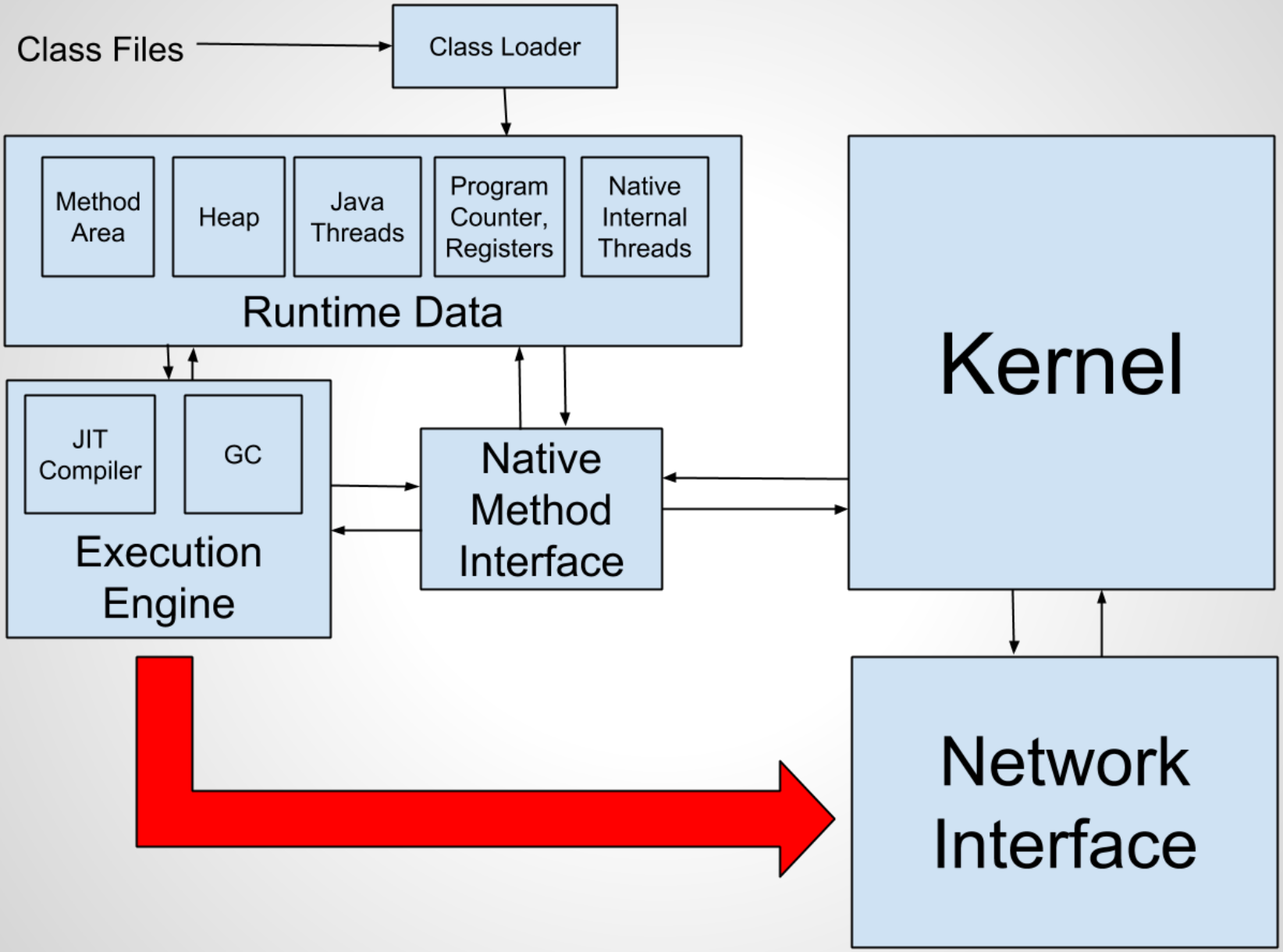
Packet Processing in Java

Ashley Hemingway

Whats the Problem?

- Middleboxes in native languages are inflexible to emerging technologies
- Poor Java networking capabilities even though high performance processing is possible


Class Files
Class Loader
Method Area
Heap
Java Threads
Program Counter, Registers
Native Internal Threads
Runtime Data
JIT Compiler
GC
Execution Engine
Native Method Interface
Kernel
Network Interface

DPDK

- NIC driver in user space
- Idea of logical cores (lcores)
- Multiple queues per port
- No interrupts, just polling
- 2 application models

DPDK-Java

- Framework wrapping DPDK
- Easy creation of middlebox applications in Java
- Eliminates error checking
- Integration with other Java based applications

