SIGCOMM 2024 – Best of CCR session – Sydney, Australia – 6th August 2024

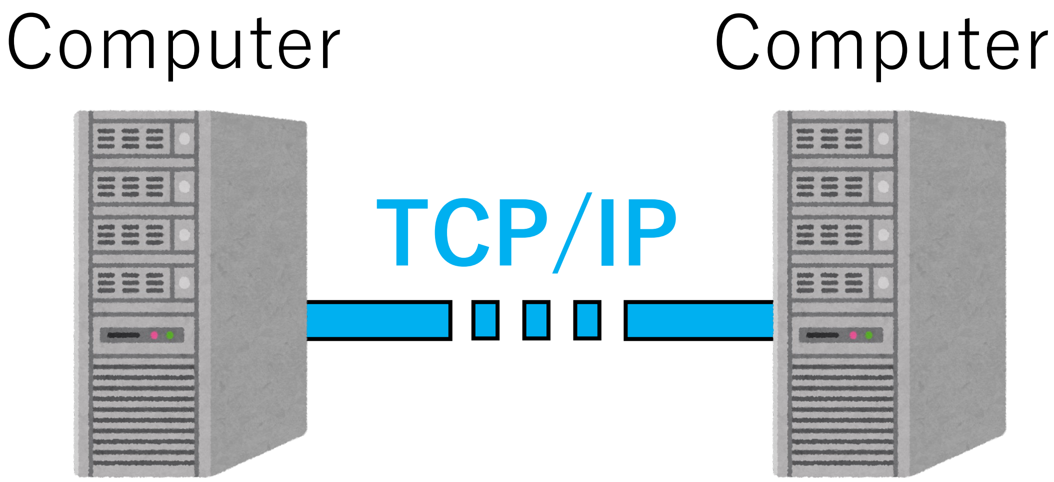
iip: an integratable TCP/IP stack

Kenichi Yasukata

IIJ
Internet Initiative Japan

TCP/IP and TCP/IP Stacks

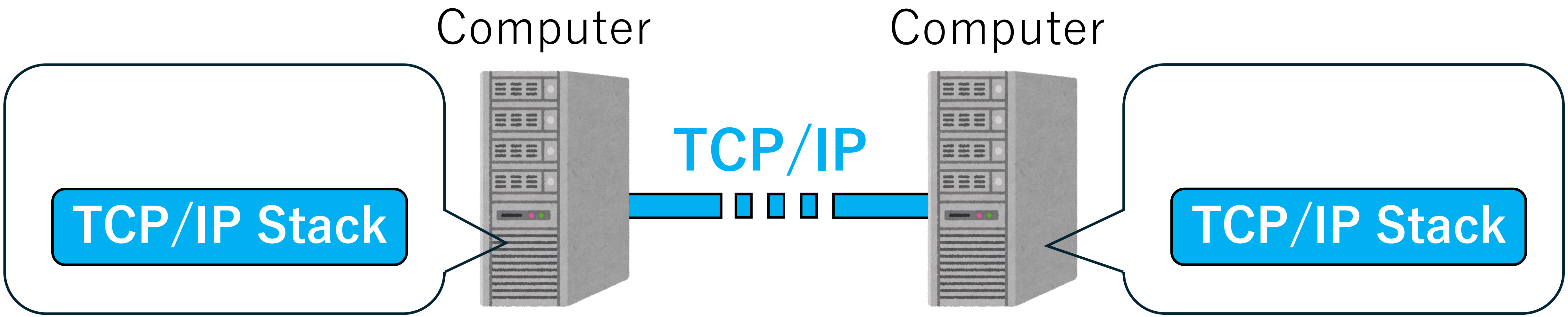
- TCP/IP is a standardized protocol suite commonly used for communication in computer networks



https://github.com/yasukata/iip

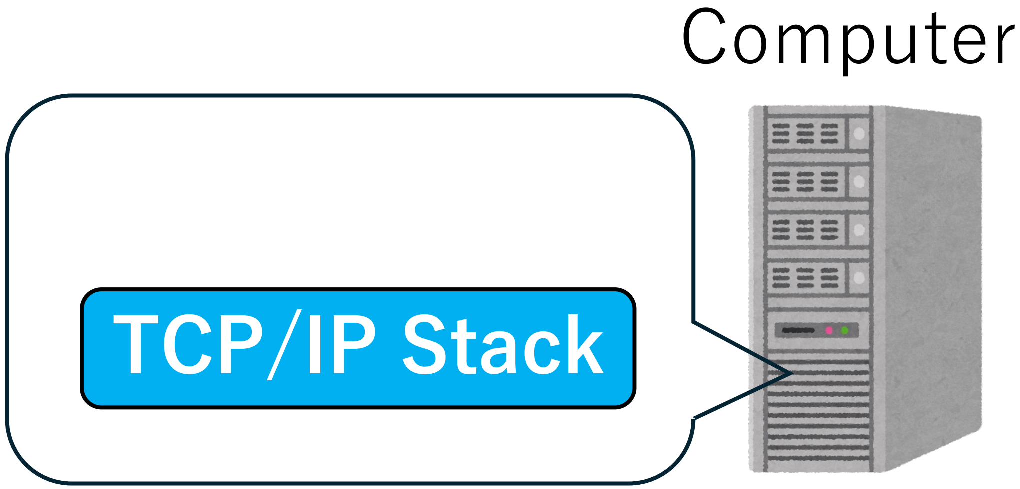
TCP/IP and TCP/IP Stacks

- TCP/IP is a standardized protocol suite commonly used for communication in computer networks
- TCP/IP stacks are typically software that implements procedures to comply with the TCP/IP standard



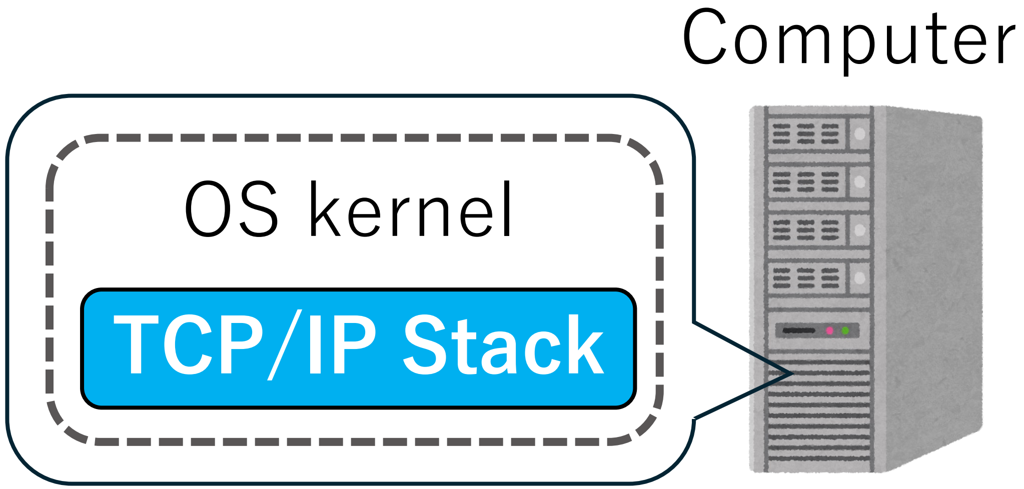
TCP/IP Stacks in Legacy OS Kernels

- TCP/IP stacks are typically maintained as part of OS kernels



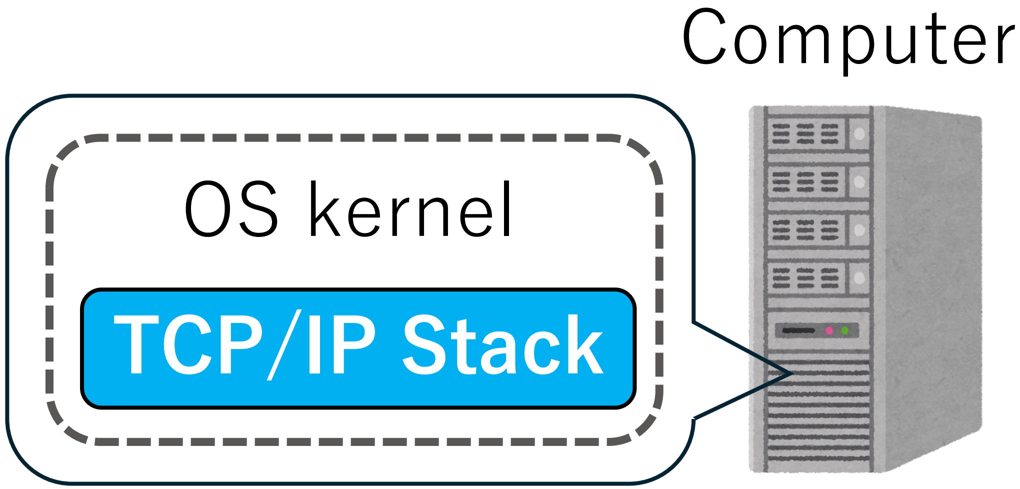
TCP/IP Stacks in Legacy OS Kernels

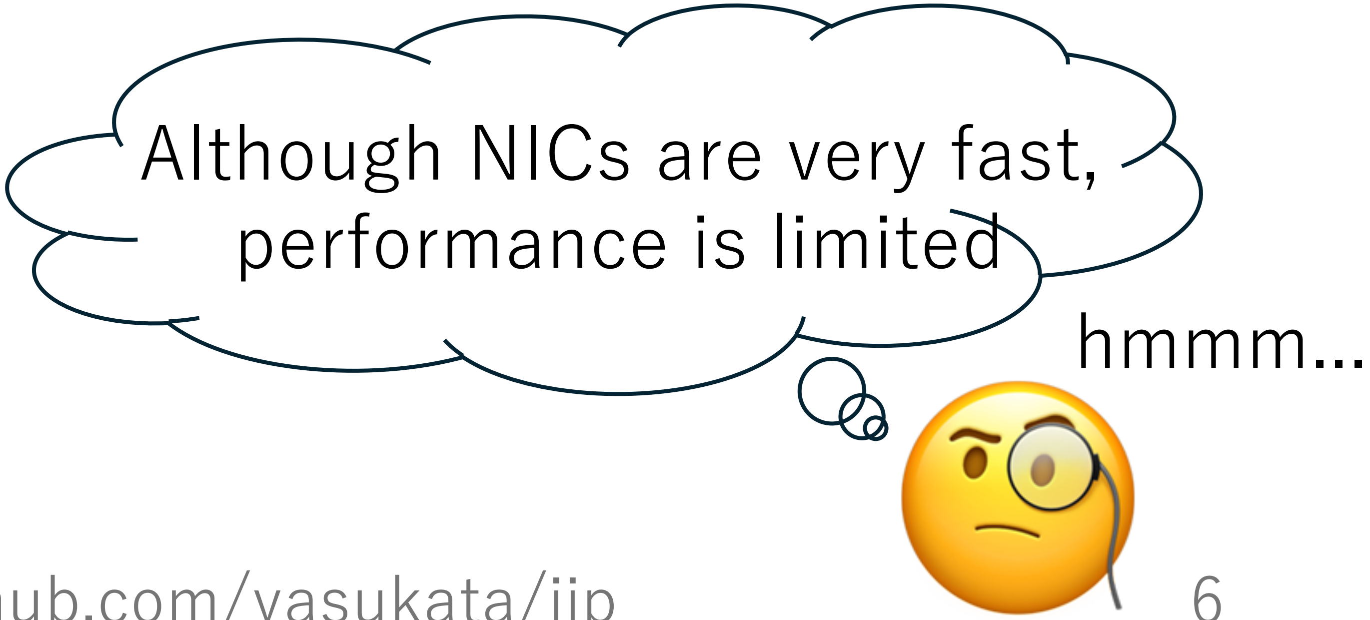
- TCP/IP stacks are typically maintained as part of OS kernels



TCP/IP Stacks in Legacy OS Kernels

- TCP/IP stacks are typically maintained as part of OS kernels
- People found it is hard for TCP/IP stacks in legacy OS kernels to effectively utilize the benefits of high-speed NICs





Performance-optimized TCP/IP Stacks

- To address this issue, research and industry communities have invented many <u>performance-optimized TCP/IP stacks</u>
  - e.g., Sandstorm (SIGCOMM'14), mTCP (NSDI'14)

Performance-optimized TCP/IP Stack

Computer

Power of NICs is unleashed by fast TCP/IP stacks!

Performance-optimized TCP/IP Stacks

Their performance is excellent

e.g., Sandstorm (SIGCOMM'14), mTCP (NSDI'14)

Performance-optimized TCP/IP Stacks

Their performance is excellent

e.g., Sandstorm (SIGCOMM’14), mTCP (NSDI’14)

They often incur high integration complexity

Category	Performance	Integration
Performance-optimized	✓	

Portability-aware TCP/IP Stacks

😄 There are TCP/IP stacks that allow for easy integration

e.g., lwIP, FNET, picoTCP

Category	Performance	Integration
Performance-optimized	✓	
Portability-aware		✓

Portability-aware TCP/IP Stacks

😄 There are TCP/IP stacks that allow for easy integration

e.g., lwIP, FNET, picoTCP

😢 They often lack the care for performance-critical factors

Category	Performance	Integration
Performance-optimized	✓	
Portability-aware		✓

Problem

- None of previous TCP/IP stack implementations allow for easy integration and good performance simultaneously

Category	Performance	Integration
Performance-optimized	✓	
Portability-aware		✓

Problem

- As a result, developers only had limited and laborious options

Category	Performance	Integration
Performance-optimized	✓	
Portability-aware		✓

Problem

- As a result, developers only had limited and laborious options
  - intensively modifying one of the existing TCP/IP stacks

Category	Performance	Integration
Performance-optimized	✓	
Portability-aware		✓

Problem

- As a result, developers only had limited and laborious options
  - intensively modifying one of the existing TCP/IP stacks
  - building a new TCP/IP stack from scratch

Category	Performance	Integration
Performance-optimized	✓	
Portability-aware		✓

Problem

- As a result, developers only had limited and laborious options
  - intensively modifying one of the existing TCP/IP stacks
  - building a new TCP/IP stack from scratch
  - accepting performance limitations of an applicable TCP/IP stack

Category	Performance	Integration
Performance-optimized	✓	
Portability-aware		✓

Problem

- As a result, developers only had limited and laborious options
  - intensively modifying one of the existing TCP/IP stacks
  - building a new TCP/IP stack from scratch
  - accepting performance limitations of an applicable TCP/IP stack
  - giving up the integration

Category	Performance	Integration
Performance-optimized	✓	
Portability-aware		✓

This Work

- We develop iip, an integratable TCP/IP stack, that allows for easy integration and good performance simultaneously

Category	Performance	Integration
Performance-optimized	✓	
Portability-aware		✓
This work	✓	✓

https://github.com/yasukata/iip

Issues of Existing TCP/IP Stacks

- Performance-optimized TCP/IP stacks
  - Dependencies on other components
  - Functionality conflicts
  - Limited choices for CPU core assignment models

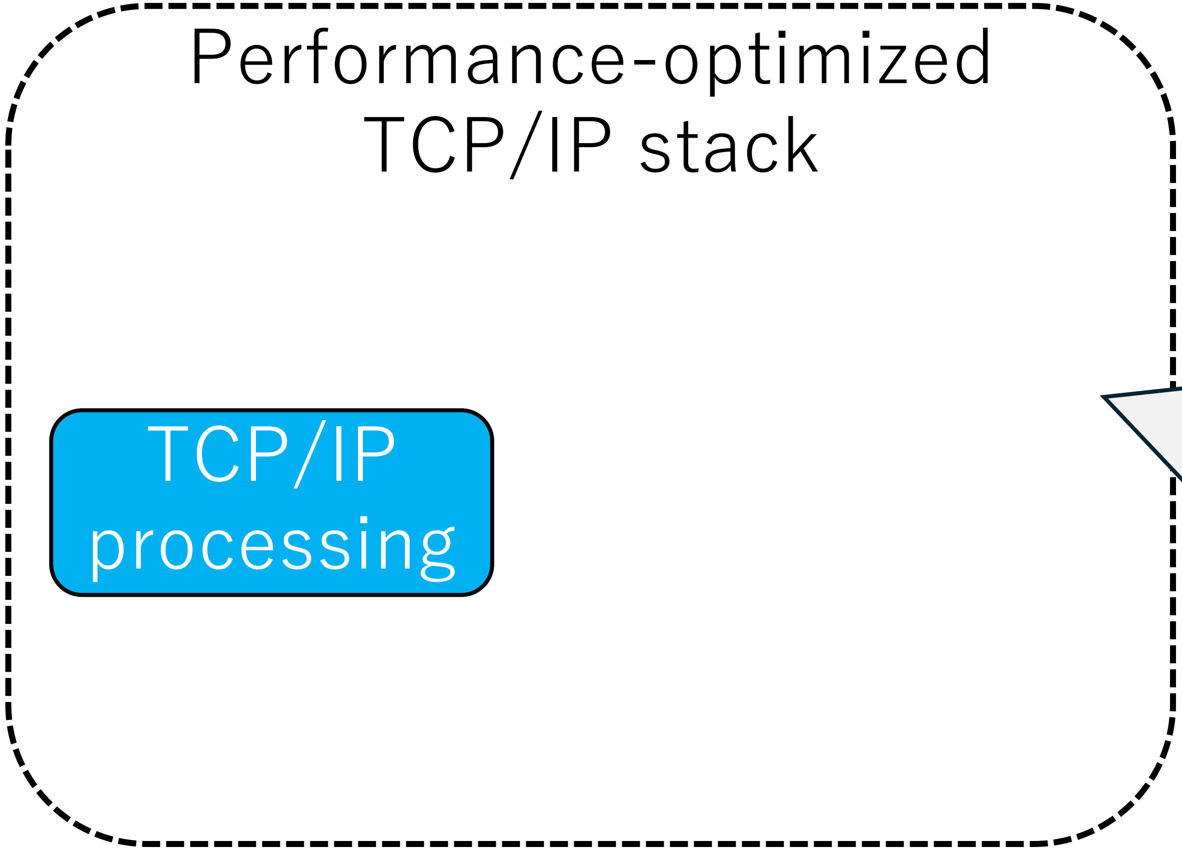
- Portability-aware TCP/IP stacks
  - Unaware of NIC offloading features
  - Lack of zero-copy I/O capability
  - Lack of multi-core scalability

Issues of Existing TCP/IP Stacks

Performance-optimized
TCP/IP stack

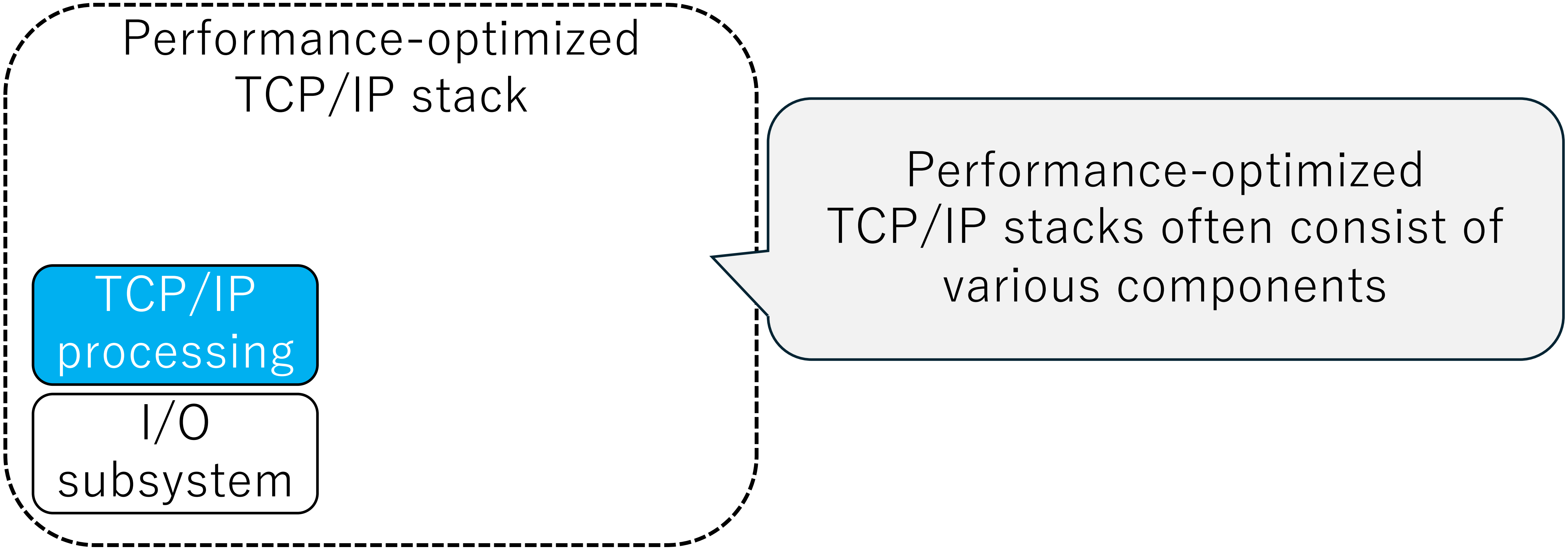
Performance-optimized
TCP/IP stacks often consist of
various components

