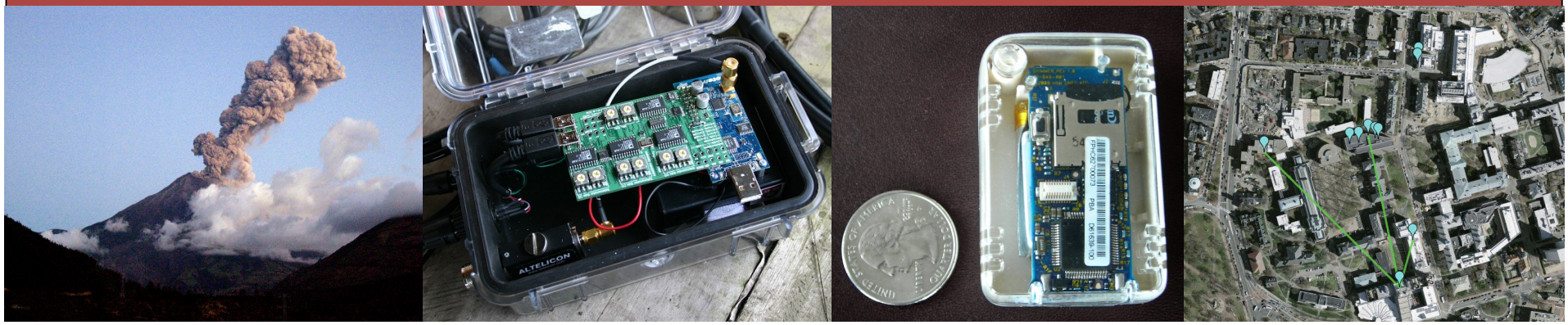
# How to Program a Macroscope



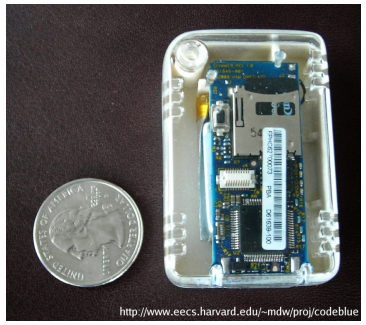

Matt Welsh

Harvard University

# Sensor Networks: A New Computing Platform

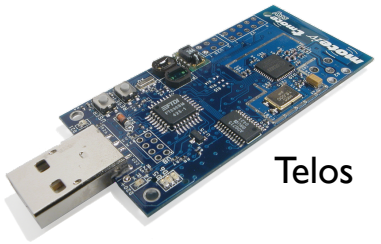
Embedded devices integrating a modest amount of computation, communication, and sensing

SHIMMER




Typical "mote" hardware

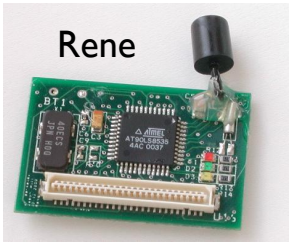
- 8 MHz CPU, 10 KB RAM, 60 KB ROM
- Low-power radio (IEEE 802.15.4 – 250 Kbps PHY rate)
- Various sensors (e.g., light, temp, humidity)
- Cost: Around $75



Telos

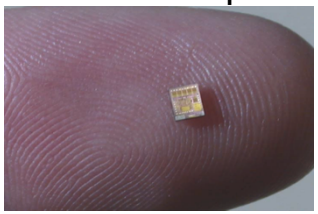
Designed for low power consumption:

- 70 mW active (CPU+radio); 6 uW sleep

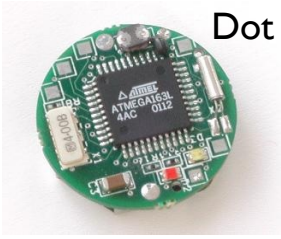


Rene

WeC



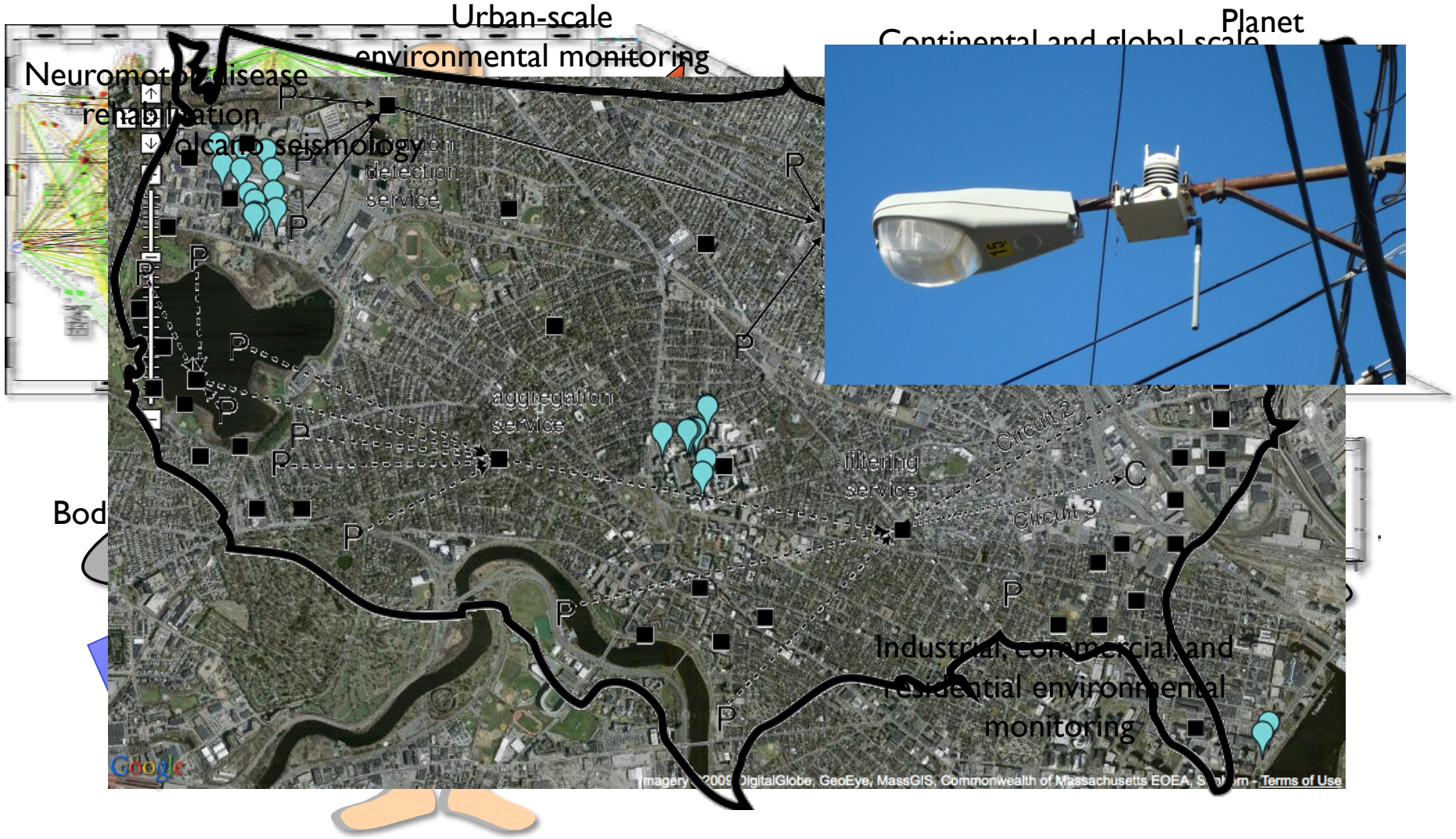
Speck



Dot

MicaZ

# Applications at many scales…



# The Macroscope

*For the first time*, sensor networks allow us to:

1) Observe the world (environment, buildings, people, etc.) at very *high spatial resolutions*;

2) Make these observations *continuously*; and

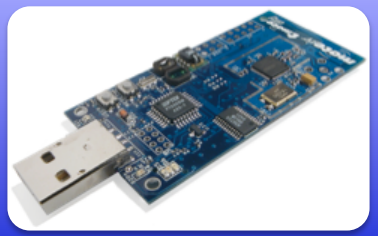
3) Collect the observations in *digital form*.

Some have referred to this concept as a "macroscope" – a scientific instrument that observes entire systems.

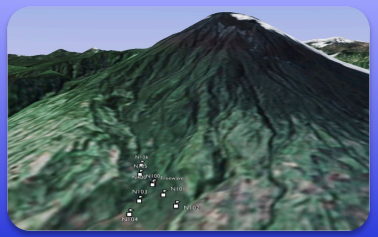



After Joël de Rosnay, *Le Macroscope. Vers une vision globale,* 1975

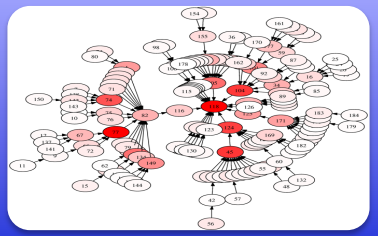
# Key Research Questions



How do we manage scarce resources at the device and network level?

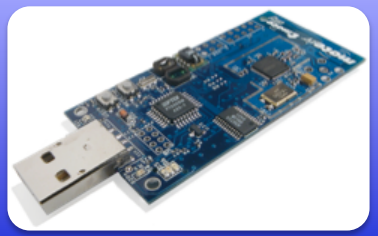


How do we ensure high data fidelity for real-world applications?

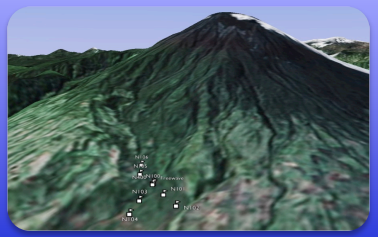


How do we program complex network behavior at a high level?

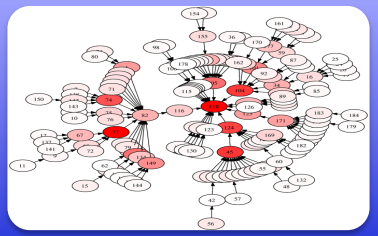
# Our Approach



New OS designs supporting *resource aware programming*:
**Pixie**, **Lance**, and **Mercury**



Extensive field research with domain scientists



New programming models and languages:
**NesC**, **Regiment**, **Flask**, **SORA**, **Abstract Regions**

# Talk Outline

Two application vignettes.

Resource Aware Programming:

- Make resources first class in the programming model.

Macroprogramming:

- Program the network, rather than the individual device.

Wrap-up and conclusions.

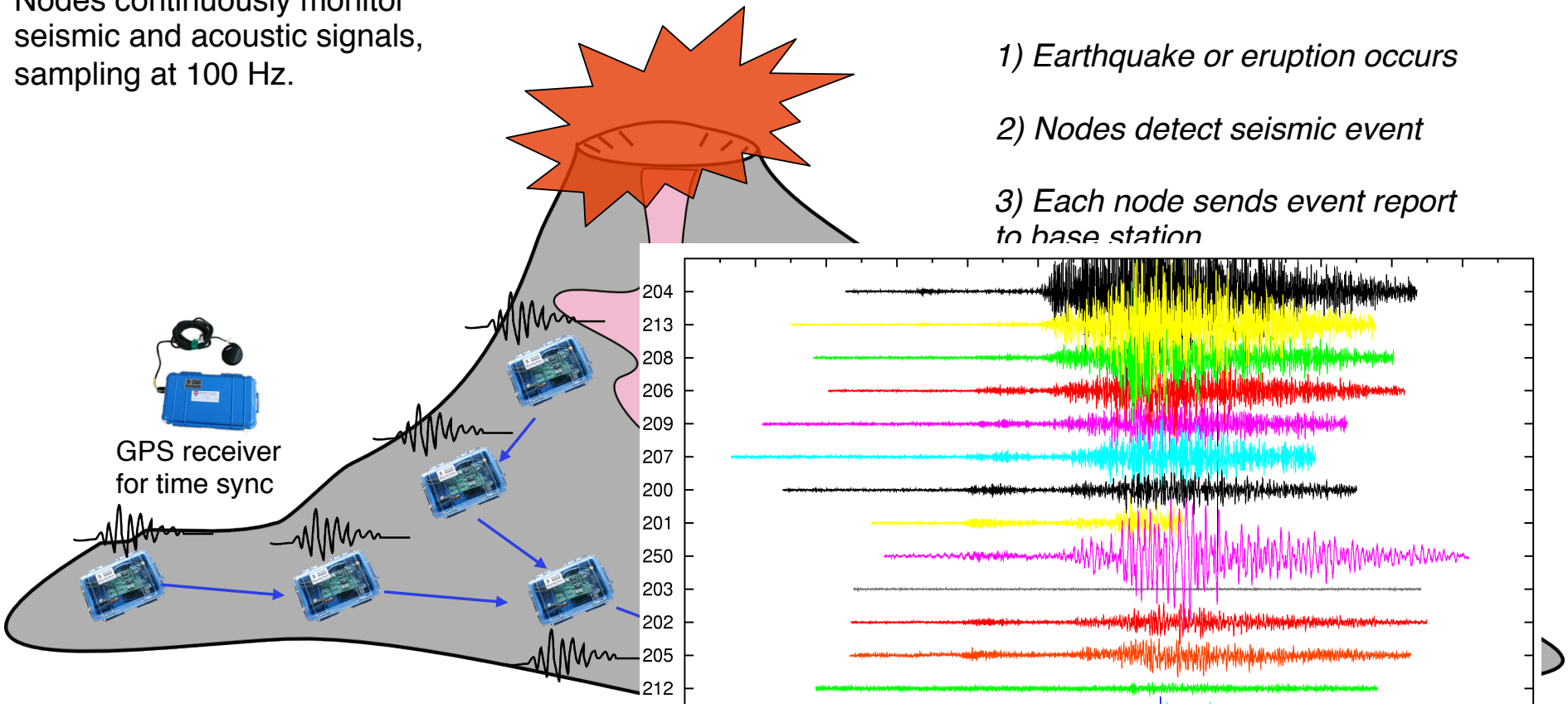
# Application Vignette: Wireless Sensors for Volcanic Monitoring

Nodes continuously monitor seismic and acoustic signals, sampling at 100 Hz.

*1) Earthquake or eruption occurs*

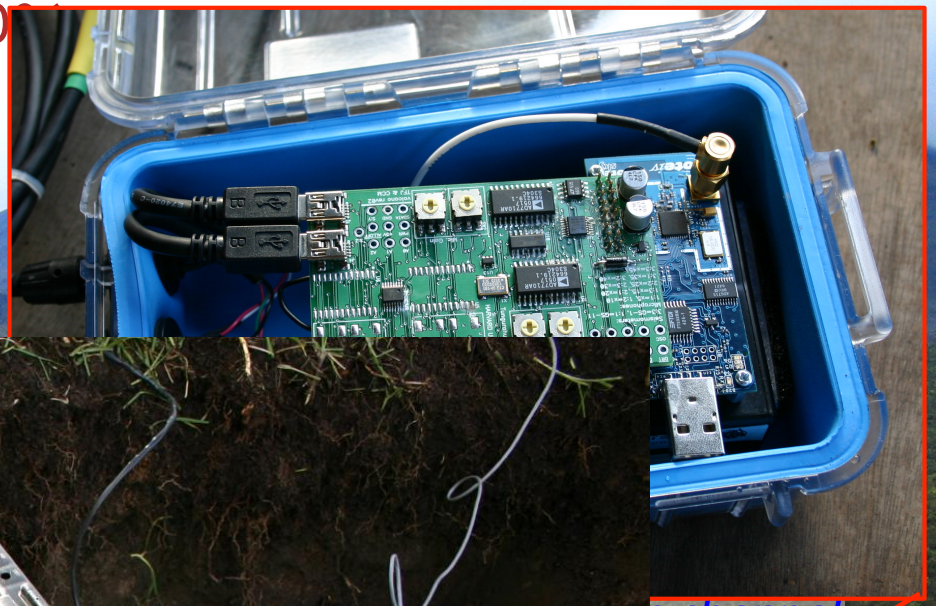
*2) Nodes detect seismic event*

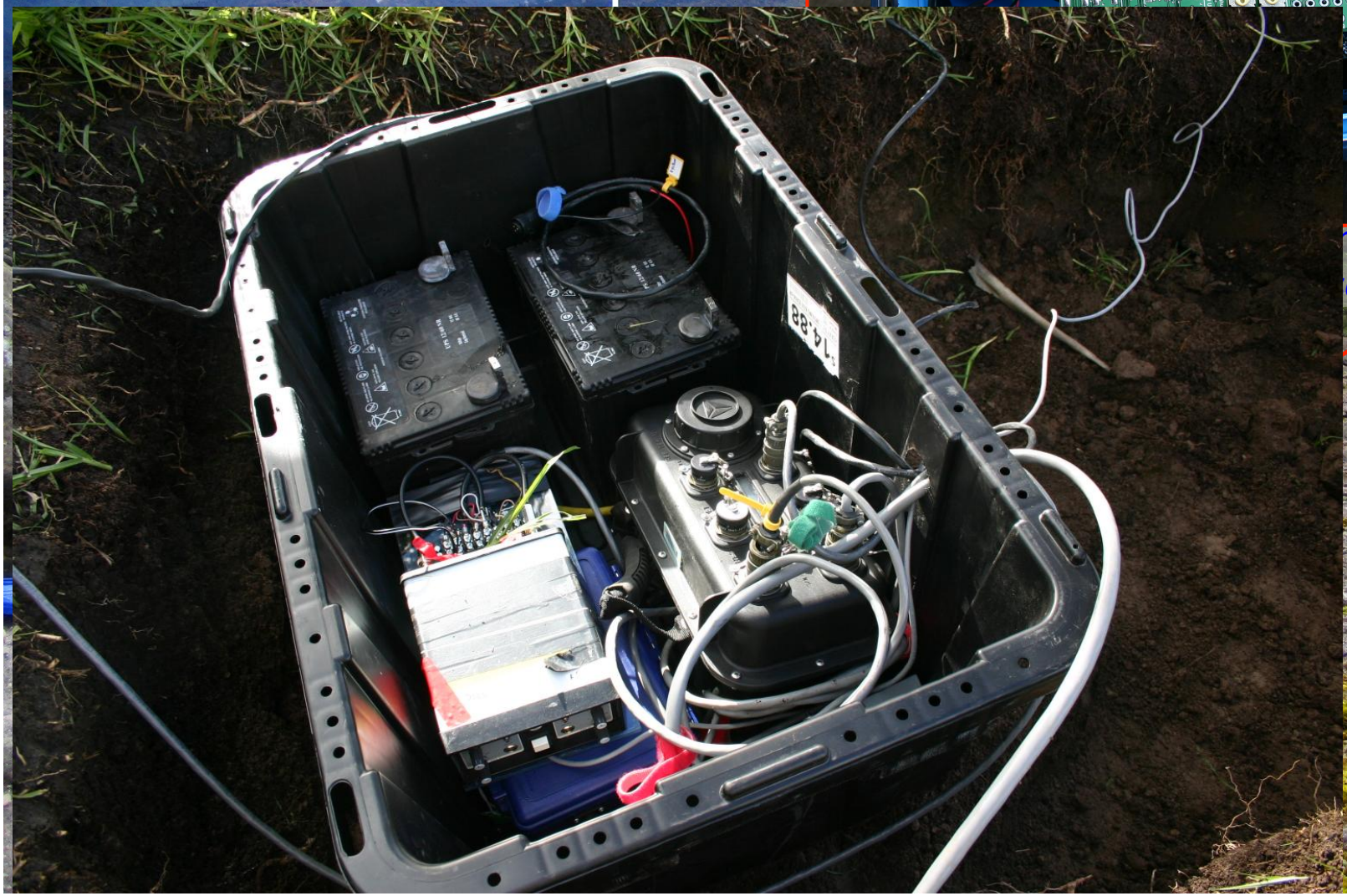
*3) Each node sends event report to base station*



