Controlling Network Processors by using Packet-processing Cores

Yasusi Kanada
Hitachi, Ltd., Japan

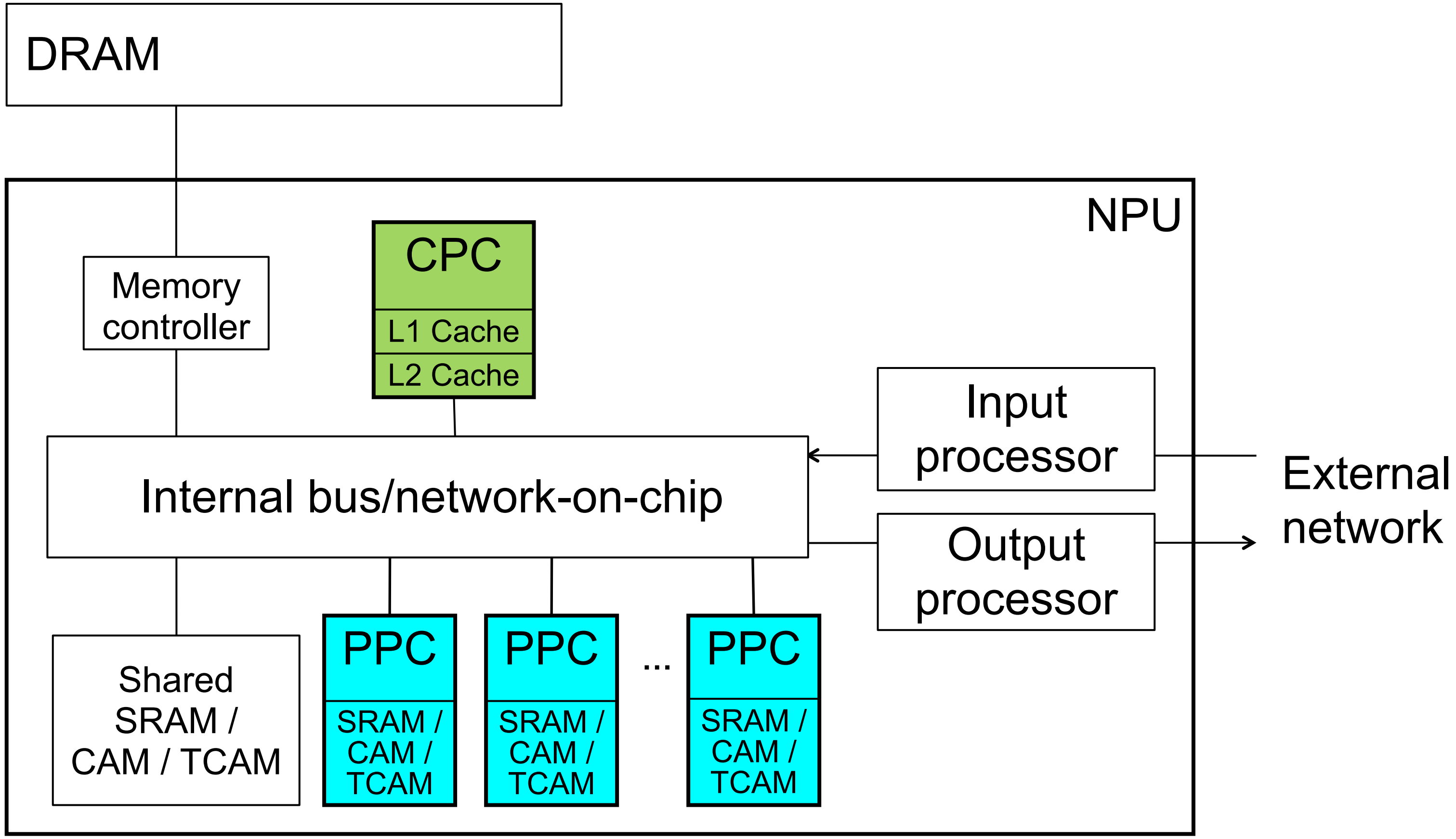
Introduction

- **Network processors (NPs) are used for customizable and high-performance networking.**

