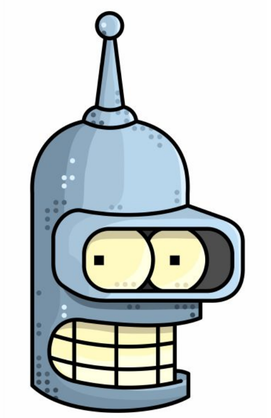


BENDER

A dependency management tool for hardware design projects

Fabian Schuiki

STATUS QUO

IPApprox / iptools

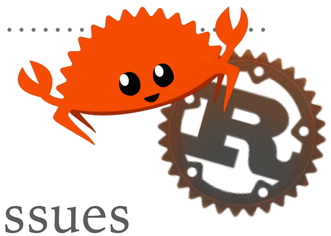
- Distinction between IPs/chips
- No transitive dependencies
  - **IPs don’t know their deps**
  - **Chips must list all deps**
- Tool embedded into chip repository
- Mixes:
  - IPs worked on by the user
  - IPs checked out by the tool

STATUS QUO

IPApprox / iptools

- Distinction between IPs/chips
- No transitive dependencies
  - **IPs don’t know their deps**
  - **Chips must list all deps**
- Tool embedded into chip repository
- Mixes:
  - IPs worked on by the user
  - IPs checked out by the tool

A SHOT AT SOMETHING NEW



Bender

- A replacement tool to fix these issues
- A joint effort by: Andreas Kurth, Francesco Conti, Stefan Mach, Florian Zaruba
- Repository and binaries: `github.com/fabianschuiki/bender`
- Or use cargo to build it

```
> cargo install bender
```

- Or build it yourself:

```
> git clone <url> bender
> cd bender
> cargo install
```

THE WISHLIST

THE WISHLIST

- Transitive dependencies

THE WISHLIST

- Transitive dependencies
- Tier-based, hands-off, opt-in policy
  - ⭐ **Tier 1:** Resolve package dependencies
  - ⭐ **Tier 2:** Collect source files
  - ⭐ **Tier 3:** Feed the tools

THE WISHLIST

- Transitive dependencies
- Tier-based, hands-off, opt-in policy
  - ⭐ **Tier 1:** Resolve package dependencies
  - ⭐ **Tier 2:** Collect source files
  - ⭐ **Tier 3:** Feed the tools
- No central registry

THE WISHLIST

- Transitive dependencies
- Tier-based, hands-off, opt-in policy
  - ⭐ **Tier 1:** Resolve package dependencies
  - ⭐ **Tier 2:** Collect source files
  - ⭐ **Tier 3:** Feed the tools
- No central registry
- Tailored to ASIC flow
  - Ultra conservative in updating IPs
  - Reproducible builds

THE WISHLIST

- Transitive dependencies
- Tier-based, hands-off, opt-in policy
  - ⭐ **Tier 1:** Resolve package dependencies
  - ⭐ **Tier 2:** Collect source files
  - ⭐ **Tier 3:** Feed the tools
- No central registry
- Tailored to ASIC flow
  - Ultra conservative in updating IPs
  - Reproducible builds
- Written in compiled language for static checks

TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs

TRANSITIVE DEPENDENCIES

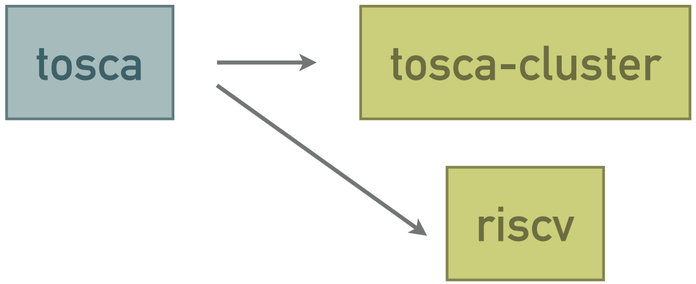
- IPs cannot declare their dependencies
- no standalone build for IPs

TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs

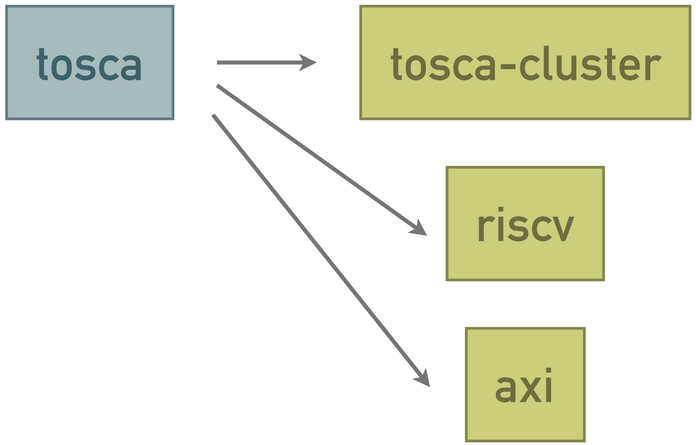
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs



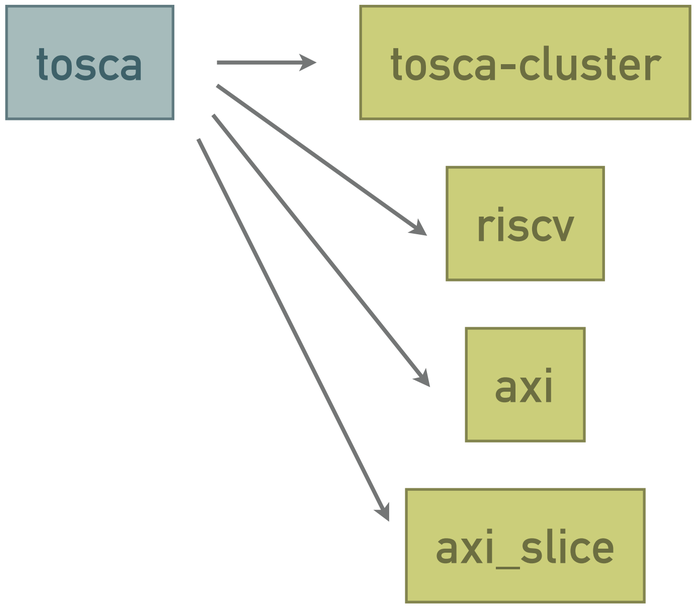
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs



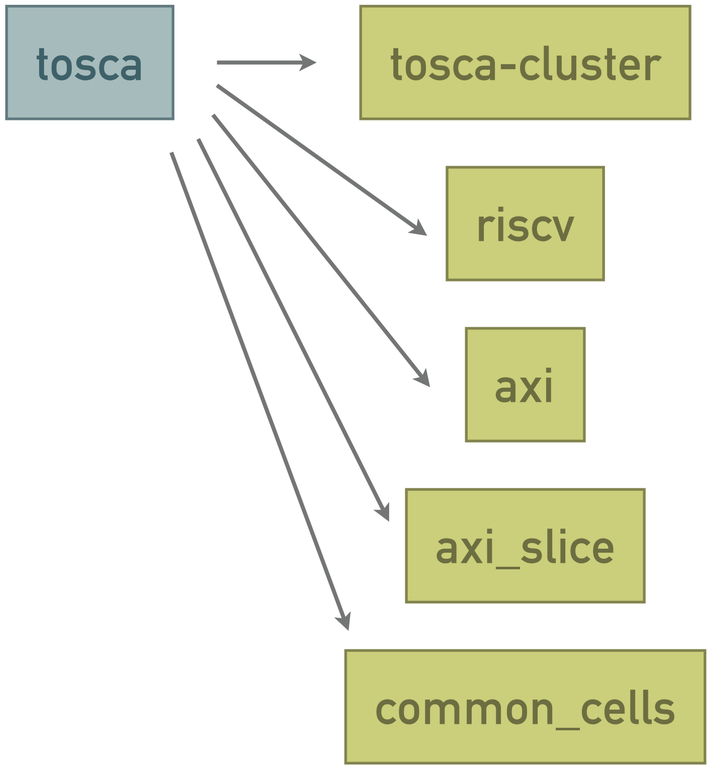
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs



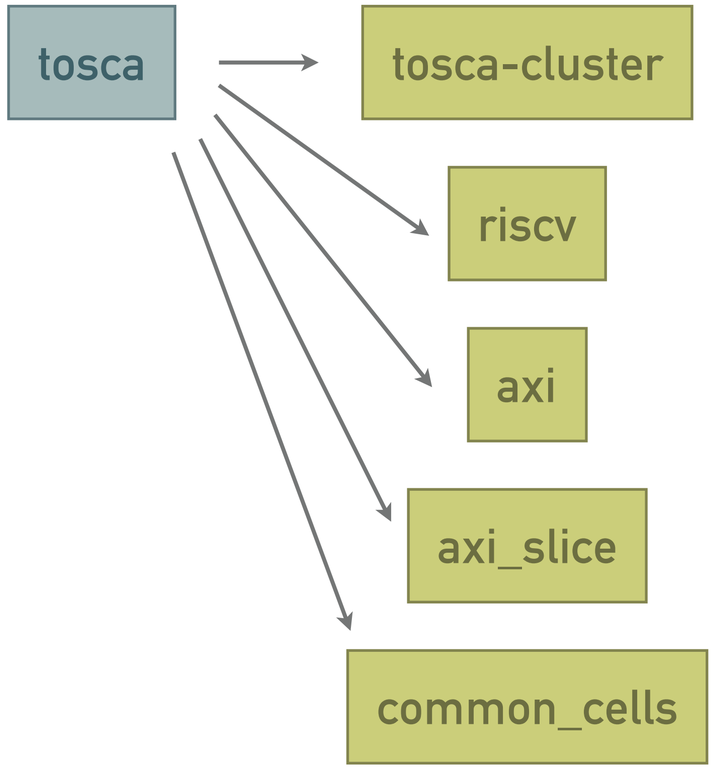
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs



TRANSITIVE DEPENDENCIES

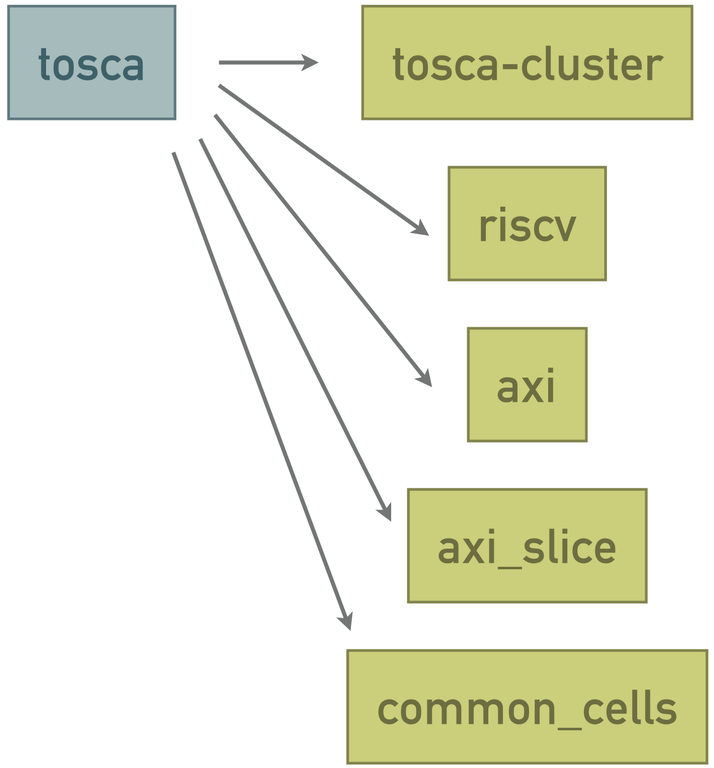
- IPs cannot declare their dependencies
- no standalone build for IPs



