Efficient P4+FPGA-based Forwarding for SCION, a Path-Aware Internet Architecture

A Software Engineer's Peek at P4 on FPGAs

Kamila Součková
Research Engineer
Network Security Group, ETH Zurich

1 Intro

2 Thinking about hardware

3 Building the SCION router

4 Lessons Learned

Intro

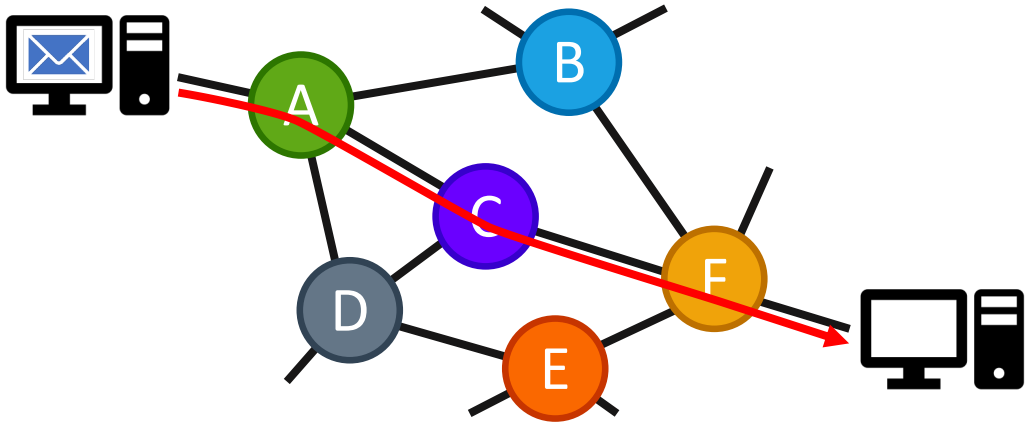
Why? How? What?

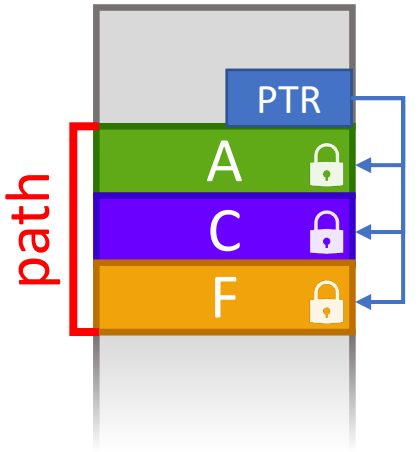
Why?

https://scion-architecture.net ● https://scionlab.org

Why?


SCION
SCALABILITY, CONTROL, AND ISOLATION
ON NEXT-GENERATION NETWORKS


A
B
C
D
E
F


PTR
A
C
F
path

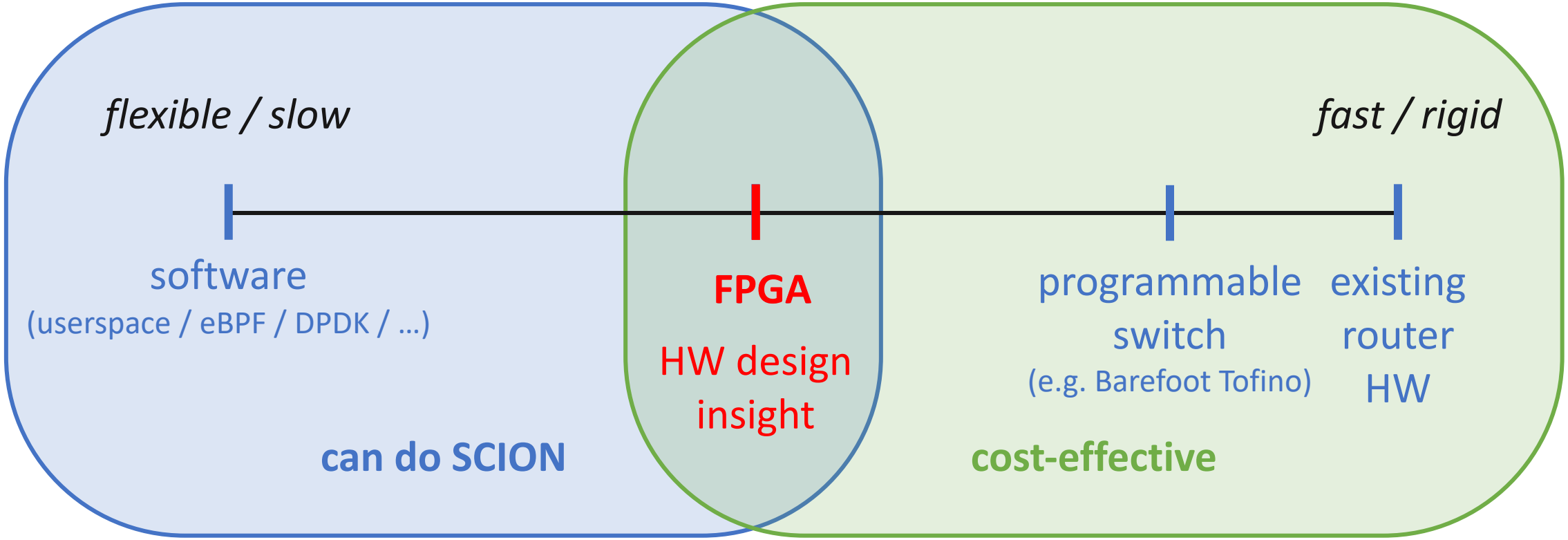
Why?

To go beyond research, it must be practical to build the routers.

- Is it practical and economical to implement it at high speeds?
- If so, how can we make the protocol easier to implement efficiently?

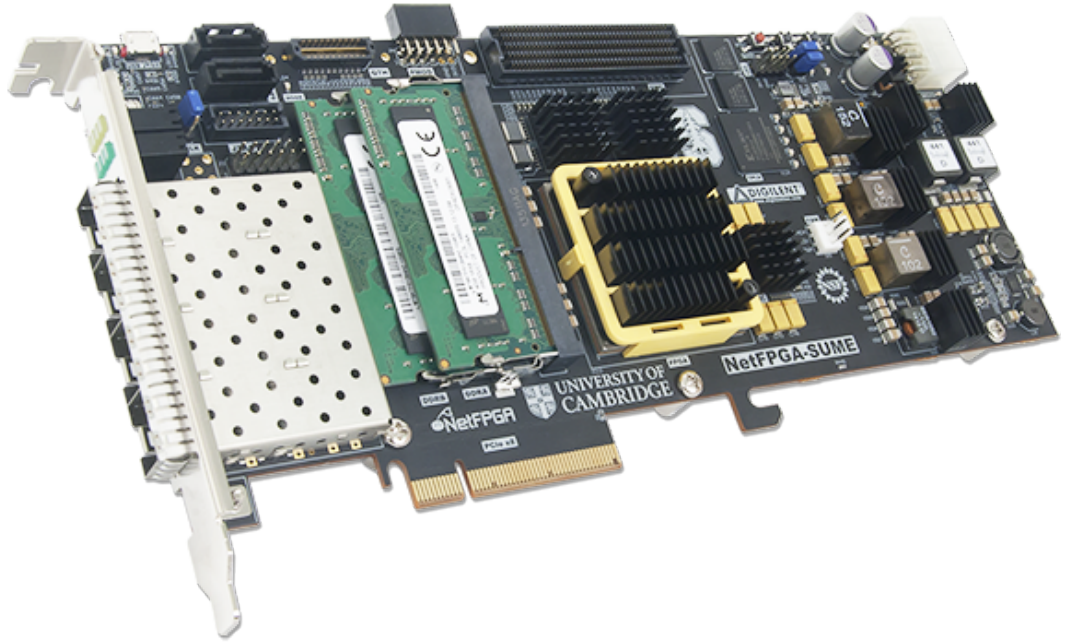
How?

