Resource Aware Programming for Sensor Networks

Matt Welsh
mdw@eecs.harvard.edu

Harvard
School of Engineering
and Applied Sciences

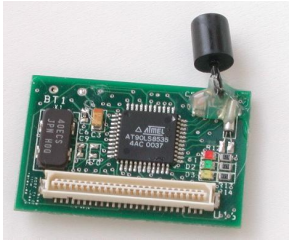
Sensor Networks: A New Computing Platform

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

WeC (1999)

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 (light, temp, humidity)
- Cost: Around $75

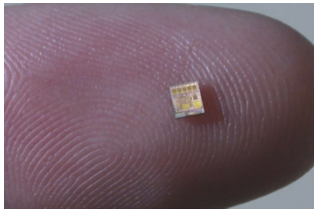


Rene (2000)

Designed for low power consumption:

- 25 mA active (CPU+radio); 1.8 uA sleep

Telos (2004)

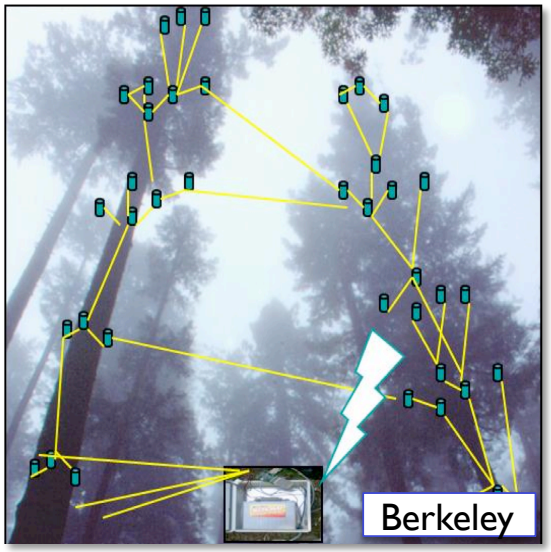


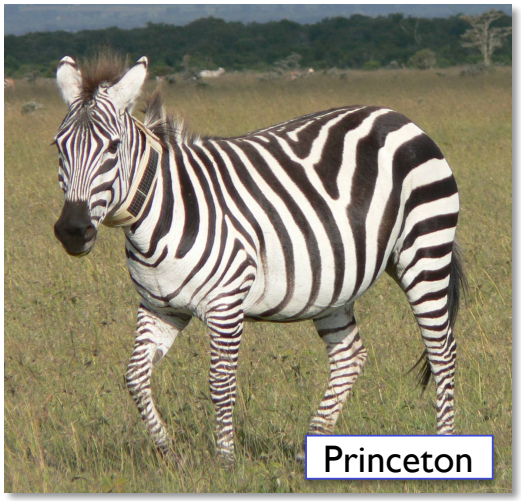
Speck (2003)

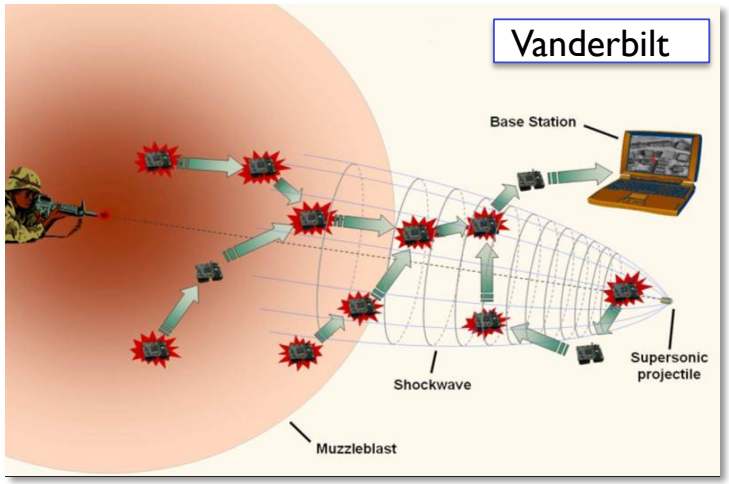
MICA (2002)

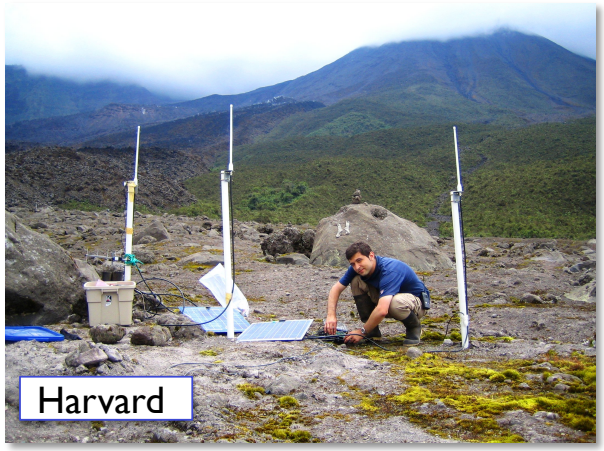
Dot (2001)

Lots of exciting application domains…





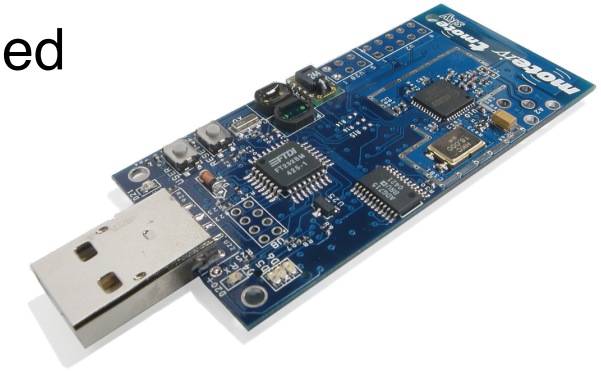




Key Challenge: Resource Management

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.

Talk roadmap

Two application vignettes.

Elucidation of key challenges.

Pixie: An OS design for resource-aware programming

Lance: A framework for network-wide resource management.

Future vision and wrap-up.

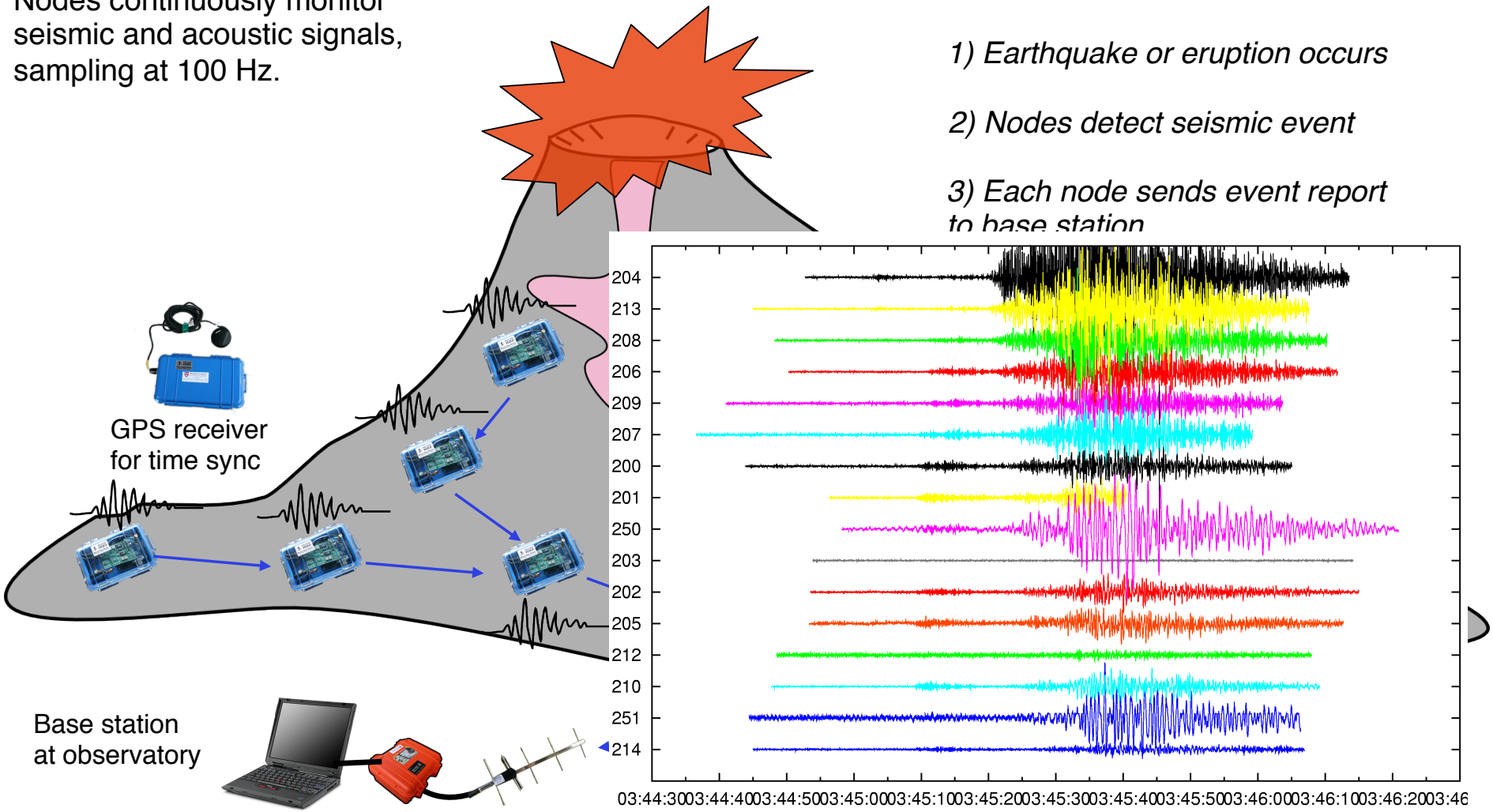
Application Vignette: Wireless Sensors for Volcanic Monitoring

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

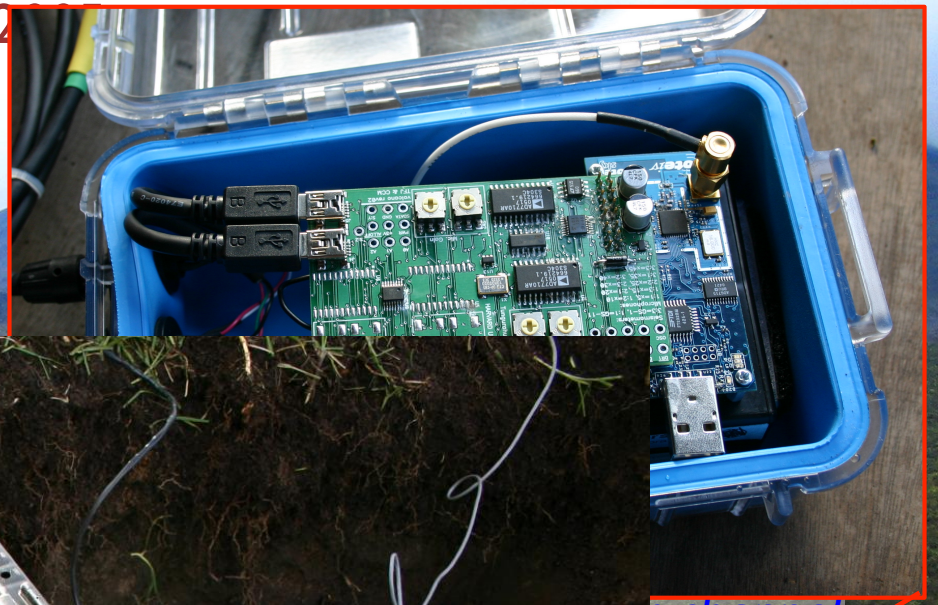
1) Earthquake or eruption occurs

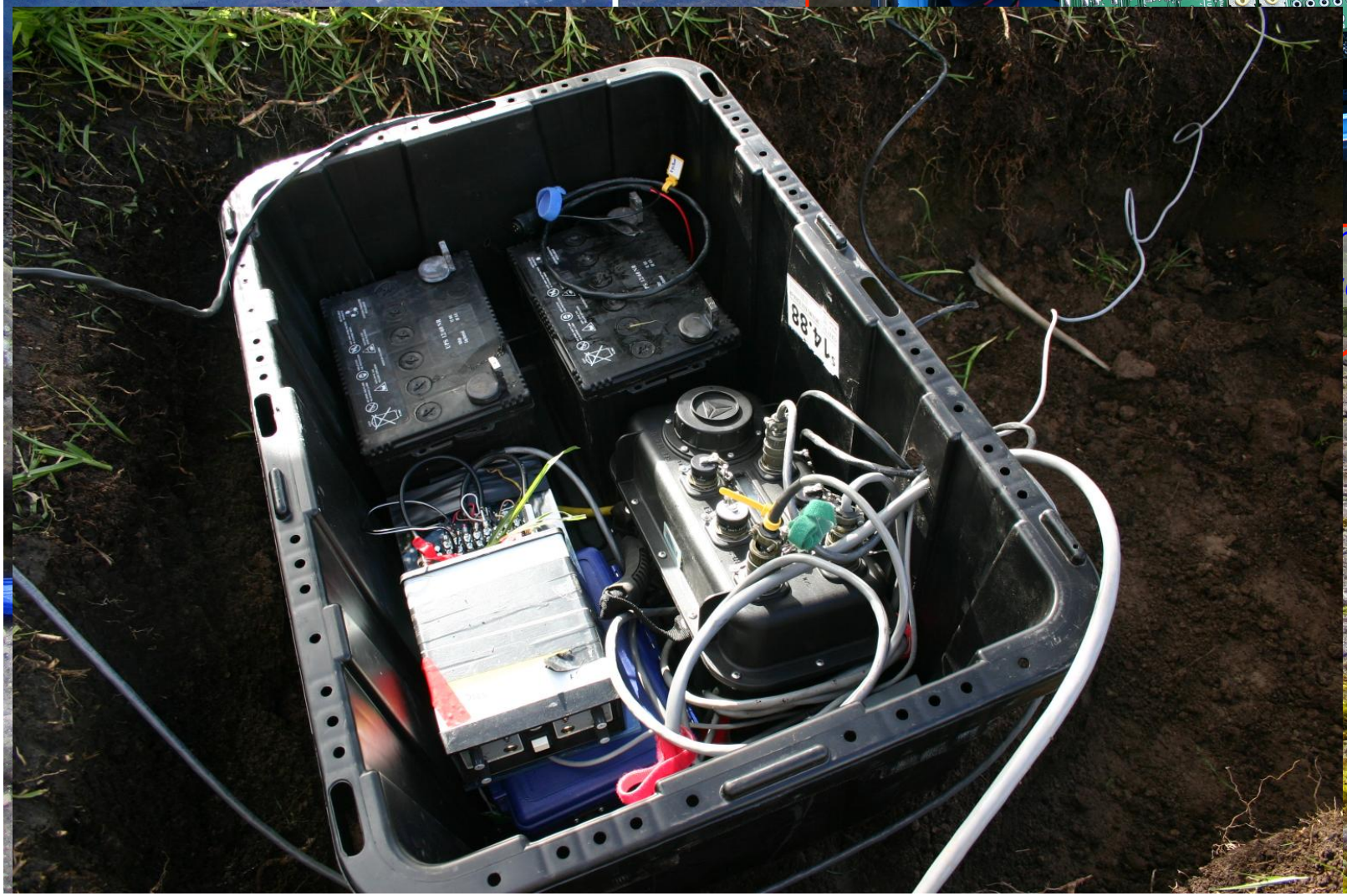
2) Nodes detect seismic event

3) Each node sends event report to base station



Reventador Volcano, Ecuador, August 2

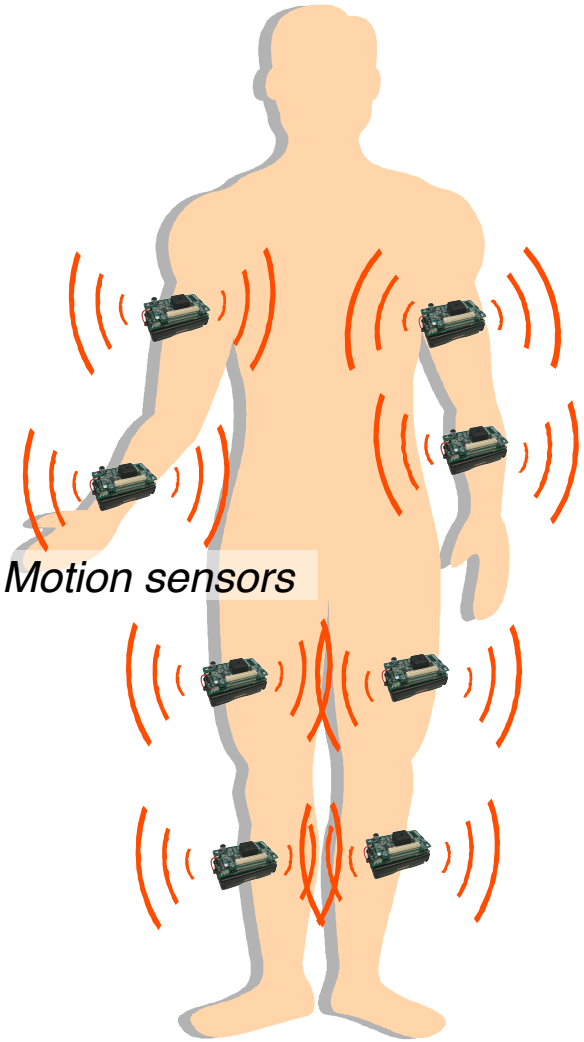




-channel
or node

Parkinson's Disease, stroke, and epilepsy monitoring

with P. Bonato, Spaulding Rehabilitation Hospital



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

Things to notice about these applications...

High data rates with fine-grained time synchronization requirements

- 100 Hz sampling rate, multiple channels per node
- Timing accuracy is paramount to support signal processing!

