OATSCON23

Rethinking Connectivity in Agriculture

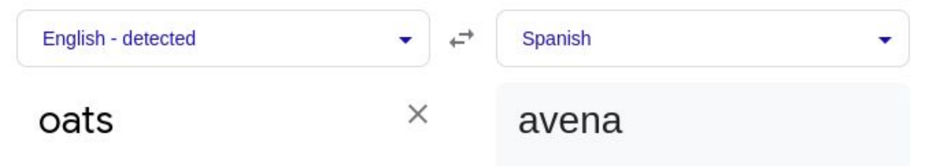
an Avena use case

Andrew D. Balmos, Fabio A. Castiblanco,
James V. Krogmeier, Dennis R. Buckmaster,
David J. Love

February 23rd, 2023

OATS IoT4Ag PURDUE UNIVERSITY

What is Avena?



English - detected ⇄ Spanish

oats ×

avena

Avena is an *open source*

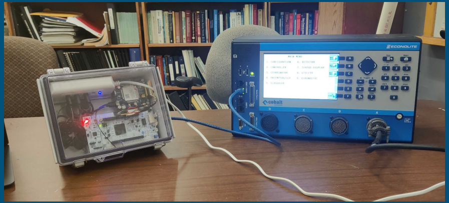
software and *communication* stack

It's not intended as a commercial product
(but could be used in one)

Our goal: architecture research and disruption

We eat our own dog food:

- ISOBlue
- Purdue OATS DataStation (POD)
- Data Diode (connectivity)
- *… future edge computing research …*



Avena design goal

Create opportunity.

Borrowing ideas: Be a matchmaker

Android is an **open source software stack** created for a wide array of devices with different form factors. Android's primary purpose is to create an **open software platform available for carriers, OEMs, and developers to make their innovative ideas a reality** and to introduce a successful, real-world product that improves the mobile experience for users.

Android is designed so that there's **no central point of failure, where one industry player restricts or controls the innovations of another.** The result is a full, production-quality consumer product with source code open for customization and porting.

https://source.android.com/docs/setup/about

Borrowing ideas: Be the matchmaker

Android abstracts *hardware vendors* from *software vendors* via a standard API.

Android's pre-competitive interface enables a much larger market than one of the vendors could create alone.

Consider: *What happened to Windows phones? Blackberry? Tizen?*

Why not just use Android?
Android focus on devices. Avena focuses on a full system of things (including Android things).

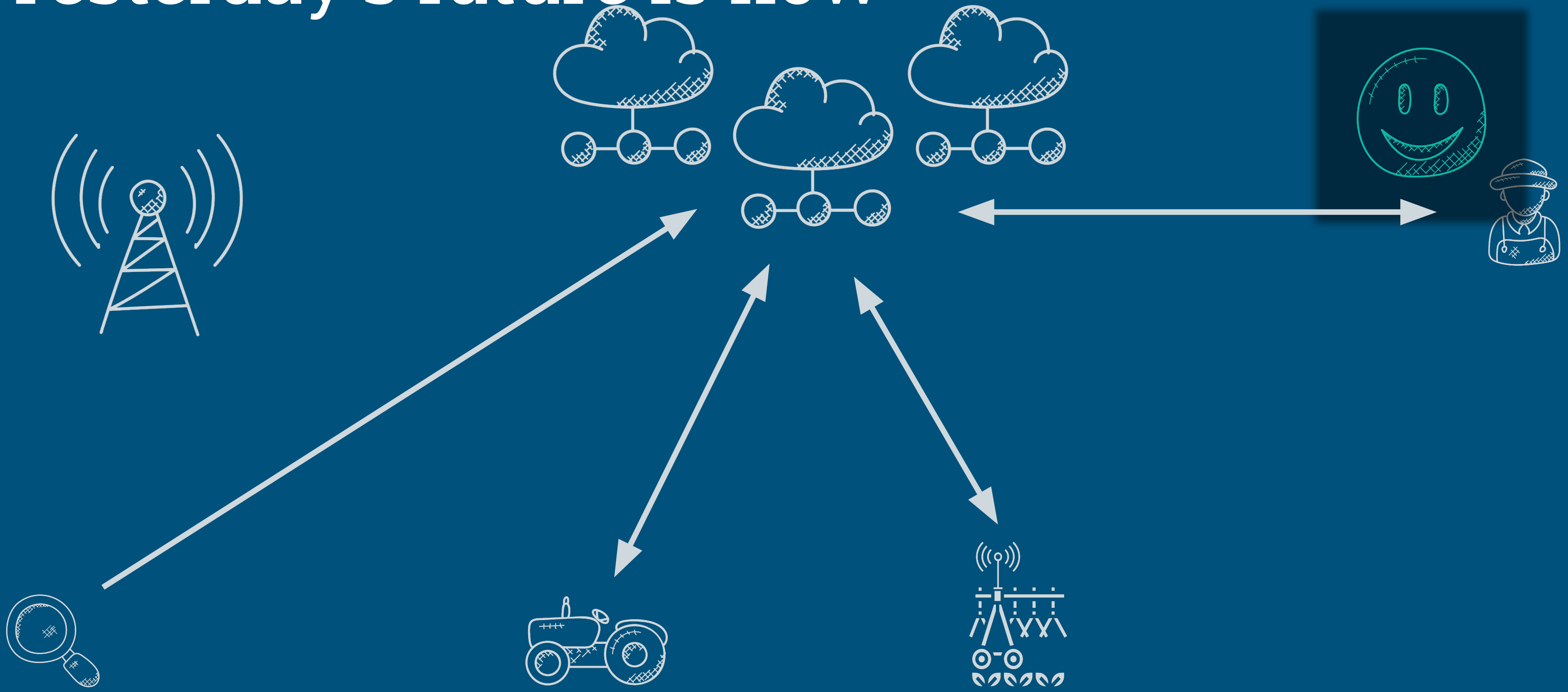
Already partly there?
Avena on a Deere MTG?

READY

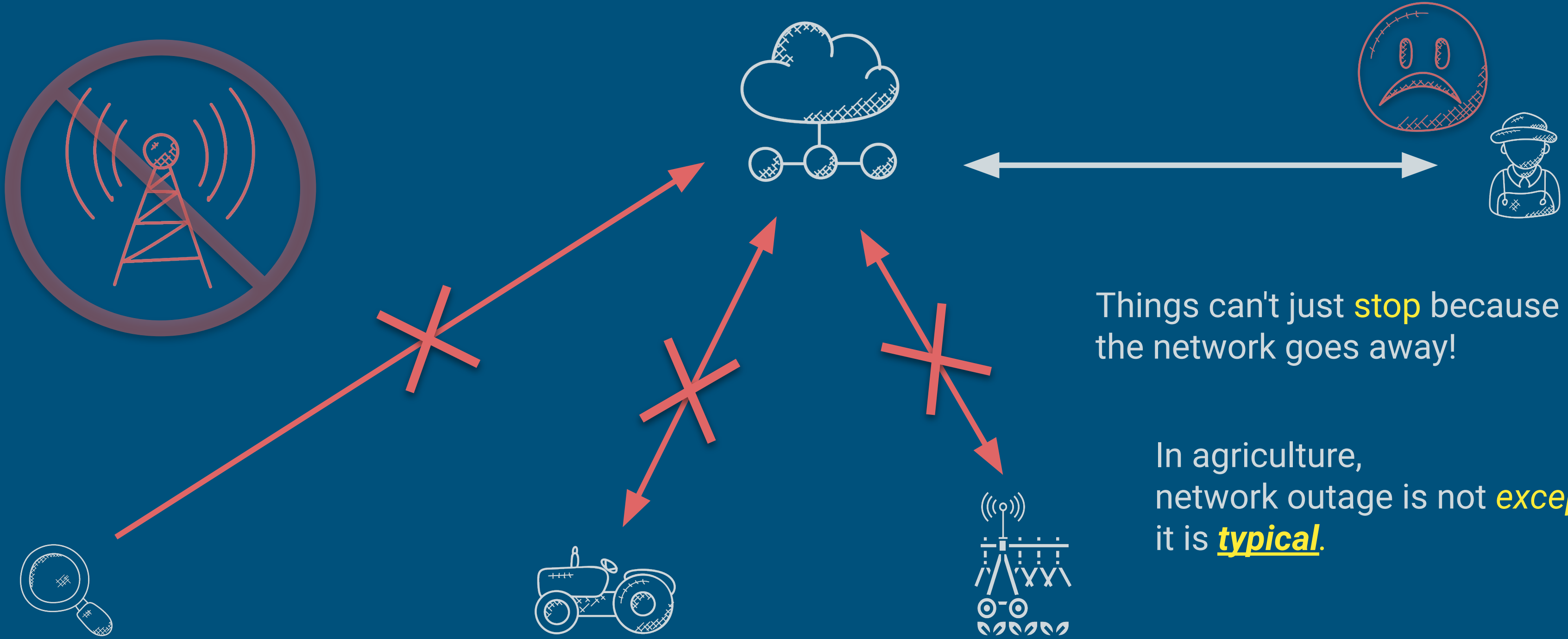
Avena design goal

Build software local first.