Options for high-speed packet forwarding:



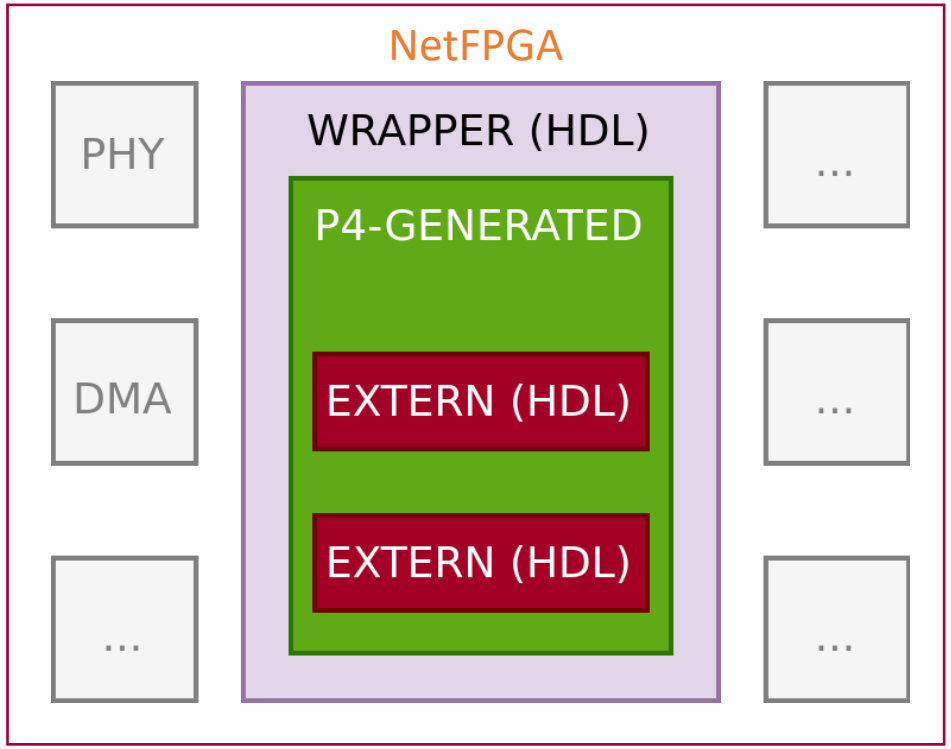
How?

NetFPGA SUME:



4x 10GbE, Xilinx Virtex 7 FPGA

P4 support (proof-of-concept):



What?

Performance target:

- 4 ports at 10 Gbps full duplex
  ⇒ total sustained throughput: **40 Gbps**
- line rate with smallest possible frames (86 B in SCION)
  ⇒ almost **60 Mpps**

