WaveScope

Ryan Newton

MIT/CSAIL

Work with:
Lewis Girod, Kyle Jamieson, Yuan Mei, Stan Rost,
Arvind Thiagarajan, Hari Balakrishnan, Sam Madden

http://wavescope.csail.mit.edu/

Motivation:
Stream + Signal Processing

Motivation:
Stream + Signal Processing

- Pipeline leak detection and localization

Are there anomalies in the frequency response to an introduced pulse?

http://wavescope.csail.mit.edu/

Motivation: Stream + Signal Processing

- Pipeline leak detection and localization
- Seizure onset detection using EEG

Are there anomalies in the frequency response to an introduced pulse?

Is a seizure imminent given signals from various brain regions?

http://wavescope.csail.mit.edu/

Motivation: Stream + Signal Processing

- Pipeline leak detection and localization
- Seizure onset detection using EEG
- *In situ* animal behavior studies

Are there anomalies in the frequency response to an introduced pulse?

Is a seizure imminent given signals from various brain regions?

What time ranges contained marmot calls?

http://wavescope.csail.mit.edu/

Motivation: Stream + Signal Processing

- Pipeline leak detection and localization
- Seizure onset detection using EEG
- *In situ* animal behavior studies

Are there anomalies in the frequency response to an introduced pulse?

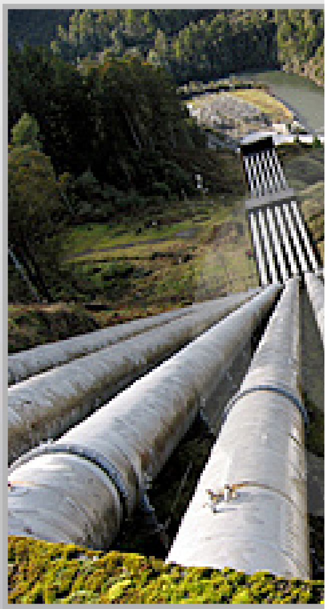
Is a seizure imminent given signals from various brain regions?

What time ranges contained marmot calls?

http://wavescope.csail.mit.edu/

Motivation: Stream + Signal Processing

- Pipel
- Seizu
- *In sit*



Limitations of Streaming DBMS

- Difficult to extend operator set
  - Outcalls to Matlab, etc
- Embedded support
- High per-sample/operator overhead

Are there anomalies in the frequency response to an introduced pulse?

Is a seizure imminent given signals from various brain regions?

What time ranges contained marmot calls?

http://wavescope.csail.mit.edu/

WaveScope Features

WaveScope Features

High data-rate

4 channels
x 48 khz
= 400,000 bytes/sec (per node)

x 20 nodes

WaveScope Features

Embedded, low-power devices

High data-rate

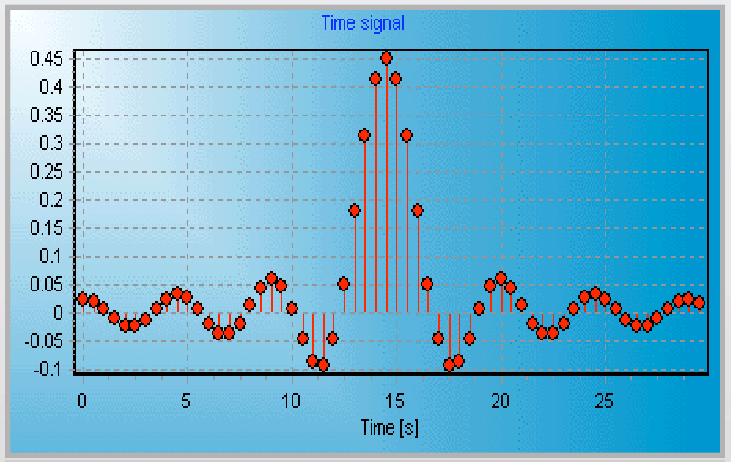
4 channels
x 48 khz
= 400,000 bytes/sec (per node)

x 20 nodes

WaveScope Features

Embedded, low-power devices

High data-rate



Signal-oriented data-model

4 channels
x 48 khz
= 400,000 bytes/sec (per node)

x 20 nodes

WaveScope Features

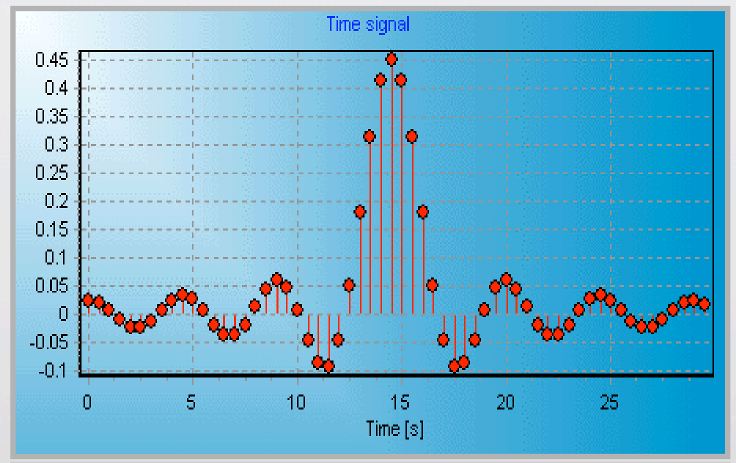
High data-rate

4 channels
x 48 khz

= 400,000 bytes/sec (per node)

x 20 nodes

Embedded, low-power devices



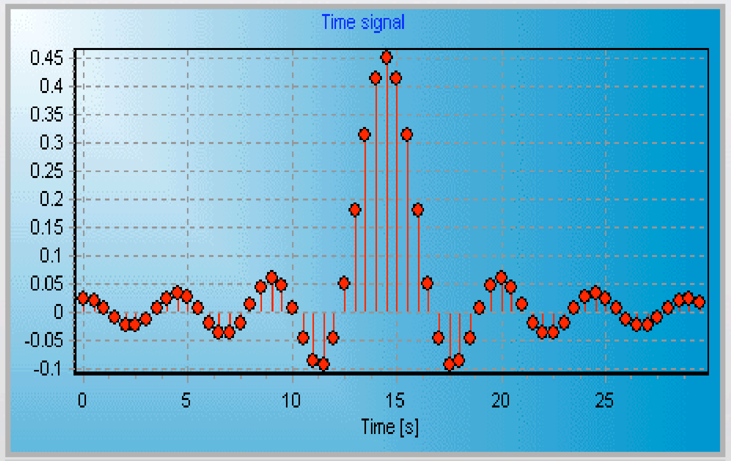
Signal-oriented data-model

- Flexible windowing
- Efficient time-stamping

WaveScope Features

Embedded, low-power devices

High data-rate



Signal-oriented data-model

- Flexible windowing
- Efficient time-stamping

4 channels
x 48 khz
= 400,000 bytes/sec (per node)

x 20 nodes

Ch0 = AudioSource(0, 48000, 1024);
S = stream_map(FFT, **Ch0**);

WaveScript Language

WaveScope Features

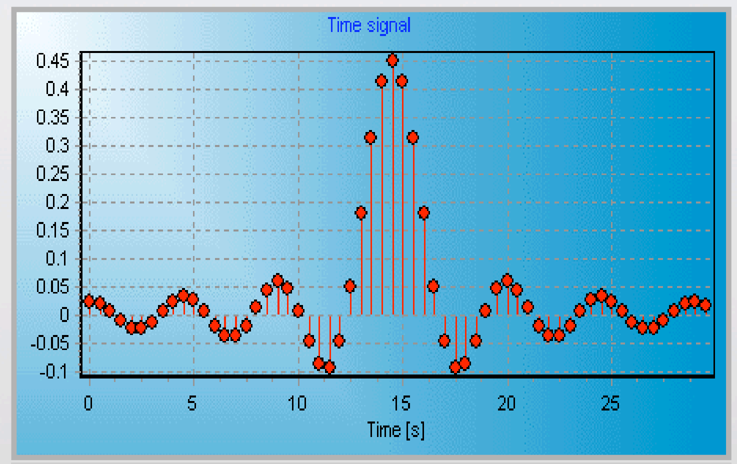
High data-rate

4 channels
x 48 khz
= 400,000 bytes/sec (per node)

x 20 nodes

Embedded, low-power devices

Signal-oriented data-model



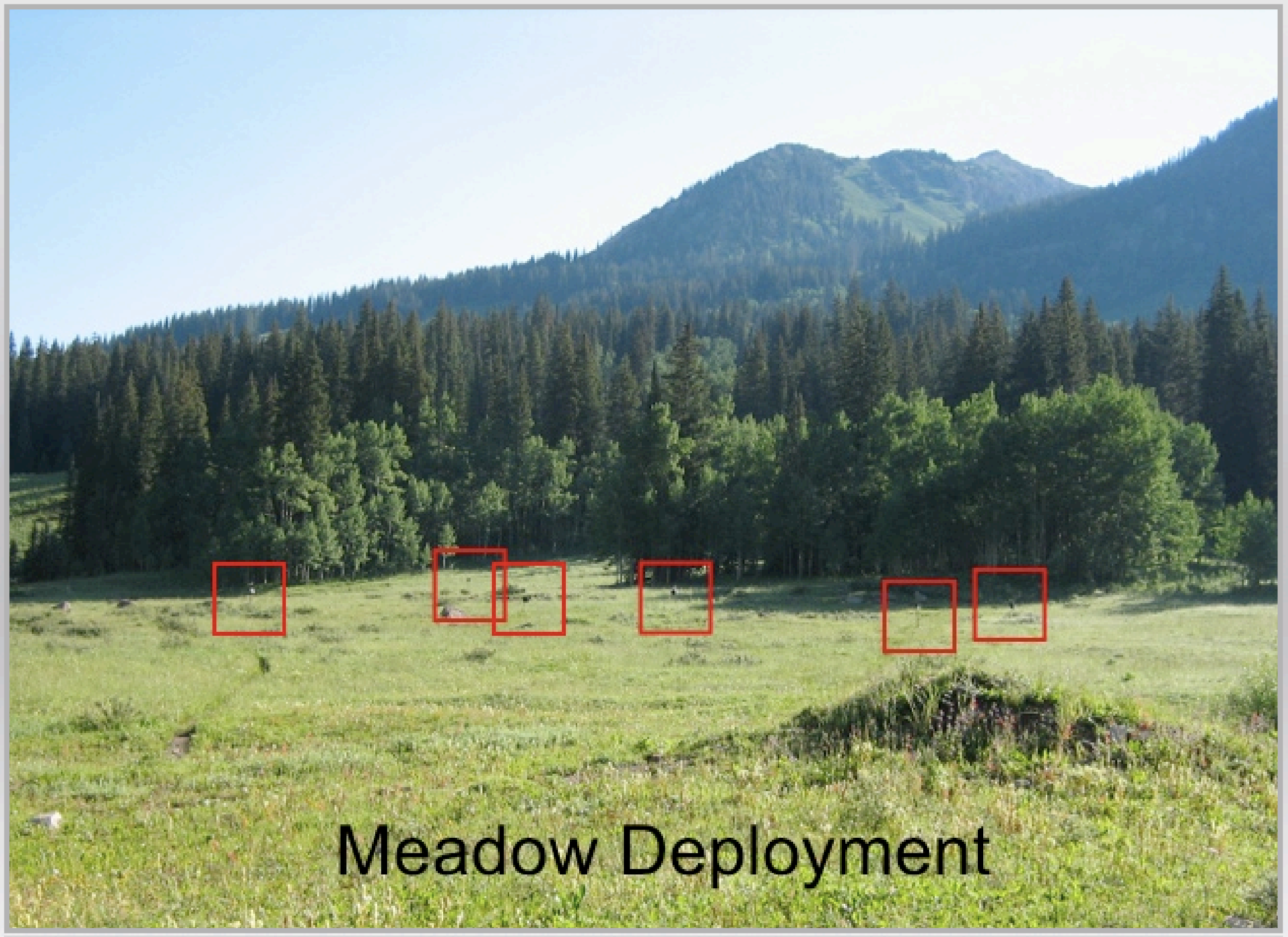
- Flexible windowing
- Efficient time-stamping