Yesterday's future is now



...except when it isn't



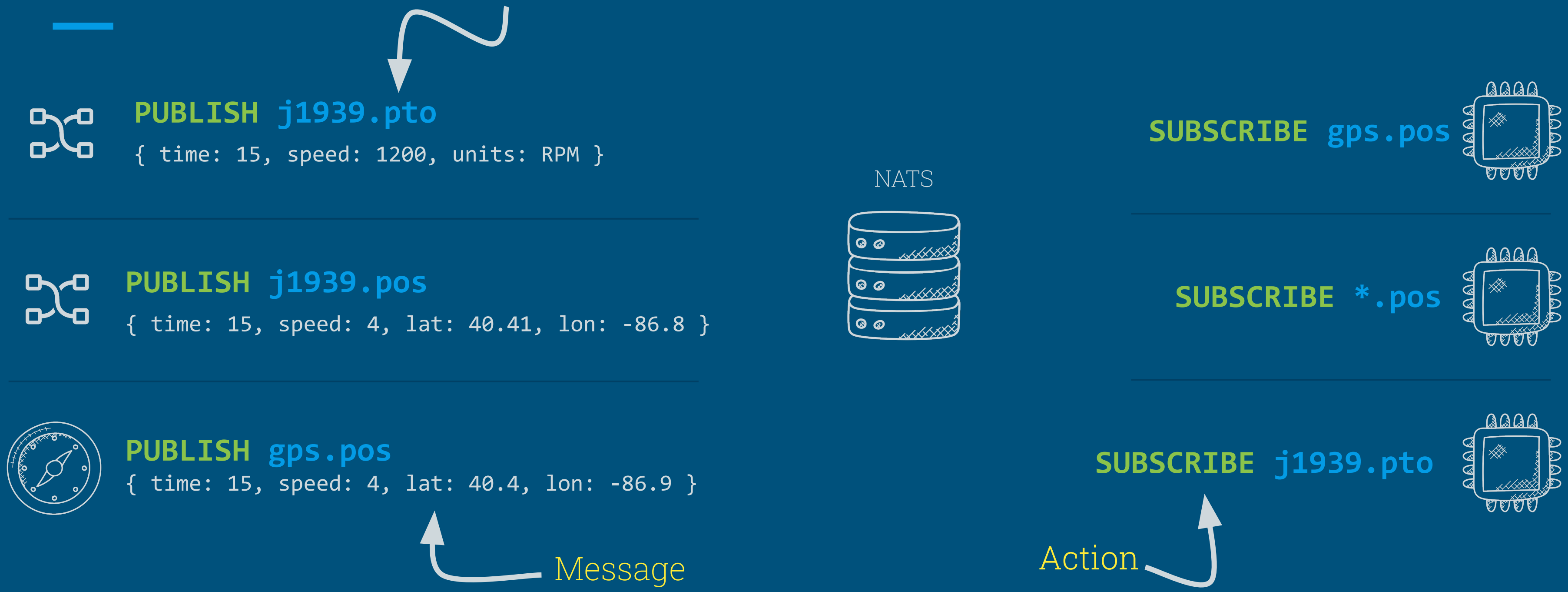
Things can't just stop because the network goes away!

In agriculture,
network outage is not *exceptional*,
it is ***typical***.

Aside: NATS

NATS is an open source project that passes messages between a set of publishers and subscribers

Publishers and subscribers don't know about each other

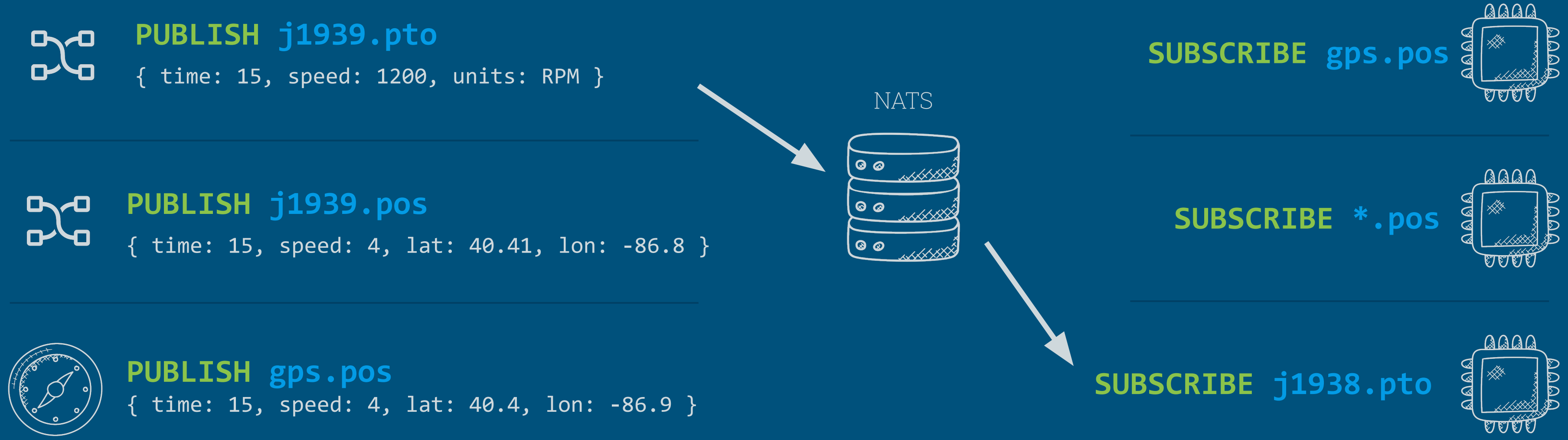


```
PUBLISH j1939.pto
{ time: 15, speed: 1200, units: RPM }

PUBLISH j1939.pos
{ time: 15, speed: 4, lat: 40.41, lon: -86.8 }

PUBLISH gps.pos
{ time: 15, speed: 4, lat: 40.4, lon: -86.9 }
```

```
SUBSCRIBE gps.pos

SUBSCRIBE *.pos

SUBSCRIBE j1939.pto
```

Aside: NATS

Publish and subscribe



PUBLISH j1939.pto

`{ time: 15, speed: 1200, units: RPM }`

PUBLISH j1939.pos

`{ time: 15, speed: 4, lat: 40.41, lon: -86.8 }`

PUBLISH gps.pos

`{ time: 15, speed: 4, lat: 40.4, lon: -86.9 }`

NATS

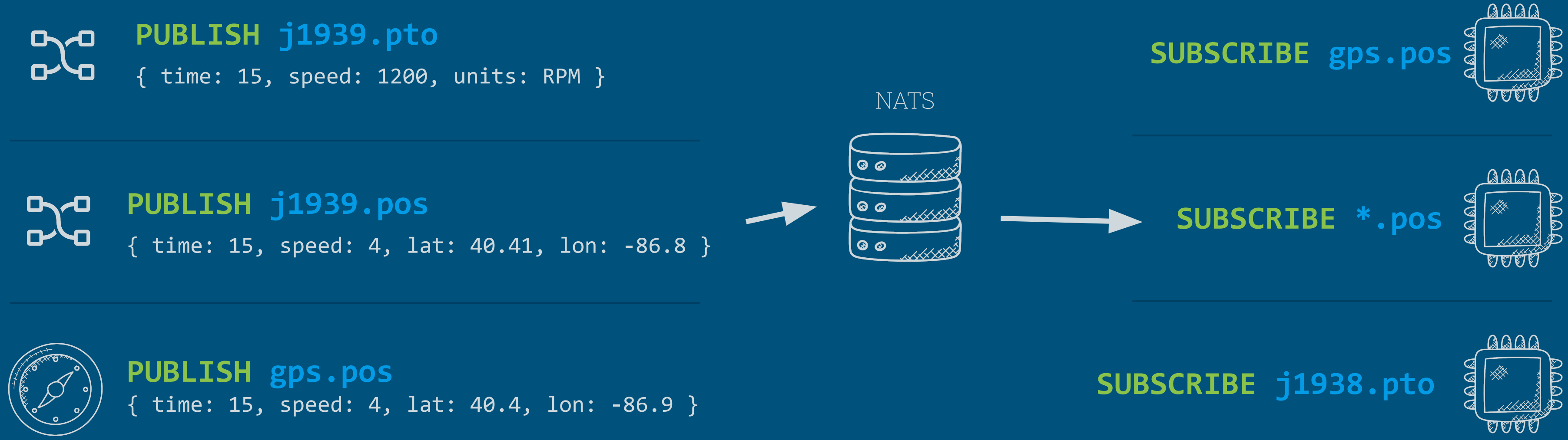
SUBSCRIBE gps.pos

SUBSCRIBE *.pos

SUBSCRIBE j1938.pto

Aside: NATS

Publish and subscribe



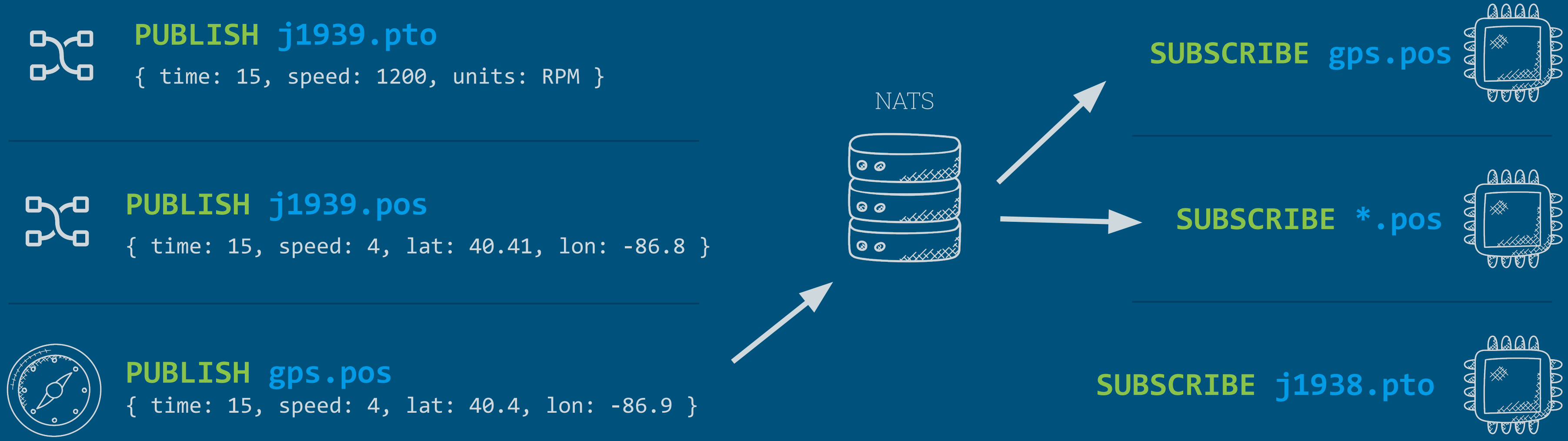
```
PUBLISH j1939.pto
{ time: 15, speed: 1200, units: RPM }

PUBLISH j1939.pos
{ time: 15, speed: 4, lat: 40.41, lon: -86.8 }

PUBLISH gps.pos
{ time: 15, speed: 4, lat: 40.4, lon: -86.9 }
```