# Reventador Volcano, Ecuador, August 20

with Johnson (NMT) and Lees (UNC)
[OSDI'06, J.Volc. Geo. Res.'06]

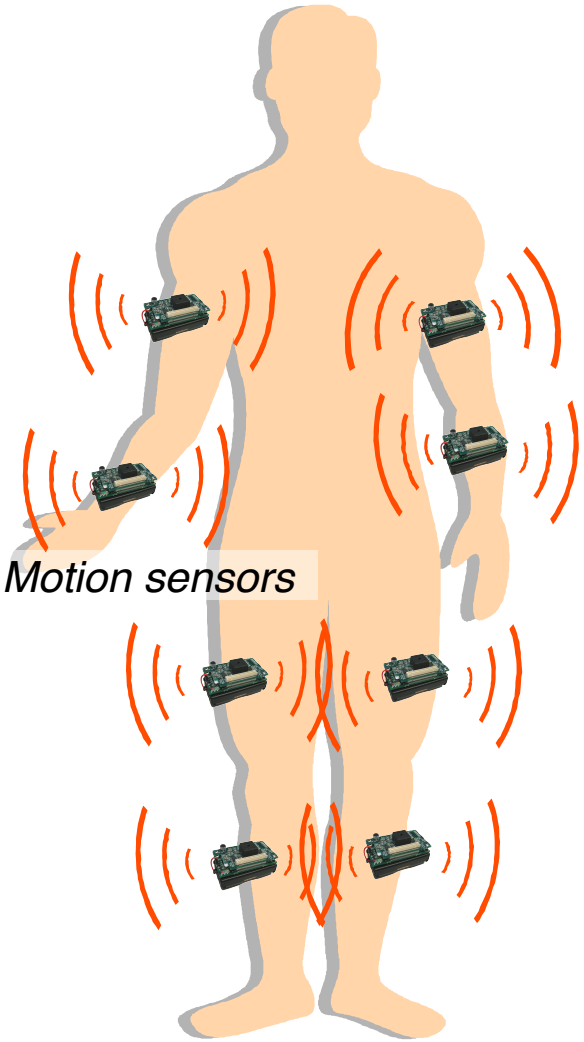




*-channel*
*or node*

# Parkinson's Disease, stroke, and epilepsy monitoring

with P. Bonato, Spaulding Rehabilitation Hospital
[SenSys'09, IEEE Trans. Inf. Technol. Biomed. 2009]



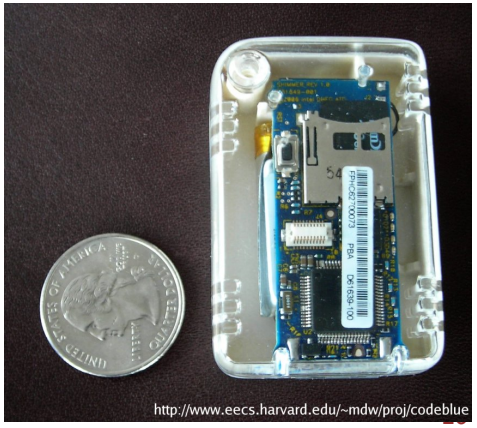

## High-fidelity monitoring of limb motion

- Triaxial accelerometer, triaxial gyroscope
- 6 channels per node, 100 Hz per channel
- Full resolution signal exceeds radio bandwidth
- Store raw data to flash (2 GB MicroSD)

## Nodes perform local feature extraction

- RMS, jerk, dominant frequency, other features...
- Computationally intensive processing
- Requires communication (e.g., time sync, and signal correlation across nodes)

## Offline classification to map features to clinical scores

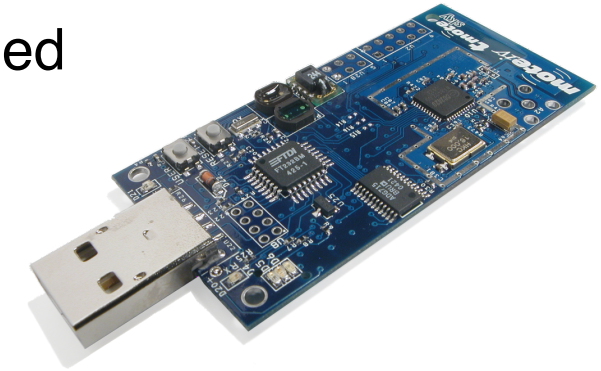



*SHIMMER mote*

# Challenge #1: Resources are extremely limited

Sensor nodes are *severely* resource constrained

- 8 MHz CPU
- 10 KB of memory
- ~100 Kbps of radio link bandwidth (best case)
- 200 mAh – 2000 mAh batteries



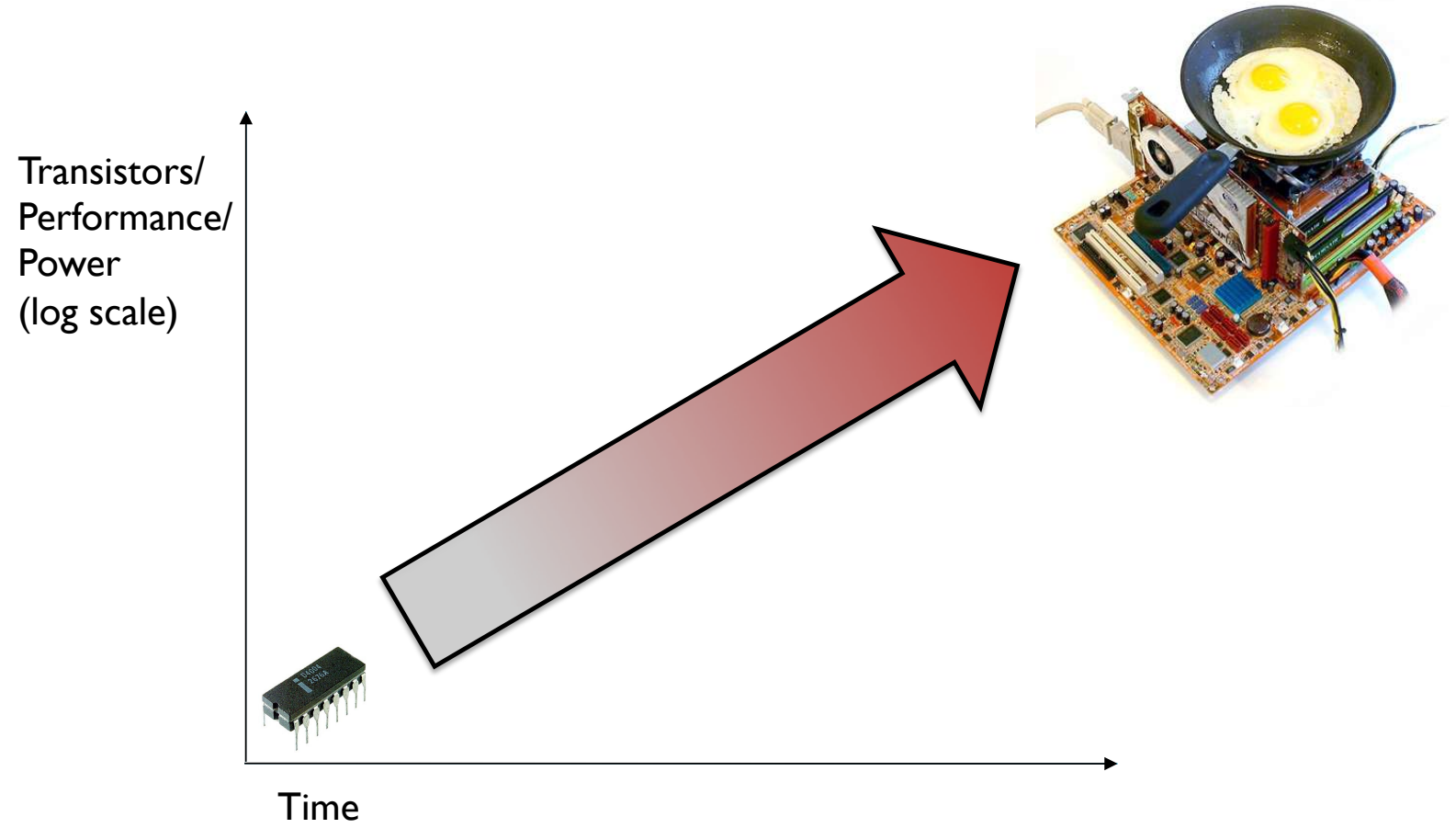
Demands new approach to software design

- Careful management of resources at the **node level**
- **Adaptation** to changing load and resource availability
- Coordination of resource allocation **across the network**

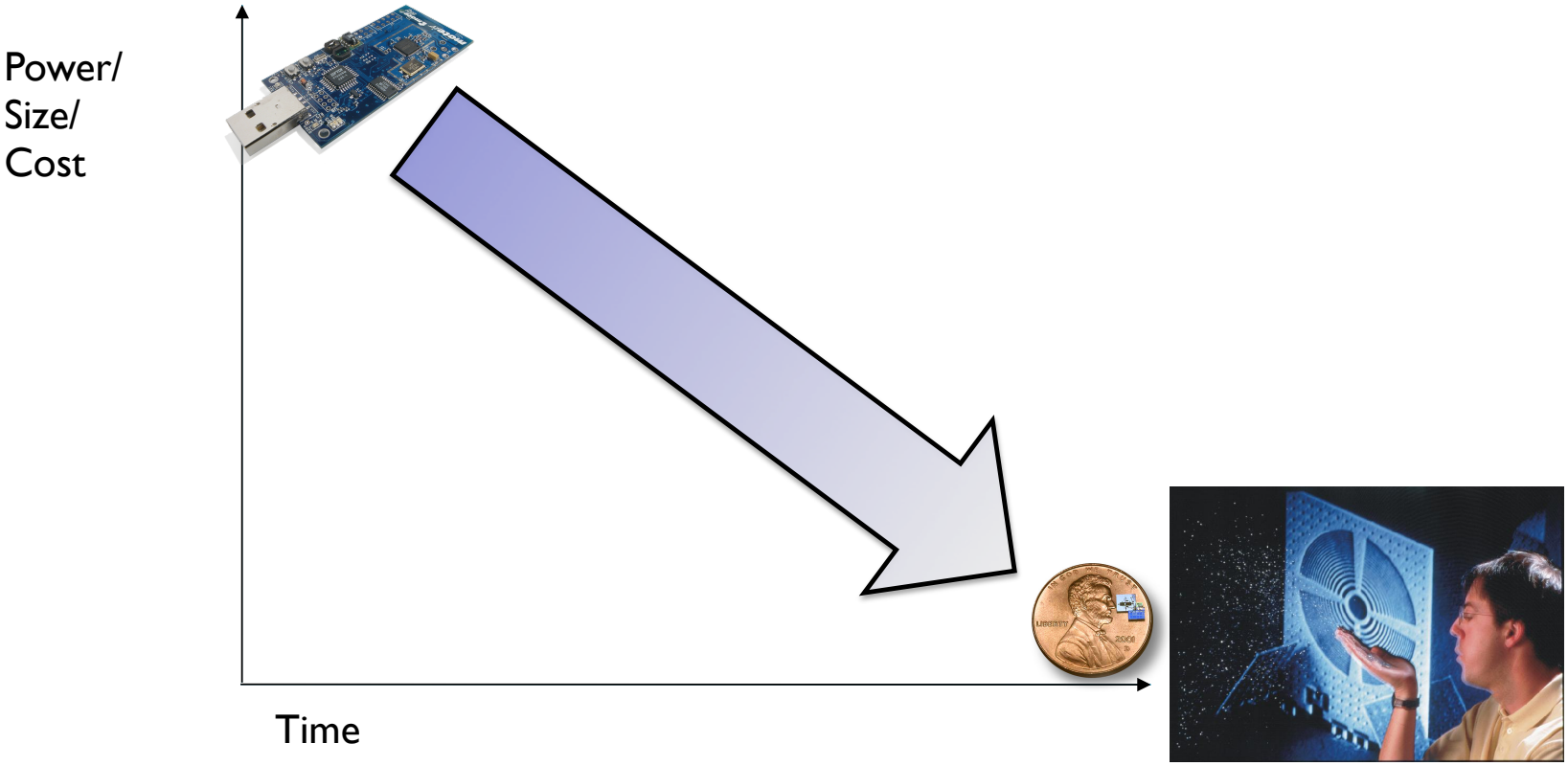
Our mantra: **Resource Aware Programming**

- Make resources a first class primitive in the programming model.

# What about Moore's Law?



# What about Moore's Law?

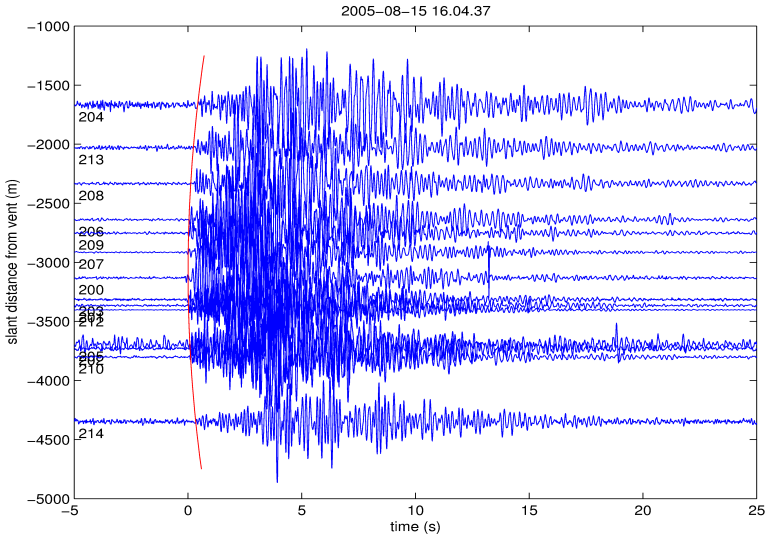


# Challenge #2: Need for high data fidelity

Sensor networks must yield scientifically meaningful data.

Gives rise to many unique challenges:

- Sensor calibration
- Timing precision
- Reasoning about the impact of failures
- Establishing ground truth



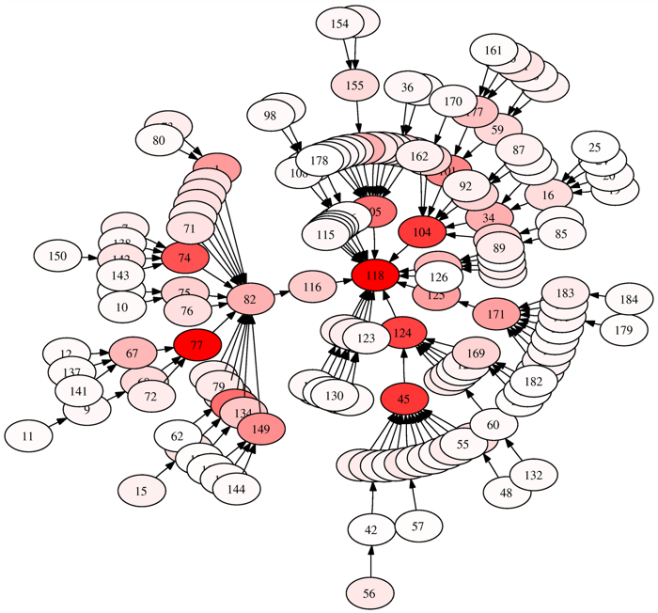

… all coupled with the aforementioned resource limitations.

Many of these challenges are not obvious at first blush.

# Challenge #3: Programming Complex Network Behavior

Current approaches focus on programming individual sensor nodes

- Example: NesC [PLDI'03] – Component oriented variant of C for embedded systems




Extremely low level:

- Radio messaging
- Sensor acquisition
- Hardware duty cycling
- Memory management

Difficult to reason about complex interactions between nodes.

# Talk Outline

Two application vignettes.

Resource Aware Programming:

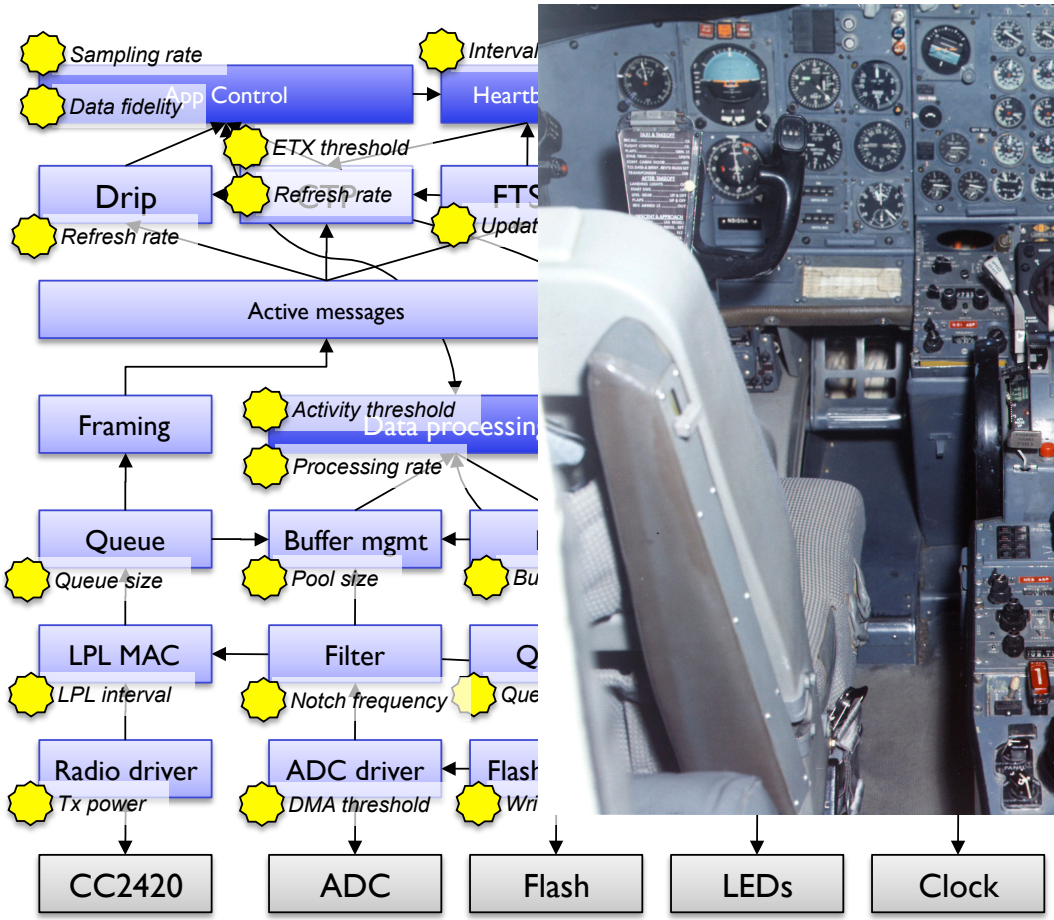
- Make resources first class in the programming model.

