Corey: An Operating System For Many Cores

Silas Boyd-Wickizer° Haibo Chen‡ Rong Chen‡ Yandong Mao‡
Frans Kaashoek° Robert Morris° Aleksey Pesterev°
Lex Stein§ Ming Wu§
Yuehua Dai† Yang Zhang° Zheng Zhang§

°MIT
‡Fudan University
†Microsoft Research Asia
§Xi'an Jiaotong University

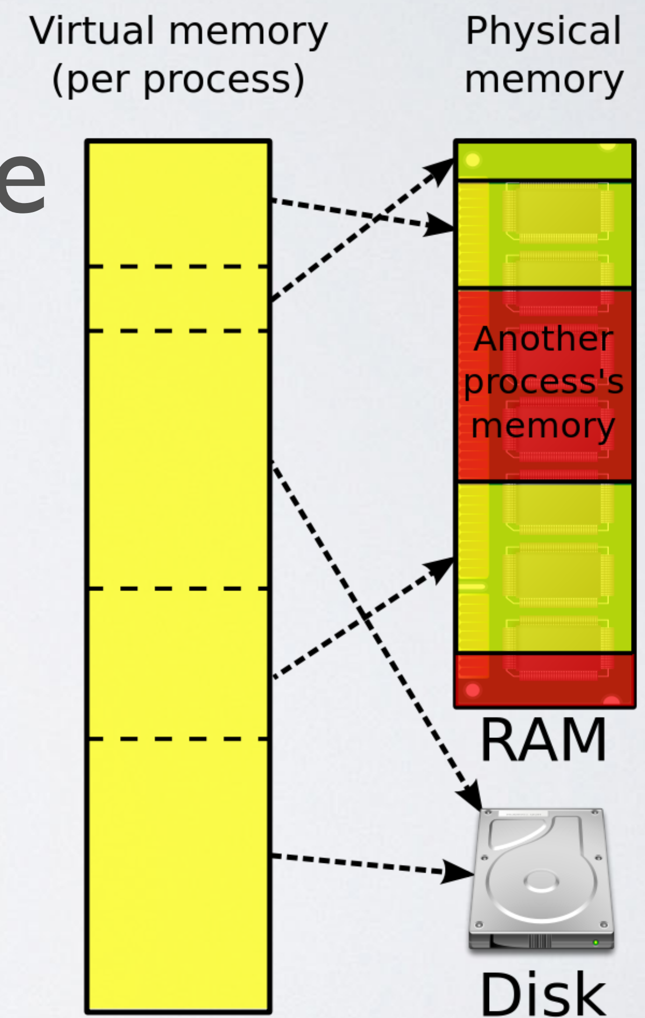
OSDI'08

Background, Motivation

- what is an operating system?
  - a layer between applications and hardware
  - resource/service provider
  - it tries to generalize possible conditions
- observation: as there are more cores, some systems suffer from unnecessary resource sharing
- applications know better what they need, return the power to the applications

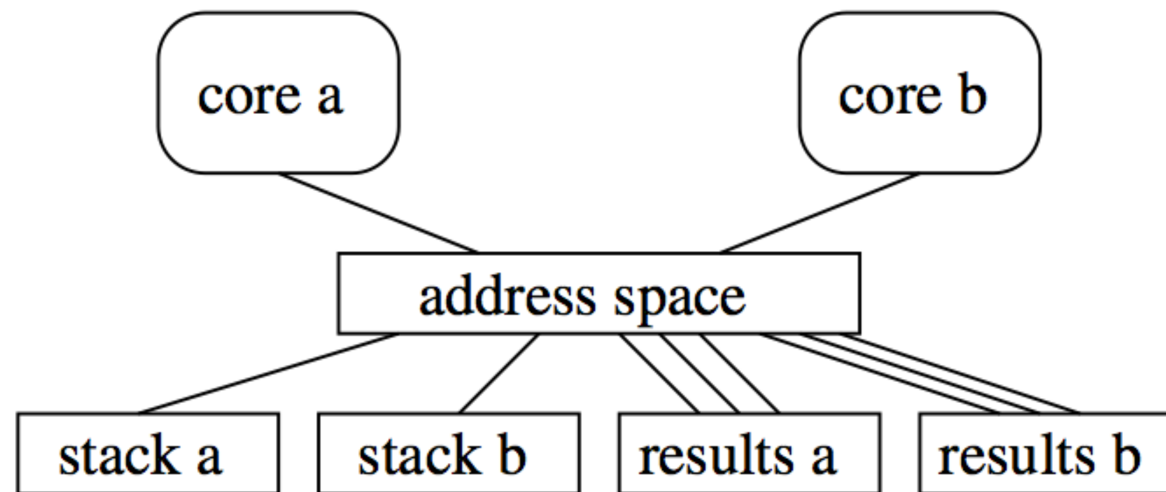
Virtual Address Space

- Each process sees memory address space as linear, but in fact it is not
- Each process has its own PageTable
- Kernel has its own PageTable in kernel space
- Translation Lookaside Buffer(TLB)

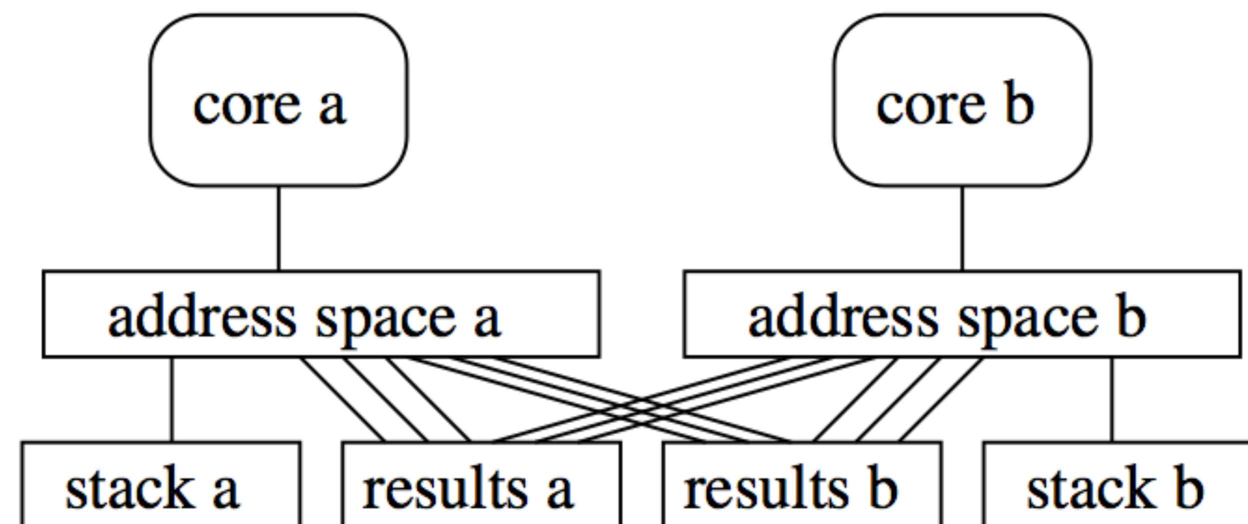


source http://en.wikipedia.org/wiki/File:Virtual_memory.svg

Traditional Address Space Management



(a) A single address space.



(b) Separate address spaces.

PROBLEM #1

file descriptor duplication

- unnecessary shared resource contention
  - **shared data structures**
  - locks: (cache miss cost)

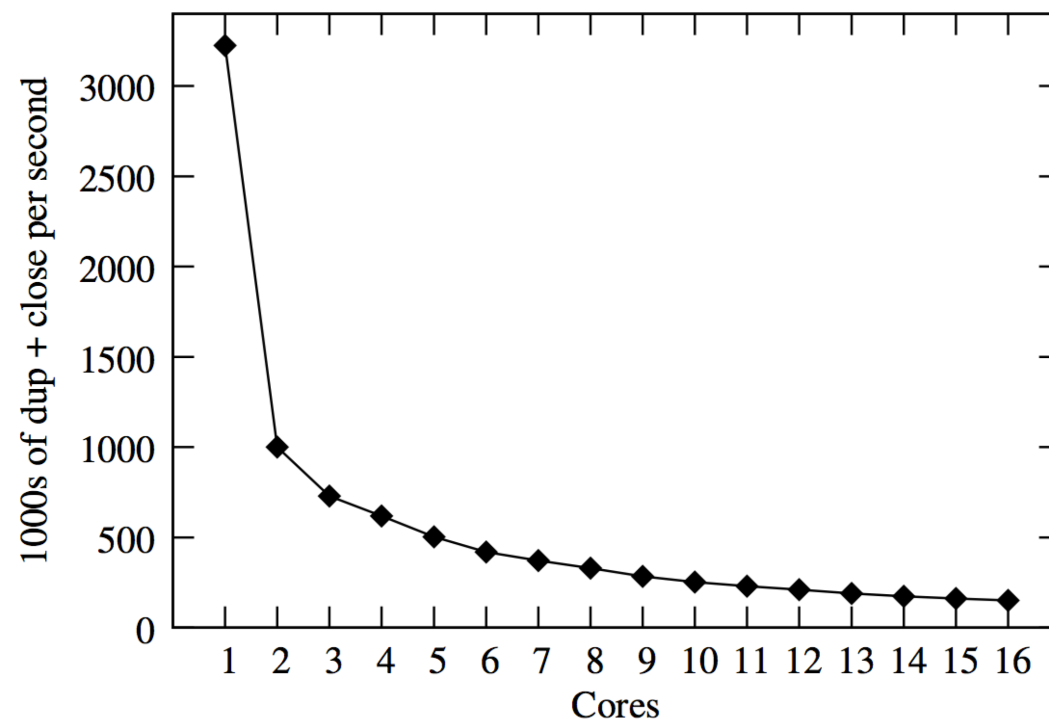


Figure 1: Throughput of the file descriptor `dup` and `close` microbenchmark on Linux.

main()

sys_dup()

rcu_read_lock()

fget()

spin_lock()

dupfd()

spin_lock()

fd_install()

close()

Loop, executed by many threads

sys_dup()

```
asmlinkage long sys_dup(unsigned int fildes)
{
    int ret = -EBADF;
    struct file * file = fget(fildes);

    if (file)
        ret = dupfd(file, 0, 0);
    return ret;
}
```

fget()

look up fd in fd_table

```
struct file *fget(unsigned int fd)
{
    struct file *file;
    struct files_struct *files = current->files;

    rcu_read_lock();
// internally it is implemented by a global mutex locked by a read_lock
    file = fcheck_files(files, fd);
    if (file) {
        if (!atomic_inc_not_zero(&file->f_count)) {
            /* File object ref couldn't be taken */
            rcu_read_unlock();
            return NULL;
        }
    }
    rcu_read_unlock();

    return file;
}
```

dupfd()

duplicate given file descriptor

```
static int dupfd(struct file *file, unsigned int start, int cloexec)
{
    struct files_struct * files = current->files;
    struct fdtable *fdt;
    int fd;

    spin_lock(&files->file_lock);
    fd = locate_fd(files, file, start);
    if (fd >= 0) {
        /* locate_fd() may have expanded fdtable, load the ptr */
        fdt = files_fdtable(files);
        FD_SET(fd, fdt->open_fds);
        if (cloexec)`
            FD_SET(fd, fdt->close_on_exec);
        else
            FD_CLR(fd, fdt->close_on_exec);
        spin_unlock(&files->file_lock);
        fd_install(fd, file);  // write to fd array
    } else {
        spin_unlock(&files->file_lock);
        fput(file);
    }

    return fd;
}
```

fd_install()

