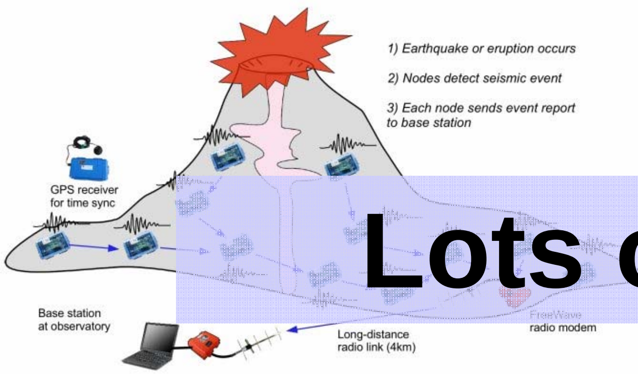
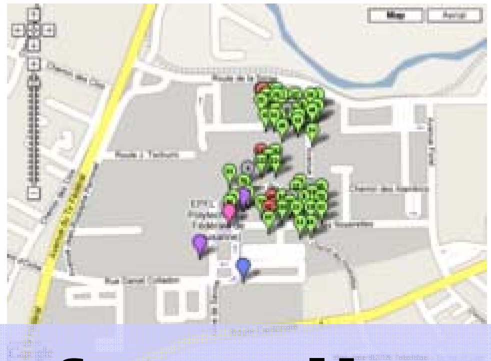
Programming Sensor Networks: A Tale of Two Perspectives

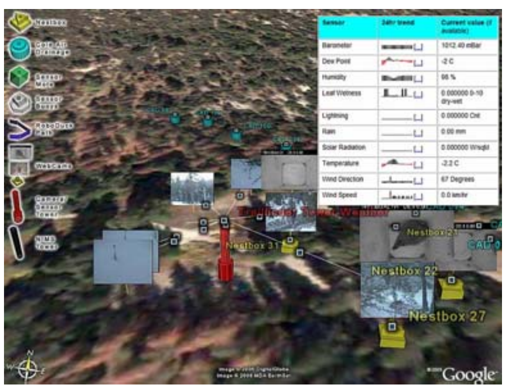
Ramesh Govindan
ramesh@usc.edu
Embedded Networks Laboratory
http://enl.usc.edu

Wireless Sensing: Applications





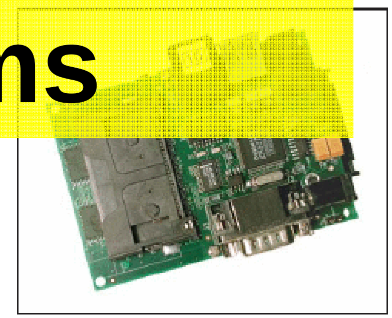
Lots of applications



Wireless Sensing: Platforms

Motes: 8 or 16 bit sensor devices

Lots of platforms



32-bit embedded single-board computers

Wireless Sensing Research

Collaborative Event Processing

Querying, Triggering

Programming Systems

Data-centric Routing

Aggregation and Compression

Data-centric Storage

Lots of research!

Collaborative Signal Processing

Localization

Time Synchronization

Medium Access

Calibration

Operating Systems

Processor Platforms

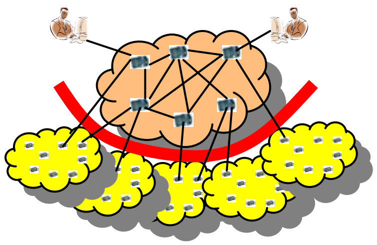
Radios

Sensors

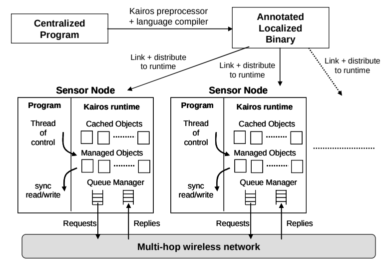
Monitoring

Security

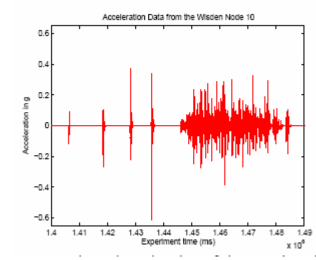
... some of it from our Lab



Architecture

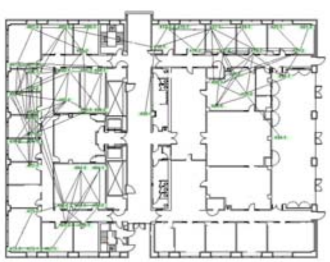


Macro-programming

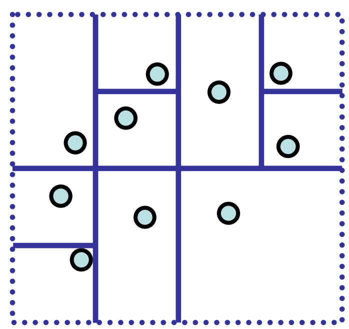


Structural Health Monitoring

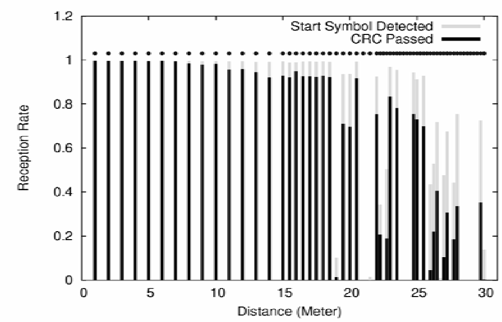
ENL
Embedded Networks Laboratory



Routing and
Data Dissemination



Data-Centric Storage



Measurements and
Testbeds

But, there is a problem!

Six pages of 158 pages
of code from a wireless structural data
acquisition system called Wisden

Programming these networks is hard!

Three Responses

OS/Middleware

Event-based programming on an OS that supports no isolation, preemption, memory management or a network stack is hard.

Therefore, we need OSes that support preemption and memory management, we need virtual machines, we need higher-level communication abstractions.

Three Responses

Networking

Tiny sensor nodes (motes) are resource-constrained, and we cannot possibly be re-programming them for every application.

Therefore, we need a *network architecture* that constrains what you can and cannot do on the motes.

Three Responses

Programming Languages

Today, we're programming sensor networks in the equivalent of assembly language.

What we need is a *macroprogramming system*, where you program the network as a whole, and hide all the complexity in the compiler and the runtime

Three Responses

OS/Middleware

The Tenet Architecture

Networking

Programming Languages

The Pleaides Macroprogramming System

The Tenet Architecture

Omprakash Gnawali, Ben Greenstein, Ki-Young Jang, August Joki, Jeongyeup Paek,
Marcos Vieira, Deborah Estrin, Ramesh Govindan, Eddie Kohler,
The TENET Architecture for Tiered Sensor Networks,
In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (Sensys),* November 2006.

The Problem

Sensor data fusion within the network

… can result in energy-efficient implementations

But implementing *collaborative* fusion on the *motes* for each application separately

… can result in fragile systems that are hard to program, debug, re-configure, and manage

We learnt this the hard way, through many trial deployments

An Aggressive Position

Why not design systems without sensor data fusion on the *motes*?

