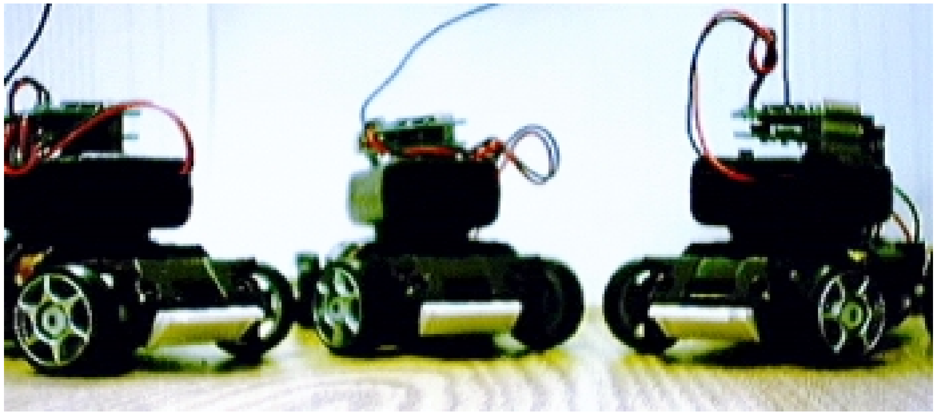
Embedded Software for Sensor Networks:
from Synchronous Specification to Distributed Architecture

BAY Team

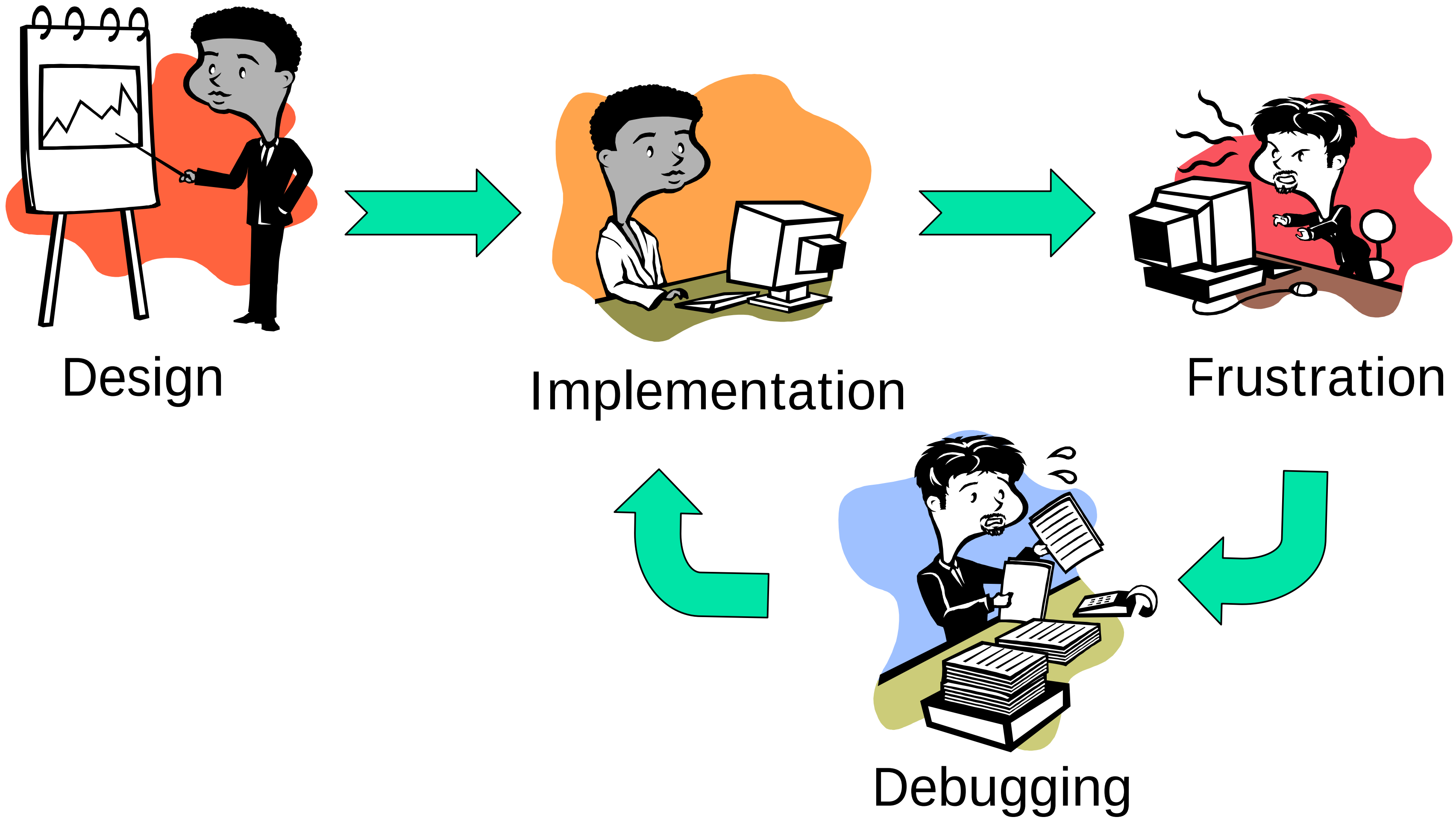
Yanmei Li, Alessandro Pinto, Bruno Sinopoli



