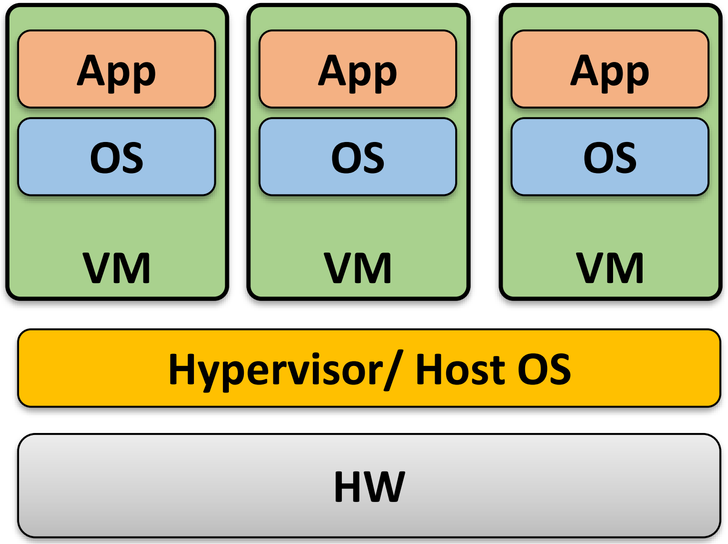
Iron: Isolating Network-based CPU in Container Environments

Junaid Khalid[1], Eric Rozner[2], Wesley Felter[2,3], Cong Xu[2], Karthick Rajamani[2], Alexandre Ferreira[2,4], Aditya Akella[1]

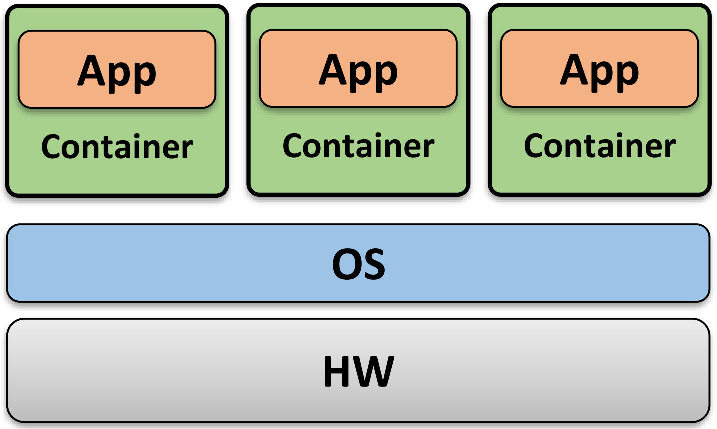
[3]Now at State Street, [4]Now at ARM

Virtualization

Virtual machines

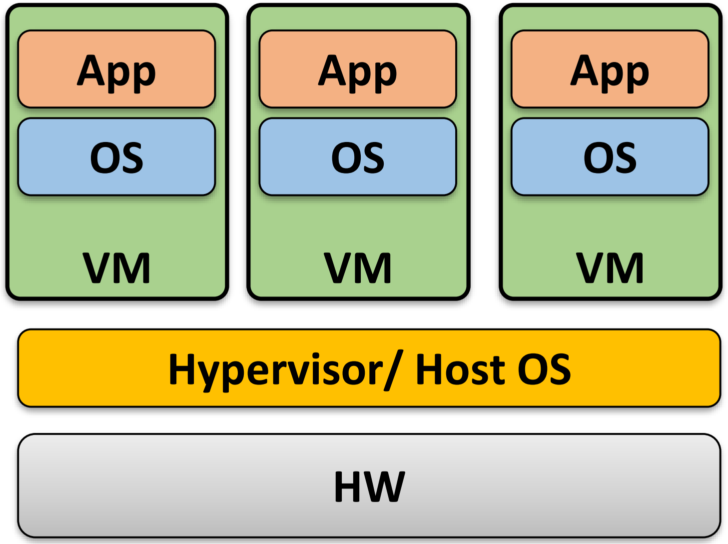


Containers



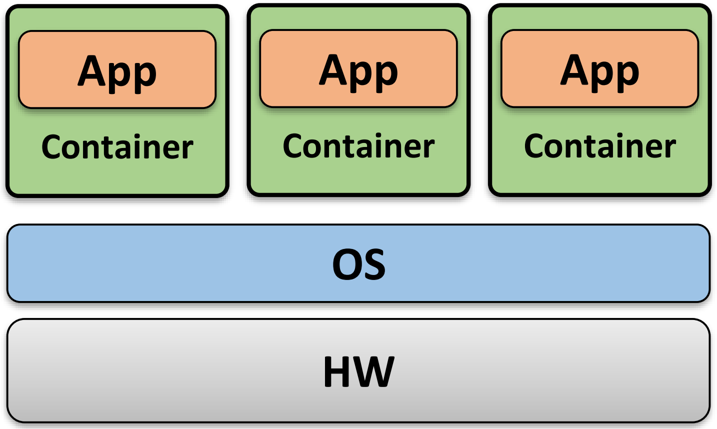
Virtualization

Virtual machines



- ✗ Heavy weight
- ✗ Slow

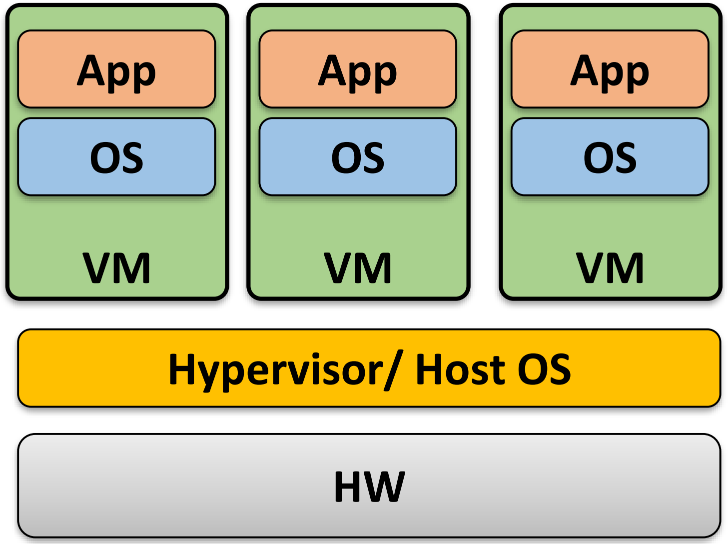
Containers



- ✓ Light weight
- ✓ Fast

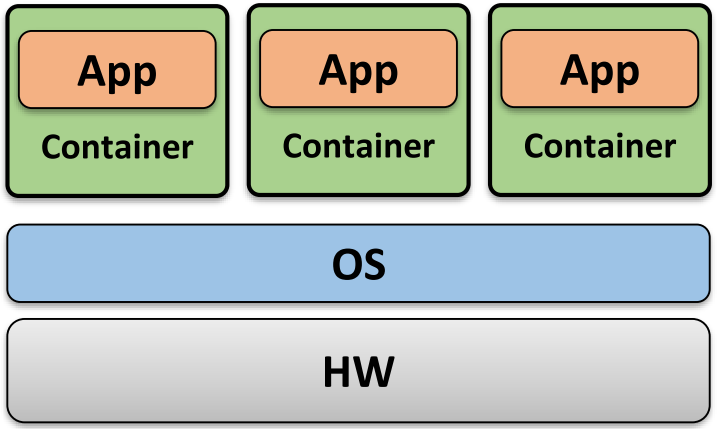
Virtualization

Virtual machines



- ✗ **Heavy weight**
- ✗ **Slow**

Containers

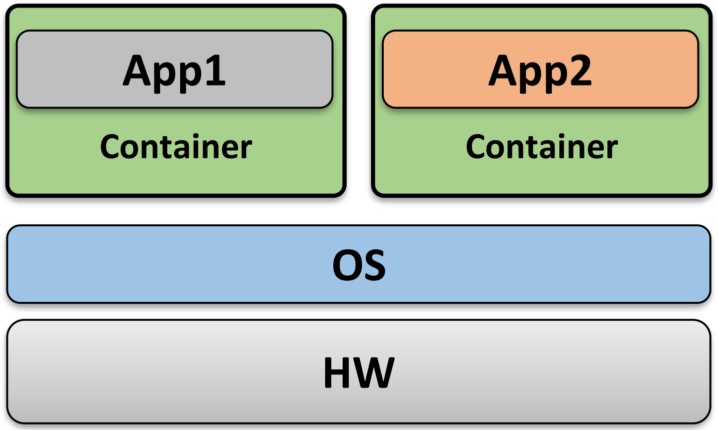


- ✓ **Light weight**
- ✓ **Fast**

Supports new workloads such as microservices and serverless

Isolation

Containers

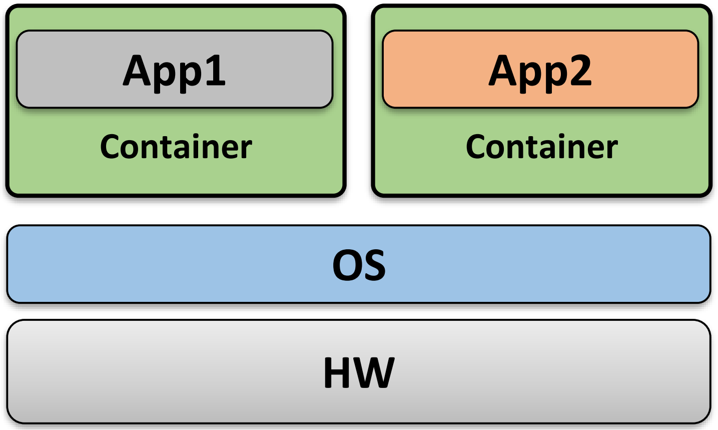


Isolation

Containers

Containers require strong resource isolation

- Memory
- Network
- CPU



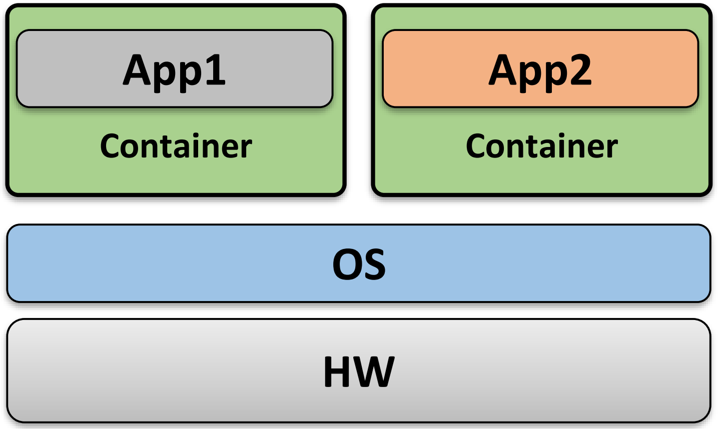
Isolation

Containers

Containers require strong resource isolation

- Memory
- Network
- CPU

Administrators want to strongly control resource allocation in multi-tenant environments



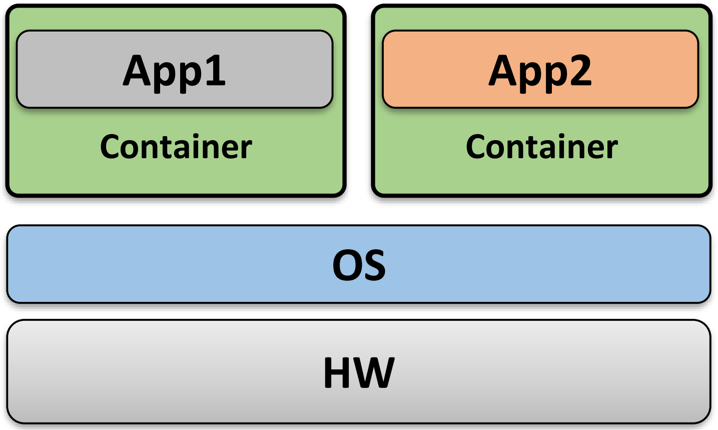
Isolation

Containers

Containers require strong resource isolation

- Memory
- Network
- CPU

Administrators want to strongly control resource allocation in multi-tenant environments

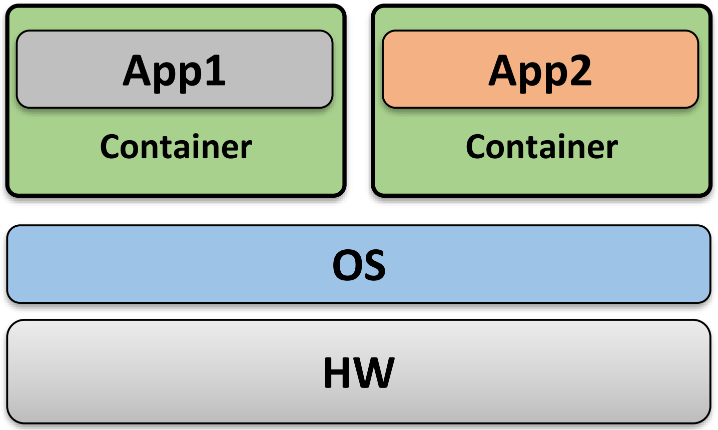


Strong isolation is important for ***performance***, ***predictability*** and ***efficiency***

Isolation

Containers

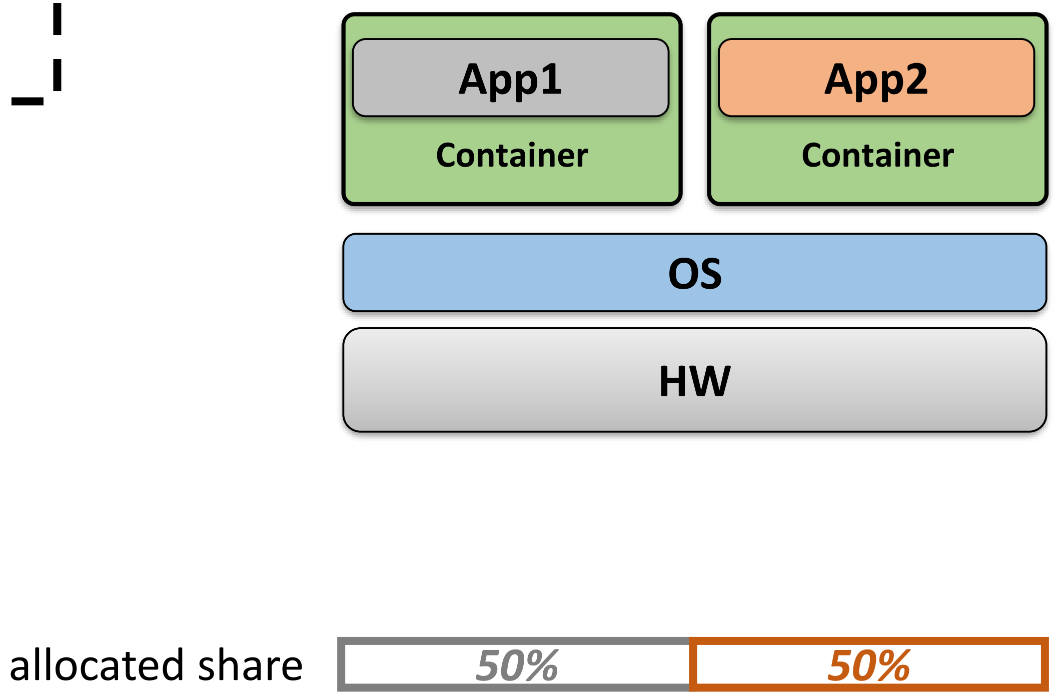
Isolation: A container ***shouldn't consume*** more than its assigned share of ***resources***



Isolation

Containers

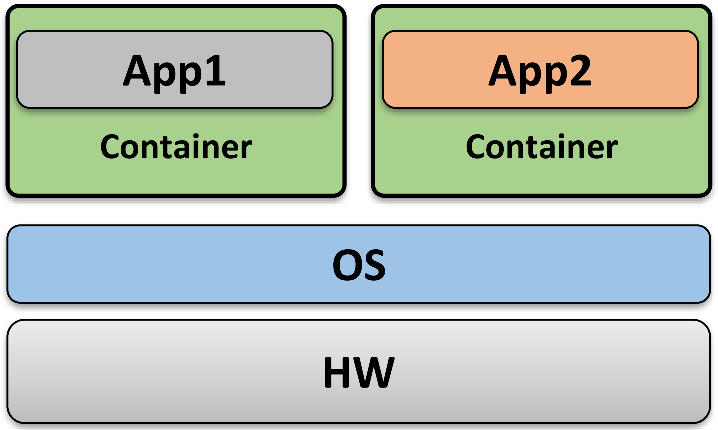
Isolation: A container ***shouldn't consume*** more than its assigned share of ***resources***



Isolation

Containers

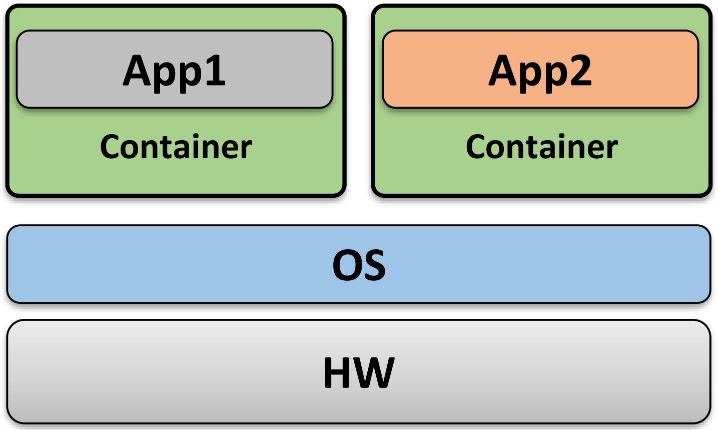
Isolation: A container ***shouldn't consume*** more than its assigned share of ***resources***



Isolation

Containers

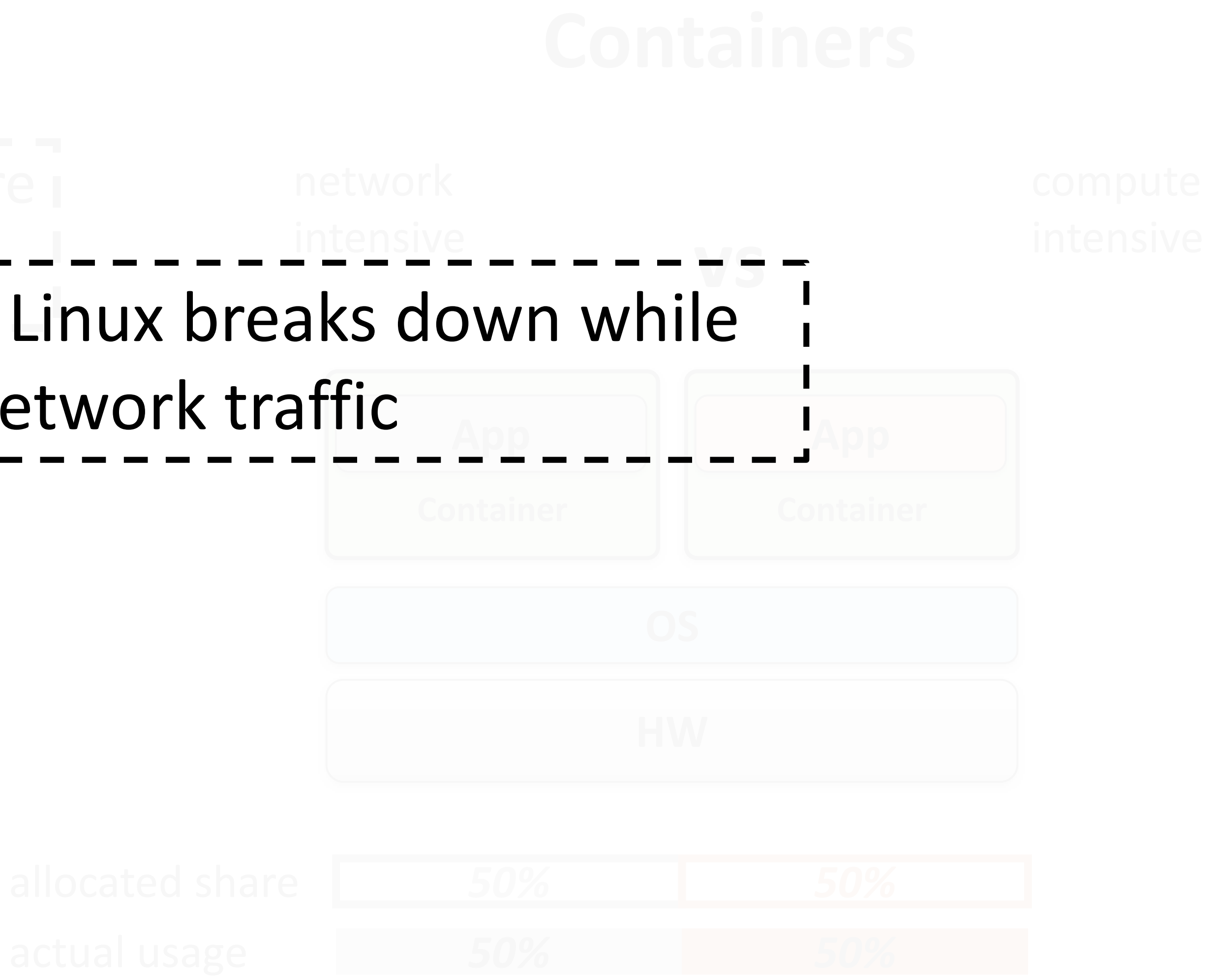
Isolation: A container ***shouldn't consume*** more than its assigned share of ***resources***



cgroups ensures **CPU isolation** by ***allocating***, ***metering***, and ***enforcing*** resource usage in the kernel

Isolation

Isolation: A container ***shouldn't consume*** more than its assigned share of ***resources***



CPU isolation provided by Linux breaks down while handling the network traffic

cGroup ensures **resource isolation** by ***allocating***, ***metering***, and ***enforcing*** resource usage in the kernel

Outline

- How and by how much is isolation broken

Outline

- How and by how much is isolation broken
- Iron's design
  - Accounting of per-packet processing cost
  - Ensuring isolation via enforcement
    - Integration with Linux scheduler
    - Hardware-based packet dropping

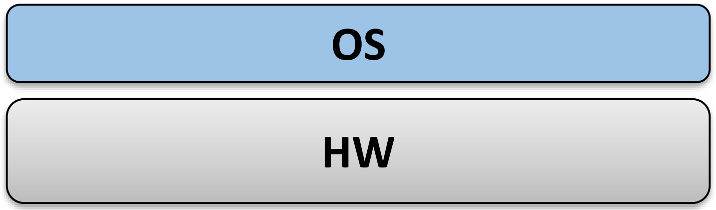
Outline

- How and by how much is isolation broken
- Iron's design
  - Accounting of per-packet processing cost
  - Ensuring isolation via enforcement
    - Integration with Linux scheduler
    - Hardware-based packet dropping
- Evaluation
  - Controlled workload
  - Realistic workload

Isolation is Broken

Containers

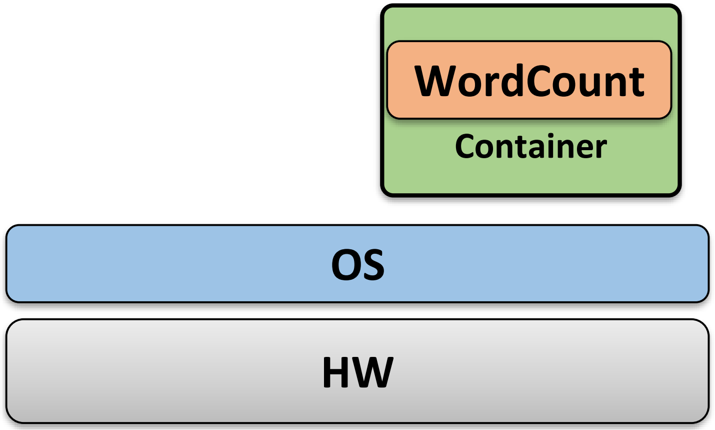
$$\textit{Penalty factor} = \frac{\textit{Time that job takes when competing with traffic}}{\textit{Time that job takes when competing with compute}}$$



Isolation is Broken

$$\textit{Penalty factor} = \frac{\textit{Time that job takes when competing with traffic}}{\textit{Time that job takes when competing with compute}}$$

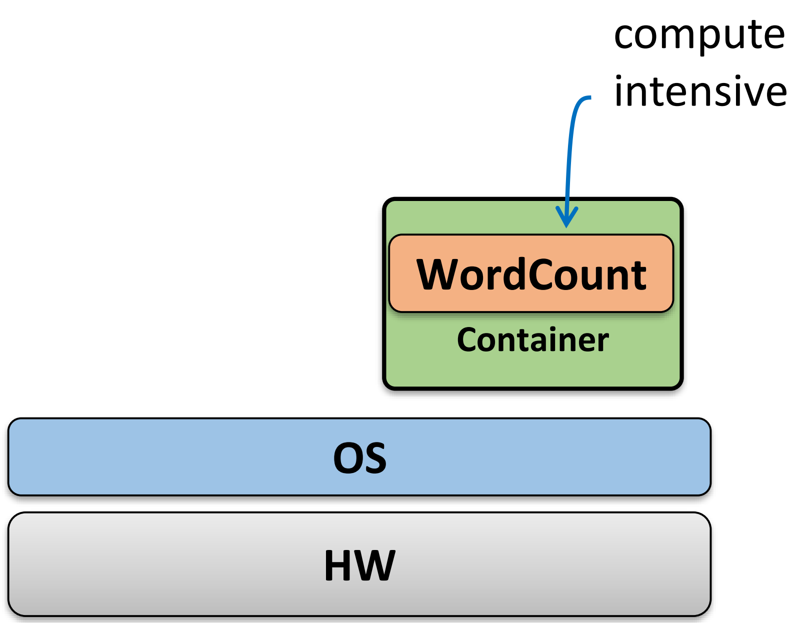
Containers



Isolation is Broken

Penalty factor = *Time that job takes when competing with traffic* / *Time that job takes when competing with compute*

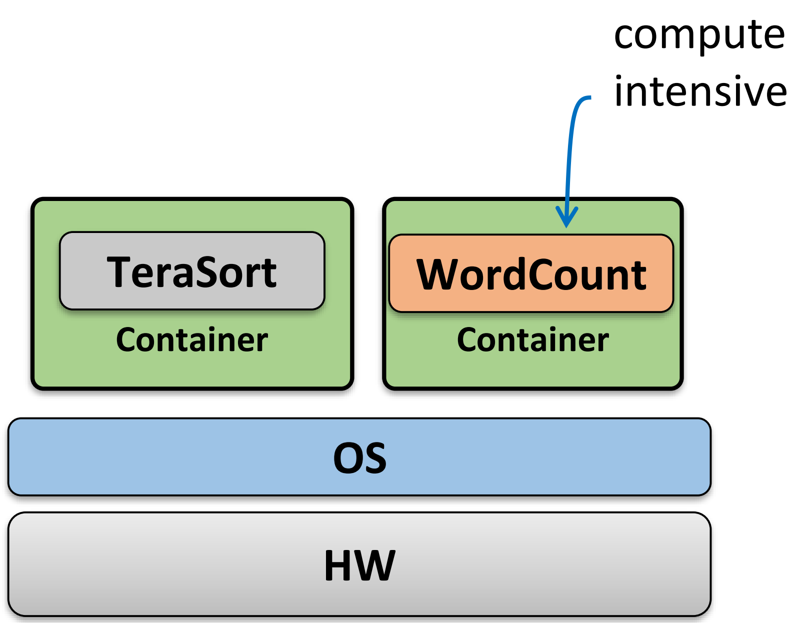
Containers



Isolation is Broken

Penalty factor = *Time that job takes when competing with traffic* / *Time that job takes when competing with compute*

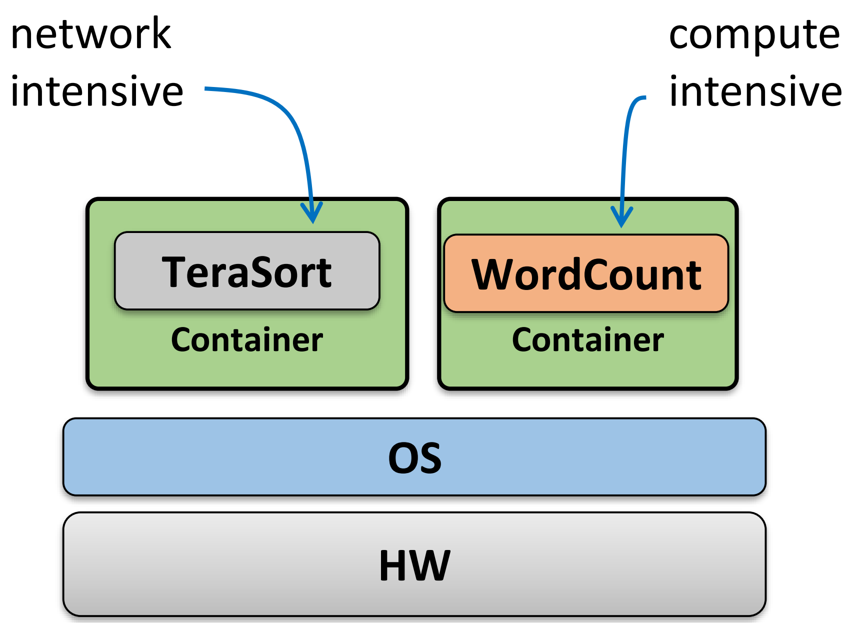
Containers



Isolation is Broken

Containers

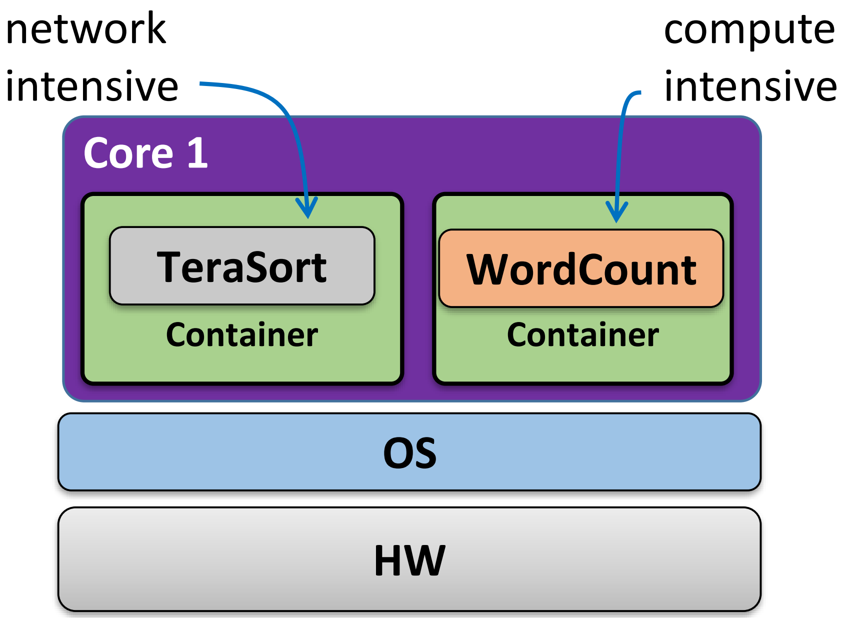
$$\textit{Penalty factor} = \frac{\textit{Time that job takes when competing with traffic}}{\textit{Time that job takes when competing with compute}}$$



Isolation is Broken

Containers

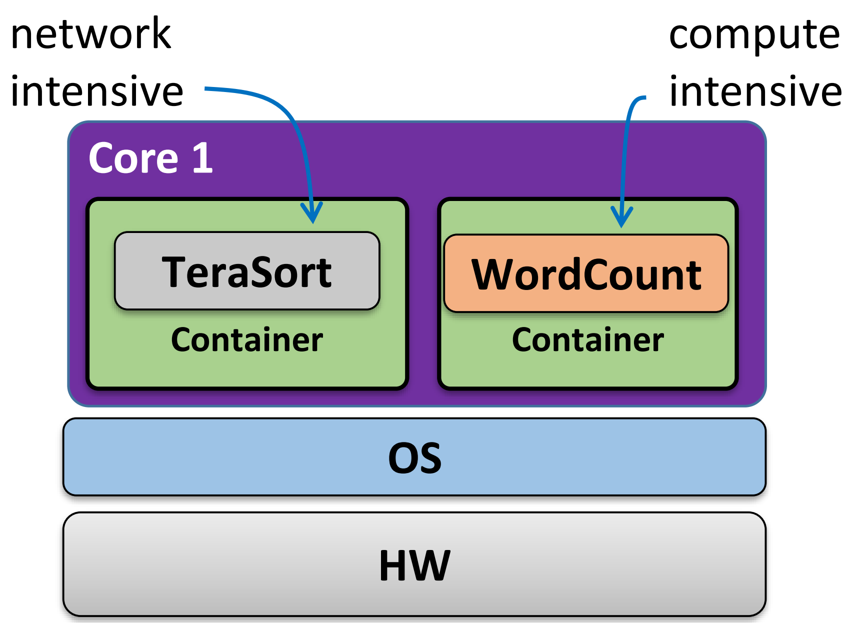
$$\textit{Penalty factor} = \frac{\textit{Time that job takes when competing with } \textit{\textcolor{red}{traffic}}}{\textit{Time that job takes when competing with } \textit{\textcolor{blue}{compute}}}$$



Isolation is Broken

Containers

Penalty factor = *Time that job takes when competing with traffic* / *Time that job takes when competing with compute*

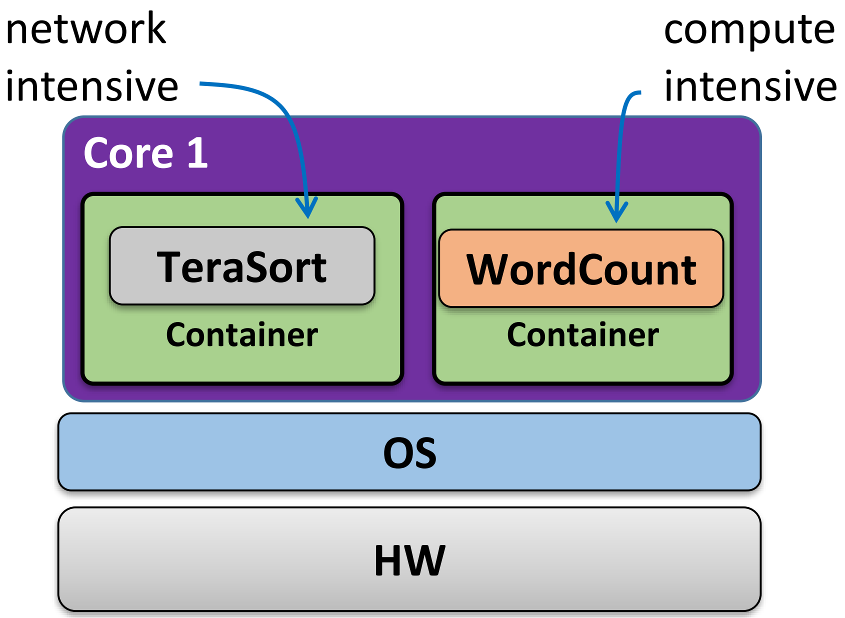


Wordcount can take ***1.5x*** longer when it shares the core with ***TeraSort***

Isolation is Broken

Containers

Penalty factor = *Time that job takes when competing with traffic* / *Time that job takes when competing with compute*



Wordcount can take ***1.5x*** longer when it shares the core with ***TeraSort***

vs

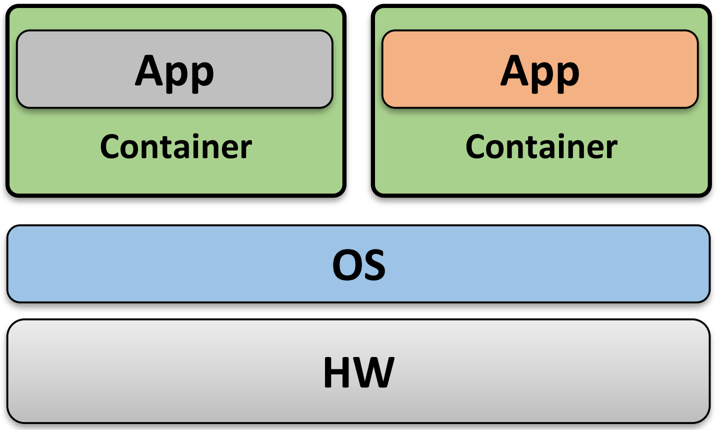
running alone

Practical Implications

Containers

1) Insufficient isolation → overcharging & high variance in the performance

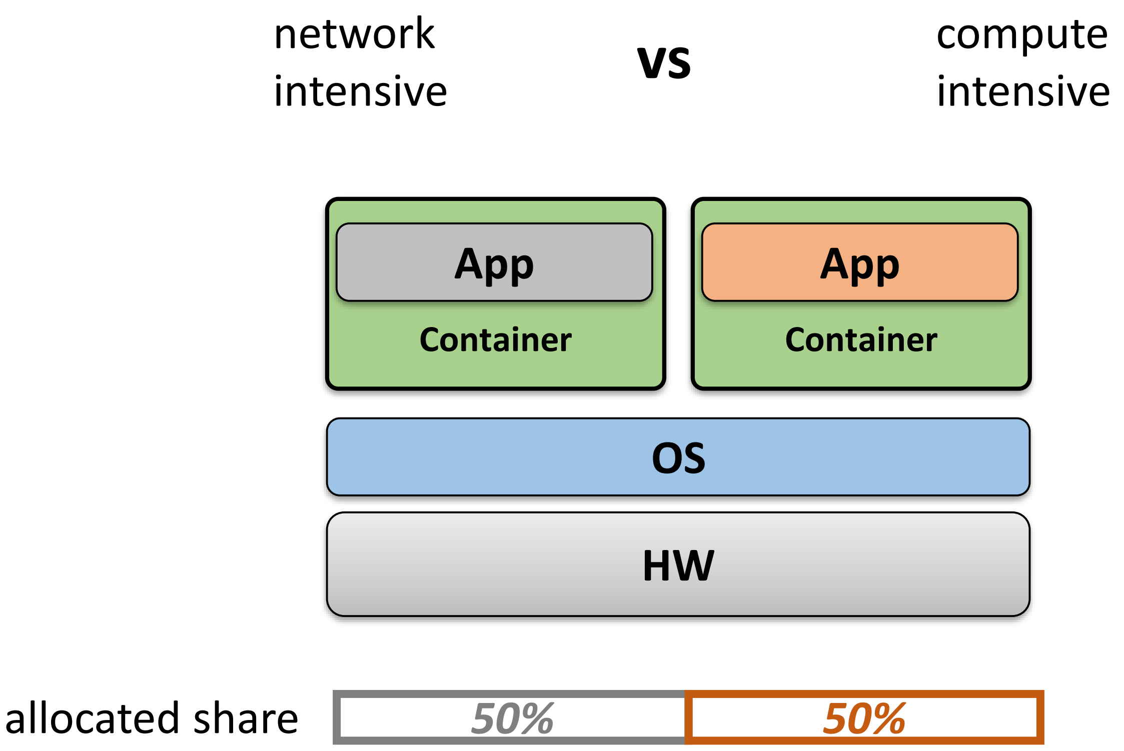
network intensive **vs** compute intensive



Practical Implications

Containers

1) Insufficient isolation → overcharging & high variance in the performance

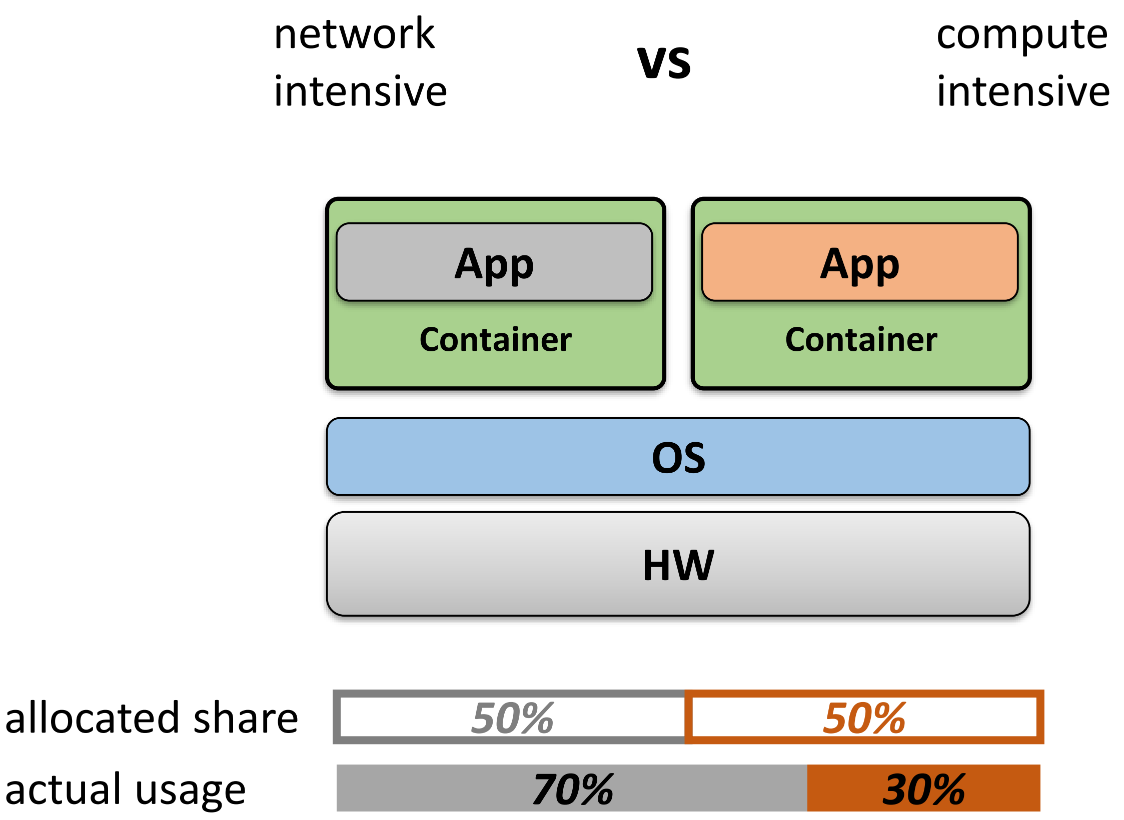


Practical Implications

1) Insufficient isolation →

overcharging & high variance in the performance

Containers



Practical Implications

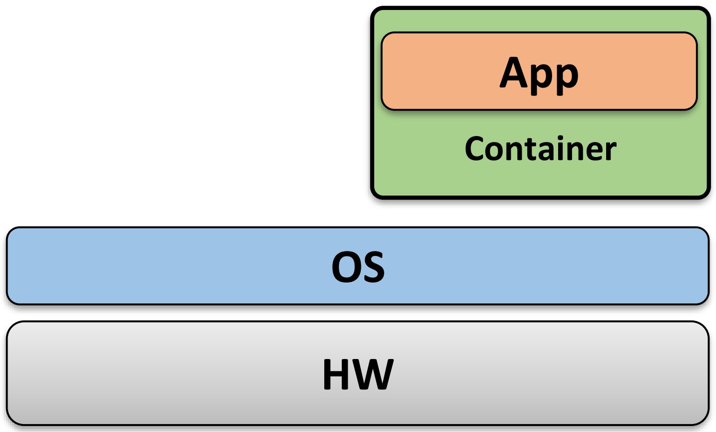
Containers

1) Insufficient isolation →

overcharging & high variance
in the performance

2) Under provisioning →

waste of potential revenue



Practical Implications

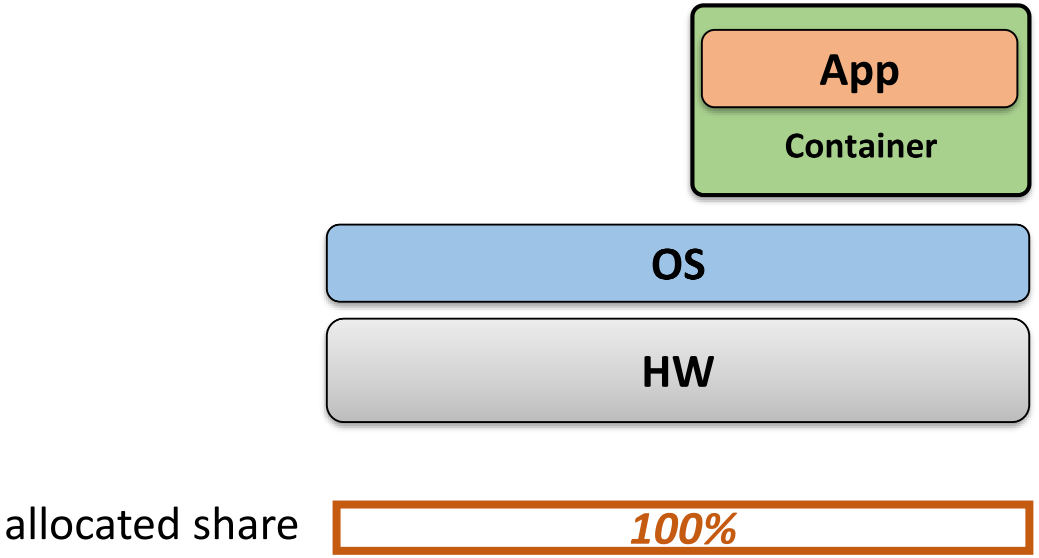
Containers

1) Insufficient isolation →

overcharging & high variance
in the performance

2) Under provisioning →

waste of potential revenue



Practical Implications

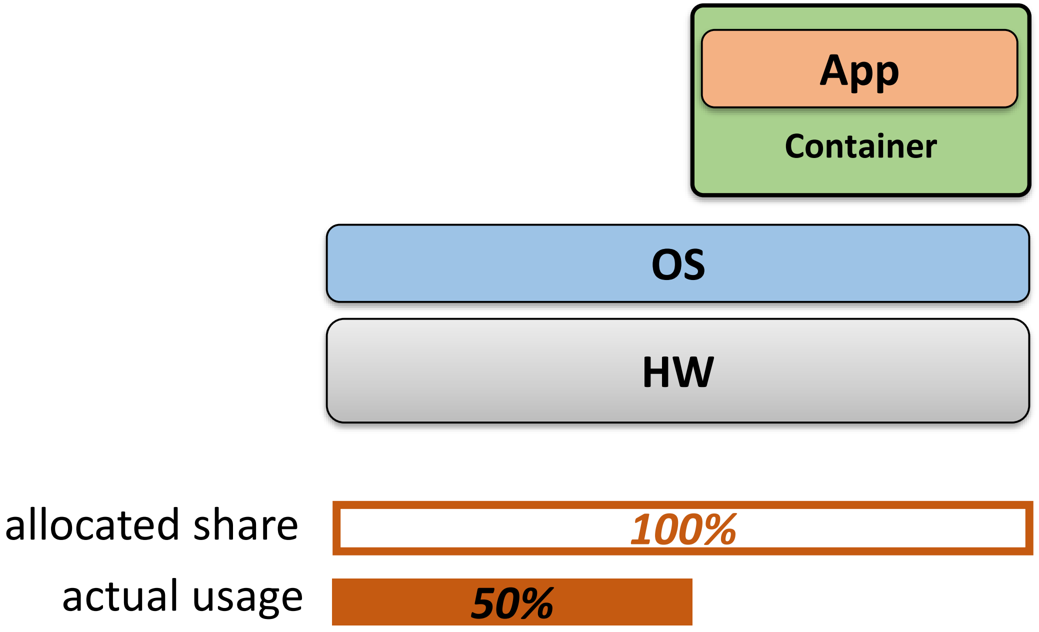
Containers

1) Insufficient isolation →

overcharging & high variance in the performance

2) Under provisioning →

waste of potential revenue



Practical Implications

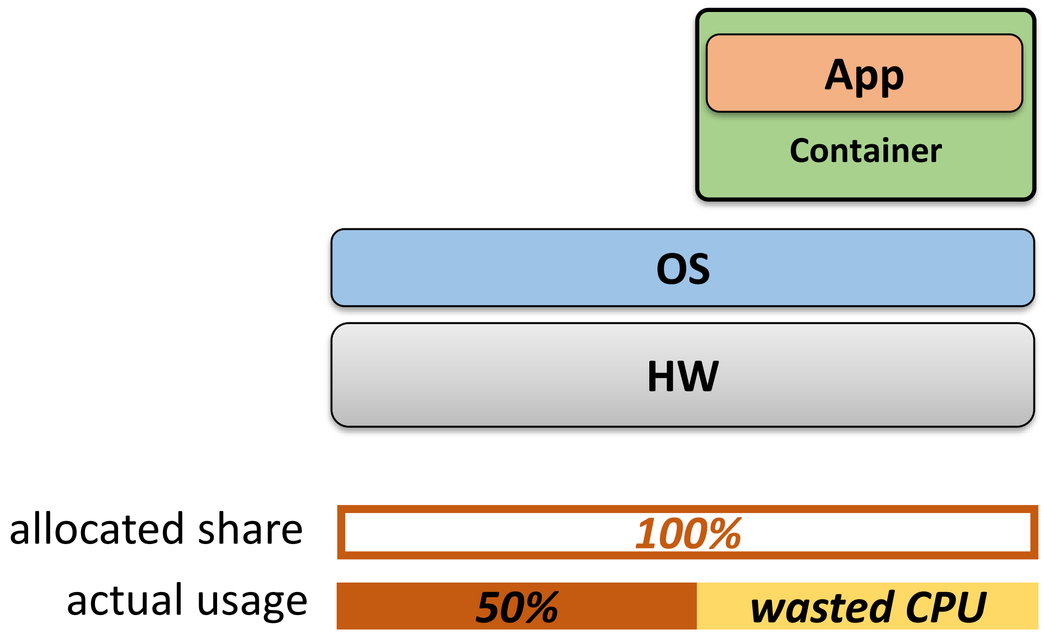
Containers

1) Insufficient isolation →

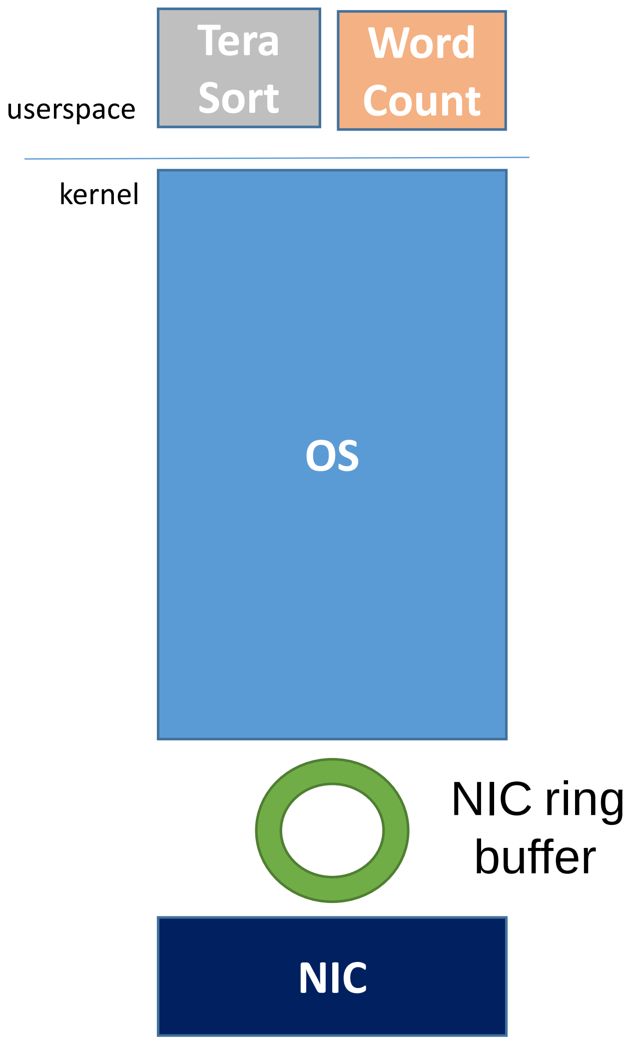
overcharging & high variance in the performance

