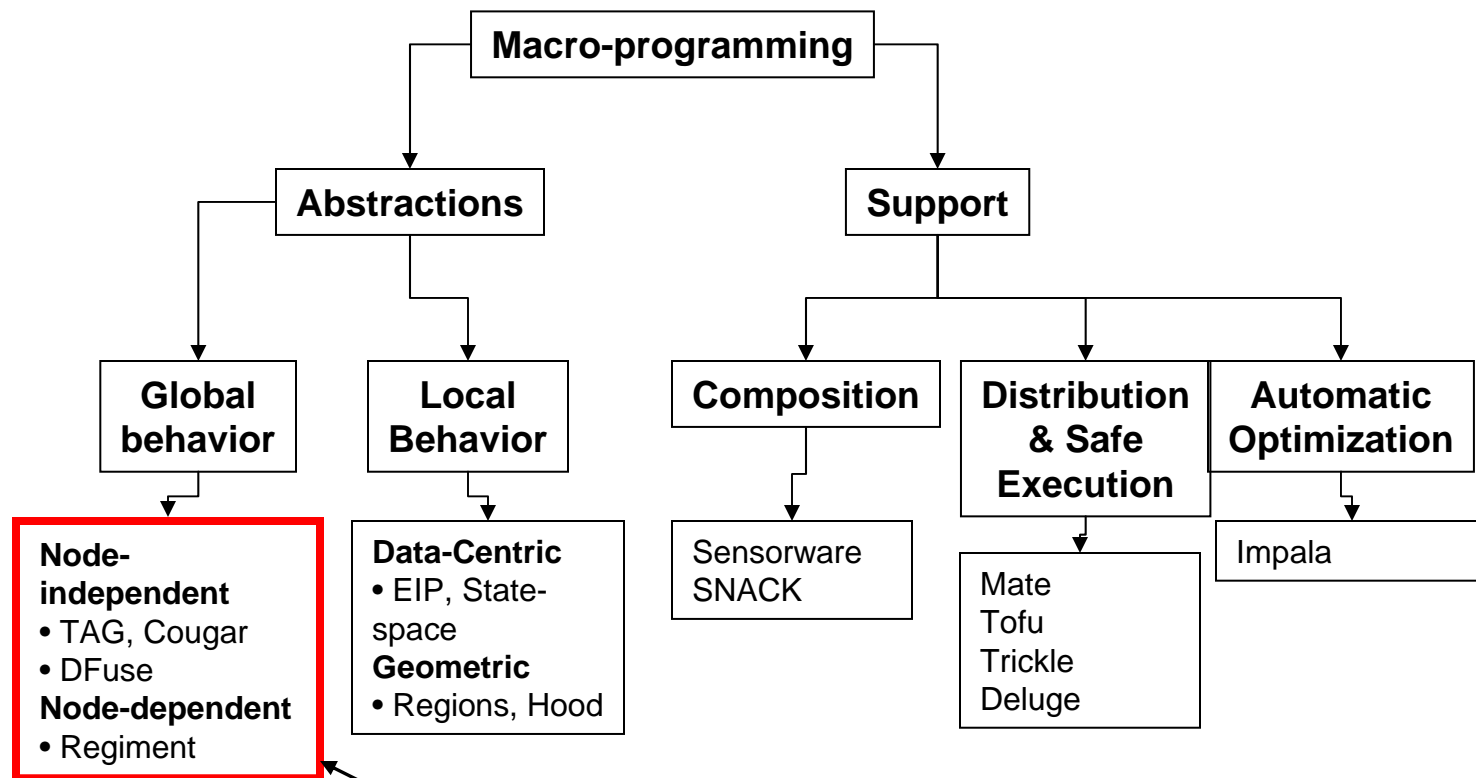


Macro-programming Wireless Sensor Networks using Kairos

Ramesh Govindan

Ramakrishna Gummadi, Todd Millstein
Nupur Kothari and Om Gnawali

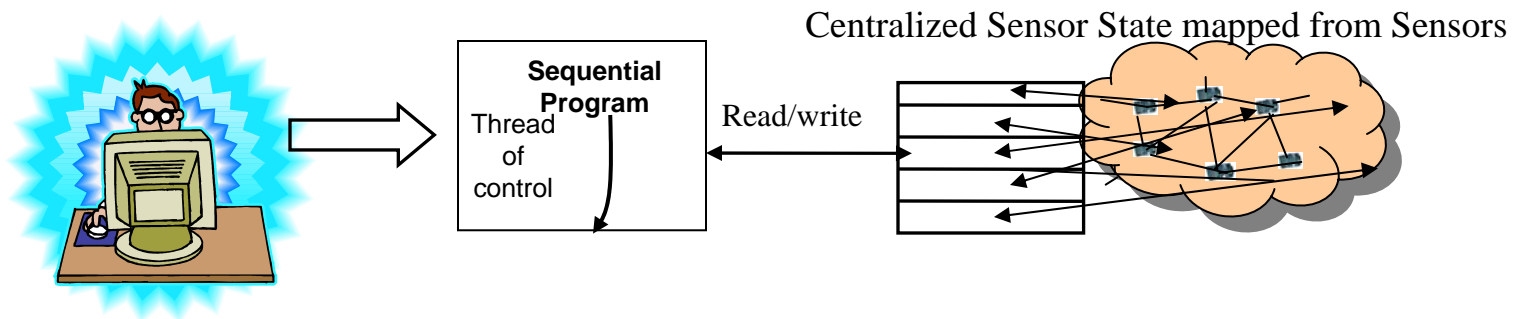
Macro-programming



How do you program a collection of nodes?

One Approach

- Allow programmers to specify the global behavior of a distributed sensor network computation as a **single sequential program** using a simple **centralized memory model**
 - Challenge: Designing the compile-time and run-time systems for synthesizing the local interactions
- Has been tried before in parallel systems
 - Energy, consistency/synchronization requirements are different



A Toy Example

- To build a shortest path tree rooted at *Root*, the centralized program must capture the global behavior:

For each node n in the network, its parent is that neighbor whose distance to *Root* is shortest.

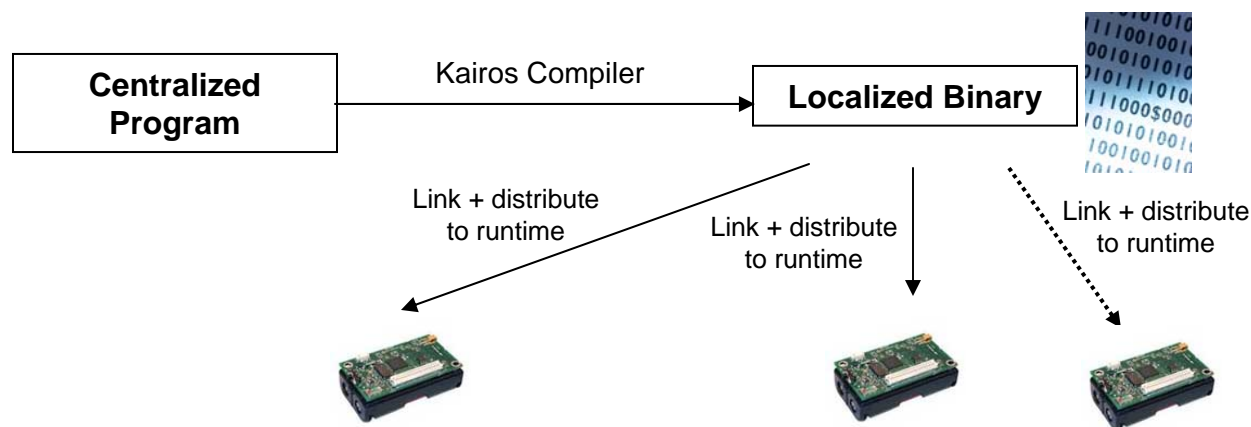
- Today...
 - At each node, run an explicitly distributed program to pick best neighbor as parent
 - Must handle low level messaging details at app level
 - Several files, > 500 lines of code (including TinyOS hooks)
 - At runtime, check for loops
 - Does not always work in practice
 - Complex, slow, and error-prone

Advantage

- Centralized sequential programs easier to specify, code, understand and debug than hand-coded distributed versions
 - Reuse “textbook” algorithms for sophisticated tasks
- Ignoring latency and energy considerations, a dumb but obviously trivial “distributed” implementation always possible, by shipping sensor nodes’ state to and from a central location

