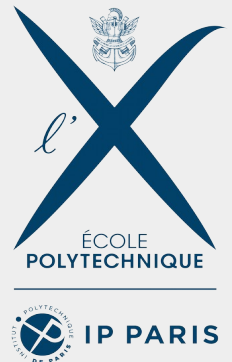


# Automated firmware generation and continuous testing for the CMS HGCAL trigger primitive generator

Journées des Métiers de l'Electronique  
de l'IN2P3 et de l'IRFU – 12/10/21

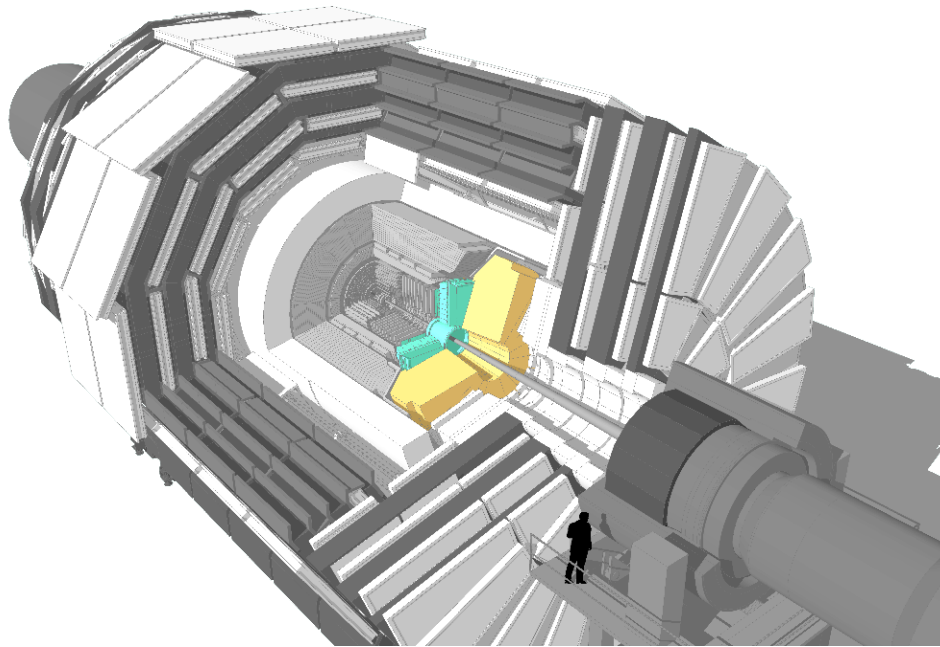
F. Beaujean  
T. Romanteau  
J.-B. Sauvan

LLR CNRS / École Polytechnique



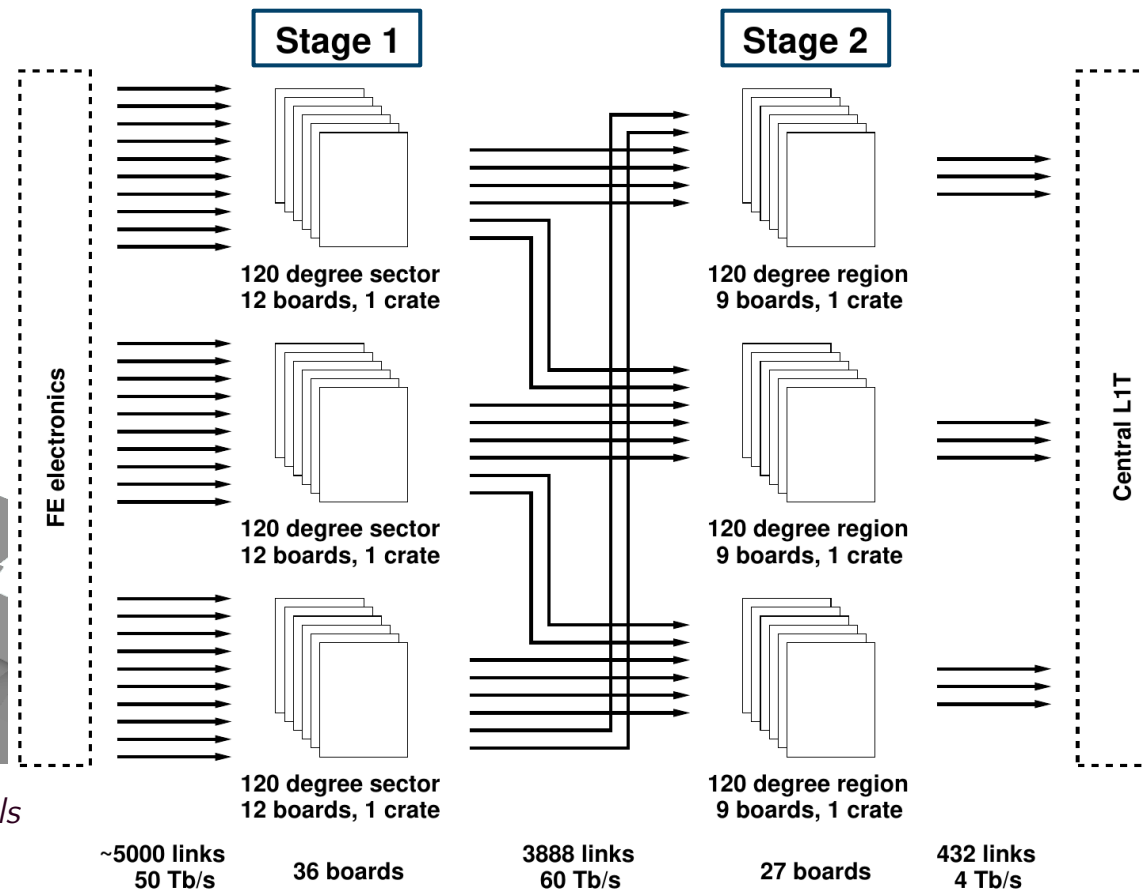
# The HGCal Trigger Primitives Generation (TPG)

