CISTER - Research Center in
Real-Time & Embedded Computing Systems

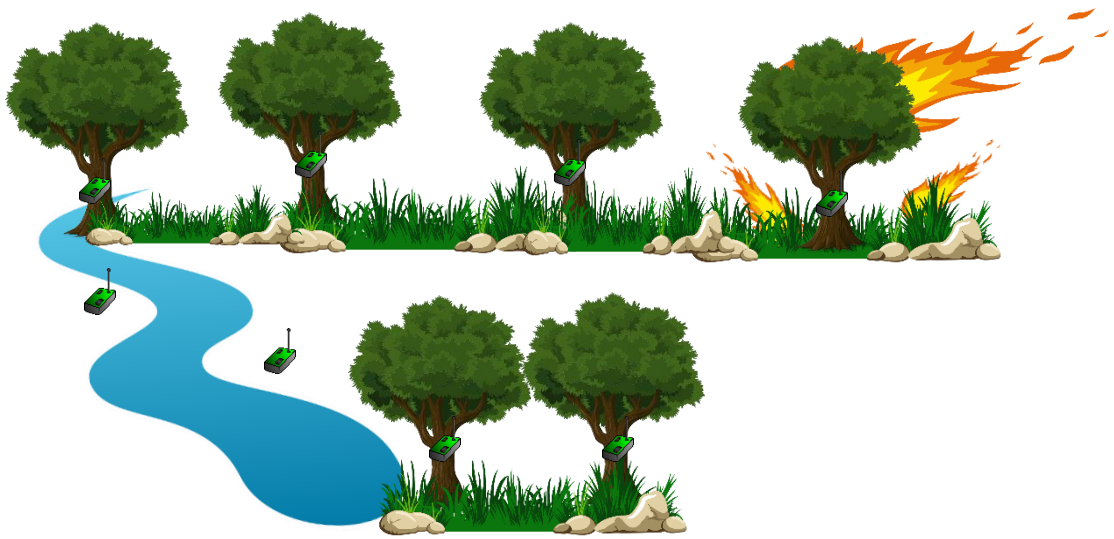
Programming for Wireless Sensor Networks

