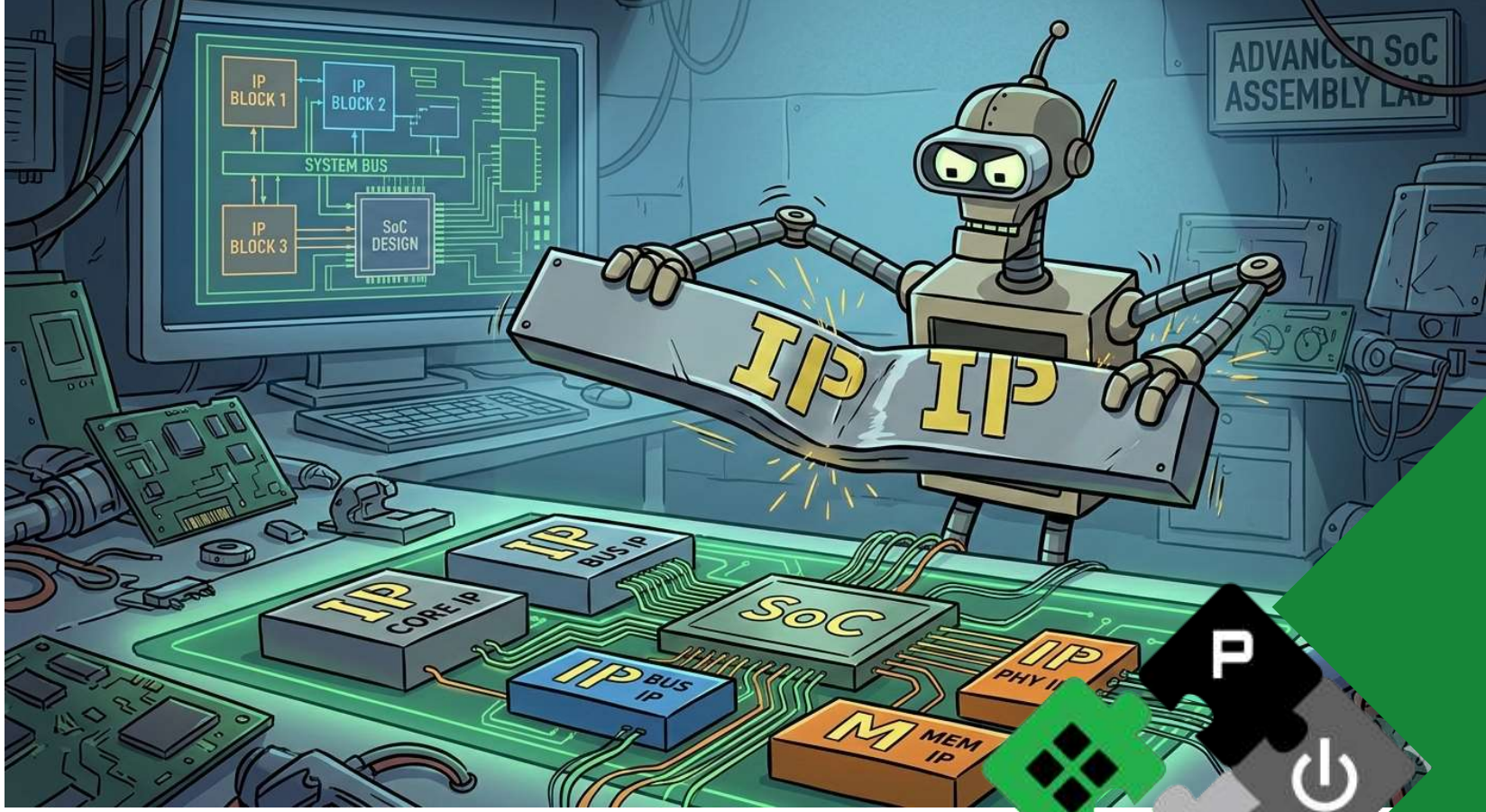
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Hardware Dependency Management with Bender



Michael Rogenmoser
michaero@iis.ee.ethz.ch
Tim Fischer
fischeti@iis.ee.ethz.ch

PULP Platform
Open Source Hardware, the way it should be!

@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

Who am I?

- **Michael Rogenmoser**
  - PhD Student at ETH Zurich's PULP Platform since 2021
  - Research on fault tolerant SoC design
  - Interact with many components and IPs throughout the PULP group

- **Why am I talking about bender?**
  - Started working on bender in 2020
  - Maintainer of bender since 2021

Who am I?

- **Tim Fischer**
  - PhD Student at ETH Zurich's PULP Platform since 2021
  - Working on Network-on-Chips (FlooNoC)
  - And large-scale systems (picobello)

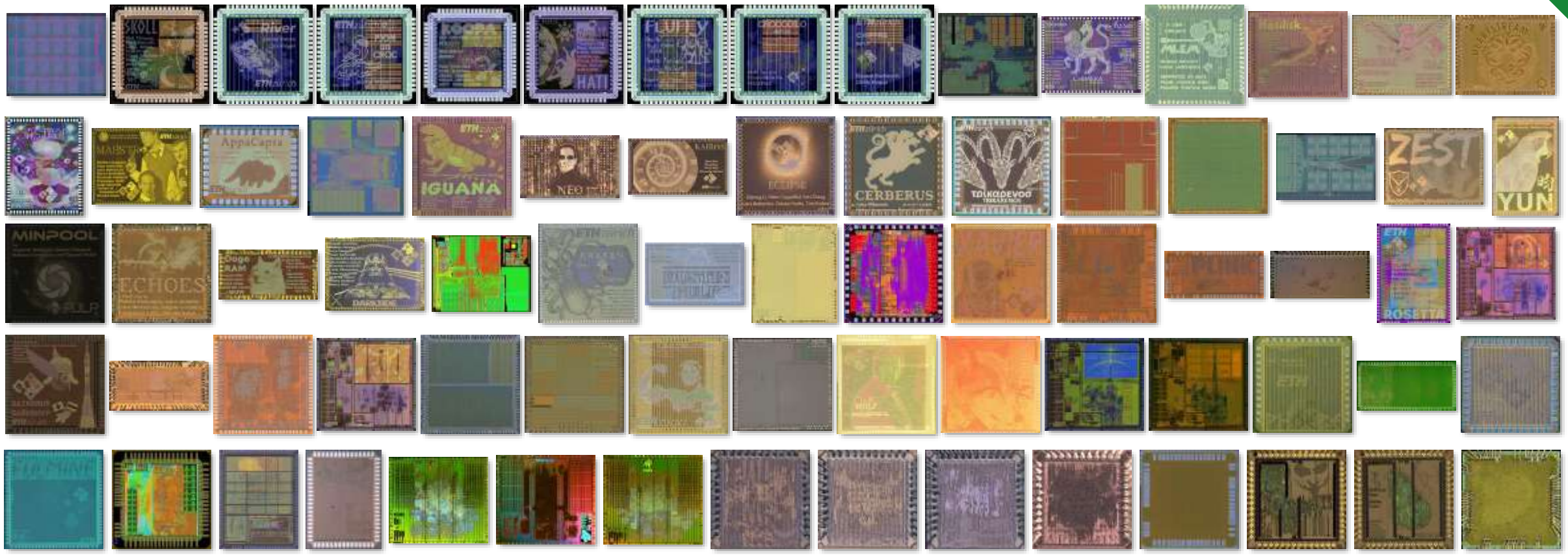
- **Why am I talking about bender?**
  - Learned that proper Dependency management is very important
    - especially for large-scale tapeouts
    - E.g. picobello has 40+ dependencies
  - Wanted to learn Rust and contribute back to bender, fell in love with it ❤