- **NPs usually contains two different types of cores.**
  - Packet processing core (PPC)
  - Control processing core (CPC)
- **Problems of NP programming**
  - Synchronization and communication of PPCs and CPC
  - Hardware- and vendor-dependence of NP software
  - Lack of portability of NP software
- **A method for solving these problems are proposed in this study.**

NP Architecture

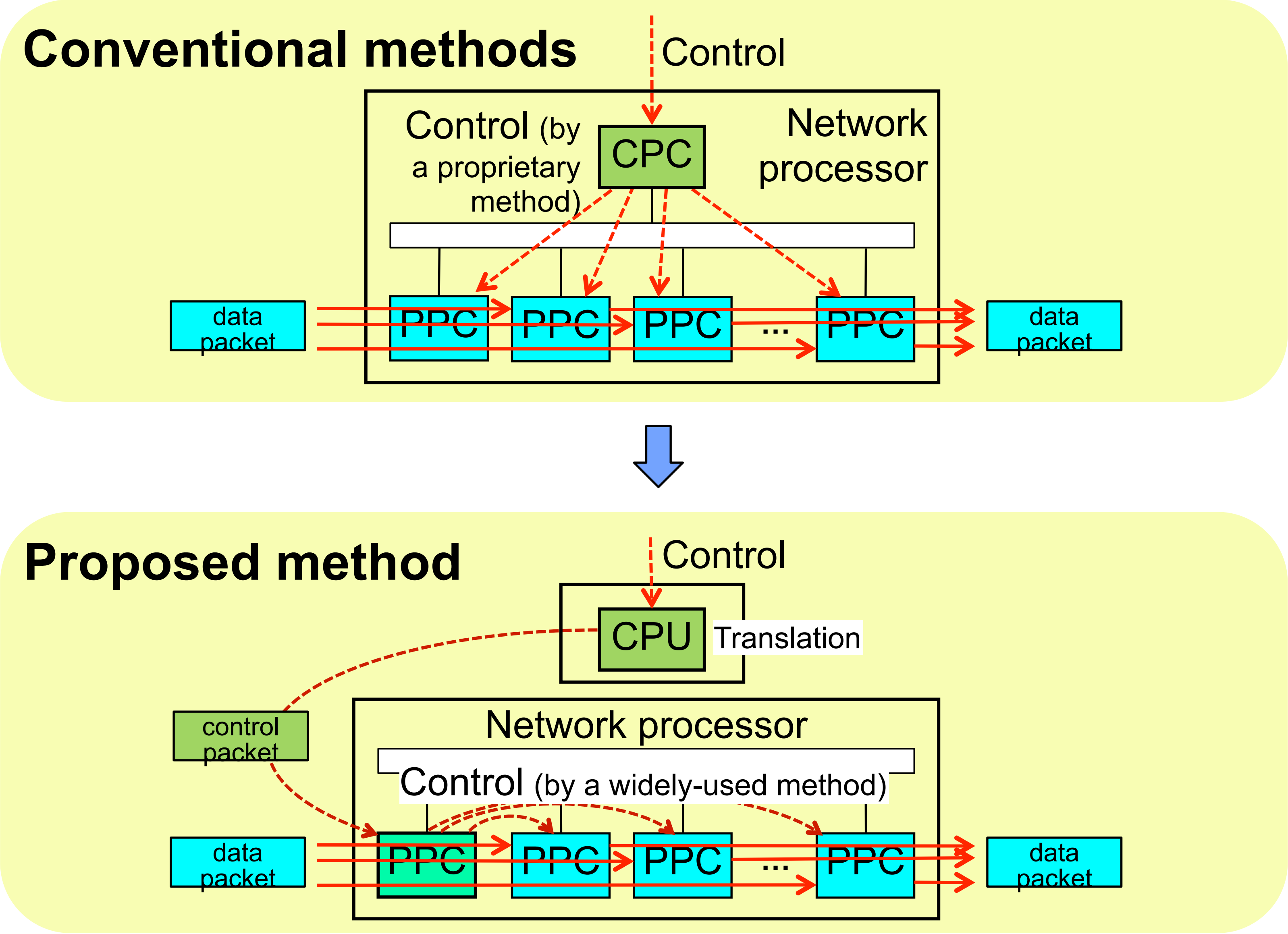
- ▶ **There are various (proprietary) NP architectures.**
  - ■ NDA is required to develop NP programs.
- ▶ **They may be summarized to ...**



