xFOCE Software Architecture Strategies

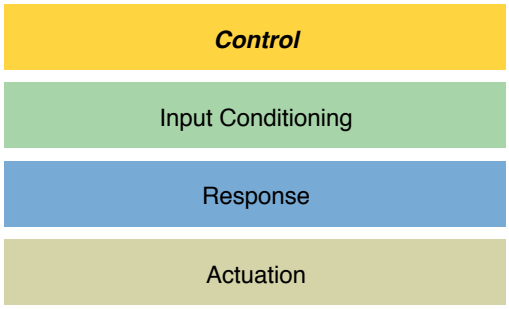
- Do a few things, simply and well
- Architecture should be tweakable by design
- Reduce barriers to using other hardware and reusing/modifying software
  - architectural pattern
  - software component choices

xFOCE has two main concerns:

controlling pH in a box
enable users to measure the effect on something

that are supported by two related concerns:

collecting data (recording measurements and context)
distributing data (internally and externally)



Consistency, symmetry and repetition make it easier to understand software.

- Separate concerns to keep components as simple as possible
- Minimize component count and arrange into simple patterns
- Reuse the patterns as much as possible
- Be consistent when applying/using patterns
- If something doesn't fit into an existing pattern
  - re-factor patterns (especially if several similar things don't quite fit)
  - create a new one

Features

- Small footprint
  - 25 MB disk space
  - 5 MB RAM (2% of 256MB)

- Download complete Beaglebone disk image
  - Easy setup scripts

- IO options
  - SensorNode
  - Local serial port
  - Digital IO
  - Analog IO
  - Terminal servers
  - USB

- Simple to update
  - configure/make/make install

- Simple, robust data storage
  - Segmented archives
  - Indexed for fast access

- JSON configuration

- Remote configuration

- Unit tests

- [? install on VM]
- [? run multiple instances on single platform]
- [? portable...]

