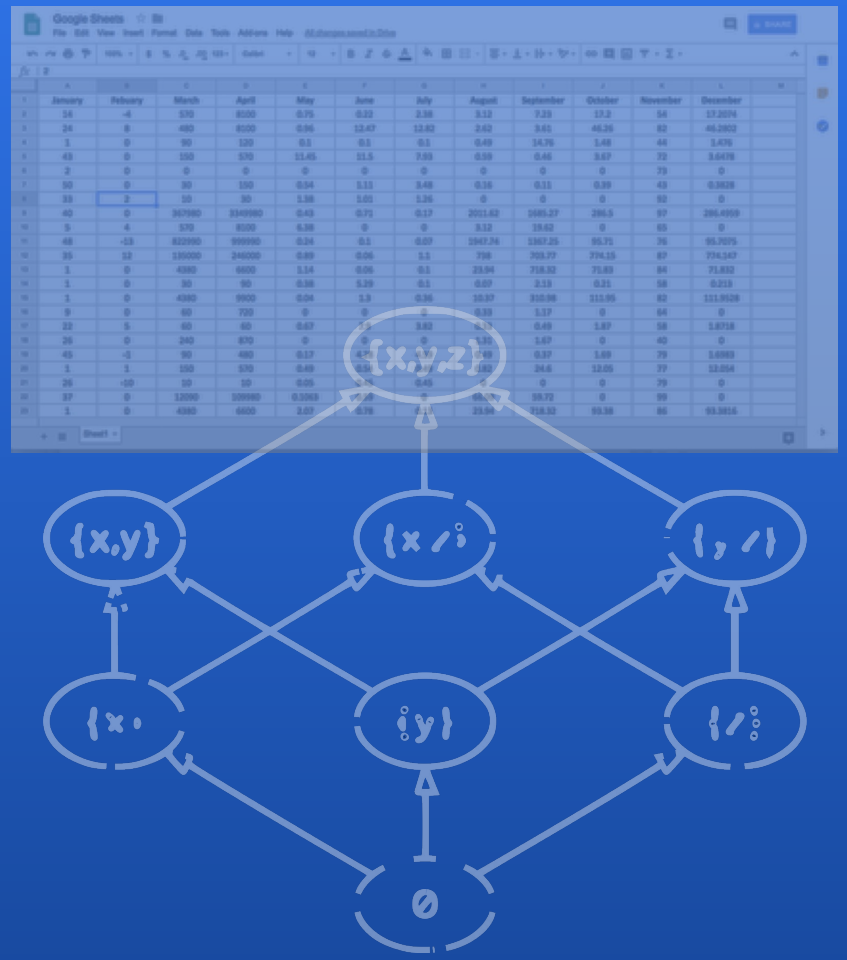
A practical guide to CUE:
patterns for everyday use

Marcel van Lohuizen and Paul Jolly
CUE Authors
@mpvl_, @_myitcv
https://cuelang.org

FOSDEM 2022 - Config Management Camp

What is CUE?



Values in CUE

CUE

```
brussels: {
    name:       "Brussels"
    population: 1.2M
    capital:    true
}
```

JSON

```
{
    "brussels": {
        "name": "Brussels",
        "population": 1200000,
        "capital": true
    }
}
```

CUE is a JSON superset.

Types in CUE

CUE

```
municipality: {
  name:       string
  population: int
  capital:    bool
}
```

Go

```
type municipality struct {
  Name       string
  Population int
  Capital    bool
}
```

Types are just values in CUE.

Constraints and Policy in CUE

CUE

```
largishCapital: {
    name:       string
    population: >1M
    capital:    true
}
```

JSON Schema

```
{
    "type": "object",
    "properties": {
        "name": {
            "type": "string",
            "format": "string"
        },
        "population": {
            "type": "number",
            "minimum": 1200000,
            "exclusiveMinimum": true
        },
        "capital": {
            "type": "boolean",
            "enum": [ true ]
        }
}
```

Types *and* constraints are values.

