「ネットワーク・プロセッサのための オープンで高級で移植可能なプログラミング環境」 のための導入

金田 泰

電子情報通信学会の２つの研究会による合同研究会

情報ネットワーク研究会（IN）

[schedule] [select]

専門委員長　浅見 徹 (東大)
副委員長　小林 秀承 (NTT)
幹事　五十嵐 弓将 (NTT)，　荒井 大輔 (KDDI研)
幹事補佐　野村 啓仁 (NTT)，　大下 裕一 (阪大)

日時	2013年 7月18日(木)　10:45 - 18:20 2013年 7月19日(金)　09:45 - 16:55
議題	クラウドネットワーク技術、SDN、OpenFlow、プライベートネットワーク（VPN）、オーバーレイネットワーク・P2P、ネットワーク構成技術及び一般
会場名	北海道大学 工学部 アカデミックラウンジ３
住所	〒060-0814 札幌市北区北8条西8丁目
交通案内	JR札幌駅北口から徒歩10分 http://www.hokudai.ac.jp/introduction/campus/campusmap/

電子情報通信学会の 2 つの研究会による合同研究会

7月19日(金) 午後 NV 13:00 - 15:00		
(19)	13:00-15:00	◆第二種研究会 ネットワーク仮想化時限研究会(NV) プログラムは下記の URL を参照： http://www.ieice.org/~nv/%e7%a0%94%e7%a9%b6%e4%bc%9a/

第七回研究会プログラム
7月19日(金) 13:00-15:00

1. 「招待講演」 “SDNに対する通信事業者の取組状況と今後の期待”, ○佐藤陽一(NTTコミュニケーションズ)
2. “ネットワーク・プロセッサのためのオープンで高級で移植可能なプログラミング環境”, ○金田泰(日立製作所 中央研究所)
3. “Mobility support with Information Centric Networking: M-CATT”, ○ Eum Suyong, Kiyohide Nakauchi, Yozo Shoji, Nozomu Nishinaga(NICT), Masayuki Murata(Osaka University/NICT)
4. “大規模広域センサーネットワークの設計”, ○寺西裕一(NICT)
5. “ネットワーク抽象化モデルとSDN制御基盤への応用に関する一検討”, 小出俊夫・○森本昌治・芦田優太・下西英之(NEC 情報・ナレッジ研究

ネットワーク仮想化再説: 仮想化とは?

■ 仮想化とは, 物理的なコンピュータやネットワークがもつ機能とは質や量においてことなる機能を実現することである.

■ 仮想化の分類

◆質の仮想化

◆量の仮想化

●分割型の仮想化

●融合型の仮想化

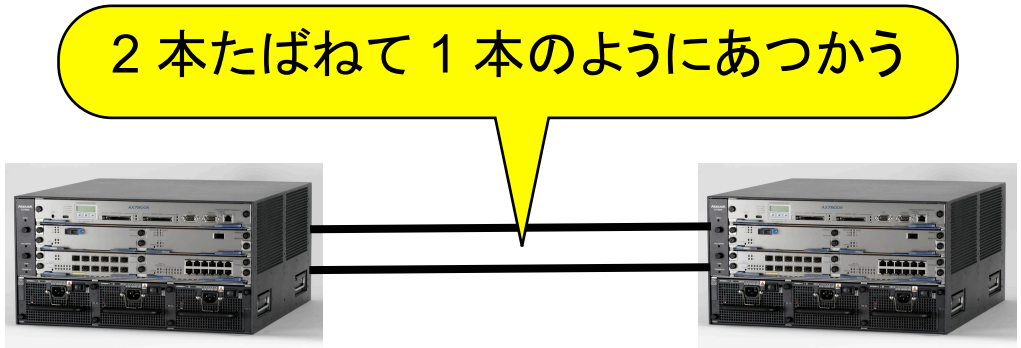
ネットワーク仮想化

■ 分割型仮想化の例

- VLAN
- VPN

■ 融合型仮想化の例

- リンク･アグリゲーション: 複数の物理リンクをたばねて，１個のリンクにみせる.