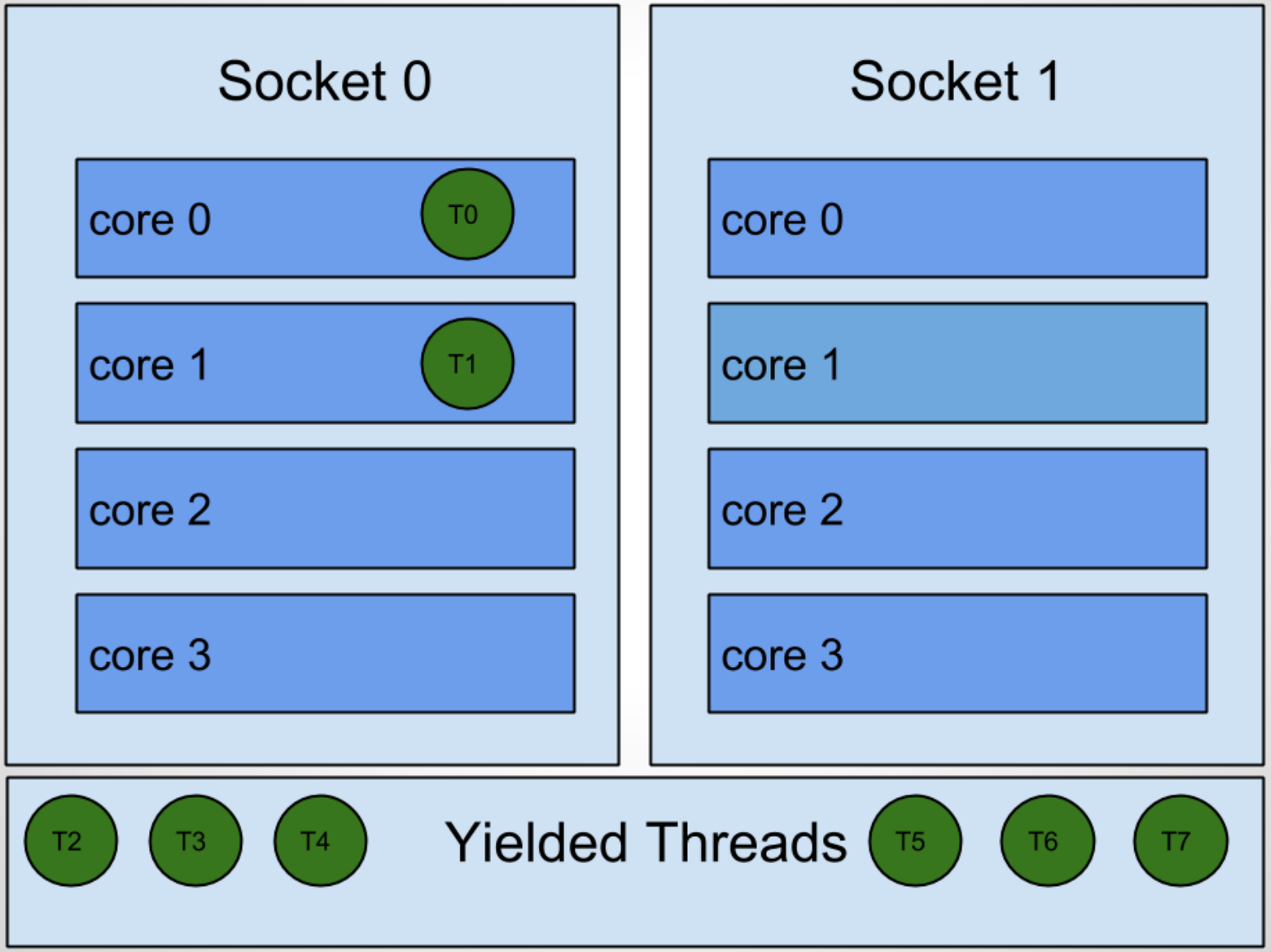
```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <errno.h>
#include <assert.h>
#include <sys/queue.h>

#include <rte_ip.h>
#include <rte_memory.h>
#include <rte_memzone.h>
#include <rte_launch.h>
#include <rte_tailq.h>
#include <rte_eal.h>
#include <rte_per_lcore.h>
#include <rte_lcore.h>
#include <rte_debug.h>
#include <rte_ethdev.h>
#include <rte_ring.h>
#include <rte_mempool.h>
#include <rte_mbuf.h>
#include <rte_timer.h>
#include <rte_cycles.h>

#include "main.h"

#define MAX_PKT_BURST 512

#define MAX_PKT_SIZE (2*1024 + sizeof
(struct rte_mbuf) +
RTE_PKTMBUF_HEADROOM)

#define NB_RX_QUEUE 2

#define NB_TX_QUEUE 2

#define         NB_RX_DESC 256

#define         NB_TX_DESC 256

struct rte_mbuf *rx_mbufs[MAX_PKT_BURST];

// For configuring an ethernet port
static struct rte_eth_conf eth_conf = {
    .rxmode = {
        .mq_mode = ETH_MQ_RX_RSS,
        .split_hdr_size = 0,
        .header_split = 0,
        .hw_ip_checksum = 1,
        .hw_vlan_filter = 0,
        .jumbo_frame = 0,
        .hw_strip_crc = 0,
    },
    .rx_adv_conf = {
        .rss_conf = {
            .rss_key = NULL,
            .rss_hf = ETH_RSS_IP,
        },
    },
    .txmode = {
        .mq_mode = ETH_MQ_TX_VMDQ_ONLY
    },
};

// Configures a TX ring of an Ethernet port
static struct rte_eth_txconf tx_conf = {
    .tx_thresh = {
        .pthresh = 36,
        .hthresh = 0,
        .wthresh = 0,
    },
    .tx_rs_thresh = 0,
    .tx_free_thresh = 0,

// Configures a RX ring of an Ethernet port
static struct rte_eth_rxconf rx_conf = {
    .rx_thresh = {
        .pthresh = 8,
        .hthresh = 8,
        .wthresh = 4,
    },
    .rx_free_thresh = 64,
    .rx_drop_en = 0,
};

struct mbuf_list {
    unsigned len;
    struct rte_mbuf *list[MAX_PKT_BURST];
} free_list, send_list;

// RTE mempool structure
static struct rte_mempool *rx_pool;
static void send_packet(struct rte_mbuf *);
static void free_packet(struct rte_mbuf *);
static void send_burst(void);
//static void free_burst(void);

static uint32_t blacklist[] = {0, 1};
int size = 2;

static void send_burst(void) {
    struct rte_mbuf **list = (struct rte_mbuf **)
send_list.list;
    int i;
    if (rte_lcore_id() == 1) {
      i = 0;
    } else {
      i = 1;
    }
    int ret = rte_eth_tx_burst(0, i, list,
MAX_PKT_BURST);

    if (unlikely(ret < MAX_PKT_BURST)) {
        do {
            free_packet(list[ret]);
        } while (++ret < MAX_PKT_BURST);
    }
}

static void send_packet(struct rte_mbuf *m) {
    unsigned len = send_list.len;
    send_list.list[len] = m;
    len++;

    /* enough pkts to be sent */
    if (unlikely(len == MAX_PKT_BURST)) {
        send_burst();
        len = 0;
    }
    send_list.len = len;
}

static void free_packet(struct rte_mbuf *m) {

int main_loop(void *);

int main_loop(__attribute__ ((unused)) void *arg) {
  //  long pktcount = 0;
    int recv_cnt, i, ifidx = 0;
    struct ipv4_hdr *iphdr;
    struct rte_mbuf *m;
    int id = rte_lcore_id();
    printf("Core %i starting\n", id);
    fflush(stdout);
    int b;
    if (id == 1) {
        b = 0;
    } else {
        b = 1;
    }
    while(1) {
        recv_cnt = rte_eth_rx_burst(ifidx, b, rx_mbufs,
MAX_PKT_BURST);
        if (recv_cnt < 0) {
            if (errno != EAGAIN && errno != EINTR) {
                perror("rte_eth_rx_burst()");
                assert(0);
            }
        }
        if ( recv_cnt > 0) {
//            pktcount += recv_cnt;
            for (i = 0 ; i < recv_cnt; i++) {
                m = rx_mbufs[i];
                iphdr = (struct ipv4_hdr *)rte_pktmbuf_adj
(m, (uint16_t)sizeof(struct ether_hdr));
                //RTE_MBUF_ASSERT(iphdr != NULL);

                uint32_t dest_addr = rte_be_to_cpu_32
(iphdr->dst_addr);

                int drop = 0;

                int a;
                for (a = 0; a < size; a++) {
                    if (blacklist[a] == dest_addr) {
                        drop = 1;
                    }
                }

                if (drop == 0) {
                    send_packet(m);
                    continue;
                }

                free_packet(m);
            }
        }
    }
    return 0;
}

void timer_setup(void);
void do_stats(struct rte_timer *, void *);

//received bytes
uint64_t pre_ibytes = 0;
//received packets - successful
uint64_t pre_ipackets = 0;

void do_stats(__attribute__ ((unused)) struct
rte_timer *tim, __attribute__ ((unused)) void *arg) {
    //printf("IN STATS");
    //fflush(stdout);
    struct rte_eth_stats stats;

    rte_eth_stats_get(0, &stats);
    uint64_t diff_bytes = stats.ibytes - pre_ibytes;
    pre_ibytes = stats.ibytes;
    uint64_t diff_packets = stats.ipackets -
pre_ipackets;
    pre_ipackets = stats.ipackets;
    printf("Bytes: %lu\n", diff_bytes);
    printf("Packets: %lu\n", diff_packets);
    fflush(stdout);
}

static struct rte_timer timer;

void timer_setup(void) {
    int lcore_id = rte_get_master_lcore();
    rte_timer_subsystem_init();
    rte_timer_init(&timer);
    int ret = rte_timer_reset(&timer,
rte_get_timer_hz(), PERIODICAL, lcore_id,
do_stats, NULL);
    if (ret != 0) {
        printf("TIMER_ERROR");
        fflush(stdout);
    }
}

int main(int argc, char **argv) {
    int ret, ifidx = 0;
    //unsigned lcore_id;
    uint8_t count;

    struct rte_eth_link link;

    // initialise eal
    ret = rte_eal_init(argc, argv);
    if (ret < 0) {
        rte_panic("Cannot init EAL\n");
    }

    count = rte_eth_dev_count();
    printf("# of eth ports = %d\n", count);
    //memset(&eth_conf, 0, sizeof eth_conf);
    ret = rte_eth_dev_configure(0, 2, 2, &eth_conf);
    if (ret < 0) {
        rte_exit(EXIT_FAILURE, "Cannot configure
device: error=%d, port=%d\n", ret, 0);
    }
    printf("If %d rte_eth_dev_configure()
successful\n", ifidx);

    rx_pool = rte_mempool_create("rx_pool",
16*1024, MAX_PKT_SIZE, 0,
                                 sizeof (struct
rte_pktmbuf_pool_private),
                                 rte_pktmbuf_pool_init, NULL,
                                 rte_pktmbuf_init, NULL, 1, 0);


    if (rx_pool == NULL) {
        rte_exit(EXIT_FAILURE,
"rte_mempool_create(): error\n");
    }

  ret = rte_eth_rx_queue_setup(ifidx, 0, NB_RX_DESC, 1,
&rx_conf, rx_pool);
    if (ret < -1) {
        rte_exit(EXIT_FAILURE,
"rte_eth_rx_dev_queue_setup(): error=%d, port=%d\n", ret,
ifidx);
    }
    printf("If %d rte_eth_rx_queue_setup() successful\n",
ifidx);

  ret = rte_eth_rx_queue_setup(0, 1, NB_RX_DESC, 1,
&rx_conf, rx_pool);
    if (ret < -1) {
                rte_exit(EXIT_FAILURE, "2 probs");
    }

    ret = rte_eth_tx_queue_setup(ifidx, 0, NB_TX_DESC, 1,
&tx_conf);
    if (ret < 0) {
        rte_exit(EXIT_FAILURE, "rte_eth_tx_queue_setup():
error=%d, port=%d\n", ret, ifidx);
    }
    printf("If %d rte_eth_tx_queue_setup() successful\n",
ifidx);

    ret = rte_eth_tx_queue_setup(ifidx, 1, NB_TX_DESC, 1,
&tx_conf);
    if (ret < 0) {
        rte_exit(EXIT_FAILURE, "rte_eth_tx_queue_setup():
error=%d, port=%d\n", ret, ifidx);
    }
    printf("If %d rte_eth_tx_queue_setup() successful\n",
ifidx);

    ret = rte_eth_dev_start(ifidx);
    if (ret < 0) {
        rte_exit(EXIT_FAILURE, "rte_eth_dev_start(): error=%
d, port=%d\n", ret, ifidx);
    }
    printf("If %d rte_eth_dev_start() successful\n", ifidx);

    rte_eth_link_get(ifidx, &link);
    if (link.link_status == 0) {
        rte_exit(EXIT_FAILURE, "DPDK interface is down: %
d\n", ifidx);
    }
    printf("If %d is UP and RUNNING\n", ifidx);

    rte_eth_promiscuous_enable(ifidx);

    struct lcore_config lc = lcore_config[0];
    printf("%i", lc.core_id);

    uint32_t a;
    for (a = 0; a < 32; a++) {
        if (a == rte_get_master_lcore() || !rte_lcore_is_enabled
(a)) {
            continue;
        }
        ret = rte_eal_remote_launch(main_loop, NULL, a);
        if (ret != 0) {
            printf("ERROR");
        }
    }
    rte_delay_ms(1000);
    timer_setup();

    for(;;) {
        rte_timer_manage();
    }
```

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;

public class Firewall {

	public static void main(String[] args) {

		ApplicationStarter as = new ApplicationStarter();

		try {
			as.readConfig(new FileInputStream("config.properties"));
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}

		as.sendDPDKInformation();

		List<CoreThread> threads = new ArrayList<CoreThread>();
		ReceivePoller rp1 = new ReceivePoller(0, 0);
		ReceivePoller rp2 = new ReceivePoller(0, 1);
		PacketSender ps1 = new PacketSender(0, 0);
		PacketSender ps2 = new PacketSender(0, 1);
		PacketFreeer pf1 = new PacketFreeer();
		PacketFreeer pf2 = new PacketFreeer();

		threads.add(new FirewallProcessor(ps1, pf1, rp1, "PROCESS 1", 3));
		threads.add(new FirewallProcessor(ps2, pf2, rp2, "PROCESS 2", 5));

		as.createAffinityThreads(threads);

		as.dpdk_init_eal();
		as.dpdk_create_mempool("mbufs", 8192*4, 32, 1);
		as.dpdk_check_ports();
		as.dpdk_configure_dev(0, 2, 2);
		as.dpdk_configure_rx_queue(0, 0, 1);
		as.dpdk_configure_tx_queue(0, 0, 1);
		as.dpdk_configure_rx_queue(0, 1, 1);
		as.dpdk_configure_tx_queue(0, 1, 1);
		as.dpdk_dev_start(0);
		as.dpdk_check_ports_link_status();

		as.dpdk_enable_pro();
		as.start_native_stats(1);
		as.dpdk_get_mac_info();

		as.startAll();
	}

}

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

public class FirewallProcessor extends PacketProcessor {

	Set<Long> blacklist;

	PacketFreeer pf_ind;
	ReceivePoller rp_ind;
	PacketSender ps_ind;

	public FirewallProcessor(PacketSender ps, PacketFreeer pf, ReceivePoller rp, int core) {
		super(ps, pf, rp, core);
		blacklist = new HashSet<Long>();
		readBlacklist();
		pf_ind = pf;
		rp_ind = rp;
		ps_ind = ps;
	}

	public FirewallProcessor(PacketSender ps, PacketFreeer pf, ReceivePoller rp, String name, int core) {
		super(ps, pf, rp, name, core);
		blacklist = new HashSet<Long>();
		readBlacklist();
		pf_ind = pf;
		rp_ind = rp;
		ps_ind = ps;
	}

	private boolean inspect(Packet currentPacket) {
		int version = currentPacket.whichIP();

		if (version == 4) {
			Ipv4Packet cp = (Ipv4Packet)currentPacket;
			if (blacklist.contains(cp.getSrcAddr())) {
				pf_ind.freePacket(currentPacket);
			} else {
				ps_ind.sendPacket(currentPacket);
				return true;
			}
		} else if (version == 6) {
			pf_ind.freePacket(currentPacket);
		} else {
			pf_ind.freePacket(currentPacket);
		}
		return false;
	}

	private void readBlacklist() {
		File file = new File("blacklist.txt");
		FileReader fileReader;
		try {
			fileReader = new FileReader(file);
			BufferedReader bufferedReader = new BufferedReader(fileReader);
			String line;
			try {
				while ((line = bufferedReader.readLine()) != null) {
					blacklist.add(Utils.IpToInt(line));
				}
			} catch (IOException e1) {
				e1.printStackTrace();
			}
			try {
				fileReader.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
	}