All configurations are defined in a single configuration space


```
municipality: {
  name:    string
  pop:     int
  capital: bool
}

largishCapital: {
  name:    string
  pop:     >1M
  capital: true
}

brussels: {
  name:    "Brussels"
  pop:     1.2M
  capital: true
}
```

The CUE Continuum

Abstract Types

Types

API

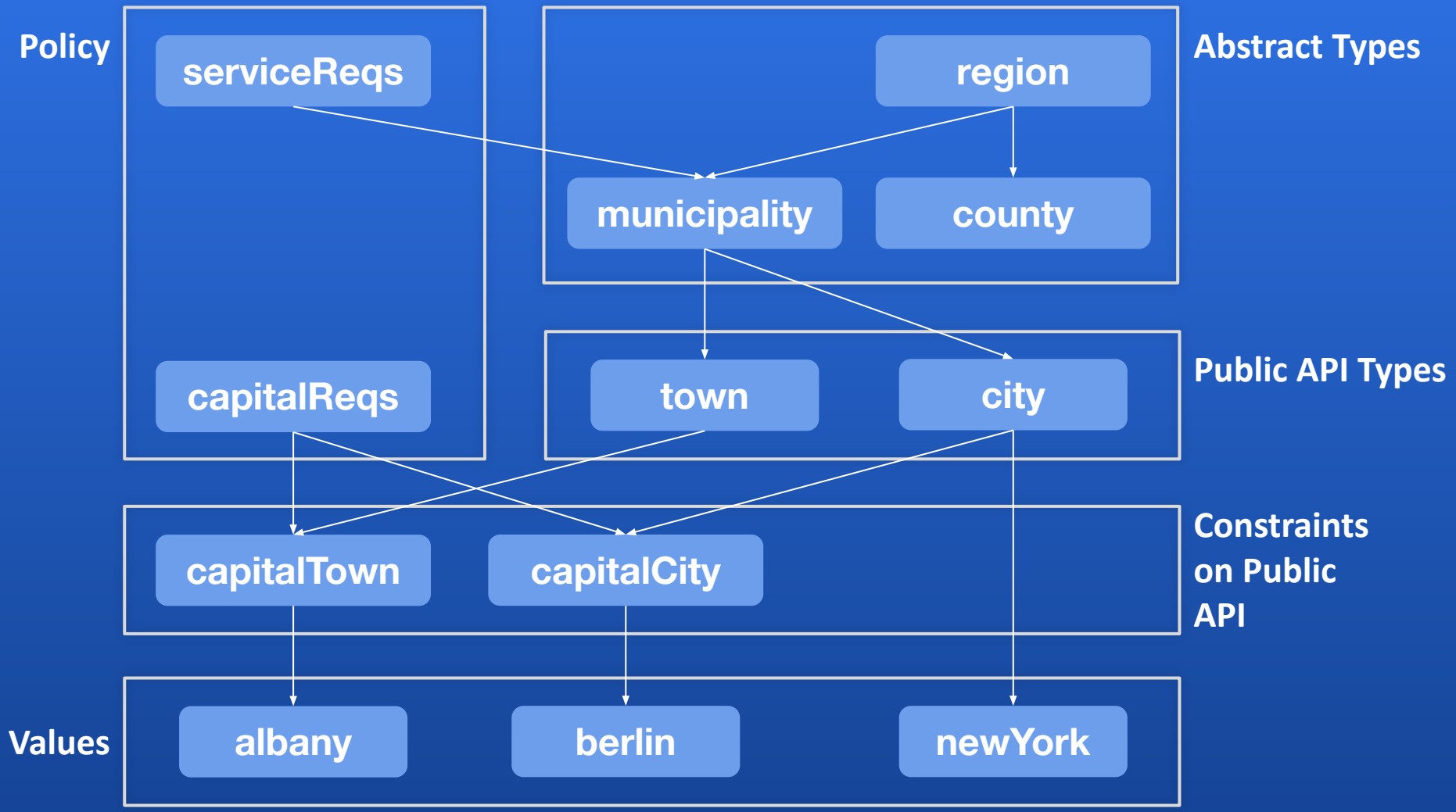
Meta Policy

Policy

Templates

Values

Cross-cutting nature of configuration



Composition is key

Config
Management
Camp

How does CUE achieve composition?

