Microsoft Research

Microsoft

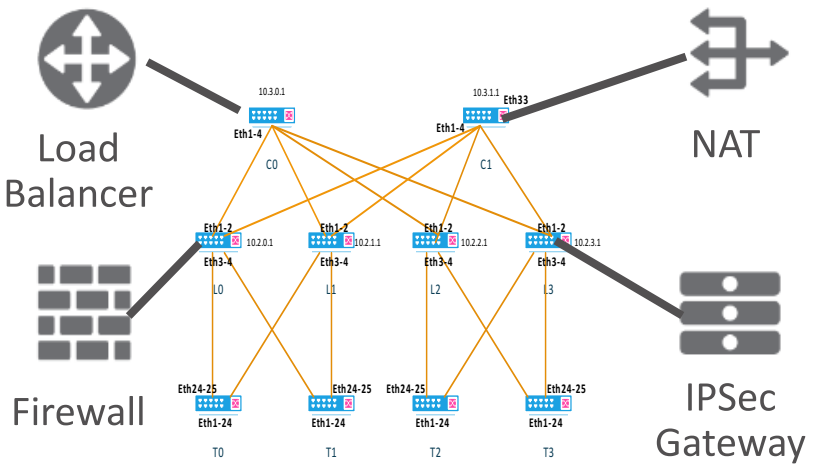
# ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware


Bojie Li[1,2], Kun Tan[1], Layong (Larry) Luo[1], Yanqing Peng[1,3], Renqian Luo[1,2], Ningyi Xu[1], Yongqiang Xiong[1], Peng Cheng[1], Enhong Chen[2]

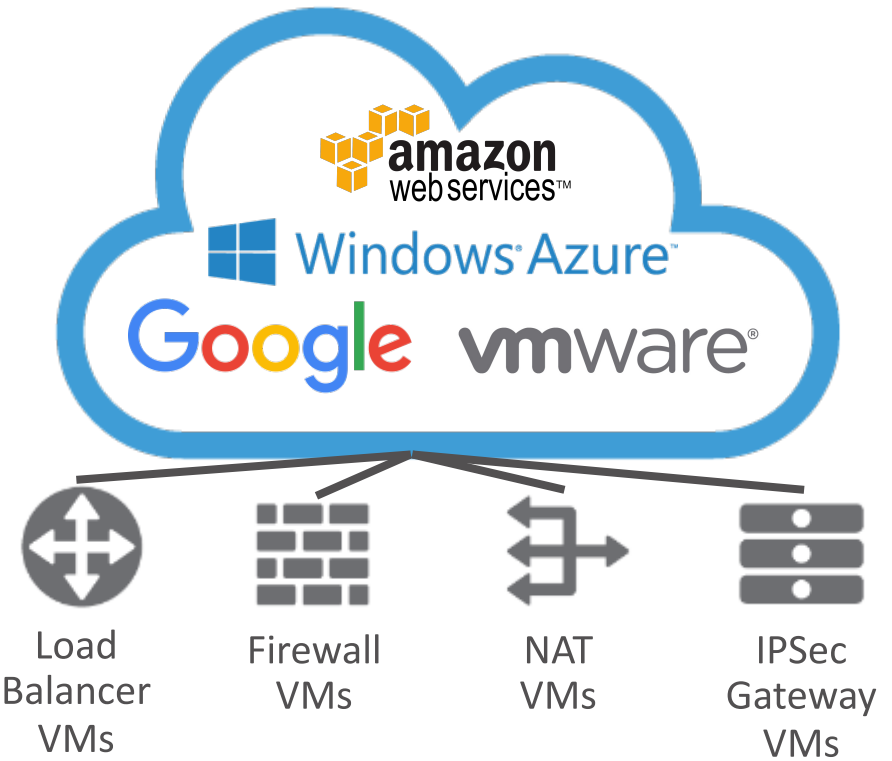
[1]Microsoft Research, [2]USTC, [3]SJTU

# Virtualized network functions

## Dedicated hardware NFs are not flexible




## Virtualized NFs on servers to maximize flexibility



# Scale-up challenges for software NF

## Limited processing capacity

Number of CPU cores needed for 40 Gbps line rate

| Network function | Implementation | 1500B pkt @ 40 Gbps (normal case) | 64B pkt @ 40 Gbps (worst-case estimate) |
|---|---|---|---|
| NVGRE tunnel encapsulation | Hyper-V virtual switch | **5** | **100** |
| Firewall (8K rules) | Linux iptables | **21** | **480** |

# Scale-up challenges for software NF

## Limited processing capacity

Number of CPU cores needed for 40 Gbps line rate

| Network function | Implementation | 1500B pkt @ 40 Gbps (normal case) | 64B pkt @ 40 Gbps (worst-case estimate) |
|---|---|---|---|
| NVGRE tunnel encapsulation | Hyper-V virtual switch | **5** | **100** |
| Firewall (8K rules) | Linux iptables | **21** | **480** |

## Inflated and unstable latency

Add tens of microseconds to milliseconds latency to data plane

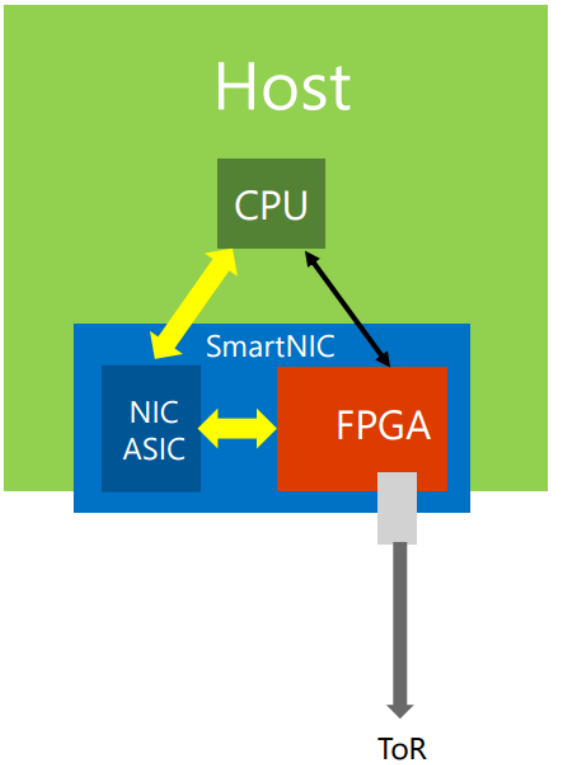
Latency may grow to milliseconds under high load

1 ms occasional delay would violate SLA (e.g., trading services)

# FPGA in the cloud

## FPGA-based SmartNIC

Bump-in-the-wire processing between NIC and ToR switch[1]

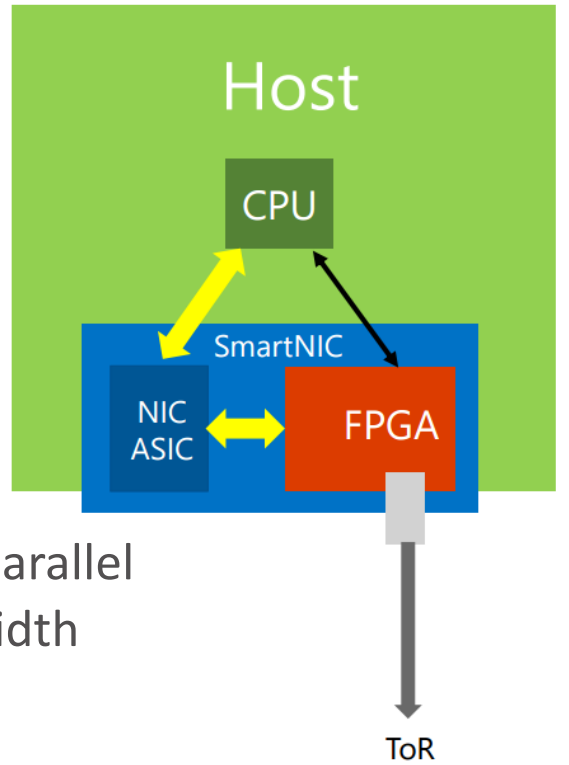



[1] SIGCOMM'15 keynote (also the image source)

# FPGA in the cloud

## FPGA-based SmartNIC

Bump-in-the-wire processing between NIC and ToR switch[1]




## Why FPGA?

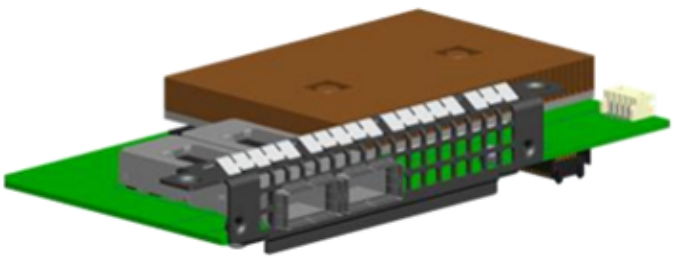
Massive parallelism

- Millions of logic elements ⇨ Thousands of "cores" in parallel
- Thousands of memory blocks ⇨ High memory bandwidth

Low power consumption (~20W)

General computing platform (vs. GPU, NP) to accelerate various cloud services

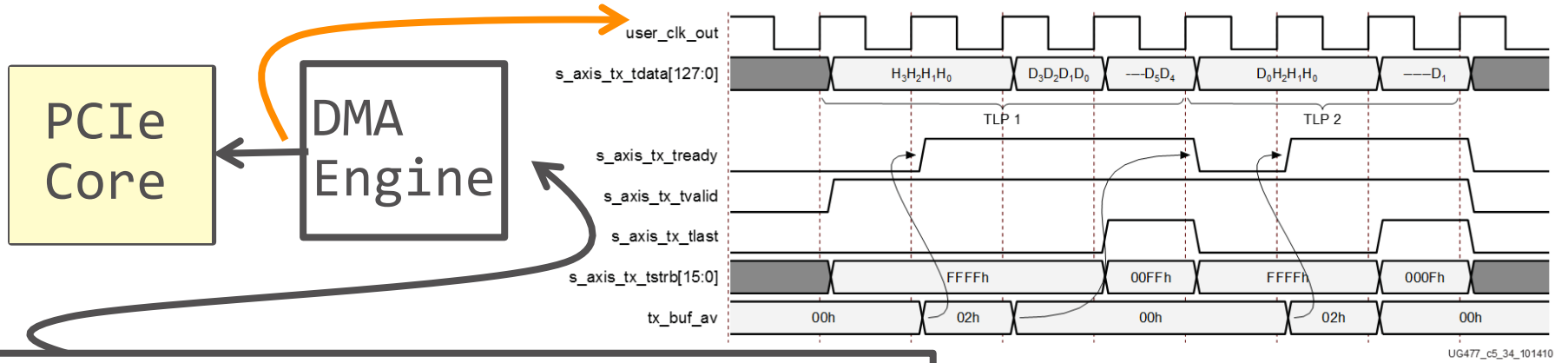
Mature technology with a reasonable price



[1] SIGCOMM'15 keynote (also the image source)

# FPGA challenge: *Programmability*

## Hardware description language (HDL): push many software developers away

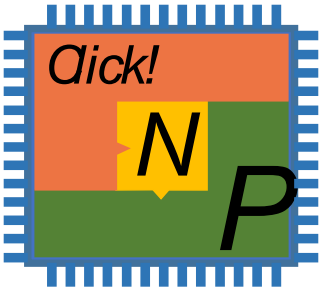



```
always @ (posedge SYSCLK or negedge RST_B)
begin
 if(!RST_B)
  DMA_TX_DATA <= `UD 8'hff;
 else
  DMA_TX_DATA <= `UD DMA_TX_DATA_N;
end

//send: "hello world !"
always @ (posedge SYSCLK)
begin
   if (rst) begin
            dma_data <= 88'h0;
            dma_valid <= 1'b0;
      else
          if (dma_start) begin
              dma_data <=
88'h68656C6C6F20776F726C64;
              dma_valid <= 1'b1;
              end
   else begin
          dma_data <= 88'h0;
          dma_valid <= 1'b0;
   end
   end
   end
```

Ahhhhhhhhhhhh!