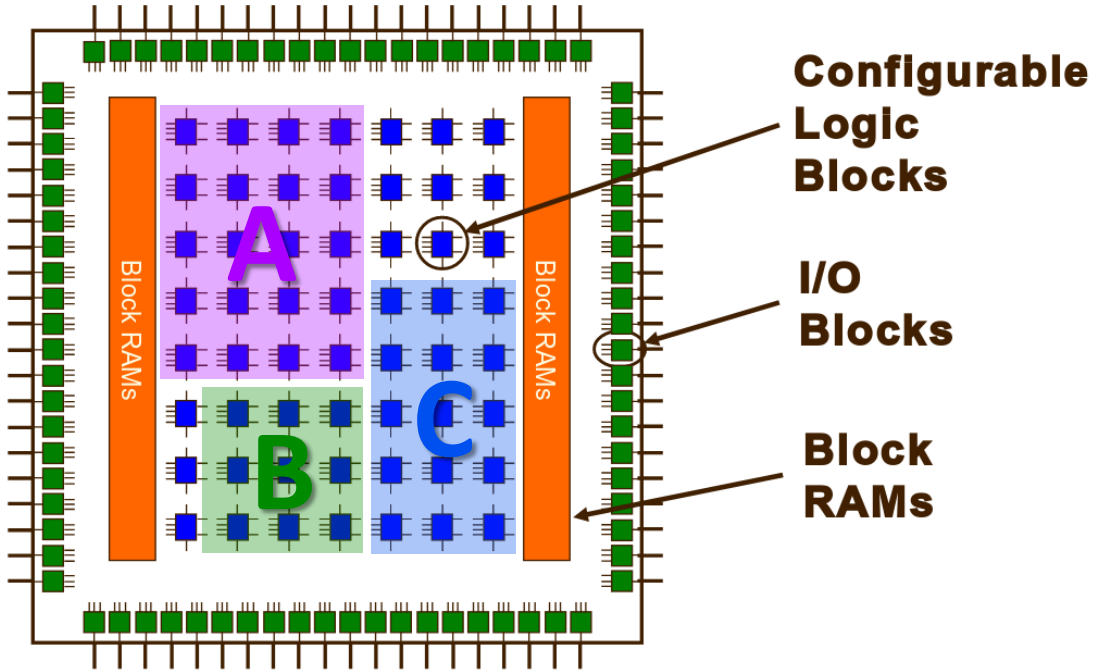
Thinking about hardware

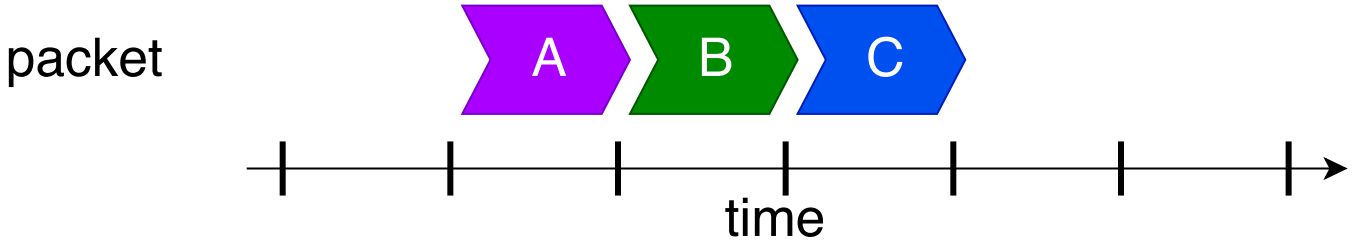
A software engineer’s perspective

Hardware vs Software

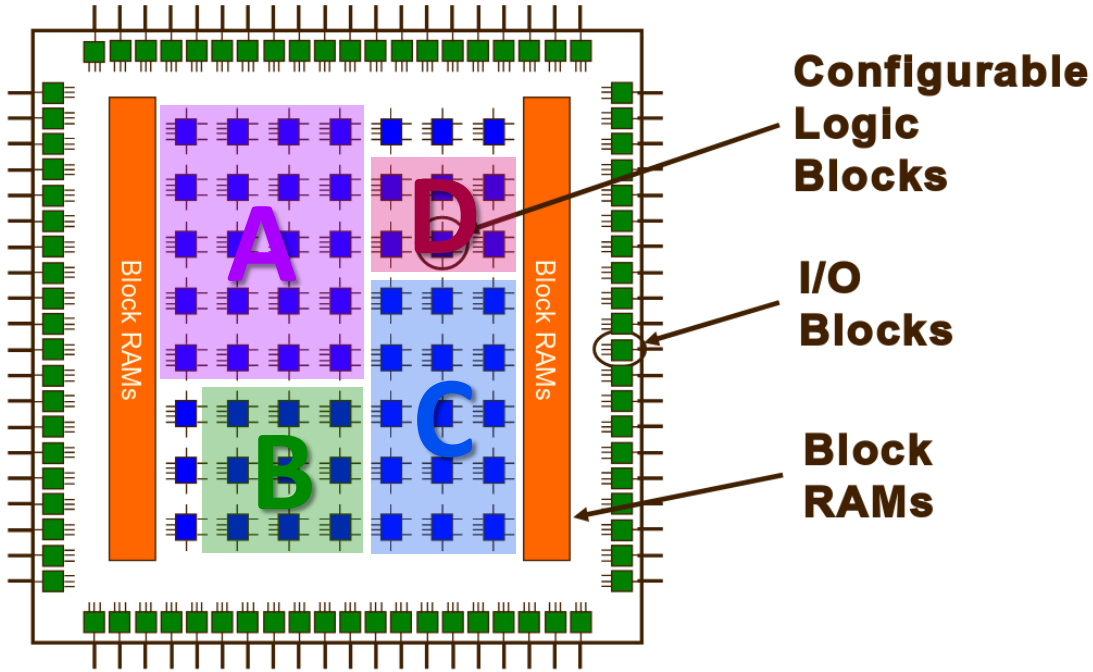
- software: sequence of instructions operates on data one at a time
- hardware: **fixed** circuit: all operations happen all the time, data flows through instructions

⇒ “think in space, not in time”

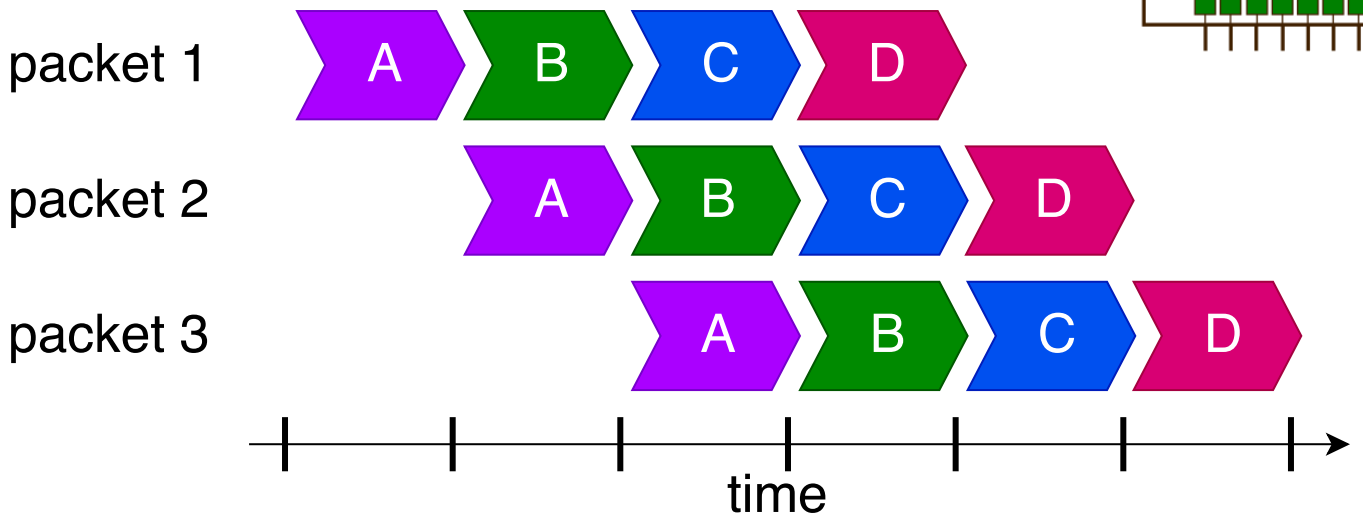




Hardware vs Software



- pipelining:



throughput:

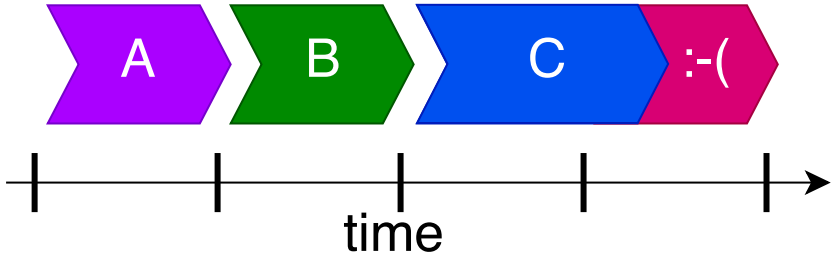
1 per cycle

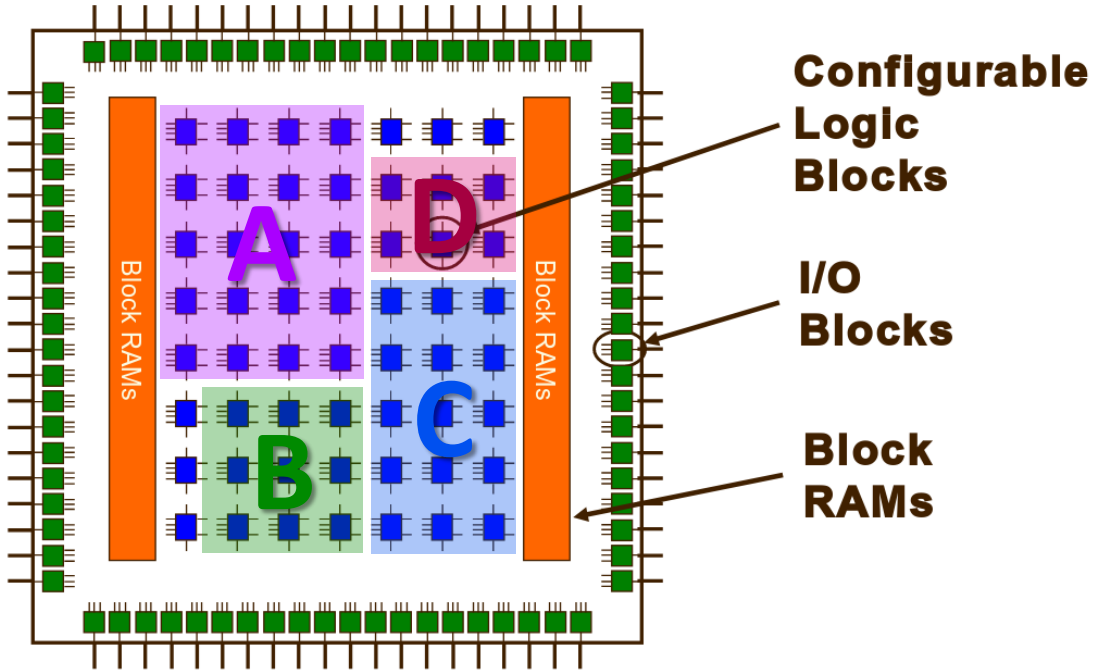
regardless of # of stages

Hardware vs Software

Too much logic in a single stage

⇒ large delay:



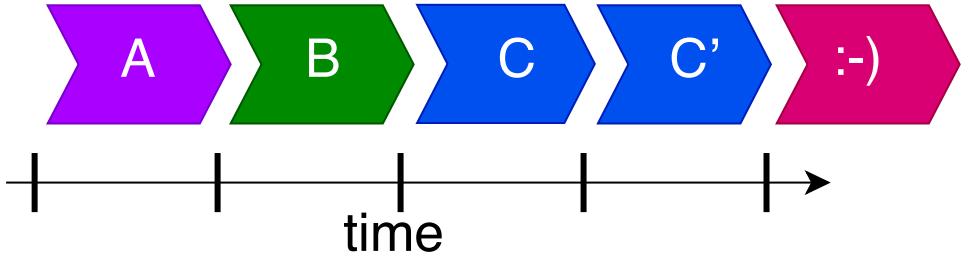


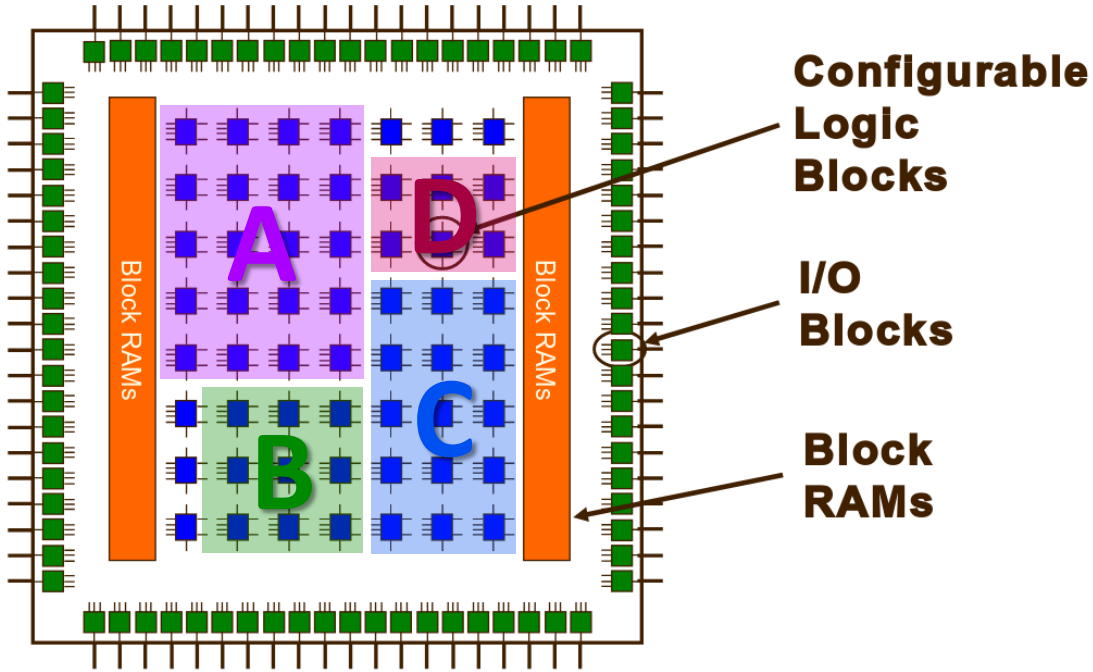
timing constraint violation ⇒ the circuit won't work at the intended speed

Hardware vs Software

Too much logic in a single stage

⇒ large delay:



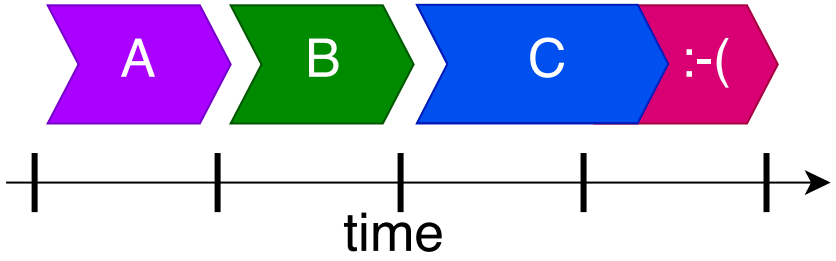


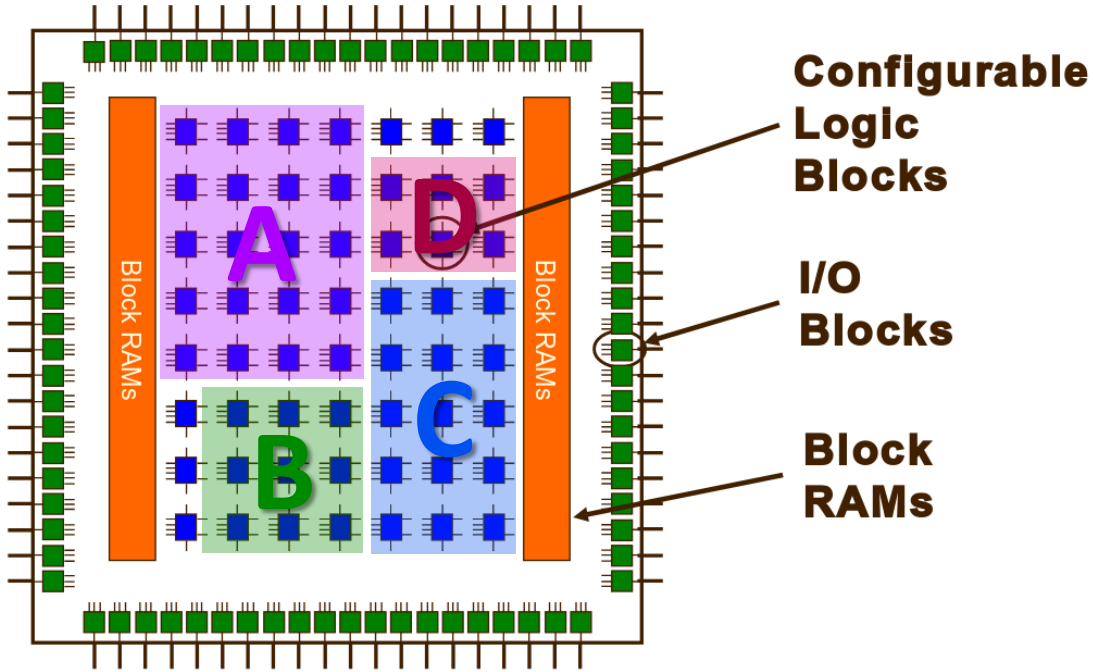
timing constraint violation ⇒ the circuit won't work at the intended speed

Hardware vs Software

Too much logic in a single stage

⇒ large delay:



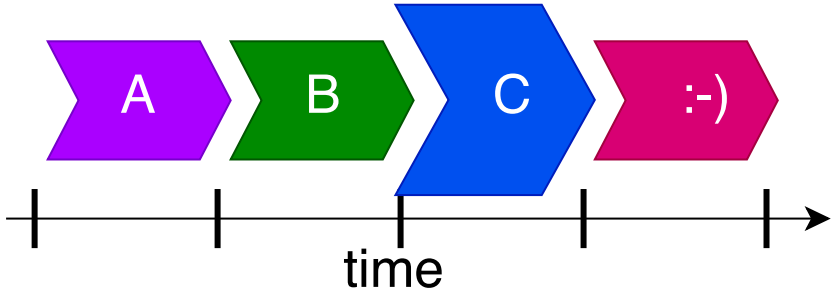


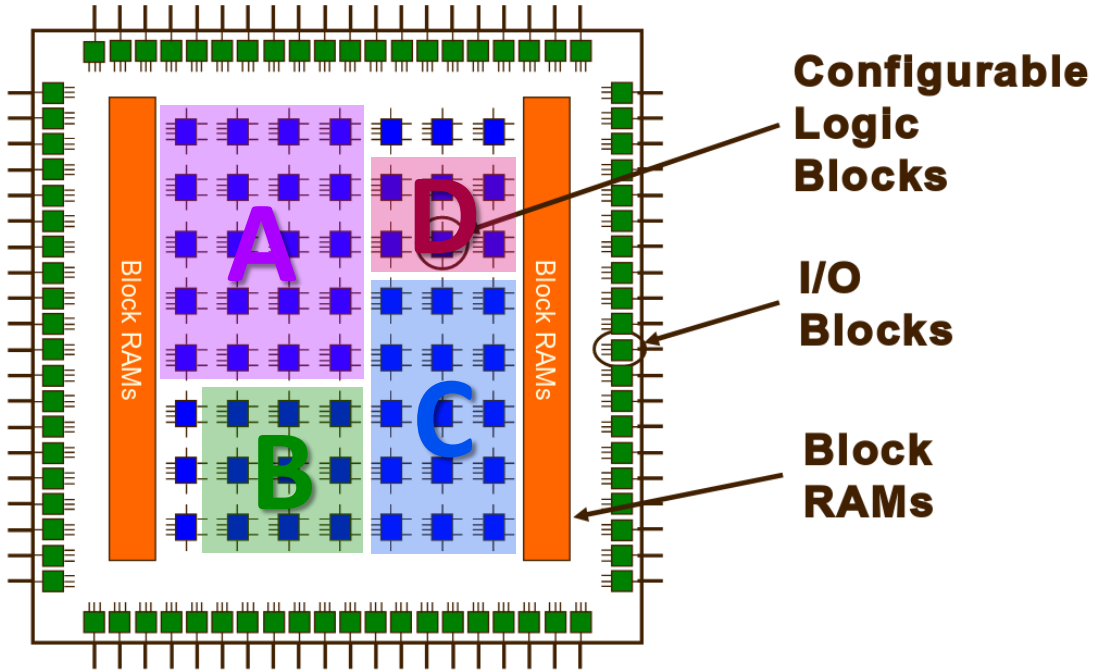
timing constraint violation ⇒ the circuit won't work at the intended speed

Hardware vs Software

Too much logic in a single stage

⇒ large delay:





timing constraint violation ⇒ the circuit won't work at the intended speed

Building the SCION router

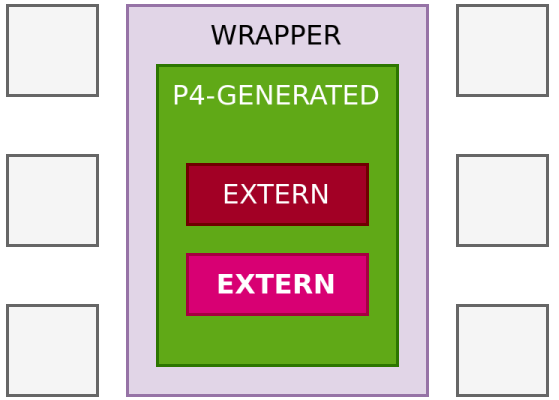
Tricks, challenges, lessons learned

Adding crypto

Mix and match: embed native in P4

SCION path hops are cryptographically authenticated (AES-CMAC)

```
@Xilinx_MaxLatency(10)
extern void my_aes128(
    in  bit<128> K,
    in  bit<128> data,
    out bit<128> result
);
```

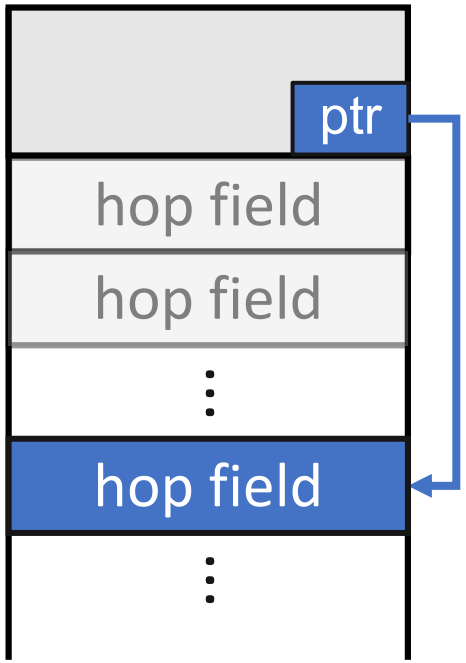


generates an interface in Verilog, you provide native implementation

- must be pipelined
- declare as `X_aes128` (naming matters)
- can only be called once per declaration (declare multiple if needed)

Coping with the SCION header

SCION header:



- contains the path ⇒ variable number* of hop fields in header
- pointer to current hop
- path needs to come out unchanged on the output

* can be **long**: at least up to 64

Obvious approaches:

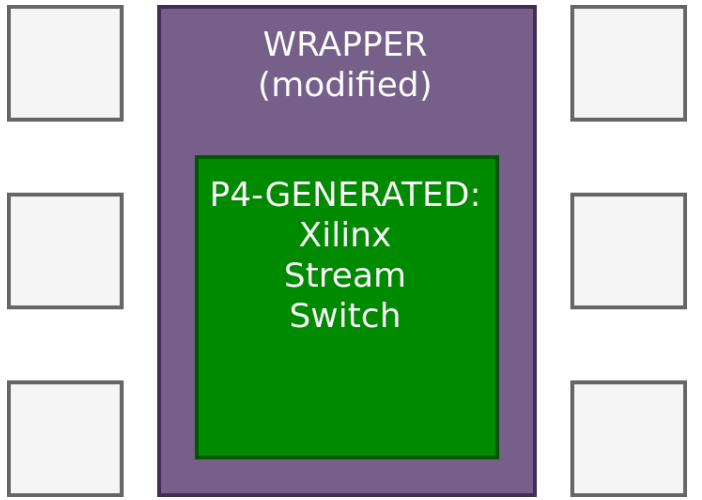
- ➢header stack ✗ not supported
- ➢varbit ✗ not supported

Sub-problem 1:
Outputting the unchanged path

Avoid needing to recreate the header on output:

packet_mod (Xilinx extension, requires modifying native wrapper):

```
parser ExampleModDeparser(packet_mod p, in headers_t h) {
    state start {
        p.update(h.ethernet);
        transition select(h.ethernet.ethertype) {
            ETHERTYPE_IPV4: deparse_ipv4;
            ETHERTYPE_IPV6: deparse_ipv6;
        }
    }
…
}
```



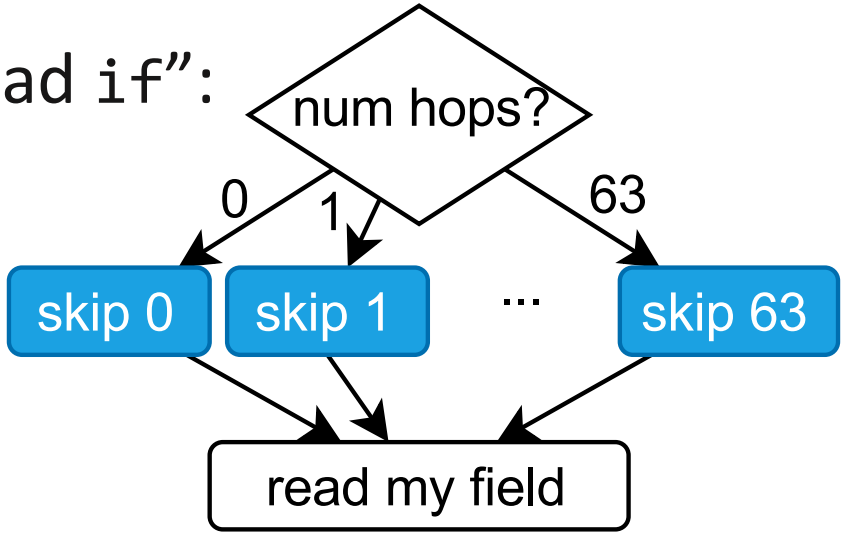
Mix and match: P4 is part of a larger design

Sub-problem 2:
Getting to my hop field

Strategy: Skip over unused hops, read my hop

- `pkt.advance(x)` ✗ only works when *x* is a compile-time constant
- loop ✗ needs to be unrolled ⇒ very deep circuit :-(

- "big bad `if`":



Wide pipelines better than deep

But: requires a **lot** of FPGA area

Sub-problem 2:
Getting to my hop field

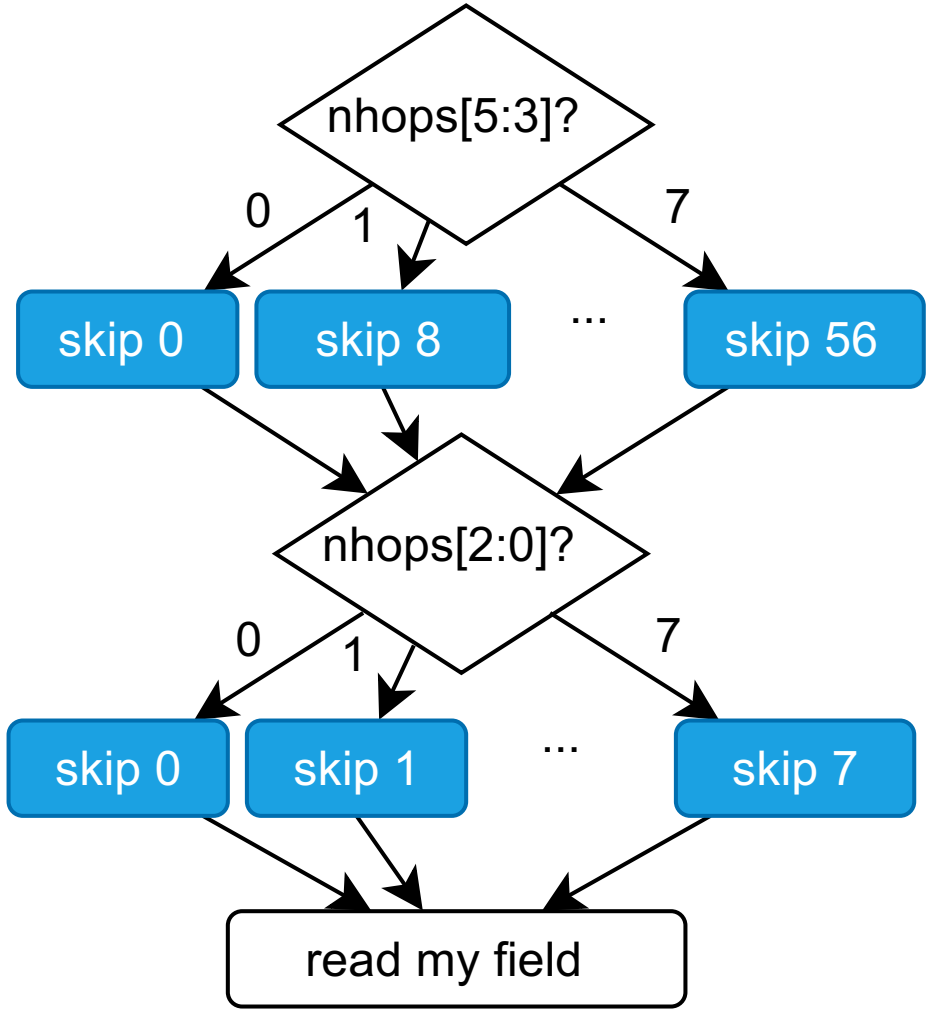
- "big bad `if`" **in two stages**:

  2x latency for $\sqrt{\text{area}}$

  less full FPGA

  ⇒ better placement

  ⇒ better performance

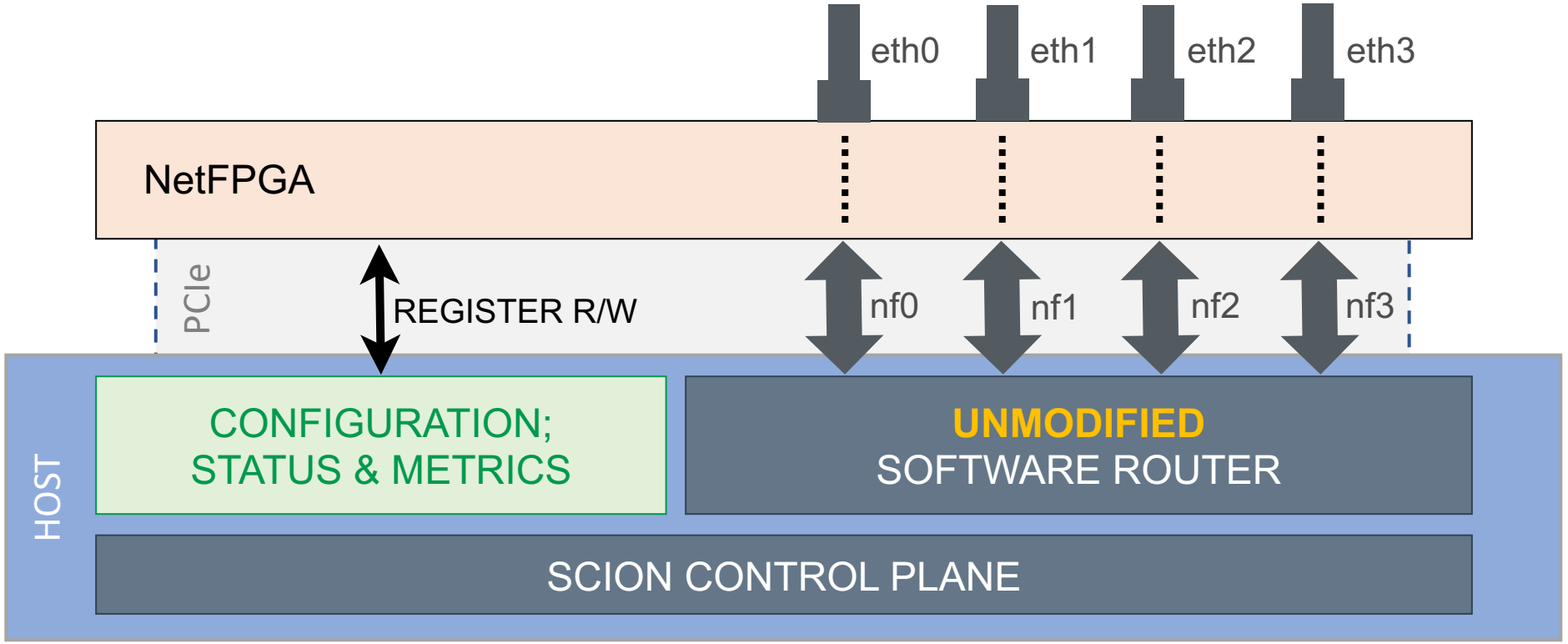


Depth vs width tradeoffs are worth thinking about

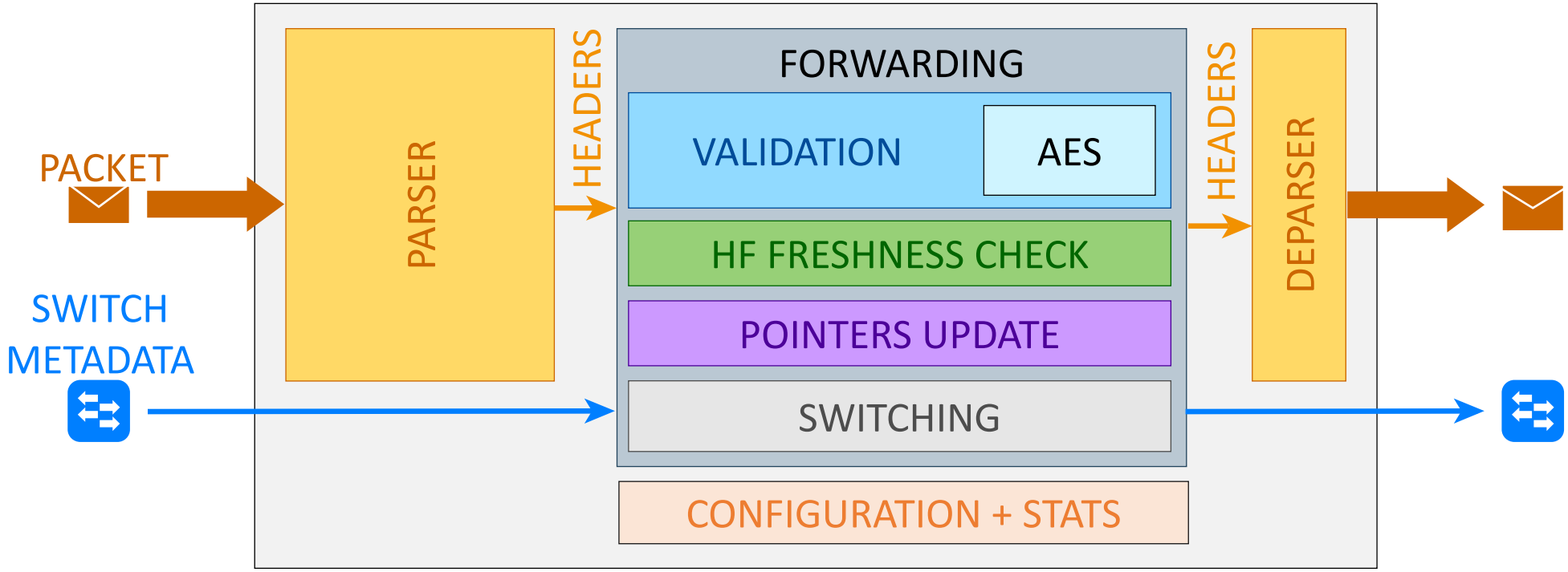
Reduce, Reuse, Recycle

Not everything belongs in HW

Expose network interfaces to the host,
pass through any packets that cannot be handled to SW



Area and timing constraints



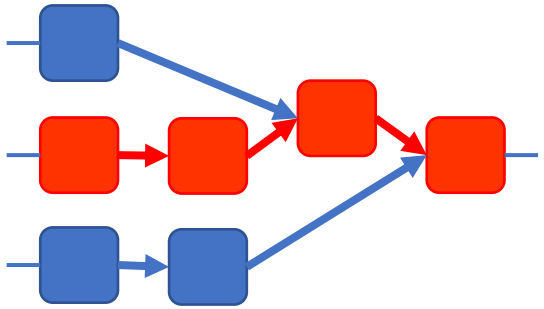
- inherent complexity + workarounds for bugs ⇒ lots of logic
- no control over pipeline stages, P4-NetFPGA compiler makes too few

⇒ couldn’t meet timing

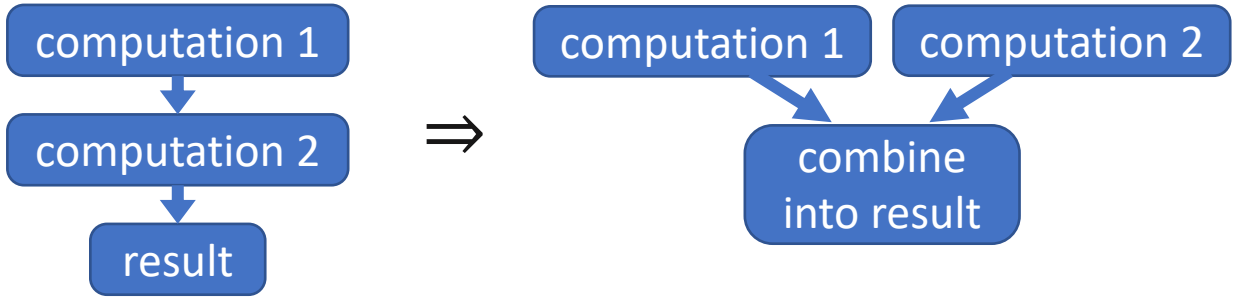
Meeting timing

Signal moving through circuit ⇒ data locality matters; parallelism is free; being late is Bad

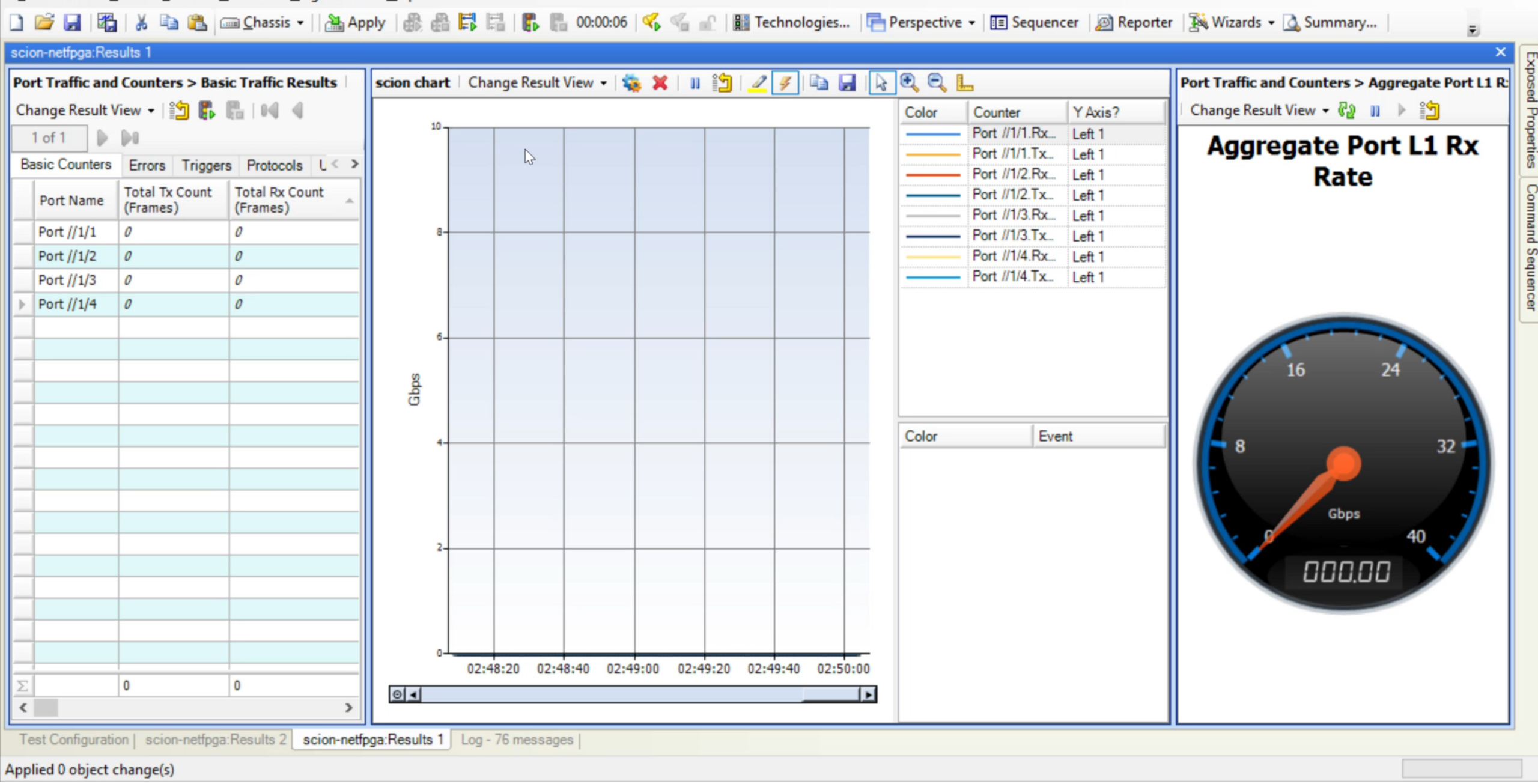
- *critical path* (maximum delay):



wider/parallel is (usually) better than deeper/serial:



- think about data locality / dependencies
  - small, simple, **self-contained** modules
  - **"tight" interfaces**: correct use of `in`/`out` (avoid `inout`); no extraneous parameters



https://github.com/AnotherKamila/scion-p4netfpga

Lessons learned

Next time I’m doing this…

Network protocol design

- **avoid variable length fields (really)**
- should be parseable without interpretation
  - a tag implicitly defining the next field’s length is bad
  - explicit lengths, not "continue" flags

easier to pipeline + parallelise

faster & cheaper HW

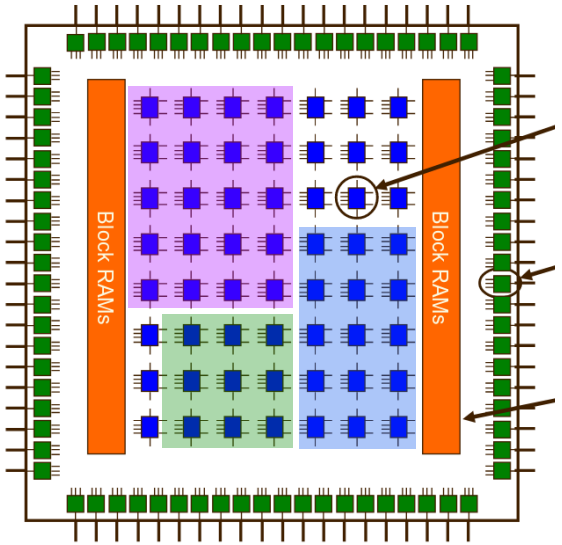
High-speed packet processing with P4

★★★★☆ *would buy again* :-)

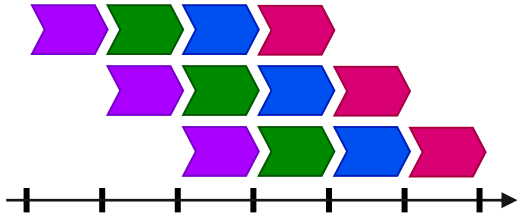
+ suitable even for complex protocols
+ useful abstraction: high-level, yet can run fast
+ pipelining and parallelism handled by compiler
- not really target-independent:
  - targets have different strengths and limitations
  - to get performance, code needs to be target-specific
  - conversion of P4 code to target-specific implementation is not obvious

FPGA-friendly P4 code

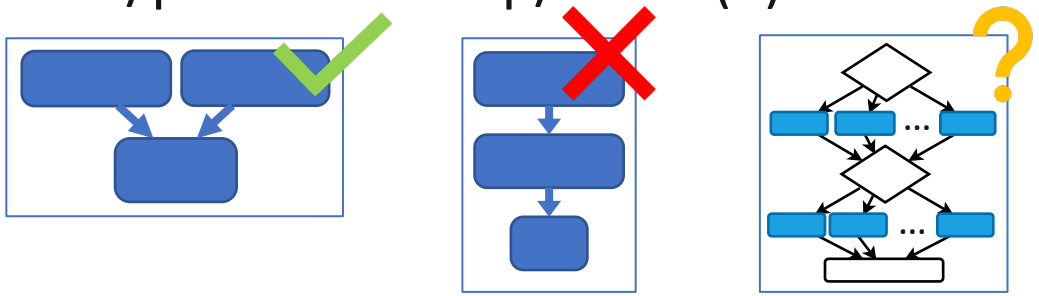
Key idea:
Signal moving through the circuit



pipelining



timing ⇒ wide/parallel > deep/serial (*)



⇒ small, self-contained modules
⇒ tight interfaces (`in`/`out` vs `inout`)

If I had a time machine…

- I would use P4
- I would better understand that P4-NetFPGA is a proof of concept ⇒ prototype early, prototype often
  - discover problems and workarounds earlier
- I would get paid for it ;-)

Thank You

email: skamila@ethz.ch
Matrix: @kamila:unchat.cat
Twitter: @AnotherKamila

https://scion-architecture.net