

MANAGING CONFIGURATIONS EXPLOSION 🌟

WHOAMI

THE SPEAKER



GUILLAUME MAUDOUX

@layus, in most places

Build systems engineer, working on Bazel and Nix.

TWEAG
by Modus Create

We maintain `rules_nixpkgs` , `rules_haskell` and more,
in our Bazel consultancy team,
part of the Scalable Builds group
at Tweag, the software innovation lab.

Come visit our booth!

THE CLIENT PROJECT

INTUITIVE SURGICAL ROBOT

- + Programmed from a large Bazel repository
- + Many embedded boards, all different:
 - + its chip, of various makes and models
 - + its purpose, and the IO devices around it
 - + its configuration and capabilities

Each board is a variant to build for. All these variants are exactly the problem this talk tackles.



CONTEXT

ASSUMPTIONS AND DISCLAIMER

- + We assume C and C++ code, built with `cc_library` and `cc_binary`
- + All names in this talk are AI generated: targets, boards, config settings
- + None of them are real, but the build structure is
- + The current implementation at Intuitive is the work of Nikhil Kalige

The examples are made up; the problems and the solutions are not.

THE PROBLEM

THE PROBLEM

CONFIG SETTINGS DESCRIBE THE TARGET

What each board needs is encoded in config settings:

- + **Target platform**, which picks the toolchain: `--platforms=//platforms:cortex_m7`
- + **Chip**, its family and model: `--//hw:chip=stm32h7`
- + **Operating system**, bare metal, RTOS or Linux: `--//hw:os=freertos`
- + **Board layout**, which devices are wired where: `--//boards:board=joint`
- + ...and more: build mode, features, debug options

Each of these is a dimension. Together, they span the configuration space.

THE PROBLEM

EVERY CONFIGURATION IS A NEW OUTPUT TREE

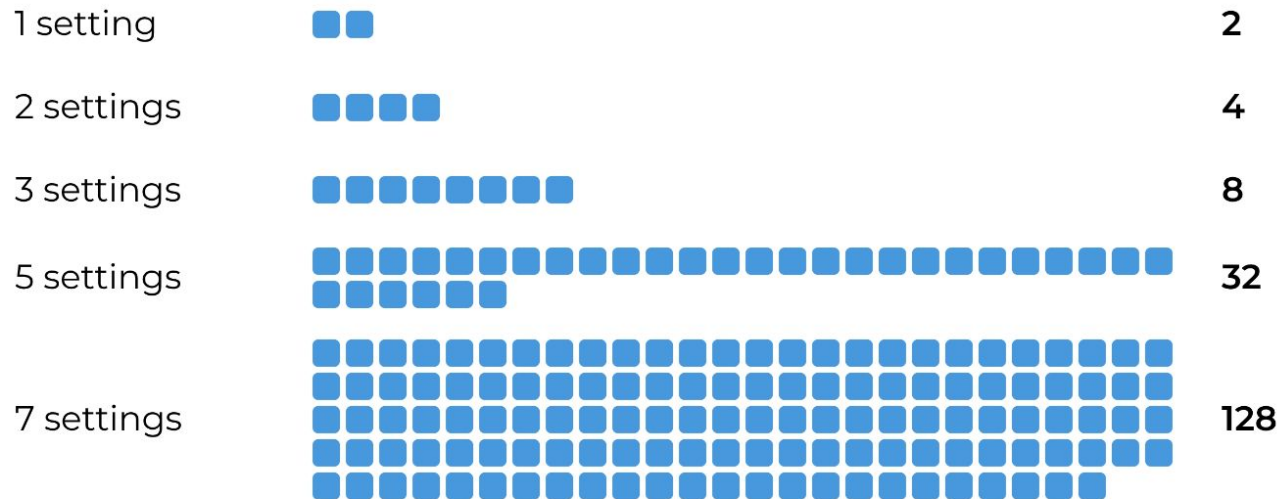
```
bazel-out/  
├─ k8-fastbuild/bin/lib/sensors.o  
├─ k8-fastbuild-ST-3f1a9c2d/bin/lib/sensors.o      # board=joint  
├─ k8-fastbuild-ST-b72e04aa/bin/lib/sensors.o      # board=instrument  
├─ k8-fastbuild-ST-5e81c0f3/bin/lib/sensors.o      # board=joint, os=freertos  
├─ k8-fastbuild-ST-91d4a7b2/bin/lib/sensors.o      # board=instrument, os=freertos  
├─ k8-dbg-ST-3f1a9c2d/bin/lib/sensors.o            # board=joint, debug  
├─ k8-fastbuild-ST-c07e2d19/bin/lib/sensors.o      # board=joint, os=linux: no such board!  
└─ ...
```

- + Each configuration gets its own output tree, where everything can be built again
- + Some of these files are identical: sometimes the only difference is the configuration hash, not the sources, not even the toolchain
- + Some combinations should not exist at all...
- + ...but once a config setting exists, it is hard to restrict where it applies

How to ensure there are no useless builds, and no builds in incorrect combinations?

THE PROBLEM

CONFIG SETTINGS CREATE A POWERSET OF CONFIGURATIONS



n boolean config settings: 2^n possible configurations.

THE PROBLEM, MEASURED

IN REAL NUMBERS

>12

string config
settings

>72M

possible
combinations
at least 72,726,528

856

configurations in
the graph

57

target platforms

39

registered C++
toolchains

856 configurations in the graph, but only 39 ways to compile a given source code.