Shashank Gaur, Luca Mottola, Eduardo Tovar

isep Instituto Superior de Engenharia do Porto

INESCTEC

Wireless Sensor Networks

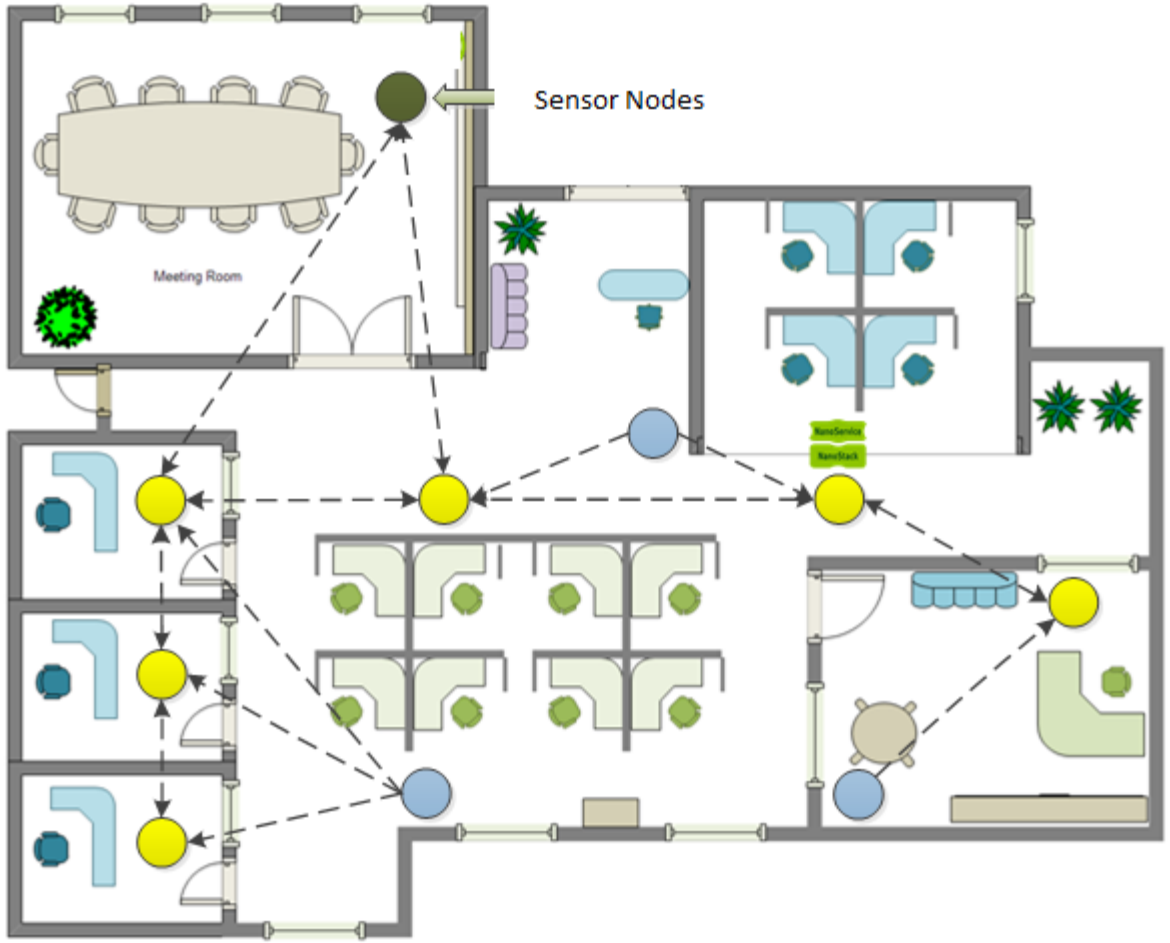


- **Application**
  - Personal Health Care
  - Smart City/Home
  - Structural Health
  - Forrest Fire Detection
- **Sense, Collect & Act**
