

Open Networking Conference Japan 2024 – 2024 年 10 月 11 日

# 何故今時 TCP/IP スタックを 新しく実装しようとするのか

株式会社インターネットイニシアティブ  
技術研究所 安形 憲一



Internet Initiative Japan

# 本発表の内容

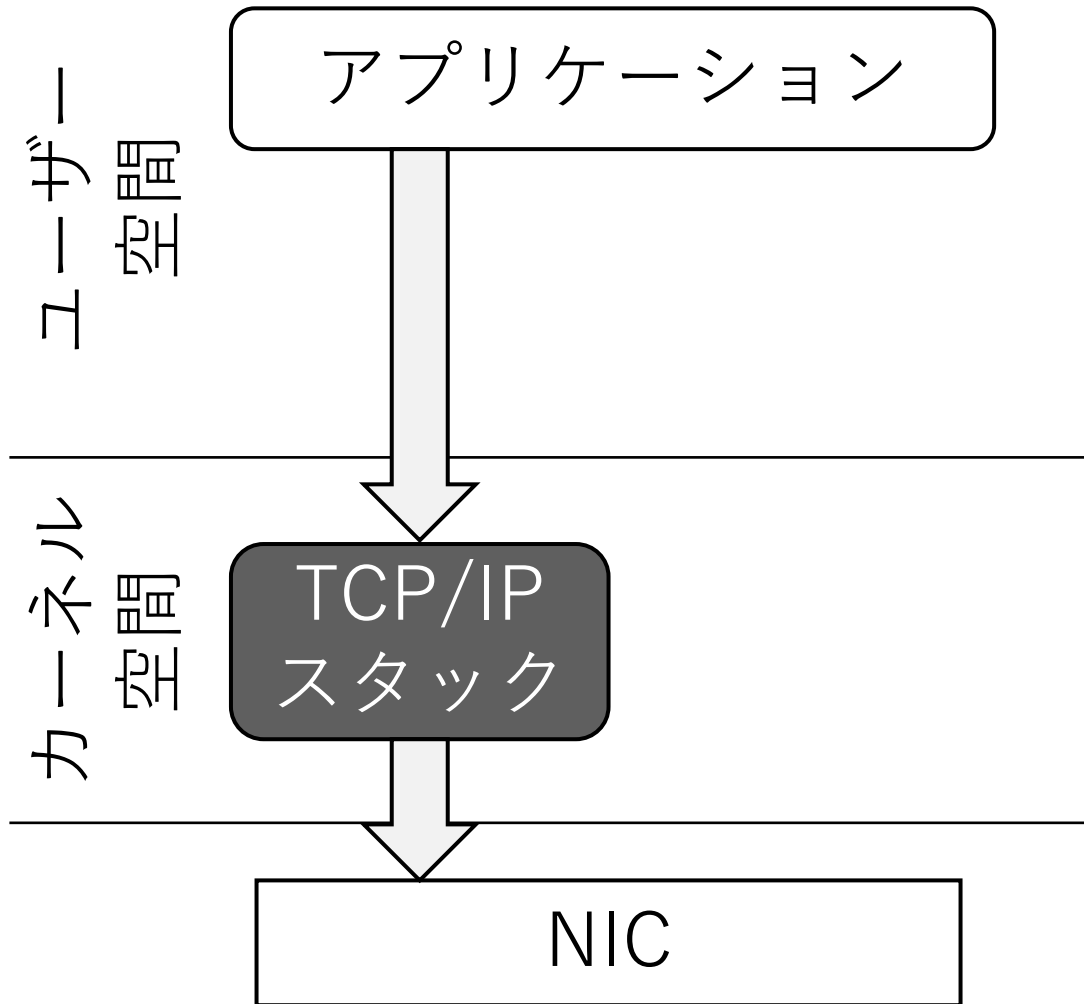
- IJ 技術研究所では新しく TCP/IP スタックを実装しています
  - <https://eng-blog.ij.ad.jp/archives/25852>
- 一方、TCP/IP スタックは数十年前から OS の一部として開発が継続されており、既に成熟した実装が広く利用されています
- そんな枯れた実装があるにも関わらず、何故また新しく TCP/IP スタックを実装しているのかについてご説明します

# 表題への直接的な回答

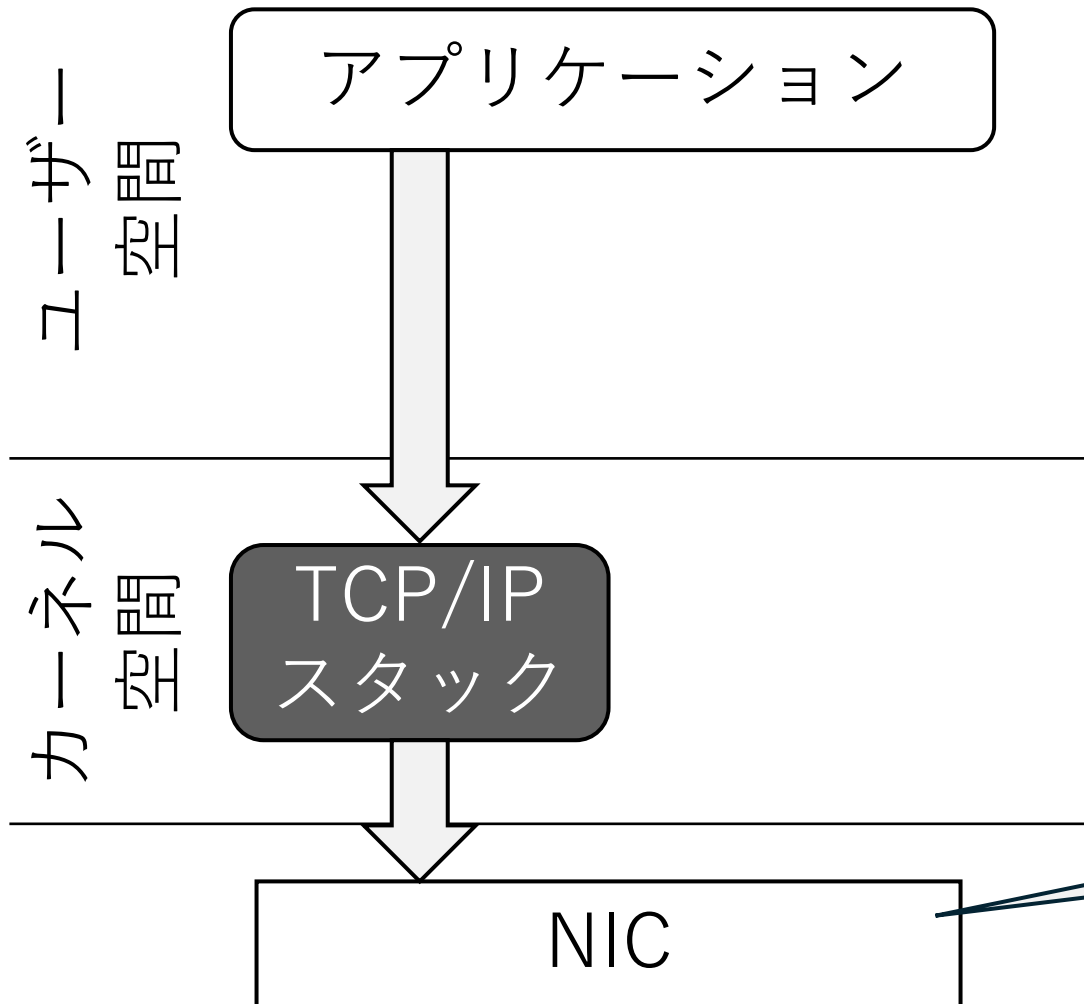
Q. 何故今時 TCP/IP スタックを新しく実装しようとするのか

A. 既存の広く利用されている実装では比較的新しい  
通信ハードウェアの性能を活かすことが難しいから

# 既存の TCP/IP スタック実装の性能

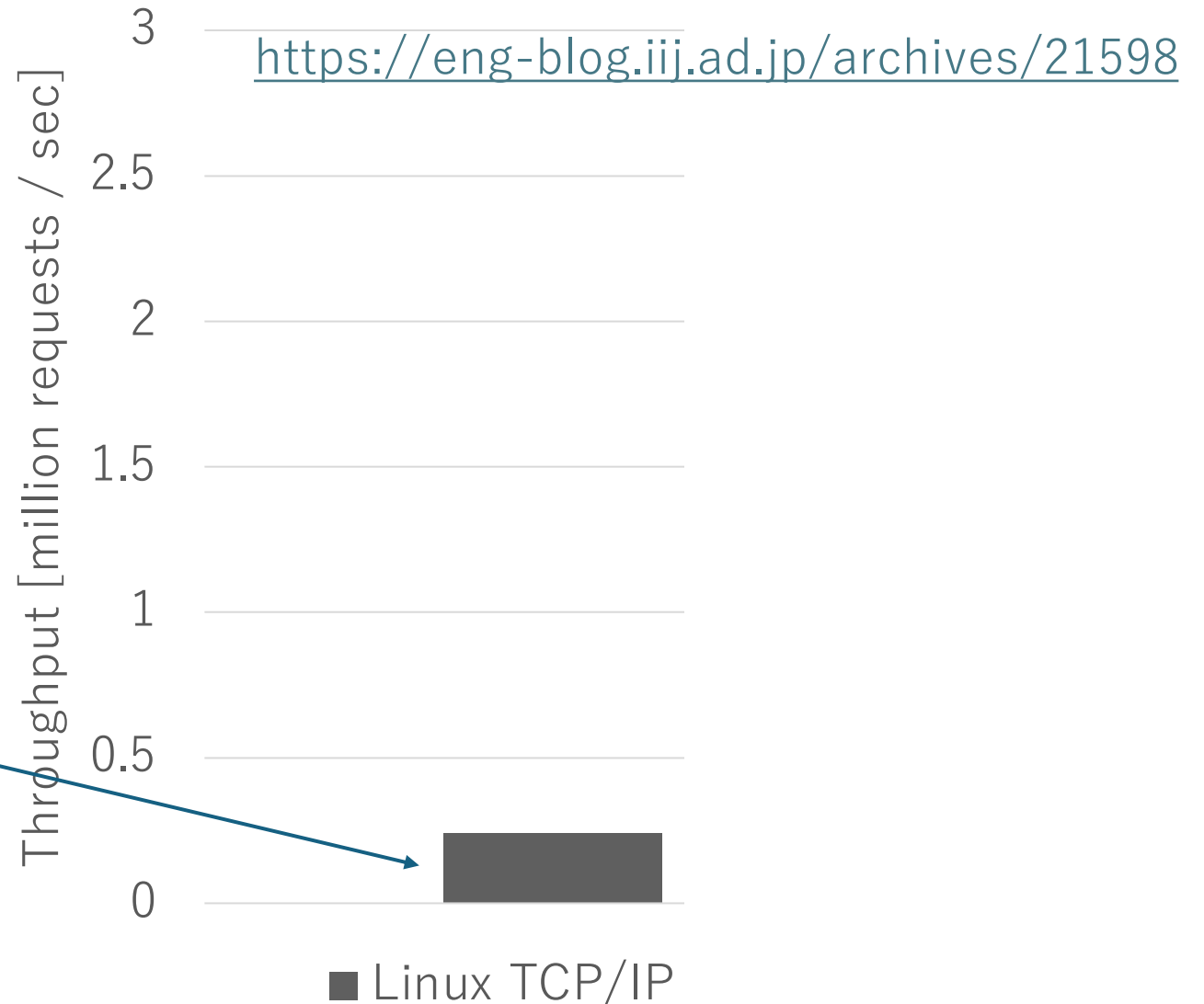
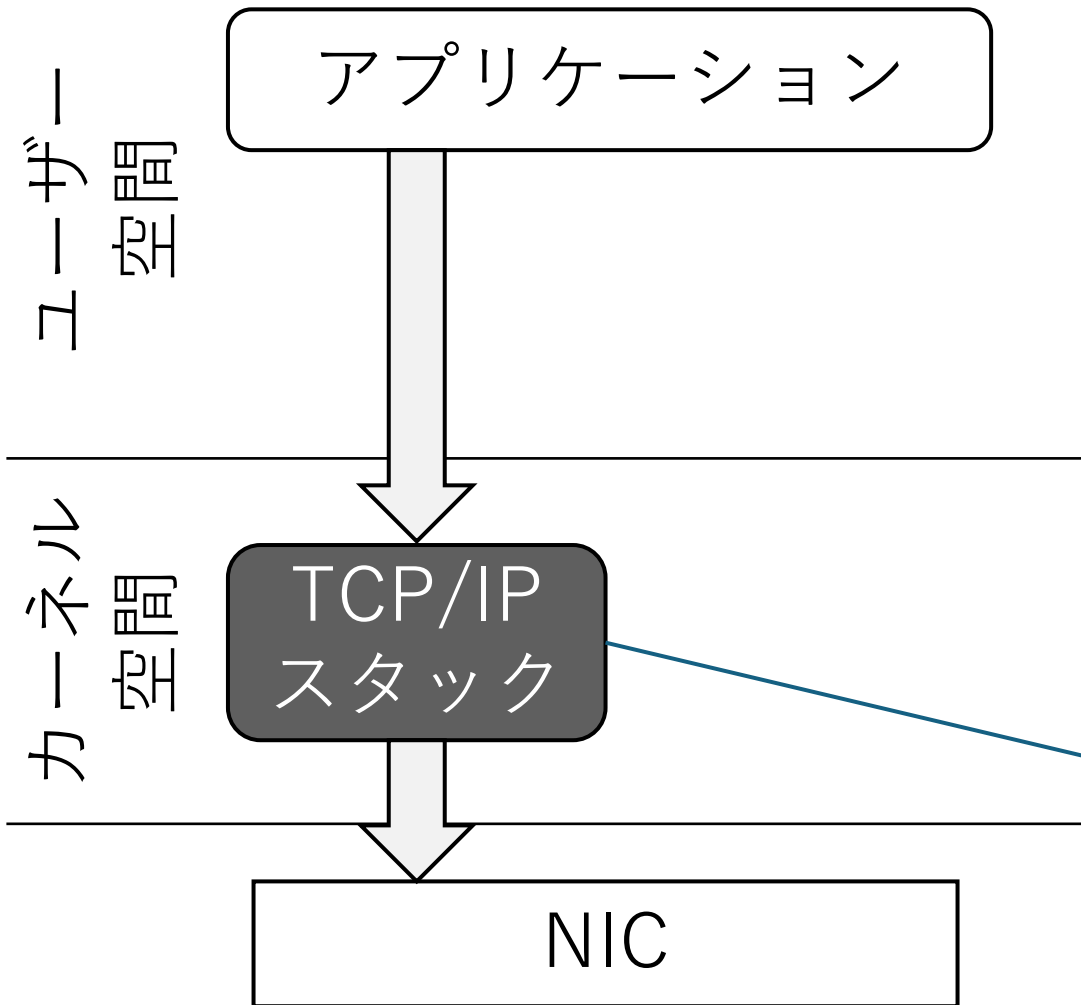


# 既存の TCP/IP スタック実装の性能

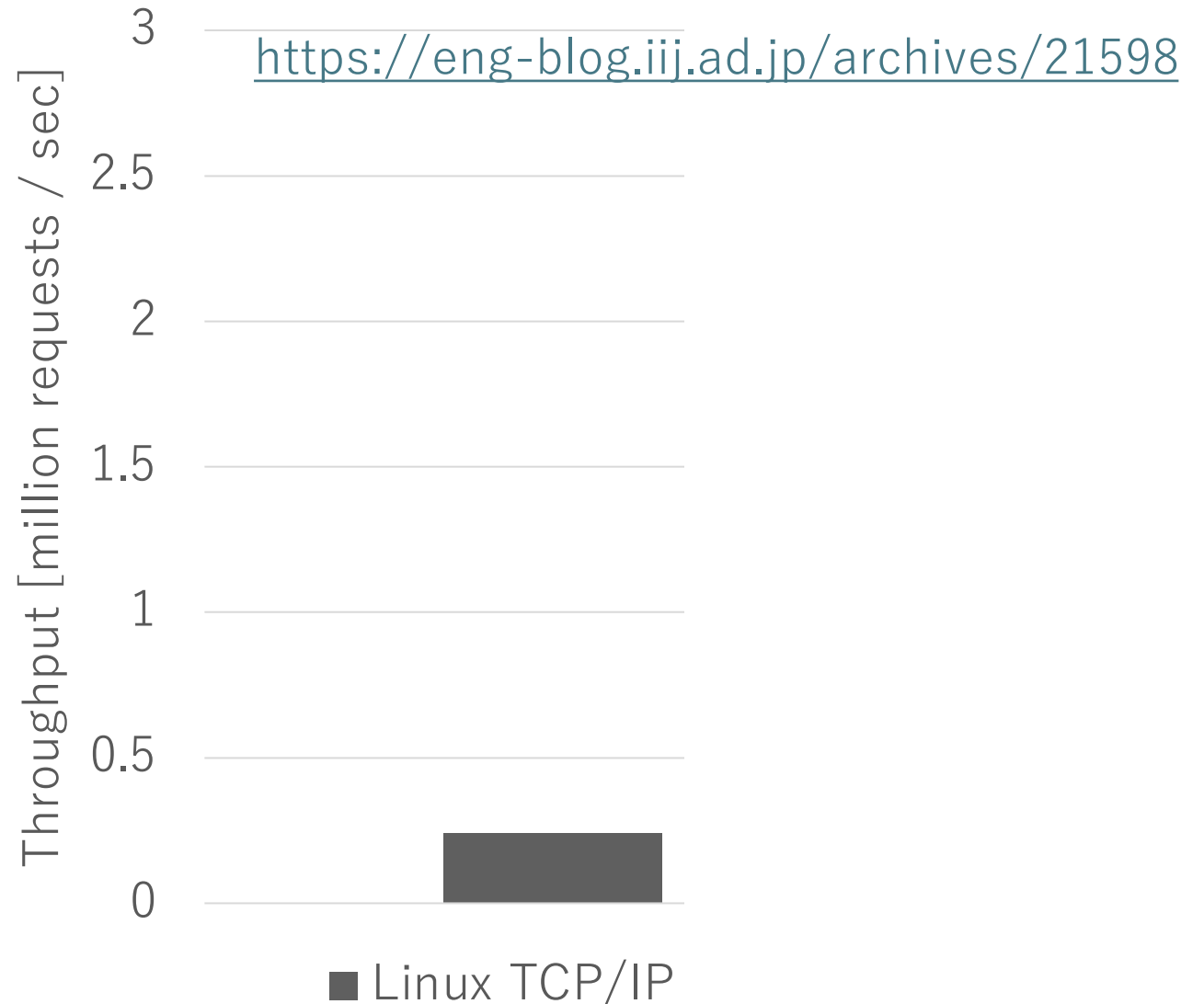
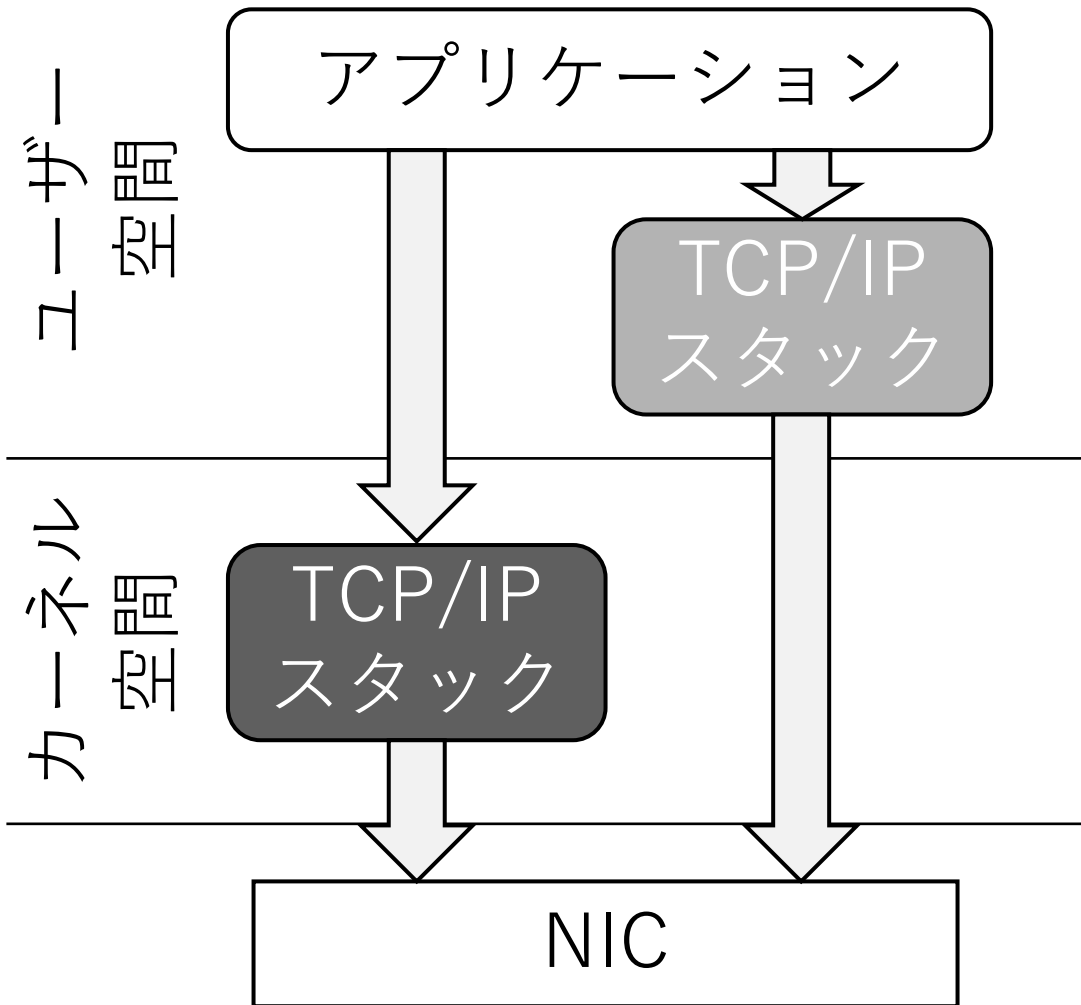


データセンター等の環境では  
10 Gbps を超える速度が一般的

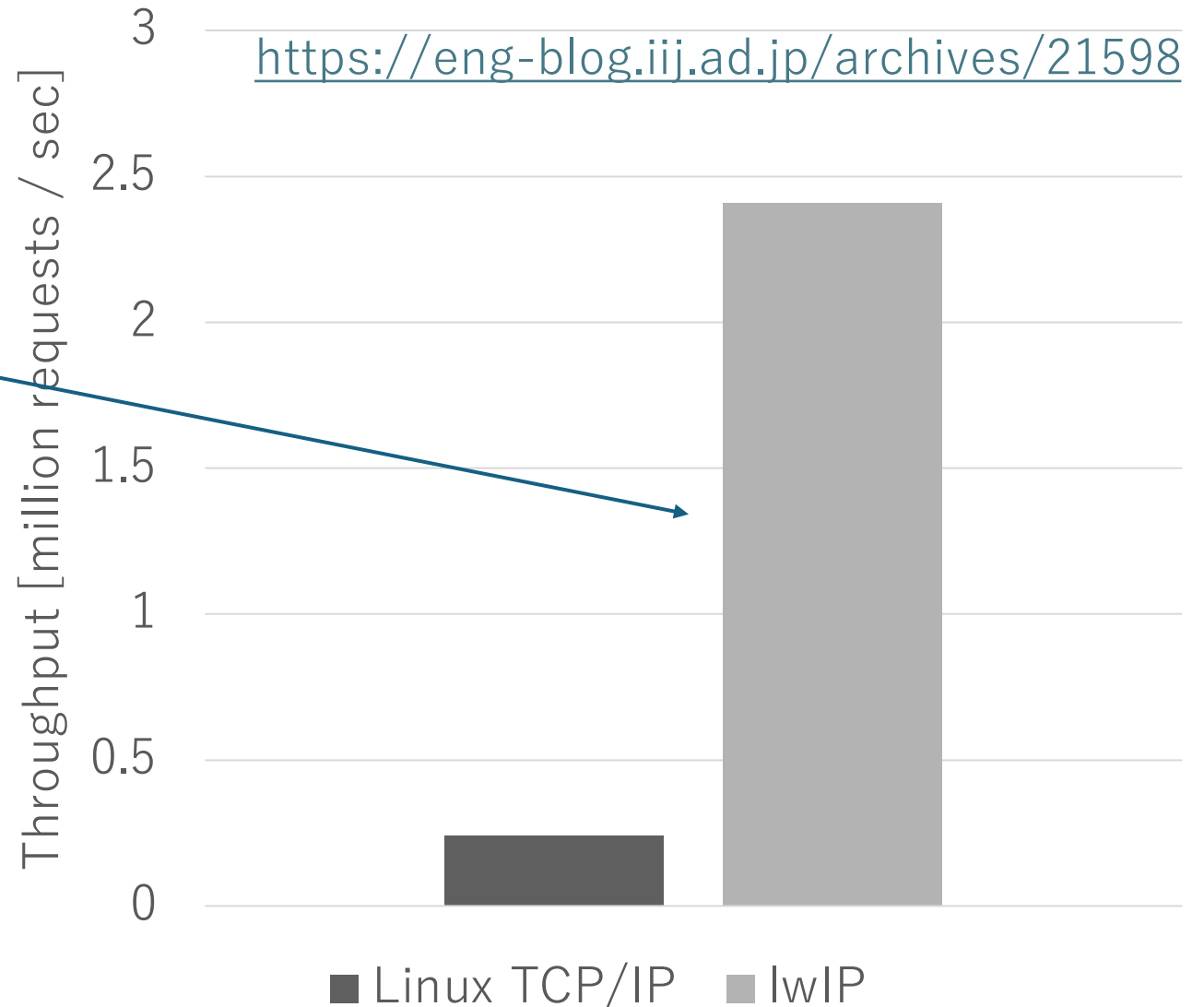
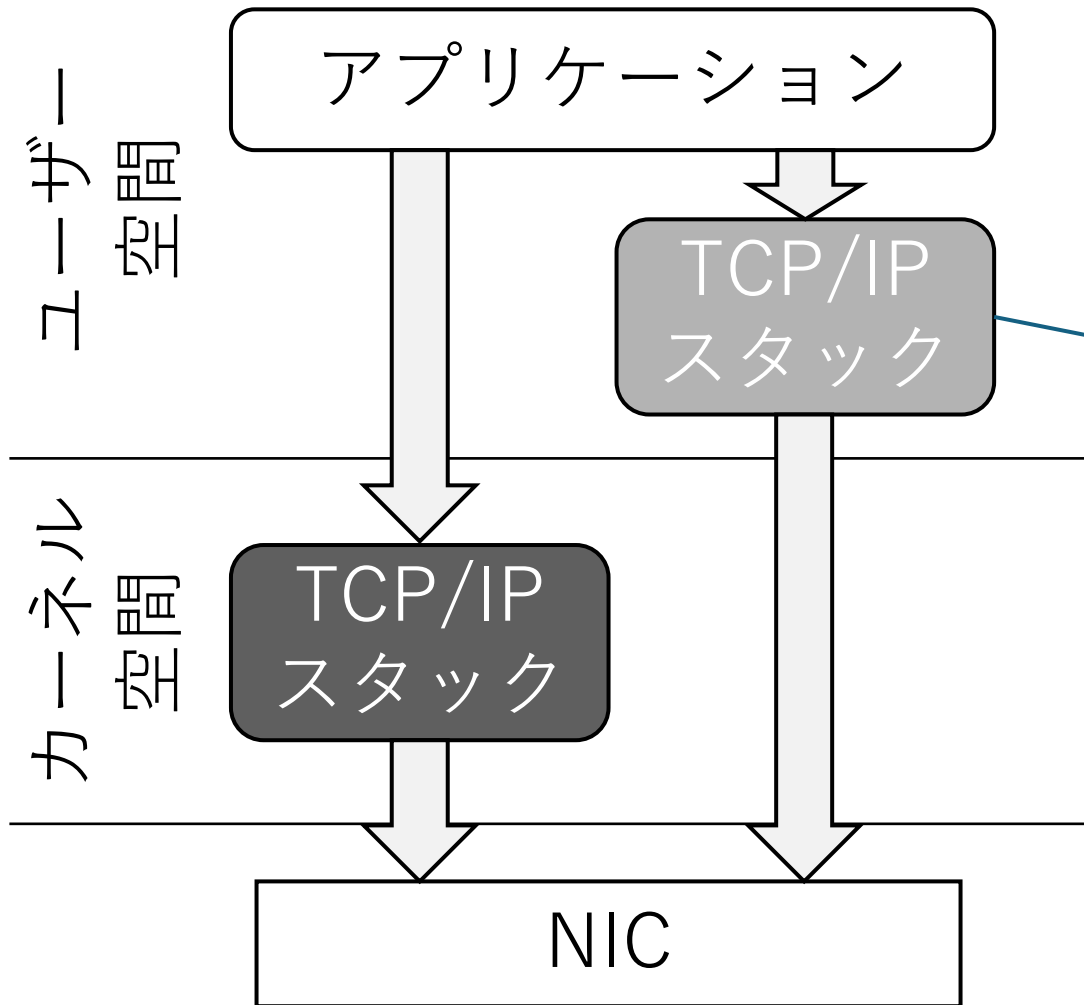
# 既存の TCP/IP スタック実装の性能



# 既存の TCP/IP スタック実装の性能

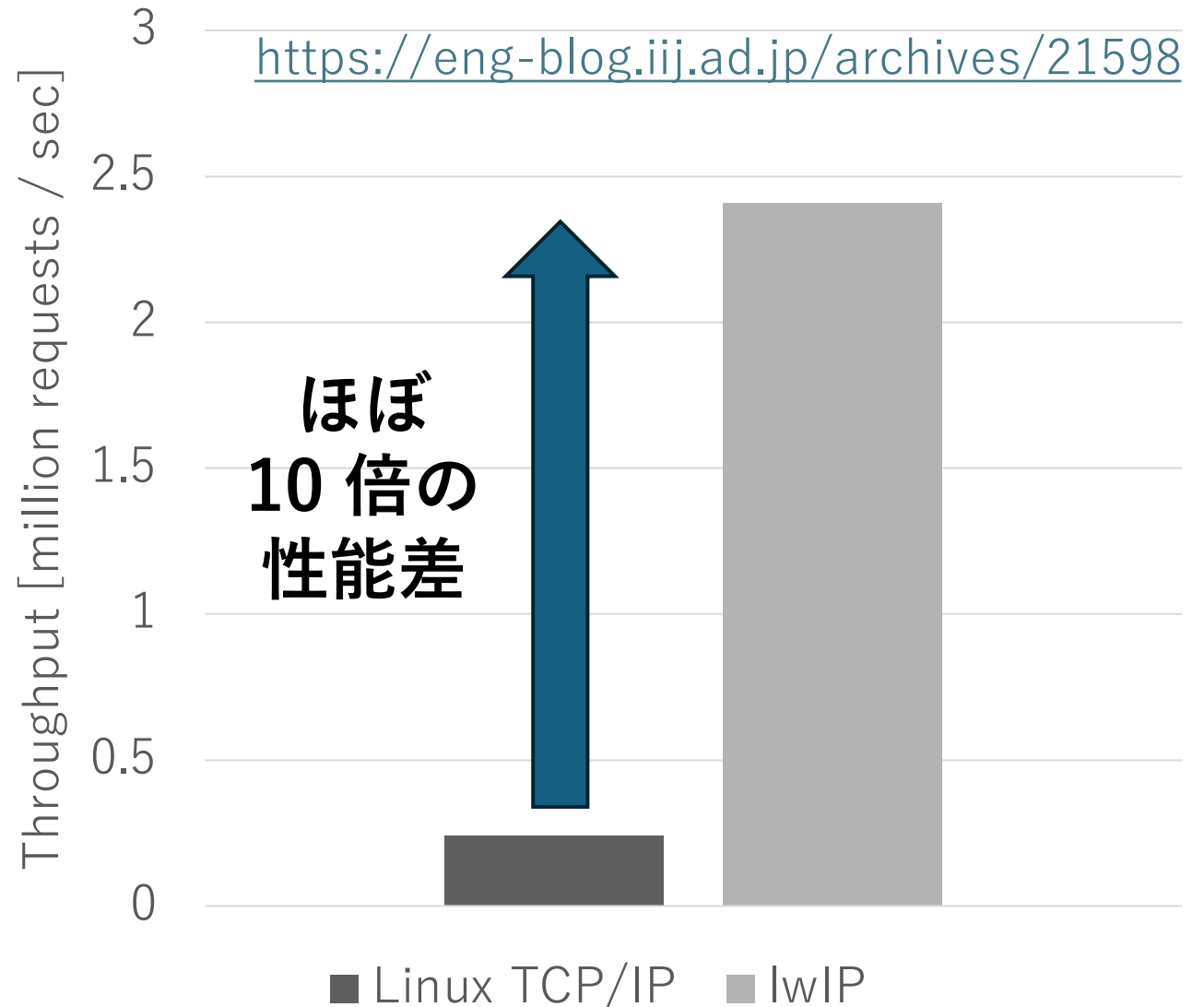
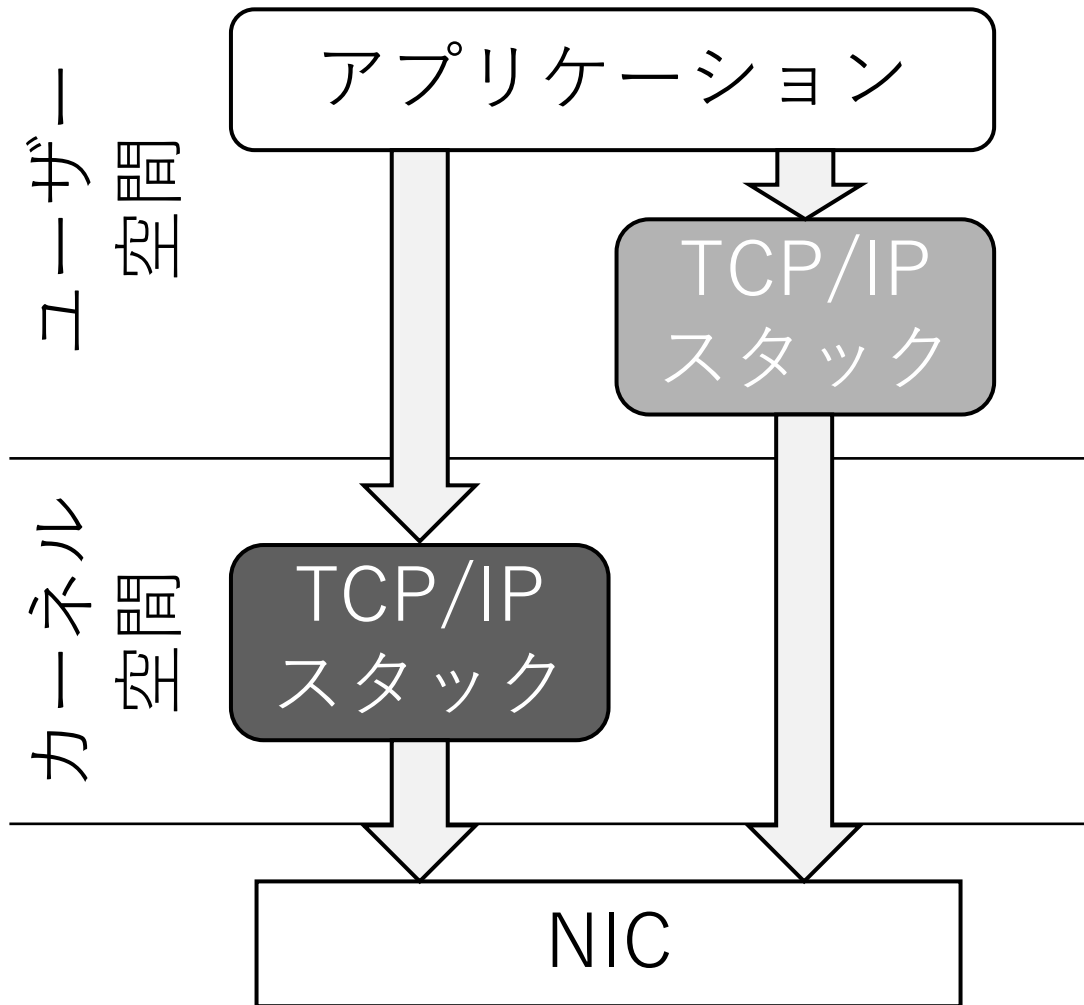


