090109

A Protocol Description Language for Heterogeneous Multicore

要約

プロトコルを記述するためのプログラミング言語を提案

(プロトコル = TCP/IP 等ネットワークプロトコル)

要求

- C コンパイラが行わないような最適化を自動で行いたい

本研究の貢献

- プロトコル実装の新たなモデルを提案
  - プロトコル記述言語 + 動的言語
- 演算プロセッサの新たな応用
- 現実の well-known プロトコルに対する検討

背景 (1/2)

プロトコル処理における課題

課題：10GbE **等の高速ネットワークでは** CPU **がボトルネックに**

TCP **パフォーマンスは** 1bps/1Hz[Foong03]

↓

10Gbps **の** 1 **ストリームあたり** 10GHz **の** CPU **が必要**

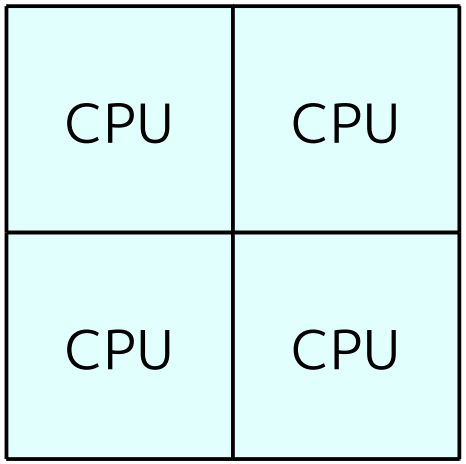
解決策

- CPU **の高速化 (すでに限界)**
- CPU **のマルチコア化 (× 電力効率)**
- **専用ハードウェア (× 柔軟性)**
- **ヘテロジニアスマルチコア** (New!)

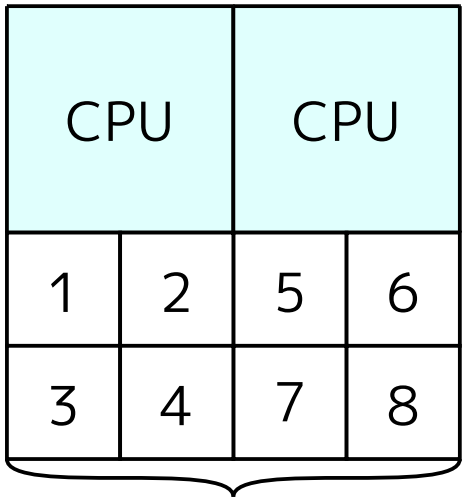
背景 (2/2)

ヘテロジニアスマルチコアとは

[Homogenious multicore]



[Heterogenious multicore]



- **機能を省略した小さいコアを多数搭載**
- **代表例**：Cell B.E.
- **主に演算に使用** (**先行研究で新用途を提案**)

先行研究

Cell B.E. による TCP/IP 実装 [Kawamura,HoTI08]

- 市販の TCP/IP スタックをそのまま移植
- 手動のデータ構造調整と、プロファイリングによる分岐予測
- 10GbNIC の直接制御で、SPE により 8GbpsTCP を実現

