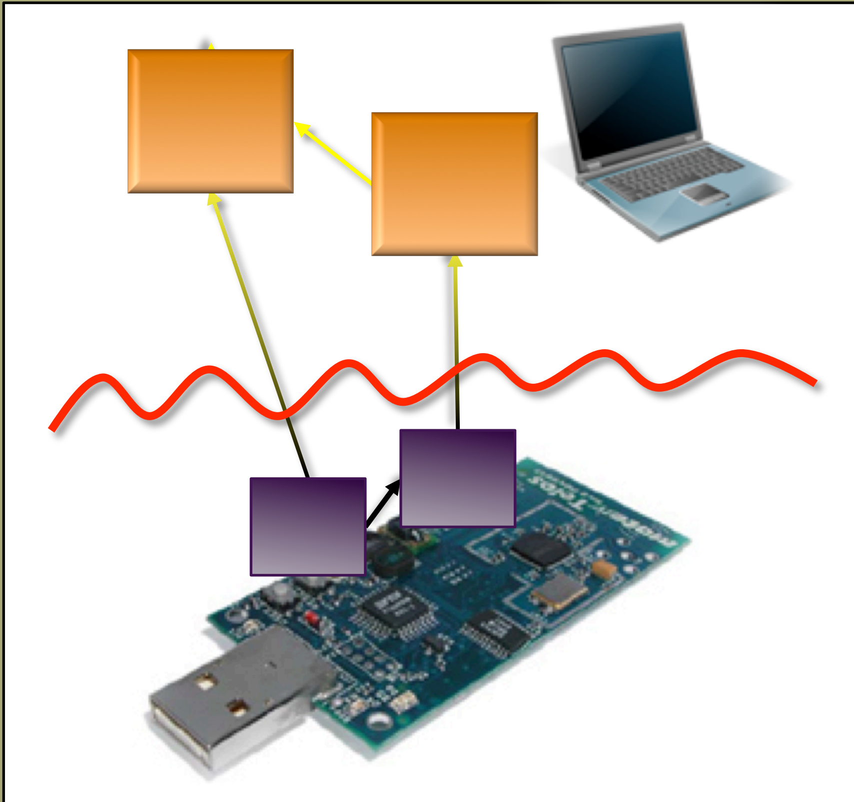


Wishbone: Profile-based Partitioning for Sensornet Applications

- **Ryan Newton**, Sivan Toledo, Lewis Girod, Hari Balakrishnan, Samuel Madden

CSAIL

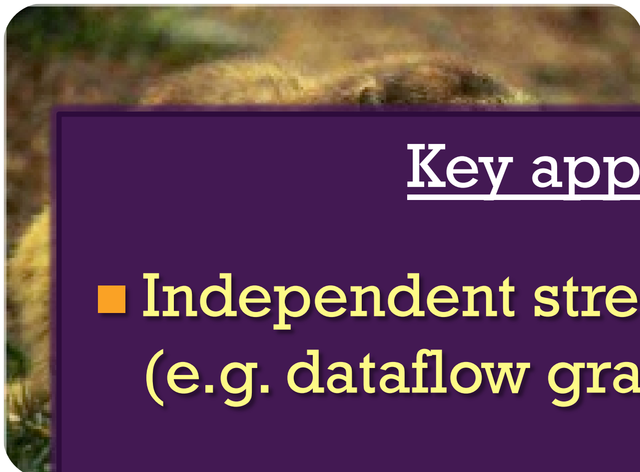
+ Example Application: Locating Marmots

- Gothic, CO deployment August 2007
- Voxnet Platform
  - 2x PXA255, 64MB RAM, 8GB Flash, 802.11B, Mica2 supervisor, Li+ battery, Charge controller
  - Sensors: 4x48KHz audio, 3-axis accel, GPS, Internal temp

with Lewis Girod & UCLA Blumstein Lab

+ We target sensing applications

Animal localization



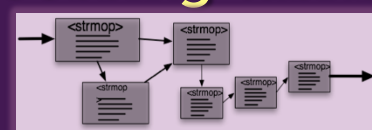
Pothole detection

Compu
Vis

Pipeline
leak detection

Key application features

- Independent stream processing tasks (e.g. dataflow graphs)