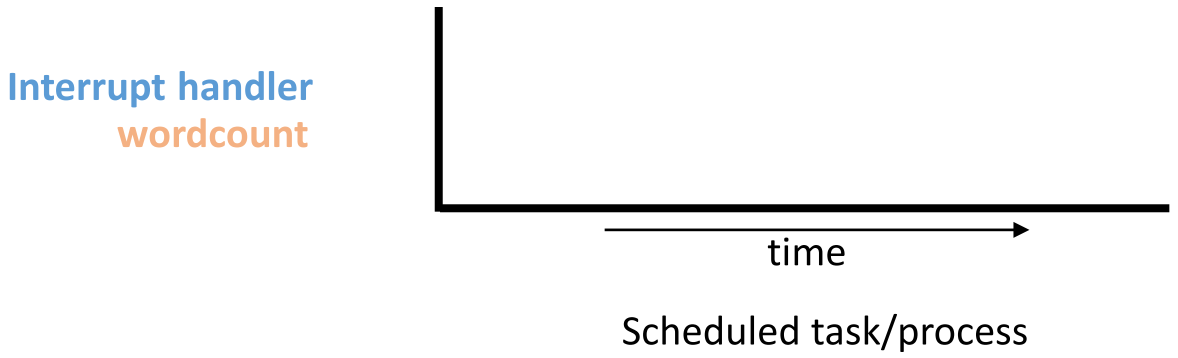
2) Under provisioning →

waste of potential revenue

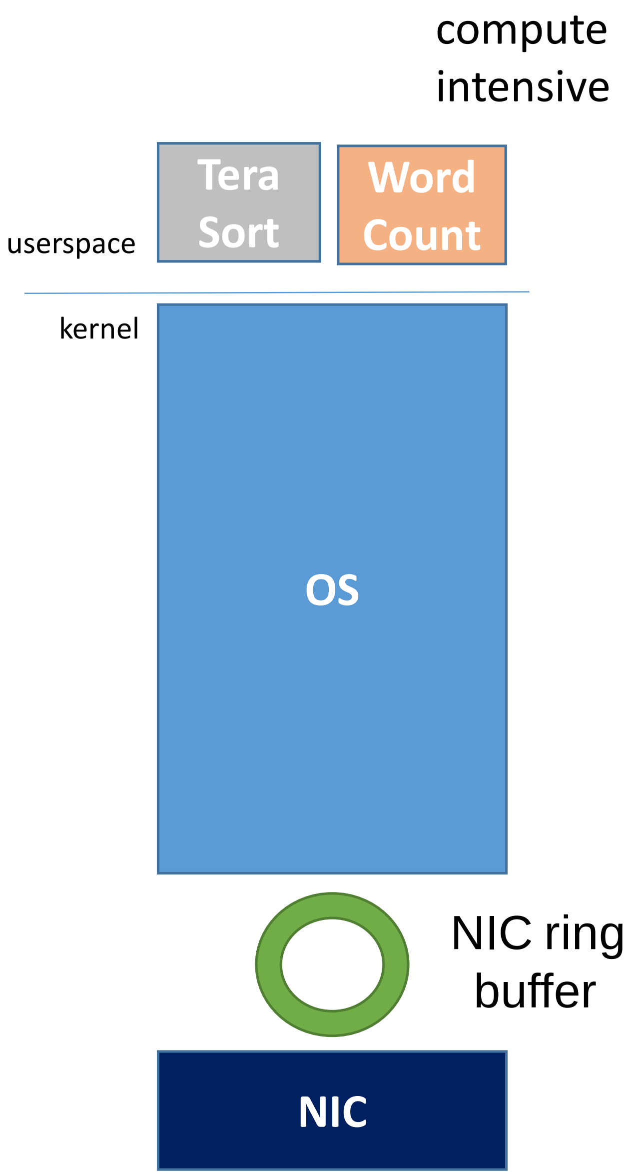


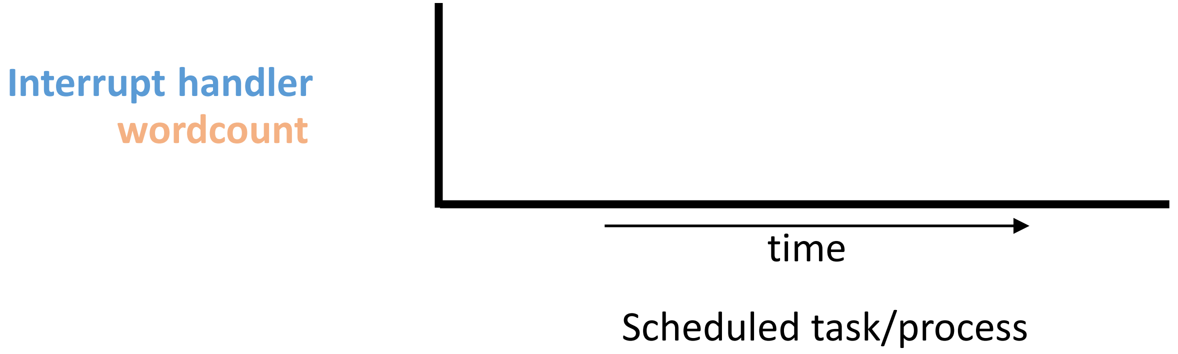
How is Isolation Broken?



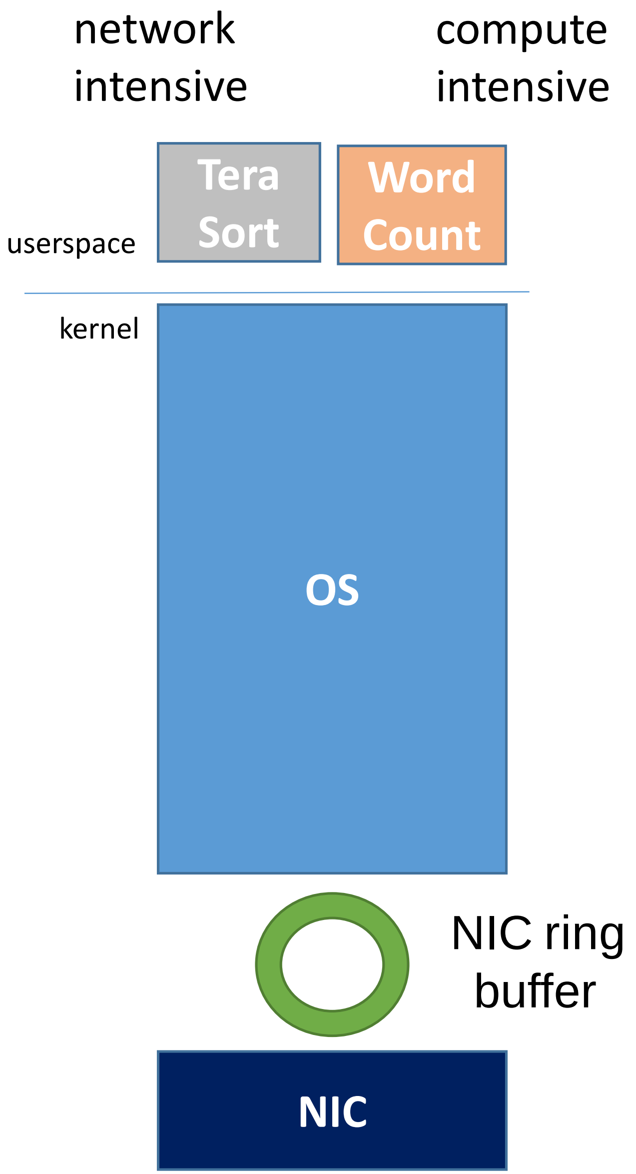


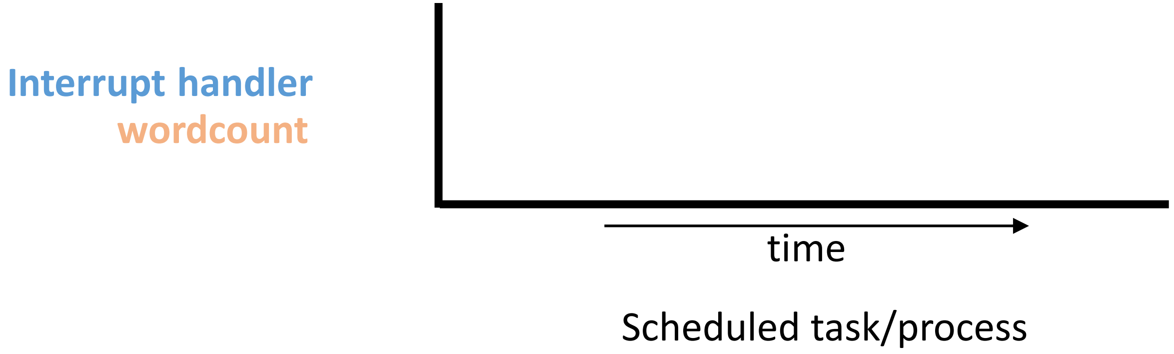
How is Isolation Broken?



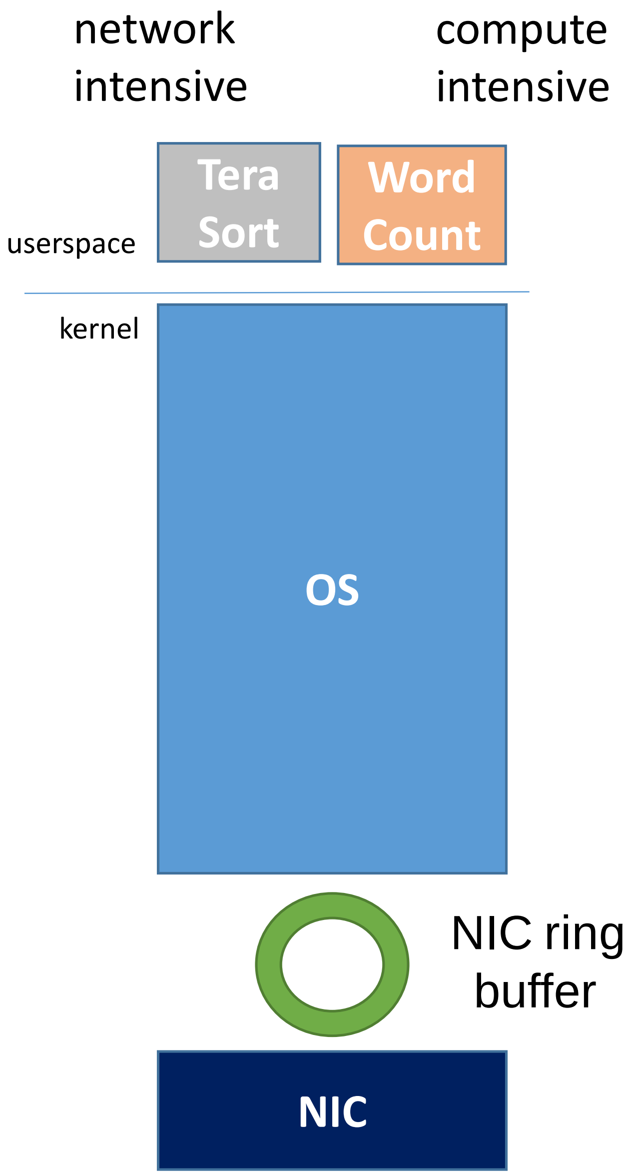


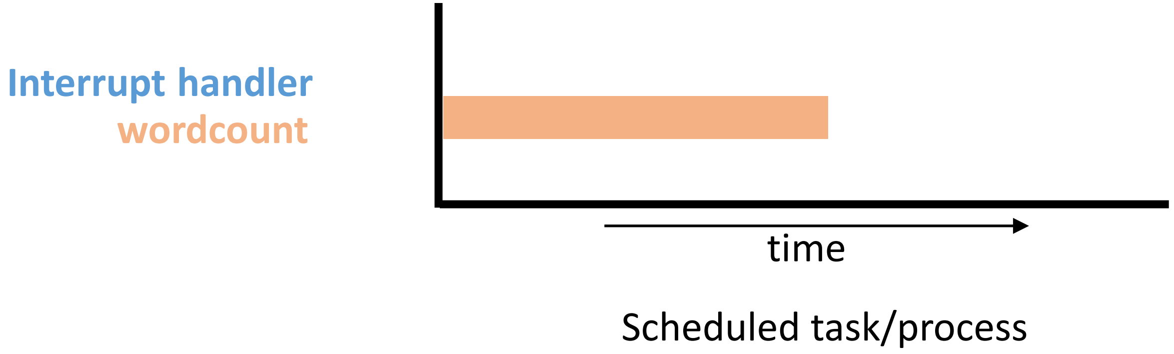
How is Isolation Broken?



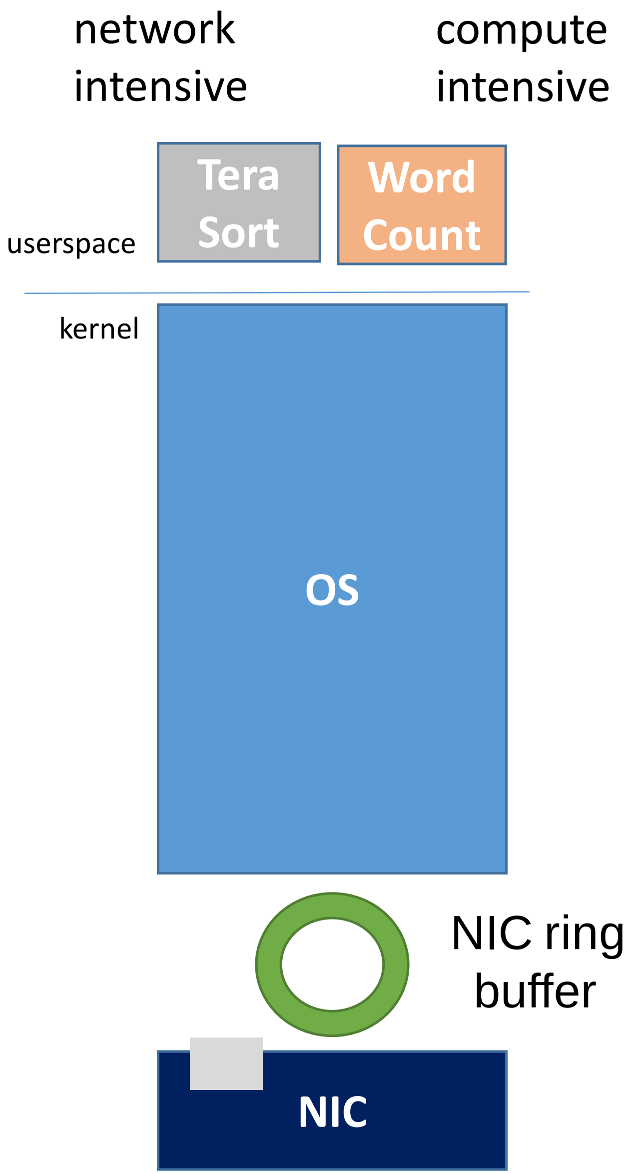


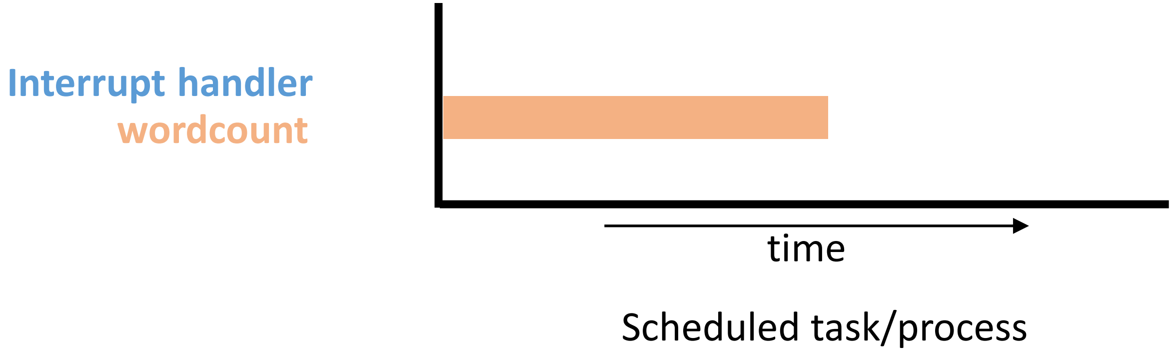
How is Isolation Broken?



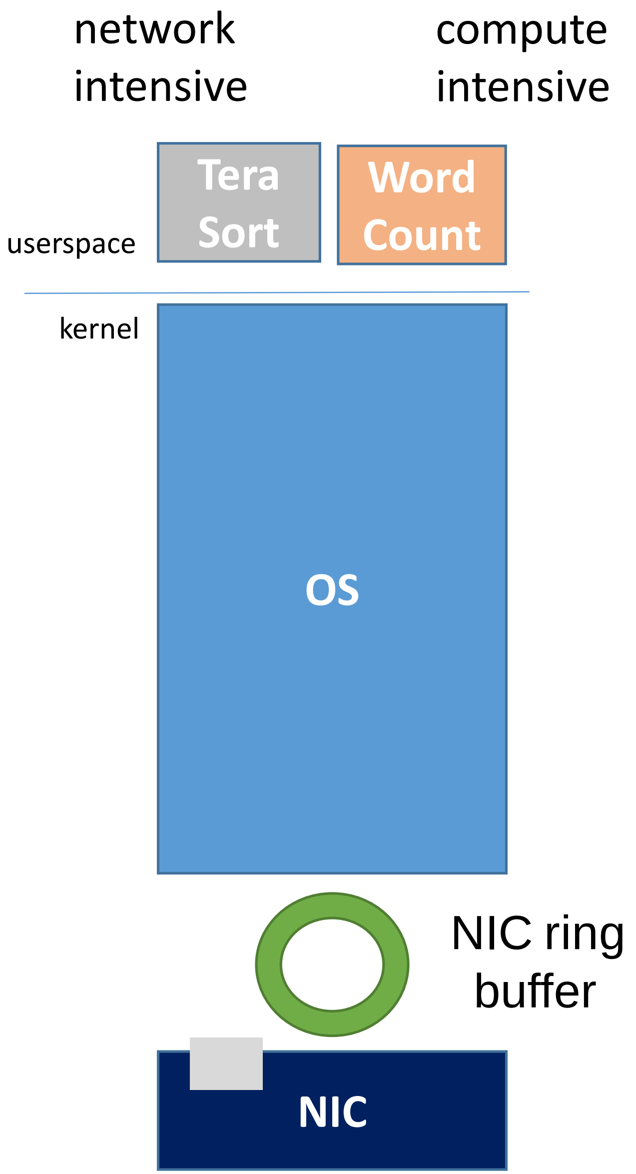


How is Isolation Broken?

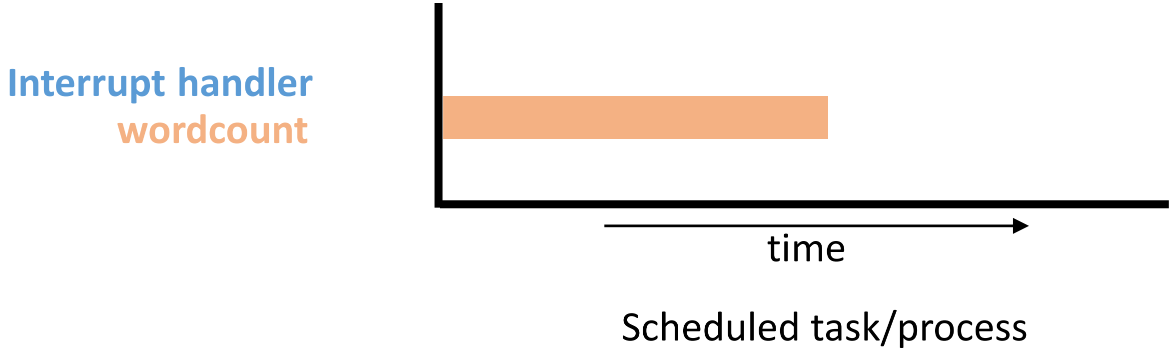




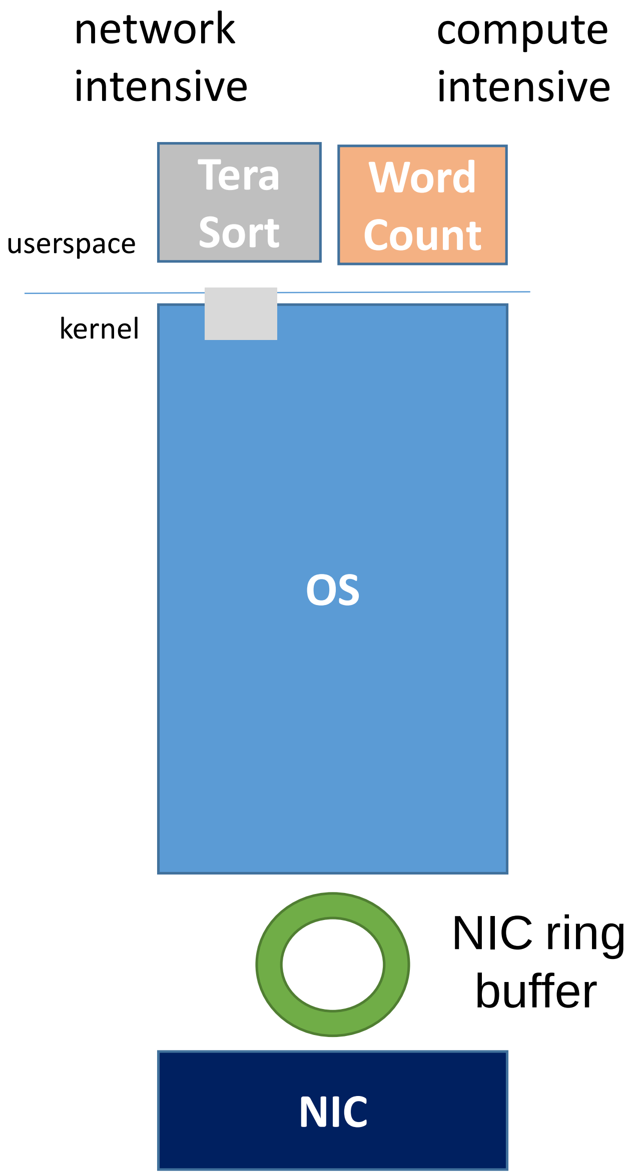
How is Isolation Broken?



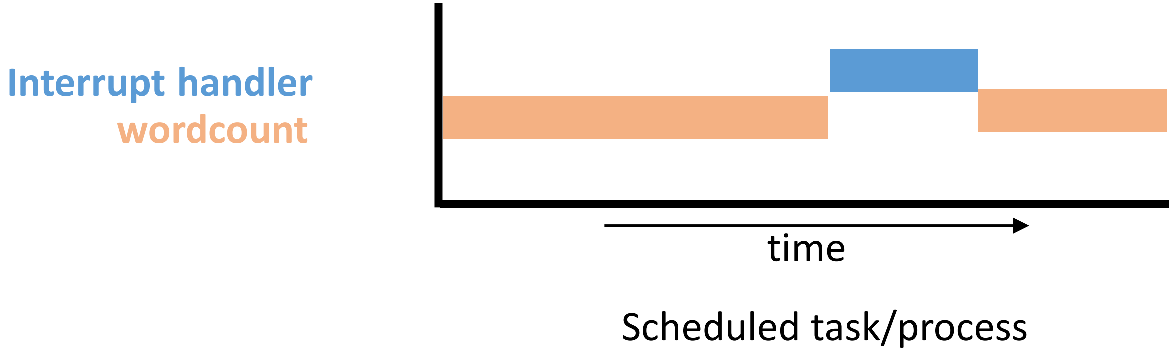
- Kernel processes network traffic via interrupts



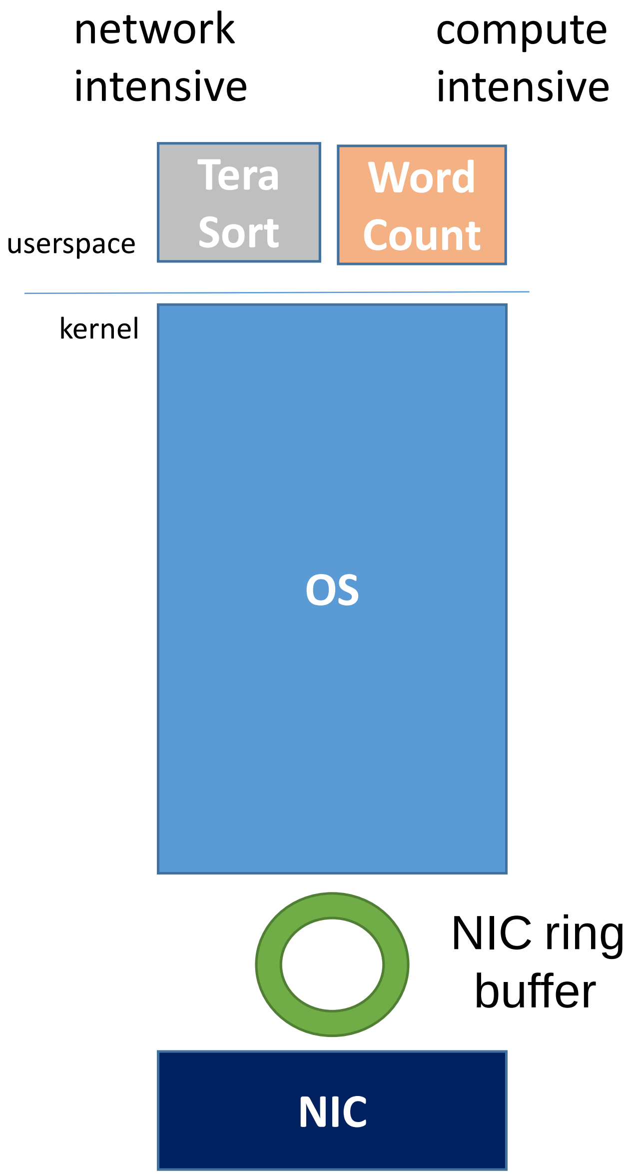
How is Isolation Broken?



- Kernel processes network traffic via interrupts

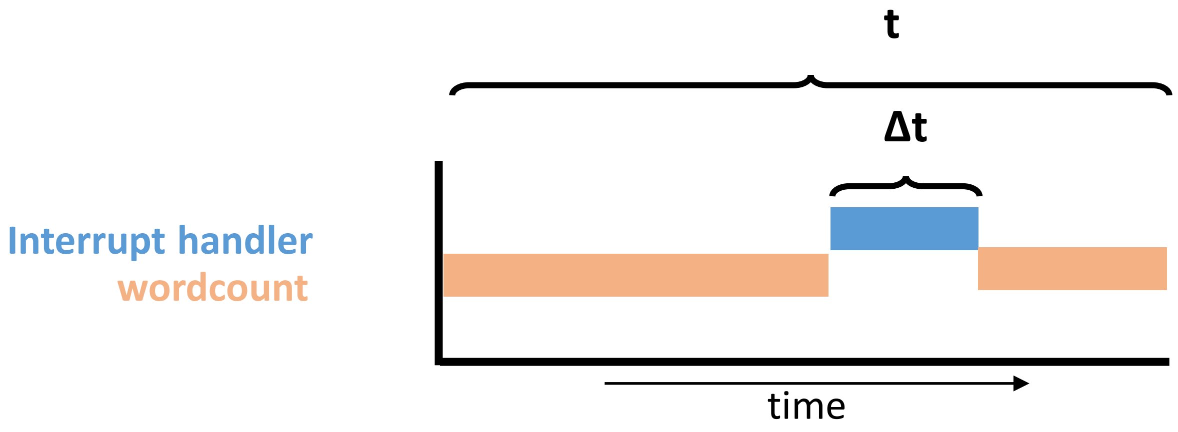


How is Isolation Broken?



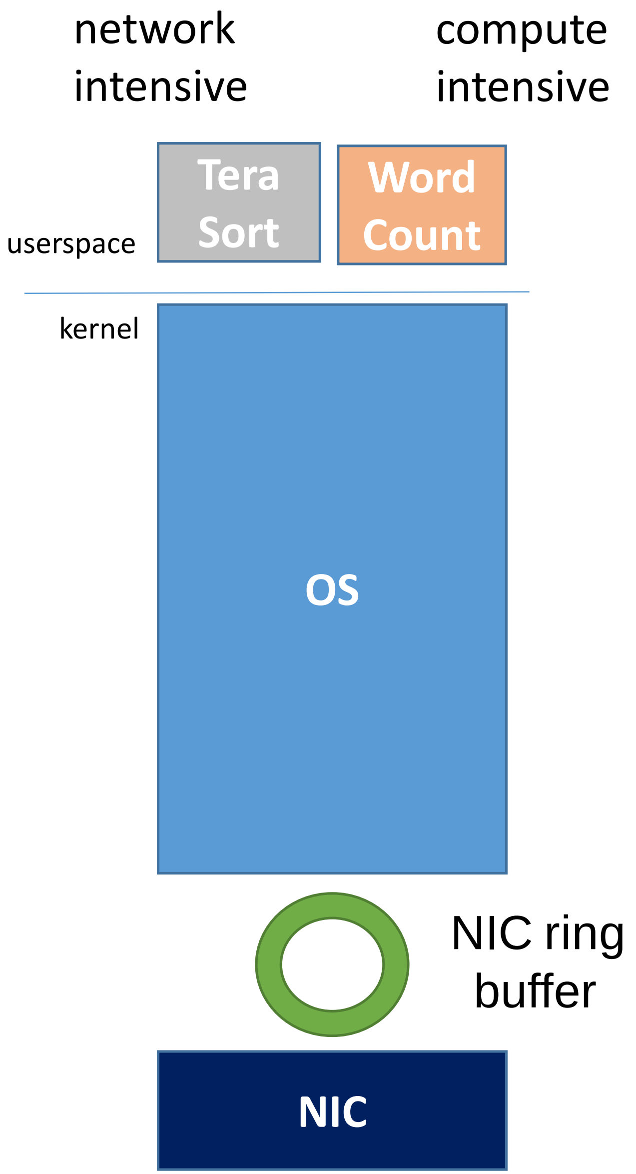
- Kernel processes network traffic via interrupts

Scheduler **charges** wordcount for **"t"**, instead of **"t - Δt"**



Scheduled task/process

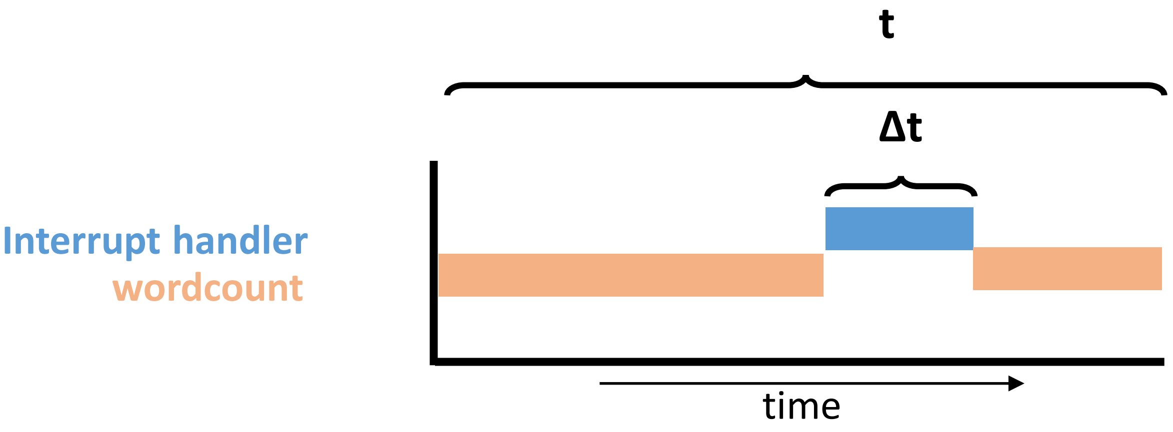
How is Isolation Broken?



Charging: Reduction in the runtime of a container

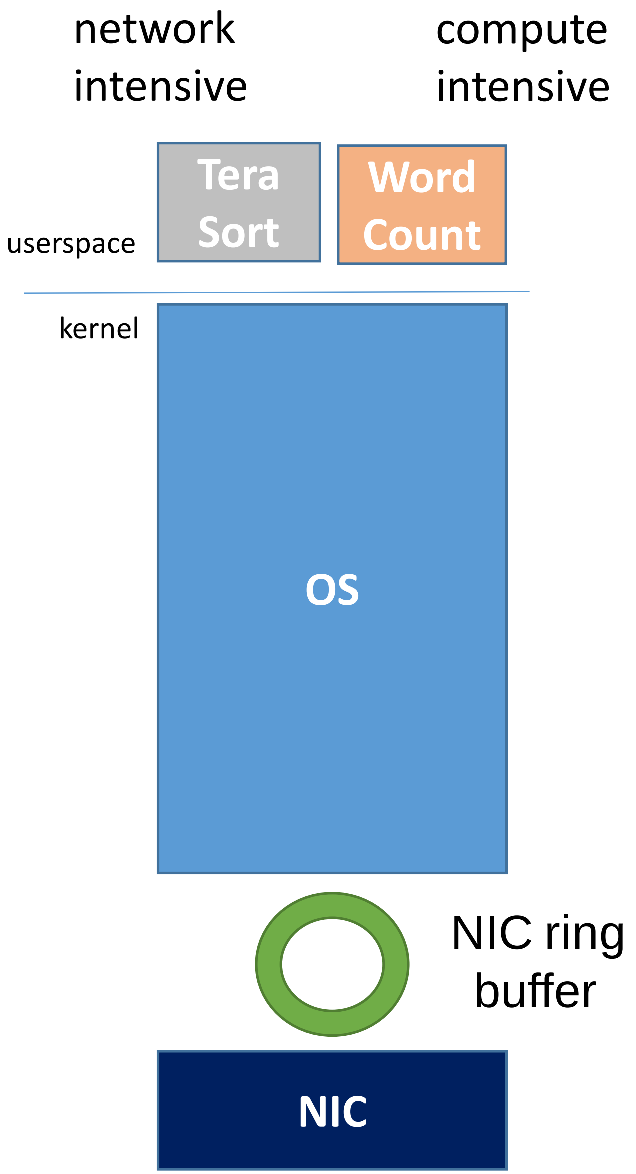
- Kernel processes network traffic via interrupts

Scheduler **charges** wordcount for **“t”**, instead of **“t - Δt”**



Scheduled task/process

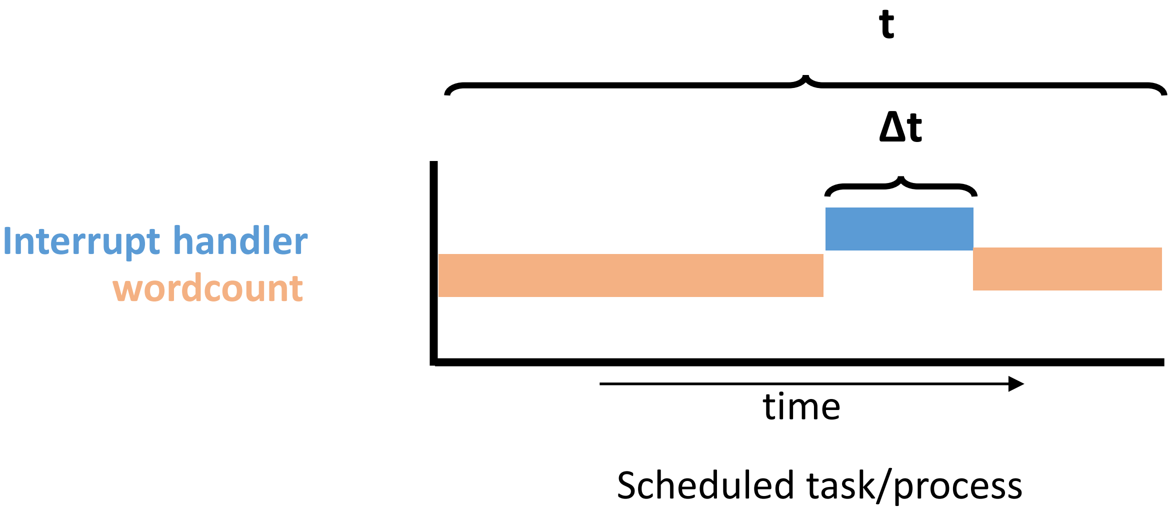
How is Isolation Broken?



Charging: Reduction in the runtime of a container

- Kernel processes network traffic via interrupts
- Time spend in servicing interrupts is incorrectly charged

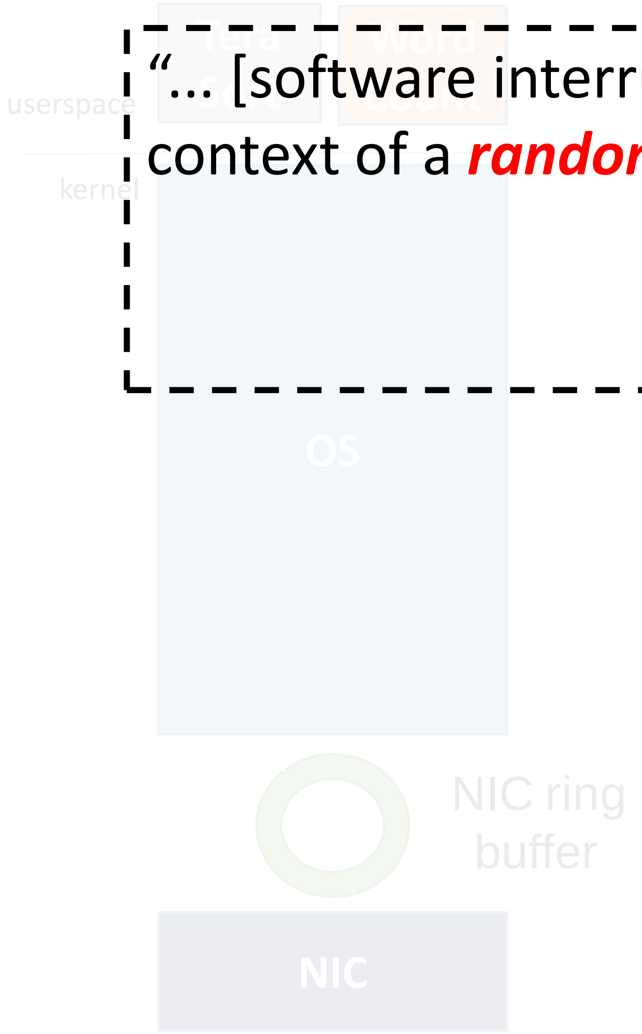
Scheduler **charges** wordcount for **"t"**, instead of **"t - Δt"**



How did Linux get here?

"… [software interrupts] a conglomerate of mostly unrelated jobs, which run in the context of a ***randomly chosen victim*** w/o the ability to put any control on them."