# Project ClickNP

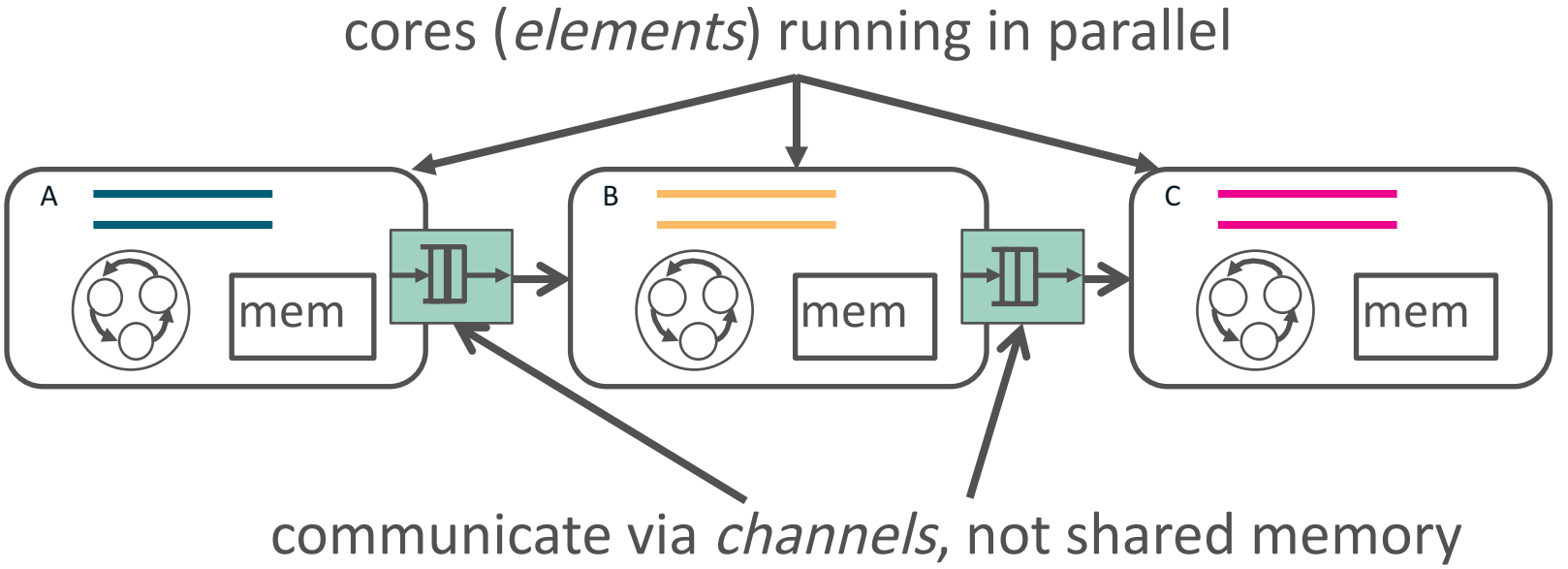



## Making FPGA accessible to software developers

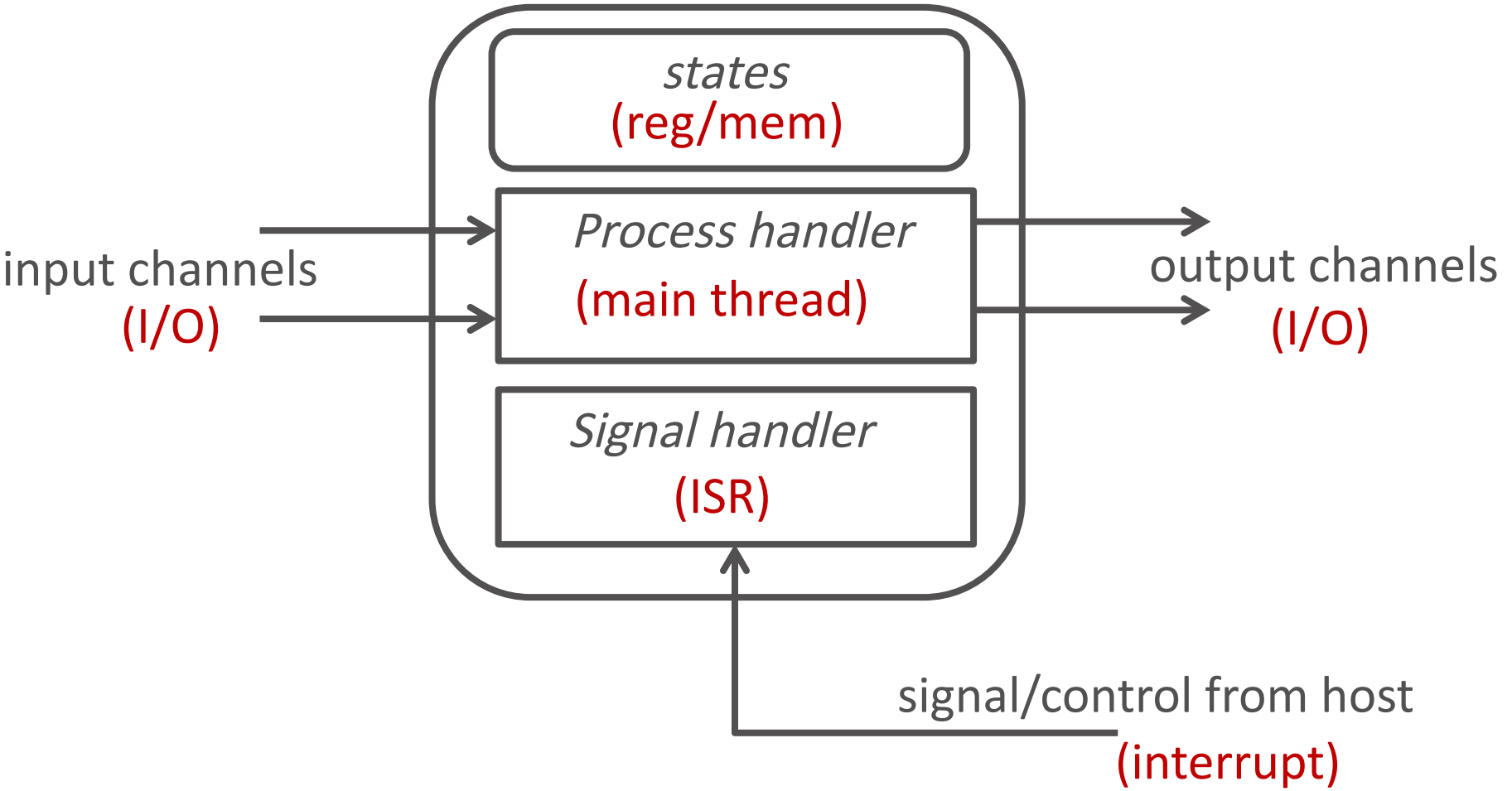
- Flexibility: *fully programmable using high-level language*
- Modularized: *Click abstractions familiar to software developers; easy code reuse*
- High performance: *high throughput; microsecond-scale latency*
- Joint CPU/FPGA packet processing: *FPGA is no panacea; fine-grained processing separation*

# Programming model

## *as if* programming on a *multi-core* processor



# Element: single-threaded core



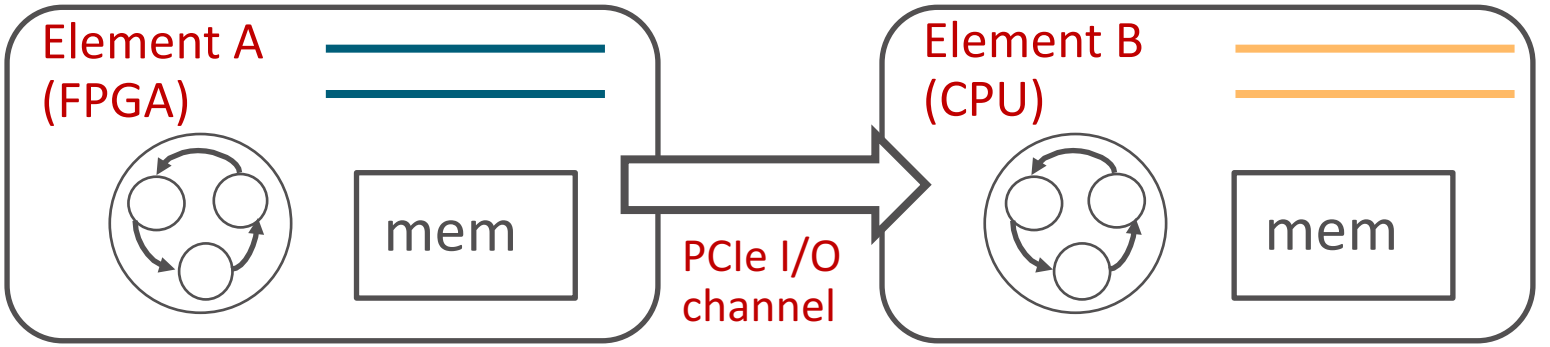
# Extension 1: CPU element

## Write once, run on both FPGA and CPU

Compile to a logic block on FPGA or a binary running on CPU

Enable joint CPU/FPGA programming

Simplify debugging




## PCIe I/O channel

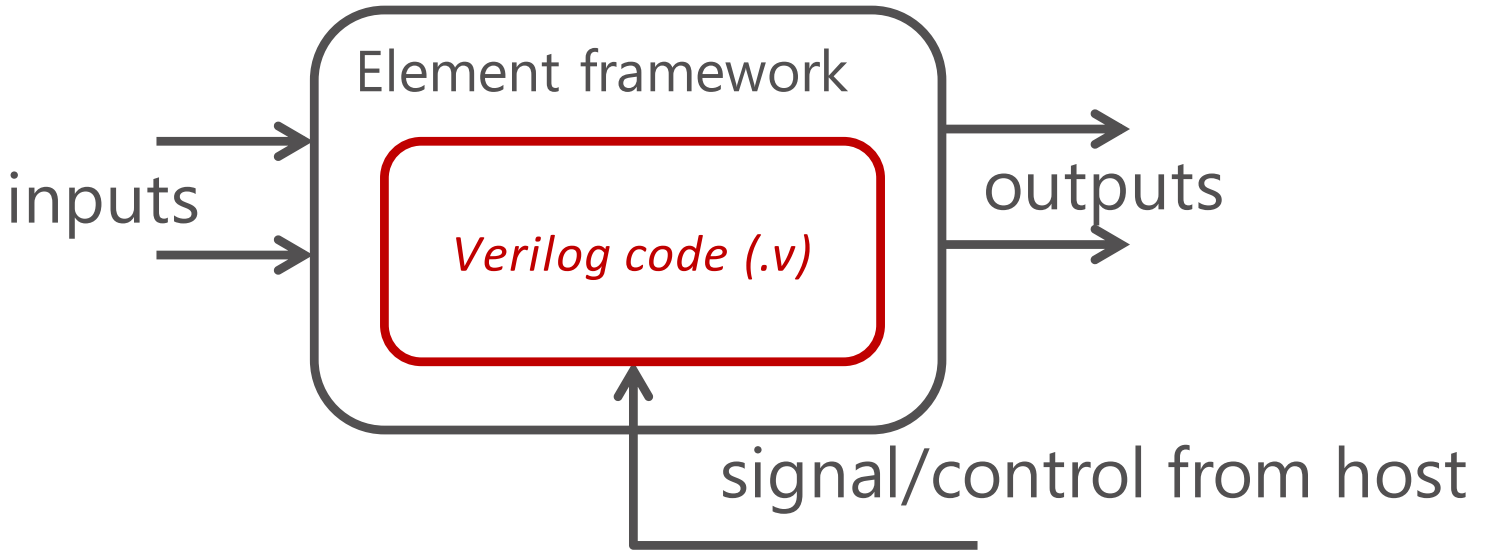
Enable fast communication between elements on FPGA and CPU

- Low latency: 1 µs
- High throughput: 25.6 Gbps (PCIe Gen2 x8)

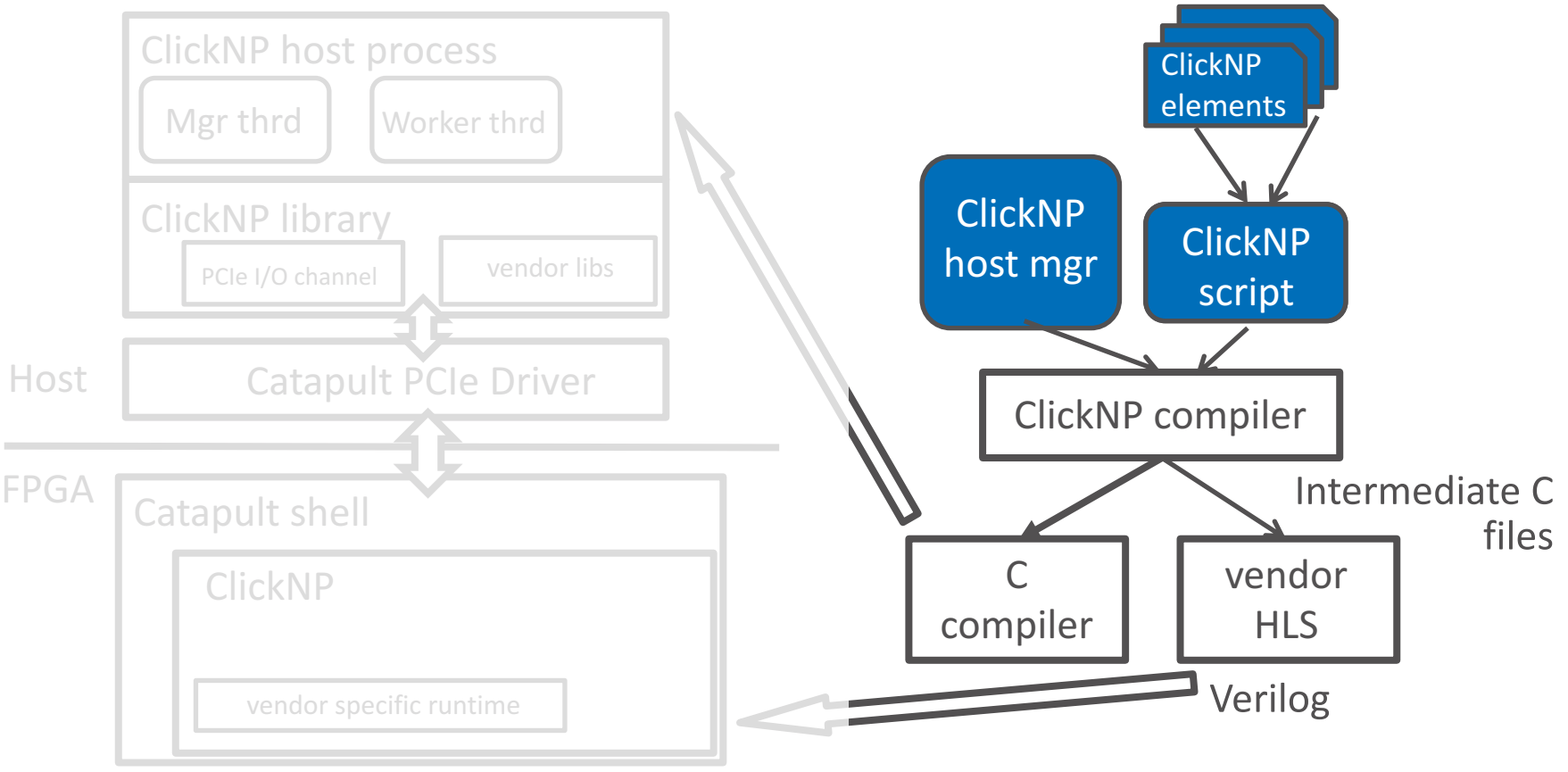
# Extension 2: Verilog Element

## Embedding native Verilog code in ClickNP element

Enable hybrid C/Verilog programming
(analog to C/ASM in CPU world)



