Automatic Compilation, Deployment & Debugging of DNNs on Cloud FPGAs

What are the DevOps besides the papers?

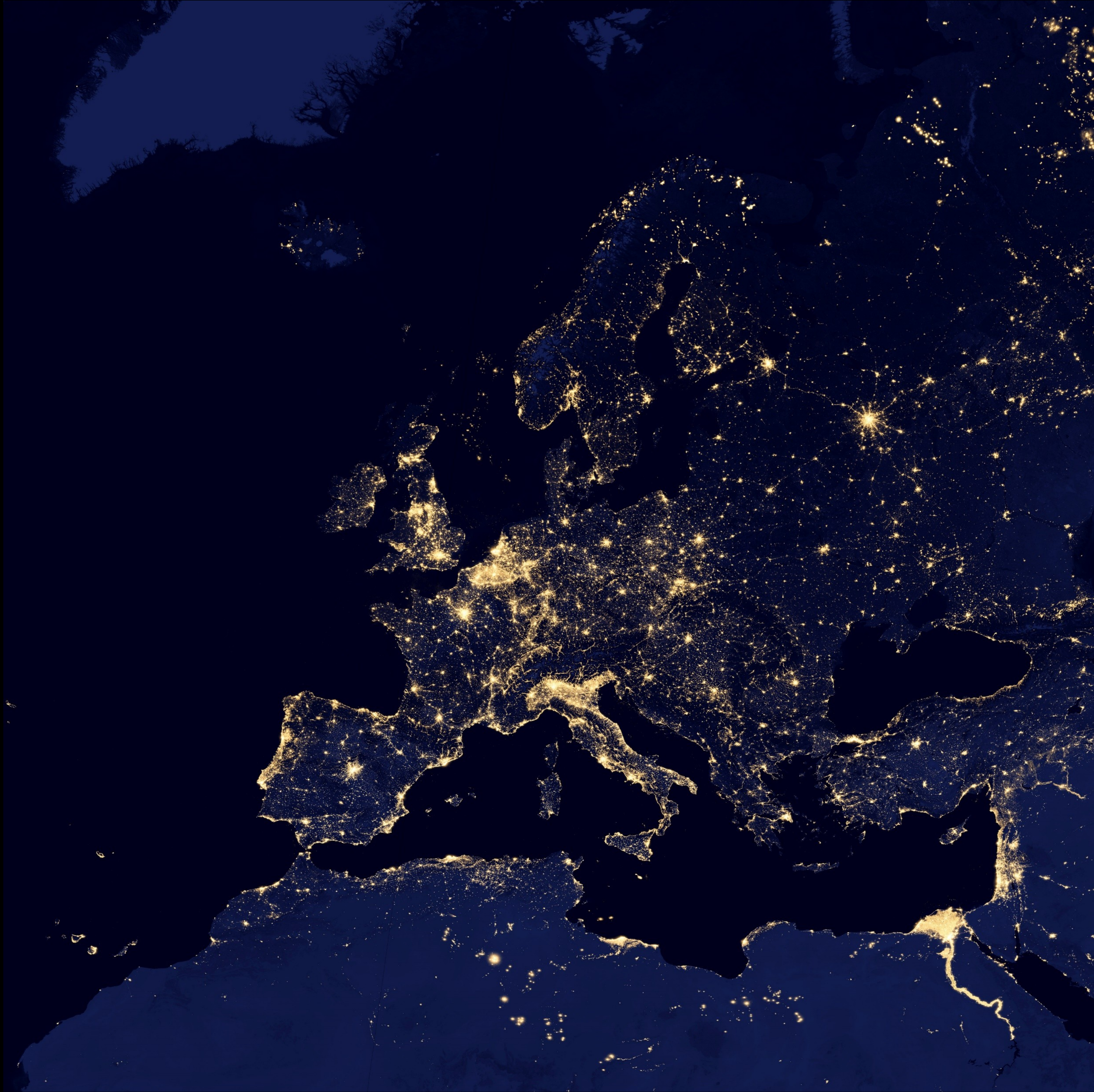
—

Burkhard Ringlein

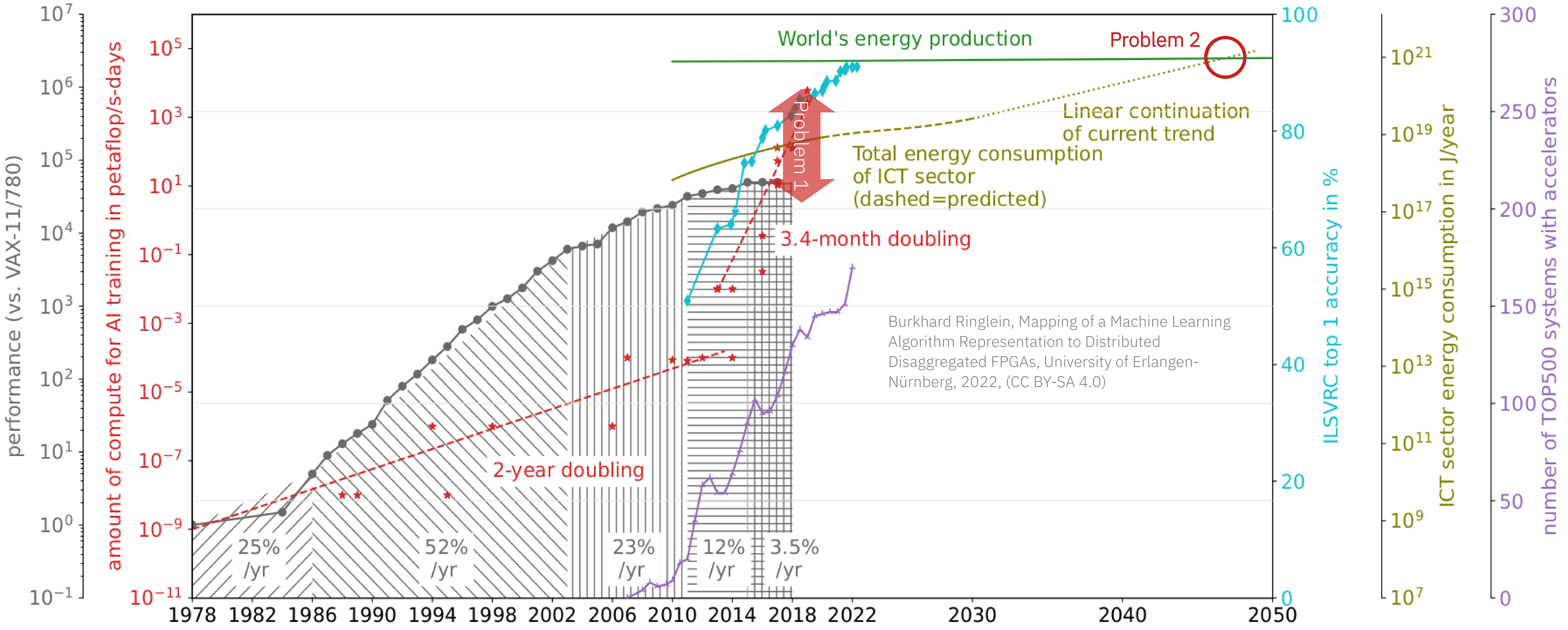
IBM Research – Europe

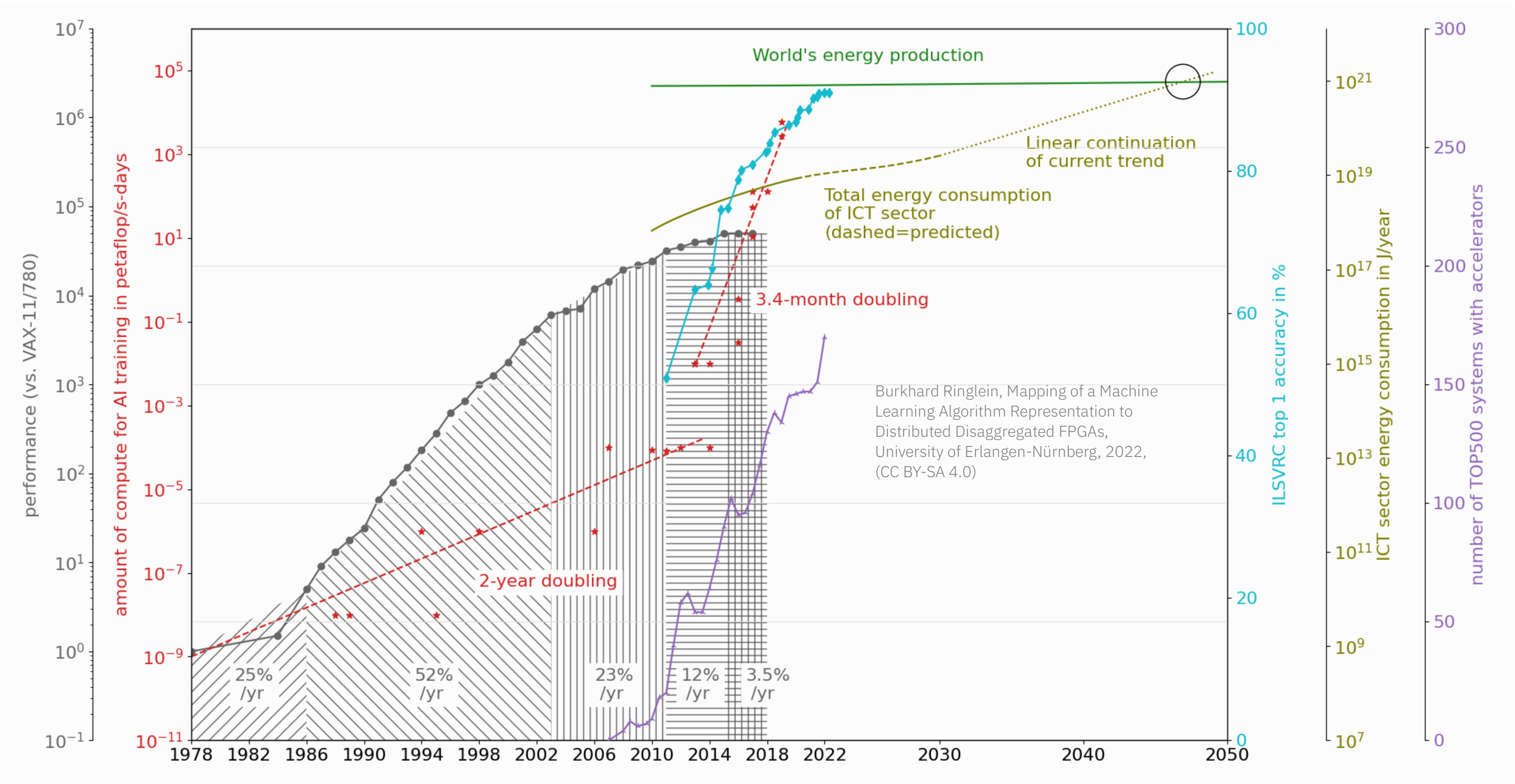
Zurich, Switzerland

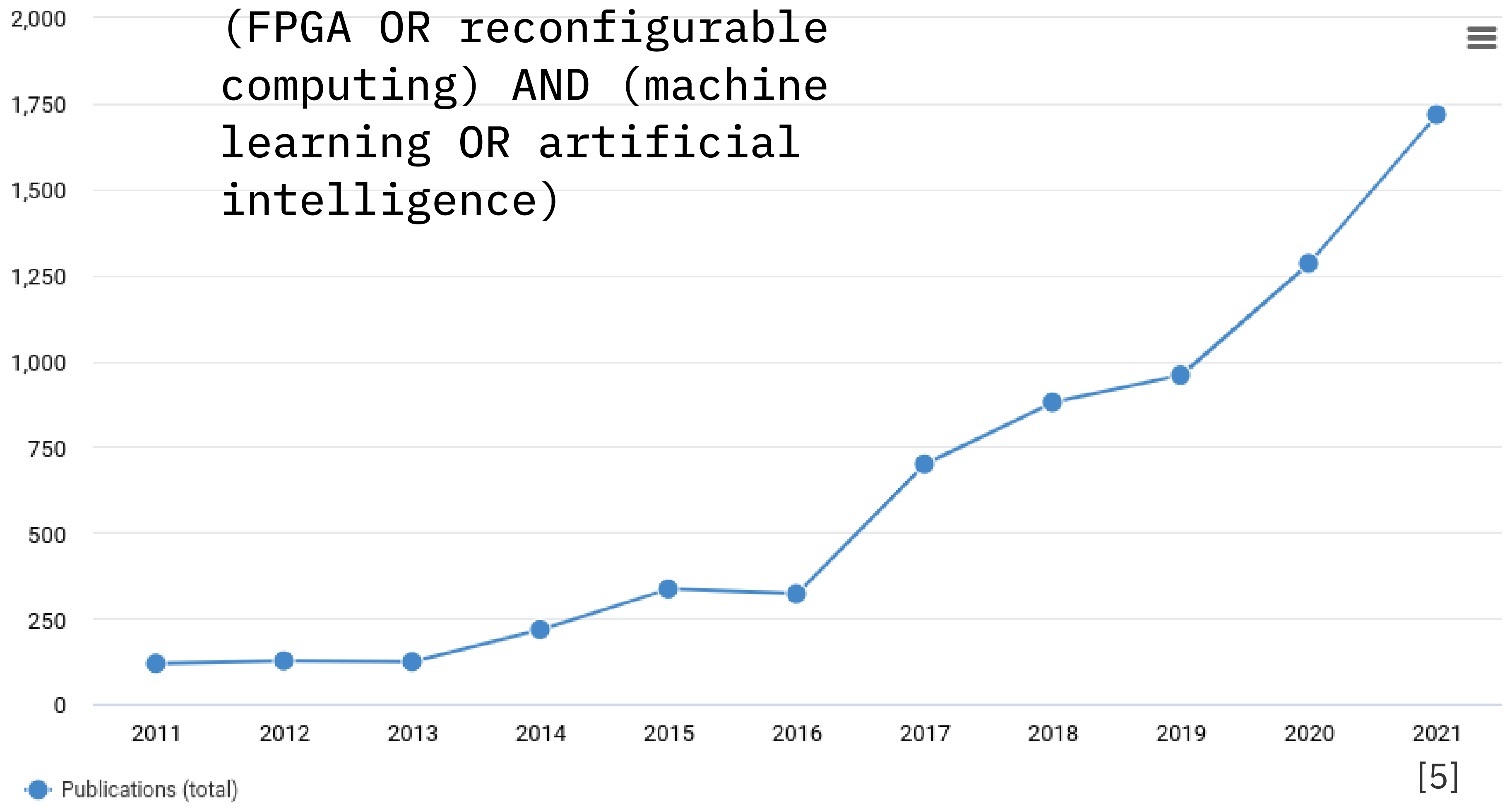
Presentation at cFDevOps22, 2022-09-01, Belfast



The mess we are in: Computing is running out of steam and energy...







[5]

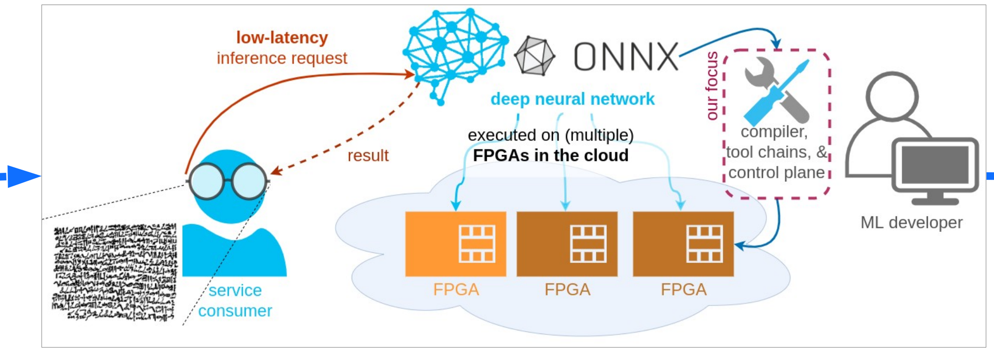
...FPGAs to the rescue?!?

- Computing is running out of steam and energy...
  - Especially for compute demanding workloads like AI/ML and HPC
- *Yeah*, there are thousands of papers about using FPGAs for AI/ML...
- **But**:
  - Most of the accelerators are GPUs (~8 – 10% of *global* compute capacity [1])
  - Largest FPGA deployments (AFAIK):
    - 48 FPGAs at PC² (“production”, Alveos)
    - 96 FPGAs at IBM Research Zurich (“experimental”, cloudFPGA platform)
    - Cloud services hard to measure, but no large growth observable...
- So, why aren’t there more FPGAs “in the real world”?
- ...maybe it has something to do with tools and Development & Operations support?

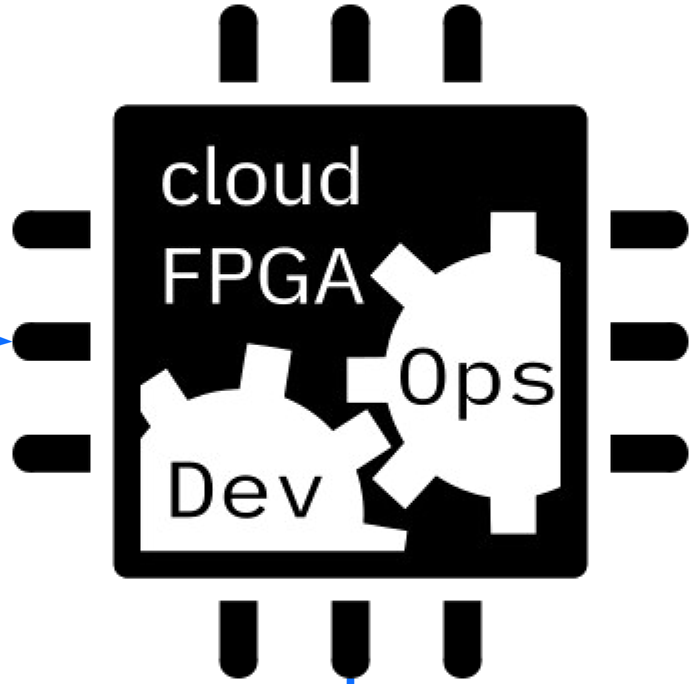
Agenda: Our Journey to DevOps for DNNs on Cloud FPGAs

→ **In this presentation**, I will analyze the challenges of deploying a distributed AI inference application on FPGAs in the Cloud and present how we worked around them.

Some background



Our use case & missing parts

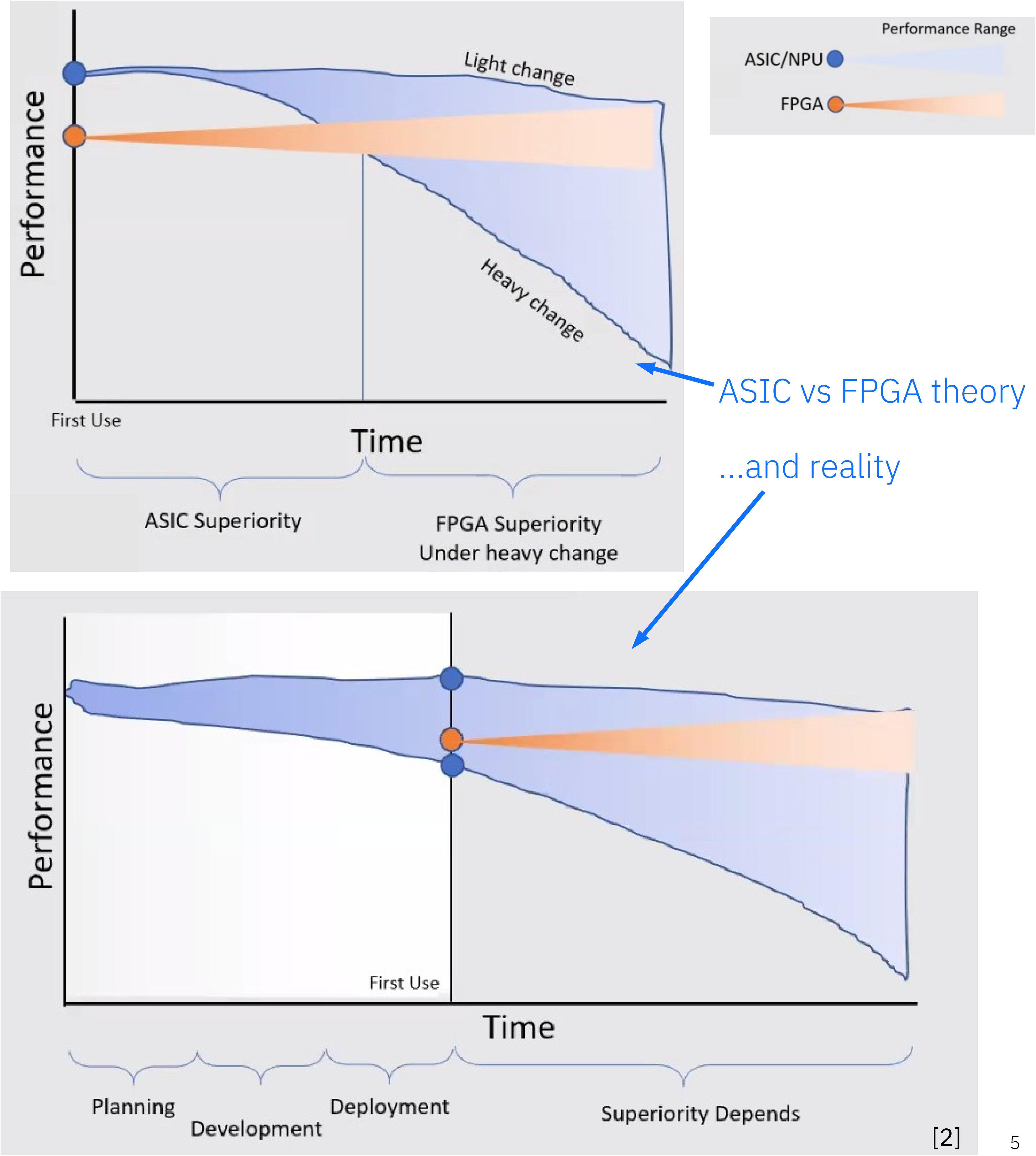


Conclusion?

→ Goal: (1) Highlight blind spots of current state of the art and (2) make *you all* eager to use our tools!

ML Acceleration: Why FPGAs are becoming popular

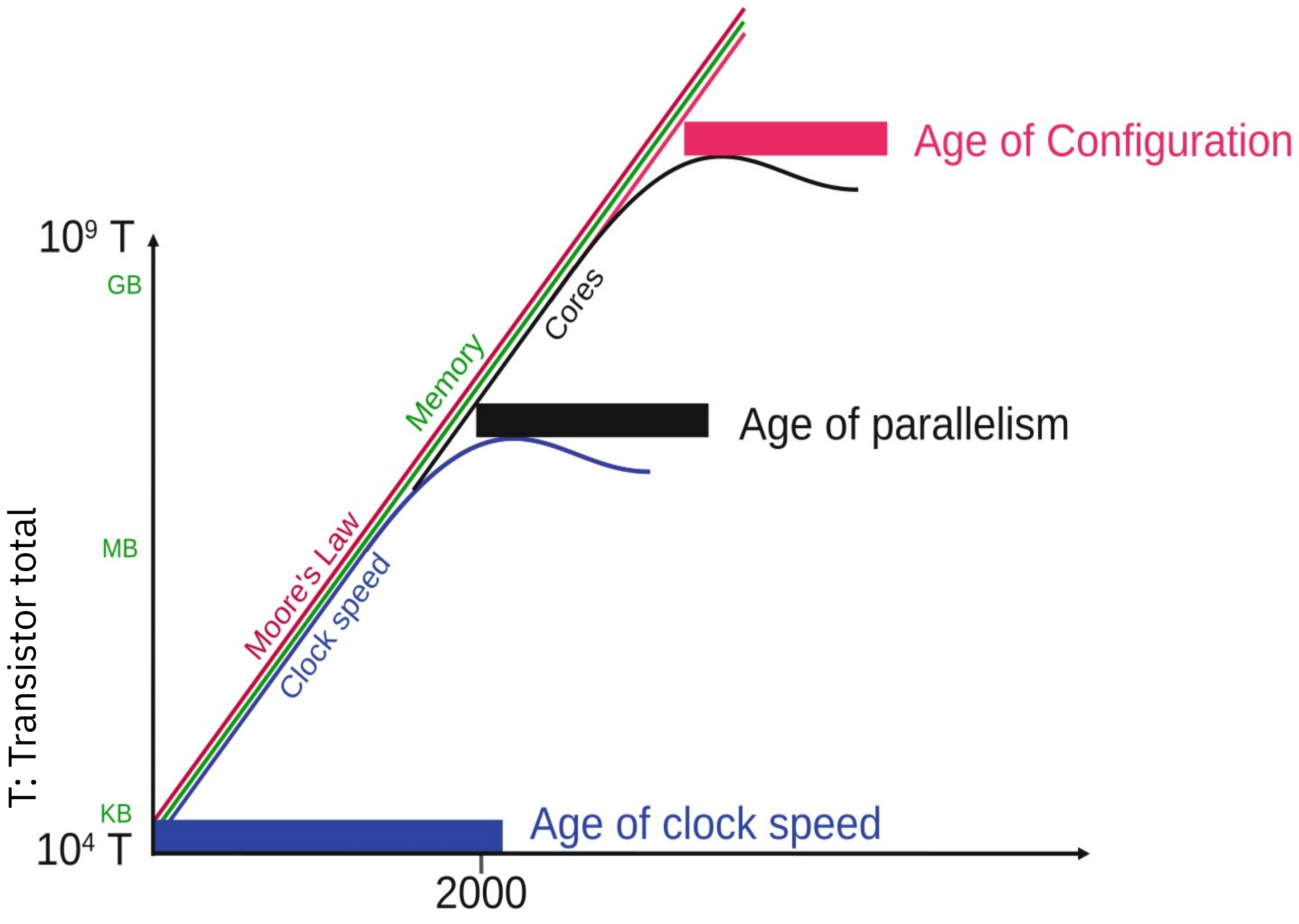
- FPGAs have a performance penalty compared to specialized chips (i.e. ASICs, e.g. “TPUs”)
  - due to the resources “overhead” necessary to be reconfigurable
- *On the contrary*: ML algorightms and models, especially Deep Neuronal Networks (DNN), change frequently
  - → FPGAs can adapt instantly, ASICs can’t adapt at all
  - → This becomes even more relevant if the development time of ASICs are taken into account
- Equally, used data types vary increasingly



ML Acceleration: Why FPGAs are necessary

- Some AI researchers point to a "Hardware Lottery" [3] :
  - Success of ML algorithms depends on their fit to current hardware, not on 'superior' concepts
  - Some novel concepts run e.g. faster on a CPU than GPU or TPU
  - But: "*Coding even simple algorithms on FPGAs remains very painful and time-consuming.*"
- We can still increase the efficiency of hardware based on domain specific tasks using reconfigurable computing

[3]



[4]

Using FPGAs for non-domain experts: Current obstacles

- Despite consolidated tool chains and high-level synthesis, using FPGAs for HPC with current industry tools…
  - Is still not straight-forward
  - Requires a high frustration tolerance
  - And requires still some architectural knowledge and re-coding of targeted kernels
- Research delivers far more narrow-scoped "proof of concepts" than end-to-end examples
  - *(luckily, this starts to change…)*
- In the end: FPGA beats CPUs regularly by two orders of magnitude and can beat GPUs [6]
  - … after investing months of optimization
  - not mentioning debugging, deployment, operation, etc.

Description	Performance GFLOPs	% CPU performance	% theoretical performance
24 cores of Xeon (Cascade Lake) CPU	65.74	-	-
Initial FPGA port	0.020	0.03%	[illegible]
Optimised for dataflow	0.28	0.43%	4.06%
Optimised memory access	0.42	0.63%	6.09%
Optimise matrix multiplications	12.72	19.35%	20.85%
Ping-pong buffering	27.78	42.26%	45.54%
Remove pipeline stalls	59.14	89.96%	96.95%
Increase clock frequency to 400 Mhz	77.73	118%	95.73%

Von-Neumann based algorithm

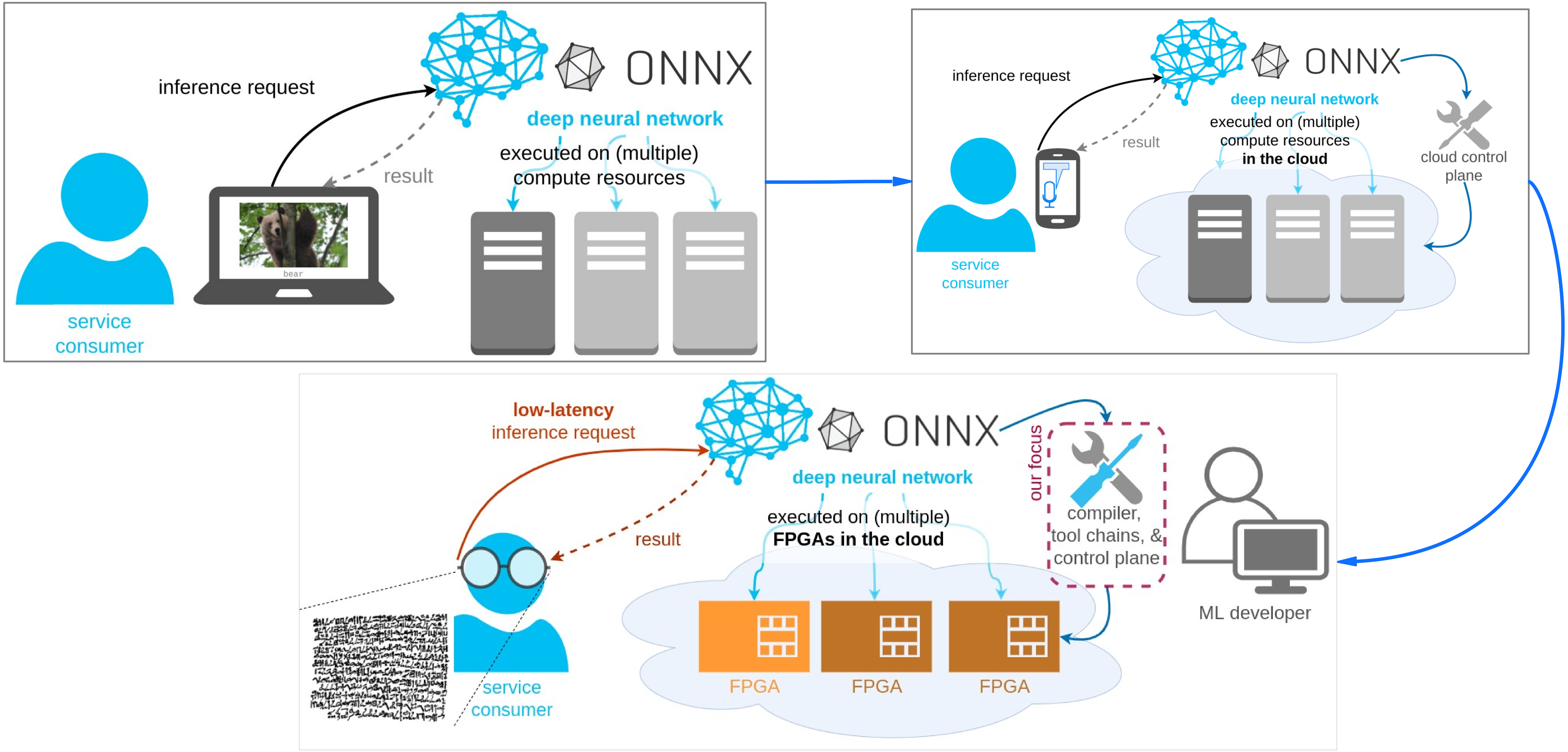
Approx. 4000 times difference in performance

Optimised dataflow based algorithm

Description	Performance (GFLOPS)	Power usage (Watts)	Power efficiency (GFLOPS/Watt)
1 CPU core	5.38	65.16	0.08
24 CPU cores	65.74	176.65	0.37
V100 GPU	407.62	173.63	2.34
1 FPGA kernel	74.29	45.61	1.63
2 FPGA kernels	146.94	52.47	2.80
4 FPGA kernels	289.02	71.98	4.02

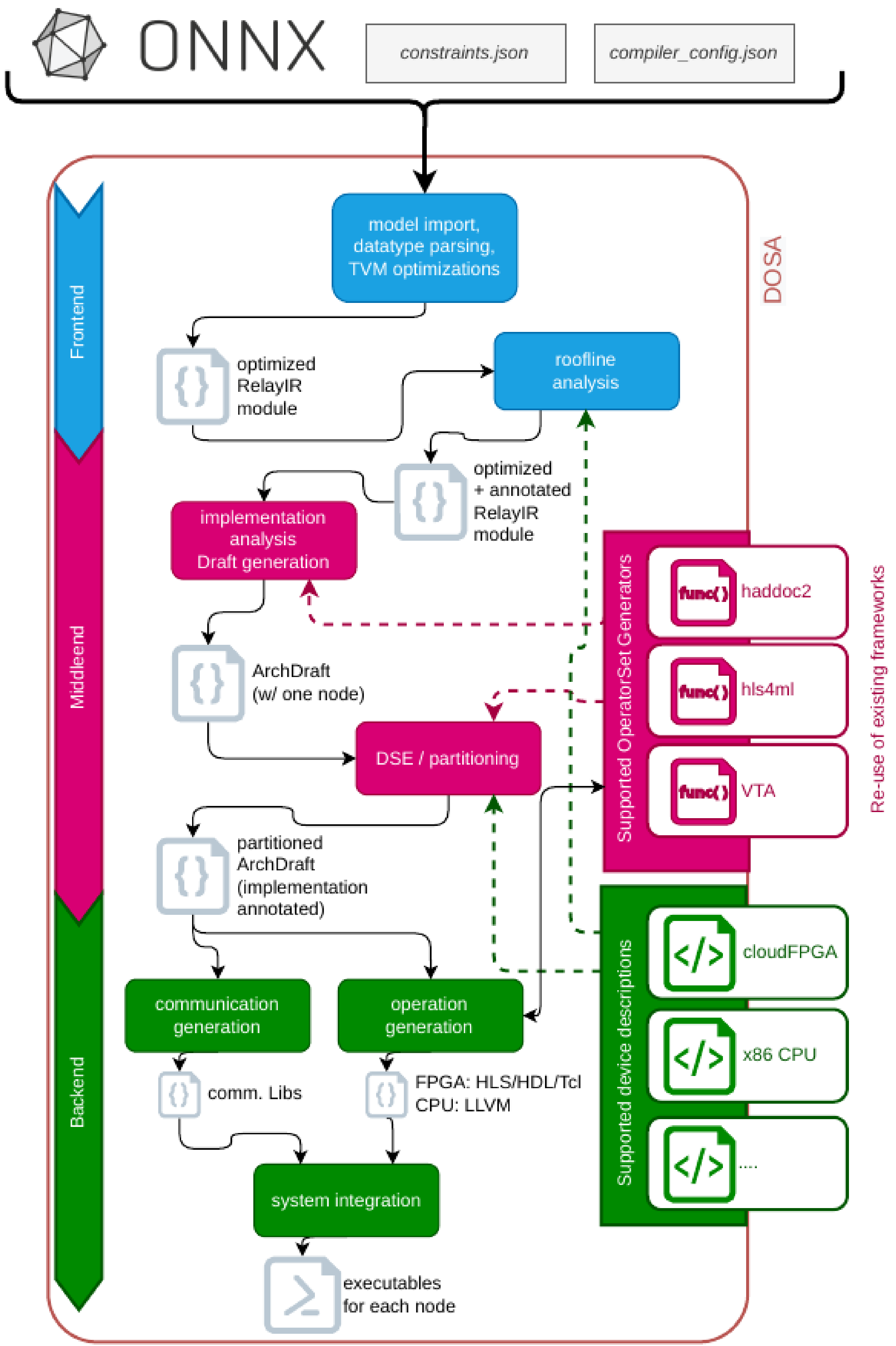
[6]

Example application: Accelerated Inference-as-a-Service

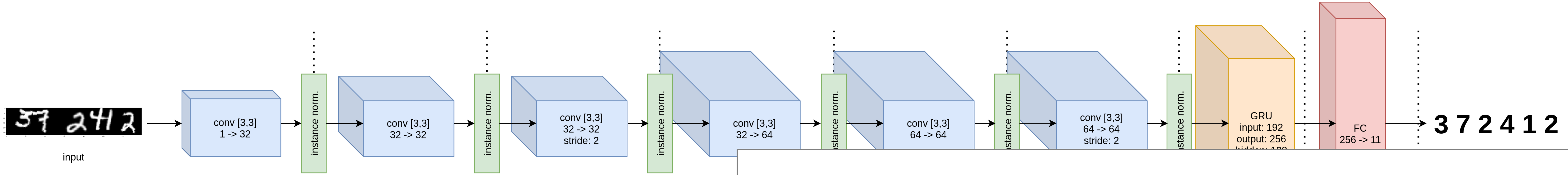


Our tool: **DOSA**, automated compilation of CNN to distributed FPGAS

- Large CNN automatically distributed & partitioned across FPGA (e.g., in cloudFPGA)
  - Target-specific transparent selection of optimal implementations across frameworks
  - Combining of different FPGA micro architectures
- Imports community standards ONNX and leverages published open source tools: TVM, hls4ml, haddoc2, VTA, ...
- Hardware agnostic, heterogeneous communication framework
- Device support:
  - Current: cloudFPGA, x86CPU
  - Upcoming: Alveo



Finding the optimal mapping: one example trade-off



Parameter (Bytes): 320 | 9.248 | 9.248

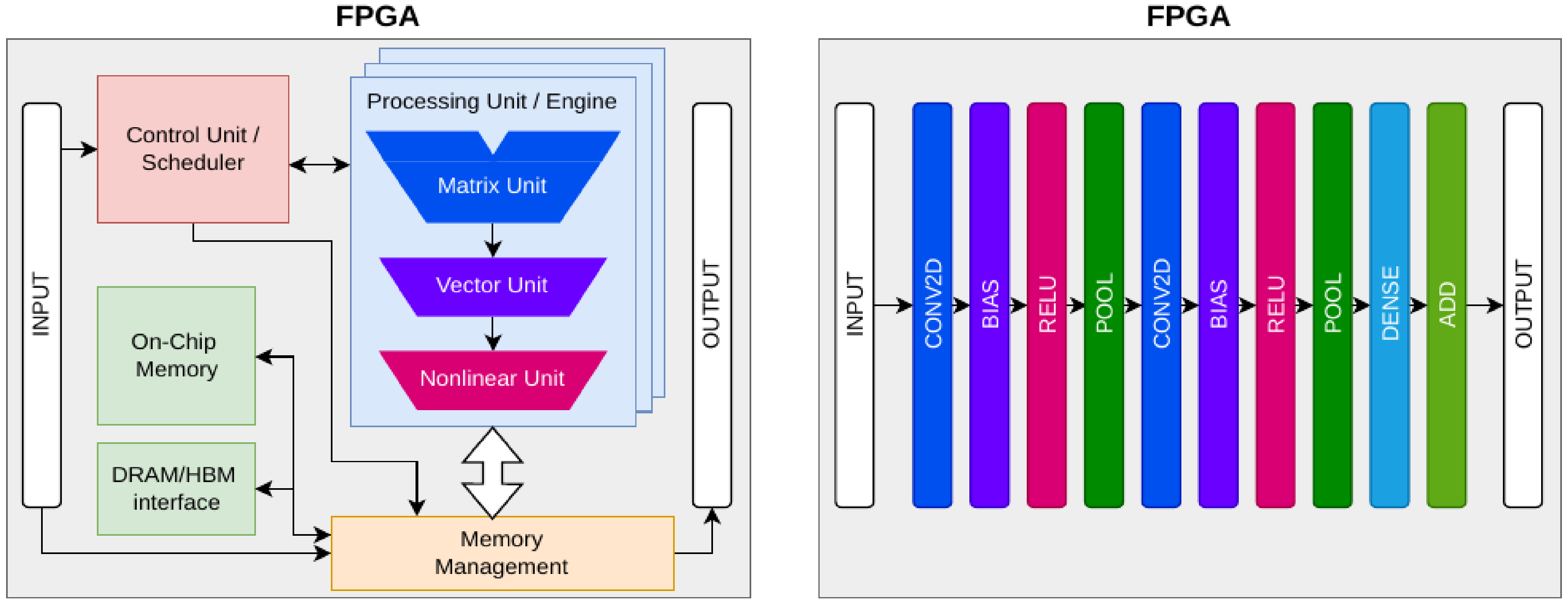
Bandwidth (B/frame) (conv & norm summarized):

125.440 | 3.674.112 | 3.342.336 | 754.688

for 1k fps: (MB/s)

125 | 3.674 | 3.342 | 755

Two types of possible FPGA internal design (i.e. micro architecture):



(a) Engine-type accelerator.

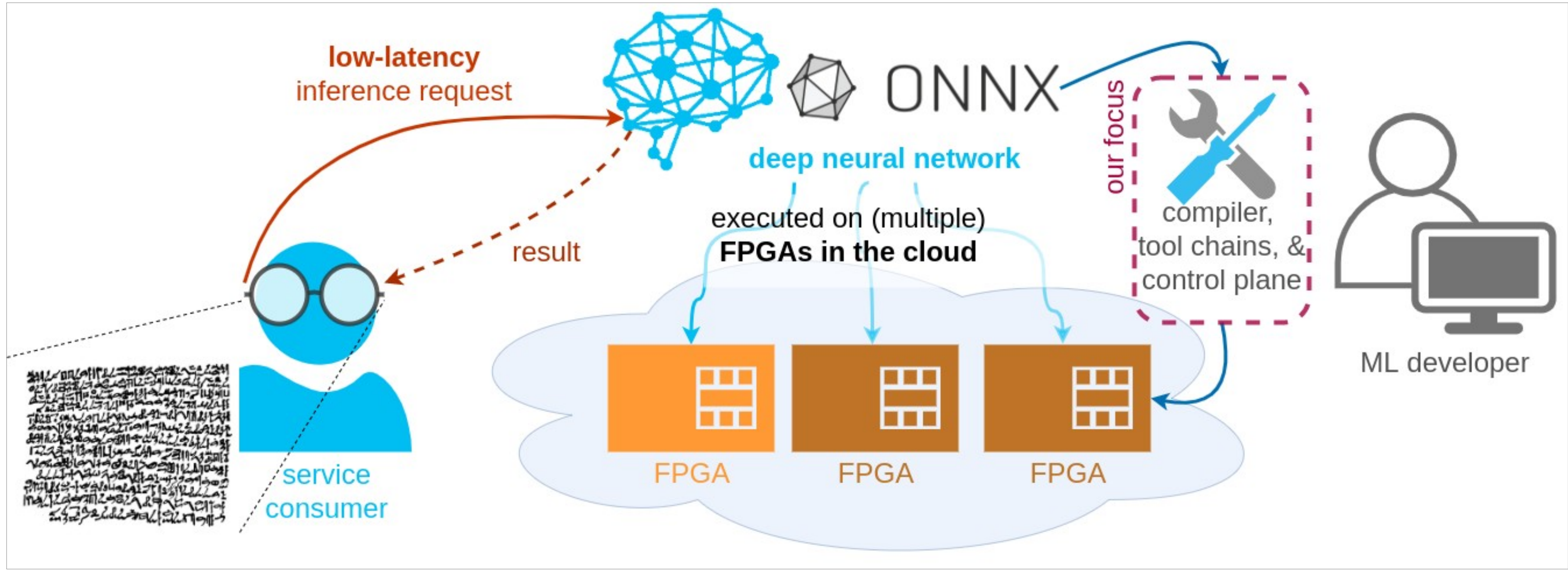
(b) Streaming-type accelerator.

or "summarized":

- Parameter:
- required Bandwidth:

→ *best trade-off?*

Example CRNN from https://codingvision.net/pytorch-crnn-seq2seq-digits-recognition-ctc



Requirements for an accelerated INFaaS

Covered "sufficiently" in literature

- FPGA Cloud:
  - Operation resource abstraction (i.e. SRA) √
  - Control plane integration of FPGAs √
  - Debugging of resource errors ✕
  - Deployment processes ~
  - Security √
- Communication:
  - To and from the consumer √
  - If distributed: communication & synchronization between FPGA nodes √
  - Debugging of communication ~/✕
- Accelerated inference application:
  - ML kernel implementation, data representation / quantization √
  - If distributed: model partitioning ~
  - Debugging of inference kernel ~/✕
  - Portability of the design ~/✕
  - System generation at compile time ~/✕

Our focus today

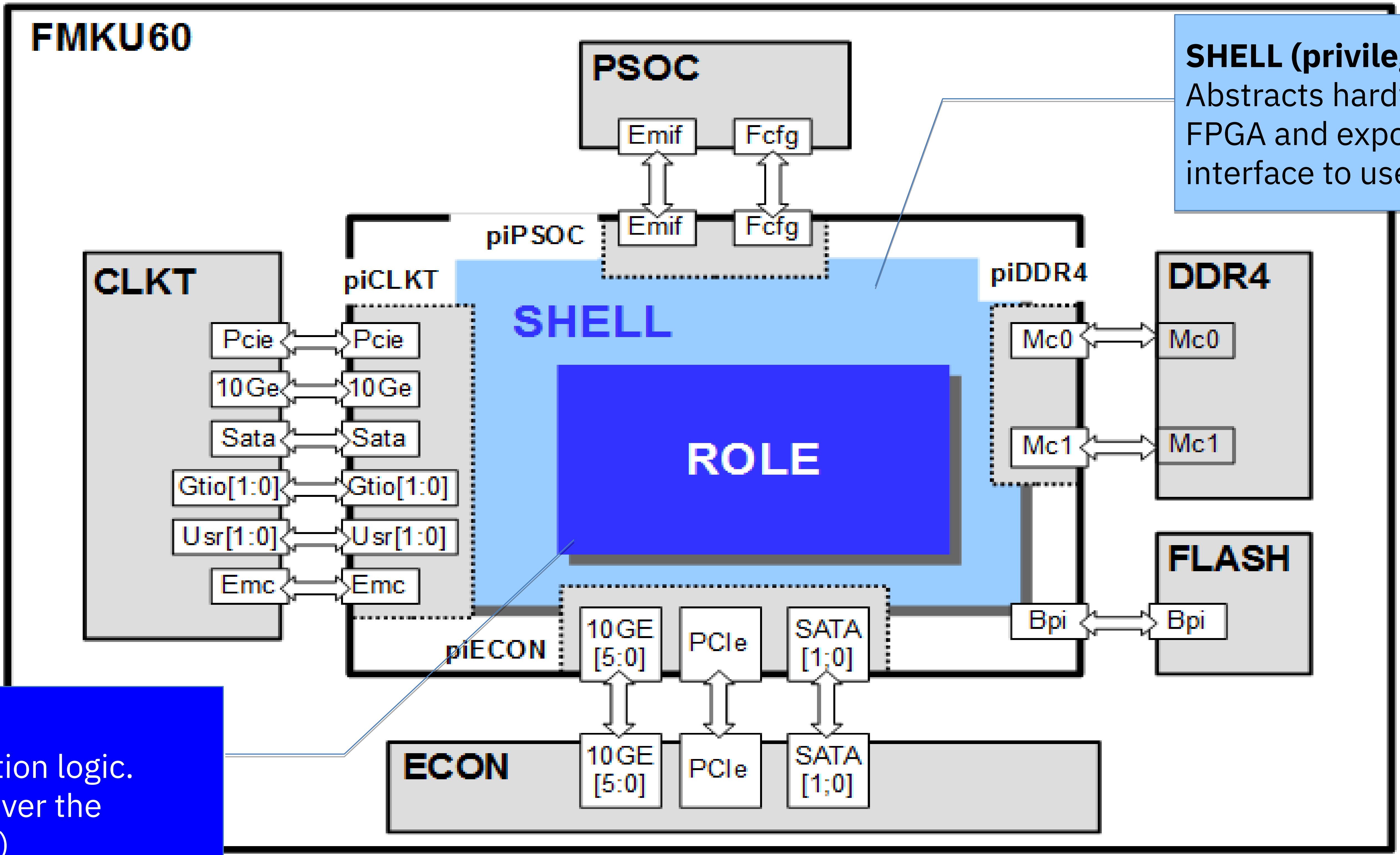
System generation

- We all know how to implement an FPGA application kernel…that's what we are here for
- But how to connect different kernels within one Role automatically?
- Common bus protocol between IP cores necessary
  - Calculate required bandwidth
  - Generate FIFOs/AXIs in VHDL and tcl
  - "Register" adapter within system design
- Automatic generation of Wrappers is also important for generating debugging (→ later)

```python
def get_vhdl_entity_declaration(self):
    if self._calc_bitw_ != self.bitwidth:
        self._calculate_bitws()
    single_decl = ('component {name} is\n' +
                   '    port (\n' +
                   '        clk     : in std_logic;\n' +
                   '        srst    : in std_logic;\n' +
                   '        din     : in std_logic_vector({width} downto 0);\n' +
                   '        full    : out std_logic;\n' +
                   '        wr_en   : in std_logic;\n' +
                   '        dout    : out std_logic_vector({width} downto 0);\n' +
                   '        empty   : out std_logic;\n' +
                   '        rd_en   : in std_logic );\n' +
                   'end component;\n')
    total_decl = single_decl.format(name=self.name + '_tdata', width=(self.tdata_bitw-1))
    total_decl += '\n'
    total_decl += single_decl.format(name=self.name + '_tkeep', width=(self.tkeep_bitw-1))
    total_decl += '\n'
    total_decl += single_decl.format(name=self.name + '_tlast', width=(self.tlast_bitw-1))
    # yes, results in (0 downto 0), but vivado_hls does this as well..
    return total_decl
```

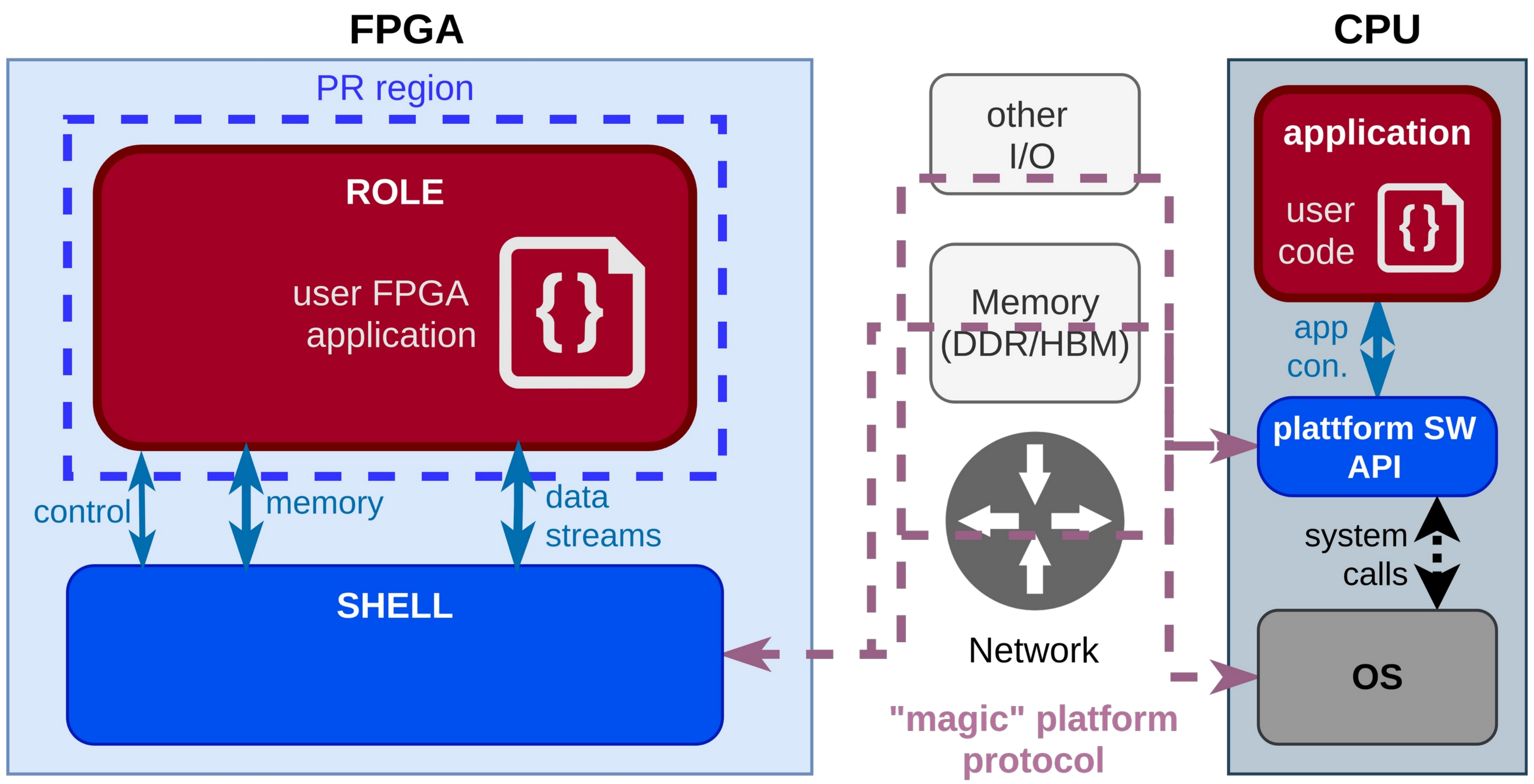
```tcl
#------------------------------------------------------------------------------
# VIVADO-IP : FIFO Generator
#------------------------------------------------------------------------------

set ipModName "Fifo_input_0_tdata"
set ipName    "fifo_generator"
set ipVendor  "xilinx.com"
set ipLibrary "ip"
set ipVersion "13.2"
set ipCfgList [ list CONFIG.Performance_Options {First_Word_Fall_Through} CONFIG.Input_Data_Width {64}
CONFIG.Output_Data_Width {64} \
                    CONFIG.Input_Depth {512} CONFIG.Output_Depth {512} \
                ]

set rc [ my_customize_ip ${ipModName} ${ipDir} ${ipVendor} ${ipLibrary} ${ipName} ${ipVersion}
${ipCfgList} ]

if { ${rc} != ${::OK} } { set nrErrors [ expr { ${nrErrors} + 1 } ] }
```

Recap: Hardware Abstraction → Shell Role Architecture



SHELL (privileged logic)
Abstracts hardware components of FPGA and exposes standard AXI(S) interface to user

ROLE (non-privileged)
Embeds user's application logic. Partially reconfigured over the network. (typically HLS)

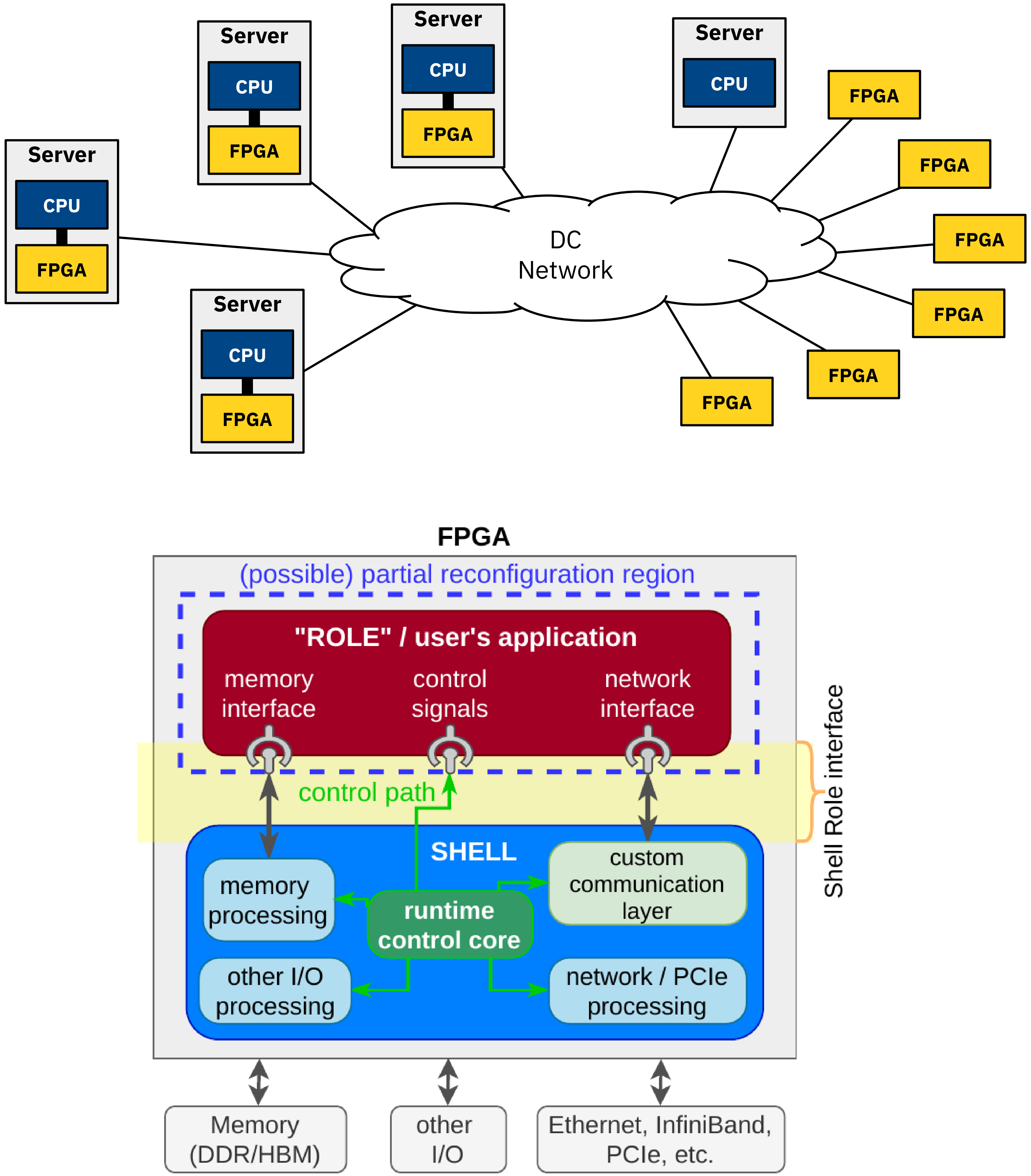


Portable system generation

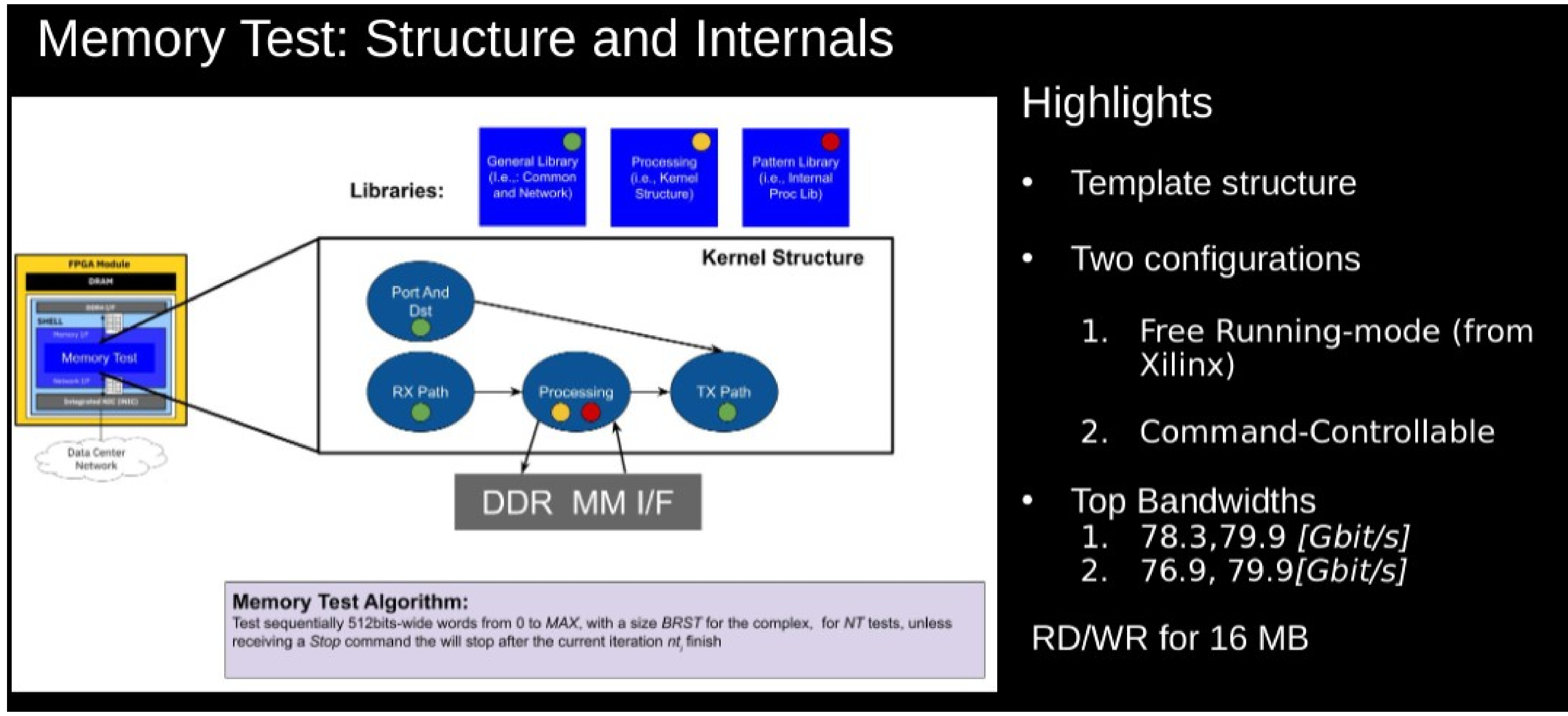
- usually FPGA applications exist not alone: See as part of a (complex) application communication schema
- In best case, we don't want to re-write a compiler for every new platform
- Besides configuration & control registers, there are usually (one of) two communication channels:
  - **address based**: PCIe, via Memory (AXI4 Full)
  - **stream based**: network or PCIe abstraction (AXI4 Stream)
- System generation should be able to adapt within this template
  - DOSA connects the IP cores based on their bus protocol specification and dependencies among each other
  - Additionally: creates adapters if necessary

Debugging of “operation resources”

- “Operation resources”: resources offered to the application during operation: e.g. memory space, network access, configuration registers
- If…data doesn’t arrive at the right time at the right place…*who to blame?*
  - Bugs in the application?
  - Lost in the data center / communication fabric?
  - Lost in the Shell?
  - Memory failures?
- → Best: Provide control counters at the Shell Role interface & verifying the memory prior to deployment



cFDK: “flight recorder data” and memory test



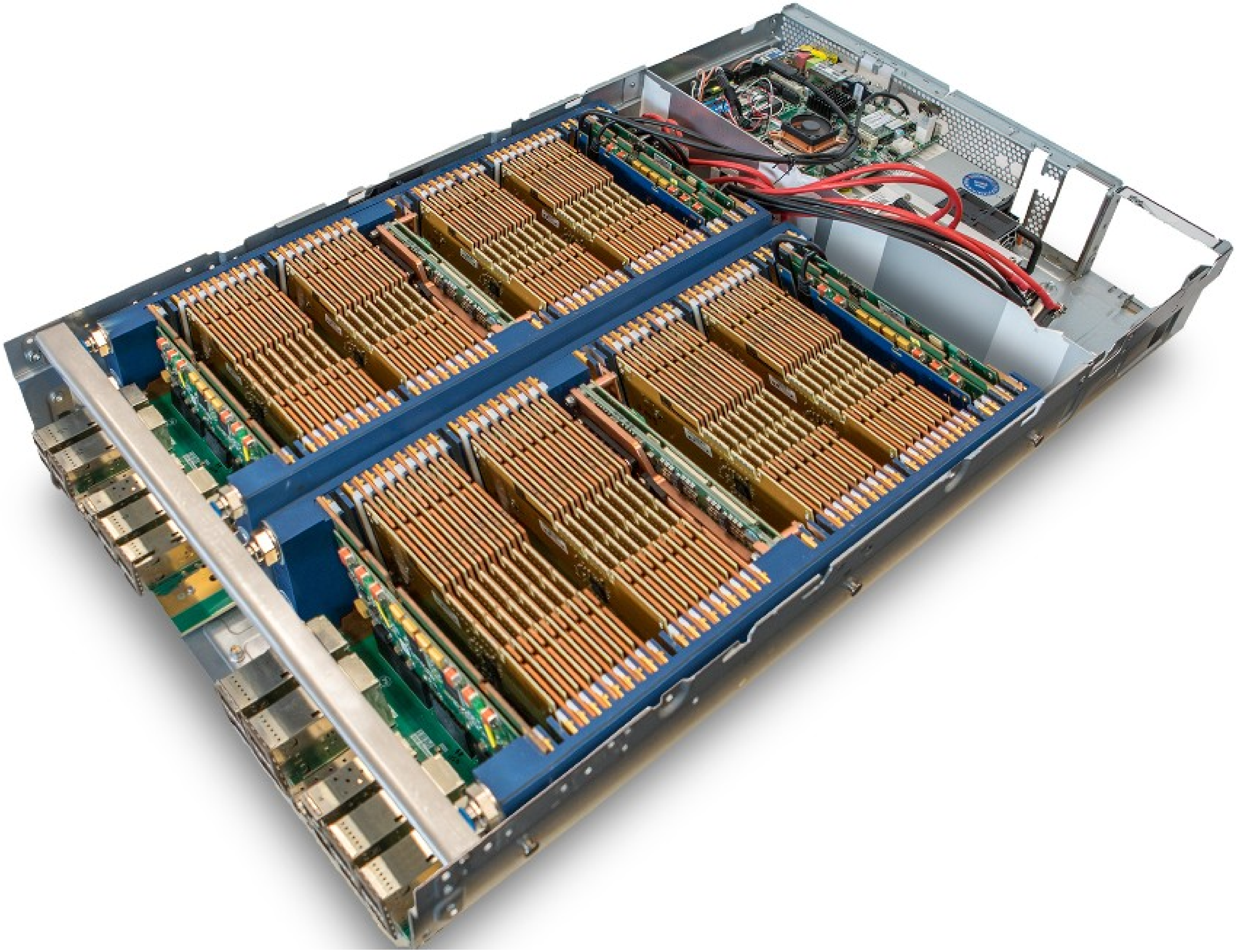
- At boot-time: Check physical health
  - DDR4 synchronization
  - ETH clock & synchronization
- Occasional complete memory tests
- Monitor live data in the FPGA
  - Compromise between amount (i.e. 5 last packages) and overhead → focus on most expressive data
  - Request via REST API

GET `/clusters/{cluster_id}/flight_recorder_data` Requests network runtime information of all instances

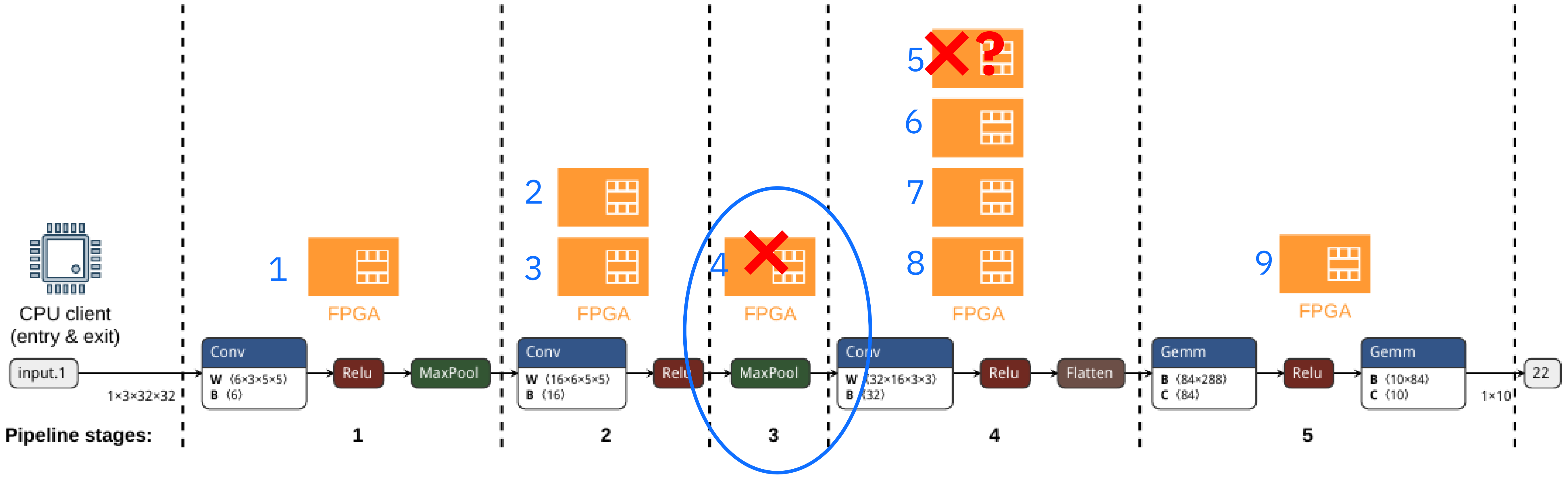

```
"72": [
  "Rank: 12",
  "Size: 23",
  "Last RX port: 2718",
  "Last RX id: 22",
  "Last TX port: 2718",
  "Last TX id: 22",
  "RX packet count: 6026",
  "TX packet count: 4017",
  "cFDK/FMC version: 1.0",
  "FPGA uptime: 17:11:06",
  "current ROLE version: 318",
  "Layer 4 (TCP/UDP) is ENABLED.",
  "Layer 6 (Network Routing) is ENABLED.",
  "Layer 7 (ROLE) is ENABLED.",
  "UDP RX drop count: 0",
  "Invalid node-id/ip-address RX count: 0",
  "Invalid port TX count: 0",
  "Invalid node-id/ip-address TX count: 0",
  "Failed creation of TCP connections (TX) count : 0",
  "TCP RX notif drop count: 0",
  "TCP RX meta drop count: 0",
  "TCP RX data drop count: 0",
  "TCP RX CRC drop count: 0",
  "TCP RX Session drop count: 0",
  "TCP RX Out-of-Order drop count: 0"
],
```

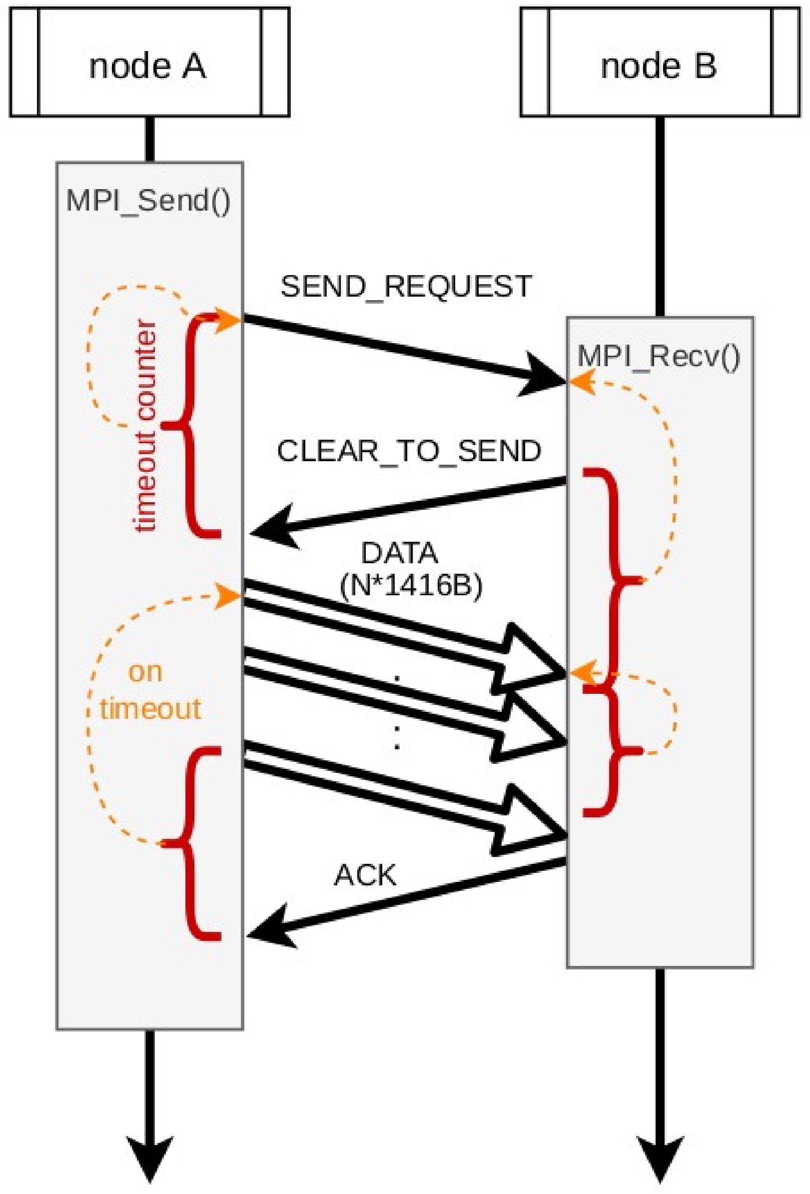
As context, our platform: The IBM cloudFPGA Platform (19”x2U w/64 FPGAs)

(more information at github.com/cloudfpga)



Debugging of distributed applications





- "Flight recorder data" in combination with
  - communication plan
  - and synchronous protocol (MPI)
- Hence, if combined →we can say where packages are missing
  - As part of the "flight recorder data" we know in which state the protocol engine is
  - (To much packets can't be discovered easily, since re-transmissions could occur)

Communication plan of node 4:

step	to	from	no. packets
2		3	20
3	5		~~5~~ 3
4	6		5

Debugging generated by compiler

- Once we know which FPGA node “misbehave”, we still have to look into it
- Hence, DOSA automatically generates debug probes between IP cores
  - Because we use standardized interfaces between IP cores → easily generate able by compiler
  - In VHDL and tcl
- We deploy bitstreams using partial reconfiguration → debug bridge support
- …then we still have to look at waveforms…

```vhdl
--###########################################################
--  Debug Core instantiation
--###########################################################

DBG: ila_dosa_role_0
  port map (
    clk => piSHL_156_25Clk
    , probe0      =>   siNRC_Udp_Data_tdata
    , probe1      =>   siNRC_Udp_Data_tkeep
    , probe2(0)   =>   siNRC_Udp_Data_tvalid
    , probe3(0)   =>   siNRC_Udp_Data_tlast
    , probe4(0)   =>   siNRC_Udp_Data_tready
    -- ...
    , probe52     =>   sMPE_Debug
    , probe53     =>   sZRLMPI_Wrapper_Debug
    , probe54(0)  =>   sResetApps_n
    , probe55     =>   sToFifo_input_0_tdata_din
    , probe56(0)  =>   sToFifo_input_0_tdata_full_n
    , probe57(0)  =>   sToFifo_input_0_tdata_full
    , probe58(0)  =>   sToFifo_input_0_tdata_write
    , probe59     =>   sToFifo_input_0_tkeep_din
    , probe60(0)  =>   sToFifo_input_0_tkeep_full_n
    , probe61(0)  =>   sToFifo_input_0_tkeep_full
    , probe62(0)  =>   sToFifo_input_0_tkeep_write
    , probe63     =>   sToFifo_input_0_tlast_din
    , probe64(0)  =>   sToFifo_input_0_tlast_full_n
```

```tcl
#------------------------------------------------------------------------------
# VIVADO-IP : ILA Core
#------------------------------------------------------------------------------
set ipModName "ila_dosa_role_0"
set ipName    "ila"
set ipVendor  "xilinx.com"
set ipLibrary "ip"
set ipVersion "6.2"
set ipCfgList [list  CONFIG.C_NUM_OF_PROBES 112 \
                     CONFIG.C_DATA_DEPTH 2048 \
                     CONFIG.C_PROBE0_WIDTH {64}\
                     CONFIG.C_PROBE1_WIDTH {8}\
                     CONFIG.C_PROBE2_WIDTH {1}\
                     CONFIG.C_PROBE3_WIDTH {1}\
                     CONFIG.C_PROBE4_WIDTH {1}\
                     CONFIG.C_PROBE5_WIDTH {64}\
                     CONFIG.C_PROBE6_WIDTH {1}\
                     CONFIG.C_PROBE7_WIDTH {1}\
                     CONFIG.C_PROBE8_WIDTH {8}\
```

Deployment with “one-click”

- Configuring 10+ FPGAs manually could be time consuming…
- We developed a resource manager that deploys clusters of FPGAs based on JSON description
  - Combination of FPGA and CPU nodes possible
  - Automatic configuration of “firewall”, routing tables etc.
- “cloudFPGA support package” as a command line tool
- Additionally, we use partial reconfiguration via network (TCP) to parallelize deployments


```
{¶
    name: mnist_v2_int8 (Net3),¶
    total_nodes: 23,¶
    nodes: [¶
        {¶
            folder: node_0,¶
            ranks: [¶
                0¶
            ],¶
            type: CPU_dummy_x86-1¶
        },¶
        {¶
            folder: node_1,¶
            ranks: [¶
                1¶
            ],¶
            type: cF_FMKU60_Themisto-Role_1¶
        },¶
        {¶
            folder: node_2,¶
            ranks: [¶
                2¶
            ],¶
            type: cF_FMKU60_Themisto-Role_1¶
        },¶
```


```
$ cfsp cluster post -description=file.json
```

operation	**file size** in MiB	**total time** in $seconds$	**effective speed** in $\frac{kiB}{s}$
JTAG config. of the compl. design	24.5	55.09	455.39
JTAG partial reconfig. of Mantle	1.8	11.07	166.43
JTAG partial reconfig. of app logic	12.8	30.85	424.82
`POST /configure` of partial Mantle logic via TCP	1.8	0.17	10,788.41
`POST /configure` of partial app logic via TCP	12.8	1.07	12,215.09

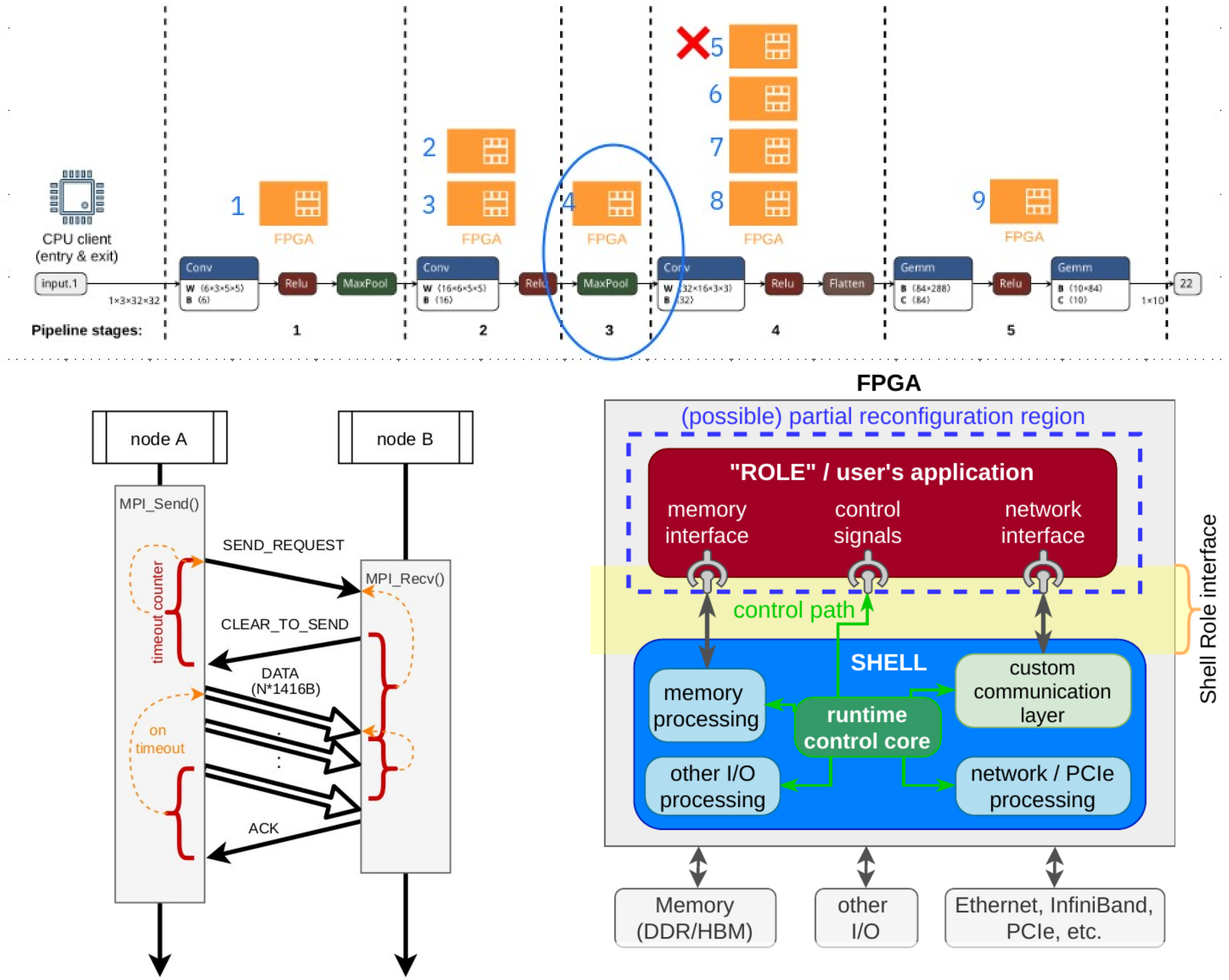
Conclusion?

- After "End of line": To increase performance, systems must become more efficient
  → more specialization
  → more reconfigurable computing
- FPGAs are a valuable option, because:
  - great flexibility, low costs
  - high performance, growing ecosystem
- But: not yet used at scale because
  - "still hard to use"
  - A lot of progress around "proof of concept" but not real end2end use cases

➔ better tools, frameworks, compilers necessary

➔ better re-usability and cooperation in the community

➔ more open source and research around DevOps!



...looking forward to all your questions!

Burkhard Ringlein

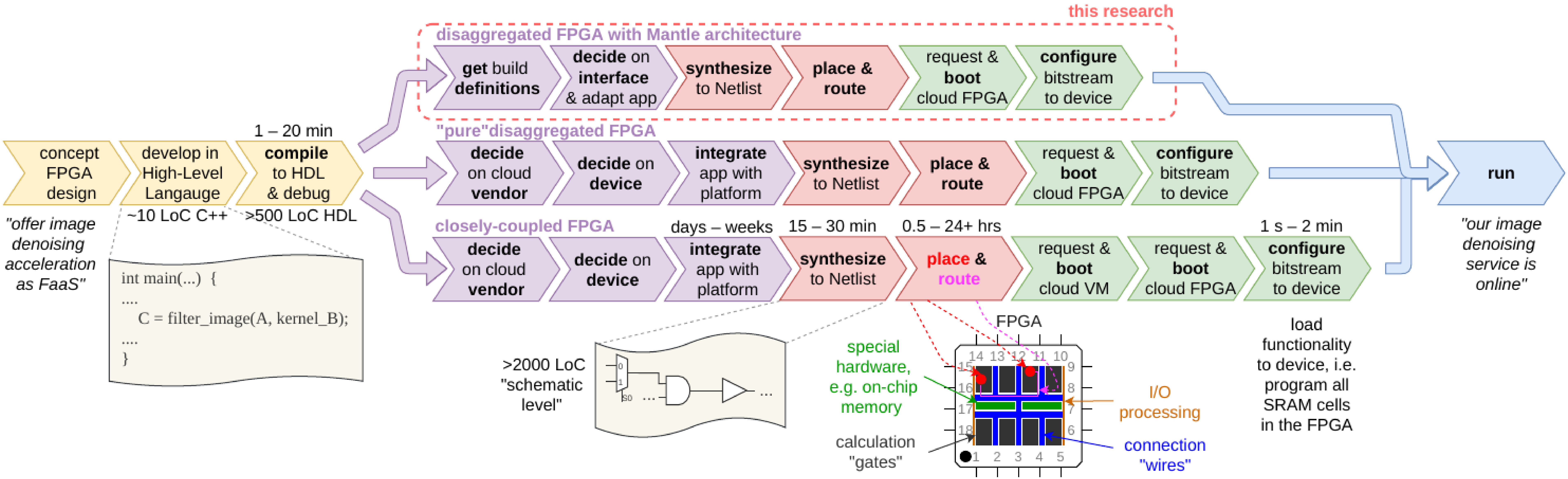
ngl@zurich.ibm.com

zurich.ibm.com/cci/cloudFPGA/

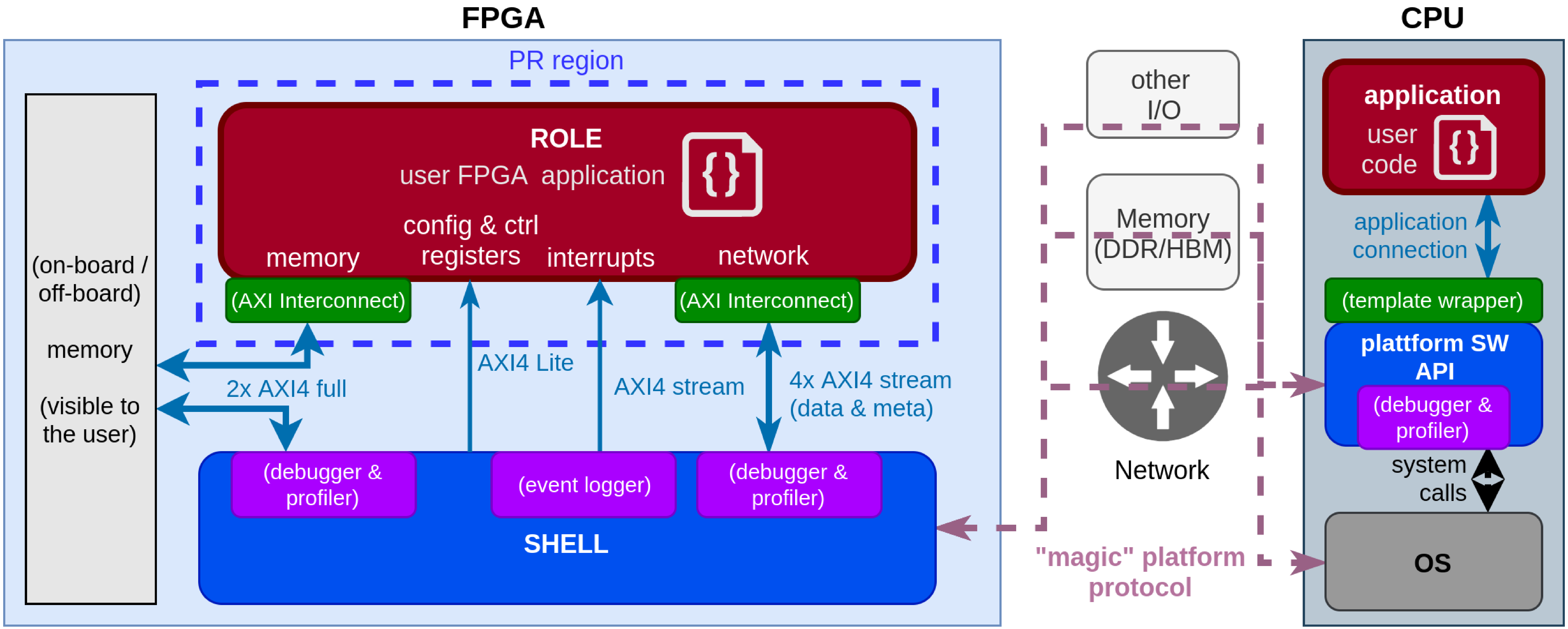
@0xcaffee

Appendix

A Brief Overview of Designing with FPGAs in the Cloud



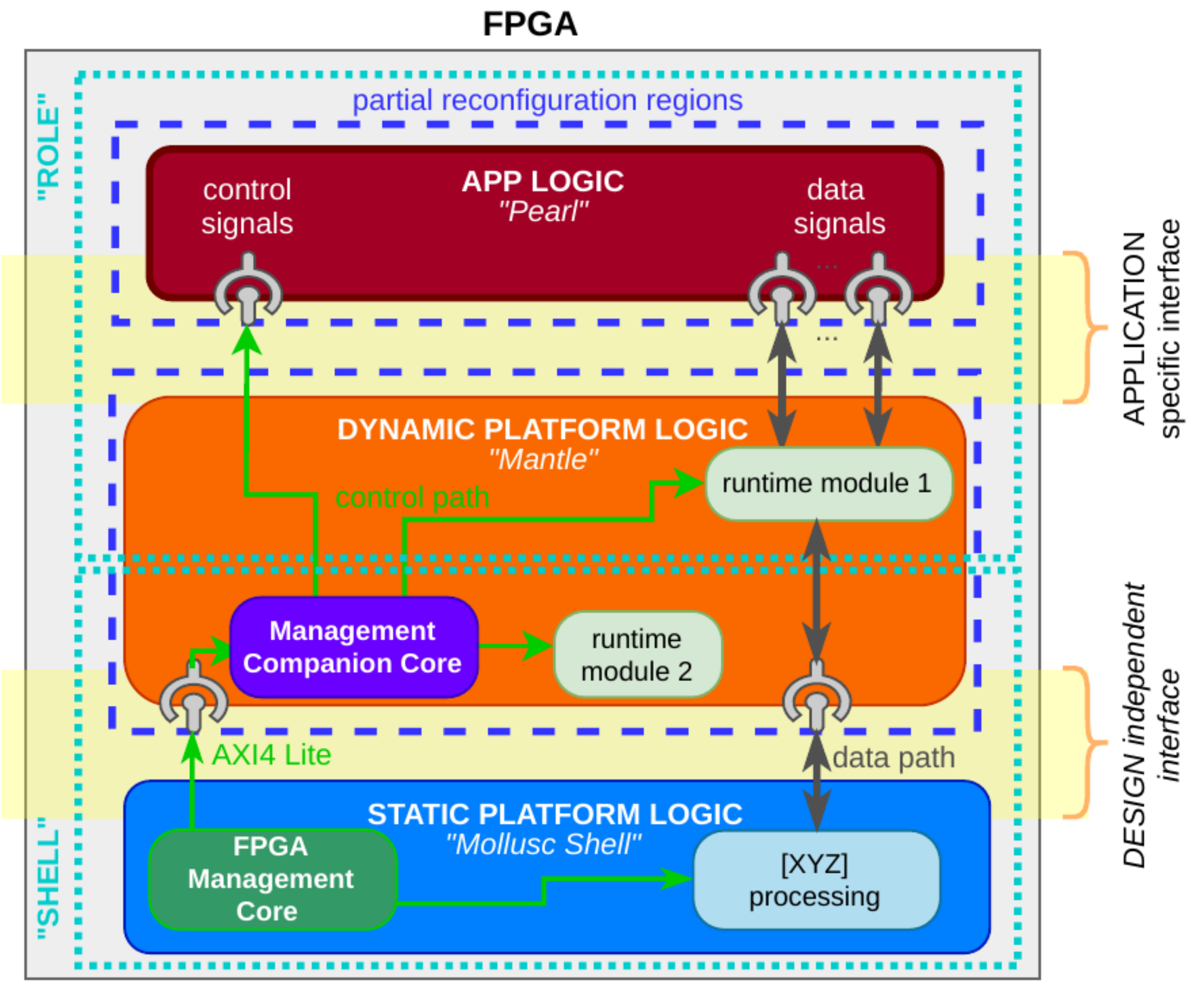
Portability: A system view



Evaluation: Configuration

Configuration times on cF:

operation	file size in MiB	total time in $seconds$	effective speed in $\frac{kiB}{s}$
JTAG config. of the compl. design	24.5	55.09	455.39
JTAG partial reconfig. of Mantle	1.8	11.07	166.43
JTAG partial reconfig. of app logic	12.8	30.85	424.82
`POST /configure` of partial Mantle logic via TCP	1.8	0.17	10,788.41
`POST /configure` of partial app logic via TCP	12.8	1.07	12,215.09



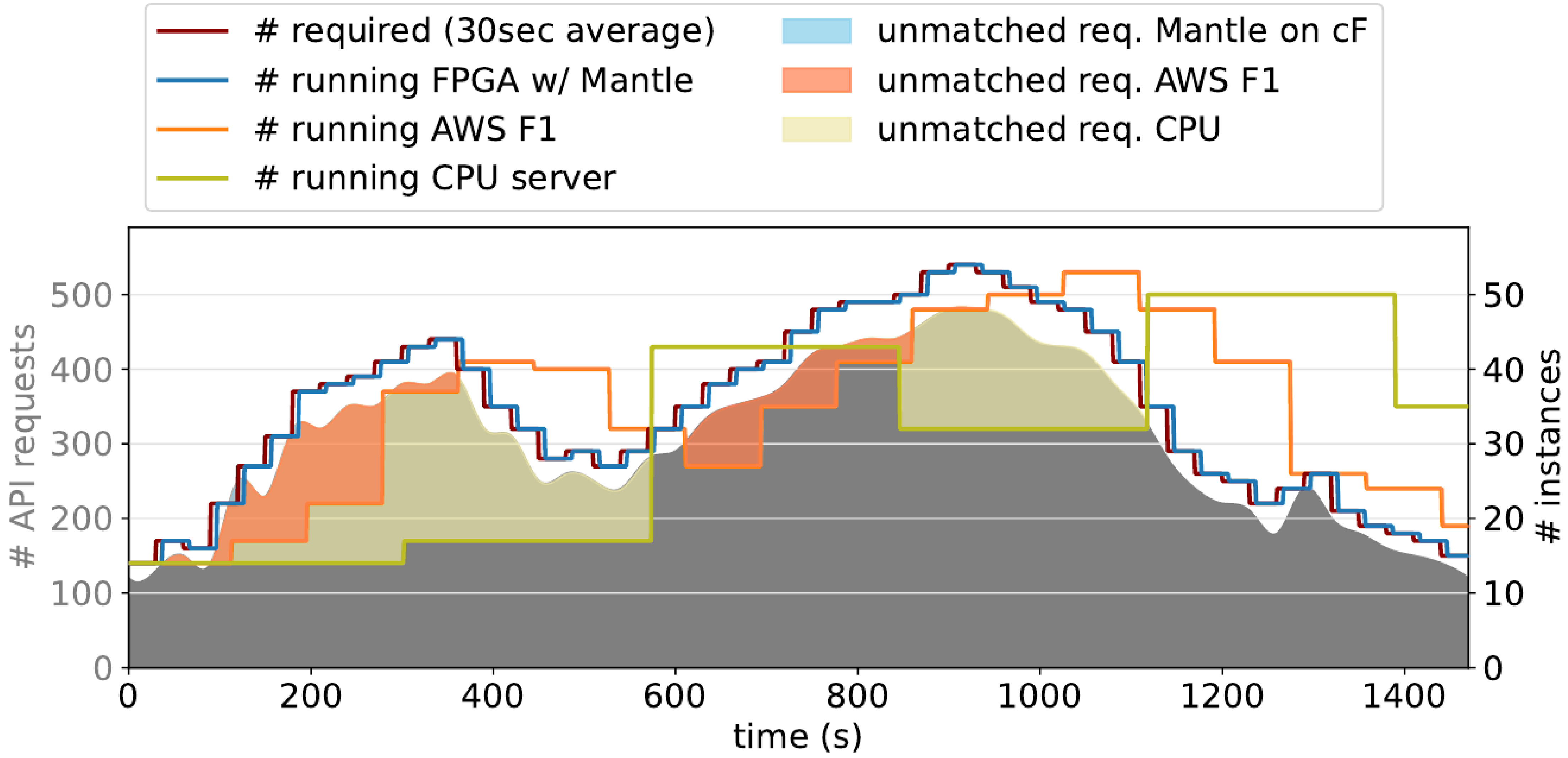
- Joint Test Action Group (JTAG) bus at 5 Mbit/s
- TCP based on 10GbE
- Result:
  - partial reconfiguration via network outperforms classical JTAG approach by a factors of 28 – 65
  - → reduced switching-costs / provisioning-time of a service

Evaluation: Provisioning

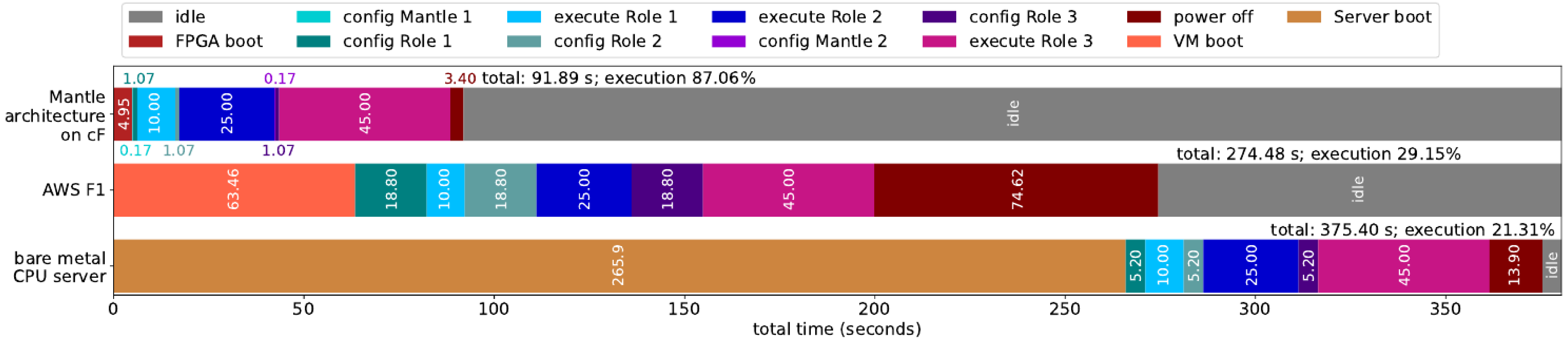
- measured time of cold-boot until application execution by FPGA/CPU, based on requirements of FaaS framework
  - cF outperforms CPU 40 times, and AWS F1 10 times
    (for AWS time of VM provisioning instead of hardware boot)
- modeled behavior for dynamic request scenarios, the systems can meet the requirements:
  - cF : 97.7%
  - AWS F1: 61.3%
  - CPU: 42.1%

PROVISIONING TIMES

cold boot of	**time** in *seconds*
CPU	271.10
AWS EC2 F1	82.26
cloudFPGA (cF)	6.20



Evaluation: Cold-boot and Switching Costs



- efficiency of FaaS depends (also) on boot and switching times → our main goal
- measured total time to execute three different functions on one device from boot to power off
  - but application agnostic: replacing application execution time with placeholders (10, 25, and 45 sec)
- Result: Mantle architecture finishes before CPU is booted and while AWS F1 is executing the first app
  - cF with Mantle architecture spends close to 90% of the total time on execution

References and Notes

[1] Semiconductor Research Corporation, ‘Decadal plan for semiconductors — full report,’ Semiconductor Research Corporation, Tech. Rep., Feb. 2021.
[2] Pictures from: Doug Burger, “Will Programmable Hardware Reach Scale”, Keynote FPL 2020, September 2020.
[3] S. Hooker, ‘The hardware lottery,’ Commun. ACM, vol. 64, no. 12, pp. 58–65, Nov. 2021. DOI: 10.1145/3467017.
[4] Bernd Klauer, The convey hybrid-core architecture. (High-Performance Computing Using FPGAs, Springer, New York, 2013)
[5] Screenshot from app.dimensions.ai/, August 2022.
[6] Both from: Nick Brown (EPCC at the University of Edinburgh), "Exploring the acceleration of Nekbone on reconfigurable architectures", Sixth International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC'20), 2020. Presentation slides: https://h2rc.cse.sc.edu/2020/slides/03_Brown.pdf

Partly, references and sources are given in the slides directly.
All remaining images are from IBM DAM or IBM Websites or created by the author.

cloudFPGA: Further Reading

- B. Ringlein, F. Abel, D. Diamantopoulos, B. Weiss, C. Hagleitner, M. Reichenbach and D. Fey, “A Case for Function-as-a-Service with Disaggregated FPGAs” in Proceedings of the 2021 IEEE 14th International Conference on Cloud Computing (CLOUD 2021), 2021.
- B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner and D. Fey, “Programming Reconfigurable Heterogeneous Computing Clusters Using MPI With Transpilation” in Proceedings of the IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC), 2020.
- B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner and D. Fey, “ZRLMPI: A Unified Programming Model for Reconfigurable Heterogeneous Computing Clusters” in 28th IEEE International Symposium On Field-Programmable Custom Computing Machines (FCCM), 2020.
- B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner and D. Fey, “ System architecture for network-attached FPGAs in the cloud using partial reconfiguration,” in 29th International Conference on Field Programmable Logic and Applications (FPL), 2019.
- F. Abel, J. Weerasinghe, C. Hagleitner, B. Weiss, S. Paredes, “An FPGA Platform for Hyperscalers,” in IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI), Santa Clara, CA, pp. 29–32, 2017.
- F. Abel, “How do you squeeze 1000 FPGAs into a DC rack?” online at LinkedIn: https://www.linkedin.com/pulse/how-do-you-squeeze-1000-fpgas-dc-rack-francois-abel/

- The **cloudFPGA project page at ZRL:** https://www.zurich.ibm.com/cci/cloudFPGA/

IBM