Macroprogramming:

- Program the network, rather than the individual device.

Wrap-up and conclusions.

# Resource tuning in sensor networks




# What we really want

# Our Approach: The Pixie Operating System

[SenSys'08, HotEmNets'08]

Pixie is a new operating system for sensor networks that supports:

- Resources as a first-class programming primitive
- Key abstractions: **resource tickets** and **brokers**

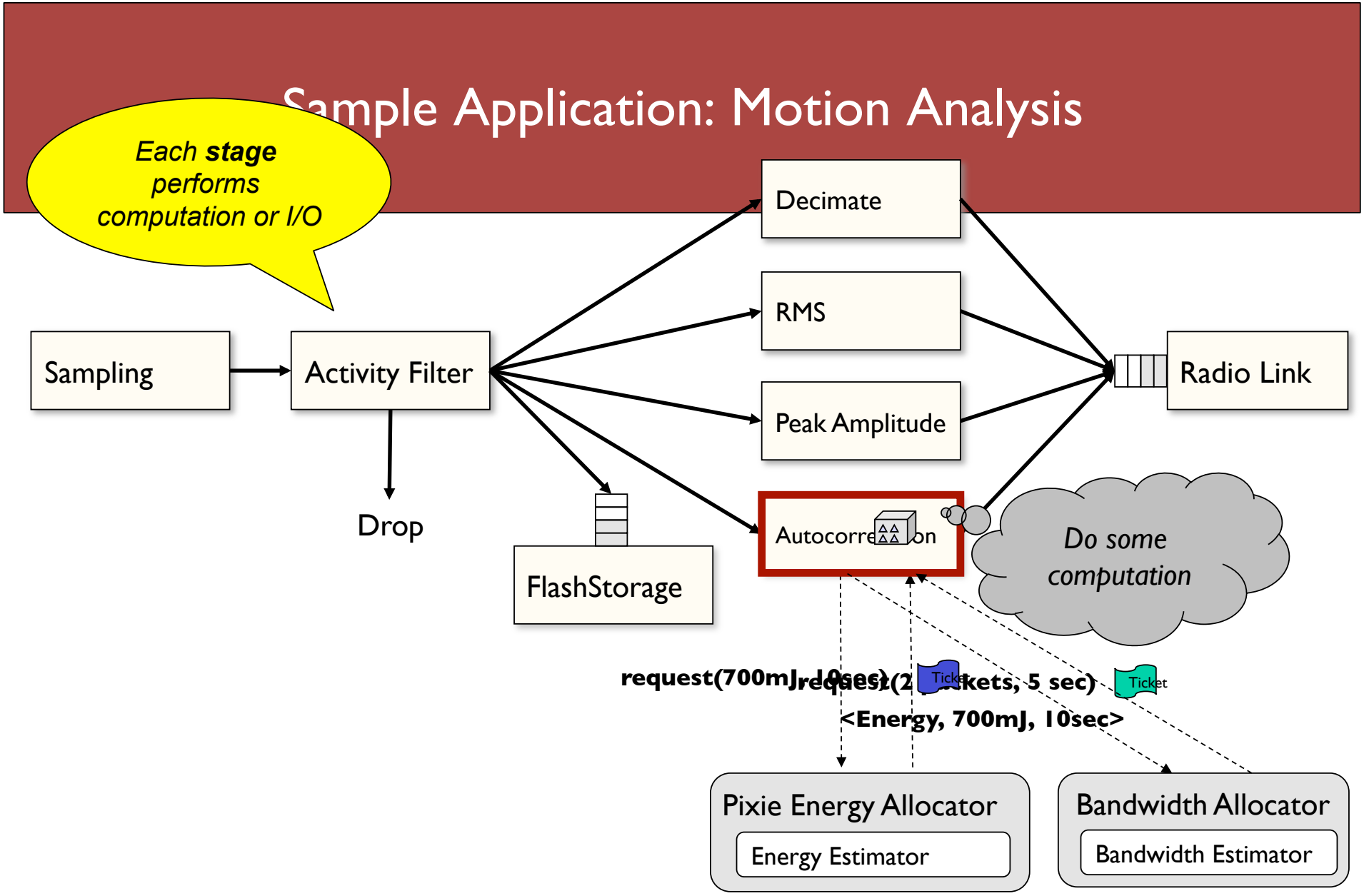
Pixie is intended to make it easy to write adaptive applications

- With such limited sensor node resources, application **must** contend with varying resource conditions!

Fundamental challenge:

- How to enable resource awareness without placing undue burden on the programmer?

# Sample Application: Motion Analysis



# Resource tickets in Pixie

Tickets are / key abstraction for resource management

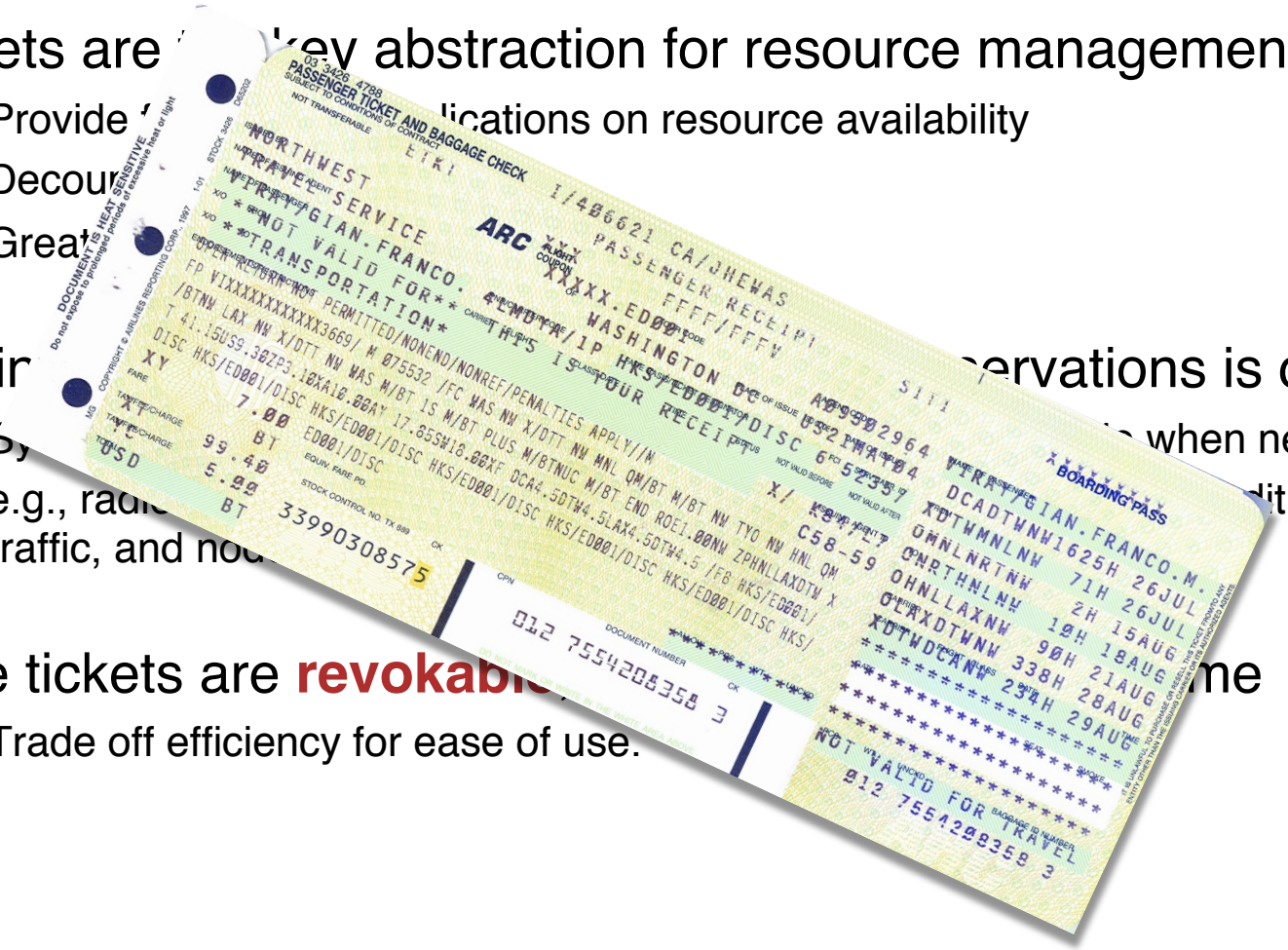
- Provide / lications on resource availability
- Decou
- Grea

Makir servations is difficult:

- S when needed.
- e.g., radi ditions, cross traffic, and no

Pixie tickets are **revokabl** me

- Trade off efficiency for ease of use.



# Resource Brokers

Resource tickets are intended to be low-level and fine-grained

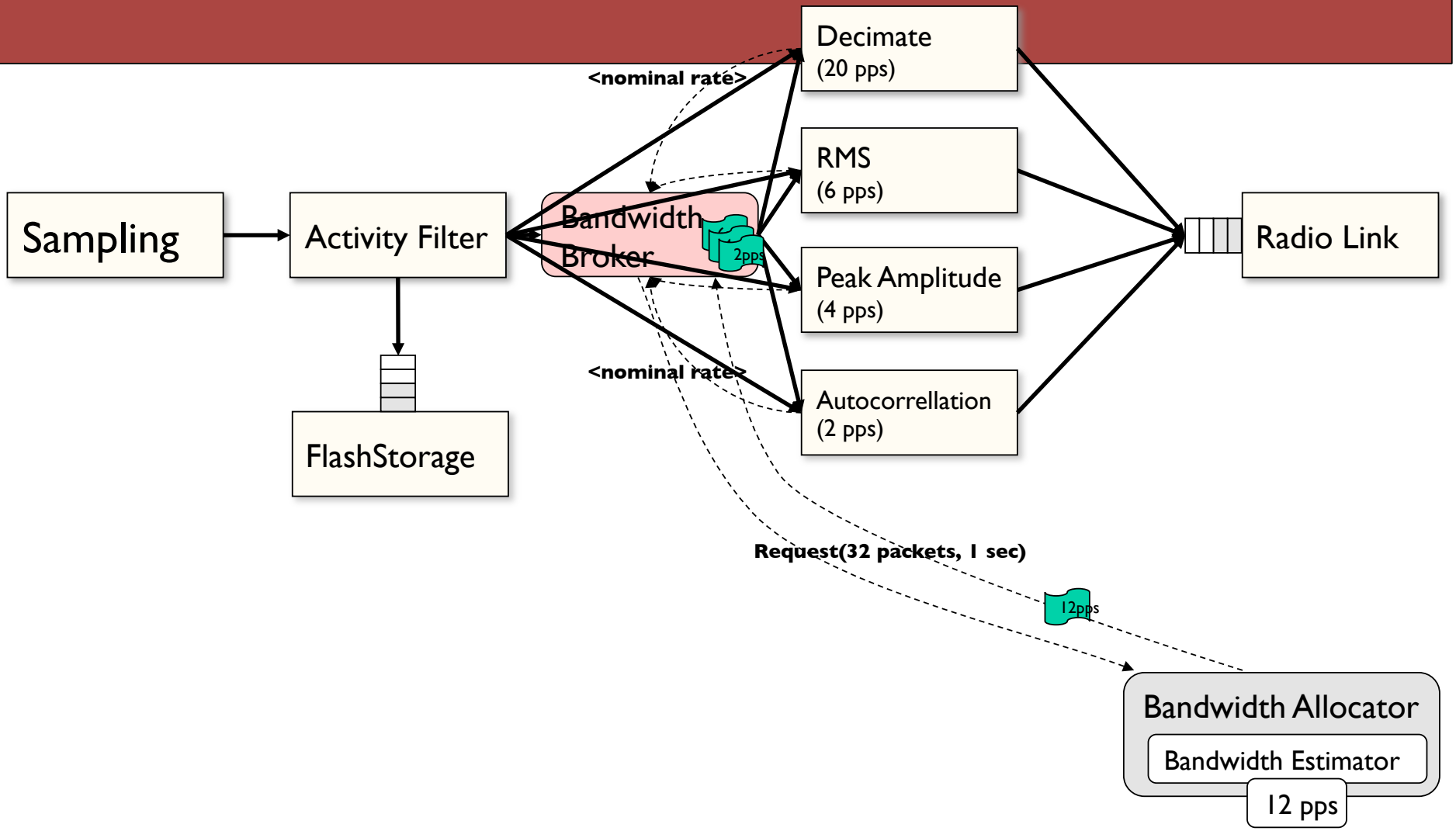
- Very flexible and powerful, but sometimes difficult to use

To simplify app design, we introduce **resource brokers**

- Brokers request and manage resource tickets on behalf of the application
- Each broker implements some policy and provides a high-level API to the app

Tickets provide **mechanism**; brokers implement **policy**.

# Example: Bandwidth Broker



# Evaluation: Bandwidth Adaptation in Motion Analysis

Goal: Adapt type and quantity of data transmitted by each node as link quality varies

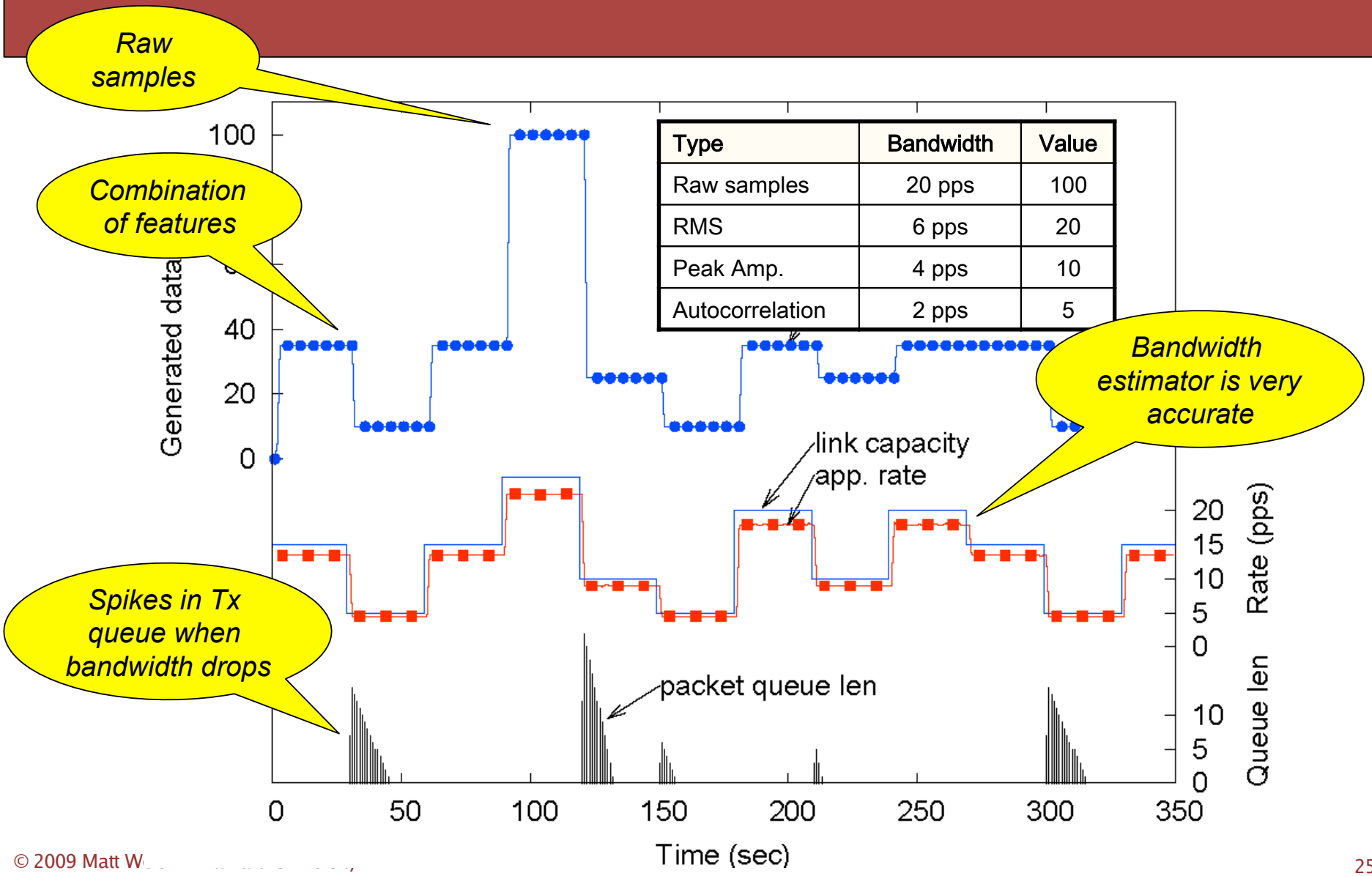
- Don't want to transmit data that won't get to the base station.