Proposal: to control PPCs by a PPC

- **PPCs are conventionally controlled by a CPC.**
  - This control method causes complexity and the problems because of proprietary hardware and software between CPC and PPCs.
  - The complexity comes from the architectural differences between CPC and PPCs.
    - E.g., CPC has virtual memory, but PPCs does not.
    - E.g., CPC runs OS, but PPCs are bare-bone (i.e., OS-less).
- **To simplify the control, a method for controlling PPCs by using a PPC is proposed.**

Comparison of Conventional and Proposed Methods

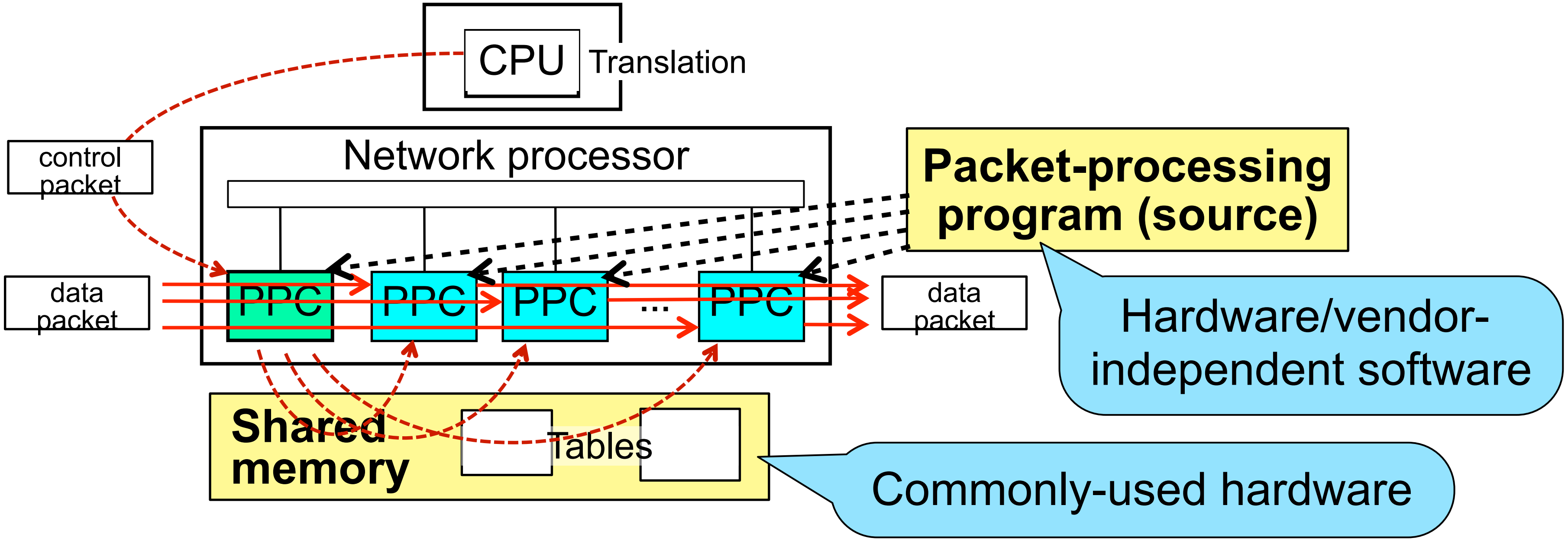


How to Solve Problems by Proposed Method

- ► 1. Communication and synchronization in PPC
- ► 2. Control message simplification in CPU
- ► 3. Core allocation of PPCs

