

# Kelvin

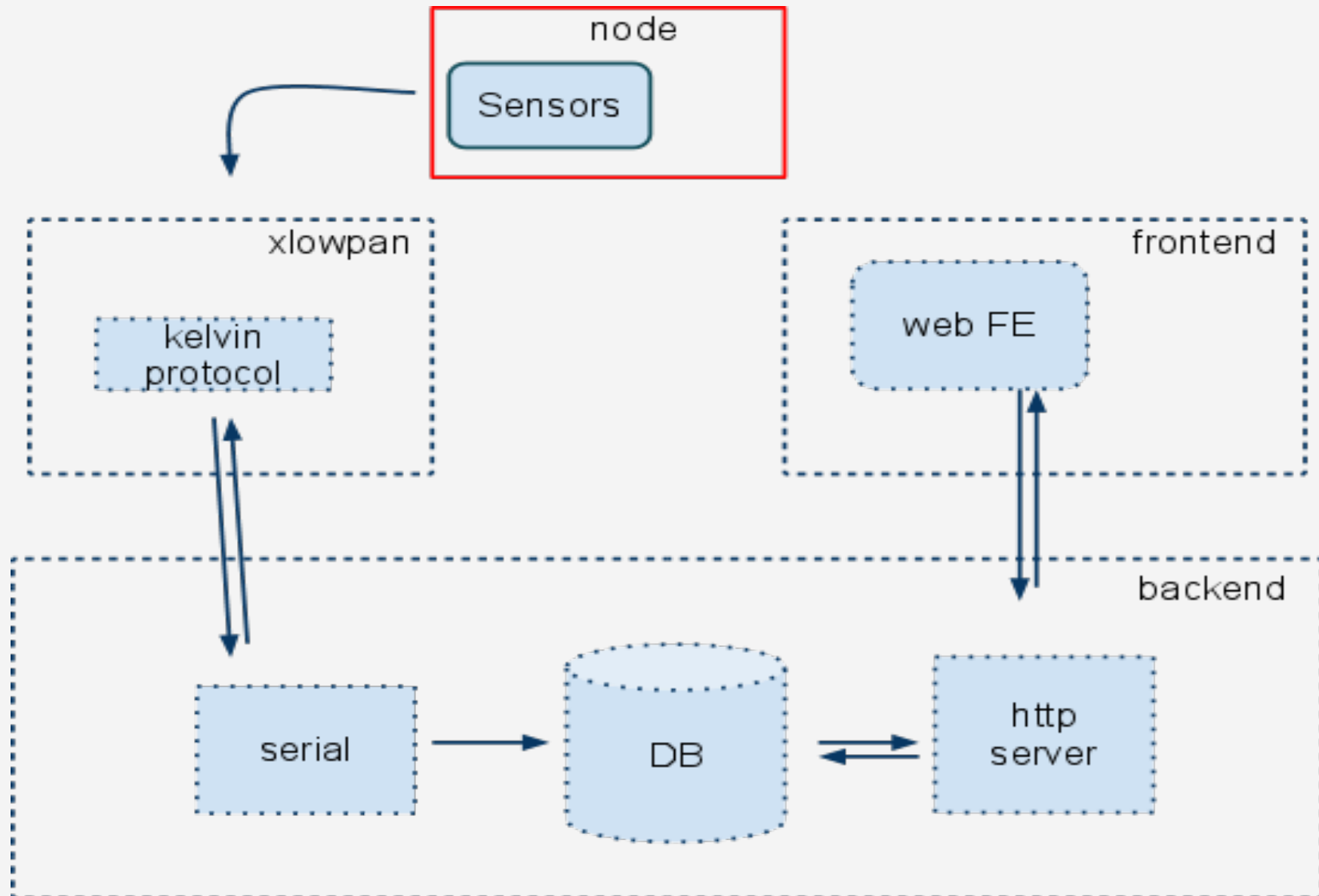
## Flexible Monitoring of Built Environments

**Nicky Ellakirk**  
**Marco Elver**  
**Ivan Konstantinov**  
**Peter Nock**  
**Maciej Soczka**

# Project Outline

- Kelvin is a wireless sensor network built on the Prospeckz platform.
- The aim of the project was to create a building monitoring system without the usual installation overheads.
- To achieve this we use a wireless mesh network and a central server.
- Every node hosts one sensor and will relay packets to its neighbours.
- The server issues requests and collects results which are output to a web based front end.
- The web front end allows configuration and graphs data.

