



BAZELCON 2026

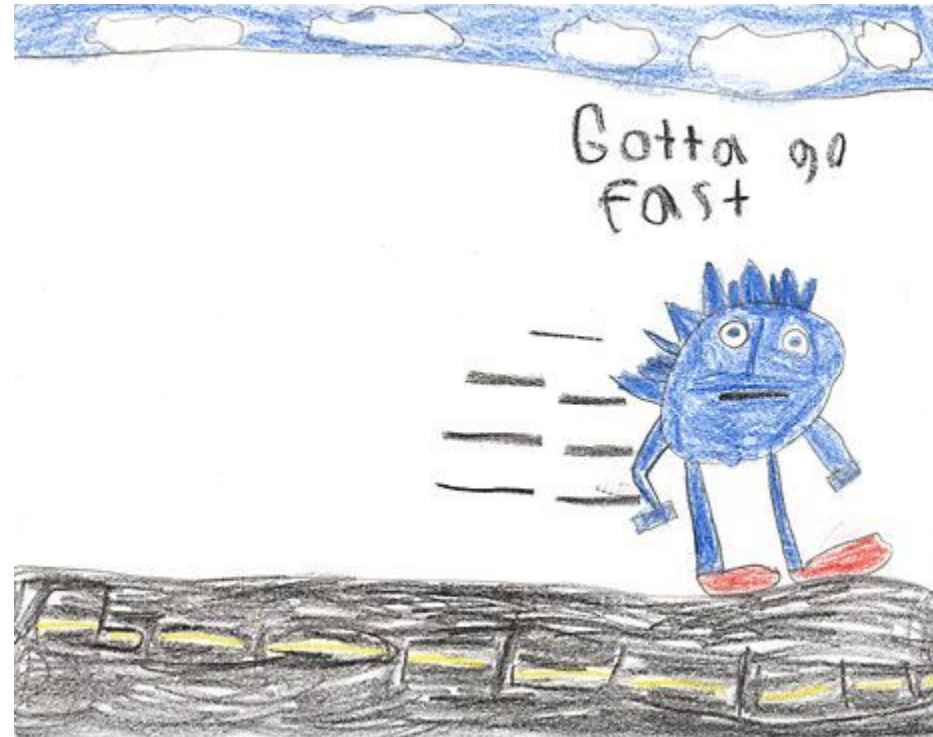
One Build, Many Configurations

Comparing Configuration Features in Bazel and Buck2



Scott Cao · Meta

Warning: Baseline knowledge of configuration is assumed



Overview



Bazel and Buck2 have both been working on similar configuration problems.

This talk looks at the general trends in their approaches and where they diverge.

PROBLEM 1A

Making builds work out of the box



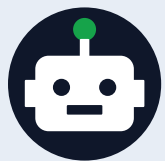
Every target should be able to build without extra flags

```
bazel build //services/search:indexer
```

```
bazel build //ml/vision:trainer
```

```
bazel build //android:app
```

```
bazel build //ios:app
```



An agent can build any target in the monorepo without additional context.

Building out of box may mean different configurations on different host

On a Linux-x86_64 host:

```
$ buck2 build buck2  
# builds buck2 for linux-x86_64
```

On a Mac-arm64 host:

```
$ buck2 build buck2  
# builds buck2 for mac-arm64
```

PROBLEM 1B

"Modes"



Every target can be built with a small set of “modes”

linux

mac

arm64

asan

debug

opt

```
$ buck2 build buck2 @linux
```

```
$ buck2 build buck2 @asan
```

```
$ buck2 build buck2 @debug @windows
```

```
$ buck2 build buck2 @mac @arm64
```

Every target can be built with a small set of “modes”



```
buck2 build //android:app @opt
```



```
buck2 build //search:indexer @opt
```