# Architecture

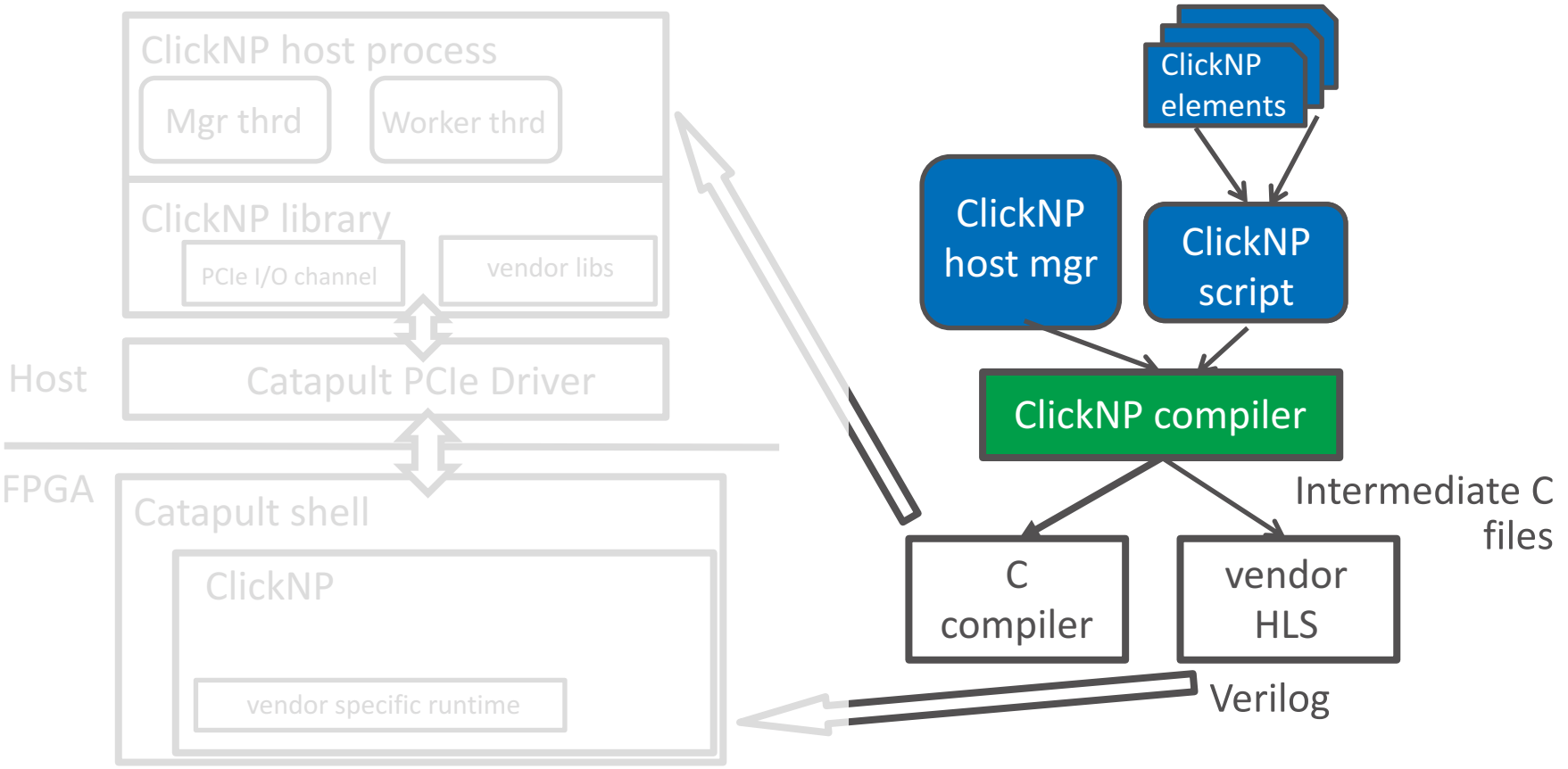



Cross-platform toolchain

Altera OpenCL / Vivado HLS

Visual Studio / GCC

# Architecture

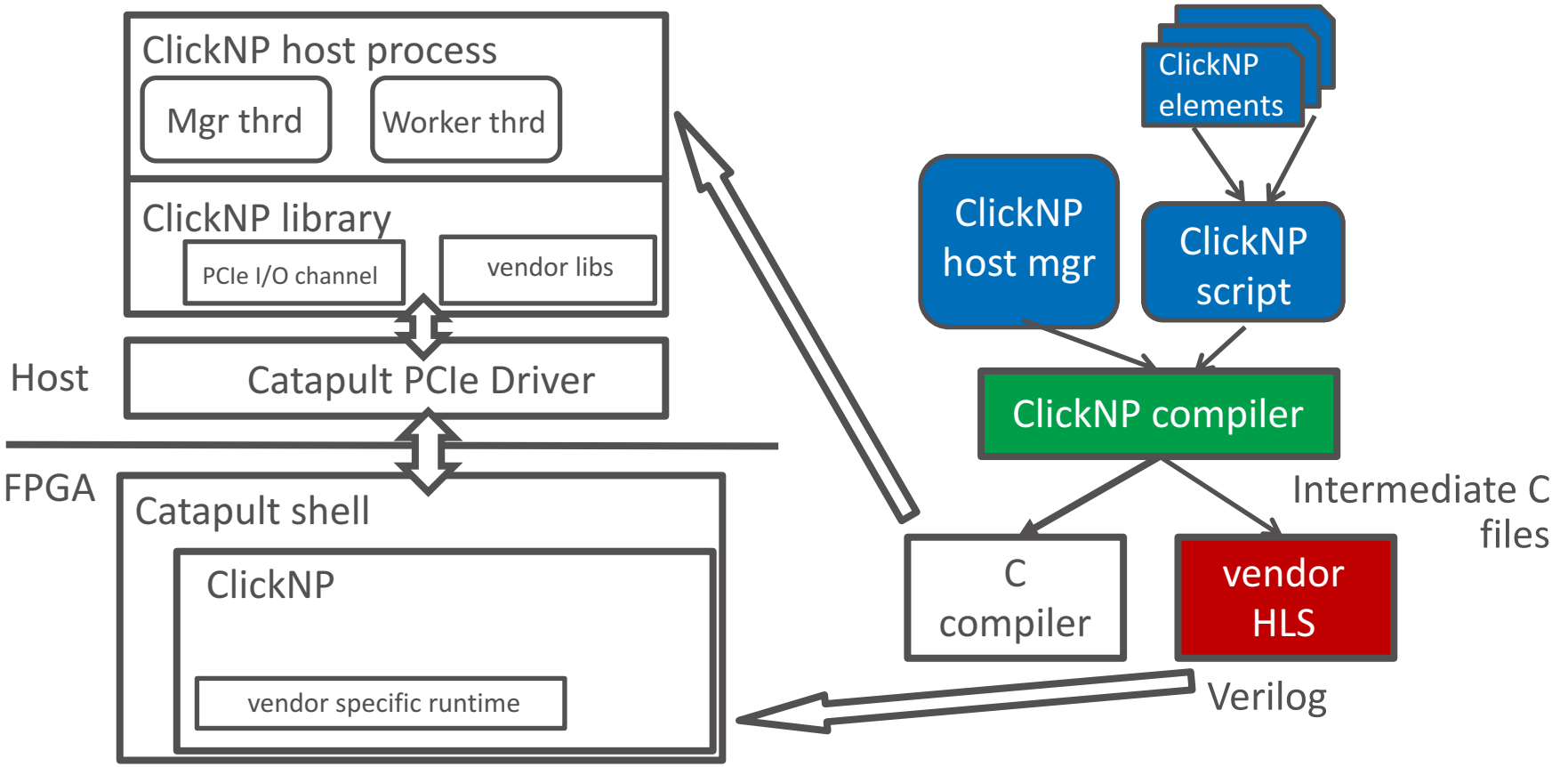



Cross-platform toolchain

Altera OpenCL / Vivado HLS

Visual Studio / GCC

# Architecture

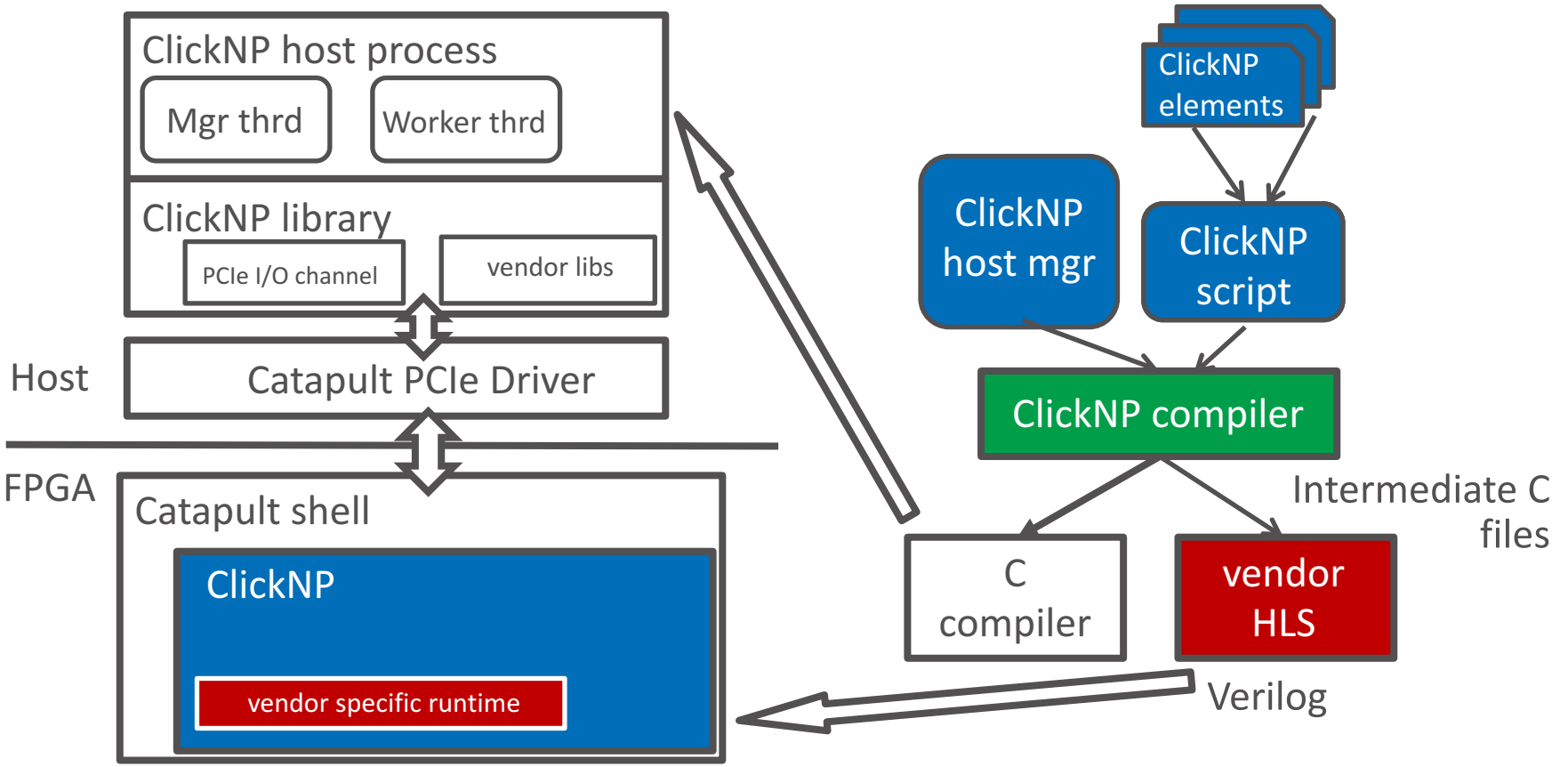



Cross-platform toolchain

Altera OpenCL / Vivado HLS

Visual Studio / GCC

# Architecture

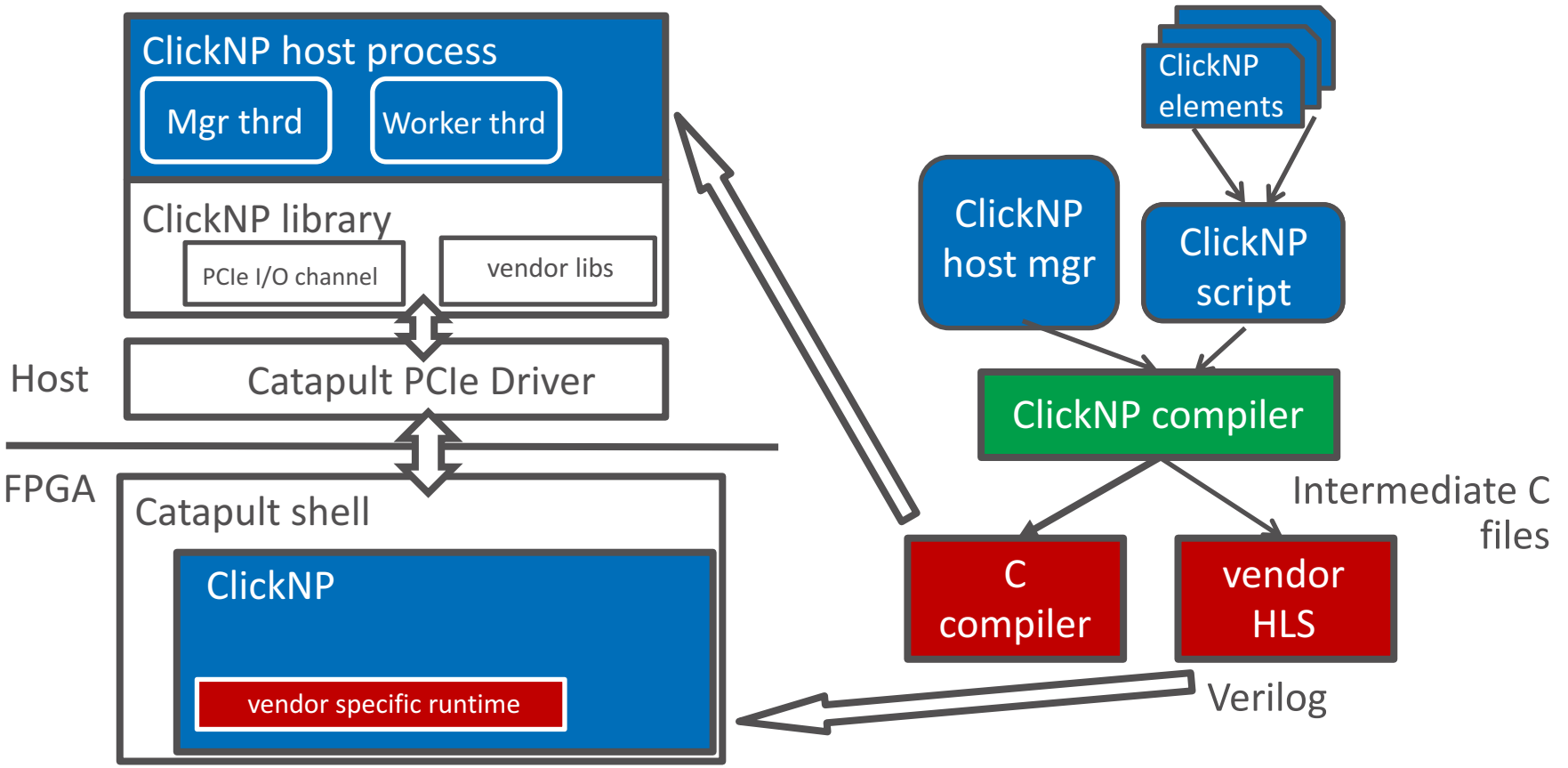



Cross-platform toolchain

Altera OpenCL / Vivado HLS

Visual Studio / GCC

# Architecture

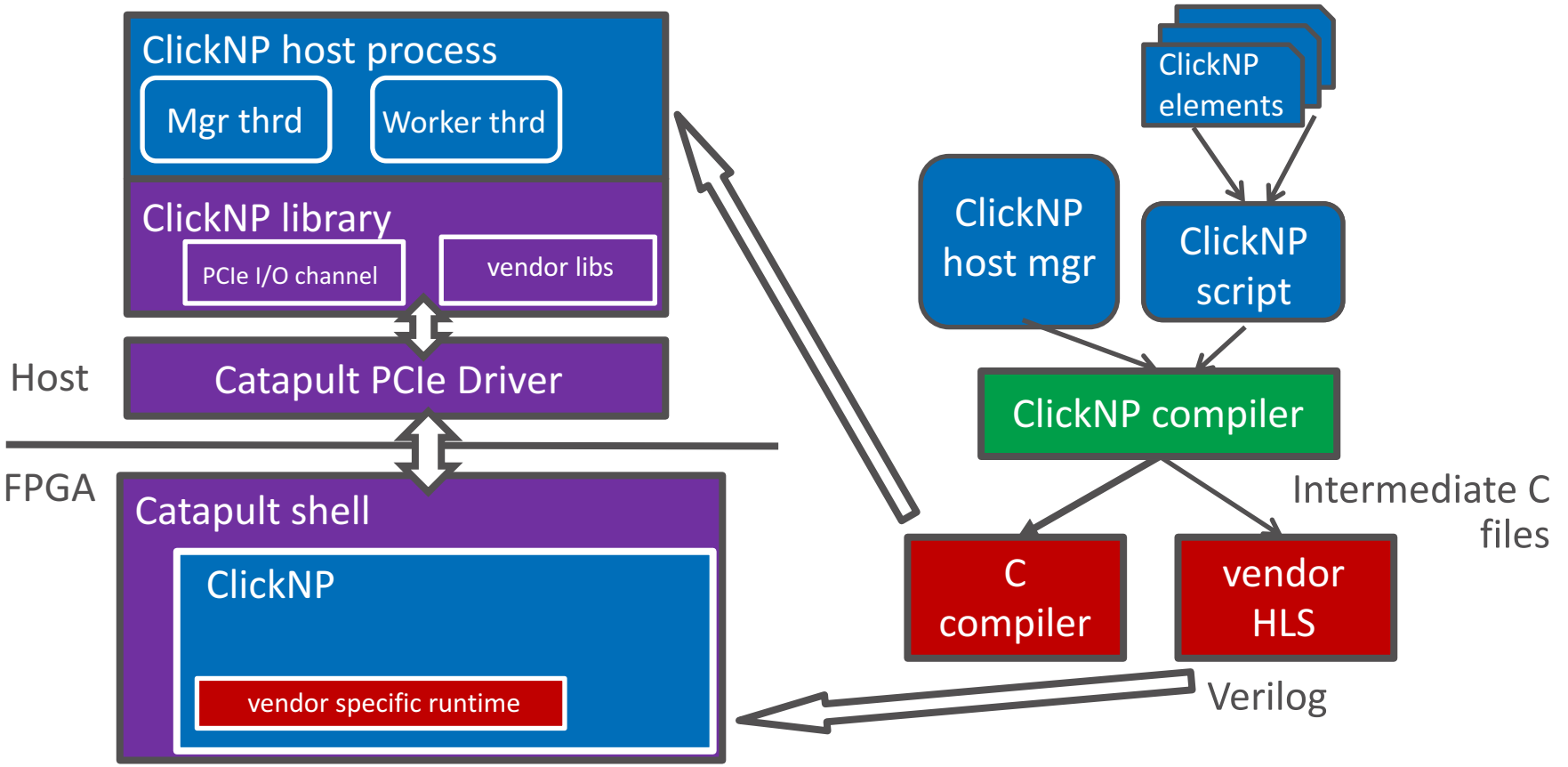



Cross-platform toolchain

Altera OpenCL / Vivado HLS

Visual Studio / GCC

# Architecture

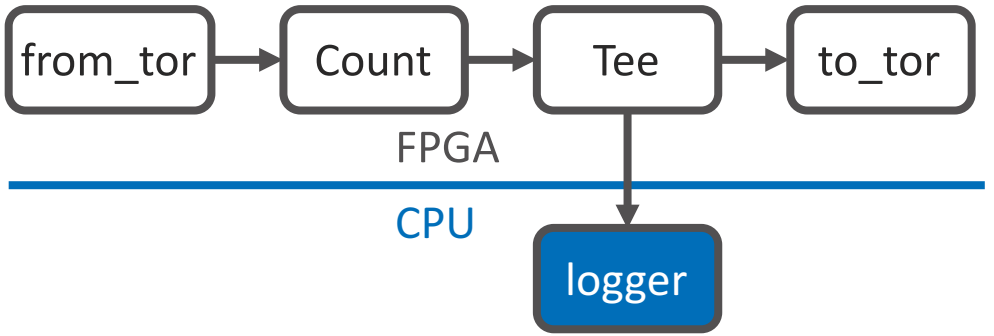



Cross-platform toolchain

Altera OpenCL / Vivado HLS

Visual Studio / GCC

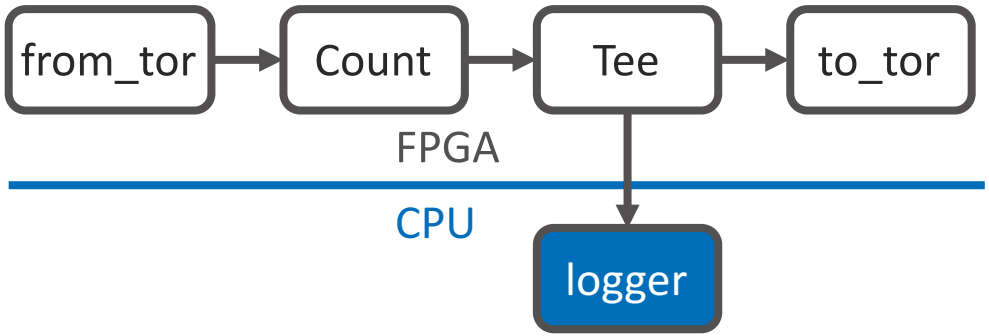
# Example: Packet logger




ClickNP Configuration:

```
1 Count :: cnt @
2 Tee :: tee
3 host PktLogger :: logger
4
5 from_tor -> cnt -> tee [1] -> to_tor
