# Making Kernel Bypass Practical for the Cloud with Junction

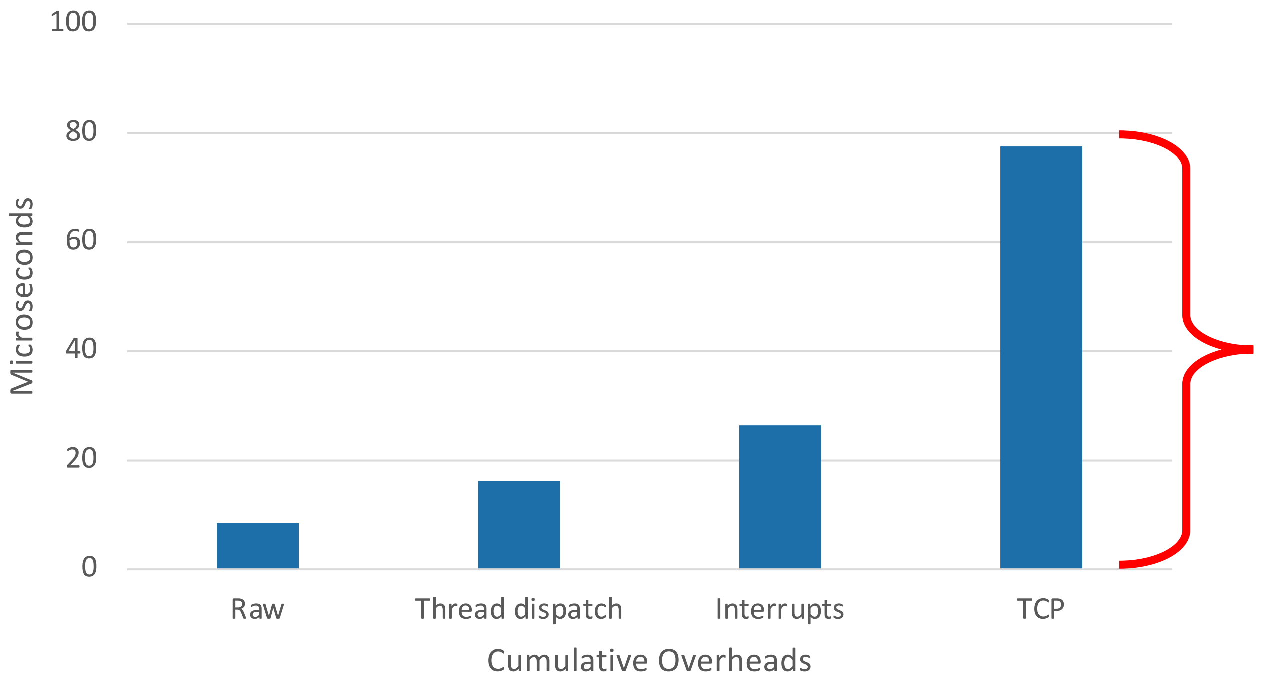
**Joshua Fried (MIT)**, Gohar Irfan Chaudhry, Enrique Saurez, Esha Choukse, Íñigo Goiri, Sameh Elnikety, Rodrigo Fonseca, and Adam Belay

NSDI 2024

# Motivation: high I/O overheads

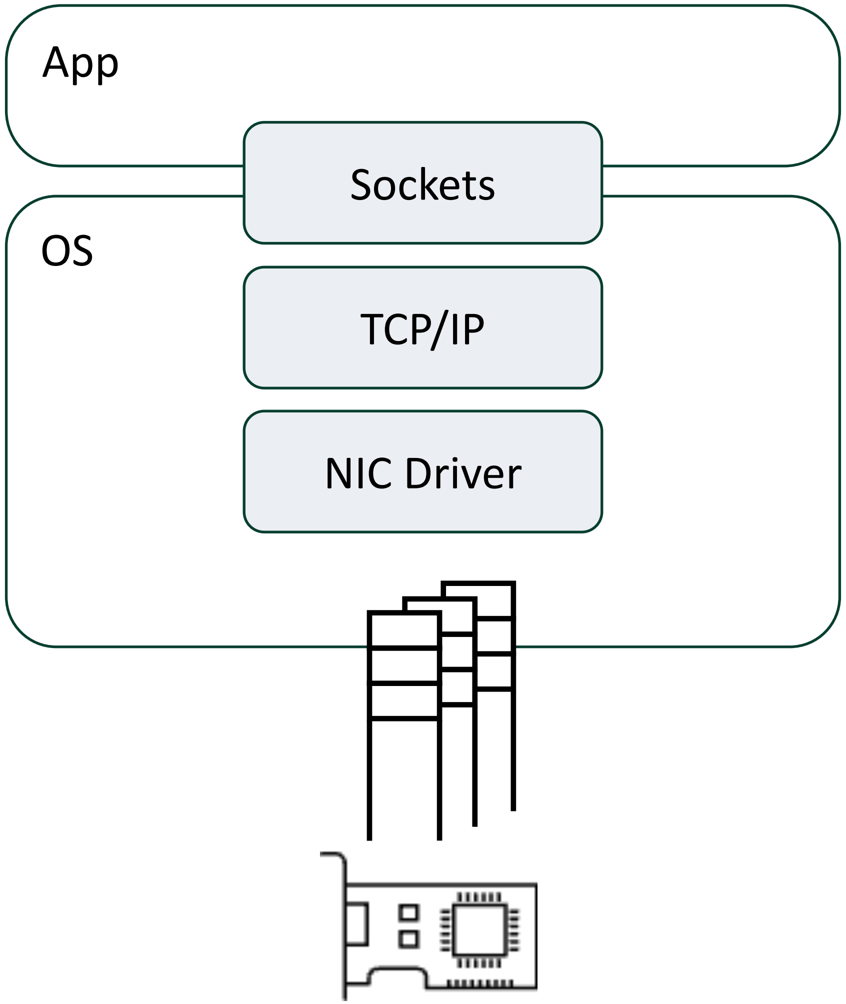
- Datacenters today:
  - High network bandwidths, microsecond-scale latencies

- OS kernels add large overheads to I/O operations
  - Applications can't exploit full hardware performance



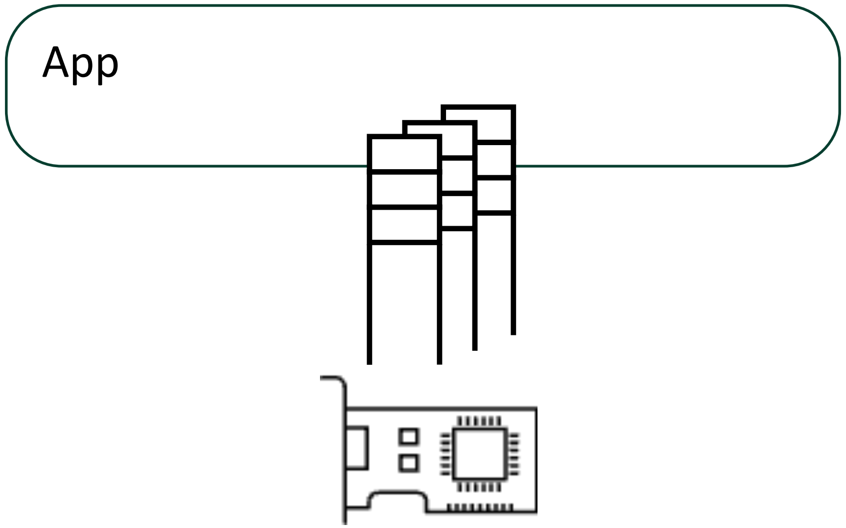

*Attack of the killer microseconds, Barroso et al. CACM 2017*

# Solution: bypass the kernel?



# Solution: bypass the kernel?

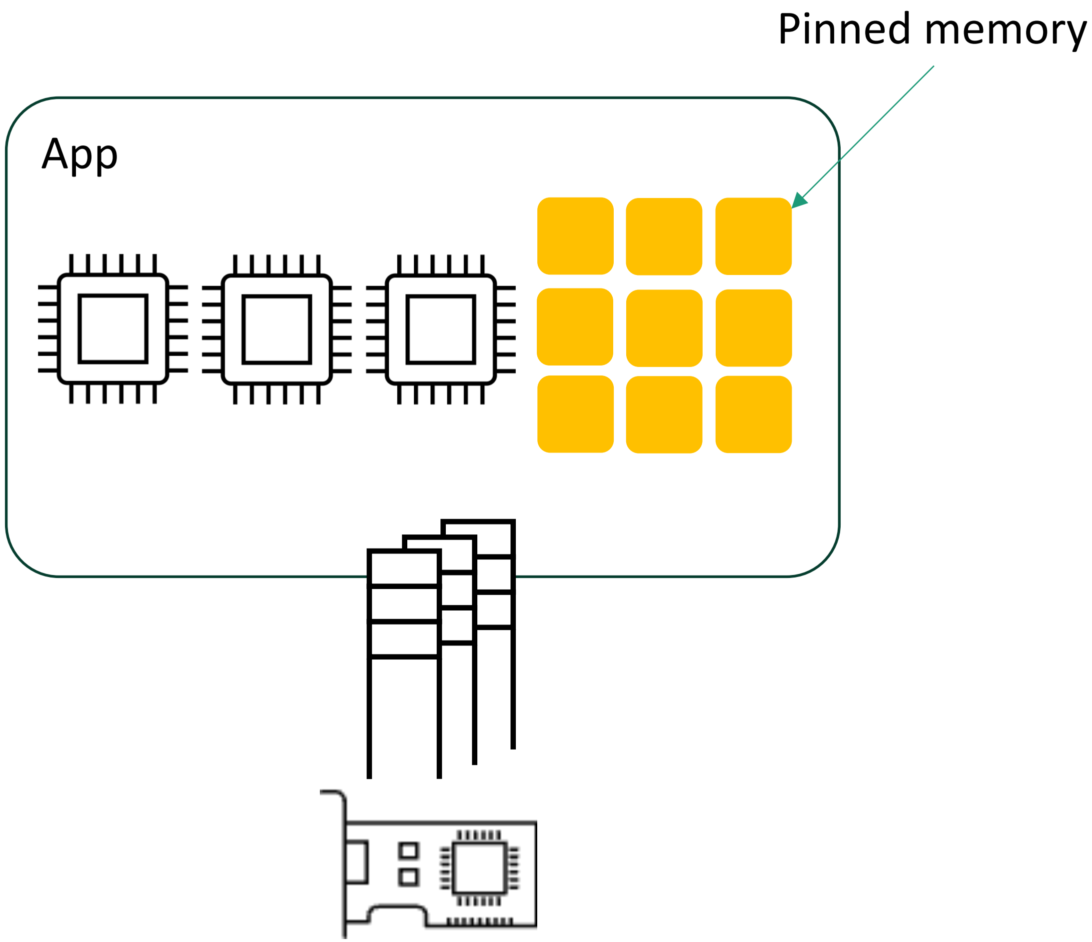
Map packet queues directly into application's address space