Uses Pixie's bandwidth broker

- Multiple data types with associated **cost** and app-assigned **value**
- Estimate available link bandwidth
- Assign bandwidth tickets to maximize value subject to bandwidth limit

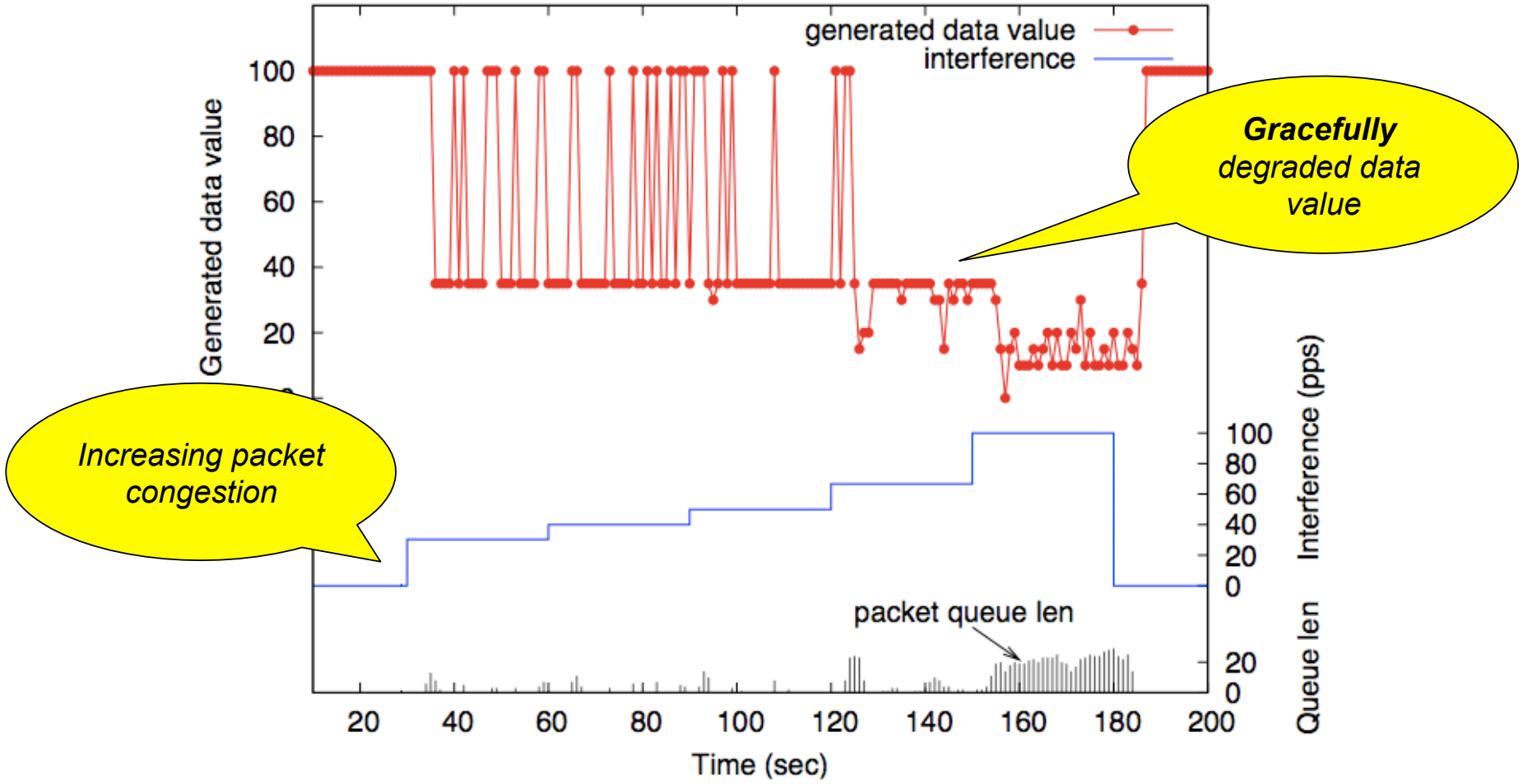
| Type | Bandwidth | Value |
|---|---|---|
| Raw samples | 20 pps | 100 |
| RMS | 6 pps | 20 |
| Peak Amp. | 4 pps | 10 |
| Autocorrelation | 2 pps | 5 |

# Experiment: Bandwidth Adaptation




| Type | Bandwidth | Value |
|---|---|---|
| Raw samples | 20 pps | 100 |
| RMS | 6 pps | 20 |
| Peak Amp. | 4 pps | 10 |
| Autocorrelation | 2 pps | 5 |

# Adaptation to radio channel interference



# Network-Wide Resource Management

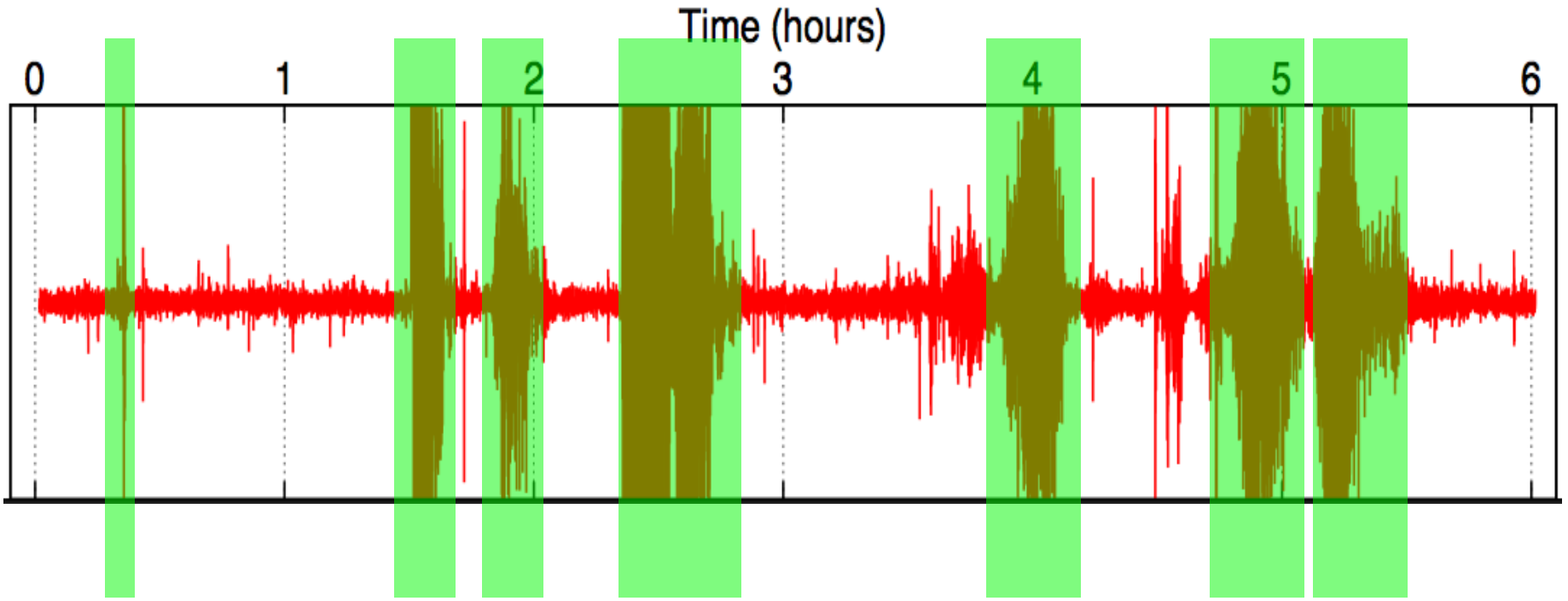
Large sensor networks constrained in two important ways:

- Bandwidth: Reliable data transfer is very slow.
- Energy availability: Even if bandwidth were plentiful, can't run radios all the time.

To optimize data fidelity, need to carefully manage resources **across the network** as a whole.

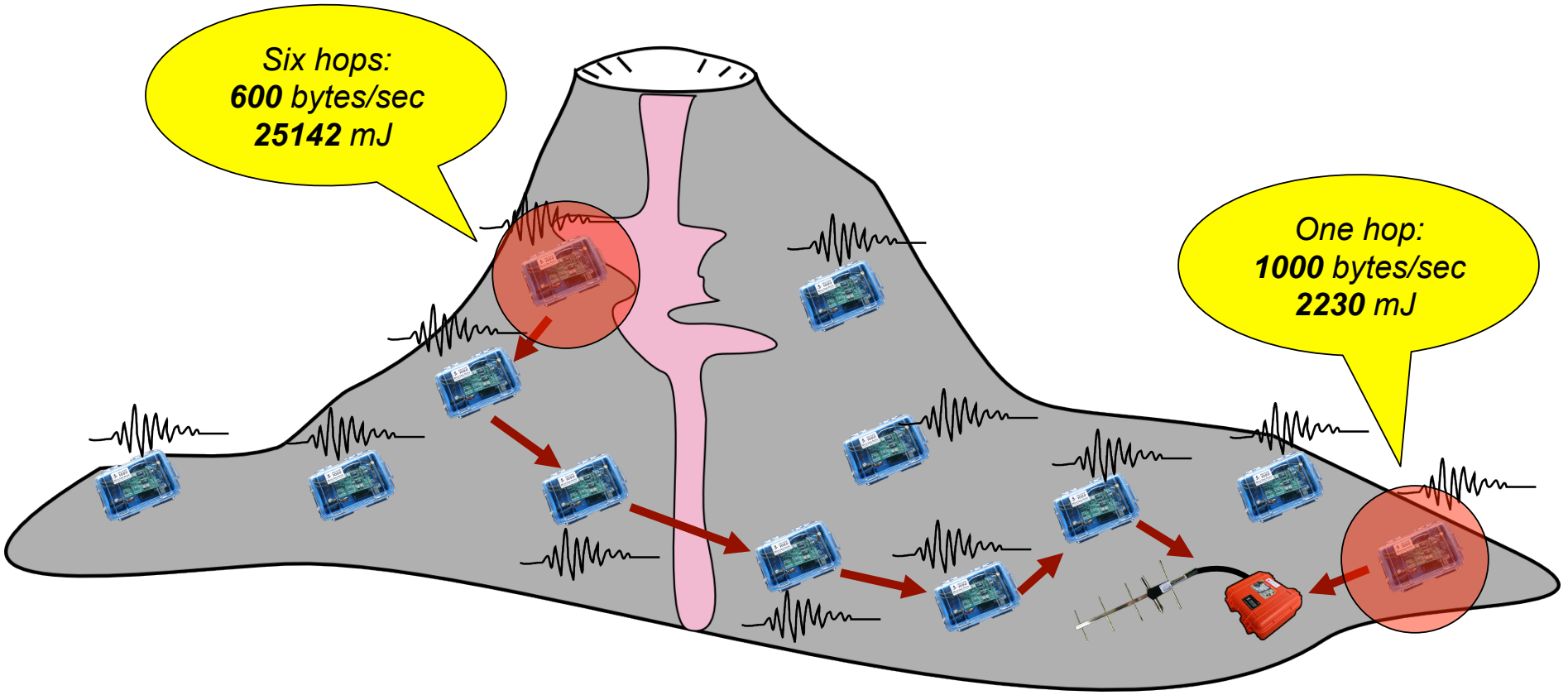
Let's focus on one key problem: Reliable signal collection

# Observation: Data Differs in Value




interesting data

# Observation: Data Differs in Acquisition Cost



# Our Approach: Lance
[SenSys'08]

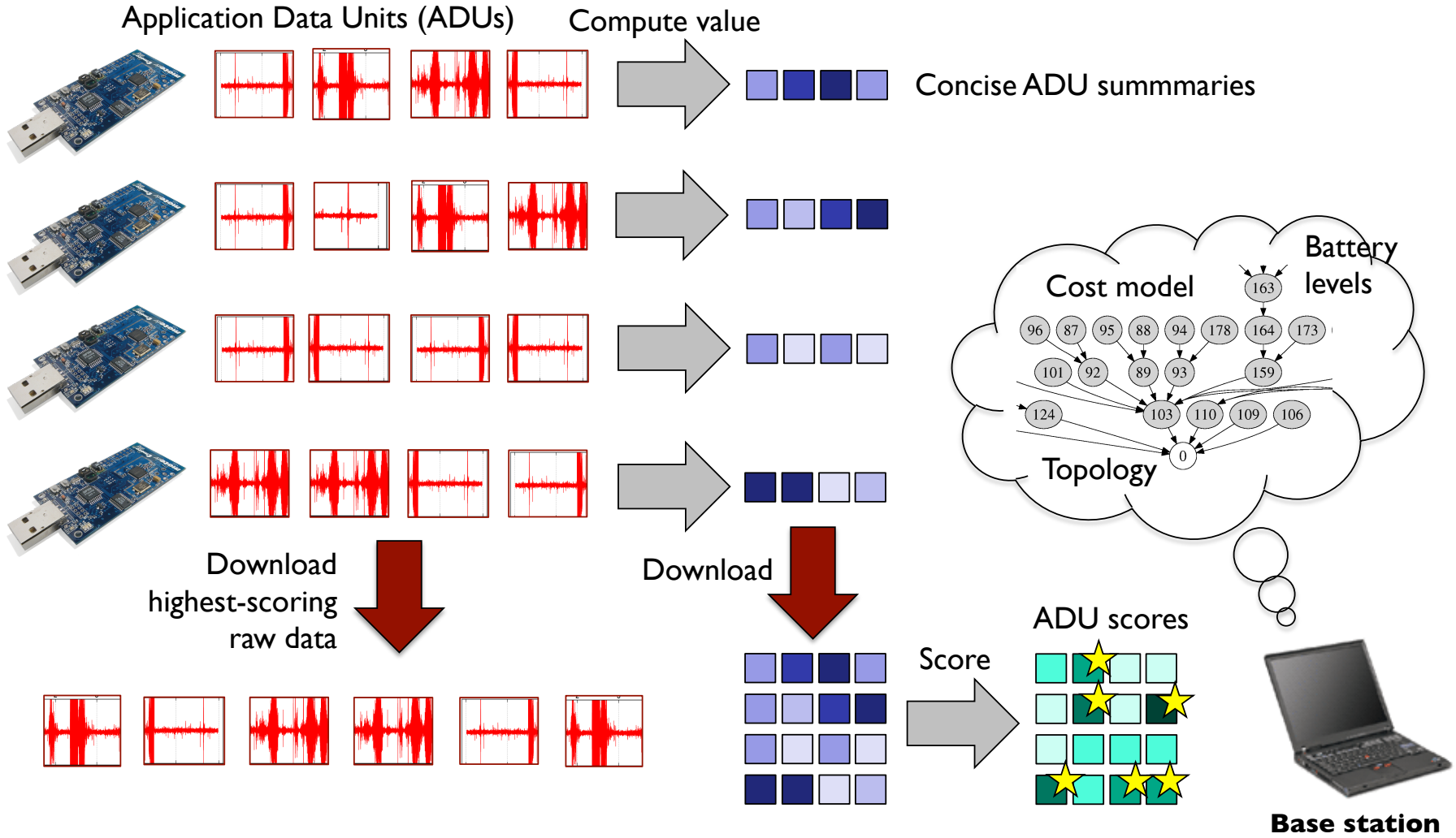
System for priority driven allocation of bandwidth and storage resources within a sensor network

Goal: Optimize data quality subject to energy and bandwidth constraints

Lance decouples mechanisms for resource allocation from app-specific policies

- Can target many optimization metrics
- e.g., fairness, spatial/temporal data distribution, etc.

# Lance Design



# Optimization Goal

***U*** = Set of all possible ADUs

***E*** = Vector of energy capacity of each node

*c(u), v(u)* = cost and value for each ADU *u*

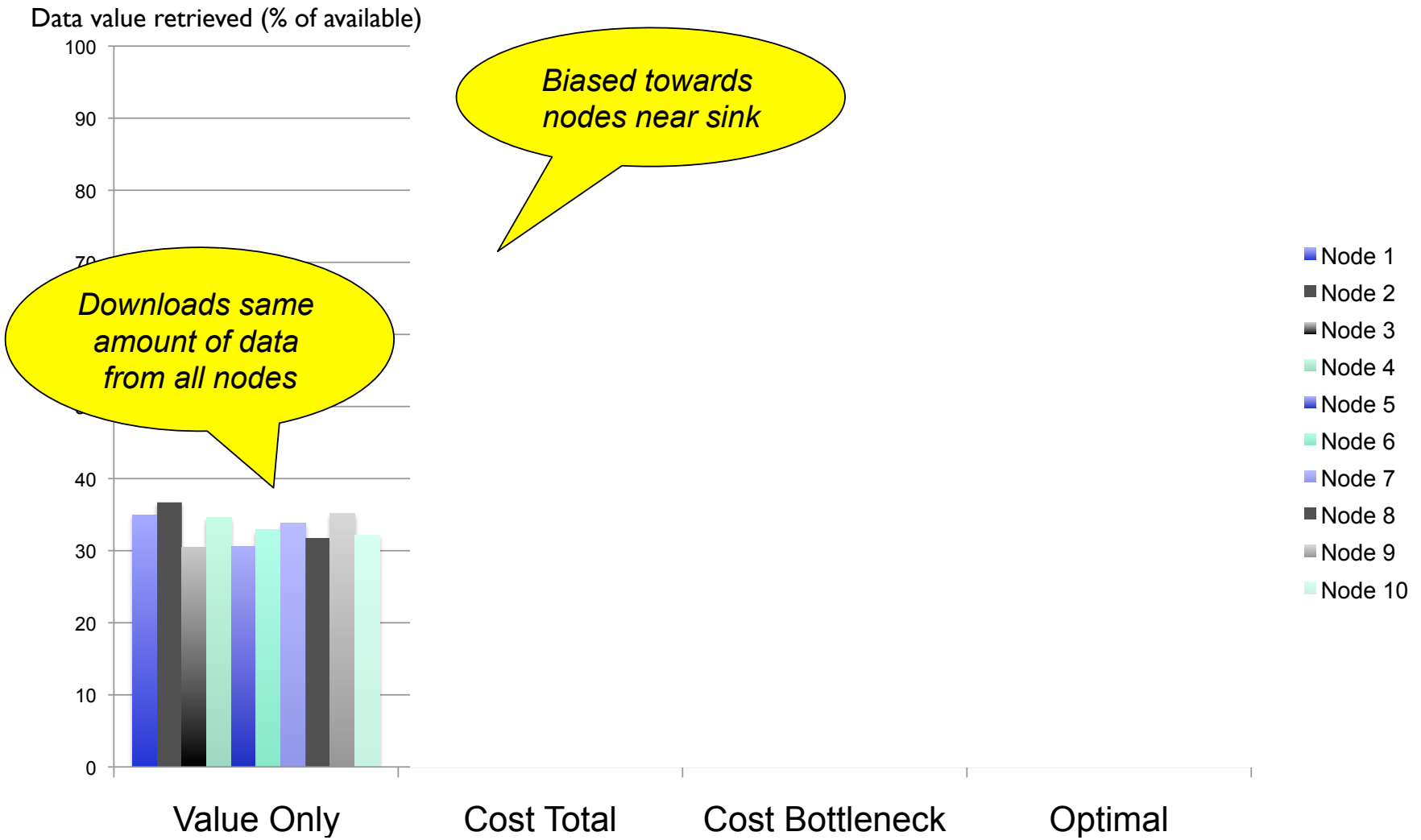
The optimal set $u_o$ is the set of ADUs maximizing

