Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

Supporting Parallelism in Operating Systems & Programming Languages

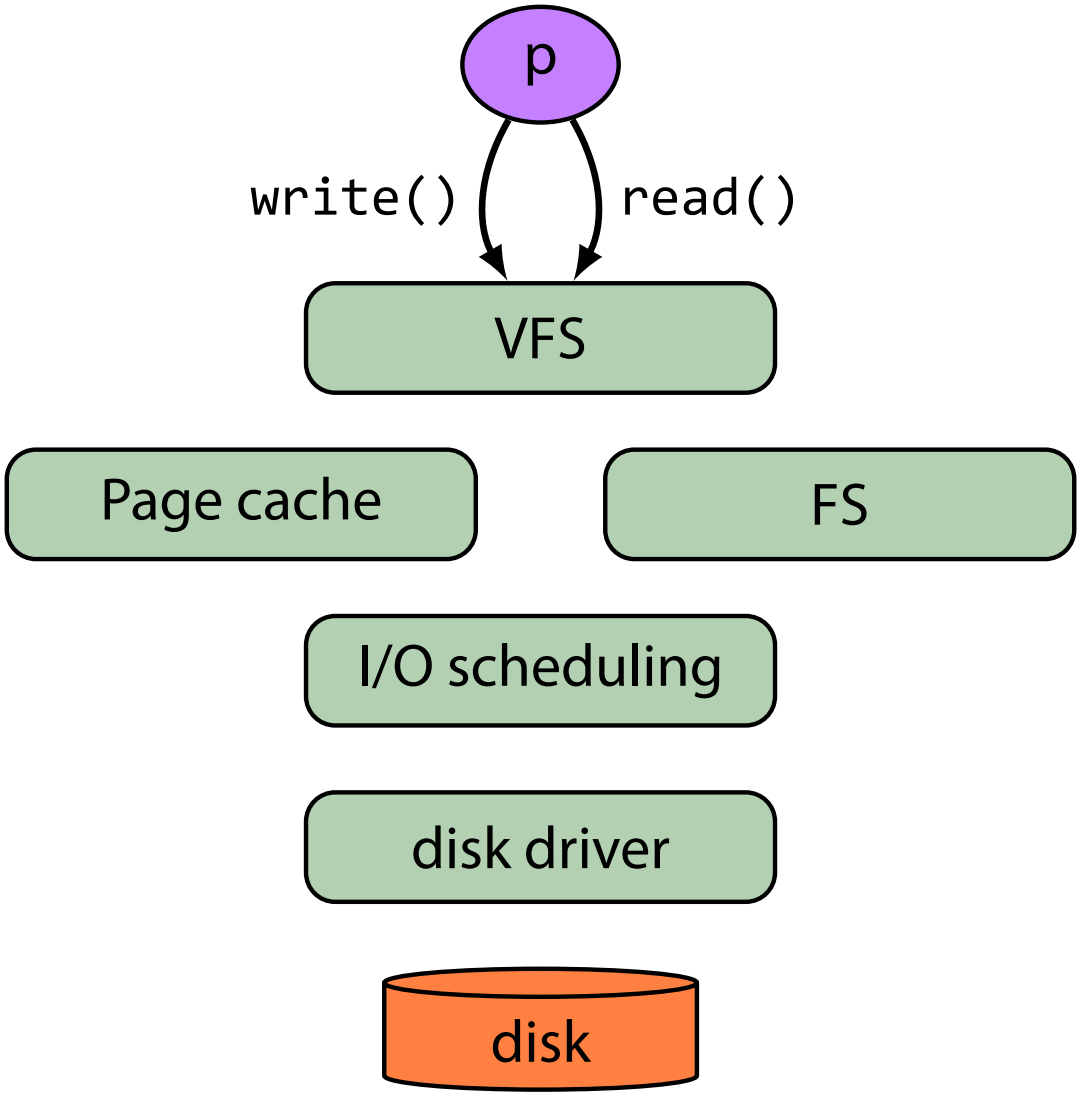
I/O

Kornilios Kourtis
<kornilios.kourtis@inf.ethz.ch>

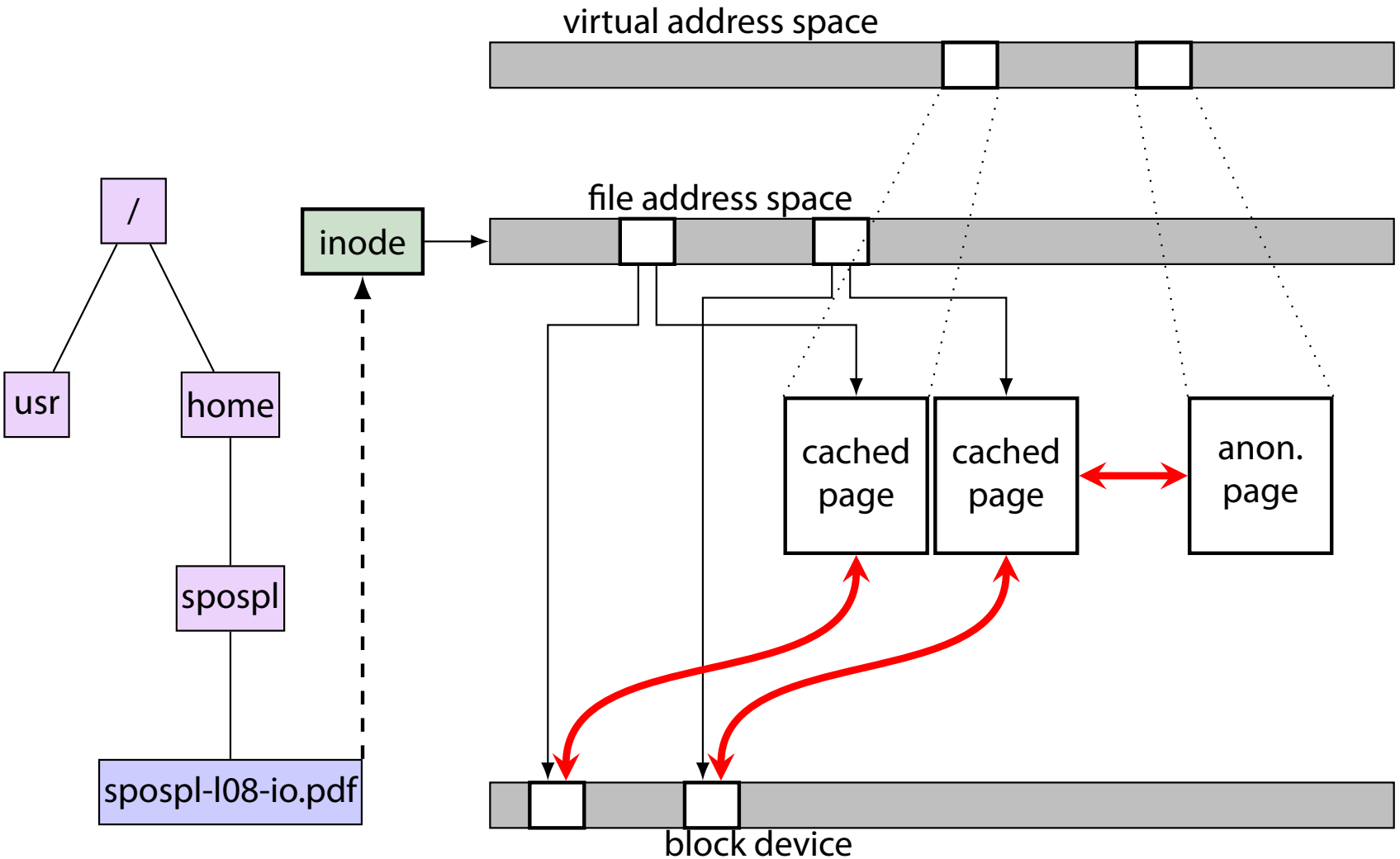
Lockless Page Cache

Network Scaling

FS Stack


p
write()
read()
VFS
Page cache
FS
I/O scheduling
disk driver
disk

Page Cache


virtual address space
file address space
inode
/
usr
home
spospl
spospl-l08-io.pdf
cached page
cached page
anon. page
block device