# Roadmap:Nodes



# Node Hardware

Built on the Prospeckz platform.

One sensor per node to give maximum possible flexibility.

Unified firmware architecture - any sensor works on any node.

Modular sensor design - Quick and easy to create new nodes.

# Sensor Hardware

At every node there is a sensor which monitors some aspect of the environment.

We are currently using a small set of sensors to demonstrate the system working, other sensors can be added for more specific measurements.

The sensors deployed here are:

Temperature and humidity

- Provided by a digital calibrated chip.

Atmospheric pressure

- Provided by a small analogue sensor.

Ambient light level.

- Provided by a transistor based analogue chip.

# Sensor Hardware

Sensor modules are constructed with off the shelf components, these were chosen with voltage/current characteristics that suit the Prospeckz.

Light and pressure sensors are analogue and have linear transfer functions allowing easy conversion of ADC readings to the required units. Calibration is done using formula provided in data sheets.

Temperature and Humidity sensor is a single chip using digital communication on a 2 wire serial interface. This chip is pre-calibrated and has extremely fine sample resolution.

# Sensor Hardware

Challenges associated with implementing sensor sampling:

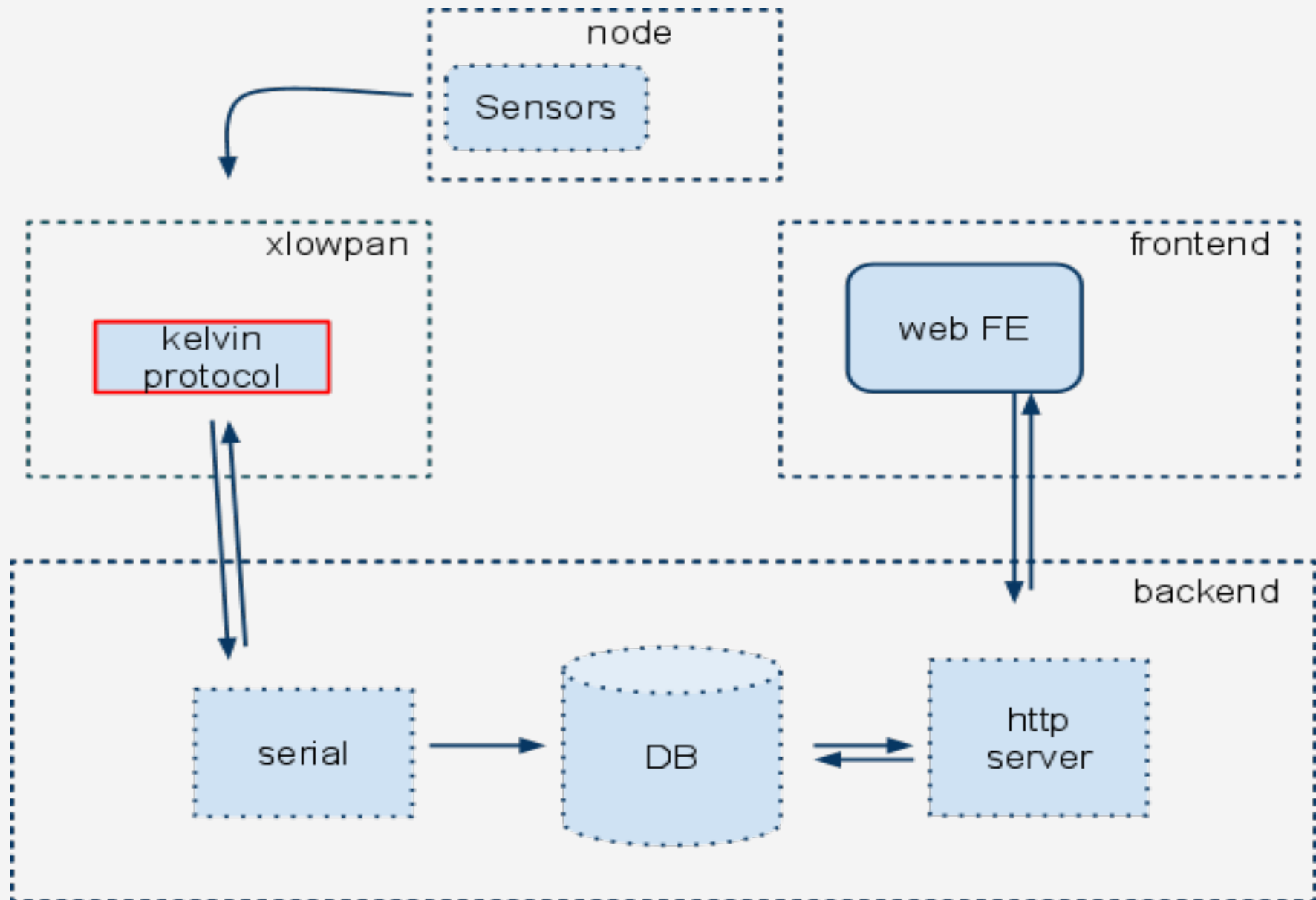
ADC readings were found to be unstable during testing and pin configuration was limited.