Issues of Existing TCP/IP Stacks

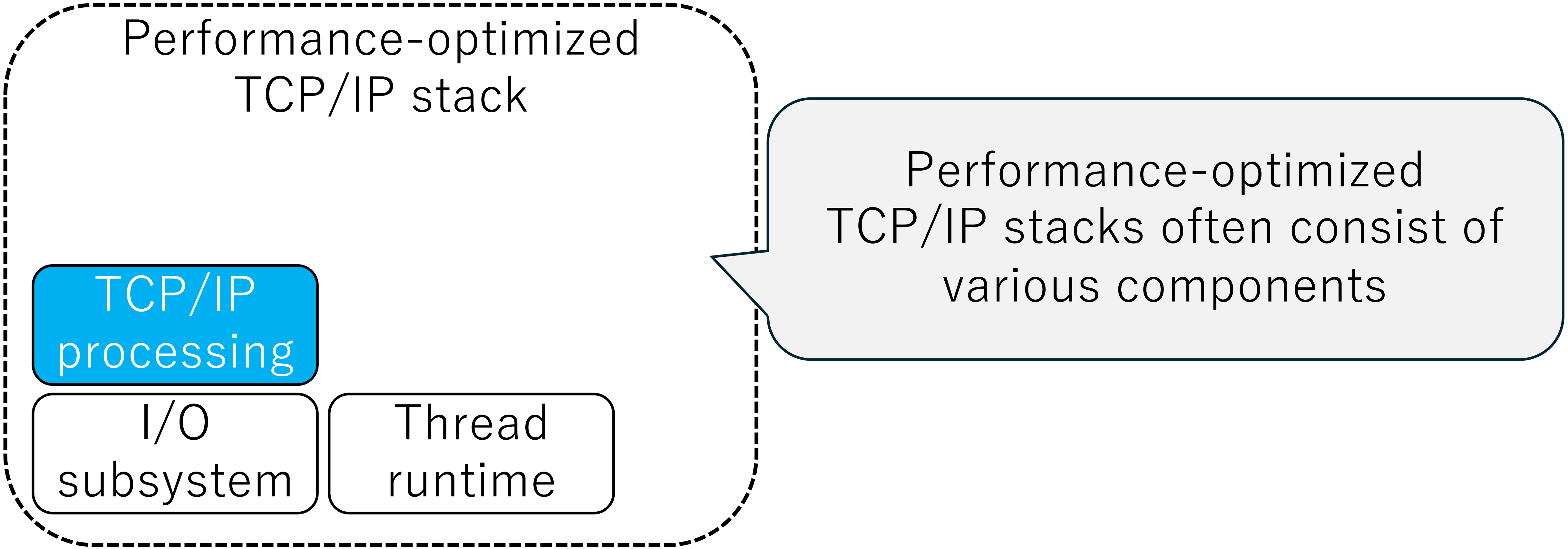


Performance-optimized TCP/IP stacks often consist of various components

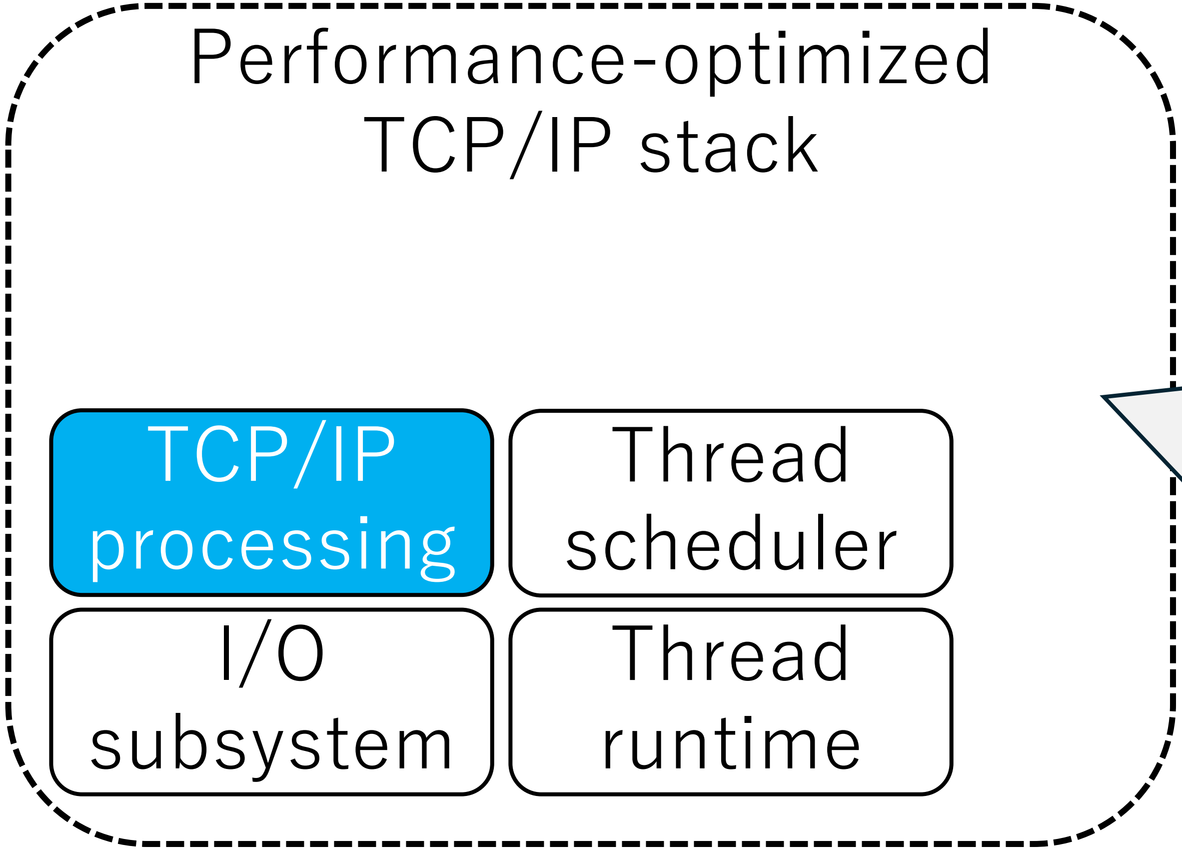
Issues of Existing TCP/IP Stacks



Issues of Existing TCP/IP Stacks

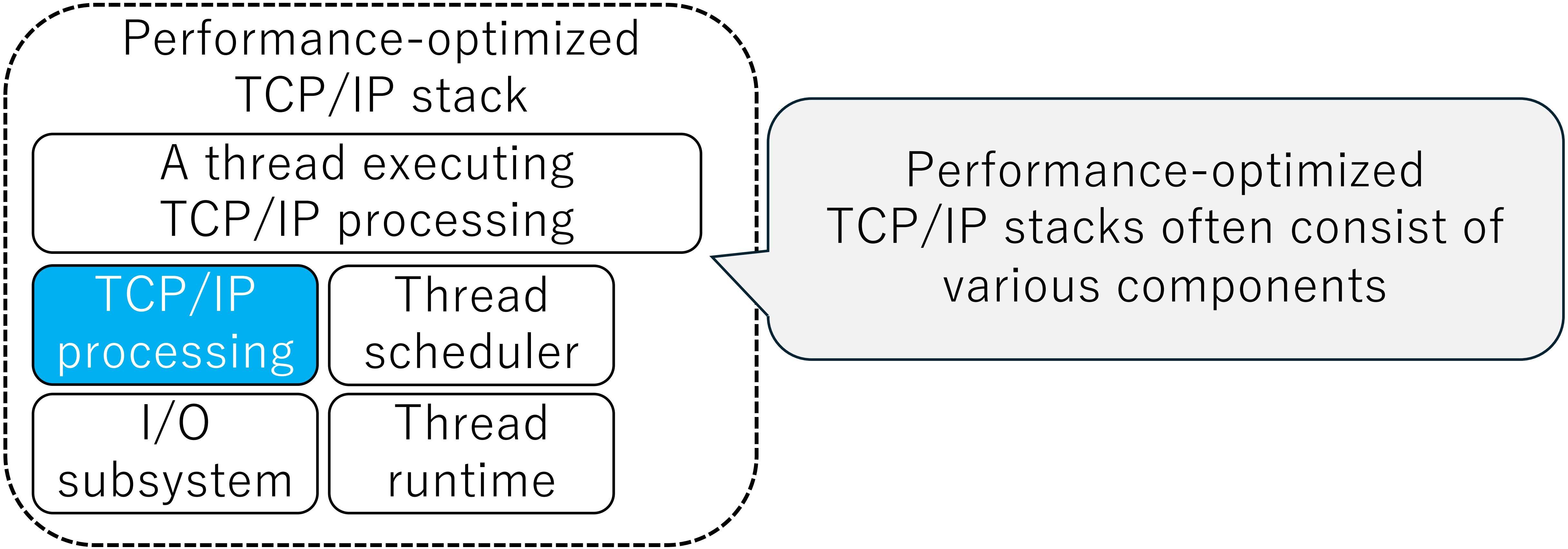


Issues of Existing TCP/IP Stacks

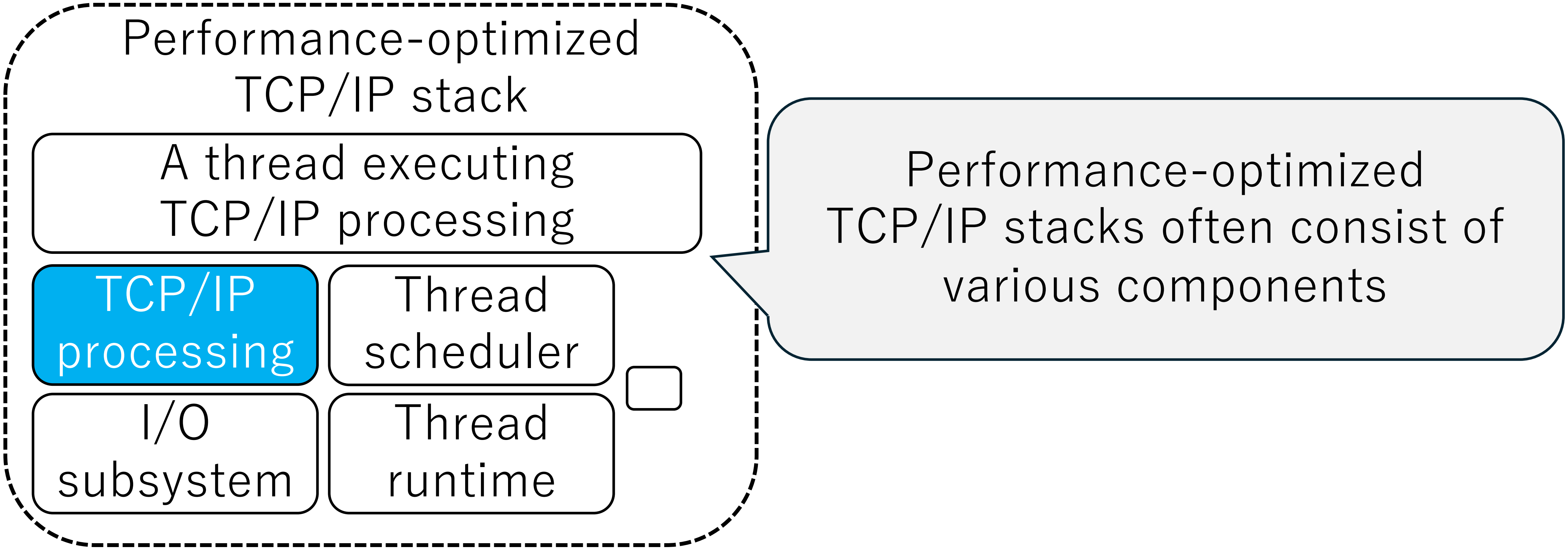


Performance-optimized TCP/IP stacks often consist of various components

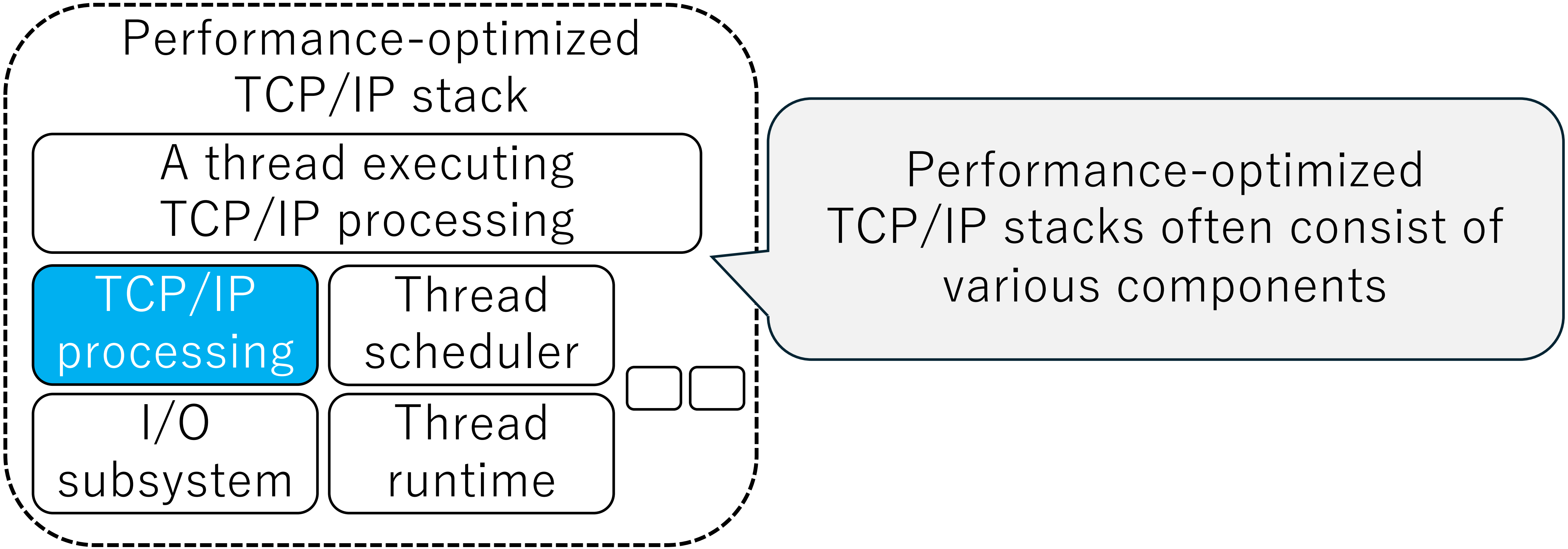
Issues of Existing TCP/IP Stacks



Issues of Existing TCP/IP Stacks

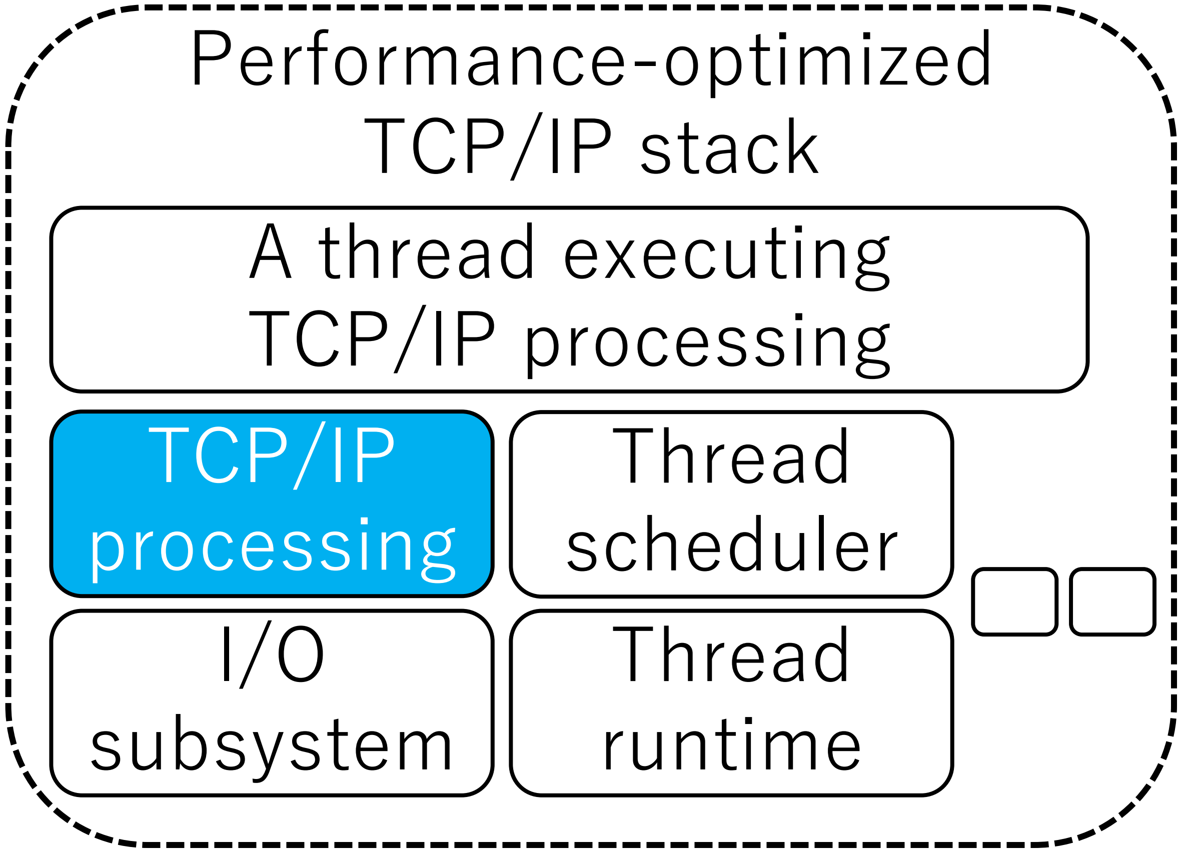


Issues of Existing TCP/IP Stacks



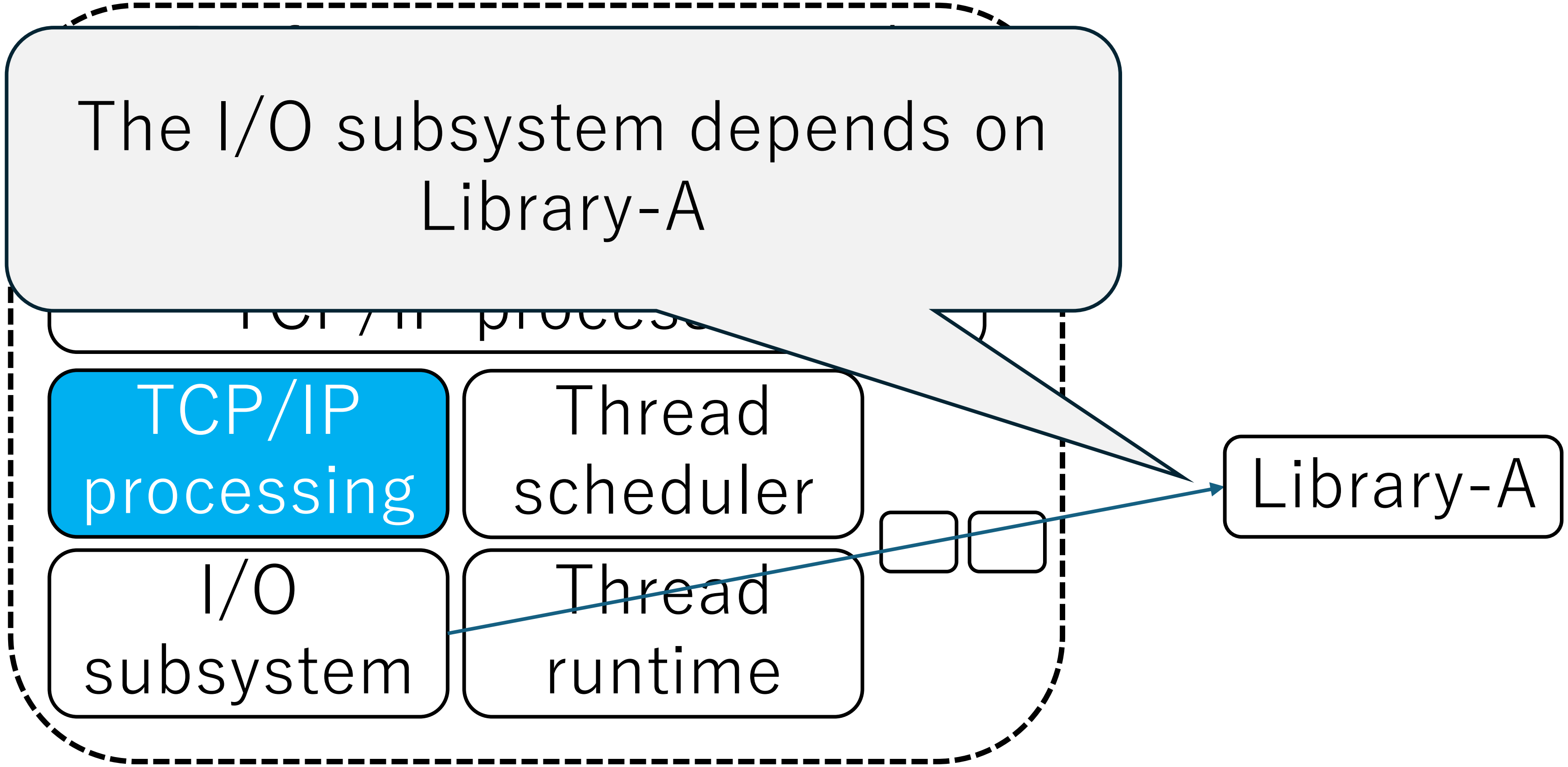
Issues of Existing TCP/IP Stacks

Dependencies on Other Components



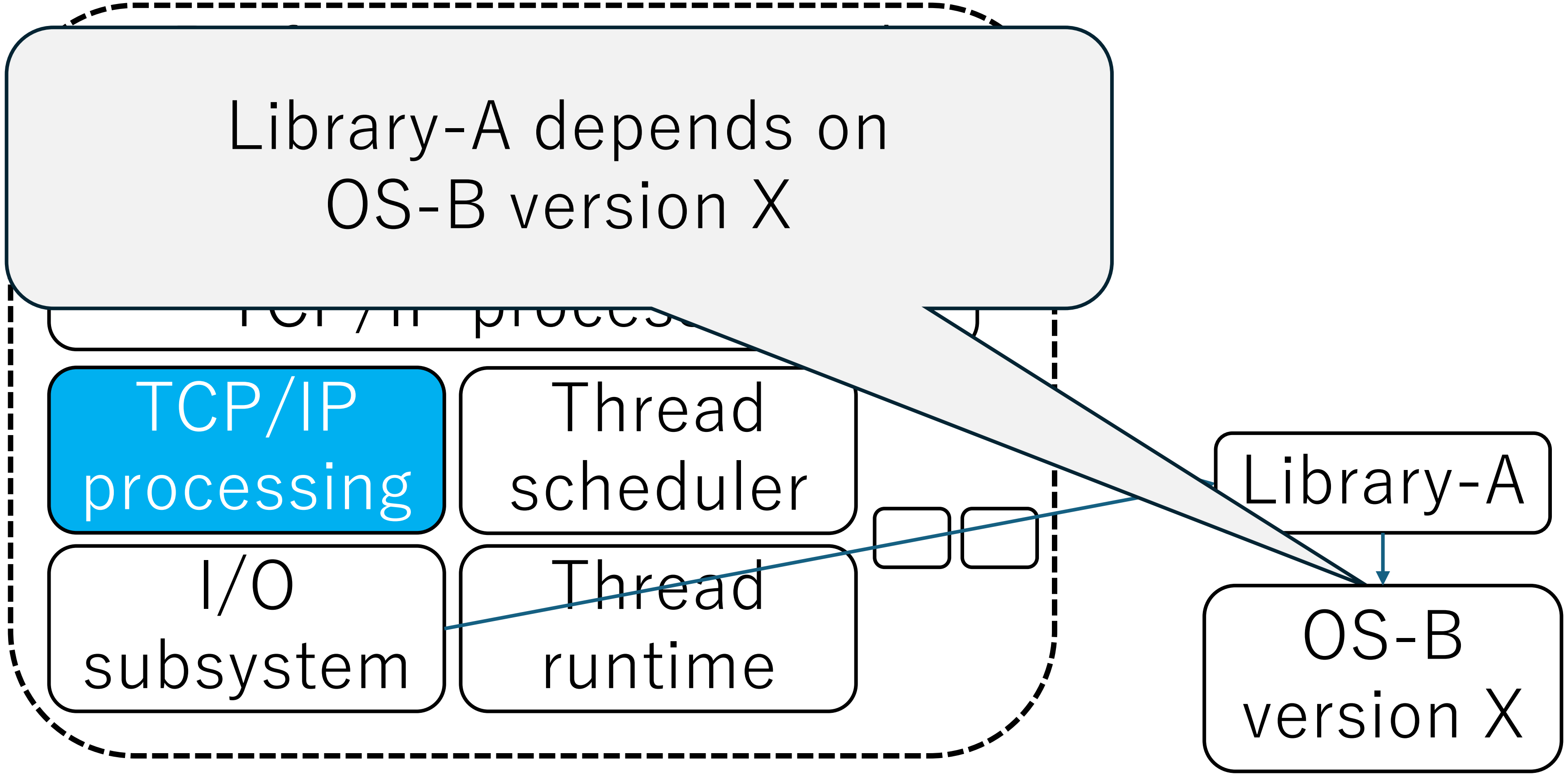
Issues of Existing TCP/IP Stacks

Dependencies on Other Components



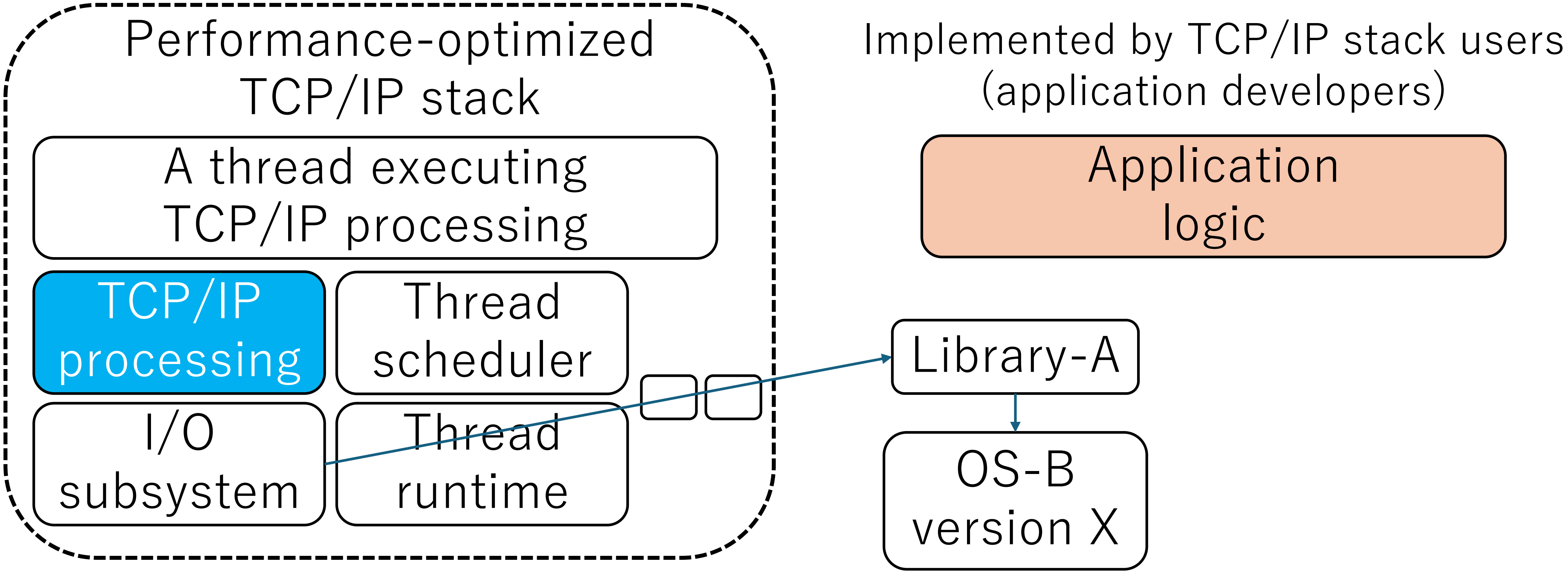
Issues of Existing TCP/IP Stacks

Dependencies on Other Components



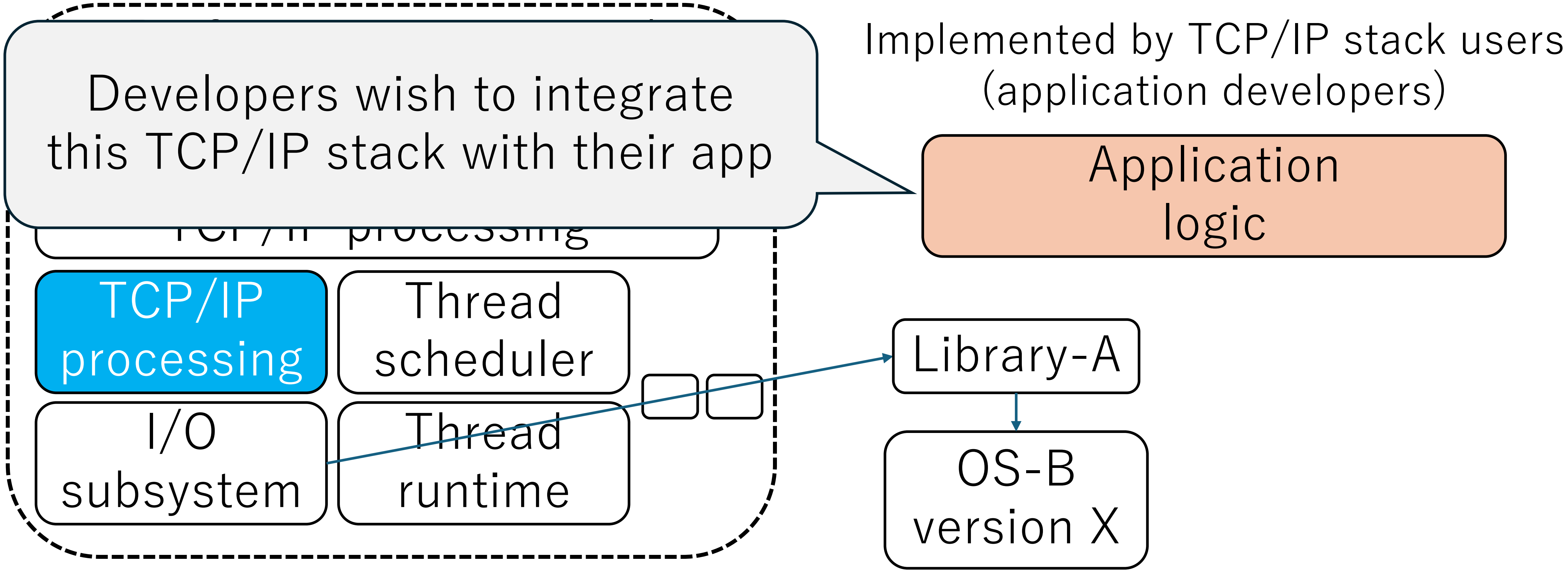
Issues of Existing TCP/IP Stacks

Dependencies on Other Components



Issues of Existing TCP/IP Stacks

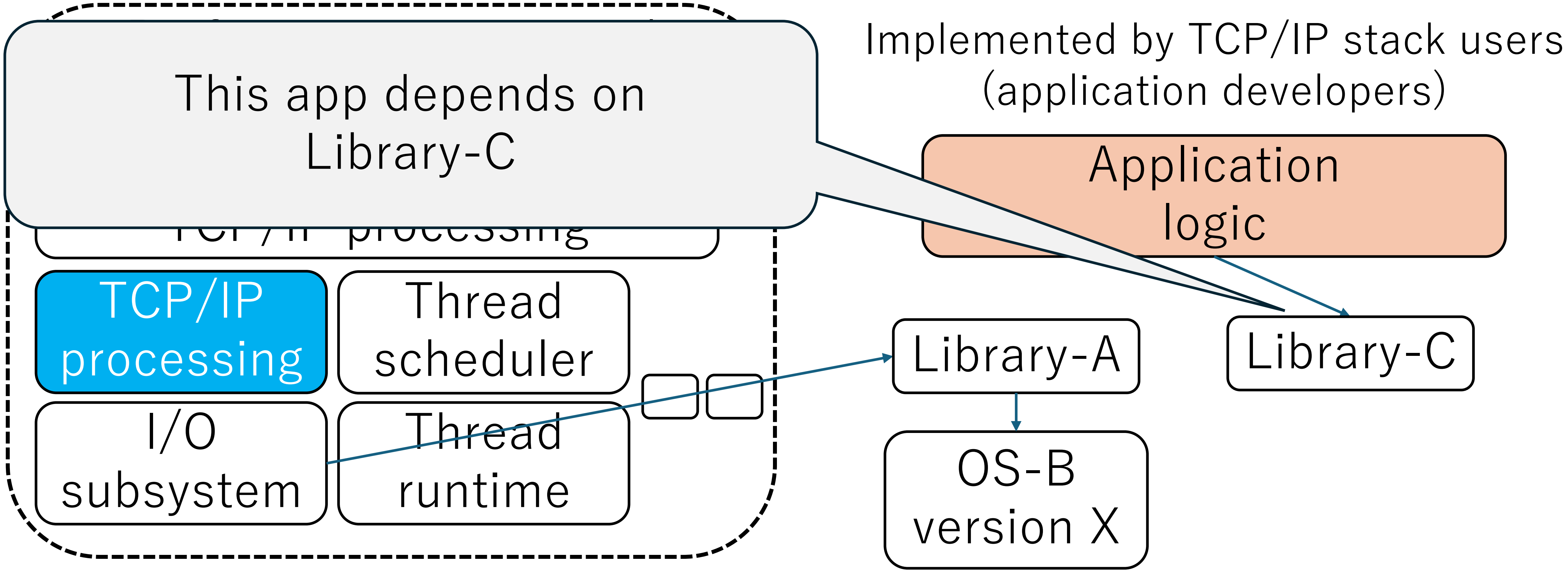
Dependencies on Other Components



https://github.com/yasukata/iip

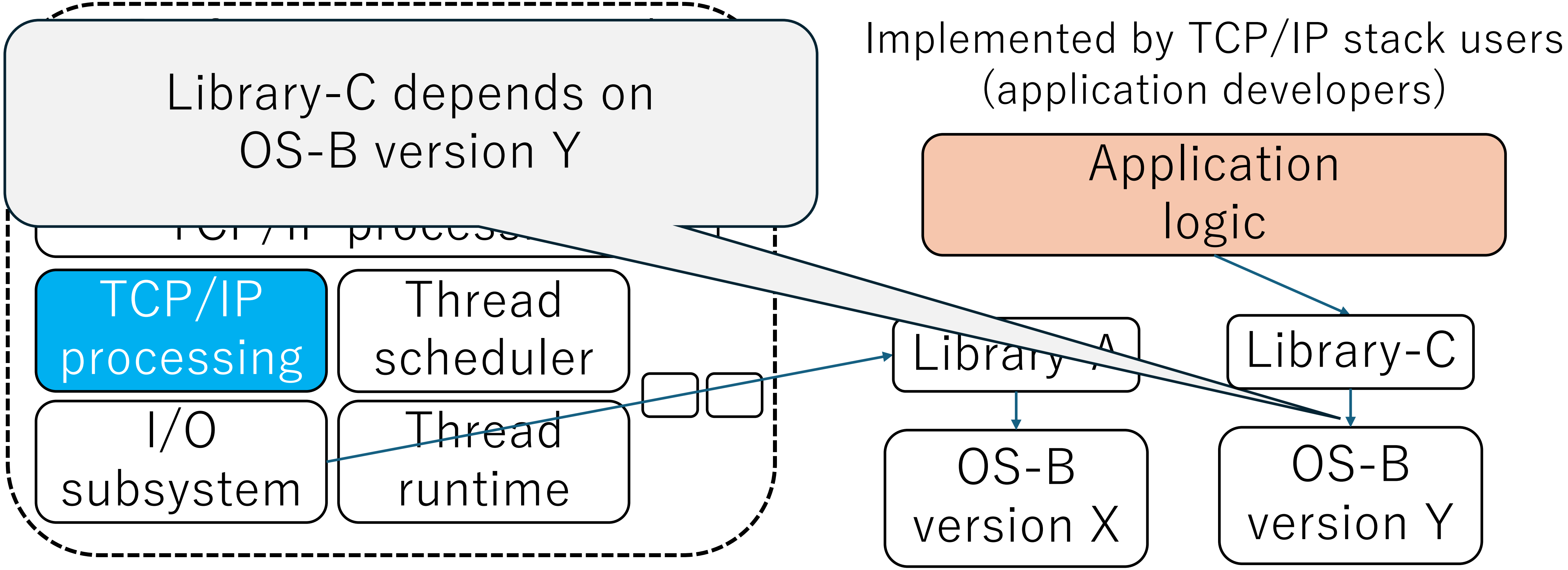
Issues of Existing TCP/IP Stacks

Dependencies on Other Components



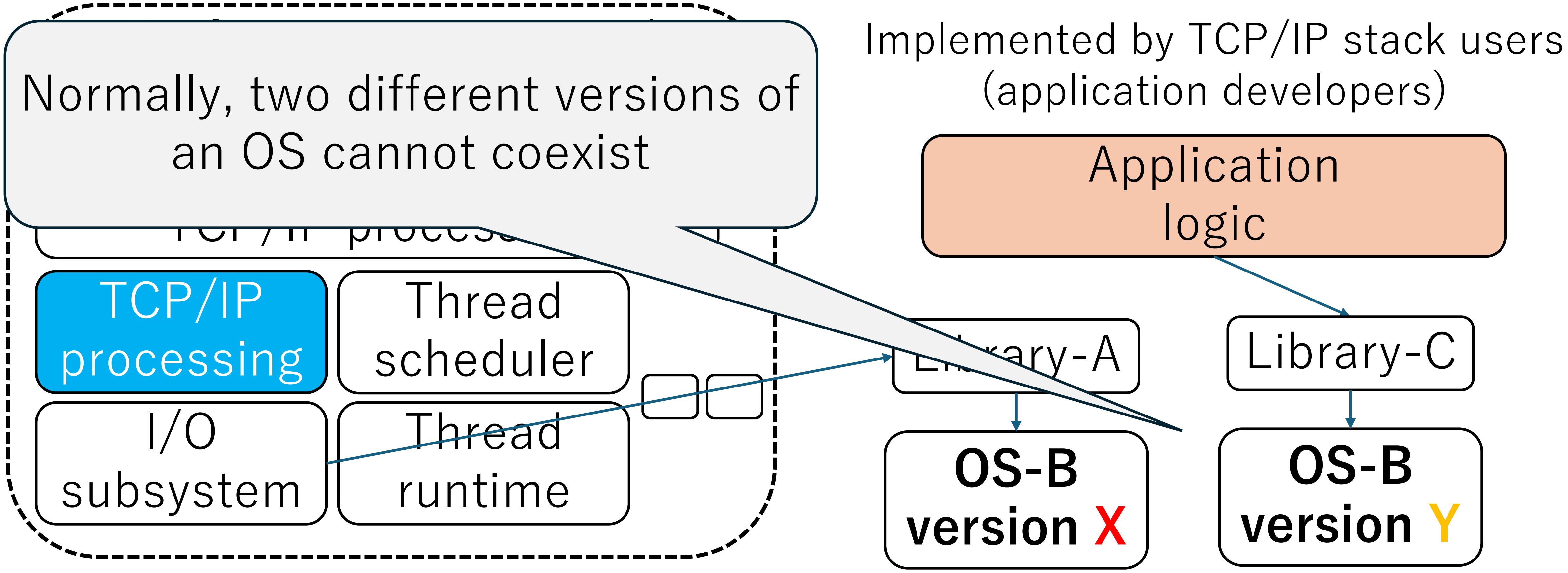
Issues of Existing TCP/IP Stacks

Dependencies on Other Components



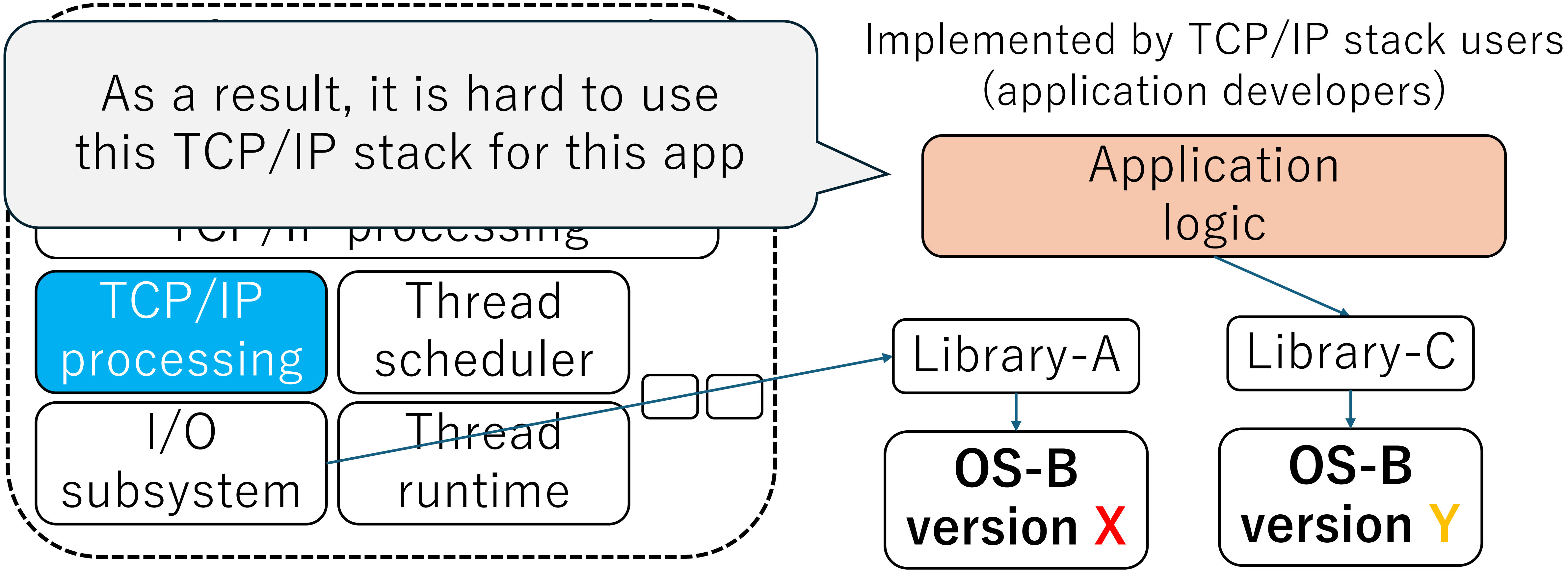
Issues of Existing TCP/IP Stacks

Dependencies on Other Components



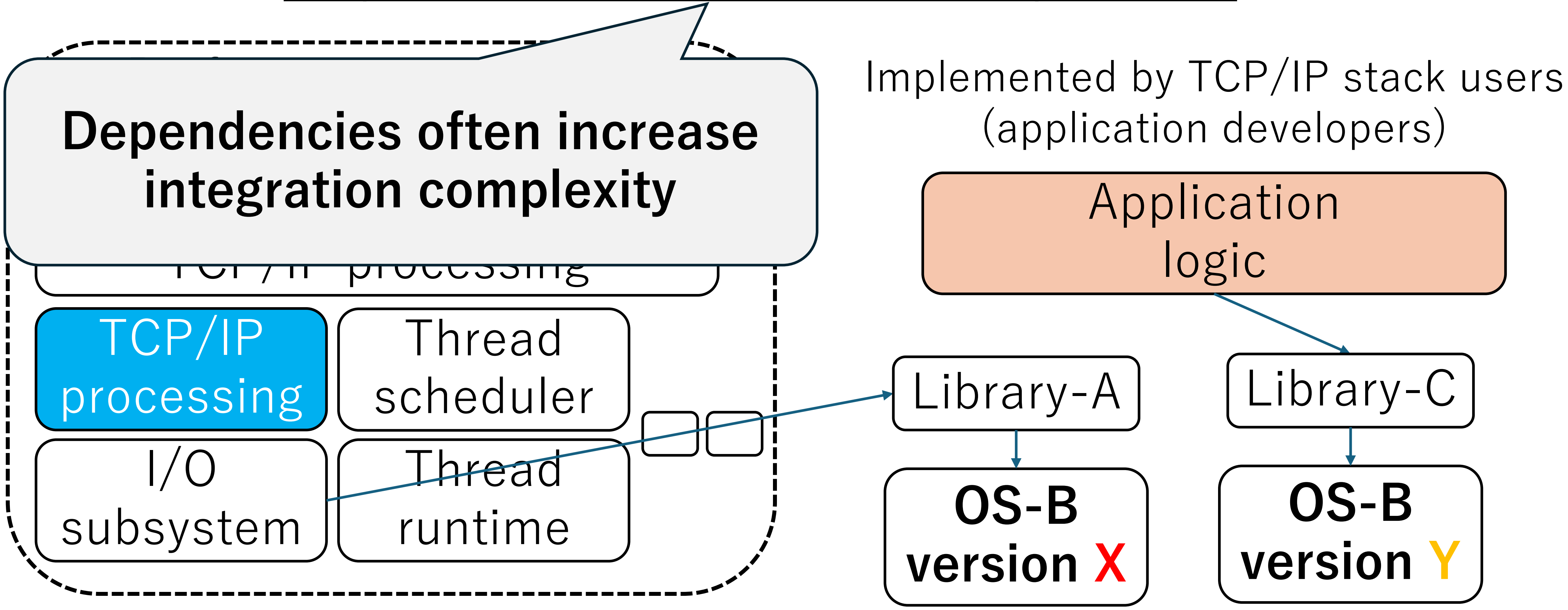
Issues of Existing TCP/IP Stacks

Dependencies on Other Components



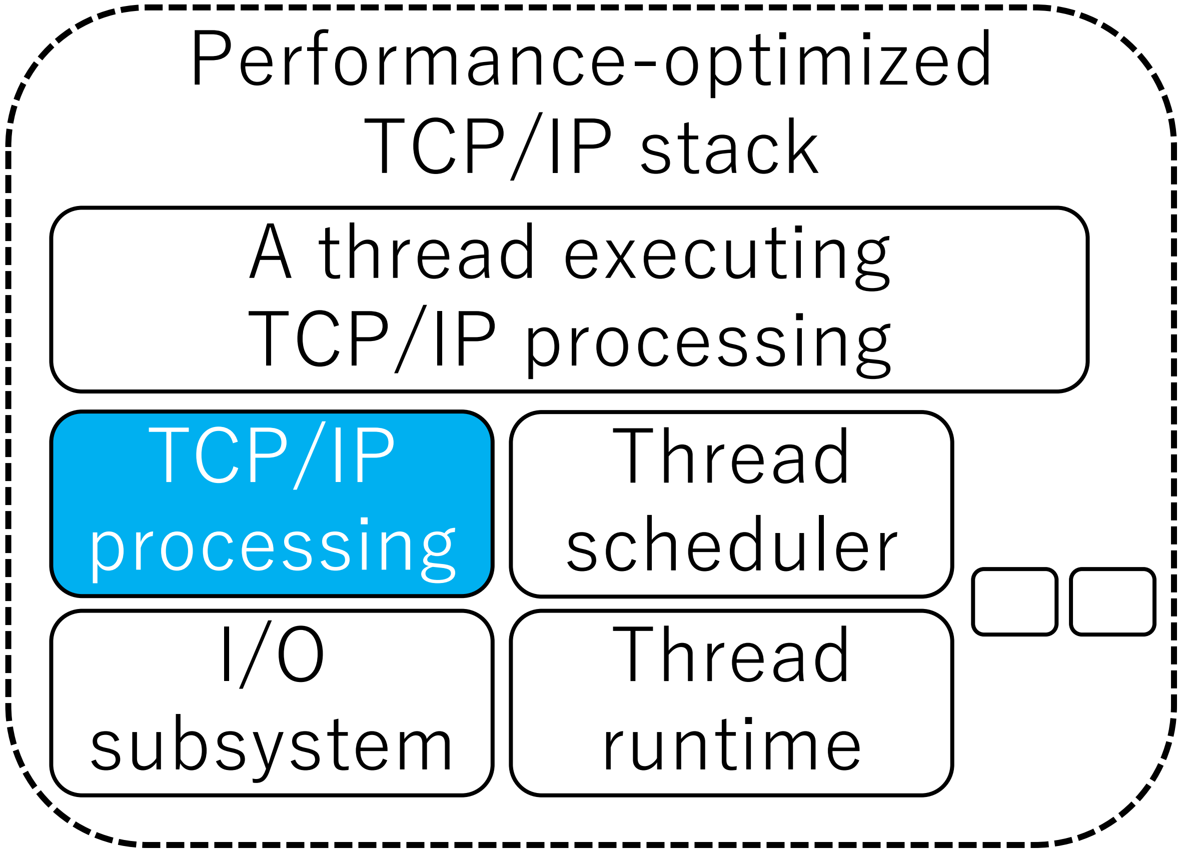
Issues of Existing TCP/IP Stacks

Dependencies on Other Components



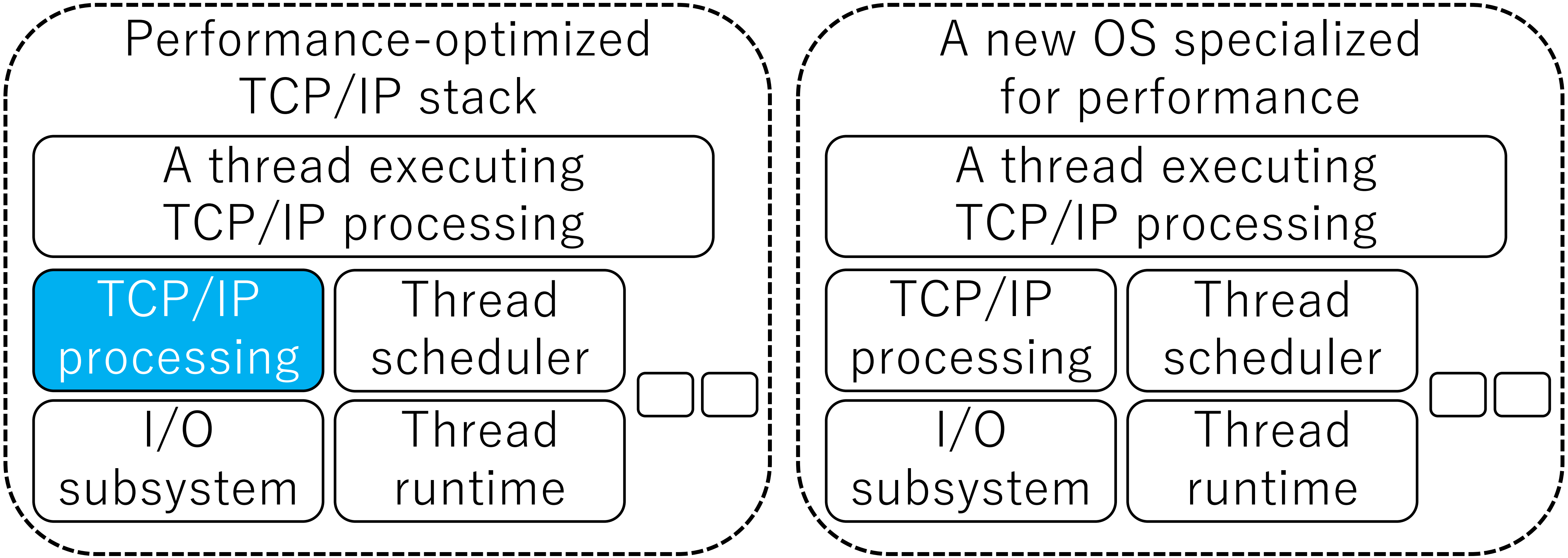
Issues of Existing TCP/IP Stacks

Functionality Conflicts



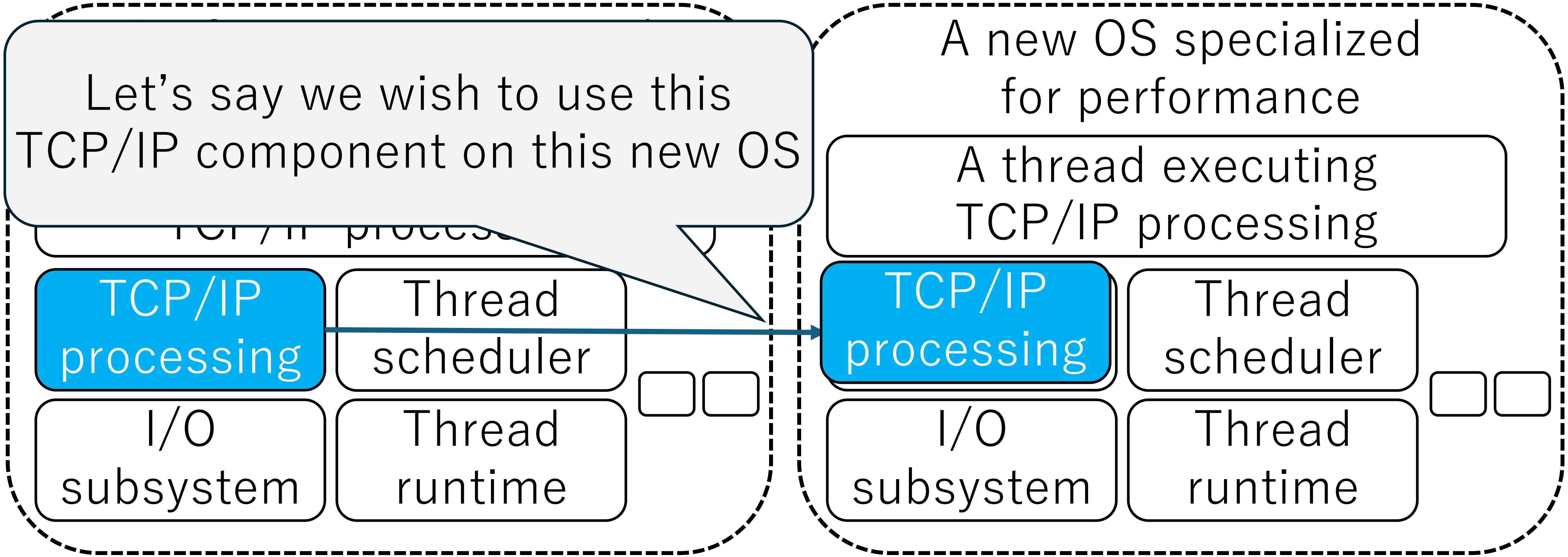
Issues of Existing TCP/IP Stacks

Functionality Conflicts



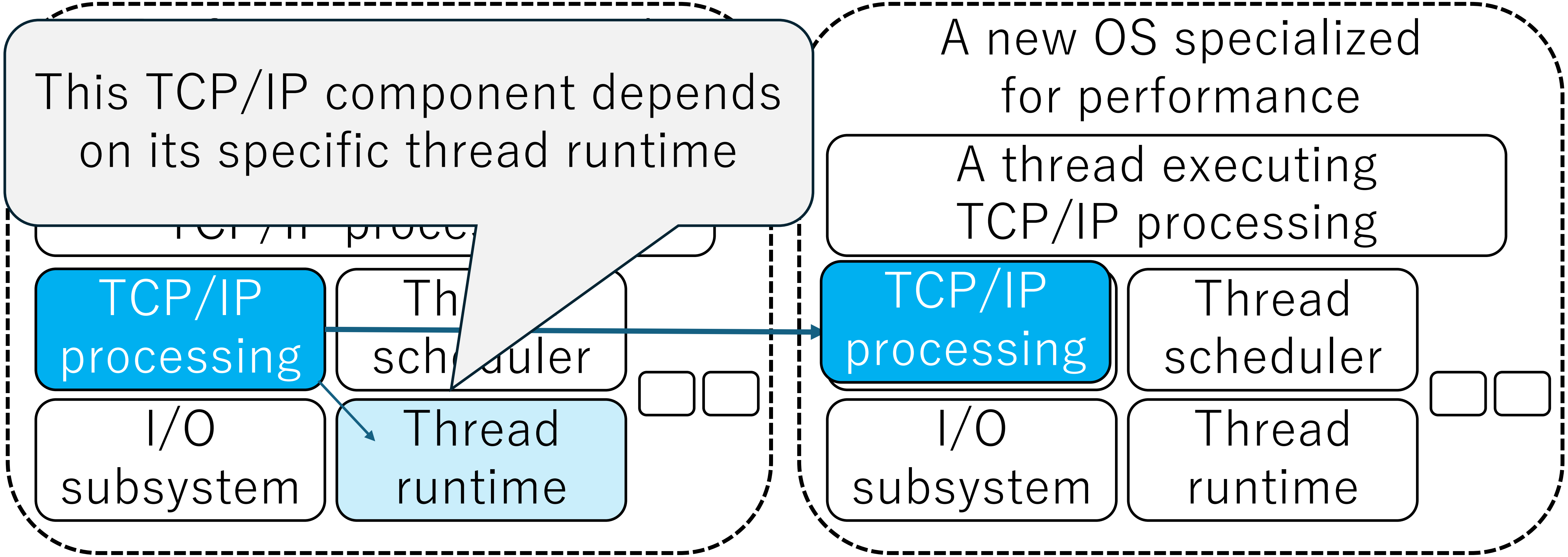
Issues of Existing TCP/IP Stacks

Functionality Conflicts



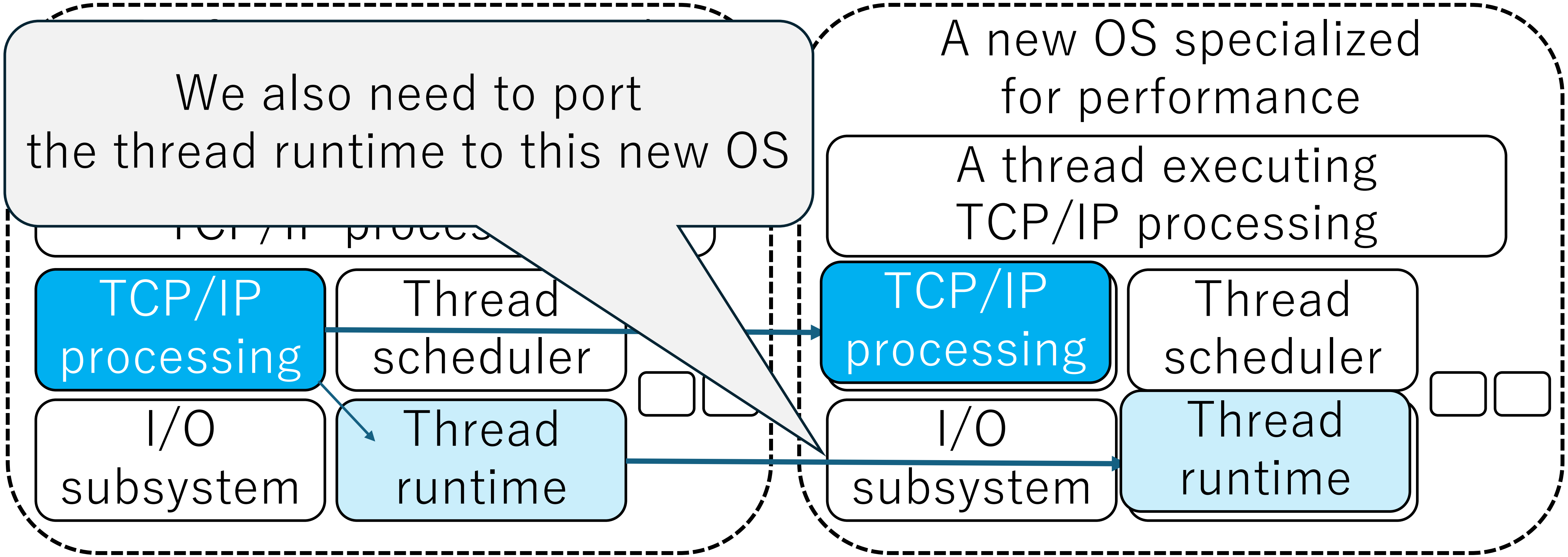
Issues of Existing TCP/IP Stacks

Functionality Conflicts



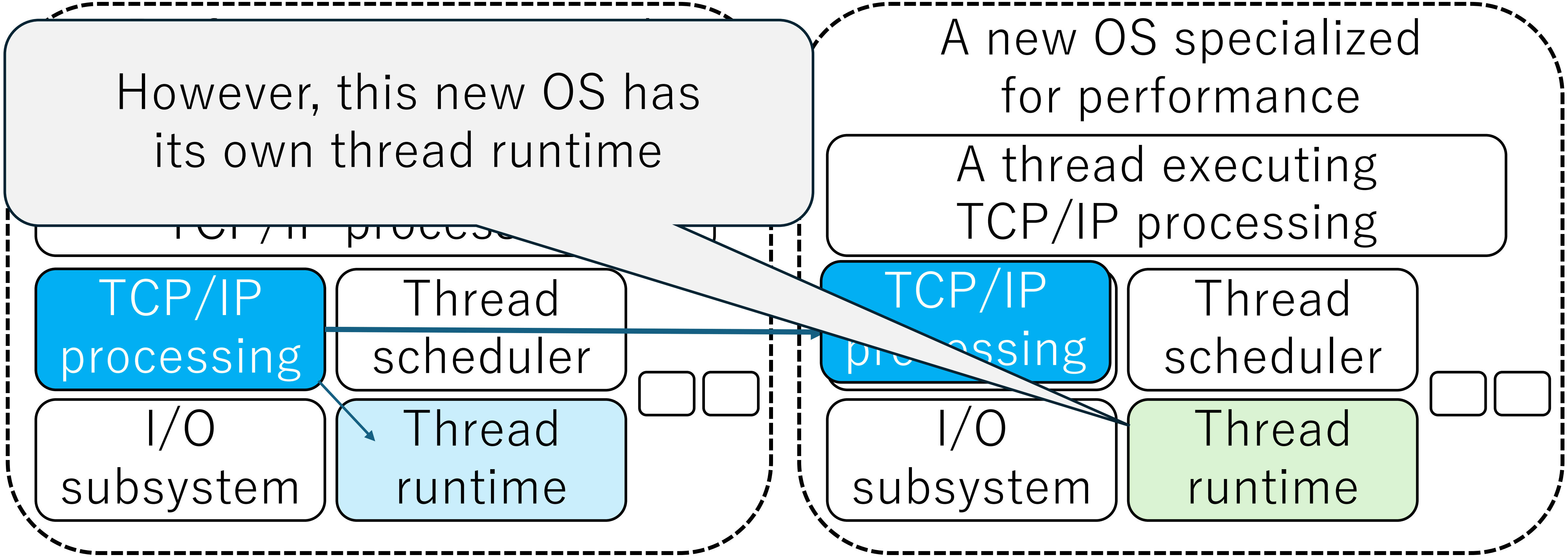
Issues of Existing TCP/IP Stacks

Functionality Conflicts



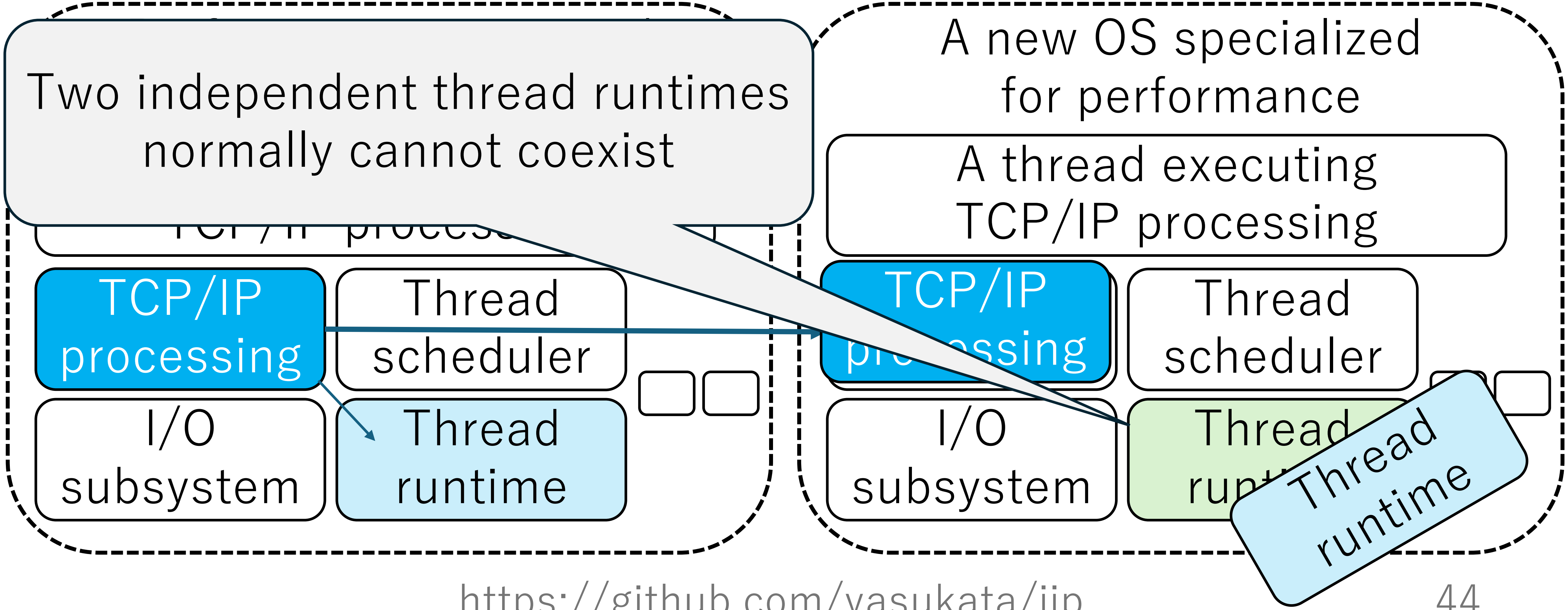
