The Linux Virtual Memory System

Patrick Ladd
Technical Account Manager
pladd@redhat.com / pmladd@gmail.com

NY Red Hat Users Group
June 8, 2016

Slides are available at http://people.redhat.com/pladd/

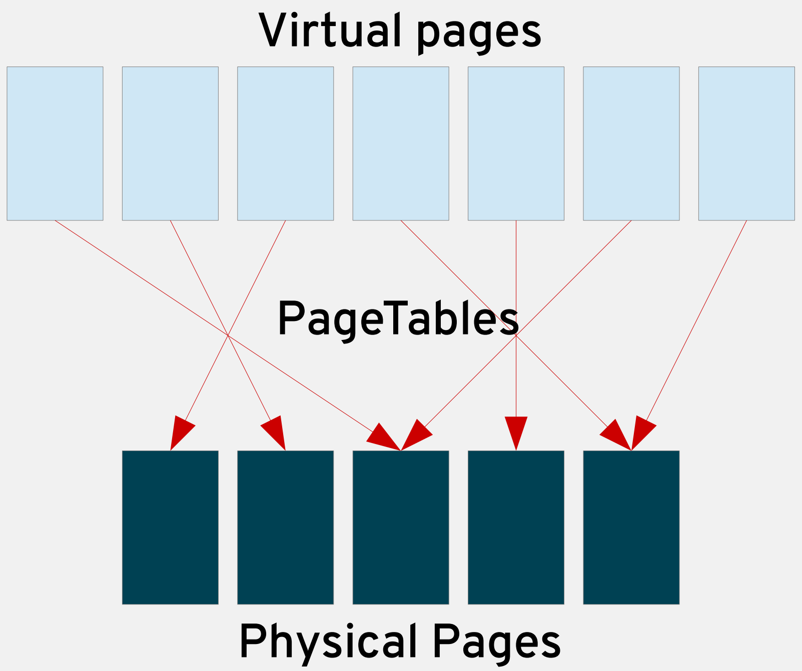
Topics

- Evolution of the Linux Memory System
- Evolution of system memory architectures
- Latest innovations in Linux Memory System
  - NUMA
  - Hugepages

Virtual Memory

Virtual Memory System (simplified)

Click to add subtitle



Virtual pages

- Cost “nothing”
- Basically unlimited on 64 bit architecture

Physical Pages

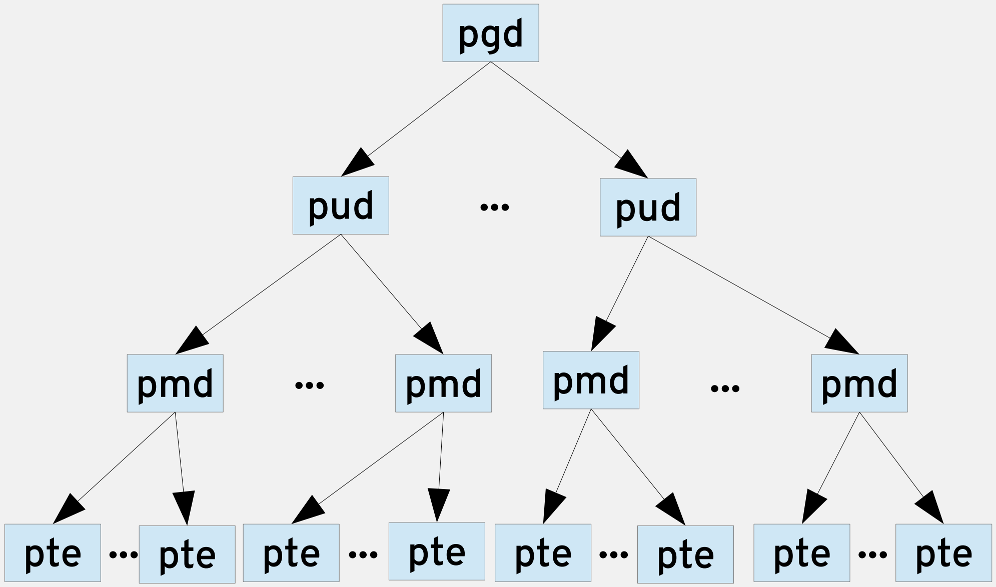
- RAM of system
- Cost money!

PageTables

- Virtual → Physical Mapping

PageTables

Click to add subtitle



Data Structure

- Common code and x86 formats are the same
- All 4K in size
- 2^9 (512) pointers per table
- `grep PageTables /proc/meminfo`

Accessible Memory

- Total:
  - (2^9)^4 ** 4096 → 48 bits
  - 281.5 TB

Virtual Memory Fabric

Click to add subtitle

Data structures for connecting

Hardware constrained structures

- pagetables

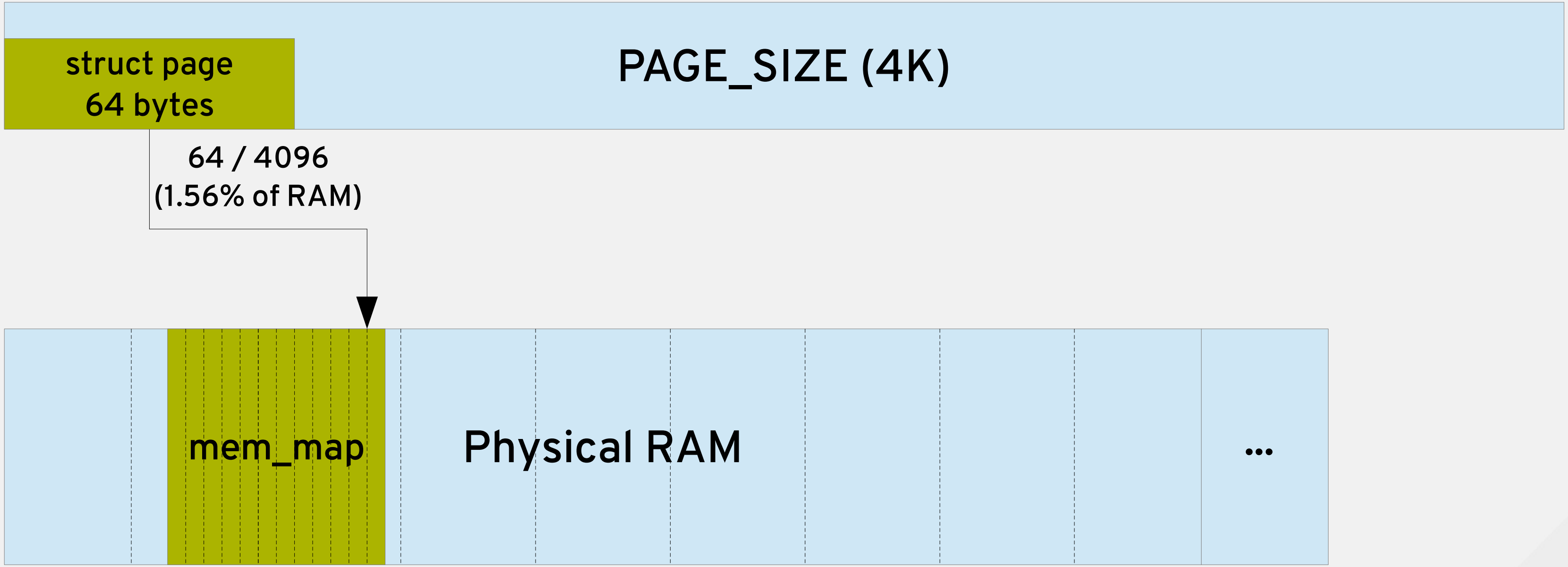
to

Software abstractions

- Tasks
- Processes
- Virtual memory areas (VMAs)
- **mmap (glibc `malloc()`)**

Physical Page and `struct page`

Click to add subtitle



redhat.

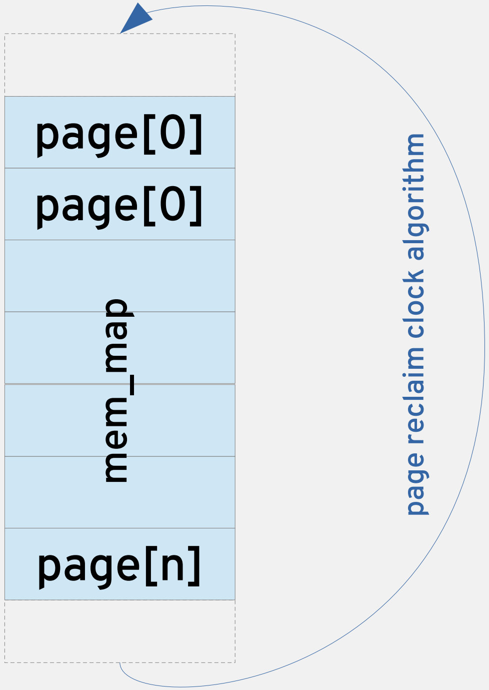
Memory Management Algorithms

Memory System Heuristics

- Algorithms doing computations on memory fabric structures
- Need to solve hard problems with no guaranteed perfect solution
    - When is the right time to unmap pages? (swappiness)
    - Which page should I page out?
- Some design is unchanged
  - Measurement of how hard it is to free memory
  - Free memory used as cache
  - Overcommit by default

