# Shackbus

interconnecting your Shack

Tobias Wellnitz
@DH1TW


EDIR
PEPA-3 CONTEST CLUB

# Today we have:

- Proprietary Protocols
- Legacy Hardware
- No / very limited interoperability


■ ■ ■

■ ■ ■

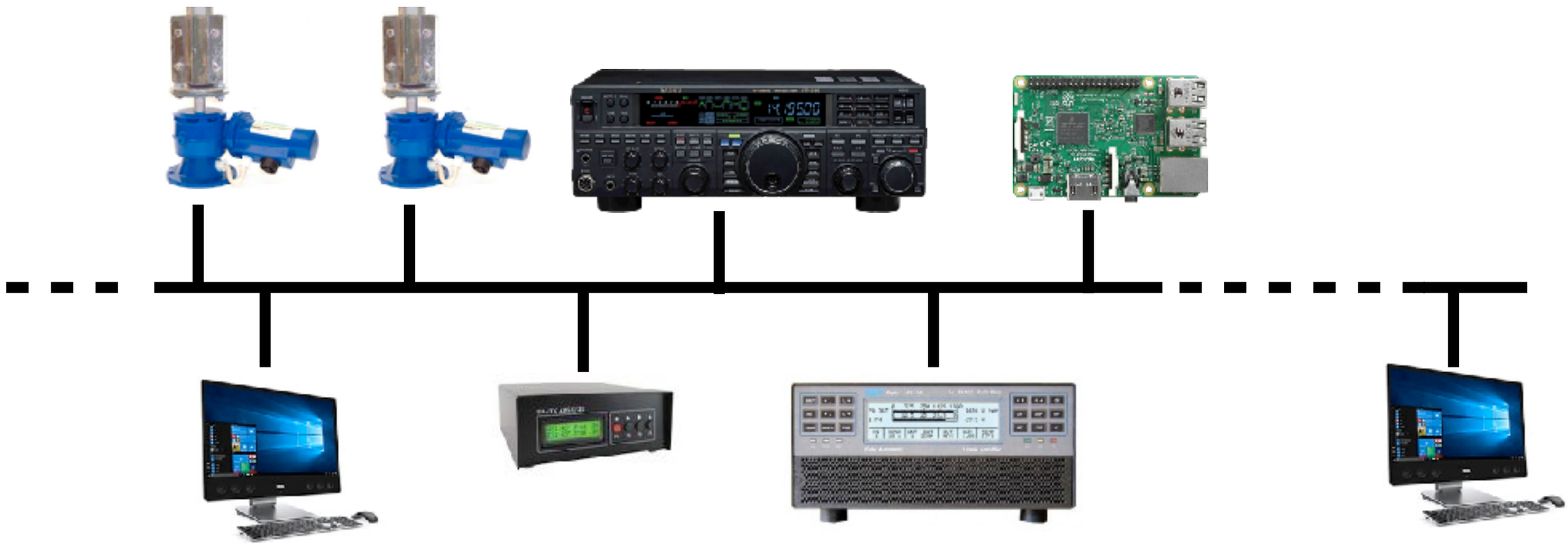
■ ■ ■

# What we want:

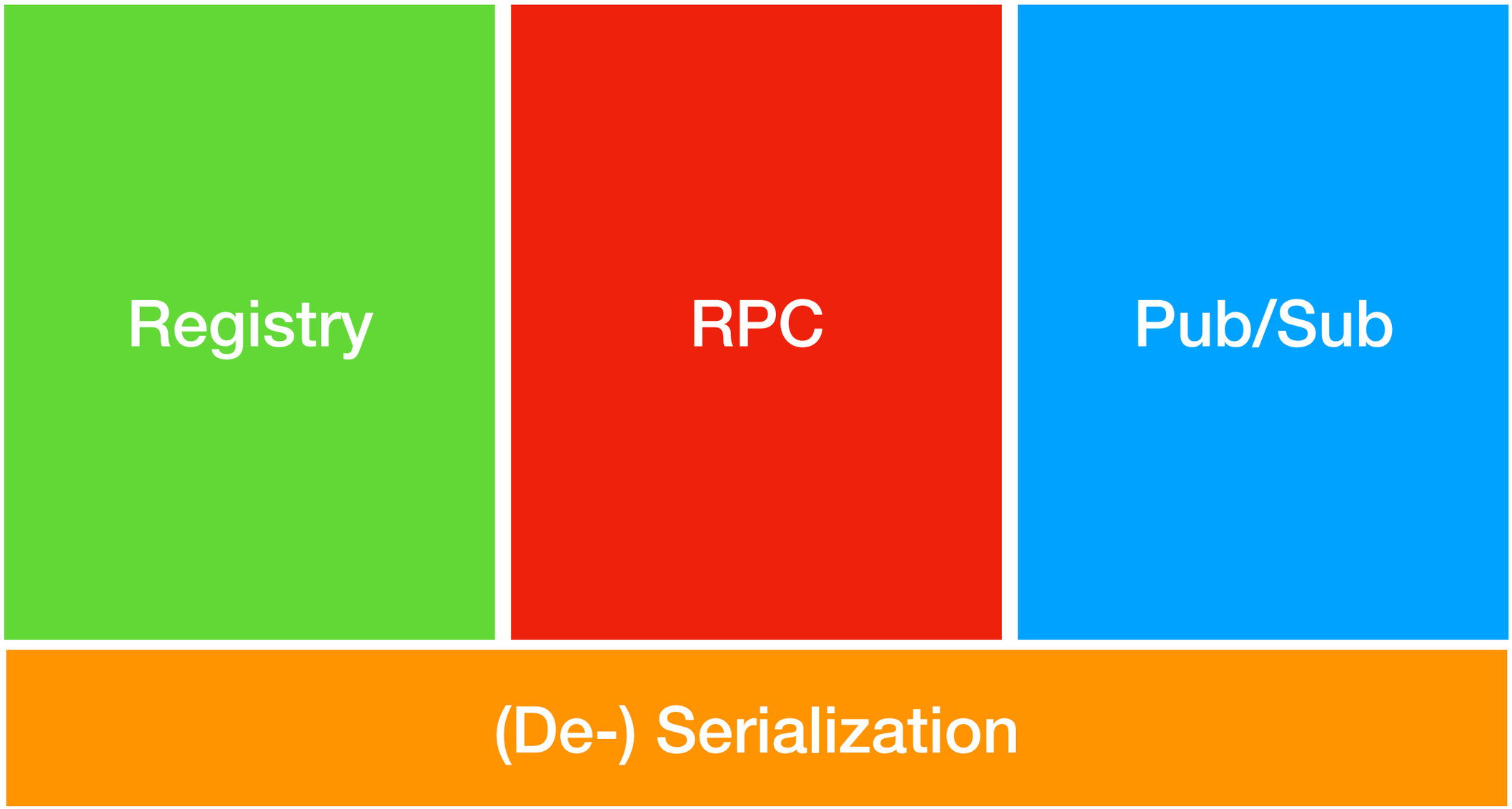
## Interconnected devices in our Shacks!

