Network stack specialization for performance

Ilias Marinos[§], Robert N.M. Watson[§], Mark Handley*

[§] *University of Cambridge, * University College London*

Motivation

Providers are scaling out rapidly. Key aspects:

- ~~1 machine:N functions~~ → N machines:1 function
- Performance is critical
- Scalability on multicore systems
- Cost & energy concerns

Motivation

Providers are scaling out rapidly. Key aspects:

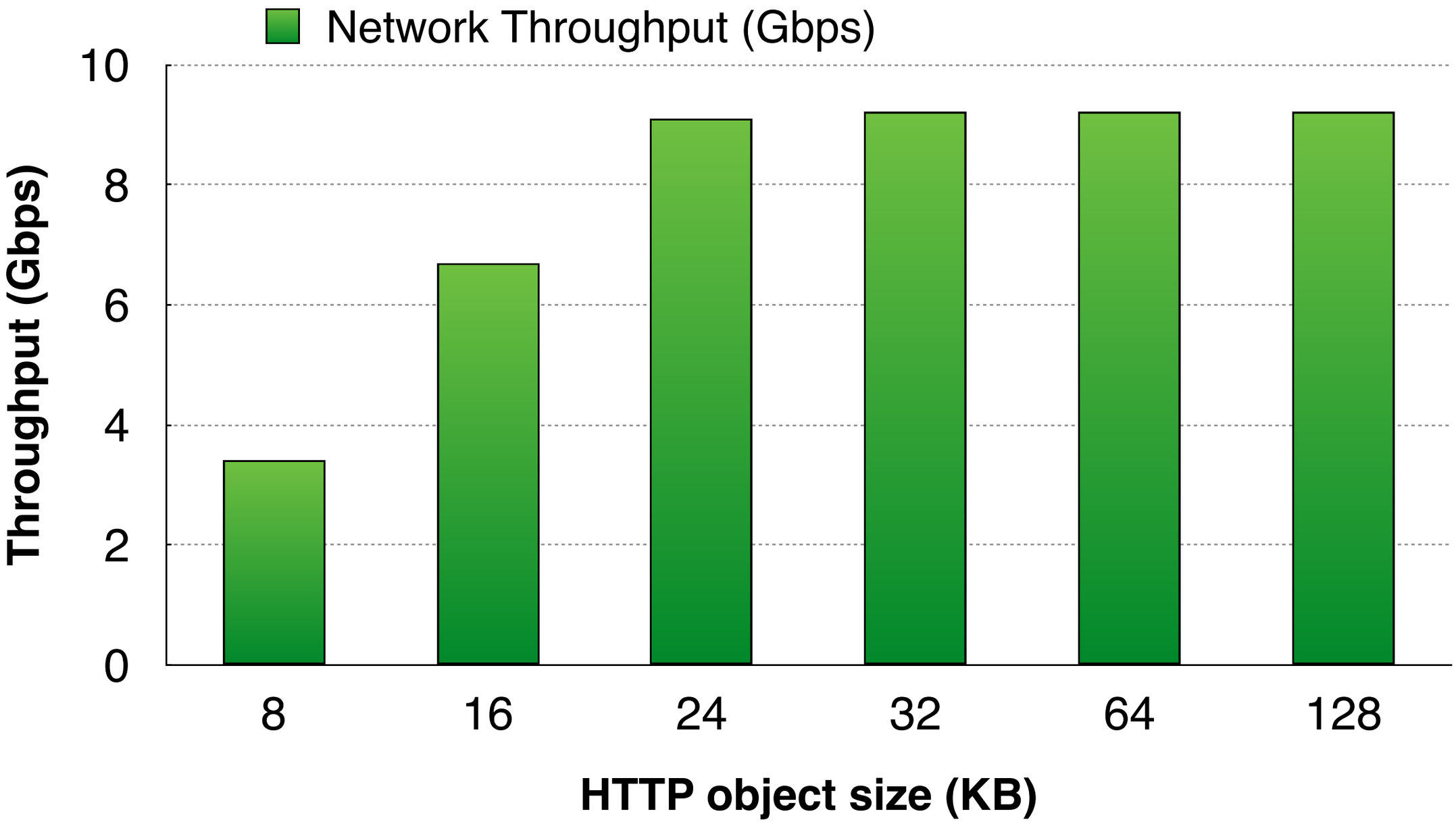
- ~~1 machine:N functions~~ → N machines:1 function
- Performance is critical
- Scalability on multicore systems
- Cost & energy concerns

Are general-purpose stacks the right solution for that kind of role?

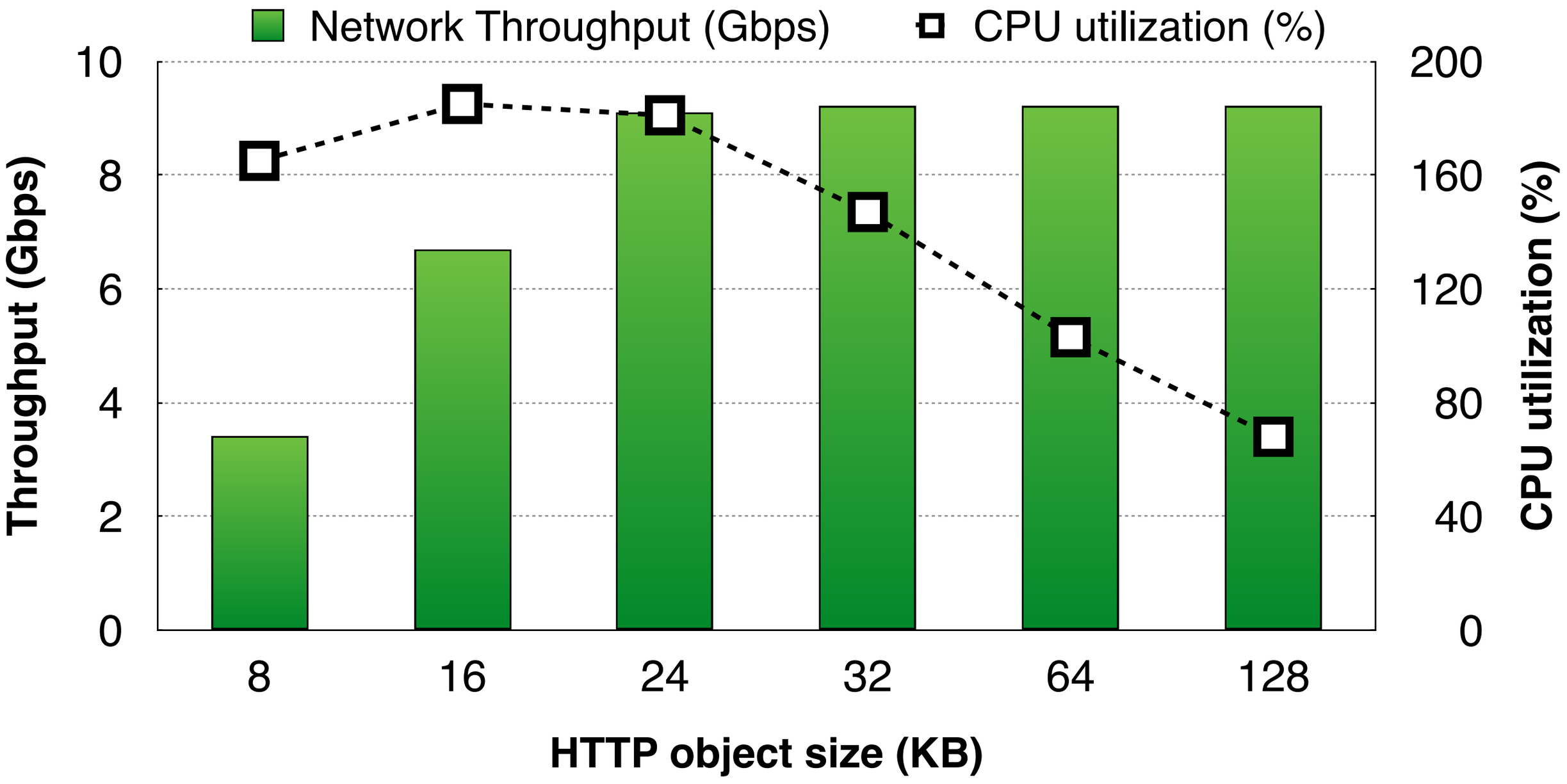
The Problem

- Conventional stacks are great for bulk transfers, but what about short ones?

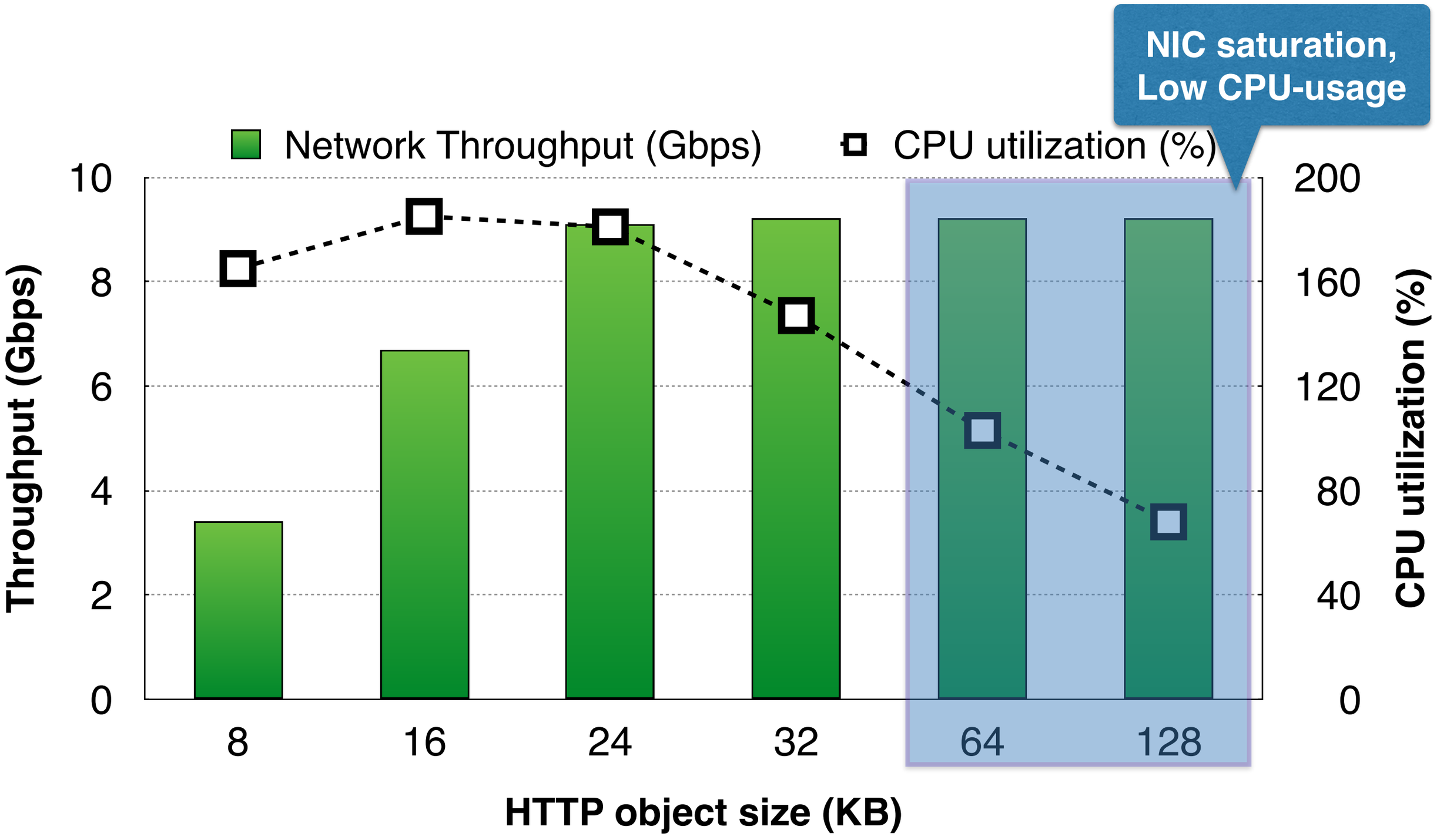
The Problem


Network Throughput (Gbps)
Throughput (Gbps)
10
8
6
4
2
0
8
16
24
32
64
128
HTTP object size (KB)

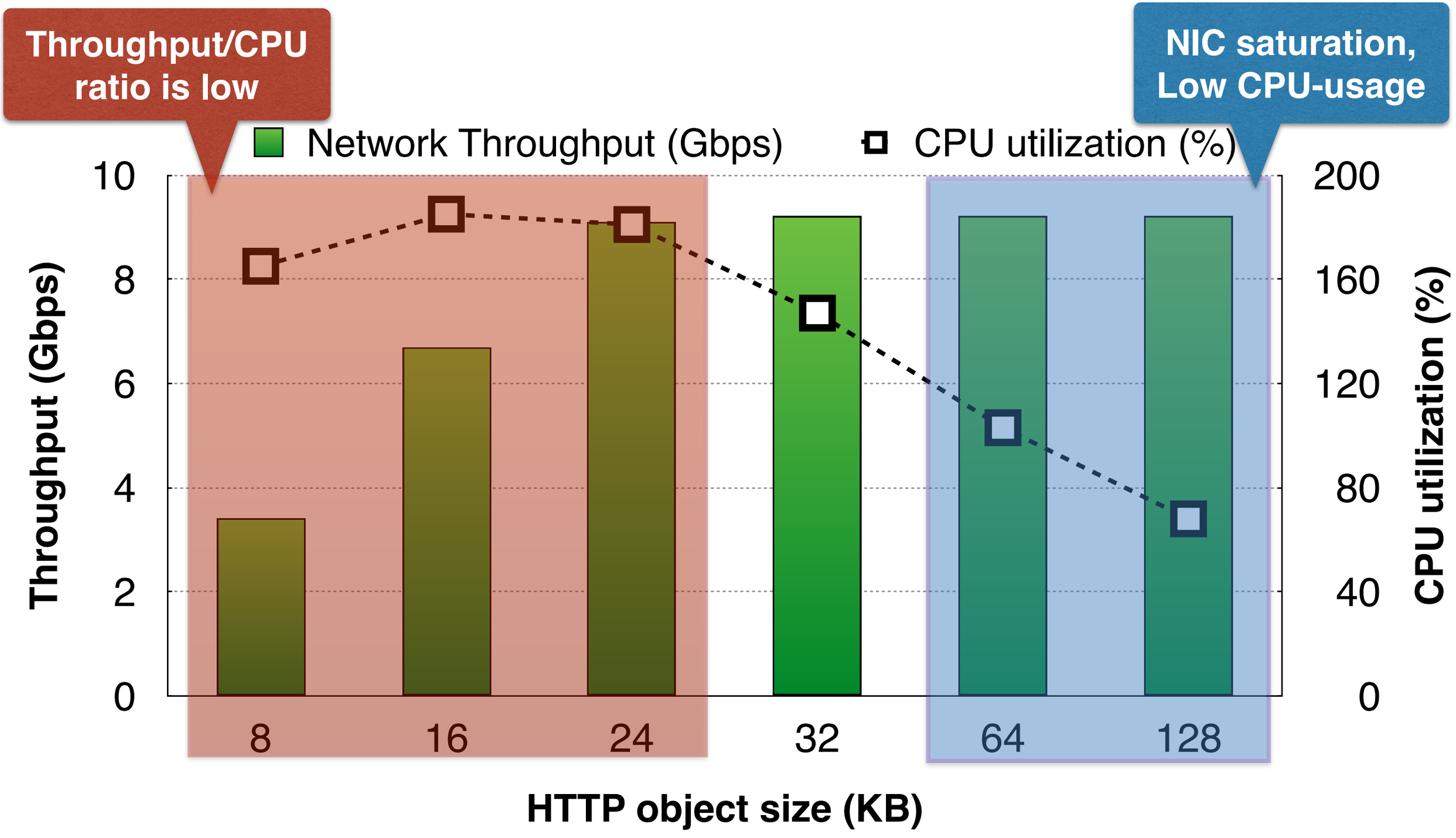
The Problem


Network Throughput (Gbps)
CPU utilization (%)
Throughput (Gbps)
CPU utilization (%)
0
2
4
6
8
10
0
40
80
120
160
200
8
16
24
32
64
128
HTTP object size (KB)

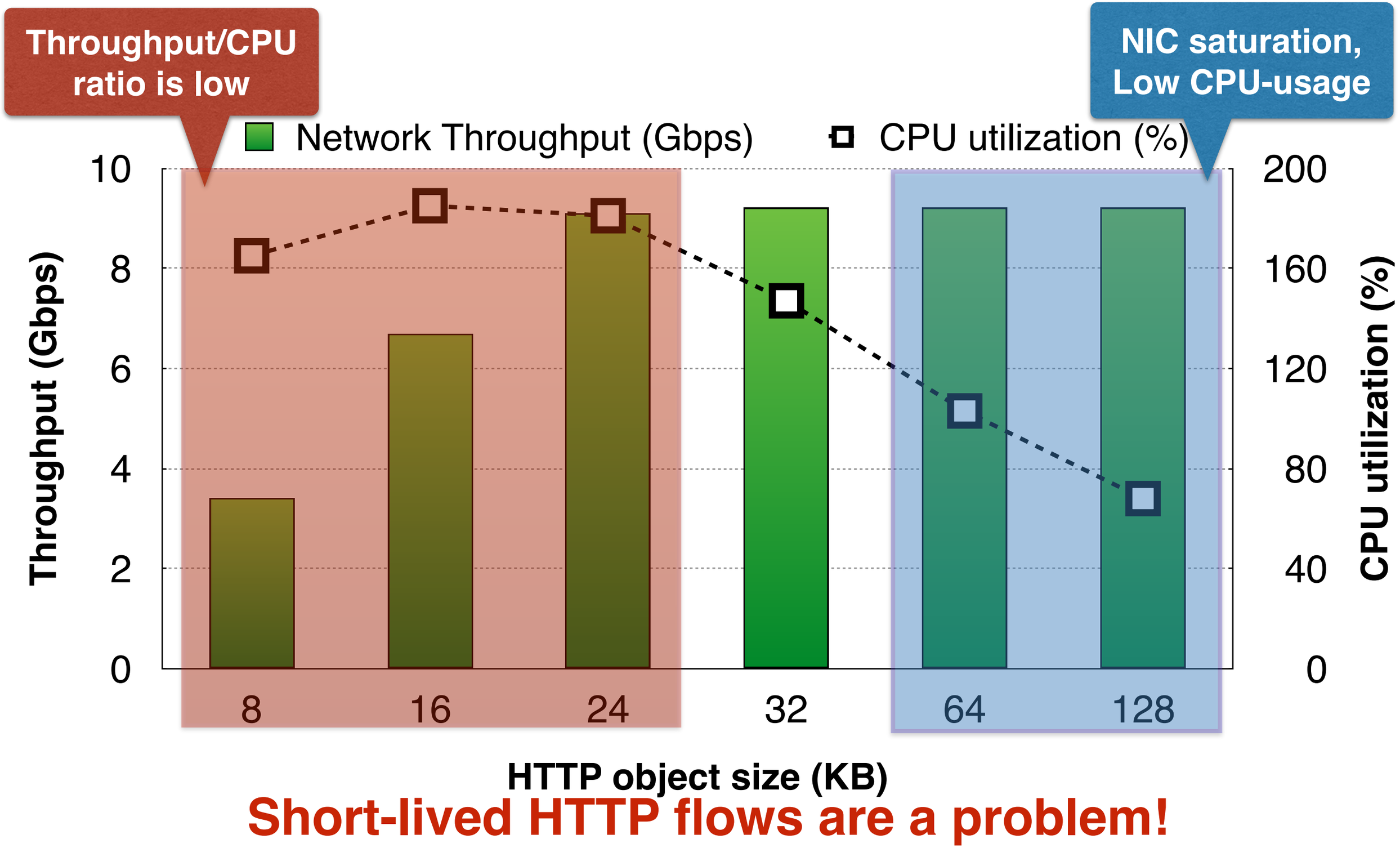
The Problem


NIC saturation,
Low CPU-usage
Network Throughput (Gbps)
CPU utilization (%)
Throughput (Gbps)
CPU utilization (%)
0
2
4
6
8
10
0
40
80
120
160
200
8
16
24
32
64
128
HTTP object size (KB)

The Problem


Throughput/CPU ratio is low
NIC saturation, Low CPU-usage
Network Throughput (Gbps)
CPU utilization (%)
Throughput (Gbps)
CPU utilization (%)
0
2
4
6
8
10
0
40
80
120
160
200
8
16
24
32
64
128
HTTP object size (KB)

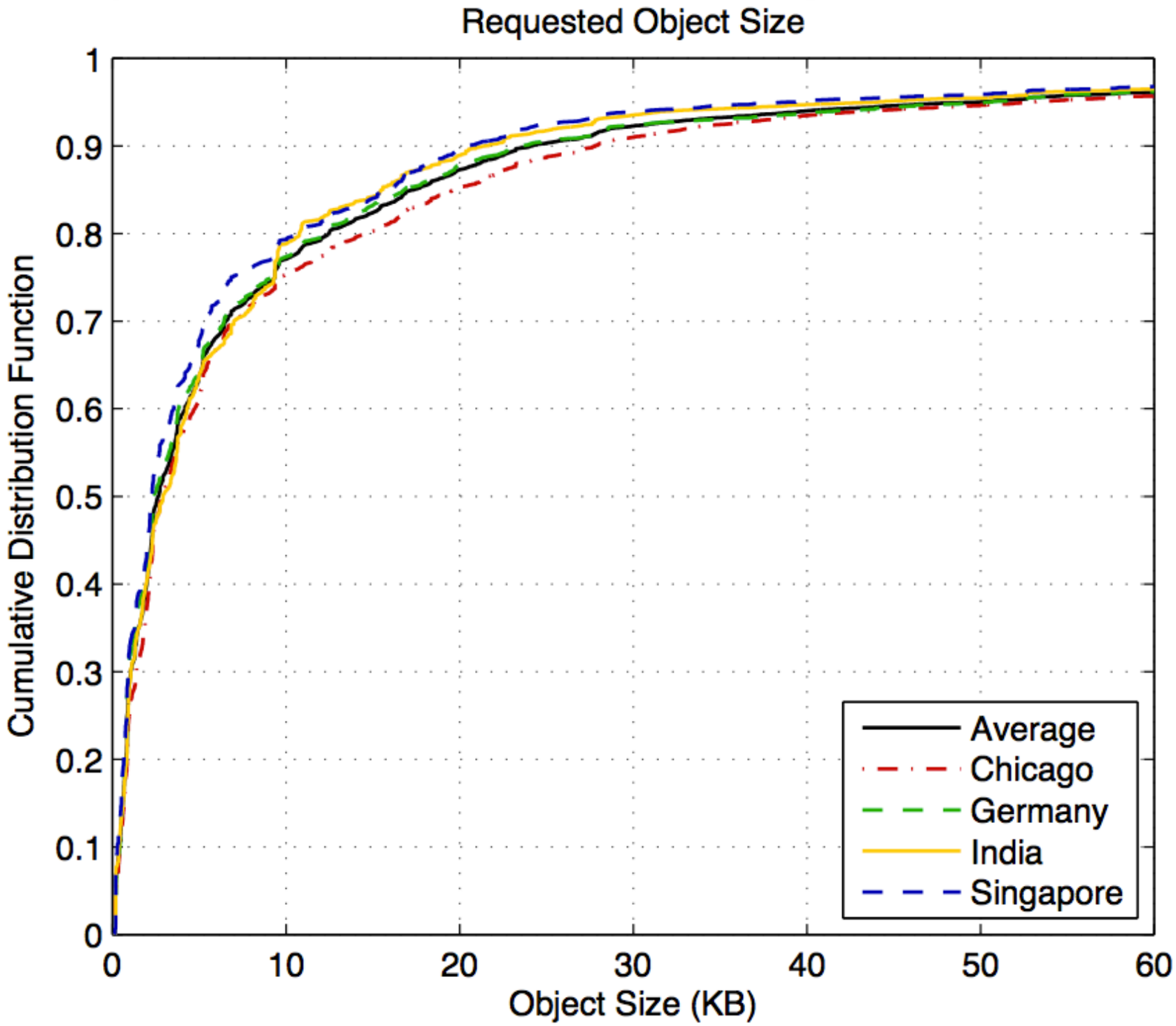
The Problem



Short-lived HTTP flows are a problem!