Kairos: An Instance of this Approach

Kairos translates a centralized sequential program into programs that execute on individual nodes after adding some runtime support

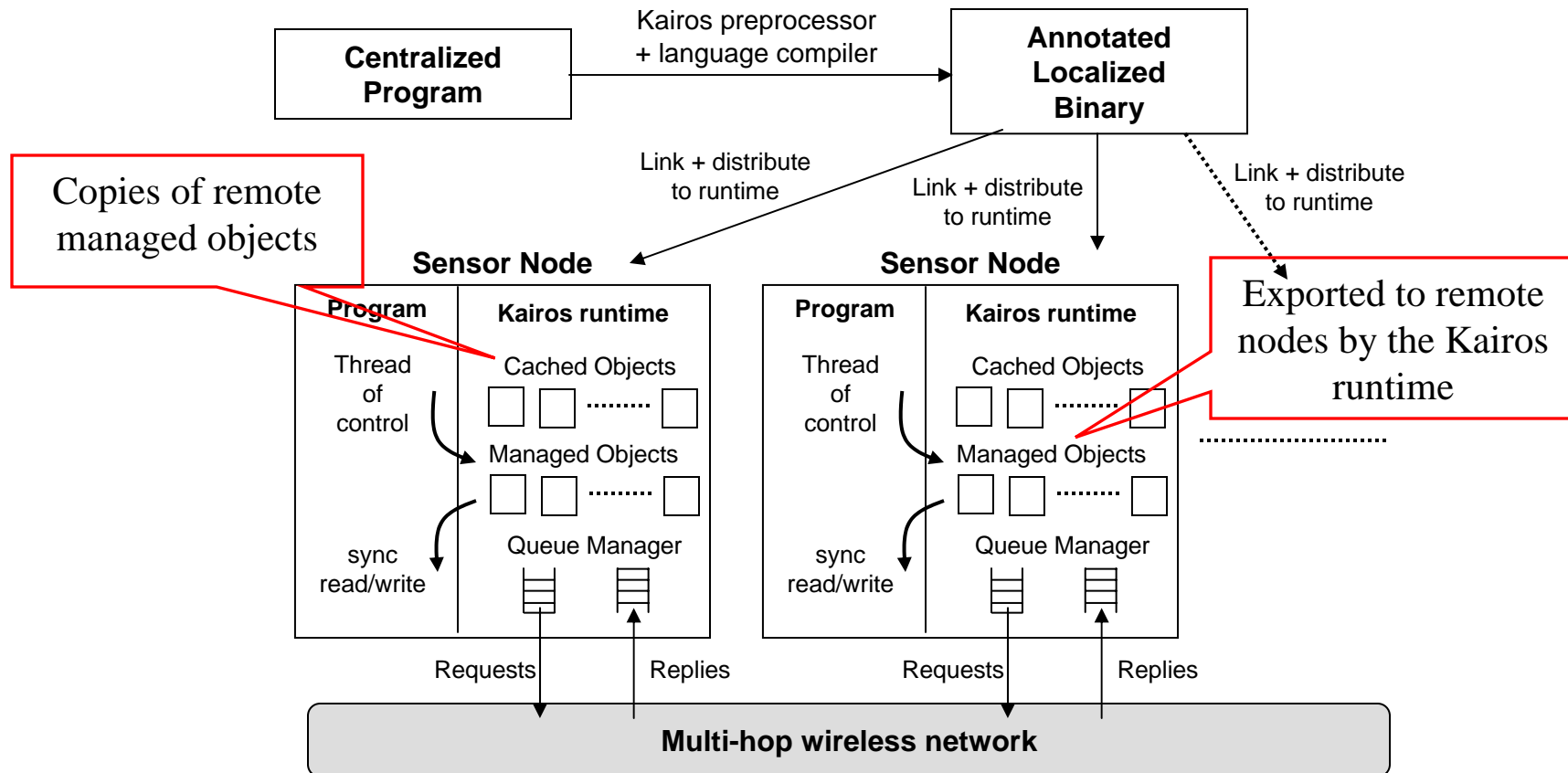


In the shortest path tree example, the generated localized binary at a node causes its runtime to retrieve the node's neighbors' data about their current distances to *Root*, and then processes this information