  - Periodically
  - Event triggered
- **Capabilities**
  - Many Physical Quantities
  - Better Processing
  - Single Physical Quantities
  - Limited Processing

Normal Habitat
Check Presence
And
Maintain Heating

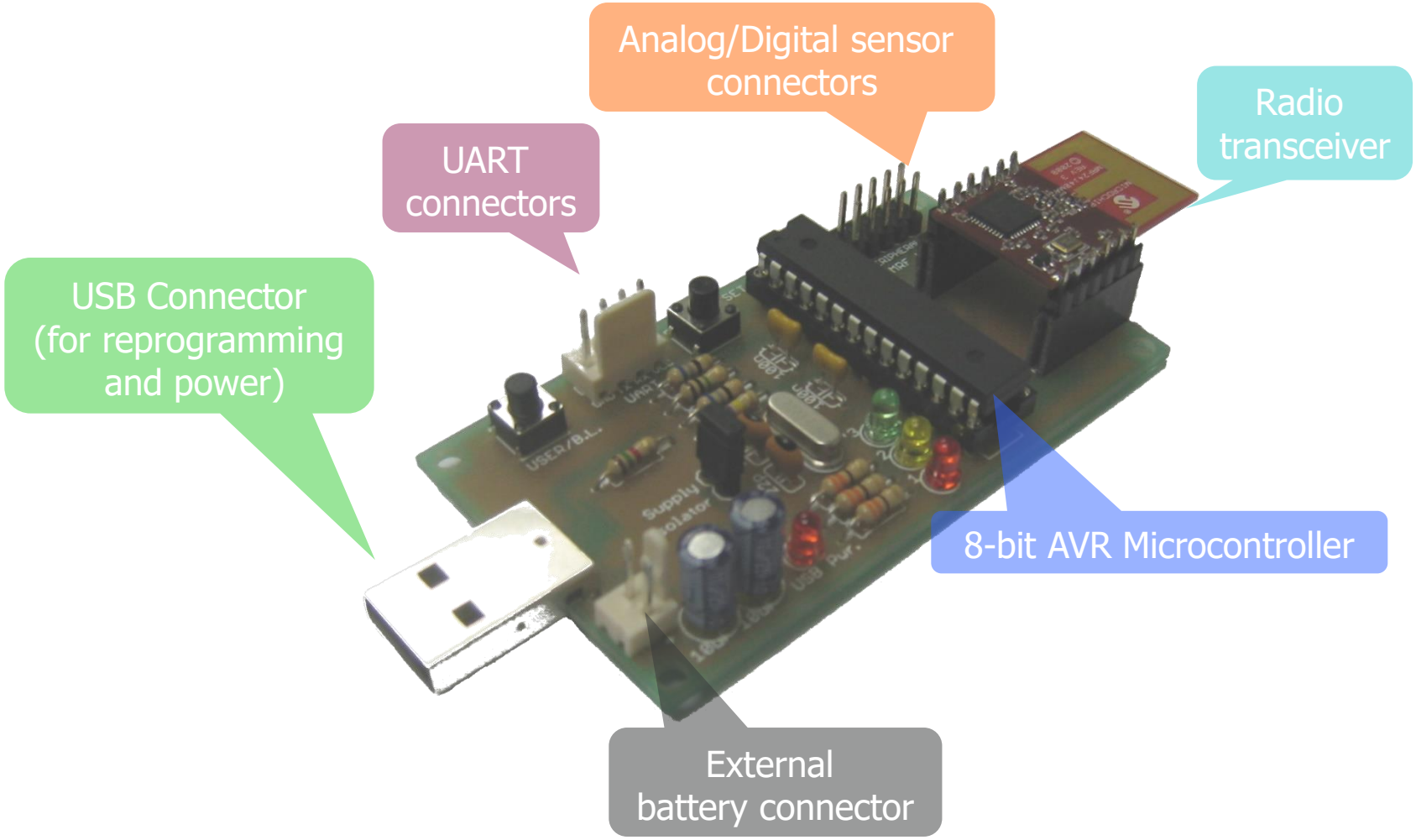
Fire Alarm
Detect Presence
And
Open Doors