Outline

- Motivation and Background
- Key Concept
- Design Flow
- Our system
- GALS mapping
- Demo

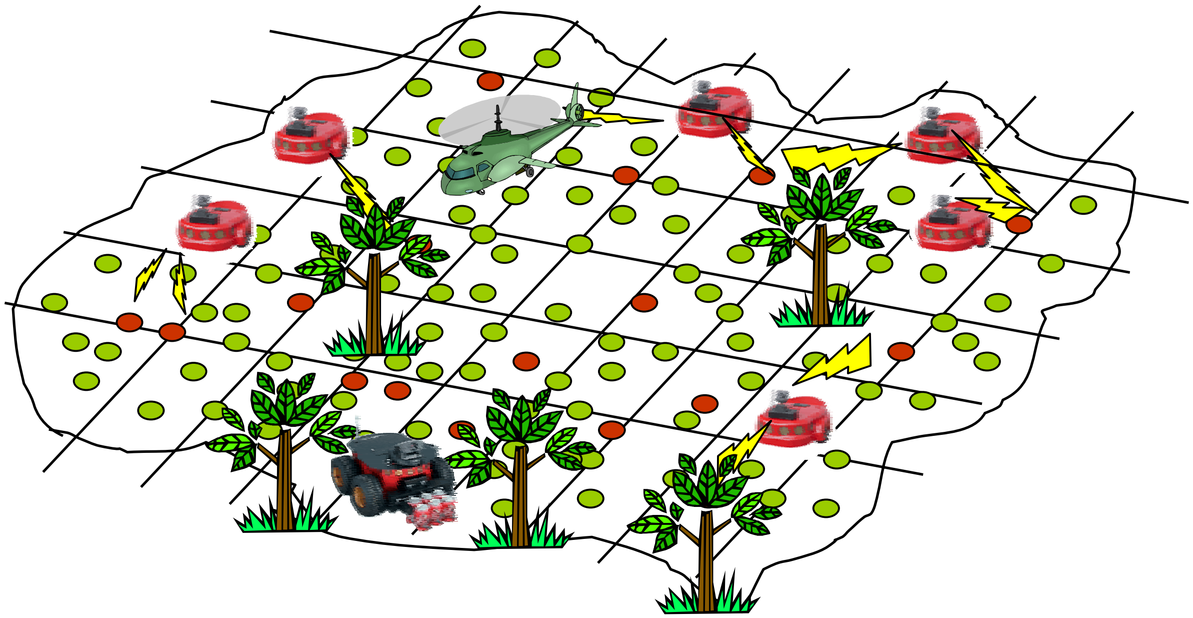
The designer problem



Motivation

- Is synchronous model appropriate to describe distributed sensor networks?
- Define a design flow from high level specification and verify its behavior
- Automatic code generation
- Distribute synchronous specification on GALS architecture