Page reclaim clock algorithm

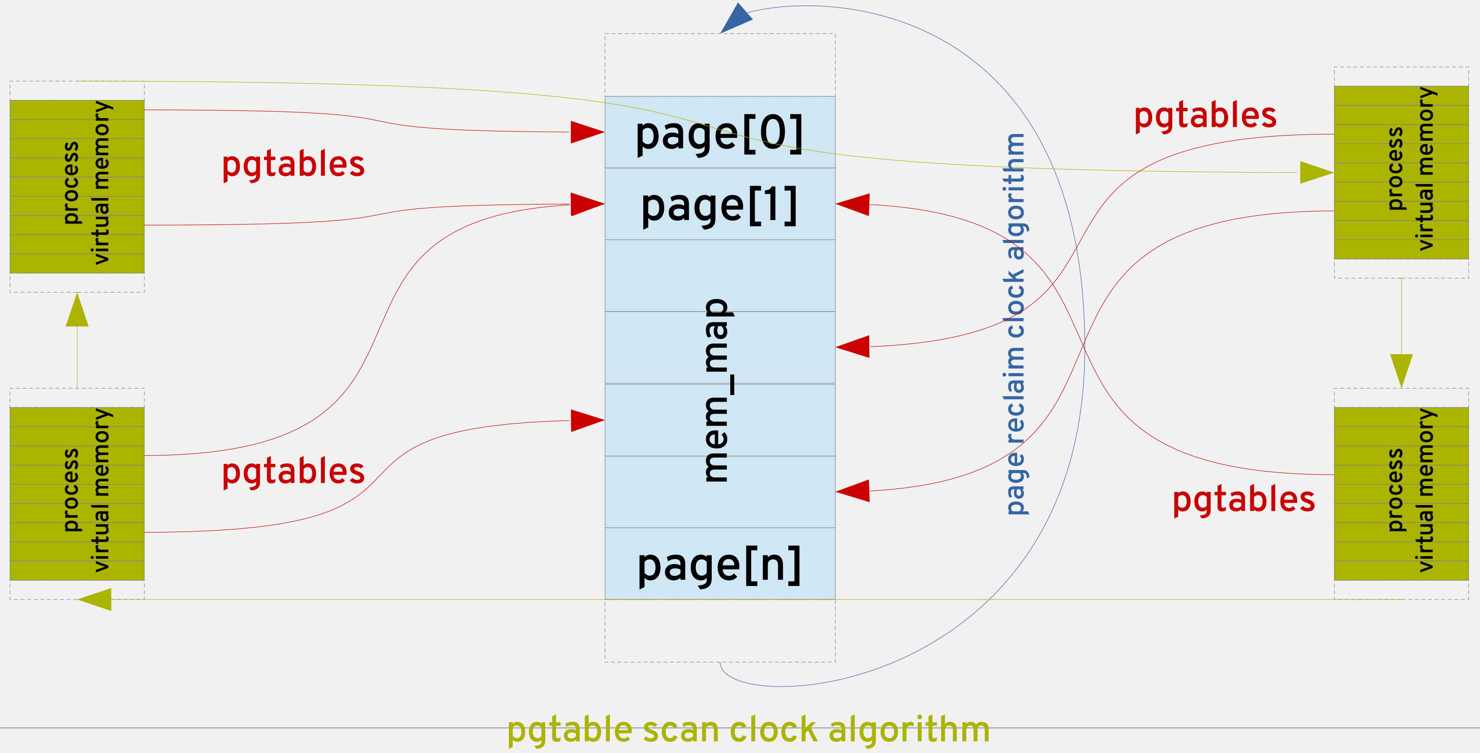
Early kernels (~2.2) Early '90s
Small system RAM sizes



redhat.

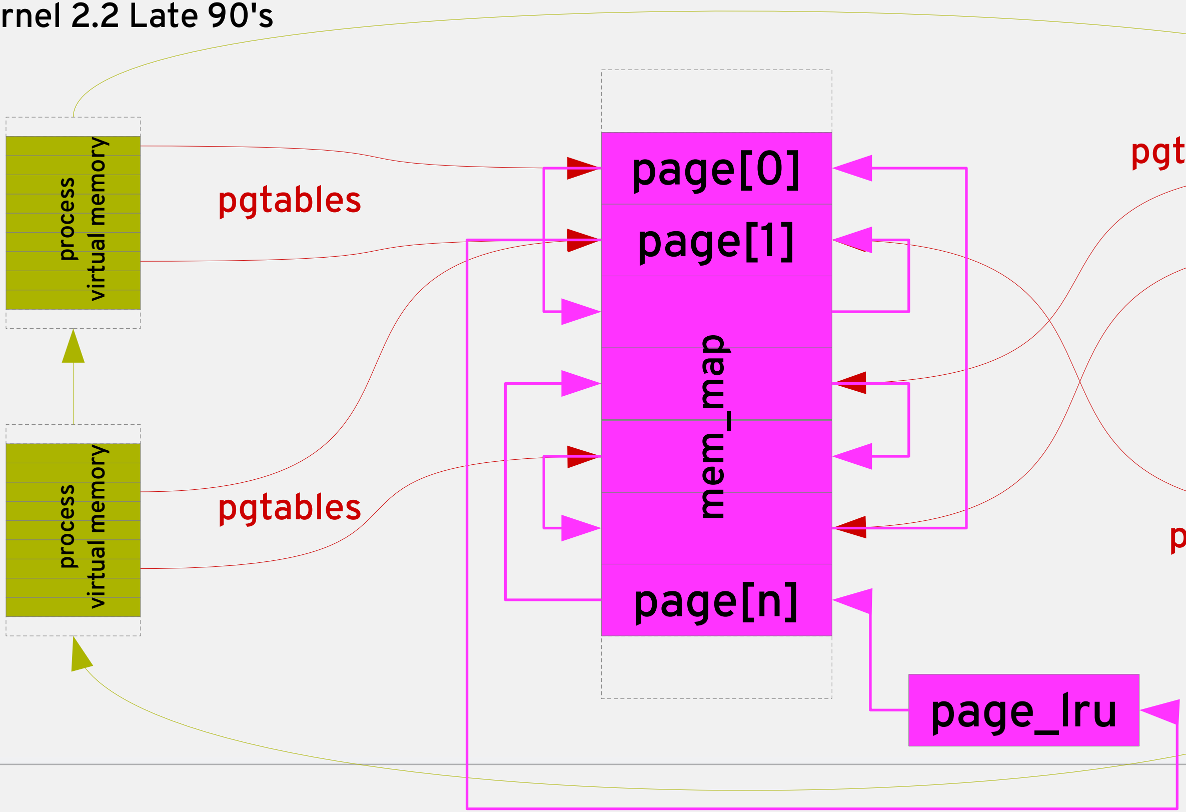
pgtable scan clock algorithm

Kernel 2.2 Mid 90's



Last Recently Used List

Kernel 2.2 Late 90's



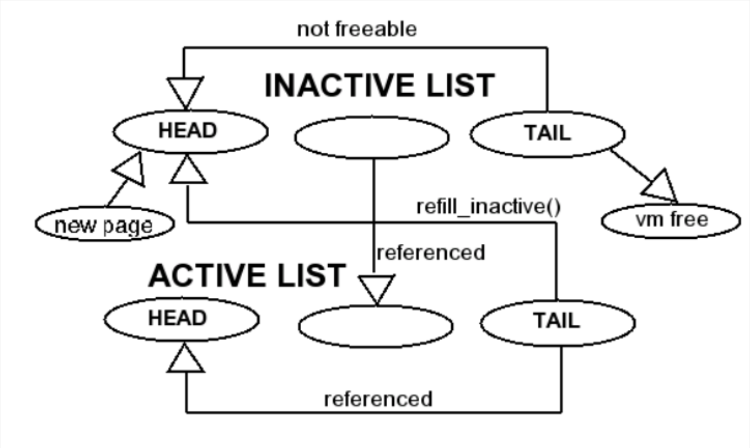
redhat.