Changing modes should be fast/efficient

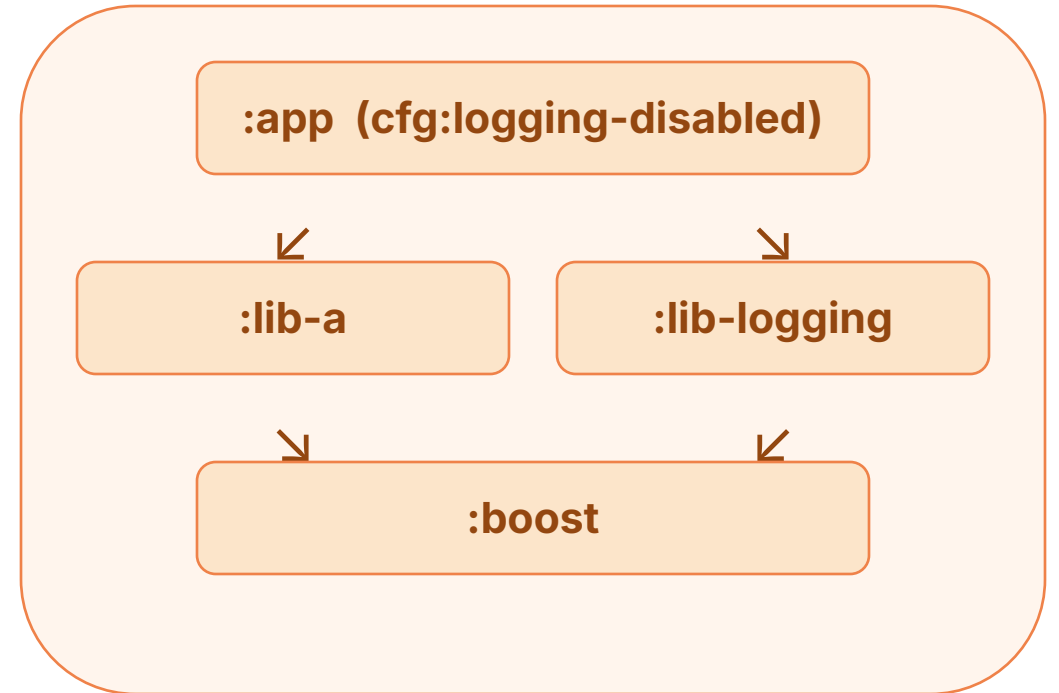
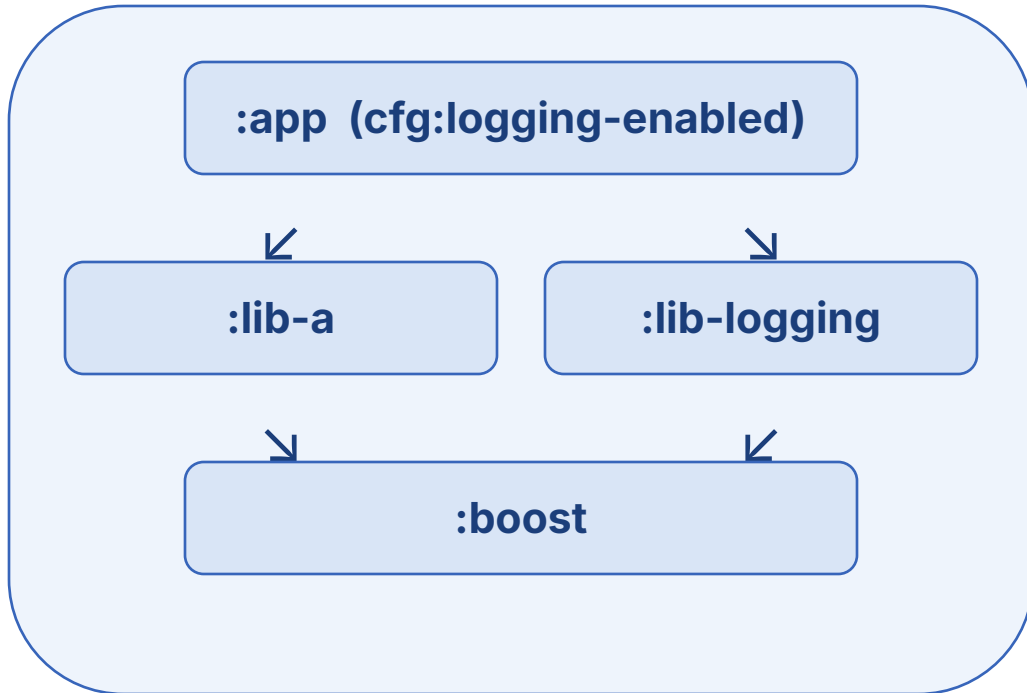