Associative

(A & B) & C ≡ A & (B & C)

Commutative

A & B ≡ B & A

Idempotent

A & A ≡ A

Which is a fancy way of saying that…

Associative

(A & B) & C ≡ A & (B & C)

ORDER DOESN'T MATTER

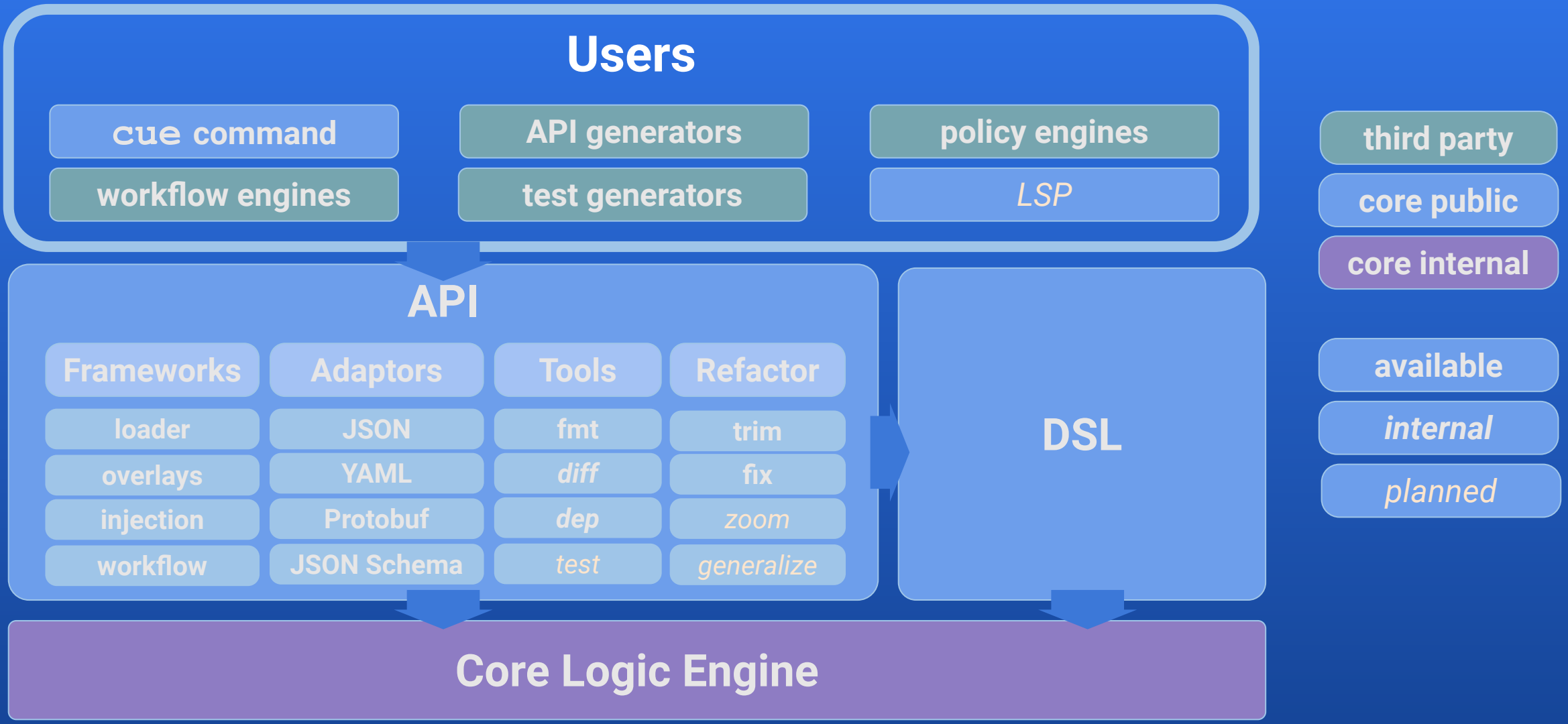
Commutative

A & B ≡ B & A

Idempotent

A & A ≡ A

CUE Components



What is CUE used for?