**Protocol that is/has:**

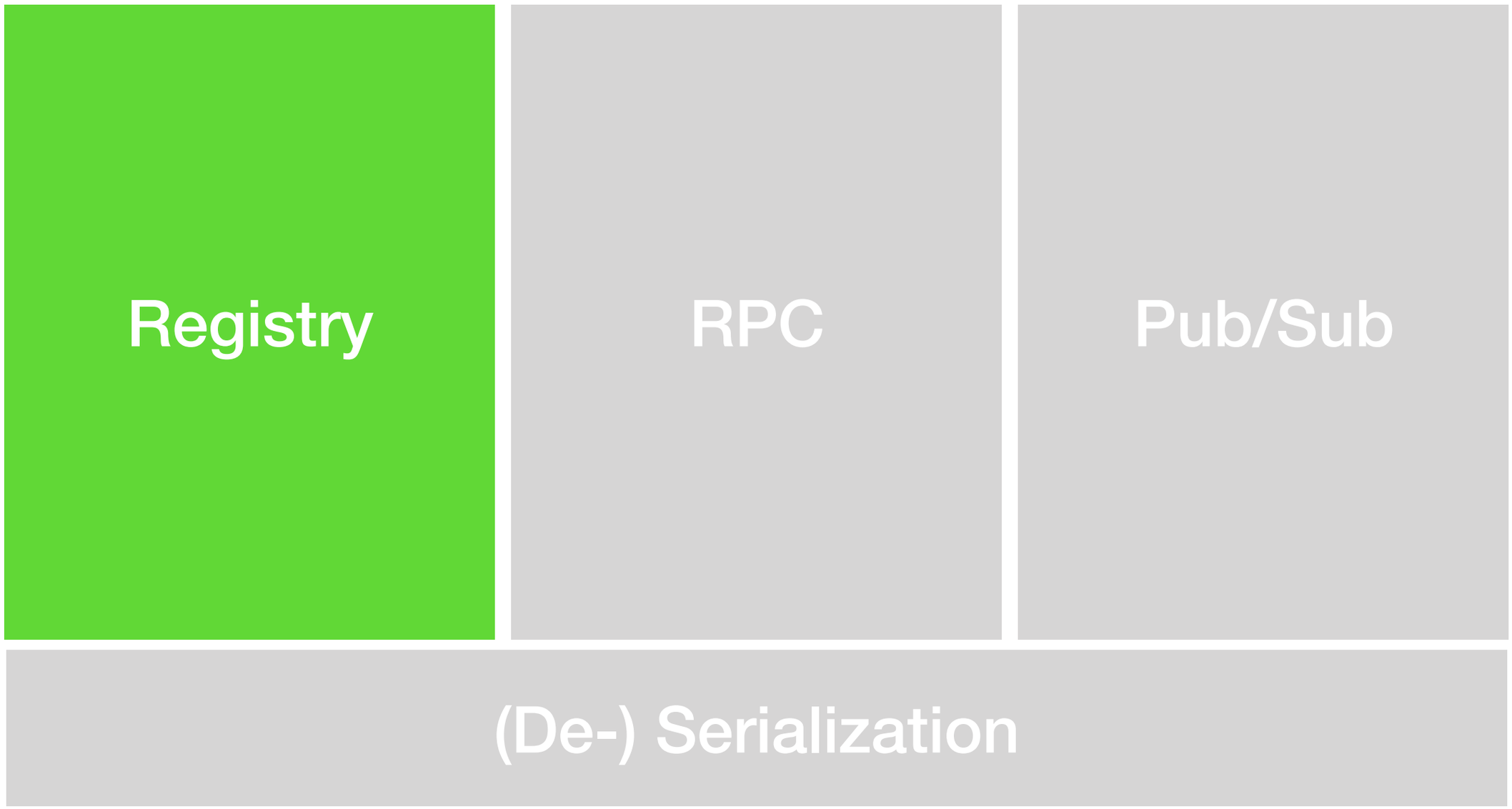
- **Creative Common**
- **Vendor neutral**
- **Open source implementations**