OUR SETUP

THE BUILD

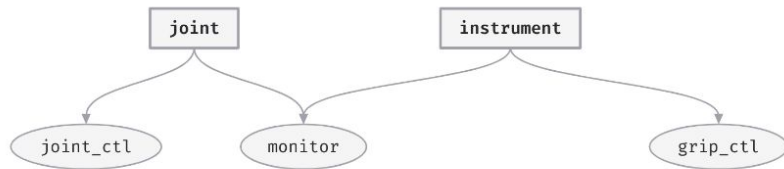
joint

instrument

Images, flashed on boards

OUR SETUP

THE BUILD

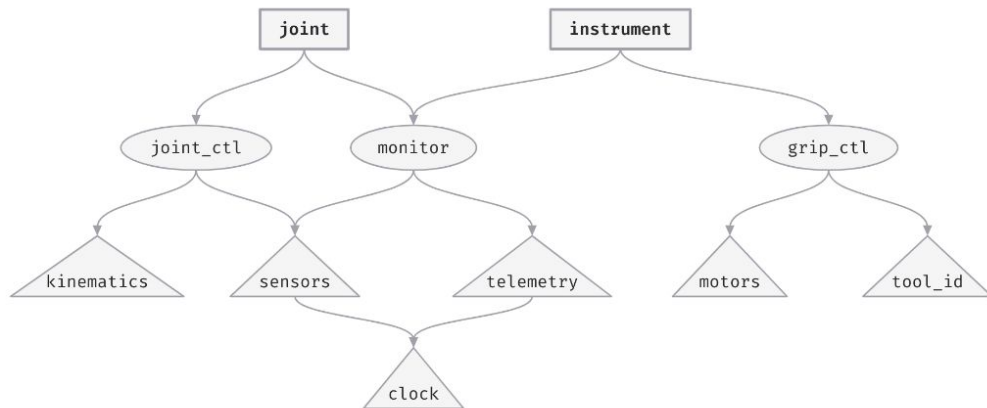


Images, flashed on boards

Apps, running on boards

OUR SETUP

THE BUILD



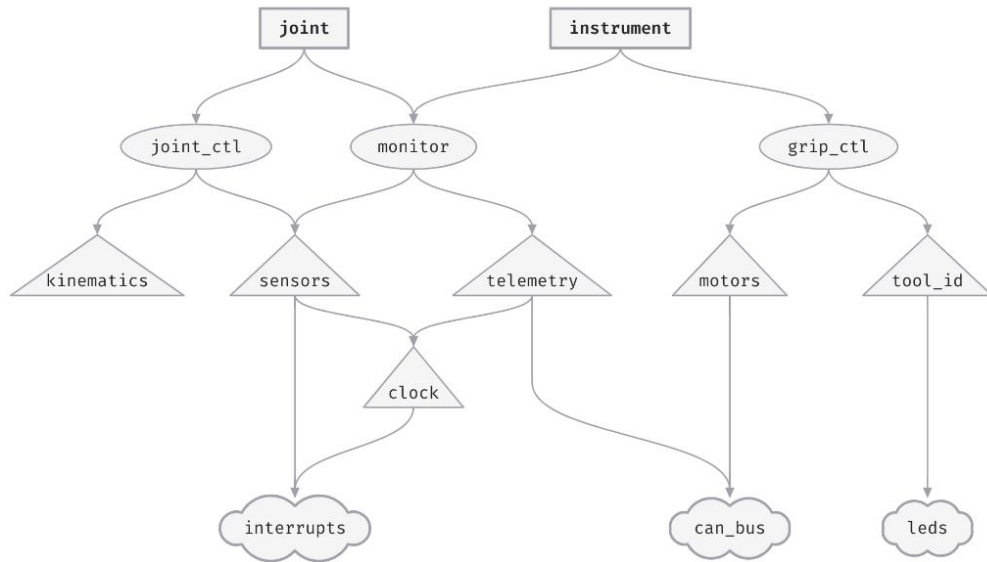
Images, flashed on boards

Apps, running on boards

Libs, shared across the build

OUR SETUP

THE BUILD



Images, flashed on boards

Apps, running on boards

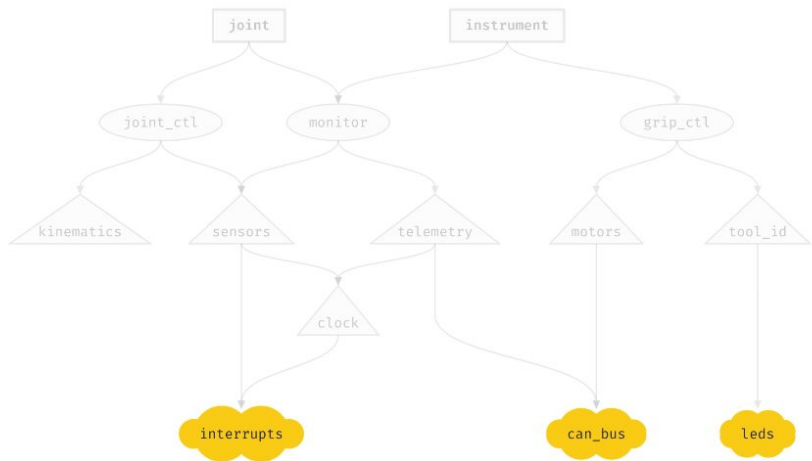
Libs, shared across the build

Hardware abstraction layer
One interface, multiple implementations

OUR SETUP

HARDWARE ABSTRACTION

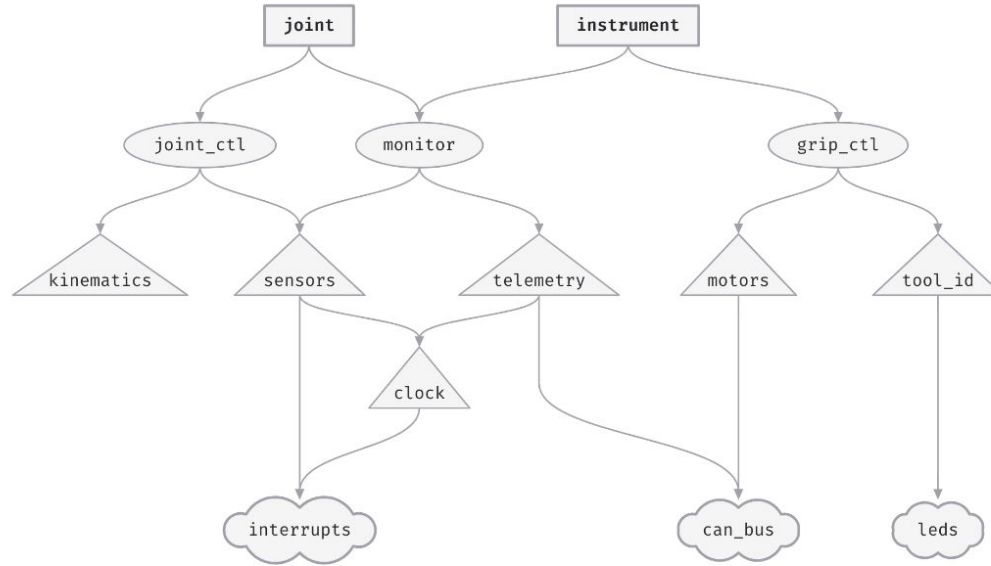
```
cc_library(  
    name = "interrupts",  
    # One interface  
    hdrs = ["interrupts.h"],  
    # One implementation per board  
    deps = select({  
        "//boards:joint":  
            ["//interrupts/joint"],  
        "//boards:instrument":  
            ["//interrupts/instrument"],  
    }),  
)
```



- + interrupts: `irq_enable(line)`, `irq_set_handler(line, handler)`
- + can_bus: `can_send(id, frame)`, `can_receive(...)`
- + leds: `led_set(color)`
- + Same calls everywhere, a different implementation per board

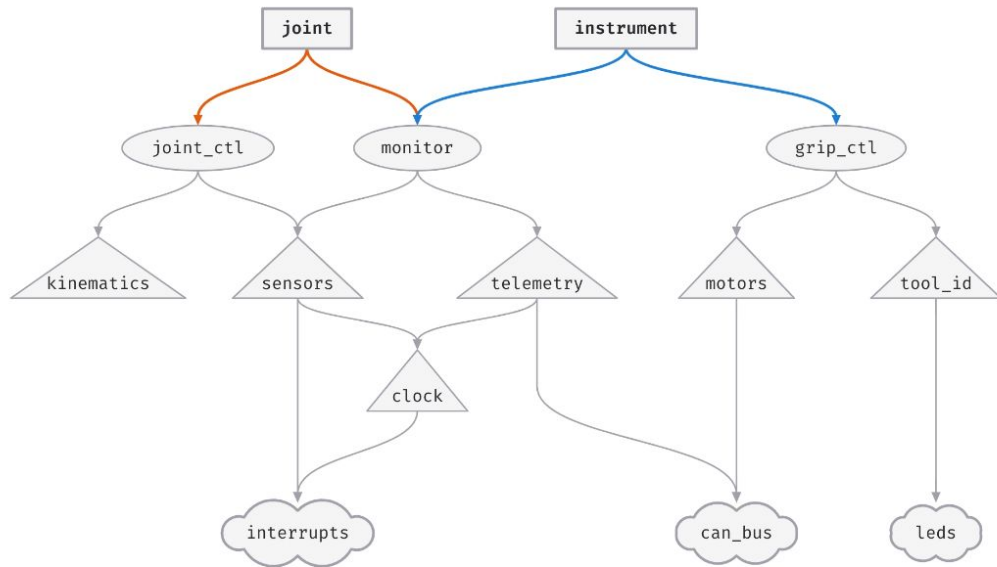
THE PROBLEM, ILLUSTRATED

EACH IMAGE BUILDS IN ITS OWN CONFIGURATION



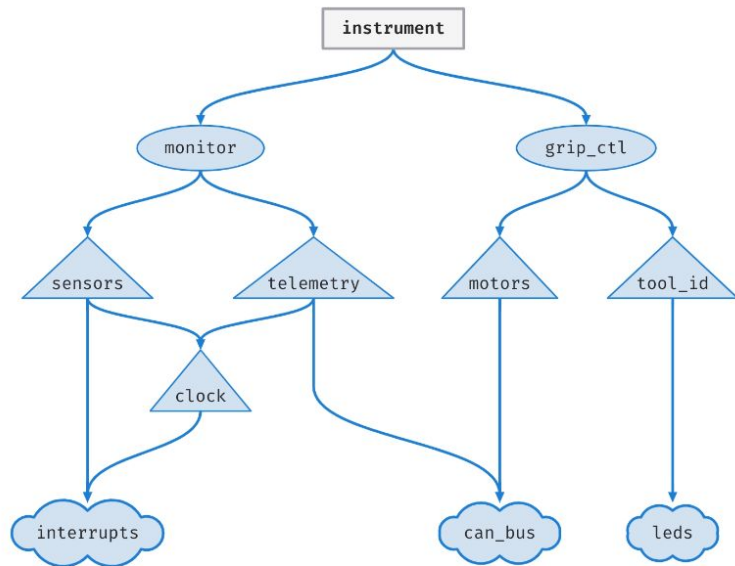
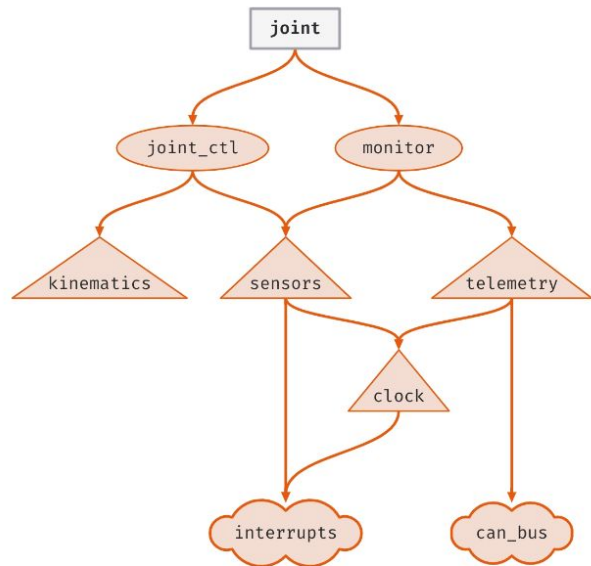
THE PROBLEM, ILLUSTRATED