Kairos Features

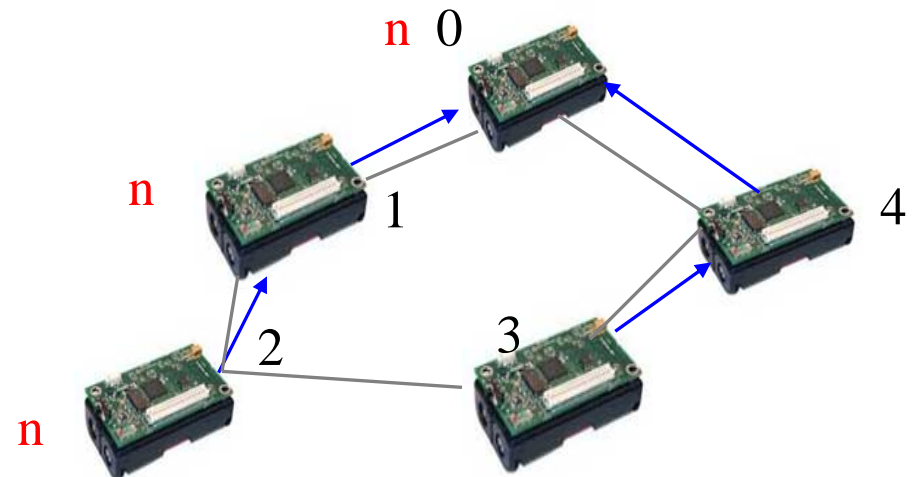
- Four constructs with which to write programs
 - `node` (a first-class datatype) and `node_list` (iterator on nodes) that facilitate topology independent programming
 - `get_neighbors()` to obtain current one-hop neighbors of a node
 - `var@node` to access data and program state of node's
 - a `time_queue` abstraction for temporal consistency
- These constructs are language-agnostic

Kairos Programming System



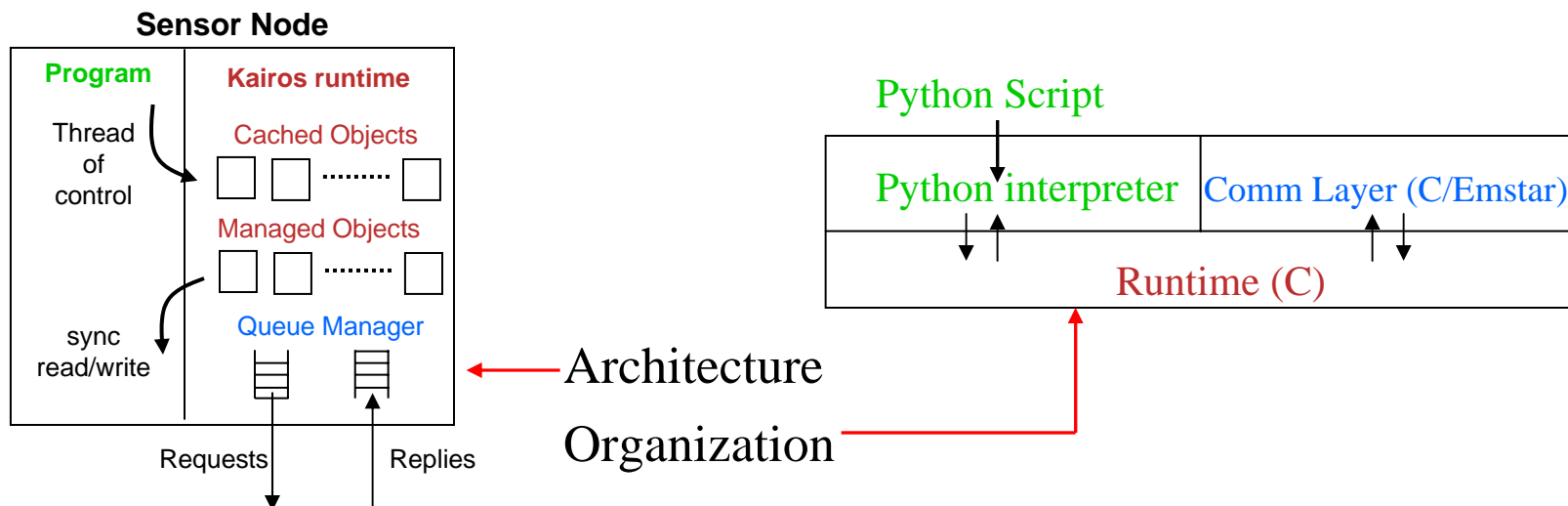
Toy Example

```
//Every sensor node has a dist_from_root integer variable and a parent node variable
void build_tree (node root=0) {
  node n, n';
  //get the current list of all available nodes in the network
  node_list avail_n=get_available_nodes();
  for (;;) {
    for (n=get_first(avail_n); n!=NULL; n=get_next(avail_n)) {
      node_list neigh_n=get_neighbors()@n;
      for (n'=get_first(neigh_n); n'!=NULL; n'=get_next(neigh_n)) {
        if (dist_from_root@n'<dist_from_root@n+1) {
          parent@n=n'; dist_from_root@n=dist_from_root@n'+1;
        }
      }
    }
  }
}
```