# Solution: bypass the kernel?

Map packet queues directly into application's address space

Pre-assign cores and memory to get the OS out of the way

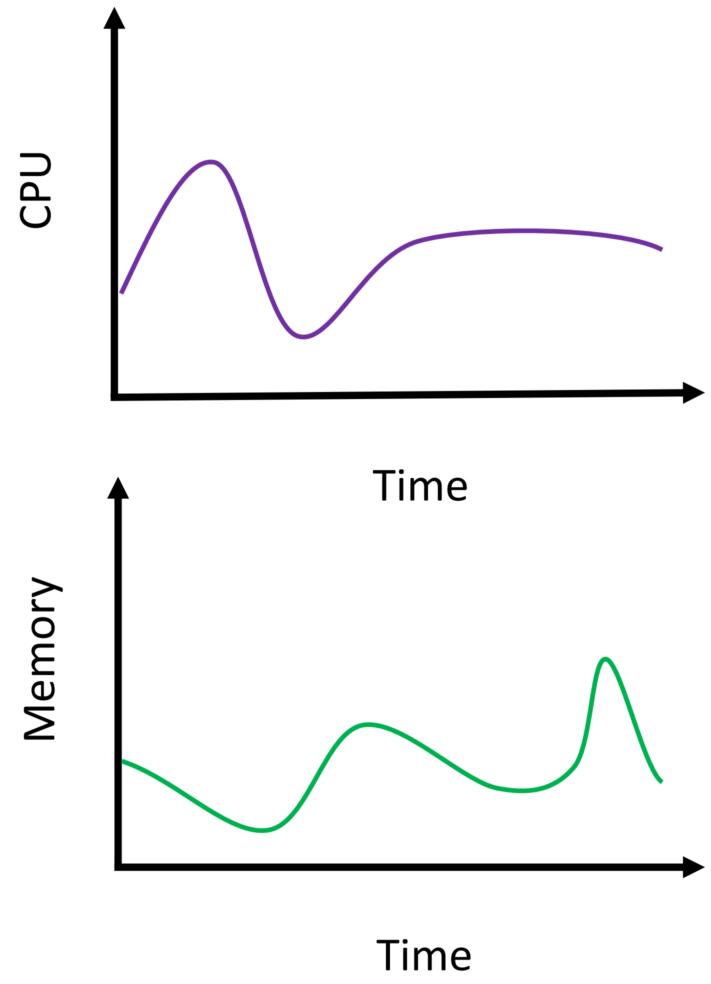


# Drawback #1: Density

Resource usage varies over time

Densely packing applications keeps utilization high

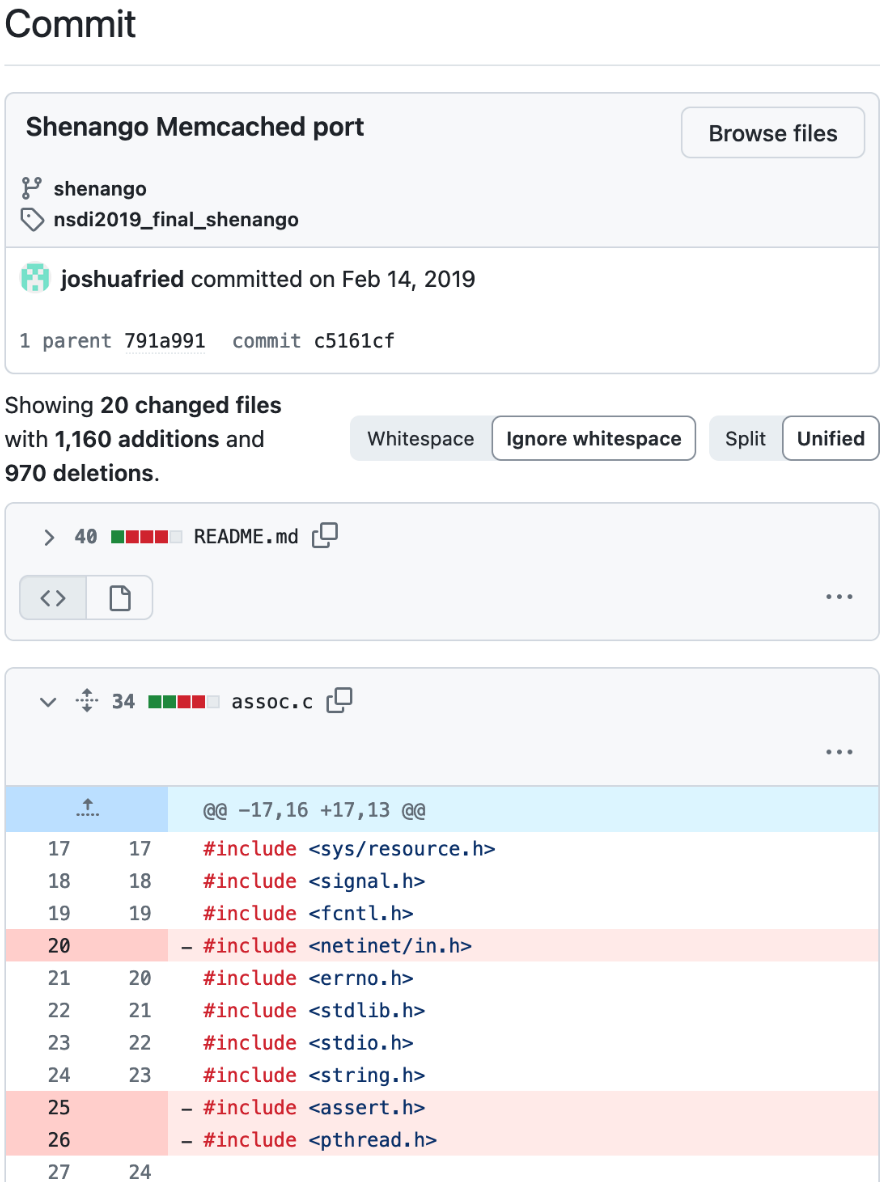
Pre-assigned resources can't be shared between applications



# Drawback #2: Compatibility

Rewriting all applications would waste prior investments in software engineering

Only a small handful of applications have been ported to run on today's kernel bypass systems!



# Junction's contributions

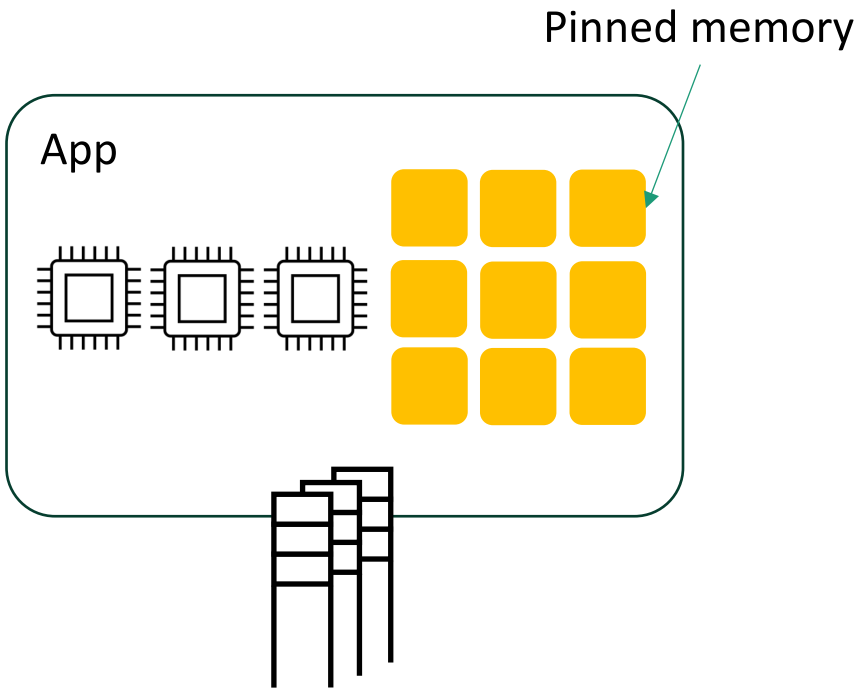
- Enabling dense deployment of kernel bypass apps
  - Buffer management scheme to reduce pinned memory
  - NIC-assisted core scheduling to avoid polling
- Achieving compatibility with unmodified apps
  - Userspace implementation of Linux syscall interface using kernel bypass
  - Modified libc to convert syscall instructions to function calls
  - User Interrupts to implement POSIX signals
  - Optimizations to avoid sacrificing performance
    - Use newer CPU instructions (WRFSBASE, RDRAND) to avoid syscalls
    - Exploit fate sharing to remove security overheads