EACH IMAGE BUILDS IN ITS OWN CONFIGURATION



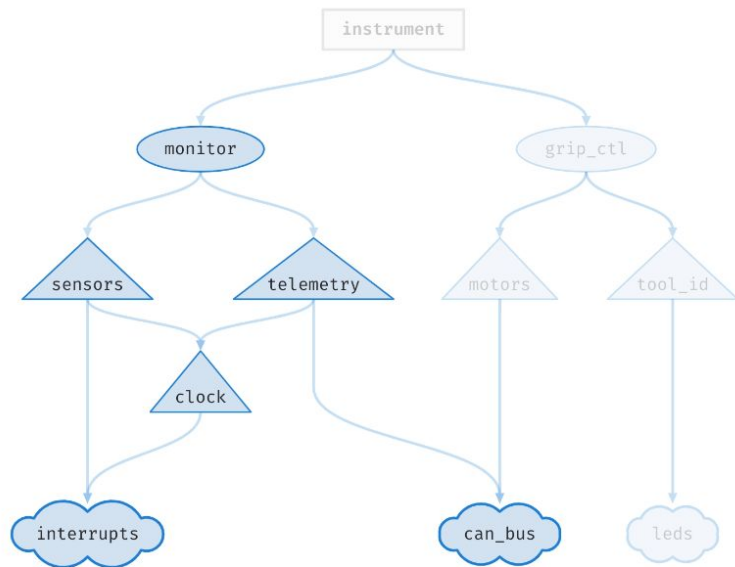
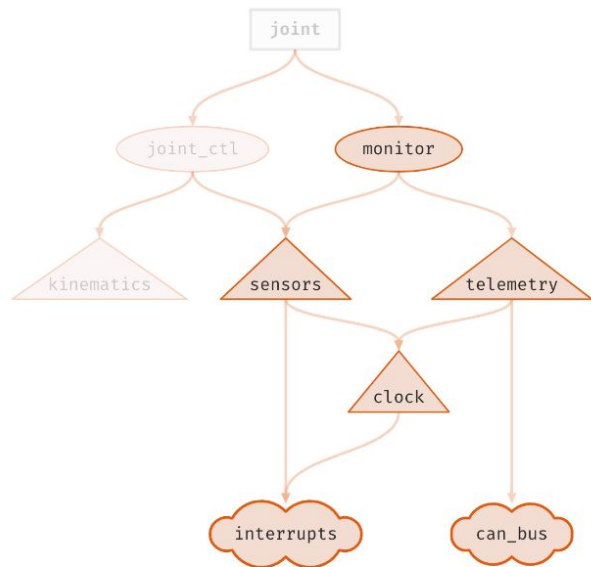
THE PROBLEM, ILLUSTRATED

EACH IMAGE BUILDS IN ITS OWN CONFIGURATION



THE PROBLEM, ILLUSTRATED

EACH IMAGE BUILDS IN ITS OWN CONFIGURATION



14 targets, 20 configured targets: everything shared is built twice, with no real difference.

AGENDA

THREE WAYS TO TAME CONFIGURATIONS



1. RESET TRANSITIONS

Drop the settings a subgraph does not use



2. PATH MAPPING

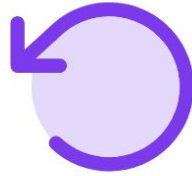
Let identical actions share a cache entry



3. FLAG SCOPES

Limit where a flag applies

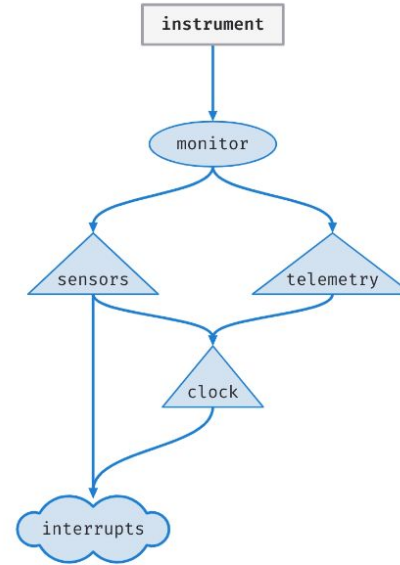
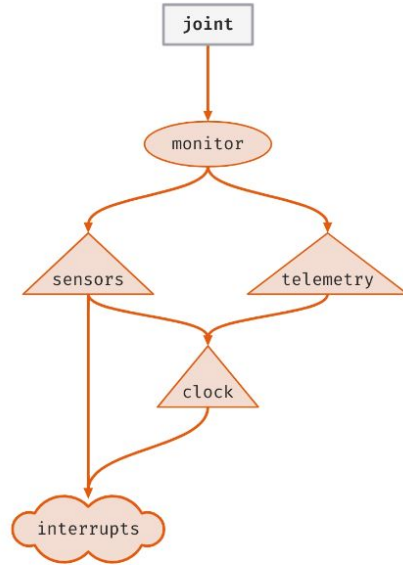
...and a wish for what could come next.



1. RESET TRANSITIONS

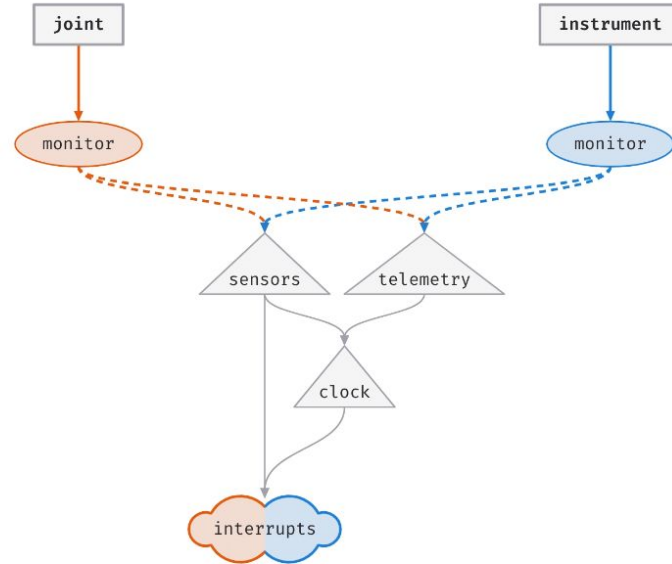
THE GOAL

WHAT WE WANT



THE GOAL

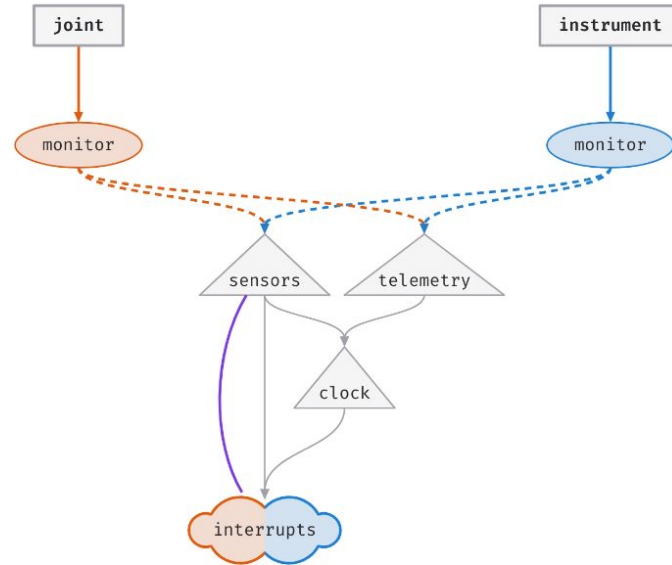
WHAT WE WANT



Reset the configuration of the libraries, so each is built once...

THE GOAL

WHAT WE WANT



Reset the configuration of the libraries, so each is built once... but the hardware abstraction is below them, and needs the board.

THE SOLUTION

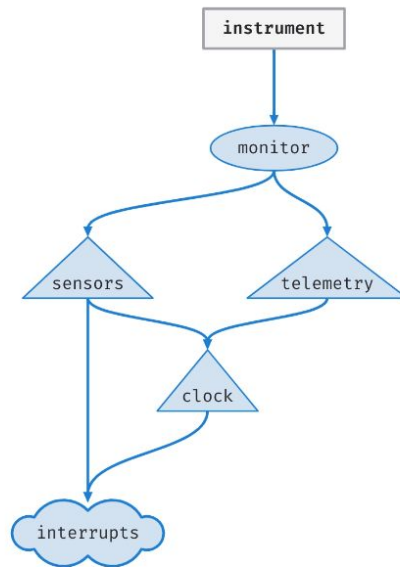
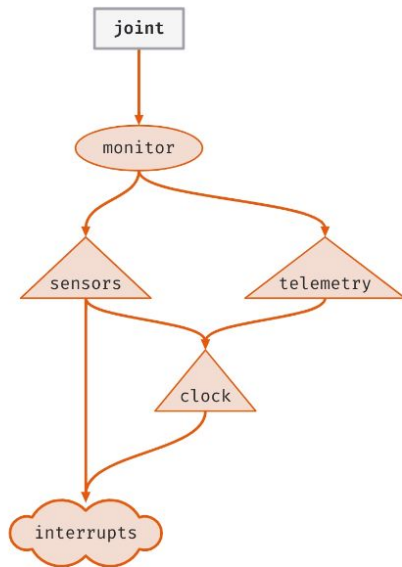
SPLIT THE INTERFACE FROM THE IMPLEMENTATION

```
def _ha_lib_impl(ctx):  
    return [  
        CcInfo(compilation_context = headers), # what the libraries compile against  
        HardwareInfo(lib = "interrupts"),      # what the application must link  
    ]
```

