Loom: Flexible and Efficient NIC Packet Scheduling

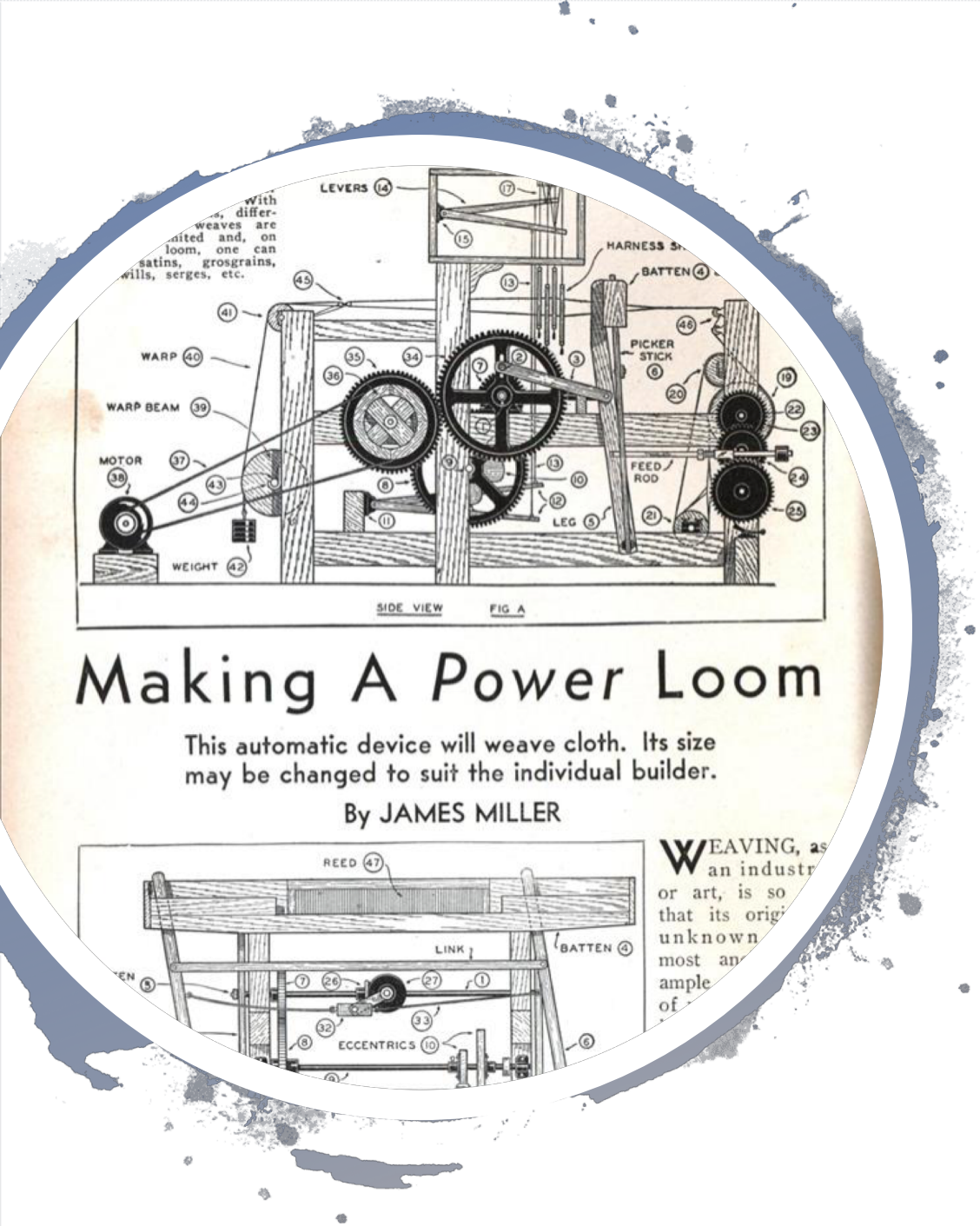
NSDI 2019

Brent Stephens

Aditya Akella, Mike Swift

UIC UNIVERSITY OF ILLINOIS AT CHICAGO

THE UNIVERSITY of WISCONSIN MADISON

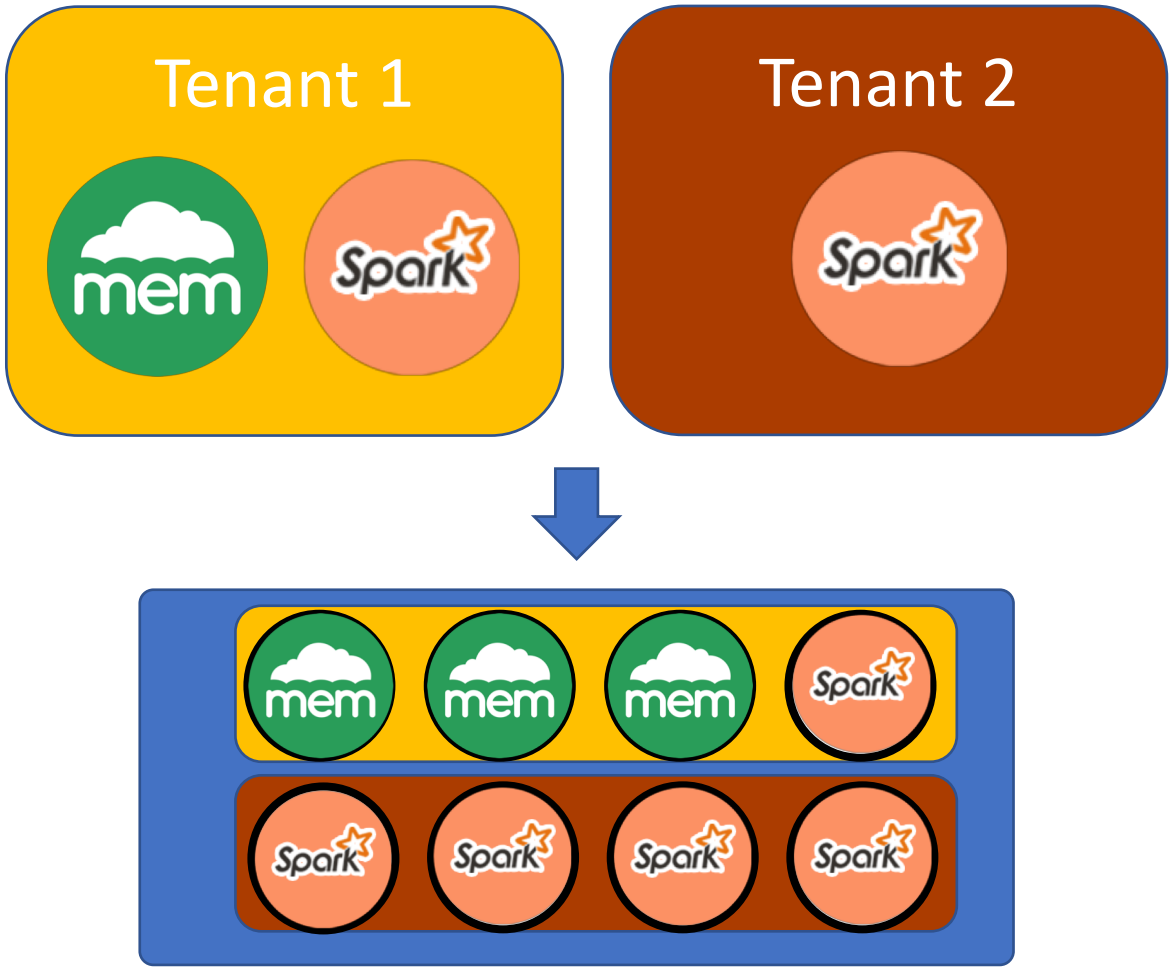


Loom is a new Network Interface Card (NIC) *design* that offloads *all* per-flow scheduling decisions out of the OS and into the NIC

- Why is packet scheduling important?
- What is wrong with current NICs?
- Why should *all* packet scheduling be offloaded to the NIC?

Why is packet scheduling important?

Collocation (Application and Tenant) is Important for Infrastructure Efficiency



CPU Isolation Policy:

```
Tenant 1:
        Memcached: 3 cores
        Spark: 1 core
Tenant 2:
        Spark: 4 cores
```

Network Performance Goals

Different applications have differing network performance goals

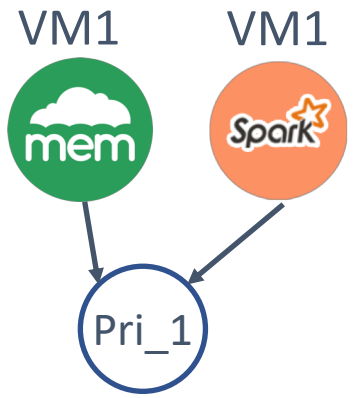
Low Latency

High Throughput

Network Policies

Pseudocode

```
Tenant_1.Memcached -> Pri_1:high
Tenant_1.Spark -> Pri_1:low
Pri_1 -> RL_WAN(Dst == WAN: 15Gbps)
Pri_1 -> RL_None(Dst != WAN: No Limit)
RL_WAN -> FIFO_1; RL_None -> FIFO_1
FIFO_1-> Fair_1:w1
Tenants_2.Spark -> Fair_1:w1
Fair_1 -> Wire
```



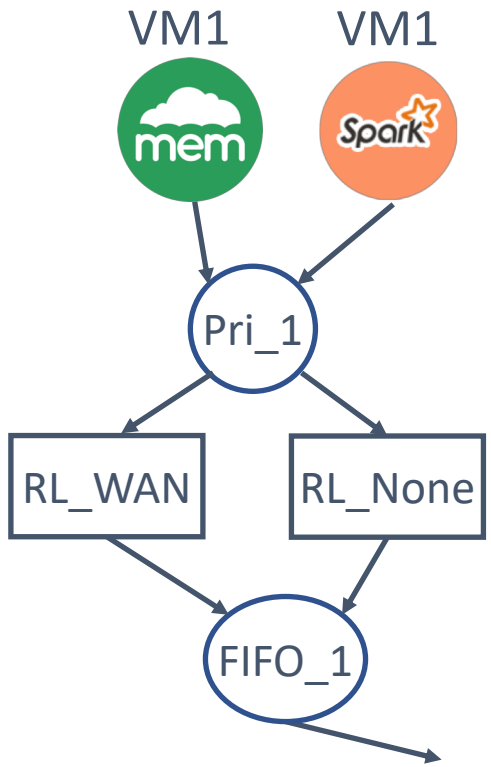
Network operators must *specify* and *enforce* a network isolation policy

- Enforcing a network isolation policy requires scheduling

Network Policies

Pseudocode

```
Tenant_1.Memcached -> Pri_1:high
Tenant_1.Spark -> Pri_1:low
Pri_1 -> RL_WAN(Dst == WAN: 15Gbps)
Pri_1 -> RL_None(Dst != WAN: No Limit)
RL_WAN -> FIFO_1; RL_None -> FIFO_1
FIFO_1-> Fair_1:w1
Tenants_2.Spark -> Fair_1:w1
Fair_1 -> Wire
```



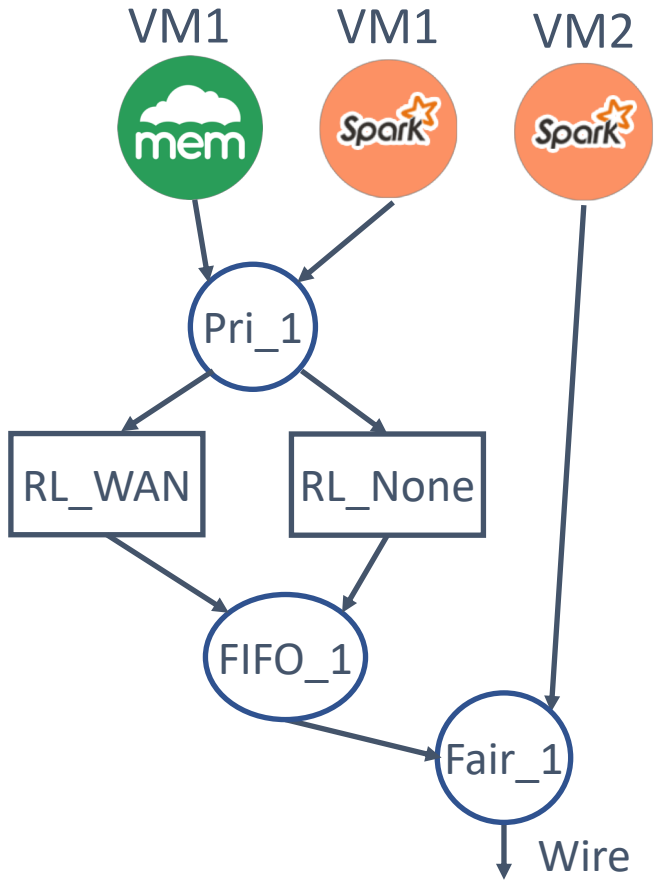
Network operators must *specify* and *enforce* a network isolation policy

- Enforcing a network isolation policy requires scheduling

Network Policies

Pseudocode

```
Tenant_1.Memcached -> Pri_1:high
Tenant_1.Spark -> Pri_1:low
Pri_1 -> RL_WAN(Dst == WAN: 15Gbps)
Pri_1 -> RL_None(Dst != WAN: No Limit)
RL_WAN -> FIFO_1; RL_None -> FIFO_1
FIFO_1-> Fair_1:w1
Tenants_2.Spark -> Fair_1:w1
Fair_1 -> Wire
```

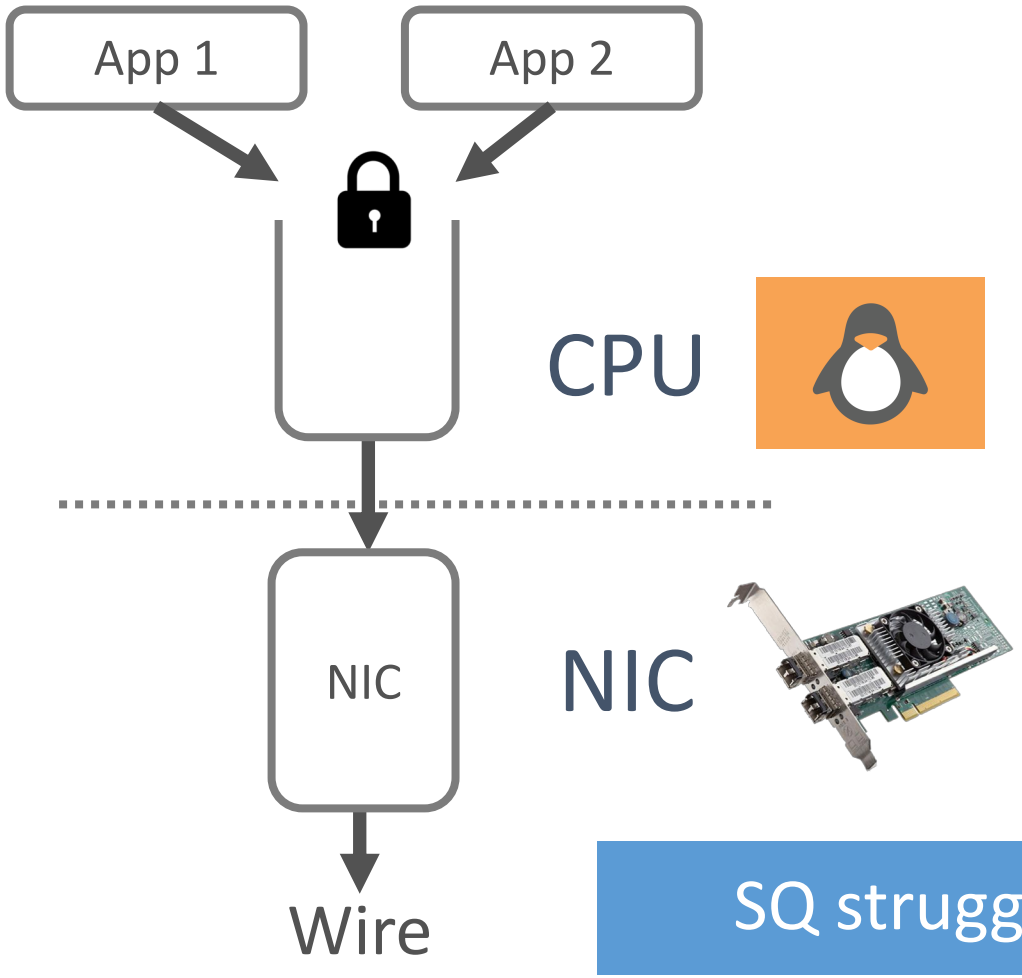


Network operators must *specify* and *enforce* a network isolation policy

- Enforcing a network isolation policy requires scheduling

What is wrong with current NICs?