→ヘテロジニアスコアもネットワークプロセサとして有望

課題と本研究の提案

C 言語では Heterogeneous core の特性を十分に表現できない。

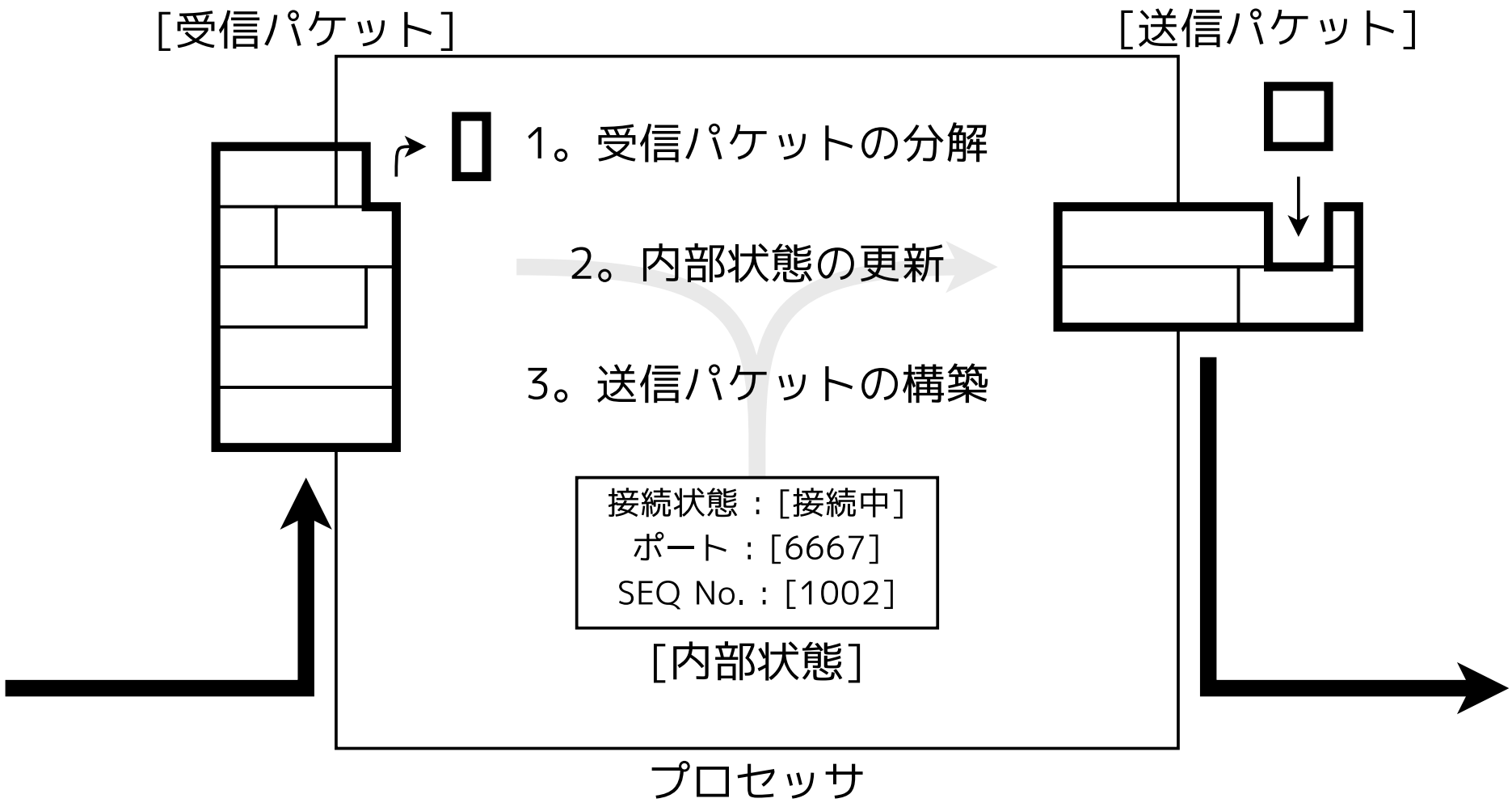
↓

専用のプログラミング言語を設計し、自動最適化

(先行研究：C 言語を使用し、手動とプロファイリングの mix)