A more aggressive position: Why not design an *architecture* that prohibits collaborative data fusion on the *motes?*

Questions:

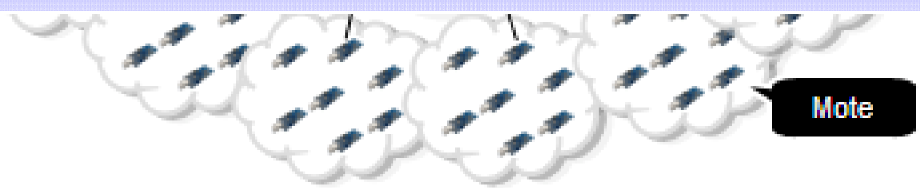
- How do we design this architecture?
- Will such an architecture perform well?

Tiered Sensor Networks

Real world deployments at,

- Great Duck Island (UCB, [Szewczyk,`04]),
- James Reserve (UCLA, [Guy,`06]),
- Exscal project (OSU, [Arora,`05]),
- …

Future large-scale sensor network deployments will be tiered



Many real-world sensor network deployments are tiered

Masters

Provide greater network capacity, larger spatial reach

Motes

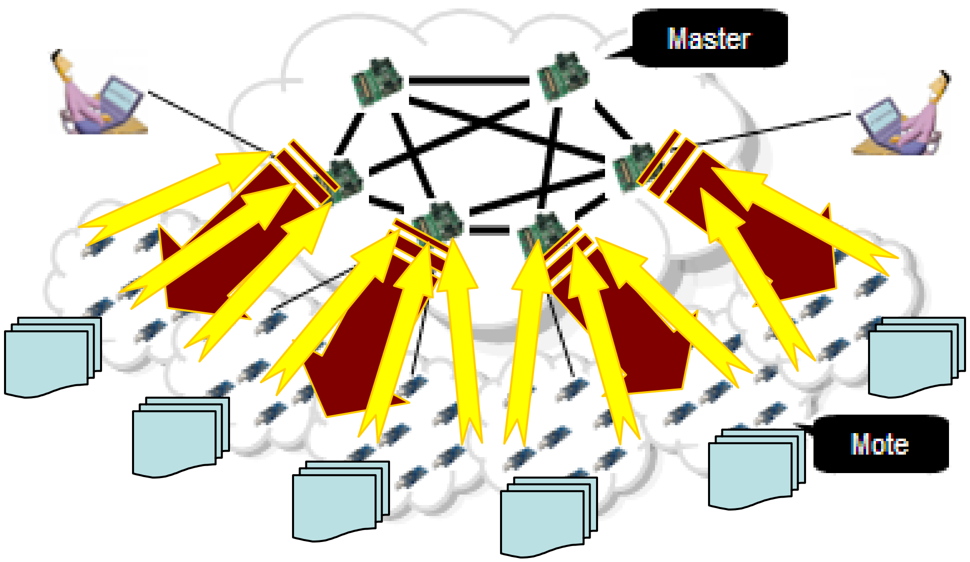
Enable flexible deployment of dense instrumentation

Tenet Principle

Multi-node data fusion functionality and multi-node application logic should be implemented only in the master tier. The cost and complexity of implementing this functionality in a fully distributed fashion on motes outweighs the performance benefits of doing so.

Aggressively use tiering to simplify system !

Tenet Architecture



Masters control motes

Applications run on masters, and masters task motes

Motes process data,

and may return responses

No multi-node fusion at the mote tier

What do we gain ?

Simplifies application development

Application writers do not need to write or debug embedded code for the motes

- Applications run on less-constrained masters

What do we gain ?

Enables significant code re-use across applications

Simple, generic, and re-usable mote tier

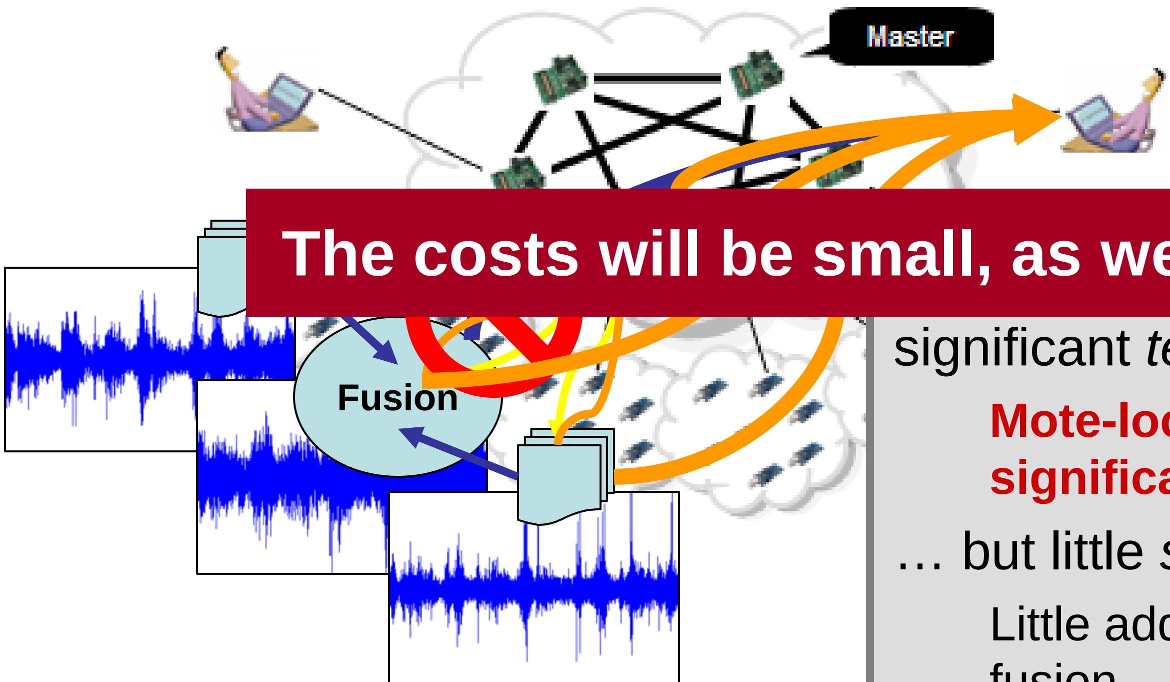
- Multiple applications can run concurrently with simplified mote functionality

Robust and scalable network subsystem

- Networking functionality is generic enough to support various types of applications

Challenges

Communication over longer hops?