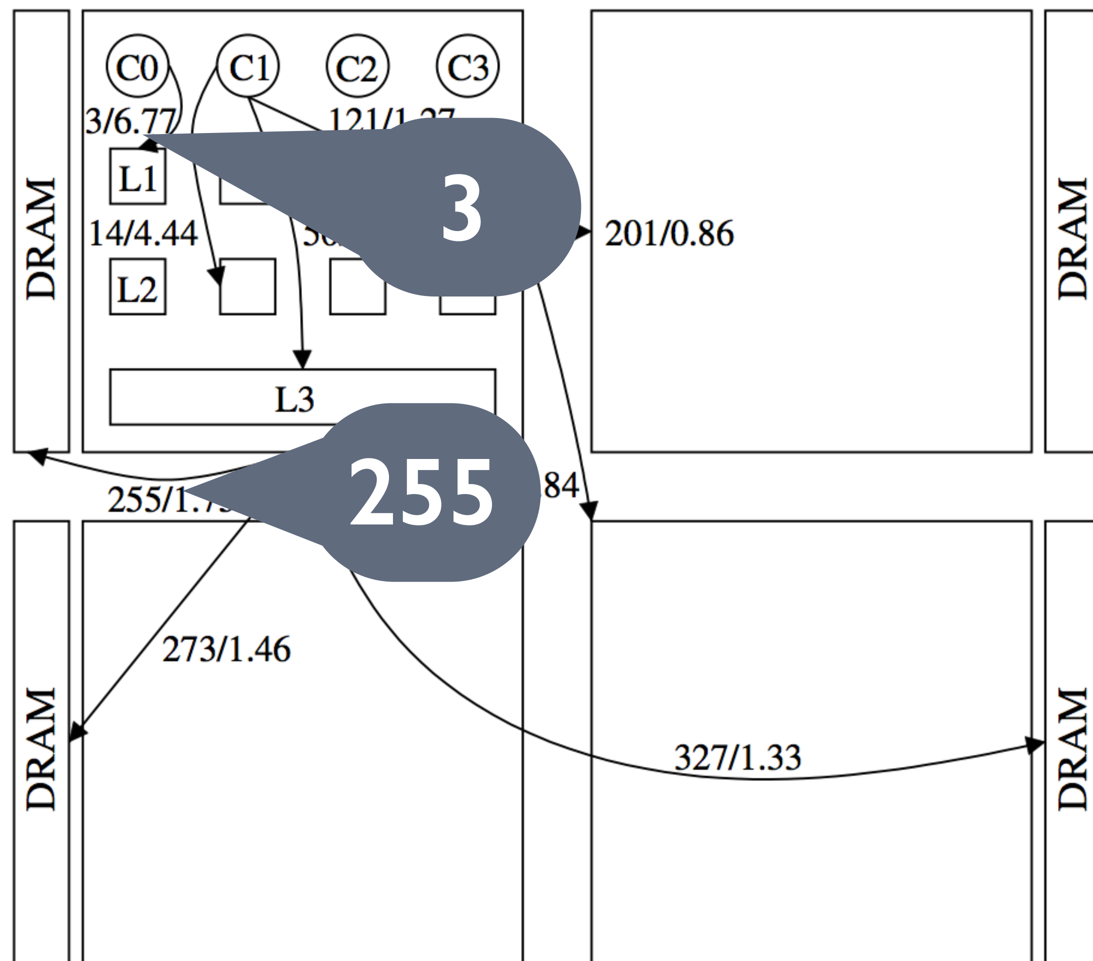
where things really went wrong

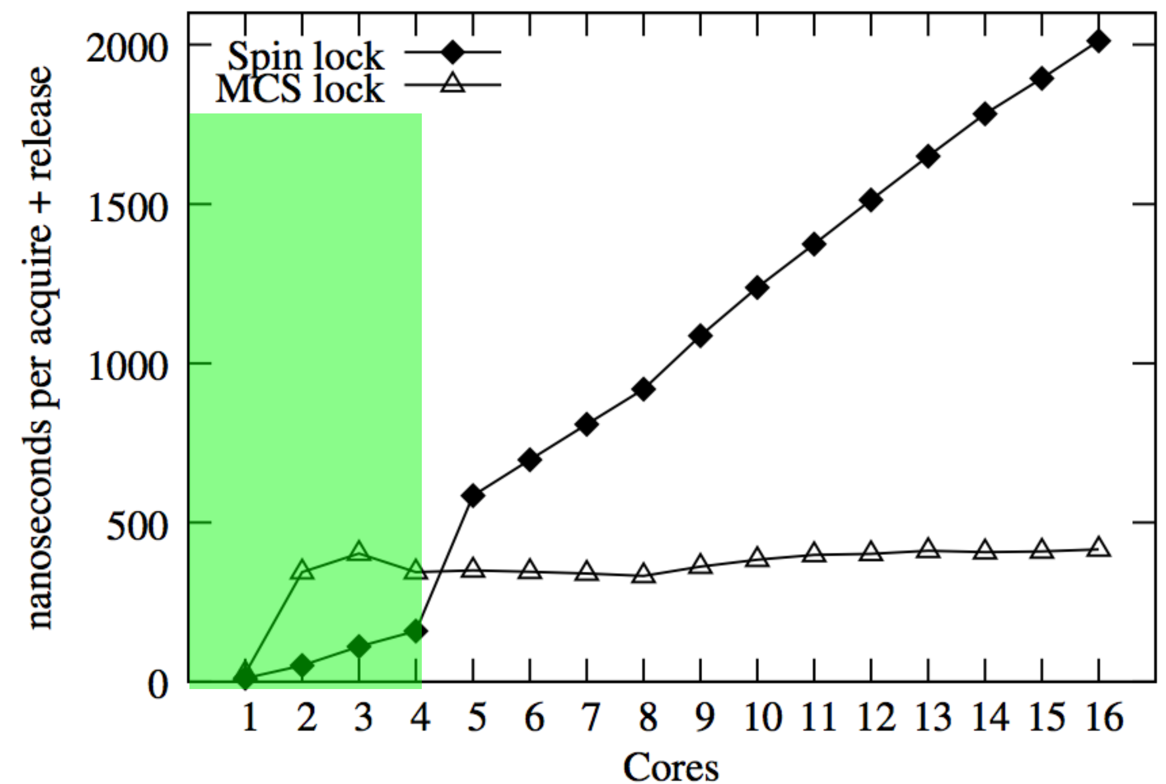
```
void fd_install(unsigned int fd, struct file *file)
{
    struct files_struct *files = current->files;
    struct fdtable *fdt;
    spin_lock(&files->file_lock);
    fdt = files_fdtable(files);
    BUG_ON(fdt->fd[fd] != NULL);
    rcu_assign_pointer(fdt->fd[fd], file);
    spin_unlock(&files->file_lock);
}

// recall that fd_install is called by every thread
```

PROBLEM #2

cache miss is expensive, ft. lock contention

- unnecessary shared resource contention
  - shared data structures
  - **locks: (cache miss cost)**





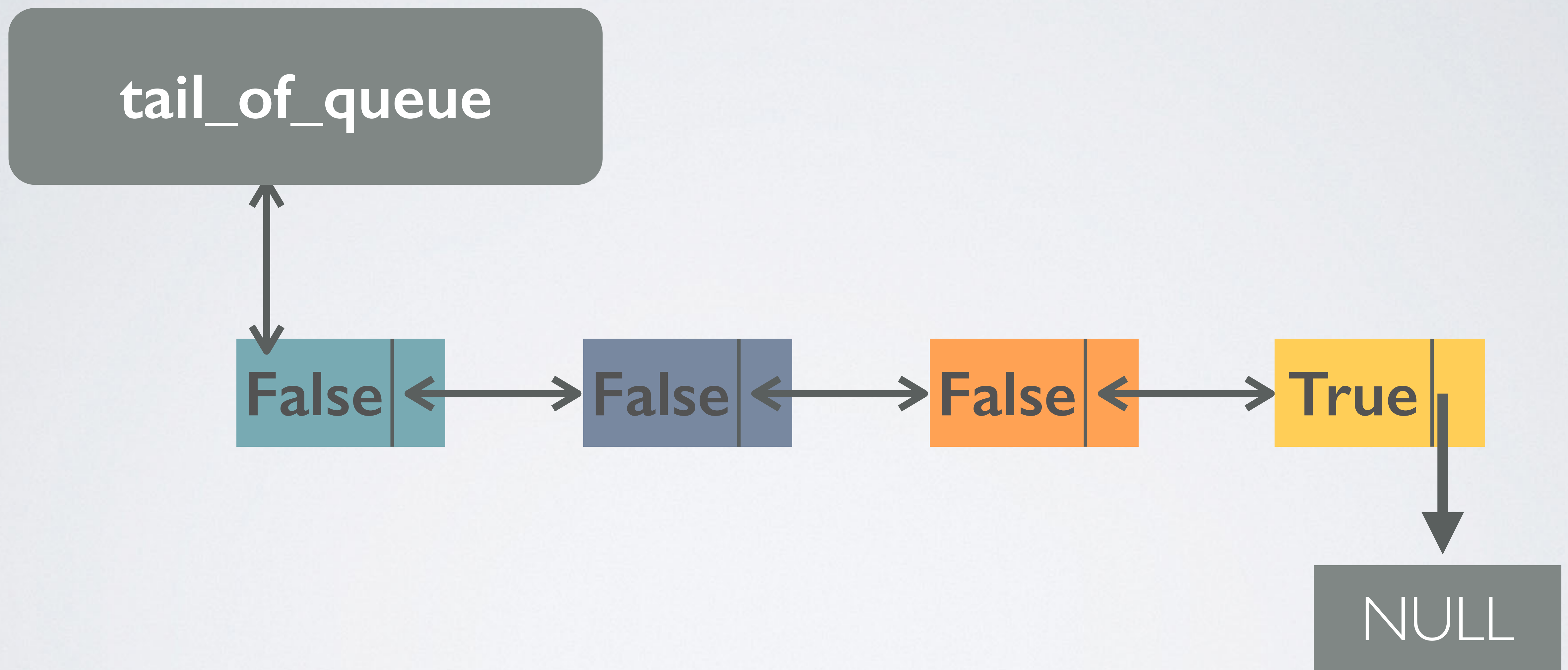
LOCKS

spin lock: spin on global variable, cache miss happens when a thread release or acquire a lock, high cache coherence traffic

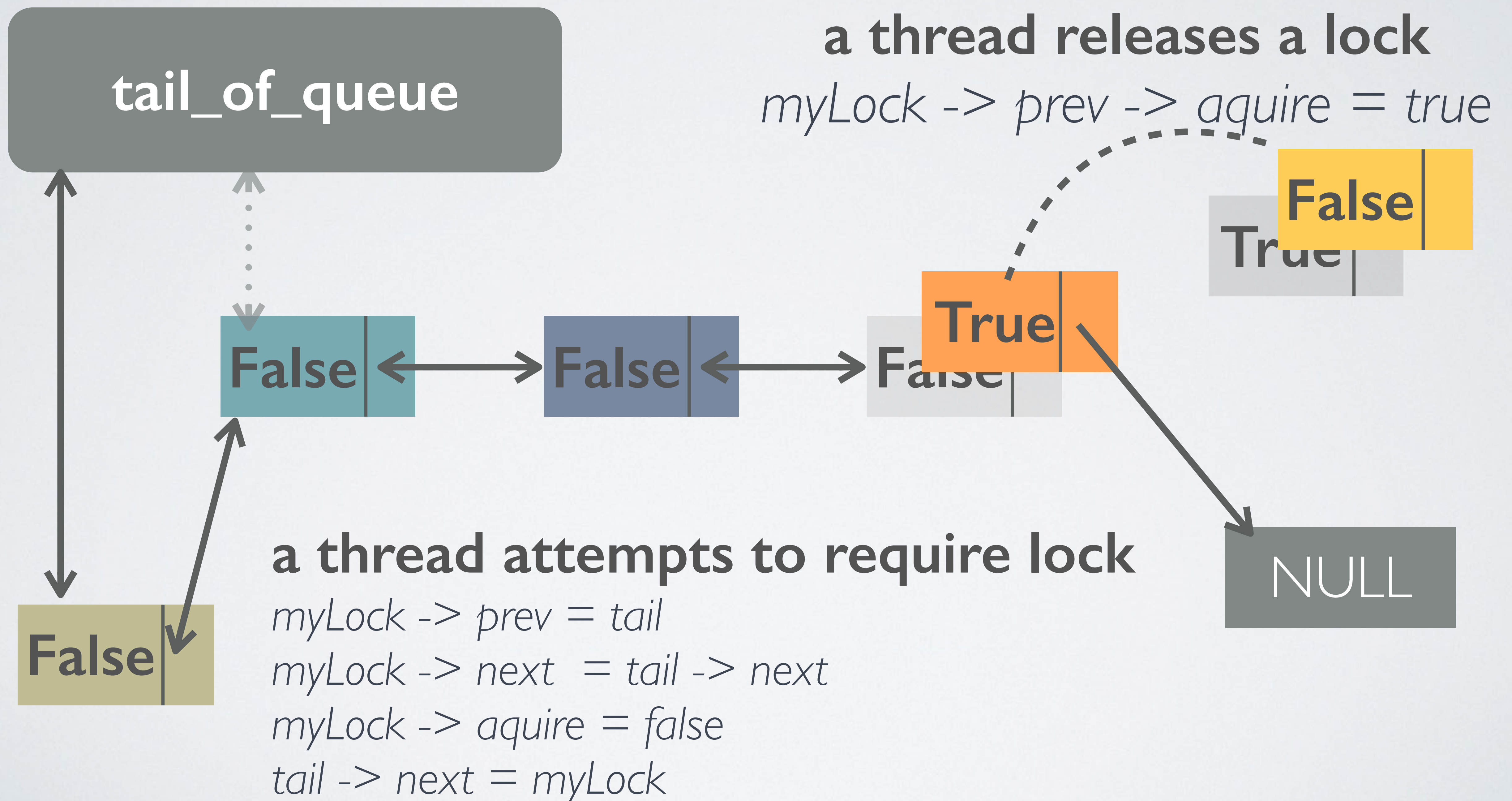
Test And Set (TAS) lock: spin on local variable, atomic, better than spin lock, but **no fairness guarantee**

MCS lock: spin on local variable, atomic, FIFO queue, when a thread release its own lock, it handles over the ownership to the next node in queue

MCS LOCK (QUEUE)



MCS LOCK (QUEUE)