PROBLEM 2

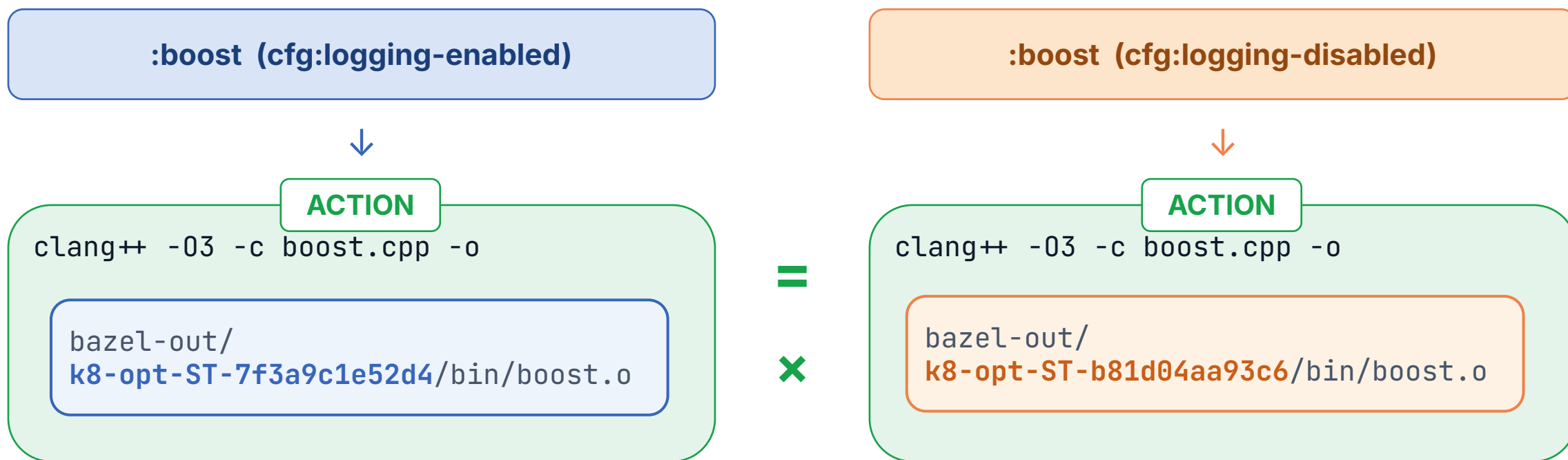
Configuration Explosion



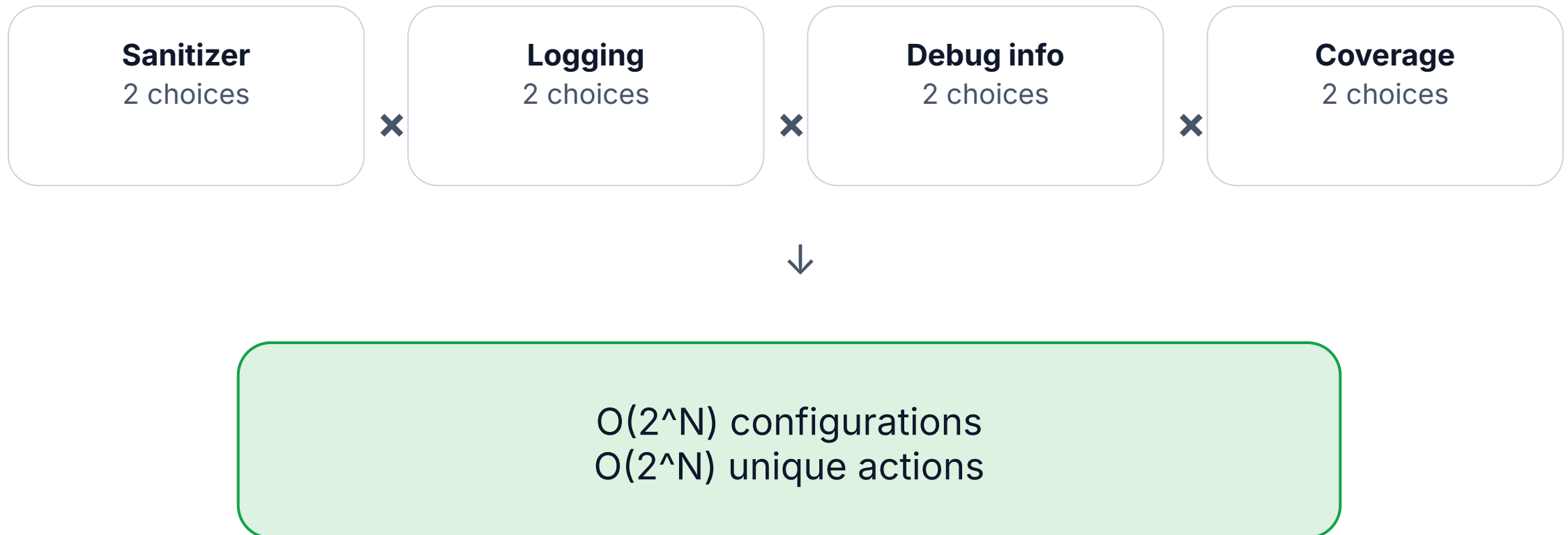
Different configurations diverge the whole graph



Configurations can unnecessarily diverge action keys



Feature combinations multiply configurations





SOLVING PROBLEM 1

Out-of-the-box builds and “modes”



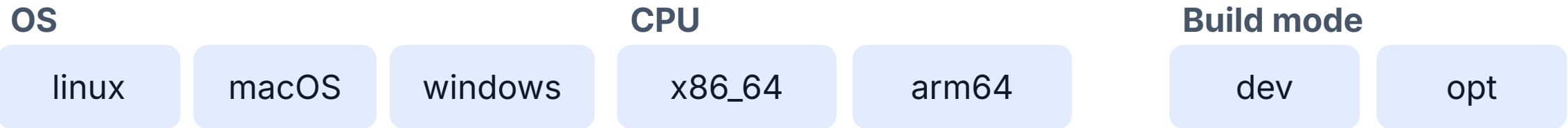
Case study: Buck2 Repository



buck2

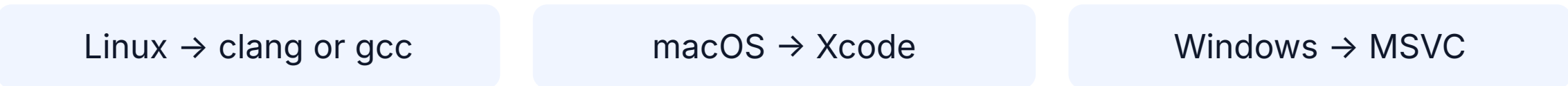
starlark-rust

What building buck2 requires



+ ASAN · while not commonly used, it could be added to any configuration above

Some toolchain properties follow the OS



starlark-rust has different defaults

buck2

starlark-rust

Building starlark-rust on its own

starlark-rust

`paging = disabled`

Building buck2

buck2



starlark-rust

`paging = enabled`



SOLVING PROBLEM 1

Transitions





Transition

```
# //flags/BUILD
string_flag(
    name = "paging",
    build_setting_default = "disabled",
    scope = "target",
)
```

```
# //buck2/BUILD
load("@with_cfg.bzl", "with_cfg")

buck2_binary, _ = with_cfg(rust_binary).set(
    "//flags:paging", "enabled",
).build()

buck2_binary(
    name = "buck2", # ...
)
```

bazel build //buck2:buck2



//flags:paging=enabled



Buck2 transitions can modify individual constraints in platform

```
def _paging_transition(cfg, refs):  
    constraints = dict(cfg.constraints)  
    constraints[refs.paging.setting.label] = refs.paging  
    return ConfigurationInfo(  
        constraints = constraints,  
        values = cfg.values,  
    )
```



Buck2 only supports constraints and not build flags

```
# BUCK file

constraint(
  name = "os",
  default = "none",
  values = ["linux", "macos", "windows", "none"],
)

# paging is a constraint, just like OS

constraint(
  name = "paging",
  default = "disabled",
  values = ["disabled", "enabled"],
)
```



PROBLEMS WITH TRANSITIONS

Transitions cannot apply to libraries

`buck2 binary
paging on`

`buck2 library
paging off`

Buck2 libraries, when built directly from CLI, will not have paging enabled.

This may unnecessarily explode configurations when someone builds buck2 libraries.