	@Override
	public void run() {
		while (true) {
			PacketList packets = rp_ind.getBurst();
			if (packets != null) {
				for (Packet p : packets) {
					inspect(p);
				}
			}
		}
```

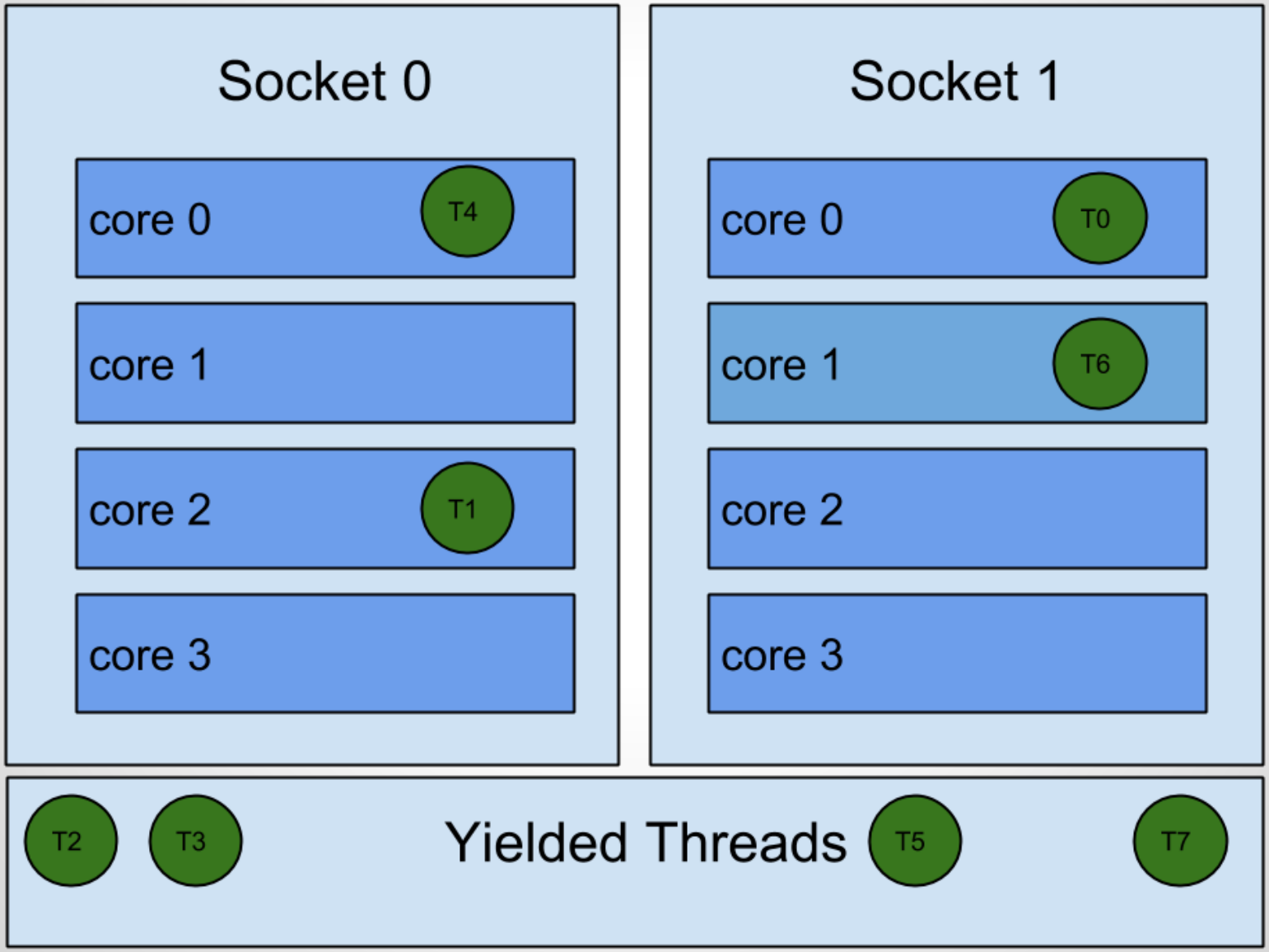
Processor Affinity (CPU Pinning)

- Management of OS thread scheduling
- Stops context switching of threads between cores
- Ideal for continuous looping threads
- For application with little intra-thread communication