The costs will be small, as we shall see…

s a significant *temporal* correlation

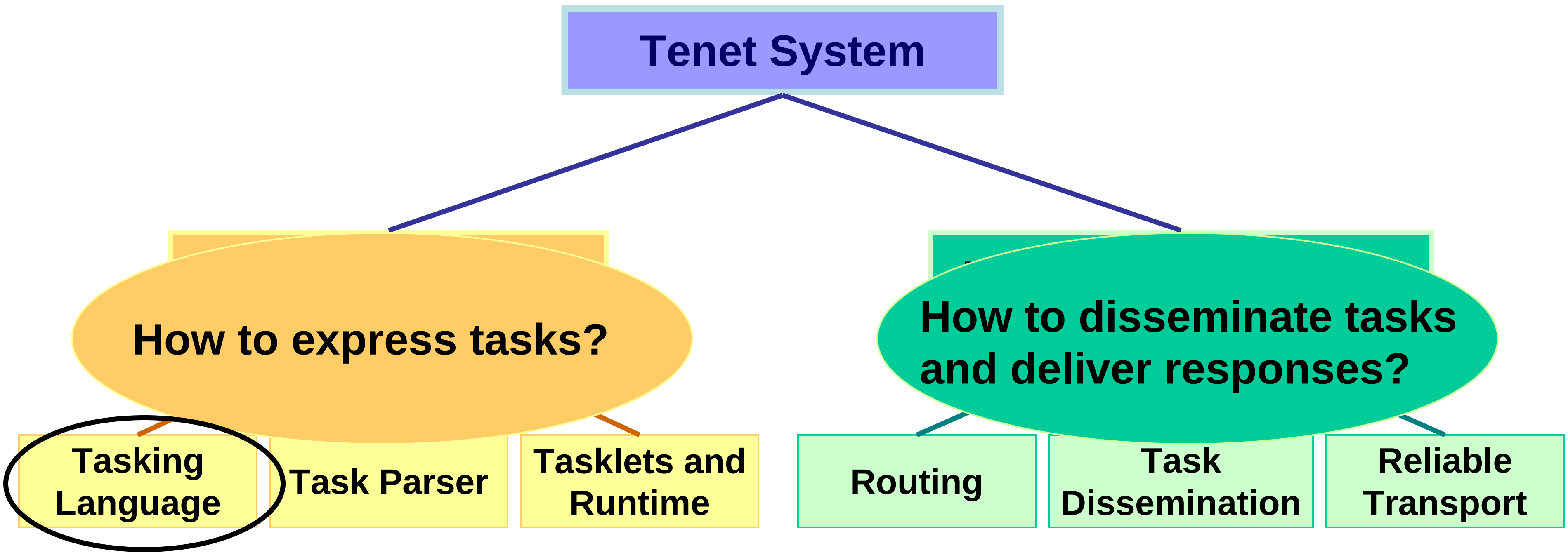
- **Mote-local processing can achieve significant compression**

… but little *spatial* correlation

- Little additional gains from mote tier fusion

Mote-local processing provides most of the aggregation benefits.

System Overview



Tasking Language

- Linear data-flow language allowing flexible composition of ***tasklets***
  - A tasklet specifies an elementary sensing, actuation, or data processing action
  - Tasklets can have several parameters, hence flexible
  - Tasklets can be ***composed*** to form a task
    - **Sample(500ms, REPEAT, ADC0, LIGHT) → Send()**
- No loops, branches: eases construction and analysis
  - Not Turing-complete: aggressively simple, but supports wide range of applications

Data-flow style language natural for sensor data processing

Task Composition

CntToLedsAndRfm

Wait → Count → Lights → Send

SenseToRfm

Sample → Send

With time-stamp and seq. number

Sample → StampTime → Count → Send

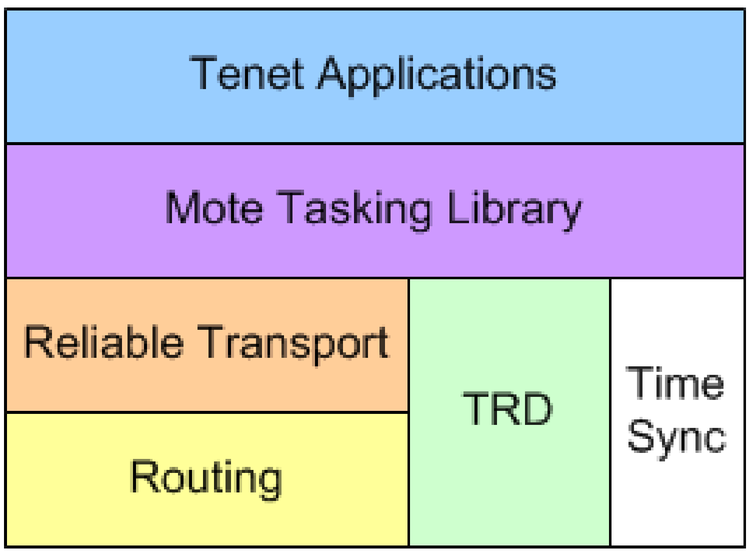
Get memory status for node 10

Address → NEQ(10) → DeleteIf → MemStats → Send

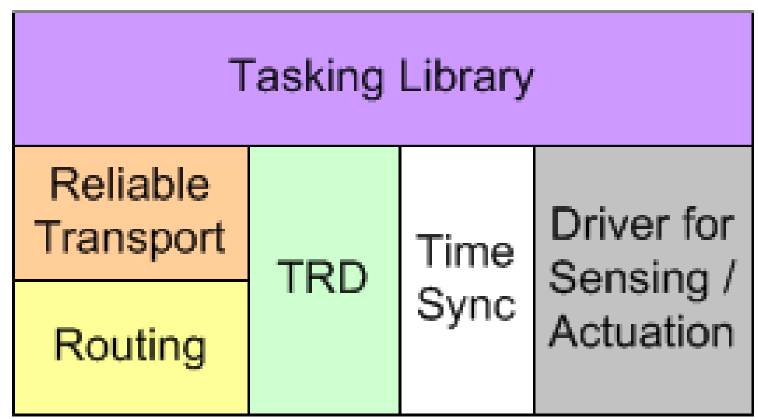
If sample value is above 50, send sample data, node-id and time-stamp