Fire Alarm
Monitor high
temp area &
Notify Firemen



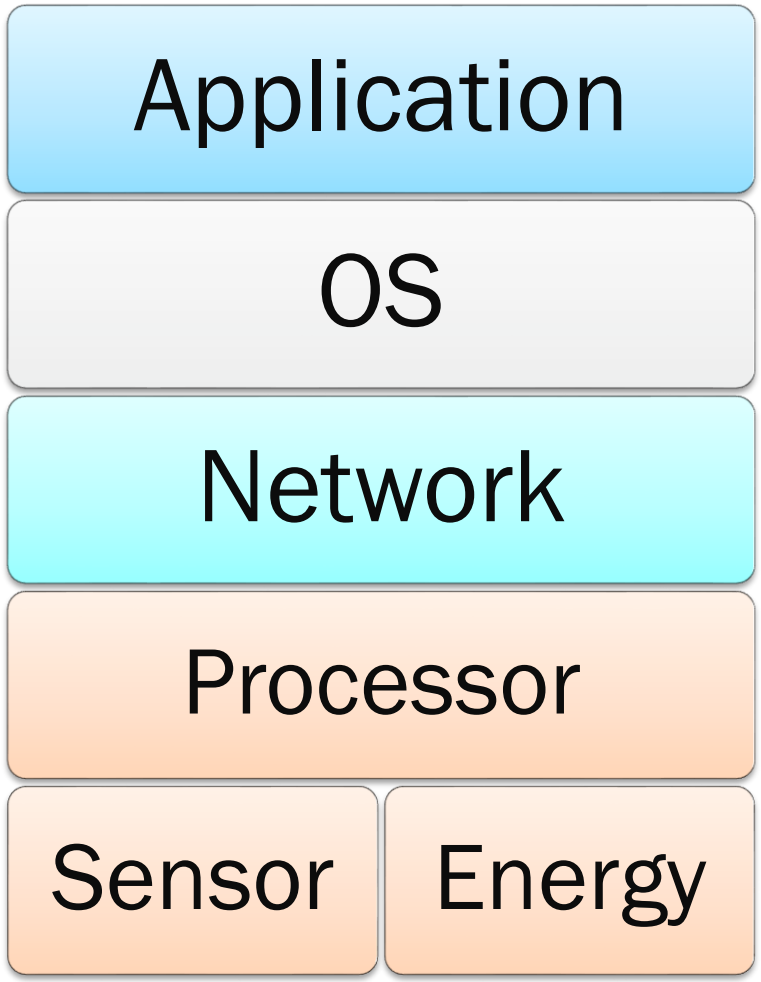
Hardware for Sensor Networks

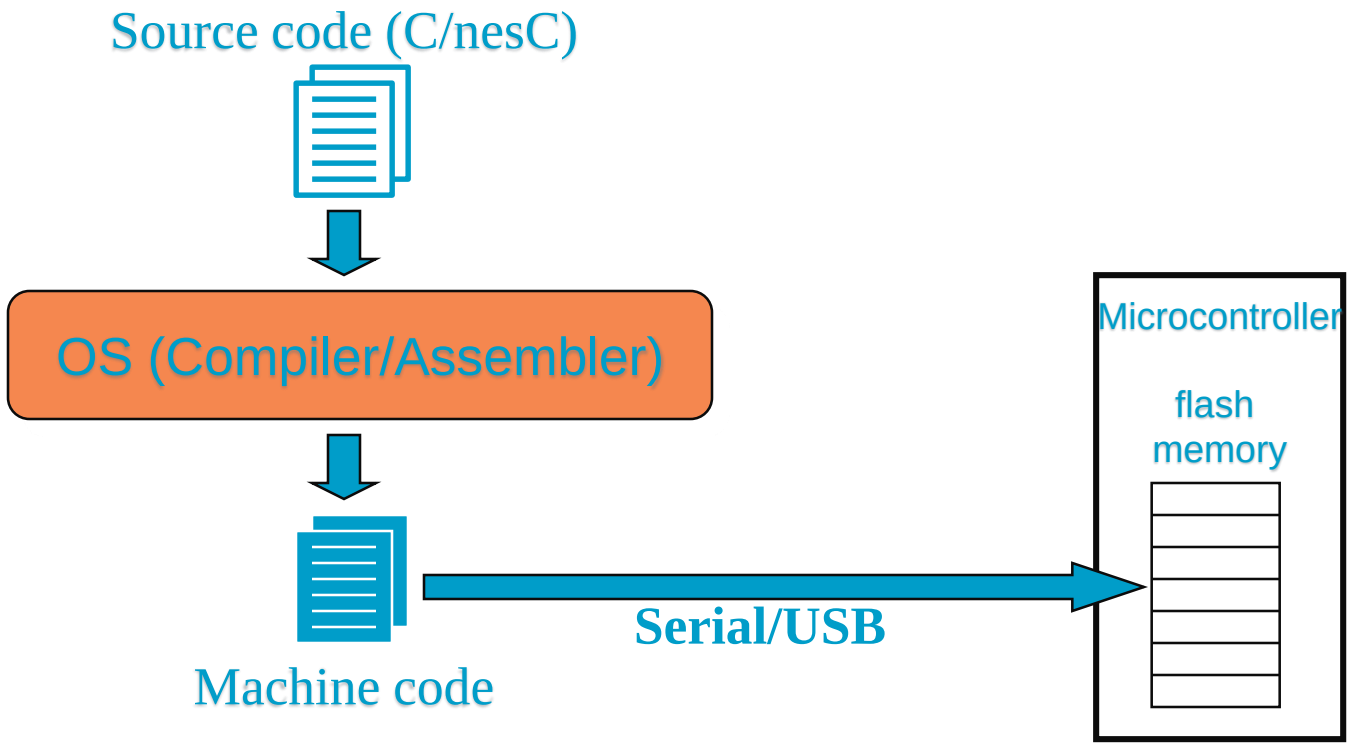
- Sensors or Actuators
- Processor unit (low power in mW)
  - mostly microcontrollers with ADC/DAC, UART etc
- Wireless Communication (Range in meters)
  - IEEE 802.15.4 compliant
- RAM (2-10 KB)
- Memory/Flash (50-250 KB)


Analog/Digital sensor connectors
Radio transceiver
UART connectors
USB Connector (for reprogramming and power)
8-bit AVR Microcontroller
External battery connector

Software for Sensor Networks

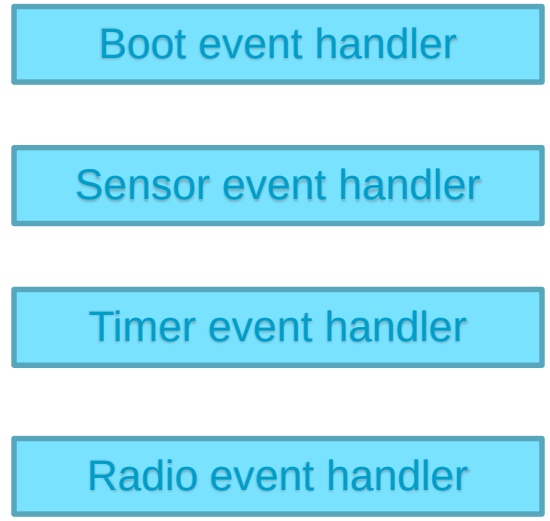
- Operating Systems (TinyOS, Contiki)
  - Resource Management (motes/nodes)
  - Protocols such as MAC, Routing
- Programming Languages
  - Low Level for hardware nodes (C, nesC)
- Additional Services
  - Localization
  - Sync the Clocks (RBS)
  - Code deployment (Trickle, CITA)