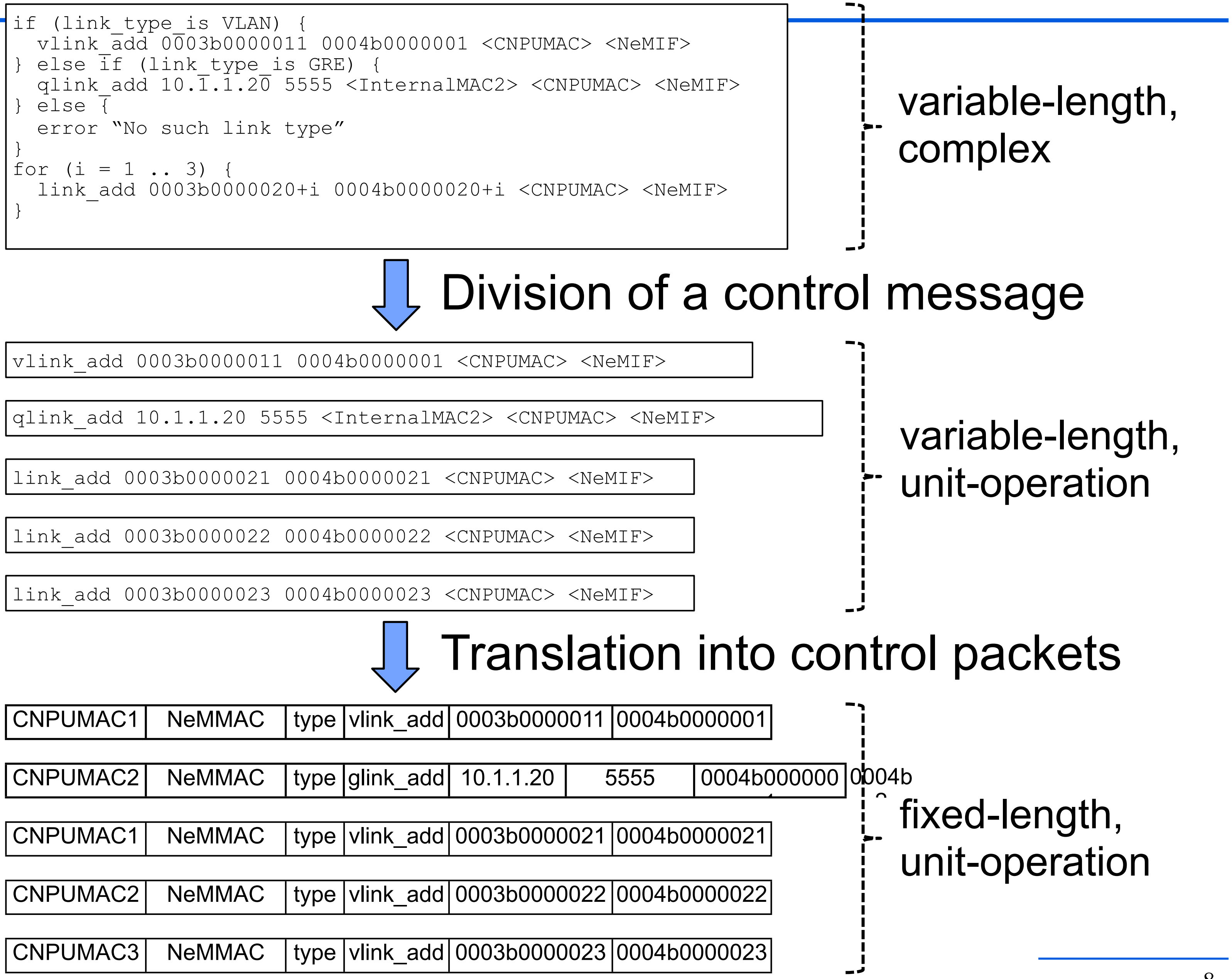
Issue 1: Communication and Synchronization in PPC

- **Uniform and simpler communication and synchronization (C&S) hardware can be used.**
  - C&S hardware between PPCs, such as shared memory, are simpler than that between PPC and CPC.
- **C&S can be programmed in a simpler and hardware- and vendor-*in*dependent method.**
  - A high-level language “Phonepl” is being developed for this purpose.