- Predictable data rates

Challenges

- Heterogeneous platforms
- CPU & radio bottlenecks

Speaker identification

EEG Seizure detection

+ Heterogeneous Platforms

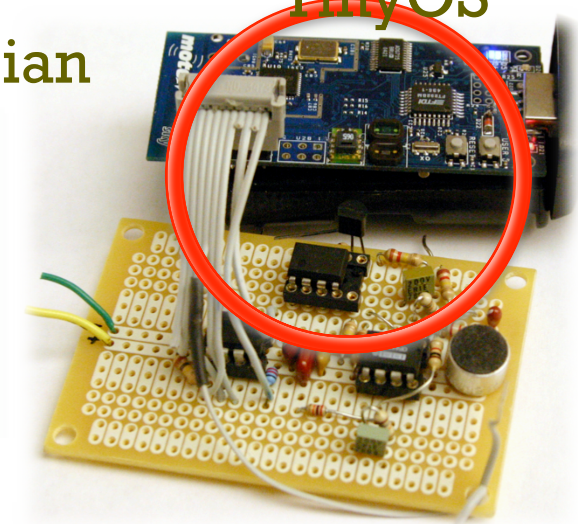
Router
weak cpu,
strong radio

Smartphones
medium cpu,
strong radio

Brew
JavaME
Symbian
Android
iPhone SDK

Low power sensors
weak cpu/radio

TinyOS
Contiki

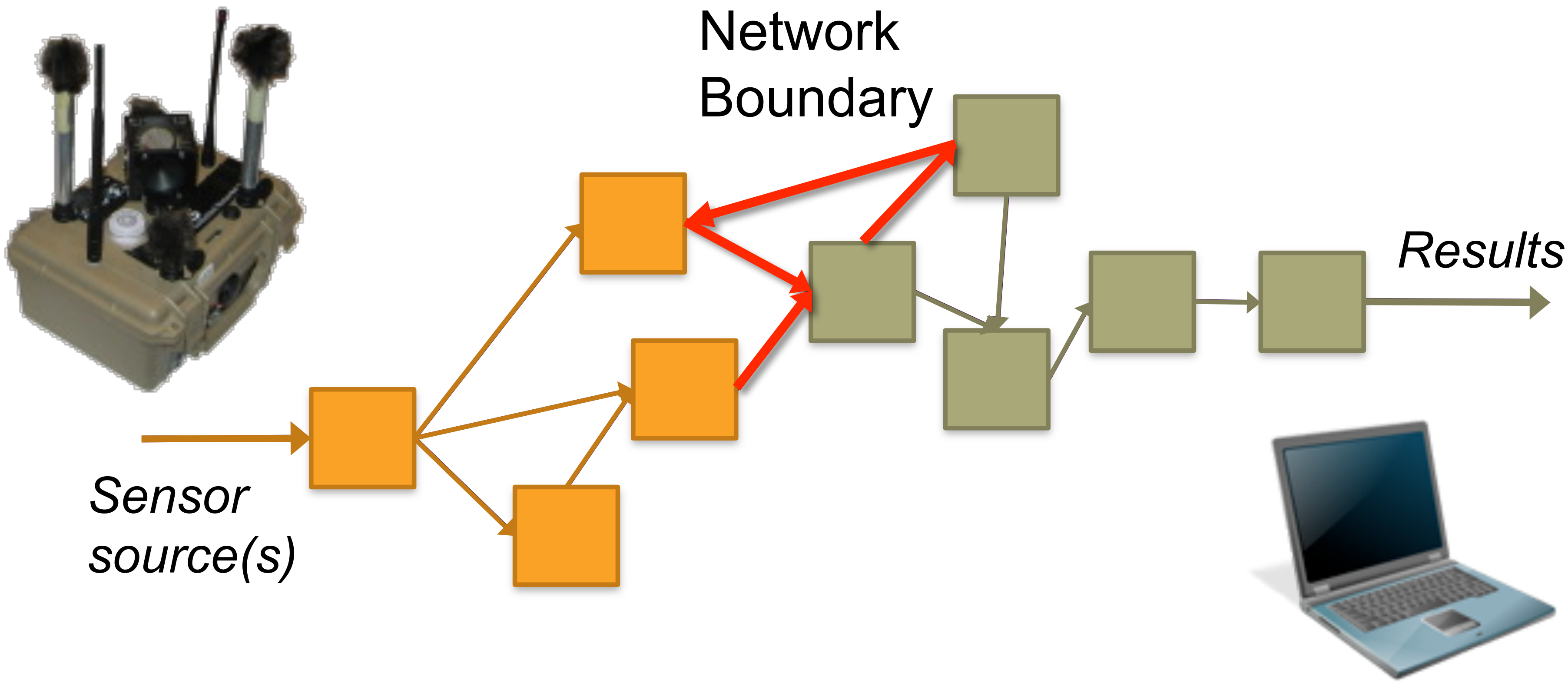


Linux
microserver

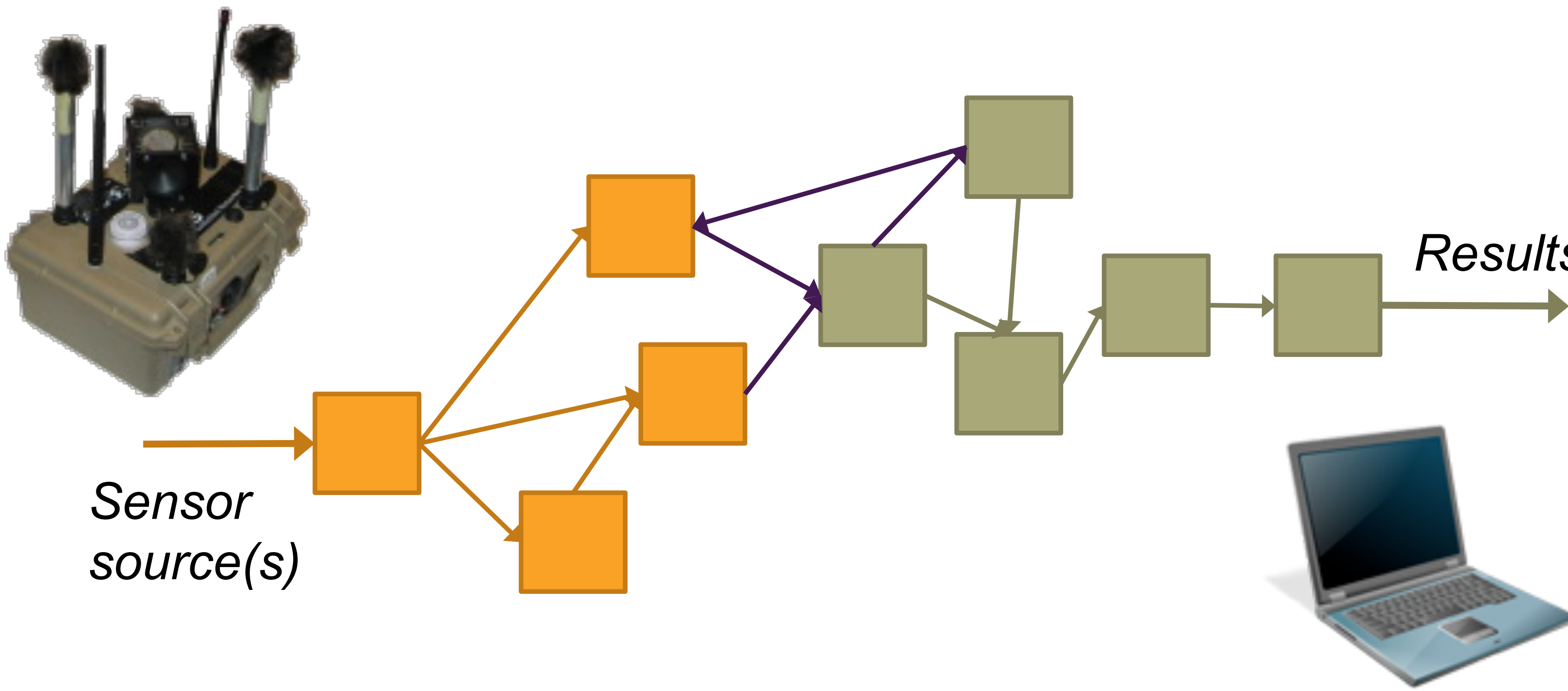
C++
Python
Java

Mix and Match!