## The future HGCal within CMS



See the [HGCal TDR](#) and the [L1T Phase 2 TDR](#) for more details

## HGCal TPG system architecture



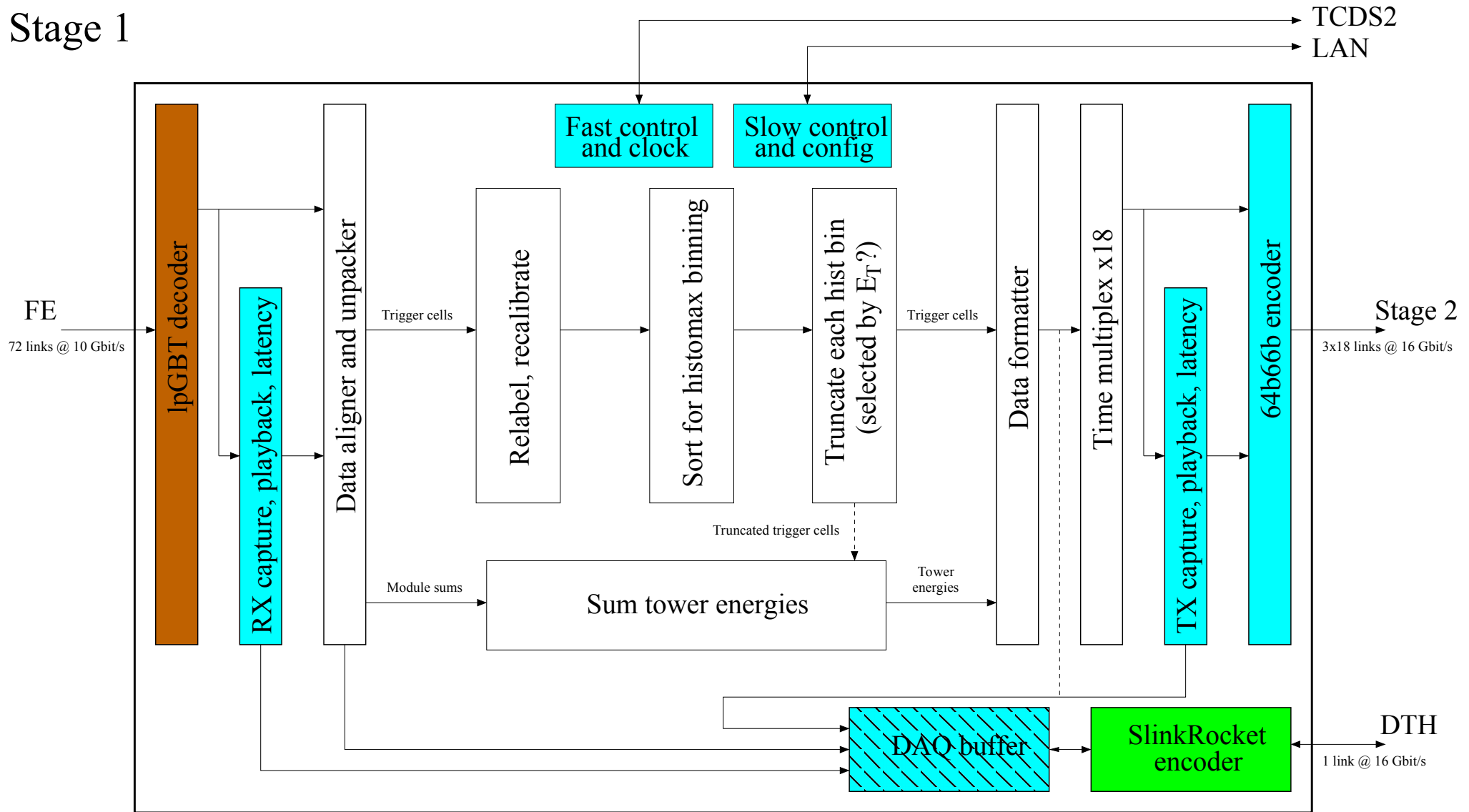
- HL-LHC: replacement of the **endcap calorimeters of CMS** with the HGCal
  - > **6 million channels** providing a 3D view + precise timing measurement
- TPG system: builds **3D Clusters** of energies from **Trigger Cells**<sup>(\*)</sup>
  - Off-detector **Two-Stage** system composed of FPGAs on ATCA boards (Serenity)

