Integrating Concurrency Control and Energy Management in Device Drivers

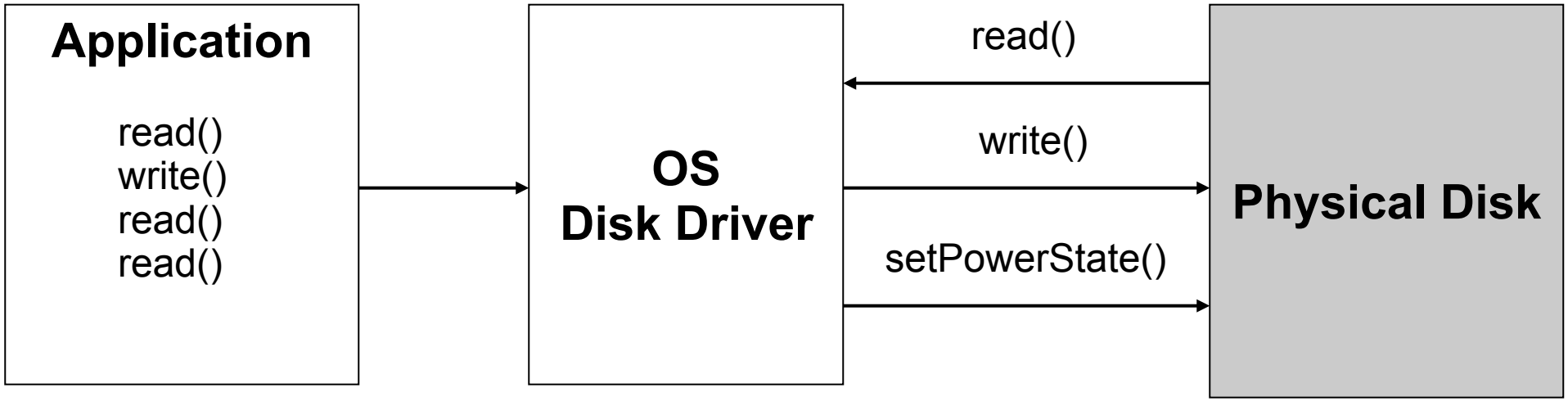
Kevin Klues, Vlado Handziski, Chenyang Lu, Adam Wolisz, David Culler, David Gay, and Philip Levis

Overview

- Concurrency Control:
  - Concurrency of I/O operations alone, not of threads in general
  - Synchronous vs. Asynchronous I/O
- Energy Management:
  - Power state of devices needed to perform I/O operations
  - Determined by pending I/O requests using Asynchronous I/O

Overview

- Concurrency Control:
  - Concurrency of I/O operations alone, not of threads in general
  - Synchronous vs. Asynchronous I/O
- Energy Management:
  - Power state of devices needed to perform I/O operations
  - Determined by pending I/O requests using Asynchronous I/O



Overview

- Concurrency Control:
  - Concurrency of I/O operations alone, not of threads in general
  - Synchronous vs. Asynchronous I/O
- Energy Management:
  - Power state of devices needed to perform I/O operations
  - Determined by pending I/O requests using Asynchronous I/O

The more workload information an application can give the OS, the more energy it can save when scheduling that workload

Outline

- **Background Information**
- Platform and Application
- Driver architecture
- Evaluation
- Conclusion

Motivation

- Difficult to manage energy in traditional OSs
  - Hard to tell OS about future application workloads
  - All logic pushed out to the application
  - API extensions for hints?

Existing OS Approaches

- Dynamic CPU Voltage Scaling
  - Vertigo - Application workload classes
  - Grace OS - Explicit realtime deadlines
- Disk Spin Down
  - Coop-IO - Application specified timeouts

Existing OS Approaches

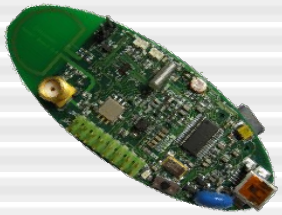
- Dynamic CPU Voltage Scaling
  - Vertigo - Application workload classes
  - Grace OS - Explicit realtime deadlines
- Disk Spin Down
  - Coop-IO - Application specified timeouts

Saving energy is a complex process

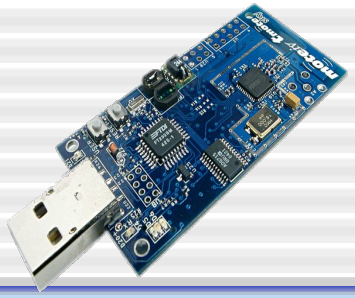
Existing OS Approaches

- Dynamic CPU Voltage Scaling
  - Vertigo - Application workload classes
  - Grace OS - Explicit realtime deadlines
- Disk Spin Down
  - Coop-IO - Application specified timeouts

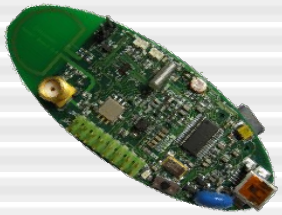
Saving energy is a complex process
A little application knowledge can help us alot



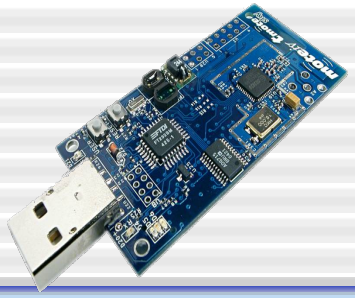
Sensor Networks



- Domain in need of unique solution to this problem
  - Harsh energy requirements
  - Very small source of power (2AA batteries)
  - Must run unattended from months to years



Sensor Networks



- Domain in need of unique solution to this problem
  - Harsh energy requirements
  - Very small source of power (2AA batteries)
  - Must run unattended from months to years
- First generation sensornet OSes (TinyOS, Contiki, Mantis, ...)
  - Push all energy management to the application
  - Optimal energy savings at cost of application complexity

ICEM: Integrated Concurrency and Energy Management

- A device driver architecture that automatically manages energy
  - Implemented in TinyOS 2.0 -- all drivers follow it
  - Introduces *Power Locks,* split-phase locks with integrated energy and configuration management
  - Defines three classes of drivers: dedicated, shared, virtualized
  - Provides a component library for building drivers

ICEM: Integrated Concurrency and Energy Management

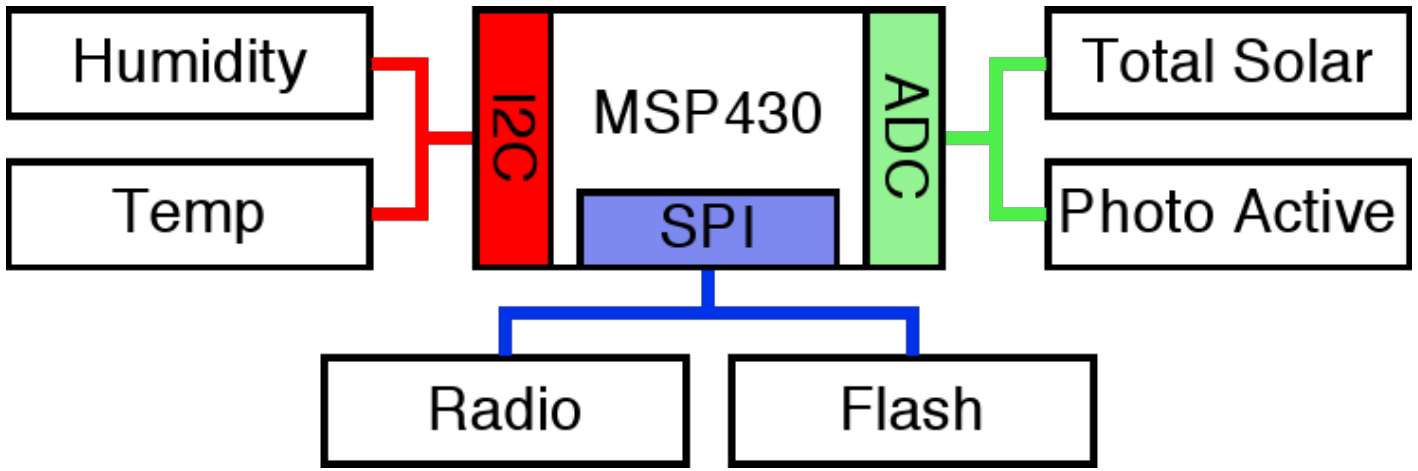
- A device driver architecture that automatically manages energy
  - Implemented in TinyOS 2.0 -- all drivers follow it
  - Introduces *Power Locks,* split-phase locks with integrated energy and configuration management
  - Defines three classes of drivers: dedicated, shared, virtualized
  - Provides a component library for building drivers
- Advantages of using **ICEM**
  - Energy efficient – At least 98.4% as hand-tuned implementation
  - Reduces code complexity – 400 vs. 68 lines of code
  - Enables natural decomposition of applications

Outline

- Introduction and Motivation
- **Platform and Application**
- ICEM architecture
- Evaluation
- Conclusion

The Tmote Platform

- Six major I/O devices
- Possible Concurrency
  - I^2C, SPI, ADC
- Energy Management
  - Turn peripherals on only when needed
  - Turn off otherwise



Representative Logging Application

Producer

```
Every 5 minutes:
  Write prior samples
  Sample photo active
  Sample total solar
  Sample temperature
  Sample humidity
```

Consumer

```
Every 12 hours:
  For all new entries:
    Send current sample
    Read next sample
```

Flash

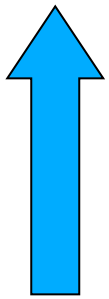
Sensors

Radio

Representative Logging Application

Producer

```
Every 5 minutes:
  Write prior samples
  Sample photo active
  Sample total solar
  Sample temperature
  Sample humidity
```



Sensors

Flash

Consumer

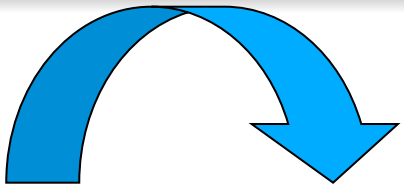
```
Every 12 hours:
  For all new entries:
    Send current sample
    Read next sample
```

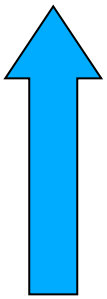
Radio

Representative Logging Application

Producer

```
Every 5 minutes:
  Write prior samples
  Sample photo active
  Sample total solar
  Sample temperature
  Sample humidity
```





Flash

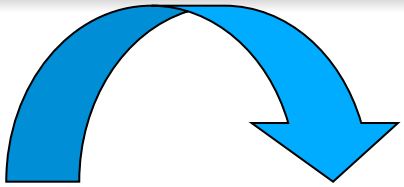
Sensors

Consumer

```
Every 12 hours:
  For all new entries:
    Send current sample
    Read next sample
```

Radio

Representative Logging Application

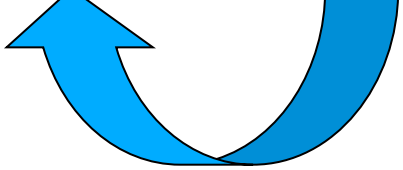


Producer

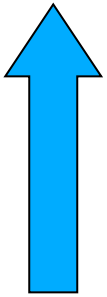
```
Every 5 minutes:
  Write prior samples
  Sample photo active
  Sample total solar
  Sample temperature
  Sample humidity
```

Consumer

```
Every 12 hours:
  For all new entries:
    Send current sample
    Read next sample
```



Flash



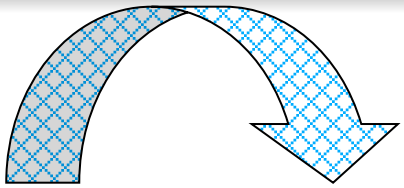
Sensors

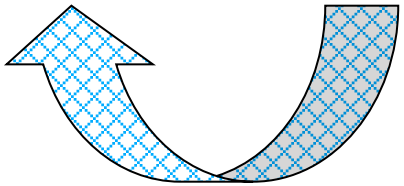
Radio

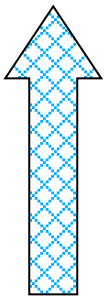
Representative Logging Application

Producer

```
Every 5 minutes:
  Write prior samples
  Sample photo active
  Sample total solar
  Sample temperature
  Sample humidity
```







Flash

Sensors

Consumer

```
Every 12 hours:
  For all new entries:
    Send current sample
    Read next sample
```

Radio

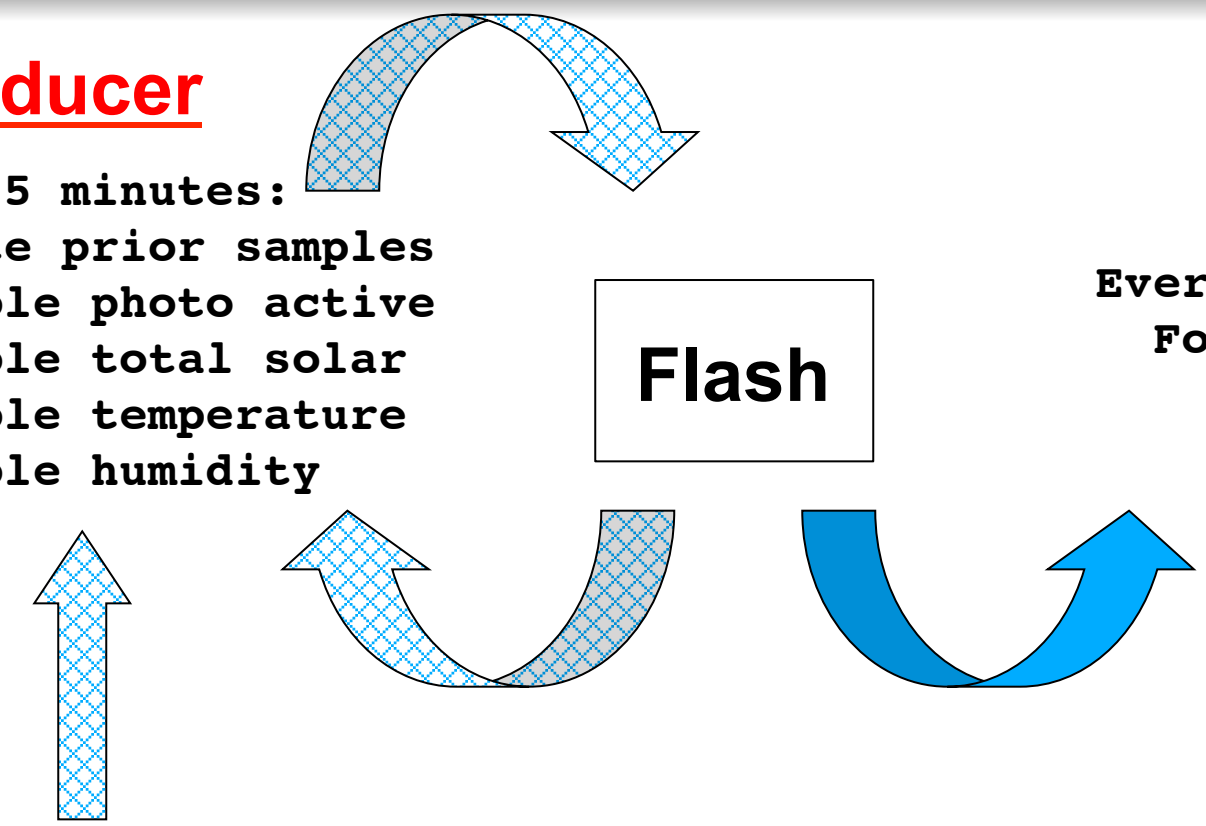
Representative Logging Application

Producer

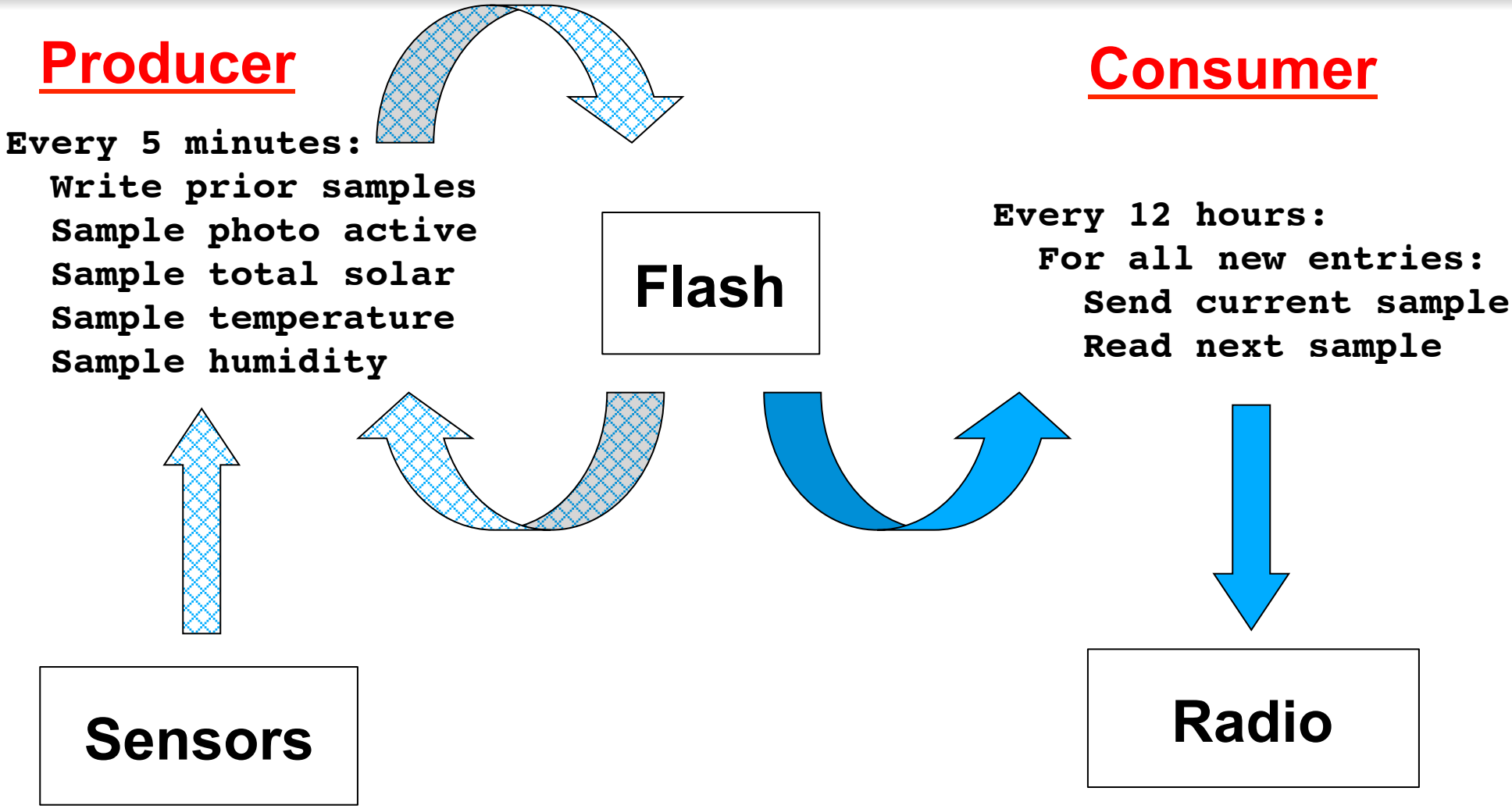
```
Every 5 minutes:
  Write prior samples
  Sample photo active
  Sample total solar
  Sample temperature
  Sample humidity
```

Consumer

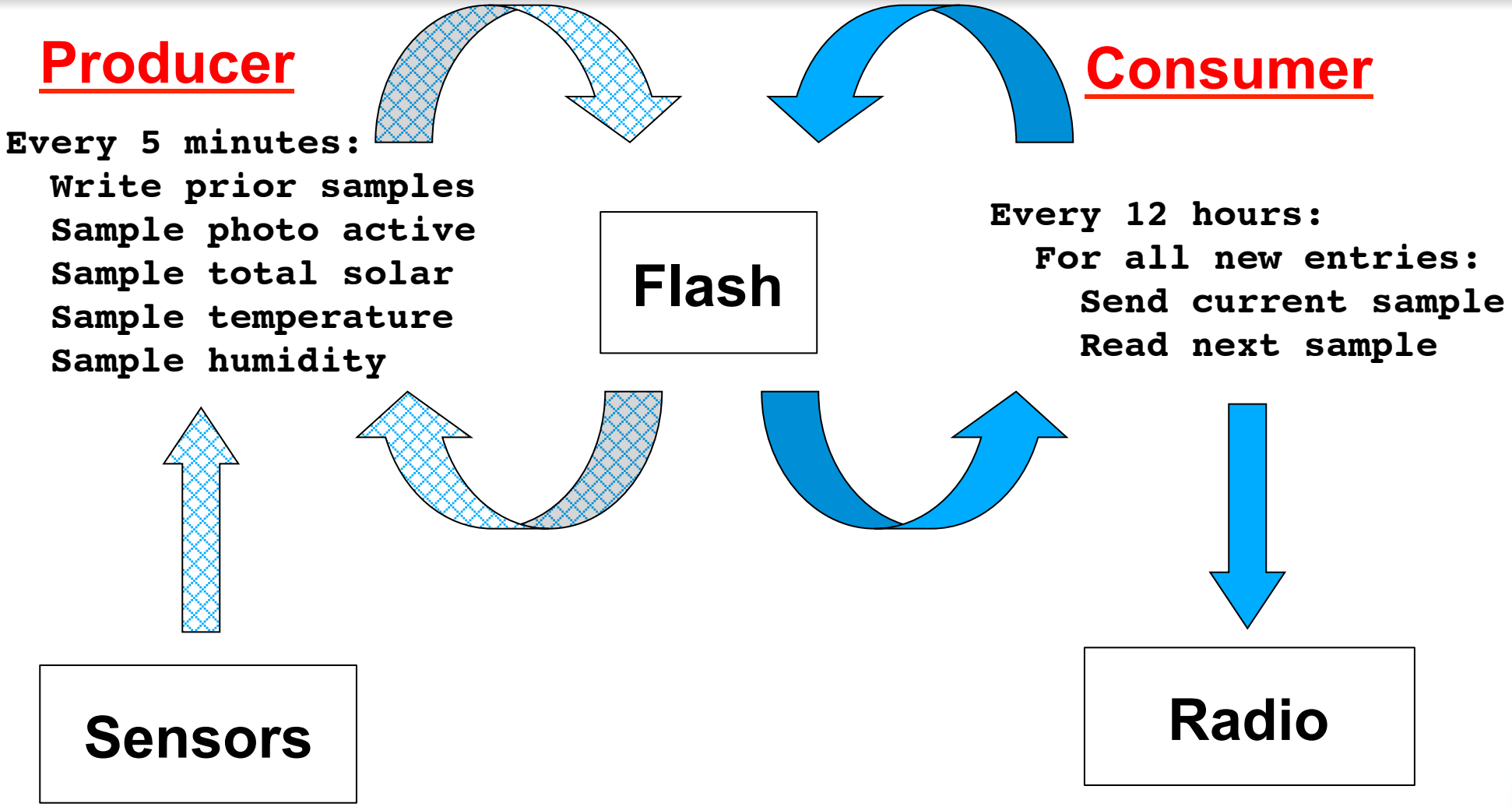
```
Every 12 hours:
  For all new entries:
    Send current sample
    Read next sample
```