- + Libraries keep depending on the headers
- + `HardwareInfo` travels up to the application, which adds the implementation itself

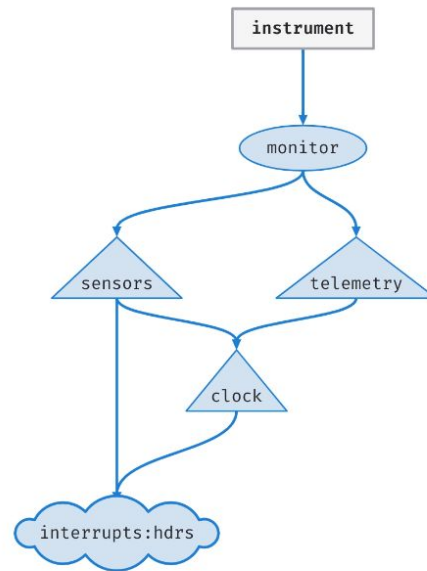
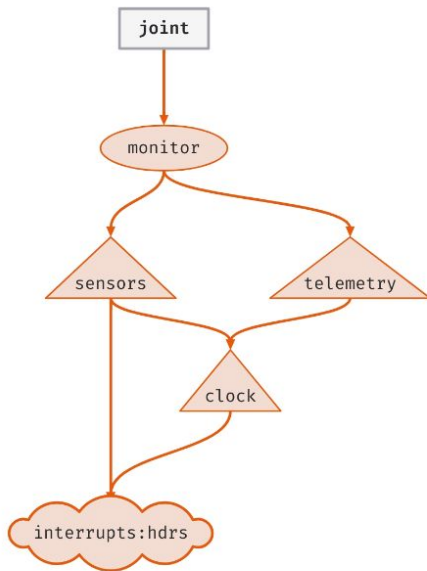
THE SOLUTION

HEADERS STAY, THE IMPLEMENTATION MOVES UP



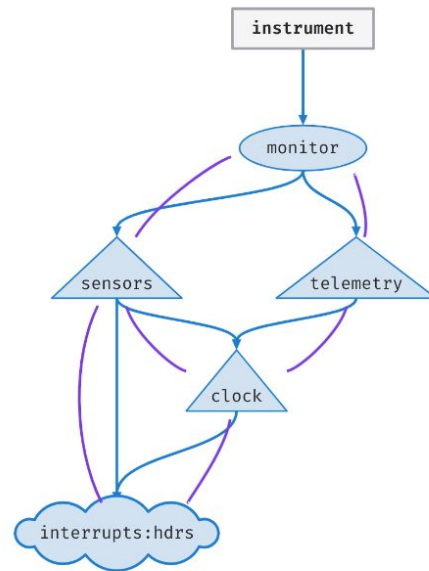
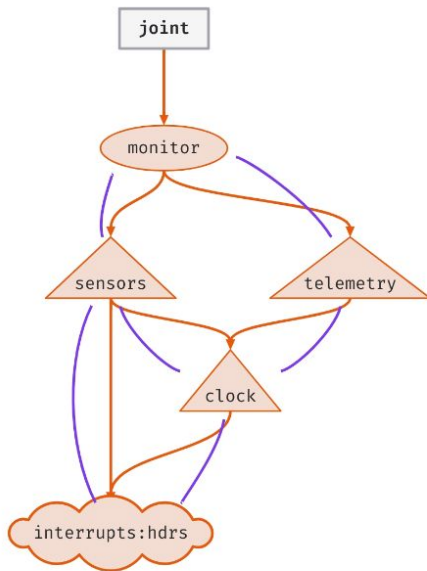
THE SOLUTION

HEADERS STAY, THE IMPLEMENTATION MOVES UP



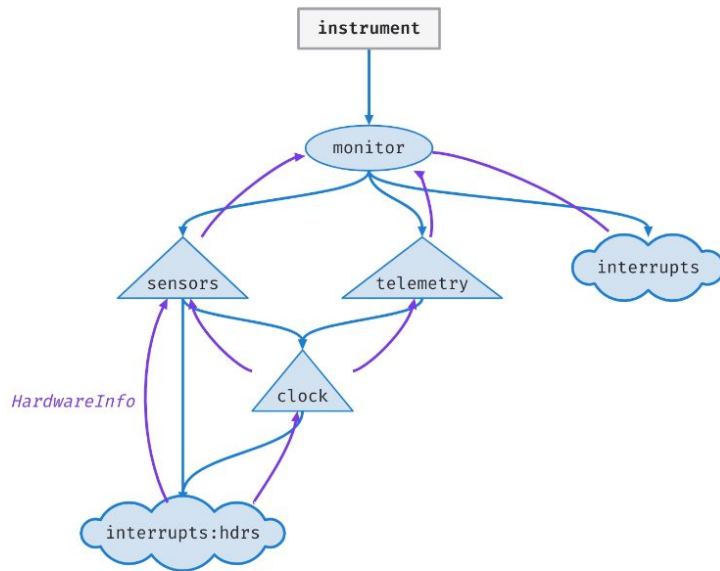
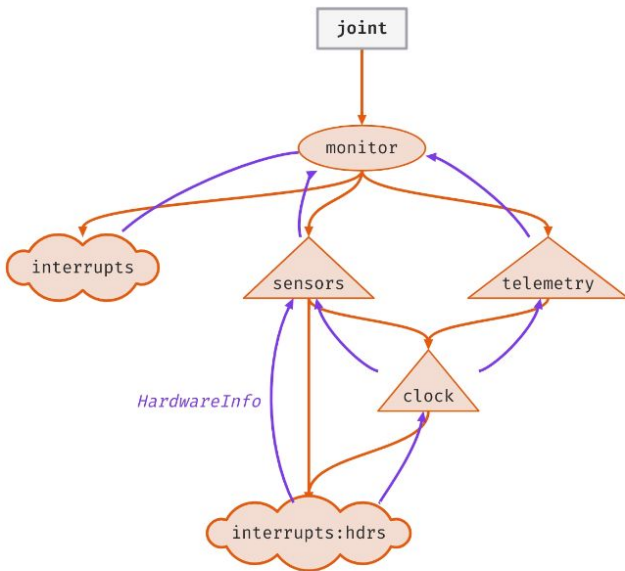
THE SOLUTION

HEADERS STAY, THE IMPLEMENTATION MOVES UP



THE SOLUTION

HEADERS STAY, THE IMPLEMENTATION MOVES UP



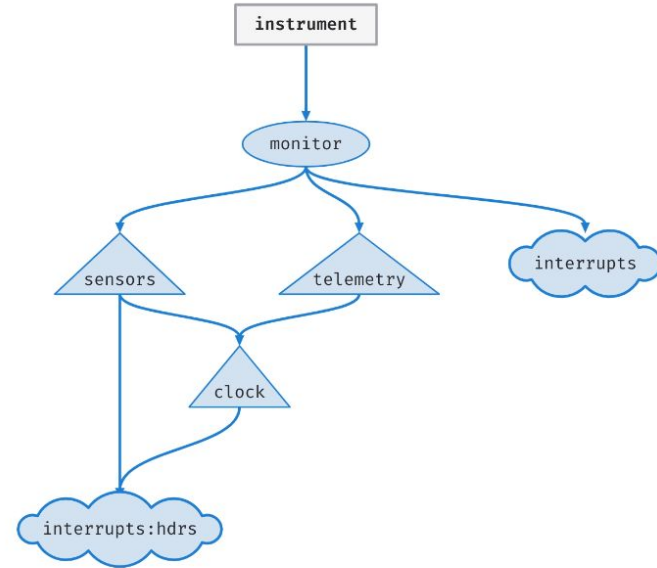
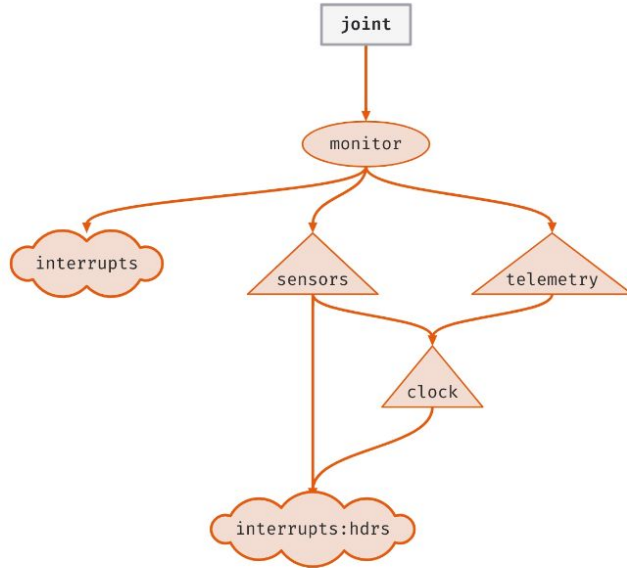
THE SOLUTION, IN CODE

A RESET TRANSITION

```
def _reset_impl(settings, attr):  
    # Back to the default value. Only what the libraries do not use:  
    # the target platform stays, as it picks the toolchain.  
    return {"//boards:board": "none"}  
  
reset = transition(  
    implementation = _reset_impl,  
    inputs = [],  
    outputs = ["//boards:board"],  
)  
  
app = rule(  
    attrs = {"deps": attr.label_list(cfg = reset)}, # libs  
    ...  
)
```