Both problems were solved by passing inputs through an amplifier stage.

Bi-directional digital communication over a single port requires the data pin drive mode to be changed during operation.

This is easily done at burn time but proved some what more difficult between interface clock pulses.

# Roadmap: Kelvin Protocol

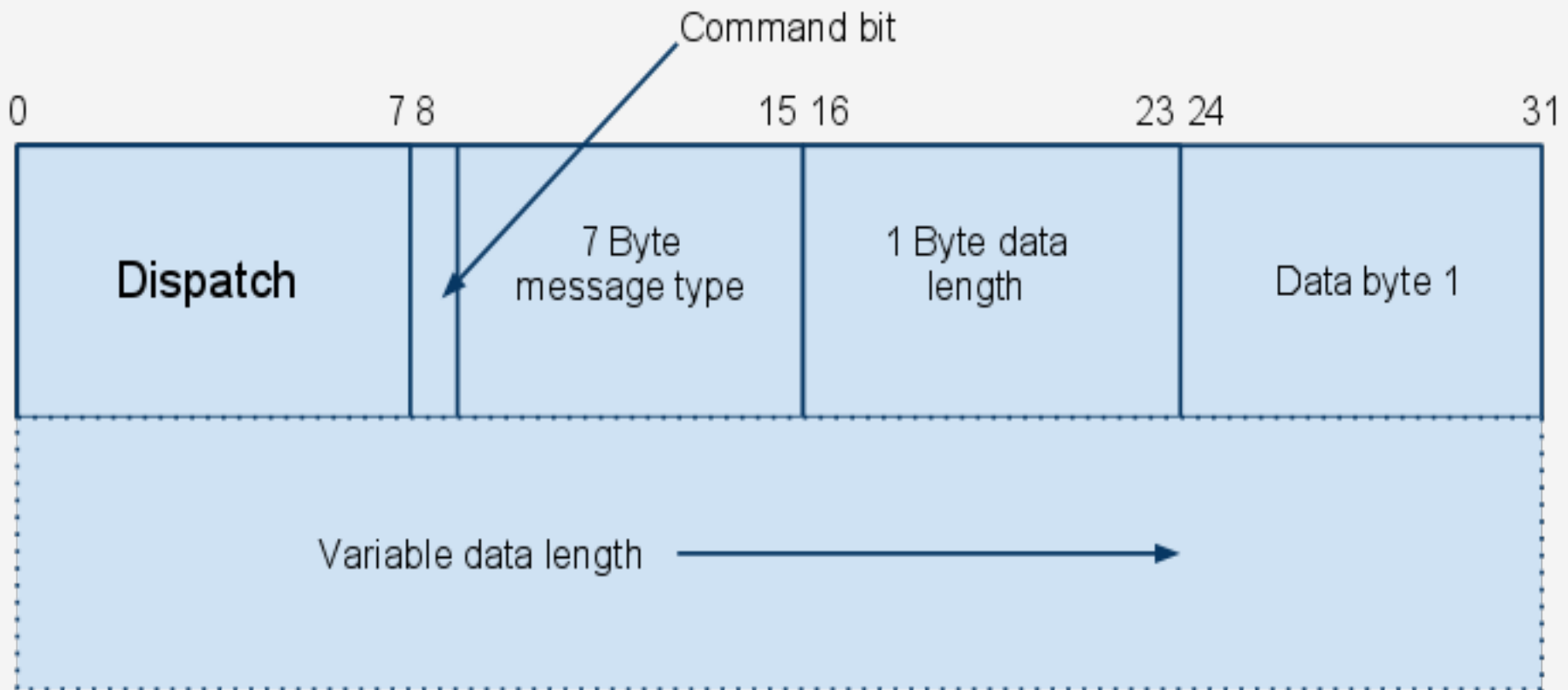


# Kelvin Protocol

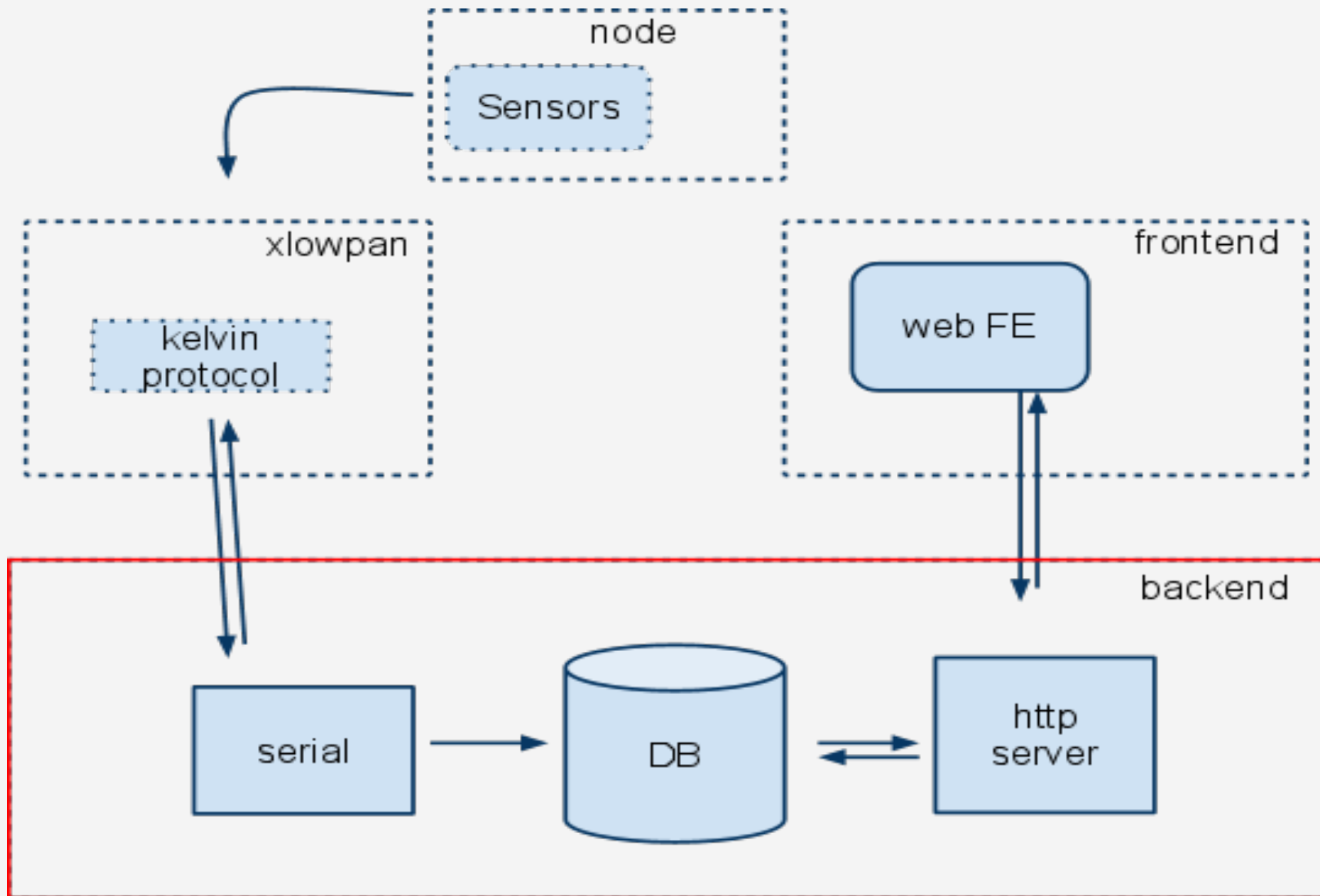
## Application Layer Protocol

- Binary packed
- Fixed length header, variable length data
- Tasked with inter-node communication and control