# The building blocks



# The building blocks



# Registry

## We need:

- a place where devices an register / de-register
- a timeout for dead devices
- fail safe / redundant implementation

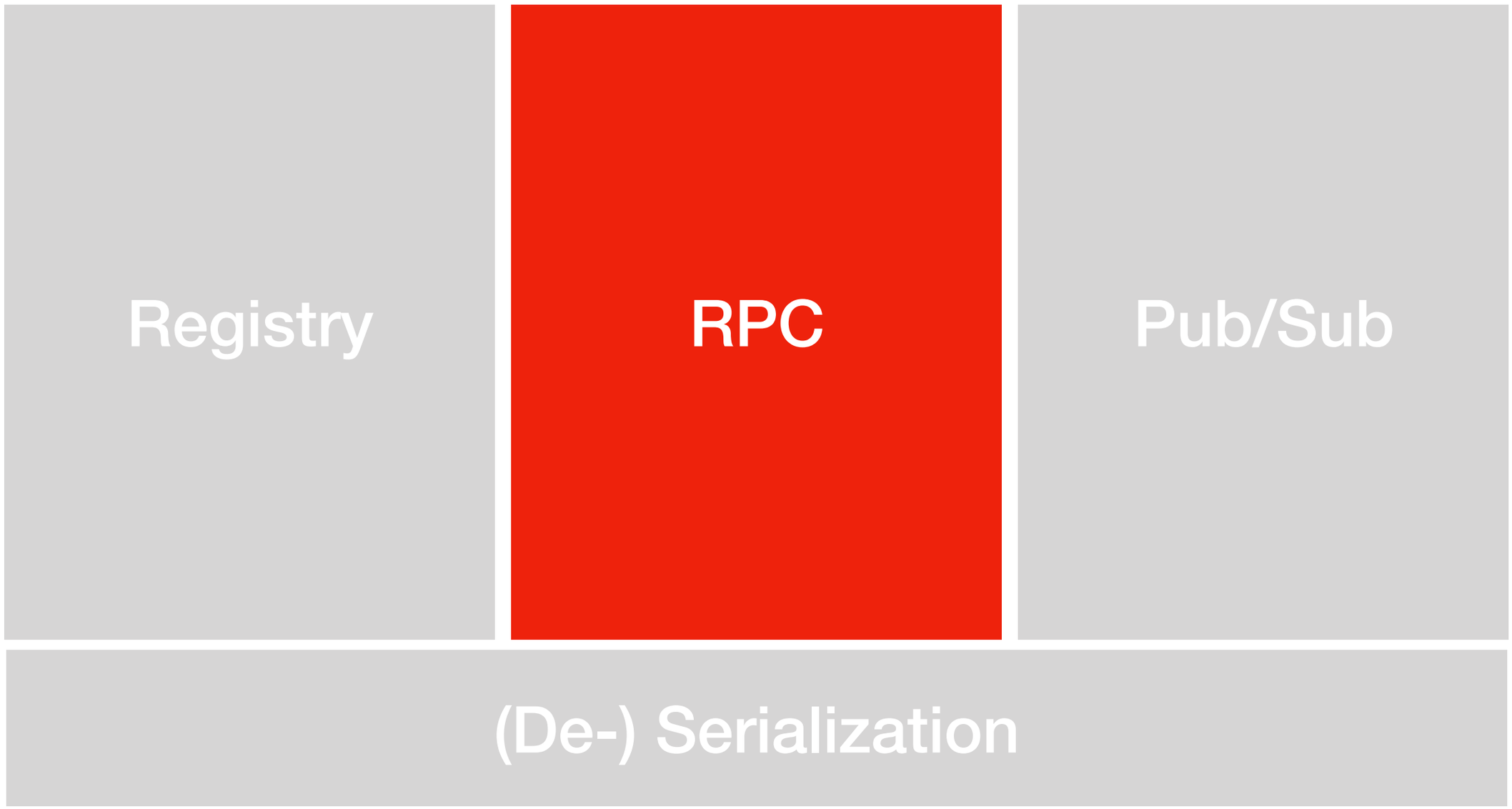
- **an API to query**
  - which devices are available
  - what their capabilities are
  - how we can reach them

## Example:

**mDNS, Multicast UDP, Distributed Key/Value Stores with HTTP/DNS API, …**


```
$ curl http://localhost:8500/v1/catalog/service/rotator
[{"Node":"Raspi1","Address":"192.168.1.55","ServiceID":"tower1", \
 “ServiceName”:"tower1","ServiceTags":["rotators"],"ServicePort":80}]
```