Motivation

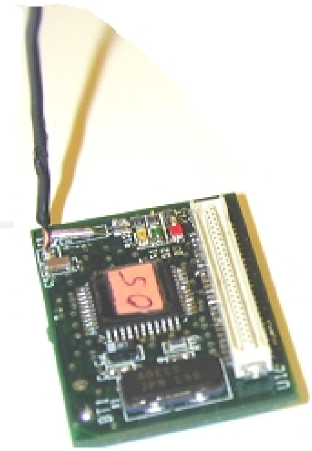


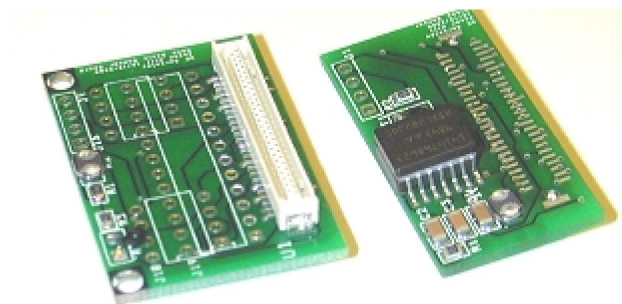
Large scale distributed systems necessarily exploit very complex behavior

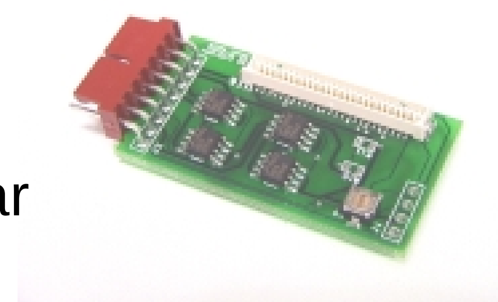
Need to make complex behavior as deterministic as possible

The hardware platform

- System board (rene2)
  - ATMEL 4Mhz, 8bit MCU, 512 bytes RAM, 16K pgm flash
  - 900Mhz Radio: 1-100 ft. range
  - I2C EPROM (logging)
  - Base-station ready
  - stackable expansion connector
    - all ports, i2c, pwr, clock…
- Sensor boards (basic)
  - basic photo, temp
- **Motor-Servo** board interfaces any combination of two motors, servos, and solenoids to a toy car platform