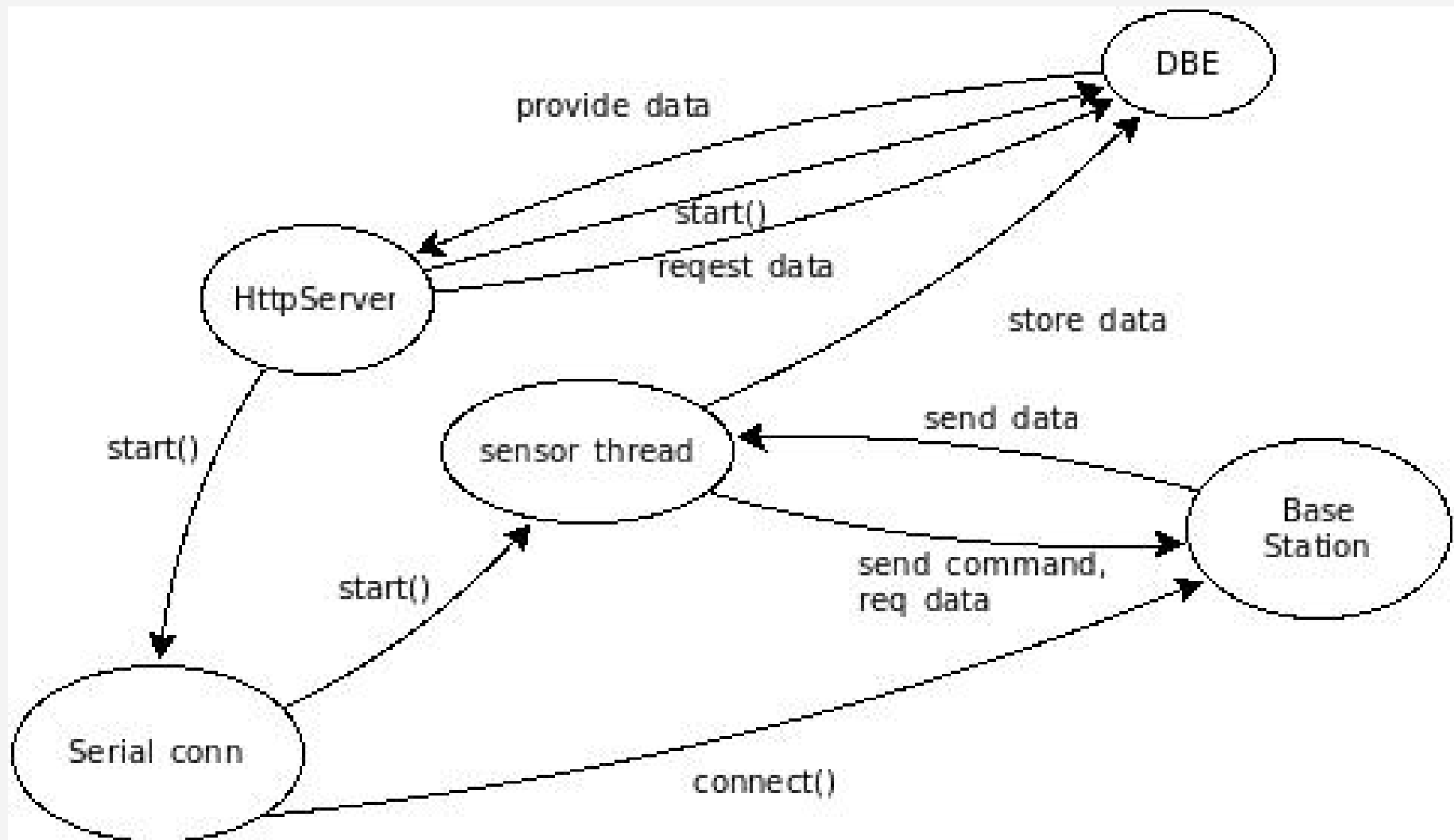
# Kelvin protocol packet header



# Roadmap: backend



- Server system, written in Python3.1



- *Http Server*
- - web server, connects all crucial components of the system
  - handler for processing user requests and outputting data
  - if `__name__ == "__main__"`:
    - initiates two `PriorityQueues`, for communication with the database
    - starts the `Database` thread
    - initiates the serial communication between back-end and base station

- *Database*

- database thread, uses sqlite3 module.
- edits, adds, removes and retrieves stored data, such as sensor data, chip address, etc.
- communication done using two separate queues, shared by the Http server and the sensor thread
  - http server or sensor thread put requests in a queue
  - DBE retrieves and processes them one at a time
  - puts results (if any) in a separate queue
- queues necessary because different threads aren't allowed to access DBE's functions
- uses XMLGenerator to convert results to requests to xml, sent to http server for outputting

- *Serial connection*
  - Instrumental to establishing serial communication with Base Station
  - uses **serial** module to access COMM ports
  - after establishing communication, starts separate thread, which issues all requests to Base Station
- *Sensor thread*
  - issues requests to Base Station, which are forwarded to all connected chips on a regular basis
  - reads received messages, calibrates data (when necessary)
  - issues requests to store data in database
  - checks for chip activity: if chip inactive for some time, maybe disconnected/faulty?

# Scheduling task

## Hardware Limitations:

- 2KB of RAM 32KB of ROM
- not enough memory for an OS (but some tiny ones were considered, embedded linux: uclinux)


## Processor:

## PSoC Family:

- CY8C2xxxx Named PSoC1 with CPU M8C

# Super loop software architecture

```
main() {  
  init(); // initialize  
  while(1){ //Super loop  
  {  
    function1();  
    function2();  
    function3();  
  }  
}
```



- function execute sequentially
- Problem: unknown execution time: there's no control over how long it will take to execute each function
- cannot handle precise time framework
- minimal hardware resource overhead
- high portability
- operates at full-speed, not energy efficient

# Software Delay

```
main() {  
    init();  
    while(1) { //super loop  
        function1();  
        delay_50ms();  
        function2();  
        delay_50ms();  
        function3();  
        delay_50ms();  
    }  
}
```

- using system clock to increment counters
- clock 32kHz
- period 32, frequency 1000
- this would be fine if execution times were known beforehand (rarely happens) and
- execution time never change (unrealistic)