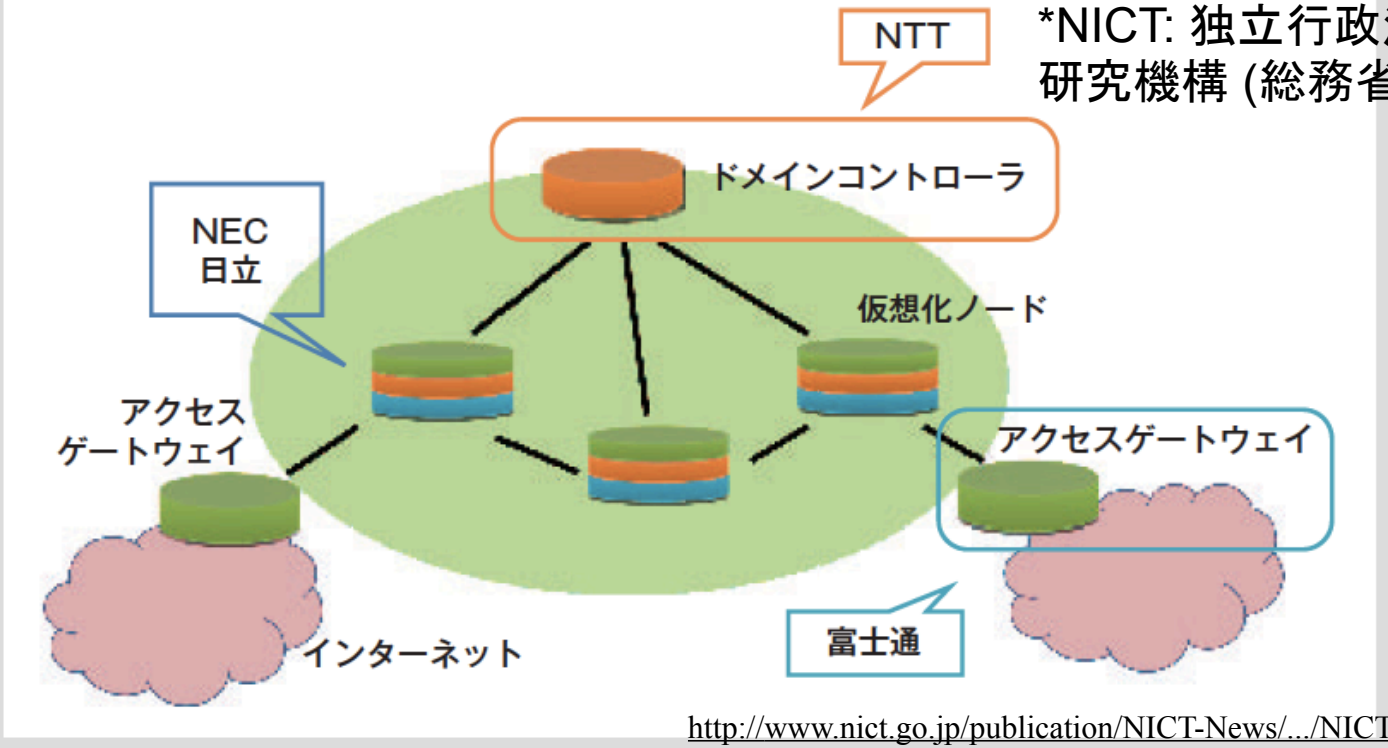


VNode -- 日本におけるネットワーク仮想化研究

- 日本の代表的なネットワーク仮想化研究プロジェクトとして「仮想化ノード (VNode) プロジェクト」(とその後継プロジェクト) がある.
  - VNode プロジェクト (2009-2010) では, NICT という場で東大, NTT, 富士通研究所, NEC, 日立が共同研究してきた.
  - 現在 (2011-2014) はその後継プロジェクトとして NICT の委託研究プロジェクトがすすんでいる.

ネットワーク仮想化基盤

- 仮想化ノードプロジェクトとその後継プロジェクトでは，仮想化ノードによって構成される「ネットワーク仮想化基盤」を開発してきた.



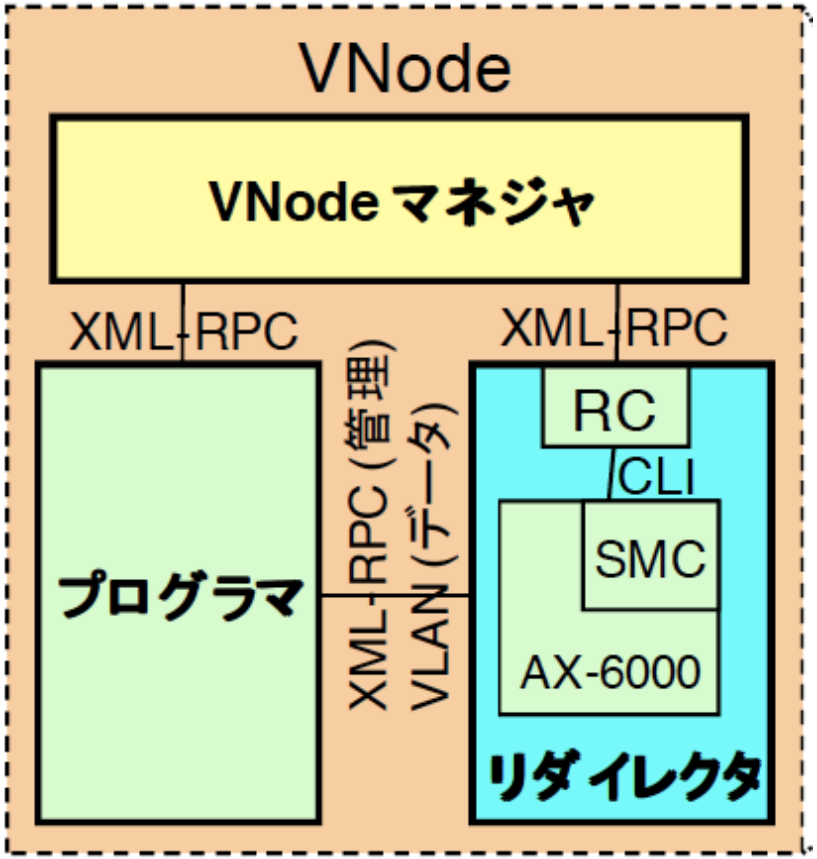
*NICT: 独立行政法人 情報通信研究機構 (総務省の外郭組織)