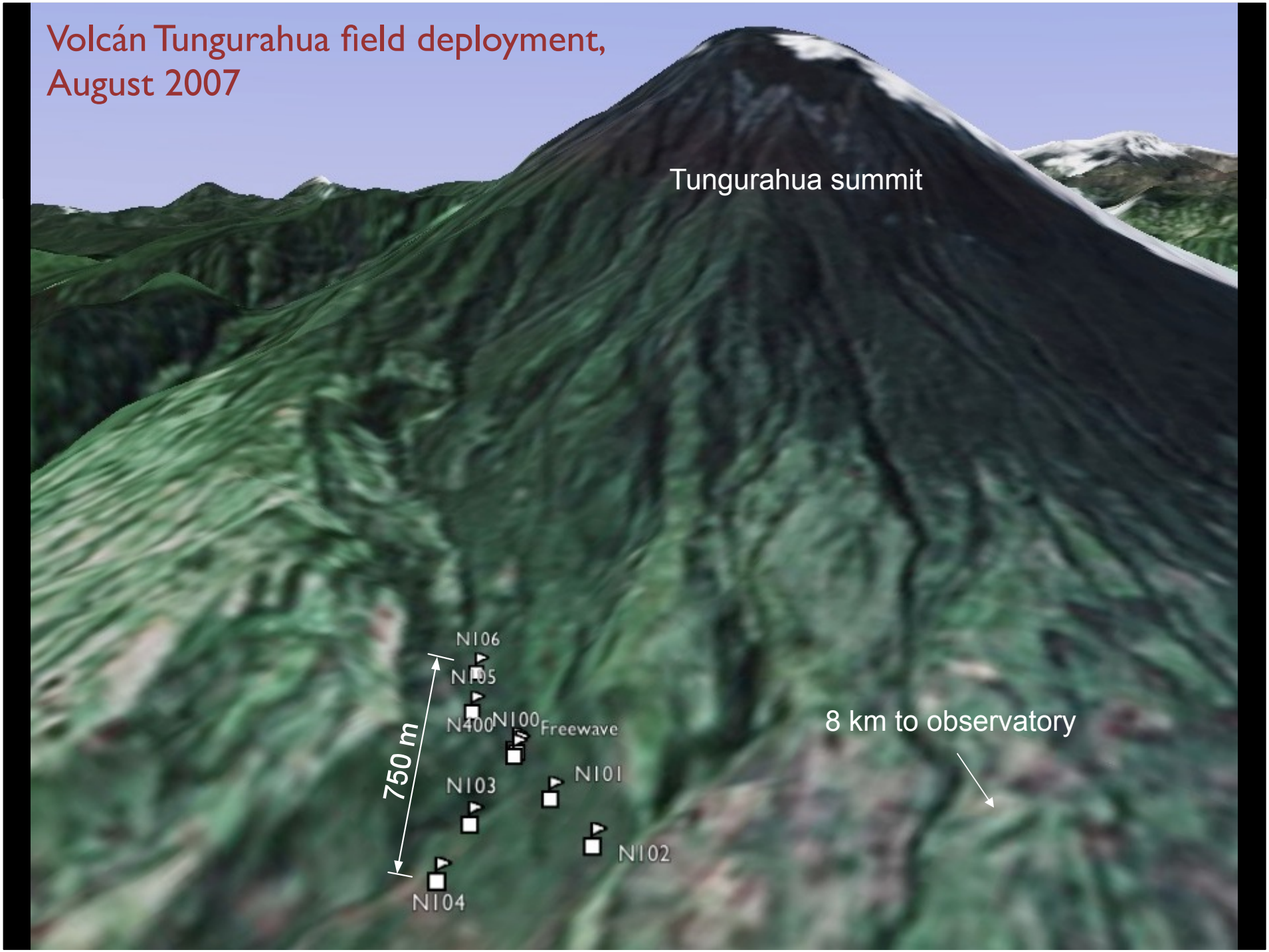
$$\sum \boldsymbol{v(u_o)} \text{ subject to } \sum \boldsymbol{c(u_o)} \leq \boldsymbol{E}$$

Offline Solution: **Multidimensional Knapsack Problem**

But we need an online, greedy approximation.

# Scoring function comparison
## 10-node linear chain




Volcán Tungurahua field deployment,
August 2007
Tungurahua summit
N106
N105
N400
N100
Freewave
N101
N103
N102
N104
750 m
8 km to observatory

# Deployment Results

Deployed 8 sensor nodes for a total of 71 hours

- Lance used to manage storage and bandwidth
- Experimented with different prioritization functions and policy modules

Successfully downloaded 1232 ADUs (77 MB of data)

- 308 downloads failed due to timeouts: success rate 80%

Total storage summaries span 11012 ADUs (688 MB of data)

- Lance downloaded 11% of the data acquired by the network

Lance downloaded **99.5%** of the *optimal* dataset

# Talk Outline

Two application vignettes.

Resource Aware Programming:

- Make resources first class in the programming model.

Macroprogramming:

- Program the network, rather than the individual device.

Wrap-up and conclusions.

# Macroprogramming

I hope I've convinced you that sensor net programming is hard.

- Can we do better?

Key idea: Program the network **as a whole.**

Compile down automatically to node-level programs.

Key challenges:

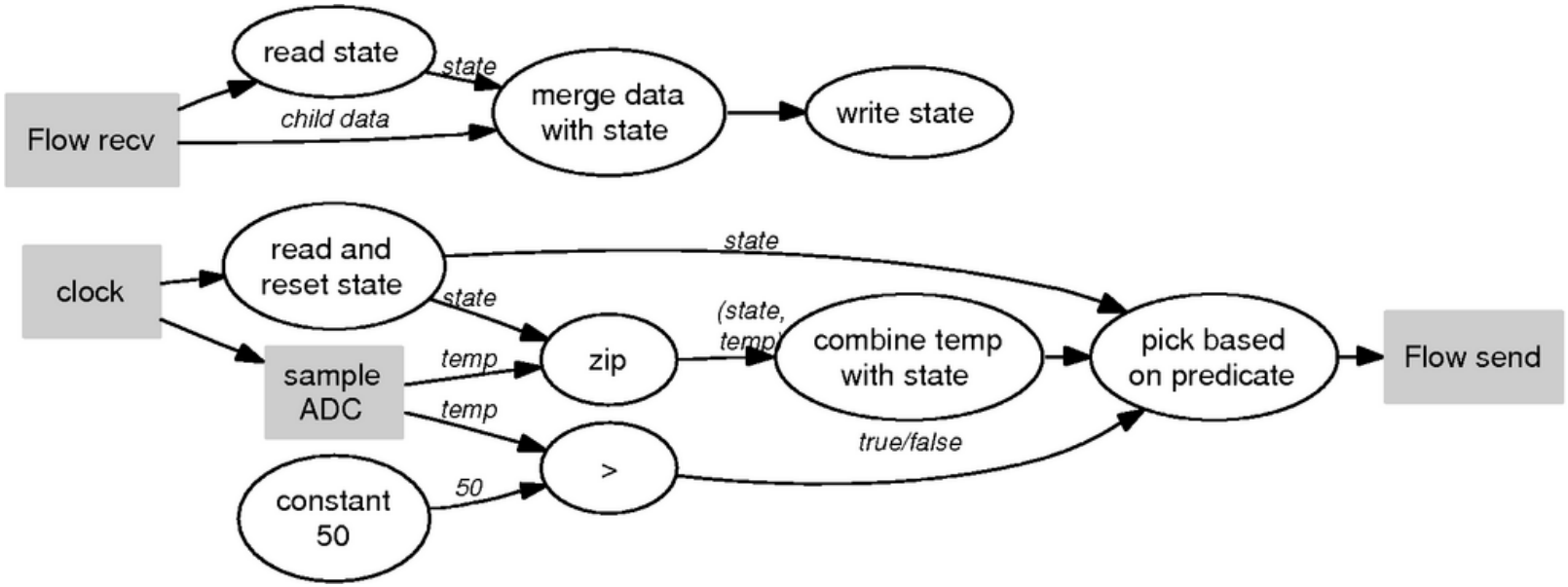
- What are the appropriate programming abstractions?
- How do we retain good resource efficiency?
- How to automate the compilation process?

# Flask: Dataflow Macroprogramming
[ICFP'08]

Synthesize sensor net programs from high-level dataflow

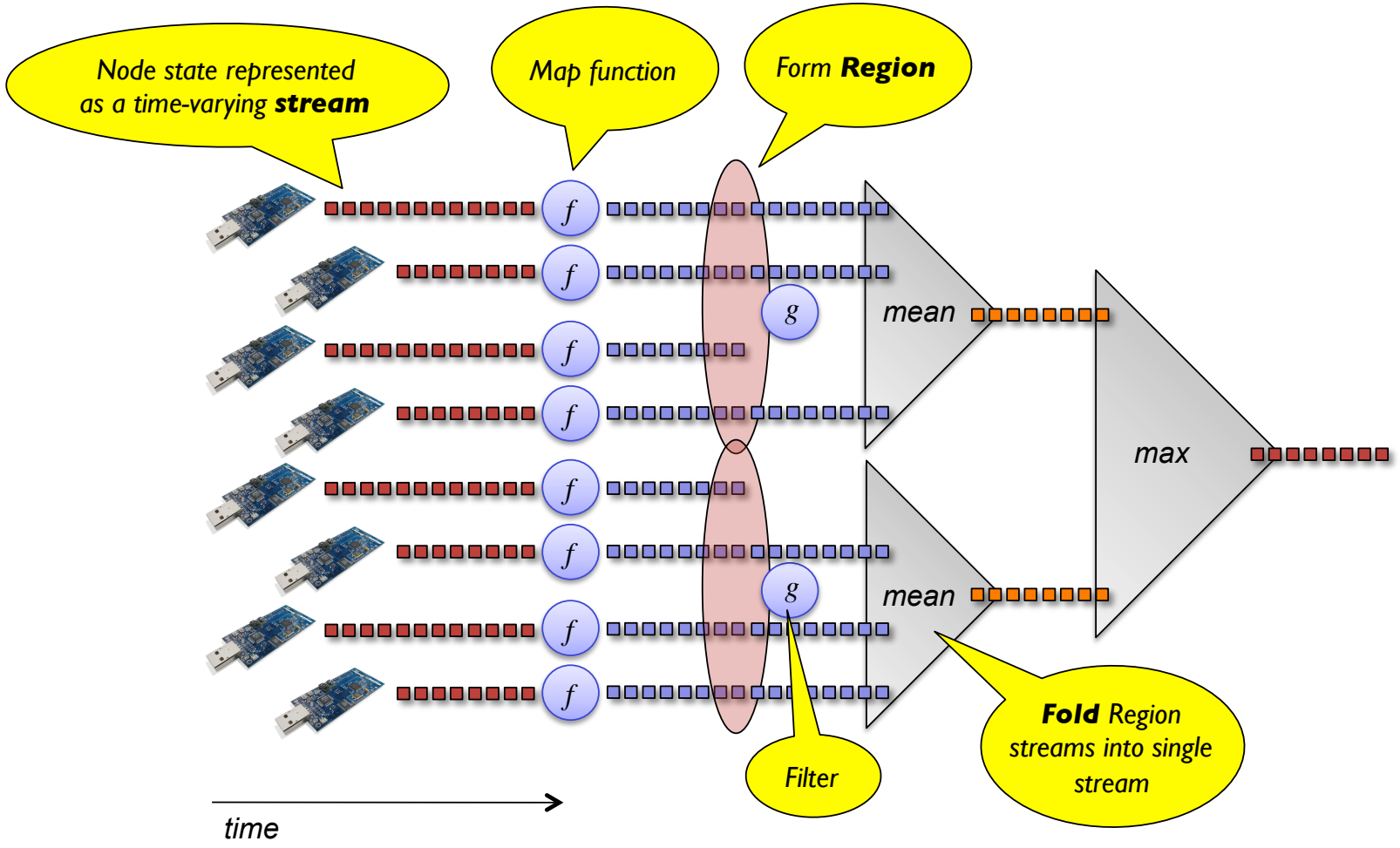
Example: Compiling an SQL query

```
SELECT AVERAGE(temp) WHERE temp > 50 PERIOD 10 s
```

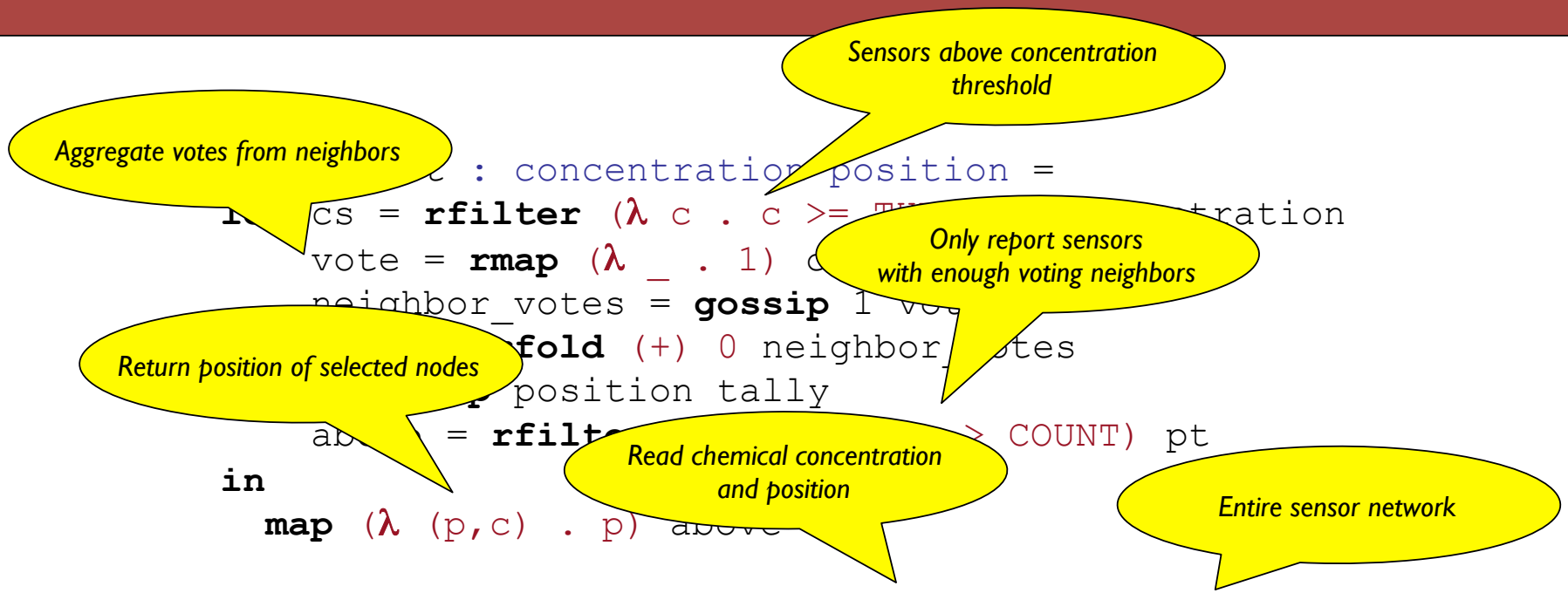


# Regiment: A Stream-Oriented Macroprogramming Language

[IPSN'07, IPSN'05, DMSN'04]



# Regiment Example: Chemical plume "Hotspot" Detection

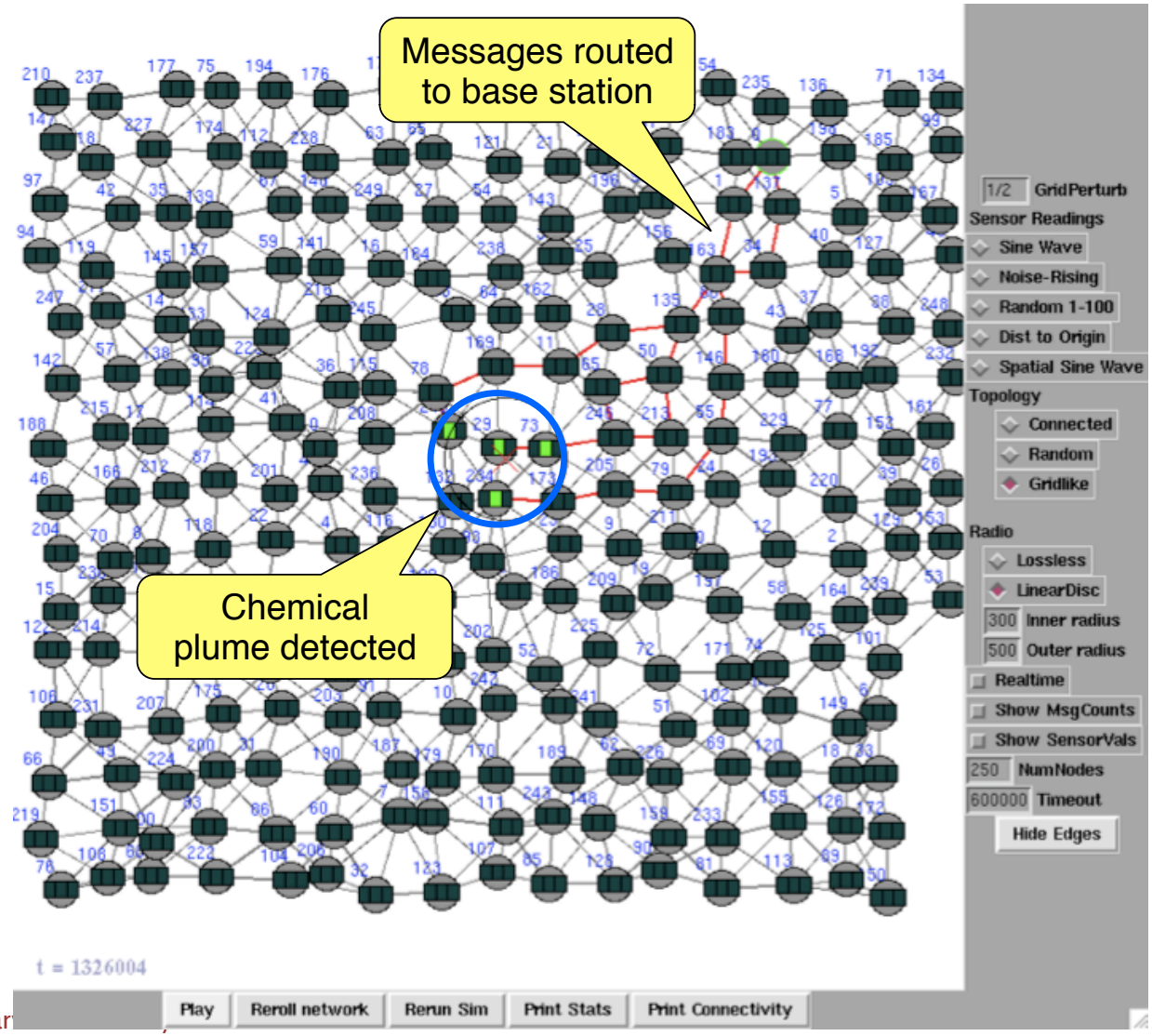



```
                c : concentration*position =
            cs = rfilter (λ c . c >= TH          tration
                vote = rmap (λ _ . 1) c
                neighbor_votes = gossip 1 vo
                fold (+) 0 neighbor_votes
                position tally
                ab      = rfilt                > COUNT) pt
        in
          map (λ (p,c) . p) above

    let conc_stream = rmap (λ n . get_conc(n)) world in
    let pos_stream = rmap (λ n . get_pos(n)) world in
      hot_spot (conc_stream pos_stream)
```

Compare to thousands of lines of C code.