WaveScript Language

```
Ch0 = AudioSource(0, 48000, 1024);
S = stream_map( FFT, Ch0 );
```

- *Not* StreamSQL
- Write script to <u>*generate*</u> query network
- Query network is optimized, compiled to native code, executed by engine

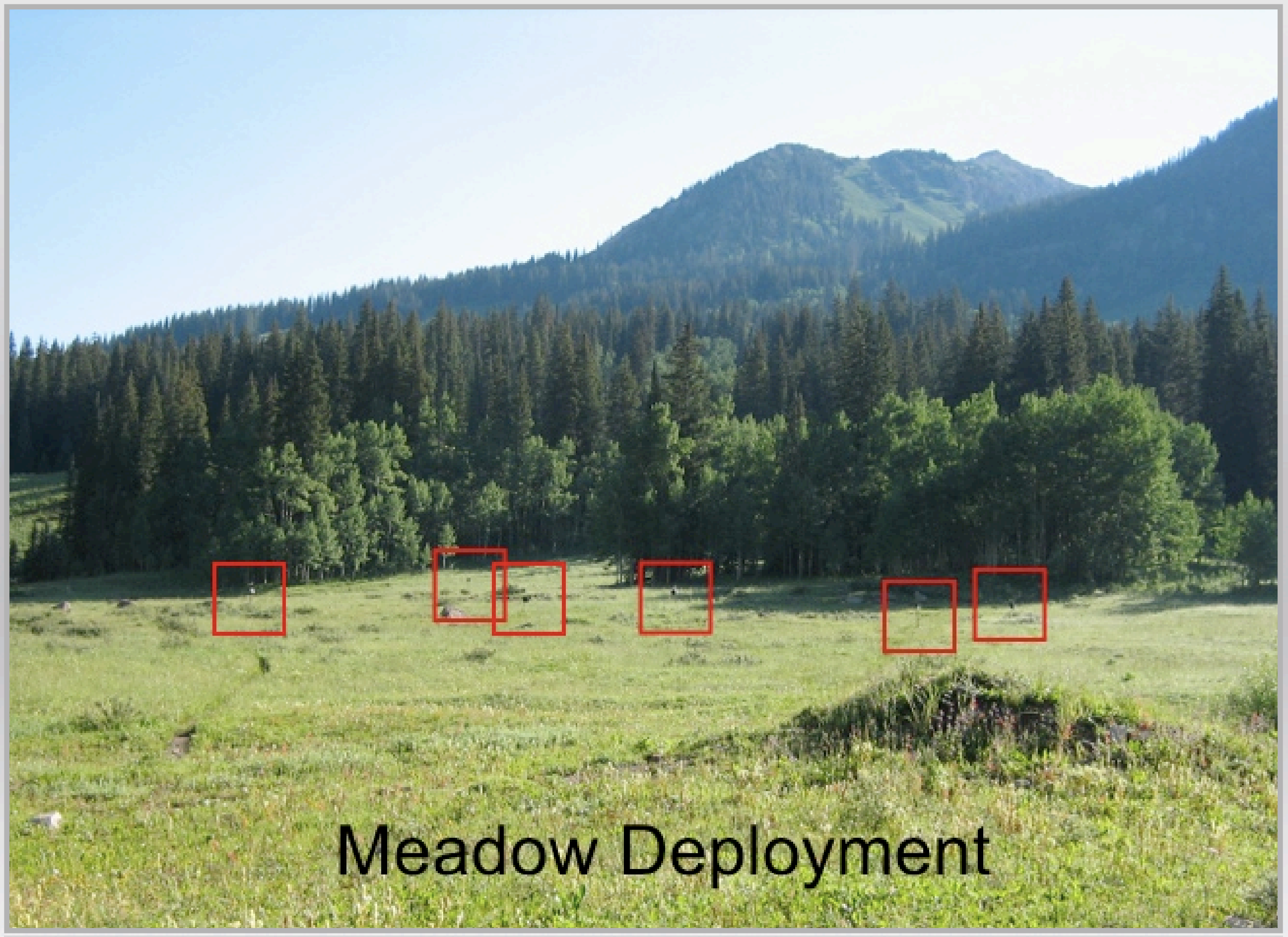
Drilling down: Marmot-call application



http://wavescope.csail.mit.edu/

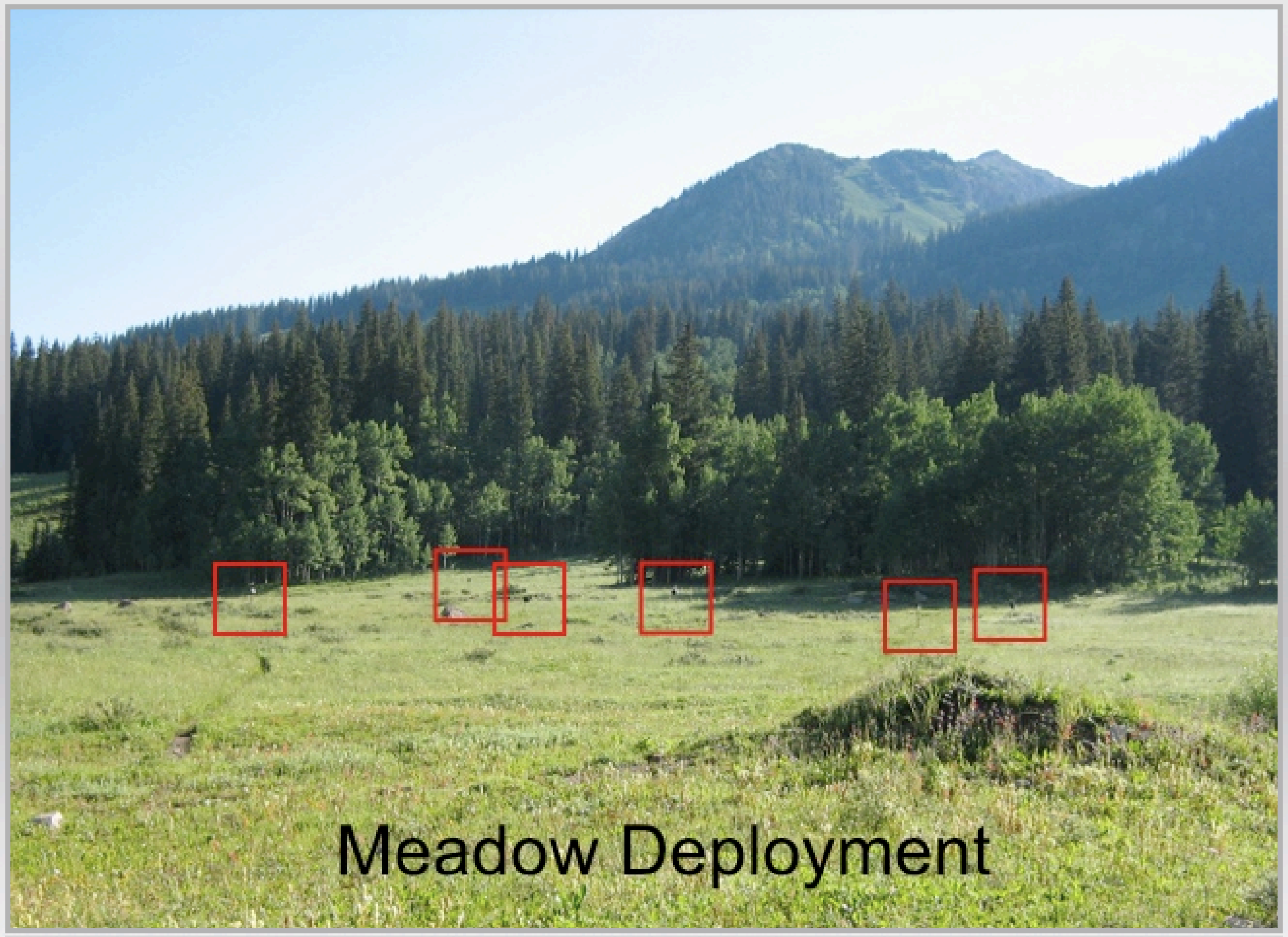
Drilling down: Marmot-call application

- Goal: study calling behavior.