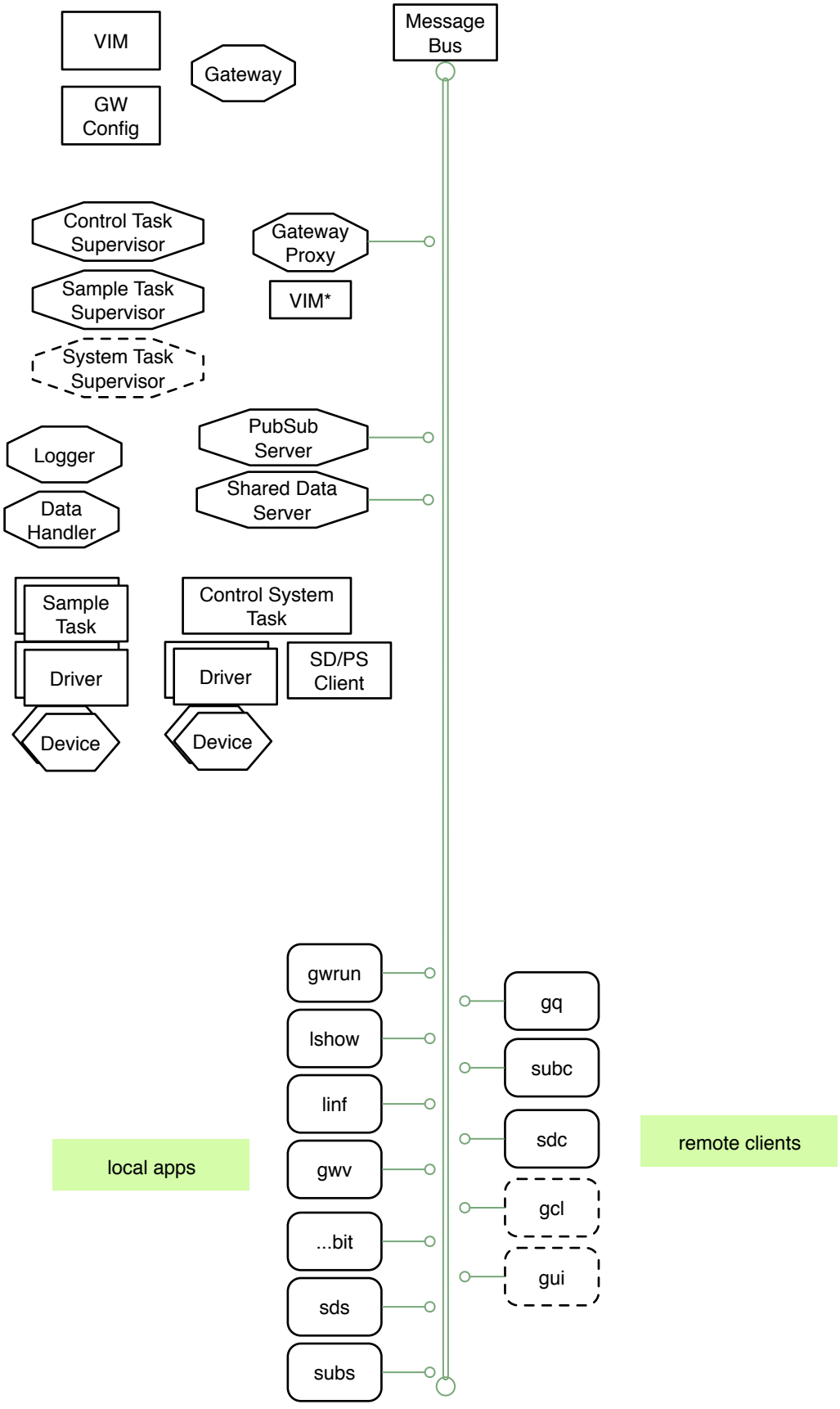
Specs

- Base system footprint
  - 25 MB disk space
  - 5 MB RAM (2% of 256MB)

- Beaglebone A6
  - 720MHz ARM (AM3359)
  - 256 MB RAM
  - 5 serial
  - Ethernet
  - GPIO
  - Analog IO
  - Terminal servers
  - USB
  - MicroSD storage

- Beaglebone stock Angstrom Linux
  - support packages pre-installed

- Performance Benchmark
  - 10 services @ 2 Hz
  - 4% CPU (@720 MHz)
  - 1.4 % Mem (10.25/256MB)
  - low jitter (metric?)



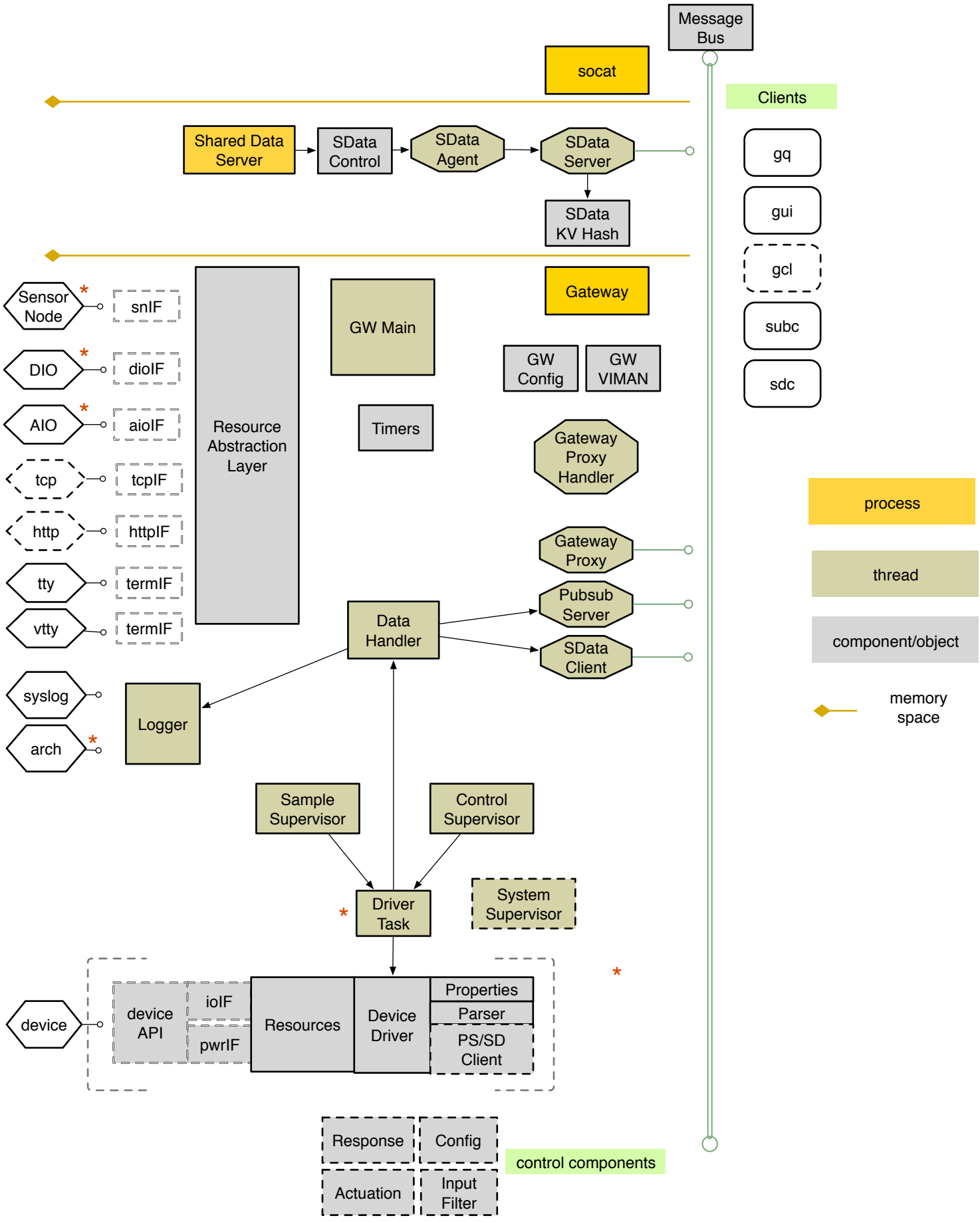
Sampling components direct sampling:
- what, when