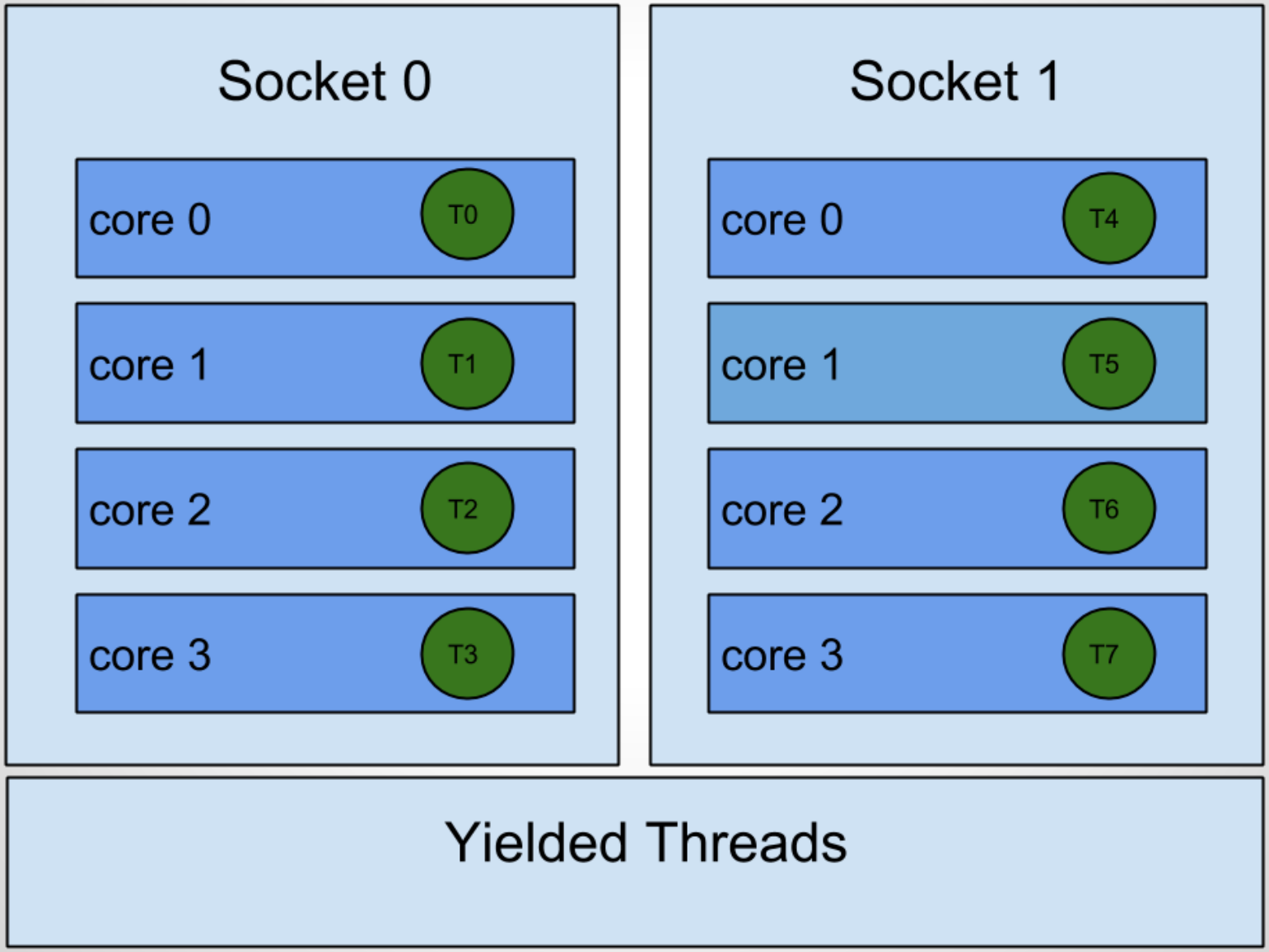
Processor Affinity (CPU Pinning)



Processor Affinity (CPU Pinning)



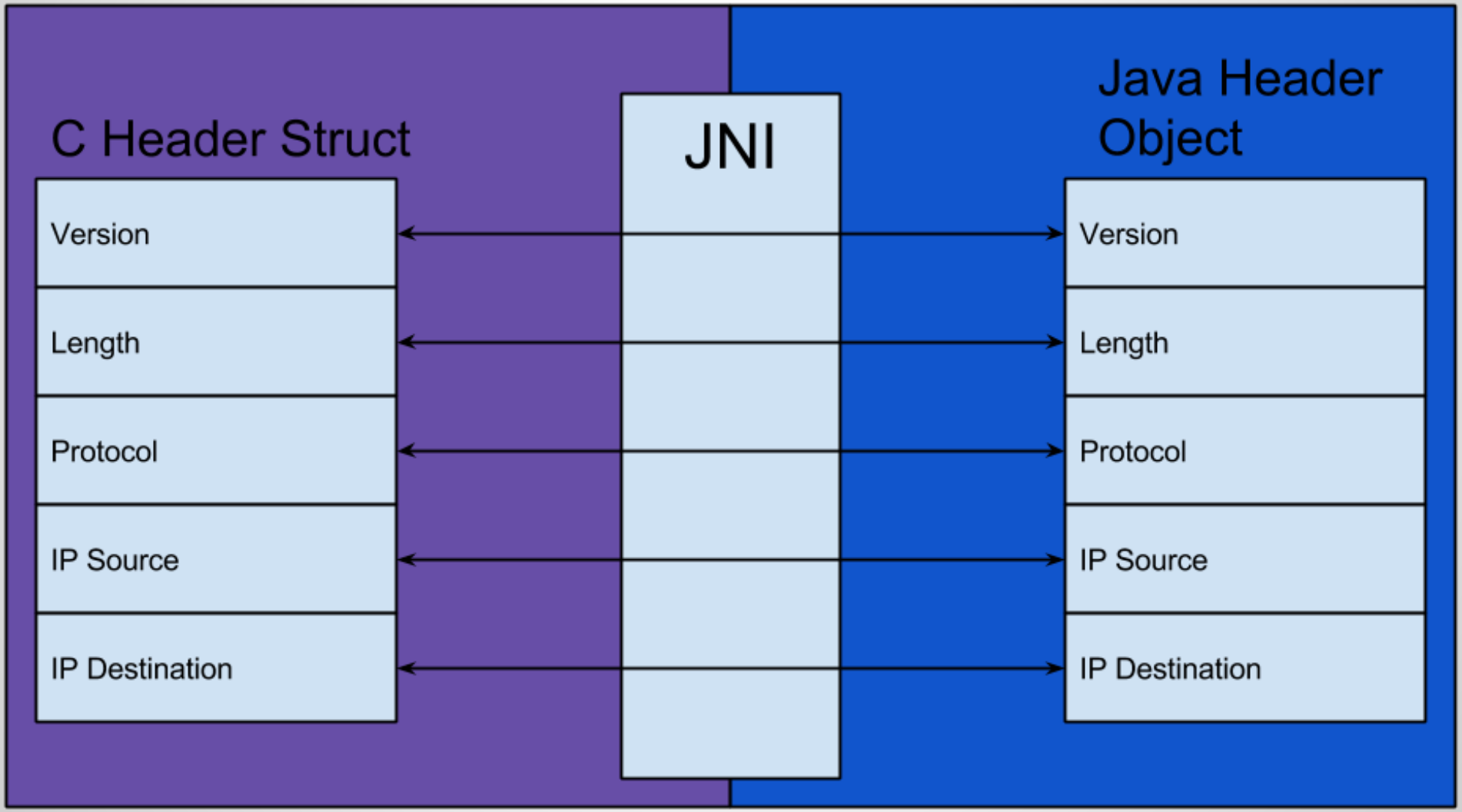
Processor Affinity (CPU Pinning)



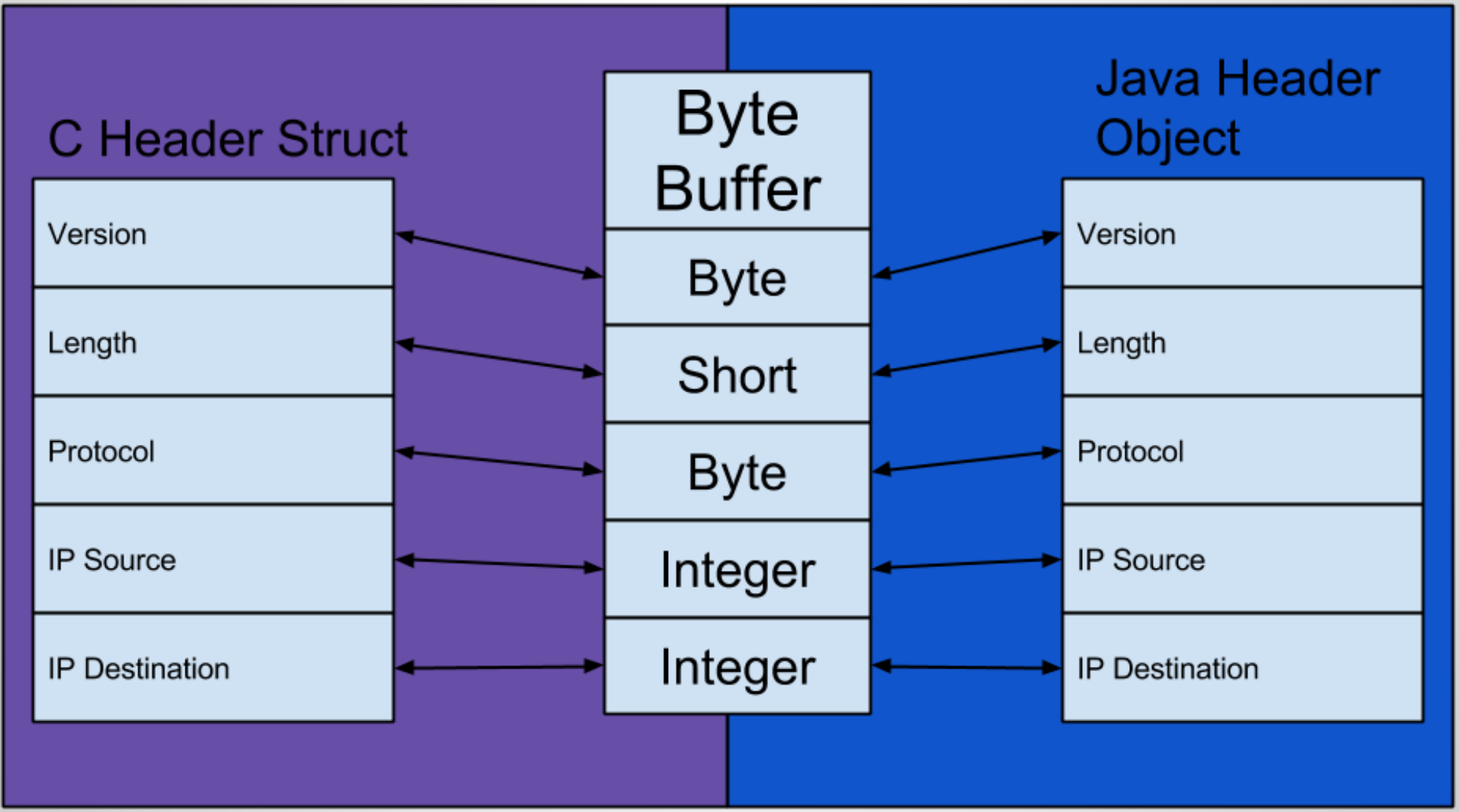
Data Sharing

- Between native and Java code
- Primarily packet meta-data
- Accessing C structs in Java with OO
- 4 possible methods

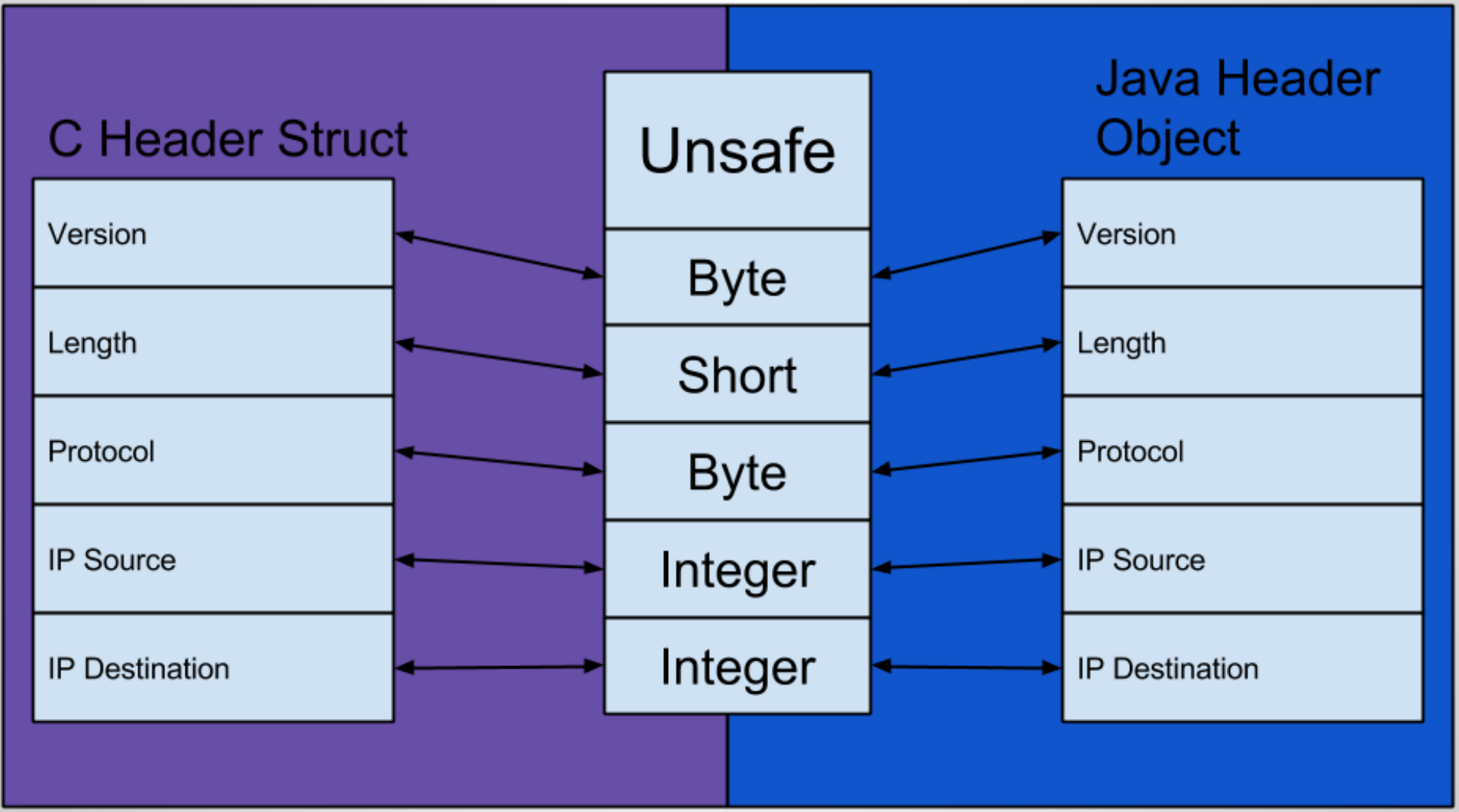
