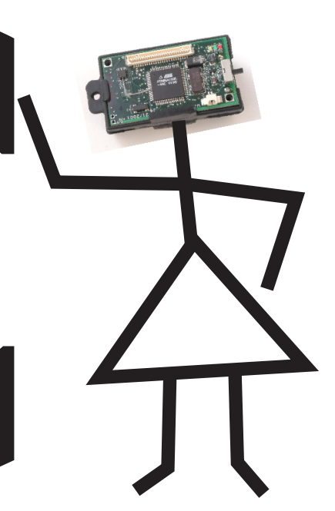
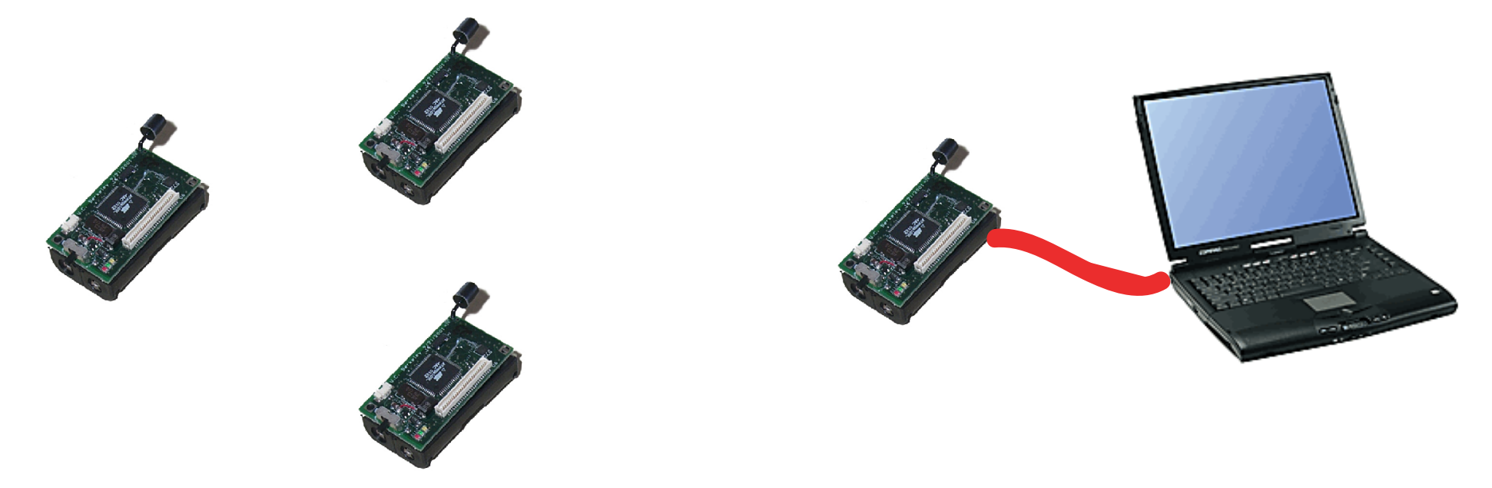
# galsC A Language for Event-Driven Embedded Systems



**Elaine Cheong**
University of California, Berkeley
**Jie Liu**
Palo Alto Research Center

## Motivation

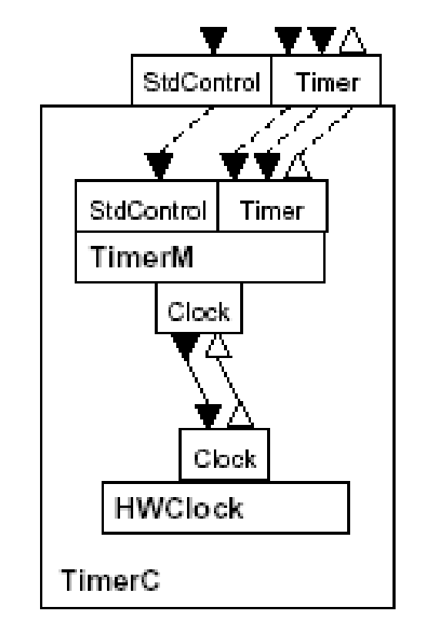
Ad-hoc sensor networks are flexible and easy to deploy...



...but their ad-hoc, reactive nature makes them difficult to program.

TinyOS is an event-driven operating system for networked sensors. Its flat hierarchy makes it difficult to predict the flow of control between synchronous and asynchronous parts of the application.

### nesC / TinyOS Architecture



```
configuration TimerC {
  provides {
    interface StdControl;
    interface Timer;
  }
}
implementation {
  components TimerM, HWClock;

  StdControl = TimerM.StdControl;
  Timer = TimerM.Timer;

  TimerM.Clk -> HWClock.Clock;
}
```

galsC extends nesC (the language used to specify TinyOS programs) to provide a GALS programming model that separates the rates of control of the system — the reactive part and the computational part — via asynchrony. galsC programs have minimal overhead, are easy to write and allow reuse of software components.