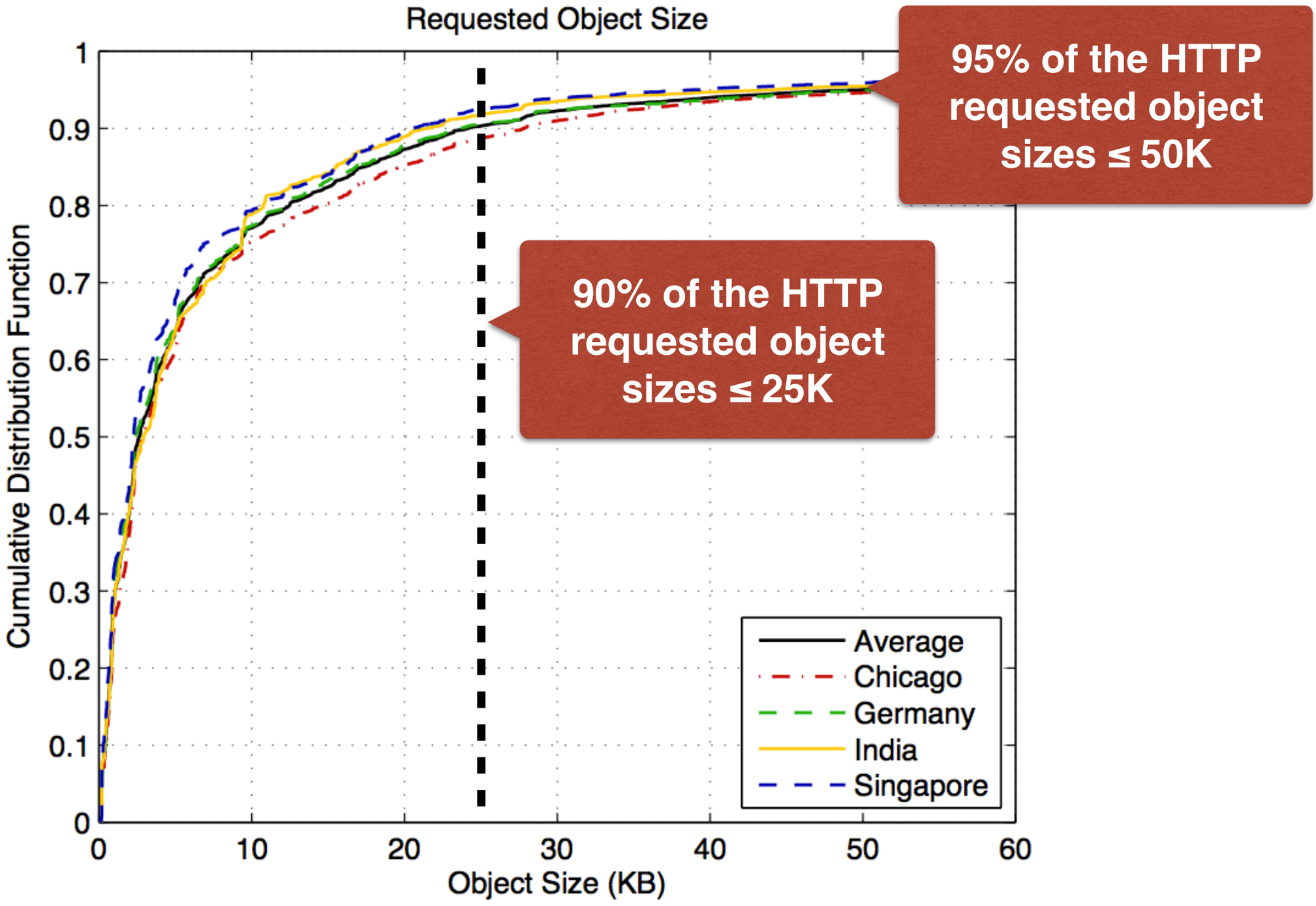
Why is this important?

Why is this important?



Distribution based on traces from Yahoo! CDN [Al-Fares et'al 2011]

Why is this important?



Distribution based on traces from Yahoo! CDN [Al-Fares et'al 2011]

Design Goals

Design a network stack that:

- Allows transparent flow of memory from NIC to the application and vice versa

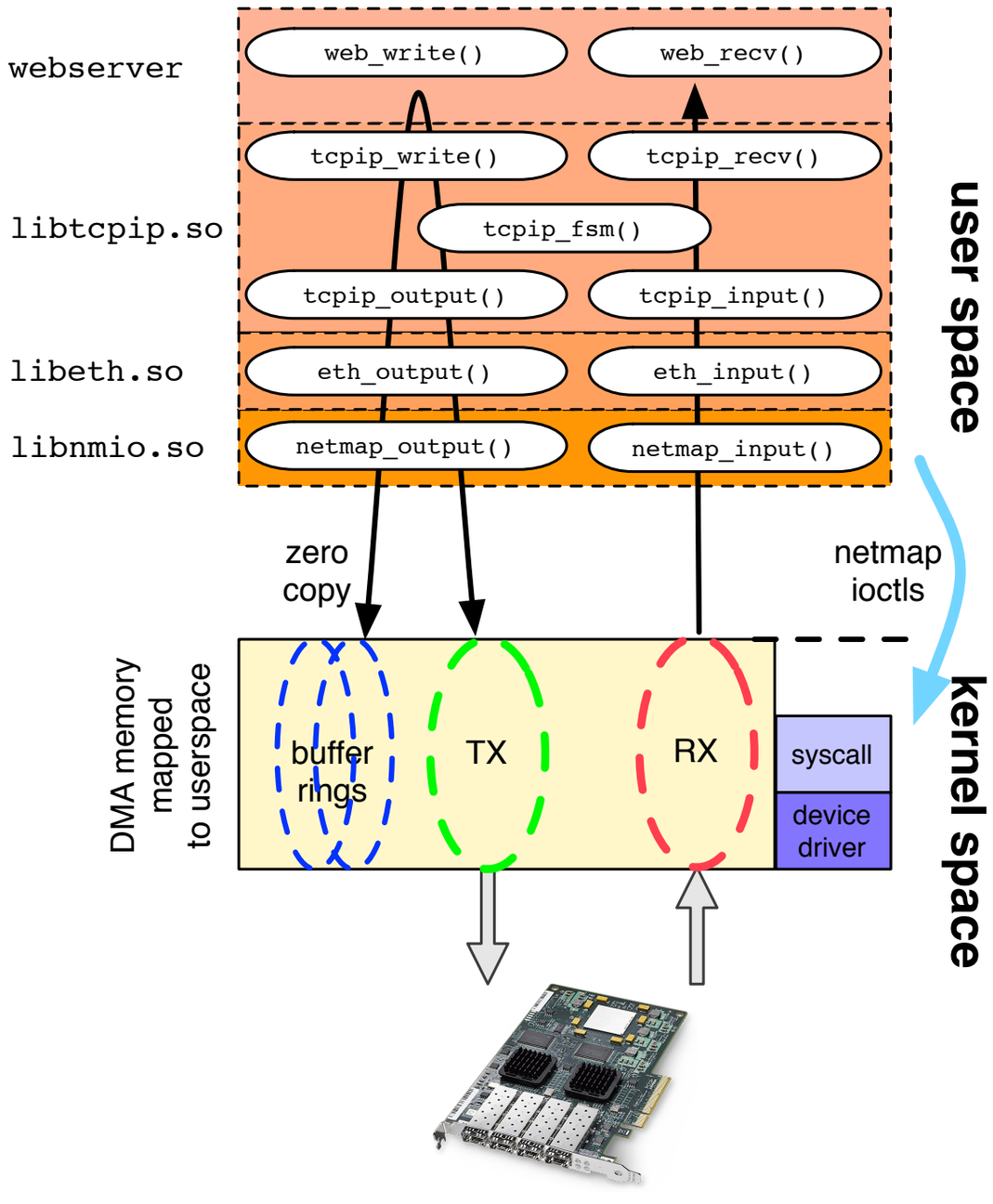
- Reduces system costs (e.g., batching, cache-locality, lock- and sharing-free, CPU-affinity)

- Exploits application-specific knowledge to reduce repetitive processing costs (e.g. TCP segmentation of web objects, checksums)

Sandstorm: A specialized webserver stack

Prototyped on top of FreeBSD's netmap framework:

- **`libnmio`**: abstracting netmap-related I/O
- **`libeth`**: lightweight ethernet layer
- **`libtcpip`**: optimized TCP/IP layer
- application: simple HTTP server that serves static content

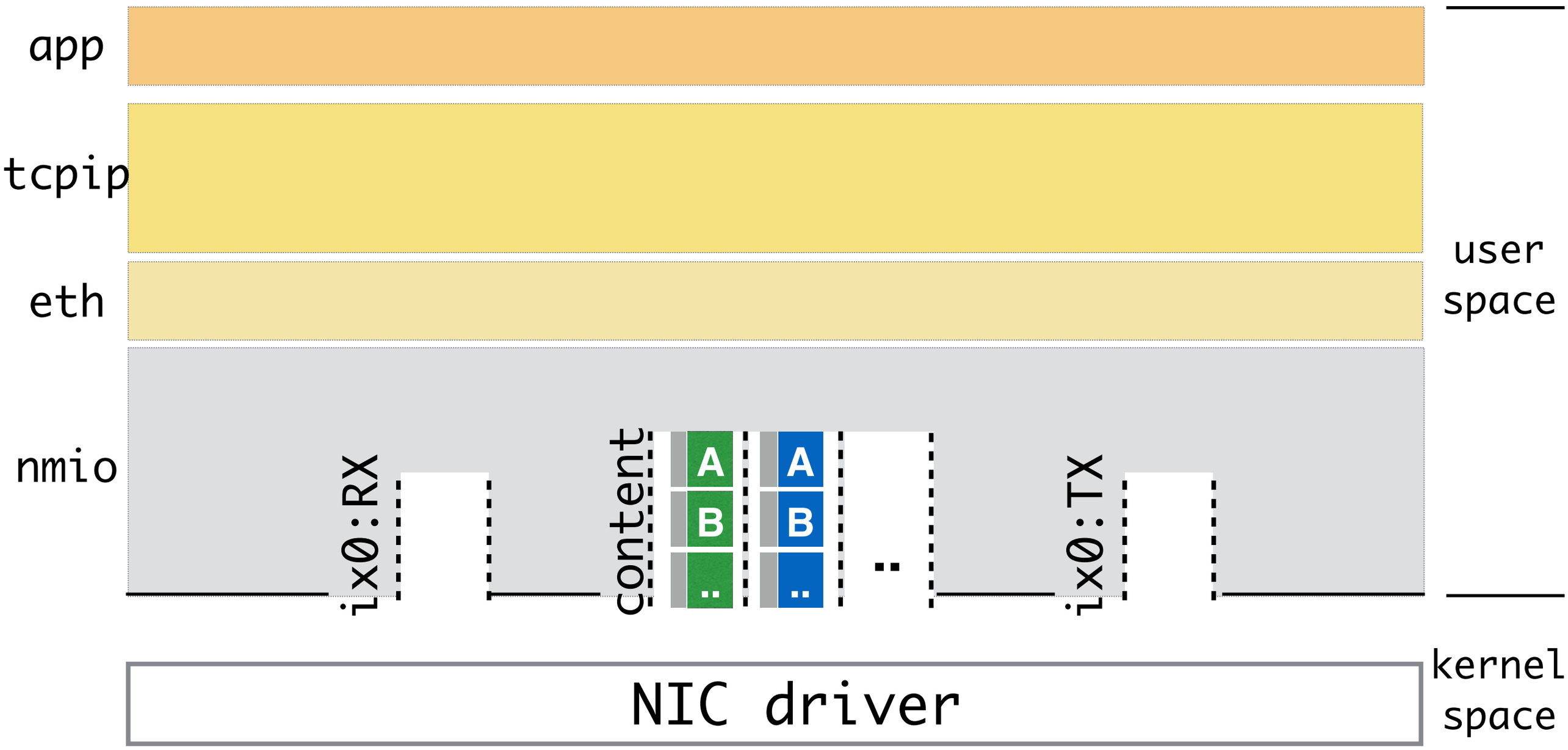


Sandstorm: A specialized webserver stack

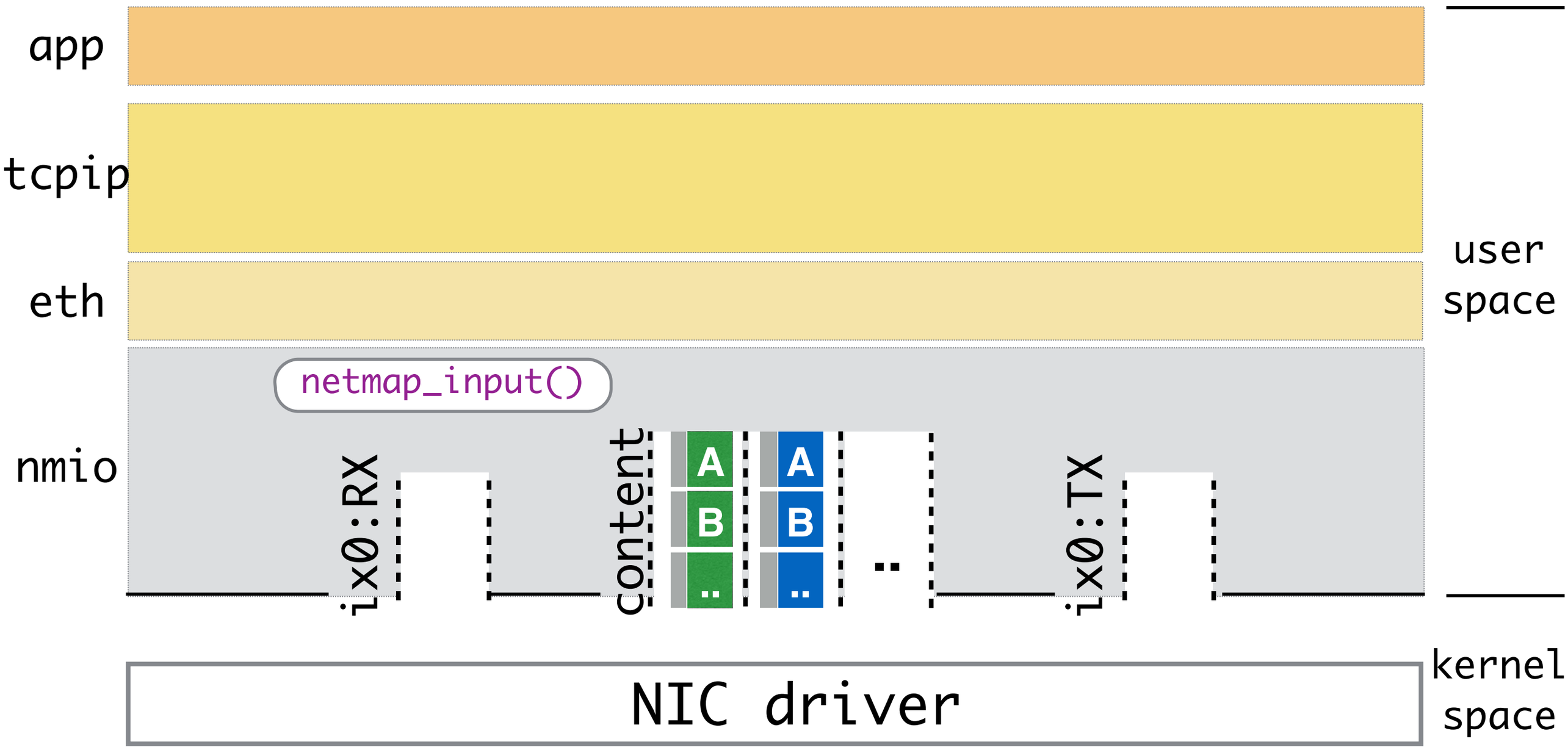
Key decisions (some of them):

- Application & stack are merged into the same process address space
- Static content is pre-segmented into network packets and *a-priori* loaded to DRAM
- Received packet frames are processed in-place on the RX rings, w/o memory copying/buffering
- RX/TX packet batching greatly amortizes the system call overhead
- Bufferless, synchronous model (no socket layer)

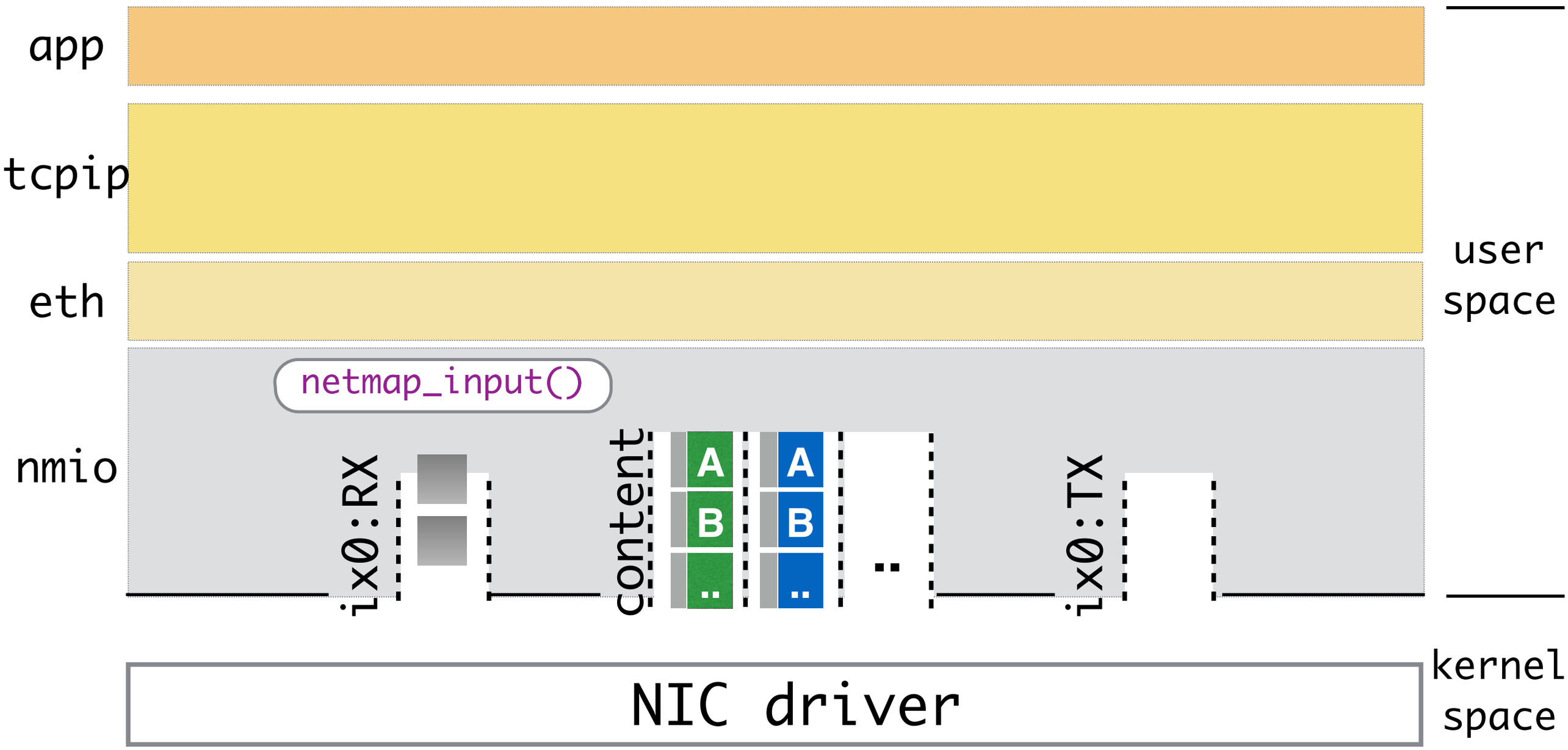
Sandstorm Architecture (10,000ft view)