# The building blocks



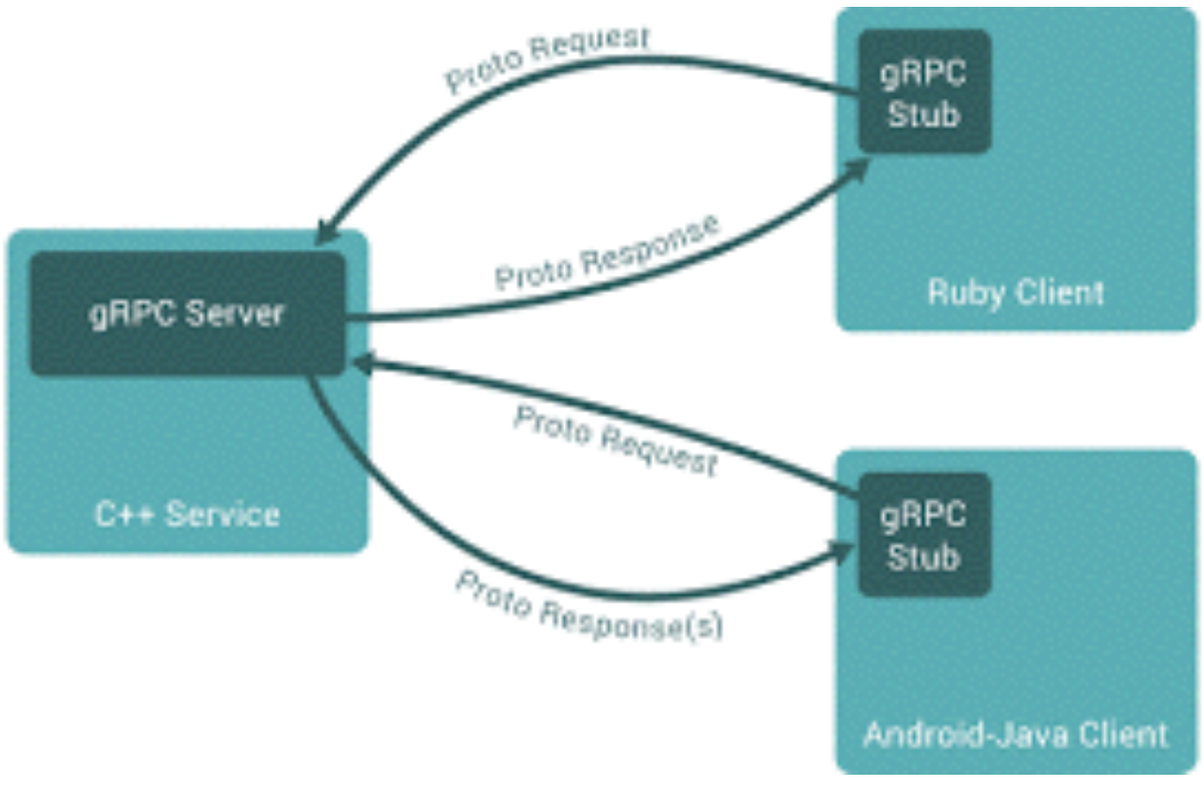
# Remote Procedure Call (RPC)

## We need:

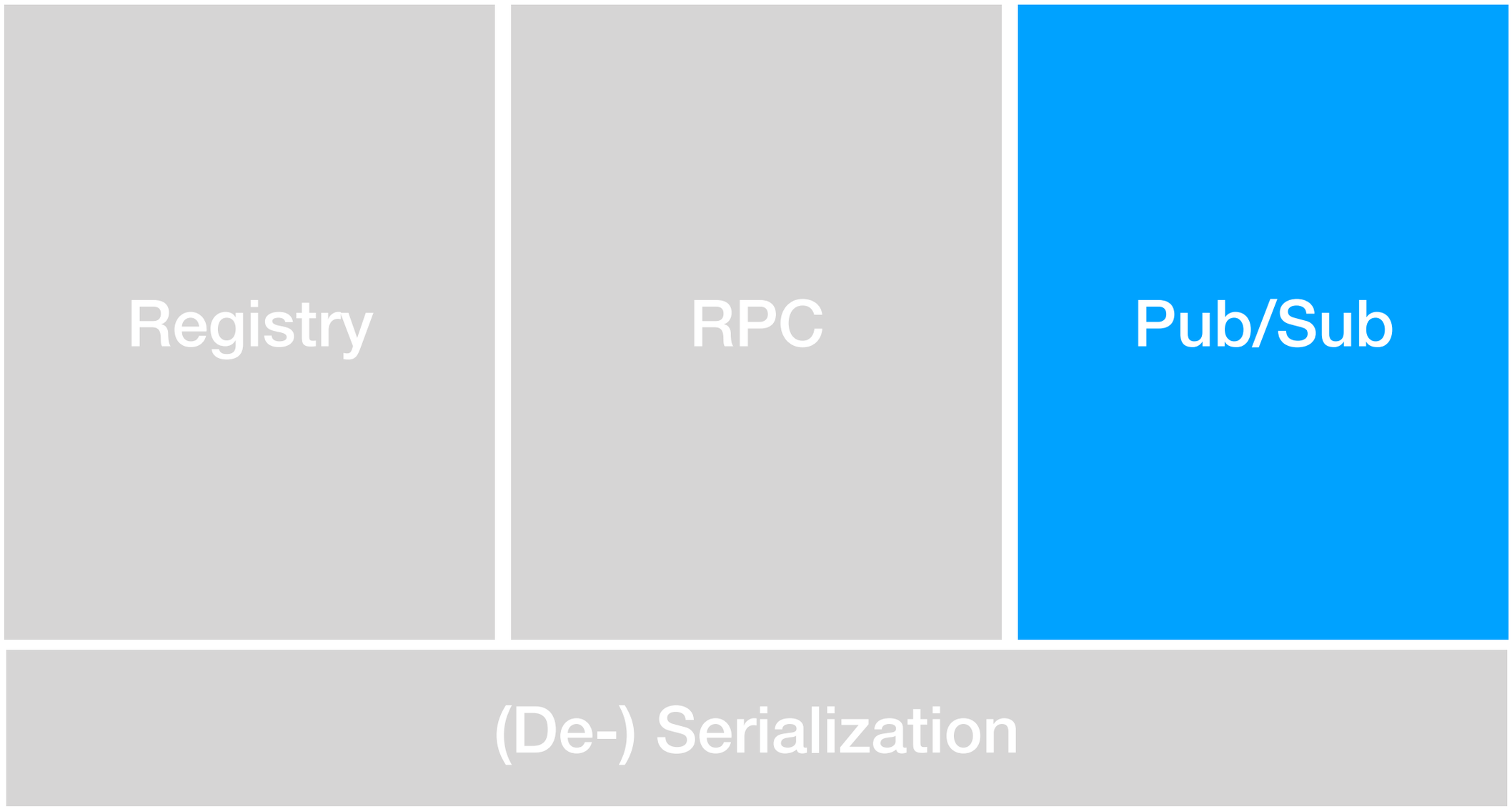
- to synchronously call a function on a remote node
- to get feedback if the call was successful or not
- a language independent implementation