Linux page cache implementation
(overview)

- one (variable-height) radix tree per inode
- radix tree is tagged
  - dirty pages
  - write-back pages
- lookup:
  1. access radix tree to find page for file area
  2. increase page reference count
- lockless (lookup):
  - access radix tree using RCU
  - speculative references

Radix Tree

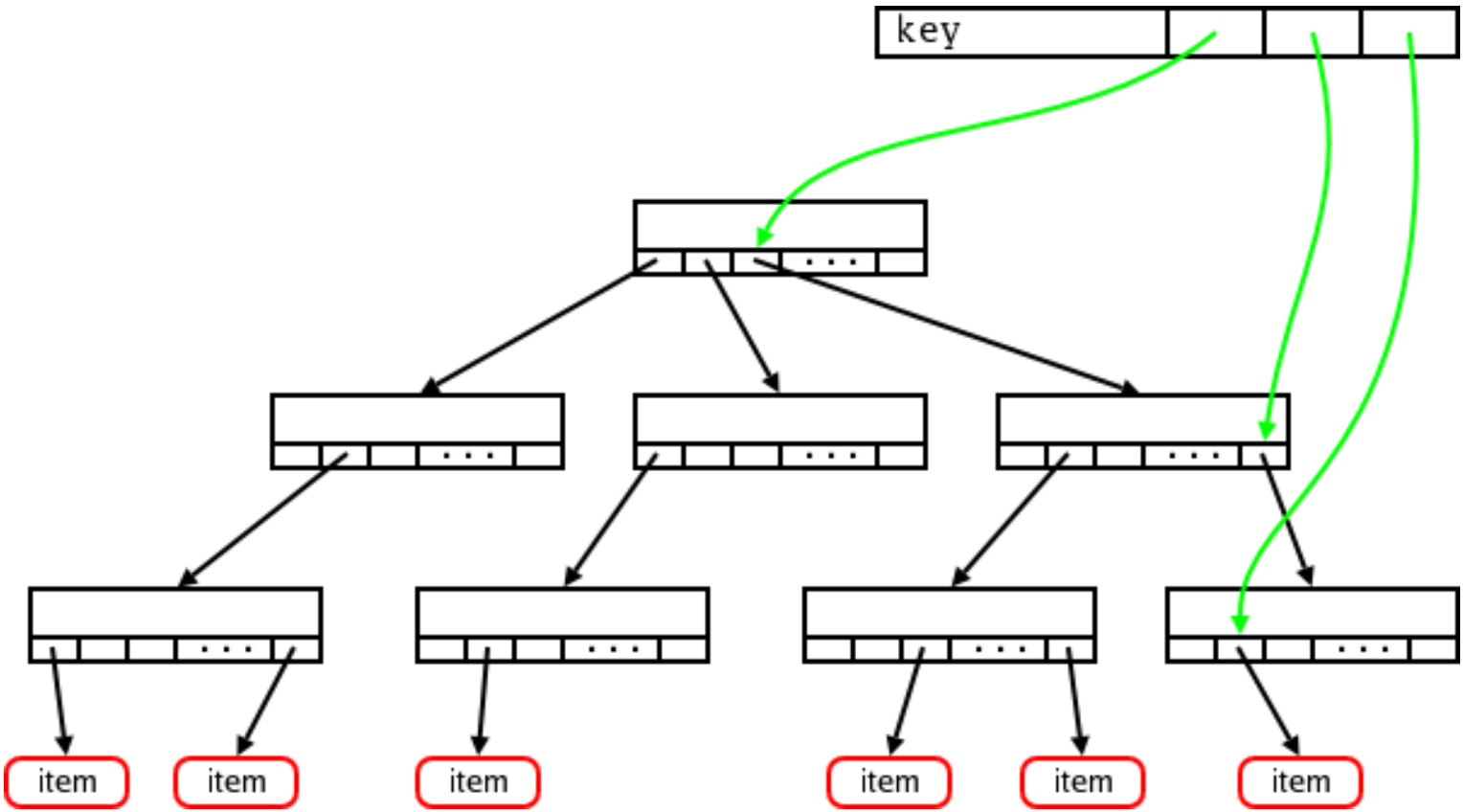
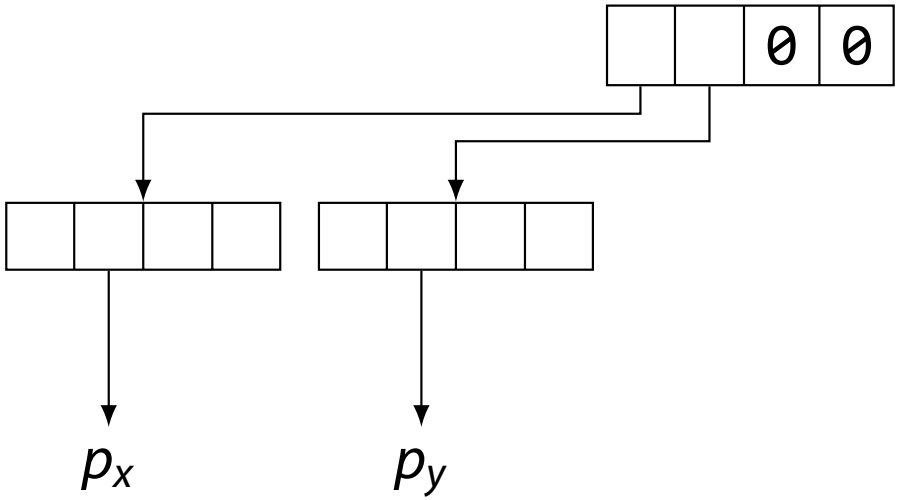