# 既存の TCP/IP スタック実装の性能



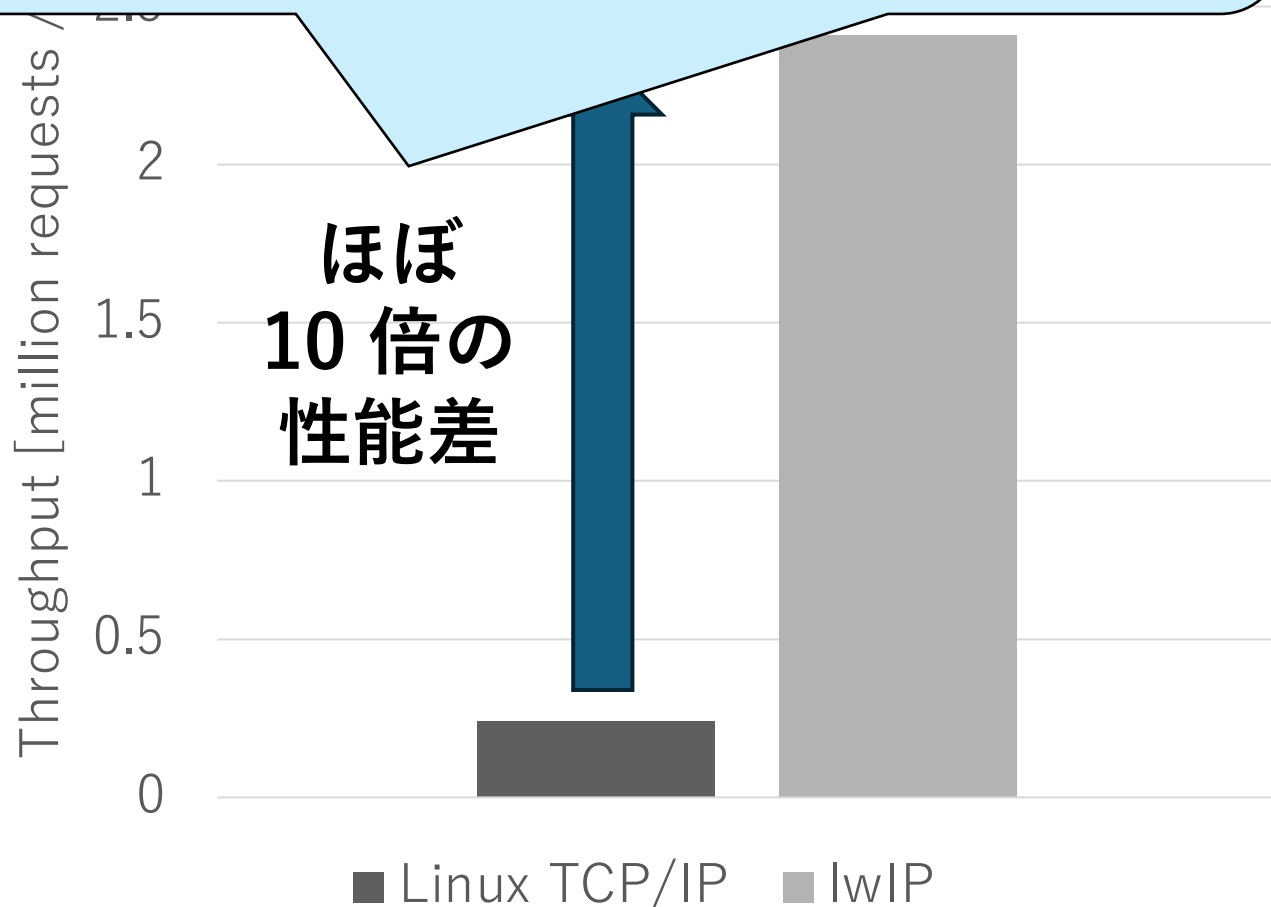
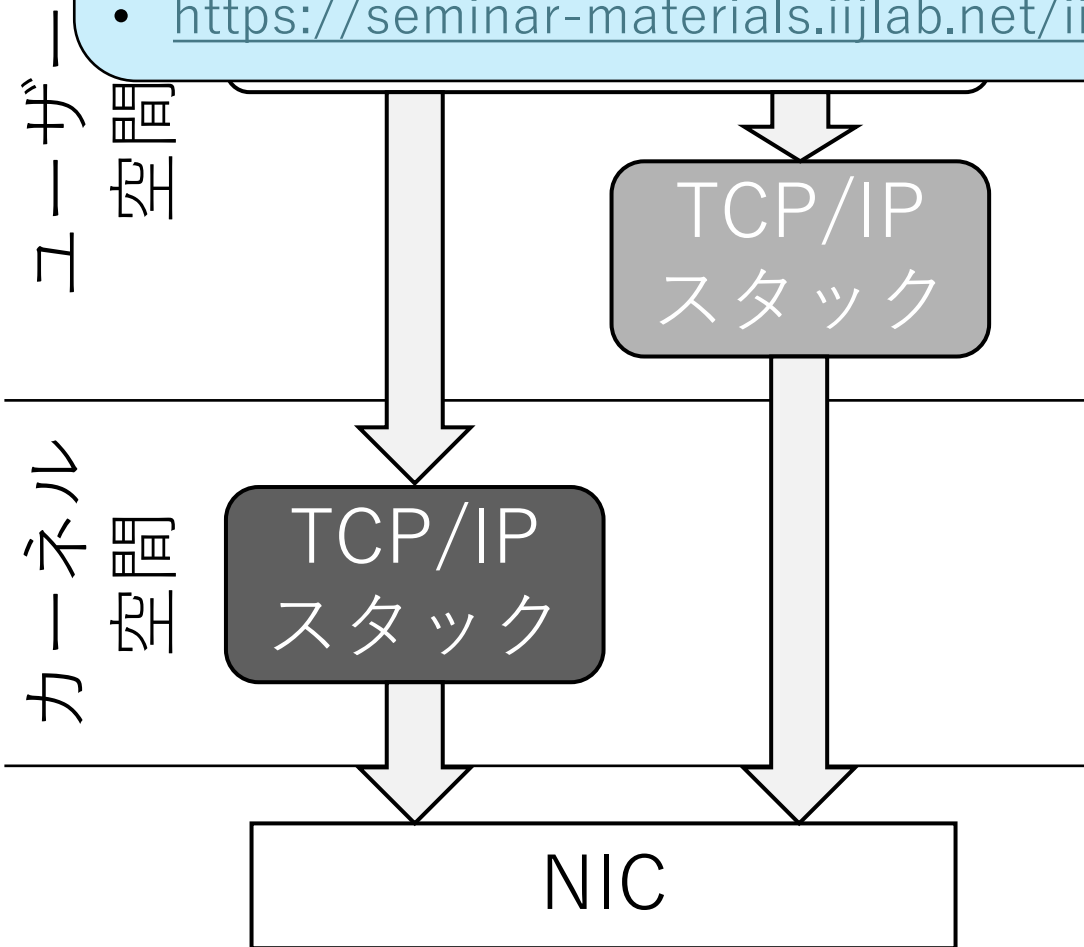


# 既存の TCP/IP スタック実装の性能



速度面で重要な点は 10 年前にはある程度研究で判明しており  
それらに注意すれば多くの場合で高速な実装を実現できる

- <https://www.iij.ad.jp/dev/report/iir/060/02.html>
- <https://seminar-materials.iijlab.net/iijlab-seminar/iijlab-seminar-20231017.pdf> (16 MB)

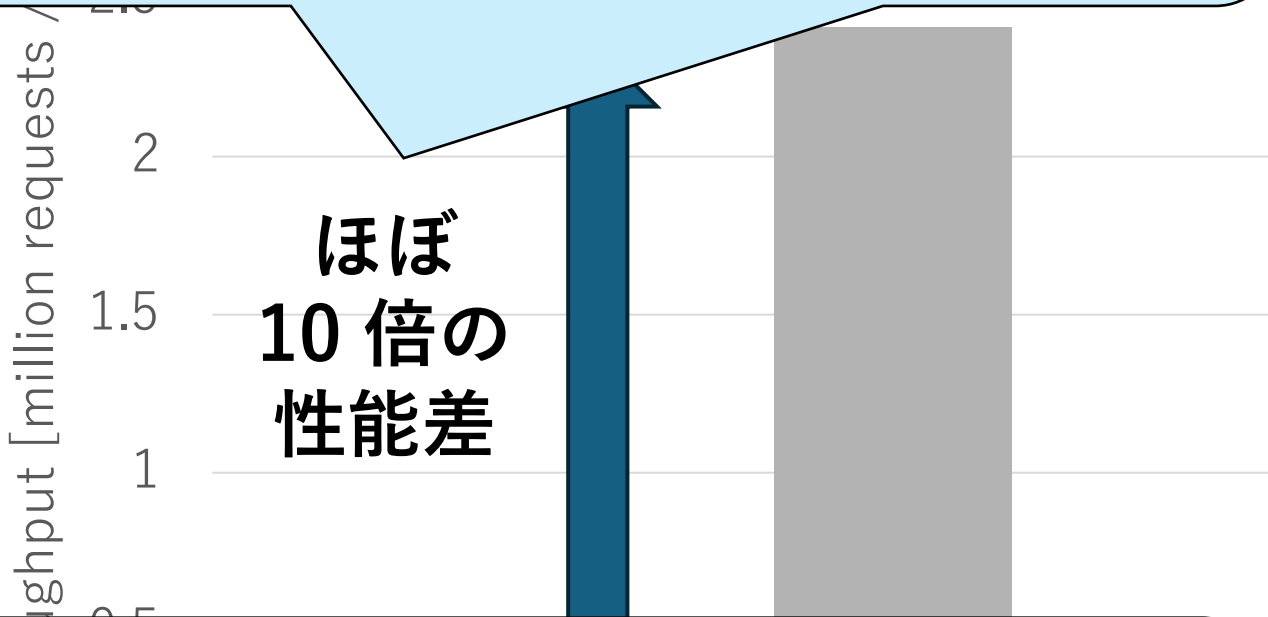
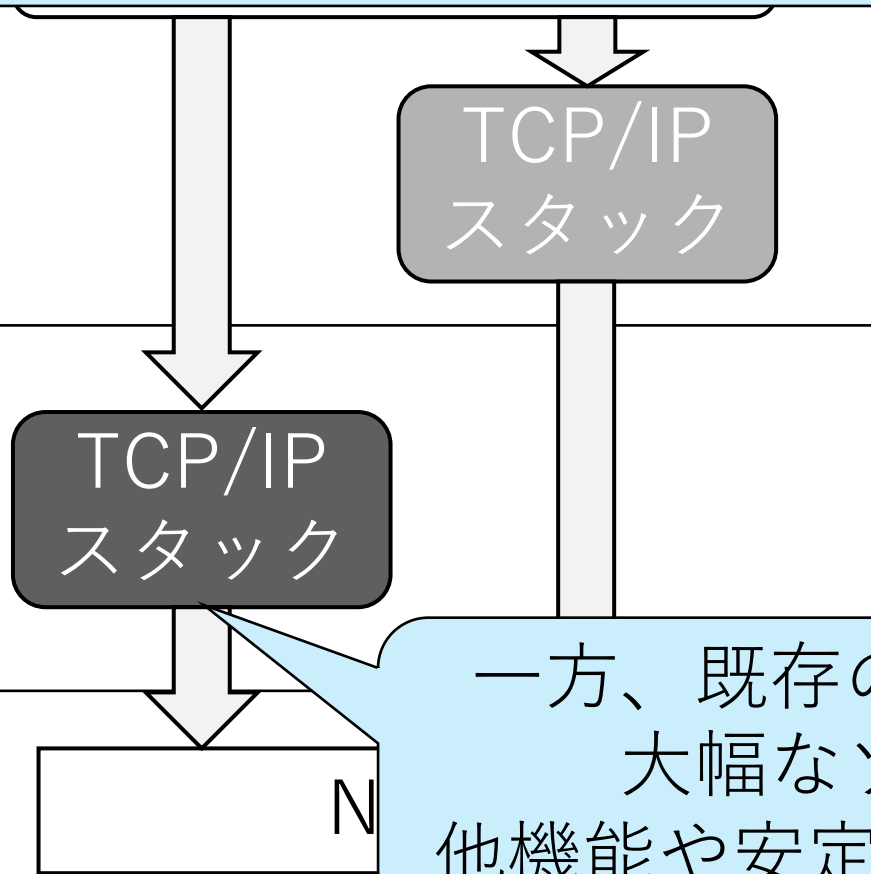


速度面で重要な点は 10 年前にはある程度研究で判明しており  
それらに注意すれば多くの場合で高速な実装を実現できる

- <https://www.iij.ad.jp/dev/report/iir/060/02.html>
- <https://seminar-materials.iijlab.net/iijlab-seminar/iijlab-seminar-20231017.pdf> (16 MB)

ユーザー空間

カーネル空間



一方、既存の広く利用されている実装の高速化は  
大幅なソースコードの改変が必要なため  
他機能や安定性への影響を考慮すると難易度が高い

速度面で重要な点は 10 年前にはある程度研究で判明しており  
それらに注意すれば多くの場合で高速な実装を実現できる

- <https://www.iij.ad.jp/dev/report/iir/060/02.html>
- <https://seminar-materials.iijlab.net/iijlab-seminar/iijlab-seminar-20231017.pdf> (16 MB)

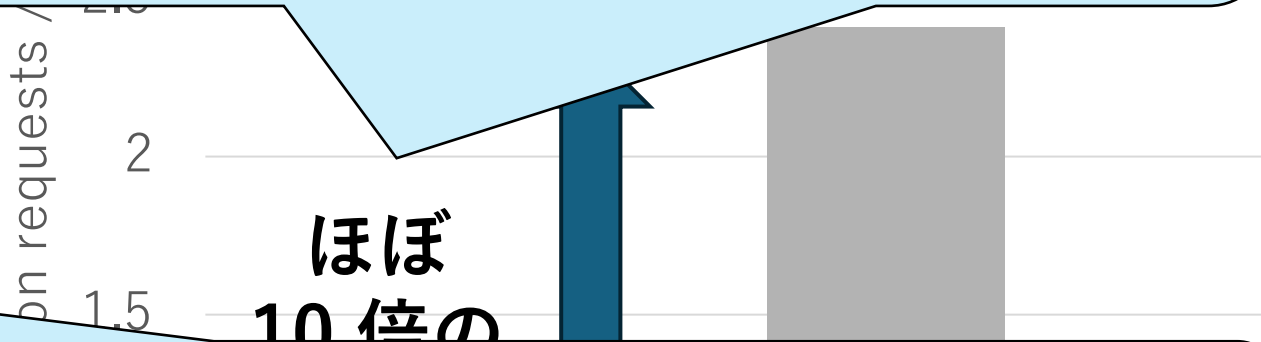
ユーザー空間

カーネル空間

TCP/IP  
スタック

TCP/IP  
スタック

N



既存のカーネル内実装とは別に TCP/IP スタックを開発していくのは現実的な方向性であると思われる

一方、既存の広く利用されている実装の高速化は大幅なソースコードの改変が必要のため他機能や安定性への影響を考慮すると難易度が高い

速度面で重要な点は **10年前にはある程度研究で判明しており**  
それらに注意すれば多くの場合で高速な実装を実現できる

- <https://www.iij.ad.jp/dev/report/iir/060/02.html>
- <https://seminar-materials.iijlab.net/iijlab-seminar/iijlab-seminar-20231017.pdf> (16 MB)

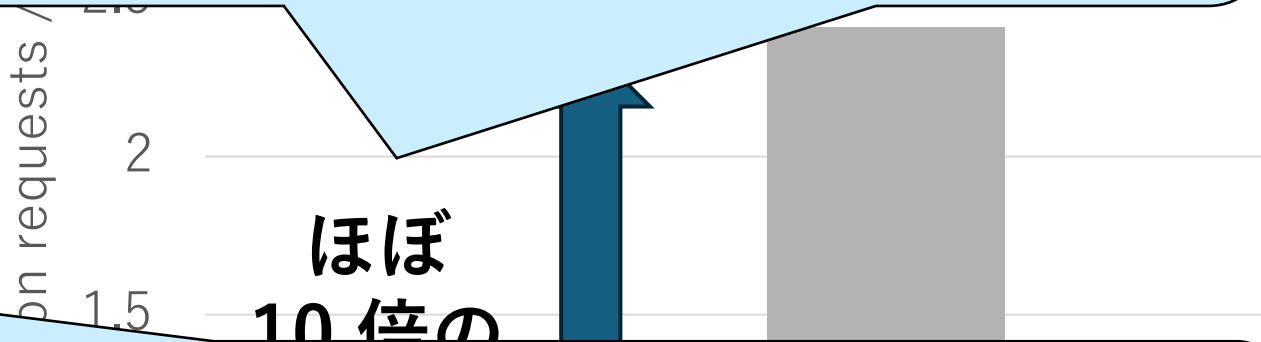
ユーザー空間

カーネル空間

TCP/IP  
スタック

TCP/IP  
スタック

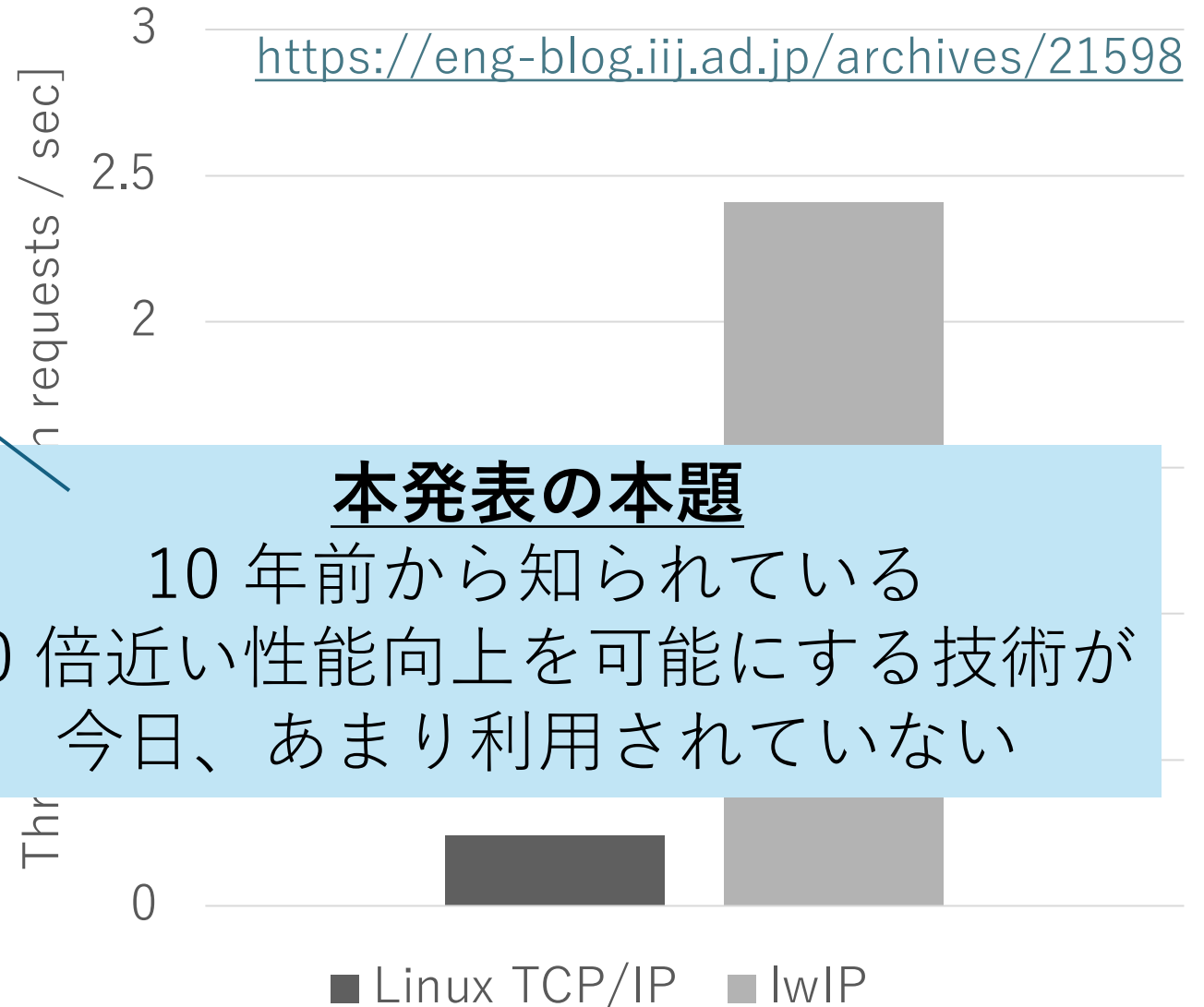
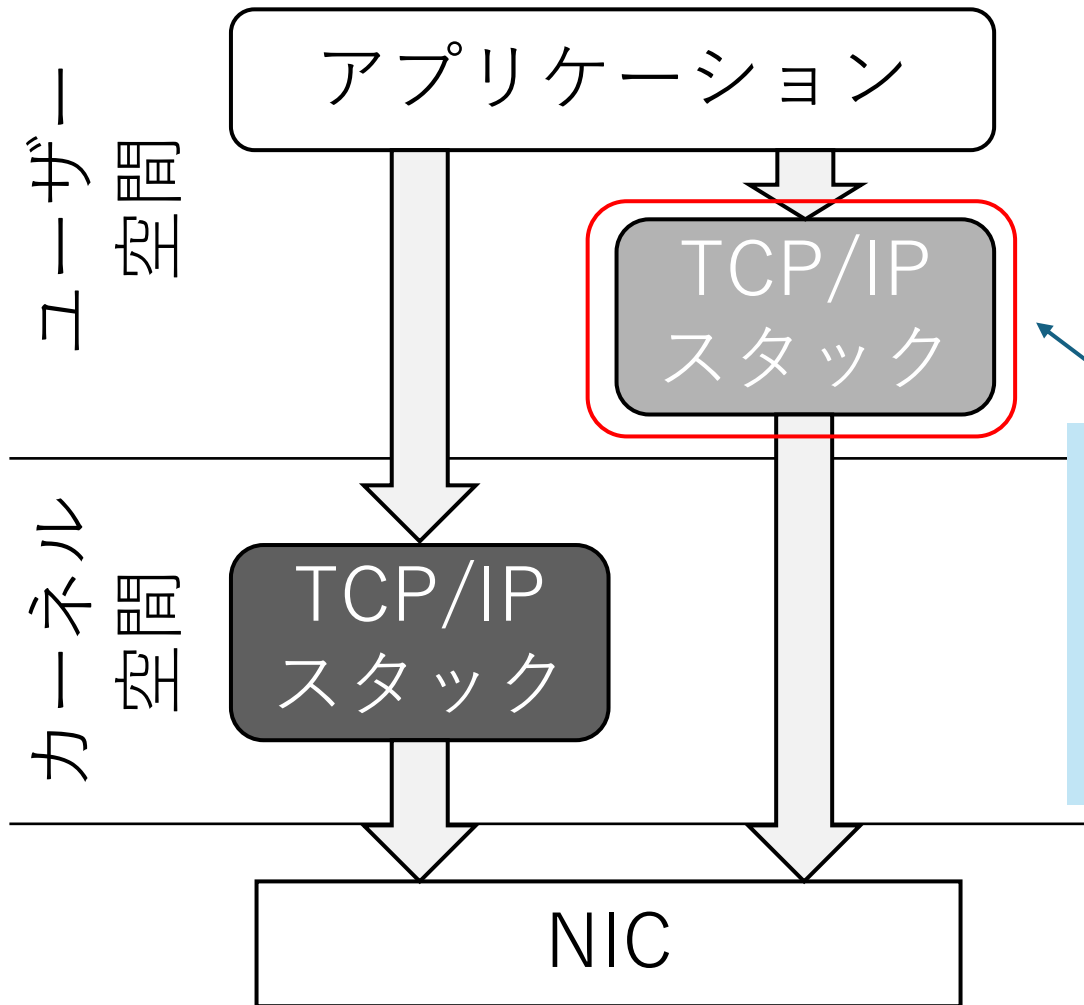
N



既存のカーネル内実装とは別に TCP/IP スタックを開発していくのは現実的な方向性であると思われる

一方、既存の広く利用されている実装の高速化は大幅なソースコードの改変が必要のため他機能や安定性への影響を考慮すると難易度が高い

# 既存の TCP/IP スタック実装の性能



# この 10 年間を鑑みて

- この 10 年間で TCP/IP スタックを高速化する手法を発見するだけでは広範な利用へは至らないことが確認された

# この 10 年間を鑑みて

- この 10 年間で TCP/IP スタックを高速化する手法を発見するだけでは広範な利用へは至らないことが確認された
- 大幅な高速化を実現する手法が広く利用されないということは何かまだ解決されなければならない課題があるはず



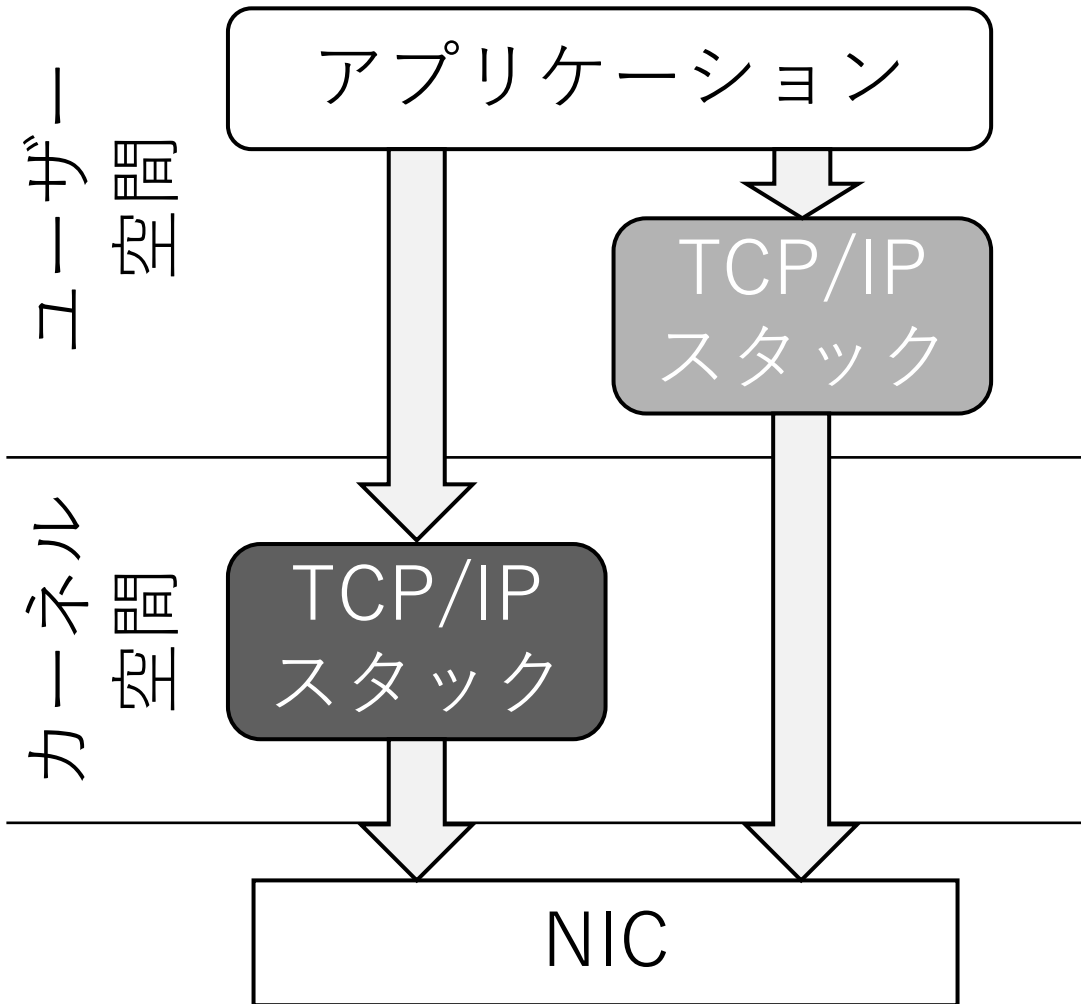
# この10年を鑑みて

- この10年間でTCP/IPスタックを高速化する手法を発見するだけでは広範な利用へは至らないことが確認された
- 大幅な高速化を実現する手法が広く利用されないということは何かまだ解決されなければならない課題があるはず