(\*) It builds also "Trigger Towers" (not covered here)

# Overview of the Stage 1 firmware

- Data alignment & pre-processing, time multiplexing
- Output format suitable for Stage 2 algorithms

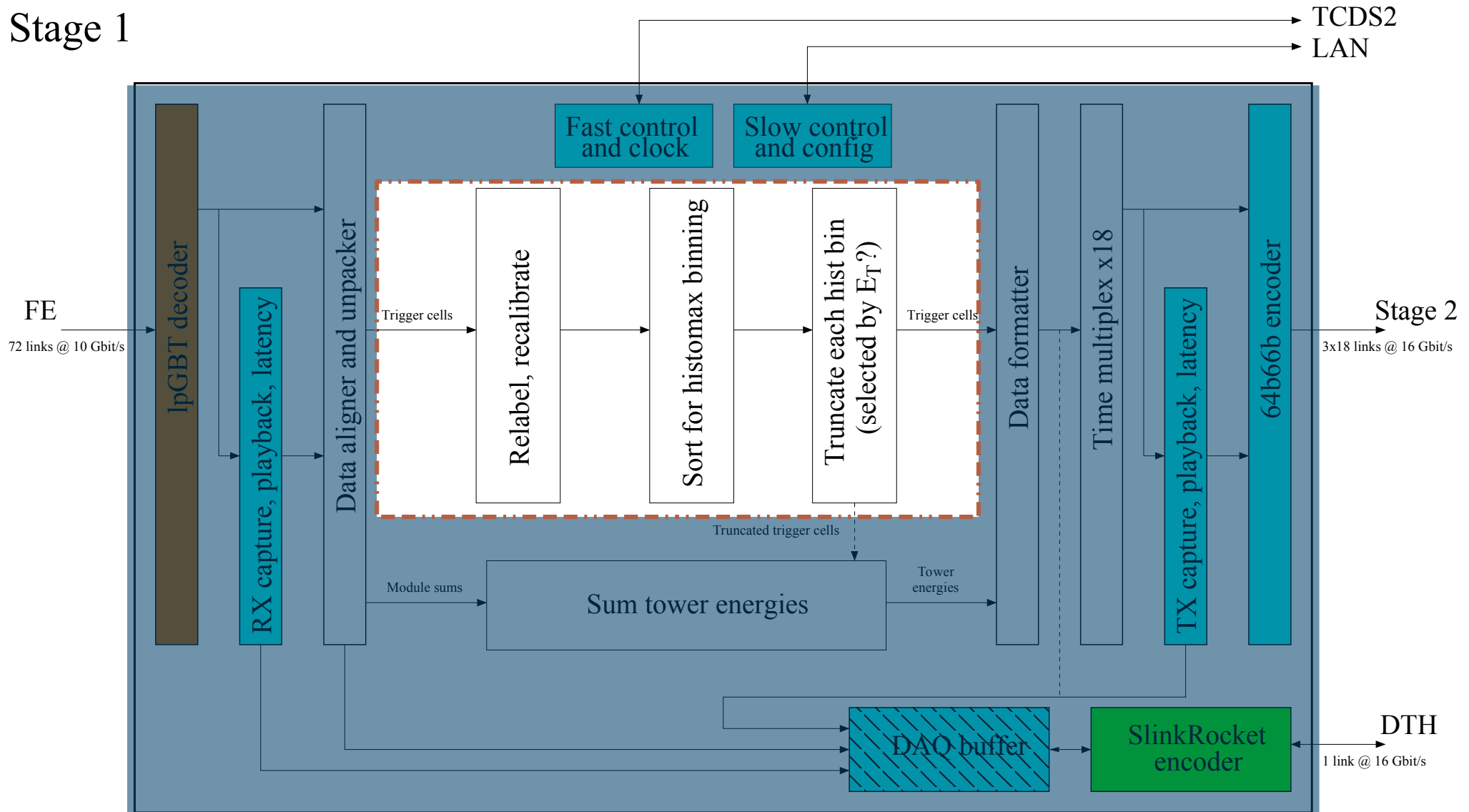
## Stage 1



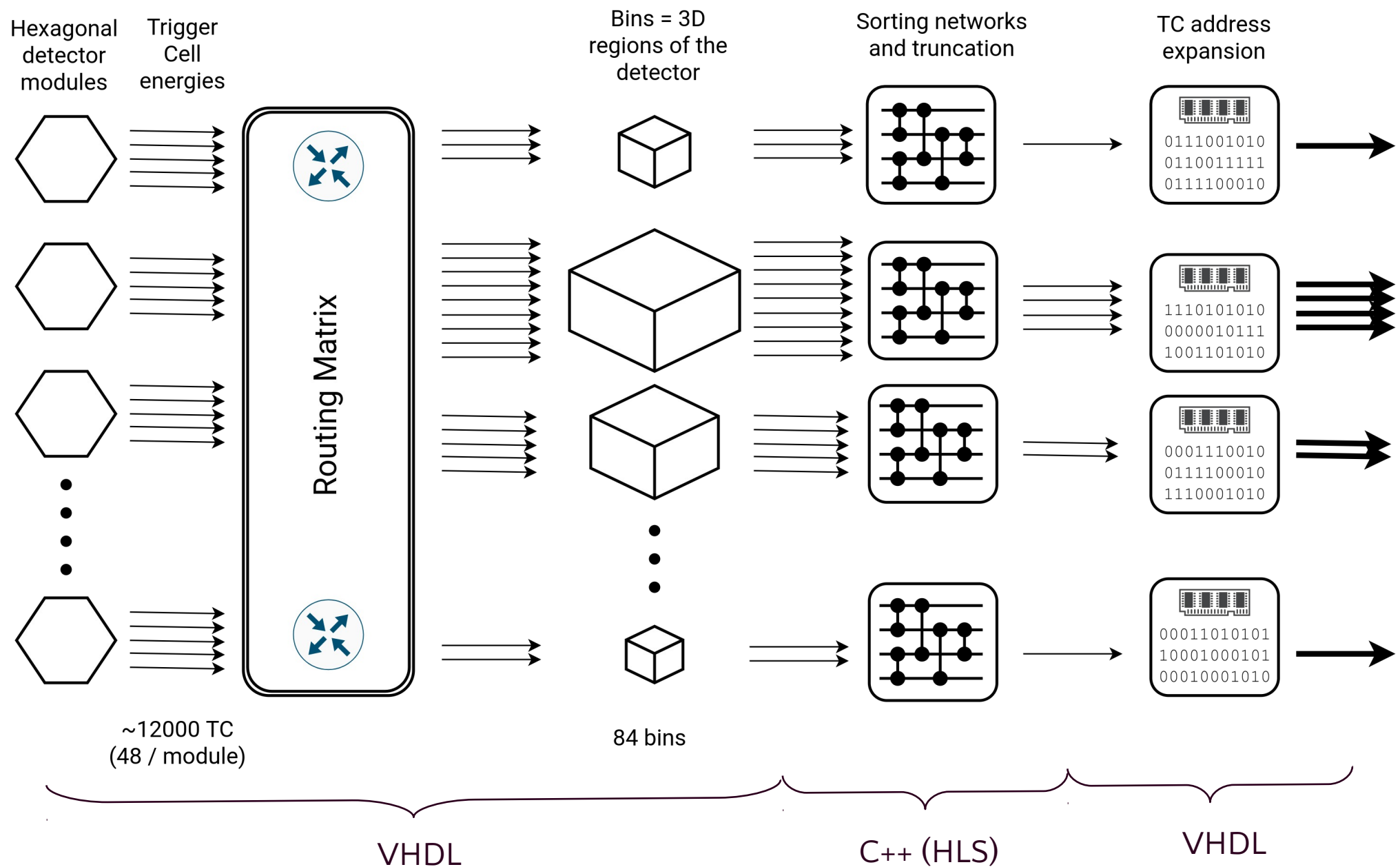
# Overview of the Stage 1 firmware

- Stage 1 blocks under consideration in this presentation:
  - Transformation, reordering and truncation of input Trigger Cells data

## Stage 1

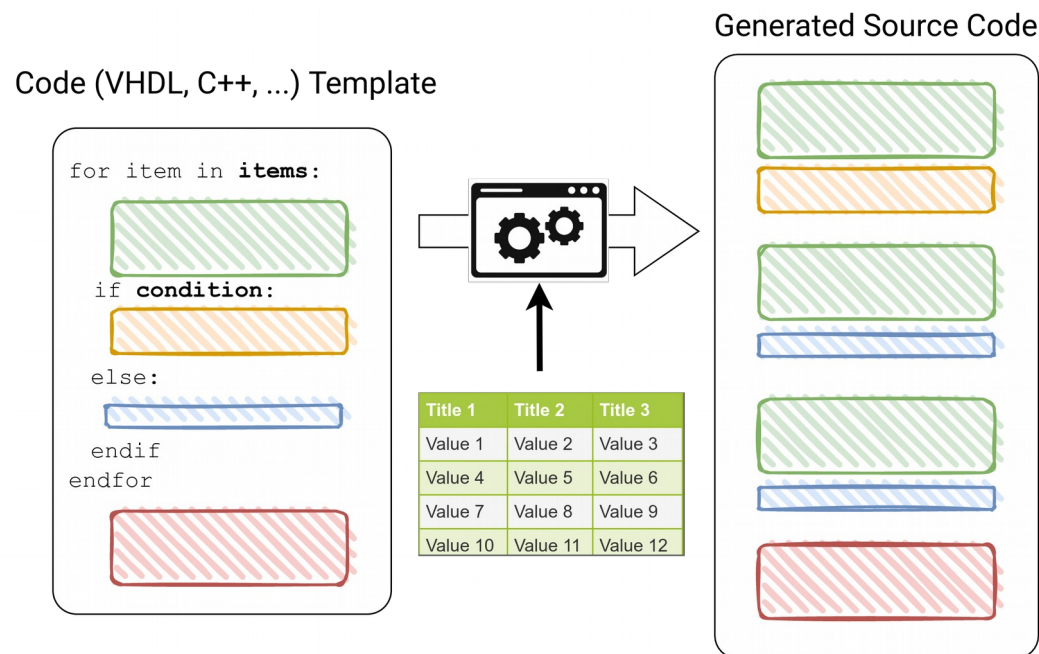


# Firmware blocks considered here



# Design constraints and choices

- Each **FPGA** in the Stage 1 sees a different **portion of the detector**
  - Different numbers of inputs, different routing, different sorting networks
- The **mapping** of FPGAs to portions of detectors is still being optimized / evolving
- Requires a highly **flexible** workflow
  - Generic code configured with configuration data (e.g. detector mappings)
- Two levels of abstraction
  - VHDL and C++ as **generic** as possible (e.g. using VHDL generics/generates, VHDL 2008)
  - Higher-level of abstraction also necessary → **code templates**