http://www.nict.go.jp/publication/NICT-News/.../NICT_NEWS_1006_J.pdf

図6●仮想化ノード・プロジェクトでの各社の分担

Deep programmability

- **仮想化ノードの重要な特徴は “deep programmability” (深いプログラム可能性)**
- **プログラマ (Programmer)**
  - パケットの加工や転送先の決定などの処理をおこなう (計算 / ストレージ・リソースをもつ).
  - パケットの処理を**プログラム**できるので 「プログラマ」とよぶ.
- **リダイレクタ (Redirector)**
  - パケットを他の VNode 等から受信してプログラマに転送し,
    プログラマからのパケットを他の VNode などに転送する.
  - リダイレクタがおこなうパケット形式の変換なども**プログラム**可能にしようとしている.

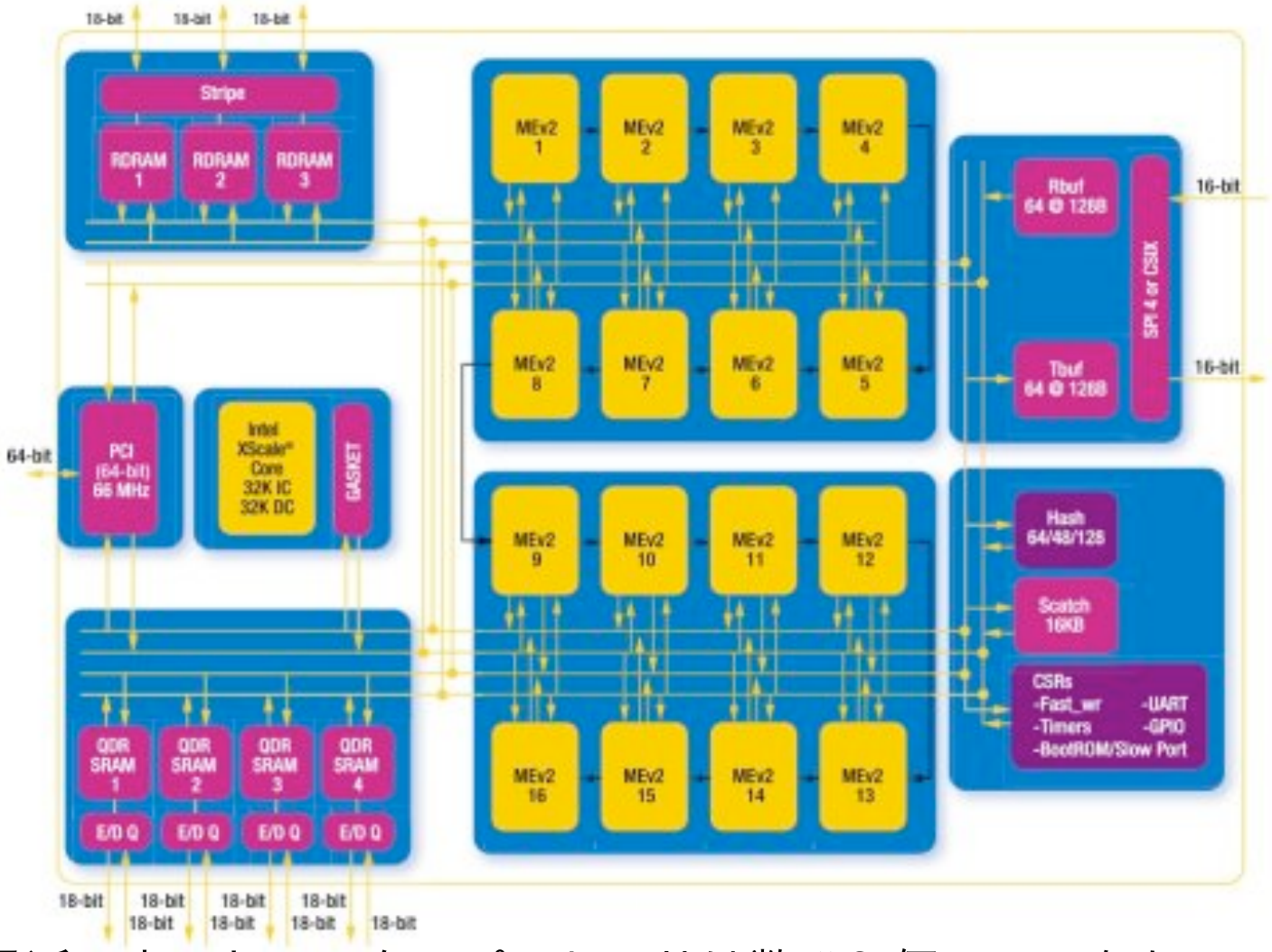


Deep programmability 実現手段としてのネットワーク・プロセッサ

- **従来のハードウェアの問題点**
  - **ネットワークはかたい**: 高速処理のために特殊化されたハードウェアを使用していて，あたらしいプロトコルが導入できない.