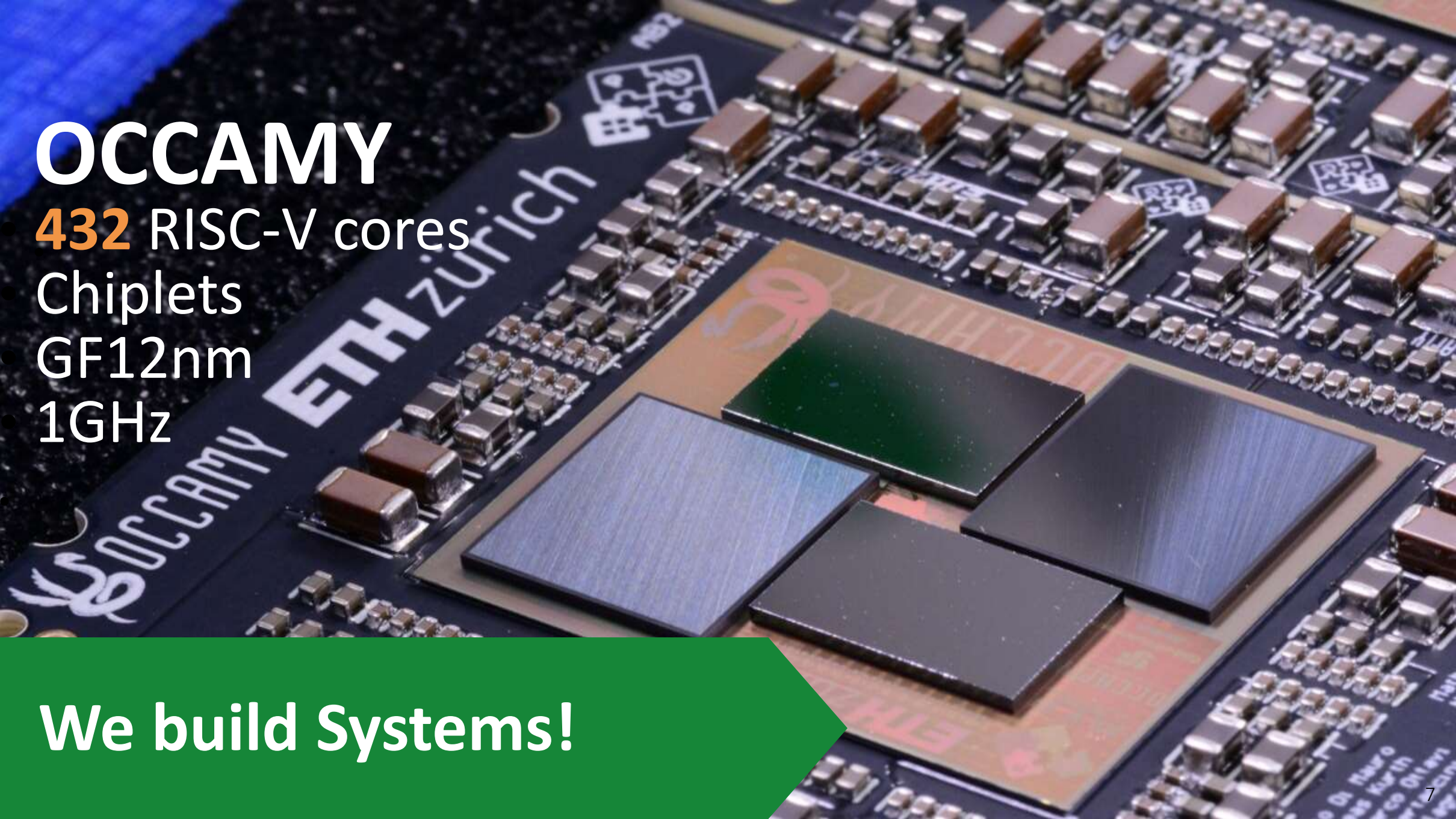
Why do we need Bender?

- Why do we need a **tool** to manage dependencies?
- Why do we need to **manage** dependencies?
- Why do we need **dependencies**?
- **Why???**

What are we doing?