Drilling down: Marmot-call application

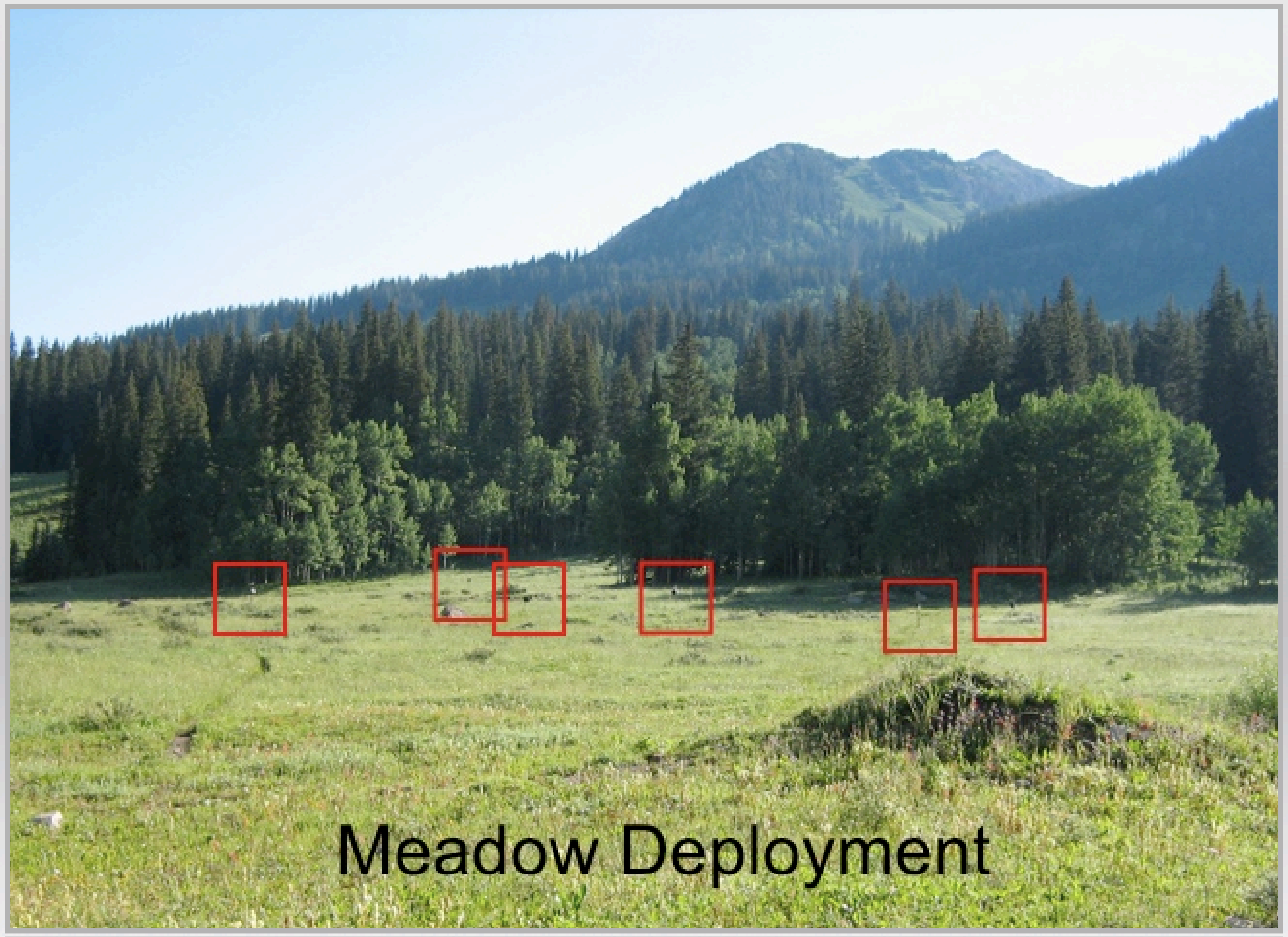
- Goal: study calling behavior.



http://wavescope.csail.mit.edu/

Drilling down: Marmot-call application

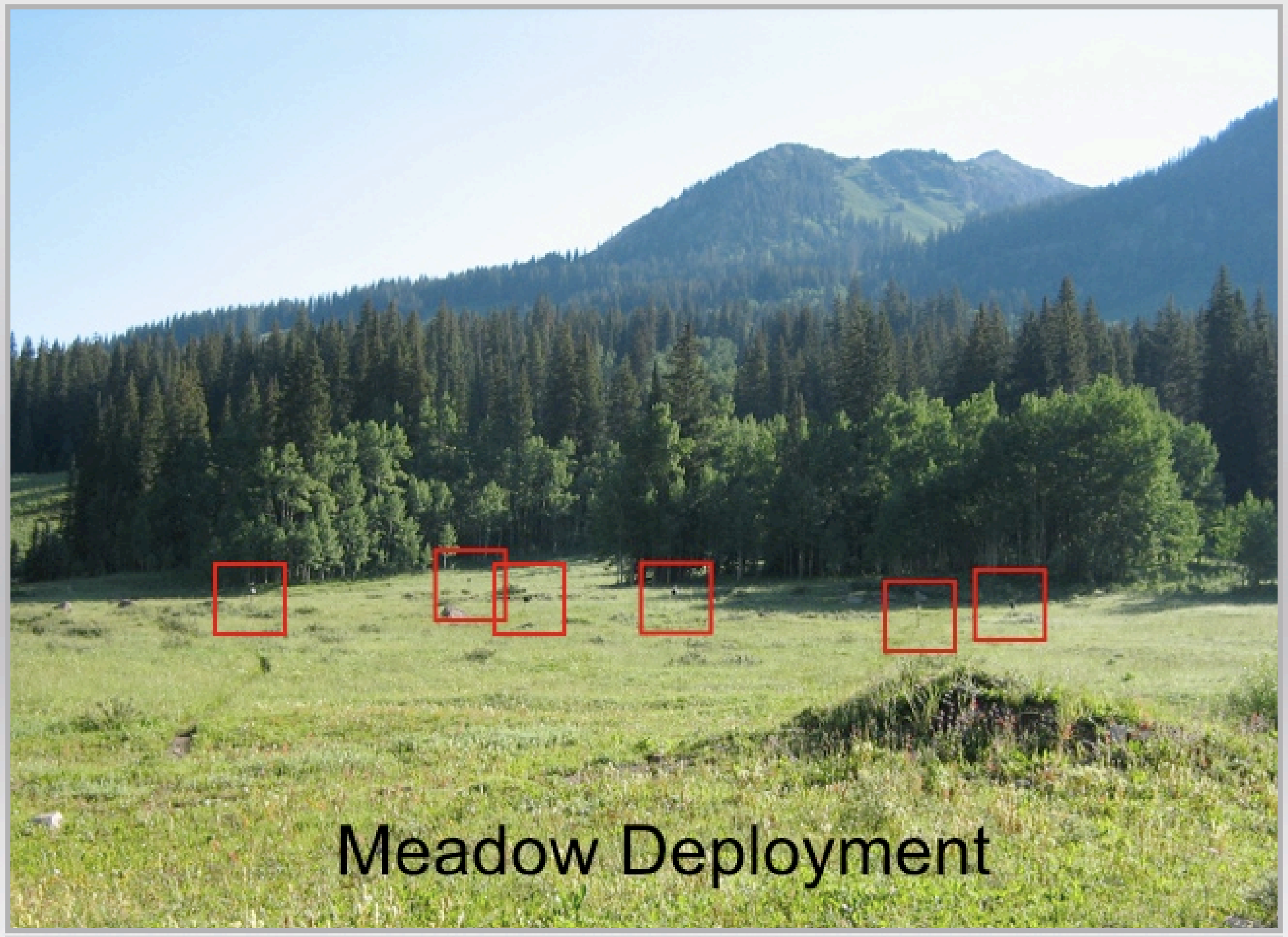
- Goal: study calling behavior.



http://wavescope.csail.mit.edu/

Drilling down: Marmot-call application

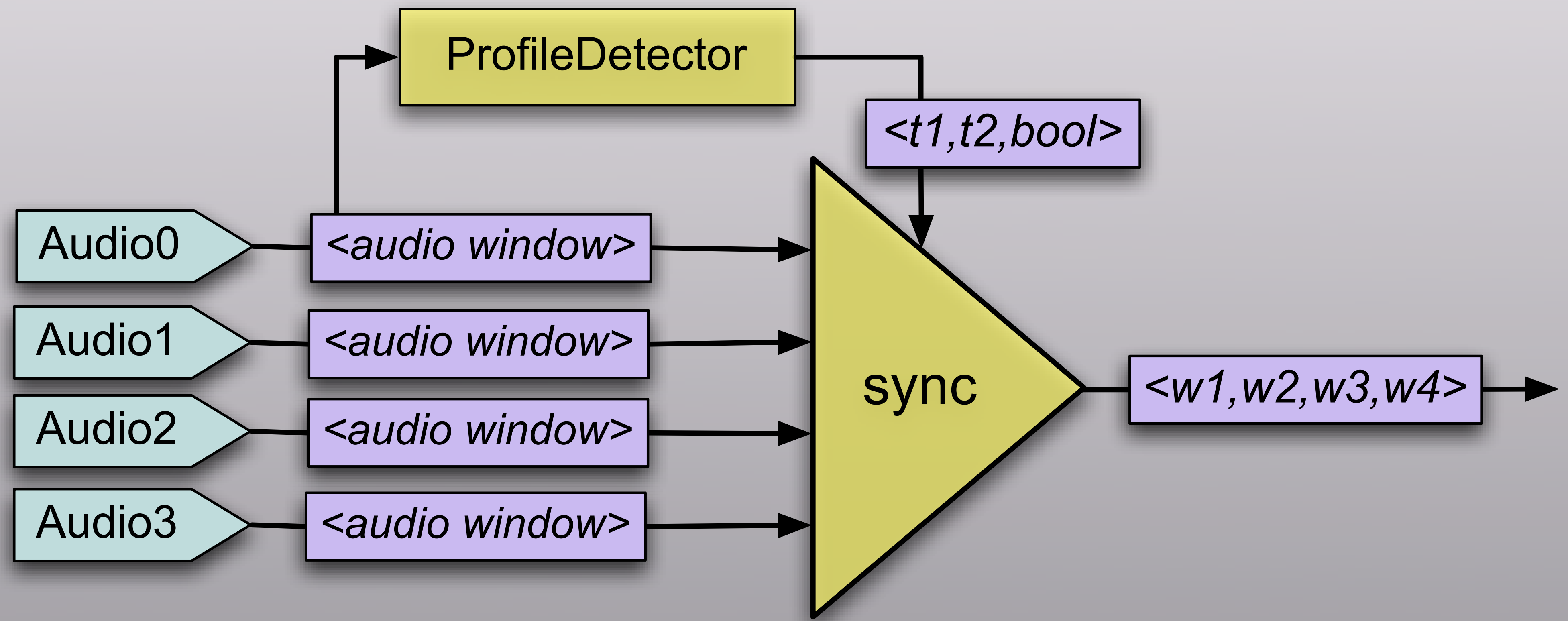
- Goal: study calling behavior.
- Detect, record, localize, classify.