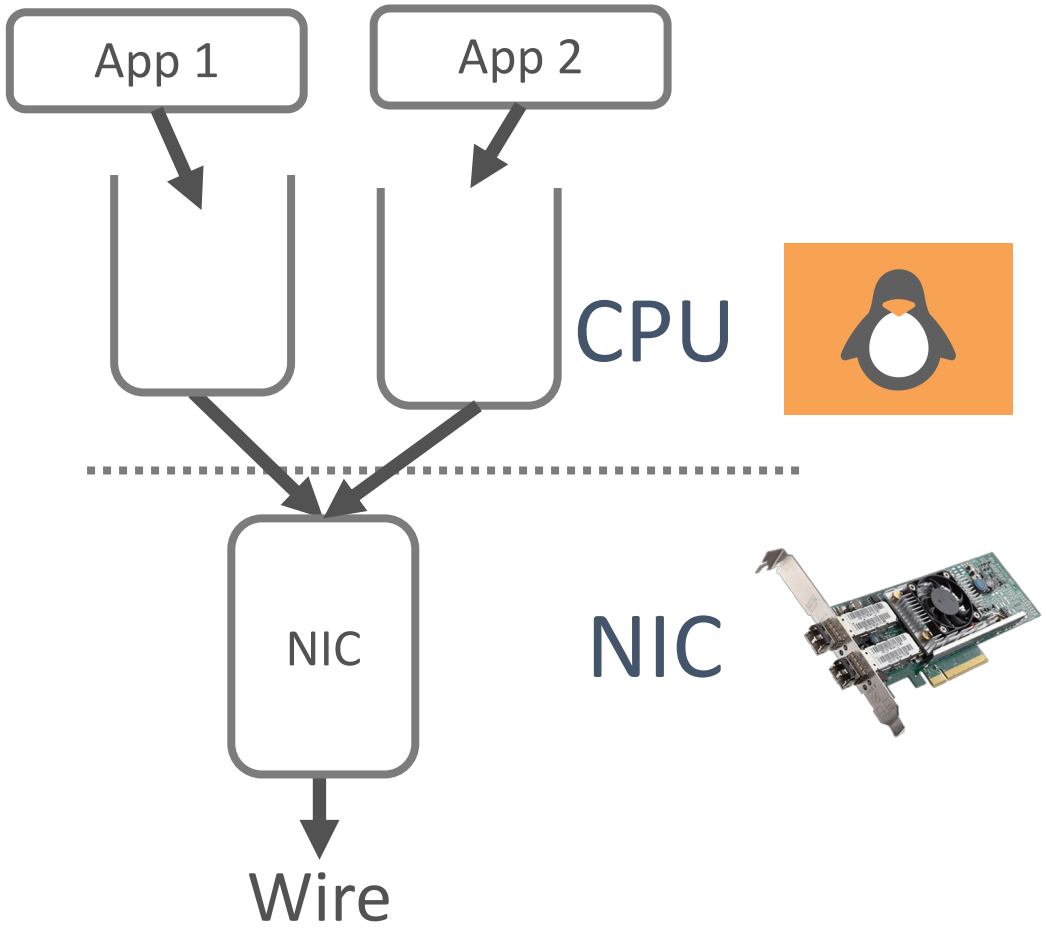
Single Queue Packet Scheduling Limitations



- Single core throughput is limited (although high with Eiffel)
  - Especially with very small packets
  - Energy-efficient architectures may prioritize scalability over single-core performance
- Software scheduling consumes CPU
- Core-to-core communication increases latency

SQ struggles to drive line-rate

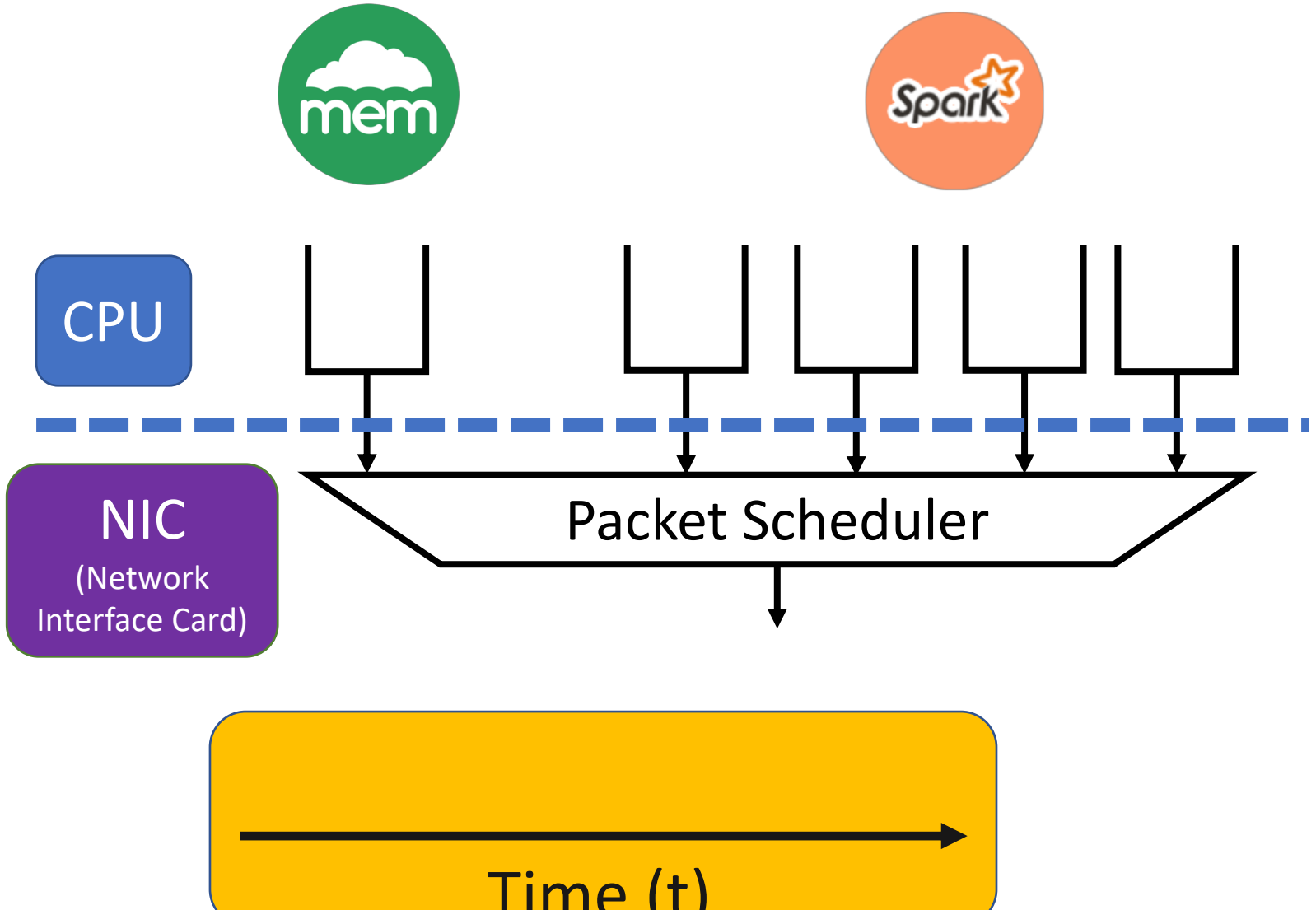
Multi Queue NIC Background and Limitations



- Multi-queue NICs enable parallelism
  - Throughput can be scaled across many tens of cores
- Multi-queue NICs have packet scheduler that chose which queue to send packets from
- The one-queue-per-core multi-queue model (MQ) attempts to enforces the policy at every core *independently*
  - This is the best possible without inter-core coordination, but it is not effective

MQ struggles to enforce policies!

MQ Scheduler Problems

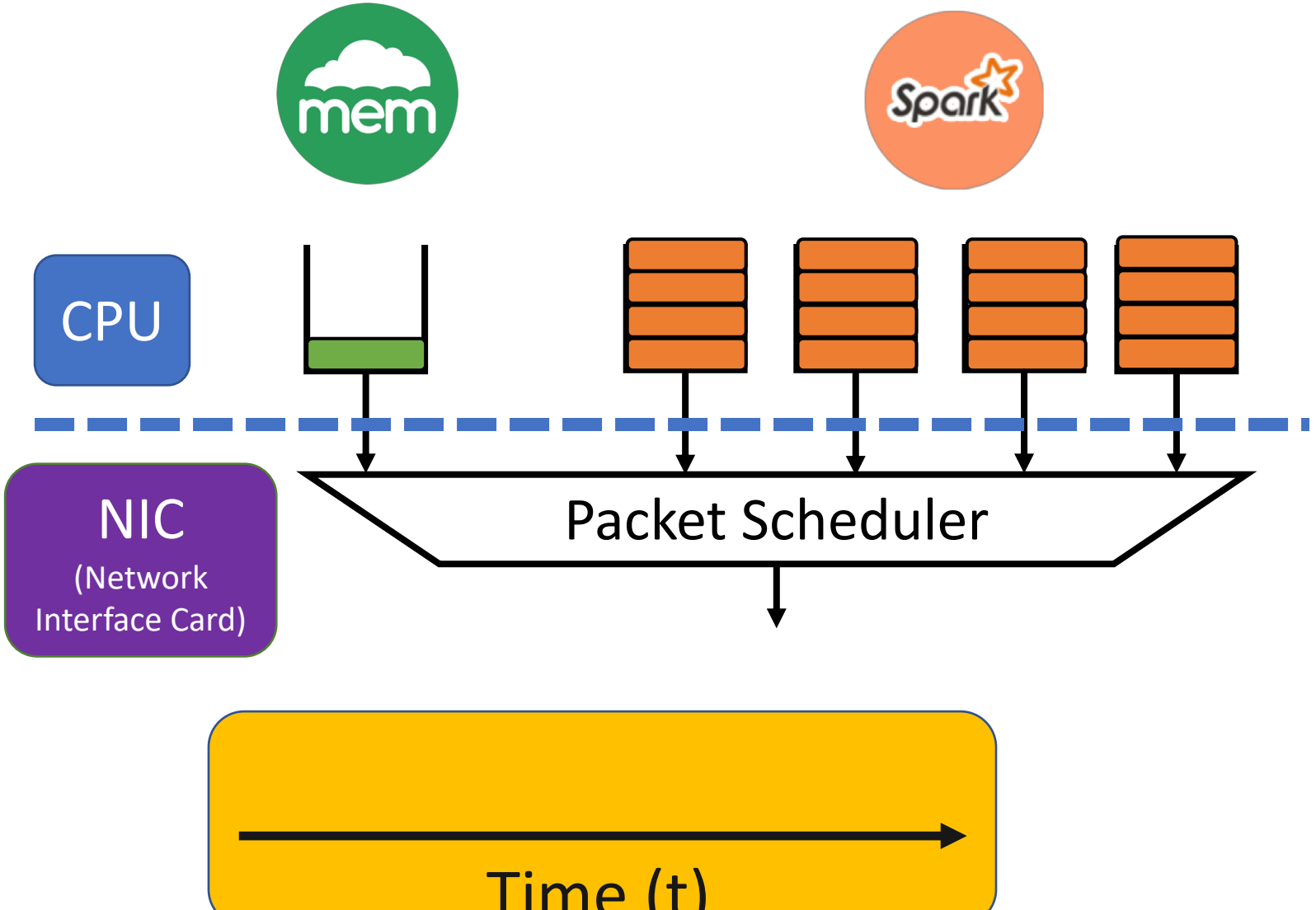


Naïve NIC packet scheduling prevents colocation!

It leads to:

- High latency
- Unfair and variable throughput

MQ Scheduler Problems



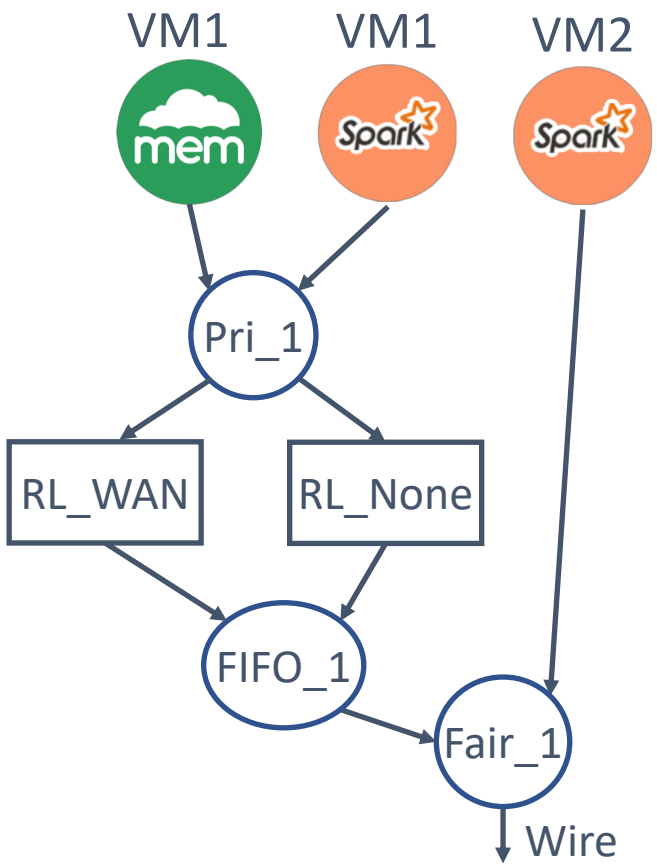
Naïve NIC packet scheduling prevents colocation!

It leads to:

- High latency
- Unfair and variable throughput

Why should *all* packet scheduling be offloaded to the NIC?

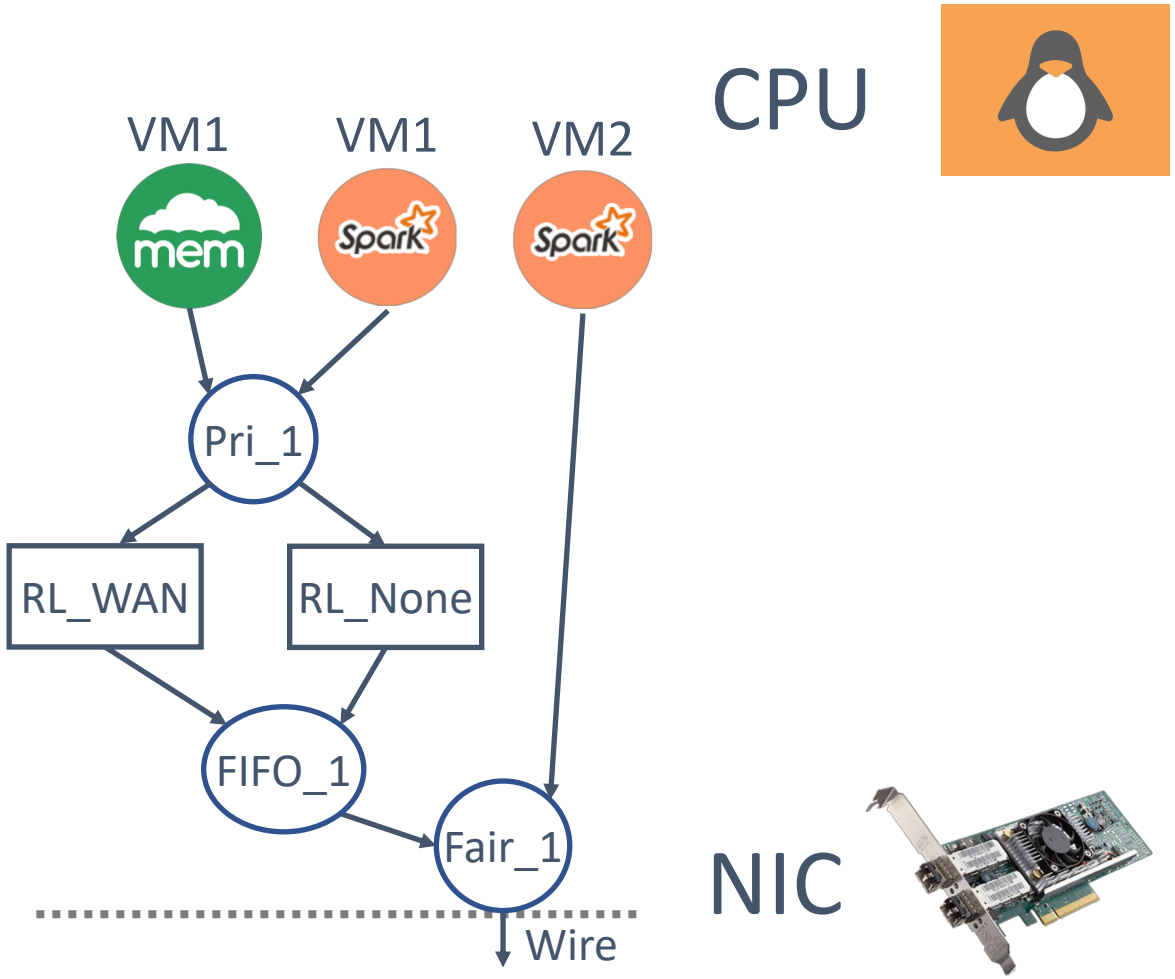
Where to divide labor between the OS and NIC?



CPU

NIC

Where to divide labor between the OS and NIC?



Option 1: Single Queue (SQ)

- Enforce entire policy in software
- **Low Tput/High CPU Utilization**

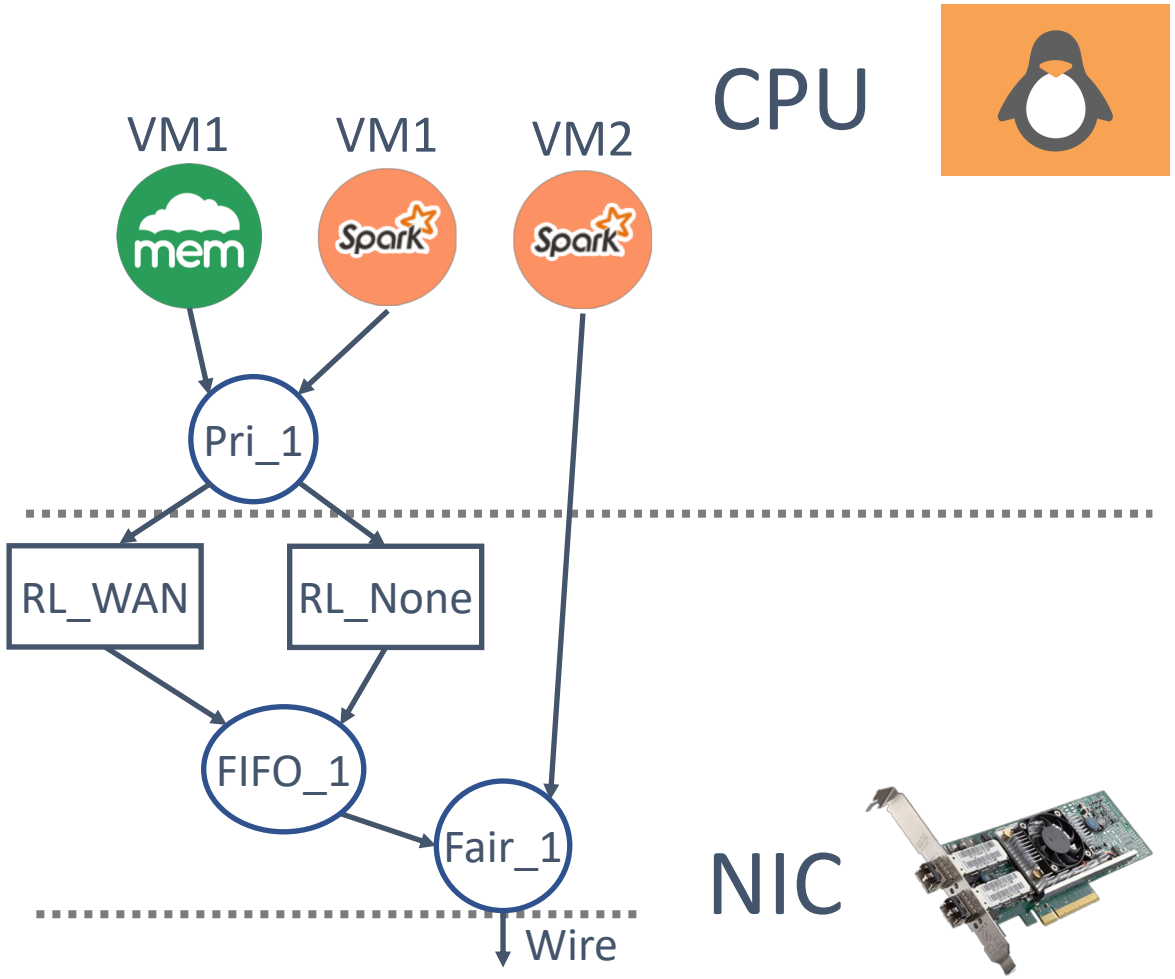
Where to divide labor between the OS and NIC?