fact discovery

OpenAPI generation

Declarative scripting

Data validation

Configuration

Data templating

Policy checking

code generation

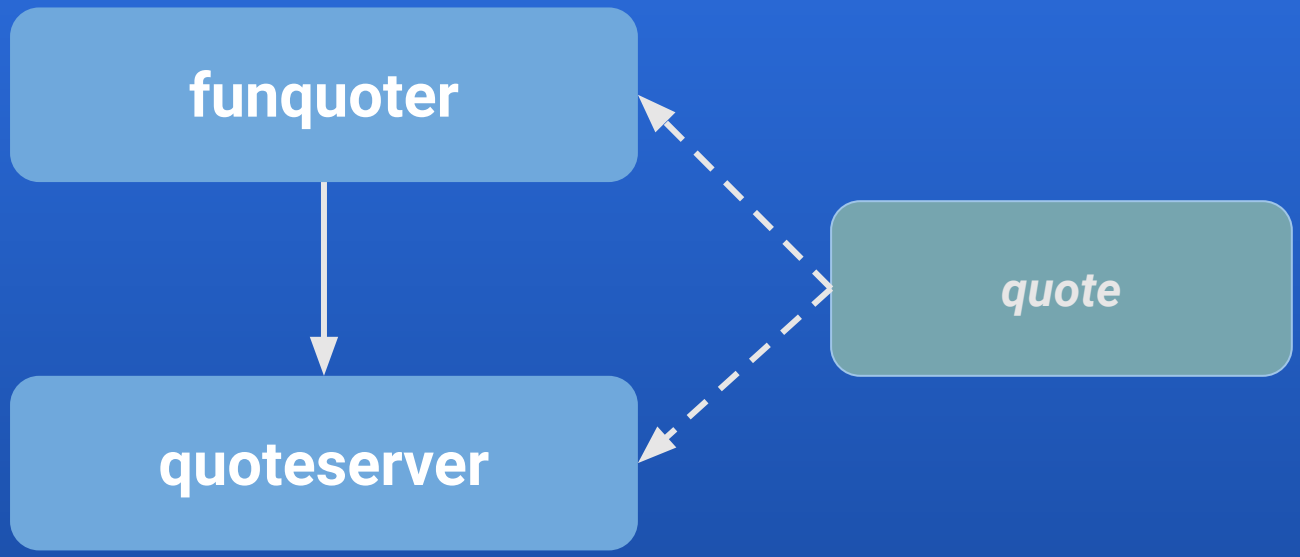
cross-language test generation

Demo time

An example of a gradual adoption of CUE
within an organization

Follow @cue_lang on Twitter!

acme.com: architecture



Demo: initial setup

Adding Validation

A single team adopts CUE for validation

- Add CUE validation rules
- Validate YAML files directly

funquoter/schema.cue – a team's constraints

```
package kube

// The K8s objects
Object:     Deployment | Service
Service:    kind: "Service"
Deployment: kind: "Deployment"

// We enforce registry usage
Deployment: spec: template: spec:
    containers: [...{
        image: =~"""
            ^k3d-registry.acme.com:5000/
            """
    }]
```

```
// Labeling and selector policy
Service: {
    metadata: labels: app: metadata.name
    spec: selector: app: metadata.name
}
Deployment: X={
    spec: template: metadata:
        labels: app: X.metadata.name
}

// Enable Prometheus monitoring.
Deployment: spec: template: metadata:
    annotations:
        "prometheus.io/scrape": "true"
```