In 12+ years PULP team has designed and tested 75 ASICs



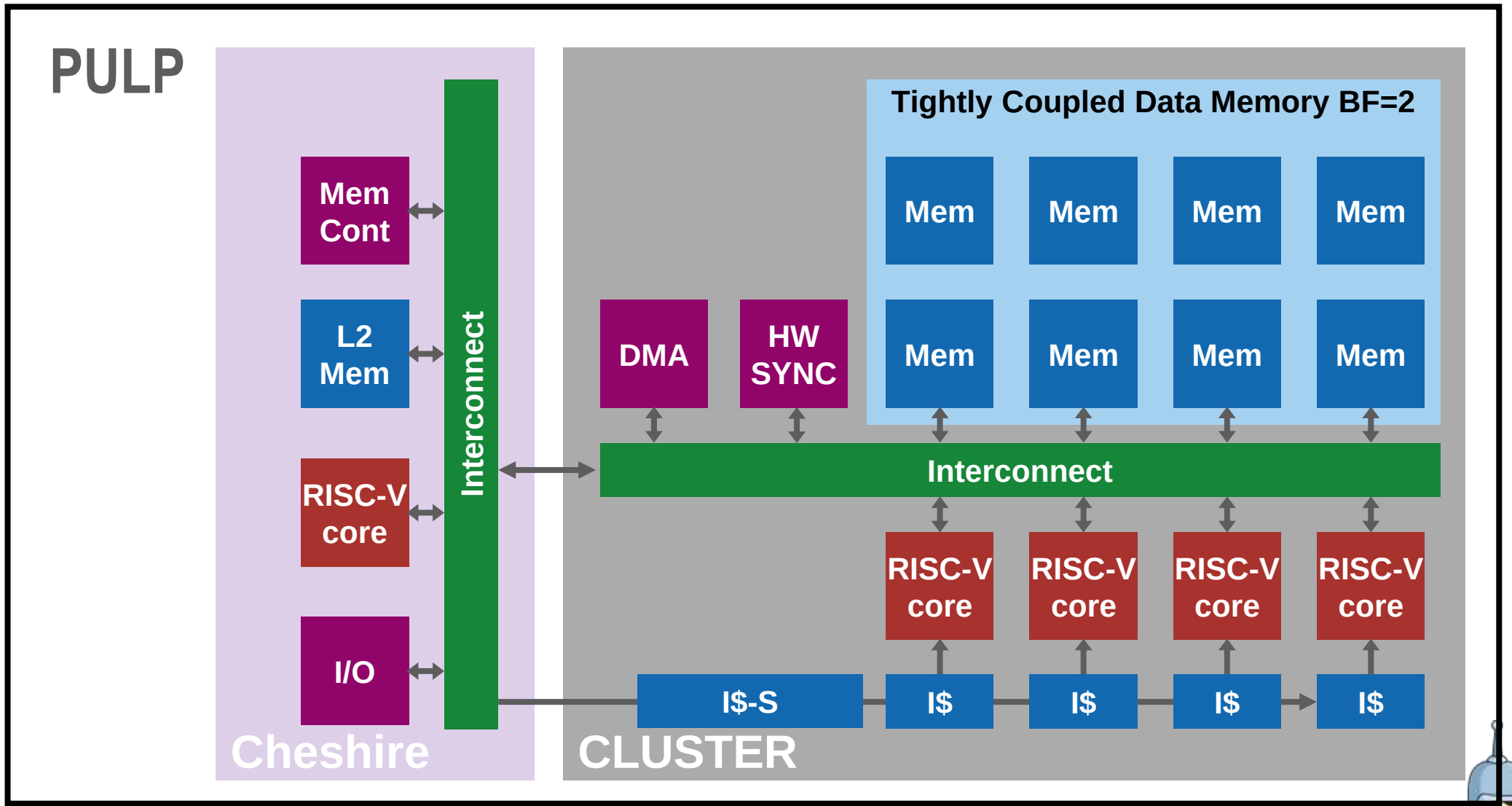


OCCAMY

- **432** RISC-V cores
- Chiplets
- GF12nm
- 1GHz

We build Systems!

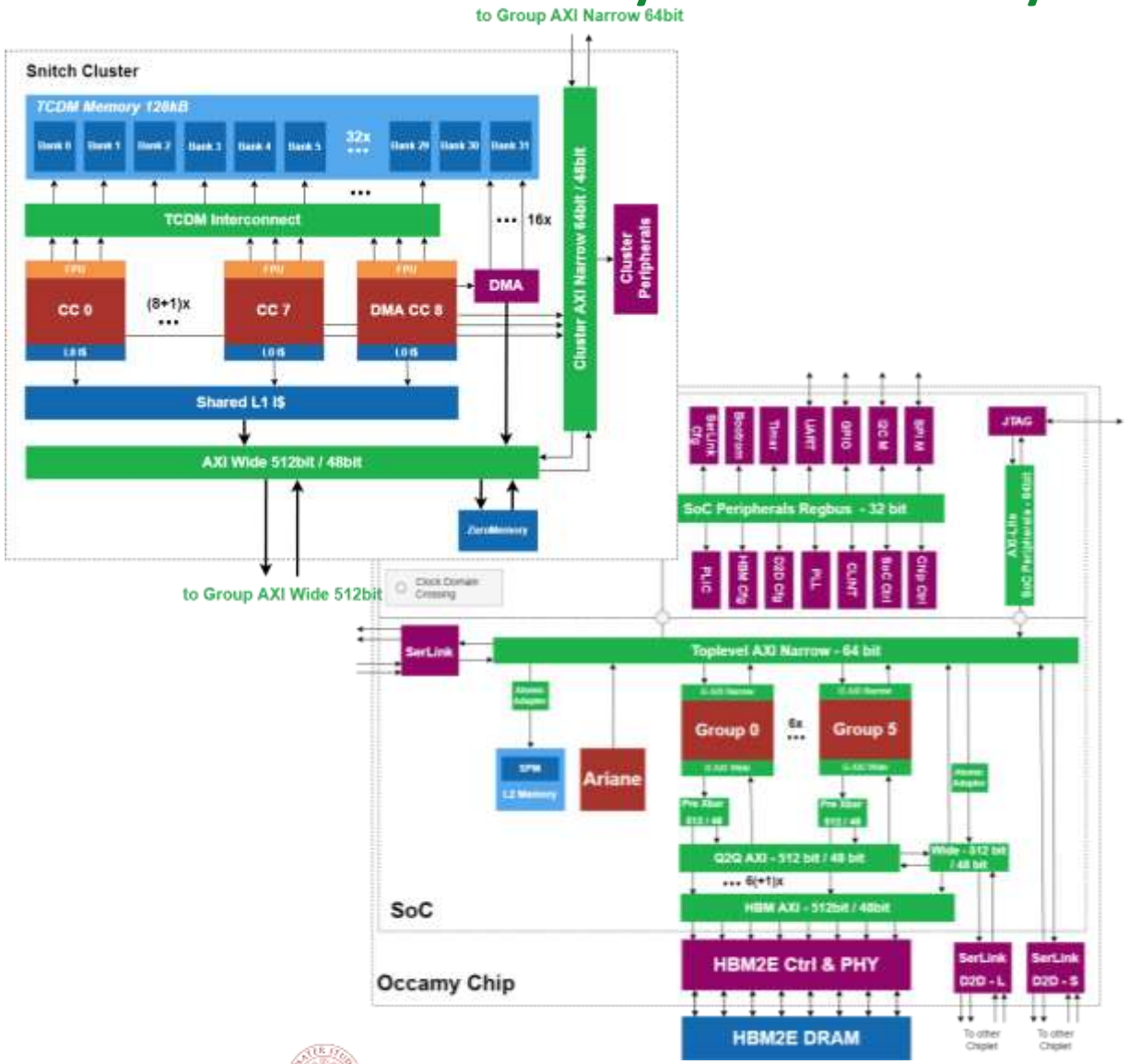
We build systems!

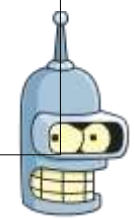


Why do we need Bender?