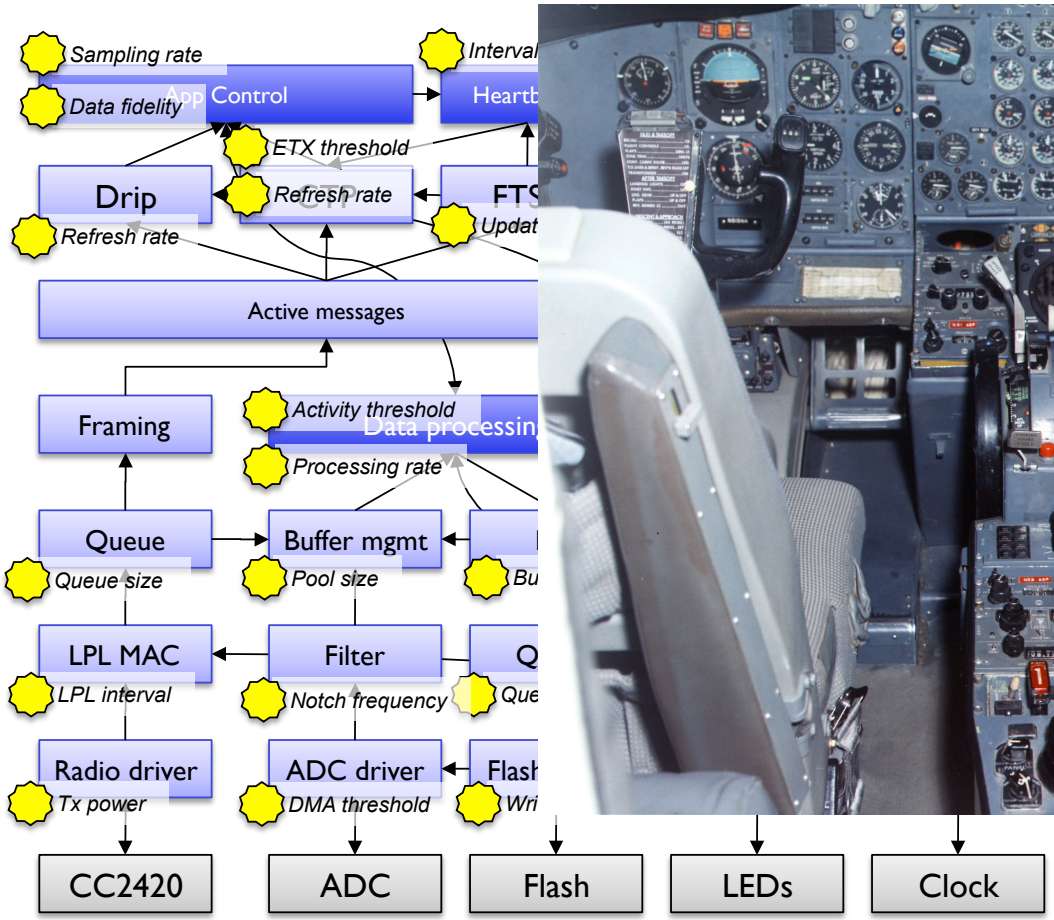
Fairly complex **domain-specific** processing:

- P-wave arrival computation, velocity model to extract earthquake locations
- Domain-specific feature extraction and classification of motion sensor data

Applications should **adapt** to changing resource availability

- e.g., Variations in sensor data, changing energy reserves, or fluctuations in radio bandwidth
- Adaptation is also highly application-specific

Resource tuning in TinyOS



What we really want

Our Approach: The Pixie Operating System

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

- Resources as a first-class programming primitive
- A **resource aware programming model** based on **resource tickets** and **brokers**

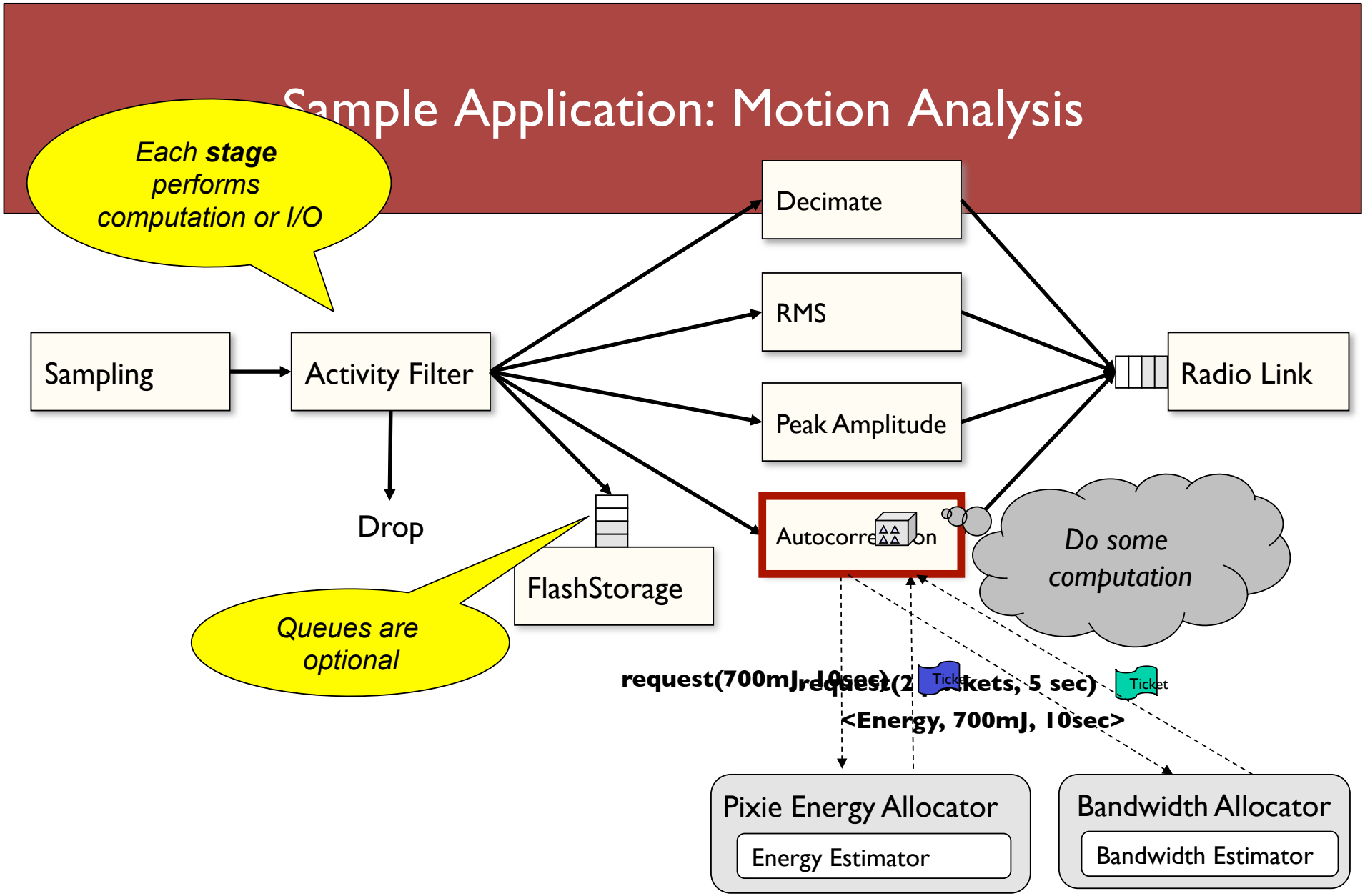
Pixie is intended to make it easy to write adaptive applications

- 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

Core abstraction for resource management in Pixie

- Ticket **<R, c, t_e>** represents a *revokable right* to consume **c** units of resource **R** until the expiry time **t_e**.
- Think of as a short-term "reservation" for some resource.
- Tickets decouple resource request from usage – permits planning.

Basic resource ticket operations:

- *Redeem*: Consume resources indicated by ticket
- *Forfeit*: Give up ticket
- *Revoke*: Reclaim resources (can happen prior to expiry time!)
- *Join*: Combine two tickets into one
- *Split*: Split a ticket into two separate tickets

Granularity depends on resource variability

- e.g., Bandwidth tickets would have a short expiry time; storage tickets need not expire.

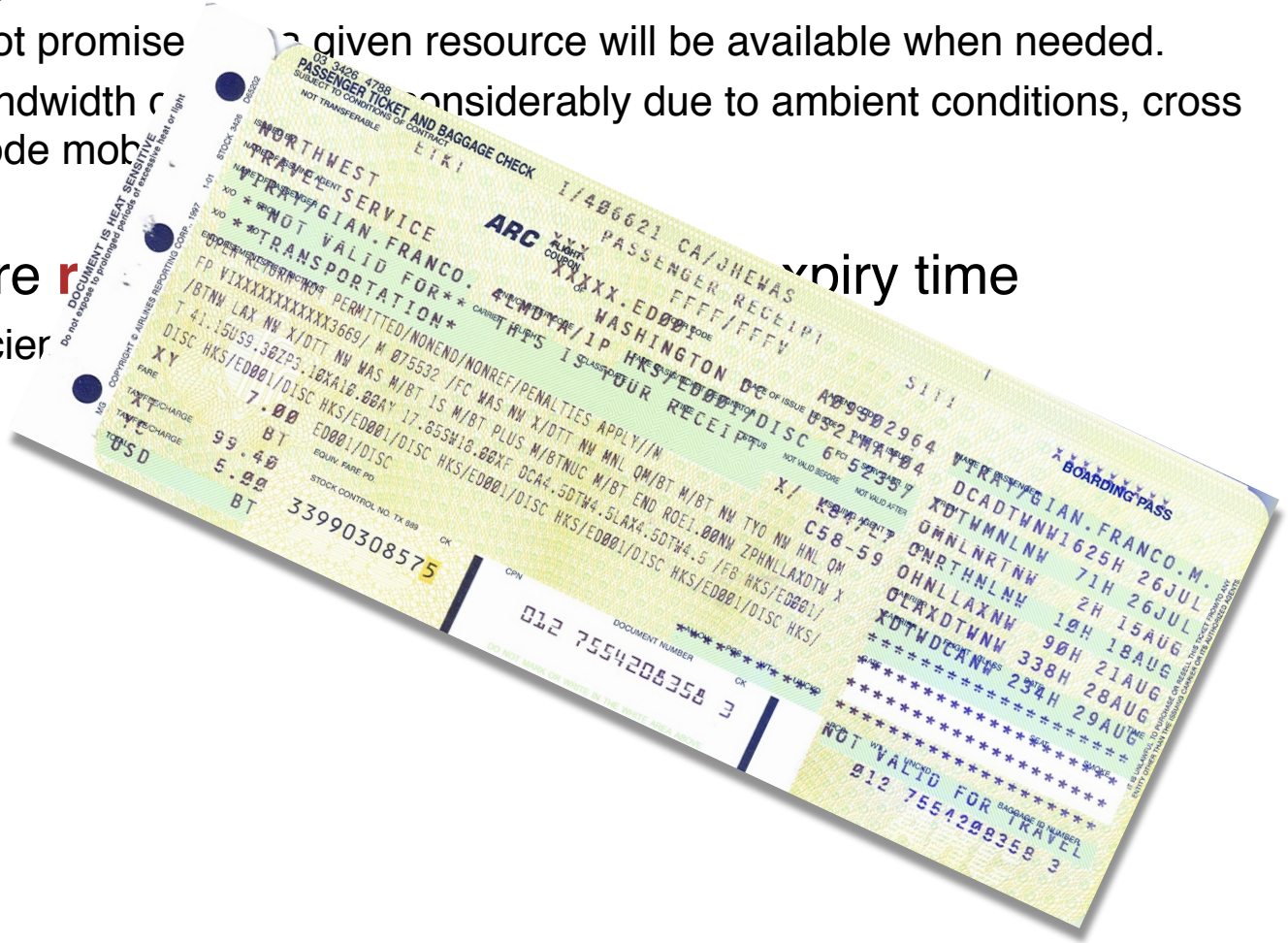
Tickets are not guarantees!

Making strict guarantees about resource reservations is difficult:

- System cannot promise [illegible] given resource will be available when needed.
- e.g., radio bandwidth [illegible] considerably due to ambient conditions, cross traffic, and node mob[illegible]

Pixie tickets are r[illegible]xpiry time

- Trade off efficien[illegible]



Resource Allocators

Each physical resource has a corresponding **allocator**

- Allocators for energy, memory, flash storage, and radio
- Implicit allocator for CPU – the scheduler

Allocators estimate available resource, and allocate tickets.

- **No policy**: Always allocate ticket if the resources are available
- Policies handled by higher-level abstractions (e.g., brokers)

Allocators also enforce ticket redemption

- e.g., To transmit a packet must have corresponding bandwidth ticket

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**

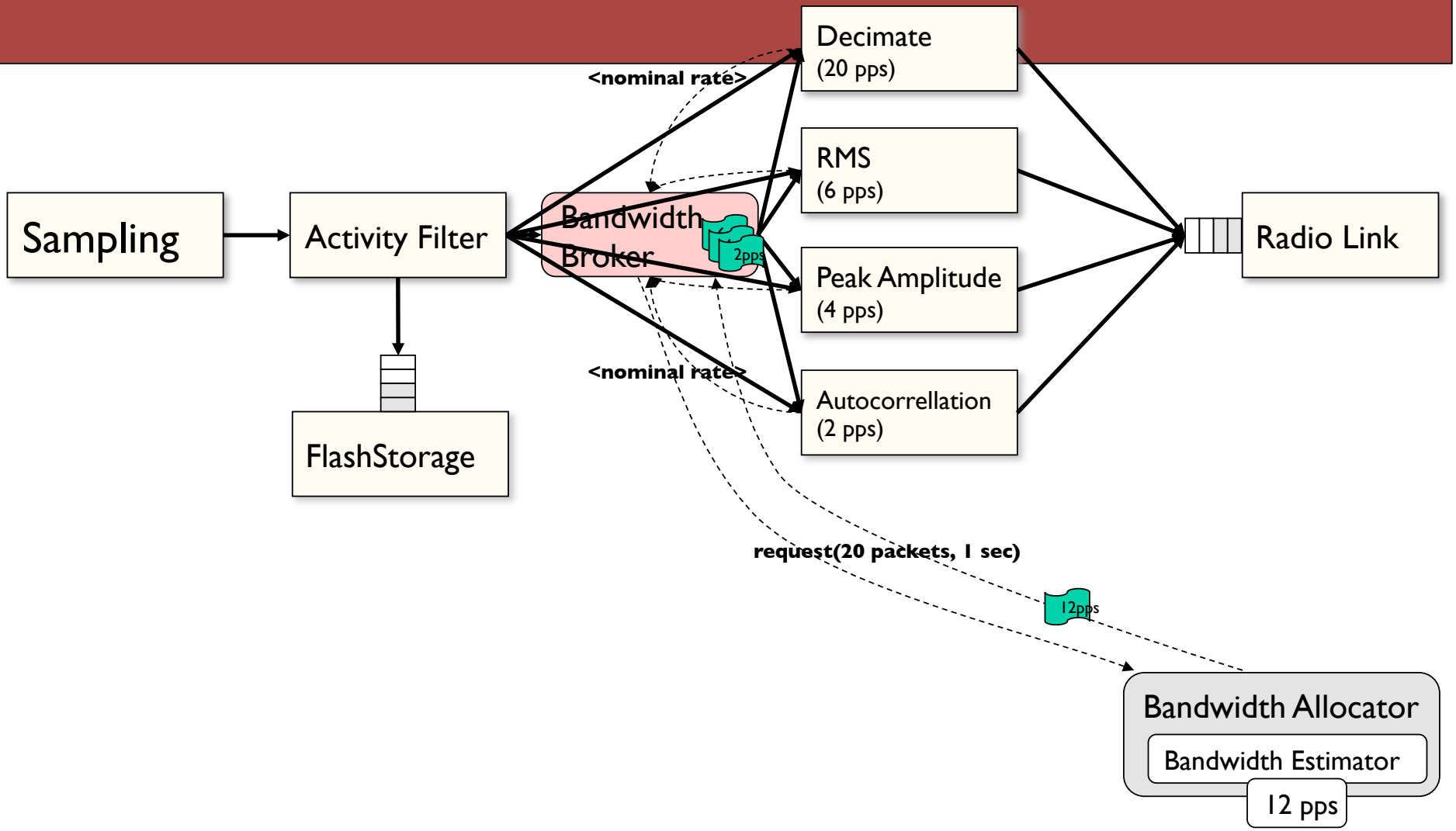
- Brokers request and manage resource tickets on behalf of the application
- Each broker implements some policy and provides a high-level API to the app

Brokers are specialized stages

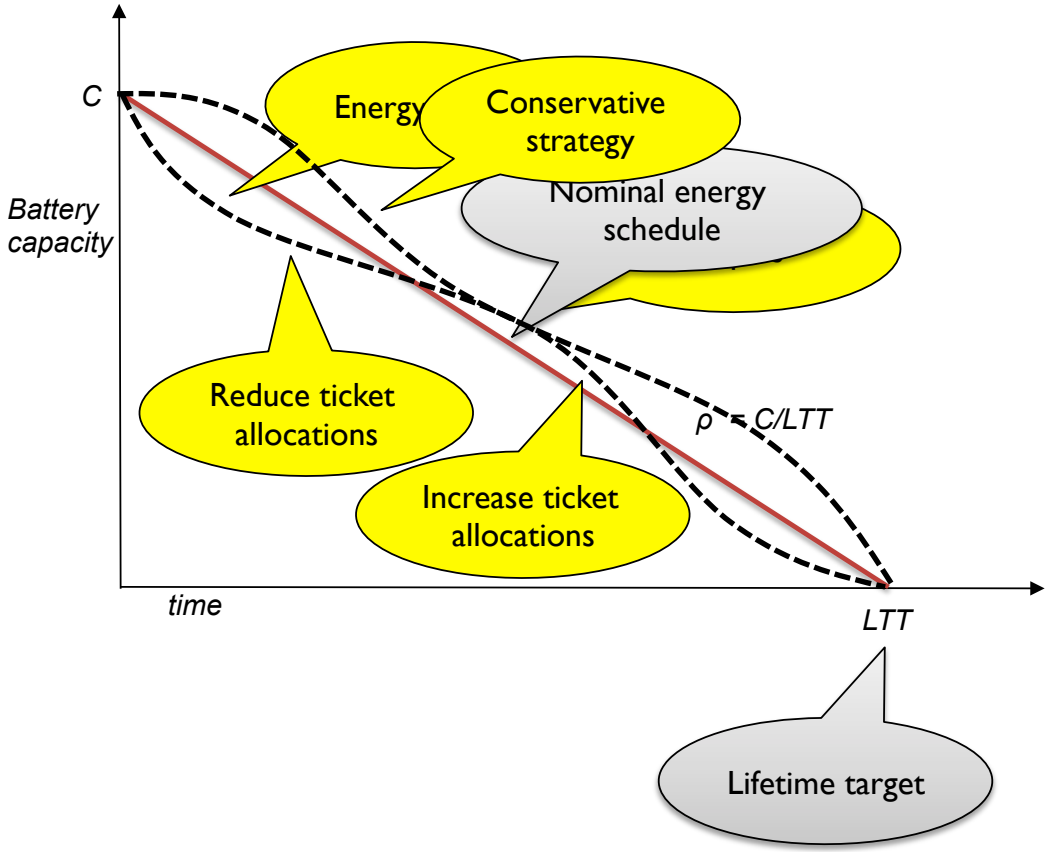
- Can interpose directly on the application's dataflow path
- Can inspect, redirect, discard, or refactor data