Option 2: Multi Queue (MQ)

- Every core *independently* enforces policy on local traffic
- **Cannot ensure polices are enforced**

Option 1: Single Queue (SQ)

- Enforce entire policy in software
- **Low Tput/High CPU Utilization**

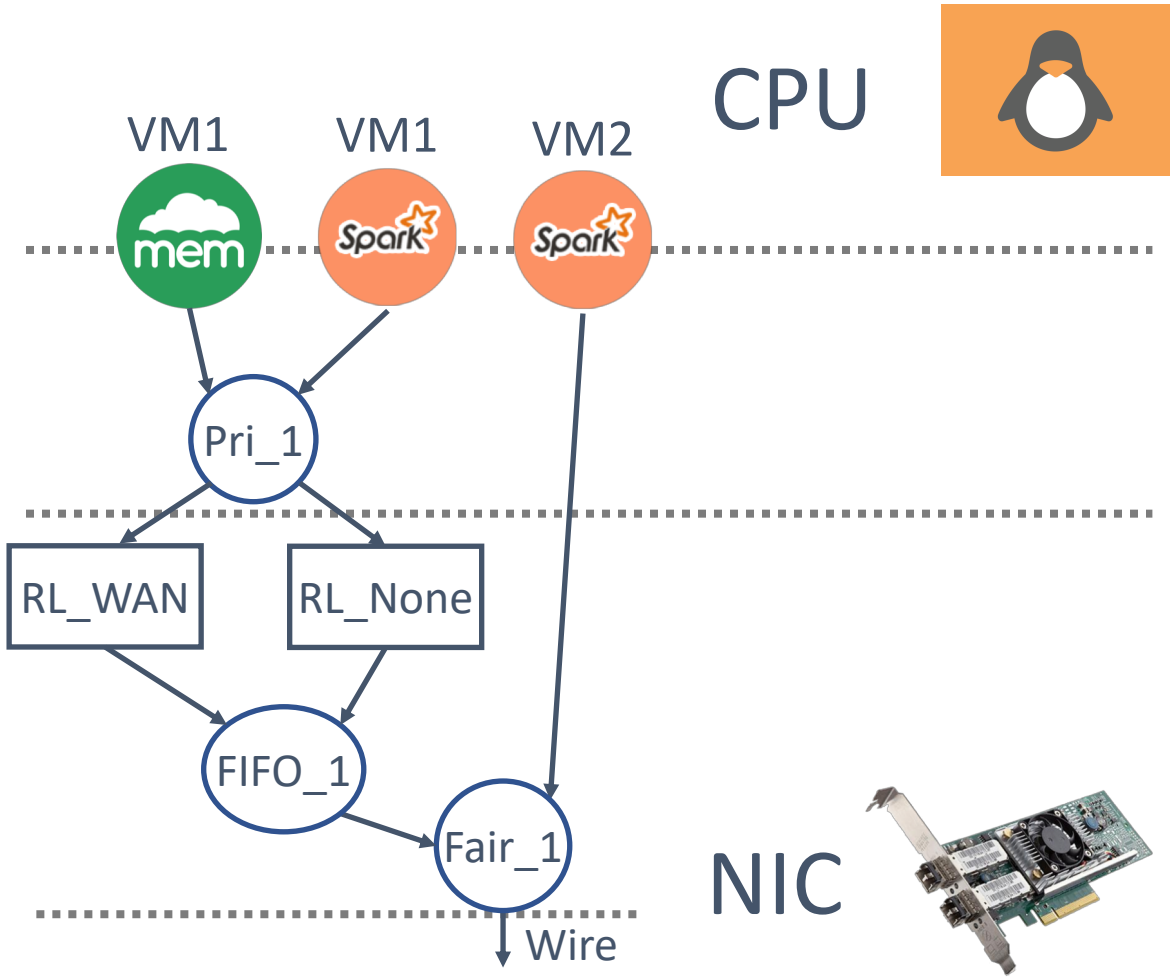


Where to divide labor between the OS and NIC?

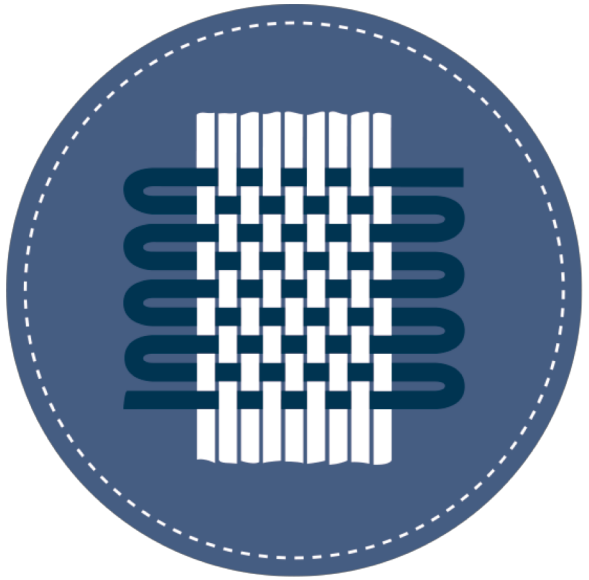
Option 3: Loom
- Every flow uses its own queue
- All policy enforcement is offloaded to the NIC
- Precise policy + low CPU

Option 2: Multi Queue (MQ)
- Every core *independently* enforces policy on local traffic
- **Cannot ensure polices are enforced**

Option 1: Single Queue (SQ)
- Enforce entire policy in software
- **Low Tput/High CPU Utilization**



Loom is a new NIC design that moves all per-flow scheduling decisions out of the OS and into the NIC



Loom uses a queue per flow and offloads all packet scheduling to the NIC

Core Problem:

It is not currently possible to offload all packet scheduling because NIC packet schedulers are **inflexible** and configuring them is **inefficient**

Core Problem:

It is not currently possible to offload all packet scheduling because NIC packet schedulers are **inflexible** and configuring them is **inefficient**

NIC packet schedulers are currently standing in the way of performance isolation!

Outline

Intro: Loom is a new NIC design that moves all per-flow scheduling decisions out of the OS and into the NIC

Contributions:

Specification: A new network policy abstraction: restricted directed acyclic graphs (DAGs)

Enforcement: A new programmable packet scheduling hierarchy designed for NICs

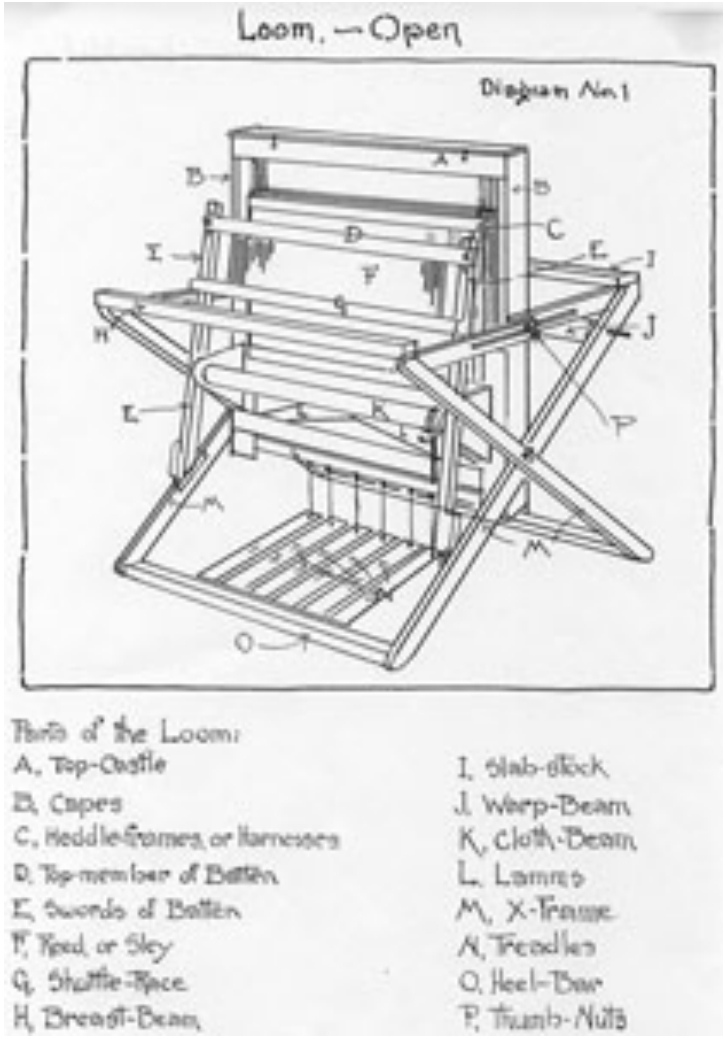
Updating: A new expressive and efficient OS/NIC interface

Implementation and Evaluation: BESS prototype and CloudLab

Outline

Contributions:

1. **Specification: A new network policy abstraction: restricted directed acyclic graphs (DAGs)**
2. Enforcement: A new programmable packet scheduling hierarchy designed for NICs
3. Updating: A new expressive and efficient OS/NIC interface



What scheduling polices are needed for performance isolation?

How should policies be specified?

Solution: Loom Policy DAG

Scheduling nodes: Work-conserving policies for sharing the local link bandwidth

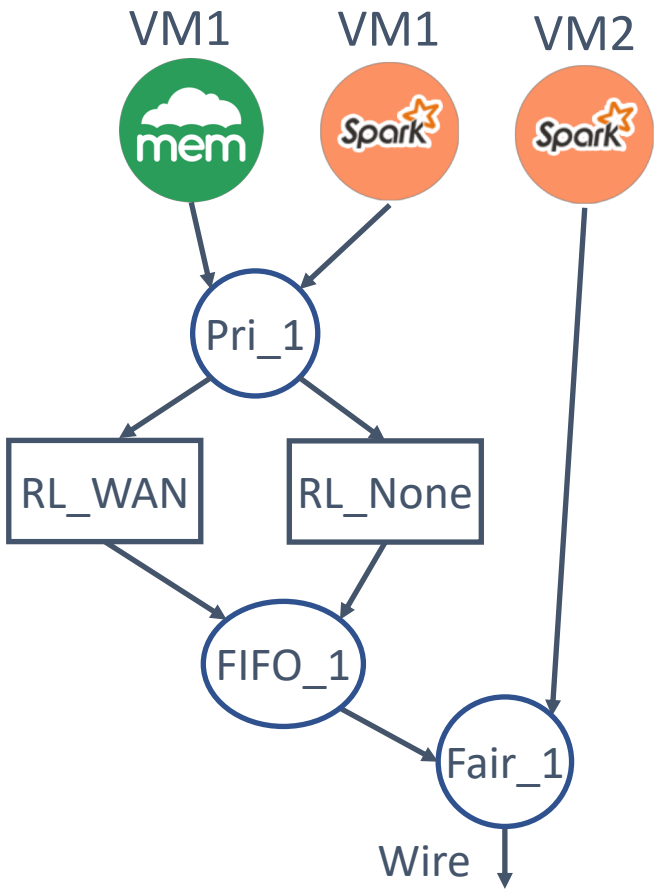
Shaping nodes: Rate-limiting policies for sharing the network core (WAN and DCN)

Programmability: Every node is programmable with a custom enqueue and dequeue function

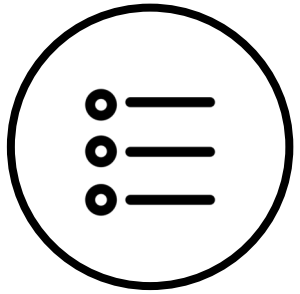
Loom can express policies that cannot be expressed with either Linux Traffic Control (Qdisc) or with Domino (PIFO)!

Important systems like BwE (sharing the WAN) and EyeQ (sharing the DCN) require Loom's policy DAG!

Two types of nodes:

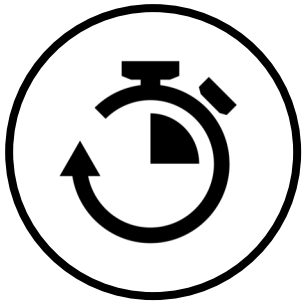


Types of Loom Scheduling Policies:



Scheduling:

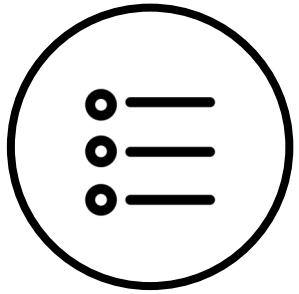
- All of the flows from competing **Spark jobs J1 and J2** in VM1 fairly share network bandwidth



Shaping:

- All of the flows from **VM1 to VM2** are rate limited to 50Gbps

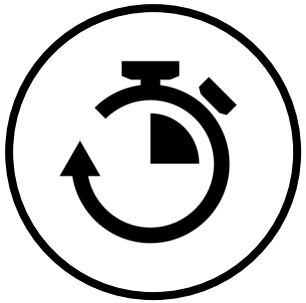
Types of Loom Scheduling Policies:



Scheduling:

Group by source

- All of the flows from competing **Spark jobs J1 and J2** in VM1 fairly share network bandwidth

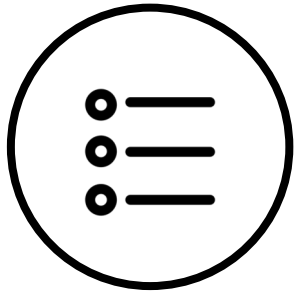


Shaping:

Group by destination

- All of the flows from **VM1 to VM2** are rate limited to 50Gbps

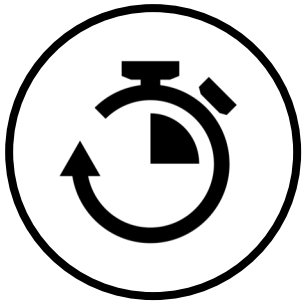
Types of Loom Scheduling Policies:



Scheduling:

Group by source

- All of the flows from competing **Spark jobs J1 and J2** in VM1 fairly share network bandwidth



Shaping:

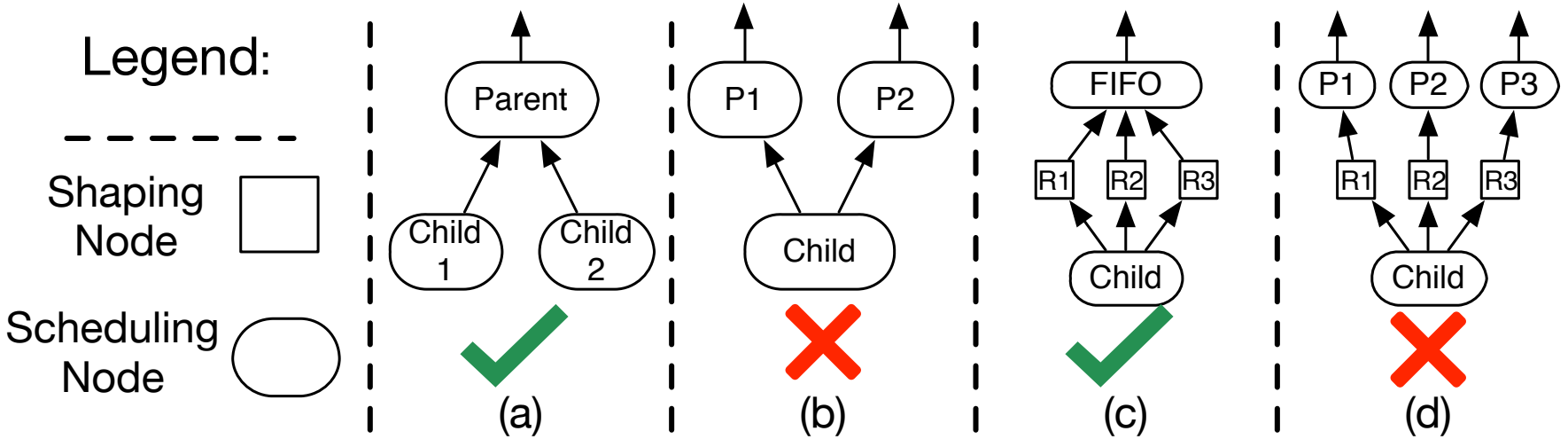
Group by destination

- All of the flows from **VM1 to VM2** are rate limited to 50Gbps

Because Scheduling and Shaping polices may aggregate flows differently, they cannot be expressed as a tree!