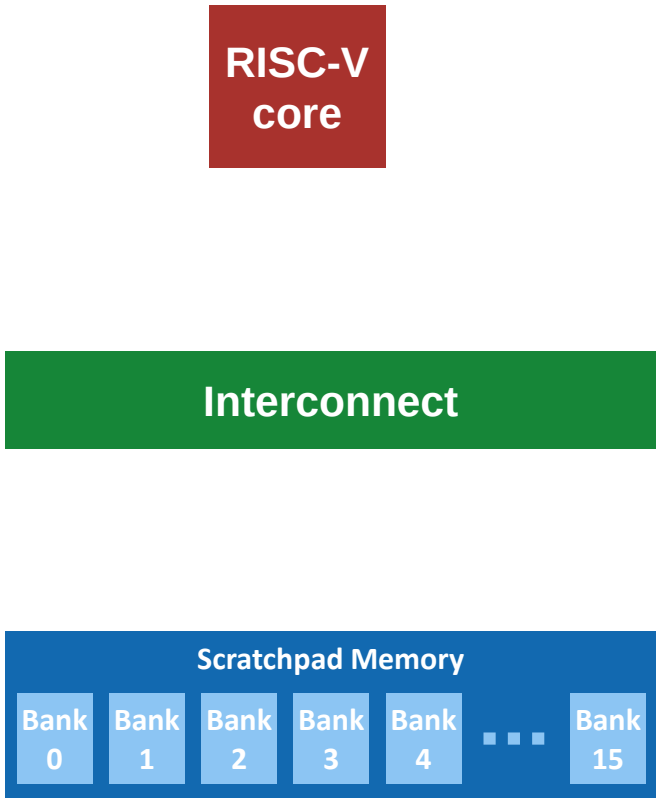
- Why do we need a **tool** to manage dependencies?
- Why do we need to **manage** dependencies?
- Why do we need **dependencies**?
- **Why???**
  - **We're building systems!**

But we build many different systems…





Similar IPs are used in ALL systems!

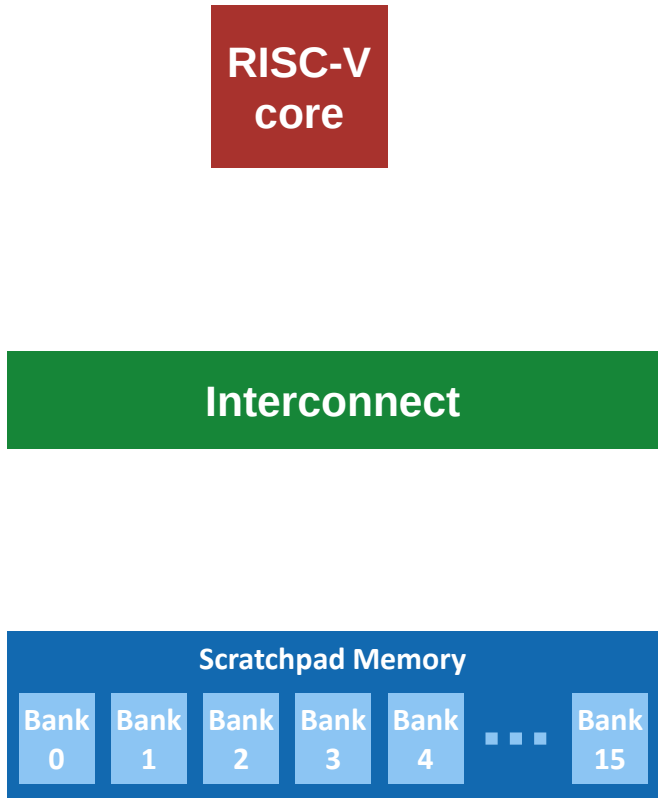


- **We want to use the same IP!**
  - Collaboration is key!
  - We need **dependencies**.

Why do we need Bender?

- Why do we need a **tool** to manage dependencies?
- Why do we need to **manage** dependencies?
- Why do we need **dependencies**?
  - **We want to reuse components**
- **Why???**
  - We're building systems!

Similar IPs are used in ALL systems!



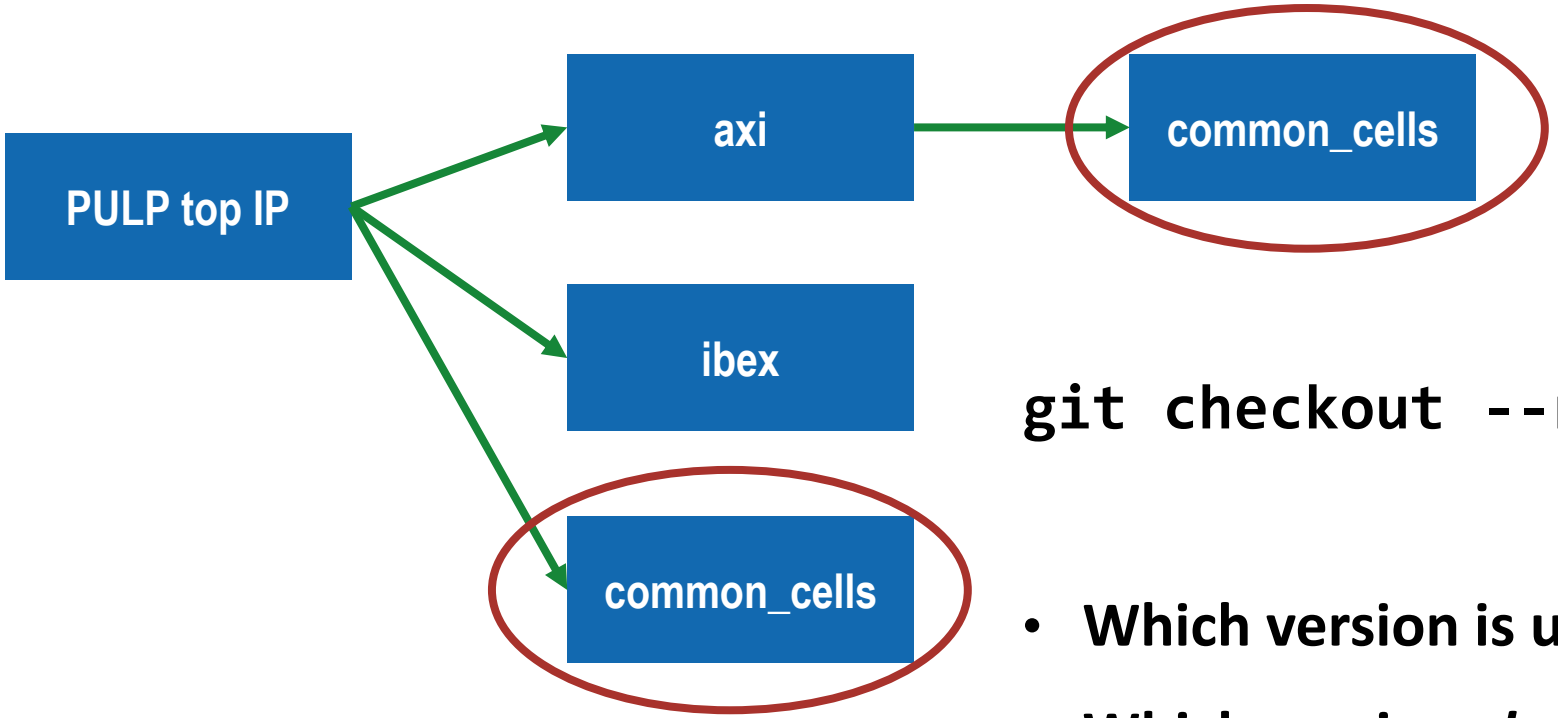
- **We want to use the same IP!**
  - Collaboration is key!
  - We need **dependencies**.
- **How? We can copy-paste the files?**
  - One system fixes a bug, how is this transferred back?
  - Systems add features, how are these transferred back?
- **Git is great to collaborate on IPs!**
  - Are all IPs in a single git project?
  - What if different features in an IP conflict?
- We need to **manage** dependencies.