Pixie provides a small library of standard brokers

Bandwidth Broker



The Pixie Energy Broker



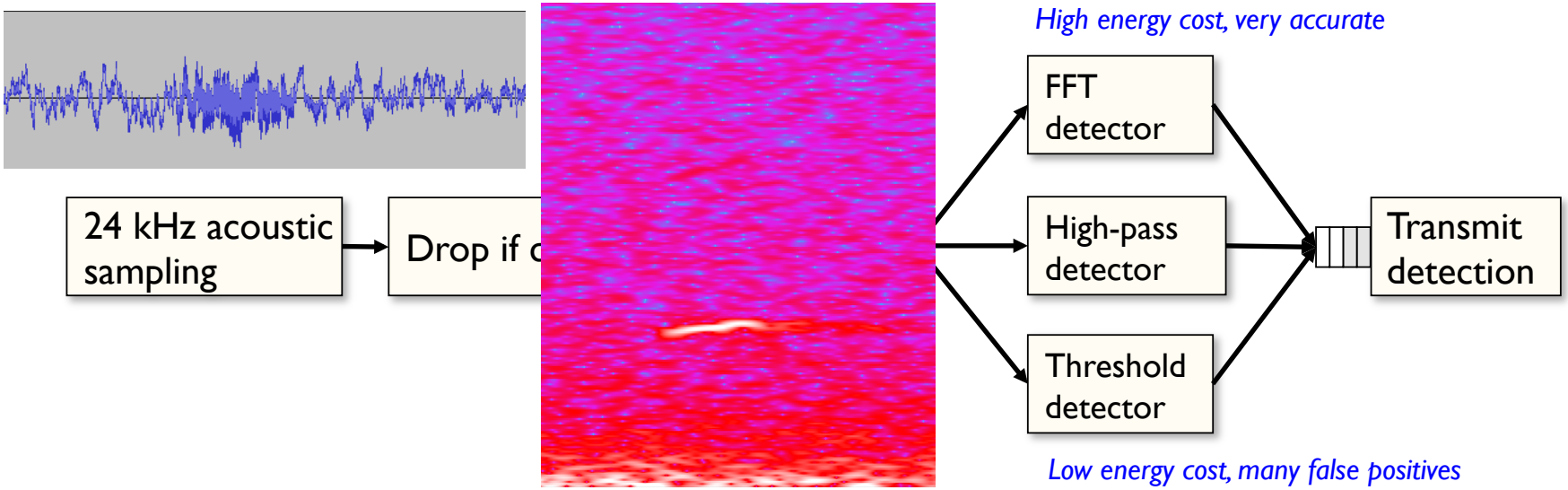
Evaluation: Acoustic Target Detection

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

- Similar to ENSBox *[Girod et al. Sensys'06]* – but no ranging

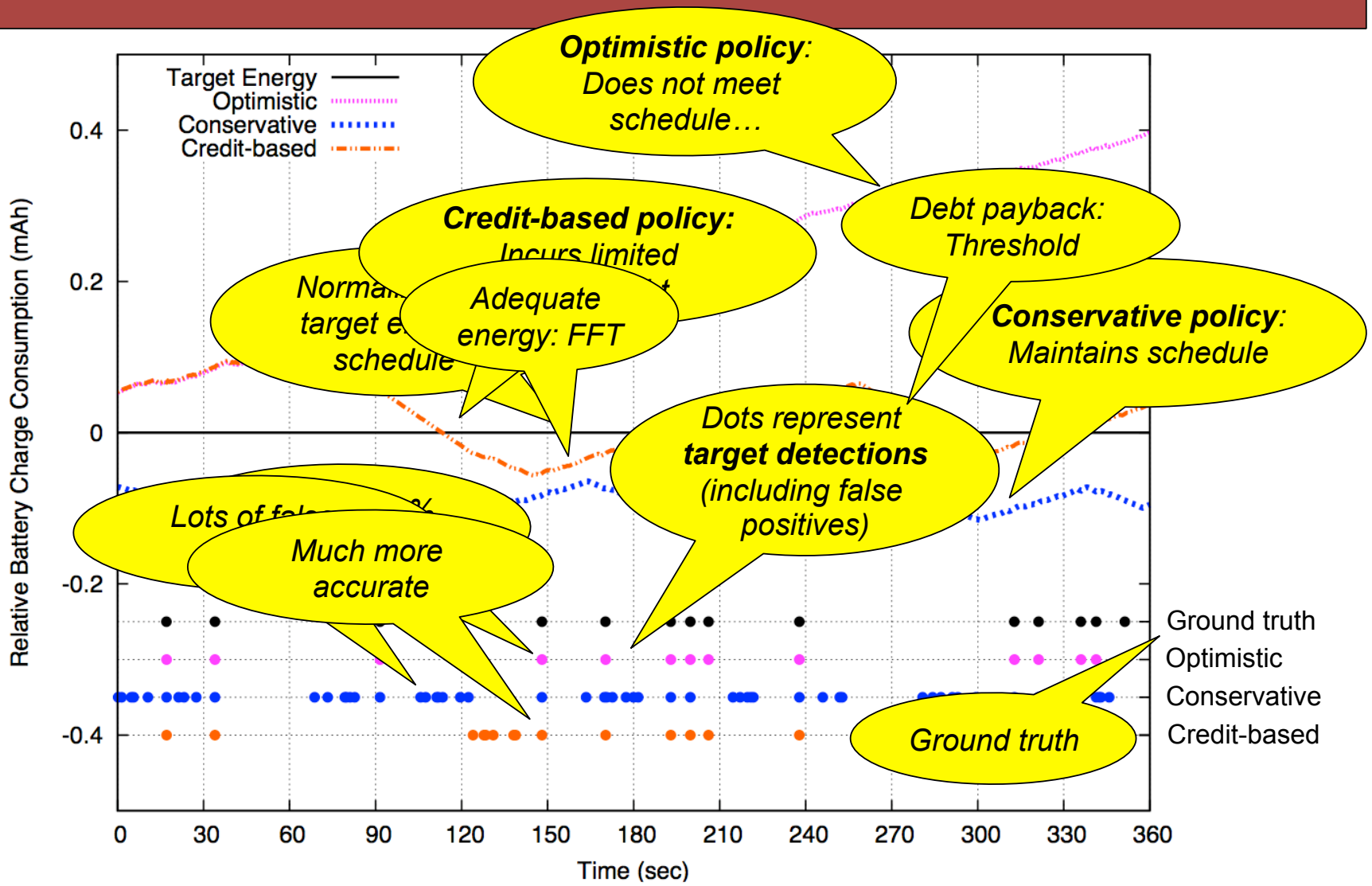
Three detection algorithms with increasing energy cost and accuracy

Goal: Maximize accuracy subject to battery lifetime target



Energy adaptivity: Varying policies

40-day lifetime target



Network-Wide Resource Management

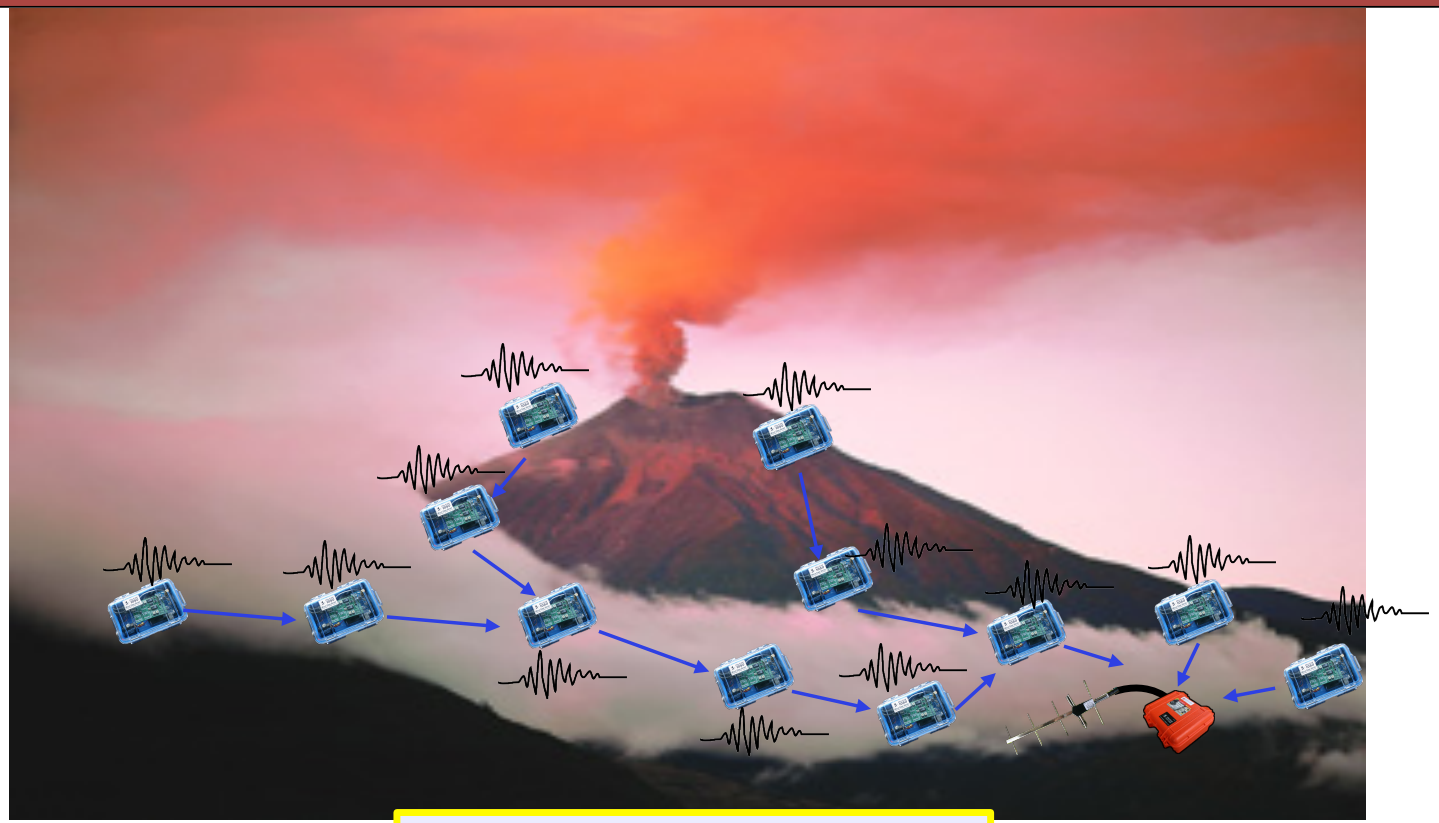
Large sensor networks constrained in two important ways:

- 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

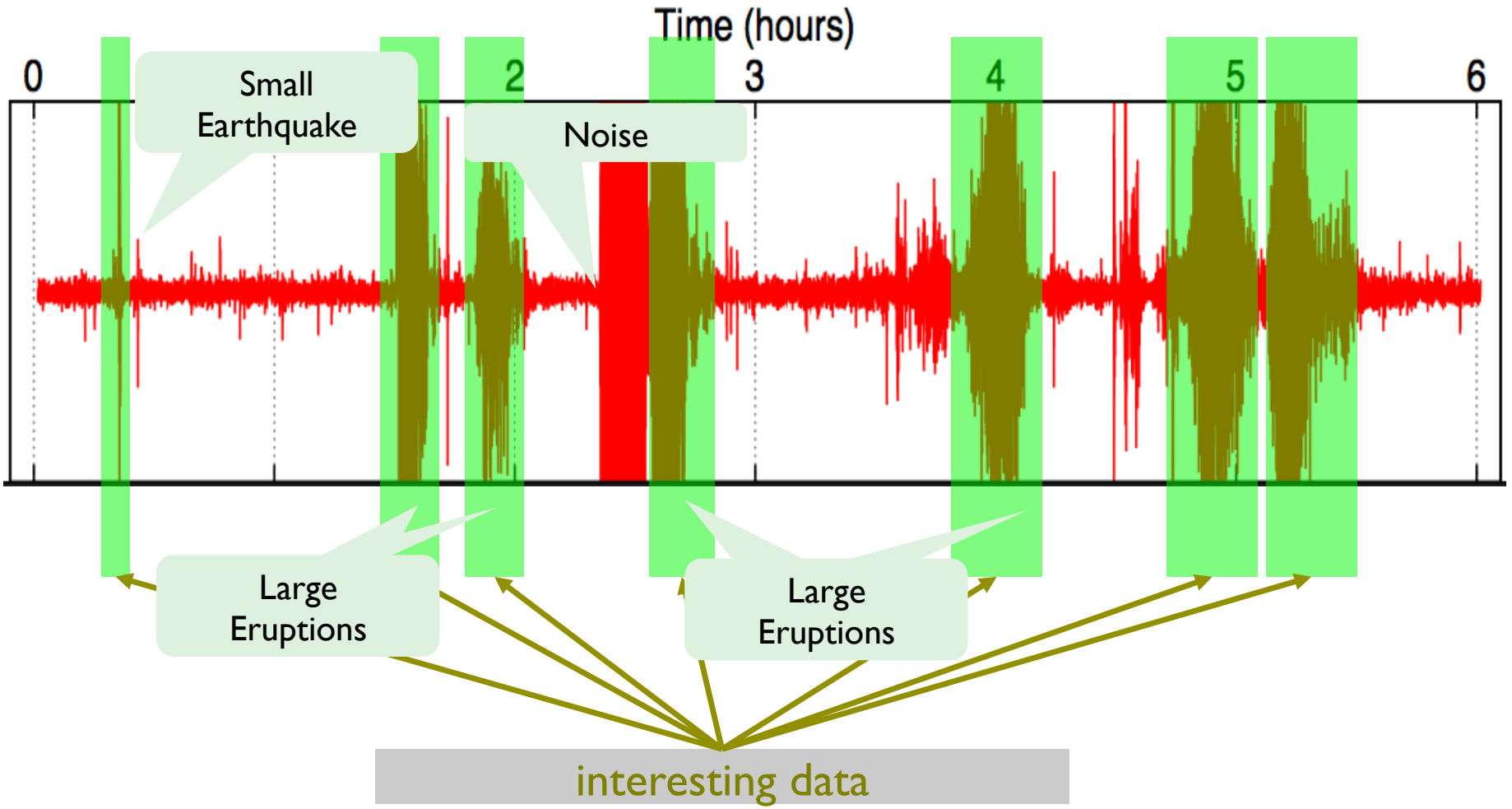
The Problem



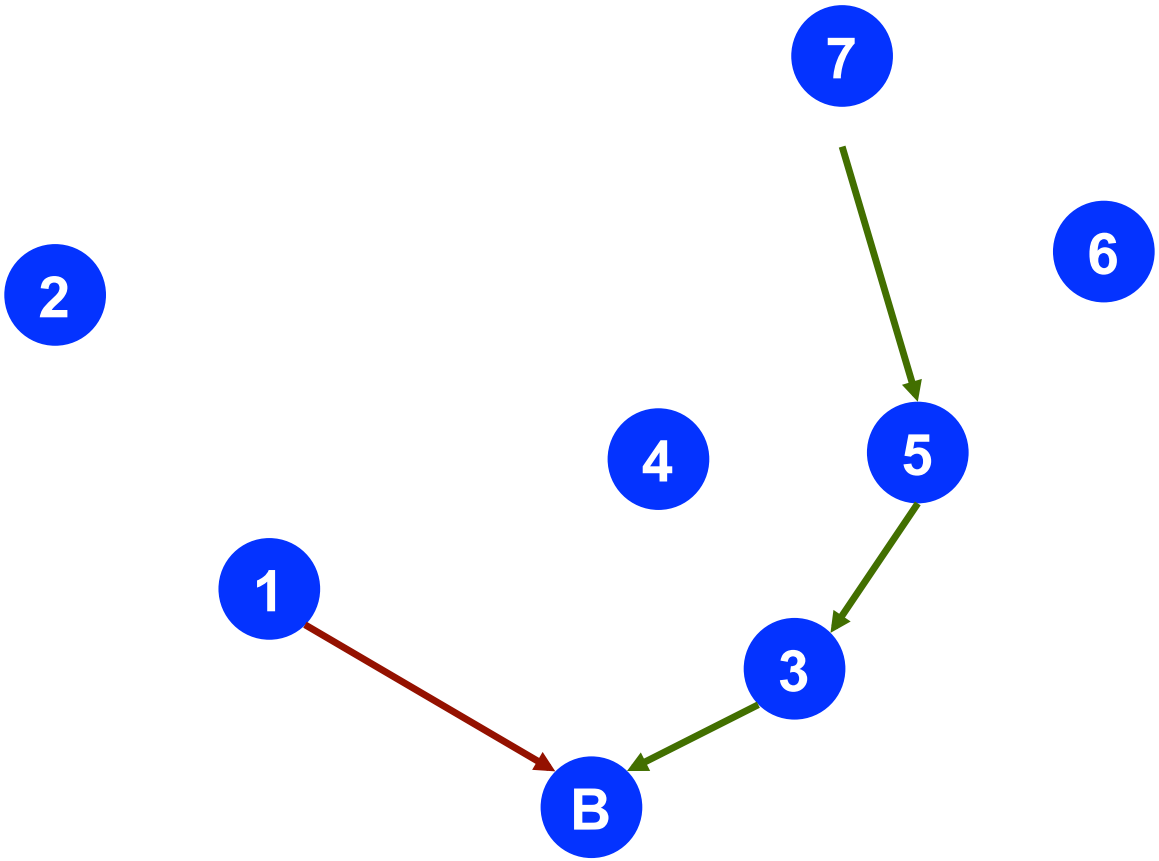
4800 b/s > 512 b/s

data generation rate > available network capacity

Observation: Data Differs in Value



Observation: Data Differs in Cost



Bandwidth to Node 1:
2048 Bytes/sec
Energy cost: 522 mJ

Bandwidth to Node 7:
512 Bytes/sec
Energy cost: 6106 mJ

Our Approach: Lance

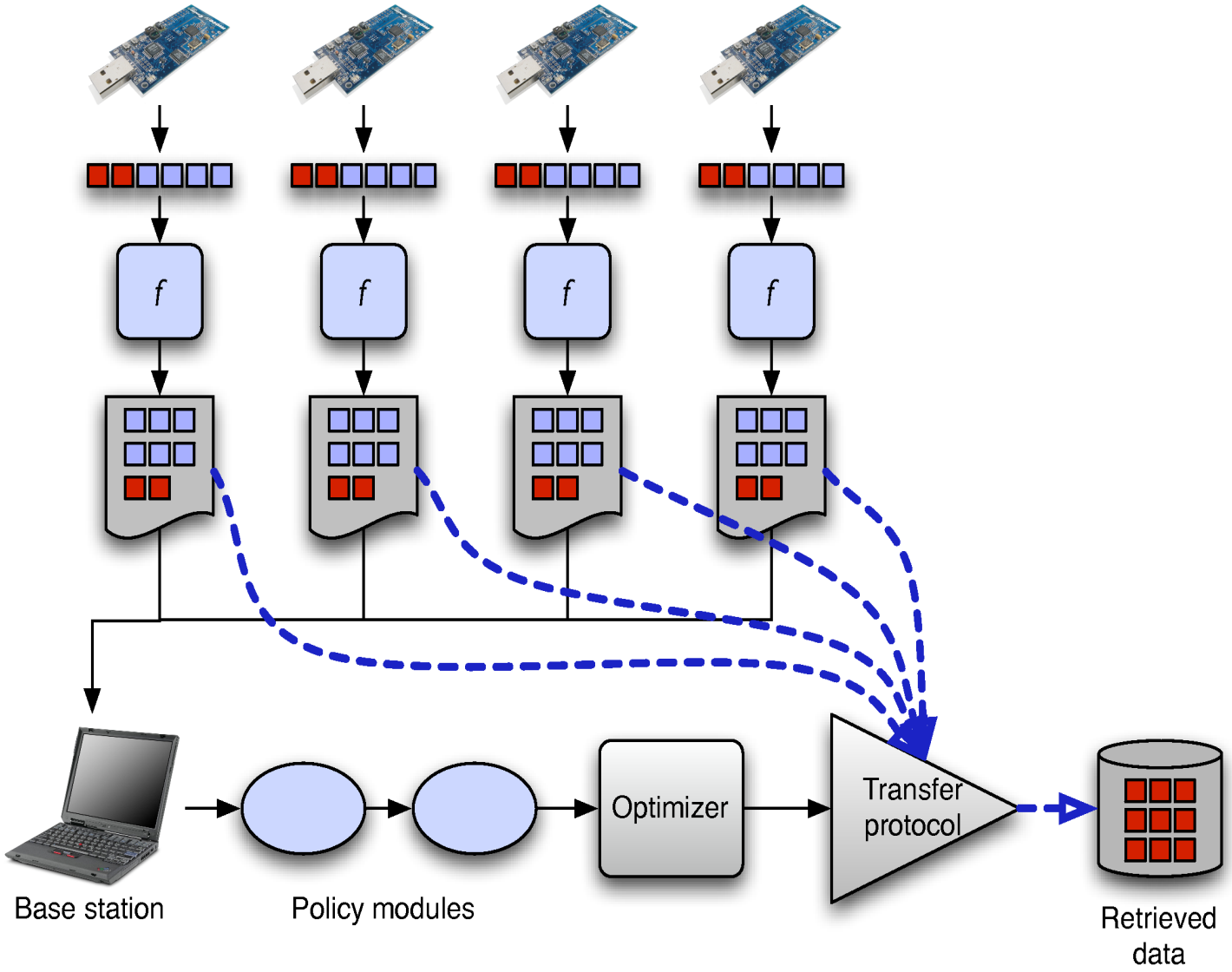
System for priority driven allocation of bandwidth and storage resources within a sensor network