Sample → LT(50) → DeleteATaskIf → Address → StampTime → Send

The Tenet Stack



Stack on a Master Node



Stack on a Mote-class device

Application Case Study: PEG

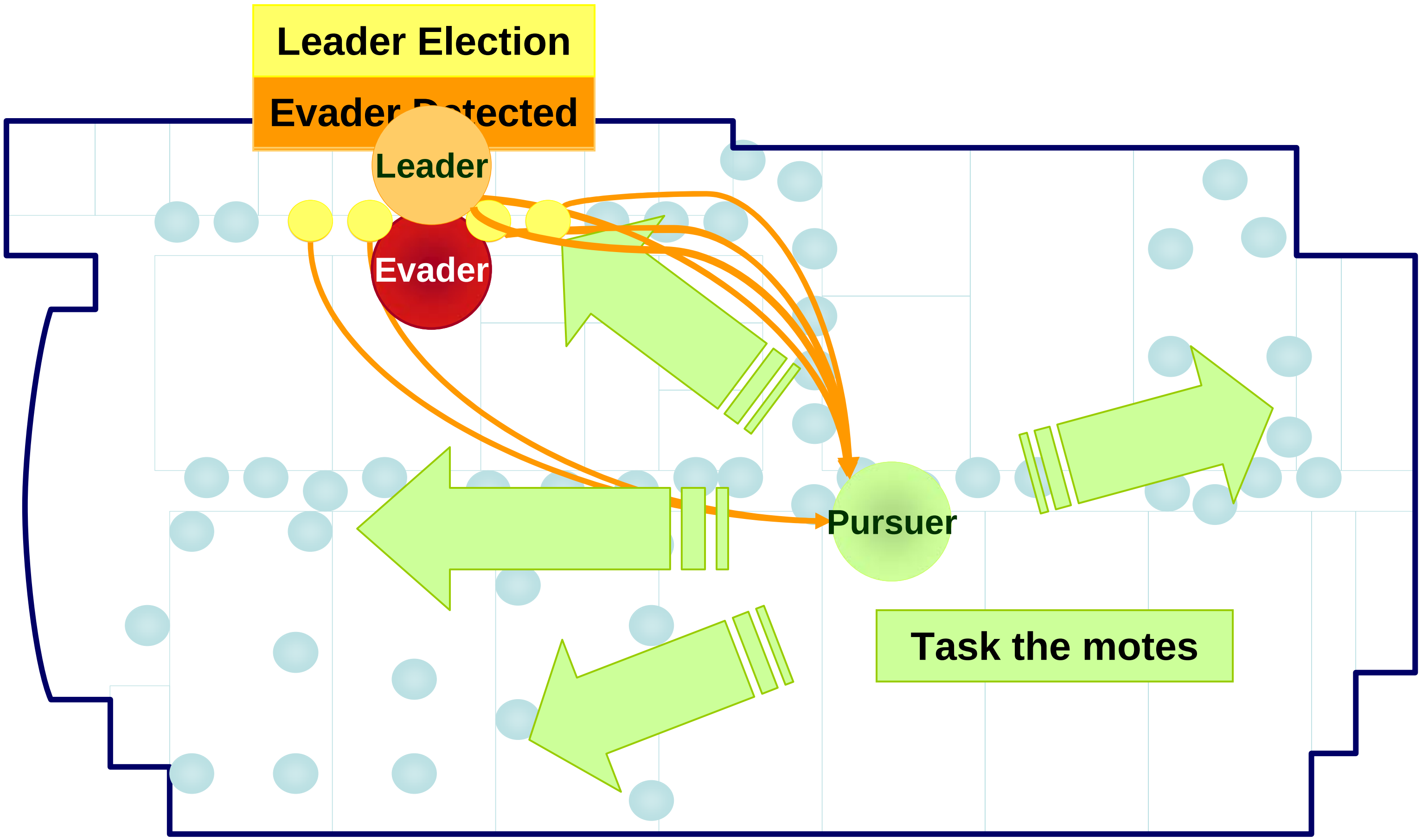
Goal

- Compare performance with an implementation that performs in-mote multi-node fusion

Pursuit-Evasion Game

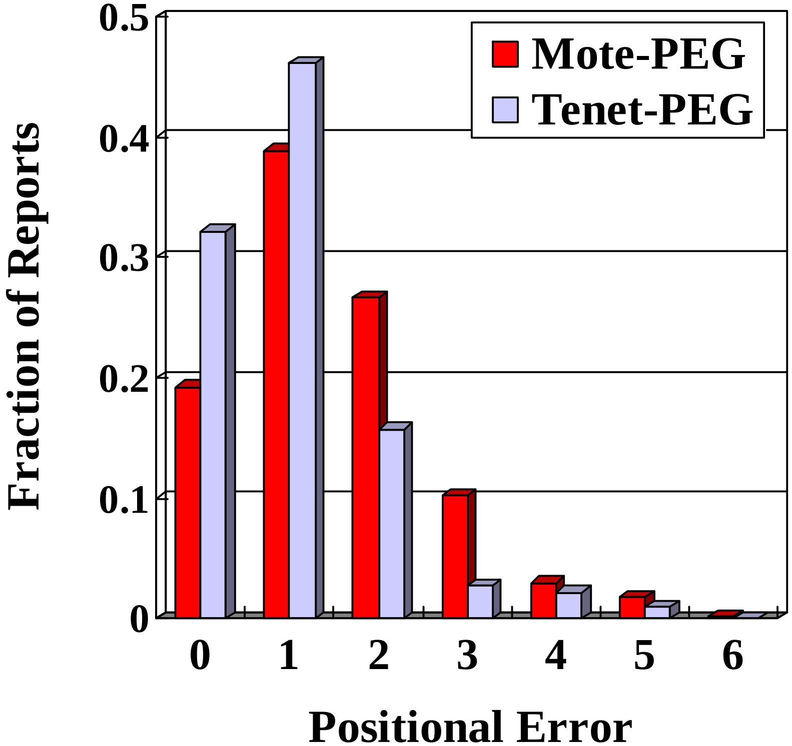
- Pursuers (robots) collectively determine the location of evaders, and try to corral them