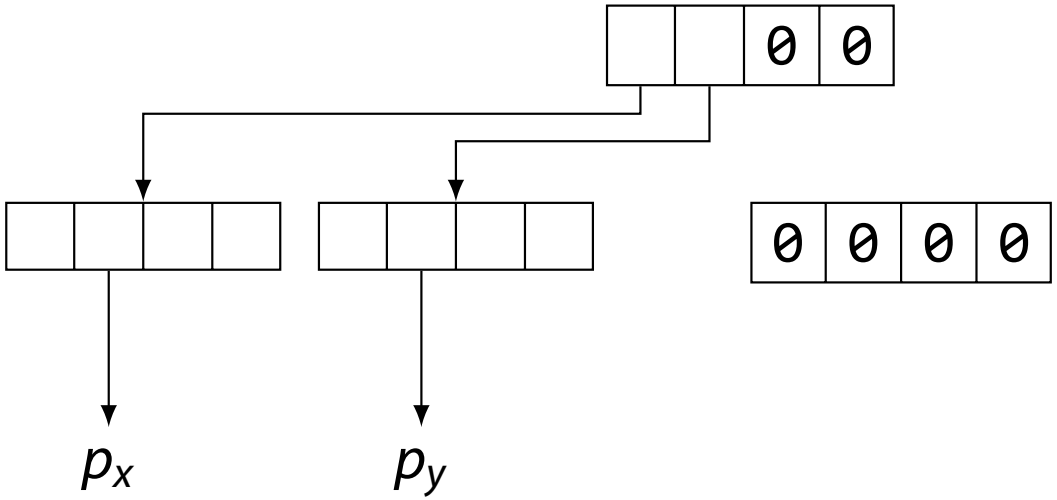
image from LWN: http://lwn.net/images/ns/kernel/radix-tree-2.png