PROBLEMS WITH TRANSITIONS

Transitions are hard to apply across a whole project

A transition is defined on a rule, so covering all of Buck2 means covering its used rules one at a time.

rust_binary

rust_test

py_test

sh_test



PROBLEMS WITH TRANSITIONS

Not suited to expressing variations

ASAN is a configuration that we only want occasionally on CLI, but a transition requires declaring wrapper targets for ASAN ahead of time.

```
# //buck2/BUILD

load("@with_cfg.bzl", "with_cfg")
buck2_asan, _ = with_cfg(rust_binary).set(
    "//flags:sanitizer", "asan",
).build()

buck2_asan(name = "buck2_asan")
```



SOLVING PROBLEM 1

Flagsets





Flagsets

```
# PROJECT.sc1

project = project_pb2.Project.create(
    buildable_units = [
        buildable_unit_pb2.BuildableUnit.create(
            name = "dev",
            flags = ["--compilation_mode=fastbuild"],
            is_default = True,
            description = "Dev build",
        ),
        buildable_unit_pb2.BuildableUnit.create(
            name = "opt",
            flags = [
                "--compilation_mode=opt",
            ],
            description = "Optimized build",
        ),
    ],
)
```



Flagsets enabling builds without extra flags

```
# buck2/PROJECT.scl

buildable_unit_pb2.BuildableUnit.create(
    name = "dev",
    flags = ["--compilation_mode=fastbuild",
            "--//flags:paging=enabled"],
    is_default = True,
)

buildable_unit_pb2.BuildableUnit.create(
    name = "opt",
    flags = ["--compilation_mode=opt",
            "--//flags:paging=enabled"],
)
```

```
bazel build //buck2:buck2
```

```
bazel build //buck2/app/buck2_util:buck2_util
```



Flagsets: a limited list for all supported configurations

```
# PROJECT.scl
```

```
dev    --compilation_mode=fastbuild  
        -- //flags:paging=enabled
```

```
opt    --compilation_mode=opt  
        -- //flags:paging=enabled
```



People



Tooling



Agents



Naive flagsets could result in exponential generation

```
dev + linux + x86_64 → dev_linux_x86_64
opt + linux + x86_64 → opt_linux_x86_64
dev + linux + arm64 → dev_linux_arm64
opt + linux + arm64 → opt_linux_arm64
dev + mac + x86_64 → dev_mac_x86_64
opt + mac + x86_64 → opt_mac_x86_64
dev + mac + arm64 → dev_mac_arm64
opt + mac + arm64 → opt_mac_arm64
dev + windows + x86_64 → dev_windows_x86_64
opt + windows + x86_64 → opt_windows_x86_64
dev + windows + arm64 → dev_windows_arm64
opt + windows + arm64 → opt_windows_arm64
```

```
dev + linux + x86_64 + asan → dev_linux_x86_64_asan
opt + linux + x86_64 + asan → opt_linux_x86_64_asan
dev + linux + arm64 + asan → dev_linux_arm64_asan
opt + linux + arm64 + asan → opt_linux_arm64_asan
dev + mac + x86_64 + asan → dev_mac_x86_64_asan
opt + mac + x86_64 + asan → opt_mac_x86_64_asan
dev + mac + arm64 + asan → dev_mac_arm64_asan
opt + mac + arm64 + asan → opt_mac_arm64_asan
dev + windows + x86_64 + asan → dev_windows_x86_64_asan
opt + windows + x86_64 + asan → opt_windows_x86_64_asan
dev + windows + arm64 + asan → dev_windows_arm64_asan
opt + windows + arm64 + asan → opt_windows_arm64_asan
```



Composing flagsets

```
# PROJECT.scl

project = project_pb2.Project.create(
    always_allowed_configs = ["--config=asan"],
    buildable_units = [
        buildable_unit_pb2.BuildableUnit.create(
            name = "opt",
            flags = ["--compilation_mode=opt",
                    "--//flags:paging=enabled"],
        ),
    ],
)
```

```
bazel build //buck2:buck2 --scl_config=opt --config=asan
```

```
bazel build //buck2:buck2 --scl_config=opt
--platforms=//platforms:linux_x86_64
```

```
# .bazelrc

build:asan
    -- //flags:sanitizer=asan
```

→

opt+asan

→

opt+linux+x86_64



buck2/PROJECT.scl

```
project = project_pb2.Project.create(  
    always_allowed_configs = ["--config=asan"],  
    buildable_units = [  
        buildable_unit_pb2.BuildableUnit.create(  
            name = "dev",  
            flags = ["--compilation_mode=fastbuild",  
                    "--//flags:paging=enabled"],  
            is_default = True,  
            description = "Dev build",  
        ),  
        buildable_unit_pb2.BuildableUnit.create(  
            name = "opt",  
            flags = ["--compilation_mode=opt",  
                    "--//flags:paging=enabled"],  
            description = "Optimized build",  
        ),  
    ],  
)
```



buck2/starlark-rust/PROJECT.scl

```
project = project_pb2.Project.create(  
    always_allowed_configs = ["--config=asan"],  
    buildable_units = [  
        buildable_unit_pb2.BuildableUnit.create(  
            name = "dev",  
            flags = ["--compilation_mode=fastbuild"],  
            is_default = True,  
            description = "Dev build",  
        ),  
        buildable_unit_pb2.BuildableUnit.create(  
            name = "opt",  
            flags = ["--compilation_mode=opt"],  
            description = "Optimized build",  
        ),  
    ],  
)
```



Flagsets cannot compose with other flagsets

opt · buck2/PROJECT.scl

`--compilation_mode=opt`

`-- //flags:paging=enabled`



asan · buck2/PROJECT.scl

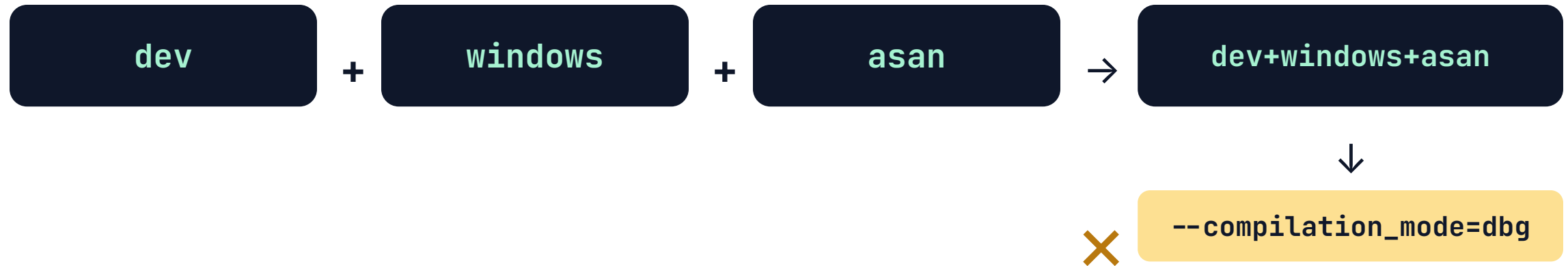
`-- //flags:sanitizer=asan`

`-- //flags:paging=disabled`

```
$ bazel build //buck2:buck2 --scl_config=opt --scl_config=asan
error: only one --scl_config can be selected
```