Issues of Existing TCP/IP Stacks

Functionality Conflicts



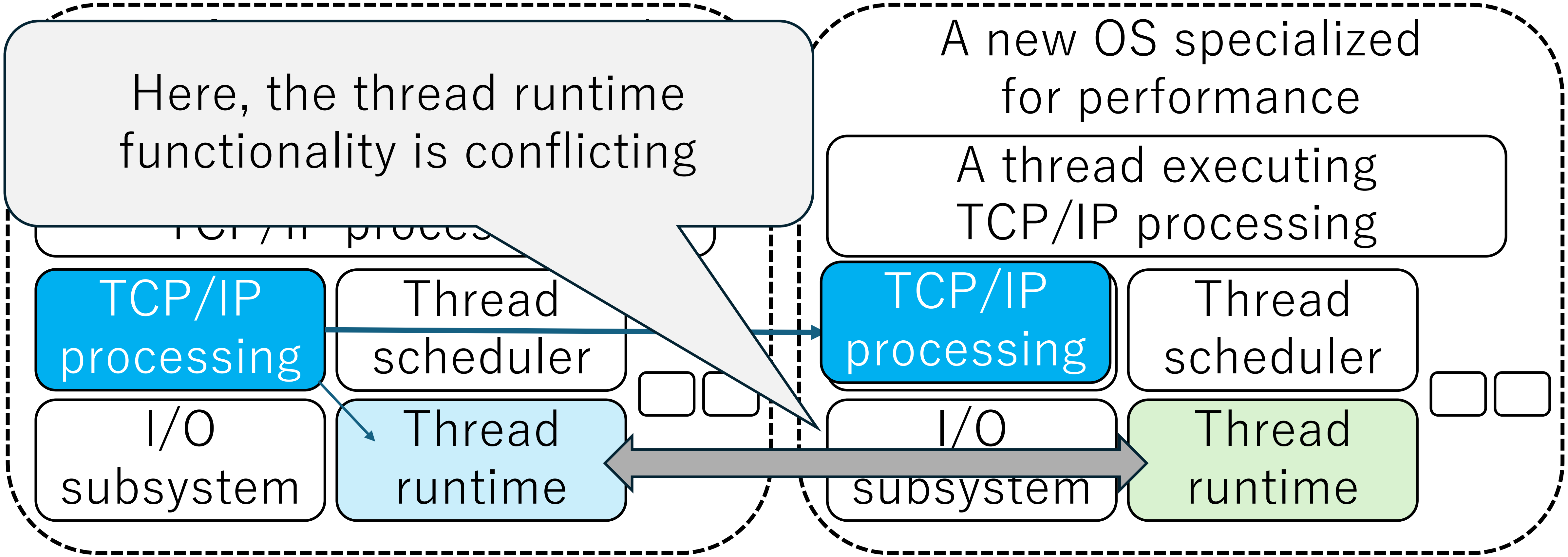
Issues of Existing TCP/IP Stacks

Functionality Conflicts



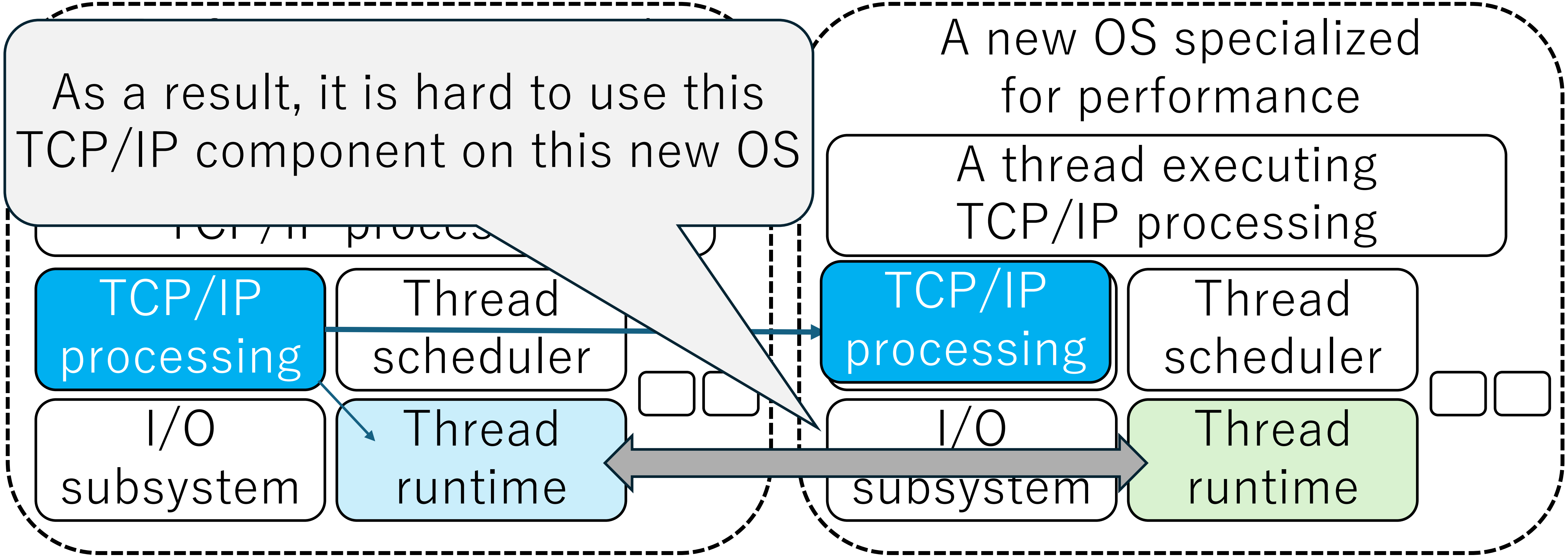
Issues of Existing TCP/IP Stacks

Functionality Conflicts



Issues of Existing TCP/IP Stacks

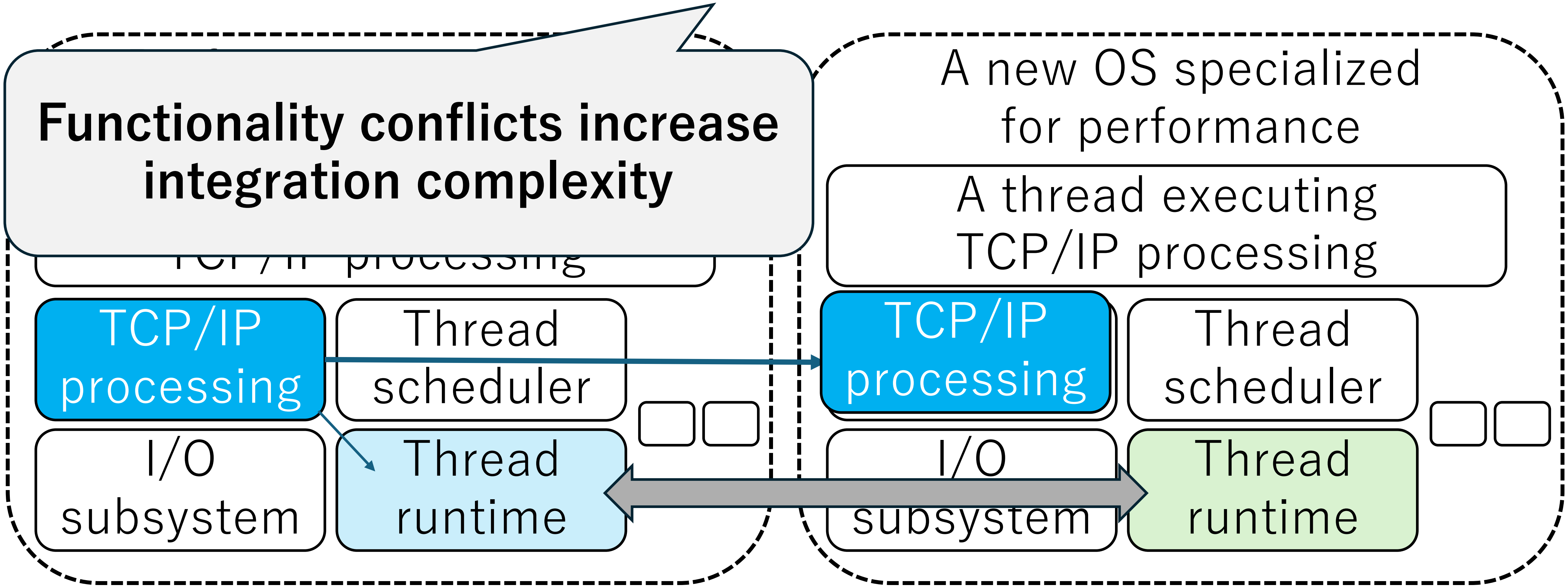
Functionality Conflicts



https://github.com/yasukata/iip

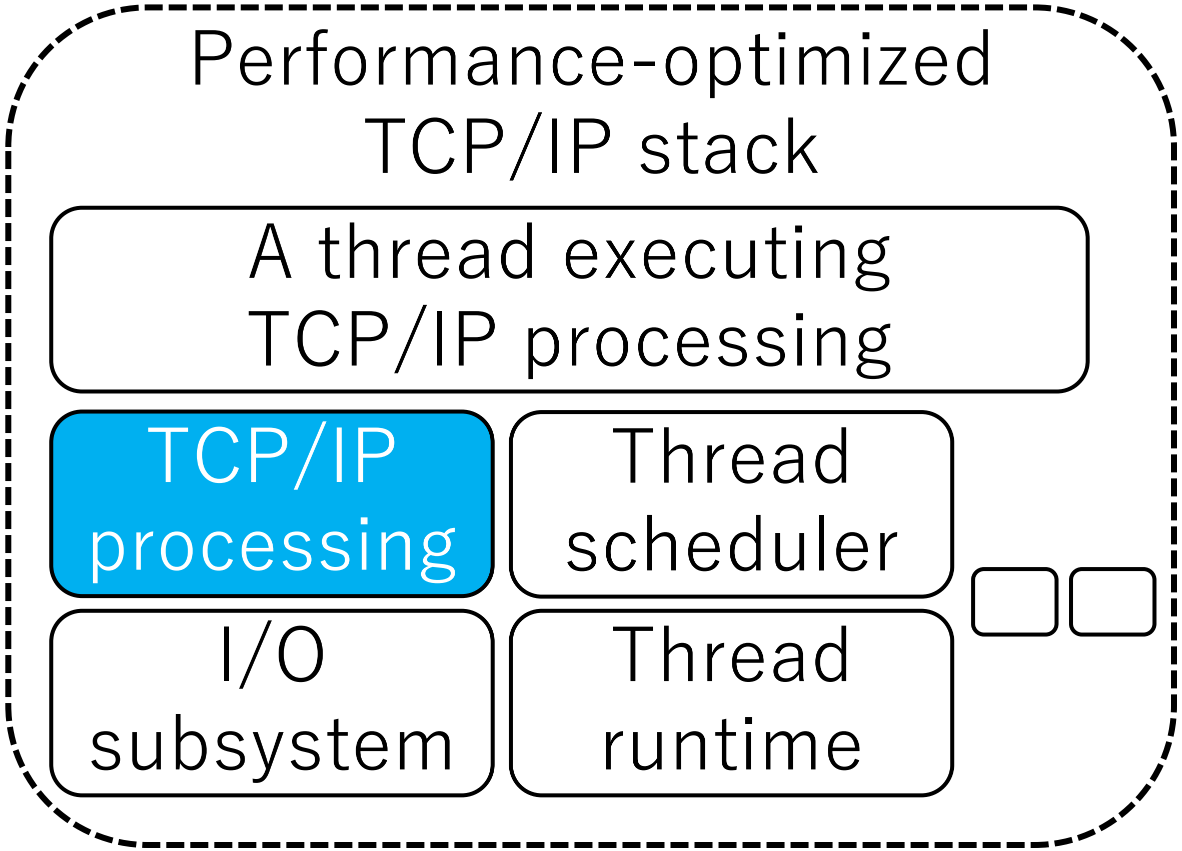
Issues of Existing TCP/IP Stacks

Functionality Conflicts



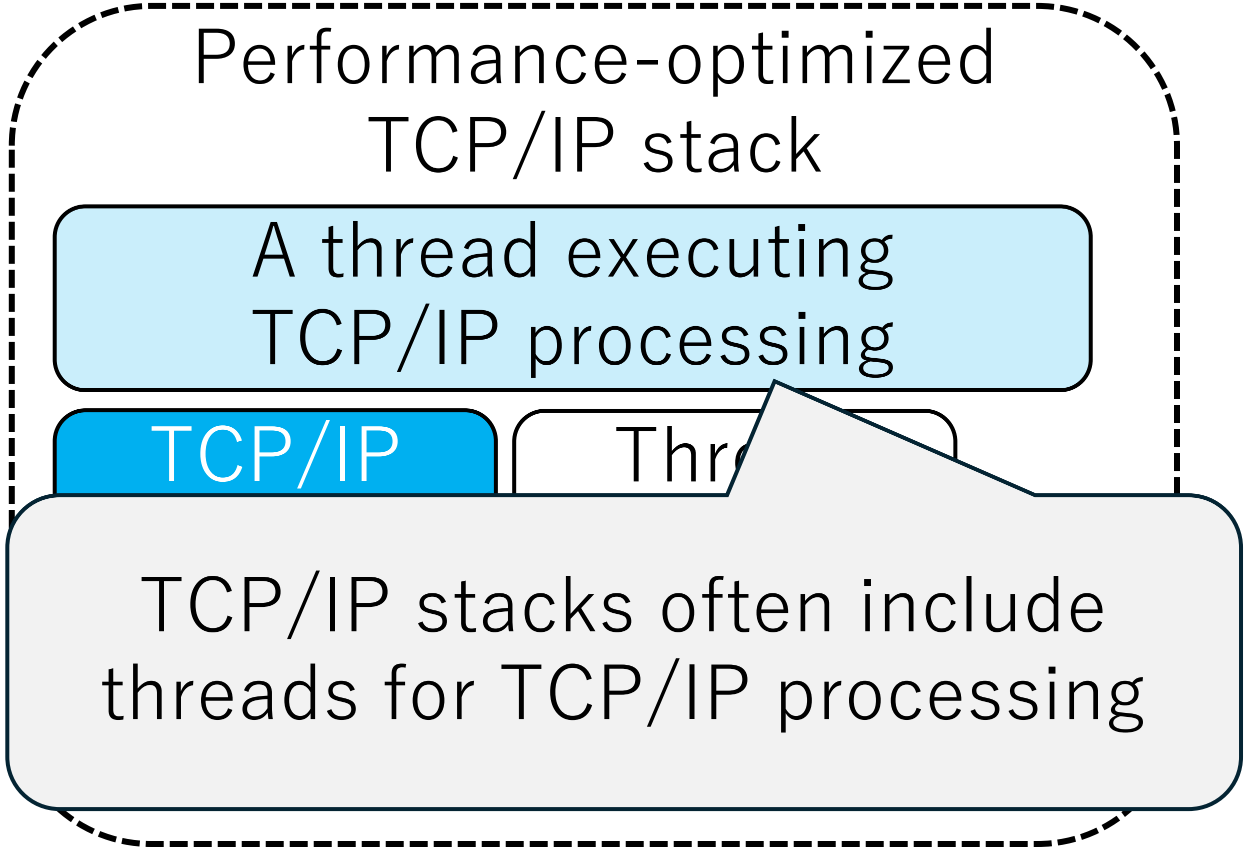
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



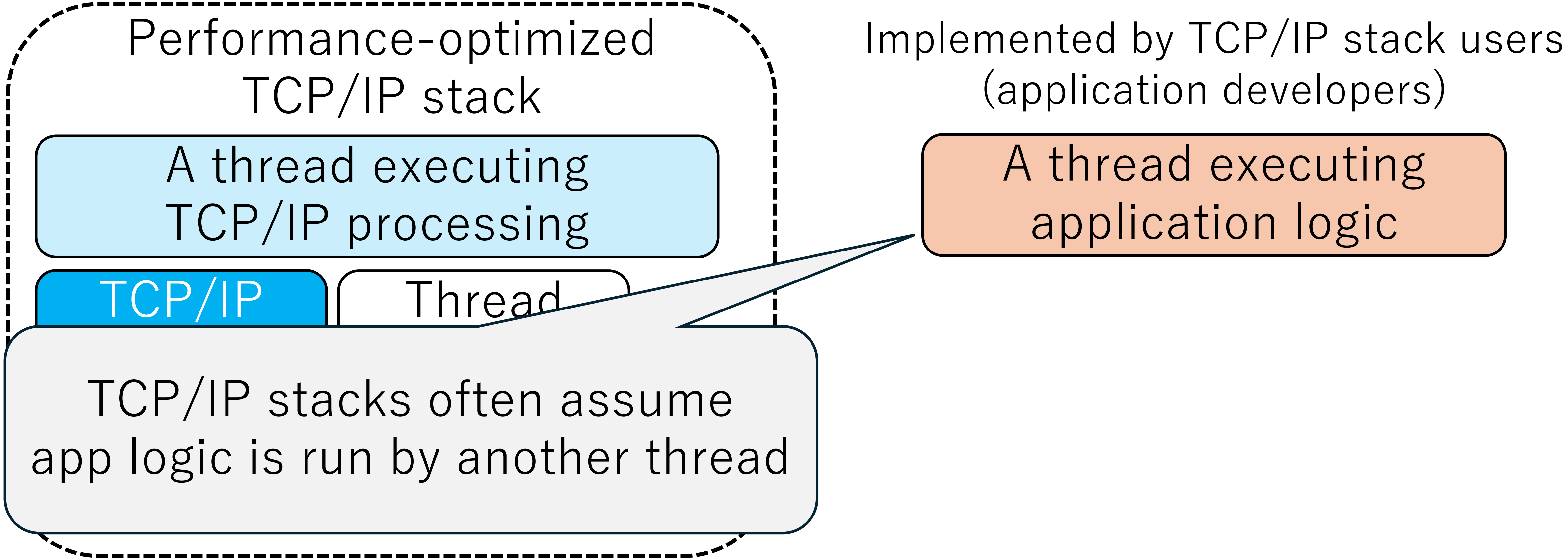
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models

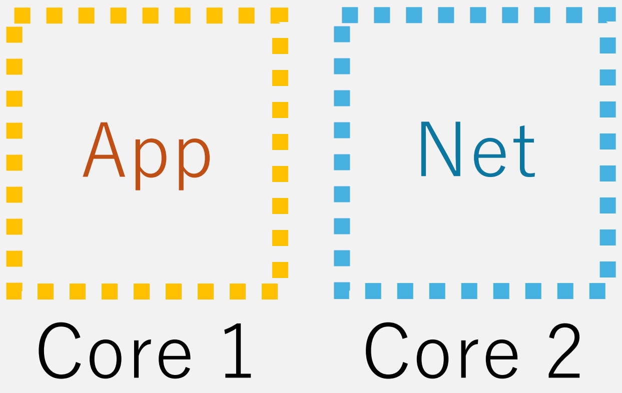
Three potential CPU core assignment models

Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models

Three potential CPU core assignment models

Split



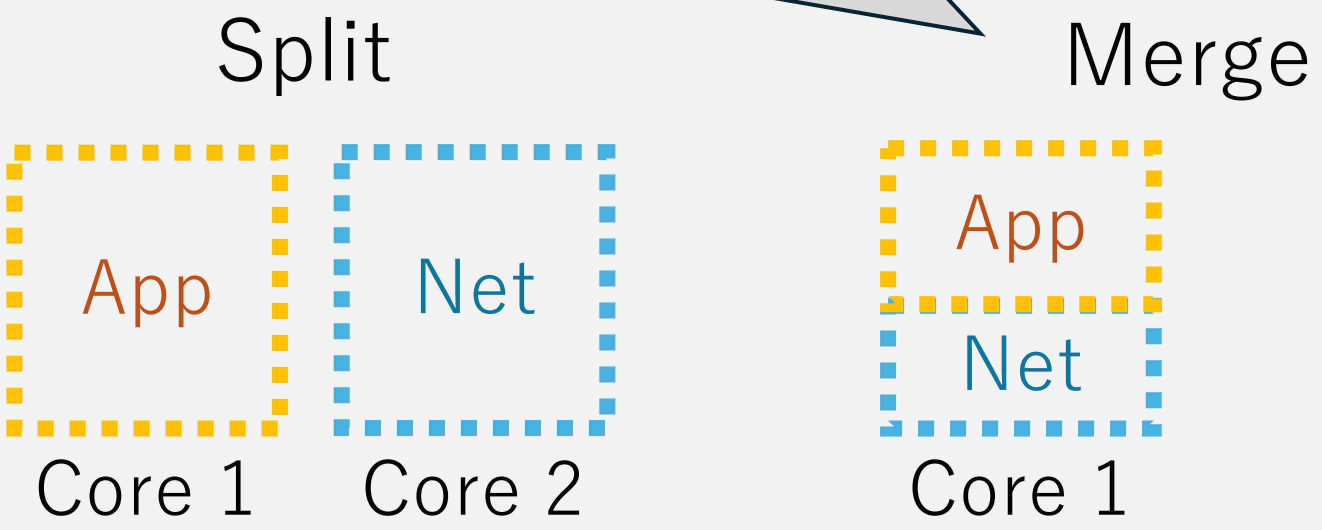
App and Net threads run on different CPU cores

Issues of Existing TCP/IP Stacks

<u>Limited ... nment Models</u>

App and Net threads run on the same CPU core

Thre... nent models



Issues of Existing TCP/IP Stacks

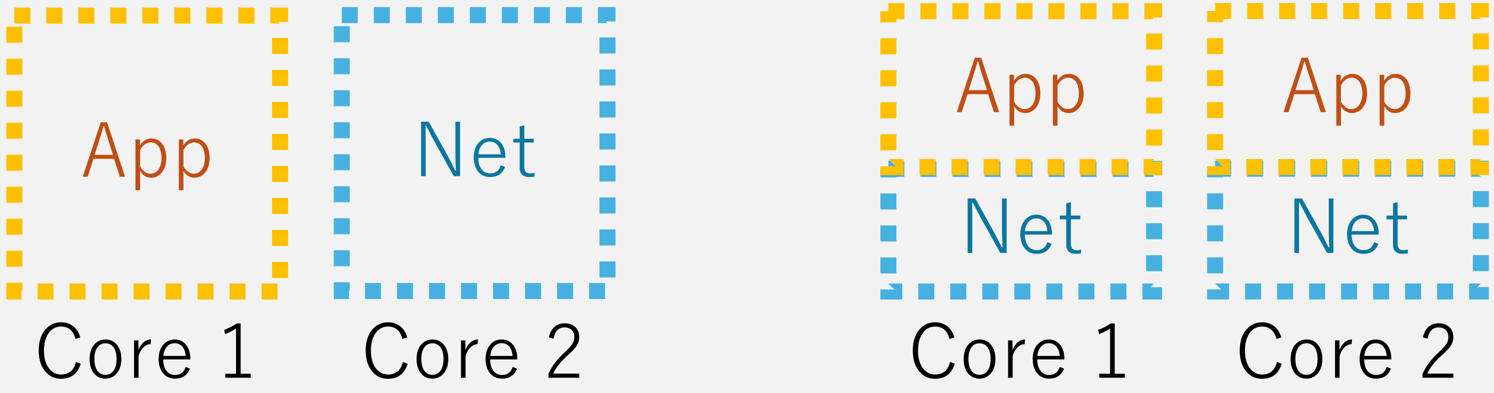
Limited nment Models

Duplicate the same setup to available CPU cores

Thre ment models

Split

Merge

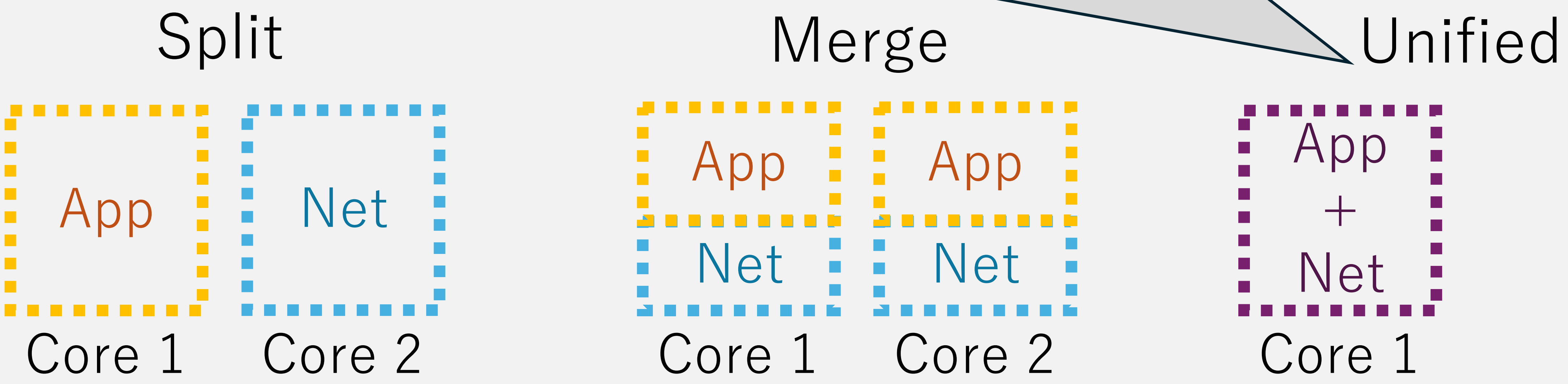


https://github.com/yasukata/iip

Issues of Existing TCP/IP Stacks

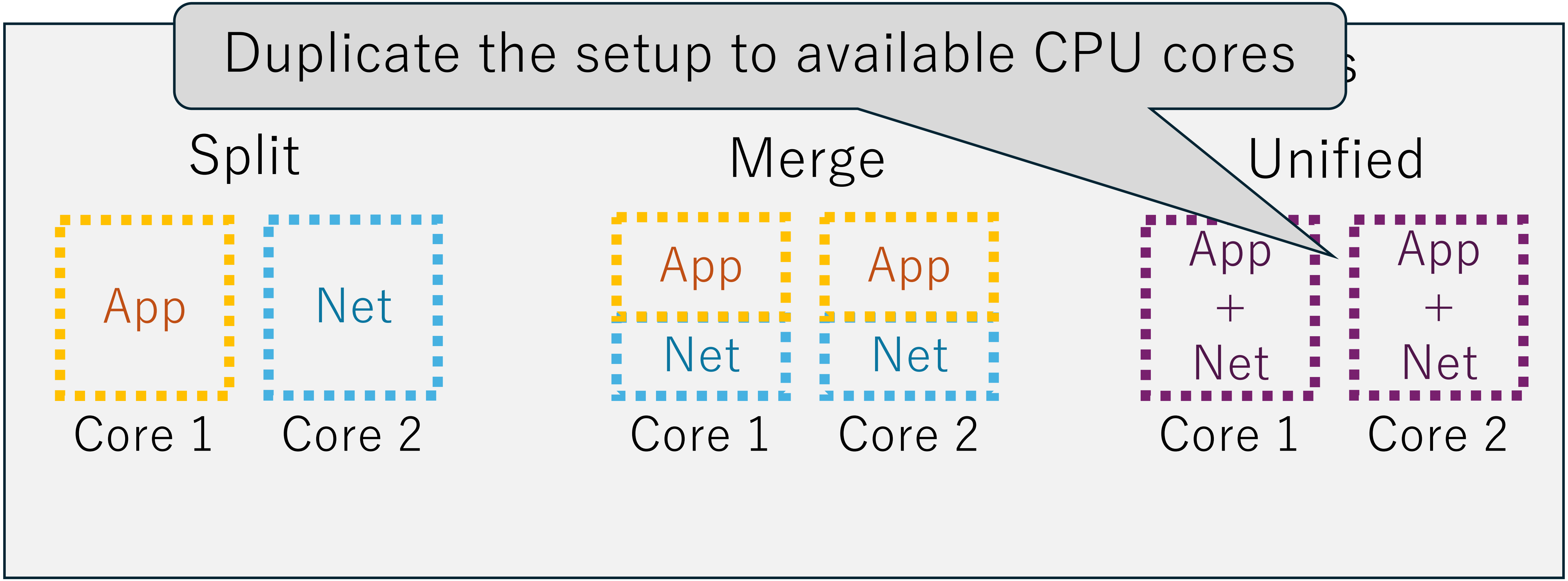
Limited Choices for CPU Core Assignment Models

A thread executes both App and Net logic



Issues of Existing TCP/IP Stacks

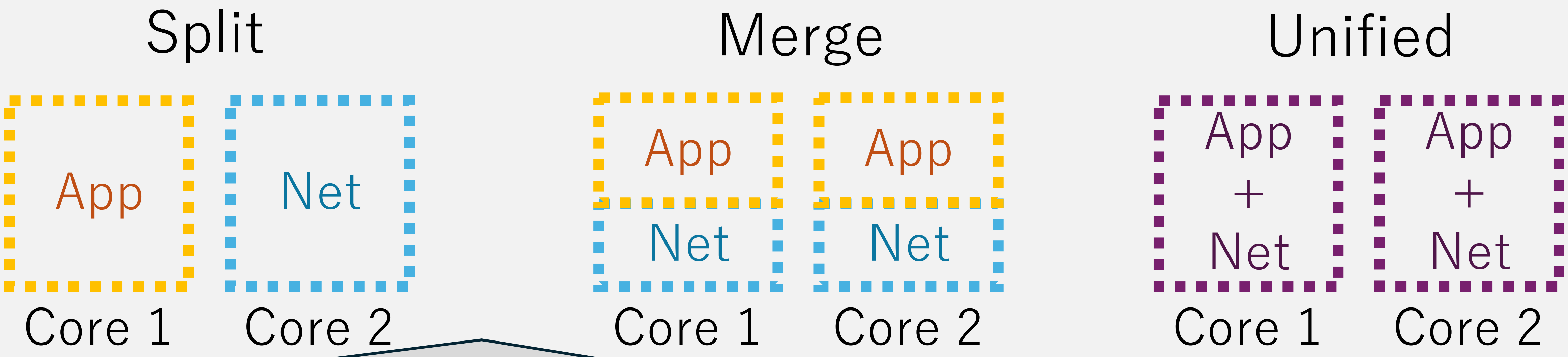
Limited Choices for CPU Core Assignment Models



Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models

Three potential CPU core assignment models

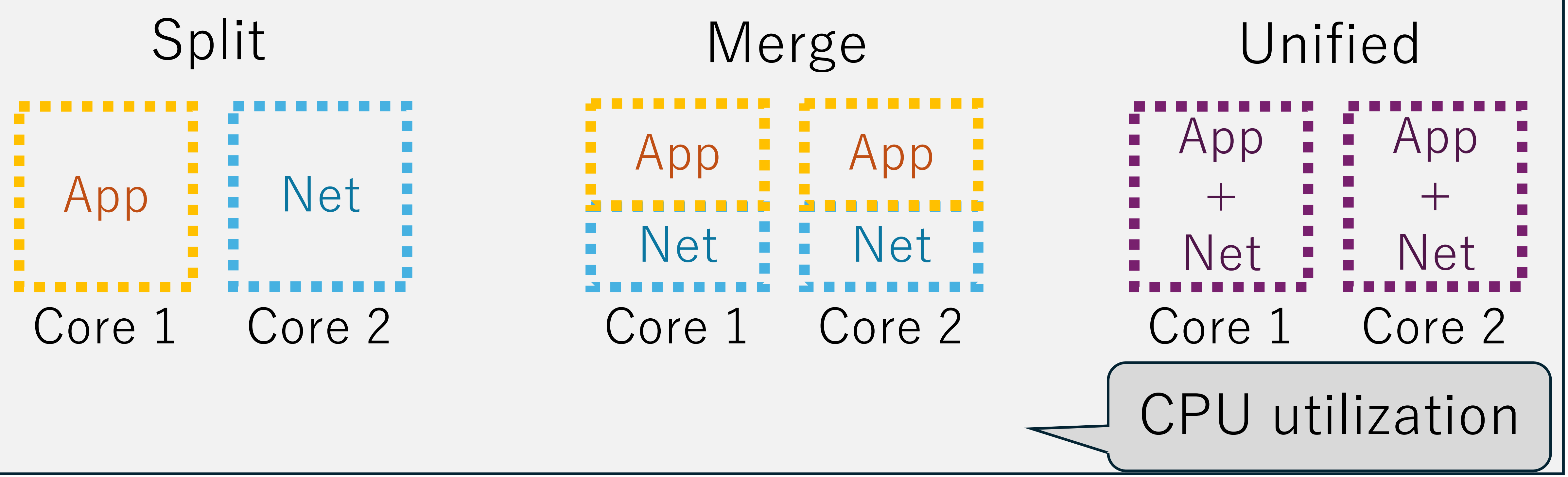


Each of them has different performance characteristics

Issues of Existing TCP/IP Stacks

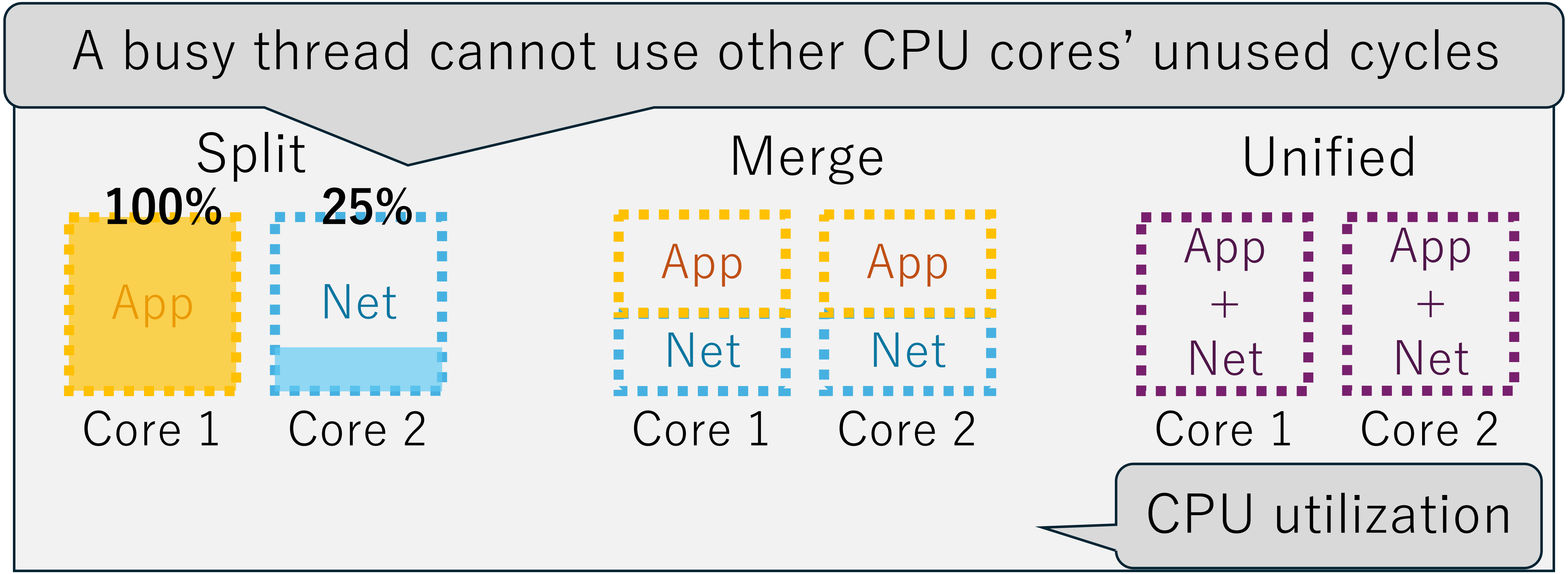
Limited Choices for CPU Core Assignment Models

Three potential CPU core assignment models



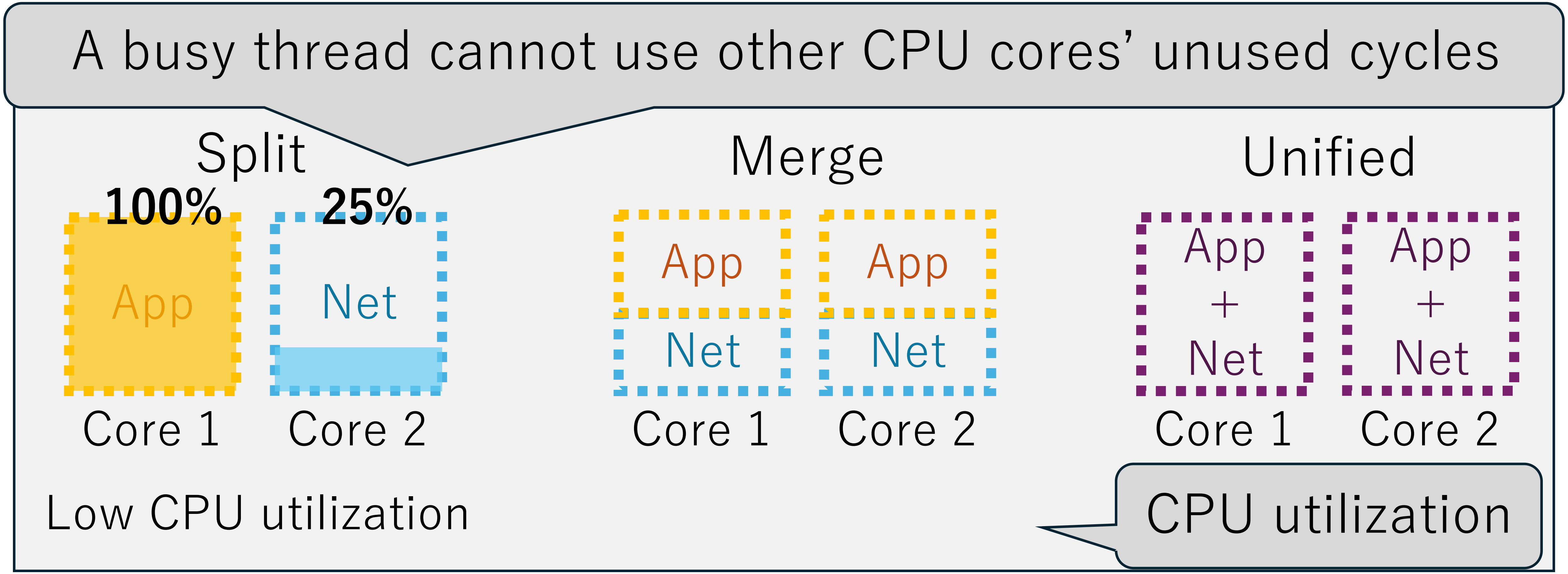
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



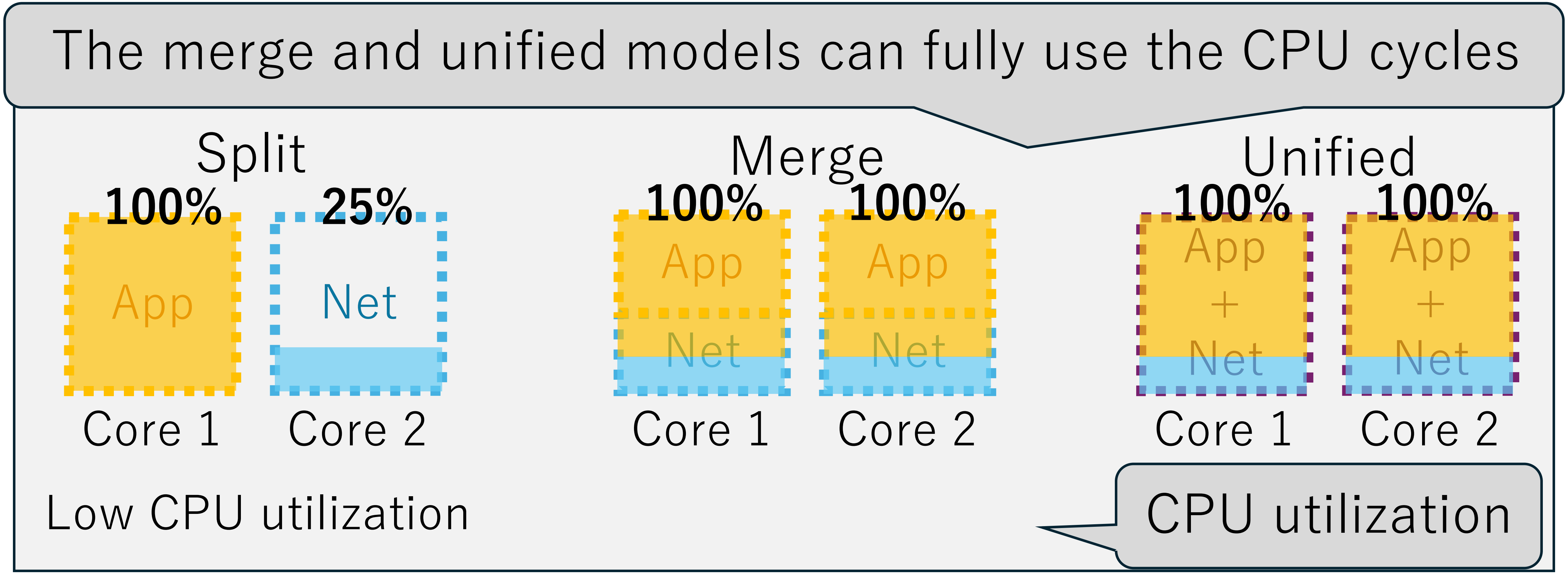
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



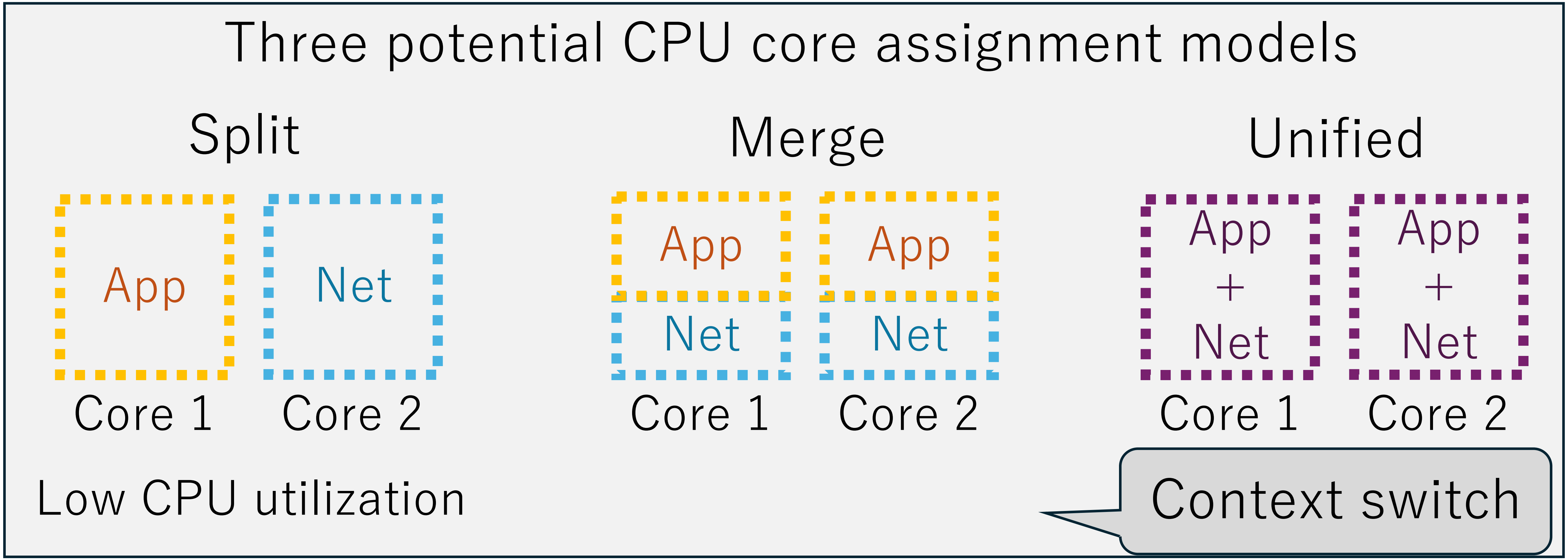
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



