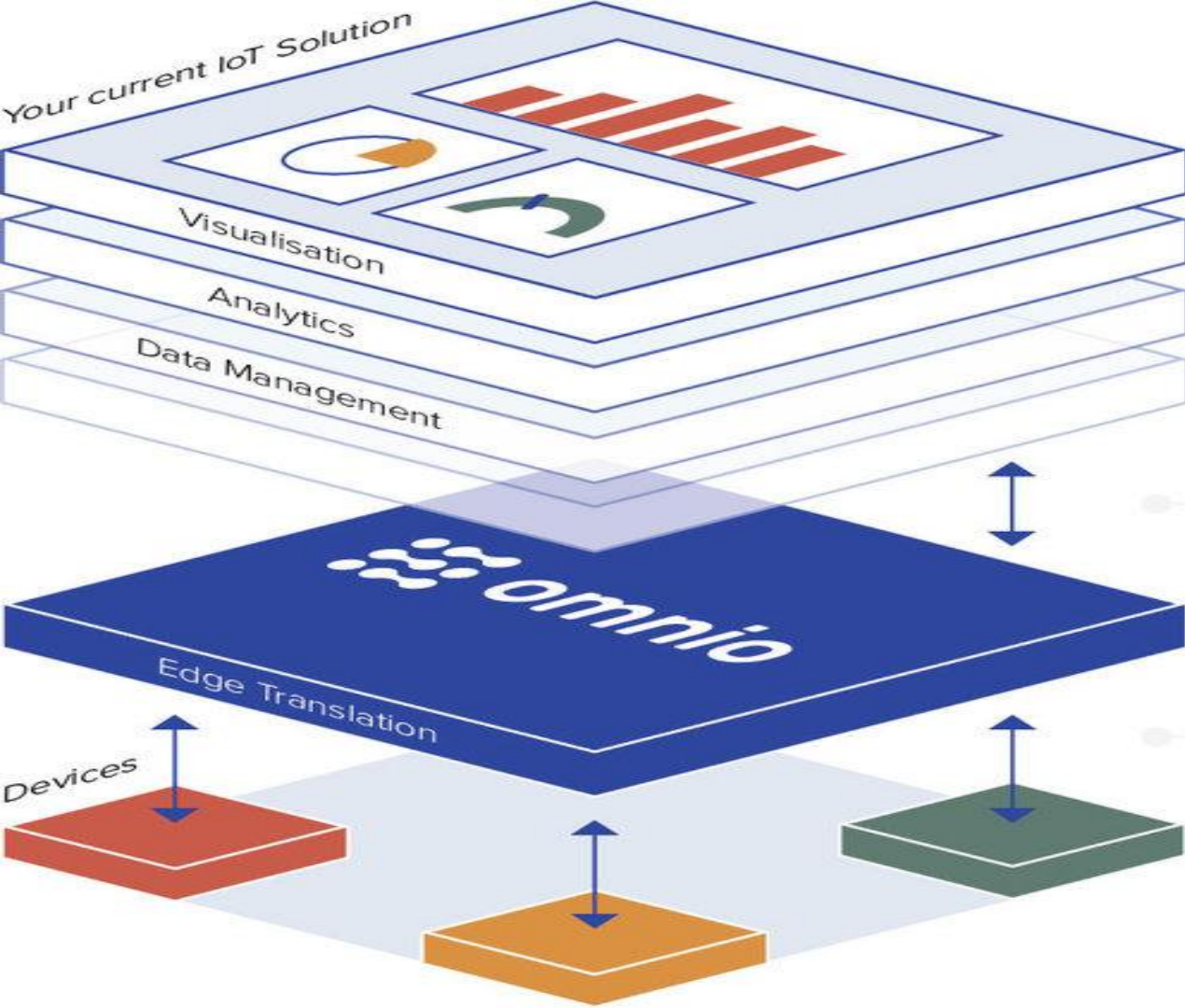


# Architecture and database access at Omnio