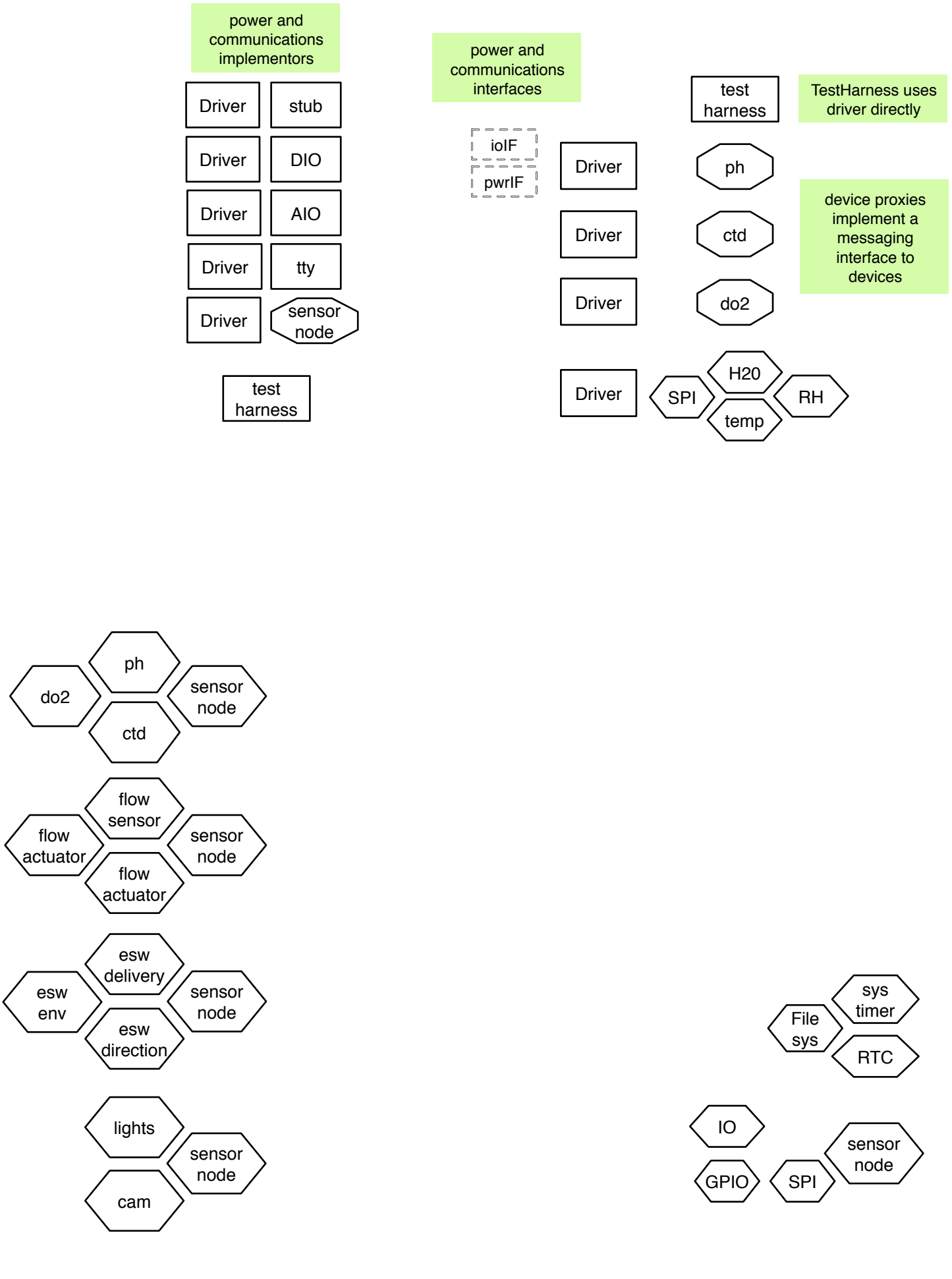
Device components direct that produce data
- acquire, separate, notify