--Thomas Gleixner (Linux developer)

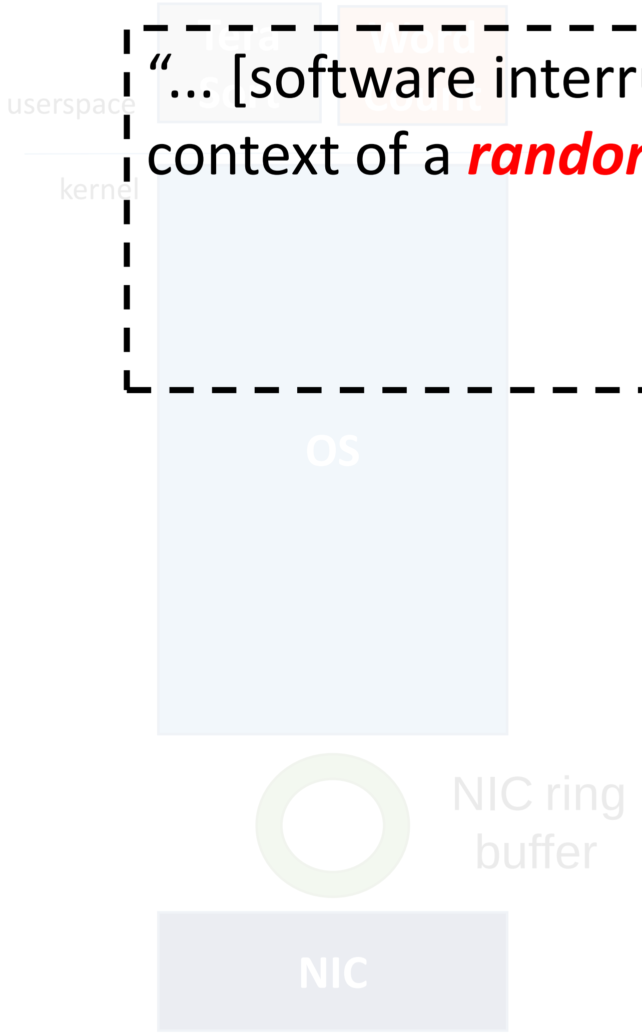


- Kernel processes network traffic via interrupts
- Time spend in servicing interrupts is not accounted properly

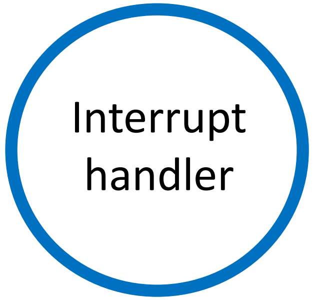
How did Linux get here?

"... [software interrupts] a conglomerate of mostly unrelated jobs, which run in the context of a ***randomly chosen victim*** w/o the ability to put any control on them."

--Thomas Gleixner (Linux developer)



- Kernel processes network traffic via interrupts
- Time spend in servicing interrupts is not accounted properly

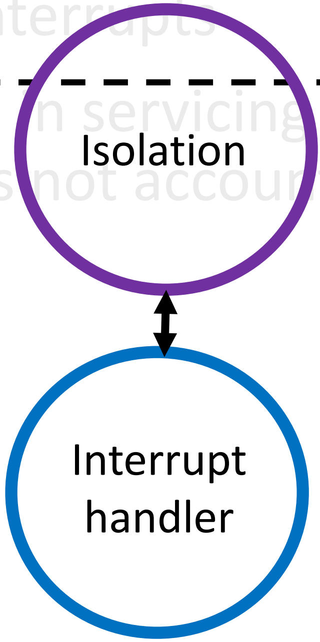


How did Linux get here?

"... [software interrupts] a conglomerate of mostly unrelated jobs, which run in the context of a ***randomly chosen victim*** w/o the ability to put any control on them."

--Thomas Gleixner (Linux developer)

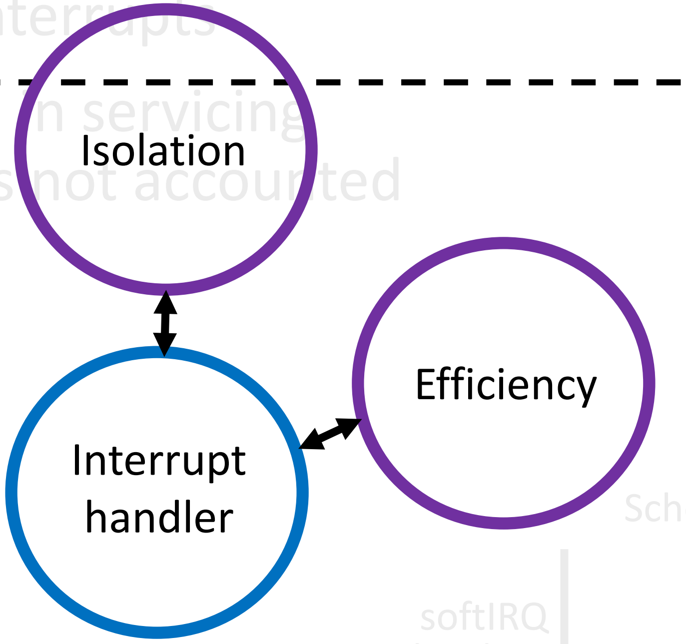
- Kernel processes network traffic via interrupts
- Time spend in servicing interrupts is not accounted properly



How did Linux get here?

"... [software interrupts] a conglomerate of mostly unrelated jobs, which run in the context of a ***randomly chosen victim*** w/o the ability to put any control on them."

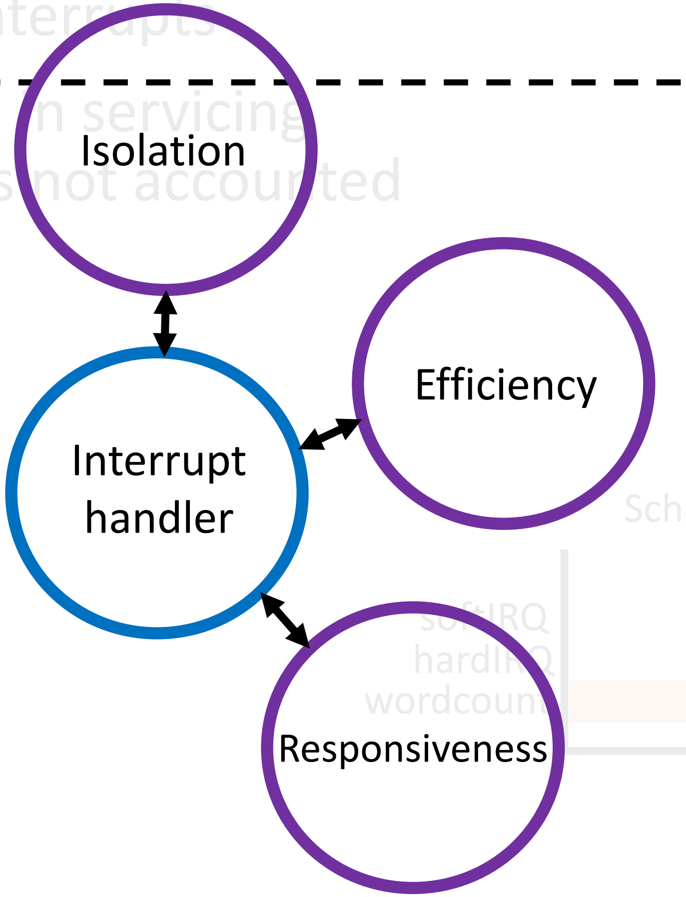
--Thomas Gleixner (Linux developer)



How did Linux get here?

"… [software interrupts] a conglomerate of mostly unrelated jobs, which run in the context of a ***randomly chosen victim*** w/o the ability to put any control on them."

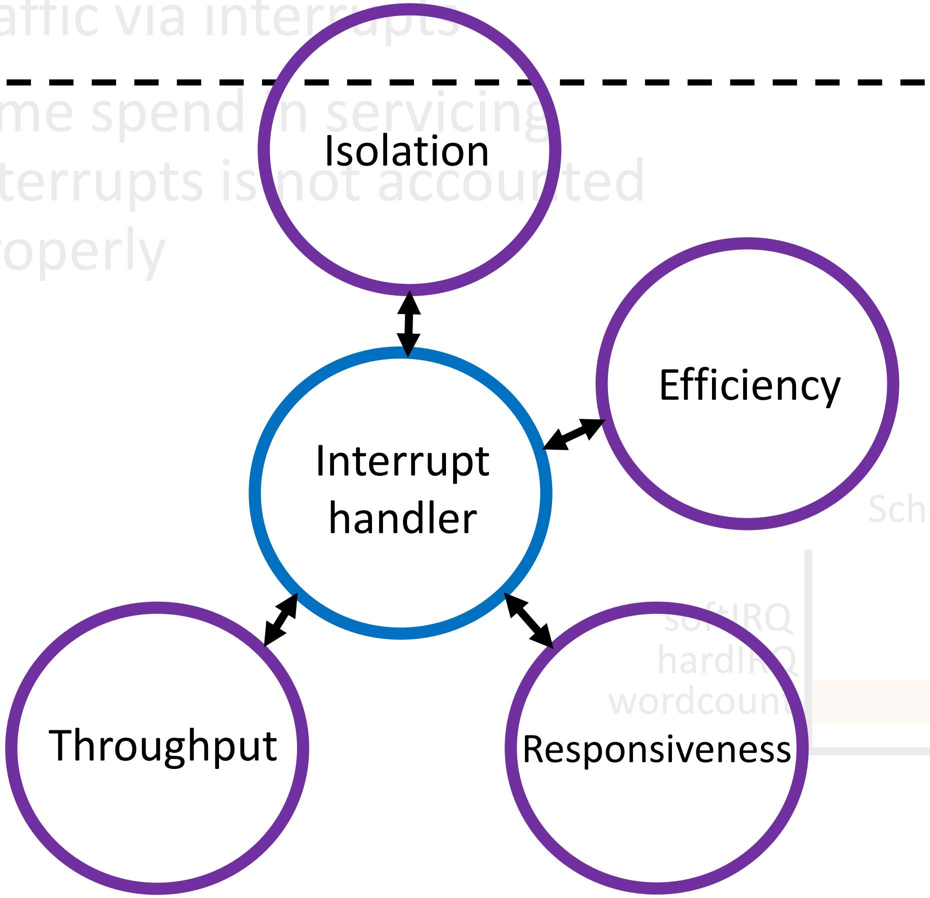
--Thomas Gleixner (Linux developer)



How did Linux get here?

"... [software interrupts] a conglomerate of mostly unrelated jobs, which run in the context of a ***randomly chosen victim*** w/o the ability to put any control on them."

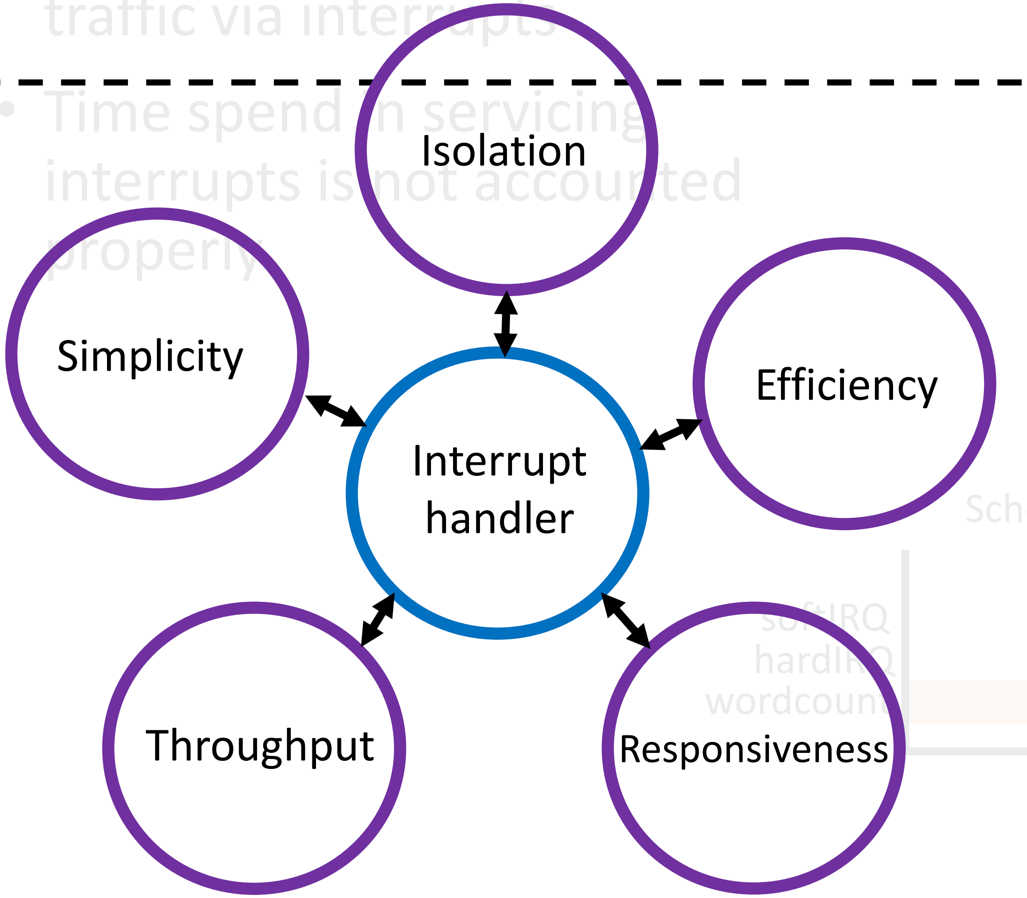
--Thomas Gleixner (Linux developer)



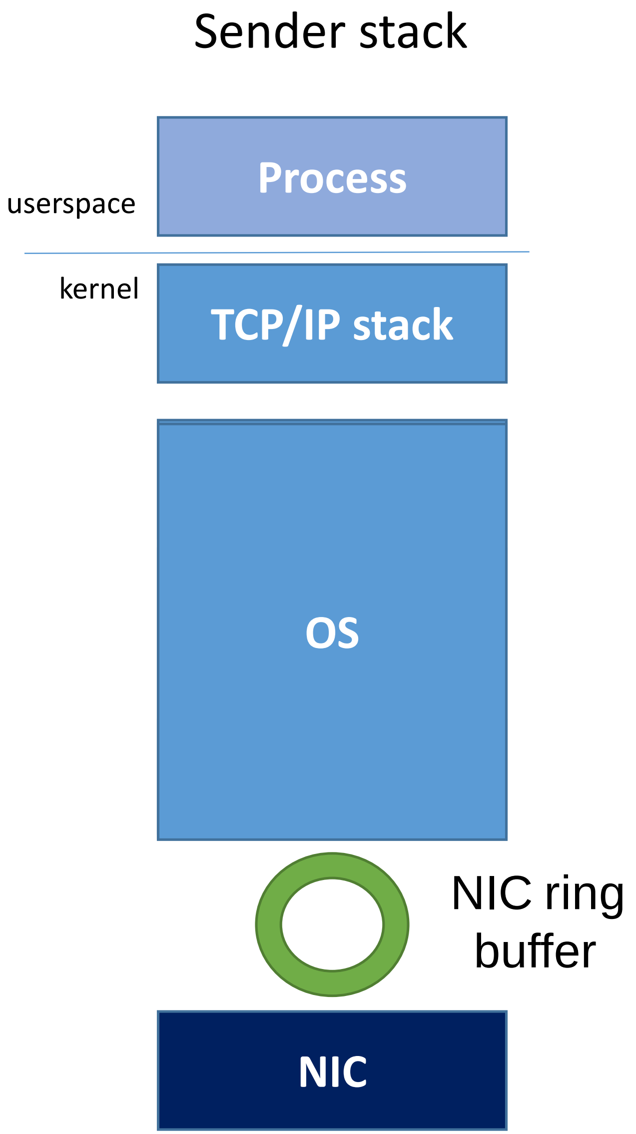
How did Linux get here?

"... [software interrupts] a conglomerate of mostly unrelated jobs, which run in the context of a ***randomly chosen victim*** w/o the ability to put any control on them."

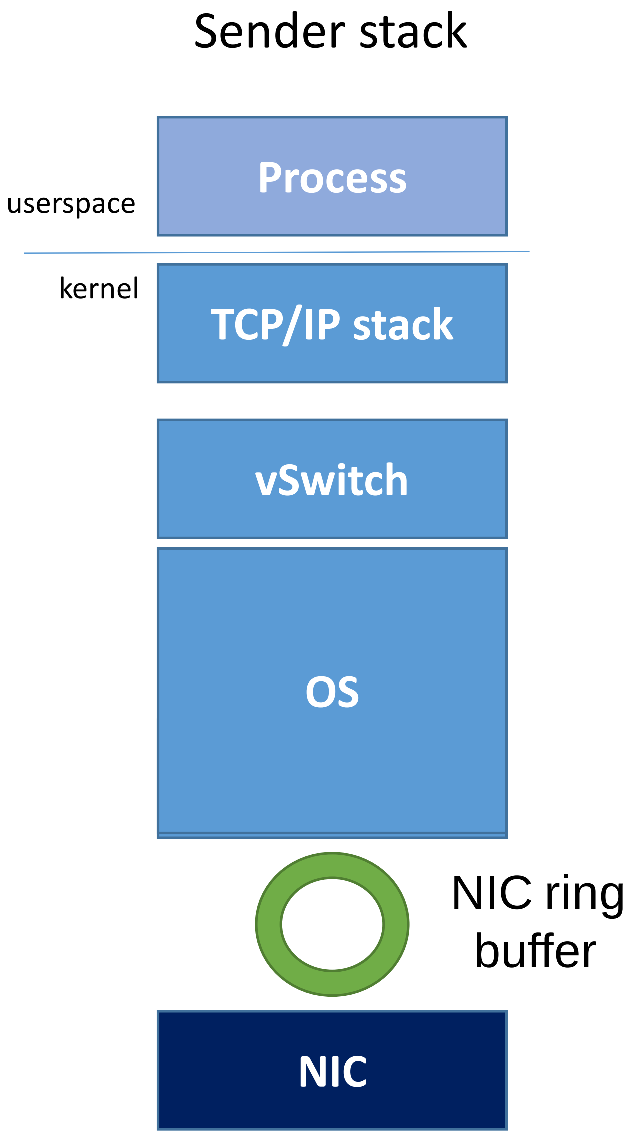
--Thomas Gleixner (Linux developer)



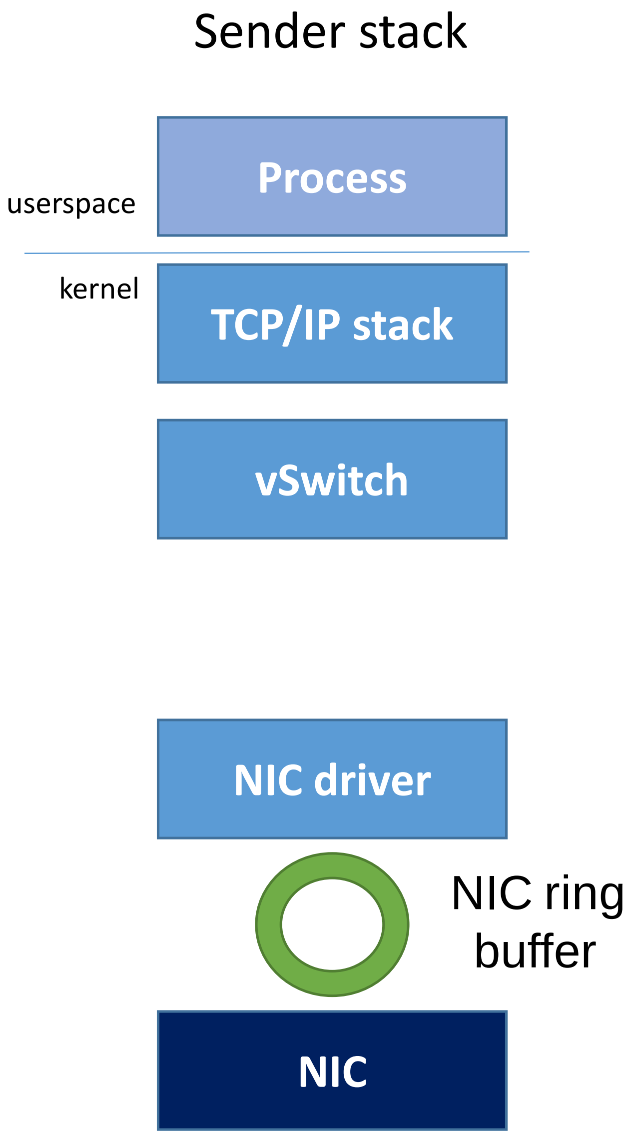
Sender Side


Sender stack
userspace
Process
kernel
TCP/IP stack
OS
NIC ring buffer
NIC

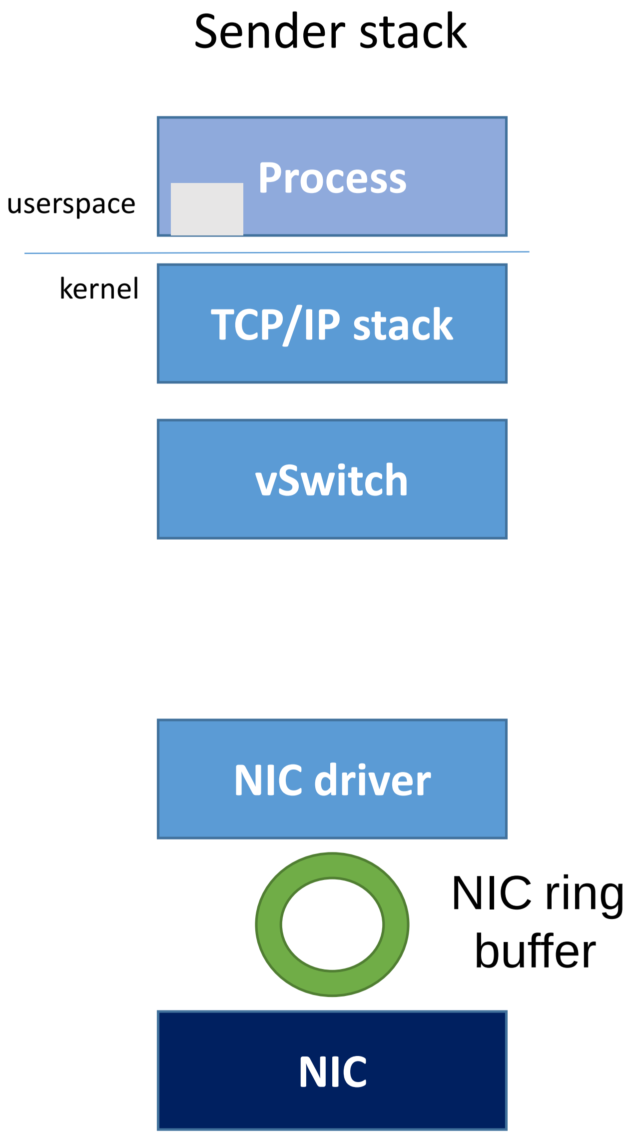
Sender Side


Sender stack
userspace
Process
kernel
TCP/IP stack
vSwitch
OS
NIC ring
buffer
NIC

Sender Side

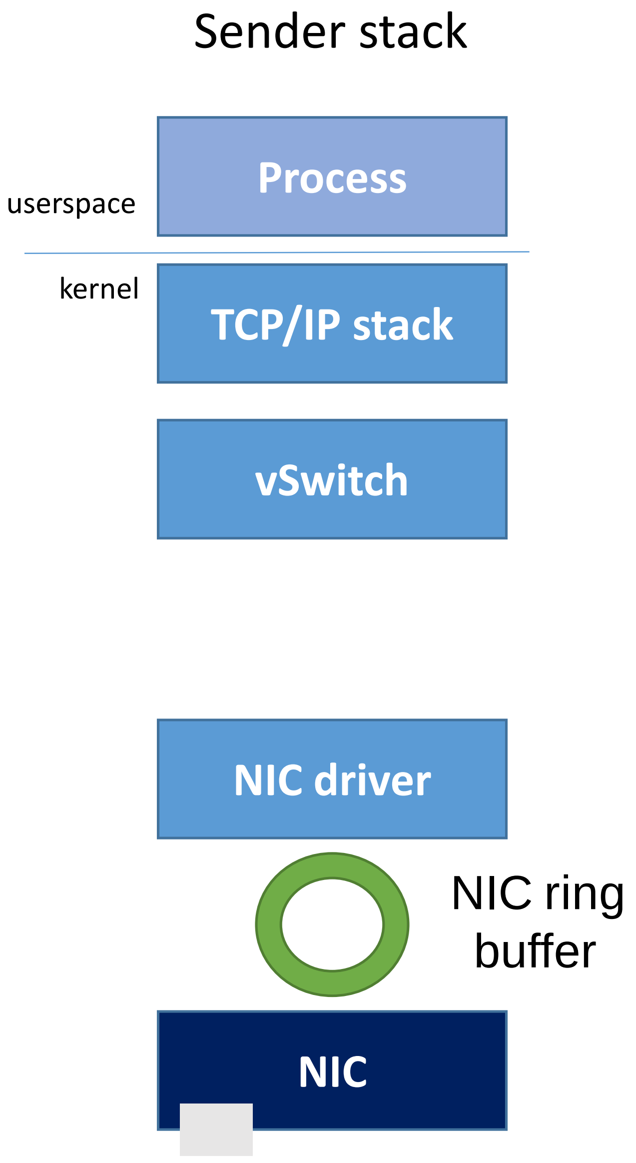

Sender stack
userspace
Process
kernel
TCP/IP stack
vSwitch
NIC driver
NIC ring
buffer
NIC

Sender Side

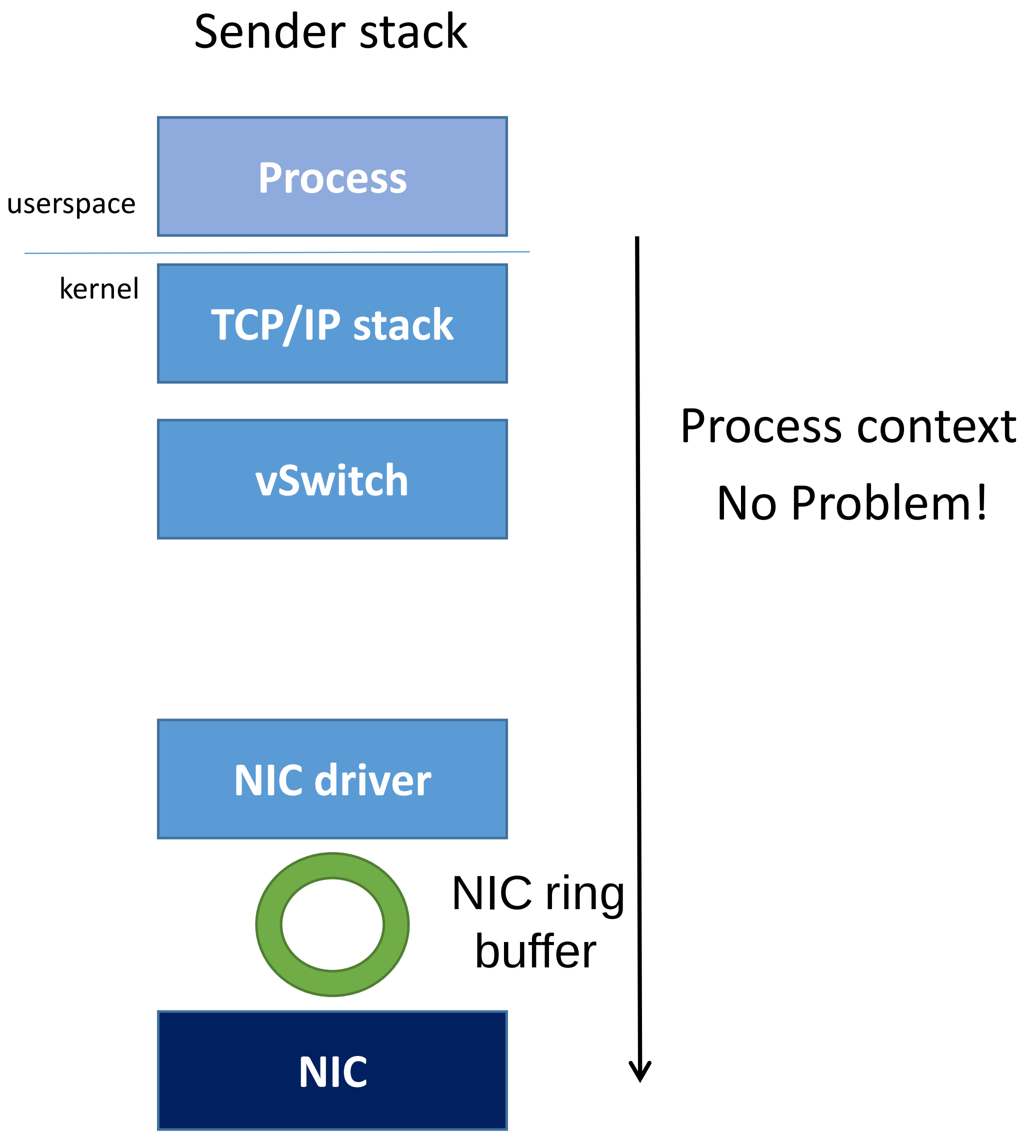


Sender stack
userspace
Process
kernel
TCP/IP stack
vSwitch
NIC driver
NIC ring buffer
NIC

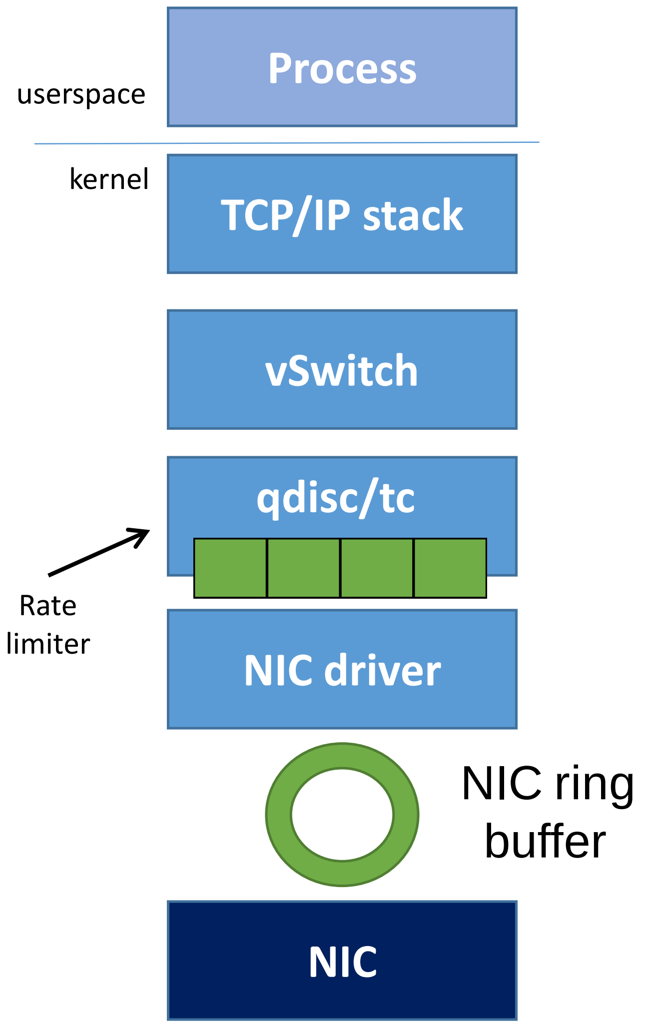
Sender Side


Sender stack
userspace
Process
kernel
TCP/IP stack
vSwitch
NIC driver
NIC ring buffer
NIC

Sender Side

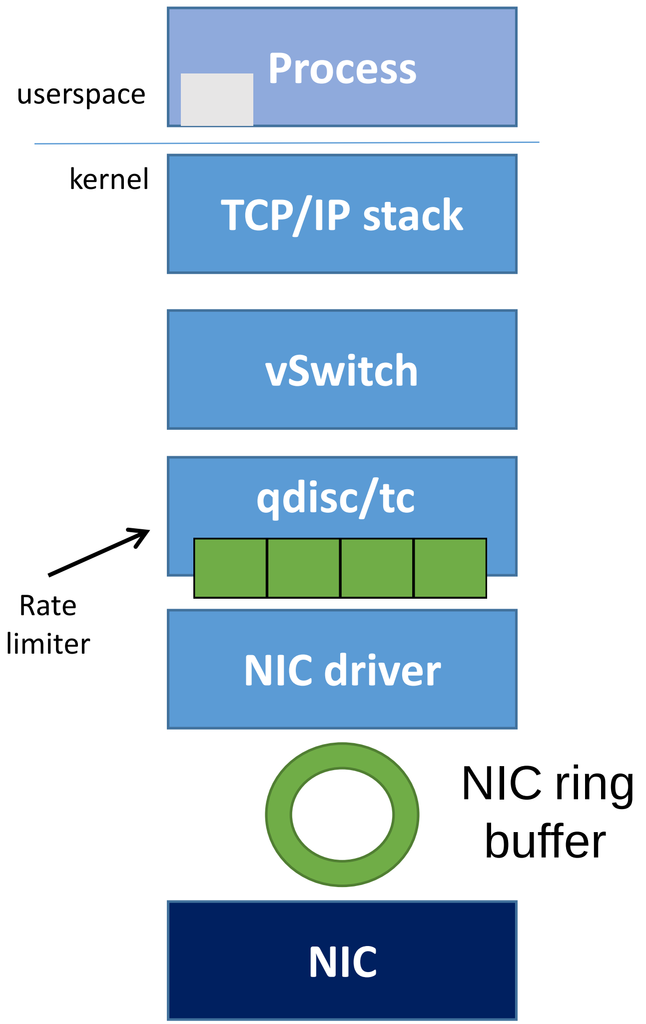

Sender stack
userspace
Process
kernel
TCP/IP stack
vSwitch
NIC driver
NIC ring buffer
NIC
Process context
No Problem!

Sender Side

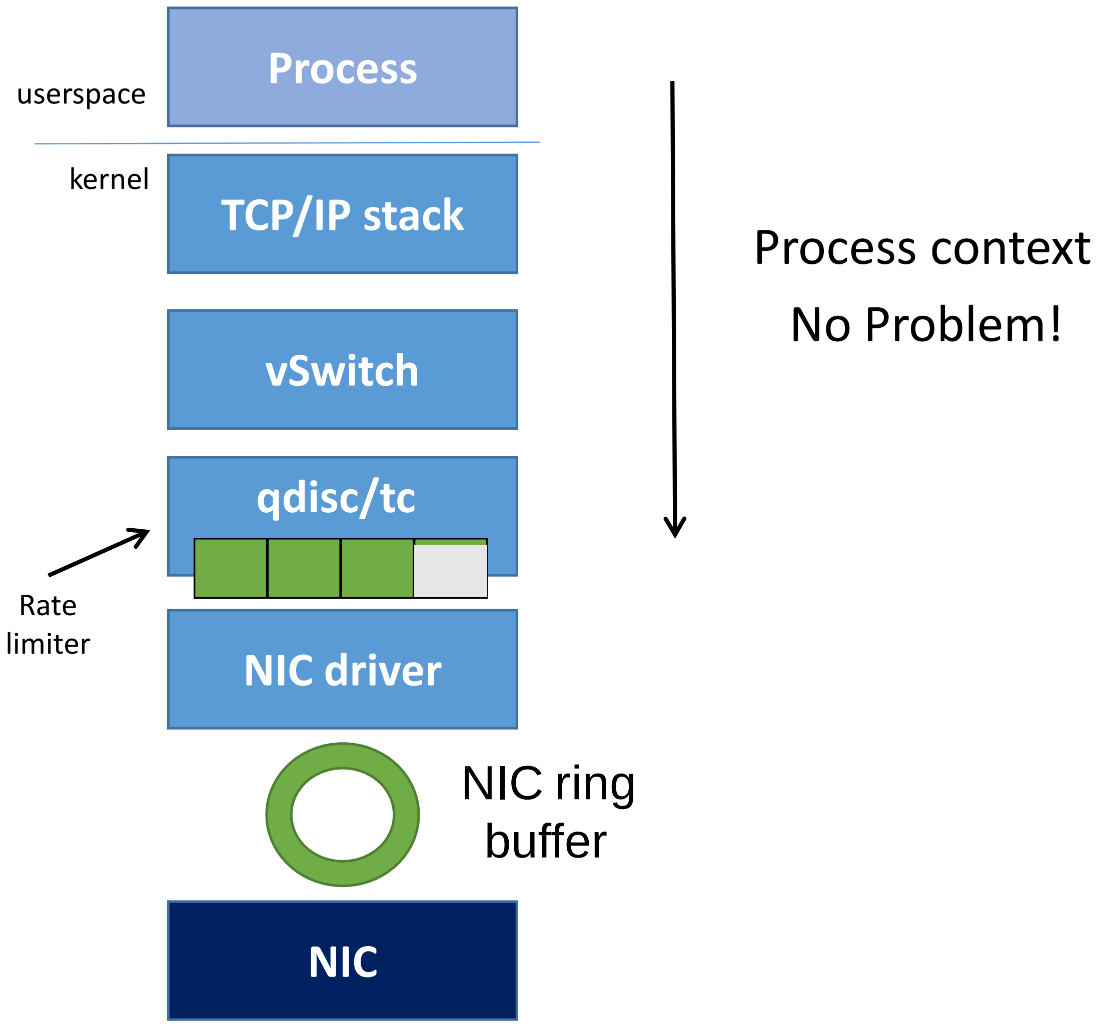


userspace
Process
kernel
TCP/IP stack
vSwitch
qdisc/tc
Rate
limiter
NIC driver
NIC ring
buffer
NIC

Sender Side

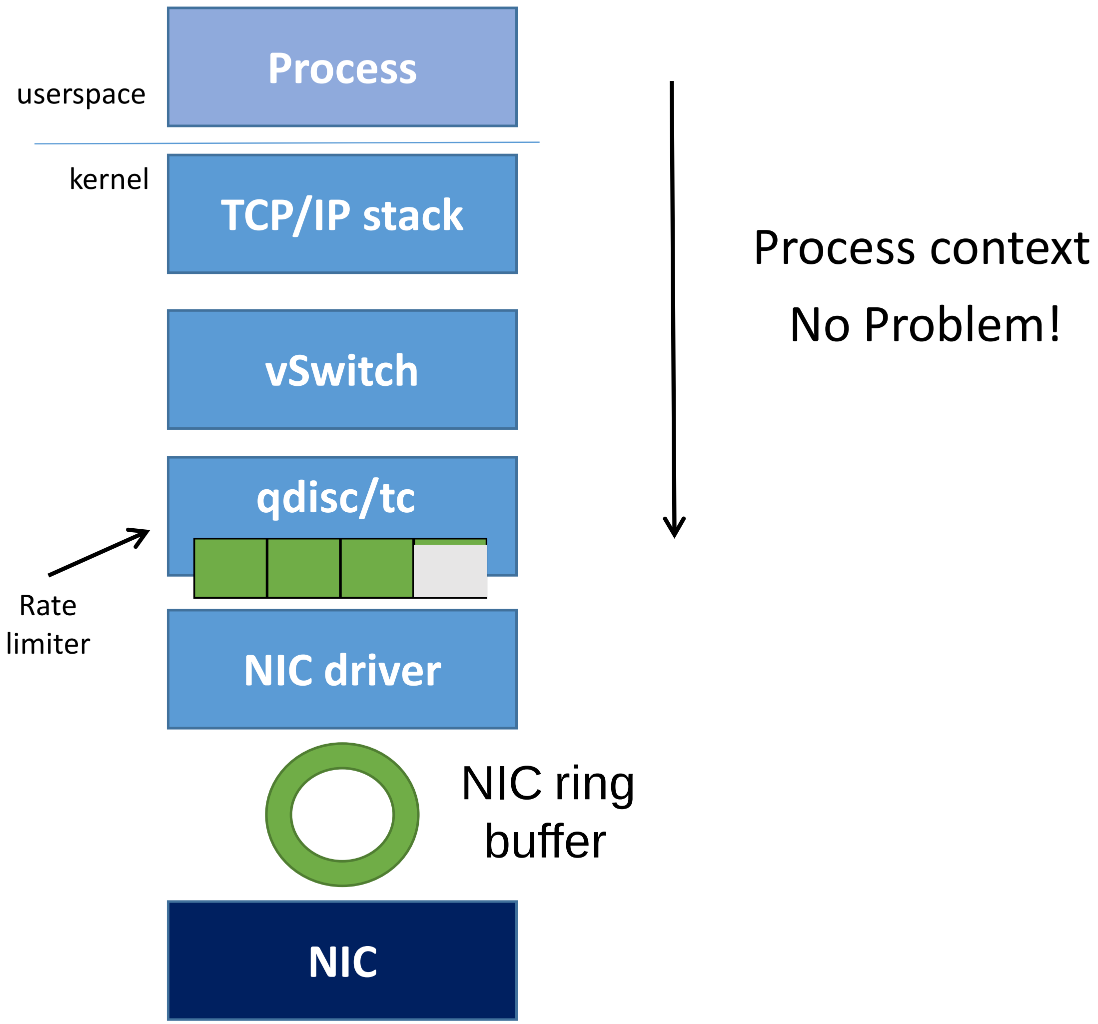

userspace
Process
kernel
TCP/IP stack
vSwitch
qdisc/tc
Rate
limiter
NIC driver
NIC ring
buffer
NIC

Sender Side



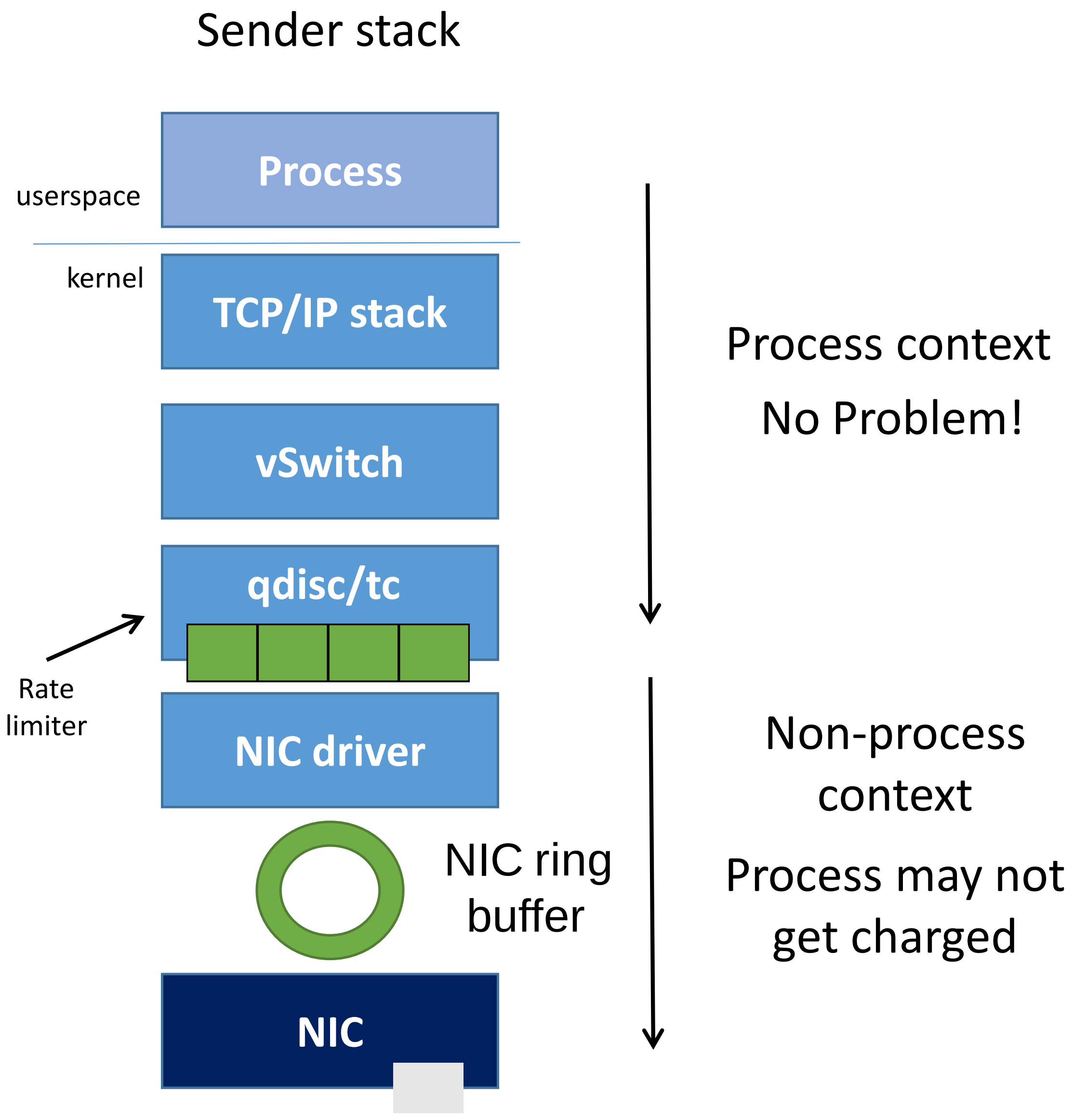
- Packet is enqueued in the process context

Sender Side



- Packet is enqueued in the process context
- System call exits after enqueuing the packet

Sender Side

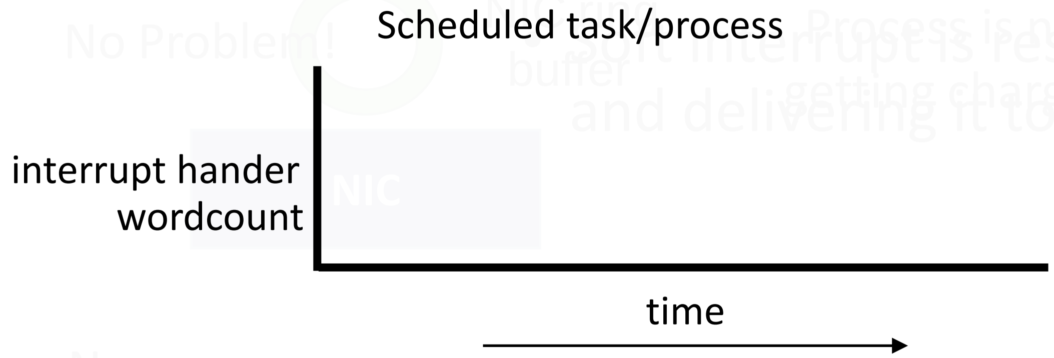


- Packet is enqueued in the process context
- System call exits after enqueuing the packet
- Soft interrupt is responsible for dequeuing and delivering it to the NIC