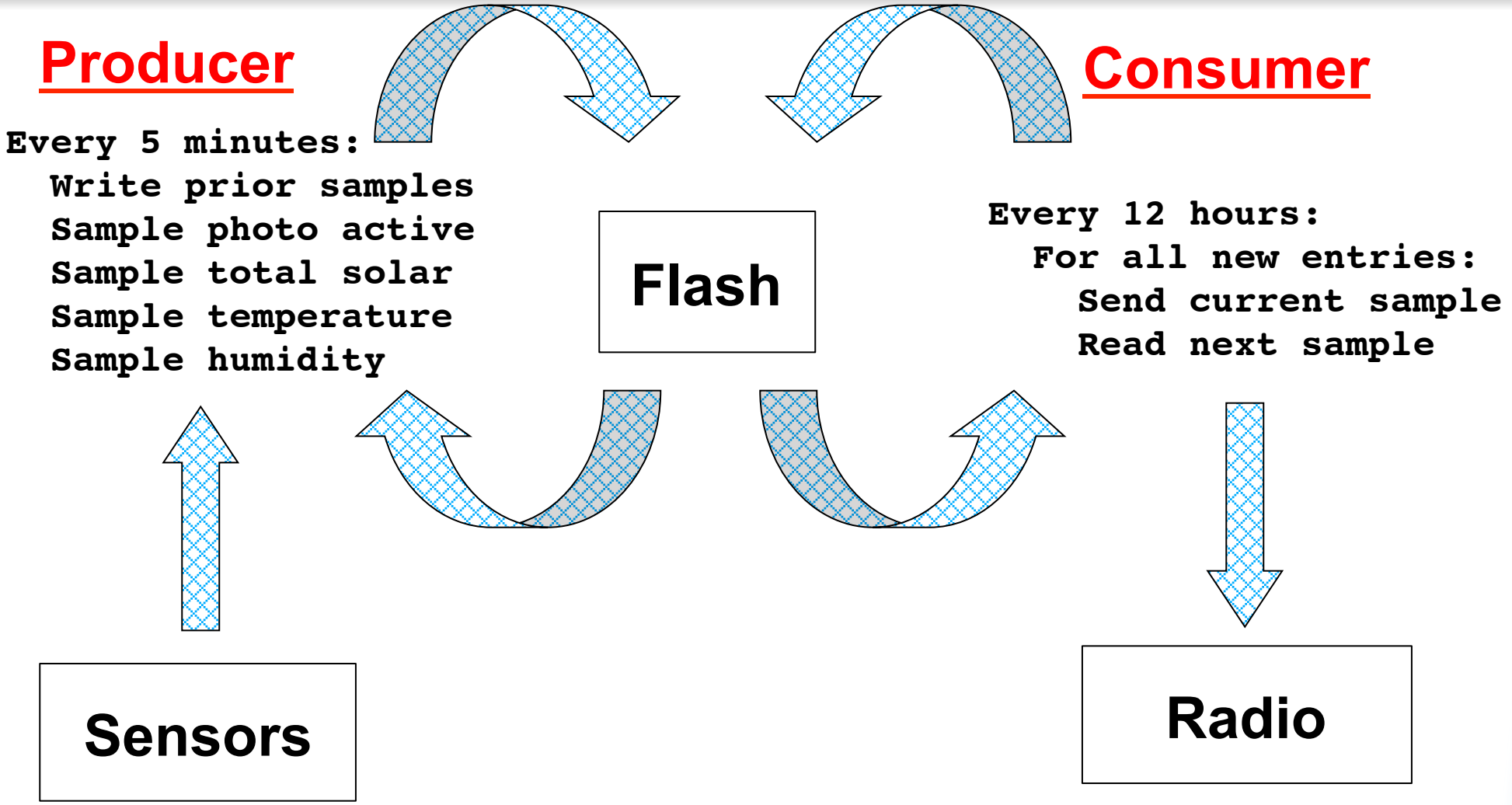
Representative Logging Application



Representative Logging Application



Representative Logging Application



Code Complexity

Hand-Tuned Application

Code Complexity

Hand-Tuned Application

```
Every 5 minutes:
  Turn on SPI bus
  Turn on flash chip
  Turn on voltage reference
  Turn on I²C bus
  Log prior readings
  Start humidity sample
  Wait 5ms for log
  Turn off flash chip
  Turn off SPI bus
  Wait 12ms for vref
  Turn on ADC
  Start total solar sample
  Wait 2ms for total solar
  Start photo active sample
  Wait 2ms for photo active
  Turn off ADC
  Turn off voltage reference
  Wait 34ms for humidity
  Start temperature sample
  Wait 220ms for temperature
  Turn off I²C bus
```

Code Complexity

Hand-Tuned Application

```
Every 5 minutes:
  Turn on SPI bus
  Turn on flash chip
  Turn on voltage reference
  Turn on I²C bus
  Log prior readings
  Start humidity sample
  Wait 5ms for log
  Turn off flash chip
  Turn off SPI bus
  Wait 12ms for vref
  Turn on ADC
  Start total solar sample
  Wait 2ms for total solar
  Start photo active sample
  Wait 2ms for photo active
  Turn off ADC
  Turn off voltage reference
  Wait 34ms for humidity
  Start temperature sample
  Wait 220ms for temperature
  Turn off I²C bus
```

Code Complexity

Hand-Tuned Application

```
Every 5 minutes:
  Turn on SPI bus
  Turn on flash chip
  Turn on voltage reference
  Turn on I²C bus
  Log prior readings
  Start humidity sample
  Wait 5ms for log
  Turn off flash chip
  Turn off SPI bus
  Wait 12ms for vref
  Turn on ADC
  Start total solar sample
  Wait 2ms for total solar
  Start photo active sample
  Wait 2ms for photo active
  Turn off ADC
  Turn off voltage reference
  Wait 34ms for humidity
  Start temperature sample
  Wait 220ms for temperature
  Turn off I²C bus
```

Code Complexity

Hand-Tuned Application

```
Every 5 minutes:
  Turn on SPI bus
  Turn on flash chip
  Turn on voltage reference
  Turn on I²C bus
  Log prior readings
  Start humidity sample
  Wait 5ms for log
  Turn off flash chip
  Turn off SPI bus
  Wait 12ms for vref
  Turn on ADC
  Start total solar sample
  Wait 2ms for total solar
  Start photo active sample
  Wait 2ms for photo active
  Turn off ADC
  Turn off voltage reference
  Wait 34ms for humidity
  Start temperature sample
  Wait 220ms for temperature
  Turn off I²C bus
```

ICEM Application

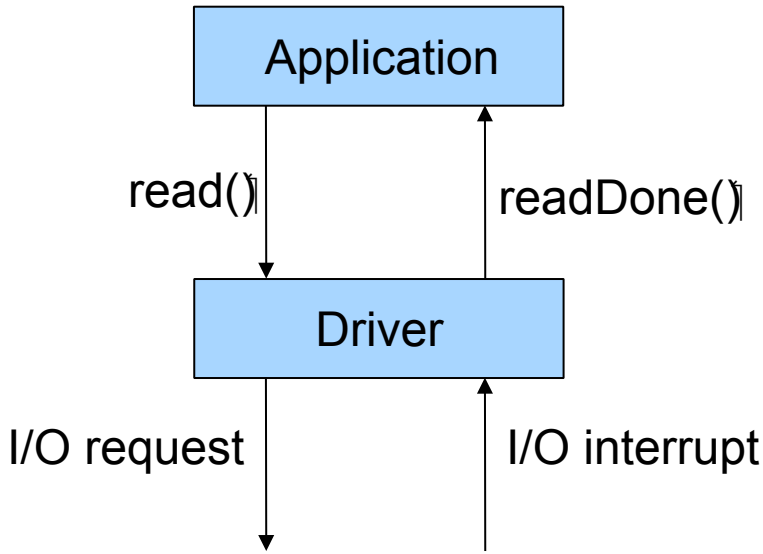
```
Every 5 minutes:
  Log prior readings
  sample humidity
  sample total solar
  sample photo active
  sample temperature
```

Outline

- Introduction and Motivation
- Platform and Application
- **ICEM architecture**
- Evaluation
- Conclusion

Split-Phase I/O Operations

- Split-phase I/O operations
  - Implemented within a single thread of control
  - Application notified of I/O completion through direct upcall
  - Driver given workload information before returning control
  - Example: read() ⇒ readDone()



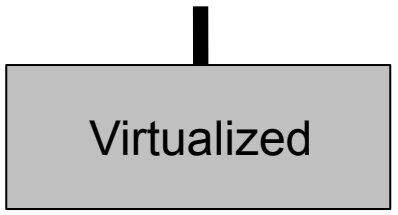

```
void readDone(uint16_t val) {
  next_val = val;
  read();
}
```

ICEM Architecture

- Defines three classes of drivers
  - Virtualized – provide only functional interface
  - Dedicated – provide functional and power interface
  - Shared – provide functional and lock interface

Virtualized Device Drivers

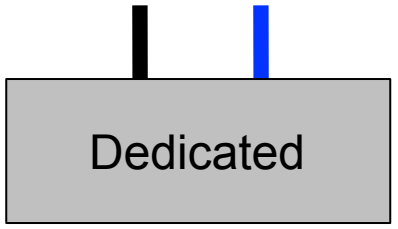
- Provide only a **Functional** interface
- Assume multiple users
- ***Implicit*** concurrency control through buffering requests
- ***Implicit*** energy management based on pending requests
- Implemented for higher-level services that can tolerate longer latencies



Energy: Implicit
Concurrency: Implicit

Dedicated Device Drivers

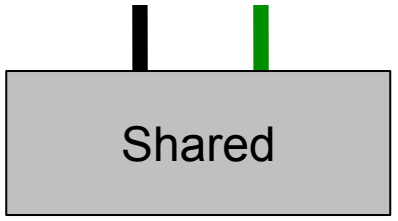
- Provide **Functional** and **Power Control** interfaces
- Assume a single user
- ***No*** concurrency control
- *Explicit* energy management
- Low-level hardware and bottom-level abstractions have a dedicated driver



Energy: Implicit
Concurrency: None

Shared Device Drivers

- Provide **Functional** and **Lock** interfaces
- Assume multiple users
- ***Explicit*** concurrency control through Lock request
- ***Implicit*** energy management based on pending requests
- Used by users with stringent timing requirements

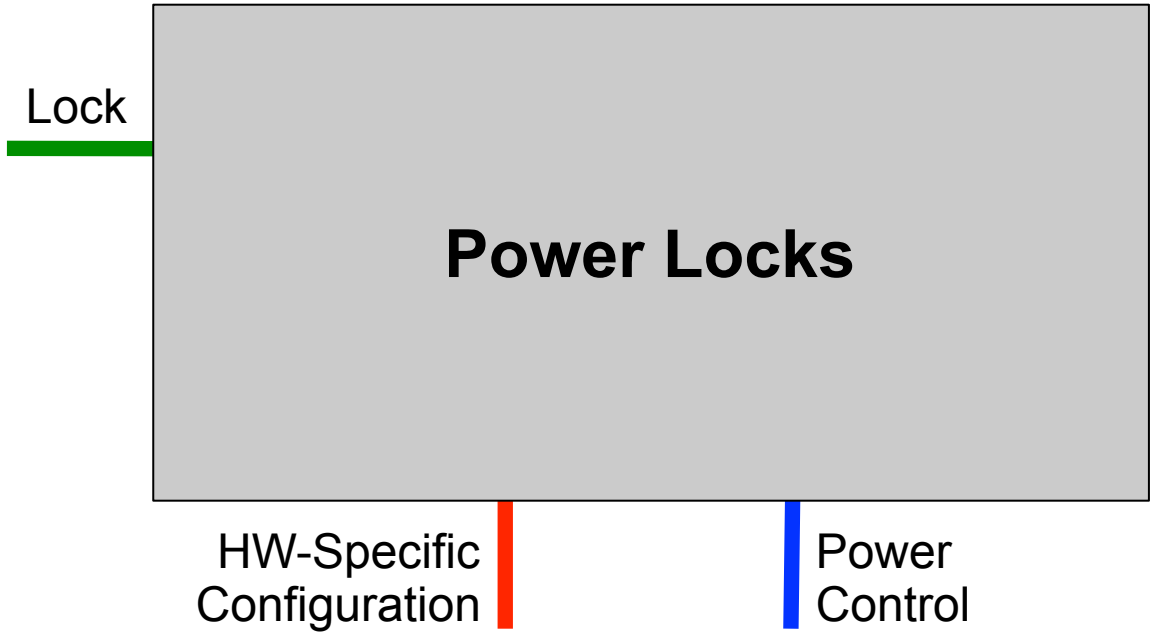


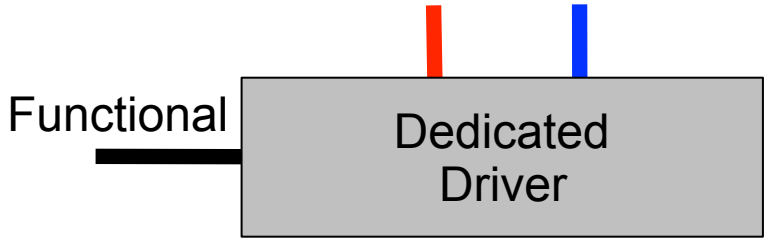
Energy: Implicit
Concurrency: Explicit

ICEM Architecture

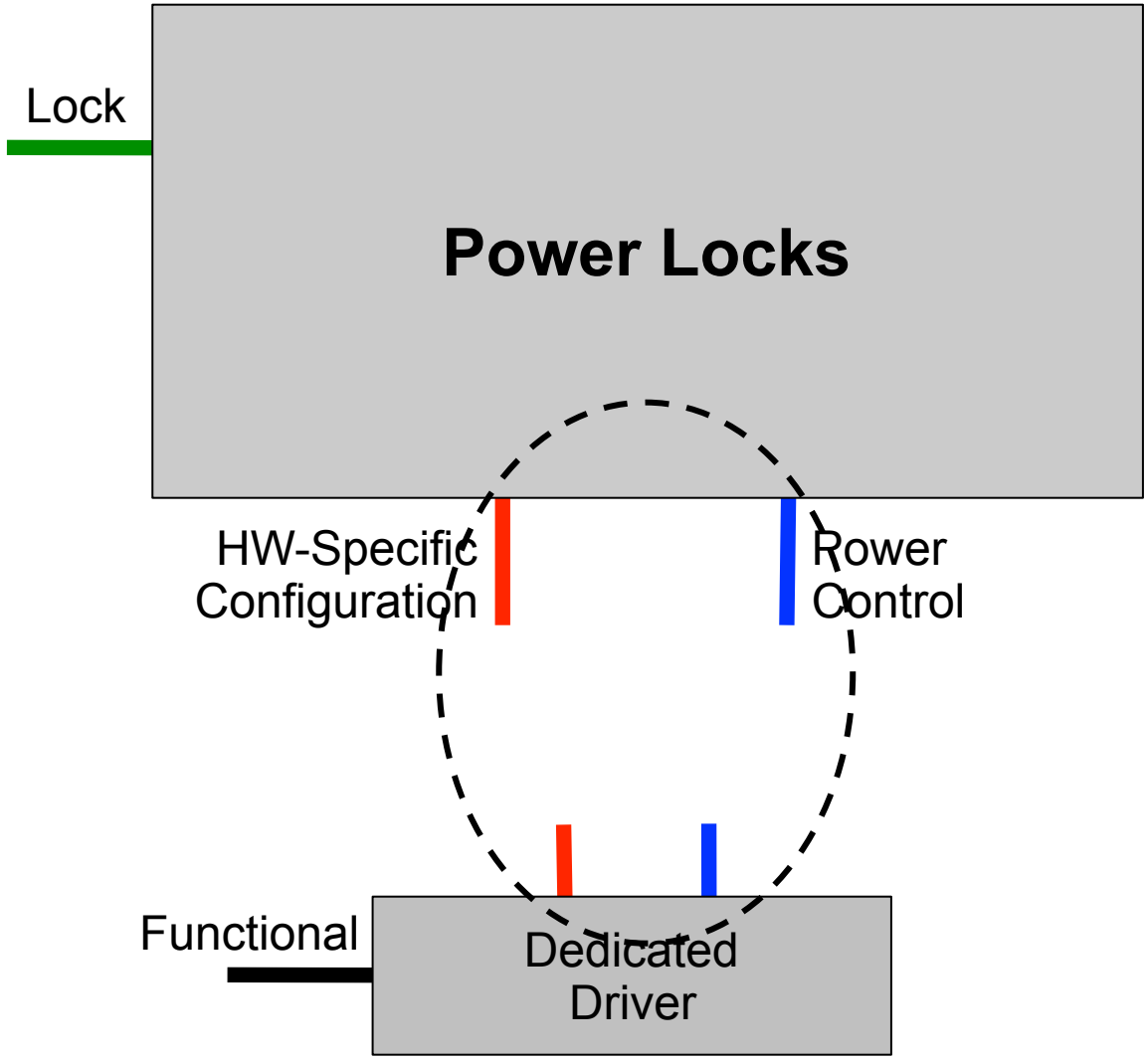
- Defines three classes of drivers
  - Virtualized – provide only functional interface
  - Dedicated – provide functional and power interface
  - Shared – provide functional and lock interface
- *Power Locks*, split-phase locks with integrated energy and configuration management