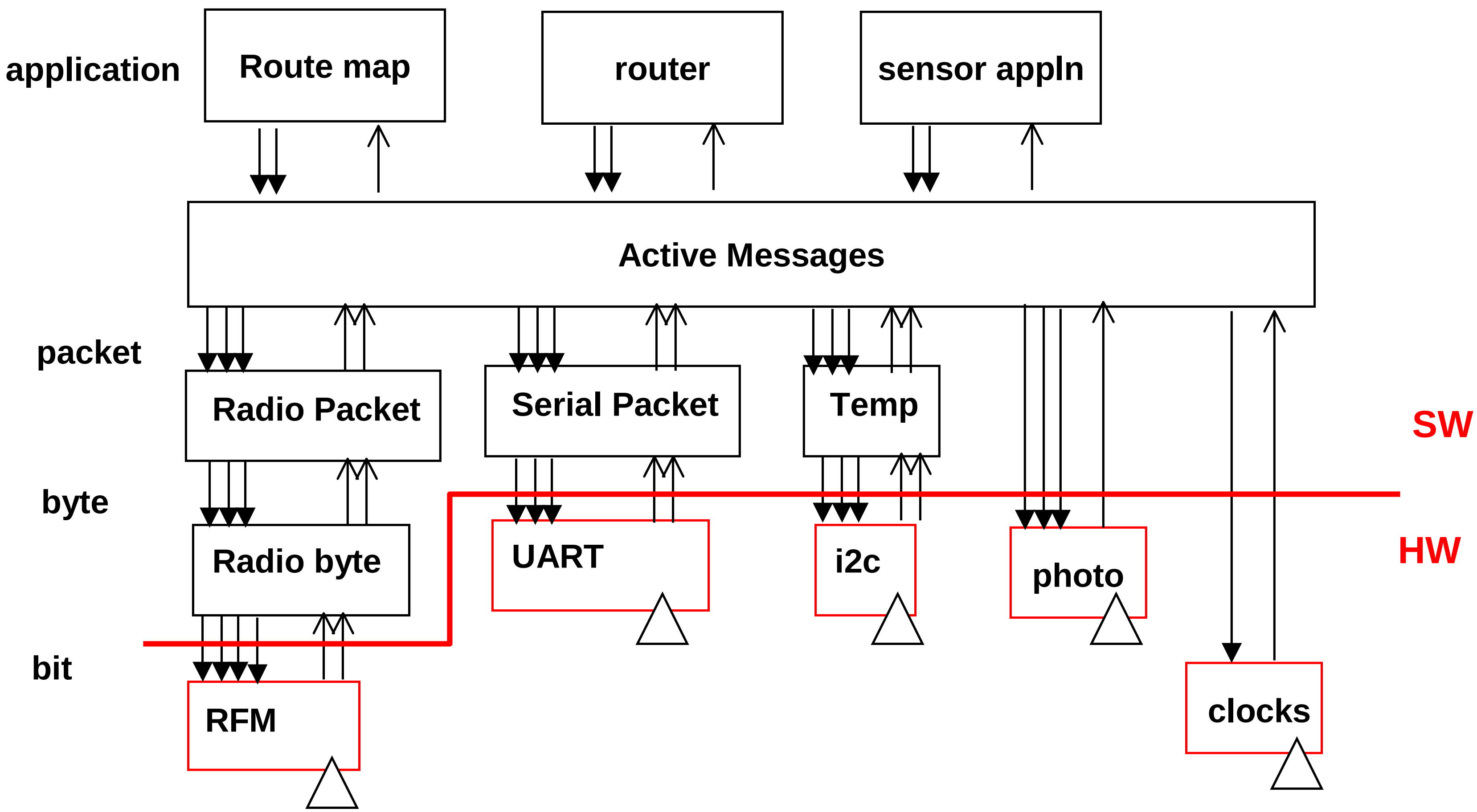




Event Based Programming Model

- System composed of state machines
- Each State Machine is a TinyOS "component"
- Command and event handlers transition a component from one state to another
  - Quick, low overhead, non-blocking state transmissions
- Allows many independent components to share a single execution context
  - Emerging as design paradigm for large scale systems
- "Tasks" are used to perform computational work
  - Run to completion, Atomic with respect to each other

Composition to Complete Application



Synch vs Asynch

- In Synchronous models
  - “Reaction based”
  - Absence ($\perp$) can be sensed and used in the specification of behaviors
  - A global tick exists

- In Asynchronous models
  - “Signal based”
  - No global tick
  - Reaction cannot be observed anymore
  - $\perp$ cannot be sensed

Endochronicity

- $\sigma \mapsto \sigma^a$ define $P \mapsto P^a$ desynchronization of P
  - This map is unique but not invertible

"If P satisfies a special condition called endochrony, then $\forall \sigma^a \in P^a$ there exists a unique $\sigma \in P$ such that $\sigma \mapsto \sigma^a$ holds"

Endochronicity: Meaning

- STS $\Phi = \langle V, \Theta, \rho \rangle$
- Set of variables $W' \subseteq W \subseteq V$
- W' clock inference of W ($W' \mapsto W$) if knowing presence/absence of $v \in W'$ allows deriving the presence absence of $v \in W$
- If $0 = V(0) \mapsto V(1) \mapsto V(2) ... \mapsto V$ then the process is endochronous