Loom: Policy Abstraction

Policies are expressed as *restricted* acyclic graphs (DAGs)

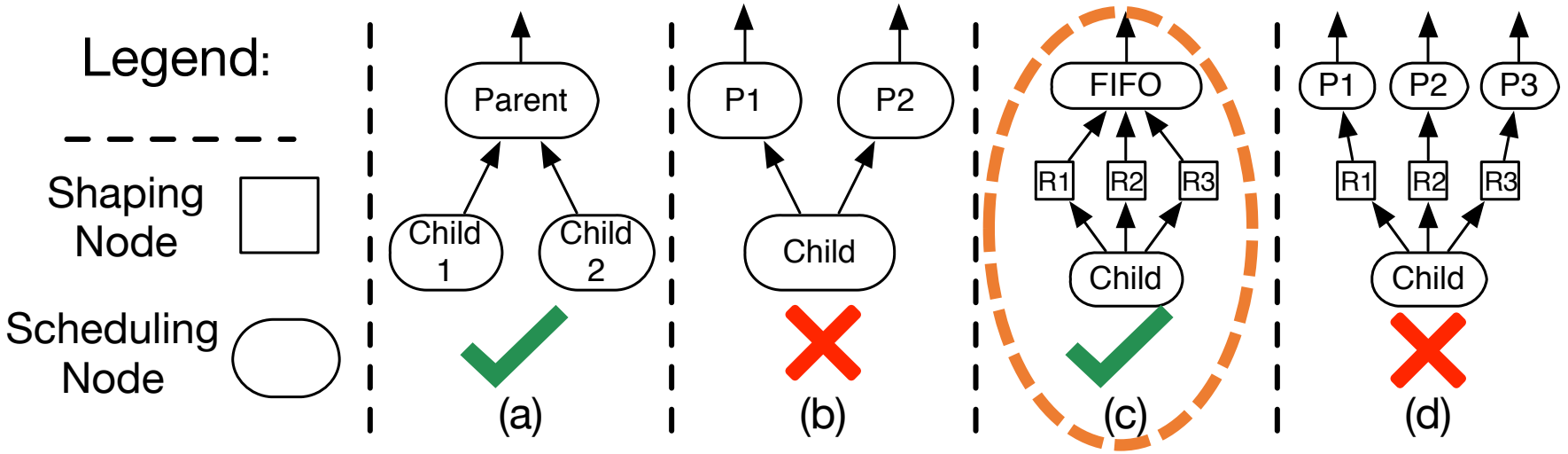


DAG restriction: Scheduling nodes form a tree when the shaping nodes are removed

(b) And (d) are prevented because they allow parents to reorder packets that were already ordered by a child node.

Loom: Policy Abstraction

Policies are expressed as *restricted* acyclic graphs (DAGs)



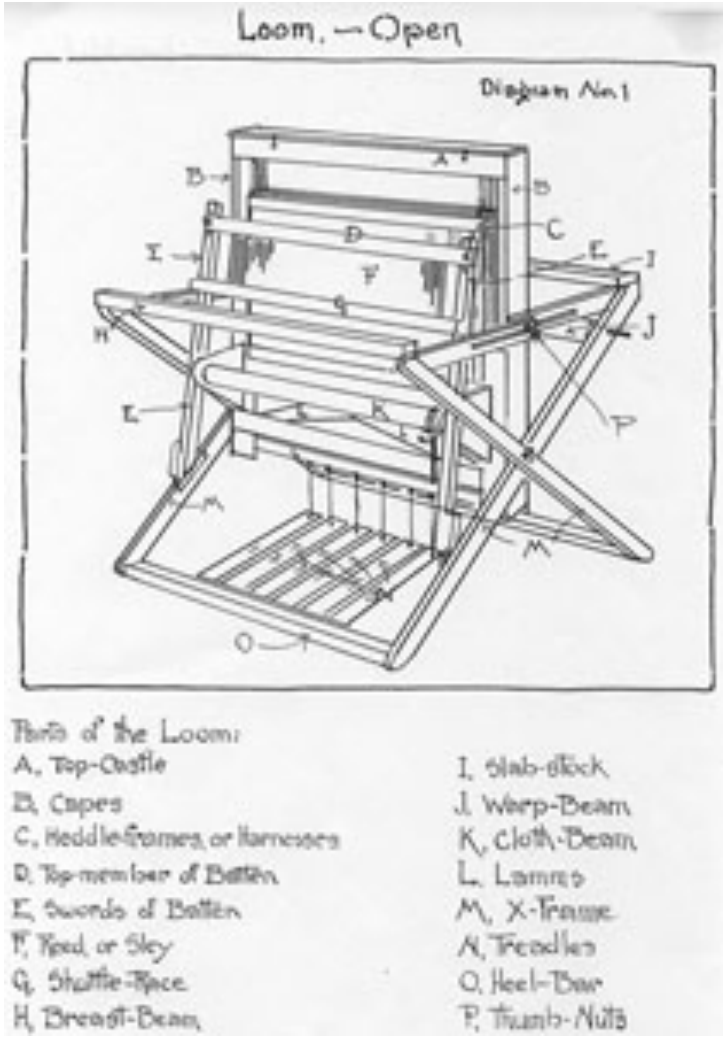
DAG restriction: Scheduling nodes form a tree when the shaping nodes are removed

(b) And (d) are prevented because they allow parents to reorder packets that were already ordered by a child node.

Outline

Contributions:

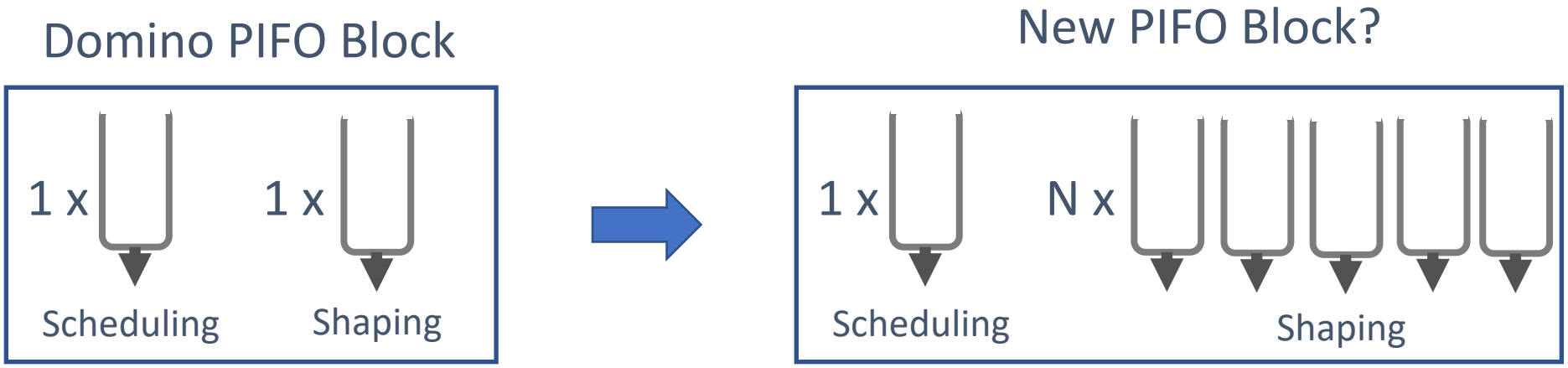
1. Specification: A new network policy abstraction: restricted directed acyclic graphs (DAGs)
2. **Enforcement: A new programmable packet scheduling hierarchy designed for NICs**
3. Updating: A new expressive and efficient OS/NIC interface



How do we build a NIC that can enforce Loom's new DAG abstraction?

Loom Enforcement Challenge

No existing hardware scheduler can efficiently enforce Loom Policy DAGs



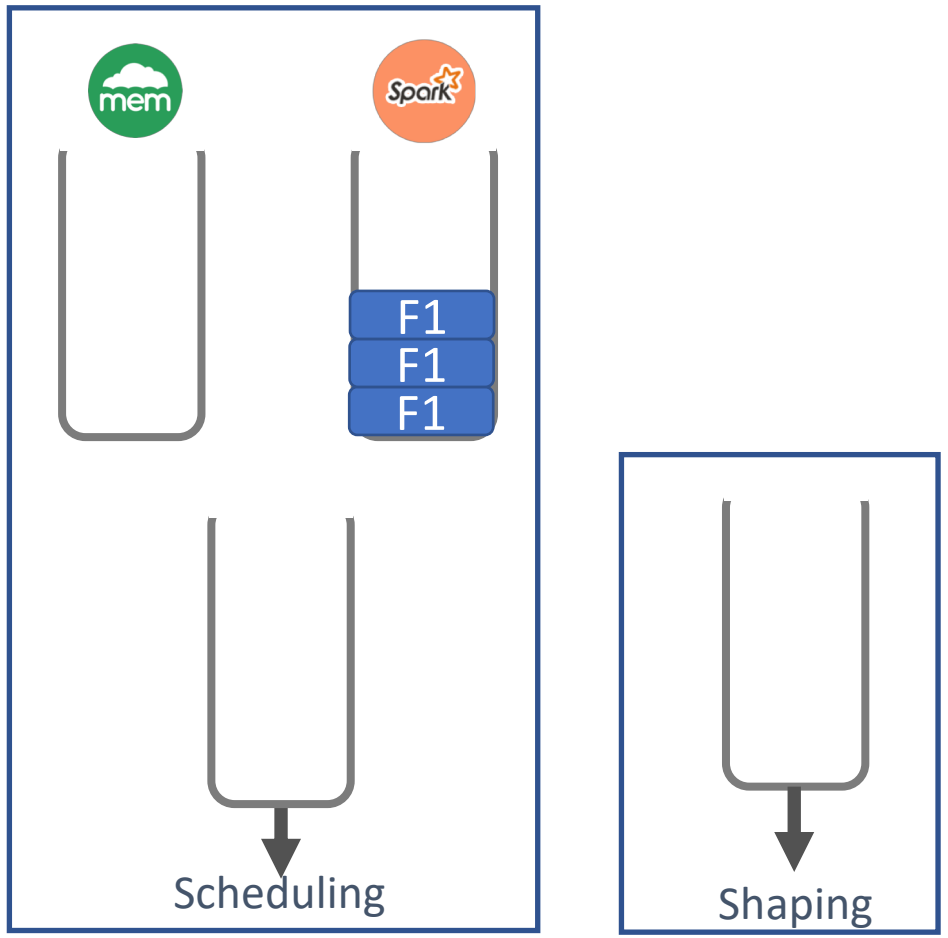
Requiring separate shaping queues for every shaping traffic class would be prohibitive!

Insight: All shaping can be done with a single queue because all shaping can use wall clock time as a rank

Loom Enforcement

In Loom, scheduling and shaping queues are separate

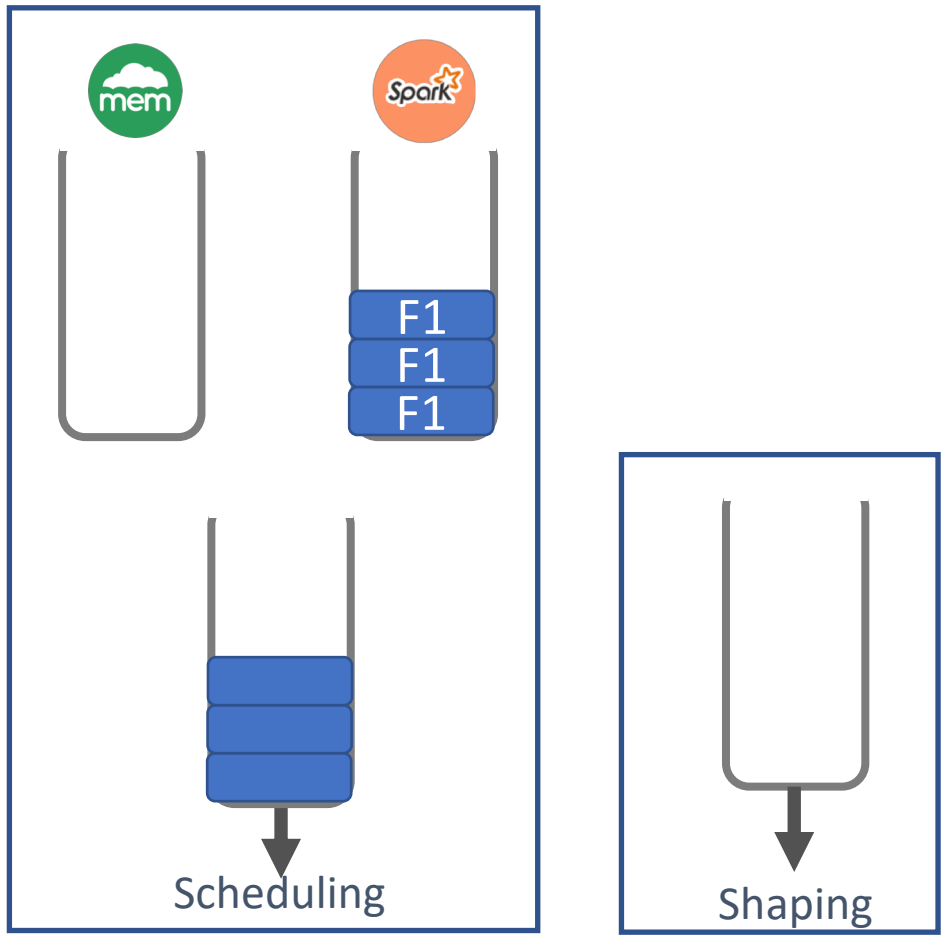
1. All traffic is first only placed in scheduling queues
2. If a packet is dequeued before its shaping time, it is placed in a global shaping queue
3. After shaping, the packet is placed back in scheduling queues



Loom Enforcement

In Loom, scheduling and shaping queues are separate

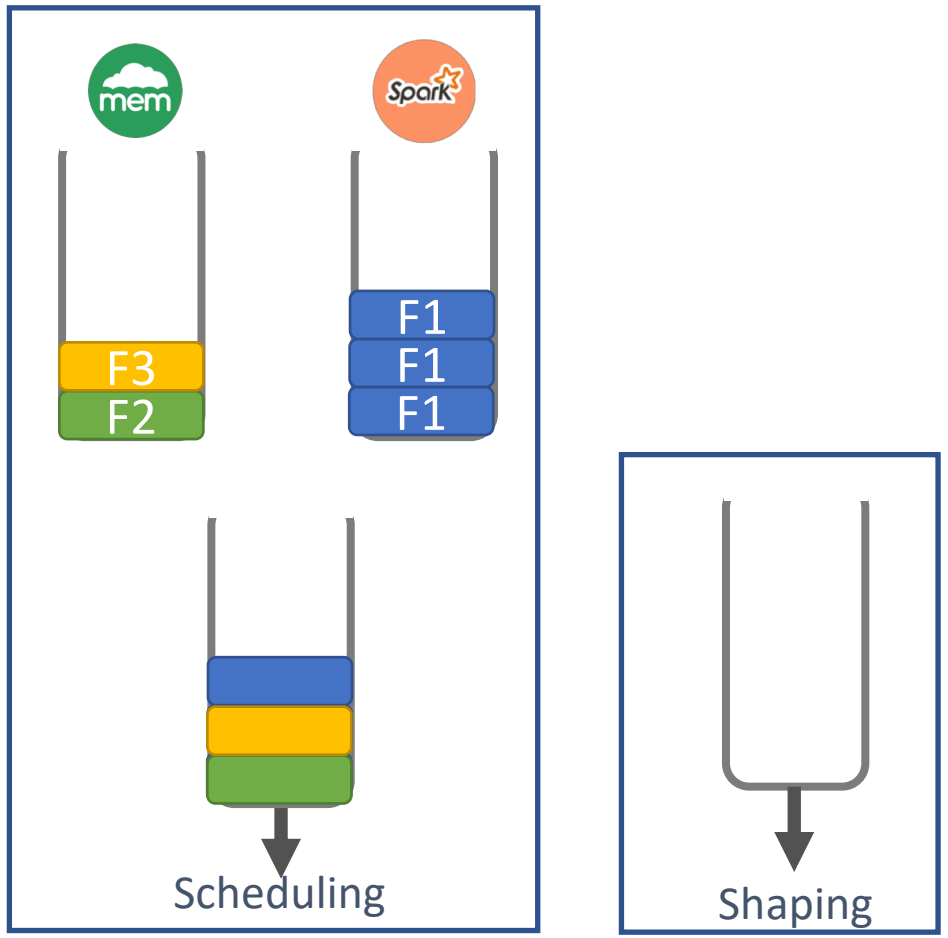
1. All traffic is first only placed in scheduling queues
2. If a packet is dequeued before its shaping time, it is placed in a global shaping queue
3. After shaping, the packet is placed back in scheduling queues



Loom Enforcement

In Loom, scheduling and shaping queues are separate

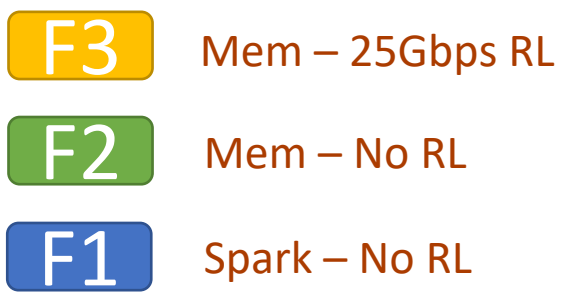
1. All traffic is first only placed in scheduling queues
2. If a packet is dequeued before its shaping time, it is placed in a global shaping queue
3. After shaping, the packet is placed back in scheduling queues

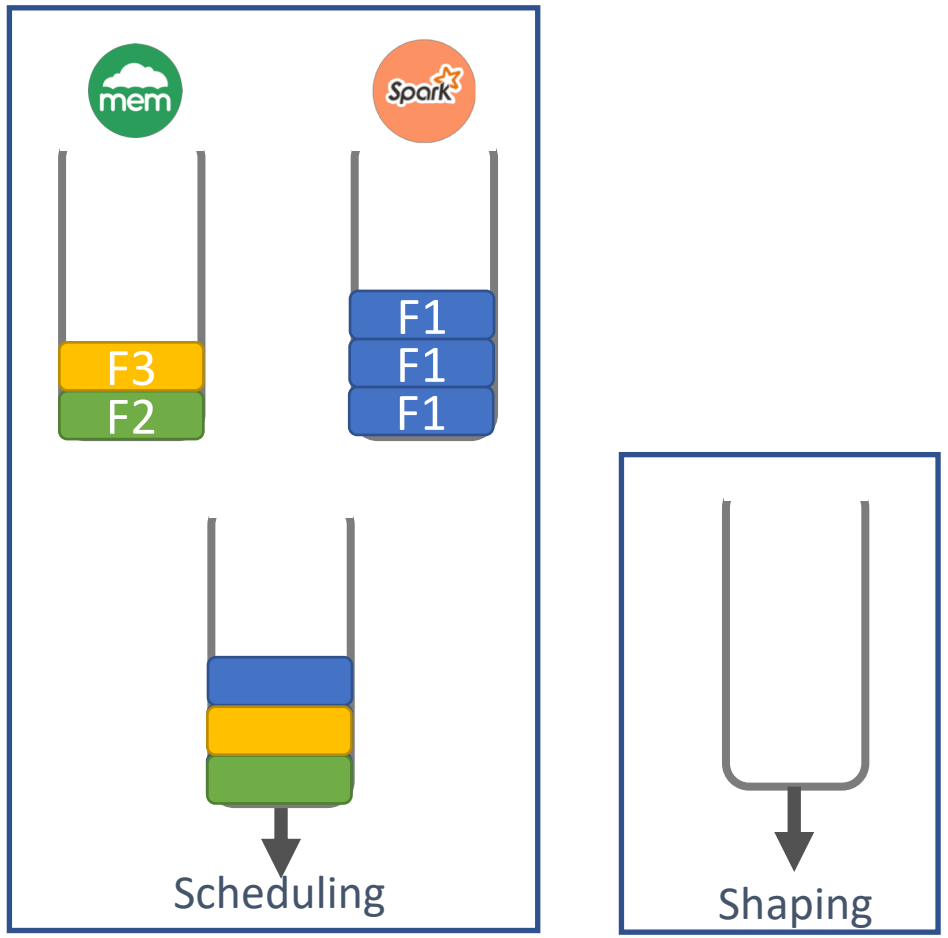


Loom Enforcement

In Loom, scheduling and shaping queues are separate

1. All traffic is first only placed in scheduling queues
2. If a packet is dequeued before its shaping time, it is placed in a global shaping queue
3. After shaping, the packet is placed back in scheduling queues