## Programming Model

Our programming model, TinyGALS, is based on the GALS (Globally Asynchronous, Locally Synchronous) model of computation.

We also include TinyGUYS as an optimization for protected, quick access to global data.

```
application SenseToLeds {
  parameter {
    uint16_t sensorData;
  } implementation {
    actor Sense, Display;

    sensorData = Sense.sensorData;
    sensorData = Display.displayData;

    Display.messageCount =[64]=>
      Sense.messageCount;

    appstart {
      // No initial tokens.
    }
  }
}
```

## Execution Model

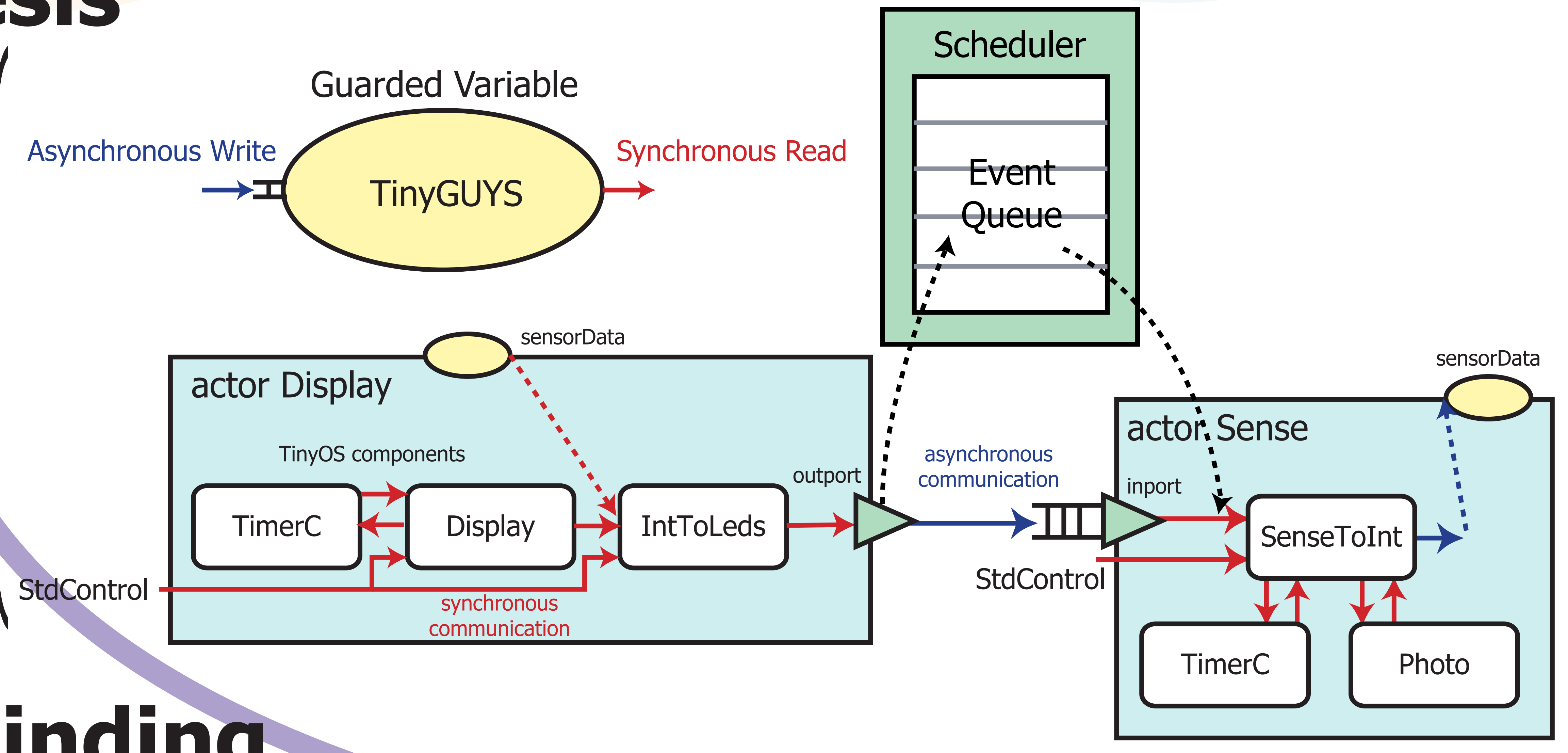
The "locally synchronous" aspect of TinyGALS is realized by method calls between components. The "globally asynchronous" aspect is realized by message passing between actors. A scheduler activates the actors to process the messages.

```
actor Sense {
  port {
    in messageCount;
  } parameter {
    uint16_t sensorData;
  } implementation {
    components SenseToInt, TimerC, Photo;

    SenseToInt.Timer ->
      TimerC.Timer[unique("Timer")];
    SenseToInt.TimerControl -> TimerC;
    SenseToInt.ADC -> Photo;
    SenseToInt.ADCControl -> Photo;
    SenseToInt.IntOutput.output -> sensorData;
    messageCount ->
      SenseToInt.IntOutput.outputComplete;

    actorControl {
      SenseToInt.StdControl;
    }
  }
}
```