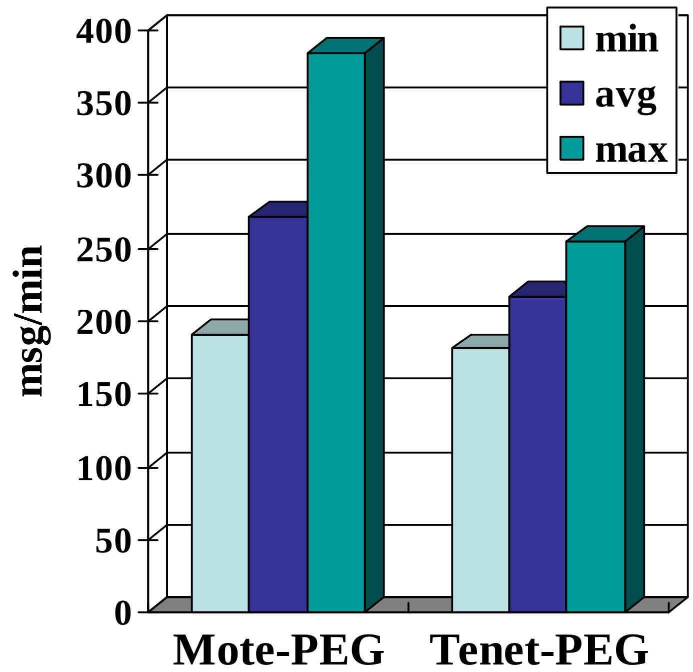
Mote-PEG vs. Tenet-PEG



Tenet-PEG

PEG Results

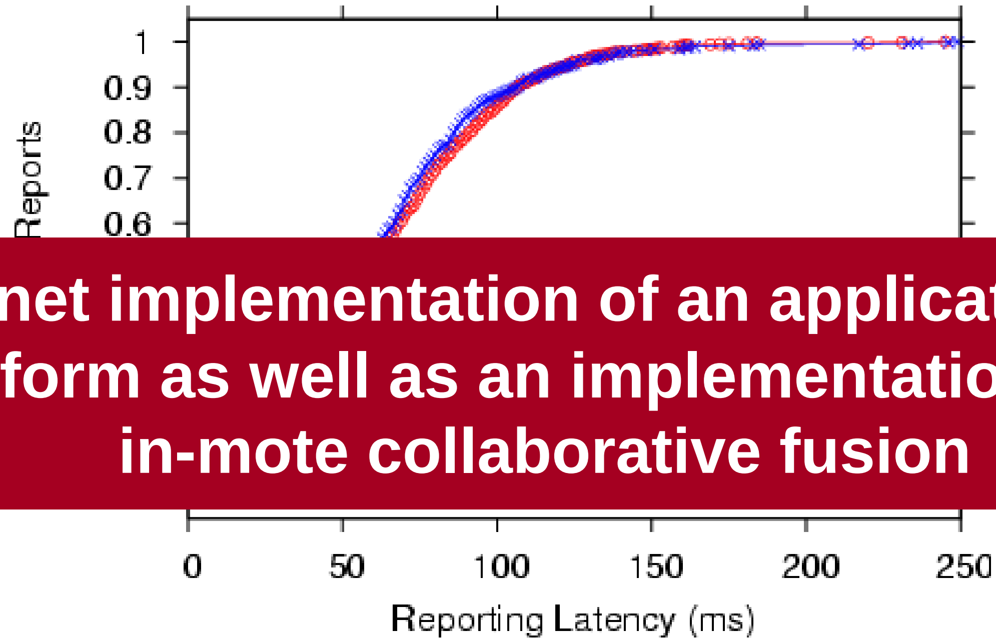




Comparable positional estimate error

Comparable reporting message overhead

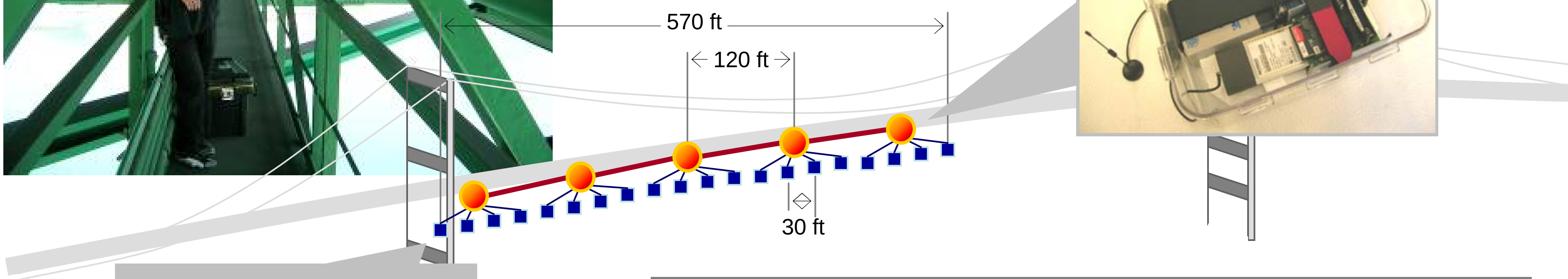
PEG Results



A Tenet implementation of an application can perform as well as an implementation with in-mote collaborative fusion

Latency is nearly identical

Real-world Tenet deployment on Vincent Thomas Bridge

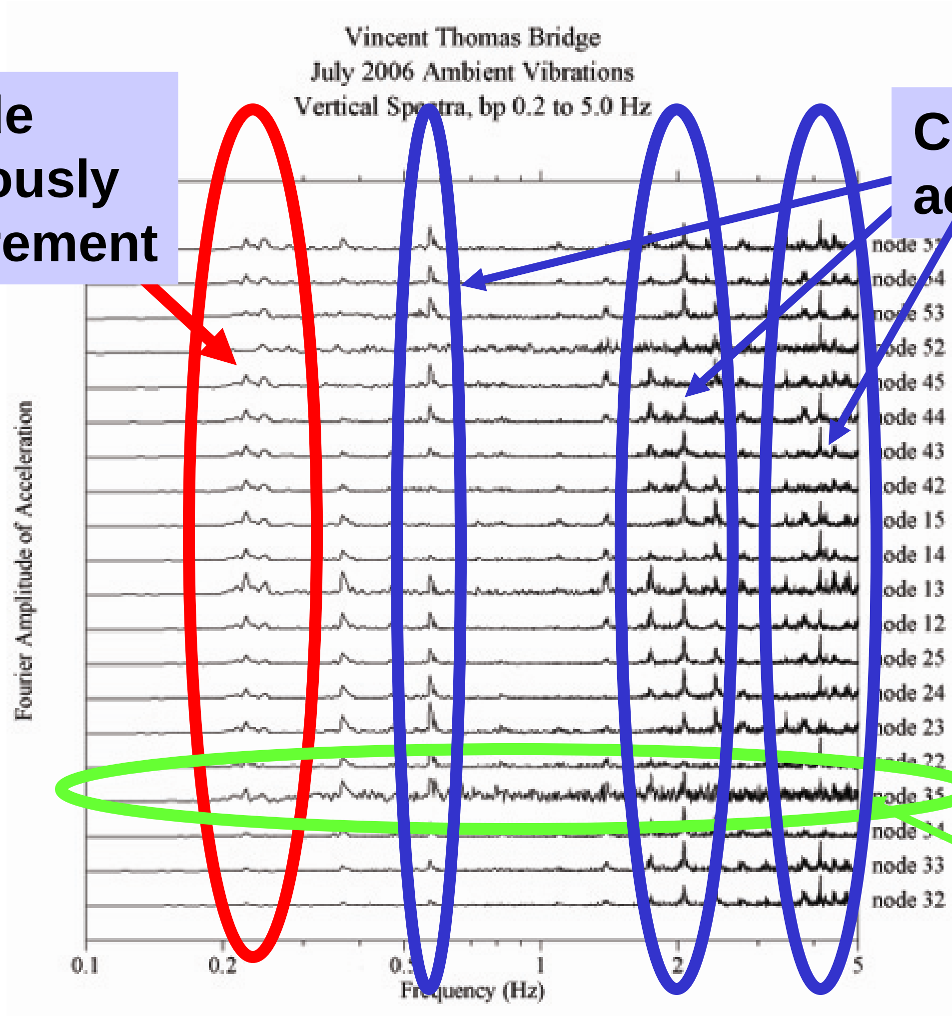


Ran successfully for 24 hours
100% reliable data delivery
Deployment time: 2.5 hours
Total sensor data received: 860 MB