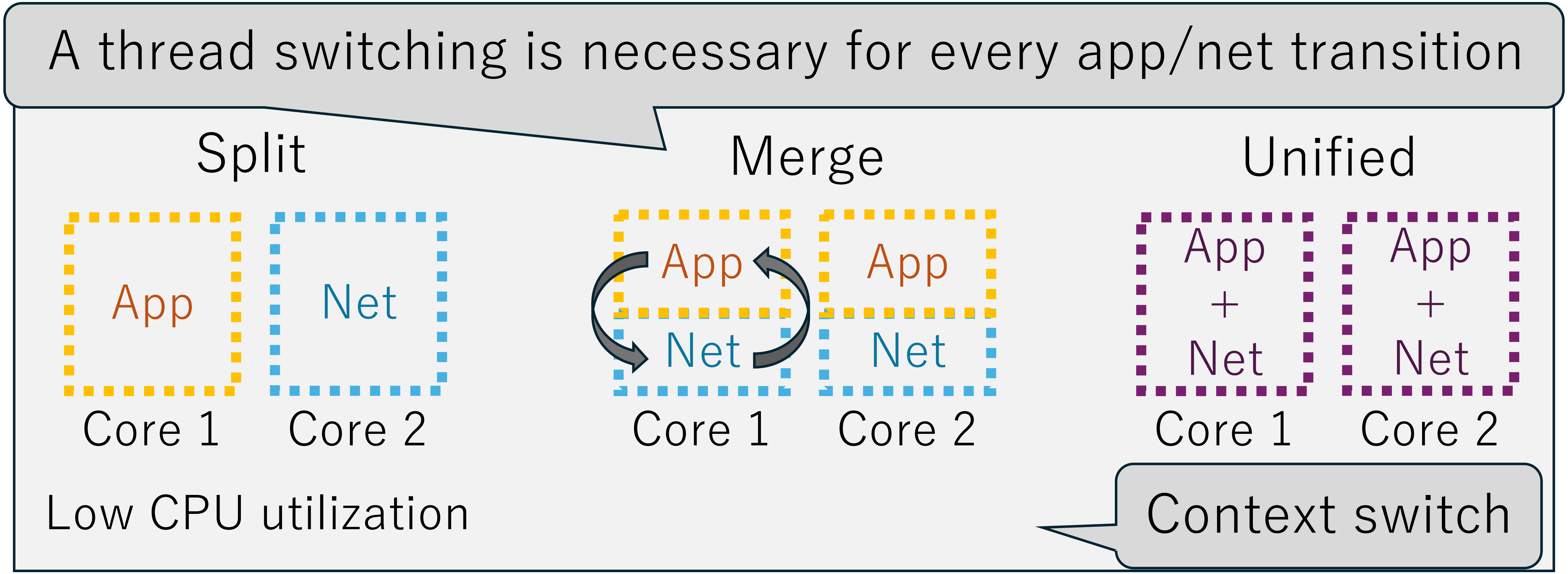
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



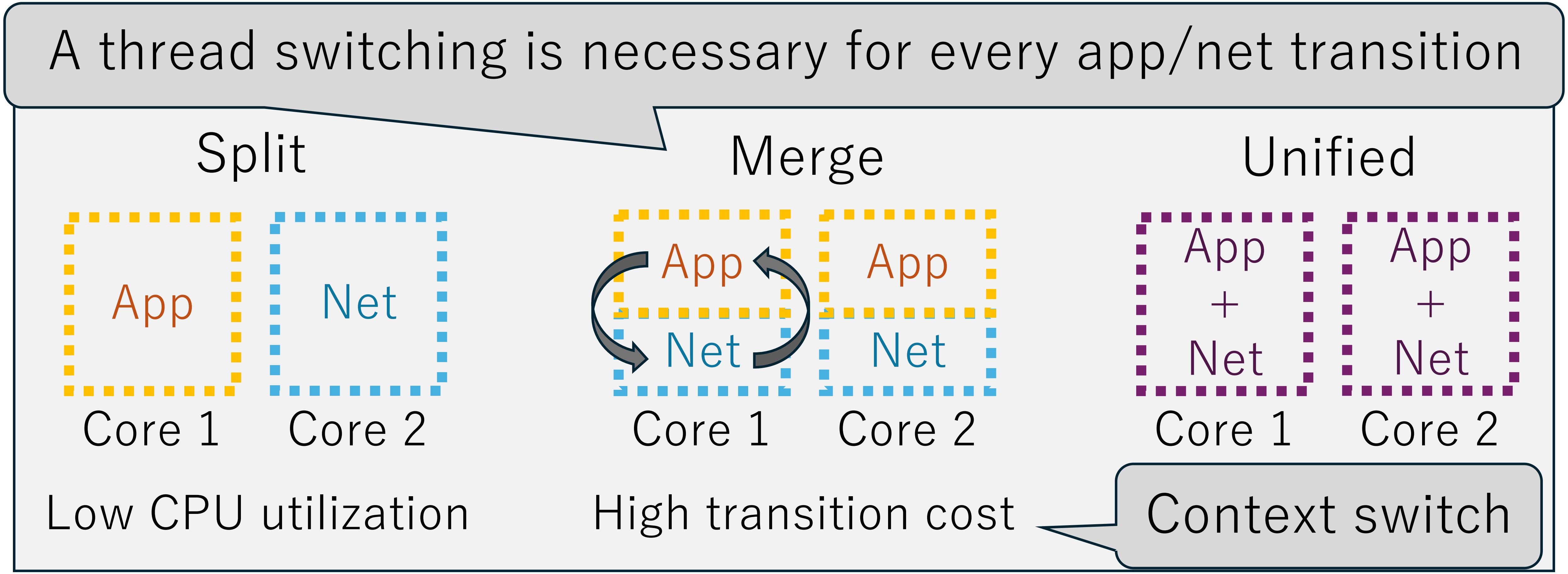
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



Issues of Existing TCP/IP Stacks

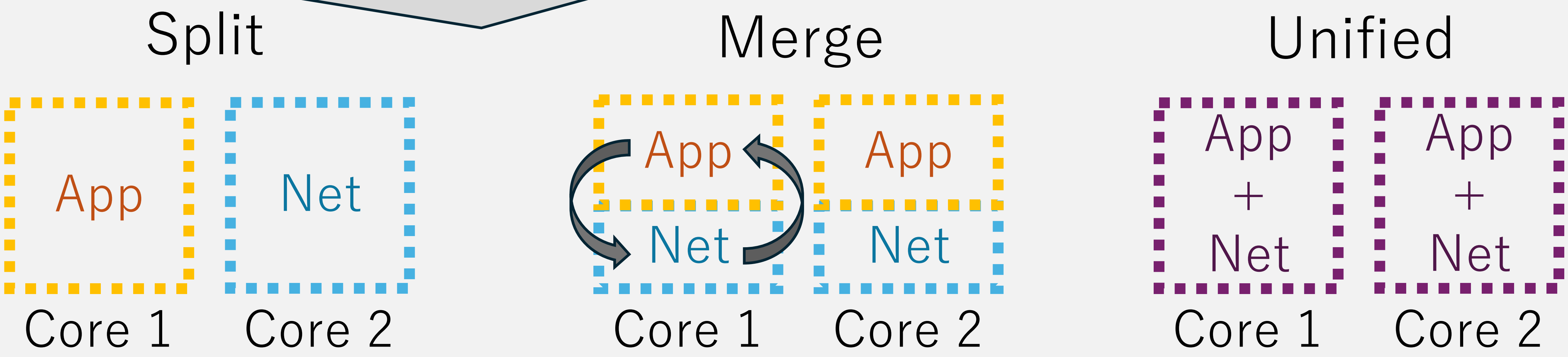
Limited Choices for CPU Core Assignment Models



Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models

The split and unified models are free from this transition cost



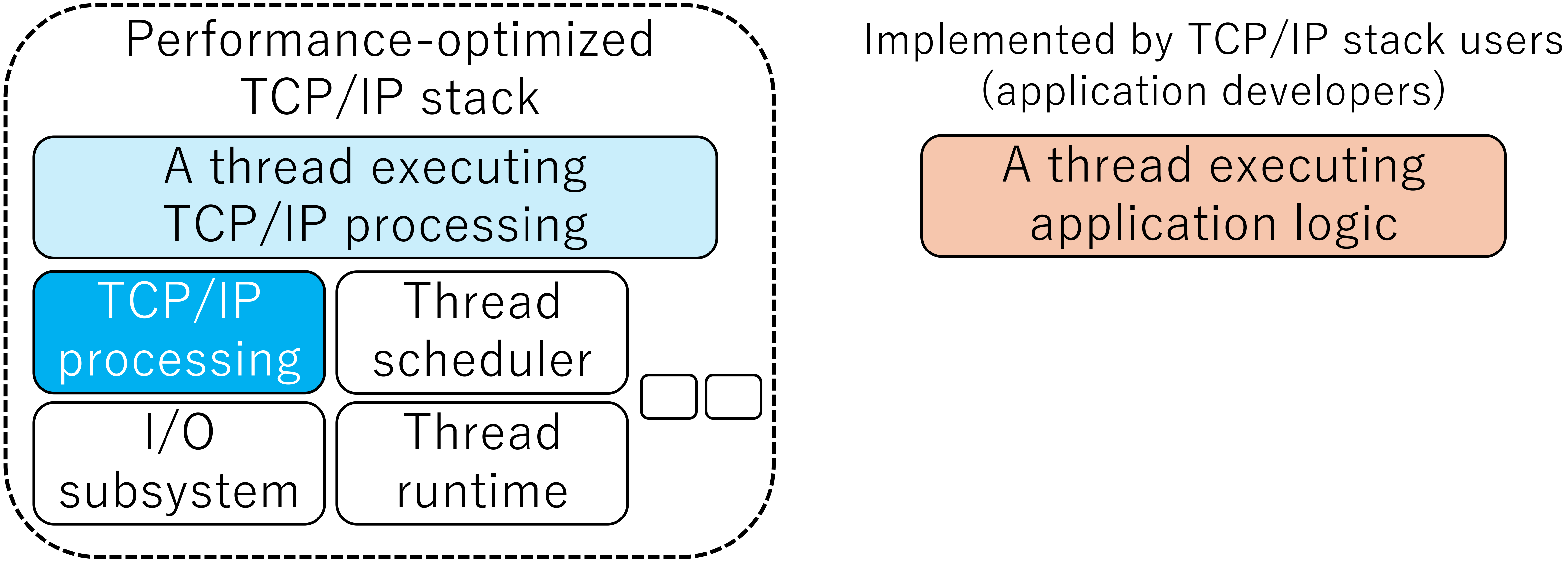
Low CPU utilization

High transition cost

Context switch

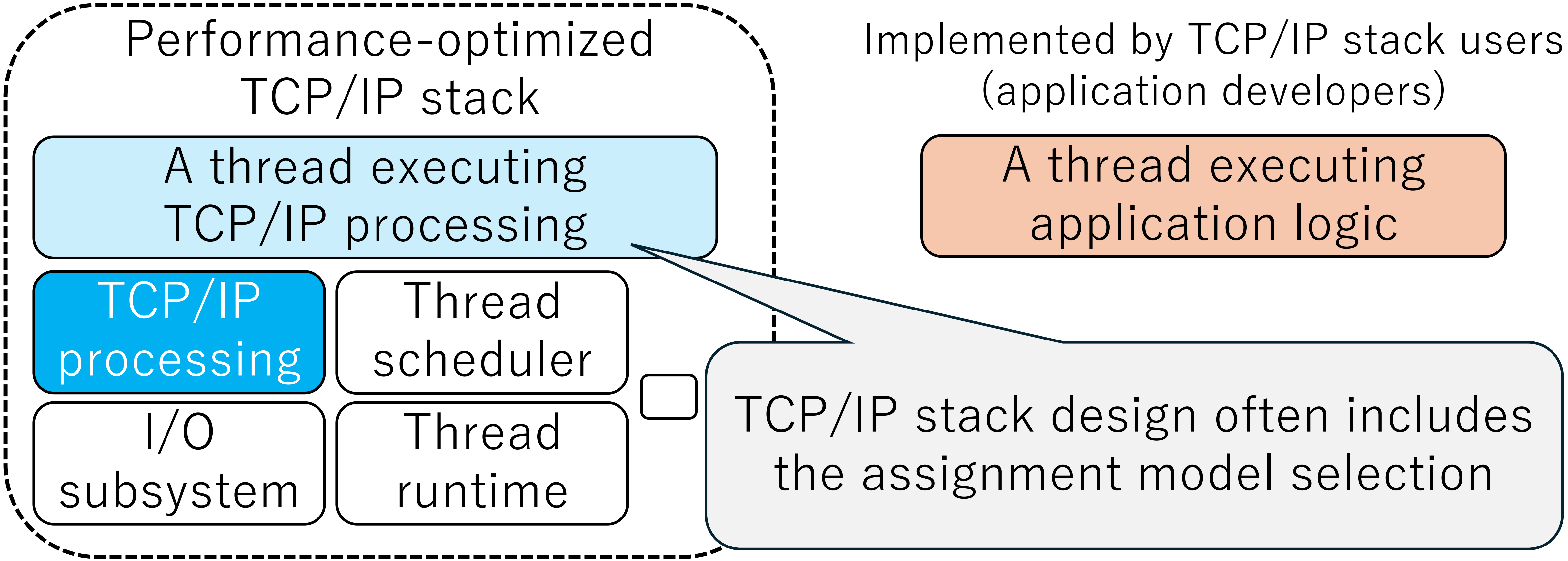
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



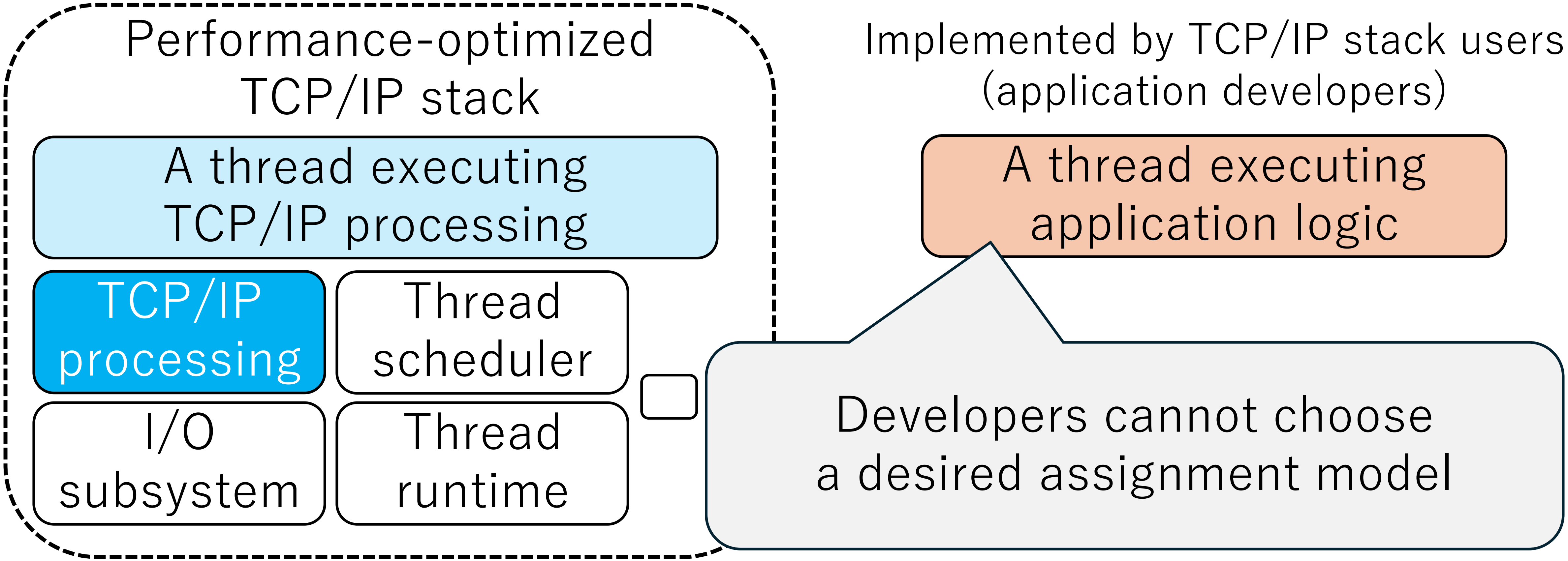
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models



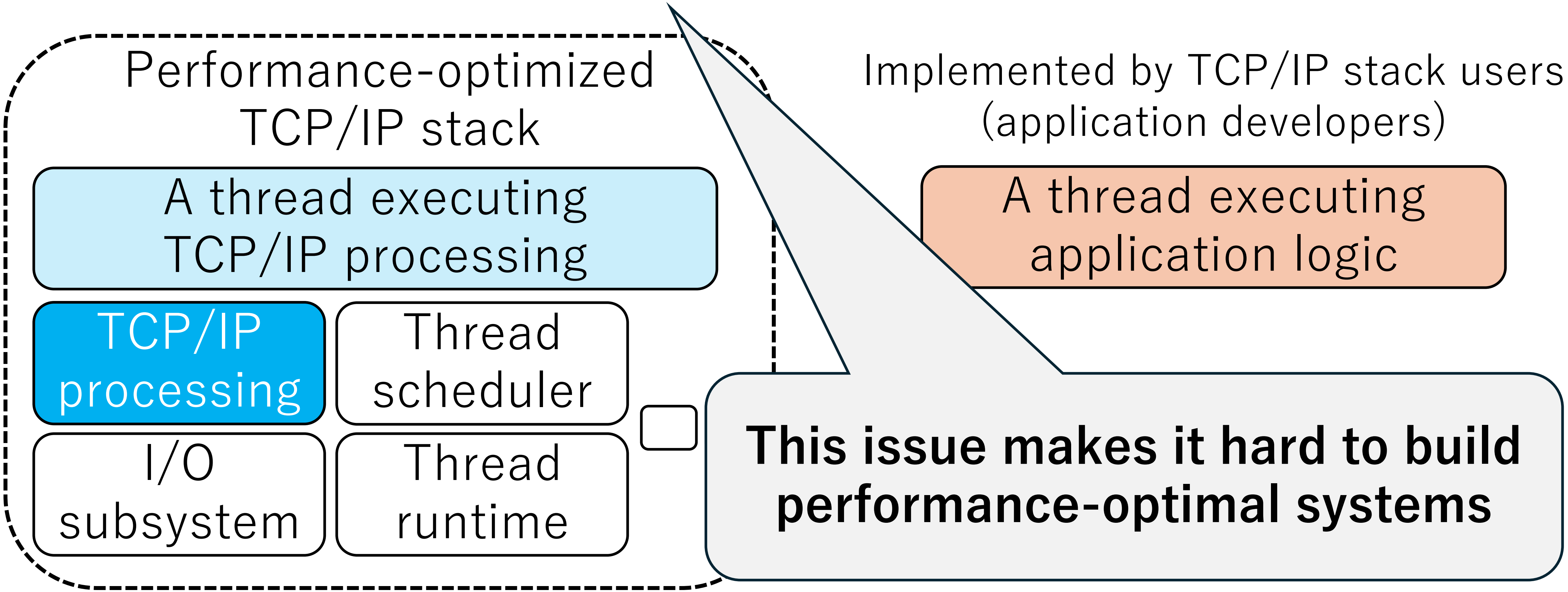
Issues of Existing TCP/IP Stacks

Limited Choices for CPU Core Assignment Models

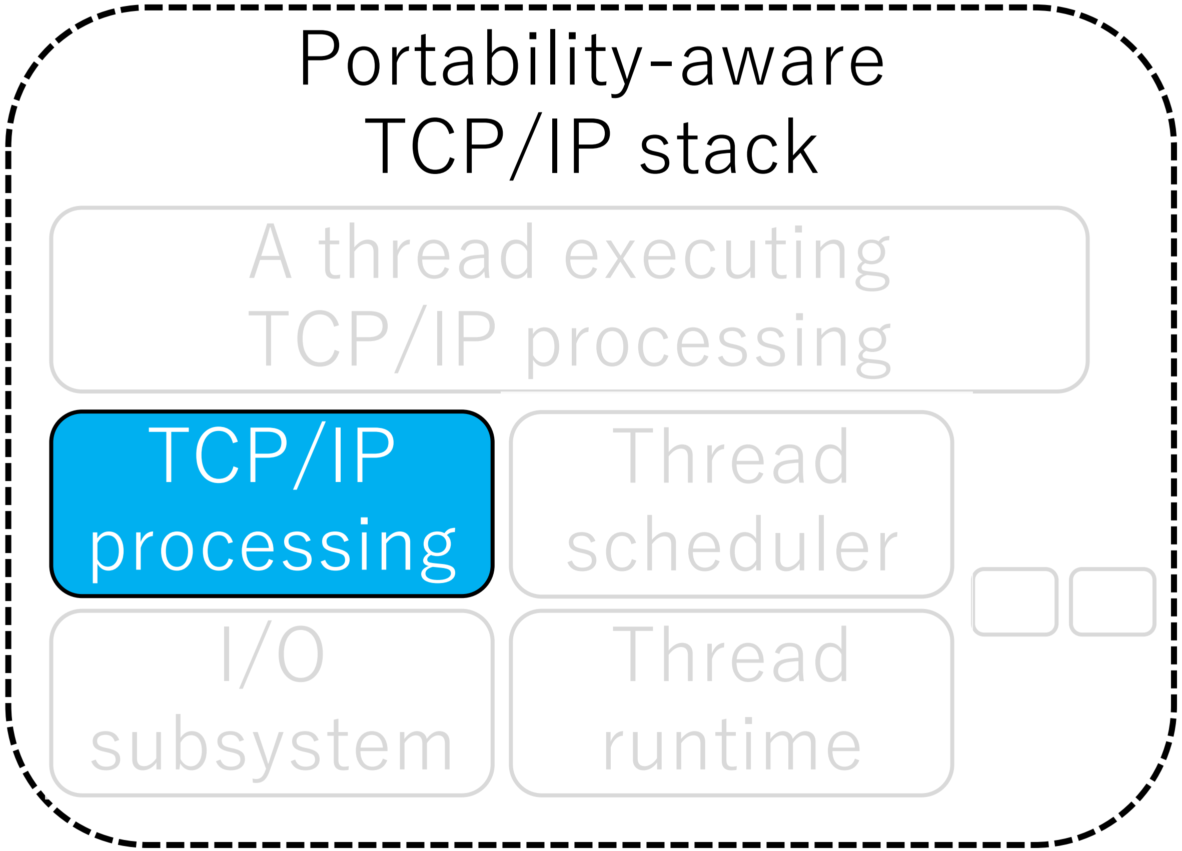


Issues of Existing TCP/IP Stacks

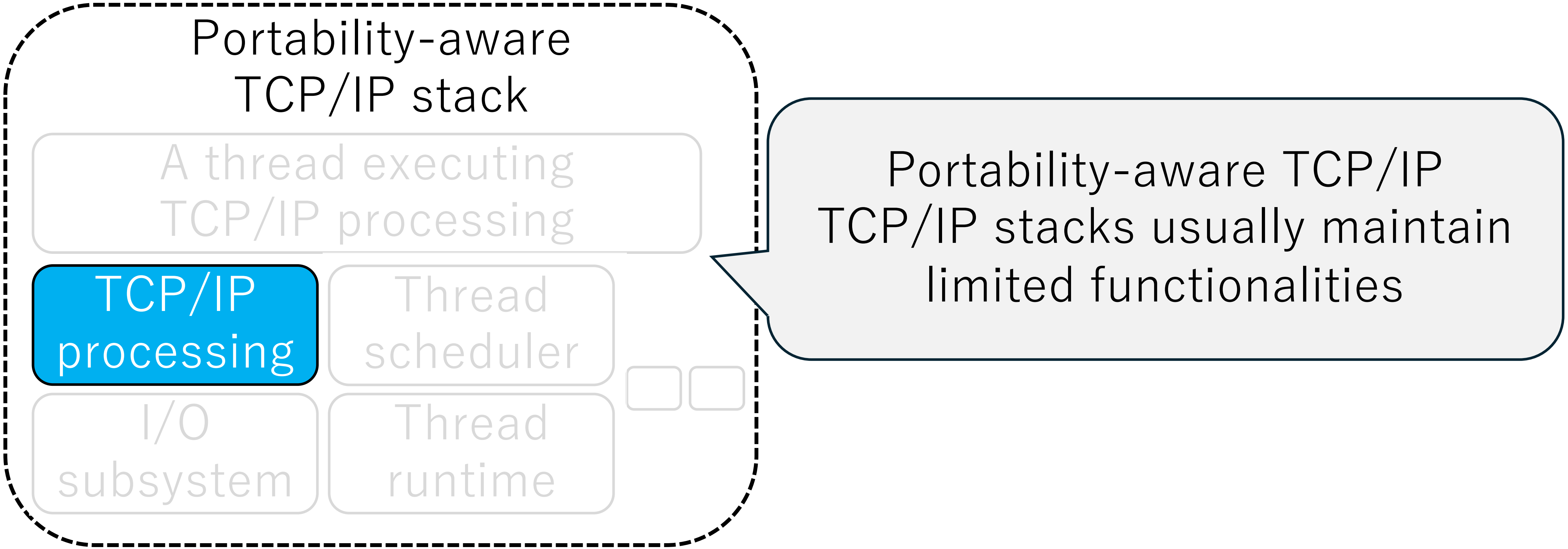
Limited Choices for CPU Core Assignment Models



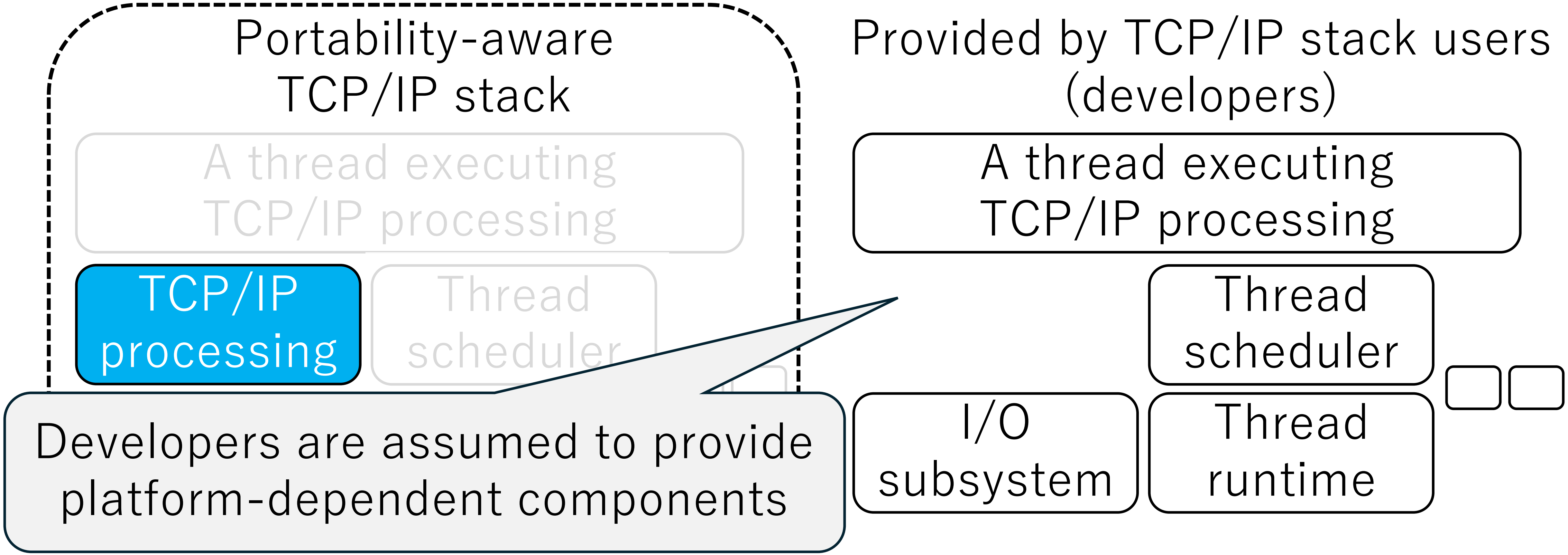
Issues of Existing TCP/IP Stacks



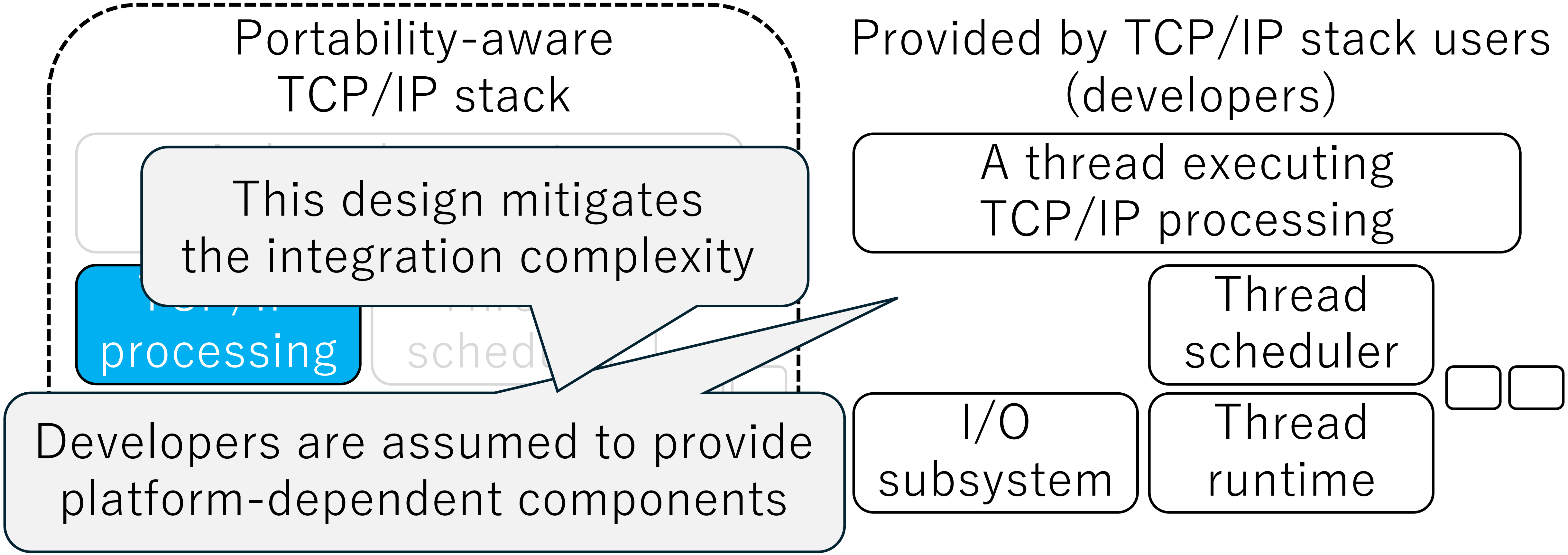
Issues of Existing TCP/IP Stacks



Issues of Existing TCP/IP Stacks

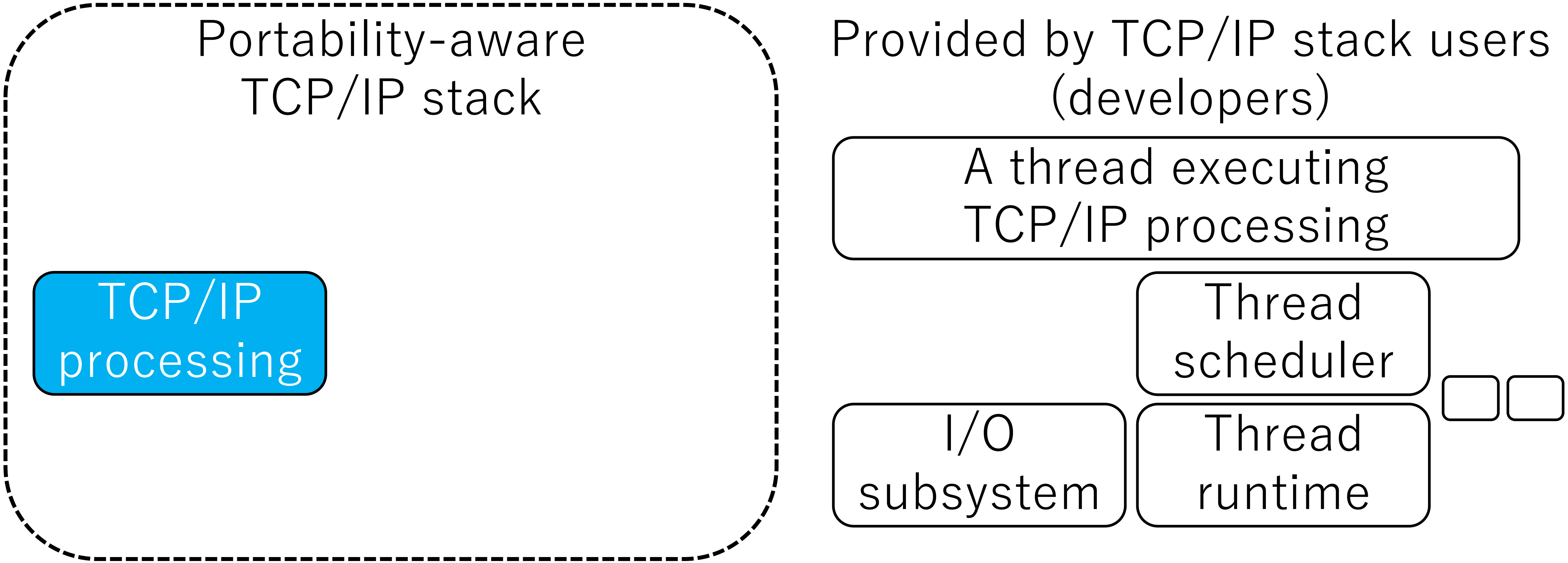


Issues of Existing TCP/IP Stacks



Issues of Existing TCP/IP Stacks

Not sufficiently aware of performance-critical factors



Issues of Existing TCP/IP Stacks

Not sufficiently aware of performance-critical factors

Portability-aware TCP/IP stack

Provided by TCP/IP stack users (developers)

A thread executing TCP/IP processing

TCP/IP processing

- Unaware of NIC offloading features

Issues of Existing TCP/IP Stacks

Not sufficiently aware of performance-critical factors

Portability-aware TCP/IP stack

Provided by TCP/IP stack users (developers)

A thread executing TCP/IP processing

TCP/IP processing

- Unaware of NIC offloading features
- Lack of zero-copy I/O capability

Issues of Existing TCP/IP Stacks

Not sufficiently aware of performance-critical factors

Portability-aware TCP/IP stack

TCP/IP processing

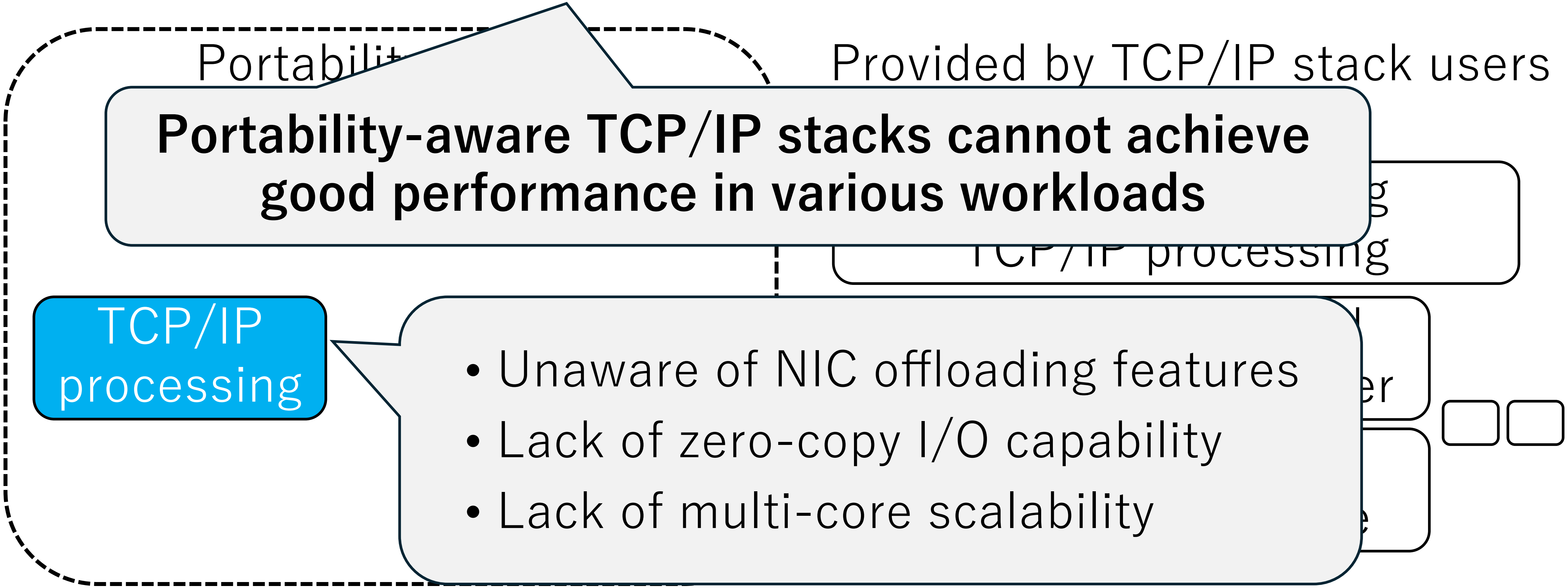
Provided by TCP/IP stack users (developers)

A thread executing TCP/IP processing

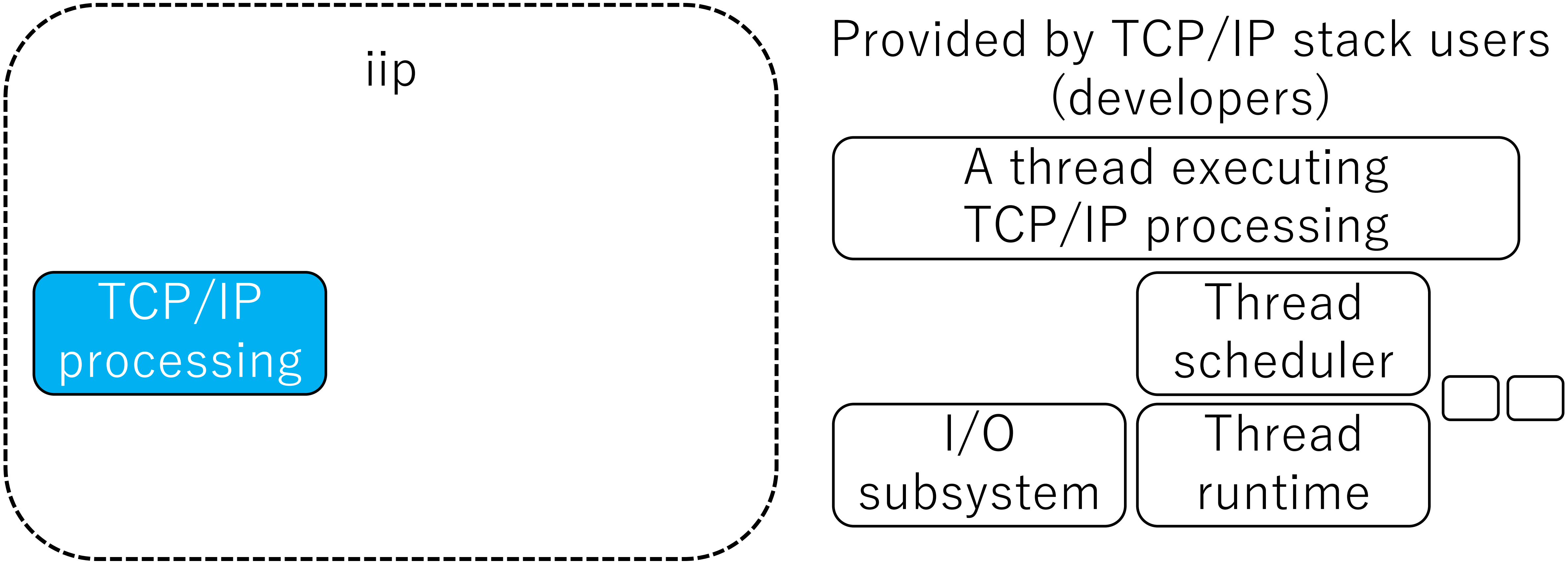
- Unaware of NIC offloading features
- Lack of zero-copy I/O capability
- Lack of multi-core scalability

Issues of Existing TCP/IP Stacks

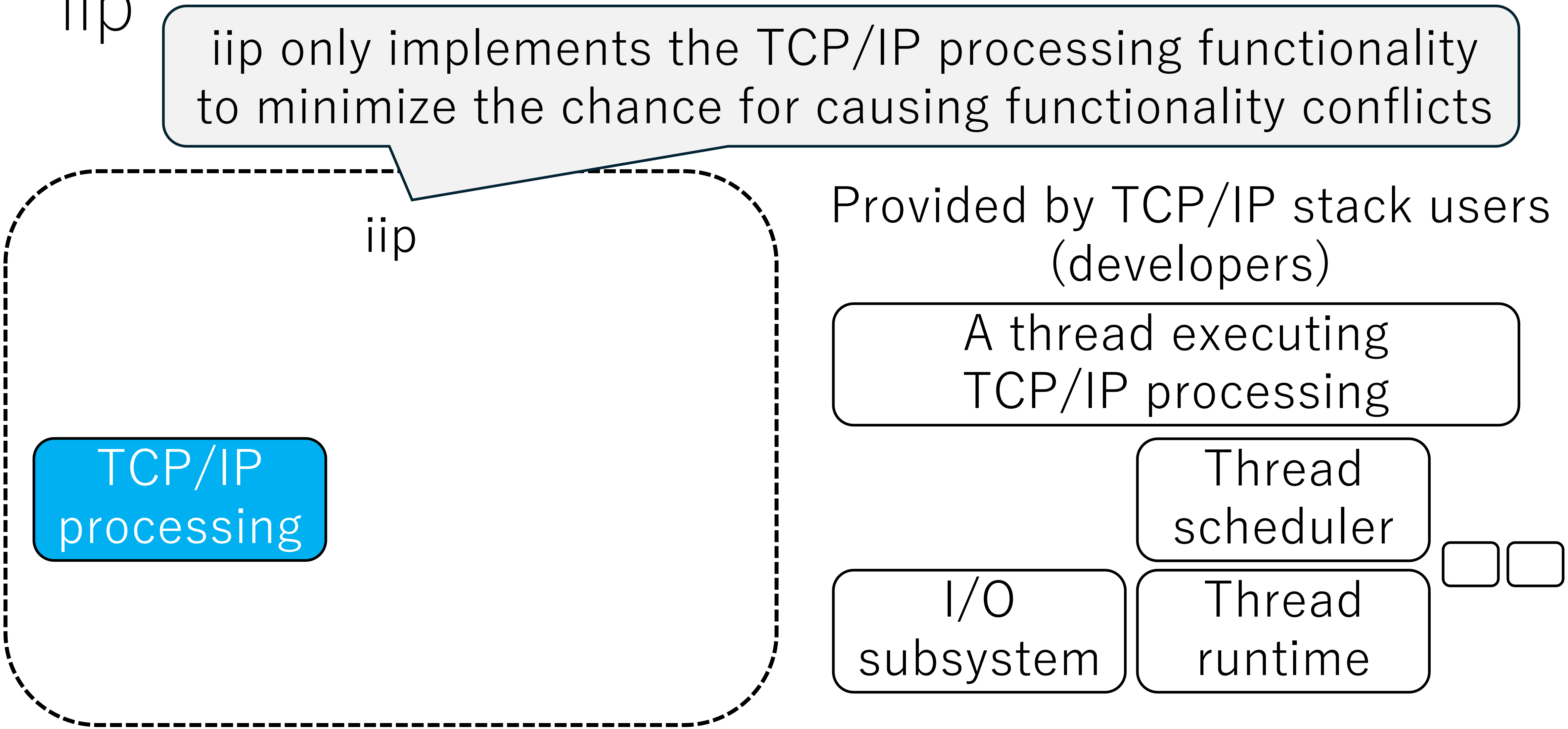
Not sufficiently aware of performance-critical factors



