Handling Edge Effects on Segment Boundaries

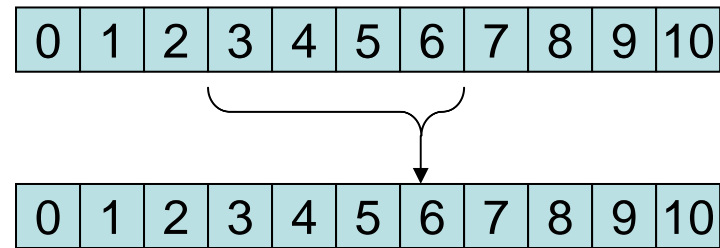
Kerry Barnes

March 16, 2005

Sliding Window

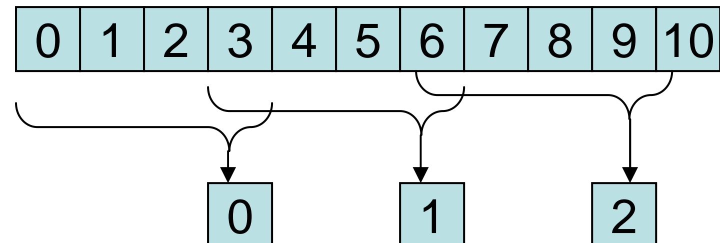
- Sliding Window
  - Output token value depends on N adjacent input token values
  - Examples:
    - FIR Filter
    - Overlapped Vector Processing
    - Image Column Processing

FIR Filter out[i] = sum(C[j]*in[i-j])



Overlap Vector Processing

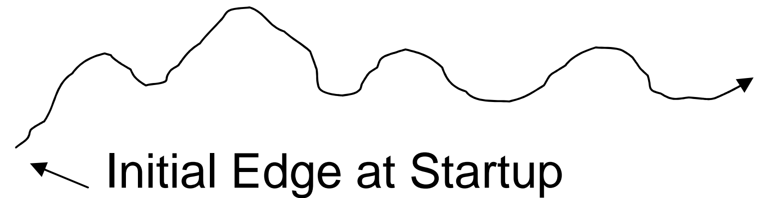
Out[i] = func(in[D*i]..in[D*i-N])



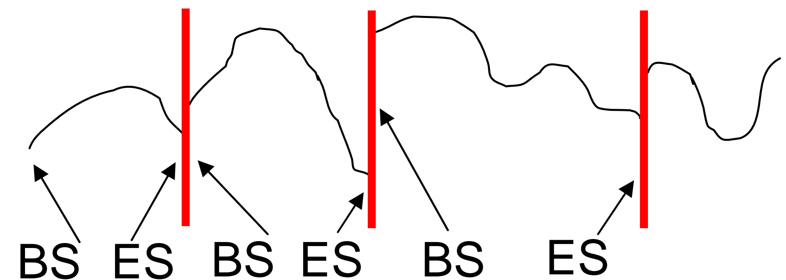
Edge Effects

- Unsegmented streams
  - Have only one edge
  - Initial transient
  - Usually not important
- Segmented streams
  - Each segment has two edges (BS, ES)
  - End of one segment unrelated to begin of next
  - Edges are significant part of data
  - How edges are handled significantly effects results

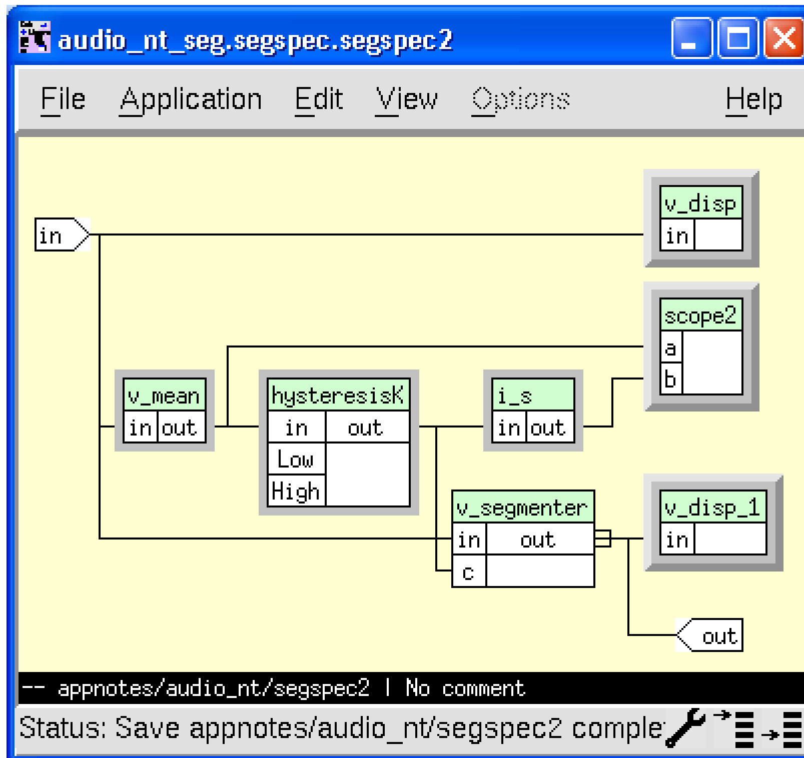
Unsegmented Stream



Segmented Stream



Example: Speech Segmentation



- Nine speech segments detected in displayed input stream
- Segments form continuous stream separated by segment markers

Why Worry about the Edges?