Interesting Observations

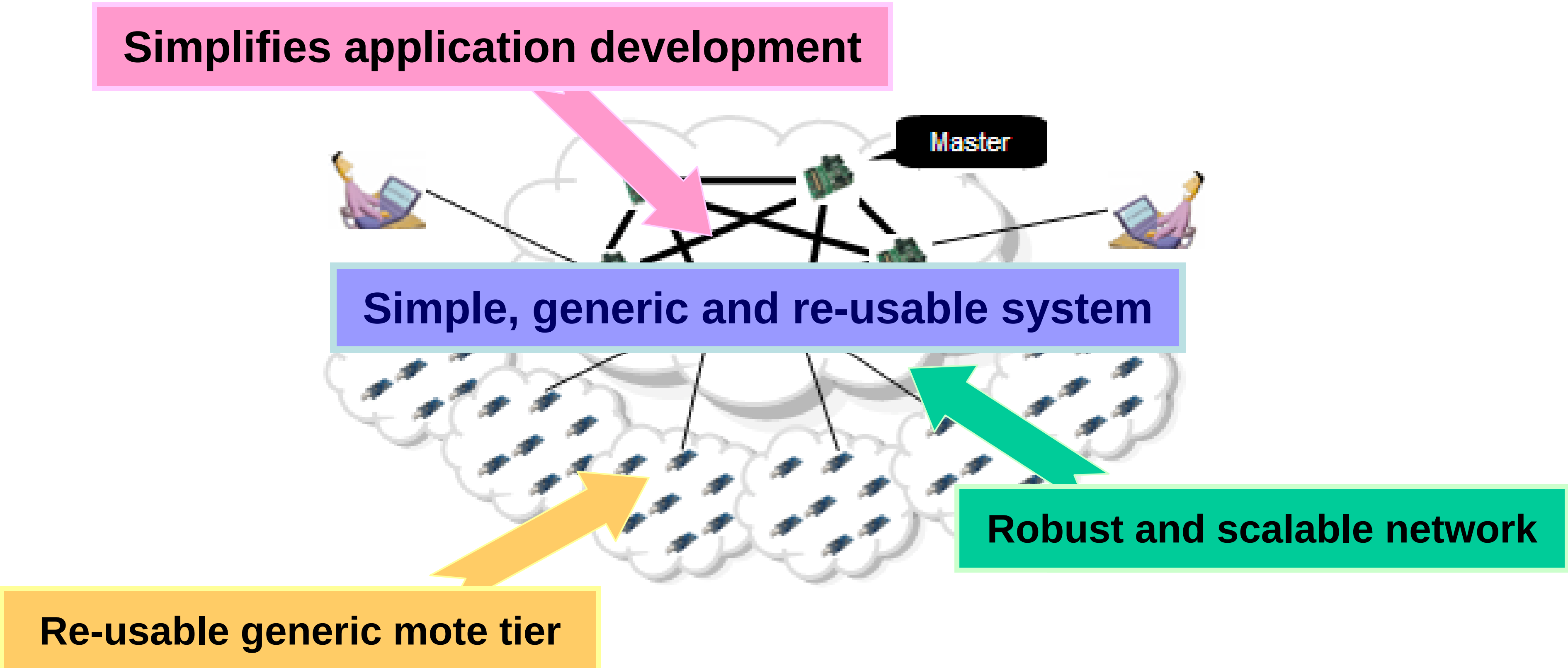
Fundamental mode agrees with previously published measurement

Consistent modes across sensors



Faulty sensor!

Summary


Simplifies application development
Master
Simple, generic and re-usable system
Robust and scalable network
Re-usable generic mote tier

Software Available

Master tier

- **Cygwin**
- **Linux Fedora Core 3**
- **Stargate**
- **MacOS X Tiger**

http://tenet.usc.edu

Mote tier

- **Tmote Sky**
- **MicaZ**
- **Maxfor**
- **Mica2**
- **Imote-2 (in progress)**

The Pleaides Macroprogramming System

Nupur Kothari, Ramakrishna Gummadi, Todd Millstein, Ramesh Govindan,
Reliable and Efficient Programming Abstractions for Wireless Sensor Networks,
Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2007.

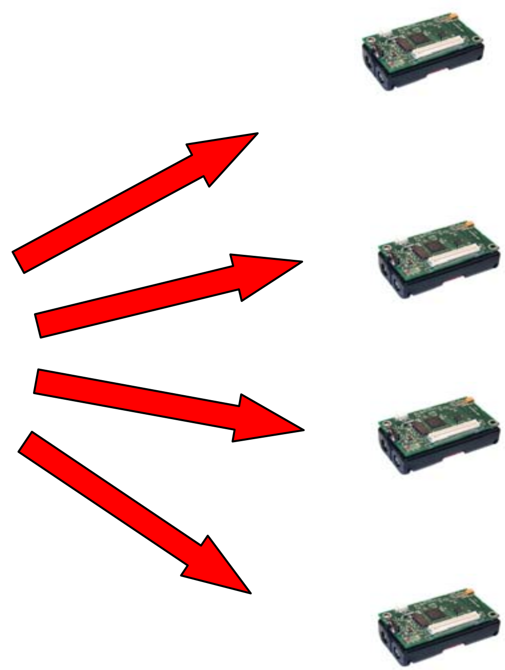
What is Macroprogramming?

Conventional sensornet programming

Node-local program written in nesC

↓

Compiled to mote binary



What is Macroprogramming?

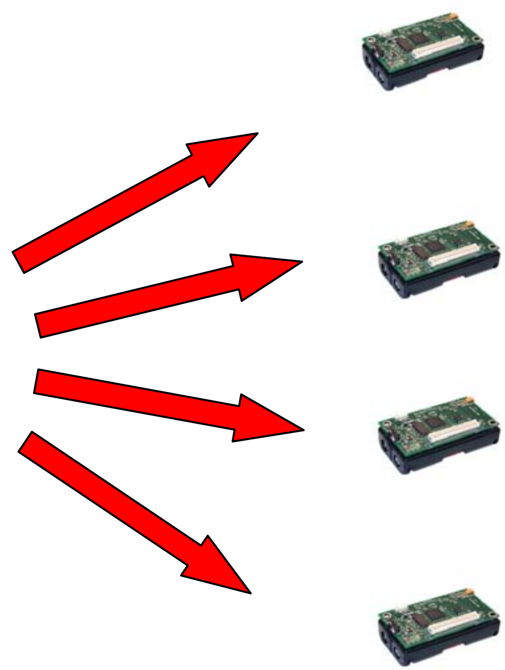
Central program that specifies application behavior

Compiler

Node-local program written in nesC

+ Runtime

Compiled to mote binary



Simplifies programming by offloading concurrency, reliability, and energy efficiency to the compiler and runtime

Change of Perspective

```
int val LOCAL;

void main() {
        node_list        all = get_available_nodes();
        int              max = 0;

        for (int i = 0, node n = get_node(all, i);
                n != -1;

                                max = val@n;
        }
}
```

Easily recognizable maximum computation loop

Pleiades: Contributions

- The Pleaides programming language
  - Centralized as opposed to node-level
- Automatic program partitioning and control-flow migration
  - Minimizes energy
- Easy-to-use and reliable concurrency primitive
  - Ensures consistency under concurrent execution
- Mote-based implementation
  - Evaluated several realistic applications

Pleiades Constructs

```
int val LOCAL;

void main() {
        nodelist        all = get_available_nodes();
        int             max = 0;

      cfor (int i = 0, node n = get_node(all, i);
                  n != -1;
                  n = get_node(all, ++i)) {
                        if (val@n > max)
                                max = val@n;
        }
}
```