# Density

Goal: pack thousands of kernel bypass apps on a machine

**Memory:** Reduce pinned memory to fit more instances

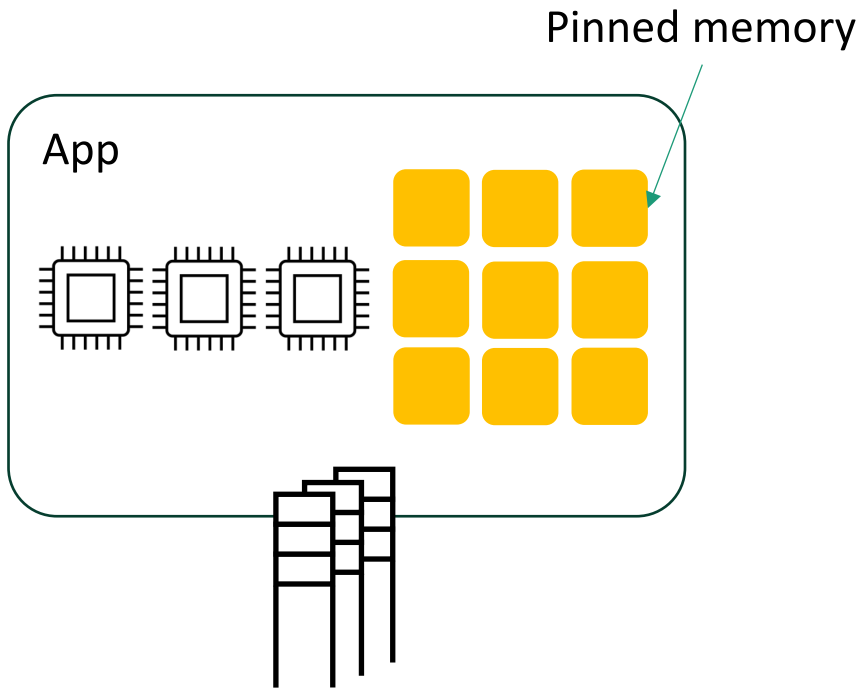
**Cores:** Avoid spin polling so cores can be shared



# Pinned memory

- Receive path: need enough buffers to absorb bursts and accommodate delays

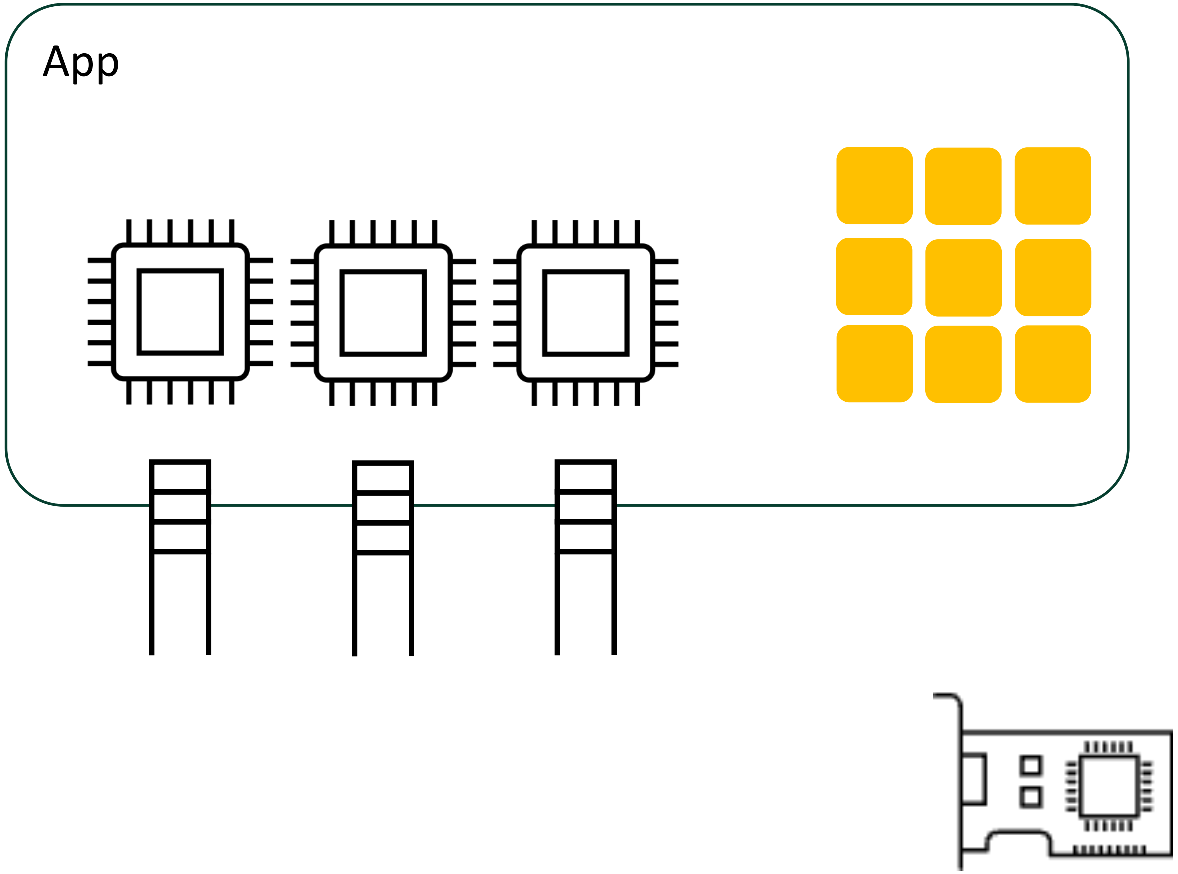
- Two traffic patterns responsible for wasted pinned memory
  - Skewed traffic
  - Small packets



# Pattern #1: skewed traffic

1 queue per-core to scale packet processing

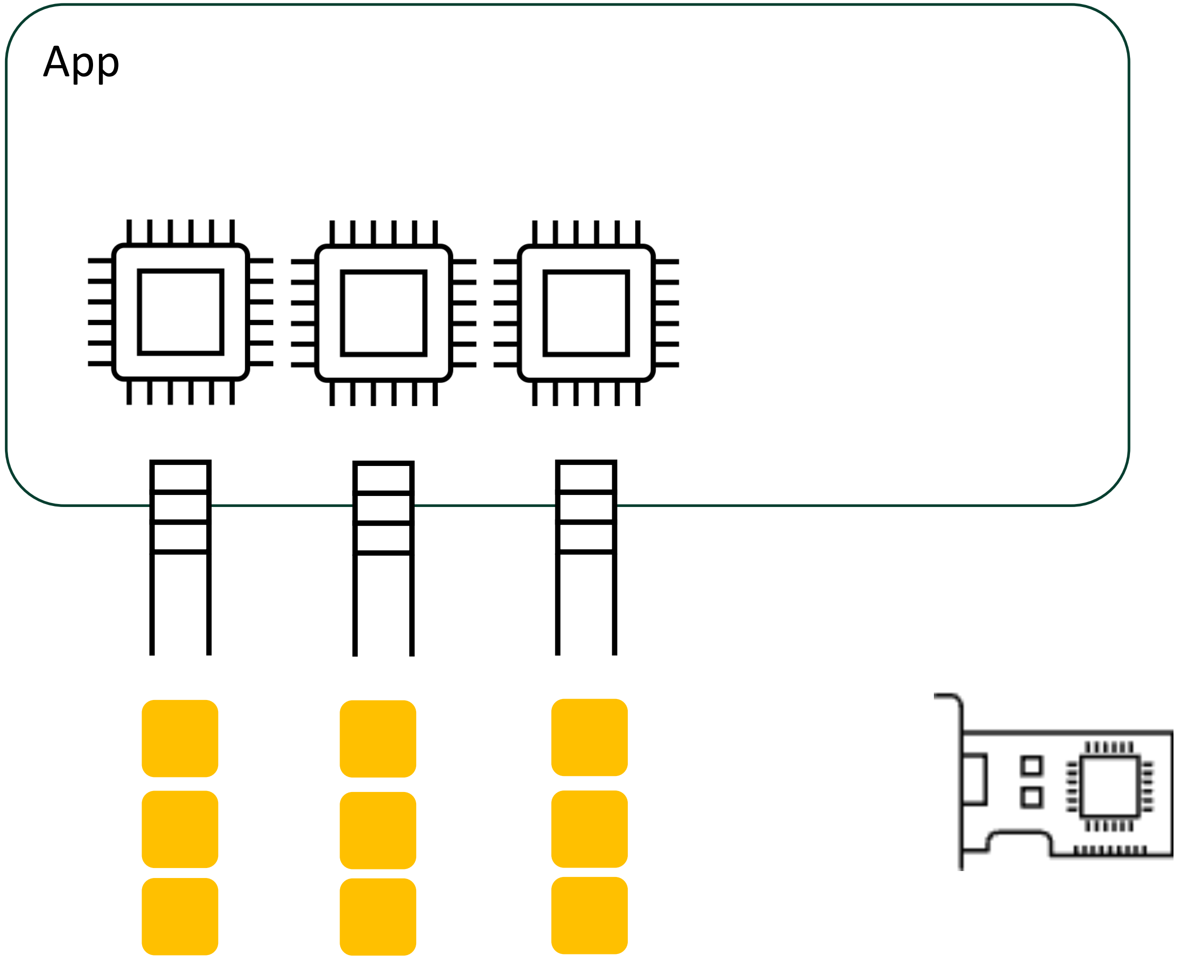
- Each queue has a set of posted buffers
- RSS assigns incoming packets to queues
- Potential for skew requires provisioning enough buffers to all queues