http://wavescope.csail.mit.edu/

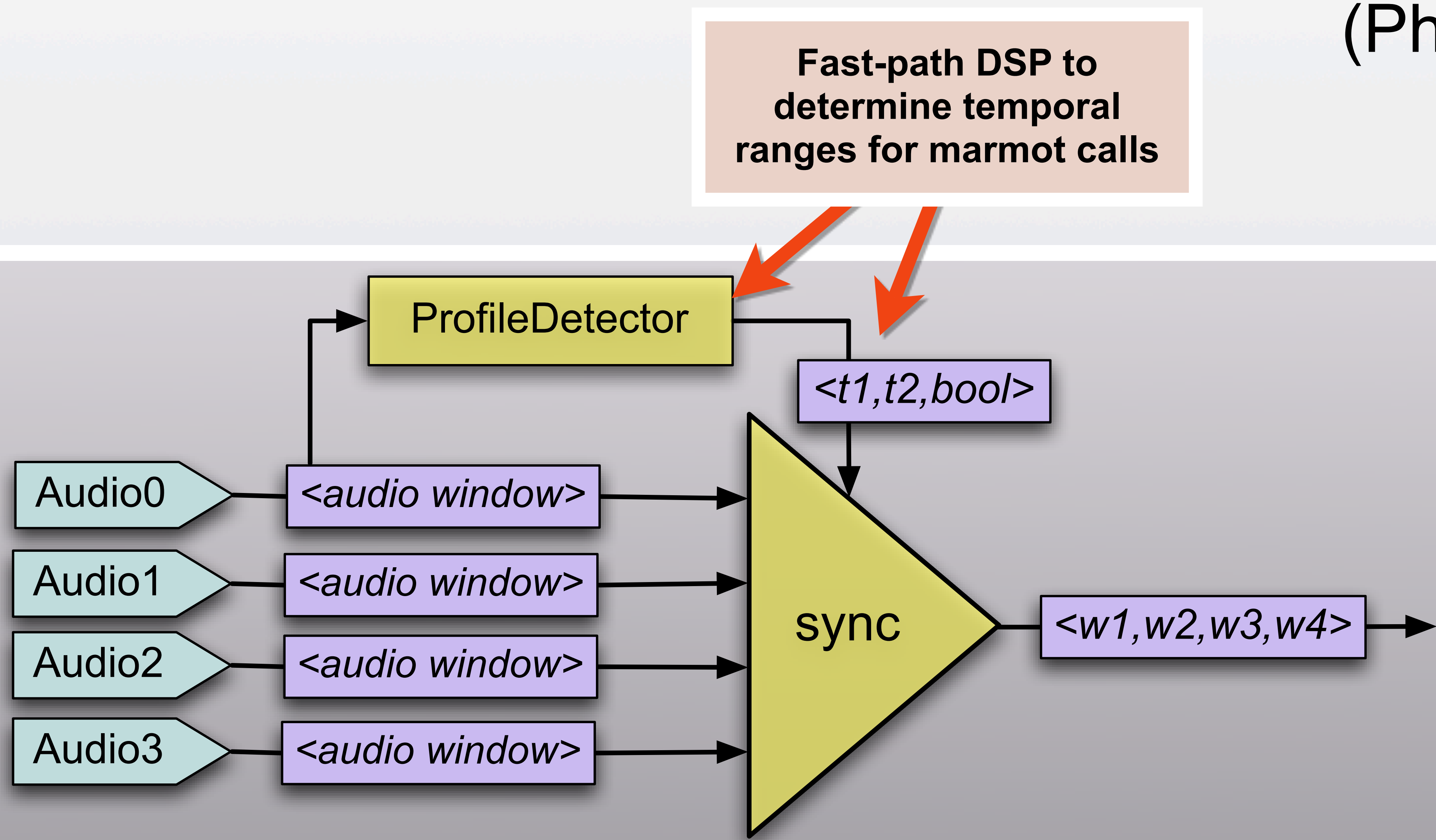
Schematic of Marmot-detector

(Phase1)



Schematic of Marmot-detector

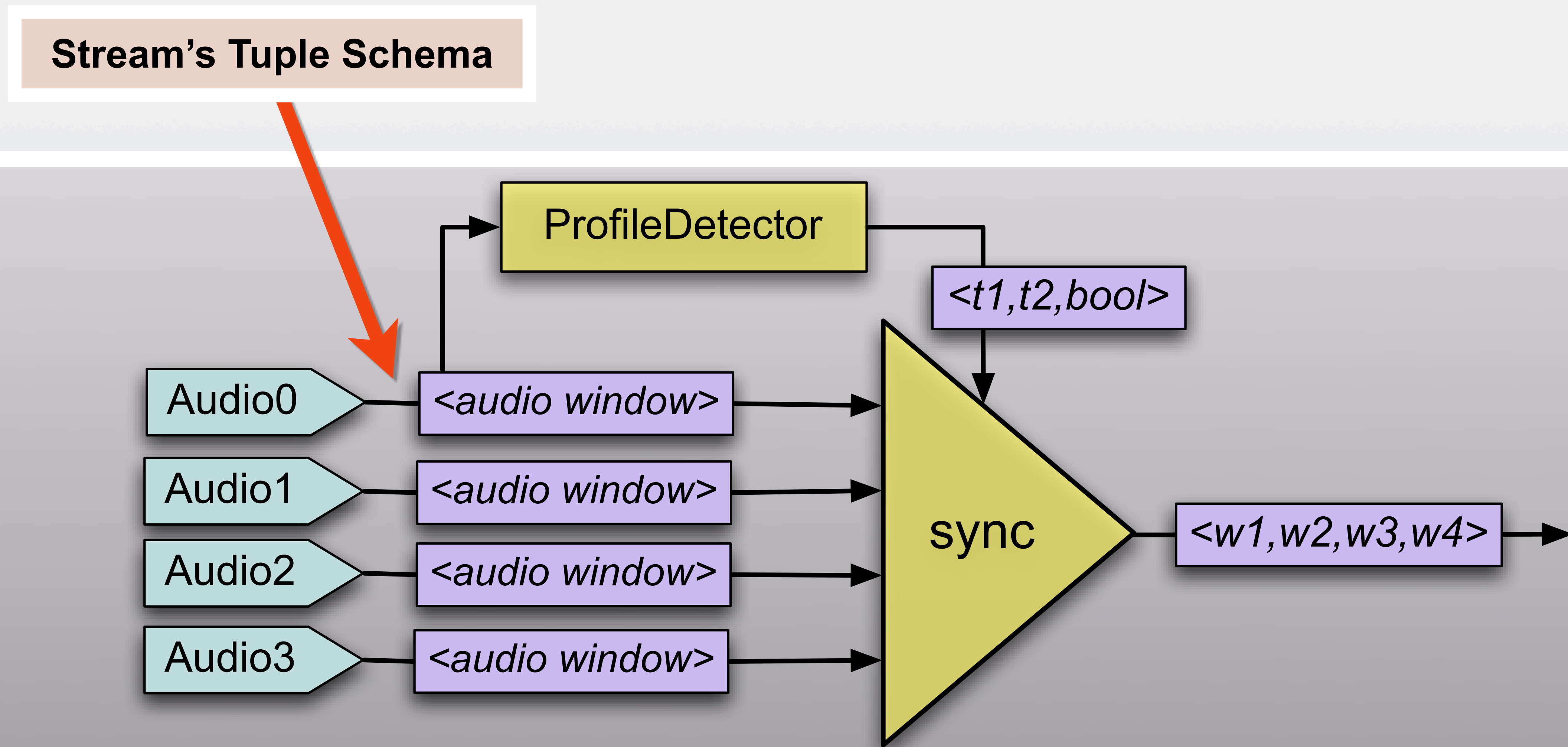
(Phase1)



http://wavescope.csail.mit.edu/

Schematic of Marmot-detector

(Phase1)



http://wavescope.csail.mit.edu/

Schematic of Marmot-detector
(Phase1)

Stream's Tuple Schema

Data Model:

- **Streams are first-class values.**
- **Streams contain Tuples**
  - One or more unnamed fields: Stream<int,float>
  - Fields may also be arrays, **Tuples**, **SigSegs**, or tagged-union datatypes, but **not Streams**.
- **SigSegs: efficiently managed windows of samples**
  - pass-by-reference
  - cheap to append, copy, forward, rewindow
  - fewer timestamps

Schematic of Marmot-detector

(Phase1)

Stream’s Tuple Schema

Data Model:

- **Streams are first-class values.**

```
Ch0 = AudioSource(...);
```

- **Streams contain Tuples**
  - One or more unnamed fields: Stream<int,float>
  - Fields may also be arrays, **Tuples**, **SigSegs**, or tagged-union datatypes, but **not Streams**.

- **SigSegs: efficiently managed windows of samples**
  - pass-by-reference
  - cheap to append, copy, forward, rewindow
  - fewer timestamps

Schematic of Marmot-detector
(Phase1)

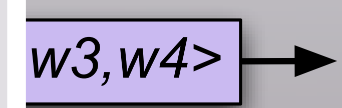
Stream's Tuple Schema

Data Model:

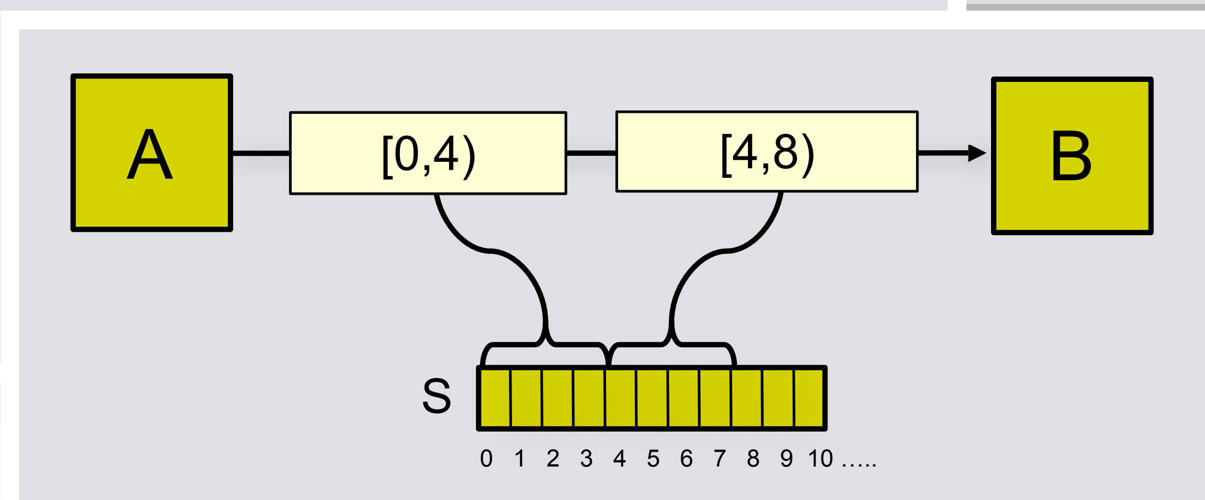
- **Streams are first-class values.**

`Ch0 = AudioSource(...);`

- **Streams contain Tuples**
  - One or more unnamed fields: Stream<int,float>
  - Fields may also be arrays, **Tuples**, **SigSegs**, or tagged-union datatypes, but **not** **Streams**.