A special case for one combination needs its own flagset and cannot compose



```
# buck2/PROJECT.scl
```

```
dev          --compilation_mode=fastbuild
```

```
opt          --compilation_mode=opt
```

```
windows_dev_asan  -- //flags:sanitizer=asan  --compilation_mode=dbg
```



Conflicting flags block builds across projects

buck2 — dev

`-- //flags:paging=enabled`



buck2/starlark-rust — dev

`-- //flags:paging=disabled`



```
$ bazel build //buck2/...  
error: conflicting flags
```



SOLVING PROBLEM 1

Modifiers





Modifier

A modifier is applying one constraint in the configuration, producing a new configuration.

```
buck2 build //:buck2          # cfg: os = linux, mode = dev, sanitizer = none
buck2 build //:buck2?cfg//:asan # cfg: os = linux, mode = dev, sanitizer = asan
```



Modifiers can be specified on project, target, or CLI level

```
# buck2/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[enabled]"])
# buck2/starlark-rust/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[disabled]"])
# buck2/starlark-rust/experimental/BUCK
rust_binary(
    name = "page-out-bench",
    modifiers = ["//buck2/app:paging[enabled]"],
)

# command line
$ buck2 build :page-out-bench
```



Modifiers can be specified on project, target, or CLI level

```
# buck2/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[enabled]"])
# buck2/starlark-rust/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[disabled]"])
# buck2/starlark-rust/experimental/BUCK
rust_binary(
    name = "page-out-bench",
    modifiers = ["//buck2/app:paging[enabled]"],
)

# command line
$ buck2 build :page-out-bench
```

**Apply PACKAGE
modifier**
buck2/PACKAGE
cfg: paging = enabled



Modifiers can be specified on project, target, or CLI level

```
# buck2/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[enabled]"])
# buck2/starlark-rust/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[disabled]"])
# buck2/starlark-rust/experimental/BUCK
rust_binary(
    name = "page-out-bench",
    modifiers = ["//buck2/app:paging[enabled]"],
)

# command line
$ buck2 build :page-out-bench
```

**Apply PACKAGE
modifier**
buck2/PACKAGE
cfg: paging = enabled



**Apply PACKAGE
modifier**
starlark-rust/PACKAGE
cfg: paging = disabled



Modifiers can be specified on project, target, or CLI level

```
# buck2/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[enabled]"])
# buck2/starlark-rust/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[disabled]"])
# buck2/starlark-rust/experimental/BUCK
rust_binary(
    name = "page-out-bench",
    modifiers = ["//buck2/app:paging[enabled]"],
)

# command line
$ buck2 build :page-out-bench
```

**Apply PACKAGE
modifier**
buck2/PACKAGE
cfg: paging = enabled



**Apply PACKAGE
modifier**
starlark-rust/PACKAGE
cfg: paging = disabled



Apply target modifier
cfg: paging = enabled



Modifiers can be specified on project, target, or CLI level

```
# buck2/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[enabled]"])
# buck2/starlark-rust/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[disabled]"])
# buck2/starlark-rust/experimental/BUCK
rust_binary(
    name = "page-out-bench",
    modifiers = ["//buck2/app:paging[enabled]"],
)

# command line
$ buck2 build :page-out-bench
```

**Apply PACKAGE
modifier**
buck2/PACKAGE
cfg: paging = enabled



**Apply PACKAGE
modifier**
starlark-rust/PACKAGE
cfg: paging = disabled



Apply target modifier
cfg: paging = enabled



Modifiers can be specified on project, target, or CLI level

```
# buck2/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[enabled]"])
# buck2/starlark-rust/PACKAGE
set_cfg_modifiers(["//buck2/app:paging[disabled]"])
# buck2/starlark-rust/experimental/BUCK
rust_binary(
  name = "page-out-bench",
  modifiers = ["//buck2/app:paging[enabled]"],
)
```

```
# command line
$ buck2 build :page-out-bench?//buck2/app:paging[disabled]
```

**Apply PACKAGE
modifier**
buck2/PACKAGE
cfg: paging = enabled



**Apply PACKAGE
modifier**
starlark-rust/PACKAGE
cfg: paging = disabled



Apply target modifier
cfg: paging = enabled



Apply CLI modifier
cfg: paging = disabled



Conditional modifiers

Sometimes, you want to specify certain constraint as being conditional on other constraints, ex.
MSVC on Windows, GCC on Linux

```
# buck2/PACKAGE

set_cfg_modifiers([
  modifiers.conditional({
    "cfg//os:linux": "cfg//toolchain:gcc",
    "cfg//os:windows": "cfg//toolchain:msvc",
  }),
])

# ?windows is an alias for ?cfg//:os[windows]

$ buck2 build //:buck2?windows # windows + msvc

# ?linux is an alias for ?cfg//:os[linux]

$ buck2 build //:buck2?linux # linux + gcc
```



Buck2 can build all targets, each in its own configuration

```
$ buck2 build // ...
```



Every target builds
its own configuration



Buck2 can change multiple parts of configuration at once

```
$ buck2 build buck2?linux+arm64+opt
```