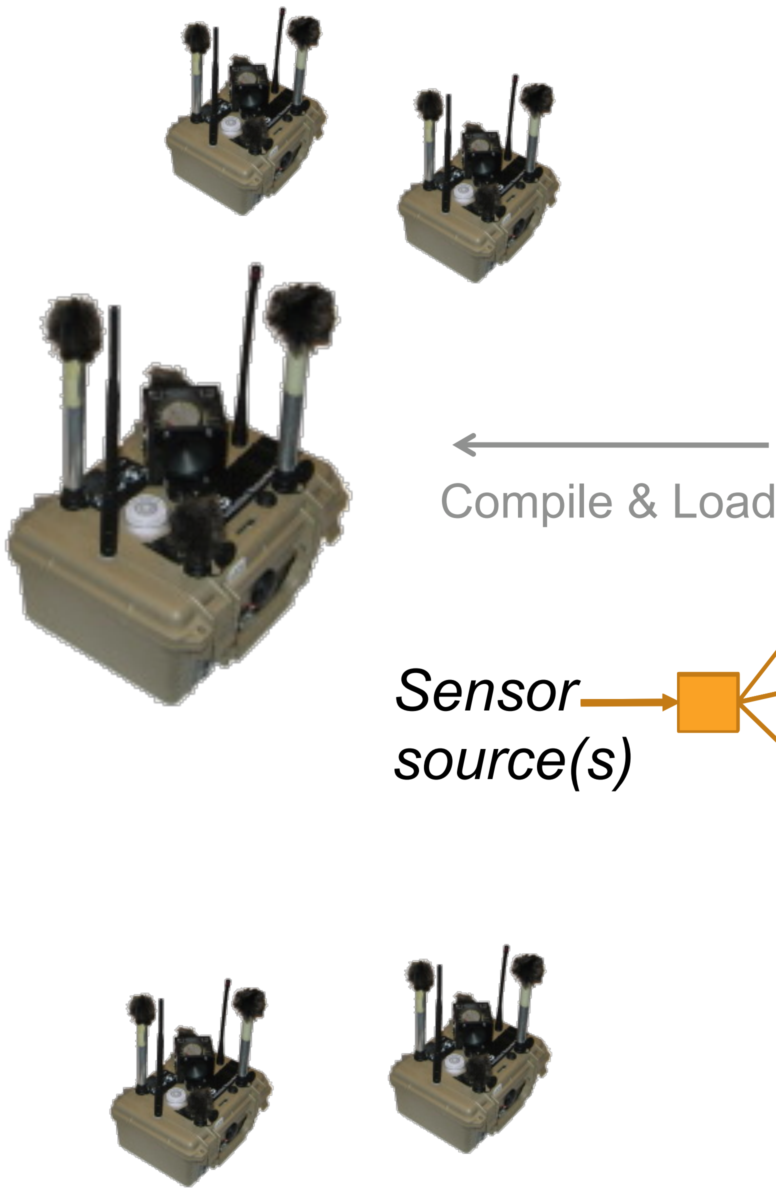
+ Contributions

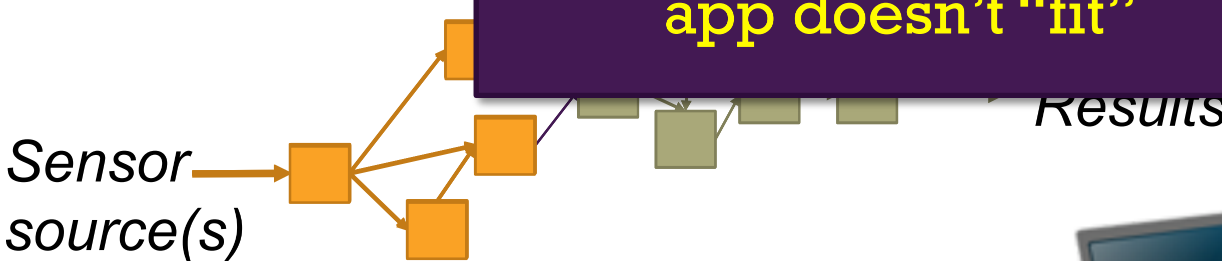


+ Contributions



+ Contributions

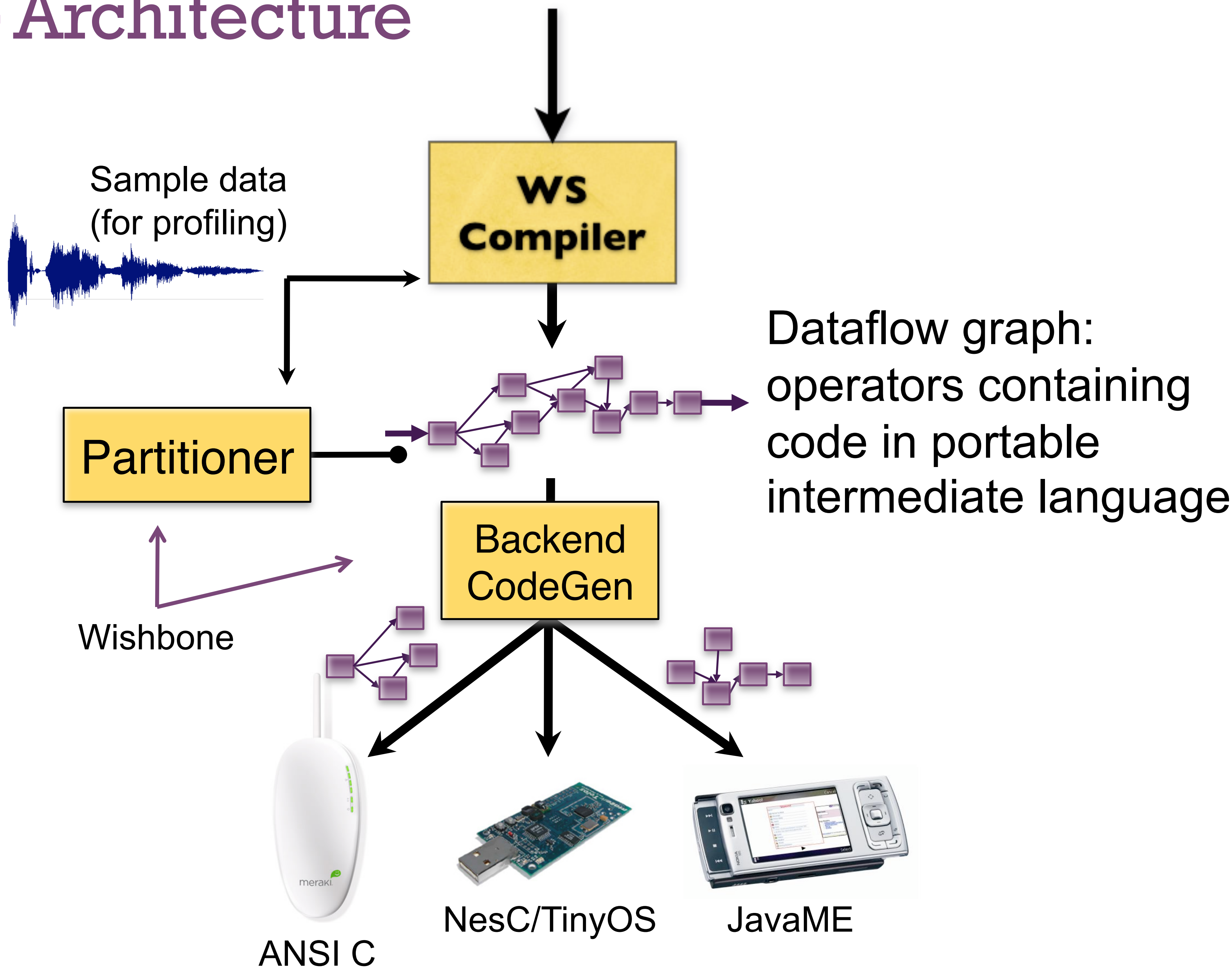




Contributions

- First broadly portable sensenet programming
- Partitioning algorithm
- Optimize CPU/radio tradeoff even if app doesn't "fit"

+ Architecture

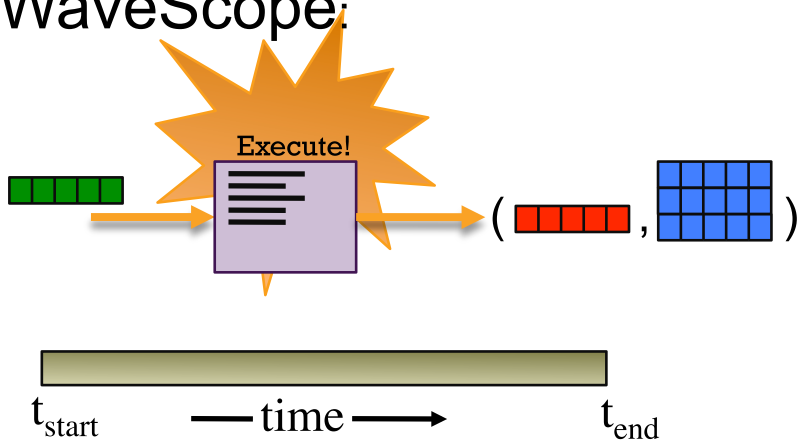


+ Targeting TinyOS

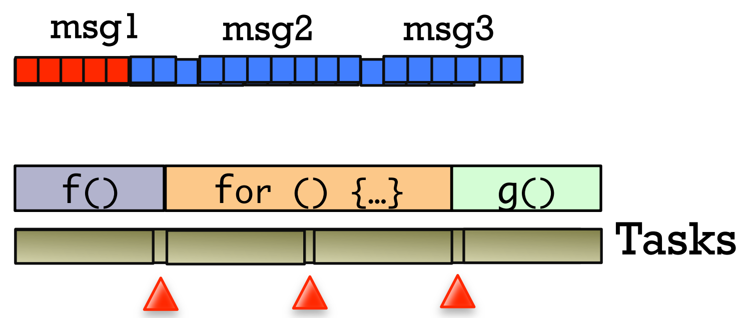
Task granularity, messaging model

- 16 bit microcontroller
- 10K RAM
- No mem. protection
- No threads

WaveScope:



TinyOS:



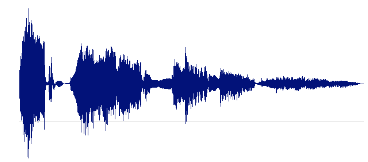
Profile-directed Cooperative Multitasking:

```
iterate x in S {
  f(); yield();
  for(i=…) {
     … if (i==387) yield(); …
  }
  yield(); g();
}
```

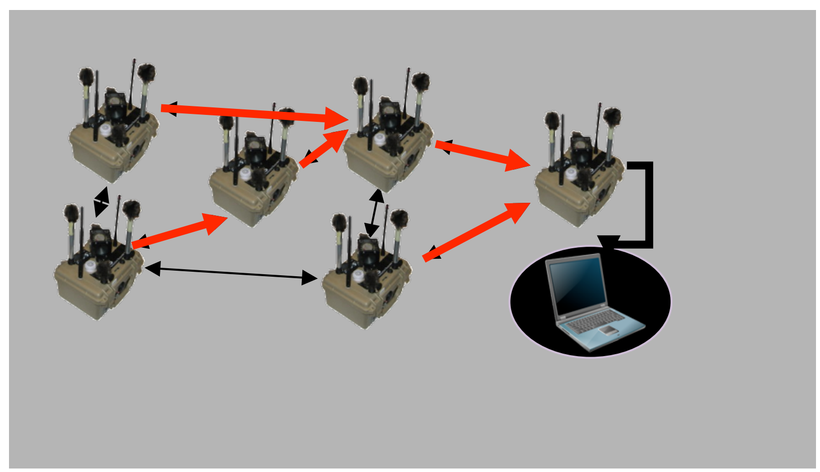
Same goal as *Protothreads*

+ Profiling Streams and Operators

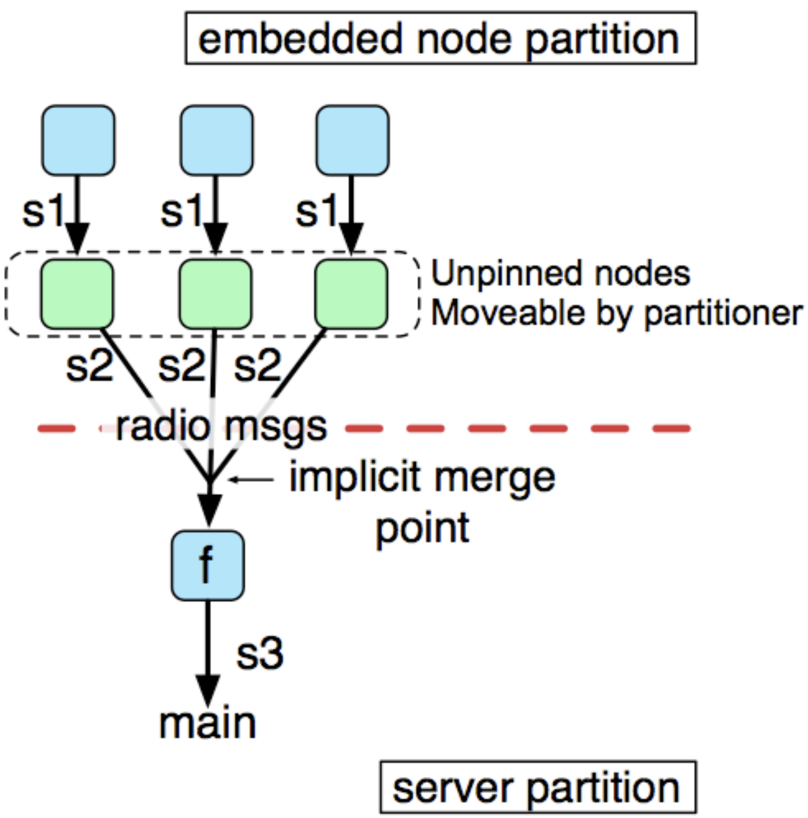
- Every sensor source is paired with sample data
  - Includes timing info
- Measure rates, execution times
- Separately: profile network channel in deployment environment
  - ➔ per-node send rate



```
audioStream =
  IFPROF(readFile("foo8kHz",
           readSensor()))
```



State, Replication, and Pinning



Pinning Constraints

- All stateless ops: unpinned
- Stateful replicated ops: unpinned
- Stateful global ops: *pinned to server – don't distribute!*

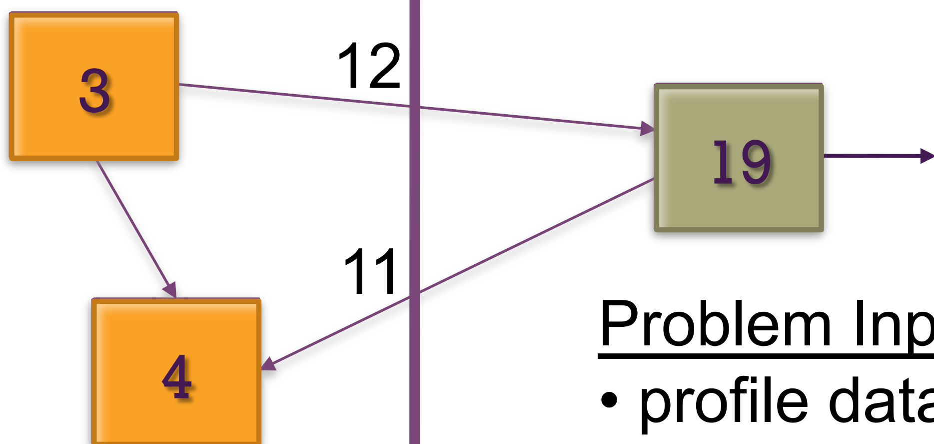
Problem Scenario

Embedded Node

Server / Base Station

<u>CPU:</u>
7

<u>Network:</u>
23



<u>Problem Inputs</u>

- profile data: net, cpu
- network channel capacity

NP-Hard

Network Boundary

Partitioning Algorithm: Integer linear program formulation

Tricky bit (see paper):

Relating *f* and *g* while staying linear

$f_u \quad \{0,1\}$ where

$g_{uv} \quad \{0,1\}$ wher

3 Parameters

C, N, α

unds

- $cpu < C$ where $cpu = \underset{u}{\quad} f_u(compute_u)$
- $net < N$ where $net = \underset{uv \quad Edges}{\quad} g_{uv}(data_{uv})$