Power Locks

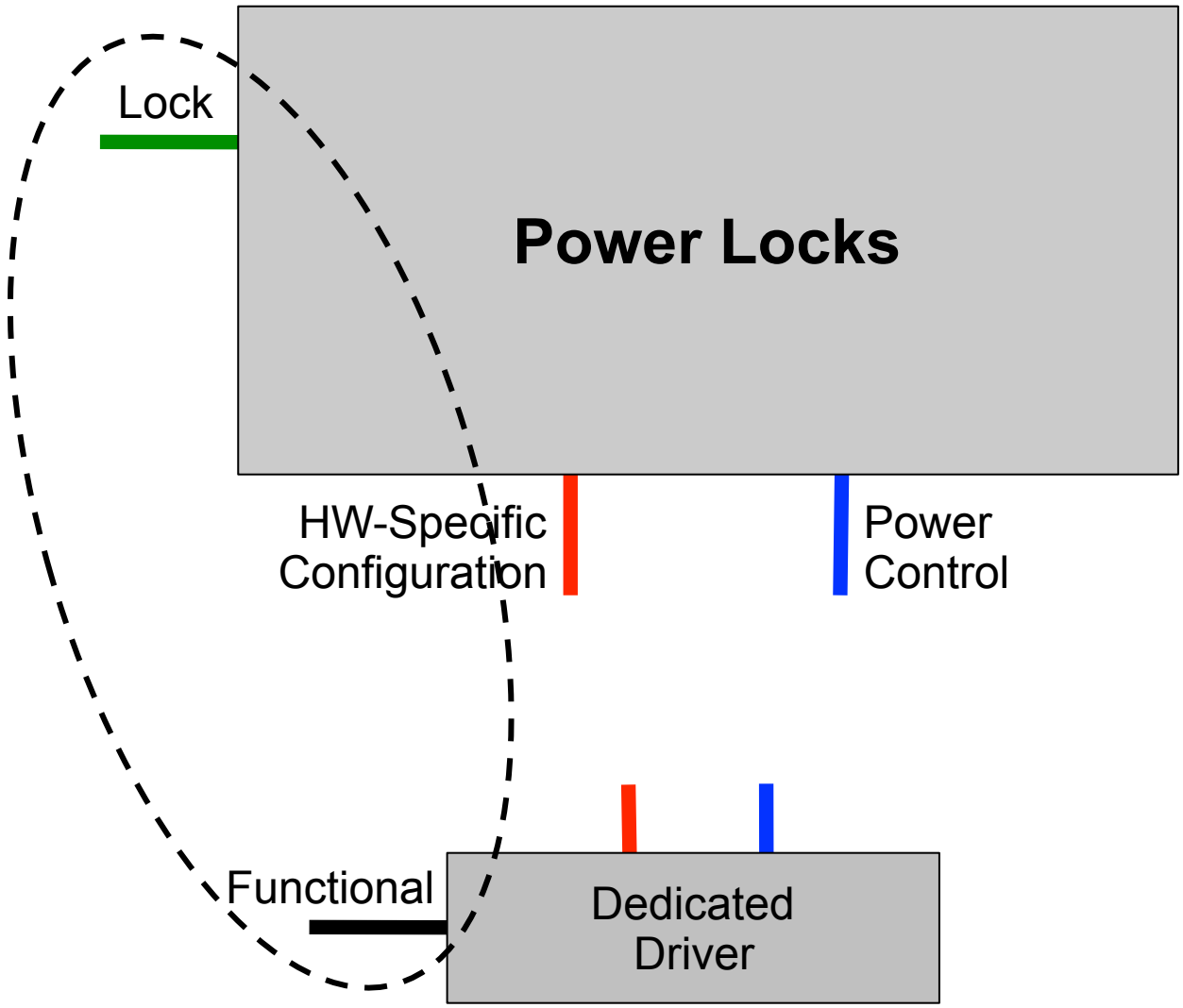

Lock
Power Locks
HW-Specific Configuration
Power Control


Functional
Dedicated Driver

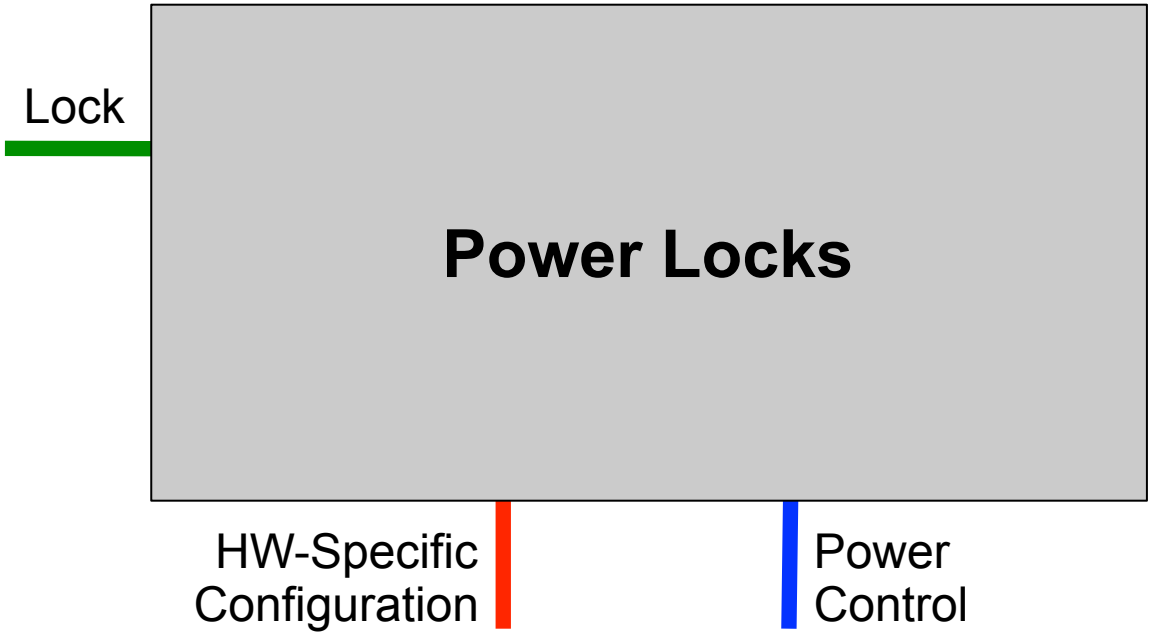
Power Locks

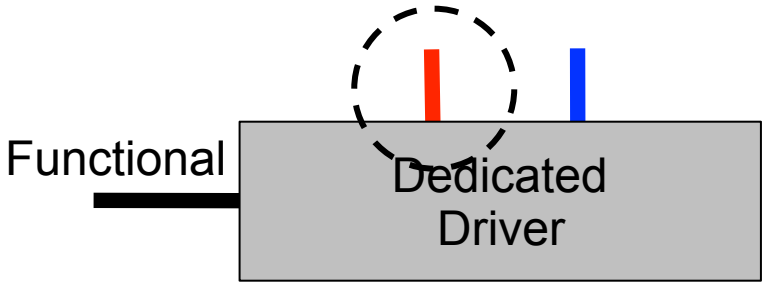

Lock
Power Locks
HW-Specific Configuration
Power Control
Functional
Dedicated Driver

Power Locks

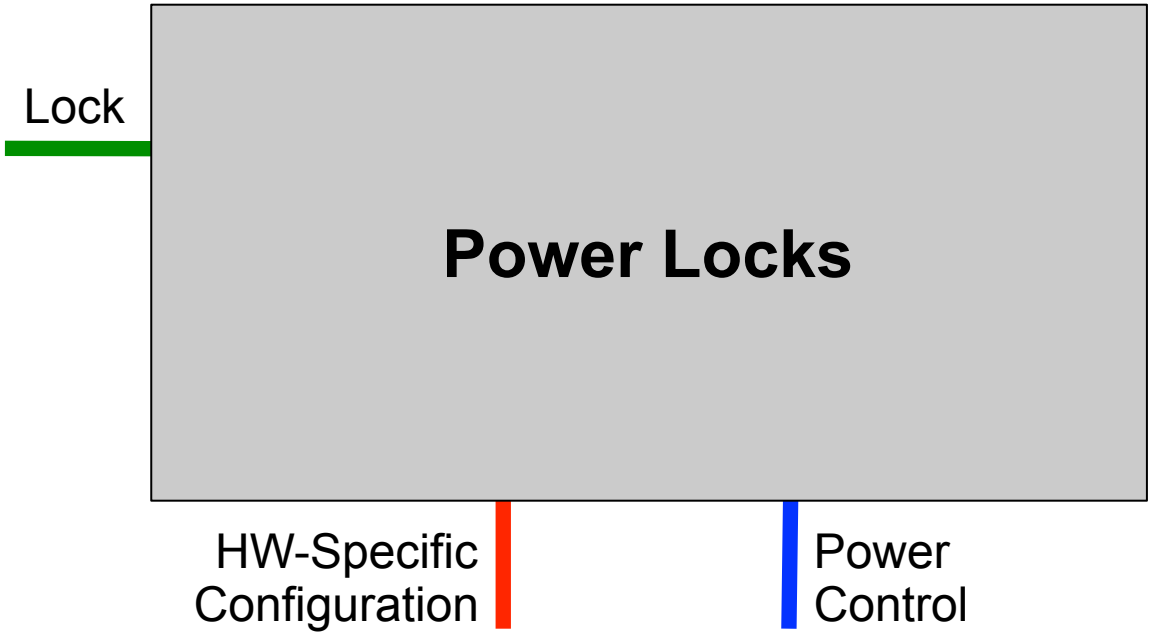

Lock
Power Locks
HW-Specific Configuration
Power Control
Functional
Dedicated Driver

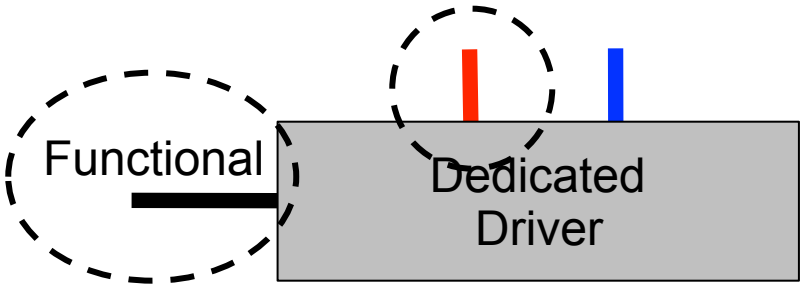
Power Locks


Lock
Power Locks
HW-Specific Configuration
Power Control

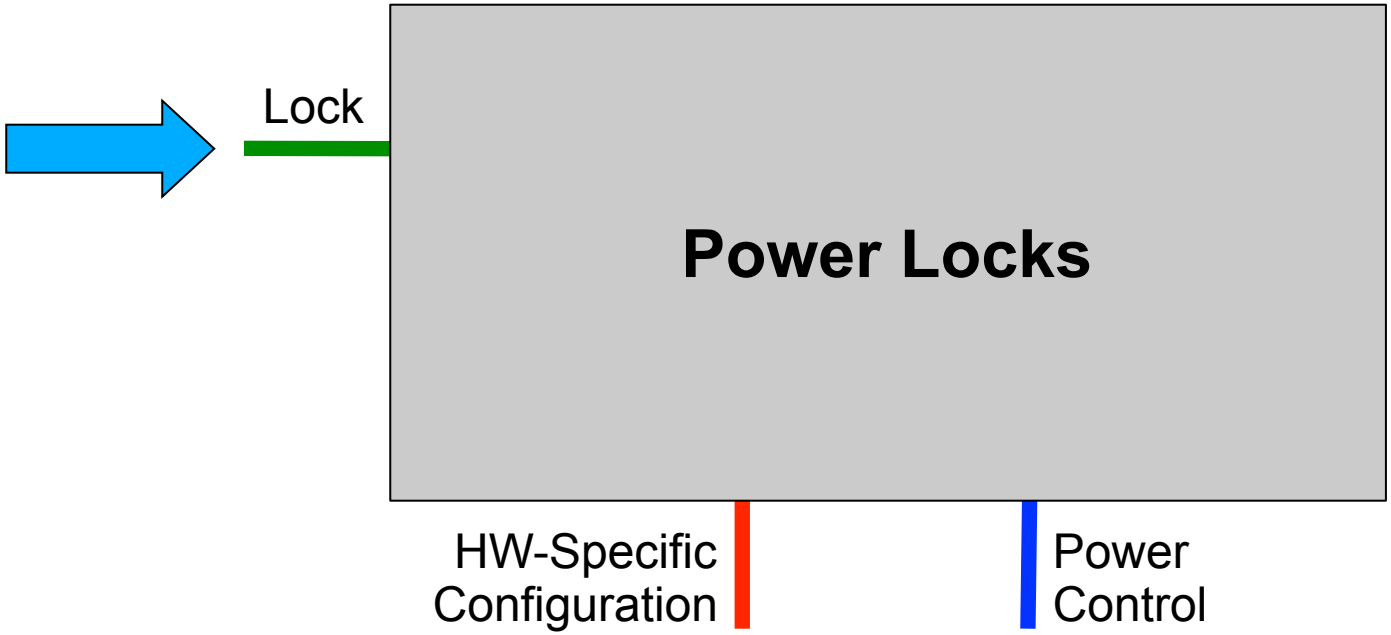

Functional
Dedicated Driver

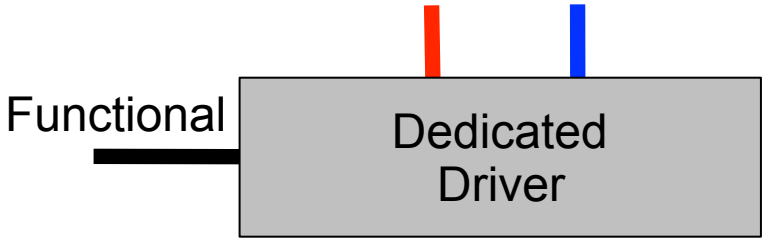
Power Locks


Lock
Power Locks
HW-Specific Configuration
Power Control

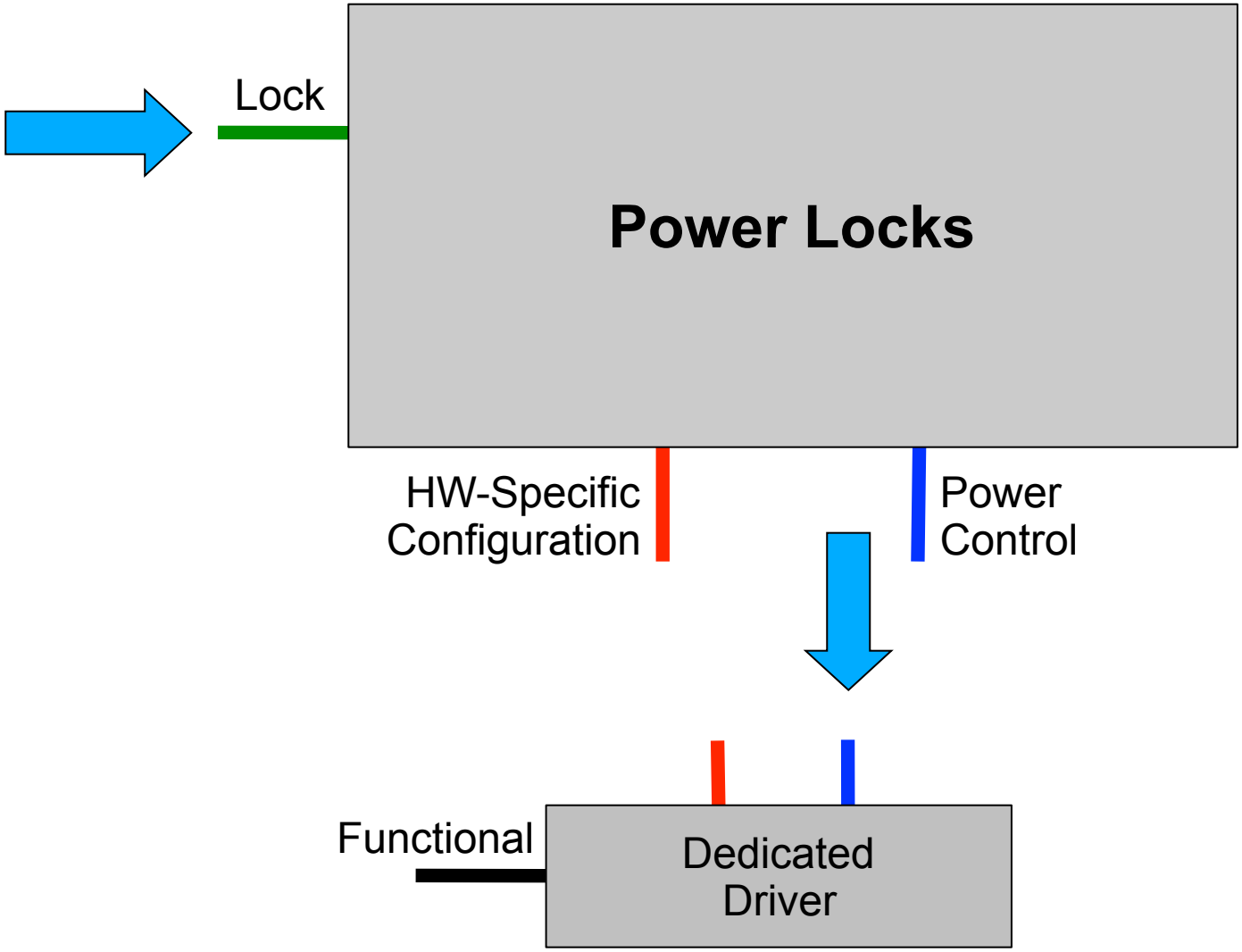

Functional
Dedicated Driver

Power Locks

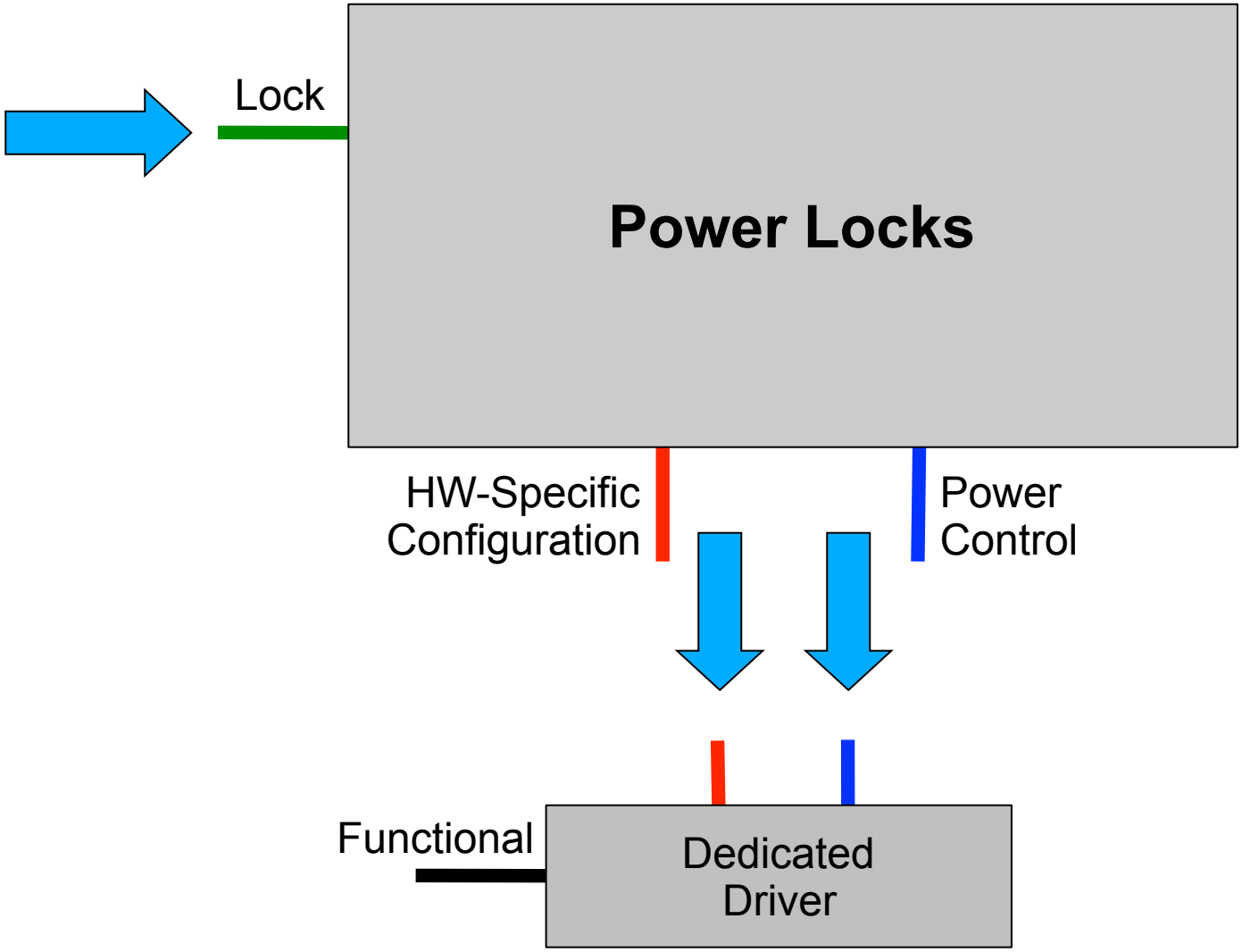

Lock
Power Locks
HW-Specific Configuration
Power Control


Functional
Dedicated Driver

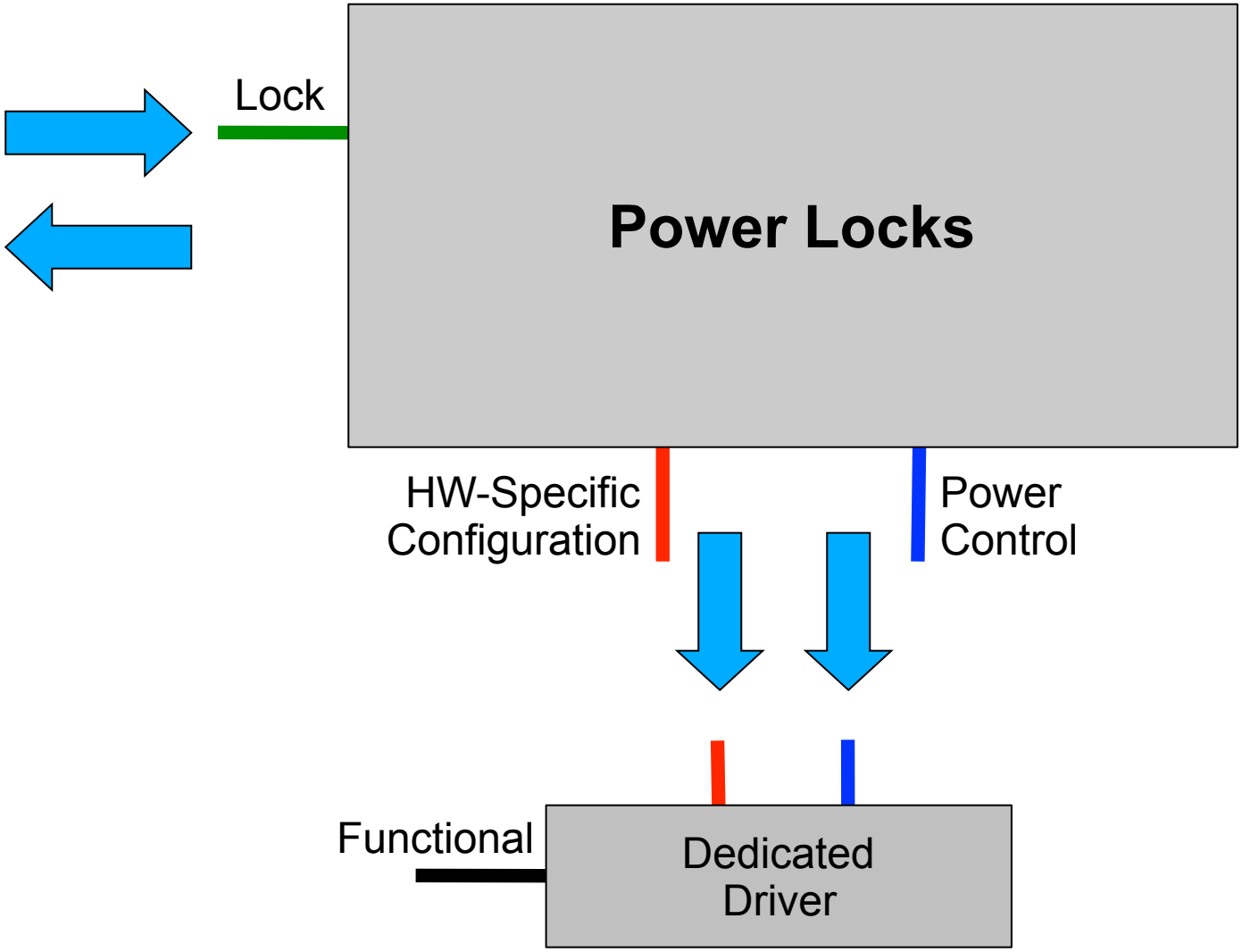
Power Locks


Lock
Power Locks
HW-Specific Configuration
Power Control
Functional
Dedicated Driver

