Go Editor Support in Bazel Workspaces

Jay Conrod (he/him)
Software Engineer, EngFlow
@jayconrod

About me

- Software engineer at EngFlow
- Previously on Go Team at Google
- Worked on Go modules, fuzzing
- Maintained rules_go, Gazelle

History

GOPATH

- GOPATH: list of directories containing Go packages.
- Lots of tools understood GOPATH, followed UNIX principle.
- Each editor had its own plugin, usually delegating to these tools.

bundle	delve	eg	errcheck
fillstruct	go-outline	godef	godoc
godoctor	gogrep	go-fuzz	goimports
gorename	goreturns	megacheck	wire

Bazel support

- Bazel was very new, and rules_go was even newer.
- No editor support for Bazel, but if you mostly followed GOPATH conventions, your editor would be happy*.
- Generated code broke everything, unless you checked it in.
- Without build-time code generation, why use Bazel?

Modules

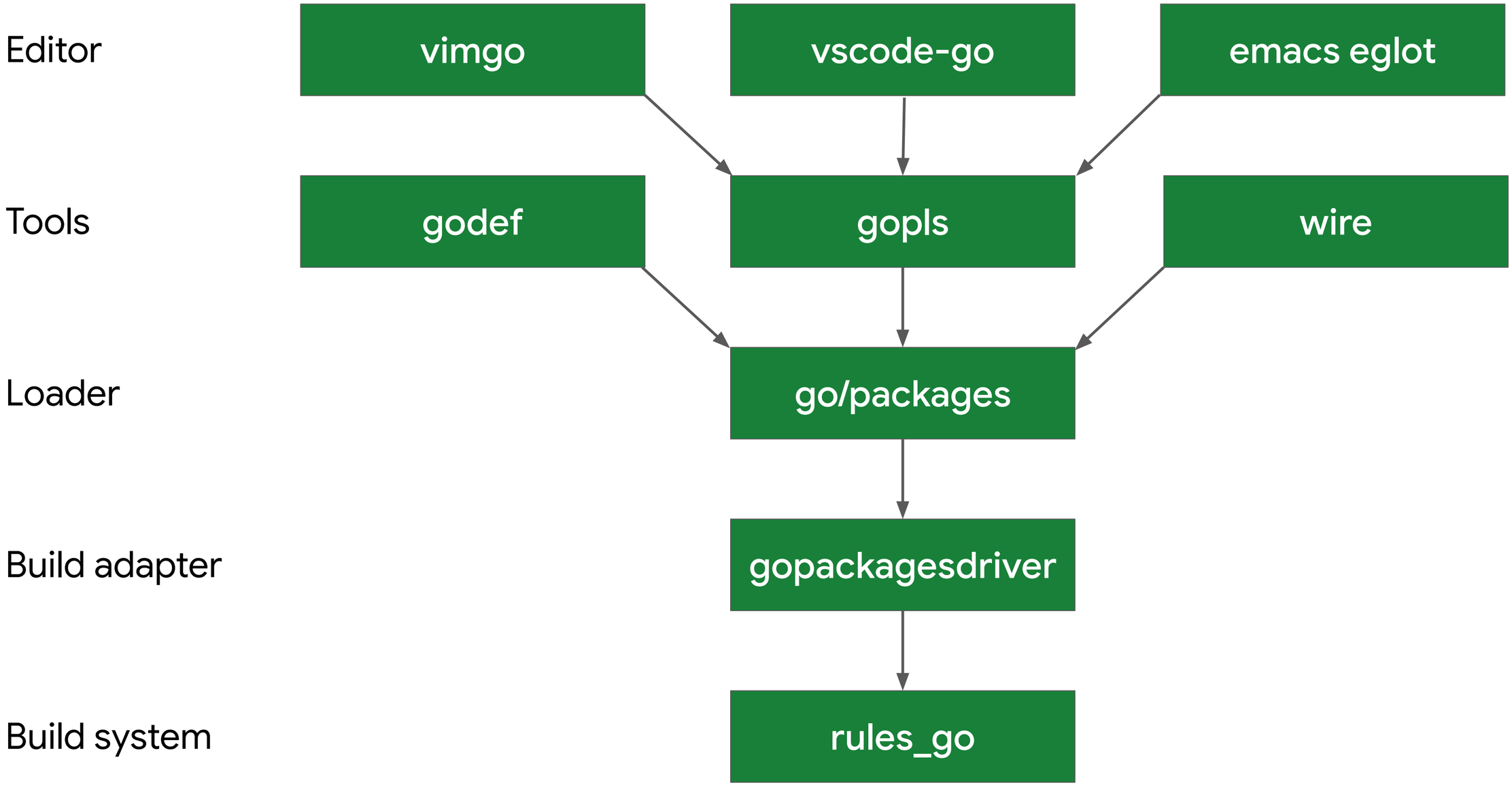
- Integrated dependency management into the toolchain. No more GOPATH.
- Totally different approach to file layout. None of the tools worked.
- We needed to rewrite everything to work with modules.
  And we needed to support GOPATH indefinitely.
  And Bazel. And Blaze. And maybe Buck.