Sender Side

Linux services a softirq

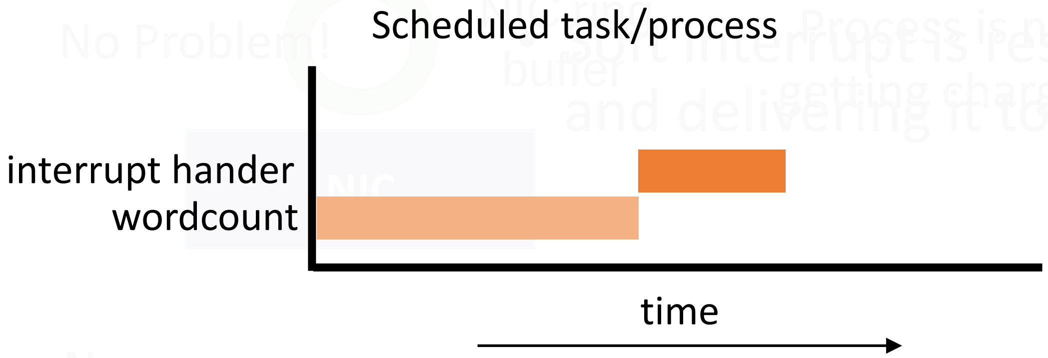
1) at the end of hardware interrupt processing, in the context of the ***currently scheduled process***



Sender Side

Linux services a softirq

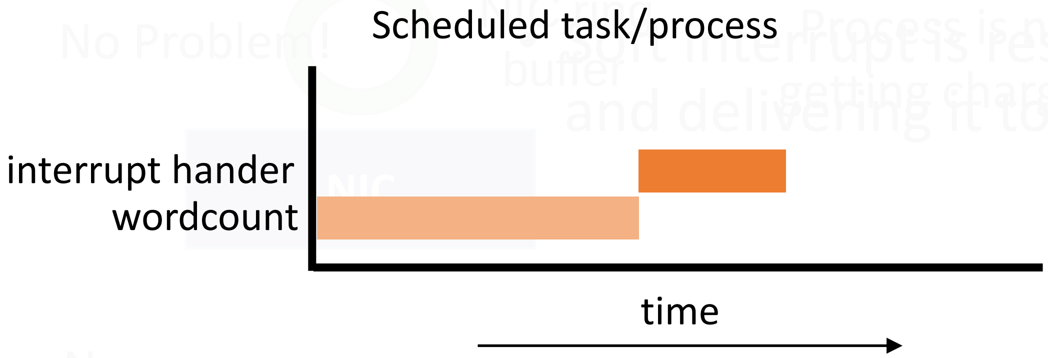
1) at the end of hardware interrupt processing, in the context of the ***currently scheduled process***



Sender Side

Linux services a softirq

1) at the end of hardware interrupt processing, in the context of the ***currently scheduled process***

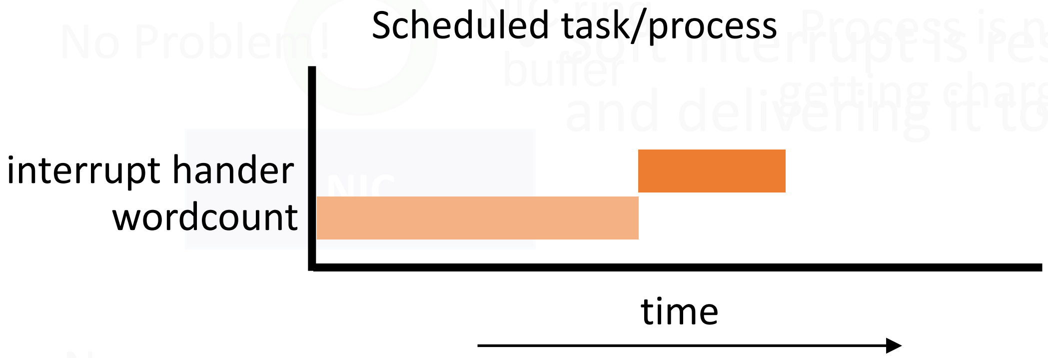


2) through ***ksoftirqd thread*** (a per core kernel thread)

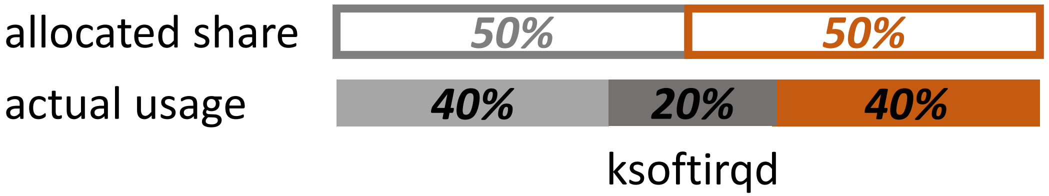
Sender Side

Linux services a softirq

1) at the end of hardware interrupt processing, in the context of the ***currently scheduled process***



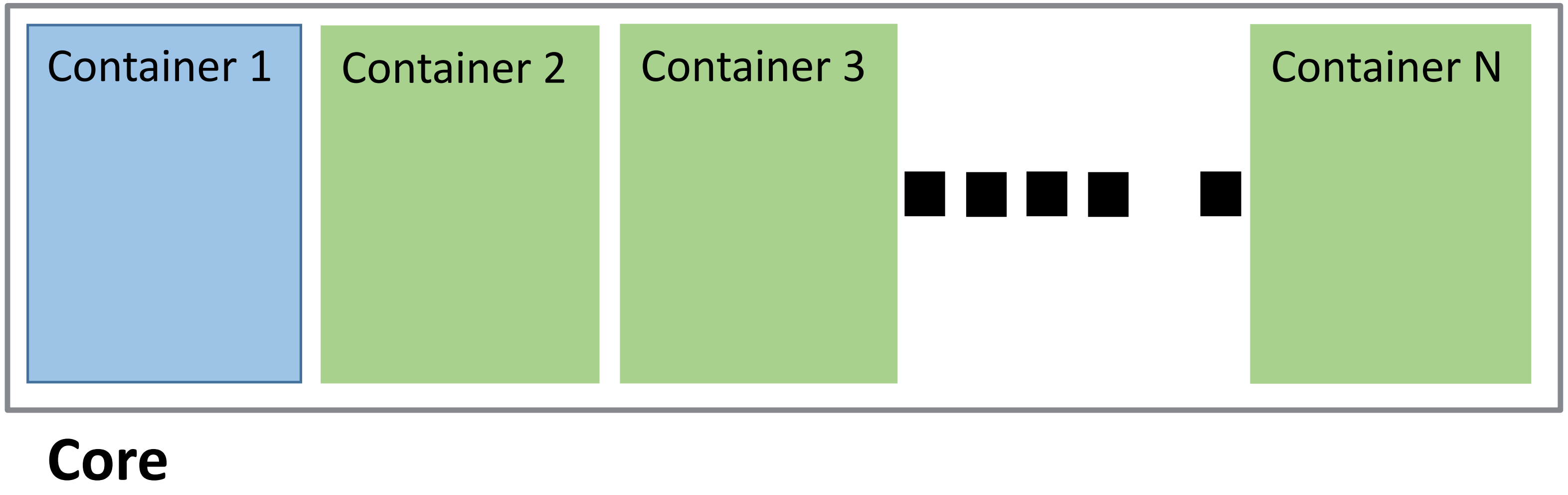
2) through ***ksoftirqd thread*** (a per core kernel thread)



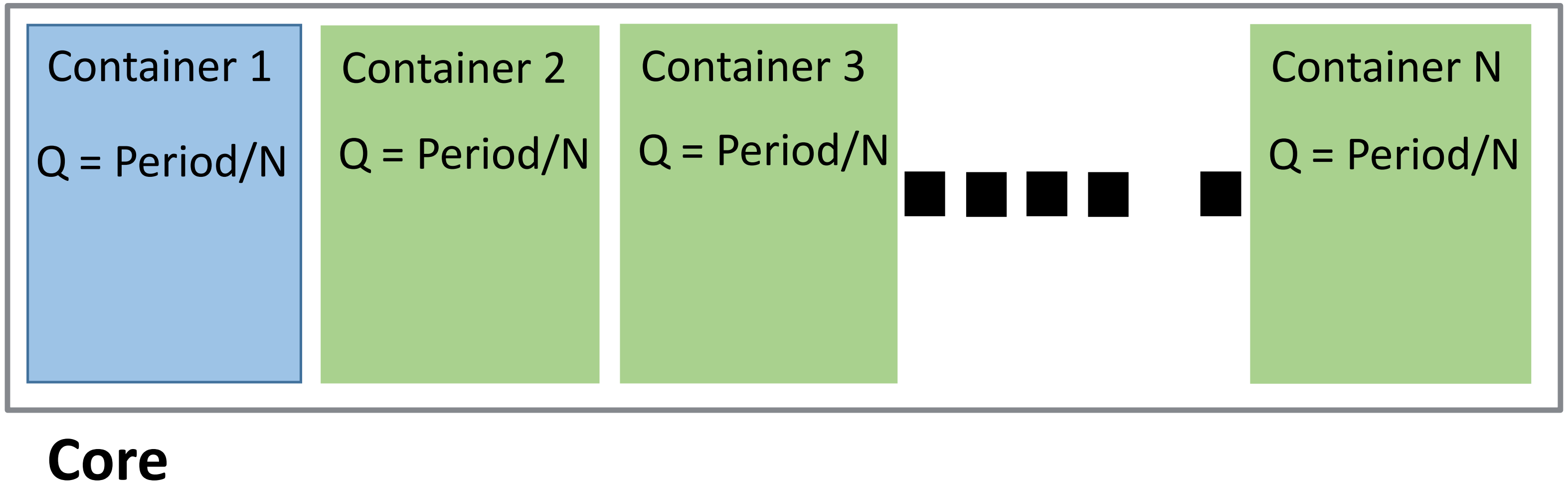
Sender Side

Softirq processing can be ***charged incorrectly*** or ***not charged*** at all to any container

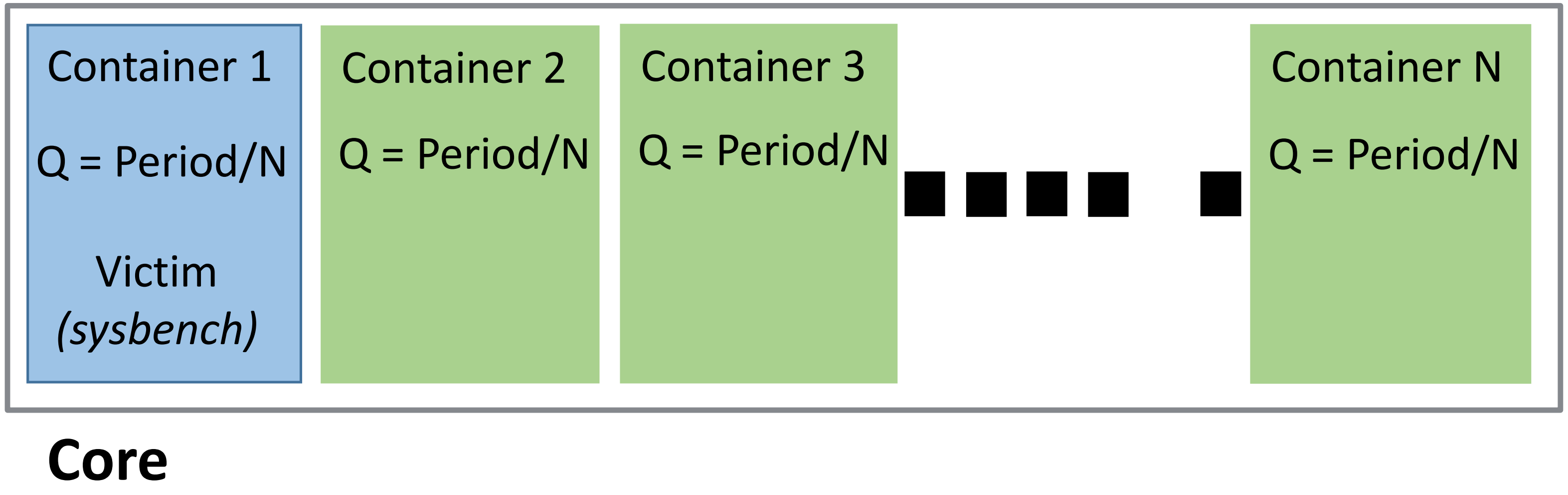
Experiment Setup



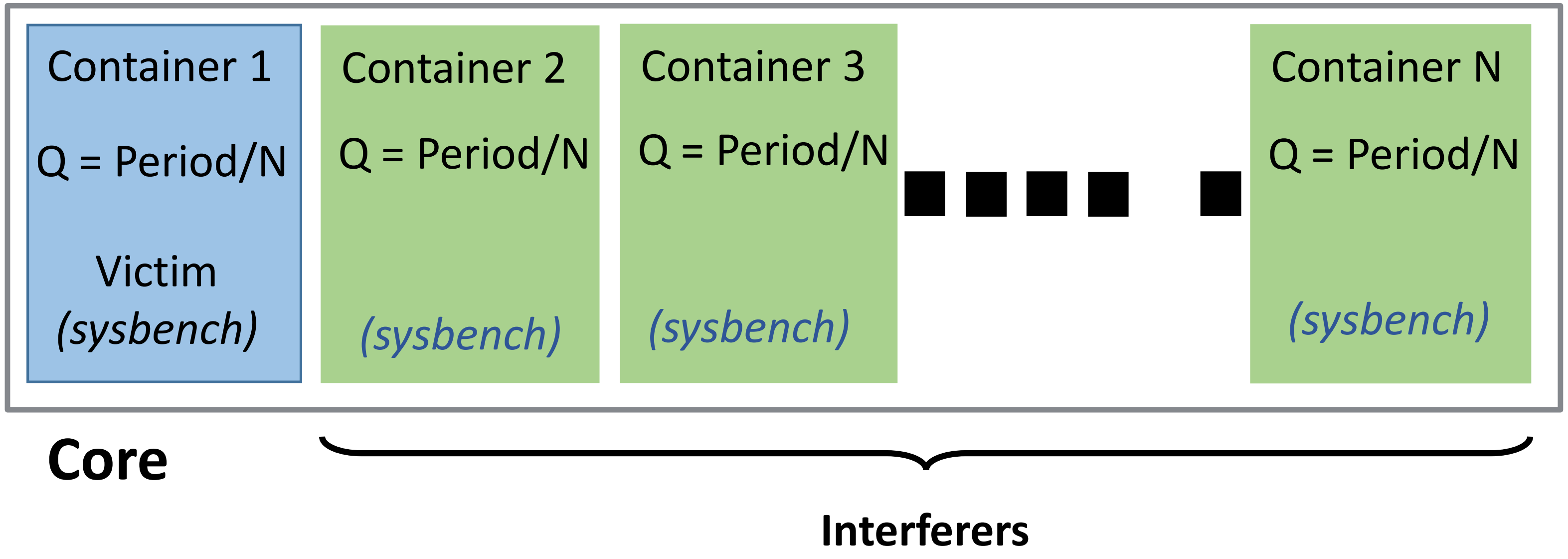
Experiment Setup



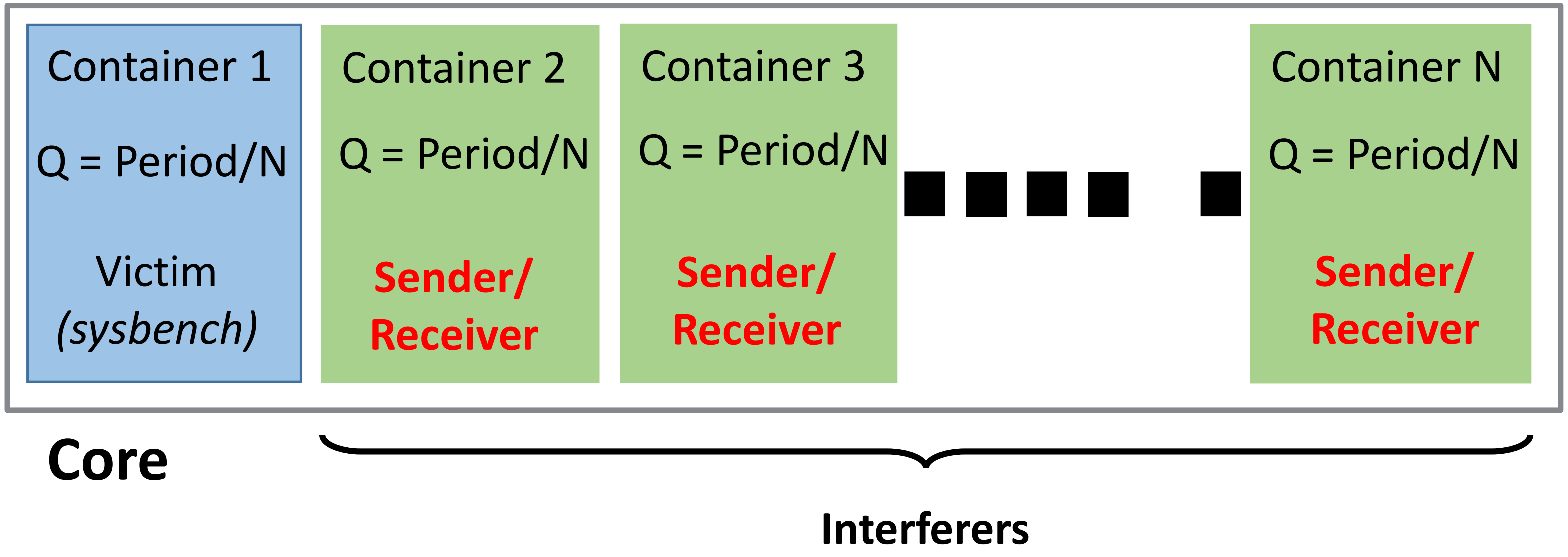
Experiment Setup



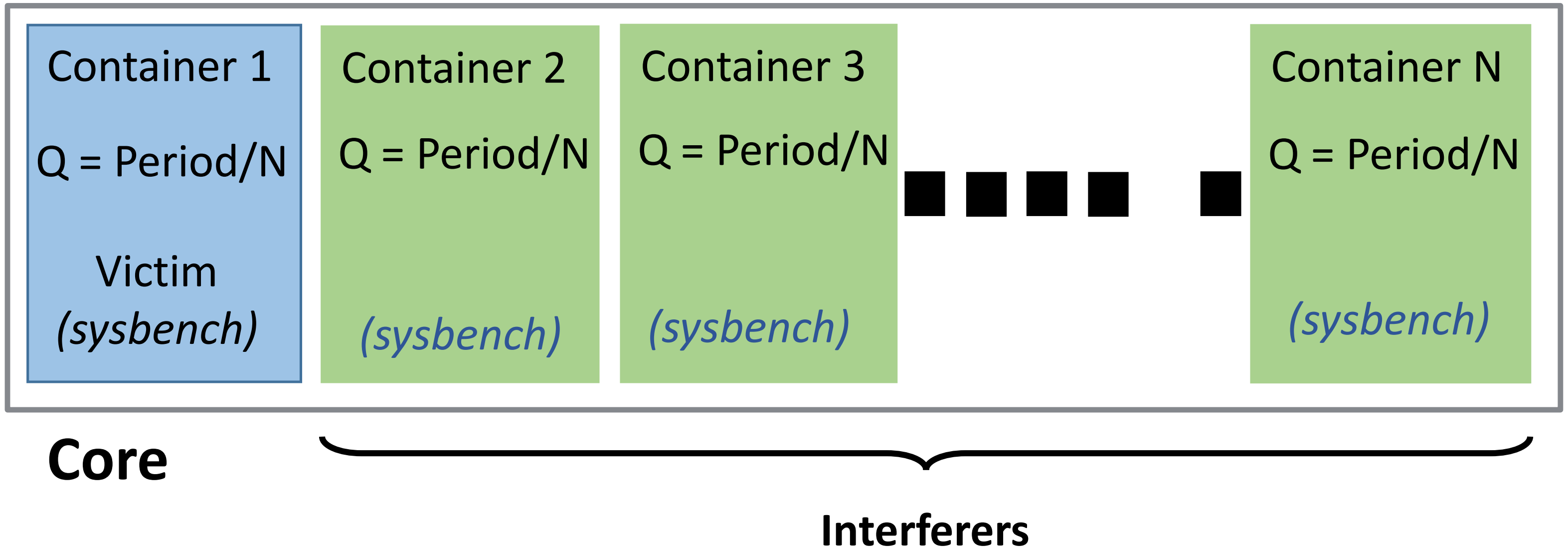
Experiment Setup



Experiment Setup



$$\textit{Penalty factor} = \frac{\textit{Time that victim takes when competing with traffic}}{\textit{Time that victim takes when competing with sysbench}}$$

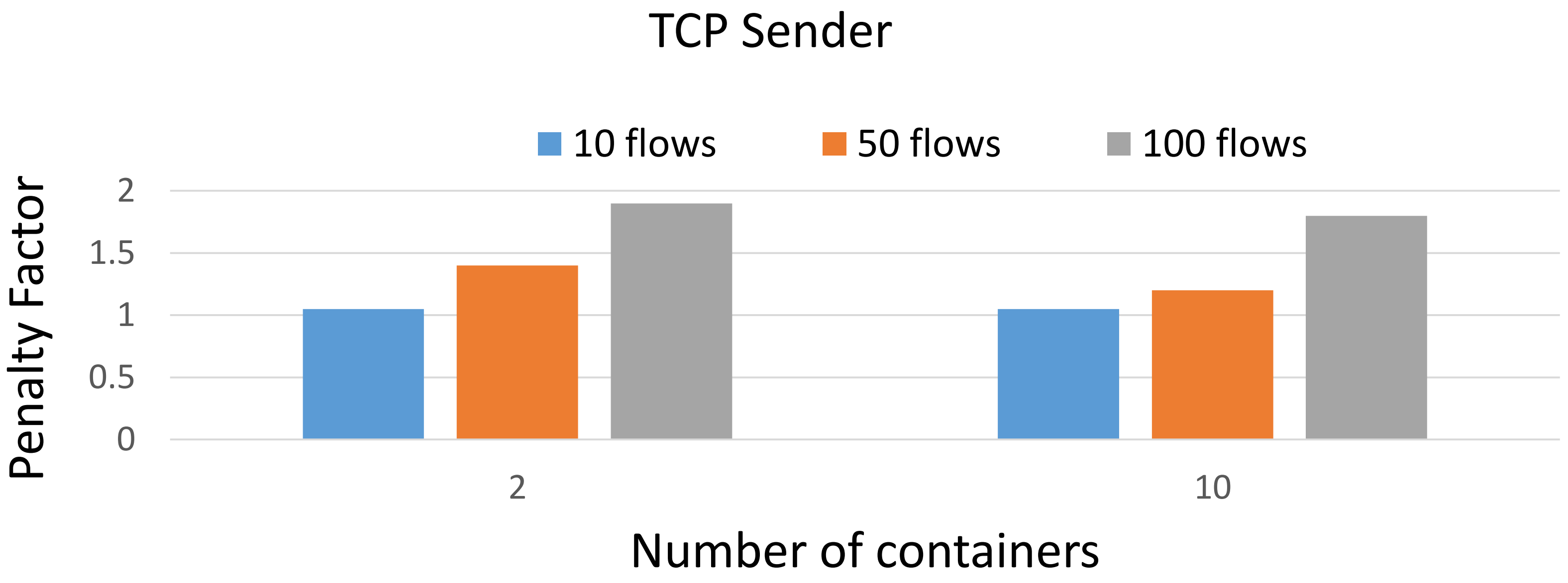


Impact Of Network Traffic

$$\textit{Penalty factor} = \frac{\textit{Time that victim takes when competing with traffic}}{\textit{Time that victim takes when competing with sysbench}}$$

Impact Of Network Traffic

Penalty factor = ***Time that victim takes when competing with traffic*** / ***Time that victim takes when competing with sysbench***

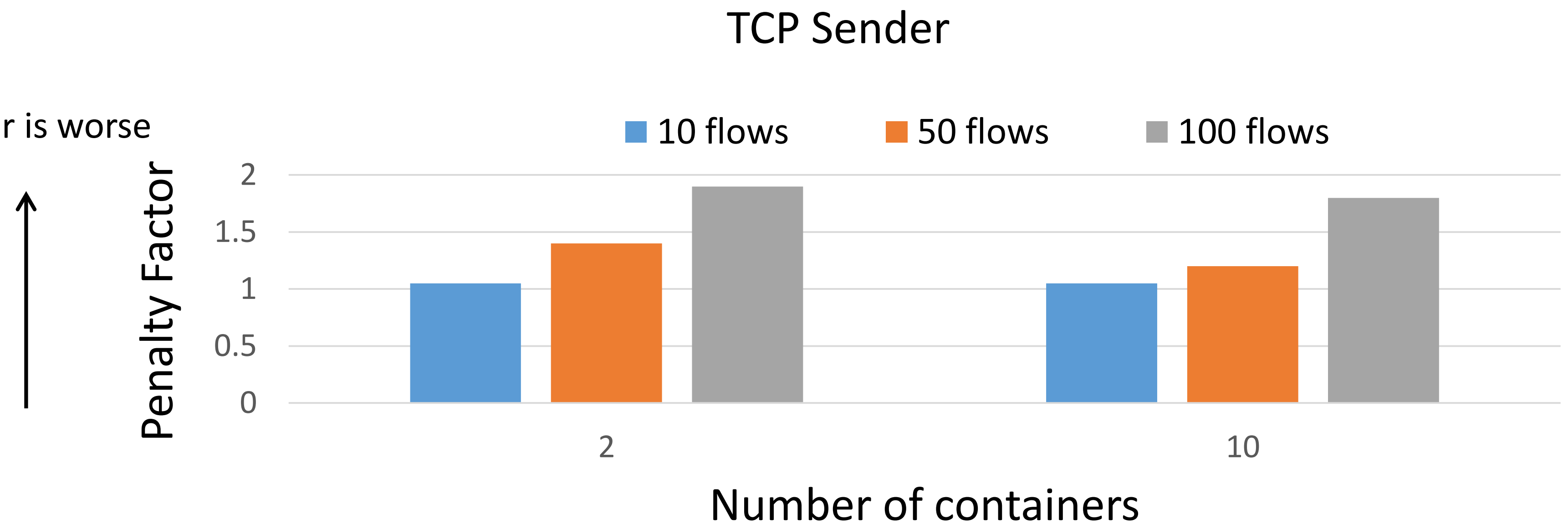


HTB is used for traffic shaping @ 5Gbps

Impact Of Network Traffic

Penalty factor = ***Time that victim takes when competing with traffic*** / ***Time that victim takes when competing with sysbench***

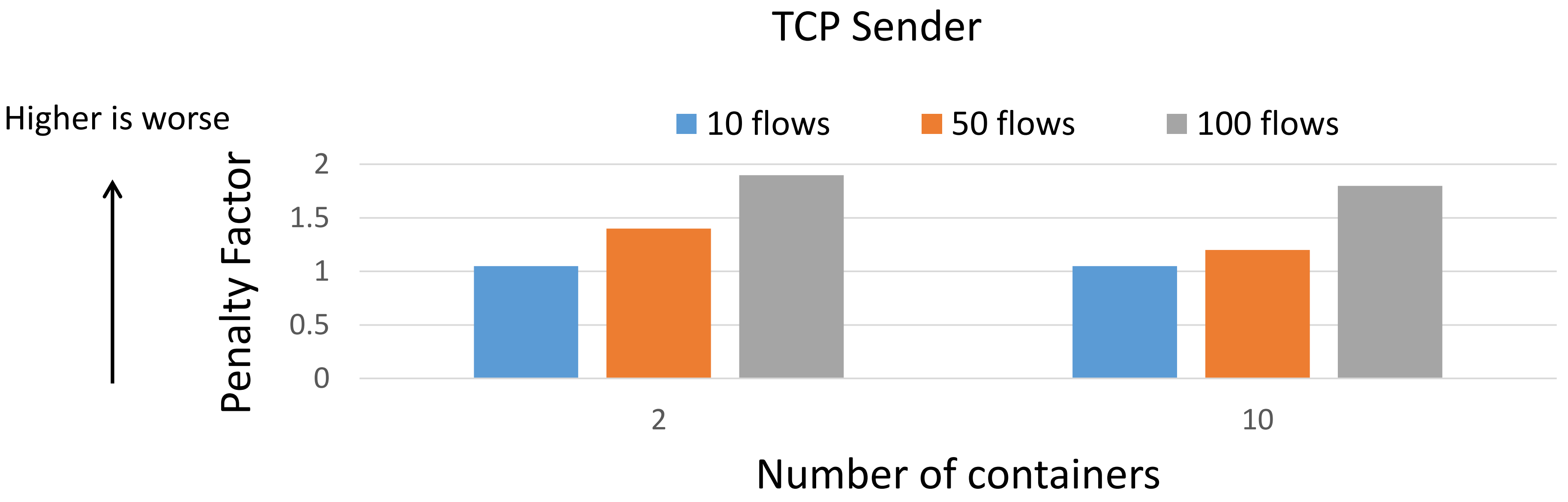
Higher is worse



HTB is used for traffic shaping @ 5Gbps

Impact Of Network Traffic

$$\text{Penalty factor} = \frac{\textit{Time that victim takes when competing with traffic}}{\textit{Time that victim takes when competing with sysbench}}$$

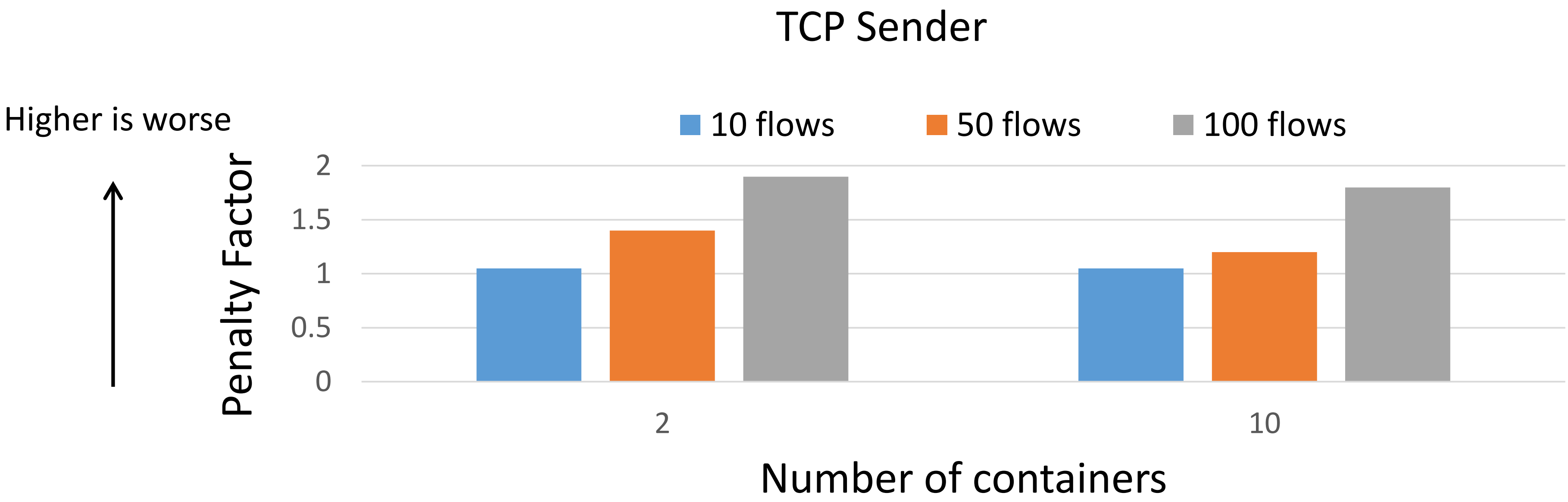


HTB is used for traffic shaping @ 5Gbps

Maximum penalty factor is around ***1.85***

Impact Of Network Traffic

$$\text{Penalty factor} = \frac{\text{Time that victim takes when competing with traffic}}{\text{Time that victim takes when competing with sysbench}}$$

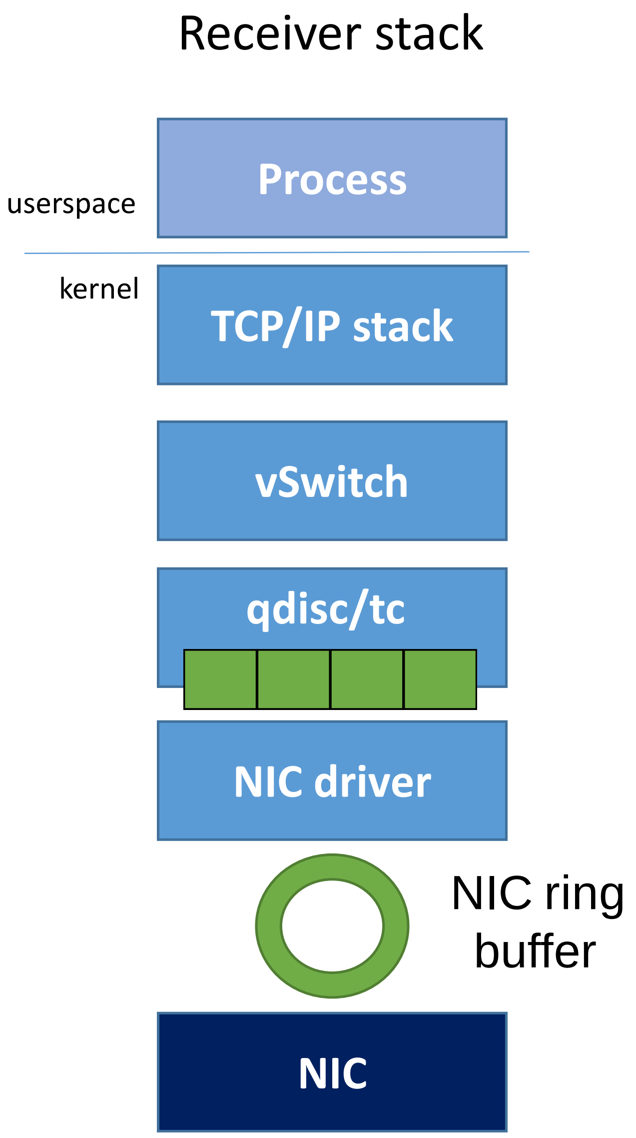


HTB is used for traffic shaping @ 5Gbps

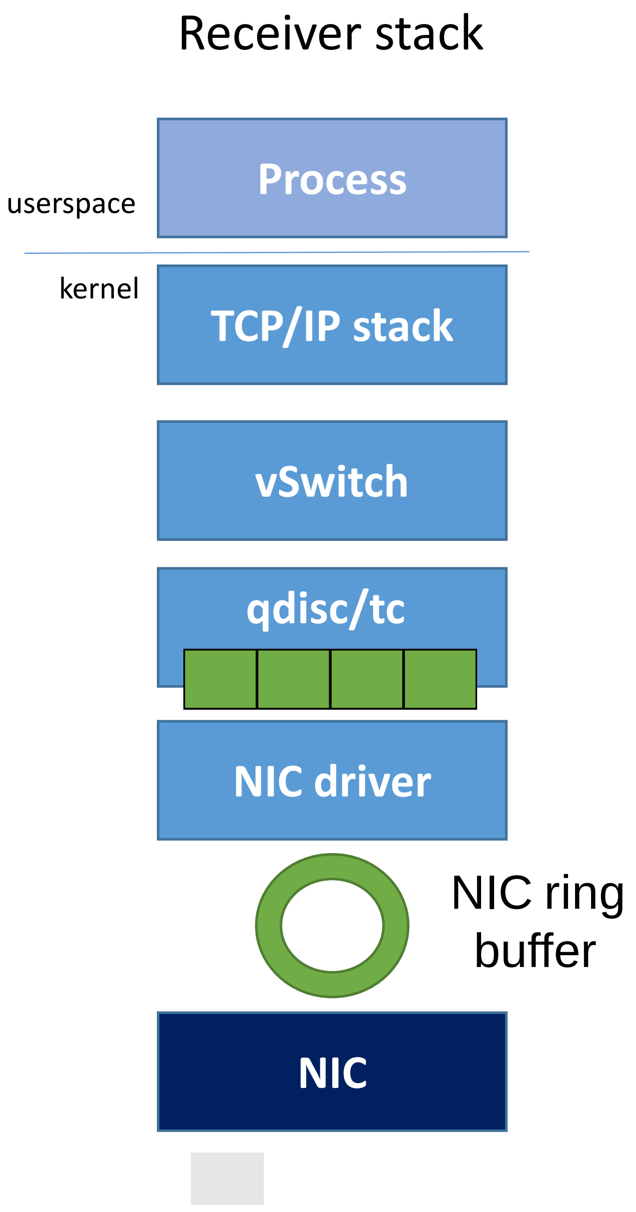
Maximum penalty factor is around ***1.85***

Look at our paper for the impact of UDP traffic

Receiver Side

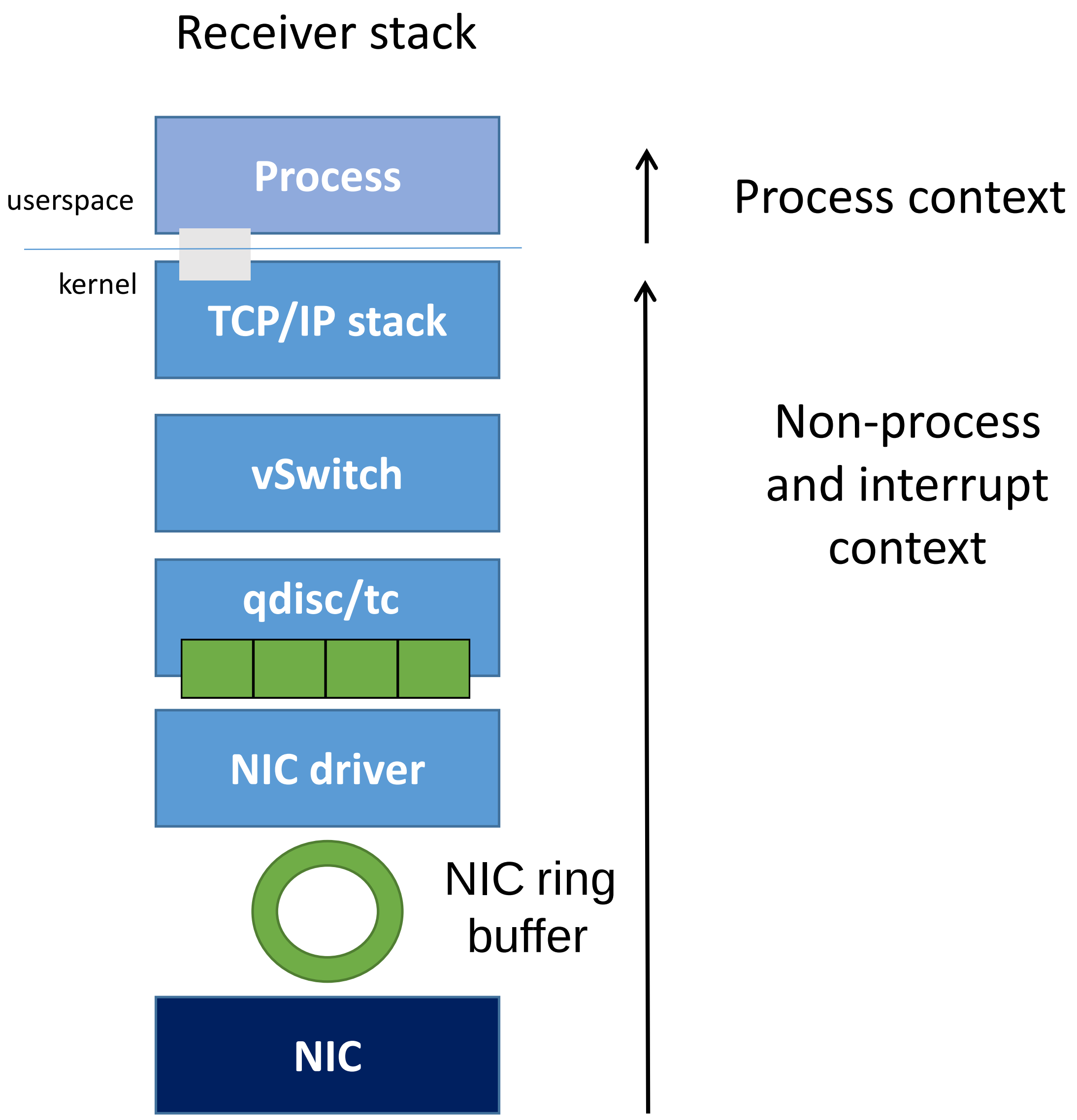


Receiver Side



- Receiver side problem is much worse than the sender

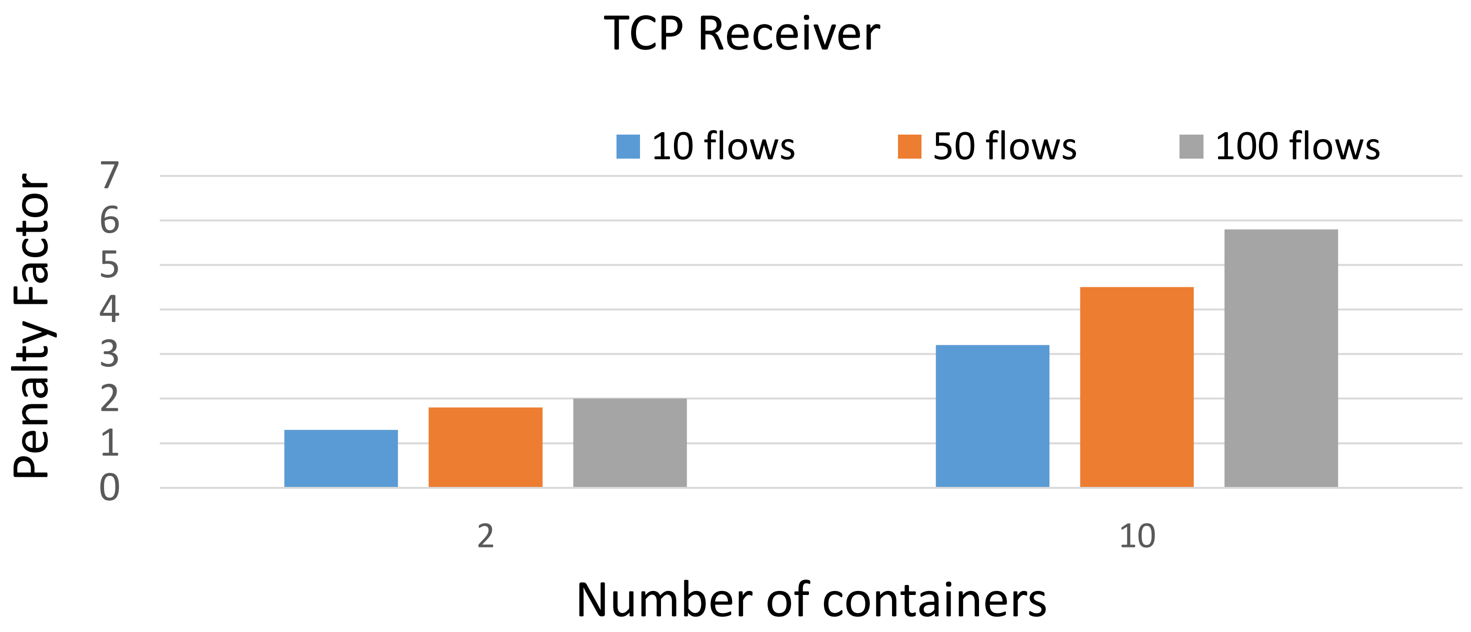
Receiver Side



- Receiver side problem is much worse than the sender
- Packet is processed in non-process context until copied to application's socket

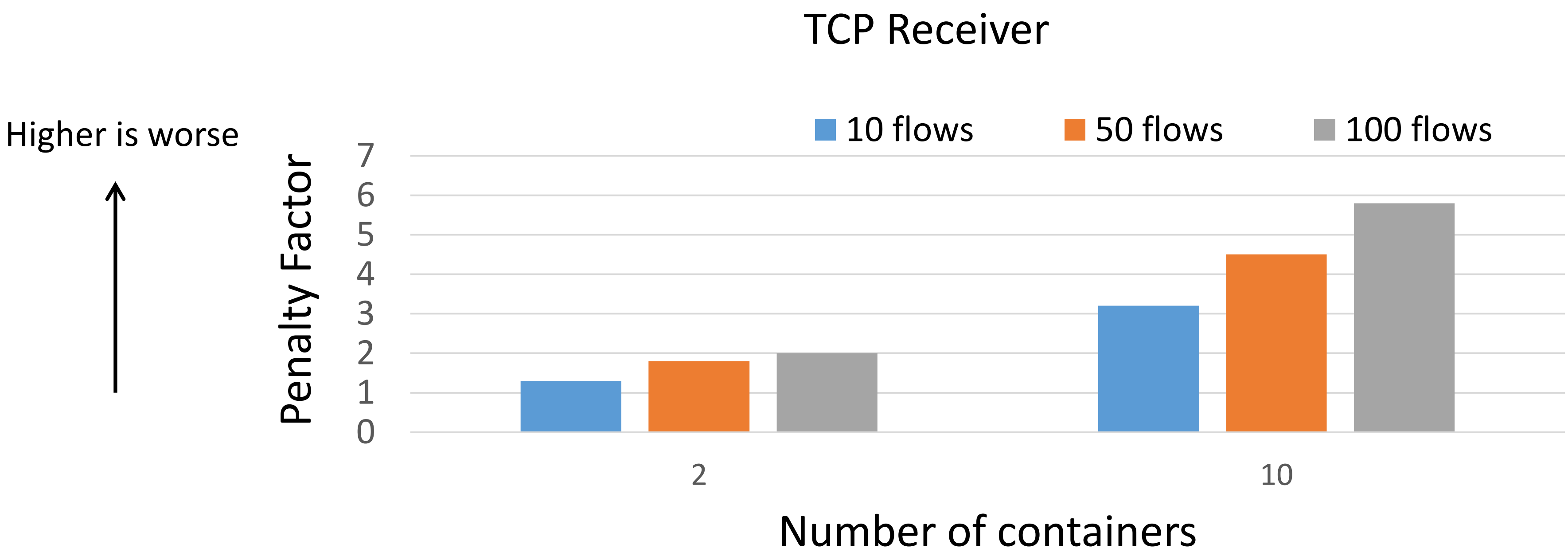
Impact Of Network Traffic

$$\textit{Penalty factor} = \frac{\textit{Time that victim takes when competing with traffic}}{\textit{Time that victim takes when competing with sysbench}}$$



Impact Of Network Traffic

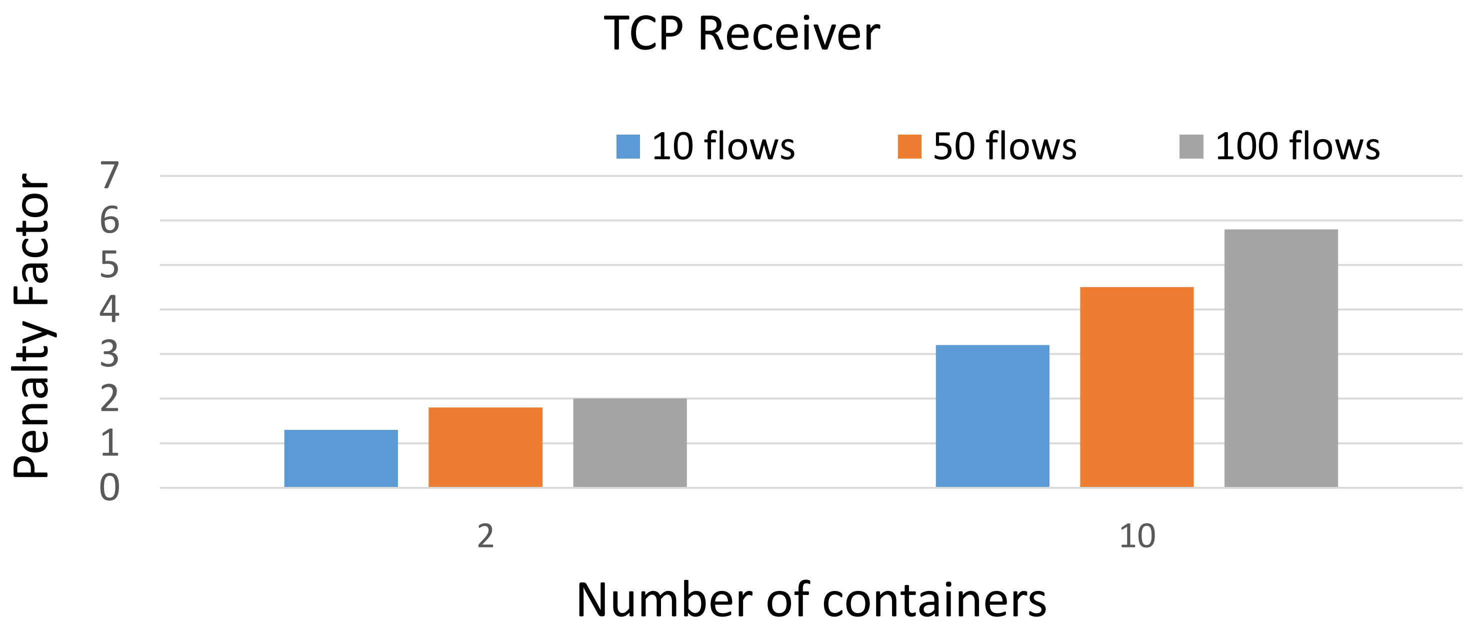
Penalty factor = *Time that victim takes when competing with traffic* / *Time that victim takes when competing with sysbench*



Impact Of Network Traffic

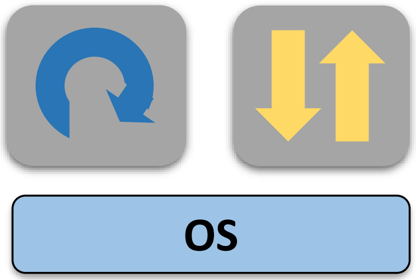
Penalty factor = *Time that victim takes when competing with traffic* / *Time that victim takes when competing with sysbench*

Higher is worse



Maximum penalty factor is around ***6***

Scenarios When Isolation Breaks

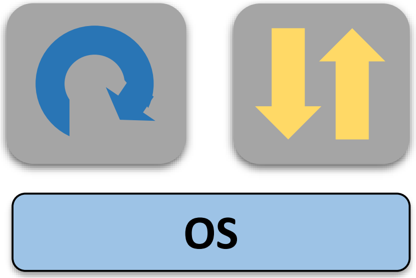


Compute intensive

vs

Network intensive

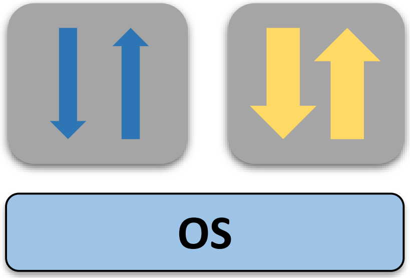
Scenarios When Isolation Breaks



Compute intensive

vs

Network intensive

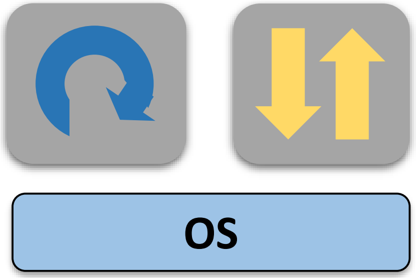


Low network workload

vs

High network workload

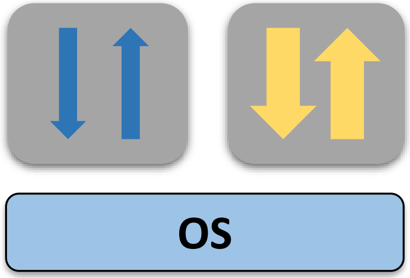
Scenarios When Isolation Breaks



Compute intensive

vs

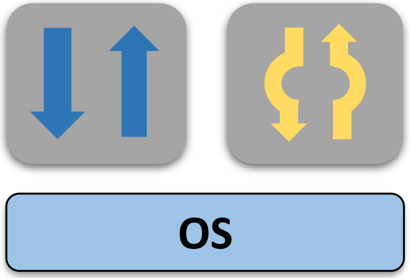
Network intensive



Low network workload

vs

High network workload



Network intensive

vs

Network intensive with kernel bypass

Iron

A scheme that ensures and enforces accounting of ***network-based CPU*** consumed in the kernel on the **behalf of a *container***.