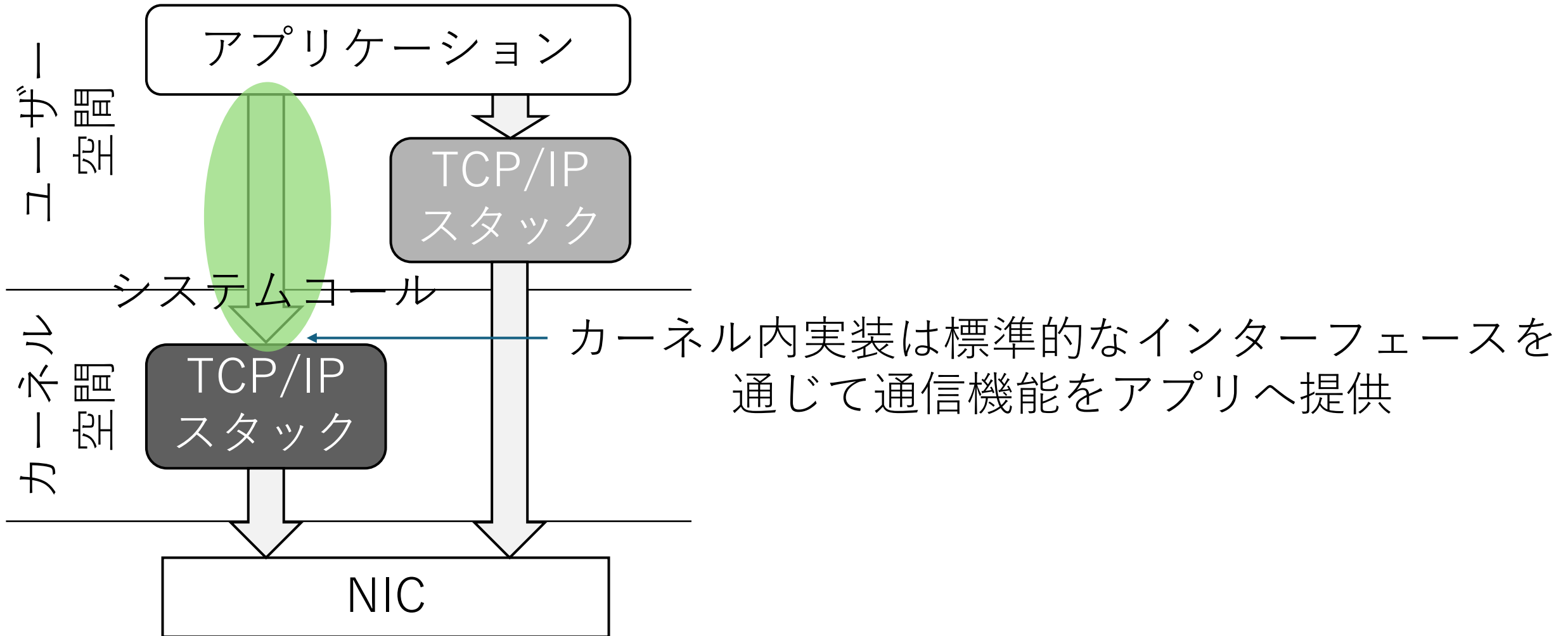
仮説

利用者にとっての使いやすさに問題があるのでは？

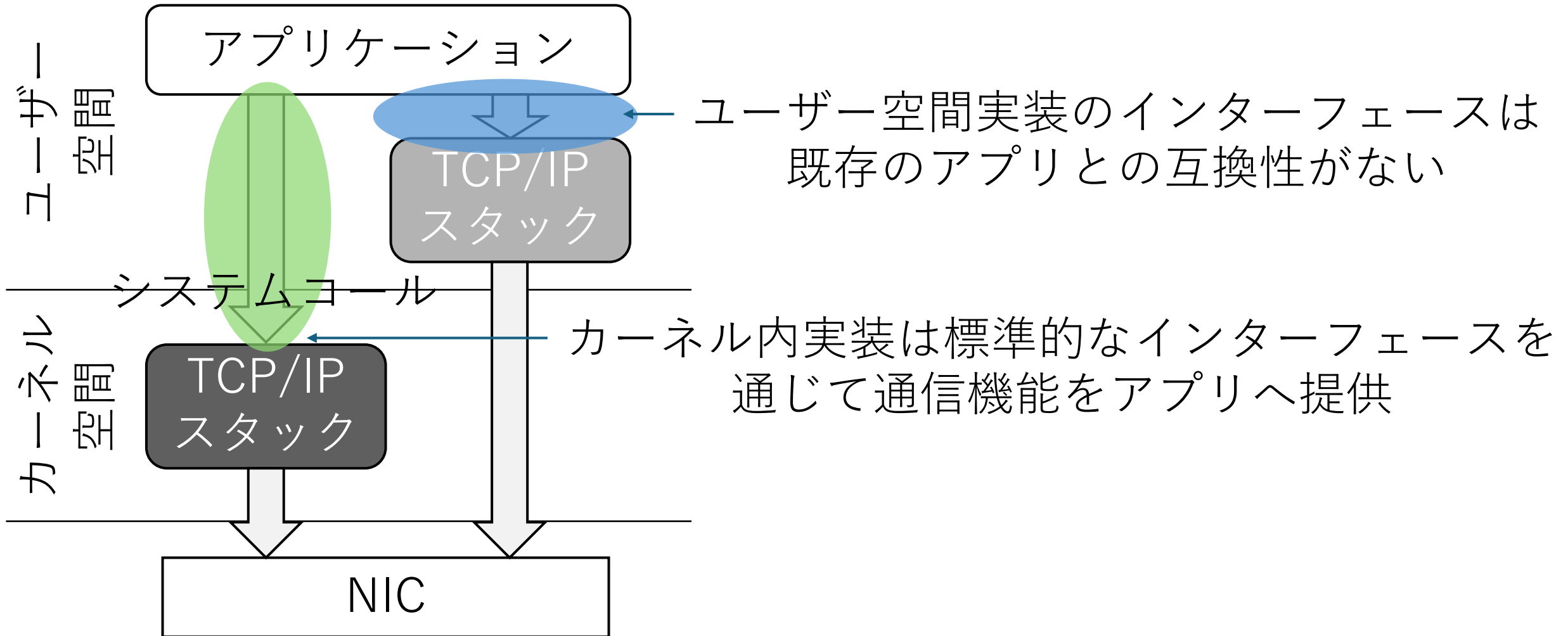
# 分析：利用者にとってのハードル



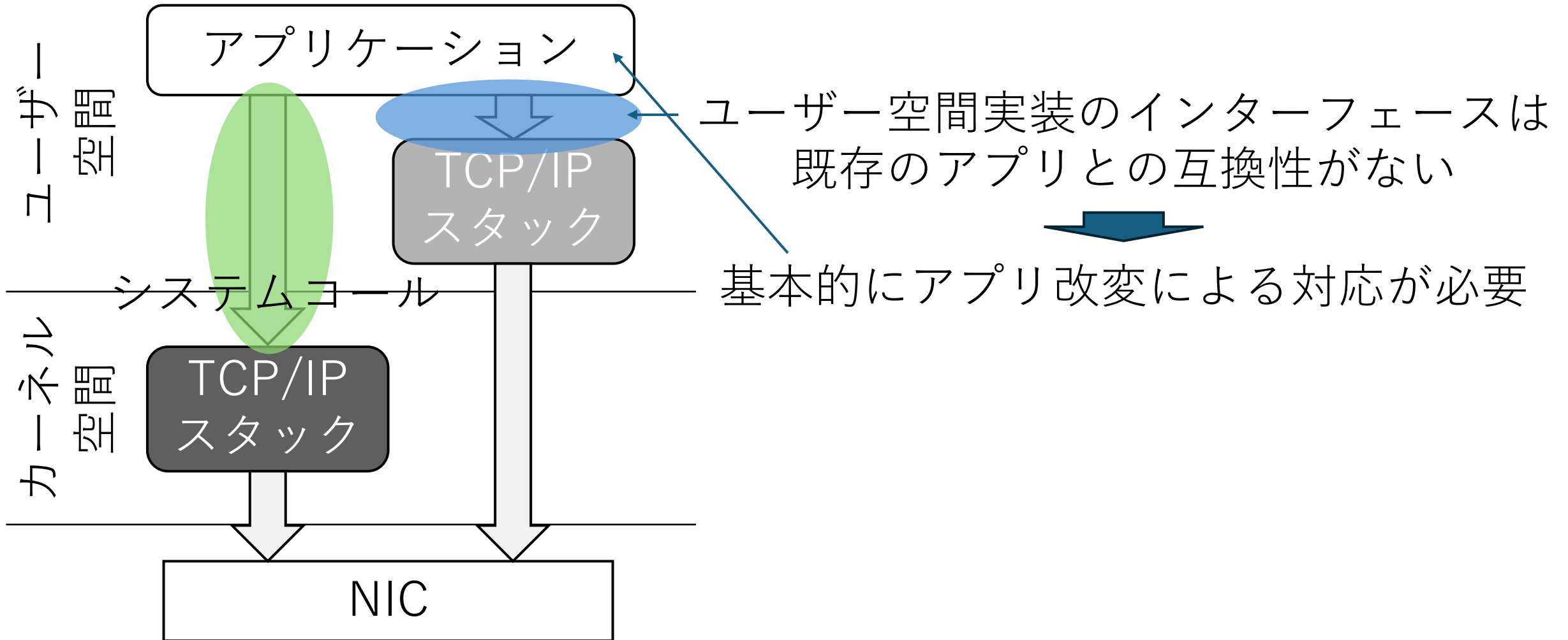
# 分析：利用者にとってのハードル



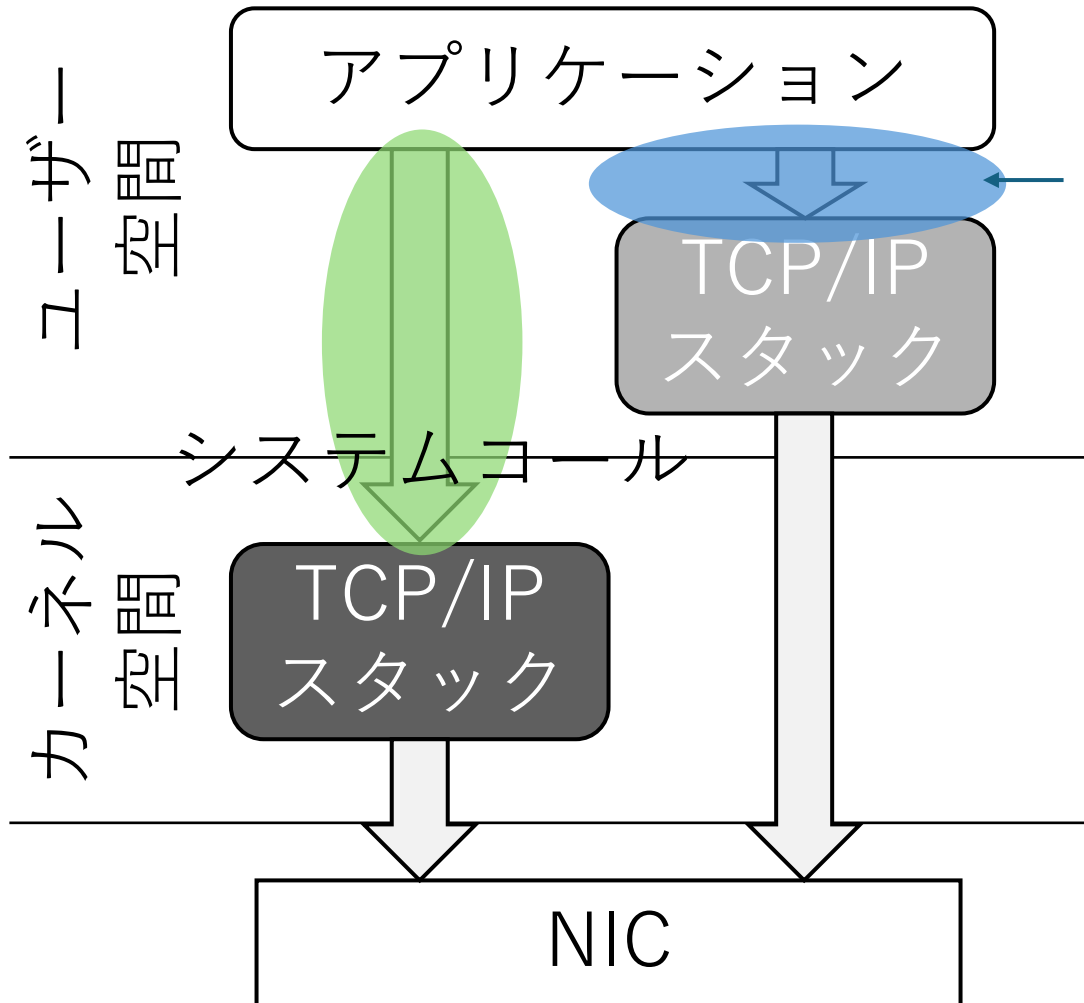
# 分析：利用者にとってのハードル



# 分析：利用者にとってのハードル



# 分析：利用者にとってのハードル



ユーザー空間実装のインターフェースは  
既存のアプリとの互換性がない

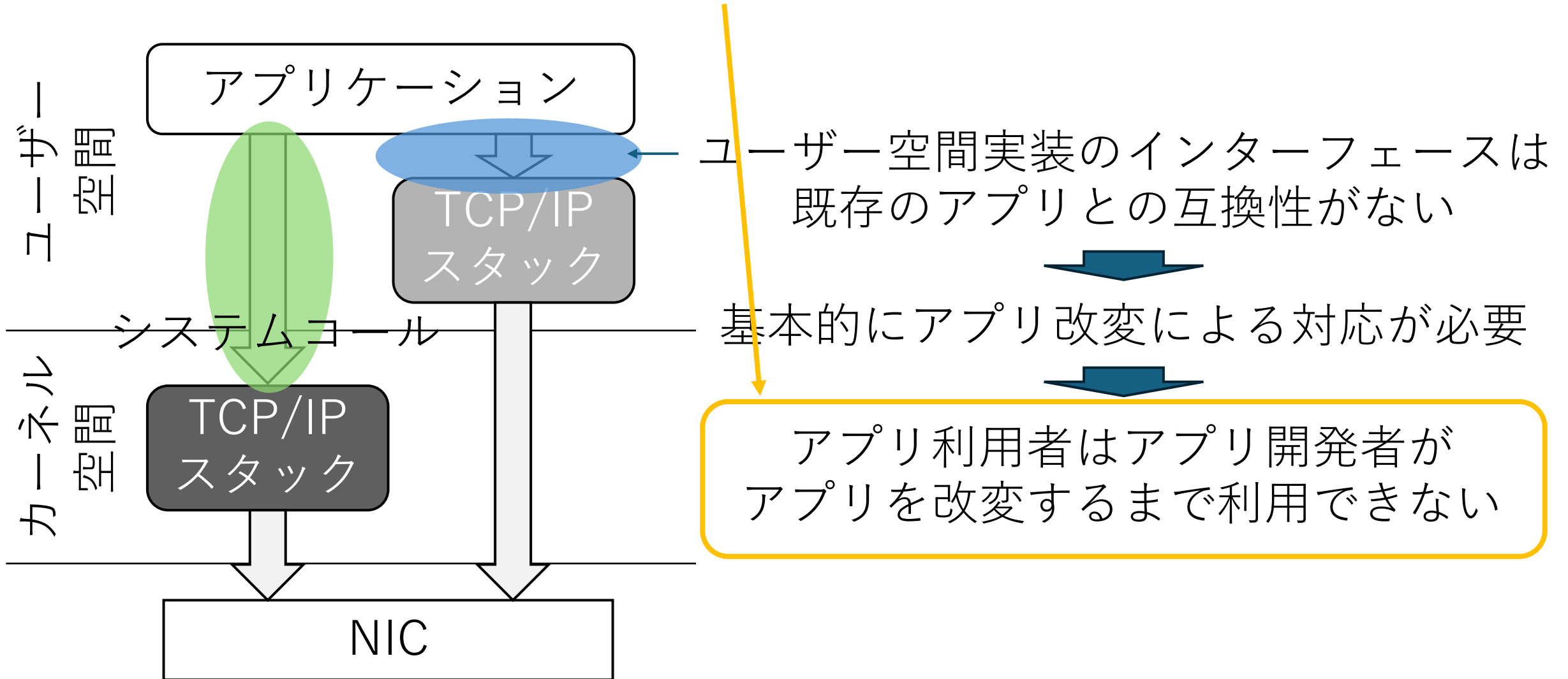


基本的にアプリ改変による対応が必要

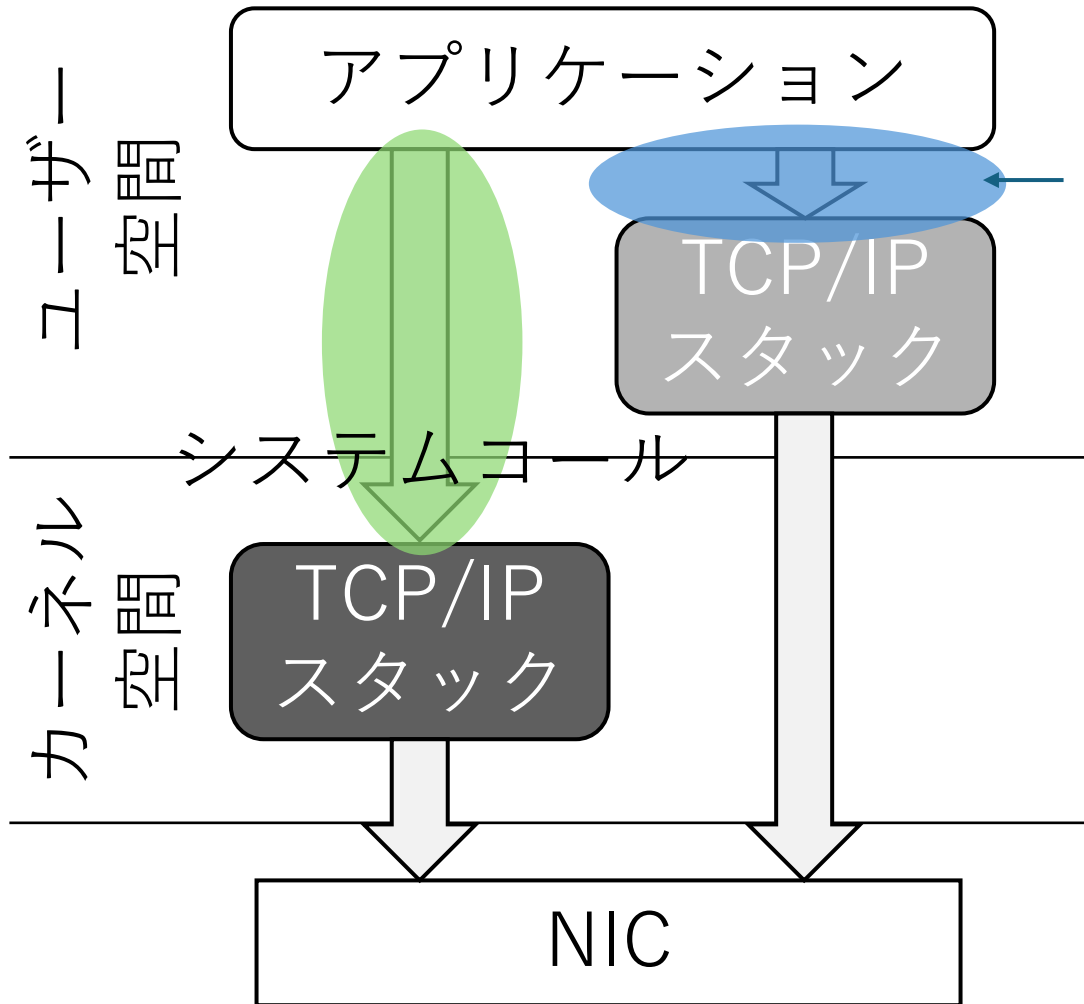


アプリ利用者はアプリ開発者が  
アプリを改変するまで利用できない

# 分析：利用者にとってのハードル



# 分析：考えられるアプローチ二通り



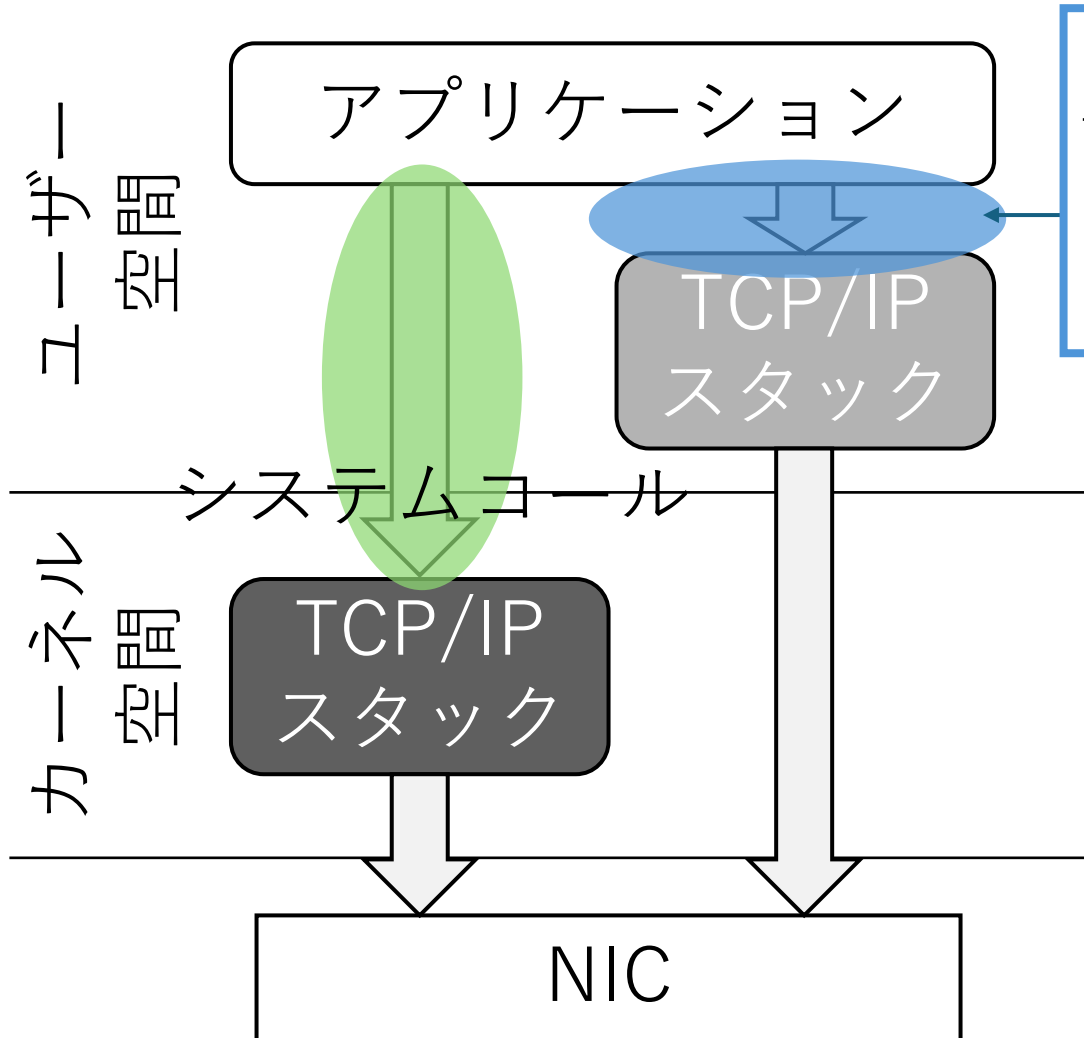
ユーザー空間実装のインターフェースは  
既存のアプリとの互換性がない

基本的にアプリ改変による対応が必要

アプリ利用者はアプリ開発者が  
アプリを改変するまで利用できない



# 分析：考えられるアプローチ二通り



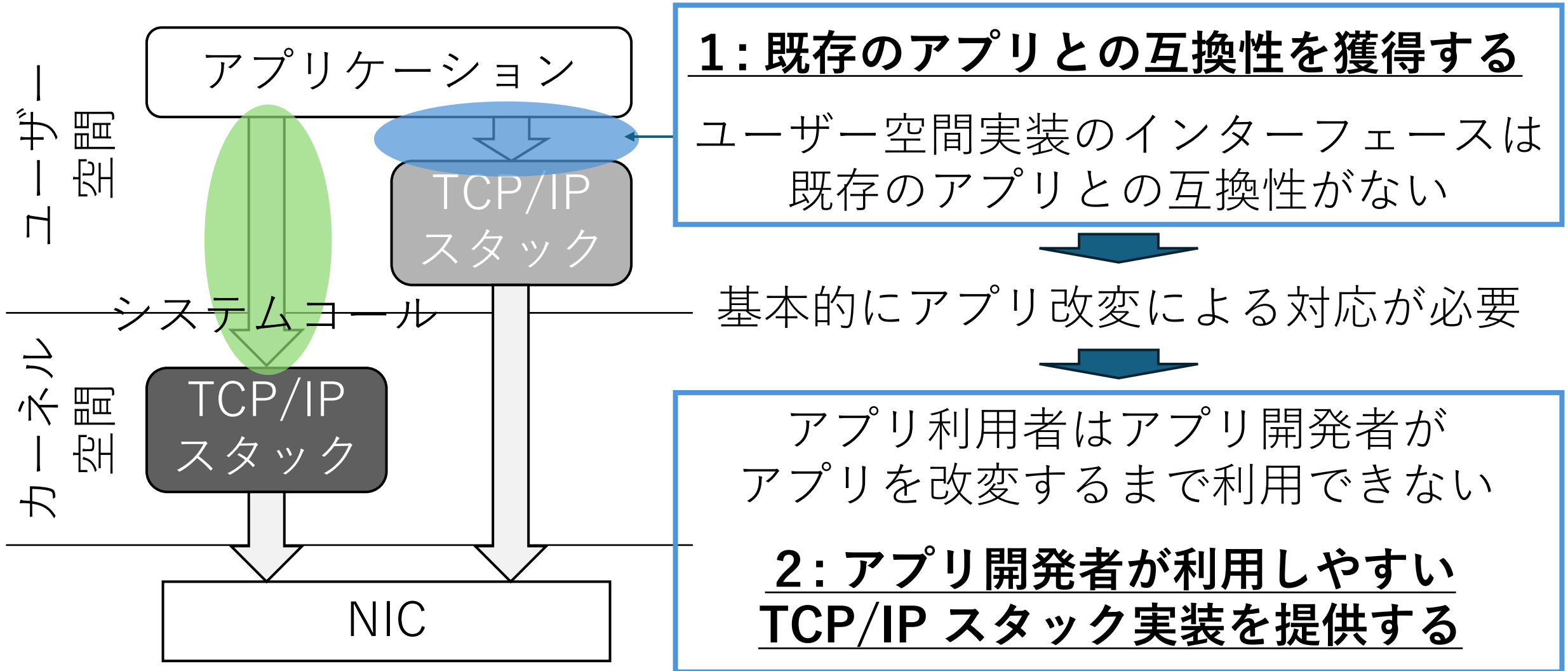
## 1: 既存のアプリとの互換性を獲得する

ユーザー空間実装のインターフェースは  
既存のアプリとの互換性がない

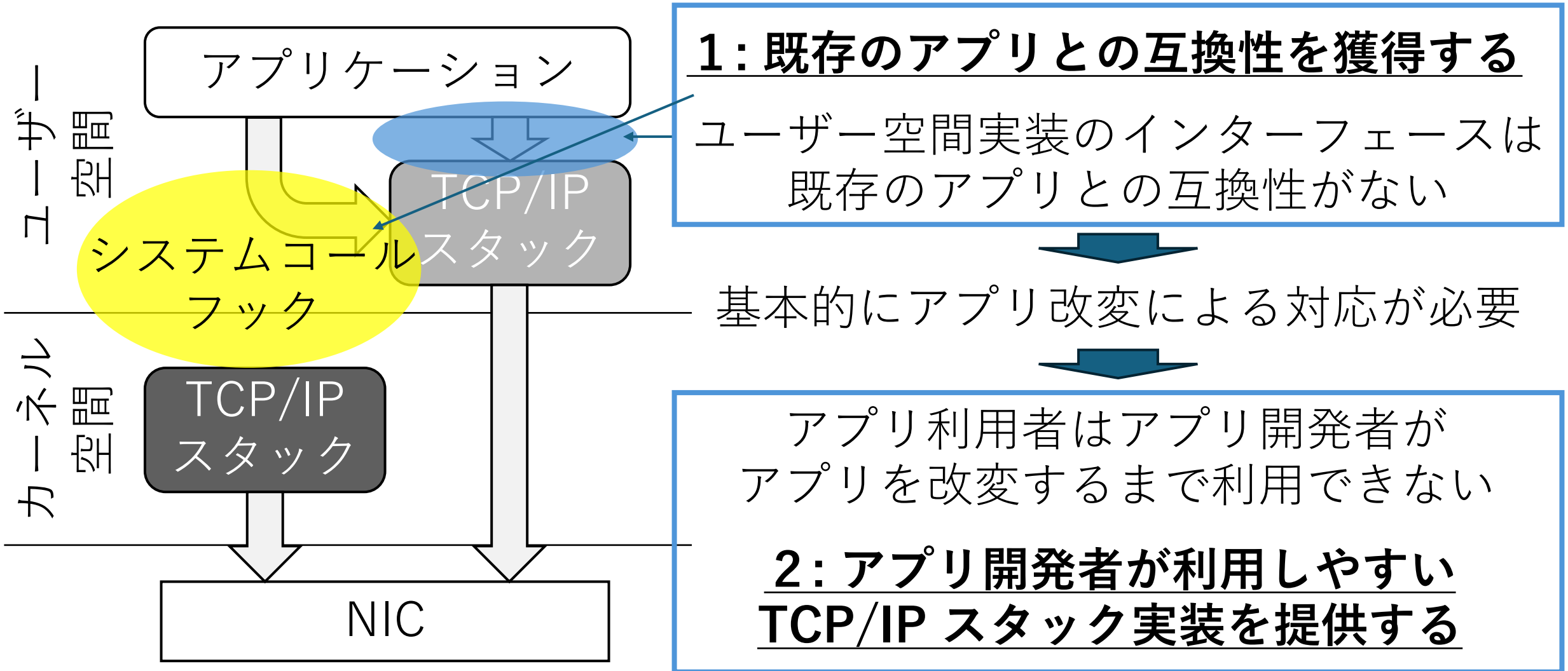
基本的にアプリ改変による対応が必要

アプリ利用者はアプリ開発者が  
アプリを改変するまで利用できない

# 分析：考えられるアプローチ二通り



# 分析：考えられるアプローチ二通り



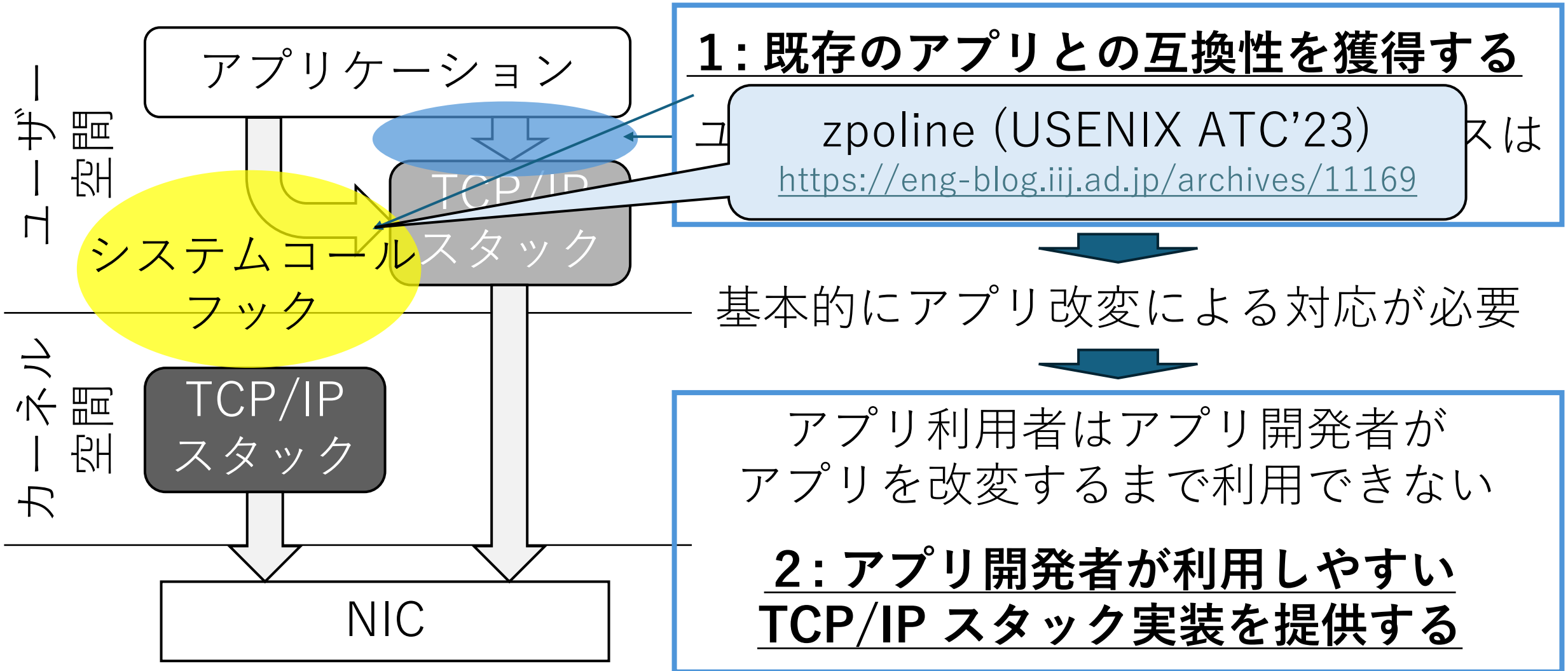
**1: 既存のアプリとの互換性を獲得する**  
ユーザー空間実装のインターフェースは既存のアプリとの互換性がない

基本的にアプリ改変による対応が必要

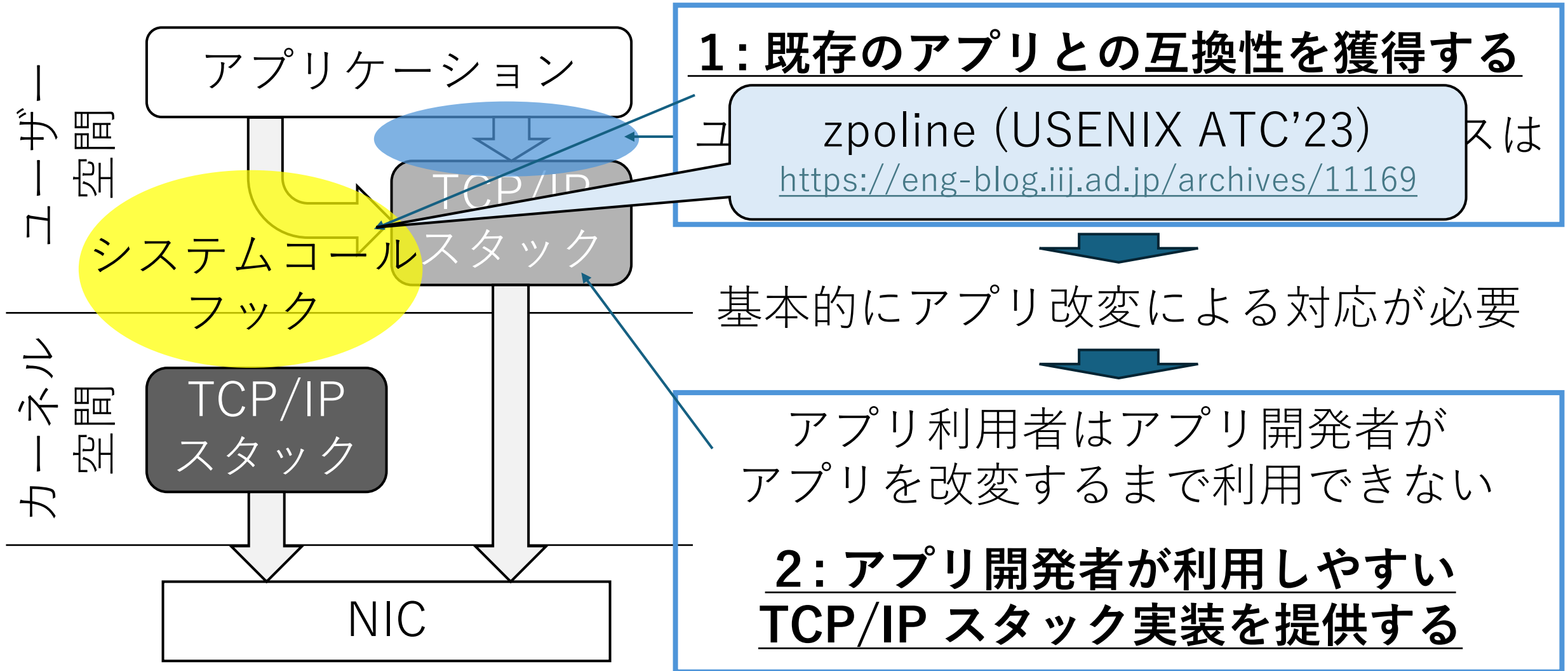
アプリ利用者はアプリ開発者がアプリを改変するまで利用できない

**2: アプリ開発者が利用しやすいTCP/IP スタック実装を提供する**

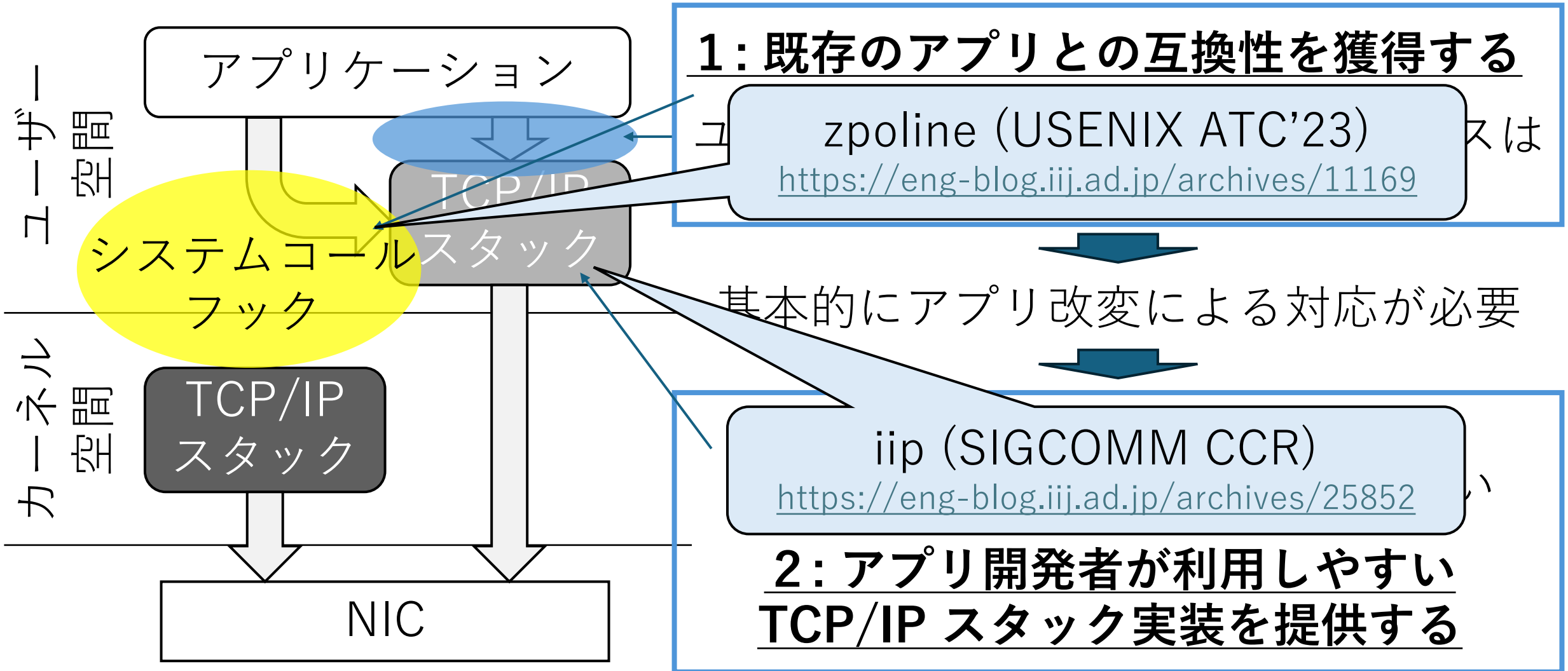
# 分析：考えられるアプローチ二通り



# 分析：考えられるアプローチ二通り



# 分析：考えられるアプローチ二通り



# 開発している TCP/IP スタック実装

- iip: an integratable TCP/IP stack
  - <https://eng-blog.iij.ad.jp/archives/25852>
  - アプリ開発者が利用しやすい TCP/IP スタック実装を目指す

開発している TCP/IP スタック実装

## 問題意識

- 既存の性能に最適化された実装は組み込みやすさに課題がある

既存実装のカテゴリ	性能	組み込みやすさ
性能への最適化	✓	



開発している TCP/IP スタック実装

## 問題意識

- 既存の性能に最適化された実装は組み込みやすさに課題がある
- 既存の可搬性に配慮した実装は性能に課題がある

既存実装のカテゴリ	性能	組み込みやすさ
性能への最適化	✓	
可搬性への配慮		✓

開発している TCP/IP スタック実装

## 問題意識

- 性能と可搬性を両立した実装がなく、高性能な TCP/IP 実装を利用したい開発者には限られた非効率な選択肢しかなかった

既存実装のカテゴリ	性能	組み込みやすさ
性能への最適化	✓	
可搬性への配慮		✓

開発している TCP/IP スタック実装

## 問題意識

- 性能と可搬性を両立した実装がなく、高性能な TCP/IP 実装を利用したい開発者には限られた非効率な選択肢しかなかった
  - 既存の TCP/IP スタック実装を大きく改変して性能を改善する

既存実装のカテゴリ	性能	組み込みやすさ
性能への最適化	✓	
可搬性への配慮		✓

開発している TCP/IP スタック実装

## 問題意識

- 性能と可搬性を両立した実装がなく、高性能な TCP/IP 実装を利用したい開発者には限られた非効率な選択肢しかなかった
  - 既存の TCP/IP スタック実装を大きく改変して性能を改善する
  - 新しく TCP/IP スタックを実装する

既存実装のカテゴリ	性能	組み込みやすさ
性能への最適化	✓	
可搬性への配慮		✓

開発している TCP/IP スタック実装

## 問題意識

- 性能と可搬性を両立した実装がなく、高性能な TCP/IP 実装を利用したい開発者には限られた非効率な選択肢しかなかった
  - 既存の TCP/IP スタック実装を大きく改変して性能を改善する
  - 新しく TCP/IP スタックを実装する
  - 既存の TCP/IP スタック実装の性能についての制限を許容する

既存実装のカテゴリ	性能	組み込みやすさ
性能への最適化	✓	
可搬性への配慮		✓

開発している TCP/IP スタック実装

## 問題意識

- 性能と可搬性を両立した実装がなく、高性能な TCP/IP 実装を利用したい開発者には限られた非効率な選択肢しかなかった
  - 既存の TCP/IP スタック実装を大きく改変して性能を改善する
  - 新しく TCP/IP スタックを実装する
  - 既存の TCP/IP スタック実装の性能についての制限を許容する
  - 高速な TCP/IP スタック実装の利用を諦める

既存実装のカテゴリ	性能	組み込みやすさ
性能への最適化	✓	
可搬性への配慮		✓

開発している TCP/IP スタック実装

## 提案実装

- 性能と可搬性を両立した TCP/IP スタック実装を提供する

既存実装のカテゴリ	性能	組み込みやすさ
性能への最適化	✓	
可搬性への配慮		✓
提案実装	✓	✓

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

- 性能に最適化された実装
  - 外部要素への依存
  - 機能の干渉
  - CPU コア割り当てパターンの制限
- 可搬性に配慮した実装
  - NIC のオフロード機能への配慮が不足
  - ゼロコピー送受信をサポートしていない
  - マルチコア環境で性能がスケールしない



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム

複数の様々な構成要素

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

外部要素への依存

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

外部要素への依存

性能に最適化された  
TCP/IP スタック

I/O 機構がライブラリ-A に  
依存しているとする

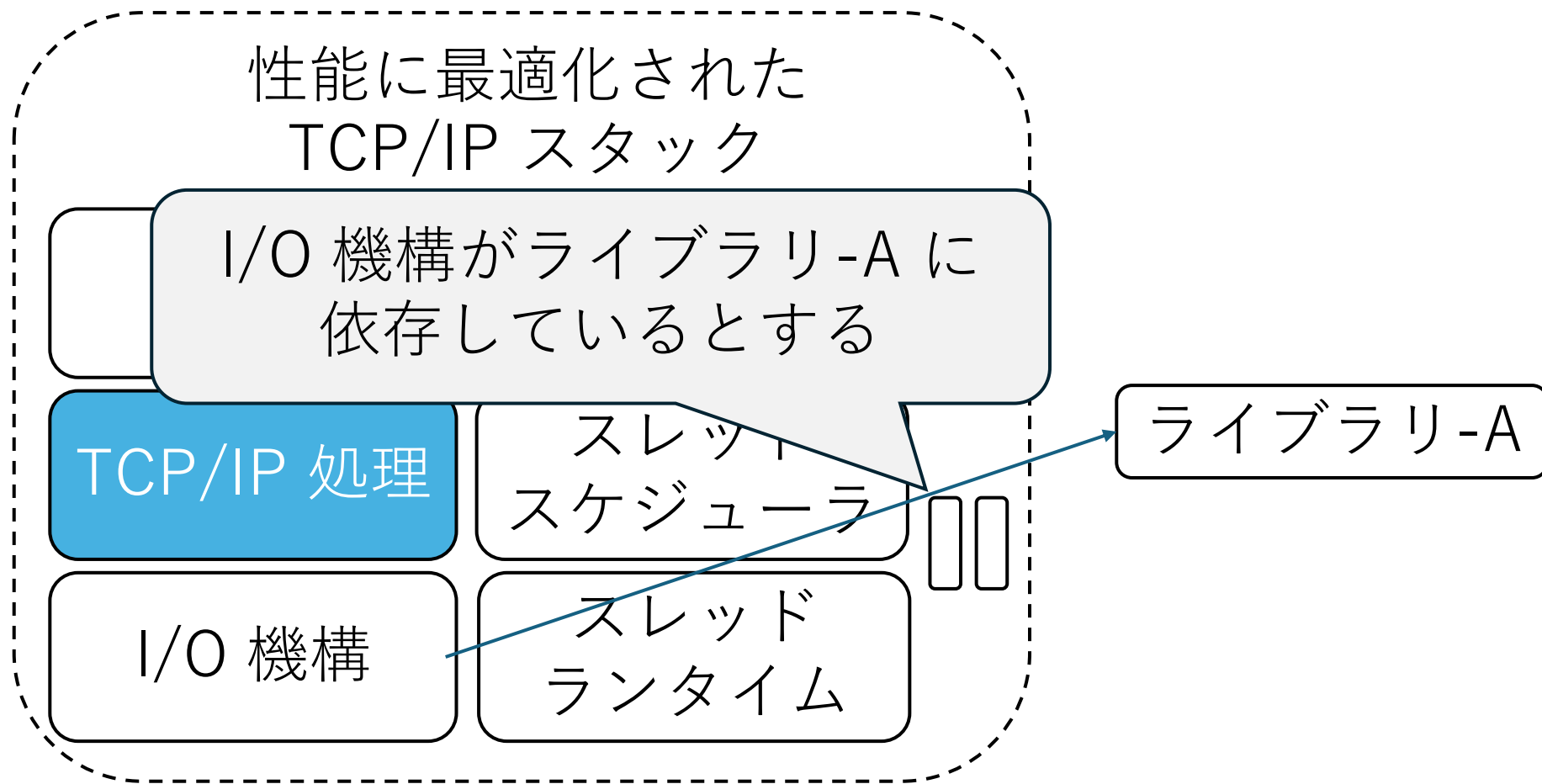
TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム

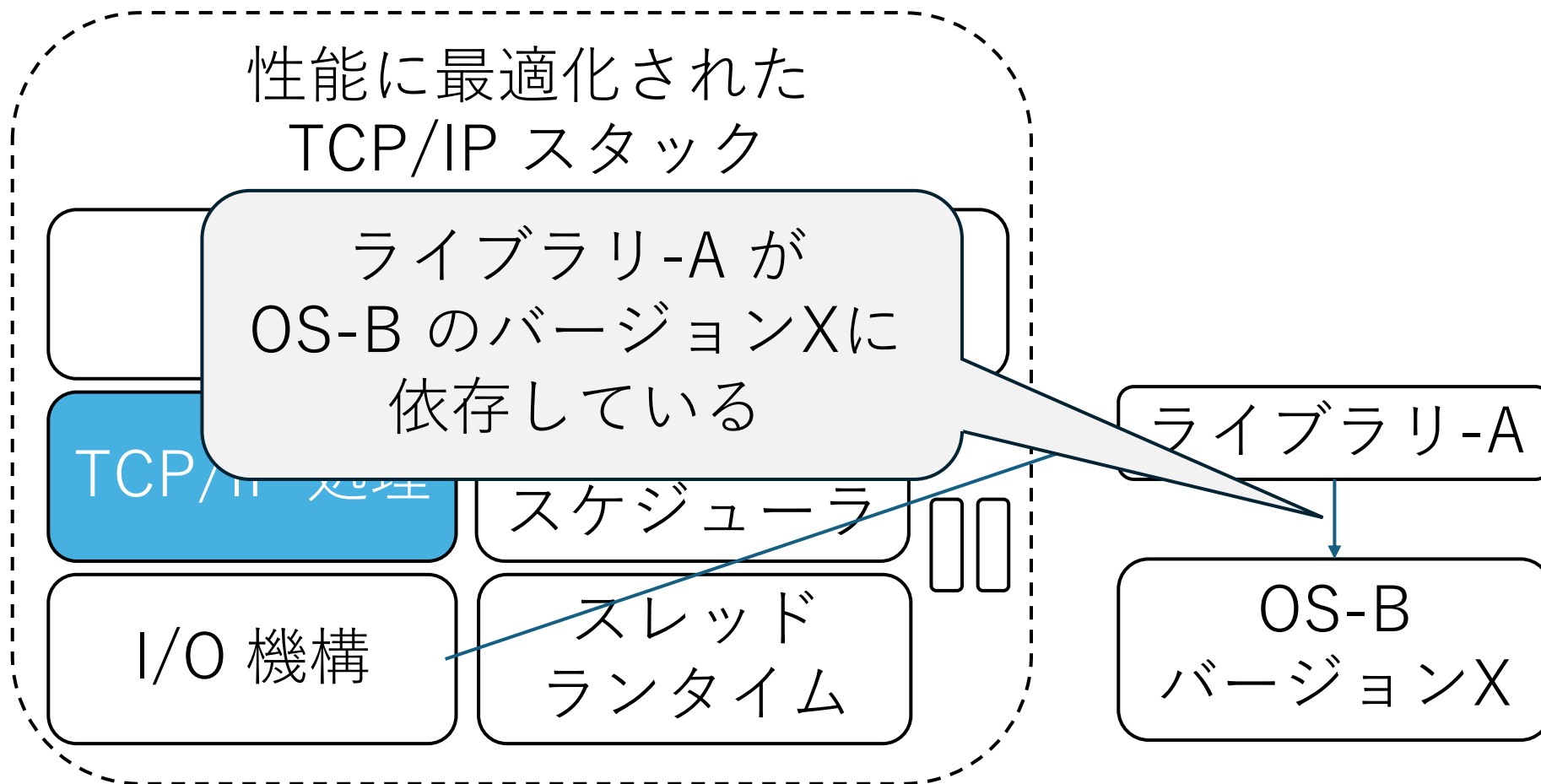
ライブラリ-A



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

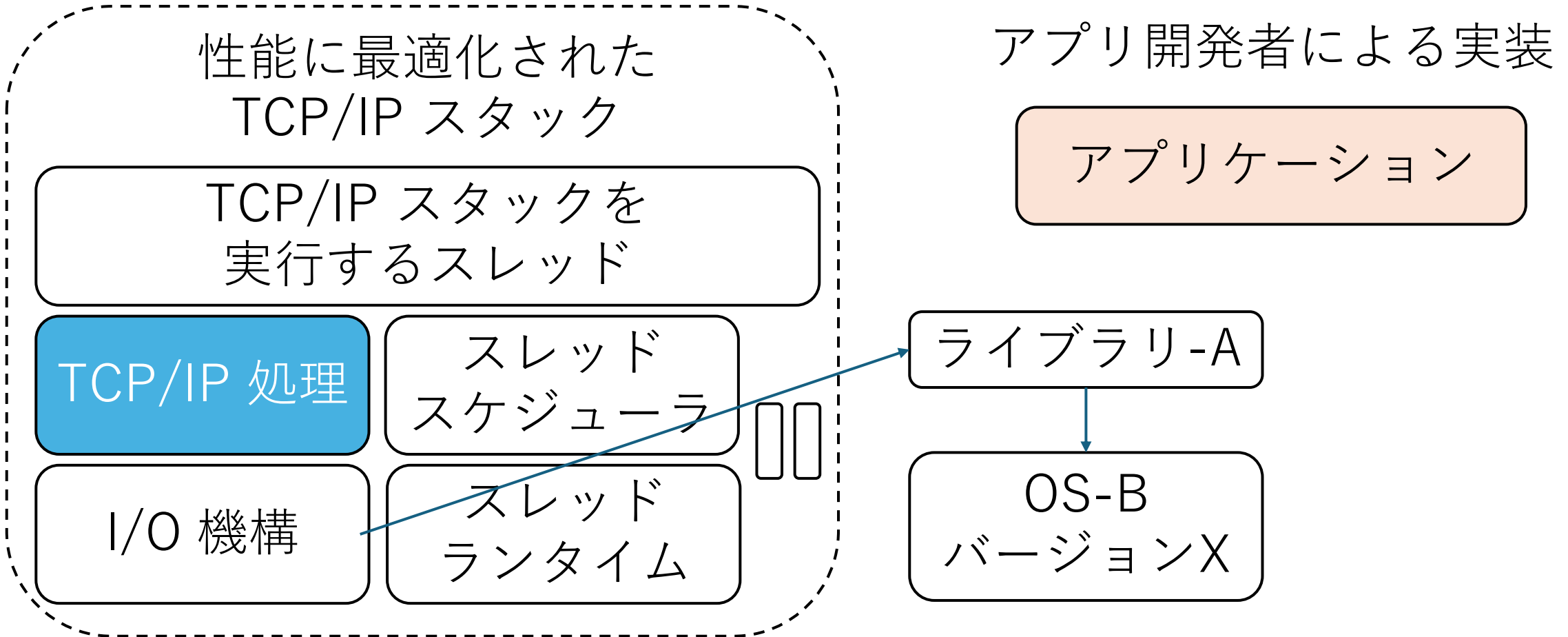
外部要素への依存



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

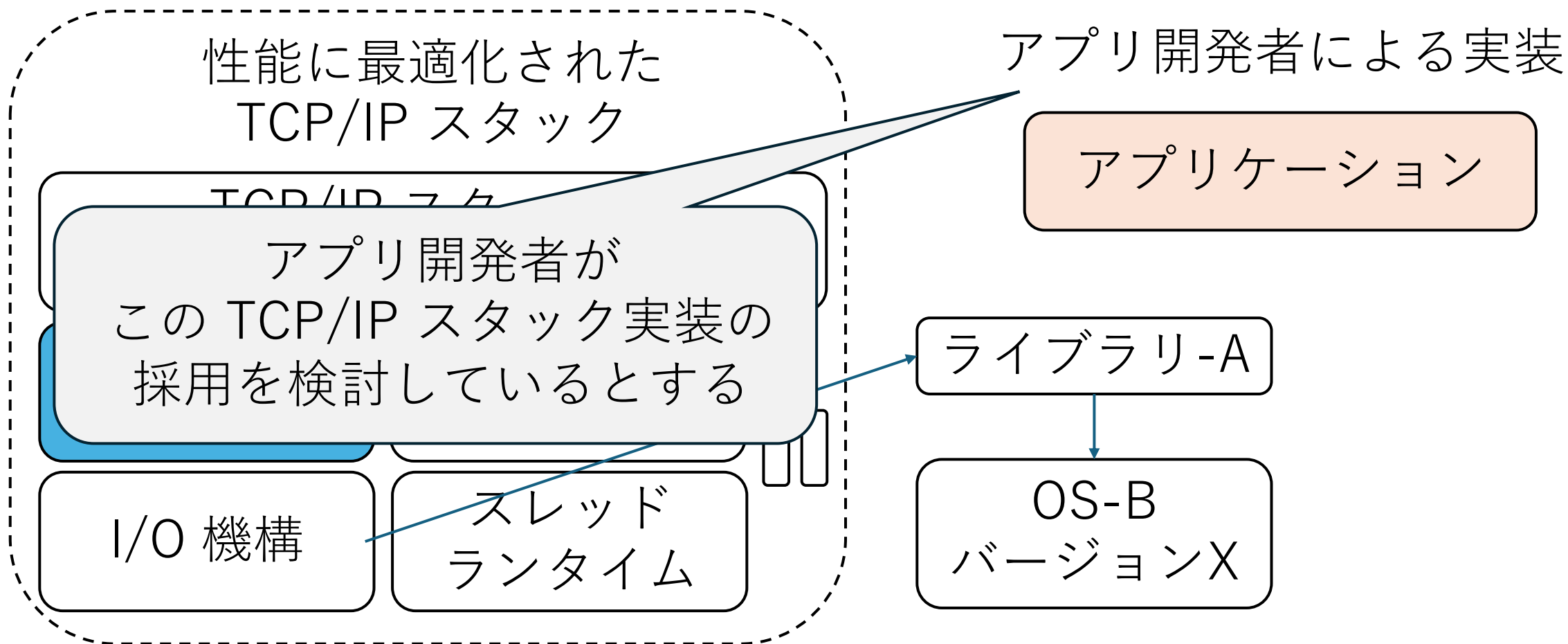
外部要素への依存



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

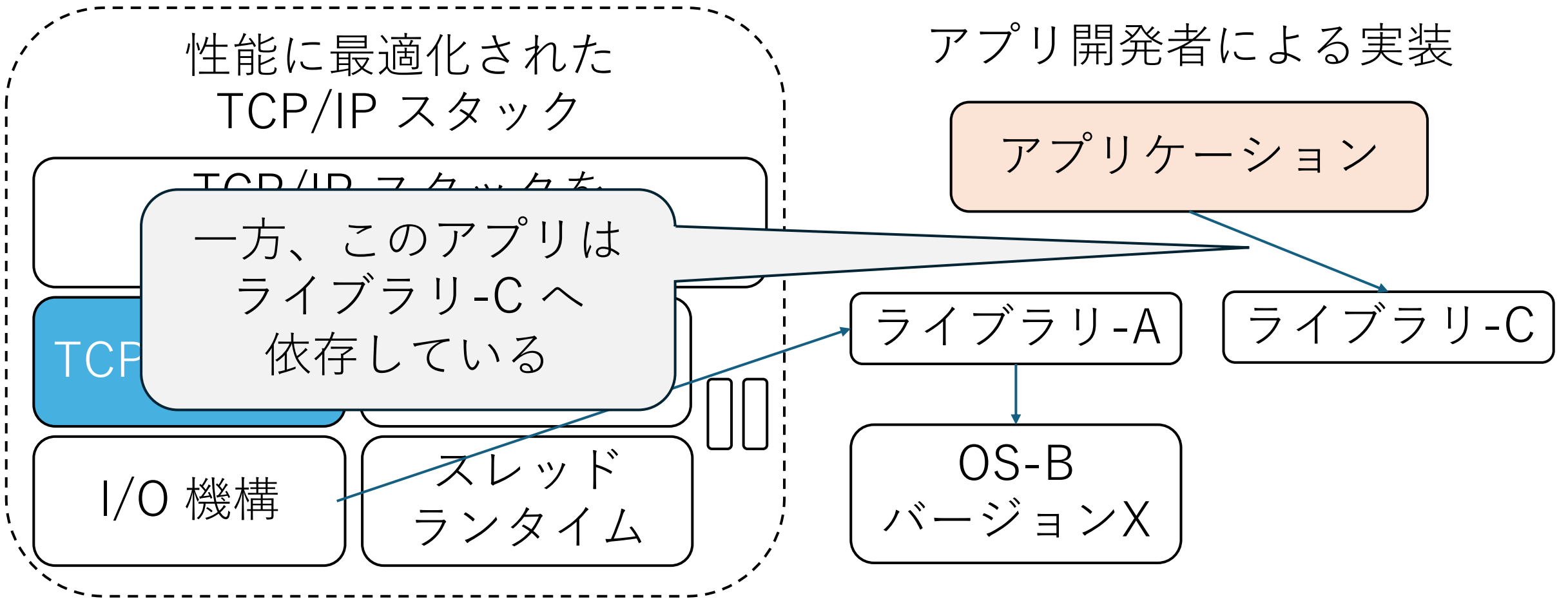
外部要素への依存



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

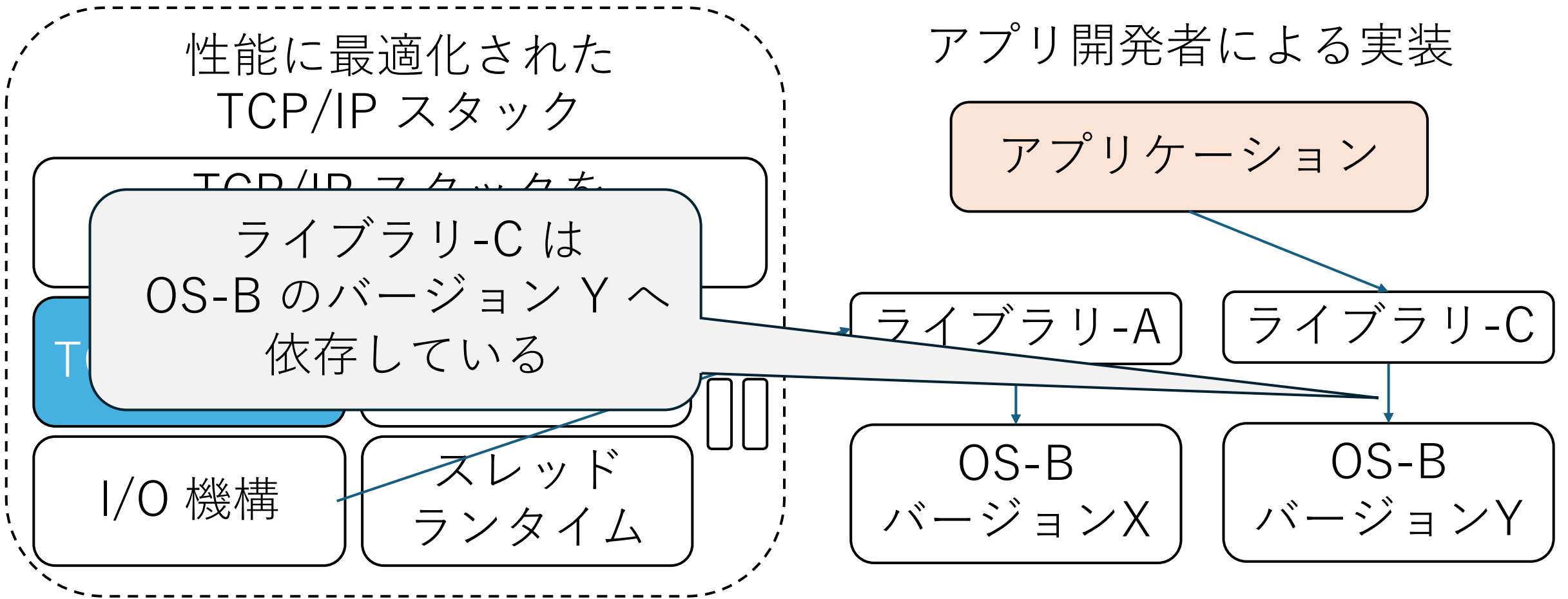
外部要素への依存



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

外部要素への依存

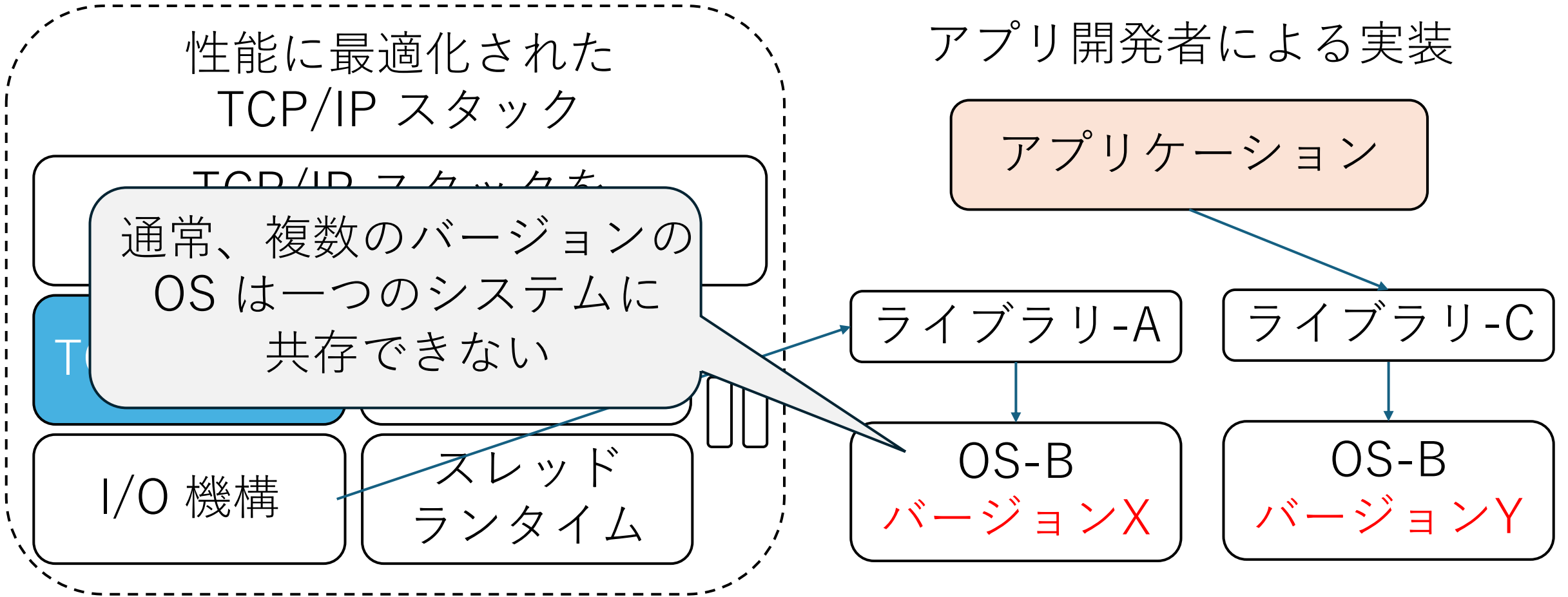




開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

外部要素への依存



性能に最適化された  
TCP/IP スタック

TCP/IP スタック実装

通常、複数のバージョンの  
OS は一つのシステムに  
共存できない

T

I/O 機構

スレッド  
ランタイム

アプリ開発者による実装

アプリケーション

ライブラリ-A

ライブラリ-C

OS-B

バージョン X

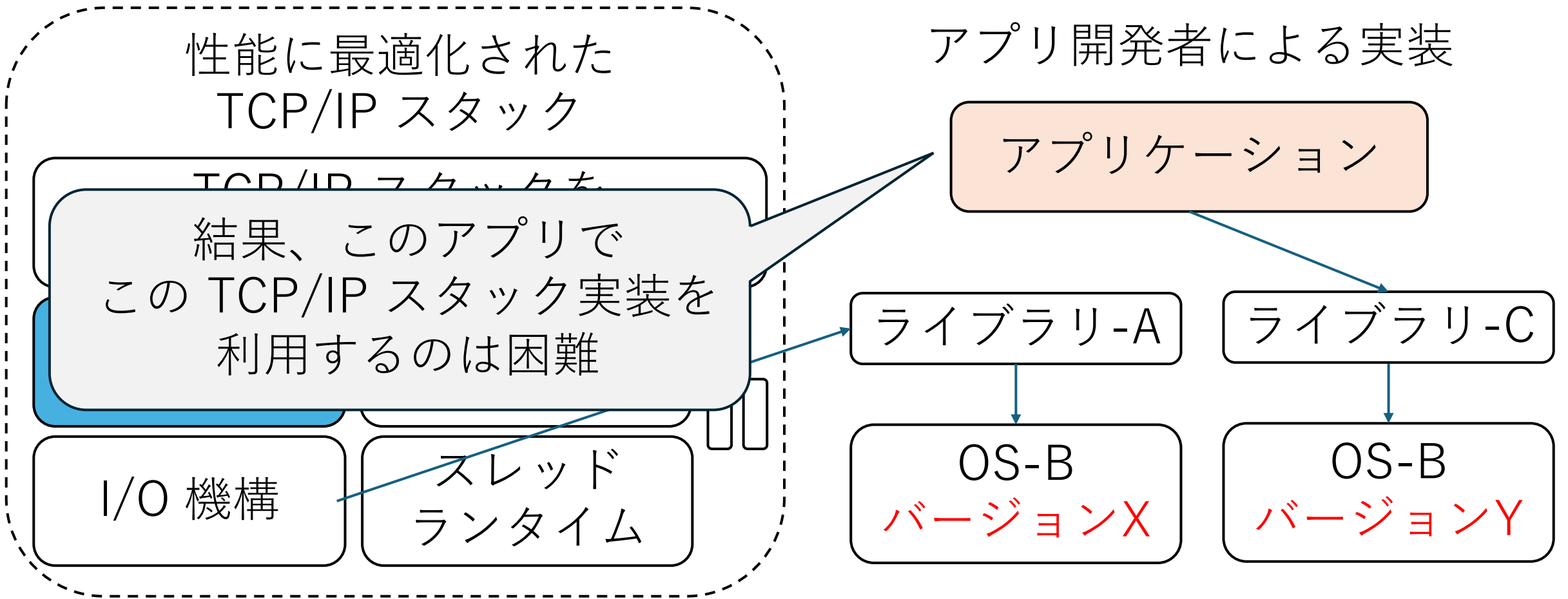
OS-B

バージョン Y

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

外部要素への依存



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## 外部要素への依存

外部要素への依存は  
TCP/IP スタック実装の  
採用の複雑性を高める

TCP/IP 処理

I/O 機構

スレッド  
スケジューラ

スレッド  
ランタイム

アプリ開発者による実装

アプリケーション

ライブラリ-A

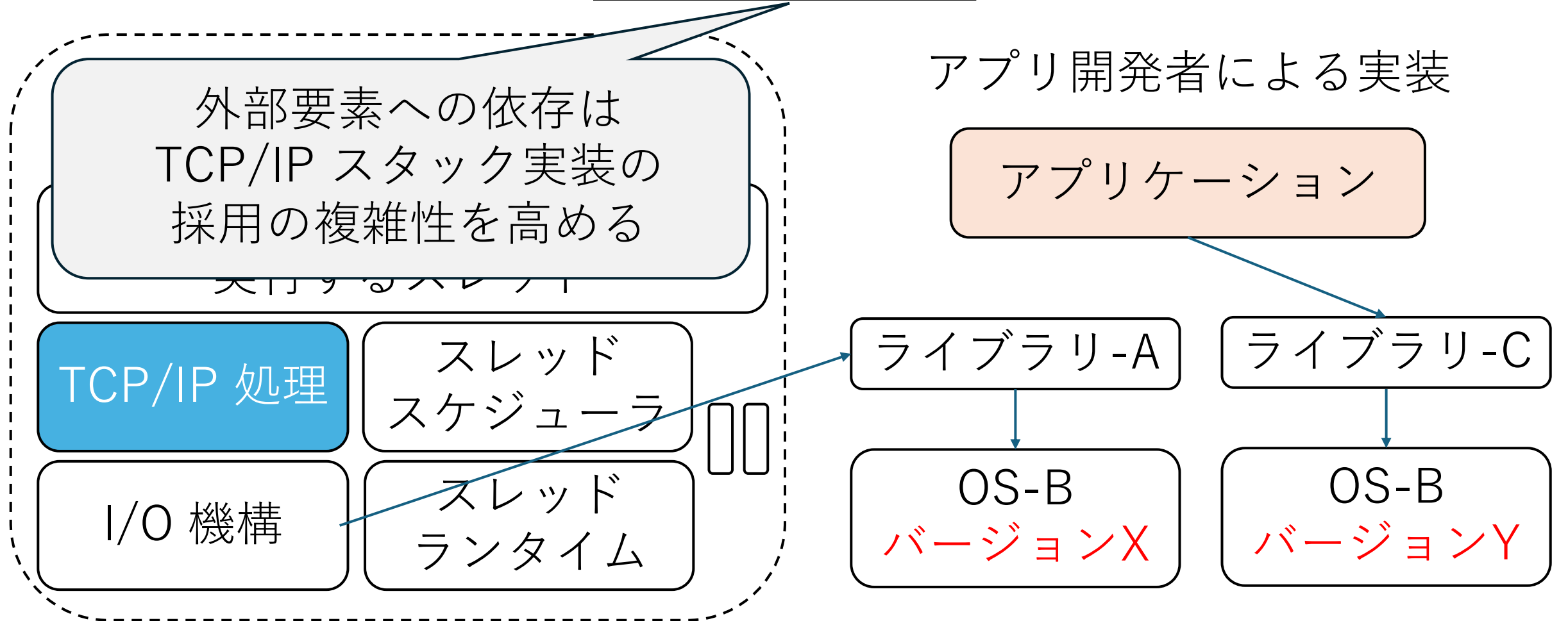
ライブラリ-C

OS-B

バージョンX

OS-B

バージョンY



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## 機能の干渉

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## 機能の干渉

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム

新しく設計された OS

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

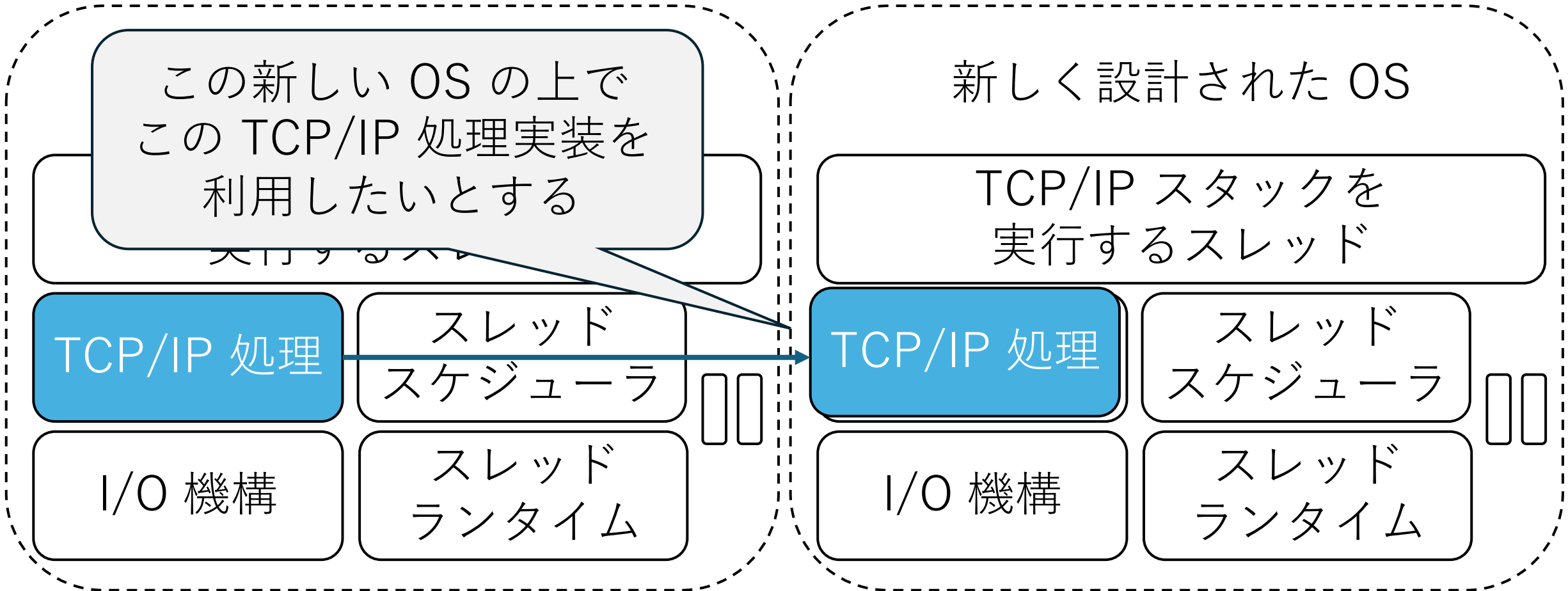
I/O 機構

スレッド  
ランタイム

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

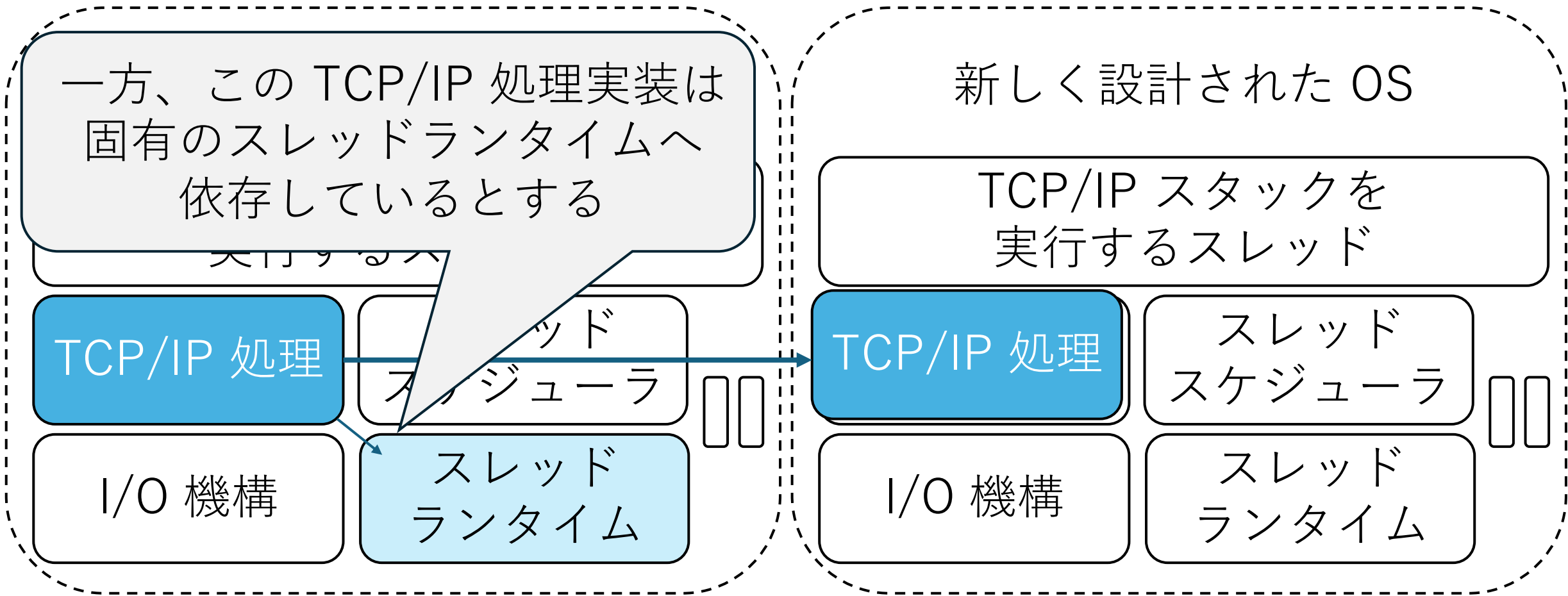
## 機能の干渉



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

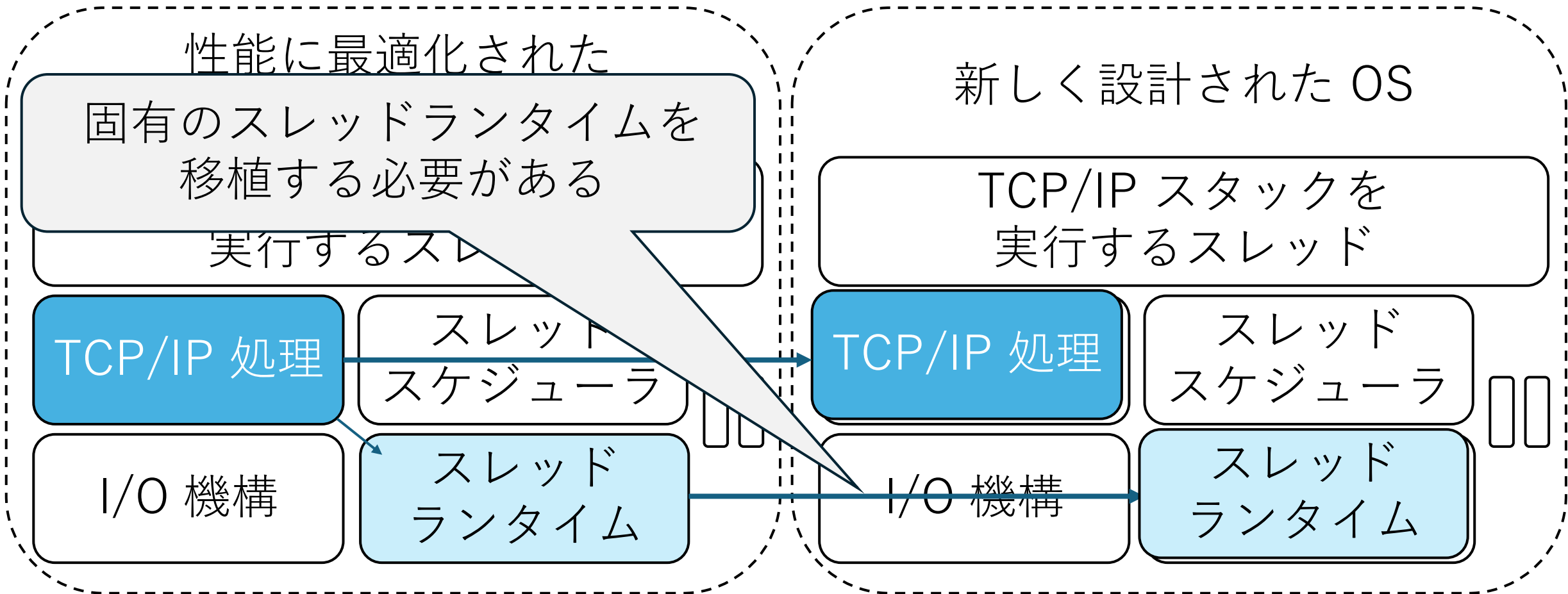
## 機能の干渉



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## 機能の干渉

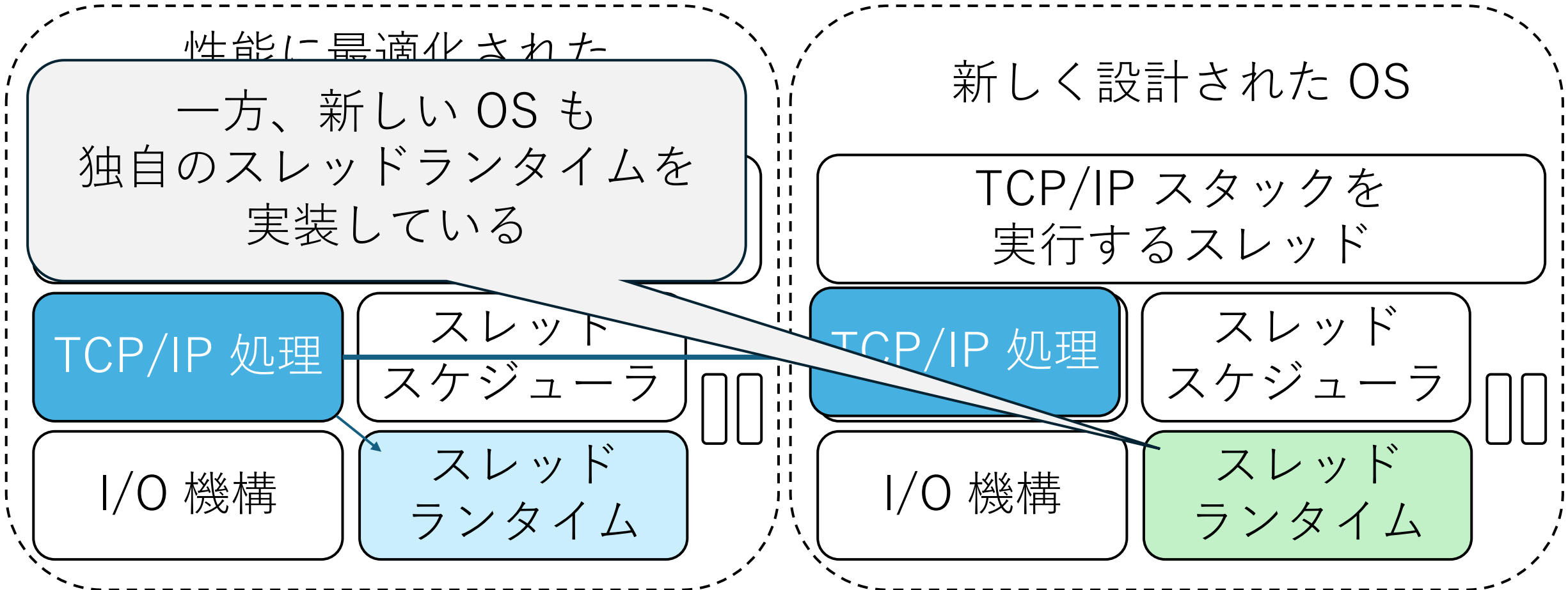




開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## 機能の干渉



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## 機能の干渉

性能に最適化された

二つのスレッドランタイムは  
一つのシステムの上で通常  
共存できない

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム

新しく設計された OS

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

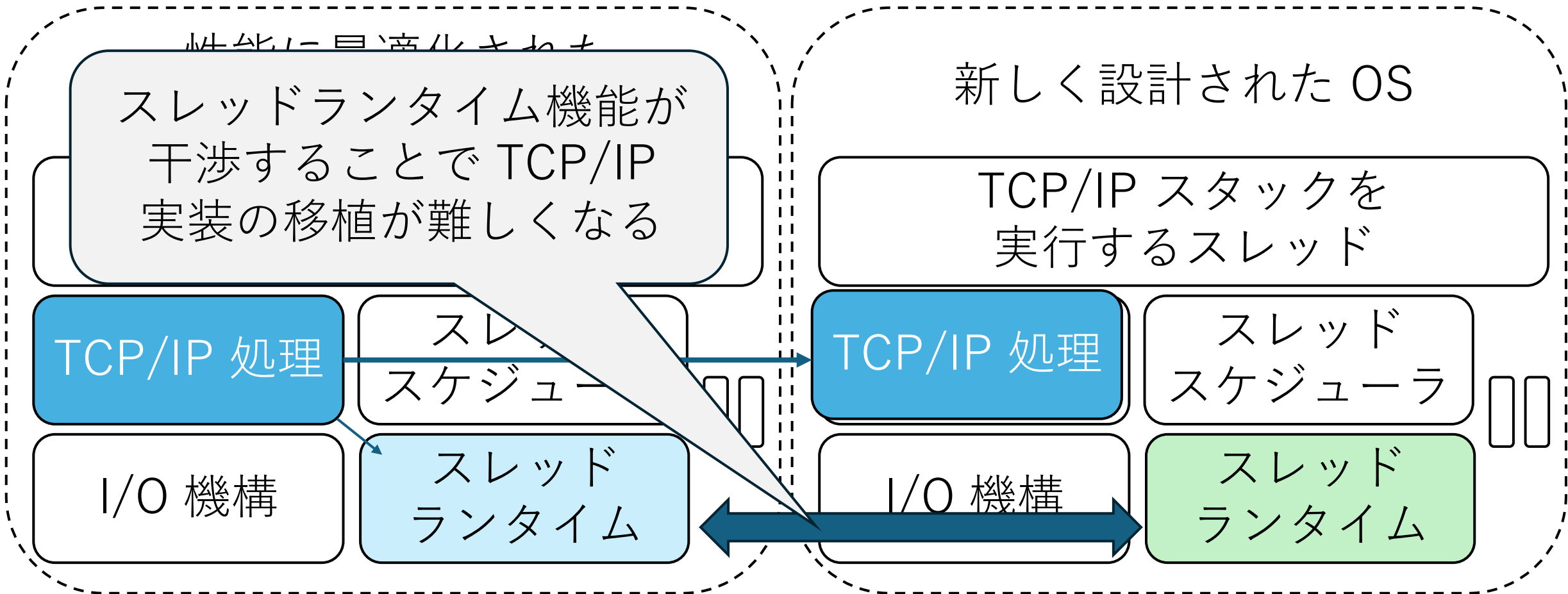
スレッド  
ランタイム

スレッド  
ランタイム

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

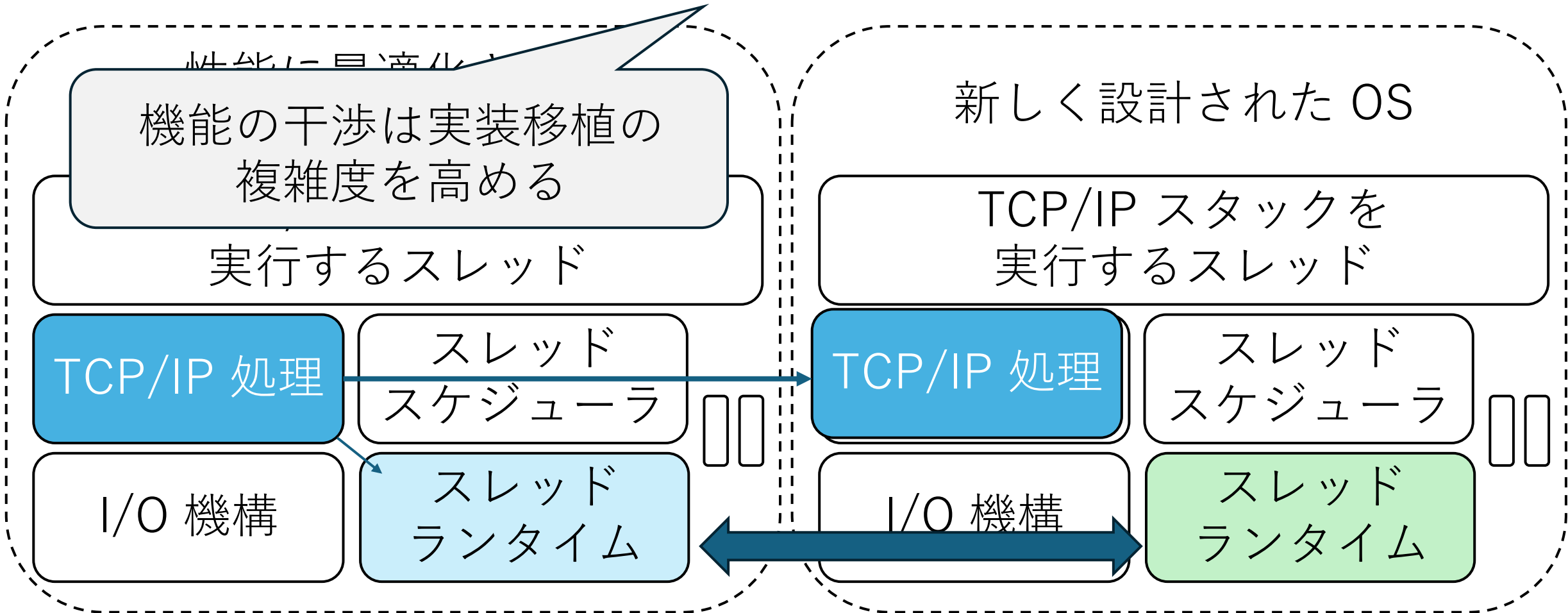
## 機能の干渉



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## 機能の干渉



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

CPU コア割り当てパターンの制限

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

CPU コア割り当てパターンの制限

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

多くの TCP/IP スタック実装は  
TCP/IP 処理を実行するスレッドを  
実装している

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジュー

多くの TCP/IP スタック実装は  
アプリは別のスレッドで  
実行されることを想定

アプリ開発者による実装

アプリケーション

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

考えられる三通りの CPU コア割り当てパターン



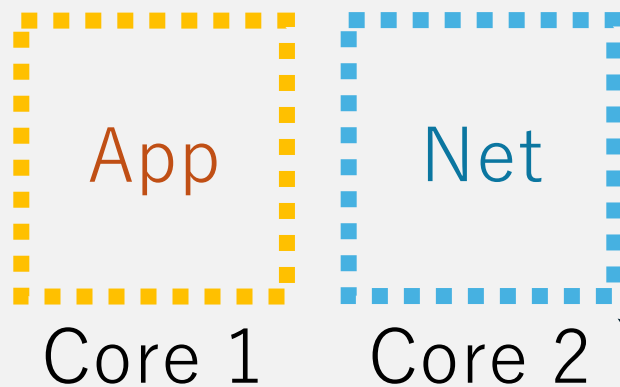
開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