Active and Inactive List LRU

Kernel 2.4 - 2001

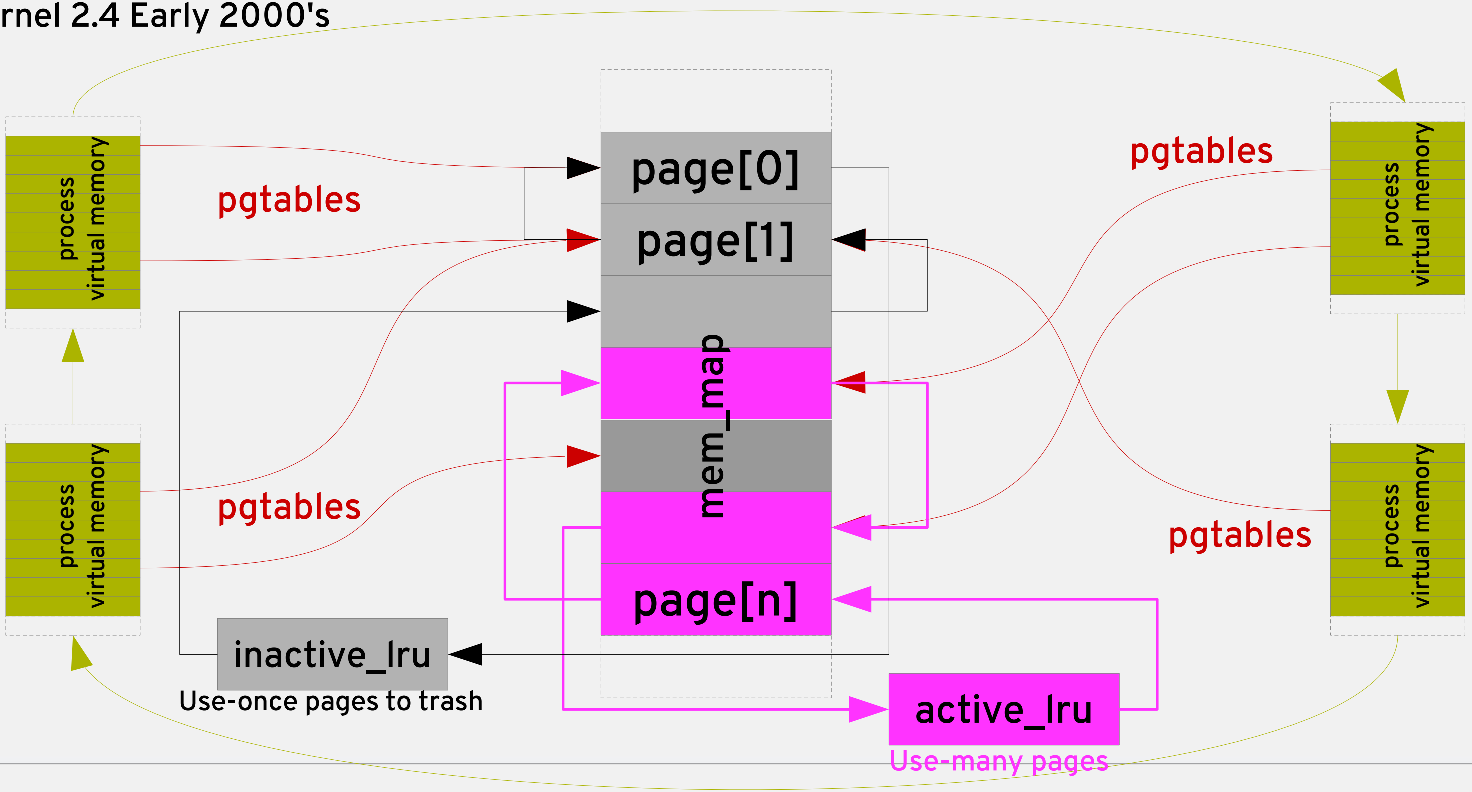
The active page LRU preserves the the active memory working set

- Only the inactive LRU loses information as fast as use-once I/O goes
- Works well enough also with an arbitrary balance
- Active/inactive list optimum balancing algorithm was solved in 2012-2014
  - Shadow radix tree nodes that detect re-faults (more patches last month)



Active & Inactive LRU Lists

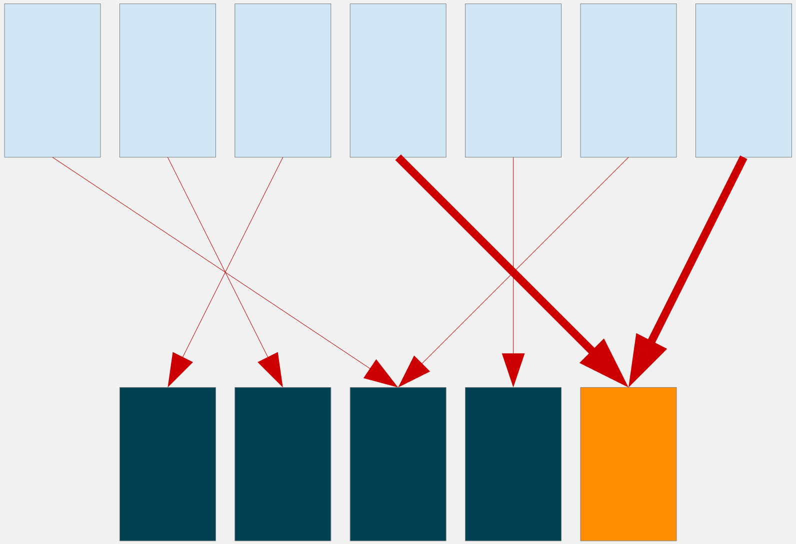
Kernel 2.4 Early 2000's



```
$ grep -i active /proc/meminfo
Active:                11192976 kB
Inactive:               2643936 kB
Active(anon):          10402692 kB
Inactive(anon):         2058248 kB
Active(file):            790284 kB
Inactive(file):          585688 kB
```

rmap Obsoletes pgtable Scan Clock Algorithm

Click to add subtitle



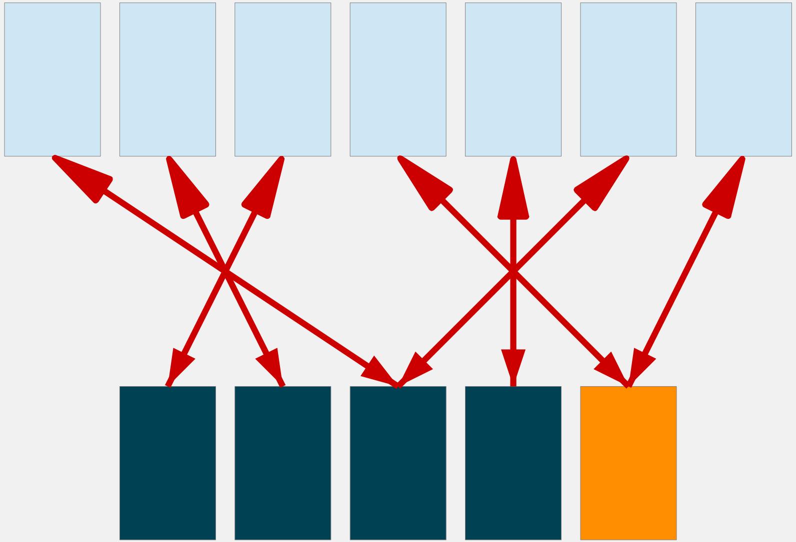
To free a candidate page, we must first drop all references to it (mark the page table entry non-present)

MM & VMA

- mm_struct aka MM
  - Memory of a process
  - Shared by all threads
- vm_area_struct aka VMA
  - Virtual memory area
  - Created and torn down by mmap & munmap
  - Defines the virtual address space of a MM

rmap Obsoletes pgtable Scan Clock Algorithm

Click to add subtitle

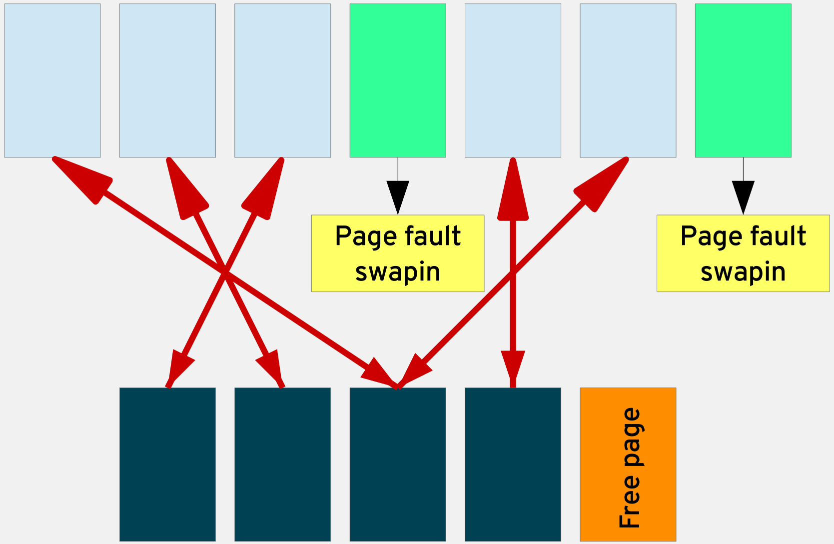


rmap – reverse mapping

- Allows direct connection to pagetables from any given physical page without scanning