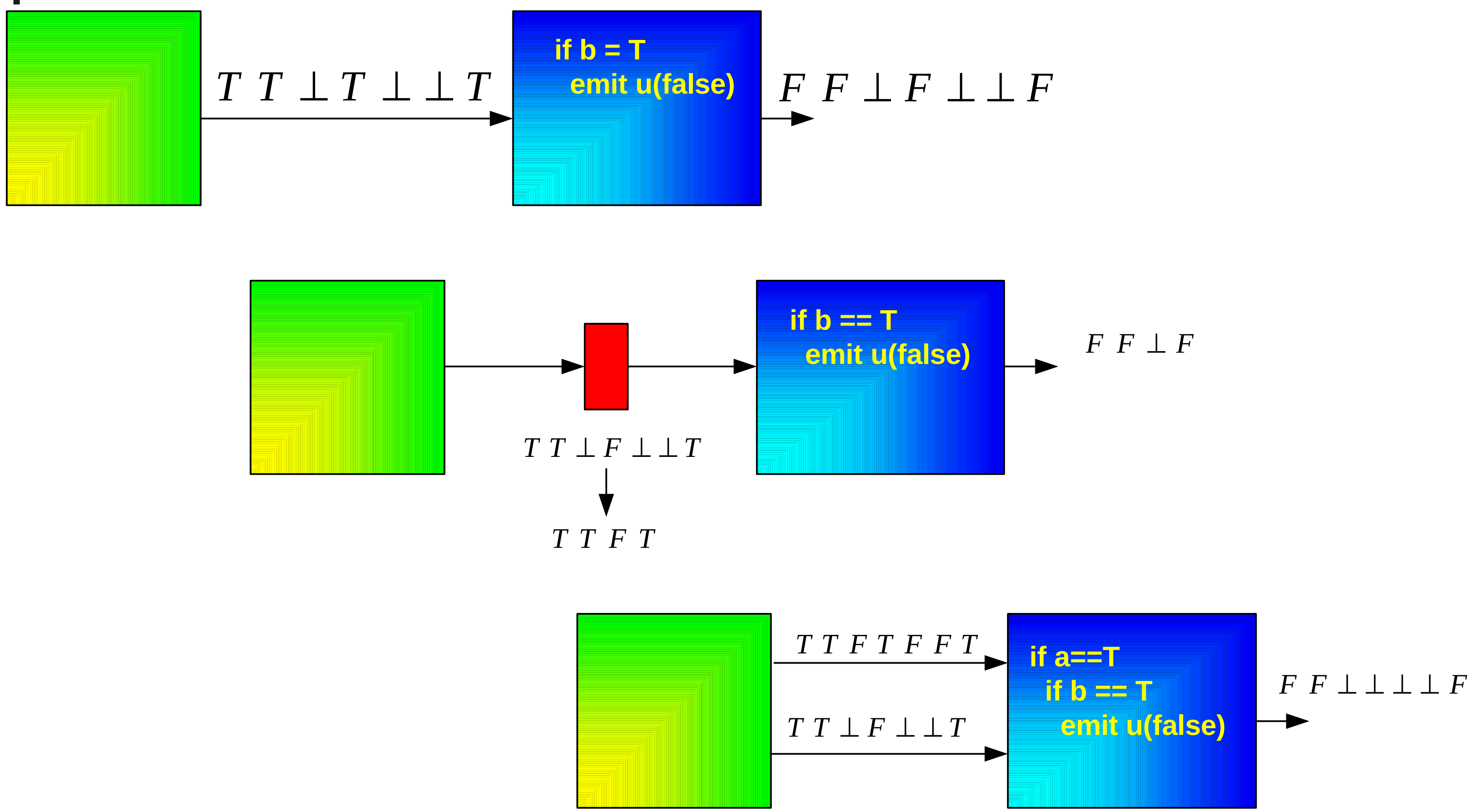
Isochronicity

- In general $(P \,\|\, Q)^a \subseteq (P^a \,\|_a\, Q^a)$

- WE want the equality to hold (no spurious behavior due to asynchronous communication)

“If (P,Q) satisfies a special condition called isochrony then the equality indeed holds”