## Example:

**HTTP, gRPC, Thrift, Twrip, NATS, XML-RPC, ZeroMQ…**



# The building blocks



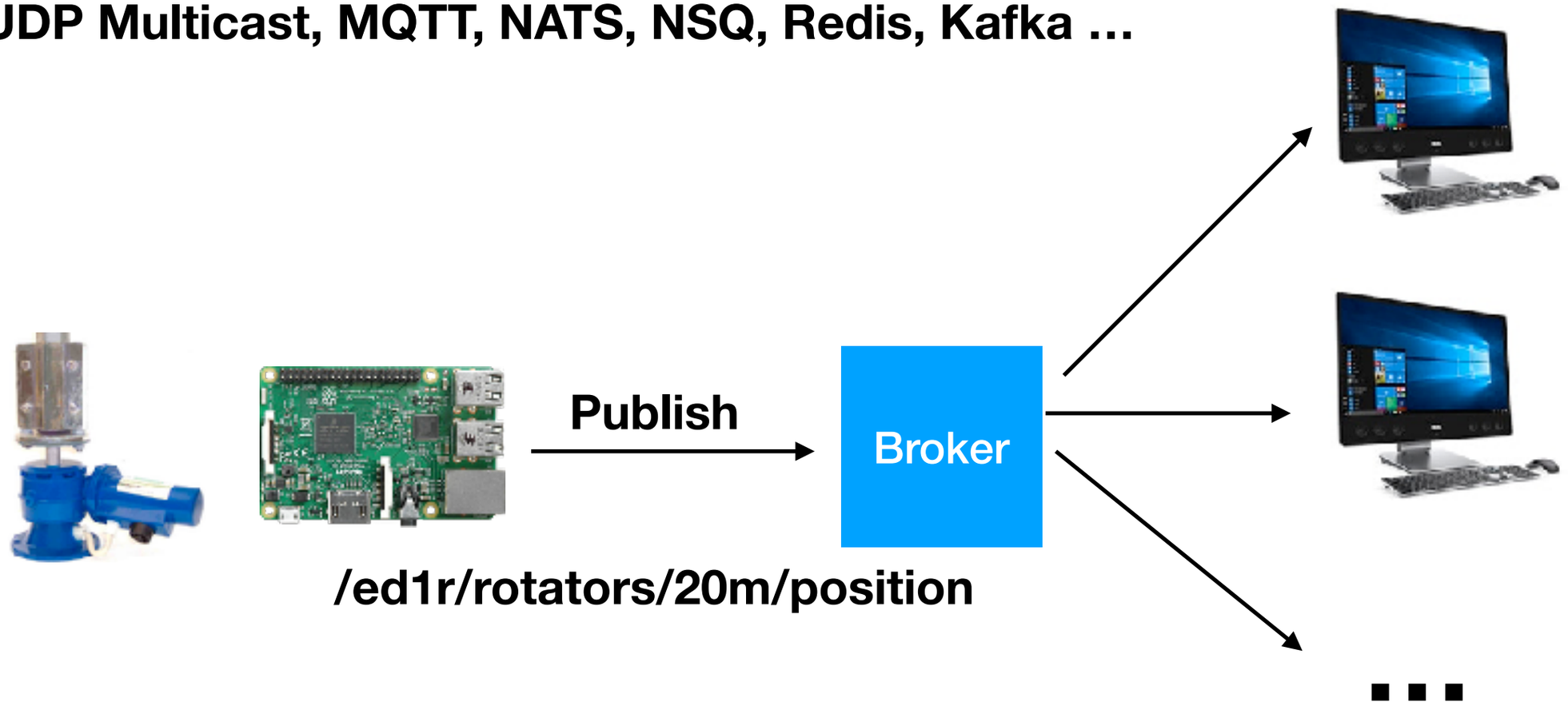
# Publish/Subscribe (Pub/Sub)