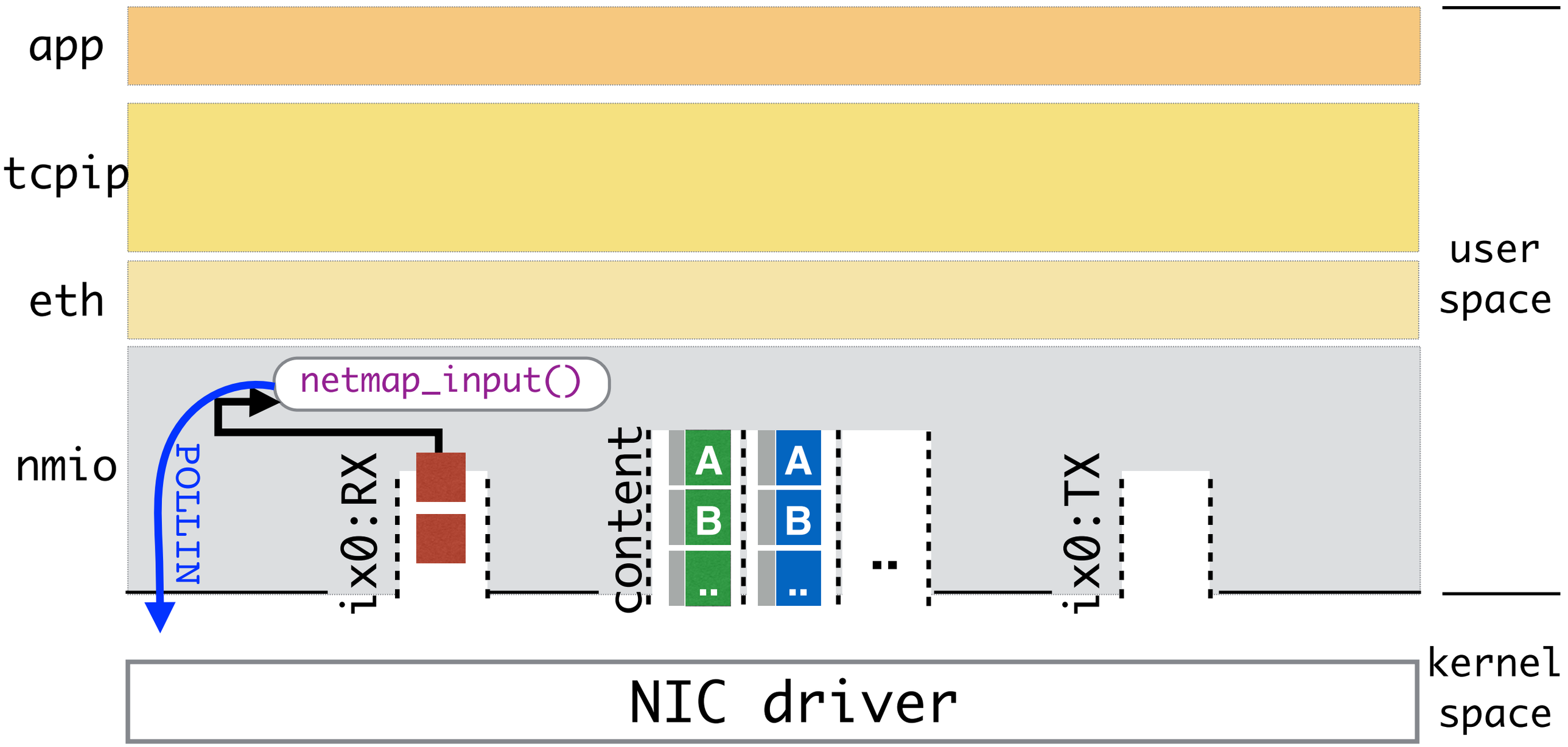
Sandstorm Architecture (10,000ft view)



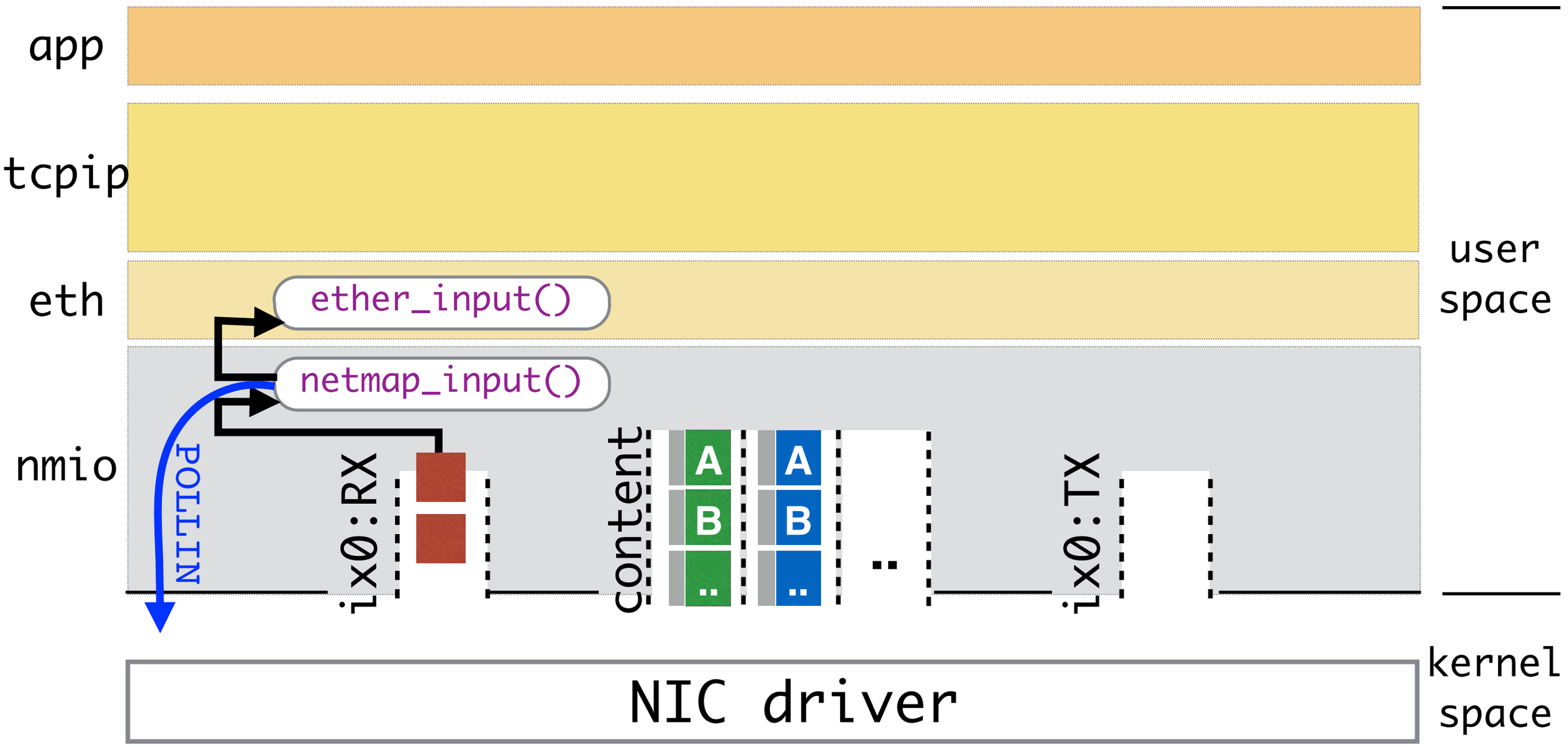
Sandstorm Architecture (10,000ft view)



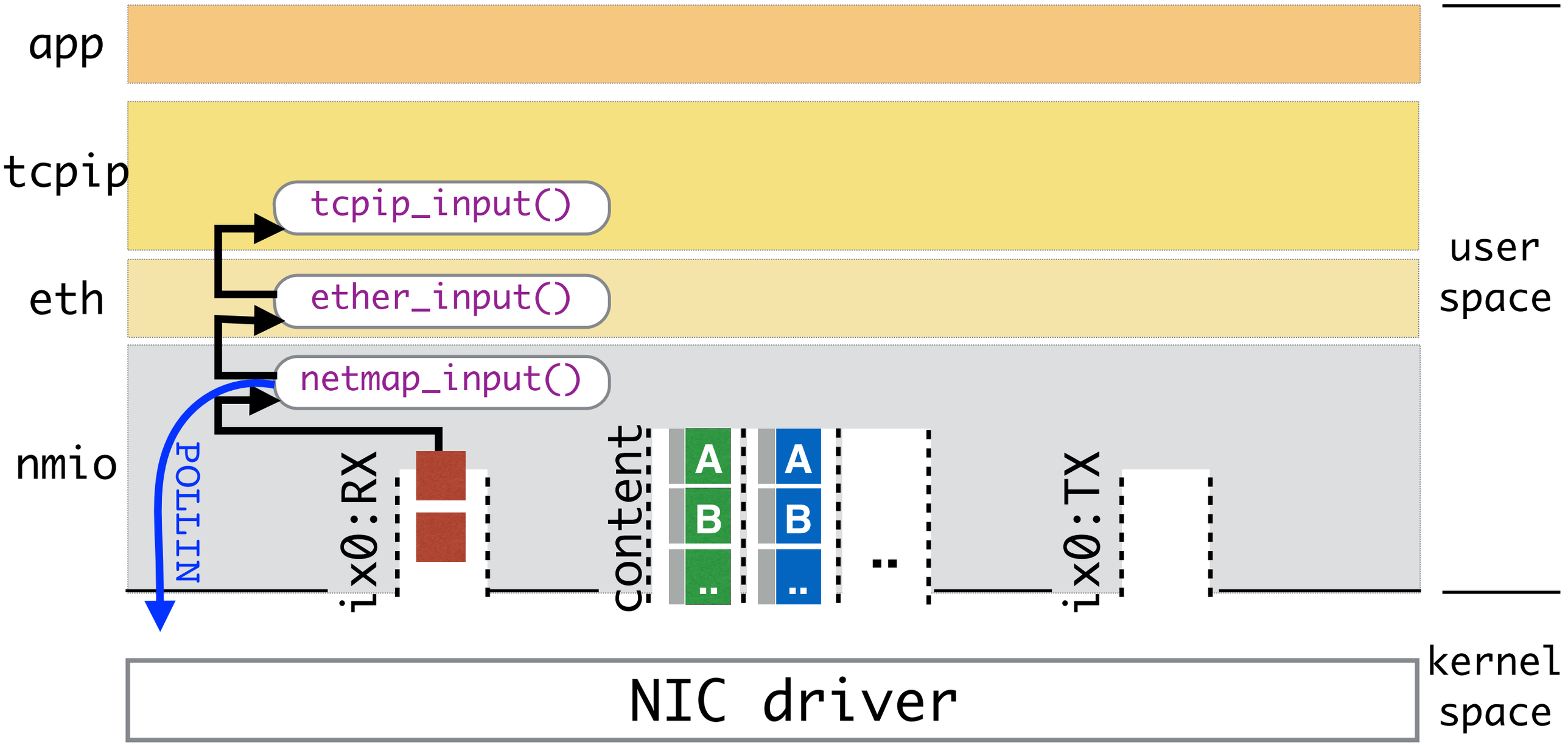
Sandstorm Architecture (10,000ft view)



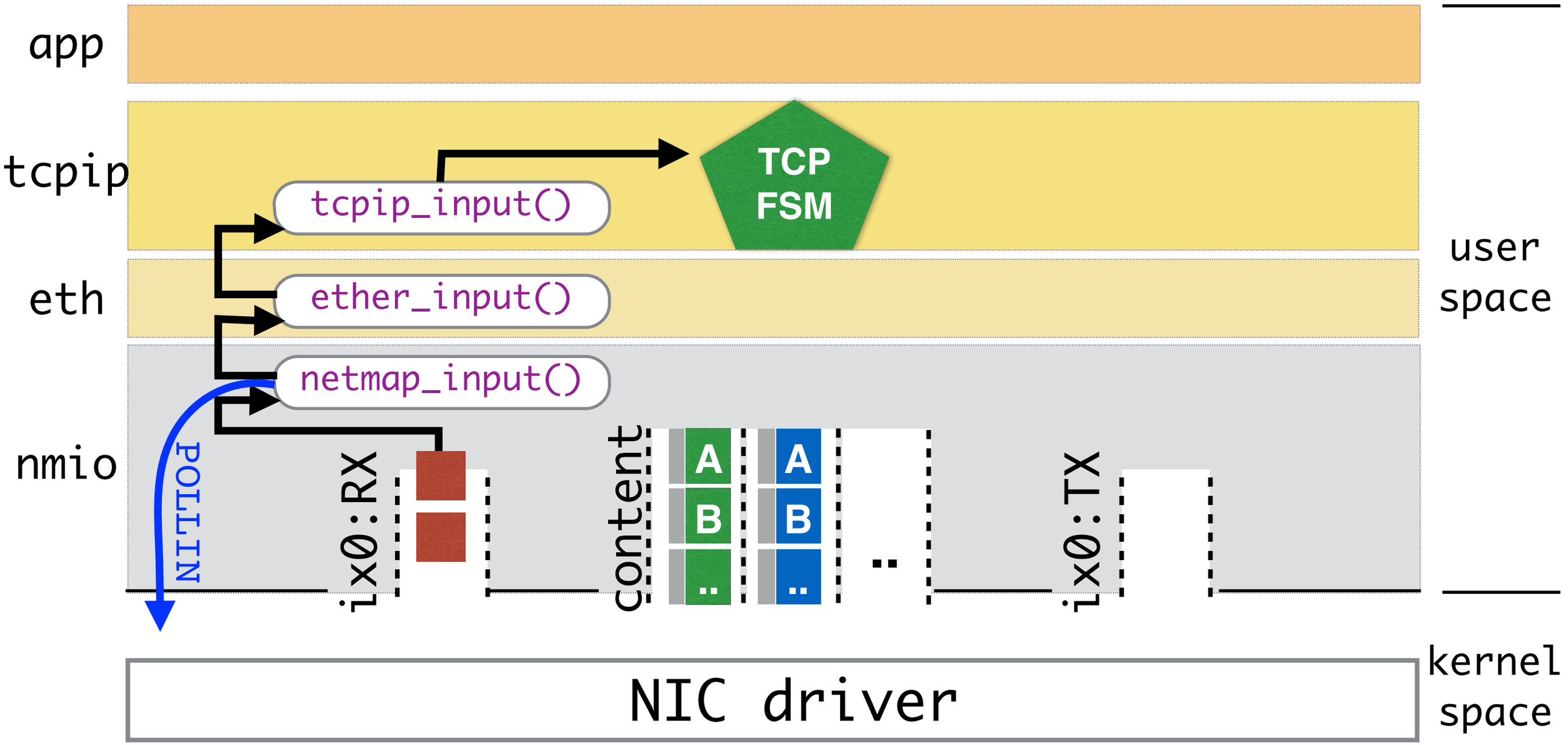
Sandstorm Architecture (10,000ft view)



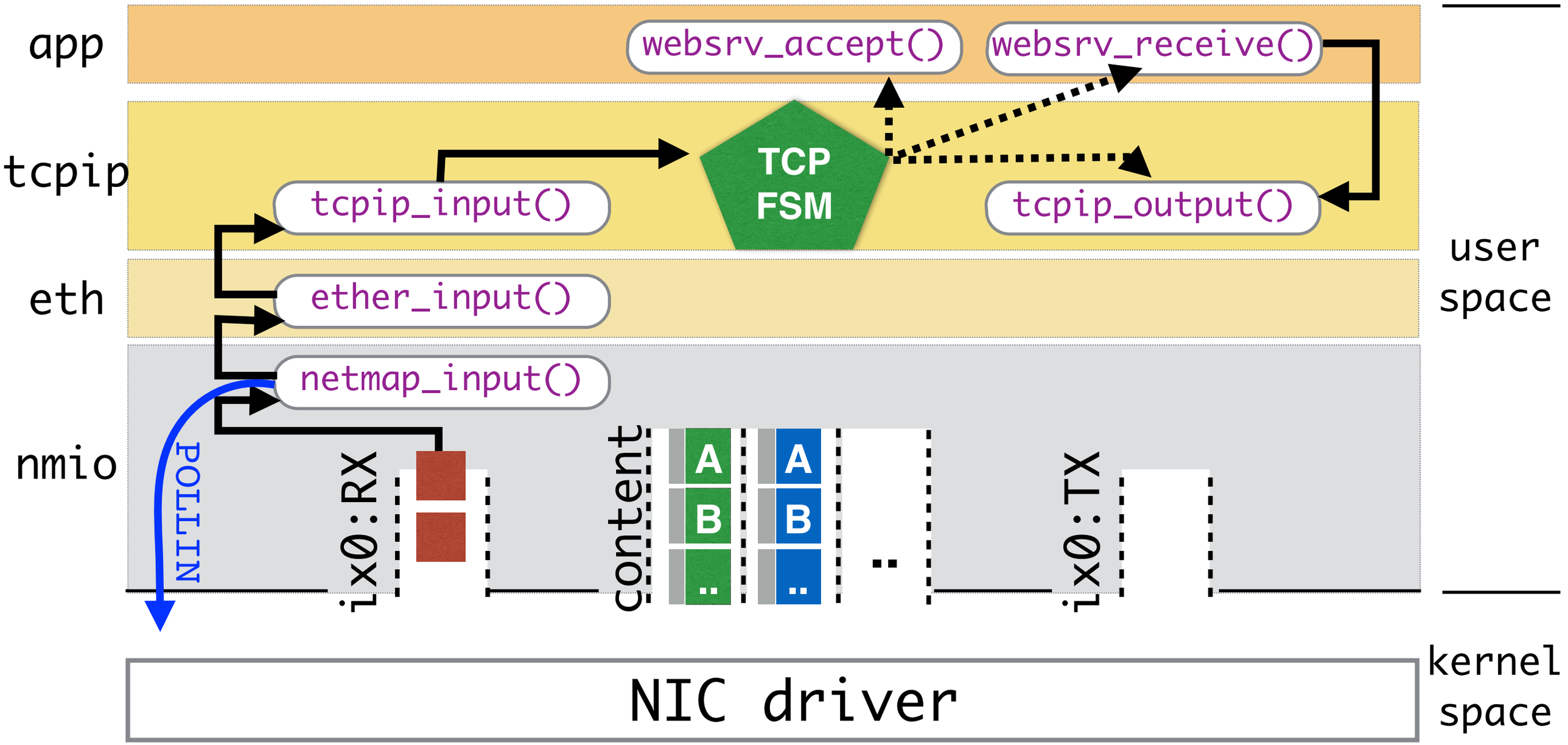
Sandstorm Architecture (10,000ft view)