Isochronicity: Meaning

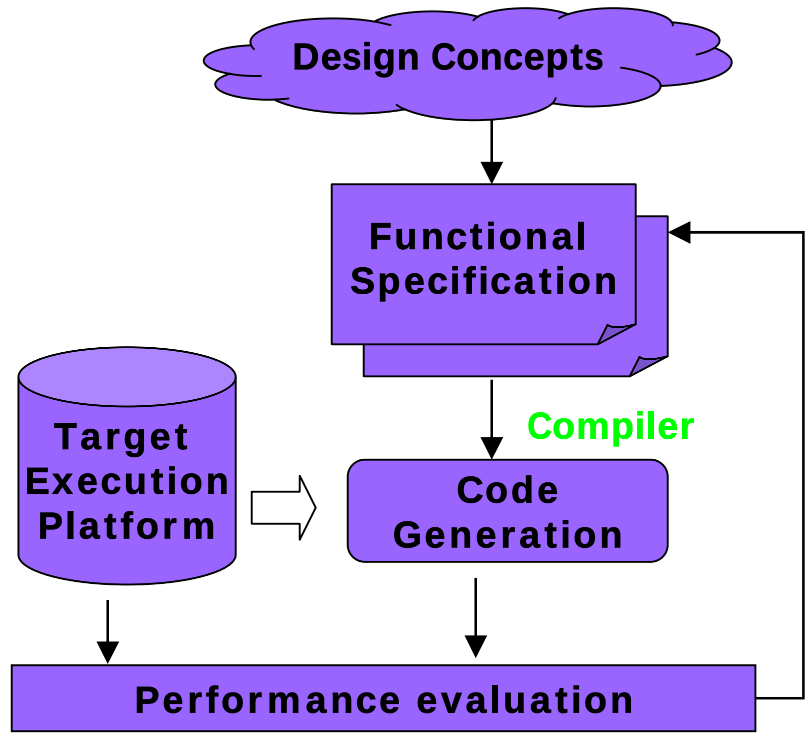


High-Level Specification: Esterel

- Synchronous language
- Control-dominated software or hardware reactive system
- High-level programming using functional models
- C code is automatically generated

Design Flow

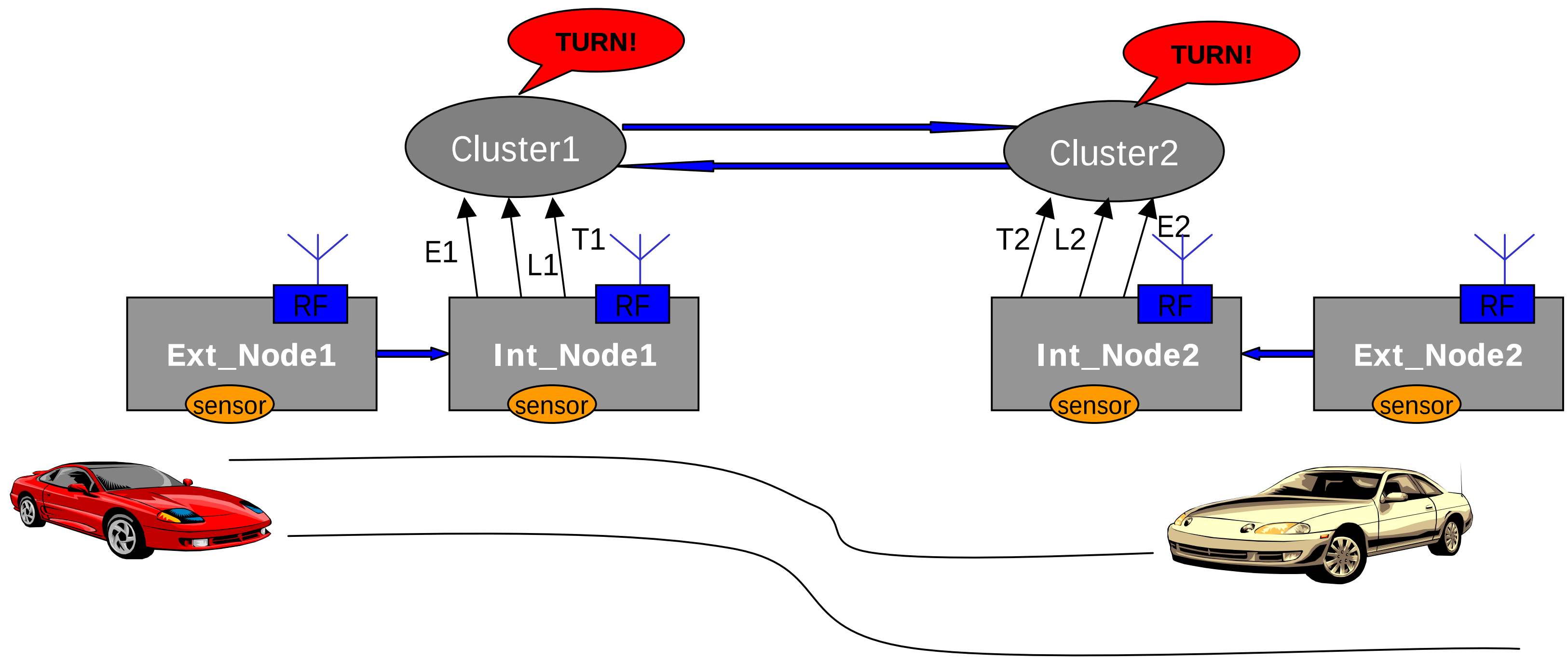
- Implement functional specification using high-level language (Esterel)
- Directly generate function modules(C code) through compiling
- Build interfaces and wrappers needed by the execution platform