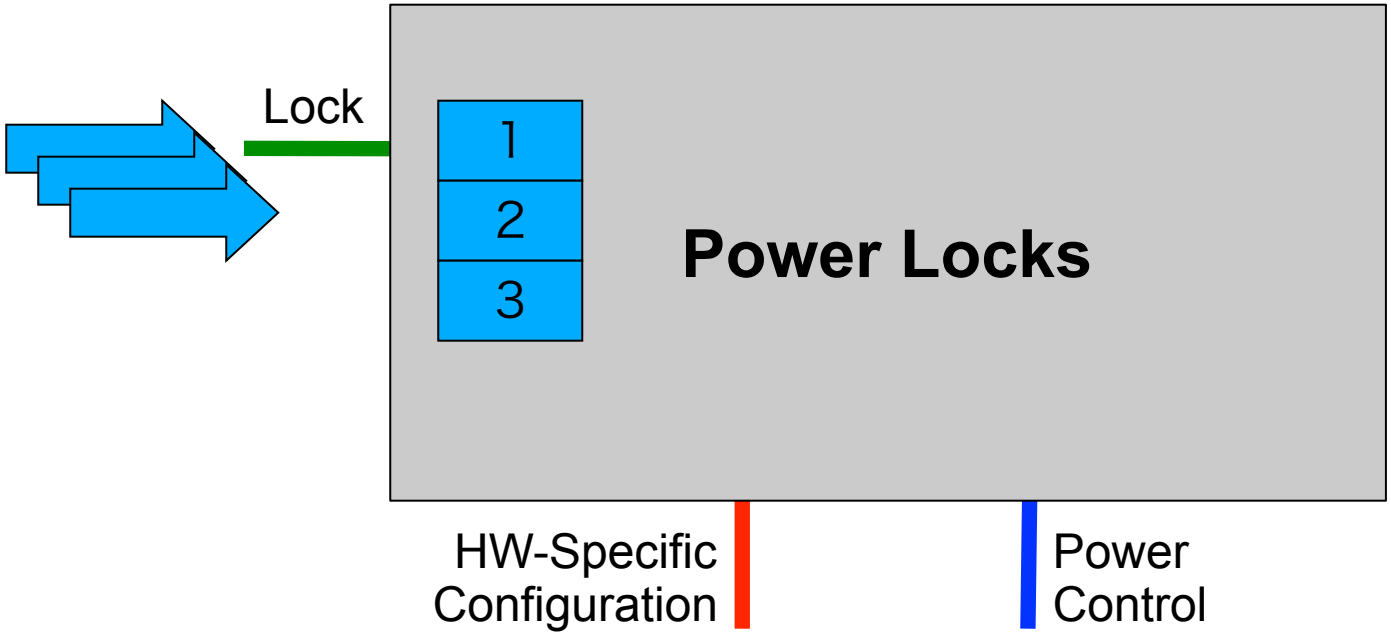
Power Locks

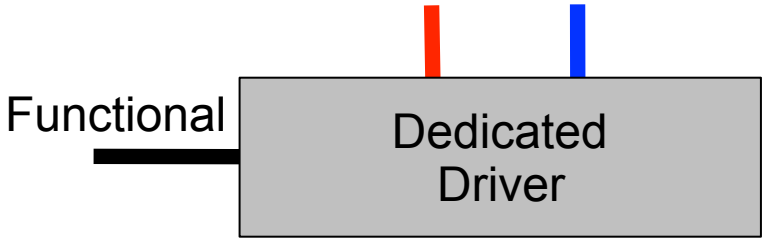

Lock
Power Locks
HW-Specific Configuration
Power Control
Functional
Dedicated Driver

Power Locks

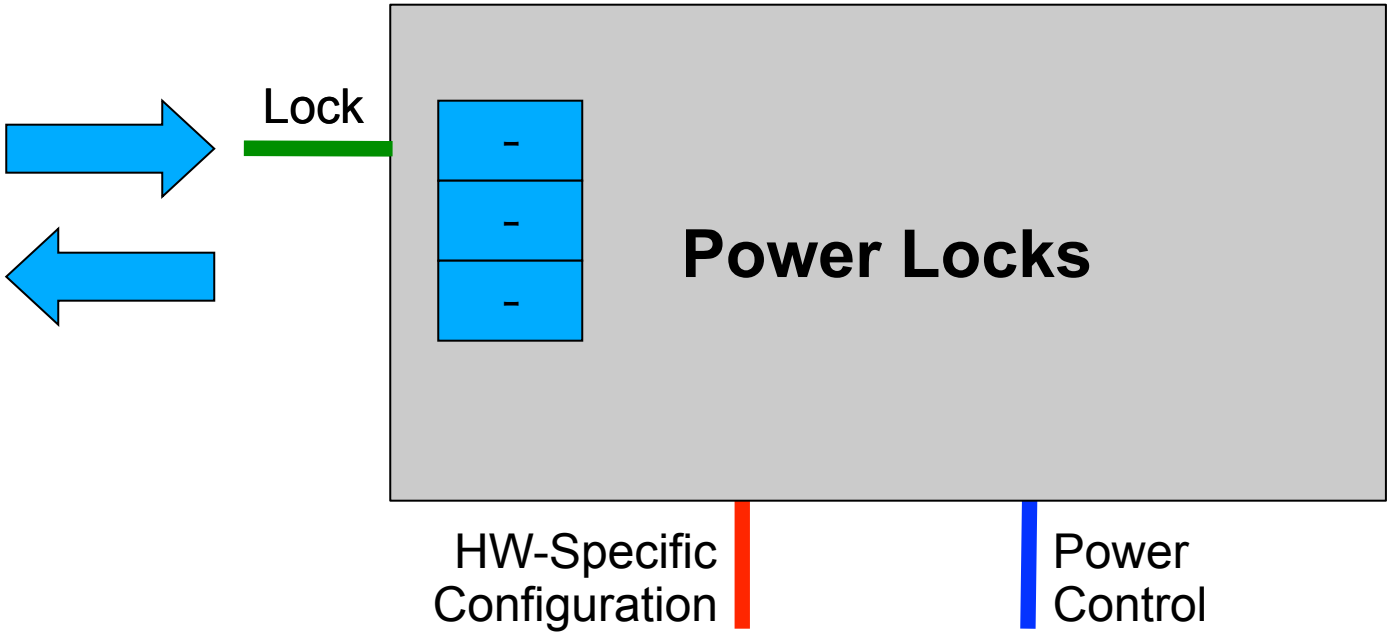

Lock
Power Locks
HW-Specific Configuration
Power Control
Functional
Dedicated Driver

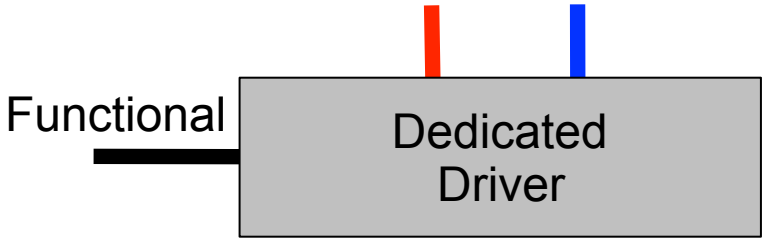
Power Locks


Lock
1
2
3
Power Locks
HW-Specific
Configuration
Power
Control

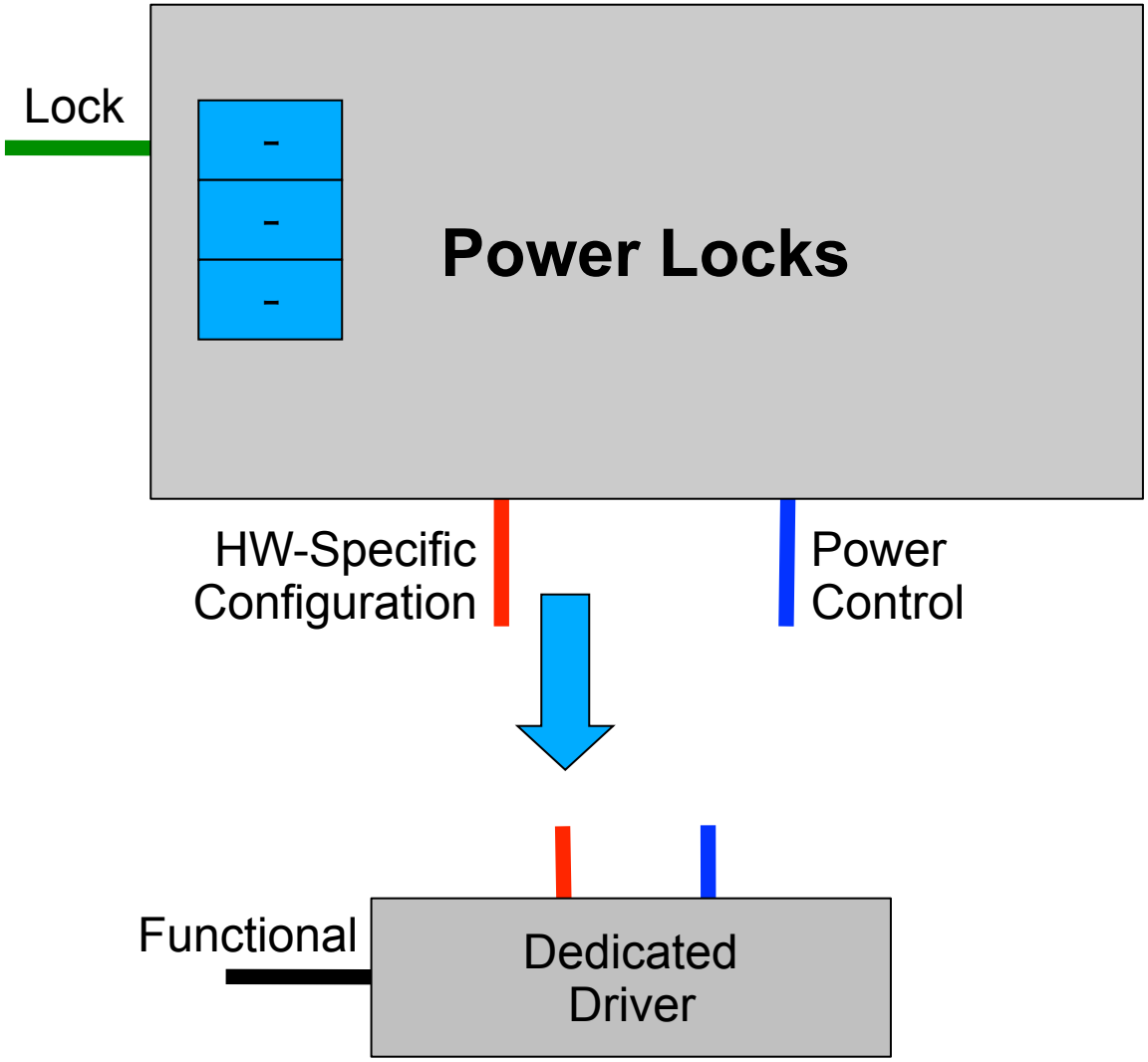

Functional
Dedicated
Driver

Power Locks

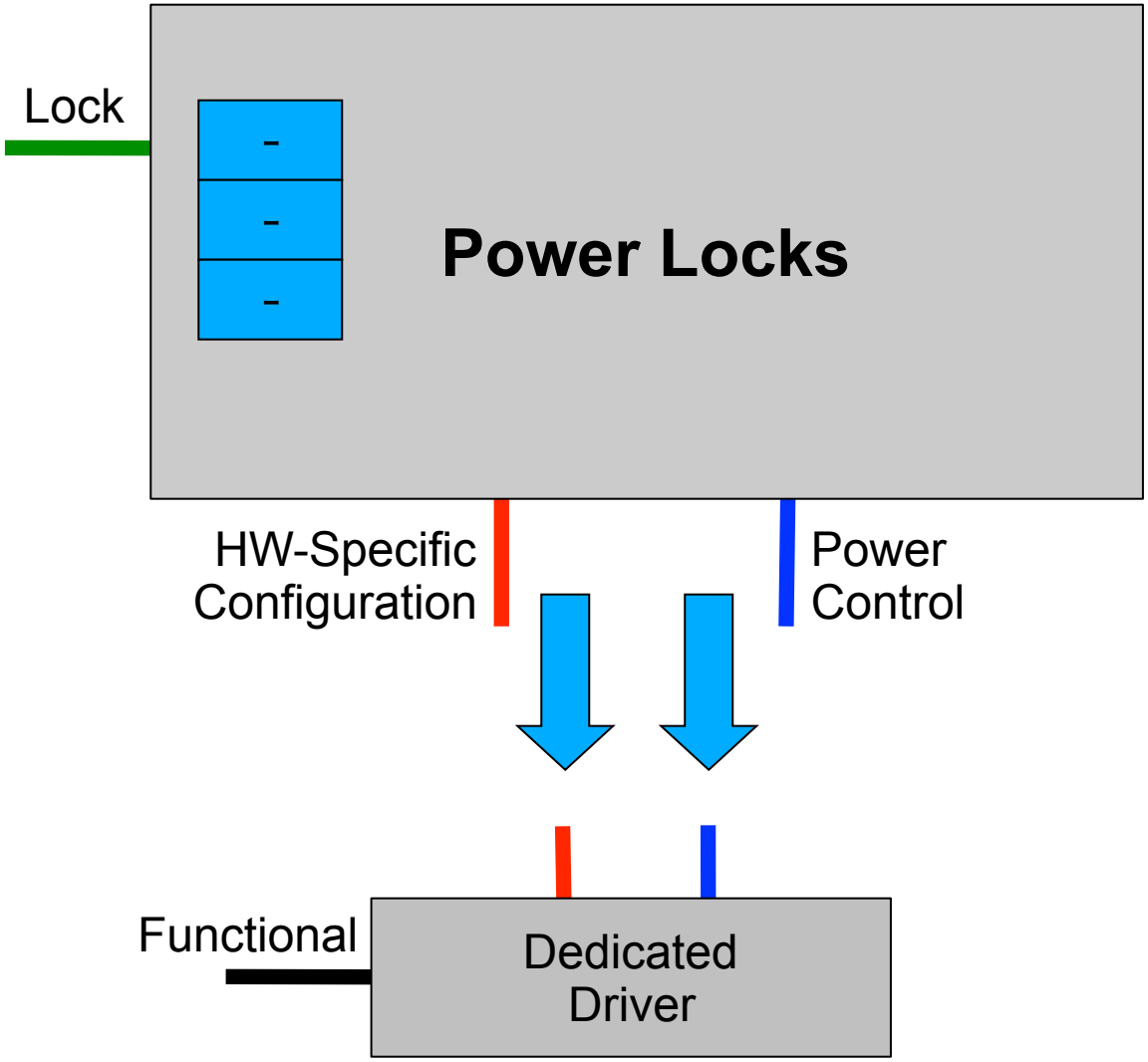

Lock
Power Locks
HW-Specific Configuration
Power Control


Functional
Dedicated Driver

Power Locks


Lock
Power Locks
HW-Specific Configuration
Power Control
Functional
Dedicated Driver

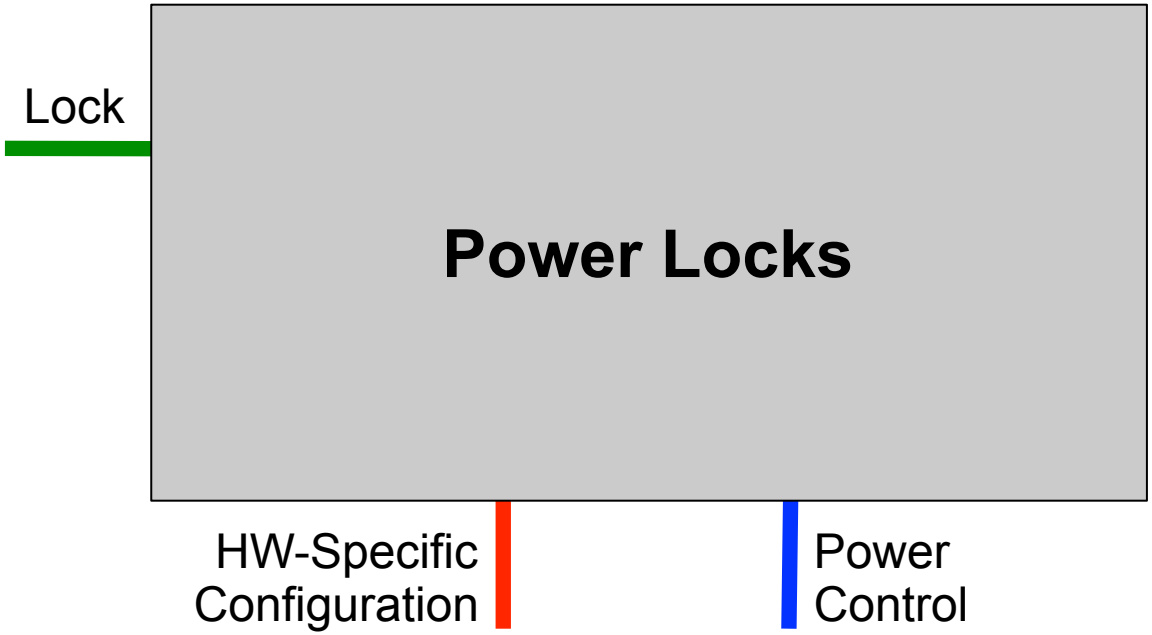
Power Locks


Lock
Power Locks
HW-Specific Configuration
Power Control
Functional
Dedicated Driver

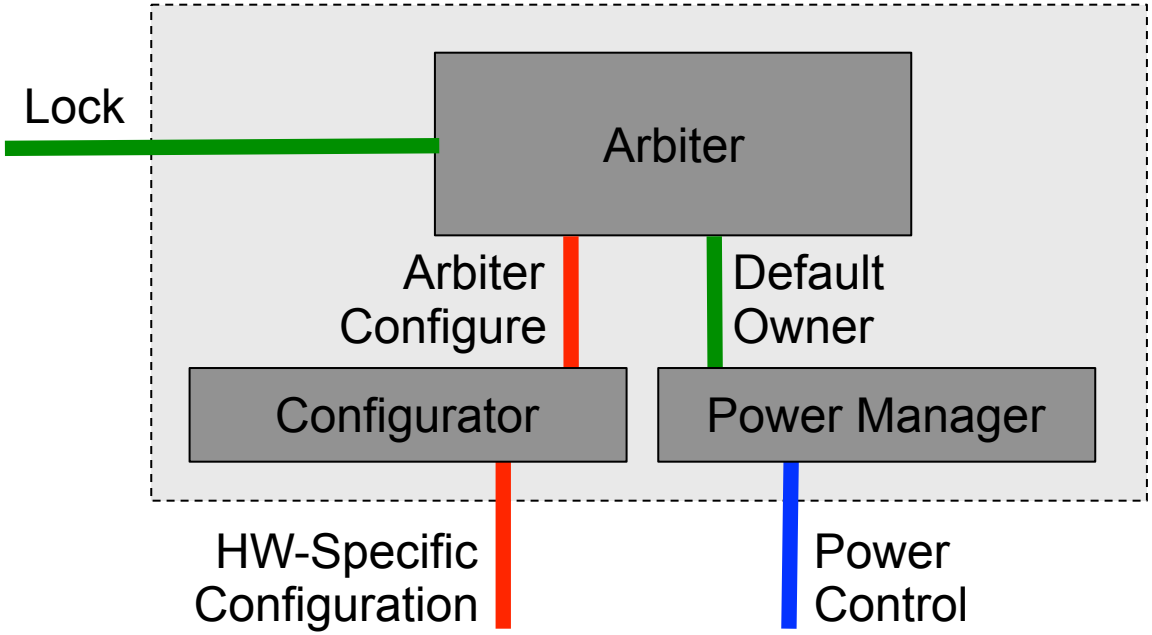
ICEM Architecture

- Defines three classes of drivers
  - Virtualized – provide only functional interface
  - Dedicated – provide functional and power interface
  - Shared – provide functional and lock interface
- *Power Locks*, split-phase locks with integrated energy and configuration management
- Component library
  - Arbiters – manage I/O concurrency
  - Configurators – setup device specific configurations
  - Power Managers – provide automatic power management