## We need:

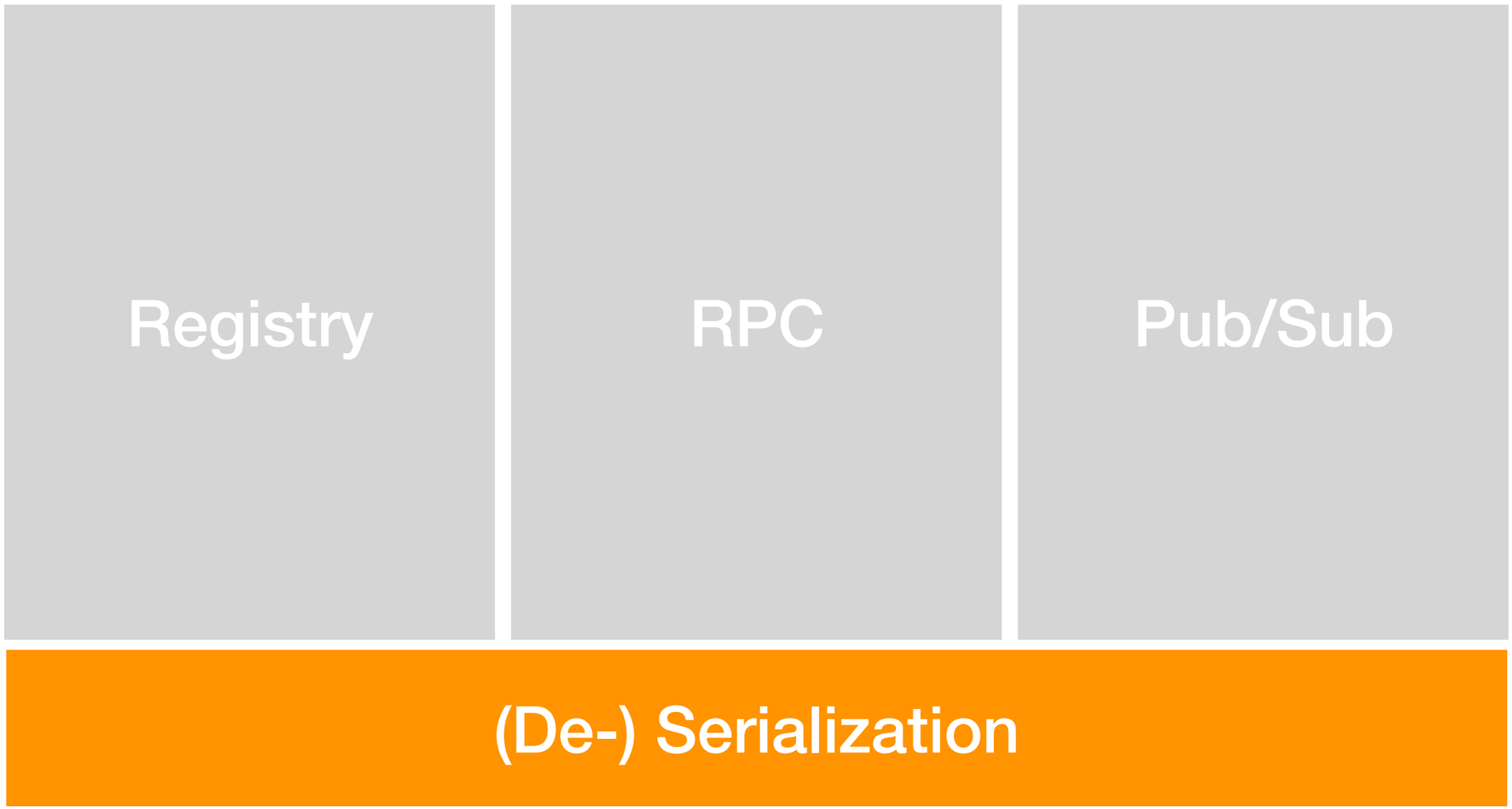
- to asynchronously publish and subscribe to events
- NOT to know who is subscribed or has received the event message
- separate messages by topics
- a language independent implementation

## Example:

**UDP Multicast, MQTT, NATS, NSQ, Redis, Kafka …**



# The building blocks



# (De-) Serialization

## We need:

- to have a schema
- to transmit binary data
- to autogenerate boilerplate code
- to have backwards compatibility
- a language independent implementation