Data components direct data:
- where, what, when

Manager components direct resources:
- creation, configuration, access

Control components direct closed loop control
- read sensor data
- set actuation values


Message Bus
socat
Clients
Shared Data Server
SData Control
SData Agent
SData Server
SData KV Hash
gq
gui
gcl
subc
sdc
Gateway
GW Main
GW Config
GW VIMAN
Timers
Gateway Proxy Handler
Resource Abstraction Layer
Sensor Node
snIF
DIO
dioIF
AIO
aioIF
tcp
tcpIF
http
httpIF
tty
termIF
vtty
termIF
syslog
arch
Logger
Data Handler
Gateway Proxy
Pubsub Server
SData Client
Sample Supervisor
Control Supervisor
Driver Task
System Supervisor
device
device API
ioIF
pwrIF
Resources
Device Driver
Properties
Parser
PS/SD Client
Response
Config
Actuation
Input Filter
control components
process
thread
component/object
memory space


power and communications implementors
Driver
stub
Driver
DIO
Driver
AIO
Driver
tty
Driver
sensor node
test harness
power and communications interfaces
ioIF
pwrIF
test harness
TestHarness uses driver directly
Driver
ph
Driver
ctd
Driver
do2
device proxies implement a messaging interface to devices
Driver
H20
SPI
RH
temp
ph
do2
sensor node
ctd
flow sensor
flow actuator
sensor node
flow actuator
esw delivery
esw env
sensor node
esw direction
lights
sensor node
cam
sys timer
File sys
RTC
IO
sensor node
GPIO
SPI

Objectives
minimize data handling
minimize network bandwidth
minimize number of packets read
minimize processes
minimize threading
minimize state maintenance
minimize serialization overhead
minimize copying (data)

pub/sub
request/response

reliable/deterministic sampling
reliable logging
scalable
distributable
portability

Easy to understand, use and modify

Implementation Strategies

Single threaded logging
Only two process (Gateway, SDS)
Develop reusable components for core functions:
- thread management
- job queues
- lists
- debug messages
- logging
- buffers
- memory allocation
- properties
- serialization

Use memory counting to reduce copies

Abstract IO and provide simple consistent interfaces for driver developers

Use mainstream open-source development tools
- Autotools, Doxygen, Mercurial

Adopt applicable standards
- JSON
- POSIX

Use Cases

get status
- sampler workers
- drivers

get/set configuration
- gateway
- supervisor
- sample worker
- driver
- control sys

start/stop/restart
- network
- device
- control sys

acquire sample
- driver

Develop a new driver

Port to new distro
Port to new OS

Metrics
pH control stability
sample jitter
services*sample rate
CPU bandwidth
Memory use
Archive overhead
Quick Start time