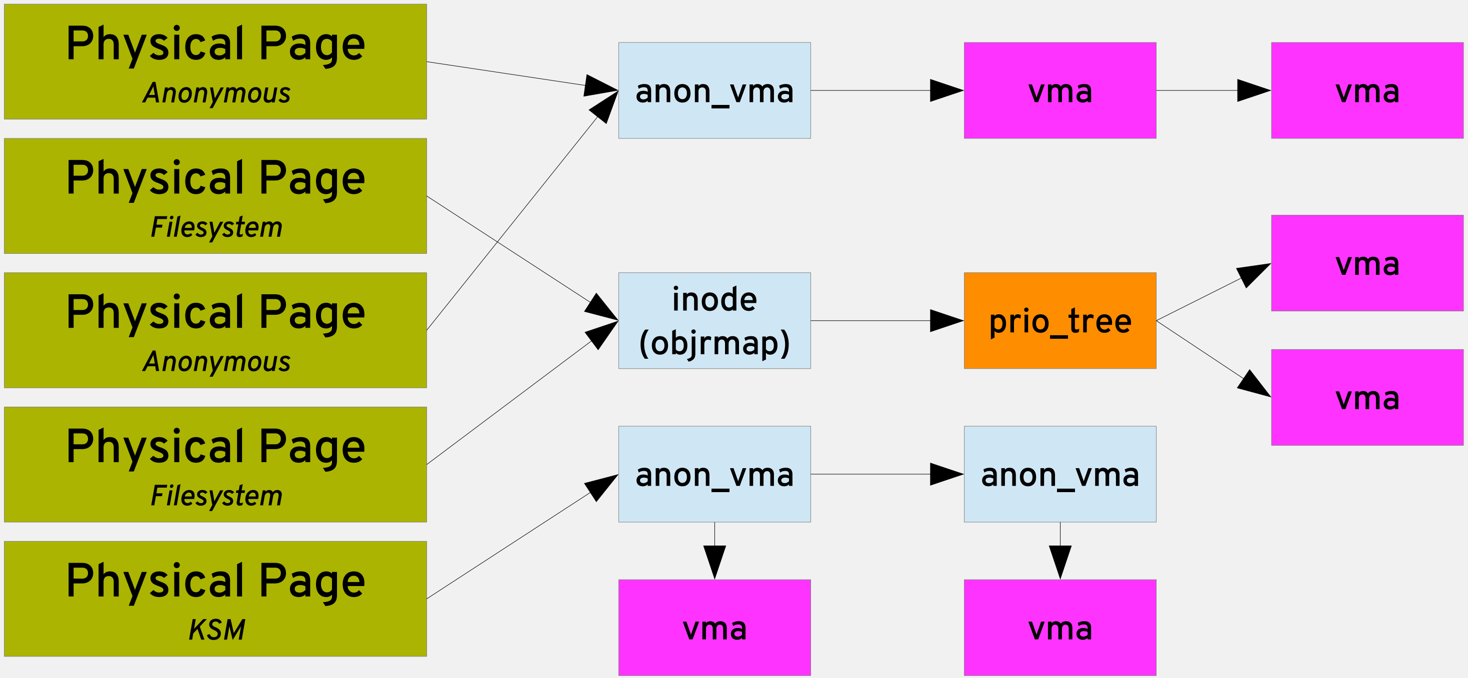
rmap after unmap event

Click to add subtitle



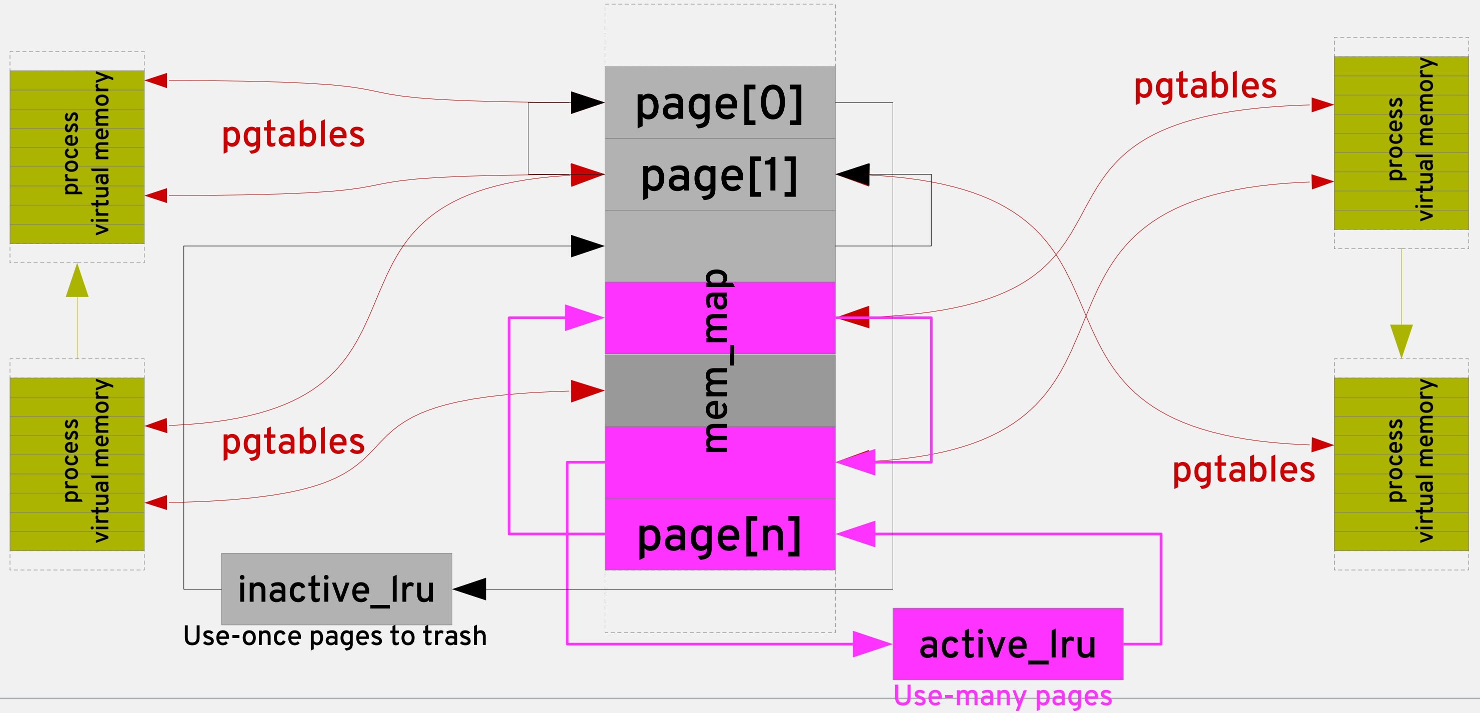
If the user space program access the page, it will trigger a pagein/swapin

objrmap / anon-vma / ksm



Active & Inactive + rmap

Kernel 2.6 Late 2000's



/proc/meminfo

```
MemTotal:        16054316 kB   SReclaimable:      644192 kB
MemFree:          1390476 kB   SUnreclaim:        218776 kB
MemAvailable:     2946740 kB   KernelStack:        23776 kB
Buffers:           112292 kB   PageTables:        147796 kB
Cached:           2652012 kB   NFS_Unstable:           0 kB
SwapCached:         39876 kB   Bounce:                 0 kB
Active:          11228856 kB   WritebackTmp:           0 kB
Inactive:         2087104 kB   CommitLimit:     16096272 kB
Active(anon):    10440148 kB   Committed_AS:    32734512 kB
Inactive(anon):   1710336 kB   VmallocTotal:    34359738367 kB
Active(file):      788708 kB   VmallocUsed:       763212 kB
Inactive(file):    376768 kB   VmallocChunk:    34358783056 kB
Unevictable:         3188 kB   HardwareCorrupted:      0 kB
Mlocked:             3188 kB   AnonHugePages:    1280000 kB
SwapTotal:        8069116 kB   HugePages_Total:        0
SwapFree:         6731456 kB   HugePages_Free:         0
Dirty:              23076 kB   HugePages_Rsvd:         0
Writeback:              0 kB   HugePages_Surp:         0
AnonPages:       10531332 kB   Hugepagesize:        2048 kB
Mapped:            844892 kB   DirectMap4k:       305040 kB
Shmem:            1599248 kB   DirectMap2M:     15040512 kB
Slab:              862968 kB   DirectMap1G:      1048576 kB
```

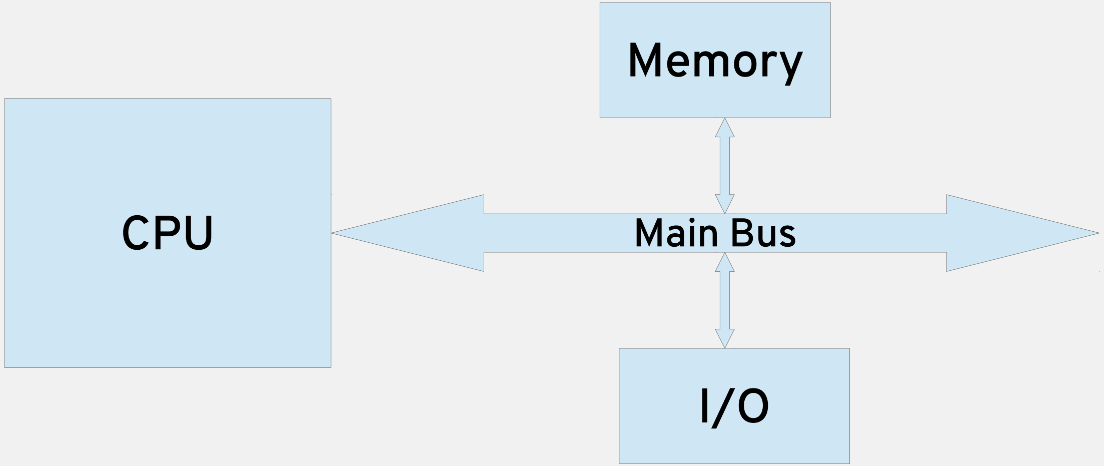
More recent changes: Many More LRUs

- Separated LRU for anon and file backed mappings
- memcg (memory cgroups) introduced per-memcg LRUs
- Removal of un-freeable pages from LRUs
  - anonymous memory with no swap
  - mlocked memory
- Transparent Hugepages in the LRU increase scalability further (lru size decreased 512 times)