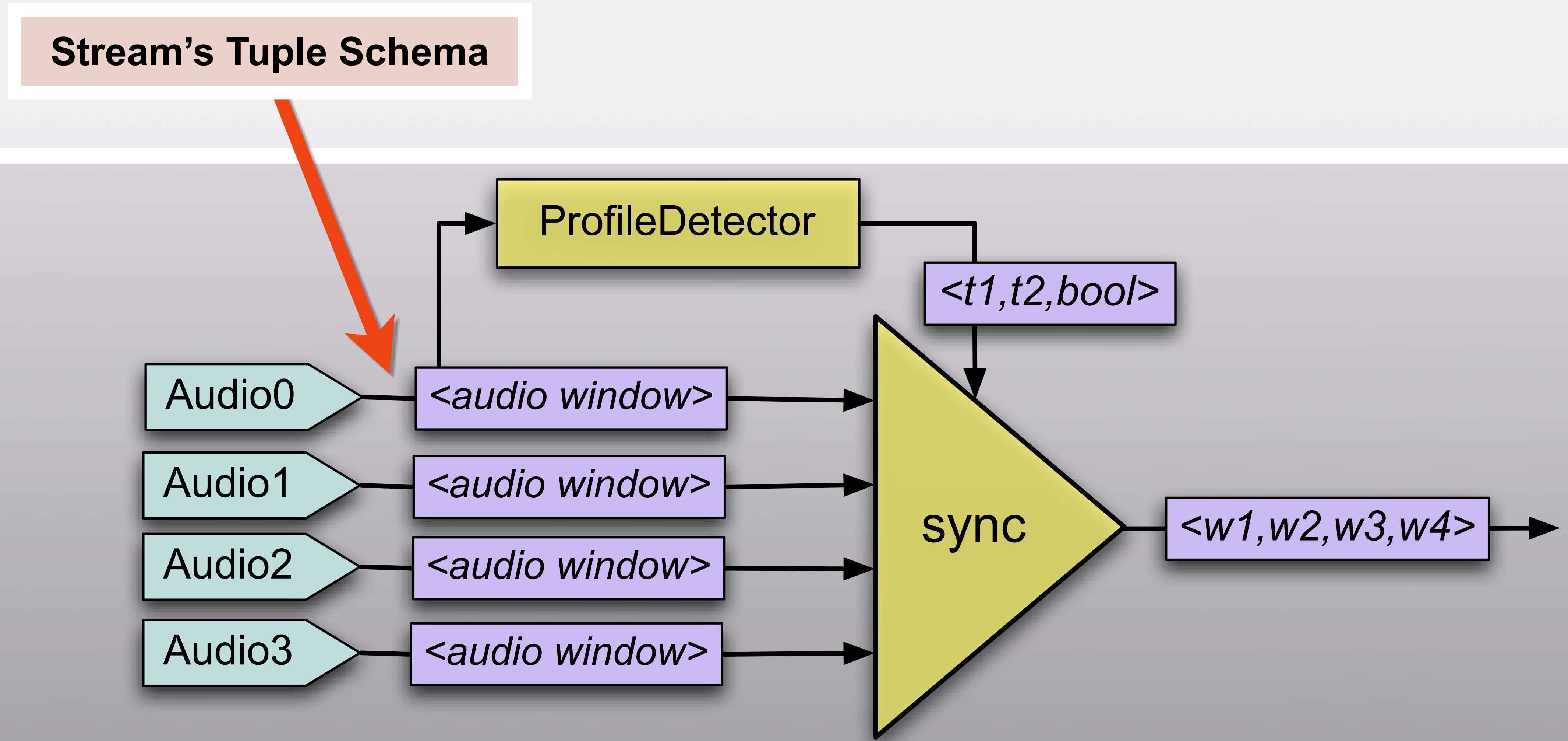


- **SigSegs: efficiently managed windows of samples**
  - pass-by-reference
  - cheap to append, copy, forward, rewindow
  - fewer timestamps



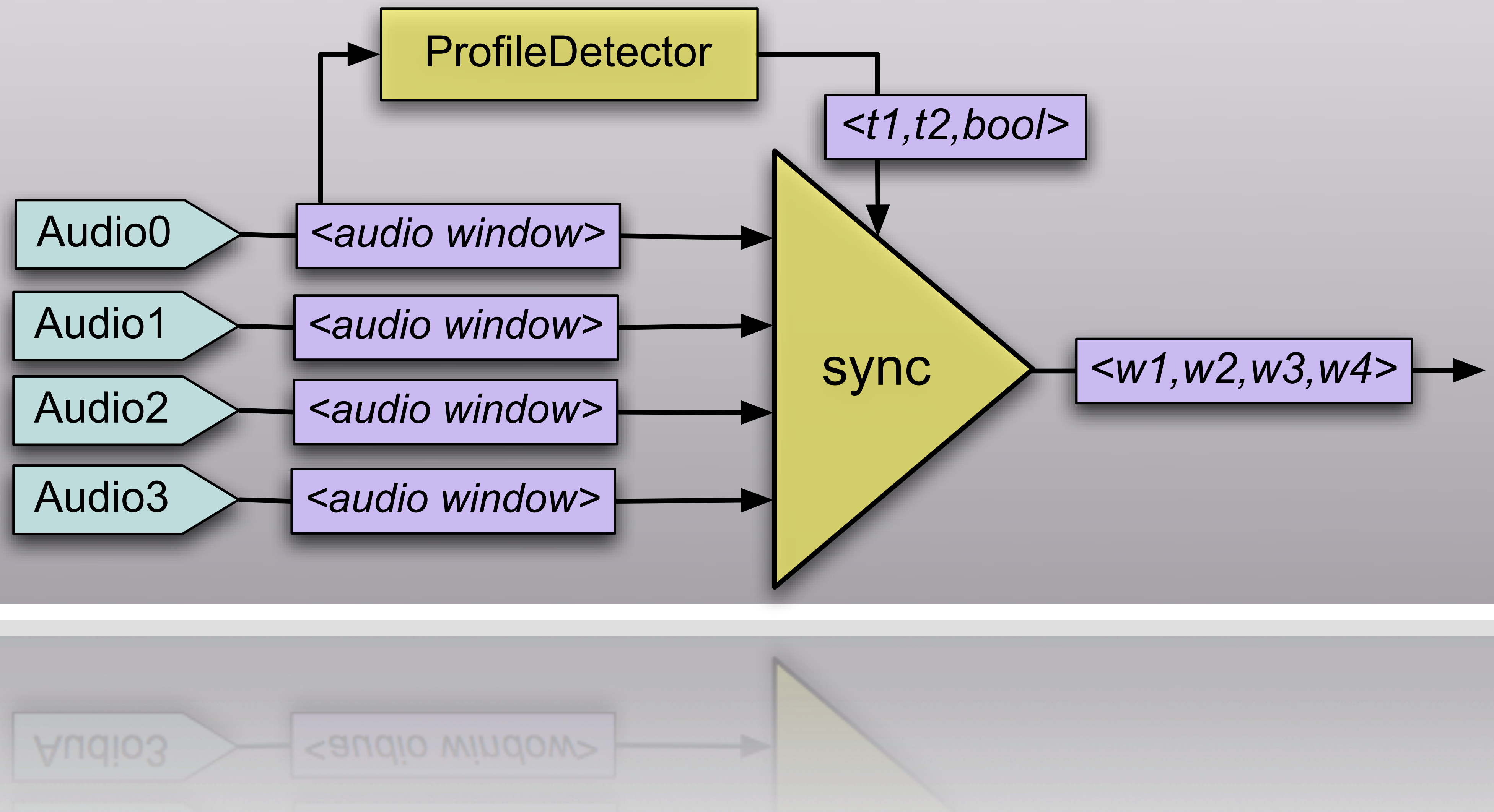
Schematic of Marmot-detector

(Phase1)