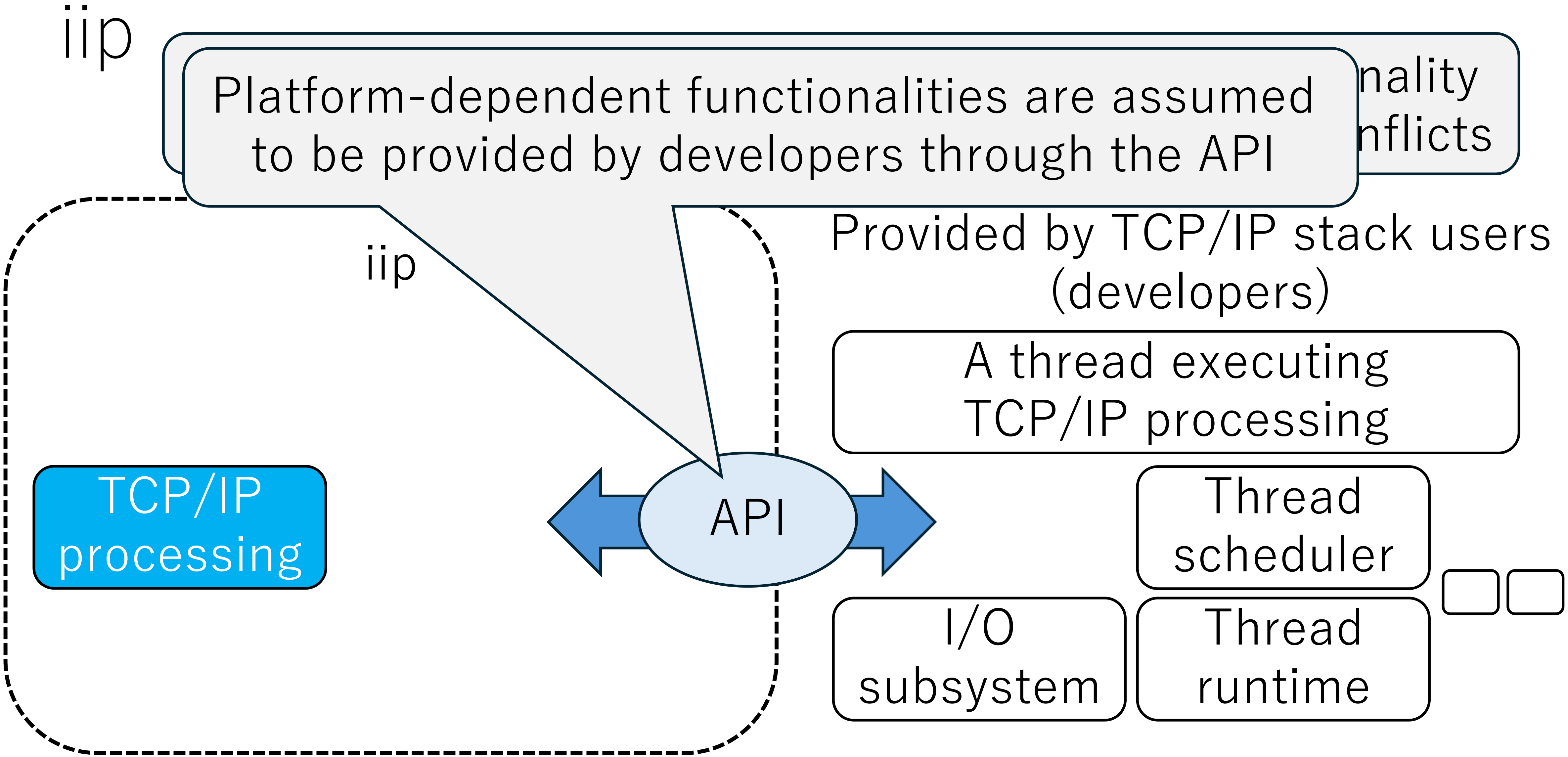
iip


iip
TCP/IP processing
Provided by TCP/IP stack users (developers)
A thread executing TCP/IP processing
Thread scheduler
I/O subsystem
Thread runtime

iip

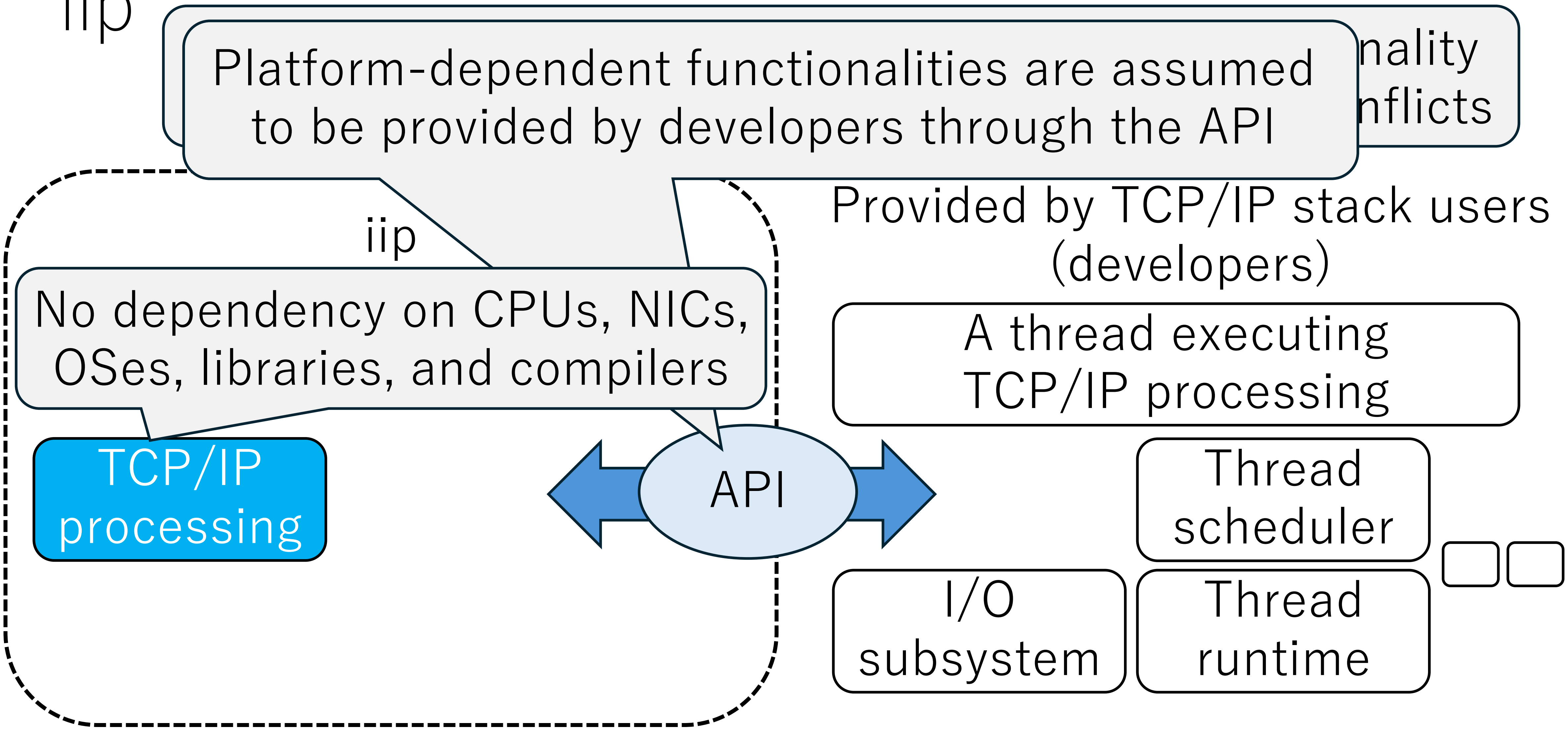


iip only implements the TCP/IP processing functionality to minimize the chance for causing functionality conflicts
iip
TCP/IP processing
Provided by TCP/IP stack users (developers)
A thread executing TCP/IP processing
Thread scheduler
I/O subsystem
Thread runtime

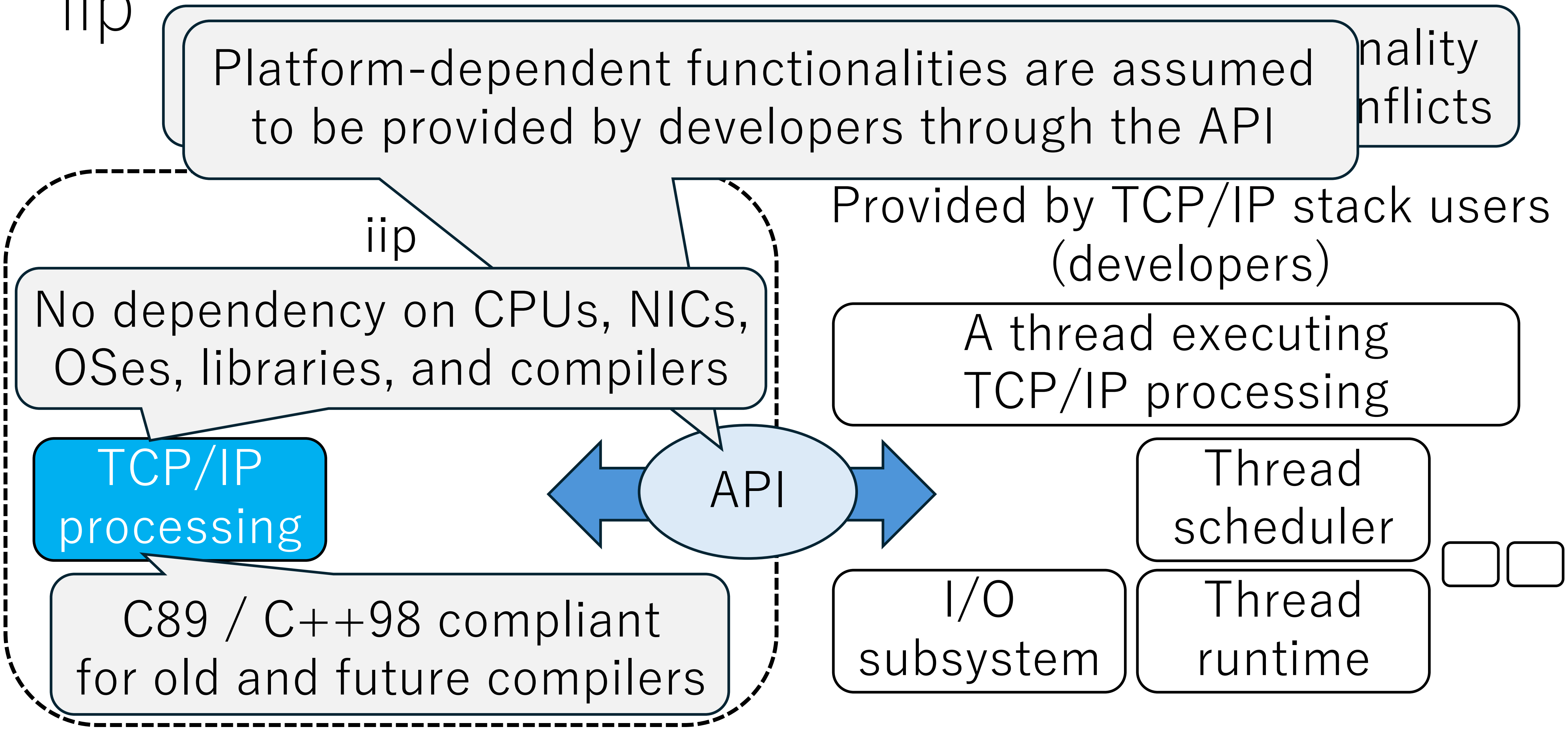


https://github.com/yasukata/iip

iip

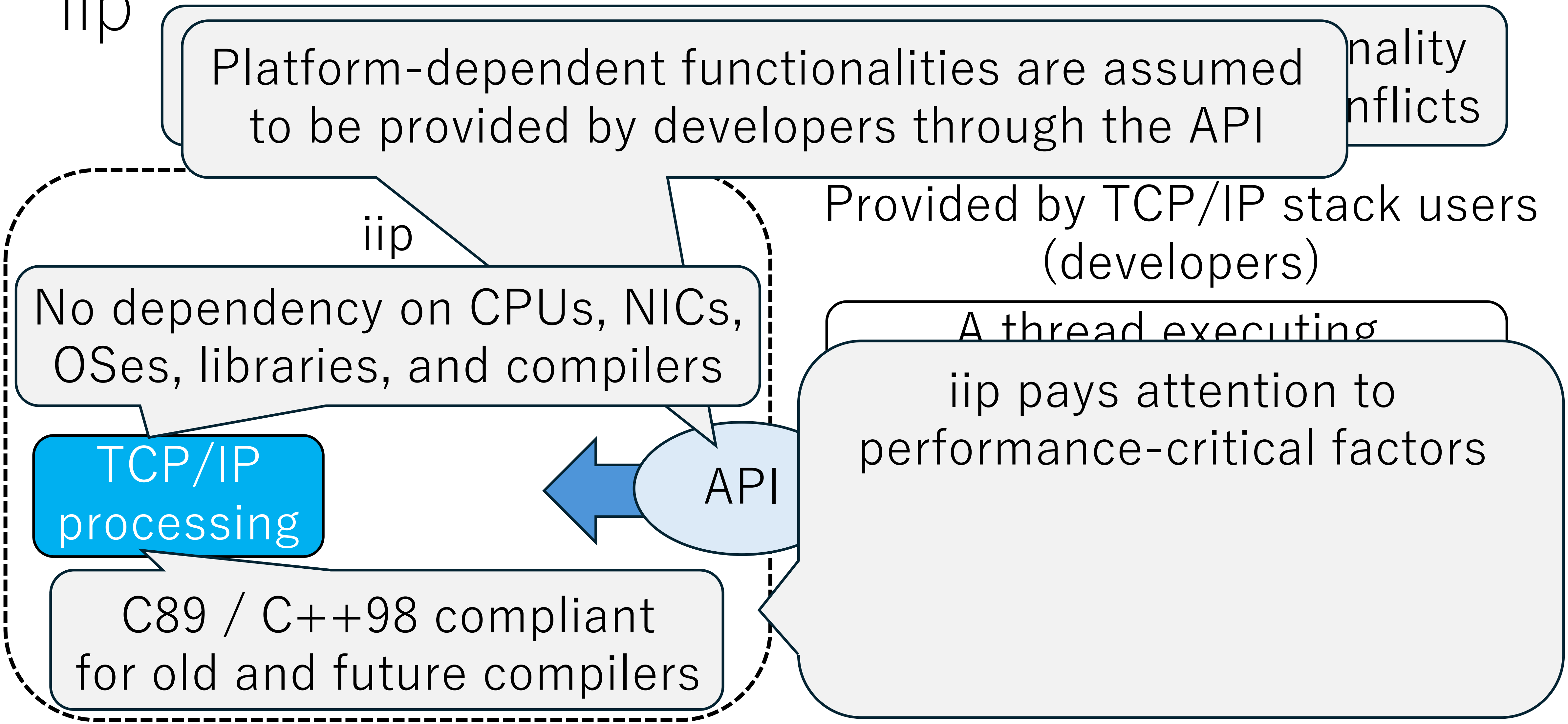

Platform-dependent functionalities are assumed to be provided by developers through the API
iip
No dependency on CPUs, NICs, OSes, libraries, and compilers
TCP/IP processing
API
Provided by TCP/IP stack users (developers)
A thread executing TCP/IP processing
Thread scheduler
I/O subsystem
Thread runtime

iip

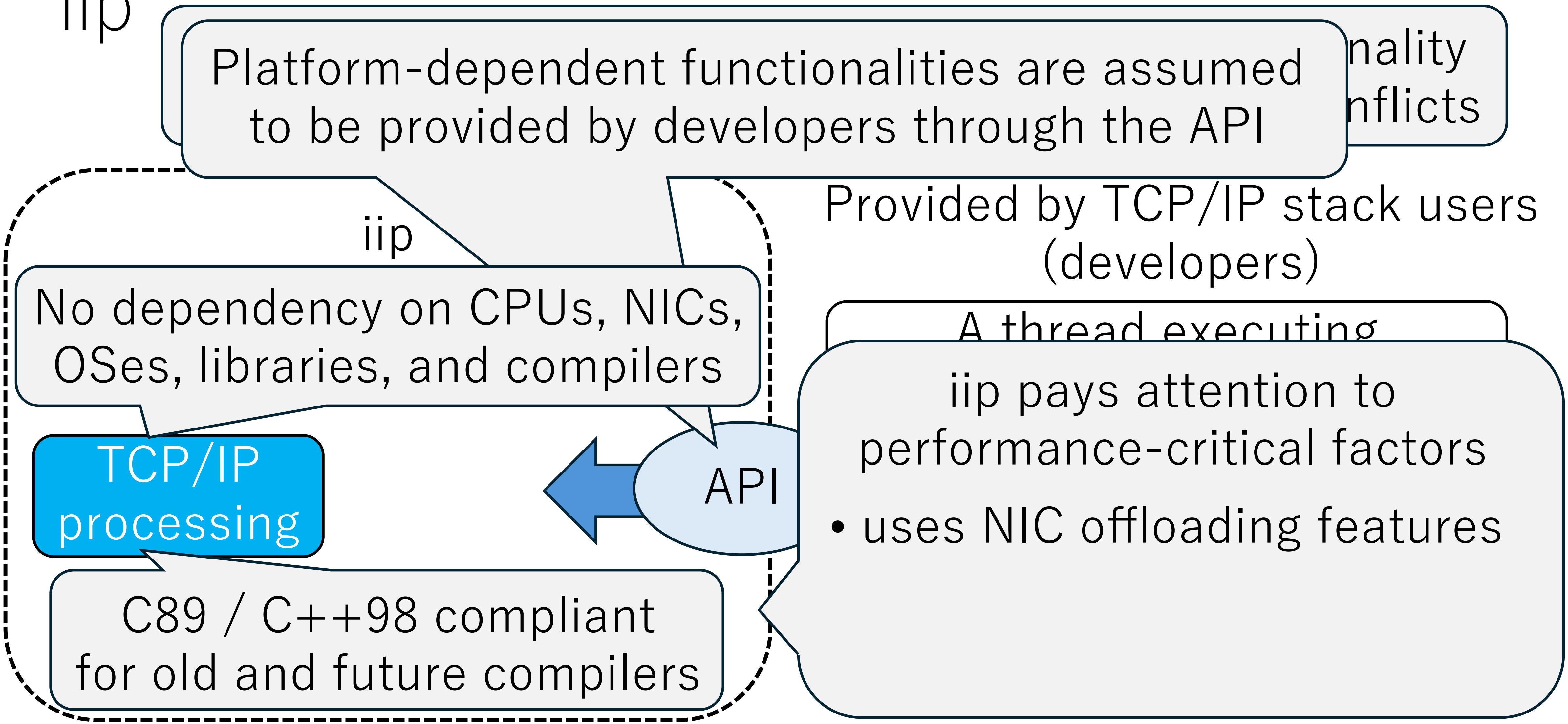


nality
nflicts
Platform-dependent functionalities are assumed to be provided by developers through the API
iip
No dependency on CPUs, NICs, OSes, libraries, and compilers
TCP/IP processing
API
C89 / C++98 compliant for old and future compilers
Provided by TCP/IP stack users (developers)
A thread executing TCP/IP processing
Thread scheduler
I/O subsystem
Thread runtime

iip


nality
nflicts
Platform-dependent functionalities are assumed to be provided by developers through the API
iip
No dependency on CPUs, NICs, OSes, libraries, and compilers
TCP/IP processing
API
C89 / C++98 compliant for old and future compilers
Provided by TCP/IP stack users (developers)
A thread executing
iip pays attention to performance-critical factors

iip


nality
nflicts
Platform-dependent functionalities are assumed to be provided by developers through the API
iip
No dependency on CPUs, NICs, OSes, libraries, and compilers
TCP/IP processing
API
C89 / C++98 compliant for old and future compilers
Provided by TCP/IP stack users (developers)
A thread executing
iip pays attention to performance-critical factors
• uses NIC offloading features

iip

nality
nflicts

Platform-dependent functionalities are assumed to be provided by developers through the API

iip

No dependency on CPUs, NICs, OSes, libraries, and compilers

TCP/IP processing

API

C89 / C++98 compliant for old and future compilers

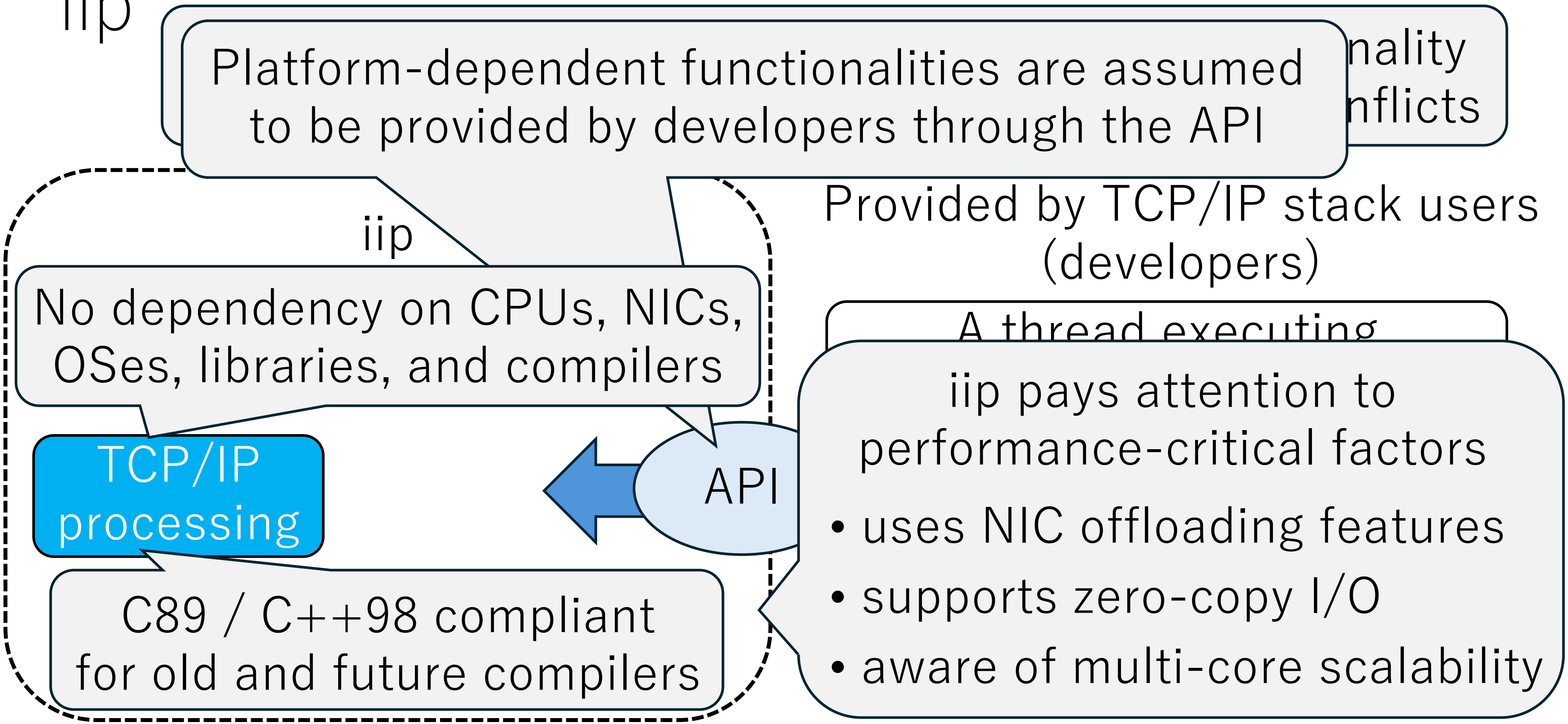
Provided by TCP/IP stack users (developers)

A thread executing

iip pays attention to performance-critical factors

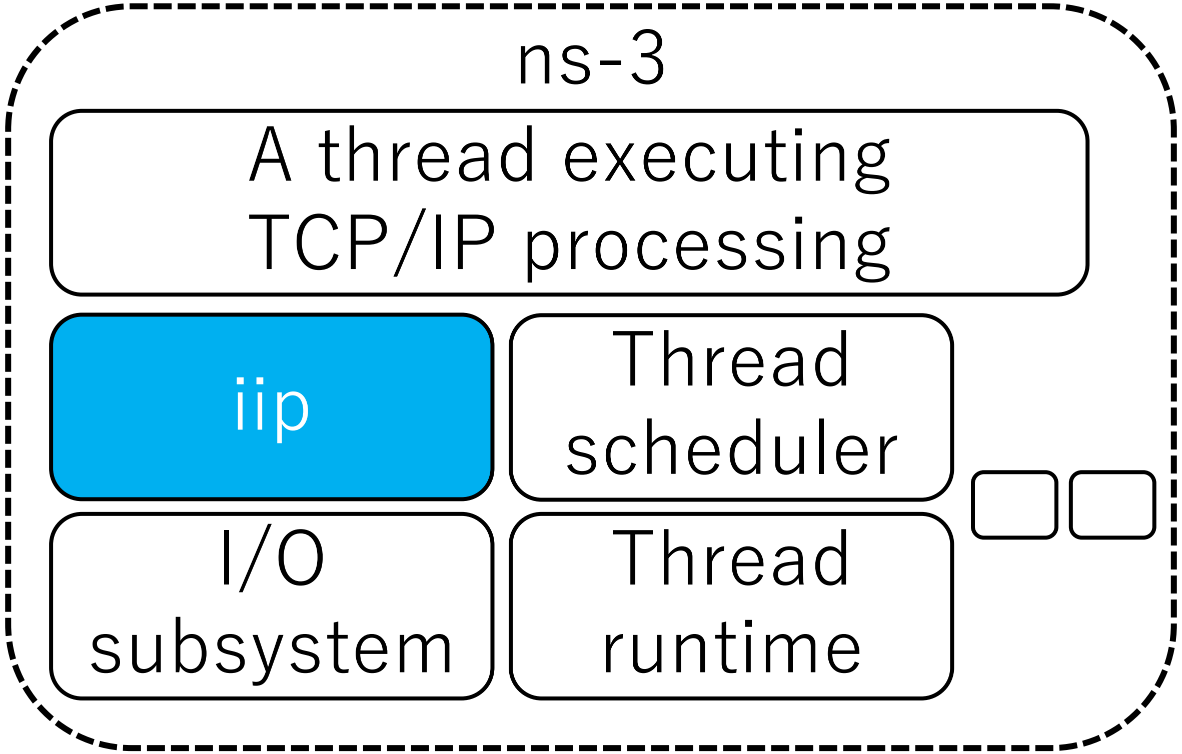
- uses NIC offloading features
- supports zero-copy I/O

iip


nality
nflicts
Platform-dependent functionalities are assumed to be provided by developers through the API
iip
Provided by TCP/IP stack users (developers)
A thread executing
No dependency on CPUs, NICs, OSes, libraries, and compilers
TCP/IP processing
API
iip pays attention to performance-critical factors
• uses NIC offloading features
• supports zero-copy I/O
• aware of multi-core scalability
C89 / C++98 compliant for old and future compilers

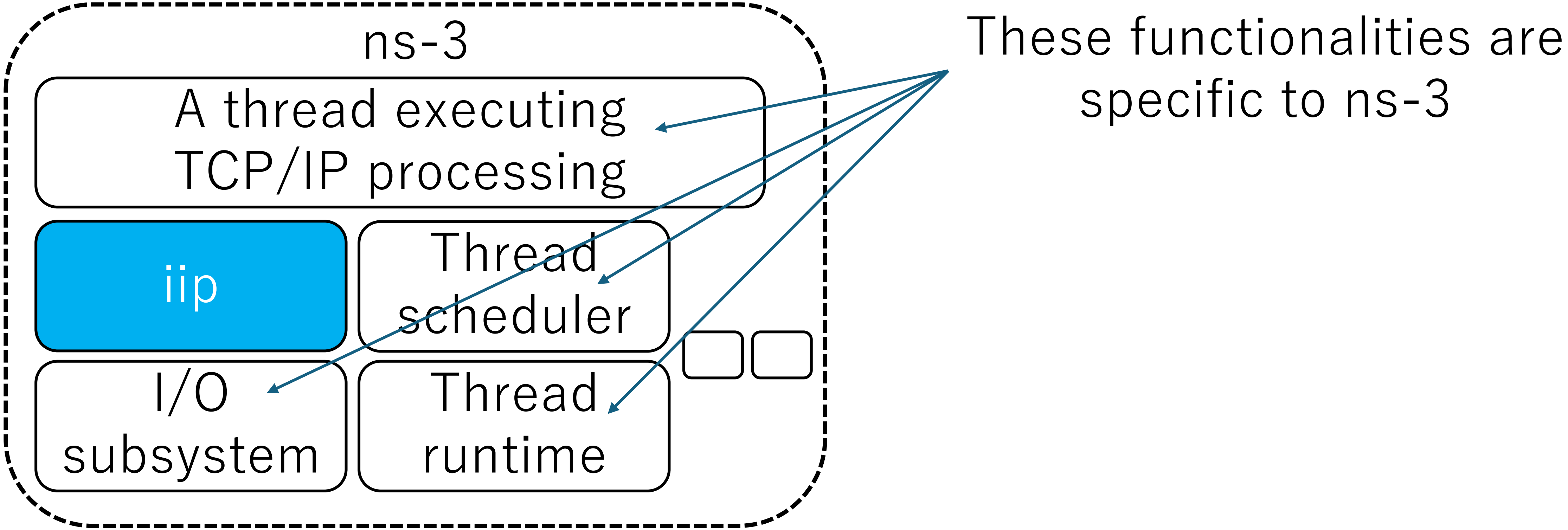
Integratability Benefits

- iip can be integrated into the ns-3 simulator written in C++
  - https://github.com/yasukata/iip-ns



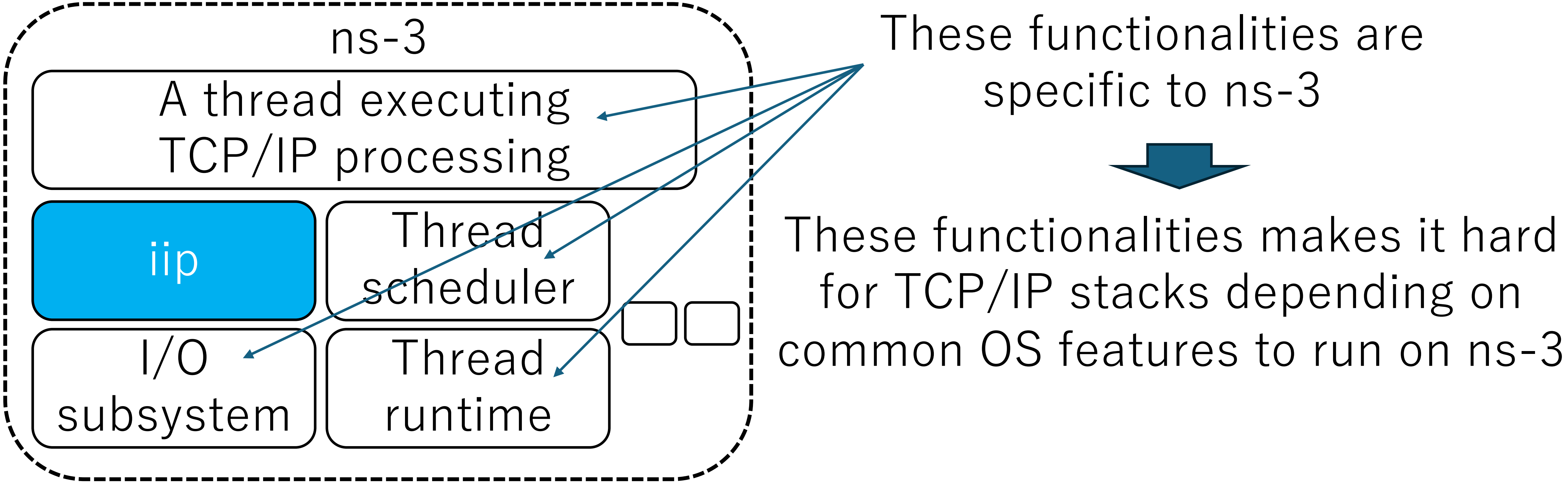
Integratability Benefits

- iip can be integrated into the ns-3 simulator written in C++
  - https://github.com/yasukata/iip-ns



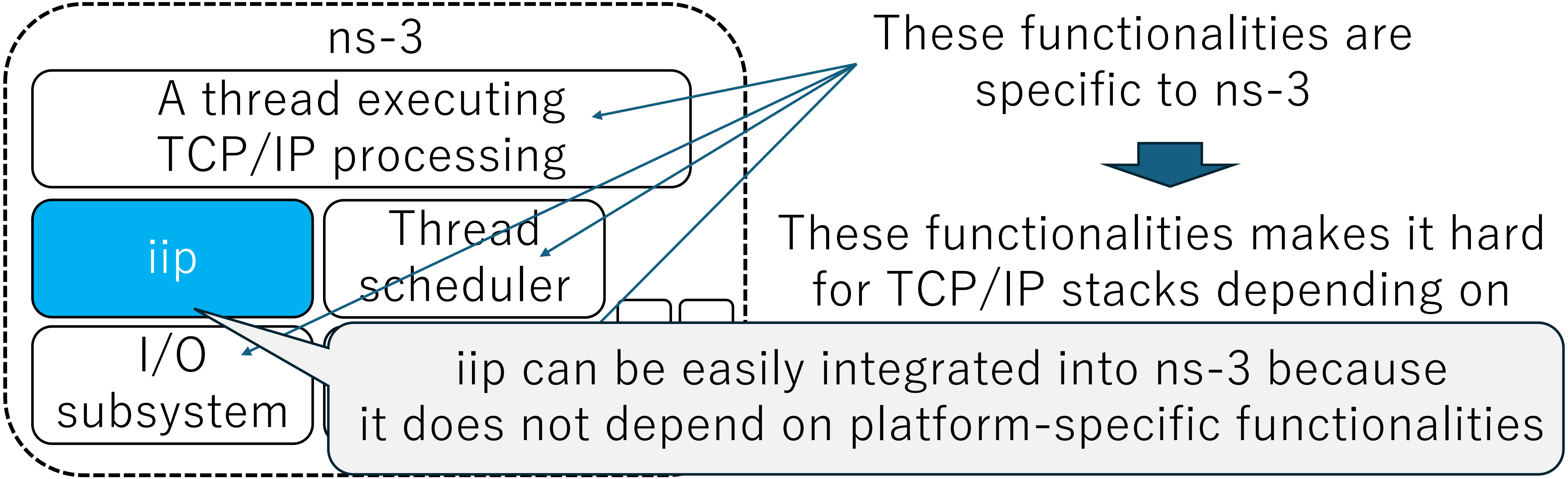
Integratability Benefits

- iip can be integrated into the ns-3 simulator written in C++
  - https://github.com/yasukata/iip-ns



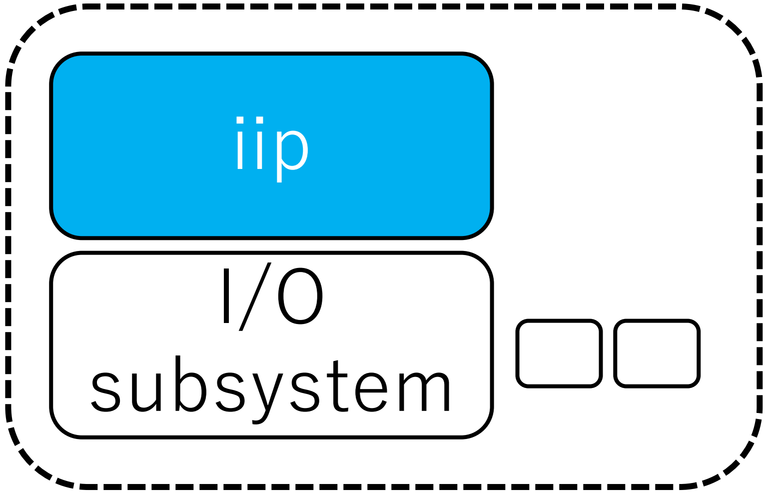
Integratability Benefits

- iip can be integrated into the ns-3 simulator written in C++
  - https://github.com/yasukata/iip-ns



Integratability Benefits

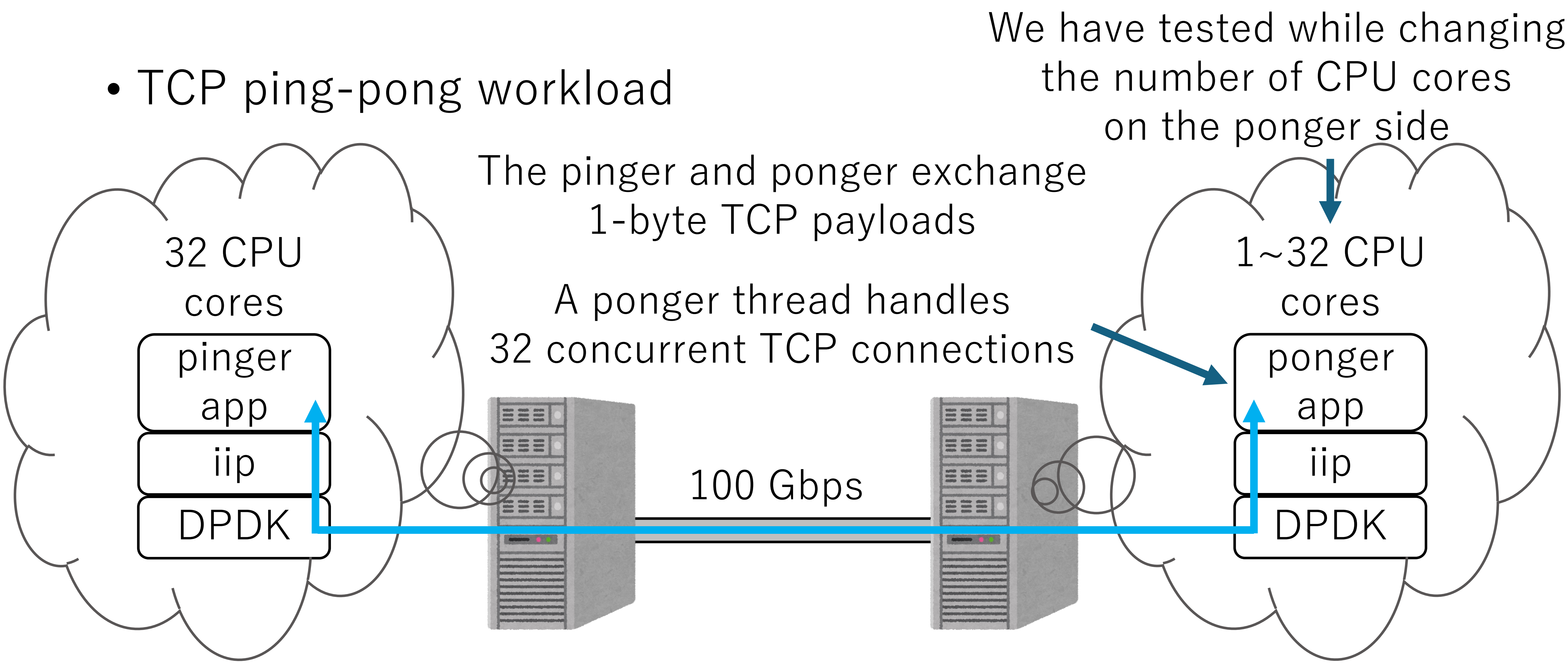
- iip runs on various I/O backends, including but not limited to:
  - DPDK (Linux): https://github.com/yasukata/iip-dpdk
  - AF_XDP (Linux): https://github.com/yasukata/iip-af_xdp
  - netmap (Linux/FreeBSD): https://github.com/yasukata/iip-netmap



iip does not depend on specific I/O subsystems

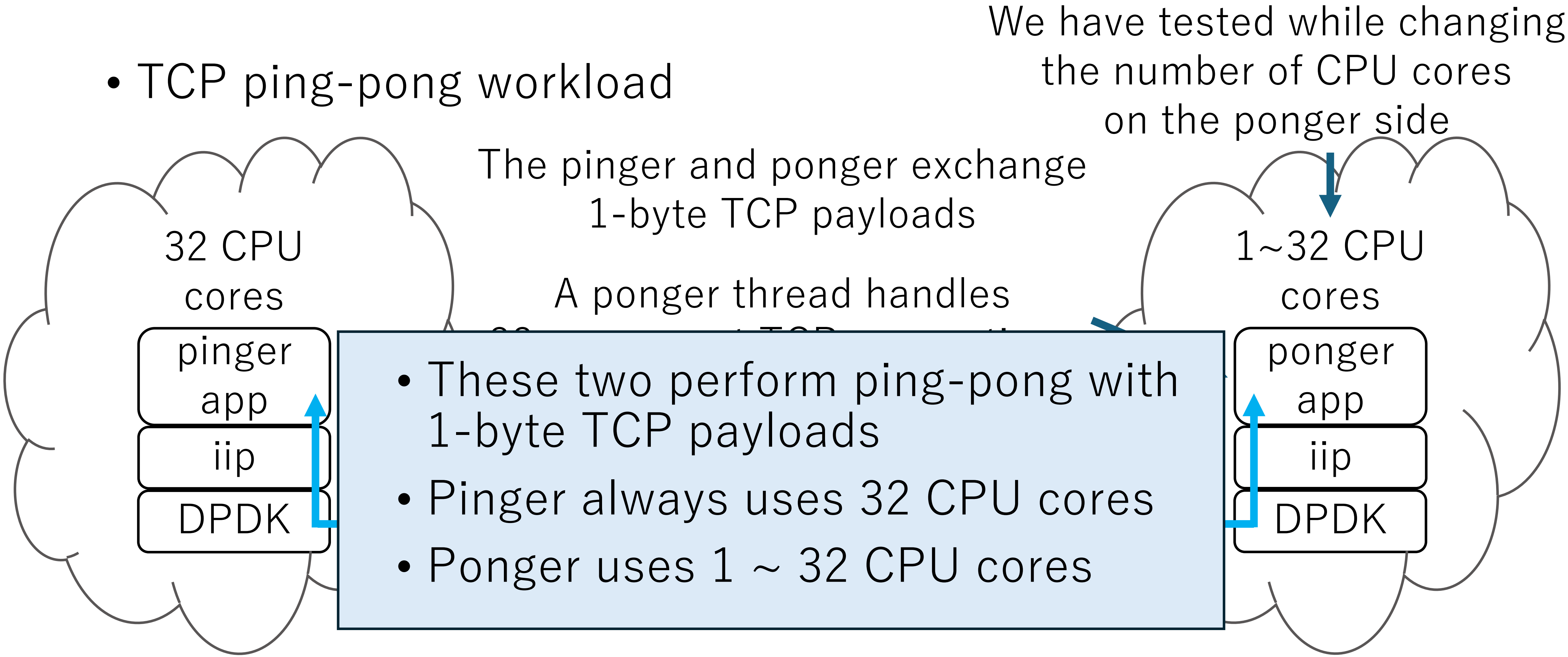
Evaluation: Small Message Exchange

- TCP ping-pong workload



Evaluation: Small Message Exchange

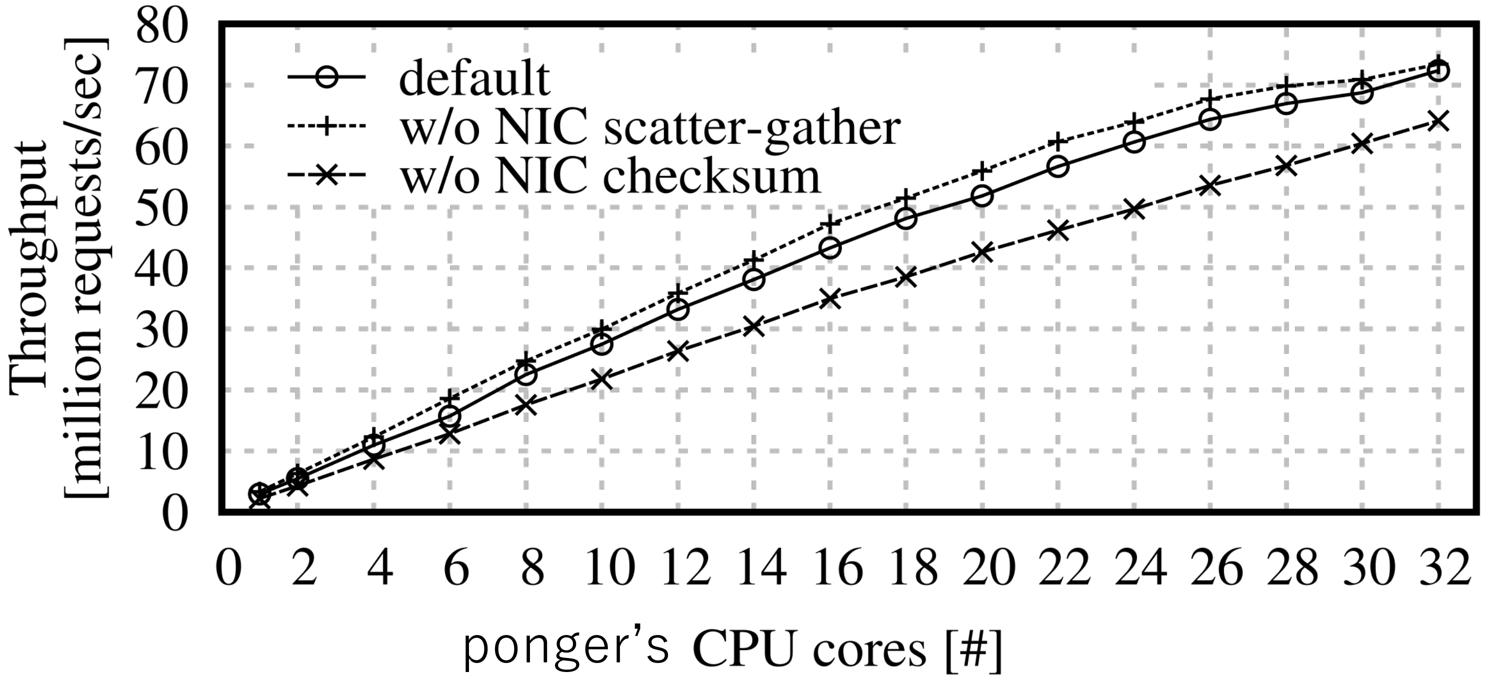
- TCP ping-pong workload



- These two perform ping-pong with 1-byte TCP payloads
- Pinger always uses 32 CPU cores
- Ponger uses 1 ~ 32 CPU cores