Current | *Bender*

TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs

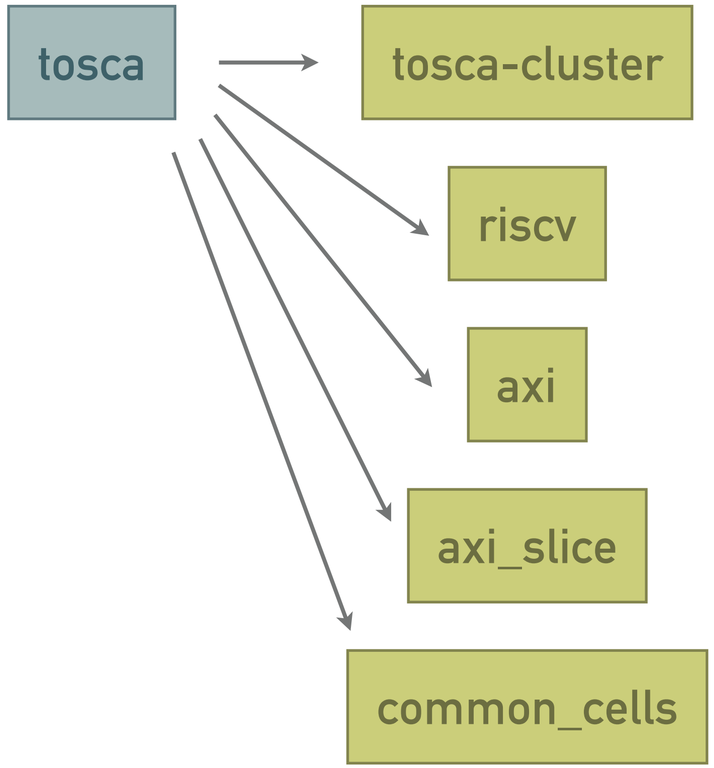


Current

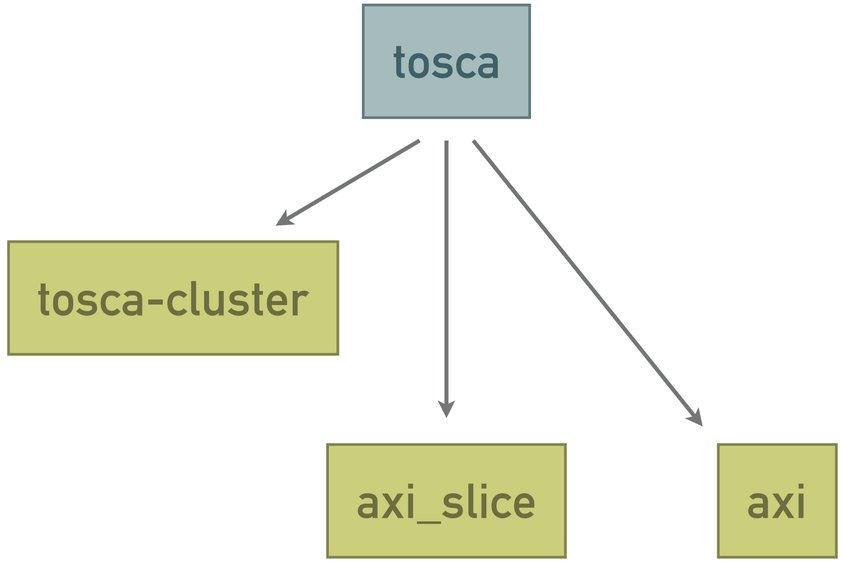
Bender

TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs



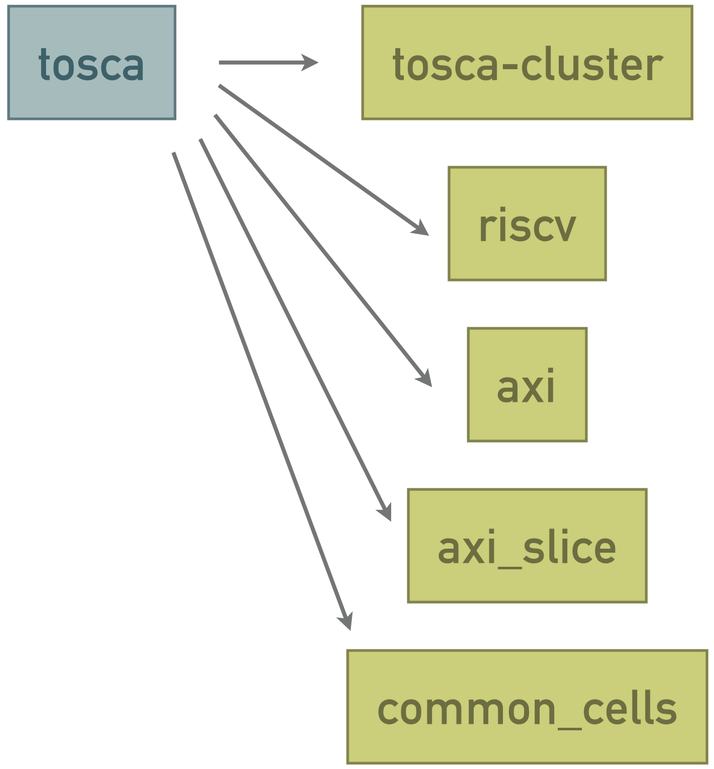
Current



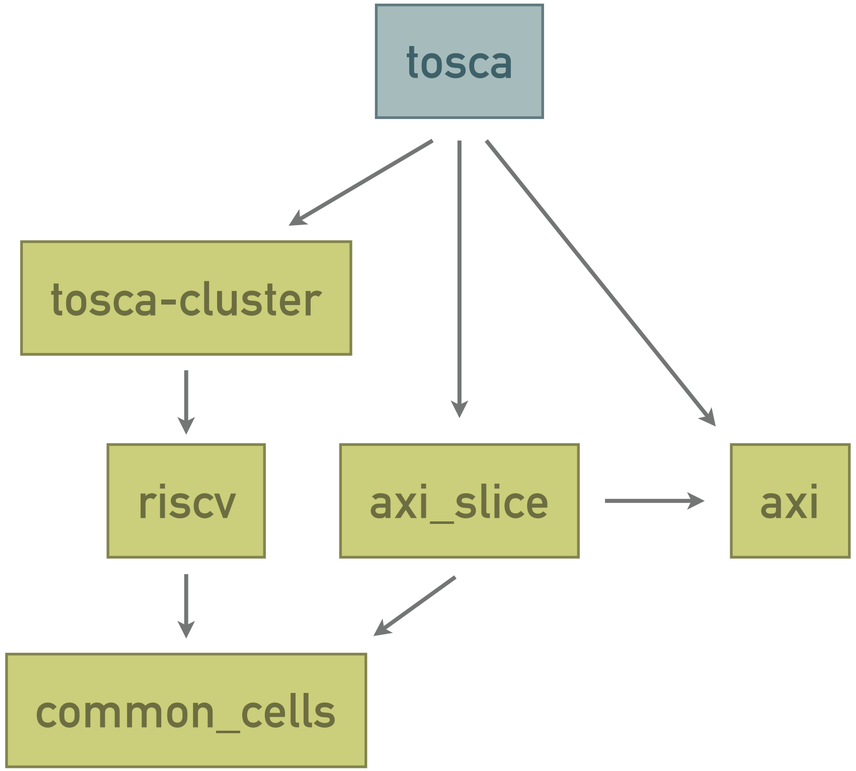
Bender

TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs



Current

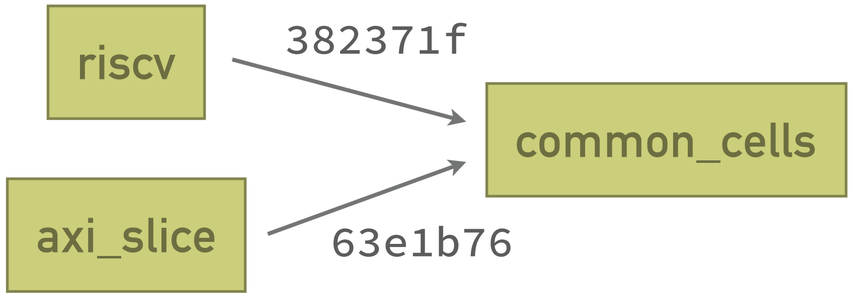


Bender

SEMANTIC VERSIONING

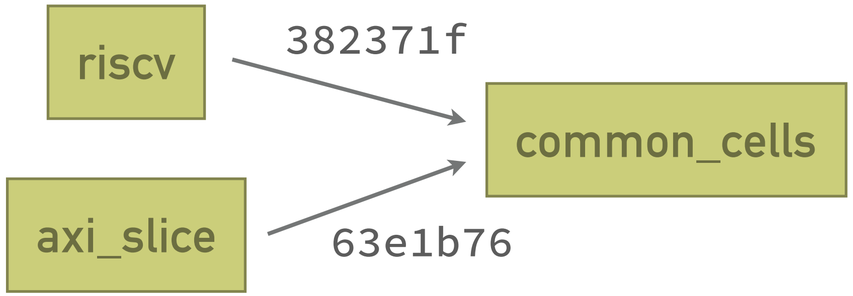
SEMANTIC VERSIONING

- The problem with transitive dependencies:



SEMANTIC VERSIONING

- The problem with transitive dependencies:

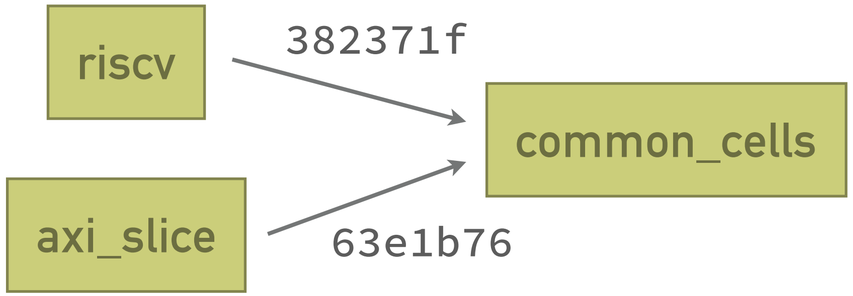


Are these commits compatible?

Which one do we pick?

SEMANTIC VERSIONING

- The problem with transitive dependencies:



Are these commits compatible?

Which one do we pick?

- The solution: **Semantic Versioning** (semver.org)

- Increment *major* version on **breaking** changes
- Increment *minor* version on **backwards-compatible** changes
- Increment *patch* version otherwise

Be careful with HDLs… many changes are breaking.

SEMANTIC VERSIONING

- The problem with tr

riscv

38

axi_slice

compatible?

ck?