# Regiment Plume Detection at Work



# Overview of Other Projects

## Novel programming models

- **Abstract Regions**: Neighborhood communication model [NSDI'04, HotNets'03]
- **SORA**: Decentralized resource allocation based on machine learning [NSDI'07]

## Planetary-scale data collection and processing

- **COBRA**: RSS and blog search engine [NSDI'06]
- **Hourglass** and **SBONs**: P2P overlay for stream processing [ICDE'06, NetDB'05, IPTPS'05]

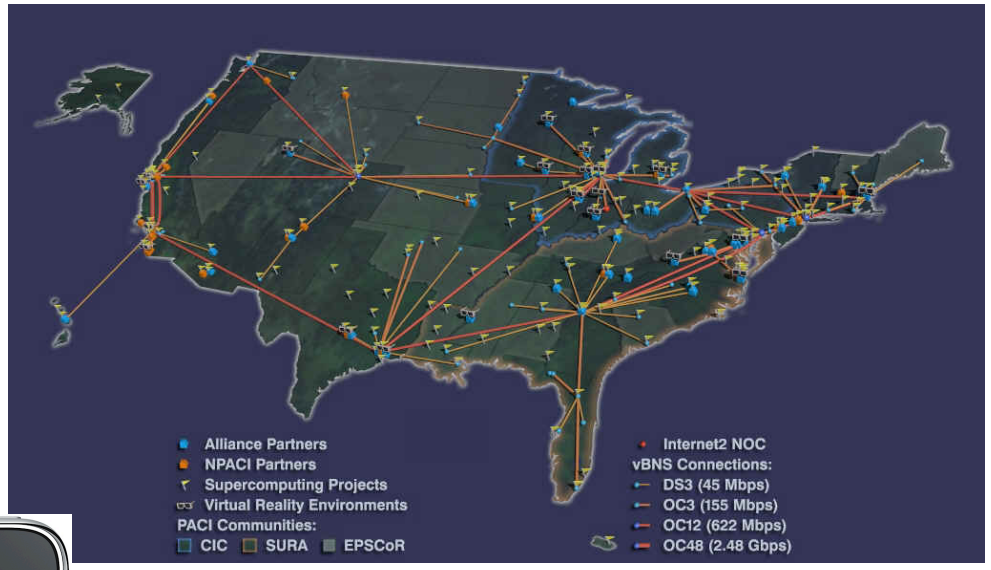
## Testbeds and tools

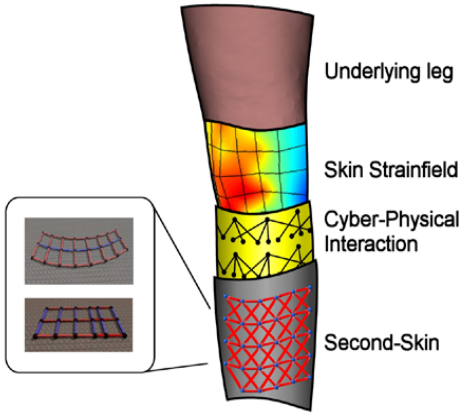
- **MoteLab**: 184 nodes in Maxwell Dworkin [SPOTS'04]
- **CitySense**: Urban-scale open sensor testbed [IEEE HST'08]
- **TOSSIM**: Accurate, energy-aware sensor network simulation [SenSys'03, SenSys'04]

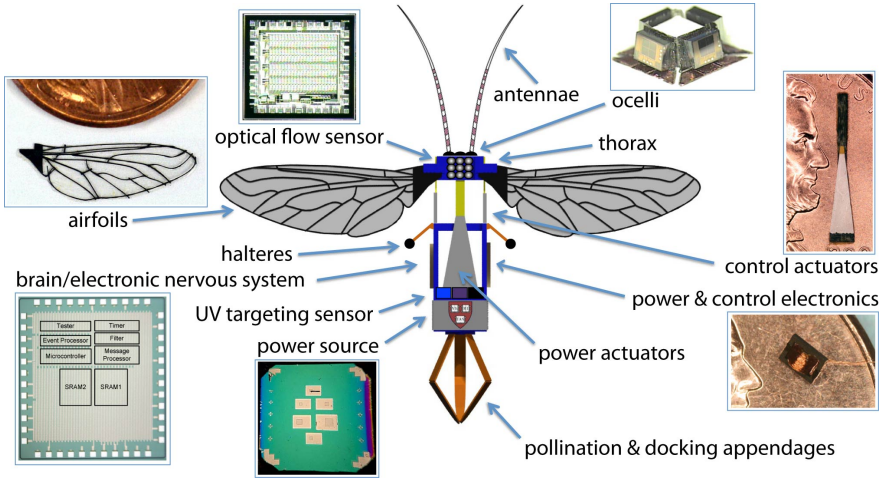
## New architectures for wireless LANs

- **Dyson**: A programmable, evolvable wireless LAN [HotNets'08]

# Implications – Beyond Sensor Networks



# Conclusions

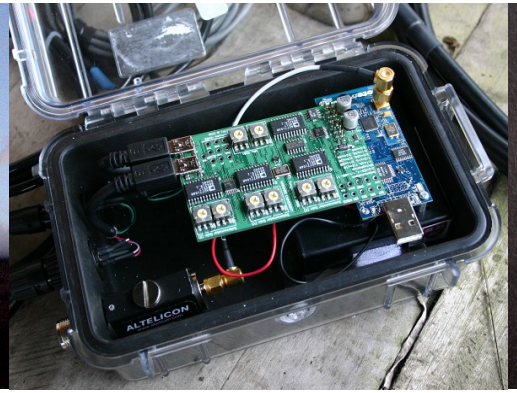
Sensor networks have tremendous potential to impact scientific discovery.

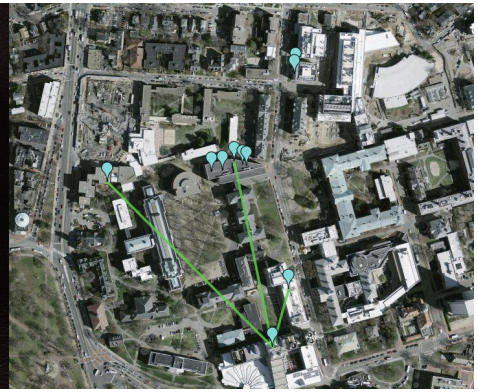
They give rise to several unique programming challenges:

- Achieving good resource efficiency at the node and network level
- Building up complex network behaviors

Our work has focused on OS, programming model, and language techniques to tackle these challenges.

- Real-world scientific applications are the key driver behind this research agenda.



# Acknowledgments

**Current and former grad students and postdocs:**
Geoffrey Challen, Dr. Bor-rong Chen, Breanne Duncan, Dr. Konrad Lorincz, Geoffrey Mainland, Rohan Murty, Dr. Peter Pietzuch, Ian Rose, Atanu Roy Chowdhury, Victor Shnayder, Jason Waterman, Dr. Alexander Wissner-Gross

**Current and former undergrads:**
Peter Bailis, Keith Berkoben, Stephen Dawson-Haggerty, Dr. Thaddeus Fulford-Jones, Jason Gao, Yang Gao, Mervin John, Qicheng Ma, Josh Montana, Nambi Nallasamy, Chris Newman, Samir Paul, Geoffrey Peterson, Danny Popper, Peter Salas, Chris Simmons, Daniel Steinbrook, Hassan Sultan, Aditya Sunderam, Patrick Swieskowski, Matt Tierney, Yara Wehbe, Andrew Wong, Mumu Xu, Dimah Yanovsky, Chelsea Zhang

**Collaborators:**
Dr. Jeffrey Johnson (NMT), Dr. Jonathan Lees (UNC), Dr. Paolo Bonato (Spaulding), Dr. Steve Moulton (Denver Children's), Tia Gao (Stanford), Dr. Jitu Padhye (Microsoft), Dr. Ranveer Chandra (Microsoft), Josh Bers (BBN), Dr. Ryan Newton (MIT), Glenn Holloway, Dr. Majid Ezzati (HSPH), Dr. Jack Spengler (HSPH), Dr. David Brooks, Dr. Greg Morrisett, Dr. Radhika Nagpal, Dr. David Parkes, Dr. Mema Roussopoulos, Dr. Margo Seltzer, Dr. Gu-Yeon Wei

**Special Thanks:**
Gioia Sweetland, Joanne Bourgeois

*fin*

# Related Work

## Resource management primitives in sensor networks

- Large amount of work in energy reduction: MAC protocols, routing, sensor duty cycling
- ICEM, SNACK: Enable cross-component energy reduction
- iCount, Quanto: Energy metering support

## Programming model support for resource management

- Eon, Levels: Tune energy consumption according to set policy
  - *We provide same functionality through Pixie resource brokers*
- TinyDB: Lifetime-targeted queries

## Mobile systems

- Odyssey: Framework allowing applications to adapt to energy, bandwidth, and CPU
- ECOsystem: Tune OS scheduling policy for battery lifetime targets
- Puppeteer: Adaptation to bandwidth variation

# Intelligent Instrumentation

Sensor networks are not just passive instruments!

We can push processing into the network.

Processing can happen at many levels:

- On individual sensor nodes.
- At aggregation points within the network.
- At the base station or gateway.

Sensor networks *fundamentally change* the notion of "scientific observation" from a *passive* process to an *active* one.

- This has a deep impact on many aspects of science.

# Resource Estimation

## Storage and memory are straightforward

- Allocators track total flash/memory consumption

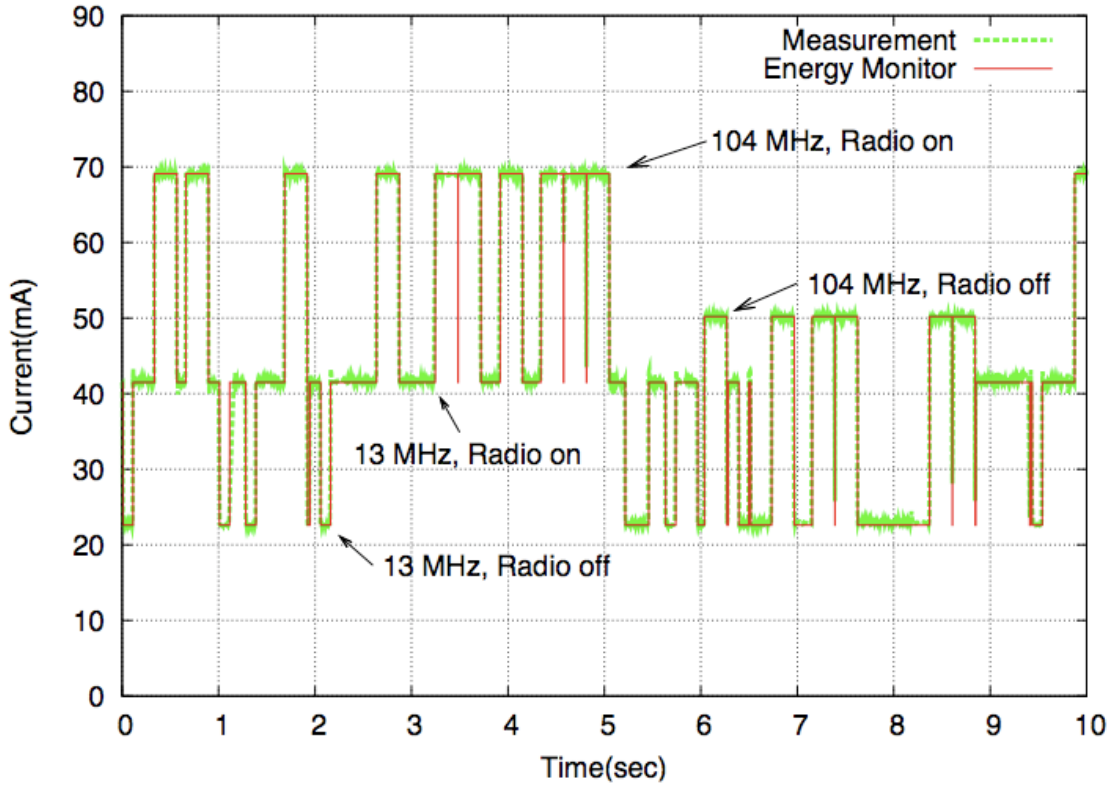
## Bandwidth: Radio link estimates packet transmission delay

- Measure total time for packet transmission plus ARQ (if used)
- Invert transmission delay to estimate instantaneous bandwidth
- Maintain separate estimate for each neighbor
- Currently assumes delay is independent of packet size

# Software energy metering

Energy estimation performed in software

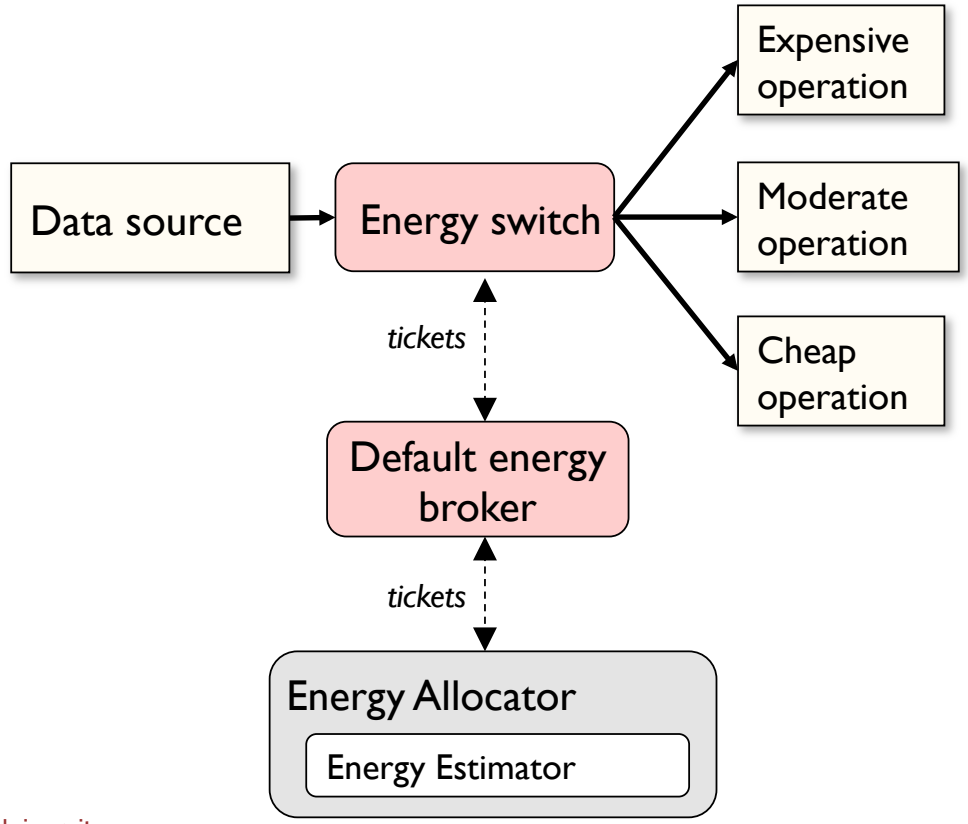
- Tracks state of each hardware device (CPU, radio, flash, sensors) in real time
- Accurate empirical model of hardware energy consumption (about 2-3% error)
- Low overhead, no hardware support required.



# Other Energy Brokers

## Energy-aware switch

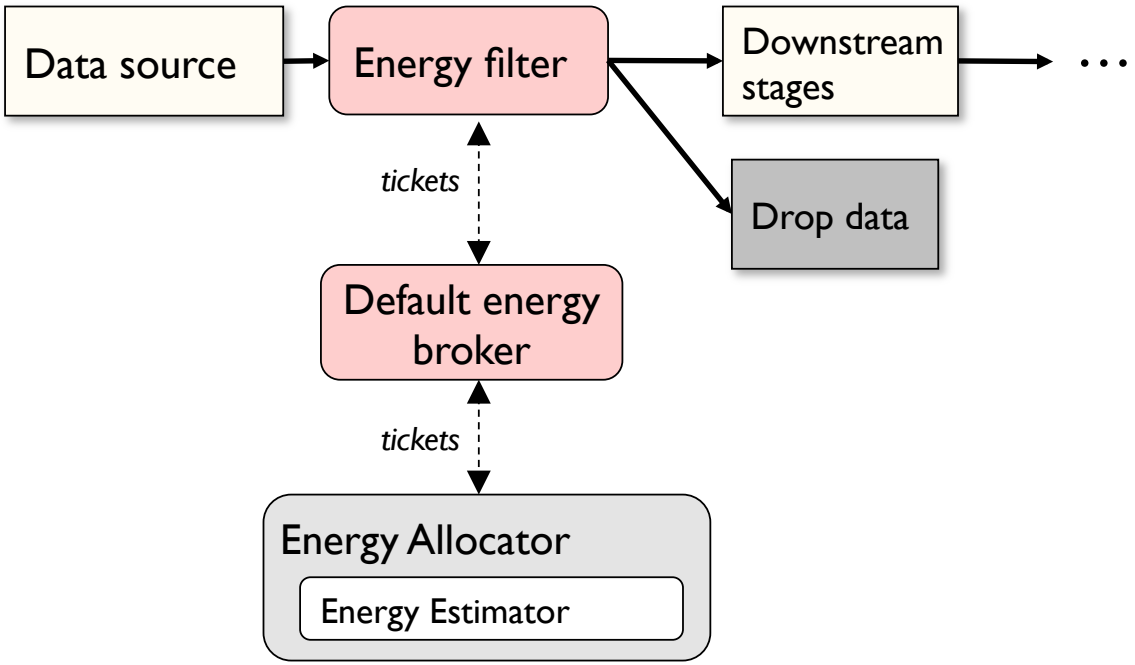
- Sends data down one or more downstream paths based on energy availability
- Mimics policy used by Eon *[Sorber et al., Sensys 2007]*