- Important information may be on edge
- Transients from edge may effect subsequent processing
- May need to exactly reproduce results from heritage application

Techniques for Handling Edges

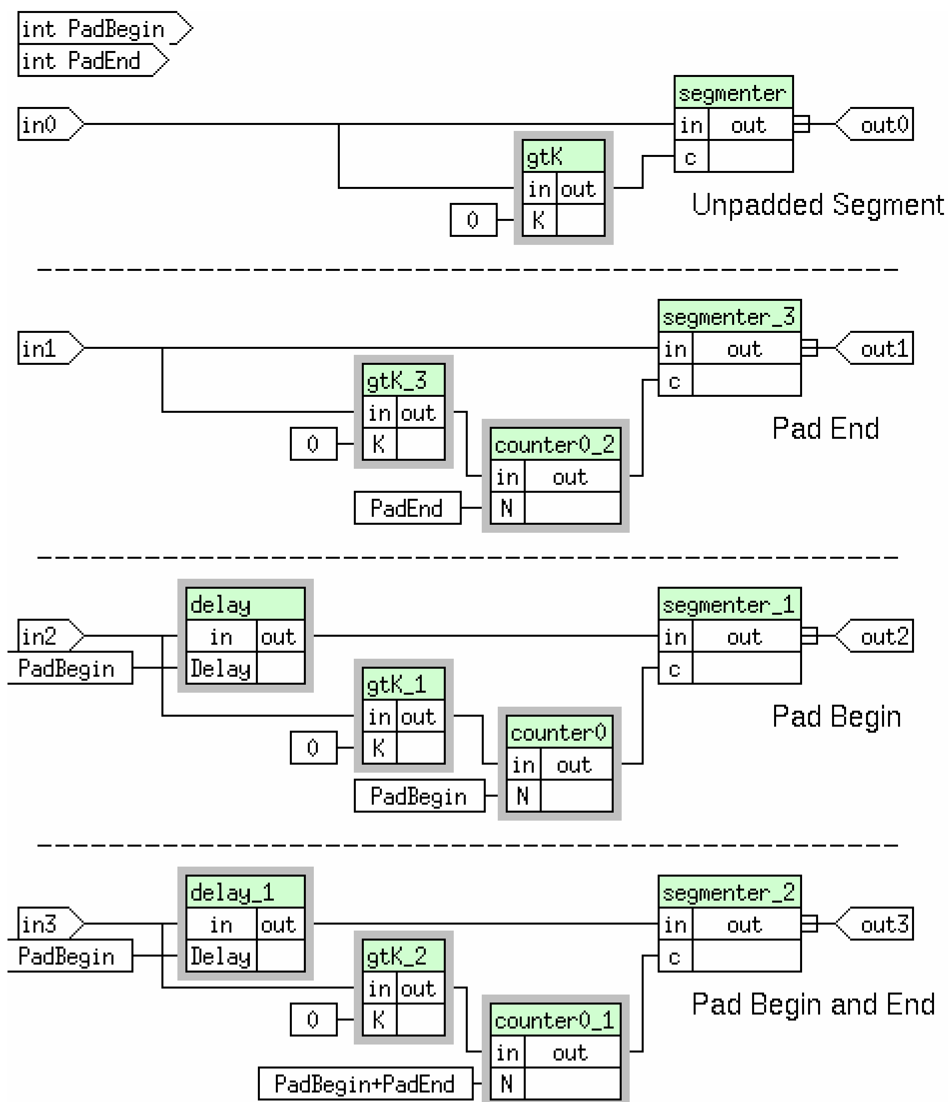
- Control how segments are produced
  - Not possible if segmentation buried in incoming data stream
  - Possible if segments are detected out of existing continuous data stream
- Control how segment edges are processed
  - Drop edge transients
  - Pad edges
  - Mirror
  - Change processing on edges

Controlling Segment Generation

- Segment dictated by external events
  - Example: Image Scan Line, Scanning Sensors
  - No control over where segments begin and end
- Segment detected out of continuous stream
  - Examples: Speech Energy, Radar Pulse, Region of Interest
  - Algorithm can choose where segments begin and end
  - Control over the begin and end of segment gives developer more options on edge handling

Segment Detected out of Continuous Stream

- Detector drives control input of segmenter box
- We may want to add tokens to the segment at either or both segment boundaries
- Example
  - gtK box outputs 1 if input is >0 is basic detector in this example
  - delay primitive delays signal Delay tokens
  - counter0 primitive adds N ones to end of pulse

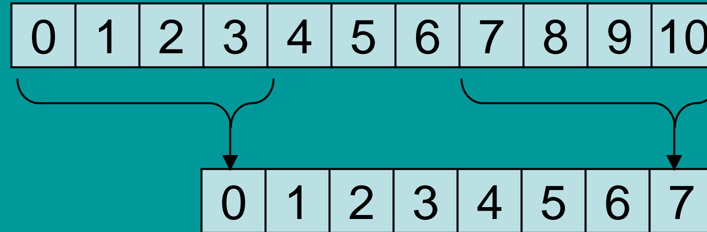


Padding Begin and End

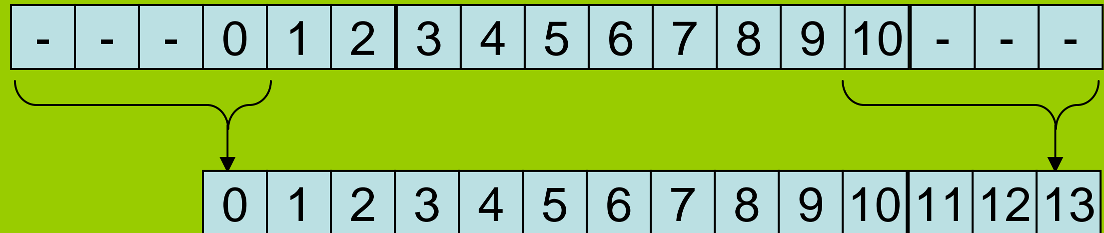
- Detection extended by PadBegin+PadEnd
- Signal delayed to segmenter by PadBegin tokens