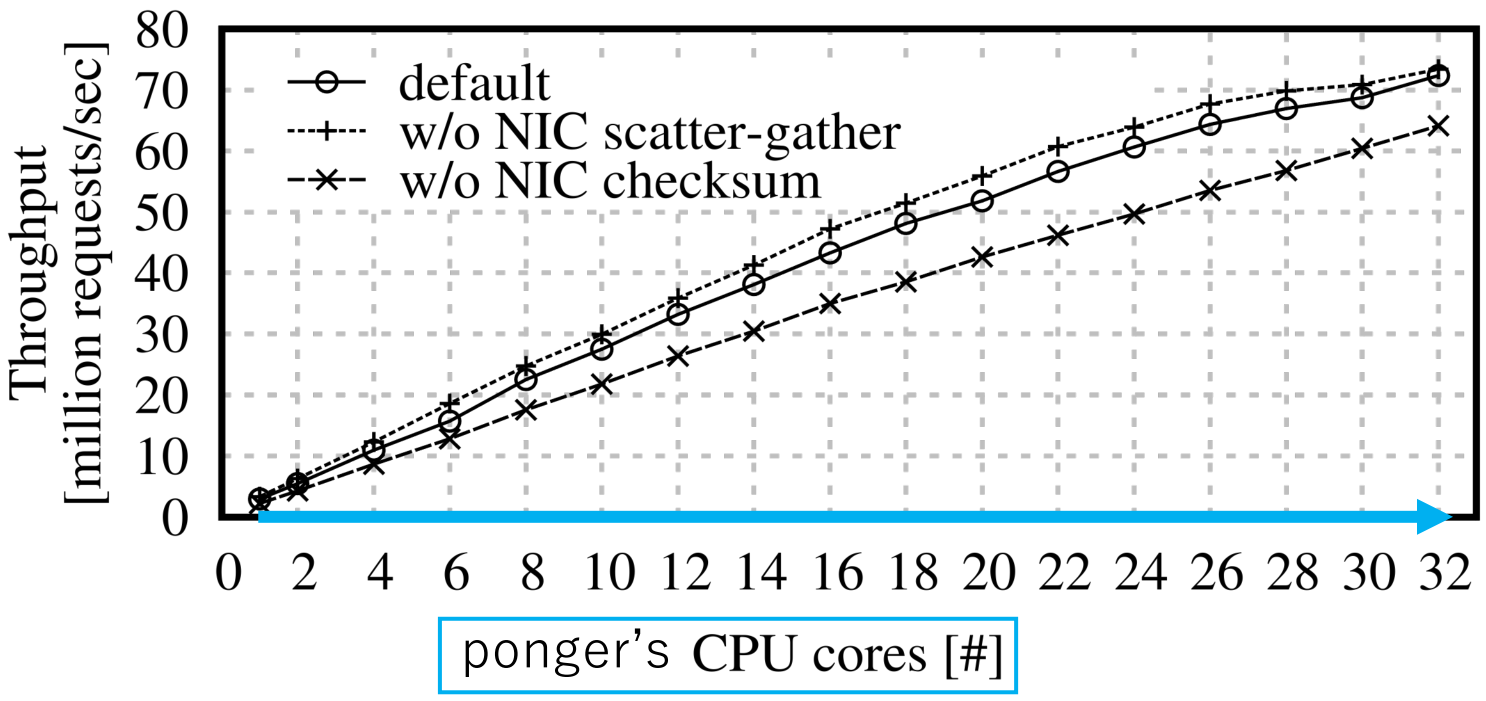
Evaluation: Small Message Exchange

- The pinger and ponger apps exchange 1-byte TCP payloads



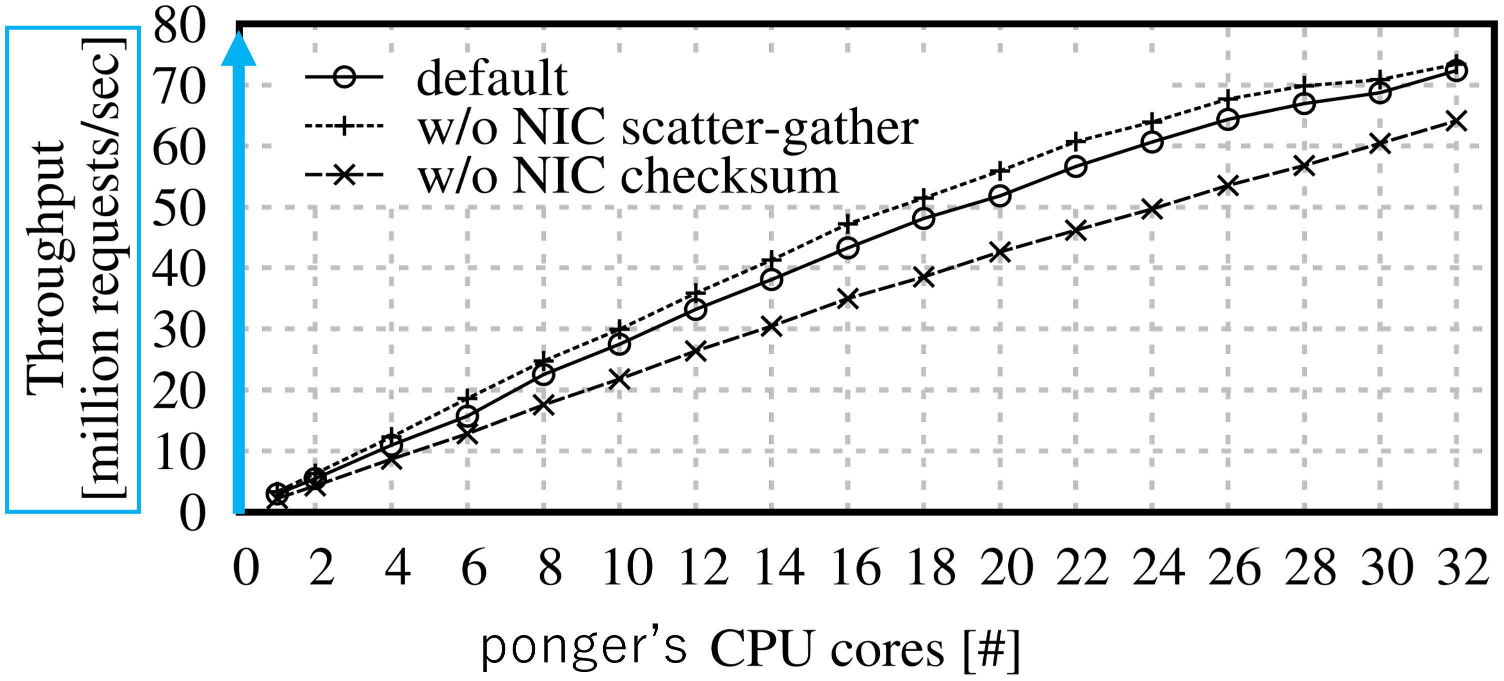
Evaluation: Small Message Exchange

- The pinger and ponger apps exchange 1-byte TCP payloads



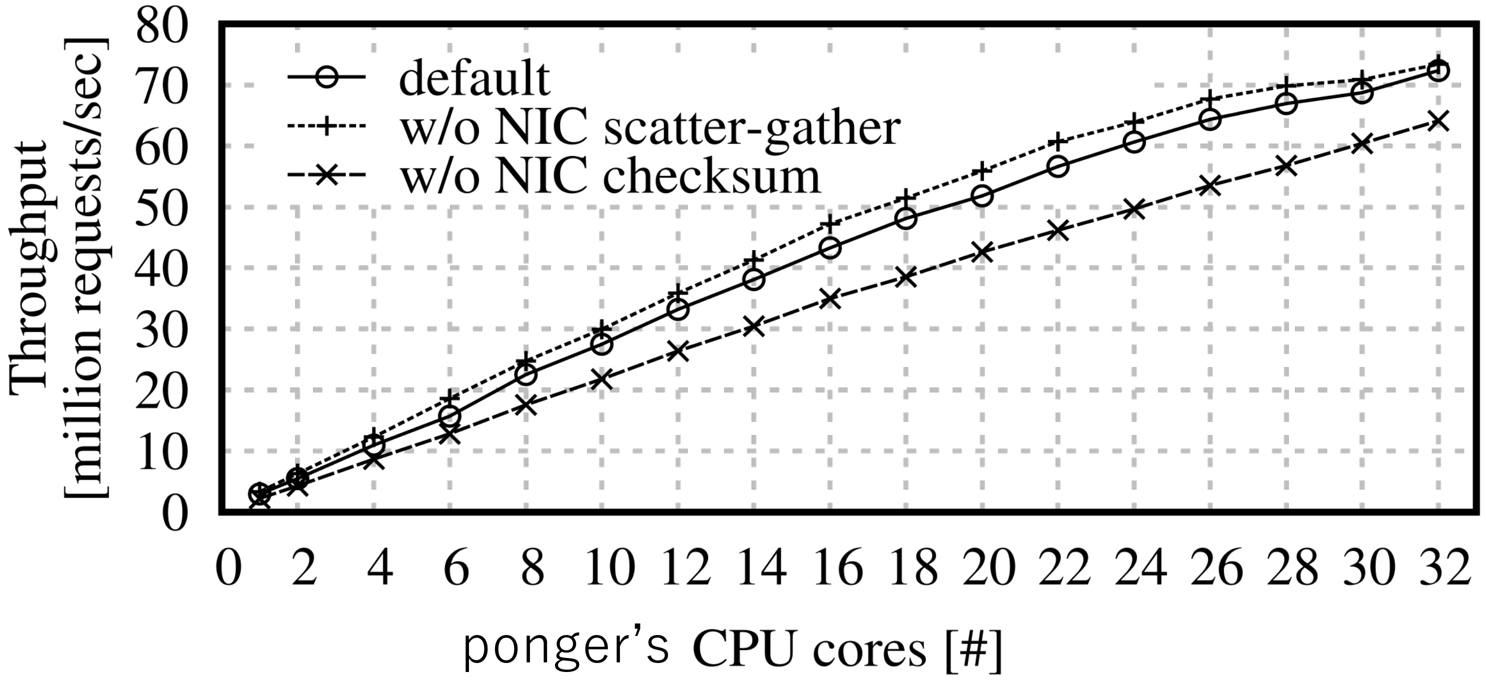
Evaluation: Small Message Exchange

- The pinger and ponger apps exchange 1-byte TCP payloads



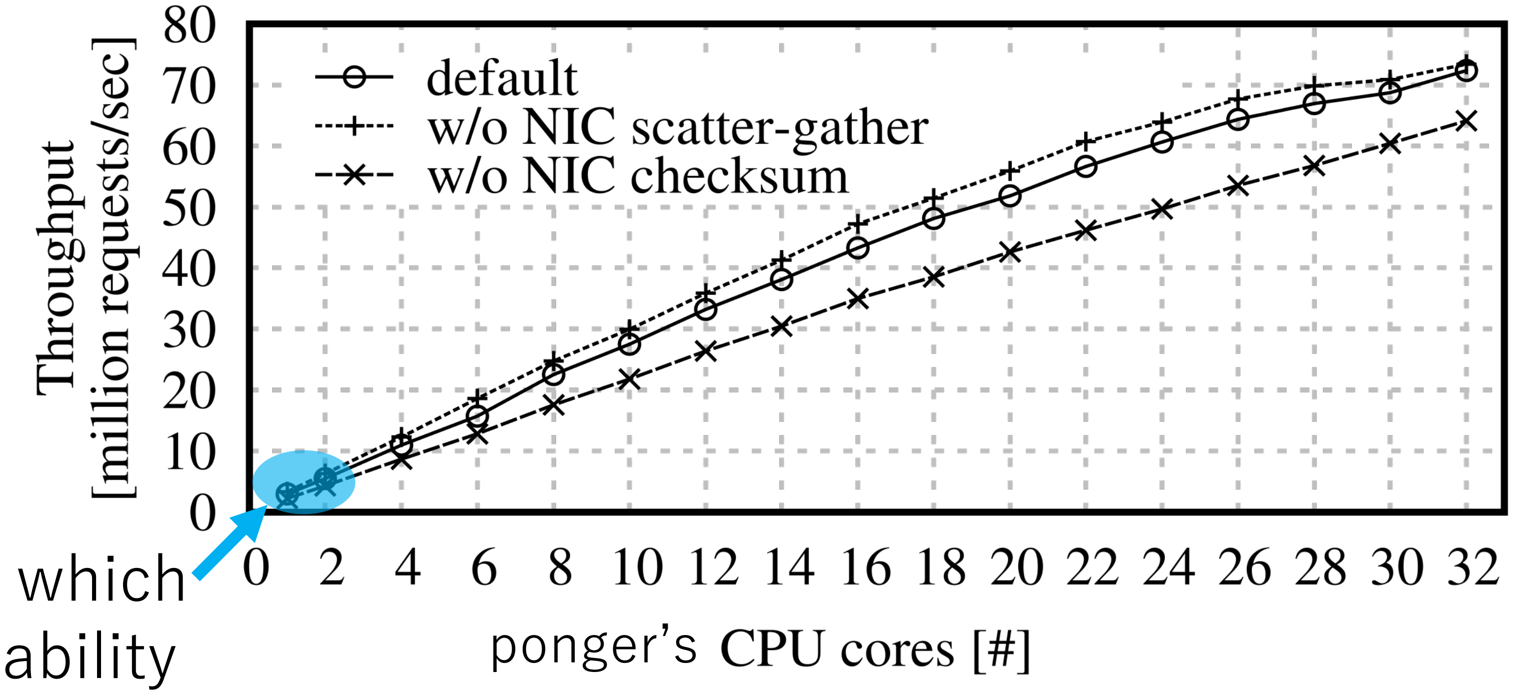
Evaluation: Small Message Exchange

- The pinger and ponger apps exchange 1-byte TCP payloads
- Performance factors
  - Multi-core scalability



Evaluation: Small Message Exchange

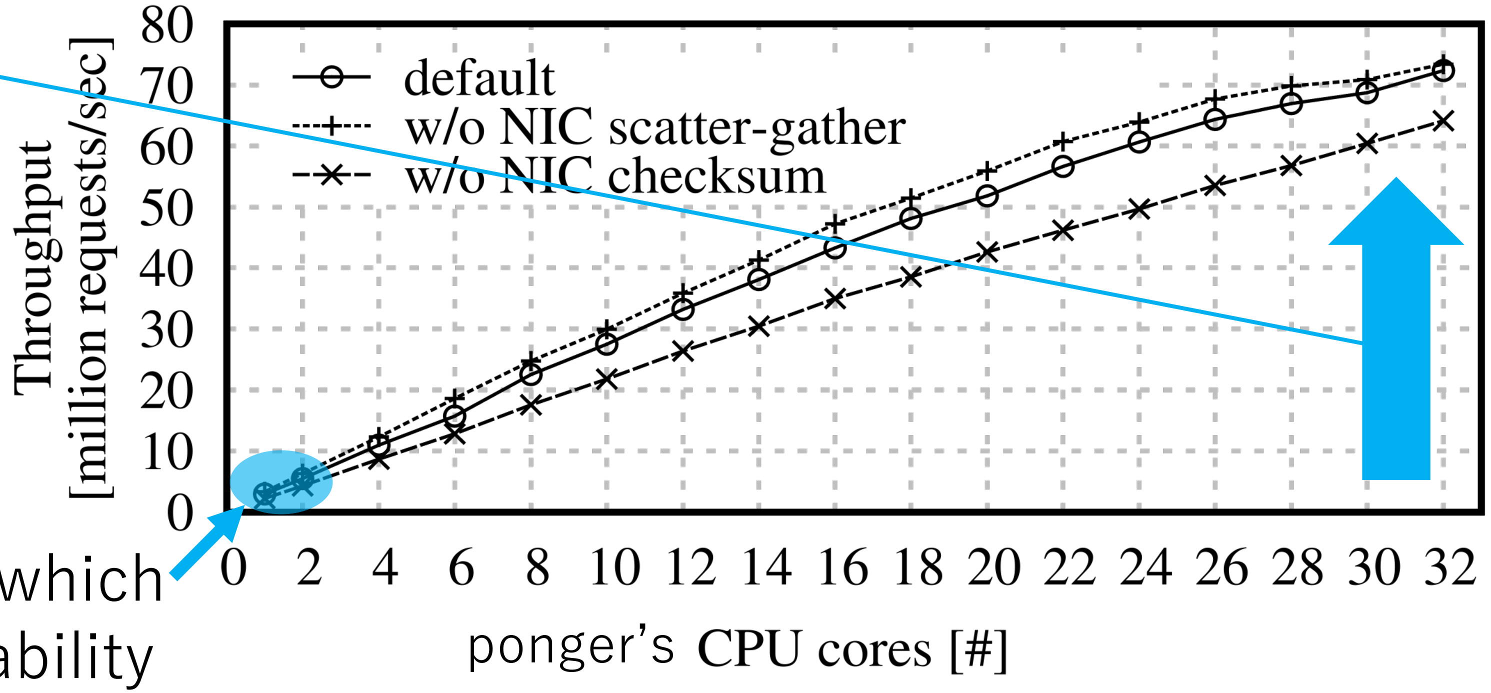
- The pinger and ponger apps exchange 1-byte TCP payloads
- Performance factors
  - Multi-core scalability



Performance of TCP/IP stacks which are unaware of multi-core scalability

Evaluation: Small Message Exchange

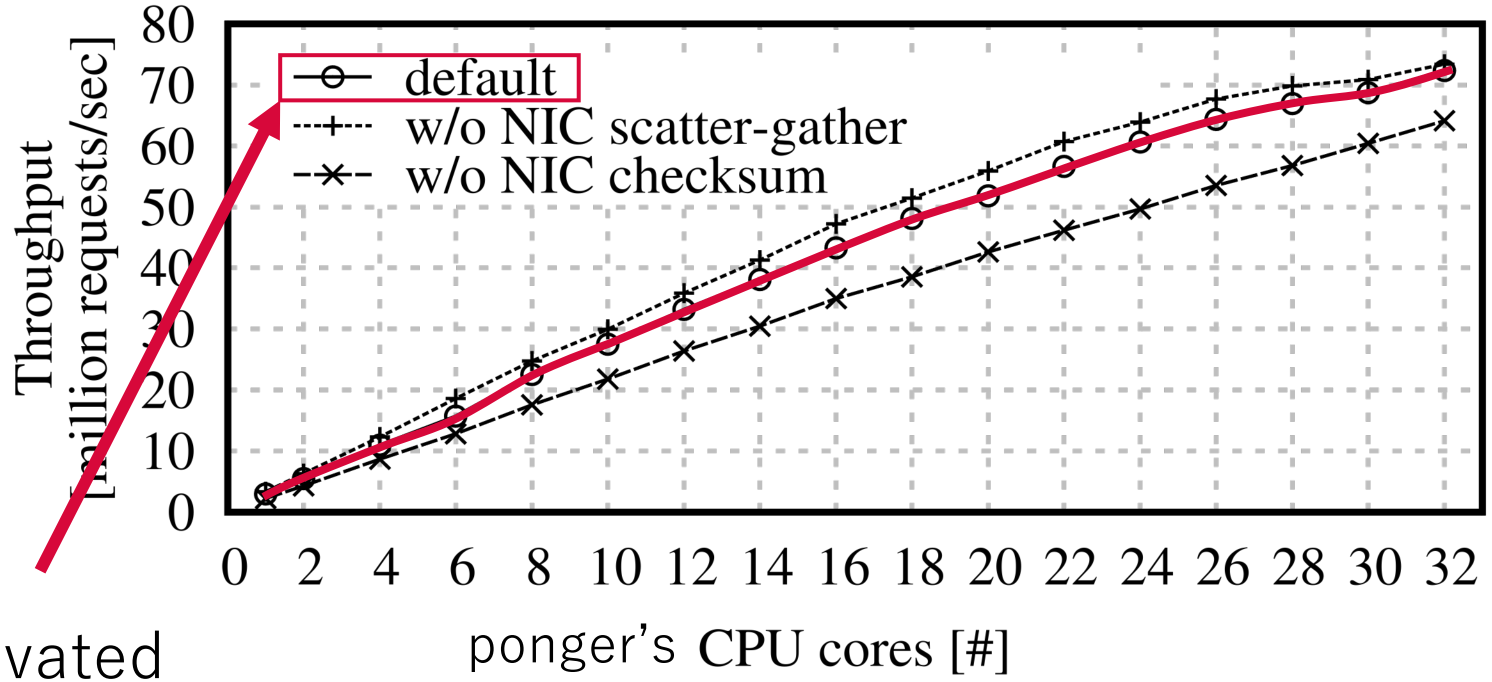
- The pinger and ponger apps exchange 1-byte TCP payloads
- Performance factors
  - Multi-core scalability



Performance of TCP/IP stacks which are unaware of multi-core scalability

Evaluation: Small Message Exchange

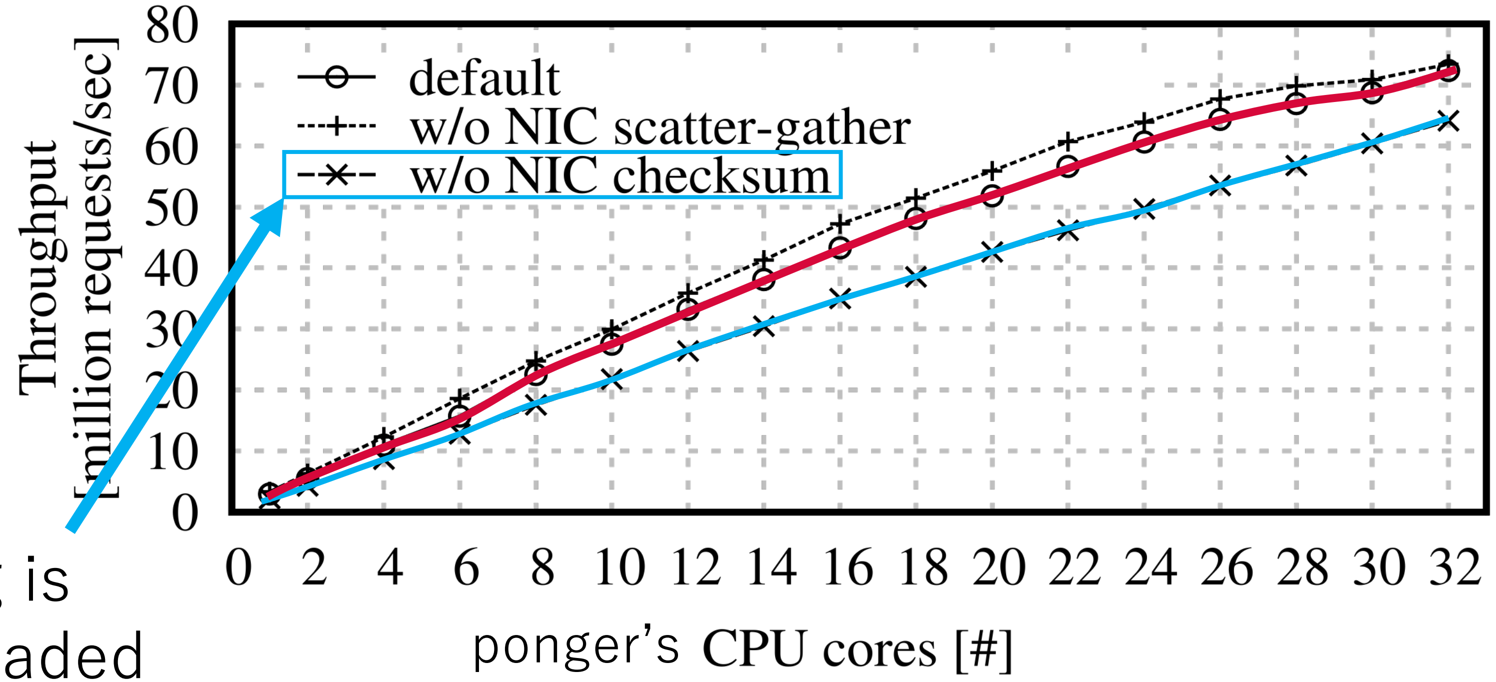
- The pinger and ponger apps exchange 1-byte TCP payloads
- Performance factors
  - Multi-core scalability



All NIC offloading features and zero-copy transmission are activated

Evaluation: Small Message Exchange

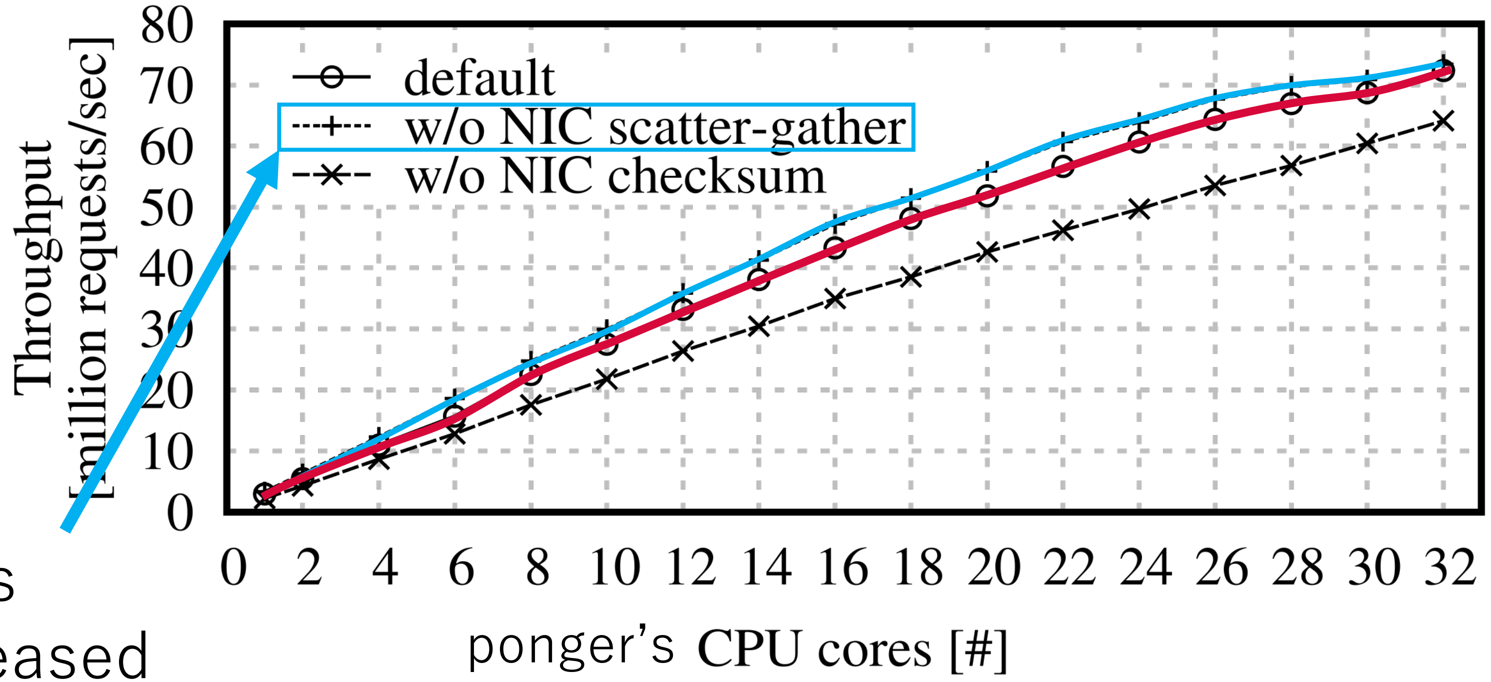
- The pinger and ponger apps exchange 1-byte TCP payloads
- Performance factors
  - Multi-core scalability
  - Checksum offloading



When NIC checksum offloading is deactivated, throughput is degraded

Evaluation: Small Message Exchange

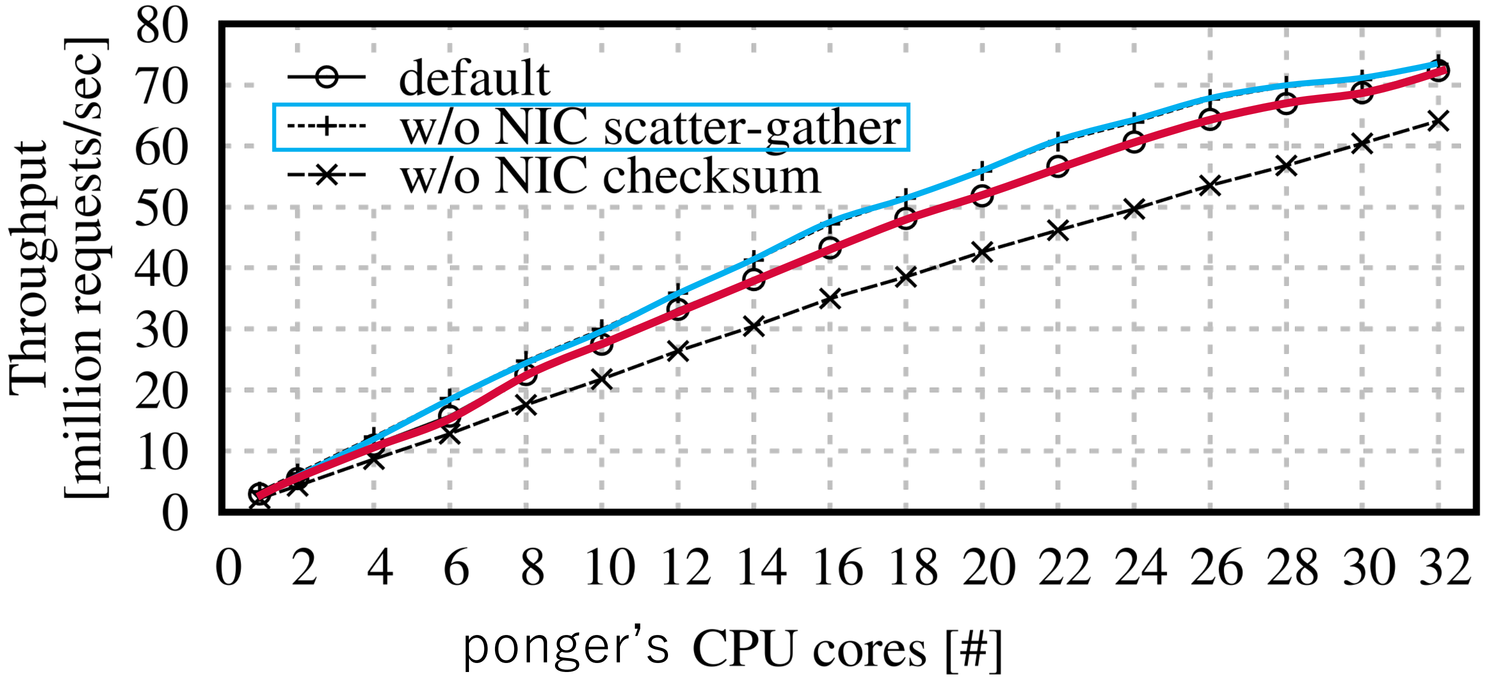
- The pinger and ponger apps exchange 1-byte TCP payloads
- Performance factors
  - Multi-core scalability
  - Checksum offloading



When zero-copy transmission is deactivated, throughput is increased

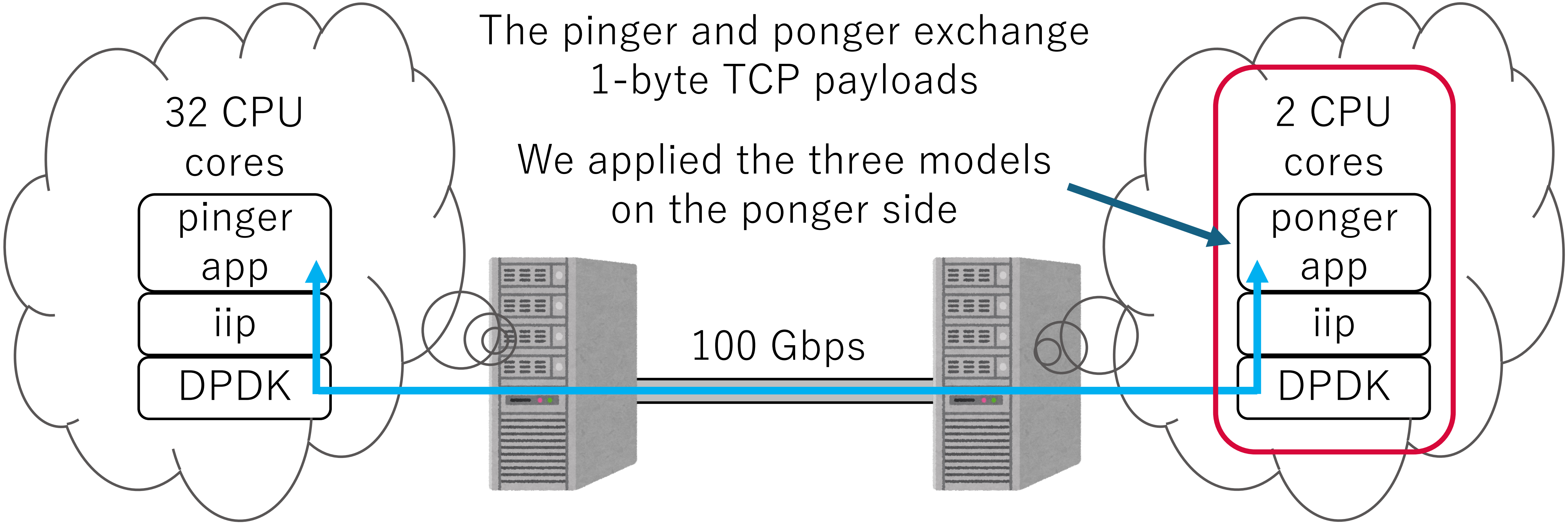
Evaluation: Small Message Exchange

- The pinger and ponger apps exchange 1-byte TCP payloads
- Performance factors
  - Multi-core scalability
  - Checksum offloading
- Tips
  - For small messages, copy is faster than zero-copy transmission



Evaluation: CPU Core Assignment Models

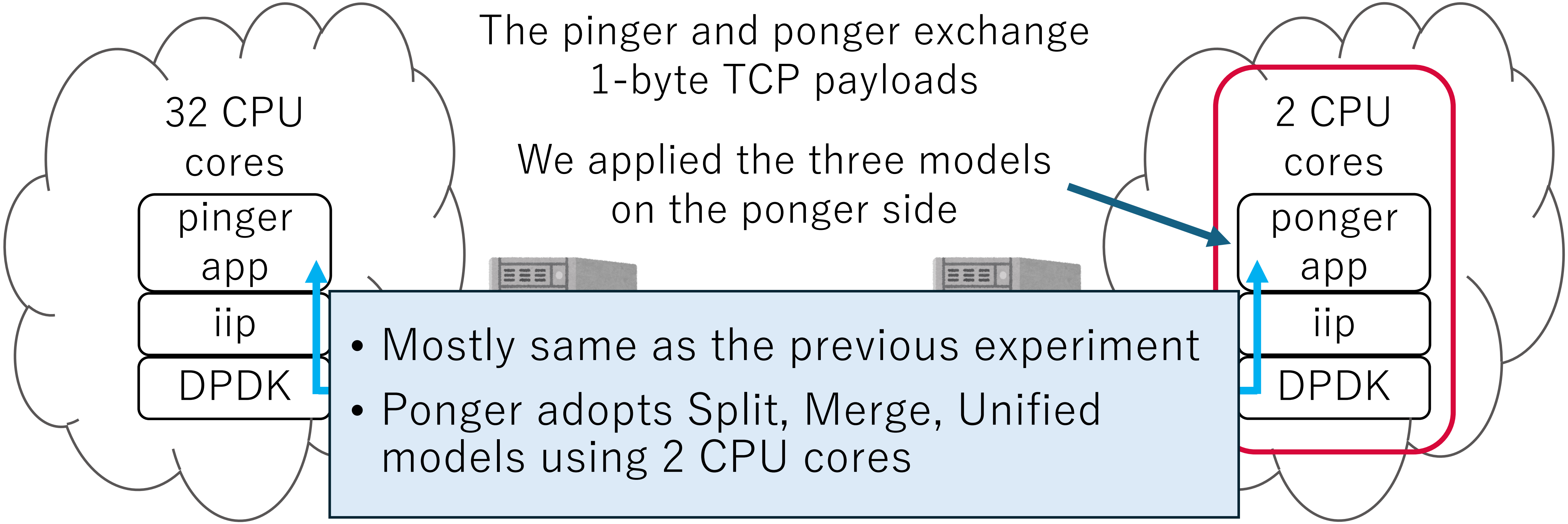
- TCP ping-pong workload



https://github.com/yasukata/iip

Evaluation: CPU Core Assignment Models

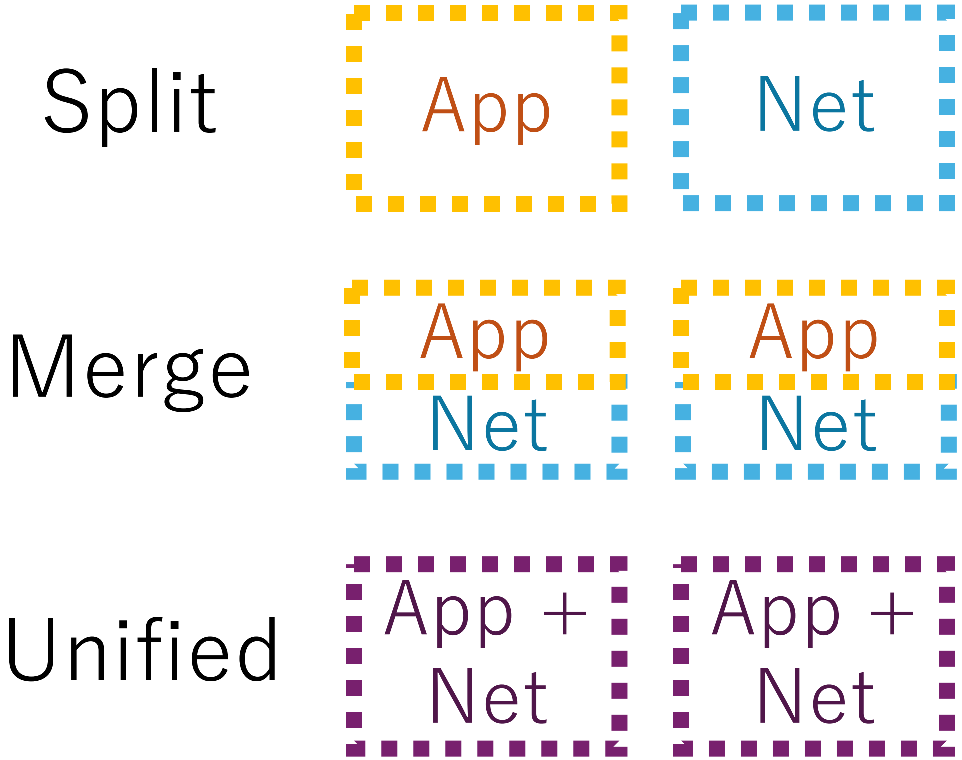
- TCP ping-pong workload

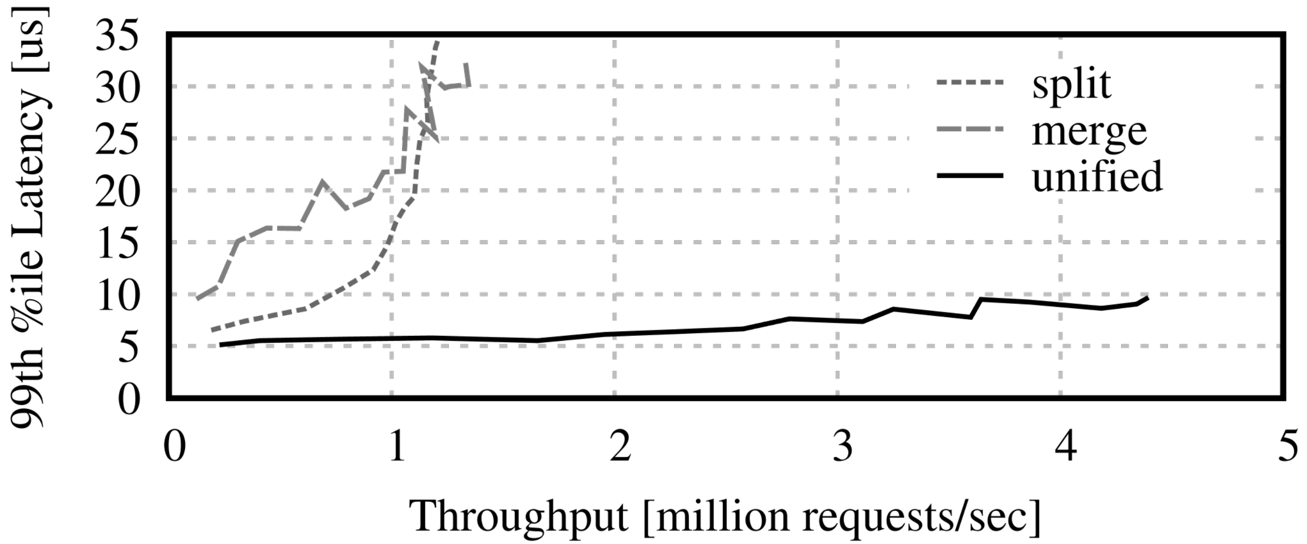


- Mostly same as the previous experiment
- Ponger adopts Split, Merge, Unified models using 2 CPU cores