Issue 2: Control Message Simplification in CPU

Control message



```
if (link_type_is VLAN) {
  vlink_add 0003b0000011 0004b0000001 <CNPUMAC> <NeMIF>
} else if (link_type_is GRE) {
  qlink_add 10.1.1.20 5555 <InternalMAC2> <CNPUMAC> <NeMIF>
} else {
  error “No such link type”
}
for (i = 1 .. 3) {
  link_add 0003b0000020+i 0004b0000020+i <CNPUMAC> <NeMIF>
}
```

variable-length, complex

Division of a control message

```
vlink_add 0003b0000011 0004b0000001 <CNPUMAC> <NeMIF>
```

```
qlink_add 10.1.1.20 5555 <InternalMAC2> <CNPUMAC> <NeMIF>
```

```
link_add 0003b0000021 0004b0000021 <CNPUMAC> <NeMIF>
```

```
link_add 0003b0000022 0004b0000022 <CNPUMAC> <NeMIF>
```

```
link_add 0003b0000023 0004b0000023 <CNPUMAC> <NeMIF>
```

variable-length, unit-operation

Translation into control packets

Control packets (for PPCs)

CNPUMAC1	NeMMAC	type	vlink_add	0003b0000011	0004b0000001		
CNPUMAC2	NeMMAC	type	glink_add	10.1.1.20	5555	0004b000000	0004b
CNPUMAC1	NeMMAC	type	vlink_add	0003b0000021	0004b0000021		
CNPUMAC2	NeMMAC	type	vlink_add	0003b0000022	0004b0000022		
CNPUMAC3	NeMMAC	type	vlink_add	0003b0000023	0004b0000023		

fixed-length, unit-operation

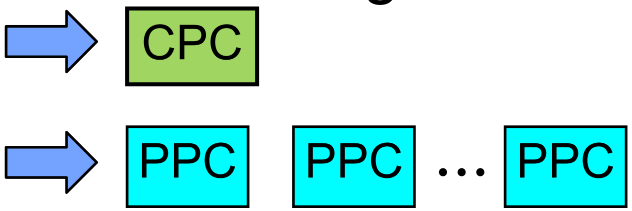
Issue 3: Core Allocation of PPCs

- ▶ **Cores may be allocated statically or dynamically.**
- ▶ **Proposed method is advantageous in both.**
- ▶ **In static allocation, load-balancing is enabled.**
  - ■ **Conventional method**: no load balancing

Control tasks: c1, c2, ..., cn

Data processing tasks:

t1, t2, ..., tm



CPC

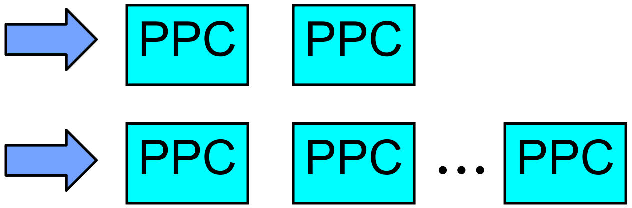
PPC PPC … PPC

  - ■ **Proposed method**: static load balancing

Control tasks: c1, c2, ..., cn

Data processing tasks:

t1, t2, ..., tm



PPC PPC

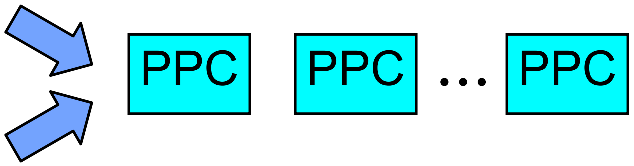
PPC PPC … PPC

- ▶ **Dynamic allocation is enabled.**
  - ■ **Proposed method**: dynamic load balancing