Control How Segment Edges are Processed

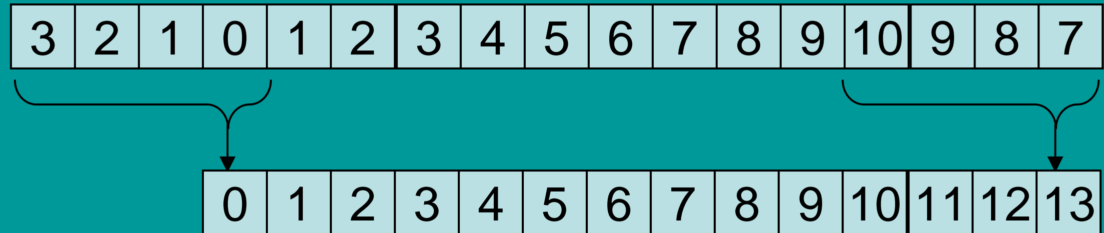
Drop (No Pad)



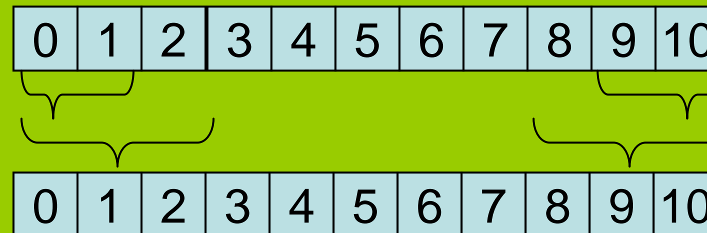
Pad



Mirror



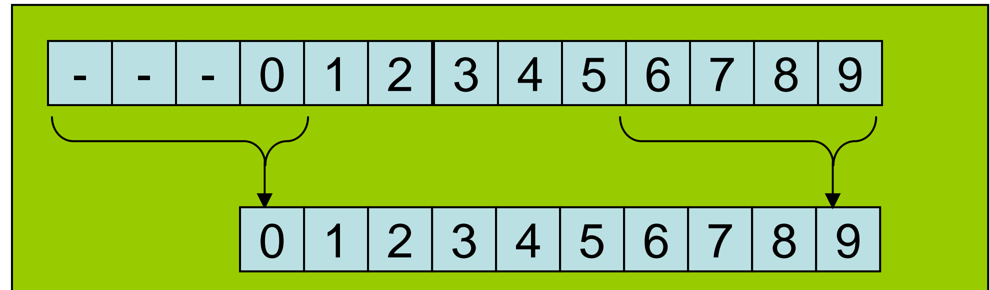
Change Window Size



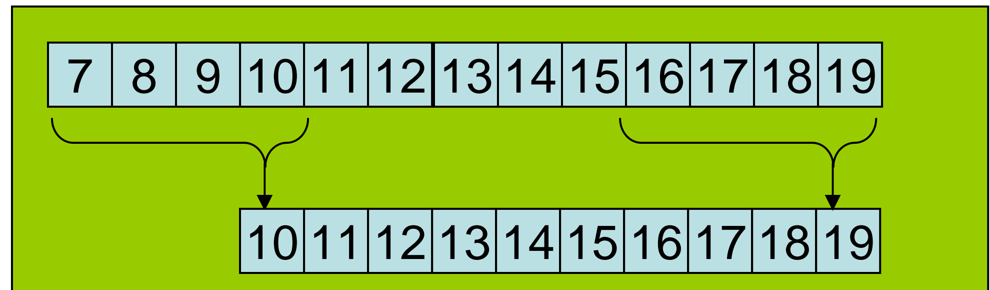
Method 1: Pad Begin, Drop End

- Example:
  - Primitive granularity of 10
  - Segment length of 26
  - Window size of 4
- N-1 = 3 tokens inserted in stream before first execution
- Intermediate execution shows steady state condition
- Final execution has primitive flexed to granularity of 6
- Static dataflow
  - Each firing produces one new output token for every new input token