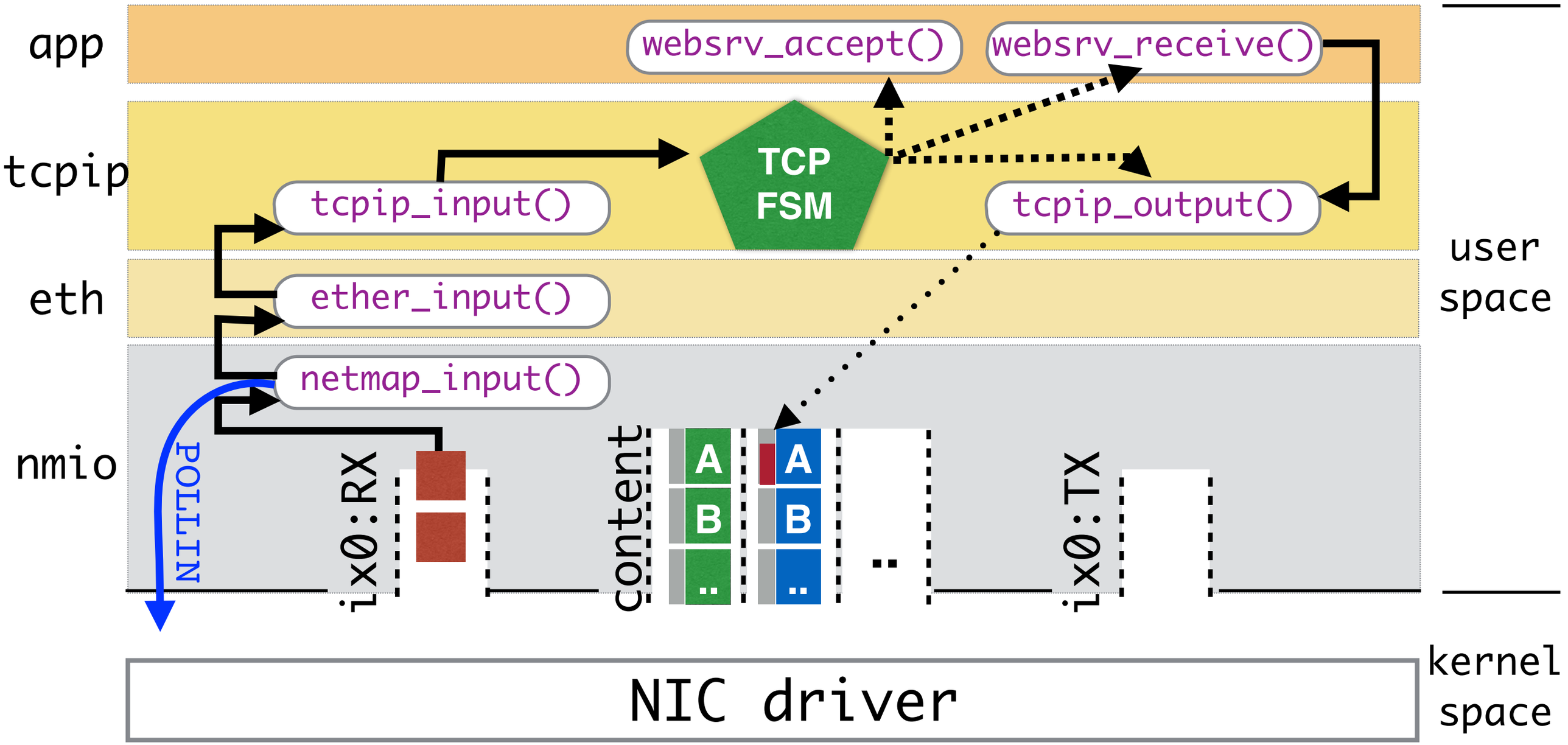
Sandstorm Architecture (10,000ft view)



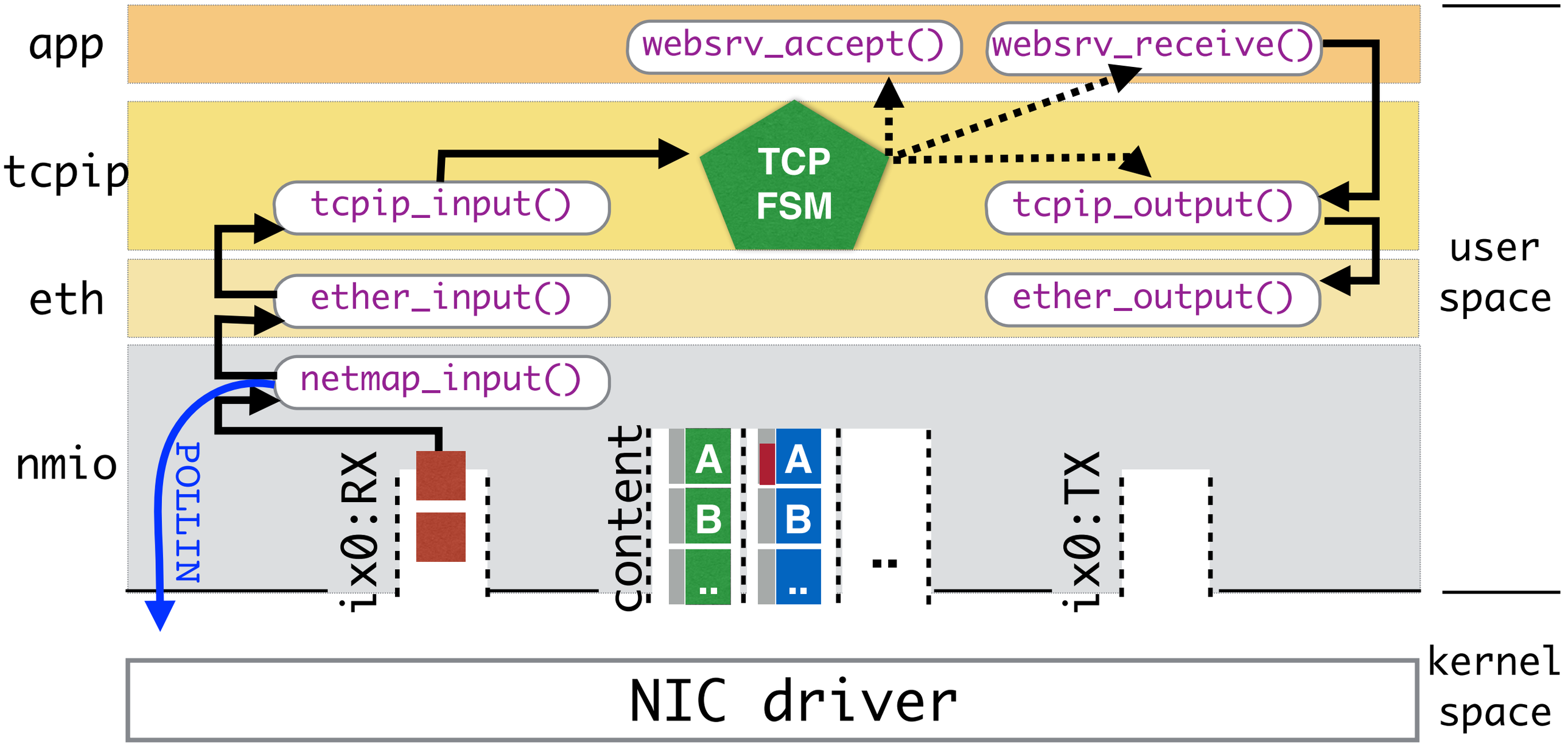
Sandstorm Architecture (10,000ft view)



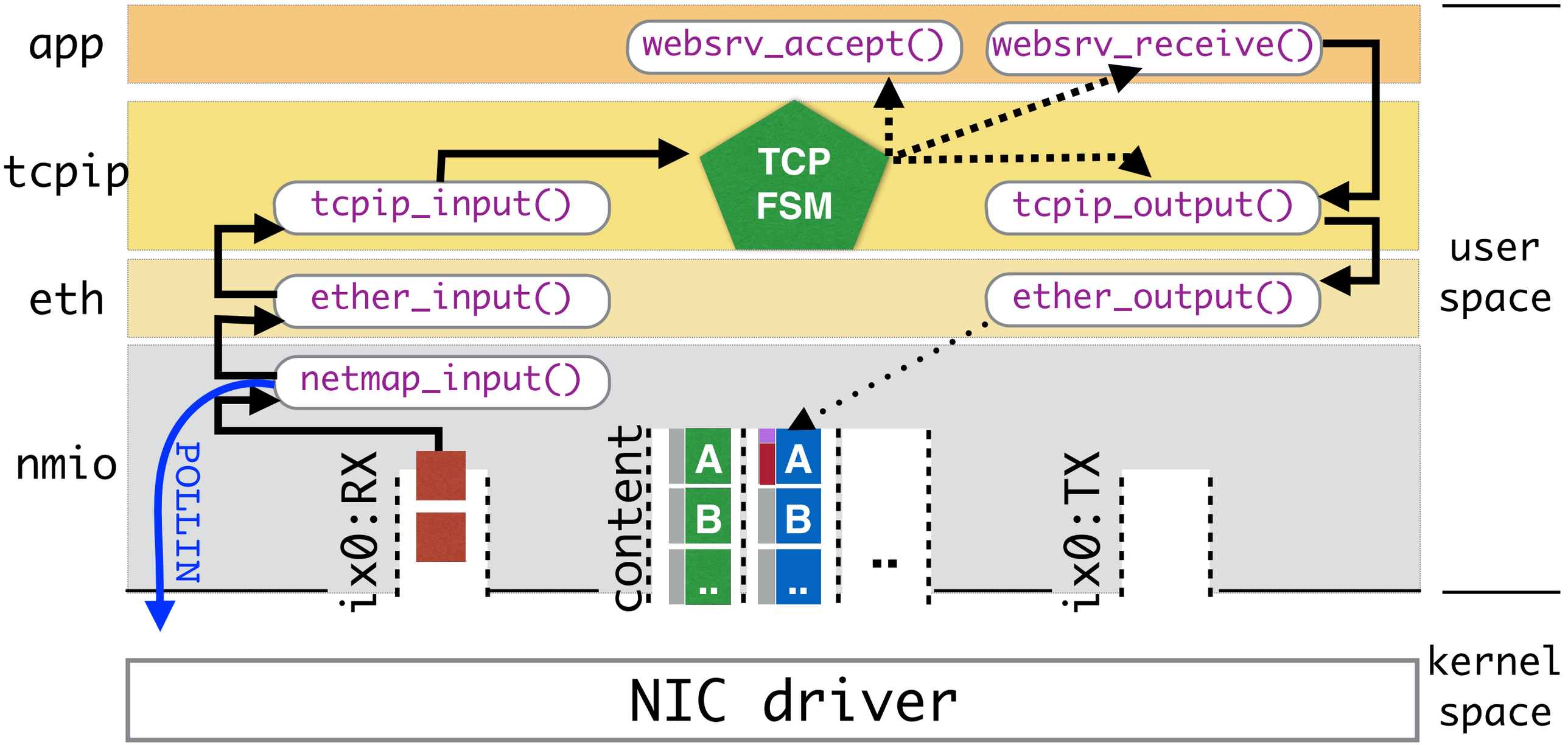
Sandstorm Architecture (10,000ft view)



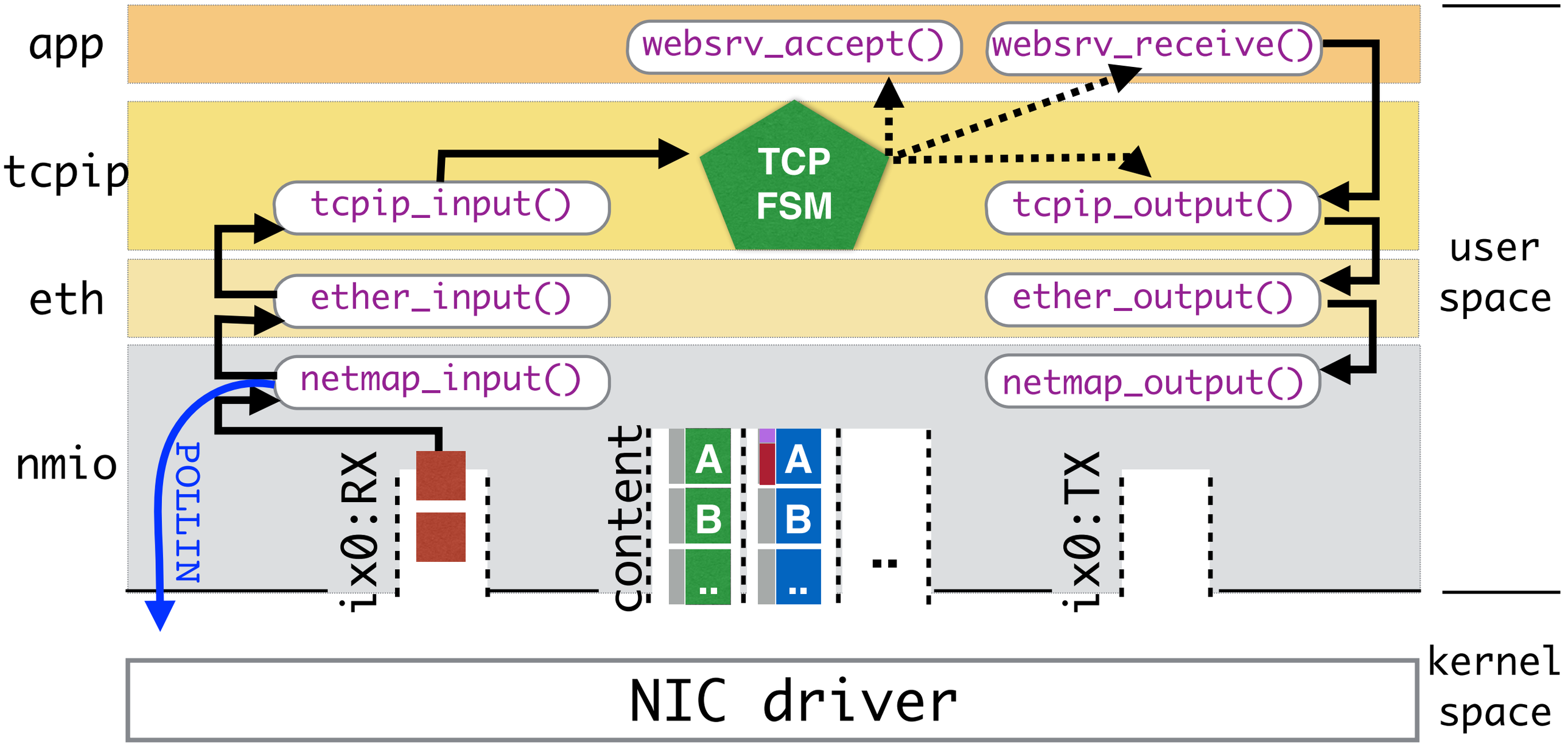
Sandstorm Architecture (10,000ft view)



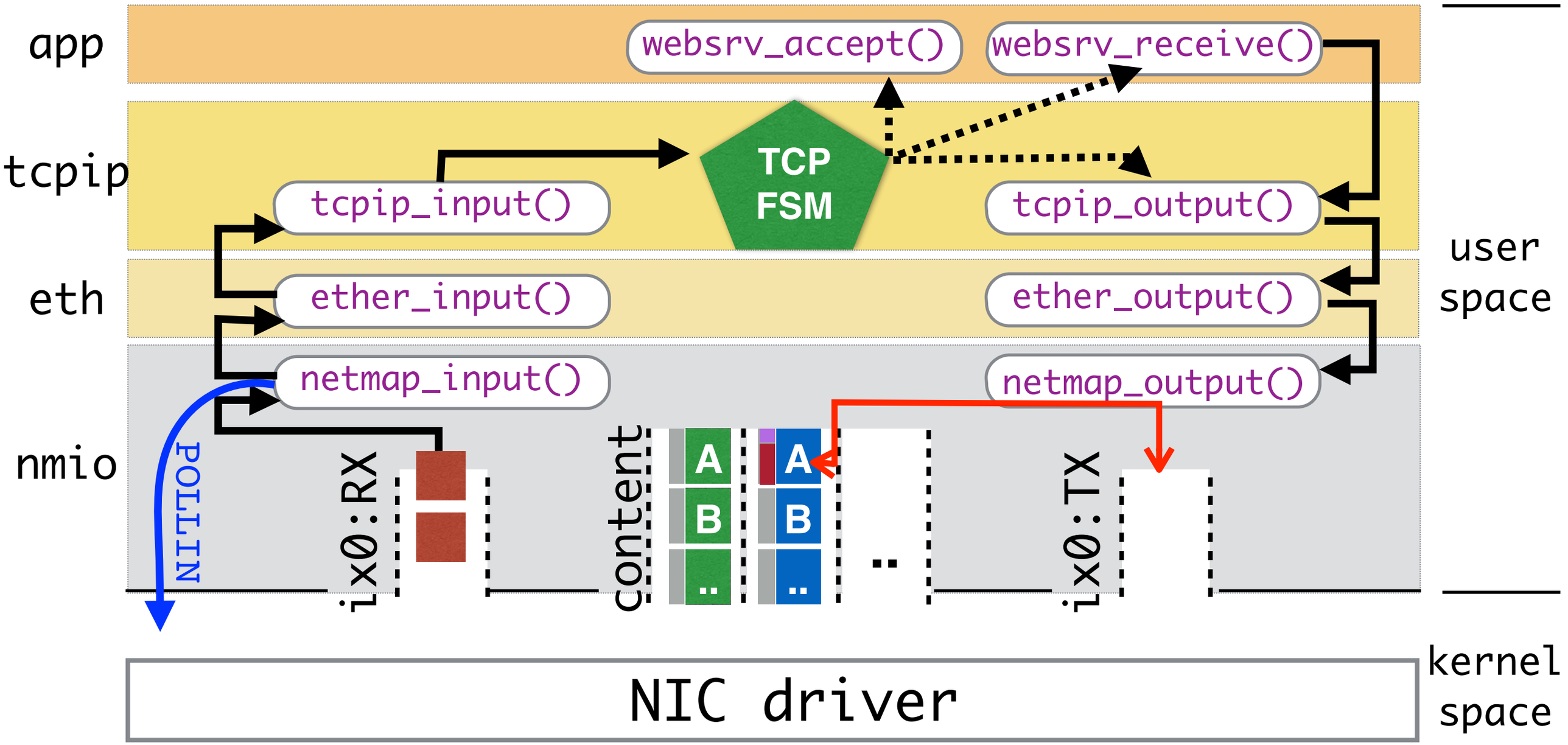
Sandstorm Architecture (10,000ft view)



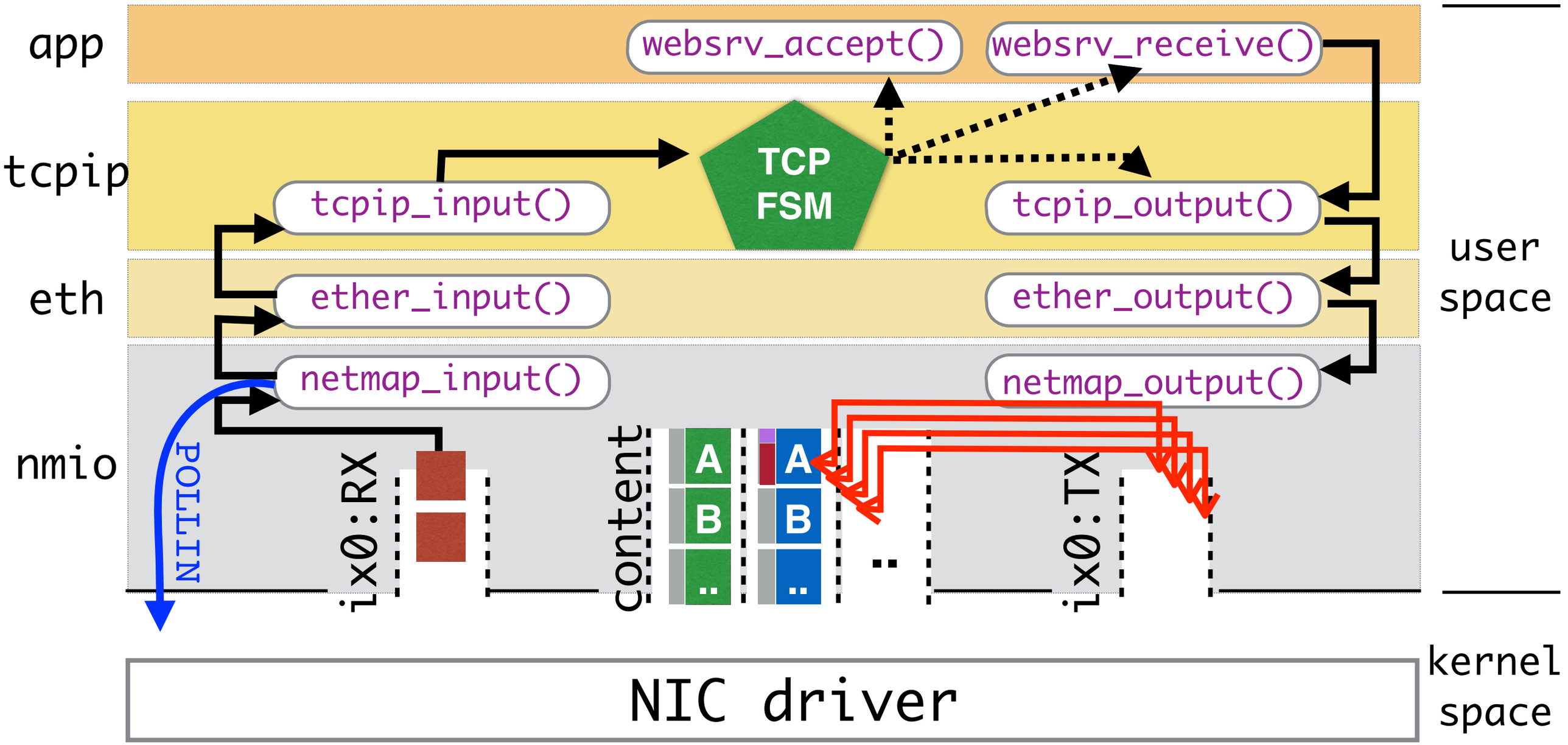
Sandstorm Architecture (10,000ft view)