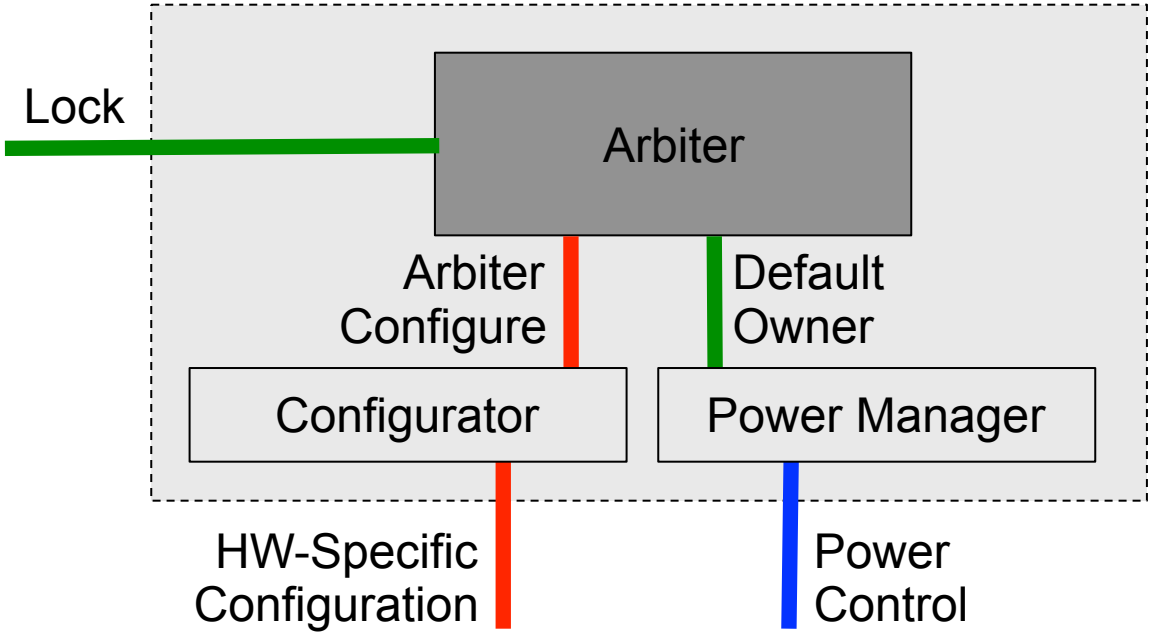
Component Library



Component Library

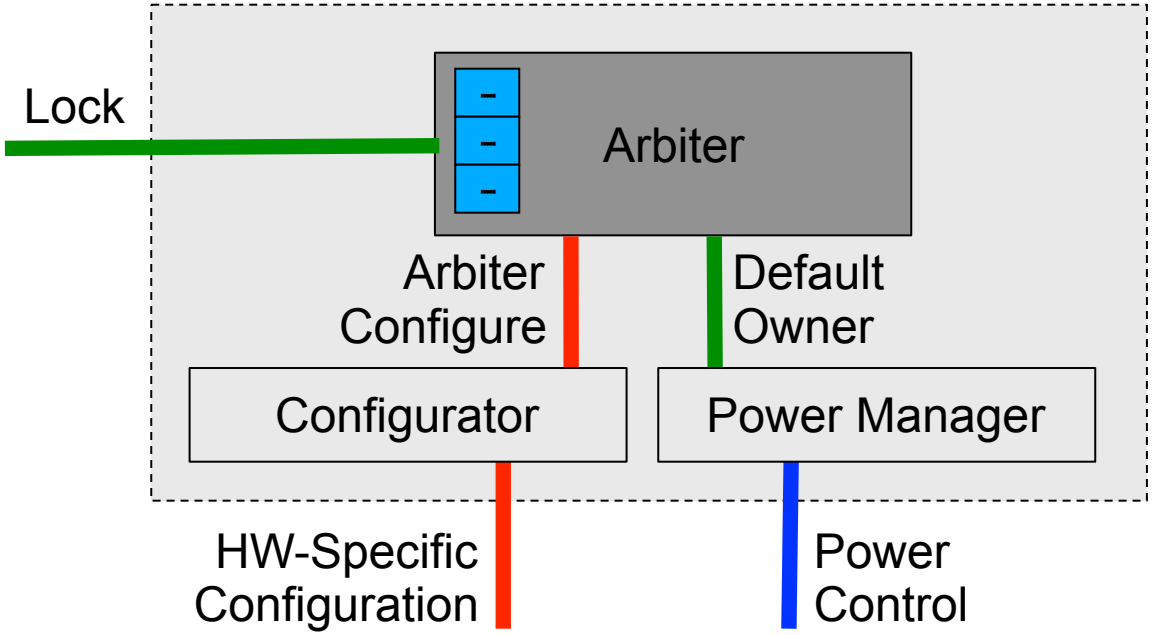


Component Library



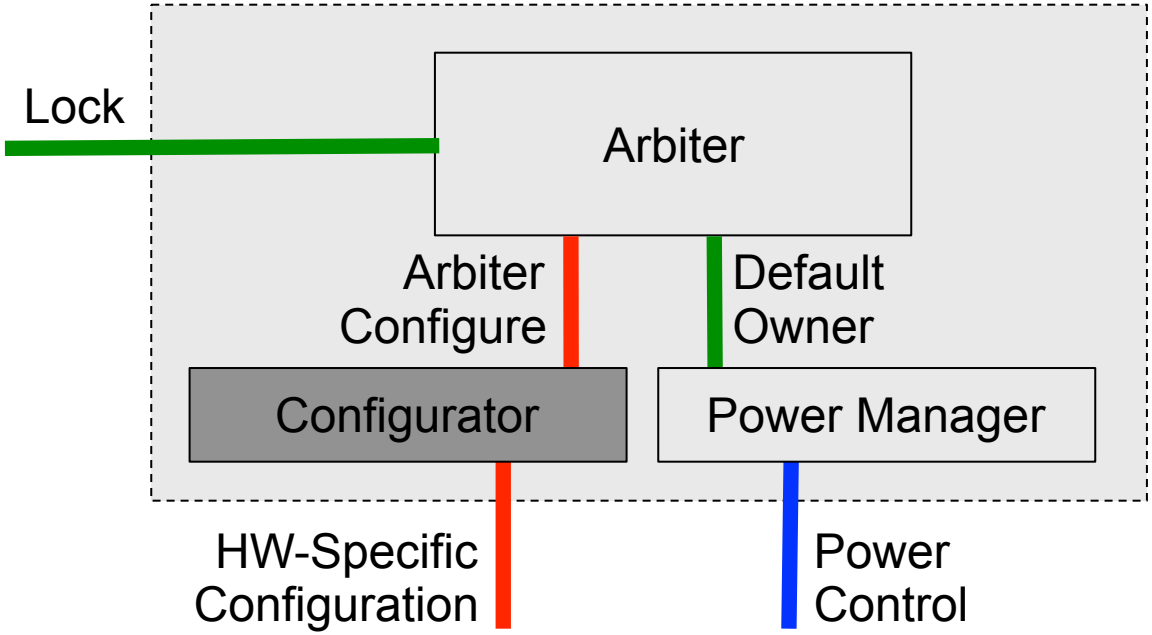
- **Lock** interface for concurrency control (FCFS, Round-Robin)
- **ArbiterConfigure** interface automatic hardware configuration
- **DefaultOwner** interface for automatic power management

Component Library



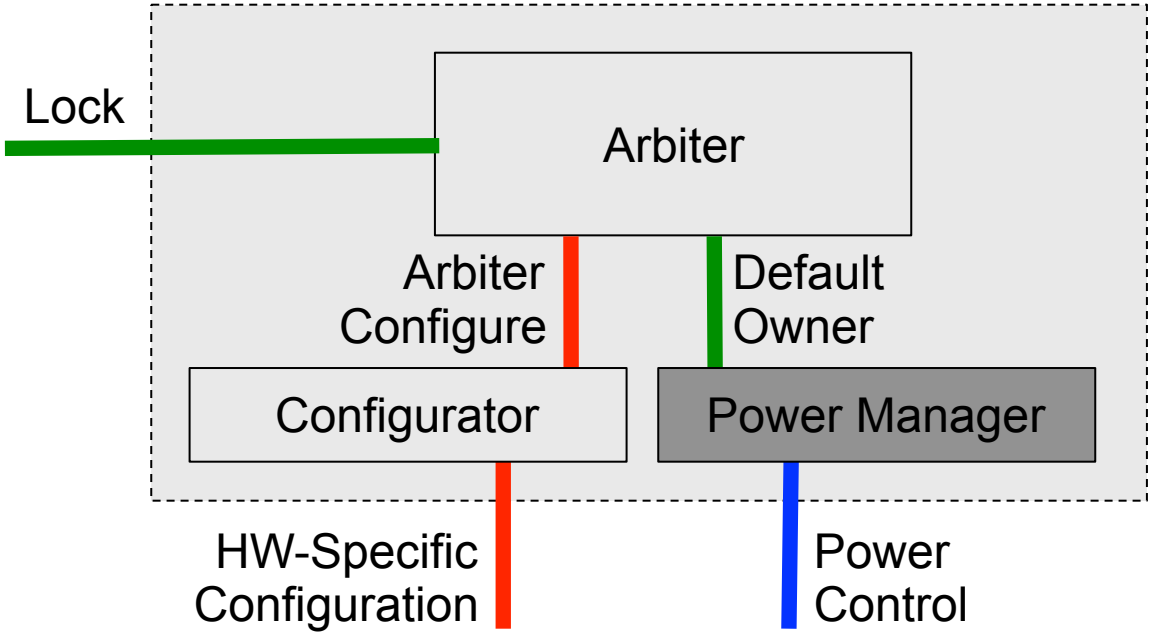
- **Lock** interface for concurrency control (FCFS, Round-Robin)
- **ArbiterConfigure** interface automatic hardware configuration
- **DefaultOwner** interface for automatic power management

Component Library



- Implement **ArbiterConfigure** interface
- Call hardware specific configuration from dedicated driver

Component Library



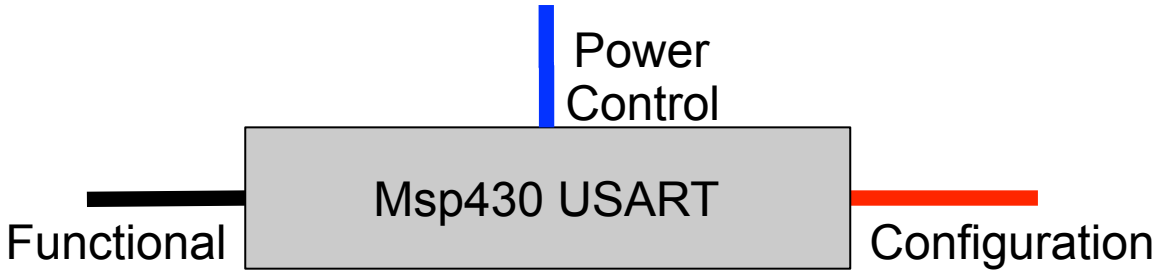
- Implement **DefaultOwner** interface
- Power down device when device falls idle
- Power up device when new lock request comes in
- Currently provide *Immediate* and *Deferred* policies

Shared Driver Example

- Msp430 USART (Serial Controller)
  - Three modes of operation – SPI, I^2C, UART

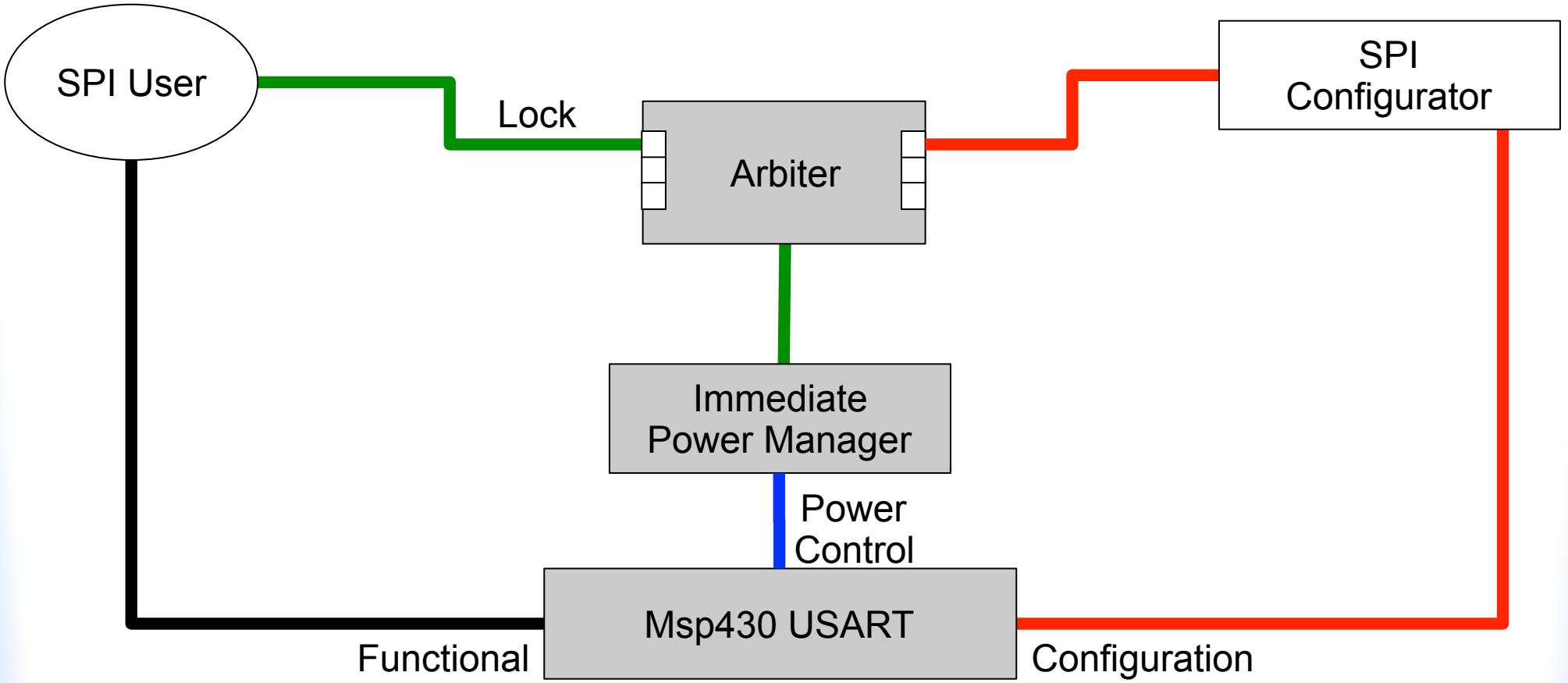
Shared Driver Example

- Msp430 USART (Serial Controller)
  - Three modes of operation – SPI, I^2C, UART



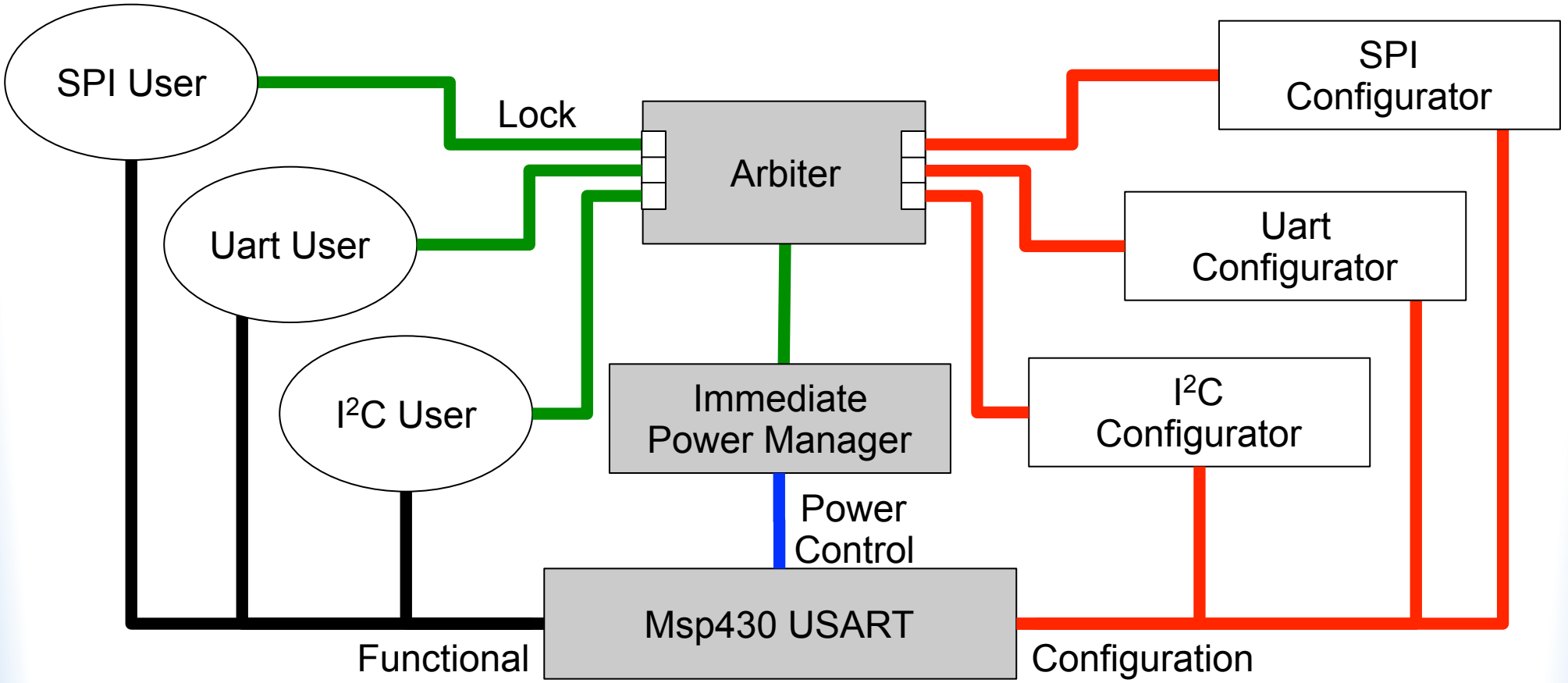
Shared Driver Example

- Msp430 USART (Serial Controller)
  - Three modes of operation – SPI, I^2C, UART