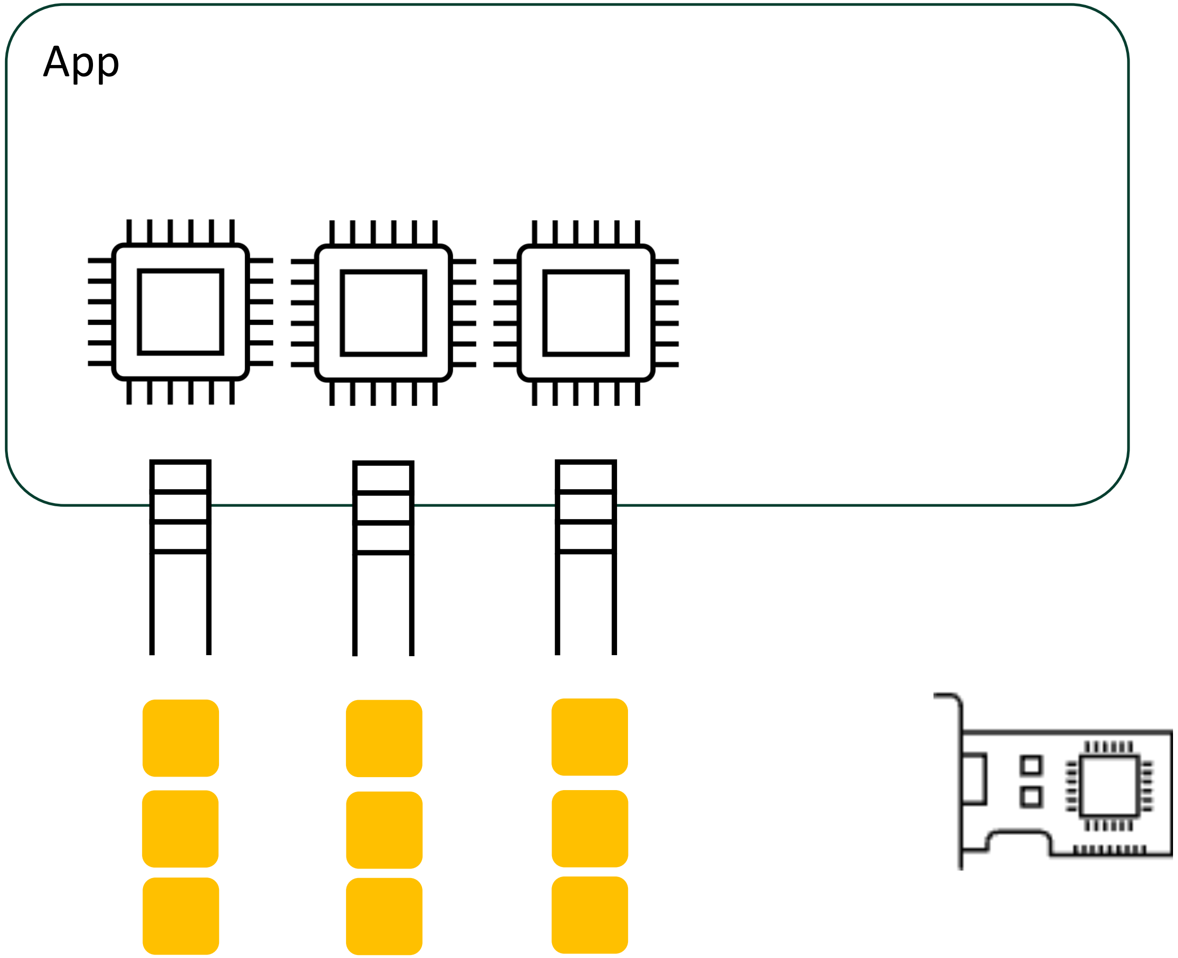
# Pattern #2: small packets

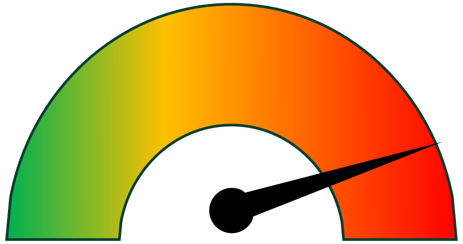
All packets consume a whole buffer regardless of size

- Results in memory fragmentation
- High rates of small packets have greater buffering needs