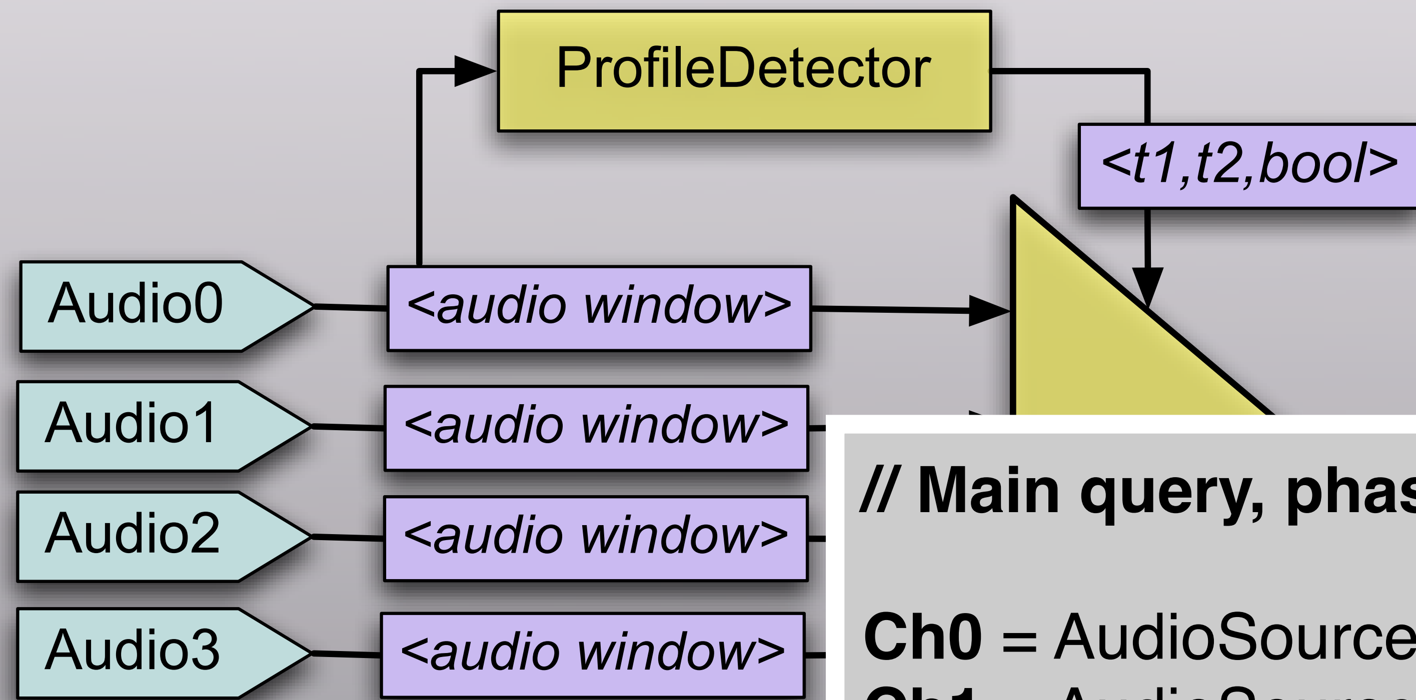
http://wavescope.csail.mit.edu/

WaveScript Code for Detector



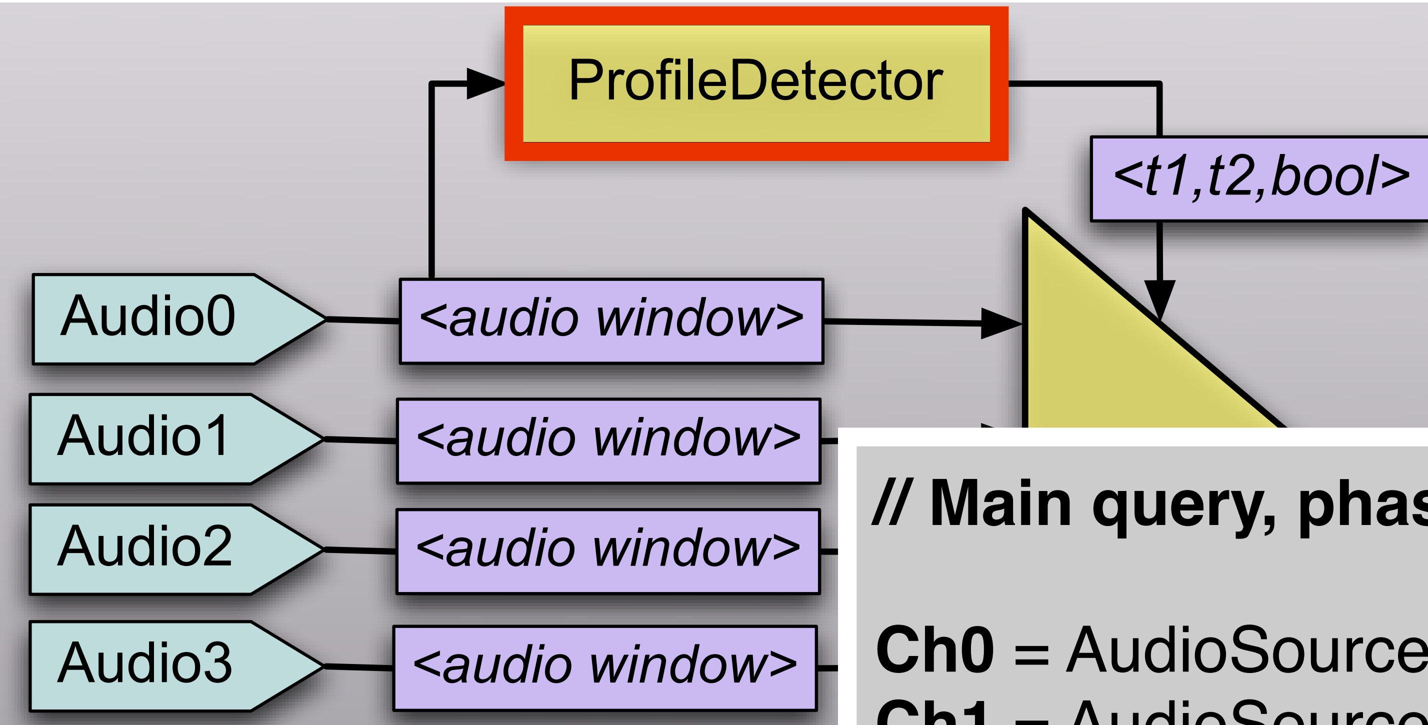
http://wavescope.csail.mit.edu/

WaveScript Code for Detector




```
// Main query, phase 1

Ch0 = AudioSource(0, 48000, 1024);
Ch1 = AudioSource(1, 48000, 1024);
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);
datawindows = sync4(control, Ch0, Ch1,
                    Ch2, Ch4);
```

WaveScript Code for Detector




```
// Main query, phase 1

Ch0 = AudioSource(0, 48000, 1024);
Ch1 = AudioSource(1, 48000, 1024);
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);
datawindows = sync4(control, Ch0, Ch1,
                                  Ch2, Ch4);
```

http://wavescope.csail.mit.edu/

WaveScript Code for Detector

```
fun profileDetect(S : Stream<SigSeg<int16>>,
                  scorefun,
                  <winsize,step>)
{
   wins = rewindow(S, winsize, step);
   scores : Stream< float >
   scores = iterate(w in wins) {
       emit scorefun( FFT(w) );
   };
   withscores : Stream<float, SigSeg<int16>>
   withscores = zip2(scores, wins);
   return threshFilter(withscores);
}
```

```
ase 1

e(0, 48000, 1024);
e(1, 48000, 1024);
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);
datawindows = sync4(control, Ch0, Ch1,
                    Ch2, Ch4);
```

WaveScript Code for Detector

```
fun profileDetect(S : Stream<SigSeg<int16>>,
                      scorefun,
                      <winsize,step>)
{
   wins = rewindow(S, winsize, step);
   scores : Stream< float >
   scores = iterate(w in wins) {
        emit scorefun( FFT(w) );
   };
   withscores : Stream<float, SigSeg<int16>>
   withscores = zip2(scores, wins);
   return threshFilter(withscores);
}
```

ProfileDetector

profileDetect(Ch0, ...)

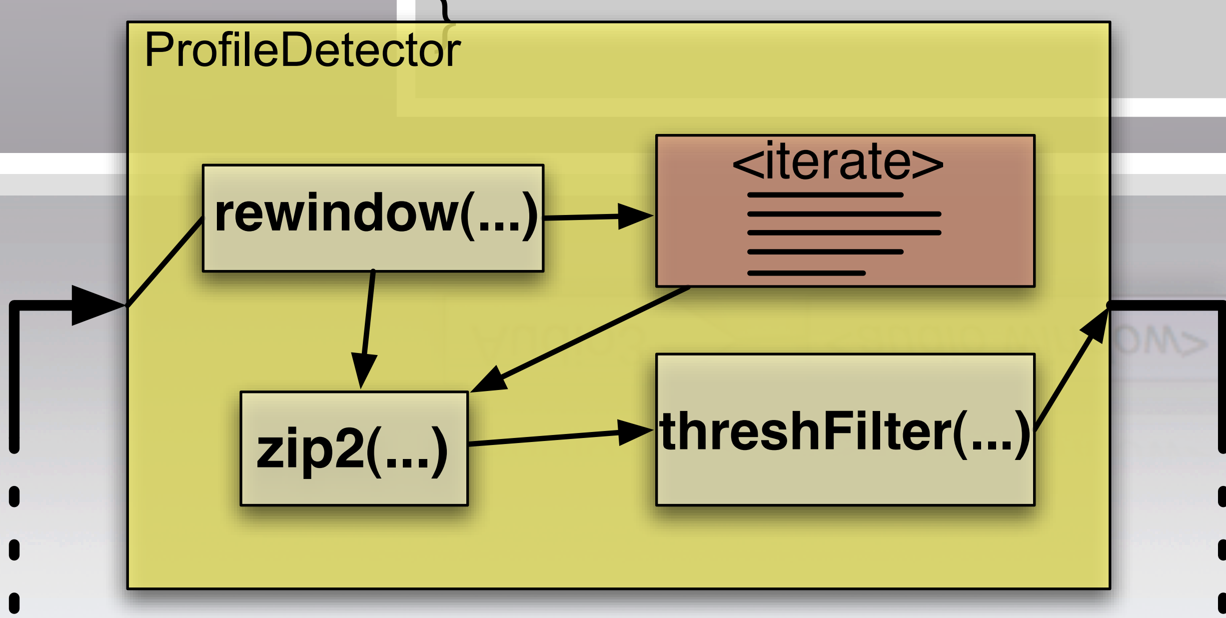
```
ase 1

e(0, 48000, 1024);
e(1, 48000, 1024);
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);

datawindows = sync4(control, Ch0, Ch1,
                    Ch2, Ch4);
```

http://wavescope.csail.mit.edu/

WaveScript Code for Detector

```
fun profileDetect(S : Stream<SigSeg<int16>>,
                  scorefun,
                  <winsize,step>)
{
   wins = rewindow(S, winsize, step);
   scores : Stream< float >
   scores = iterate(w in wins) {
        emit scorefun( FFT(w) );
   };
   withscores : Stream<float, SigSeg<int16>>
   withscores = zip2(scores, wins);
   return threshFilter(withscores);
}
```

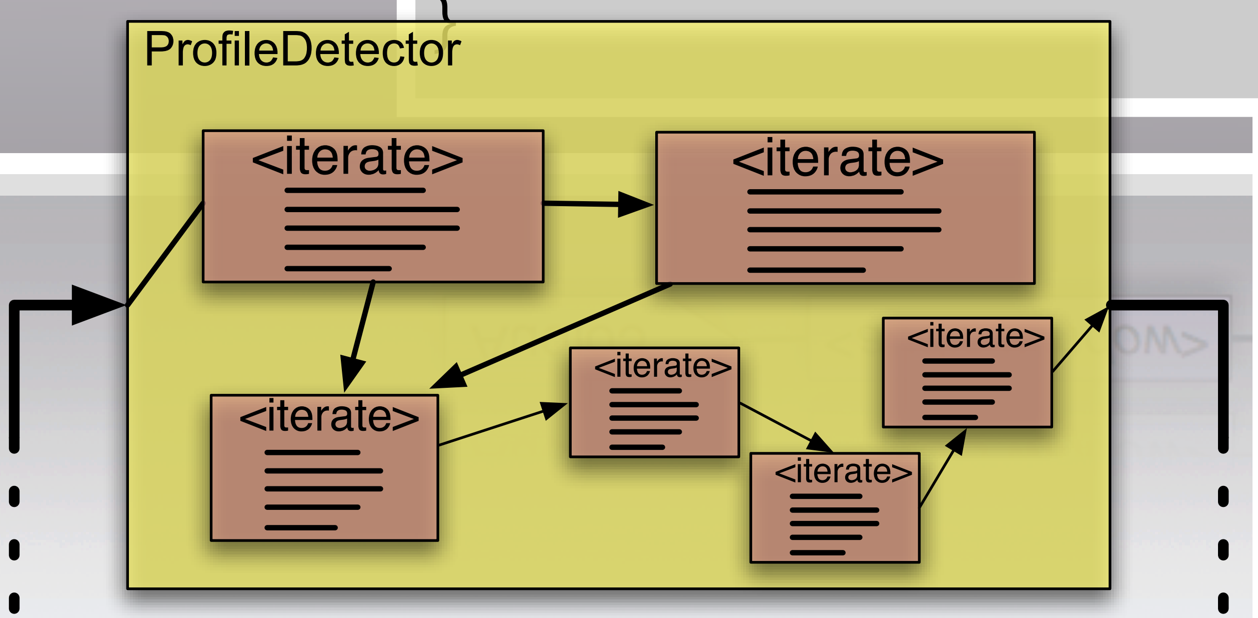
```
ase 1

e(0, 48000, 1024);
e(1, 48000, 1024);
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);
datawindows = sync4(control, Ch0, Ch1,
                             Ch2, Ch4);
```