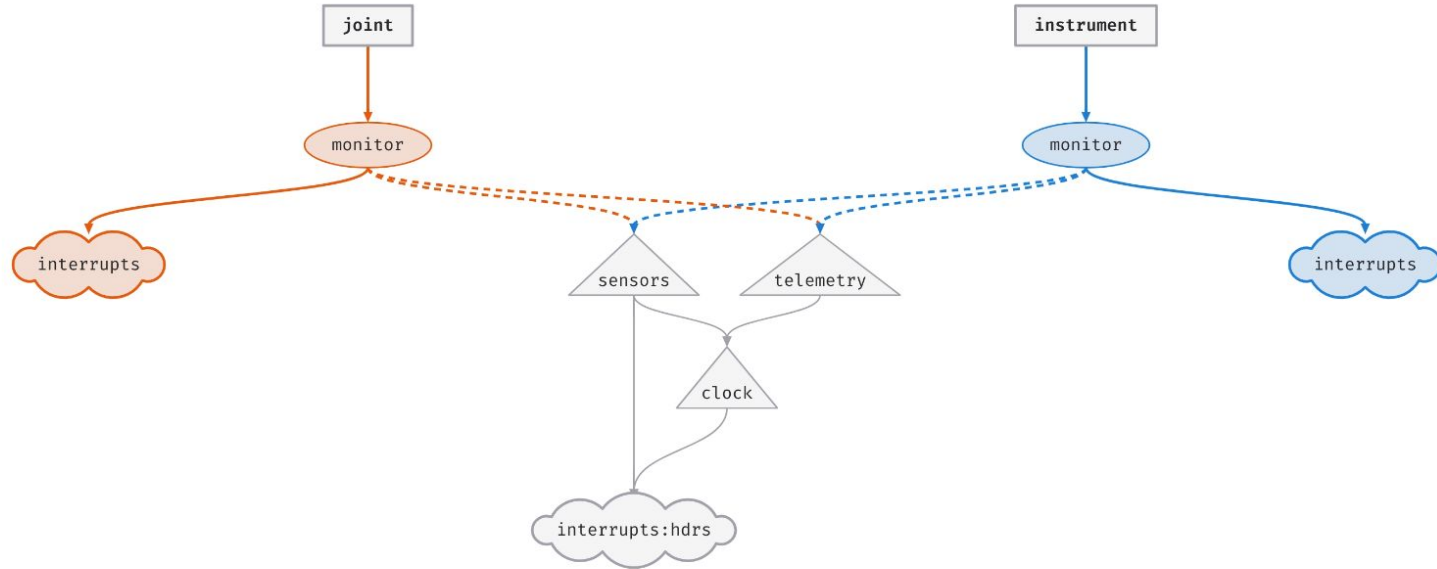
THE RESULT

ONE COPY OF EACH LIBRARY



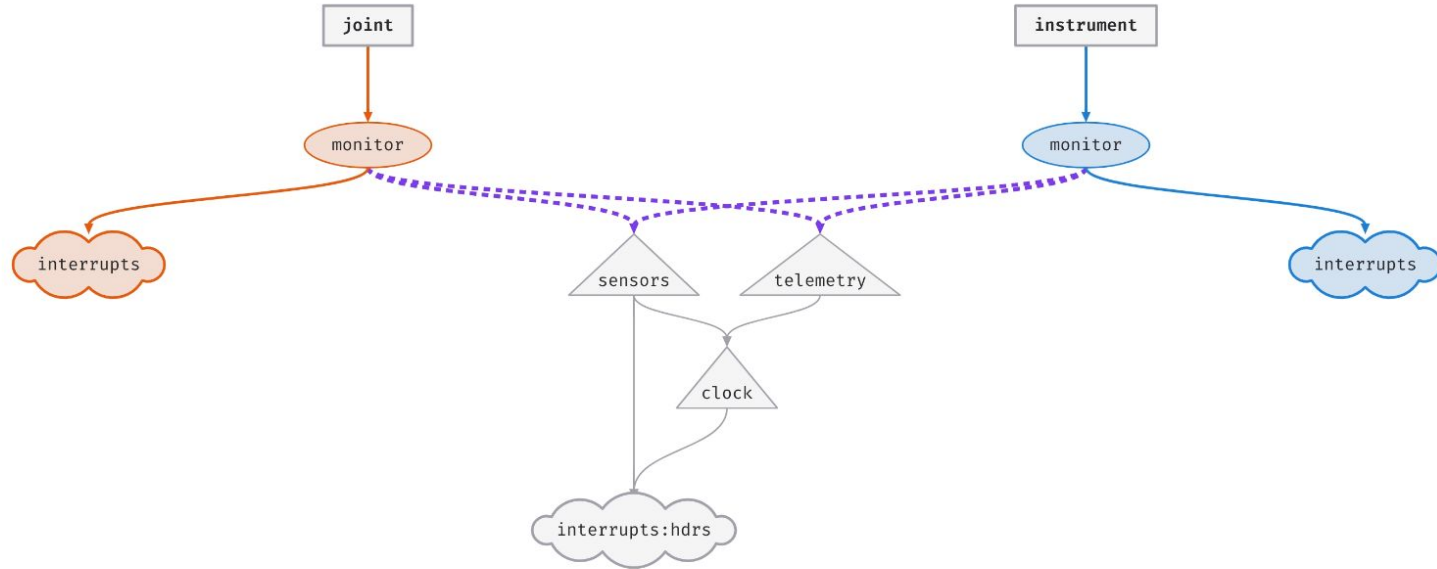
THE RESULT

ONE COPY OF EACH LIBRARY



THE RESULT

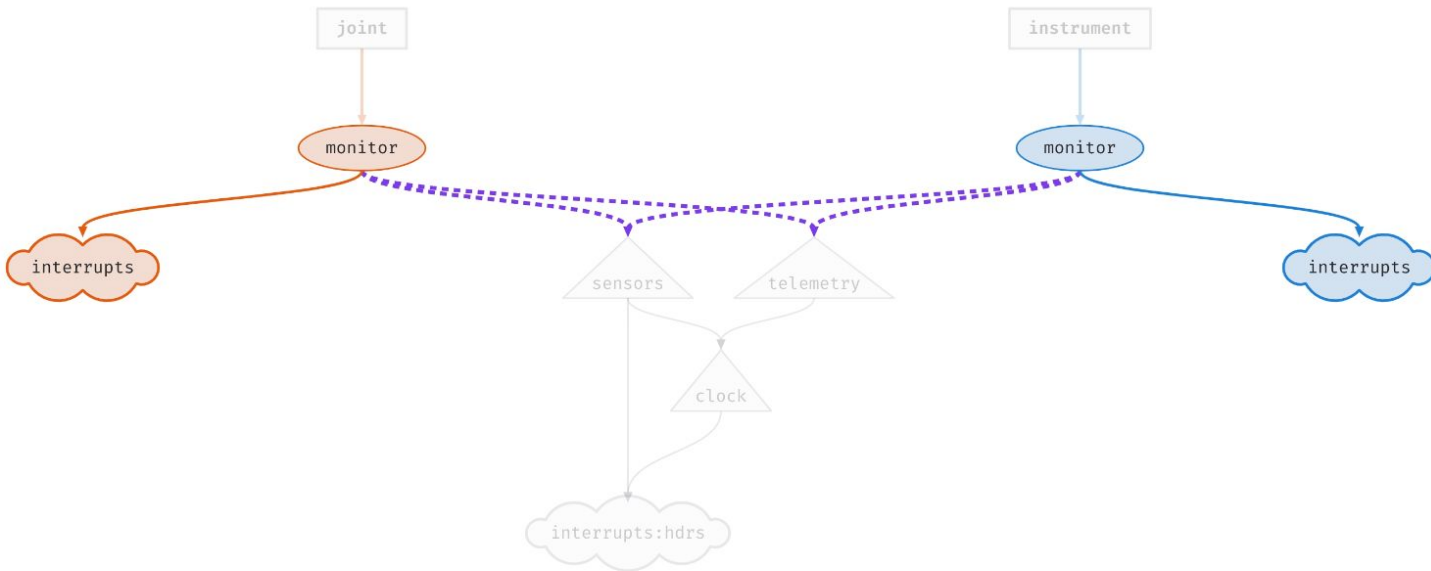
ONE COPY OF EACH LIBRARY



Reset transitions on the edges to the libraries.

THE RESULT

ONE COPY OF EACH LIBRARY



Reset transitions on the edges to the libraries.