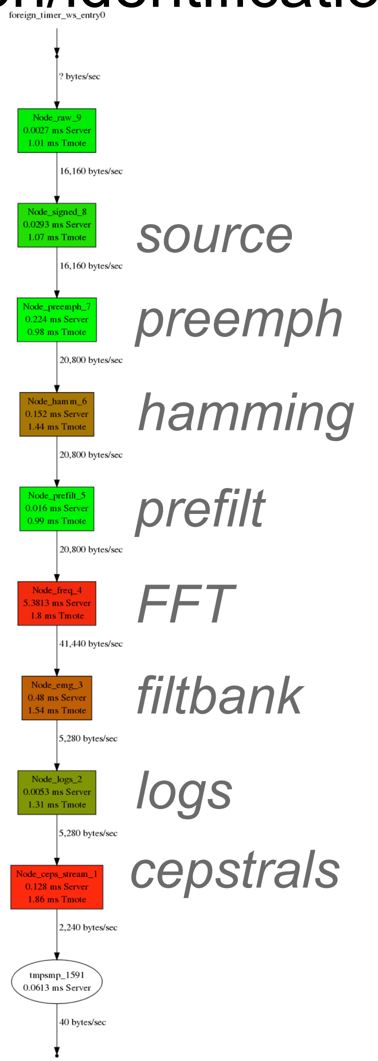
- Minimize objective function

$$\min(\quad \lozenge cpu + net)$$

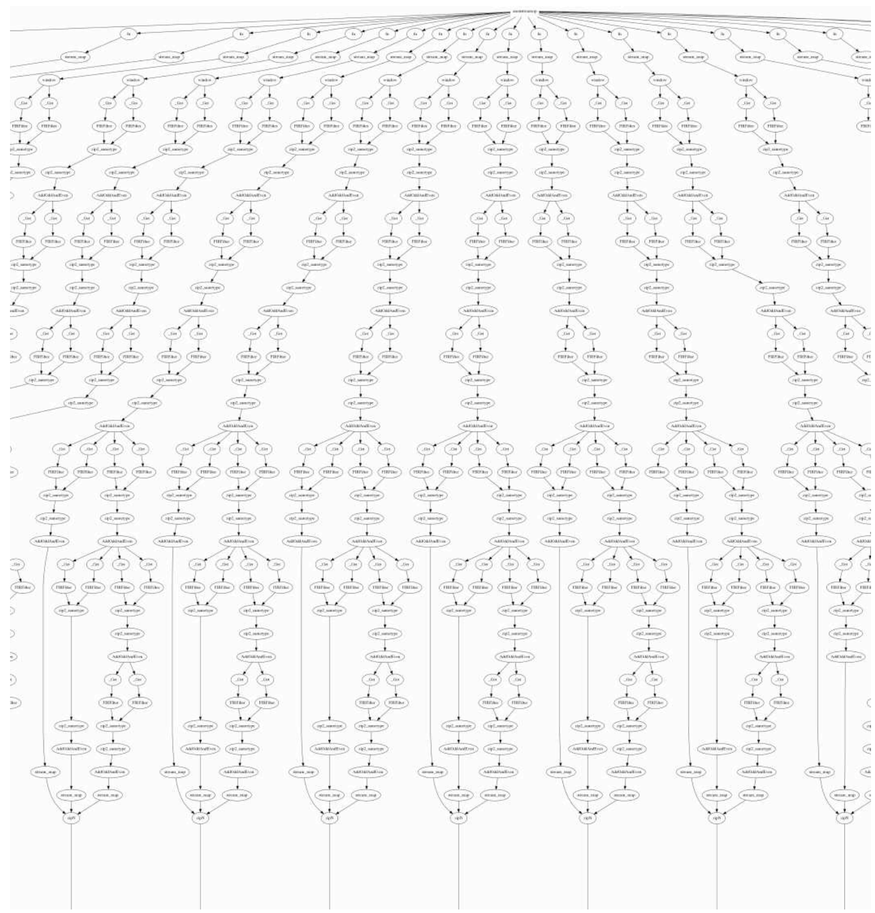
Proxy for Energy

+ Evaluation: Two Applications

Human speech detection/identification

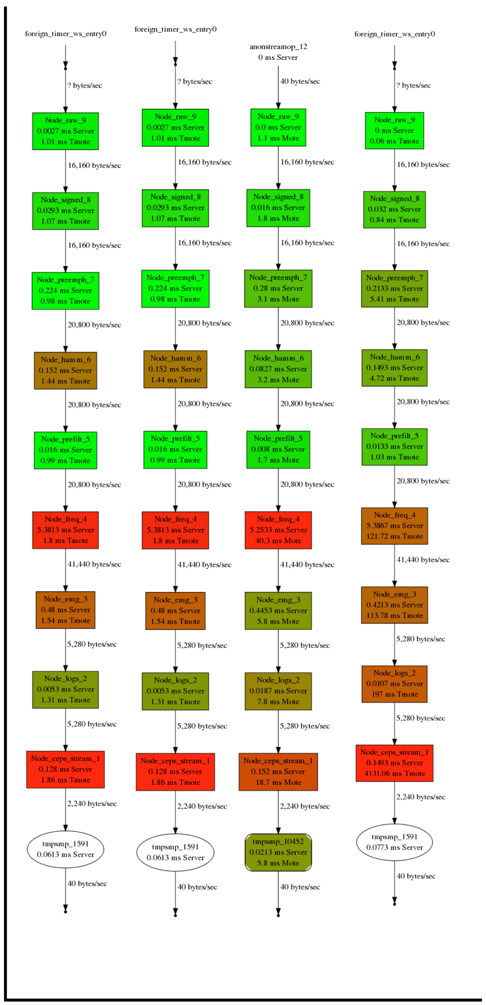