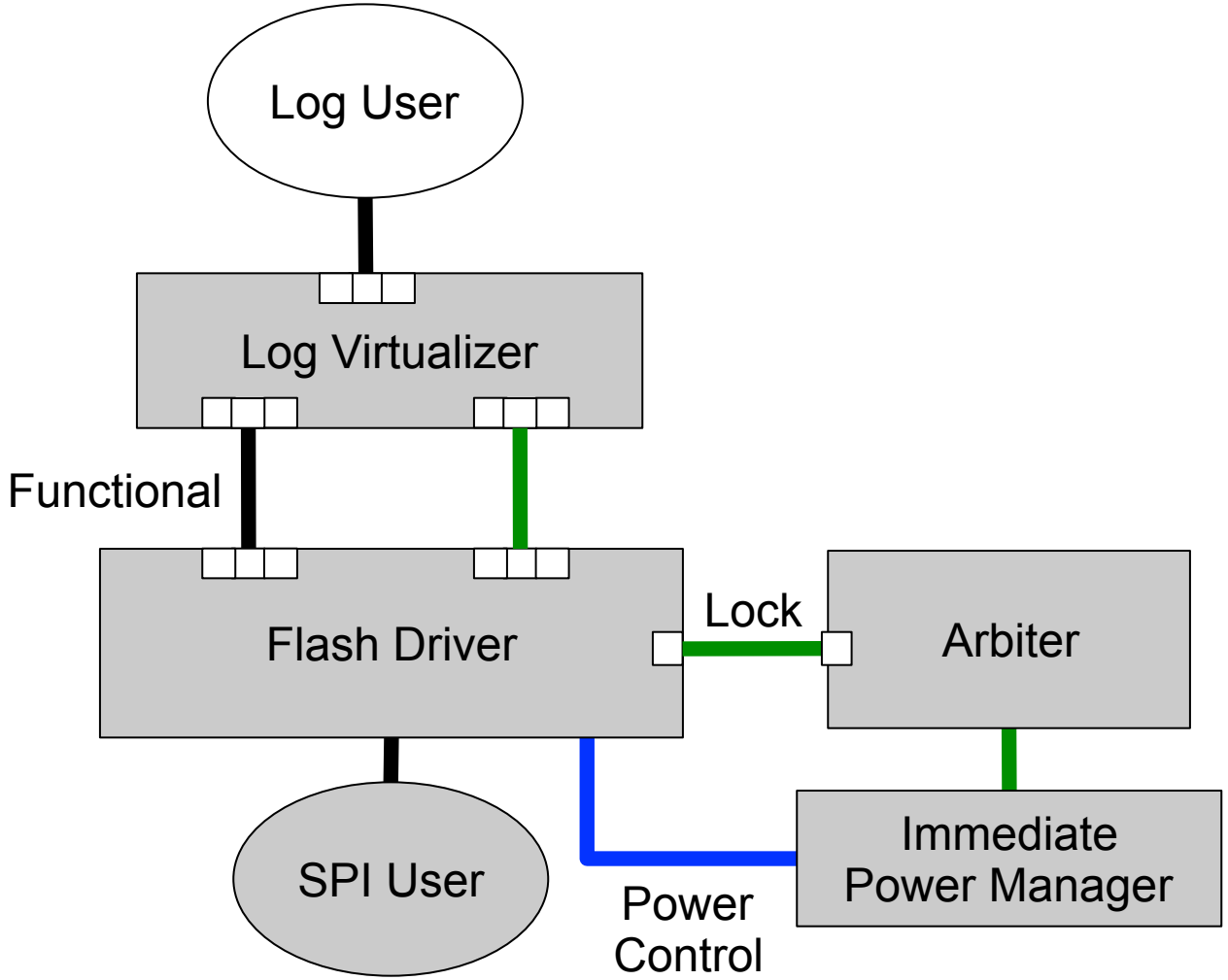
Shared Driver Example

- Msp430 USART (Serial Controller)
  - Three modes of operation – SPI, I²C, UART



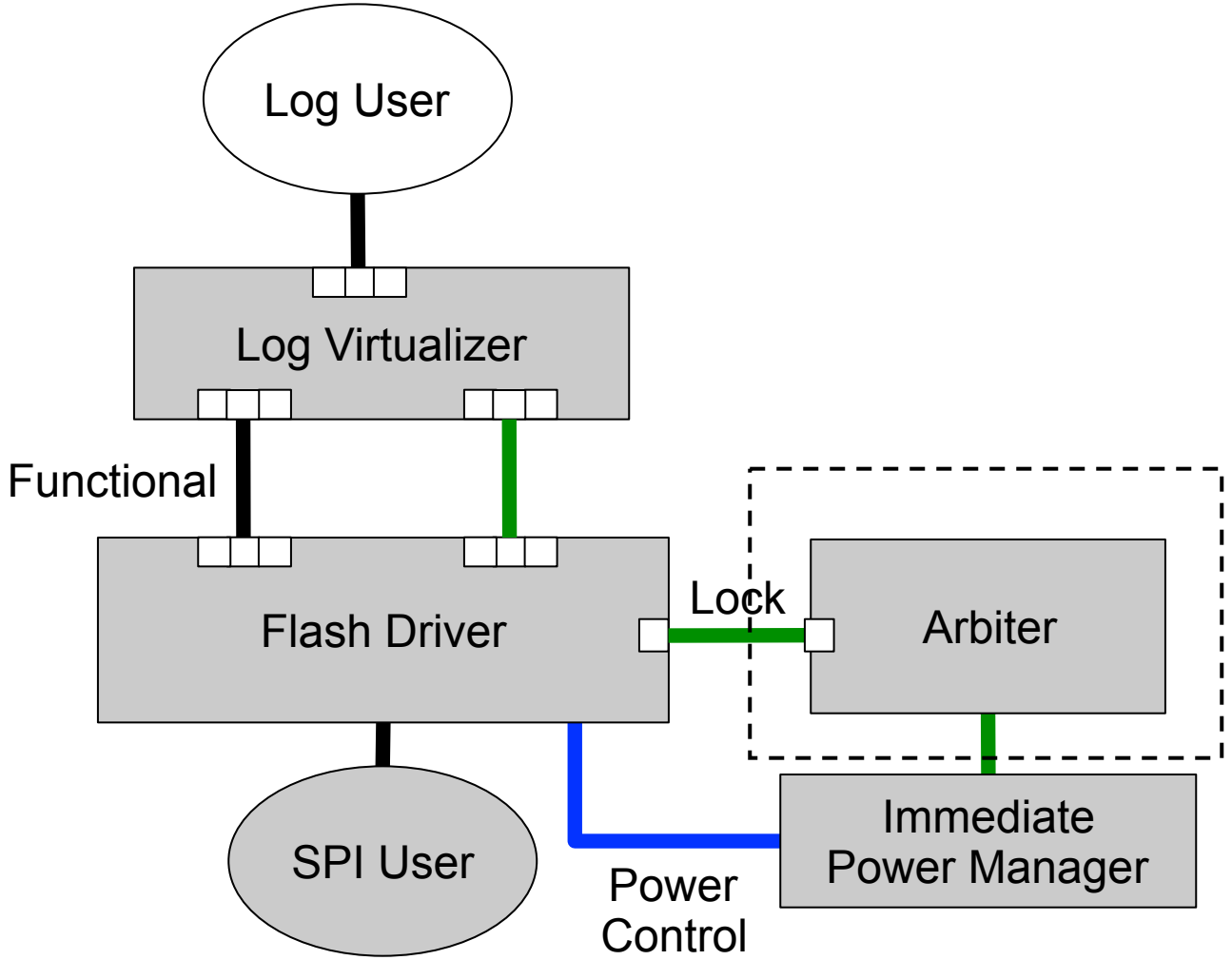
Virtualized Driver Example

- Flash Storage



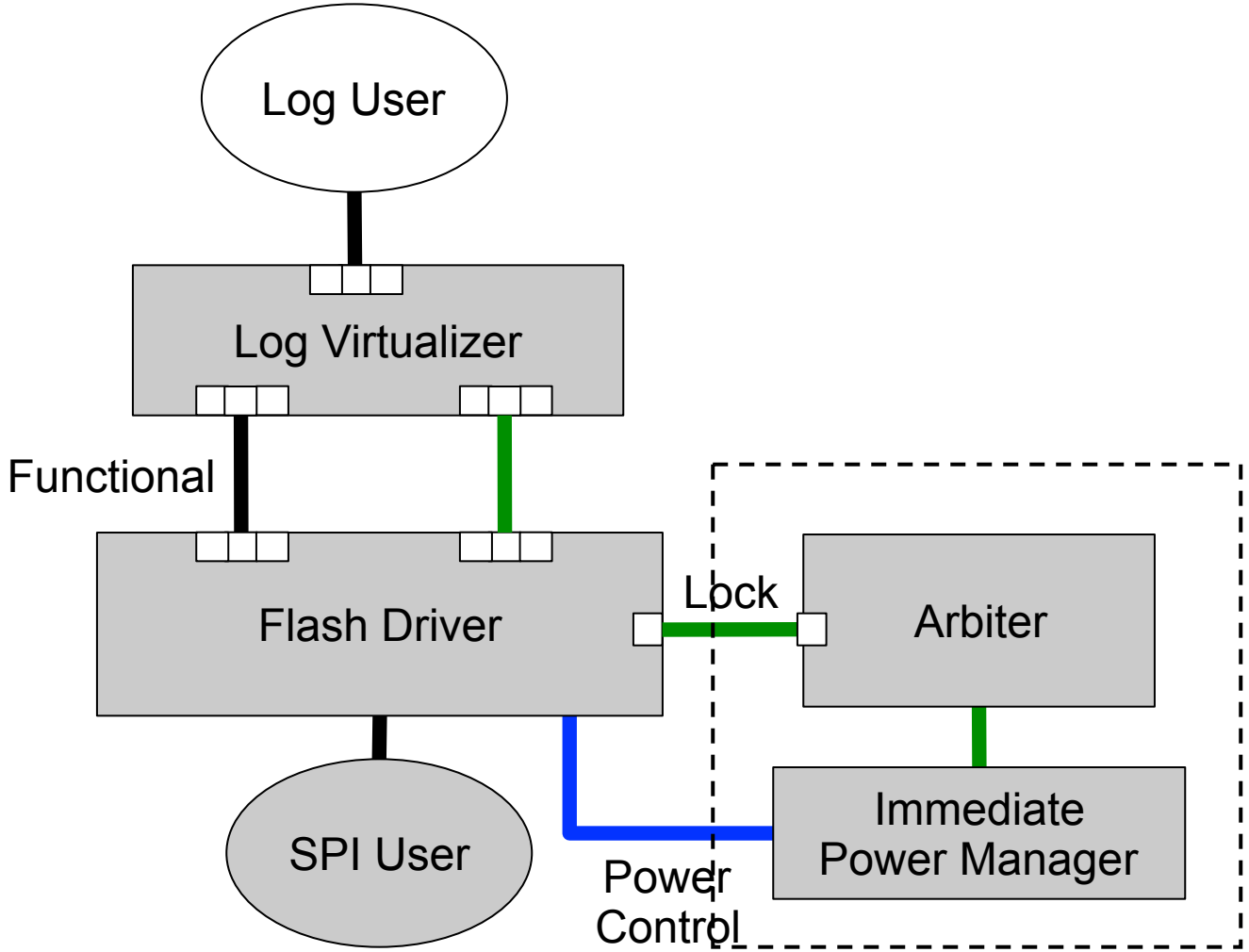
Virtualized Driver Example

- Flash Storage



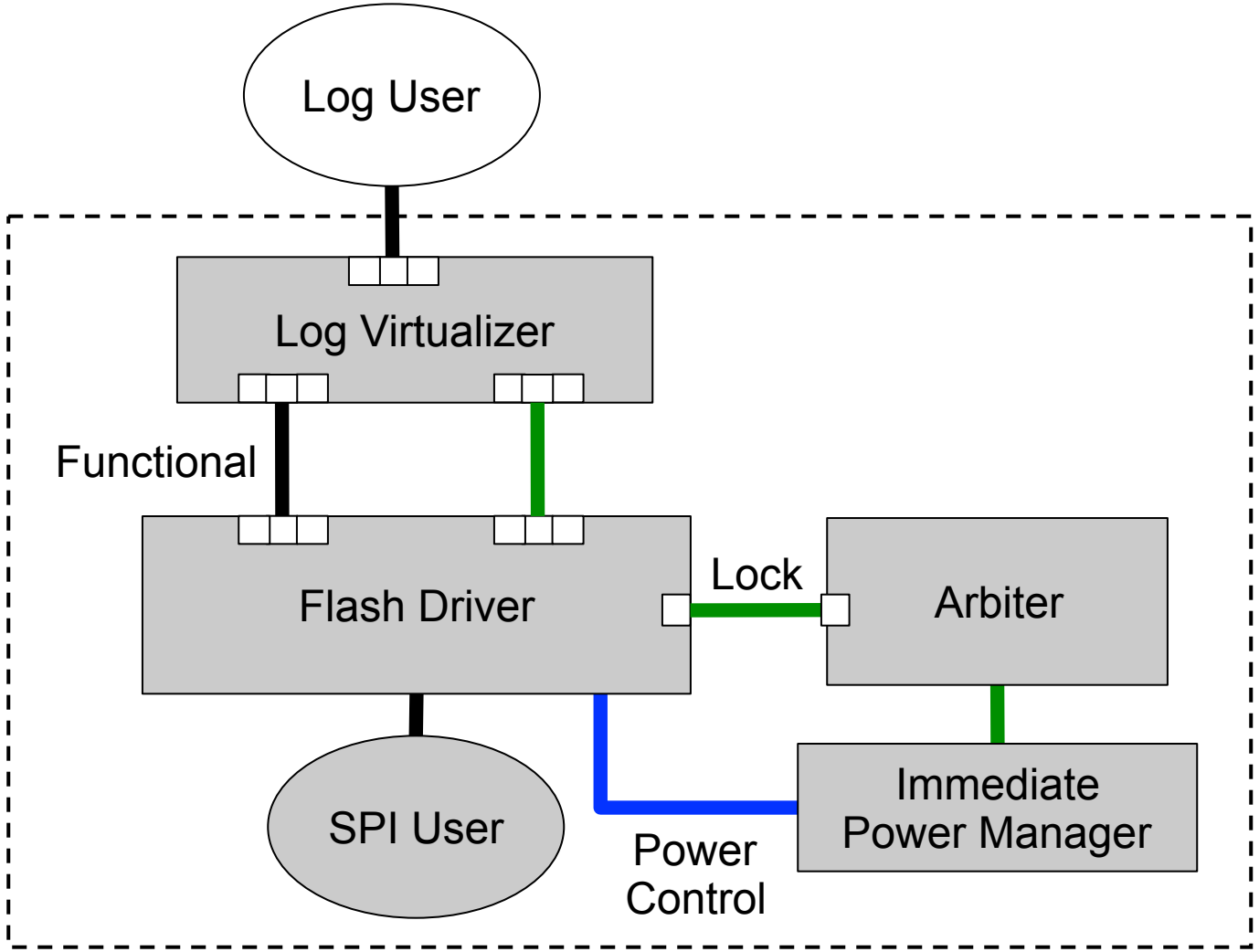
Virtualized Driver Example

- Flash Storage



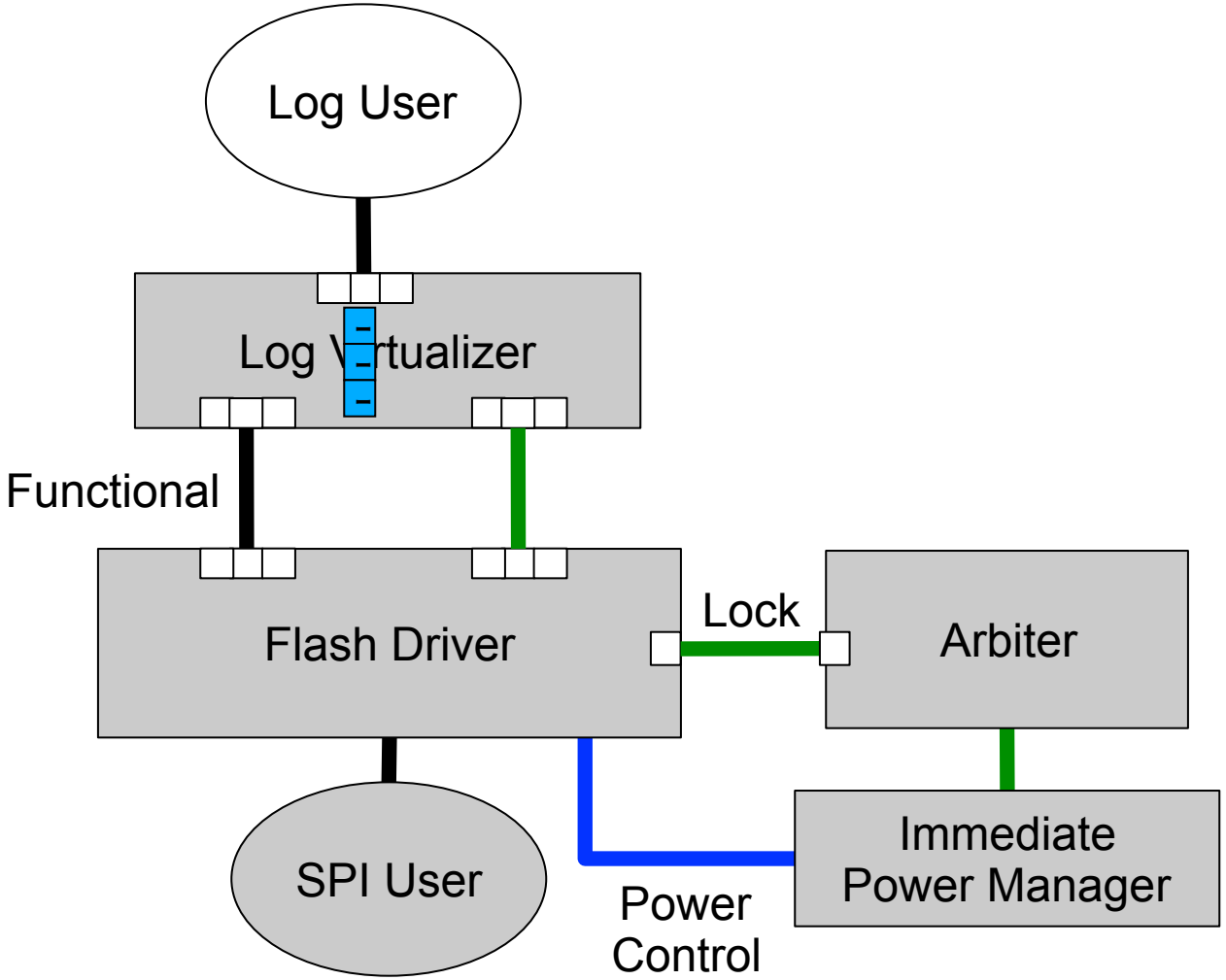
Virtualized Driver Example

- Flash Storage



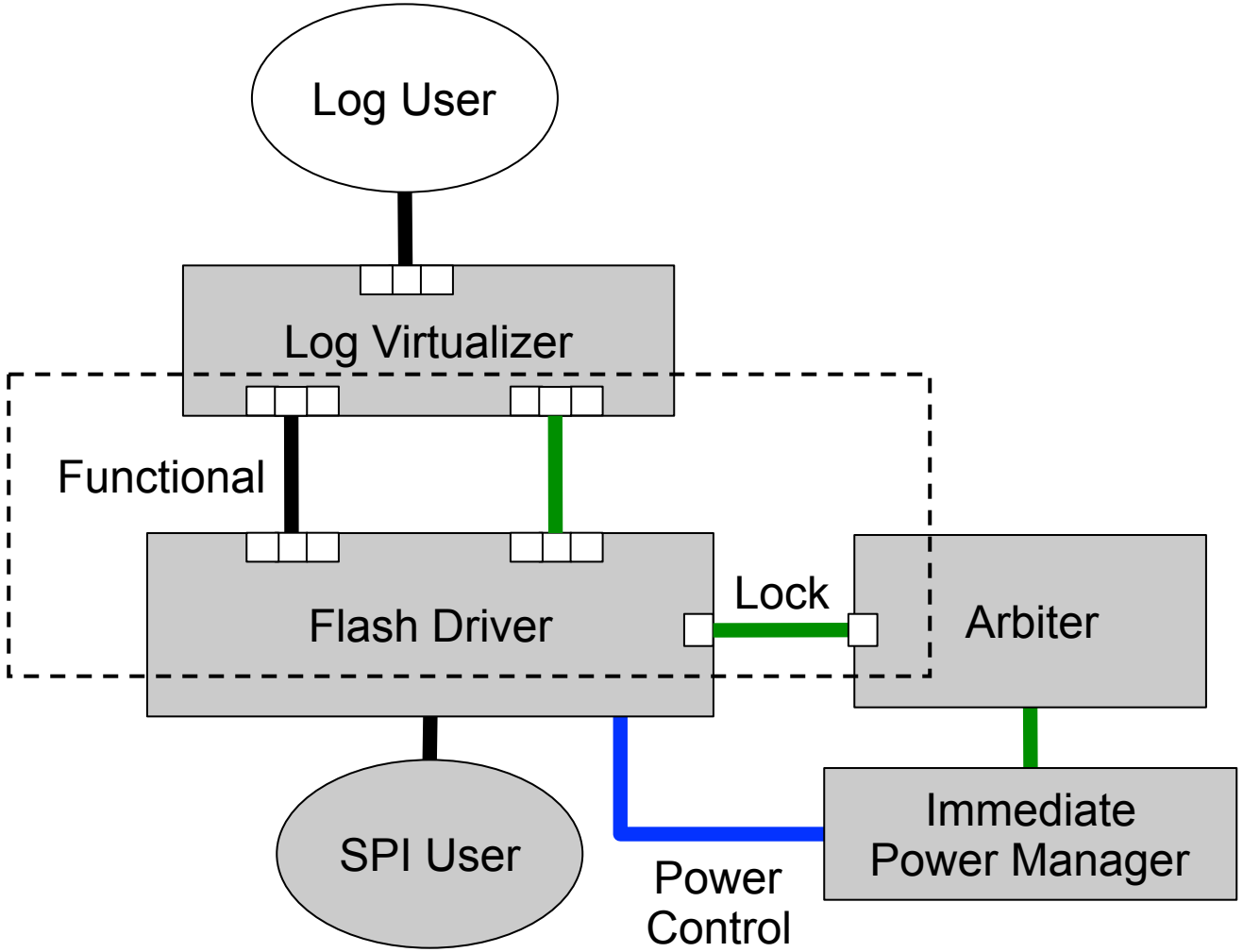
Virtualized Driver Example

- Flash Storage



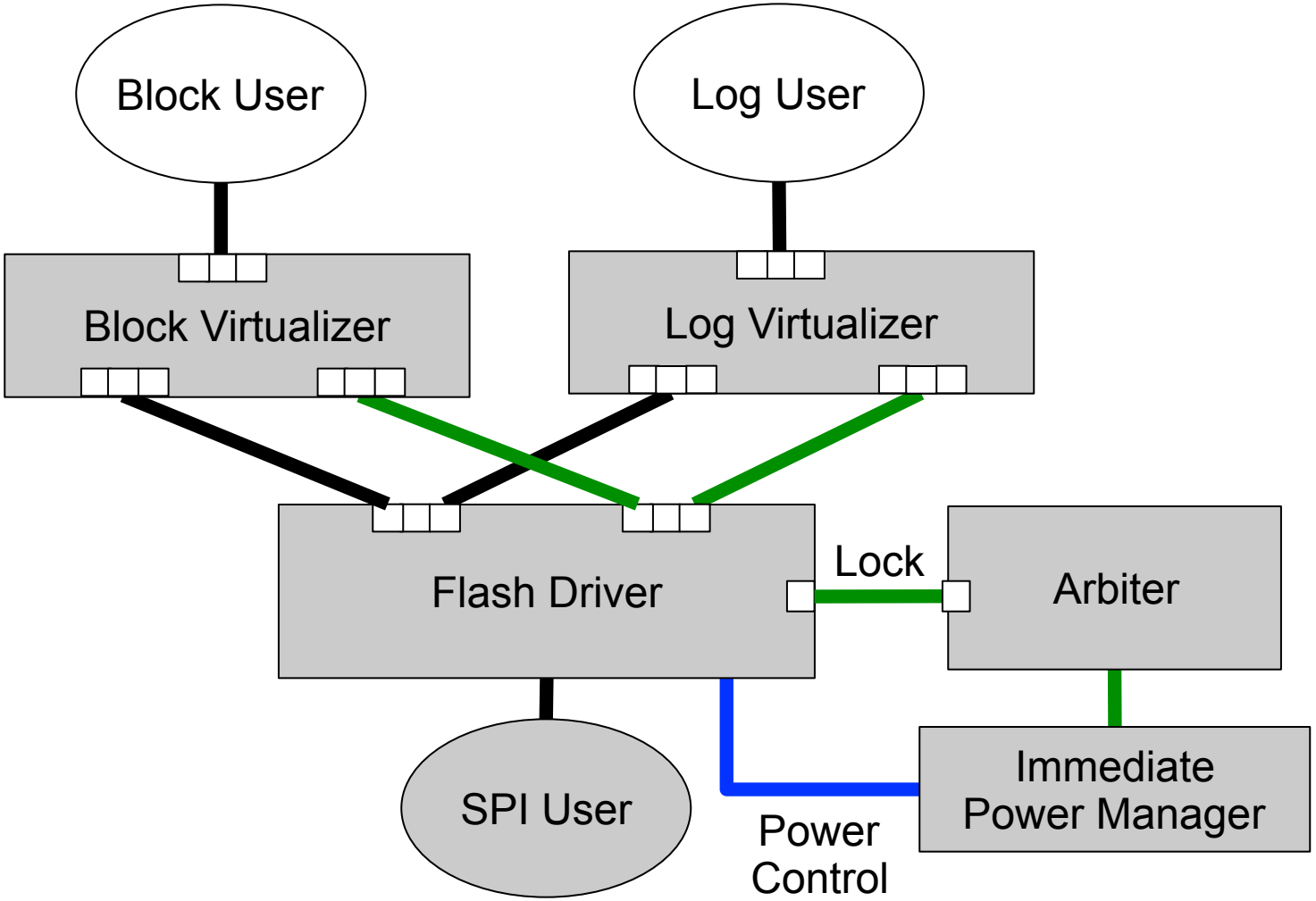
Virtualized Driver Example

- Flash Storage