6 tee [2] -> logger
```

# Example: Packet logger




ClickNP Configuration:

```
Count :: cnt @
Tee :: tee
host PktLogger :: logger

from_tor -> cnt -> tee [1] -> to_tor
tee [2] -> logger
```

Count element:

```
.element Count <1, 1> {
  .state{
    ulong count;
  }
  .init{
    count = 0;
  }
  .handler{
    if (get_input_port() != PORT_1) {
      return (PORT_1);
    }
    flit x;
    x = read_input_port(PORT_1);
    if (x.fd.sop) count = count + 1;
    set_output_port(PORT_1, x);

    return (PORT_1);
  }
  .signal{
    ClSignal p;
    p.Sig.LParam[0] = count;
    set_signal(p);
  }
}
```

# Parallelizing in FPGA

## Parallelism across elements

Pipeline parallelism

Data parallelism

## Parallelism inside an element

Minimize memory dependency

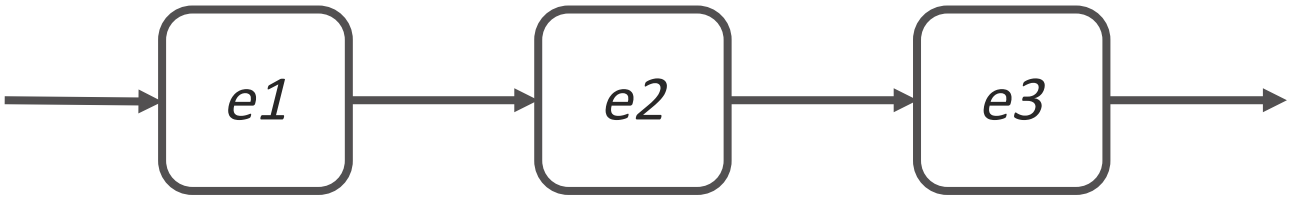
- Use registers
- Delayed write
- Memory scattering

Balance pipeline stages

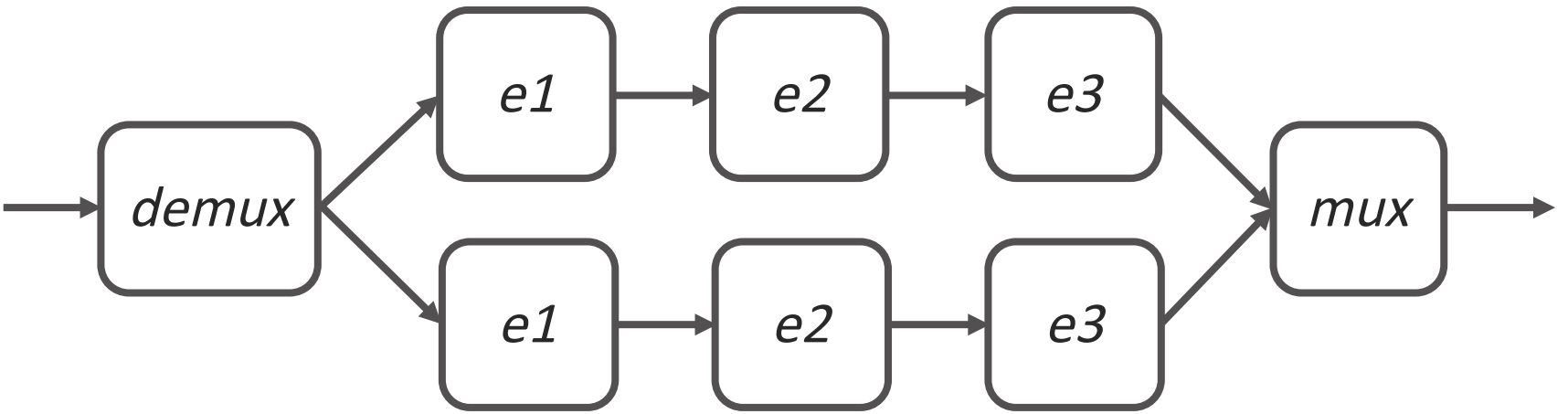
- Unroll loops
- Offload slow path to another element