EEG-based seizure onset detection

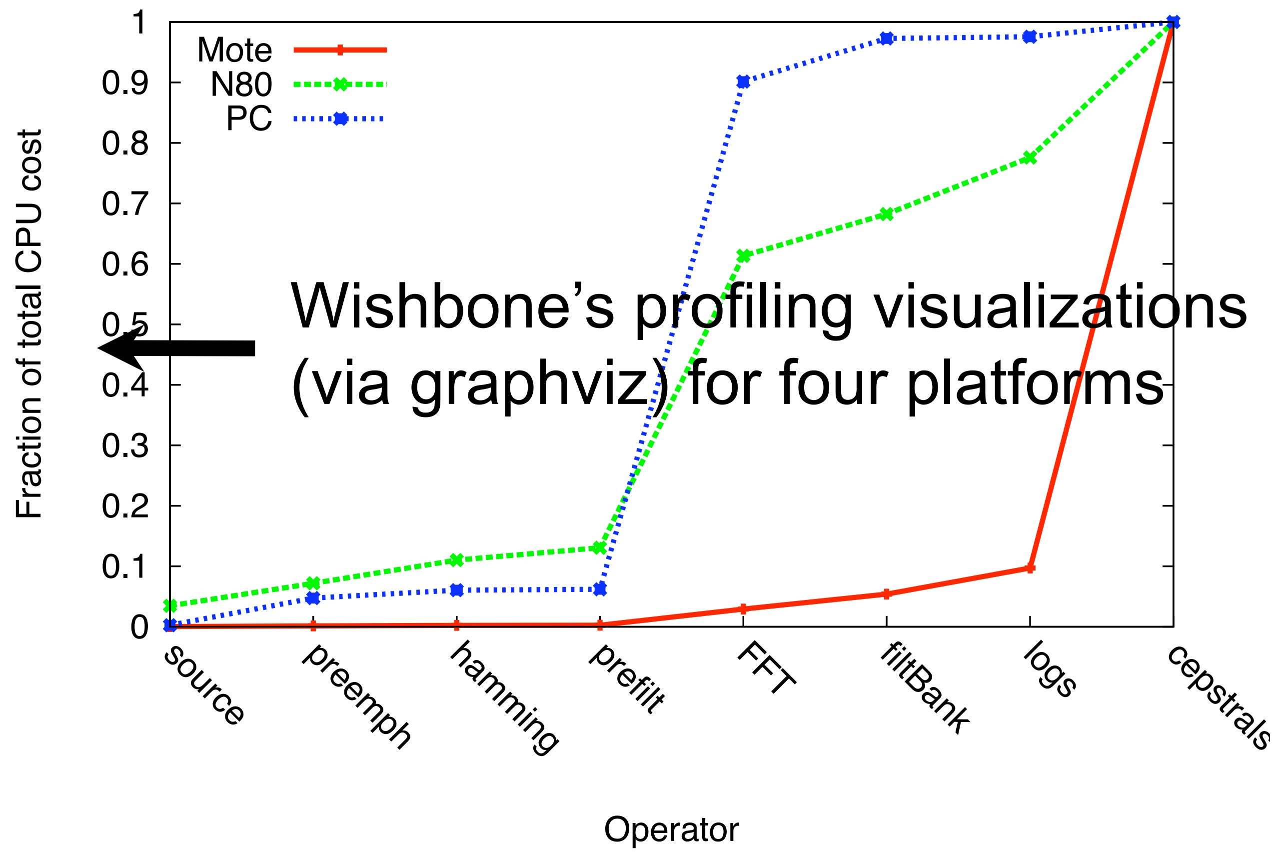


1400 operators

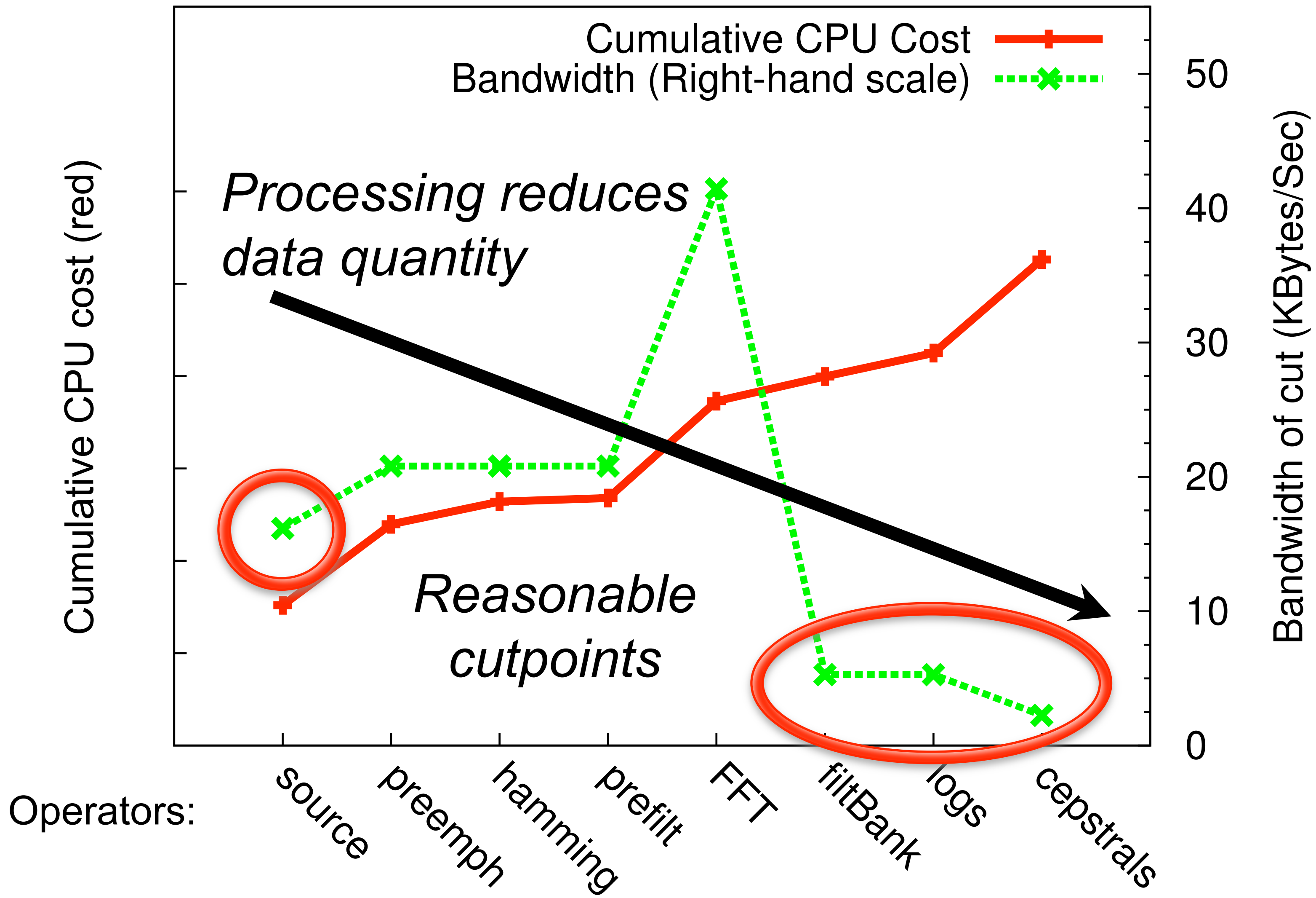
+ Observation: Relative cost varies by platform



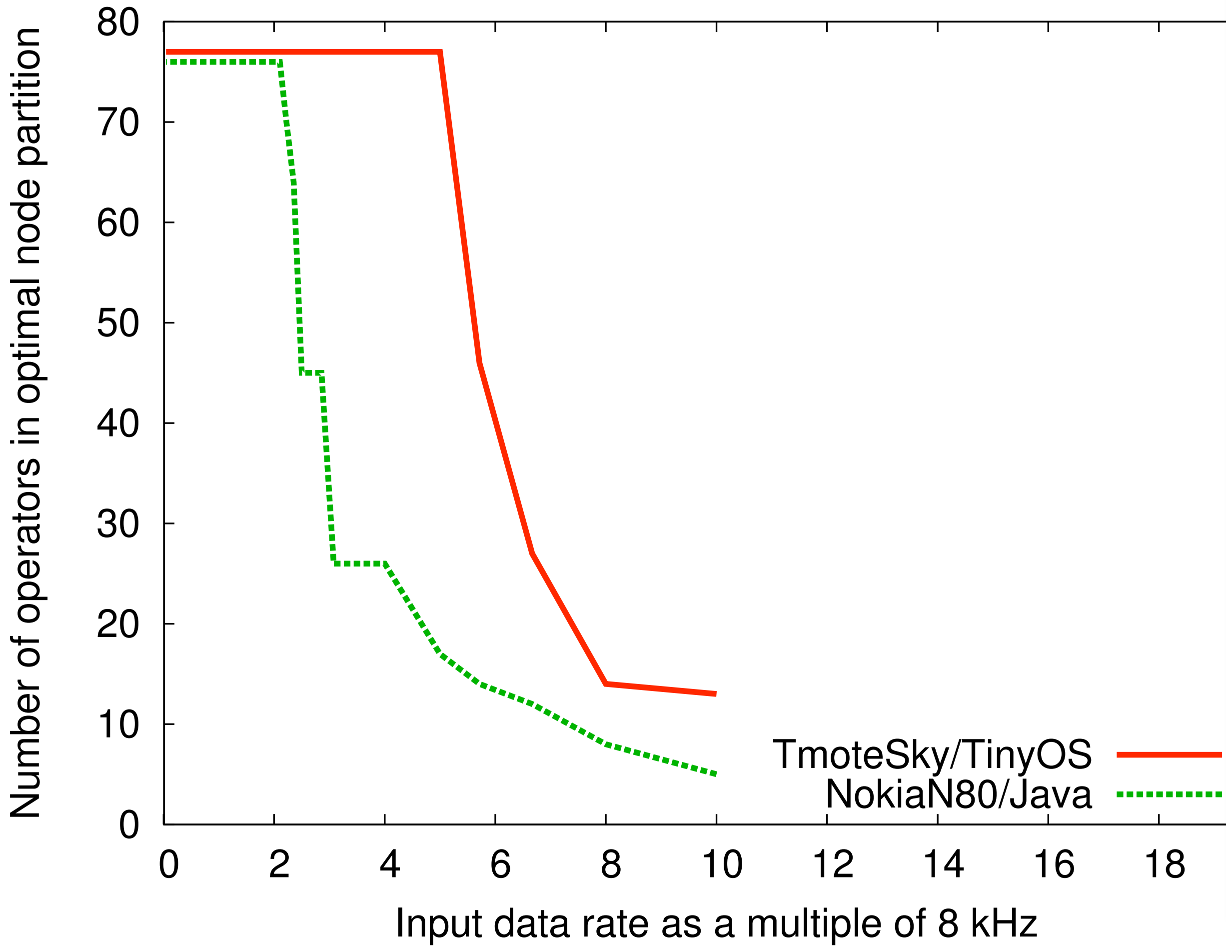
Wishbone’s profiling visualizations (via graphviz) for four platforms