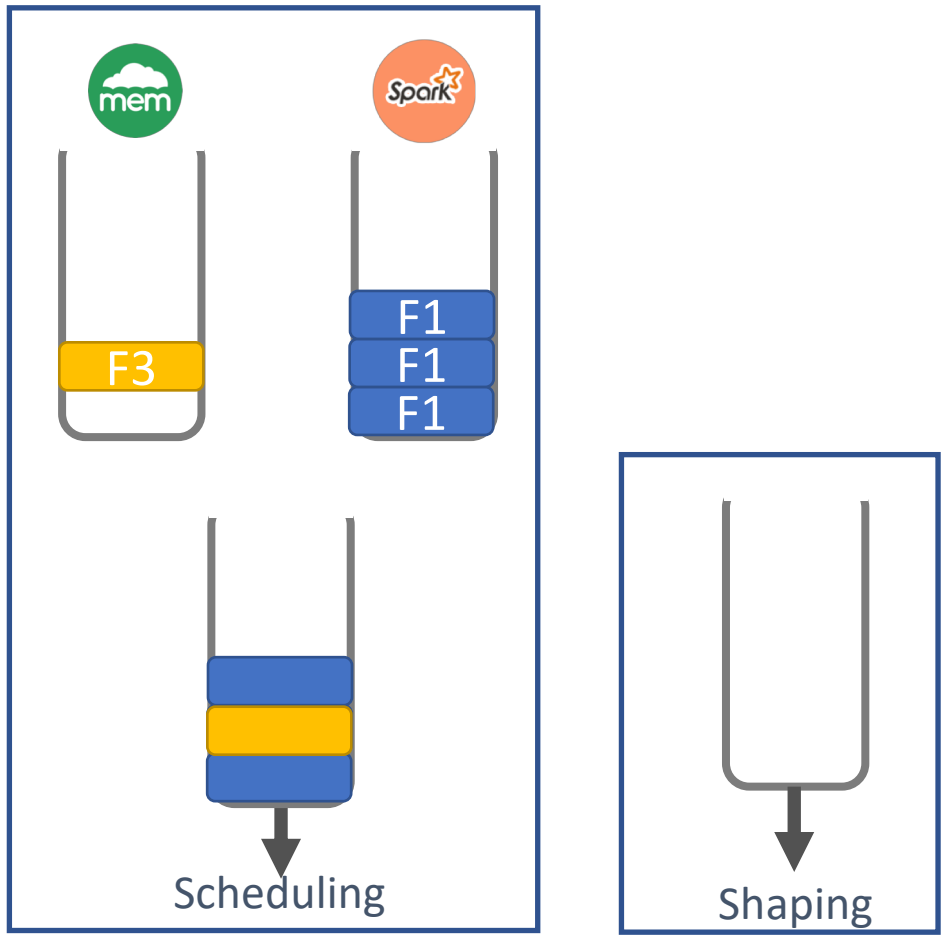




Loom Enforcement

In Loom, scheduling and shaping queues are separate

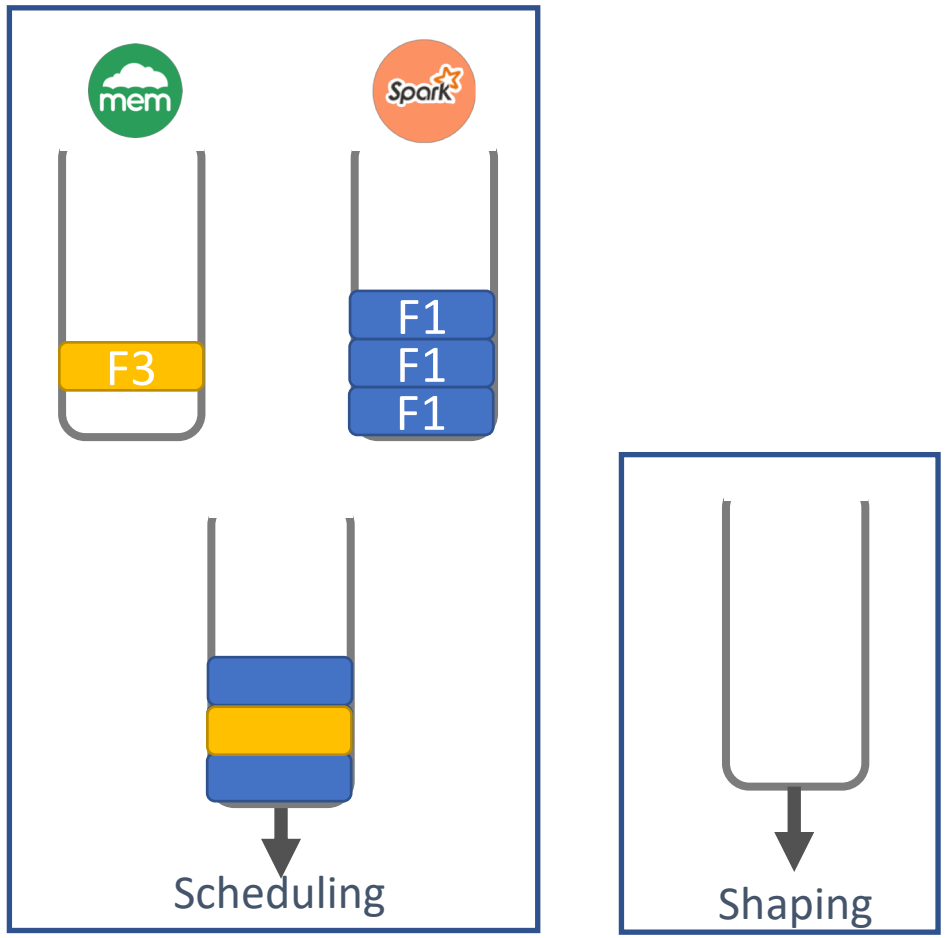
1. All traffic is first only placed in scheduling queues
2. If a packet is dequeued before its shaping time, it is placed in a global shaping queue
3. After shaping, the packet is placed back in scheduling queues



Loom Enforcement

In Loom, scheduling and shaping queues are separate

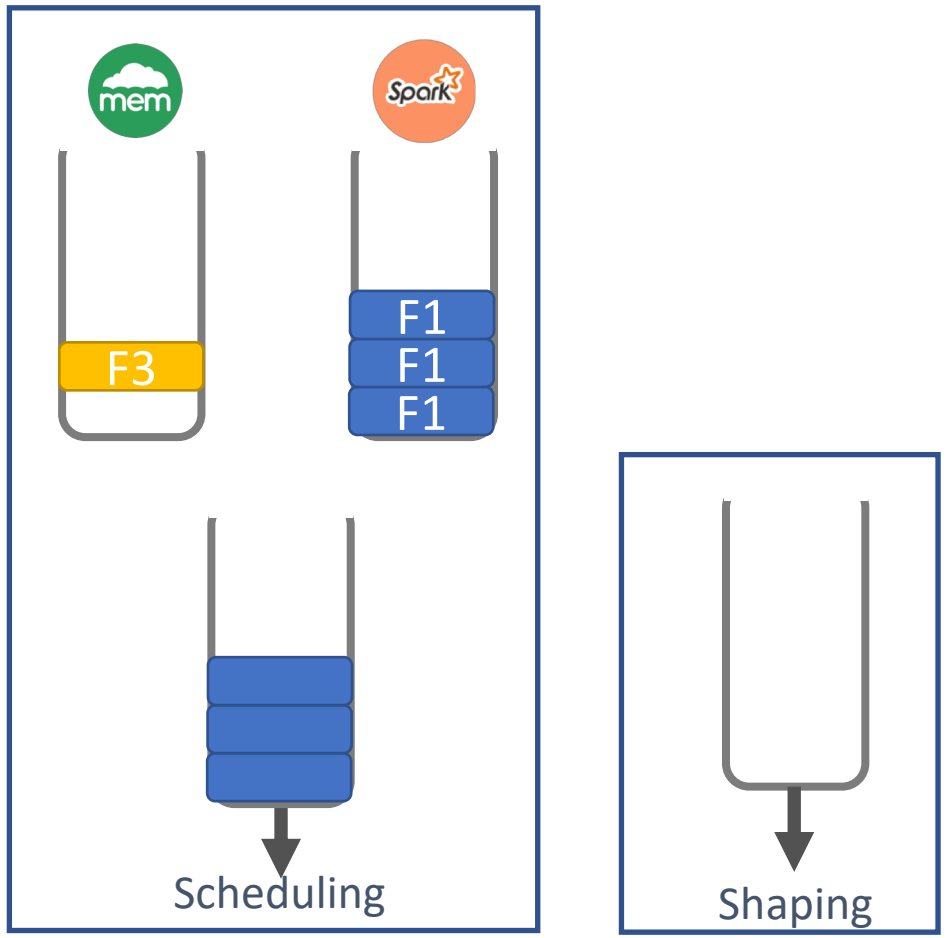
1. All traffic is first only placed in scheduling queues
2. If a packet is dequeued before its shaping time, it is placed in a global shaping queue
3. After shaping, the packet is placed back in scheduling queues



Loom Enforcement

In Loom, scheduling and shaping queues are separate

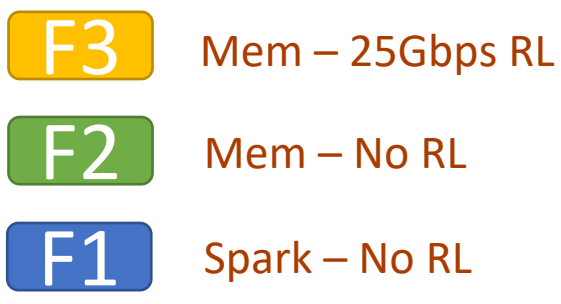
1. All traffic is first only placed in scheduling queues
2. If a packet is dequeued before its shaping time, it is placed in a global shaping queue
3. After shaping, the packet is placed back in scheduling queues

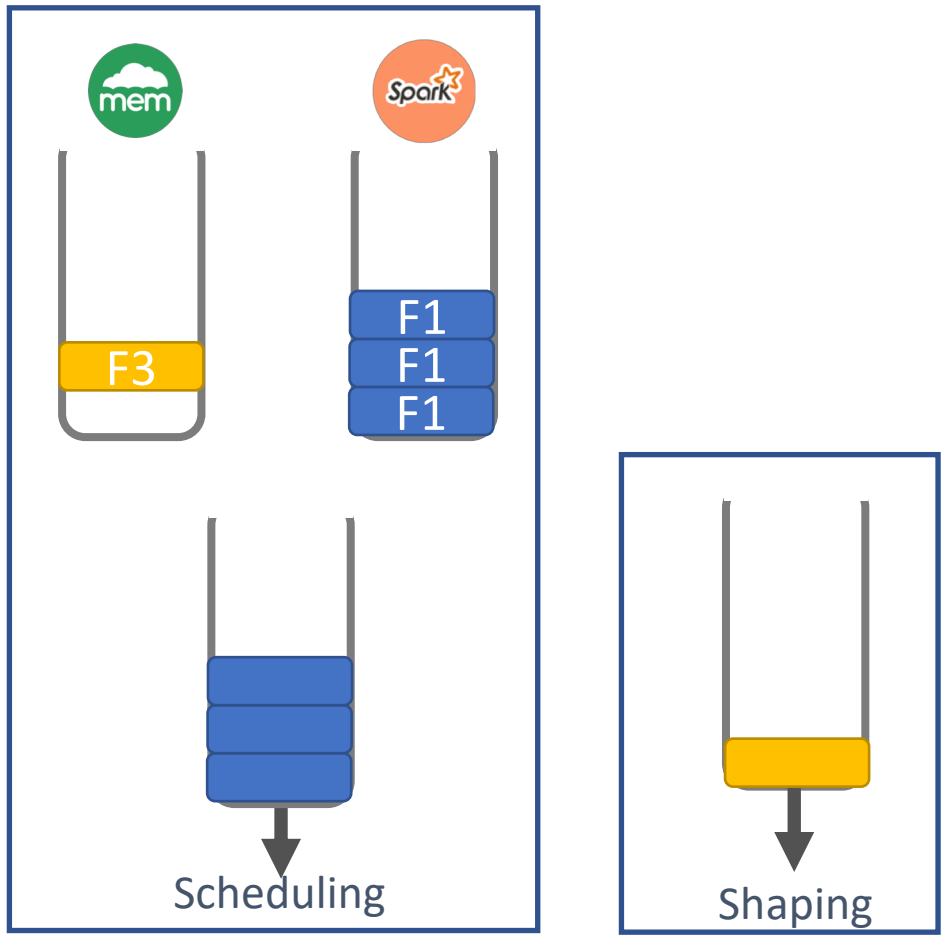


Loom Enforcement

In Loom, scheduling and shaping queues are separate

1. All traffic is first only placed in scheduling queues
2. If a packet is dequeued before its shaping time, it is placed in a global shaping queue
3. After shaping, the packet is placed back in scheduling queues

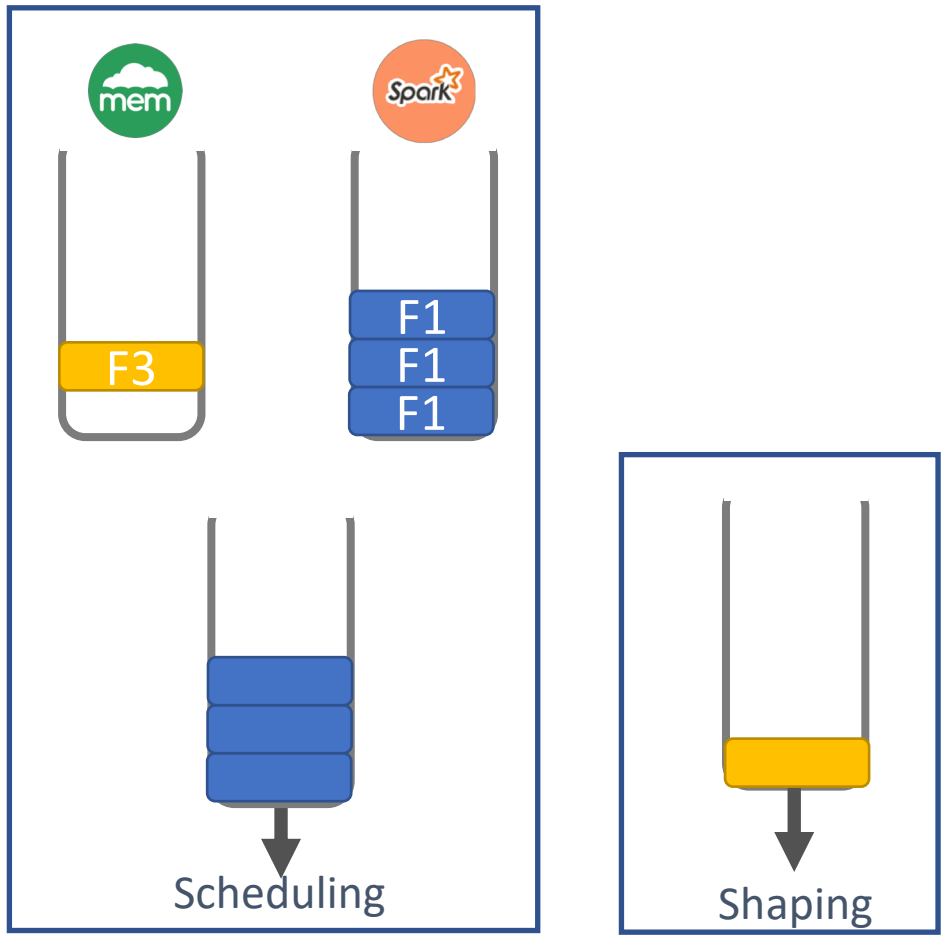




Loom Enforcement

In Loom, scheduling and shaping queues are separate

1. All traffic is first only placed in scheduling queues
2. If a packet is dequeued before its shaping time, it is placed in a global shaping queue
3. After shaping, the packet is placed back in scheduling queues