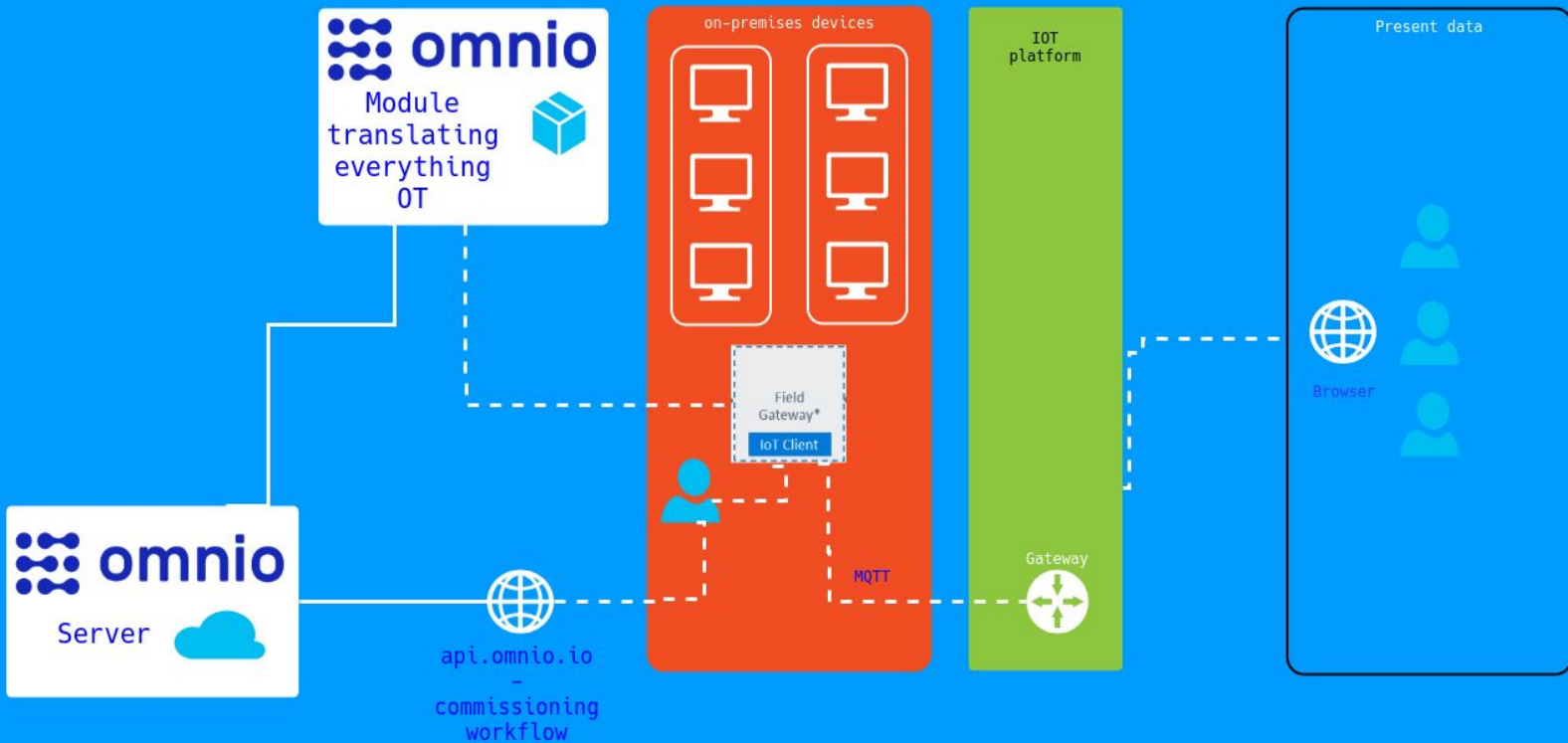
Matti Nielsen

- Responsible for the cloud
- Studying computer science at KU(BA)