NATS

```
SUBSCRIBE gps.pos

SUBSCRIBE *.pos

SUBSCRIBE j1938.pto
```

Aside: NATS

Publish and subscribe



PUBLISH j1939.pto

`{ time: 15, speed: 1200, units: RPM }`

PUBLISH j1939.pos

`{ time: 15, speed: 4, lat: 40.41, lon: -86.8 }`

PUBLISH gps.pos

`{ time: 15, speed: 4, lat: 40.4, lon: -86.9 }`

SUBSCRIBE gps.pos

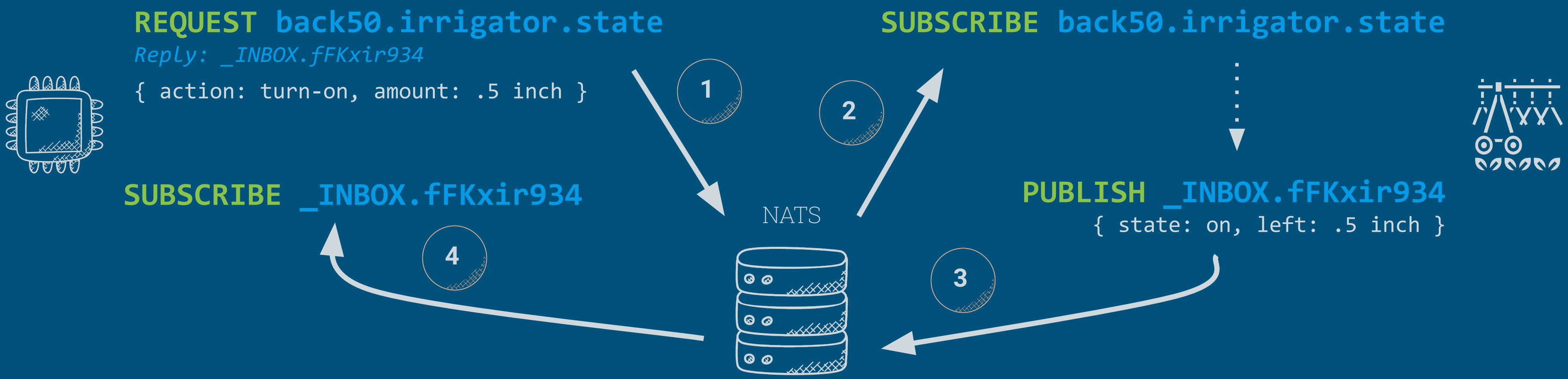
SUBSCRIBE *.pos

SUBSCRIBE j1938.pto

Aside: NATS

Request + Reply

Also known as a Remote Procedure Call (RPC)
or
Command and Control



REQUEST back50.irrigator.state
Reply: _INBOX.fFKxir934
{ action: turn-on, amount: .5 inch }

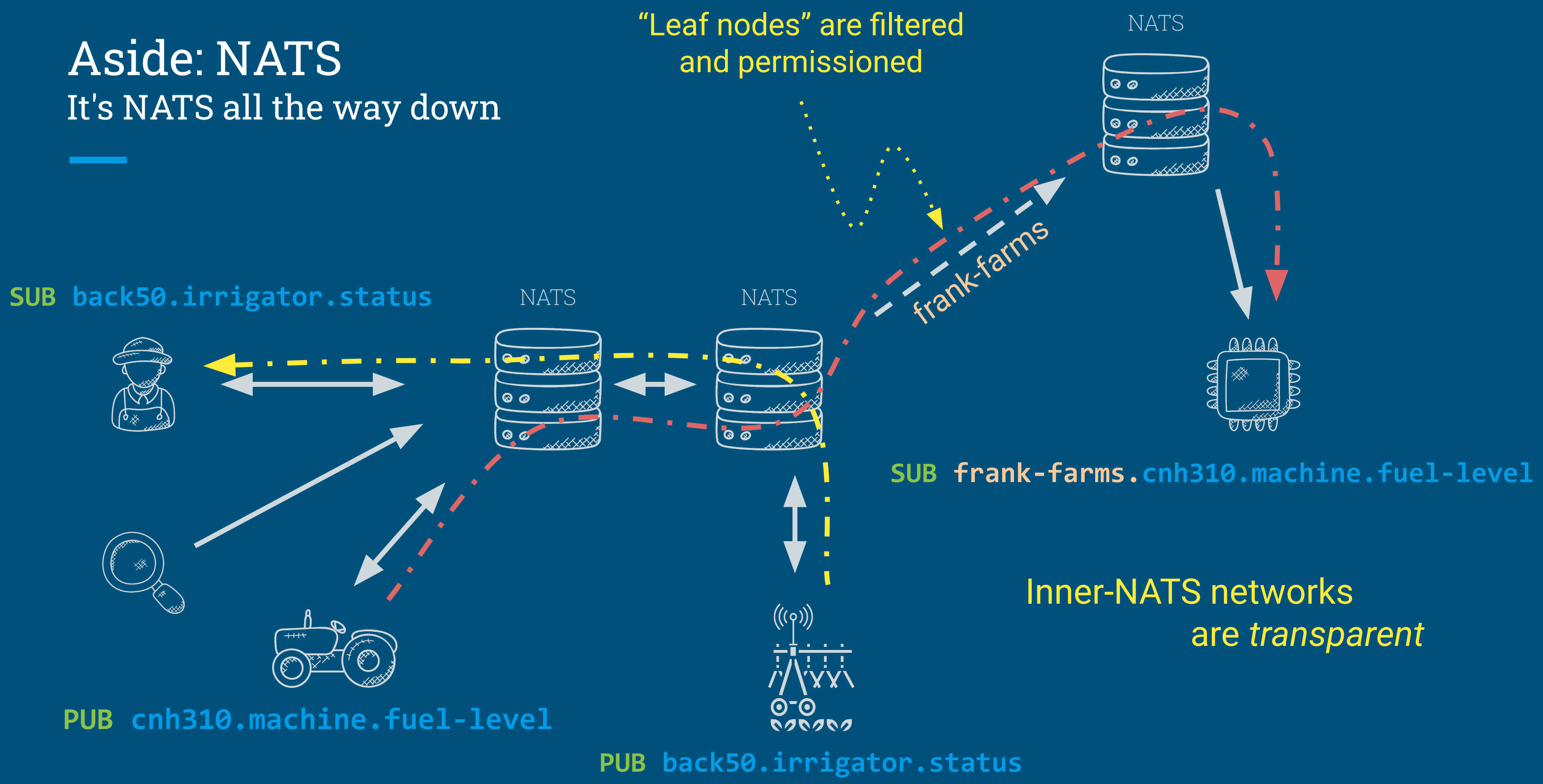
SUBSCRIBE _INBOX.fFKxir934

SUBSCRIBE back50.irrigator.state

PUBLISH _INBOX.fFKxir934
{ state: on, left: .5 inch }

Aside: NATS

It's NATS all the way down



"Leaf nodes" are filtered and permissioned
NATS
SUB back50.irrigator.status
NATS
NATS
frank-farms
SUB frank-farms.cnh310.machine.fuel-level
PUB cnh310.machine.fuel-level
PUB back50.irrigator.status

Inner-NATS networks are *transparent*

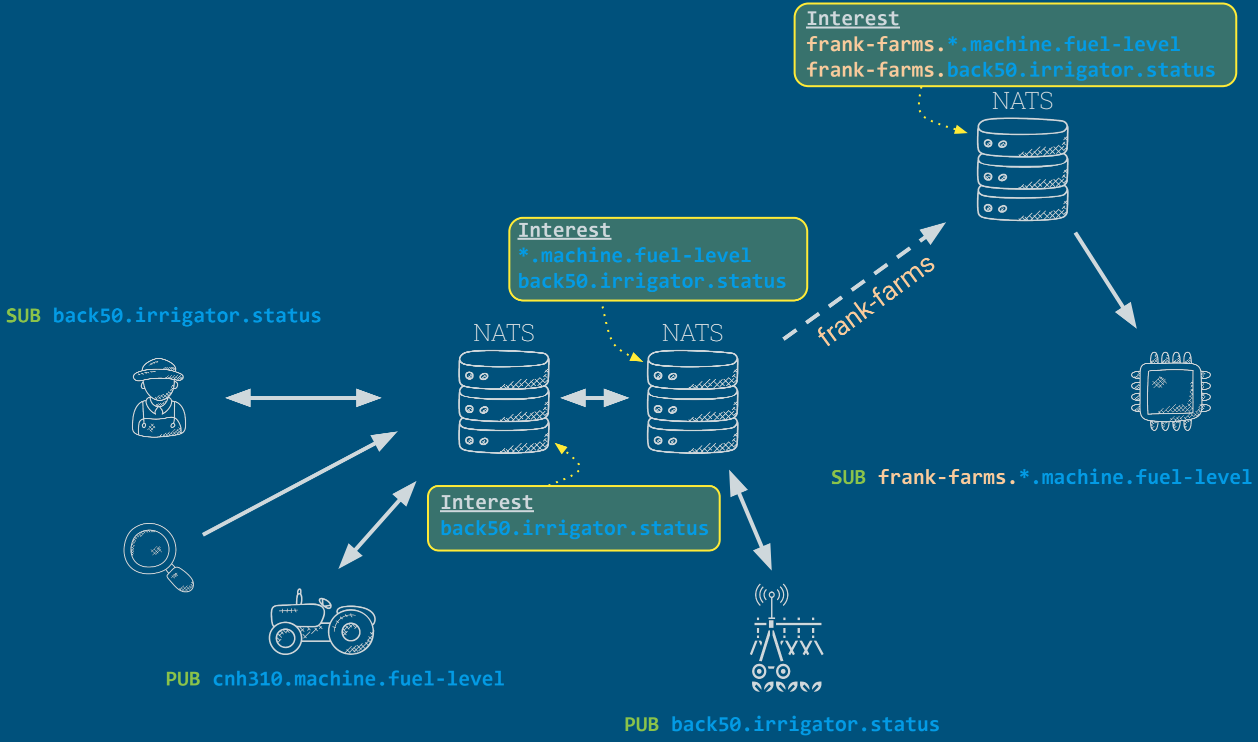
Aside: NATS

Interest graphs

Services receive data by “demanding” it through a subscription pattern.

The cumulative interest in subjects is communicated to peer NATS servers.