- The solution: Semar

1

Major Version

Mino

n

n

anges

herwise

many changes

are breaking.

v0.3.0 - 2017-07-11

Added

- Set of five inferable, synchronous BRAM-based memories:
  - `SyncDpRam` (dual-port)
  - `SyncSpRam` (single-port)
  - `SyncSpRamBeNx32` (single-port N x 32bit with byte-wise enable)
  - `SyncSpRamBeNx64` (single-port N x 64bit with byte-wise enable)
  - `SyncTpRam` (two-port)

v0.2.3 - 2017-07-11

Fixed

- `src_files.yml`: added missing comma at the end of the `AxiToAxiLitePc` entry.

v0.2.2 - 2017-07-11

Fixed

REPRODUCIBLE BUILDS

lock files

- Make sure you know exactly what dependency versions were used for tape out

REPRODUCIBLE BUILDS

lock files

- Make sure you know exactly what dependency versions were used for tape out
- Software faces this problem as well (e.g. composer, cargo, etc.)

REPRODUCIBLE BUILDS

lock files

- Make sure you know exactly what dependency versions were used for tape out
- Software faces this problem as well (e.g. composer, cargo, etc.)
- Solution: Use a lock file!
  - Tracks exact hash of each dependency

Manifest

```
# Bender.yml
axi:        master
axi_slice:  master
axi_node:   v1.0.1
riscv:      fixes
axi2mem:    master
mem2axi:    34e598c
jtag:       master
```

Lock File

```
# Bender.lock
axi:        d1a671e
axi_slice:  f2e4abb
axi_node:   ac692ad
riscv:      352a9c6
axi2mem:    ead844f
mem2axi:    34e598c
jtag:       2b5a6ca
```

REPRODUCIBLE BUILDS

- Make sure you know exactly what dependency versions were used for tape out
- Software faces this problem as well (e.g. composer, cargo, etc.)
- Solution: Use a lock file!
  - Tracks exact hash of each dependency
- Dependencies only update …
  - to resolve version conflicts
  - when you ask for it

```
> bender update
> bender update axi_slice
```

Manifest

```
# Bender.yml
axi:        master
axi_slice:  master
axi_node:   v1.0.1
riscv:      fixes
axi2mem:    master
mem2axi:    34e598c
jtag:       master
```

Lock File

```
# Bender.lock
axi:        d1a671e
axi_slice:  f2e4abb
axi_node:   ac692ad
riscv:      352a9c6
axi2mem:    ead844f
mem2axi:    34e598c
jtag:       2b5a6ca
```

DEPENDENCY RESOLUTION

Overview

DEPENDENCY RESOLUTION

- Go through each dependency, determine which version to use
  - Do the same for dependencies of dependencies, …

DEPENDENCY RESOLUTION

- Go through each dependency, determine which version to use
  - Do the same for dependencies of dependencies, …
- This is the tricky part

DEPENDENCY RESOLUTION

- Go through each dependency, determine which version to use
  - Do the same for dependencies of dependencies, …
- This is the tricky part
- Semantic versioning helps here:
  - Dependencies specified with a range of compatible versions
  - Can make a table of available versions and start crossing out

DEPENDENCY RESOLUTION

- Go through each dependency, determine which version to use
  - Do the same for dependencies of dependencies, …
- This is the tricky part
- Semantic versioning helps here:
  - Dependencies specified with a range of compatible versions
  - Can make a table of available versions and start crossing out
- Implemented as backtracking algorithm

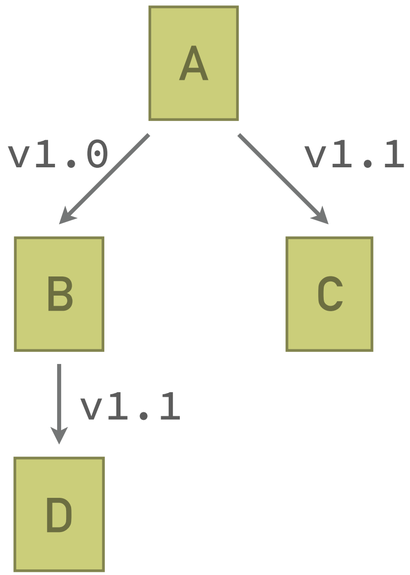
DEPENDENCY RESOLUTION

- Go through each dependency, determine which version to use
  - Do the same for dependencies of dependencies, …
- This is the tricky part
- Semantic versioning helps here:
  - Dependencies specified with a ra
  - Can make a table of availa ing out
- Implemented as backtracki

As a hardware developer, you don't want to do this!

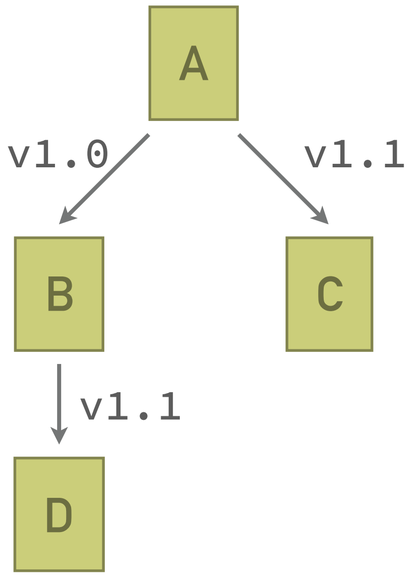
DEPENDENCY RESOLUTION

Simple Example



DEPENDENCY RESOLUTION

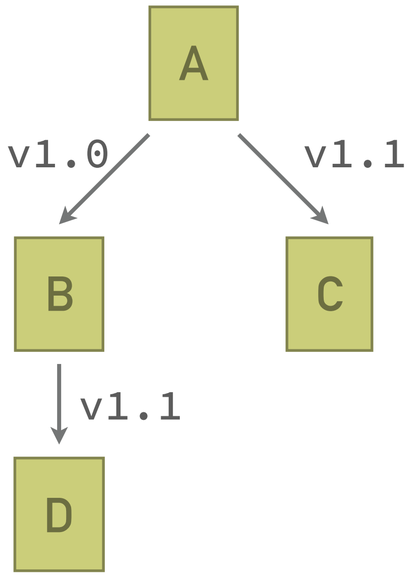
Simple Example



- We don’t have a registry with the dependency graph

DEPENDENCY RESOLUTION

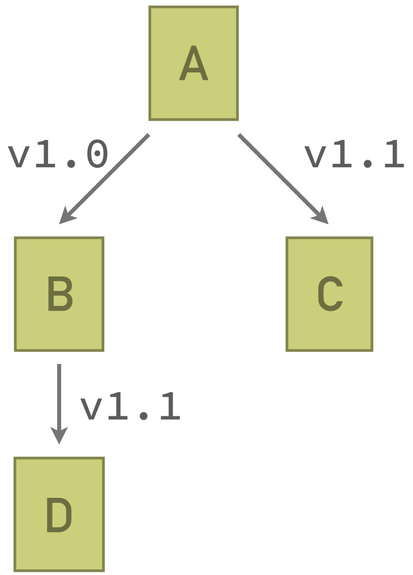
Simple Example



- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible

DEPENDENCY RESOLUTION

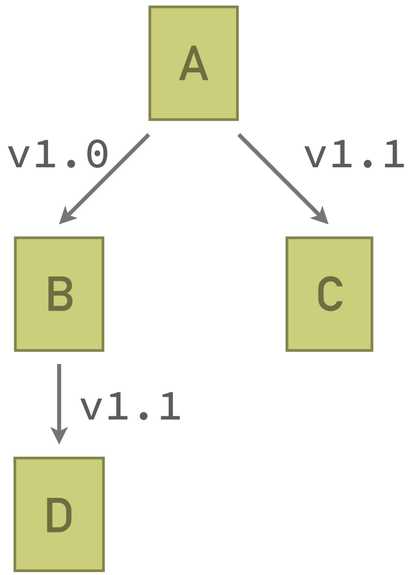
Simple Example



- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly

DEPENDENCY RESOLUTION

Simple Example



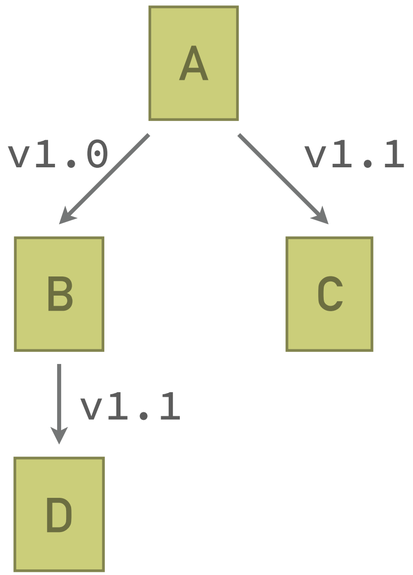
A

v1.0

- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly

DEPENDENCY RESOLUTION

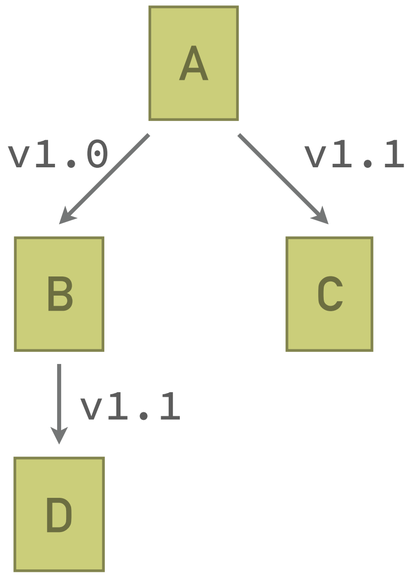
Simple Example



- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly

DEPENDENCY RESOLUTION

Simple Example

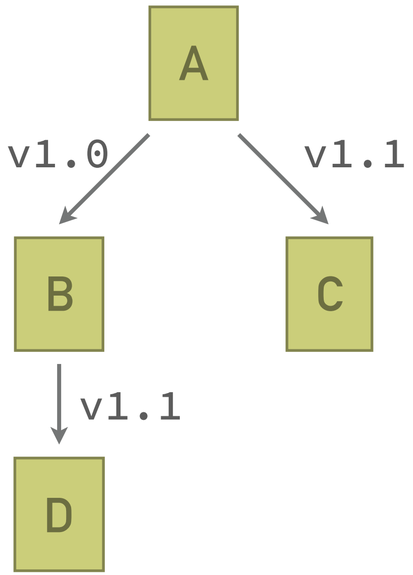


A	B	C
v1.0	*v1.1*	*v1.1*
	v1.0	*v1.0*

- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly

DEPENDENCY RESOLUTION

Simple Example

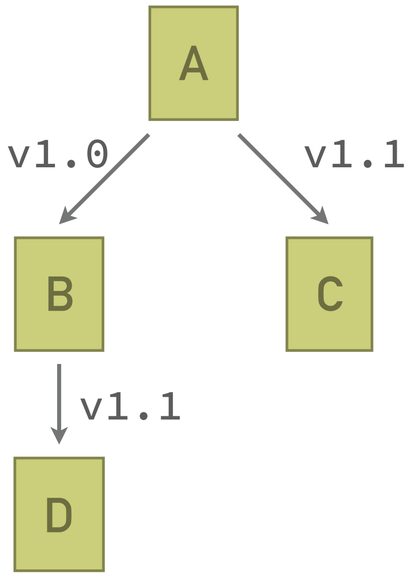


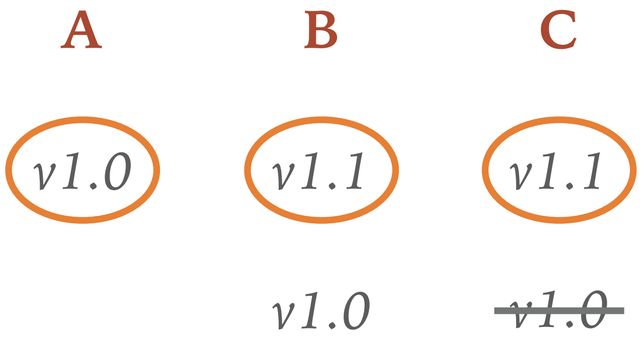
A	B	C
v1.0	*v1.1*	*v1.1*
	v1.0	~~*v1.0*~~

- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly

DEPENDENCY RESOLUTION

Simple Example

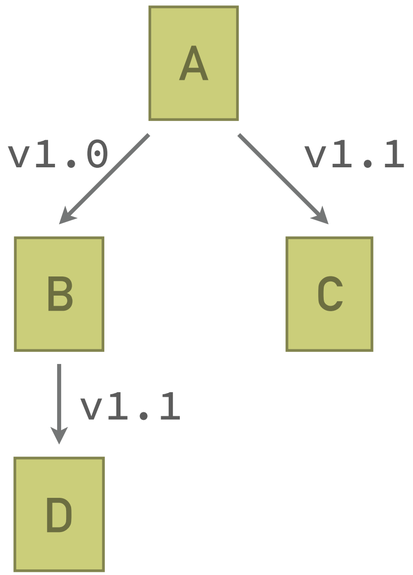




- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly

DEPENDENCY RESOLUTION

Simple Example

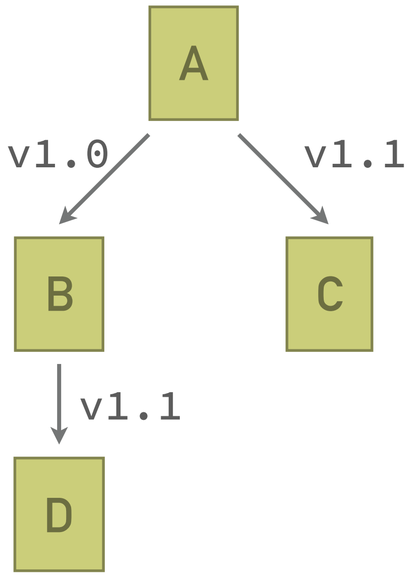


A	B	C	D
v1.0	*v1.1*	*v1.1*	*v1.2*
	v1.0	~~*v1.0*~~	*v1.1*
			v1.0

- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly

DEPENDENCY RESOLUTION

Simple Example

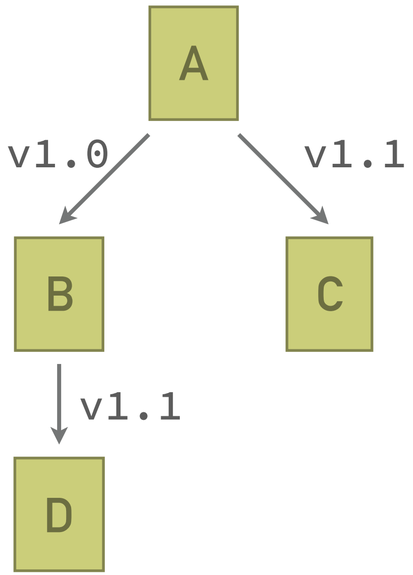


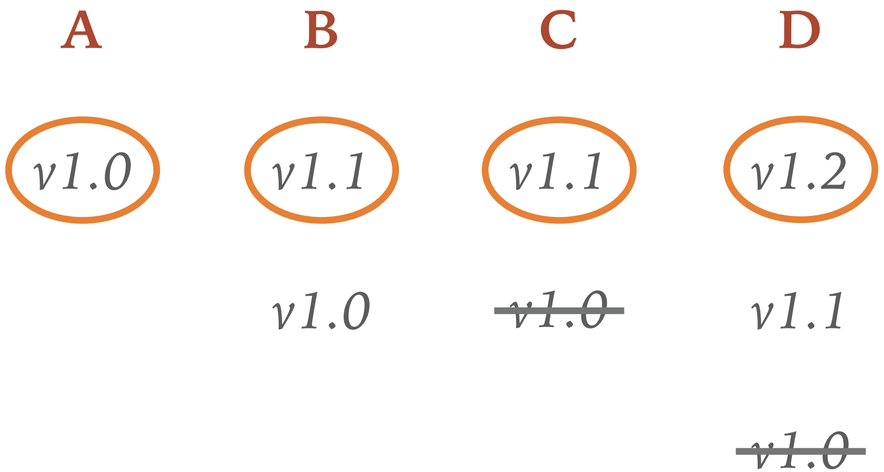
A	B	C	D
v1.0	*v1.1*	*v1.1*	*v1.2*
	v1.0	~~*v1.0*~~	*v1.1*
			~~*v1.0*~~

- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly

DEPENDENCY RESOLUTION

Simple Example

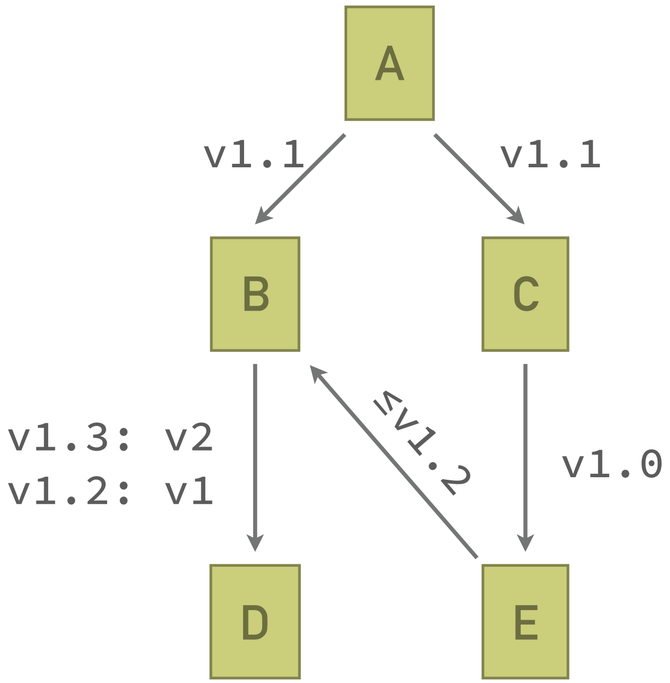




- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly

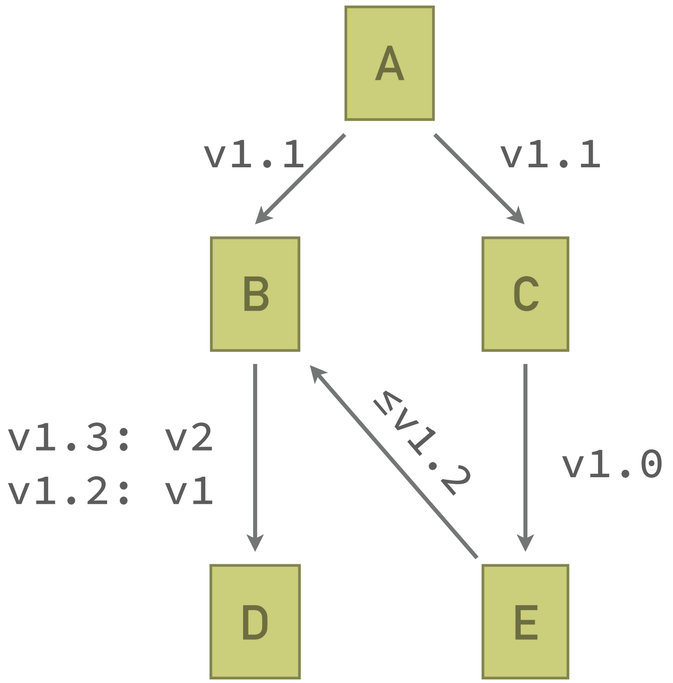
DEPENDENCY RESOLUTION

Simple Example



DEPENDENCY RESOLUTION

Simple Example

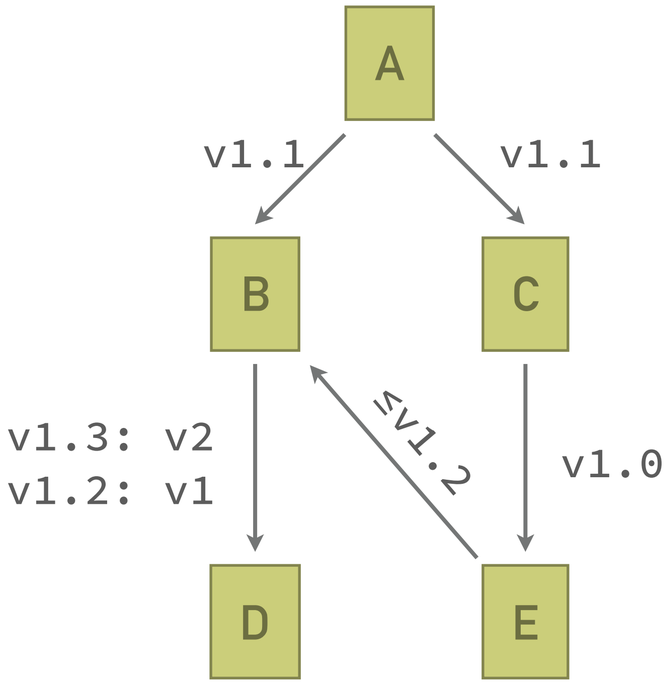


A

v1.0

DEPENDENCY RESOLUTION

Simple Example

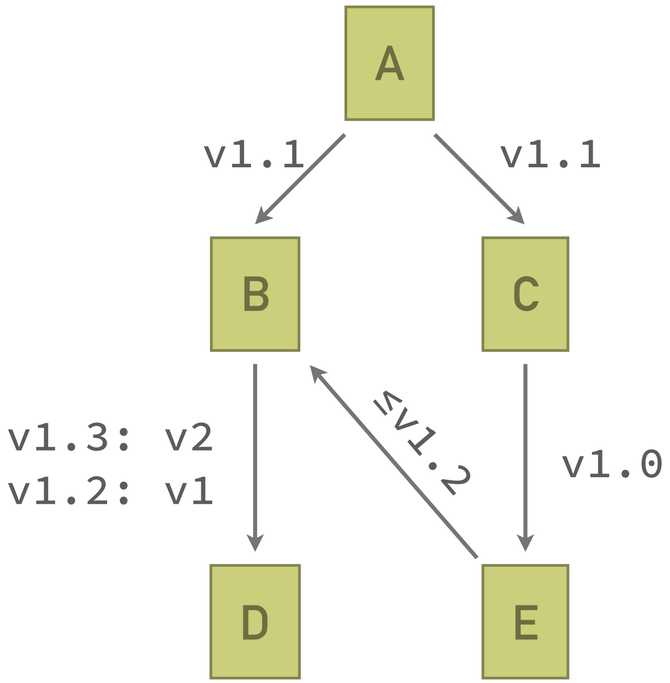


A

v1.0

DEPENDENCY RESOLUTION

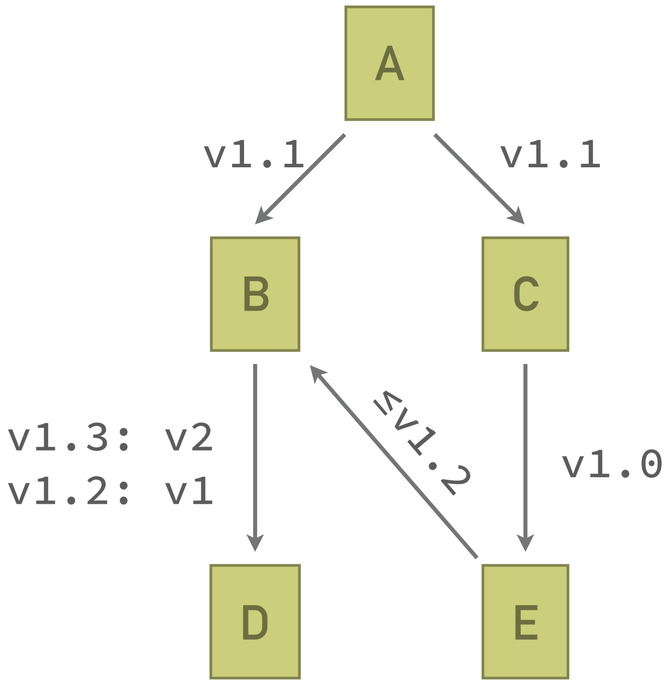
Simple Example



A	B	C
v1.0	*v1.3*	*v1.1*
	v1.2	*v1.0*
	v1.1	
	v1.0	

DEPENDENCY RESOLUTION

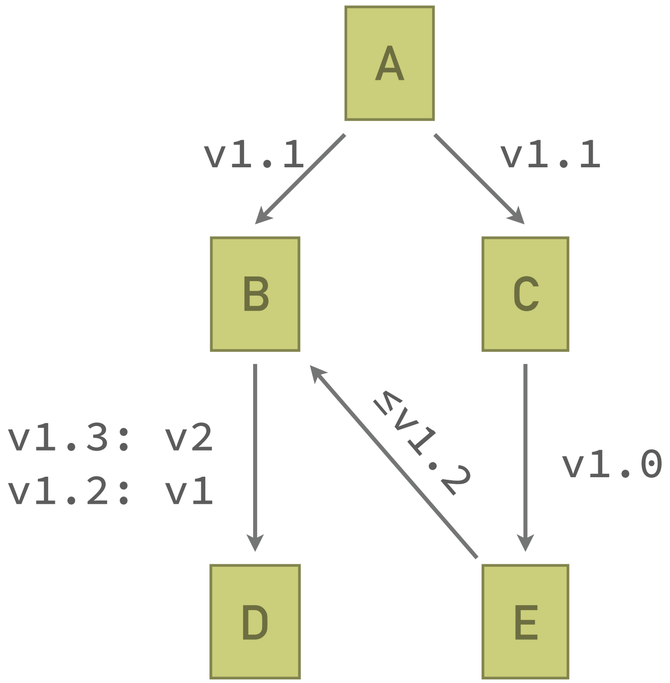
Simple Example



A	B	C
v1.0	*v1.3*	*v1.1*
	v1.2	~~*v1.0*~~
	v1.1	
	~~*v1.0*~~	

DEPENDENCY RESOLUTION

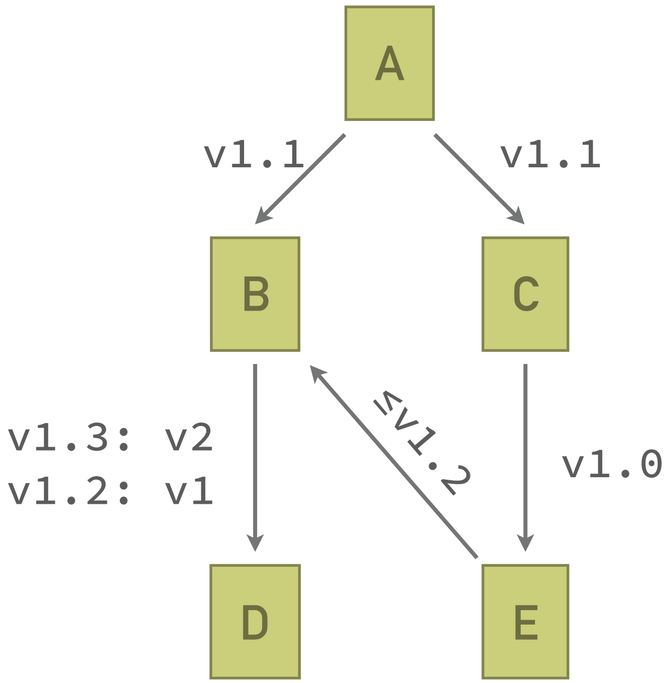
Simple Example



A	B	C
v1.0	*v1.3*	*v1.1*
	v1.2	~~*v1.0*~~
	v1.1	
	~~*v1.0*~~	

DEPENDENCY RESOLUTION

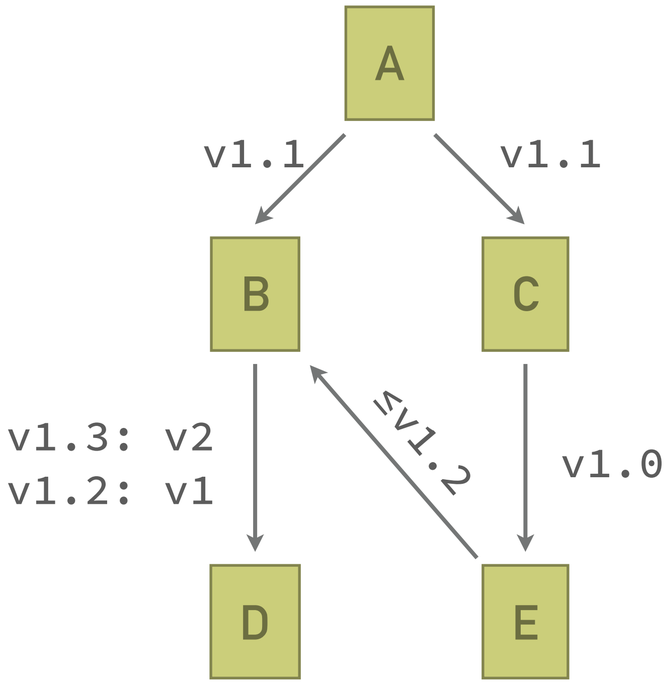
Simple Example



A	B	C	D	E
v1.0	*v1.3*	*v1.1*	*v2.0*	*v1.0*
	v1.2	~~*v1.0*~~	*v1.0*	
	v1.1			
	~~*v1.0*~~			

DEPENDENCY RESOLUTION

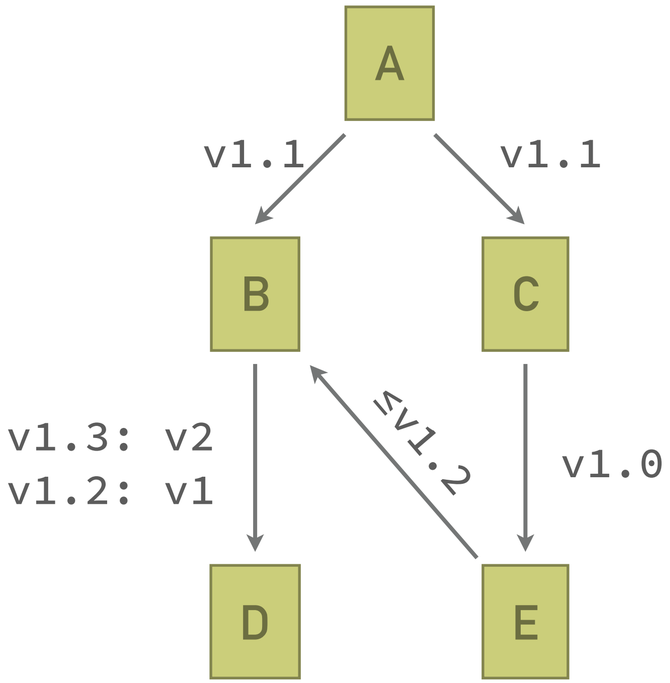
Simple Example



A	B	C	D	E
v1.0	*v1.3*	*v1.1*	*v2.0*	*v1.0*
	v1.2	~~*v1.0*~~	~~*v1.0*~~	
	v1.1			
	~~*v1.0*~~			

DEPENDENCY RESOLUTION

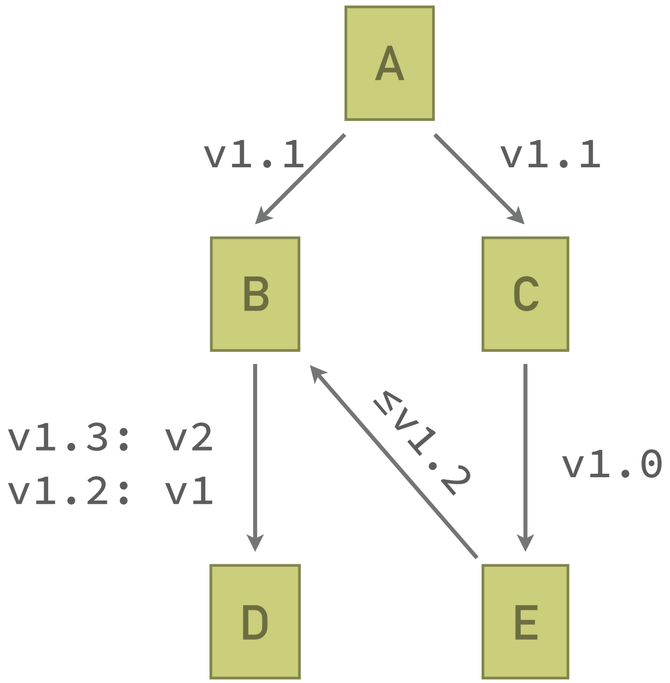
Simple Example



A	B	C	D	E
v1.0	*v1.3*	*v1.1*	*v2.0*	*v1.0*
	v1.2	~~*v1.0*~~	~~*v1.0*~~	
	v1.1			
	~~*v1.0*~~			

DEPENDENCY RESOLUTION

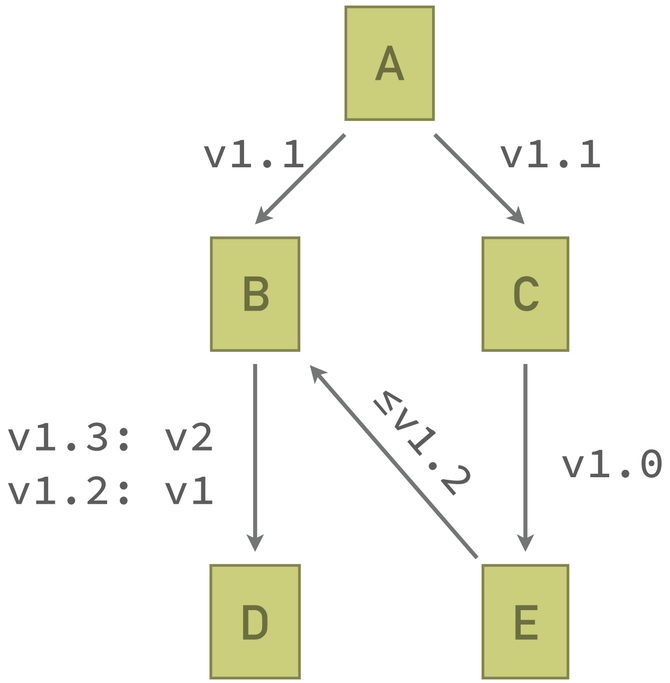
Simple Example



A	B	C	D	E
v1.0	~~*v1.3*~~	*v1.1*	*v2.0*	*v1.0*
	v1.2	~~*v1.0*~~	~~*v1.0*~~	
	v1.1			
	~~*v1.0*~~			