    - ASIC (application specific integrated circuit): 特定のプロトコル処理だけを高速に実行する LSI
  - **ソフトウェア処理はおそい**: ネットワークをやわらかくするためにソフトウェア処理を導入すると，高速処理できなくなる.
    - Intel x86 などの汎用 CPU とはネットワーク・データ処理向きでない.
    - Linux, Windows などの OS もネットワーク・データ処理向きでない.
      - Linux はネットワークの制御 (ルーティングなど) には適している.
- **この問題点の解決のためにネットワーク・プロセッサが開発された.**
  - ネットワーク・プロセッサはネットワーク処理向きのハードウェアであり，かつプログラマブル (deeply programmable) である.

ネットワーク・プロセッサの基本構造

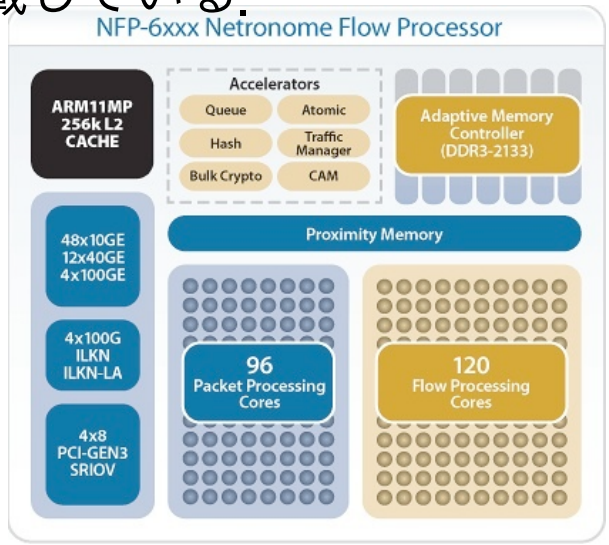
- 多数のパケット処理用プロセッサ・コアを搭載している.



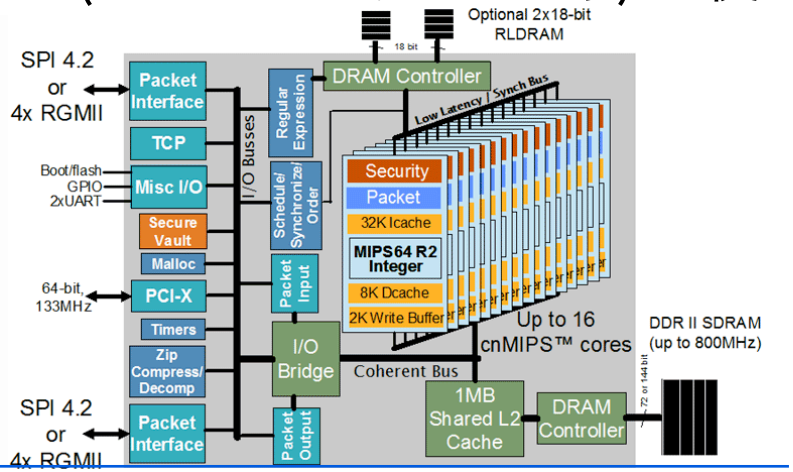
- 最近のネットワーク・プロセッサは数 10 個のコアをもつ.

さまざまなネットワーク・プロセッサ

- **Intel の IXP, Netronome の Flow Processors**
  - 機能を限定した小規模のコアを多数搭載している.