Evaluation: CPU Core Assignment Models

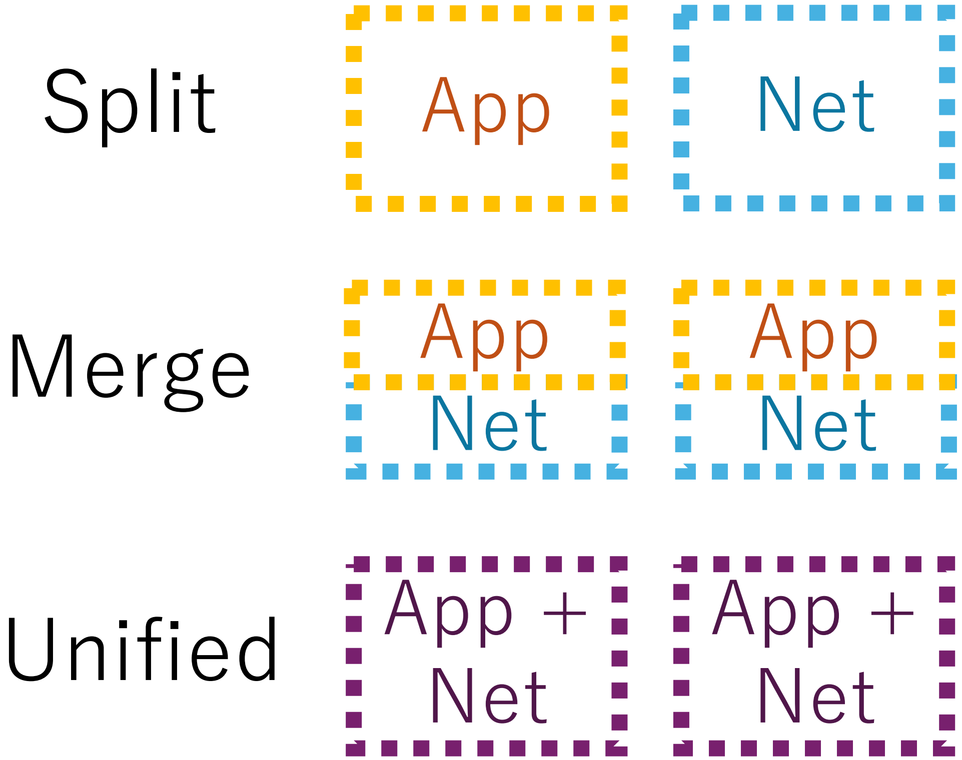
- The pinger and ponger apps exchange 1-byte TCP payloads

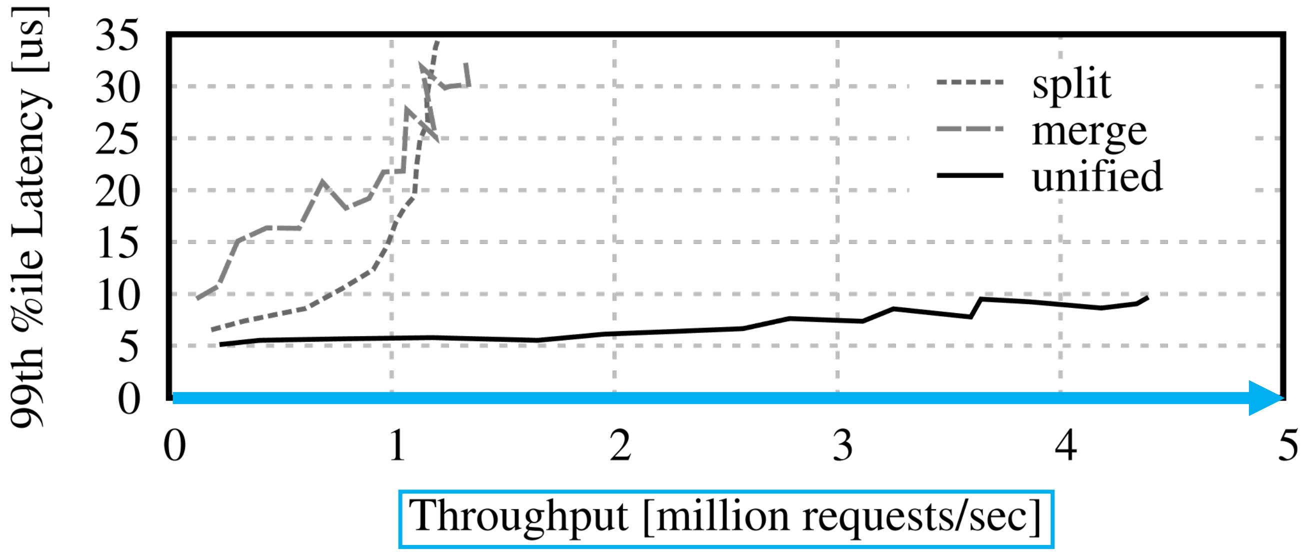




Evaluation: CPU Core Assignment Models

- The pinger and ponger apps exchange 1-byte TCP payloads

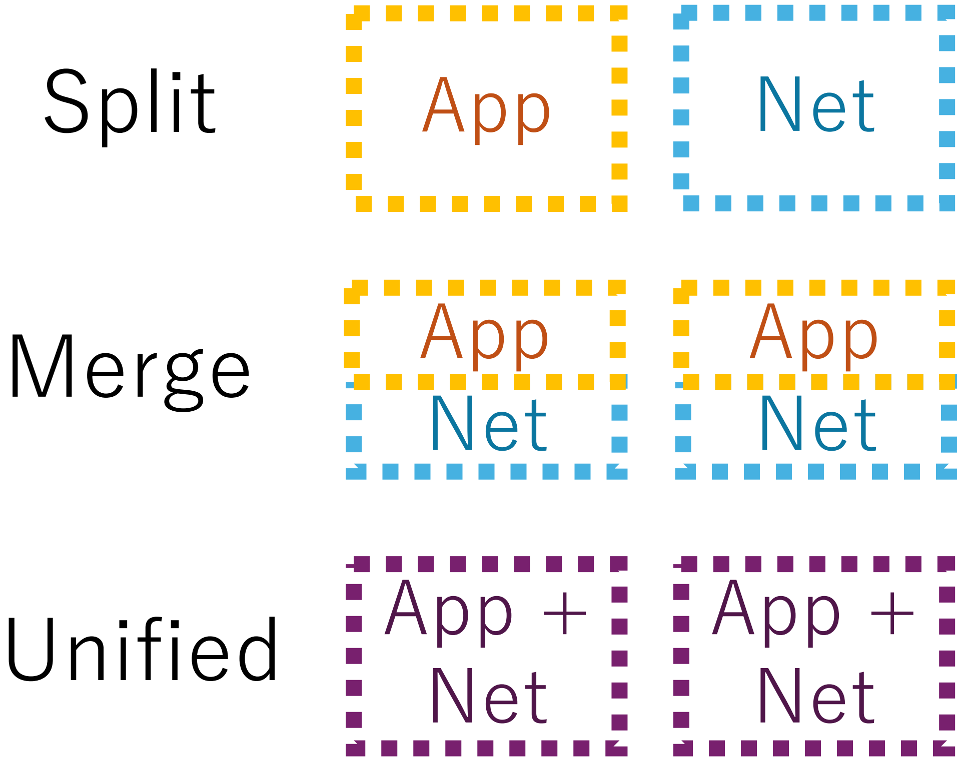


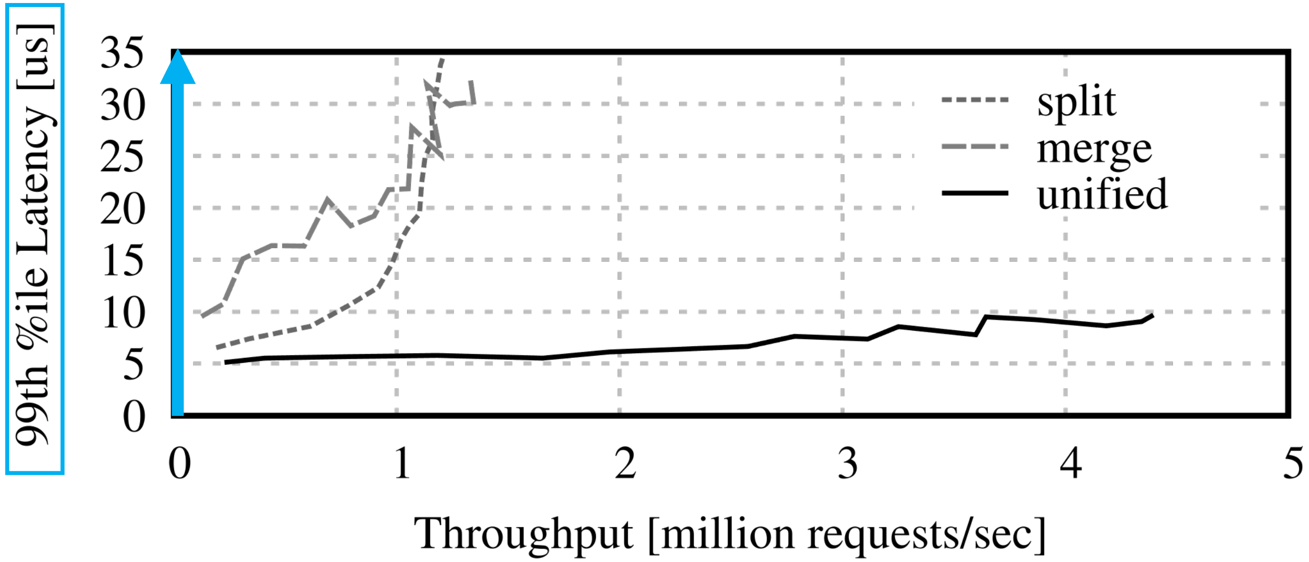


https://github.com/yasukata/iip

Evaluation: CPU Core Assignment Models

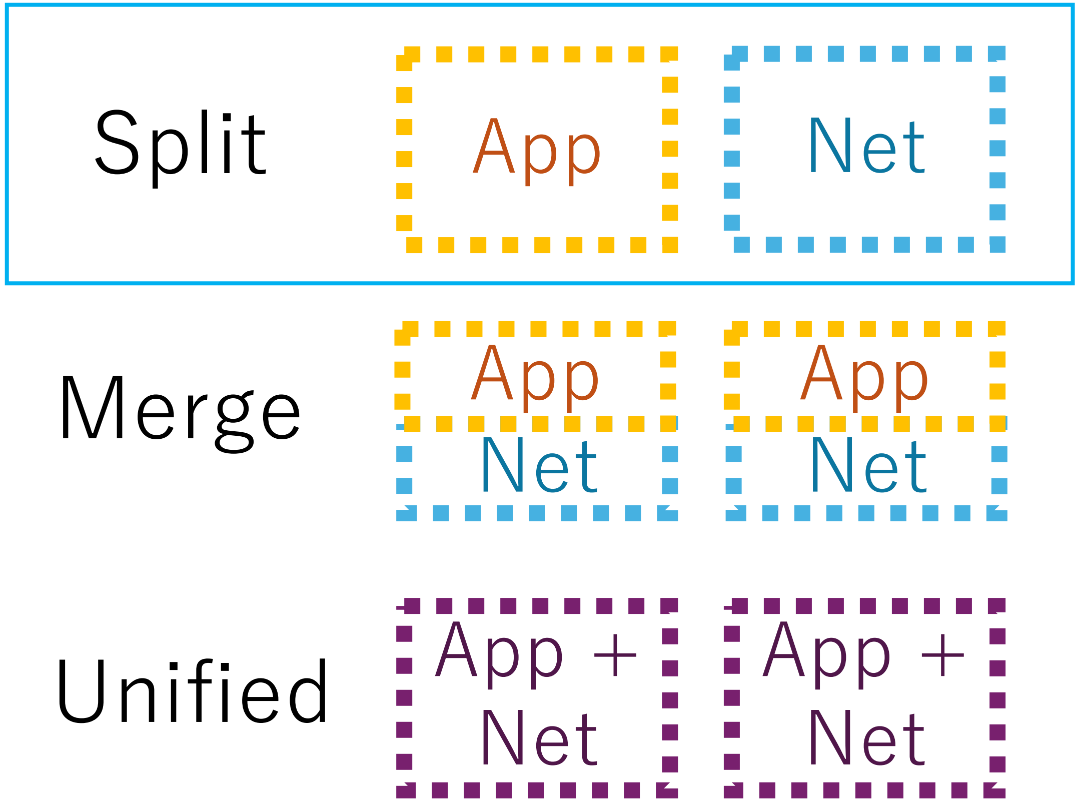
- The pinger and ponger apps exchange 1-byte TCP payloads

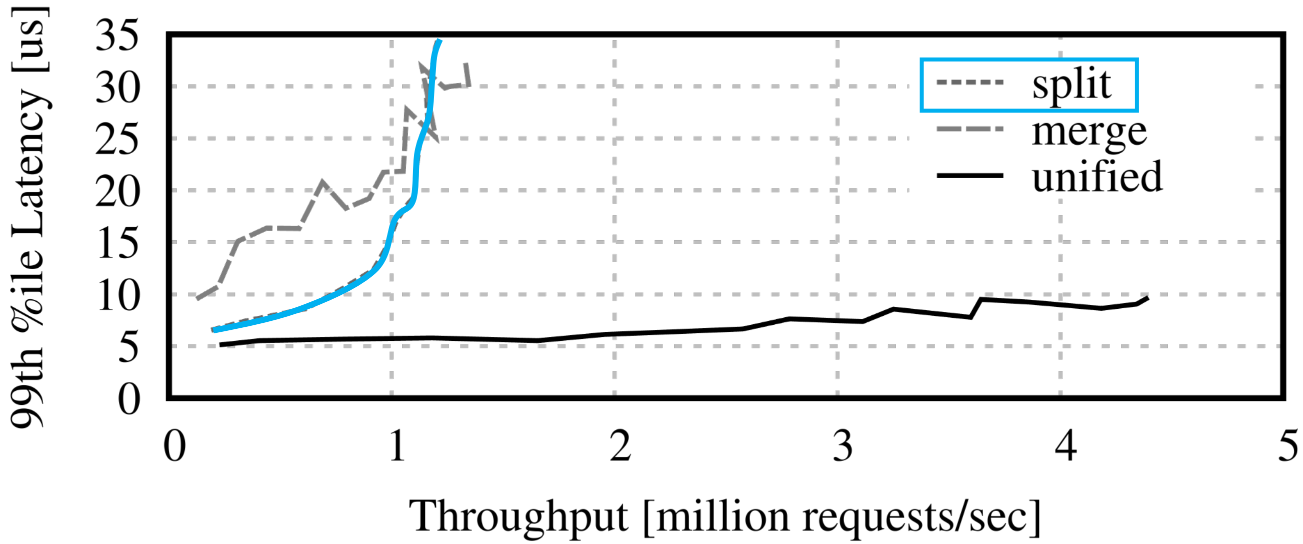




Evaluation: CPU Core Assignment Models

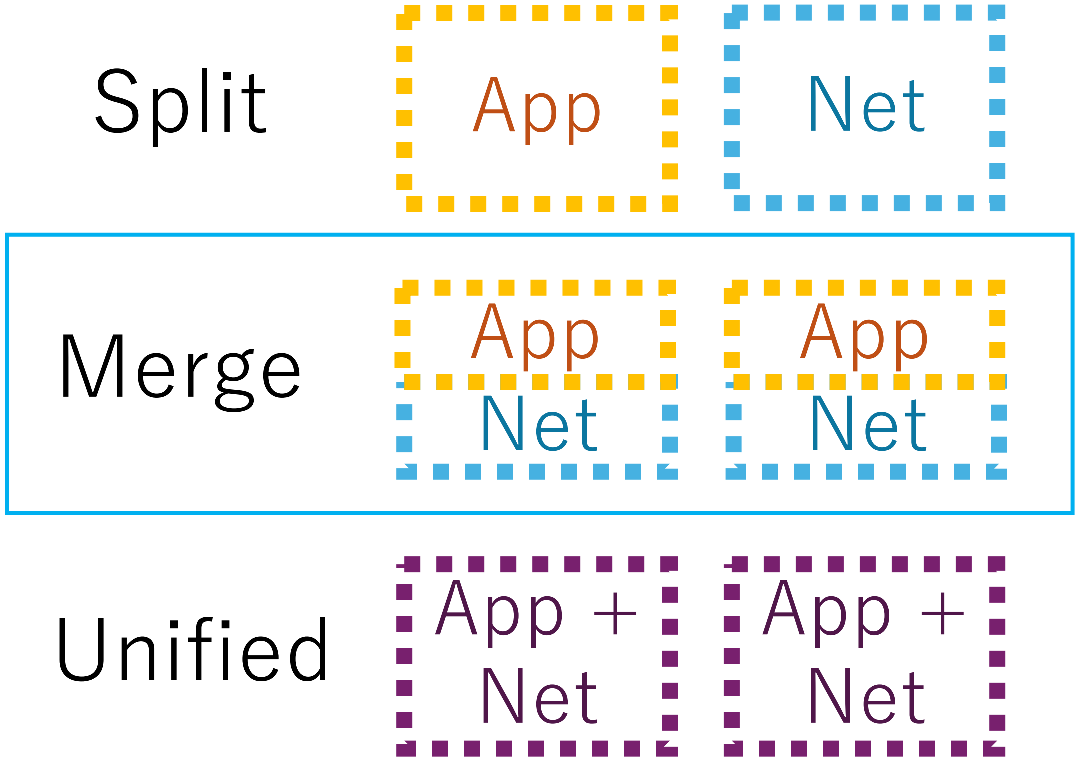
- The pinger and ponger apps exchange 1-byte TCP payloads

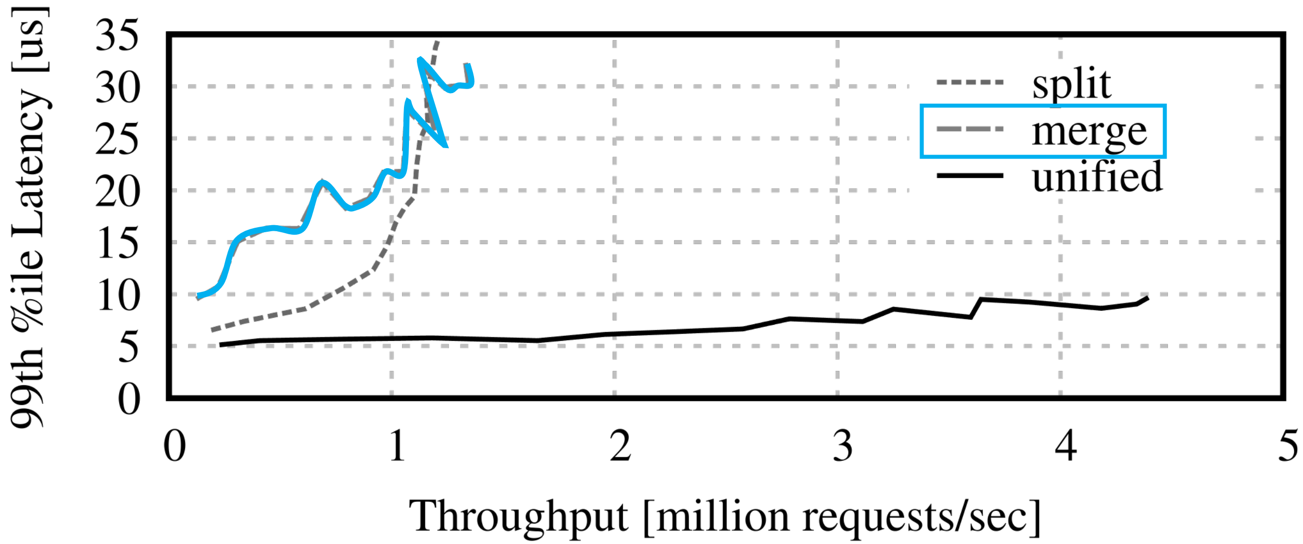




Evaluation: CPU Core Assignment Models

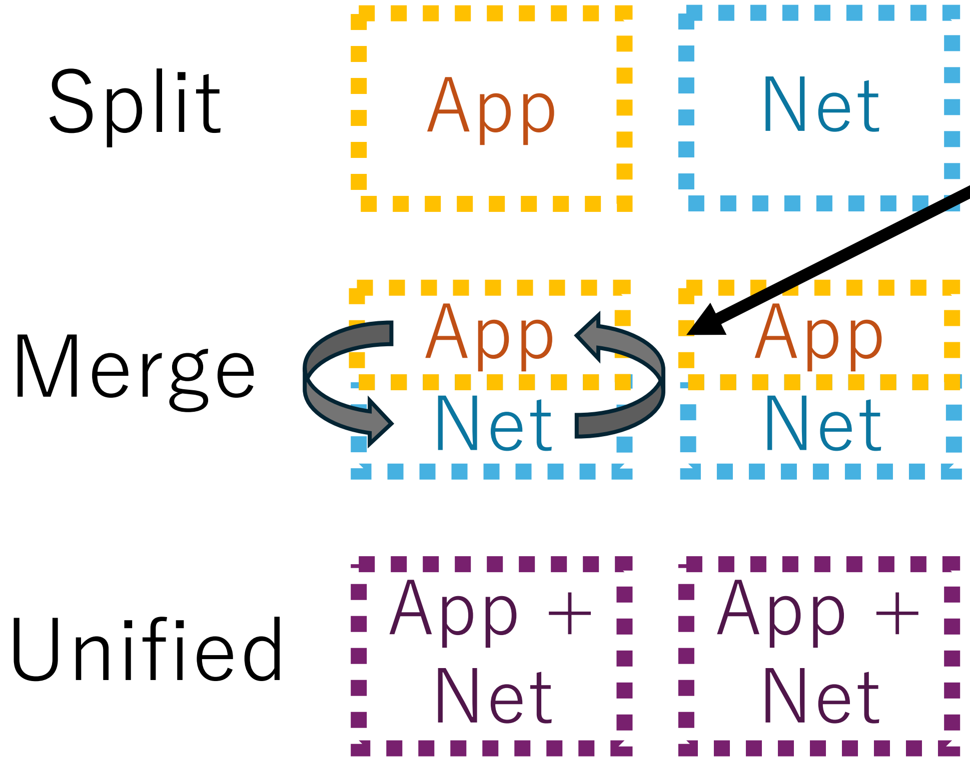
- The pinger and ponger apps exchange 1-byte TCP payloads



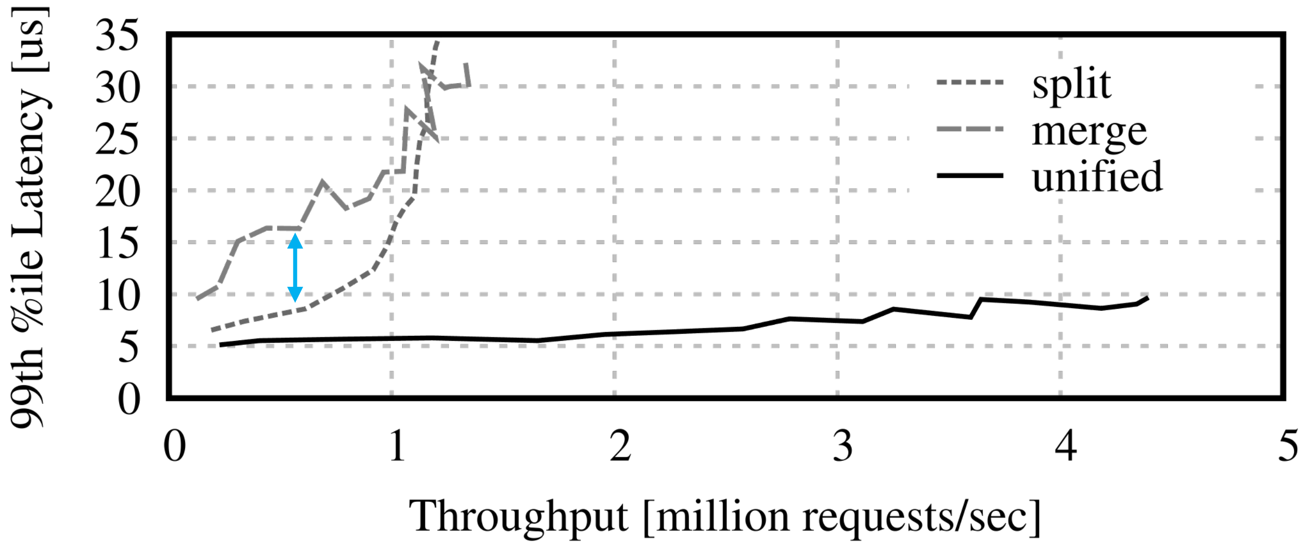


Evaluation: CPU Core Assignment Models

- The pinger and ponger apps exchange 1-byte TCP payloads

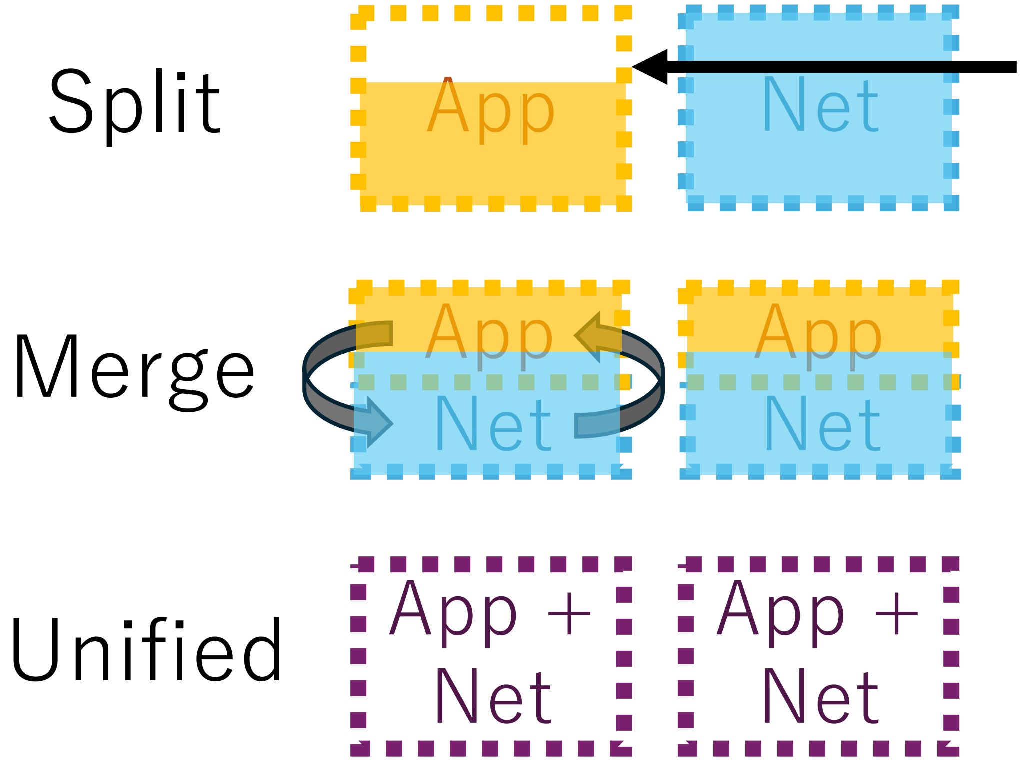


The latency gap comes from the app/net transition cost of the merge model

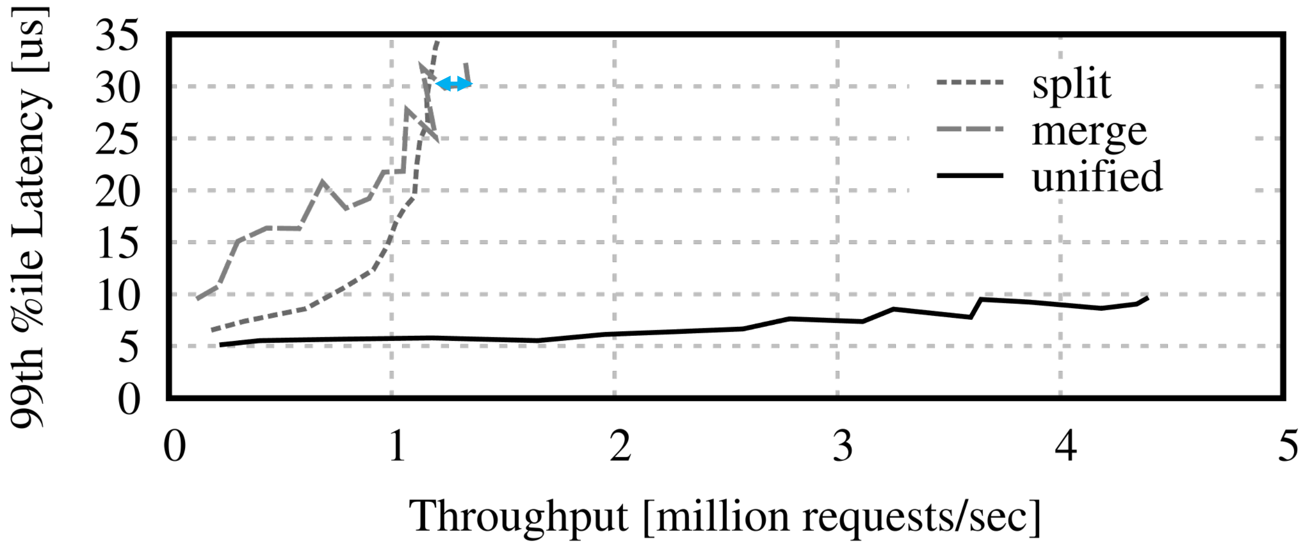


Evaluation: CPU Core Assignment Models

- The pinger and ponger apps exchange 1-byte TCP payloads

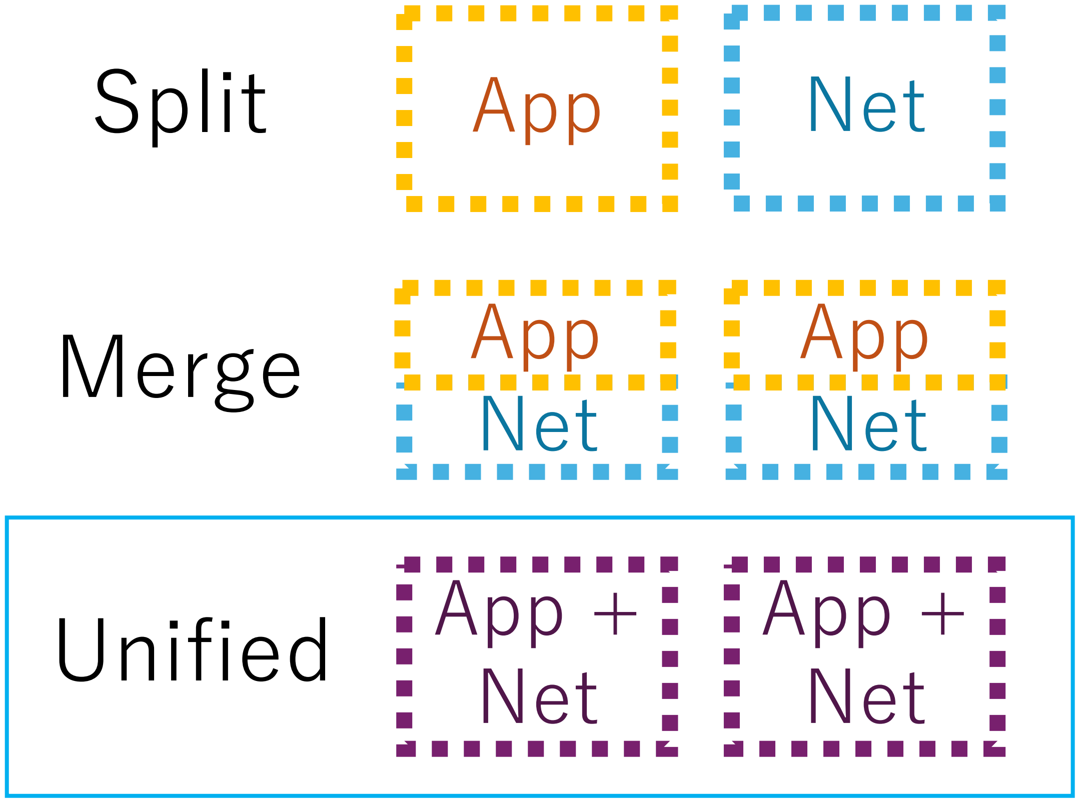


The merge model exhibits higher throughput than the split model because of CPU utilization

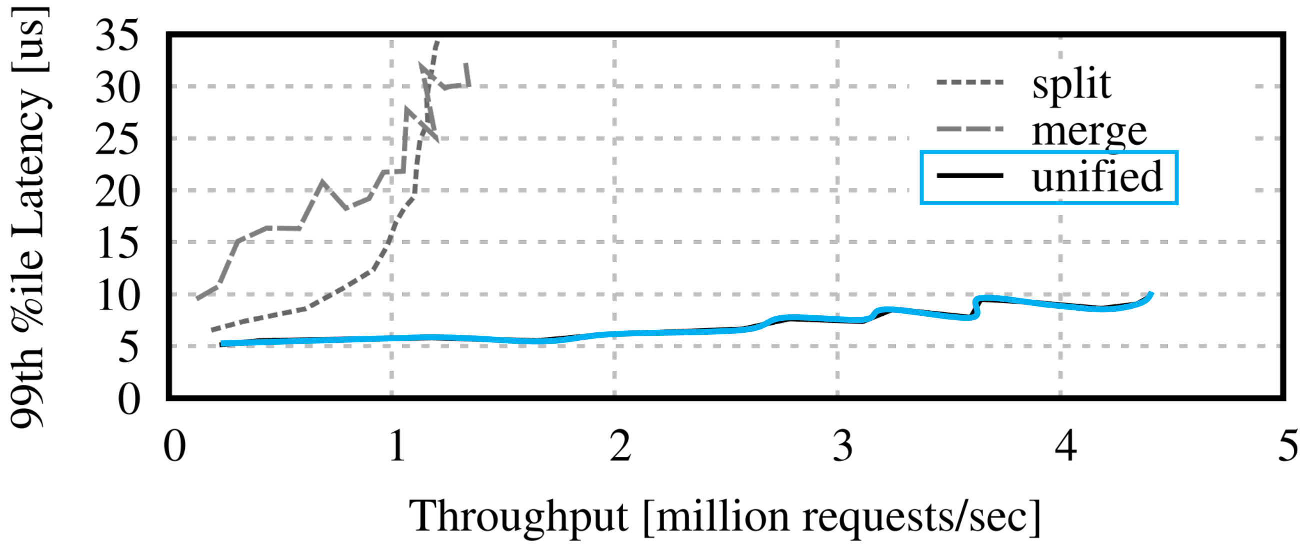


Evaluation: CPU Core Assignment Models

- The pinger and ponger apps exchange 1-byte TCP payloads



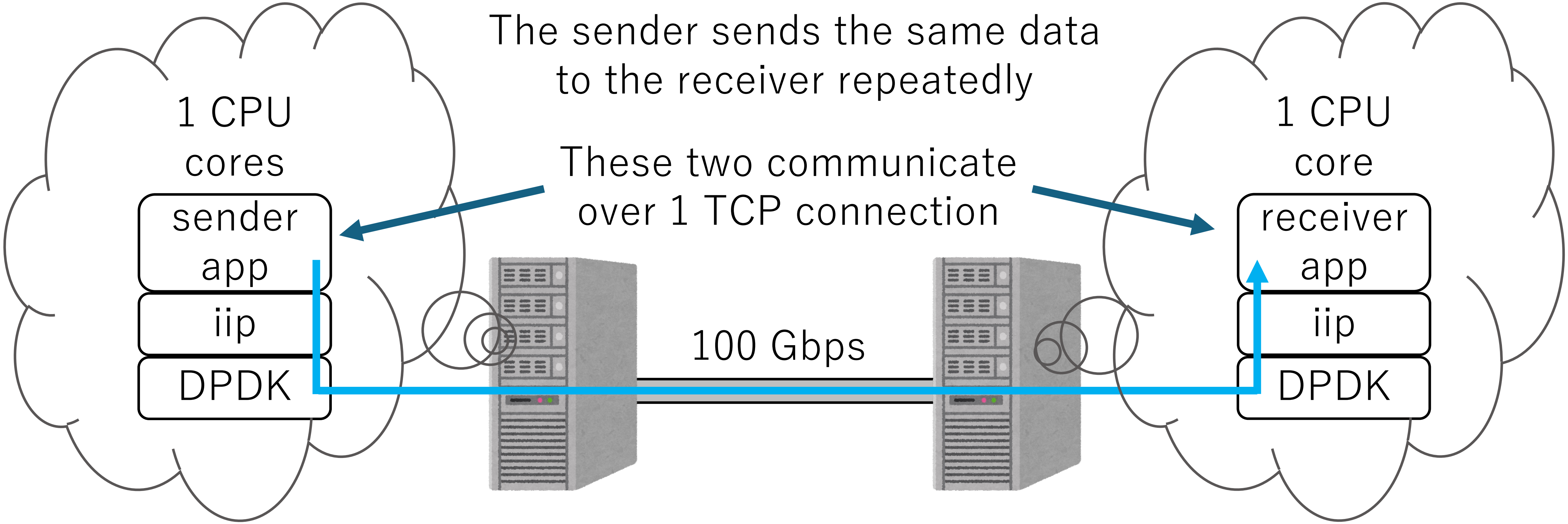
The unified model achieves the best speed because it is free from the issues of the others



Evaluation: Bulk Data Transfer

- Data transfer workload

Each uses 1 CPU core



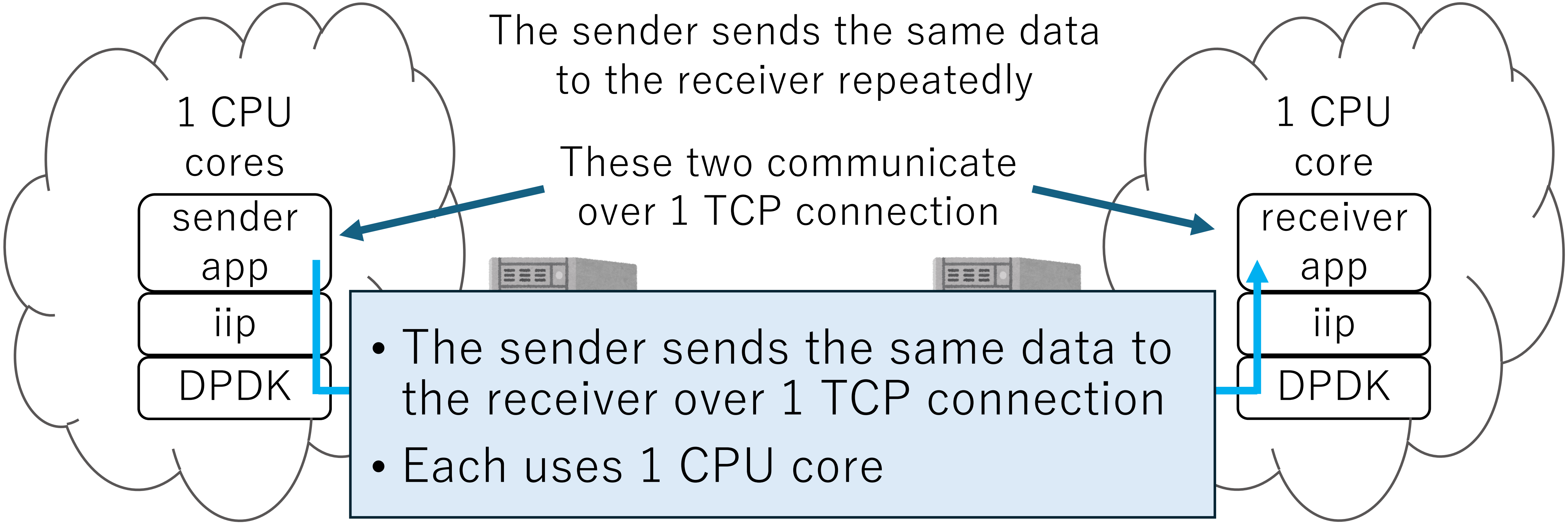
https://github.com/yasukata/iip

Evaluation: Bulk Data Transfer

Each uses 1 CPU core

- Data transfer workload

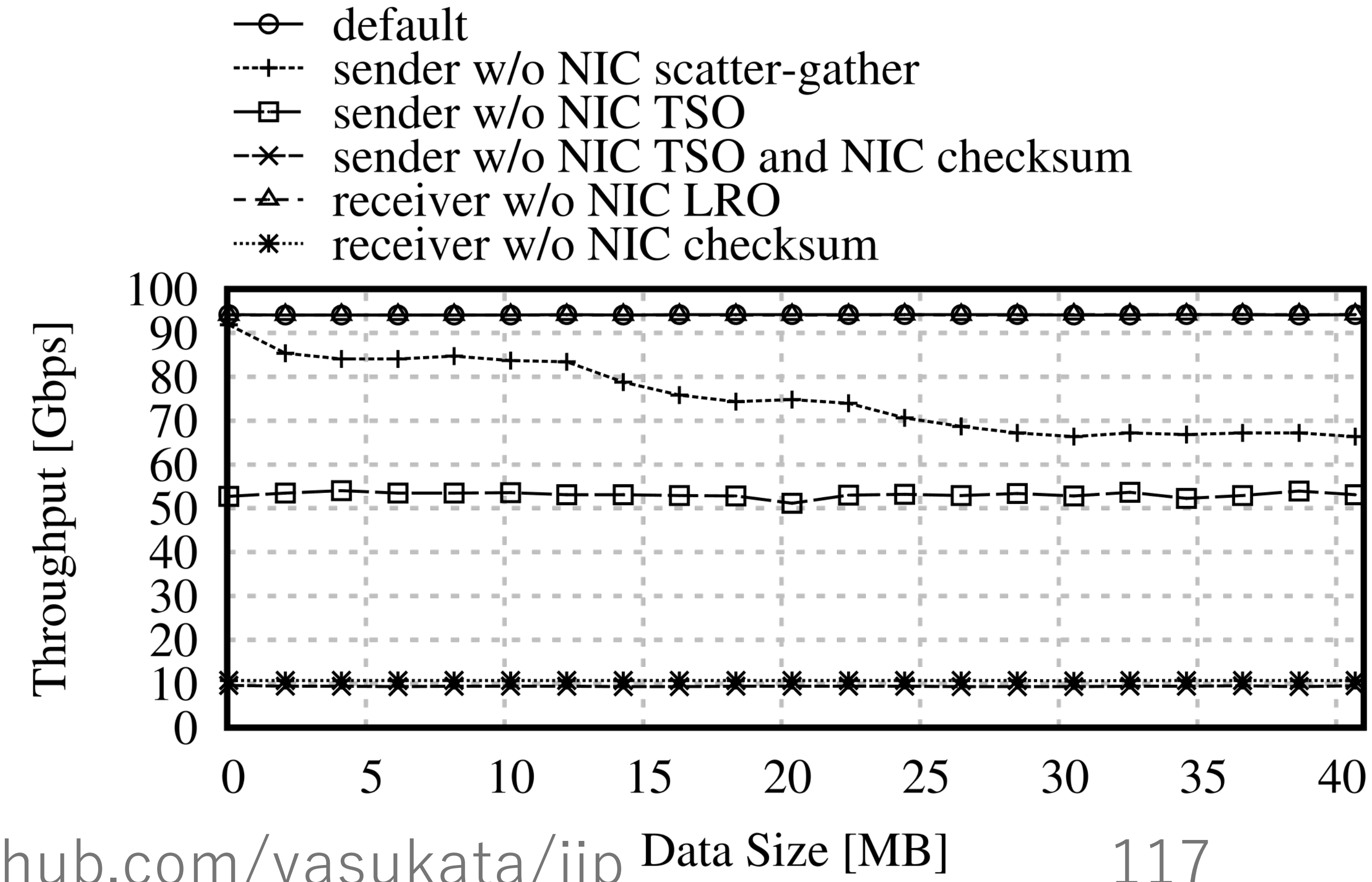
The sender sends the same data
to the receiver repeatedly