- So basically,
  we're building complete editor support for all editors, all build systems.

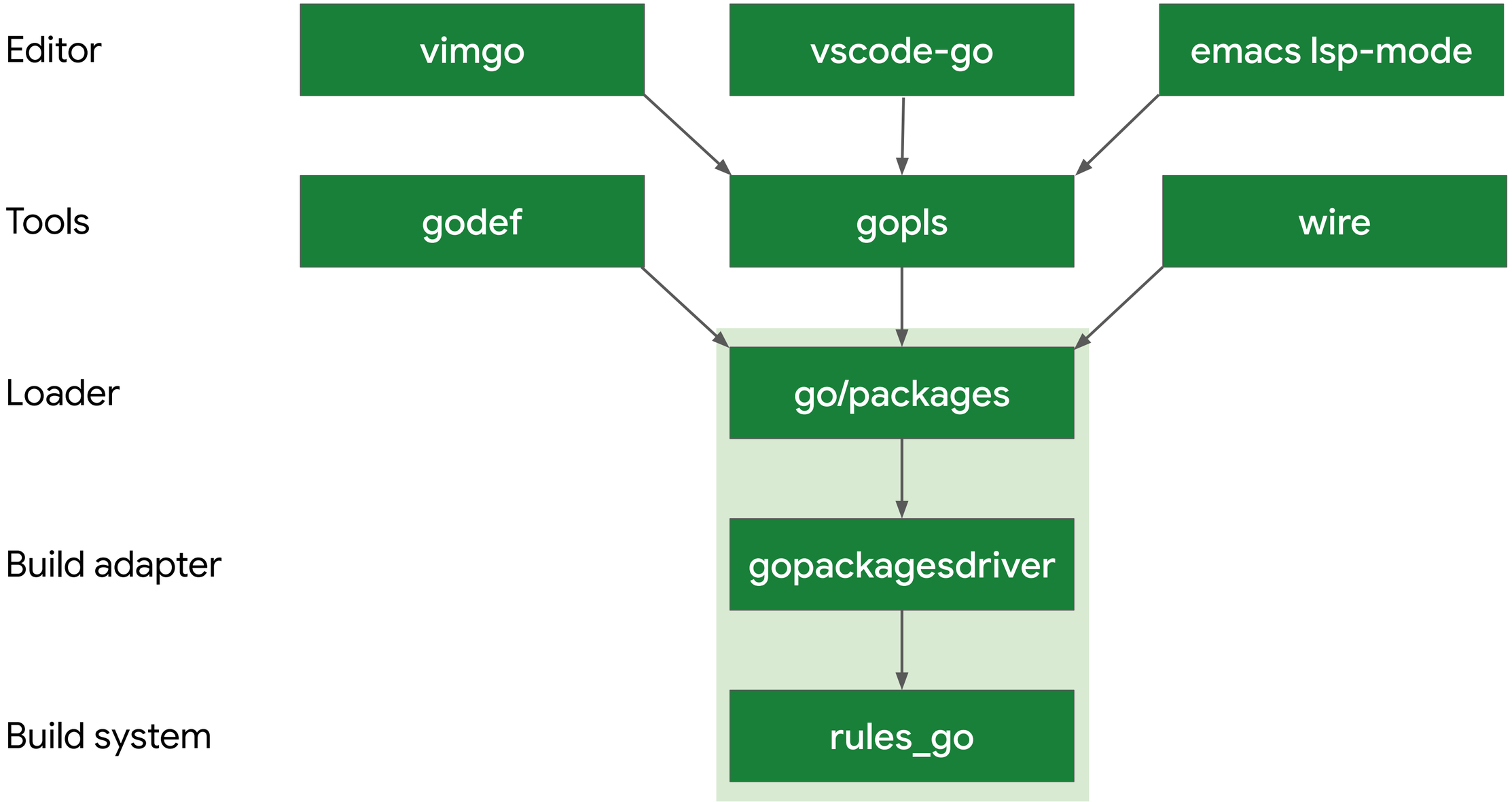
We can solve any problem by introducing an extra level of indirection.

