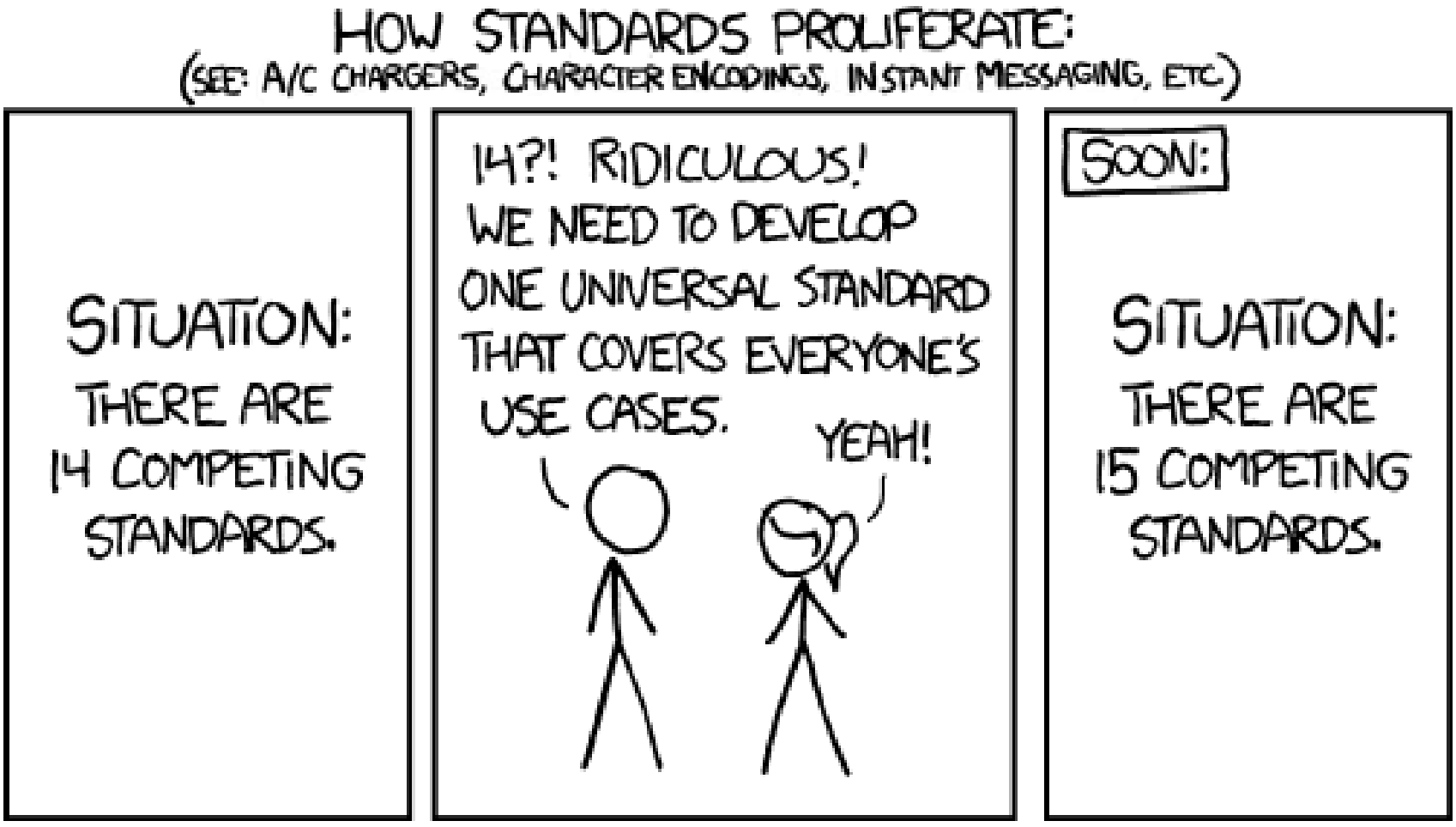
Case Studies in Robotics Toolchains

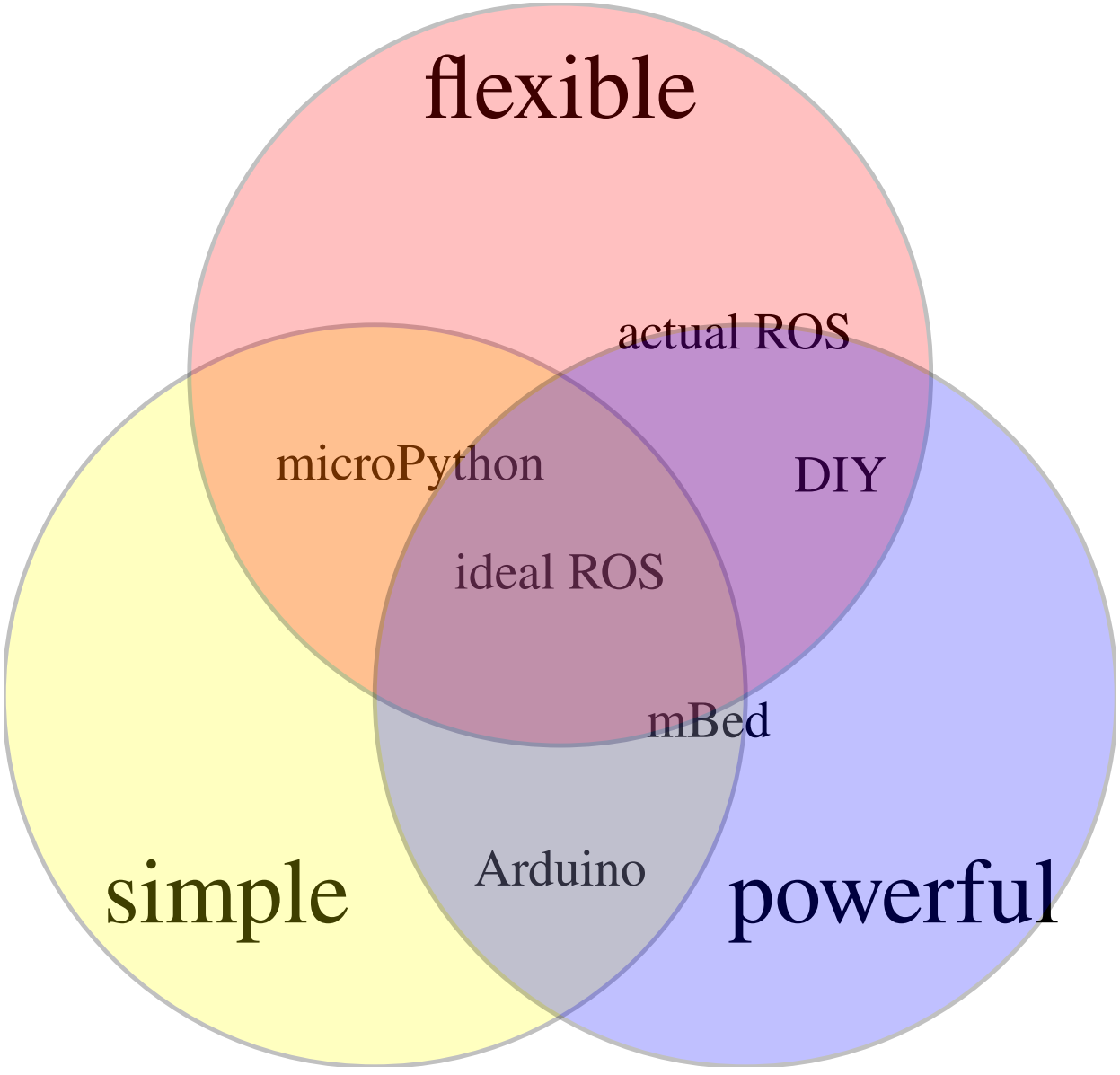
Alli Nilles



Roadmap

1. Case Studies
    1. Low level: Arduino
    2. High level: ROS
2. Lessons?
    1. power vs. simplicity vs. flexibility (vs. verifiability?)
    2. Ideas from functional programming

How to Evaluate Tools?



Case Study: Arduino Hacking

- Measure rotation of rolling ball robot, reconstruct motion
- Easy to get rough estimate, hard to get exact
- Need small footprint, onboard data logging