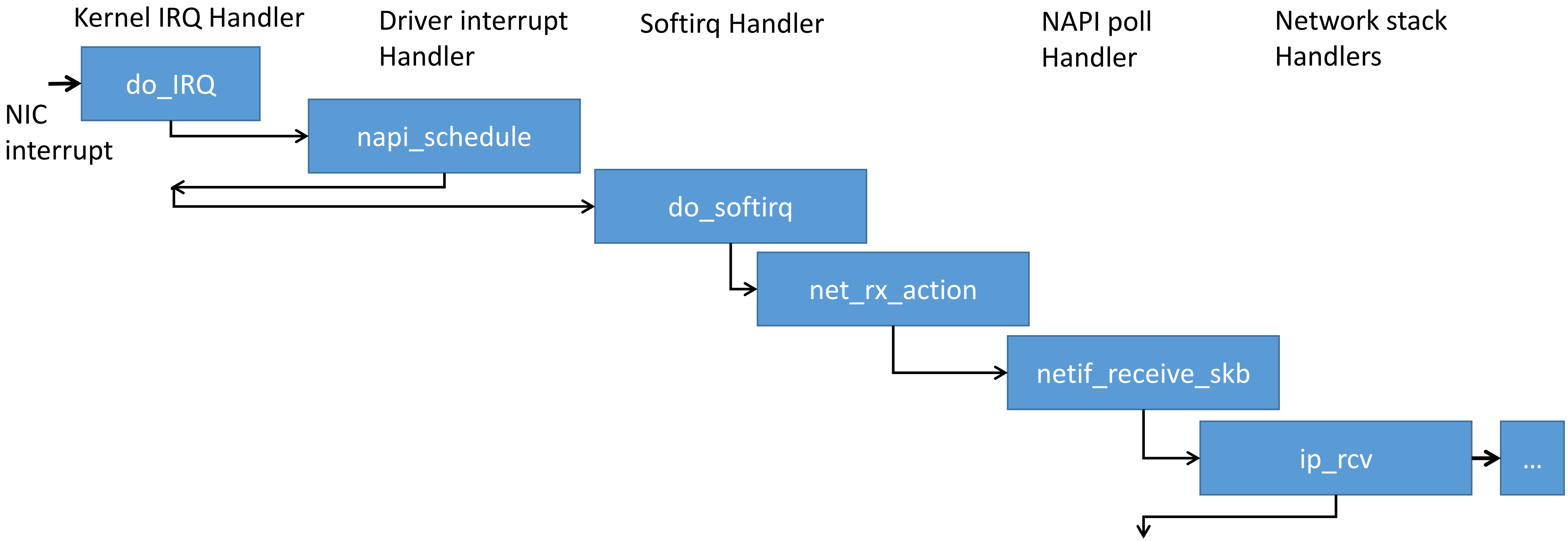
Outline

- How and by how much is isolation broken
- Iron’s design
    - **Accounting of per-packet processing cost**
    - Ensuring isolation via enforcement
        - Integration with Linux scheduler
        - Hardware-based packet dropping
- Evaluation
    - Controlled workload
    - Realistic workload

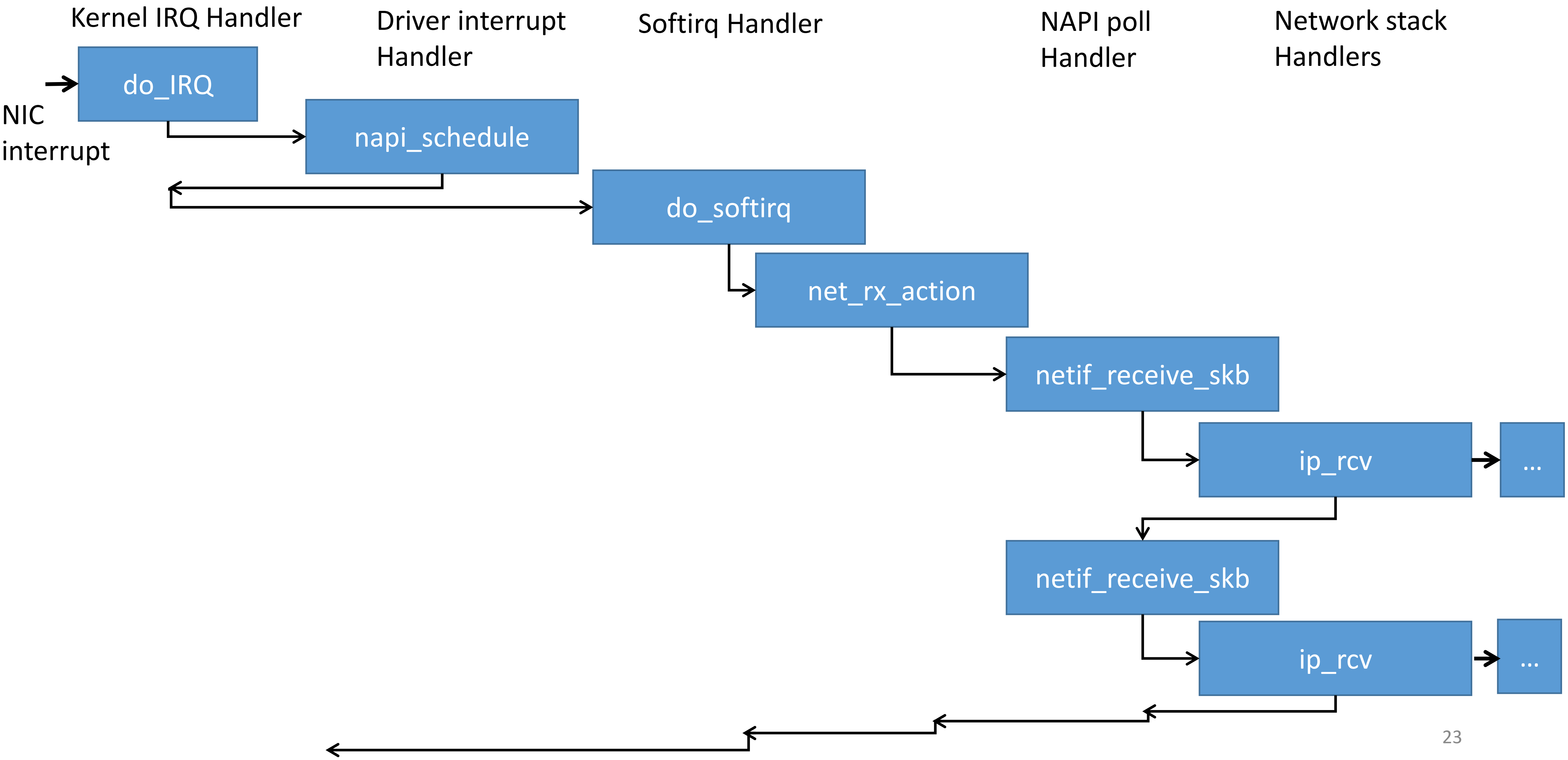
Network Call Stack – Backgound

NIC
interrupt

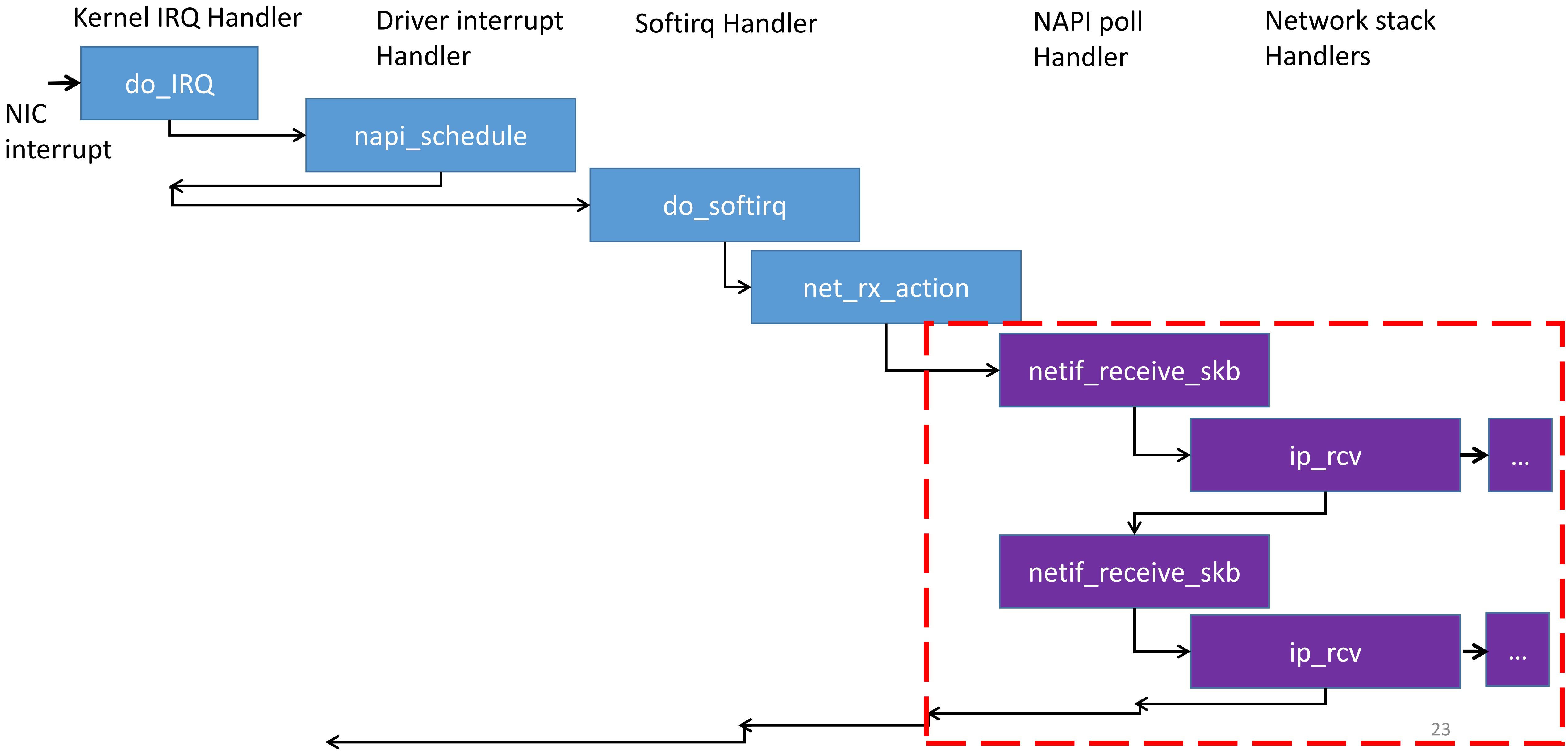
Network Call Stack – Backgound



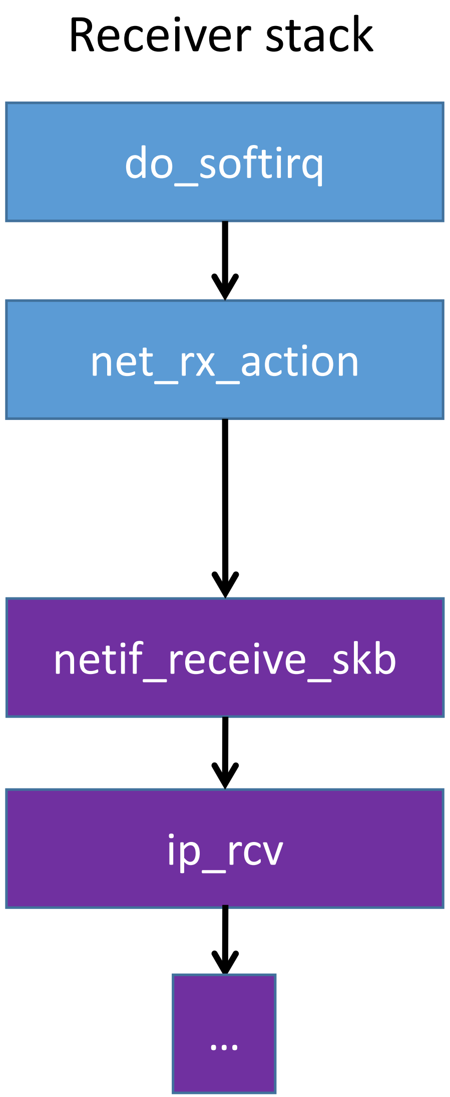
Network Call Stack – Backgound



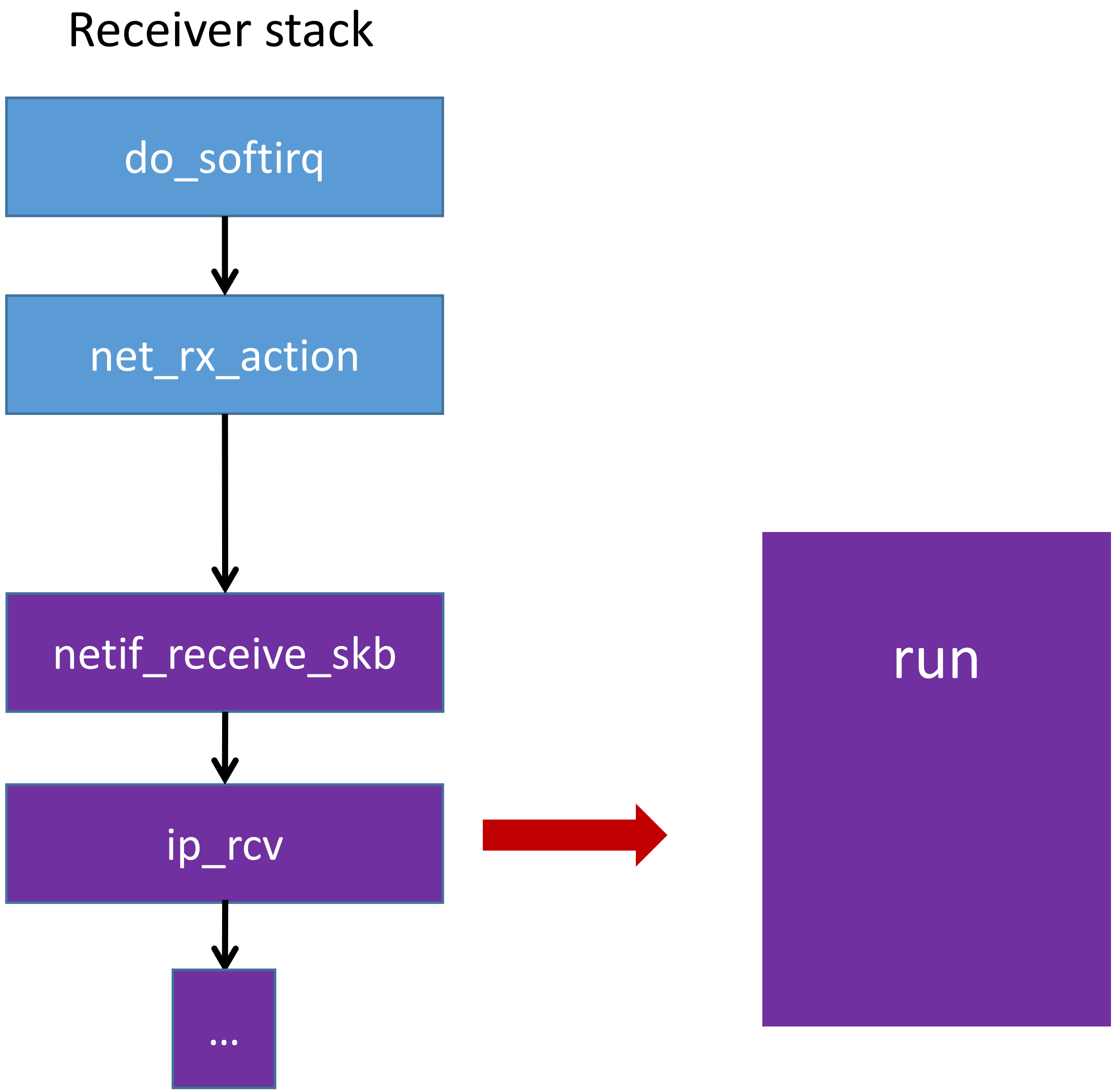
Network Call Stack – Backgound



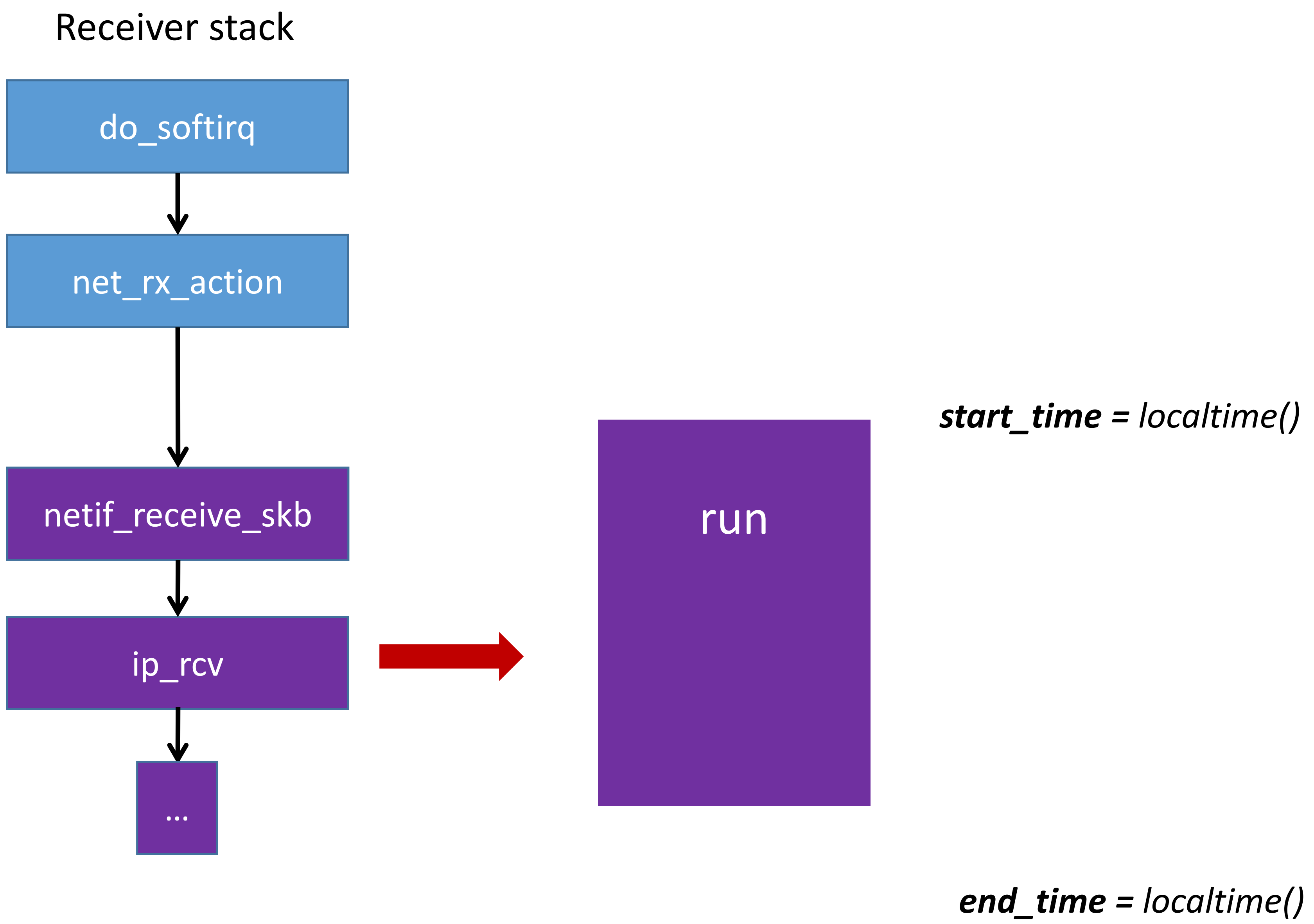
Iron – Accounting



Iron – Accounting

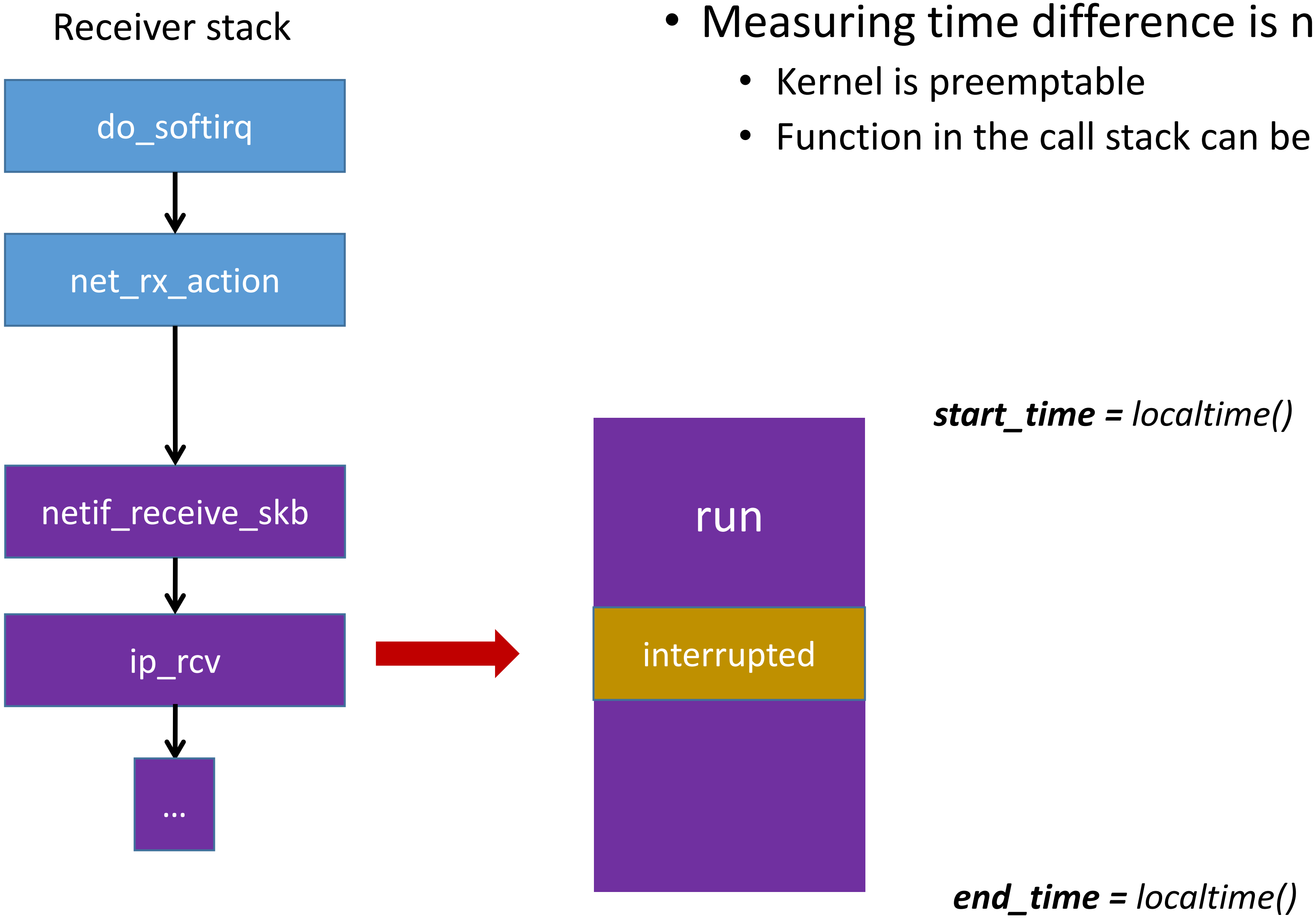


Iron – Accounting

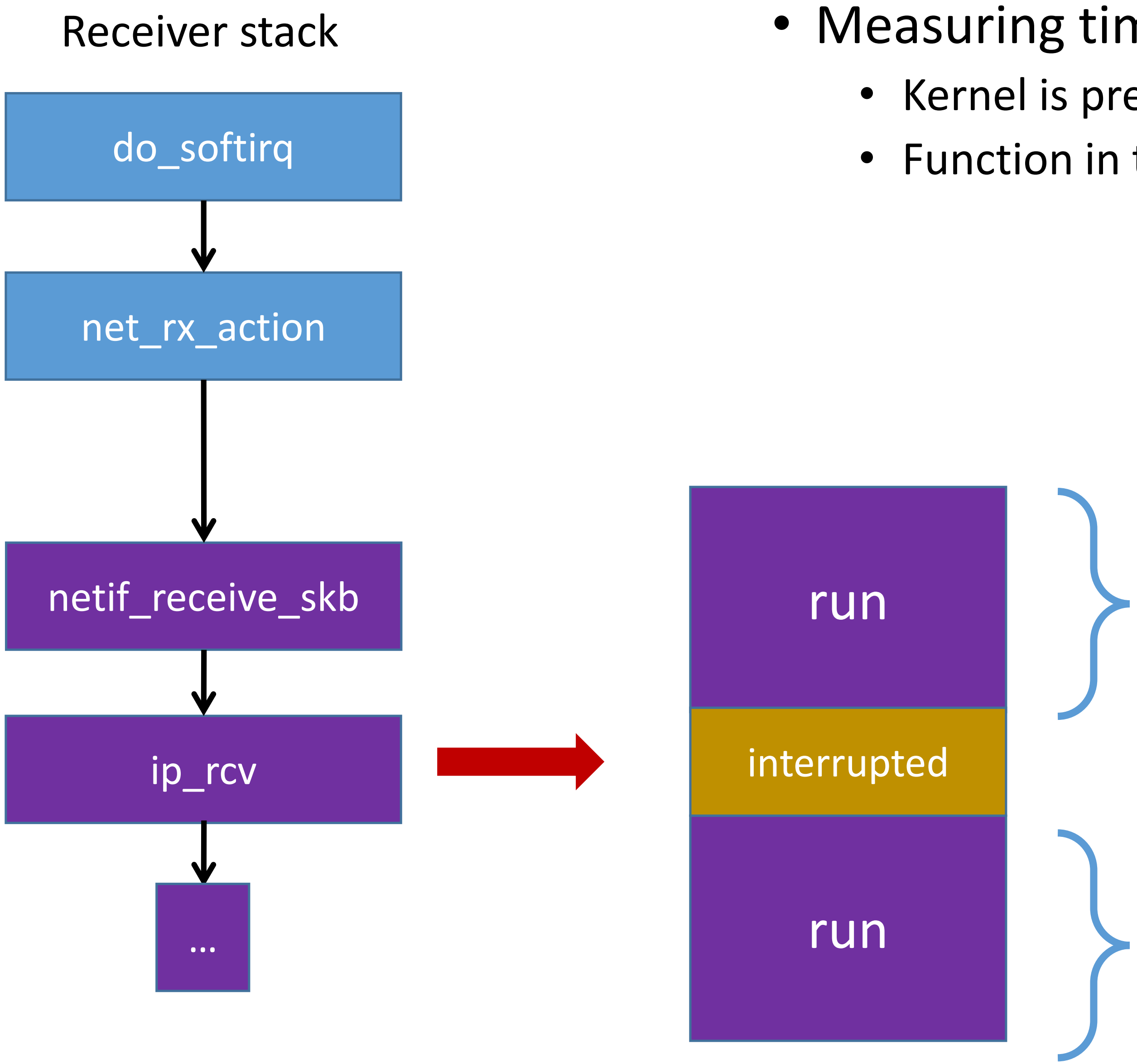


Iron – Accounting

- Measuring time difference is non-trivial
  - Kernel is preemptable
  - Function in the call stack can be interrupted at any time



Iron – Accounting

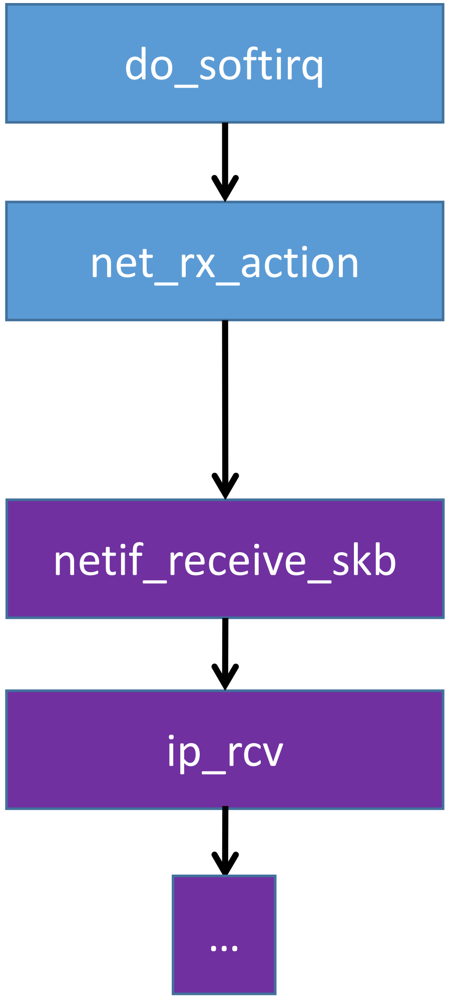


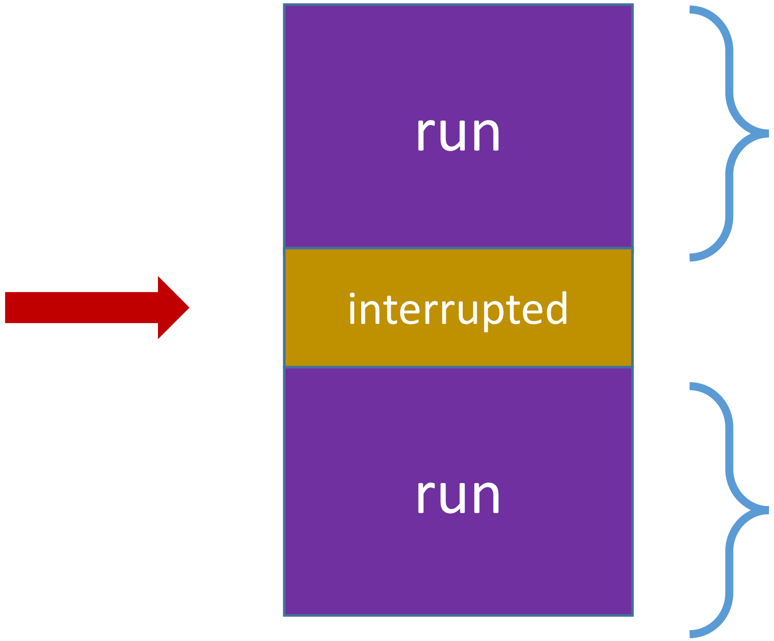
- Measuring time difference is non-trivial
  - Kernel is preemptable
  - Function in the call stack can be interrupted at any time

Iron – Accounting

- Measuring time difference is non-trivial
  - Kernel is preemptable
  - Function in the call stack can be interrupted at any time

Receiver stack

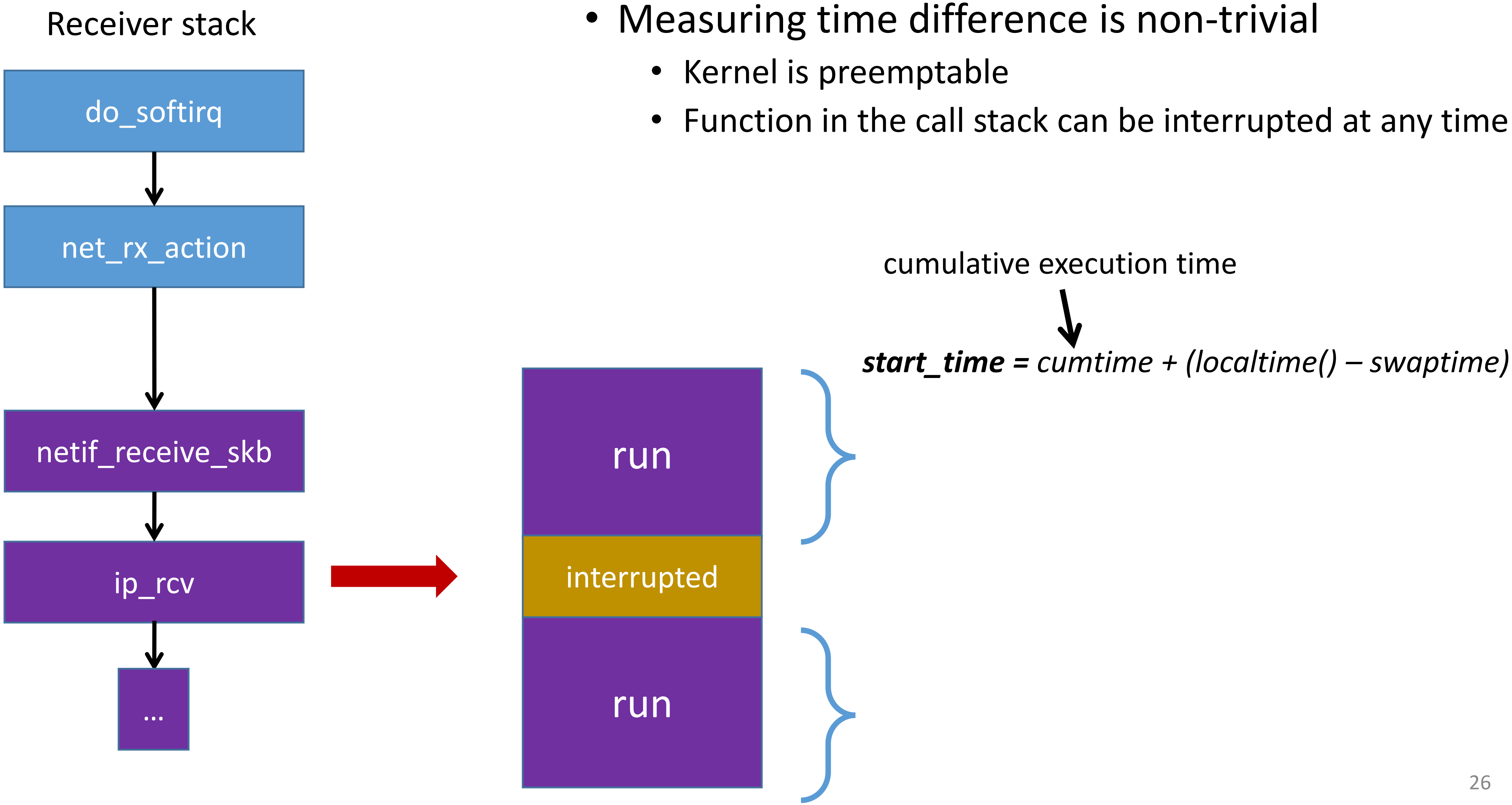




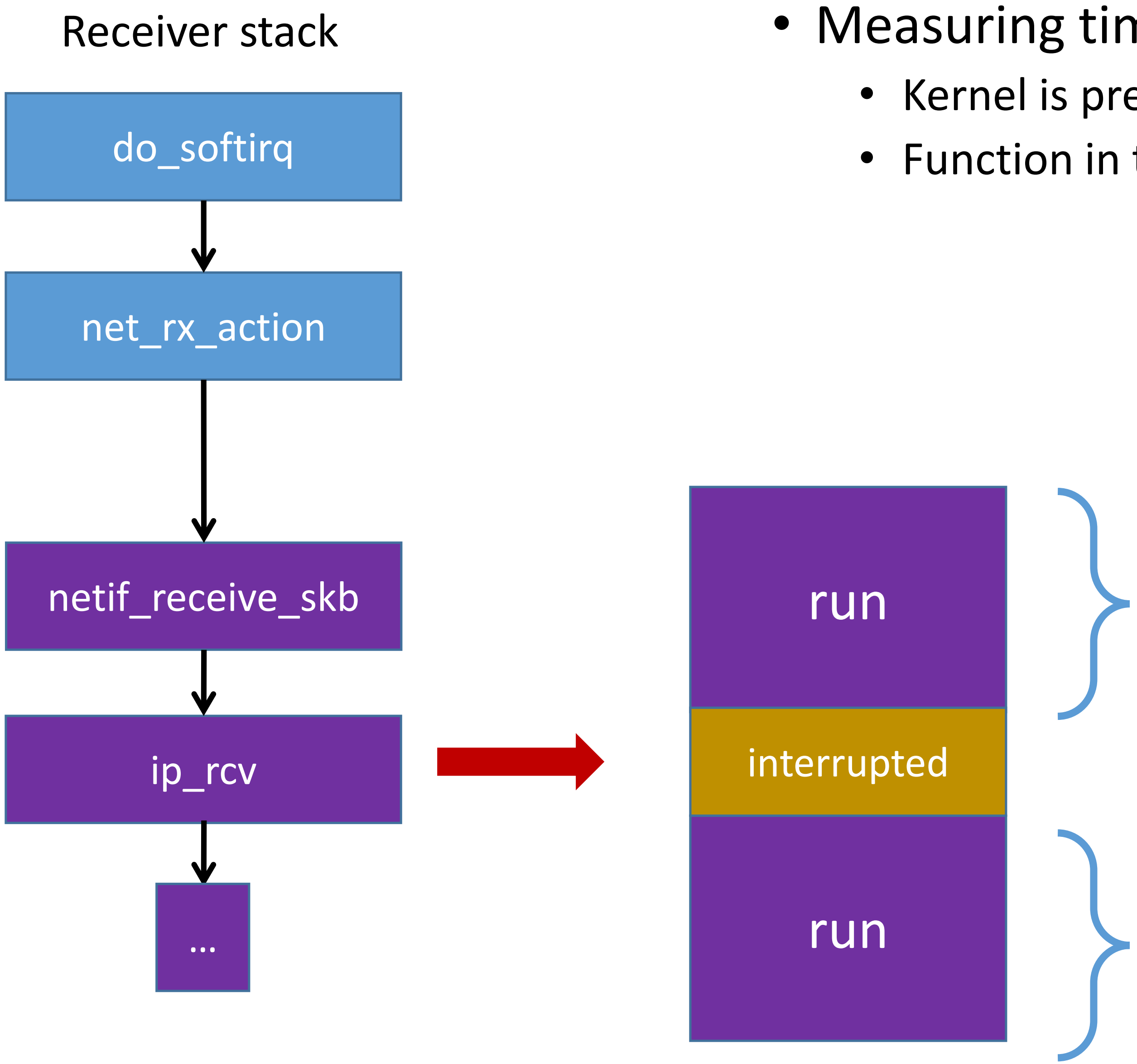
start_time = *cumtime + (localtime() – swaptime)*

Iron – Accounting

- Measuring time difference is non-trivial
  - Kernel is preemptable
  - Function in the call stack can be interrupted at any time



Iron – Accounting



- Measuring time difference is non-trivial
  - Kernel is preemptable
  - Function in the call stack can be interrupted at any time