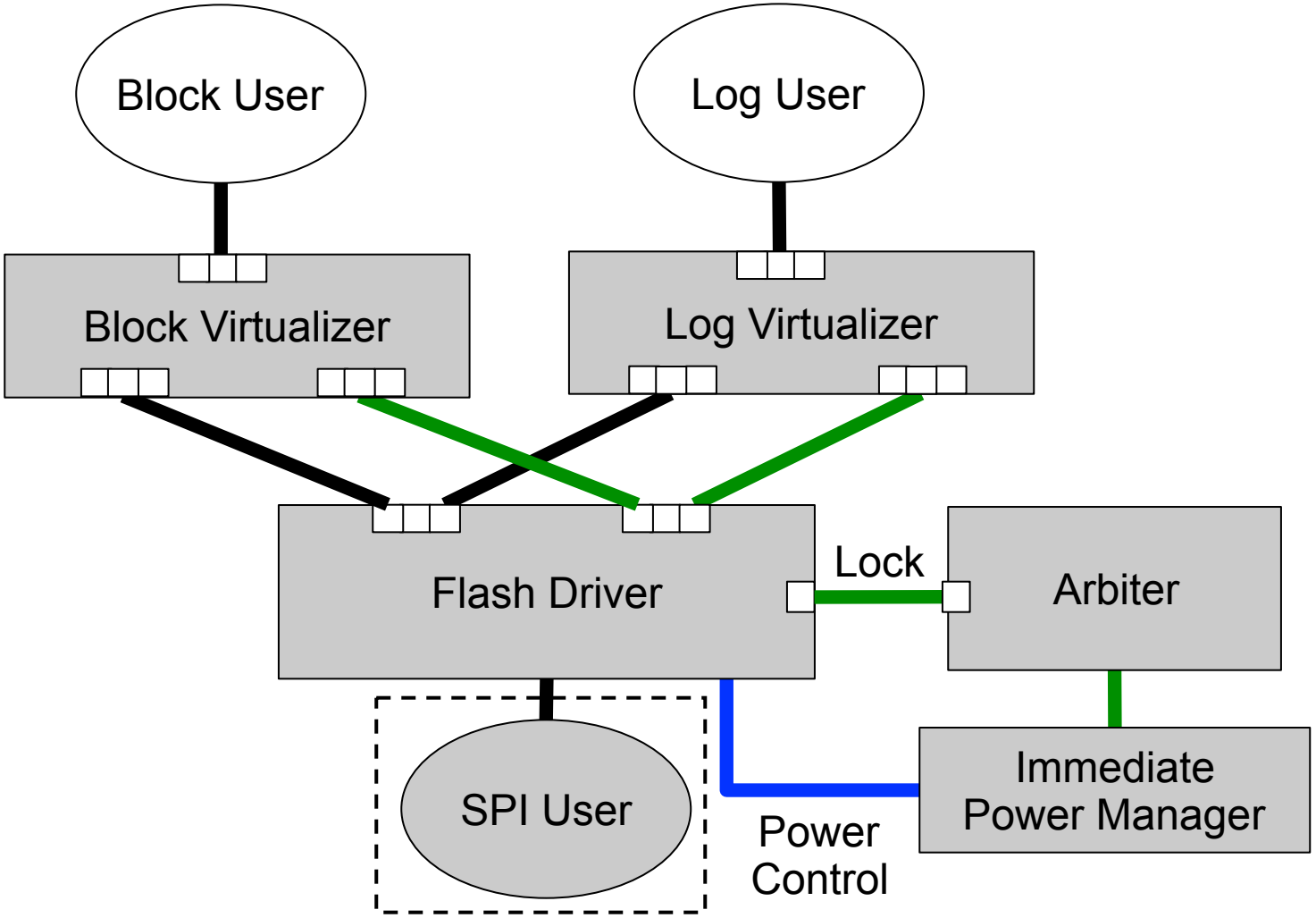
Virtualized Driver Example

- Flash Storage



Virtualized Driver Example

- Flash Storage

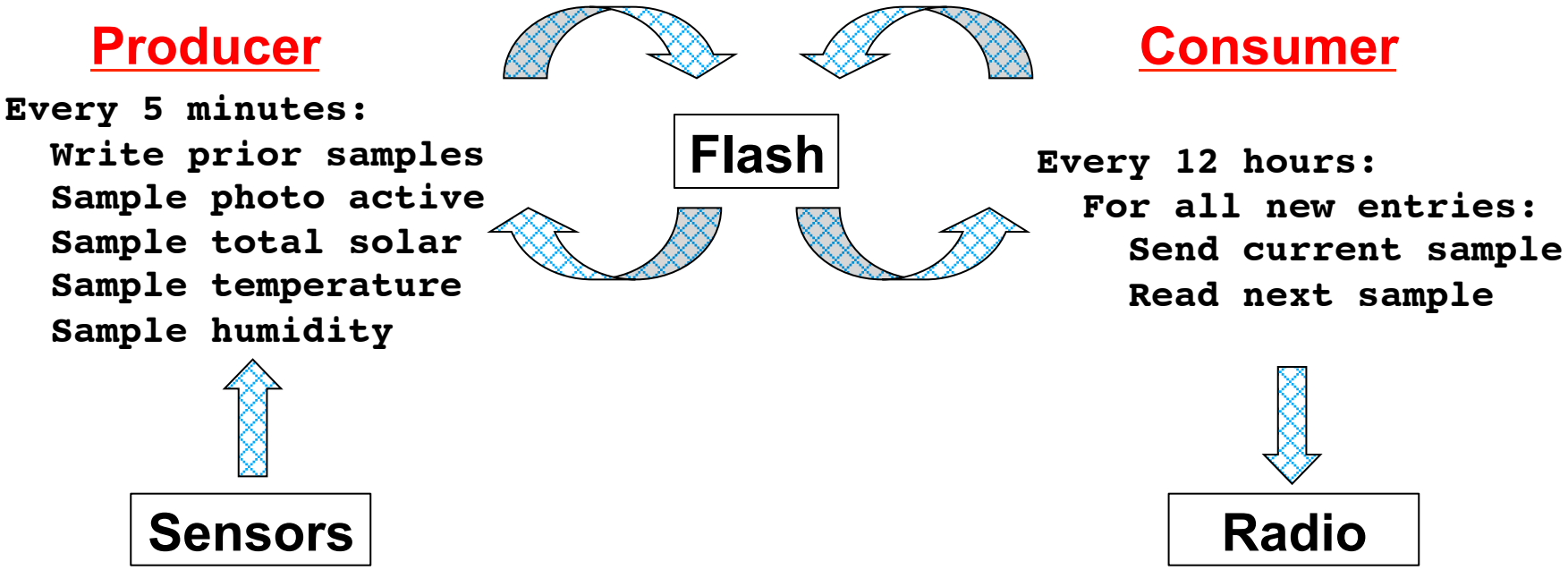


Outline

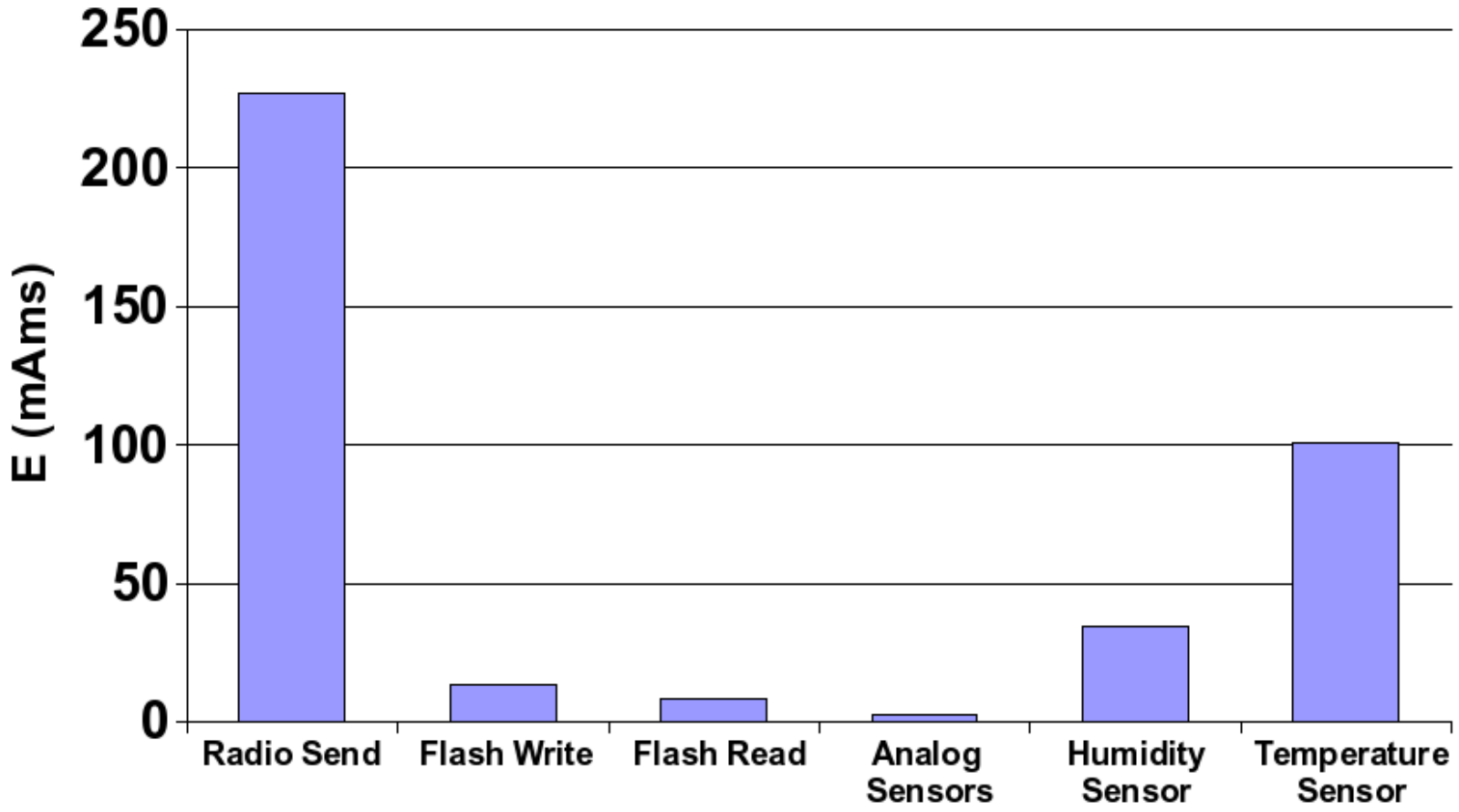
- Introduction and Motivation
- Platform and Application
- ICEM architecture
- **Evaluation**
- Conclusion

Applications

- Hand Tuned – Most energy efficient
- ICEM – All concurrent operations
- Serial + – Optimal serial ordering
- Serial – – Worst case serial ordering

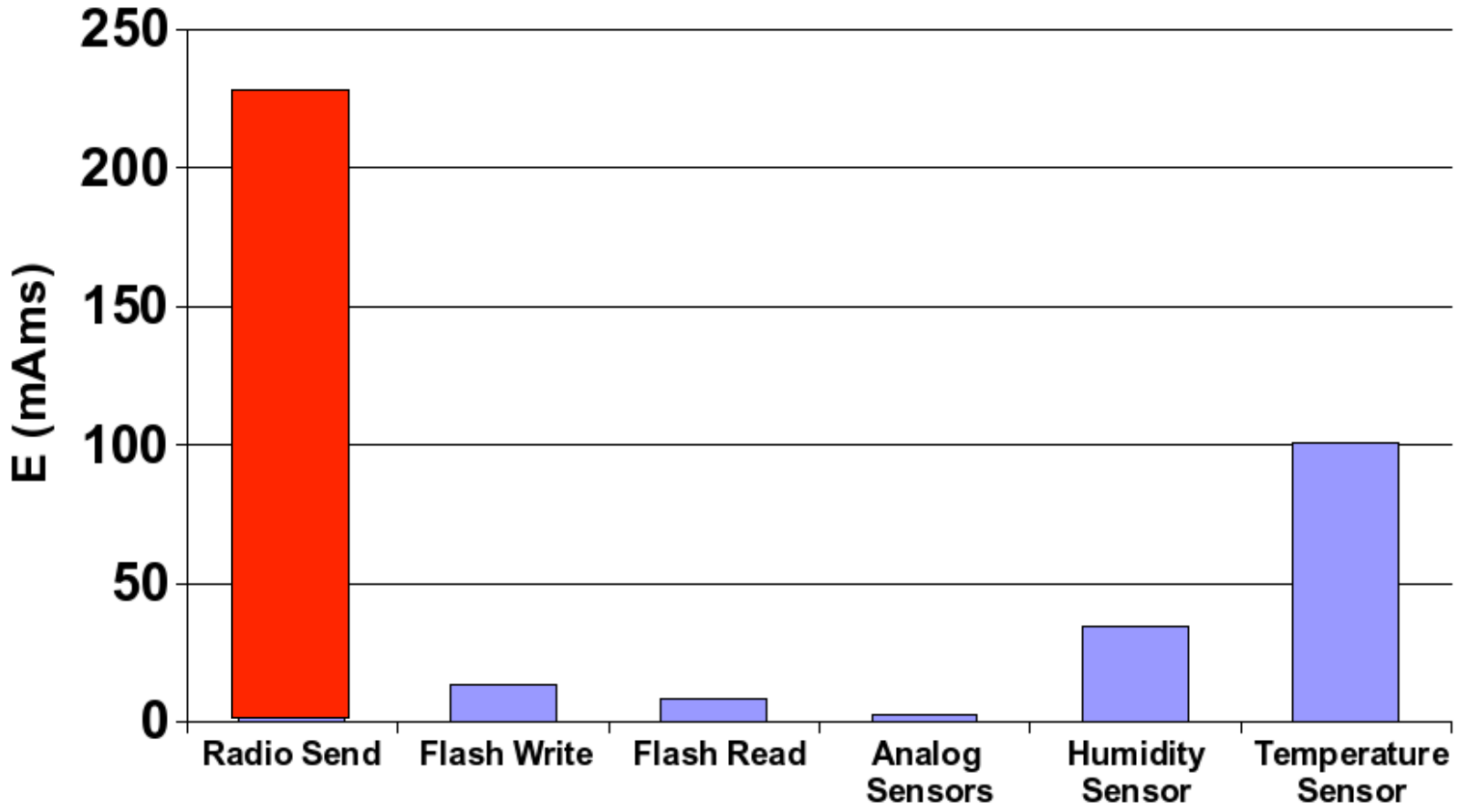


Tmote Energy Consumption



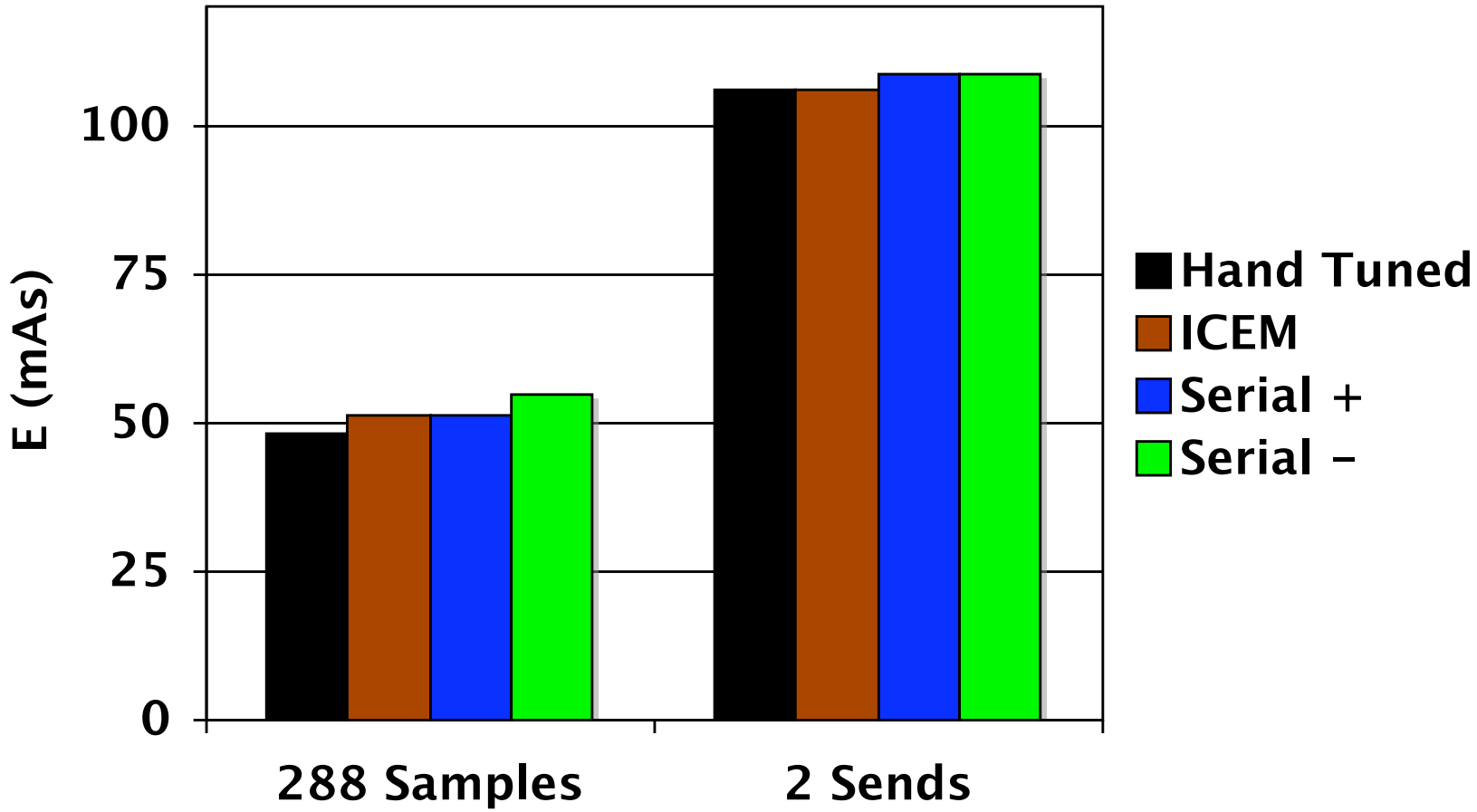
Average energy consumption for application operations