Buck2 can build in many configurations at once

```
$ buck2 build buck2?linux buck2?mac buck2?windows
```



buck2
linux



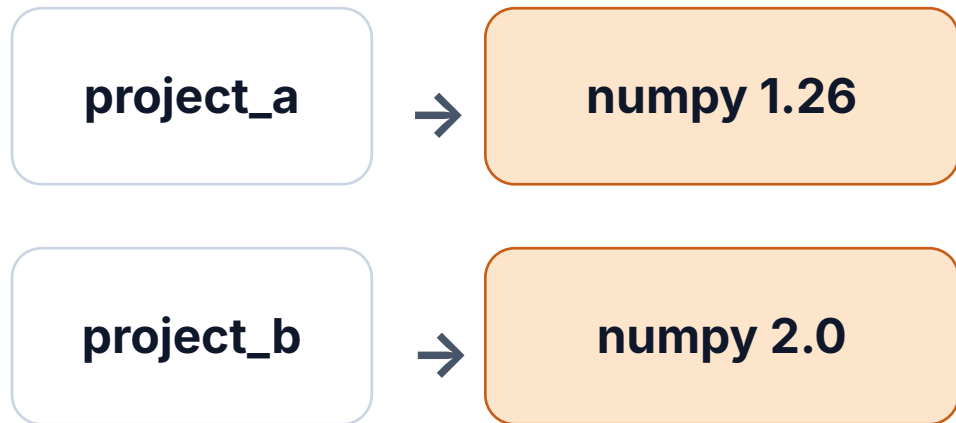
buck2
mac



buck2
windows



Use case: third-party multi-versioning in a monorepo

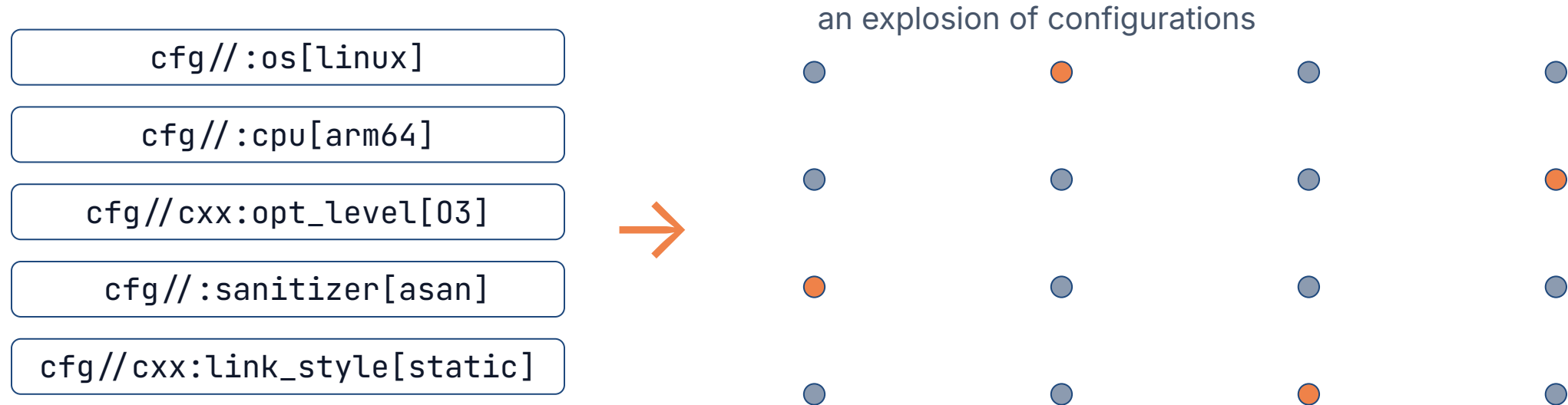


```
# project_a/PACKAGE
set_cfg_modifiers([
    "cfg//:numpy[1.26]",
])

# project_b/PACKAGE
set_cfg_modifiers([
    "cfg//:numpy[2.0]",
])
```



Flexibility can make it really easy for users to diverge configurations



The way configuration diverges can be difficult to track when configuration can be defined across multiple files and surfaces.



Missing a list of supported “modes”

```
# recommended modifiers, per project
```

```
x86_64
```

```
arm64
```

```
x86_64 + opt
```

```
arm64 + opt
```

```
x86_64 + asan
```

```
arm64 + asan
```

```
x86_64 + opt + asan
```

```
arm64 + opt + asan
```