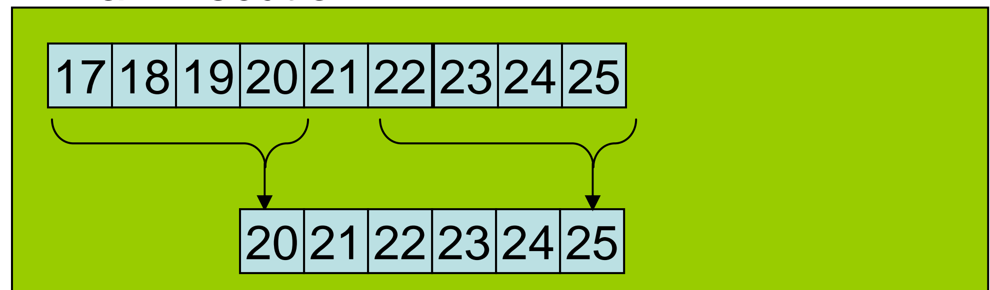
First Execution



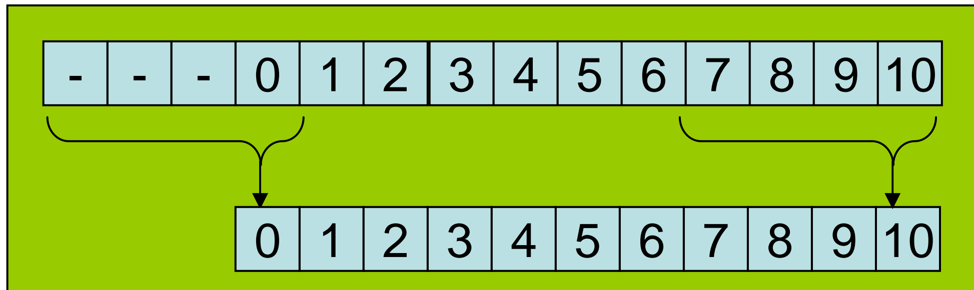
Intermediate Execution



Final Execution



Pad Begin, Drop End



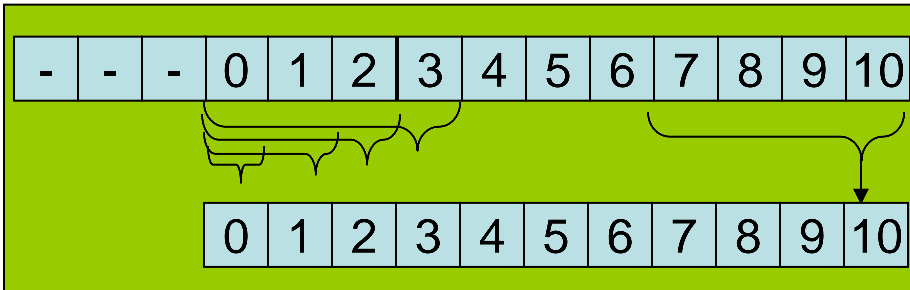
- Simplest Method
- Dataflow at beginning and end looks like middle processing
- Static Dataflow
- Directly supported by primitive input overlap parameter
- Method used by Gedae primitive library
- First choice if algorithmically acceptable

```
Name: sliding_ave
Type: static

Input: {
  stream float in(1,N-1);
  int N;
}
Output: {
  stream float out;
}

Apply: {
  int g,j;
  for(g=0; g<granularity; g++) {
    out[g] = 0;
    for (j=0; j<N; j++) {
      out[g] += in[g+j]/N;
    }
  }
}
```

Change Window Begin, Drop End



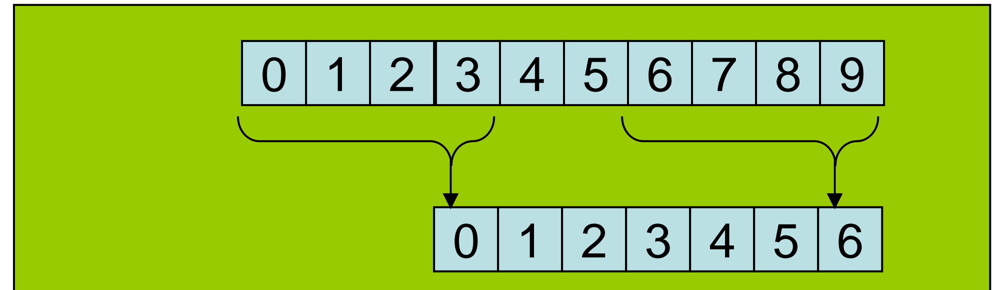
- Initial padding unused
- Has same static dataflow properties as Pad Begin/Drop End
- Same number of tokens go in as go out
- Initial values more reflective of data
- Begin and end segment processing is not symmetric

```
Name: sliding_ave2
Type: static
Input:  {stream float in(1,N-1);}
Local:  {int cnt;}
Output: {stream float out;}
Reset: {
  cnt = 1;
}
Apply: {
  int g,j;
  for(g=0; g<granularity; g++) {
    out[g] = 0;
    for (j=N-cnt; j<N; j++) {
      out[g] += in[g+j]/cnt;
    }
    if (cnt < N) cnt++;
  }
}
```

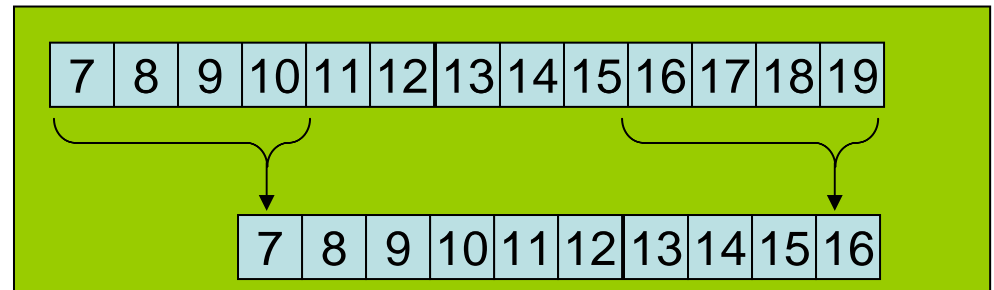
Method 2: Drop Begin, Drop End

- Example:
  - Primitive granularity of 10
  - Segment length of 26
  - Window size of 4
- No tokens inserted before first execution.
- Second execution shows steady state condition
- Final execution has primitive flexed to granularity of 6
- Dynamic dataflow
  - First exectution produces fewer tokens than are input
  - Subsequent exectutions produce one new output token for every new input token

