DIGITAL DESIGN & CI/CD

ILJA BEKMAN

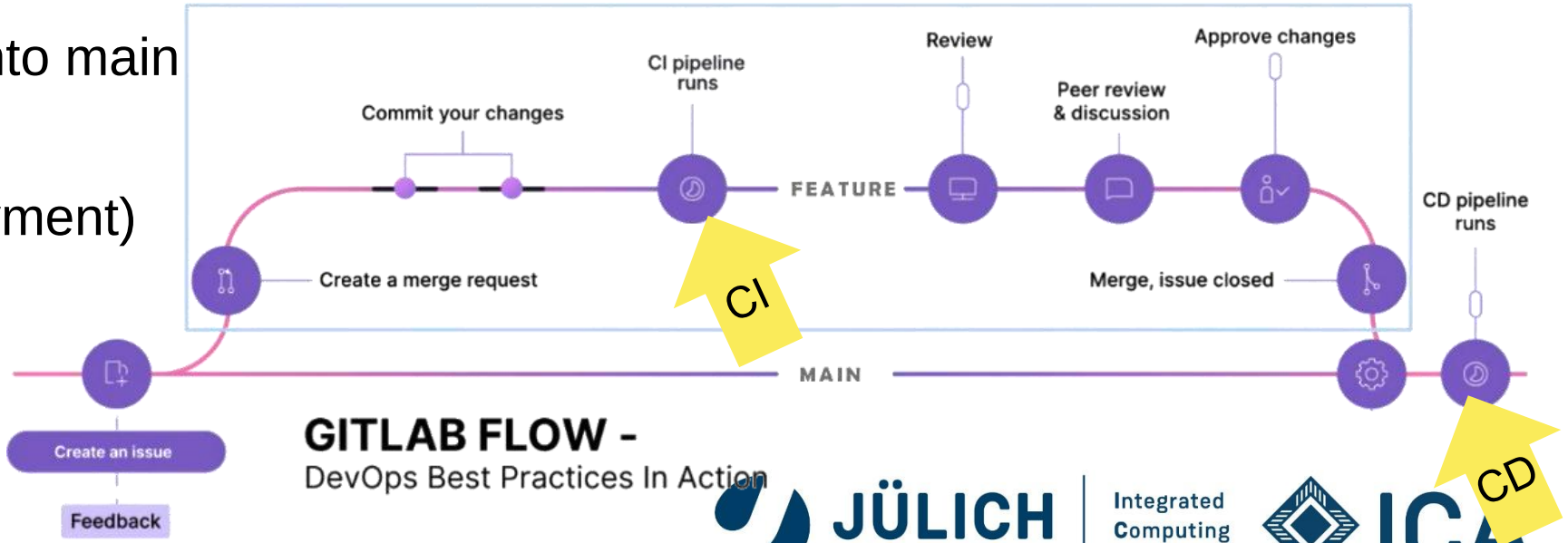
CMF IHL KICK-OFF, SIEGEN, 17.07.2026

VERSION CONTROL - TO GIT OR NOT TO GIT

- **git** is a communication tool hg, jj,…
  - with coworkers / your future and past self
- **CI** (continuous integration)
  for **automatic** code **execution / interpretation**
  - less mental load, steady test quality
- Merging → “does adding this feature break things?”
  - Merge feature branch into main
    = **Release**

  → **CD** (continuous deployment)

repository	atomic development unit
commit	one change
issue / branch	feature
group / subgroup	= directory of repos
git-submodule subtree / git-repo	nested modules, libraries, dependencies



GITLAB FLOW -
DevOps Best Practices In Action

VERSIONING AS DOCUMENTATION – CI/CD TO ACTIVATE

- Commit messages, authors, timeline, **issues** = part of **meta-documentation**
- **test benches** as self-contained unit tests (“*show, don’t tell*”), AXI-L Cheby, (Doxygen)