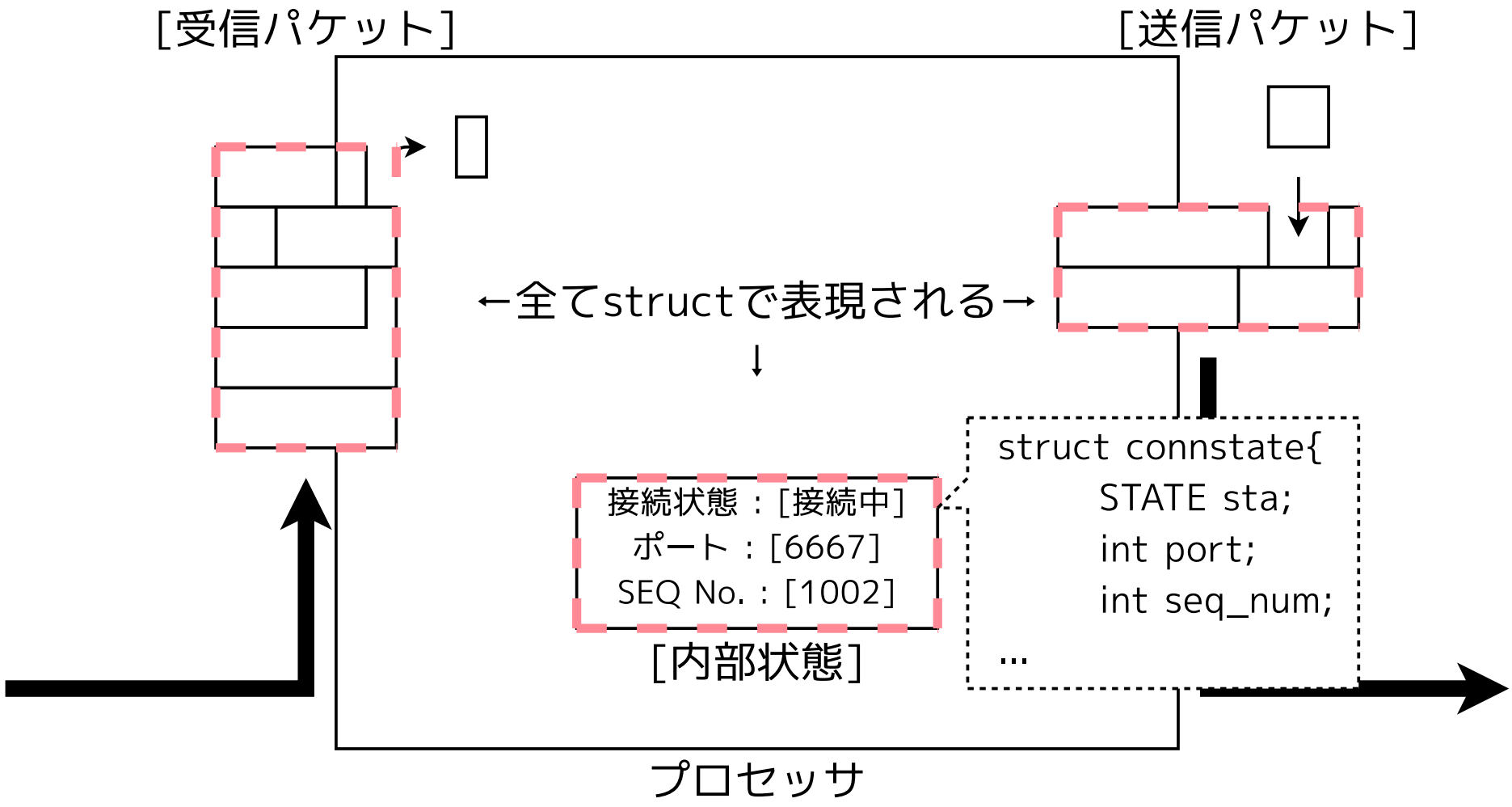
基本アイデア (1/3)

パケット処理の基本構造



基本アイデア (2/3)

C 言語での記述



基本アイデア (3/3)

`struct` はプロトコルの実現に不利 (表現力不足)

`struct` の問題点 (例)

- エンディアンを記述できない
- メモリ上の順序が固定されている
- 処理系間で互換性が無い
  - →ヘテロジニアスコアでより重要
  - (コアの種類ごとに異なる C コンパイラを使うため)
- ...