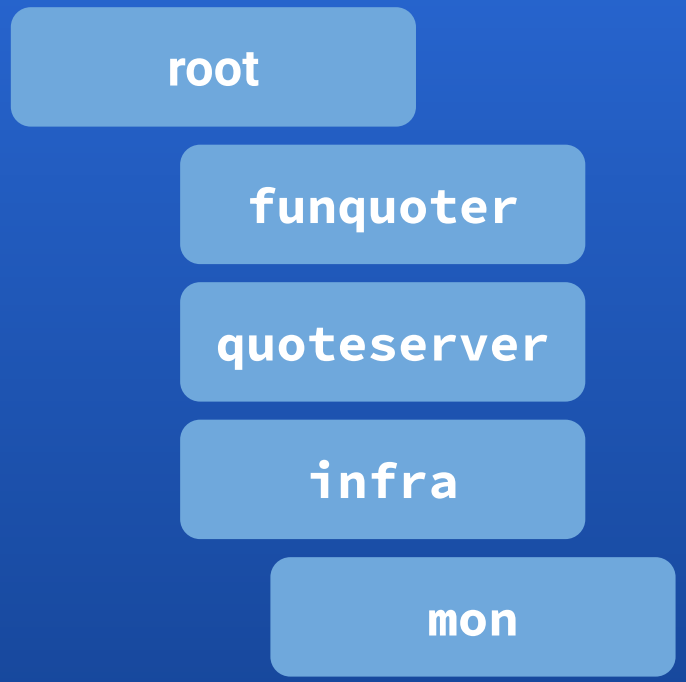
Demo: adding Validation

File organization

Sharing CUE validation and policy

- modules
- packages
- hierarchical constraints
- imports

Directory hierarchy



Pull: import other packages

infra/mon/mon.cue – adding organziation-wide constraints

```
package mon

Deployment: spec: template: metadata:
    annotations: "prometheus.io/scrape": *"true" | "false"


Deployment: spec: template: spec: containers: [...{
    livenessProbe: {
        httpGet: path: "/debug/health"
        httpGet: port: *8080 | int
        initialDelaySeconds: *40 | >10
        periodSeconds:       *3 | int
    }
}]
```