  => Active documentation (vs. static = outdated) based on source code in repo

- GitLab **CI/CD** (or GitHub Actions) → `.gitlab-ci.yml`

https://docs.gitlab.com/ci/yaml/

`pipeline`

Passed
Warning
Failed

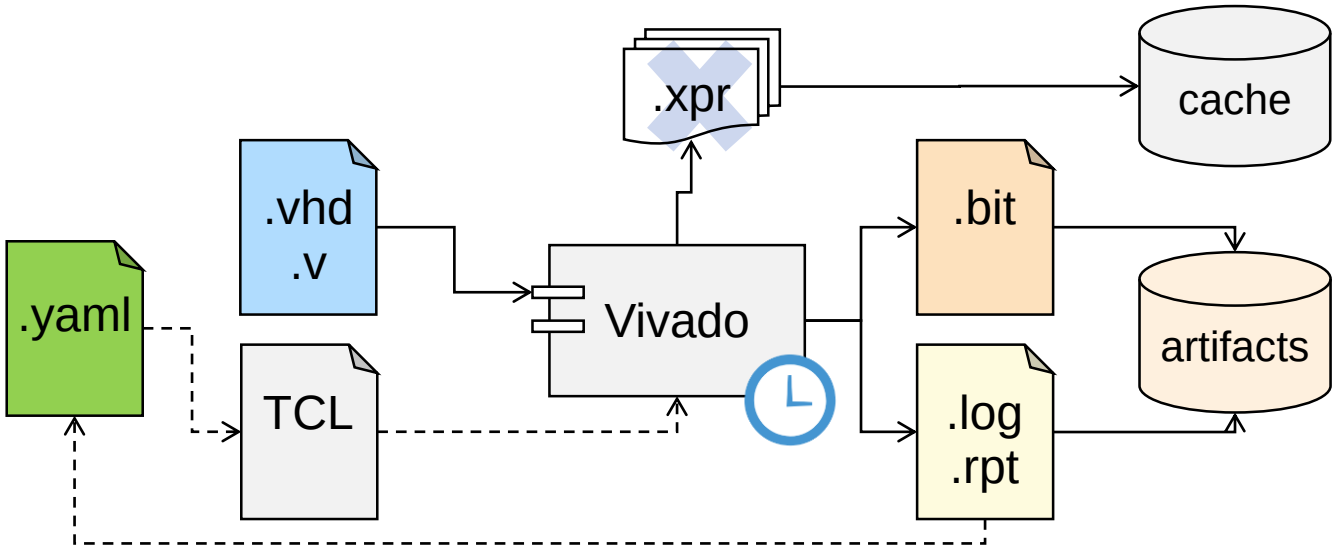
```
stages:
  - test
  - simulate
  - build
  - deploy
```

`(trigger other repos)`

… and for Firmware?

VIVADO – WORKING AGAINST THE TOOL

- FPGA design:
  - VHDL/Verilog source code
  - GUI (+.bd,.xci) → **TCL** scripts
    - TCL by Vivado is not nice…
- project dir (i.e. .xpr) is stateful



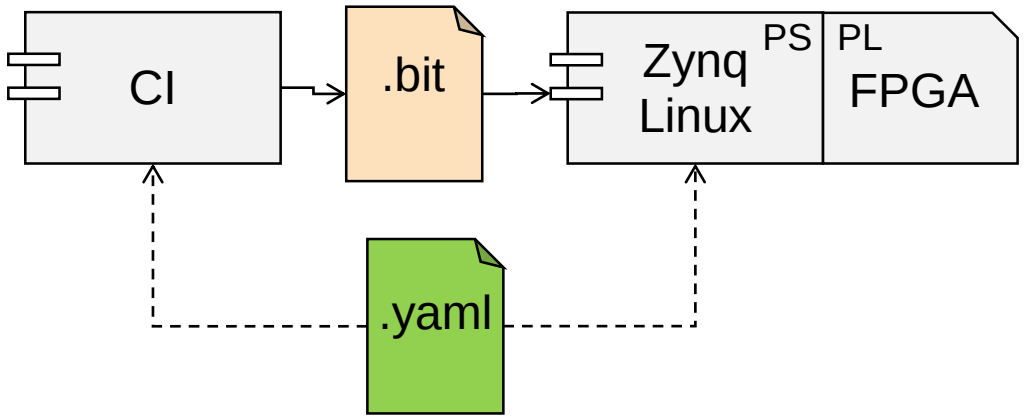
- unit tests i.e. test-benches <= **warning/error** message parsing go here
- test-bench is faster than implementation