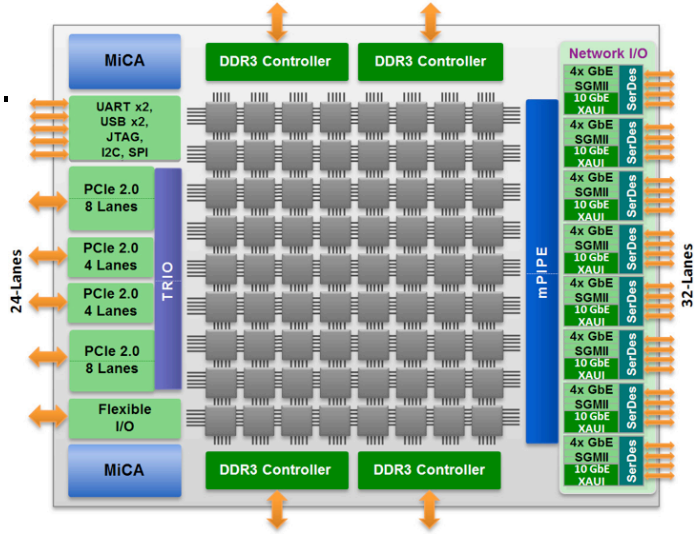


- **Cavium Octeon**
  - ネットワーク向きに拡張した汎用 CPU (MIPS アーキテクチャ) を使用している.

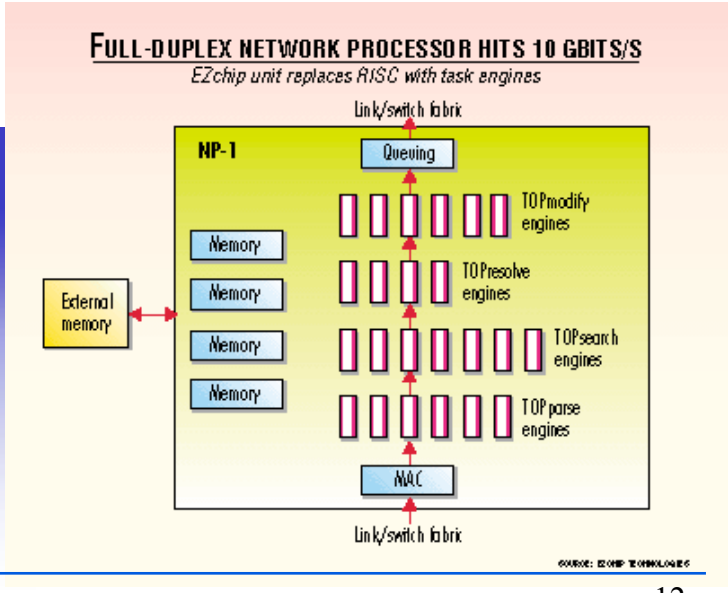


さまざまなネットワーク・プロセッサ (つづき)

- Tilera TILE Processors
  - 汎用 CPU コアを格子状にならべている.



- EZchip NP
  - 4 種類以上の専用化されたコアによって構成される.



ネットワーク・プロセッサのための
オープンで高級で移植可能なプログラミング環境

金田 泰

解決するべき課題

- **ネットワーク仮想化基盤の “deep programmability” の実現のため，ネットワーク・プロセッサが重要.**
- **ネットワーク・プロセッサによるプログラム開発の問題点**
  - **移植性がない** ← 低水準，ハードウェア依存，NDA (秘密保持契約) 依存のプログラミングが必要.
  - **開発者が限定される** ← プログラミングに特殊技術がもとめられ，必要な知識がひろく流通していない.
  - **コスト高** ← 開発者の限定，特殊技術，技術習得に時間がかかる.
- **ハードウェア依存性のひとつである，高速なパケット処理においては SRAM, DRAM のつかいわけに焦点をあてる.**
  - 大量のデータは DRAM に格納する必要があるが，DRAM 上のデータの操作はワイヤレート処理を不可能にすることがある.
  - 従来はプログラマがつかいわけを意識する必要があり，プログラミングを困難にしていた. (Intel IXP 等のキャッシュをもたないアーキテクチャだけでなく，Octeon のようにキャッシュがあっても意識が必要なことがある.)

課題解決のための提案

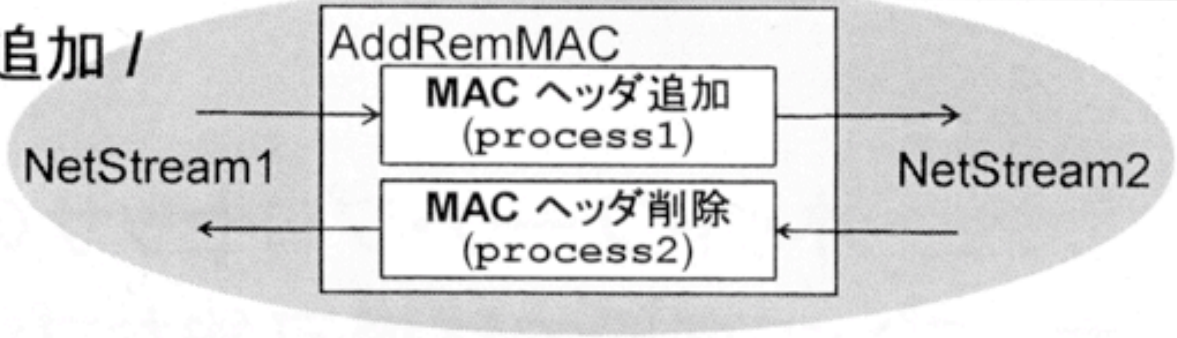
- **ネットワーク仮想化基盤においてこれらの問題を解決するために 2 つを提案する.**
  - オープン, 高級, 移植可能なプログラミング言語 CSP
    - オープン = NDA なしにネットワーク・プロセッサがプログラム可能
    - CSP = Continuous Stream Programming
  - CSP 実装におけるパケットの 4 つの表現形とそれを自動的につかいわける方法
    - SRAM/DRAM のつかいわけを, プログラマが意識せずに, ハードウェアやその専用ソフトウェアなどに依存せずに実現するため