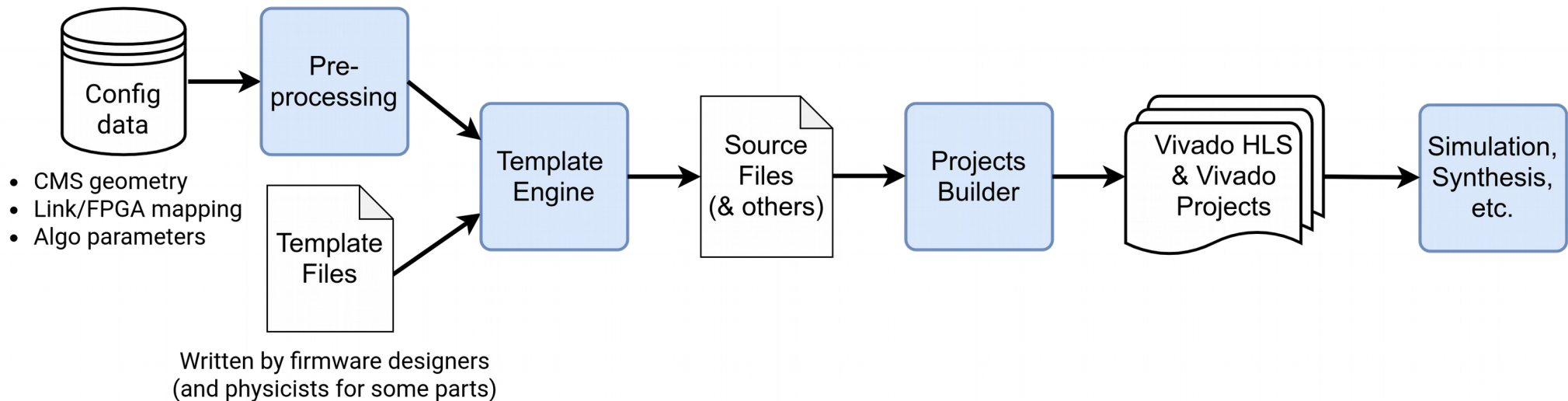
# Data-driven workflow

## ■ Two types of inputs

- **Configuration data** obtained from the CMS detector geometry/simulation and algorithms optimizing link connections/mappings
- **Code template** files (e.g. VHDL, C++ templates)

## ■ Automated steps based on these inputs

- **Generate** source files, test benches, etc. with a Template Engine
- **Build** Vivado HLS (C++ synthesis) and Vivado (RTL backend tools) projects
- **Simulate, synthesize, test** the generated designs
  - Check resource usage and latency
  - Future: comparison of output with software emulator





# Input configuration data

- Raw input configuration data are stored in various files and format
  - **Binary** (e.g. ROOT) and **text** (e.g. json) files
- **Pre-processing** step converts them into nested Python **dictionaries** and **lists**
  - Stored in **pickle** files

## Excerpt of pre-processed configuration data

```
{ 'bins': [ { 'mods': [ { 'hash': 3361, 'ntc': 5, 'tcs': [10, 36, 41, 27, 28]},  
                        { 'hash': 7456, 'ntc': 2, 'tcs': [29, 43]}],  
            'nmod': 2,  
            'ntcin': 7,  
            'ntcout': 1,  
            'phi': 0,  
            'pipenumb': 5,  
            'roz': 0},  
  { 'mods': [ { 'hash': 3361,  
                'ntc': 13,  
                'tcs': [ 8,  
                        11,  
                        13,  
                        14,  
                        38,  
                        15,  
                        37,
```



# Input templates

- Template language: **Jinja**
  - Python library: <https://palletsprojects.com/p/jinja/>
- Defines **statements** and **functions** based on the pre-processed configuration data
- Can define complex functions, e.g. through **Python functions** called within templates