考えられる三通りの CPU コア割り当てパターン

Split



アプリと通信処理を別のスレッドで実行し  
それぞれ別の CPU コアで実行

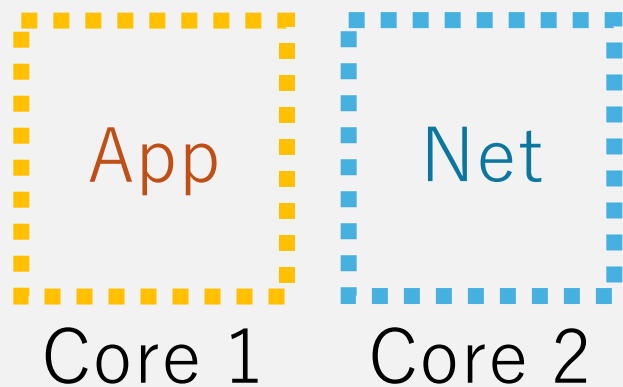
開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

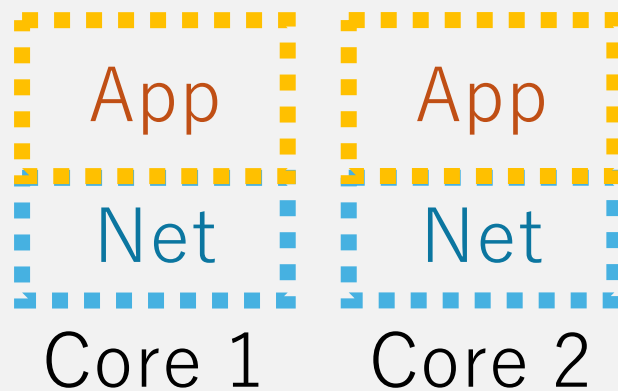
## CPU コア割り当てパターンの制限

考えられる三通りの CPU コア割り当てパターン

Split



Merge



アプリと通信処理を別のスレッドで実行し  
それぞれを同じ CPU コアで実行

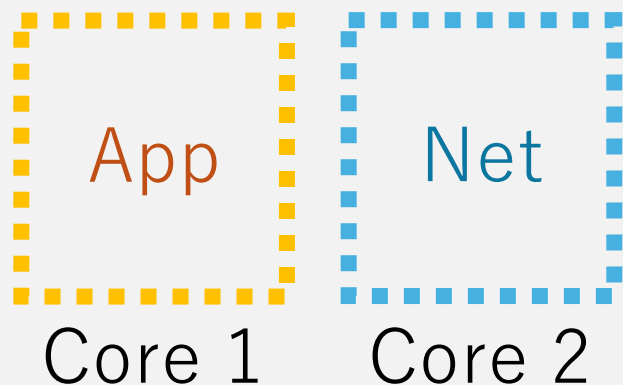
開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

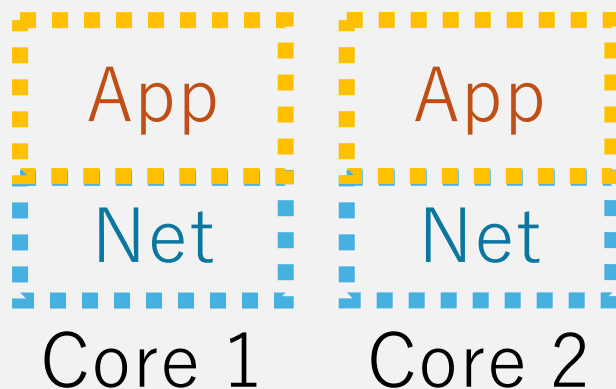
## CPU コア割り当てパターンの制限

考えられる三通りの CPU コア割り当てパターン

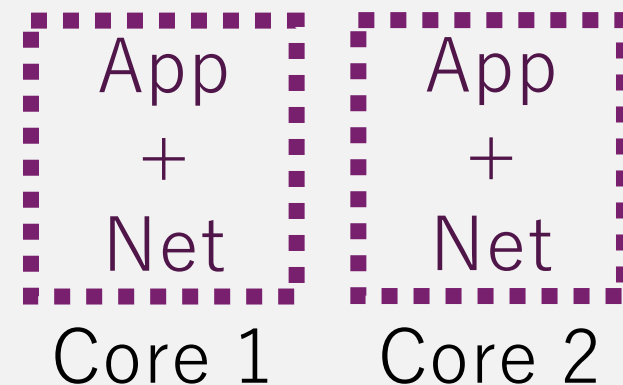
Split



Merge



Unified



アプリと通信処理を同じスレッドで実行

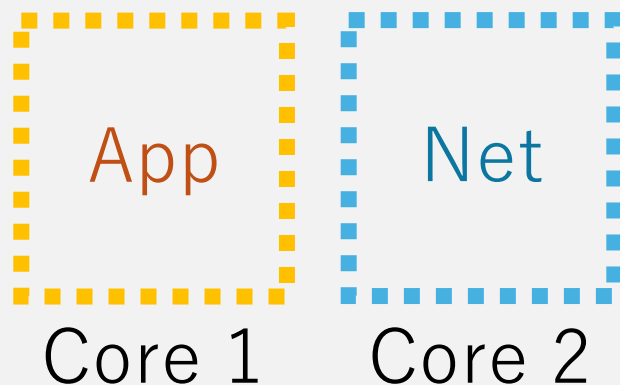
開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

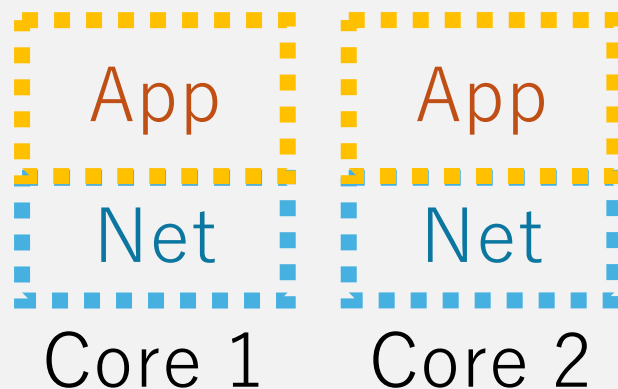
## CPU コア割り当てパターンの制限

考えられる三通りの CPU コア割り当てパターン

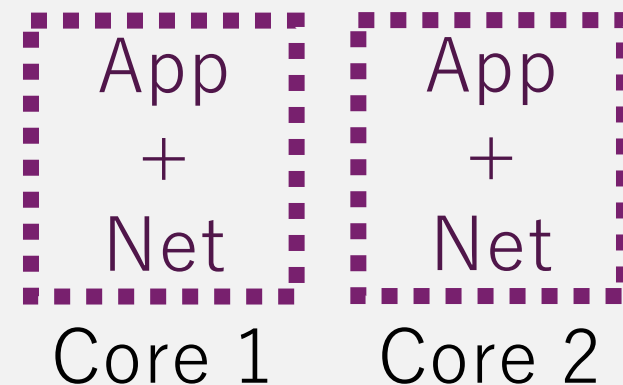
Split



Merge



Unified



それぞれ異なる性能特性がある

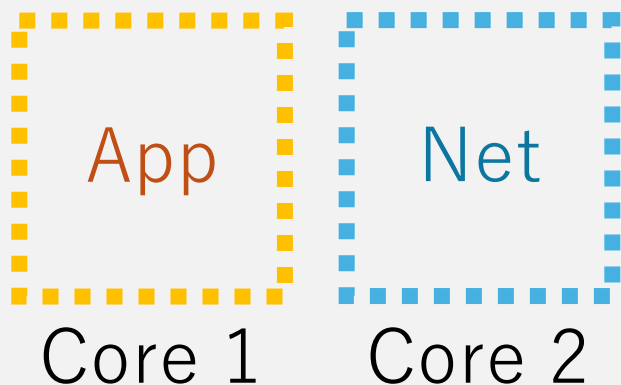
開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

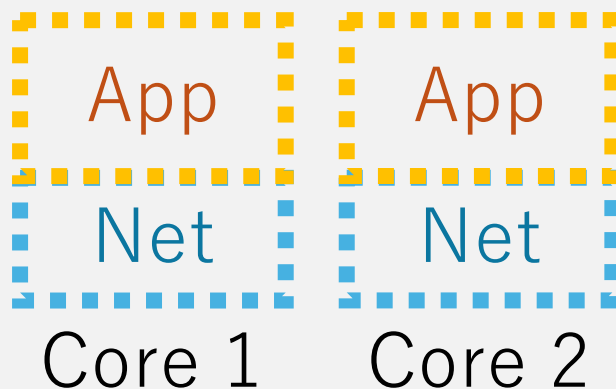
## CPU コア割り当てパターンの制限

考えられる三通りの CPU コア割り当てパターン

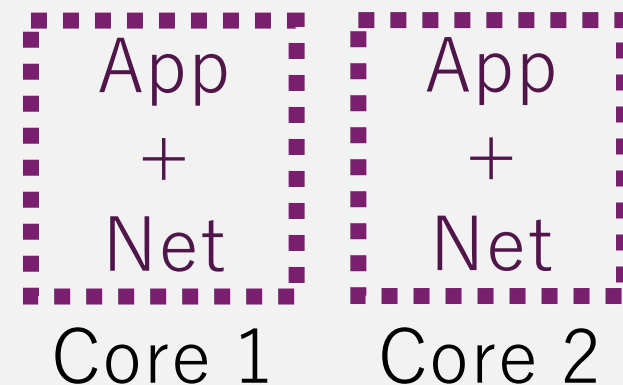
Split



Merge



Unified



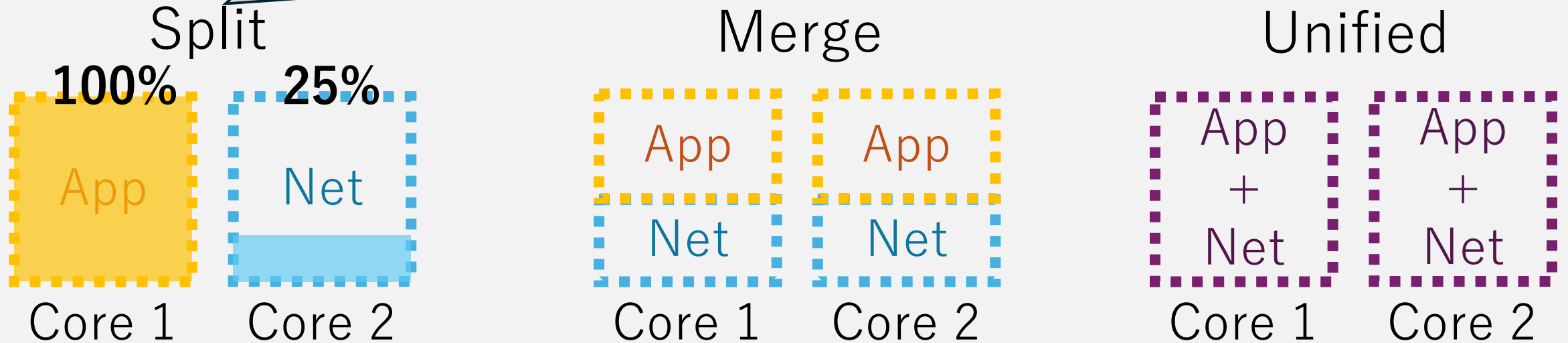
CPU 利用率

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

忙しいスレッドが、利用されていない CPU リソースを利用できない



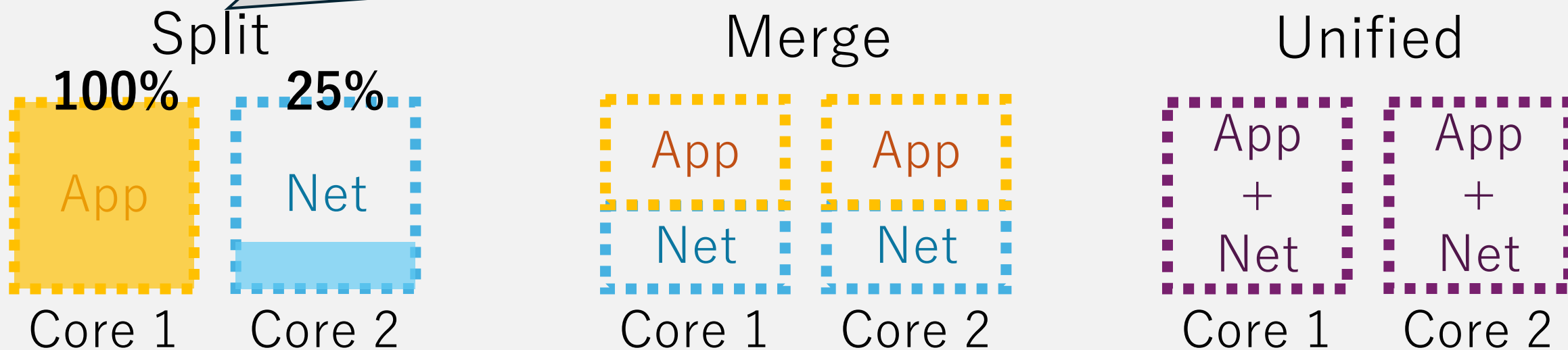
CPU 利用率

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

忙しいスレッドが、利用されていない CPU リソースを利用できない



CPU 利用率が  
低くなりやすい

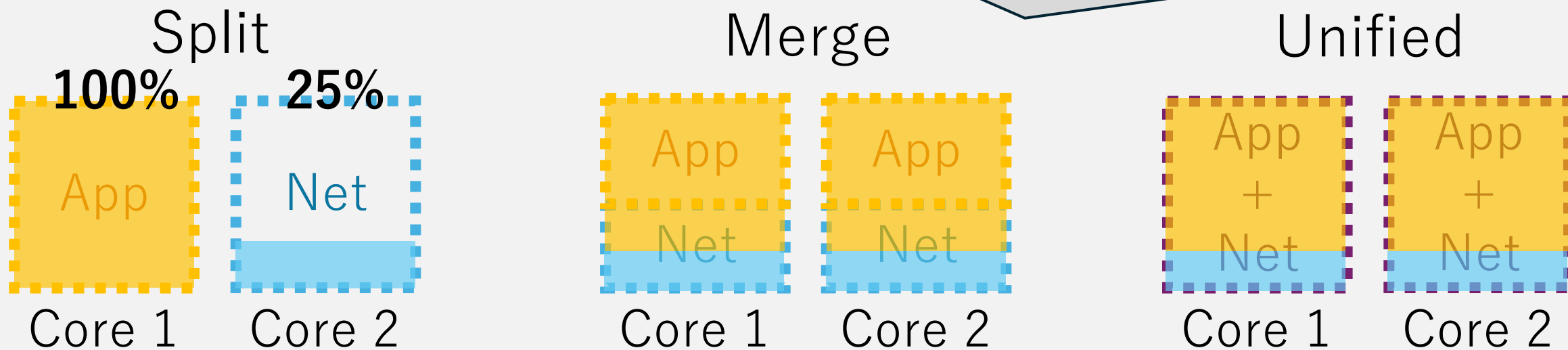
CPU 利用率

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

忙しいスレッドが、利用されていない CPU リソースを利用できる



CPU 利用率が  
低くなりやすい

CPU 利用率



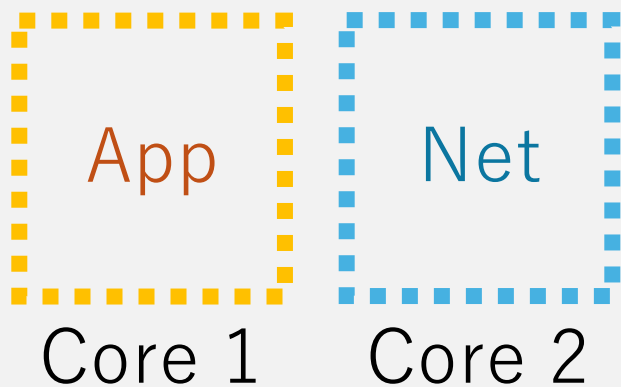
開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

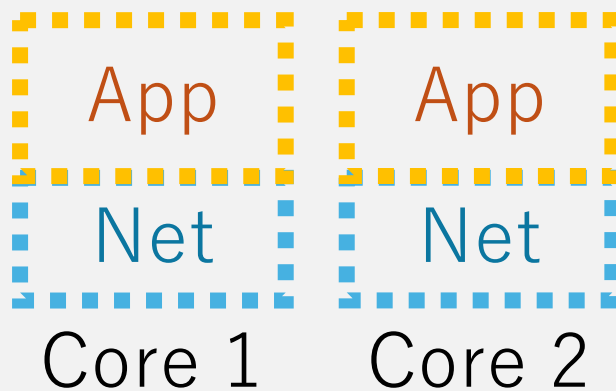
考えられる三通りの CPU コア割り当てパターン

Split

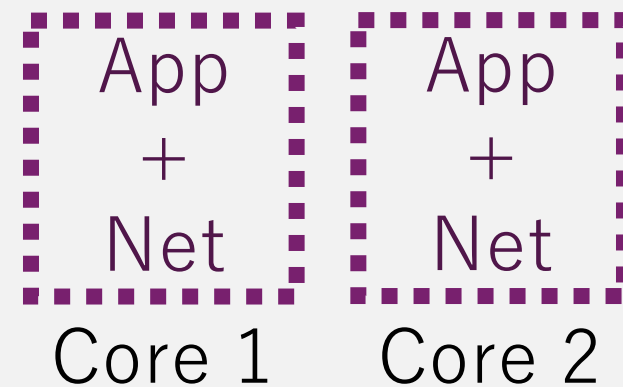


CPU 利用率が  
低くなりやすい

Merge



Unified



コンテキスト切り替え

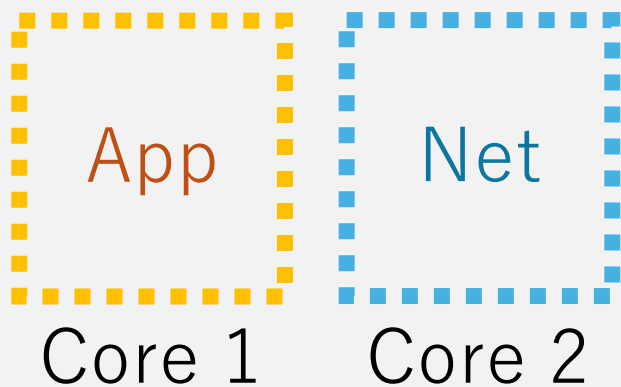
開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

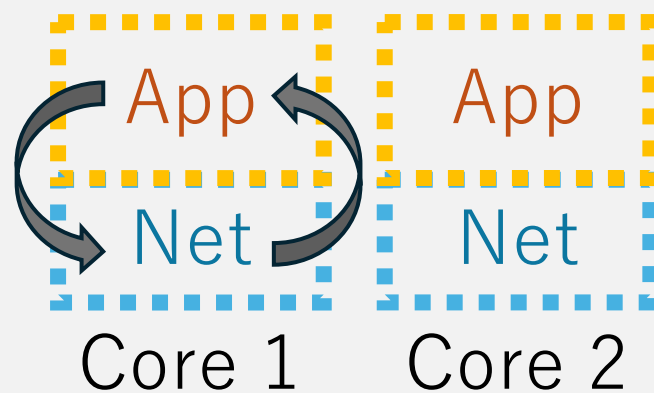
アプリと通信処理の実行切り替えにスレッド切り替えが必要

Split

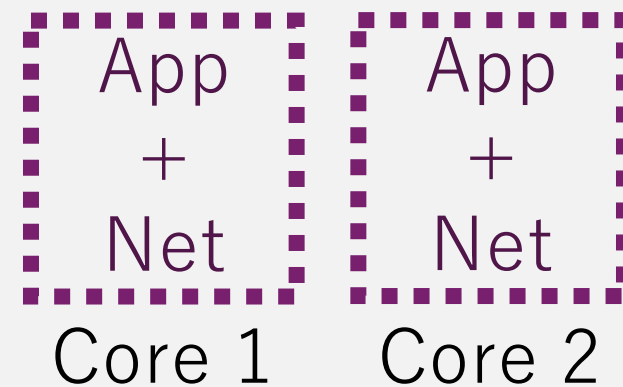


CPU 利用率が  
低くなりやすい

Merge



Unified



コンテキスト切り替え

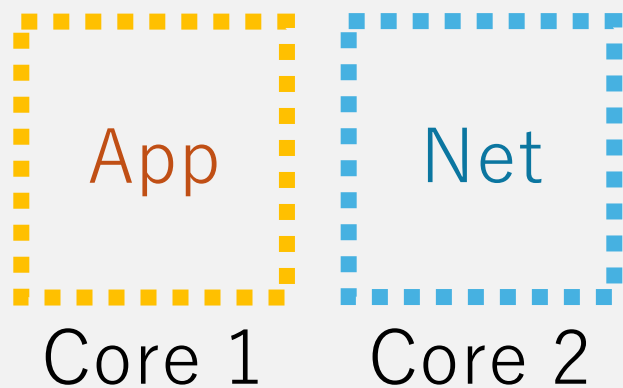
開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

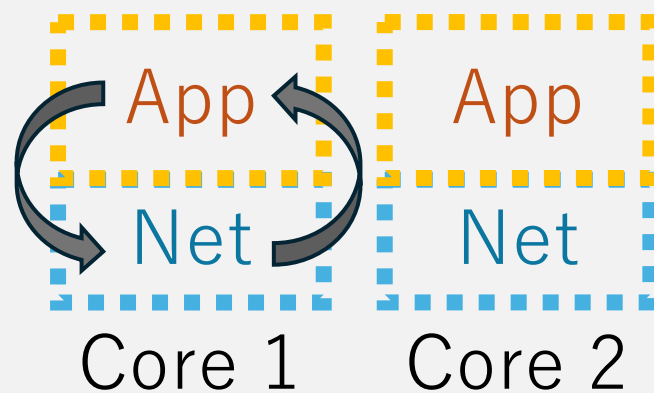
アプリと通信処理の実行切り替えにスレッド切り替えが必要

Split



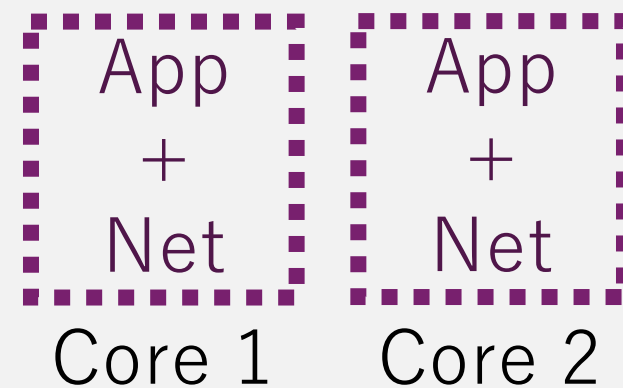
CPU 利用率が  
低くなりやすい

Merge



コンテキスト切り替え  
コストが付帯

Unified



コンテキスト切り替え

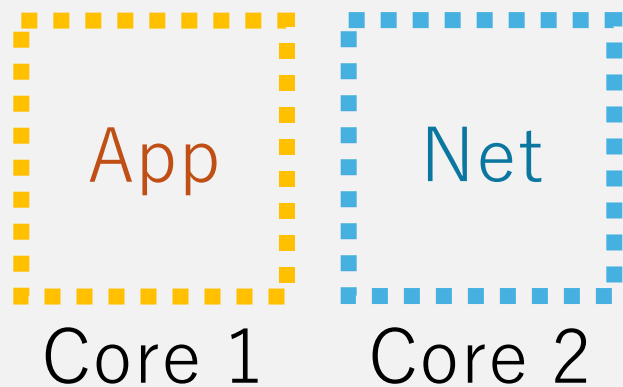
開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

CPU コア割り当てパターンの制限

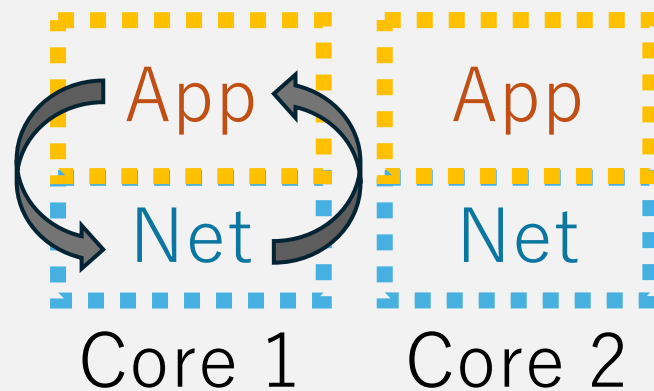
Split と Unified パターンではスレッド切り替えのコストは付帯せず

Split



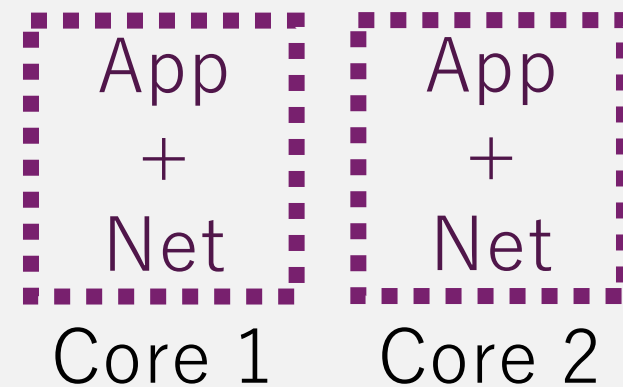
CPU 利用率が  
低くなりやすい

Merge



コンテキスト切り替え  
コストが付帯

Unified



コンテキスト切り替え

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

CPU コア割り当てパターンの制限

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム

アプリ開発者による実装

アプリケーション

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム

アプリ開発者による実装

アプリケーション

多くの TCP/IP スタック実装は  
CPU コア割り当てパターンの  
選択を設計に含んでいる

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム

アプリ開発者による実装

アプリケーション

結果、アプリ開発者が選択可能な  
CPU コア割り当てパターンが  
制限されている

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

## CPU コア割り当てパターンの制限

性能に最適化された  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム

アプリ開発者による実装

アプリケーション

この制限は  
高い性能を達成するために適した  
実装をすることを困難にする



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

可搬性に配慮した  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

可搬性に配慮した  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム

可搬性に配慮した  
TCP/IP スタックは  
限られた機能のみを  
実装している場合が多い

開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

可搬性に配慮した  
TCP/IP スタック

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

スレッド

環境依存の機能は  
アプリ開発者による  
提供が想定されている

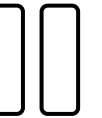
アプリ開発者による実装

TCP/IP スタックを  
実行するスレッド

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

可搬性に配慮した  
TCP/IP スタック

この設計が可搬性に  
寄与している

TCP/IP 処理

環境依存の機能は  
アプリ開発者による  
提供が想定されている

アプリ開発者による実装

TCP/IP スタックを  
実行するスレッド

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

性能に重要な項目への配慮の不足

可搬性に配慮した  
TCP/IP スタック

TCP/IP 処理

アプリ開発者による実装

TCP/IP スタックを  
実行するスレッド

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

性能に重要な項目への配慮の不足

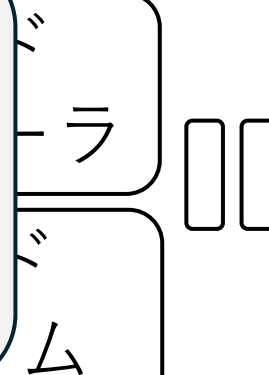
可搬性に配慮した  
TCP/IP スタック

アプリ開発者による実装

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

- NIC オフロード機能への配慮が不足
- ゼロコピー送受信をサポートしていない
- マルチコア環境で性能がスケールしない



開発している TCP/IP スタック実装

# 既存の TCP/IP スタック実装の課題

性能に重要な項目への配慮の不足

可搬性に配慮した  
TCP/IP スタック

アプリ開発者による実装

結果、特定のワークロードで高い性能を発揮できない

TCP/IP 処理

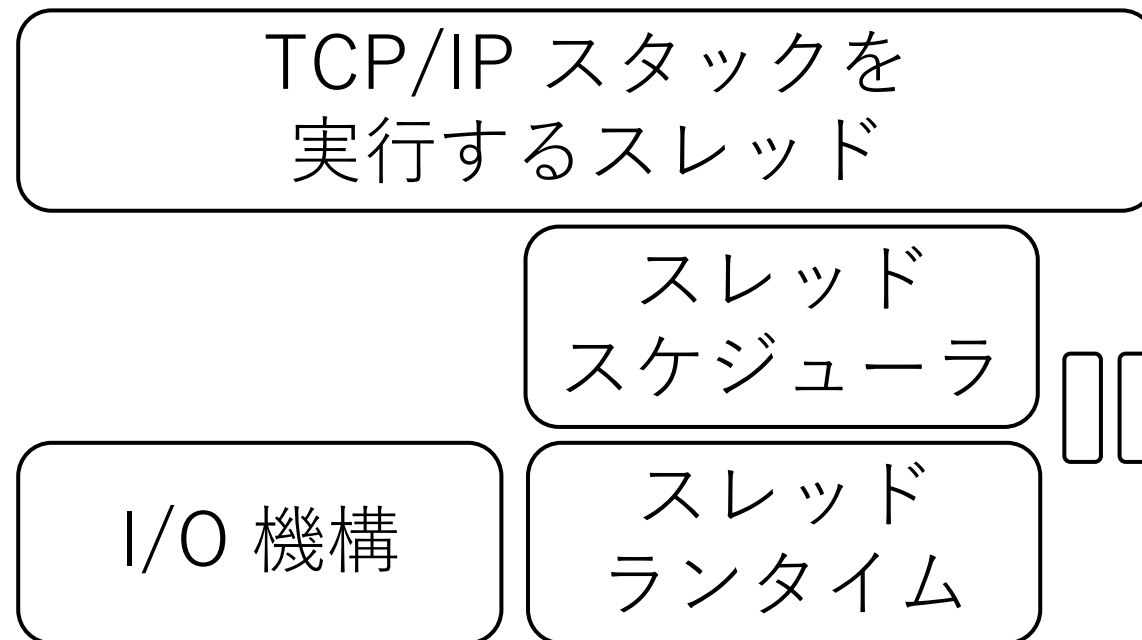
- NIC オフロード機能への配慮が不足
- ゼロコピー送受信をサポートしていない
- マルチコア環境で性能がスケールしない

開発している TCP/IP スタック実装

iip



アプリ開発者による実装





# 開発している TCP/IP スタック実装

iip

限られた機能のみを実装

iip

TCP/IP 処理

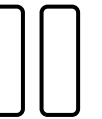
## アプリ開発者による実装

TCP/IP スタックを  
実行するスレッド

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



# 開発している TCP/IP スタック実装

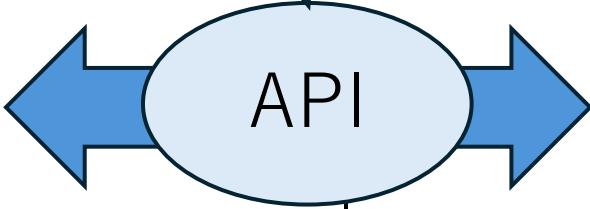
iip

環境固有機能はコールバック API を通じた  
アプリ開発者からの提供を想定

アプリ開発者による実装

TCP/IP スタックを  
実行するスレッド

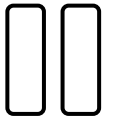
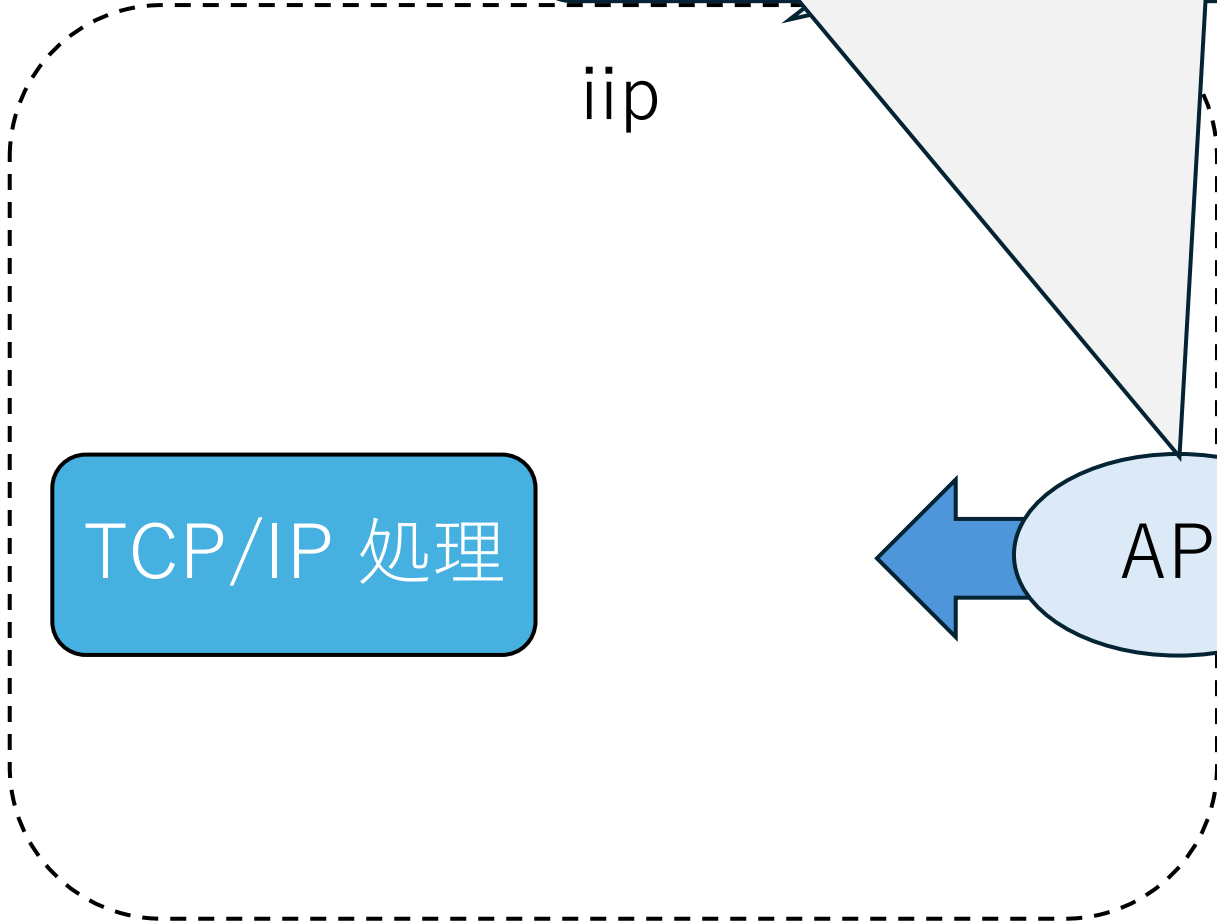
TCP/IP 処理



スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



# 開発している TCP/IP スタック実装

iip

環境固有機能はコールバック API を通じた  
アプリ開発者からの提供を想定

アプリ開発者による実装

特定の CPU、NIC、OS、  
ライブラリ、コンパイラへ依存しない

TCP/IP スタックを  
実行するスレッド

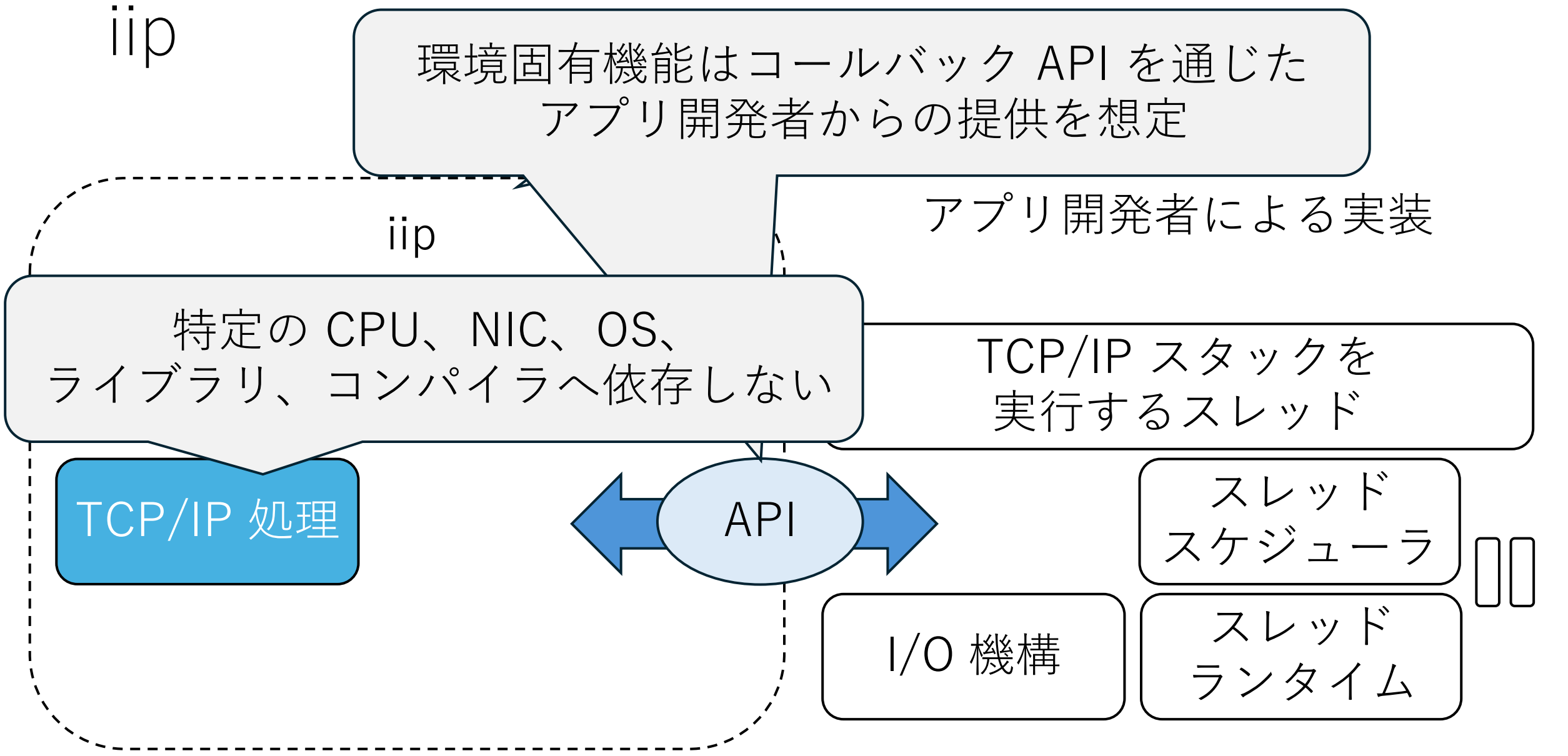
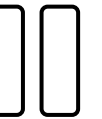
TCP/IP 処理

API

スレッド  
スケジューラ

I/O 機構

スレッド  
ランタイム



# 開発している TCP/IP スタック実装

iip

環境固有機能はコールバック API を通じた  
アプリ開発者からの提供を想定

アプリ開発者による実装

iip

特定の CPU、NIC、OS、  
ライブラリ、コンパイラへ依存しない

TCP/IP スタックを  
実行するスレッド

TCP/IP 処理

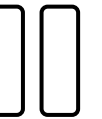
API

スレッド  
スケジューラ

新旧のコンパイラでビルド可能なよう  
C89/C++98 標準へ準拠

I/O 機構

スレッド  
ランタイム



# 開発している TCP/IP スタック実装

iip

環境固有機能はコールバック API を通じた  
アプリ開発者からの提供を想定

アプリ開発者による実装

iip

特定の CPU、NIC、OS、  
ライブラリ、コンパイラへ依存しない

TCP/IP スタックを

性能に重要な項目へ配慮

TCP/IP 処理

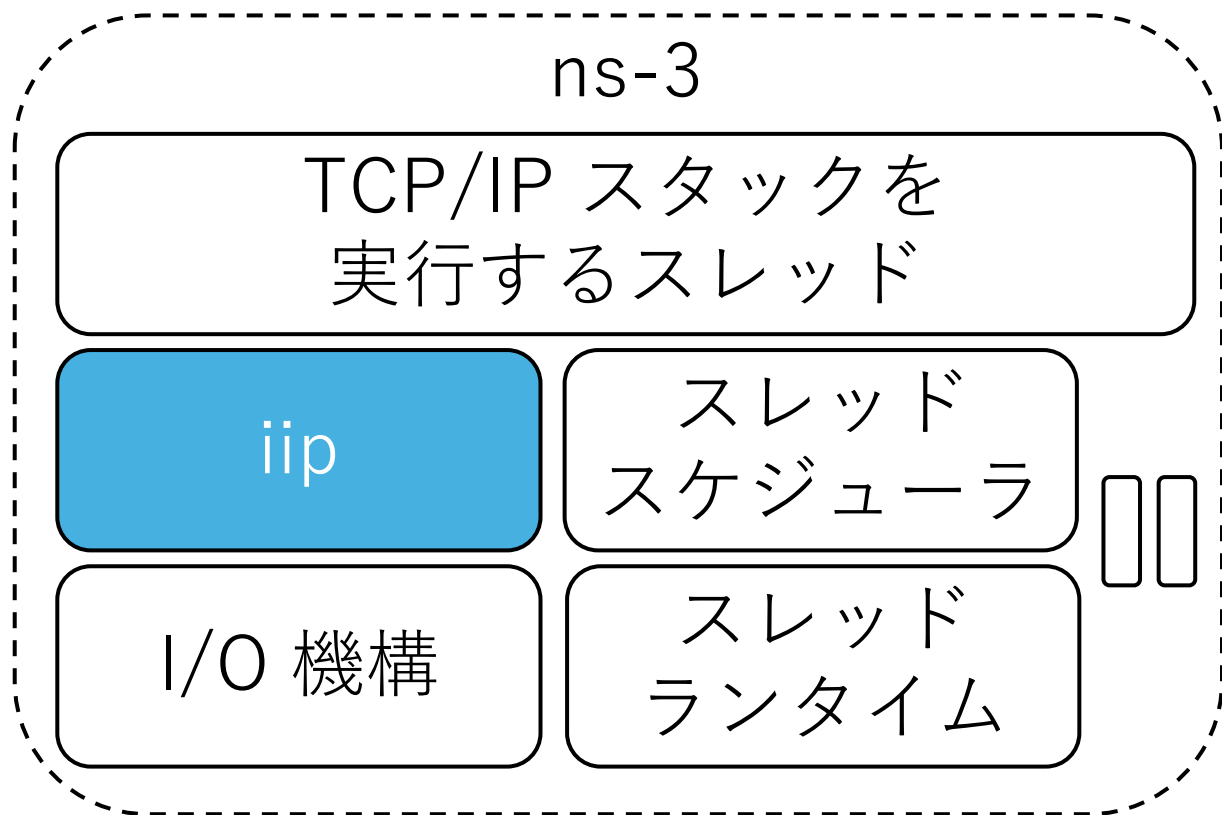
- NIC オフロード機能への配慮
- ゼロコピー送受信をサポート
- マルチコア環境での性能への配慮

新旧のコンパイラでビルド可能  
C89/C++98 標準へ準拠

開発している TCP/IP スタック実装

## 組み込みやすさへ配慮することの利点

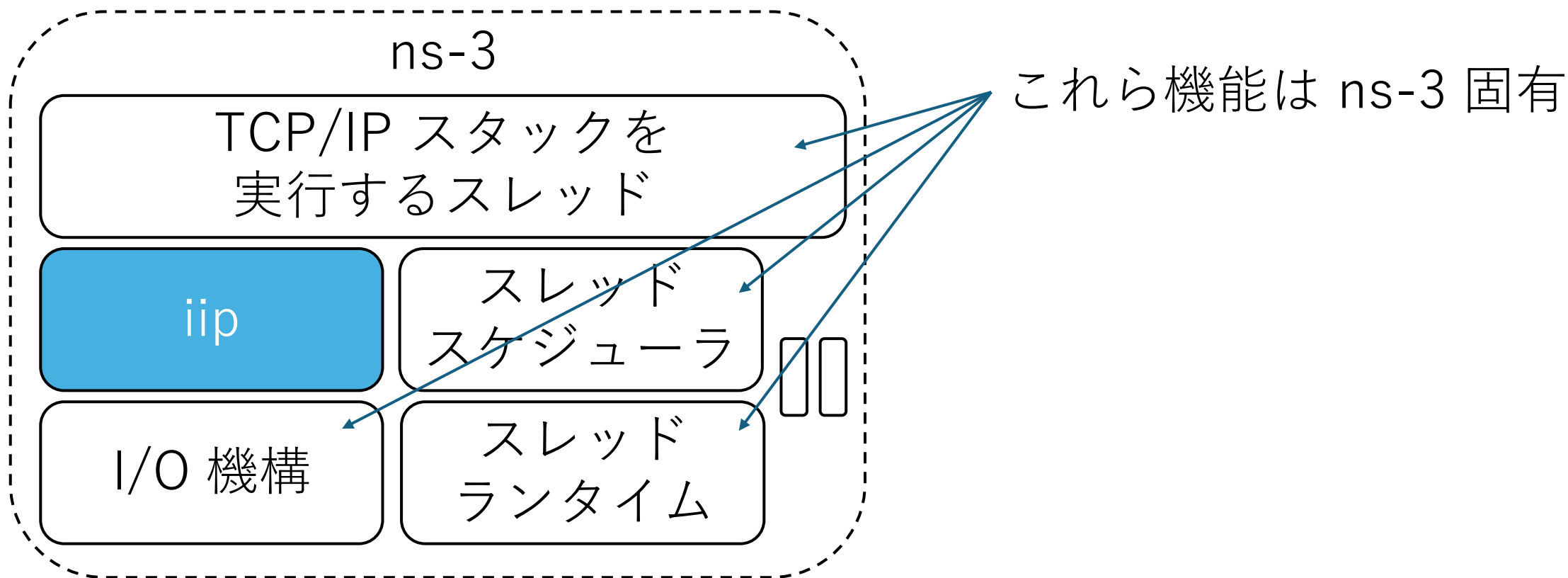
- ns-3 シミュレータの上で比較的簡単に利用可能
  - <https://github.com/yasukata/iip-ns>



開発している TCP/IP スタック実装

# 組み込みやすさへ配慮することの利点

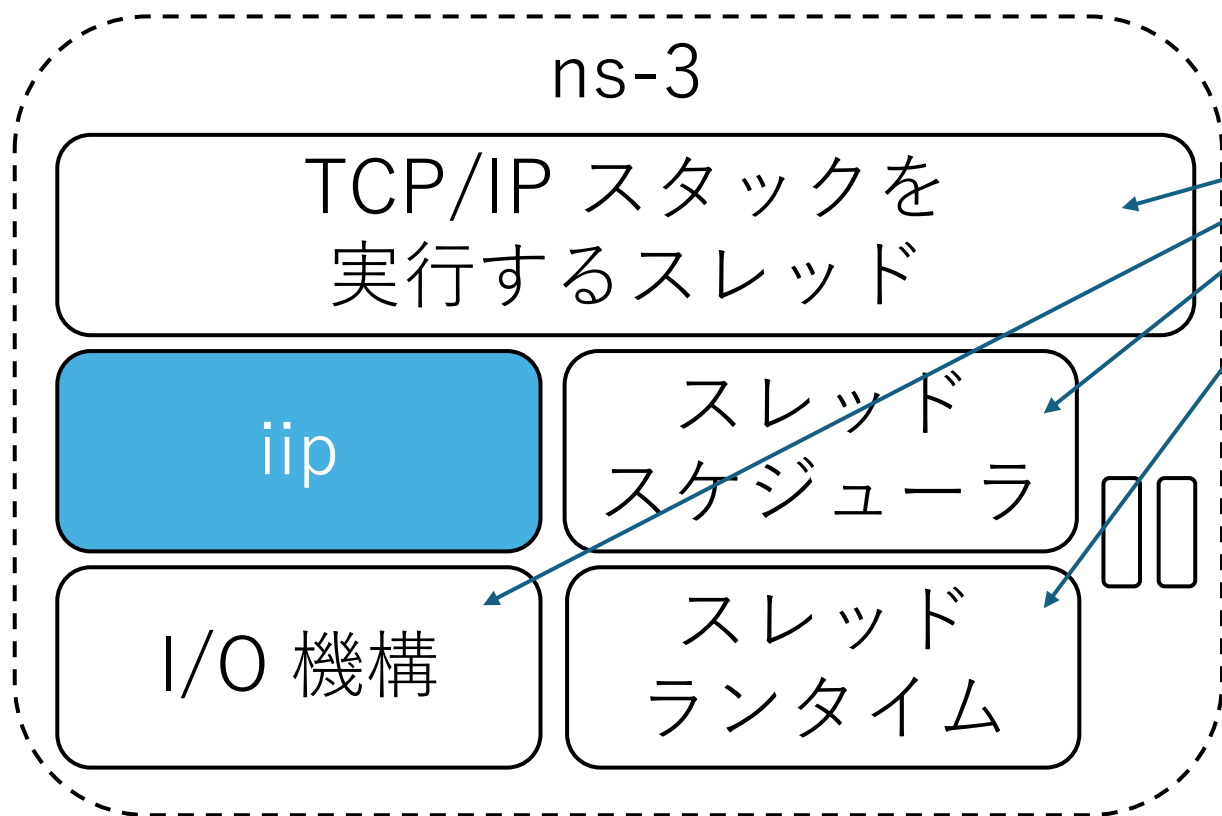
- ns-3 シミュレータの上で比較的簡単に利用可能
  - <https://github.com/yasukata/iip-ns>



開発している TCP/IP スタック実装

# 組み込みやすさへ配慮することの利点

- ns-3 シミュレータの上で比較的簡単に利用可能
  - <https://github.com/yasukata/iip-ns>



これら機能は ns-3 固有



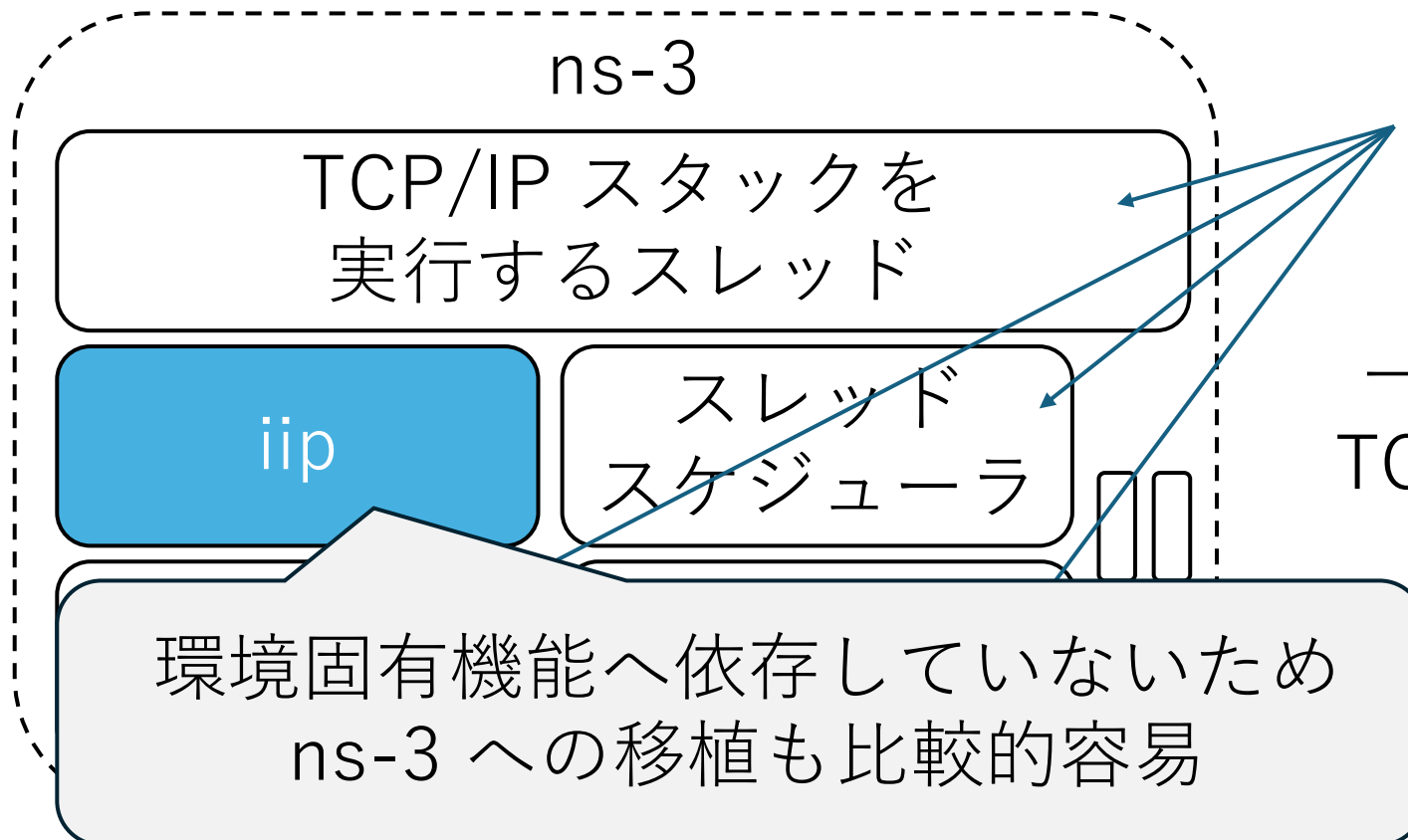
一般的な OS 機能に依存する TCP/IP スタック実装の場合は移植が複雑になりがち