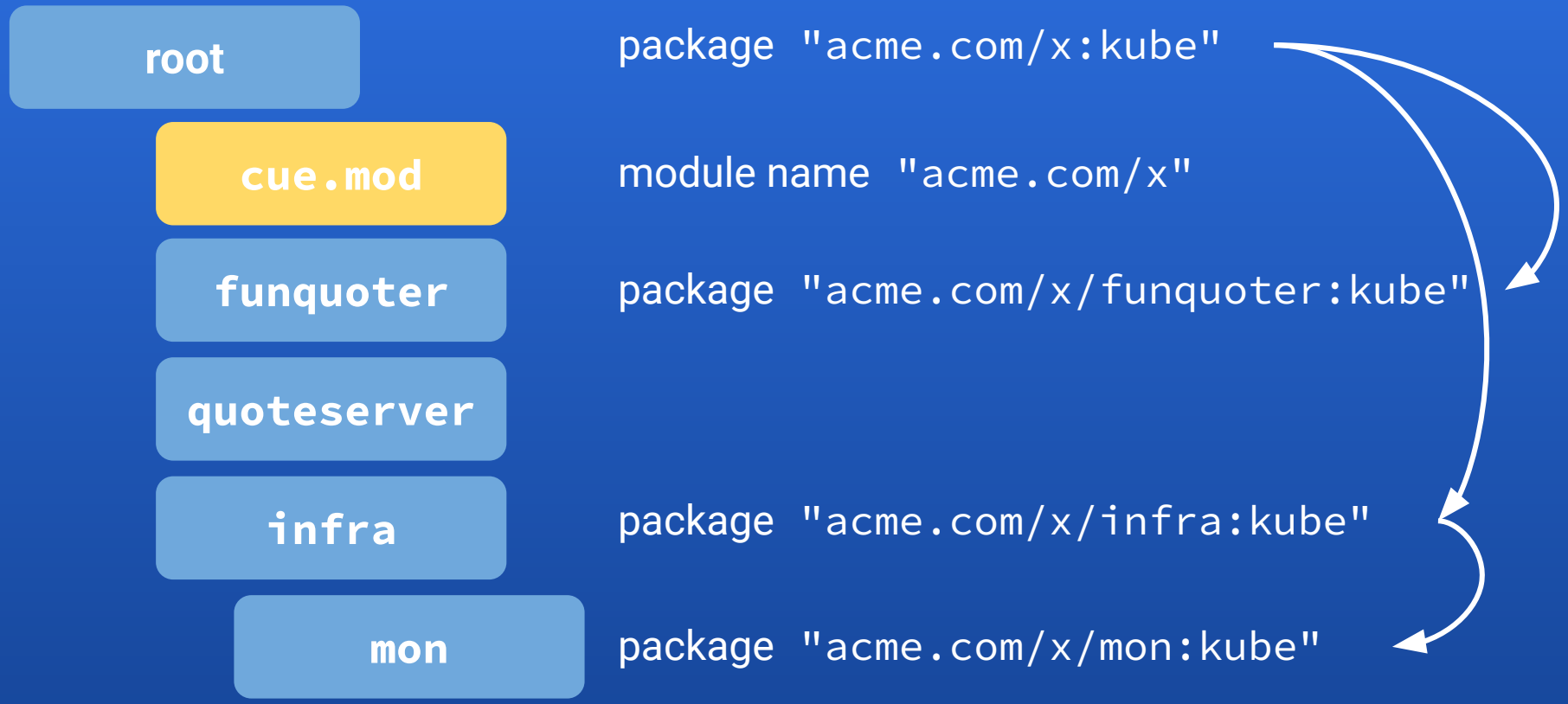
funquoter/schema.cue – the funquoter team imports mon

```
package kube

import "acme.com/x/infra/mon"

// mon.Deployment is the policy defined for Deployment as defined by the
// monitoring team.
Deployment: mon.Deployment
```

Push: package files also apply to child dirs



schema.cue – adding organiziation-wide constraints

```
package kube

import "acme.com/x/infra/mon"

// Enforce monitoring policies for all teams
Deployment: mon.Deployment
```

Demo: sharing validation and enforcing policy across teams

Using the Tooling Layer

Formalizing the process

- Locate and load the K8s YAML files
- Validate them
- Pipe them to various operations

kube_tool.cue – loading the configuration

```
package kube
import ( "encoding/yaml", "tool/file" )

globYAML: file.Glob & { glob: "*.yaml" }

open: {
    for _, f in globYAML.files {
        (f): file.Read & {
            filename: f                                  // input
            contents: _                                  // output
            _objList: yaml.UnmarshalStream(contents)     // user field
        }
    }
}
```

Package "tool/file"

```
// Glob returns a list of files.
Glob: {
    // INPUTS
    // glob specifies the pattern
    // to match files with.
    glob: !=""

    // OUTPUTS
    files: [...string]
}
```

```
// Read reads the contents of a file.
Read: {
    // INPUTS
    // filename names the file to read.
    filename: !=""

    // OUTPUTS
    // contents is the read contents.
    contents: *bytes | string
}
```

schema.cue – collating objects

```
package kube


// Set our validation template for each of the types.
objByKind: service:    [string]: Service
objByKind: deployment: [string]: Deployment
```

kube_tool.cue – linking in the validation

```
package kube

import "strings"

// Collate Kubernetes objects by kind and name.
objByKind: {
    for _, v in open for _, obj in v._objList {
        (strings.ToCamel(obj.kind)): (obj.metadata.name): obj
    }
}

// convenience
allObjects: [ for x in objByKind for y in x {y} ]
```

kube_tool.cue – Commands invokable by CUE

```
import ( "encoding/yaml", "tool/cli", "tool/exec" )

// print prints all objects as JSON.
command: print: cli.Print & {
    text: json.MarshalStream(allObjects)
}

// apply creates the objects on Kubernetes.
command: apply: exec.Run & {
    cmd:    "kubectl apply -f -"
    stdin:  yaml.MarshalStream(allObjects)
}
```