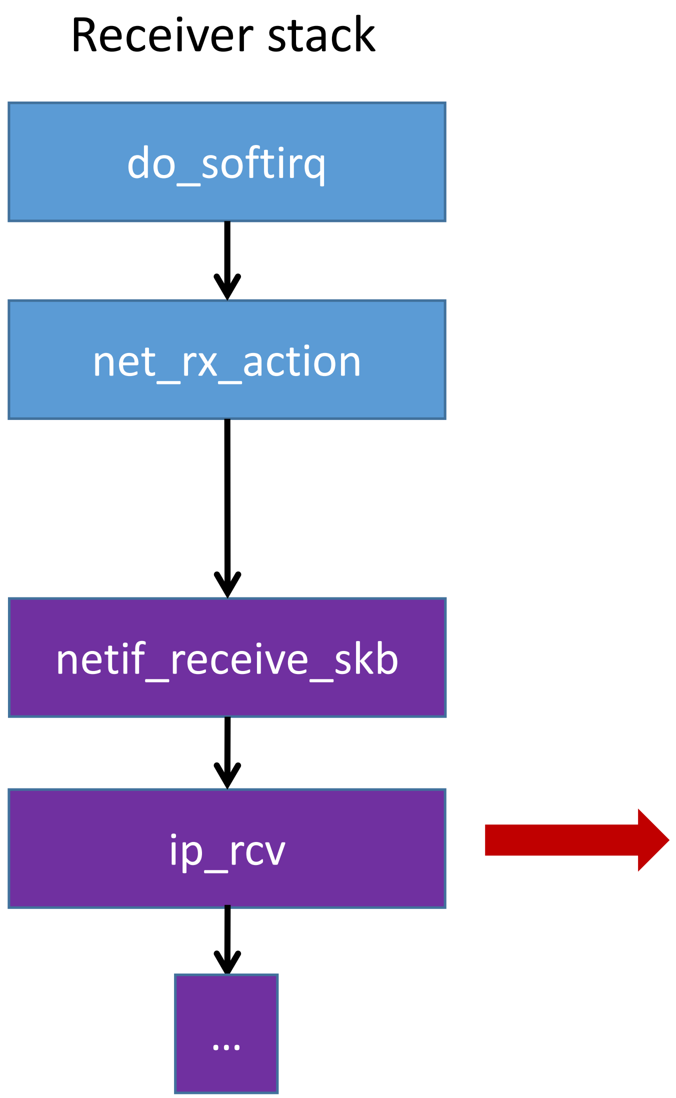
cumulative execution time

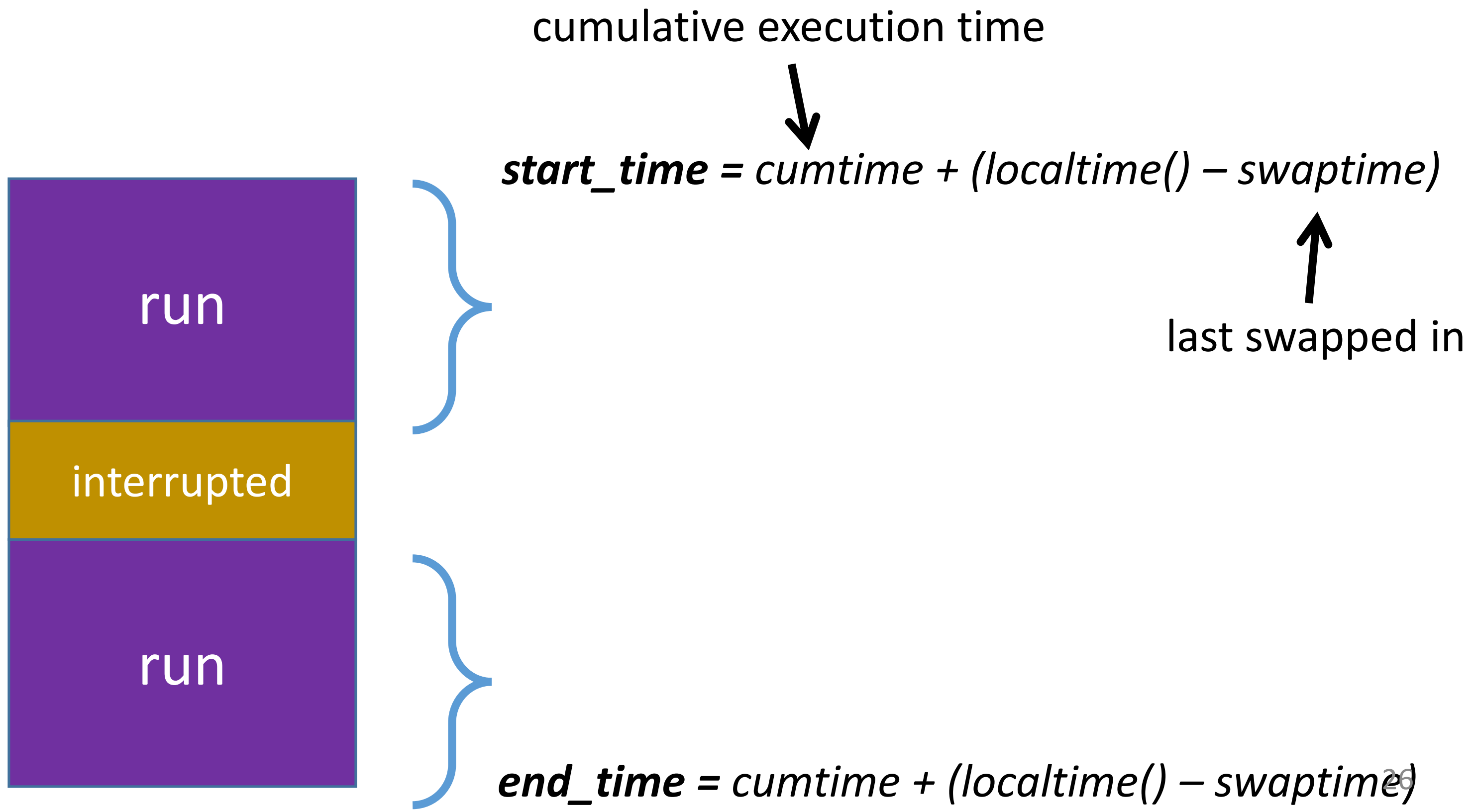
***start_time* =** *cumtime + (localtime() – swaptime)*

last swapped in

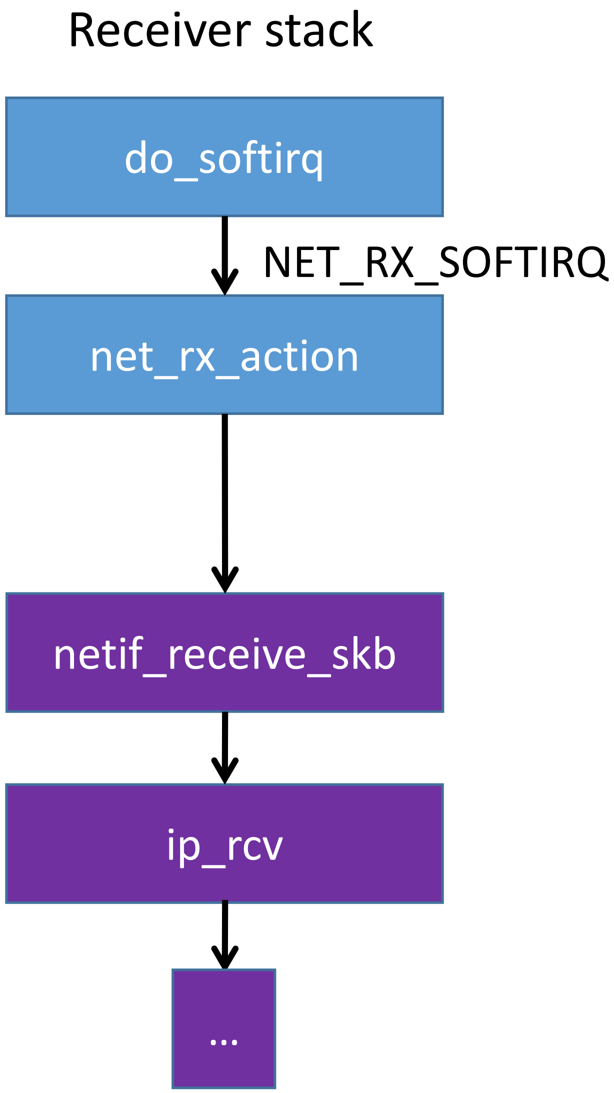
Iron – Accounting

- Measuring time difference is non-trivial
  - Kernel is preemptable
  - Function in the call stack can be interrupted at any time

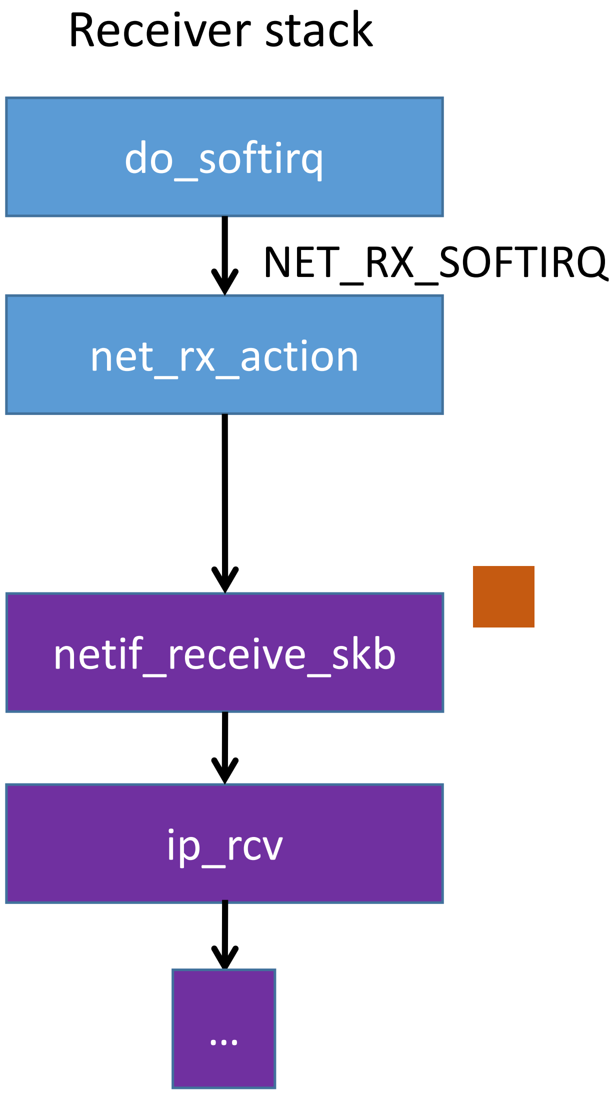




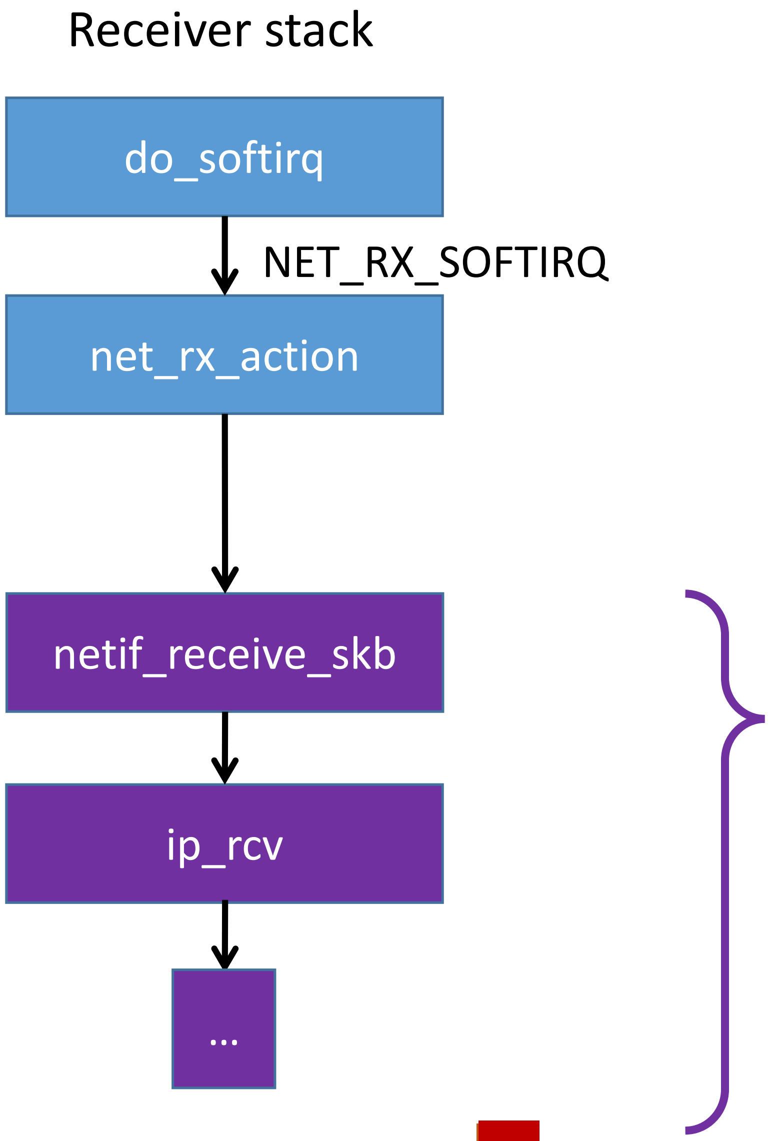
Iron – Accounting



Iron – Accounting



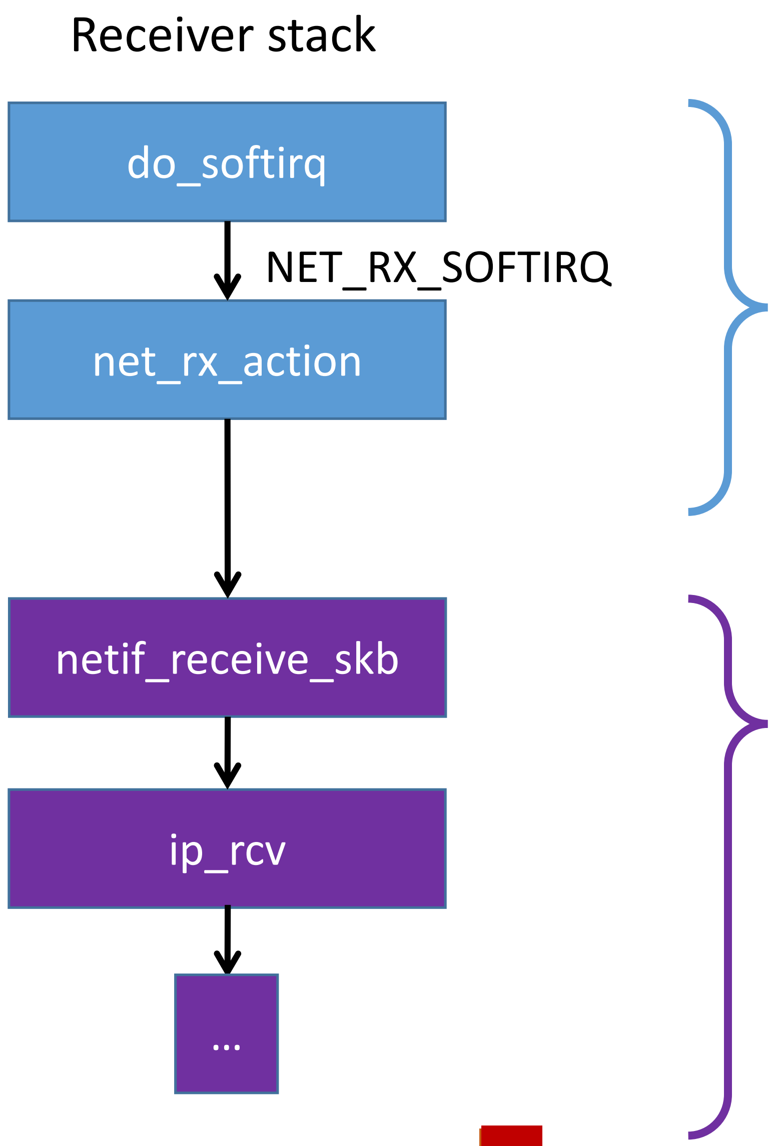
Iron – Accounting



$$P_i = per\_pkt\_cost$$

$$pktcost_i = P_i$$

Iron – Accounting

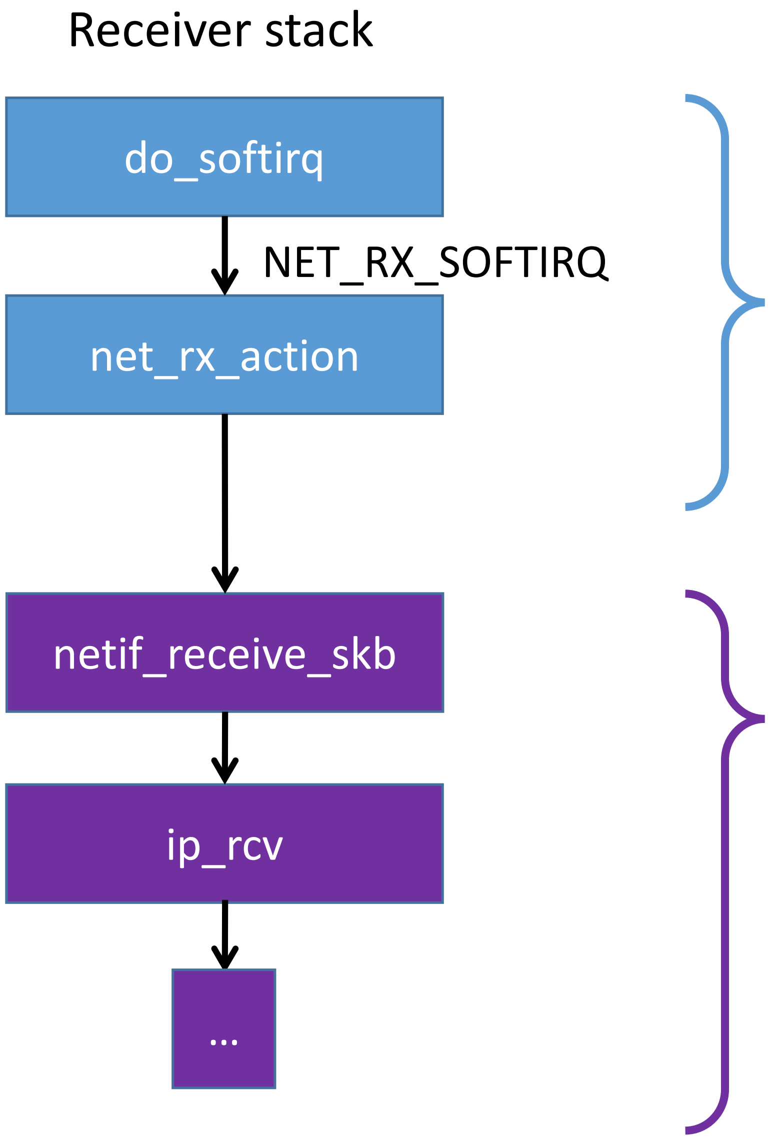


$batch\_cost$

$P_i = per\_pkt\_cost$

$pktcost_i = P_i$

Iron – Accounting



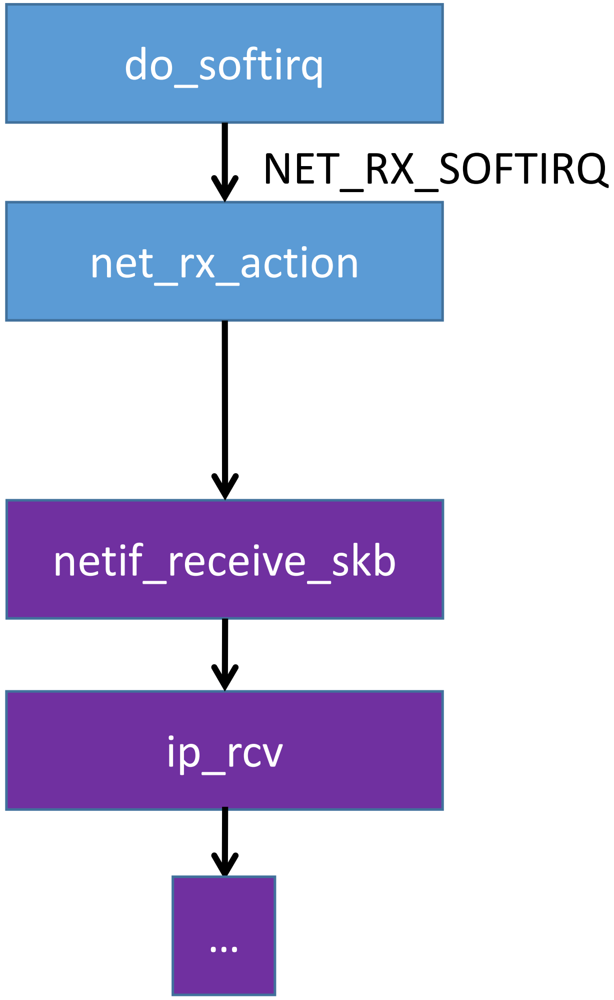
$batch\_cost$

$P_i = per\_pkt\_cost$

$$pktcost_i = P_i + \frac{batch\_cost}{|P|}$$

Iron – Accounting

Receiver stack



HI, TX, RX, TIMER, SCSI & TASKLET

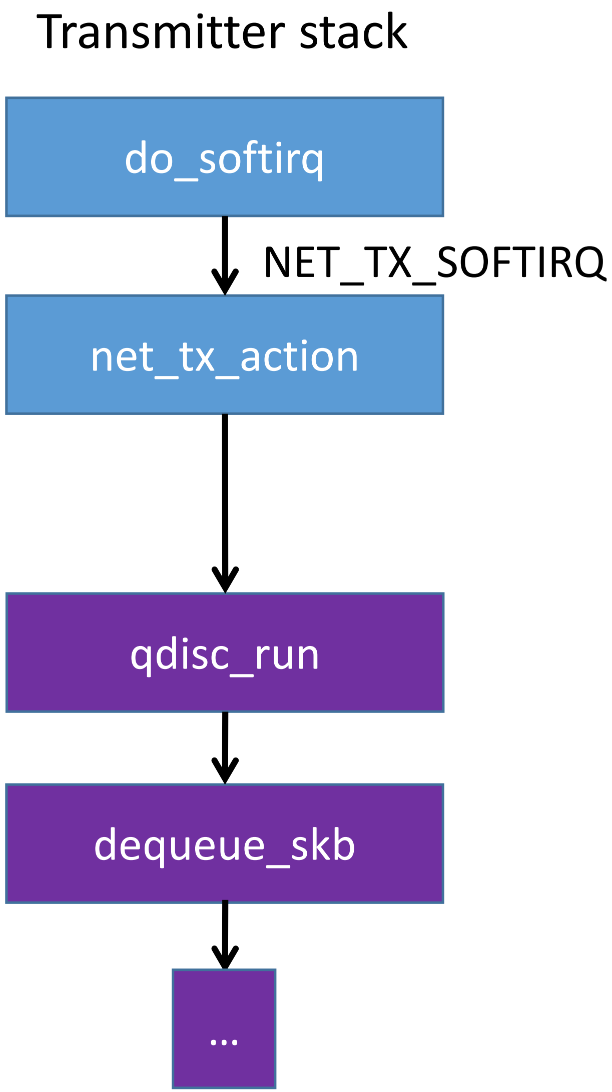
$$batch\_cost = \left(do\_softirq\_cost - \sum S_i \right) * \frac{S_{RX}}{\sum S_i}$$

cost of all the softirqs

$$P_i = per\_pkt\_cost$$

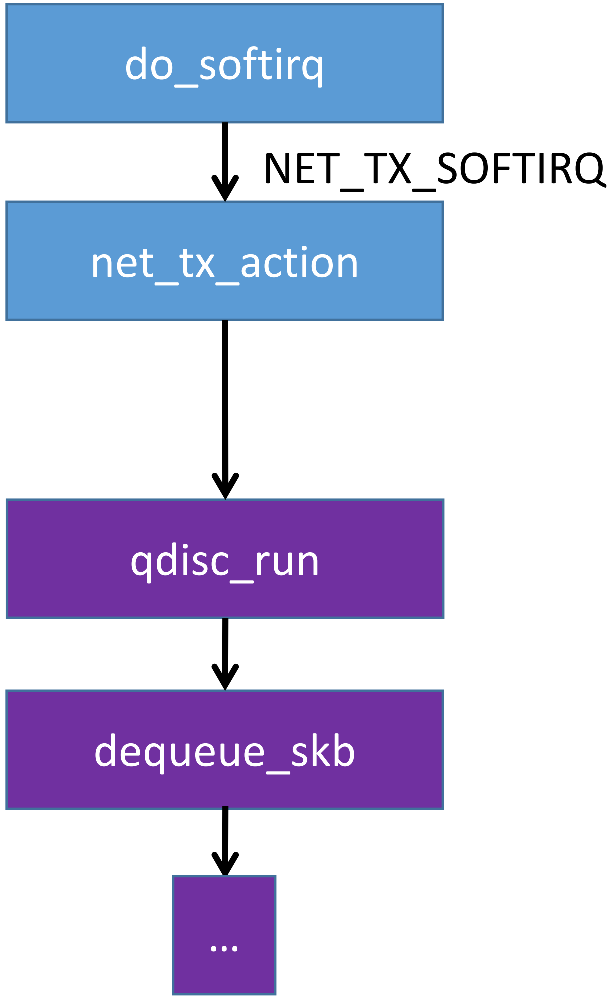
$$pktcost_i = P_i + \frac{batch\_cost}{|P|}$$

Iron – Accounting



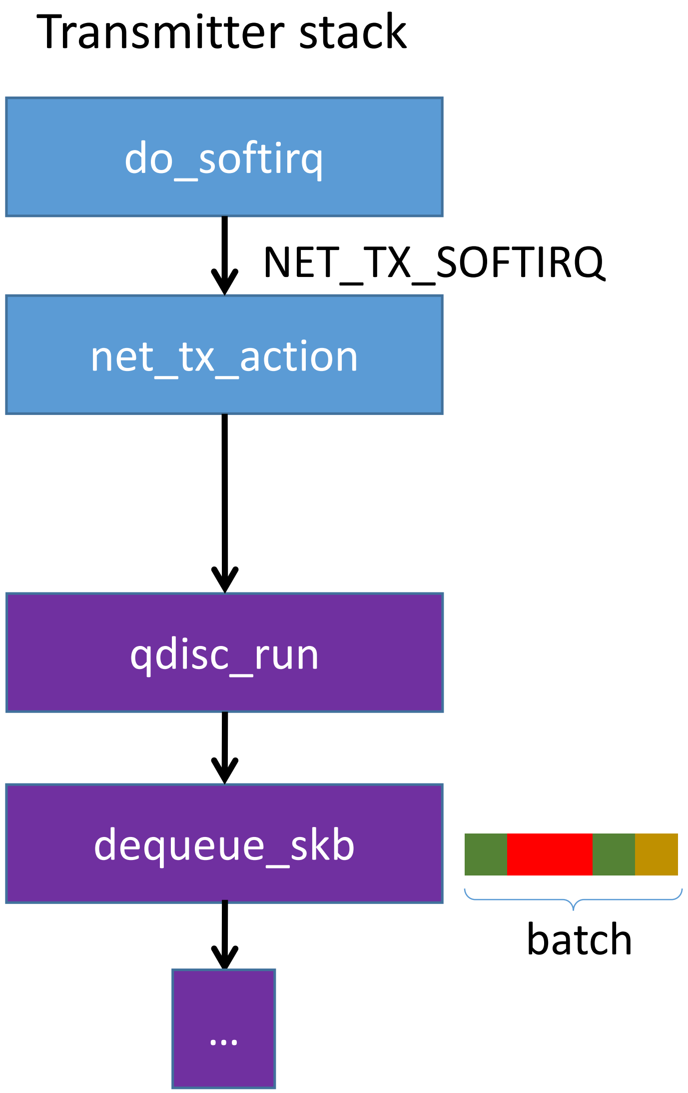
Iron – Accounting

Transmitter stack



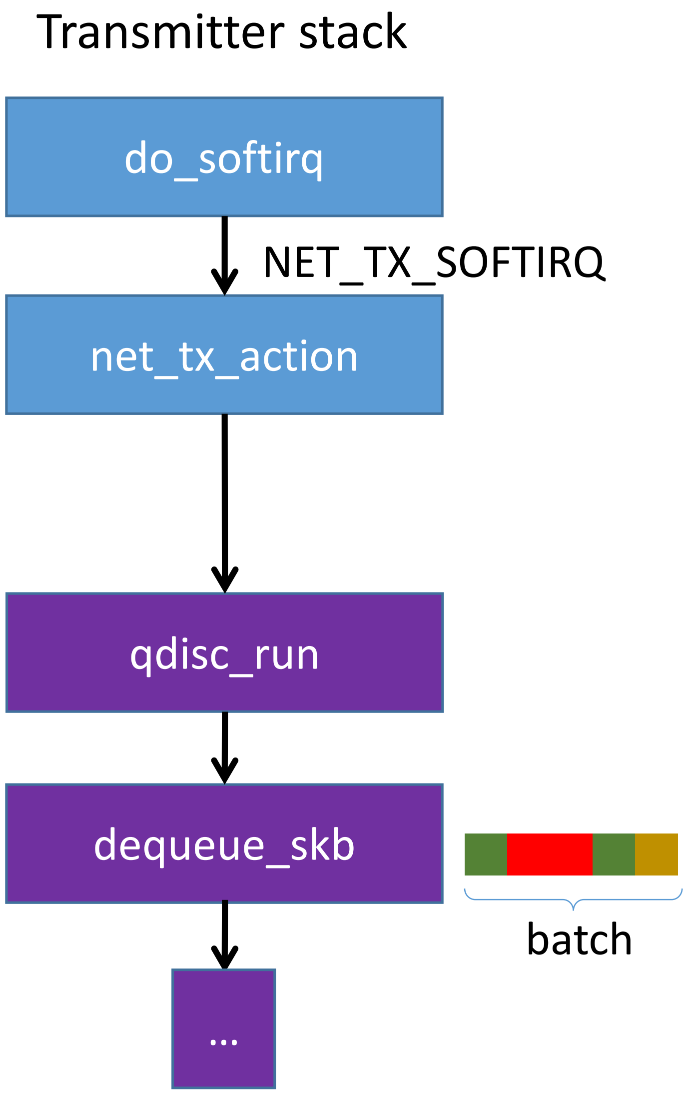
- Linux batches packets for transmission

Iron – Accounting



- Linux batches packets for transmission

Iron – Accounting

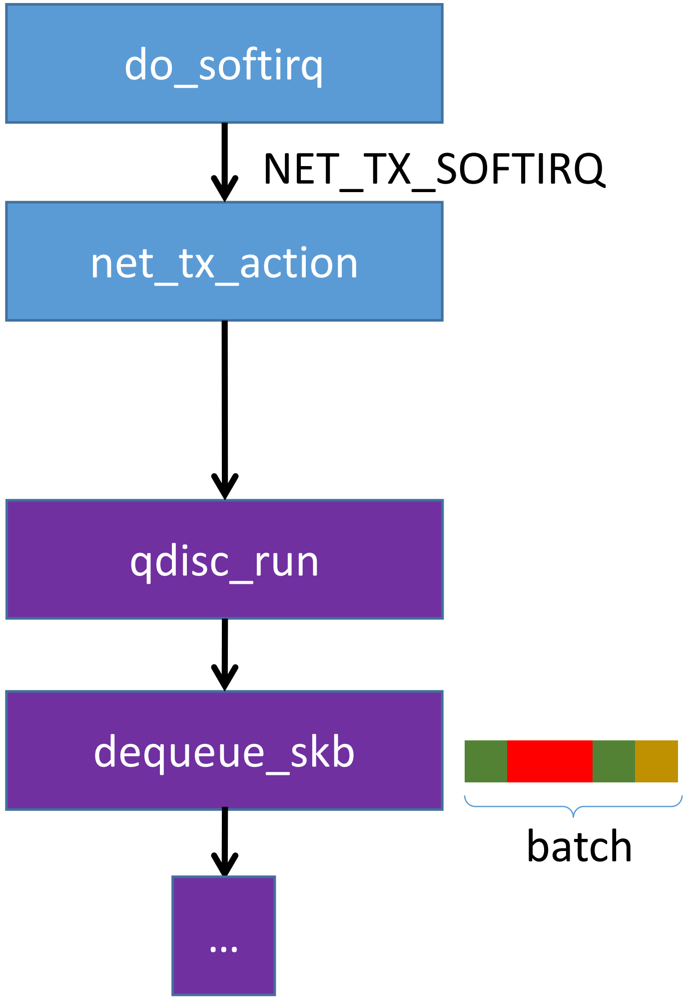


- Linux batches packets for transmission
- We measure the cost of the batch and charge each packet within the batch an equal share

$$pktcost = \frac{batch\_cost}{batch\_size}$$

Iron – Accounting

Transmitter stack



- Linux batches packets for transmission
- We measure the cost of the batch and charge each packet within the batch an equal share
- To identify the container to charge at dequeue
  - We encode the container information in the skb while enqueueing the packet

$$pktcost = \frac{batch\_cost}{batch\_size}$$

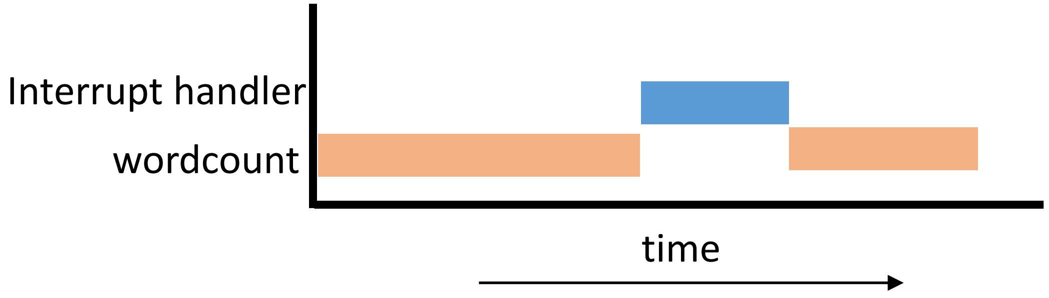
Outline

- How and by how much is isolation broken
- Iron's Design
  - Accounting of per-packet processing cost
  - **Ensuring isolation via enforcement**
    - **Integration with Linux scheduler**
    - Hardware-based packet dropping
- Evaluation
  - Controlled workload
  - Realistic workload

Iron – Enforcement

Scheduler Integration

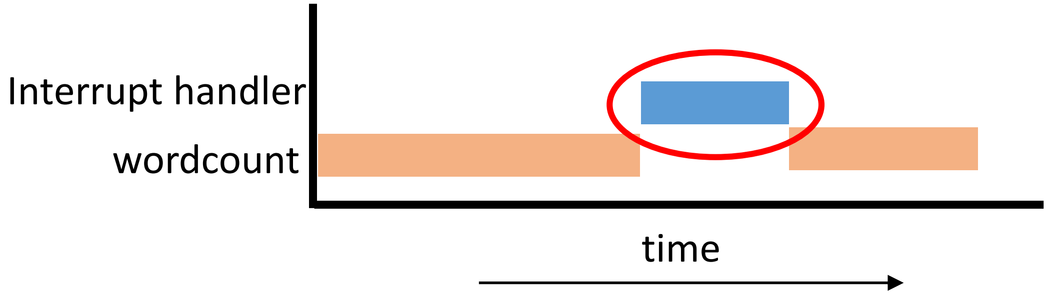
- ***Return the accounted time*** to the container which was ***incorrectly*** charged



Iron – Enforcement

Scheduler Integration

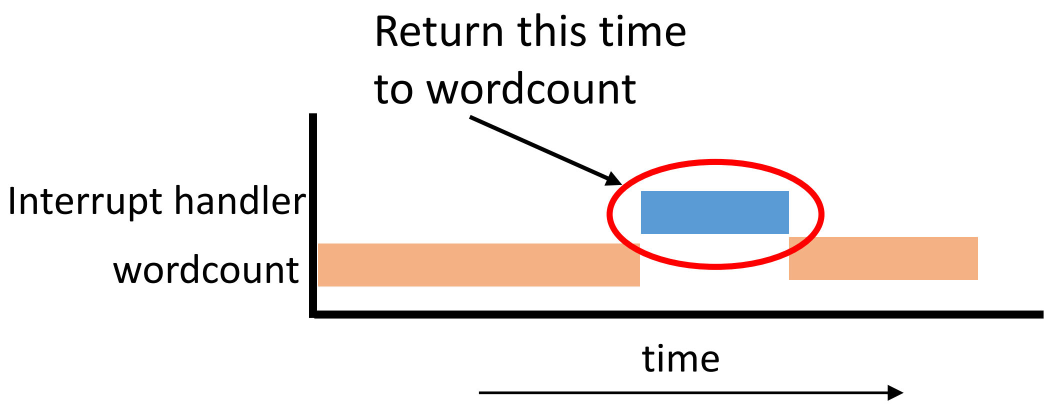
- ***Return the accounted time*** to the container which was ***incorrectly*** charged



Iron – Enforcement

Scheduler Integration

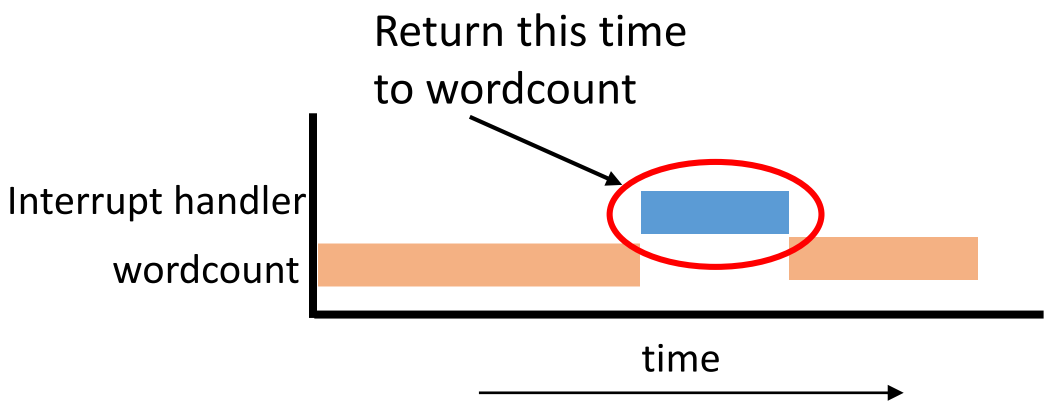
- ***Return the accounted time*** to the container which was ***incorrectly*** charged



Iron – Enforcement

Scheduler Integration

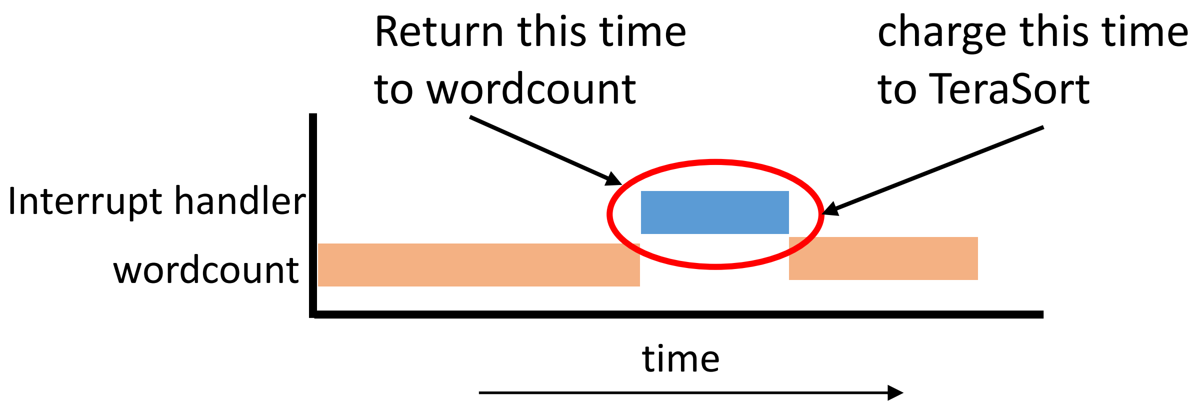
- ***Return the accounted time*** to the container which was ***incorrectly*** charged
- ***Charge the accounted time*** to the container which was ***responsible*** for the network traffic



Iron – Enforcement

Scheduler Integration

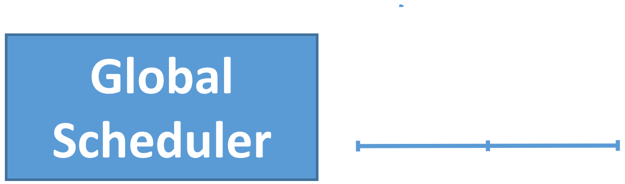
- ***Return the accounted time*** to the container which was ***incorrectly*** charged
- ***Charge the accounted time*** to the container which was ***responsible*** for the network traffic



Iron – Enforcement

Scheduler Integration

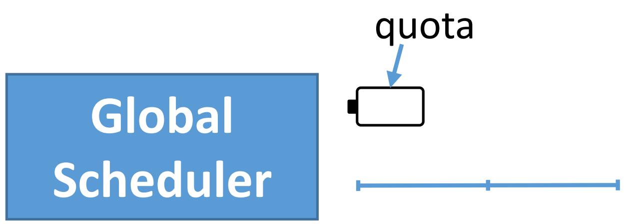
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

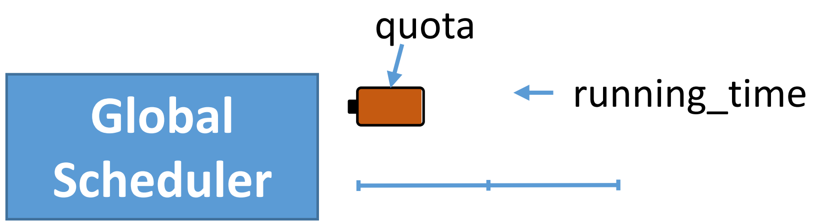
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

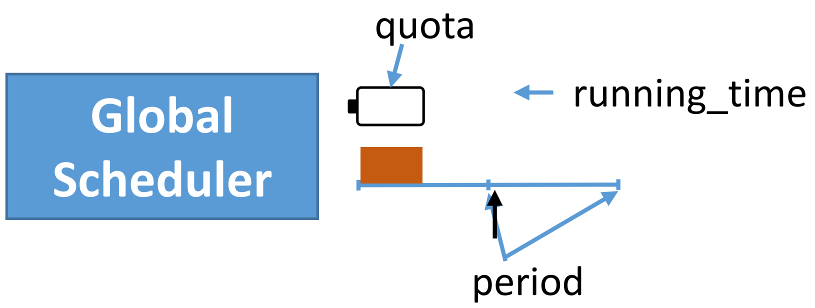
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

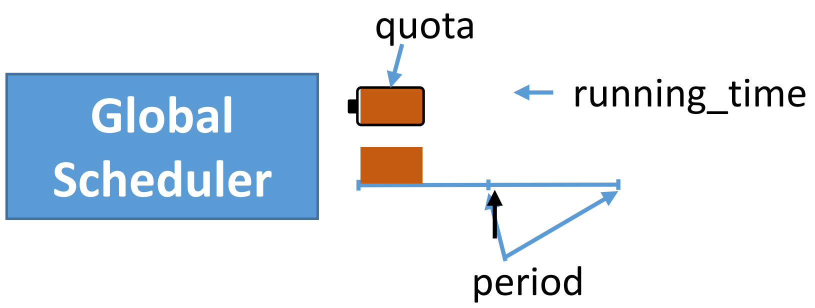
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

Reuse infrastructure from cgroup and Linux scheduler

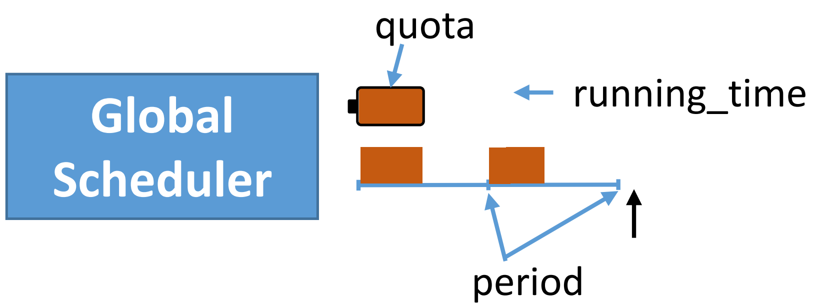


At the end of the period, *running_time* is refilled by *quota*.

Iron – Enforcement

Scheduler Integration

Reuse infrastructure from cgroup and Linux scheduler

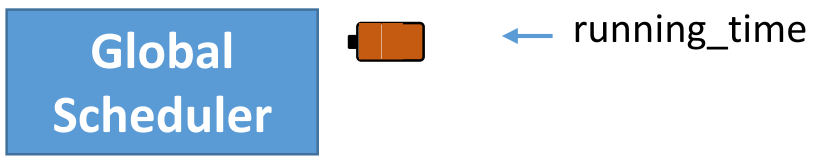


At the end of the period, *running_time* is refilled by *quota*.

