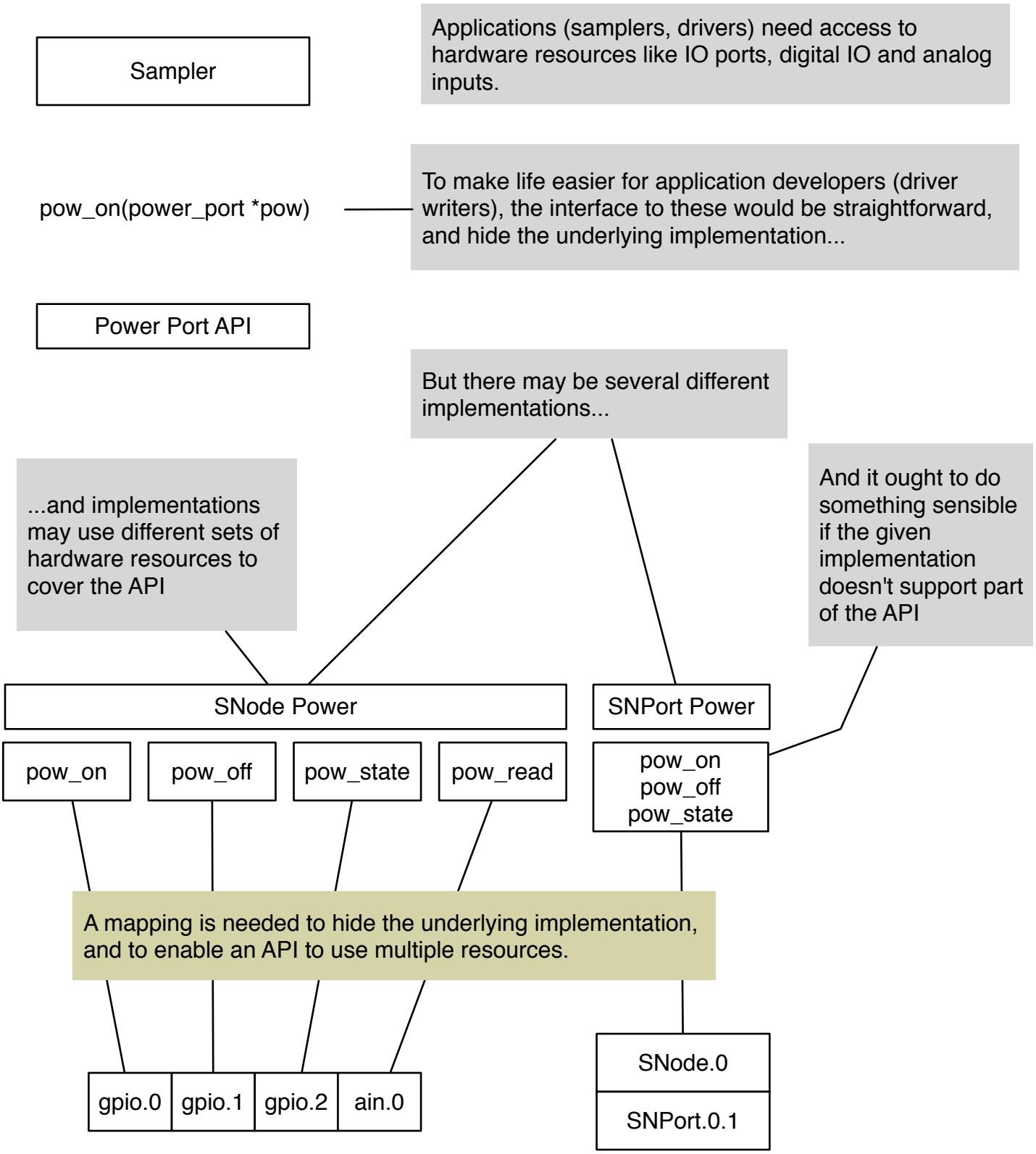
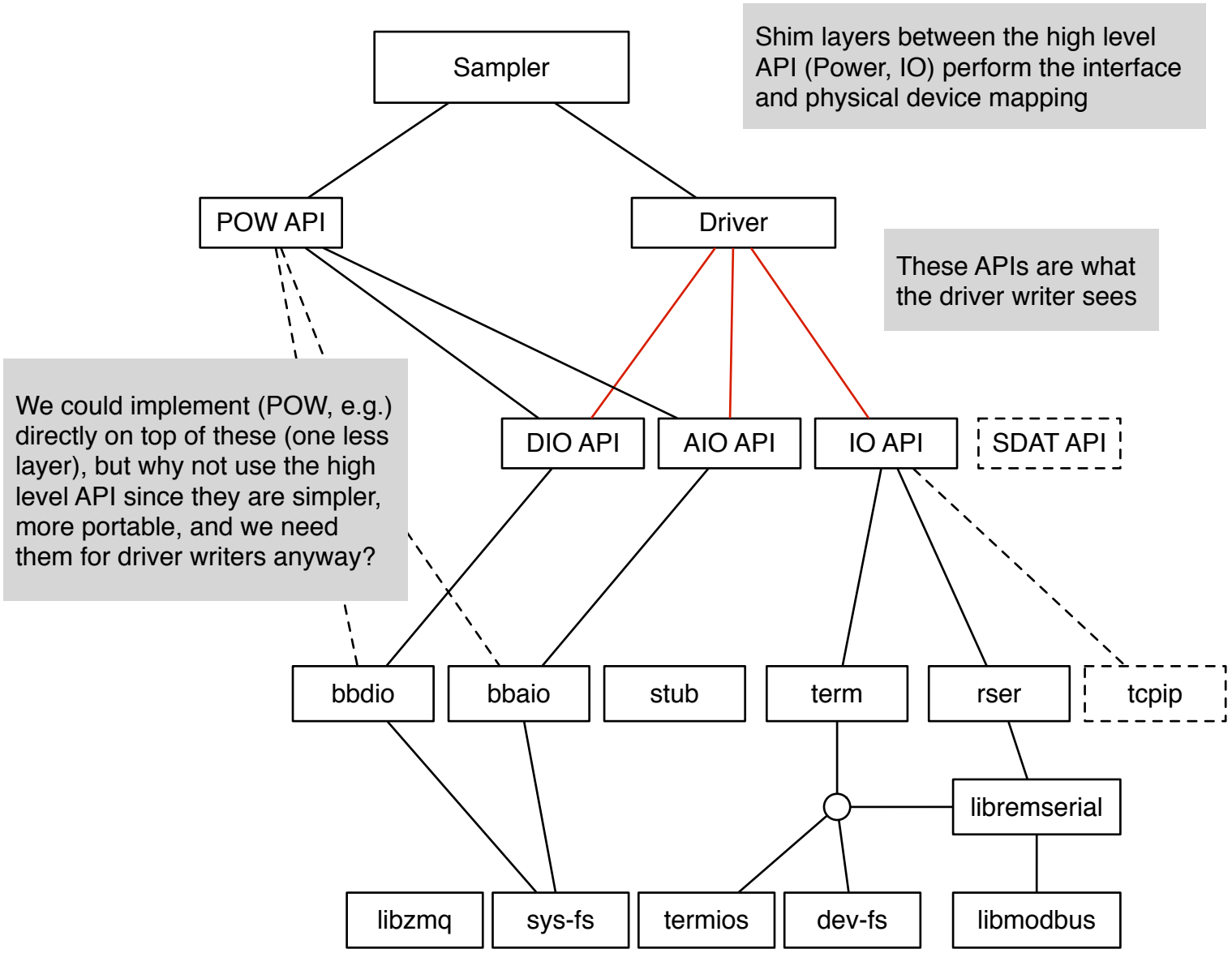
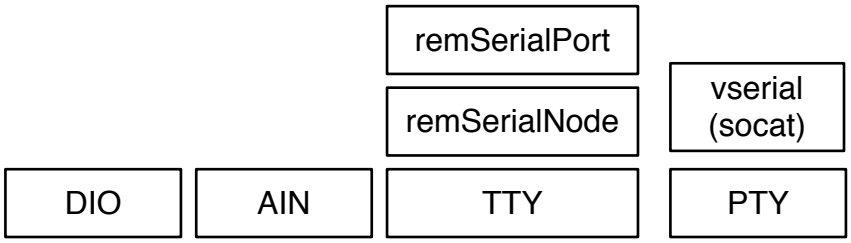
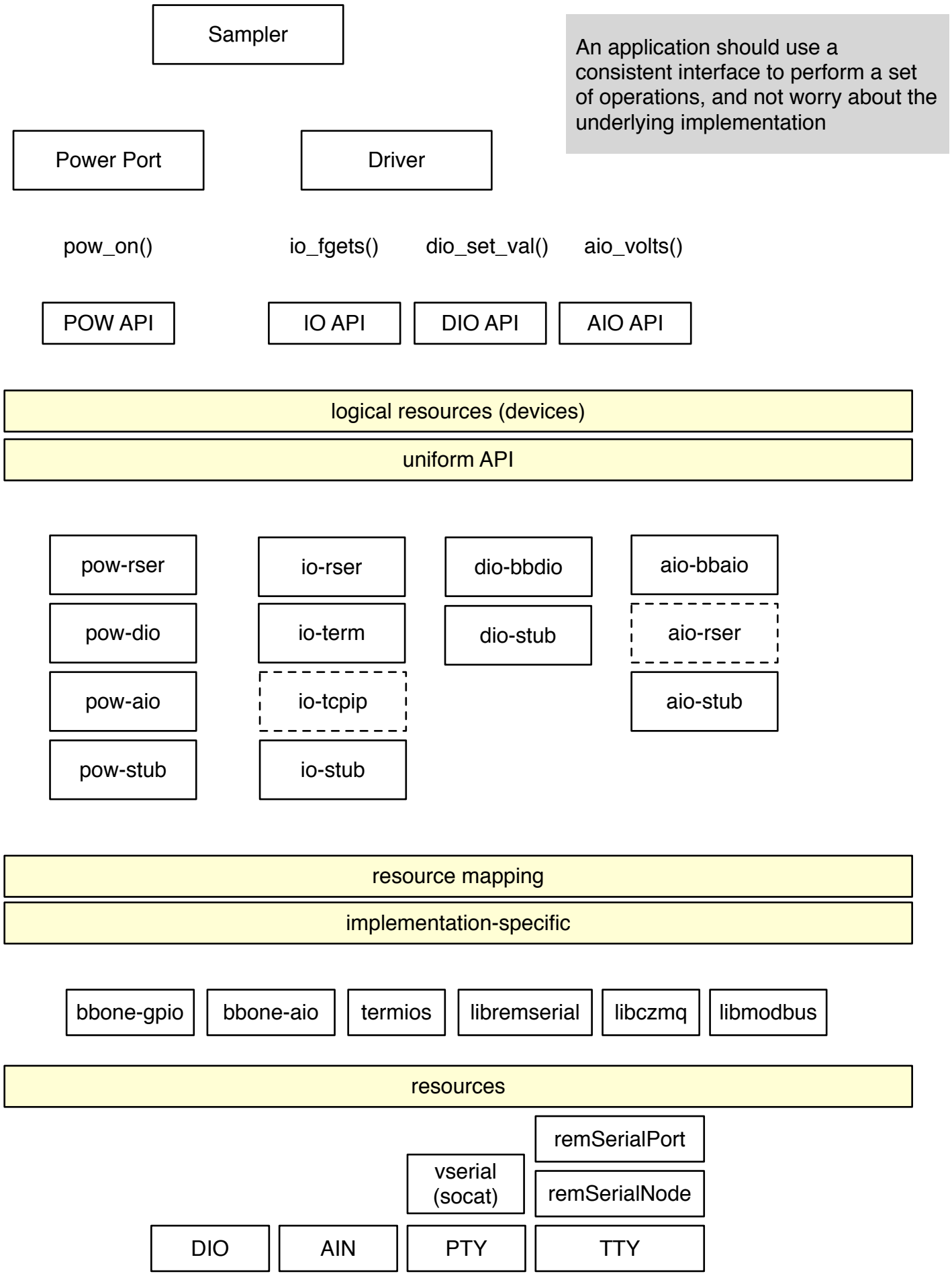

Sampler
Applications (samplers, drivers) need access to hardware resources like IO ports, digital IO and analog inputs.
pow_on(power_port *pow)
To make life easier for application developers (driver writers), the interface to these would be straightforward, and hide the underlying implementation...
Power Port API
But there may be several different implementations...
...and implementations may use different sets of hardware resources to cover the API
And it ought to do something sensible if the given implementation doesn't support part of the API