Demo: formalizing launch processes with the tooling layer

Importing schema

Making it all k8s aware

- Extract CUE templates from Go code (the SoT for k8s)
- Pushing them into the existing definitions

kube_defs.cue – linking Kubernetes schema

```
package kube

import (
    "k8s.io/api/core/v1"
    apps_v1 "k8s.io/api/apps/v1"
)

Service:     v1.#Service
Deployment:  apps_v1.#Deployment
DaemonSet:   apps_v1.#DaemonSet
StatefulSet: apps_v1.#StatefulSet
```

Demo: introduce detailed typing

Going CUE native

One of the teams decides to all in on CUE

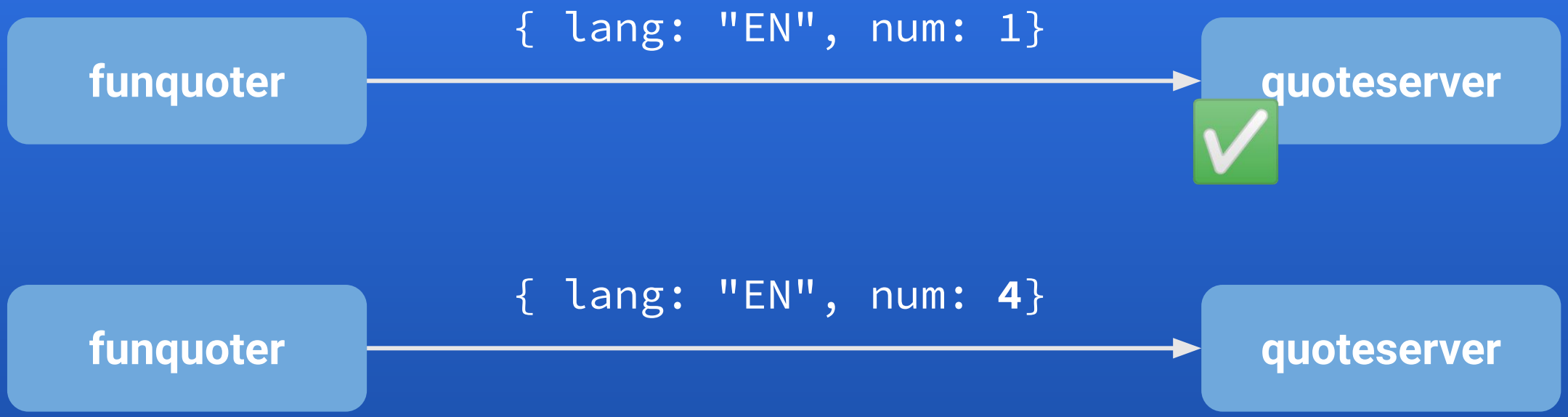
- Convert YAML and JSON to CUE
- Automatically reduce boilerplate using `cue trim`