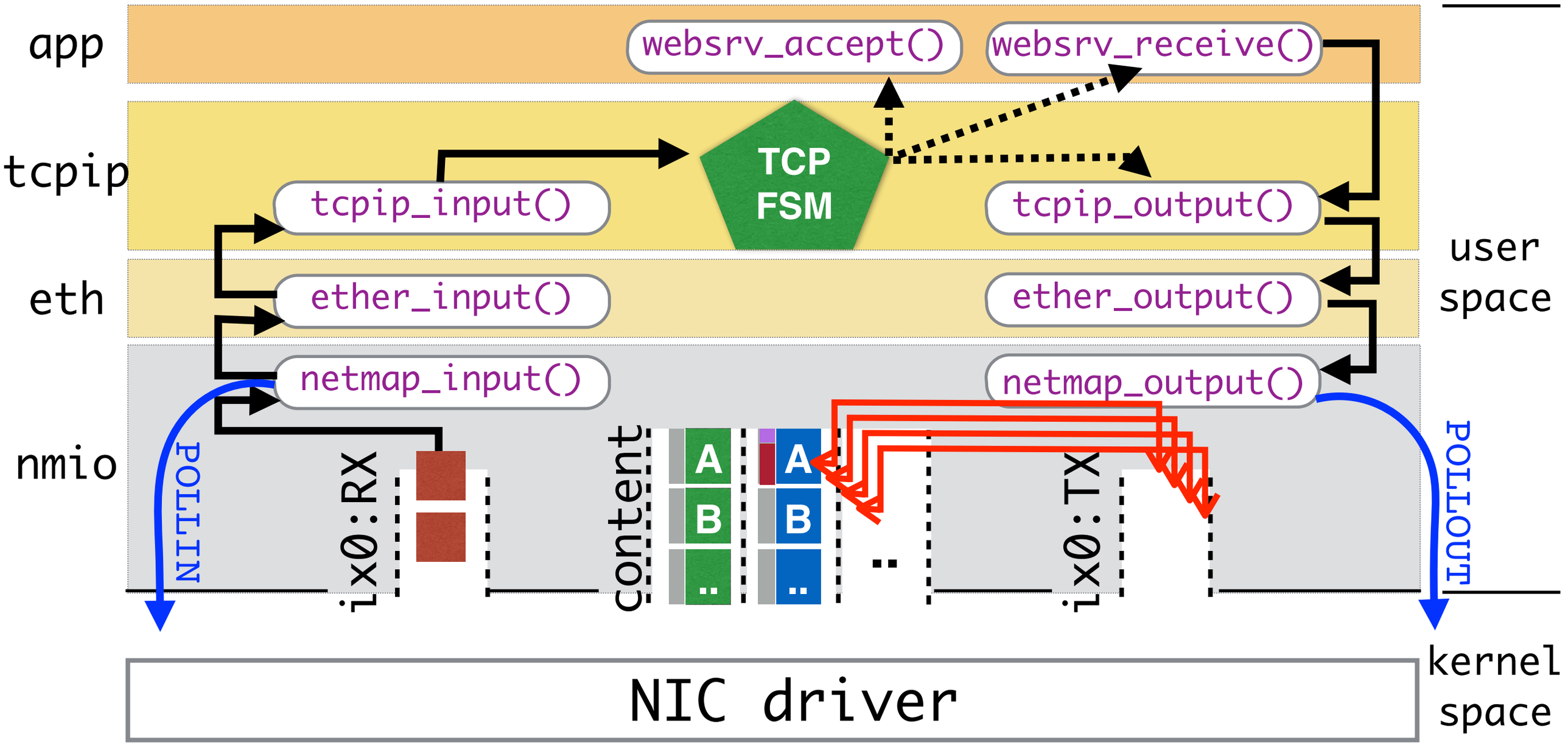
Sandstorm Architecture (10,000ft view)



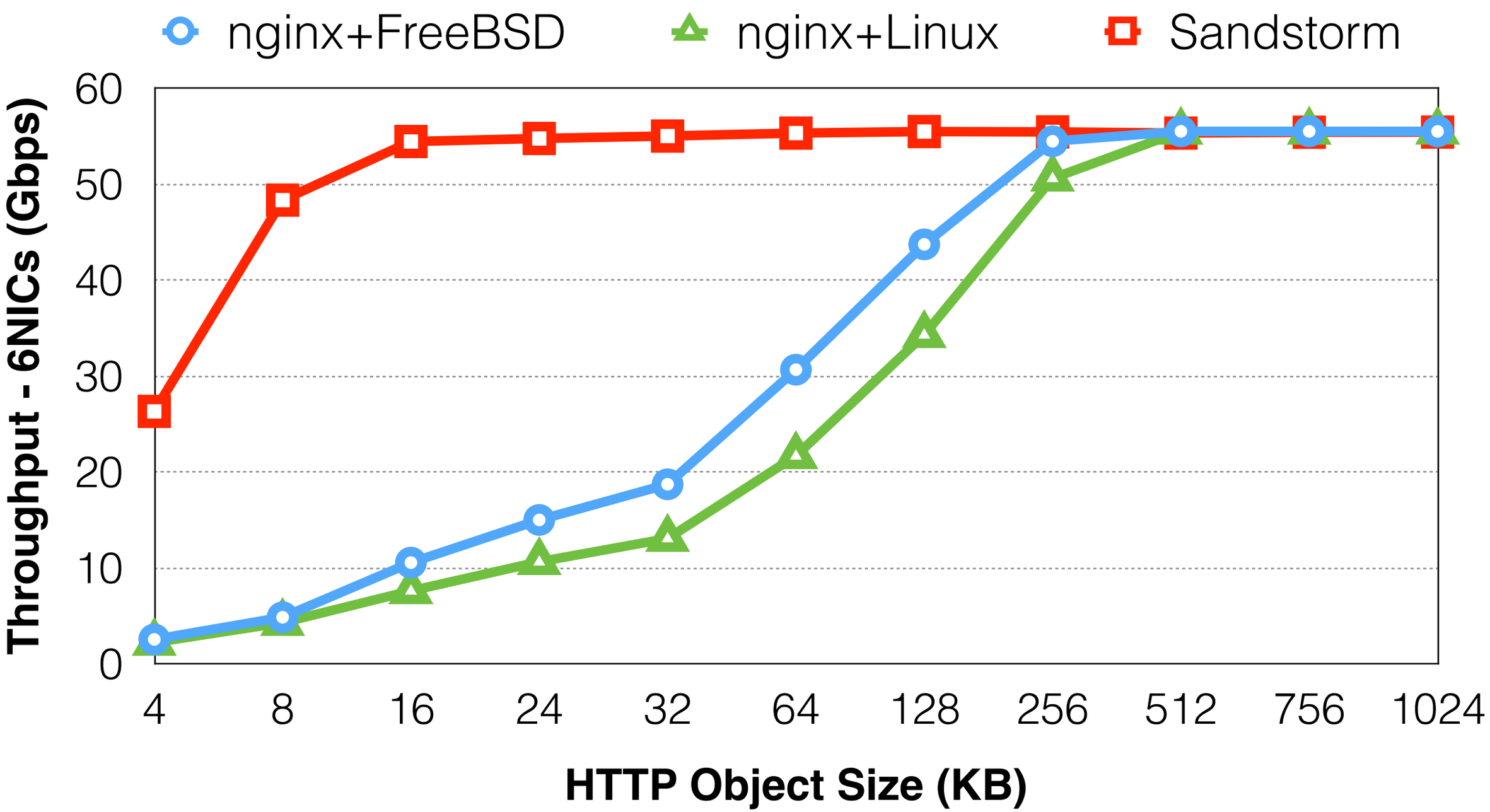
Sandstorm Architecture (10,000ft view)



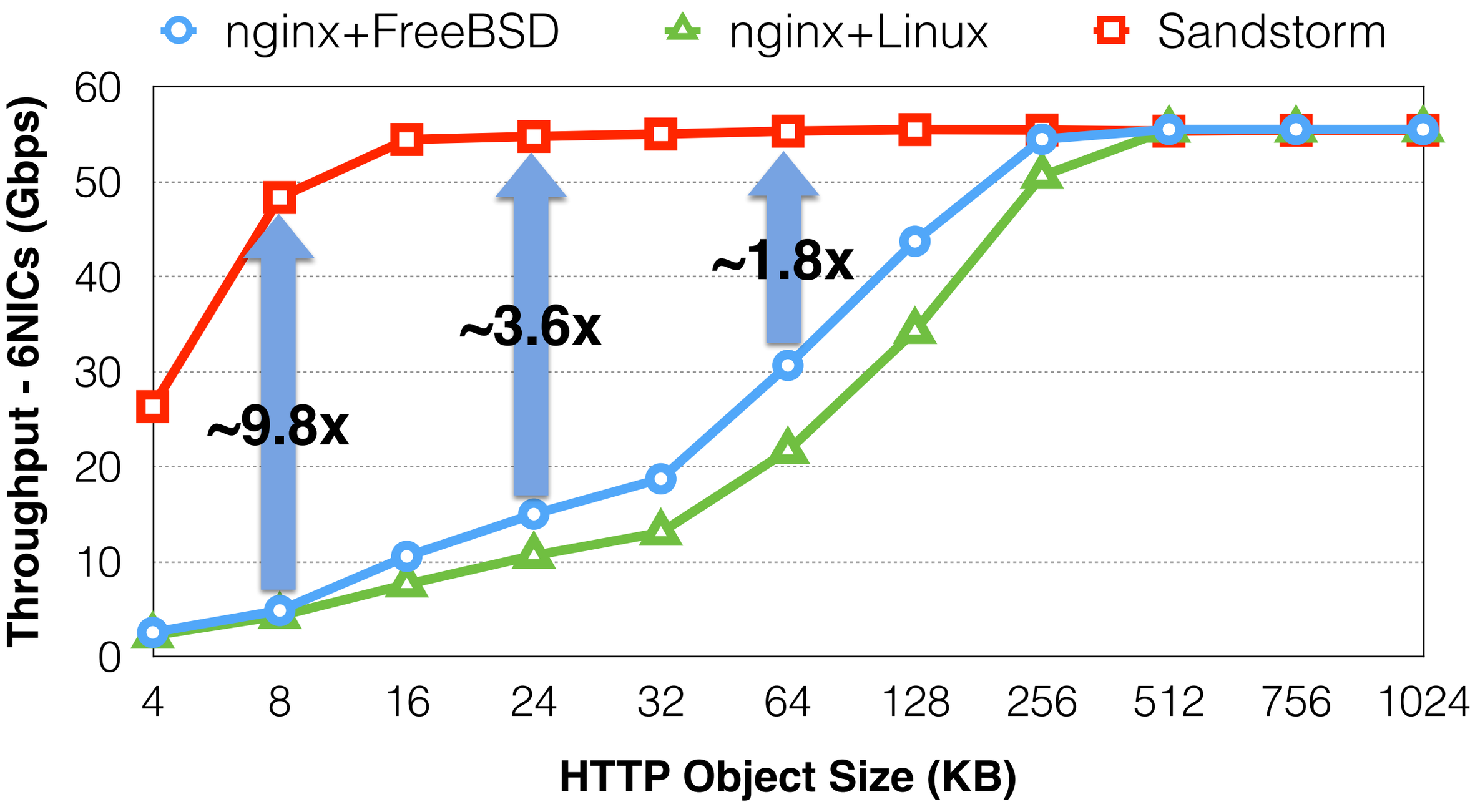
Sandstorm Architecture (10,000ft view)



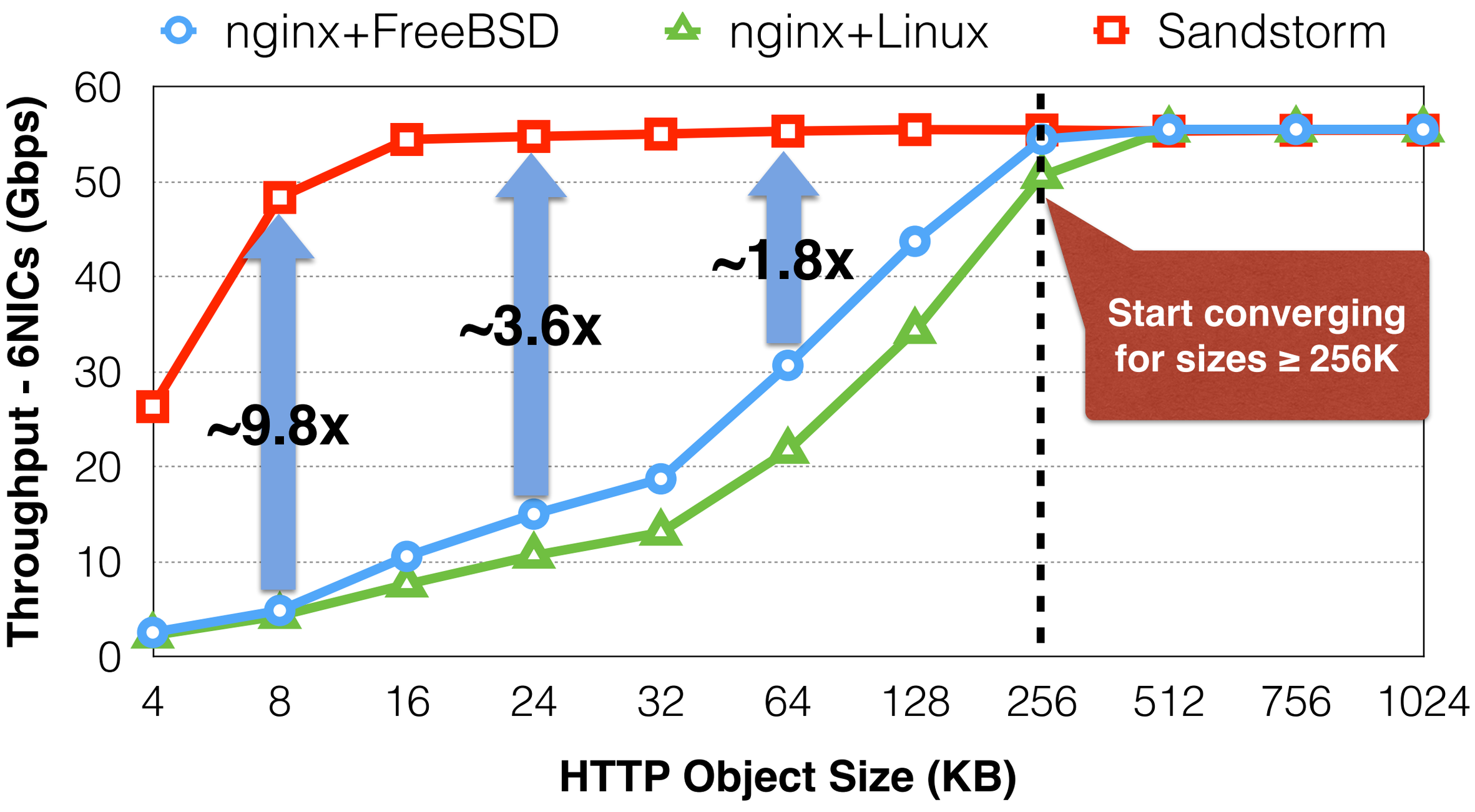
Evaluation


nginx+FreeBSD
nginx+Linux
Sandstorm
Throughput - 6NICs (Gbps)
60
50
40
30
20
10
0
4
8
16
24
32
64
128
256
512
756
1024
HTTP Object Size (KB)

Evaluation


nginx+FreeBSD
nginx+Linux
Sandstorm
Throughput - 6NICs (Gbps)
60
50
40
30
20
10
0
~9.8x
~3.6x
~1.8x
4
8
16
24
32
64
128
256
512
756
1024
HTTP Object Size (KB)

Evaluation

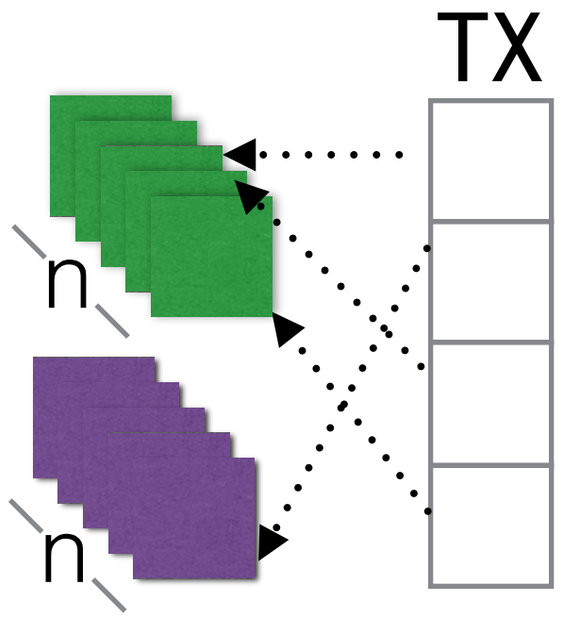

nginx+FreeBSD
nginx+Linux
Sandstorm
Throughput - 6NICs (Gbps)
60
50
40
30
20
10
0
~9.8x
~3.6x
~1.8x
Start converging for sizes ≥ 256K
4
8
16
24
32
64
128
256
512
756
1024
HTTP Object Size (KB)

To copy or not to copy?

zerocopy

```
/* Get src and destination slots */
struct netmap_slot *bf = &ppool->slot[slotindex];
struct netmap_slot *tx = &txring->slot[cur];

/* zero-copy packet */
tx->buf_idx = bf->buf_idx;
tx->len = bf->len;
tx->flags = NS_BUF_CHANGED;
```

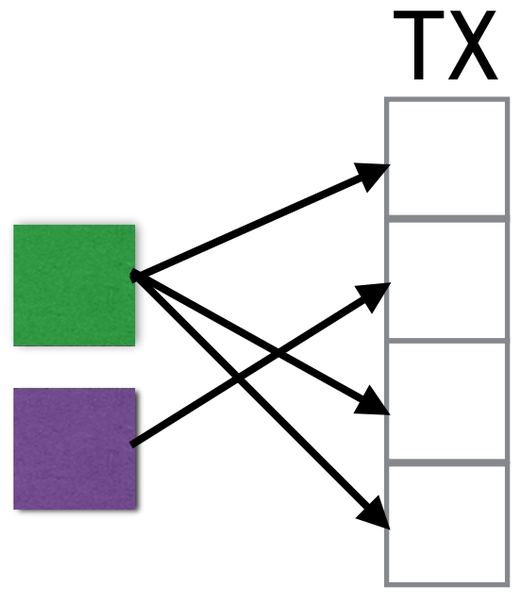


OR

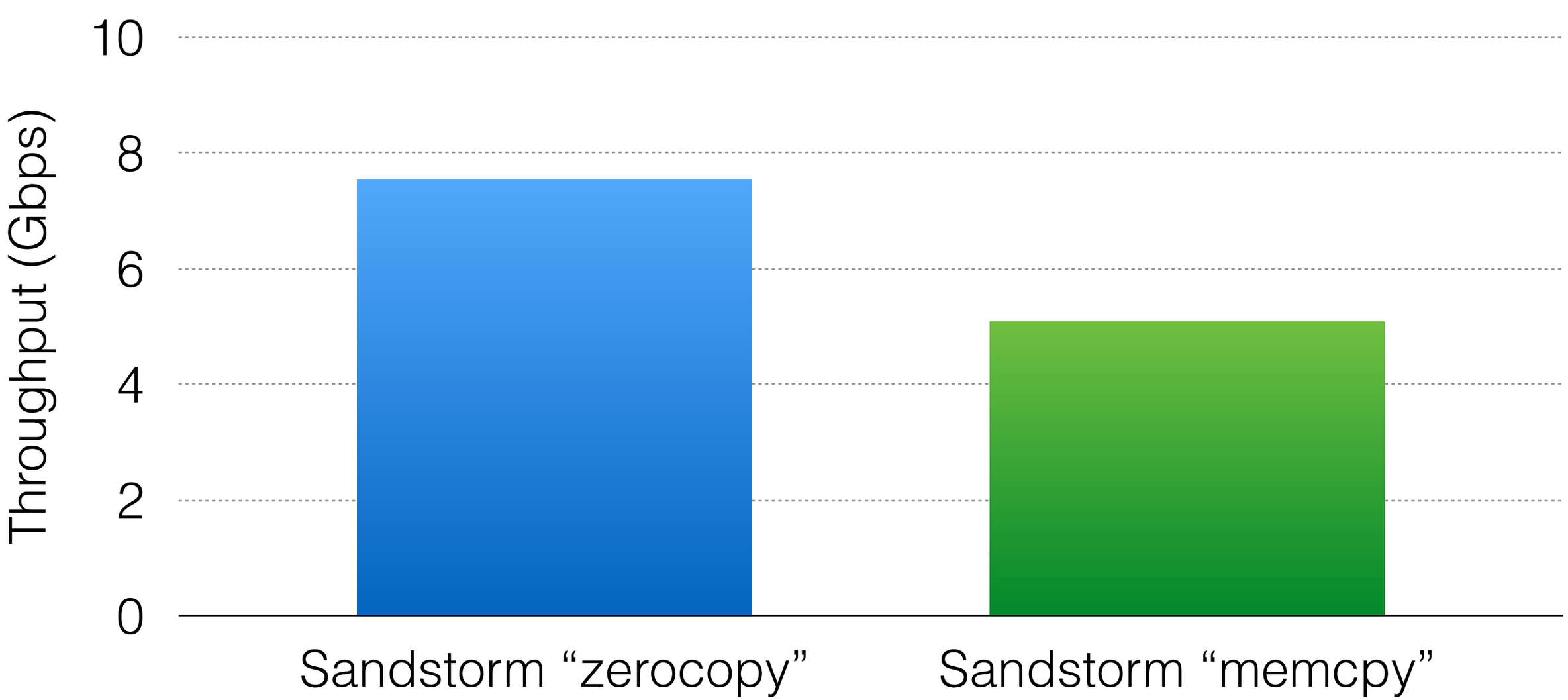
memcpy

```
/* Get source and destination bufs */
char *srcp = NETMAP_BUF(ppool, bf->buf_idx);
char *dstp = NETMAP_BUF(txring, tx->buf_idx);

/* memcpy packet */
memcpy(dstp, srcp, bf->len);
tx->len = bf->len;
```