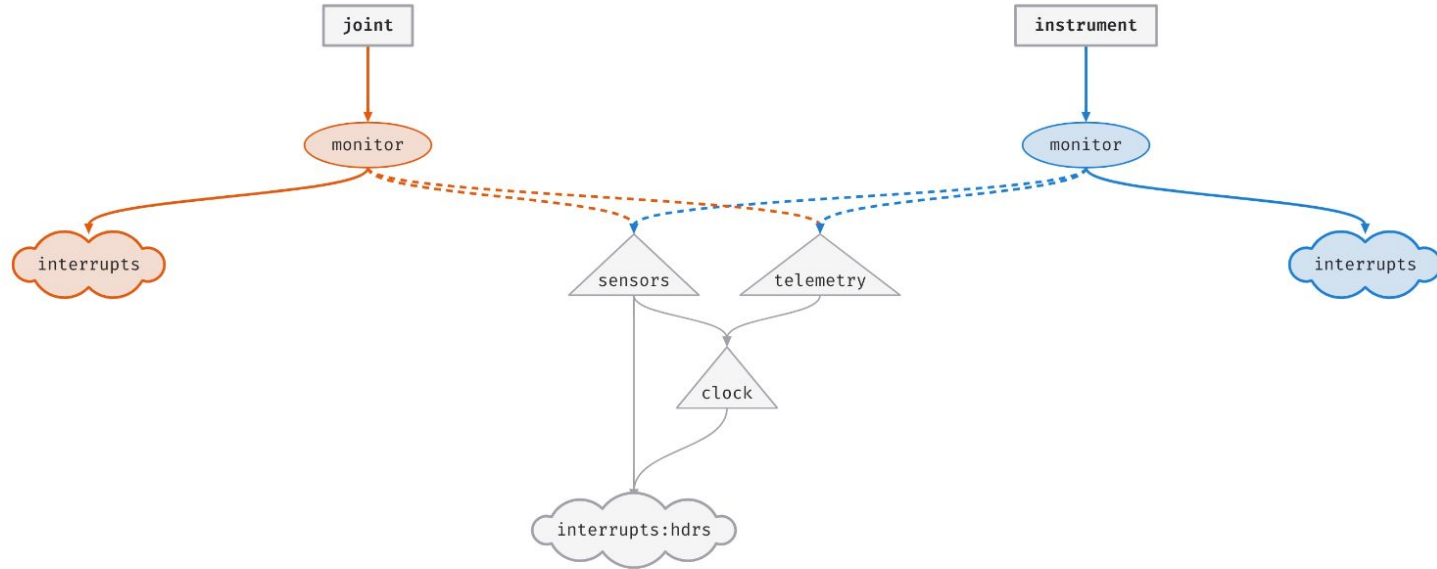
`monitor` is still linked twice: its dependencies really differ.

TAKEAWAY

Reset transitions: the art of **discarding useless configurations**, to trim down the graph.

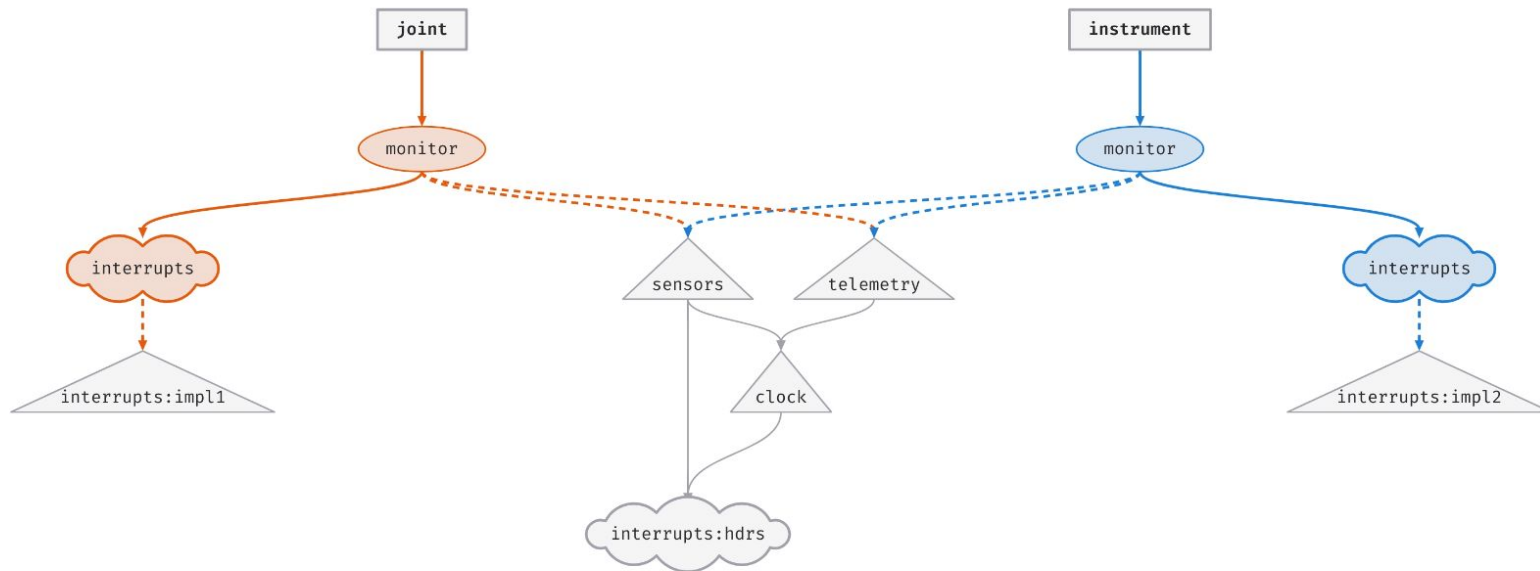
BONUS

RESET THE IMPLEMENTATION TOO



BONUS

RESET THE IMPLEMENTATION TOO



The implementation only depends on the compiler: reset it too.

Not strictly needed, but it cannot hurt.

TAKEAWAY

Reset transitions: the art of **moving decisions up the tree**, so they can be discarded where they are not needed.

RESULTS

IN PRACTICE

3.09M

configured targets, with reset transitions

3,091,863



without them: the build crashes

no way to even get the number

- + Without reset transitions, there is nothing to compare: the build does not get through
- + Removing them even introduces dependency cycles

DRAWBACKS

THE PRICE

- + `bazel query` crashes: it expands every branch of every `select()`, but only the combinations actually used are defined:

```
$ bazel query "deps(//images/...)"  
ERROR: Evaluation of query "deps(//images/...)" failed: preloading transitive closure failed:  
no such target '//interrupts/boards:_impl_grip': target '_impl_grip' not declared in package  
'interrupts/boards'
```

- + `bazel cquery` and `bazel build` only work on images: a library or an app on its own gets a configuration the build never uses, and probably an invalid one:

```
$ bazel cquery "deps(//images/...)"  
ERROR: configurable attribute "actual" in //third_party/stress:stress doesn't match this  
configuration: //third_party/stress:stress is only available for the following configurations:  
    //configs:linux_x86_64  
ERROR: Analysis of target '//images/joint/rev_2:binaries' failed; build aborted
```

- + You need to be sure of what you are doing: resets can be over-zealous, and nothing checks them

BENEFITS

WHAT WE GAIN

- + **Shorter analysis:** every configuration saved is analysis time saved

3,091,863 configured targets $\Rightarrow$ **2 minutes** of analysis

INFO: Analyzed 6602 targets (5467 packages loaded, 3091863 targets configured, 1054244 aspect applications).

- + **Full control over the configuration flow:** each edge decides which settings to keep and which to reset
- + **Flexibility:** libraries and apps can be reused without caring about implementation details

A BUG WE HIT

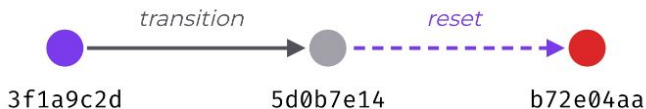
TWO CONFIGS, ONE DIFF: EMPTY

```
$ bazel config 3f1a9c2d b72e04aa  
Displaying diff between configs 3f1a9c2d and b72e04aa  
$
```

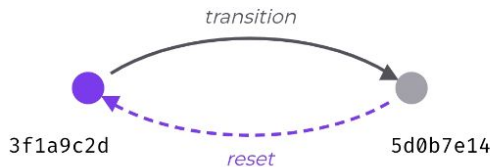
- + Different hashes, identical content
- + Why no-one found it before ?
- + Fix merged upstream, thanks to @fmeum



bazelbuild/bazel#29319



✗ the reset lands in a new configuration



✓ expected: back to the original configuration



2. PATH MAPPING

THE IDEA

A SECOND CHANCE, AT EXECUTION TIME

- + Cannot convince the graph that two nodes are the same? Catch it when running actions
- + Copies, symlinks, generated files: identical content from different configurations
- + Same inputs, same command: it should be one cache entry

HOW IT WORKS

PATH MAPPING

Two actions, one per configuration:

```
gcc -c lib/sensors.c -o bazel-out/k8-fastbuild-ST-3f1a9c2d/bin/lib/sensors.o  
gcc -c lib/sensors.c -o bazel-out/k8-fastbuild-ST-b72e04aa/bin/lib/sensors.o
```

With path mapping, the configuration part of every path becomes `cfg`:

```
gcc -c lib/sensors.c -o bazel-out/cfg/bin/lib/sensors.o  
gcc -c lib/sensors.c -o bazel-out/cfg/bin/lib/sensors.o
```

Both actions get the same cache key.