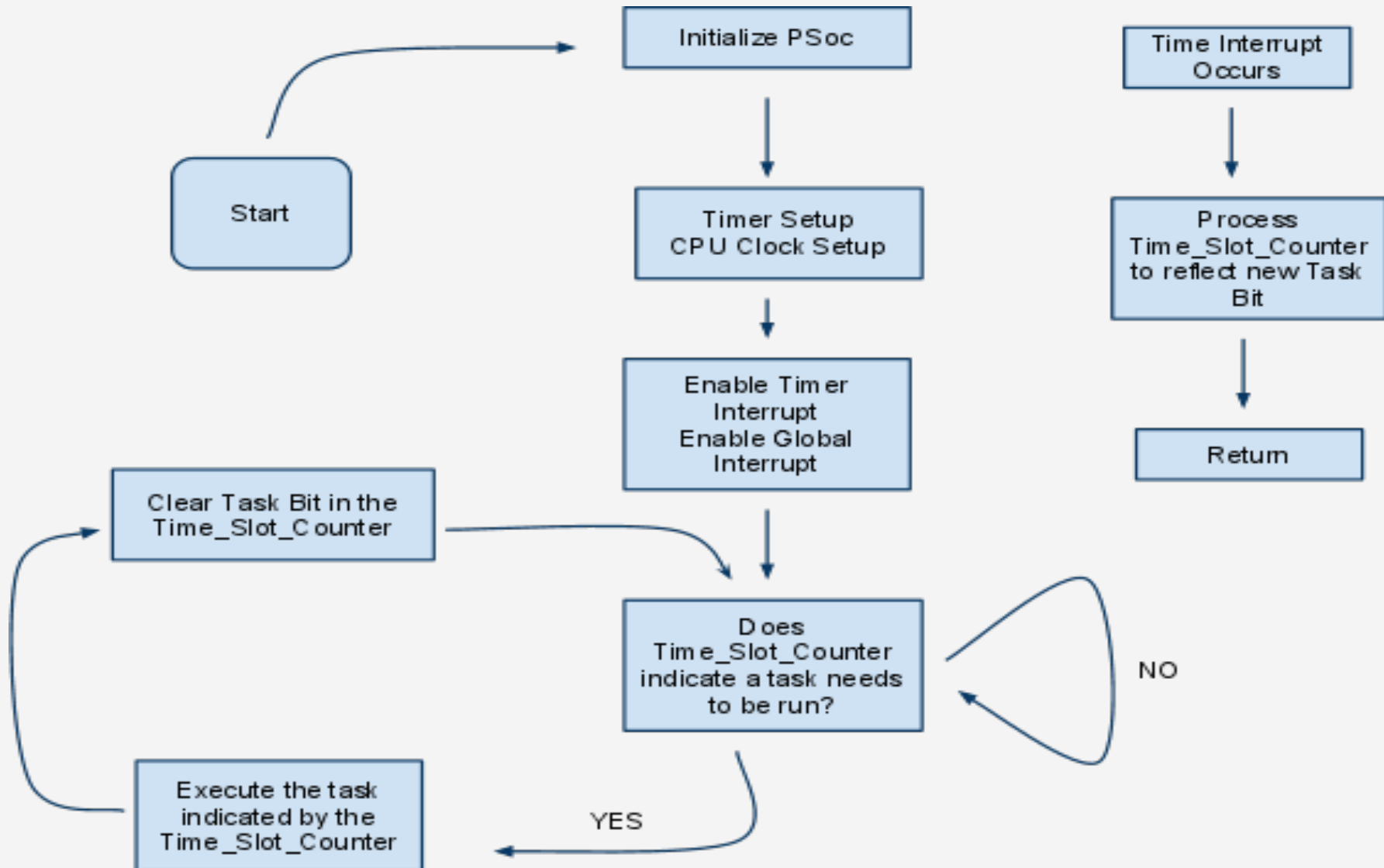
# Timer-based hardware interrupts

```
main(){
    timer_init();
    gint_enable();
    // empty infinite loop
    while(1) {}
}

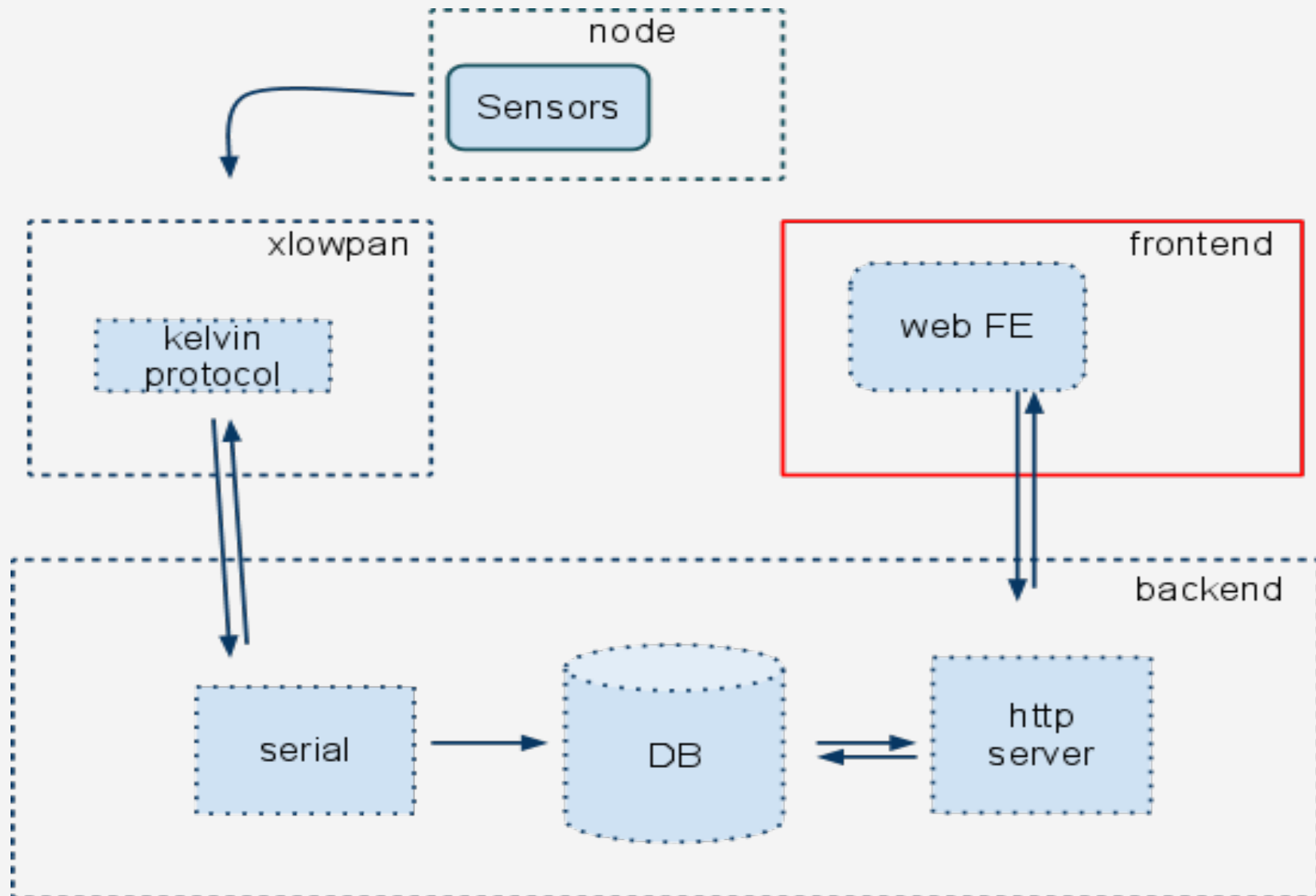
// interrupt generated
void ISR(){
    reset_timer_flag();
    function(); // do the job
}
```

- the interrupt generated by the timer overflow invokes Interrupt Service Routine (ISR)
- real-time processing
- reliable pre-emptive scheduling
- context switching
- Rate Monotonic Scheduling: Periodic tasks with shorter periods get the higher priority

# Scheduling Algorithm



# Roadmap:Nodes



# Frontend

- In order to make use the data recorded
- Data is exposed via the XML API
- Frontend is web based
- Utilises various new and emerging web technologies
  - XMLHttpRequest
  - HTML5 Slider, Canvas,...

Demo Time

# References:

1. [Cypress Semiconductor official website](#)
2. Kelvin source code: [code.google.com/p/kelvin](http://code.google.com/p/kelvin)
3. Rate Monotonic Scheduling Algorithm: [http://en.wikipedia.org/wiki/Rate-monotonic\\_scheduling](http://en.wikipedia.org/wiki/Rate-monotonic_scheduling)