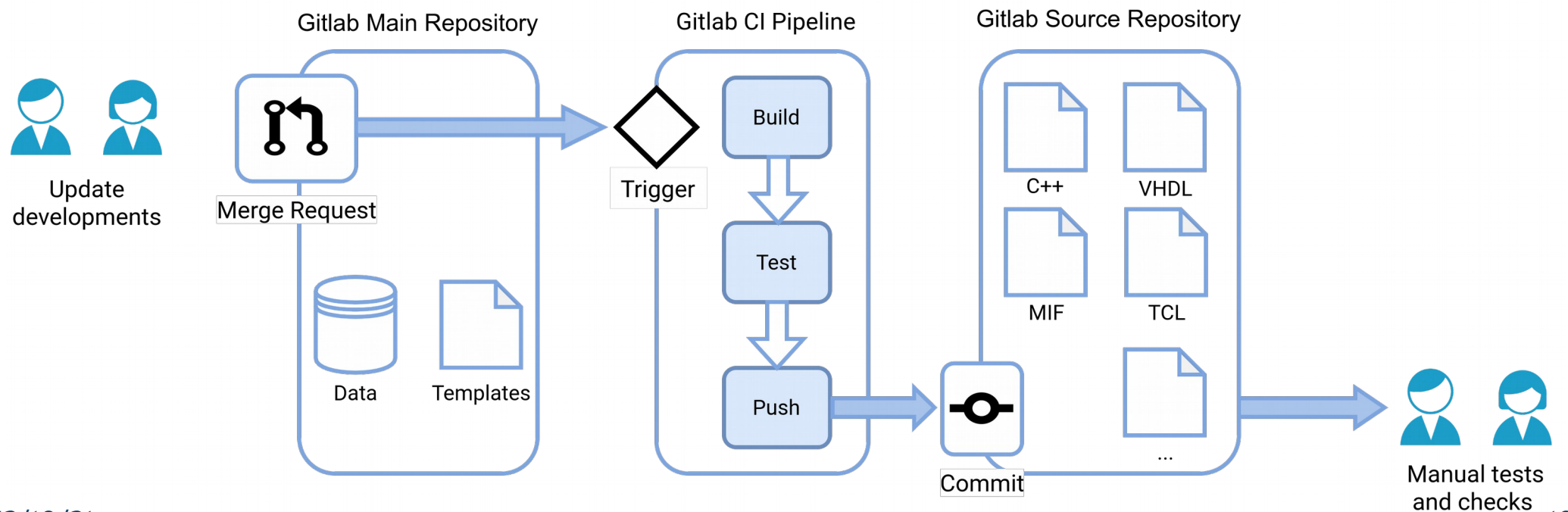
VHDL template excerpt

```
{% for bin in bins %}
  {%- for mod in bin.mods %}
    {%- if mod.tcs|length >1 %}
constant Phi{{ bin.phi }}Bin{{ bin.roz }}Mod{{ mod.hash }}_TCA :
  t_Module_TrigCellAdr(0 to Phi{{ bin.phi }}Bin{{ bin.roz }}Mod{{ mod.hash }}_TCC-1)
  := ({{ mod.tcs | join(',') }});
    {%- else %}
constant Phi{{ bin.phi }}Bin{{ bin.roz }}Mod{{ mod.hash }}_TCA : integer := {{ mod.tcs[0] }};
    {%- endif %}
  {%- endfor %}
{% endfor %}
```


























# Git workflow with Gitlab CI/CD

- Automation with Gitlab CI/CD (**Continuous Integration** / Deployment)
- The main items of this automation are
  - **Two Git repositories**
    - (1) for inputs: configuration data and templates
    - (2) for generated sources, test benches, etc.
  - Gitlab CI “**Pipeline**”: defines the steps of the workflow
    - Triggered e.g. by “Merge Requests” in repository (1)
    - Produces and commits sources in repository (2)



# Gitlab CI pipelines

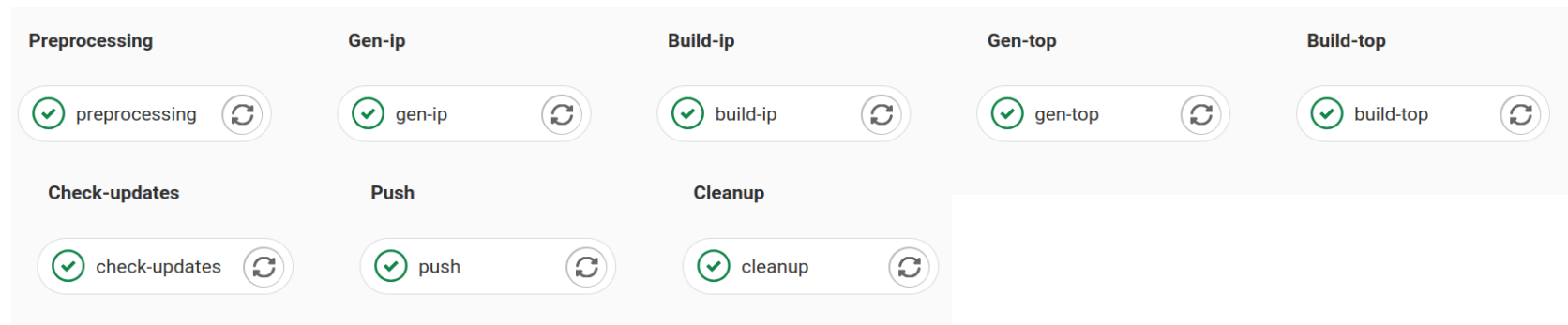
## Example of pipelines displayed in Gitlab UI

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
 passed	#3799 latest		 master → e5ac8e4a  Add reading of sorter reso...	       	🕒 00:34:52 📅 3 days ago
 passed	#3798 latest detached		 119 → bf803bd5  Add reading of sorter reso...	      	🕒 00:34:30 📅 3 days ago