HOW IT WORKS

ONE ACTION, TWO CONFIGURATIONS

MAPPING

k8-fastbuild-ST-3f1a9c2d/lib/sensors.c

k8-fastbuild-ST-b72e04aa/lib/sensors.c

EXECUTION

cfg/lib/sensors.c

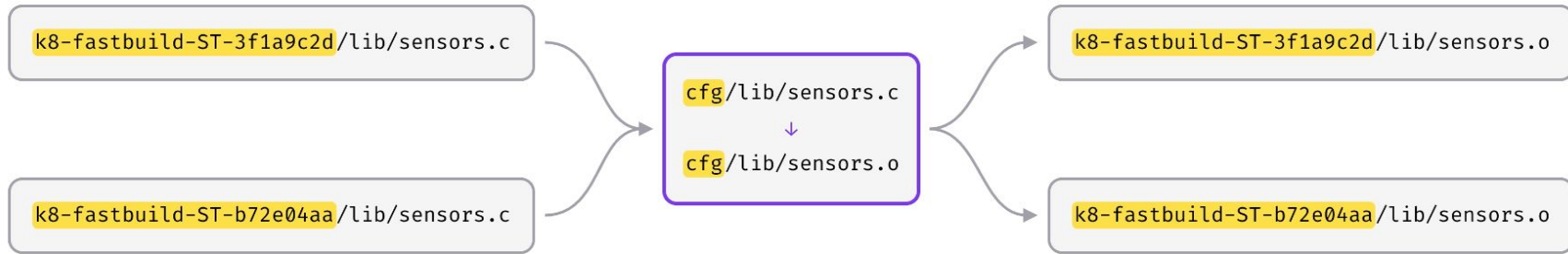


cfg/lib/sensors.o

UNMAPPING

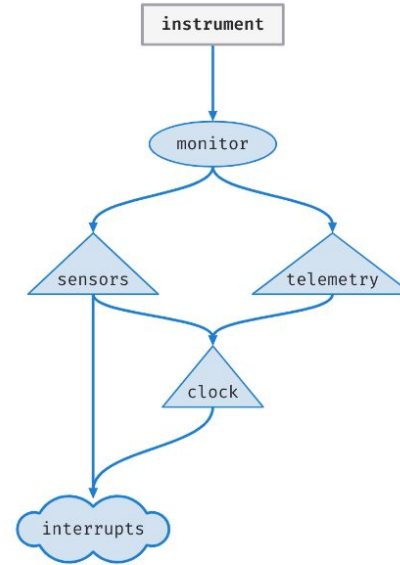
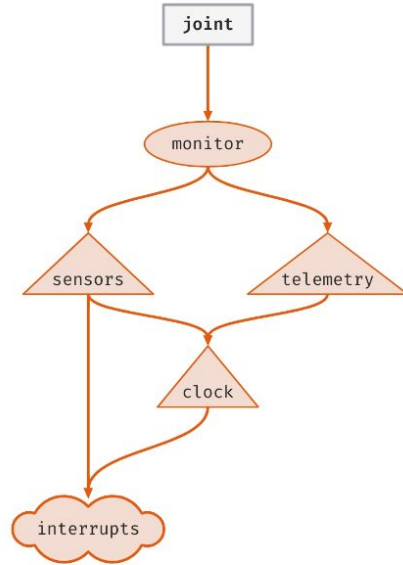
k8-fastbuild-ST-3f1a9c2d/lib/sensors.o

k8-fastbuild-ST-b72e04aa/lib/sensors.o



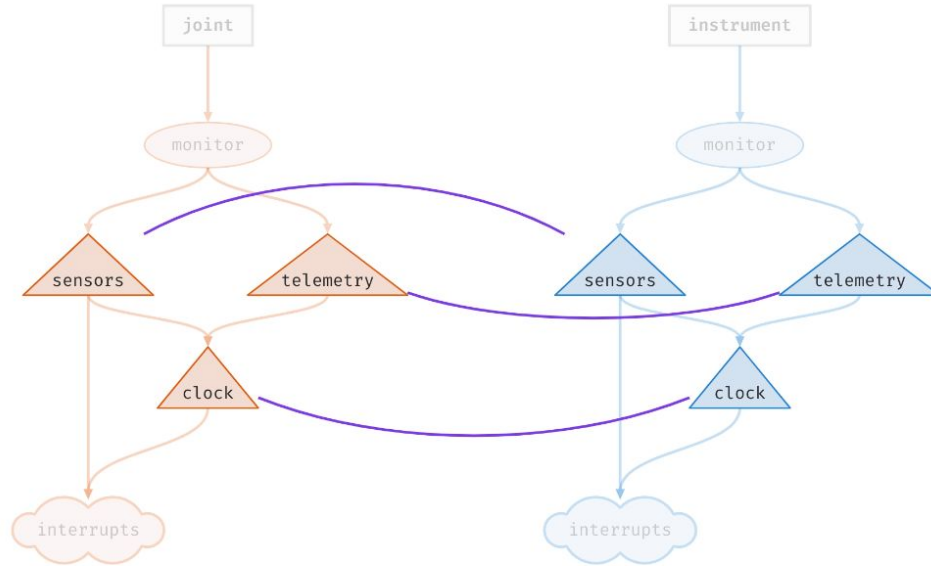
THE RESULT

SAME GRAPH, FEWER ACTIONS RUN



THE RESULT

SAME GRAPH, FEWER ACTIONS RUN



6 configured nodes, 3 actual executions.

A CAVEAT

MAPPED PATHS CAN COLLIDE

INPUTS

k8-fastbuild-ST-3f1a9c2d/gen/version.h

k8-fastbuild-ST-b72e04aa/gen/version.h

k8-fastbuild/gen/version.h

MAPPED INPUTS

cfg/gen/version.h

three different files, one path

When inputs collide, Bazel cannot map the paths, and runs the action unmapped.

DRAWBACKS

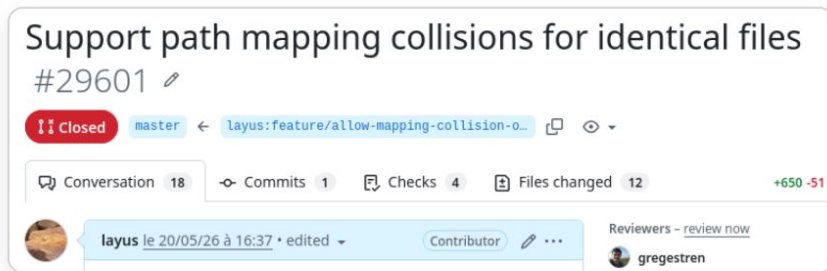
CAVEATS

- + Within one action, two inputs from different configurations must not map to the same path
- + When they do, path mapping is **silently disabled** for that action
- + Outputs still land under the configuration's directory: each action repeats the lookup
- + Needs an action cache, at least `--disk_cache`

A FIX WE MADE

IDENTICAL COPIES SHOULD NOT CONFLICT

- + Many configurations: copies of the same file, from different configurations, at the same mapped path
- + Path mapping saw a conflict and gave up
- + Our fix: same content hash, go ahead
- + Still to do: proper dependency detection



bazelbuild/bazel#29601

SUMMARY

PATH MAPPING, IN SHORT

PROS

- + Low effort deduplication
- + Very accurate, maximal

CONS

- + Silent: silently disabled, silently enabled
- + Hard to know the real gain



3. FLAG SCOPES

THE IDEA

FLAG SCOPES

- + A "recent" feature: first implementation merged about two years ago
- + Barely documented: the best reference is the tests in the Bazel source tree
- + A flag applies in a region: a set of labels
- + Outside of it, the flag always has its default value

Commit aee948b

 aranguyen authored and copybara-github committed le 19/12/24 à 20:24

Implementation for starlark flags scoping.

Fixes [#24042](#)

bazelbuild/bazel@aee948b, December 2024



Bazel

Generic API for Bazel Flag Propagation

Please read Bazel [Code of Conduct](#) before commenting.