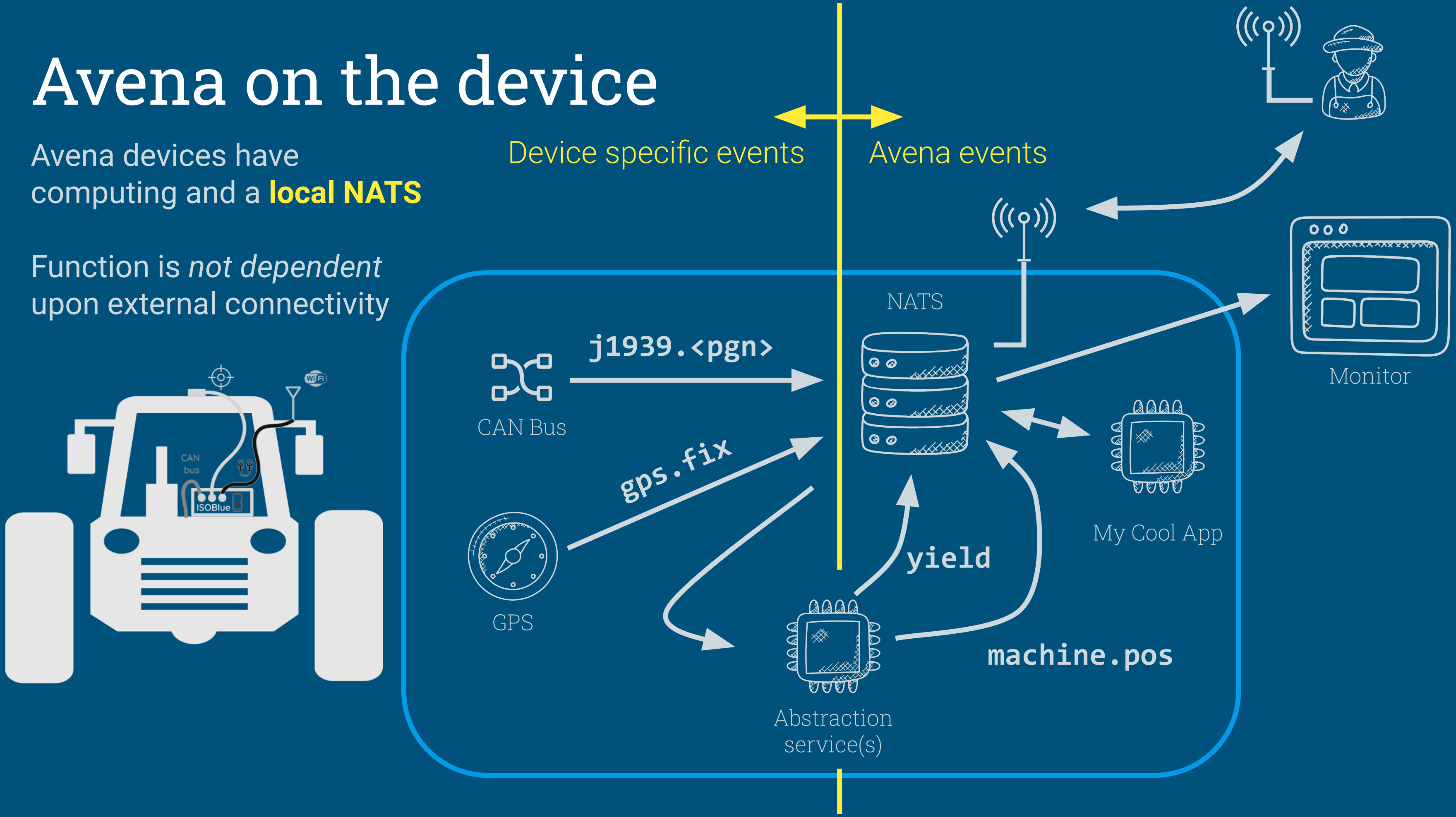
NATS *only* sends messages to peers that have interest in the data


Interest
frank-farms.*.machine.fuel-level
frank-farms.back50.irrigator.status
NATS
Interest
*.machine.fuel-level
back50.irrigator.status
frank-farms
SUB back50.irrigator.status
NATS
NATS
SUB frank-farms.*.machine.fuel-level
Interest
back50.irrigator.status
PUB cnh310.machine.fuel-level
PUB back50.irrigator.status

Avena on the device

Avena devices have computing and a **local NATS**

Function is *not dependent* upon external connectivity

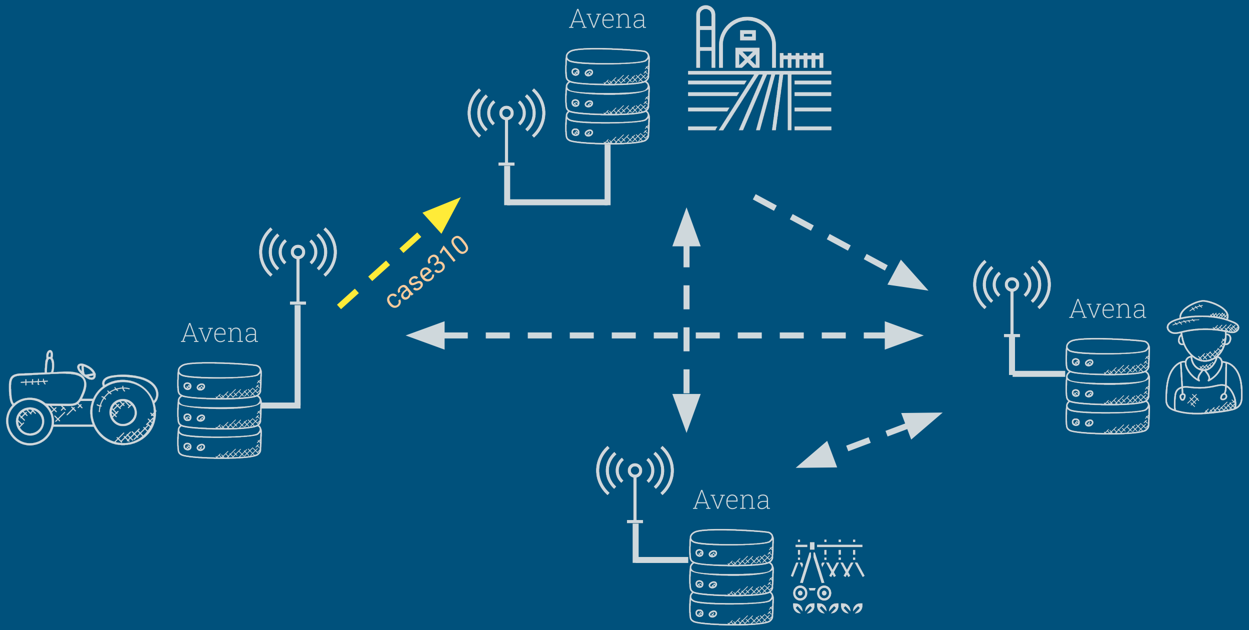


READY

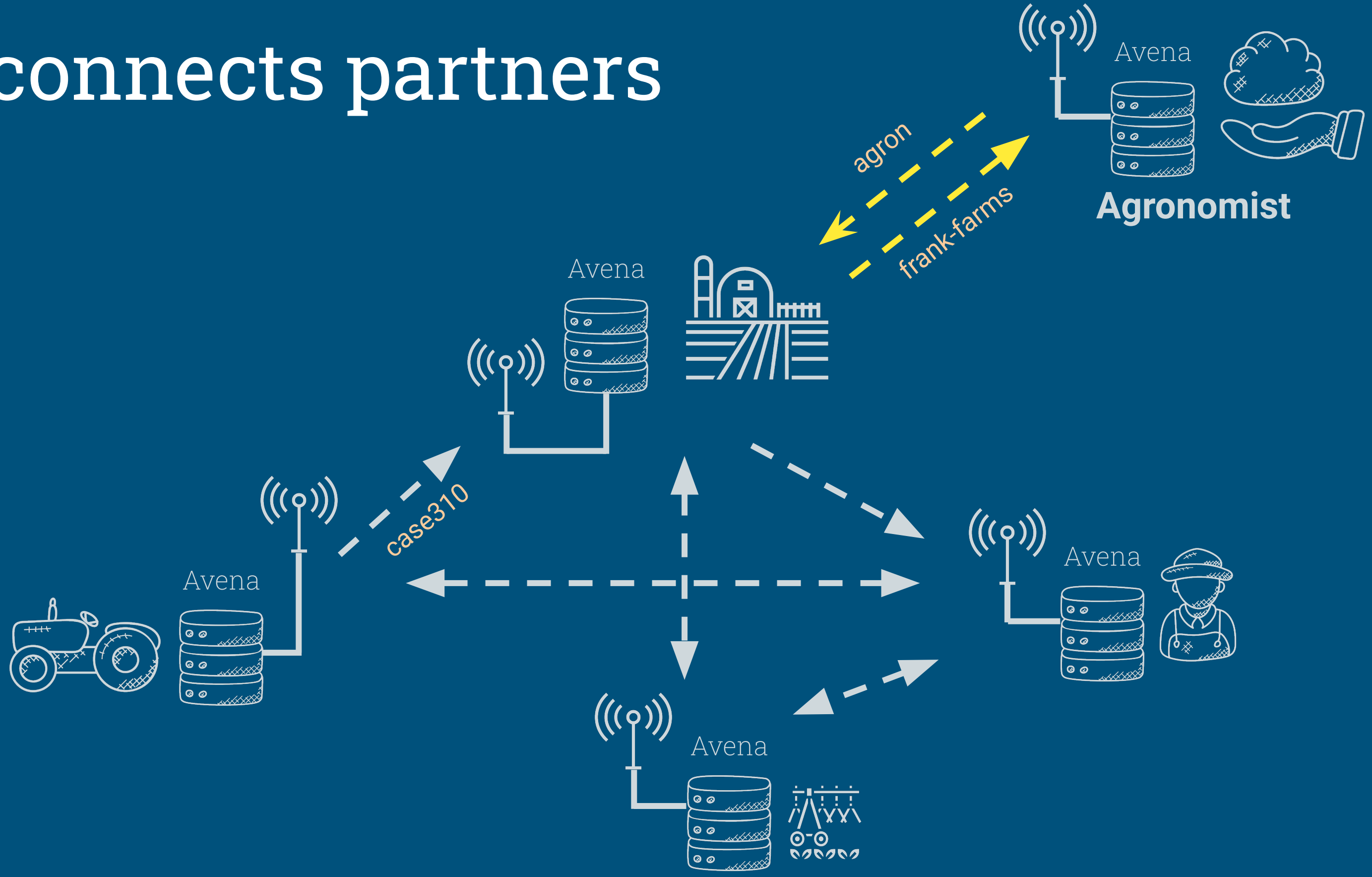
Avena design goal

Connect the things.