WaveScript Code for Detector

```
fun profileDetect(S : Stream<SigSeg<int16>>,
                  scorefun,
                  <winsize,step>)
{
   wins = rewindow(S, winsize, step);
   scores : Stream< float >
   scores = iterate(w in wins) {
        emit scorefun( FFT(w) );
   };
   withscores : Stream<float, SigSeg<int16>>
   withscores = zip2(scores, wins);
   return threshFilter(withscores);
}
```

```
...ase 1

...e(0, 48000, 1024);
...e(1, 48000, 1024);
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);

datawindows = sync4(control, Ch0, Ch1,
                    Ch2, Ch4);
```

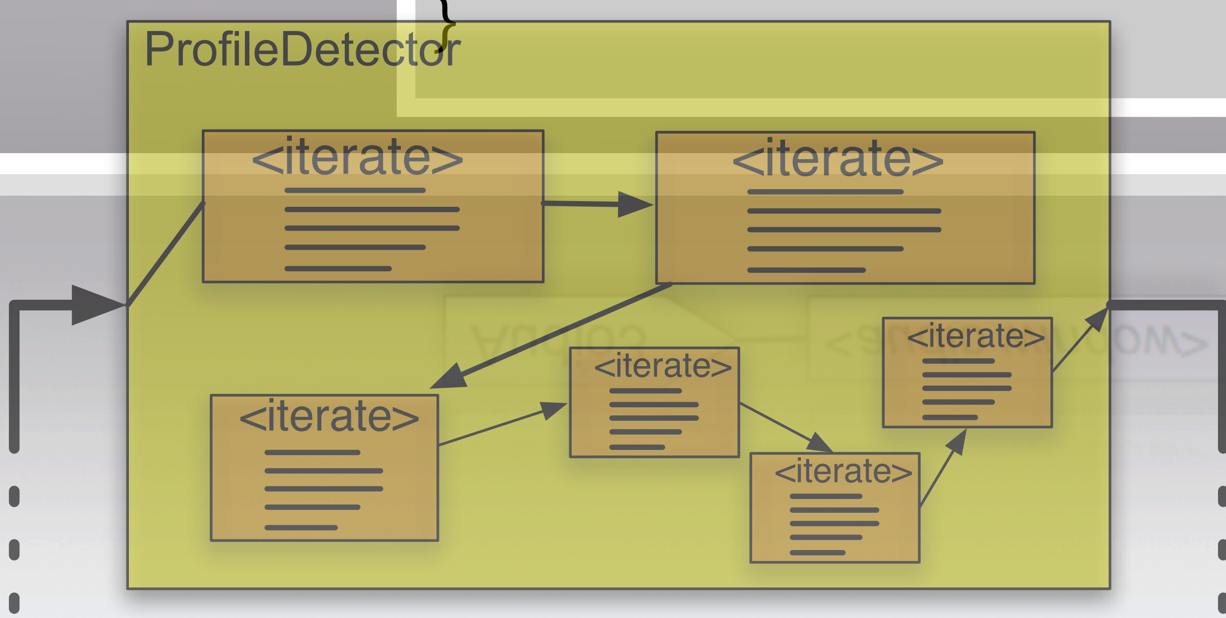


http://wavescope.csail.mit.edu/

WaveScript Code for Detector

```
fun profileDetect(S : Stream<SigSeg<int16>>,
                      scorefun,
                      <winsize,step>)
{
   wins = rewindow(S, winsize, step);
   scores : Stream< float >
   scores = iterate(w in wins) {
        emit scorefun( FFT(w) );
   };
   withscores : Stream<float, SigSeg<int16>>
   withscores = zip2(scores, wins);
   return threshFilter(withscores);
}
```

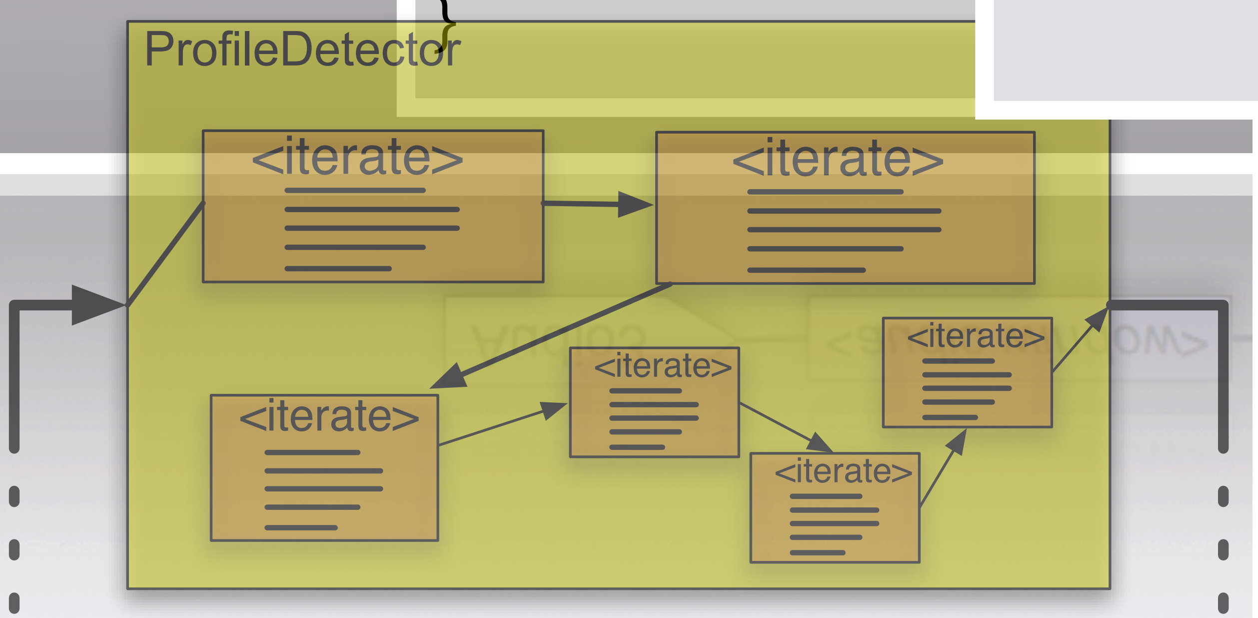
```
ase 1

e(0, 48000, 1024);
e(1, 48000, 1024);
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                              <64,192>);

datawindows = sync4(control, Ch0, Ch1,
                                  Ch2, Ch4);
```



WaveScript Code for Detector

```
fun profileDetect(S
                   sc
                   <v
{
    wins = rewindow
    scores : Stream
    scores = iterate
         emit scorefu
    };
    withscores : Stre
    withscores = zip
    return threshFilt
}
```

Integrated Language




```
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);

datawindows = sync4(control, Ch0, Ch1,
                    Ch2, Ch4);
```

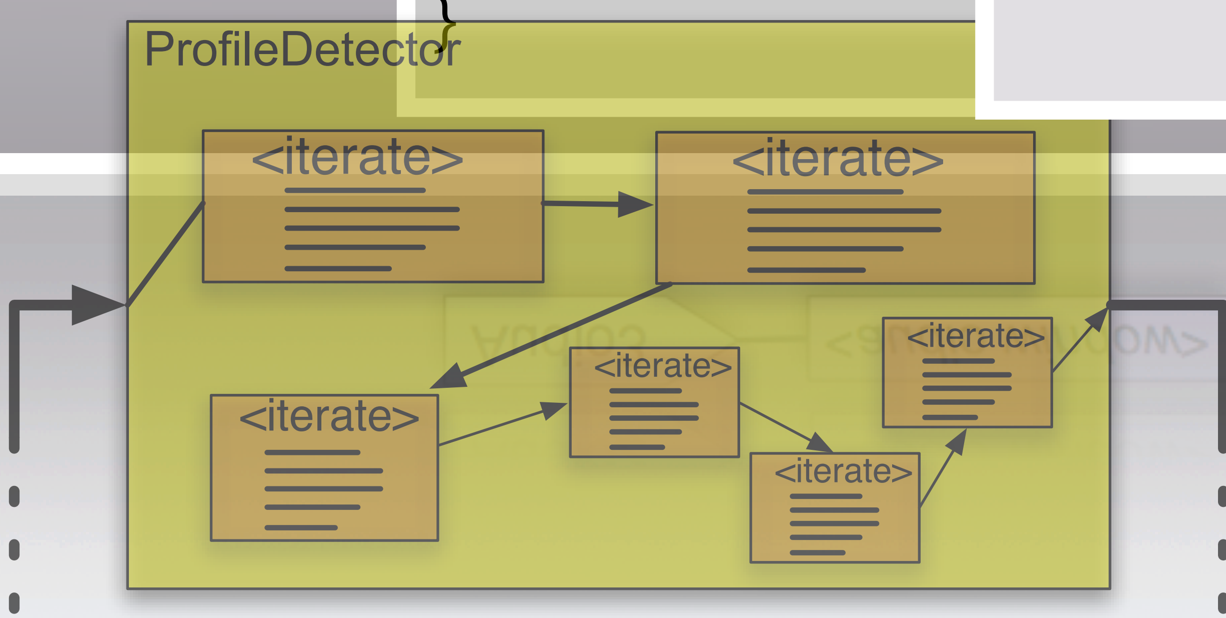
http://wavescope.csail.mit.edu/

WaveScript Code for Detector

```
fun profileDetect(S
                  sc
                  <\
{
  wins = rewindow
  scores : Stream
  scores = iterate
     emit scorefu
  };
  withscores : Stre
  withscores = zip
  return threshFilt
}
```

Integrated Language

- High-level, query-like declarative programs
  - map, project, filter streams
  - apply library signal-processing ops




```
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);
datawindows = sync4(control, Ch0, Ch1,
                    Ch2, Ch4);
```