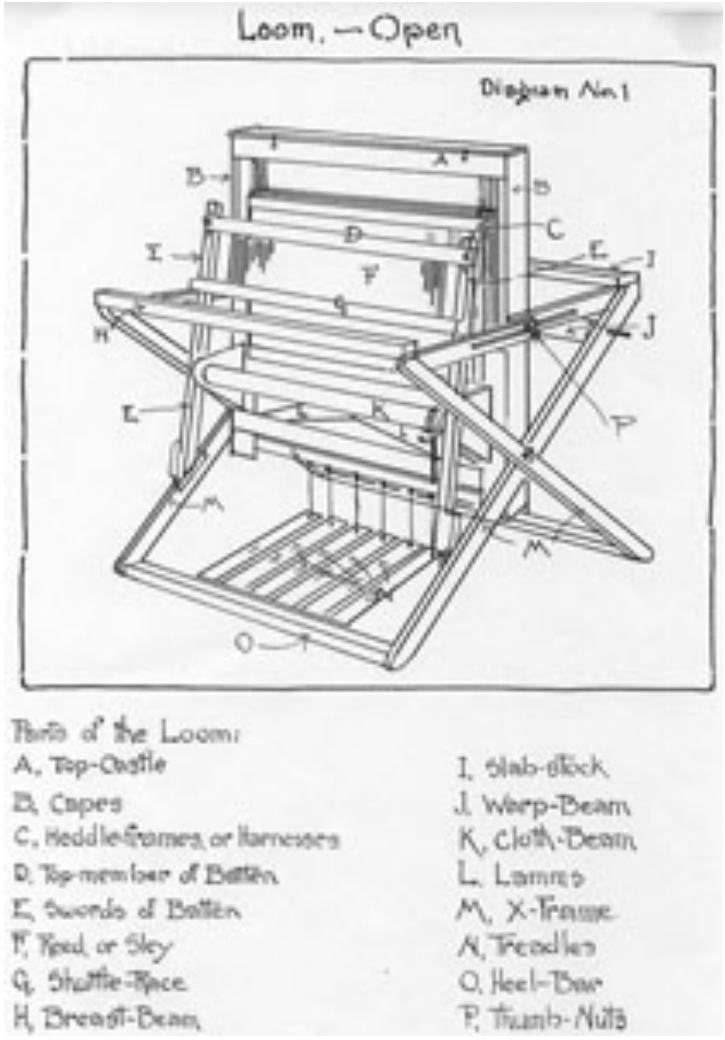
Outline

Contributions:

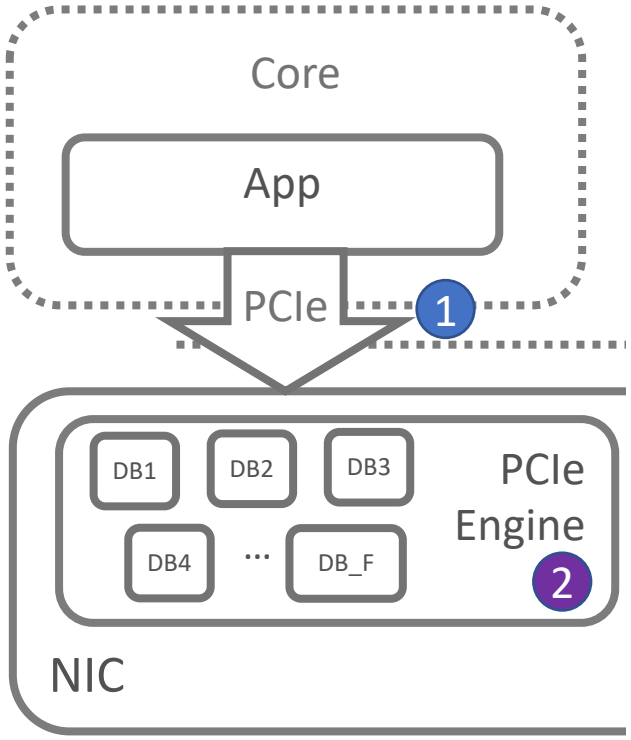
1. Specification: A new network policy abstraction: restricted directed acyclic graphs (DAGs)

2. Enforcement: A new programmable packet scheduling hierarchy designed for NICs

3. **Updating: A new expressive and efficient OS/NIC interface**



PCIe Limitations:



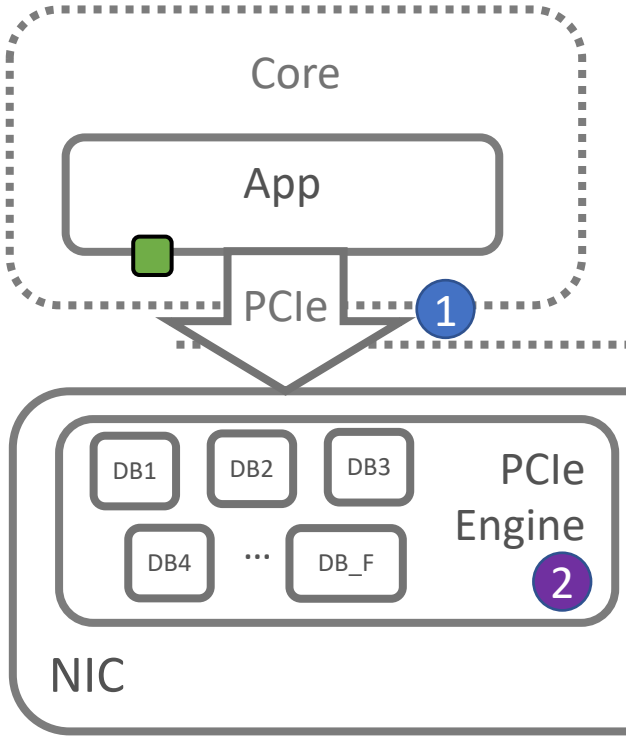
NIC doorbell and update limitations:[1]

1. Latency Limitations:
   - 120-900ns
2. Throughput Limitations:
   - ~3Mops (Intel XL710 40Gbps)

1 PSPAT: software packet scheduling at hardware speed

Luigi Rizzo[1], Paolo Valente[2], Giuseppe Lettieri[1], Vincenzo Maffione[2]
[1]Univ. di Pisa, [2]Univ.di Modena e Reggio Emilia
rizzo.unipi@gmail.com. Work supported by H2020 project SSICLOPS.

PCIe Limitations:



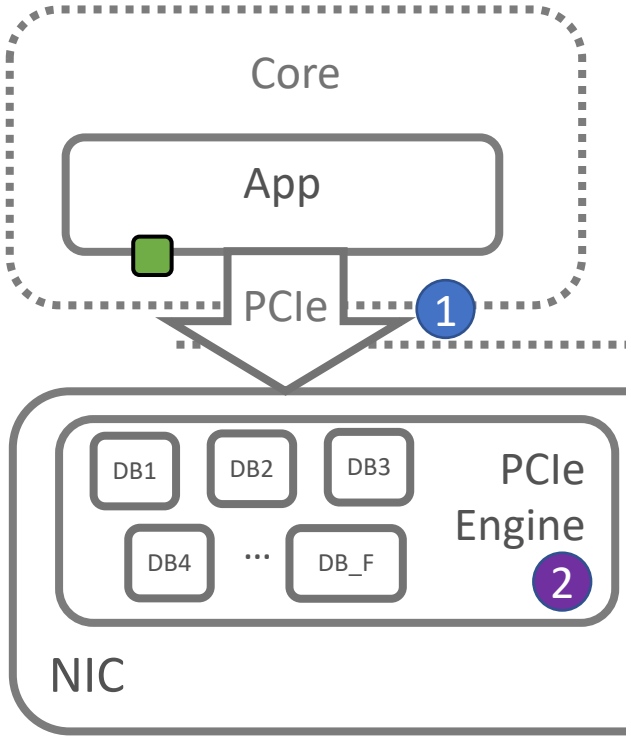
NIC doorbell and update limitations:[1]

1. Latency Limitations:
   - 120-900ns
2. Throughput Limitations:
   - ~3Mops (Intel XL710 40Gbps)

1 PSPAT: software packet scheduling at hardware speed

Luigi Rizzo[1], Paolo Valente[2], Giuseppe Lettieri[1], Vincenzo Maffione[2]
[1]Univ. di Pisa, [2]Univ.di Modena e Reggio Emilia
rizzo.unipi@gmail.com. Work supported by H2020 project SSICLOPS.

PCIe Limitations:



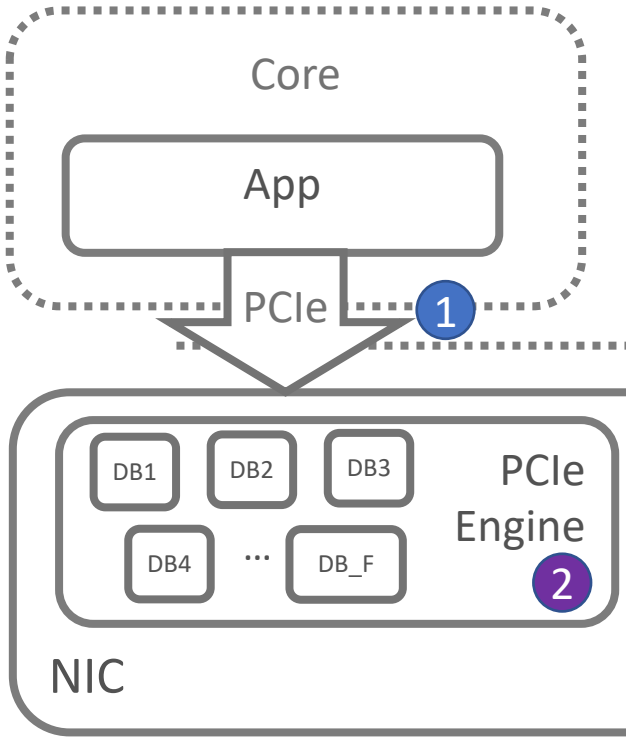
NIC doorbell and update limitations:[1]

1. Latency Limitations:
   - 120-900ns
2. Throughput Limitations:
   - ~3Mops (Intel XL710 40Gbps)

1 PSPAT: software packet scheduling at hardware speed

Luigi Rizzo[1], Paolo Valente[2], Giuseppe Lettieri[1], Vincenzo Maffione[2]
[1]Univ. di Pisa, [2]Univ.di Modena e Reggio Emilia
rizzo.unipi@gmail.com. Work supported by H2020 project SSICLOPS.

PCIe Limitations:



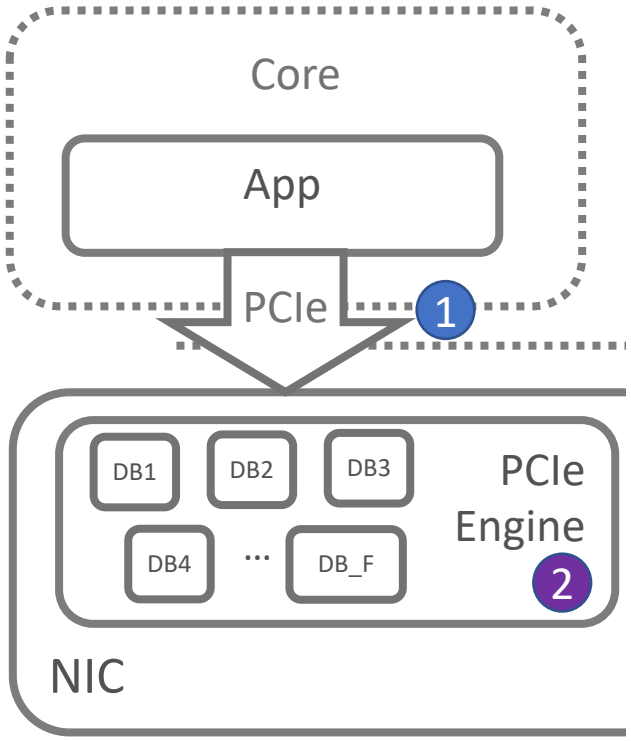
NIC doorbell and update limitations:[1]

1. Latency Limitations:
   - 120-900ns
2. Throughput Limitations:
   - ~3Mops (Intel XL710 40Gbps)

1 PSPAT: software packet scheduling at hardware speed

Luigi Rizzo[1], Paolo Valente[2], Giuseppe Lettieri[1], Vincenzo Maffione[2]
[1]Univ. di Pisa, [2]Univ.di Modena e Reggio Emilia
rizzo.unipi@gmail.com. Work supported by H2020 project SSICLOPS.

PCIe Limitations:



NIC doorbell and update limitations:[1]

1. Latency Limitations:
   - 120-900ns
2. Throughput Limitations:
   - ~3Mops (Intel XL710 40Gbps)

1 PSPAT: software packet scheduling at hardware speed

Luigi Rizzo[1], Paolo Valente[2], Giuseppe Lettieri[1], Vincenzo Maffione[2]
[1]Univ. di Pisa, [2]Univ.di Modena e Reggio Emilia
rizzo.unipi@gmail.com. Work supported by H2020 project SSICLOPS.

Loom Goal: Less than 1Mops @ 100Gbps

Loom Efficient Interface Challenges

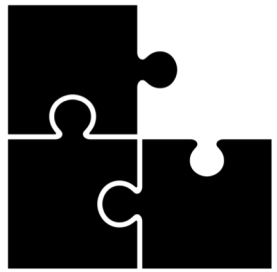
Too many PCIe writes:

In the worst case (every packet is from a new flow), the OS must generate 2 PCIe writes per-packet

2 writes per 1500B packet at 100Gbps = 16.6 Mops!

Insufficient data:

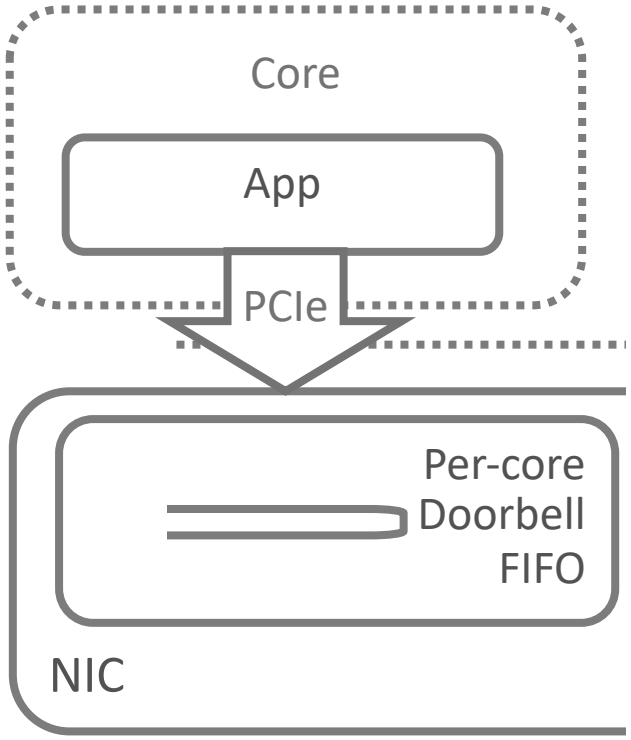
Before reading any packet data (headers), the NIC must schedule DMA reads for a queue



Loom Design

Loom introduces a new efficient OS/NIC interface that reduces the number of PCIe writes through **batched updates** and **inline metadata**

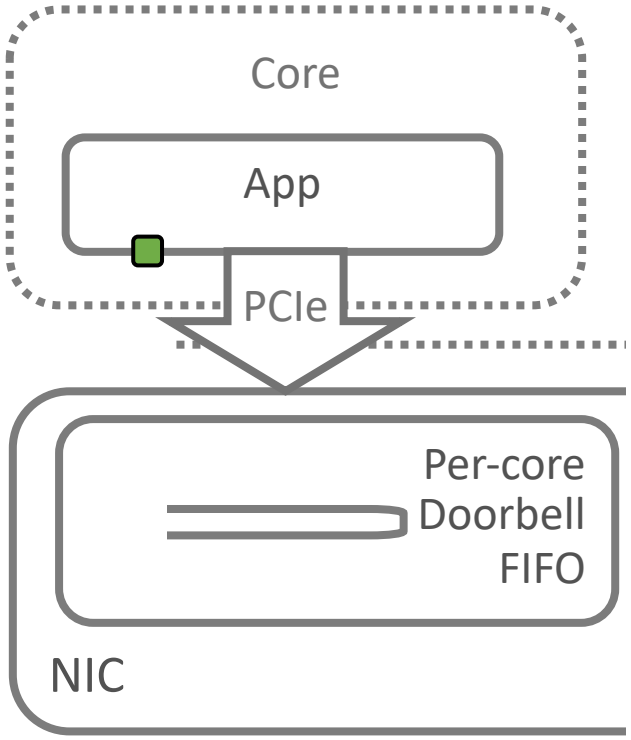
Batched Doorbells



Using on-NIC Doorbell FIFOs allows for updates to different queues (flows) to be **batched**

- Per-core FIFOs still enable parallelism

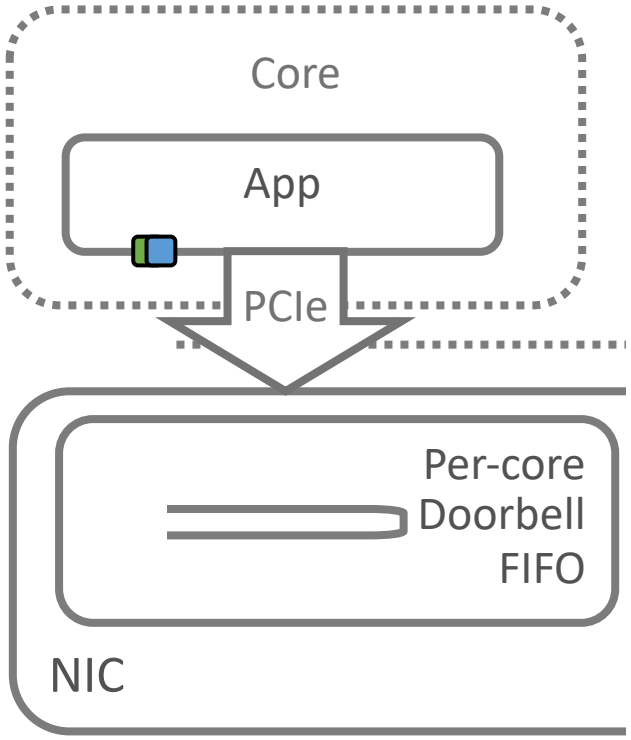
Batched Doorbells



Using on-NIC Doorbell FIFOs allows for updates to different queues (flows) to be **batched**