+ Visualizing Profile Data: Bandwidth vs. Compute



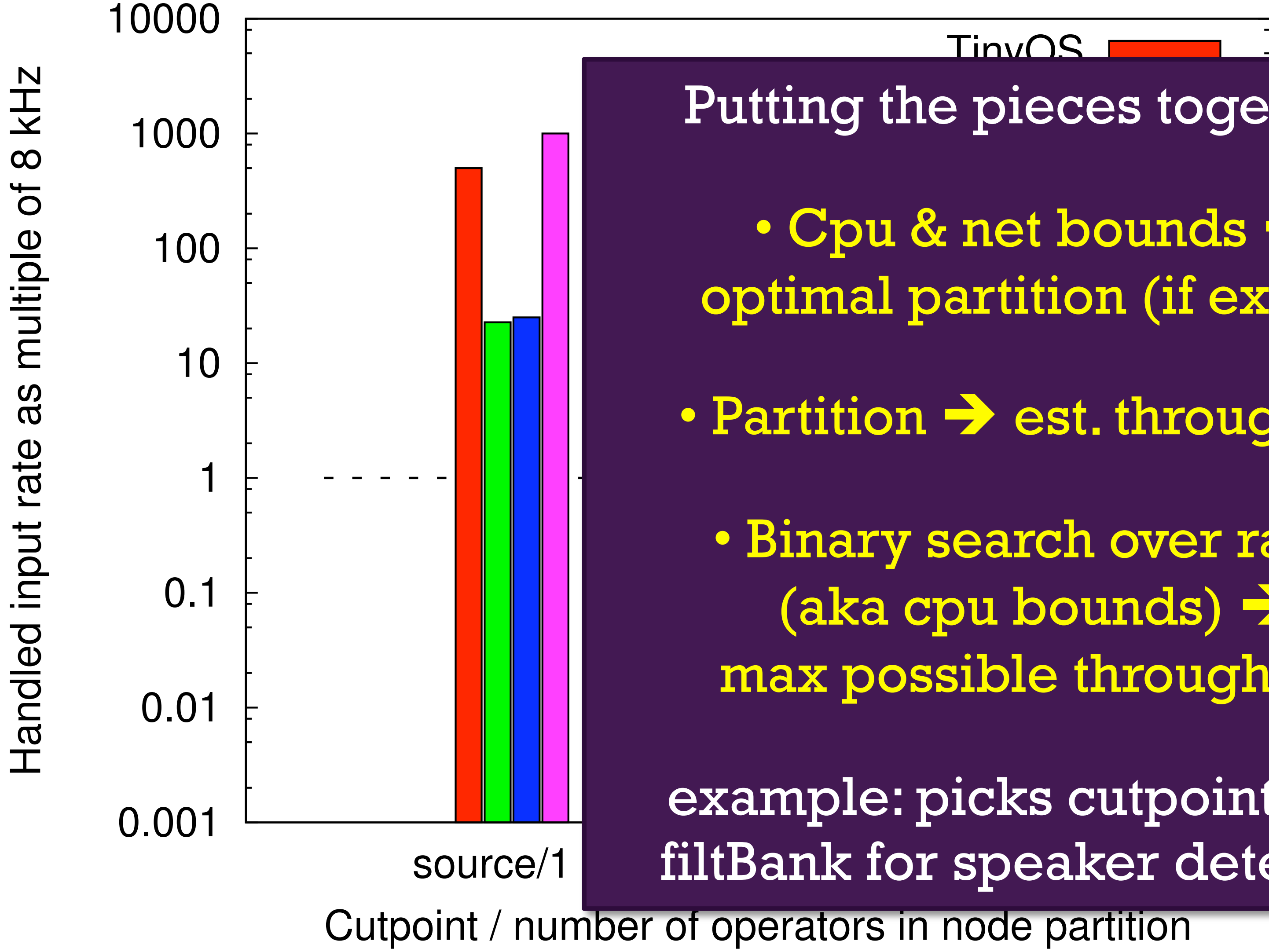
Optimal partitions across platforms



EEG Application (1 of 22 channels)

Each line represents 2100 partioner-runs

+ Speaker Detection: CPU performance across partitions/platforms



Putting the pieces together:

• Cpu & net bounds ➔ optimal partition (if exists)