Goal: Optimize data quality subject to energy and bandwidth constraints

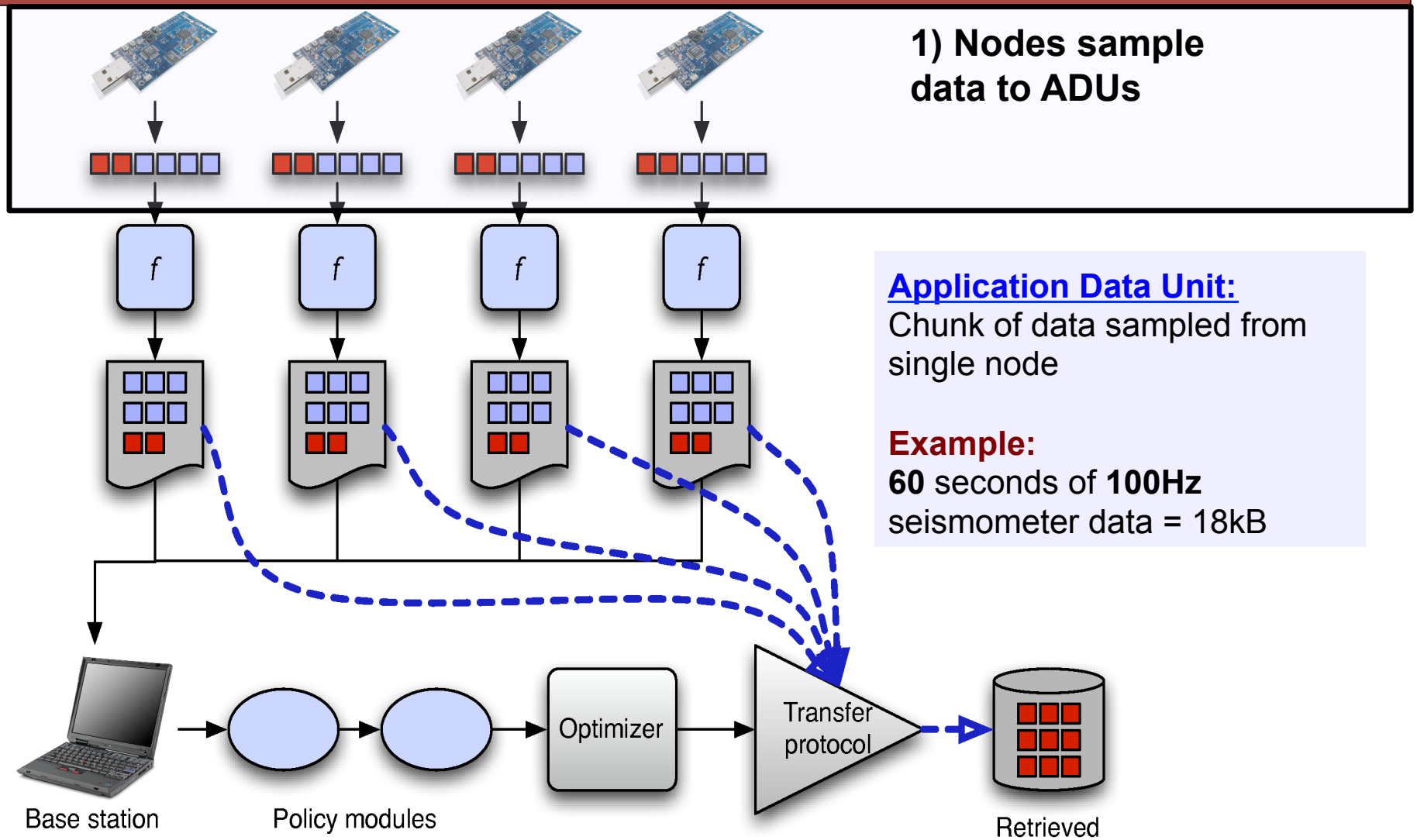
Lance decouples mechanisms for resource allocation from app-specific policies

- App-supplied **policy modules** permit a wide range of allocation policies to be implemented
- Can target many optimization metrics: priority maximization, fairness, spatial/temporal data distribution, and more

Lance System Architecture



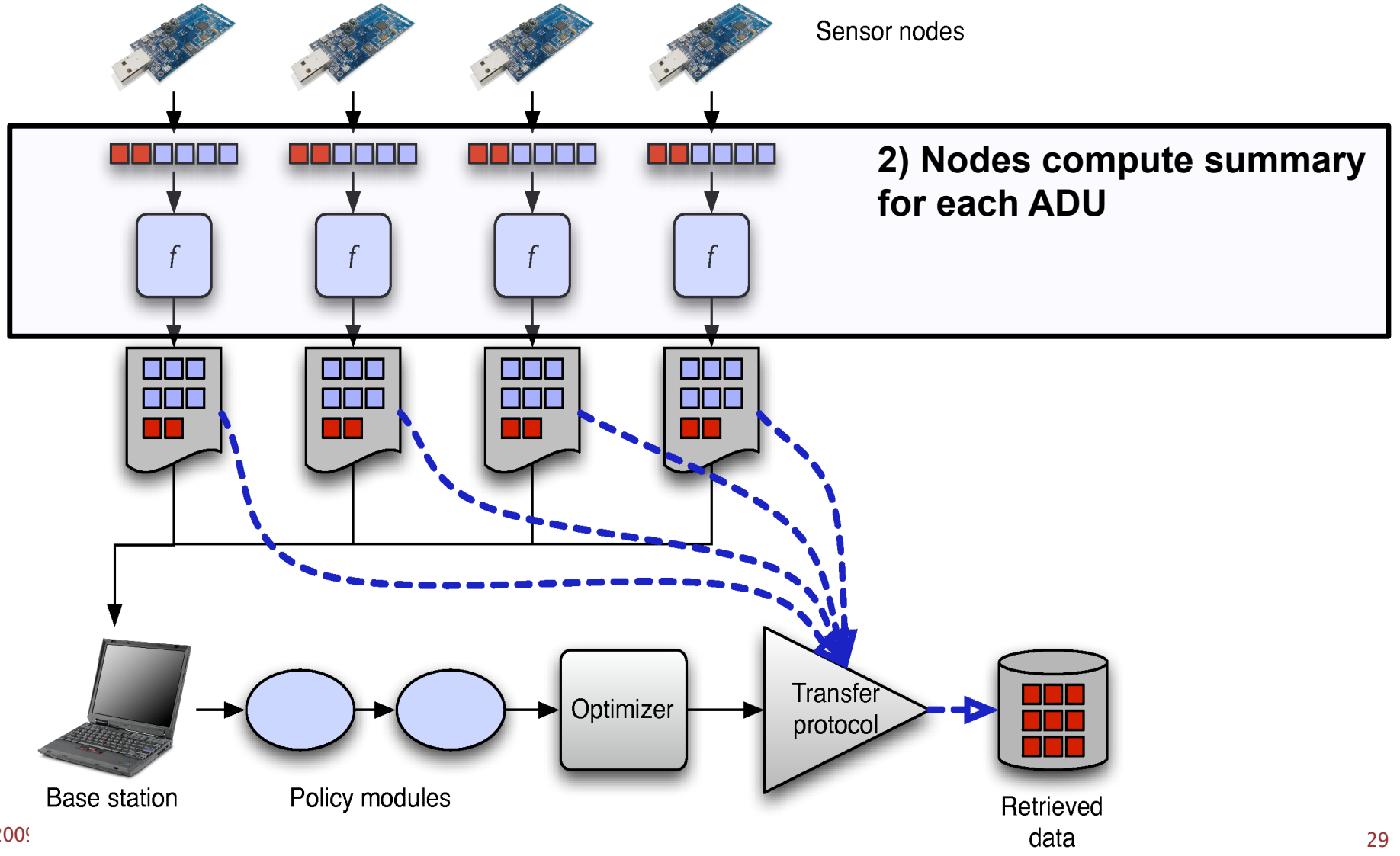
Lance System Architecture



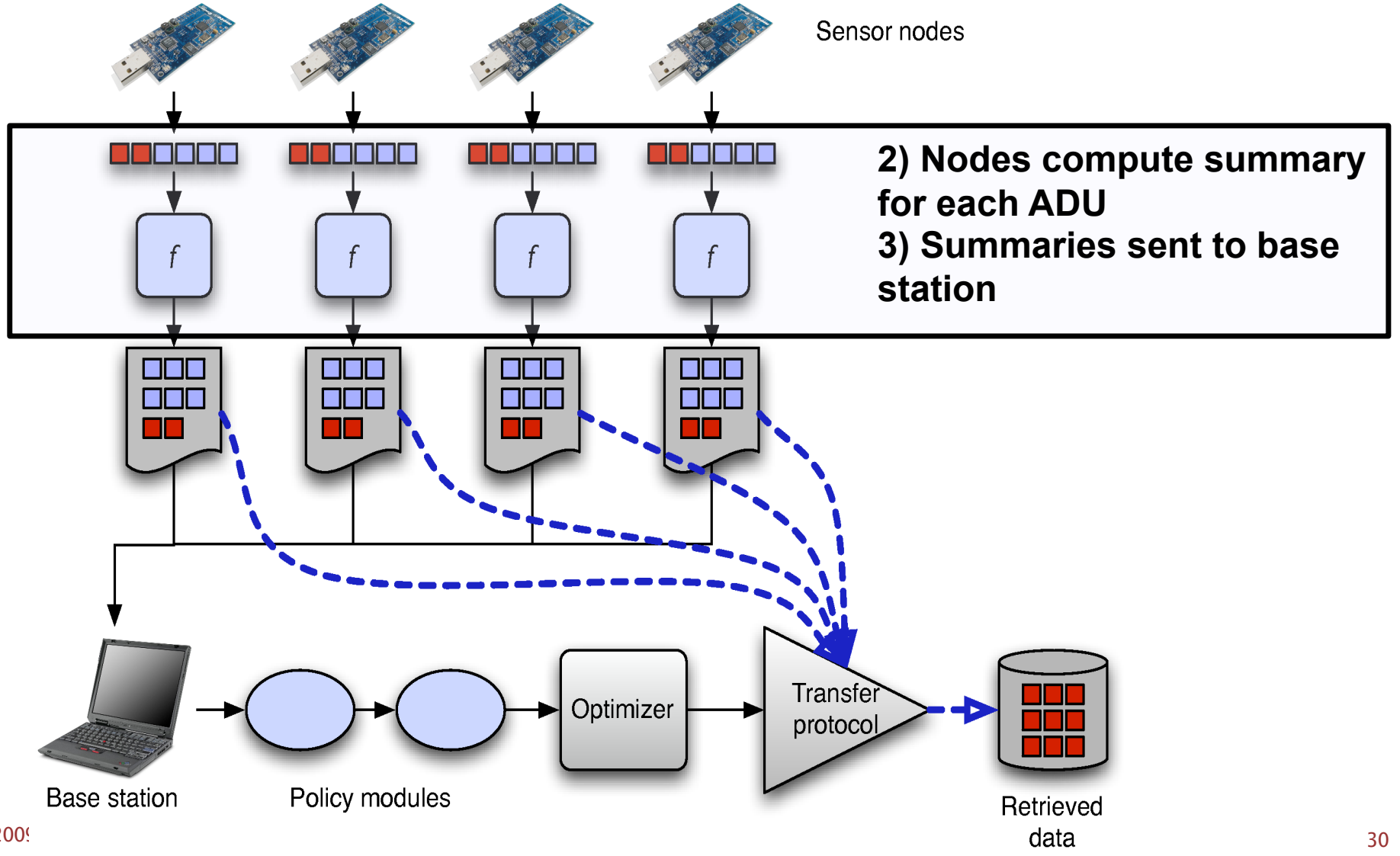
Application Data Unit:
Chunk of data sampled from single node