パケット処理言語 CSP の設計方針

- **Java にちかい構文・意味を採用する.**
  - Java や C++ のプログラマがあつかいやすいようにした.
- **パケットは immutable なバイト列とする.**
  - パケットは可変長なので文字列, バイト列と同様にあつかう.
    - concatenation, substring などが基本演算.
  - Java とちがってプロトコル処理もバイト列操作として記述する.
  - Java と同様にパケットは immutable (かきかえられない).
- **パケットとバイト列はことなる型のオブジェクトとし, それぞれに適した実装をする.**
  - これらは実際のプログラム上でのあつかいがおおきくちがうので, バイト列型 String とはことなる Packet 型を新設する.
  - パケットは基本的には先頭だけがキャッシュされ, 後続部分は DRAM にだけある (キャッシュできない) とみなす.
    - 後続部分 (ペイロード) は処理せずに出力すると仮定する.
  - パケットに関する 2 種類の部分列生成を明確に区別する.
    - subpacket はパケットを生成し, substring はバイト列を生成する.

CSP によるプログラムの例

■ 単純な MAC ヘッダ追加 / 削除のプログラム例




```
001 import NetStream1;
002 import NetStream2;
003 class AddRemMAC {
004    NetStream out1;
005    NetStream out2;
006    public AddRemMAC(NetStream<Port1> process1,
007                     NetStream<Port2> process2) {
008       out1 = port1;
009       out2 = port2;
010    }
011    void process1(Packet i) {    // Port 1 to 2 (no VLAN -> no VLAN)
012       Packet o = new Packet(i.substring(0,14),i); // MAC header of original packet
013       out2.put(o);
014    }
015    void process2(Packet i) {    // Port 2 to 1 (no VLAN -> no VLAN)
016       Packet o = i.subpacket(14); // remove MAC header (no VLAN)
017       out1.put(o);
018    }
019    void main() {
020       new AddRemMAC(new NetStream1(), new NetStream2());
022    }
023 }
```

2 個のパケット入出力ストリーム

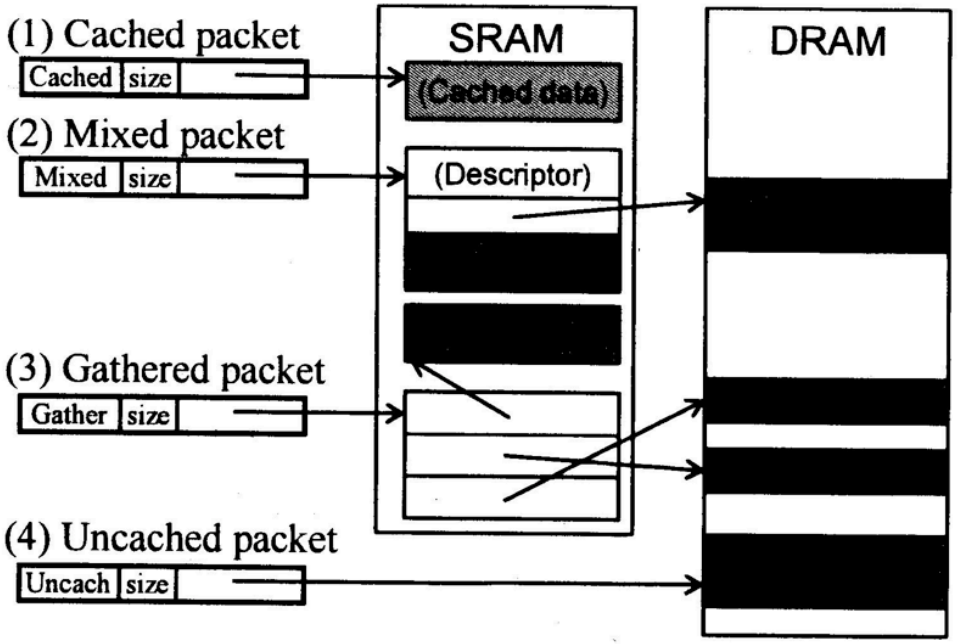
port1 への入力は process1 にわたす

port2 への入力は process2 にわたす

パケット処理オブジェクト (singleton) を生成する

CSP の実装: Packet 型の 4 つの表現形

- **パケットというデータ型に 4 つの表現形をもうけた.** (プログラマが意識しなくても SRAM と DRAM がつかいわけられるようにするため).
  - **Cached**: パケット全体が SRAM 上にある.
    - コピーが DRAM 上にあることは仮定しない.
  - **Mixed**: パケット先頭が SRAM 上にあり，パケット全体が DRAM 上にある.
    - 先頭のバイト数は可変.
  - **Gathered**: パケットが複数の部分から構成される.
    - 各部分は不連続領域にある.
    - 要素の配列 / linked listによって表現される.
  - **Uncached**: パケット全体が DRAM 上にある.
    - コピーが SRAM 上にあることは仮定しない.