Tmote Energy Consumption



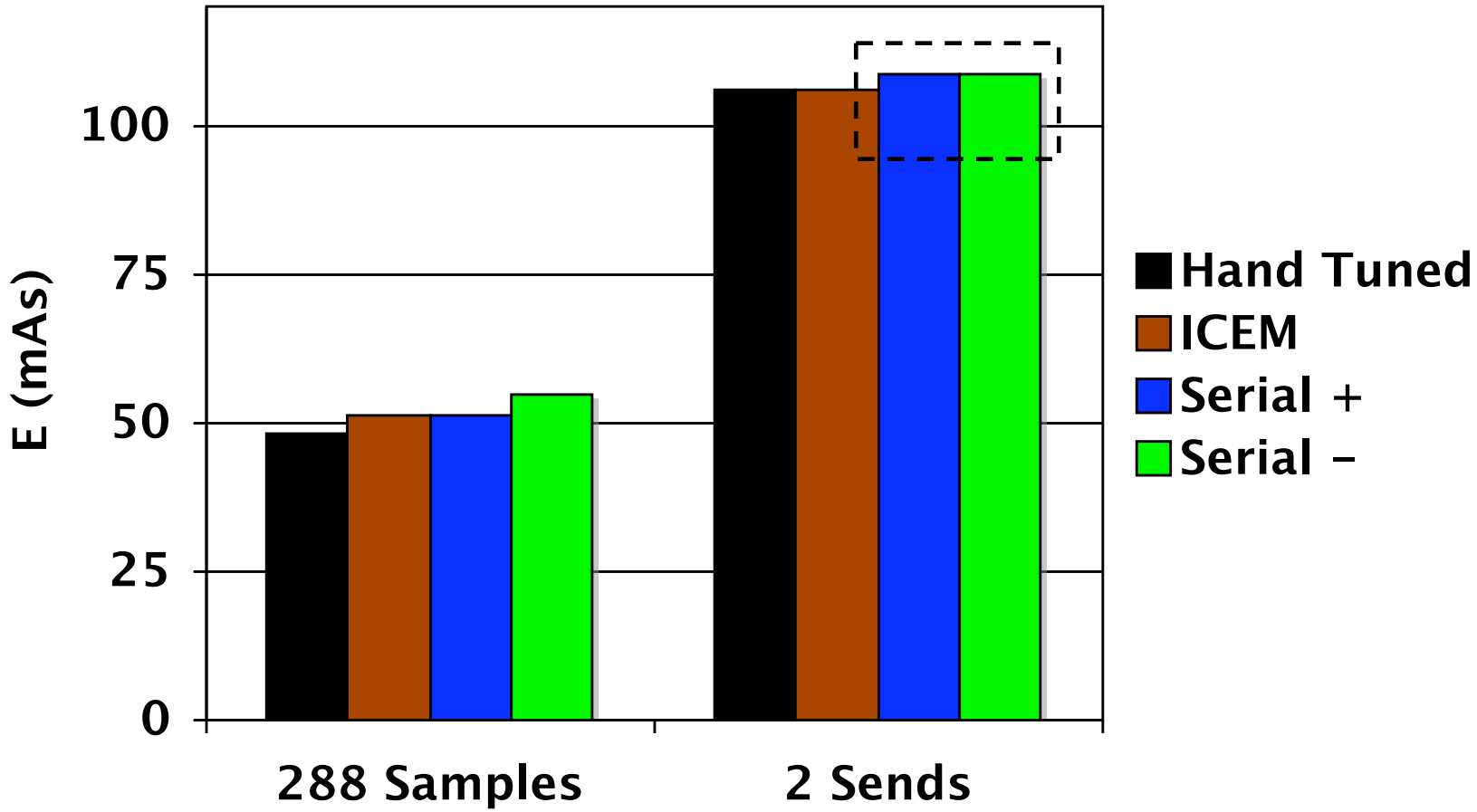
Average energy consumption for application operations

Application Energy Consumption



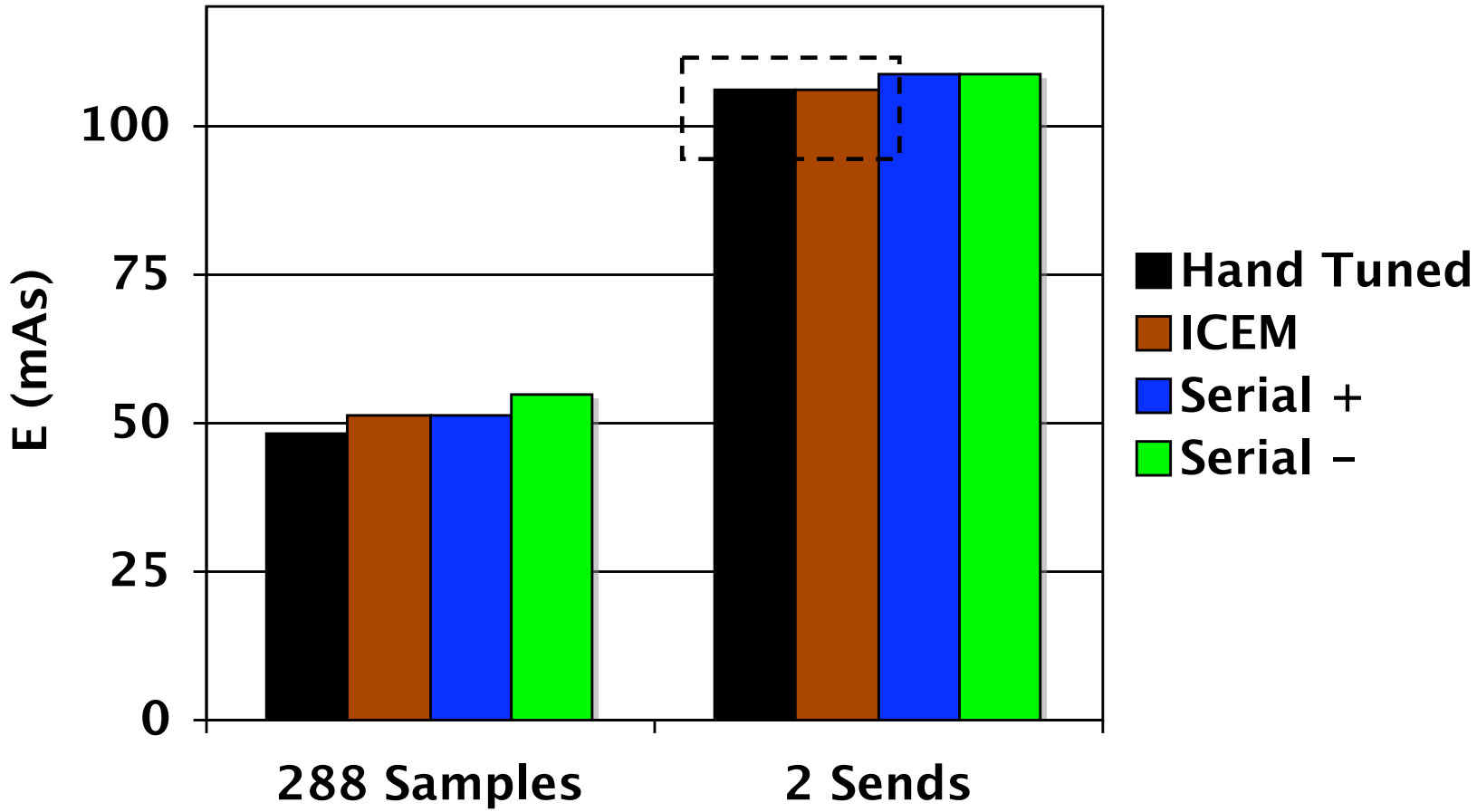
Application energy with 5 minute sampling interval and one send batch every 12 hours

Application Energy Consumption



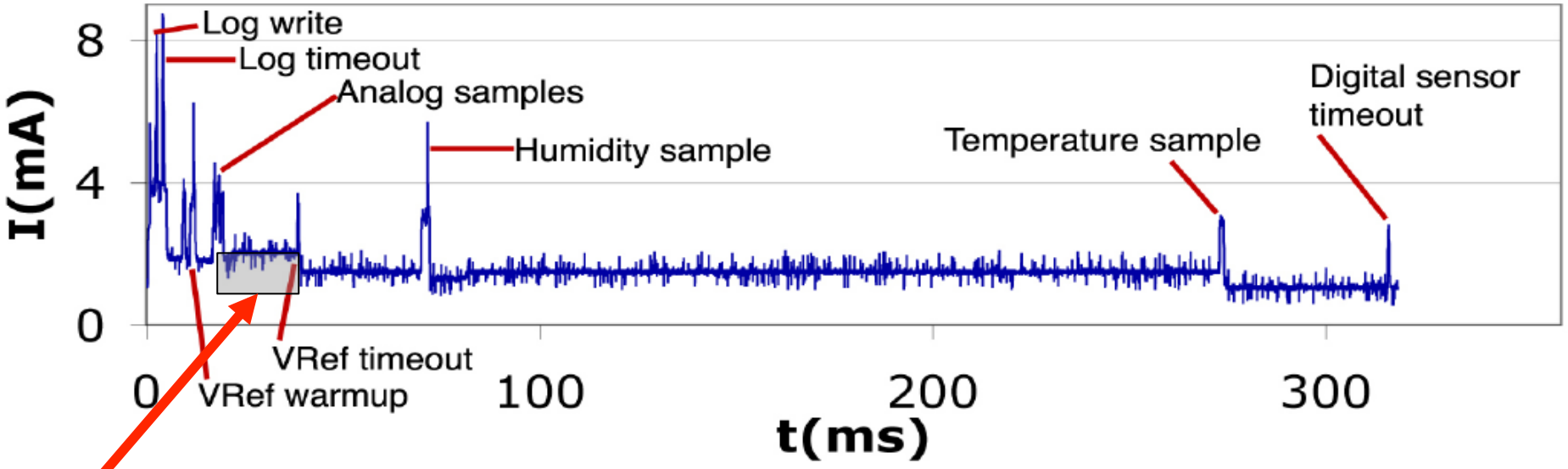
Application energy with 5 minute sampling interval and one send batch every 12 hours

Application Energy Consumption



Application energy with 5 minute sampling interval and one send batch every 12 hours

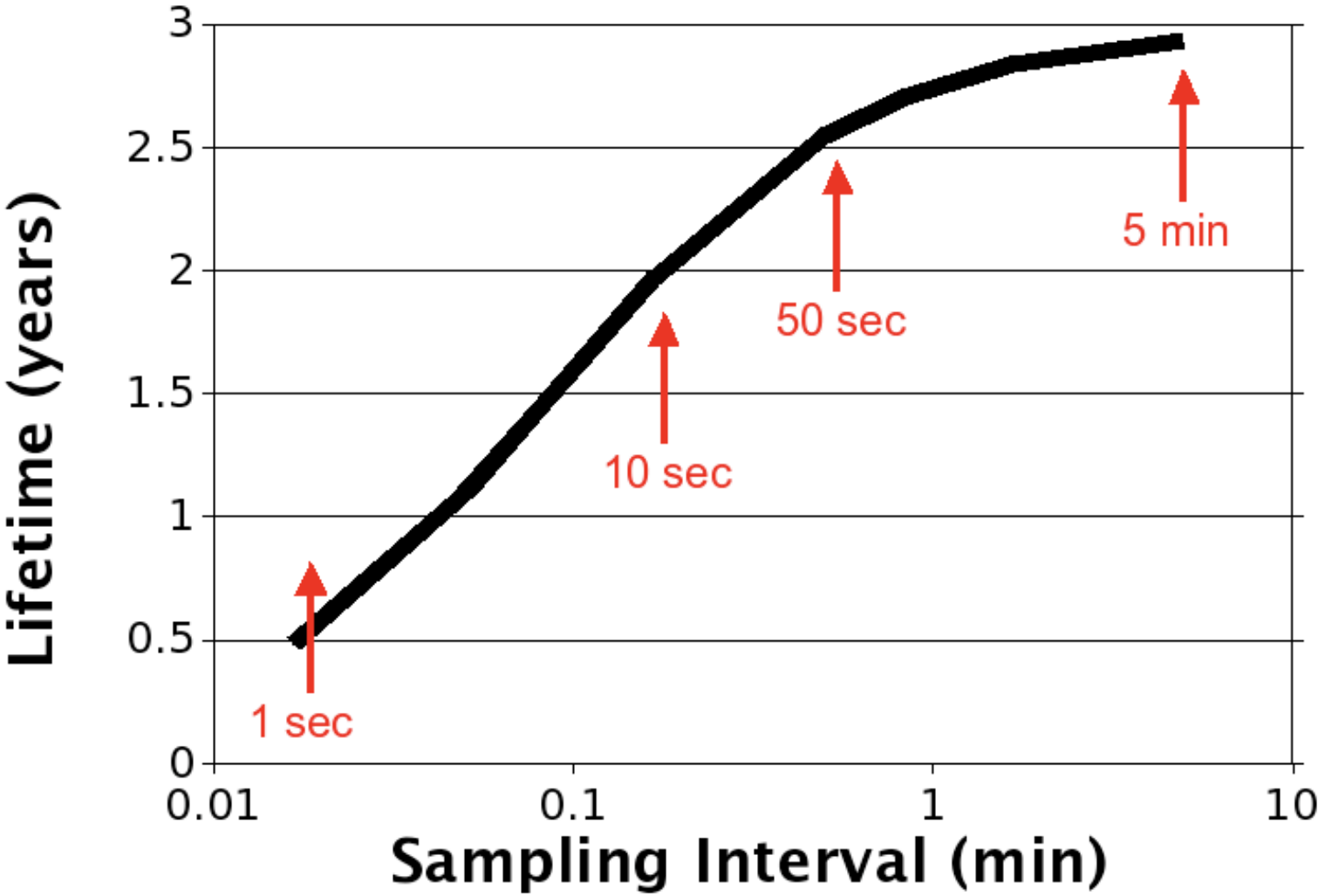
Sampling Power Trace



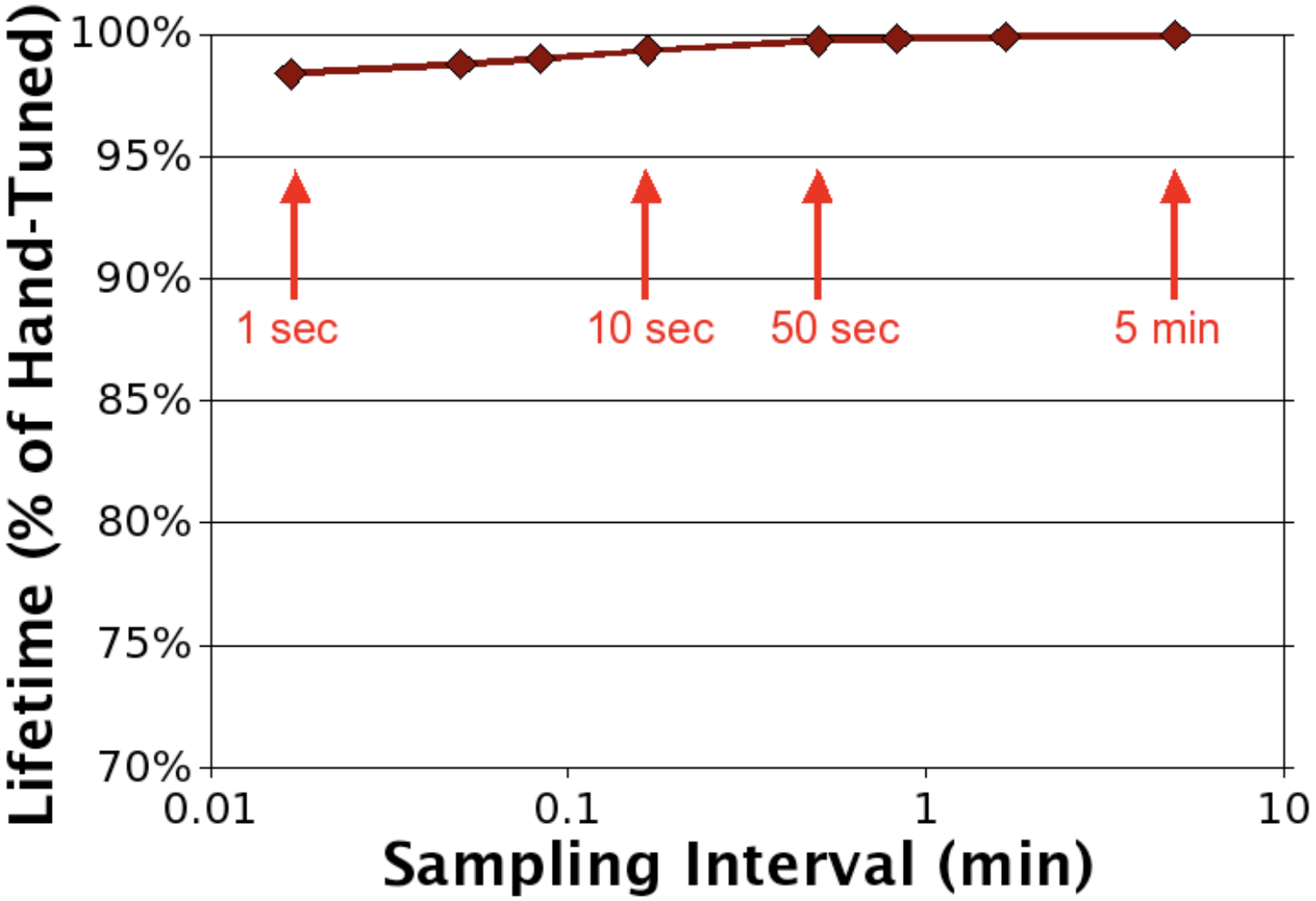
Overhead of ICEM to Hand-Tuned Implementation
= ADC Timeout + Power Lock Overheads
With 288 samples per day
≈ 2.9 mAs/day
≈ 1049 mAs/year

Insignificant compared to total
5.60% of total sampling energy
0.03% of total application energy

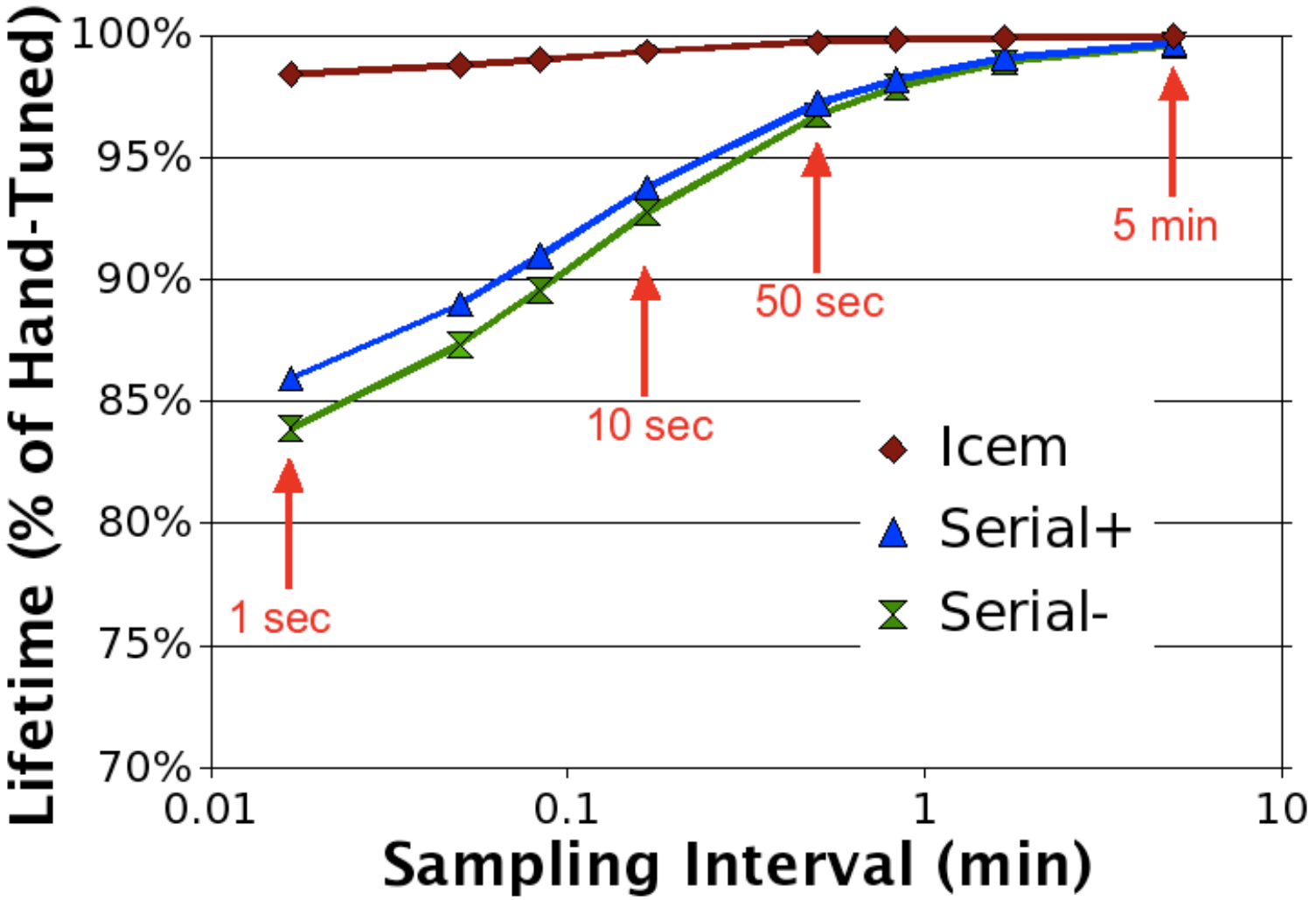
Expected Node Lifetimes



Expected Node Lifetimes



Expected Node Lifetimes



Evaluation Conclusions

- Conclusions about the OS
  - Small RAM/ROM overhead
  - Small computational overhead
  - Efficiently manages energy when given enough information
- Conclusions for the developer
  - Build drivers short power down timeouts
  - Submit I/O requests in parallel

Conclusion

- **ICEM**: Integrated Concurrency and Energy Management
  - Device driver architecture for low power devices
  - At least 98.4% as energy efficient as hand-tuned implementation of representative application
  - Simplifies application and driver development
  - Questions the assumption that applications must be responsible for all energy management and cannot have a standardized OS with a simple API

Questions?

Washington University in St. Louis

TKN Telecommunication Networks Group

ARCHROCK

intel®