Data Sharing - Object Passing



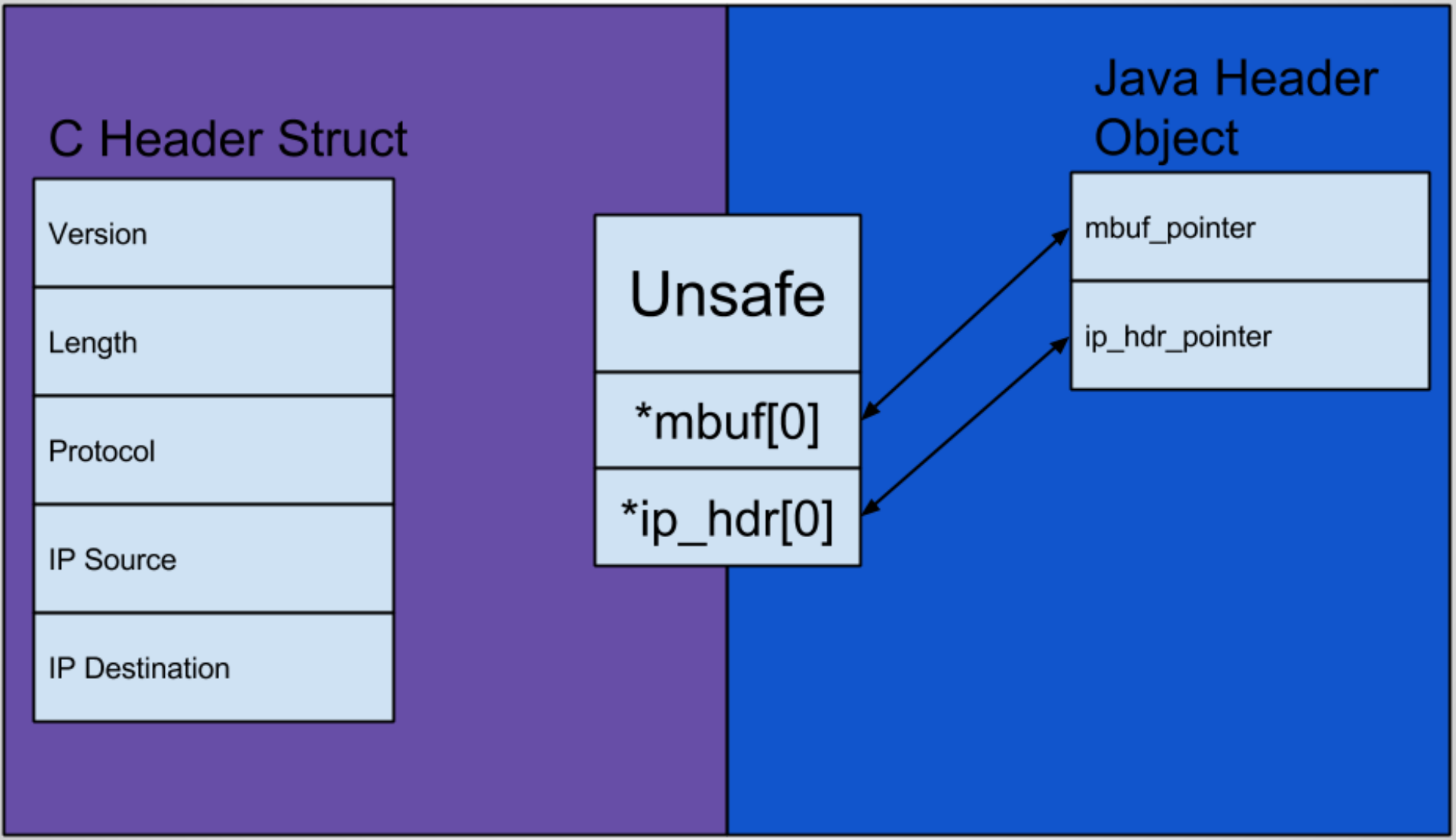
Data Sharing - Byte Buffers

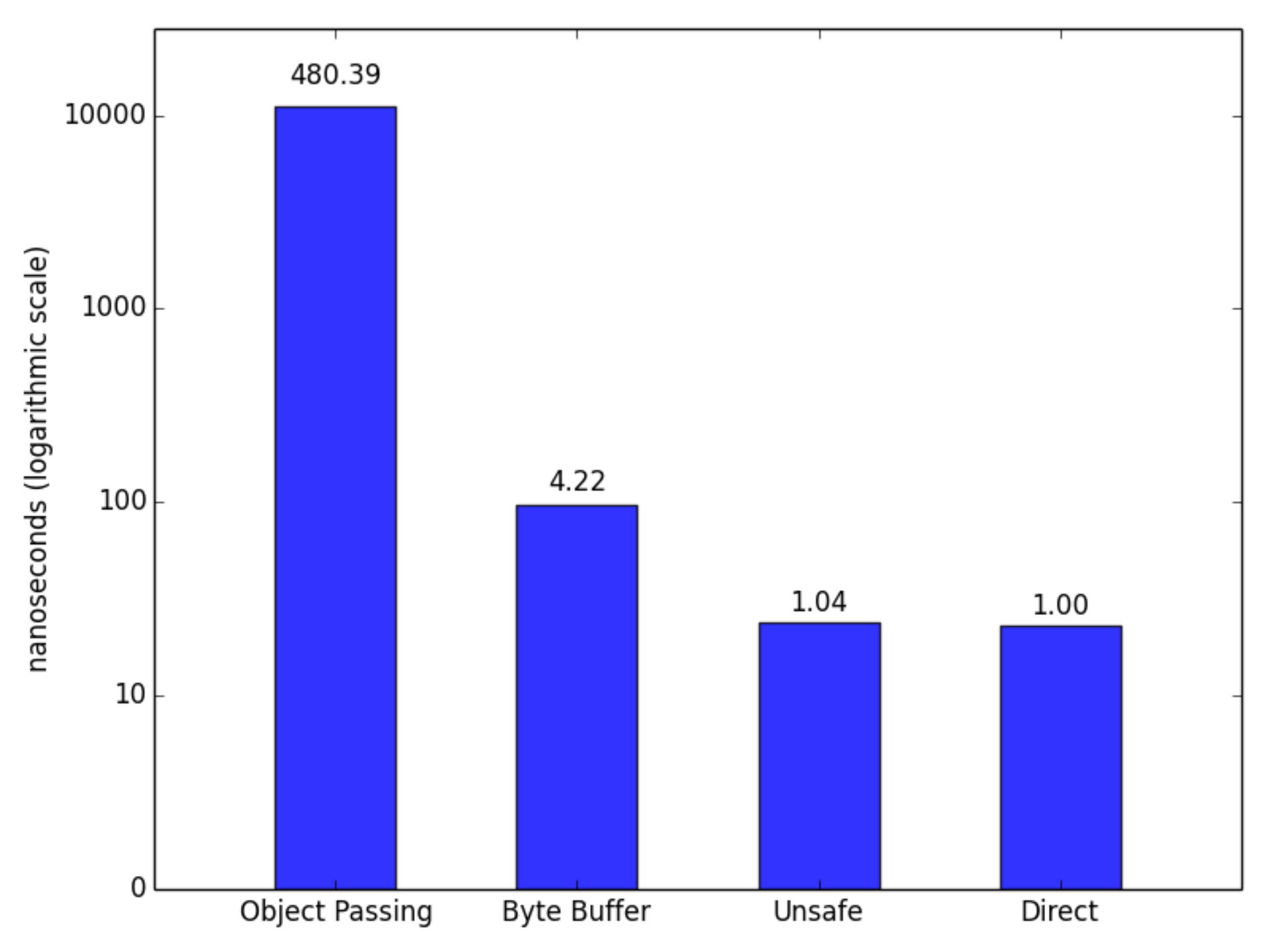


Data Sharing - Java Unsafe

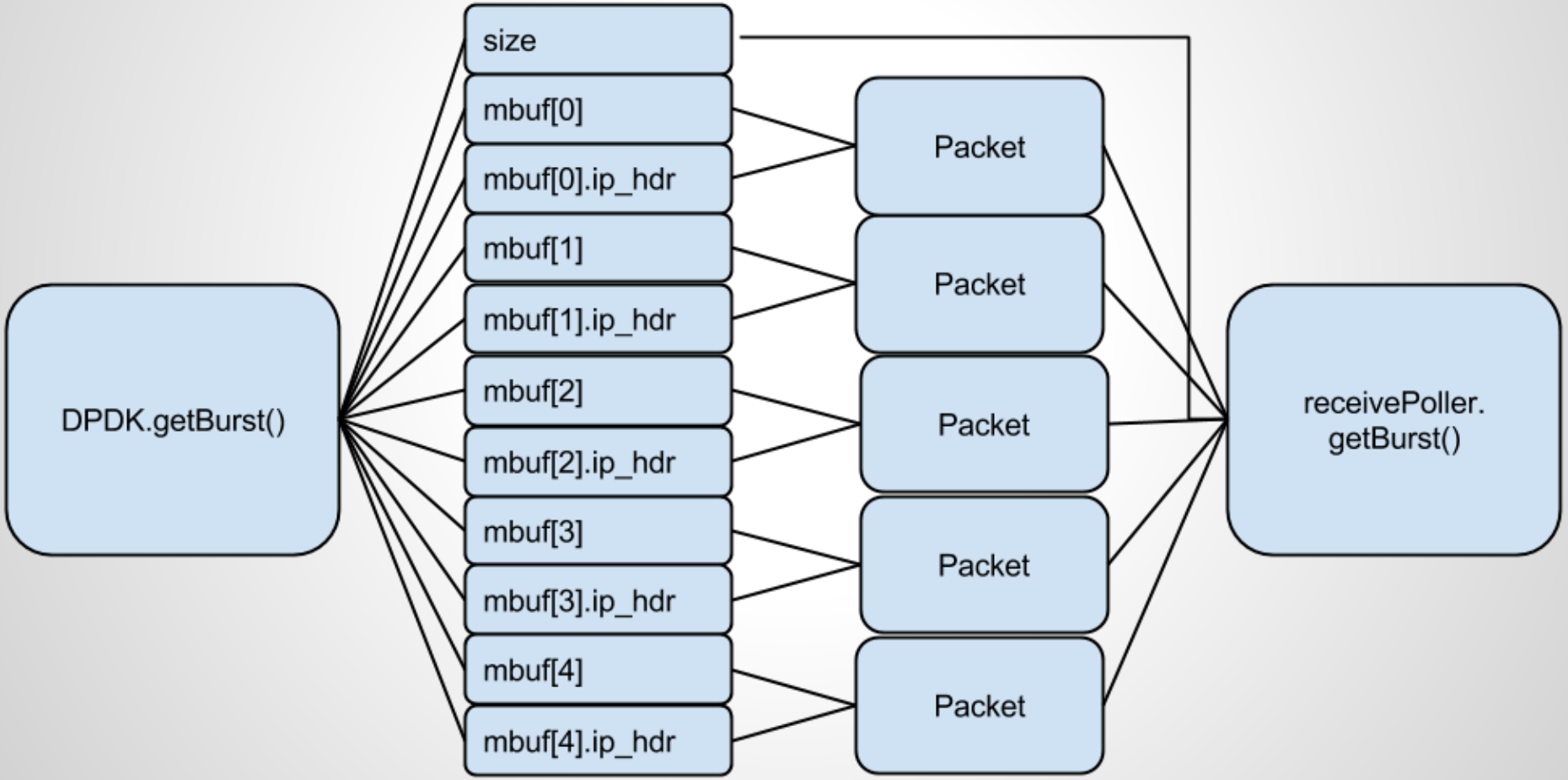


Data Sharing - Direct Access




480.39
4.22
1.04
1.00
10000
1000
100
10
0
nanoseconds (logarithmic scale)
Object Passing
Byte Buffer
Unsafe
Direct

Implementation - Data Sharing



Implementation - Packet