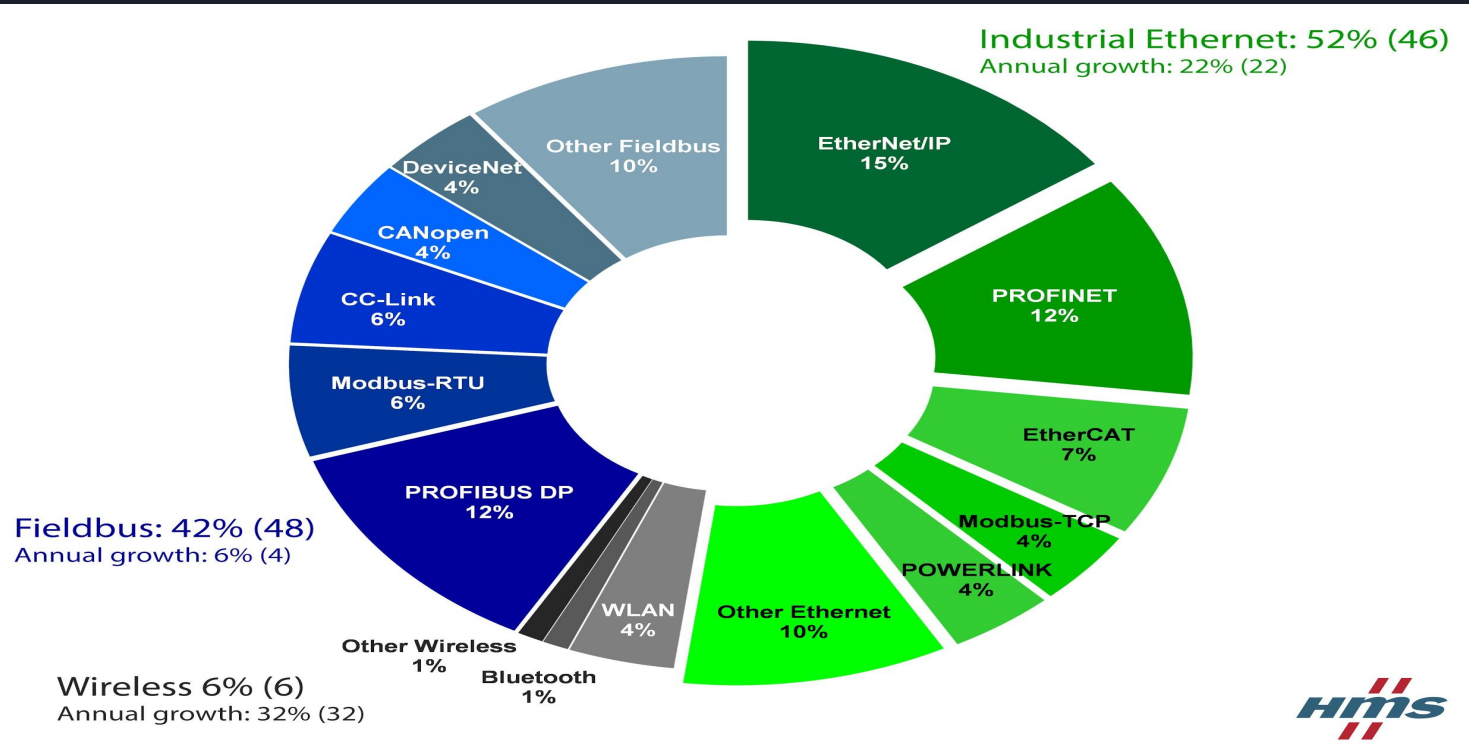
Omnio normalizes data to ensure interoperability and delivers it in readable formats e.g. JSON or OPC UA

Omnio individualizes all communication to the unique devices, protocols and data models



# Problems modeling this domain

- Any device in a customers solution can easily communicate in at least 13 different ways



# Solution?

Most protocols share the same fields but has extensions, so inheritance

1: main.go

```
1 package main
2
3 // Parent ...
4 type Parent struct {
5     Name    string `db:"name"`
6     Phone   string `db:"phone"`
7     Age     int    `db:"age"`
8     Gender  string `db:"gender"`
9 }
10
11 // Child ...
12 type Child struct {
13     Parent
14     Toys string `db:"toys"`
15 }
```

```
postgres@localhost:omnio> CREATE TABLE parent (
                             name varchar(150),
                             phone varchar,
                             age int,
                             gender varchar);
```

CREATE TABLE

Time: 0.041s

```
postgres@localhost:omnio> CREATE TABLE child(
                             toys varchar) INHERITS (parent);
```

CREATE TABLE

Time: 0.035s

```
postgres@localhost:omnio> \d child;
```

Column	Type	Modifiers
name	character varying(150)	
phone	character varying	
age	integer	
gender	character varying	
toys	character varying	