Corey

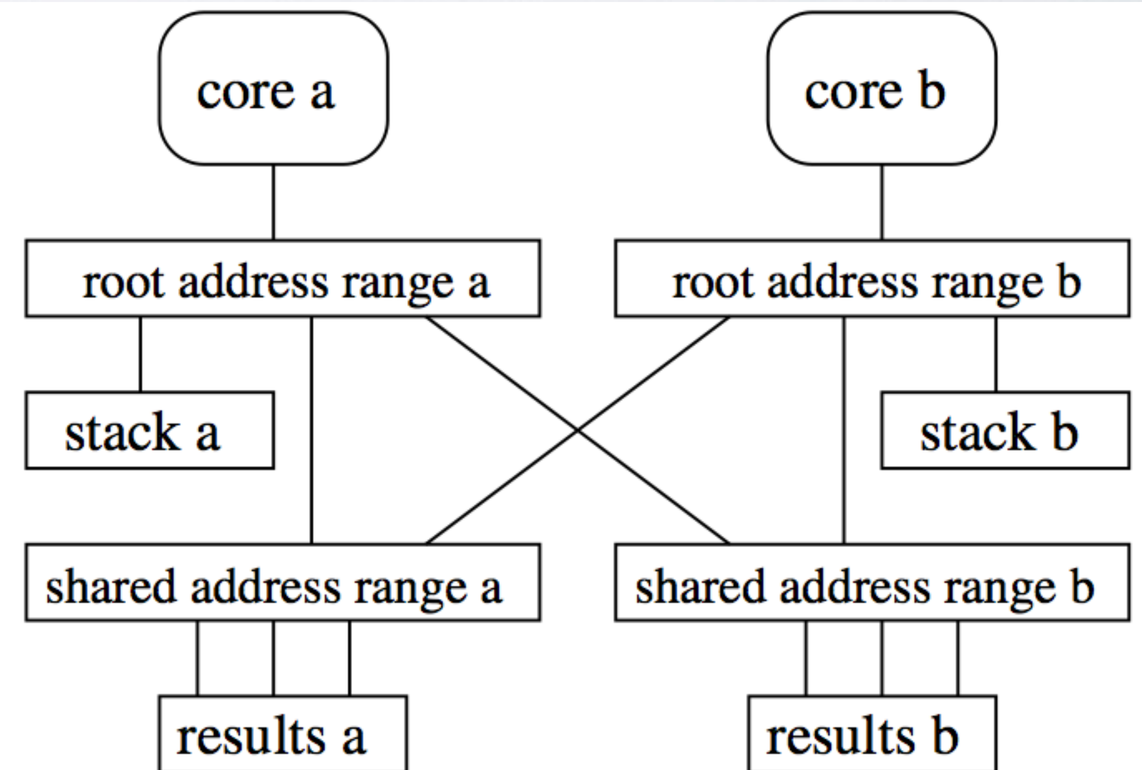
- Inspired by ExoKernel: protect but do not manage system resource

- 3 abstractions
  - Address range
  - kernel core
  - shares

Address Range

Why Not Both

- An abstraction that corresponds to a range of virtual-to- physical mappings.
- - private(default): only owner core is able to access
  - shared: assign by application
- avoid contention
- ar_alloc() to create

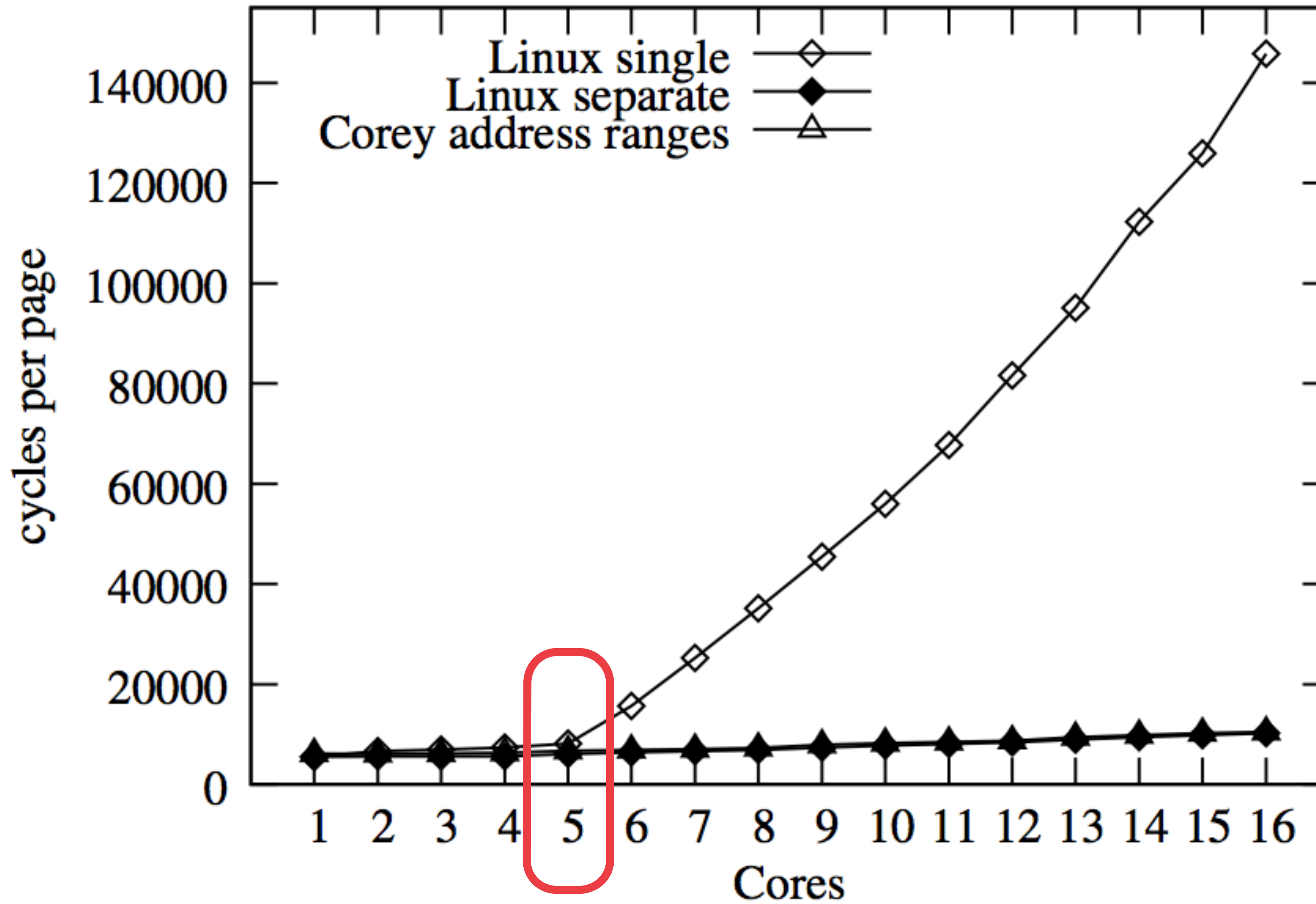


(c) Two address spaces with shared result mappings.

Address Range Evaluation

- private memory access: **memclone**
  - each core allocate 100MB on its own DRAM pool
  - use round-robin to allocate new core
- shared memory access: **mempass**
  - one core allocates 100MB
  - each core accesses every page

Memclone

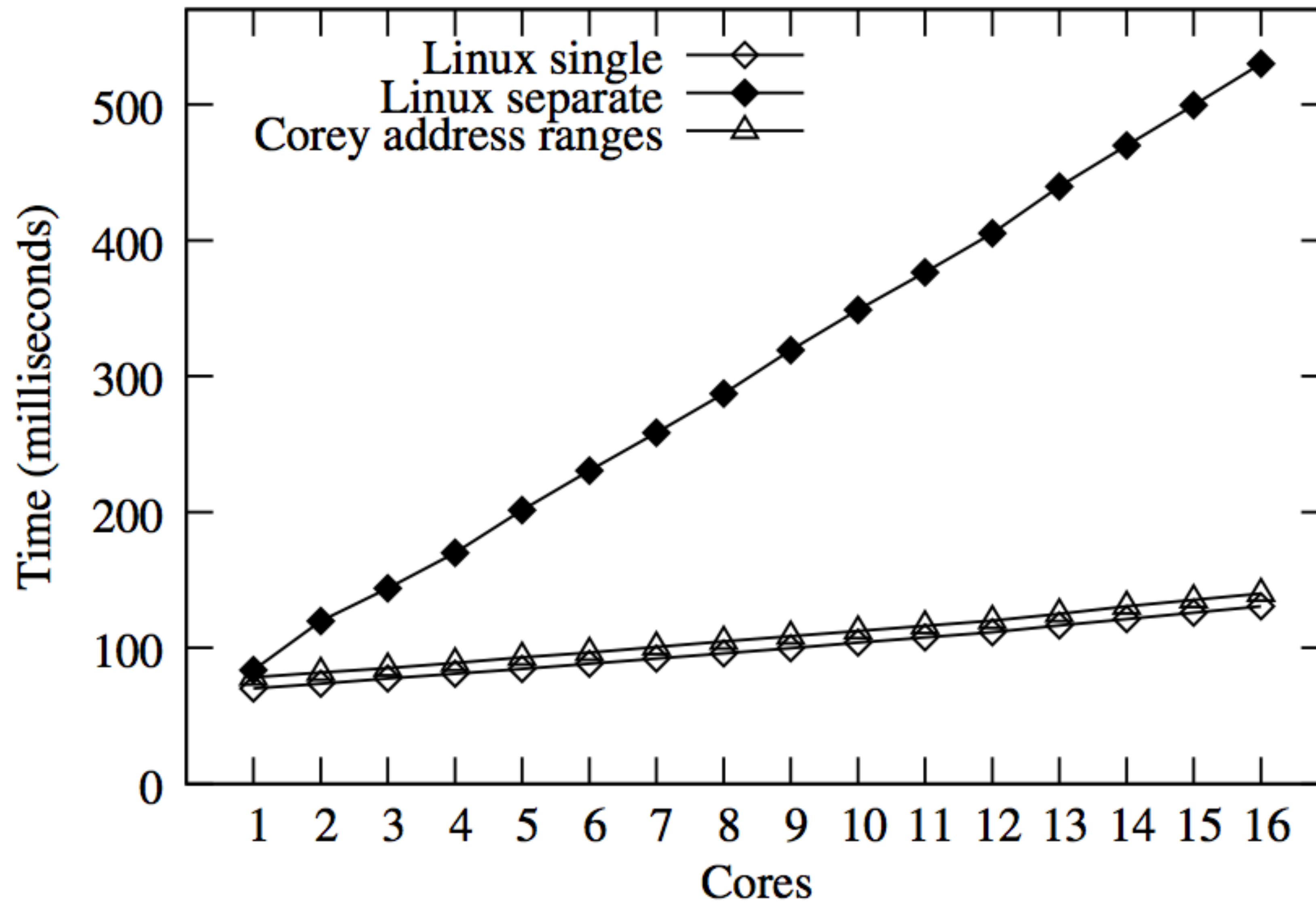


(a) memclone

Memclone

- Linux single memory
- shared memory access: **mempass**
  - one core allocates 100MB
  - each core accesses every page

Mempass



(b) `mempass`

Kernel Core

A Hint Of Resource Isolation

- an abstraction that specifies a core to kernel functions and data
- a kernel core can manage hardware devices and execute system calls
- among kernel cores, they communicate via IPC
- increase scalability by avoiding cache miss
- less TLB invalidation (TLB is cleared in every context switch)

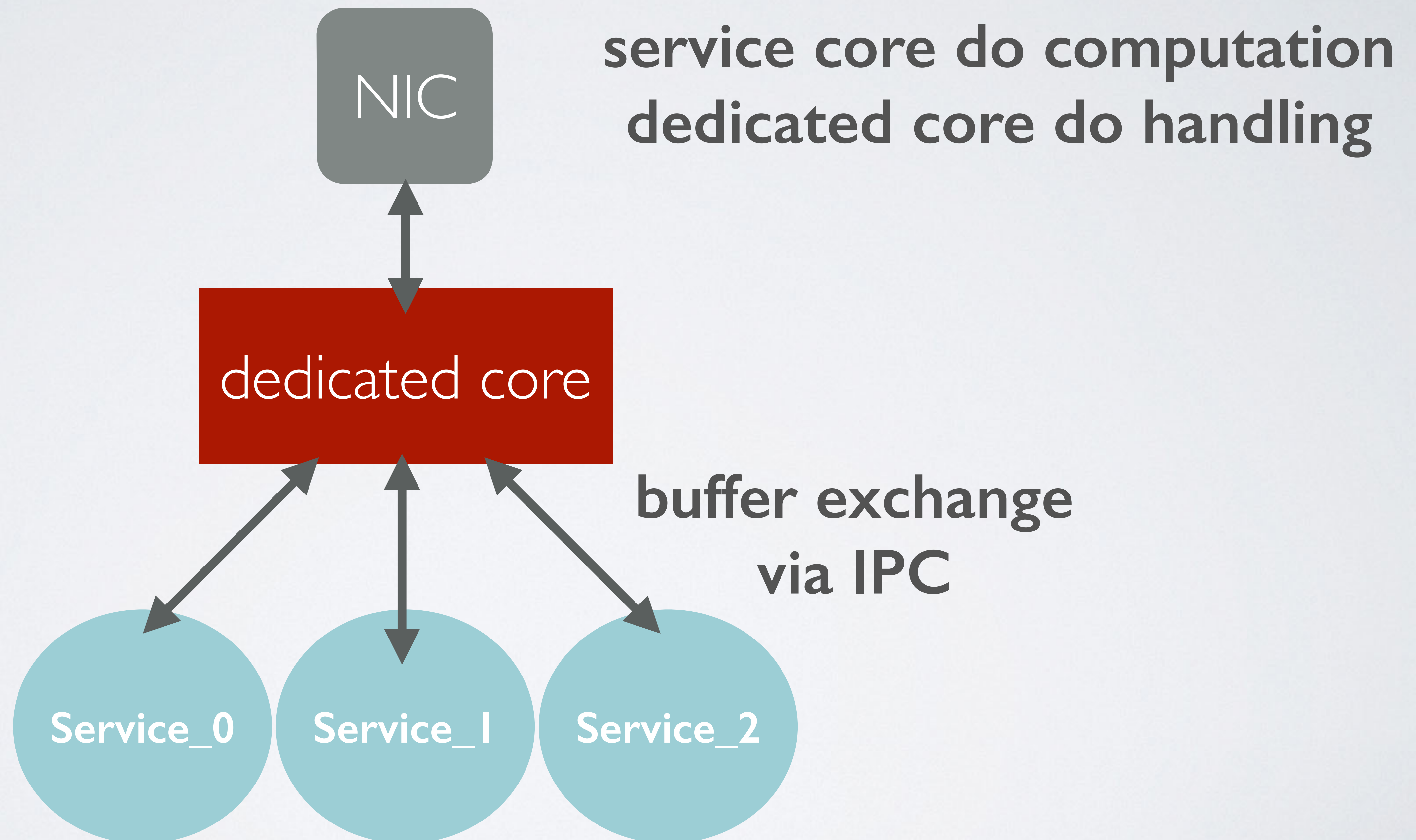
Kernel Core Evaluation

- simple TCP service that accepts and responds with a128 bytes message to each connection before closing it