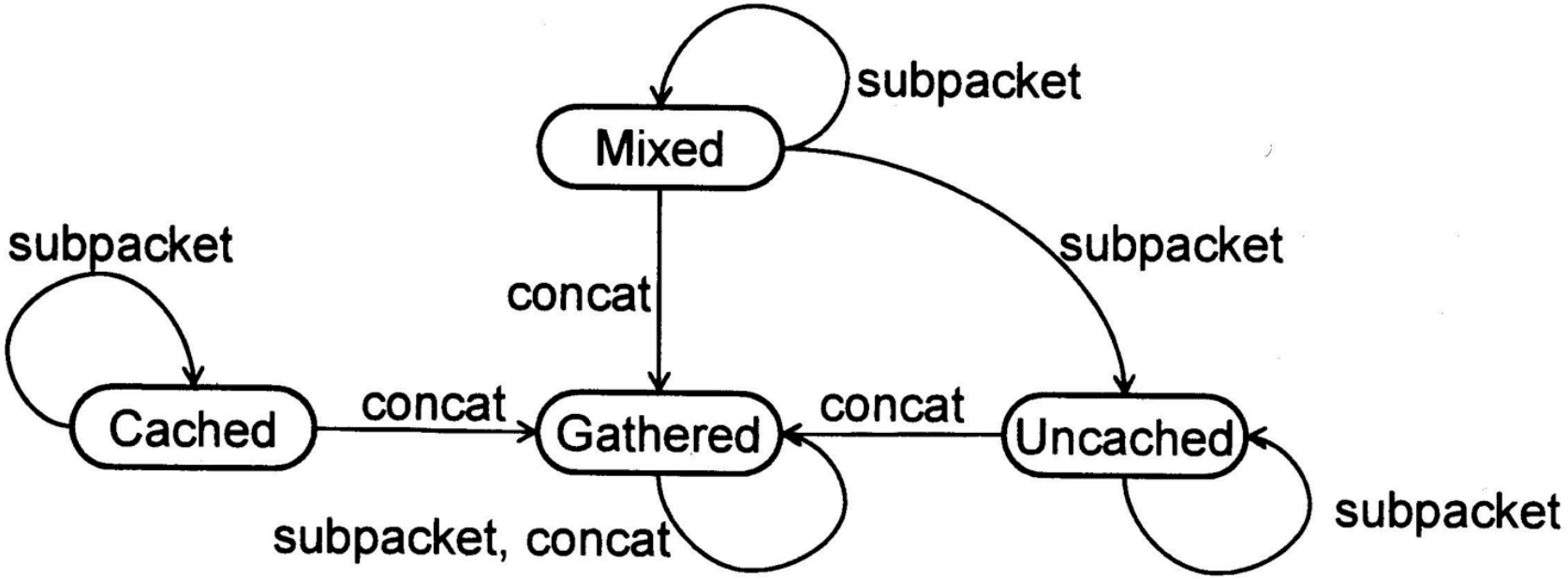


CSP の実装: Mixed 形式と gathered 形式の必要性

- **Mixed 形式が必要な理由**
  - パケット処理においては入力されたパケットのヘッダだけを加工して出力することが多い.
  - このような場合はパケット入力時にパケット先頭部分だけを SRAM に格納し，のこりを DRAM に格納することが有効である.
- **Gathered 形式が必要な理由**
  - DRAM あるいは SRAM に格納された複数のデータ (列) を連接してパケットを生成するときに必要 になる.
  - 全データをあらかじめ DRAM の連続領域におこうとすると，データを DRAM から DRAM にコピーする 必要が生じる.