Node-local variable

Central variable

List of nodes in network

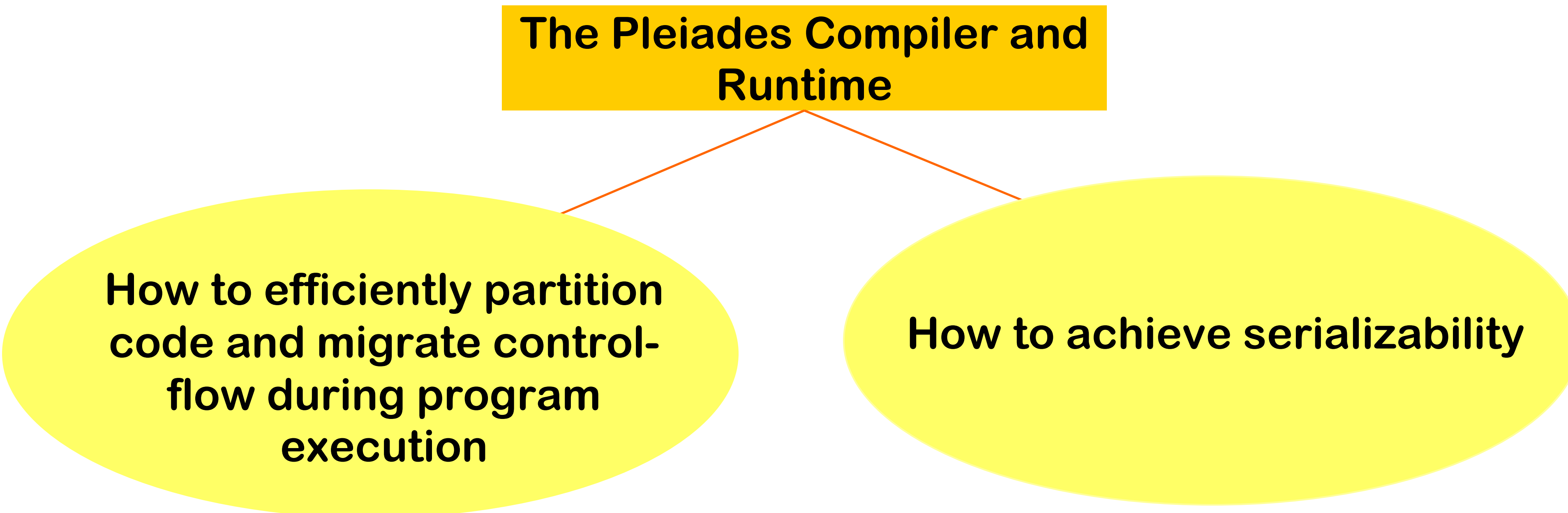
Network Node

Concurrent-for loop

s node-local variable at node

cfor execution corresponds to *some* sequential execution of the loops iterations (serializability)

Pleiades: Main Challenges



Program Execution

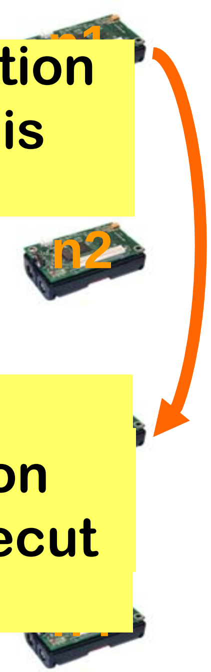
Control-flow migration as well as data movement

Nodecut

```
void main() {
  .........
  .........
}
```

Sequential Program

Uses the property that the location of variables within a nodecut is known before its execution



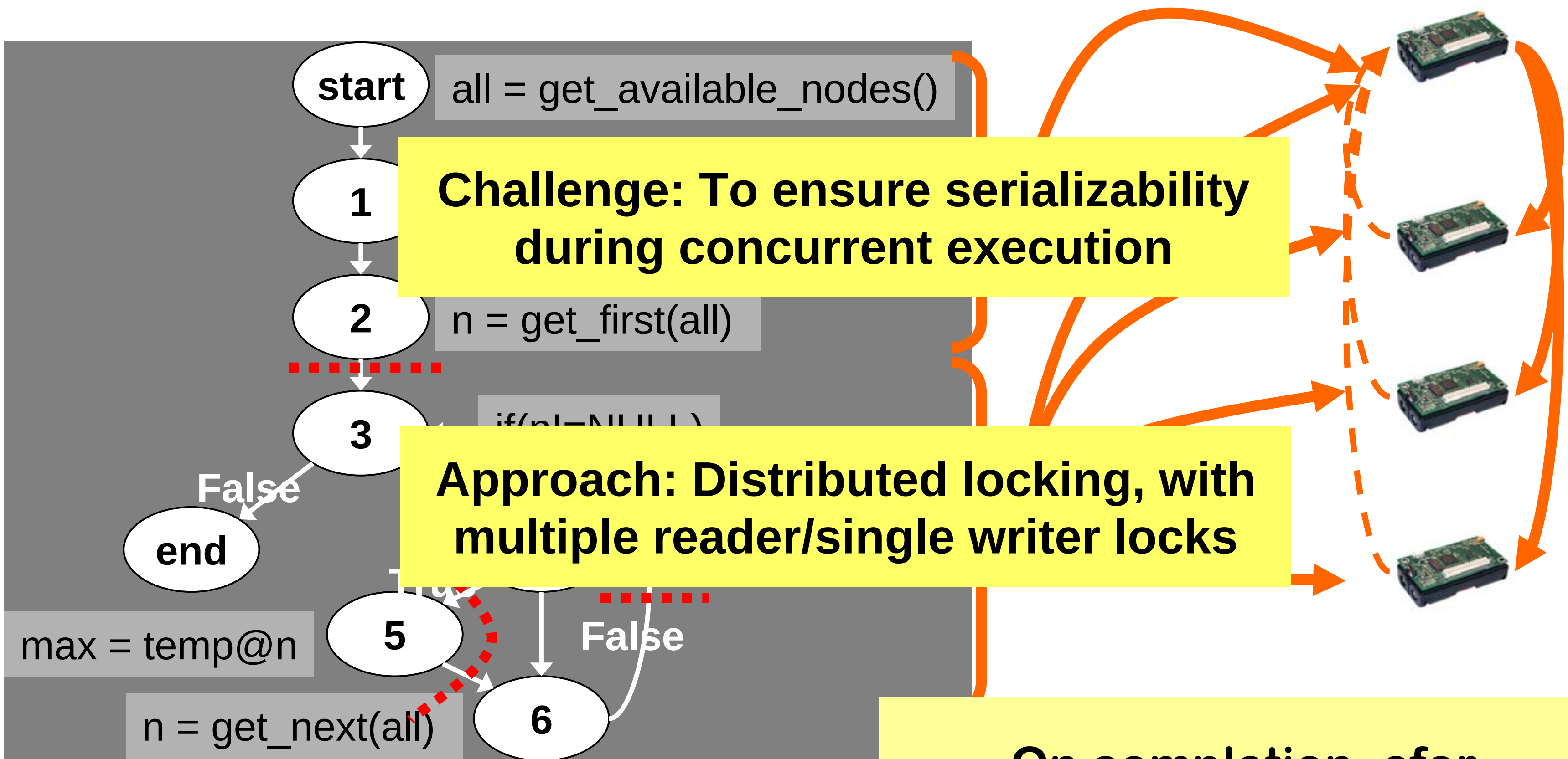
val@n1 = a;
n3 = val@n2;

Attempts to find lowest communication node, based on location of variables in the nodecut and topology information

val@n3 = b;
val@n4 = c;