Inherits: parent

# What does database access look like?

```
37 func main() {
38     db := NewDB()
39     c := &Child{Parent: Parent{Name: "Joachim Hansen", Phone: "30-13-38-30", Age: 35,
40         Gender: "Male"}, Toys: "pink horse, baseball bat"}
41     _, err := db.Exec("INSERT INTO child (name, phone, age, gender, toys) "+
42         "VALUES ($1, $2, $3, $4, $5)", c.Name, c.Phone, c.Age, c.Gender, c.Toys)
43     if err != nil {
44         panic(err)
45     }
46 }
```

## And with more fields?

```
func insertPerson(p *Parent) {
    db := NewDB()
    _, err := db.Exec("INSERT INTO parent (firstName, lastName, cpr, account_number, "+
        "registration_number, account_type, street_name, street_number, floor, "+
        "apartment_number_or_side, co, postal_area, city, country, zipcode, "+
        "phone, mobile_phone, cvr, company_name, email) INTO VALUES($1, $2, $3, $4, "+
        "$5, $6, $7, $8, $9, $10, $11, "+"$12, $13"+"$14, $15, $16, $17, $18, $19, $20)",
        p.FirstName, p.LastName, p.CPR, p.AccountNumber, p.RegistrationNumber, p.AccountType,
        p.StreetName, p.StreetNumber, p.Floor, p.ApartmentNumberOrSide, p.CO, p.PostalArea,
        p.City, p.Country, p.Zipcode, p.Phone, p.MobilePhone, p.CVR, p.CompanyName, p.Email)
    if err != nil {
        panic(err)
    }
}
```



# Solutions?

- Just writing it out like that for all entities
- Gorm (Code first ORM )
- SQLBoiler ( Generates entire model layer from database)
- SQLX (Wrapper for database/sql providing extensions)
- Another framework? Something else?





# Improving the solution with `github.com/fatih/structs`

```
65 func insertPerson2(p *Parent) {  
66     db := NewDB()  
67     values := structs.Values(p)  
68     _, err := db.Exec("INSERT INTO parent (firstName, lastName, cpr, account_number,"+  
69         "registration_number, account_type,street_name, street_number, floor,"+  
70         "apartment_number_or_side, co, postal_area, city, country, zipcode, "+  
71         "phone, mobile_phone, cvr, company_name, email) INTO VALUES($1, $2, $3, $4, "+  
72         "$5, $6, $7, $8, $9, $10, $11,"+"$12, $13"+"$14, $15, $16, $17, $18, $19, $20)", values...)  
73     if err != nil {  
74         panic(err)  
75     }
```

# Getting rid of the query string as well

```
51 // Parent where ag = auto-generated, perm = permissions
52 type Parent struct {
53     ID      string `db:"id" ag:"true" perm:"Admin"`
54     Name    string `db:"name" perm:"admin,end-user"`
55     Phone   string `db:"phone" perm:"admin,end-user"`
56     Age     int    `db:"age" perm:"admin,end-user"`
57     Gender  string `db:"gender" perm:"admin,end-user"`
58 }
59
60 // Child ...
61 type Child struct {
62     Parent
63     Toys string `db:"toys" perm:"admin,end-user"`
64 }
```

```
67 type TableName string
68
69 var insertQueryMap map[TableName]string
70
71 func genQuery(a interface{}) string {
72     query := ""
73     // ... pretty ugly reflection on 'a', but only called 1 time
74     return query
75 }
76
77 func init() {
78     insertQueryMap = map[TableName]string{}
79     insertQueryMap["child"] = genQuery(&Child{})
80 }
81
82 func insertPerson2(c *Child) {
83     db := NewDB()
84     values := structs.Values(c) // uses reflection for embedded structs
85     _, err := db.Exec(insertQueryMap["child"], values...)
86     // ... err handling
87 }
```

# Getting relationships with `jmoiron/sqlx`

```
63 func (dm *DeviceModel) GetOne(id string, fields string) (*Device, error) {
64     var d Device
65     err := dm.db.Get(&d, "SELECT "+fields+" FROM device WHERE id = $1", id)
66     // ... handle err
67     err := dm.db.Select(&d.Profiles, "SELECT * FROM profile WHERE device_id = $1", id)
68     // ... handle err
69     return &d, err
70 }
```

