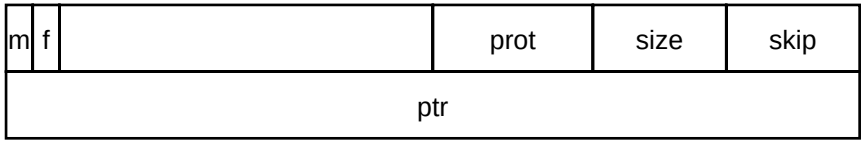
L4 implementation of recursive address spaces uses roughly the following data structures.



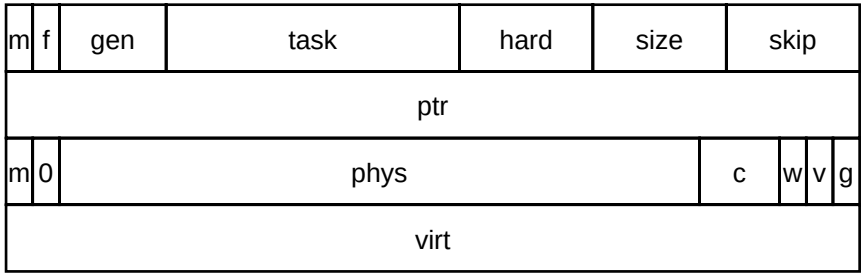
Direct pointers between GPT and mapping database (green arrow) were considered by Elphinstone, but rejected to allow PT implementation freedom.

Calypso data structures

- internal nodes store two shift amounts and a pointer



- PTE stores virtual address (and other goodies)

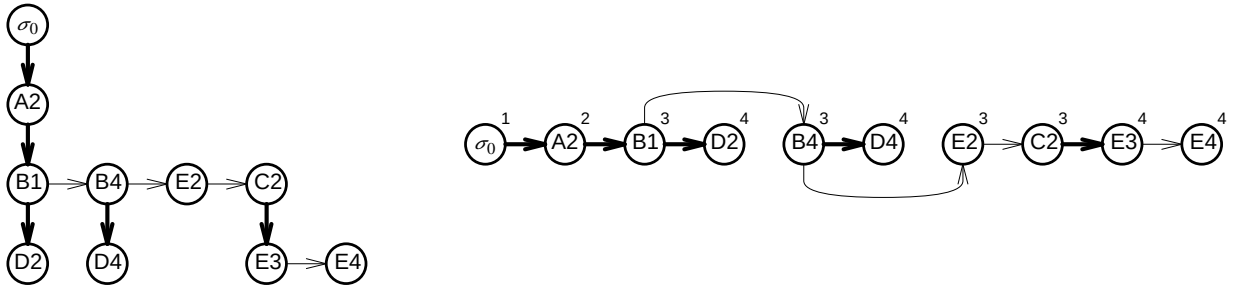


Each PTE may represent any (hard) page size.

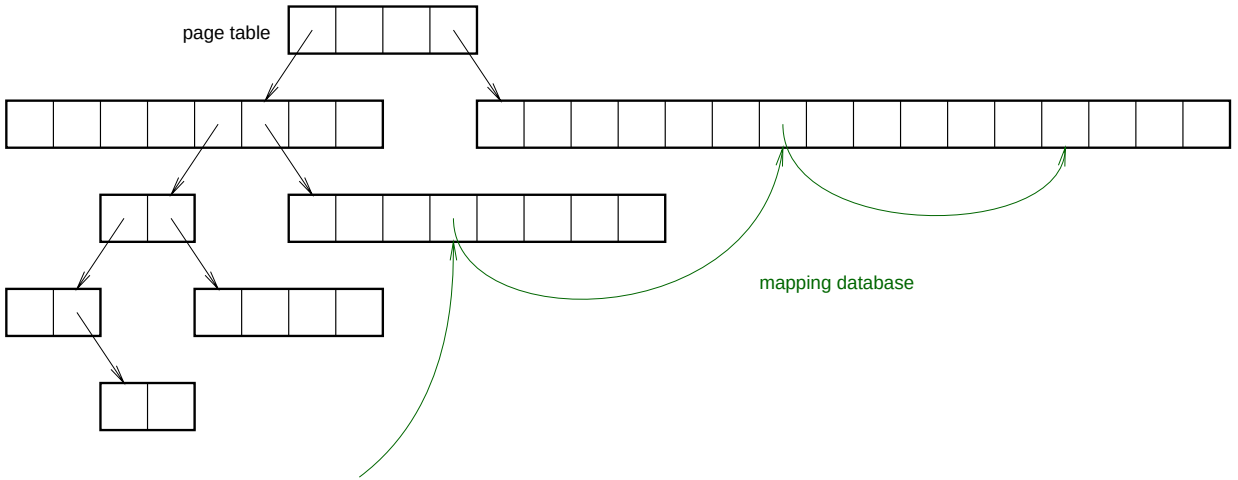
Page tables may be shared (with an addition to the L4 API).

Calypso mapping database

Topologically sort each mapping graph into a singly-linked list.



Integrate the mapping database list into the PTEs.



Sharing in L4

Difficult to cross-link page tables and use hardware sharing support.

- L4's **map** primitive can only map to one AS at a time.
- Calypso needs an API extension for sharing support.

Extensions to the L4 API must be designed with care.

- shouldn't slow down critical functions (IPC and TLB refill)
- shouldn't introduce policy into the kernel
- should match the `L4 philosophy'

Link

Experimental new VM operation: **link**, which establishes a shared domain between pager and pagee.

Link is like **map**, except that future maps and unmaps in that part of the pager's address space *are automatically seen by the pagee*.

Best illustrated by an analogy.

L4 primitive	Unix analogy
unmap	`rm`
map	`cp`
grant	`mv`
link	`ln -s`

More on link

Restrictions:

- virtual address in pager and pagee must be equal
- fpage size may be restricted

Advantages:

- natural generalization of **map** and **grant**
- reduces kernel crossings
- reduces page fault IPC
- restricted by L4's usual IPC confinement model (e.g. clans and chiefs)

Conclusions

- VM overhead is a critical performance issue in modern hardware and operating systems.
- Conventional page tables don't perform well in these conditions.
- Software TLB (or hardware TLB cache) is the best solution to a slow page table.
- VRPT can perform as well as software TLB.
  - Further performance evaluation required.
- Optimization of the critical path pays off.
  - but **only** after evaluation and measurement.

References and further information

http://www.cse.unsw.edu.au/~cls/