RCU Radix Tree



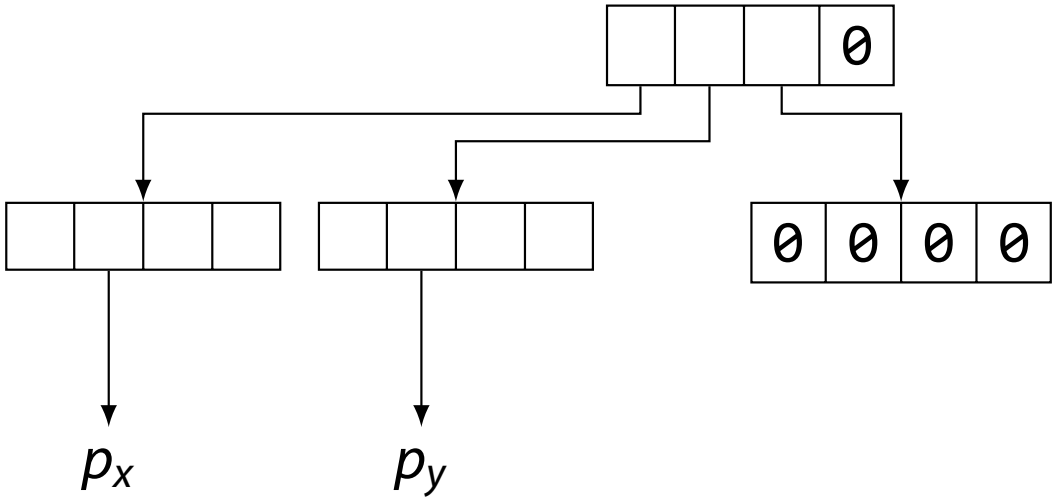
- (careful) RCU pointer assignments (`rcu_assign_pointer()`)

RCU Radix Tree



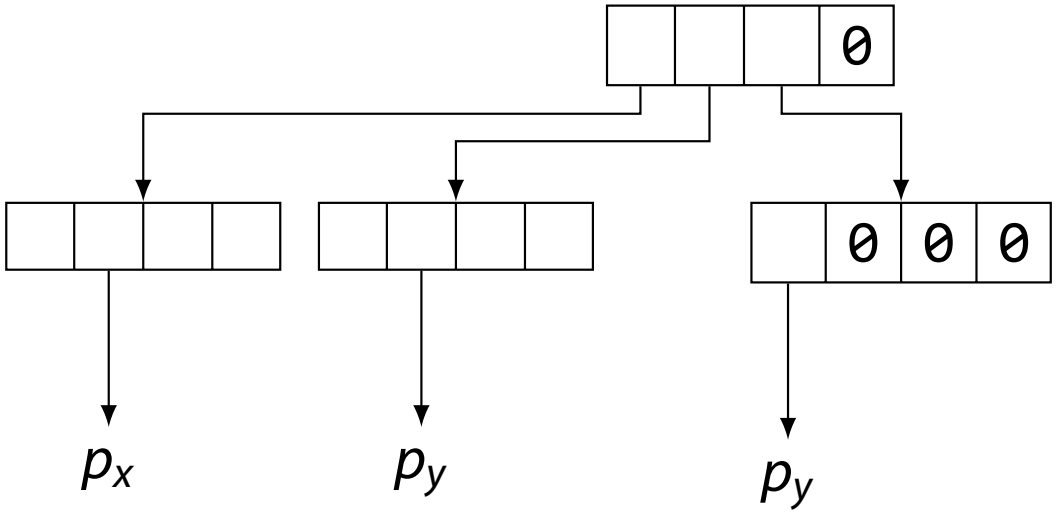
- (careful) RCU pointer assignments (`rcu_assign_pointer()`)