- 2 modes
  - **dedicated**: one kernel core handles everything except for computation
  - **polling**: a kernel core only to poll for received packet notifications and transmit completion
- In both cases, each other core runs private TCP/IP service with private TCP/IP stack

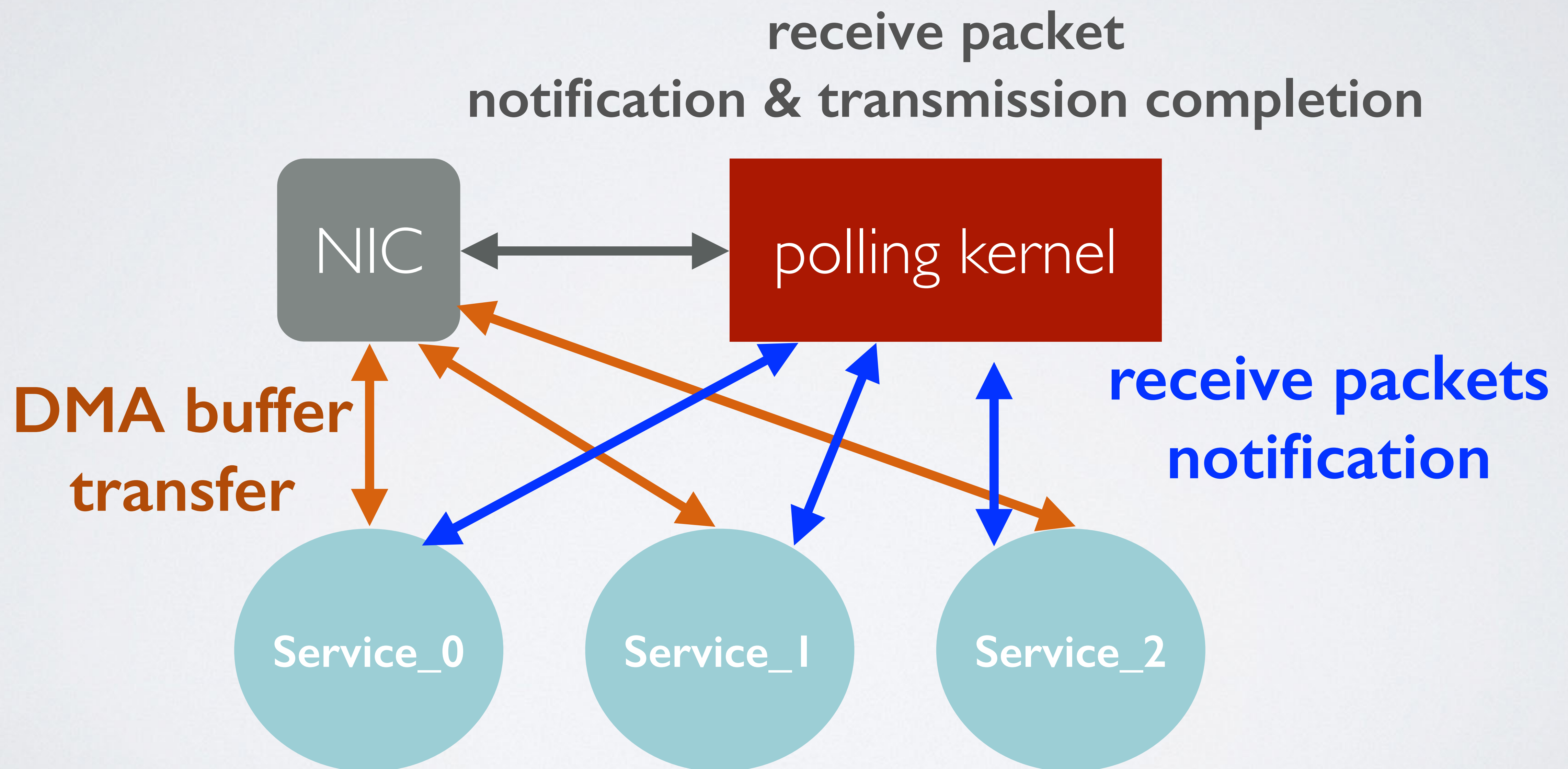
Kernel Core Evaluation

Dedicated Mode



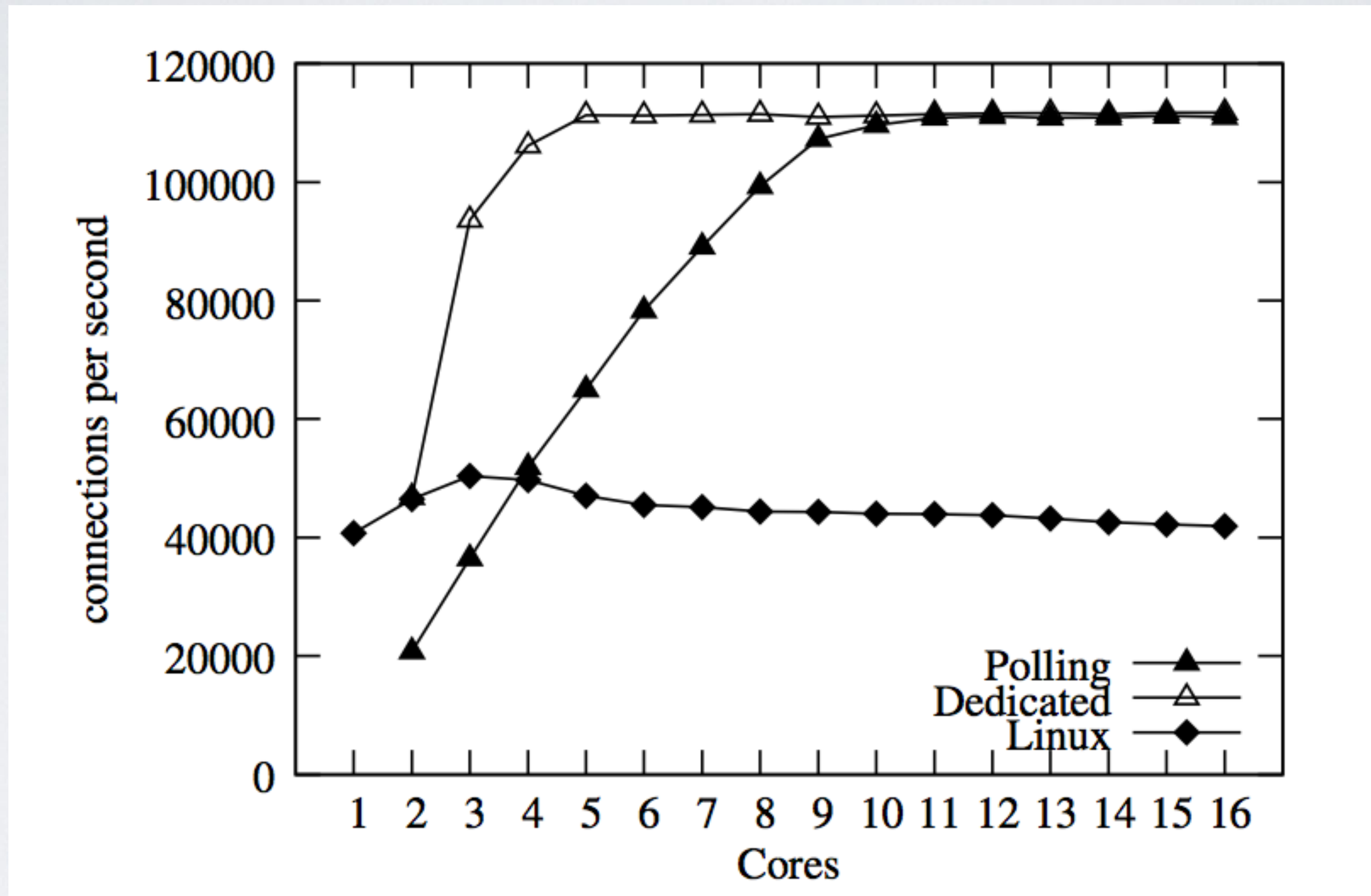
Kernel Core Evaluation

Polling Mode



Kernel Core Evaluation

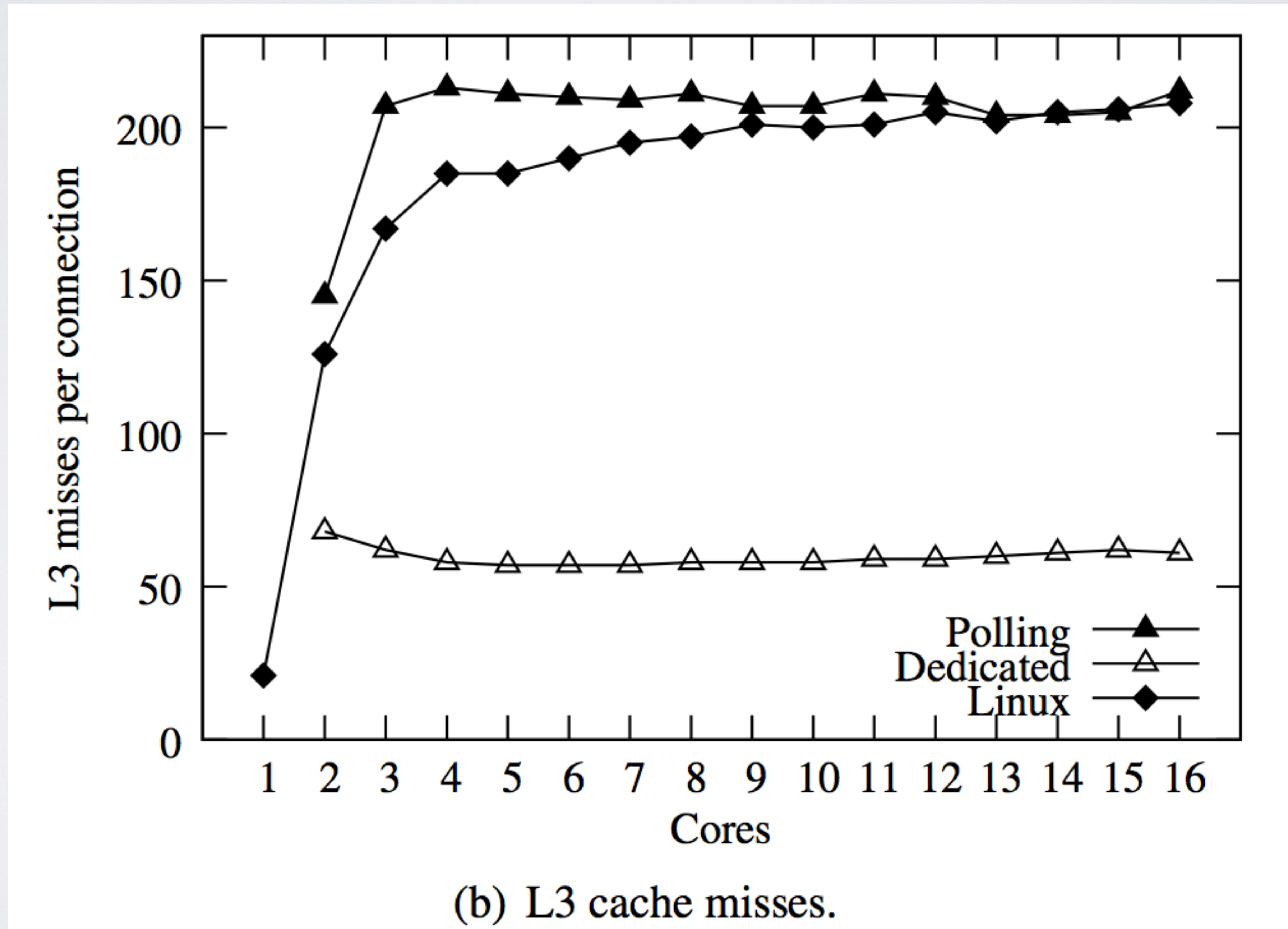
Throughput



each core is able to handle more connections per second in the Dedicated configuration than in Polling

Kernel Core Evaluation

L3 Cache Miss



(b) L3 cache misses.