Source code (C/nesC)
OS (Compiler/Assembler)
Machine code
Serial/USB
Microcontroller
flash
memory

Traditional Programming

- Programming handles the gap between Application and OS
  - Message Passing
  - Handshaking
  - Radio Cycles
  - Interrupts and Timers
  - Polling Sensors
- Event Driven Execution
  - If this happens do that
- Exposes hardware controls
- Node Centric approach



Responsibility of the Programmer

Existing Examples	Sense	Human Concern
Auto Lights On / Off	Room Activity	Convenience
File Systems	Personal Identity & Time	Finding Info
Calendar Reminders	Time	Memory
Smoke Alarm	Room Activity	Safety
Barcode Scanners	Object Identity	Efficiency

Example

```
int sense_protothread(struct pt *pt) {
        PT_BEGIN(pt);
        PT_WAIT_UNTIL(pt, condition1);
        if (something) {
                Action();
                PT_WAIT_UNTIL(pt,condition2);
        }
        PT_END(pt)
}
```

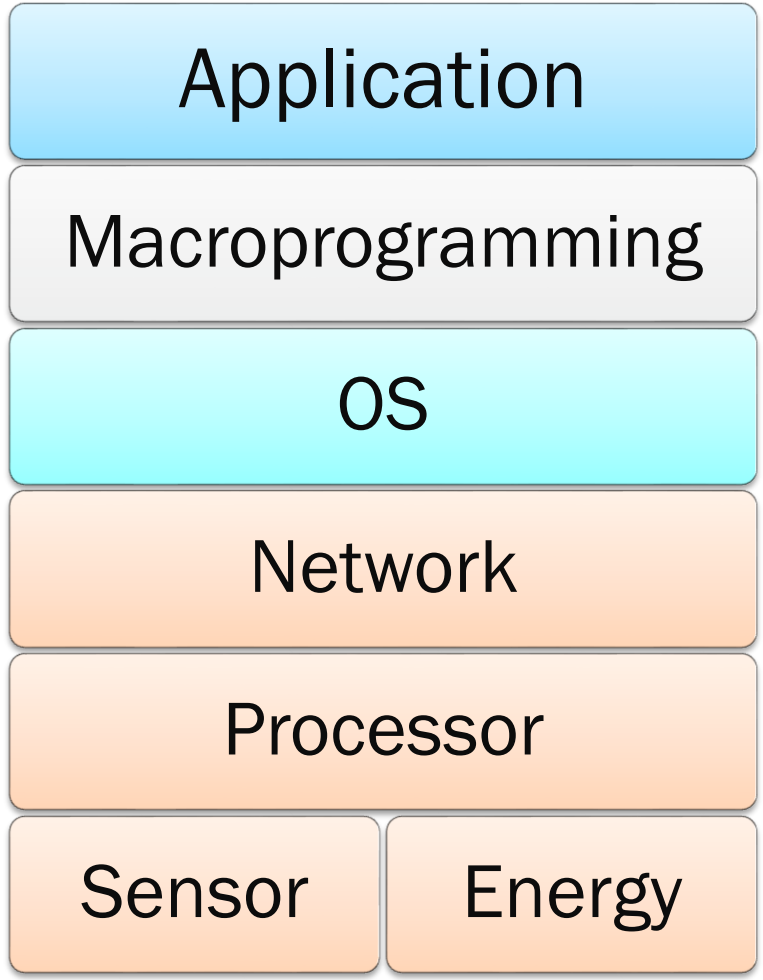
Potential Examples	Sense	Human Concern
Auto Cell Phone Off In Meetings	Identity Time Location Proximity Activity History …	Convenience
Tag Photos		Finding Info
Proximal Reminders		Memory
Health Alert		Safety
Service Fleet Dispatching		Efficiency

MacroProgramming Approach

- High level abstractions and flexibility
- Reusable Components
- Over the air programming
- Examples
  - TinyDB, Regiment, Flask, T-Res, ConesC, etc.