# Handling several big dependency graphs for testing

Packages we use:

- [github.com/vburenin/ifacemaker](https://github.com/vburenin/ifacemaker)
- [github.com/golang/mock/gomock](https://github.com/golang/mock/gomock)
- [github.com/golang/mock/mockgen](https://github.com/golang/mock/mockgen)
- [gopkg.in/DATA-DOG/go-sqlmock.v1](https://gopkg.in/DATA-DOG/go-sqlmock.v1)
- [github.com/google/go-cloud/wire](https://github.com/google/go-cloud/wire)

```
22 logger := logs.NewDebugLogger()
23 db := models.NewDB(conf, logger)
24 iDeviceModel := models.NewDeviceModel(db, logger)
25 iProtocolModel := models.NewProtocolModel(db, logger)
26 iDeviceStore := stores.NewDeviceStore(iDeviceModel, iProtocolModel,
27     logger)
28 iDeviceController := controllers.NewDeviceController(iDeviceStore,
29     logger)
30 iProfileModel := models.NewProfileModel(db, logger)
31 iProfileStore := stores.NewProfileStore(iDeviceModel, iProfileModel,
32     iProtocolModel, logger)
33 iProfileController := controllers.NewProfileController(iProfileStore, logger)
34 iInstallationModel := models.NewInstallationModel(db, logger)
35 iDeviceInstallationModel := models.NewDeviceInstallationModel(db, logger)
36 iInstallationStore := stores.NewInstallationStore(iInstallationModel,
37     iDeviceInstallationModel, logger)
38 iInstallationController := controllers.NewInstallationController(iInstallationStore,
39     conf, logger)
40 iUserModel := models.NewUserModel(db, logger)
41 iUserStore := stores.NewUserStore(iUserModel, iInstallationModel, logger)
42 iUserController := controllers.NewUserController(iUserStore, logger)
43 iCtrls := controllers.ProvideCtrls(iDeviceController, iProfileController,
44     iInstallationController, iUserController)
```




# Running `go generate -x ./...`

Maybe add it to a git pre-commit hook together with `go test ./...`

```
wire ←
ifacemaker -f ctrls.go -s Ctrls -i ICtrls -c DONT EDIT: Auto generated -p controllers -o ictrls.go
ifacemaker -f device.go -s DeviceController -i IDeviceController -c DONT EDIT: Auto generated -p controllers -o idevic
ifacemaker -f installation.go -s InstallationController -c DONT EDIT: Auto generated -i IInstallationController -p con
ifacemaker -f profile.go -s ProfileController -i IProfileController -c DONT EDIT: Auto generated -p controllers -o ipr
ifacemaker -f user.go -s UserController -c DONT EDIT: Auto generated -i IUserController -p controllers -o iusercontrol
wire
mockgen -destination ilogger.go -package logs github.com/omniodot/server/logs Logger
mockgen -destination autogen_mock.go -package models github.com/omniodot/server/models IDeviceModel, IDeviceInstallati
ifacemaker -f device.go -s DeviceModel -i IDeviceModel -p models -c DONT EDIT: Auto generated -o idevicemodel.go
ifacemaker -f deviceinstallation.go -s DeviceInstallationModel -i IDeviceInstallationModel -c DONT EDIT: Auto generate
ifacemaker -f installation.go -s InstallationModel -i IInstallationModel -p models -c DONT EDIT: Auto generated -o iin
ifacemaker -f profile.go -s ProfileModel -c DONT EDIT: Auto generated -i IProfileModel -p models -o iprofilemodel.go
ifacemaker -f protocol.go -s ProtocolModel -i IProtocolModel -p models -c DONT EDIT: Auto generated -o iprotocolmodel.
ifacemaker -f user.go -s UserModel -i IUserModel -c DONT EDIT: Auto generated -p models -o iusermodel.go
wire
ifacemaker -f device.go -s DeviceStore -i IDeviceStore -c DONT EDIT: Auto generated -p stores -o idevicestore.go
mockgen -destination autogen_mock.go -package stores github.com/omniodot/server/stores IDeviceStore, IProfileStore, IUS
ifacemaker -f installation.go -s InstallationStore -i IInstallationStore -c DONT EDIT: Auto generated -p stores -o iin
ifacemaker -f profile.go -s ProfileStore -i IProfileStore -c DONT EDIT: Auto generated -p stores -o iprofilestore.go
ifacemaker -f user.go -s UserStore -i IUserStore -c DONT EDIT: Auto generated from stores/installation.go -p stores -o
wire
```