DEPENDENCY RESOLUTION

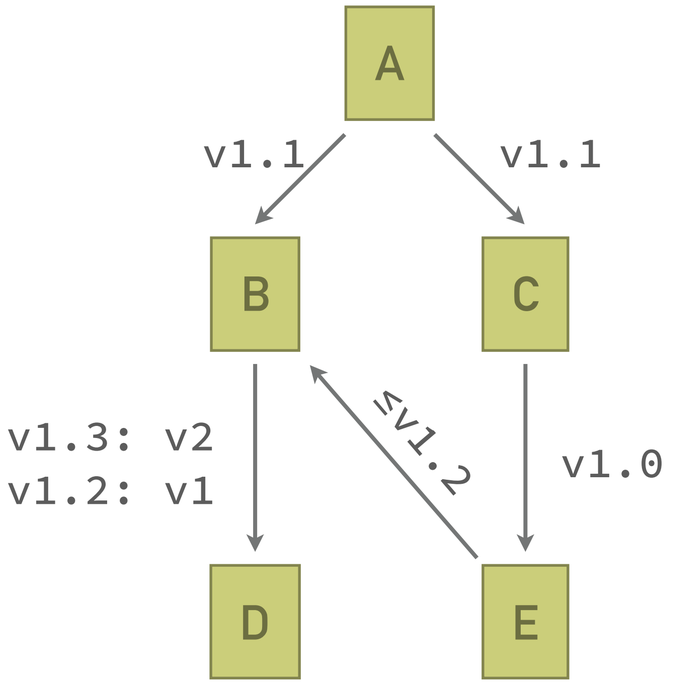
Simple Example



A	B	C	D	E
v1.0	~~*v1.3*~~	*v1.1*	*v2.0*	*v1.0*
	v1.2	~~*v1.0*~~	~~*v1.0*~~	
	v1.1			
	~~*v1.0*~~			

DEPENDENCY RESOLUTION

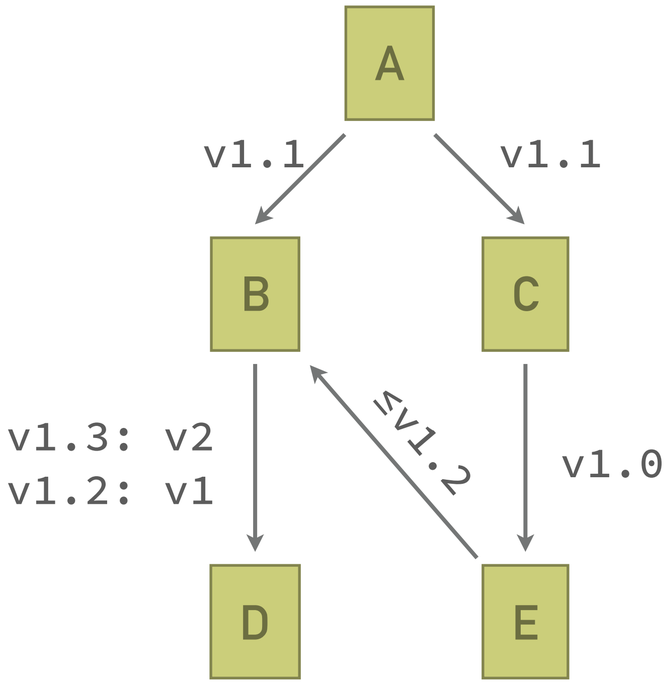
Simple Example



A	B	C	D	E
v1.0	~~*v1.3*~~	*v1.1*	*v2.0*	*v1.0*
	v1.2	~~*v1.0*~~	*v1.0*	
	v1.1			
	~~*v1.0*~~			

DEPENDENCY RESOLUTION

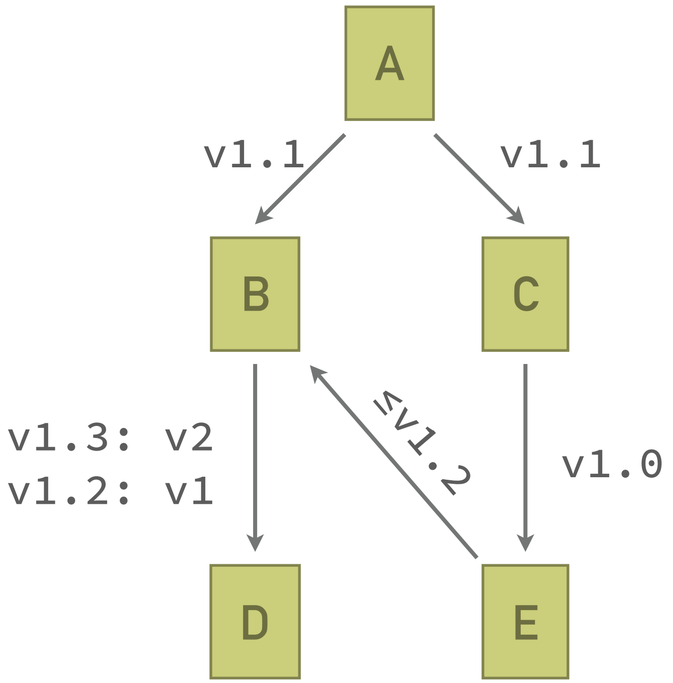
Simple Example



A	B	C	D	E
v1.0	~~*v1.3*~~	*v1.1*	*v2.0*	*v1.0*
	v1.2	~~*v1.0*~~	*v1.0*	
	v1.1			
	~~*v1.0*~~			

DEPENDENCY RESOLUTION

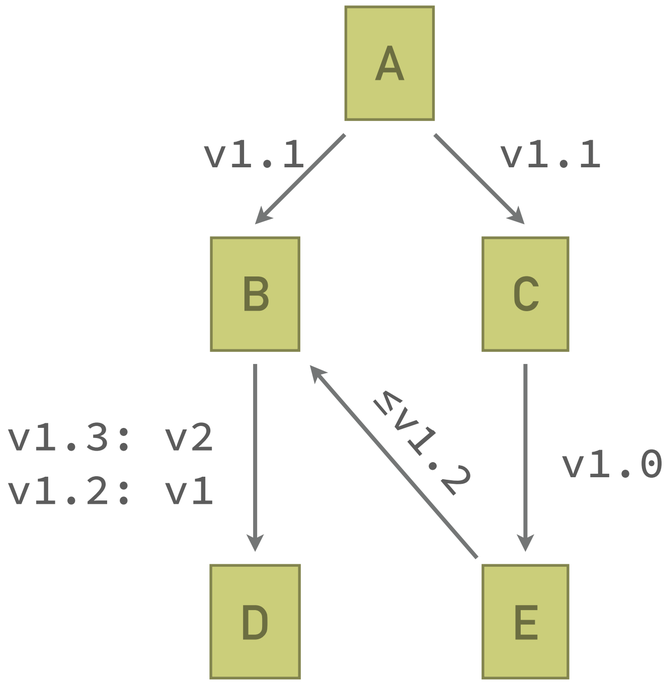
Simple Example



A	B	C	D	E
v1.0	~~*v1.3*~~	*v1.1*	~~*v2.0*~~	*v1.0*
	v1.2	~~*v1.0*~~	*v1.0*	
	v1.1			
	~~*v1.0*~~			

DEPENDENCY RESOLUTION

Simple Example



A	B	C	D	E
v1.0	~~*v1.3*~~	*v1.1*	~~*v2.0*~~	*v1.0*
	v1.2	~~*v1.0*~~	*v1.0*	
	v1.1			
	~~*v1.0*~~			

SOURCE FILES

- We have established dependency tracking as first tier

SOURCE FILES

- We have established dependency tracking as first tier
- Let’s track source files as well

```
# Bender.yml
dependencies:
  …
sources:
  - src/axi_pkg.sv
  - src/axi_intf.sv
  - src/axi2mem.sv
  - src/mem2axi.sv
```

SOURCE FILES

- We have established dependency tracking as first tier
- Let's track source files as well
- Allow for groups, include dirs, defines

```
# Bender.yml
dependencies:
  …
sources:
  - src/axi_pkg.sv
  - src/axi_intf.sv
  - src/axi2mem.sv
  - src/mem2axi.sv
```

```
# Bender.yml
…
sources:
  - src/axi_pkg.sv
  - include_dirs:
      - src/include
    defines:
      FPGA_EMUL: 1
      SKIP_TRACE: 0
    files:
      - src/axi_intf.sv
      - src/axi2mem.sv
      - src/mem2axi.sv
```