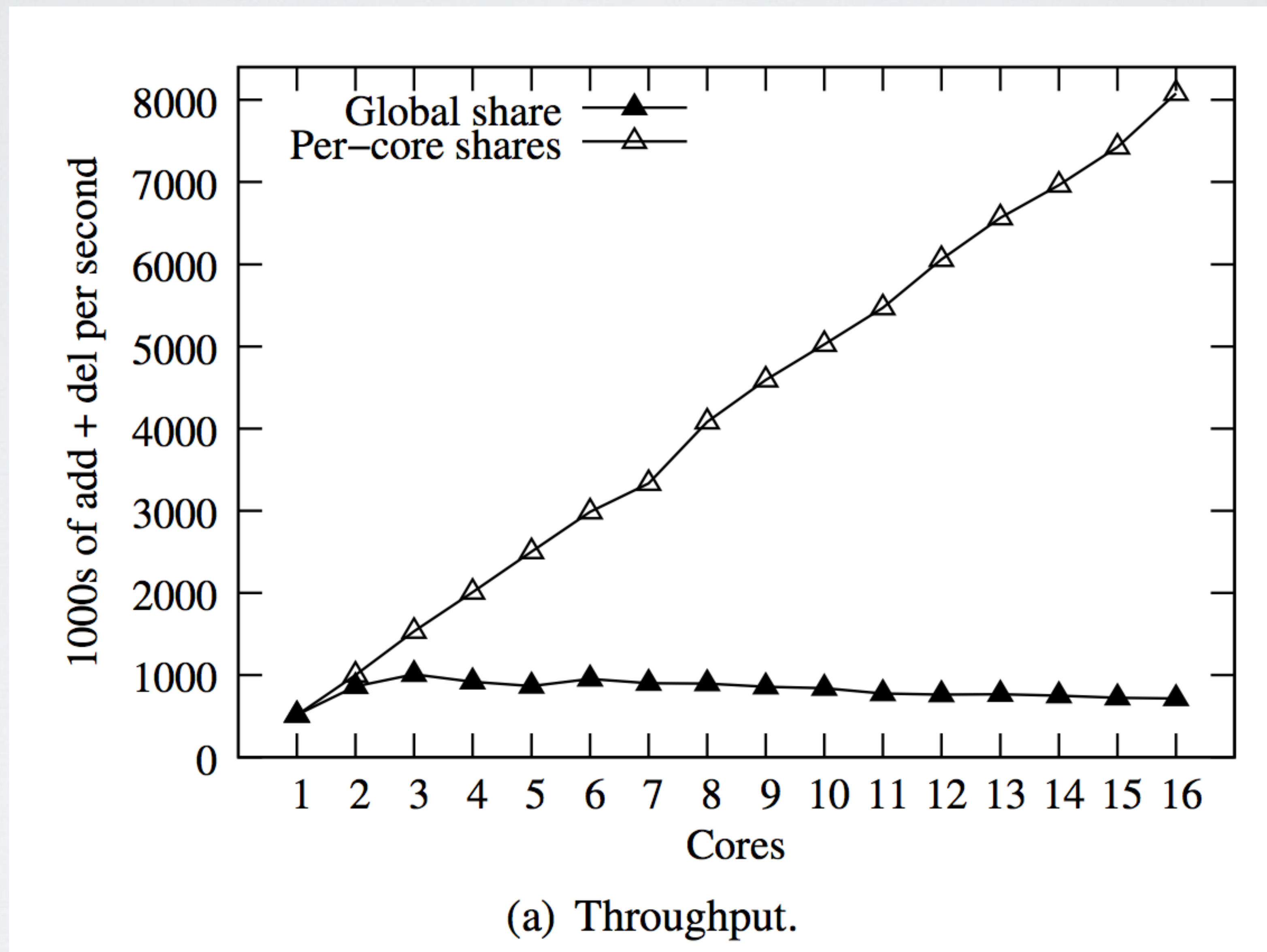
Shares

A Explicit Way Of Avoiding Cache Miss

- a book keeping mechanism
- conceptually similar to encapsulation in OOP
- applications can specify if a kernel object is shared among cores

SHARES EVALUATION

add/remove per-core segment to global/local share

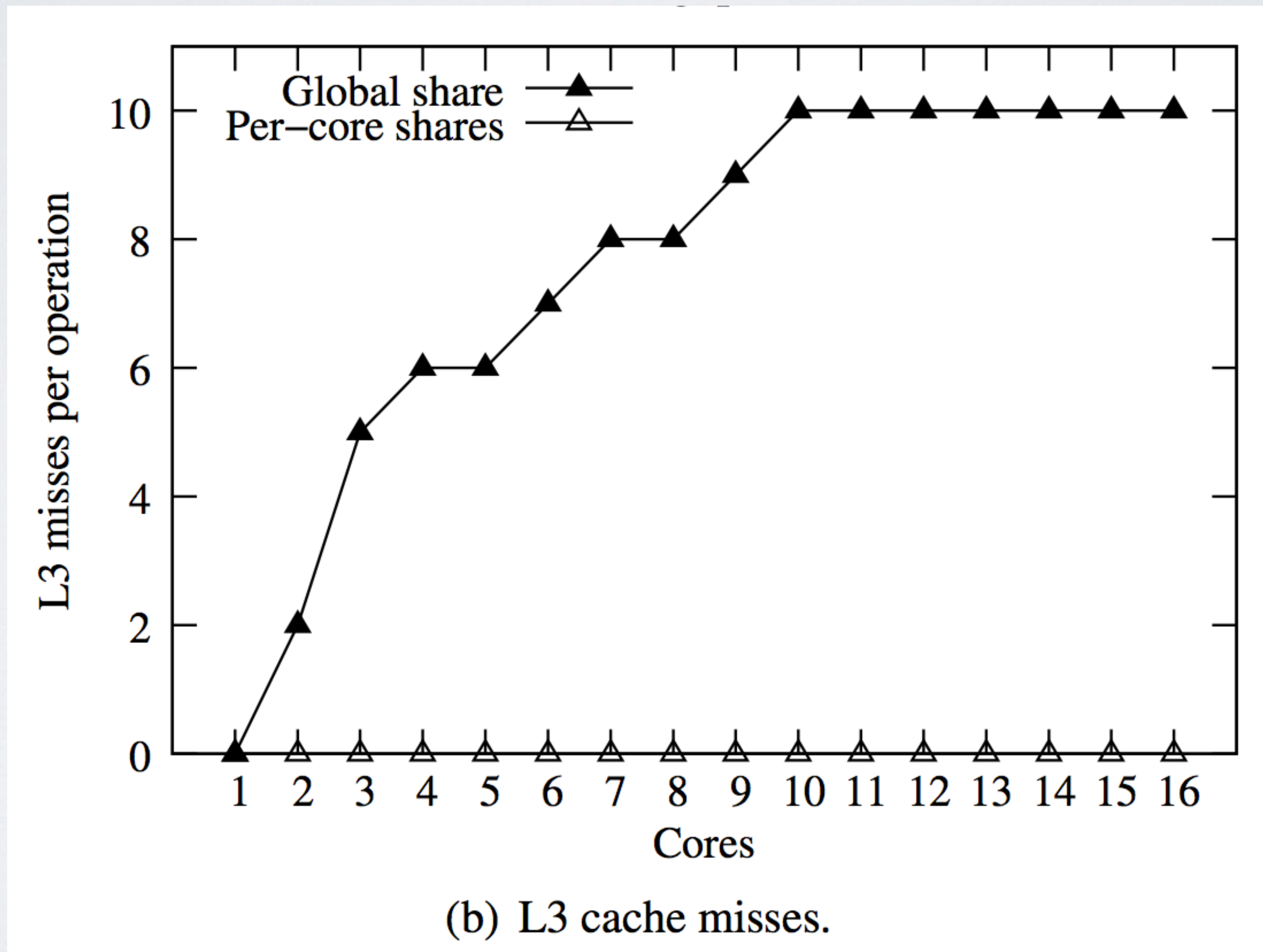


(a) Throughput.

modifying global data structure causes contention

SHARES EVALUATION

add/remove per-core segment to global/local share



(b) L3 cache misses.

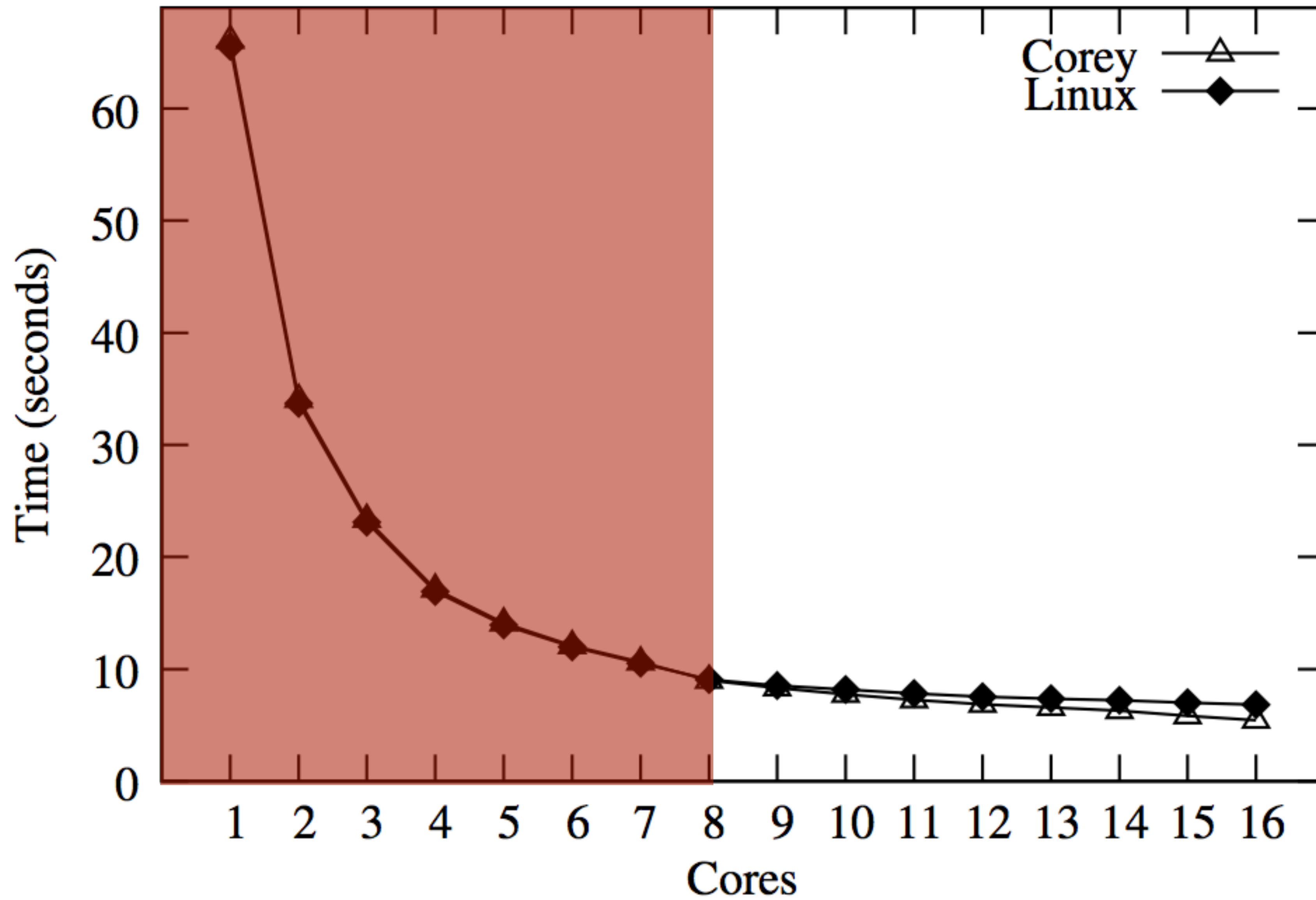
APPLICATION

- MapReduce
  - framework: Metis
  - bound to core by calling sched_setAffinity()
  - mostly benefit from address range
- webd
  - mostly benefit from kernel core