# Parallelism across elements

## Pipeline parallelism

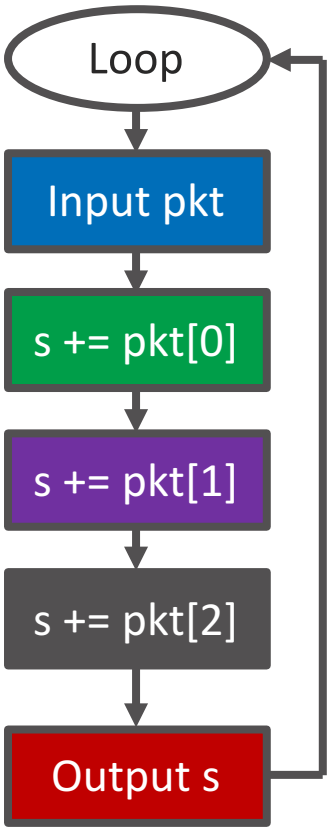


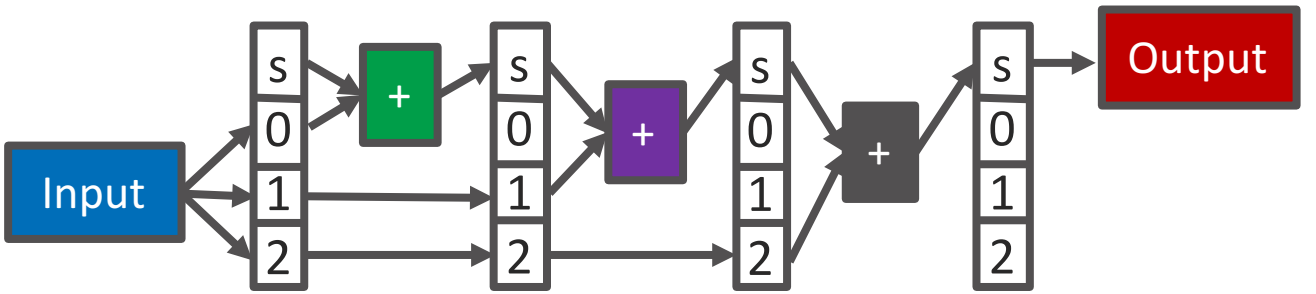

## Duplicate pipeline to leverage data parallelism

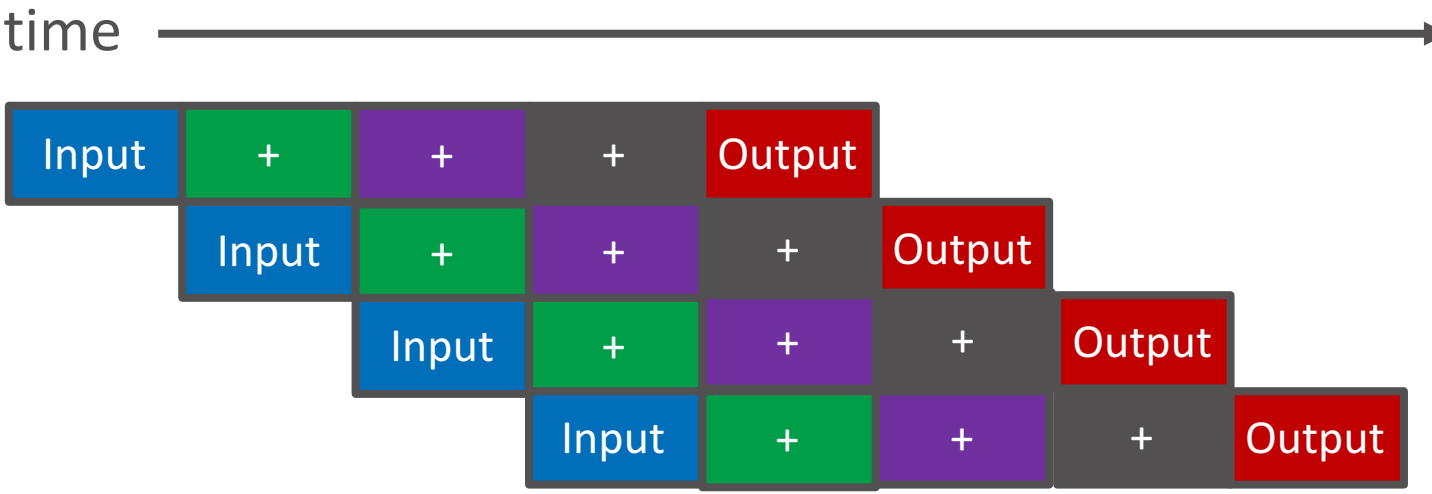


# Parallelism inside element (1)

## Element is synthesized into a pipeline

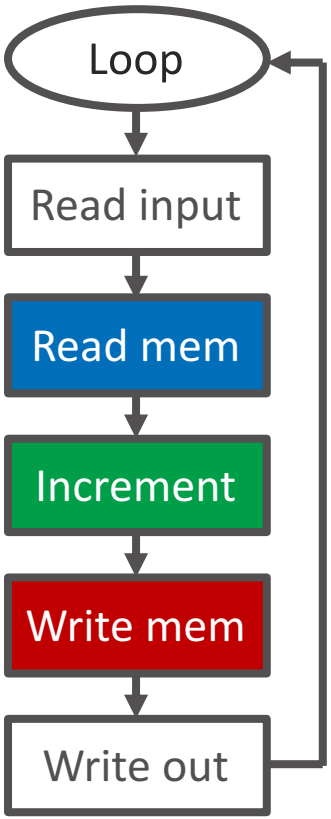


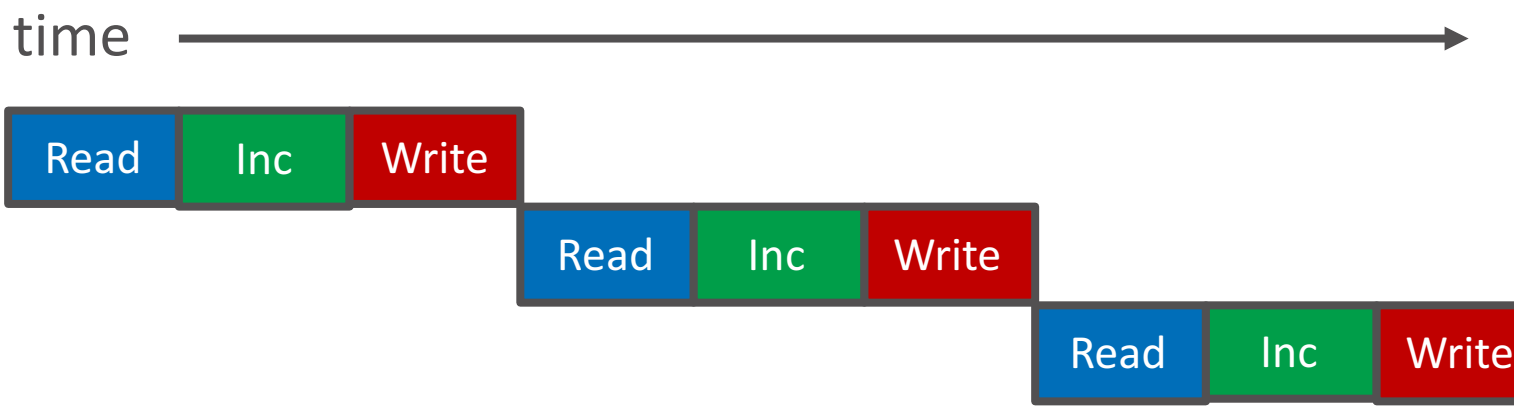



# Parallelism inside element (2)

## Principle 1: Minimize memory dependency



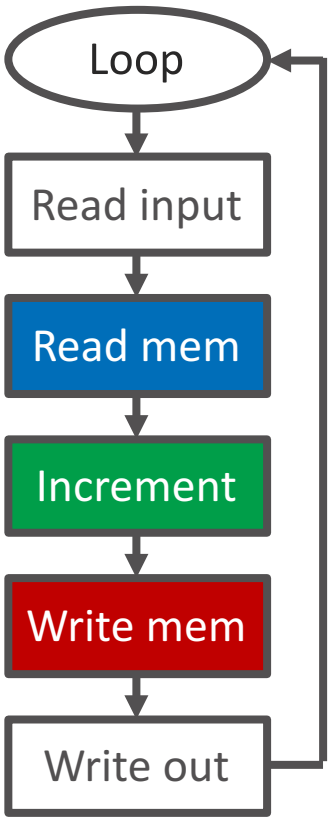



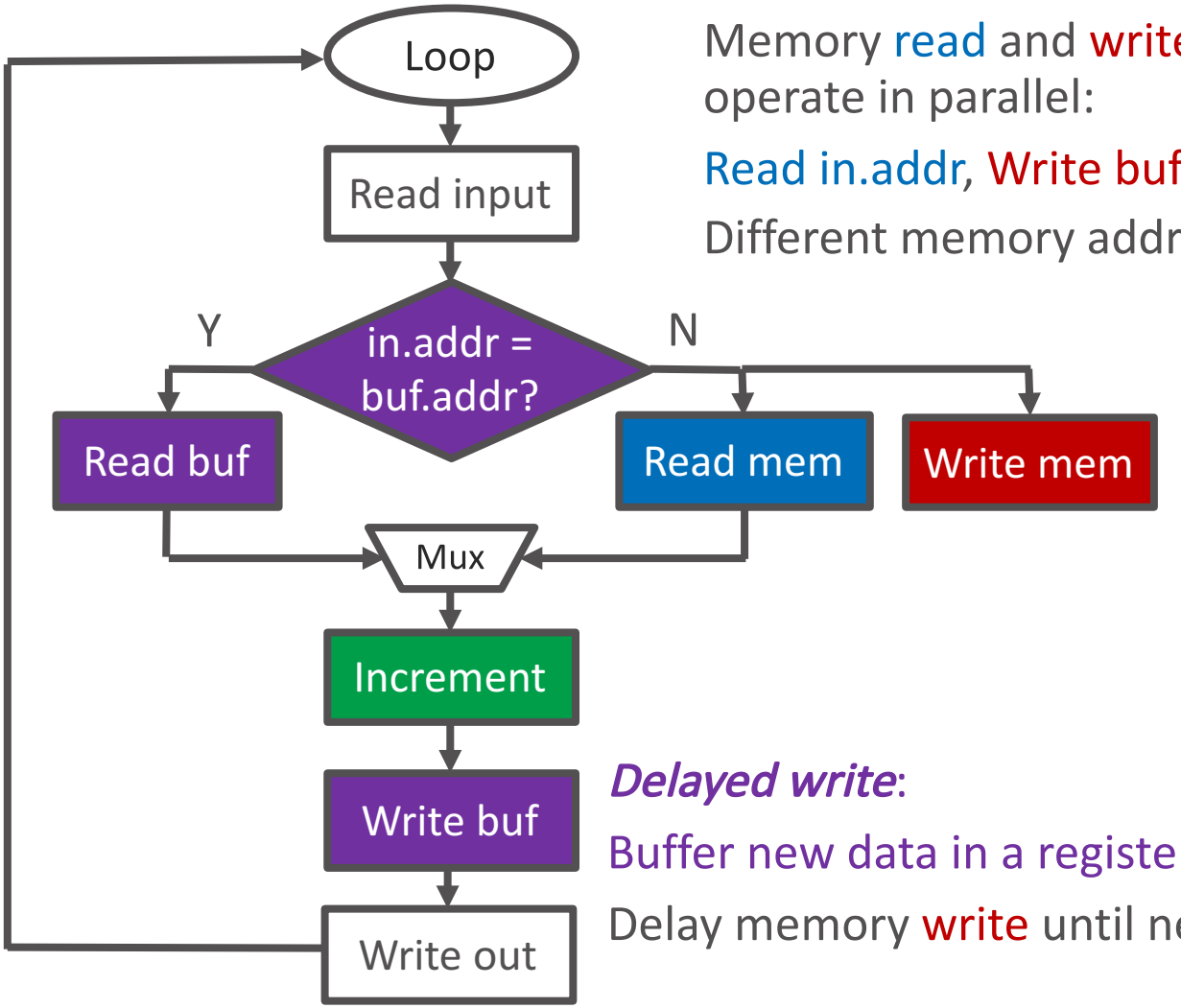

Memory dependency:

Read cannot start before last write,
because read and write addresses may conflict.

# Parallelism inside element (3)

## *Delayed write* to remove read-write dependency




Memory read and write can operate in parallel:

Read in.addr, Write buf.addr

Different memory addresses!

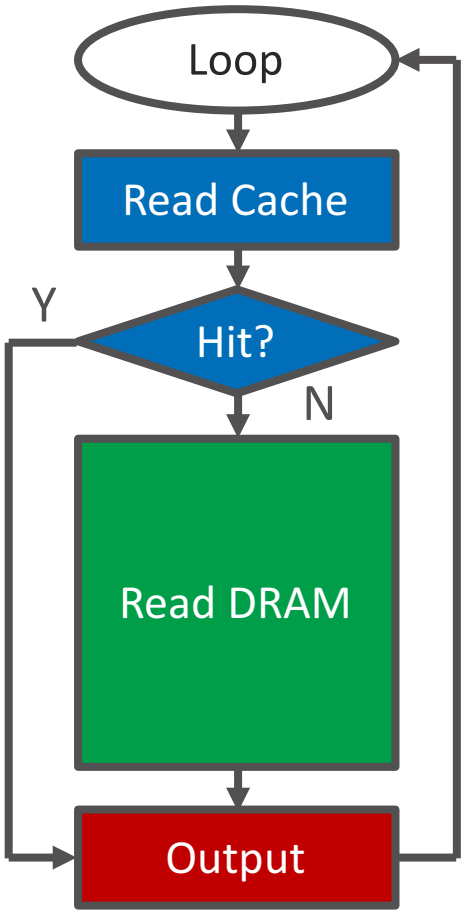
***Delayed write***:

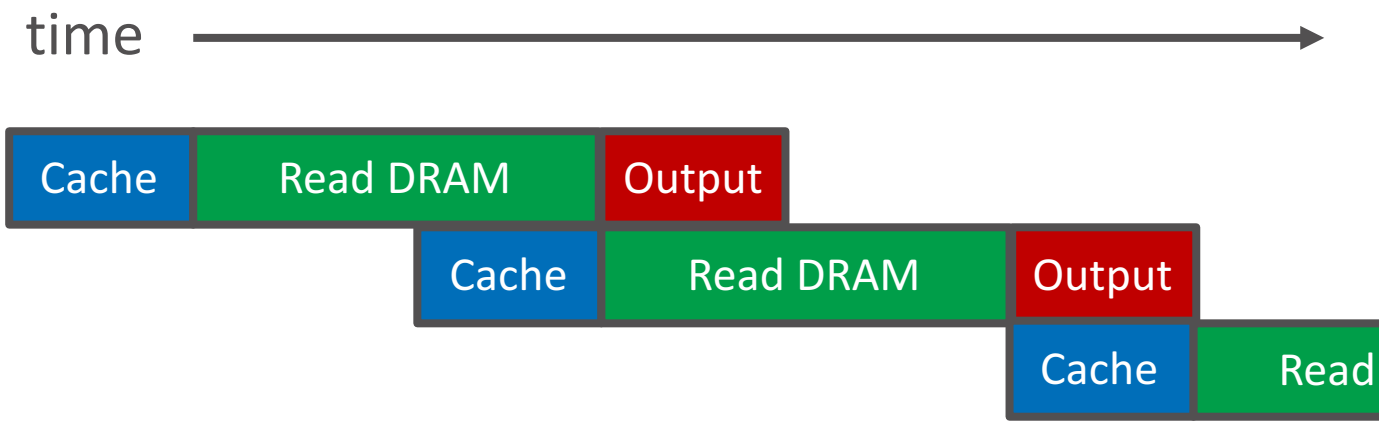
Buffer new data in a register

Delay memory write until next read

# Parallelism inside element (4)

## Principle 2: Balance pipeline stages



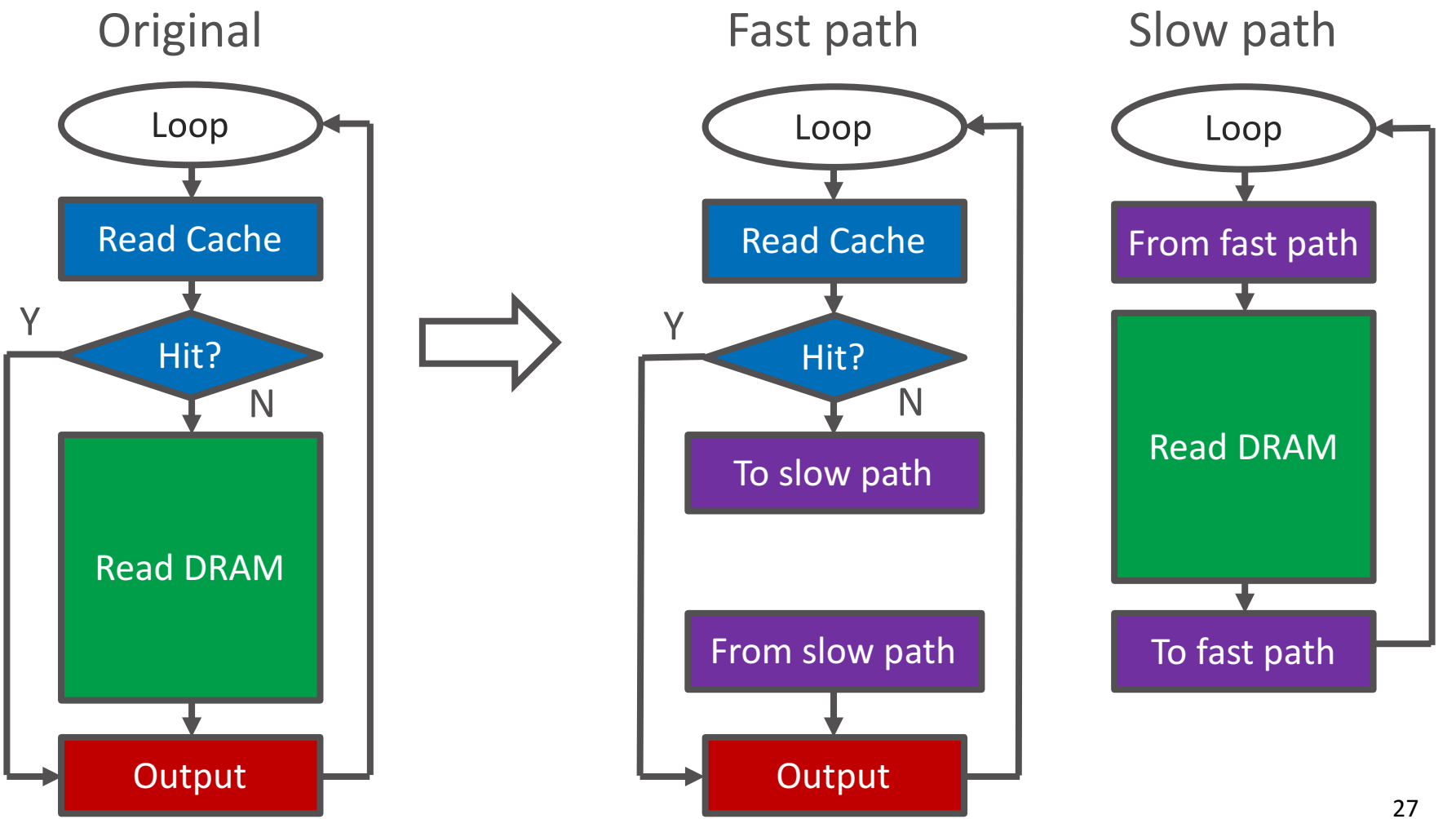

DRAM read is an unbalanced pipeline stage.

Pipeline is scheduled with a fixed interval to cover the worst-case latency.

So throughput is bottlenecked by DRAM read latency.

# Parallelism inside element (5)

## Offload slow path to another element

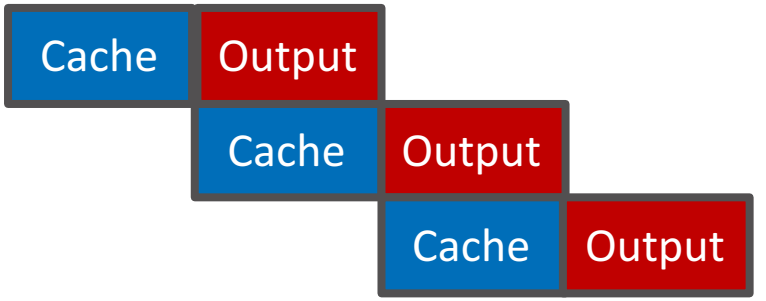


# Parallelism inside element (6)

## Offload slow path to another element

time

3 cache hits:




1 cache hit, 1 cache miss, 1 cache hit:

# ClickNP element library

| Element | Fmax (MHz) | Peak Throughput | Delay (cycles) | Resource LE % | Resource BRAM % |
|---|---|---|---|---|---|
| L4_Parser | 221.9 | 113.6 Gbps | 11 | 0.8% | 0.2% |
| IPChecksum | 226.8 | 116.1 Gbps | 18 | 2.3% | 1.3% |
| NVGRE_Encap | 221.8 | 113.6 Gbps | 9 | 1.5% | 0.6% |
| AES_CTR | 217.0 | 27.8 Gbps | 70 | 4.0% | 23.1% |
| SHA1 | 220.8 | 113.0 Gbps | 105 | 7.9% | 6.6% |
| CuckooHash | 209.7 | 209.7 Mpps | 38 | 2.0% | 65.5% |
| HashTCAM | 207.4 | 207.4 Mpps | 48 | 18.7% | 22.0% |
| LPM_Tree | 221.8 | 221.8 Mpps | 181 | 4.3% | 13.2% |
| SRPrioQueue | 214.5 | 214.5 Mpps | 41 | 2.6% | 0.6% |
| RateLimiter | 141.5 | 141.5 Mpps | 14 | 16.9% | 14.1% |

Nearly 100 elements
20% re-factored from Click modular router
Cover packet parsing, checksum, tunnel encap/decap, crypto, hash tables, prefix matching, packet scheduling, rate limiting…

Throughput: 200 Mpps / 100 Gbps

Mean delay: 0.19 us, max delay: 0.8 us

Mean LoC: 80, max LoC: 196

# ClickNP element library

| Element | Fmax (MHz) | Peak Throughput | Delay (cycles) | Resource LE % | Resource BRAM % |
|---|---|---|---|---|---|
| L4_Parser | 221.9 | 113.6 Gbps | 11 | 0.8% | 0.2% |
| IPChecksum | 226.8 | 116.1 Gbps | 18 | 2.3% | 1.3% |
| NVGRE_Encap | 221.8 | 113.6 Gbps | 9 | 1.5% | 0.6% |
| AES_CTR | 217.0 | 27.8 Gbps | 70 | 4.0% | 23.1% |
| SHA1 | 220.8 | 113.0 Gbps | 105 | 7.9% | 6.6% |
| CuckooHash | 209.7 | 209.7 Mpps | 38 | 2.0% | 65.5% |
| HashTCAM | 207.4 | 207.4 Mpps | 48 | 18.7% | 22.0% |
| LPM_Tree | 221.8 | 221.8 Mpps | 181 | 4.3% | 13.2% |
| SRPrioQueue | 214.5 | 214.5 Mpps | 41 | 2.6% | 0.6% |
| RateLimiter | 141.5 | 141.5 Mpps | 14 | 16.9% | 14.1% |

Nearly 100 elements
20% re-factored from Click modular router
Cover packet parsing, checksum, encap/decap, hash tables, prefix matching, rate limiting, crypto, packet scheduling…

Throughput: 200 Mpps / 100 Gbps

Mean delay: 0.19 us, max delay: 0.8 us

Mean LoC: 80, max LoC: 196

# ClickNP element library

| Element | Fmax (MHz) | Peak Throughput | Delay (cycles) | Resource LE % | Resource BRAM % |
|---|---|---|---|---|---|
| L4_Parser | 221.9 | 113.6 Gbps | 11 | 0.8% | 0.2% |
| IPChecksum | 226.8 | 116.1 Gbps | 18 | 2.3% | 1.3% |
| NVGRE_Encap | 221.8 | 113.6 Gbps | 9 | 1.5% | 0.6% |
| AES_CTR | 217.0 | 27.8 Gbps | 70 | 4.0% | 23.1% |
| SHA1 | 220.8 | 113.0 Gbps | 105 | 7.9% | 6.6% |
| CuckooHash | 209.7 | 209.7 Mpps | 38 | 2.0% | 65.5% |
| HashTCAM | 207.4 | 207.4 Mpps | 48 | 18.7% | 22.0% |
| LPM_Tree | 221.8 | 221.8 Mpps | 181 | 4.3% | 13.2% |
| SRPrioQueue | 214.5 | 214.5 Mpps | 41 | 2.6% | 0.6% |
| RateLimiter | 141.5 | 141.5 Mpps | 14 | 16.9% | 14.1% |