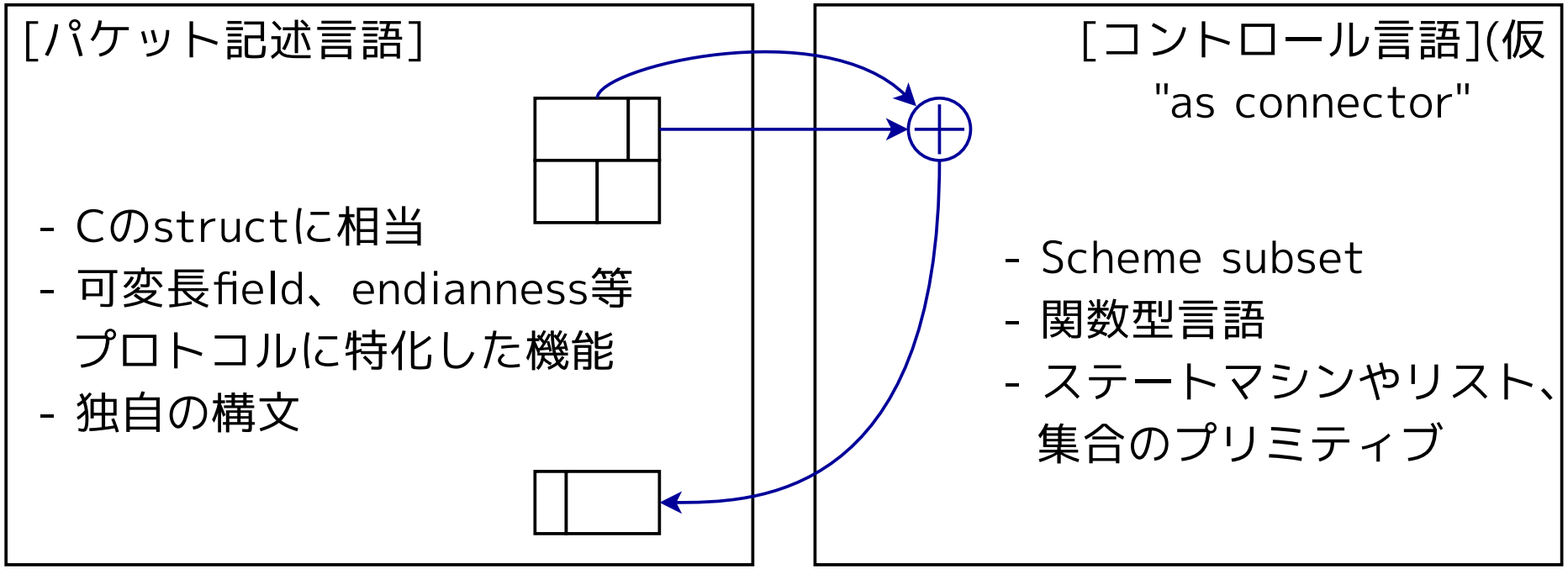
→ 独自のプロトコル記述言語を提案

提案するプロトコル記述言語

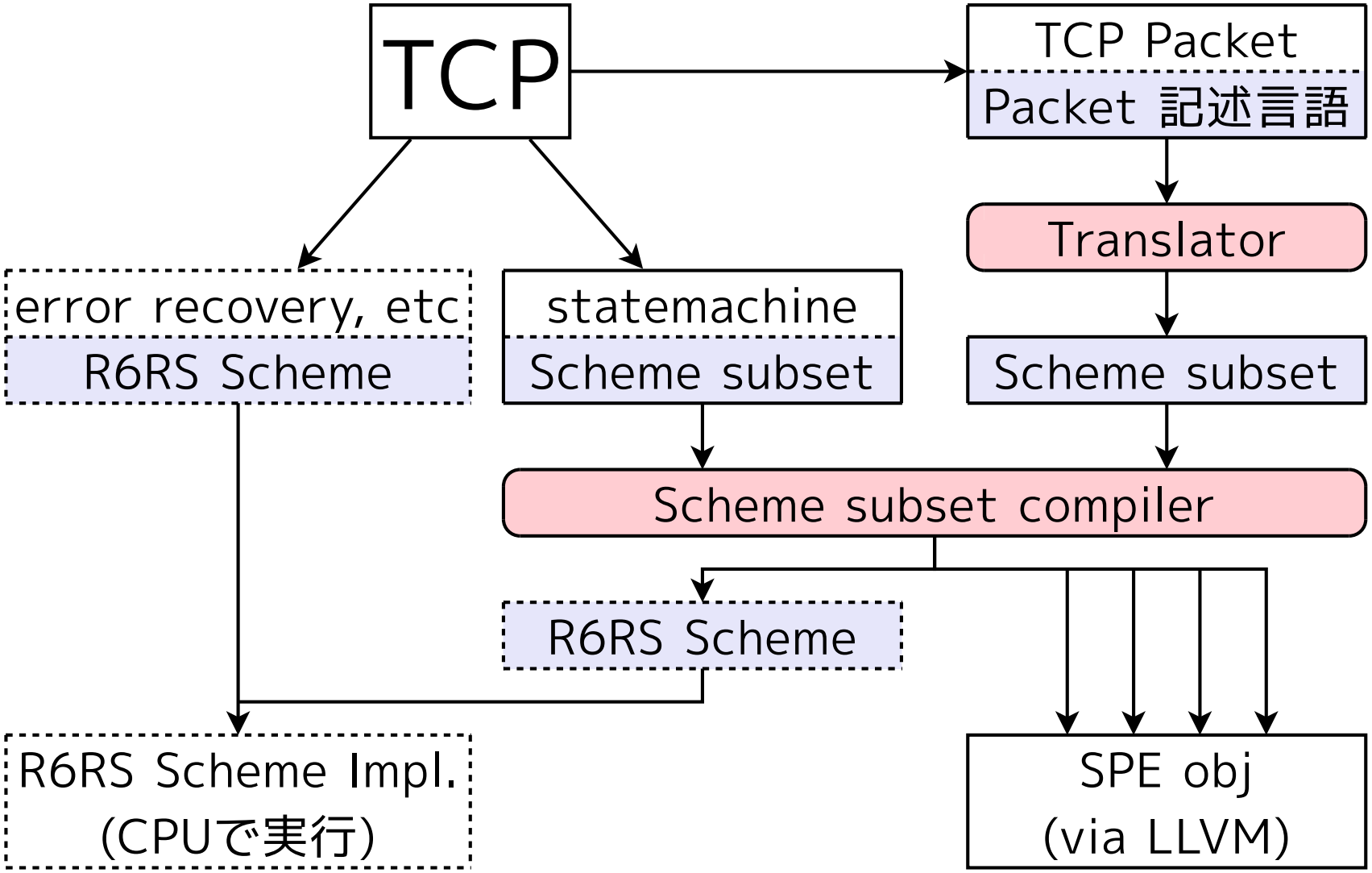
機能を限定したコンパクトな言語で高度な最適化を狙う

処理系の構成 (1/2)

プロトコル要素を 2 つに分割



処理系の構成 (2/2)–TCP を記述する場合



今後の予定 (てきとう)

Q1-Q2

- **最適化以外の完成**
- TCP/IPv6+SSH **の記述**
- R6RS Scheme **との連携**

Q3

- **最適化バックエンドの作成**
- **ベンチマーク**

余白

すごい余白

余白

すごい余白

ボツスライド

とても余白

専用言語の例

OpenCL(HP 向き言語)

- 演算タスク専用の C 言語バリアント
- SIMD プロセッサに有利なプリミティブ (型) を導入
- プロトコル処理に必要なプリミティブが存在しない

Preccs[服部 06]