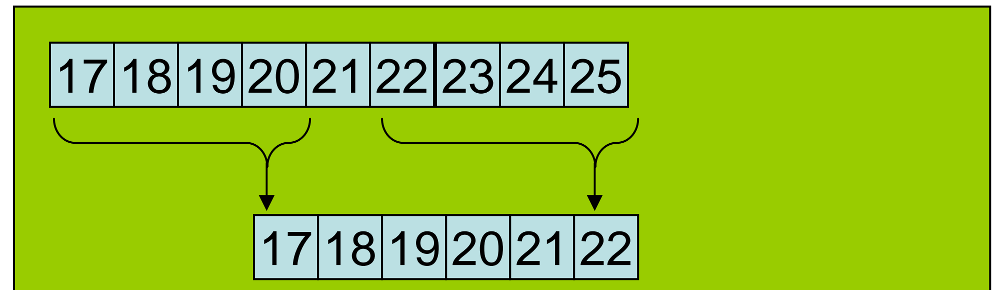
First Execution



Second Execution

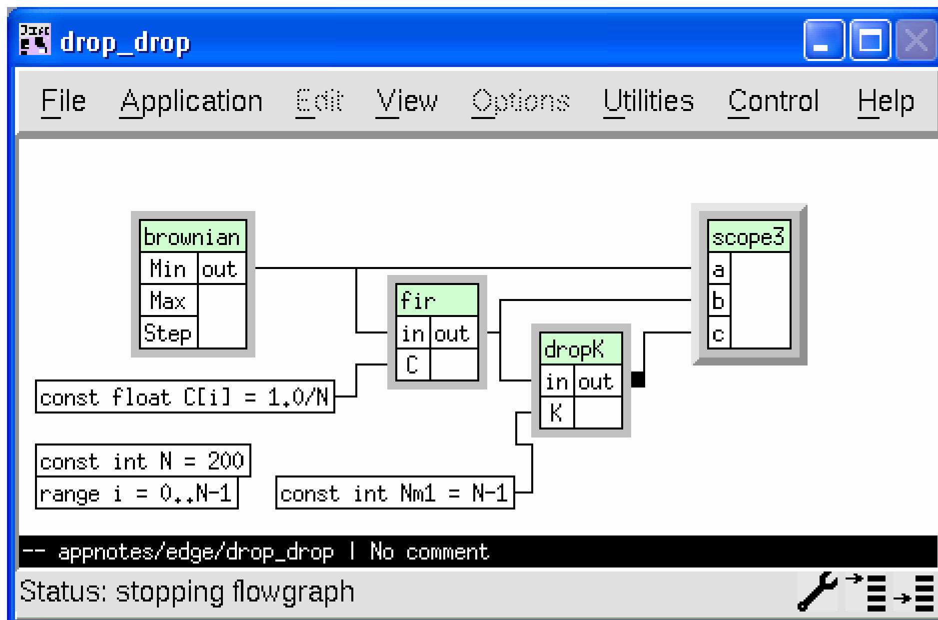


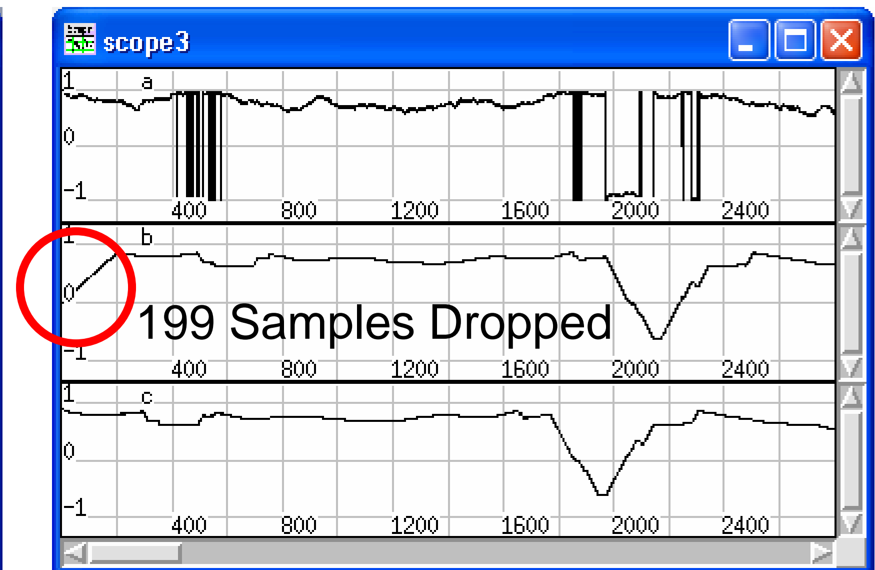
Final Execution



Method 2: Drop Begin, Drop End

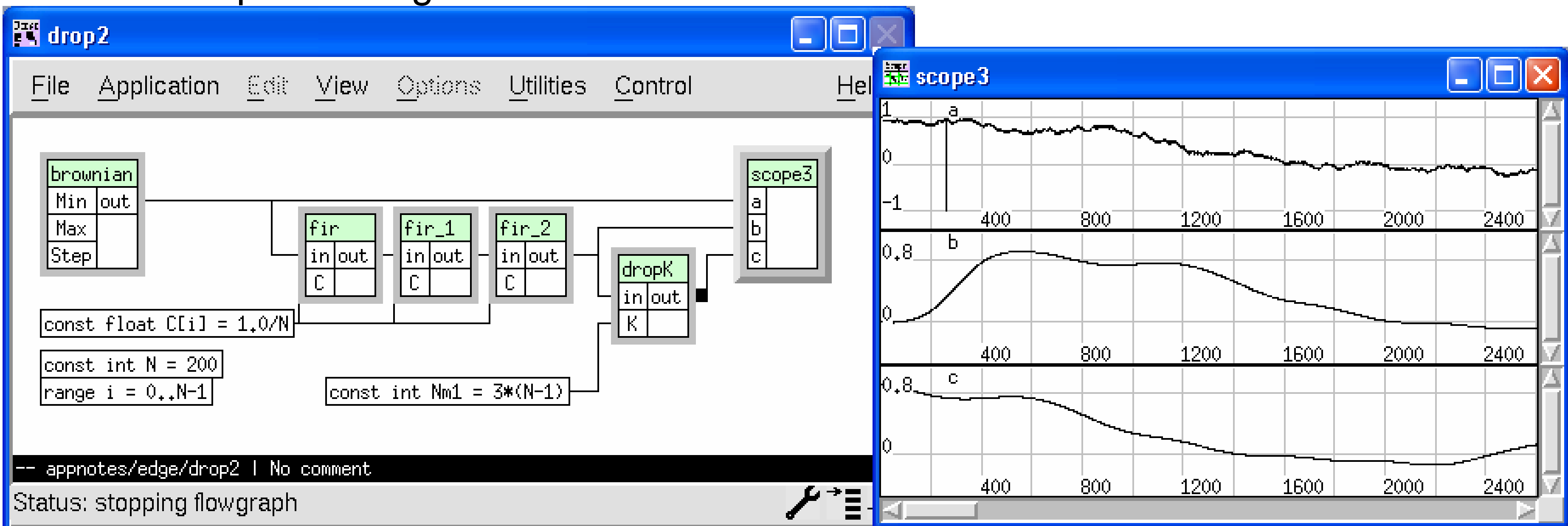
- More difficult than Method 1
- Requires a dynamic dataflow output
- Implement by modifying Method1 primitive or adding a “drop” box after the primitive