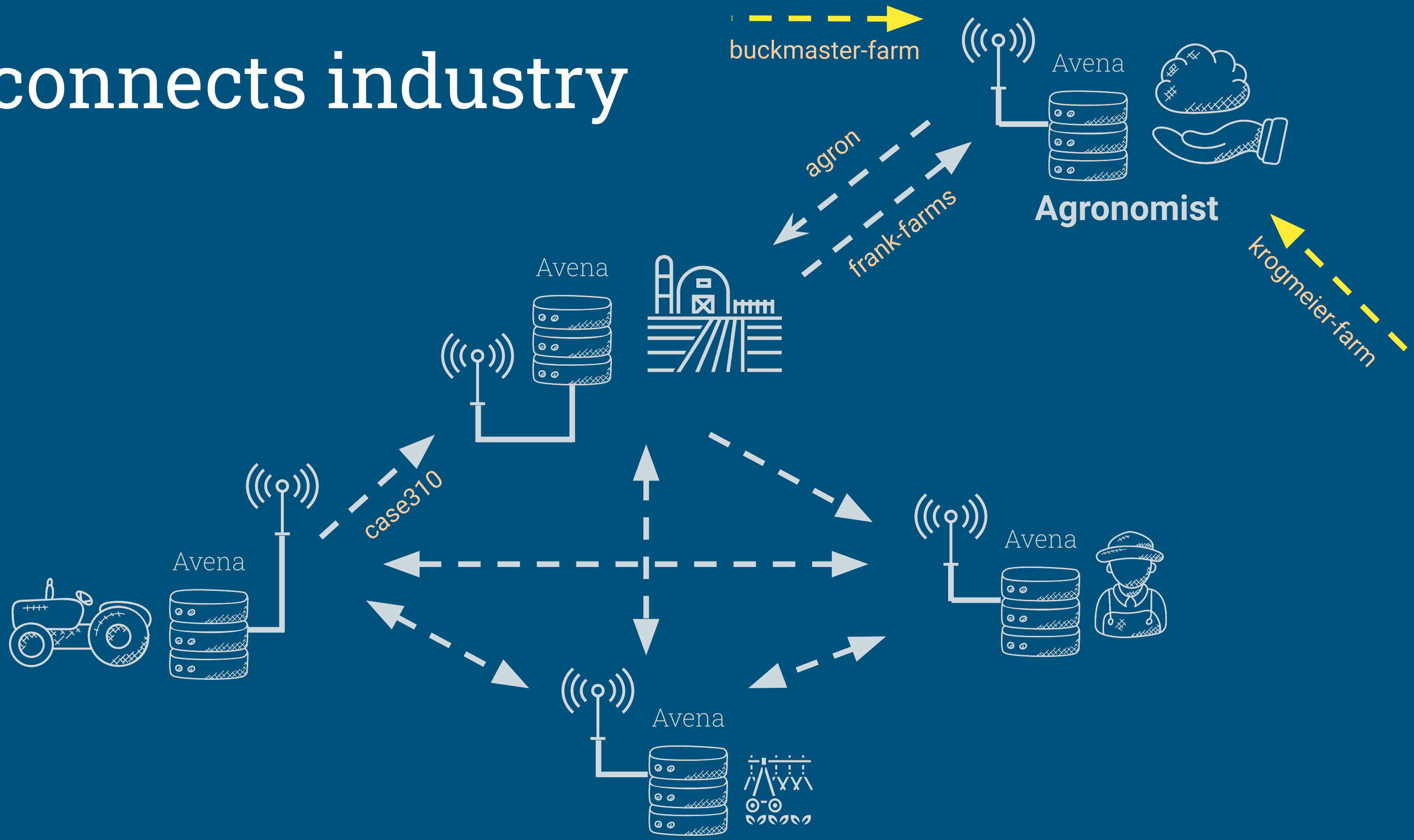
Avena connects the farm



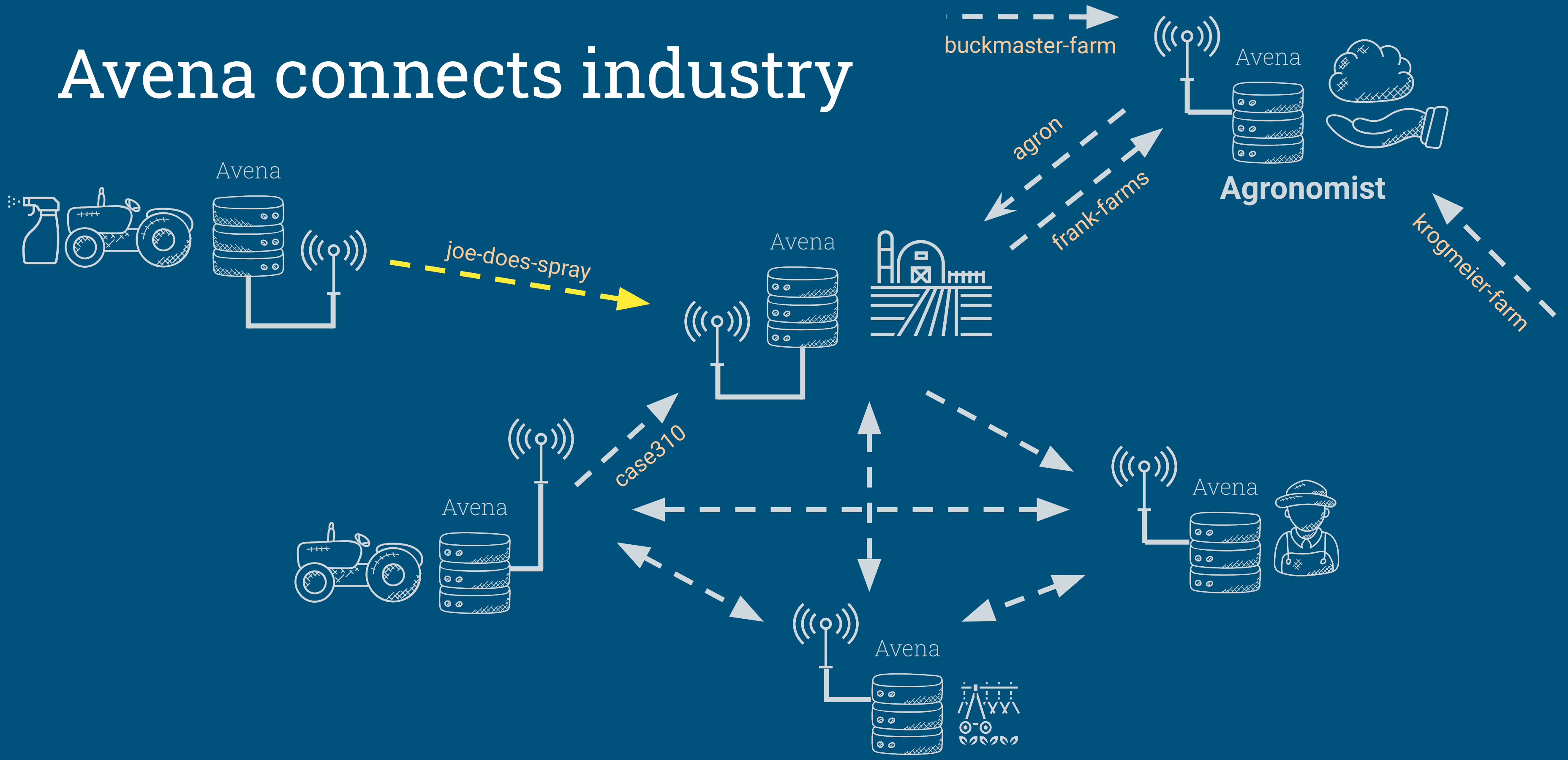
Avena connects partners



Avena connects industry



Avena connects industry



RESEARCH

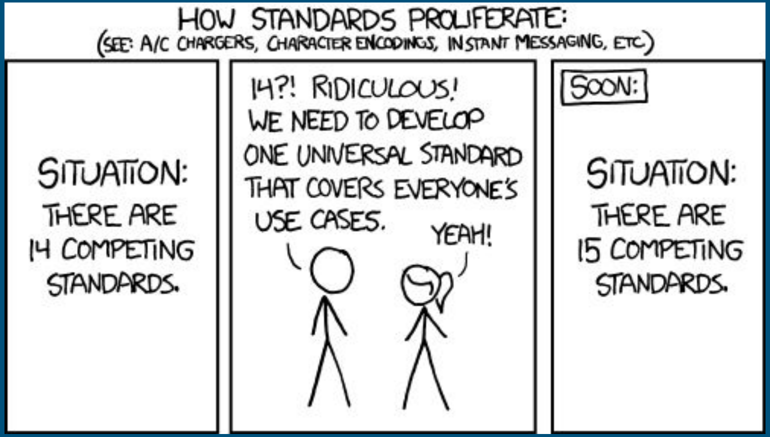
Avena design goal

Interoperability is message passing.

(and also how one solves distributed system)

Messages are the data.

Based on leading distributed system design patterns, we lean on sharing messages, not *data files*.


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Data files should be created by the *consumer,* however best *fits their needs.*