Nearly 100 elements
20% re-factored from Click modular router
Cover packet parsing, checksum, encap/decap, hash tables, prefix matching, rate limiting, crypto, packet scheduling…

Throughput: 200 Mpps / 100 Gbps

Mean delay: 0.19 us, max delay: 0.8 us

Mean LoC: 80, max LoC: 196

# ClickNP element library

| Element | Fmax (MHz) | Peak Throughput | Delay (cycles) | Resource LE % | Resource BRAM % |
|---|---|---|---|---|---|
| L4_Parser | 221.9 | 113.6 Gbps | 11 | 0.8% | 0.2% |
| IPChecksum | 226.8 | 116.1 Gbps | 18 | 2.3% | 1.3% |
| NVGRE_Encap | 221.8 | 113.6 Gbps | 9 | 1.5% | 0.6% |
| AES_CTR | 217.0 | 27.8 Gbps | 70 | 4.0% | 23.1% |
| SHA1 | 220.8 | 113.0 Gbps | 105 | 7.9% | 6.6% |
| CuckooHash | 209.7 | 209.7 Mpps | 38 | 2.0% | 65.5% |
| HashTCAM | 207.4 | 207.4 Mpps | 48 | 18.7% | 22.0% |
| LPM_Tree | 221.8 | 221.8 Mpps | 181 | 4.3% | 13.2% |
| SRPrioQueue | 214.5 | 214.5 Mpps | 41 | 2.6% | 0.6% |
| RateLimiter | 141.5 | 141.5 Mpps | 14 | 16.9% | 14.1% |

Nearly 100 elements
20% re-factored from Click modular router
Cover packet parsing, checksum, encap/decap, hash tables, prefix matching, rate limiting, crypto, packet scheduling…

Throughput: 200 Mpps / 100 Gbps

Mean delay: 0.19 us, max delay: 0.8 us

Mean LoC: 80, max LoC: 196

# Sample network functions

## Each NF takes about one week to develop

| Network Function | Lines of Code * | Number of Elements | Resource LE % | Resource BRAM % |
|---|---|---|---|---|
| Pkt generator | 13 | 6 | 16% | 12% |
| Pkt capture | 12 | 11 | 8% | 5% |
| OpenFlow firewall | 23 | 7 | 32% | 54% |
| IPSec gateway | 37 | 10 | 35% | 74% |
| L4 load balancer | 42 | 13 | 36% | 38% |
| pFabric scheduler | 23 | 7 | 11% | 15% |

* ClickNP configuration file

# Case study: IPSec gateway (1)

## Why IPSec datapath offloading?

CPU is the bottleneck for computation intensive processing

Even with AES-NI, software AES processing is far from 40 Gbps

Currently X86 has no SHA-1 acceleration instructions

# Case study: IPSec gateway (1)

## Why IPSec datapath offloading?

CPU is the bottleneck for computation intensive processing

Even with AES-NI, software AES processing is far from 40 Gbps

Currently X86 has no SHA-1 acceleration instructions

## Challenges to FPGA offloading

AES

- Single AES element throughput is 27.8 Gbps (< 40 Gbps)

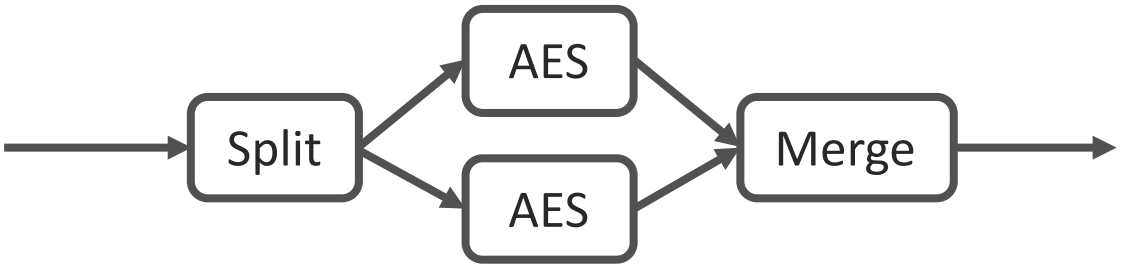
SHA-1

- Packet payload is split into 64-byte data blocks
- Dependency between successive data blocks in a packet
- Only 1.07 Gbps if processed sequentially!

# Case study: IPSec gateway (2)

## Solution: Parallelizing in FPGA

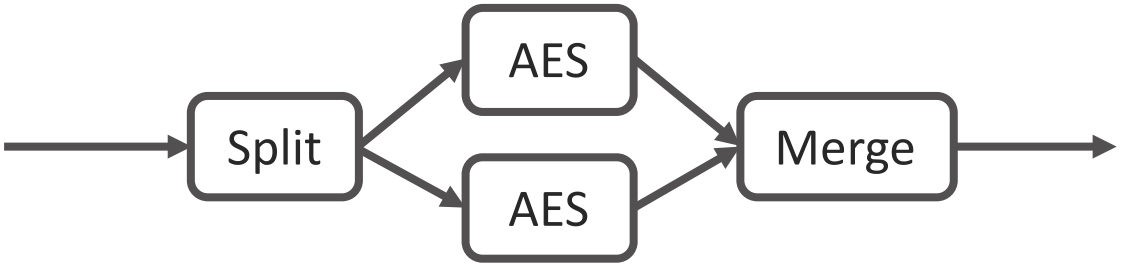
## AES-256-CTR: Leverage data parallelism



# Case study: IPSec gateway (2)

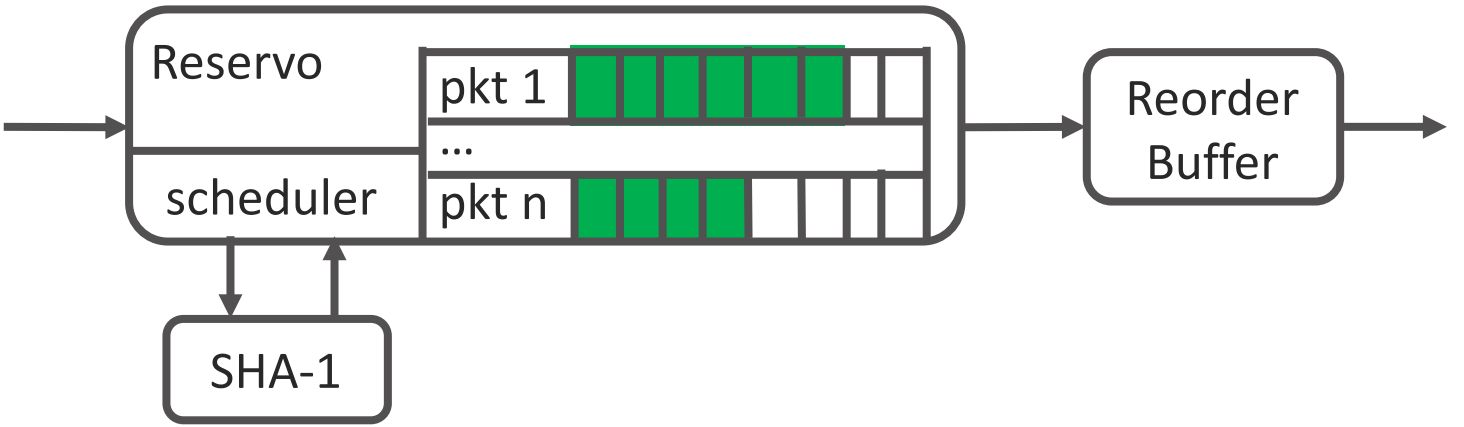
## Solution: Parallelizing in FPGA

## AES-256-CTR: Leverage data parallelism




## SHA-1: Leverage packet-level parallelism

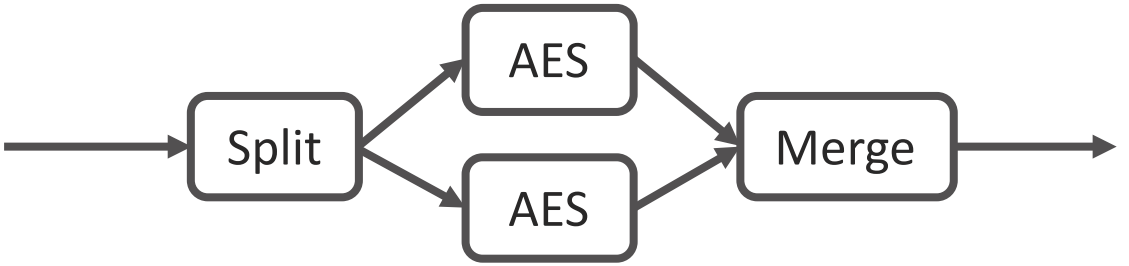
Process data blocks from different packets in parallel



# Case study: IPSec gateway (2)

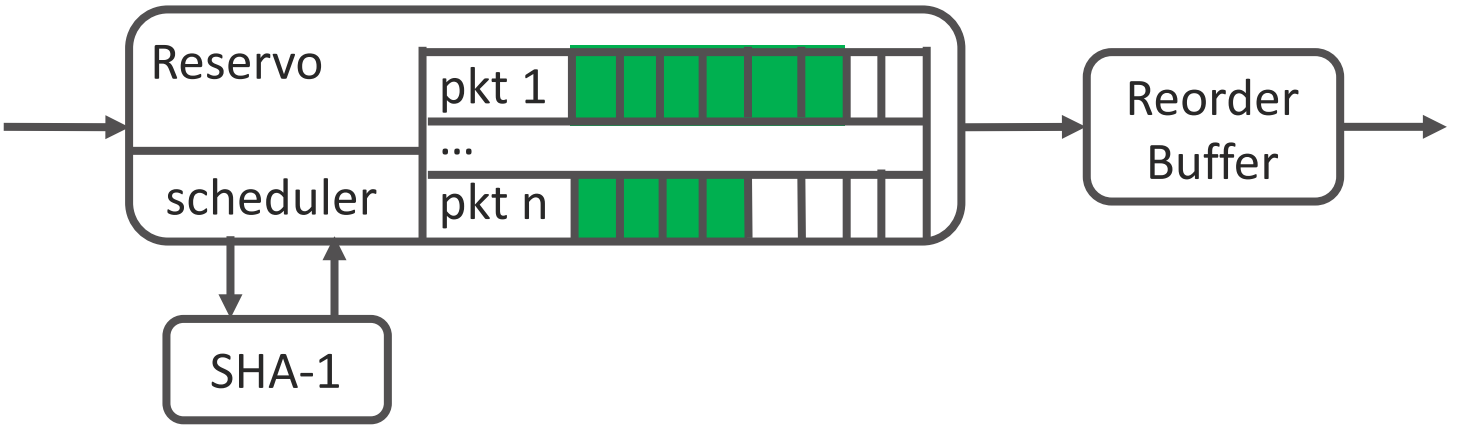
## Solution: Parallelizing in FPGA

## AES-256-CTR: Leverage data parallelism




## SHA-1: Leverage packet-level parallelism

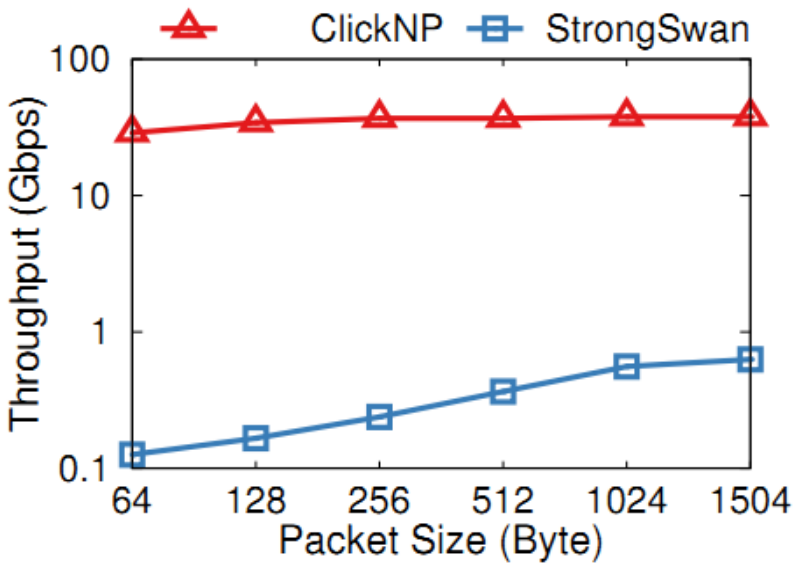
Process data blocks from different packets in parallel

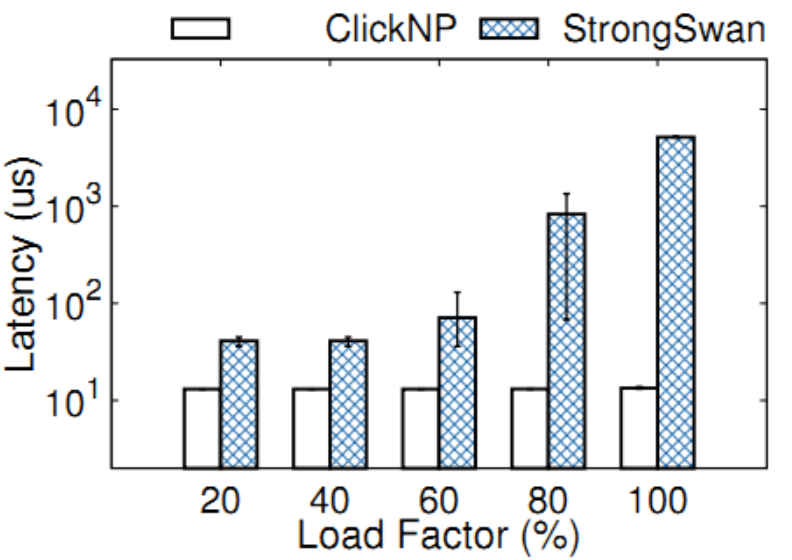


# Case study: IPSec gateway (3)

## Single-tunnel performance

| | ClickNP | StrongSwan / Linux (out of the box) |
|---|---|---|
| Throughput | 37.8 Gbps | 628 Mbps |
| Latency | 13 us (stable) | 50us ~ 5ms |