Run on an embedded target !

System Structure

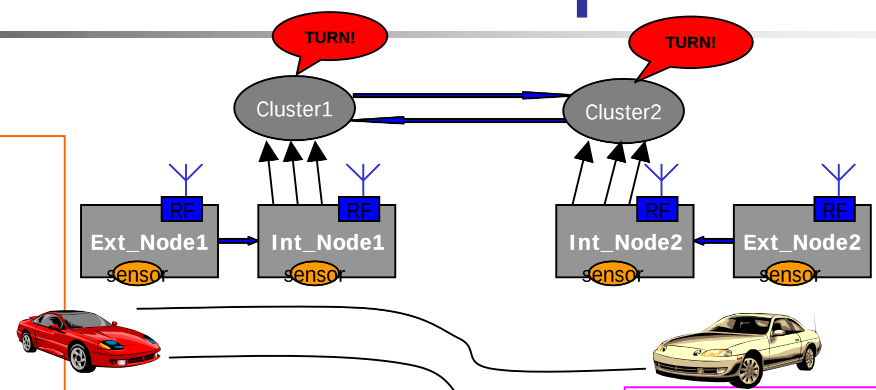
Distributed wireless sensor nodes detect the coming cars and their directions, communicate with neighbor nodes, and control the cars to avoid collision



High-Level Description

module Cluster:

```
loop
  await FROM_NC;
  await E;
                  emit CAR_TURN
each L
||
loop
   await E;
                  emit TO_NC
each L
||
loop
   await T;
                  emit CAR_TURN;
                  emit TO_NC
each L
```



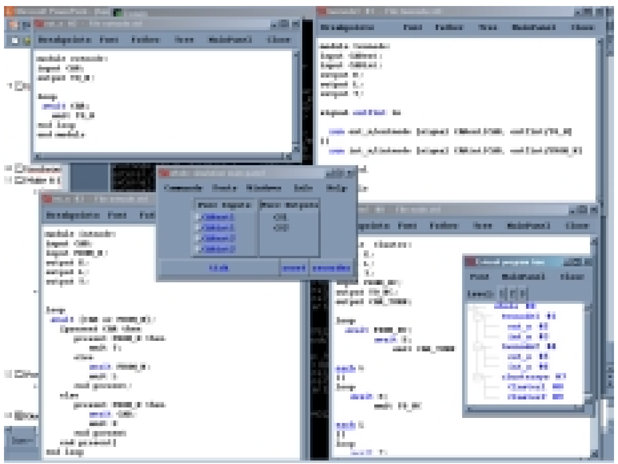
module intnode:

```
loop
   await [CAR or FROM_N];
   present CAR then
       present FROM_N then
                   emit T
       else
                   [await FROM_N;
                    emit L]
      end present
   else
      present FROM_N then
                   [await CAR;
                    emit E]
      end present
   end present
end loop
```