SOURCE FILES

- Each dependency declares its source files

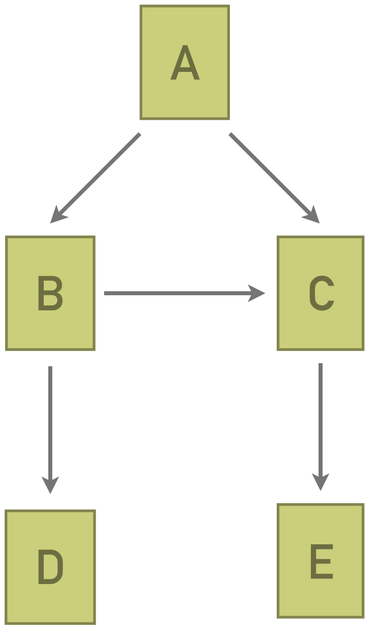
SOURCE FILES

Topological Ordering

- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order

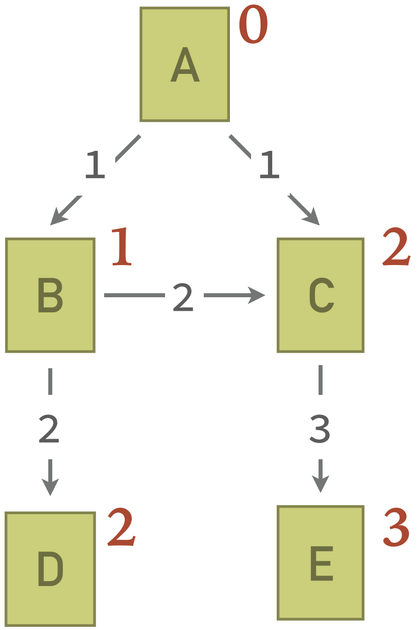
SOURCE FILES

- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order



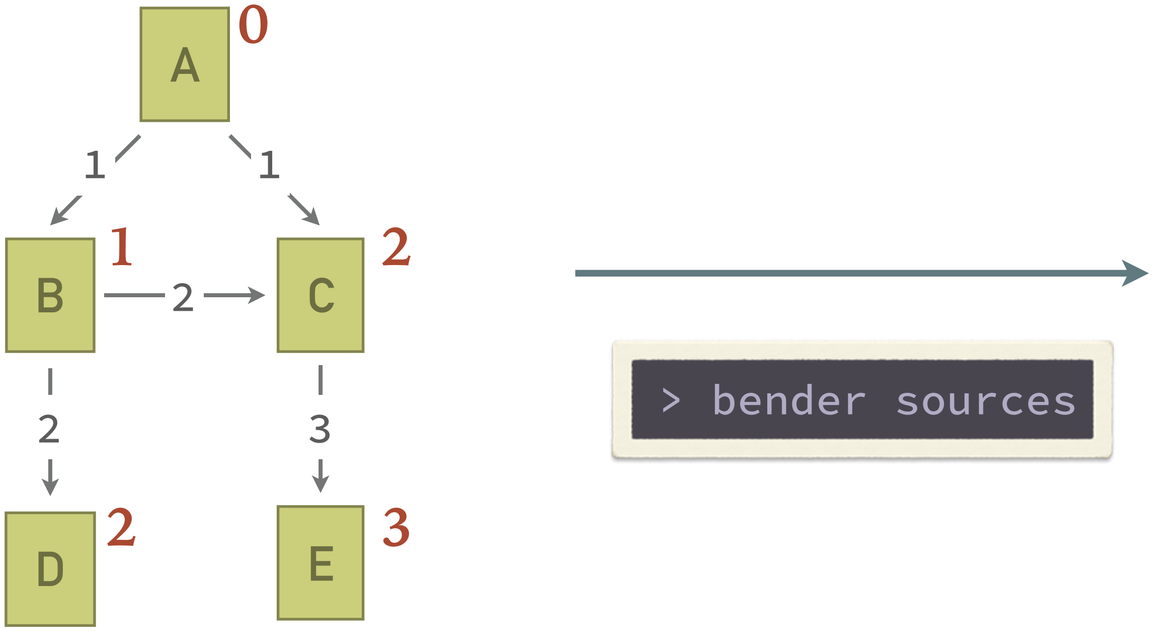
SOURCE FILES

- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order



SOURCE FILES

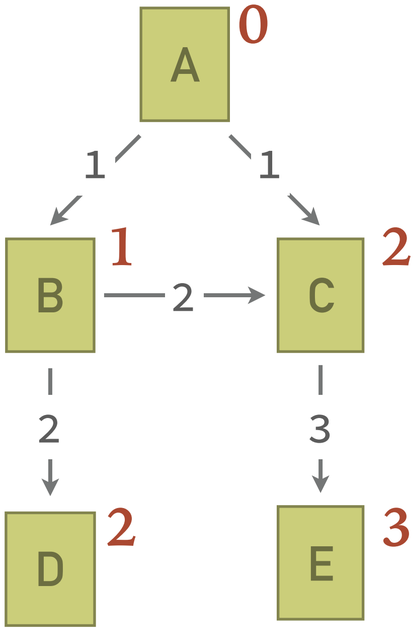
- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order




```
> bender sources
```

SOURCE FILES

- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order

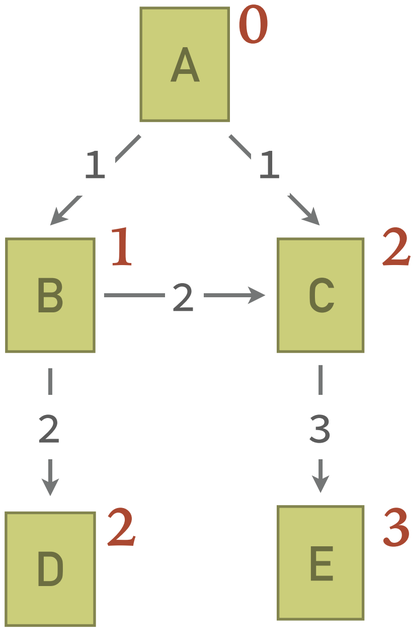



```
> bender sources
```

```
- E/top.sv
- E/pkg.sv
- C/foo.vhd
- C/bar.vhd
- D/ctrl.sv
- D/datapath.sv
- D/export.sv
- B/top.sv
- A/padframe.sv
- A/top.sv
```

SOURCE FILES

- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order



Tier 2

Don't want to do this by hand!

```
> bender sources
```

```
- E/top.sv
- E/pkg.sv
- C/foo.vhd
- C/bar.vhd
- D/ctrl.sv
- D/datapath.sv
- D/export.sv
- B/top.sv
- A/padframe.sv
- A/top.sv
```

TARGETS

- We have the same source files for …

TARGETS

- We have the same source files for …
- … different technologies:
  - ASIC (gf22, umc65, tsmc45, smic130, etc.)
  - FPGA (xilinx, altera)

TARGETS

- We have the same source files for …
- … different technologies:
  - ASIC (gf22, umc65, tsmc45, smic130, etc.)
  - FPGA (xilinx, altera)
- … different use cases:
  - RTL simulation
  - RTL synthesis
  - Post-synthesis simulation
  - Post-layout simulation
  - Linting

TARGETS

- We have the same source files for …
- … different technologies:
  - ASIC (gf22, umc65, tsmc45, smic130, etc.)
  - FPGA (xilinx, altera)
- … different use cases:
  - RTL simulation
  - RTL synthesis
  - Post-synthesis simulation
  - Post-layout simulation
  - Linting

```
sources:
  - src/queue.sv
  - target: fpga
    files:
      - src/fifo_fpga.sv
  - target: not(fpga)
    files:
      - src/fifo_generic.sv
```

TARGETS

- We have the same source files for …
- … different technologies:
  - ASIC (gf22, umc65, tsmc45, smic130, etc.)
  - FPGA (xilinx, altera)
- … different use cases:
  - RTL simulation
  - RTL synthesis
  - Post-synthesis simulation
  - Post-layout simulation
  - Linting

```
sources:
  - src/queue.sv
  - target: fpga
    files:
      - src/fifo_fpga.sv
  - target: not(fpga)
    files:
      - src/fifo_generic.sv
```

Target Syntax:

```
Names:   fpga, asic, umc65
AND:     all(fpga, xilinx)
OR:      any(fpga, asic)
NOT:     not(fpga)
```

EXAMPLE

- Let’s make a simple package without dependencies:

common_cells

```
# Bender.yml
package:
  name: common_cells
  author: [“John Doe <john@doe.com>"]

sources:
  - src/generic_fifo.sv
  - src/round_robin.sv
  - src/leading_zero.sv
```

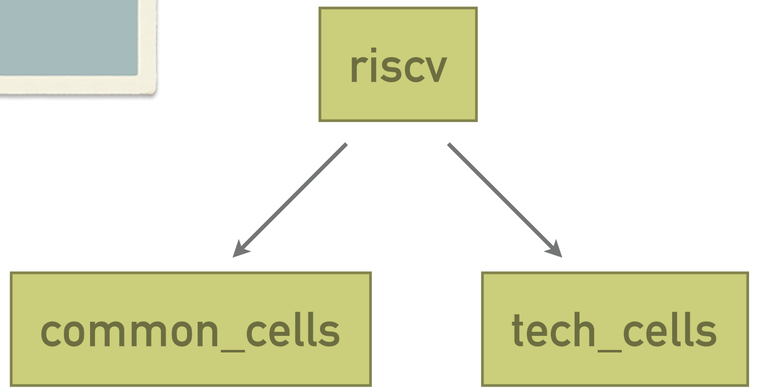
```
> tree
Bender.yml
LICENSE
README.md
src/
  generic_fifo.sv
  round_robin.sv
  leading_zero.sv
```

EXAMPLE

- A RISC-V core that depends on a few other repositories:

```
# Bender.yml
package:
  name: riscv
  author: [“John Doe <john@doe.com>"]
dependencies:
  common_cells: { git: “…/common_cells.git”, version: 1.0.2 }
  tech_cells: { git: “…/tech_cells.git”, version: 0.5.3 }
sources:
  - src/riscv_core.sv
  - src/riscv_ctrl.sv
```

```
> tree
Bender.yml
LICENSE
README.md
src/
  riscv_core.sv
  riscv_ctrl.sv
```

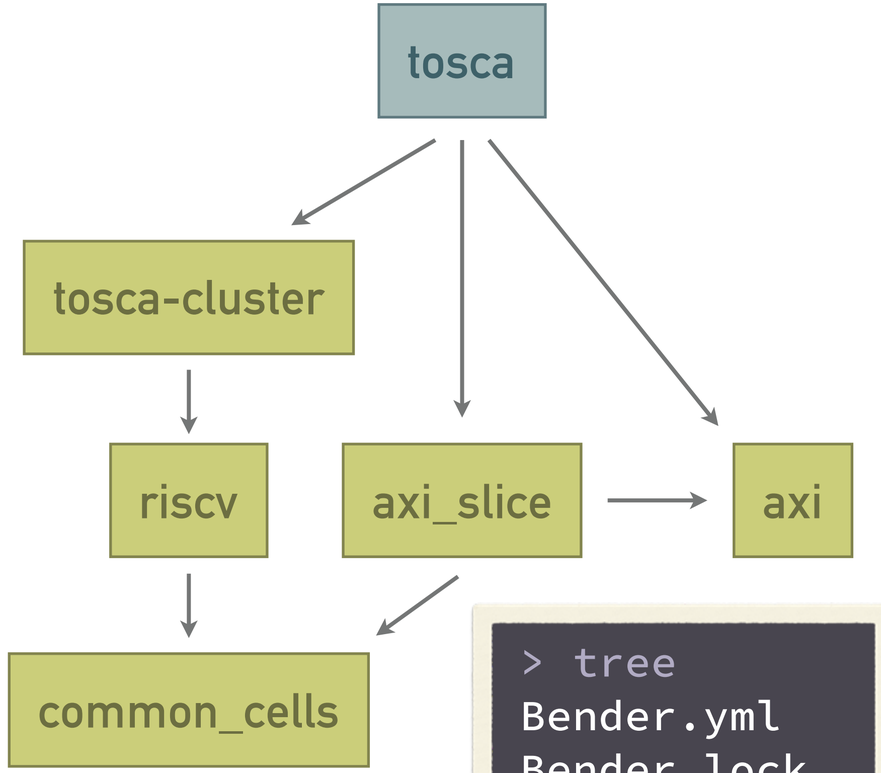


EXAMPLE

An entire chip

- A chip repository that will be taped out
  - Before: Put *Bender.lock* in .gitignore
  - Here: **Commit** *Bender.lock* to get **reproducible** builds!

```
# Bender.yml
package:
  name: tosca
  author: [“John Doe <john@doe.com>"]
dependencies:
  tosca-cluster: { git: … }
  axi: { git: … }
  axi_slice: { git: … }
sources:
  - src/top.sv
  - src/padframe.sv
```




```
> tree
Bender.yml
Bender.lock
LICENSE
README.md
src/
  top.sv
  padframe.sv
```

FEEDING THE TOOLS

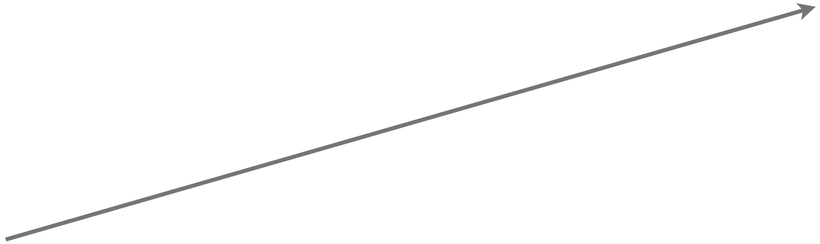
- We have all source files for an entire dependency graph.