Example:
60 seconds of **100Hz** seismometer data = 18kB

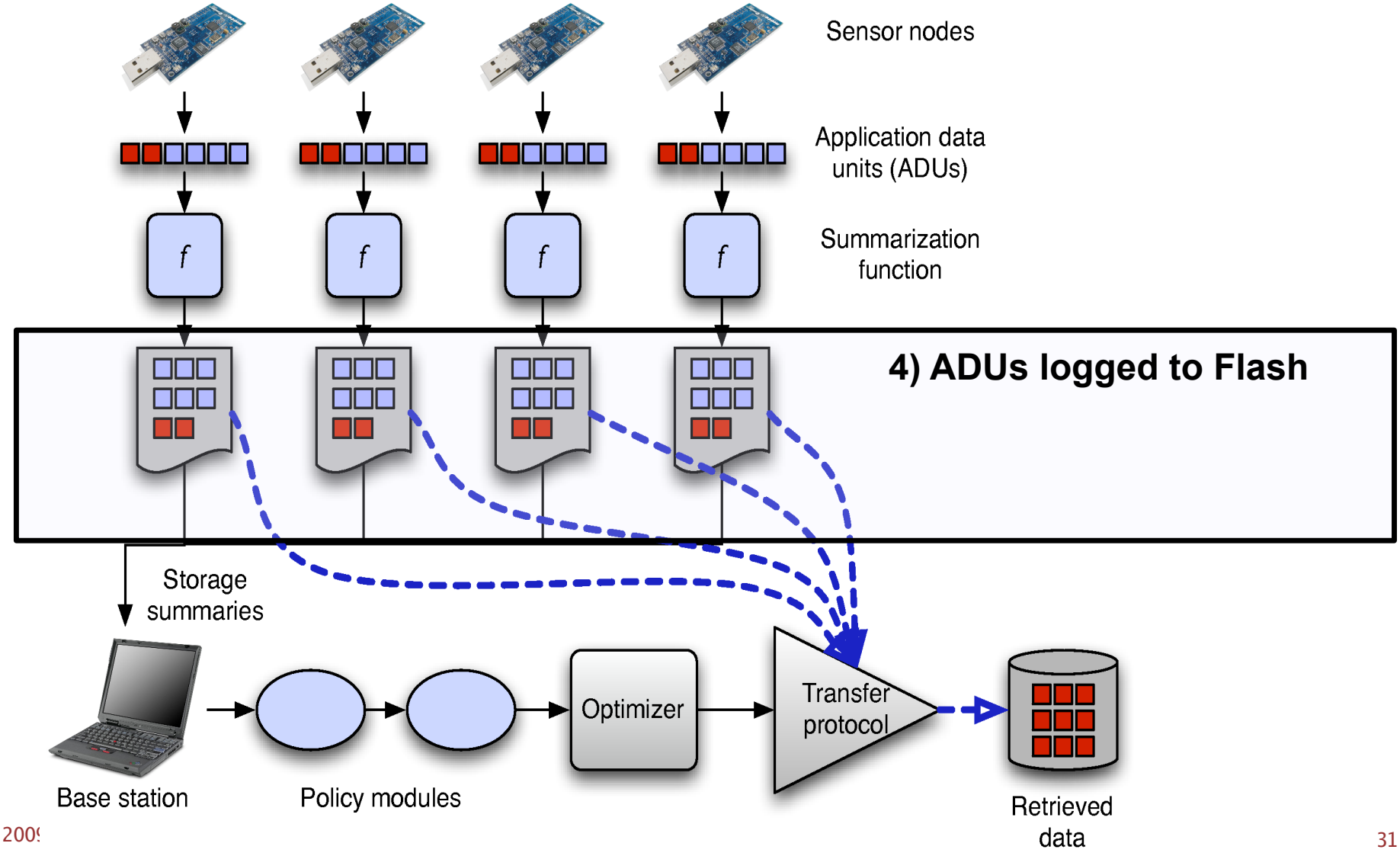
Lance System Architecture



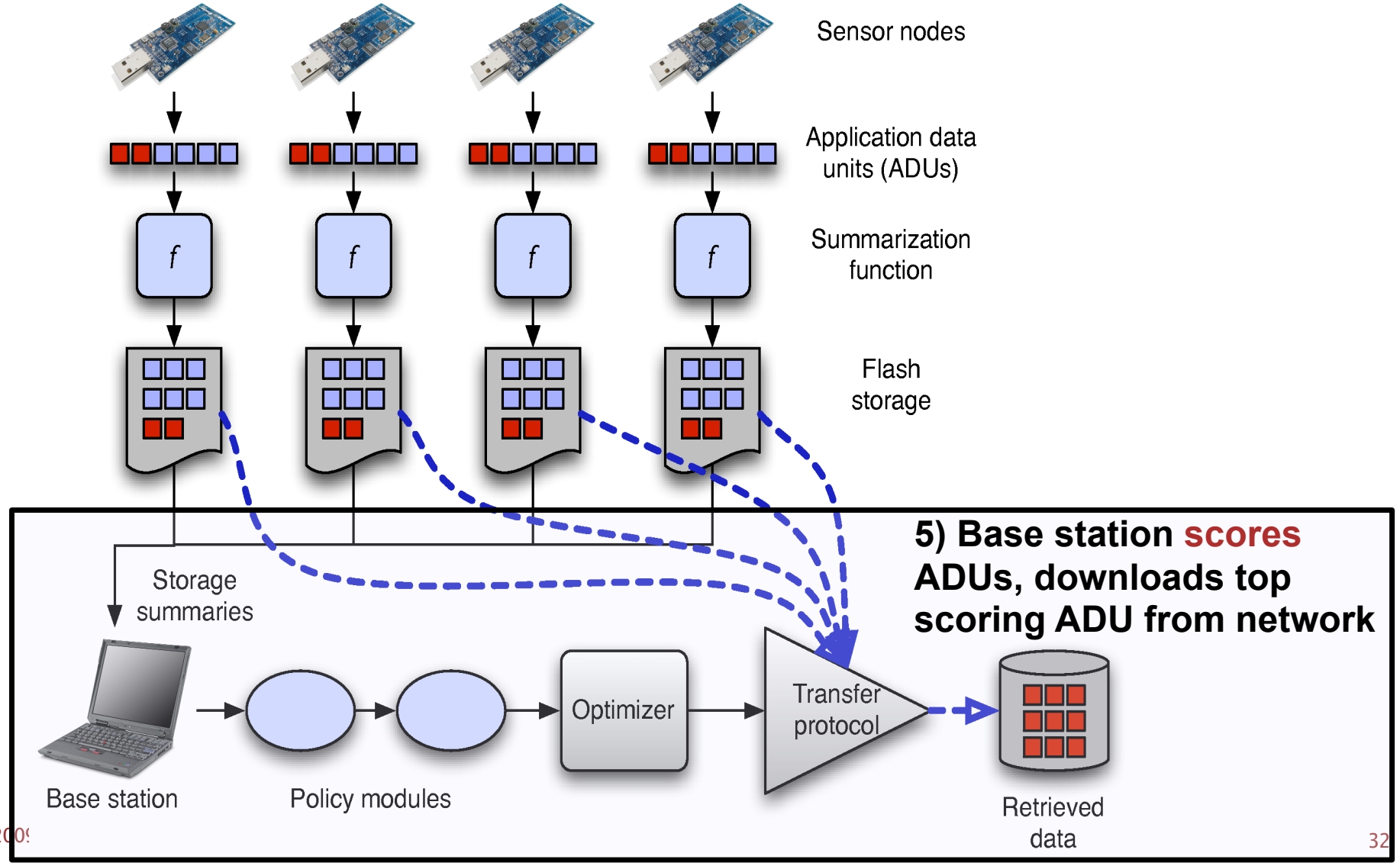
Lance System Architecture



Lance System Architecture



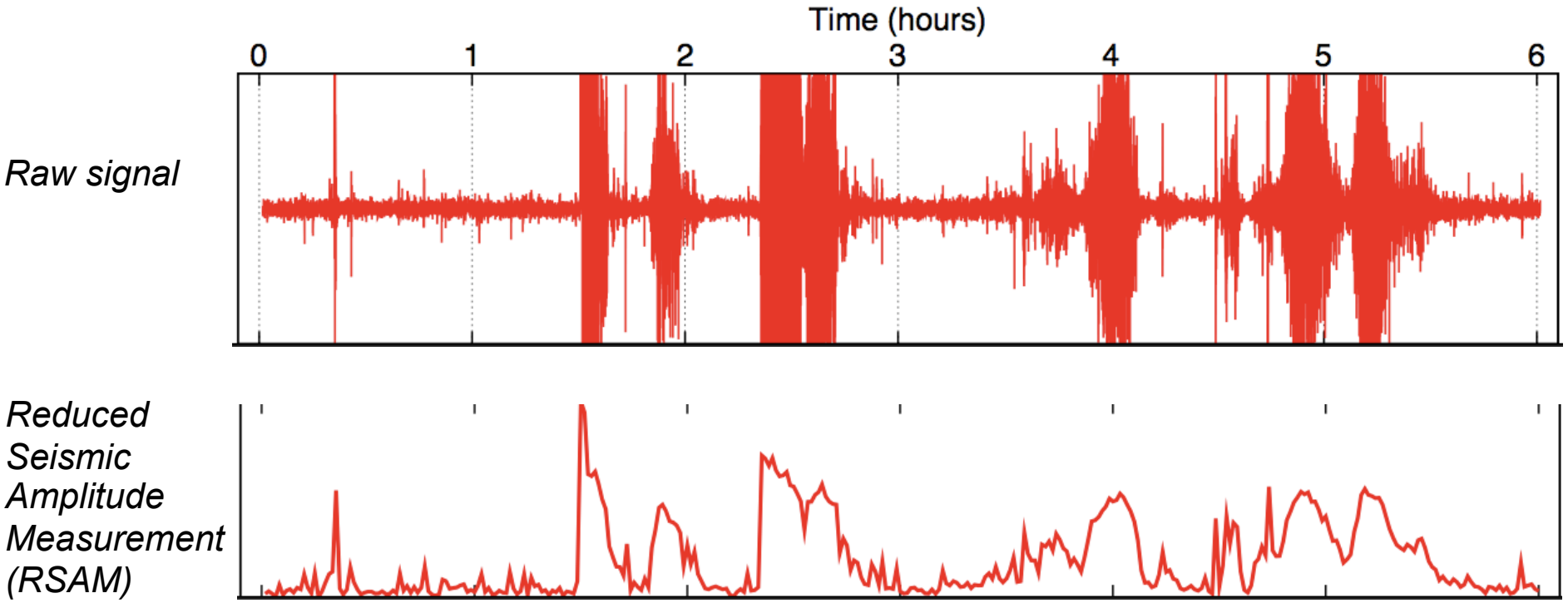
Lance System Architecture



Defining Data Value

Assumption: nodes compute initial value for each ADU.

- This determination is inherently app-specific



Optimization Goal

U = Set of all possible ADUs

E = Vector of energy capacity of each node

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

The optimal set u_o is the set of ADUs maximizing

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

Offline Solution: **Multidimensional Knapsack Problem**

Optimization Strategy

Achieving optimal behavior requires complete knowledge of all data sampled by nodes over time.

- We need an **online** greedy approximation.

Algorithm operates as follows:

1) Assign a **score** to each ADU currently stored by the network
   - We explore three separate scoring functions (next slide)

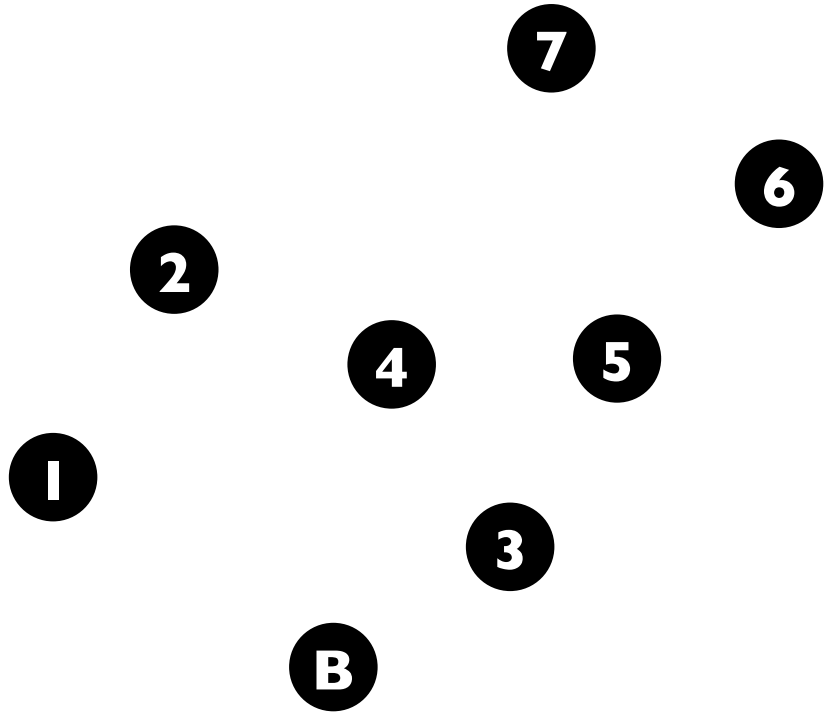
2) Exclude data stored on nodes without enough energy:
   - Given battery capacity C and lifetime target L, require node consume energy at a rate no greater than C/L.
   - Model energy cost for data download, routing, and overhearing on nearby nodes.

3) Download the ADU with the highest score.

Energy Cost for Data Collection

Empirically measure:

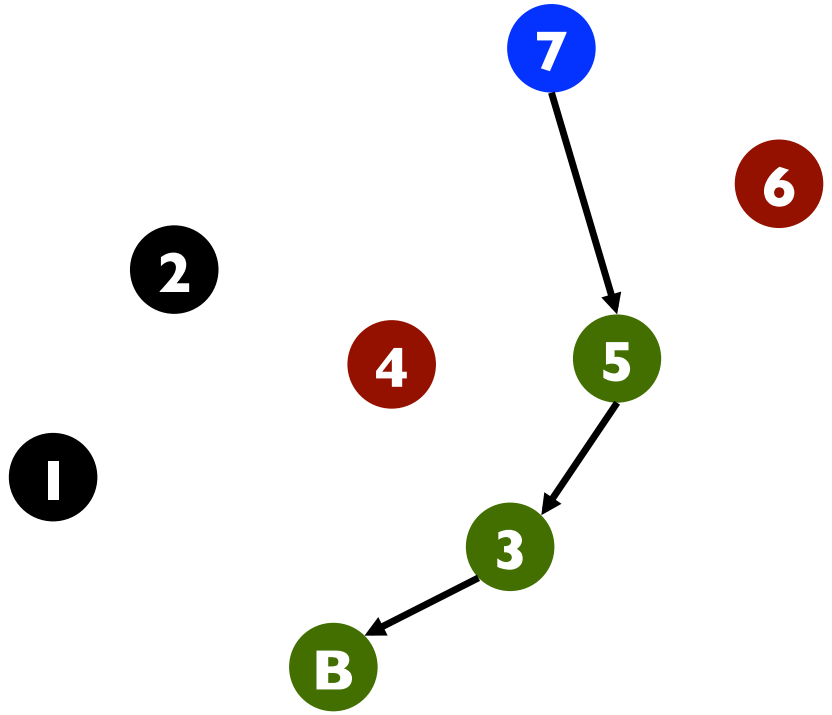
- e_d: the energy drain during **downloading (58 mJ/s)**
- e_r: the energy drain during **routing (55 mJ/s)**
- e_o: the energy drain during **overhearing (6.6 mJ/s)**



Energy Cost for Data Collection

Empirically measure:

e_d: the energy drain during **downloading (58 mJ/s)**
e_r: the energy drain during **routing (55 mJ/s)**
e_o: the energy drain during **overhearing (6.6 mJ/s)**

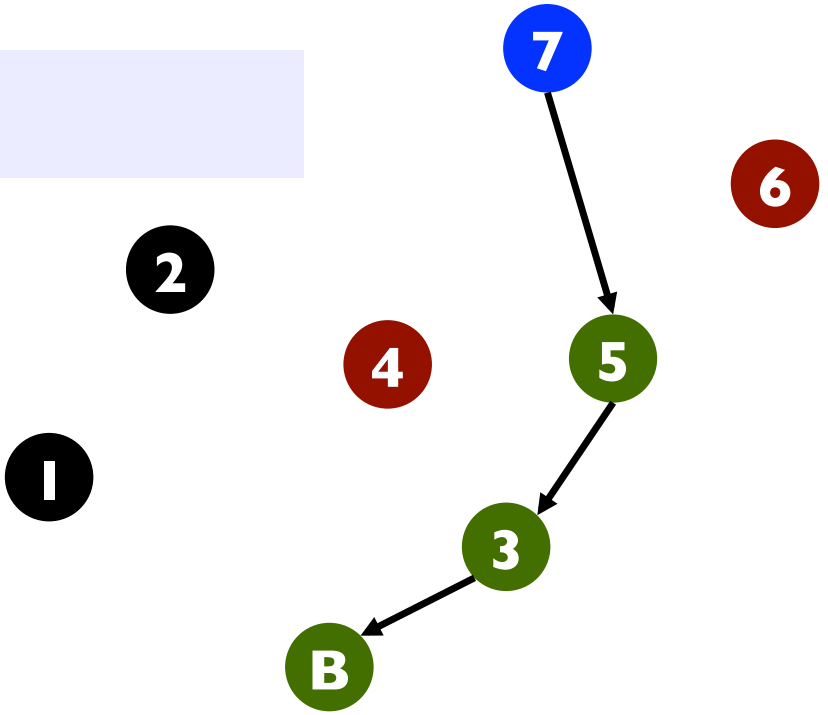


Energy Cost for Data Collection

Empirically measure:

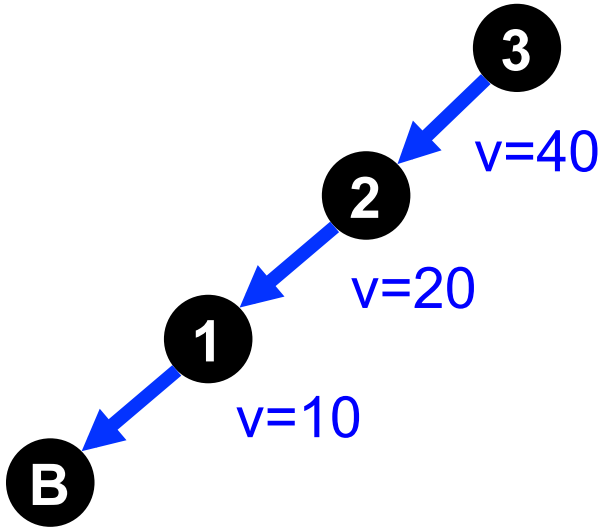
e_d: the energy drain during **downloading (58 mJ/s)**
e_r: the energy drain during **routing (55 mJ/s)**
e_o: the energy drain during **overhearing (6.6 mJ/s)**

$$c_7 = (0, 0, e_r, e_o, e_r, e_o, e_d) * t_7$$



Lance Scoring Functions

Inputs: ADU with value, cost vector
Output: Score used to rank ADUs



ADU_1: [v=10, c = (58, 6.6, 0)]
ADU_2: [v=20, c = (55, 58, 6.6)]
ADU_3: [v=40, c = (58, 58, 55)]

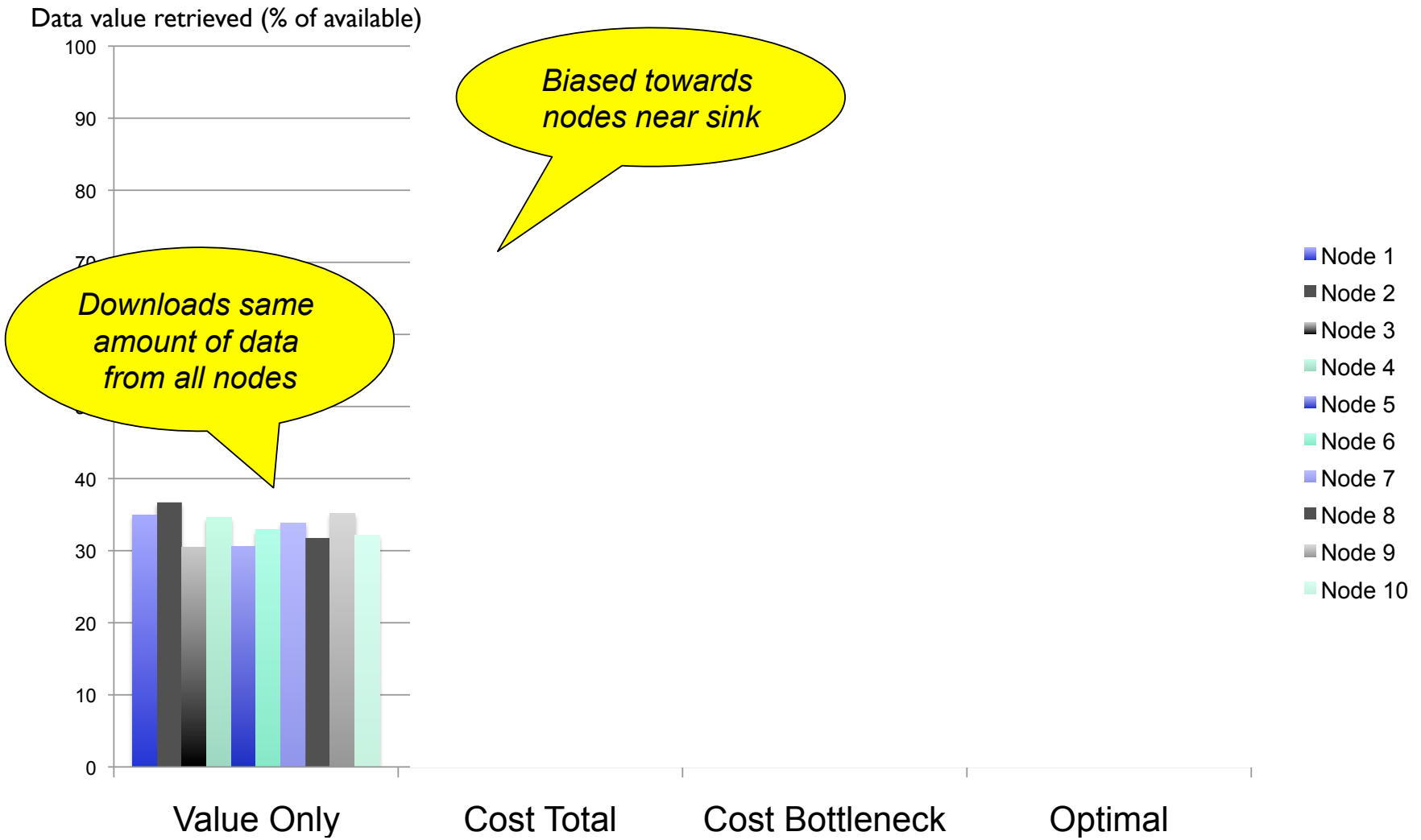
Three scoring functions:

Value only: Score ADU by its value alone

Cost total: Score ADU by value scaled by total cost to download

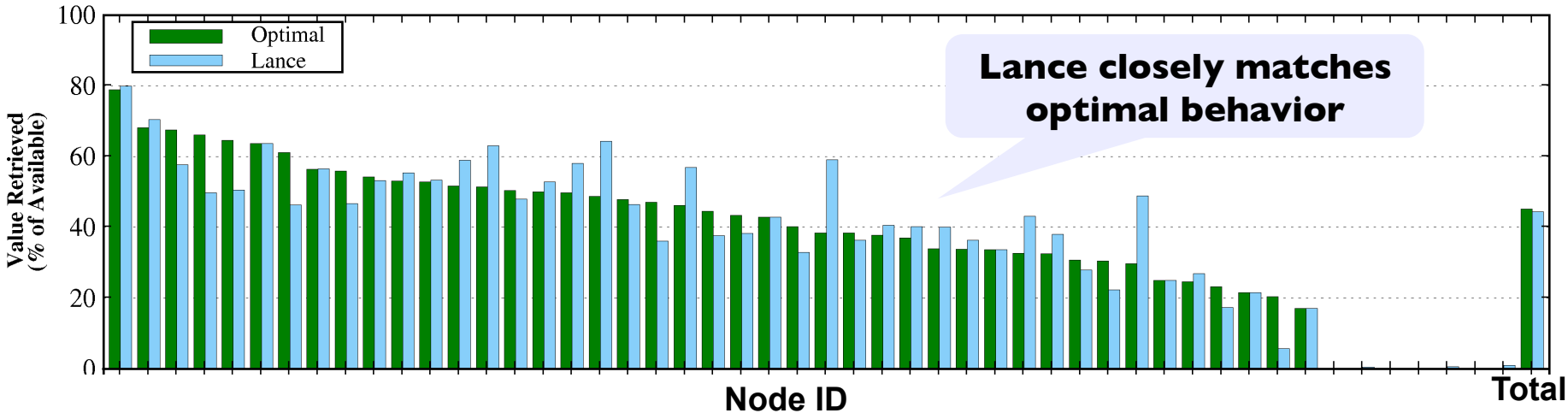
Cost bottleneck: Score ADU by value scaled by most energy-constrained node on routing path

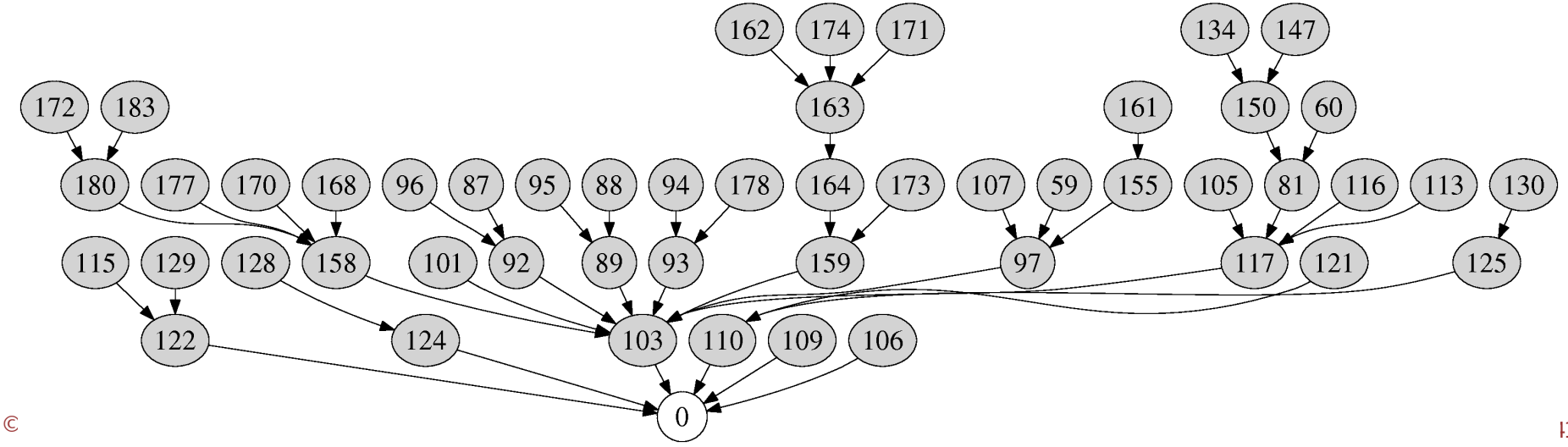
Scoring function comparison
10-node linear chain

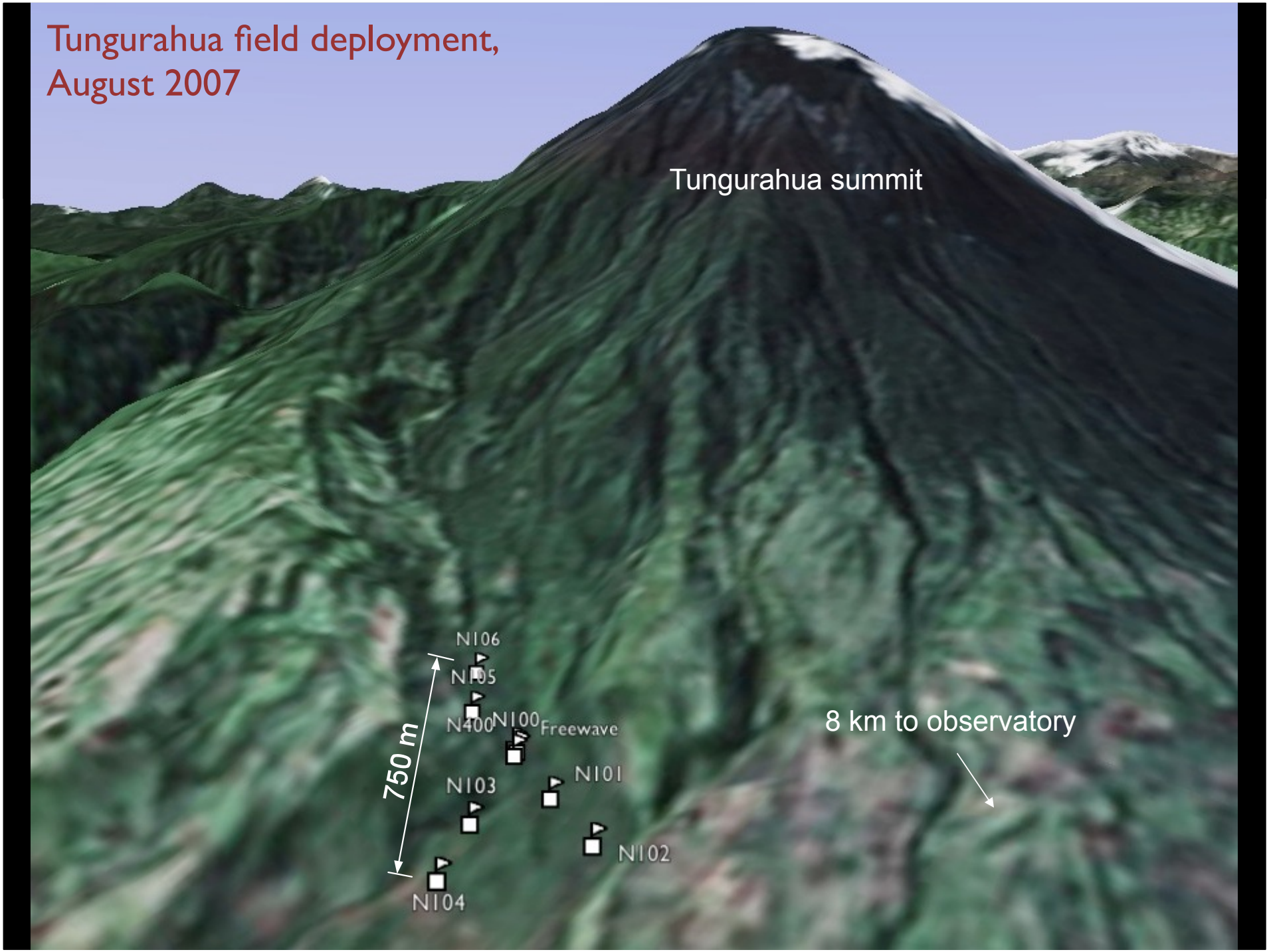


MoteLab Testbed Experiments

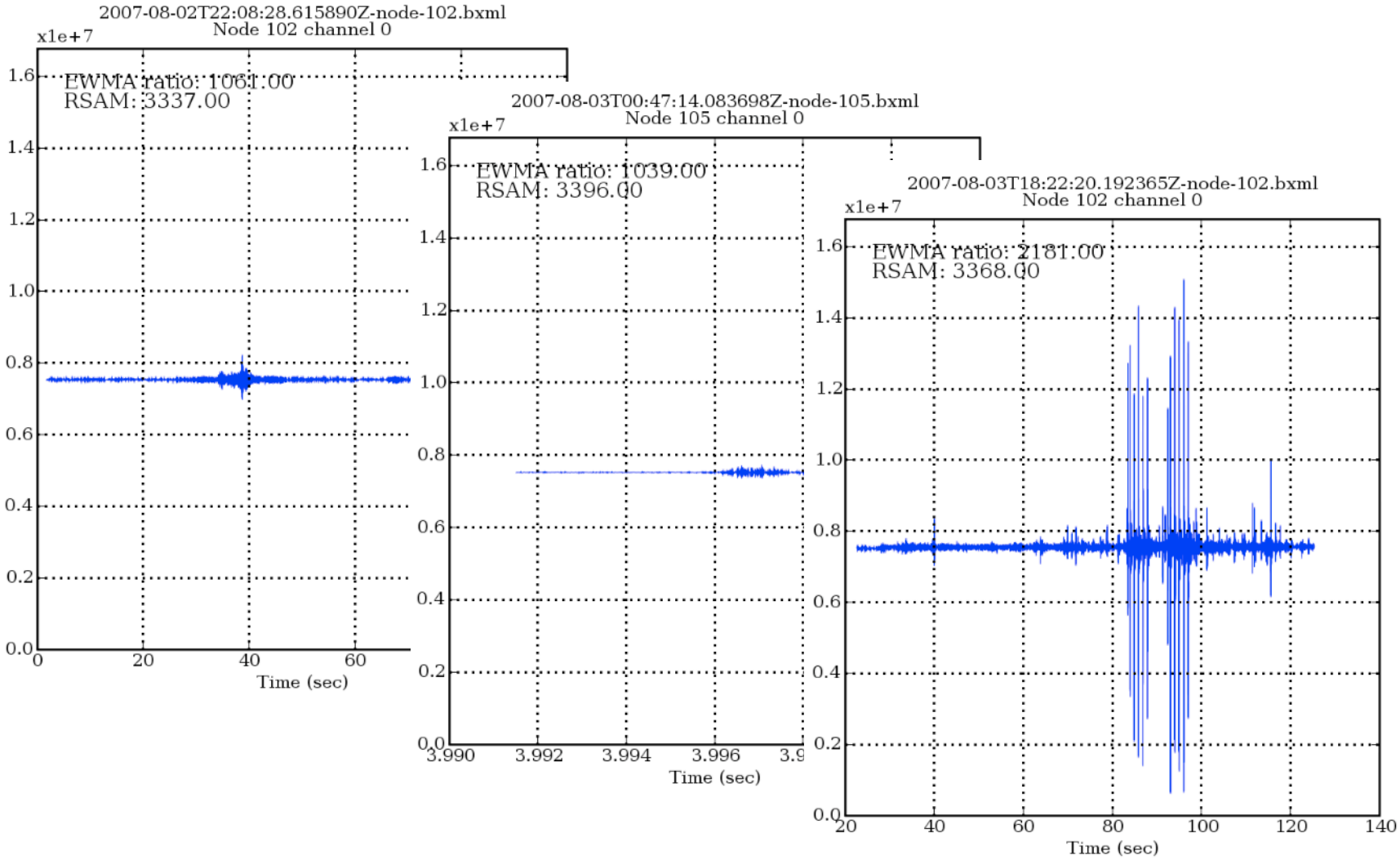
50 node tree topology, Zipf value distribution






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

We fixed the volcano...



Deployment Results

Ran 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 *offline optimal* data

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**

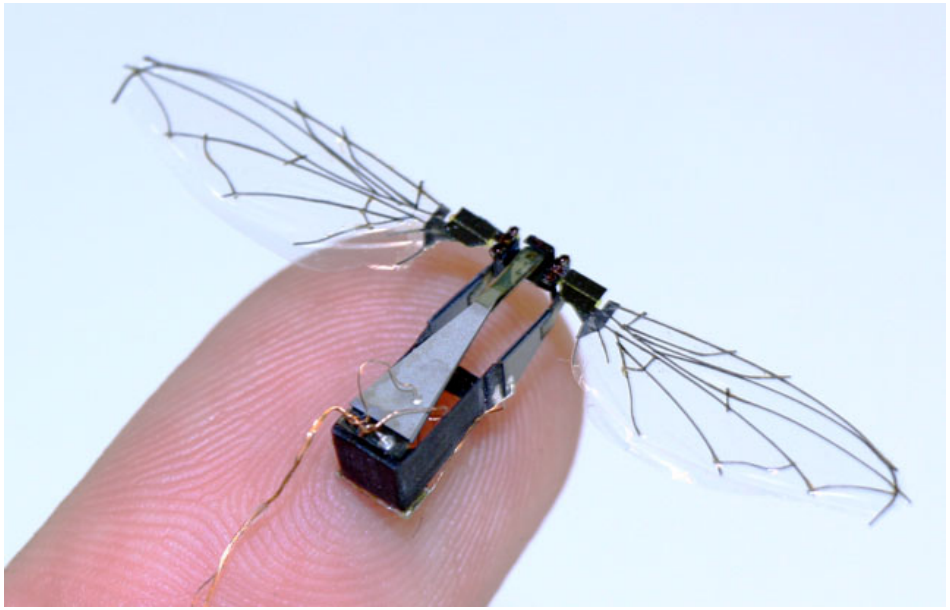
Peloton – A Distributed OS for resource-aware programming

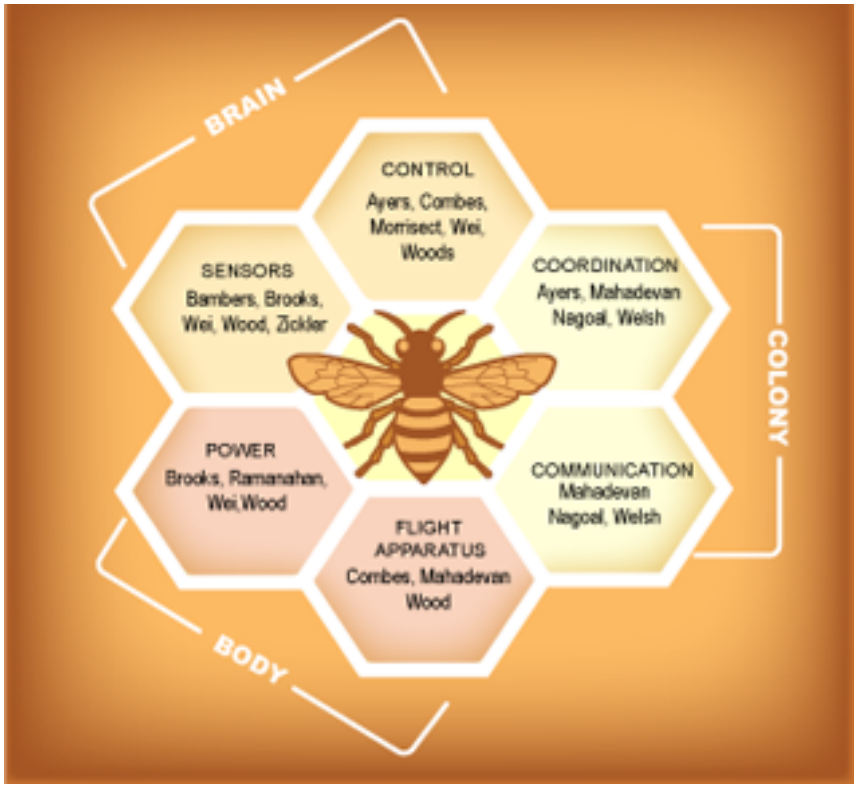
- Decentralizes resource management within the network
- Extend Pixie's abstractions to span allocations on multiple nodes.
- [HotOS 2009]

Future Directions: RoboBees

NSF Expeditions in Computing

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





Conclusions

Sensor networks have tremendous potential for data-intensive science, but resource management remains a significant challenge.

- Too many low-level knobs that impact performance and lifetime.

We advocate **resource-aware programming** as a fundamental design principle.

- Applications cannot hide from the need to manage and adapt.
- **Pixie** provides OS abstractions for resource awareness at the node level.
- **Lance** offers a framework for network-wide resource management.
- These systems provide a great deal of flexibility and control to programmers, while hiding the complexity of low-level resource management.

For more information, papers, and code:

`http://www.eecs.harvard.edu/~mdw`

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

Resource Estimation

Storage and memory are straightforward

- Allocators track total flash/memory consumption

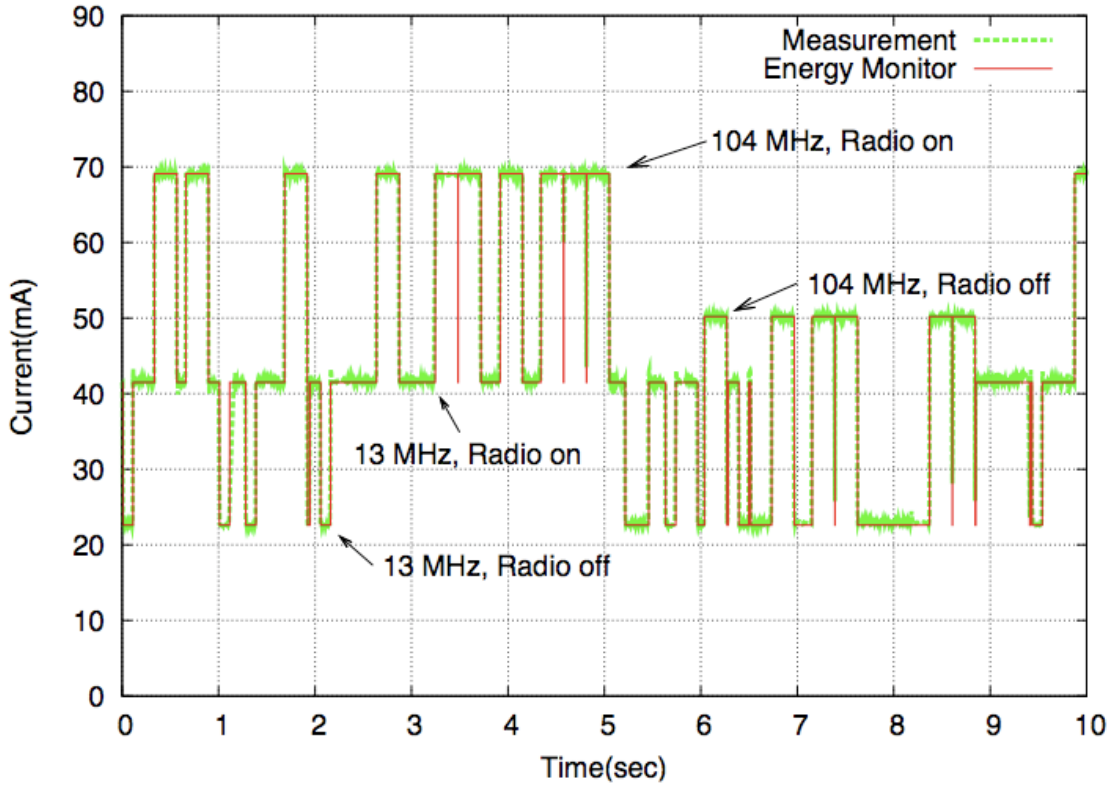
Bandwidth: Radio link estimates packet transmission delay