- The sender sends the same data to the receiver over 1 TCP connection
- Each uses 1 CPU core

Evaluation: Bulk Data Transfer

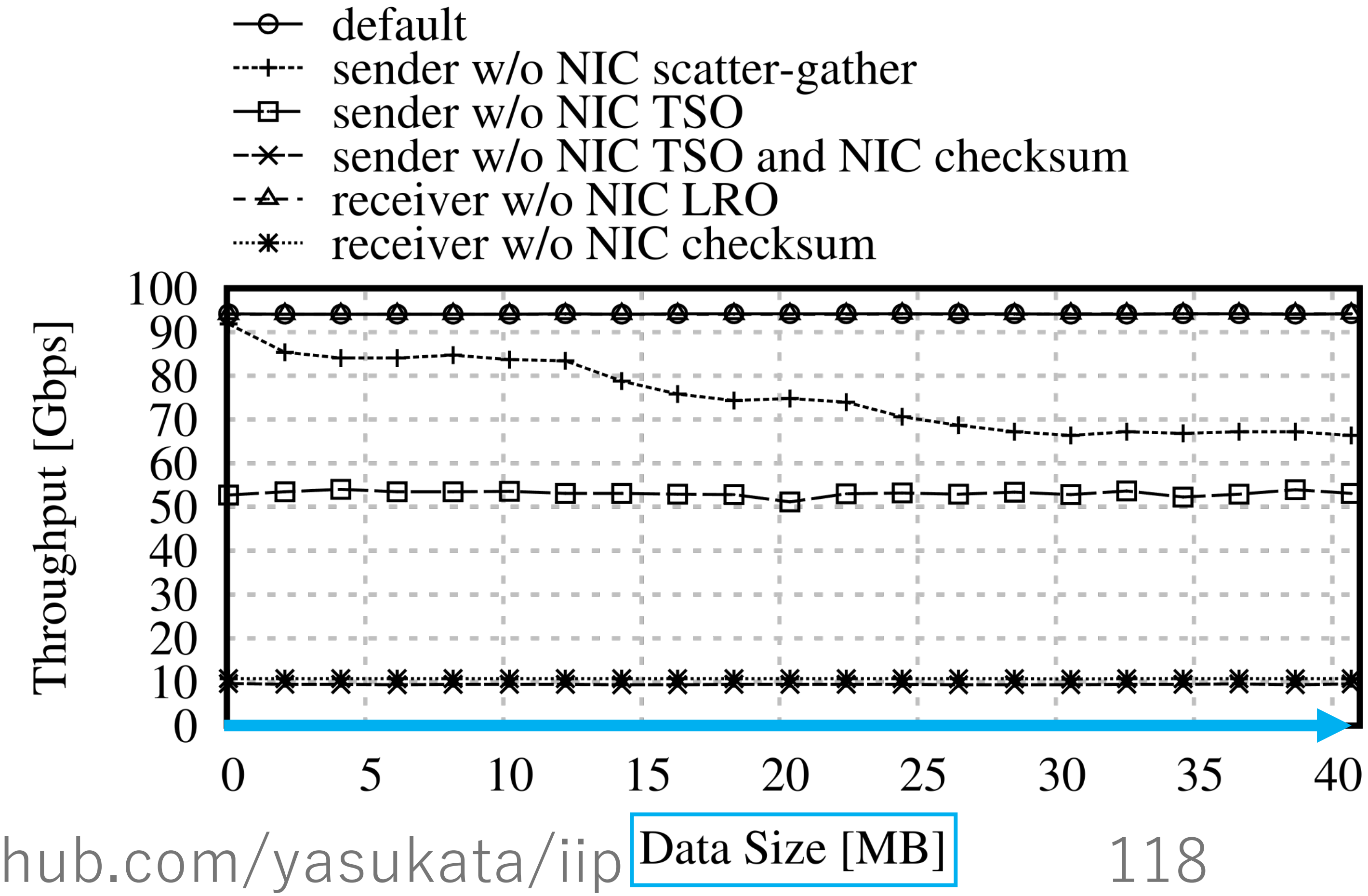
- The sender repeatedly sends the same data to the receiver



https://github.com/yasukata/iip

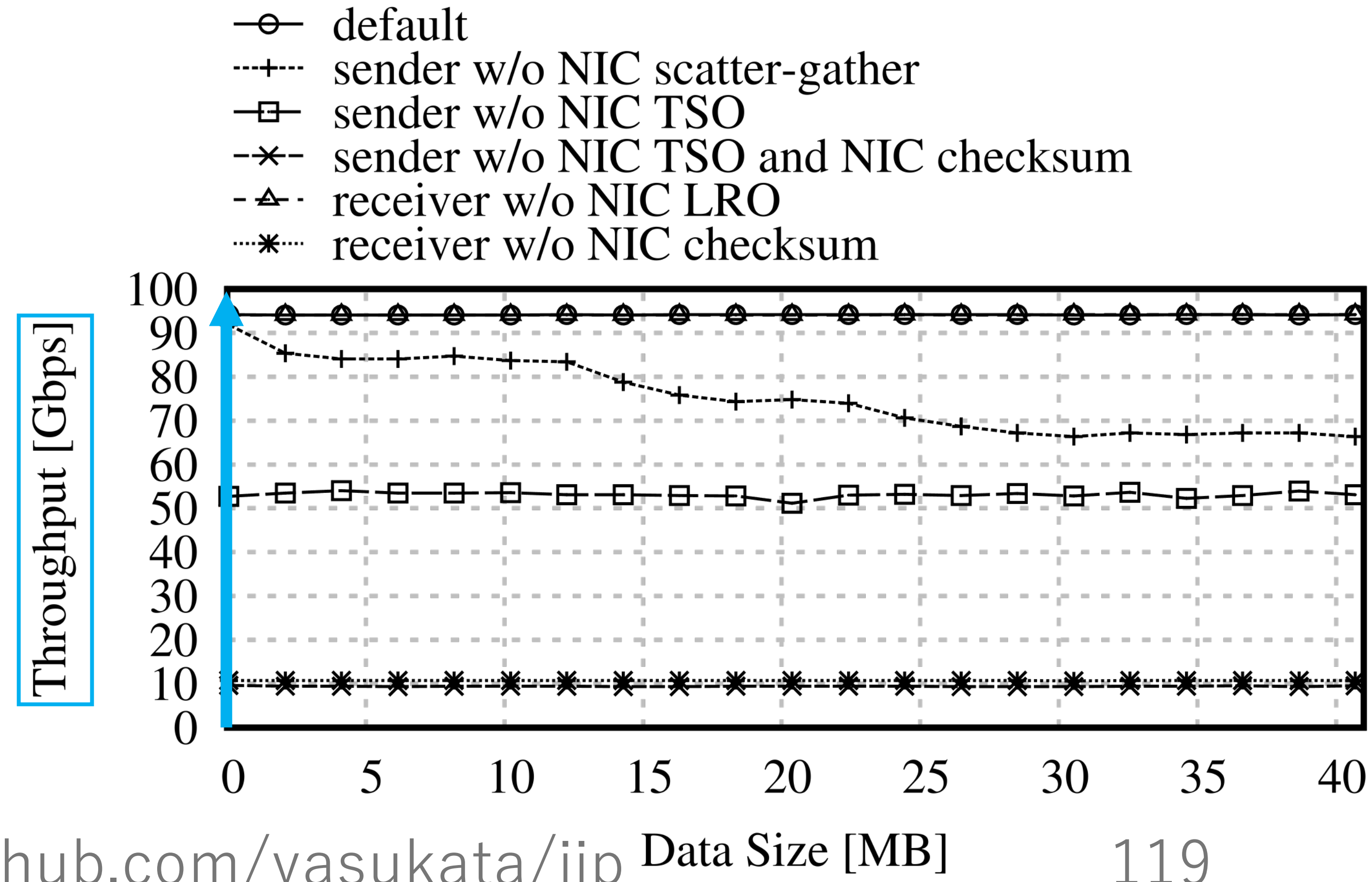
Evaluation: Bulk Data Transfer

- The sender repeatedly sends the same data to the receiver



Evaluation: Bulk Data Transfer

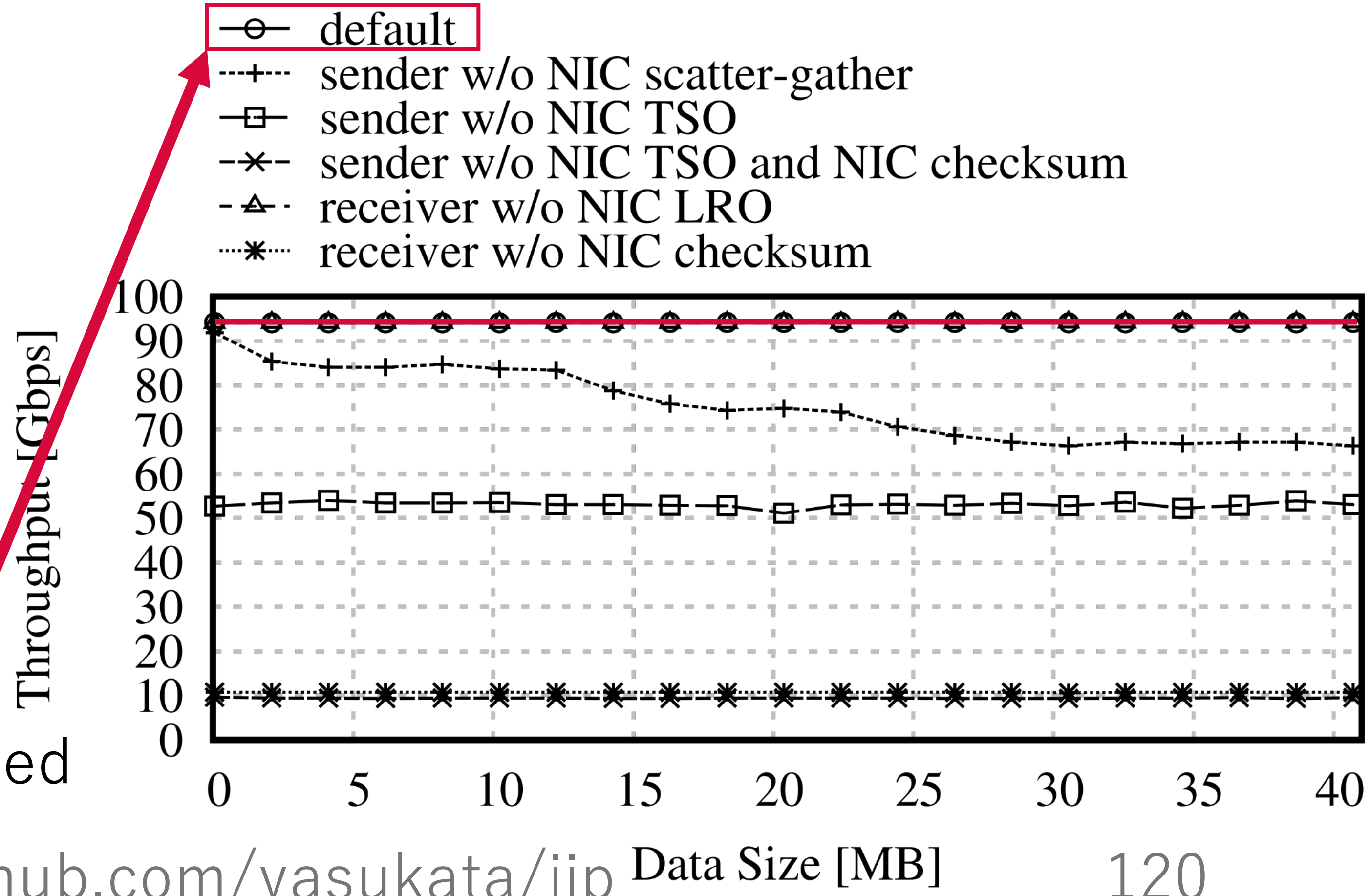
- The sender repeatedly sends the same data to the receiver



https://github.com/yasukata/iip

Evaluation: Bulk Data Transfer

- The sender repeatedly sends the same data to the receiver

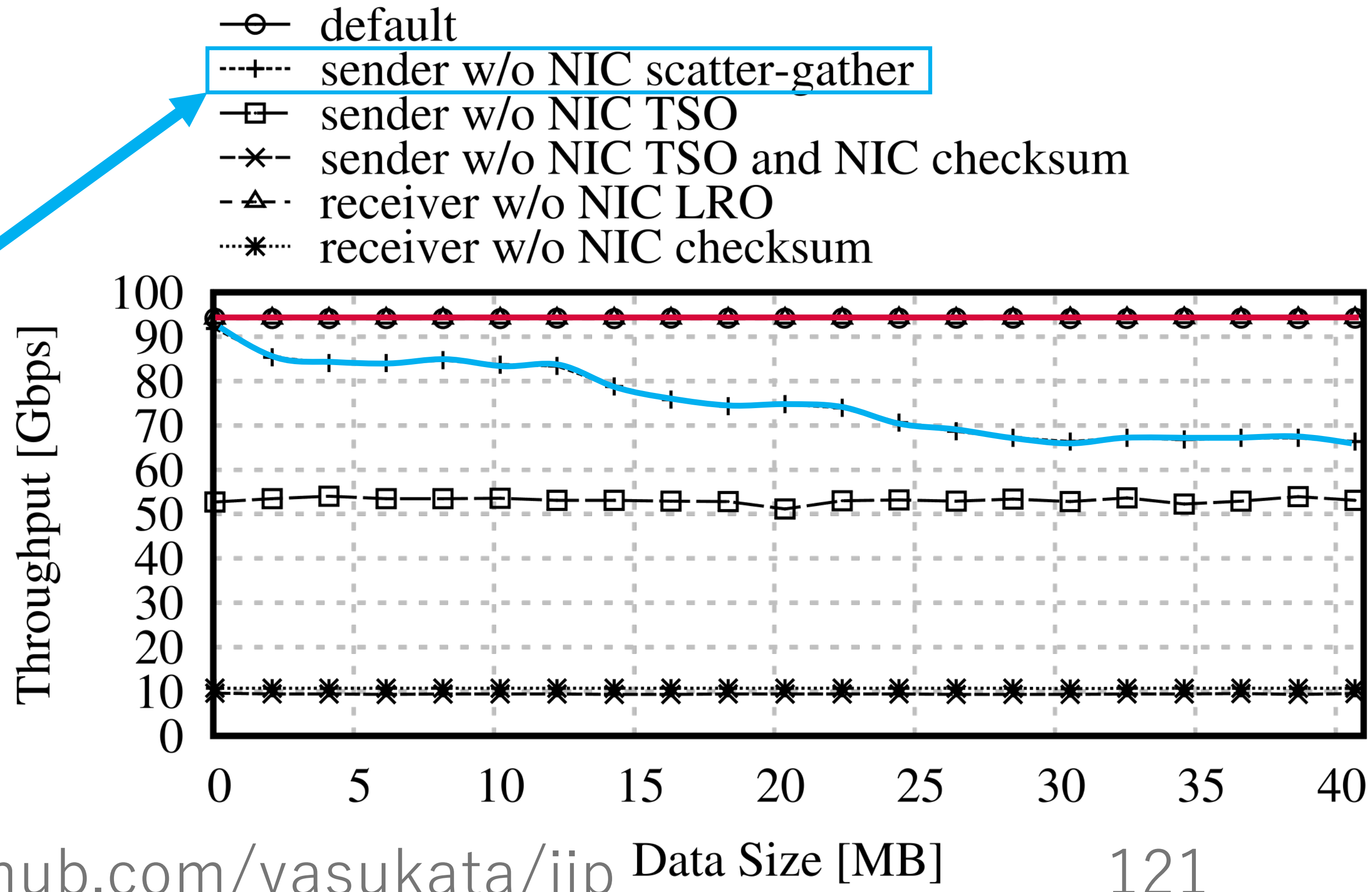


All NIC offloading features and zero-copy transmission are activated

Evaluation: Bulk Data Transfer

- The sender repeatedly sends the same data to the receiver
- Performance factors
    - Zero-copy transmission

When the sender deactivates zero-copy transmission, throughput is degraded according to the size of the data

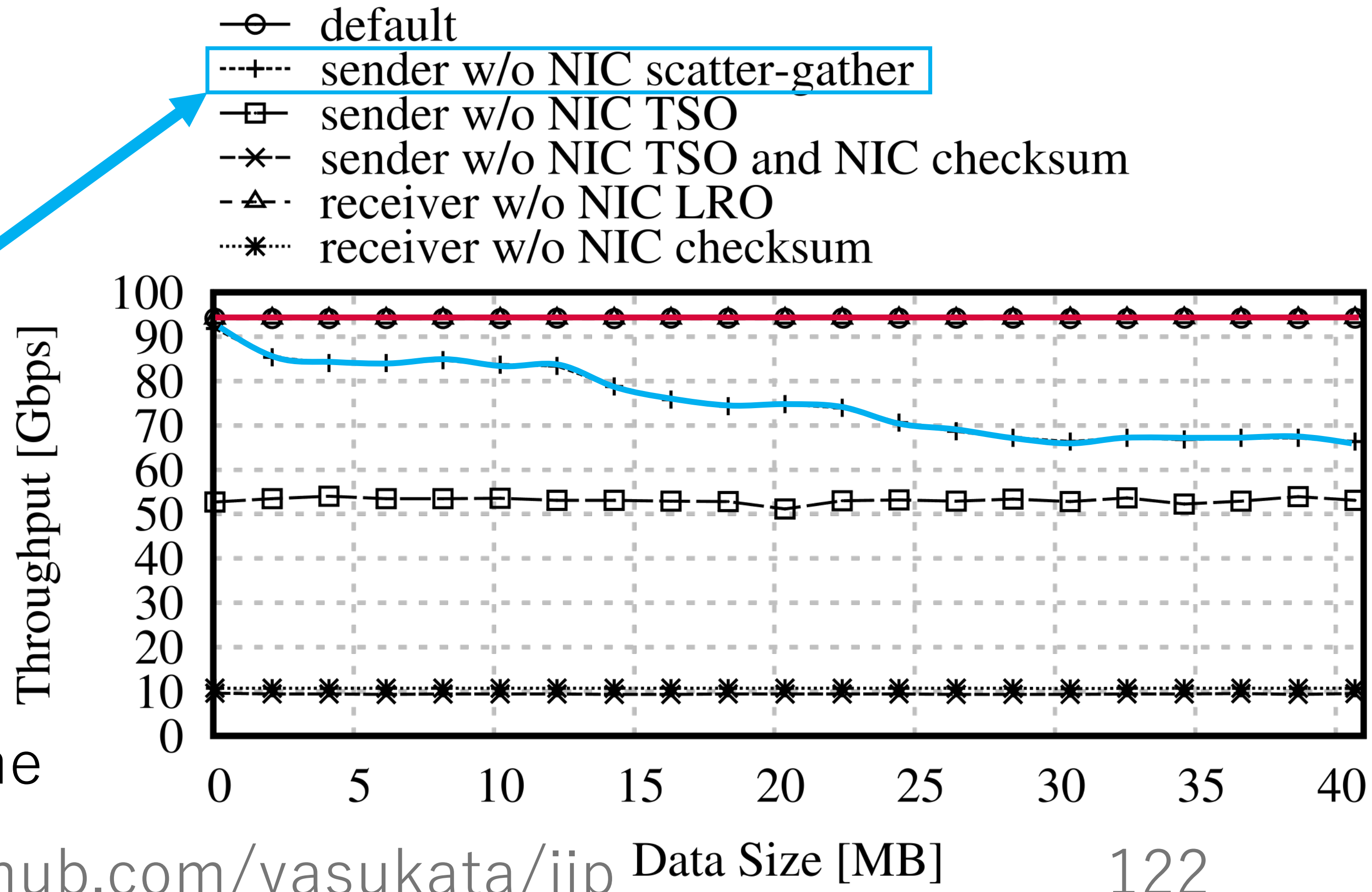


Evaluation: Bulk Data Transfer

- The sender repeatedly sends the same data to the receiver
- Performance factors
  - Zero-copy transmission

When the sender deactivates zero-copy transmission, throughput is degraded according to the size of the data

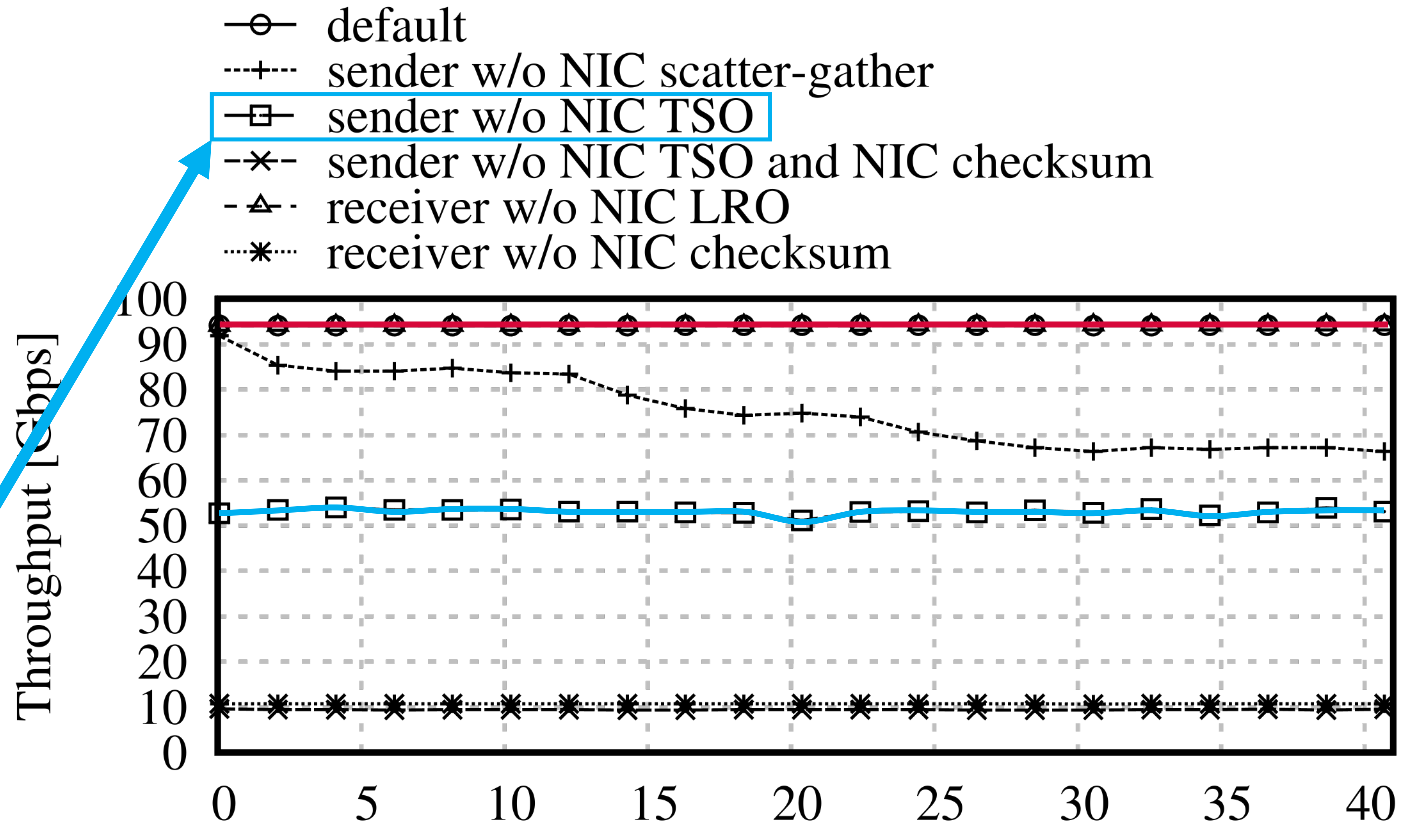
We consider this happens because the payload occupies the CPU cache



Evaluation: Bulk Data Transfer

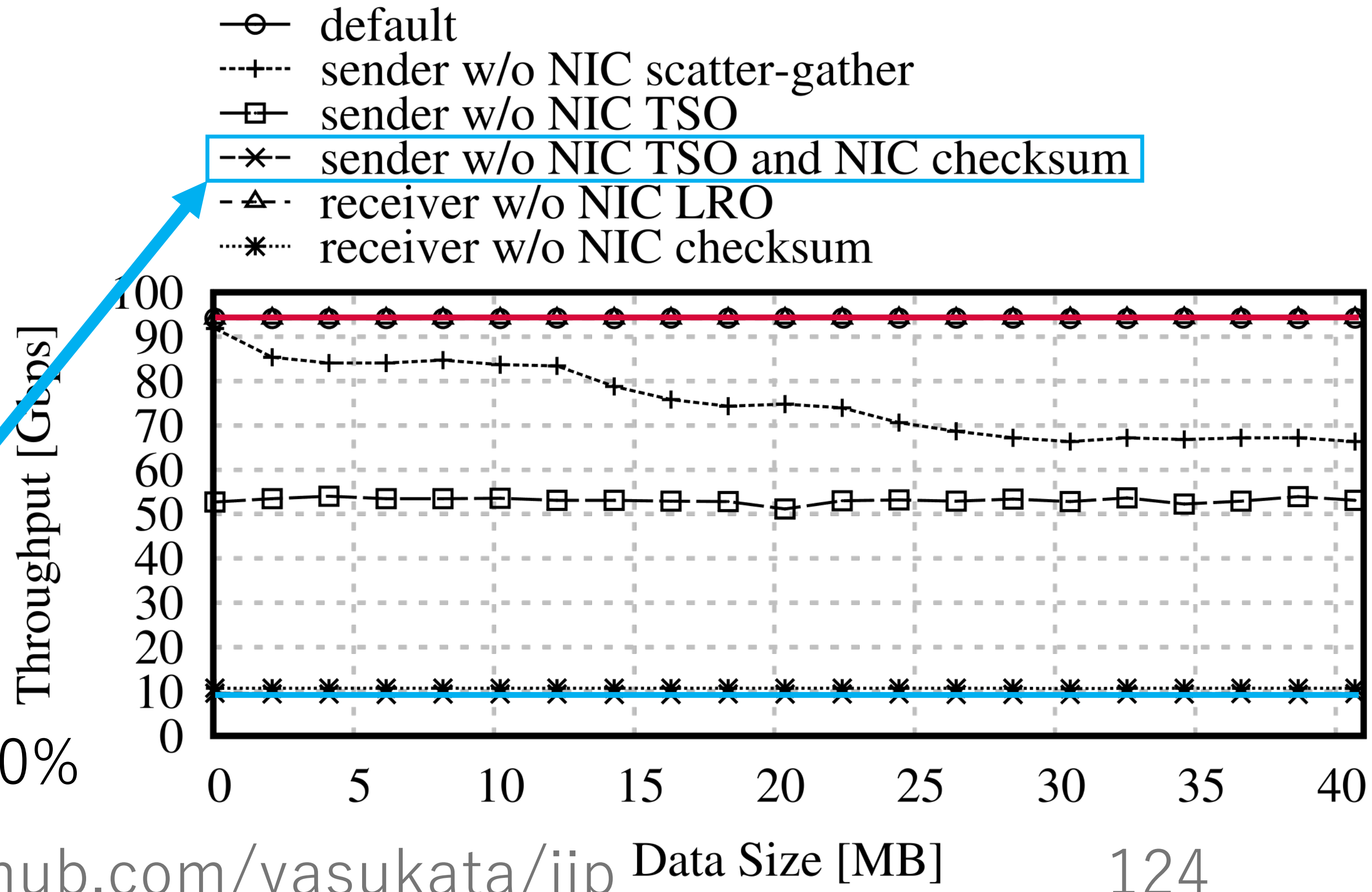
- The sender repeatedly sends the same data to the receiver
- Performance factors
  - Zero-copy transmission
  - TSO

When the sender deactivates TSO, throughput is almost halved



Evaluation: Bulk Data Transfer

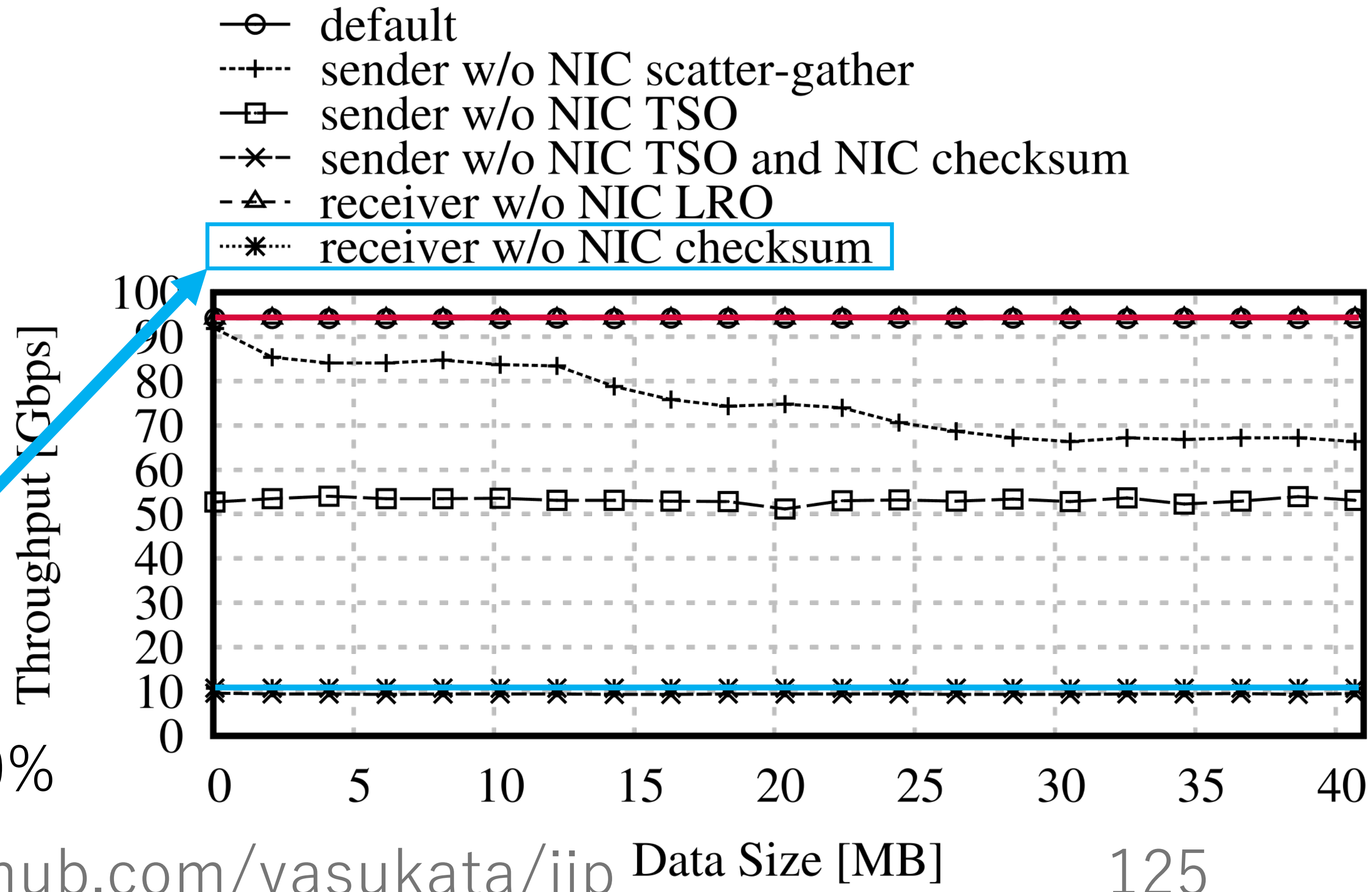
- The sender repeatedly sends the same data to the receiver
- Performance factors
  - Zero-copy transmission
  - TSO
  - Checksum offloading



When the sender deactivates TSO and checksum offloading, throughput goes down to around 10%

Evaluation: Bulk Data Transfer

- The sender repeatedly sends the same data to the receiver
- Performance factors
  - Zero-copy transmission
  - TSO
  - Checksum offloading

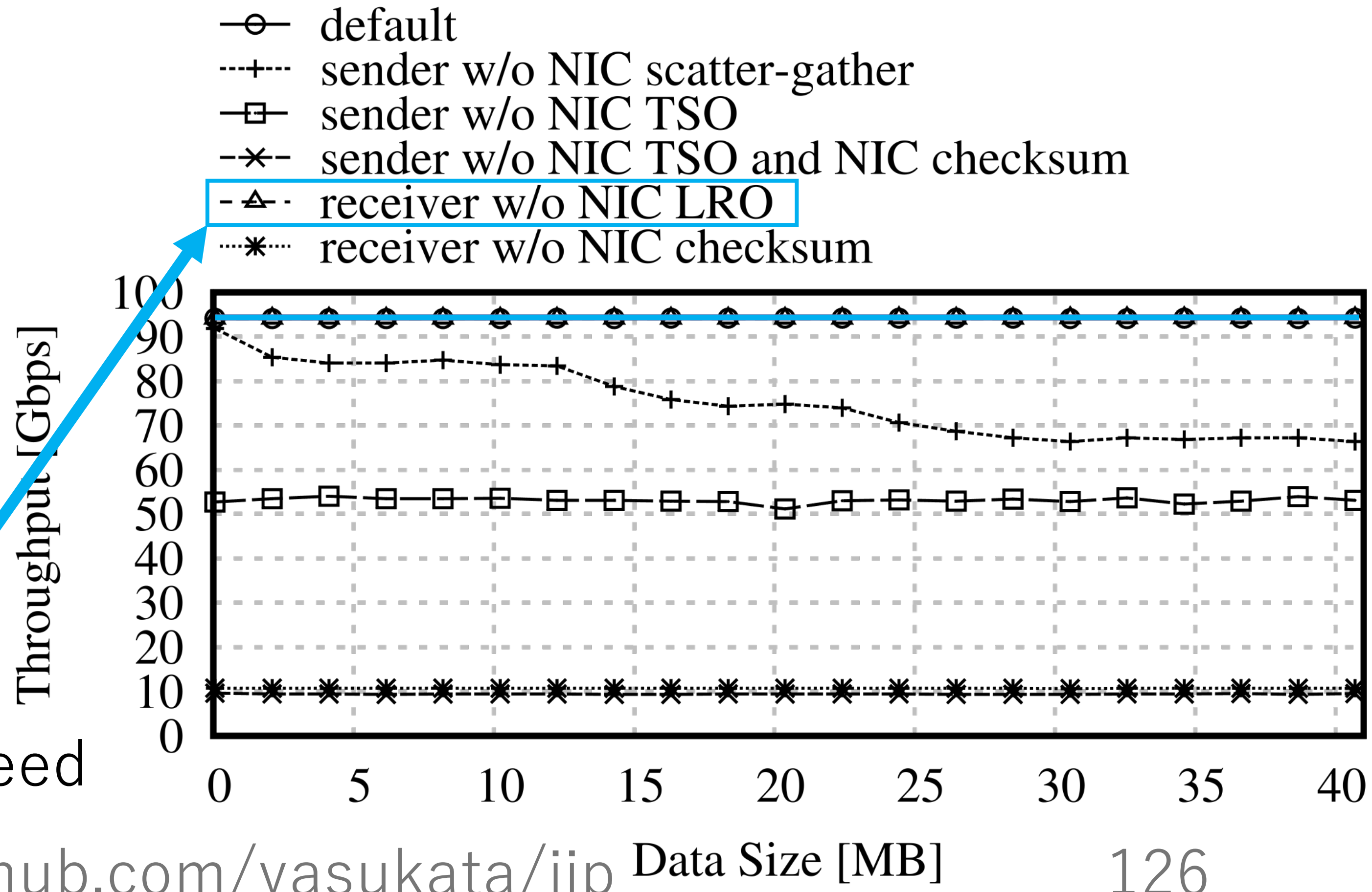


When the receiver deactivates checksum offloading, throughput goes down to around 10%

https://github.com/yasukata/iip

Evaluation: Bulk Data Transfer

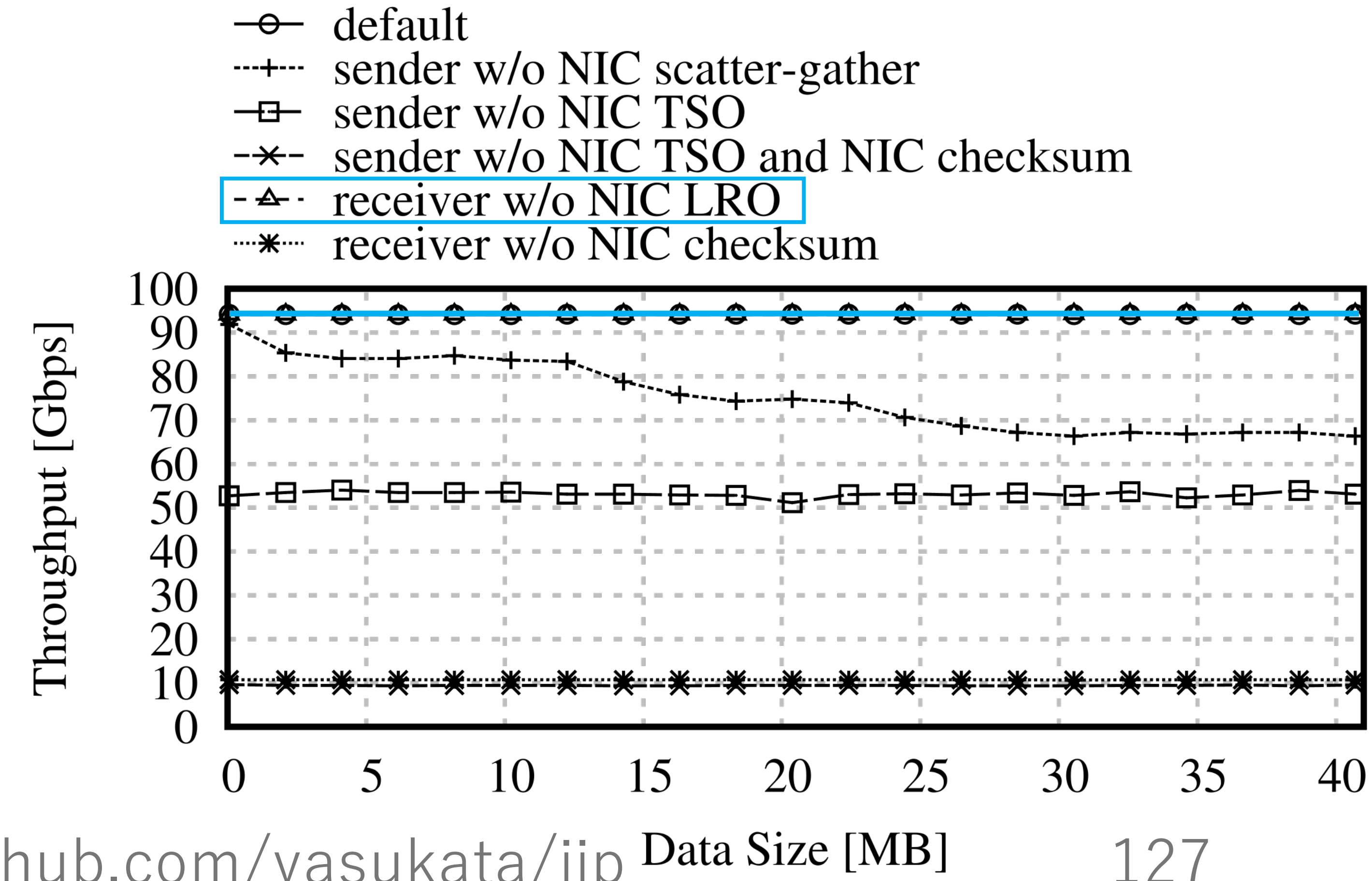
- The sender repeatedly sends the same data to the receiver
- Performance factors
  - Zero-copy transmission
  - TSO
  - Checksum offloading



When the receiver disables LRO,
it can still catch up with the link speed

Evaluation: Bulk Data Transfer

- The sender repeatedly sends the same data to the receiver
- Performance factors
  - Zero-copy transmission
  - TSO
  - Checksum offloading
- Note
  - This does not mean LRO is not necessary
    - Just we did not see differences in this workload



Summary

- iip is a TCP/IP stack implementation that aims to allow for easy integration and good performance simultaneously

Please try it if you are interested

- Main page: https://github.com/yasukata/iip
- Assets used in the paper: https://github.com/yasukata/bench-iip/tree/9cf2488ec93ae51f4bd7b18923a5d1a233852f66