Why do we need Bender?

- Why do we need a **tool** to manage dependencies?

- Why do we need to **manage** dependencies?
  - **Components get updates that we should track and share**
- Why do we need **dependencies**?
  - We want to reuse components
- **Why???**
  - We're building systems!

Dependency Conflict



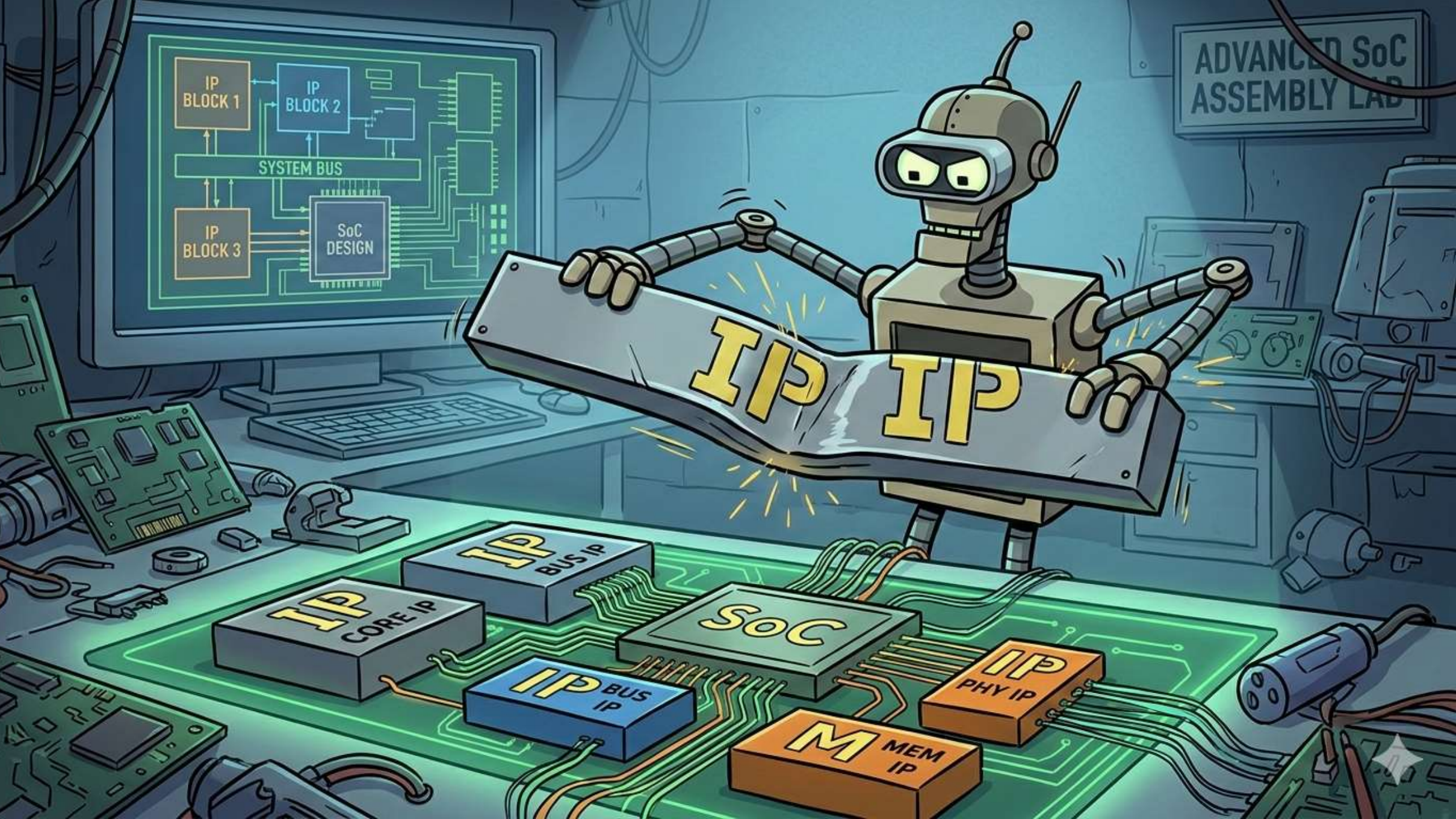

```
git checkout --recursive
```

- **Which version is used?**
- **Which version *should* be used?**

- **We need a tool to manage dependencies.**

Why do we need Bender?

- Why do we need a **tool** to manage dependencies?
  - **Updates and requirements can conflict**
- Why do we need to **manage** dependencies?
  - Components get updates that we should track and share
- Why do we need **dependencies**?
  - We want to reuse components
- **Why???**
  - We're building systems!


IP BLOCK 1
IP BLOCK 2
SYSTEM BUS
IP BLOCK 3
SoC DESIGN
ADVANCED SoC ASSEMBLY LAB
IP IP
IP BUS IP
IP CORE IP
SoC
IP BUS IP
IP PHY IP
M MEM IP

The Bender vision

- **IPs are developed in their *dedicated project***
- **IPs specify their dependencies**
  - Software-like development
  - Linked to a specific version of the dependency
- **Easy *IP re-use* in a variety of systems**
  - Incorporate updates from all projects!