Mapreduce Evaluation

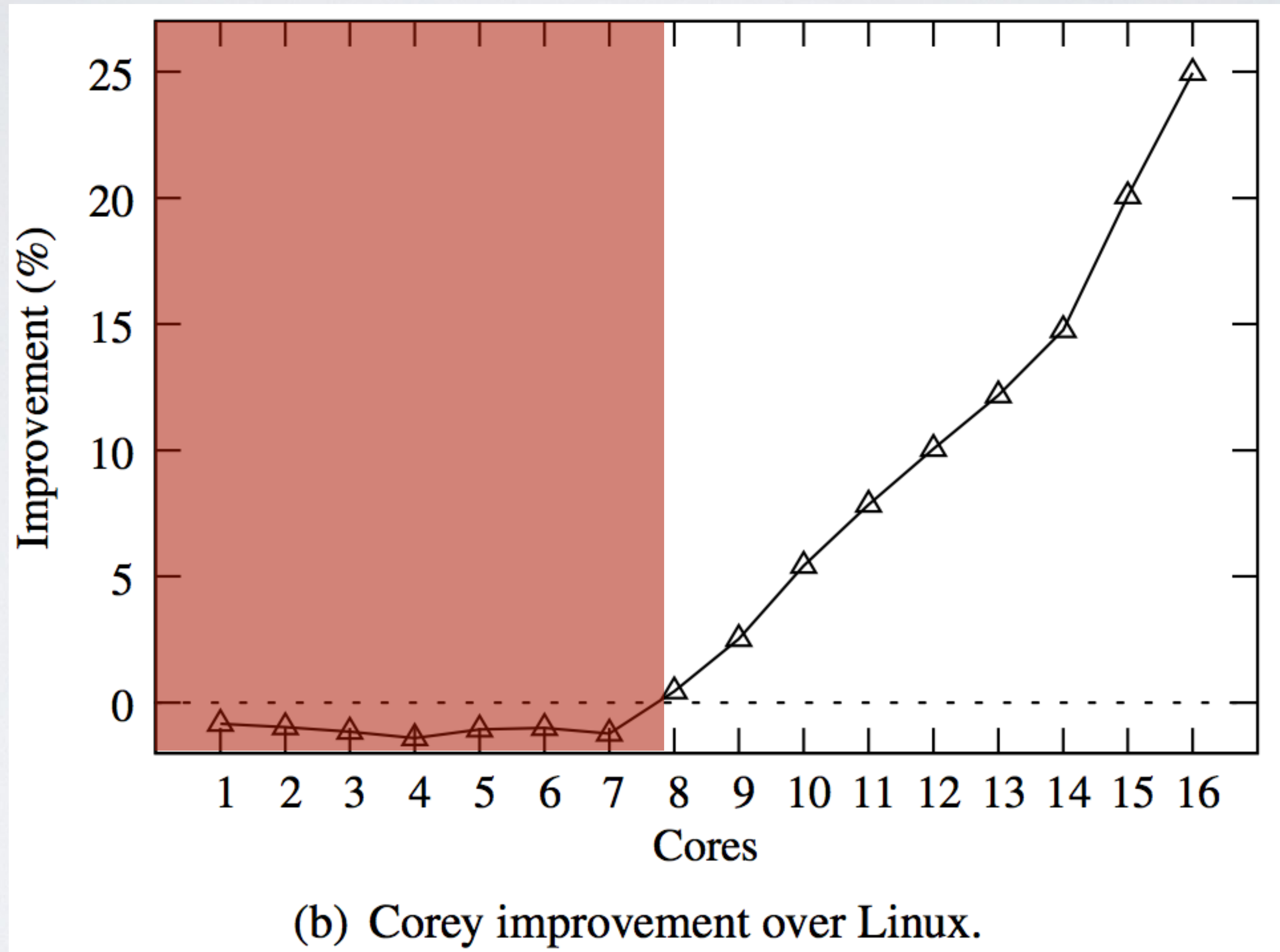
- word inverted index
- 1GB input, 2GB for intermediate value
- reducer copy intermediate value to shared address space

Mapreduce Evaluation



(a) Corey and Linux performance.

Mapreduce Evaluation (Cont.)



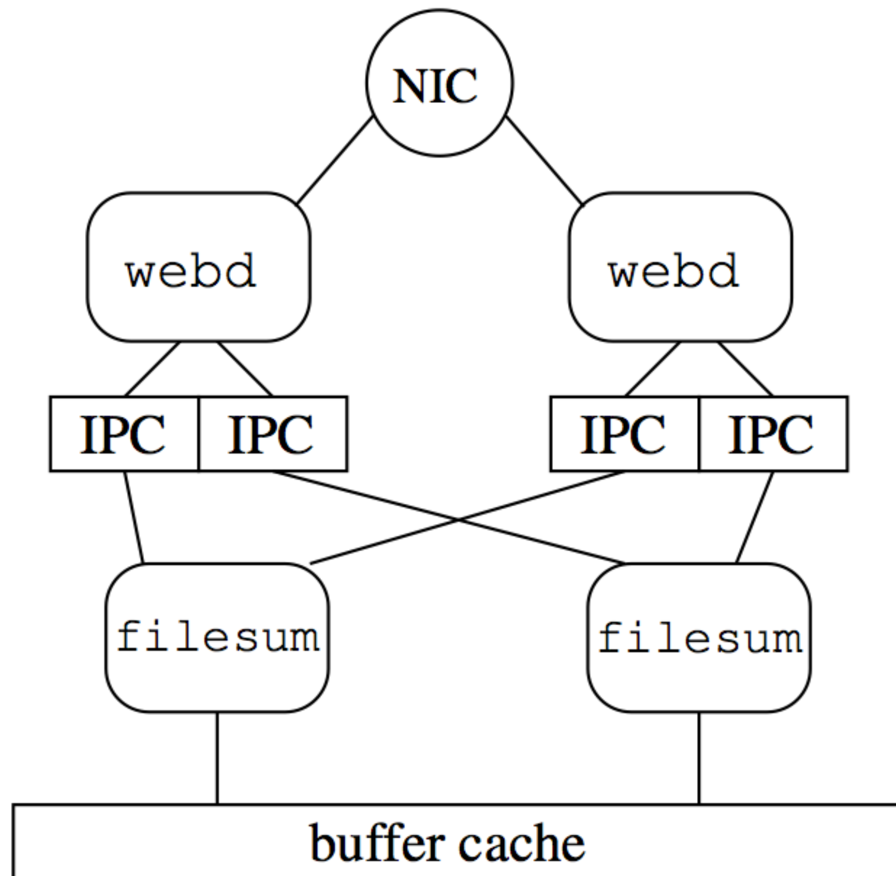
(b) Corey improvement over Linux.

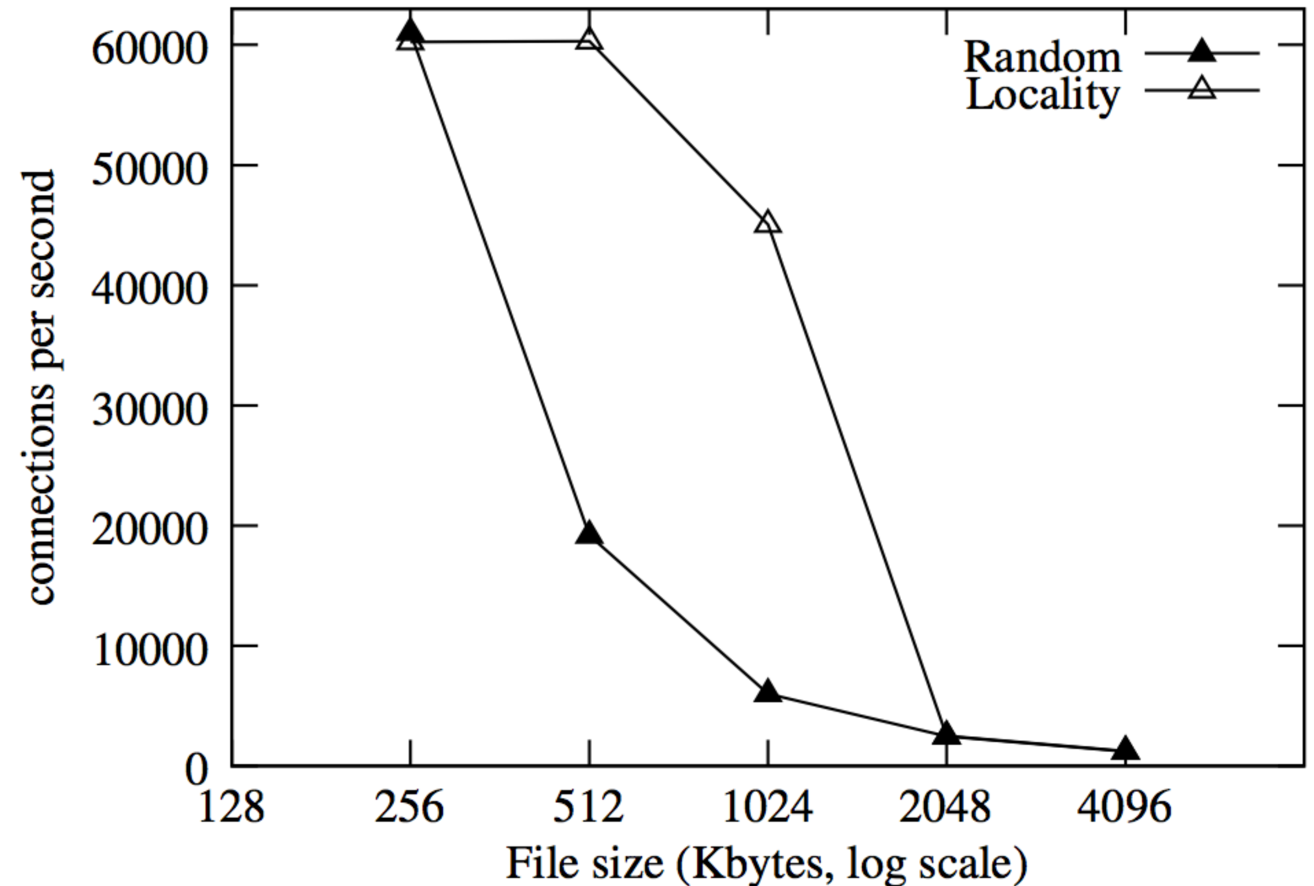
Linux's soft page fault handler is about 10% faster than Corey's when there is no contention

Webd Evaluation

- 8 webd core, 8 application core
- FileSum: returns sum of bytes of a given file
- 2 modes
  - **random**: webd is allowed to pass request to any application cores
  - **locality**: each each webd will only passes request to a certain application core

Webd Evaluation (Cont.)





when file is small, both mode is limited by webd's network stack
when file size is big, locality mode has (some) advantage of being able to cache bigger files (L3 Cache = 2MB)

Discussion

Baumann, Andrew, et al. "**The multikernel: a new OS architecture for scalable multicore systems**." Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009.

Han, Sangjin, et al. "**PacketShader: a GPU-accelerated software router**." ACM SIGCOMM Computer Communication Review 41.4 (2011): 195-206.

Boyd-Wickizer, Silas, et al. "**An analysis of Linux scalability to many cores.**" (2010).

Discussion

Baumann, Andrew, et al. "**The multikernel: a new OS architecture for scalable multicore systems**." Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009.

Han, Sangjin, et al. "**PacketShader: a GPU-accelerated software router**." ACM SIGCOMM Computer Communication Review 41.4 (2011): 195-206.

Boyd-Wickizer, Silas, et al. "**An analysis of Linux scalability to many cores.**" (2010)

Linux still has hope

Boyd-Wickizer, Silas, et al. "**An analysis of Linux scalability to many cores.**" (2010)

Linux still has hope

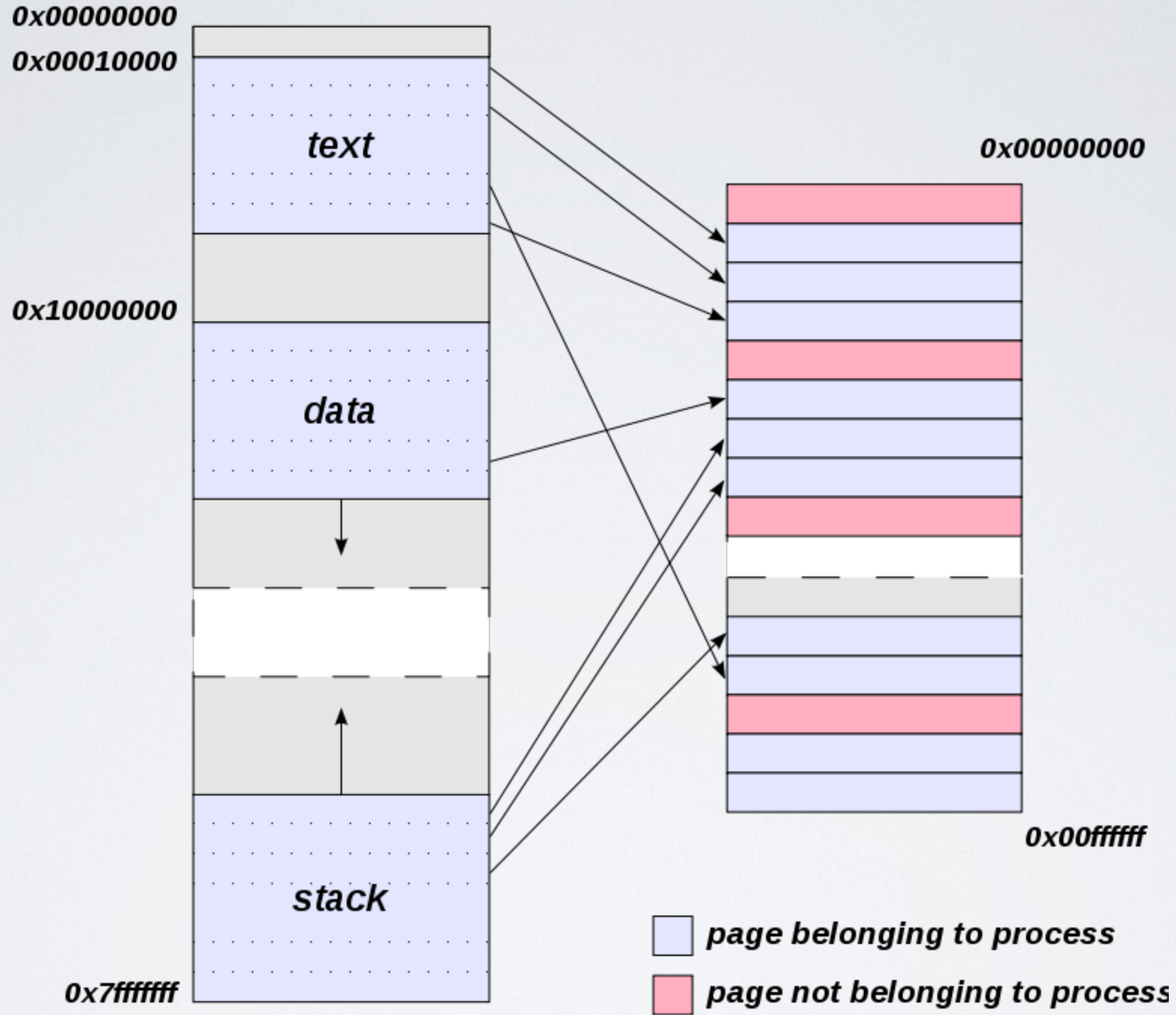
Discussion (Cont.)