It is more than NATS

NATS provides many positive benefits and is an excellent base, but

Avena events…

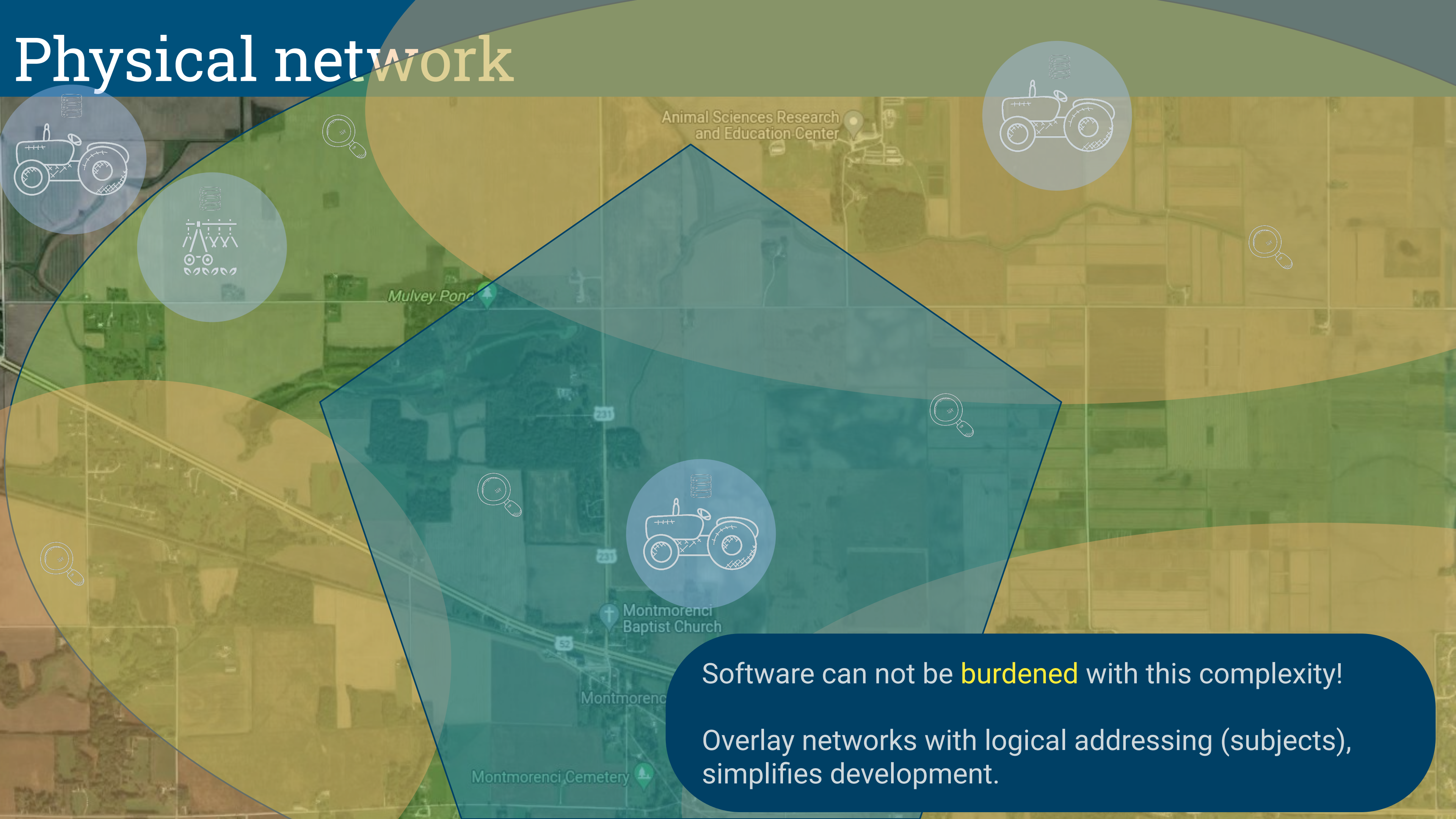
- Need standardized schemas
- Need standardized subjects
- Need distributed tolerant timestamps (global order)
- Must be secure and allow for (distributed + disconnected) permissions
- Opportunistically move messages even when connections are unstable
- Services must be discoverable
- Etc.

RESEARCH

Avena design goal

Software doesn't know about the physical network

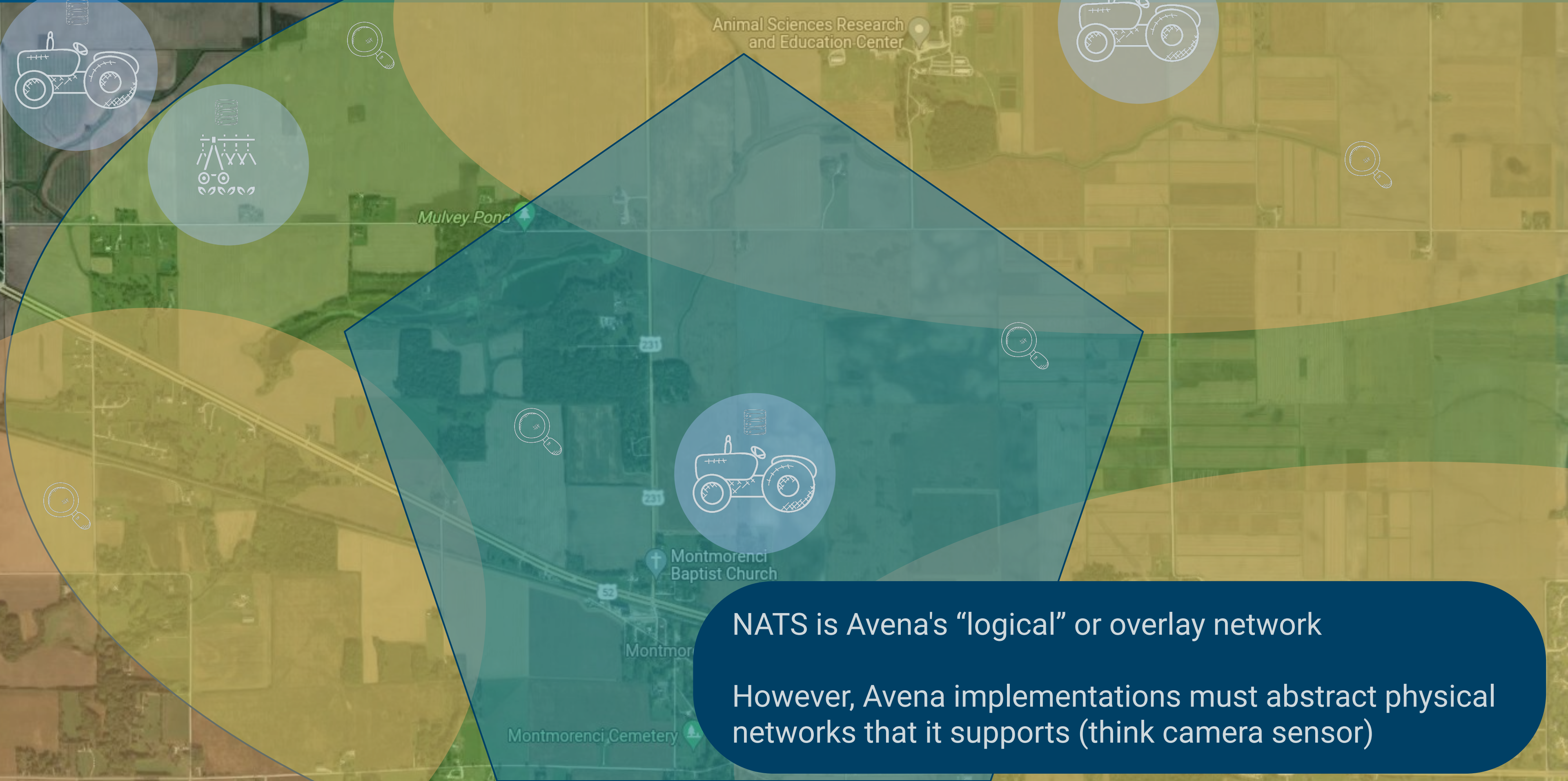
Physical network


Animal Sciences Research and Education Center
Mulvey Pond
Montmorenci Baptist Church
Montmorenci
Montmorenci Cemetery
231
52

Software can not be burdened with this complexity!

Overlay networks with logical addressing (subjects), simplifies development.

Physical network



Animal Sciences Research and Education Center
Mulvey Pond
231
231
52
Montmorenci Baptist Church
Montmor
Montmorenci Cemetery

NATS is Avena's “logical” or overlay network

However, Avena implementations must abstract physical networks that it supports (think camera sensor)

Physical network

Animal Sciences Research
and Education Center
Mulvey Pond
231
231
52
Montmorenci
Baptist Church
Montmorenci
Montmorenci Cemetery

Research Question
How best to route messages?

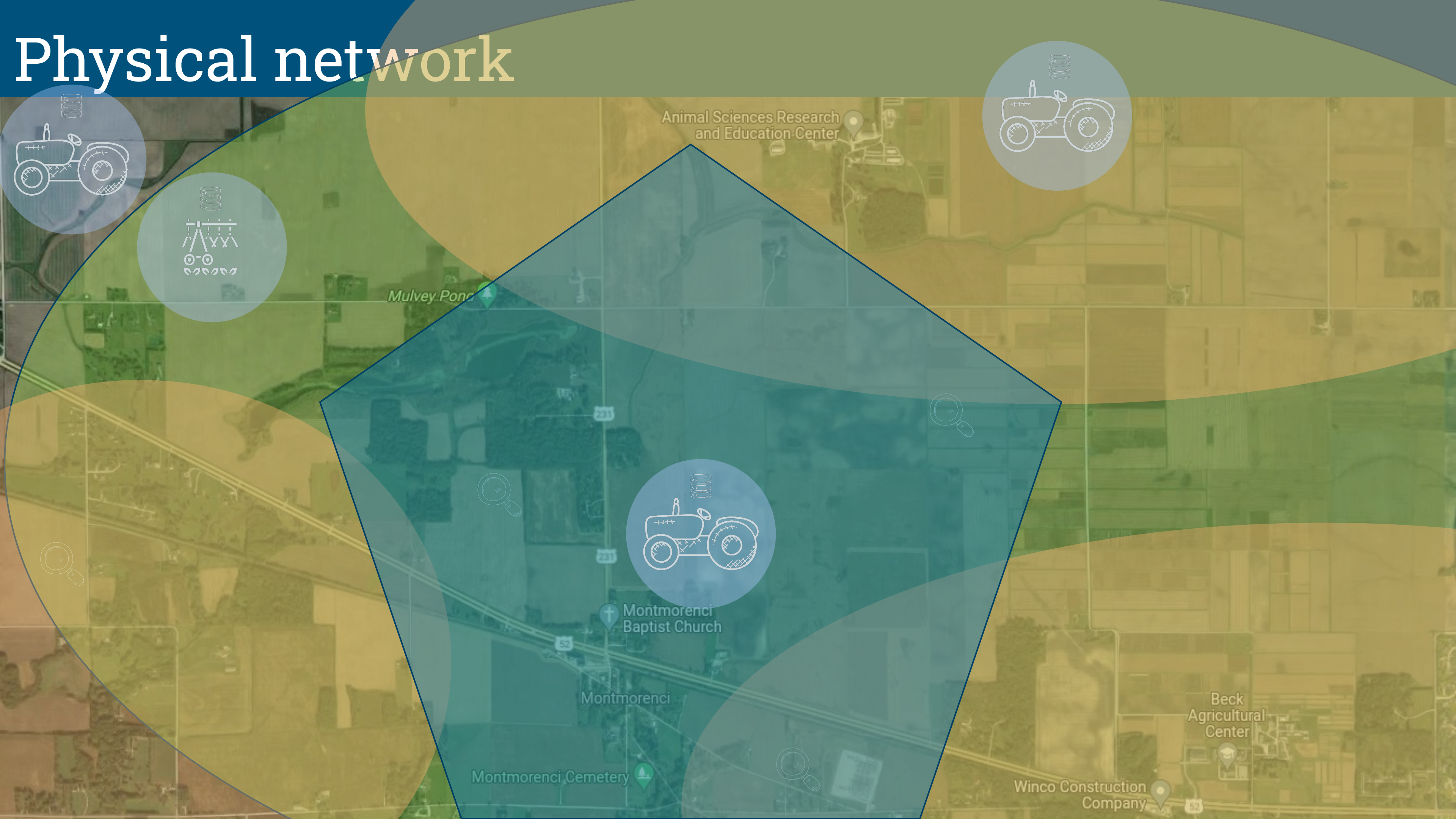
RESEARCH

Avena design goal (we think?)