CPU Memory Architectures

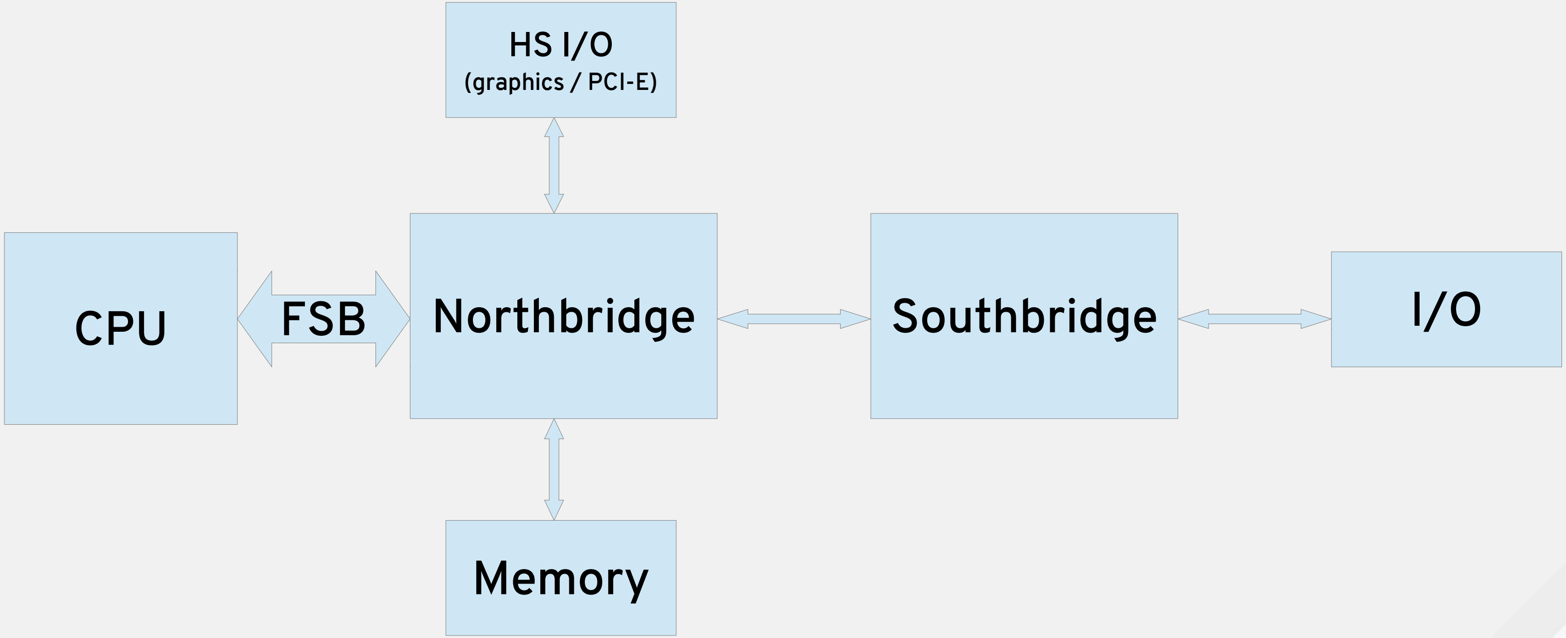
Older Architectures

Direct memory bus



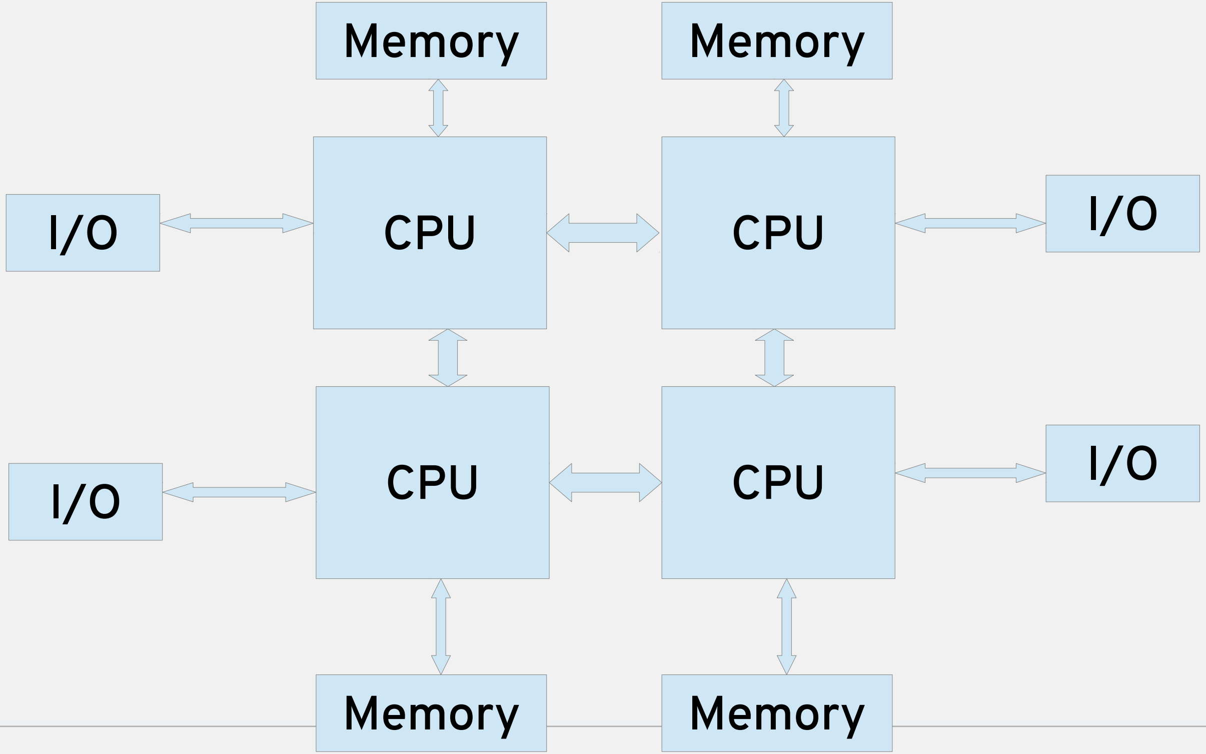
Older Architectures

Northbridge / Southbridge memory bus



NUMA Architecture

Multi-core Multi-Bus Architecture



NUMA – Non-Uniform Memory Architecture

- Multiple Nodes in a NUMA System
  - Each Node
    - CPU
    - Memory
    - PCI/Devices
- All Nodes are interconnected
  - Interconnects are slow
- Why NUMA?

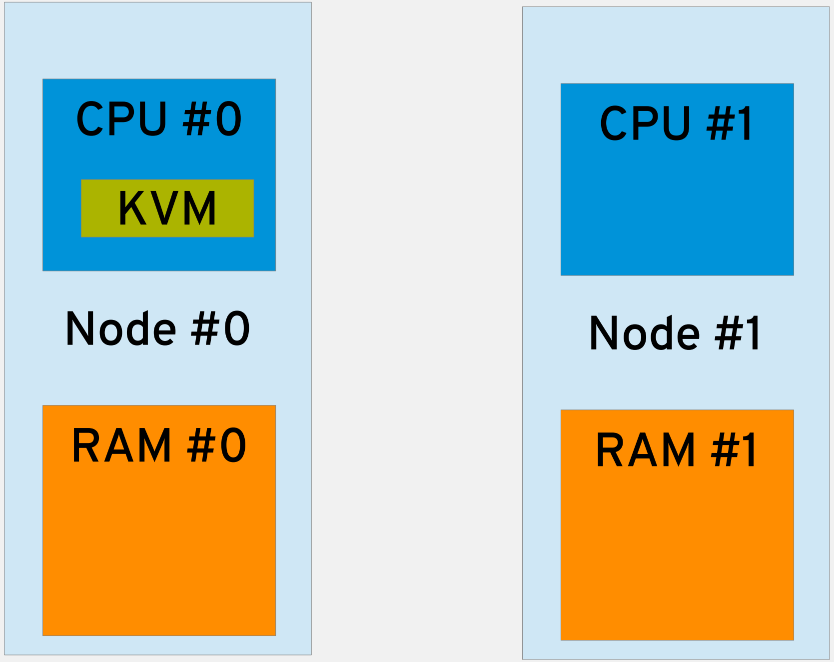
Linux and NUMA

NUMA memory policies

```
{
        MPOL_DEFAULT,    /* no numactl */
        MPOL_PREFERRED,  /* --preferred=node */
        MPOL_BIND,          /* default numactl /
        MPOL_INTERLEAVE, /* --interleave=nodes */
        MPOL_LOCAL,      /* --localalloc */
}
```