# Can we use NIC queues differently?

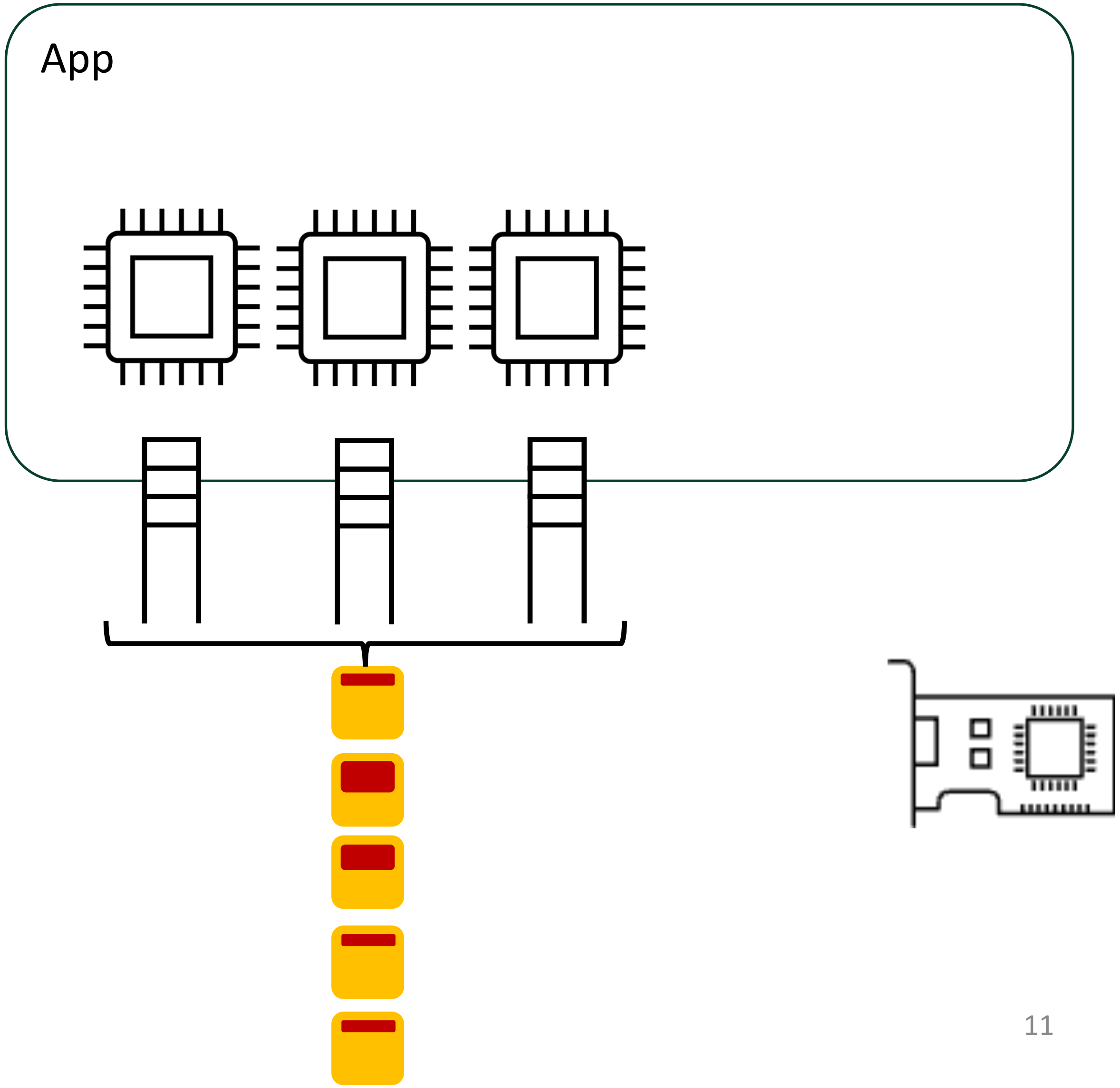


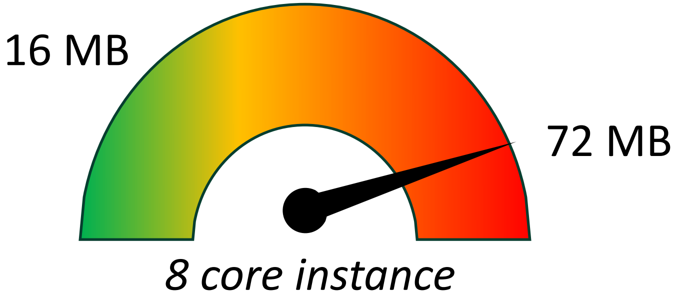




*8 core instance*

# Can we use NIC queues differently?

1. Sharing a buffer queue

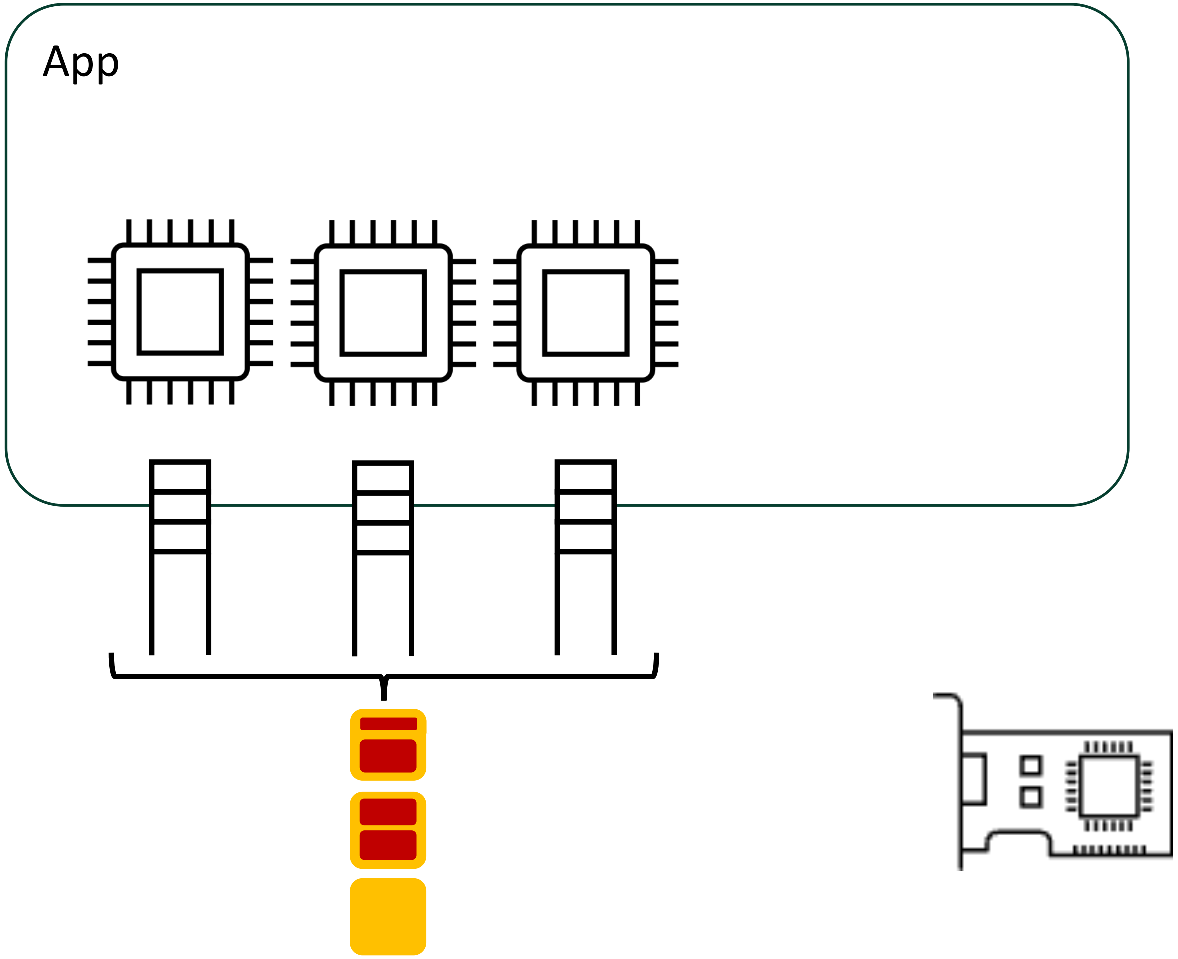


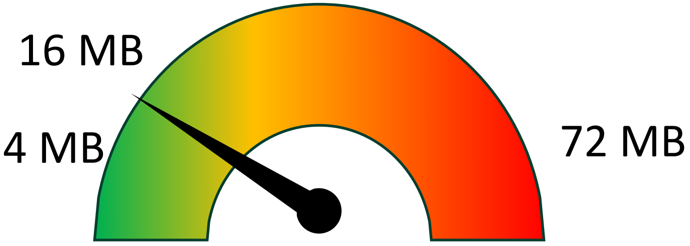



# Can we use NIC queues differently?

1. Sharing a buffer queue

2. Multiple packets per buffer

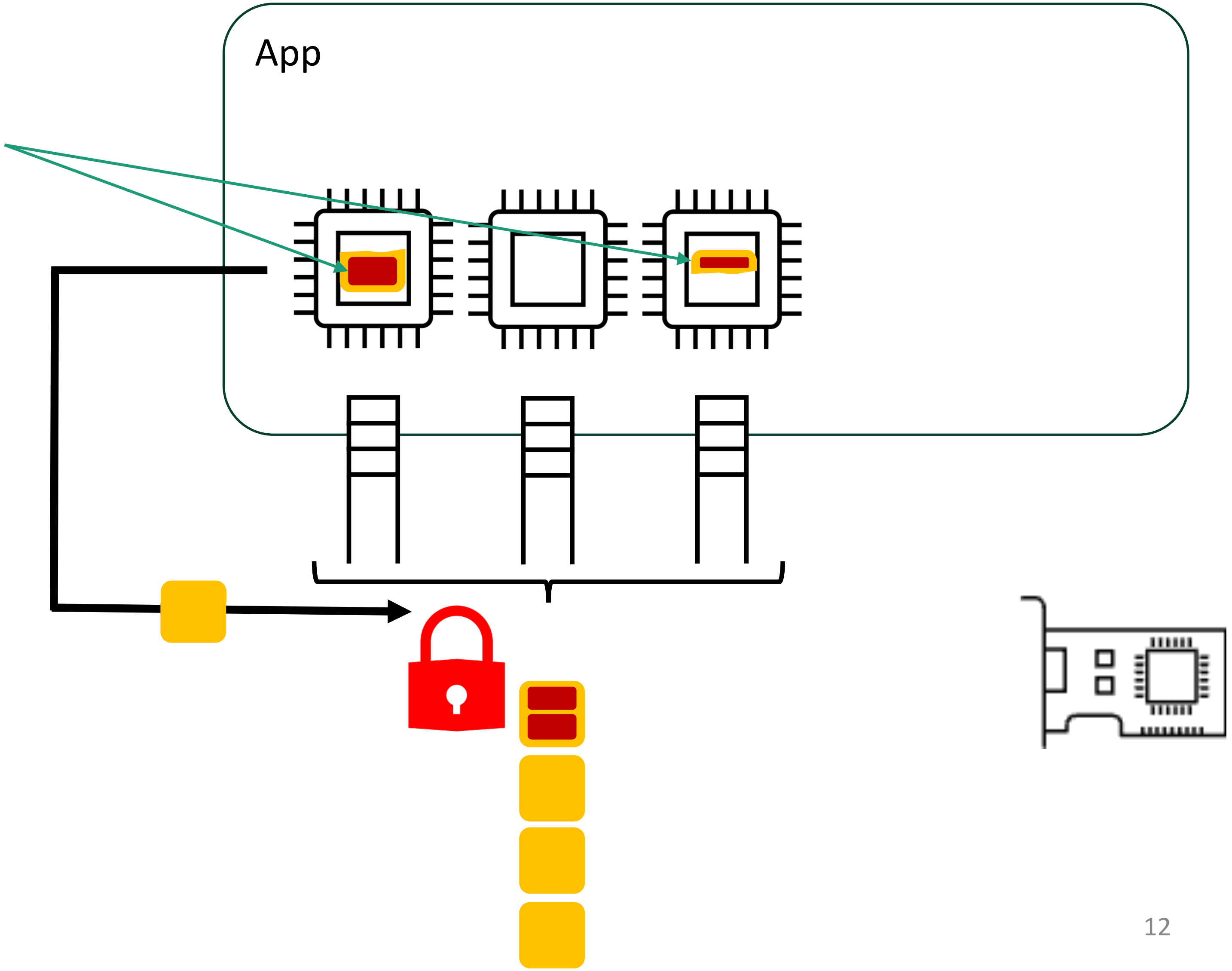


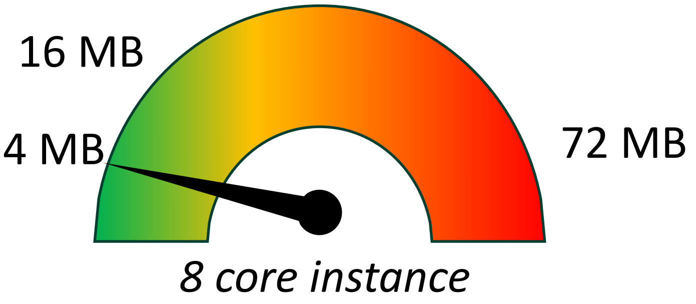




*8 core instance*

# Sharing requires synchronizing

- Buffer can't be reused until all cores are done with it
- Updates to shared buffer queue must be synchronized