Iron – Enforcement

Scheduler Integration

Reuse infrastructure from cgroup and Linux scheduler

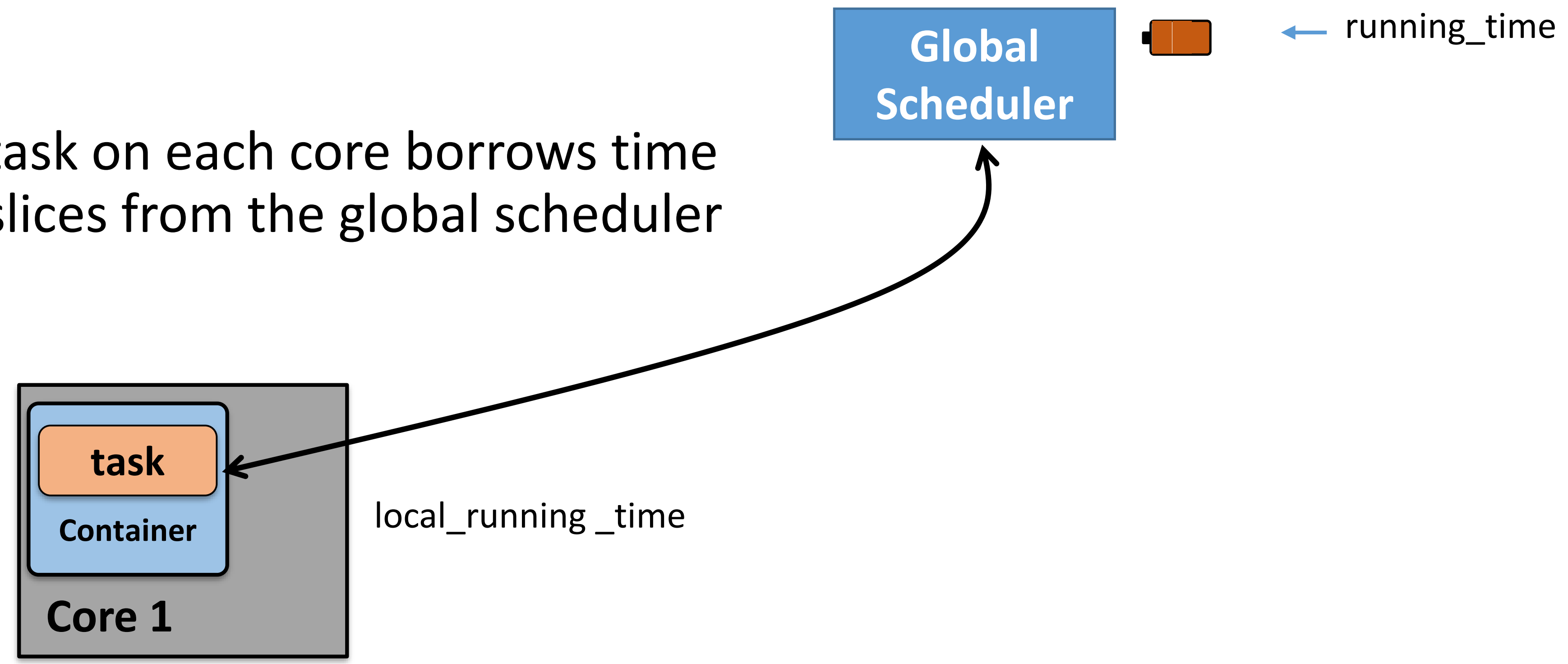


Iron – Enforcement

Scheduler Integration

Reuse infrastructure from cgroup and Linux scheduler

task on each core borrows time slices from the global scheduler

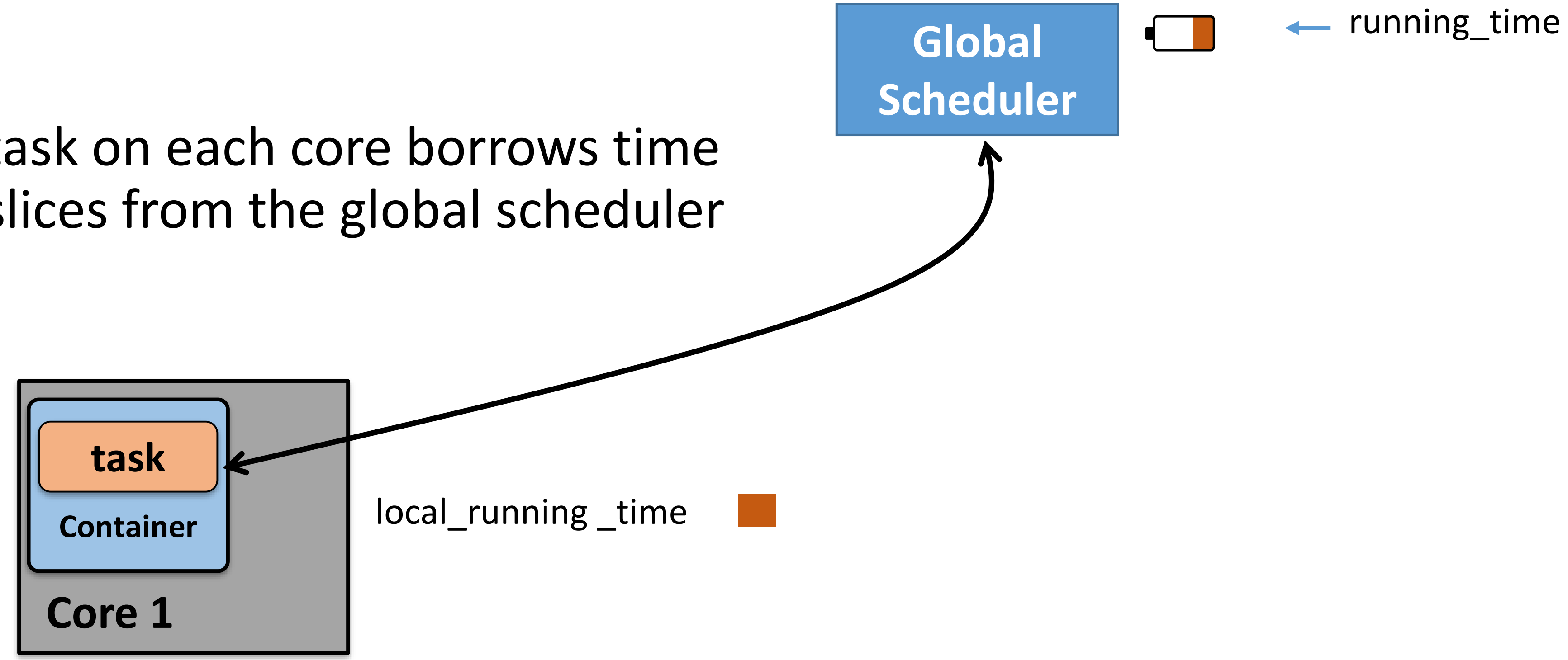


Iron – Enforcement

Scheduler Integration

Reuse infrastructure from cgroup and Linux scheduler

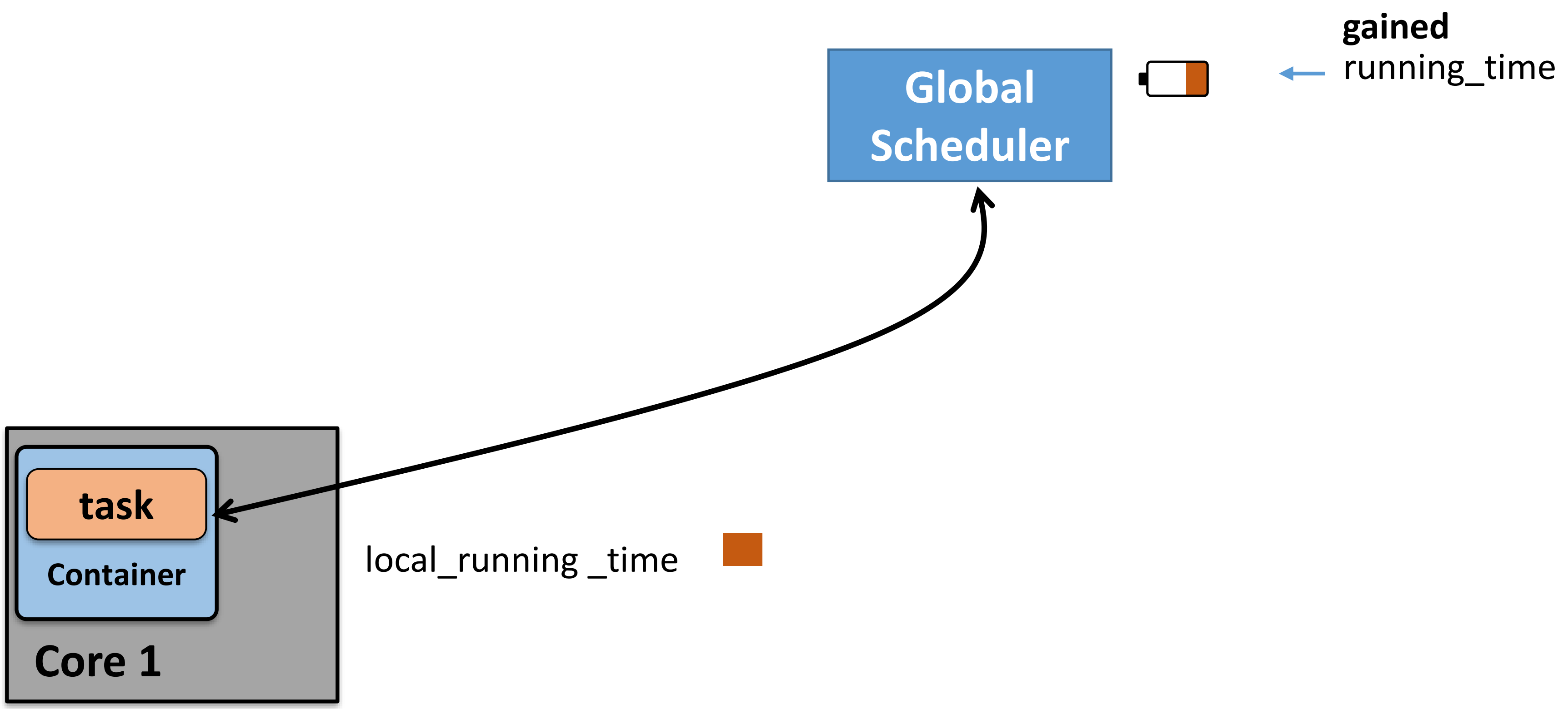
task on each core borrows time slices from the global scheduler



Iron – Enforcement

Scheduler Integration

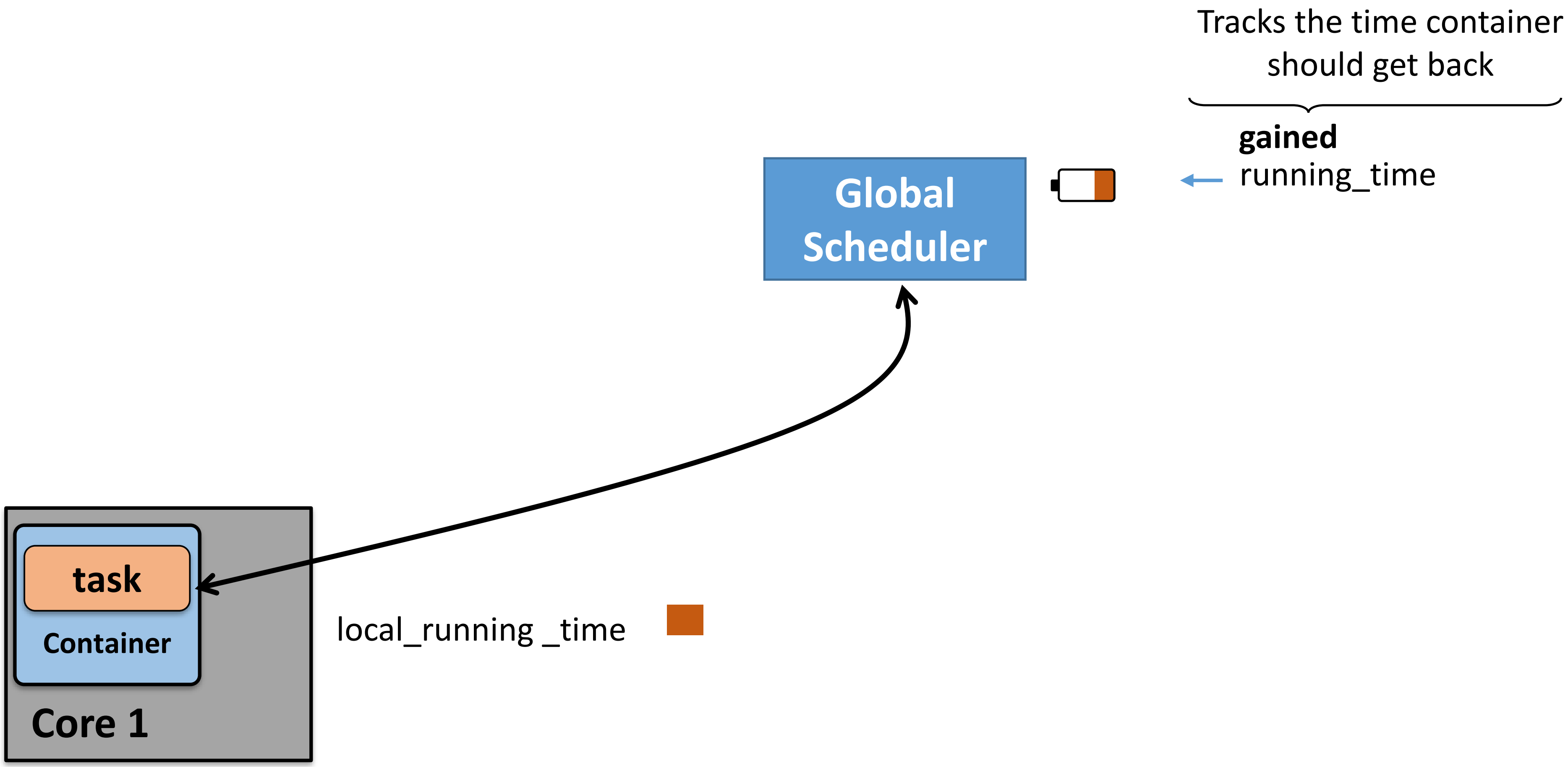
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

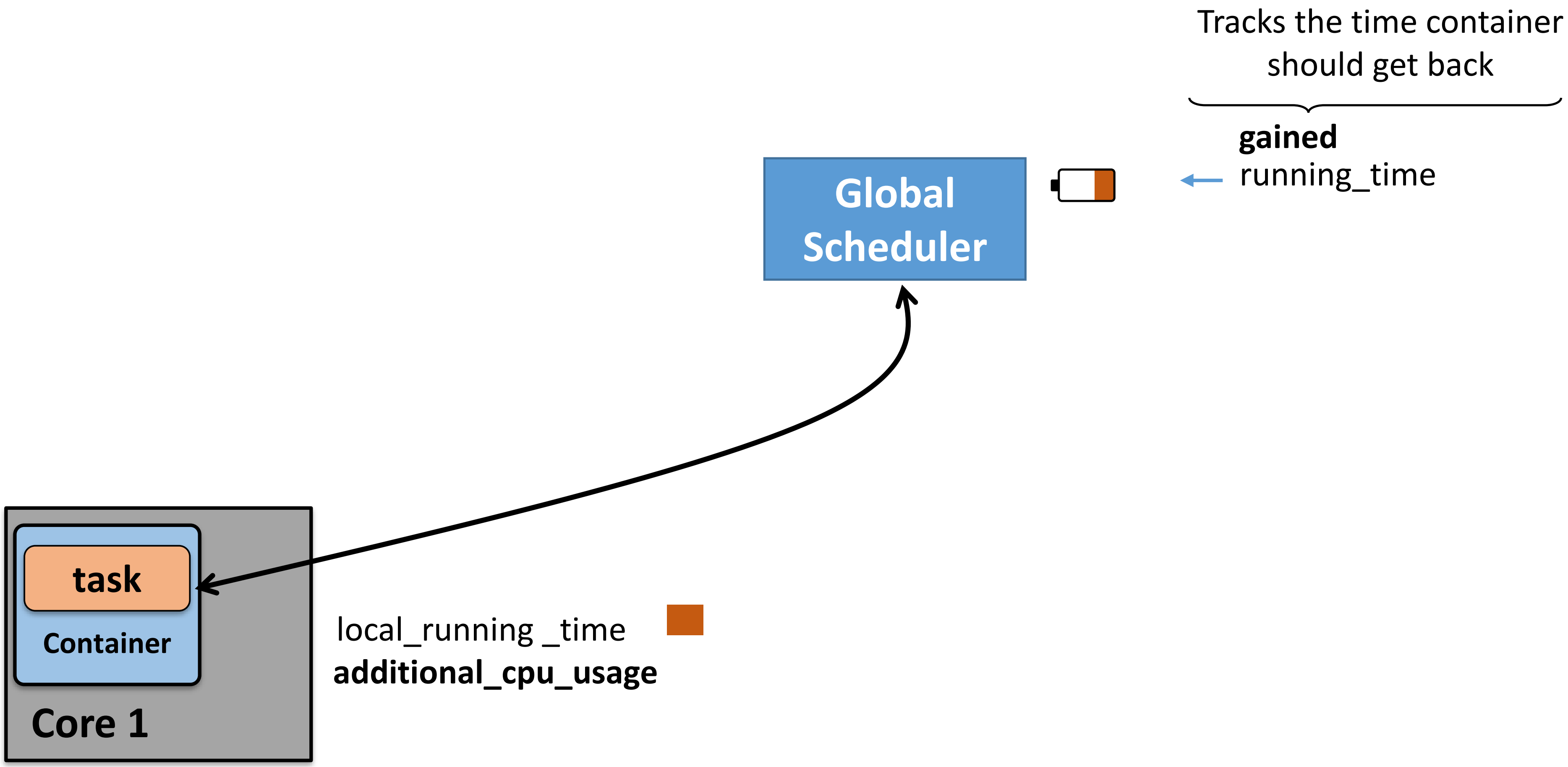
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

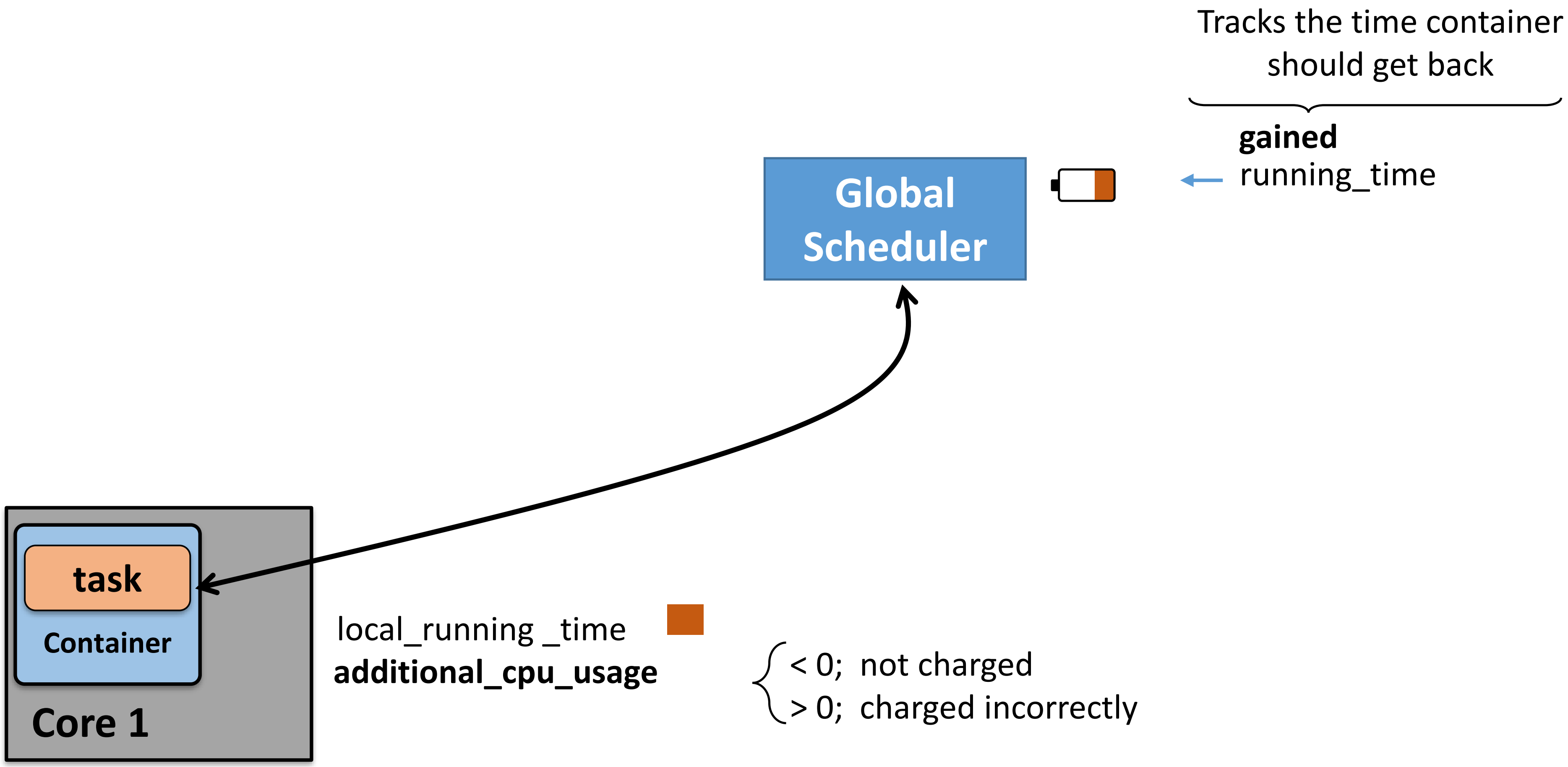
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

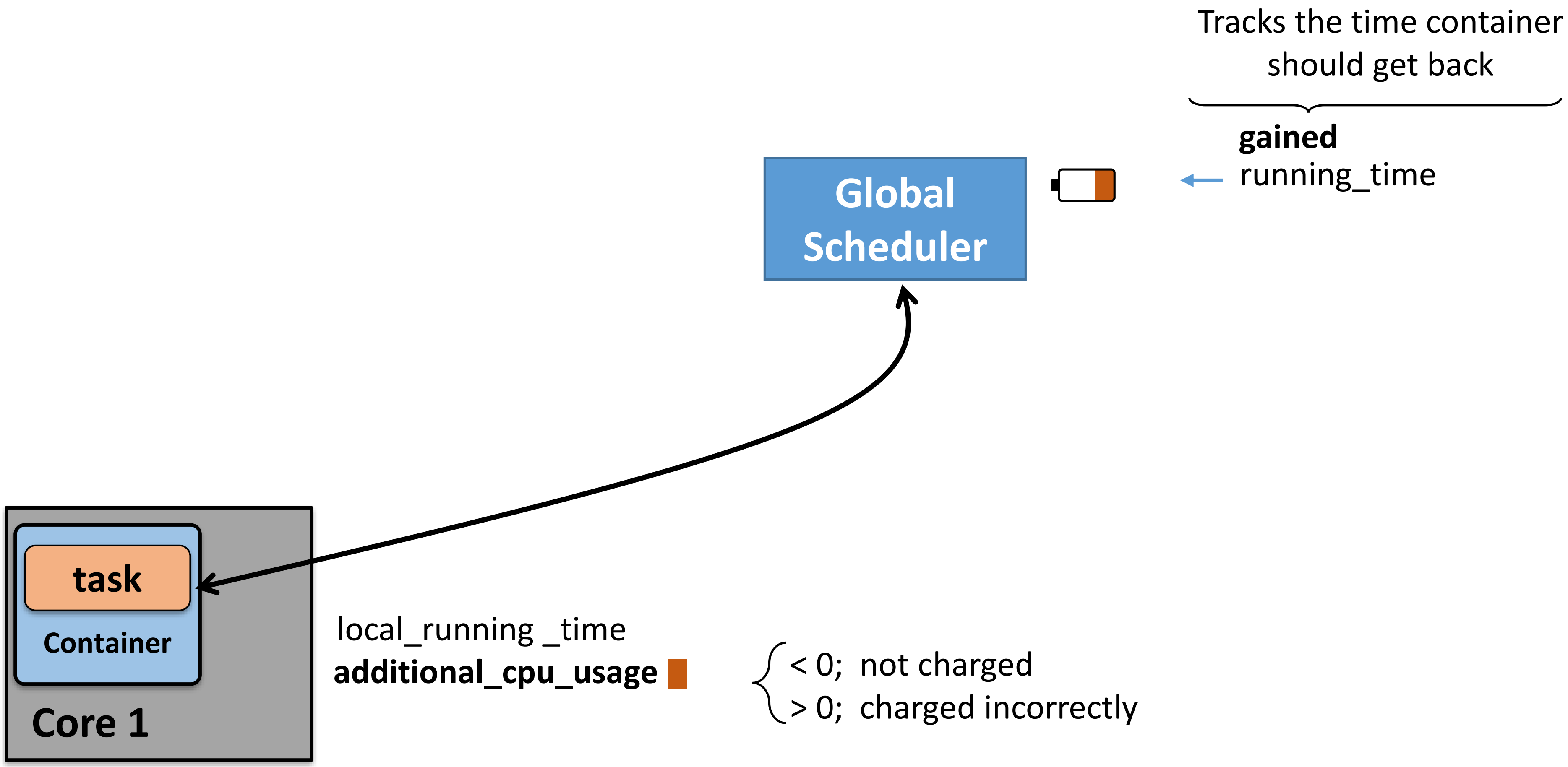
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

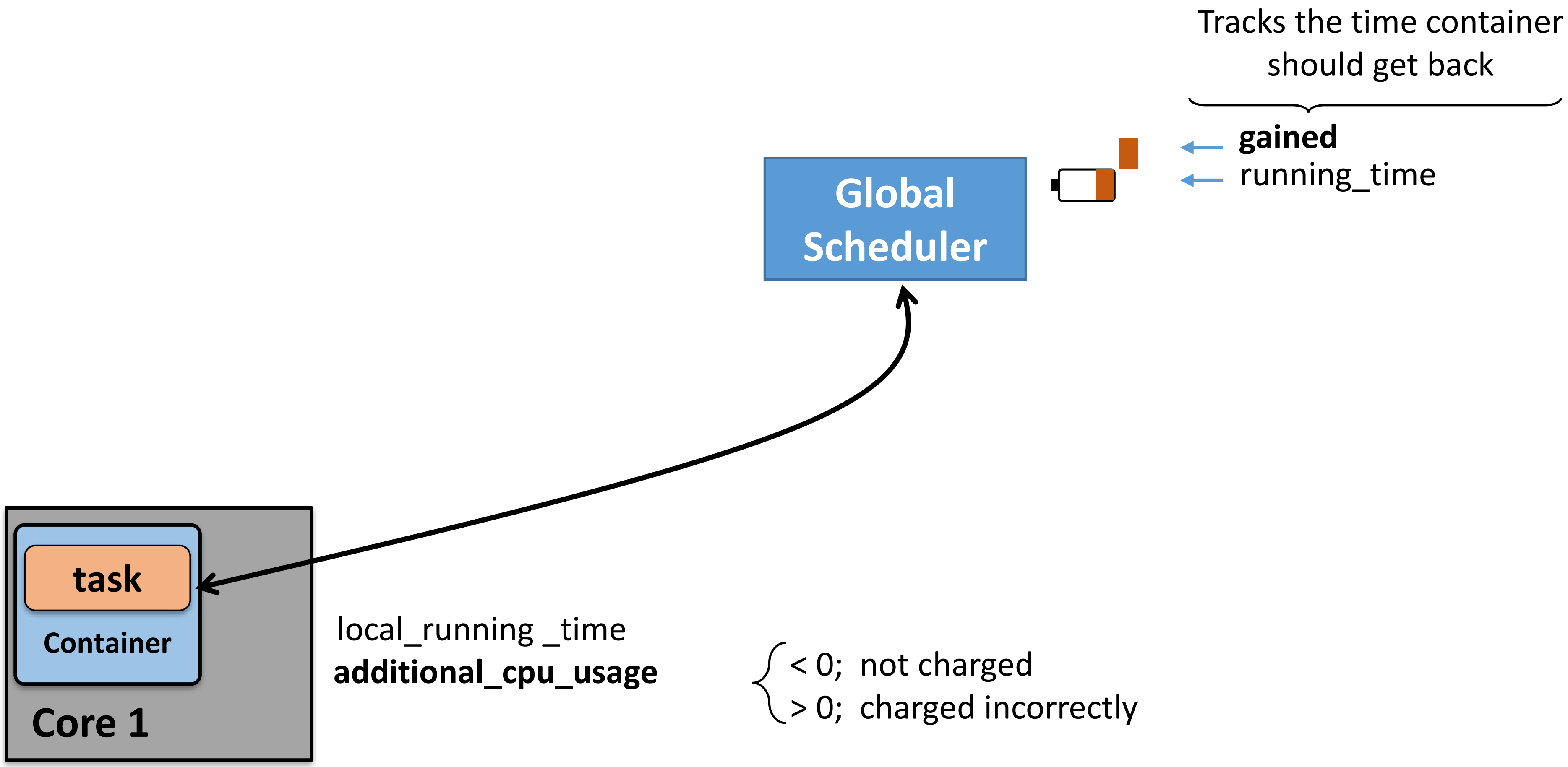
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

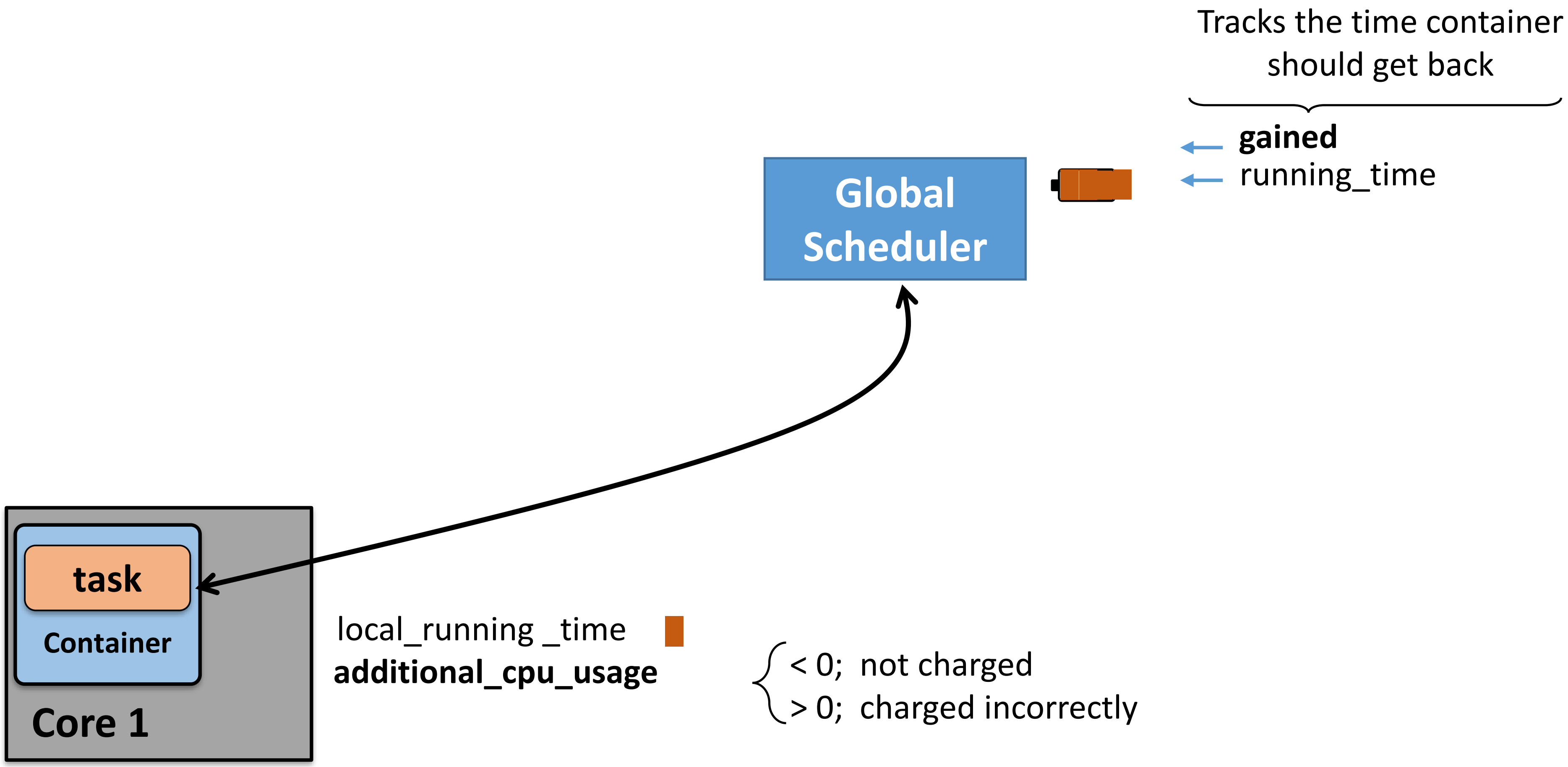
Reuse infrastructure from cgroup and Linux scheduler



Iron – Enforcement

Scheduler Integration

Reuse infrastructure from cgroup and Linux scheduler

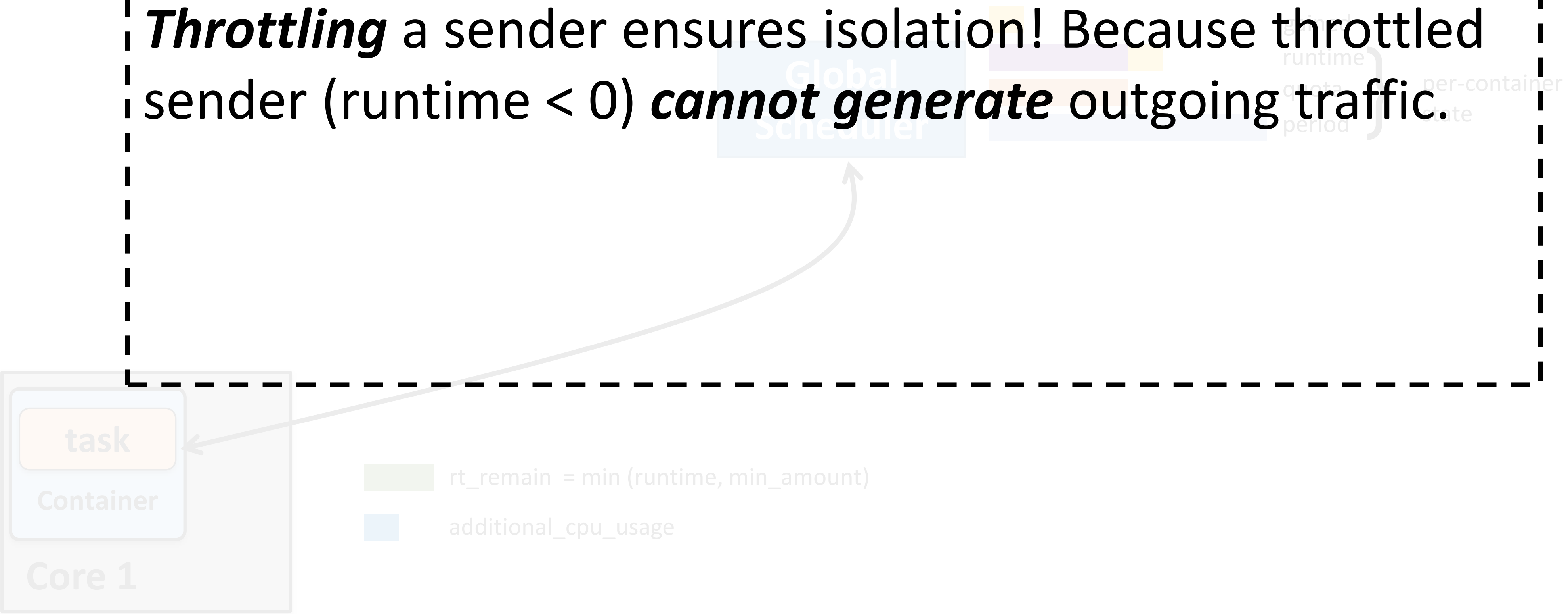


Iron – Enforcement

Scheduler Integration

Reuse infrastructure from cgroup and Linux scheduler

Throttling a sender ensures isolation! Because throttled sender (runtime < 0) ***cannot generate*** outgoing traffic.



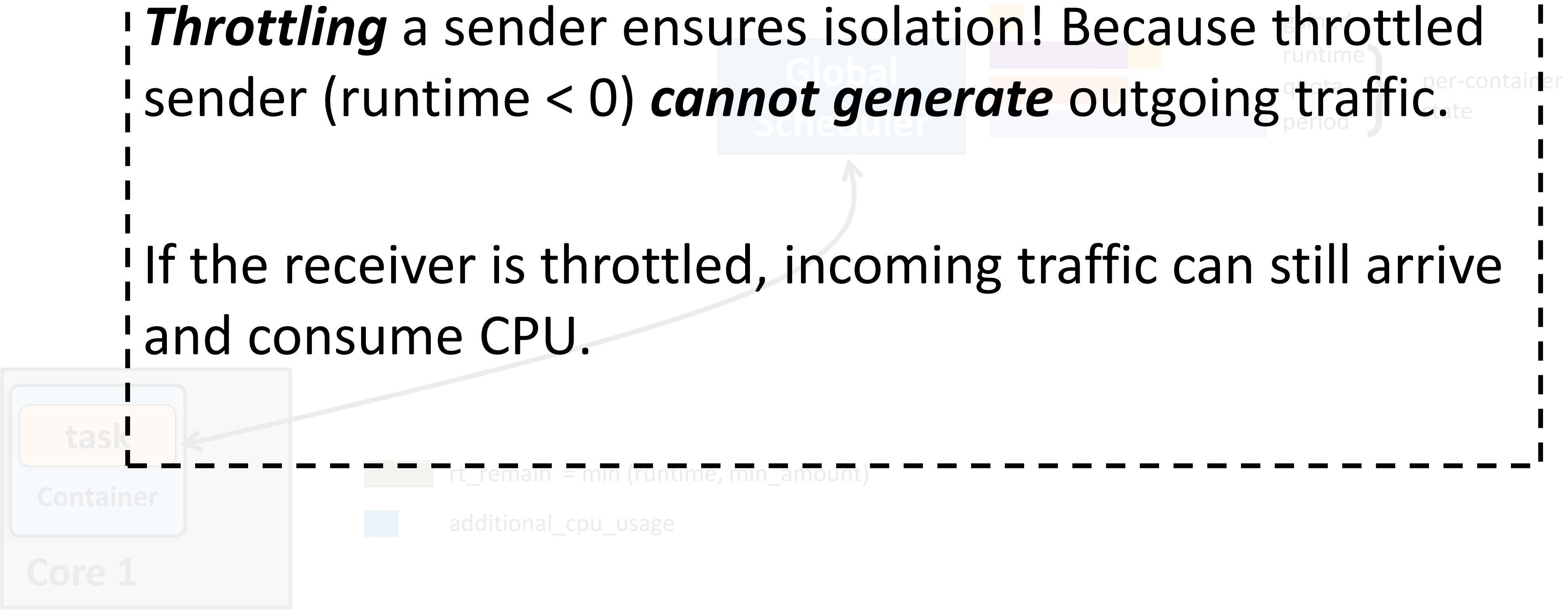
Iron – Enforcement

Scheduler Integration

Reuse infrastructure from cgroup and Linux scheduler

Throttling a sender ensures isolation! Because throttled sender (runtime < 0) ***cannot generate*** outgoing traffic.

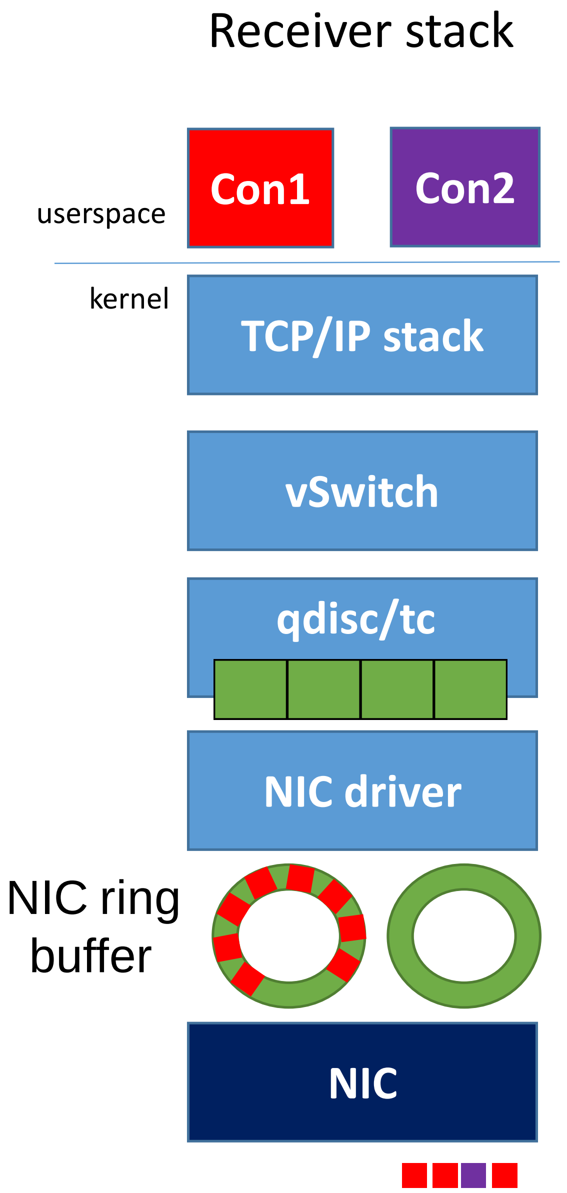
If the receiver is throttled, incoming traffic can still arrive and consume CPU.



Outline

- How and by how much is isolation broken
- Iron's Design
  - Accounting of per-packet processing cost
  - **Ensuring isolation via enforcement**
    - Integration with Linux scheduler
    - **Hardware-based packet dropping**
- Evaluation
  - Controlled workload
  - Realistic workload

Iron – Enforcement

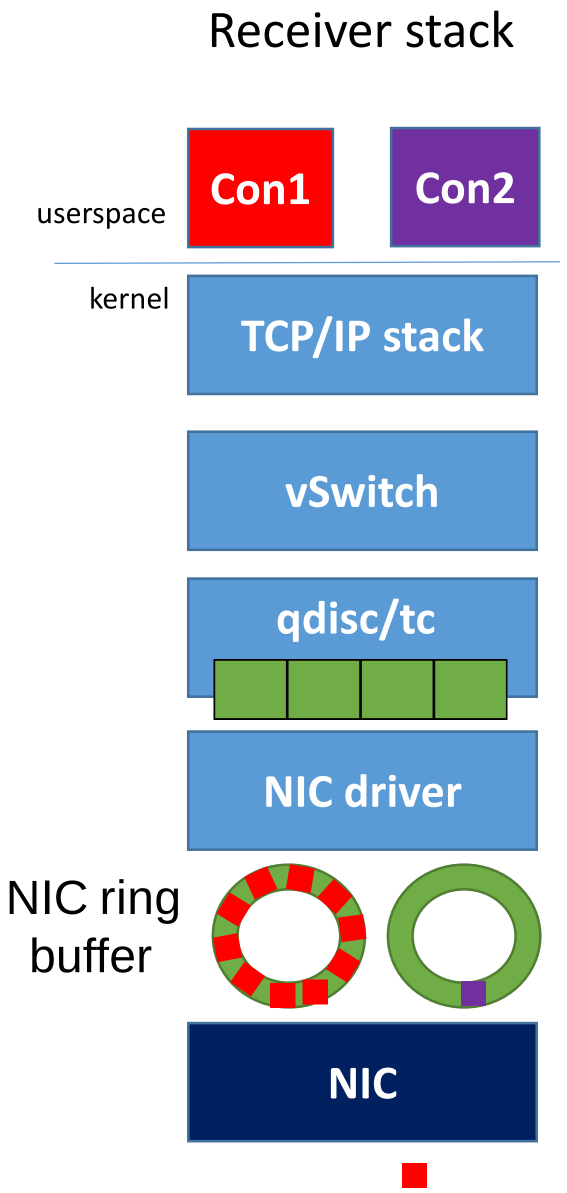


Dropping Packets

Modifies the Linux's polling mechanism (NAPI)

- Assigns a queue (ring buffer) to a container

Iron – Enforcement

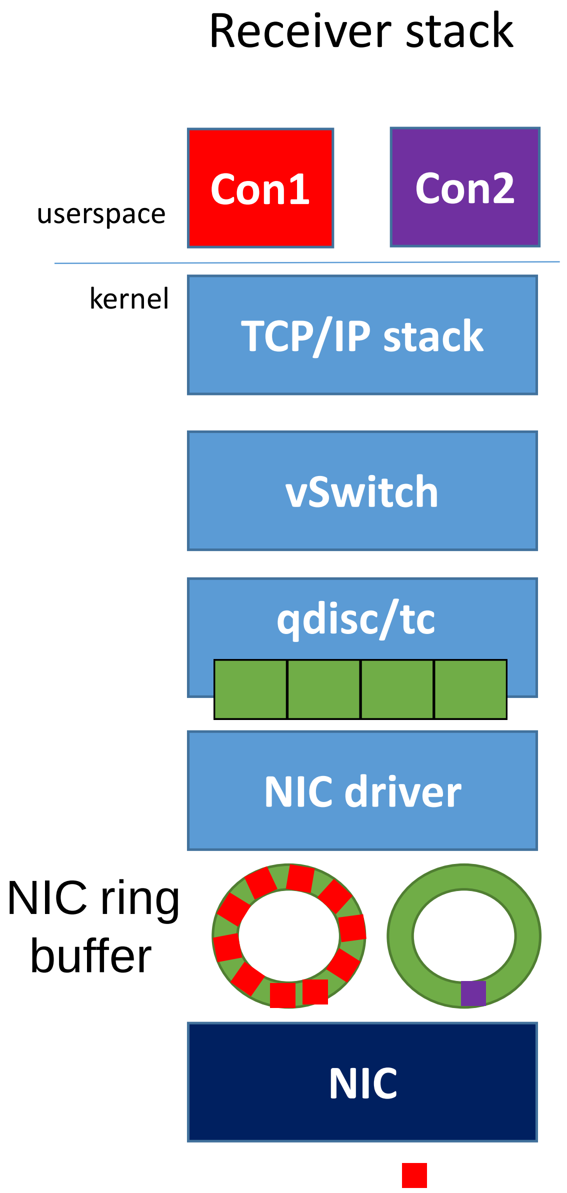


Dropping Packets

Modifies the Linux's polling mechanism (NAPI)

- Assigns a queue (ring buffer) to a container
- Iron strips the throttled queue from the polling list

Iron – Enforcement

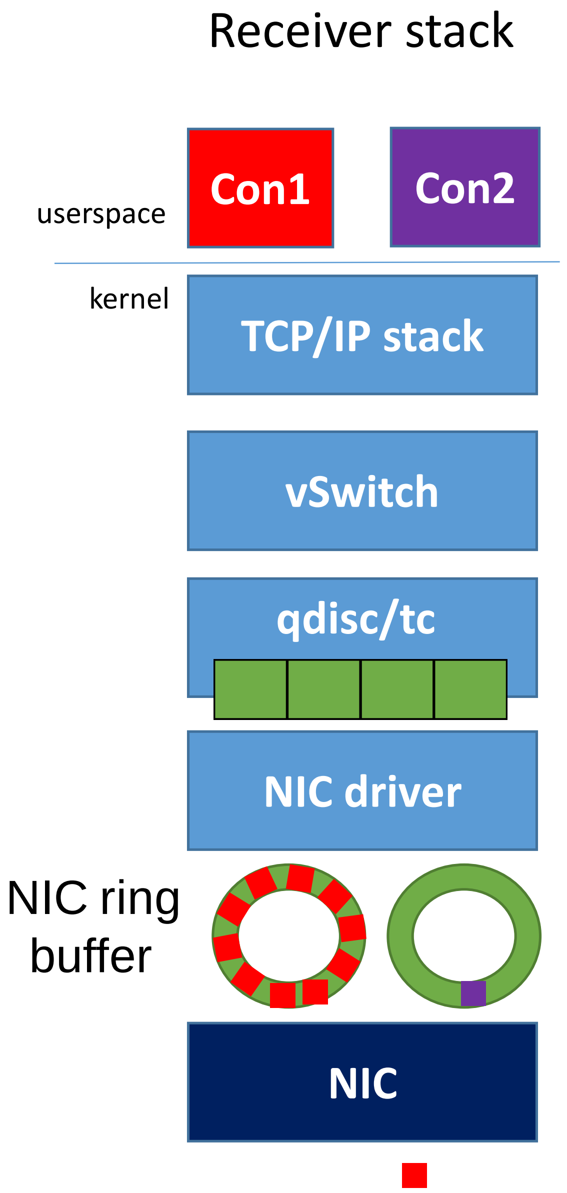


Dropping Packets

Modifies the Linux's polling mechanism (NAPI)

- Assigns a queue (ring buffer) to a container
- Iron strips the throttled queue from the polling list
  - From kernel's point of view, there are no more interrupts – no packets

Iron – Enforcement

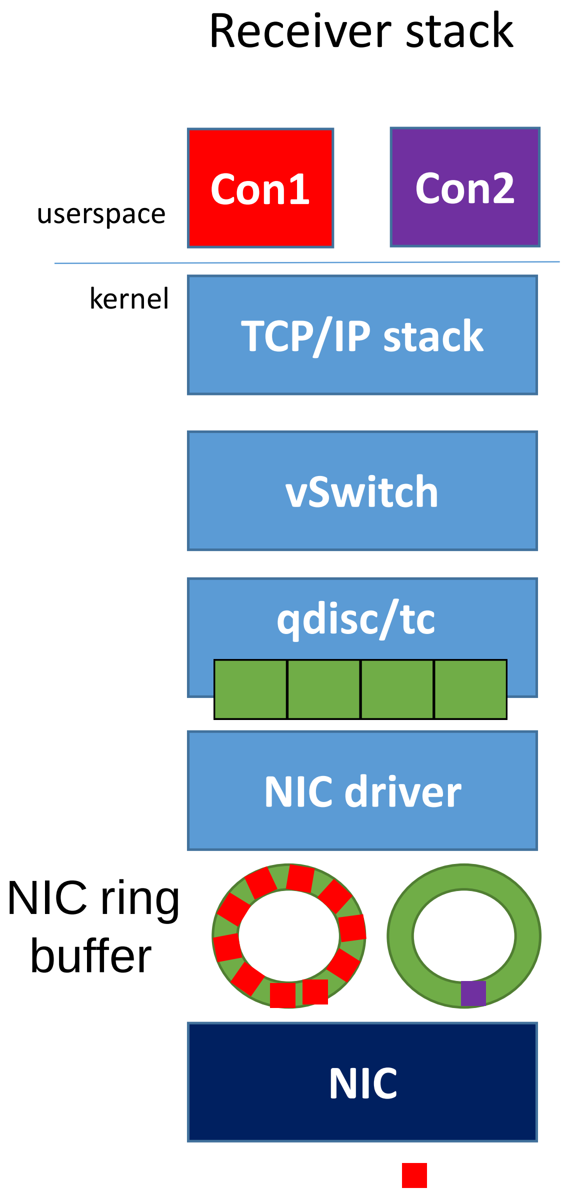


Dropping Packets

Modifies the Linux's polling mechanism (NAPI)

- Assigns a queue (ring buffer) to a container
- Iron strips the throttled queue from the polling list
  - From kernel's point of view, there are no more interrupts – no packets
  - From NIC's point of view, kernel is busy and is not polling packets from the queue, so it stays in the polling mode.