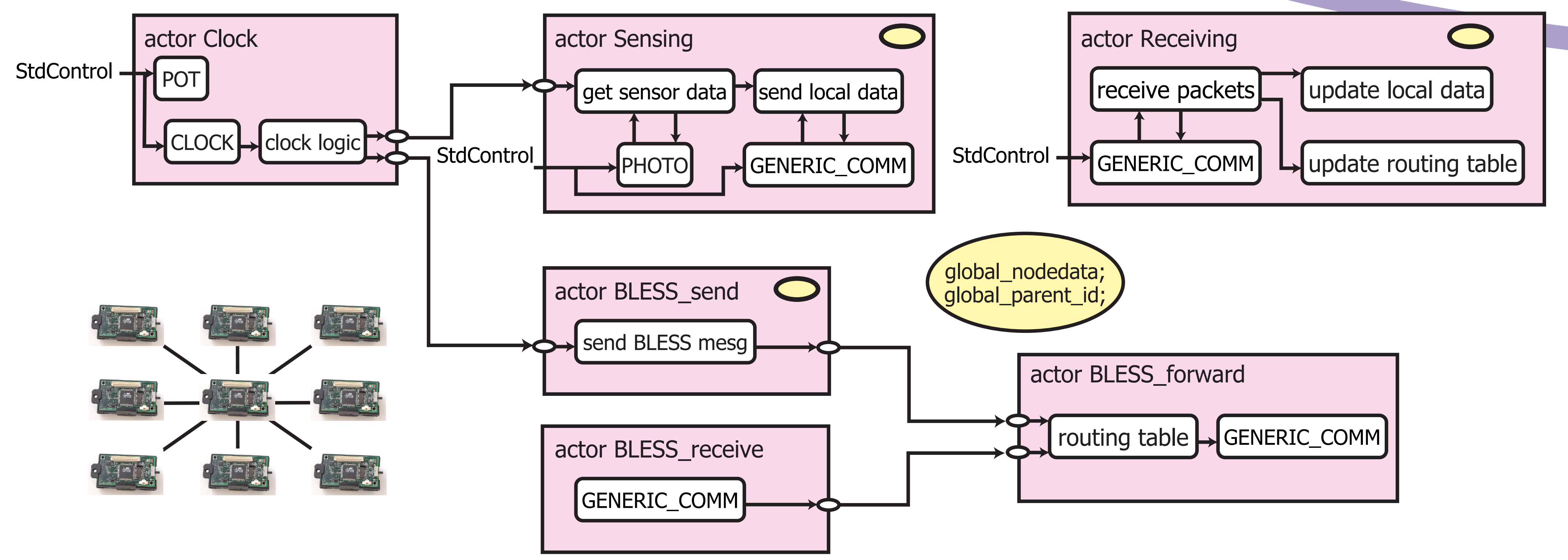
## Software Synthesis

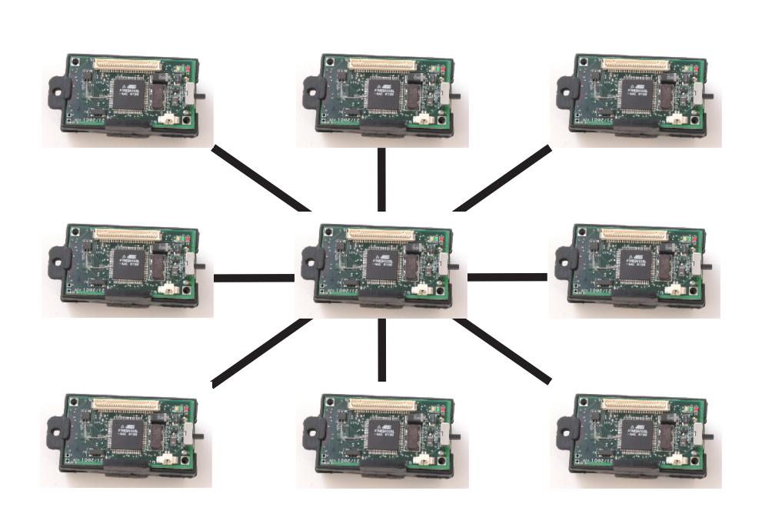
After the programmer specifies the actors comprising the application and the connections between actor ports, the galsC compiler generates code for the scheduling of and communication between actors.

The galsC compiler also generates access functions for TinyGUYS, access calls, and buffers for the data.



## Example: Peak Finding





## Future Work

- Generate from Ptolemy domain (compare to CI).
- Blocking write: retry when queue is full.
- Priority scheduling algorithm with ☐queue insertions.
- Determine queue sizes automatically.
- Run-time reconfigurability of actors.
- Heterarchy: distributed multi-tasking.

Supported by an Intel Open Collaborative Research Fellowship, Palo Alto Research Center, and UC Berkeley CHESS (Center for Hybrid and Embedded Software Systems).