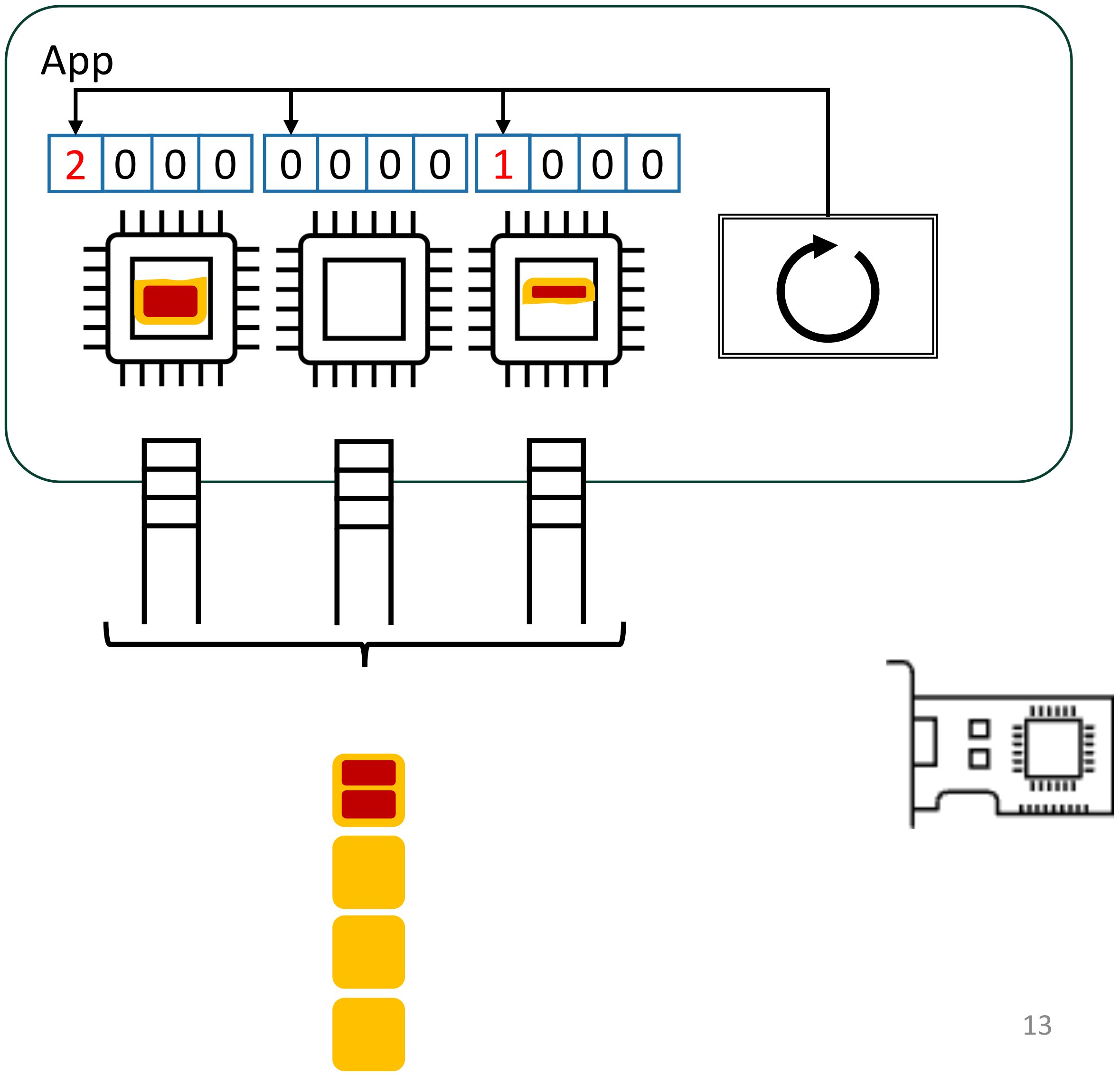


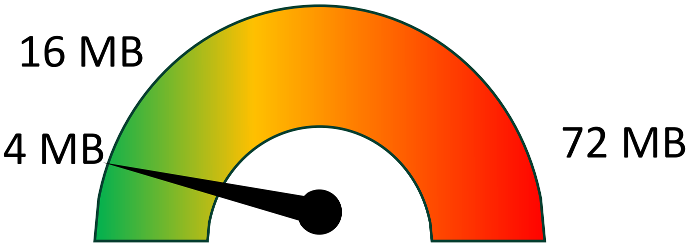



# Synchronization-free refill

Per-core buffer reference counters

Refill thread scans counters and re-posts buffers



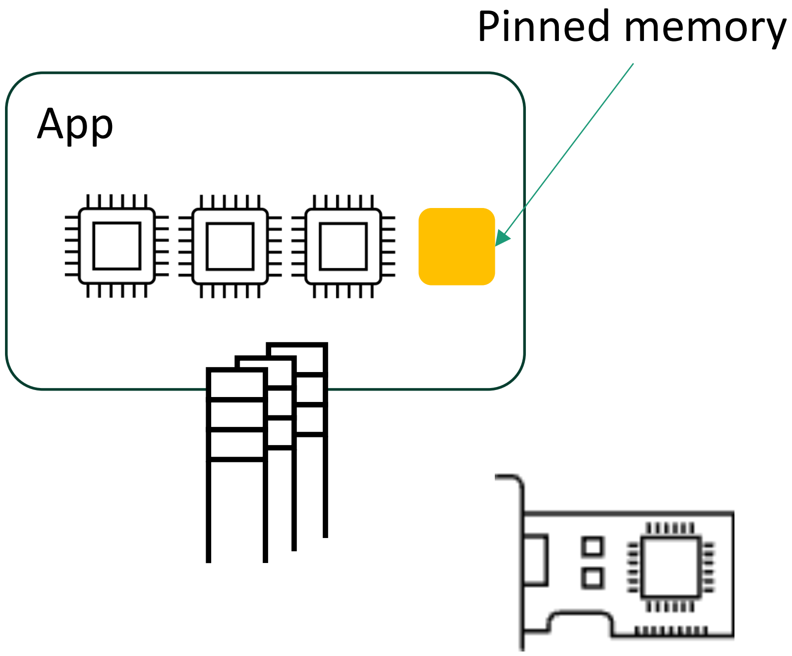

*8 core instance*

# Density

Goal: pack thousands of kernel bypass apps on a machine

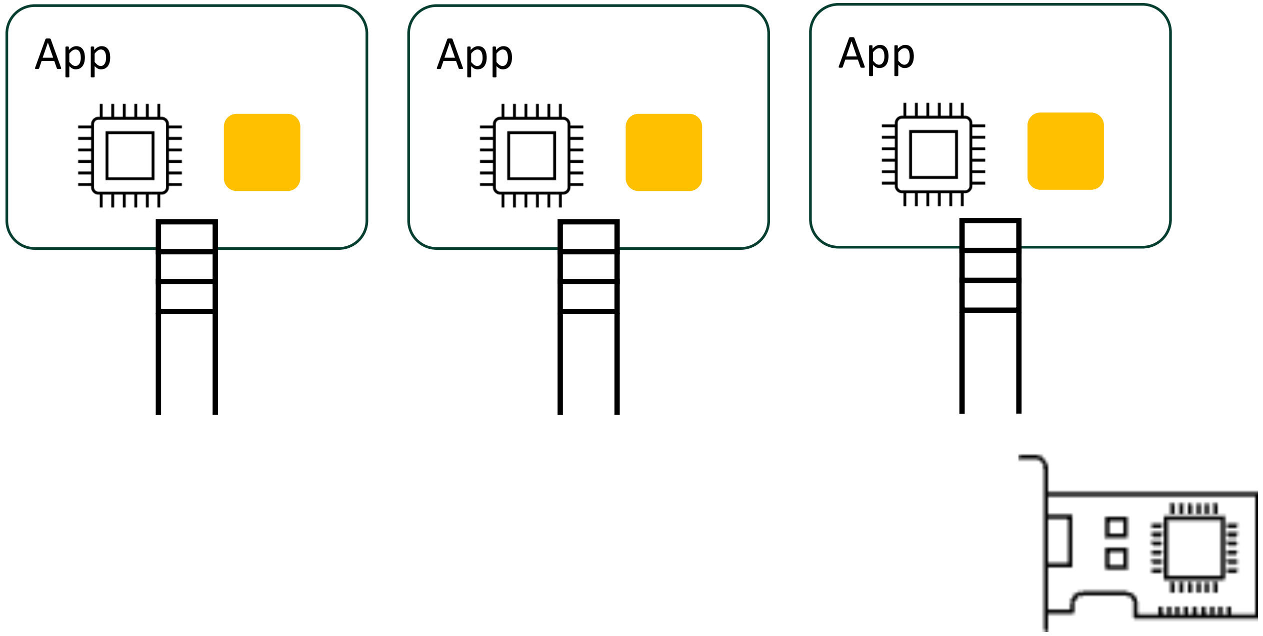
**Memory:** Reduce pinned memory to fit more instances ✓