## Example:

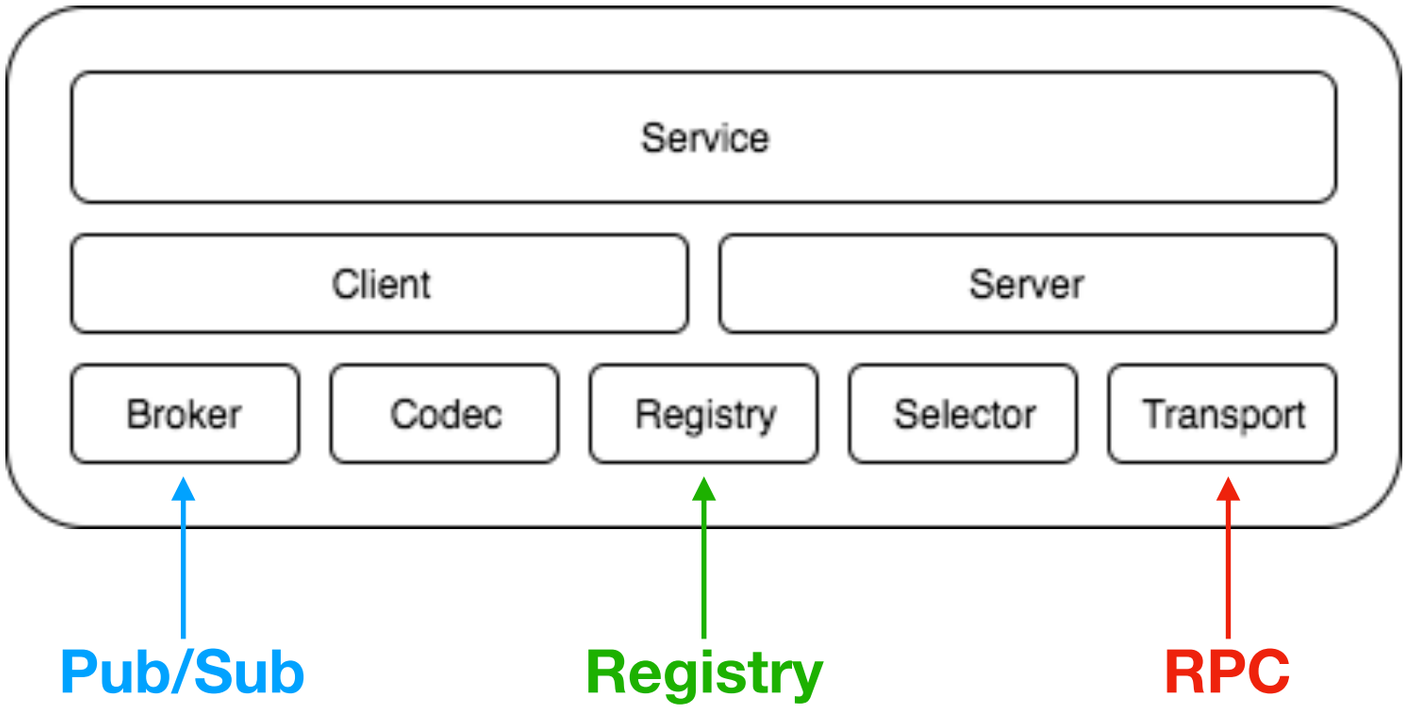
**Protocol buffers, Thrift, Capt'n Proto, Fastbuffers…**

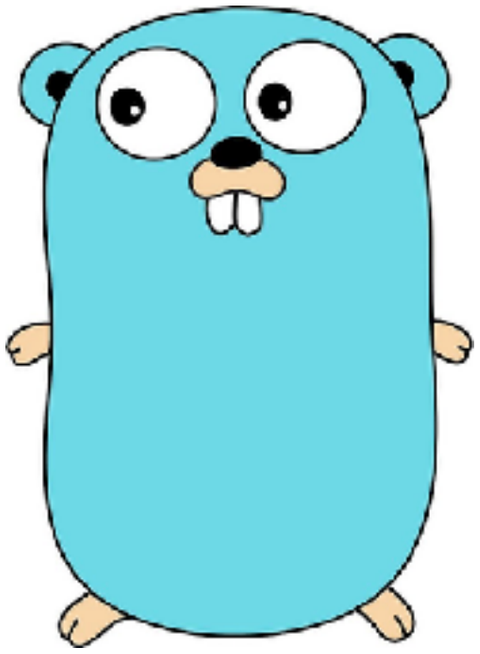
```
syntax ="proto3";

package shackbus.rotator;

message State{
    int32 azimuth = 1;
    int32 azimuth_preset = 2;
    int32 elevation = 3;
    int32 elevation_preset = 4;
}
```

```
state := sbRotator.State{
  Azimuth:         250,
  AzimuthPreset:   120,
  Elevation:       0,
  ElevationPreset: 0,
}

data, err := proto.Marshal(&state)
if err ≠ nil {
  log.Println(err)
}
```

# Micro-services to the rescue

## go-micro Architecture



| Pub/Sub | Registry | RPC |
|---|---|---|
| http | consul | grace |
| kafka | mDNS | http |
| mqtt | etc | rabbitmq |
| nats | nats | nats |
| nsq | zookeeper | tcp |
| redis | kubernetes | utp |
| ... | ... | ... |

https://micro.mu
https://github.com/micro

# Examples

# remoteRotator

https://github.com/dh1tw/remoteRotator

7x0,75mm

7x0,75mm


Client Application

Web Server

Server App

Yaesu

15m Yagi

Reg-istry

Server App

Yaesu

15m Yagi

Reg-istry

Reg-istry

List of Rotators

Proxy Obj

15m Yagi'

Proxy Obj

20m Yagi'