```
public class Packet {

    long mbuf_pointer;
    long hdr_pointer;
    UnsafeAccess ua;

    public long getSrcAddr() {
        ua.setCurrentPointer(packet_pointer + SRC_ADDR_OFFSET);
        return ua.getInt();
    }

    public void setSrcAddr(long src_addr) {
        ua.setCurrentPointer(packet_pointer + SRC_ADDR_OFFSET);
        ua.putInt(src_addr);
    }
}
```

Unsafe Abstraction

- Handles pointer arithmetic
- Conversions between unsigned and signed
- Bound checking for negatives and overflows
- Endianness

C Struct Packing

```
struct foo {
    char *p;          // 8 bytes
    char c;           // 1 byte
    char pad[7];      // 7 bytes
    long x;           // 8 bytes
}
```

C Struct Packing

```
struct __attribute__((packed)) foo {
    char *p;            // 8 bytes
    char c;             // 1 byte
    char pad[7];        // 7 bytes
    long x;             // 8 bytes
}
```

C Struct Packing

Advantages

- Reduced Memory
- Allows for packet member pointer arithmetic
- Data usable over different architectures

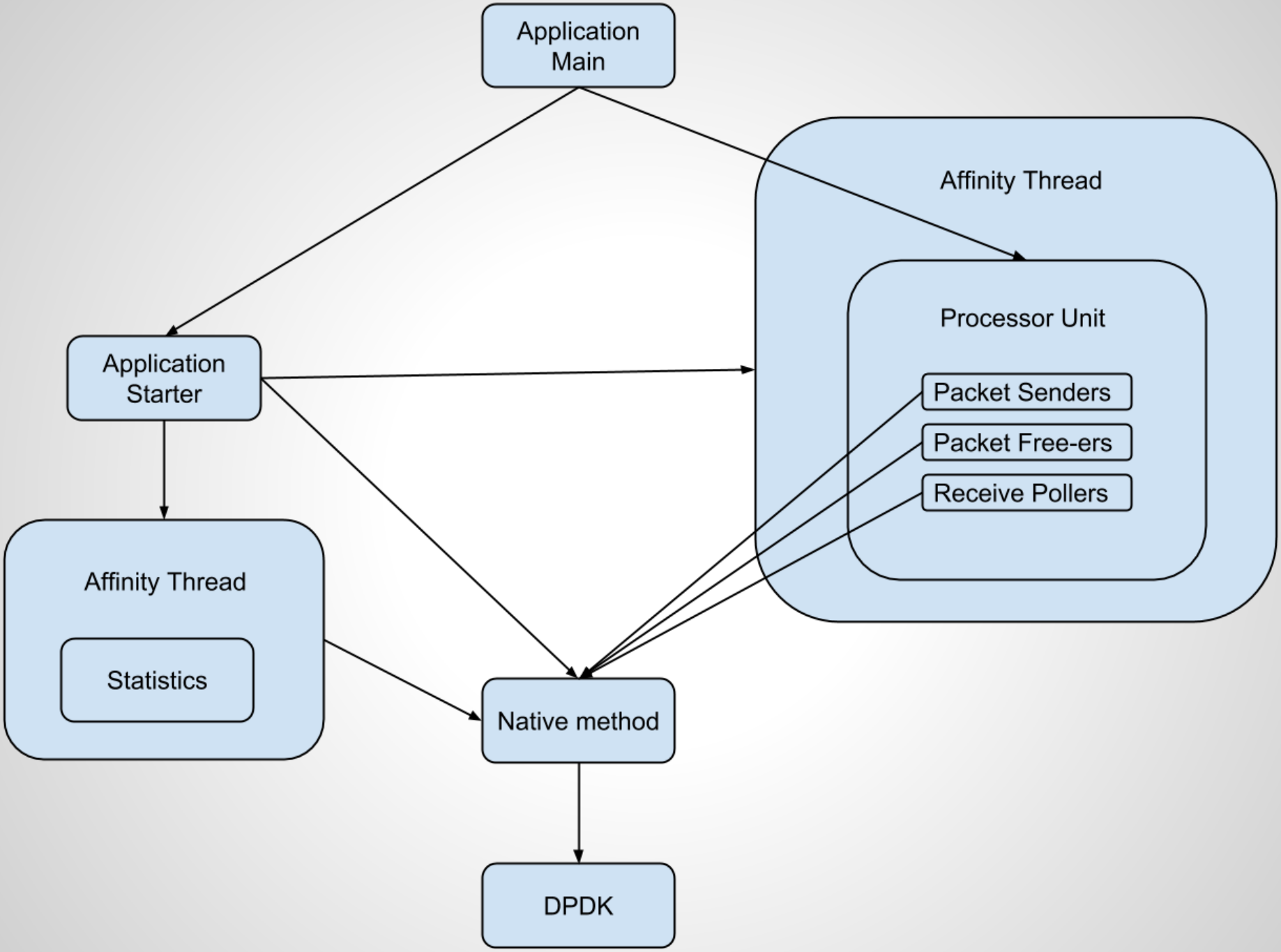
Disadvantages

- Reduced access speed to members

Stat Collecting

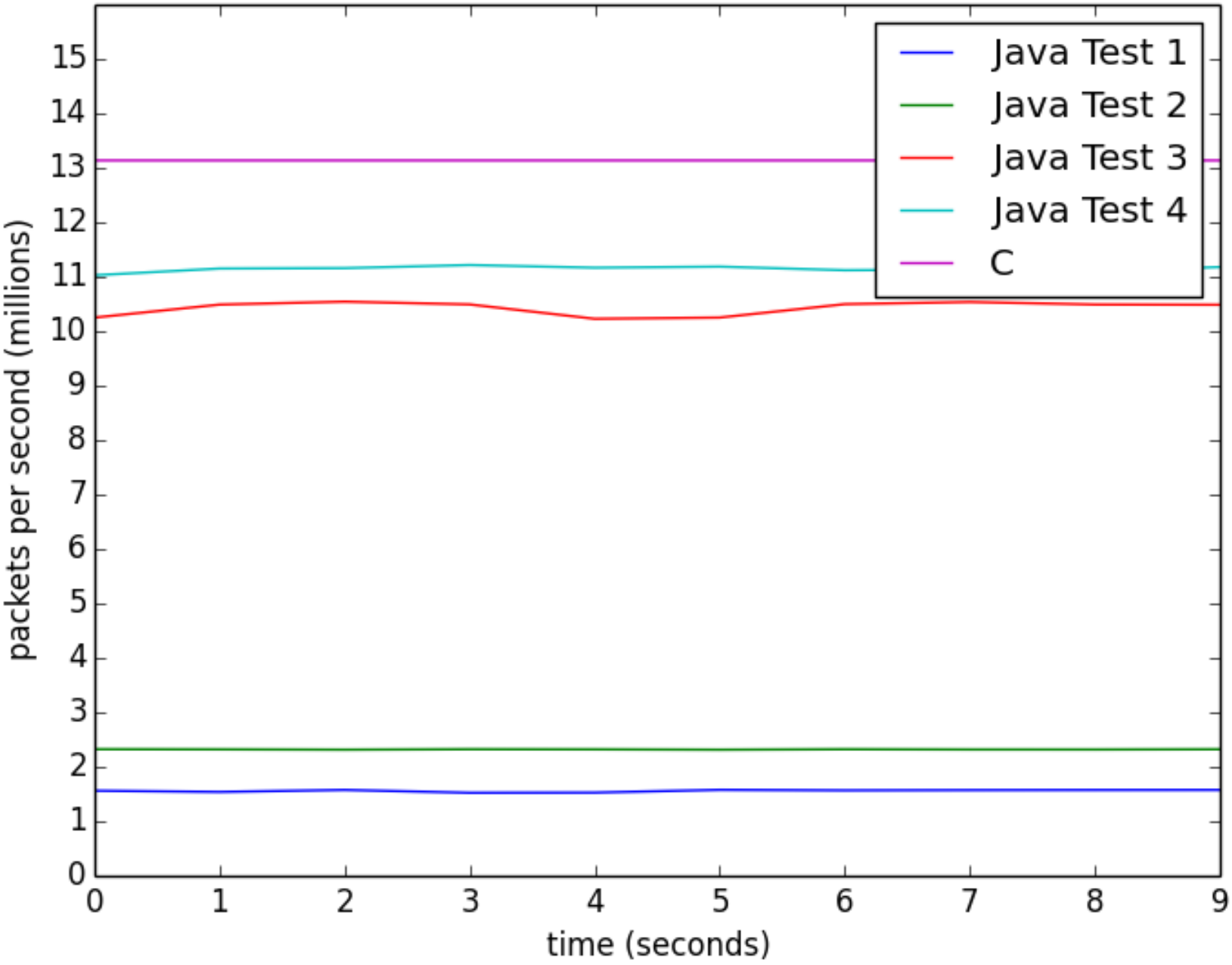
- Needed for performance testing
- 2 methods

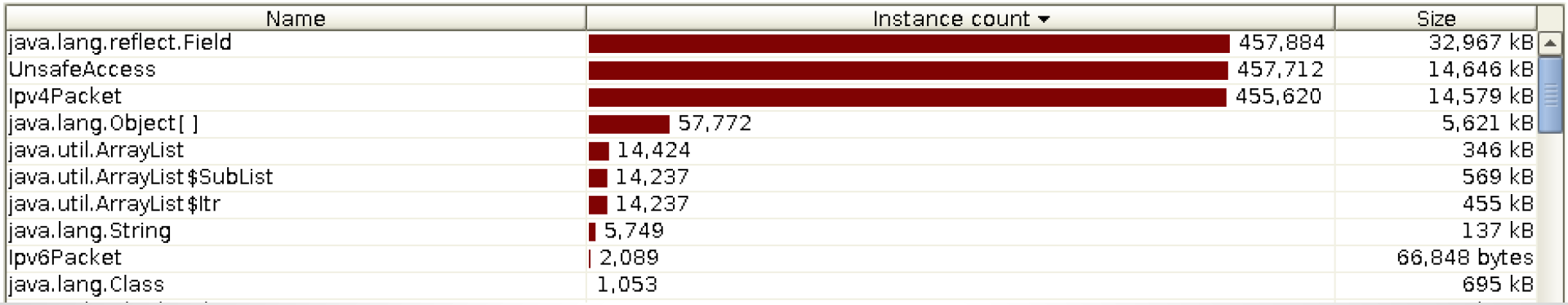
1. Direct from NIC
2. Java side from packet throughput


Application Main
Affinity Thread
Processor Unit
Packet Senders
Packet Free-ers
Receive Pollers
Application Starter
Affinity Thread
Statistics
Native method
DPDK

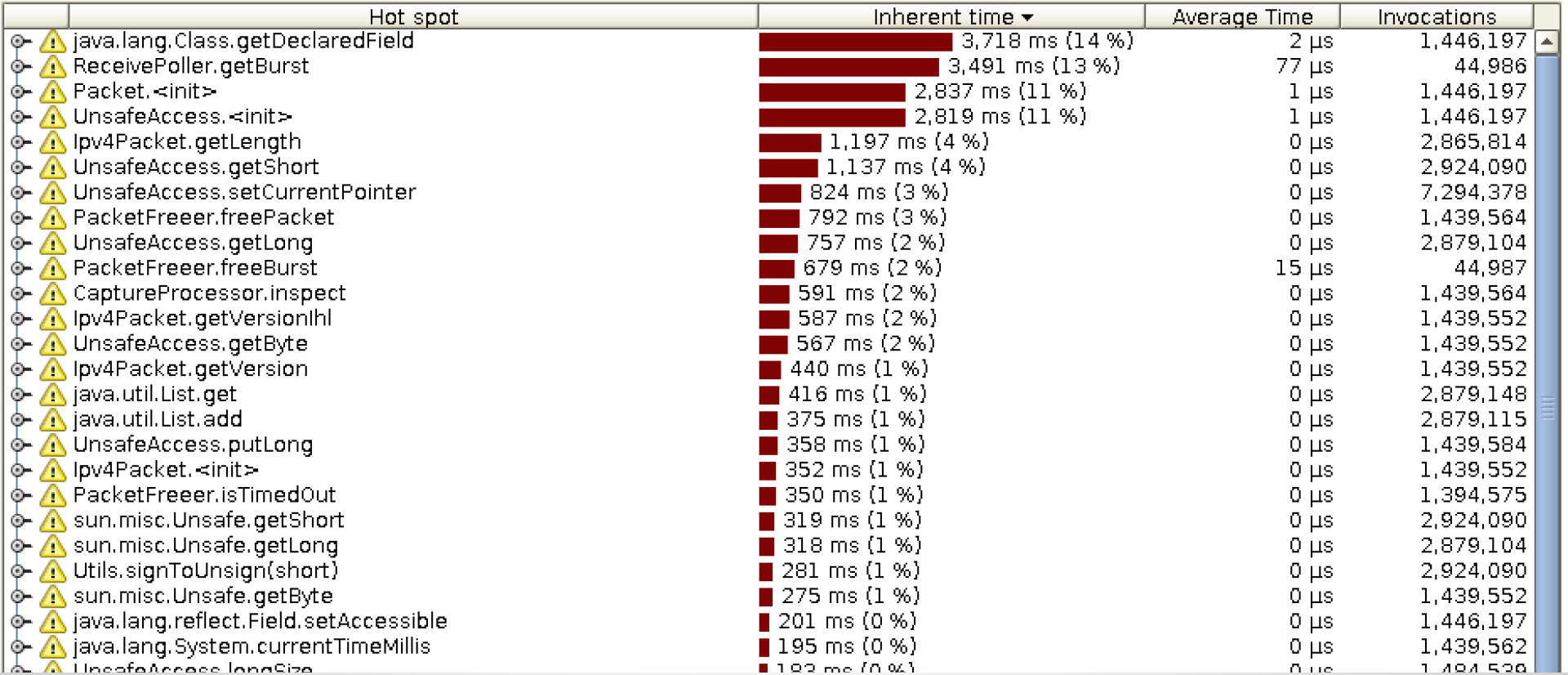