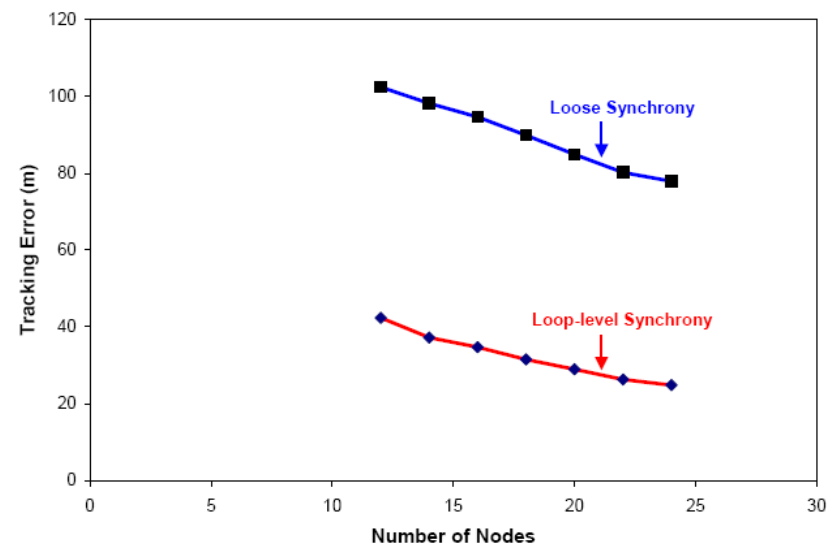
Status

- Initial implementation of language constructs in Python
 - runs on a network of Stargates
 - working on mote/nesC implementation
- Have implemented and evaluated several applications
 - object tracking
 - localization
 - median computation using q-digests



Challenges: Consistency

- Many applications can tolerate inconsistencies
 - e.g. routing protocols require *eventual* consistency
 - enables energy-efficient implementations of the shared memory model
- ... but not all
 - e.g. object tracking error can be high without a consistent computation of tracking state
 - need mechanisms that enable *temporally consistent* computations (not necessarily tightly synchronized).
- Exploring a *time queue* construct



Challenges: Fault Resilience

- Node or communication failures can render computations inconsistent
- Research questions:
 - Do there exist generic *recovery* mechanisms?
 - Can they be *automatically* applied?
 - Can the recovery actions be made *transparent* to the programmer?
 - Are the primitives we have proposed sufficient for specifying recovery actions?
- Key insight:
 - Compile-time and run-time have significant visibility into the nature and progress of the computation

Challenges: Performance Optimizations

- Topology-aware optimizations
 - Knowing the specific deployment, the run-time system can adapt communication structures for a given program
- Adaptive cached object push-pull
 - The compiler or the run-time can detect frequently written variables, and *push* the corresponding cached objects
- Hierarchy
 - The run-time can leverage hierarchical elements within the network to improve performance

Challenges: Energy Management

- Performance optimizations can improve the energy efficiency
- ... but what hooks can we provide so that the programmer has finer-grain control of energy management?

Other Interesting Ideas

- Debugging
 - run-time has visibility into the state of the computation
 - can automatically (or with minimal programmer intervention) export program state to a network management or debugging system
- Composition
 - how to support multiple Kairos programs running concurrently on a network?
 - simply extend shared memory semantics across instances?

<http://kairos.usc.edu/>