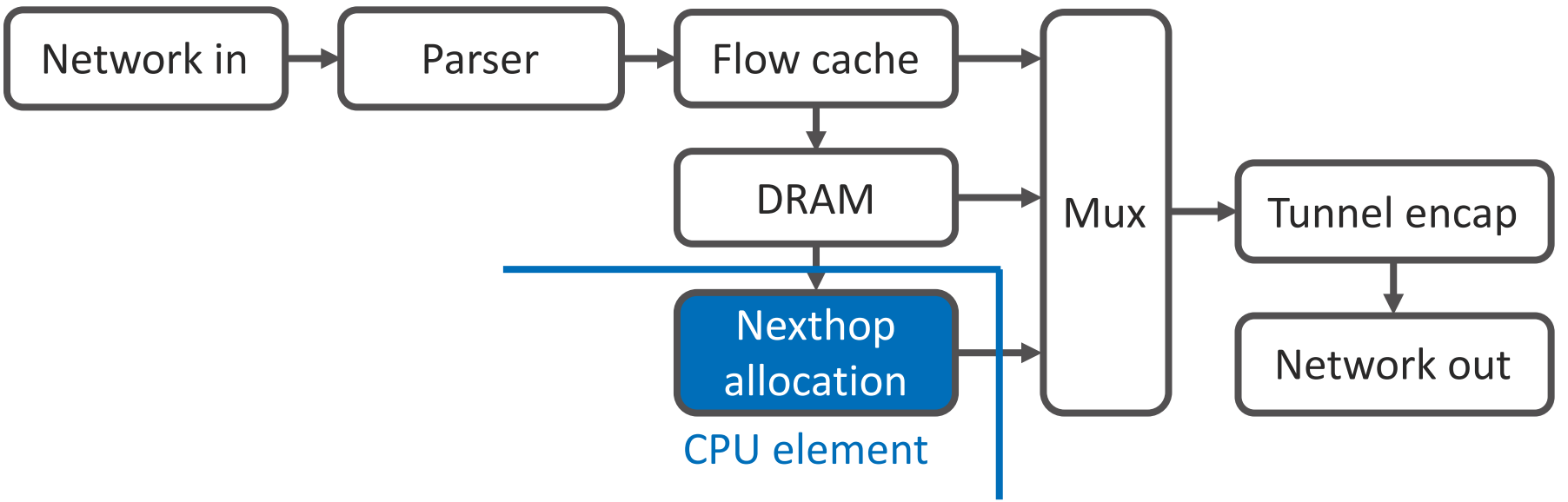



# Case study: L4 load balancer (1)

## Requires per-flow states for many flows

32M concurrent flows in off-chip DRAM

16K flow cache in on-chip BRAM



# Case study: L4 load balancer (1)

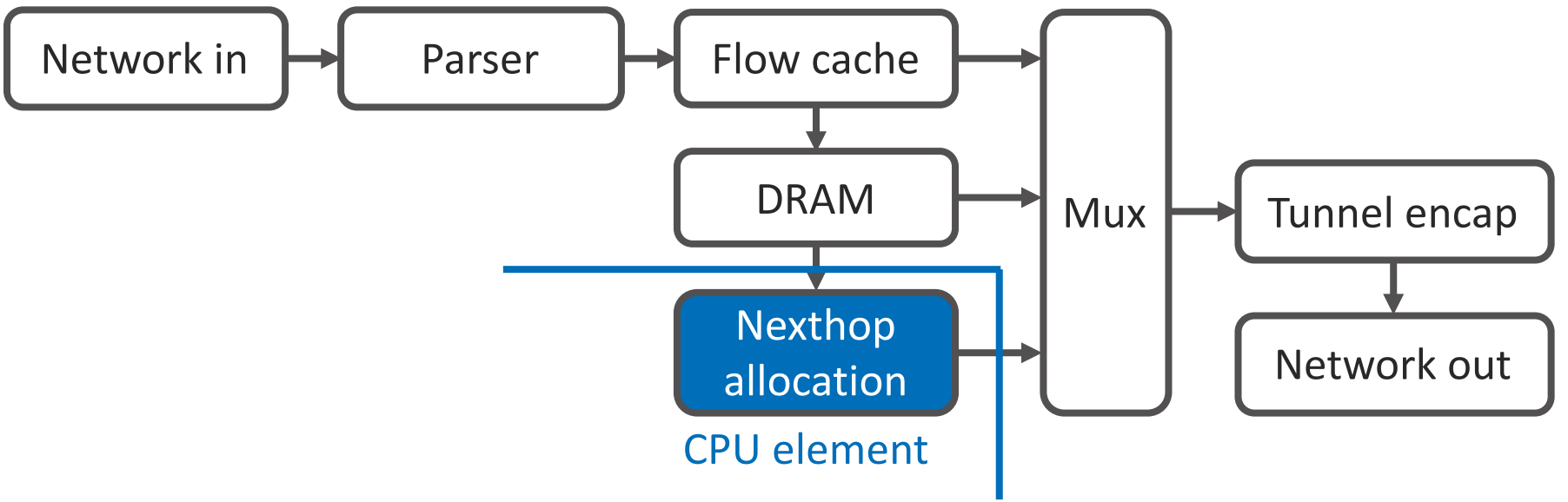
## Requires per-flow states for many flows

32M concurrent flows in off-chip DRAM

16K flow cache in on-chip BRAM

## Requires complex nexthop allocation policy

Nexthop allocation in CPU element: joint CPU/FPGA processing



# Case study: L4 load balancer (2)

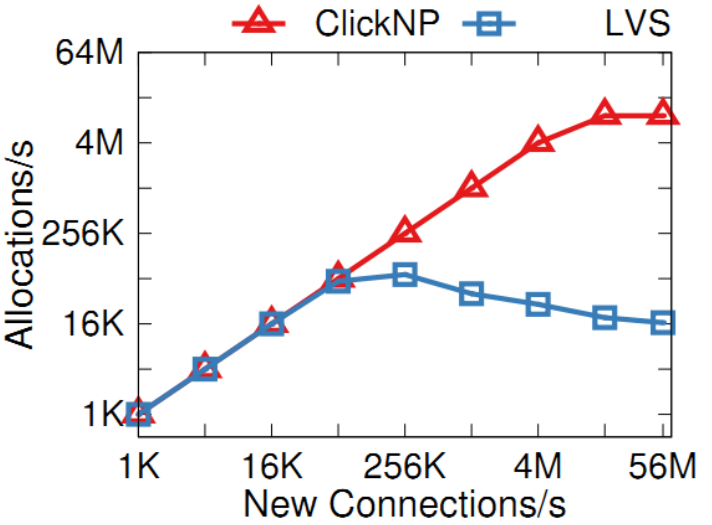
## Performance

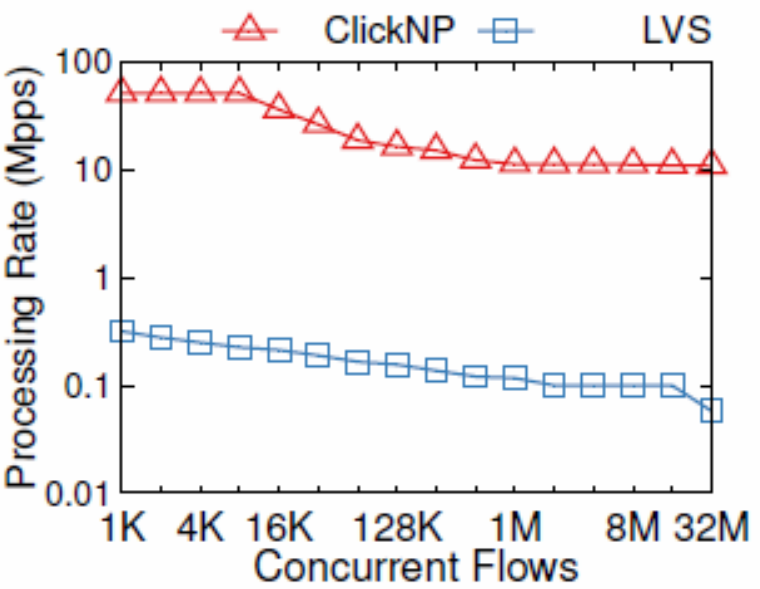
10M new flows per second

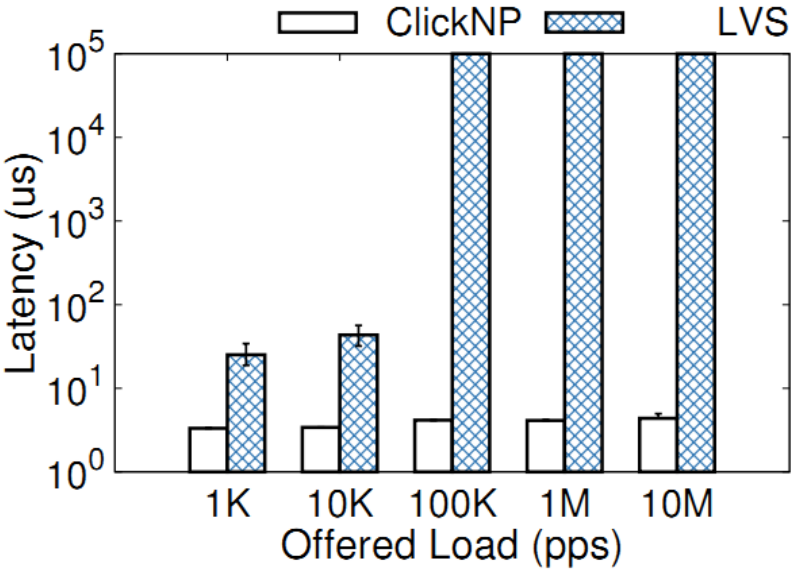
51Mpps for 8K flows

11Mpps for 32M flows

4us stable latency



# Area cost

## Overhead compared to hand-written Verilog

| NetFPFA Function | Resource Utilization Min / Max | | |
|---|---|---|---|
| | LUTs | Registers | BRAMs |
| Input arbiter | 2.1x / 3.4x | 1.8x / 2.8x | 0.9x / 1.3x |
| Output queue | 1.4x / 2.0x | 2.0x / 3.2x | 0.9x / 1.2x |
| Header parser | 0.9x / 3.2x | 2.1x / 3.2x | N/A |
| Openflow table | 0.9x / 1.6x | 1.6x / 2.3x | 1.1x / 1.2x |
| IP checksum | 4.3x / 12.1x | 9.7x / 32.5x | N/A |
| Encap | 0.9x / 5.2x | 1.1x / 10.3x | N/A |

# Area cost

## Overhead compared to hand-written Verilog

| NetFPFA Function | Resource Utilization Min / Max | | |
|---|---|---|---|
| | LUTs | Registers | BRAMs |
| Input arbiter | 2.1x / 3.4x | 1.8x / 2.8x | 0.9x / 1.3x |
| Output queue | 1.4x / 2.0x | 2.0x / 3.2x | 0.9x / 1.2x |
| Header parser | 0.9x / 3.2x | 2.1x / 3.2x | N/A |
| Openflow table | 0.9x / 1.6x | 1.6x / 2.3x | 1.1x / 1.2x |
| IP checksum | 4.3x / 12.1x | 9.7x / 32.5x | N/A |
| Encap | 0.9x / 5.2x | 1.1x / 10.3x | N/A |

# Area cost

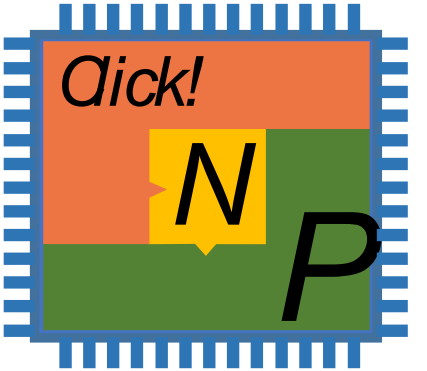
## Overhead compared to hand-written Verilog

| NetFPFA Function | Resource Utilization Min / Max | | |
|---|---|---|---|
| | LUTs | Registers | BRAMs |
| Input arbiter | 2.1x / 3.4x | 1.8x / 2.8x | 0.9x / 1.3x |
| Output queue | 1.4x / 2.0x | 2.0x / 3.2x | 0.9x / 1.2x |
| Header parser | 0.9x / 3.2x | 2.1x / 3.2x | N/A |
| Openflow table | 0.9x / 1.6x | 1.6x / 2.3x | 1.1x / 1.2x |
| IP checksum | 4.3x / 12.1x | 9.7x / 32.5x | N/A |
| Encap | 0.9x / 5.2x | 1.1x / 10.3x | N/A |

## Area cost will be less of a concern

Quickly growing FPGA capacity: 1x (2013), 2.5x (2015), 10x (2018)

# Conclusion




## FPGA in the cloud

Fast, general and mature

## ClickNP: *programming hardware as software*

*Highly flexible*: Program with high-level language

*Modular*: Click-like abstractions

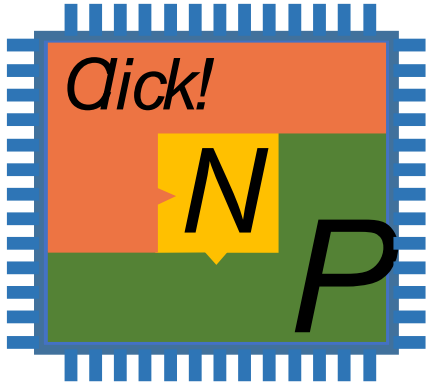
*High performance*: Utilize massive parallelism in FPGA

*Joint CPU/FPGA packet processing*

## Start from network processing, but it goes beyond…

General framework for programming on FPGA

Azure storage, Bing search, machine learning…




# Thank you!
# Questions!

Check out our demo on Thursday!

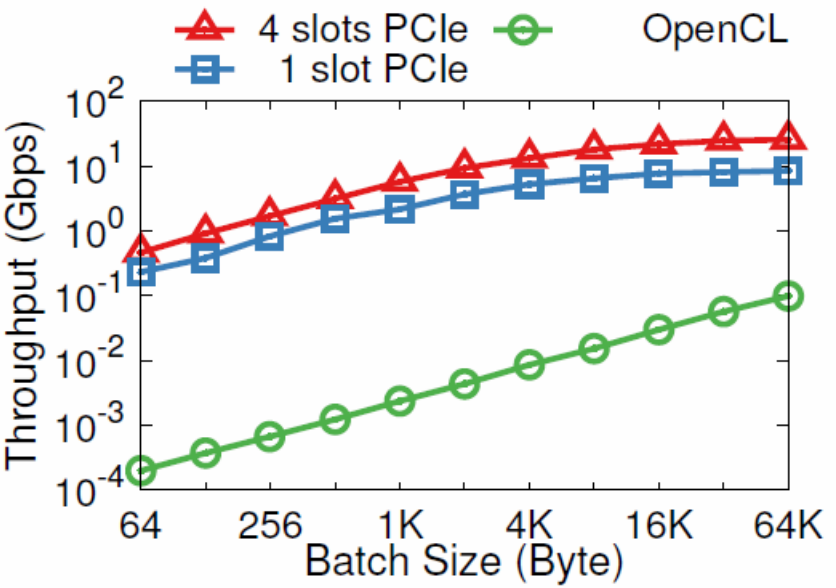
# Acceleration in host

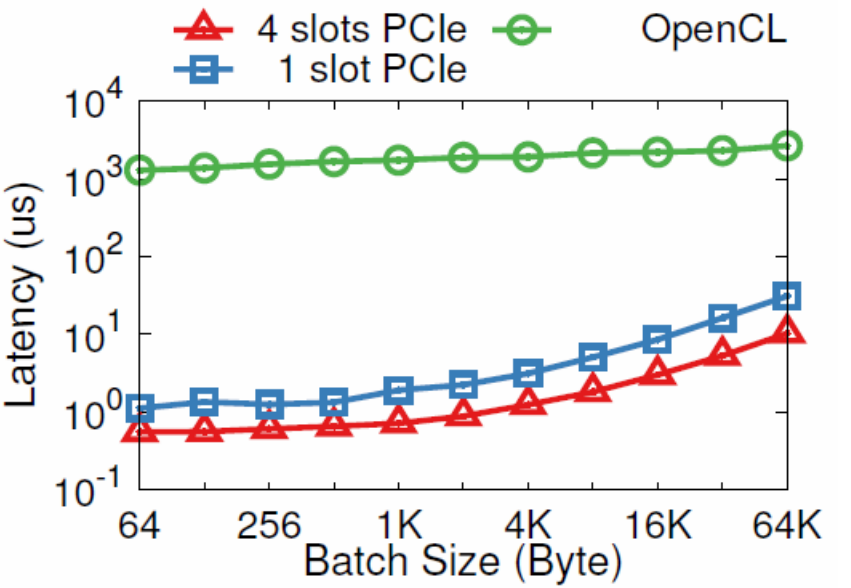
## Offloading software NFs to accelerator




| | GPU | NP | FPGA |
|---|---|---|---|
| Throughput | High | High | High |
| Latency | High | Low | Low |
| Power | High | Low | Low |
| General computing | Yes | No | Yes |

# Performance of PCIe I/O channel



# High level synthesis not sufficient

## HLS tools (e.g. Vivado HLS)

- Only auxiliary tools for HDL tool chains
- Generated hardware modules (IP cores) need to be integrated manually