Control tasks: c1, c2, ..., cn

Data processing tasks:

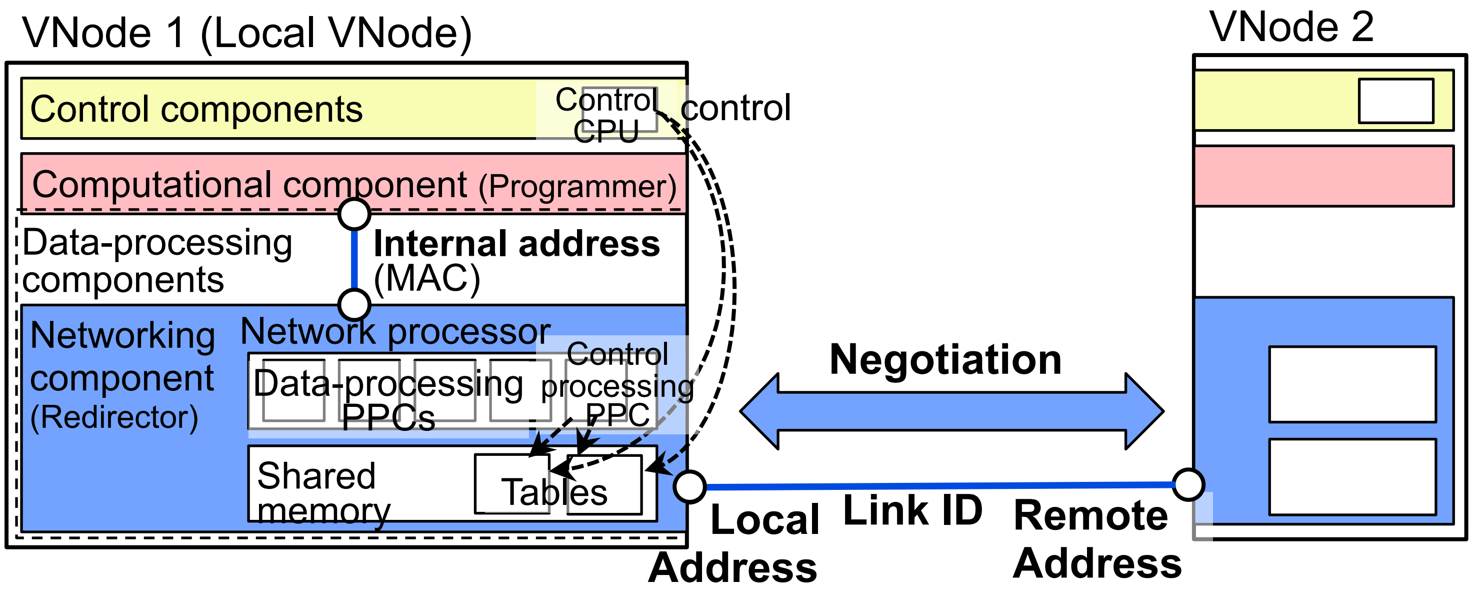
t1, t2, ..., tm



PPC PPC … PPC

Application: Creating a New Type of Virtual Links

- ► A network node with network virtualization function, which is called VNode, has been developed.
- ► NPs are used for adding new functions to a VNode.
- ► By using NP-based plug-ins and the proposed method, a new type of virtual link is created and managed.
  - ■ Built-in virtual-link creation/management mechanism is extended.



Implementation and Evaluation

► Implementation

- Cavium Octeon NPs were used for data processing (packet header conversions).
- Data and control processing tasks were programmed for the PPCs by Phonepl (a high-level languge).