- 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

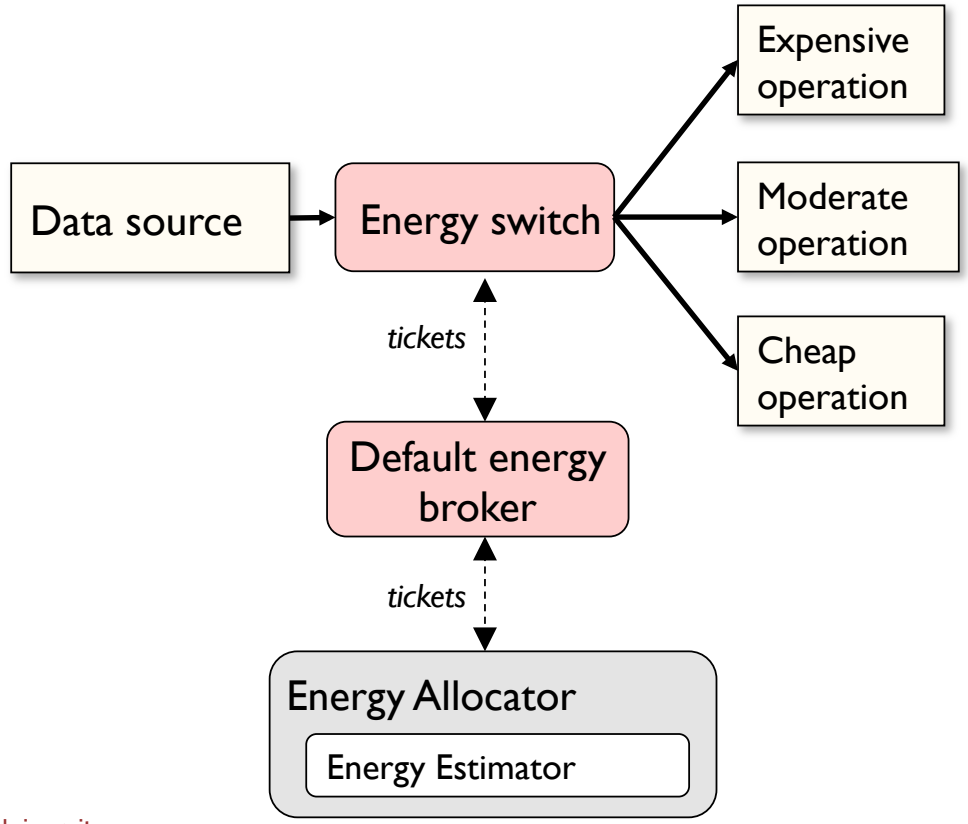
- Tracks state of each hardware device (CPU, radio, flash, sensors) in real time
- Accurate empirical model of hardware energy consumption (about 2-3% error)
- Low overhead, no hardware support required.



Other Energy Brokers

Energy-aware switch

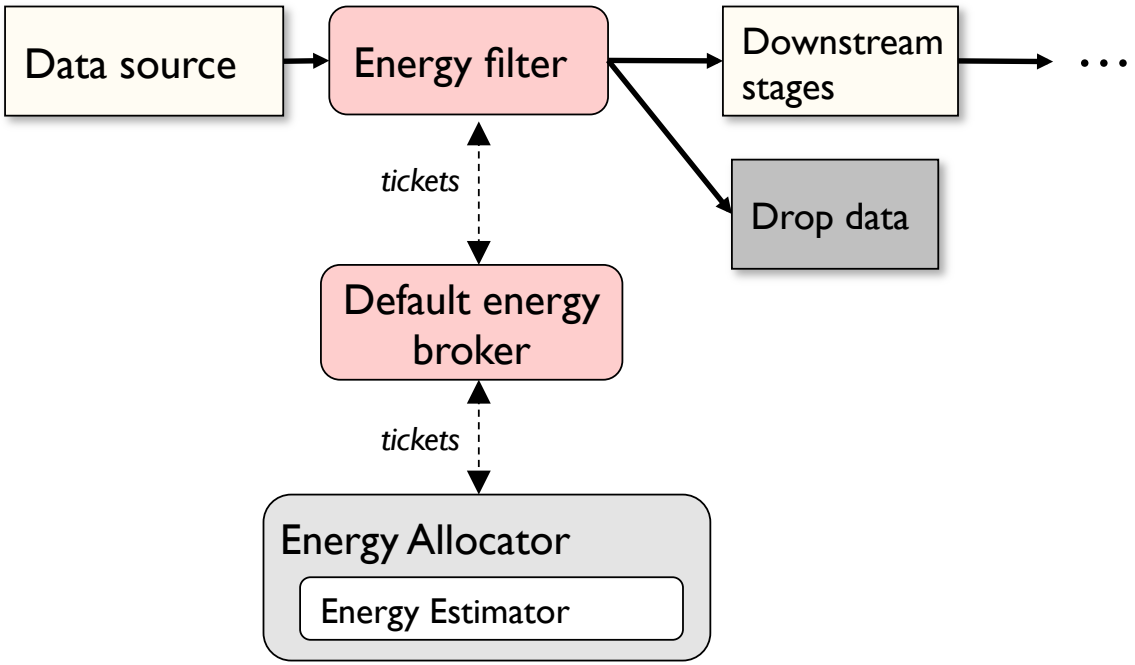
- Sends data down one or more downstream paths based on energy availability
- Mimics policy used by Eon *[Sorber et al., Sensys 2007]*



Other Energy Brokers

Energy-aware filter

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



Bandwidth Adaptation in the Motion Analysis Application

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

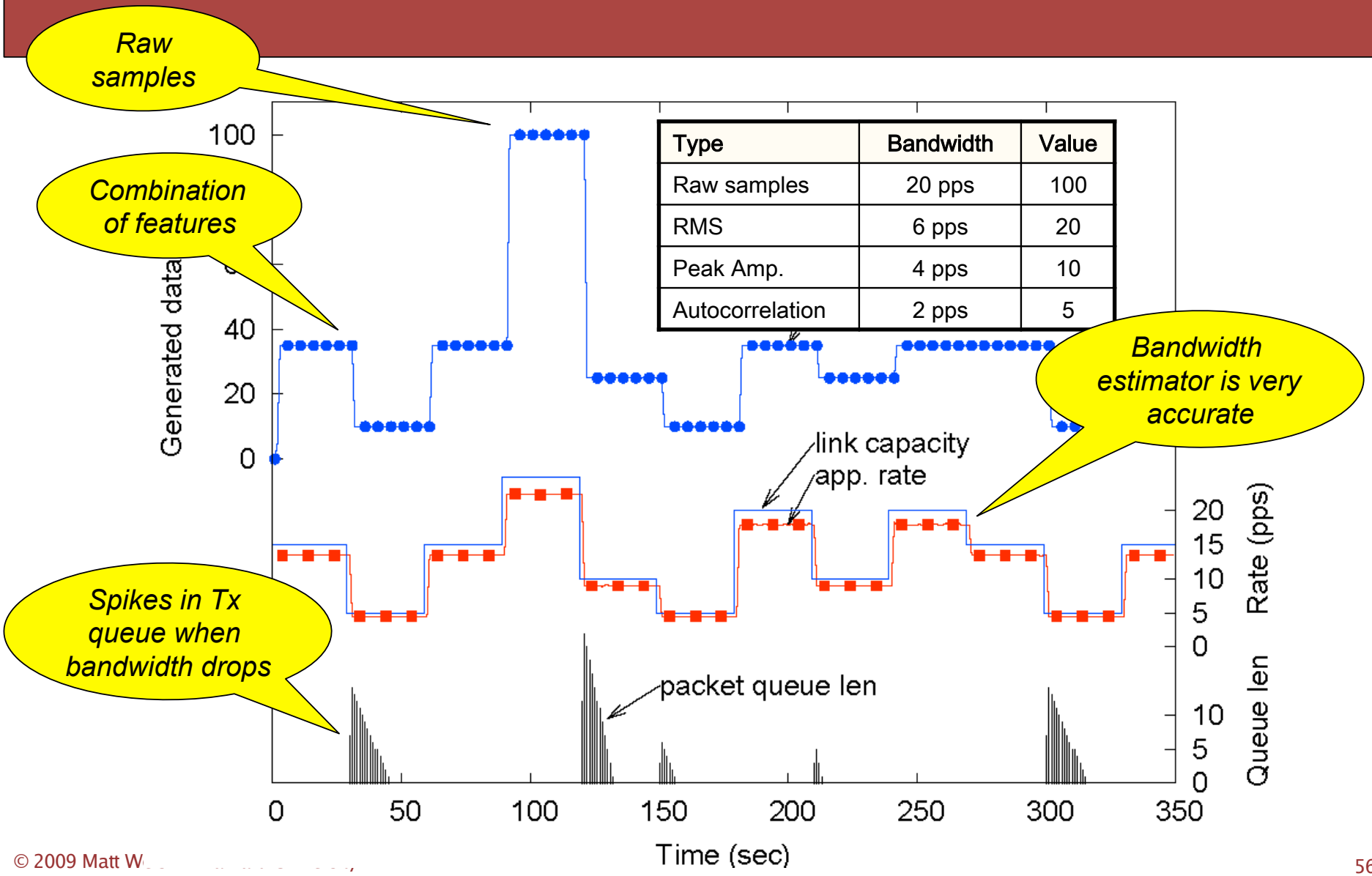
- Don't want to transmit data that won't get to the base station.

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

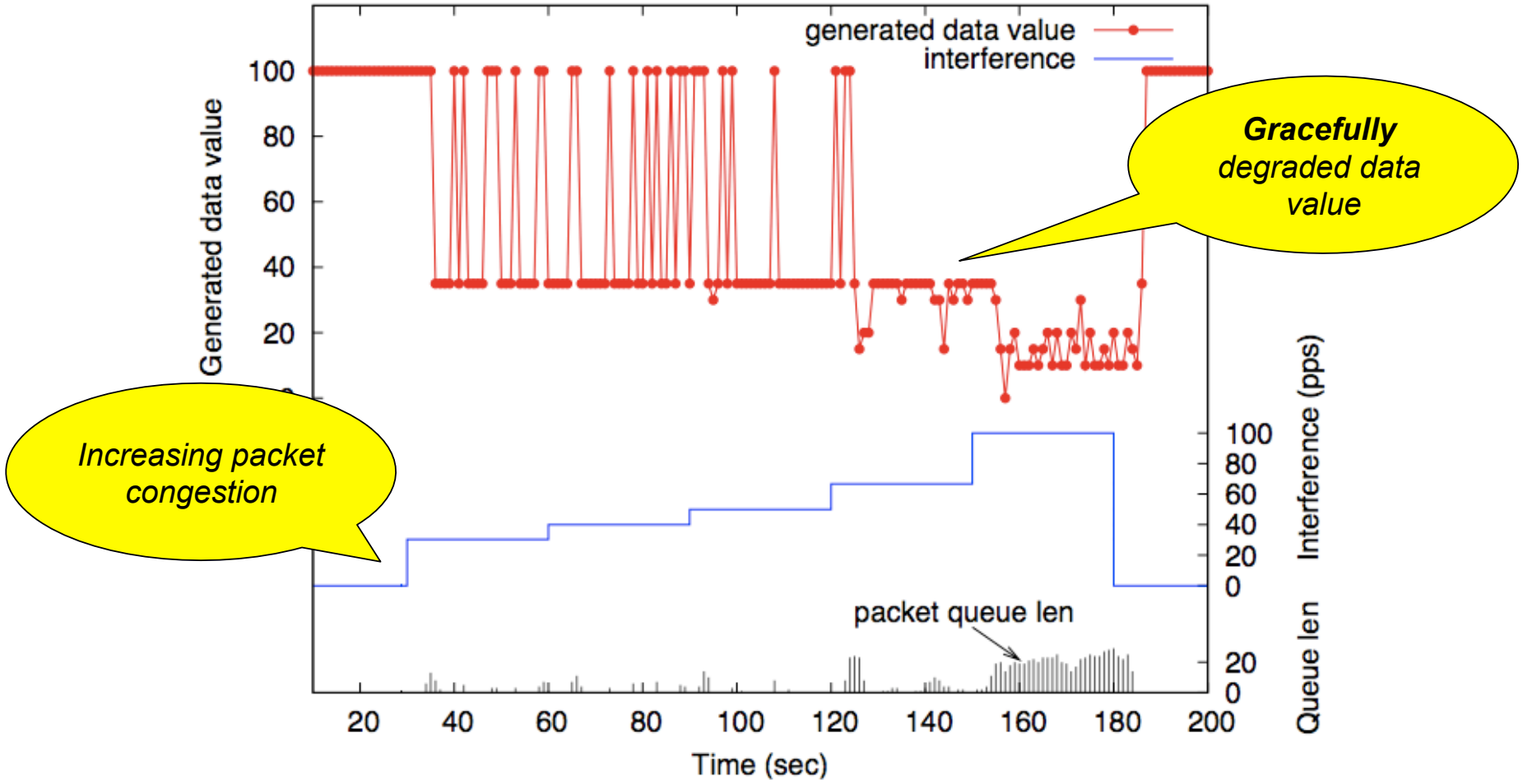
Type	Bandwidth	Value
Raw samples	20 pps	100
RMS	6 pps	20
Peak Amp.	4 pps	10
Autocorrelation	2 pps	5

Experiment: Bandwidth Adaptation



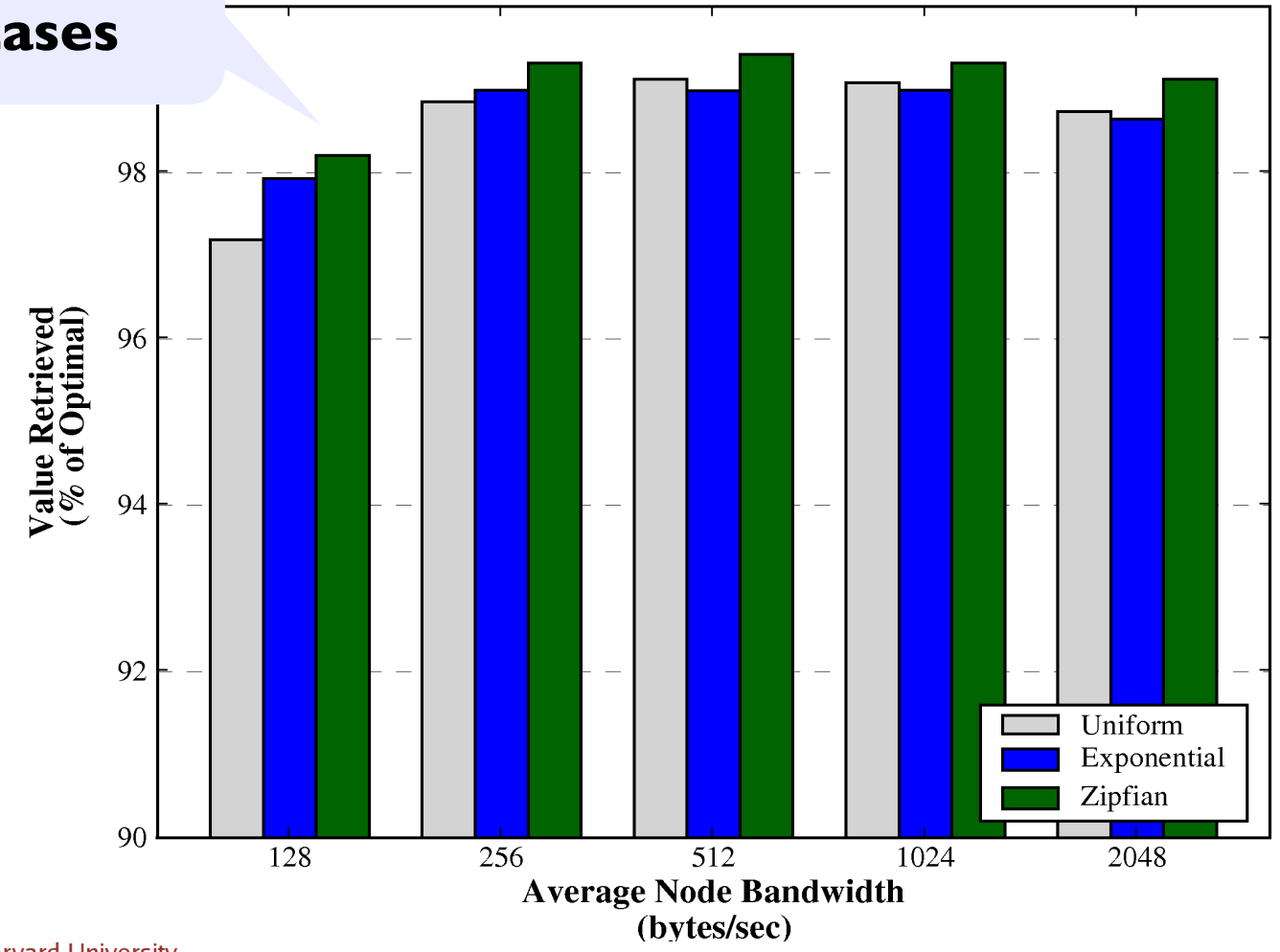
Type	Bandwidth	Value
Raw samples	20 pps	100
RMS	6 pps	20
Peak Amp.	4 pps	10
Autocorrelation	2 pps	5

Adaptation to radio channel interference



Varying Data Value Distribution

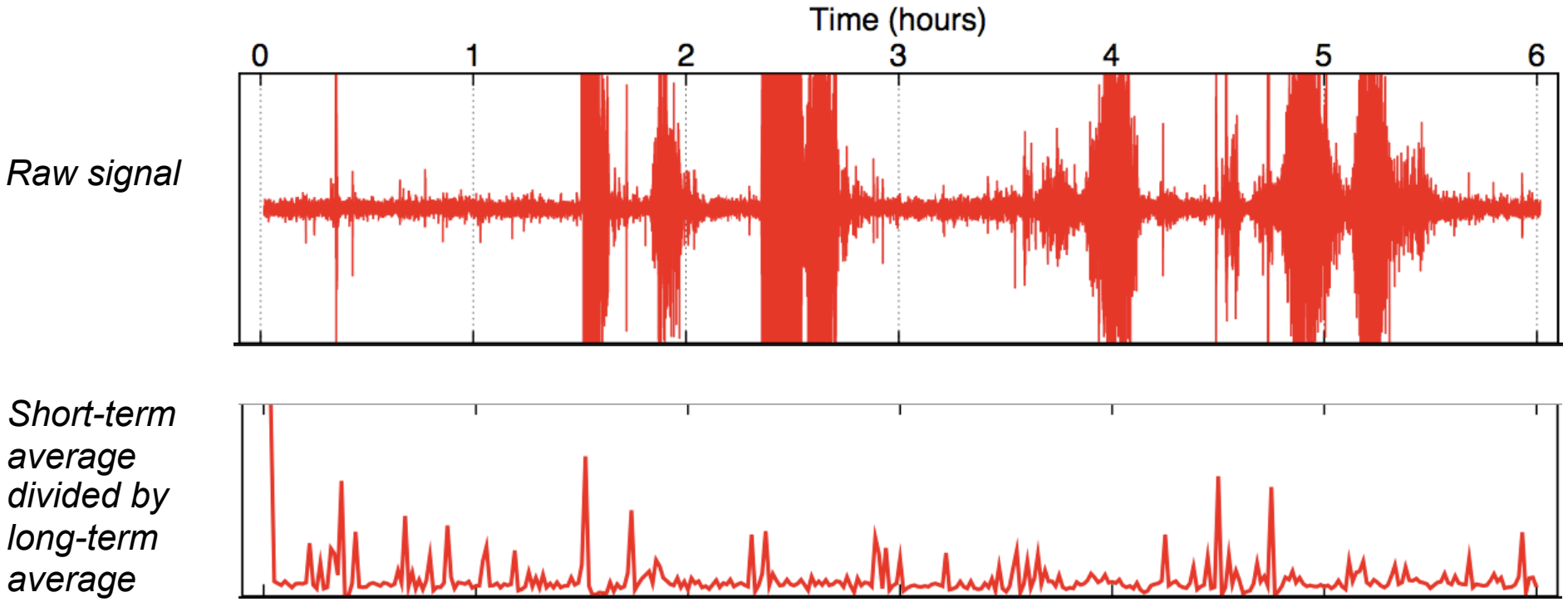
Above 96% in all cases



Defining Data Value

Assumption: nodes compute initial value for each ADU.