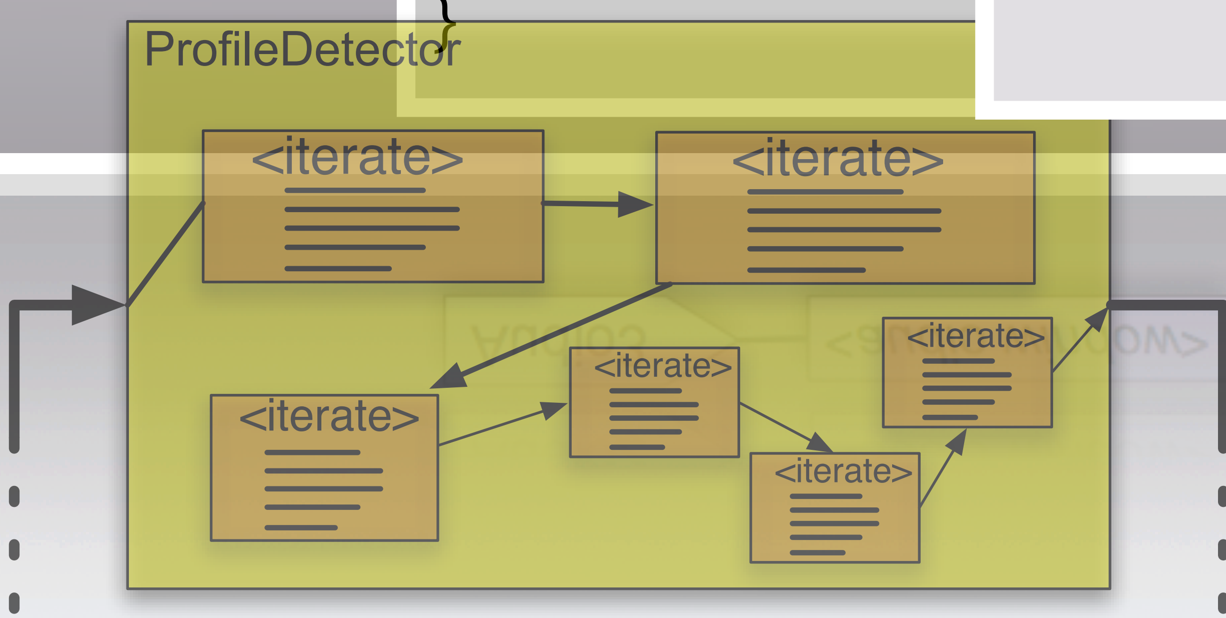
http://wavescope.csail.mit.edu/

WaveScript Code for Detector

```
fun profileDetect(S
                  sc
                  <\
{
   wins = rewindow
   scores : Stream
   scores = iterate
      emit scorefu
   };
   withscores : Stre
   withscores = zip
   return threshFilt
}
```

Integrated Language

- High-level, query-like declarative programs
  - map, project, filter streams
  - apply library signal-processing ops
- Low-level, imperative code within custom-operators
  - use iterate to introduce

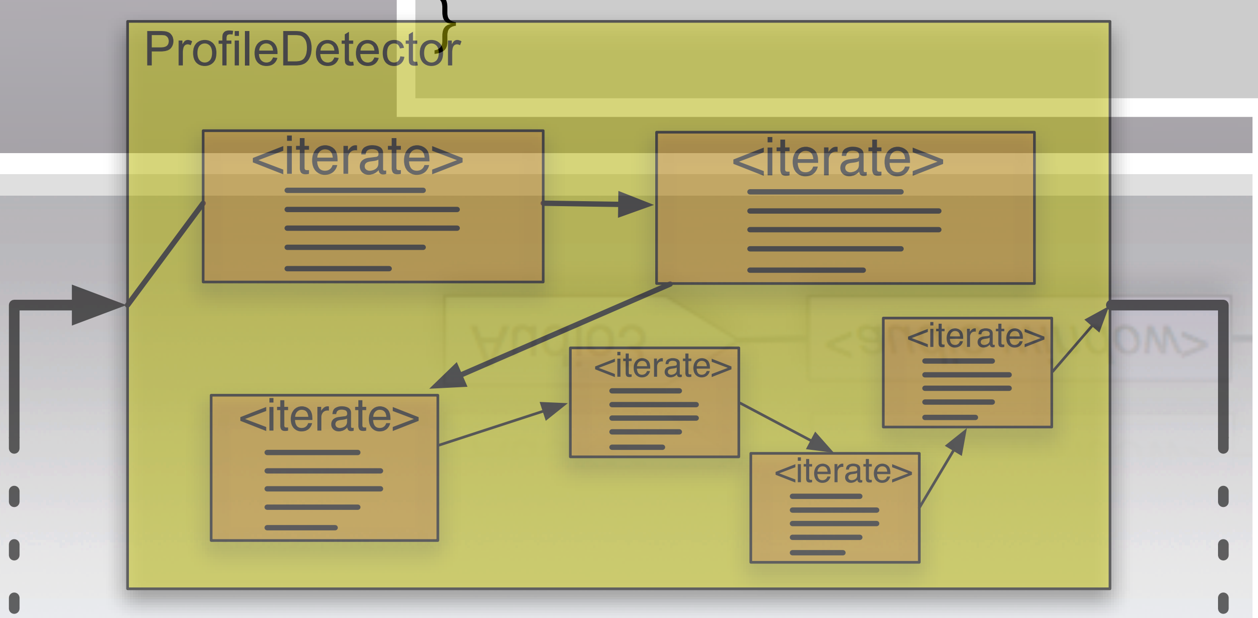



```
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);
datawindows = sync4(control, Ch0, Ch1,
                    Ch2, Ch4);
```

http://wavescope.csail.mit.edu/

WaveScript Code for Detector

```
fun profileDetect(S : Stream<SigSeg<int16>>,
                  scorefun,
                  <winsize,step>)
{
   wins = rewindow(S, winsize, step);
   scores : Stream< float >
   scores = iterate(w in wins) {
        emit scorefun( FFT(w) );
   };
   withscores : Stream<float, SigSeg<int16>>
   withscores = zip2(scores, wins);
   return threshFilter(withscores);
}
```




```
ase 1

e(0, 48000, 1024);
e(1, 48000, 1024);
Ch2 = AudioSource(2, 48000, 1024);
Ch3 = AudioSource(3, 48000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);

datawindows = sync4(control, Ch0, Ch1,
                    Ch2, Ch4);
```

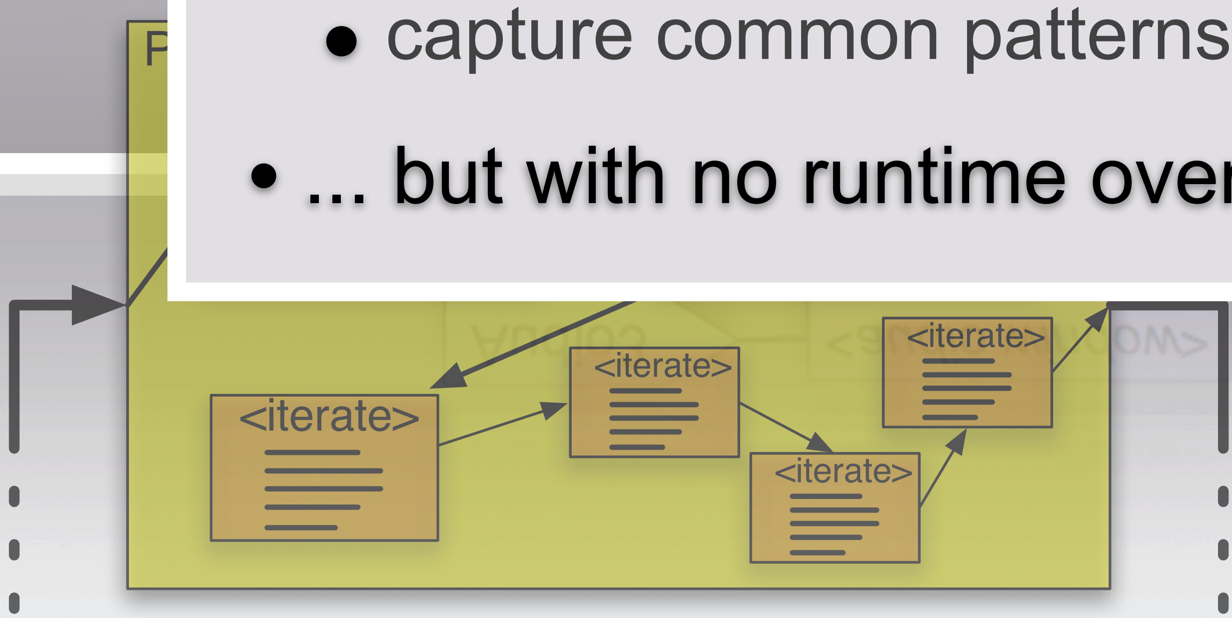
http://wavescope.csail.mit.edu/

WaveScript Code for Detector

```
fun profileDetect(S : Stream<SigSeg<int16>>,
                     scorefun,
                     <winsize,step>)
{
```

Abstraction, Abstraction, Abstraction

- Functions that build query network
- Generic library routines
  - int16, int32, float, etc...
  - take a *list* of input streams
  - capture common patterns; e.g. split-join
- ... but with no runtime overhead




```
3000, 1024);
3000, 1024);
3000, 1024);
3000, 1024);

control = profileDetect(Ch0, marmotScore,
                        <64,192>);

datawindows = sync4(control, Ch0, Ch1,
                    Ch2, Ch4);
```

Conclusions

- High-rate event stream + signal processing important in a number of emerging apps
- WaveScope:
  Language, compiler, distributed + embedded runtime
- Key ideas:
  - Sigseg data model: efficient windowing construct
  - Integrated language, custom operators, abstraction through query generation
- Status: Prototype nearly complete; preliminary measurements show 1-7 million samples/s on real apps

http://wavescope.csail.mit.edu/

Optimization

- Algebraic
  - General: operator merging, predicate reordering, etc
  - Domain specific: FFT(IFFT(...))
- Compiler
  - Whole program optimization (a la HPC)
  - The usual
    - (But the C compiler can handle some of these.)

Programmatically Generated StreamSQL

Last week I was talking with the head of software development at one of our customers about StreamSQL. He had been encouraged to look at the language by colleague at IBM research. I spent some time describing the model and the details of our implementation. All this was of course to his liking. But he had one major question: can I write programs that generate StreamSQL? Of course, I was pleased to respond. That is one of the main reasons we developed StreamSQL, and some customers are already using it that way.

I look forward to many tools using generated StreamSQL, just as many tools use generated SQL. StreamSQL is suitable for automated generation for many of the same reasons as traditional SQL:

- **Textual representation.** Firstly, SQL and StreamSQL have a basic textual representation, which can be used to describe applications in their entirety. Data definition and manipulation are both part of the same language. StreamSQL can be generated without using XML or other data format manipulation libraries. Queries can be embedded directly in most languages using their native string manipulation.
- **Declarative.** SQL and StreamSQL are declarative languages. They express the desired results, rather than the details of the computation. Concerns about algorithms, efficiency, and shared computation are offloaded to the implementation. The benefit to code generation is substantial. Generated SQL and StreamSQL can be quite succinct and powerful. With a non-declarative language, code generation quickly becomes tied into performance optimization, greatly increasing complexity.
- **Standardized.** Standardization is responsible for much of the success of SQL generation, and the potential success of StreamSQL generation. Standardization means that a code generating application, library or toolkit can be utilized with multiple implementations. As a result, development is more economical.