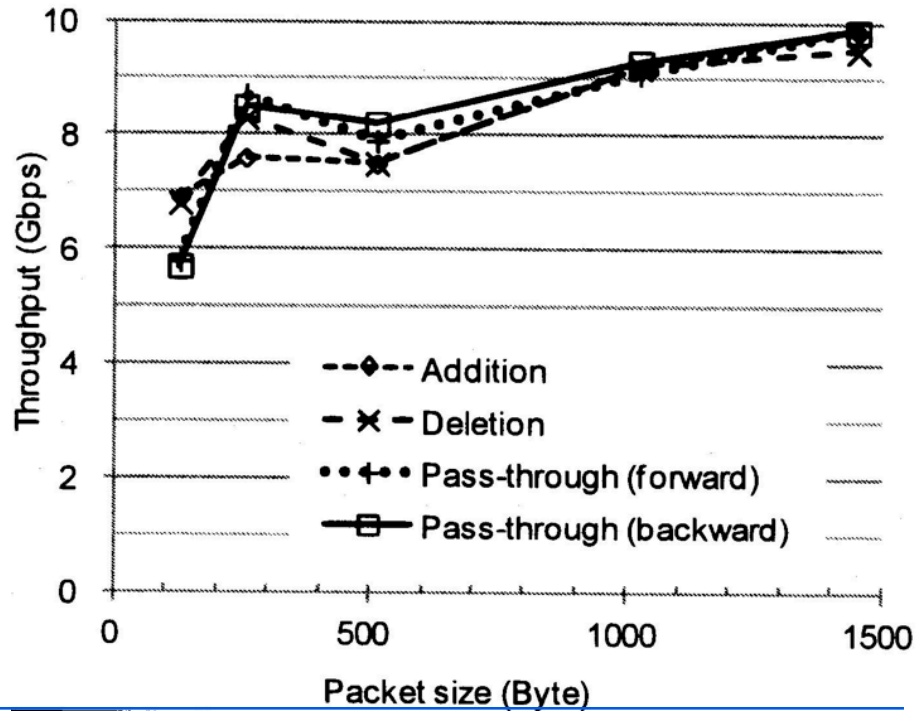
CSP の実装: パケット・データ表現間の状態遷移図

- データ表現ごとにわけてパケット操作を処理する.
- パケットからパケットへの操作によってデータ表現が変化するので, データ表現間の状態遷移図がえがける.
  - エラーが発生するケースもあるが, この図では省略している.



プロトタイピングと評価

- **プロトタイプ**
  - プログラミング環境 +Net を試作し，そのうえに Octeon のための CSP 処理系のプロトタイプ +Net CSP を開発した.
- **2 つの CSP プログラムによる評価**
  - MAC ヘッダ削除 / 追加する AddRemMAC と，パケットを加工せずに通過する Pass-through を使用して評価した.
  - ネットワーク仮想化基盤の一部であるネットワーク収容装置に Octeon ボード (GE 社 WANic-56512) を接続して評価した.
- **性能測定の結果**
  - パケットサイズが 256 バイト以上のとき，ワイヤレートにちかい 7.5 Gbps をこえるスループットが実現されている.



結言

- **まとめ**
  - **ネットワーク・プロセッサのためのオープンで高級で移植性のあるプログラミング言語 CSP とその実装法を提案した.**
    - CSP を使用すると，プログラマは SRAM と DRAM のくべつを意識せずにそれらをつかいわ けることができる.
    - 言語実装にあたってパケットというひとつのデータ型の 4 つのデータ表現をつかいわけることを提案した.
  - **Octeon のための CSP 処理系のプロトタイプを開発・評価した結果，7.5 Gbps をこえるスループットをえた.**
    - 単純なパケット・ヘッダ追加 / 削除の CSP プログラムなどにおいて，パケット・サイズが 256 バイト以上のときの測定値.
- **結論**
  - この結果はこの方法がネットワーク・プロセッサのプログラミングを容易に低コストにするために 有望なものであることをしめす.
  - ネットワーク仮想化基盤 (JGN-X 等) にそのプログラミング機能を提供できる可能性がある.

付録: CSP の設計: パケット操作*

- **パケット入出力**
  - パケットが入力されると，指定されたメソッドがよびだされる.
    - void process1(Packet i), void process2(Packet i)
  - put メソッドによって出力する.
    - out2.put(o), out1.put(o)
- **部分列生成**
  - 生成されるのがパケットかどうかをプログラマは意識する必要がある.
  - プログラマはパケットが SRAM にあるか DRAM にあるかを意識する必要はない.
  - 効率的に処理できないケースではエラーが発生することがある.
    - Packet Packet.subpacket(from); -- 指定範囲がすべて SRAM 上だけにあると実行時に判定されたとき，エラーと判定
    - String Packet.substring(from [, to]); -- 指定範囲にキャッシュされていない (DRAM 上だけにある) データがふくまれるときエラーと判定
    - String String.substring(from [, to]); -- 指定された範囲がすべて SRAM 上にあるとき以外はエラーと判定する.

付録: CSP の設計: パケット操作 (つづき)*

- **列の連接**
  - new Packet(String, …, String, Packet);
  - String String.concat(String);
- **列の生成**
  - new Packet(String);
  - 構築子と同様に複数個の引数を許容することも可能である (構築子とのちがいは, すべての引数が String だということ).
- **列-整数 間変換**
  - String.itos(ivalue, length)
  - Integer.stoi(svalue)