RCU Radix Tree



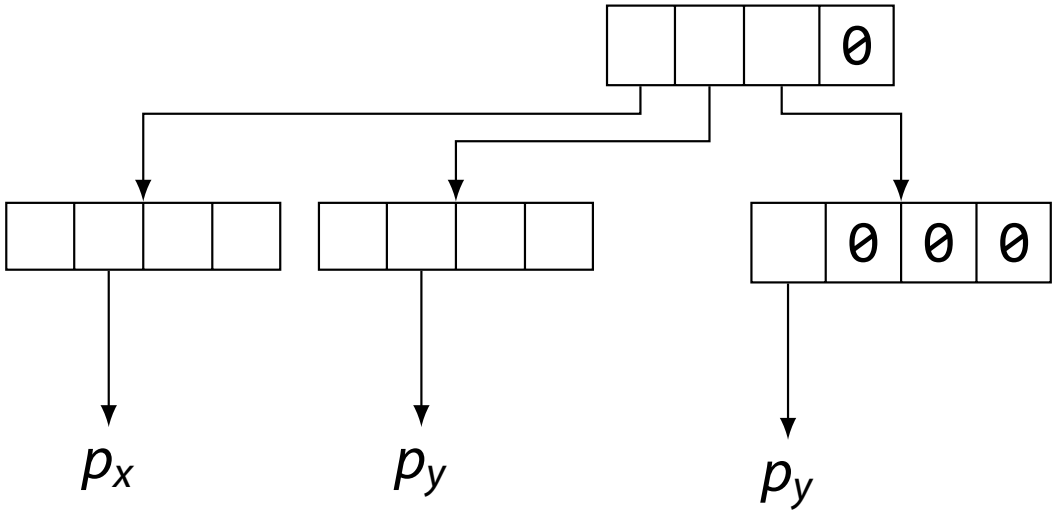
- (careful) RCU pointer assignments (`rcu_assign_pointer()`)

RCU Radix Tree



- (careful) RCU pointer assignments (`rcu_assign_pointer()`)

RCU Radix Tree



- (careful) RCU pointer assignments (`rcu_assign_pointer()`)
- defer deallocation pointer after RCU grace period (`call_rcu()`)

Speculative References

- lookup:
  1. access radix tree to find page for file area
  2. increase page reference count
- RCU radix tree is not enough
  - a page can be moved between above steps
- speculative references
  - get a (speculative) reference by increasing reference count
  - after the reference, check that page has not moved

find_get_page()

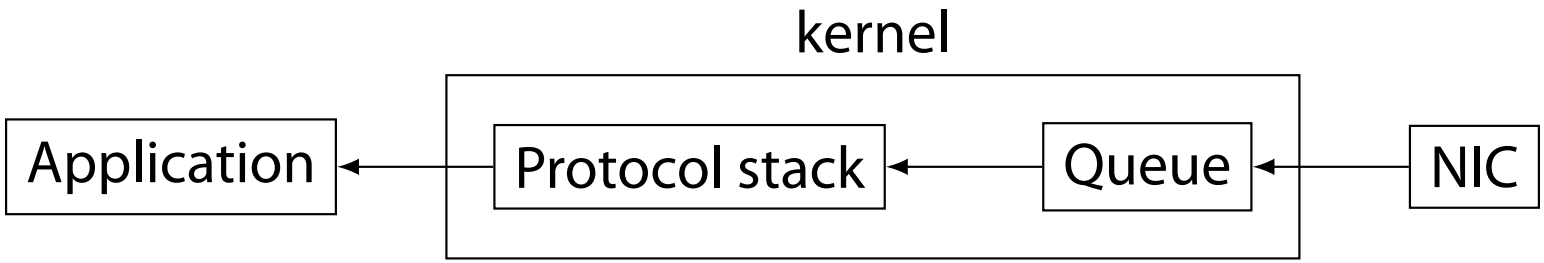
(many details omitted)

```
    rcu_read_lock();
repeat:
    page = NULL; // page
    pagep = radix_tree_lookup_slot(...); // page pointer
    if (pagep) {
        page = radix_tree_deref_slot(pagep);
        if (!page) goto out;

        if (!atomic_inc_not_zero(&p->cnt))
            goto repeat;

         if (page != *pagep) {
            page_cache_release(page); // release reference
            goto repeat;
         }
    }
out:
    rcu_read_unlock();
    return page;
```

Lockless Page Cache

Network Scaling

Traditional packet processing



- interrupts (interrupt coalescing/polling)
- ring-buffers
- DMA transfers

Network rates vs CPU clock rates

- NICs: GbE, 10GbE
- CPUs: clock rates remain the same
  - instead, multiple cores