開発している TCP/IP スタック実装

# 組み込みやすさへ配慮することの利点

- ns-3 シミュレータの上で比較的簡単に利用可能
  - <https://github.com/yasukata/iip-ns>



これら機能は ns-3 固有

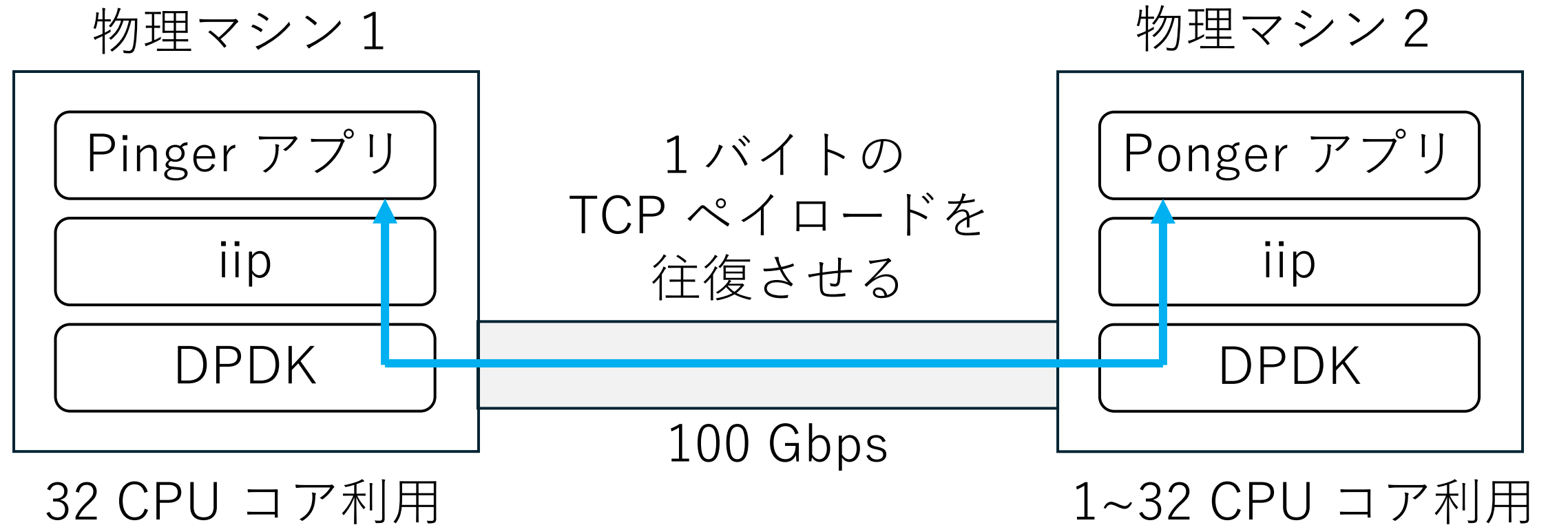


一般的な OS 機能に依存する TCP/IP スタック実装の場合は移植が複雑になりがち

開発している TCP/IP スタック実装

# 評価：小さいメッセージの交換

- TCP ping-pong ワークロード

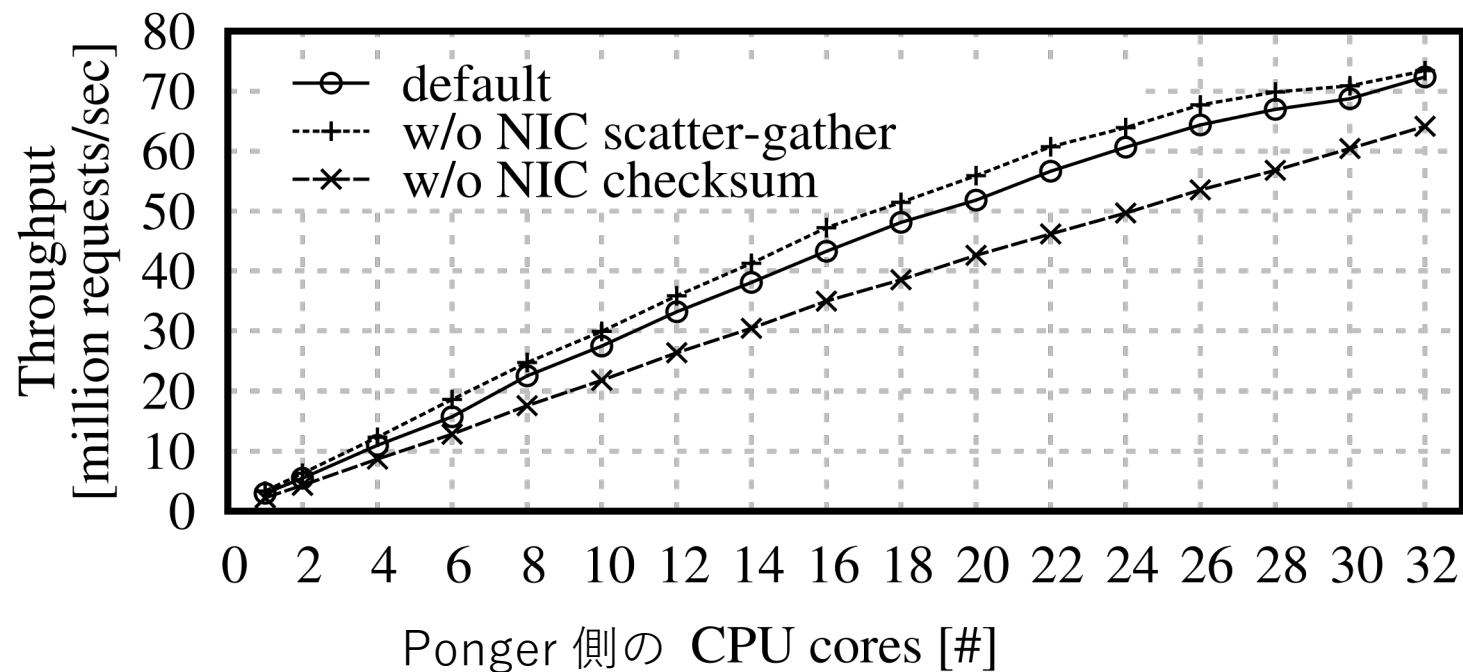


Ponger 側で 1 スレッドが 32 並列接続へ対応するよう接続数を設定

開発している TCP/IP スタック実装

# 評価：小さいメッセージの交換

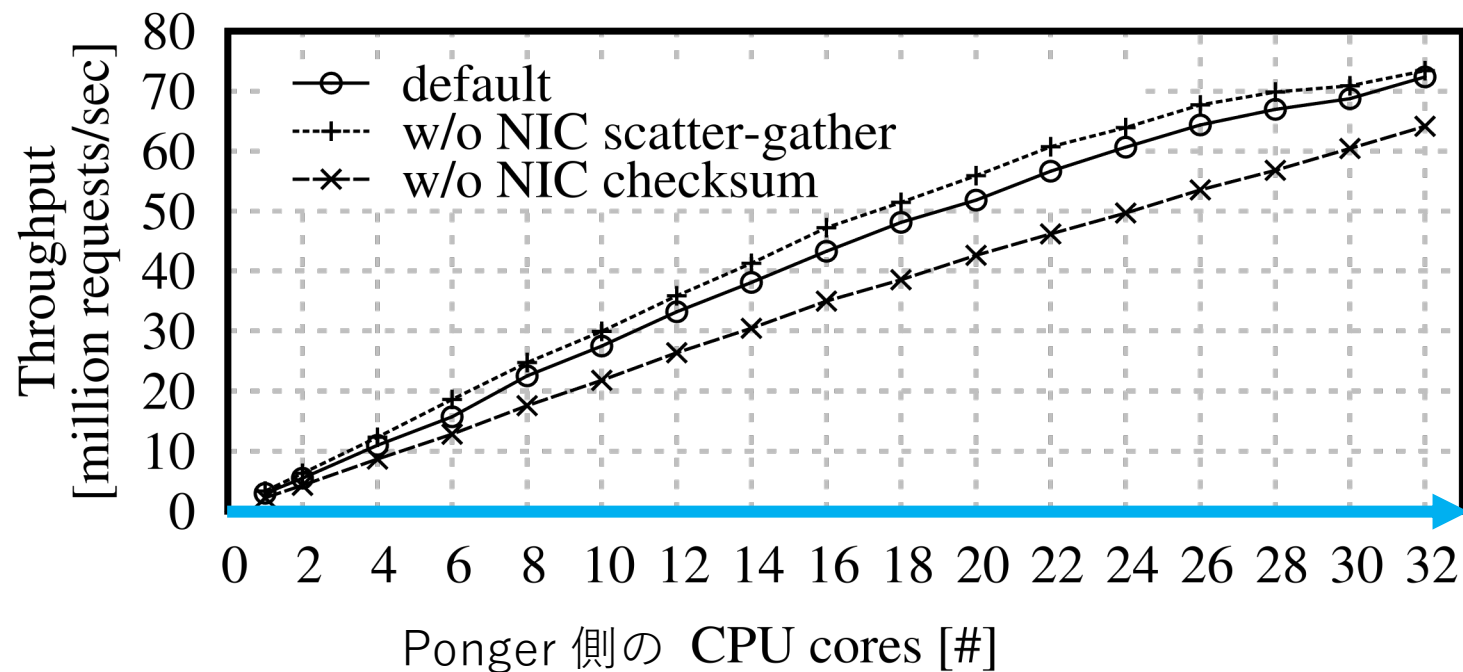
- 1バイトの TCP ペイロードを往復させる



開発している TCP/IP スタック実装

## 評価：小さいメッセージの交換

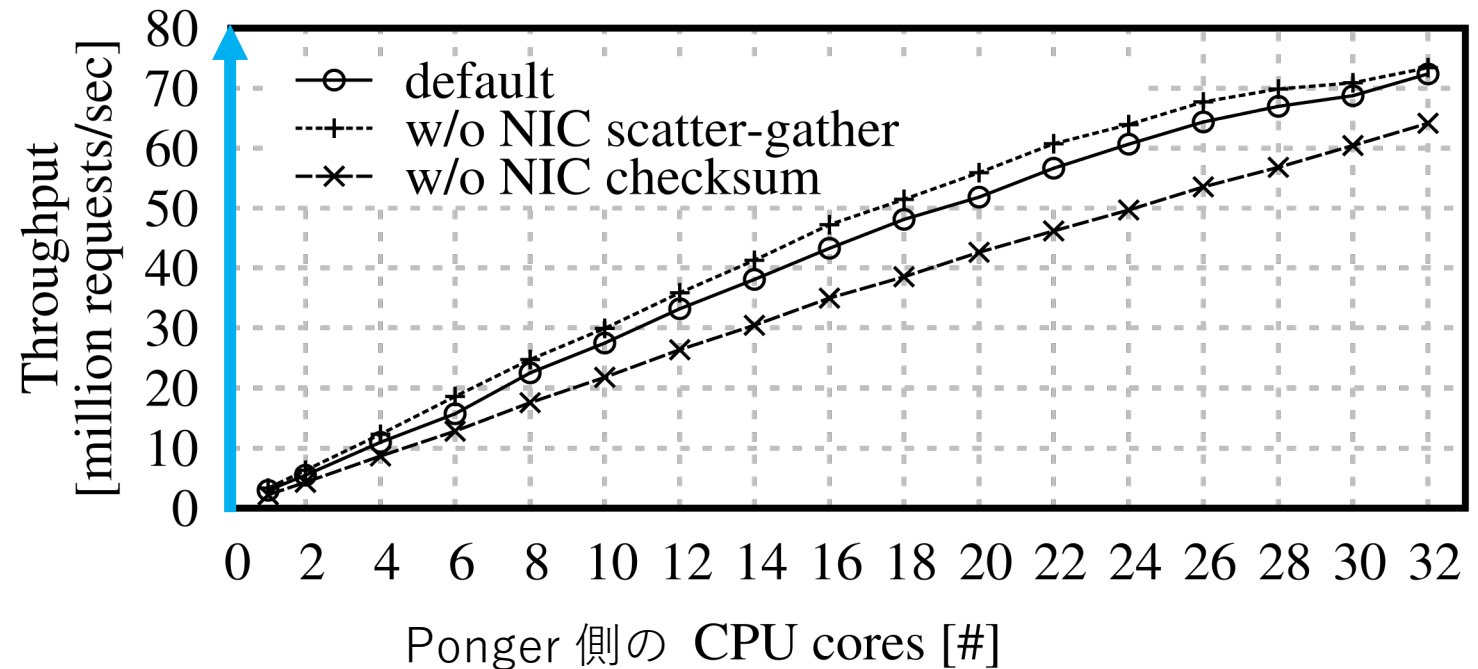
- 1バイトの TCP ペイロードを往復させる



開発している TCP/IP スタック実装

# 評価：小さいメッセージの交換

- 1バイトの TCP ペイロードを往復させる

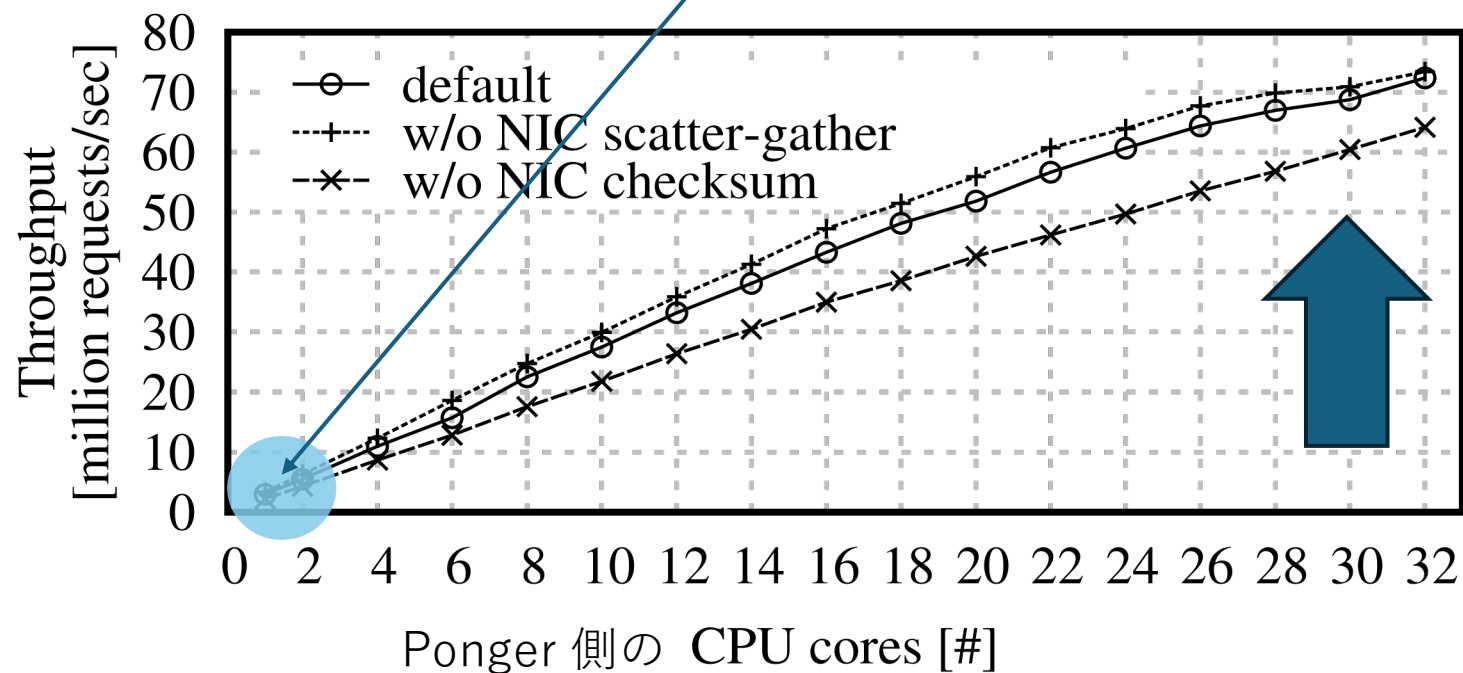


開発している TCP/IP スタック実装

## 評価：小さいメッセージの交換

- 1バイトの TCP ペイロードを往復させる
- 性能に寄与する要因
  - 複数 CPU コアの効果的な利用

マルチコア環境で  
スケールしない実装の性能

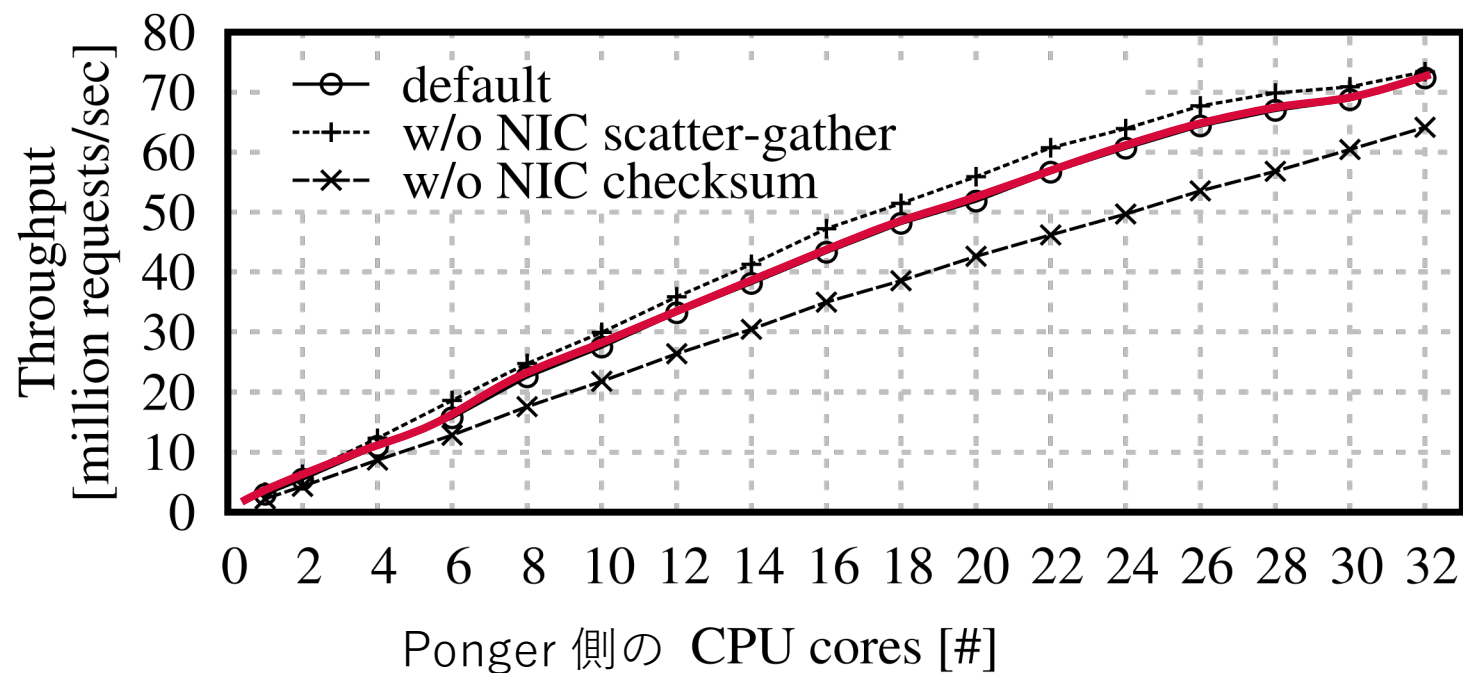


開発している TCP/IP スタック実装

## 評価：小さいメッセージの交換

- 1バイトの TCP ペイロードを往復させる
- 性能に寄与する要因
  - 複数 CPU コアの効果的な利用

全てのオフロード機能を有効にした場合

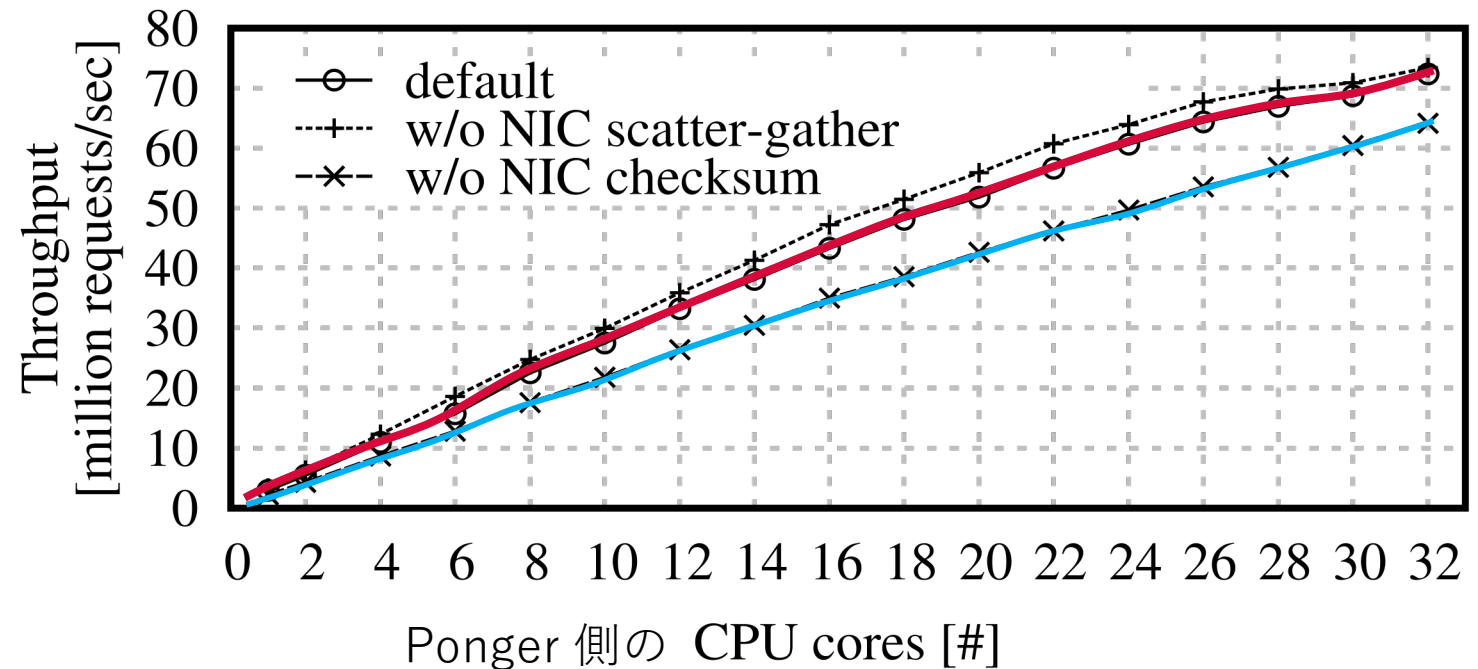


開発している TCP/IP スタック実装

# 評価：小さいメッセージの交換

- 1バイトの TCP ペイロードを往復させる
- 性能に寄与する要因
  - 複数 CPU コアの効果的な利用
  - チェックサムオフロード

チェックサムオフロード機能を  
無効にした場合





開発している TCP/IP スタック実装

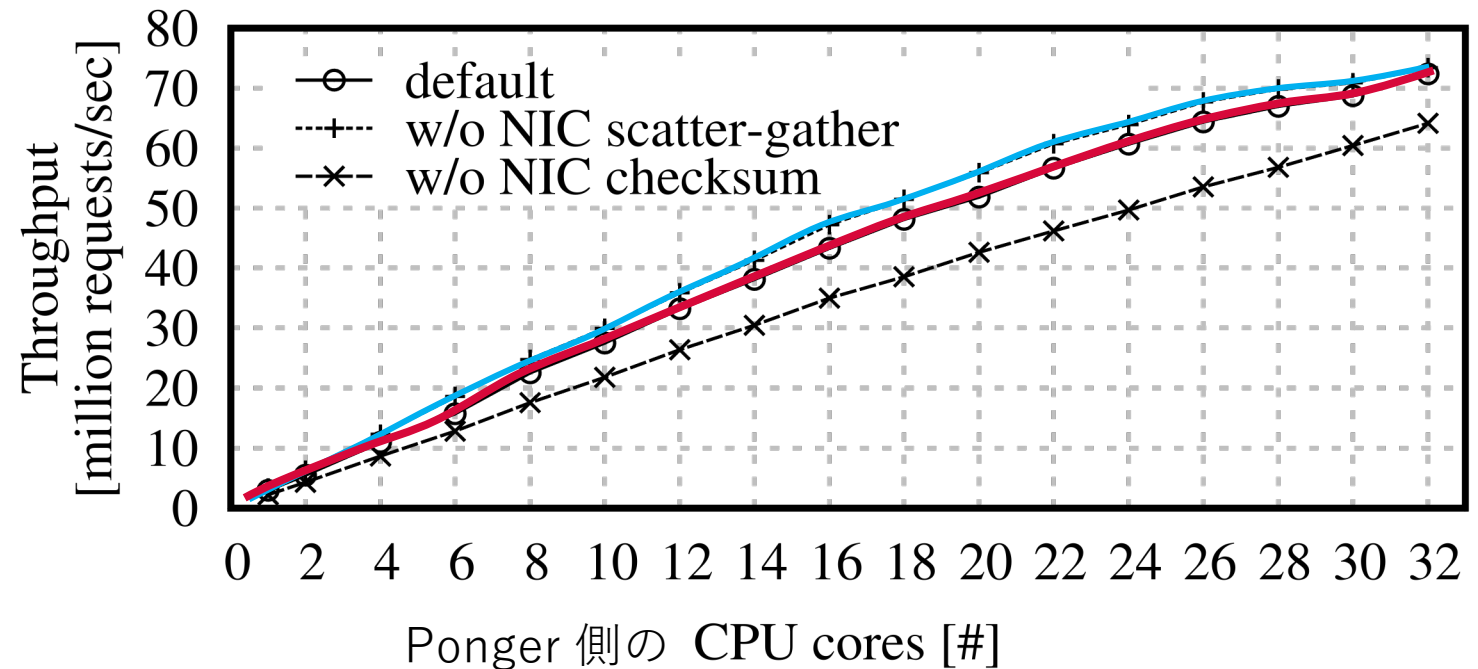
# 評価：小さいメッセージの交換

- 1バイトの TCP ペイロードを往復させる
- 性能に寄与する要因
  - 複数 CPU コアの効果的な利用
  - チェックサムオフロード

## • Tips

- データが小さい場合  
ゼロコピー送信は  
性能を向上しない

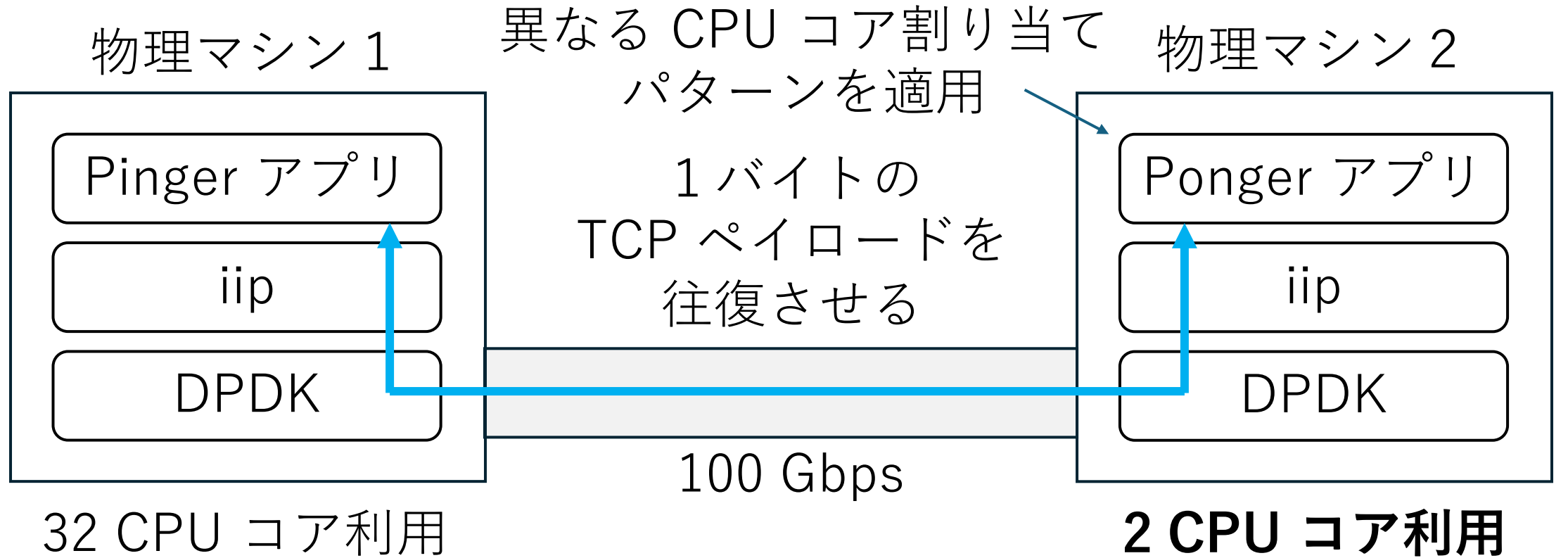
ゼロコピー送信を無効にした場合



開発している TCP/IP スタック実装

# 評価：CPU コア割り当て方法の影響

- 1 バイトの TCP ペイロードを往復させる

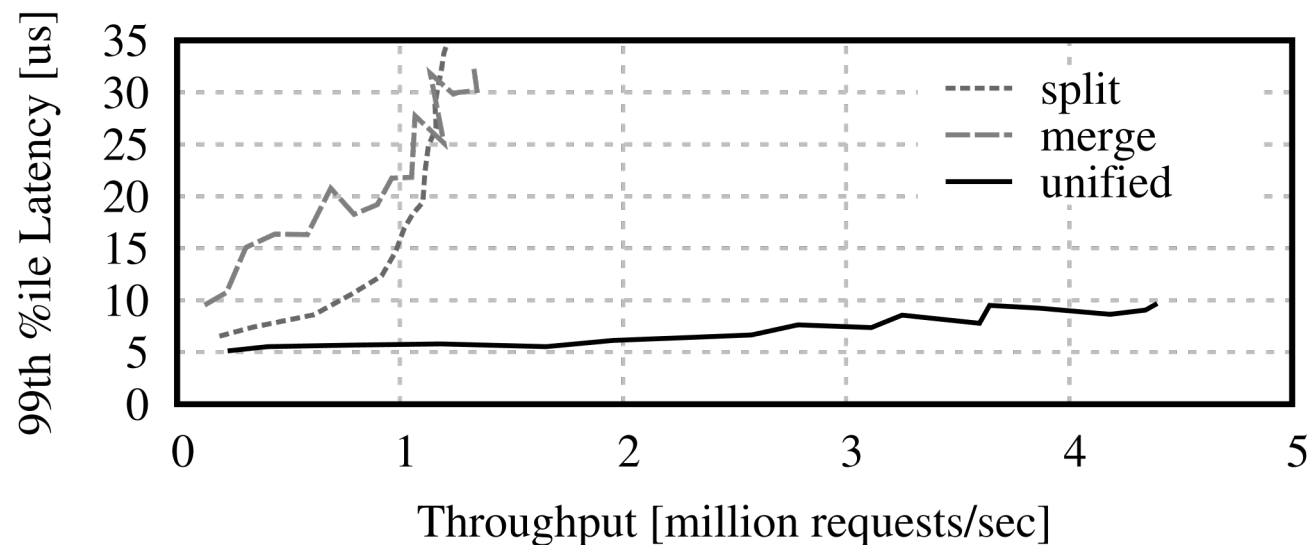


Ponger 側で 1 スレッドが 32 並列接続へ対応するように接続数を設定

開発している TCP/IP スタック実装

# 評価：CPU コア割り当て方法の影響

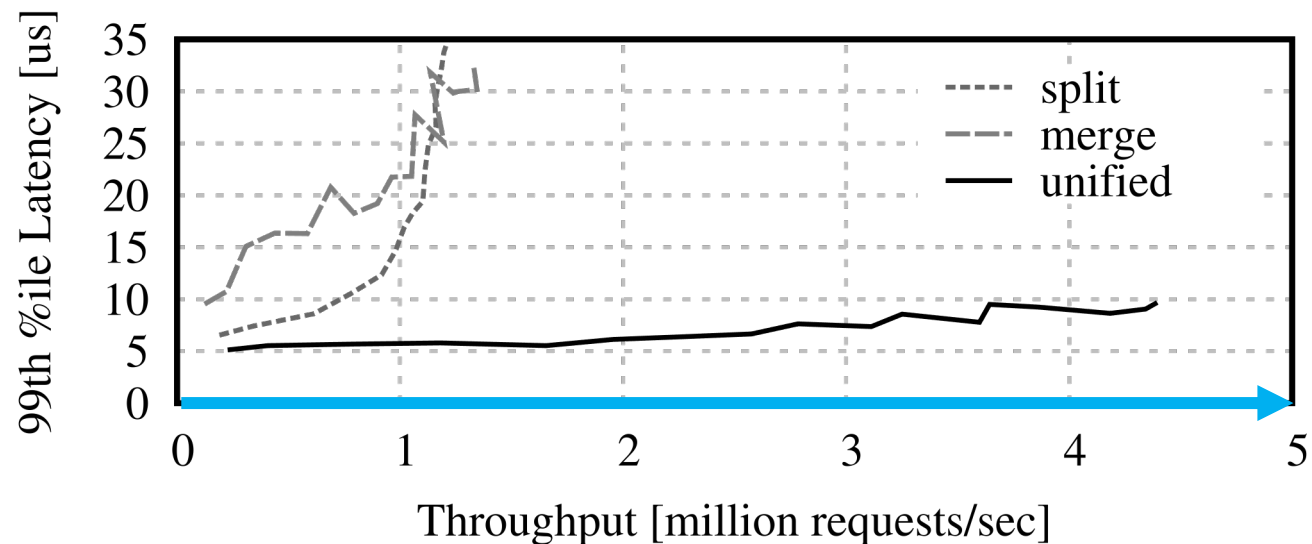
- 1 バイトの TCP ペイロードを往復させる



開発している TCP/IP スタック実装

# 評価：CPU コア割り当て方法の影響

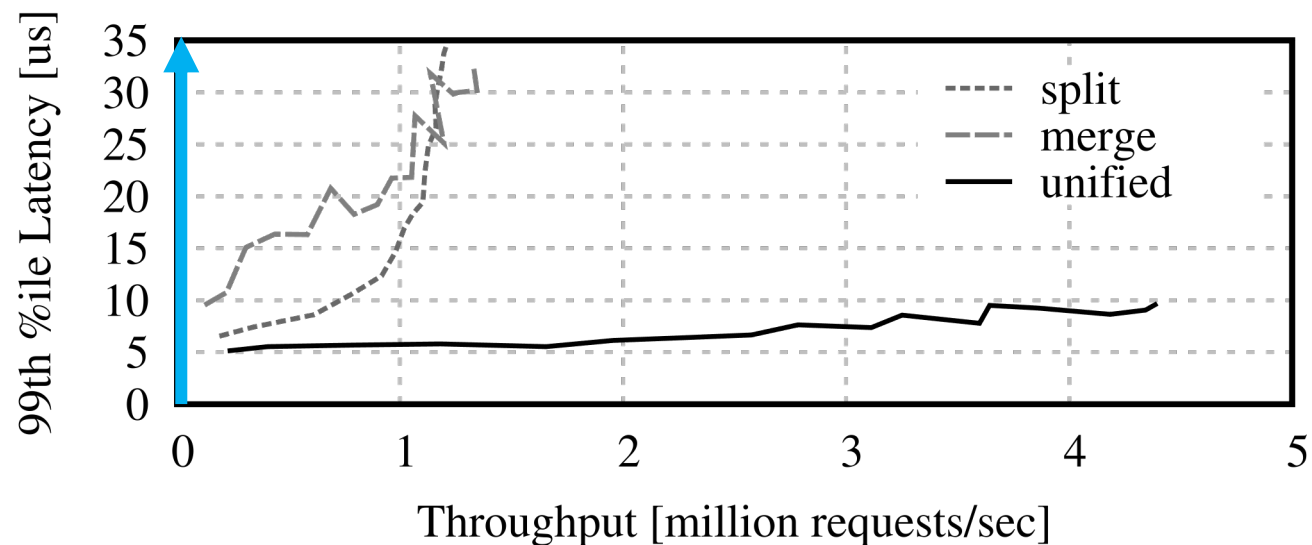
- 1 バイトの TCP ペイロードを往復させる



開発している TCP/IP スタック実装

# 評価：CPU コア割り当て方法の影響

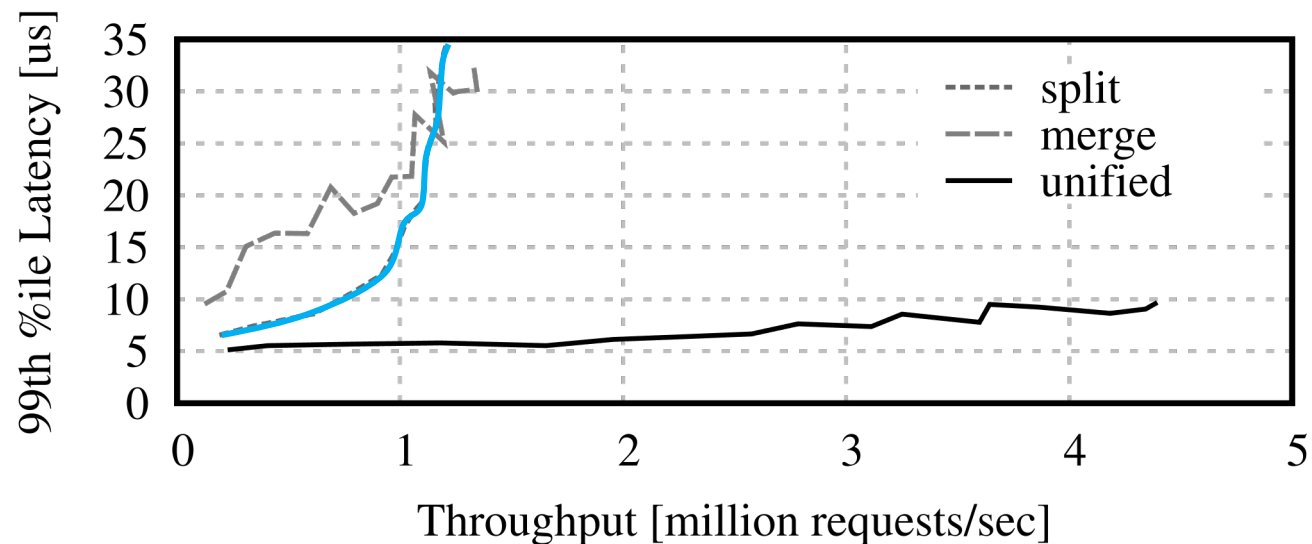
- 1バイトの TCP ペイロードを往復させる



開発している TCP/IP スタック実装

# 評価：CPU コア割り当て方法の影響

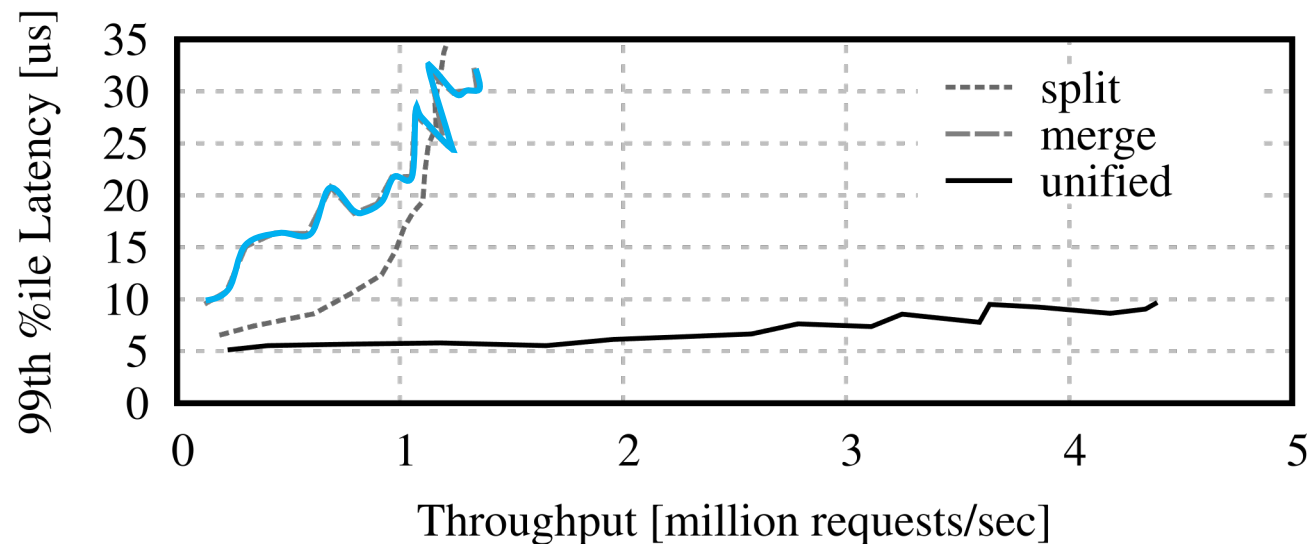
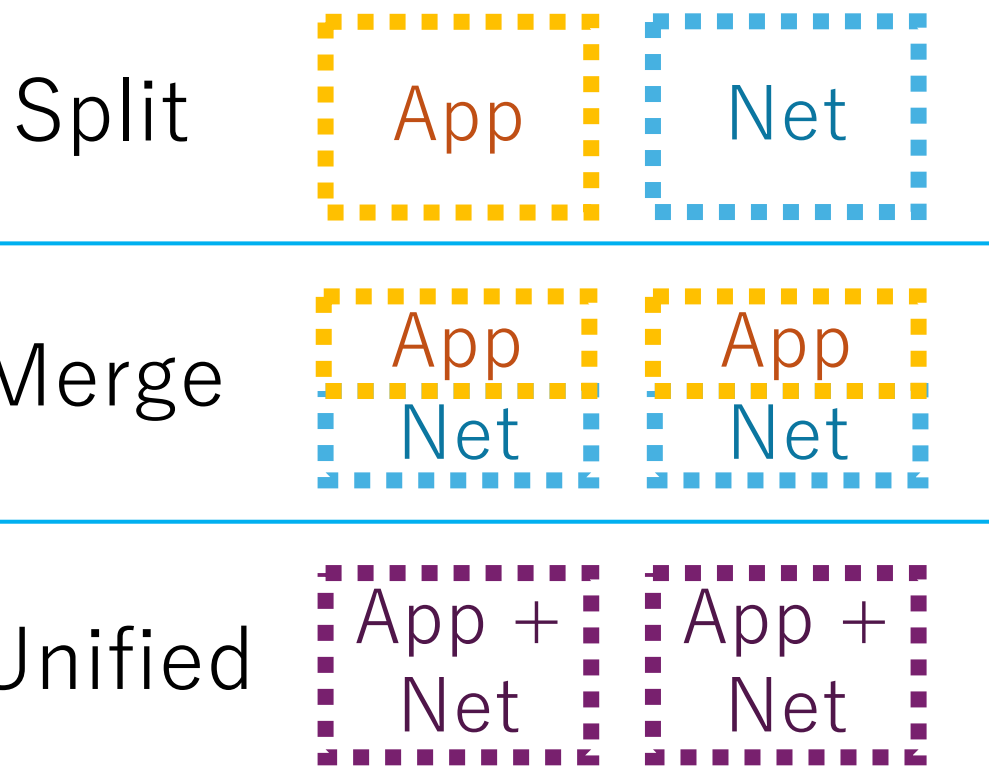
- 1バイトの TCP ペイロードを往復させる