Demo: converting YAML to CUE

Sharing validation logic

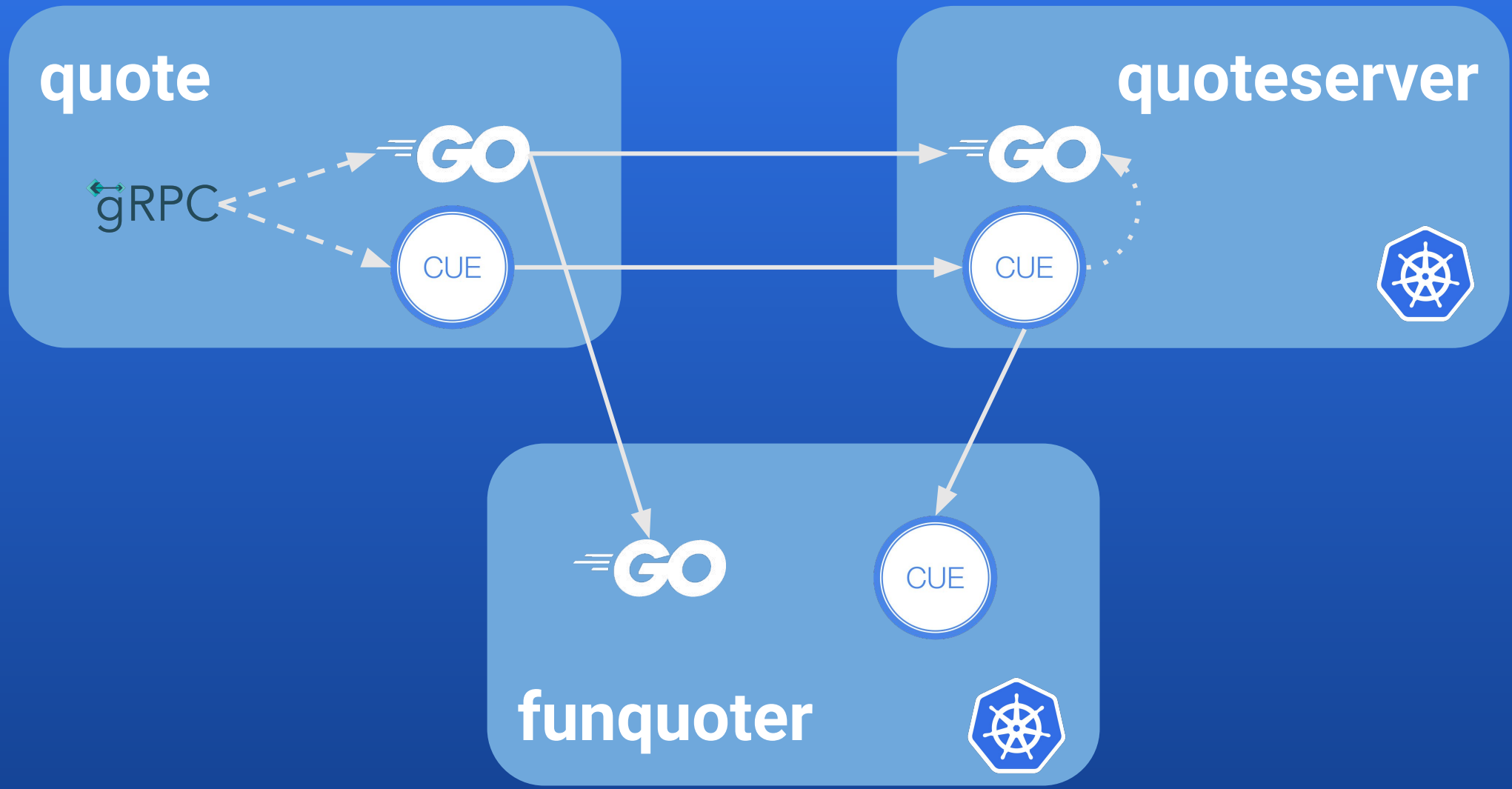
- Embed constraints in Go
- Use these same constraints to validate configuration

Team funquoter - celebrating too early?




```
error: 4 exceeds the max number of quotes 3
```

Sharing validation logic with CUE



Conclusions

State of the Union

- Open Source
- Forked from Google Jun 2021
- Over 4,300 combined GitHub stars
- Main focus: v1 + backwards compatibility guarantee
- https://github.com/cue-lang/cue

Thanks!

https://cuelang.org

@cue_lang