Ag networks should be opportunistic.

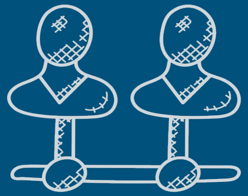
"Delay tolerant"

Physical network

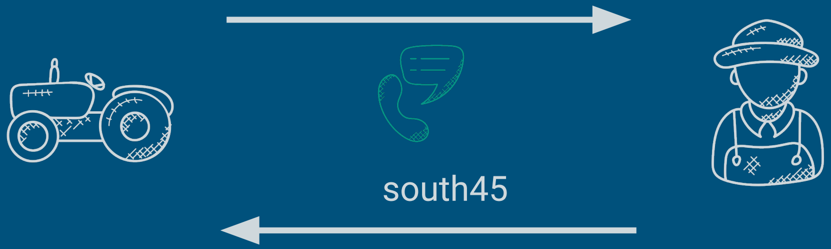


Animal Sciences Research and Education Center
Mulvey Pond
231
231
Montmorenci Baptist Church
52
Montmorenci
Montmorenci Cemetery
Beck Agricultural Center
Winco Construction Company
52

We're already doing it!



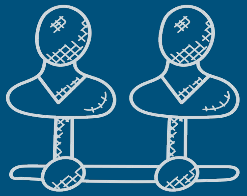
HumanNet



Almost done, what's next?

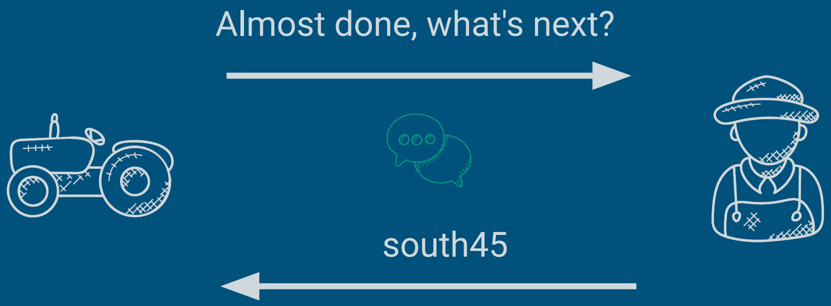
south45

We're already doing it!



HumanNet

Scenario: What if voice calls won't go through?

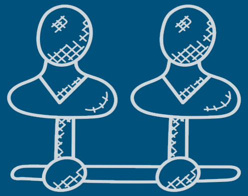


Almost done, what's next?

south45

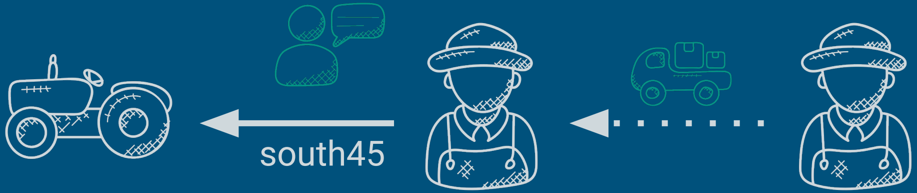
Opportunistically select text message network

We're already doing it!



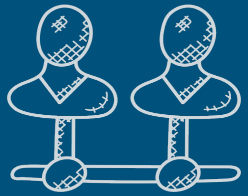
HumanNet

Scenario: What if no cell at all?



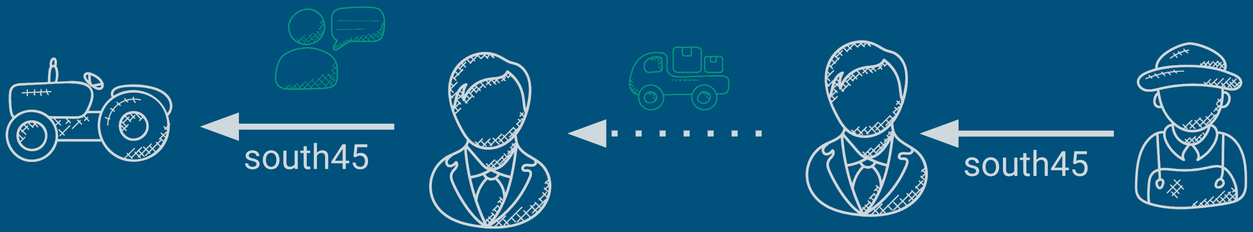
Physically move message to the field, and use the "local voice" network

We're already doing it!



HumanNet

Scenario: What if voice calls won't go through?
(alternative)



Relay message to seed tender, seed tender physically moves message to the field, and uses the "local voice" network.

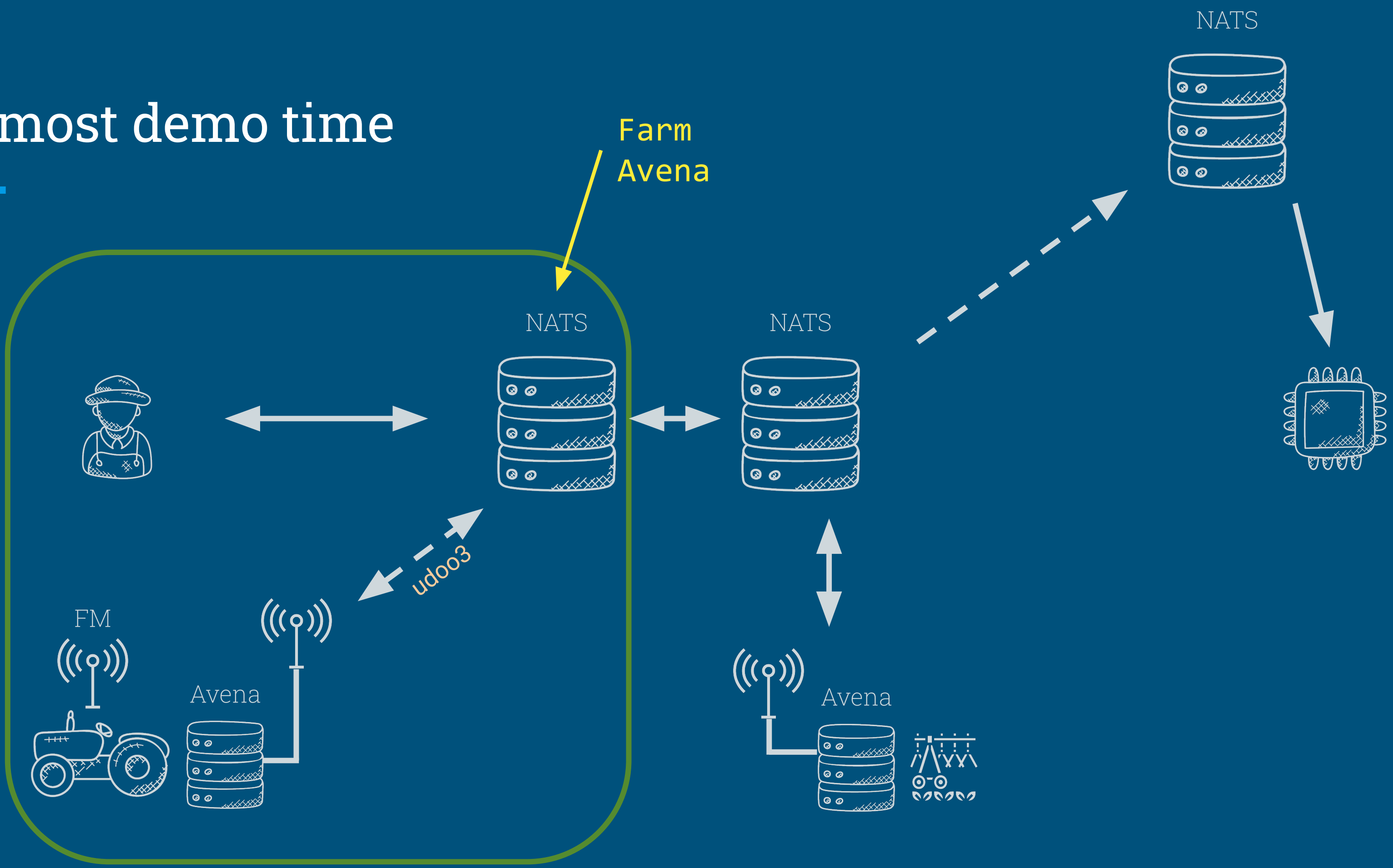
More effective routing algorithm.

Ag is a distributed system

Always has been.

Always will be (probably).