➔ Delivery

- CI – **continuous** integration
  - YAML + bash (test of exit codes)
  - **log**, .rpt = text pattern matching
  - artifacts = **files**, (especially **.bit**)

REMOTE FIRMWARE DELIVERY – AKA LAPAROSCOPY

- Continuous Delivery:
  - (JTAG) → PCAP via Linux
  - In practice: SCP + SSH + bash
- Executing remote command (from .yaml)
  - yaml on gitlab
  - bash on runner
  - bash on zynq
  - flashing of FPGA



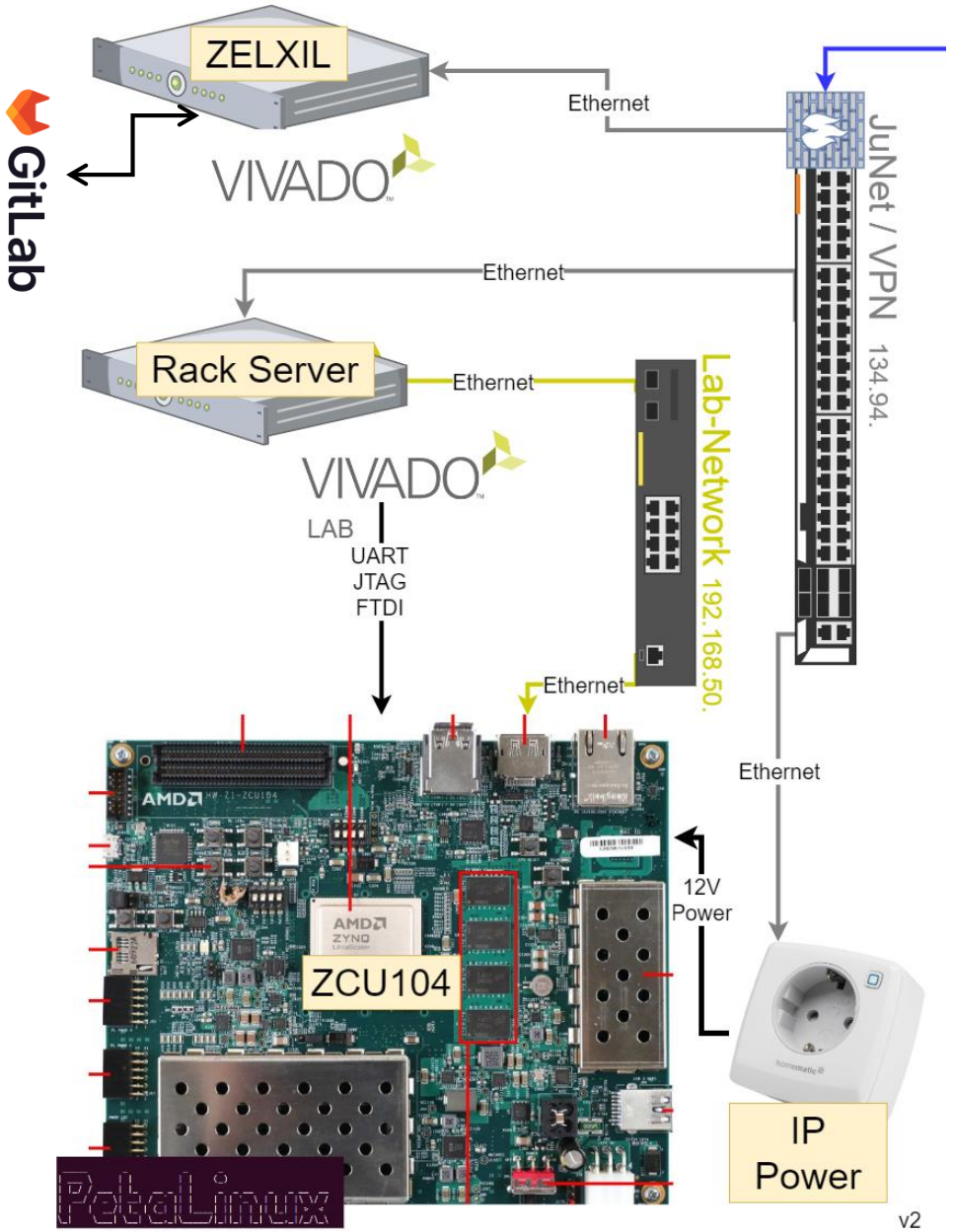

```
deploy:
  script:
    - scp ./firmware.bit deployer@zynq:/firmware/
    - ssh deployerb@zynq "eval '$(cat ./flash_fw.sh)'"
```