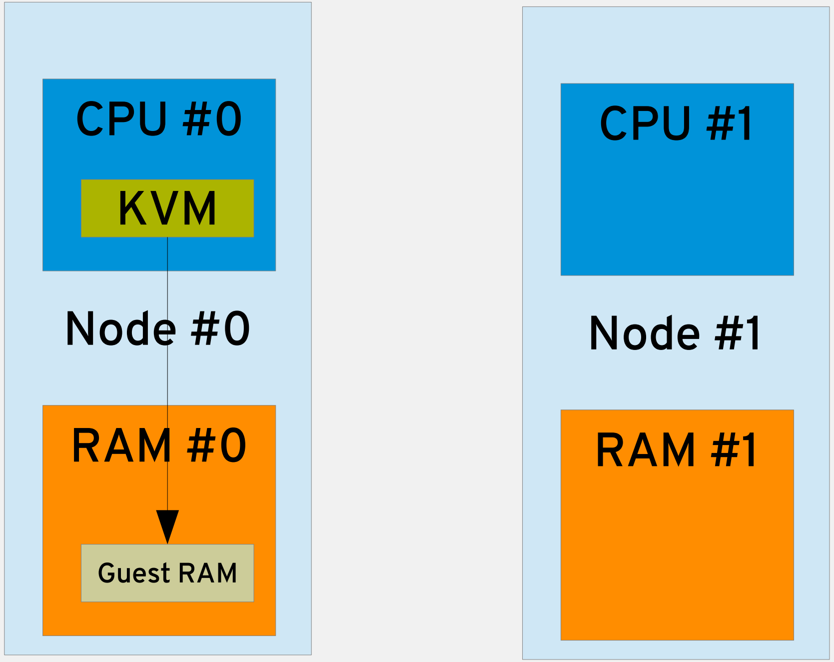
Virt startup on CPU #0

MPOL_DEFAULT



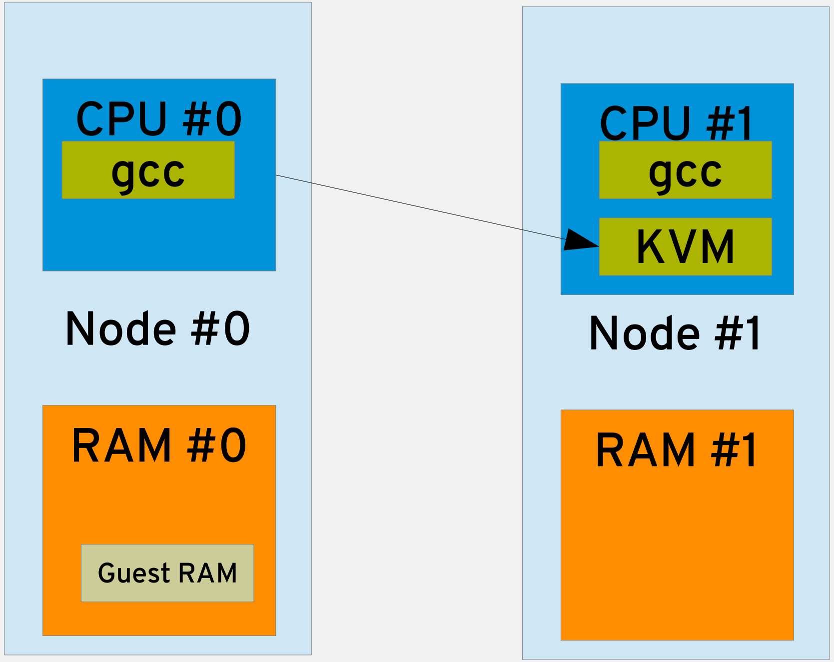
Virt Mach allocates from RAM #0

MPOL_DEFAULT, no bindings



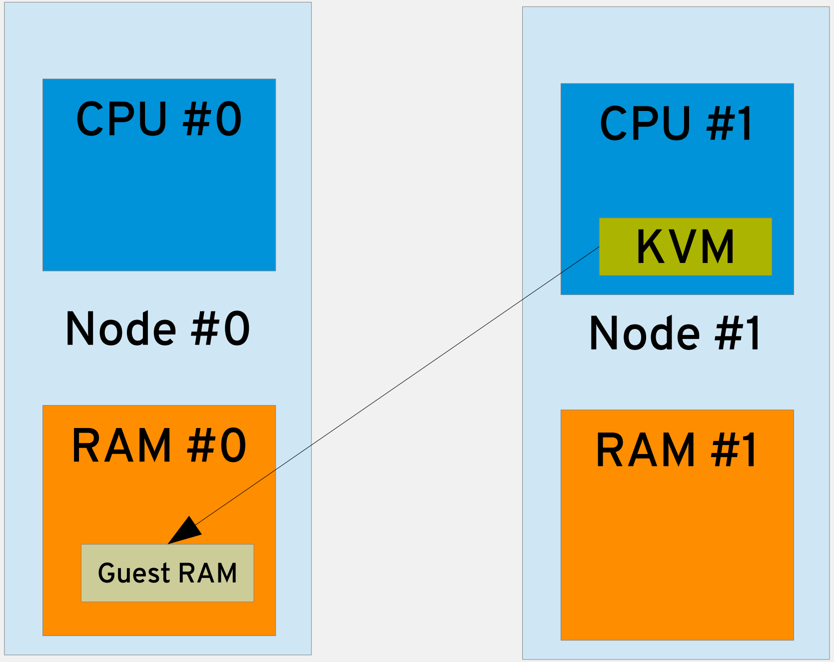
Scheduler CPU migration to #1

MPOL_DEFAULT, no bindings



Load goes away

Linux scheduler is blind – KVM will stay on CPU#1 with slow memory access



Hard NUMA bindings

- man numactl
- man numastat
- Kernel API
  - sys_mempolicy
  - sys_mbind
  - sys_sched_setaffinity
  - sys_move_pages
  - /dev/cpuset
- Full topology available in /sys

Numad can use the kernel API to monitor memory pressure and act accordingly

No NUMA affinity

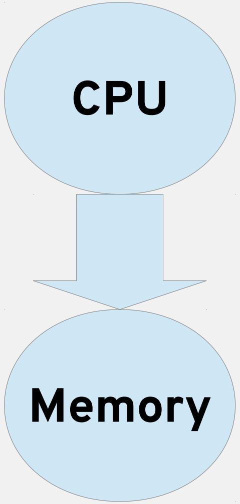
```
# numastat -c qemu-kvm
Per-node process memory usage (in Mbs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51722 (qemu-kvm)     68     16    357   6936      2      3    147    598  8128
51747 (qemu-kvm)    245     11      5     18   5172   2532      1     92  8076
53736 (qemu-kvm)     62    432   1661    506   4851    136     22    445  8116
53773 (qemu-kvm)   1393      3      1      2     12      0      0   6702  8114
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total              1769    463   2024   7462  10037   2672    169   7837 32434
```

NUMA affinity

```
# numastat -c qemu-kvm
Per-node process memory usage (in Mbs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------- ------ ------ ------ ------ ------ ------ ------ ------ -----
51722 (qemu-kvm)      0      0      7      0   8072      0      1      0  8080
51747 (qemu-kvm)      0      0      7      0      0      0   8113      0  8120
53736 (qemu-kvm)      0      0      7      0      0      0      1   8110  8118
53773 (qemu-kvm)      0      0   8050      0      0      0      0      0  8051
---------------- ------ ------ ------ ------ ------ ------ ------ ------ -----
Total                 0      0   8072      0   8072      0   8114   8110 32368
```