- **Bender takes care of fetching all dependencies**
  - The tool is told where to find the dependency and picks a correct version
- **Bender helps with running EDA tools**

Workflow: init

- **Project defined with a `Bender.yml` file at the project root**
- **IP/package built inside a git project**
- **Convenience function to initialize a project**

```
   # new workspace folder
$> mkdir my_new_ip
$> cd my_new_ip

   # initialize git project
$> git init

   # initialize bender project
$> bender init
```

Defining an IP

- **Package definition at the top of `Bender.yml`**
- **License header**
  - Strongly encouraged
  - Open-source license information
  - Confidentiality specification if closed
- **IP Name**
- **IP Authors**
  - optional, but recommended
- **IP description**
  - optional

```
# Copyright (c) 2026 ETH Zurich and
University of Bologna.
# Solderpad Hardware License, Version
0.51, see LICENSE for details.
# SPDX-License-Identifier: SHL-0.51


package:
  name: my_new_ip
  authors:
    - “Michael Rogenmoser
          <michaero@iis.ee.ethz.ch>”
    - “Bender Workshop Audience”
```

Workflow: dependencies

- **Every project declares the IPs/packages it depends on**
- **Bender collects these dependencies for your projects**

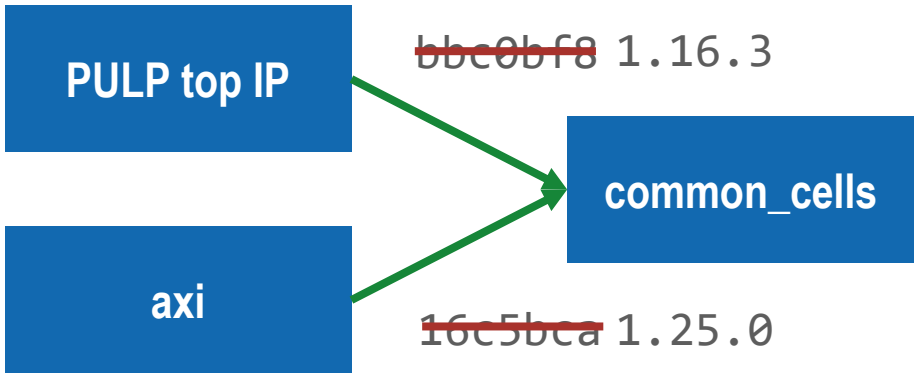
Managing Dependencies

- **Specify dependencies**
  - Use dependency's name defined in its `Bender.yml`
- **Git Version dependency**

```
dependencies:
  axi: { git: "https://github.com/pulp-
      platform/axi.git", version: 0.38.0 }
```

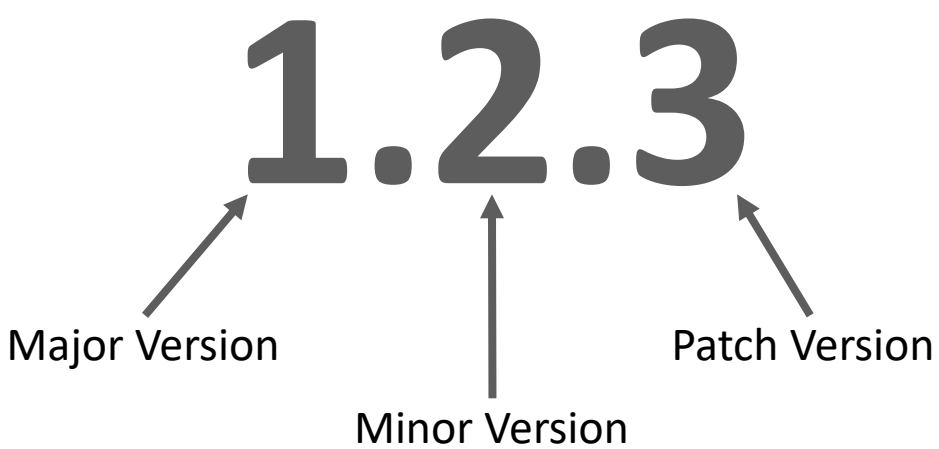
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Git version linked dependencies



- **Which commit should we pick?**
- **Are these commits compatible?**

- Use Semantic Versioning!
  - semver.org



- Major Version for ***breaking*** changes
- Minor Version for ***backwards-compatible*** fixes
- Patch Version for small fixes

Git version linked dependencies

- **Bender performs *version resolution*!**
  - Bender will figure out for you which version is the correct one to use.
- **Picks the *newest, compatible* version of the IP**
  - This assumes the IP maintainers ***properly*** tag their repositories!
- **If versions conflict, the user is asked to resolve**
- **IP repository is automatically downloaded and linked with bender.**

Semver requirements supported by bender

- **Caret ^ (default when nothing specified)**
  - `^I.J.K` (for I>0) — equivalent to >=I.J.K, <(I+1).0.0
  - `^0.J.K` (for J>0) — equivalent to >=0.J.K, <0.(J+1).0
  - `^0.0.K` — equivalent to =0.0.K
  - `^I.J` (for I>0 or J>0) — equivalent to ^I.J.0
  - `^0.0` — equivalent to =0.0
  - `^I` — equivalent to =I
- **Equals =**
  - `=I.J.K` — exactly the version I.J.K
  - `=I.J` — equivalent to >=I.J.0, <I.(J+1).0
  - `=I` — equivalent to >=I.0.0, <(I+1).0.0
- **Tilde ~**
  - `~I.J.K` — equivalent to >=I.J.K, <I.(J+1).0
  - `~I.J` — equivalent to =I.J
  - `~I` — equivalent to =I
- **`<`, `>`, `<=`, `>=` also allowed**
  - Comma-separated list in quotes
- **Wildcard ***
  - `I.J.*` — equivalent to =I.J
  - `I.*` or `I.*.*` — equivalent to =I
- **Resources:**
  - https://semver.org/
  - https://docs.rs/semver/1.0.24/semver/enum.Op.html

Managing Dependencies

- **Specify dependencies**
  - Use dependency's name defined in its `Bender.yml`
- **Git Version dependency**
  - Semantic Versioning
- **Git Revision dependency**
  - Using unique git tags
- **Path dependency**
- **Default remote can be specified for git version dependencies**

```
dependencies:
  axi: { git: "https://github.com/pulp-
      platform/axi.git", version: 0.38.0 }

  ibex: { git:"https://github.com/lowRISC/
      ibex.git", rev: pulpissimo-v6.1.1 }

  secret_sauce: { path:
      "~/personal_ips/secret_sauce" }
```

New: Default remotes

- **Convenience for inferring git URL**
  - 99% of dependencies are from PULP Github
- **Multiple remotes are also possible**
  - E.g. for internal Gitlab dependencies
  - Requires `default` selection
  - Check Bender README

```
remotes:
  pulp: "https://github.com/pulp-platform"


dependencies:
  # Much shorter this way
  axi: 0.38.0
  # Equivalent to:
  axi: { git: "https://github.com/pulp-
                platform/axi.git",
         version: 0.38.0 }
  # or
  axi: { version: 0.38.0 }
```

Lock the dependencies!