People



Tooling



Agents



SOLVING PROBLEM 2

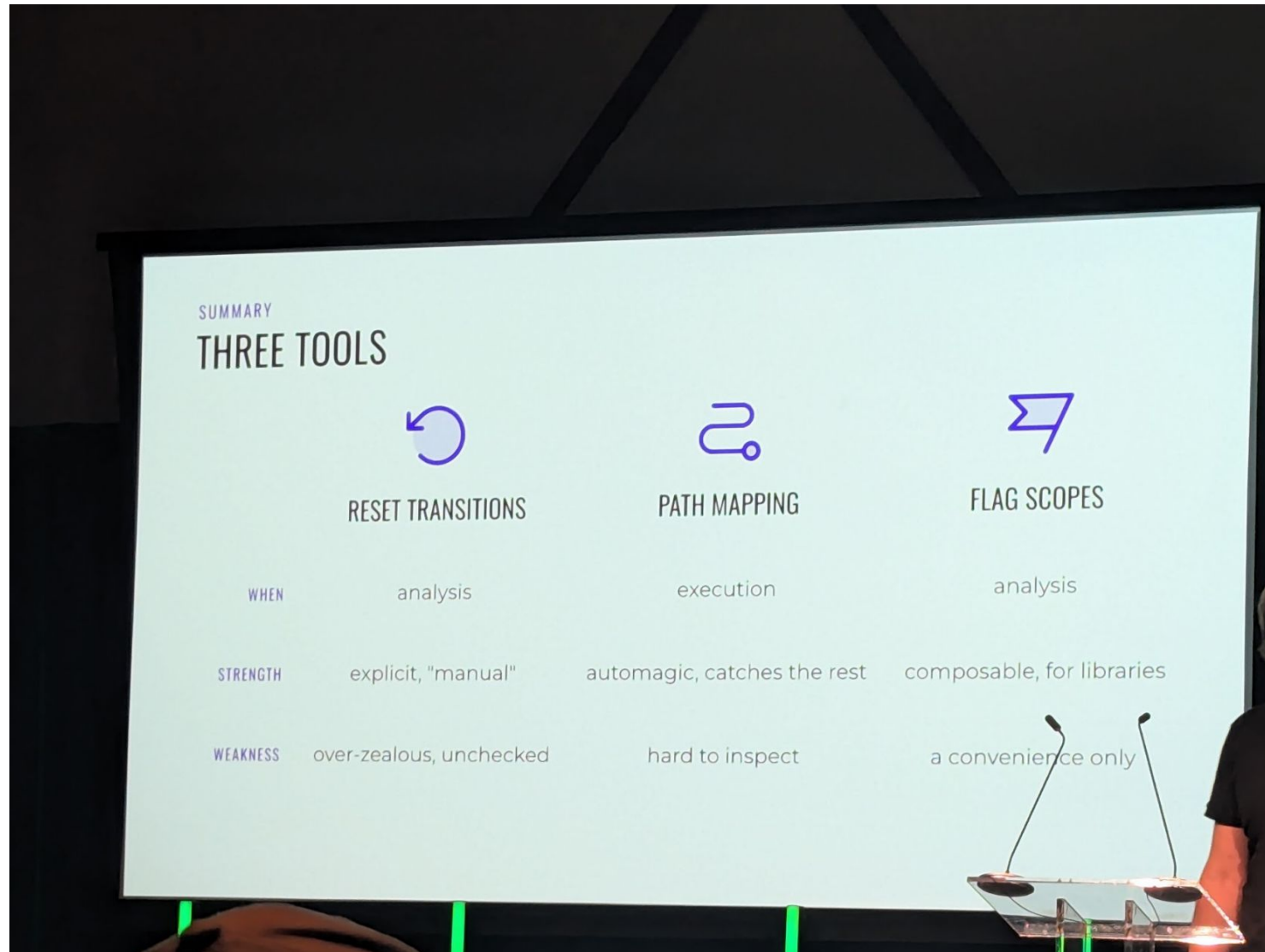
Configuration Explosion



Guillaume Maudoux's talk on
"Taming Configurations Explosion "
explained Problem 2 very well
yesterday.

I could explain it again, but I would explain it worse.

In conclusion



Credit:

Taming Configurations
Explosion 🌟

by Guillaume Maudoux

He gave the talk. I took
the photo.