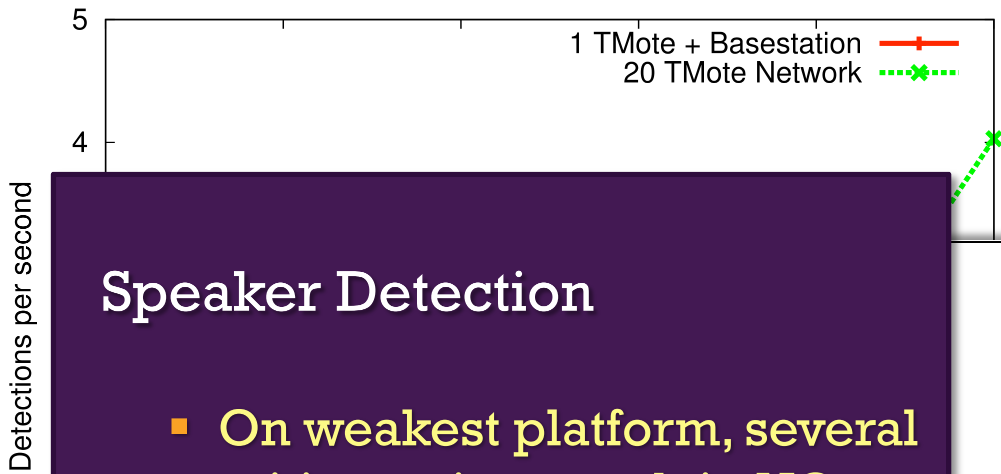
• Partition ➔ est. throughput

• Binary search over rates (aka cpu bounds) ➔ max possible throughput

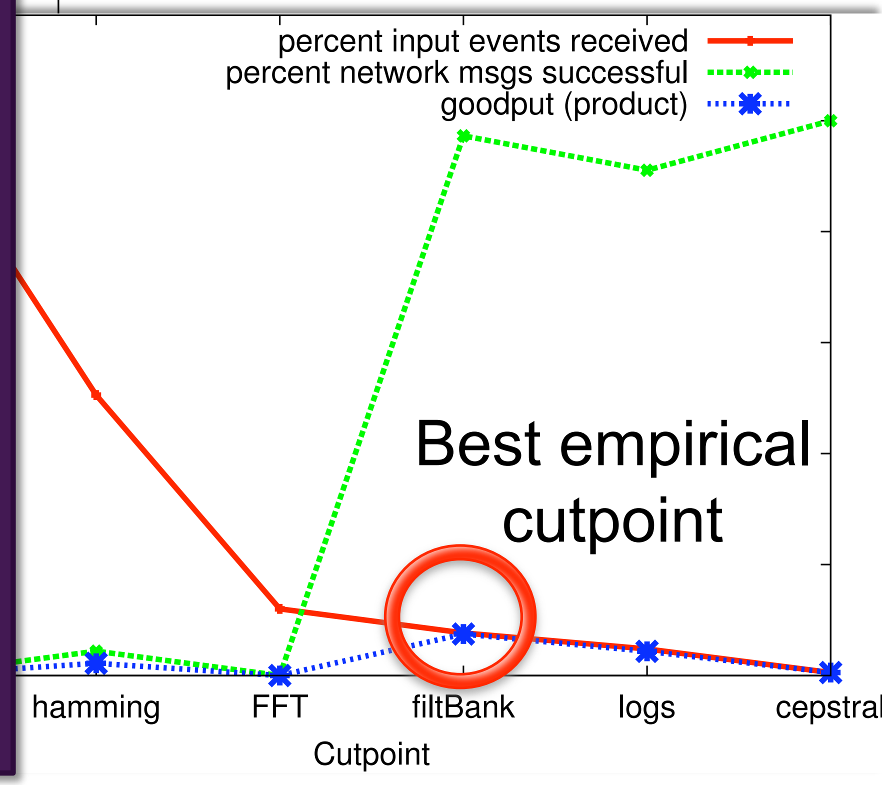
example: picks cutpoint after filtBank for speaker detection

Speaker Detection Application

+ Groundtruth: Testbed deployment, 20 motes



How many detections can we actually get out of the network?



Speaker Detection

- On weakest platform, several partition points result in NO data
- But among "working" partition points, best is 20X better than worst

Pick the right one!

+ Related Work

- Graph partitioning for scientific codes
  - balanced, heuristic – e.g. Zoltan
- Task scheduling, commonly list scheduling
- Dynamic: Map-reduce, Condor, etc.
- Sensor network context: Tenet and Vango
  - Linear pipeline of operators
  - Manual partition
  - Run TinyOS code on both server and sensor

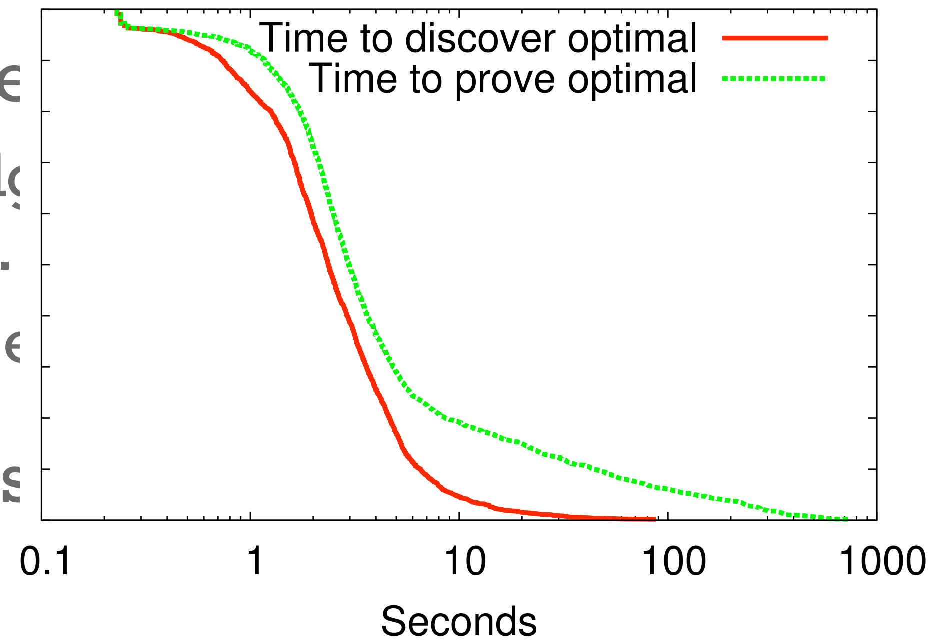
CONCLUSION

- **WS/Wishbone System**: *(available, open source)*
  - Convenient, efficient way to run one program on many devices
  - Dataflow graph profiling and visualization extremely useful even outside of auto partitiioning
- Contributions
  - Algorithm for partitioning streaming programs
  - Backend(s) enable high level programming even on motes
- Techniques for node/server partitioning could be extended to handle heterogeneous cores intra-device.

Contact: Ryan Newton - newton@mit.edu

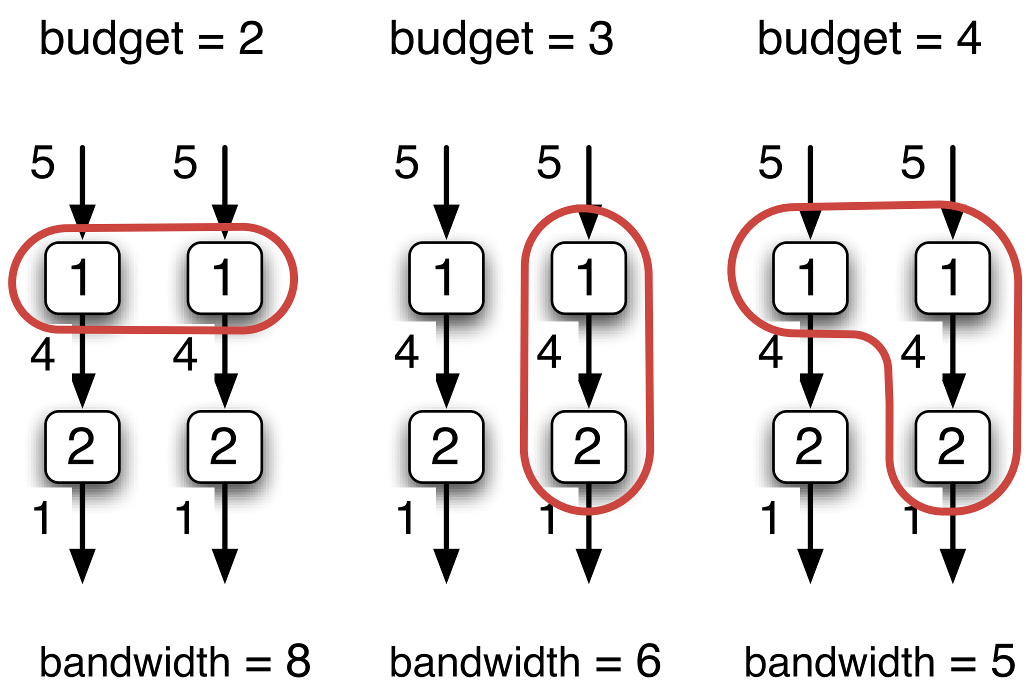
Partitioning: Algorithm Runtime

- Graph Preprocessing ste
  - Merge vertices until all edg monotonically decreasing.
  - Eliminates the majority of e
- Even without preprocess



  - 8000 runs,
  - partitioning the 1400-node EEG dataflow graph,
  - with different CPU budget,
  - took under 10 seconds 95% of the time.
  - But there is a long tail… luckily ILP solvers produce approximate solutions as well!

Motivating Example



Unstable optimal partition.
Flips between horizontal and vertical partition.