Yoo, Richard M., Anthony Romano, and Christos Kozyrakis. "**Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system**." Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on. IEEE, 2009

Soares, Livio, and Michael Stumm. "**FlexSC: Flexible system call scheduling with exception-less system calls.**" Proceedings of the 9th USENIX conference on Operating systems design and implementation. USENIX Association, 2010.

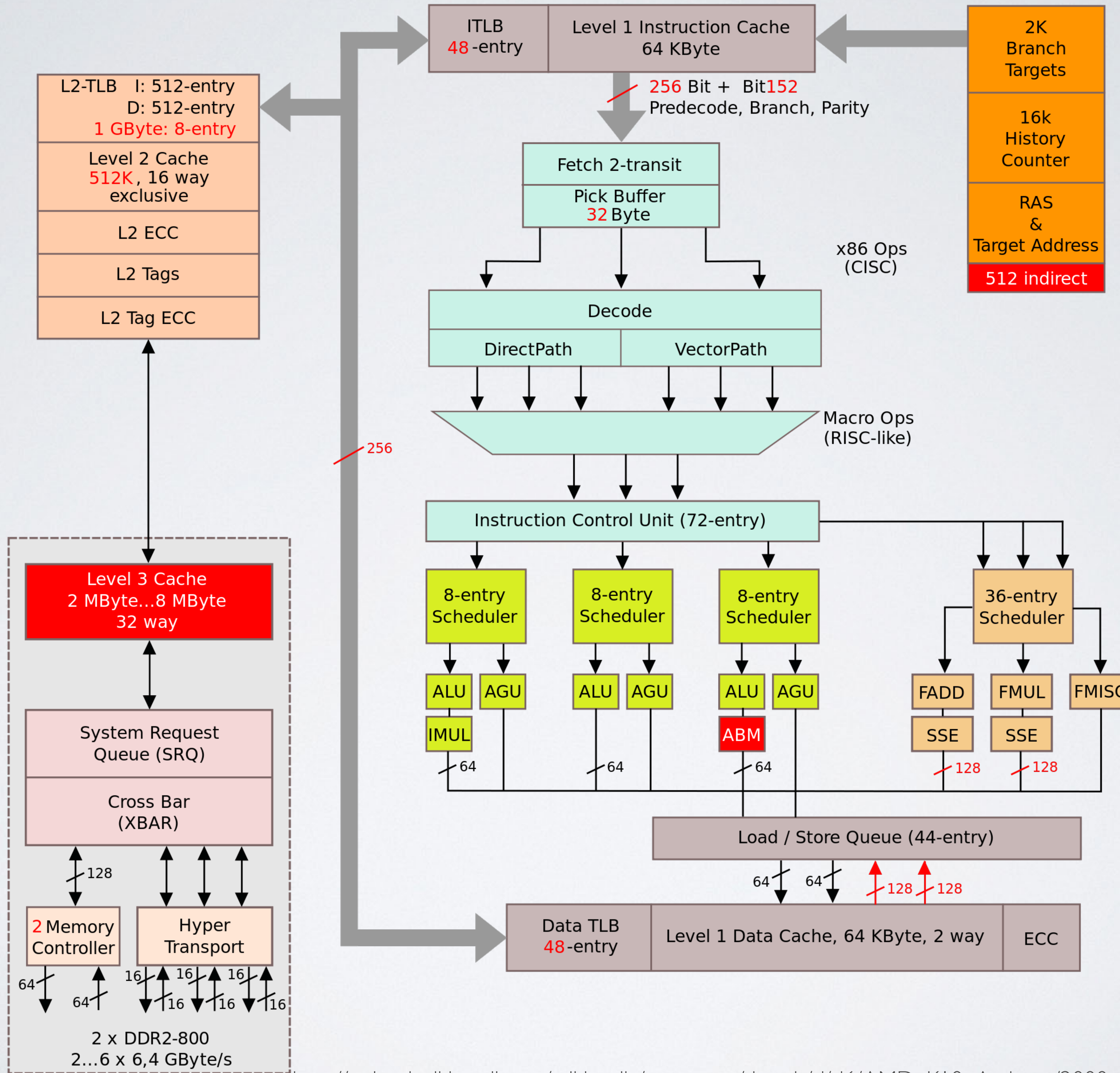
THE END

SUPPLEMENT SLIDE



source: http://upload.wikimedia.org/wikipedia/commons/thumb/3/32/Virtual_address_space_and_physical_address_space_relationship.svg/773px-Virtual_address_space_and_physical_address_space_relationship.svg.png

AMD K10 Architecture
Red: Difference between K8 and K10 Architecture
(Die Änderungen zwischen der K8- und K10-Architektur sind rot markiert)



http://upload.wikimedia.org/wikipedia/commons/thumb/d/d6/AMD_K10_Arch.svg/2000px-AMD_K10_Arch.svg.png

"The cache-management logic for the L3 cache is unique. When an item is loaded from L3 cache into a core's L1 cache (the L2 cache is always by-passed), the item is sometimes removed from the L3 cache and sometimes not. The determining factor is whether other cores are still accessing the item. If so, it's not removed from L3 and a copy of the data is loaded into L1. If no other cores are accessing the data item, then it is removed from the L3 cache"

http://upload.wikimedia.org/wikipedia/commons/thumb/d/d6/AMD_K10_Arch.svg/2000px-AMD_K10_Arch.svg.png

SYS_CLOSE

CLOSE FILE POINTED BY FD

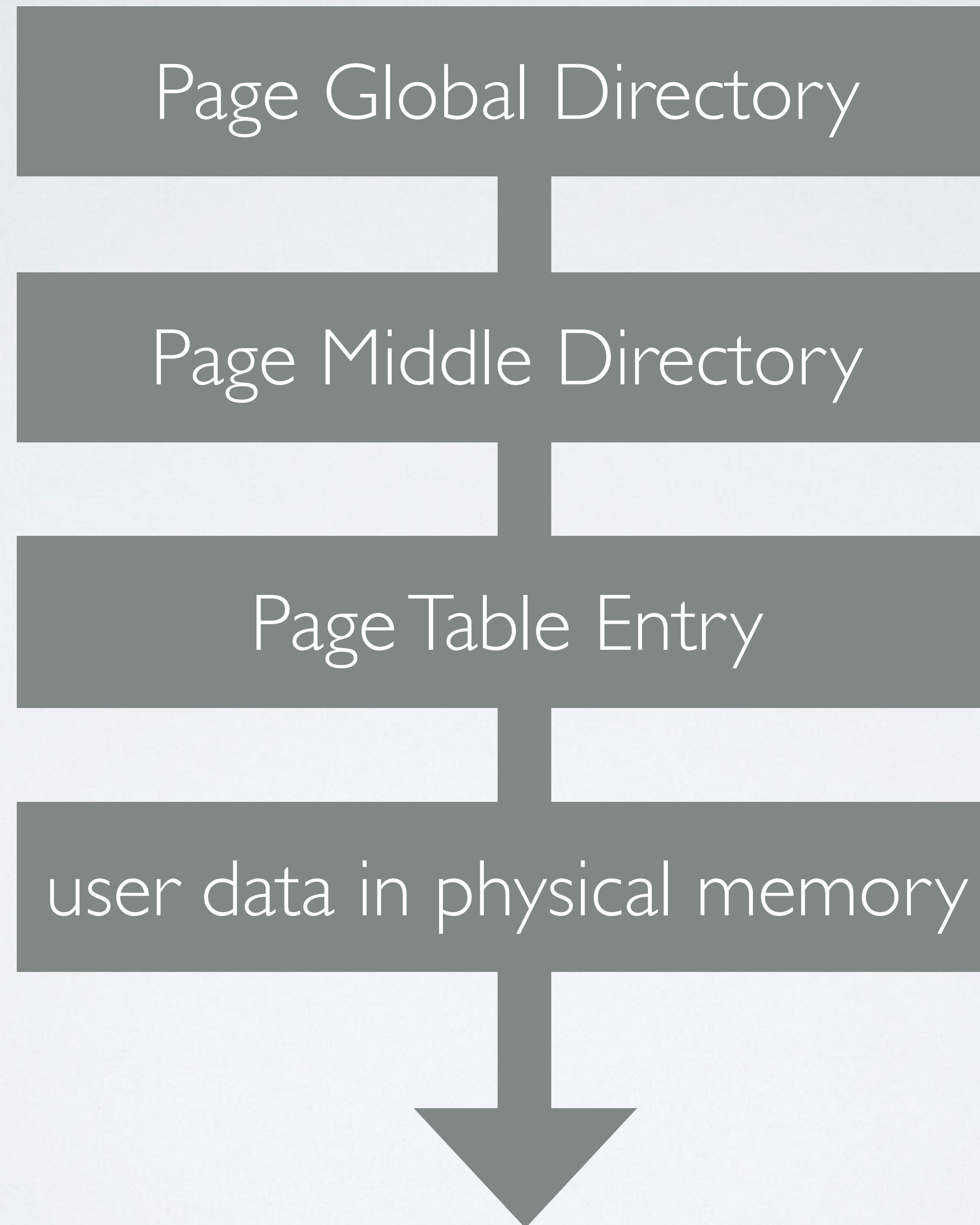
```
asmlinkage long sys_close(unsigned int fd)
{
      //... initialize
    spin_lock(&files->file_lock);
    fdt = files_fdtable(files);
                    //magic that releases the file descriptor
   spin_unlock(&files->file_lock);
   retval = filp_close(filp, files); // more locks in flip_close()
         //magic that checks error, in case of error, go to out_unlock
    return retval;
out_unlock:

    spin_unlock(&files->file_
    return -EBADF;
}
#define files_fdtable(files)
   (rcu_dereference((files)->fdt))
```

mm_struct

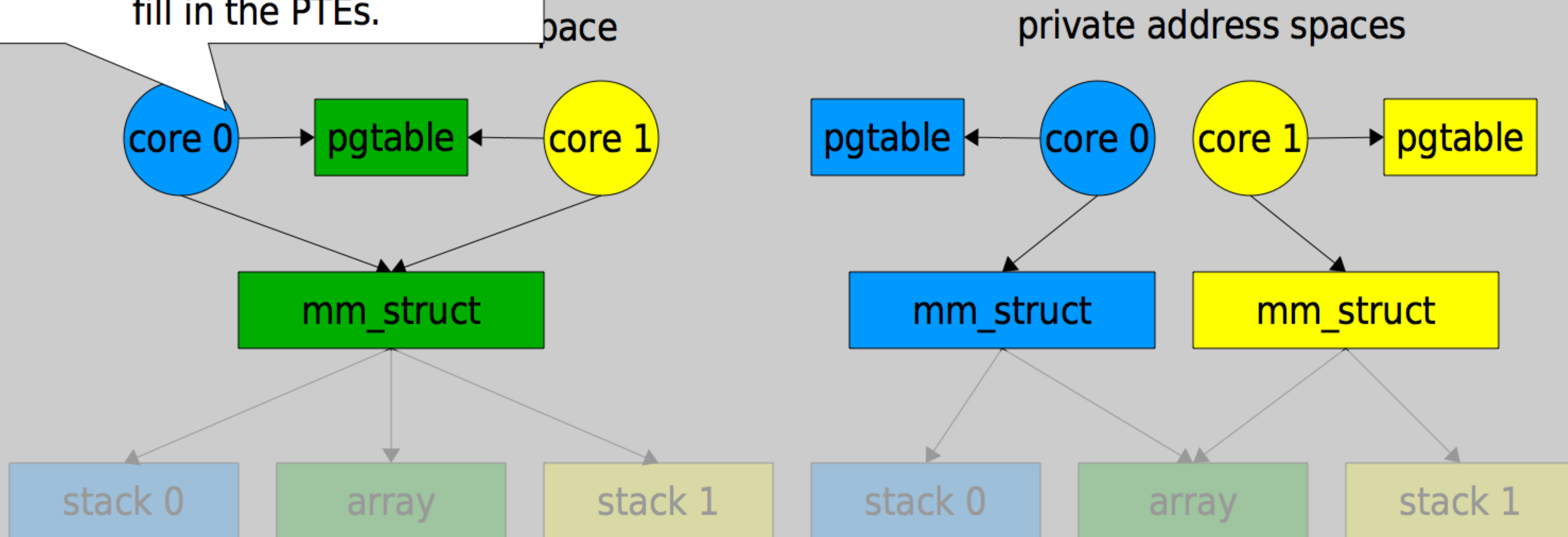
How Processes Use Page Table

```
struct mm_struct {
  int count;
  pgd_t * pgd; //page global directory
  unsigned long context;
  unsigned long start_code, end_code, start_data, end_data;
  unsigned long start_brk, brk, start_stack, start_mmap;
  unsigned long arg_start, arg_end, env_start, env_end;
  unsigned long rss, total_vm, locked_vm;
  unsigned long def_flags;
  struct vm_area_struct * mmap;
  struct vm_area_struct * mmap_avl;
  struct semaphore mmap_sem;
};
```

mm_struct (cont.)