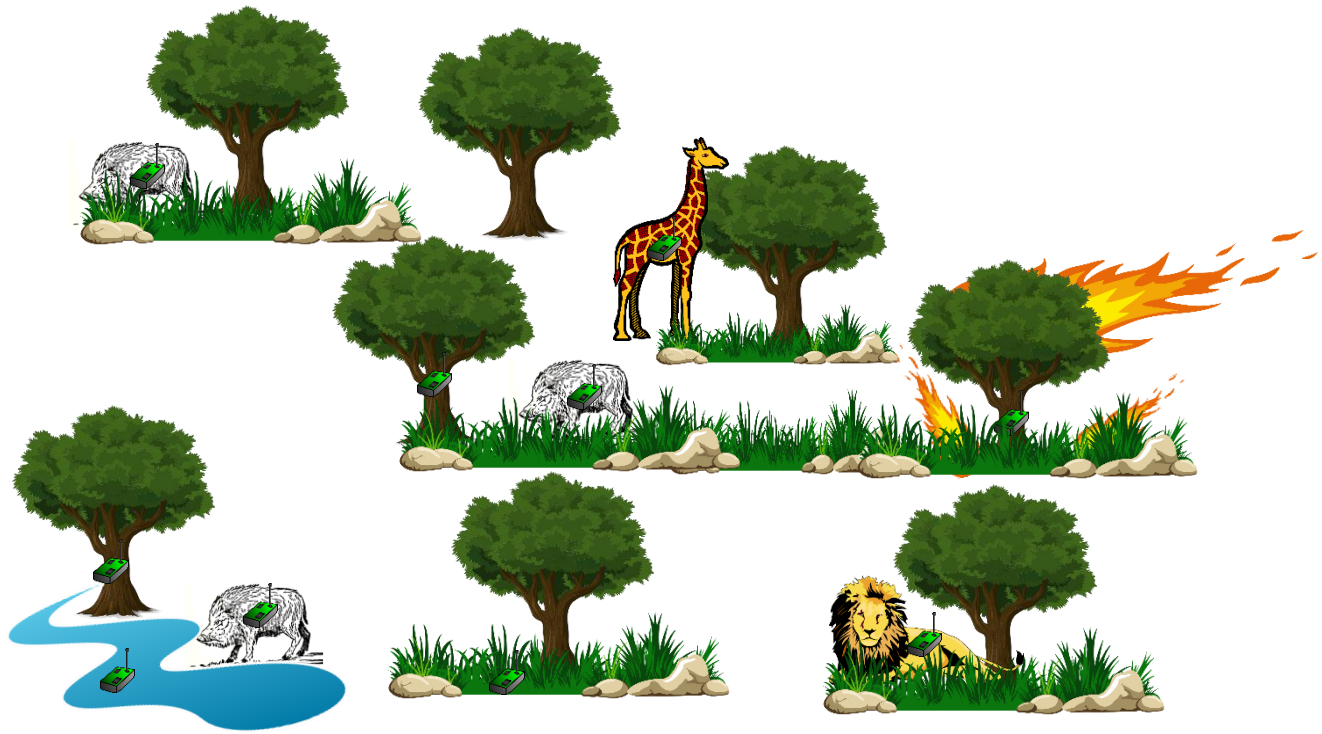
MacroProgramming Approach

- Program System Behavior
  - Not node centric
- Provides support for
  - Scaling
  - Separation
  - Adaptability
  - Validation of behavior



Adaptation in Macroprogramming

- Adaptation for Sensor Networks
  - Network & Load dynamics
  - Requirement changes
  - Performance Scaling
- Ability to modify distributed system behavior
- Reprogrammable system-level services instead of node-centric approach



Adaptation Policies

- Simple tasks may require adaptation policies
  - Scale to more nodes
  - Survive minimum thresholds
  - Respond to input changes
- Functions which react to the data behavior [online]
  - Changes to other applications/functions
  - Changes to the same application
- Can User express desired outcome in simple way?
  - Use some templates to get the exact policies

Use Case #1

- Wildlife tracking devices on animal

For Speed(S) and BatteryLevel(B):