Almost demo time



DEMO: ISOBlue and Avena in action

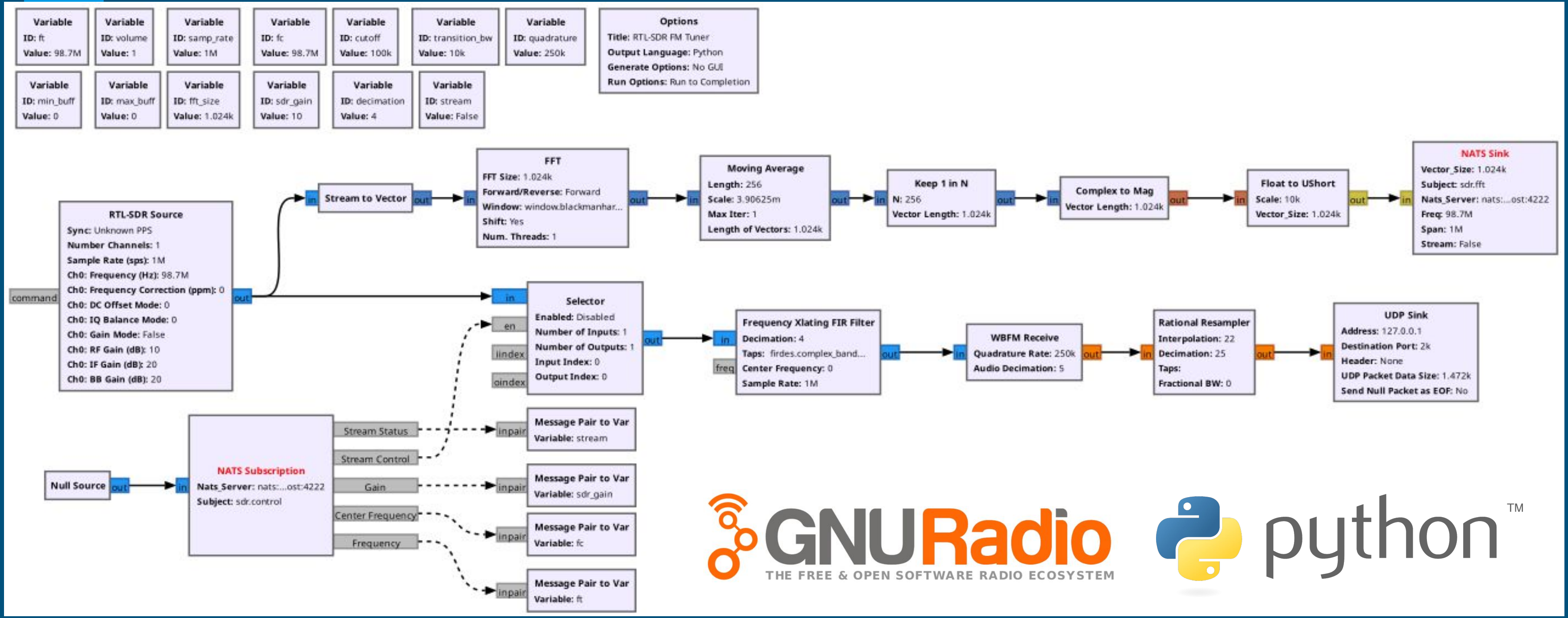
Scan me!

Go to:
avena.oatscenter.org

Open Source:
github.com/oats-center/avena-app

Password:
iot4ag

Only need to add your code... or blocks!



Variable
ID: ft
Value: 98.7M

Variable
ID: volume
Value: 1

Variable
ID: samp_rate
Value: 1M

Variable
ID: fc
Value: 98.7M

Variable
ID: cutoff
Value: 100k

Variable
ID: transition_bw
Value: 10k

Variable
ID: quadrature
Value: 250k

Options
Title: RTL-SDR FM Tuner
Output Language: Python
Generate Options: No GUI
Run Options: Run to Completion

Variable
ID: min_buff
Value: 0

Variable
ID: max_buff
Value: 0

Variable
ID: fft_size
Value: 1.024k

Variable
ID: sdr_gain
Value: 10

Variable
ID: decimation
Value: 4

Variable
ID: stream
Value: False

RTL-SDR Source
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 1M
Ch0: Frequency (Hz): 98.7M
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Stream to Vector

FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Moving Average
Length: 256
Scale: 3.90625m
Max Iter: 1
Length of Vectors: 1.024k

Keep 1 in N
N: 256
Vector Length: 1.024k

Complex to Mag
Vector Length: 1.024k

Float to UShort
Scale: 10k
Vector_Size: 1.024k

NATS Sink
Vector_Size: 1.024k
Subject: sdr.fft
Nats_Server: nats:...ost:4222
Freq: 98.7M
Span: 1M
Stream: False

Selector
Enabled: Disabled
Number of Inputs: 1
Number of Outputs: 1
Input Index: 0
Output Index: 0

Frequency Xlating FIR Filter
Decimation: 4
Taps: firdes.complex_band...
Center Frequency: 0
Sample Rate: 1M

WBFM Receive
Quadrature Rate: 250k
Audio Decimation: 5

Rational Resampler
Interpolation: 22
Decimation: 25
Taps:
Fractional BW: 0

UDP Sink
Address: 127.0.0.1
Destination Port: 2k
Header: None
UDP Packet Data Size: 1.472k
Send Null Packet as EOF: No

Null Source

NATS Subscription
Nats_Server: nats:...ost:4222
Subject: sdr.control

Message Pair to Var
Variable: stream

Message Pair to Var
Variable: sdr_gain

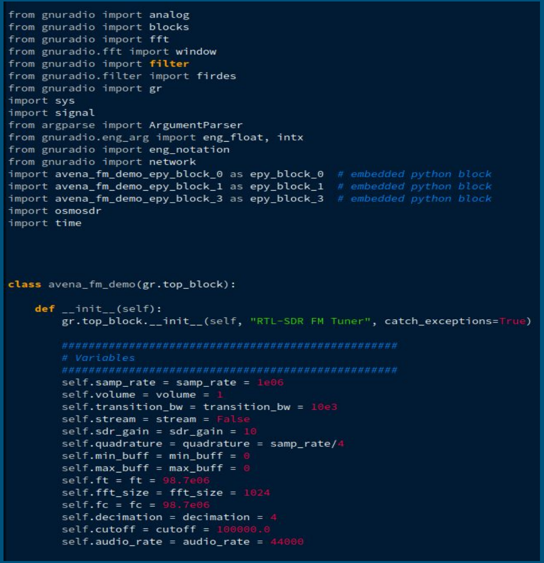
Message Pair to Var
Variable: fc

Message Pair to Var
Variable: ft

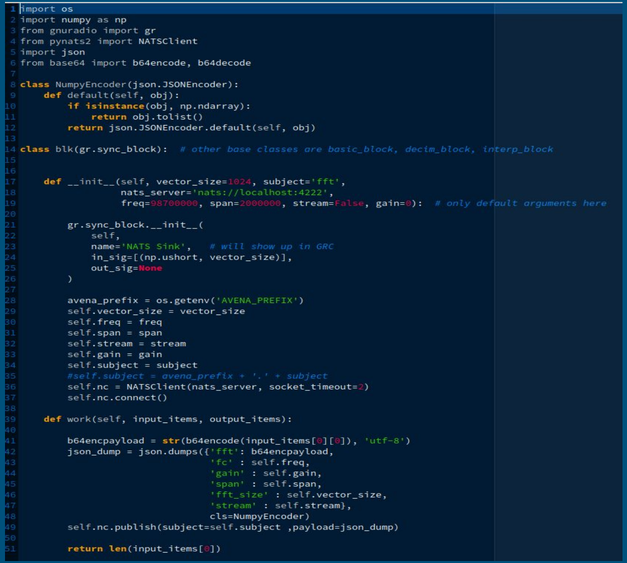
GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

python™

Only need to add your code… or blocks!



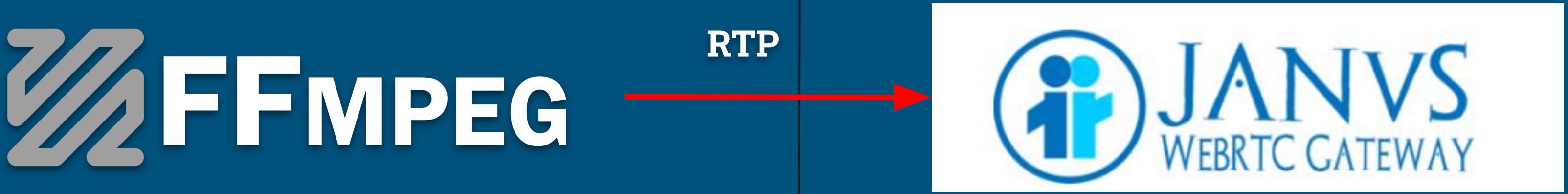
```python
from gnuradio import analog
from gnuradio import blocks
from gnuradio import fft
from gnuradio.fft import window
from gnuradio import filter
from gnuradio.filter import firdes
from gnuradio import gr
import sys
import signal
from argparse import ArgumentParser
from gnuradio.eng_arg import eng_float, intx
from gnuradio import eng_notation
from gnuradio import network
import avena_fm_demo_epy_block_0 as epy_block_0  # embedded python block
import avena_fm_demo_epy_block_1 as epy_block_1  # embedded python block
import avena_fm_demo_epy_block_3 as epy_block_3  # embedded python block
import osmosdr
import time



class avena_fm_demo(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "RTL-SDR FM Tuner", catch_exceptions=True)

        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 1e06
        self.volume = volume = 1
        self.transition_bw = transition_bw = 10e3
        self.stream = stream = False
        self.sdr_gain = sdr_gain = 10
        self.quadrature = quadrature = samp_rate/4
        self.min_buff = min_buff = 0
        self.max_buff = max_buff = 0
        self.ft = ft = 98.7e06
        self.fft_size = fft_size = 1024
        self.fc = fc = 98.7e06
        self.decimation = decimation = 4
        self.cutoff = cutoff = 100000.0
        self.audio_rate = audio_rate = 44000
```



```python
import os
import numpy as np
from gnuradio import gr
from pynats2 import NATSClient
import json
from base64 import b64encode, b64decode

class NumpyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.ndarray):
            return obj.tolist()
        return json.JSONEncoder.default(self, obj)

class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block


    def __init__(self, vector_size=1024, subject='fft',
                 nats_server='nats://localhost:4222',
                 freq=98700000, span=2000000, stream=False, gain=0):  # only default arguments here

        gr.sync_block.__init__(
            self,
            name='NATS Sink',   # will show up in GRC
            in_sig=[(np.ushort, vector_size)],
            out_sig=None
        )

        avena_prefix = os.getenv('AVENA_PREFIX')
        self.vector_size = vector_size
        self.freq = freq
        self.span = span
        self.stream = stream
        self.gain = gain
        self.subject = subject
        #self.subject = avena_prefix + '.' + subject
        self.nc = NATSClient(nats_server, socket_timeout=2)
        self.nc.connect()

    def work(self, input_items, output_items):

        b64encpayload = str(b64encode(input_items[0][0]), 'utf-8')
        json_dump = json.dumps({'fft': b64encpayload,
                                'fc' : self.freq,
                                'gain' : self.gain,
                                'span' : self.span,
                                'fft_size' : self.vector_size,
                                'stream' : self.stream},
                                cls=NumpyEncoder)
        self.nc.publish(subject=self.subject ,payload=json_dump)

        return len(input_items[0])
```

Python code generated from GNURadio Companion

Controlled Data Streaming



SDR FM tuner at ISOBlue

WebRTC Multimedia gateway
in the cloud (server at Purdue)