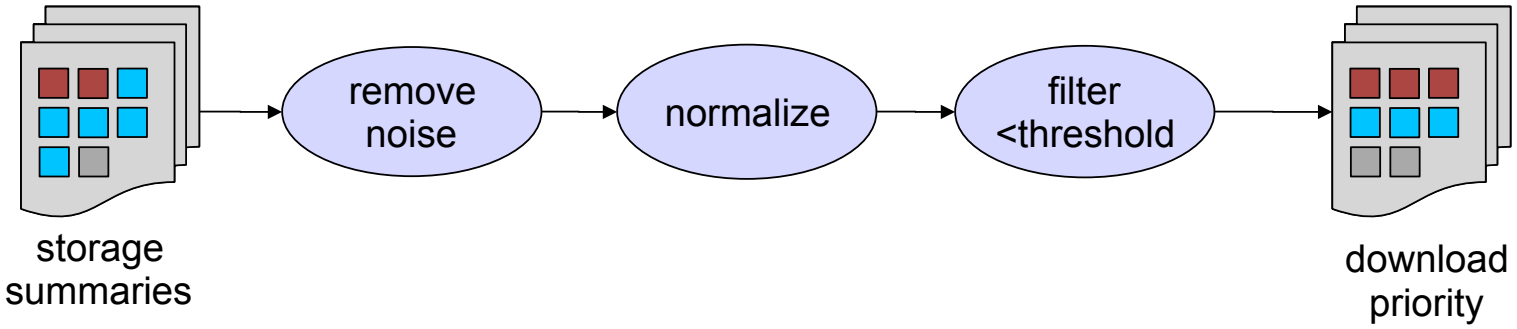
- This determination is inherently app-specific



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

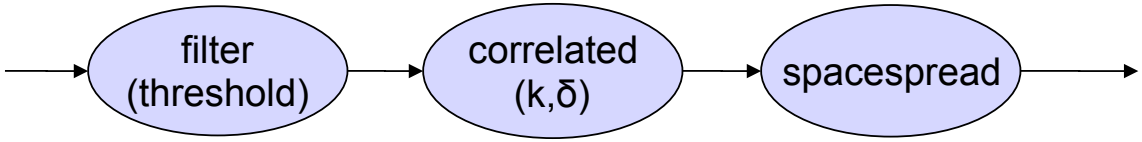
- **costweight**: scale ADU value by cost to download

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

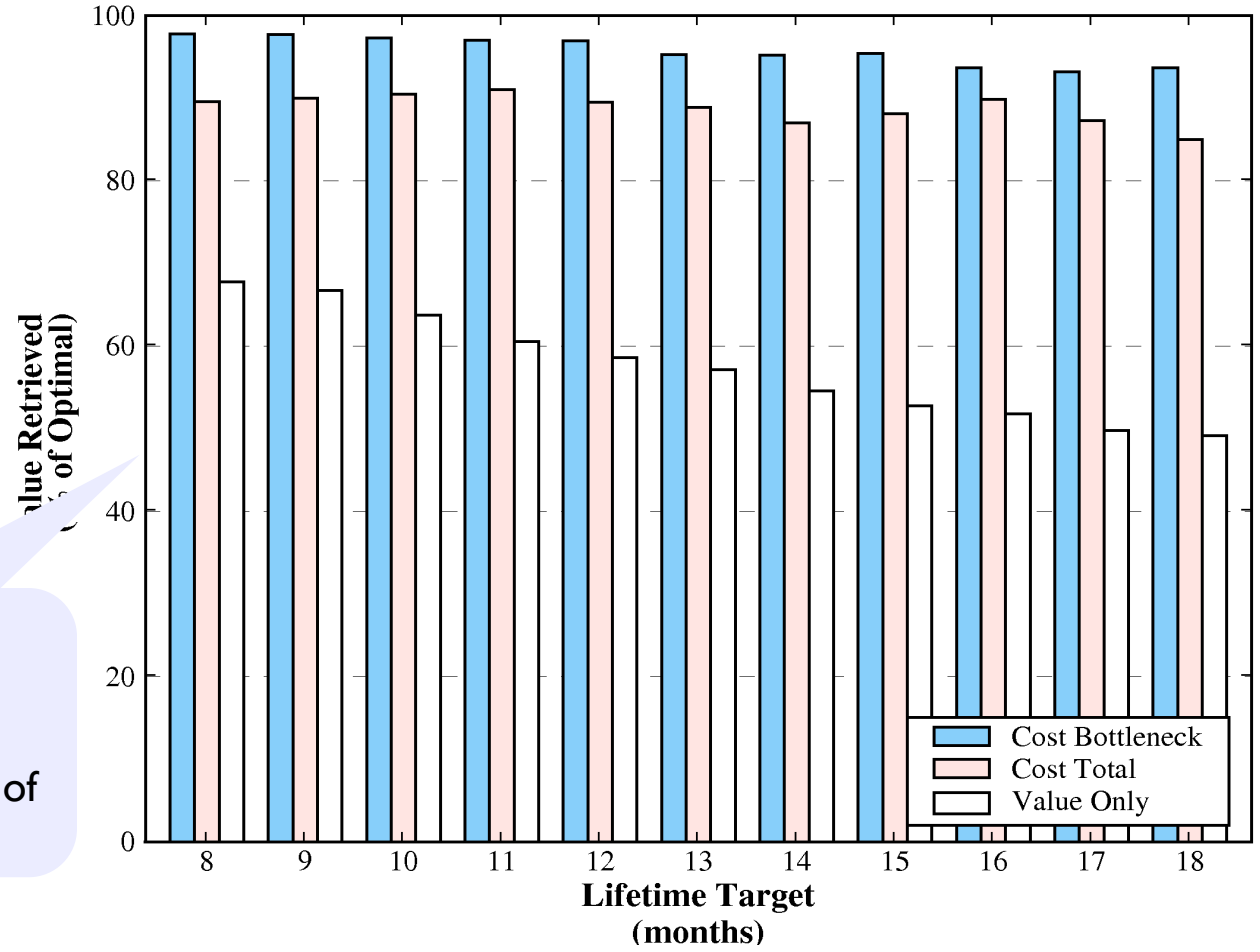
Network-wide earthquake detection in Lance:



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

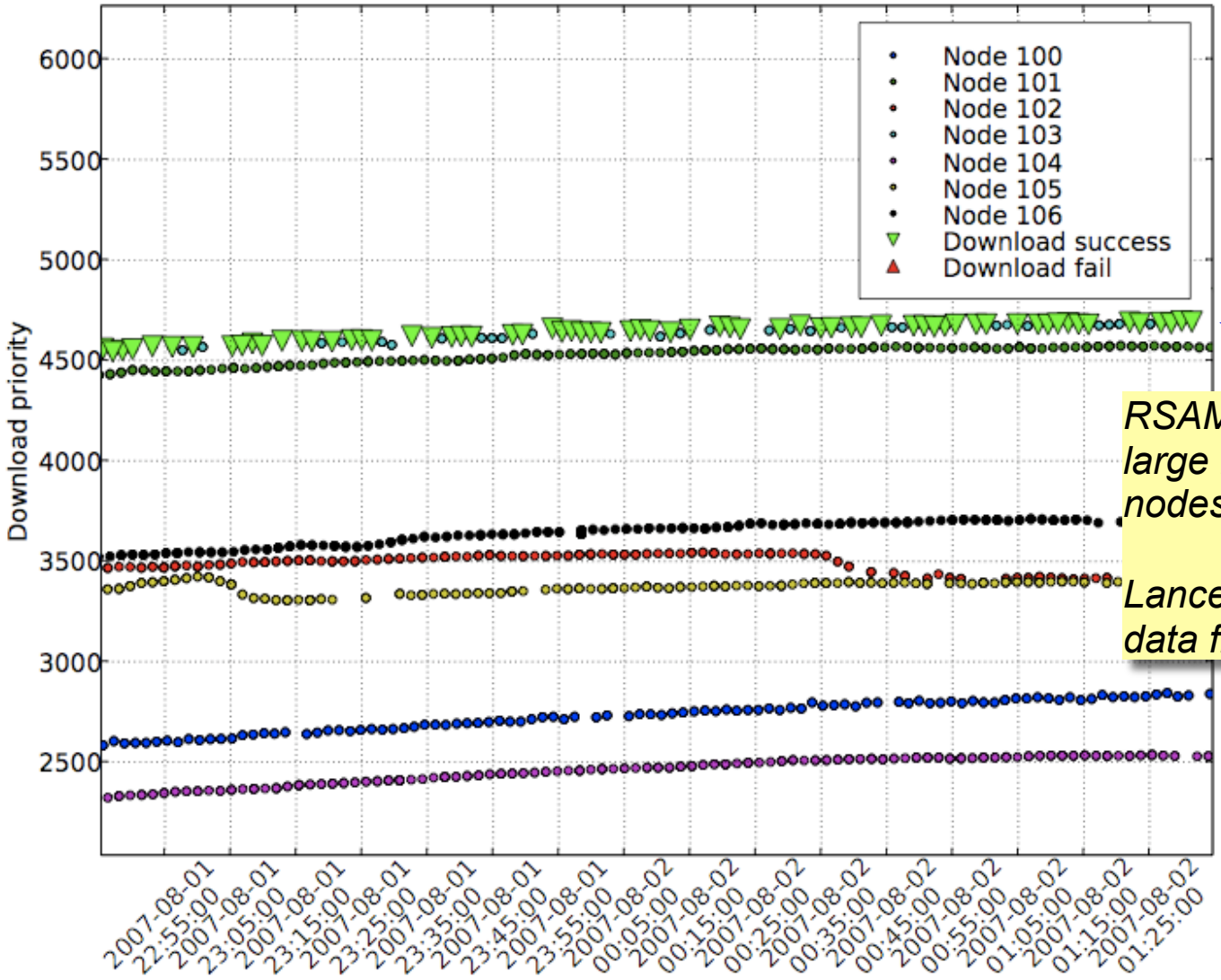
Results under varying lifetime target

25-node tree, Zipf value distribution



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

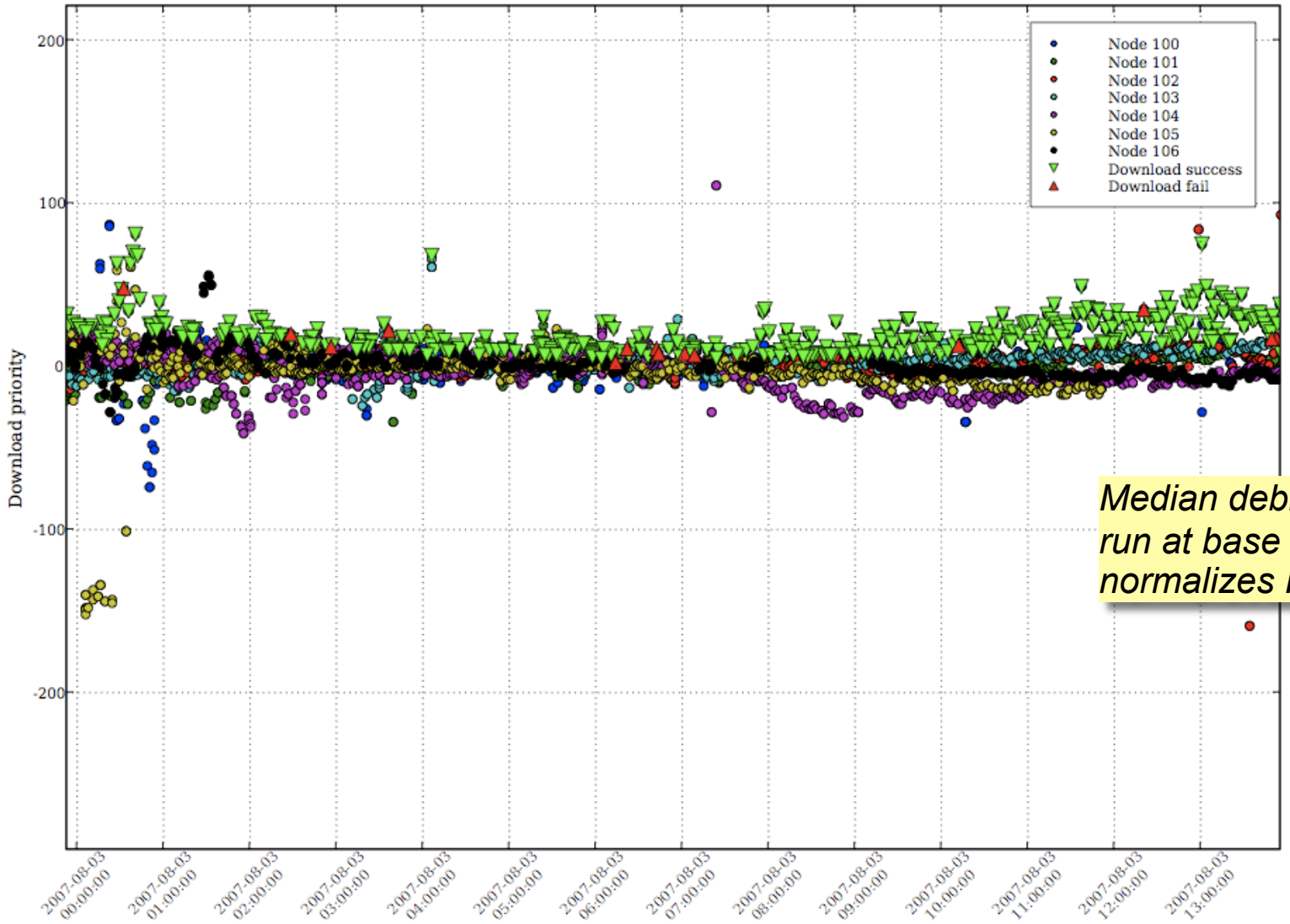
RSAM prioritization: DC bias



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

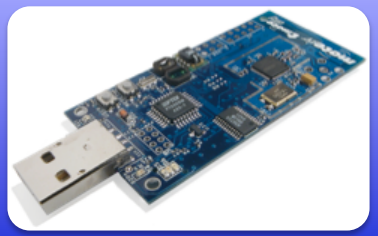
Brokers decouple resource management policies from mechanisms

- Enables composable resource adaptations

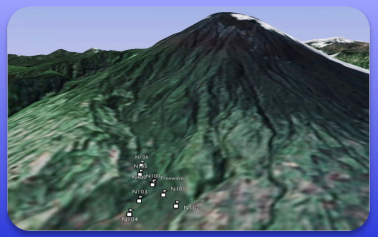
Resource-awareness is pervasive in the programming model

- 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

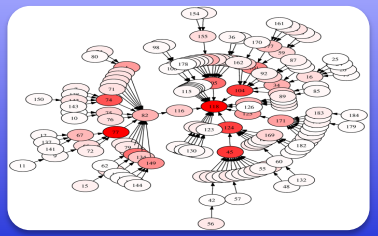
Key Research Questions



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

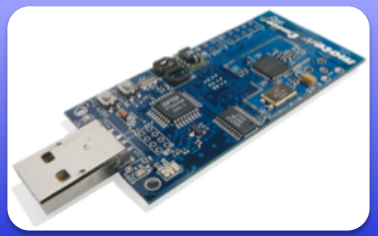


How do we ensure high data fidelity for real-world applications?

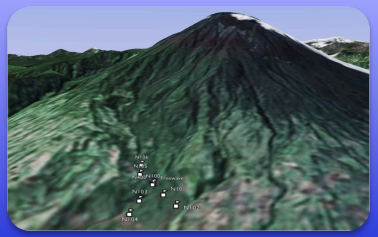


How do we program complex network behavior at a high level?

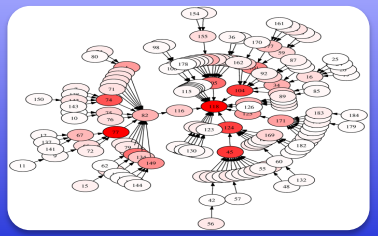
Our Approach



New OS designs to support resource-aware programming:
Pixie and **Lance**

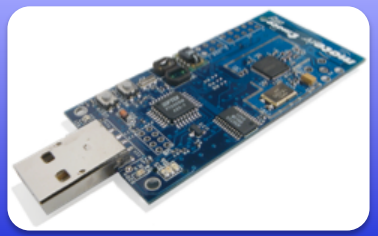


Extensive field research with domain scientists: volcanoes, emergency medical care, neuromotor disease rehab, and city-wide environmental monitoring.

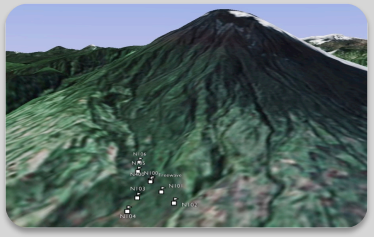


New programming languages and models:
NesC, **Regiment**, **Flask**, **SORA**, **Abstract Regions**

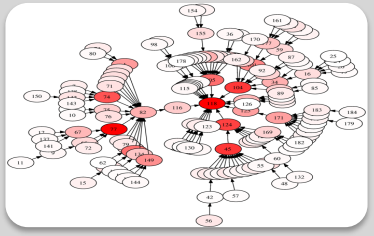
Today's Talk



New OS designs to support resource-aware programming:
Pixie and **Lance**



Extensive field research with domain scientists: volcanoes, emergency medical care, neuromotor disease rehab, and city-wide environmental monitoring.



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