Method 2: Drop Begin, Drop End

- To drop data for a chain of sliding window boxes requires only one drop box at the end
- Requires that all the dropped tokens be processed and then thrown away (inefficient)
- But creates just one dynamic boundary (efficient)
- For segments of sufficient length reducing dynamic boundaries justifies the extra processing



Method 2: Drop Begin, Drop End

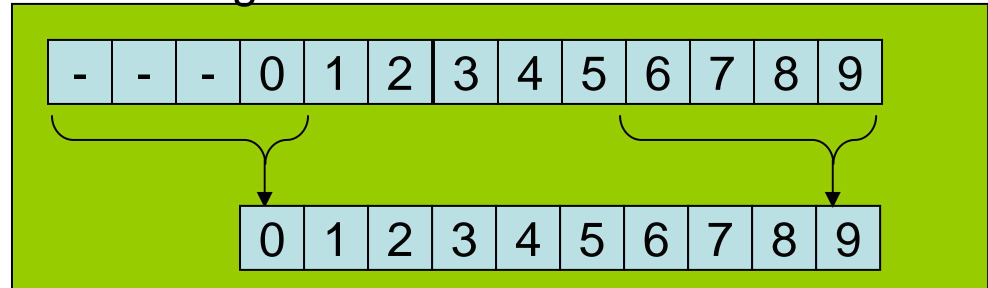
```
Name: dropK
Type: static
Input: {
  stream float in;
  int K;
}
Local: {
  int cnt;
}
Output: {
  dynamic stream float out;
}
Reset: {
  cnt = K;
}
```

```
Apply: {
  int G = granularity;
  if (cnt) {
    G  -= cnt;
    if (G < 0) {
      cnt -= granularity;
    } else {
      in += cnt;
      cnt = 0;
    }
  }
  if (cnt == 0) {
    e_vmov(in,1,out,1,G);
    produce(out,G);
  }
}
```

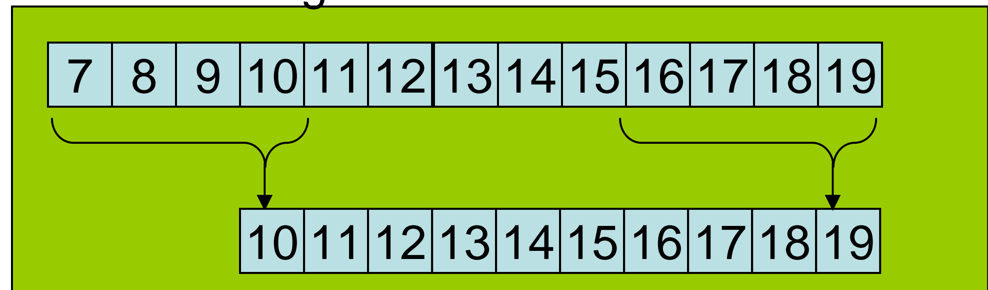
Method 3: Pad Begin, Pad End

- Processing with max primitive granularity of 10 and segment length of 26 tokens
- First firing processes first 10 segment tokens padding initial data with 0's, produces 10 tokens
- Second firing processes 13 tokens – 3 overlapped from first firing and the next 10 tokens in segment, produces 10 tokens
- Final firing processes 9 tokens – 3 overlapped from second firing and the final 6 tokens in the segment, produces 9 tokens.