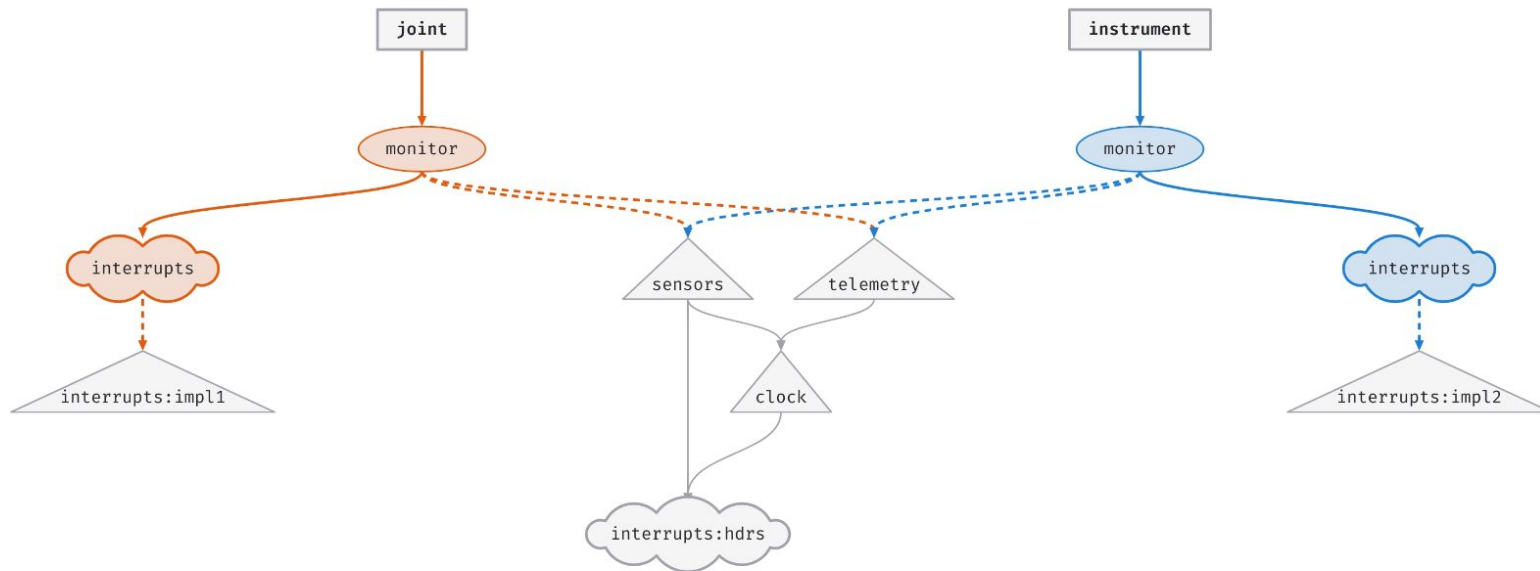
Authors: Ara Nguyen

Status: In Review

Design document, still "In Review"

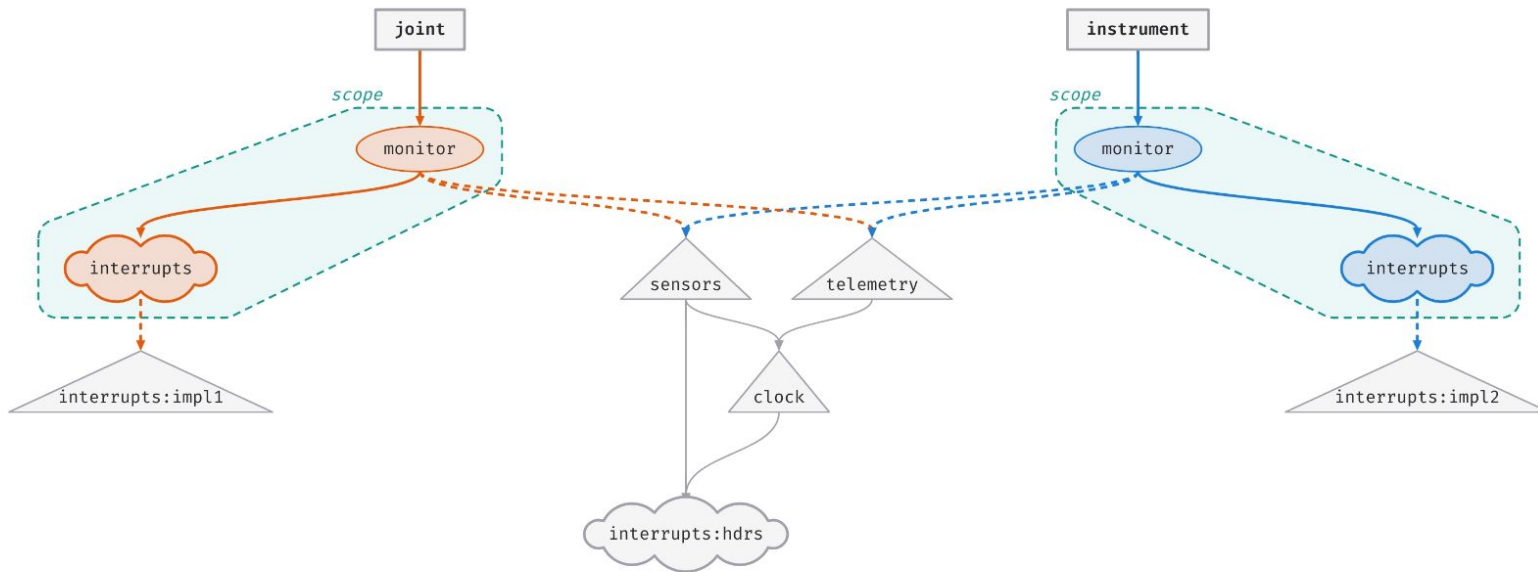
HOW IT WORKS

SAME GRAPH, WITH A SCOPE



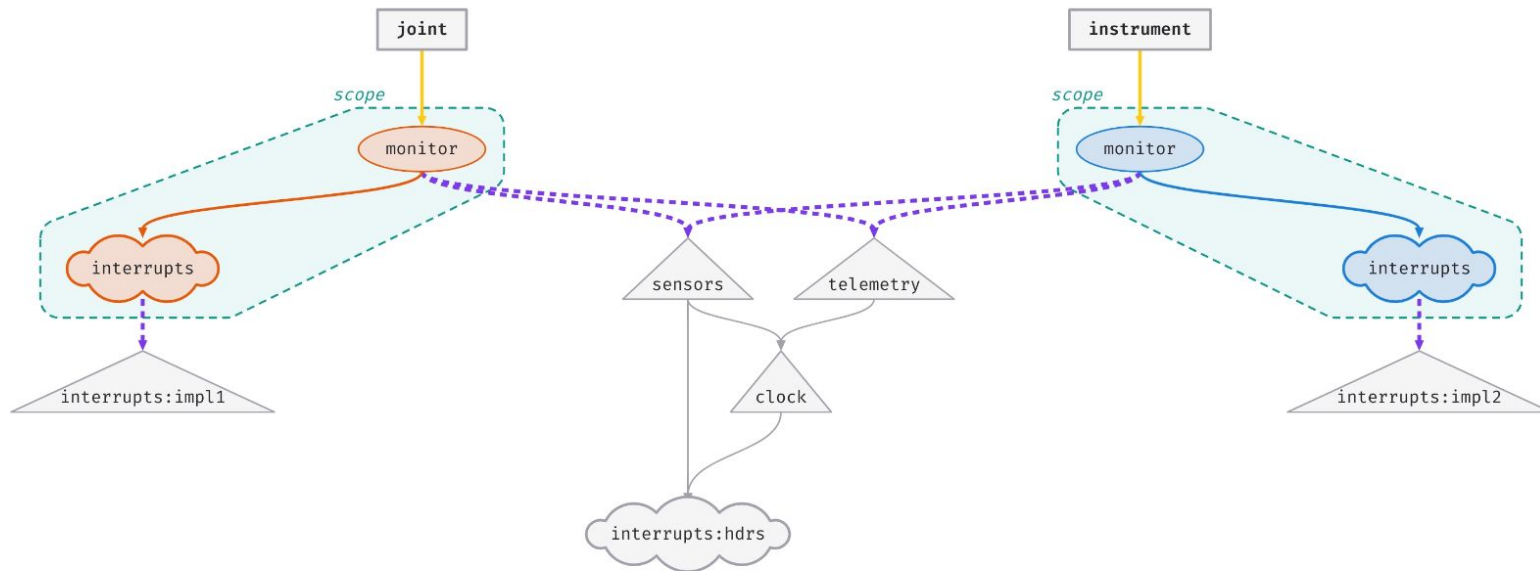
HOW IT WORKS

SAME GRAPH, WITH A SCOPE



HOW IT WORKS

SAME GRAPH, WITH A SCOPE



Entering still needs a transition that sets the flag.
Leaving resets it, imperatively: the value is lost.

HOW IT WORKS

DECLARING A FLAG SCOPE

`boards/PROJECT.scl` , next to the flags it scopes:

```
load("//project:project_proto.scl", "project_pb2")

project = project_pb2.Project.create(
    project_directories = [
        "//apps/monitor",
        "//interrupts",
    ],
)
```

`boards/BUILD` : the flag opts in to scoping:

```
string_flag(
    name = "board",
    build_setting_default = "none",
    scope = "project",
)
```

Outside `//apps/monitor` and `//interrupts` , `//boards:board` is back to its default value.

ASSESSMENT

A CONVENIENCE

- + Saves writing reset transitions by hand, but that is all
- + Great for libraries: settings set at the entry, unset when leaving
- + ...especially when you do not own the boundary
- + Scopes are declared with target names: cumbersome for us, better suited to rule sets

OPEN QUESTIONS

HARD PARTS

- + Depending on something *in a given configuration* is tricky: something must guard that line
- + Deep subtrees carry providers without using them: headers are used, but object files are only collected, and linked by `cc_binary`
- + Something to explore?

CONCLUSION

THE PATTERN

A FREQUENT PATTERN

A configuration that is only used to **select** in an alias, and does not influence the rules themselves.

SUMMARY

THREE TOOLS



RESET TRANSITIONS

WHEN

analysis

STRENGTH

explicit, "manual"

WEAKNESS

over-zealous, unchecked



PATH MAPPING

execution

automagic, catches the rest

hard to inspect



FLAG SCOPES

analysis

composable, for libraries

a convenience only

THANK YOU!



MODUS CREATE



by Modus Create