RPC

Pub/ Sub

RPC

Pub/ Sub

RPC

Pub/ Sub

NATS

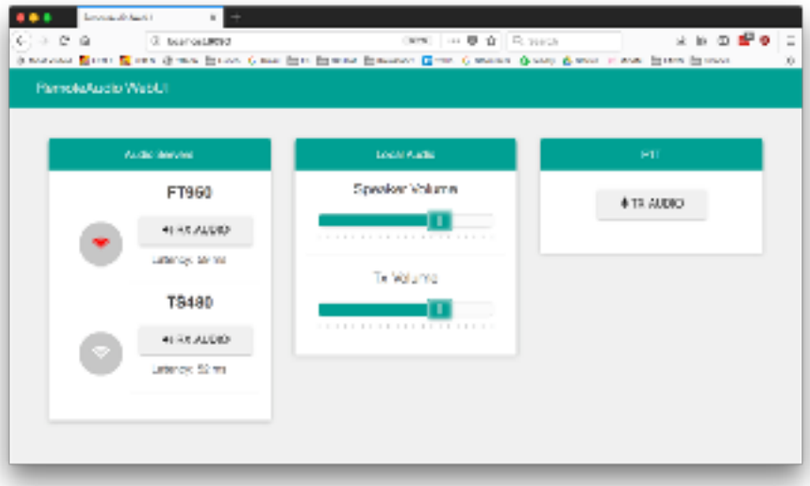
```proto
syntax ="proto3";

package shackbus;

service Rotator{
    rpc SetAzimuth(HeadingReq) returns (None);
    rpc SetElevation(HeadingReq) returns (None);
    rpc StopAzimuth(None) returns (None);
    rpc StopElevation(None) returns (None);
    rpc GetMetadata(None) returns (Metadata);
    rpc GetState(None) returns (State);
}

message None{}

message Error{
    string error = 1;
    int32 code = 2;
    string description = 3;
}

message HeadingReq{
    int32 heading = 1;
}

message HeadingResp{
    int32 heading = 1;
    int32 preset = 2;
}

message State{
    int32 azimuth = 1;
    int32 azimuth_preset = 2;
    int32 elevation = 3;
    int32 elevation_preset = 4;
}

message Metadata{
    int32 azimuth_stop = 1;
    int32 azimuth_min = 2;
    int32 azimuth_max = 3;
    int32 elevation_min = 4;
    int32 elevation_max = 5;
    bool has_azimuth = 6;
    bool has_elevation = 7;
}
```

```
import (
    natsBroker "github.com/micro/go-plugins/broker/nats"
    natsReg "github.com/micro/go-plugins/registry/nats"
    natsTr "github.com/micro/go-plugins/transport/nats"
)

func main() {

    // create instances of our nats Registry, Broker and Transport
    reg := natsReg.NewRegistry(regNatsOpts)
    br := natsBroker.NewBroker(brNatsOpts)
    tr := natsTr.NewTransport(trNatsOpts)

    rs := micro.NewService(
        micro.Name(serviceName),
        micro.RegisterInterval(time.Second*10),
        micro.Broker(br),
        micro.Transport(tr),
        micro.Registry(reg),
        micro.Server(svr),
    )

   rs.Init()

   sbRotator.RegisterRotatorHandler(rs.Server(), rpcRot)

    if err := rs.Run(); err != nil {
        os.Exit(1)
    }
}
```

```go
type rpcRotator struct {
    initialized bool
    service     micro.Service
    rotator     rotator.Rotator
    pubSubTopic string
}

func (r *rpcRotator) SetAzimuth(ctx context.Context, req *sbRotator.HeadingReq, resp *sbRotator.None) erro
{
    if r.rotator.HasAzimuth() {
        err := r.rotator.SetAzimuth(int(req.Heading))
        return err
    }
    return fmt.Errorf("rotator does not support azimuth")
}

func (r *rpcRotator) GetState(ctx context.Context, req *sbRotator.None, resp *sbRotator.State) error {
    info := r.rotator.Info()
    resp.Azimuth = int32(info.Azimuth)
    resp.AzimuthPreset = int32(info.AzPreset)
    resp.Elevation = int32(info.Elevation)
    resp.ElevationPreset = int32(info.ElPreset)
    return nil
}
```

# remoteAudio

https://github.com/dh1tw/remoteAudio




Server App

FT950 Audio Server

Reg-istry

RPC

Pub/ Sub

Server App

TS480 Audio Server

Reg-istry

RPC

Pub/ Sub

Client Application

Web Server + REST API

Reg-istry

List of Audio Server

Proxy Obj FT950’

Proxy Obj TS480’

RPC

Pub/ Sub

NATS

# gorigctl*

https://github.com/dh1tw/gorigctl

Server App

FT950 CAT

Reg-istry

Pub/ Sub

Client Application

CLI or CLI GUI

Reg-istry

Proxy Obj FT950’

Pub/ Sub

MQTT

* pending migration to go-micro -> separation between pub/sub and RPC

# Summary

**Shackbus is an initiative to define interoperable Services / Devices typically found in our Amateur Radio shacks.**

**To operate micro services, we need:**
- Registry
- Remote Procedure Calls (RPC)
- Publish & Subscribe
- A common data model

**Micro services should:**
- do one thing really good
- be loosely coupled
- be designed for failure

## Checkout the example implementations and get involved!

# Thats it!