Access node-local variables from nearby nodes

Cfor Execution



Challenge: To ensure serializability during concurrent execution

Approach: Distributed locking, with multiple reader/single writer locks

On completion, cfor iterations send DONE message to originating node

Implementation and Evaluation

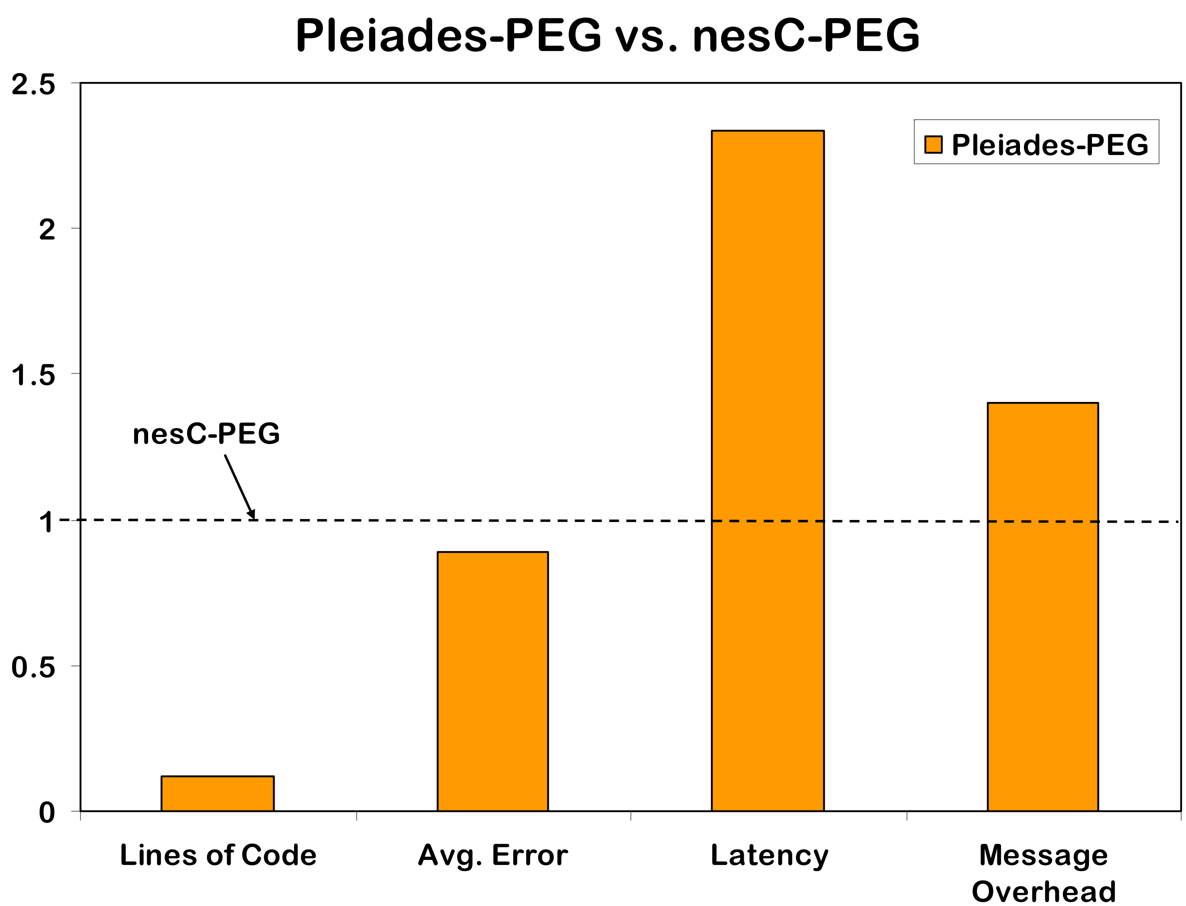
Compiler built as an extension to the CIL infrastructure for C analysis and transformation

Pleiades compiler generates nesC code

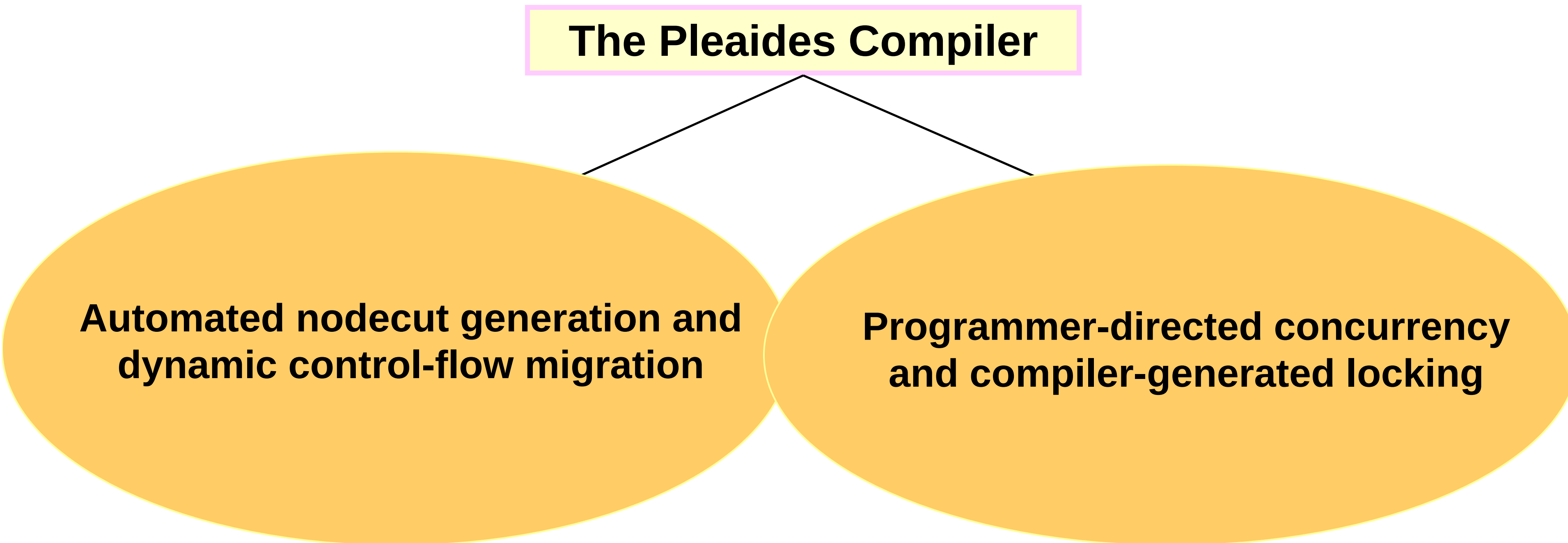
Pleiades evaluated on TelosB motes

Experience with several applications: pursuit-evasion, car parking, etc.

Pursuit-Evasion in Pleiades



Summary


The Pleaides Compiler
Automated nodecut generation and dynamic control-flow migration
Programmer-directed concurrency and compiler-generated locking

Which is Better?

Networking

The Tenet Architecture

Programming Languages

The Pleaides Macroprogramming System

Head-to-Head

	Tenet	Pleaides
Expressivity	**Low, by design**	**High**
Cuteness	**Low: Some interesting protocol design questions, but focus is on simplicity**	**High: Lots of interesting compiler optimization questions, consistency models**
Time-to-develop	**~ 3 student years**	**~ 3 student years**
Papers	**2**	**3, potential for more**

Head-to-Head

	Tenet	Pleaides
Missing Components	**Sleep scheduling, security**	**Any-to-any routing, energy management, robustness**
Maturity	**Seen two deployments, have external users**	**Code still needs much handholding**
What I believe in	√	
What I like		√

http://enl.usc.edu