- Per-core FIFOs still enable parallelism

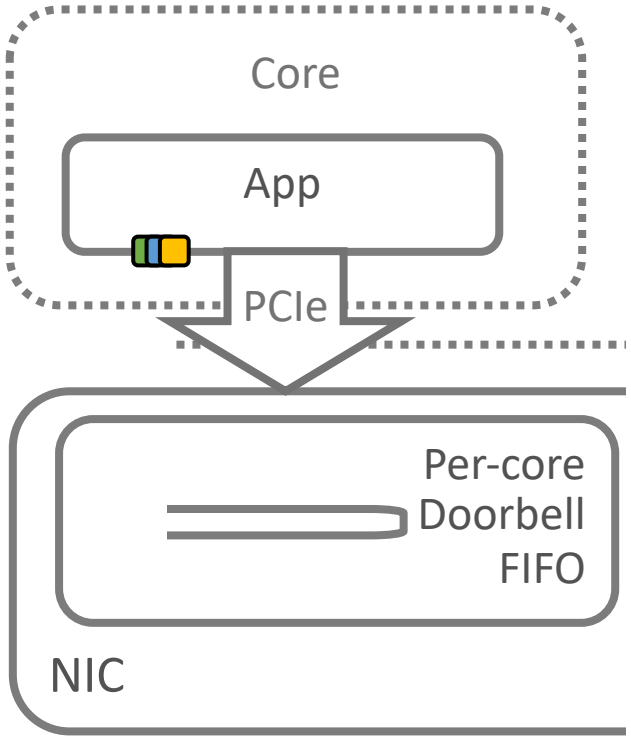
Batched Doorbells



Using on-NIC Doorbell FIFOs allows for updates to different queues (flows) to be **batched**

- Per-core FIFOs still enable parallelism

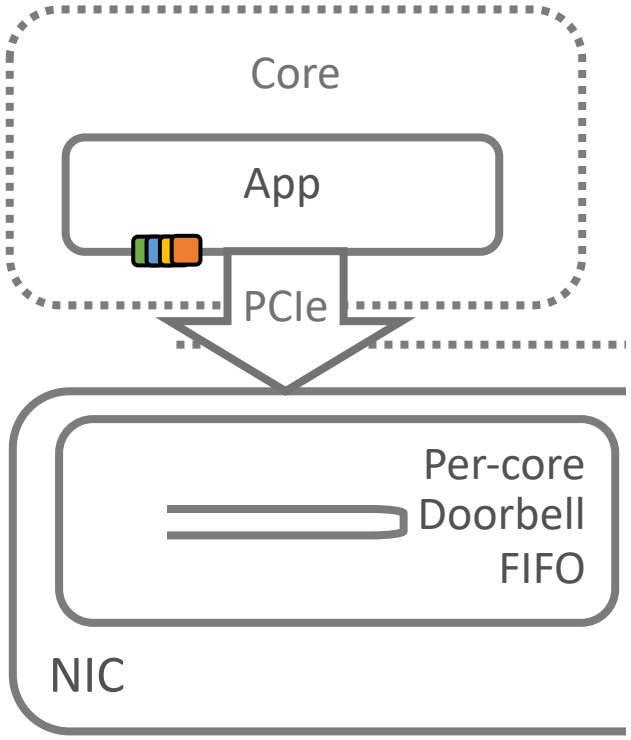
Batched Doorbells



Using on-NIC Doorbell FIFOs allows for updates to different queues (flows) to be **batched**

- Per-core FIFOs still enable parallelism

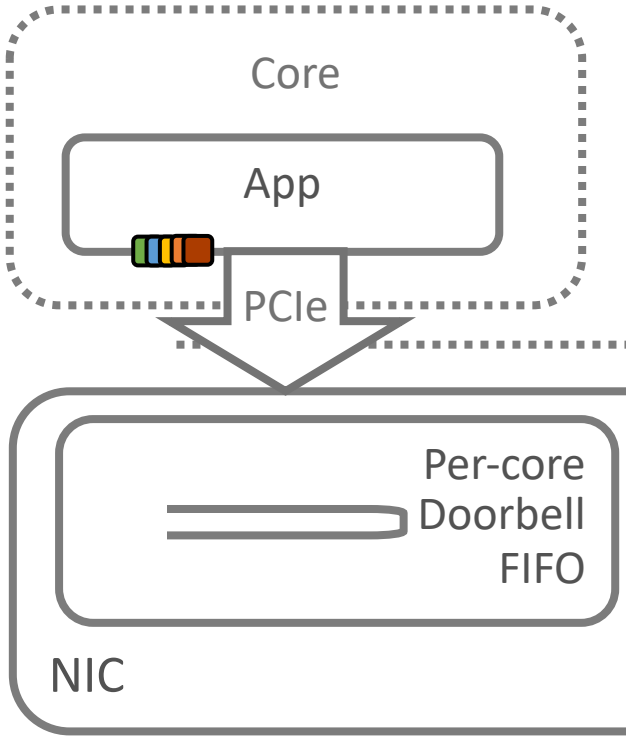
Batched Doorbells



Using on-NIC Doorbell FIFOs allows for updates to different queues (flows) to be **batched**

- Per-core FIFOs still enable parallelism

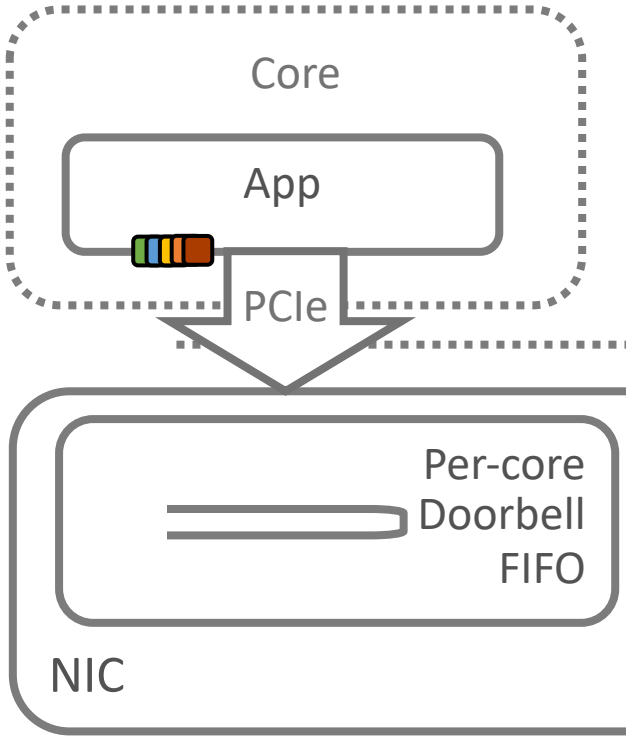
Batched Doorbells



Using on-NIC Doorbell FIFOs allows for updates to different queues (flows) to be **batched**

- Per-core FIFOs still enable parallelism

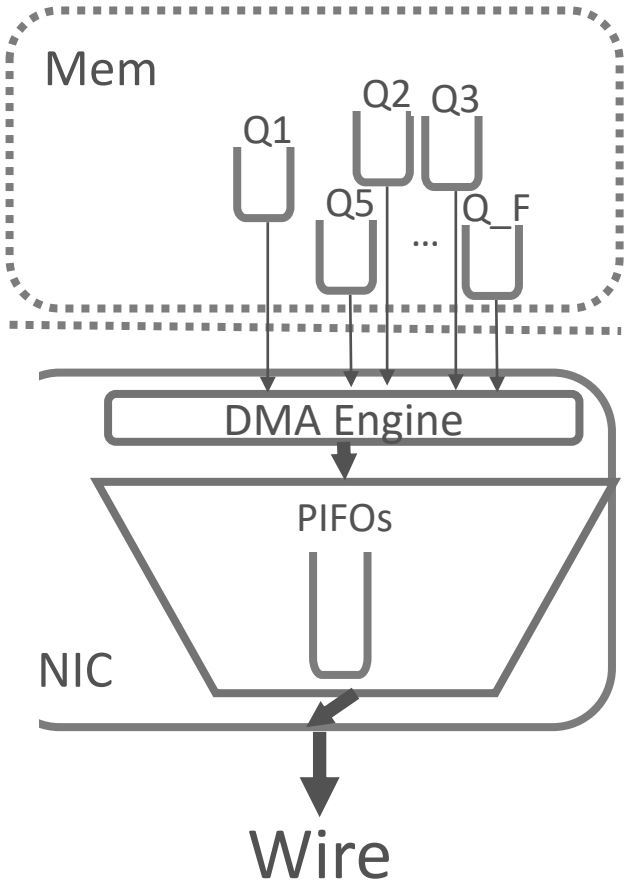
Batched Doorbells



Using on-NIC Doorbell FIFOs allows for updates to different queues (flows) to be **batched**

- Per-core FIFOs still enable parallelism

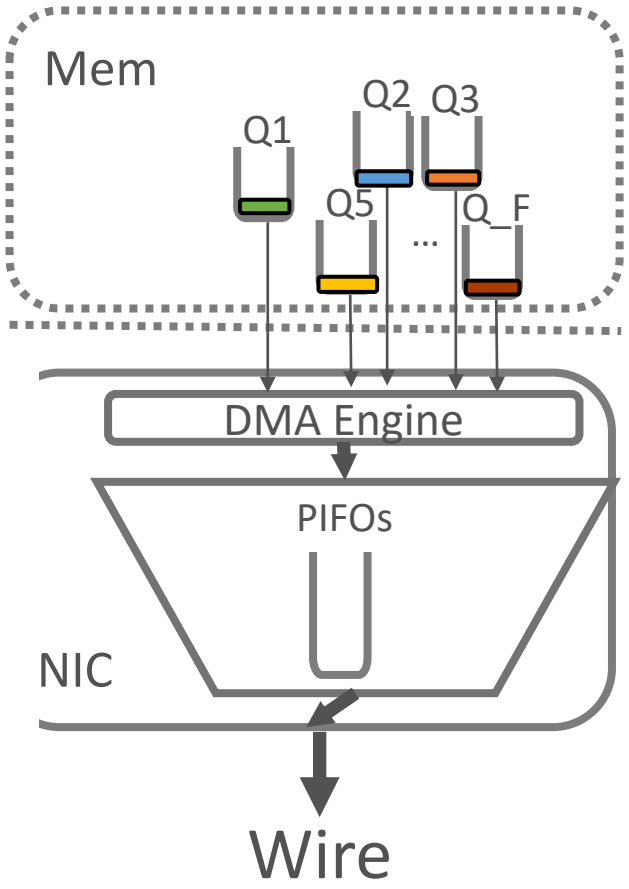
Inline Metadata



Scheduling metadata (traffic class and scheduling updates) is **inlined** to reduce PCIe writes

- Descriptor inlining allows for scheduling before reading packet data

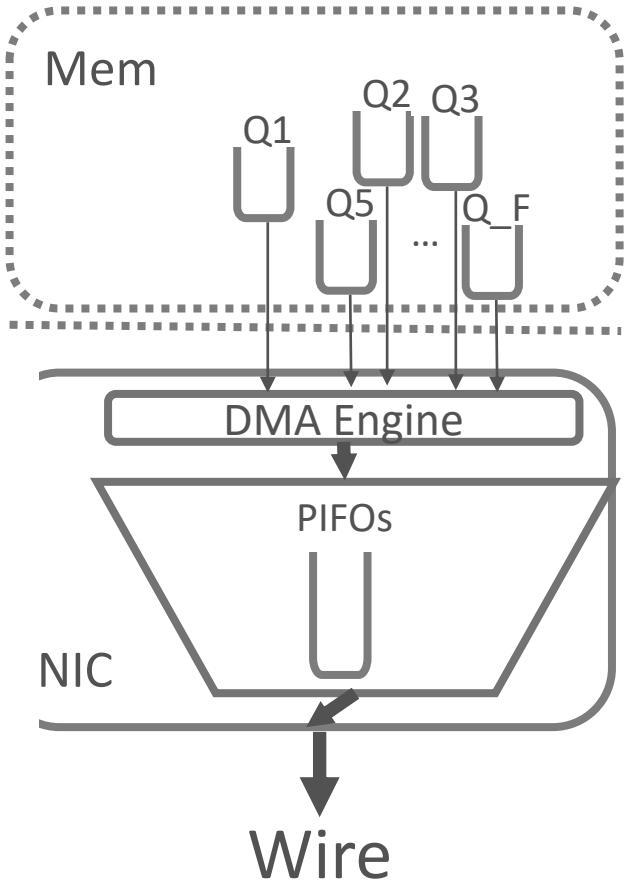
Inline Metadata



Scheduling metadata (traffic class and scheduling updates) is **inlined** to reduce PCIe writes

- Descriptor inlining allows for scheduling before reading packet data

Inline Metadata



Scheduling metadata (traffic class and scheduling updates) is **inlined** to reduce PCIe writes

- Descriptor inlining allows for scheduling before reading packet data

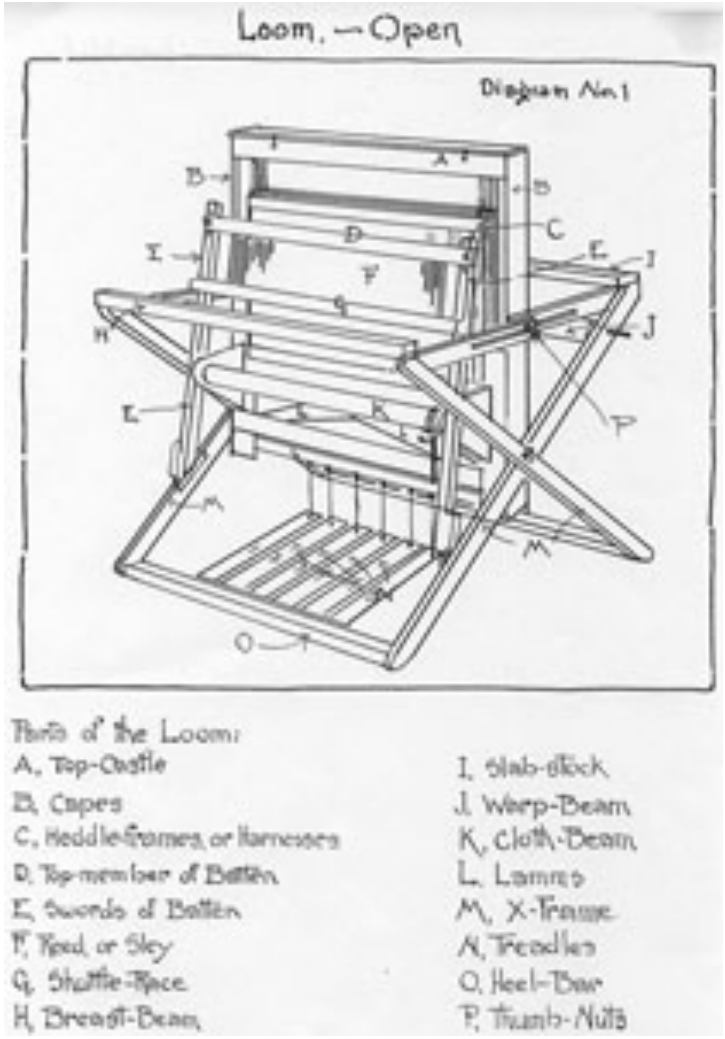
Outline

Contributions:

1. A new network policy abstraction: restricted directed acyclic graphs (DAGs)
2. A new programmable packet scheduling hierarchy designed for NICs
3. A new expressive and efficient OS/NIC interface

Evaluation:

1. **Implementation and Evaluation: BESS prototype and CloudLab**



Loom Implementation

- Software prototype of Loom in Linux on the Berkeley Extensible Software Switch (BESS)[1]
- C++ PIFO[2] implementation is used for scheduling
- 10Gbps and 40Gbps CloudLab evaluation

http://github.com/bestephe/loom

1

BESS

Berkeley Extensible Software Switch

2

Programmable Packet Scheduling at Line Rate

Anirudh Sivaraman*, Suvinay Subramanian*, Mohammad Alizadeh*, Sharad Chole‡, Shang-Tse Chuang‡, Anurag Agrawal†, Hari Balakrishnan*, Tom Edsall‡, Sachin Katti+, Nick McKeown+
*MIT CSAIL, †Barefoot Networks, ‡Cisco Systems, +Stanford University

Loom Evaluation

Can Loom drive line rate? Can Loom enforce network policies?

Experiment: Microbenchmarks with iPerf

Can Loom isolate real applications?

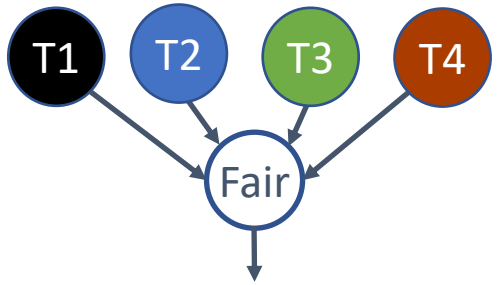
Experiment: CloudLab experiments with memcached and Spark

How effective is Loom's efficient OS/NIC interface?

Experiment: Analysis of PCIe writes in Linux (QPF) versus Loom

Loom 40Gbps Evaluation

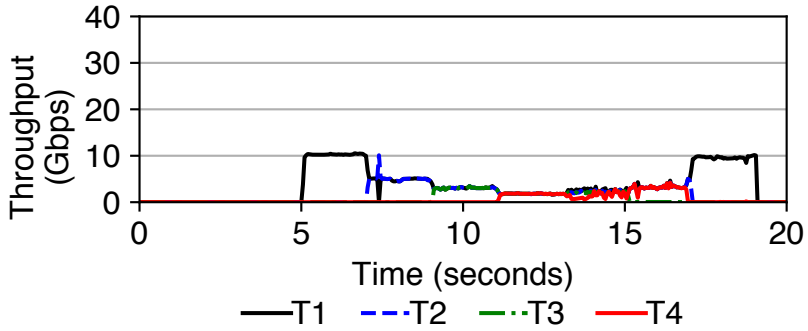
Policy: All tenants should receive an equal share.