► Comparison of proposed and conventional methods

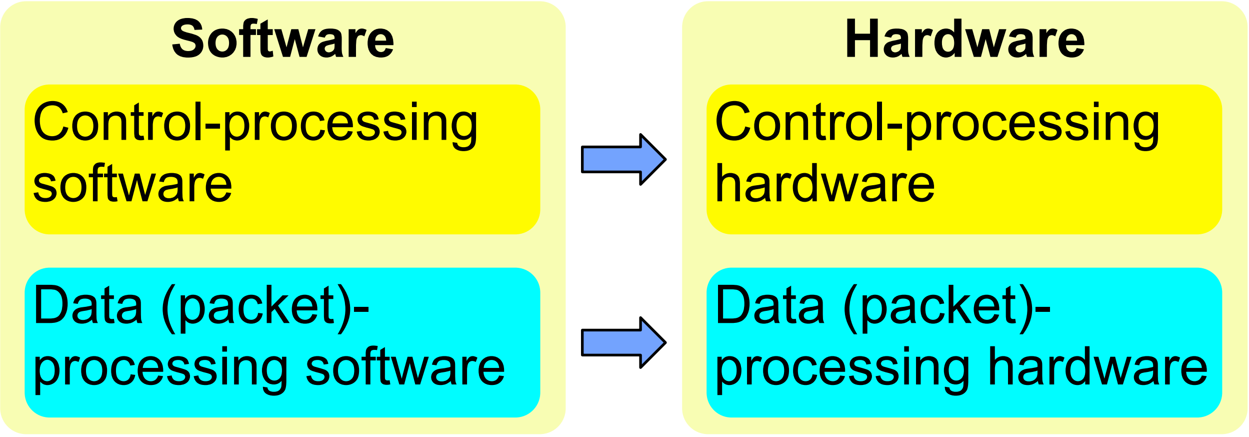
	Data (packet) processing		Control processing		Interface (memory set-up) between D/C	
	Program Length	Description Language	Program Length	Description Language	Program Length	Description Language
Control by PPC (proposed method)	26	Phonepl	21	Phonepl	30	Phonepl
			230	C (Linux)		
Conventional method	160	C (bare metal)	200	C (Linux)	80	C (bare metal)

Conclusion

- A method for controlling packet processing in NPs by using PPCs was proposed.
- This method makes
  - synchronization and communication tasks and programming control/data-processing tasks easier and hardware/vendor-independent.
  - porting between different types of NPs much easier.
- Future work includes application of the proposed method to other types of NPs.