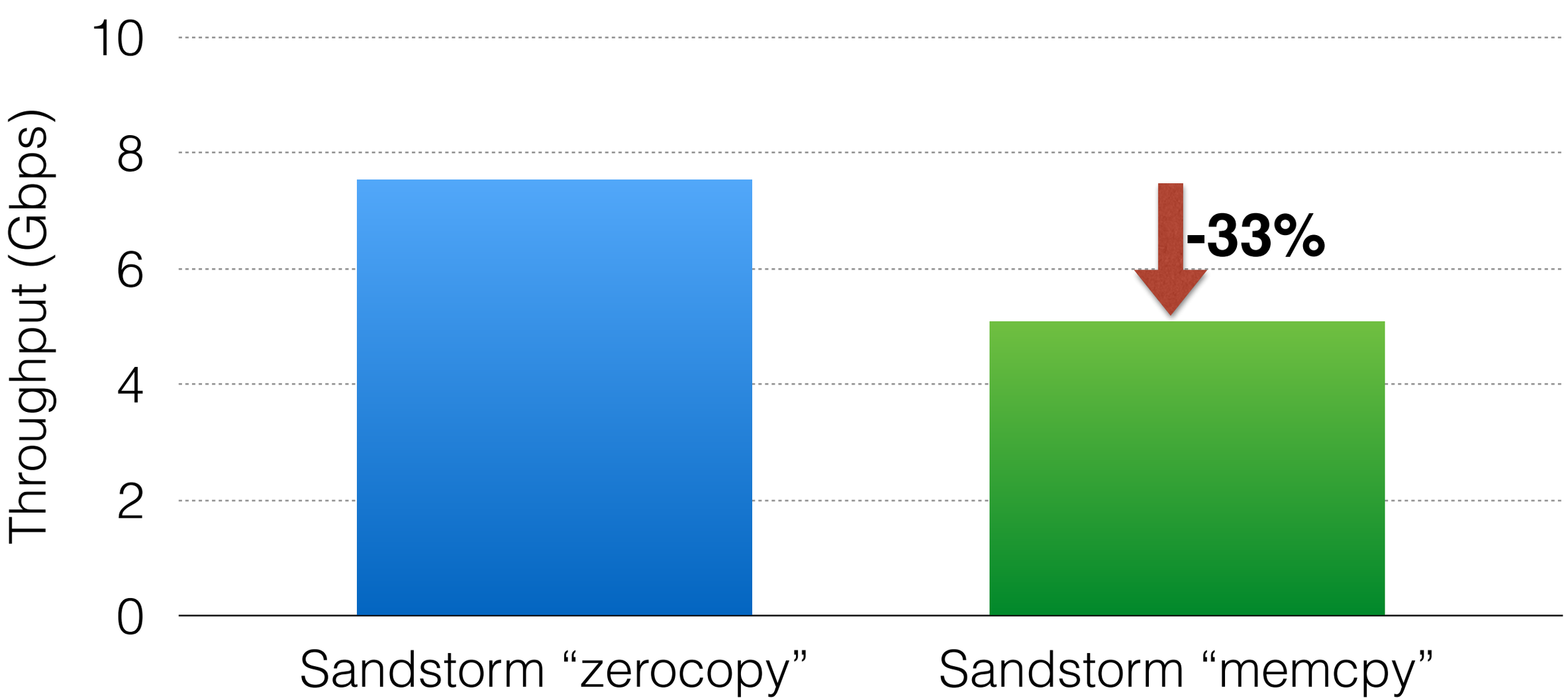
To copy or not to copy?



Intel Core 2 (2006)

Serving a 24KB HTTP object

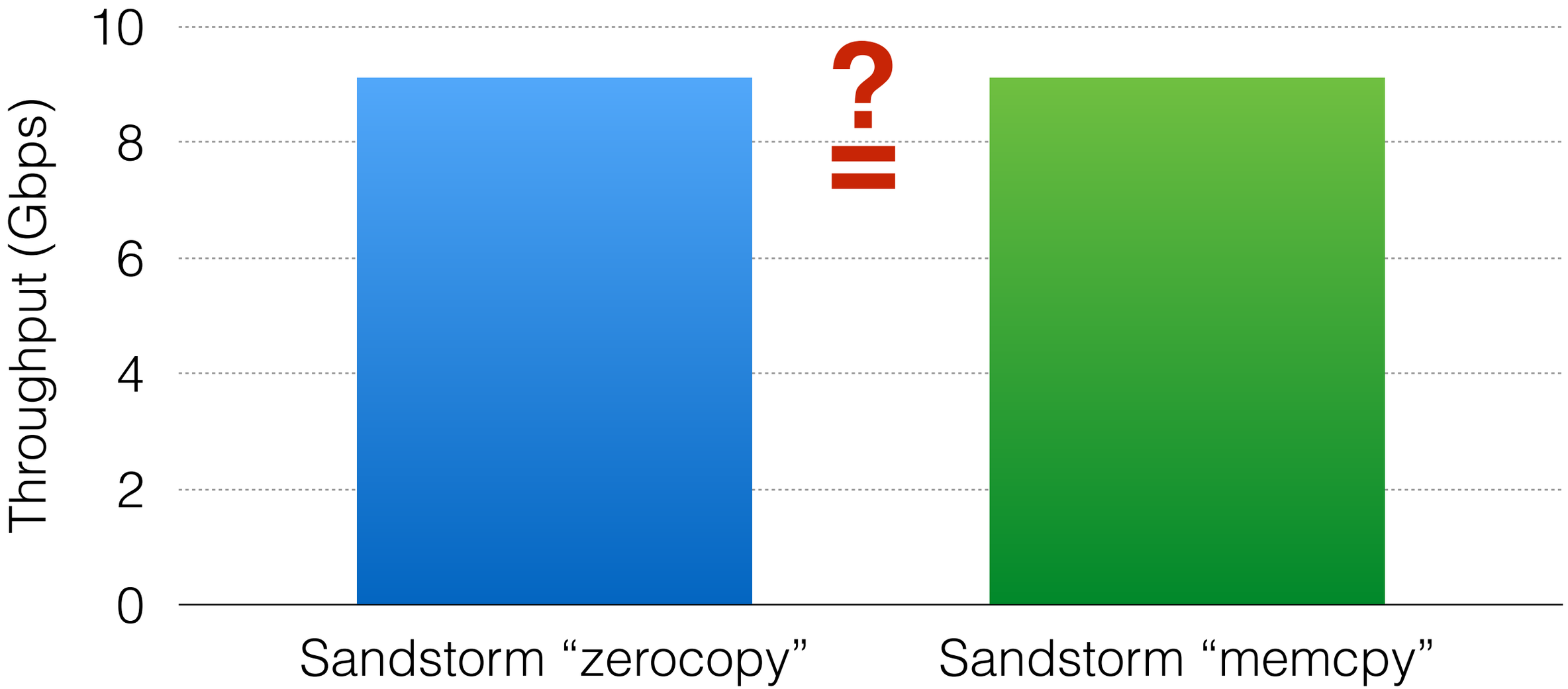
To copy or not to copy?



Intel Core 2 (2006)

Serving a 24KB HTTP object

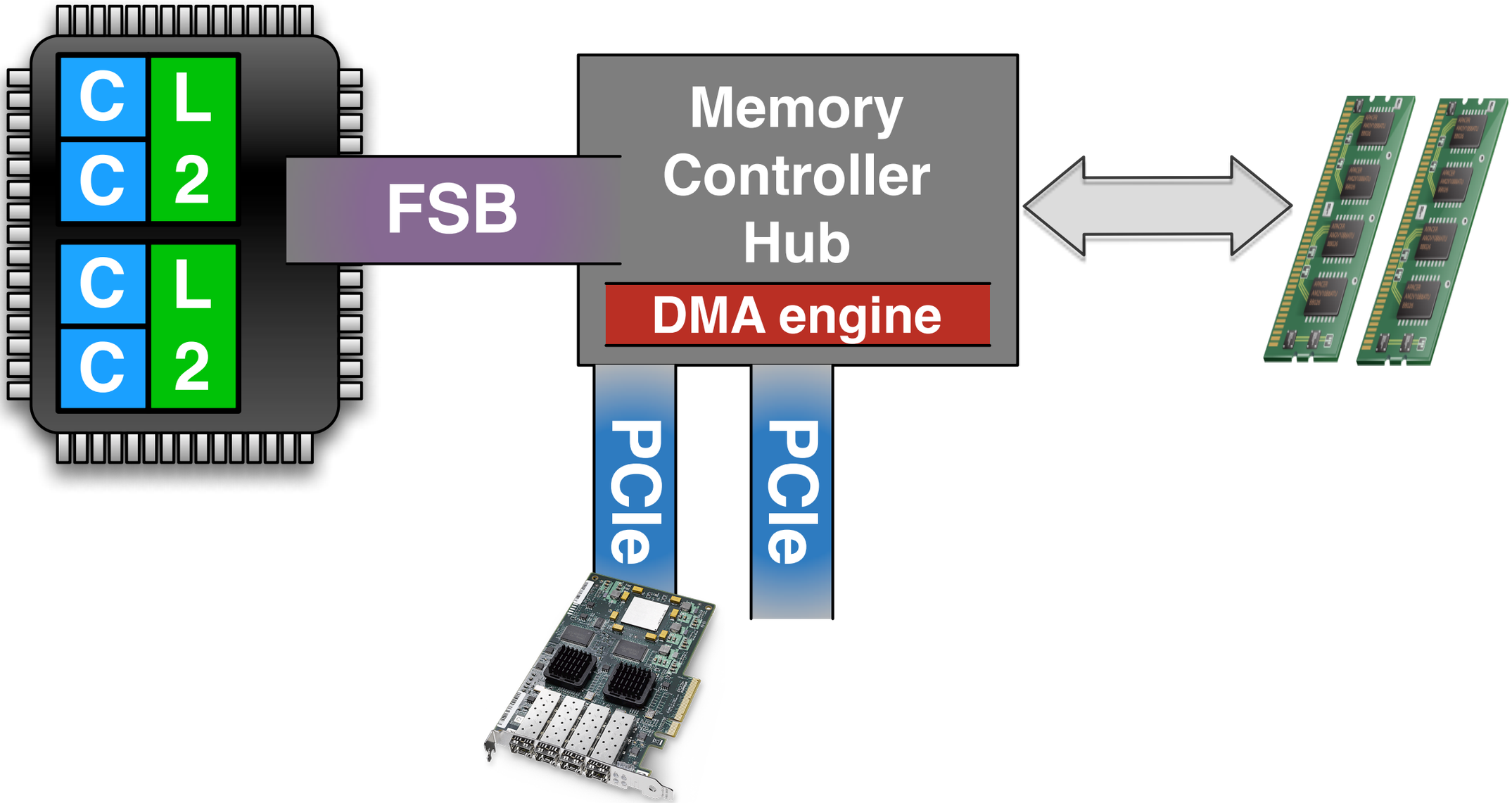
To copy or not to copy?



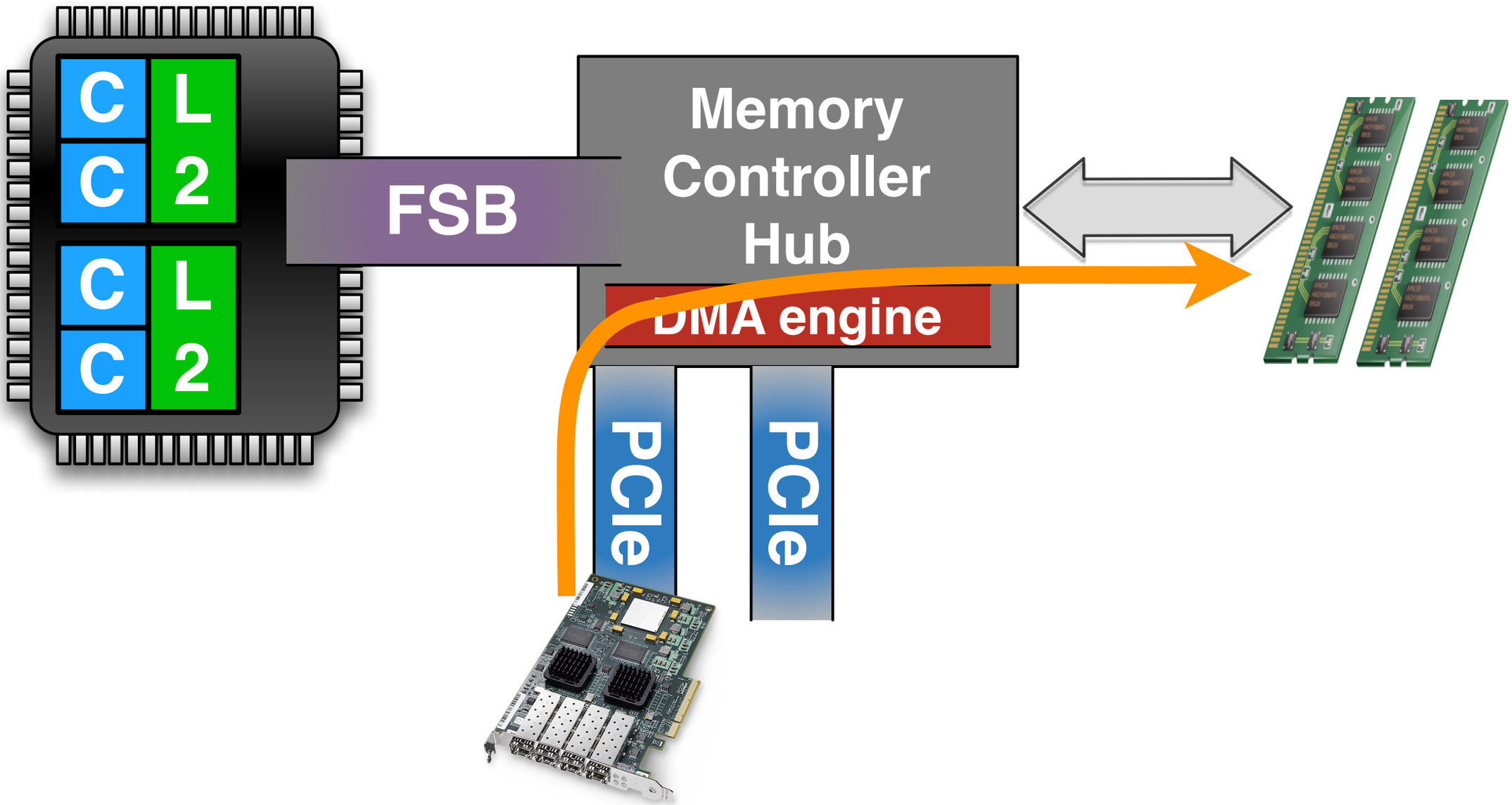
Intel Sandybridge (2013)