### ■ Gitlab CI Pipelines triggered here with

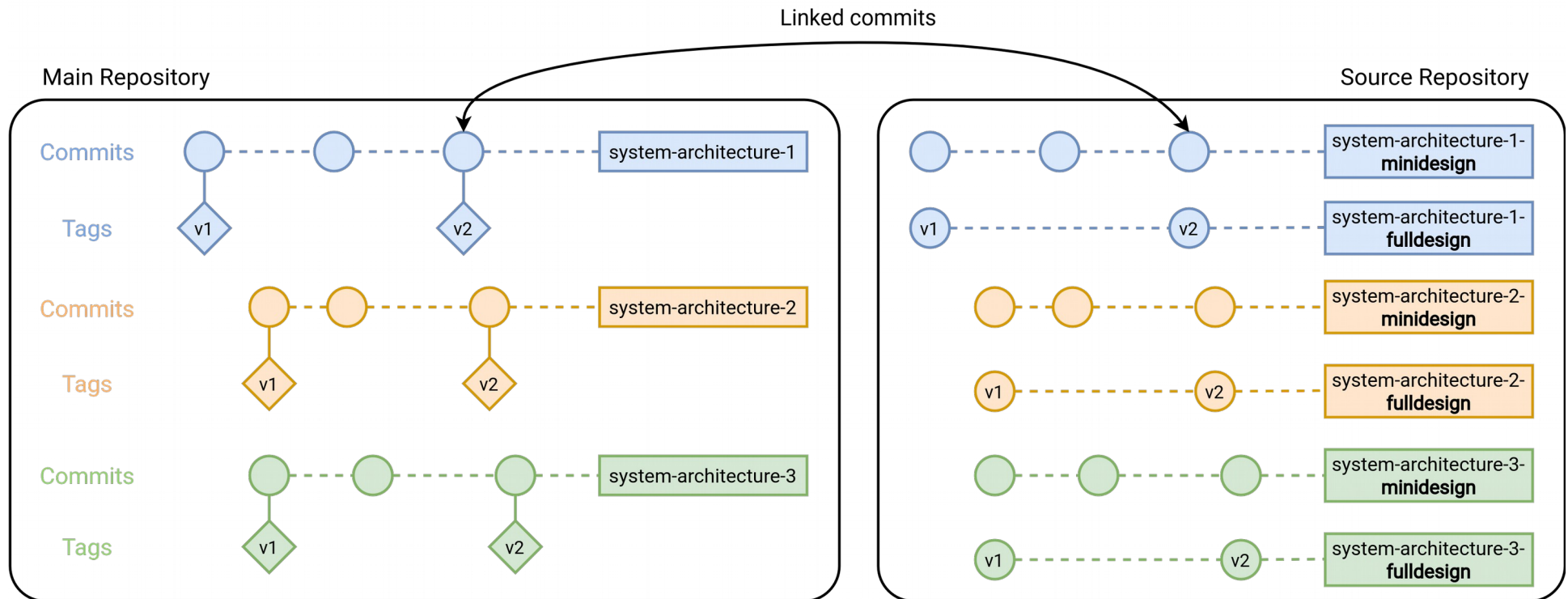
- New Merge Request update → Only **Build** and **Test**
- Merge Request actually merged → **Build**, **Test** and **Push** generated sources

## Example of pipeline steps ran one after the other



# Multi-design with multi-branching

- The details of the TPG system architecture are not yet completely frozen
  - Changes in **FPGA model**, changes in number of **links per FPGA**, etc.
- Need to handle **multiple designs**, for multiple “architectures” to be studied
- Will be done with **multiple branches** in the two Git repositories
  - Same templates but different configuration
  - Branches dedicated to quick tests, with **reduced designs** (“minidesigns”)
  - Branches dedicated to “production” and large tests, with **full designs**



# Conclusions

- HGICAL TPG Stage 1 firmware heavily depends on detector geometry and module → FPGA connections, which are still evolving
- Requirement of fully **generic code**, easily **configurable**
  - VHDL generics/generates, simpler code with 2008-specific syntax
  - High-Level Synthesis from C++
  - Higher-level template language (Jinja) for complex generation rules
- Automatized workflow with Gitlab **Continuous Integration**
  - Quick generation and testing of reduced designs (“minidesigns”)
  - Generation and testing of full designs when creating new releases
  - Resource usage and latency checks. Comparison with software emulator planned in the future.
- Management of **multiple designs** for different system variations
  - Parallel branching
  - Same code templates – Different configurations