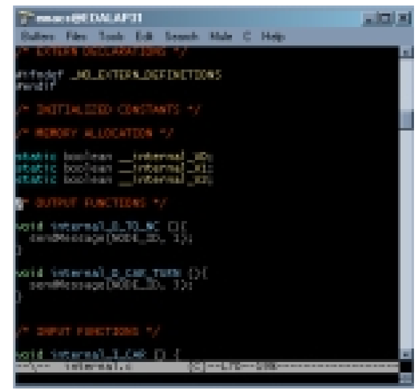
module extnode:

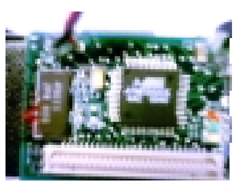
```
Loop
    await CAR;
                   emit TO_N
end loop
```

The real World



Esterel –A <modulename>.strl



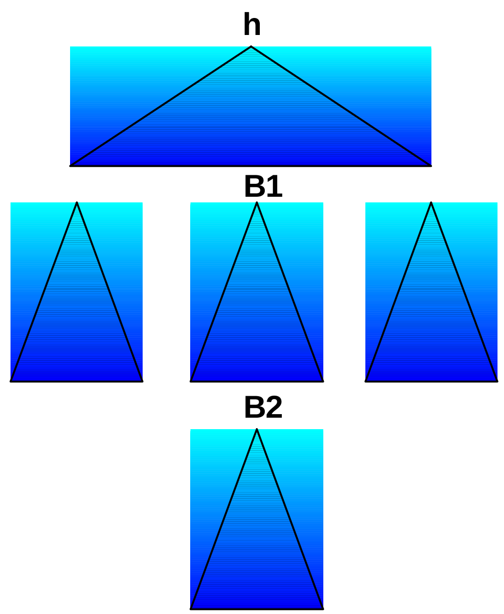


Make it Endochronous

- Simple solution: add signaling
  - For each signal add a boolean flag which indicates presence/absence
  - During the execution the functions will wait for all the flags and then will react
- Very expensive in general:
  - If (flag==T) { set_input;set_arrived}
  - If (all_flag) {react;reset_arrived}

Possible Optimization

- Clock analysis: reduce the number of signals in the protocol



$$B1 \mapsto B2$$

Our Case

```
module extnode:
Loop
    await CAR;
                emit TO_N
end loop
```

This is endochronous:

$$V(1)=\{CAR\}\mapsto V(2)=\{CAR, TO\_N\}$$

```
module intnode:
loop
   await [CAR or FROM_N];
   present CAR then
       present FROM_N then
                   emit T
       else
                   [await FROM_N;
                    emit L]
      end present
   else
      present FROM_N then
                   [await CAR;
                    emit E]
      end present
   end present
end loop
```

This is not endochronous since CAR and FORM_N are not related:

```
if (messagePayload_ptr->sourceNodeID == DUMB_NODE) {
internal_I_FROM_N()}
else if (messagePayload_ptr->sourceNodeID ==
OTHER_SMART_NODE) {
internal_I_FROM_NC()}
internal()
```

Some discussion

- External node:
  - Code size overhead 7%
- Internal node
  - Code size overhead 18%
- The external node is already endochronous
- Internal node needs some extra signaling

Demo

- Assumption:
  - Distance_TwoCarsInSameDirection>Distance(ext_node1, ext_node2)
- Cases:
  - Two cars are entering this region
  - One is entering, the other is leaving

Summary

- Very fast and easy design flow
- The overhead is acceptable
- There is enough room for optimization
- It makes sense if a centralized description of the algorithm makes sense

Credits

- Albert Benveniste for inspiring our work
- Prof. Culler and his students for tinyOS support and mote platform development
- Sarah Bergbreiter & Kris Pister for building the robot platform