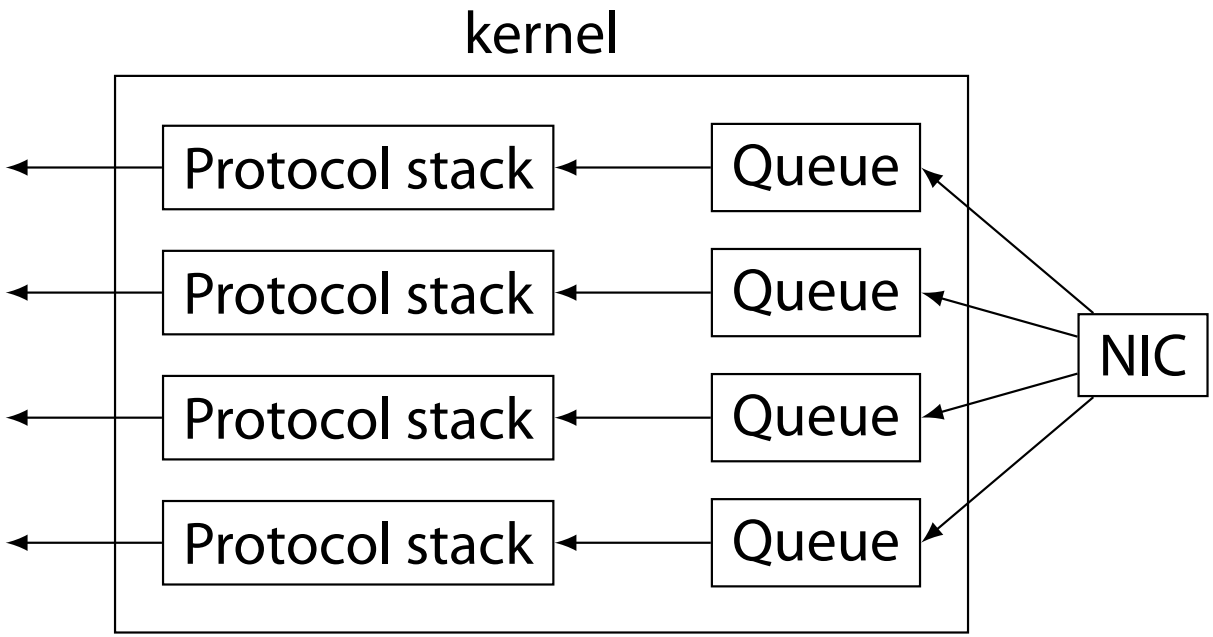
solutions:

- specialized hardware for network processing
- exploit multiple cores

Linux Networking scaling mechanisms

- RSS: Receive Side Scaling
- RPS: Receive Packet Steering
- RFS: Receive Flow Steering
- Accelerated Receive Flow Steering
- XPS: Transmit Packet Steering

Receive Side Scaling
(RSS)