Performance Testing

- Initially on packet capture to test max speeds
- Use Pktgen software
- Compare C & Java implementation
- Further testing on IP firewall

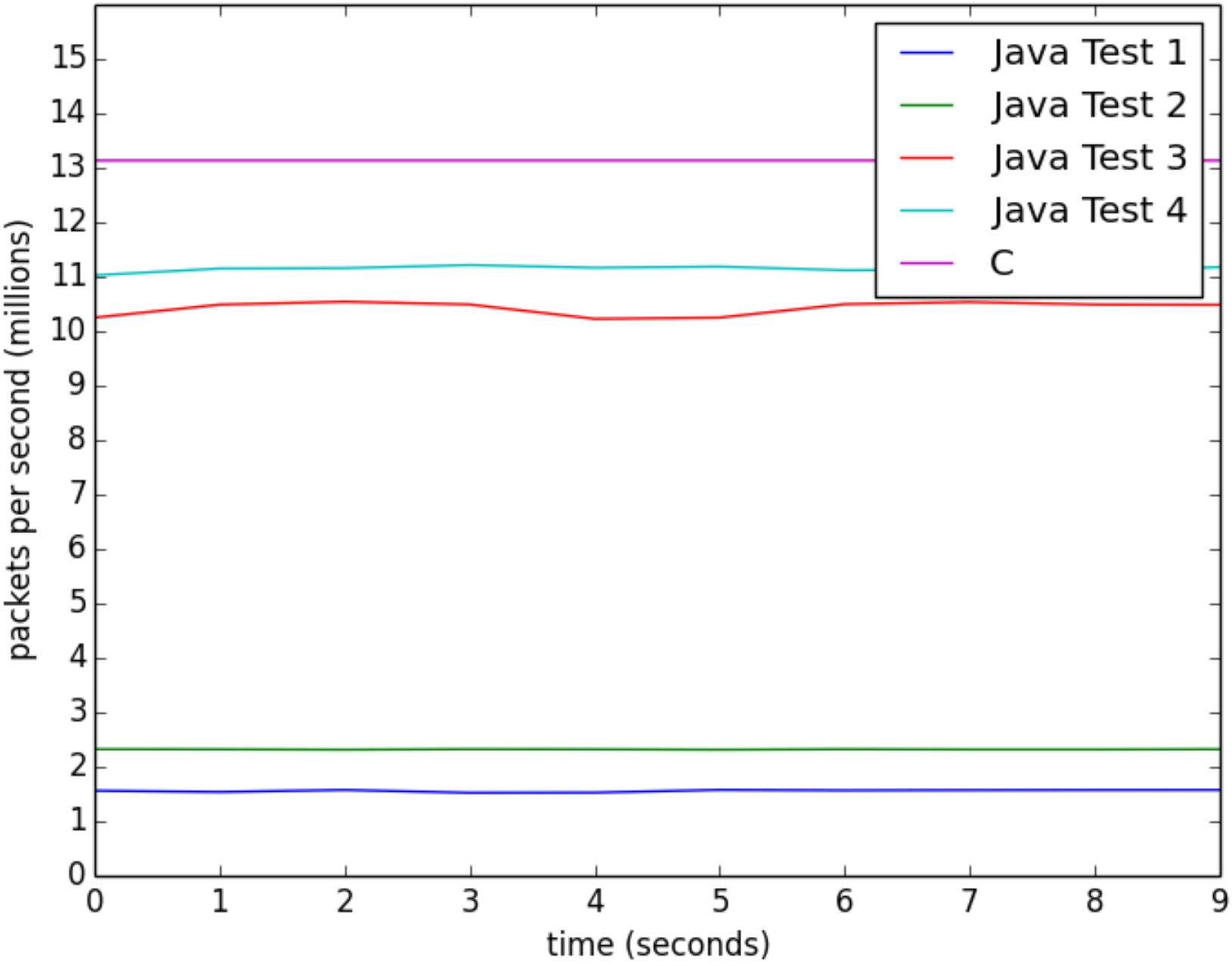

Java Test 1
Java Test 2
Java Test 3
Java Test 4
C
packets per second (millions)
time (seconds)
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

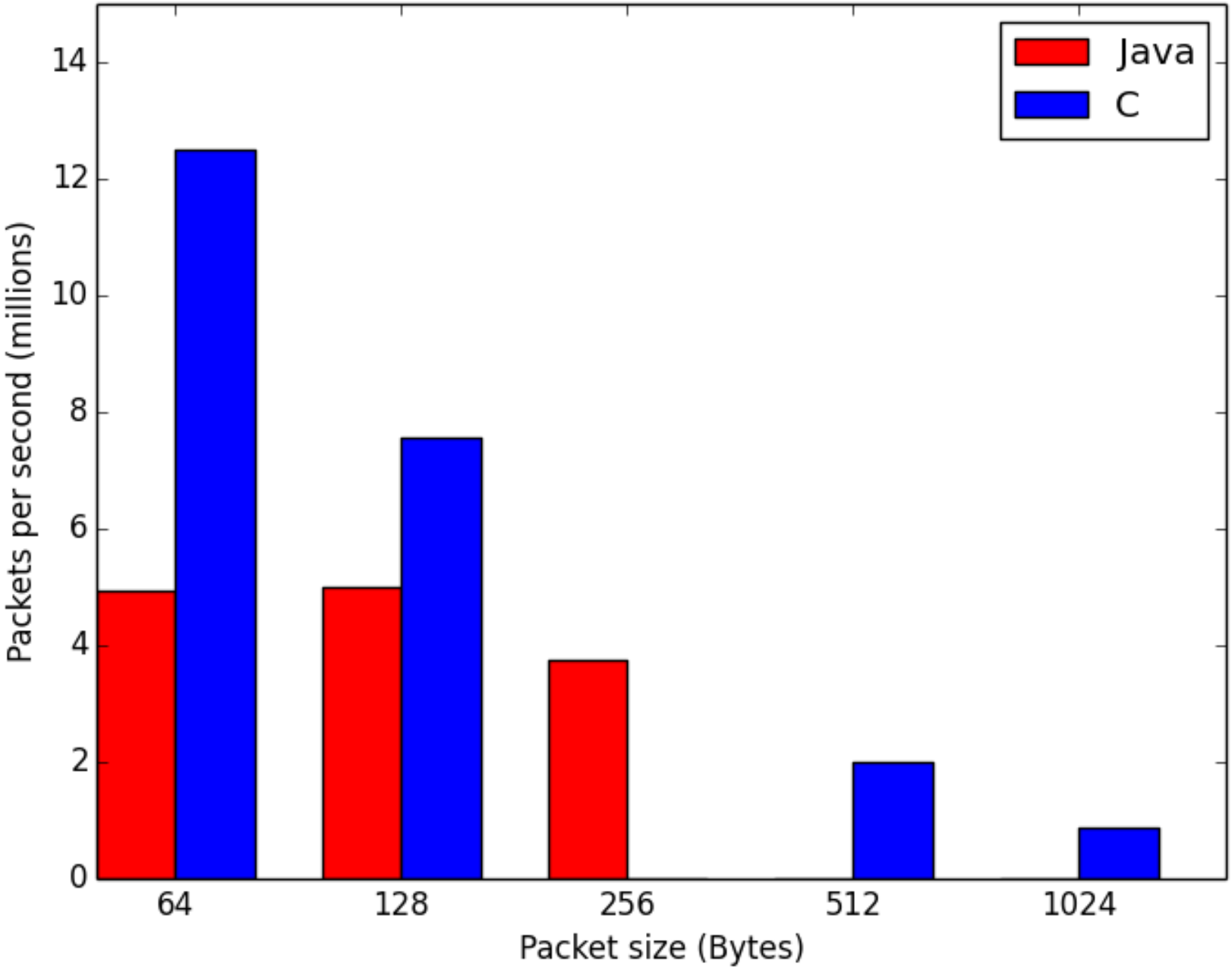


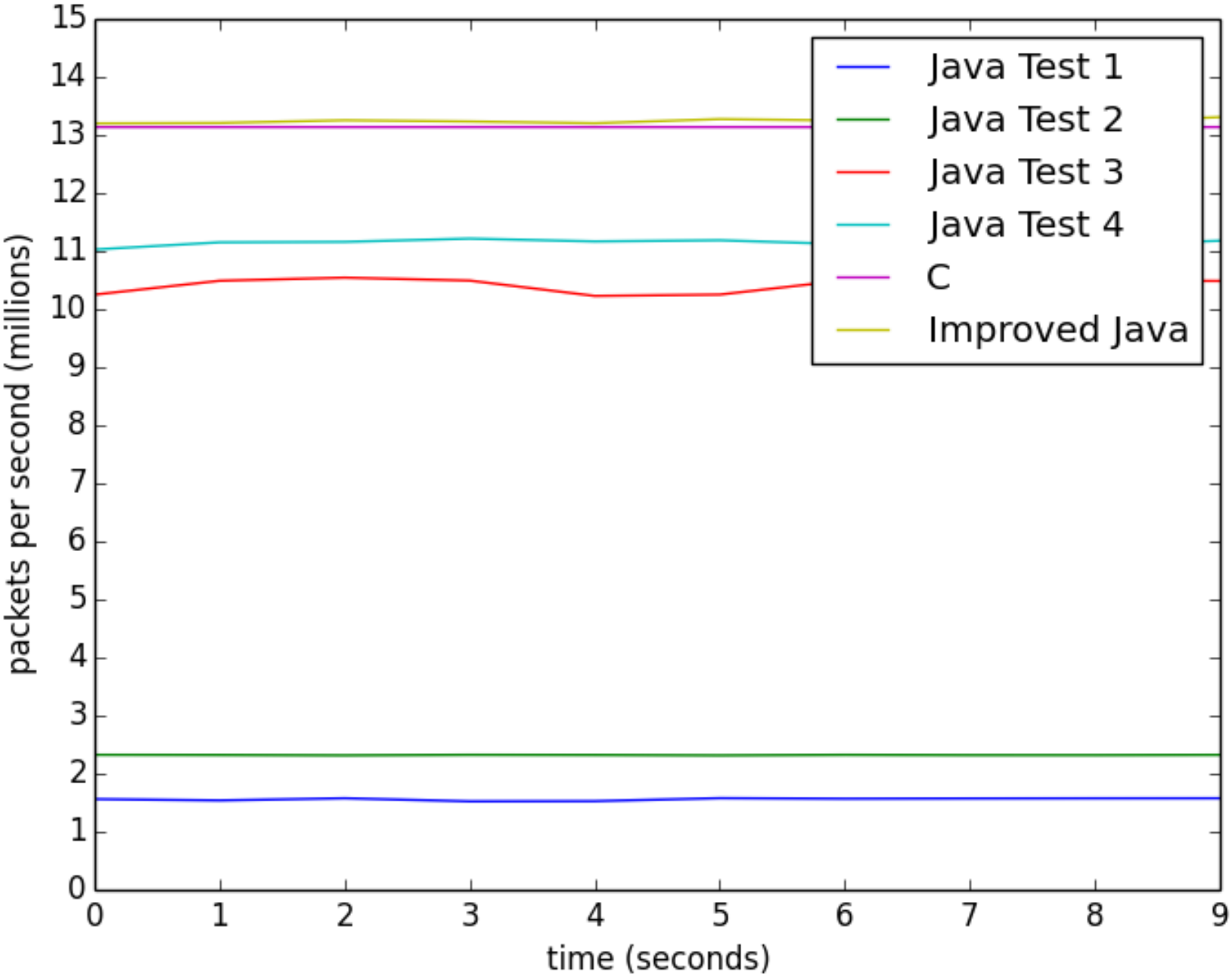
Name	Instance count	Size
java.lang.reflect.Field	457,884	32,967 kB
UnsafeAccess	457,712	14,646 kB
Ipv4Packet	455,620	14,579 kB
java.lang.Object[ ]	57,772	5,621 kB
java.util.ArrayList	14,424	346 kB
java.util.ArrayList$SubList	14,237	569 kB
java.util.ArrayList$Itr	14,237	455 kB
java.lang.String	5,749	137 kB
Ipv6Packet	2,089	66,848 bytes
java.lang.Class	1,053	695 kB

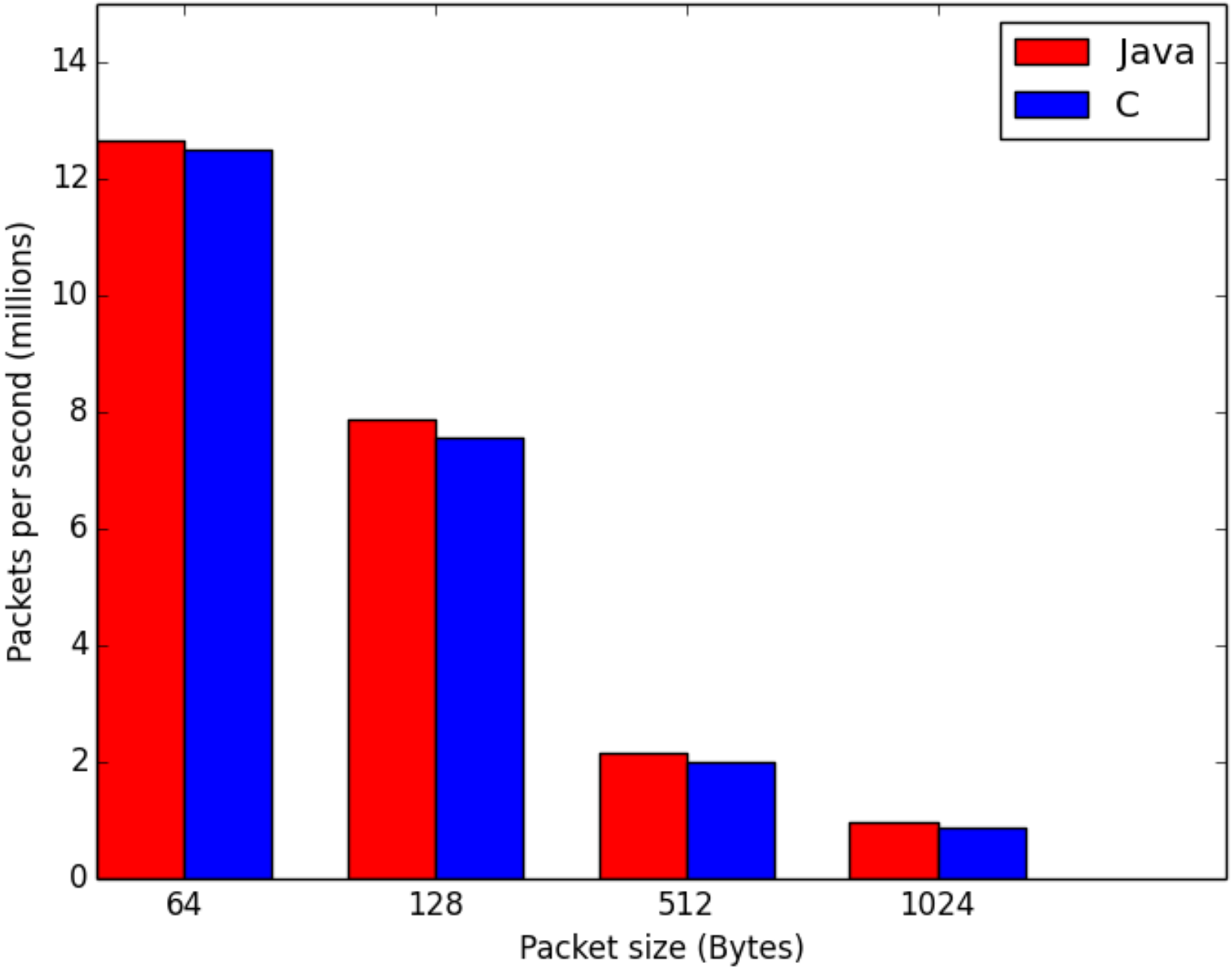


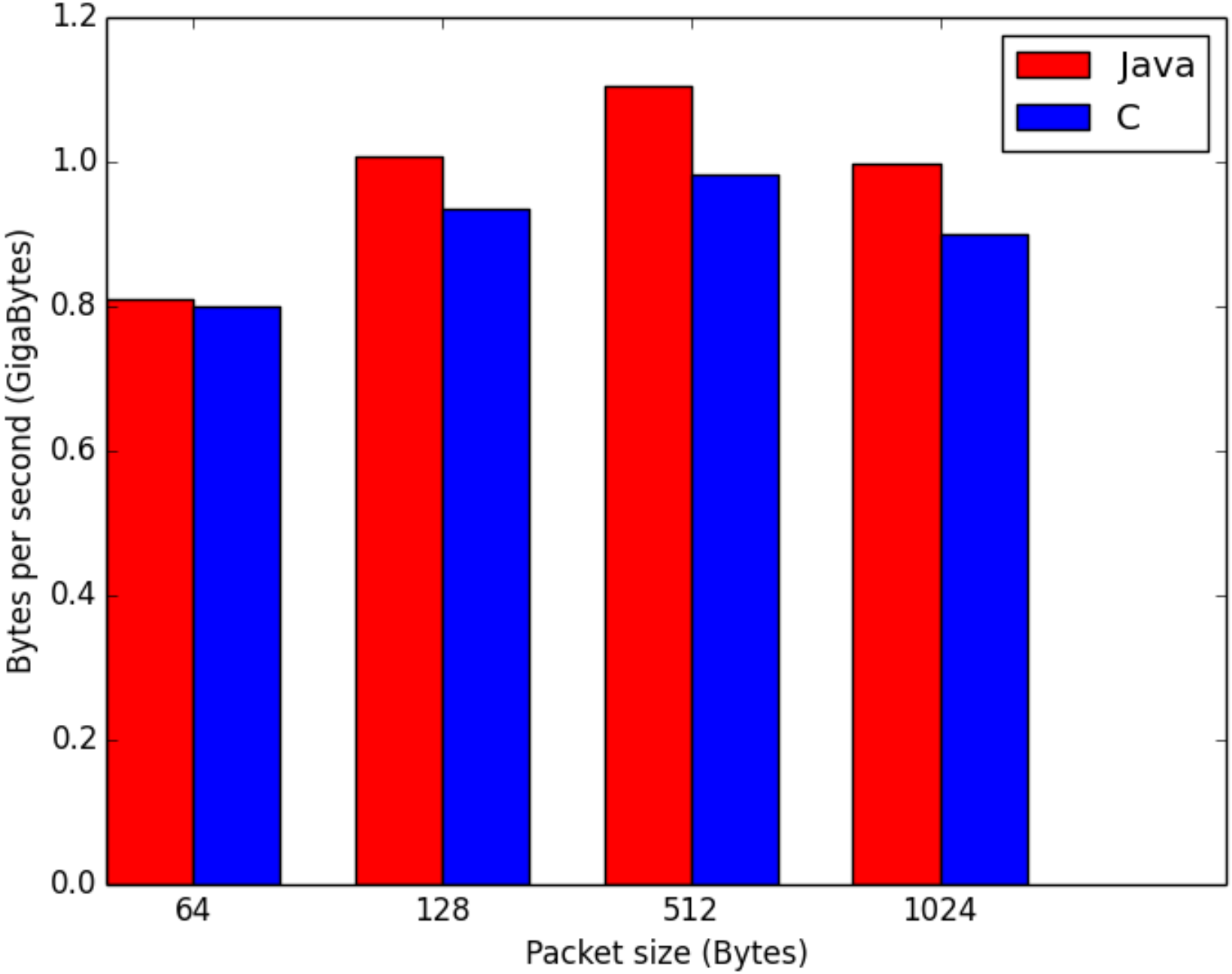
Hot spot	Inherent time	Average Time	Invocations
java.lang.Class.getDeclaredField	3,718 ms (14 %)	2 µs	1,446,197
ReceivePoller.getBurst	3,491 ms (13 %)	77 µs	44,986
Packet.<init>	2,837 ms (11 %)	1 µs	1,446,197
UnsafeAccess.<init>	2,819 ms (11 %)	1 µs	1,446,197
Ipv4Packet.getLength	1,197 ms (4 %)	0 µs	2,865,814
UnsafeAccess.getShort	1,137 ms (4 %)	0 µs	2,924,090
UnsafeAccess.setCurrentPointer	824 ms (3 %)	0 µs	7,294,378
PacketFreeer.freePacket	792 ms (3 %)	0 µs	1,439,564
UnsafeAccess.getLong	757 ms (2 %)	0 µs	2,879,104
PacketFreeer.freeBurst	679 ms (2 %)	15 µs	44,987
CaptureProcessor.inspect	591 ms (2 %)	0 µs	1,439,564
Ipv4Packet.getVersionIhl	587 ms (2 %)	0 µs	1,439,552