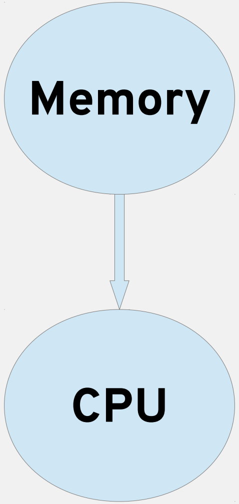
Automatic NUMA balancing

Introduces “gravity” between CPU and memory



Memory attracts CPU AGGRESSIVELY

- If idle load balancing permits

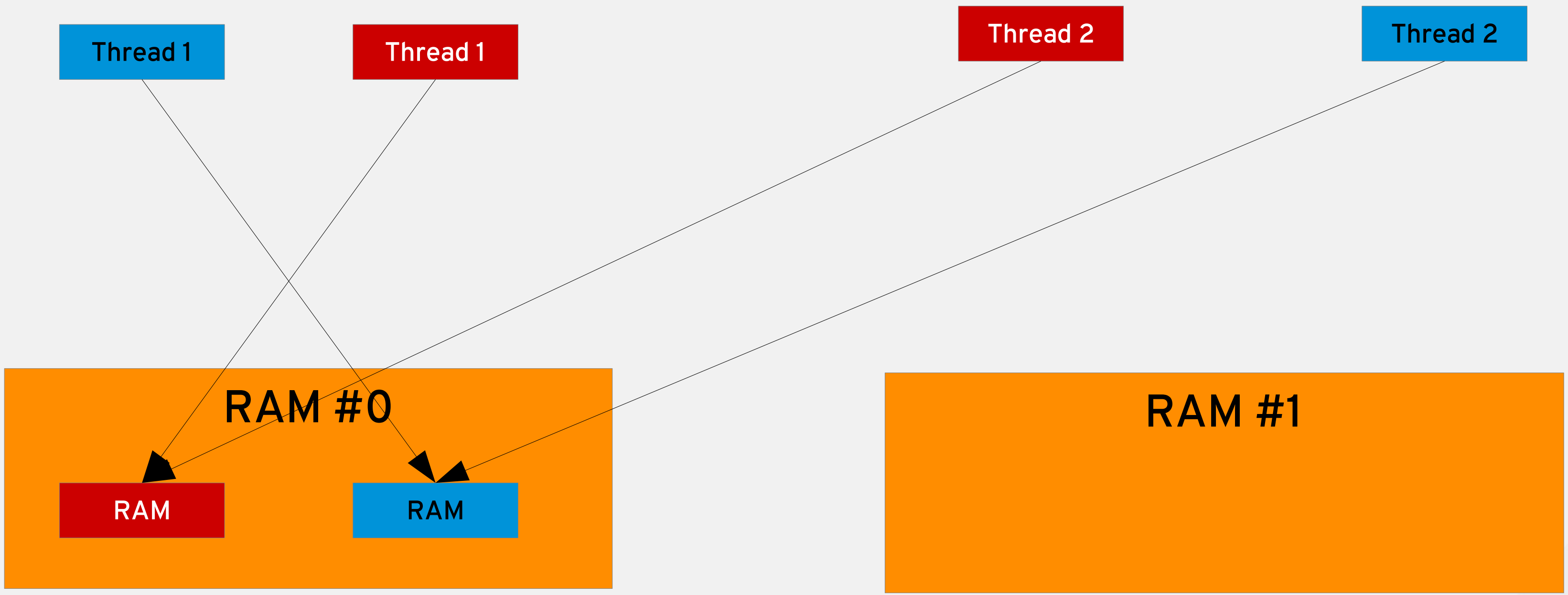


CPU attracts memory slowly

- If anti-false sharing permits

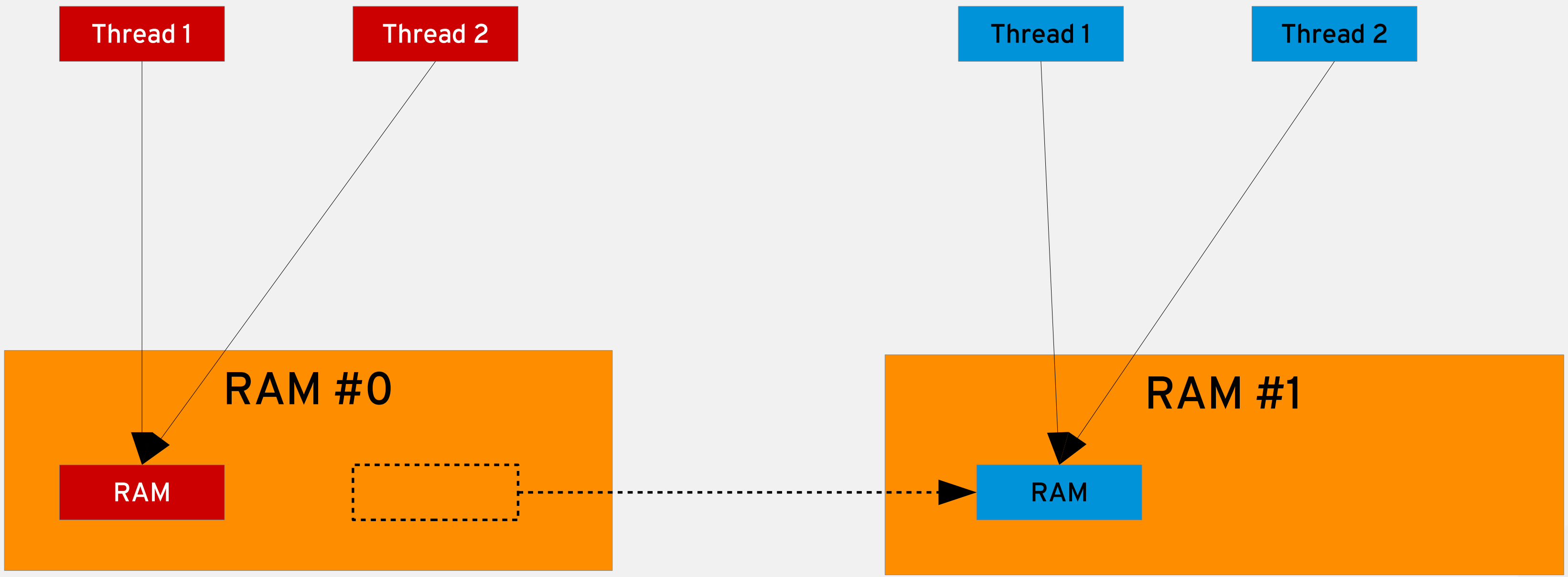
NUMA example

Startup state



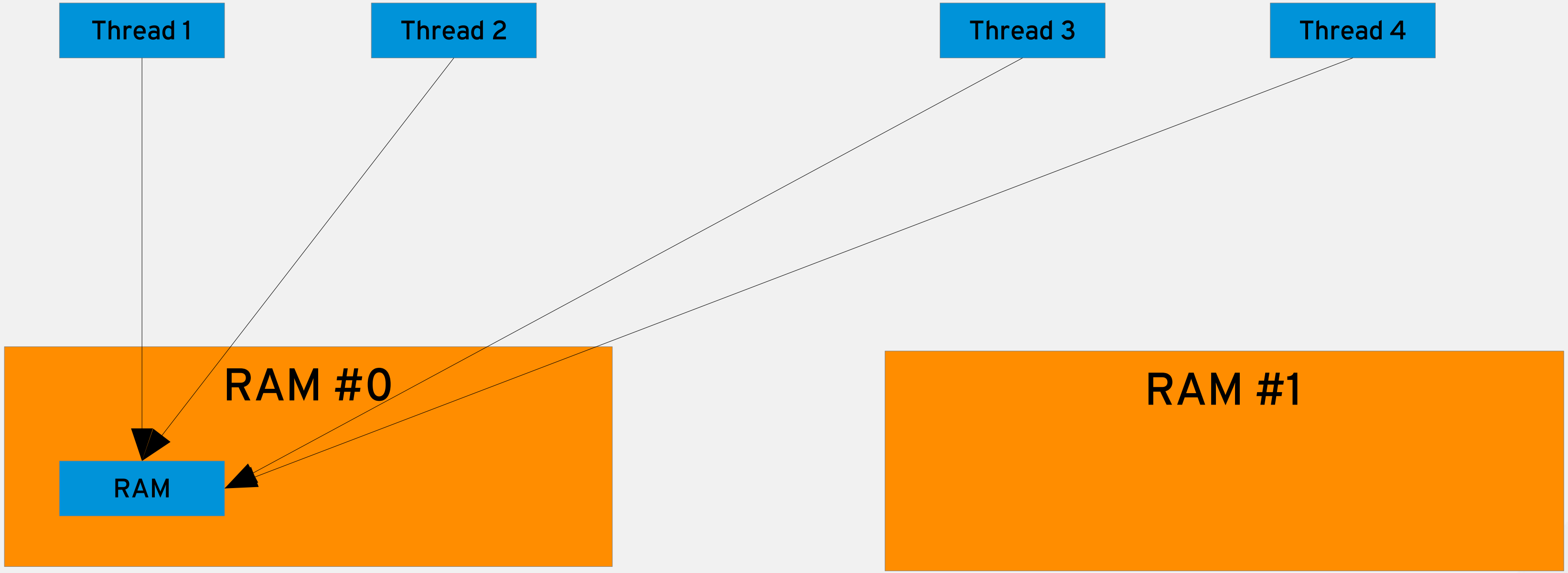
NUMA example

Converged state



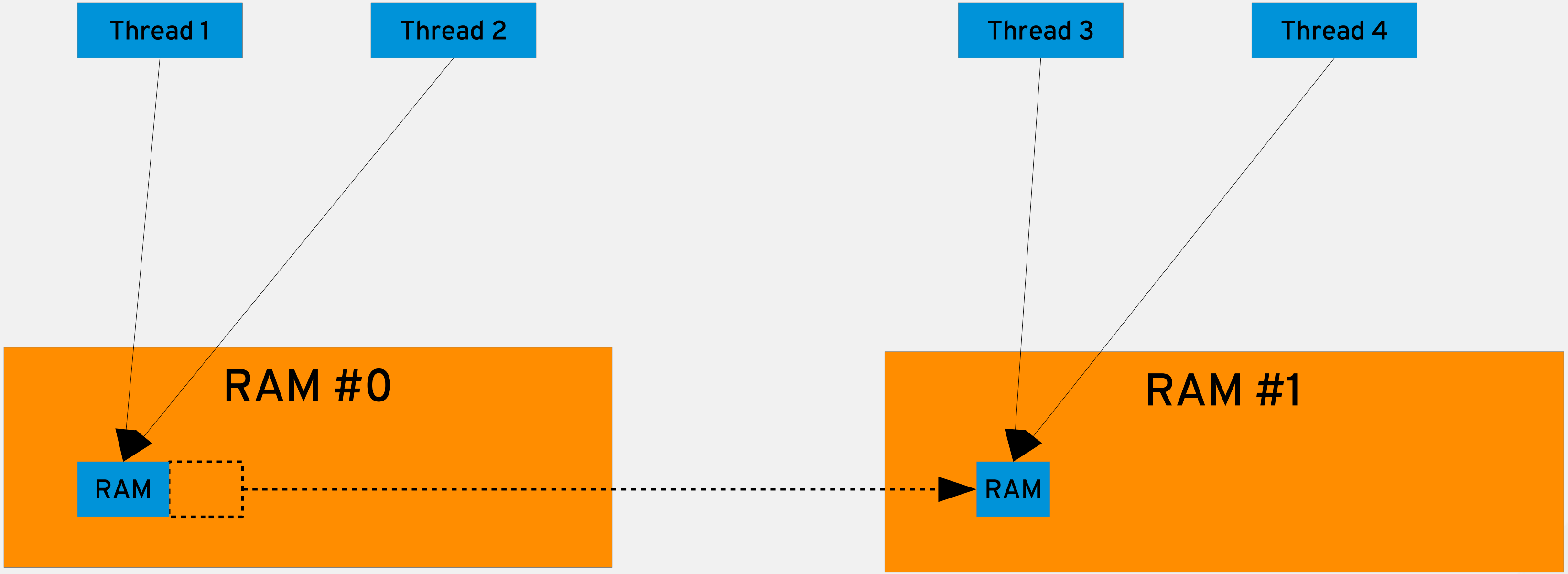
NUMA threading example

Startup state



NUMA threading example

Converged state

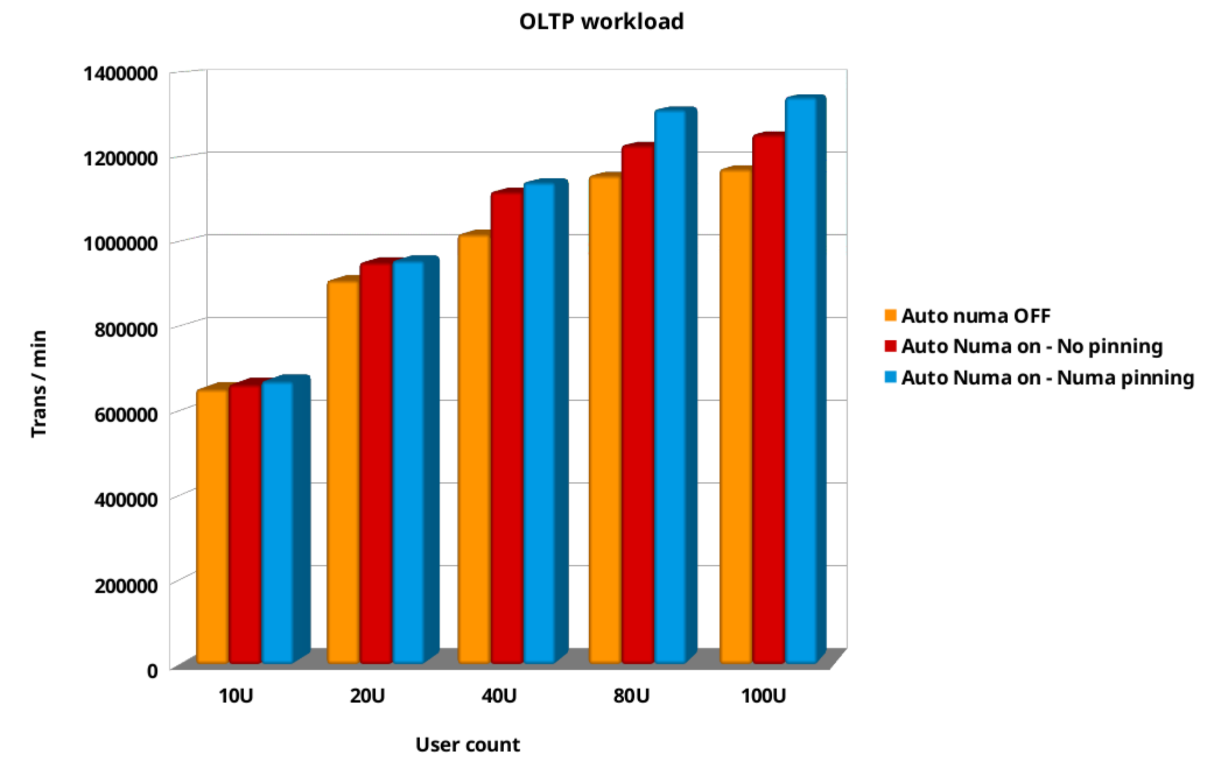


redhat.

Automatic NUMA balancing benchmark

- Intel SandyBridge (Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz)
  2 Sockets – 32 Cores with Hyperthreads
  256G Memory
- Software:
  - RHEV 3.6
  - Host bare metal – 3.10.0-327.el7 (RHEL7.2)
  - VM guest – 3.10.0-324.el7 (RHEL7.2)
  - VM – 32P , 160G (Optimized for Server)
  - Oracle – 12C , 128G SGA
- Storage – Violin 6616 – 16G Fibre Channel
- Test – Running Oracle OLTP workload with increasing user count and measuring Trans / min for each run as a metric for comparison

4VMs with different NUMA options



Automatic NUMA Balancing Configuration

- In RHEL7 Automatic NUMA balancing is enabled when:

  `# numactl --hardware` shows multiple nodes

- To disable automatic NUMA balancing:

```
# echo 0 > /proc/sys/kernel/numa_balancing
```

- To enable automatic NUMA balancing:

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

- At boot:

```
numa_balancing=enable|disable
```

Automatic NUMA balancing benchmark

- Intel SandyBridge (Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz)