— David J Wheeler

The Stack


Editor
Tools
Loader
Build adapter
Build system
vimgo
vscode-go
emacs eglot
godef
gopls
wire
go/packages
gopackagesdriver
rules_go

Demo


Editor
Tools
Loader
Build adapter
Build system
vimgo
vscode-go
emacs lsp-mode
godef
gopls
wire
go/packages
gopackagesdriver
rules_go

How does this work?

Need to know:

- What go_library target contains a file name?
- What files are in a go_library target? What does it depend on?
- Given an import string, like "google.golang.org/grpc",
  where is its go_library target?

golang.org/x/tools/go/packages

```
type Config struct {
  Mode       LoadMode
  Dir        string
  Env        []string
  BuildFlags []string
  ...
}
```

```
type Package struct {
  ID      string
  PkgPath string
  GoFiles []string
  Imports map[string]*Package
  ...
}
```

```
func Load(cfg *Config, patterns ...string) ([]*Package, error)
```

gopackagesdriver

- @io_bazel_rules_go//go/tools/gopackagesdriver
- Set GOPACKAGESDRIVER in editor's environment

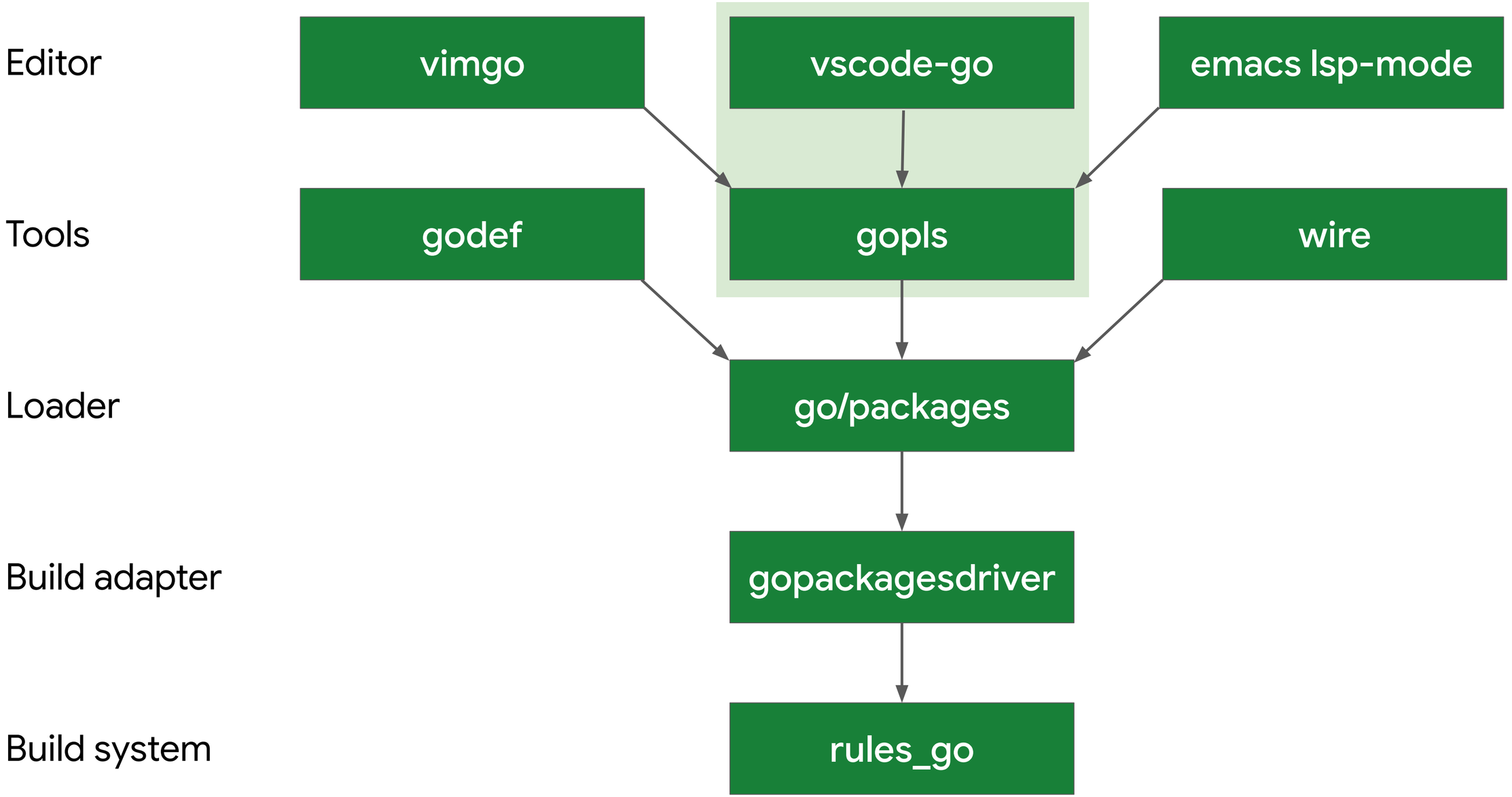
- Arguments: either files (preceded by "file=") or Bazel target names
- Stdin: JSON object explaining what should be loaded
- Stdout: JSON objects for each package