UnsafeAccess.getByte	567 ms (2 %)	0 µs	1,439,552
Ipv4Packet.getVersion	440 ms (1 %)	0 µs	1,439,552
java.util.List.get	416 ms (1 %)	0 µs	2,879,148
java.util.List.add	375 ms (1 %)	0 µs	2,879,115
UnsafeAccess.putLong	358 ms (1 %)	0 µs	1,439,584
Ipv4Packet.<init>	352 ms (1 %)	0 µs	1,439,552
PacketFreeer.isTimedOut	350 ms (1 %)	0 µs	1,394,575
sun.misc.Unsafe.getShort	319 ms (1 %)	0 µs	2,924,090
sun.misc.Unsafe.getLong	318 ms (1 %)	0 µs	2,879,104
Utils.signToUnsign(short)	281 ms (1 %)	0 µs	2,924,090
sun.misc.Unsafe.getByte	275 ms (1 %)	0 µs	1,439,552
java.lang.reflect.Field.setAccessible	201 ms (0 %)	0 µs	1,446,197
java.lang.System.currentTimeMillis	195 ms (0 %)	0 µs	1,439,562
UnsafeAccess.longSize	183 ms (0 %)	0 µs	1,484,539


Java Test 1
Java Test 2
Java Test 3
Java Test 4
C
packets per second (millions)
time (seconds)
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


Java
C
Packets per second (millions)
0
2
4
6
8
10
12
14
64
128
256
512
1024
Packet size (Bytes)


packets per second (millions)
time (seconds)
Java Test 1
Java Test 2
Java Test 3
Java Test 4
C
Improved Java
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


Java
C
Packets per second (millions)
0
2
4
6
8
10
12
14
64
128
512
1024
Packet size (Bytes)


1.2
1.0
0.8
0.6
0.4
0.2
0.0
Bytes per second (GigaBytes)
Java
C
64
128
512
1024
Packet size (Bytes)

What's Next?

- Test with pipeline model
- Multiple cores
- Full-scale application in system
- Add further DPDK features to DPDK-Java

Thanks For Listening!

Any Questions?