SNode Power
SNPort Power
pow_on
pow_off
pow_state
pow_read
pow_on
pow_off
pow_state
A mapping is needed to hide the underlying implementation, and to enable an API to use multiple resources.
gpio.0
gpio.1
gpio.2
ain.0
SNode.0
SNPort.0.1


Shim layers between the high level API (Power, IO) perform the interface and physical device mapping
Sampler
POW API
Driver
These APIs are what the driver writer sees
We could implement (POW, e.g.) directly on top of these (one less layer), but why not use the high level API since they are simpler, more portable, and we need them for driver writers anyway?
DIO API
AIO API
IO API
SDAT API
bbdio
bbaio
stub
term
rser
tcpip
libremserial
libzmq
sys-fs
termios
dev-fs
libmodbus


remSerialPort
remSerialNode
vserial (socat)
DIO
AIN
TTY
PTY


Sampler
An application should use a consistent interface to perform a set of operations, and not worry about the underlying implementation
Power Port
Driver
pow_on()
io_fgets()
dio_set_val()
aio_volts()
POW API
IO API
DIO API
AIO API
logical resources (devices)
uniform API
pow-rser
pow-dio
pow-aio
pow-stub
io-rser
io-term
io-tcpip
io-stub
dio-bbdio
dio-stub
aio-bbaio
aio-rser
aio-stub
resource mapping
implementation-specific
bbone-gpio
bbone-aio
termios
libremserial
libczmq
libmodbus
resources
remSerialPort
vserial (socat)
remSerialNode
DIO
AIN
PTY
TTY