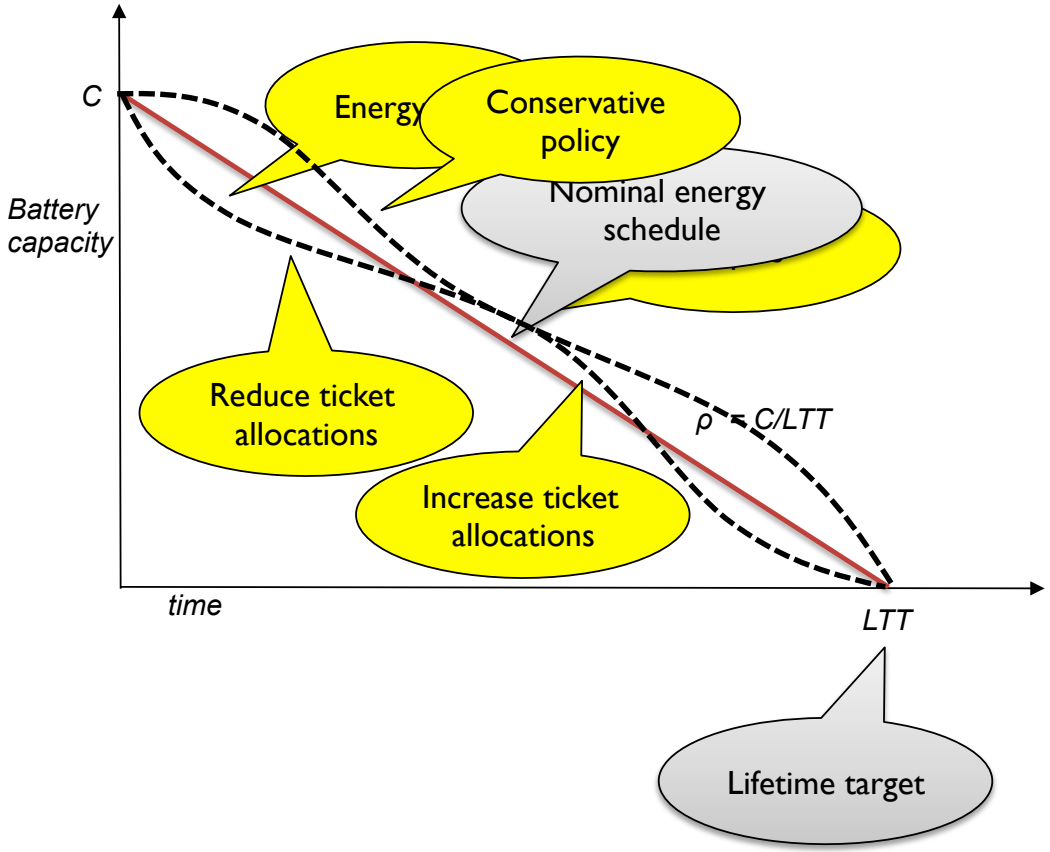
# Other Energy Brokers

## Energy-aware filter

- Selectively drops data to meet energy target
- Effectively controls scheduling and duty-cycling of downstream stages



# Example: Pixie Energy Broker



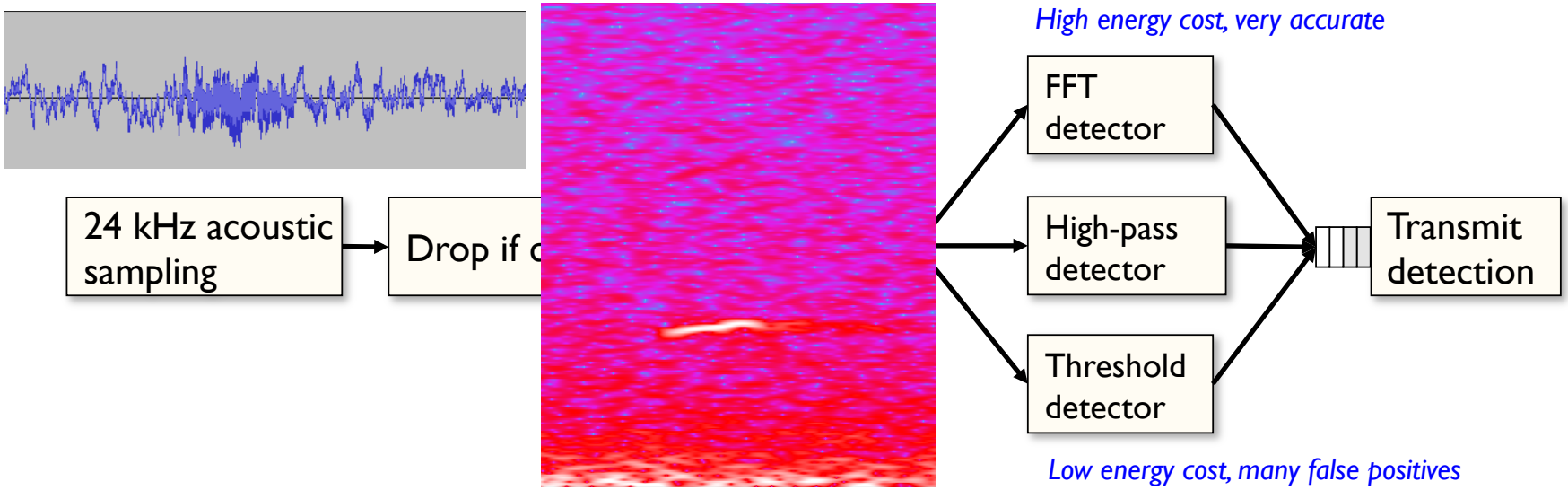
# Evaluation: Acoustic Target Detection

Detect acoustic signature of target (marmot call) in acoustic signal, subject to variable noise.

- *[Girod et al. SenSys'06]*

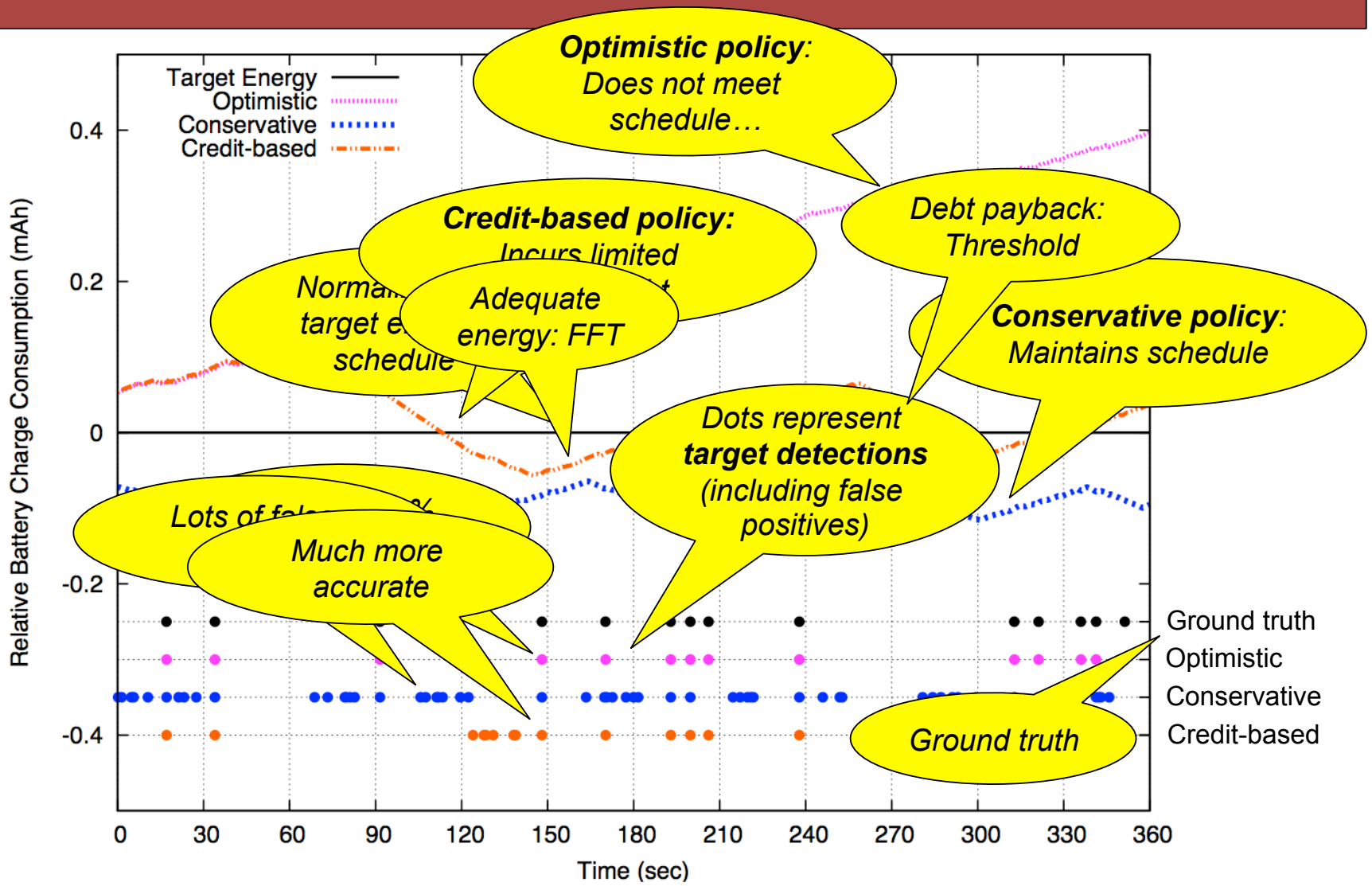
Three detection algorithms with increasing energy cost and accuracy

Goal: Maximize accuracy subject to battery lifetime target



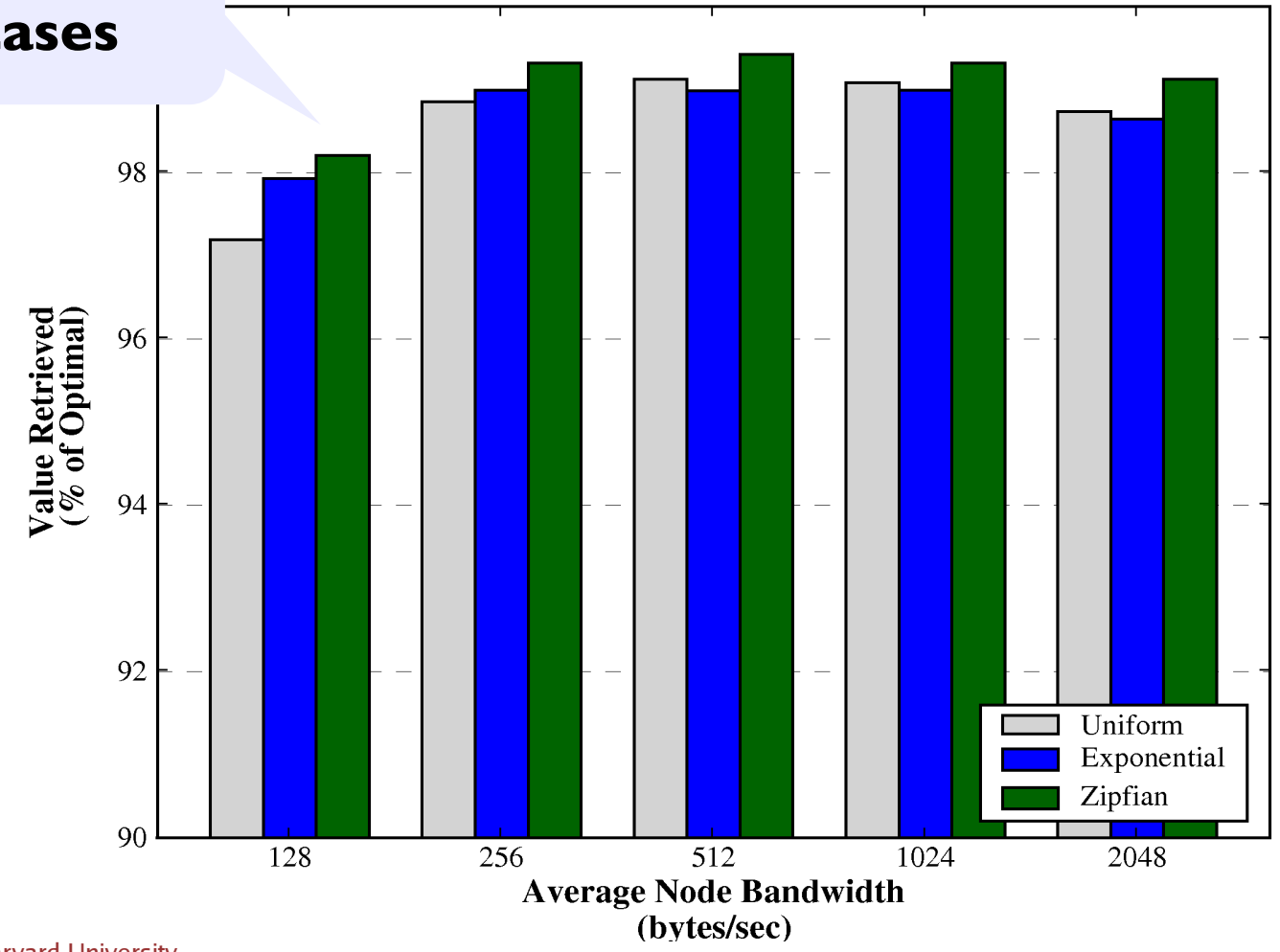
# Energy adaptivity: Varying policies

40-day lifetime target



# Varying Data Value Distribution

**Above 96% in all cases**

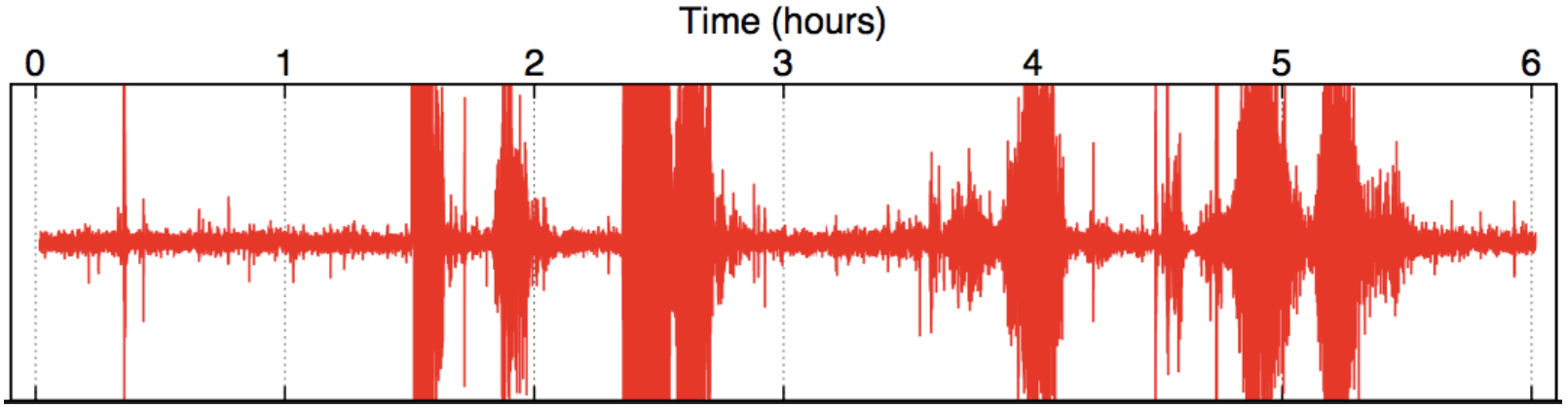


# Defining Data Value

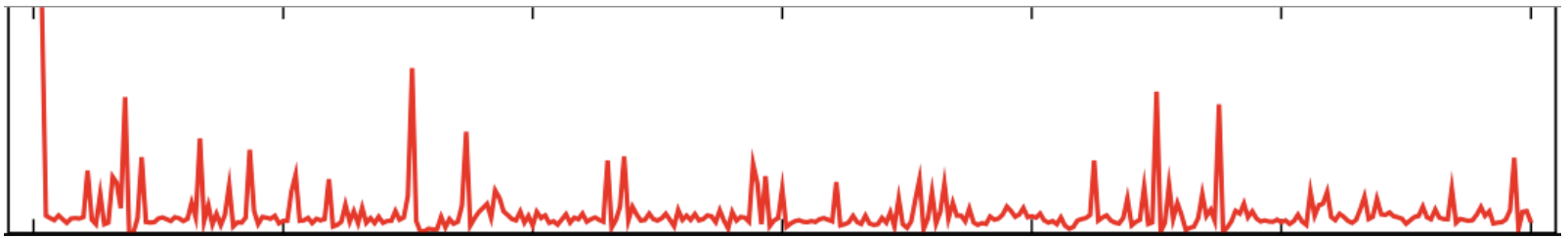
Assumption: nodes compute initial value for each ADU.

- This determination is inherently app-specific

*Raw signal*



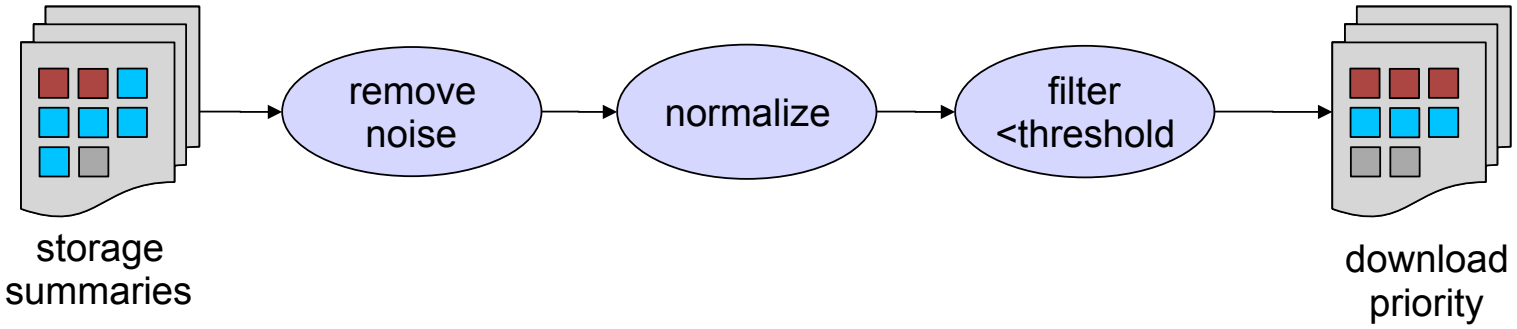

*Short-term average divided by long-term average*



# Lance Policy Modules

User-supplied functions to inspect, filter, or modify raw ADU priorities at the base station.

Composed into a linear chain:




- Take stream of ADU priorities as input, emit (possibly modified) priorities as output
- Can maintain internal state
- Must run efficiently (i.e., keep up with stream of ADU priorities from the network).