- **Re-resolving dependencies can cause issues**
  - What if someone updates an IP you use to a new version?
  - How do you keep consistency if you manually select from conflicting versions?

- **Consistency & reproducibility is key!**

- **Dependencies are locked using a `Bender.lock` file.**
  - Fixed commit hash (or location) is stored
  - This is used for all commands, unless an `update` is performed.
  - To ensure your projects remain consistent, check in this file for your project!

Workflow: dependencies

- **Every project declares the IPs/packages it depends on**
- **Bender collects these dependencies for your projects**
- `update` **determines unique version of dependency to use and locks it**
- `checkout` **fetches locked versions**
- `clean` **removes checkouts**
- **Individual updates possible**
  - Recursive also supported

```
   # Update all dependencies
$> bender update


   # checkout dependencies (Bender.lock)
$> bender checkout

   # Remove all checked out dependencies
$> bender clean


   # Update only individual dependency
$> bender update common_cells

   # Update dependency and its
   # dependencies recursively
$> bender update snitch_cluster -r
```

Dependency conflict resolution

- **Conflicting requirements require manual resolution**
- **Required versions outlined with their sources**
- **If lockfile is available, version specified is shown and can be selected**

```
Dependency requirements conflict with
  each other on dependency common_cells.

- package my_new_ip  requires  =1.16.0
  (1.16.0 <= x < 1.16.1)
- package axi        requires  ^1.37.0
  (1.37.0 <= x < 2.0.0)

The previous lockfile required =1.39.0.

To resolve this conflict manually, select
  a revision for common_cells among:
0)  =1.39.0
1)  ^1.37.0
2)  =1.16.0

Enter a number or hit enter to abort:
```

Workflow: Dep. information

- **Many commands are available to get information about dependencies**
- **What IPs/packages are included?**
  - Which order are they required?
  - How can the packages be configured?
- **Which packages require a specific dependency?**
- **Where are the packages?**
- **What updates are possible?**
- **These commands have options**
  - check `--help`

```
   # list packages/dependencies
$> bender packages

   # List packages calling a dependency
$> bender parents common_cells

   # Show path to checked out dependency
$> bender path common_cells

   # List dependencies with available
   # updates and conflicts
$> bender audit
```

Workflow: sources

- **Every project declares:**
  - Source files
  - Defines
  - Include directories

```
    # Prints internal json with
    # source file hierarchy
$> bender sources
```

Source file management

- **Order files in compile order!**
  - Packages First
    - Used by other modules
  - Lower-level modules before the higher-level modules

- **Source file groups**
  - Add custom defines
  - Local include directories

```
sources:
  # Level 0
  - src/my_new_ip_pkg.sv
  # Level 1
  - src/my_new_ip_core.sv
  # Level 2
  - src/my_new_ip_top.sv

  - files:
      - src/tb/audience.sv
      - src/tb/my_new_ip_tb.sv
    defines:
      AUDIENCE_SIZE: 64
      USE_AUDIENCE: ~
    include_dirs:
      - src/include
```

Include directories

- **Creates `+incdir+` directives**
  - For `` `include "filename.svh" `` in SystemVerilog
- **Export to dependencies calling this**
  - Also applied to local source files
  - Recommended: Add a subdirectory with dependency name
    - Avoids filename conflicts with other incdirs
- **Limit include directory to specific files**
  - Depending on script, may not be limited

```
export_include_dirs:
 - include

sources:
  - files:
      - src/tb/audience.sv
      - src/tb/my_new_ip_tb.sv
    include_dirs:
      - src/include
```

Workflow: script

- **Extract bender description for EDA tools**
- **Simplest format: `flist-plus`**
  - List of files
  - Plusargs for `+incdir+` and `+define+`
  - Supported by most tools
- **Custom formats for other tools**
  - `vsim`, `vcs`, `synopsys`, `genus`, …
  - Some formats have additional options
- **Pass additional defines**

```
   # generate script, output to file
$> bender script flist-plus > filelist.f
```

```
   # Other formats with specific options
$> bender script vsim --verilog-args
   "-suppress vlog-2583”

   # Add define to script
$> bender script vsim –D NEW_AUDIENCE
```

Workflow: script

- **Limit the included packages**
  - Determines the dependencies required by only the specified packages

```
   # Include only specific package(s)
   # and their dependencies
$> bender script flist-plus -p axi
```

- **Exclude specific packages**
  - Excludes packages and dependencies not required elsewhere

```
   # Exclude specific package(s)
$> bender script flist-plus -e axi
```

- **Exclude all dependencies/packages not specified**

```
   # Exclude all dependencies
$> bender script flist-plus -n
```

- **Flags can be mixed**

Workflow: script

- **All script formats are generated from a template**

- **https://keats.github.io/tera/docs/**

- **Custom templates accepted**

`bender script template --template <TPL>`

```
{% for group in srcs %}

{% for file in group.files %}

{{ file.file }}

{% endfor %}
{% endfor %}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Target definition

- **Group source files by a scope/feature**
  - RTL files
  - Drivers/testbenches
  - Netlists
- **Allows to selectively enable/disable groups**

```
sources:
  - files:
      - src/ip_pkg.sv
      - src/ip.sv

  - target: gate
    files:
      - netlist/ip.v

  - target: test
    files:
      - test/tb_ip.sv
```

Targets can also be combined

- **Complex target specification**
  - All `all()`
  - Any `any()`
  - Not `not()`
  - Wildcard `*`
- **You can nest and combine targets**
  - To select between technologies
  - For non-synthesizable logic
  - To exclude certain tools from using them

```yaml
sources:
  # Choose depending on technology
  - target: all(asic, tsmc7)
    files:
      - src/tsmc7/macro.sv
  - target: all(asic, gf12)
    files:
      - src/gf12/macro.sv

  # Behavioral models for PS/PL-sim
  - target: all(asic, not(synthesis))
    files:
      - technology/verilog/sram.v

  # Exclude some files for specific tool
  - target: all(test, not(verilator))
    files:
      - src/ip_test.sv
```

Targets usage

- **Some targets are pre-defined:**
  - `vsim` defines `simulation`
  - `synopsys` defines `synthesis`
- **Targets are also turned into defines**
  - E.g. `TARGET_SYNTHESIS`
- **Targets can be specified for each tool call**
  - `test` for RTL simulation
  - `asic` for backend scripts
- **Scoped targets are also supported**