**Cores:** Avoid spin polling so cores can be shared



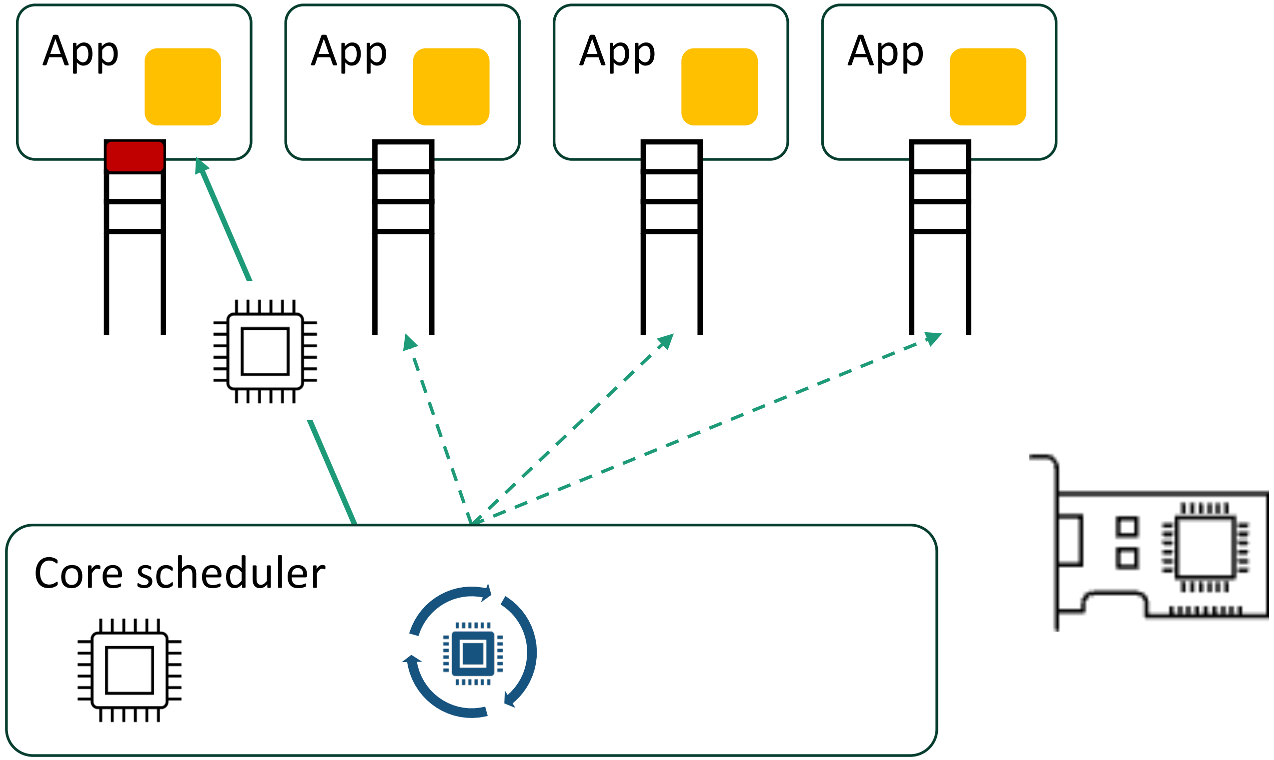
# Kernel bypass usually spin polls

- Achieves low latency
- Opportunity to share cores between packet arrivals



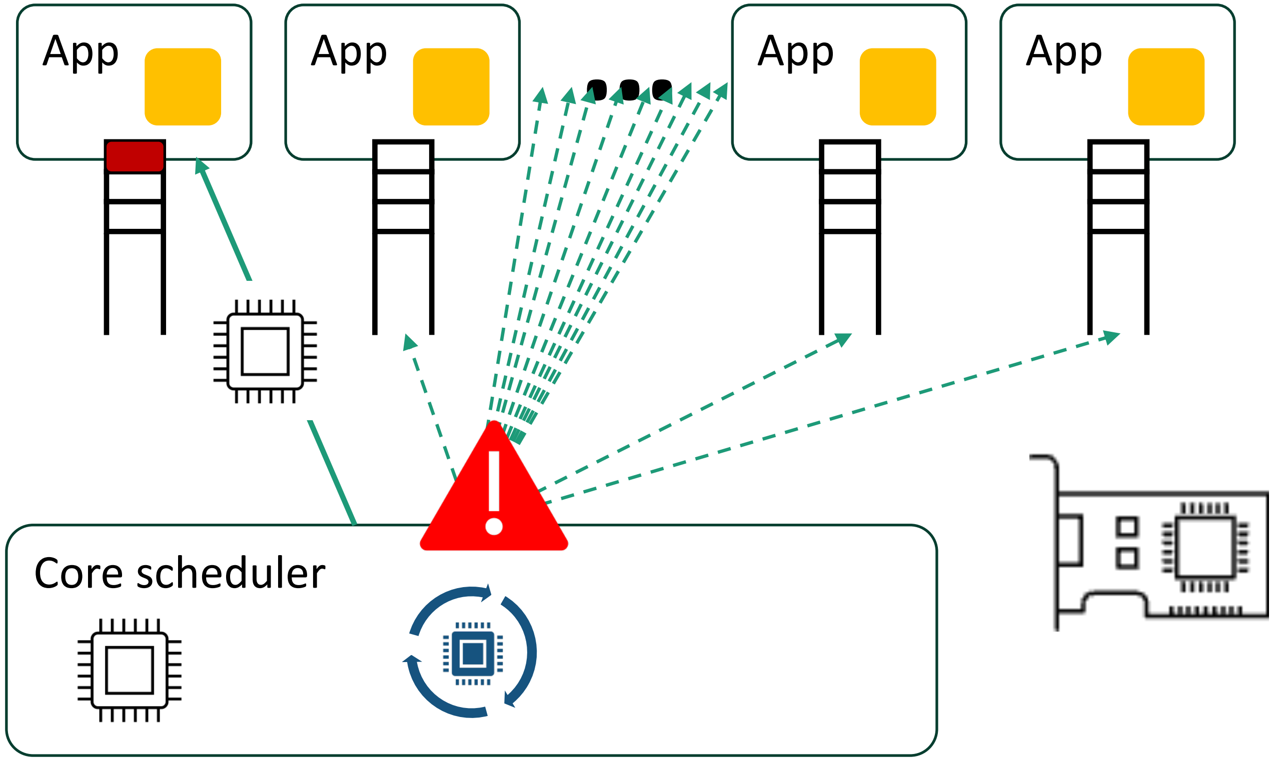
# Recent work: delegating polling

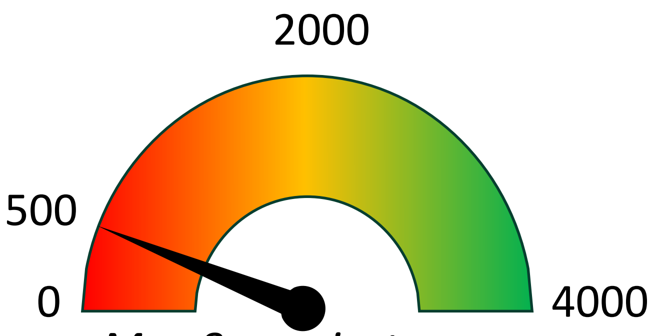
- Single spinning scheduler core decides core assignments
- Spin polls queues on behalf of idle applications
  - Wake-up on packet arrival or app-specified timeout



# Problem: polling bottleneck

- Performance collapse with many queues
  - Long delays from polling loop
  - Scheduler’s cache becomes polluted

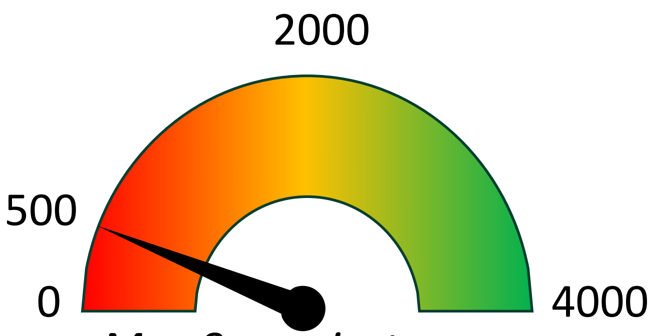


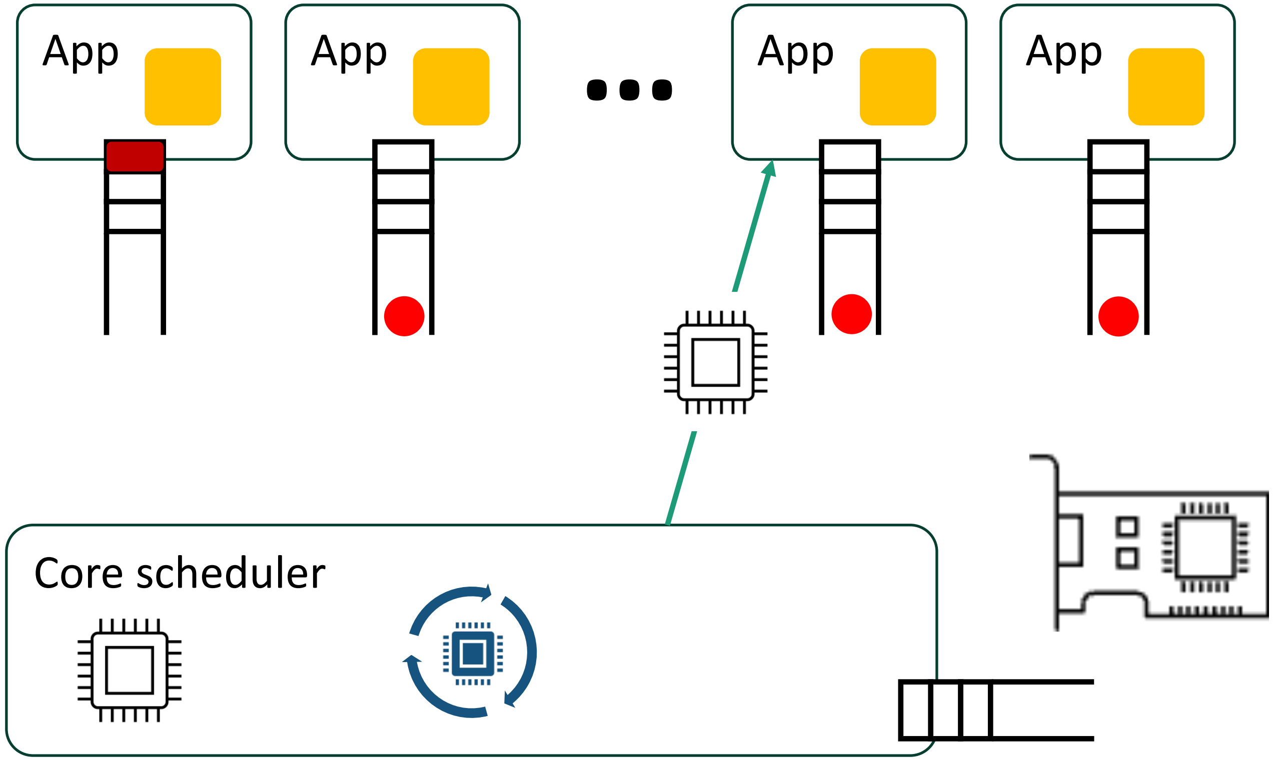

*Max 8 core instances*

# Solution: NIC notifications

- Scheduler polls a *notification queue*
- Idle queues are armed
- Packet arrivals on armed queues generate notifications