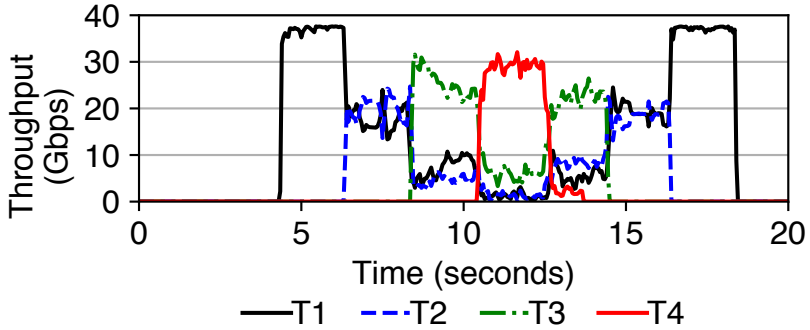
Setup:

- Every 2s a new tenant starts or stops
- Each tenant *i* starts 4^i flows (4-256 total flows)

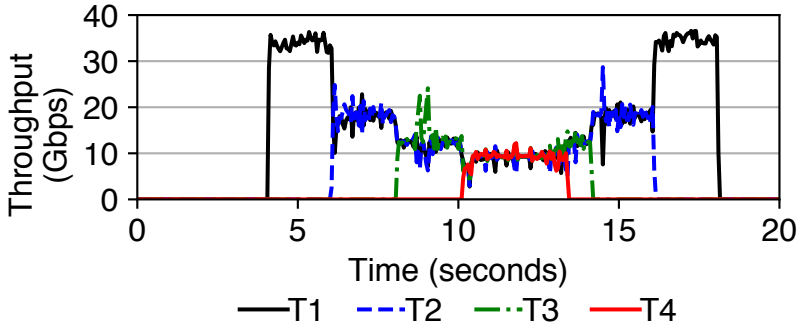
SQ



MQ

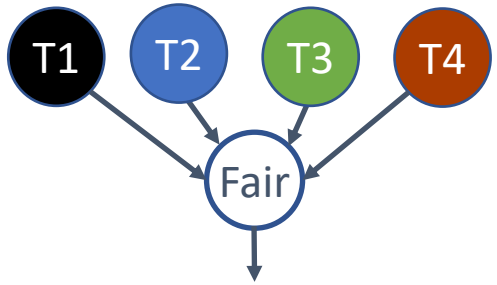


Loom



Loom 40Gbps Evaluation

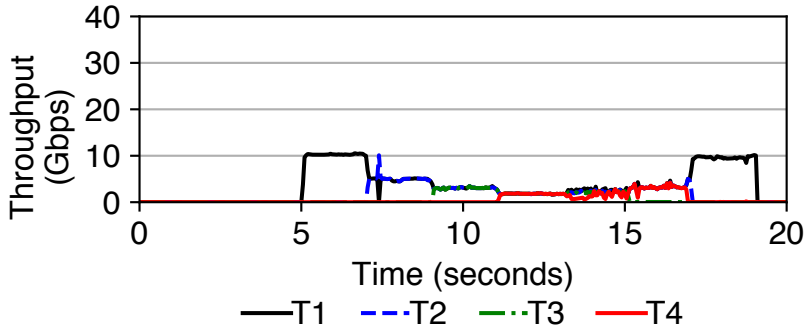
Policy: All tenants should receive an equal share.



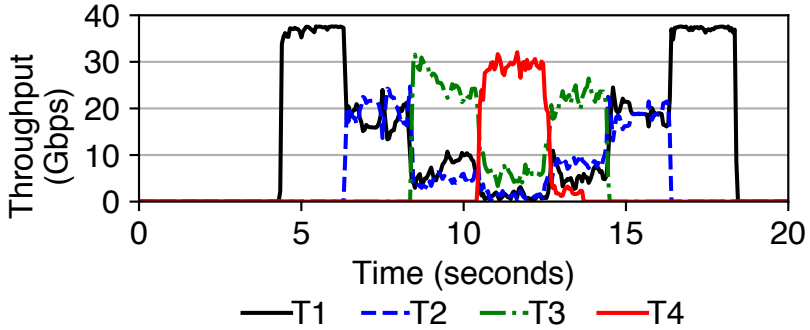
Setup:

- Every 2s a new tenant starts or stops
- Each tenant *i* starts 4^i flows (4-256 total flows)

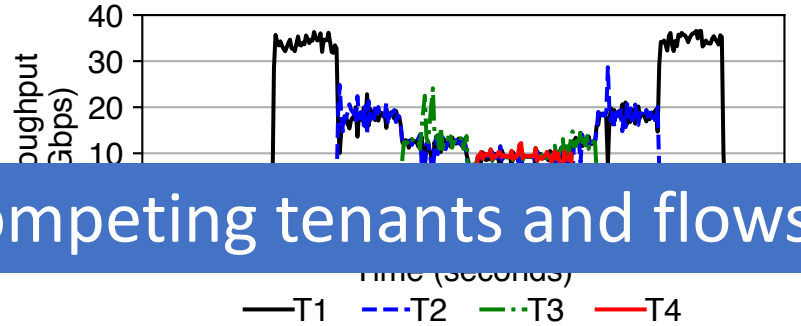
SQ



MQ



Loom



Loom can drive line-rate and isolate competing tenants and flows

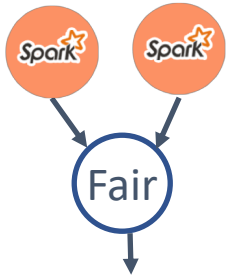
Application Performance: Fairness

vs

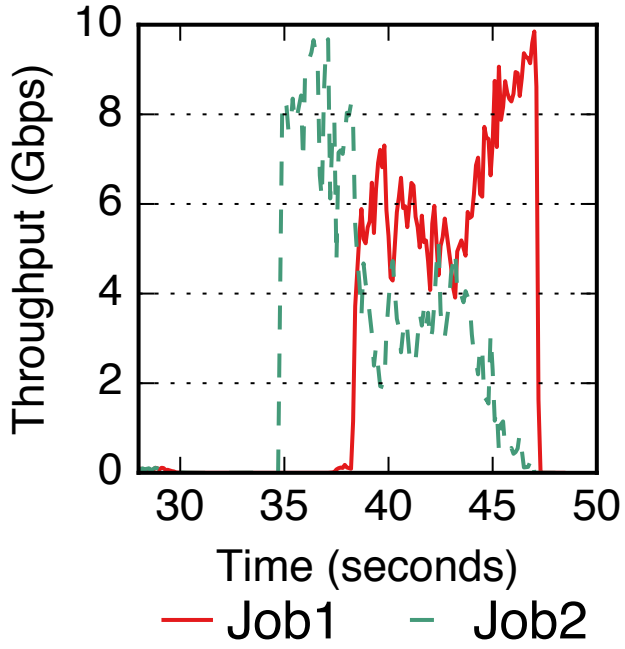
Bandwidth Hungry

Bandwidth Hungry

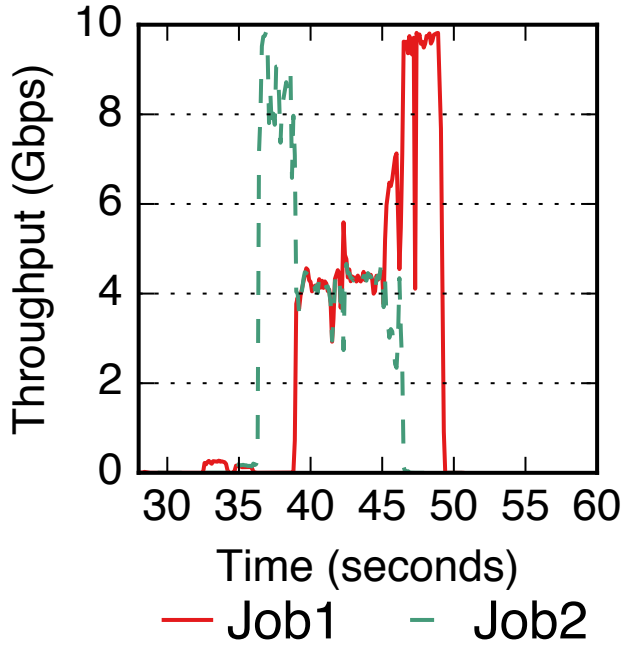
Policy: Bandwidth is fairly shared between Spark jobs



Linux



Loom

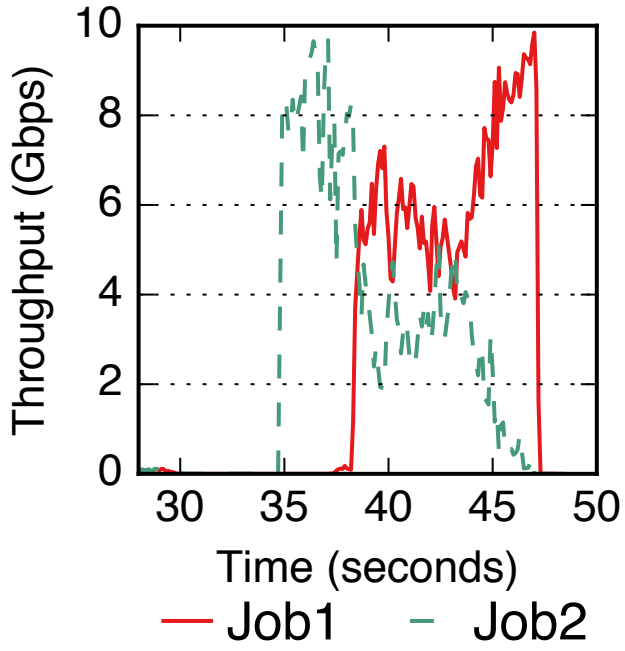


Application Performance: Fairness

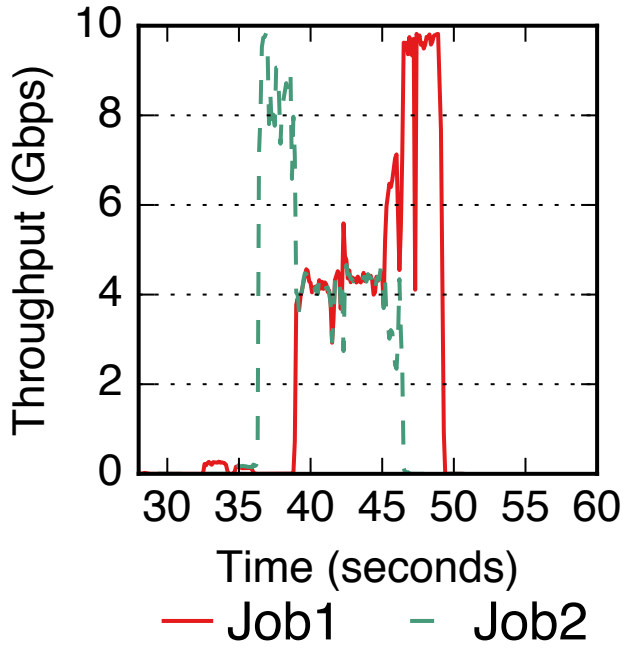
Bandwidth Hungry vs Bandwidth Hungry

Policy: Bandwidth is fairly shared between Spark jobs

Linux



Loom



Loom can ensure competing jobs share bandwidth even if they have different numbers of flows

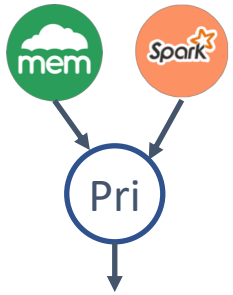
Application Performance: Latency

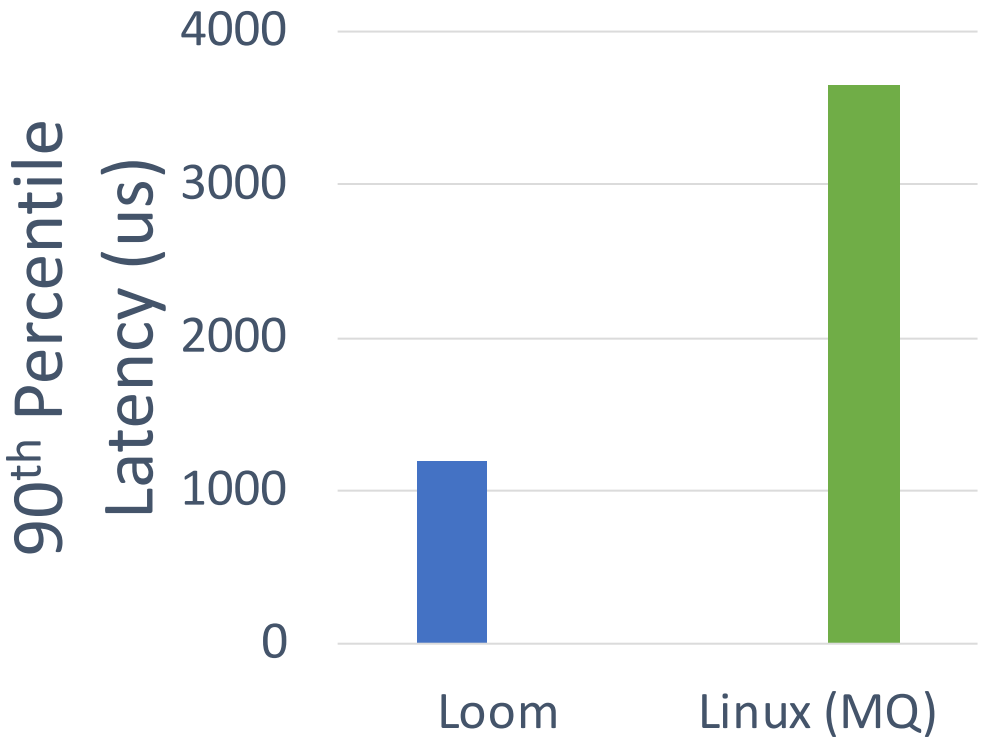
Latency Sensitive

vs

Bandwidth Hungry

Setup: Linux software packet scheduling (Qdisc) is configured to prioritize memcached traffic over Spark traffic





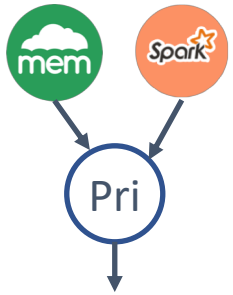
Application Performance: Latency

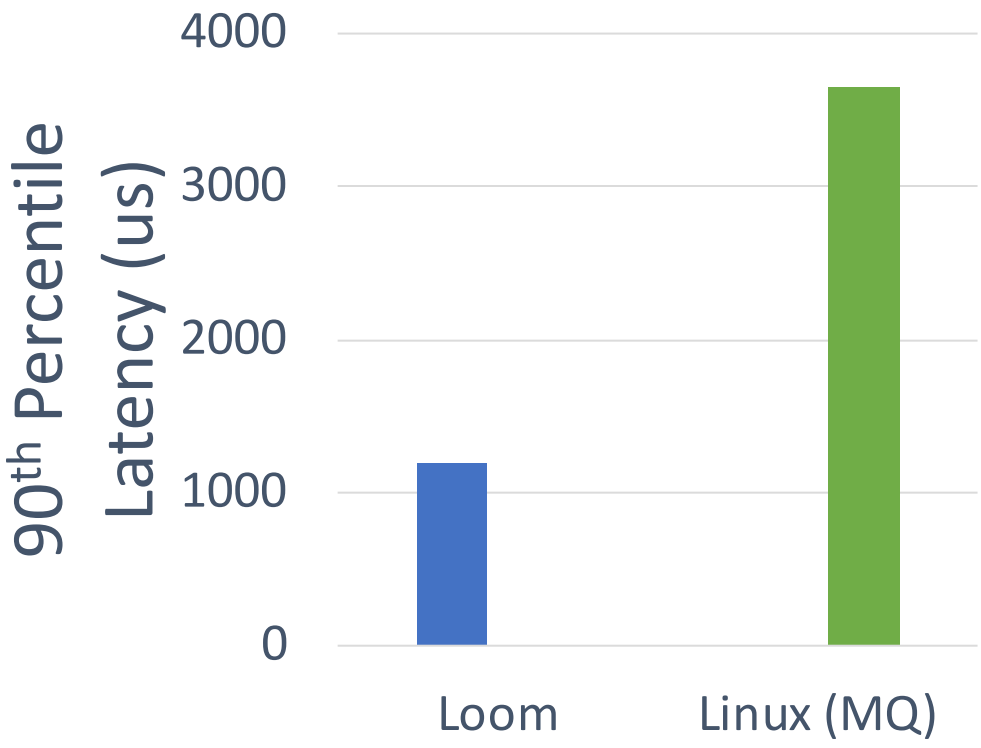
Latency Sensitive

vs

Bandwidth Hungry

Setup: Linux software packet scheduling (Qdisc) is configured to prioritize memcached traffic over Spark traffic





MQ cannot isolate latency-sensitive applications!

Loom Interface Evaluation

Worse case scenario:
Packets are sent in 64KB batches and each packet is from a different flow

Line-rate	Existing approaches: PCIe Writes per second	Loom: PCIe Writes per second
10 Gbps	833K	19K
40 Gbps	3.3M	76K
100 Gbps	8.3M	191K

Loom Interface Evaluation

Worse case scenario:
Packets are sent in 64KB batches and each packet is from a different flow

Line-rate	Existing approaches: PCIe Writes per second	Loom: PCIe Writes per second
10 Gbps	833K	19K
40 Gbps	3.3M	76K
100 Gbps	8.3M	191K

Loom Goal: Less than 1Mops @ 100Gbps

Conclusion

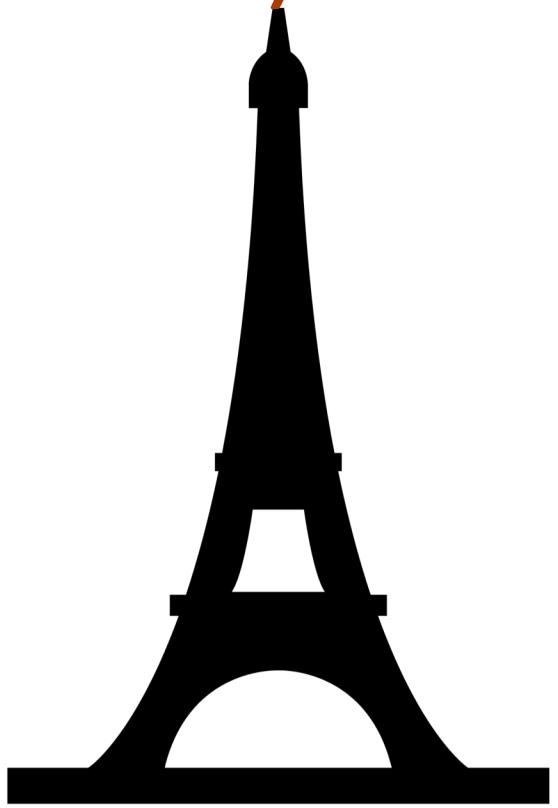
Current NICs cannot ensure that competing applications are isolated

Loom is a new NIC design that completely offloads all packet scheduling to the NIC with low CPU overhead

Loom's benefits translate into reductions in latency, increases in throughput, and improvements in fairness

Related Work (Eiffel)

- NIC Scheduling does not eliminate the need for software scheduling
- Loom and Eiffel can be used together
- Bucketed priority queues could be used to build efficient PIFOs



Eiffel