```
# Specify a global target
$> bender script vsim -t test

 # Specify a target for a dependency
$> bender script vsim -t axi:test

 # Exclude a target from a dependency
$> bender script vsim -t -axi:test
```

New: dependency targets

- **Include dependency only when target enabled**
  - E.g. to exclude test components
- **Pass target to a dependency**
  - E.g. to select a specific feature

```
dependencies:
  # Include only for testing
  common_verification:
      {version: 0.38.0, target: test}

  # Pass a target to a dependency
  cva6: { version: 0.5.3
          pass_targets: cv32a6_imac_sv32 }
```

Bender hygiene for targets

- **Guard verification dependencies**
  - They are rarely needed for the IP that integrates it
- **Use established target names**
  - `test` for testbenches
  - `rtl` for synthesizable code
  - `gate` for netlists
  - `simulation` for RTL simulation
  - `asic`/`fpga` for implementation
- **Keep it simple**
  - Lowercase, no symbols

```
sources:
  # Dos 👍
  - target: test
    files:
      - test/tb_ip.sv

  - target: rtl
    files:
      - src/ip.sv
  - target: gate
    files:
      - out/ip.v

  # Don’ts 👎
  - target: +MyIP:alu
    files:
      - src/alu.v
```

Workflow: dep. modification

1. Clone the dependency

- Use bender clone to check out the dependency locally for editing.

2. Edit the code

- Navigate to the dependency working directory and make your changes.

3. Commit & push changes

- Create a feature branch, stage, commit, and push your changes via Git.

4. Snapshot the new state

- Run bender snapshot to lock the dependency to the new commit reference.

```
   # Clone dependency to work on it
$> bender clone my_dep

   # Edit code in dependency location
$> cd working_dir/my_dep

   # Commit and push all changes
$> git checkout -b new_feature
$> git add *
$> git commit -m “updates”
$> git push -u origin new_feature

$> cd ../..

   # Run snapshot to reference commit
$> bender snapshot
```

Bender.local

- **Local configurations**
  - Database location, git configs, overrides
- **Overrides force dependency selection**
  - No manual selection required
  - Used for consistency in `clone`
  - Used for consistency in `snapshot`

```
overrides:
  obi: {git: <URL>, version: 0.1.7}

  axi: {path: working_dir/axi} # clone

  # apb: {path: working_dir/apb} # clone
  apb: {git:<URL>, rev:<HASH>} # snapshot
```

Bender hygiene for deps.

- **Versioned releases are preferred**
  - Can be resolved by bender
- **Tags are also okay**
  - They are fixed/persistent
  - Resistant against force-pushes
- **Commit hashes are okay-ish**
  - Fixed, but can become untracked
  - Add a comment from which branch!
- **Branches should be temporary**
  - (force)-pushes might lead to unwanted behaviour
    - New or untracked commits

```
dependencies:
  # Dos 👍
  axi: {version: 0.38.0 }
  axi: {rev: “some_tag” }

  # Okay-ish 🆗
  axi: {rev: “4e2aad…” } # “some_branch”

  # Don’ts 👎
  axi: {rev: “4e2aad…” }
  axi: {rev: “some_branch” }
```

So many files…

	`Bender.yml`	`Bender.lock`	`Bender.local`
What does it **do**?	Declares **package, sources & dependencies**	List of all locked **dependency versions**	**Manual configuration**
Should you **edit** it?	**Yes**	**Do not touch**	**Yes**
Does **bender** edit it?	**No** (except `` `bender init` ``)	**Yes** (`` `bender update` ``)	**Yes** (e.g. `` `bender clone/snapshot` ``)
Should you **track** it with git?	**Yes**, required	**Suggested**, (if in doubt **yes**)	**No,** refers to local configuration
Is it used for **version resolution**?	**Yes**	**No,** only hints	**Yes**

Workflow: vendor

- **Not all open-source projects include bender compatibility**
- **Functionality similar to lowrisc's `vendor.py` script**
- **Copy files from separate repository**
  - Specific hash required
  - Patch generation and application
  - Separate functionality from rest of bender

```
   # Copy files from other project
   # Applies patches specified locally
$> bender vendor init


   # Show local changes not in patch
$> bender vendor diff


   # Generate patch from local changes
$> bender vendor patch
```

What we learned

- **Why do we need bender?**
- **How do we create a bender package?**
- **How do we use dependencies?**
- **How do we specify sources?**
- **How do we connect to EDA tools?**
- **How do we work on dependencies?**

Thank you!

Appendix

Version resolution (`resolver.rs`)

- **Iteratively collect requirements**
- **Determine the best possible match**
  - Usually highest version available matching constraints
  - Path, git revision need to be unique
- **If match not found, ask user**
  - User choice is saved for resolution
- **Add requirements from newly added/selected dependencies**
- **Unlock choice if new requirement is added**

Where are dependencies (and why)?

- **Bare repository cloned to `./.bender/git/db/`**
  - Bare git clone (i.e., only `.git` folder, not checkout)
  - Used for all possible dependency sources
  - Used to determine available versions, other git operations
  - Speeds up git operations
- **Checkout defaults to `./.bender/git/checkouts/`**
  - Checkout of git repository used for code
- **Hash in dependency name?**
  - Used to differentiate source (git url)
  - Not commit hash!

How does clone/snapshot work?

Clone

- **Copy existing checkout**
  - From `.bender/git/checkouts/` to `working_dir/`
  - If checkout exists, let user fix!
  - Properly link git remote
- **Update `Bender.local` override**
  - For consistency when running `update`
- **Update `Bender.lock`**
  - Links subsequent bender executions

Snapshot

- **Determine existing checkout hash**
  - Checks for override in `working_dir/`
  - Block if there are uncommitted changes
- **Update `Bender.local` override**
  - For consistency when running `update`
- **Update `Bender.lock`**
  - Links subsequent bender executions

Ongoing work: `slang` bindings

- **Slang is THE SystemVerilog parser/analyzer/rewriter**
  - Open-source, MIT-licensed
  - Built as library to be included in downstream
  - Best in class support for SystemVerilog
  - Used in many open EDA tools (slang-yosys, CIRCT)
  - Very fast, implemented in C++
- **First use case is the `pickle` feature**
  - Concatenate all files into a single one
  - Filter out unreachable files from top module
  - Rename module/package/interface names to prevent naming collisions
  - Designated successor of `morty`

Future features

- **Slang bindings open up a whole new world**
  - Filtering script file lists from unreachable modules
  - Formatting SystemVerilog files
  - Linting SystemVerilog files
  - Extracting docstrings from SystemVerilog modules/packages
  - Allowing multiple versions of a dependency with name prefixing

- **Prefixes for dependency versions**
  - Allows custom version parallel to mainline development

- **Other proposals?**