Serving a 24KB HTTP object

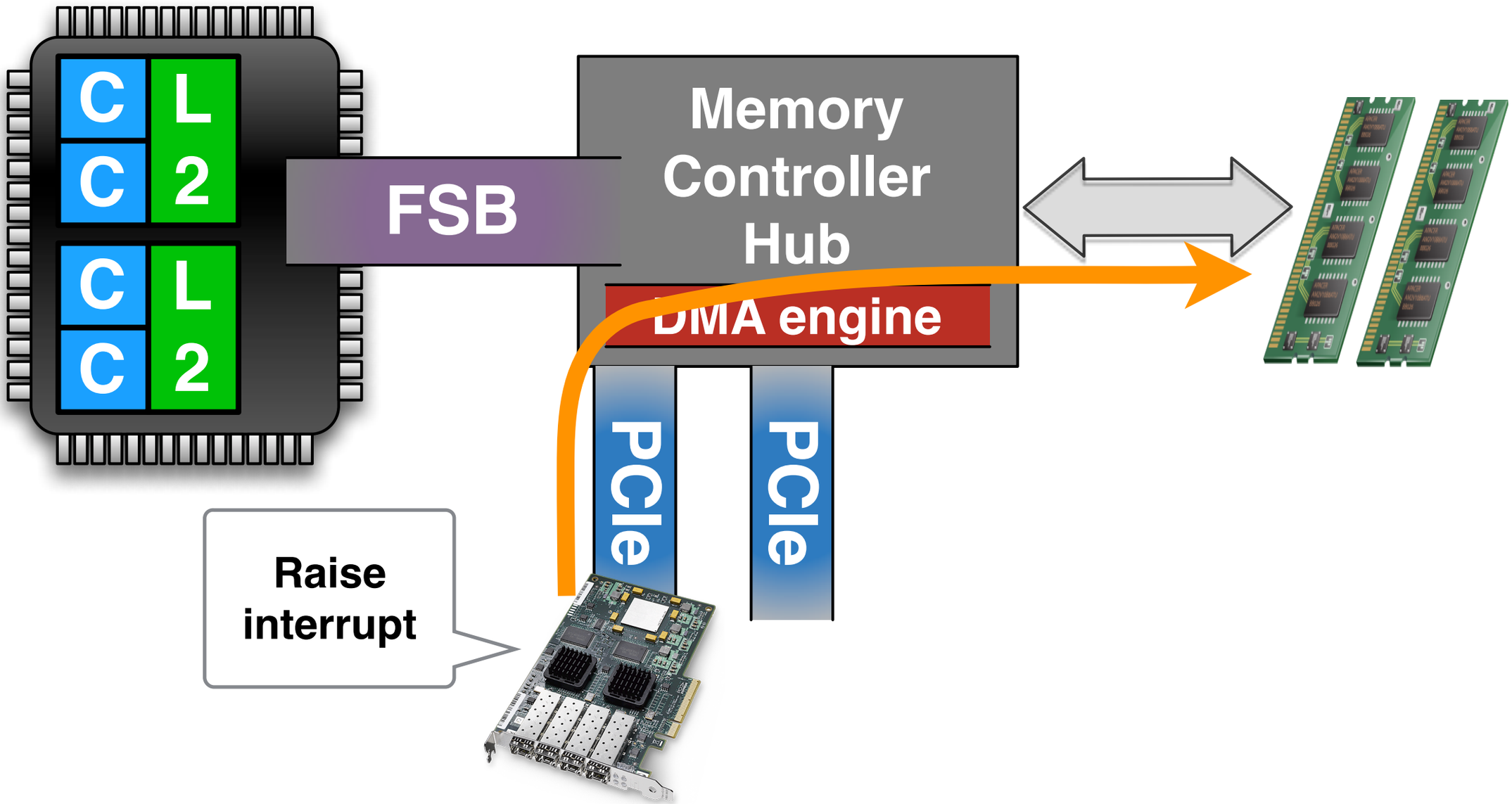
CPU microarchitecture ~2006



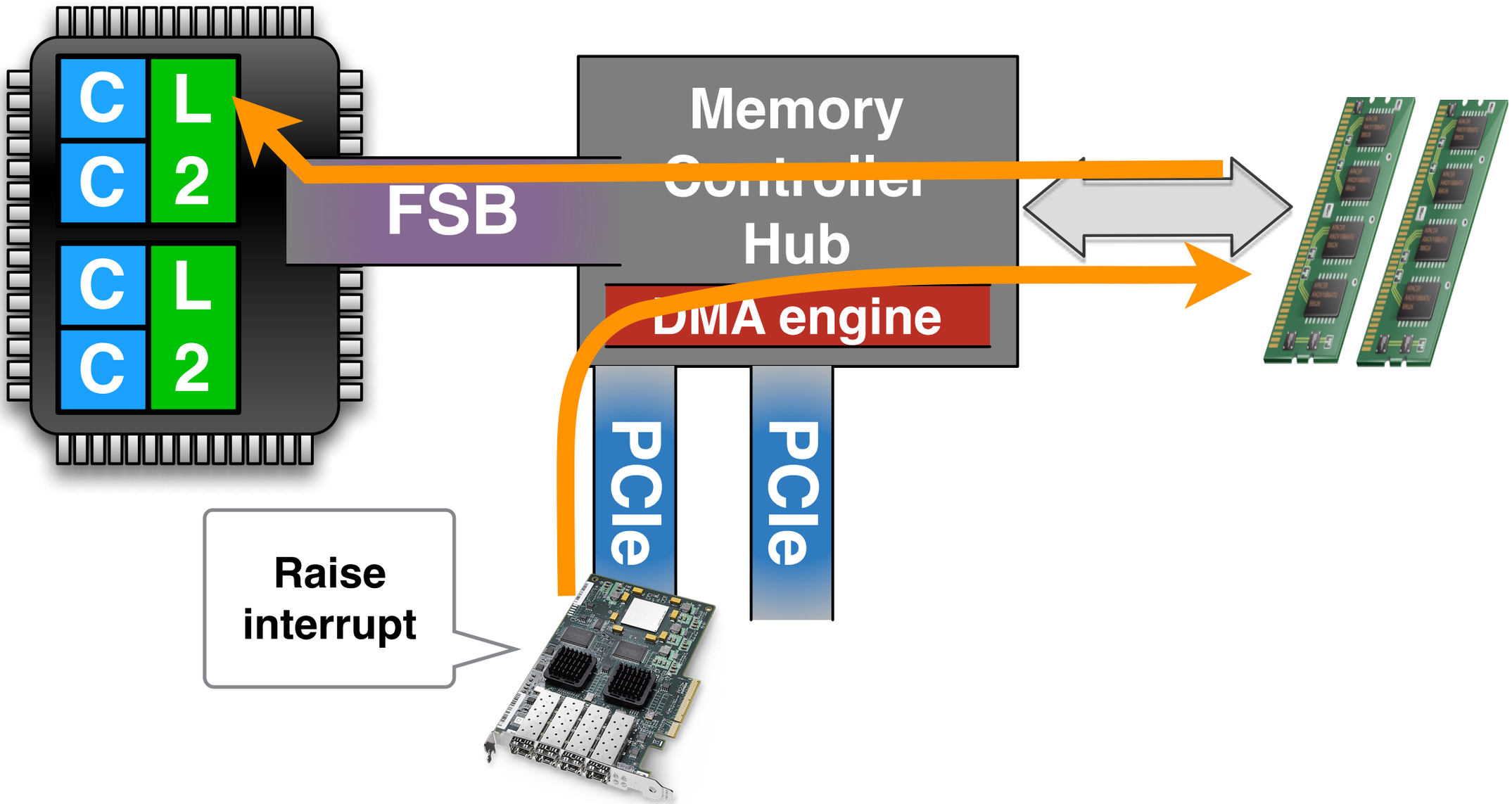
CPU microarchitecture ~2006



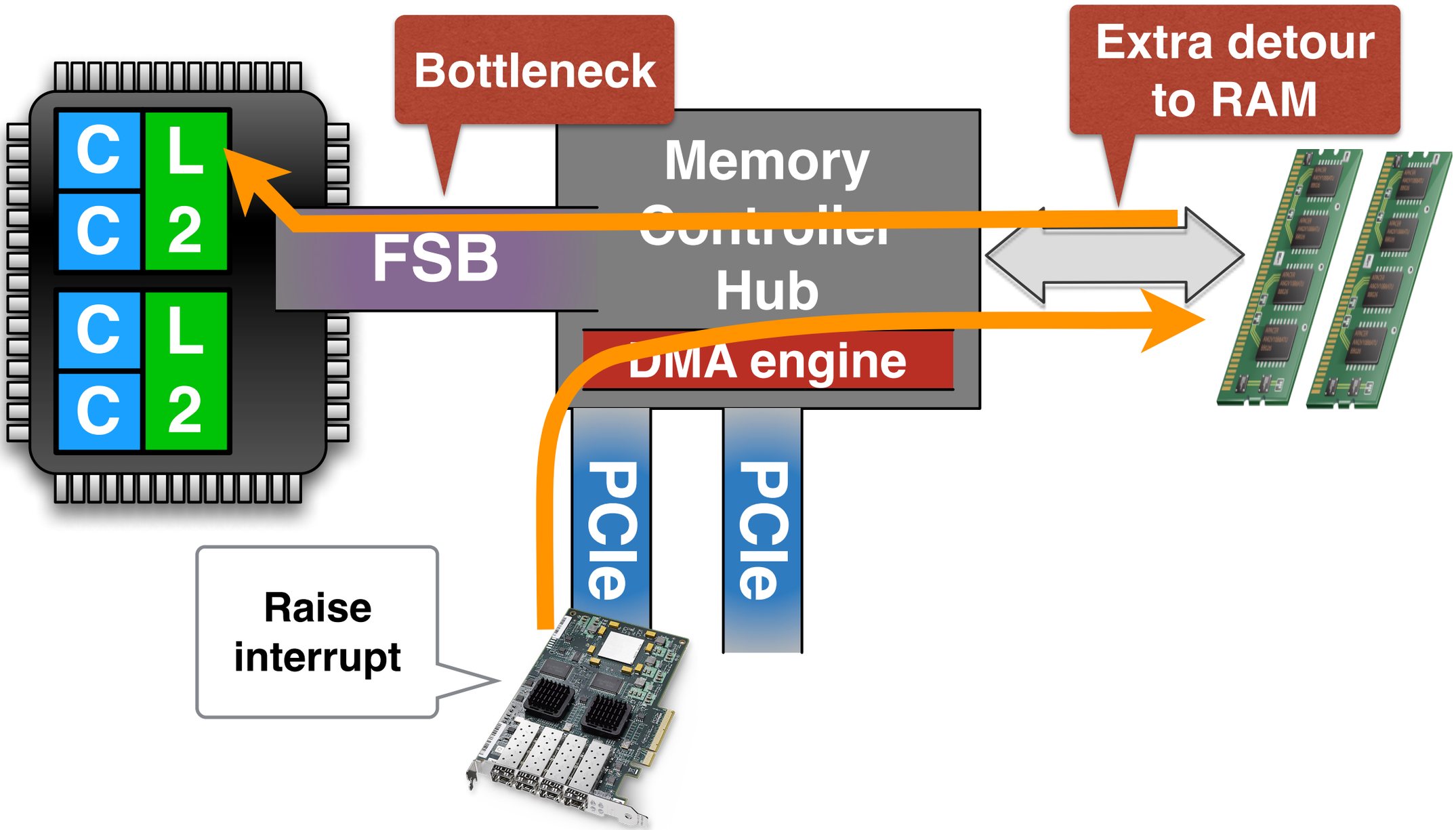
CPU microarchitecture ~2006



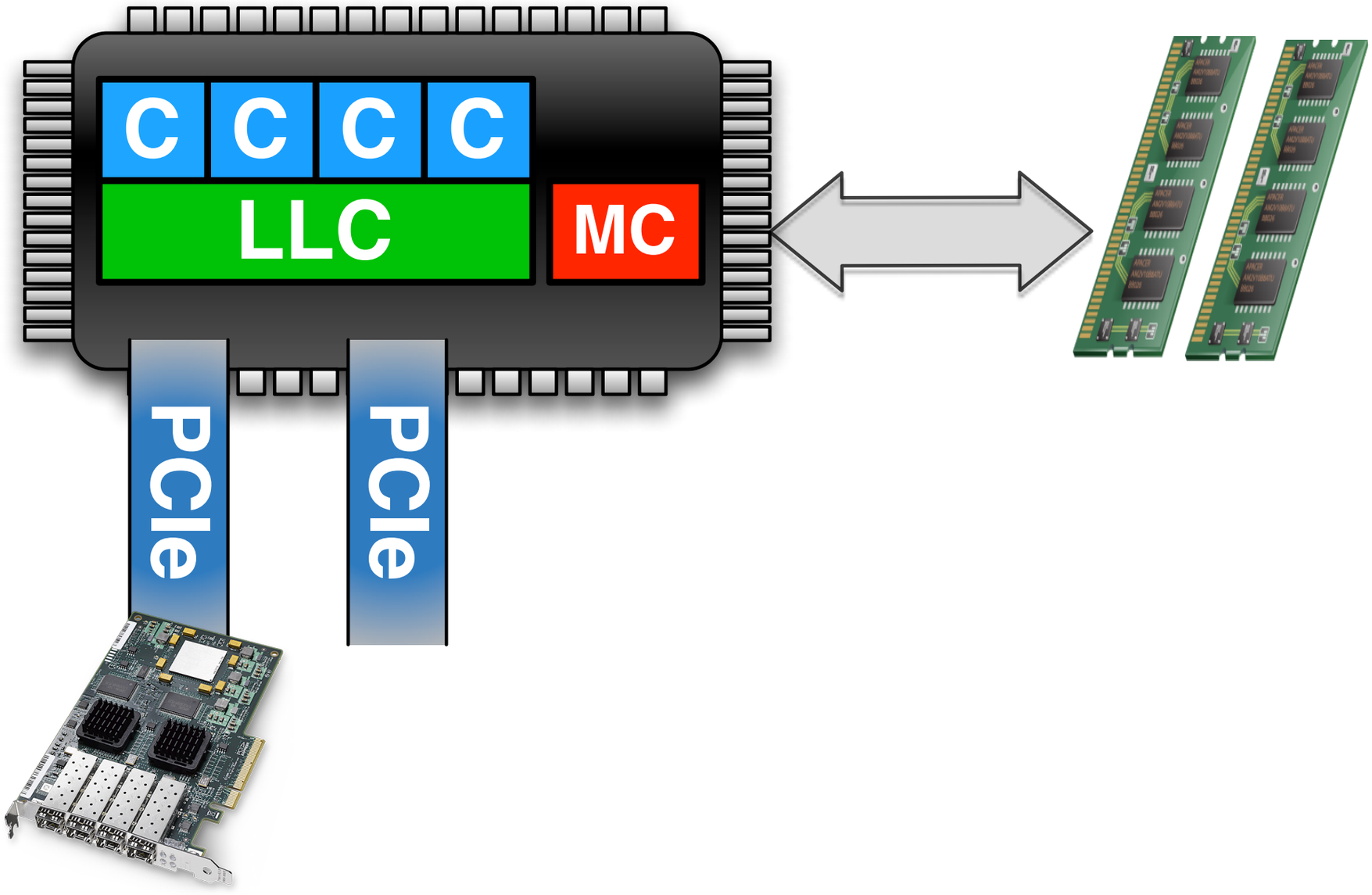
CPU microarchitecture ~2006



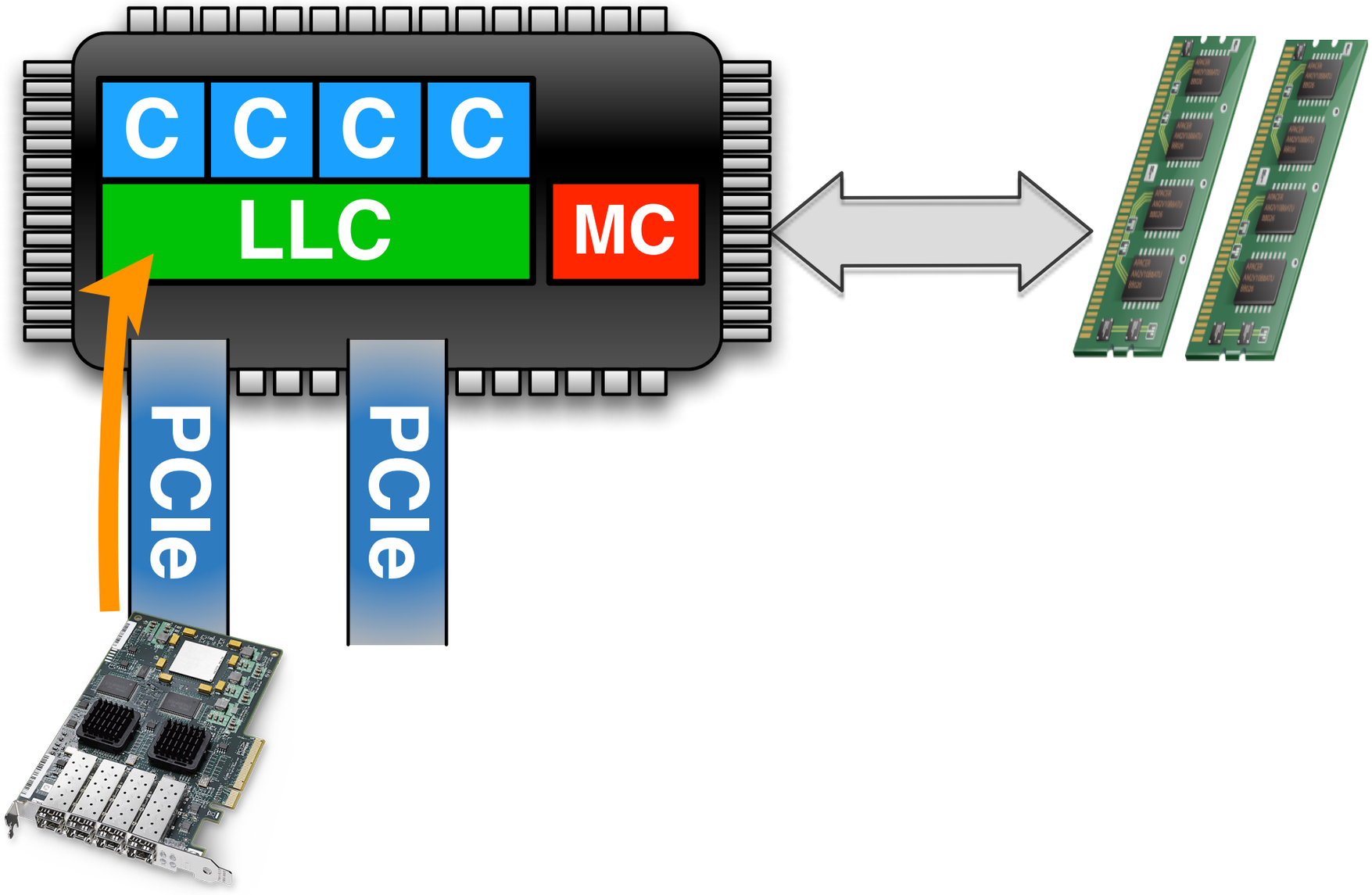
CPU microarchitecture ~2006



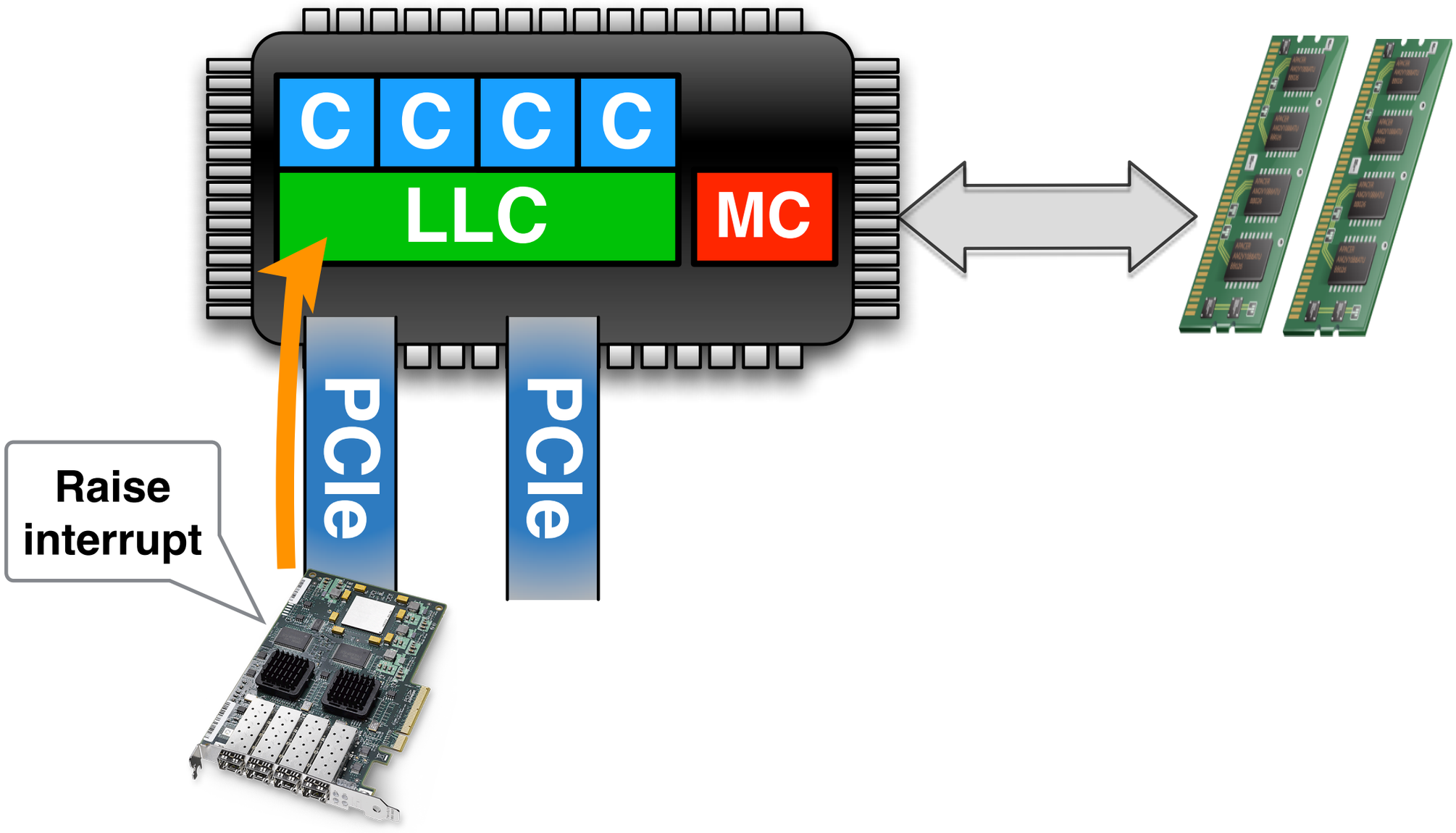
CPU microarchitecture ~2013



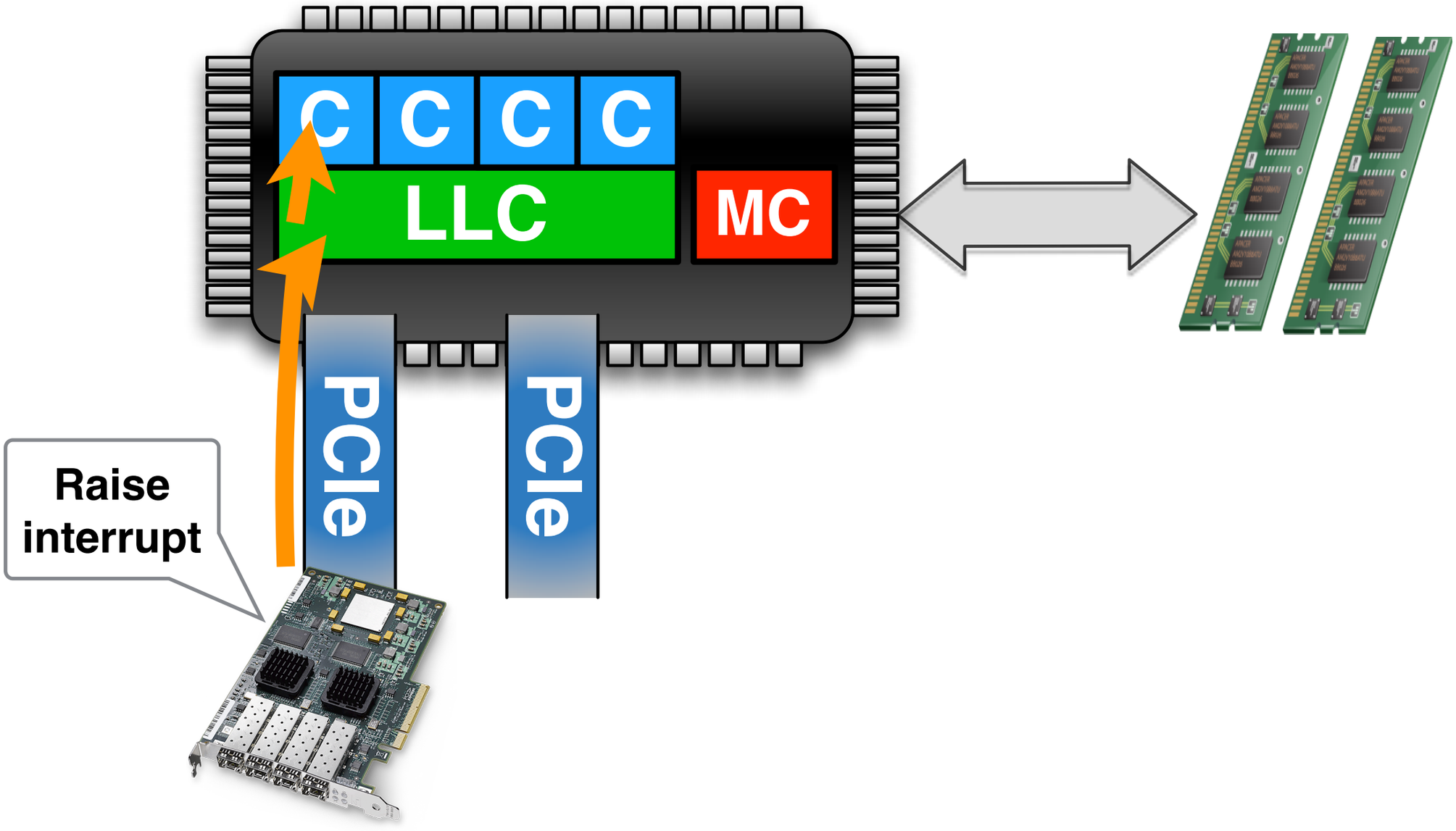
CPU microarchitecture ~2013



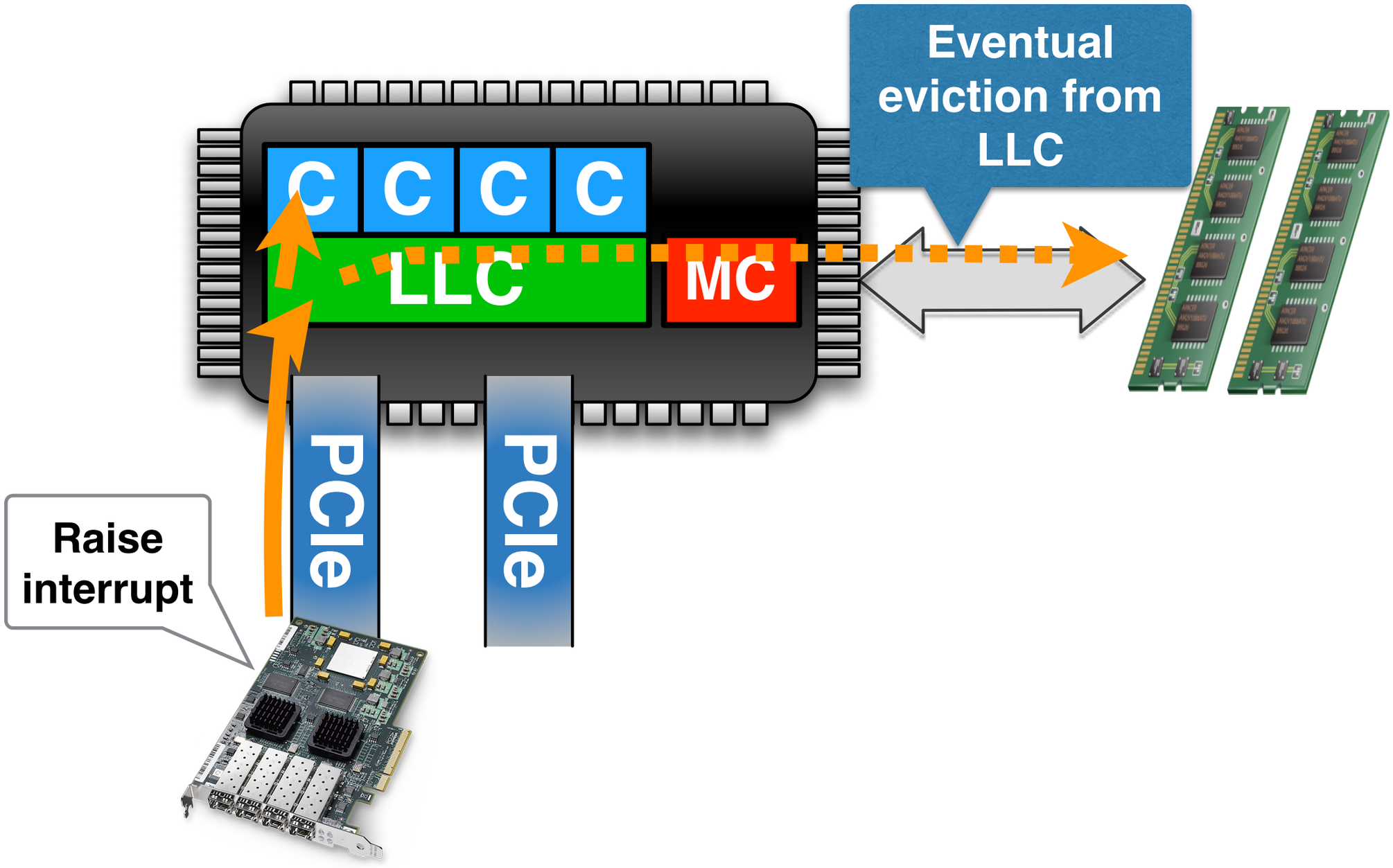
CPU microarchitecture ~2013



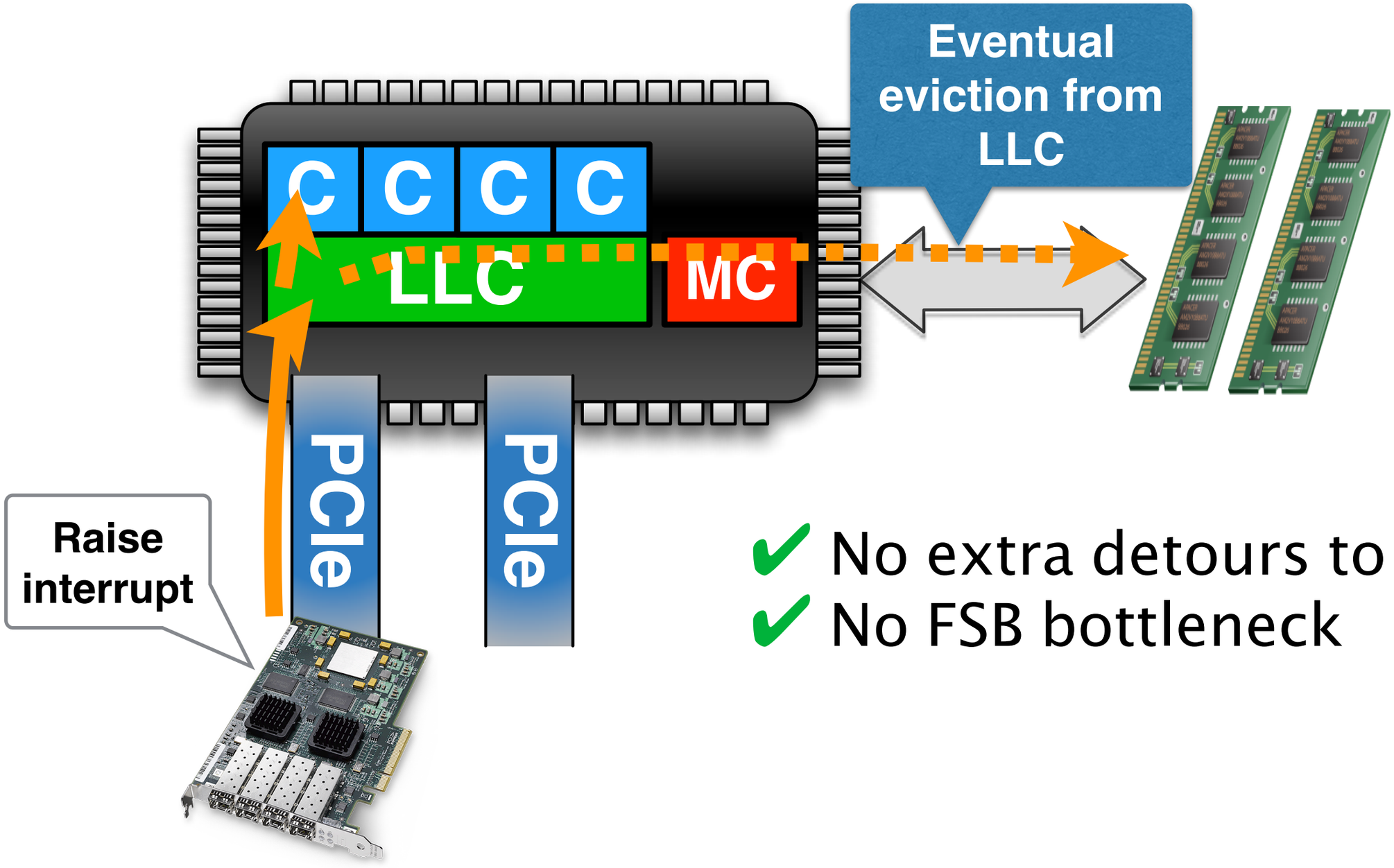
CPU microarchitecture ~2013



CPU microarchitecture ~2013

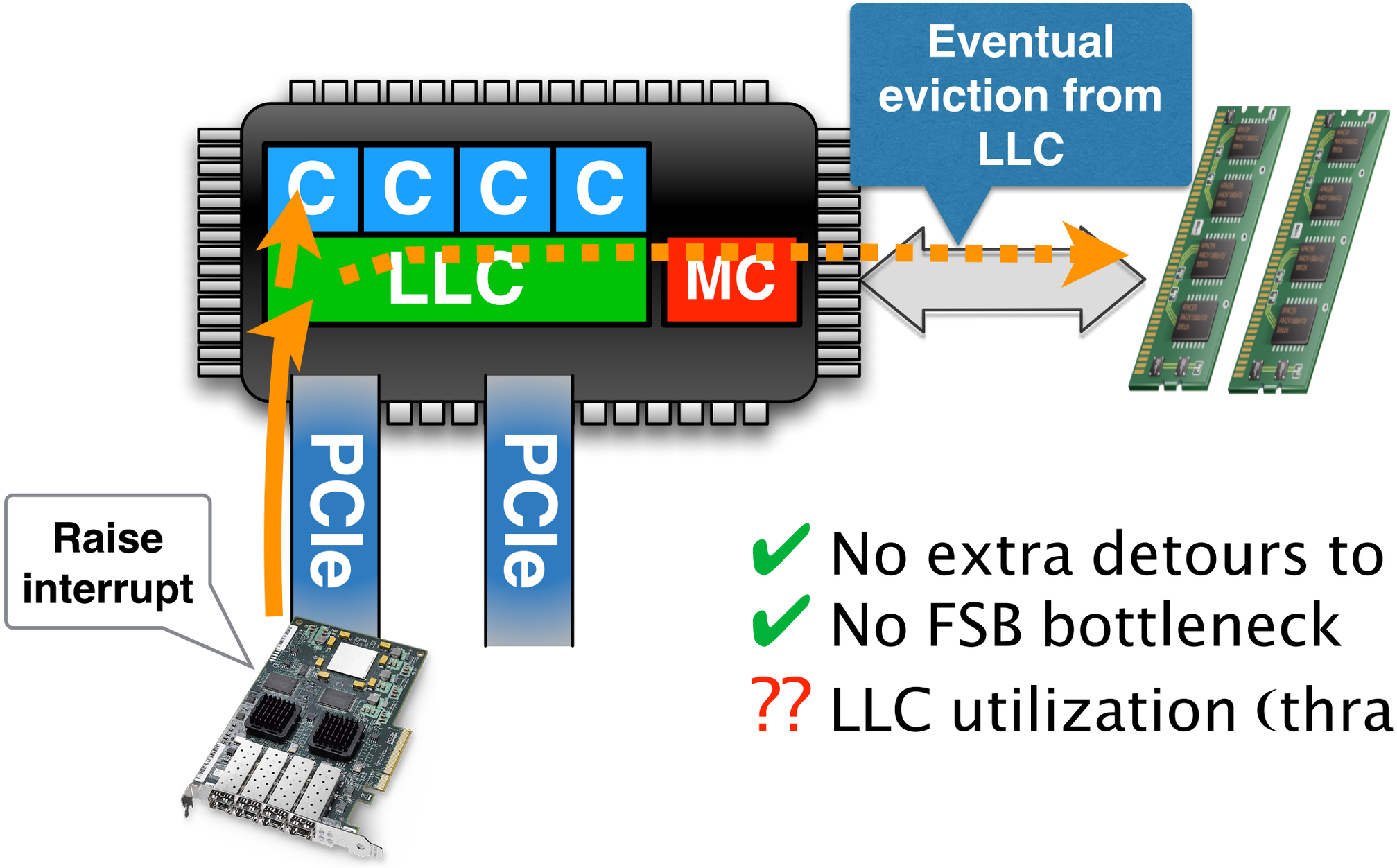


CPU microarchitecture ~2013



✔ No extra detours to DRAM
✔ No FSB bottleneck

CPU microarchitecture ~2013



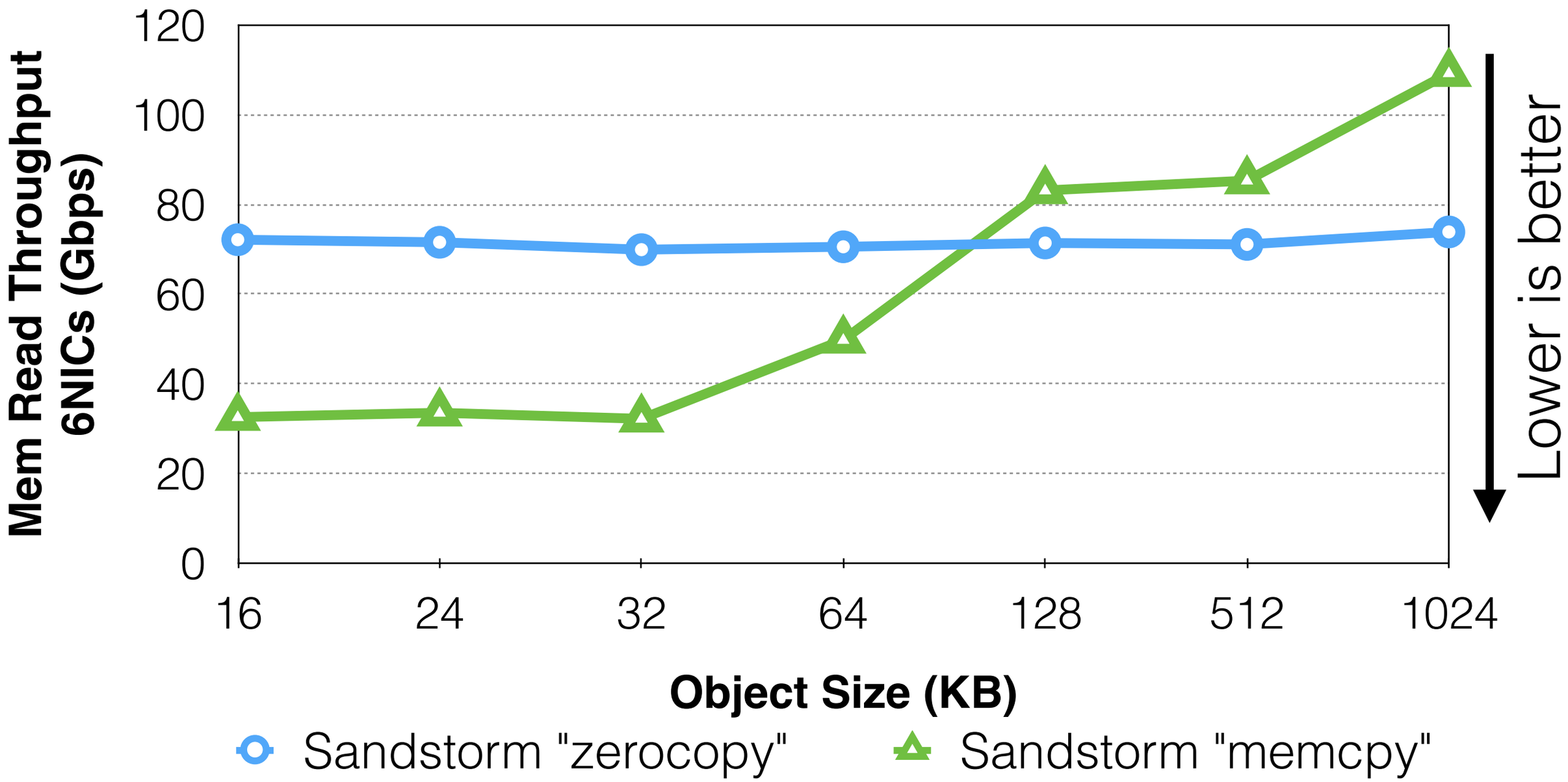
✔ No extra detours to DRAM

✔ No FSB bottleneck

?? LLC utilization (thrashing?)

HW/SW Intersection

- Should HW architecture evolution be considered a **“black box”** for networked systems development?



Generality of Specialization

Natural fit for:

- Web & DNS servers (Sandstorm, Namestorm — check our paper)
- In-memory Key-Value stores
- RPC-based services
- Rate-adaptive video streaming applications (with MPEG-DASH or Apple HLS)

Generality of Specialization

Natural fit for:

- Web & DNS servers (Sandstorm, Namestorm — check our paper)
- In-memory Key-Value stores
- RPC-based services
- Rate-adaptive video streaming applications (with MPEG-DASH or Apple HLS)

Limitations:

- Possibly not a good fit for CPU- and/or filesystem-intensive applications
- Blocking in application-layer cannot be tolerated

Conclusions

General-purpose stacks:

- Great for bulk transfers, bad for short ones (but web is dominated by small-sized objects!)
- Picked a lot of generality in favor of flexibility (we don't need it for application-specific clusters)
- Hard to tune/profile/debug

Conclusions

General-purpose stacks:

- Great for bulk transfers, bad for short ones (but web is dominated by small-sized objects!)
- Picked a lot of generality in favor of flexibility (we don't need it for application-specific clusters)
- Hard to tune/profile/debug

Specialized stacks:

- 2-10x throughput improvement for web, 9x for DNS
- Linear scaling on multicore systems
- Low CPU utilization

Conclusions

General-purpose stacks:

- Great for bulk transfers, bad for short ones (but web is dominated by small-sized objects!)
- Picked a lot of generality in favor of flexibility (we don't need it for application-specific clusters)
- Hard to tune/profile/debug

Specialized stacks:

- 2-10x throughput improvement for web, 9x for DNS
- Linear scaling on multicore systems
- Low CPU utilization

Specialized network stacks not only viable, but necessary!

Backup Slides

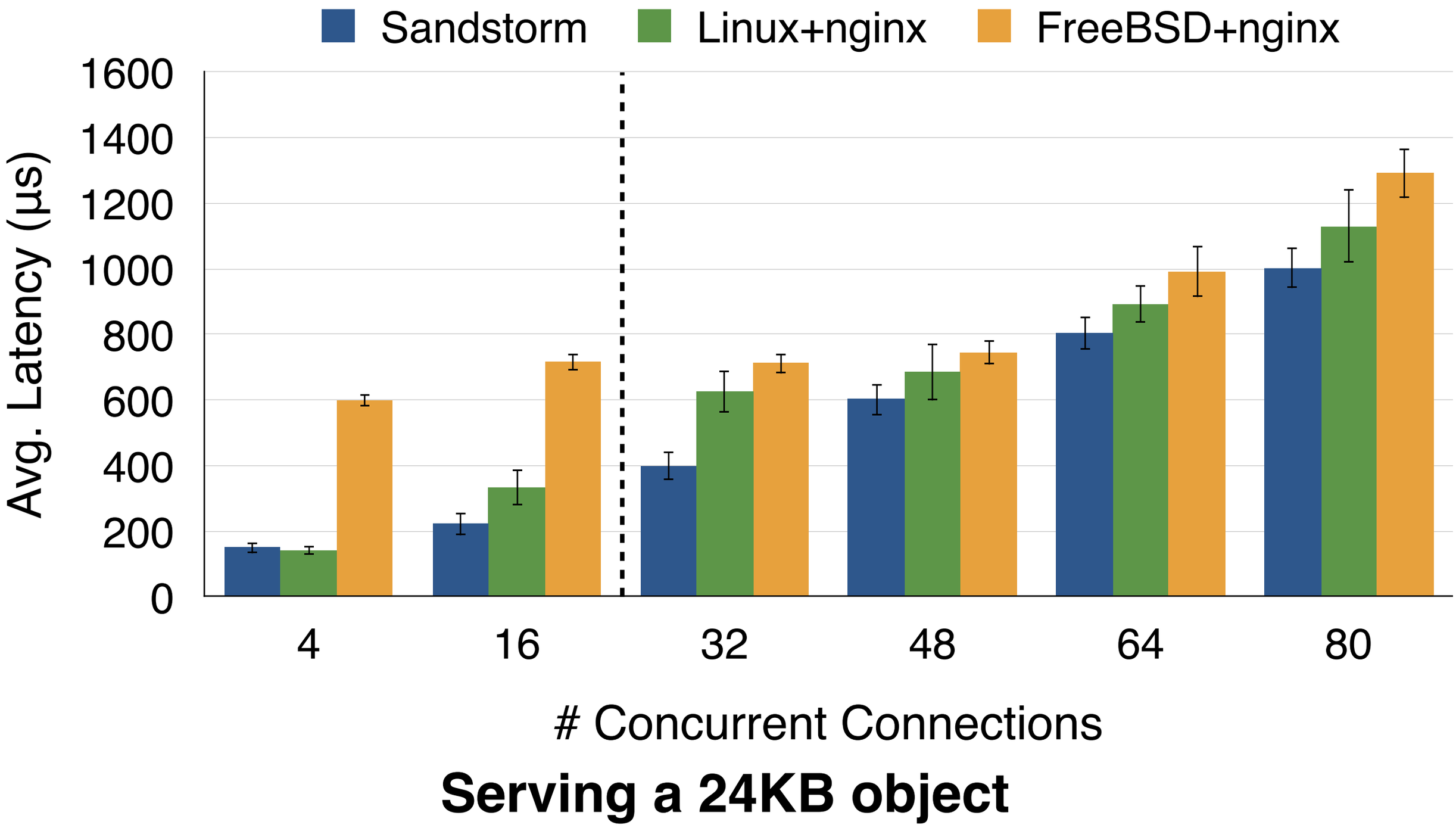
Supported TCP features

- Follows RFC 793, with Reno congestion control

Limitations:

- Support of the required TCP subset to serve incoming connections (not initiating them)
- TCP reordering not supported (not needed with typical HTTP requests)

Latency



Serving a 24KB object

Overview

Problems with general-purpose stacks:

- System-call overhead
- Shared accept-queue, PCB locks
- Cache-unfriendly due to async. design
- Memory-related overhead (e.g., mbuf alloc./copying)

Solutions with specialized stacks:

- Packet batching
- Share- & Lock-free design, per-core state
- Process-to-completion, cache-friendly, incr. cksum
- Pre-packetization, no memory copying/buffering