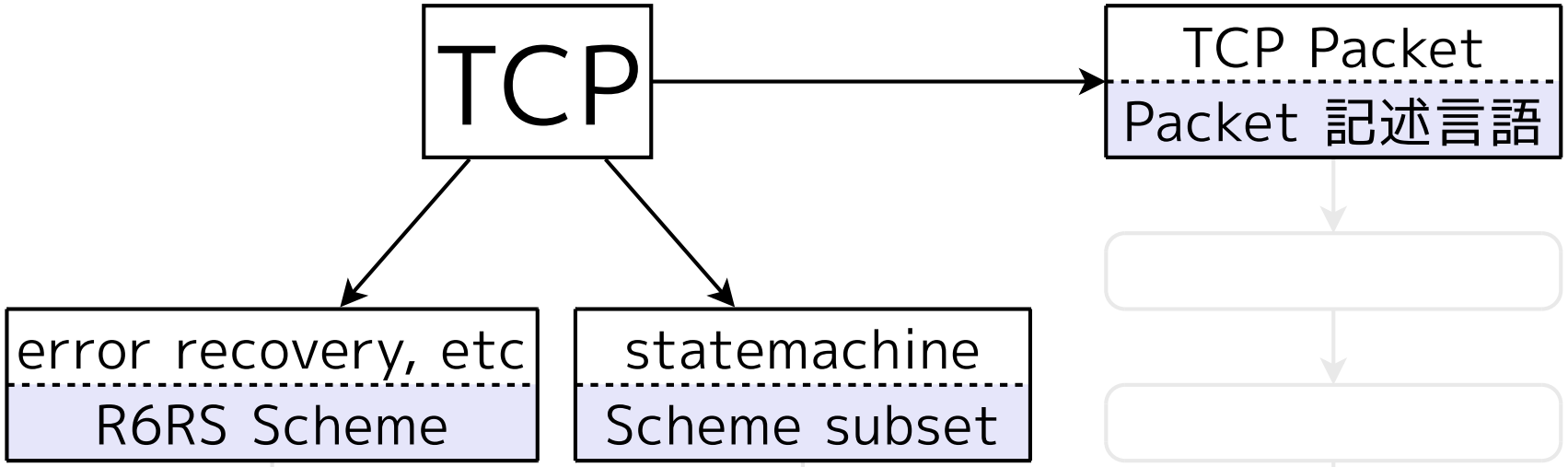
- プロトコル記述言語の一
- YACC 様のコードジェネレータ + ランタイム
- 通常の CPU をターゲットとしており HP での効率は不明

→ 本研究の目的に使えるものは知られていない

余白

すごい余白

先ほどの図の意味



TCPのプロトコル要素を、

- 構造体(パケットフォーマット)
- ステートマシンや他のプリミティブ
- エラー処理やアプリケーションとのI/F

に分割して記述

なぜ分割するの？

型の導入

- 構造体記述上のオブジェクトは全て型付き
  - コンパイラ (の最適化) で楽をするため
- S 式で型付きのオブジェクトを書くのが面倒
- (= syntactic sugar)

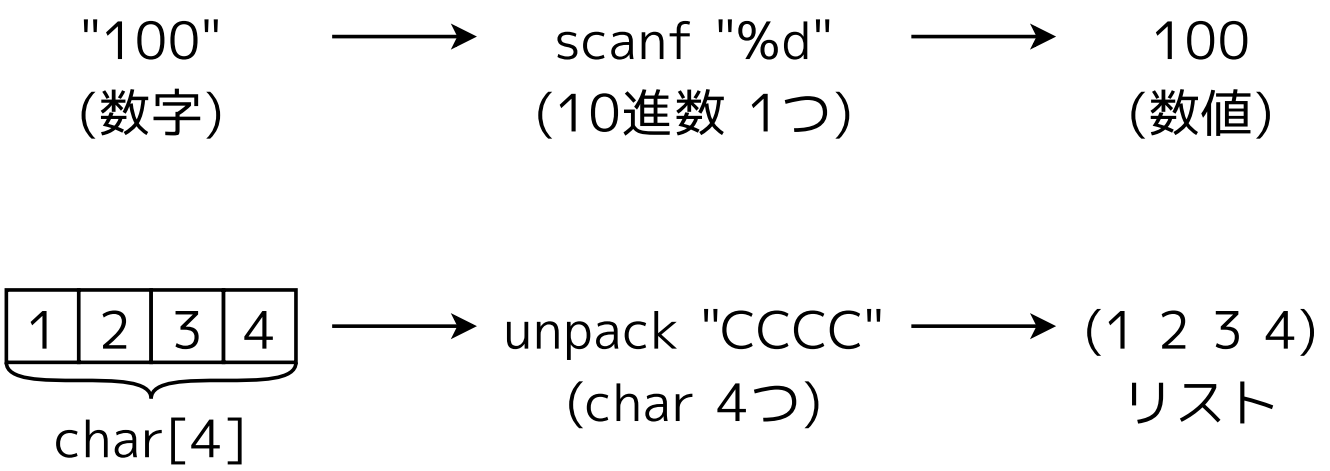
移植性

- 構造体記述だけからパケットアナライザ等が生成できる
- Preccs では無理。C 言語を実行しないといけないから。
- (自前で処理すれば良いが健全でない)
- C 言語のかわりに最低限のプリミティブを用いる。