  2 Sockets – 32 Cores with Hyperthreads
  256G Memory
- Software:
  - RHEV 3.6
  - Host bare metal – 3.10.0-327.el7 (RHEL7.2)
  - VM guest – 3.10.0-324.el7 (RHEL7.2)
  - VM – 32P , 160G (Optimized for Server)
  - Oracle – 12C , 128G SGA
- Storage – Violin 6616 – 16G Fibre Channel
- Test – Running Oracle OLTP workload with increasing user count and measuring Trans / min for each run as a metric for comparison

HugePages

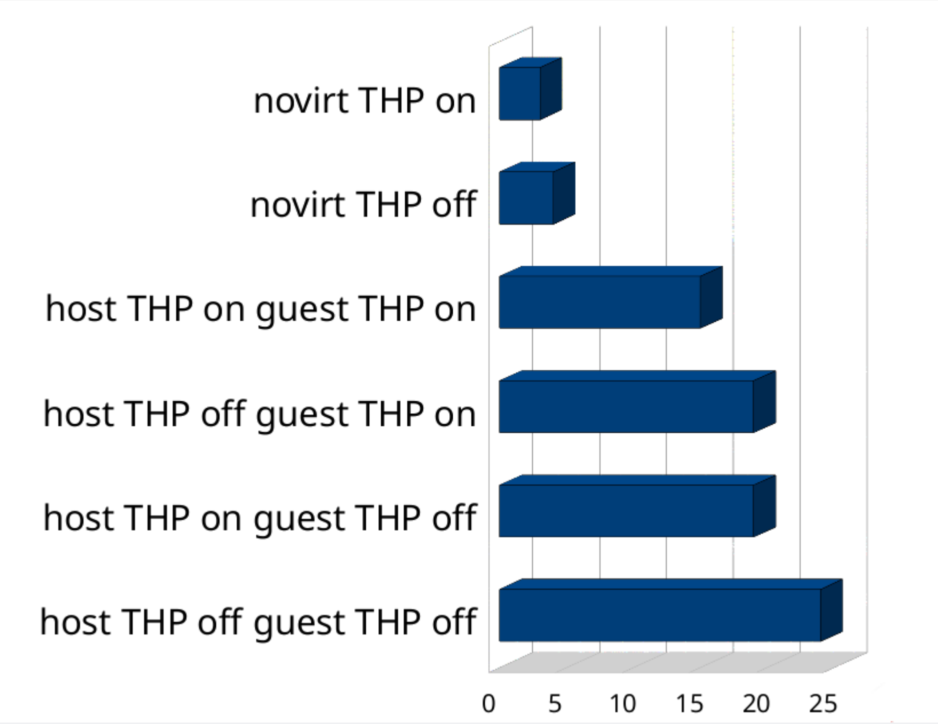
HugePages

- Traditionally x86 hardware gave us 4KiB pages
- The more memory the bigger the overhead in managing 4KiB pages
- What if you had bigger pages?
    - 512 times bigger → 2MiB

Why HugePages?

- Improve CPU performance
  - Enlarge TLB size
  - Speed up TLB miss
    - Need 3 accesses to memory instead of 4 to refill the TLB
  - Faster to allocate memory initially (minor)
  - Page colouring inside the hugepage (minor)
  - Higher scalability of the page LRUs
- Cons
  - clear_page/copy_page less cache friendly
  - higher memory footprint sometime
  - Direct compaction takes time

TLB Miss cost: # of memory accesses



Conclusion & Questions

Recent Trends

- Large Memory Usage processes → HugePages (4KB → 2MB)
- Programs or Virtual Machines duplicating Memory → KSM
- Optimization of workloads for you, without manual tuning
  - Automatic NUMA balancing
  - Transparent HugePages
- Page pinning → MMU notifier
- Direct Managed private device memory (i.e. GPU) → UVM (unified virtual memory)
- 4th Layer of pagetables
- pagetables in high memory region

Slides are available at http://people.redhat.com/pladd/

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos