2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

# Model-Based Approach for Developing Optimal HW/SW Architectures for AI systems

Petri Solanti, Siemens EDA

Russell Klein, Siemens EDA

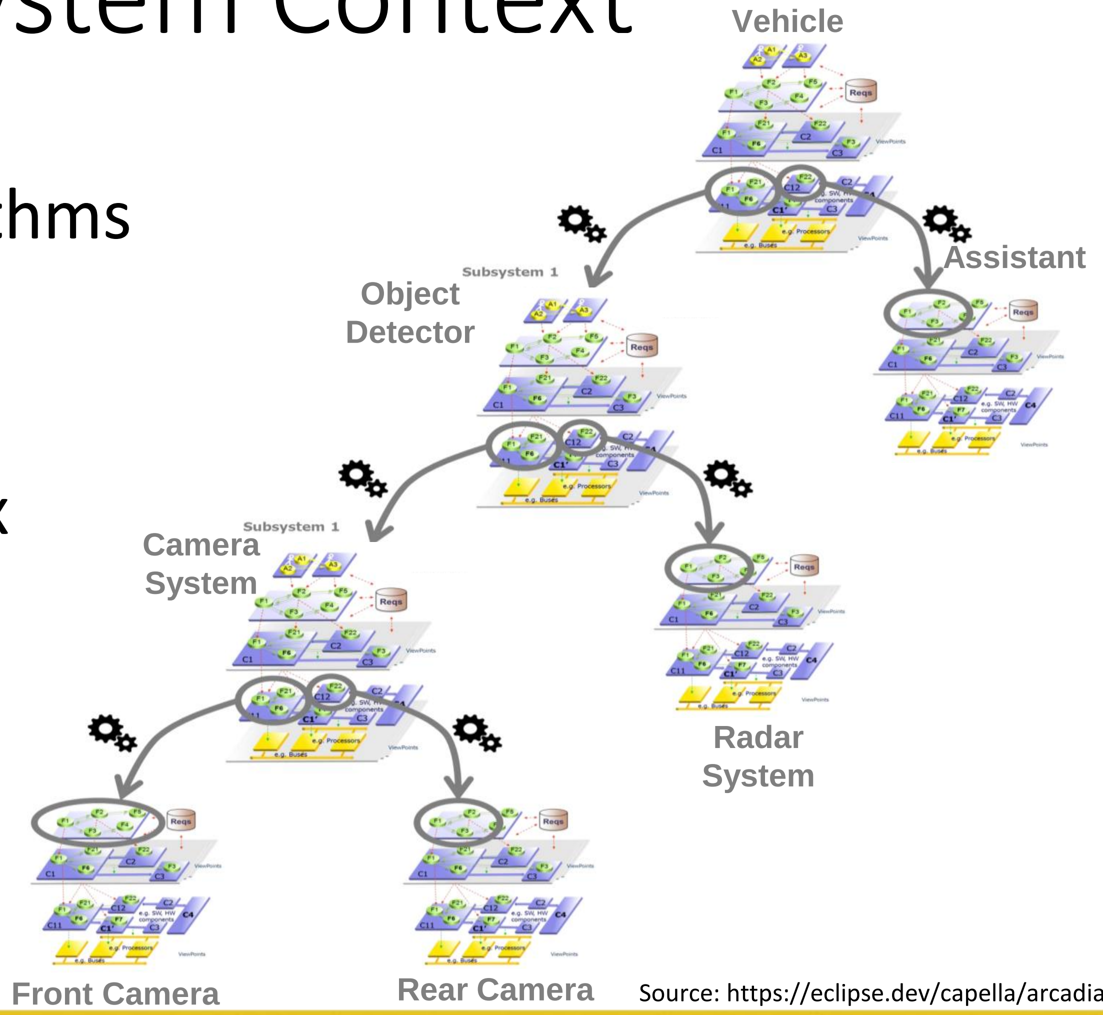
SIEMENS

accellera SYSTEMS INITIATIVE

# Artificial Intelligence in System Context

One system can have multiple AI algorithms

- Dedicated algorithms for different purposes
- Nested algorithms to provide complex functions, e.g.,
    - Filtering and FFT
    - Object recognition
    - Sensor fusion



Source: https://eclipse.dev/capella/arcadia.html

# Deploying Inferencing Systems, where and how

Where

The Cloud[1]

A Gateway[2]

The Edge[3]

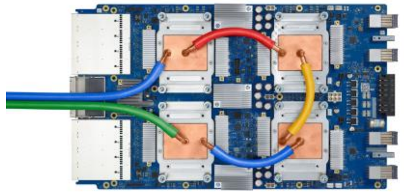
Higher Latency/Lower development costs

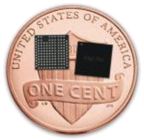
Highest performance and efficiency are achieved with specialized ASIC implementation running on the edge

How

CPU[4]

GPU[5]


TPU/NPU[6]


Edge TPU[6]

FPGA or ASIC[7]

Higher specialization/Lower energy

Sources: [1]Getty Images, [2]Stereolabs, [3]ems1.com, [4]intel.com, [5]wolfadvancedtechnology.com, [6]google.com, [7]Adobe Stock

## System Architecture Considerations

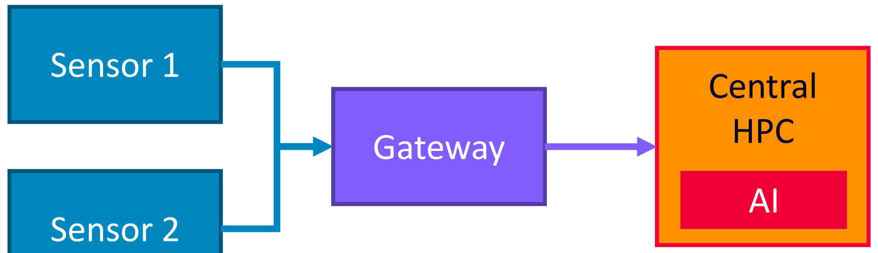
**AI algorithms can be implemented in many different ways:**

- Pure software implementation
  - Very flexible and easy to update
  - Performance and timing issues in timing critical applications
- Software with generic hardware accelerator (GPU, NPU)
  - Relies on standard HW
  - Limited flexibility
  - Power consumption and timing issues
- Software with bespoke hardware accelerator
  - Requires development of custom HW
  - Low power and predictable timing

# Centralized or Distributed Computation

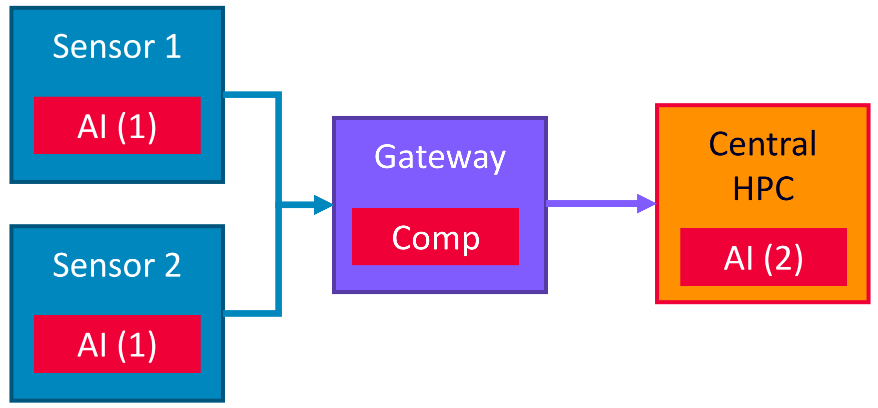
## Centralized computation

- Uncompressed data through network
- High computational load on HPC
- Flexible
- High power consumption

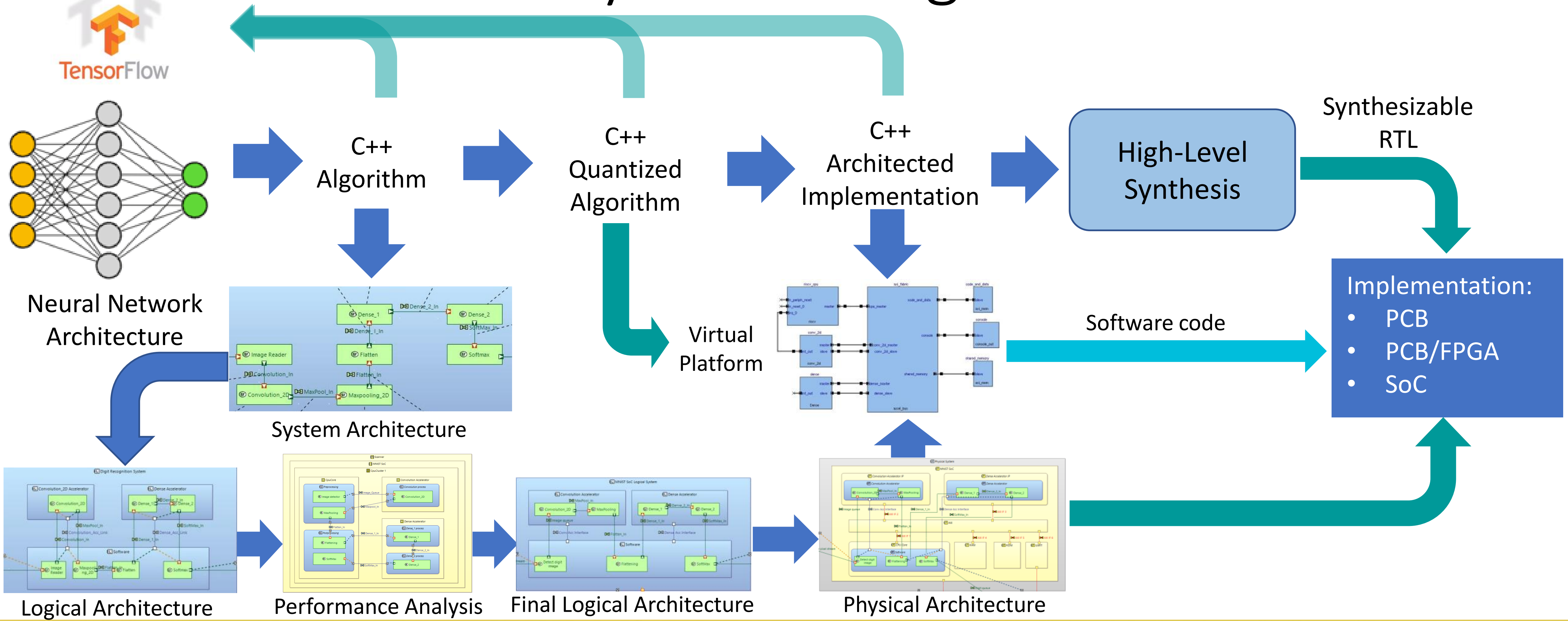


## Distributed computation

- Pre-processing data in its‘ origin
- Load shared across multiple components
- Data amount reduced by pre-processing and compression
- Low power consumption through dedicated HW



# Model-Based AI System Design



# Model-Based Architecture Exploration

# Model-Based Architecture Exploration

## Model-Based Systems Engineering (MBSE)

- Formalized application of modeling to support system design
- Covers design, analysis, verification and validation activities throughout development

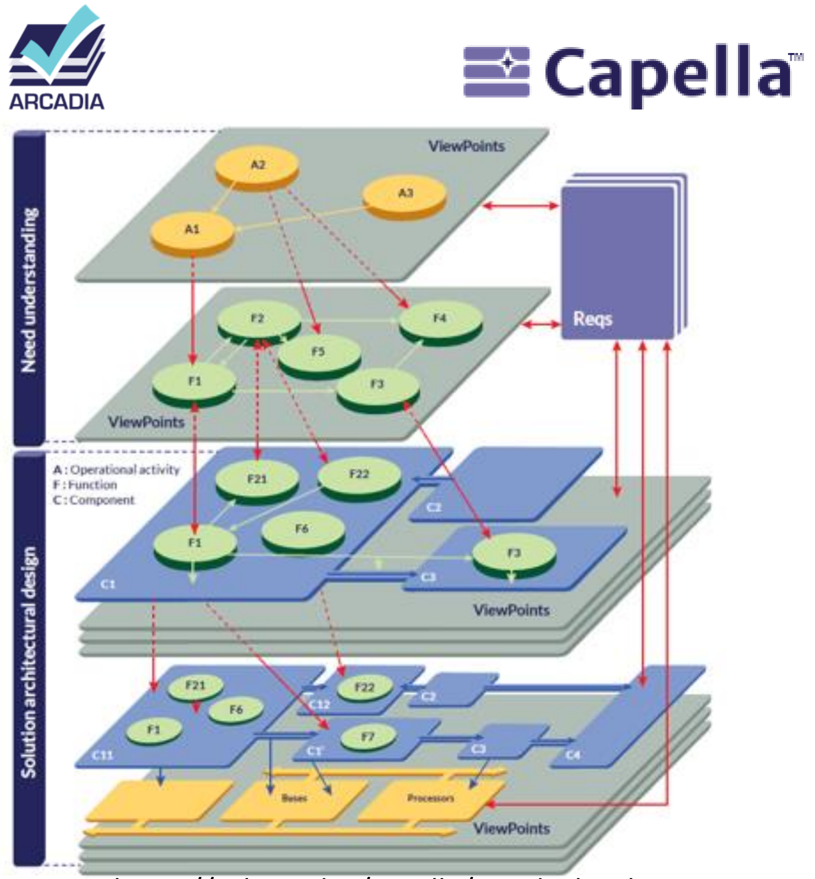
## MBSE enables abstract level system architecture exploration

- Functional analysis
- Mapping functions to architectural elements
- Architecture performance analysis
- Subsystem transitioning of system components
- Several formalized approaches available

# ARCADIA Methodology

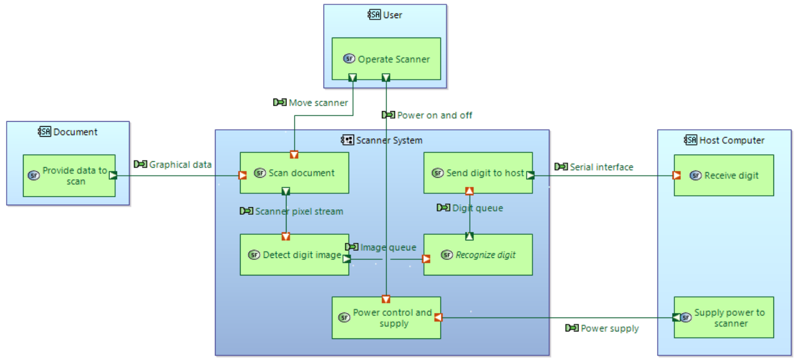
Tooled method to define, analyze, design & verify system, SW and HW architectures

- Operational Analysis
  - What the users of the system need to accomplish
- System Analysis
  - What the system has to accomplish for the user
- Logical Architecture
  - How the system will work to fulfill expectations
- Physical Architecture
  - How the system will be developed and built

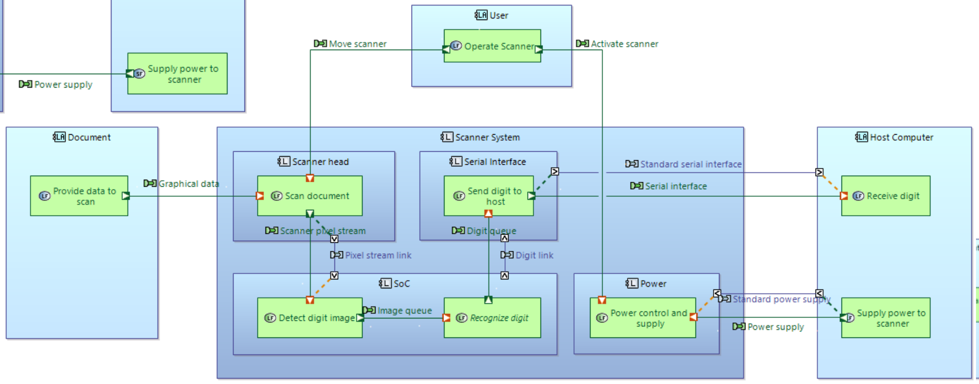


Source: https://eclipse.dev/capella/arcadia.html

# Top-Level System Exploration

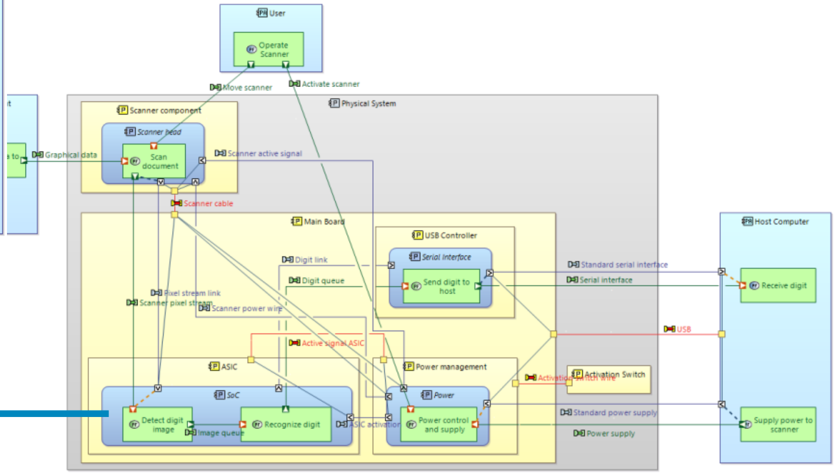


System functional architecture



System logical architecture

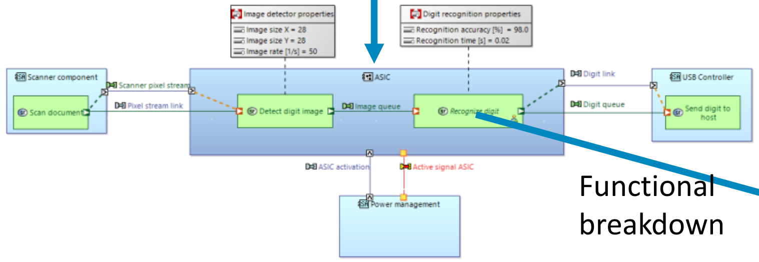
System Physical architecture



Subsystem transition

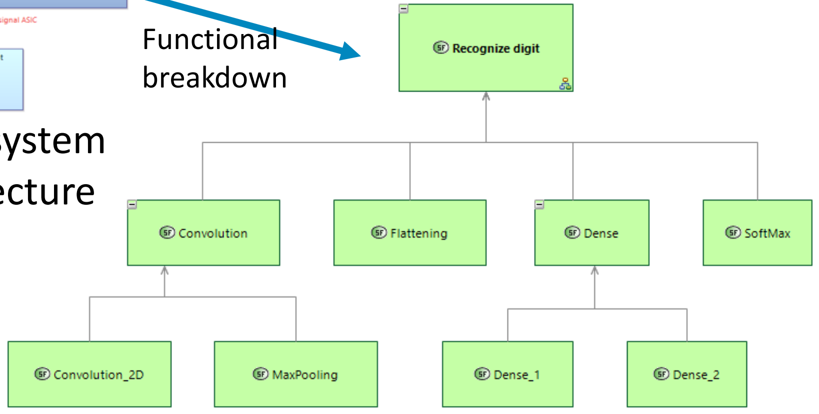
# Subsystem Transition and Decomposition

Subsystem transition



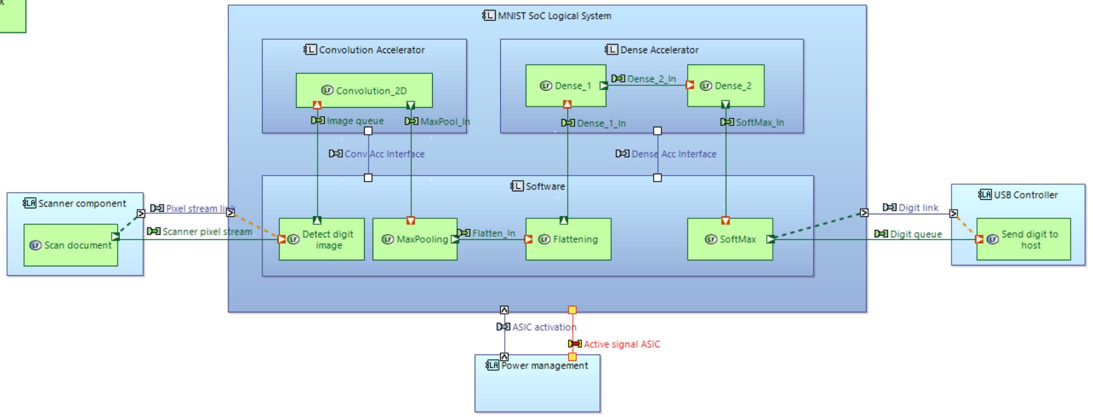
Functional breakdown

Transitioned subsystem functional architecture

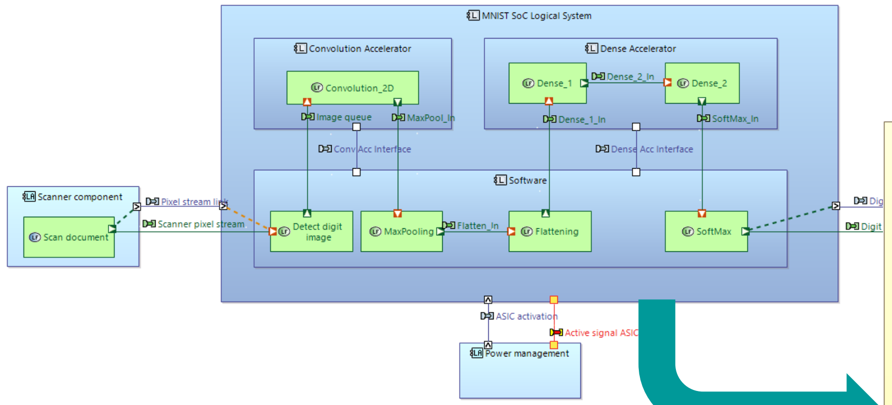


Subsystem detailed functional architecture

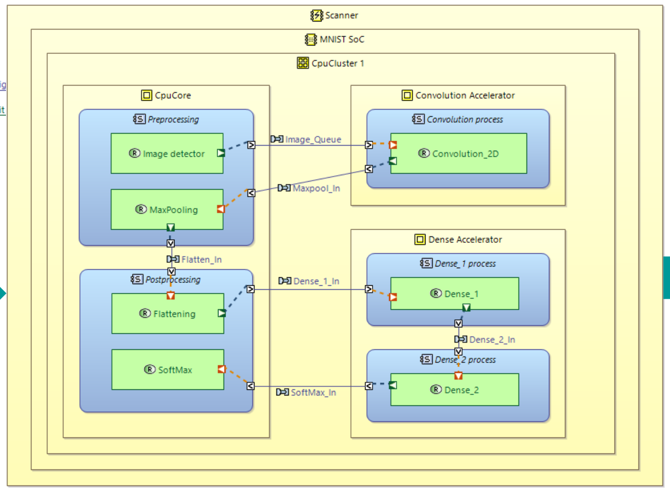
Subsystem logical architecture



# Subsystem Architecture Exploration

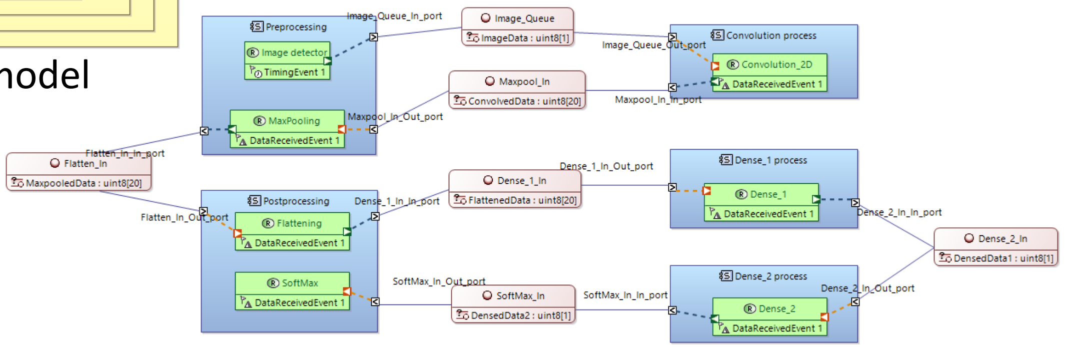


Manual design step



Subsystem performance model

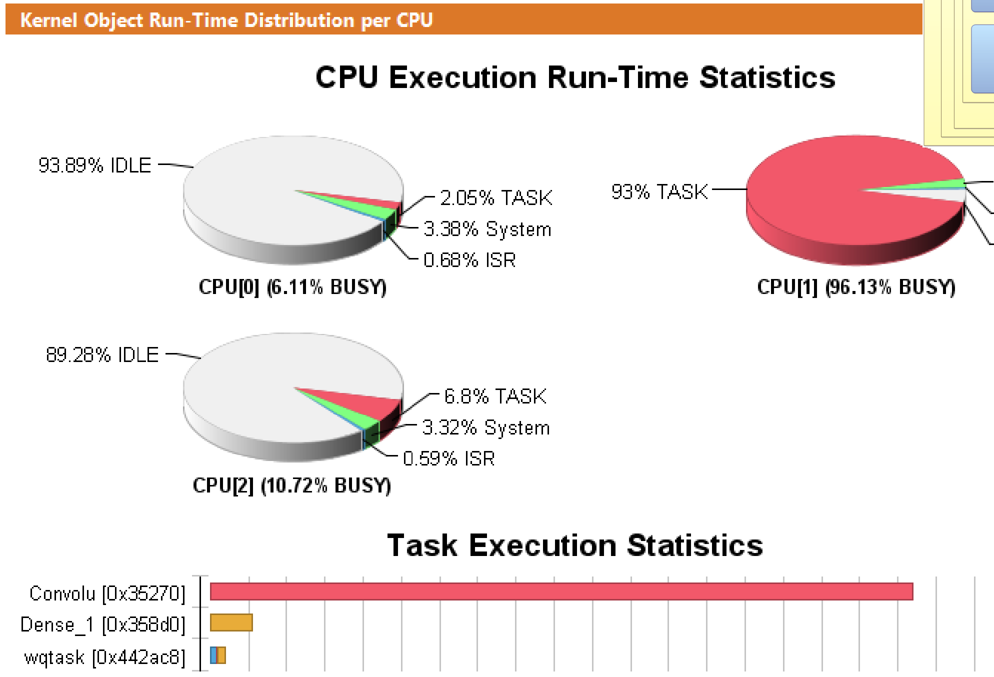
Automatic model creation

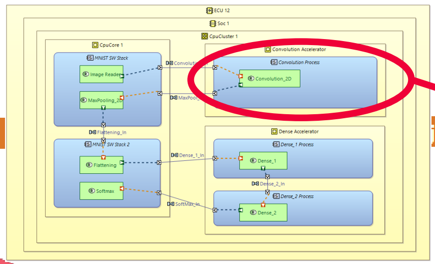


Subsystem performance simulation model

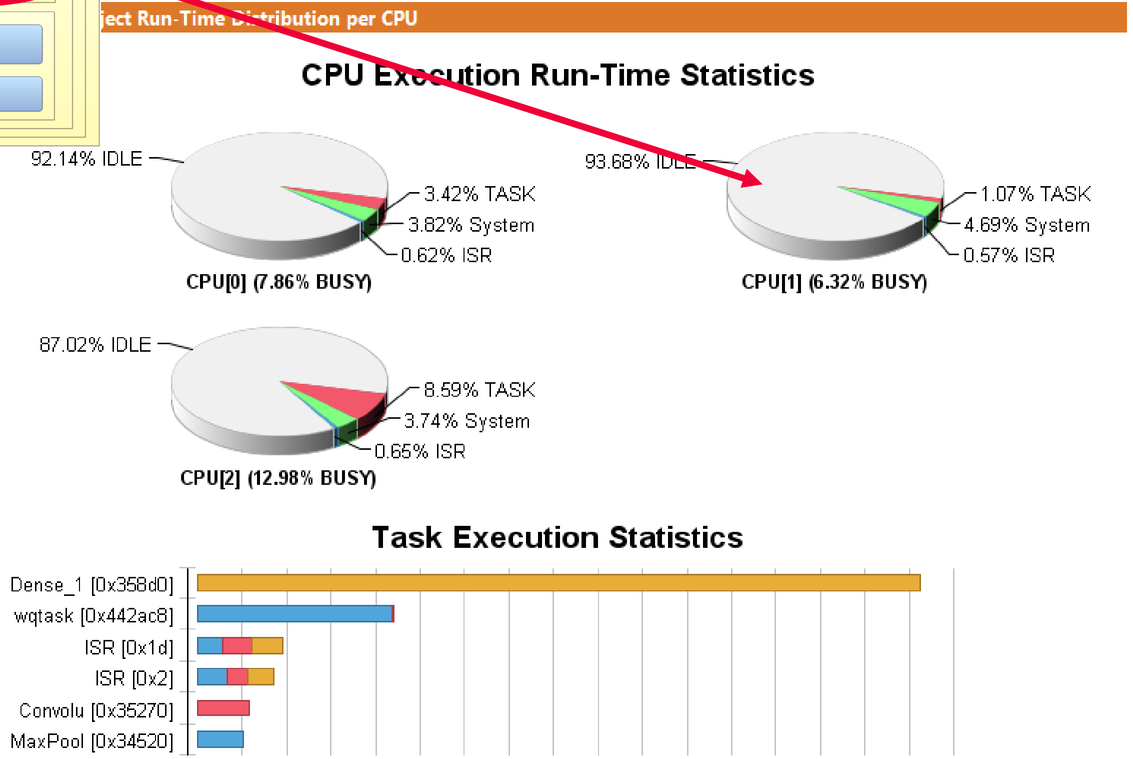
# Analyzing Performance Simulation Results

Full SW implementation with 3 CPU cores





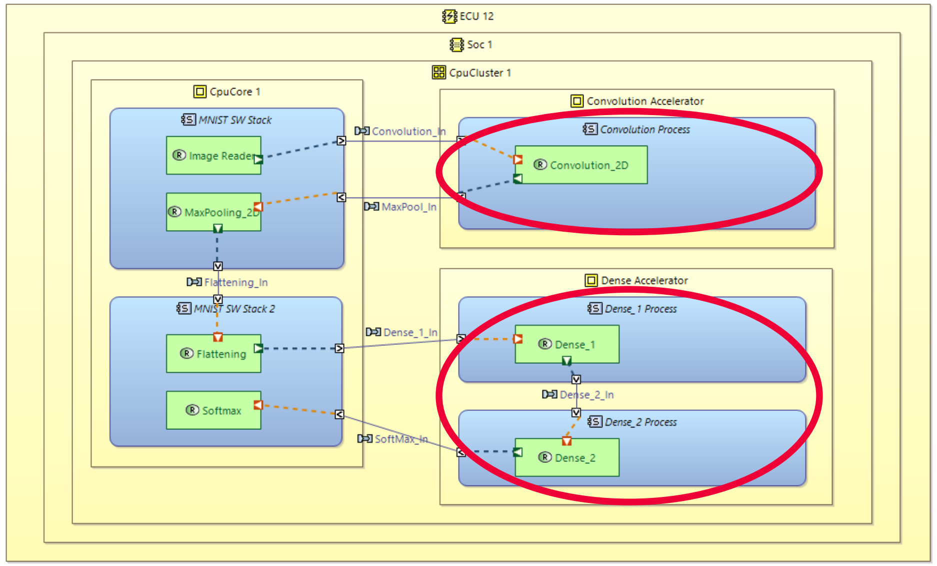
Accelerated implementation with 2 CPU cores

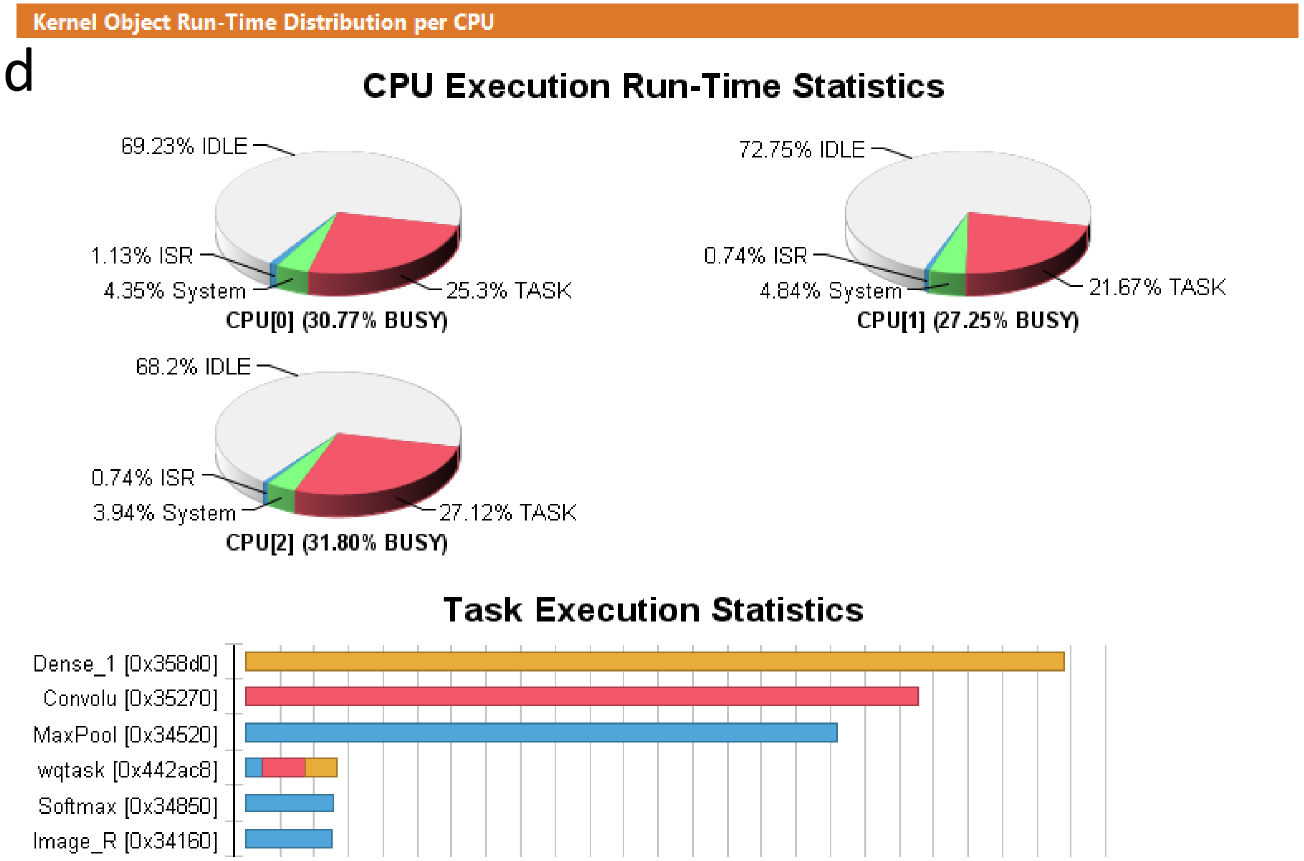


# Analyzing Performance Simulation Results contd

Architecture with 2 HW accelerators and one CPU core

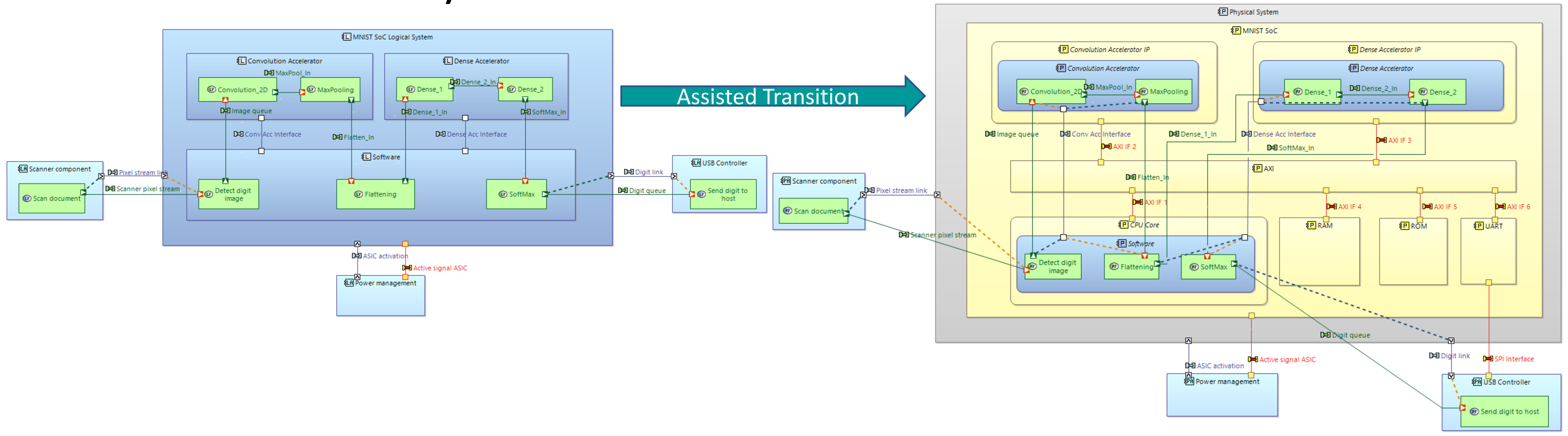
- Clock frequency: 500 MHz
- Interval between inferences: 20ms





# Final Subsystem Architecture

Logical architecture is updated based on the simulation results and transitioned to Physical Architecture



2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

## Model-Based Design Process

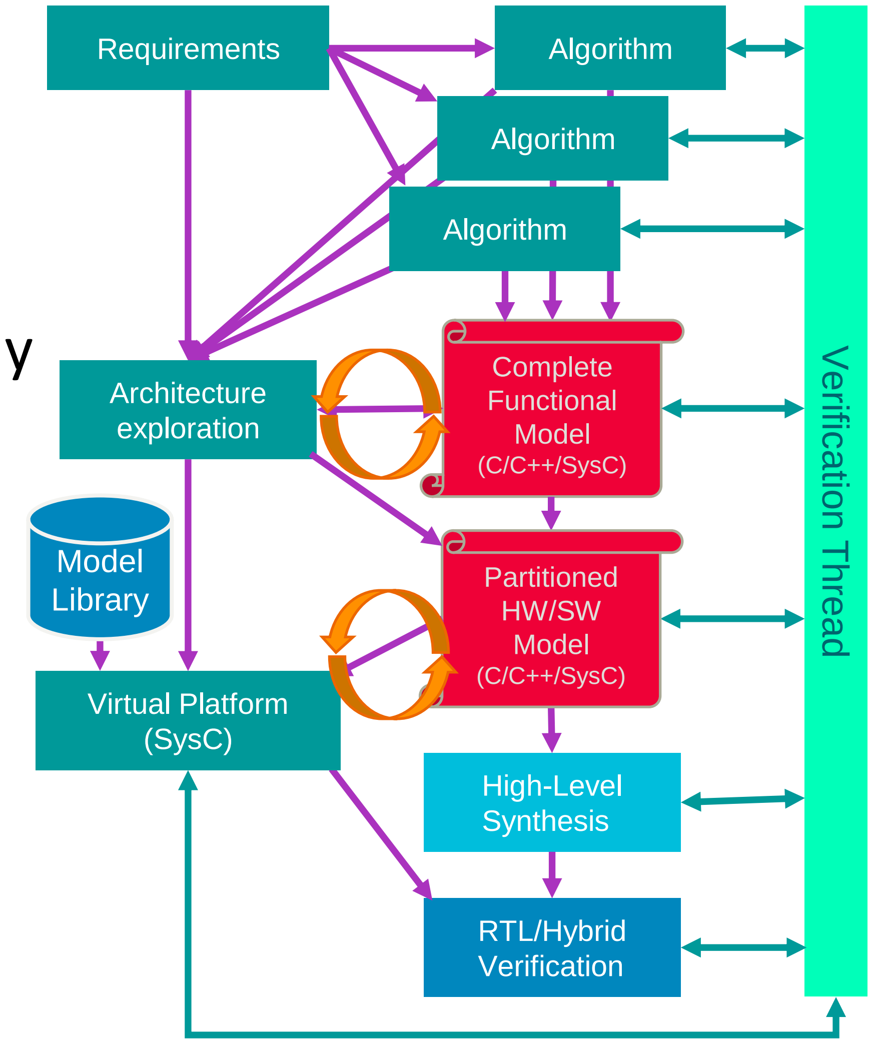
accellera SYSTEMS INITIATIVE

# HW/SW Co-Architecting

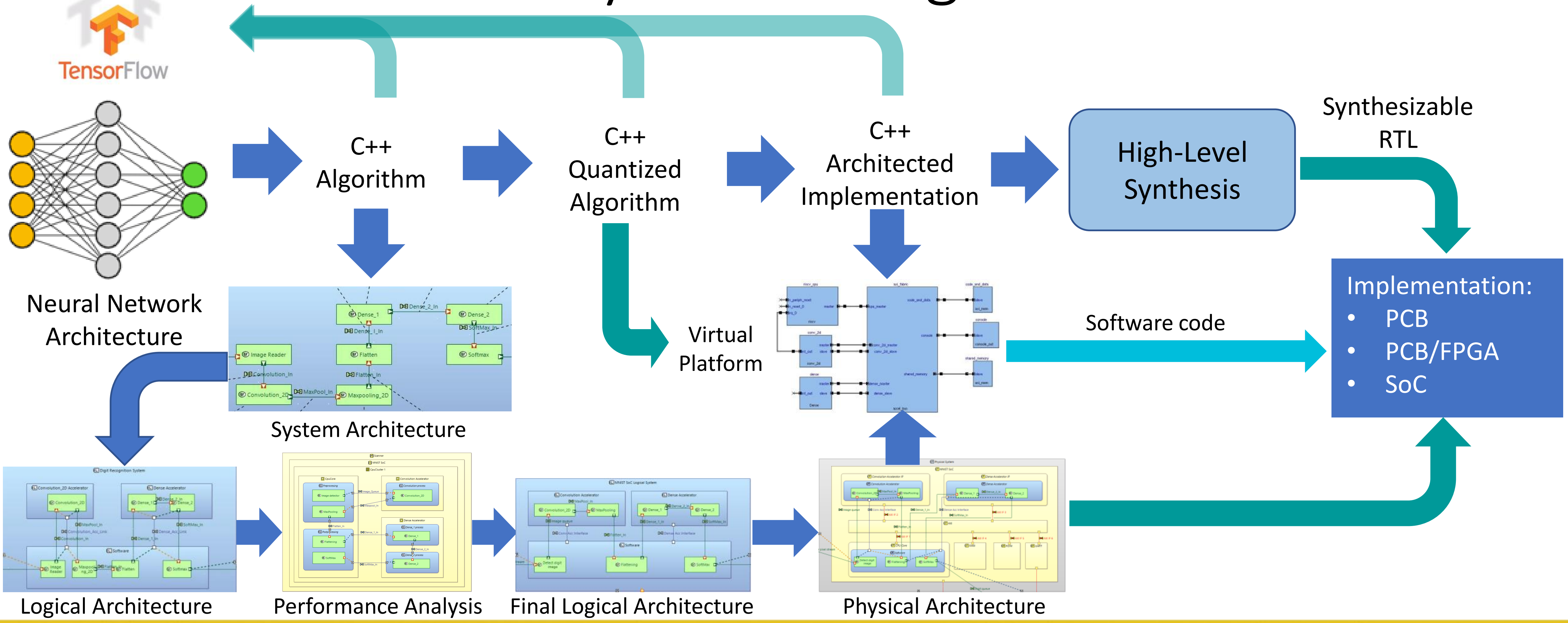
**Iterative multi-abstraction level process**

1. Individual pieces of algorithm developed separately
2. Complete functional model in C/C++
3. Architecture exploration
4. Partitioned HW/SW model
5. Virtual Hardware model with custom accelerators
6. Partitioning optimization
7. High-Level Synthesis of custom accelerators

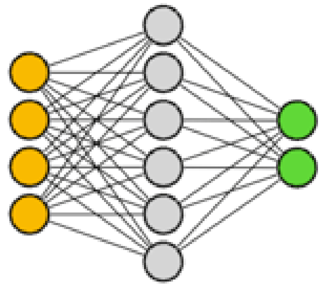
Continuous verification throughout the process



# Model-Based AI System Design



# Optimize Neural Network



Neural Network Architecture

- In the datacenter, size and speed are secondary considerations
- On the edge we need to be fast and compact
  - Fewer layers
  - Fewer channels

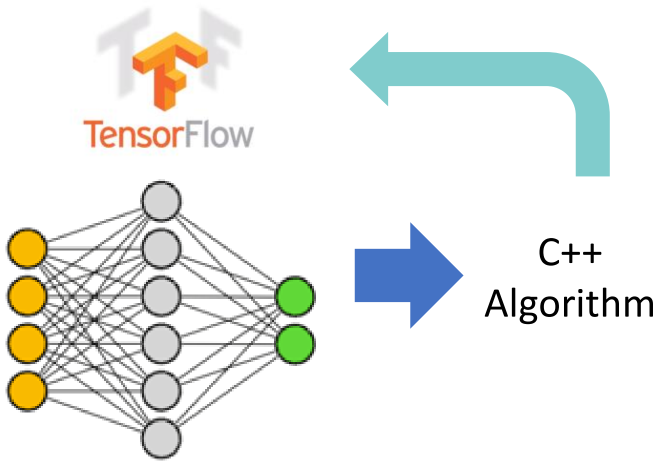
**Original MNIST network**

```
MAC operations:            2357000
Number of parameters:      1966030
Minimum data transfer:     1971244 words
```

**Optimized MNIST network**

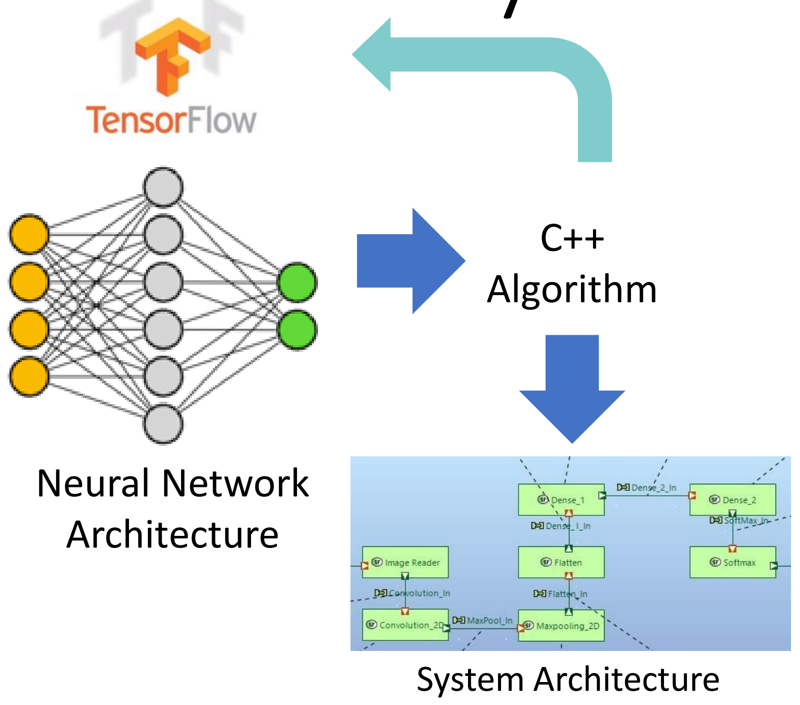
```
MAC operations:             235400
Number of parameters:        39690
Minimum data transfer:       42464 words
```

## Create an equivalent C++ model



- To use co-architecting flow we need to convert the algorithm to C++
- Only include programmability in the C++ for parameters you will vary

## Create System (Functional) Architecture



- Specifies different functions and their dependencies
- Levels of details depends on the system hierarchy level
- Relevant parameters attached to functions as properties
- Functional breakdown needed, when moving down in subsystem hierarchy

# Functional Breakdown and Parameterization

Subsystem transition

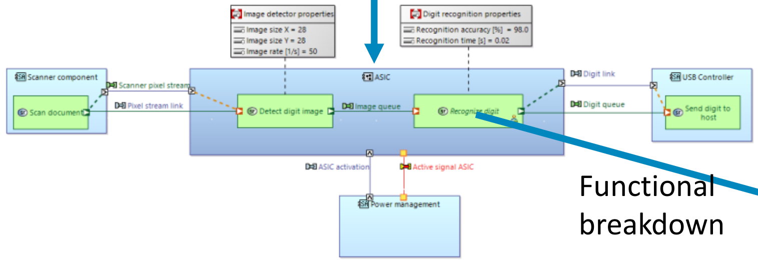
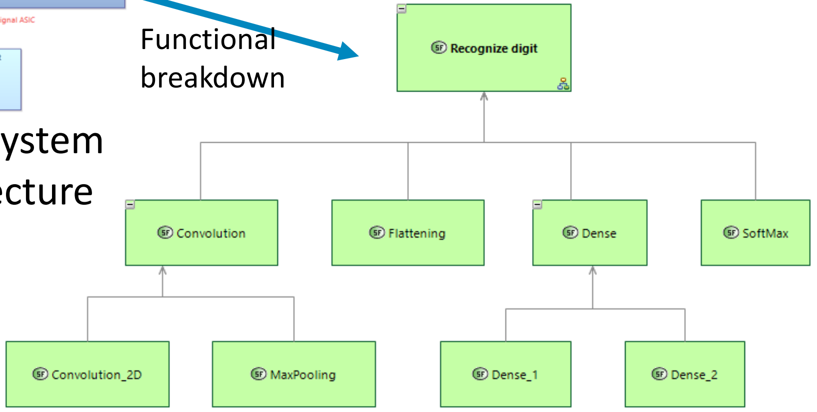


Image detector properties
- Image size X = 28
- Image size Y = 28
- Image rate [1/s] = 50

Digit recognition properties
- Recognition accuracy [%] = 98.0
- Recognition time [s] = 0.02

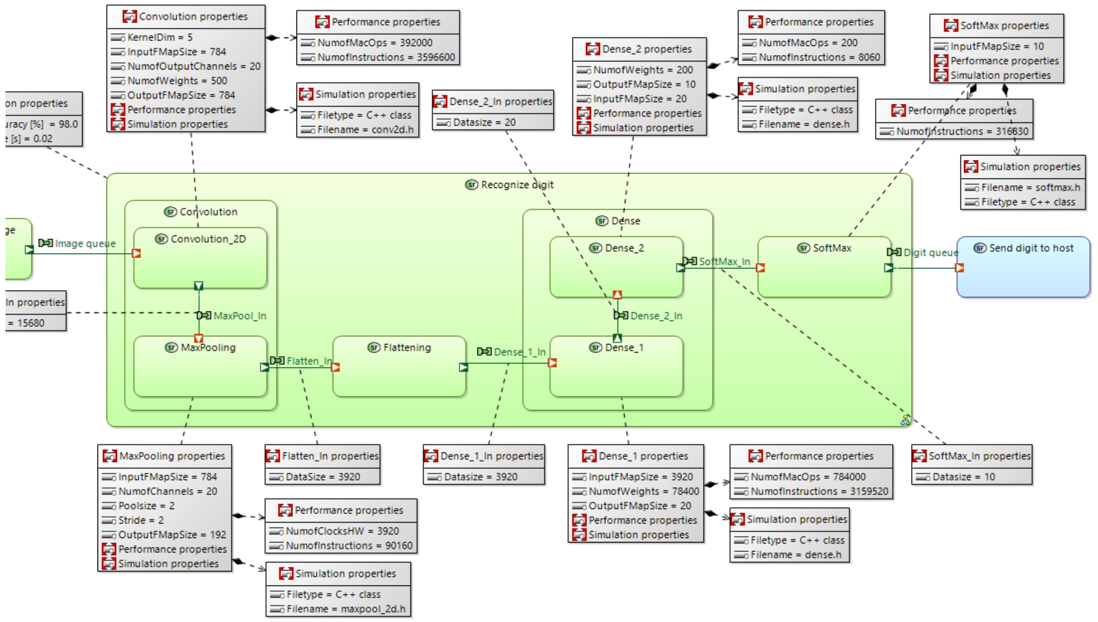
Transitioned subsystem functional architecture

Functional breakdown



Subsystem functional breakdown

Subsystem detailed parameterized functional architecture



Convolution properties
- KernelDim = 5
- InputFMapSize = 784
- NumofOutputChannels = 20
- NumofWeights = 500
- OutputFMapSize = 784
- Performance properties
- Simulation properties

Performance properties
- NumofMacOps = 392000
- NumofInstructions = 3596600

Simulation properties
- Filetype = C++ class
- Filename = conv2d.h

Dense_2_In properties
- Datasize = 20

Dense_2 properties
- NumofWeights = 200
- OutputFMapSize = 10
- InputFMapSize = 20
- Performance properties
- Simulation properties

Performance properties
- NumofMacOps = 200
- NumofInstructions = 8060

Simulation properties
- Filetype = C++ class
- Filename = dense.h

SoftMax properties
- InputFMapSize = 10
- Performance properties
- Simulation properties

Performance properties
- NumofInstructions = 316830

Simulation properties
- Filename = softmax.h
- Filetype = C++ class

on properties
- uracy [%] = 98.0
- [s] = 0.02

In properties
- = 15680

MaxPooling properties
- InputFMapSize = 784
- NumofChannels = 20
- Poolsize = 2
- Stride = 2
- OutputFMapSize = 192
- Performance properties
- Simulation properties

Performance properties
- NumofClocksHW = 3920
- NumofInstructions = 90160

Simulation properties
- Filetype = C++ class
- Filename = maxpool_2d.h

Flatten_In properties
- DataSize = 3920

Dense_1_In properties
- Datasize = 3920

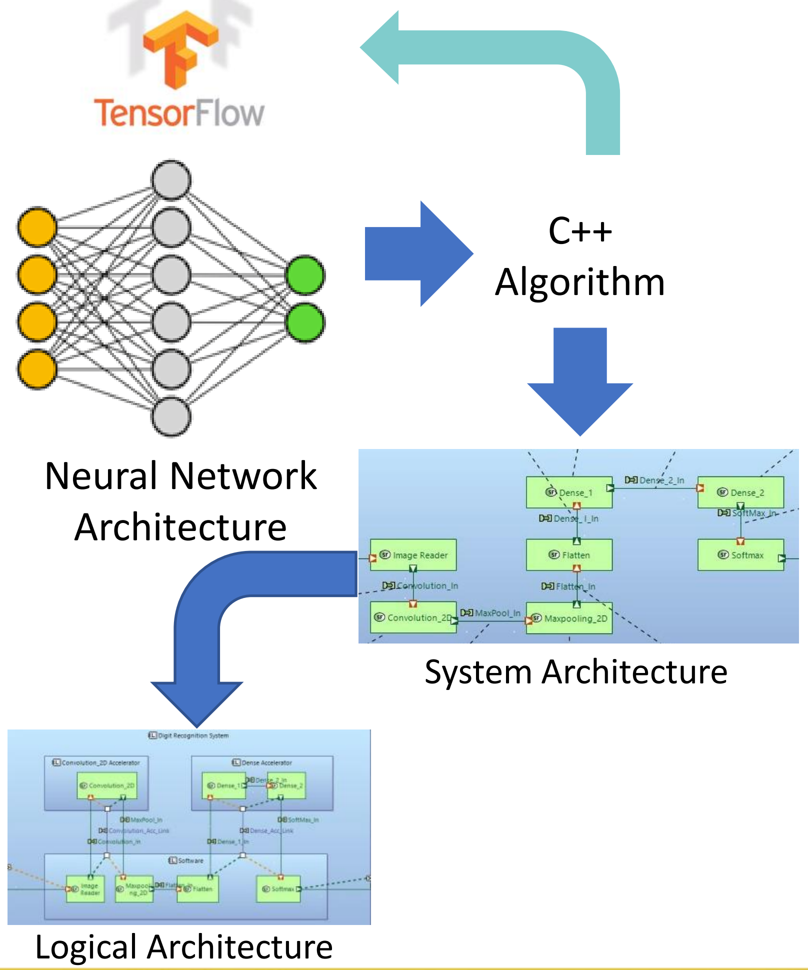
Dense_1 properties
- InputFMapSize = 3920
- NumofWeights = 78400
- OutputFMapSize = 20
- Performance properties
- Simulation properties

Performance properties
- NumofMacOps = 784000
- NumofInstructions = 3159520

Simulation properties
- Filetype = C++ class
- Filename = dense.h

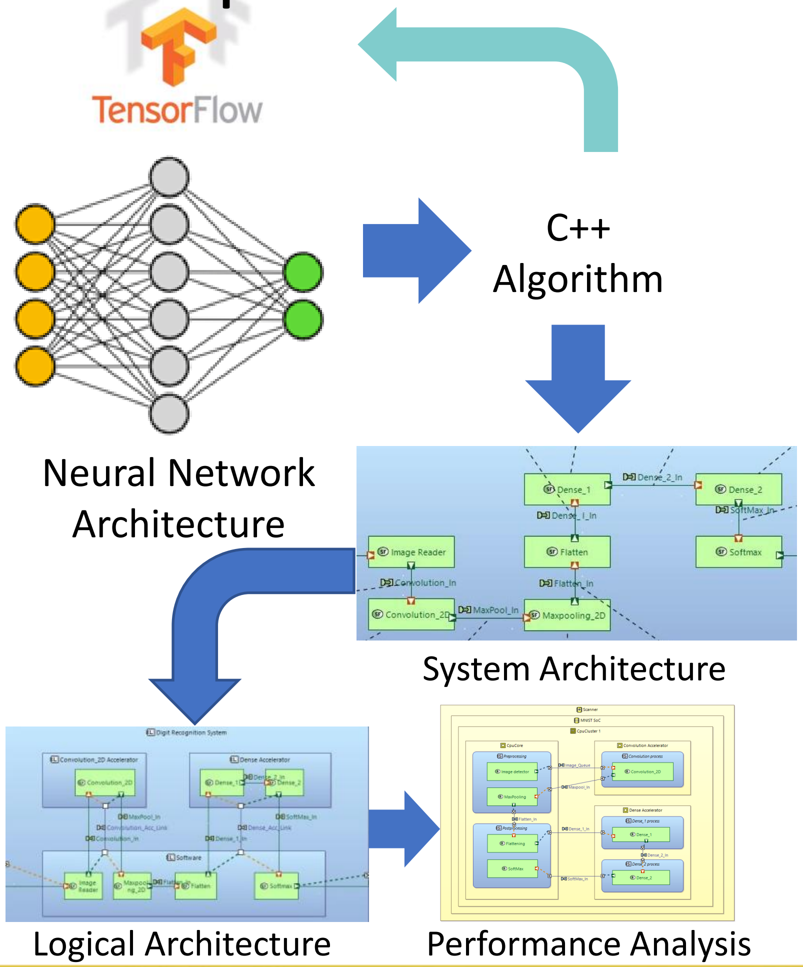
SoftMax_In properties
- Datasize = 10

# Create Initial Logical Architecture



- Initial system architecture
- Group system functions into logical components based on
  - Performance metrics (MAC operations, profiling data, etc.)
  - Data sizes of the function exchanges
- Allocate function exchanges to component exchanges

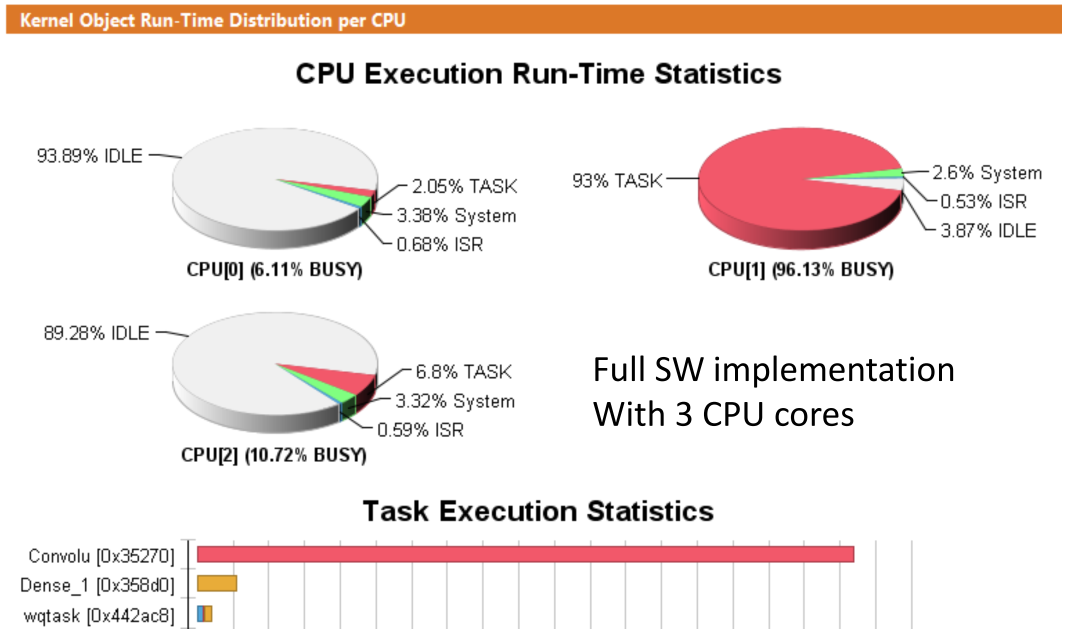
# Explore Different Architecture Options



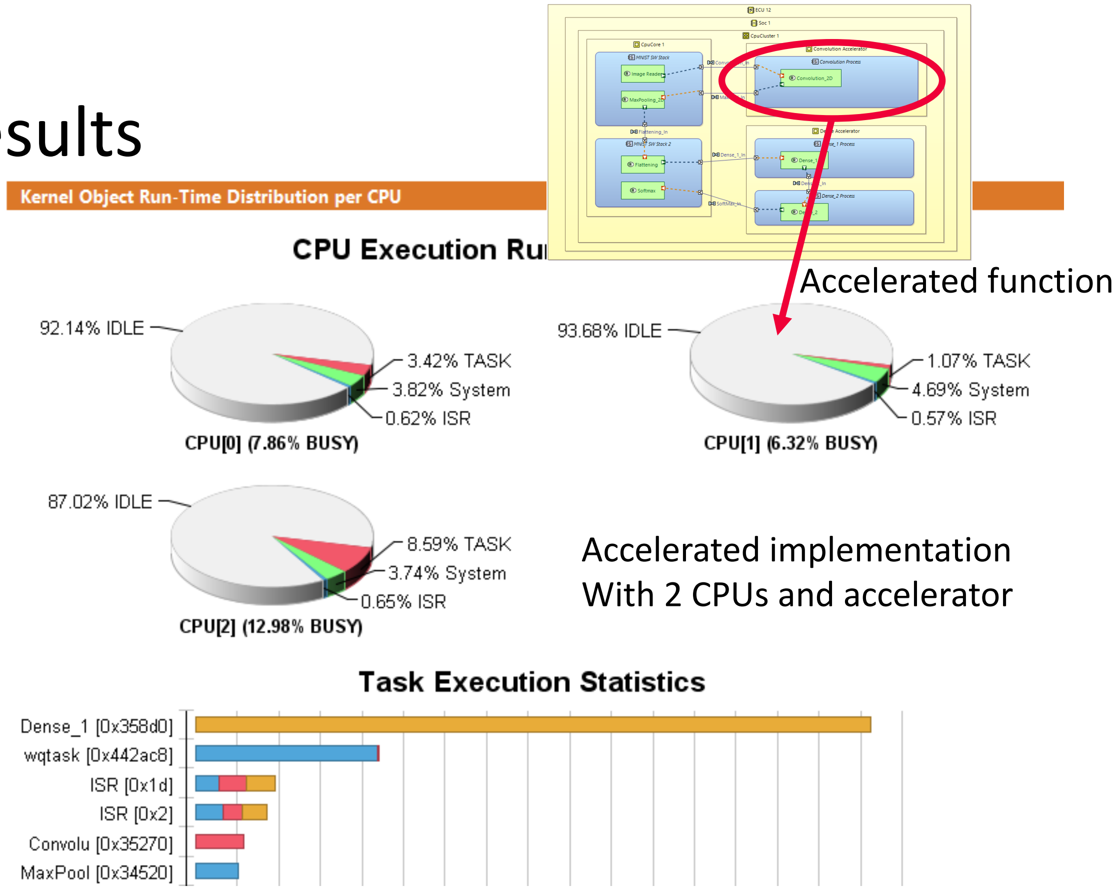
- Create initial performance analysis model based on the logical architecture
- Explore different allocations
  - Multi-core
  - Multi-cluster
  - Architectures with hardware accelerators
  - Multi-chip
  - Multiboard
  - …

# Runtime and Data Communication Analysis

## Analyzing performance simulation results

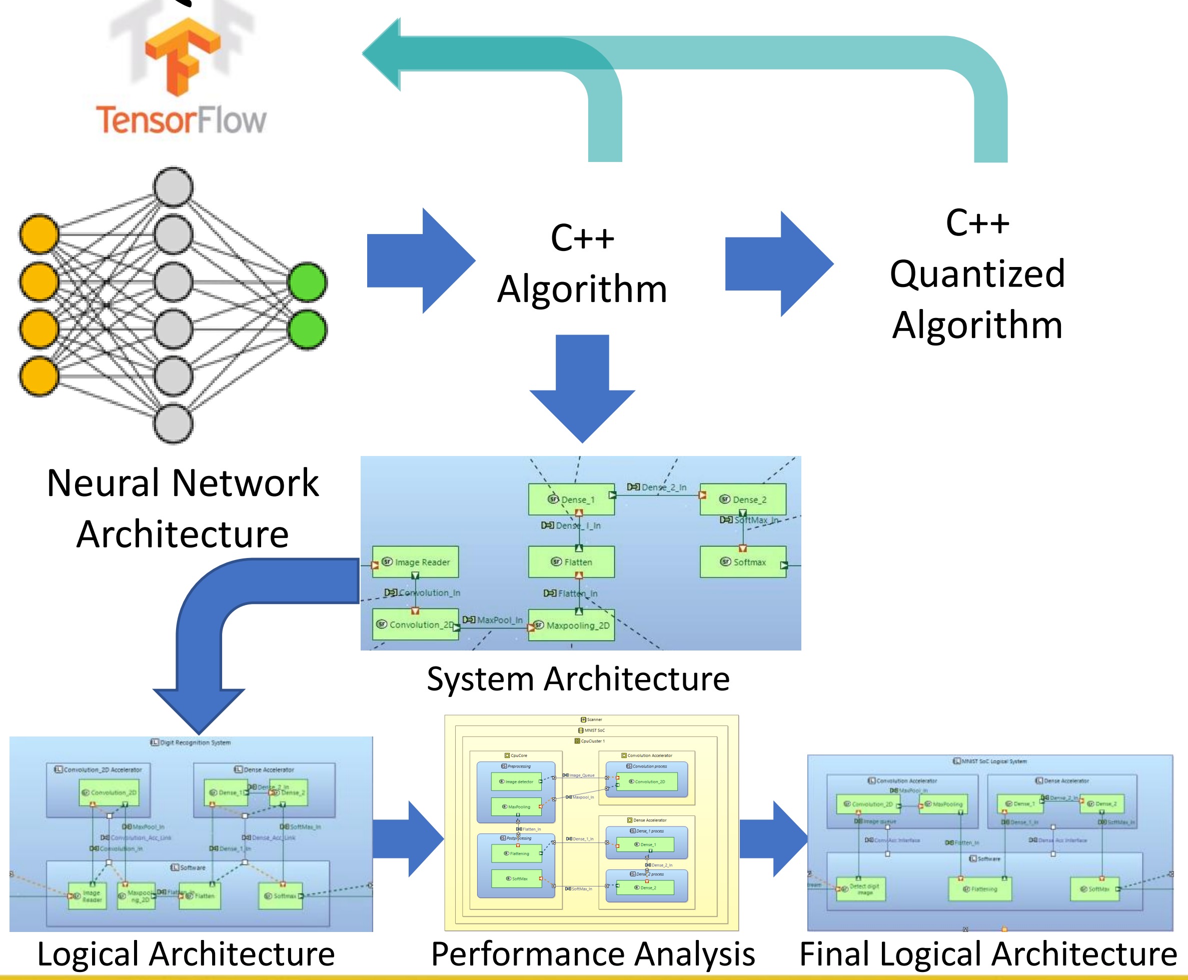


Full SW implementation
With 3 CPU cores



Accelerated implementation
With 2 CPUs and accelerator

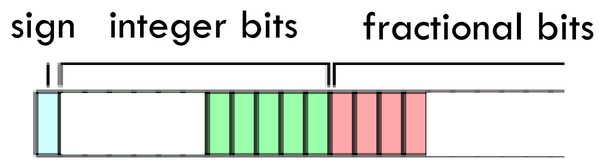
# Quantize HW Functions & Update Logical Arch.



- Update logical architecture based on performance results
- Quantize functions that are allocated to hardware accelerators
- Functional verification of quantized functions

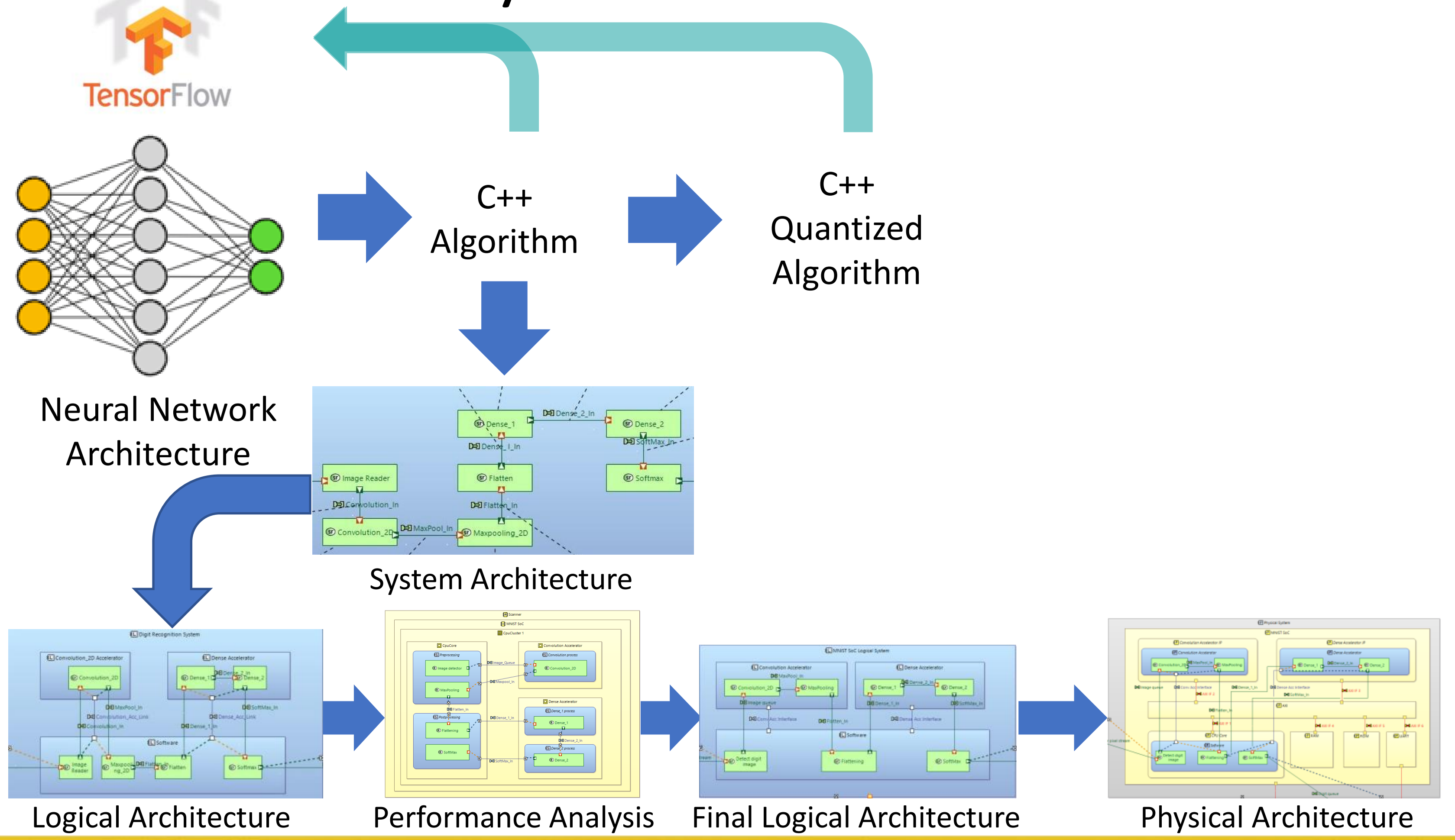
# Quantization of HW Functions

- Optimizing hardware word lengths to minimize HW area
  - Ideally every variable individually
- Fixed-point data types ideal
  - ac_fixed
  - sc_(u)fixed
- Several analysis methods available
  - Value Range Analysis -based (simulation based)
  - Static Analysis
  - Brute force



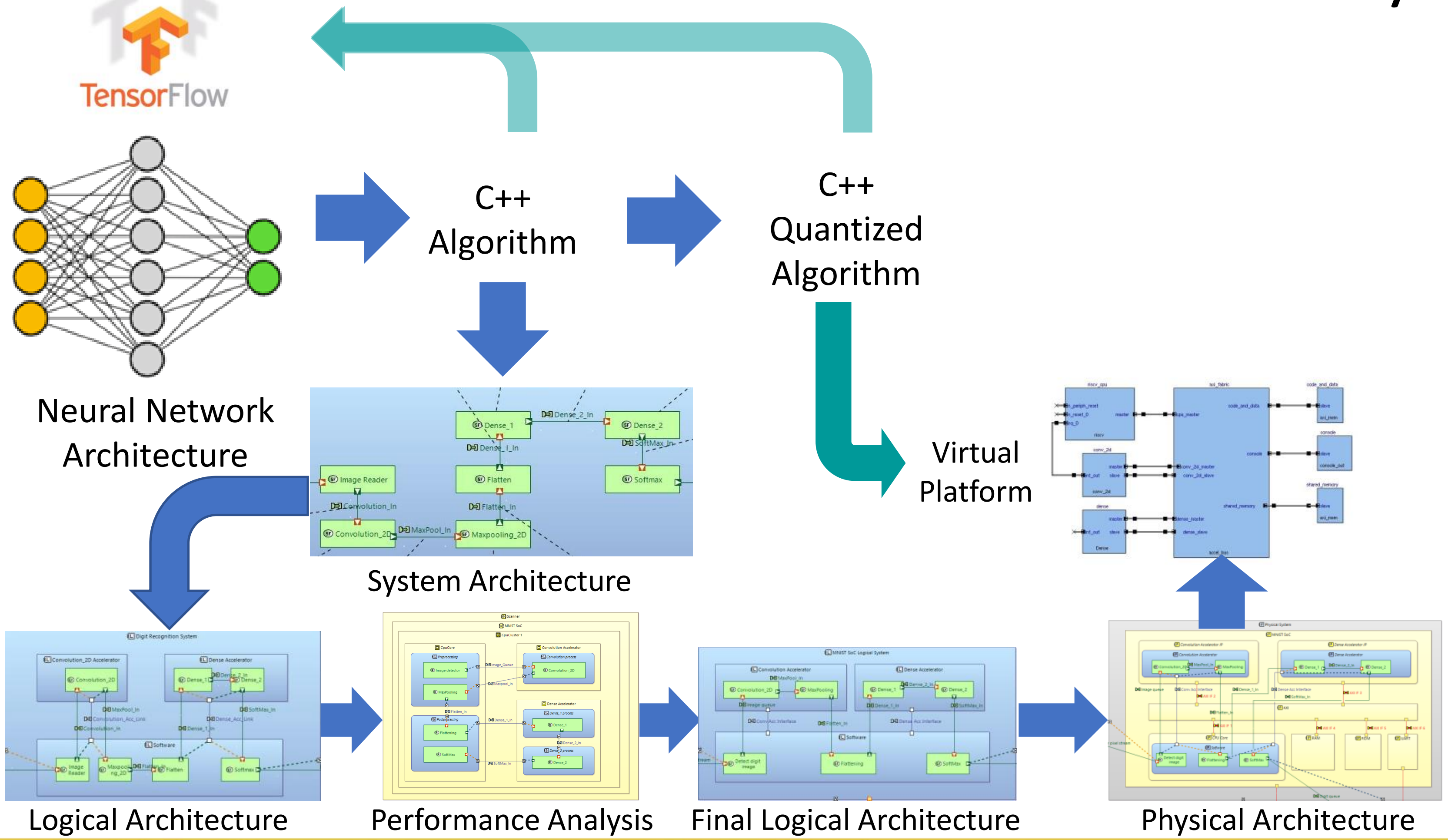
Any size you want

# Create Physical Architecture Model



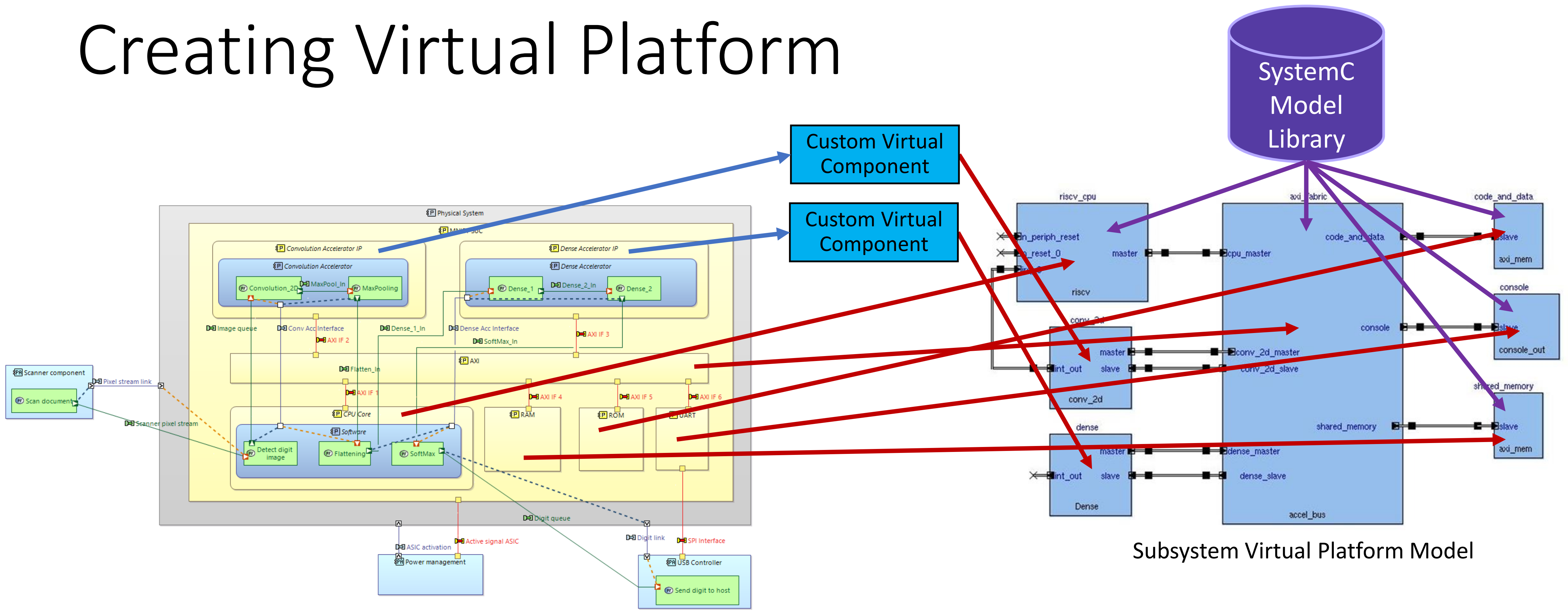
- Assisted transition of model
- Non-functional components added manually
  - Memories
  - Interconnects
  - Peripherals

# Create Virtual Platform from Physical Architecture



- Create SystemC model of hardware platform
- SW functions mapped to processors
- Initial drivers

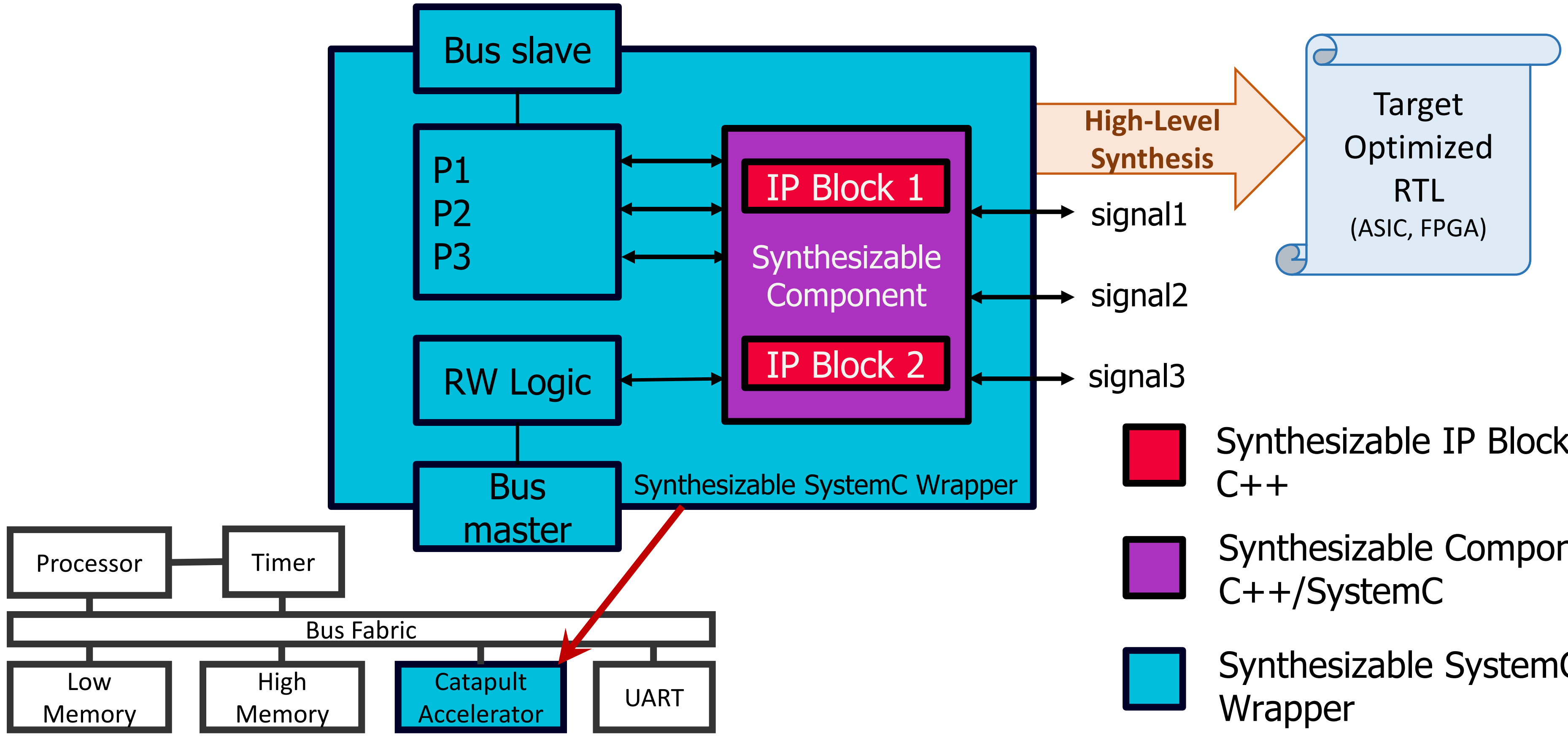
# Creating Virtual Platform



Subsystem Virtual Platform Model

Subsystem Physical Architecture Model

# Creating Custom Virtual Component



🟥 Synthesizable IP Block C++

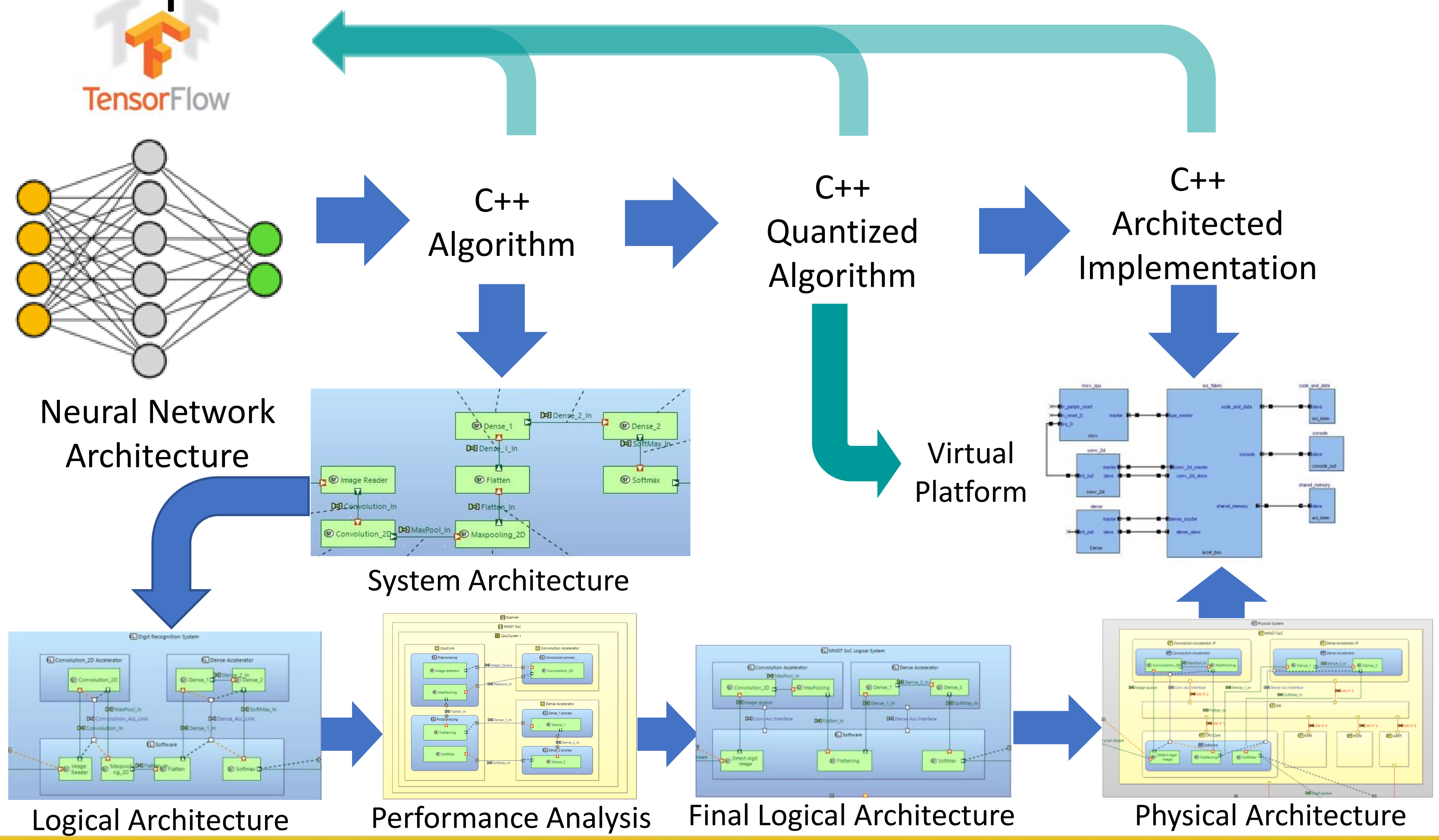
🟪 Synthesizable Component C++/SystemC

🟦 Synthesizable SystemC Wrapper

## Optimize accelerator model for HLS

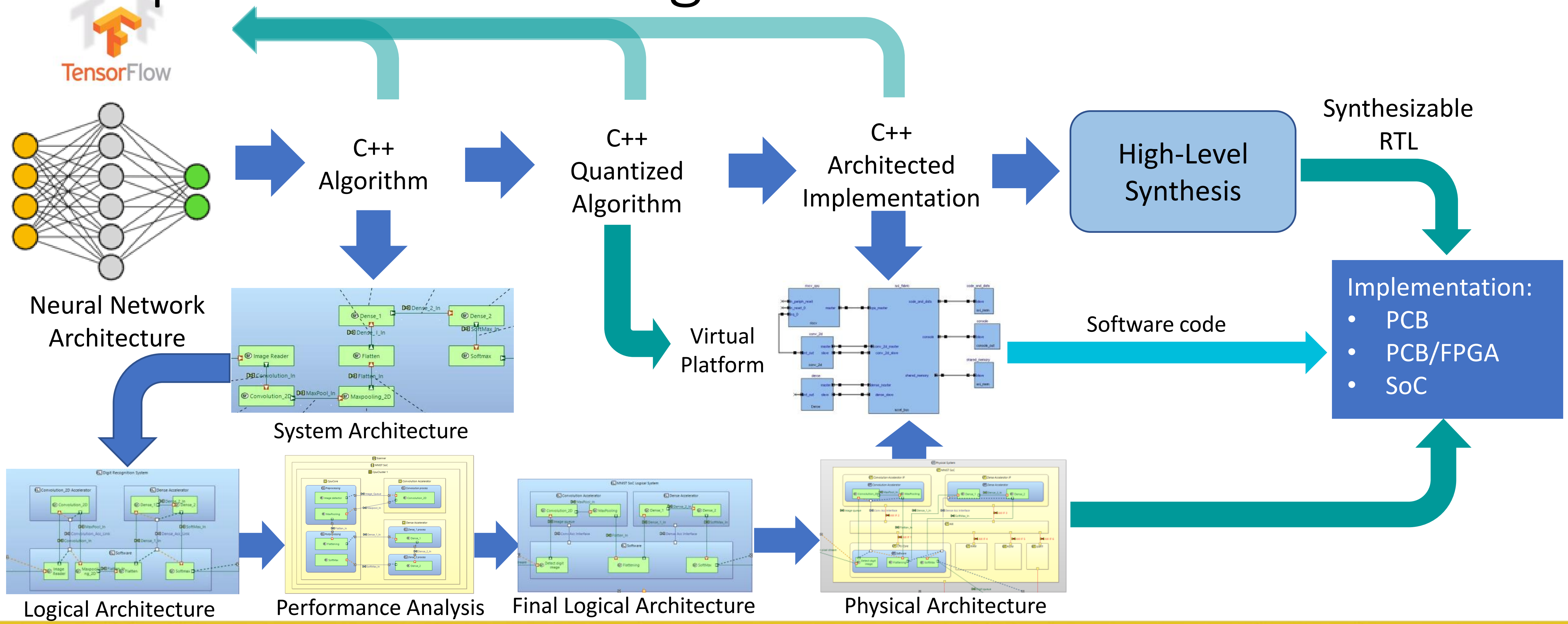
- Some code restructuring may be needed for optimal synthesis results
  - Block-level architecture
  - Loop order
  - Internal storages
  - Reusable functions
- Code modifications may influence algorithm accuracy
  - Continuous verification required

# Update Virtual Platform with detailed Accelerator

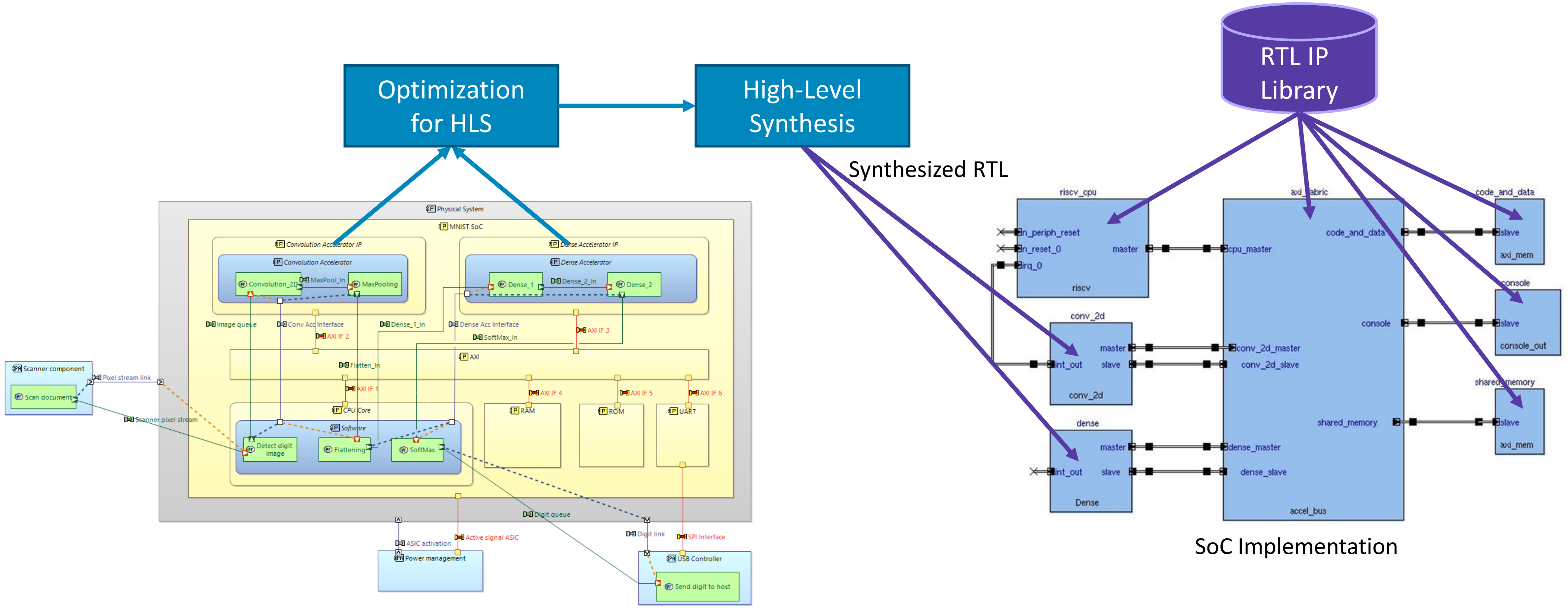


- No need to change platform architecture
- Bit-accurate function of accelerator
- Final platform for SW development

# Implement and Integrate HW and SW Modules



# Hardware Implementation



2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

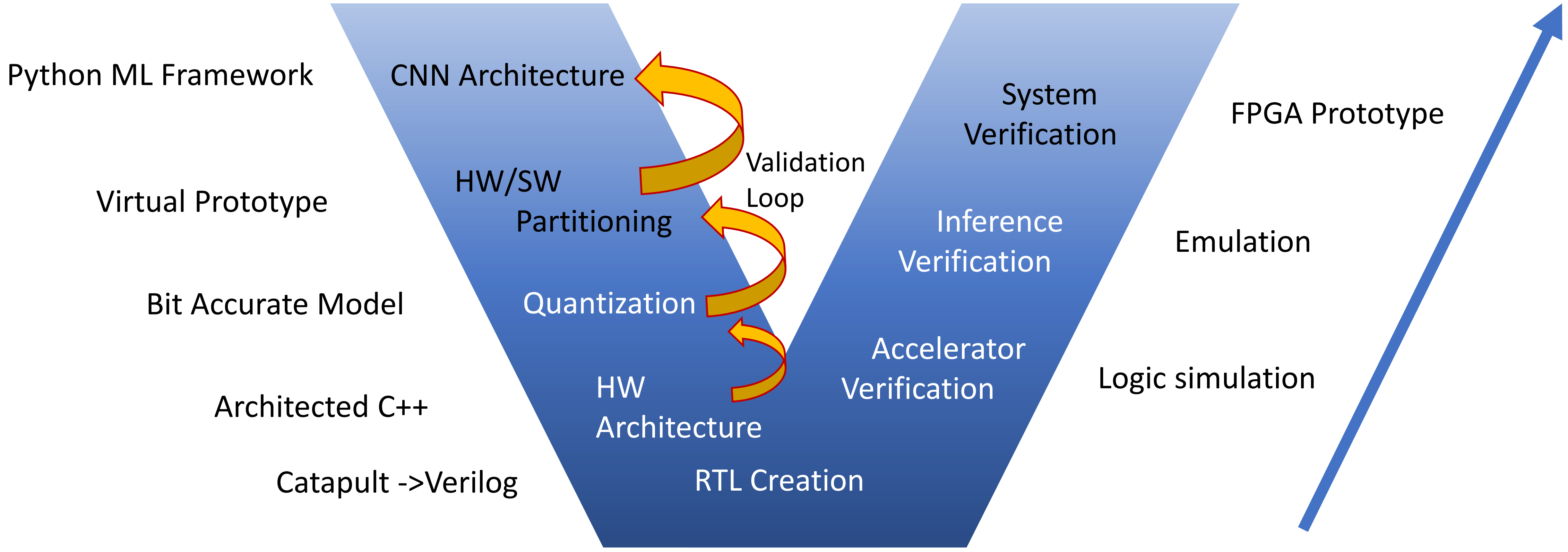
# Validation and Verification of HW/SW AI System

accellera SYSTEMS INITIATIVE

# Validation vs. Verification

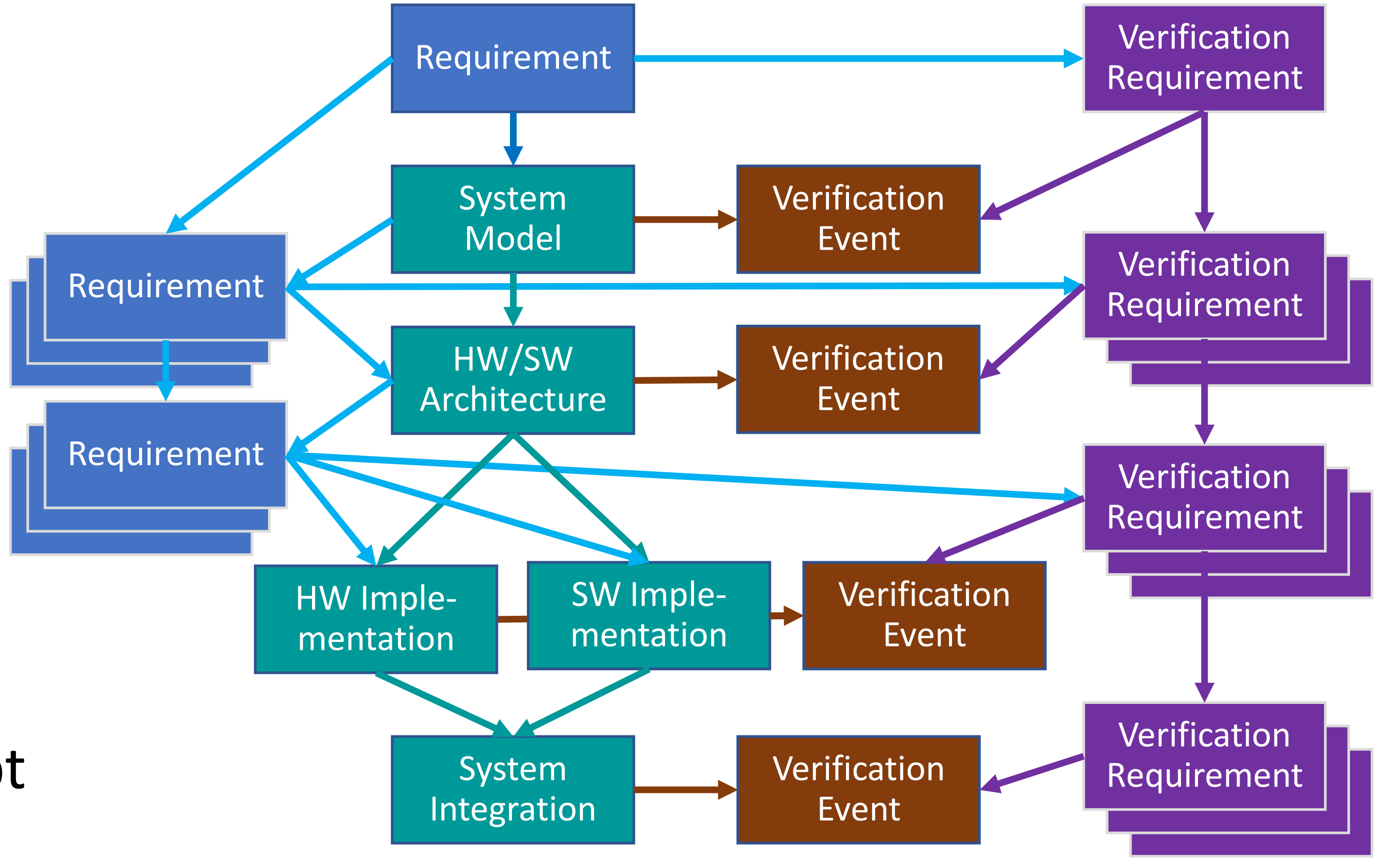
- Validation == Are we doing the right thing?
  - Performed during design phase of the project
  - Comparing the model to higher abstraction level or requirement
  - Usually simulation between consecutive design steps
- Verification == Did we implement it correctly?
  - Performed during implementation/integration phase of the project
  - Implementation vs. requirements at the same level
  - Implementation vs. model at the same level
  - Test coverage, corner cases, etc.
  - Simulations, formal analysis, emulation, prototyping

# Validation and Verification



# Multi-Abstraction-Level Verification Challenges

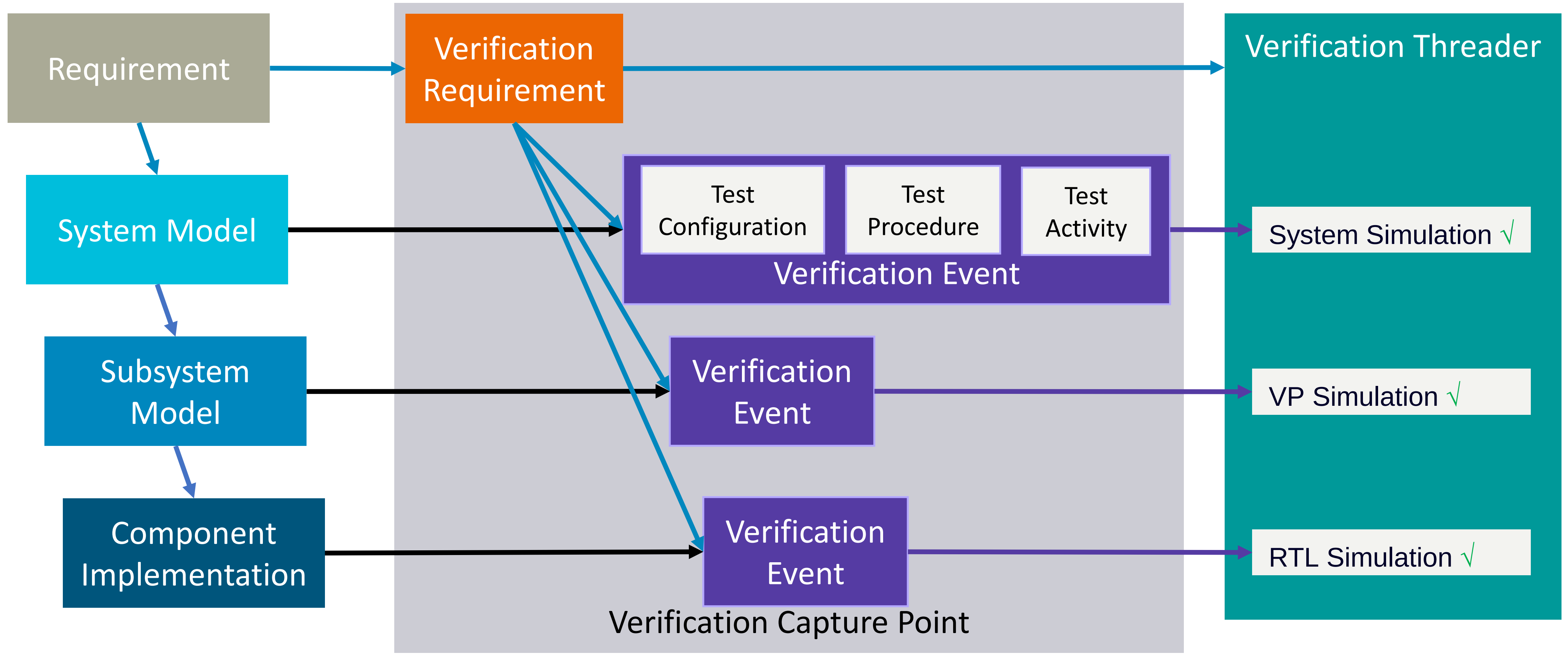
- Requirements driven verification with
  - Parameterizable requirements
  - Verification requirements
  - Requirement refinement
  - Hierarchical requirements
- Continuous verification and requirements tracing
  - Multi-layer verification concept



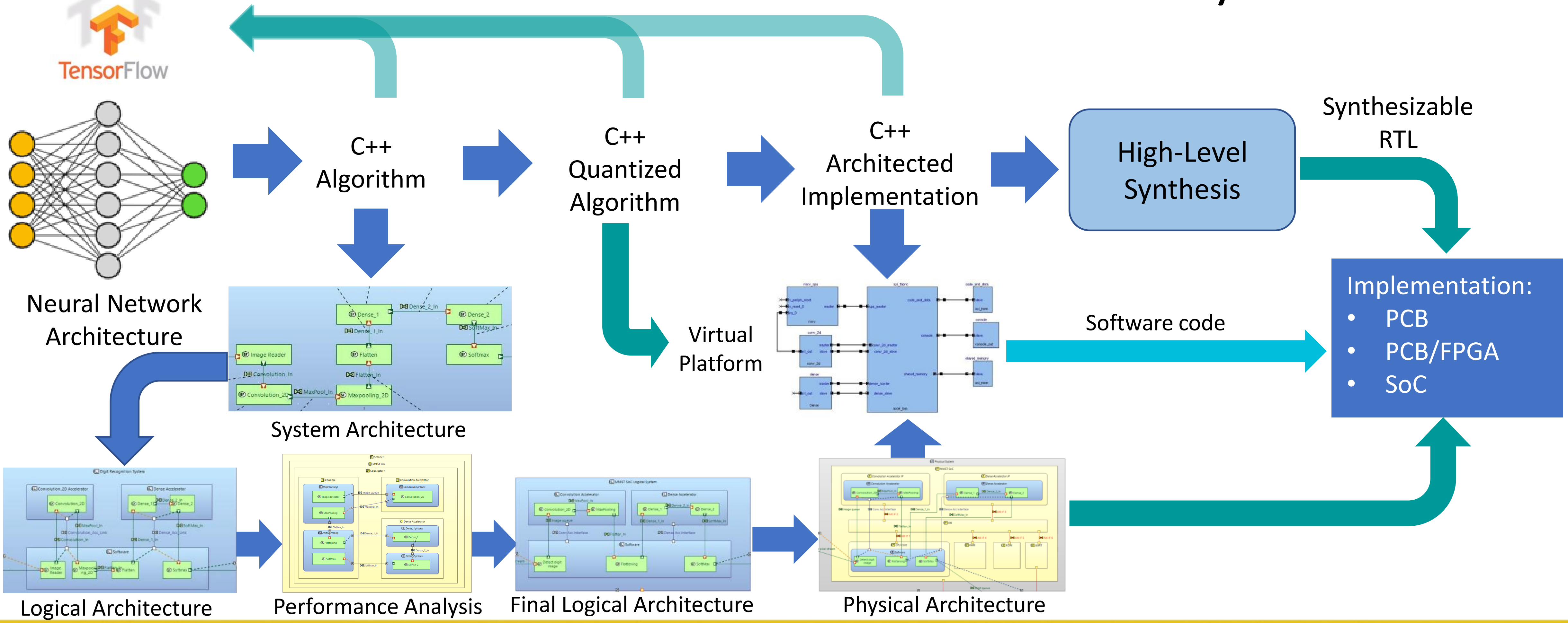
# Verification Process in Model-Based Design