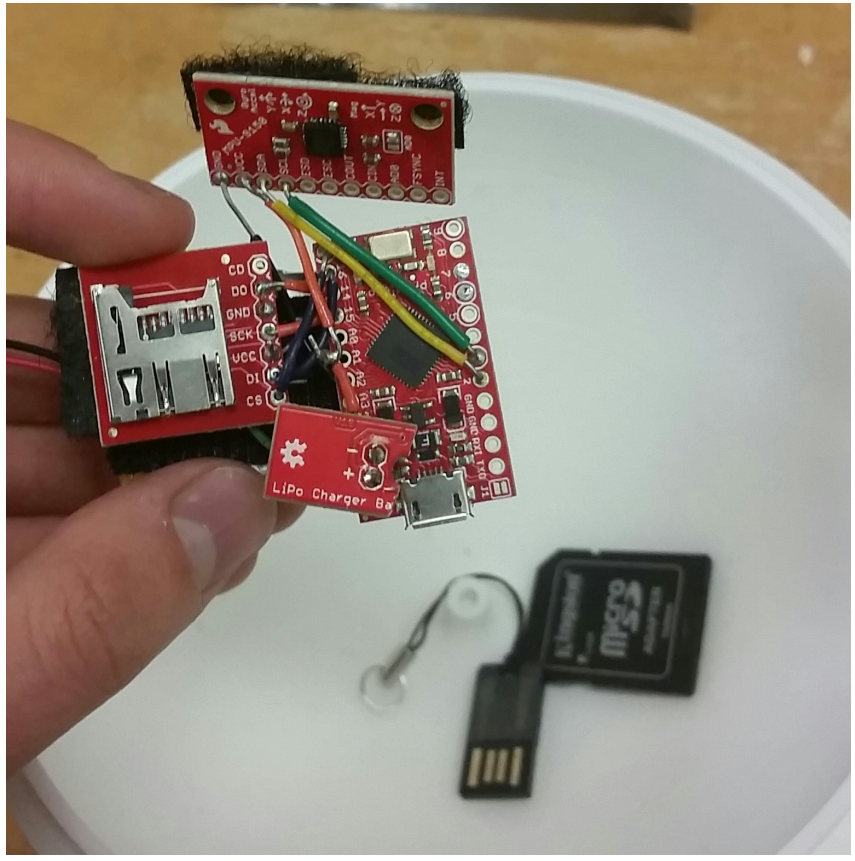


Figure 1.

Case Study: Takeaways

- Simplicity: where we tell beginners to start matters
  - Is traditional C++ style programming the best place to start?
- Powerful: large community
- Lose flexibility when tied to Arduino platforms/libraries/IDE

Possible solutions:

- mBed: about as simple and powerful, more flexible through online IDE
- possible fork into high-level and low-level languages
  - microPython
  - llvm for devices?

Case Study: ROS

My use cases:

- Data logging
- Playing with Spheros
- Bluetooth and ROS

Case Study: ROS

- flexible: large community, integrated with many platforms
- powerful: scalable framework, solid protocols
- simple?
  - networking issues
  - versioning issues: gazebo7 for Kinetic, gazebo2 for Indigo
  - XML / catkin

Case Study 3: ROS

```
<launch>
    <arg name="world" default="simple_world"/>
    <arg name="init_pos_x" default="0.0"/>
    <arg name="init_pos_y" default="0.0"/>

    <node pkg="gazebo_ros" type="spawn_model"
        name="spawn_robot"
        respawn="false" output="screen"
        args="-param robot_description
        -urdf
        -x $(arg init_pos_x)
        -y $(arg init_pos_y)
        -z $(arg init_pos_z)
        -model youbot">
    </node>
</launch>
```

Alternative

```
- world: simple_world
- init_pos_x: &init_pos_x 0.0
- init_pos_y: &init_pos_x 0.0
- node:
    - pkg: gazebo_ros
    - type: spawn_model
    - name: spawn_robot
    - respawn: false
    - output: screen
    - args:
        - param robot_description
        - urdf
        - x *arg init_pos_x
        - y *init_pos_y
        - z *arg init_pos_z
        - model youbot
```

Case Study 3: ROS

- Lack of simplicity leads to wasted time
- Low level timing issues left to user
  - callbacks, “spinOnce”
  - hard to enforce logic on Topics: what if I want to do something only when I have a pair of new values from two topics that publish at different rates?

Possible Solution: Haskell Client

Credit to Anthony Cowley,[1] UPenn GRASP lab

```
sayHello =
    Topic $ do threadDelay 1000000
        t <- getCurrentTime
        let msg = S.String ("Hello world " ++ show t)
        return (msg, sayHello)
```

Problems:

- time delay is brittle and hardcoded
- delay is orthogonal to task of operating on value
- verbose

[1] https://github.com/acowley/roshask

roshask

```
sayHello :: Topic IO S.String
sayHello = repeatM (fmap mkMsg getCurrentTime)
    where mkMsg = S.String . ("Hello world " ++) . show

main =  runNode "talker"
        $ advertise "chatter" (topicRate 1 sayHello)
```

Advantages:

- Can optimize topicRate and reuse
- Operation on data is independent of rate plumbing

Other nice things about roshask:

- `everyNew` function
- composable nodes: data on one machine is shared, not streamed, while preserving modularity

But why Haskell?

- High-level robotic control is functional in style
- Built around infinite lists: natural for robotics
- Type safety (already in ROS message types)
- Lack of side-effects means robots that act more consistently and are easier to verify
- Large community, Foreign Function Interface (FFI) is hardware-friendly