Action deduplication





Bazel: Path Mapping

For each build action with path mapping enabled

**Rewrite the paths before
action executes**

`bazel-out/<config-a>
/bin/boost.o`



`bazel-out/cfg
/bin/boost.o`



**Action with path mapping
runs in its own execroot**

Requires remote or
sandboxed execution



**Outputs return to their
configuration-based paths
after action executes**

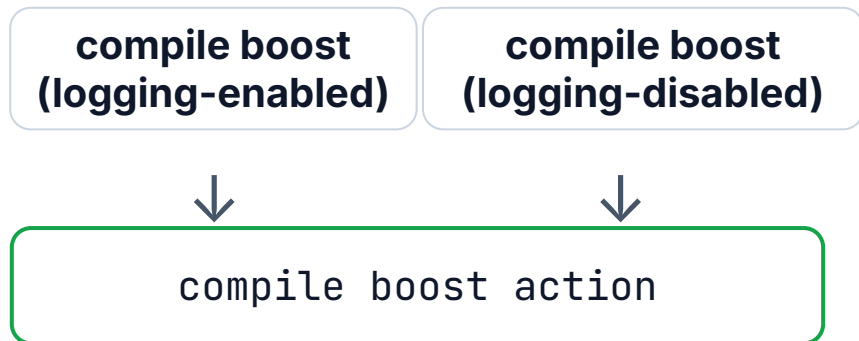
`bazel-out/cfg
/bin/boost.o`



`bazel-out/<config-a>
/bin/boost.o`



Path mapping: identical actions run once

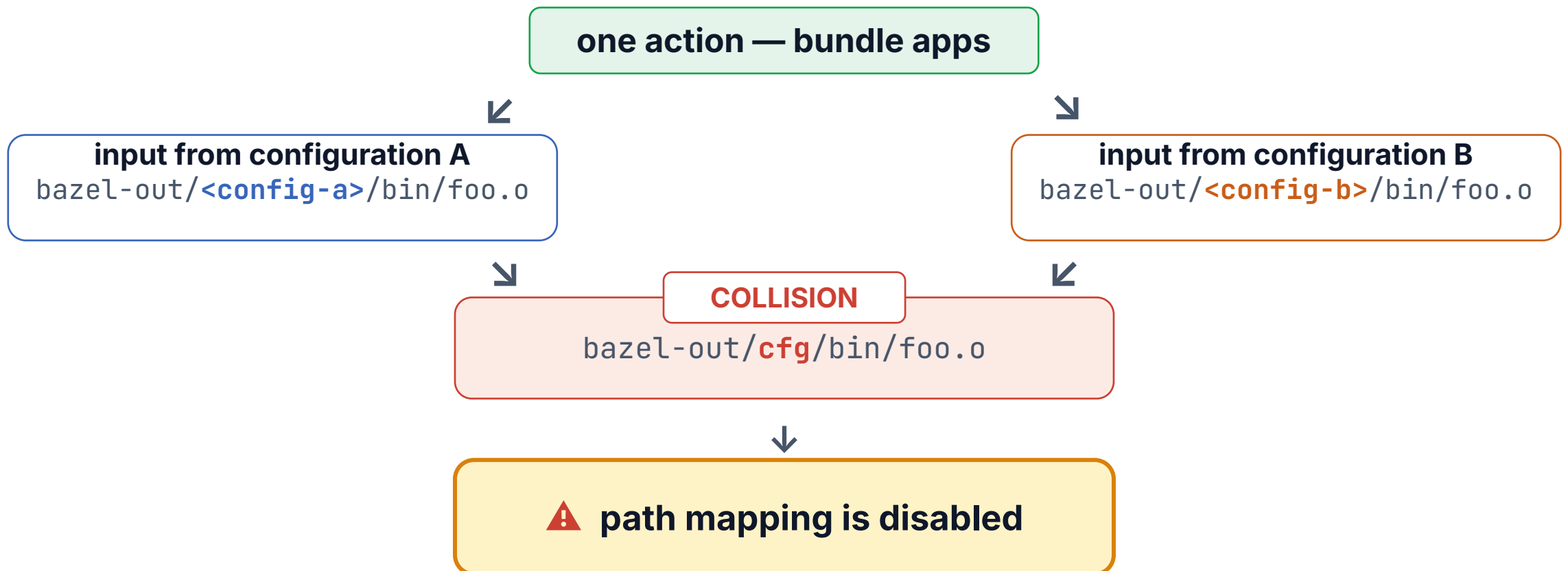


Path mapping shares work through remote caching.

Both configurations still exist in the configured graph, and each one is analyzed separately.



Path collision turns path mapping off





Buck2: Content-Based Output Paths

Action runs with placeholder output path

buck-out/.../
placeholder/foo.o

The compiler writes its output to a placeholder path.



Hash and move

buck-out/.../
<content-hash>/foo.o

The build hashes the content of the finished output and moves it to its content-based path.

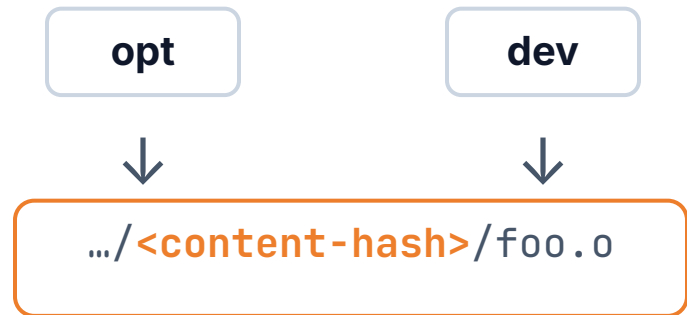


Downstream

Downstream actions read the finished artifact from its content-based path.



Content-based output paths



Works very similarly to Bazel's path mapping, except only output paths are moved.

Since each artifact is at a unique path identified by its content, there is no issue with path collisions.



Content-based paths: disk space usage

buck-out/

.../a3f9c1/foo.o

.../7b2e04/foo.o

.../c81d5a/foo.o

.../4f6b9e/foo.o

.../9d07aa/foo.o

.../1e5c38/foo.o

.../b4a72f/foo.o

.../63d9e0/foo.o

.../8c1f5b/foo.o

Work is under way to make buck-out's garbage collector (**clean-stale**) clean artifacts more aggressively.



Configuration trimming (future)





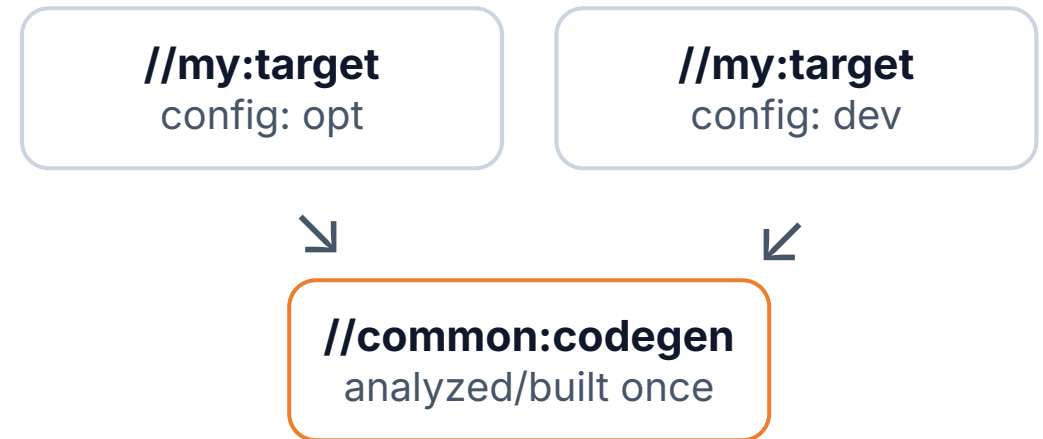
Automatically trim configured graph based on select usage

Today



`//common:codegen` does not select on build mode

With trimming





Flag scoping





Target scoping

```
# //flags/BUILD
string_flag(
  name = "paging",
  build_setting_default = "disabled",
  scope = "target",
)
```

buck2 binary
paging = enabled



deps of buck2
paging = disabled



exec deps of buck2
paging = disabled

Buck2 gets this for free since Buck2 has no build flags and only constraints



Project scoping

```
string_flag(  
    name = "paging",  
    build_setting_default = "disabled",  
    scope = "project",  
)
```

**building starlark-rust
under buck2**

paging = enabled

building starlark-rust

paging = disabled



deps outside buck2

paging = disabled

Project scoping is a really simple way for project owners to own and control the effect of their own flags



Common trends

Configuration definitions are moving to project level

Reducing unnecessary configuration explosion/divergence is a key focus for both systems.

Both systems are choosing to reduce configuration explosion with features on both trimming configurations and deduplicating actions

Buck2

Accepts higher costs for a more expressive API and multi-configuration builds from the CLI.

Bazel

Accepts a less expressive API for more control on capacity costs.