- Requirements driven process:
  - Verification requirement defines test event
    - Test procedure
    - Test activity
    - Test configuration
  - Refined and hierarchical requirements need their own test events
- Verification Capture Point (VCP)
  - Bundles all test events related to one test requirement together
  - Contains test events in different design phases and abstraction levels

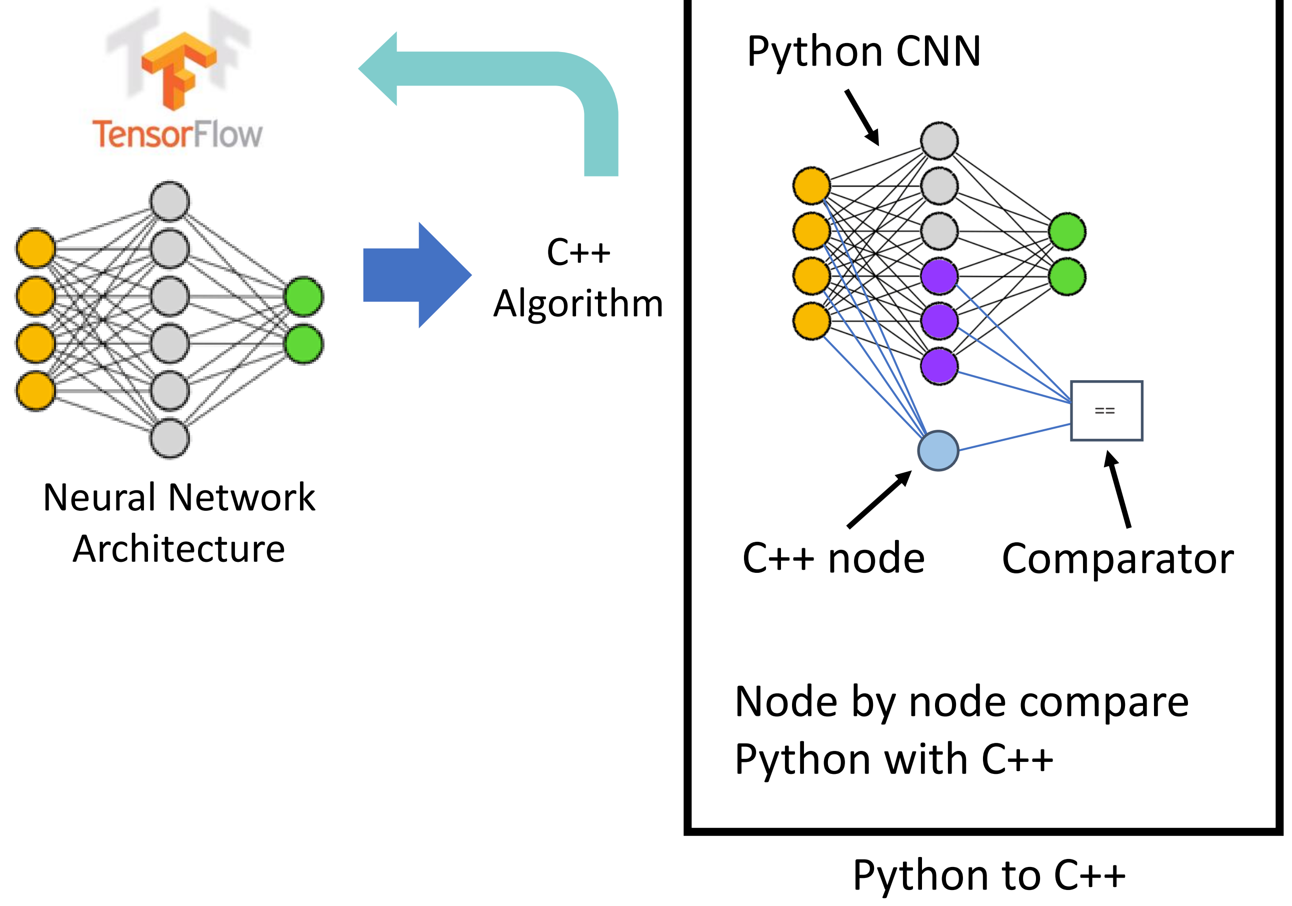
# Verification Capture Point



# Validation and Verification of MNIST System

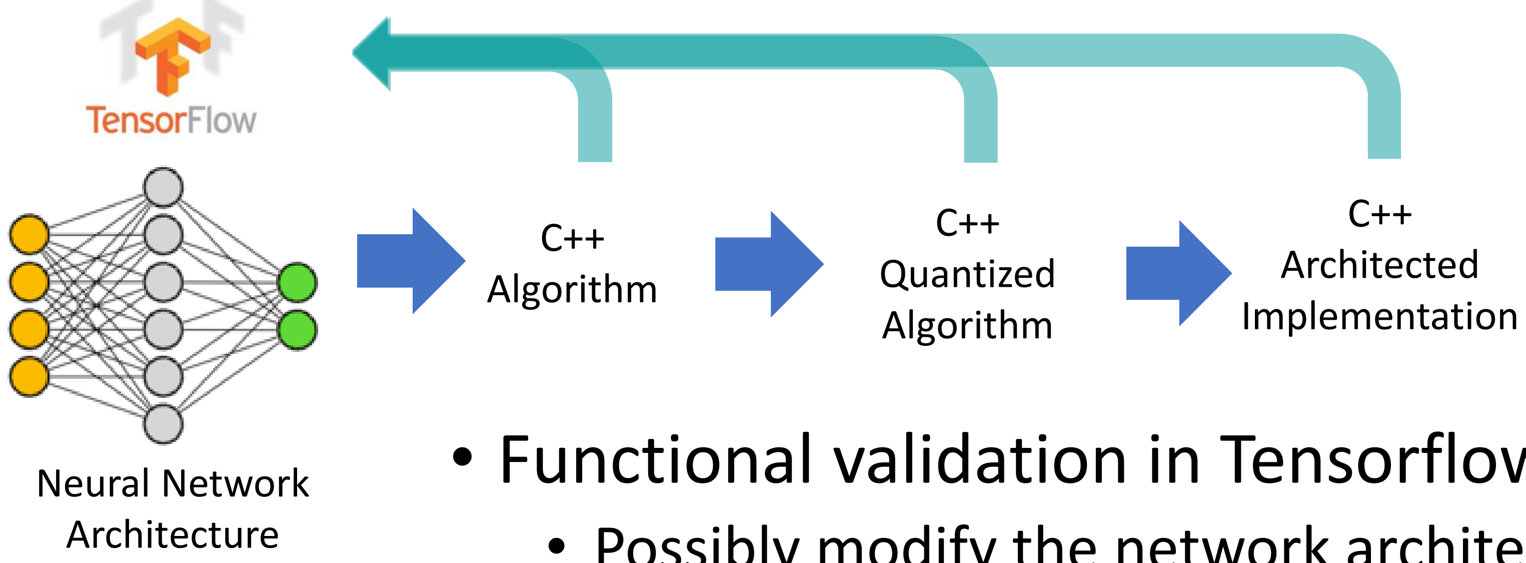


# Validating Python to C++ Translation Consistency



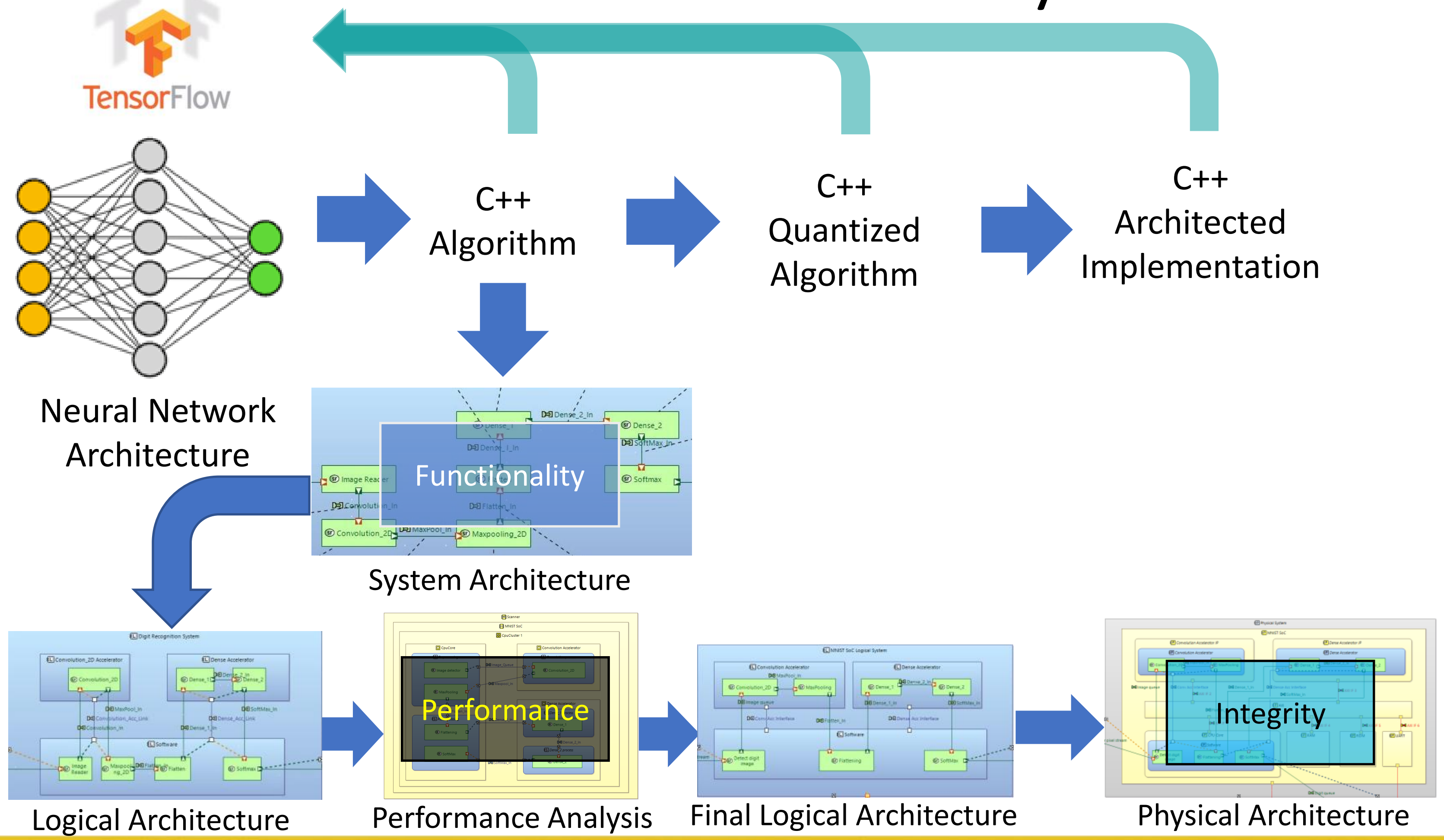
- Import a C++ node into Python and compare the outputs
- Start with one node
- Then one layer
- Then the whole network
- Should match Python results except for bottom 2 or 3 bits
- Variance results from order of operations

# Validation and Verification of C++ Code

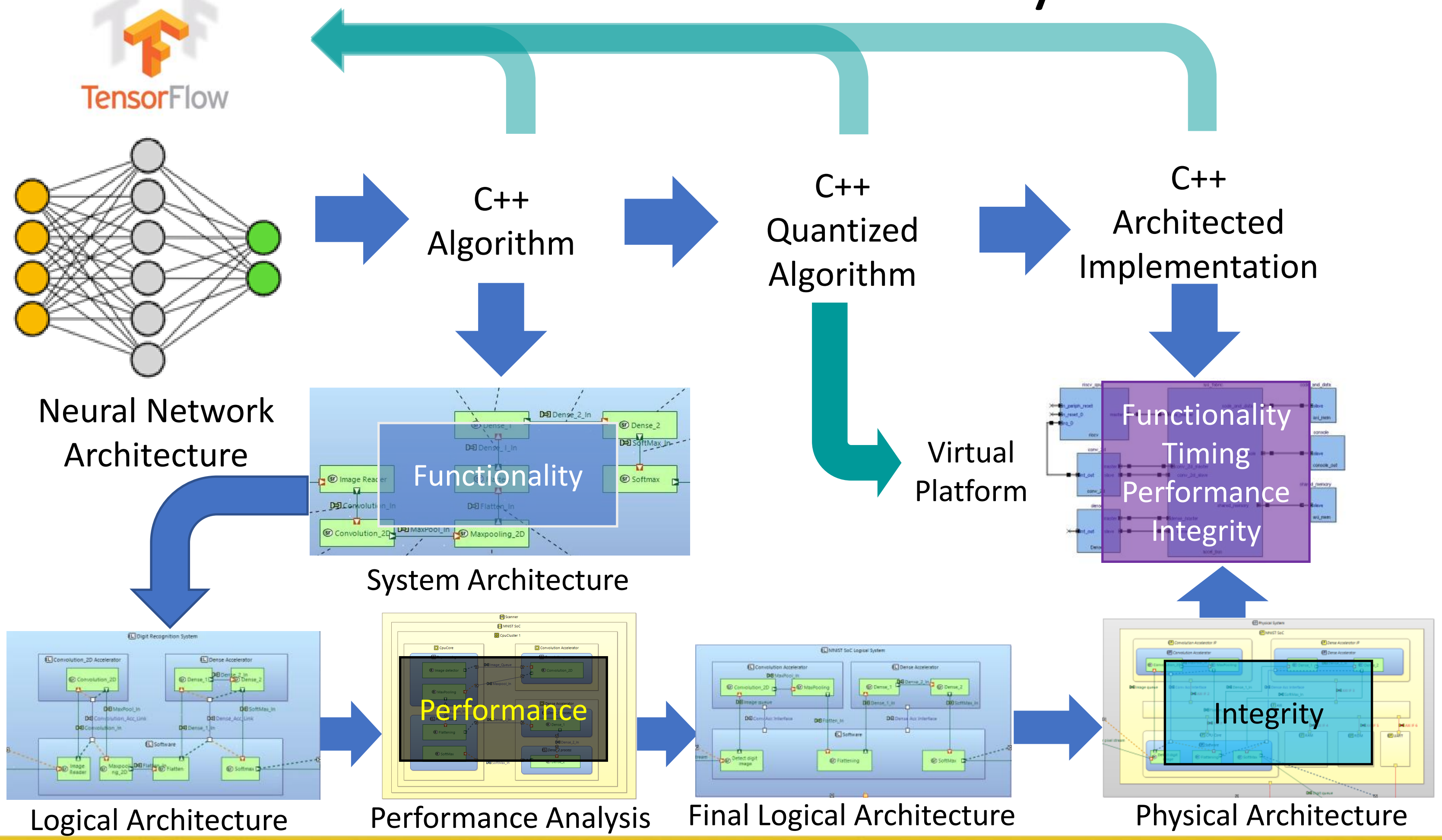


- Functional validation in Tensorflow
  - Possibly modify the network architecture for better PPA
- Verification of quantized and architected C++ code
  - Ensure that implementation meets requirements
  - In C++ domain with translated testbench
  - Coverage analysis needed to ensure test quality

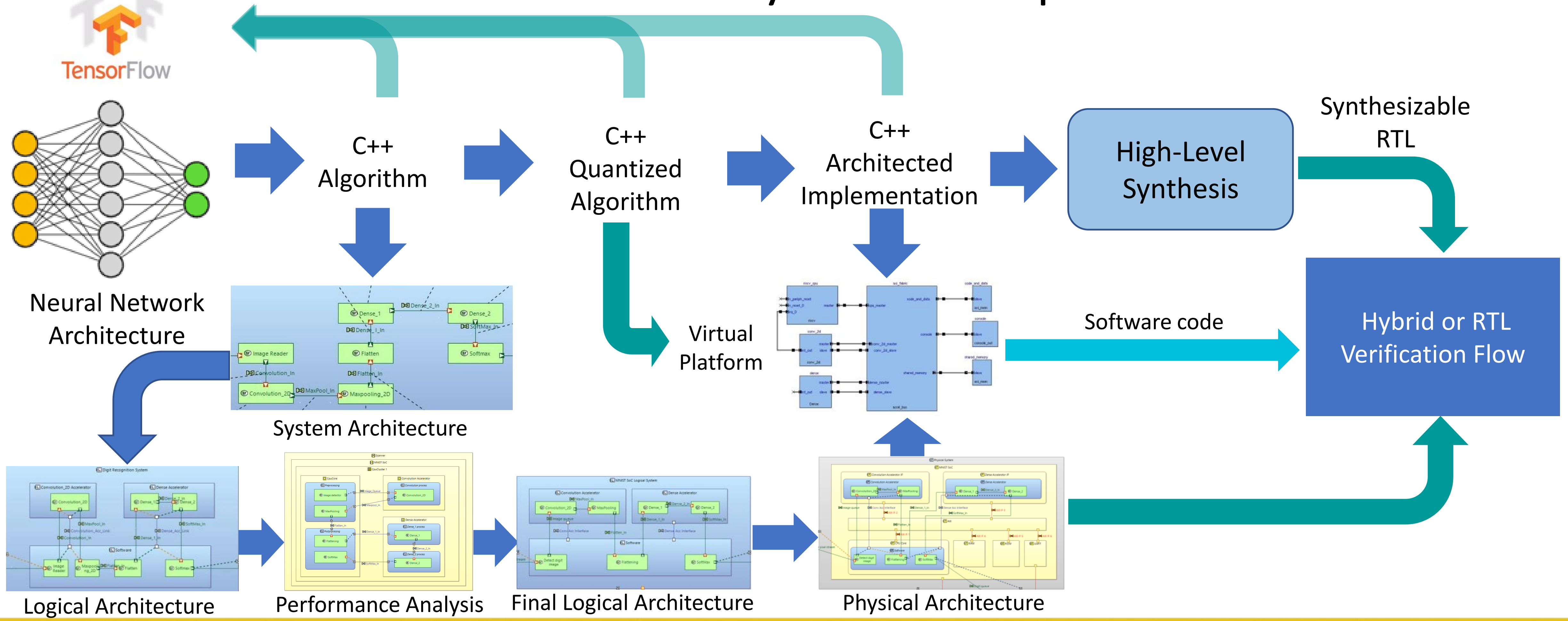
# Verification of MNIST System Architecture