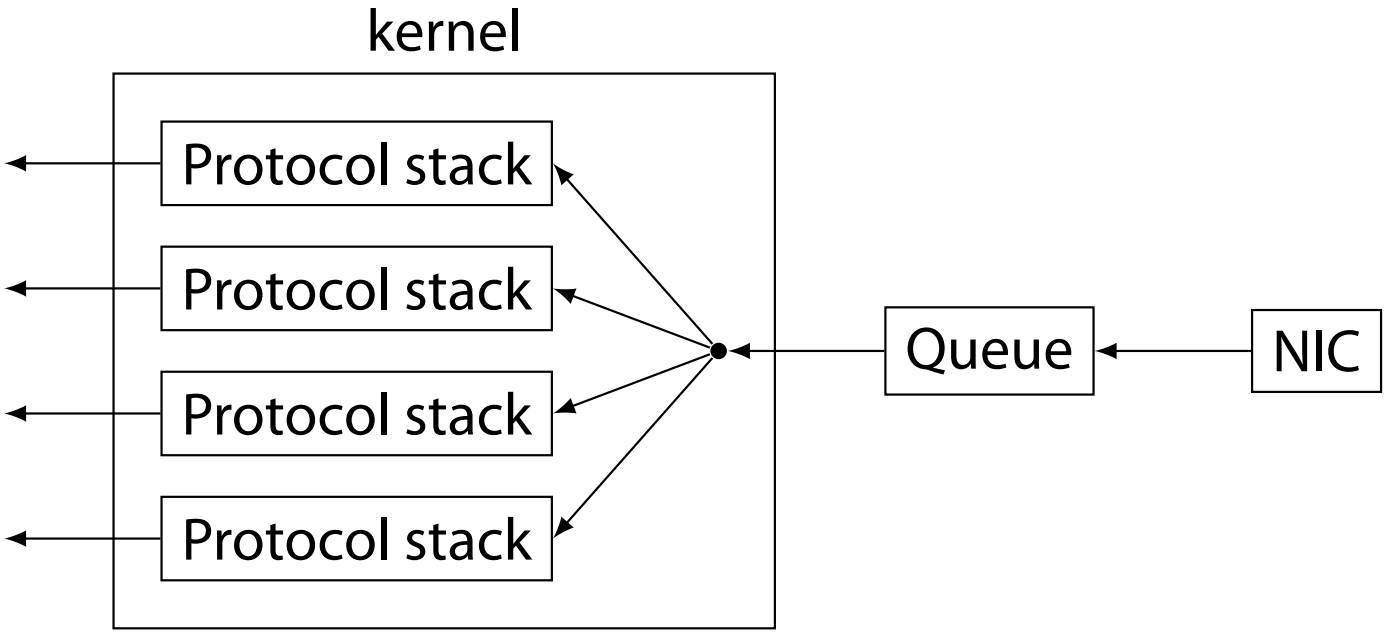


- NIC exports multiple queues
  - each with a different IRQ number
- NIC distributes packets to queues by applying filters
  - 4-tuple hash (IPs/ports)
  - advanced NICs allow programmable filters

RSS cont'd

- distribute queues to CPUs
  - one per core
  - one per NUMA domain
  - ...
- route interrupts to specific CPUs (IRQ affinity)
- application affinity

Receive Packet Steering
(RPS)



- RSS in software
- CPU is selected based on flow hash (e.g., 4-tuple)
- packet is enqueued in CPU's backlog queue and an IPI is sent

Receive Flow Steering: Motivation
(RFS)

RPS:

- uses hash to distributed packets to queues
- good load balancing
- does not consider application locality

RFS:

- aims to forward packet to CPU where relevant application runs

RFS

- table that maps flows to CPUs that last processed flow (desired CPU)
- table that maps flows to CPUs with outstanding packets (current CPU)
- mapping may differ when thread is migrated by scheduler
- current CPU only changes to desired CPU when there are no outstanding packets to avoid packet reordering

Accelerated RFS

- hardware-based (similar to RSS, support from NIC needed)
- network stack communicates to NIC appropriate queue for a flow

Transmit Packet Steering
(XPS)

- select queue when transmitting a packet
- CPU-to-queue mapping
- reduces (or eliminates) contention on queue lock
- better cache utilization

Listen sockets

- initialized to listen for connections on an (address, port) pair

  (using `listen()`)
- states for new connections:
  - pending: got SYN/waiting for ACK
  - accepted: got ACK
- application pulls accepted connections

  (using `accept()`)

Issues:

- incomming con. processing on a single socket scales poorly
- maximizing performance requires matching:
  - core that process packet for protocol stack (kernel)
  - core that application that receives packet runs (user-space)

Affinity-Acccept

[Pesterev et al. 2012]

Approach:

- per-core accept queue
- `accept()` returns a connection from the local core queue

Load Balancing:

- connection stealing (short-term)
  - when local accept queue empty, steal from other queues
- flow group migration (long-term)
  - move flows to different queues

References

- A Lockless Pagecache in Linux – Introduction, Progress, Performance
  Nick Piggin
  OLS '06
- The lockless page cache
  Jonathan Corbet (LWN)
  2008
- Radix Trees
  Jonathan Corbet (LWN)
  2006
- Scaling in the Linux Networking Stack
  `Documentation/networking/scaling.txt`
- Improving Network Connection Locality on Multicore Systems
  Pesterev et al.
  Eurosys '12