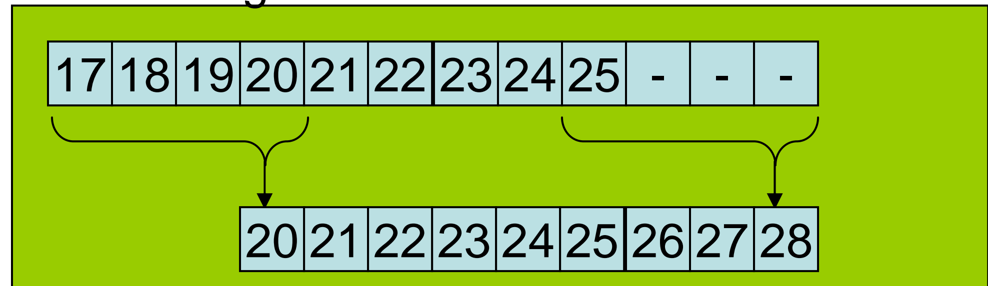
First Firing



Second Firing



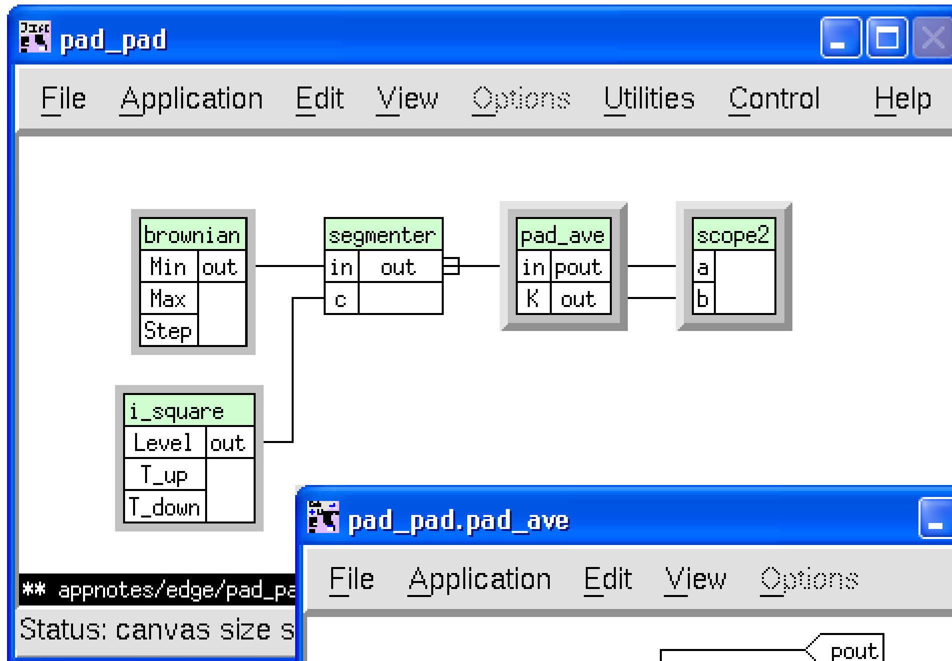
Final Firing

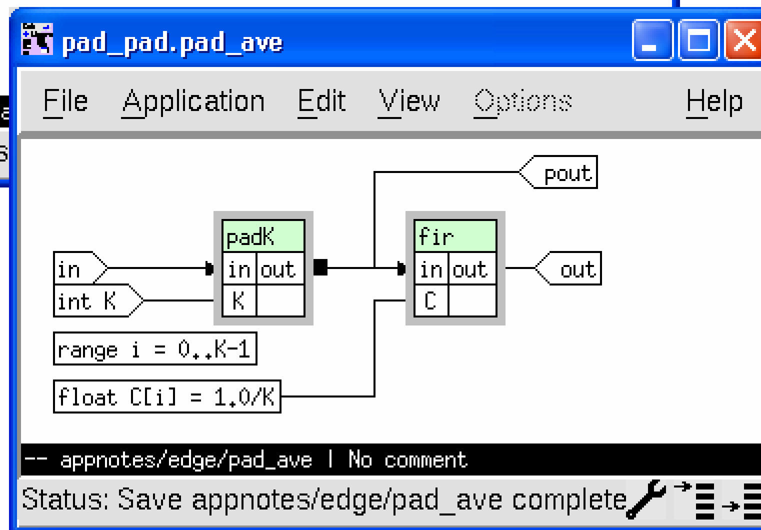


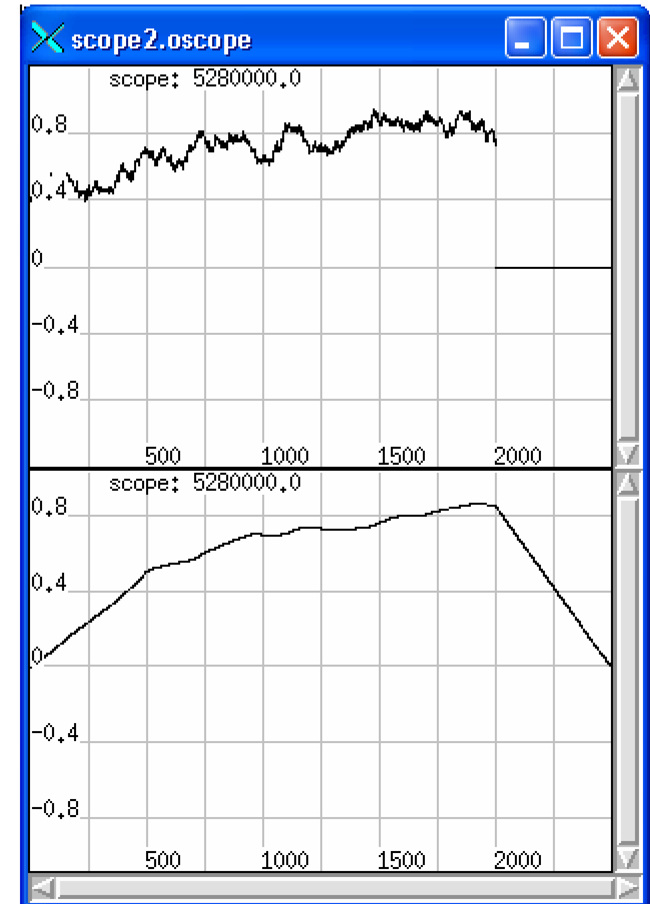
Pad Begin, Pad End

- Requires adding tokens to output data stream at the end-of-segment
- Primitive End-of-segment method provides ability to produce tokens at the end-of-segment
- Requires storing info needed to generate output in primitive local state variable
- Alternative is to precede sliding window processing with padK box

Pad Begin Pad End







Pad Begin, Pad End

- PadK box copies input to output and during normal processing
- PadK box EndOfSegment method stuffs K zeros at the end of the segment
- Room for efficiency improvement

```
Name: padK
Type: static

Input: {
  stream float in;
  int K;
}

Output: {
  dynamic stream float out(1,K);
}

Apply: {
   e_vmov(in,1,out,1,granularity);
   produce(out,granularity);
}

EndOfSegment: {
  e_vclr(out,1,K);
  produce(out,K);
}
```