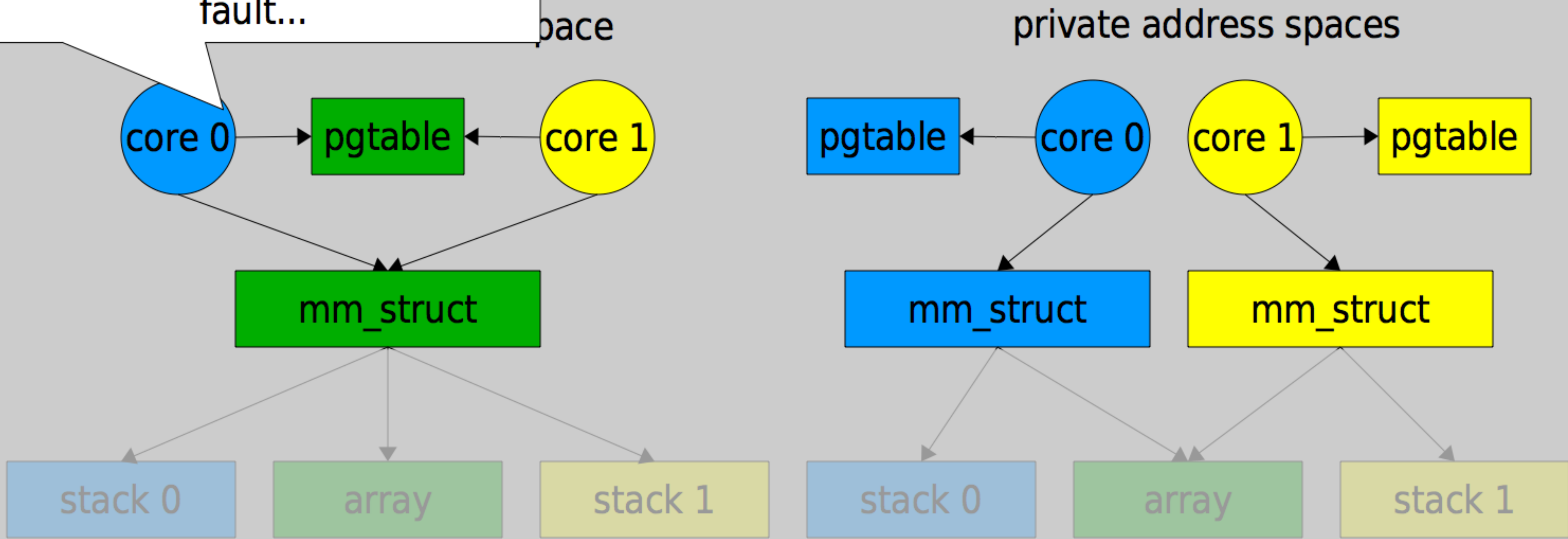
More costs: soft page faults



- Contend on mm_struct: locks, counters, lists...
- No contention on mm_struct

source: http://pdos.csail.mit.edu/~sbw/corey/osdi-12-08-2008.pdf

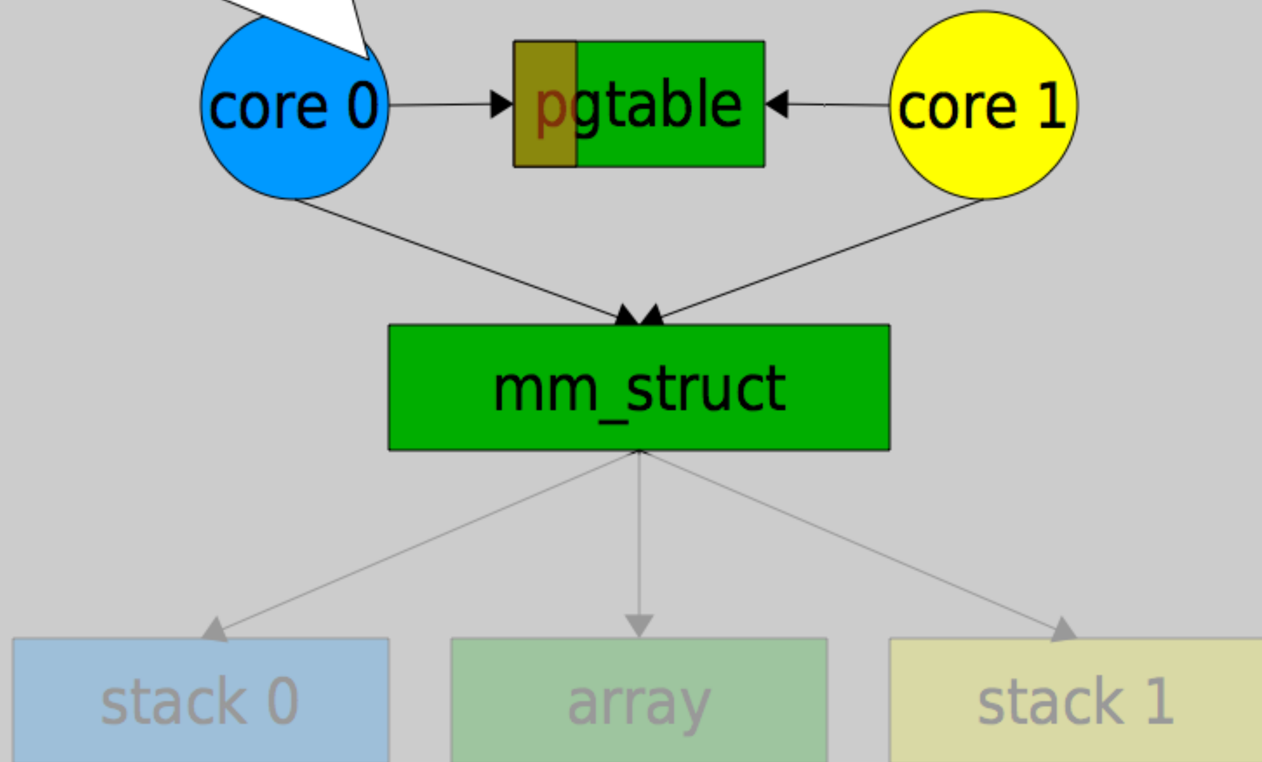
More costs: soft page faults

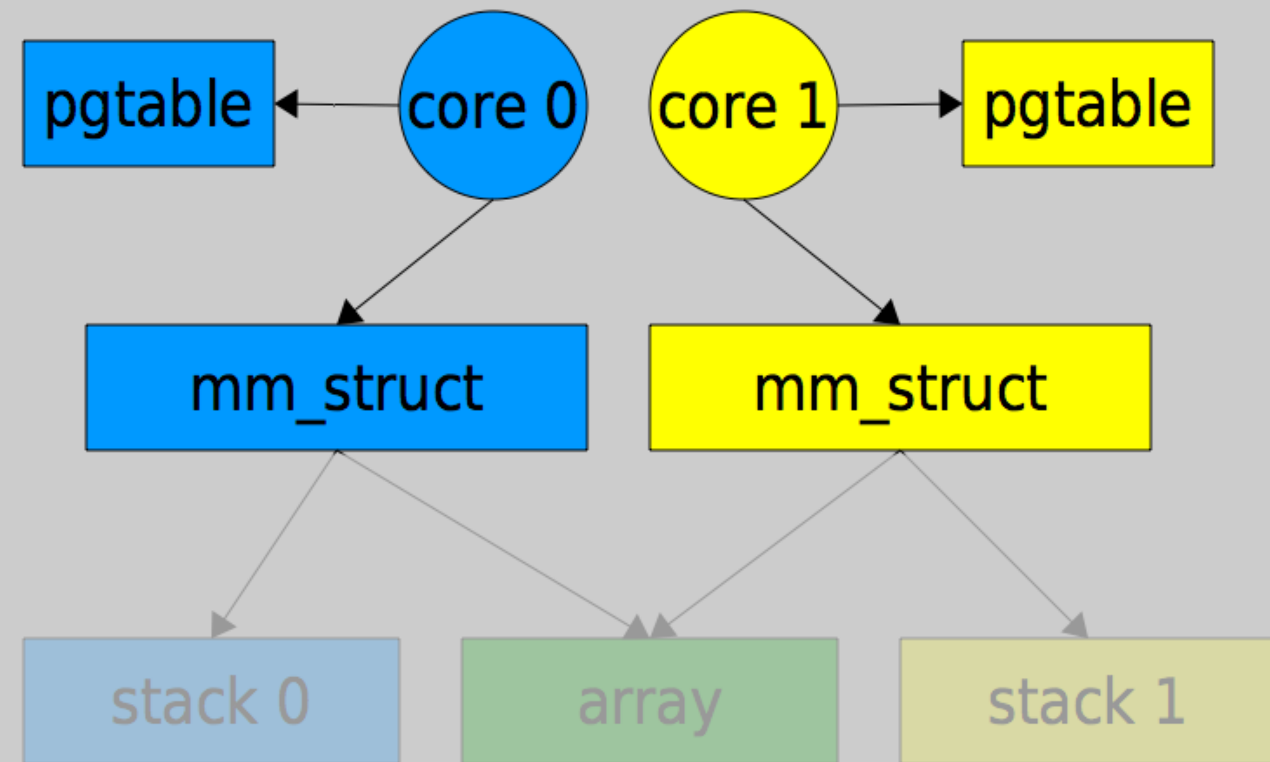


- Contend on mm_struct: locks, counters, lists...
- No contention on mm_struct

source: http://pdos.csail.mit.edu/~sbw/corey/osdi-12-08-2008.pdf

More costs: soft page faults





- Contend on mm_struct: locks, counters, lists...
- One soft page fault per page

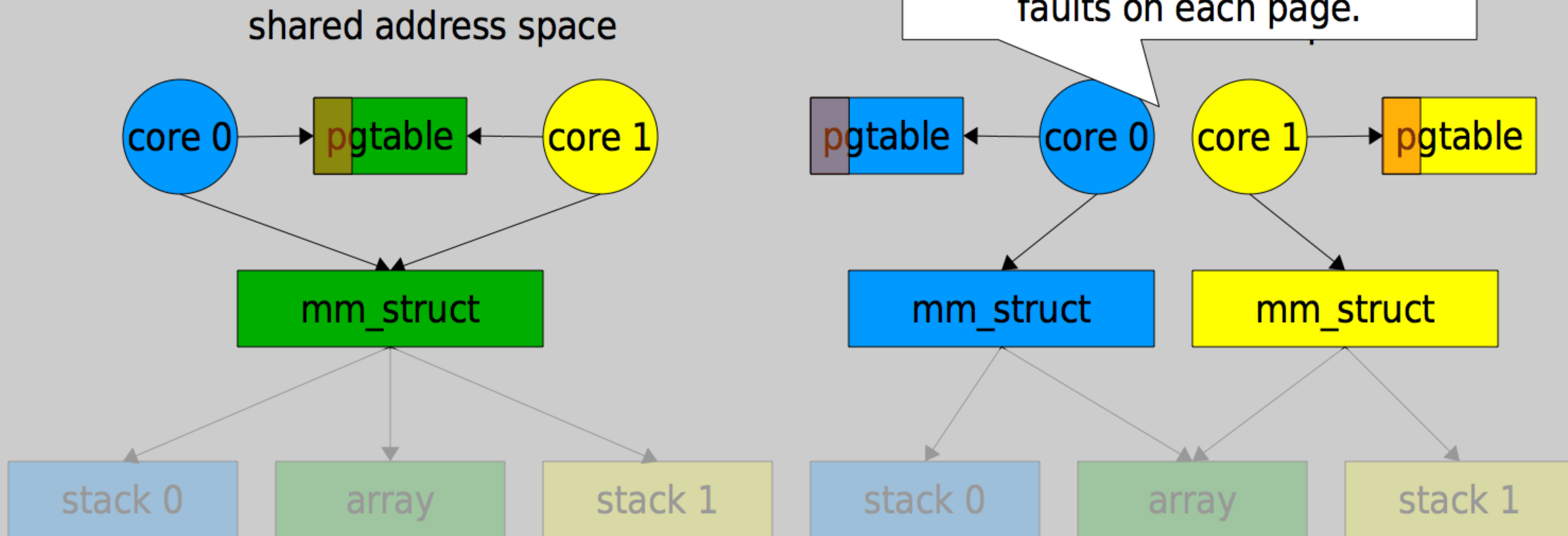
- No contention on mm_struct

source: http://pdos.csail.mit.edu/~sbw/corey/osdi-12-08-2008.pdf

More costs: soft page faults

- Linux lazily instantiates pa

Each mm_struct has a different pgtable, so each core soft page faults on each page.



- Contend on mm_struct: locks, counters, lists…
- One page fault per page

- No contention on mm_struct
- Each core soft page faults on each page

source: http://pdos.csail.mit.edu/~sbw/corey/osdi-12-08-2008.pdf

NOTES

- the presentation is for **Boyd-Wickizer, Silas, et al. "Corey: An Operating System for Many Cores." OSDI. Vol. 8. 2008.**
- graphs used in slide 4, 5, 11, 16, 18, 20, 25, 26, 28, 29, 32, 33, 35 are from original paper
- source code used in slide 7,8,9,10, 43, 44 are from Linux kernel 2.6.25 source code on http://www.cs.fsu.edu/~baker/devices/lxr/http/search?v=2.6.25
- this slide is created and presented by Jin, Yilong for CS5204 Fall 2014 on Oct. 16th 2014