# Verification of MNIST System Architecture

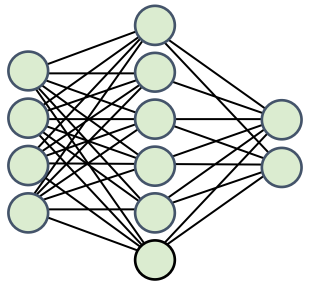


# Verification of MNIST System Implementation



# Verification – Before HLS

C++ Architected CNN



### Static Design Checks

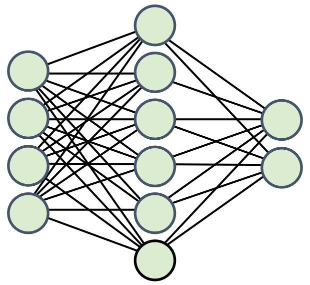
Static code analysis and synthesis checks.  Find coding errors and problem constructs

### Coverage Analysis

Determine completeness of test cases.  Statement, branch and expression coverage as well as covergroups, coverpoints, bins and crosses

# C++ to RTL consistency

C++ Architected CNN



**Formal**

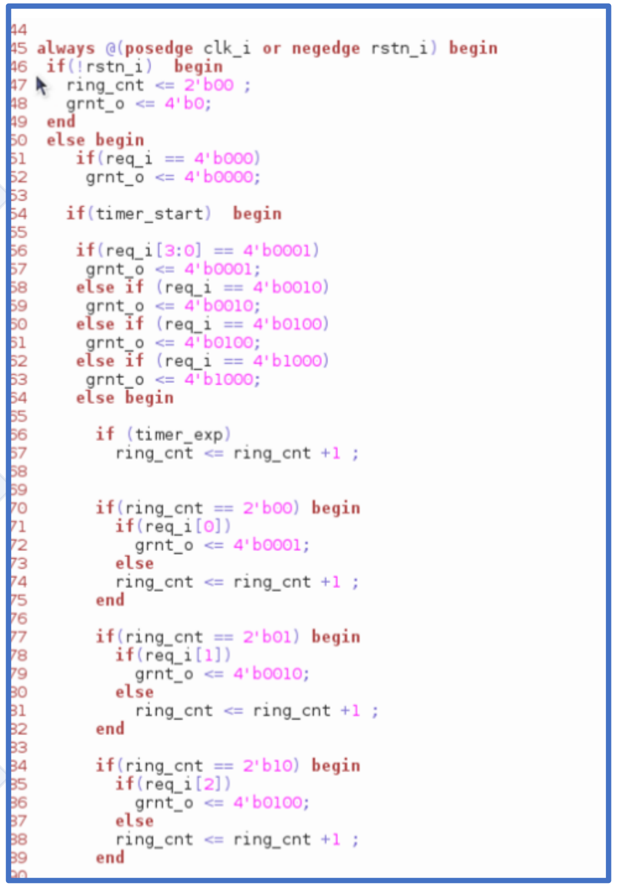
Using formal techniques, prove as much equivalency as possible

**UVM**

Architected C++ is used as a predictor for RTL verification

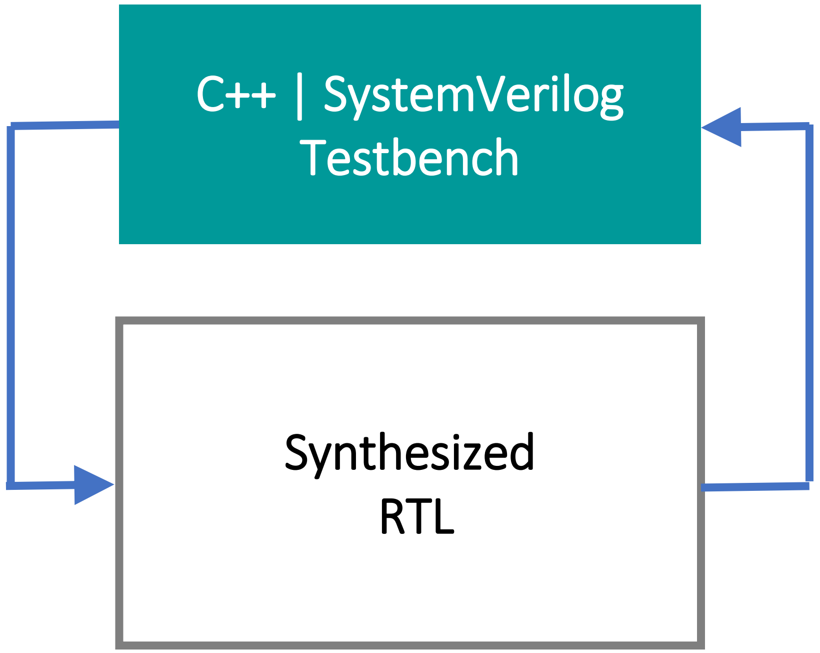
**RTL Coverage**

Determine remaining verification effectiveness through RTL coverage metrics



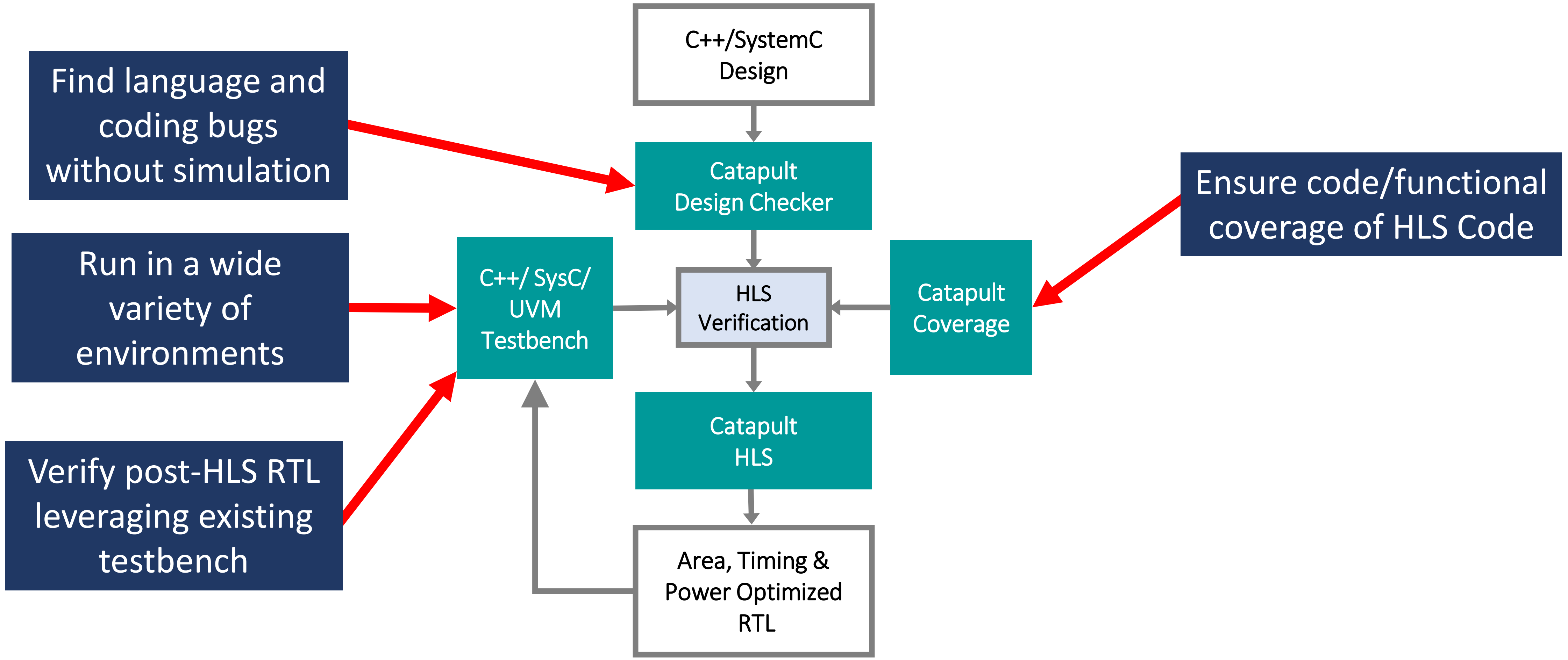
```
always @(posedge clk_i or negedge rstn_i) begin
 if(!rstn_i)  begin
   ring_cnt <= 2'b00 ;
   grnt_o <= 4'b0;
 end
 else begin
    if(req_i == 4'b000)
     grnt_o <= 4'b0000;

   if(timer_start)  begin

    if(req_i[3:0] == 4'b0001)
     grnt_o <= 4'b0001;
    else if (req_i == 4'b0010)
     grnt_o <= 4'b0010;
    else if (req_i == 4'b0100)
     grnt_o <= 4'b0100;
    else if (req_i == 4'b1000)
     grnt_o <= 4'b1000;
    else begin

      if (timer_exp)
        ring_cnt <= ring_cnt +1 ;


      if(ring_cnt == 2'b00) begin
        if(req_i[0])
          grnt_o <= 4'b0001;
        else
        ring_cnt <= ring_cnt +1 ;
      end

      if(ring_cnt == 2'b01) begin
        if(req_i[1])
          grnt_o <= 4'b0010;
        else
          ring_cnt <= ring_cnt +1 ;
      end

      if(ring_cnt == 2'b10) begin
        if(req_i[2])
          grnt_o <= 4'b0100;
        else
        ring_cnt <= ring_cnt +1 ;
      end
```

## Block Level Verification

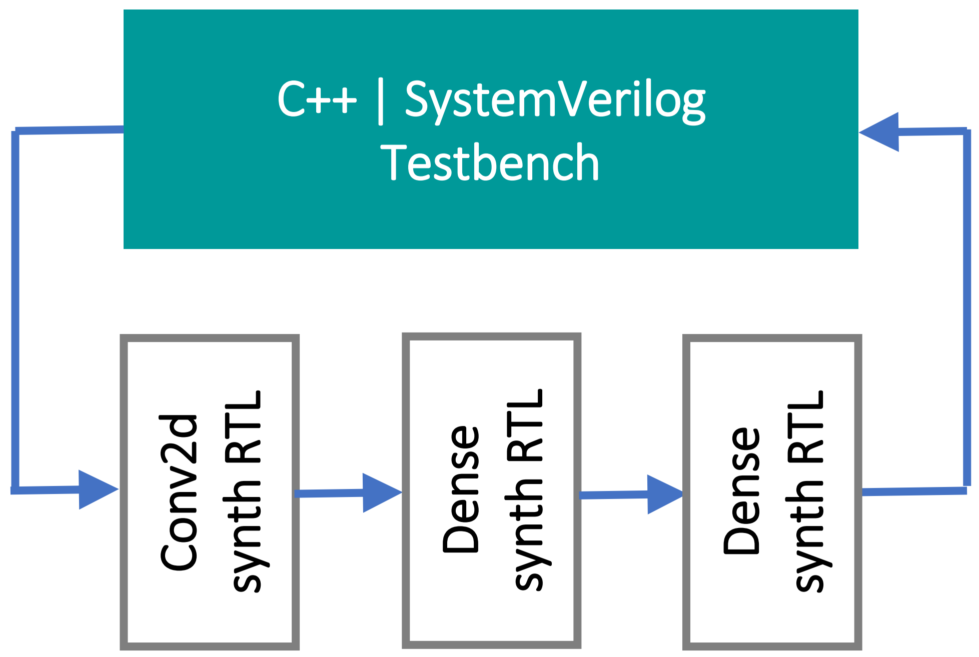


- Re-use C++ or employ new System Verilog testbench
- Prove correctness
  - Cover corner and exception conditions
- Repeat for each accelerator

# Block Level Verification - HLS Flow Tools

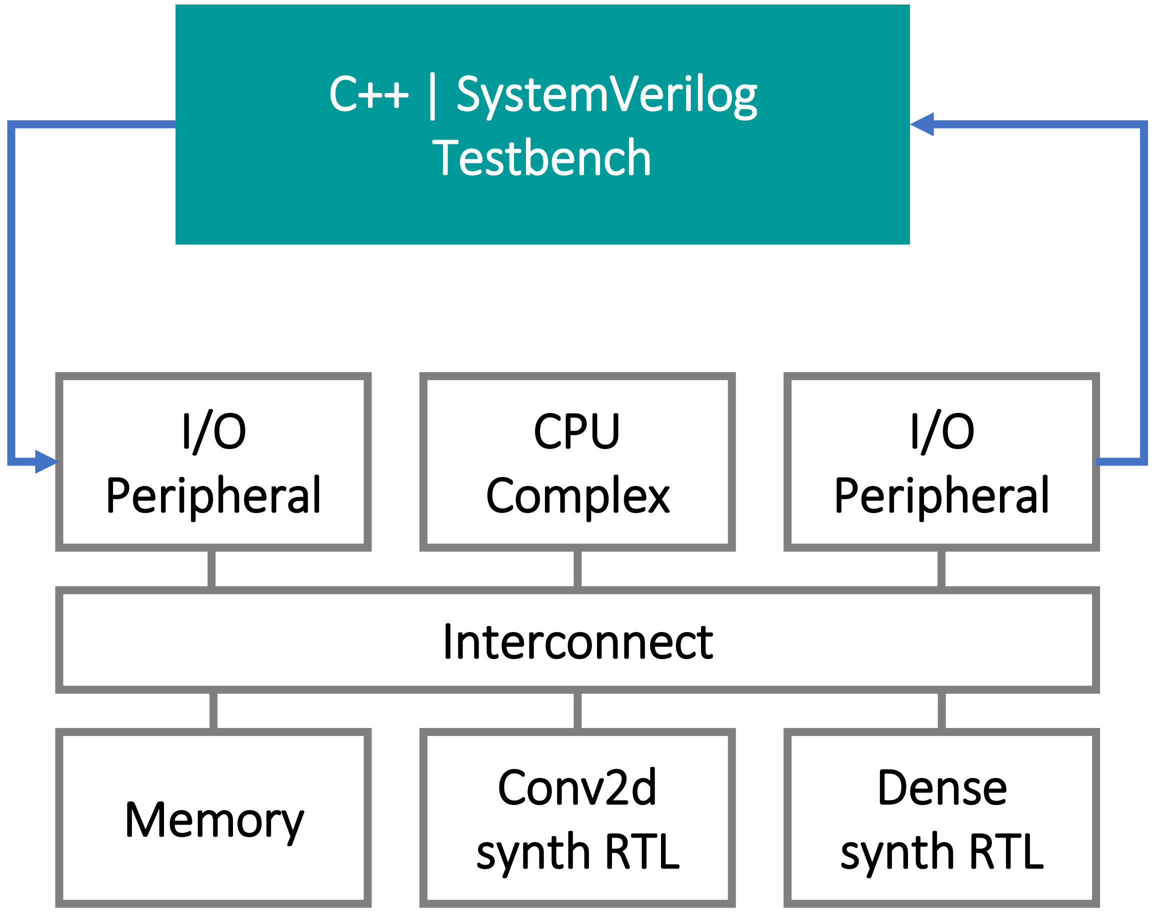

Find language and coding bugs without simulation
Run in a wide variety of environments
Verify post-HLS RTL leveraging existing testbench
C++/SystemC Design
Catapult Design Checker
C++/ SysC/ UVM Testbench
HLS Verification
Catapult Coverage
Catapult HLS
Area, Timing & Power Optimized RTL
Ensure code/functional coverage of HLS Code

## Sub-System Verification



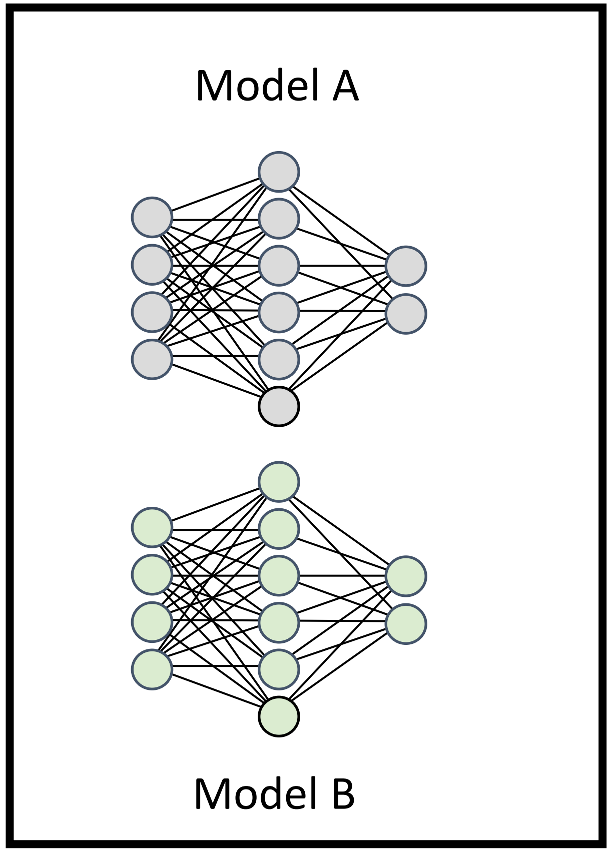
- Prove the correctness of a collection of accelerator
- Exercise larger functions
  - In this case inference
  - Low level coverage is not important here
- Run times could be impractical for logic simulation
  - May require acceleration (emulation | FPGA prototype)

## System Verification



- Includes processors, software, interconnect
  - Exercises HW and SW interfaces
- Execute typical and exceptional use cases
- Testbench drives I/O, clock, and reset
  - Software and processor orchestrate operation (as in final system)
- Likely to require FPGA prototype

## Debugging – When Things Go Wrong



- Log all intermediate values to memory or log file
  - This includes output from each layer
- Have scripts that can compare intermediate values from different model representations
  - This identifies the first point of divergence between models
  - Immediately find layer and node where problem resides
- Intermediate values from the Python can be recorded to a file for comparison

Questions?

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

# Model-Based Approach for Developing Optimal HW/SW Architectures for AI systems

Petri Solanti, petri.solanti@siemens.com

Russell Klein, russell.klein@siemens.com

accellera SYSTEMS INITIATIVE