gopackagesdriver

1. Maps command line patterns to Bazel targets using `bazel query`.
2. Builds targets using `bazel build` with an aspect.
    - For each target, the aspect reads the GoArchive provider and writes a .json file.
    - Also builds generated srcs and export data if needed.
3. Reads JSON files, resolves file names, resolves imports, prints on stdout.

rules_go

- No special support needed in the rules themselves.
- GoArchive provider returned by every Go-compatible rule.
  - name, label, importpath
  - file, srcs, orig_srcs, runfiles
  - direct and transitive dependencies


Editor
Tools
Loader
Build adapter
Build system
vimgo
vscode-go
emacs lsp-mode
godef
gopls
wire
go/packages
gopackagesdriver
rules_go

gopls

- Implements Language Server Protocol (JSON RPC). Runs in separate process.
- When started, gopls loads *package metadata graph* for entire workspace, then loads *diagnostics* for each package.
- After start, editor sends commands like "definition", which require a response.
- Editor also sends notifications like "didChange".

gopls

How does this scale?

- **snapshot:** logical view of the workspace at a specific time. Created when the user changed something. Re-uses data from previous snapshot.
- Package metadata graph regenerated only for relevant changes.
- **Cache:** re-use deterministic results.
  - Keys are hashes of inputs.
  - Values could be anything: typically diagnostics, type info
- **gopls is basically a build system.**

vscode-go

- Most popular Go editor, followed by GoLand, vimgo, emacs.
- Originally by Microsoft, adopted by Go Tools Team.
- Written in TypeScript. Keeps the project small.
- Exposes features, installs tools, communicates with gopls, delve, vet.

Wrap up

Make things better!

- If you work in Go and want to make this better, get involved!
  rules_go, Gopher slack, github.com/golang/go

- If you work in another language, please steal all of this!

Acknowledgements

Go: Rebecca Stambler, Hana Kim, Rob Findley, Michael Matloob, Peter Weinberger, Suzy Mueller, Alan Donovan, Ian Cottrell, everyone who worked on vscode-go, gopls, go/packages, everyone who worked on tools, editors, IDE support.

rules_go: Steeve Morin, Zhongpeng Lin, Fabian Meumertzheim, everyone who contributed.

Thanks!