## OpenCL (e.g. Altera)

- The OpenCL programming model is designed for batch processing in GPU => high latency
- Packet processing should use stream processing model
- Does not support joint CPU/FPGA processing well

## Click2NetFPGA

- Low performance because not fully pipelined
- Unable to update configuration while data plane running

# cross-platform toolchain

## three parts of source files

Define *elements* → ClickNP extended C

Define a *configuration* of elements → ClickNP script language

Host manager program → C/C++

## *make* tool to glue all compile procedures

Source files (.cl/.cfg/.cpp)→ ClickNP compiler (clc) → intermediate source files (.cl/.cpp)

FPGA program → aoc/Vivado

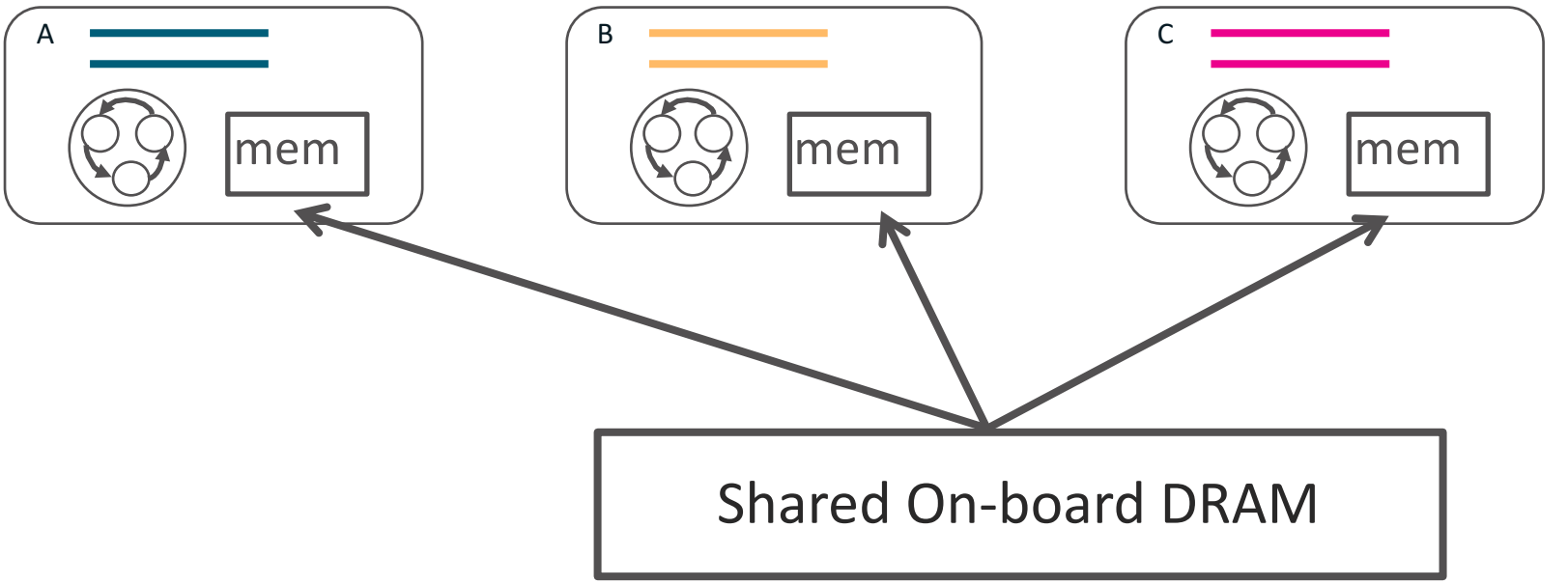
Host program → Microsoft cl + OpenCL libs

## platform supported

Windows/Linux, Altera/Xilinx

# Why do not simply use OpenCL?

## OpenCL is for batch processing on GPU



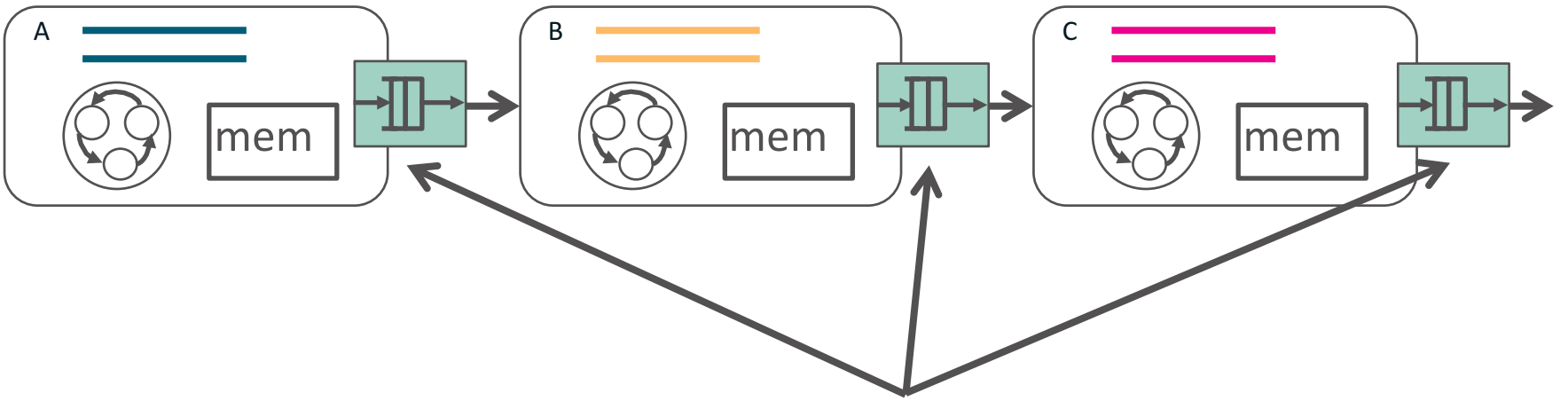

Communicate by sharing memory

*Shared memory is the bottleneck!*

*Batch processing has large latency!*

# Programming model

## ClickNP is for stream processing on FPGA



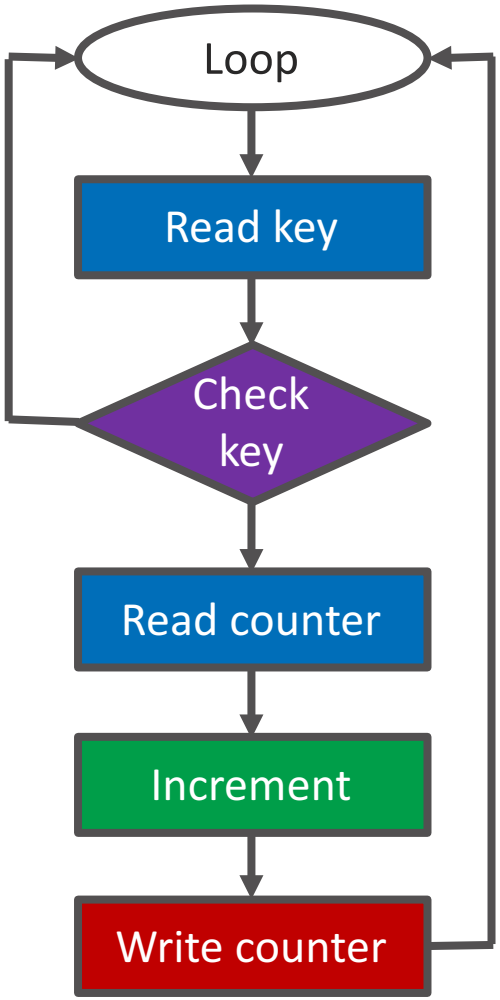

Channels: more efficient than shared memory

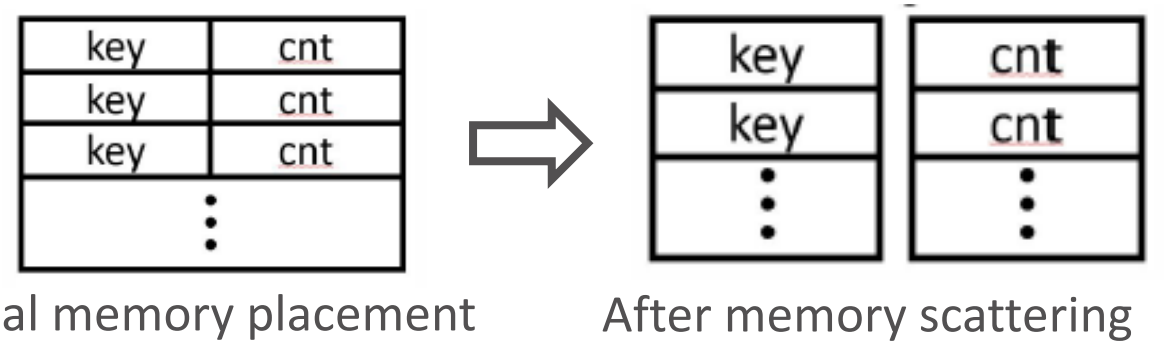
*Do not communicate by sharing memory; instead, share memory by communicating.*

-- The slogan of Go language

# Parallelism inside element (2)

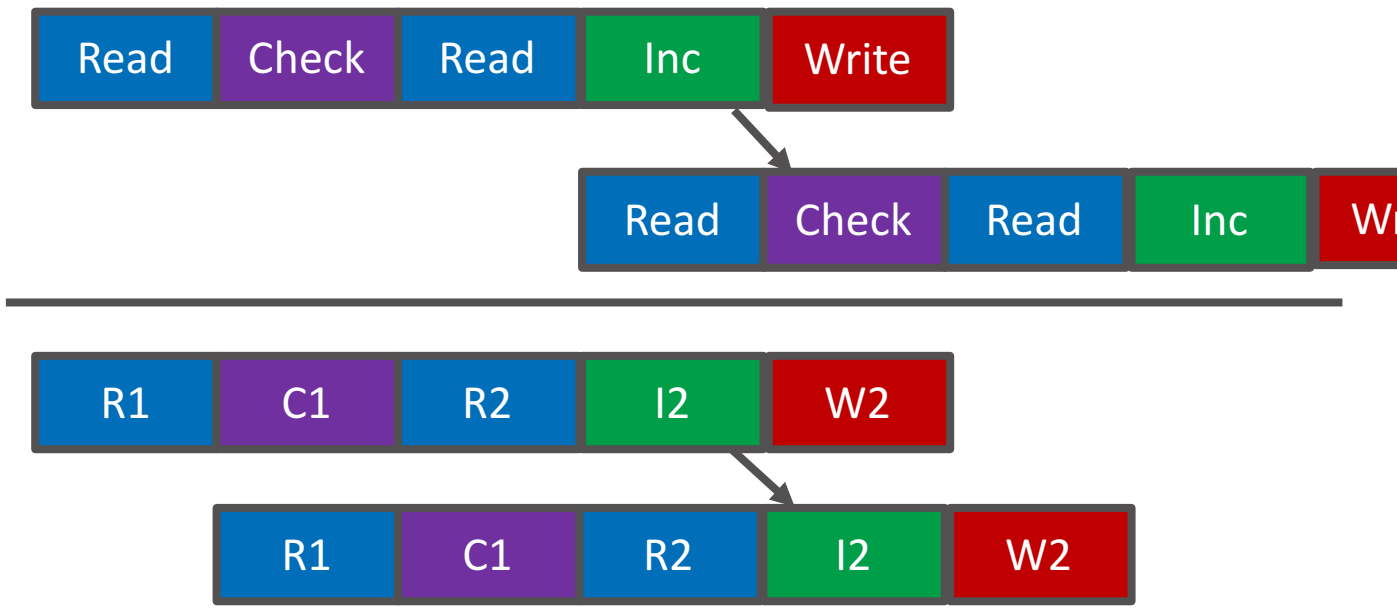
## Memory scattering: avoid false dependency



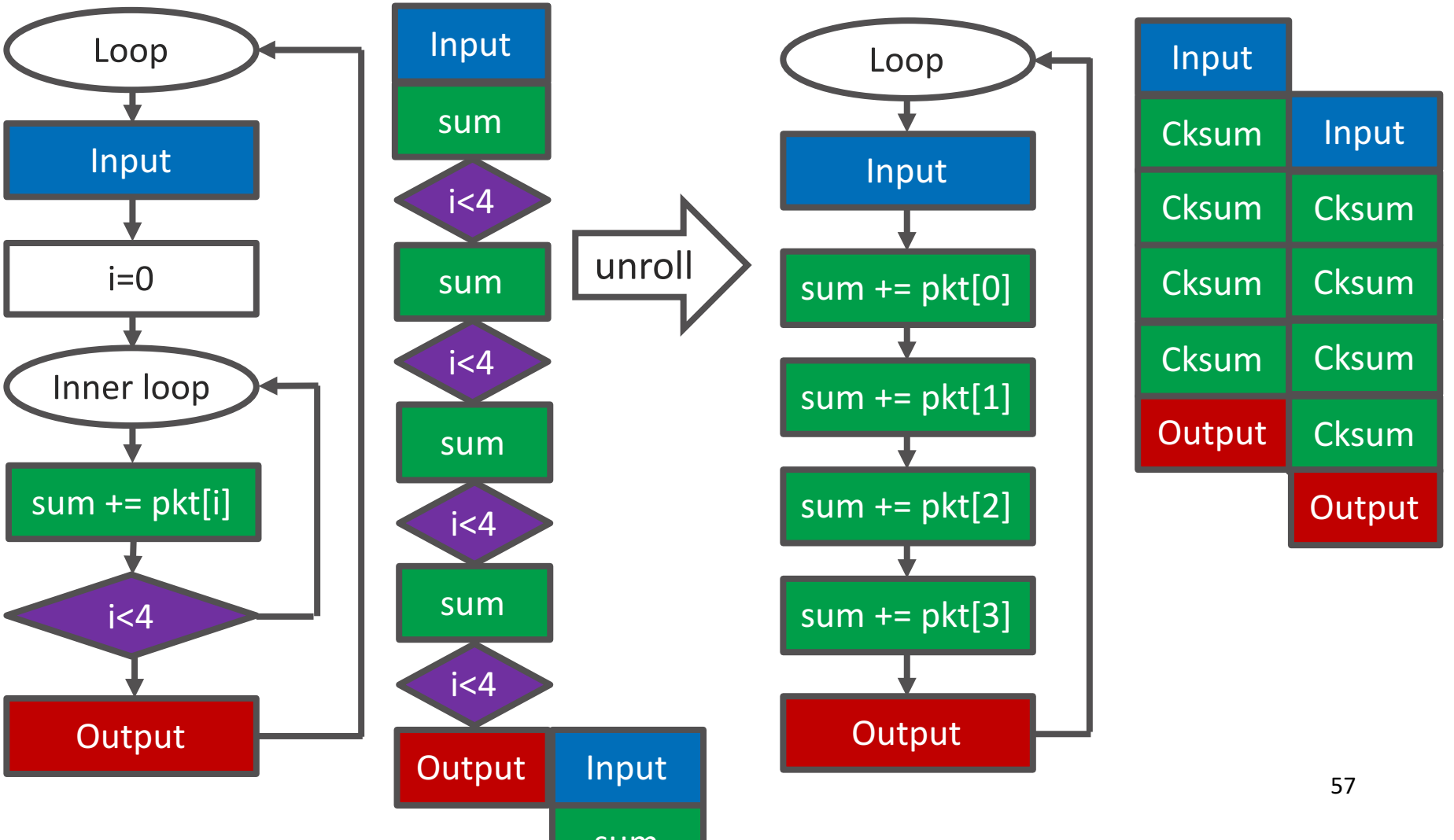

Normal memory placement

After memory scattering



# Parallelism inside element (3)

## Unroll loops to balance pipeline stages



# Parallelism inside element (4)

## Offload slow path to another element