Appendix: Comparison

▶ Conventional control schema



Software

Control-processing software

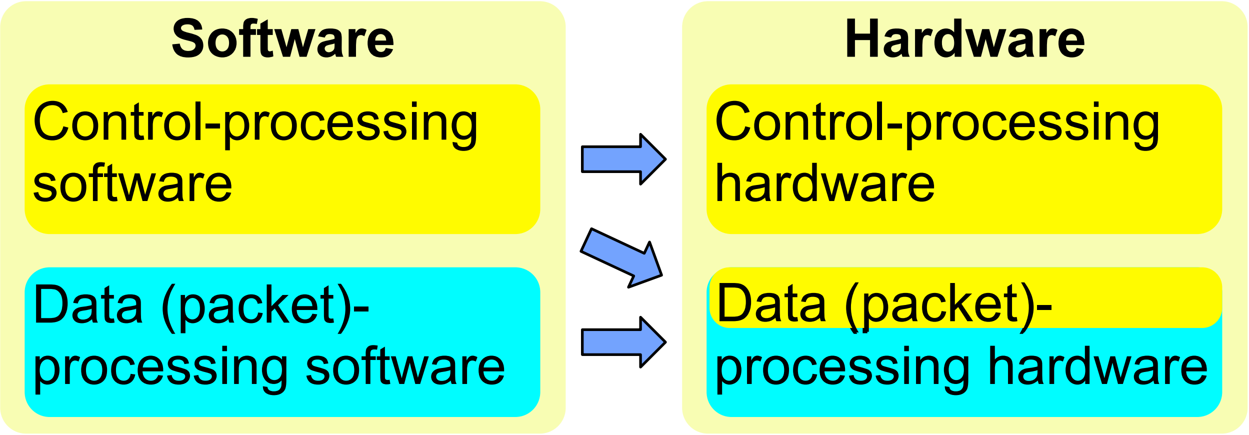
Data (packet)-processing software

Hardware

Control-processing hardware

Data (packet)-processing hardware

▶ Proposed control schema



Software

Control-processing software

Data (packet)-processing software

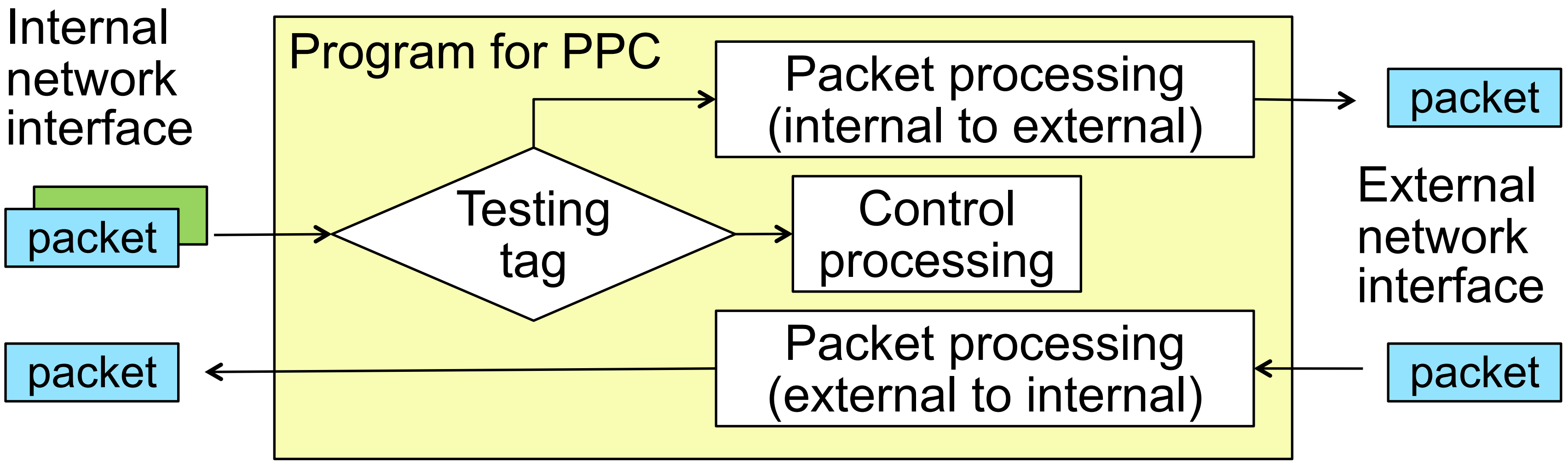
Hardware

Control-processing hardware

Data (packet)-processing hardware

Appendix: Packet & Control Processing in the Application

- ► PPCs are dynamically allocated.
- ► Data/control packets are processed in the following.


Internal network interface
packet
packet
Program for PPC
Testing tag
Packet processing (internal to external)
Control processing
Packet processing (external to internal)
packet
External network interface
packet

Appendix: Packet & Control Processing in the Application

- **“Phonepl” language is used for high-level NP programming.**
  - Packet and control processing are not separated, but they can be separated.

```
import IStream;  // Internal stream
import EStream;  // External stream
class ControlAndDataProcessing {
...
public ControlAndDataProcessing(
     NetStream iport > itoe,
       NetStream eport > etoi) {
 // Initialization
}
void processControl(Packet i) {   // Process a control packet
 // Control-packet processing
}
void itoe(Packet i) {             // Process an i-to-e data packet
 int tag = i....;
 if (tag == ControlPacketTagValue) {
  processControl(i);
 } else {
  // Data-packet processing (internal to external)
 }
}
void etoi(Packet i) {             // Process an e-to-i data packet
 // Data-packet processing (external to internal)
}
void main() {
 new ControlAndDataProcessing(new IStream(), new EStream());
}
}
```