# Example Policy Modules

## Priority thresholding

- **filter**: Drop ADU if value below threshold

## Noise removal and calibration

- **adjust**: adjust raw value by adding or subtracting fixed offset
- **debias**: normalize data values across nodes

## Priority dilation

- **timespread**: assign high ADU values to ADUs adjacent in time
- **spacespread**: assign high ADU values to ADUs sampled by different nodes

## Cost weighting

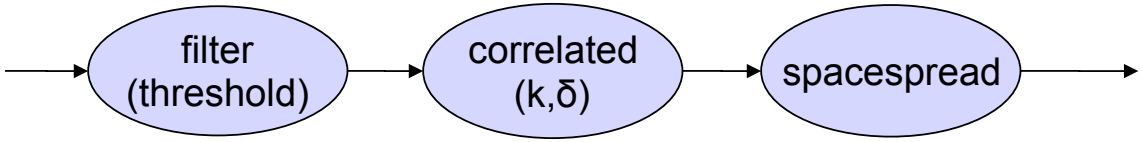
- **costweight**: scale ADU value by cost to download

# Example: Correlated Event Detection

**correlated** policy module W(k,δ)

- counts number of ADUs within a time window δ with a nonzero value
- if at least k ADUs match, retain ADUs in the window
- otherwise, drop this set of ADUs

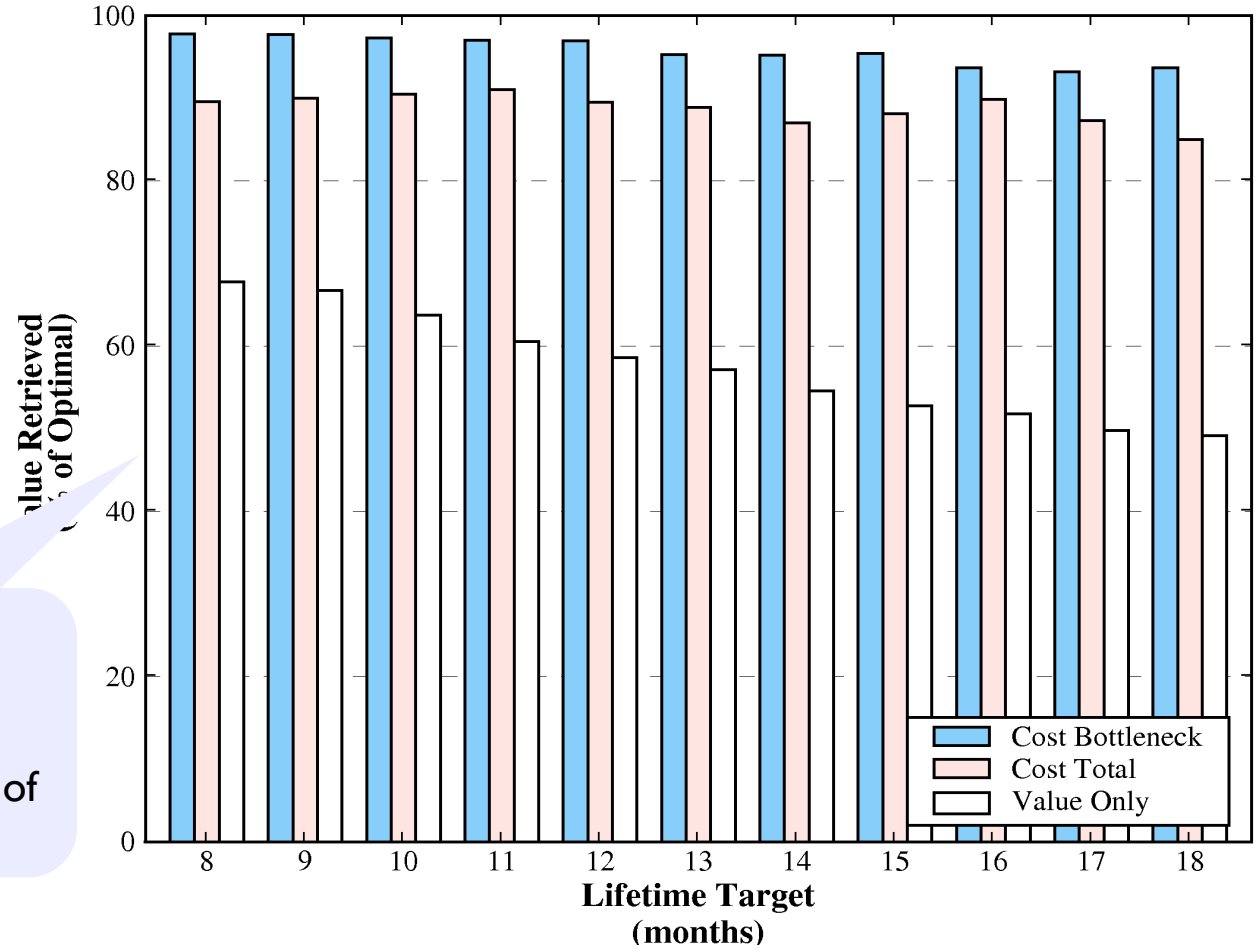
Network-wide earthquake detection in Lance:




- Policy implemented at the base station, rather than on motes.
- Easy to modify behavior of network just by changing policy modules.

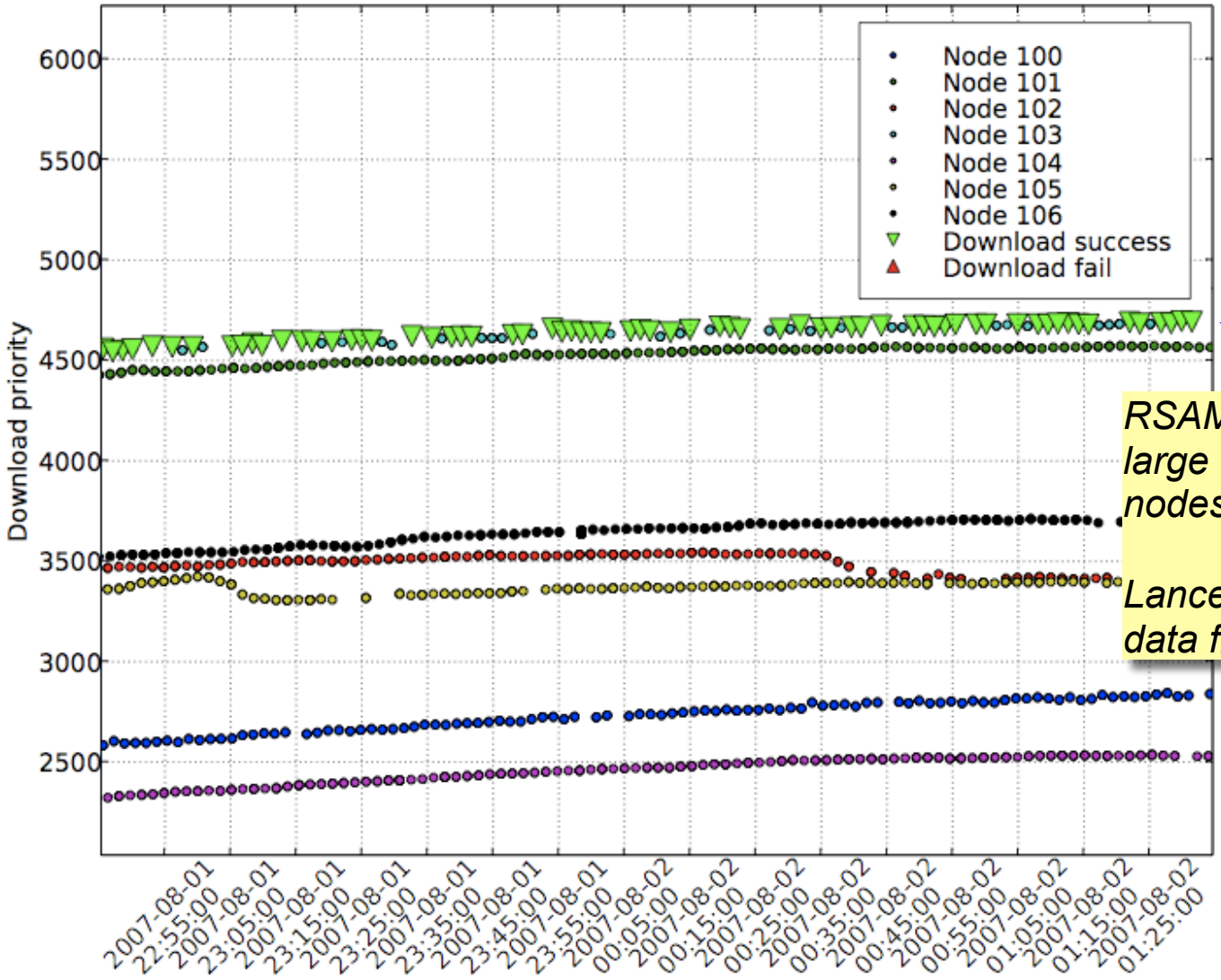
# Results under varying lifetime target

25-node tree, Zipf value distribution




Cost bottleneck achieves > 95% optimality across a wide range of target lifetimes

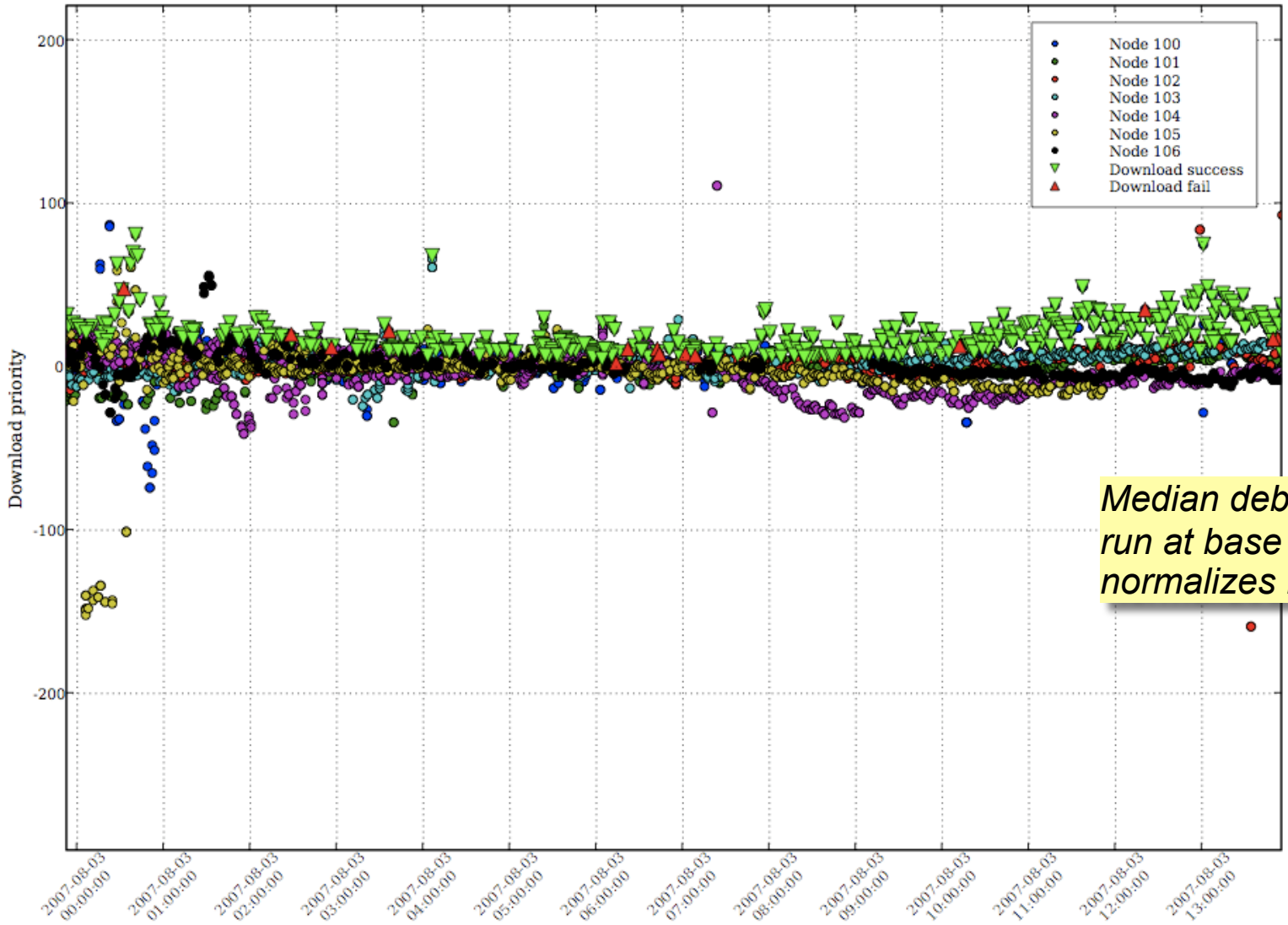
# RSAM prioritization: DC bias




*RSAM values exhibited large DC bias across nodes.*

*Lance only downloaded data from 1 or 2 nodes!*

# The fix: Debiasing policy module




*Median debiasing filter run at base station normalizes RSAM values*

# Programming Benefits

Dataflow programming model maps well onto application structure

- Similar to Eon, Tenet, WaveScope, and Flask

Tickets offer a great deal of flexibility and power

- Fine-grained feedback and control of resource usage at all levels

Brokers decouple resource management policies from mechanisms

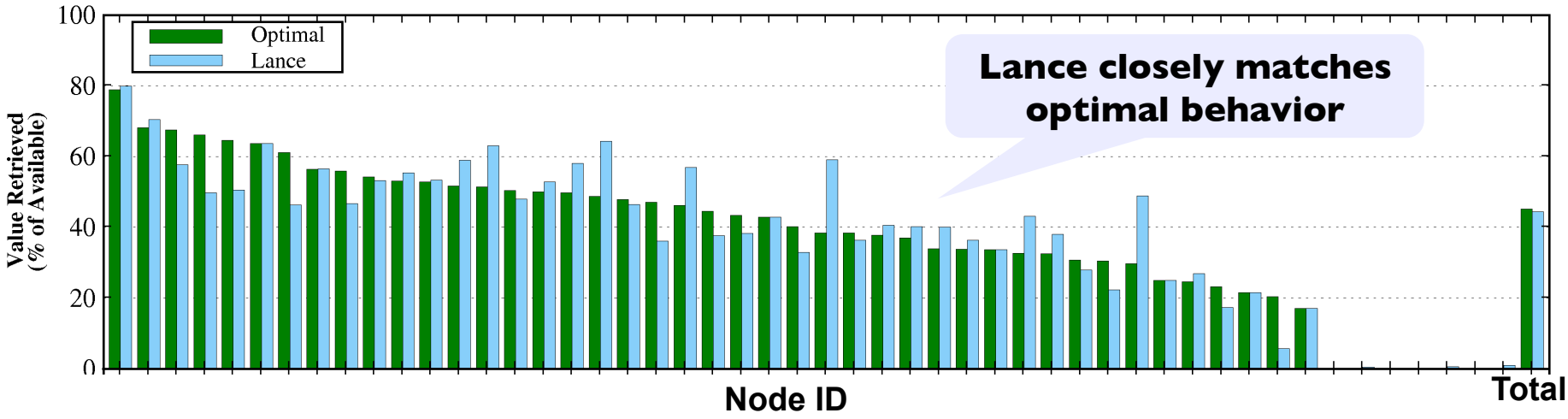
- Enables composable resource adaptations

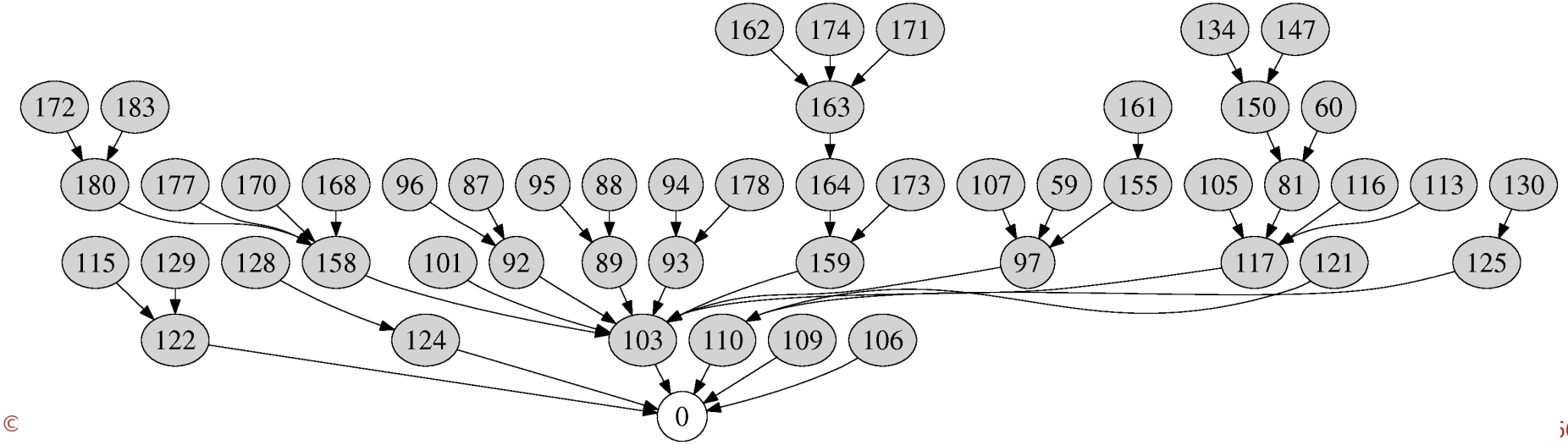
Resource-awareness is pervasive in the programming model

- Not “off to the side”
- Annoying at first, but makes sense when you get used to programming in this way
- Exposes resource dependencies, rather than relying on hidden magic knobs

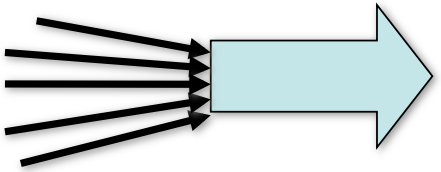
# MoteLab Testbed Experiments

50 node tree topology, Zipf value distribution



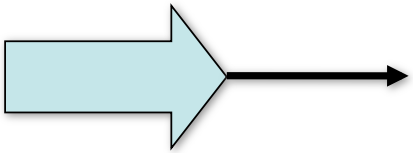
# Some Region Primitives



```
everywhere : α stream → α region
gossip     : int → α stream → α region

rmap : (α → β) exp → α region → β region
rfilter : (α → bool) exp → α region → α region

rfold   : (α *β → β) exp * β exp →
            α region → β stream
rfunnel : α region → α stream
```



# Future Directions: Peloton

What about coordinated resource management within the network?

- Example: Group of nodes detect an event, elect leader to collect signals, process data, deliver results to the base station.
- Nodes need to **share** information on resource availability, **coordinate** resource allocation decisions, and represent **cross-node allocations**

*Peloton* – A Distributed OS for resource-aware programming

- Decentralizes resource management within the network
- Extend Pixie's abstractions to span allocations on multiple nodes.
- [HotOS 2009]

# Future Directions: RoboBees

NSF Expeditions in Computing

My group: Swarm OS for distributed resource management and programming models