先行技術： Pack と Unpack

(C の構造体は解説しなくてもいいよね)

- printf のバイナリ版。%d で数値が出るように、C で 1byte 書き出す
- Gauche ではこれらの変換ルーチンをクロージャにできる (これをやりたい)
- 既存のプロトコルも頑張れば書ける



OTL

Object Templete Language

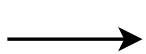
pack/unpack との差異

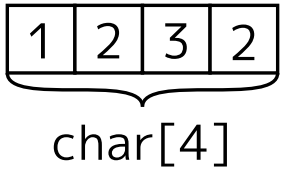
- **個々の要素に名前を付けられる** (C **の構造体と同様**)
- enum。
- **エンディアン記述** (C **に無い、**pack **に有る**)
- **式の束縛が可能** (**プログラマはチェックサムを埋めなくてよい**)

```
(set! [packet]/[A] 1)
(set! [packet]/[B] 2)
(set! [packet]/[C] 3)
```



```
*[packet] : [seq]
    [A] : [byte]
    [B] : [byte]
    [C] : [byte]
    [D] : [byte](add 1 [A])
```





構造体操作コードの生成

この研究のハイライトその 1。

生成ステップの詳細

- S 式に変換
- 1bit まで構造体を分解
- 必要な演算を型推論で導出 (= 論理合成)
  - この段階で 5bit-ADD のような bit 単位の演算が出てくる
- 必要に応じて内部データ構造を生成 (←ハイライト)
- 束縛された式を出力

内部データ構造と外部データ構造

この研究の中心アイデアその 0。未だに名称をサーベイ中。

- 内部データ構造：コンパイラが自由に設定できるデータ構造
- 外部データ構造：コンパイラが自由に設定できない〜
- (エッジデータ構造：プログラマが自由に設定できない〜)

要するに、実行時には全てが外部データ構造に変換される。

bit 精度構造体

この研究の中心アイデアその 1。

- 根拠レス (職人技としてはよく使われる)
- 通常の C 言語では 8bit 未満の幅の演算は扱わない
  - 通常は数値フォーマットが決まっているので十分
- 1word により多くの情報が入る
  - ステートマシンの実現や各種 enum で有利である可能性が有る
  - SIMD 演算器は幅が広いのでさらに重要
- 例 : 32 ステートのステートマシン = 5bit(でも `int` なら 32bit)

内部データ構造の生成

この研究の中心アイデアその 2。

- 内部データ構造から外部データ構造への変換
- 既存のコンパイラは無意識にやっている (レジスタ割りつけ等で表出)
- スタック中のデータ割り付けも？
- SIMD 演算器は一般にレジスタが多い (128 個とか)
  - greedy な方法はうまくいかない [捜す]
  - 1 オブジェクト = 1 レジスタでは効率が悪い?(packing)
- 本研究はこの部分のモデル化にとりくむ
- (生成したデータ構造を OTL に書き出すとデバッグに便利)

先行研究 (struct-reorg)

概要

C 言語の struct を安全に分解、再配置してキャッシュ効率を改善。

OTL 語で言うと...

- 既存の C の struct は全て外部データ構造
  - padding が可能であったりするが、順序は変えられない
- 一部を内部データ構造だと見なして外部データ構造を作りなおす

Scheme subset

Scheme とは基本的には名ばかり。なんか ML みたいに...

無いもの

- `cons`, `car`, `cdr`, ...
  - vector や list は OTL 側のプリミティブを使う
- `eval`, `call/cc`, ...
- GC
  - object の寿命は packet を dispose するまで
  - 安全かどうかはまだ不明

これは不便なので、既存の R6RS Scheme と連携したい。ただしアイデアなし。

GC やクロージャのやりとりで問題になりそう。