Iron – Enforcement

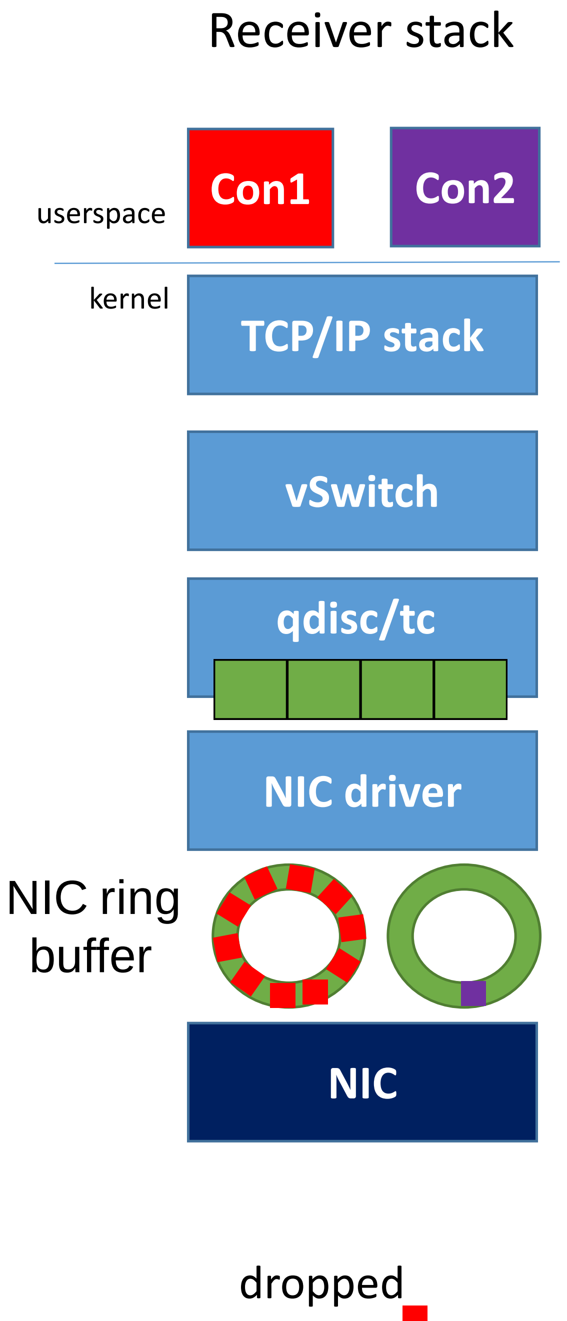


Dropping Packets

Modifies the Linux's polling mechanism (NAPI)

- Assigns a queue (ring buffer) to a container
- Iron strips the throttled queue from the polling list
  - From kernel's point of view, there are no more interrupts – no packets
  - From NIC's point of view, kernel is busy and is not polling packets from the queue, so it stays in the polling mode.
  - If a new packet arrives and the ring buffer is full, it gets dropped

Iron – Enforcement



Dropping Packets

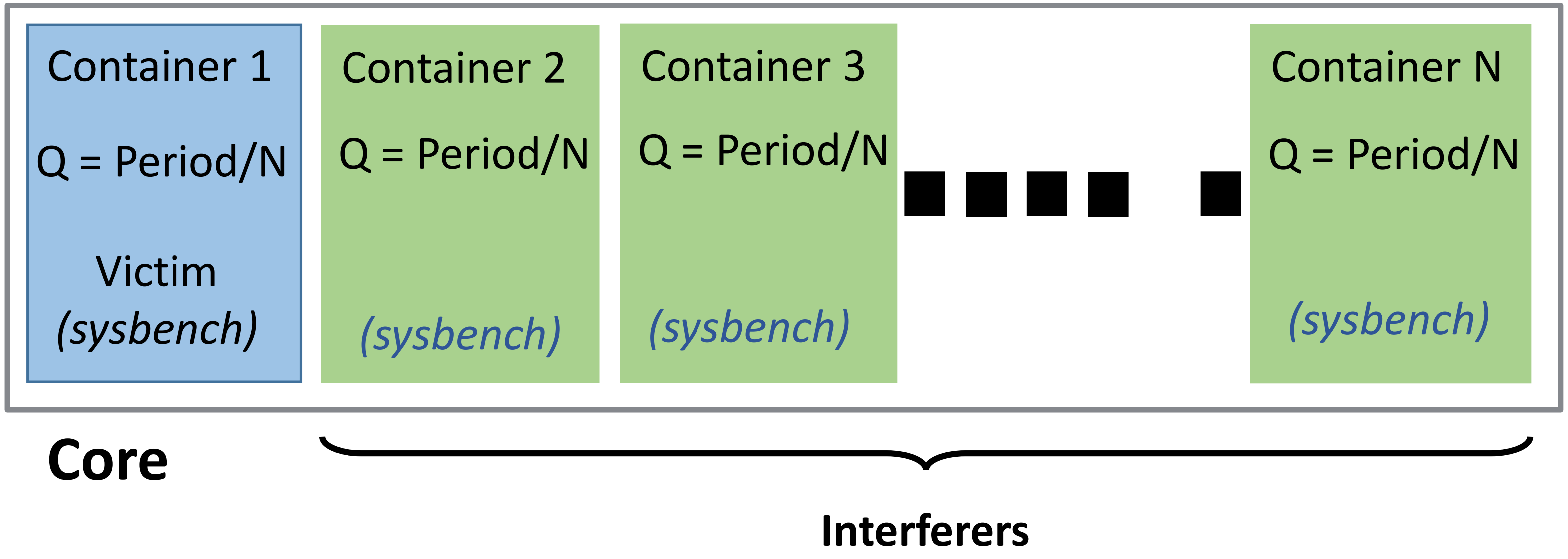
Modifies the Linux's polling mechanism (NAPI)

- Assigns a queue (ring buffer) to a container
- Iron strips the throttled queue from the polling list
  - From kernel's point of view, there are no more interrupts – no packets
  - From NIC's point of view, kernel is busy and is not polling packets from the queue, so it stays in the polling mode.
  - If a new packet arrives and the ring buffer is full, it gets dropped

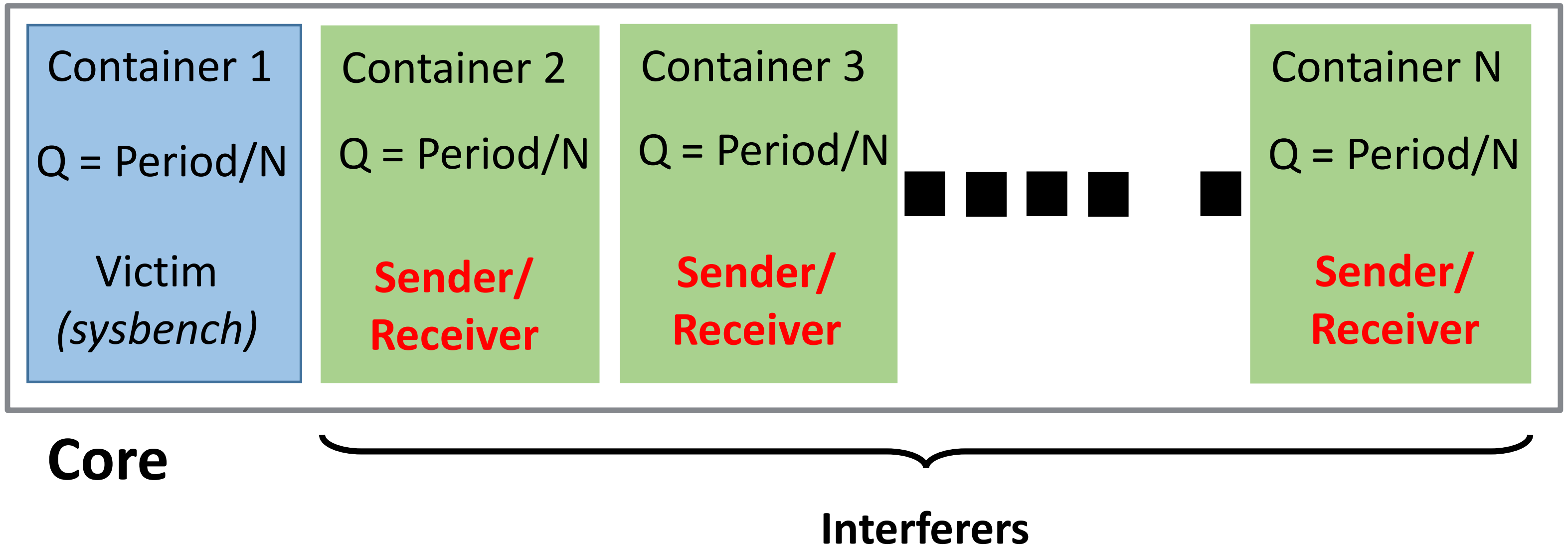
Outline

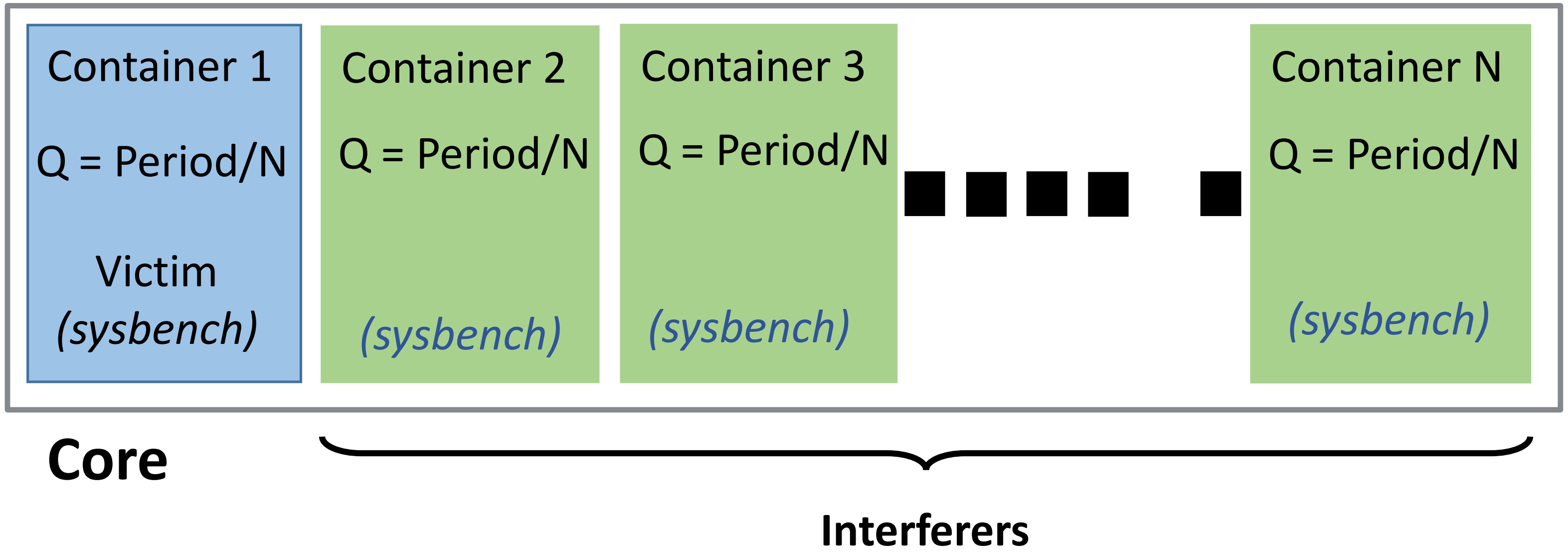
- How and by how much is isolation broken
- Iron's design
  - Accounting of per-packet processing cost
  - Ensuring isolation via enforcement
    - Integration with Linux scheduler
    - Hardware-based packet dropping
- **Evaluation**
  - **Controlled workload**
  - Realistic workload

Experiment Setup

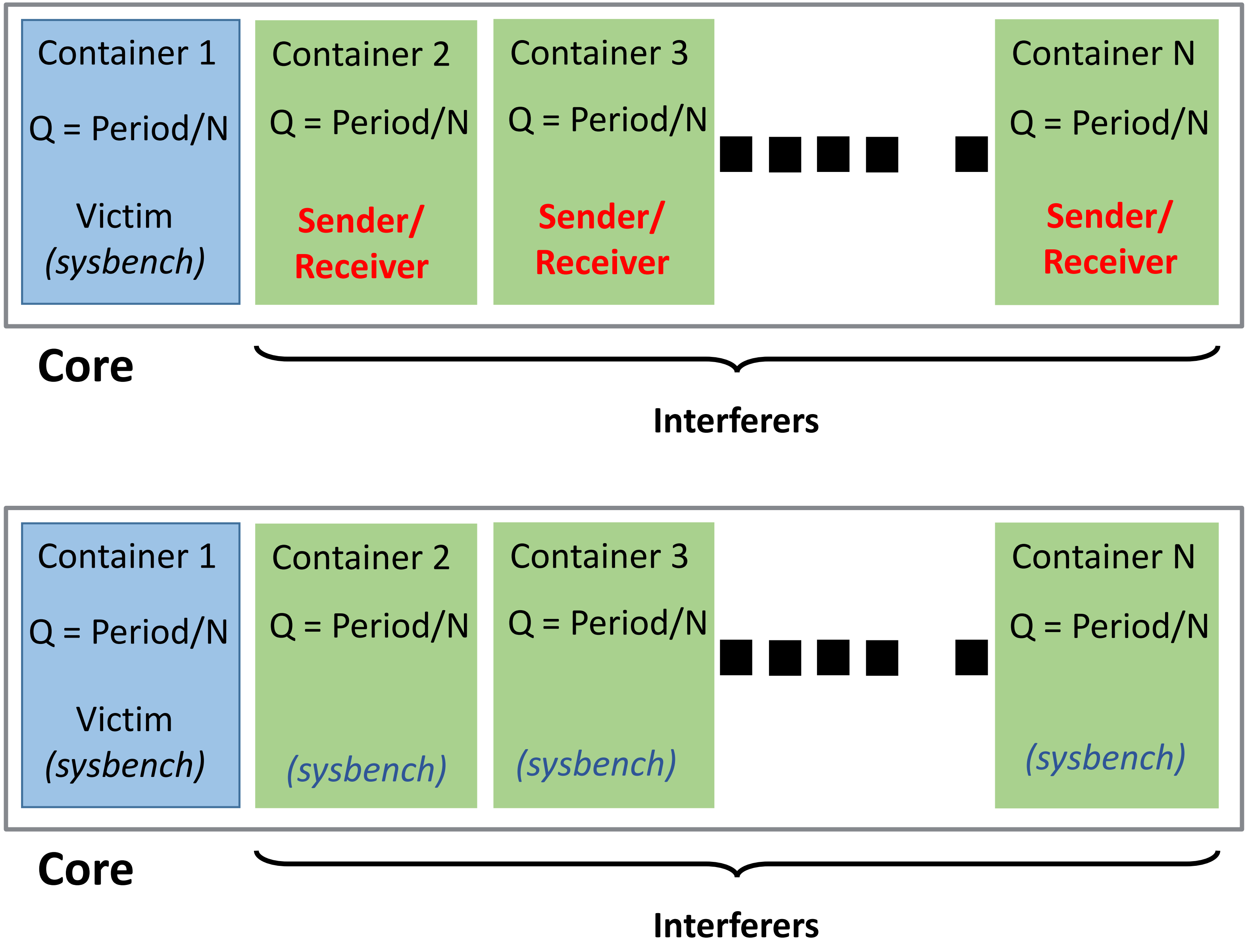


Experiment Setup





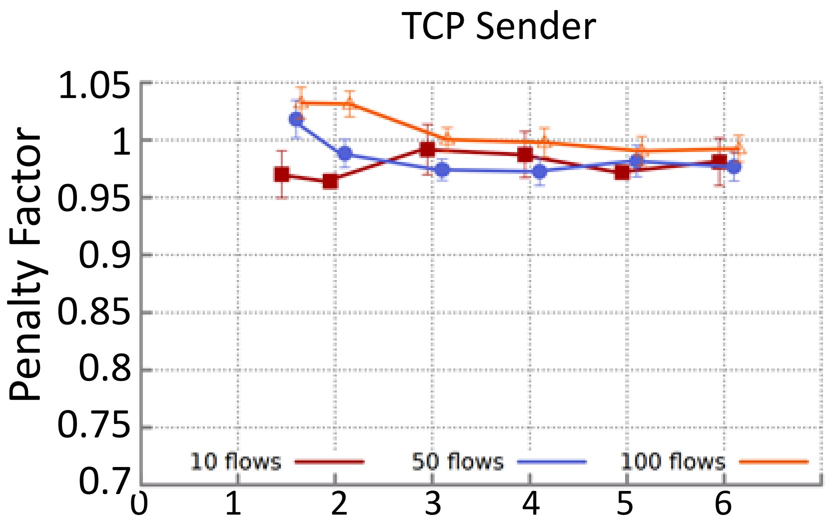
Experiment Setup



$$\textit{Penalty factor} = \frac{\textit{Time that victim takes when competing with traffic}}{\textit{Time that victim takes when competing with sysbench}}$$

Sender Interference With Iron

Penalty factor = *Time that victim takes when competing with traffic* / *Time that victim takes when competing with sysbench*

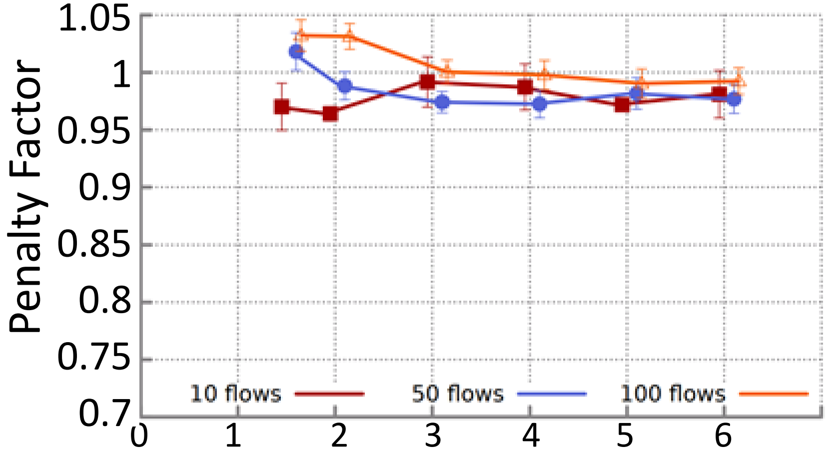


Traffic rate (Gbps)
2 containers per core

Sender Interference With Iron

Penalty factor = **_Time that victim takes when competing with traffic_** / **_Time that victim takes when competing with sysbench_**

TCP Sender

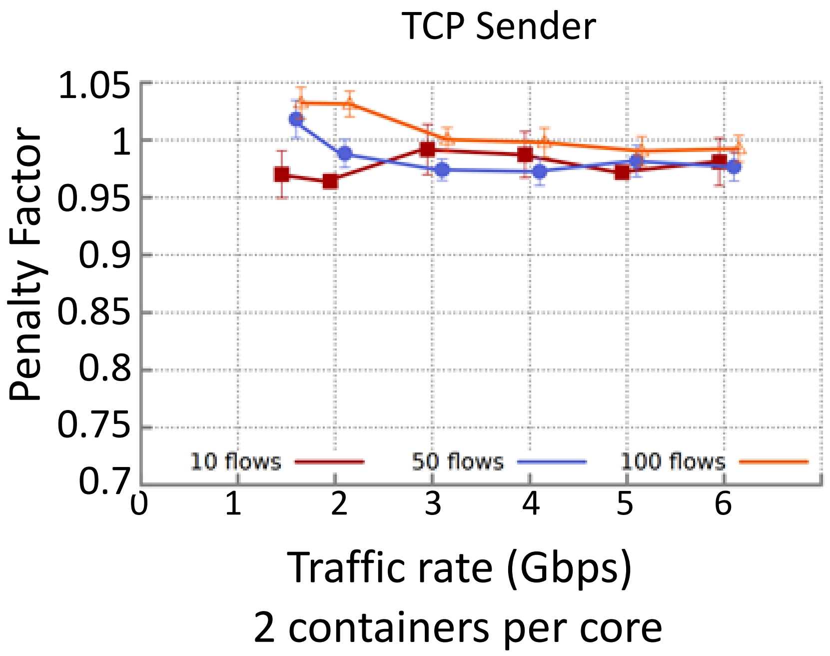


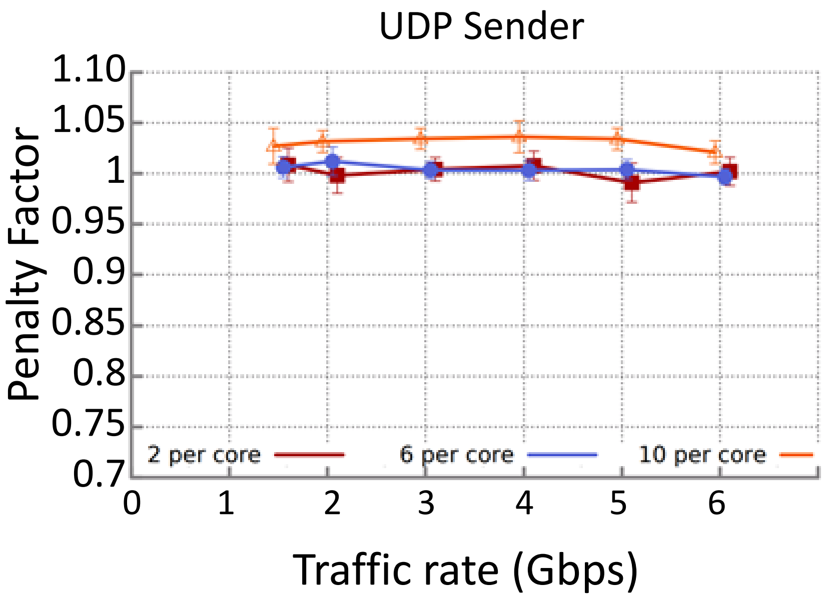
Traffic rate (Gbps)
2 containers per core

Penalty factor remains below ***1.04***, significant decrease from ***1.85***

Sender Interference With Iron

Penalty factor = *Time that victim takes when competing with traffic* / *Time that victim takes when competing with sysbench*





Penalty factor remains below ***1.04***, significant decrease from ***1.85***

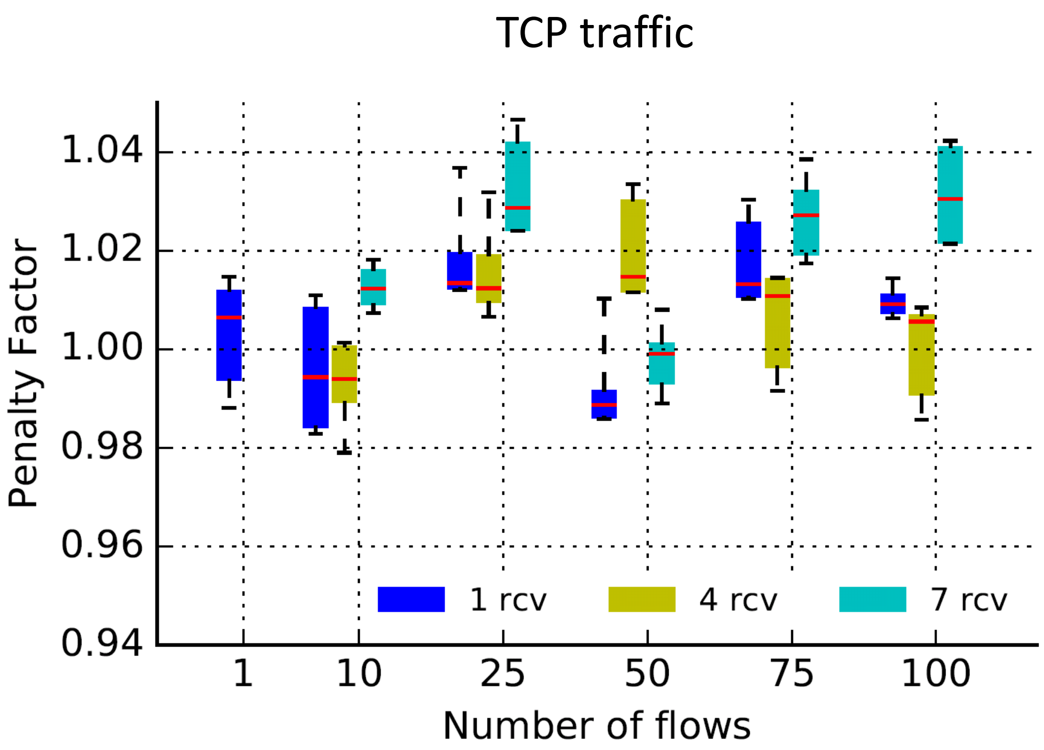
Penalty factor remains below ***1.04***, significant decrease from ***1.18***

Receiver Interference With Iron

$$\textit{Penalty factor} = \frac{\textit{Time that victim takes when competing with traffic}}{\textit{Time that victim takes when competing with sysbench}}$$

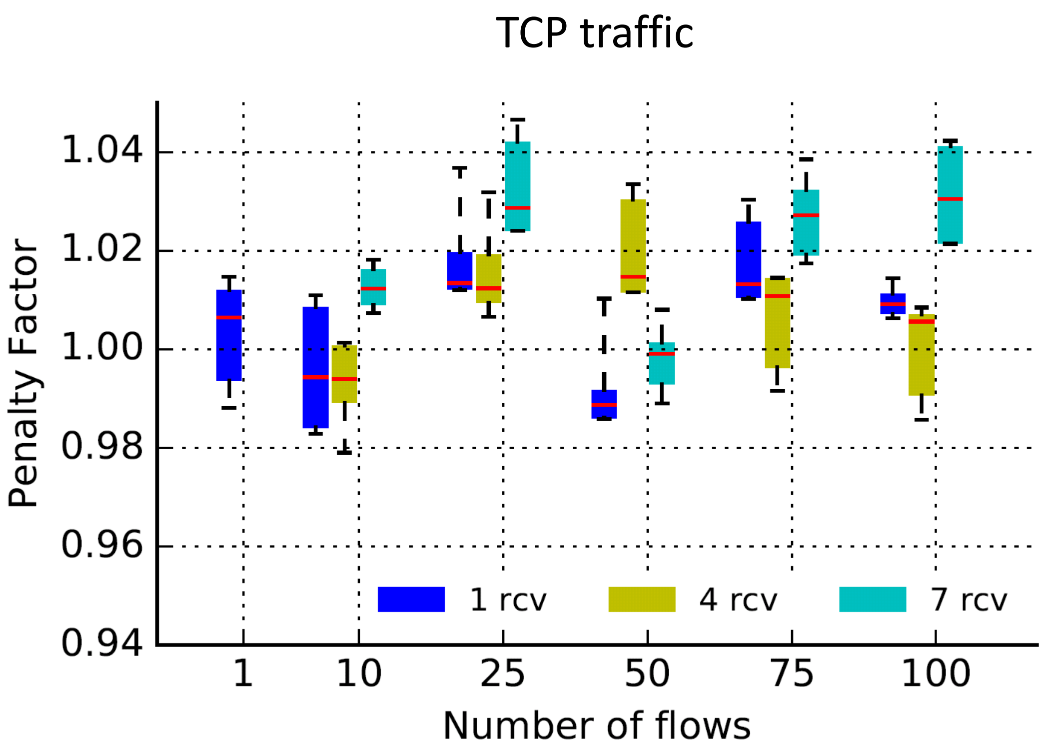
Receiver Interference With Iron

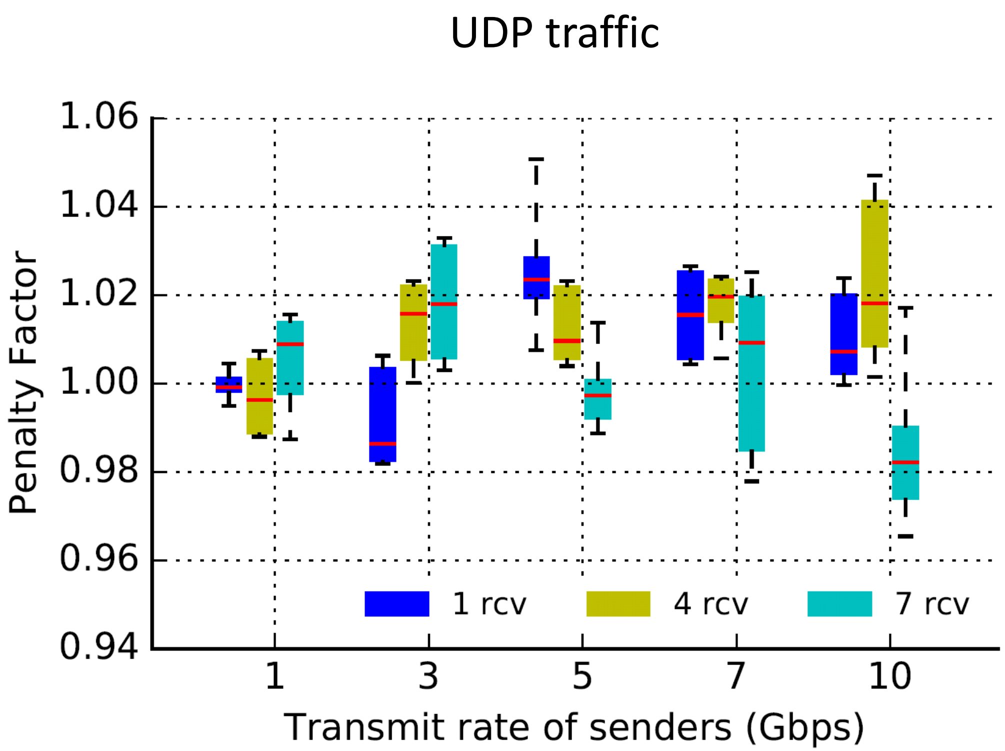
Penalty factor = *Time that victim takes when competing with traffic* / *Time that victim takes when competing with sysbench*



Receiver Interference With Iron

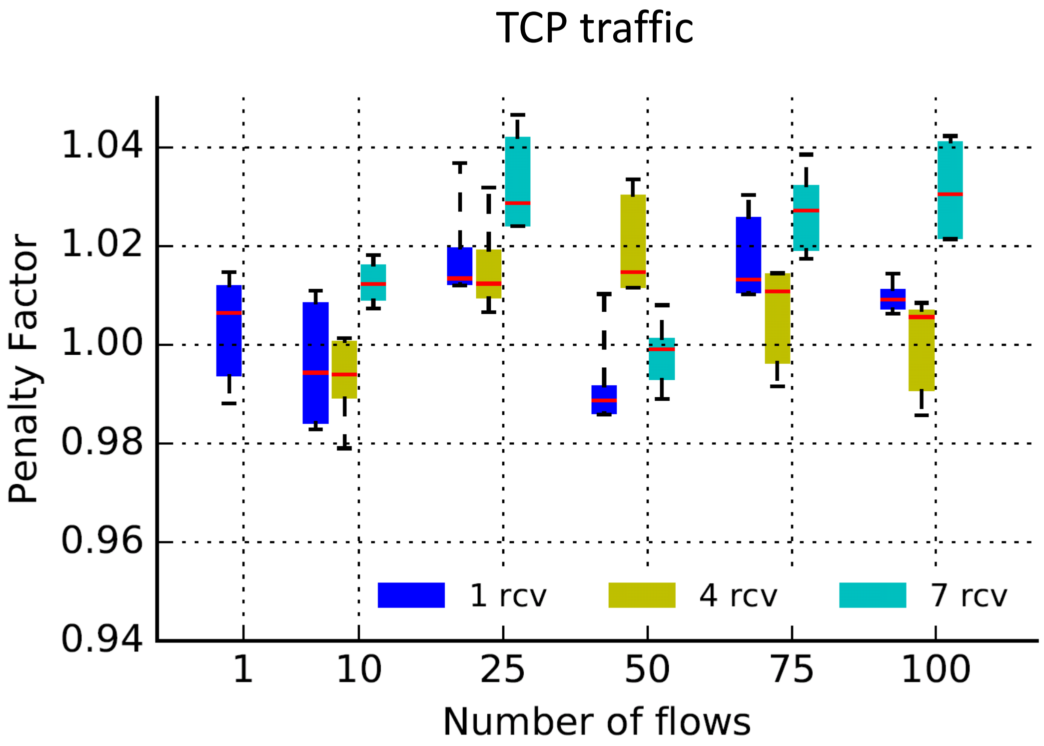
$$\text{Penalty factor} = \frac{\text{Time that victim takes when competing with traffic}}{\text{Time that victim takes when competing with sysbench}}$$

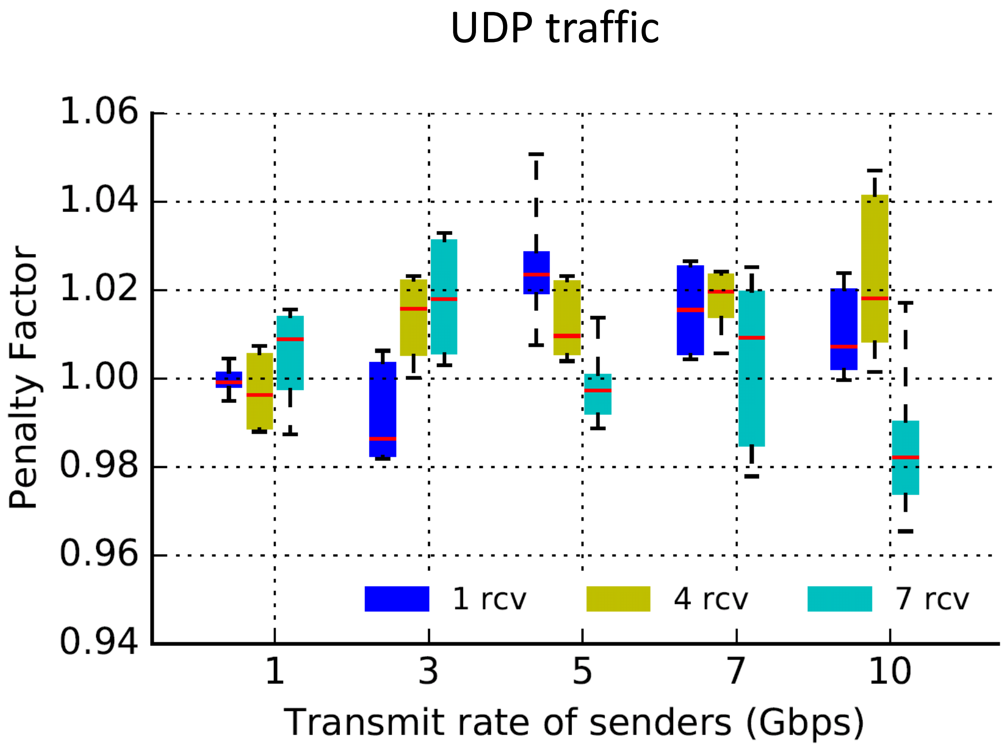




Receiver Interference With Iron

Penalty factor = ***Time that victim takes when competing with traffic*** / ***Time that victim takes when competing with sysbench***





Penalty factor never exceeds ***1.05,*** significant decrease from ***6 for TCP*** *and* ***4.45 for UDP***

Outline

- How and by how much is isolation broken
- Iron's Design
  - Accounting of per-packet processing cost
  - Ensuring isolation via enforcement
    - Integration with Linux scheduler
    - Hardware-based packet dropping
- **Evaluation**
  - Controlled workload
  - **Realistic workload**

Impact on Big Data Applications

Setup
- 48 containers spread over 6 machines
- Each job runs over 24 containers

Impact on Big Data Applications

Setup

- 48 containers spread over 6 machines
- Each job runs over 24 containers

MapReduce jobs as victim:

- **wordcount**: counts word frequency
- **pi**: computes the value of pi
- **grep**: searches for a given word

Trace based Interferer:

- Shuffle phase of TeraSort job with

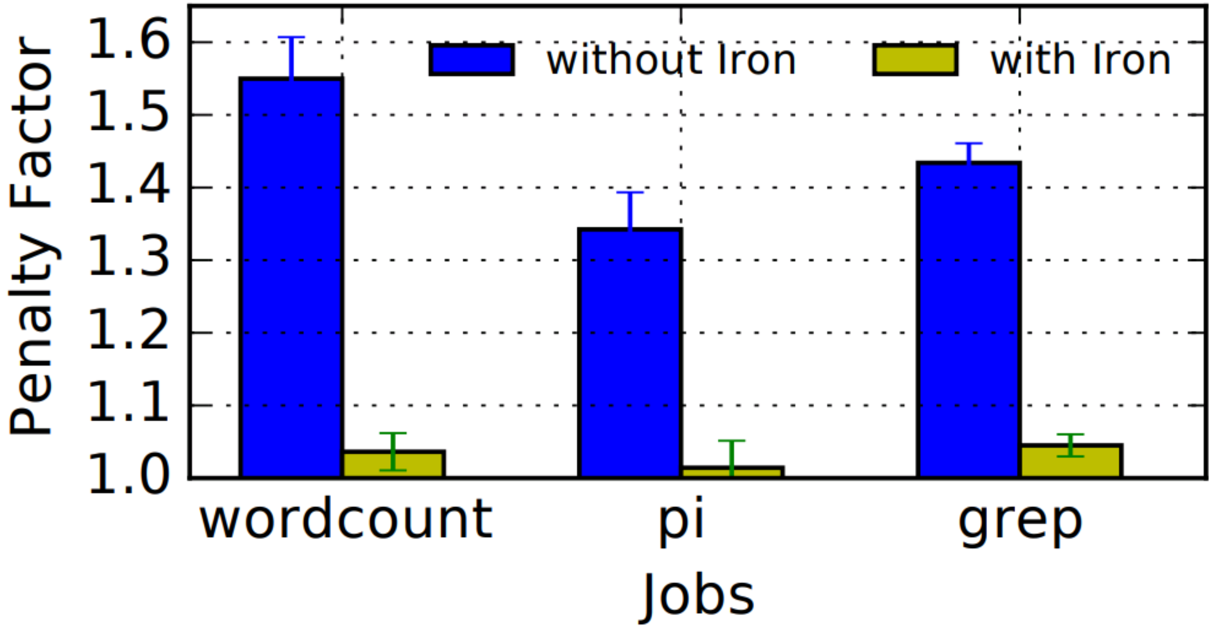
115GB input file

Impact on Big Data Applications

Setup
- 48 containers spread over 6 machines
- Each job runs over 24 containers

MapReduce jobs as victim:
- **wordcount**: counts word frequency
- **pi**: computes the value of pi
- **grep**: searches for a given word

Trace based Interferer:
- Shuffle phase of TeraSort job with 115GB input file



Impact on Big Data Applications

Setup

- 48 containers spread over 6 machines
- Each job runs over 24 containers

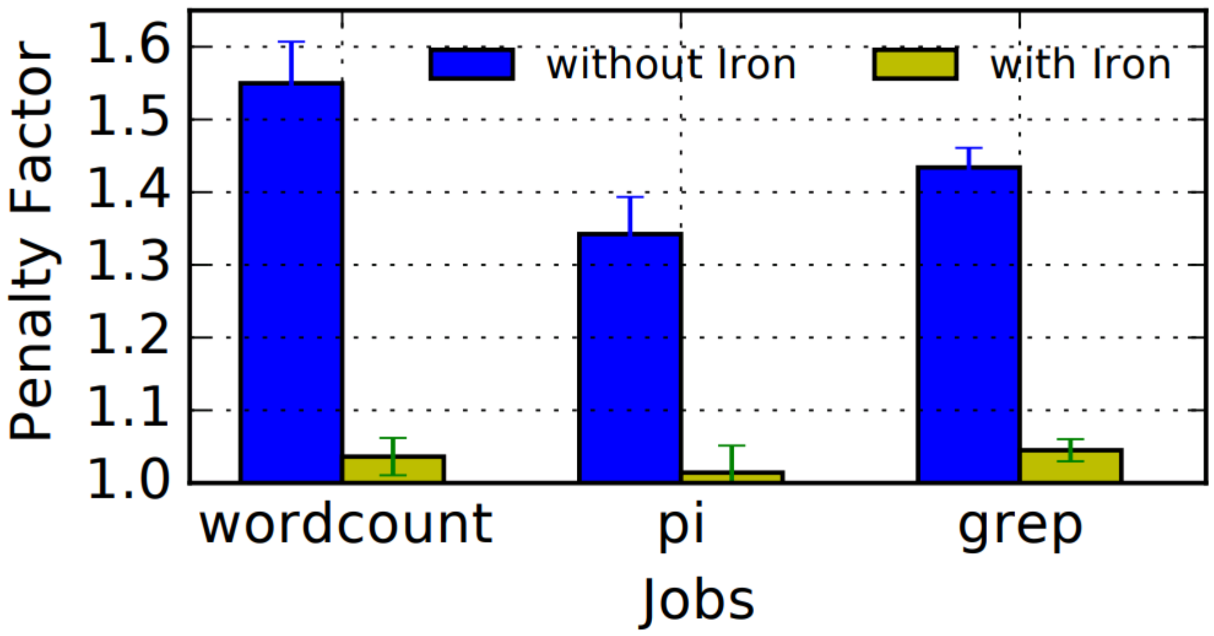
MapReduce jobs as victim:

- **wordcount**: counts word frequency
- **pi**: computes the value of pi
- **grep**: searches for a given word

Trace based Interferer:

- Shuffle phase of TeraSort job with

115GB input file



Penalty factor never exceeds ***1.04***

Summary

- Evaluated the interference caused by network-based containers.
- Provided hardened isolation for network-based processing in containerized environment.
- Ensures accurate accounting of the time spent processing network traffic in softirq.
- Integrated with Linux scheduler with minimal changes.
- Novel packet dropping mechanism to limit the interference.