- full .bit-file generation >15min? not for every commit
  - sim CI for **feature branches**,
  - full CI for **staging branches**,
  - CD for **deployment branch**
    - e.g. after approval process of a merge request

HARDWARE

- **Development Server**
  - remote connection
  - Vivado (+ Sim)
  - shell **gitlab-runner**
- Local LAN + **Delivery Server**
  - physical connection
  - debug/fallback **JTAG + UART**
- **Zynq Board**
  - (Peta/EDF)-Linux
  - Golden Image + **IP**/PoE **power-cycle**

  ^ <u>fast recovery from mistakes</u>



v2

JÜLICH Forschungszentrum | Integrated Computing Architectures ICA

FINAL REMARKS

- Jupyter on Zynq (`PYNQ`)
  - python on Zynq Linux (not for crunching), access via browser
  - users can load prepared .bit firmware
- test-benches in python - see **`cocotb`**
- Vivado-CI can trigger Peta/EDF-Linux build

- CI/CD fits Digital Design
- more unit-tests!
- **git and CI provide safety for devs and reproducibility for maintainers**

Thank you for your attention.

Questions?

doi: 10.1109/LES.2023.3314148

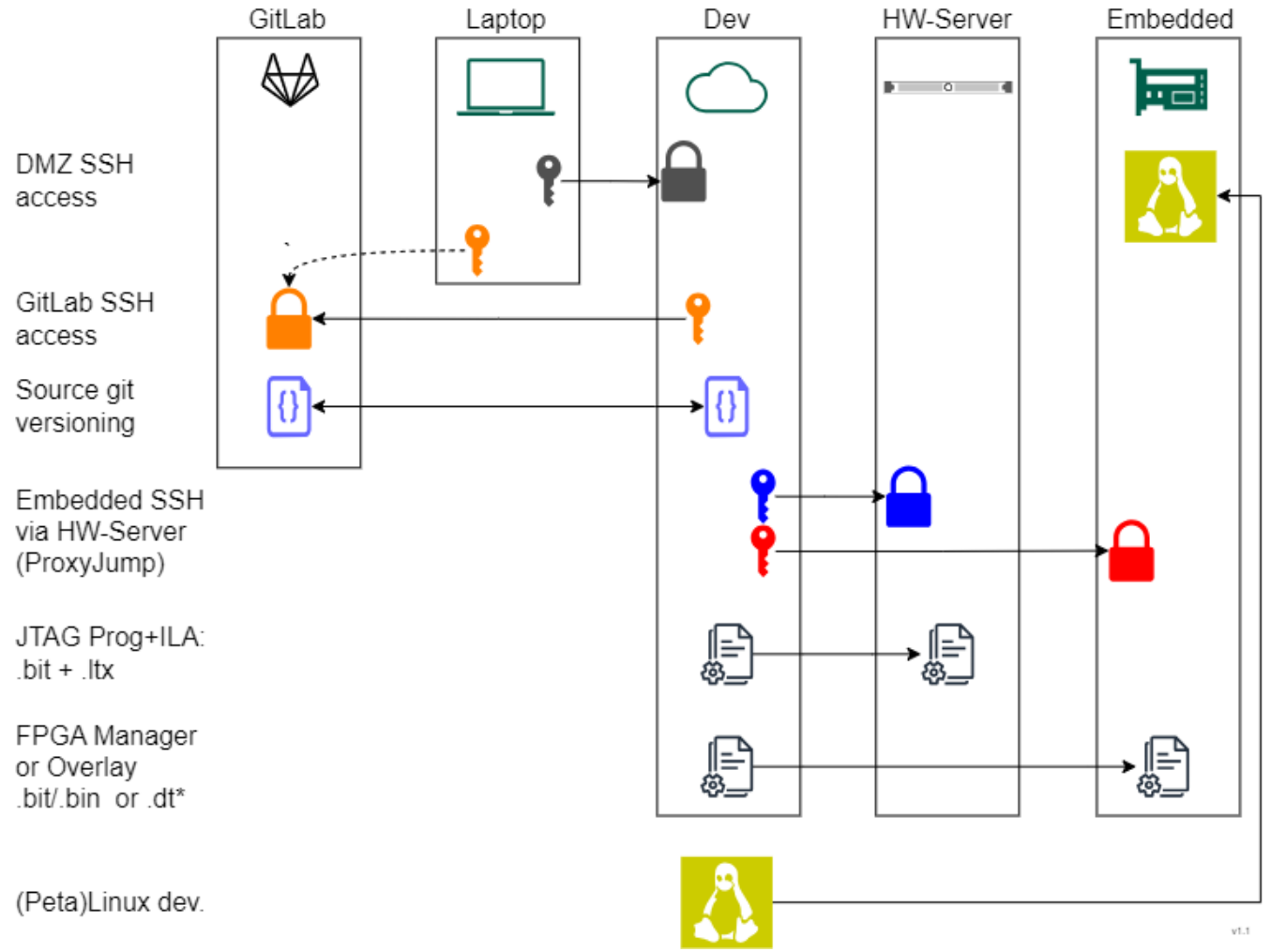
BACKUP

CI/CD EXAMPLES

- "make" of C++ / "maven" of Java / ... source to produce and publish an executable
- publish a static website with a doxygen / manual / blog (i.e. "Pages") as <repo-name>.<domain>.io
- create a .pdf from a MarkDown or LaTeX source
- run a python script to analyse data and create a plot.png
- try out a Docker image
- execute a .tcl script to create a Vivado project, check for timing closure, generate a .bit-file
- build a PetaLinux project, generate an SD-card image

- Cache for Job-Job files
- Trigger for Proj-Proj communication

SSH PRIVATE-PUBLIC KEY PAIRS



```yaml
build_vivado_job:
  stage: build
  needs: ["test_env_job"]