FEEDING THE TOOLS

- We have all source files for an entire dependency graph.
- We need to feed many tools (different from software):
  - vsim
  - ncsim
  - synopsys
  - genus
  - spyglass
  - vivado
  - quartus

FEEDING THE TOOLS

- We have all source files for an entire dependency graph.
- We need to feed many tools (different from software):
  - vsim
  - ncsim
  - synopsys
  - genus
  - spyglass
  - vivado
  - quartus
- Can be done manually



```
> bender sources
{
  “include_dirs”: […],
  “defines”: {…},
  “files”: […],
}
```

FEEDING THE TOOLS

- We have all source files for an entire dependency graph.
- We need to feed many tools (different from software):
  - vsim
  - ncsim
  - synopsys
  - genus
  - spyglass
  - vivado
  - quartus
- Can be done manually
- Or have Bender do it for you…

```
> bender sources
{
  “include_dirs”: […],
  “defines”: {…},
  “files”: […],
}
```

SCRIPT GENERATION

SCRIPT GENERATION

- Bender can maintain tool scripts for you

SCRIPT GENERATION

- Bender can maintain tool scripts for you
- Currently supported targets:
  - Synopsys Design Compiler “analyze” scripts
  - QuestaSim compile scripts

SCRIPT GENERATION

- Bender can maintain tool scripts for you
- Currently supported targets:
  - Synopsys Design Compiler “analyze” scripts
  - QuestaSim compile scripts

```
> bender script vsim > compile.tcl
```

```
# compile.tcl
vlog +define+TARGET_VSIM \
    “src/riscv_top.sv” \
    …
```

SCRIPT GENERATION

- Bender can maintain tool scripts for you
- Currently supported targets:
  - Synopsys Design Compiler “analyze” scripts
  - QuestaSim compile scripts

```
> bender script synopsys > analyze.tcl
```

```
# analyze.tcl
lappend search_path "src/include"
analyze -format sv -define { \
    TARGET_SYNOPSYS \
    TARGET_SYNTHESIS \
} [list \
    “src/riscv_top.sv” \
]
```

```
> bender script vsim > compile.tcl
```

```
# compile.tcl
vlog +define+TARGET_VSIM \
    “src/riscv_top.sv” \
    …
```

SCRIPT GENERATION

- Bender can maintain tool scripts for you
- Currently supported targets:
  - Synopsys Design Compiler “analyze” scripts
  - QuestaSim compile scripts
- Experimental support for edalize

```
> bender script synopsys > analyze.tcl
```

```
# analyze.tcl
lappend search_path "src/include"
analyze -format sv -define { \
    TARGET_SYNOPSYS \
    TARGET_SYNTHESIS \
} [list \
    “src/riscv_top.sv” \
]
```

```
> bender script vsim > compile.tcl
```

```
# compile.tcl
vlog +define+TARGET_VSIM \
    “src/riscv_top.sv” \
    …
```

SCRIPT GENERATION

- Bender can maintain tool scripts for you
- Currently supported targets:
  - Synopsys Design Compiler “analyze” scripts
  - QuestaSim compile scripts
- Experimental support for edalize
- Scripts can be checked into version control
  - Takes bender out of the EDA loop
  - Can share environment with collaborators that do not have bender installed

Opt-in!

```
> bender script synopsys > analyze.tcl
```

```
# analyze.tcl
lappend search_path "src/include"
analyze -format sv -define { \
    TARGET_SYNOPSYS \
    TARGET_SYNTHESIS \
} [list \
    “src/riscv_top.sv” \
]
```

```
> bender script vsim > compile.tcl
```

```
# compile.tcl
vlog +define+TARGET_VSIM \
    “src/riscv_top.sv” \
    …
```

PLUGINS

- Plugins allow Bender to be extended easily with custom commands
- A regular dependency
- Offers commands to the user:
  - can be simple scripts
  - can be entire executables

PLUGINS

- Plugins allow Bender to be extended easily with custom commands
- A regular dependency
- Offers commands to the user:
  - can be simple scripts
  - can be entire executables

```
# Bender.yml
package:
  name: bender-vsim
  author: [“John Doe <john@doe.com>"]

plugins:
  vsim: “do_stuff.sh”
```

PLUGINS

- Plugins allow Bender to be extended easily with custom commands
- A regular dependency
- Offers commands to the user:
  - can be simple scripts
  - can be entire executables

```
# Bender.yml
package:
  name: bender-vsim
  author: [“John Doe <john@doe.com>"]

plugins:
  vsim: “do_stuff.sh”
```

```
#!/bin/bash
# do_stuff.sh

SOURCES=`$BENDER sources`
for FILE in $SOURCES; do
  vlog-10.6b $FILE
done

echo “run -all” | vsim-10.6b -c
```

TESTING

One-button testing

TESTING

One-button testing

- I would like testing to be one button away

```
> bender test --all
All 4 tests passed.
```

TESTING

One-button testing

- I would like testing to be one button away
- Can be implemented as another plugin:
  1. Compilation tests with installed tools
     - ➤ vsim/ncsim
     - ➤ synopsys/genus
     - ➤ spyglass/verilator
     - ➤ vivado
  2. Run unit/regression tests
     - ➤ vsim/ncsim

```
> bender test --all
All 4 tests passed.
```

TESTING

One-button testing

- I would like testing to be one button away
- Can be implemented as another plugin:
  1. Compilation tests with installed tools
     - ➤ vsim/ncsim
     - ➤ synopsys/genus
     - ➤ spyglass/verilator
     - ➤ vivado
  2. Run unit/regression tests
     - ➤ vsim/ncsim

Ensures that IP is compatible with different tools.

```
> bender test --all
All 4 tests passed.
```

TESTING

One-button testing

- I would like testing to be one button away
- Can be implemented as another plugin:
  1. Compilation tests with installed tools
     - ➤ vsim/ncsim
     - ➤ synopsys/genus
     - ➤ spyglass/verilator
     - ➤ vivado
  2. Run unit/regression tests
     - ➤ vsim/ncsim
- Can be easily integrated into CI

Ensures that IP is compatible with different tools.

```
> bender test --all
All 4 tests passed.
```

TESTING

One-button testing

- I would like testing to be one button away
- Can be implemented as another plugin:
  1. Compilation tests with installed tools
     - ➤ vsim/ncsim
     - ➤ synopsys/genus
     - ➤ spyglass/verilator
     - ➤ vivado
  2. Run unit/regression tests
     - ➤ vsim/ncsim
- Can be easily integrated into CI

Ensures that IP is compatible with different tools.

```
> bender test --all
All 4 tests passed.
```

Future!

```
# Bender.yml
package:
  name: bender-vsim
  author: [“John Doe <john@doe.com>"]

test:
  compile: [vsim, vivado, synopsys]
  benches:
    - test/tb_one.sv
    - test/tb_two.sv
  cases:
    a: [tb_one, NUM_MASTER=[1,2,3]],
    b: [tb_two, NUM_SLAVE=[3,4,9]],
```

REGISTRY

for convenience and open source releases

REGISTRY

for convenience and open source releases

- Makes it easy to find existing IPs (“Has anyone created a protocol adapter?”)

REGISTRY

for convenience and open source releases

- Makes it easy to find existing IPs (“Has anyone created a protocol adapter?”)
- Typing Git URLs for dependencies is tedious and error prone:


```
common_cells: { git: “…/common_cells.git”, version: 1.0.2 }
tech_cells:   { git: “…/tech_cells.git”,   version: 0.5.3 }
axi:          { git: “…/axi.git”,          version: 0.2   }
```

REGISTRY

for convenience and open source releases

- Makes it easy to find existing IPs (“Has anyone created a protocol adapter?”)
- Typing Git URLs for dependencies is tedious and error prone:

```
common_cells: { git: “…/common_cells.git”, version: 1.0.2 }
tech_cells:   { git: “…/tech_cells.git”,   version: 0.5.3 }
axi:          { git: “…/axi.git”,          version: 0.2   }
```

- Solution: Create a registry!
  - Simply a file on a web server which lists Git repositories
  - Can have multiple registries (pulp-restricted vs. pulp-open)

```
common_cells: 1.0.2
tech_cells:   0.5.3
axi:          0.2
```

REGISTRY

for convenience and open source releases

- Makes it easy to find existing IPs (“Has anyone created a protocol adapter?”)
- Typing Git URLs for dependencies is tedious and error prone:

```
common_cells: { git: “…/common_cells.git”, version: 1.0.2 }
tech_cells:   { git: “…/tech_cells.git”,   version: 0.5.3 }
axi:          { git: “…/axi.git”,          version: 0.2   }
```

- Solution: Create a registry!
  - Simply a file on a web server which lists Git repositories
  - Can have multiple registries (pulp-restricted vs. pulp-open)

```
common_cells: 1.0.2
tech_cells:   0.5.3
axi:          0.2
```

- Helps with open-source releases

I HATE YOUR TOOL!

I HATE YOUR TOOL!

Alternatives

- Bazel

{Fast, Correct} - Choose two

I HATE YOUR TOOL!

{Fast, Correct} - Choose two

- Bazel
- The award-winning FuseSoC

I HATE YOUR TOOL!

{Fast, Correct} - Choose two

- Bazel
- The award-winning FuseSoC
- npm?

I HATE YOUR TOOL!

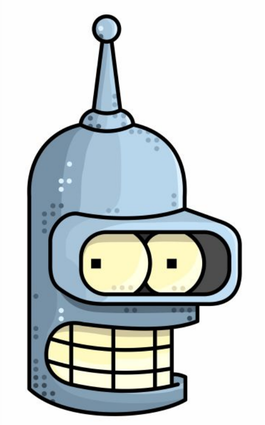
{Fast, Correct} - Choose two

- Bazel
- The award-winning FuseSoC
- npm?
- others?

FUTURE WORK

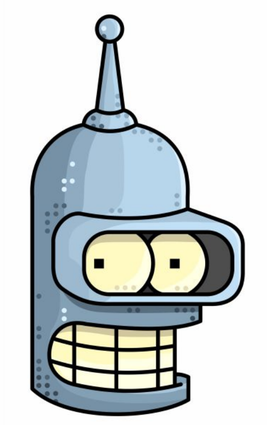
- Integration with FuseSoC/edalize? 🙂
- Add support for more tools
- Features
- Automation/conventions for unit tests

CONCLUSION



*Bender is here to help **you**!*

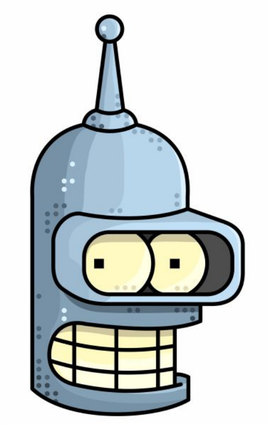
CONCLUSION



*Bender is here to help **you**!*

1. Transitive dependency resolution

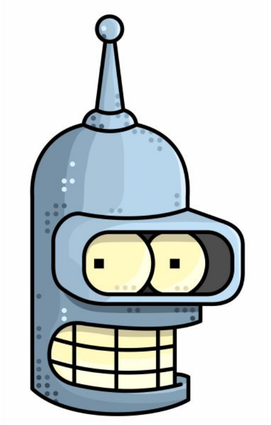
CONCLUSION



Bender is here to help you!

1. Transitive dependency resolution
2. Source file ordering and management

CONCLUSION



Bender is here to help you!

1. Transitive dependency resolution
2. Source file ordering and management
3. Registry and feeding the tools

Thanks!

https://github.com/fabianschuiki/bender

— and —

```
> cargo install bender
```

— and —

```
> git clone <url> bender
> cd bender
> cargo install
```