開発している TCP/IP スタック実装

# 評価：CPU コア割り当て方法の影響

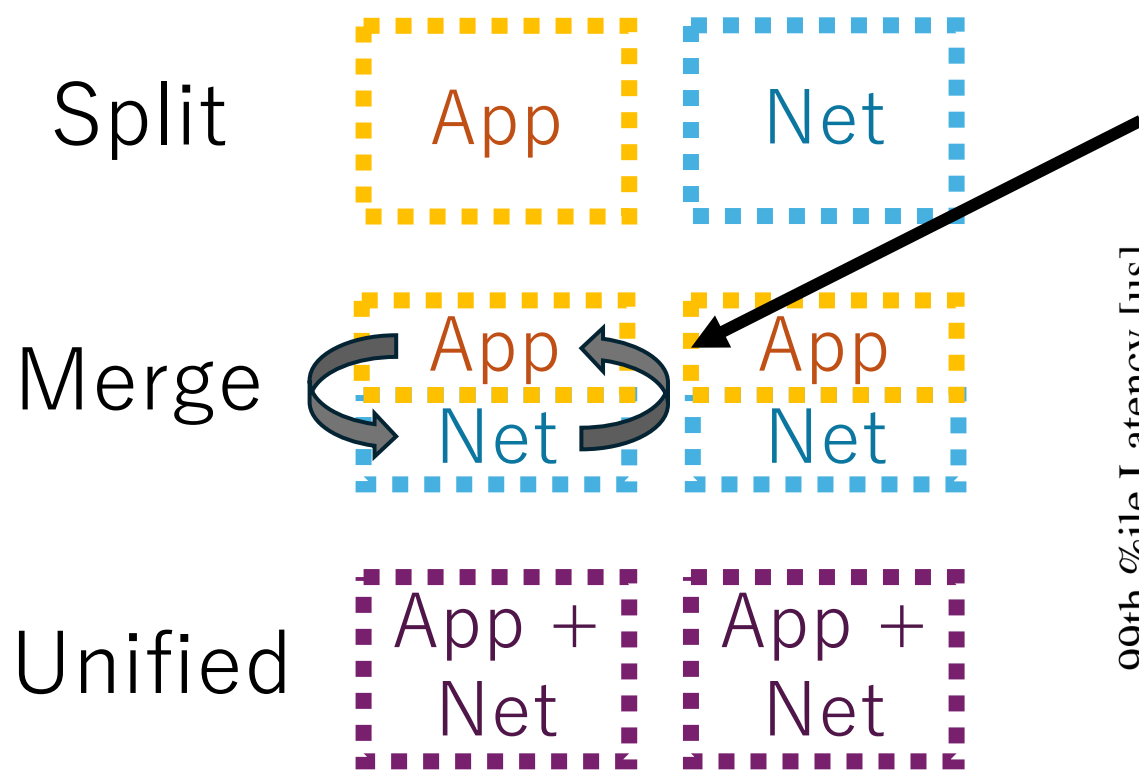
- 1バイトの TCP ペイロードを往復させる



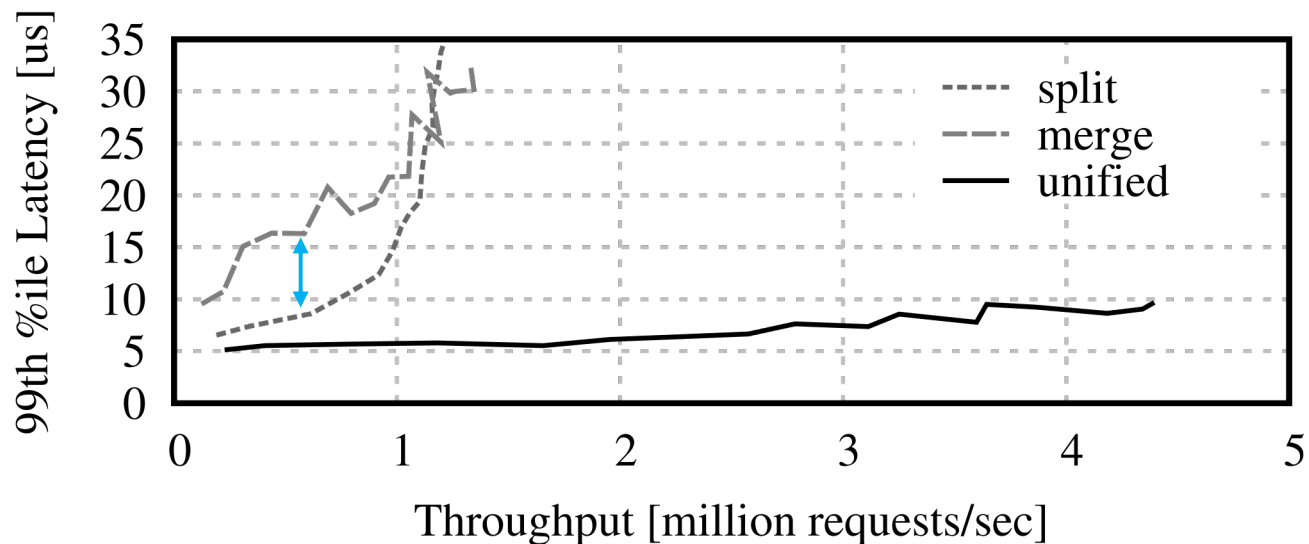
開発している TCP/IP スタック実装

# 評価：CPU コア割り当て方法の影響

- 1 バイトの TCP ペイロードを往復させる



コンテキスト切り替えコストによる遅延の増加

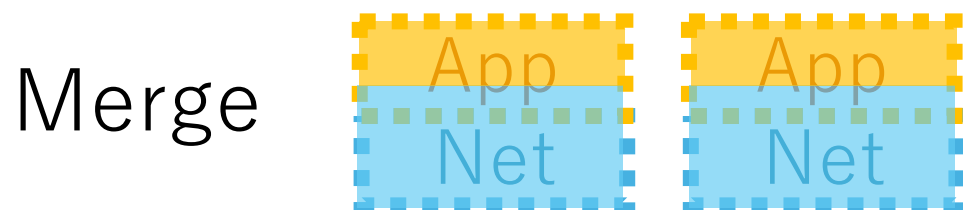




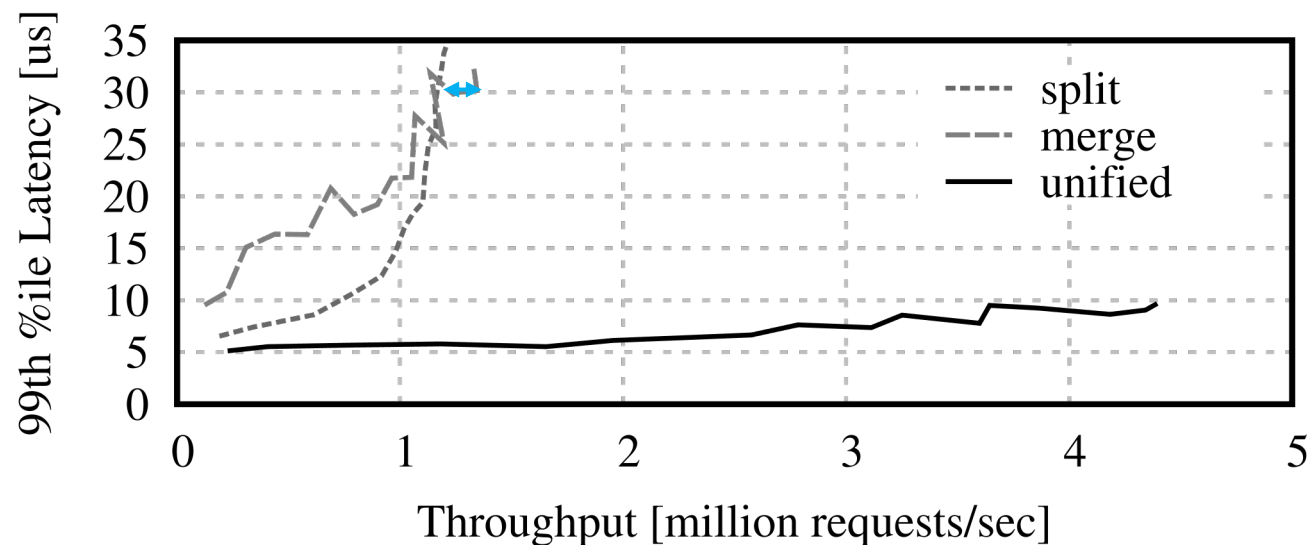
開発している TCP/IP スタック実装

# 評価：CPU コア割り当て方法の影響

- 1 バイトの TCP ペイロードを往復させる



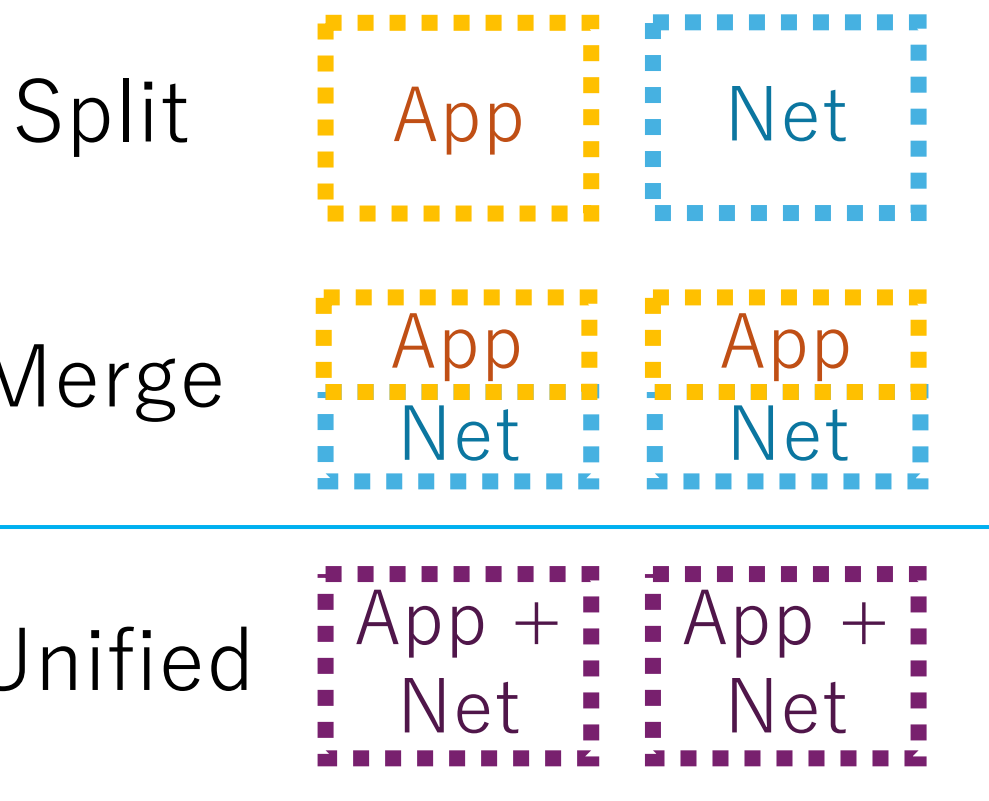
CPU を最大限利用できないことによる  
最大スループットの低下



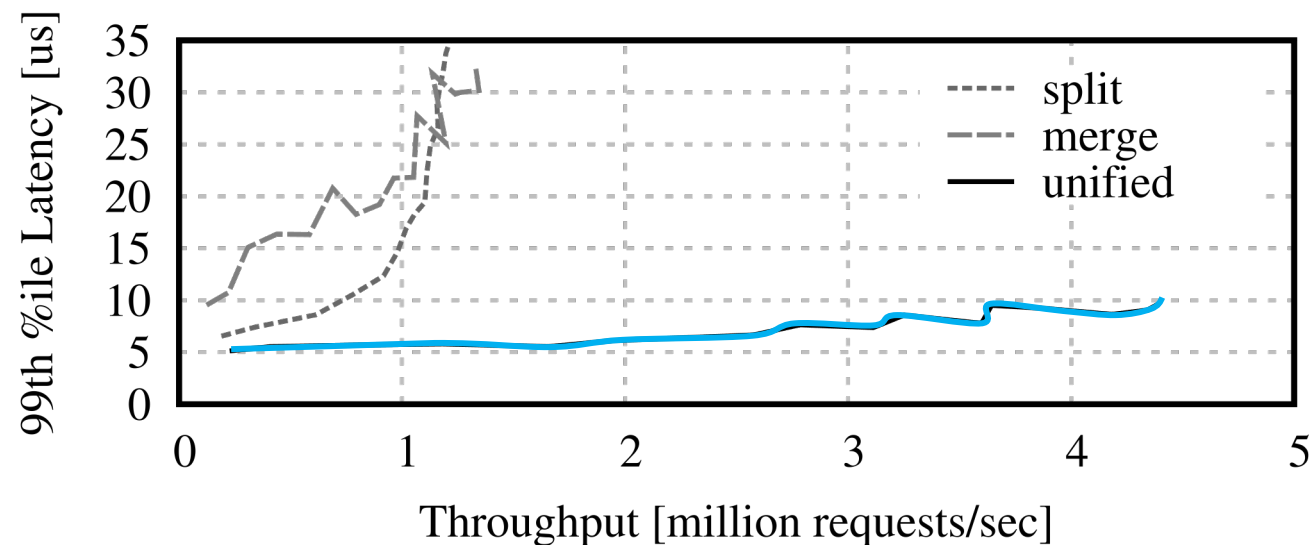
開発している TCP/IP スタック実装

# 評価：CPU コア割り当て方法の影響

- 1 バイトの TCP ペイロードを往復させる



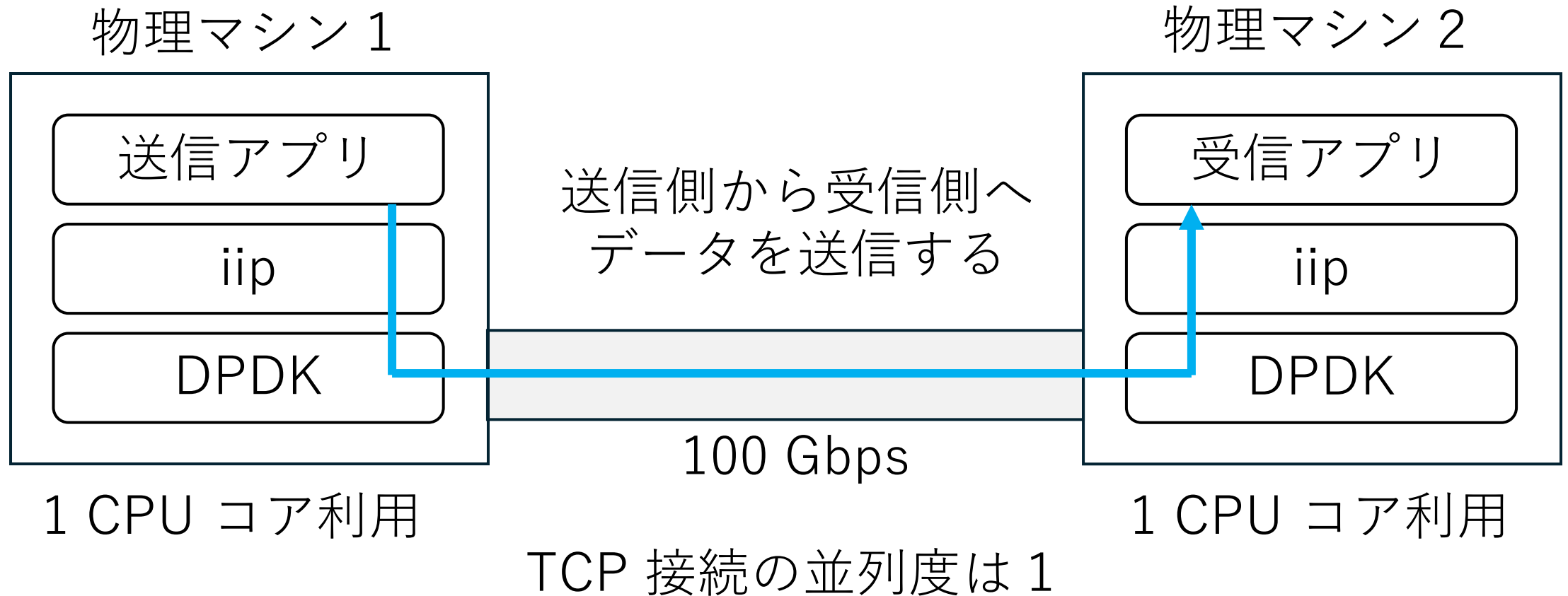
コンテキスト切り替えと CPU 利用率の制約がない場合の性能



開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

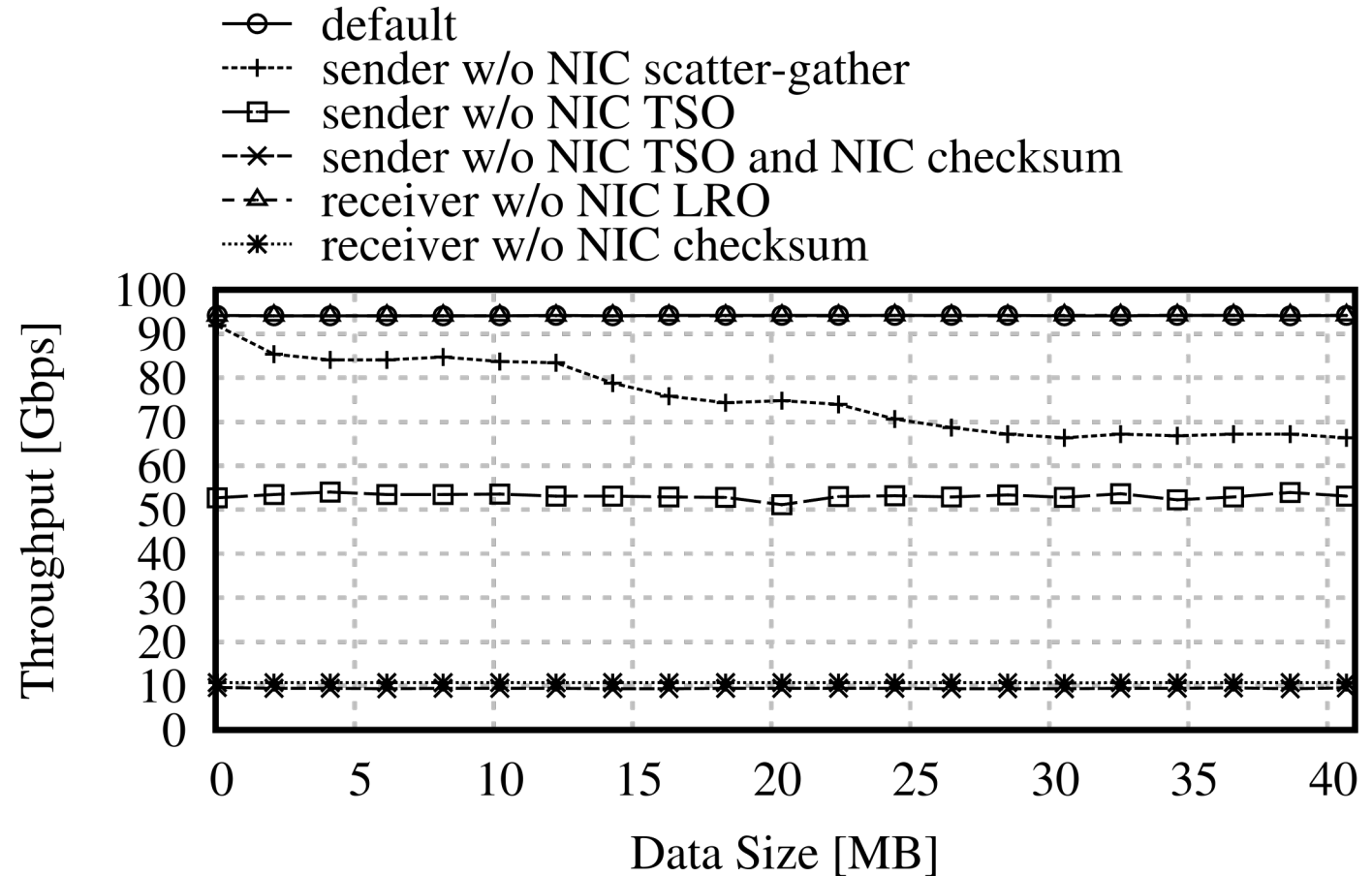
- 大きなデータの転送



開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

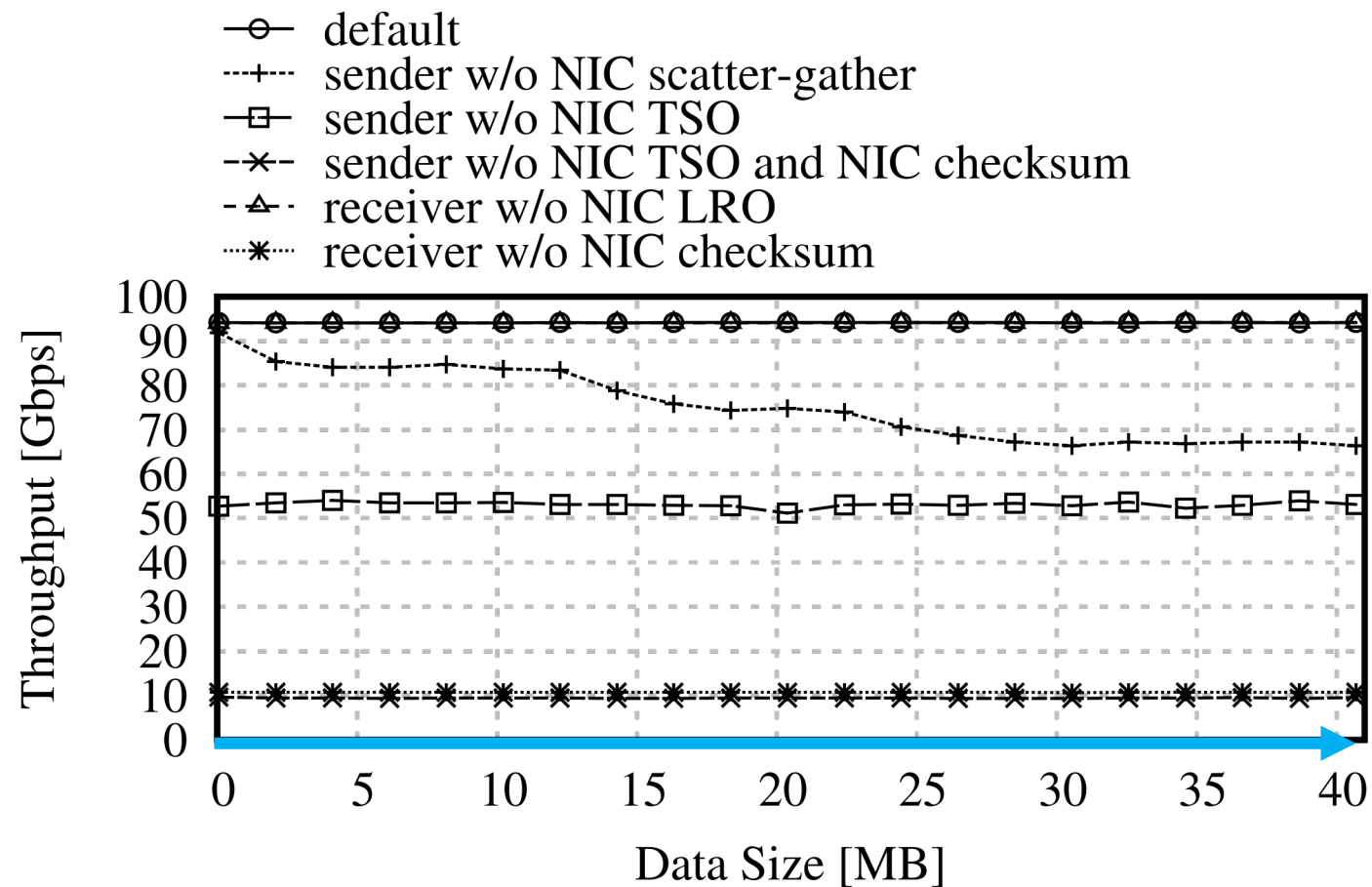
- 送信側は受信側へ同一のデータを繰り返し送信



開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

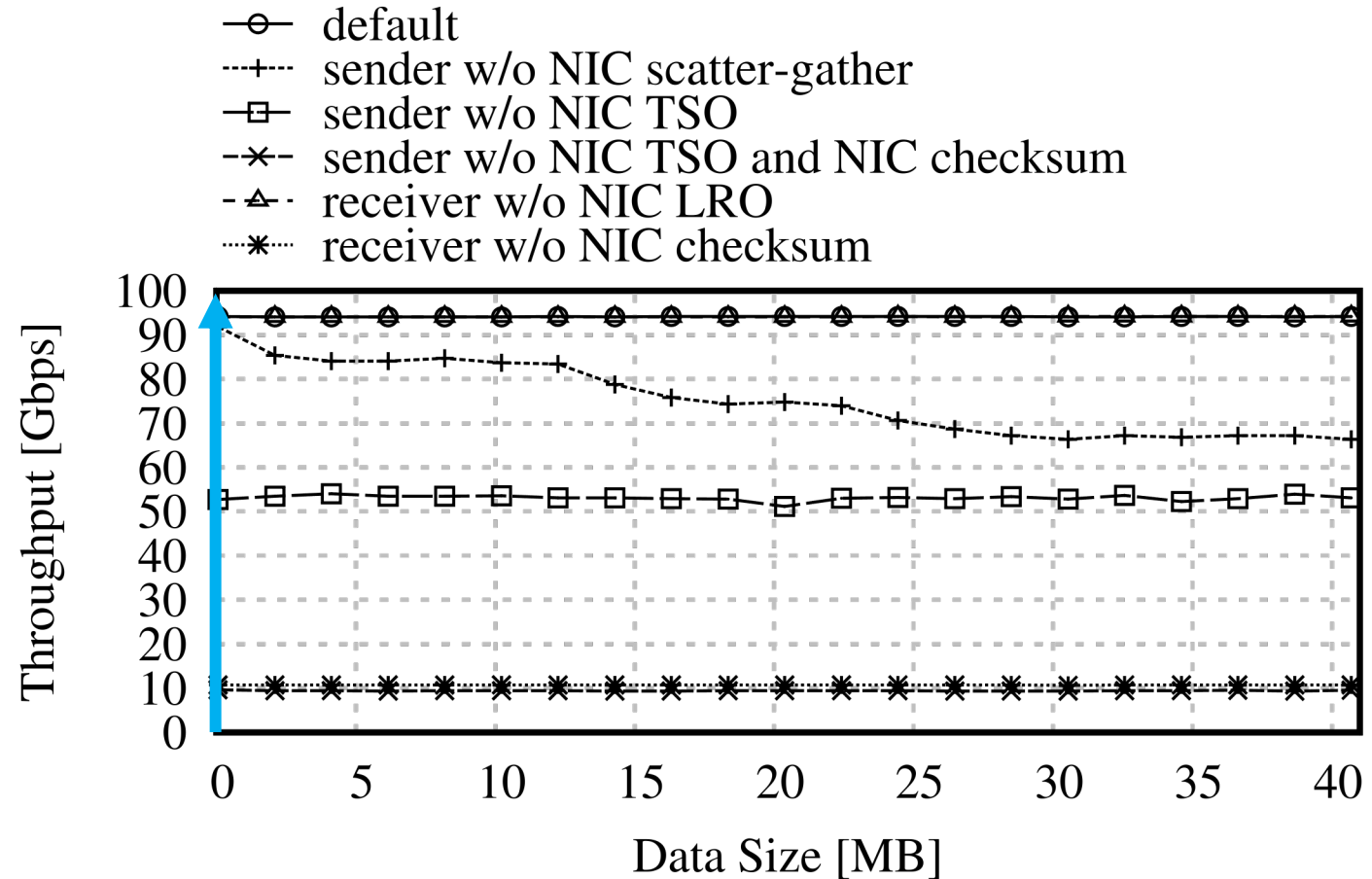
- 送信側は受信側へ同一のデータを繰り返し送信



開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

- 送信側は受信側へ同一のデータを繰り返し送信

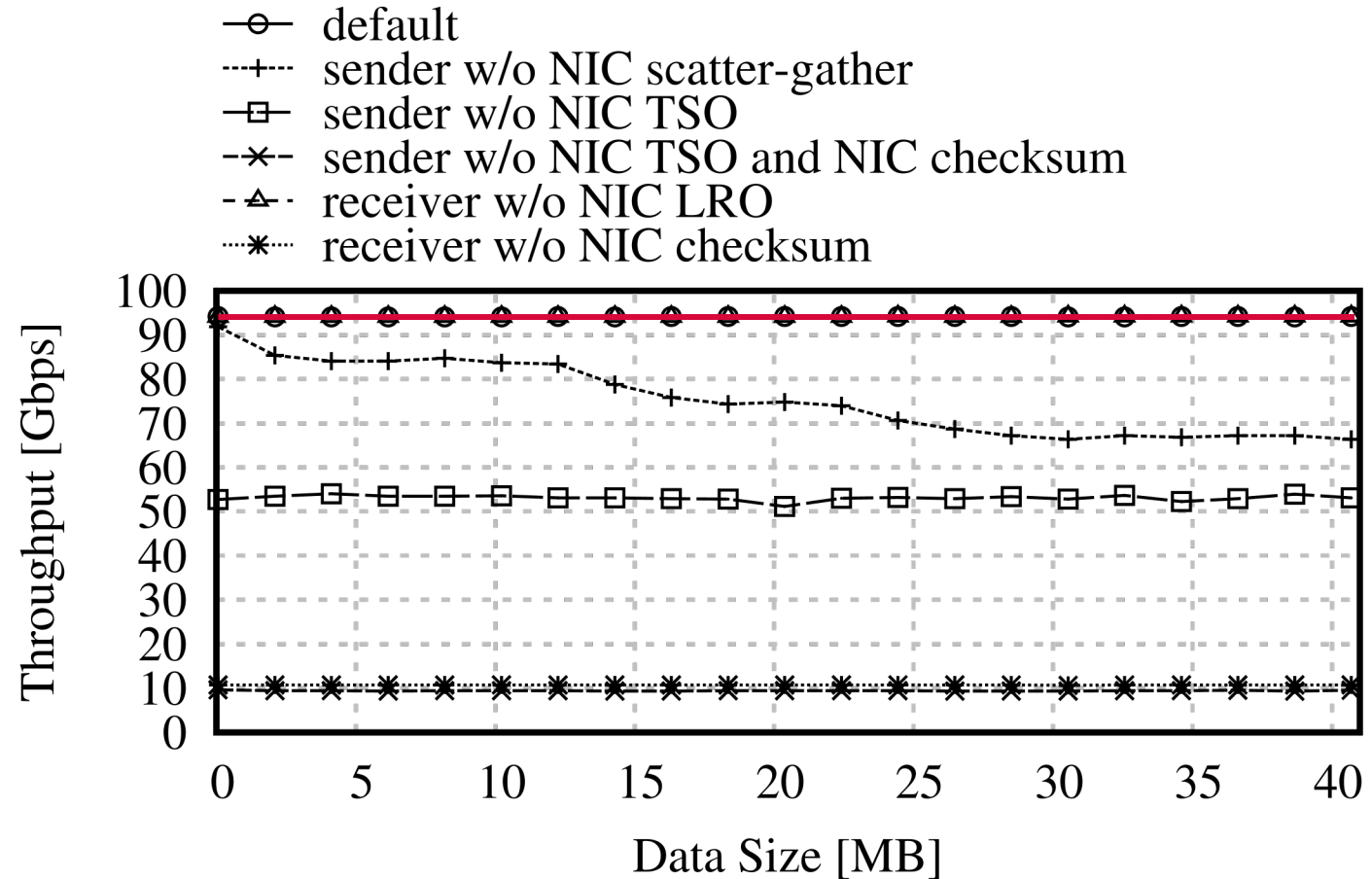


開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

- 送信側は受信側へ同一のデータを繰り返し送信

全てのオフロード機能と  
ゼロコピー送信が有効

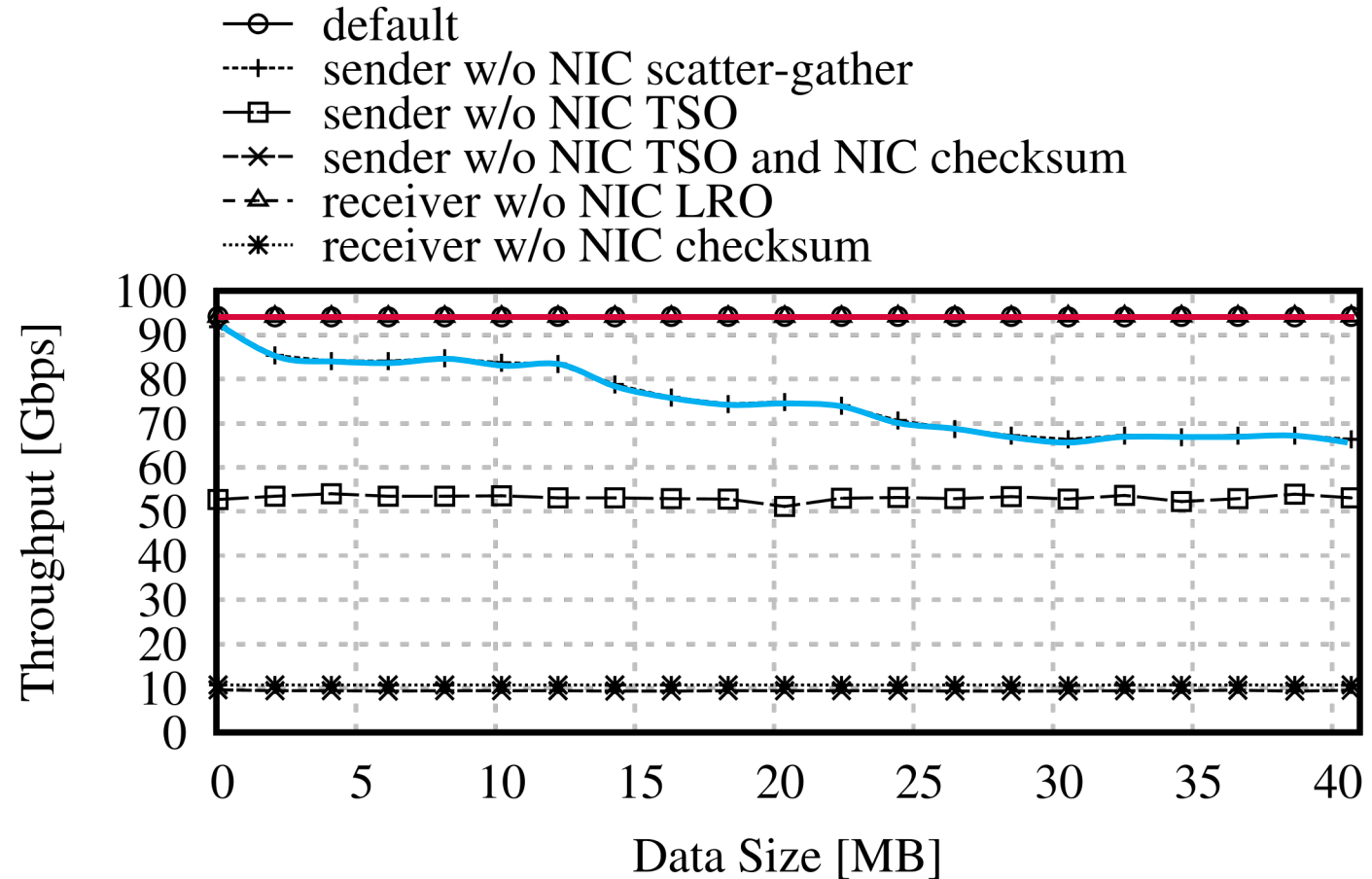


開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

- 送信側は受信側へ同一のデータを繰り返し送信
- 性能に寄与する要因
  - ゼロコピー送信

ゼロコピー送信を無効にすると  
送信データサイズに応じて  
スループットが低下



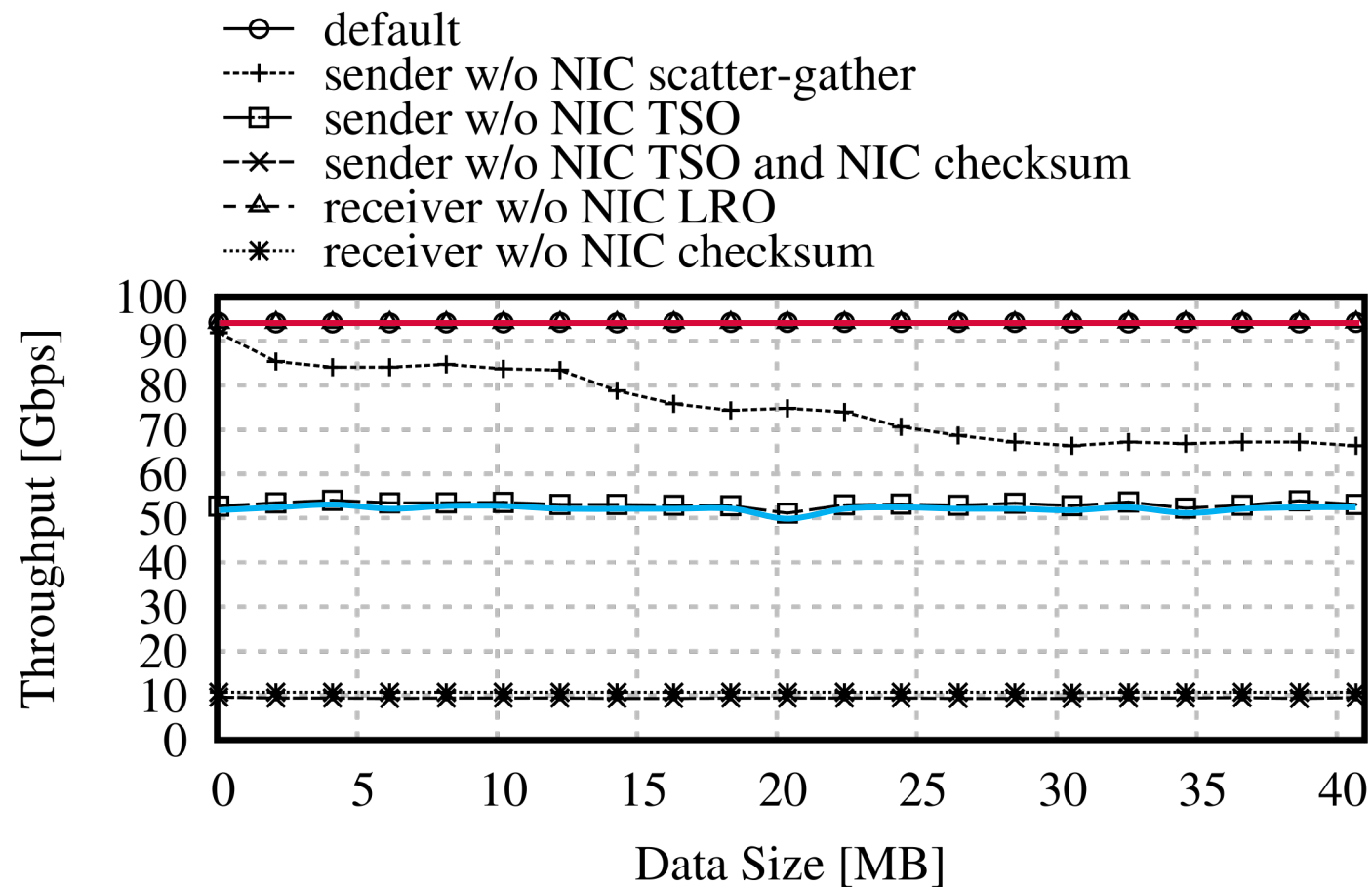


開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

- 送信側は受信側へ同一のデータを繰り返し送信
- 性能に寄与する要因
  - ゼロコピー送信
  - TSO

TSO を無効にすると  
スループットがほぼ半減

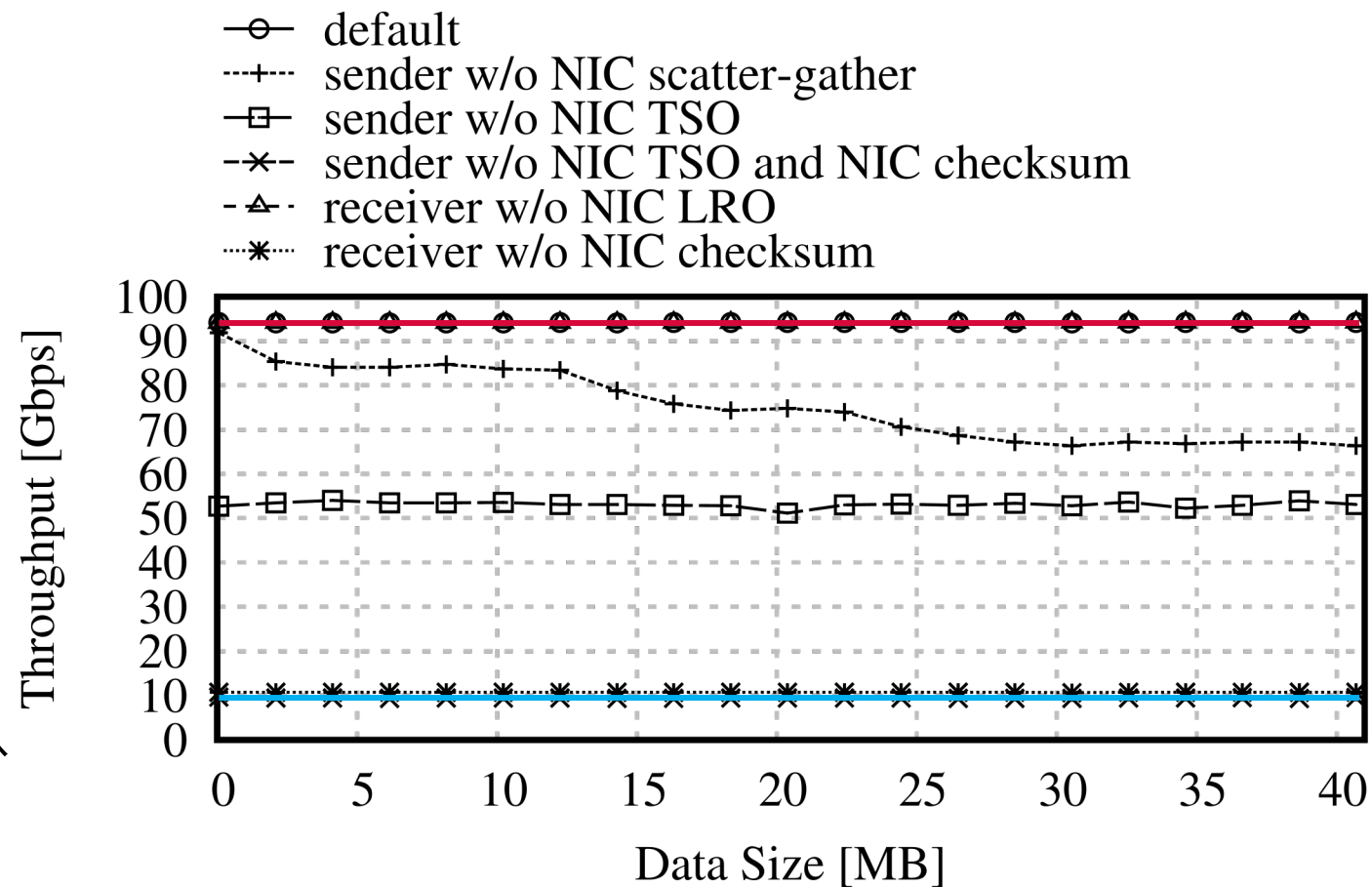


開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

- 送信側は受信側へ同一のデータを繰り返し送信
- 性能に寄与する要因
  - ゼロコピー送信
  - TSO
  - チェックサムオフロード

送信側で  
チェックサムオフロードを  
無効にすると  
スループットは約十分の一に減少

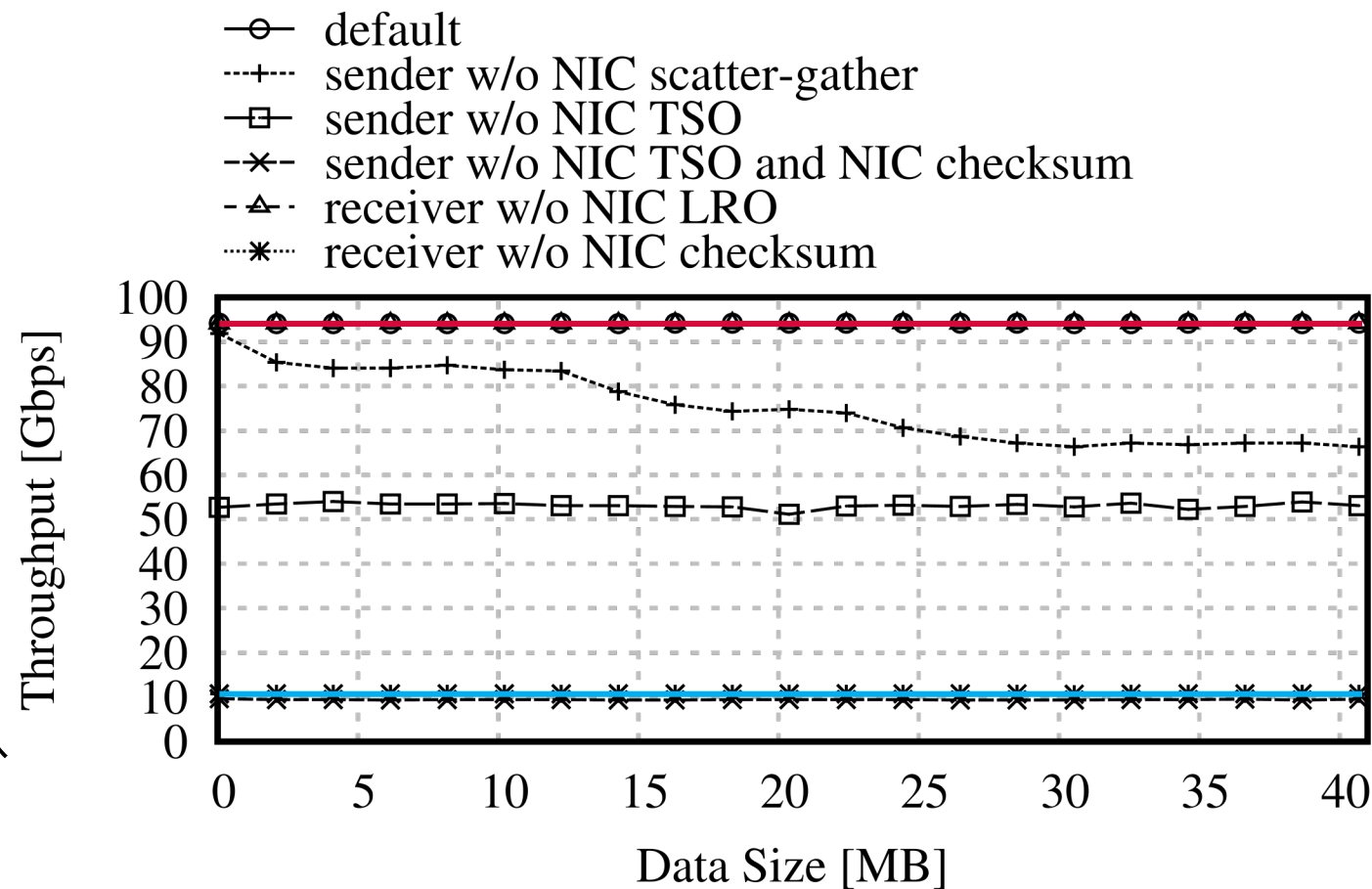


開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

- 送信側は受信側へ同一のデータを繰り返し送信
- 性能に寄与する要因
  - ゼロコピー送信
  - TSO
  - チェックサムオフロード

受信側で  
チェックサムオフロードを  
無効にした場合も  
スループットは約十分の一に減少

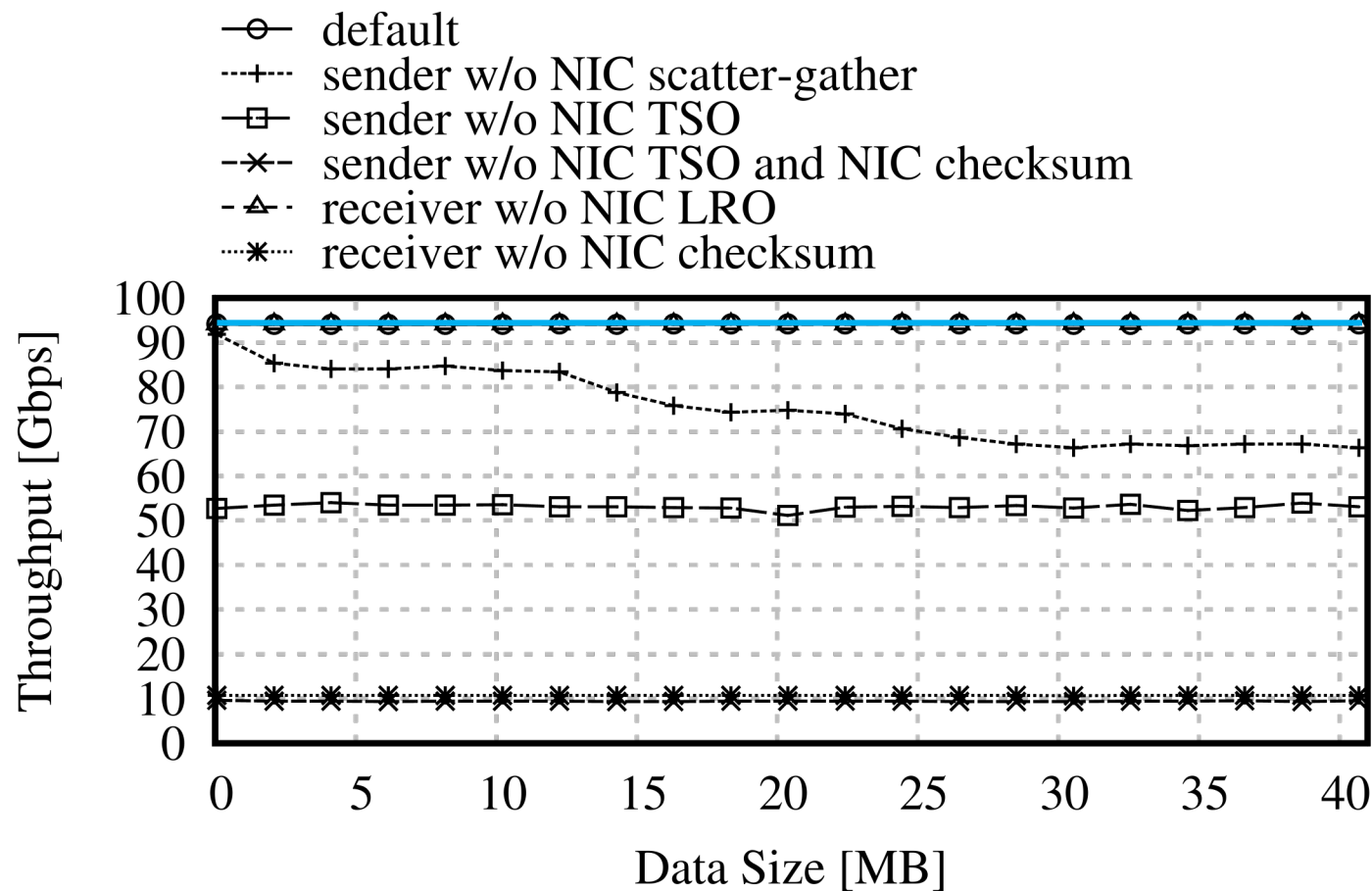


開発している TCP/IP スタック実装

# 評価：大きなサイズのデータの転送

- 送信側は受信側へ同一のデータを繰り返し送信
- 性能に寄与する要因
  - ゼロコピー送信
  - TSO
  - チェックサムオフロード

LRO による差は見られなかった  
注意：今回の計測において  
差が見られないというだけで  
意味がないということではない



開発している TCP/IP スタック実装

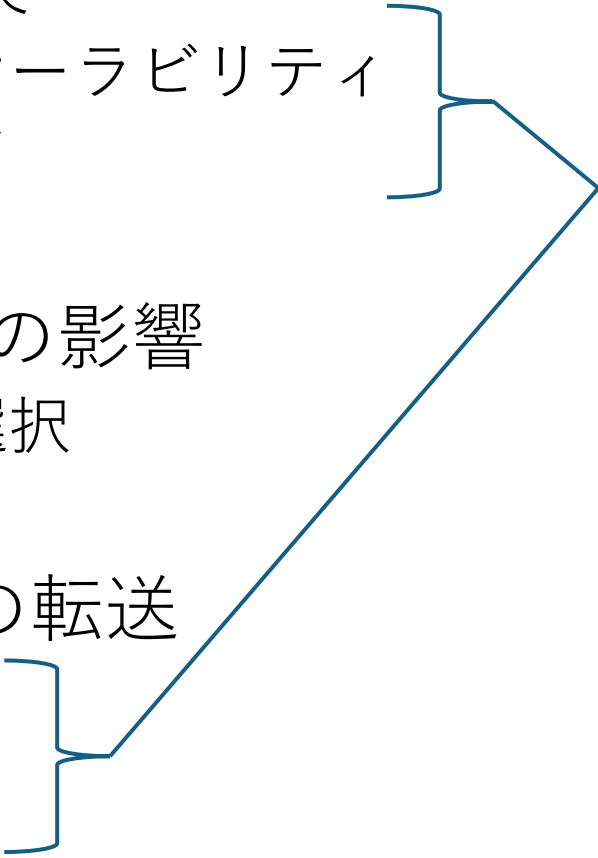
## 評価：性能に寄与する要因のまとめ

- 小さいメッセージの交換
  - マルチコア環境でのスケーラビリティ
  - チェックサムオフロード
- CPU コア割り当て方法の影響
  - 適切な割り当て方法の選択
- 大きなサイズのデータの転送
  - ゼロコピー送信
  - NIC のオフロード機能

開発している TCP/IP スタック実装

## 評価：性能に寄与する要因のまとめ

- 小さいメッセージの交換
  - マルチコア環境でのスケラビリティ
  - チェックサムオフロード
- CPU コア割り当て方法の影響
  - 適切な割り当て方法の選択
- 大きなサイズのデータの転送
  - ゼロコピー送信
  - NIC のオフロード機能



既存の可搬性に配慮した  
実装に不足している要素

開発している TCP/IP スタック実装

## 評価：性能に寄与する要因のまとめ

- 小さいメッセージの交換
  - マルチコア環境でのスケーラビリティ
  - チェックサムオフロード
- CPU コア割り当て方法の影響
  - 適切な割り当て方法の選択
- 大きなサイズのデータの転送
  - ゼロコピー送信
  - NIC のオフロード機能

既存の可搬性に配慮した  
実装に不足している要素

既存の性能に最適化された  
実装に不足している要素

# 表題への回答

Q. 何故今時 TCP/IP スタックを新しく実装しようとするのか

A. 既存の広く利用されている実装では比較的新しい  
通信ハードウェアの性能を活かすことが難しいから



# 表題への回答

Q. 何故今時 TCP/IP スタックを新しく実装しようとするのか

A. 既存の広く利用されている実装では比較的新しい通信ハードウェアの性能を活かすことが難しいから

- 既存のカーネル内の実装の変更の難易度を鑑みて新しく TCP/IP スタックを実装し直すことは現実的と思われる
- 既に高速な TCP/IP スタック実装はいくつか公開されているが広範な利用へ至るにはまだ解決されるべき課題がありそう
- その解決策を模索する取り組みの一つとして使いやすさに配慮した TCP/IP スタックを新しく実装しています