  tags:
    - shell
    - vivado
  cache:
    policy: push
  timeout: 100 minutes
  before_script:
    - source /opt/Xilinx/Vivado/2023.2/settings64.sh
    - export $(cat /etc/environment | xargs)
  script:
    - NOW=$CI_COMMIT_SHORT_SHA
    - ./build.sh $NOW
    - ./report.sh > report.md
    # require no errors in logs for success
    - (! grep -r 'ERROR:' */*.runs/*/*.log )
    # show critical warnings in logs
    - grep -r 'CRITICAL WARNING:' */*.runs/*/*.log
    # require .bit file for success
    - test -e */*.runs/*/*.bit
```

or timestamp

```yaml
  artifacts:
    when: always
    expire_in: 1 day
    paths:
      - "vivado*.log"
      - "vivado*.jou"
      - "*/*.runs/*/*.rpt"
      - "*/*.runs/*/*.log"
      - "*/*.runs/*/*.bit"
      - "*/*.xsa"
      - "*/*.pdf"
      - "*.bi?"
      - "*.xsa"
      - "*.lts"
      - "*.csv"
      - "report.md"
      - "version"
      - "*/*.xpr"
      - “proj*.viv/"
```

Vivado proj (optional)

BUILD.SH – SIMPLE EXAMPLE

```
#!/bin/bash

### timestamp build or use provided argument to make
unique directory
NOW=$(date '+%Y%m%d_%H%M%S%z')
NOW=${1:-$NOW}

### Project Name
PROJ_NAME=BAB_TE0835
PROJ=${PROJ_NAME}_${NOW}

  VIVADO_BIN="vivado"
  VIVADO_BIN+=" -journal vivado_${NOW}.jou"
  VIVADO_BIN+=" -log vivado_${NOW}.log"
  VIVADO_BIN+=" -verbose"
  VIVADO_BIN+=" -tempDir /run/user/$(id -u)"
  # -source build.tcl

# run with gui locally, batch mode for CI
if [ -n "$CI_PROJECT_NAME" -o -z "$DISPLAY" ]; then
    VIVADO_BIN+=" -mode batch " ;
  else
    VIVADO_BIN+=" -mode gui " ;
fi
```

```
#Dependencies:
if [ -e dependencies.sh ]; then
  . ./dependencies.sh
fi

# relies on sourced binary and project tcl-Script
$VIVADO_BIN -source build.tcl -tclargs $NOW

pwd
ls -la
```