To mock entire dependency graphs, all constructors need to take interfaces as arguments and return interfaces

We use: [github.com/vburenin/ifacemaker](https://github.com/vburenin/ifacemaker) to auto-generate all of the interfaces required

```
1 //go:generate ifacemaker -f <file>.go -s <struct> -c "DONT EDIT: Auto generated" -i <interface> -p <package> -o <output_file>.go
2
```

```
1 // DONT EDIT: Auto generated
2 package models
3
4 type IModbusProfileModel interface {
5     // Delete ...
6     Delete(id string) error
7     // Create ...
8     Create(modbusProfile *ModbusProfile) (id string, err error)
9     // GetOne ...
10    GetOne(id string, fields string) (*ModbusProfile, error)
11    // Get ...
12    Get(fields string, where string, values []interface{}) ([]ModbusProfile, error)
13    // GetBits retrieves all of the bits that has the id as it's parent. This
14    // could e.g. be a bitmask or some other type using our hierarchy.
15    GetBits(id string) (profiles []ModbusProfile, err error)
16 }
```



# Using mockgen to generate mocks for all of your unit tests

```
1 go:generate mockgen -destination autogen_mocks.go -package stores github.com/omniodot/server/stores IDeviceStore,IProfileStore,IUserStore,IInstallationStore
```

```
24 // NewMockIProfileStore creates a new mock instance
25 func NewMockIProfileStore(ctrl *gomock.Controller) *MockIProfileStore {
26     mock := &MockIProfileStore{ctrl: ctrl}
27     mock.recorder = &MockIProfileStoreMockRecorder{mock}
28     return mock
29 }
30
31 // NewMockIDeviceStore creates a new mock instance
32 func NewMockIDeviceStore(ctrl *gomock.Controller) *MockIDeviceStore {
33     mock := &MockIDeviceStore{ctrl: ctrl}
34     mock.recorder = &MockIDeviceStoreMockRecorder{mock}
35     return mock
36 }
37
```



# `google/go-cloud/wire` Completely auto-generates all of these dependencies

Setting up all of the constructors in a set, and wire will figure out their initialization order:

```
31 // ModbusProfileModelMockSet ...
32 var ModbusProfileModelMockSet = wire.NewSet(NewModbusProfileModel,
33     ProvideMockLogger,
34     ProvideMockSQLXForModbusProfileModel)
35
36 func initializeModbusProfileModel(ctx context.Context, ctrl *gomock.Controller) (IModbusProfileModel, error) {
37     wire.Build(ModbusProfileModelMockSet)
38     return &ModbusProfileModel{}, nil
39 }
```

By just running `wire` we generate this mocked dependency graph:

```
17 func initializeModbusProfileModel(ctx context.Context, ctrl *gomock.Controller) (IModbusProfileModel, error) {
18     db := ProvideMockSQLXForModbusProfileModel()
19     logger := ProvideMockLogger(ctrl)
20     iModbusProfileModel := NewModbusProfileModel(db, logger)
21     return iModbusProfileModel, nil
22 }
```





## Using a fully mocked dependency graph for the modbus model unit tests

```
10 func setupModbusProfileModel(t *testing.T) (IModbusProfileModel, string) {
11     ctrl := gomock.NewController(t)
12     fields := FieldMap["profile_modbus"][AdminRole]
13     ctx := context.Background()
14     modbusProfileModel, err := initializeModbusProfileModel(ctx, ctrl)
15     if err != nil {
16         t.Fatalf("Could not create the modbusProfileModel, err: %+v\n", err)
17     }
18     return modbusProfileModel, fields
19 }
20
```



Presentation at:  
[github.com/denlillemand/presentations/go-cph](https://github.com/denlillemand/presentations/go-cph)