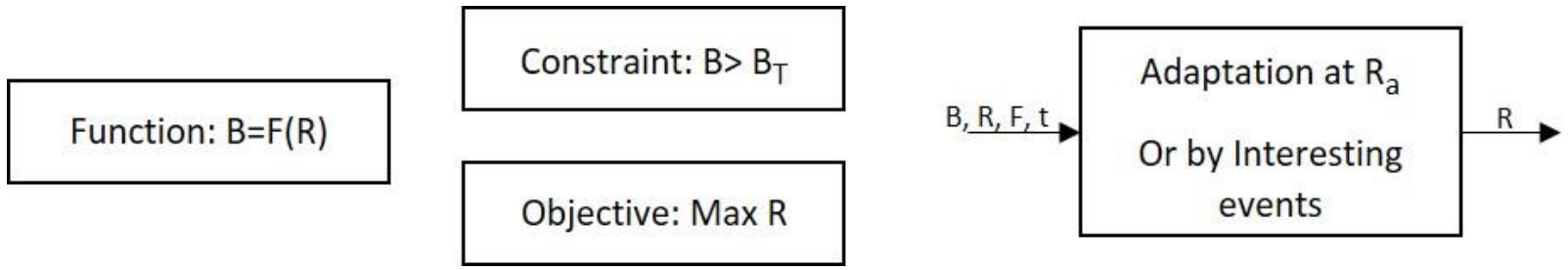
Poll GPS with Rate(R)

so that BatteryLevel(B)>Threshold(B_T) after Time(T) or at pre-defined location/distance

AND so that Maximize R

Use Case #1

- Wildlife tracking devices on animal



Use Case #2

- Refrigeration/Storage

For Volume(V) and Time period(t):

Maintain Temperature(T_F)

so that Power Usage(P)<Threshold

AND so that minimize Cost(C)

t= Time of the day or life time of the item

Use Case #2

- Refrigeration/Storage

Functions:

$T_F = f(V,P)$

$C = g(P,t)$

Constraint: $T_S - \delta > T_F > T_S + \delta$

Objective: Min C

V, t, g, T' → Adaptation at R_a

Or by Interesting events → P

T' = current temperature

Components for Adaptation Policies

- Functions
  - Relationship between input variables
- Adaptation
  - At a constant rate
  - By a trigger in the system behavior
- Constraints
  - Operational
  - Behavioral
- Solution
  - To provide the desired output

```
1  int Solve(Speed, BatteryLevelcurrent ){
2        // Calculate R using the solution for optimization
3        Return R;
4  }
5  int Function(Rate){
6        BatteryLevel = alpha*Rate;
7        return BatteryLevel
8  }
9  void sensing_thread (){
10       while()
11       {
12             timer = clock();
13             sleep(Rate);
14             GPS[time] = getGPS();
15       }
16       Return 0;
17 }
18 void timer_thread (){
19       while(){
20             timer_set(timer2, Ra);
21             if(timer_expired(timer2)){
22                   adaptation_trigger = 1;
23             }
24             Timer_reset(timer2);
25 }}
26 void adaptation_thread (){
27       While(){
28         If( adaptation_trigger == 1){
29             BatteryLevel = Function(Rate);
30             If(BatteryLevel < Batterythreshold){
31                   Speed = haversine(GPS[time-1:time]);
32                   Rate = Solve(Speed,BatteryLevel);
33             }
34             Timer_reset(timer2)// reset the timer
35              adaptation_trigger = 0
36 }}}
```

Listing 1: Pseudo-C code for GPS Use Case.

```
Block Trigger T {
       //has the ability to consolidate different
           triggers
       // a fixed rate
       Use consecutive_time 10s
       // at fixed system time
       Use time_stamp 00:00
       // use different flags or events
       Use flags
}
Block Solution S{
       Use Function f
       Use Function g
       Use Constraint
       Uses Variables a b c d
       //solve
       return a
       }
Block Constraint B{
       //define the constraint
       return true/false
}
Block Function f{
       Use variables b, c, d
       //operation
       return b
}
Block Function g{
       Use variables a,c
       //operation
       return a
}
// Adaptation here
Block Adaptation{
       If Trigger = Active:
             Solve a
             return a
}
```

Abstraction for Adaptation Policies

- Implementation in Python
  - Provide middleware for each block for Contiki OS
  - Support to scale each block to multiple instances
- Evaluation
  - Variable, Function, and other declarations are minimized by at least 50%
  - To add a new to existing adaptation policies in abstraction
    - Necessary to modify 5 lines with a fixed structure instead of several lines in C/nesC
  - Size of the compiled code remains approximately same

Thank you