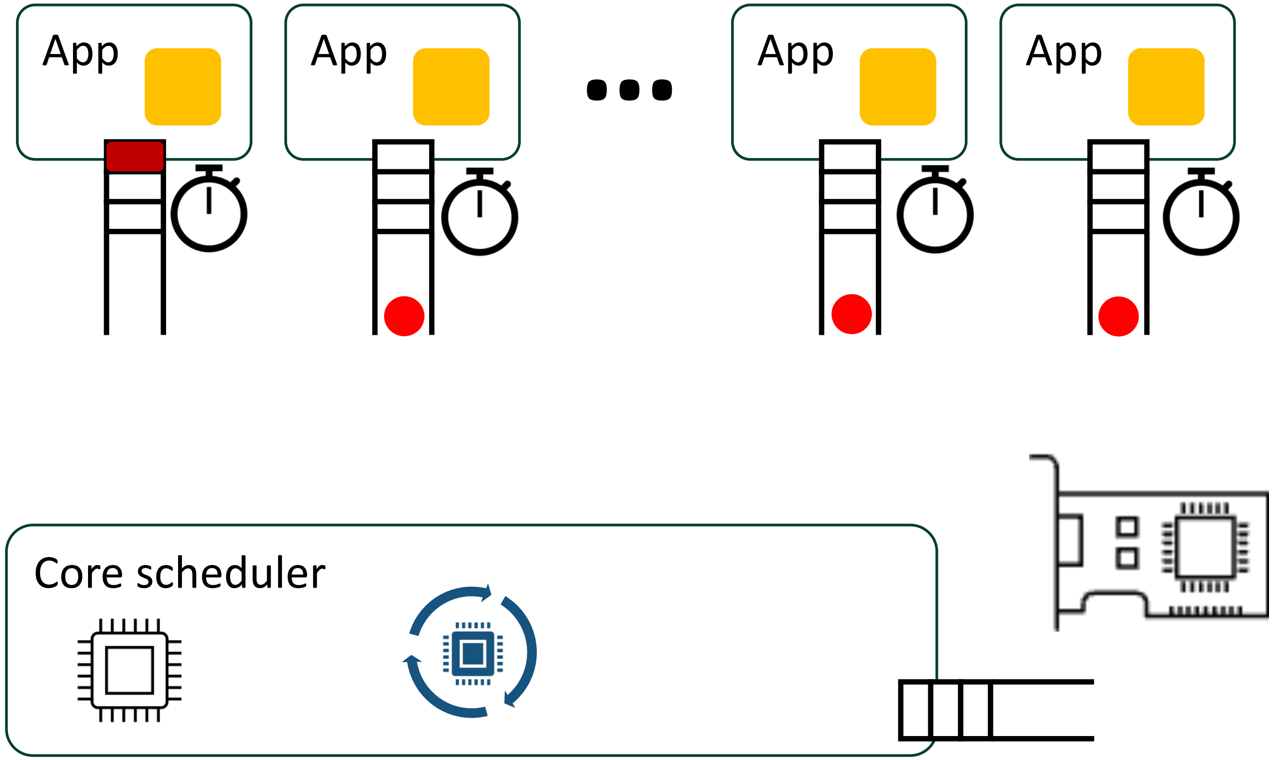

*Max 8 core instances*

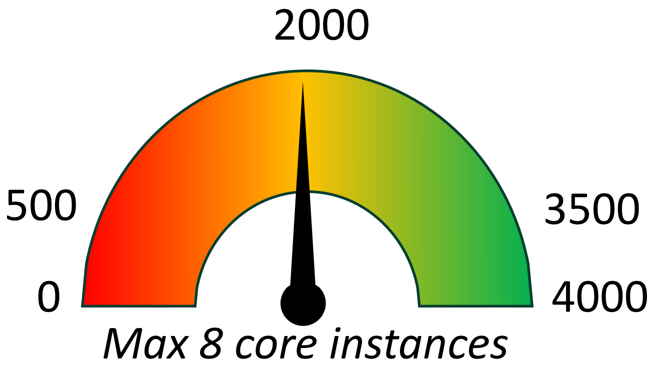


# Scaling further

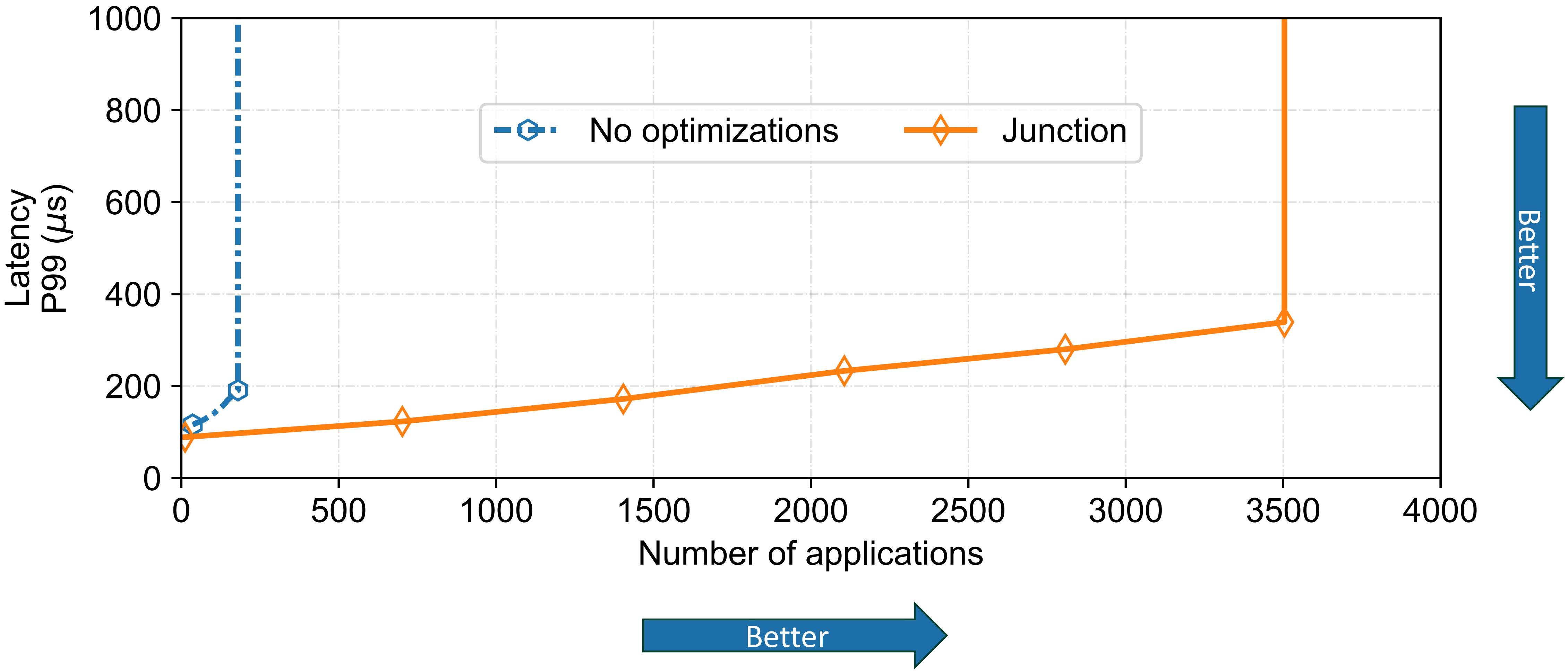
- Scheduler checks timeouts for idle applications

- Use timer wheel to track timeouts



# Evaluation



Latency
P99 ($\mu s$)
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000
3500
4000
Number of applications
No optimizations
Junction
Better
Better

# Demo

https://joshfried.io/junction_demo

# Related work

- Scheduling

  ZygOS [SOSP ‘17], Shinjuku [NSDI ‘19], Shenango [NSDI 19’], Caladan [OSDI ‘20], Persephone [SOSP ‘21]

- Hardware portability

  Demikernel [SOSP ‘21]

- Dataplane OSes

  mTCP [NSDI ‘14], IX [OSDI ‘14], Arrakis [OSDI ‘14], eRPC [NSDI ‘19],

# Conclusion

Junction aims to eliminate OS overheads by making kernel bypass ubiquitous in the datacenter.

Preserves high performance of kernel bypass but delivers density and compatibility

Available open-source: https://github.com/JunctionOS