Mirroring

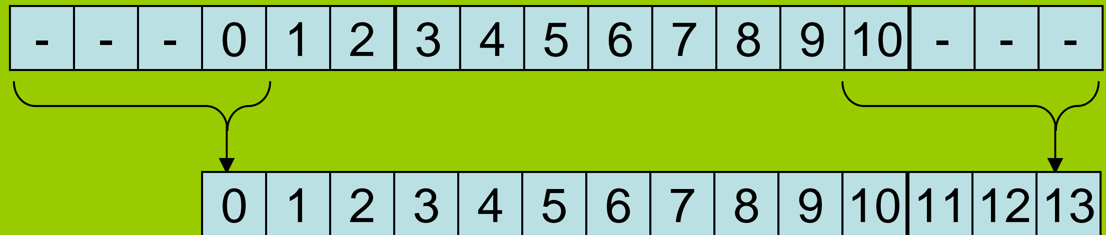
Padding →

Mirroring →

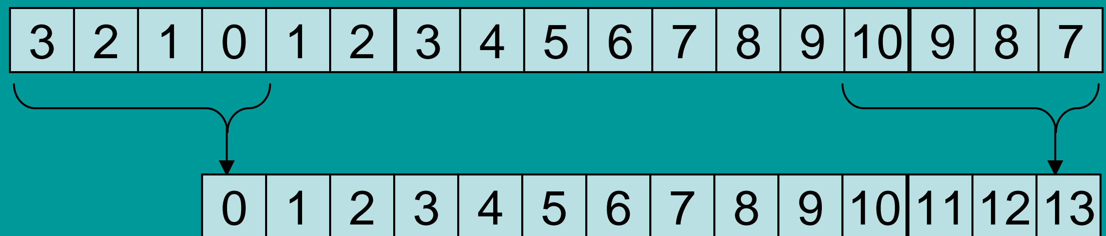
Mirroring

- First impression is mirroring and padding have the same dataflow
- However if length of window is greater than granularity then mirroring is actually quite different.

Pad



Mirror



Mirroring

```
Name: mirrorK
Type: static
Comment: "Modify segment so first N samples are mirror images of samples
1..N of the segment
and also mirror the last N samples"

Input: {
  stream float in;
  int N;
}

Local: {
  float first[N];  /* mirroring at beginning */
  int nfirst;      /* number of samples in first */
  int produced_first; /* have we mirrored the begining yet?
  float last[N+1]; /* samples to mirror at the end */
}

Output: {
  dynamic stream float out(1,3*N);
}
```

Mirroring Box Local Variables used to Collect First Samples and Remember Last.

Mirroring

```
Reset: {
  nfirst = 0;
  produced_first = 0;
}

EndOfSegment: {
  if (produced_first) {
    int i;
    for (i=0; i<N; i++) {
      out[i] = last[N-i-1];
    }
    produce(out,N);
  }
}
```

Reset Method used to note that First samples need to be collected and produced. EndOfSegment Method used to produce last samples

Mirroring

```
Apply: {
  int g;
  int G = granularity;
  int tokens_produced = 0;
  int tokens_mirrored_this_firing = 0;
  for (g = 0; g<granularity && nfirst <N; g++) {
    first[nfirst++] = in[g];
    tokens_mirrored_this_firing++;
  }
  if (!produced_first && nfirst == N) {
    int i;
    for (i=0; i<N; i++) {
       out[i] = first[N-i-1];
    }
    for (i=1; i<N; i++) {
      out[i+N-1] = first[i];
    }
    produce(out,2*N-1);
    out += 2*N-1;
    in += tokens_mirrored_this_firing;
    G -= tokens_mirrored_this_firing;
    tokens_produced = 2*N-1;
    produced_first = 1;
  }
```

In beginning of Apply method primitive remembers the first input tokens and when enough have been collected produces them both mirrrored and forwards

Mirroring

```
if (produced_first) {
    int i;
    int tokens_produced_this_firing = tokens_produced+G;
    int keep_last = N+1-tokens_produced_this_firing;
    e_vmov(in,1,out,1,G);
    produce(out,G);
    if (keep_last > 0) {
      e_vmov(last+N+1-keep_last,1,last,1,keep_last);
      e_vmov(out+G-N-1+keep_last,1,last+keep_last,1,N+1-keep_last);
    } else {
      e_vmov(out+G-N-1,1,last,1,N+1);
    }
  }
}
```

Once past the initial collection of data mirroring primitive copies the input to the output and also remembers the last tokens. Remembering last tokens must be done every time even though only EndOfSegment method will use the tokens.

Decimating Sliding Window

- Sliding window that advances D samples each time
- Example is s_ovrl_v box, window size is N, overlap is Ovrl and decimation rate D = N-Ovrl
- How are segments that are not a multiple of D in length handle?
  - Can dump data at the end of segment
    - (easy approach – what naturally happens)
  - Can pad data at the end of segment to a multiple of D
    - Responsibility of segmenter
    - Responsibility of segment processor (modular approach)
- If segment processing primitive must pad input to a multiple of D it needs to know how many tokens are available on its input
  - Stream input modifier “pad” gives primitive this ability
  - Primitive can use avail function to determine how many tokens are on an input (which may be less than granularity*D) if at the end-of-segment

Padding inputs

- Padding inputs are used when decimate rate is > 1
- A padding input allows the Apply method to query the number of tokens on the input
- If avail(in) < granularity*D then Apply method must internally pad the input with the number of tokens needed to fire at granularity.

Gedae Features Used

- Segmented output stream
- Static stream input overlap parameter
- Static stream output delay parameter
- Dynamic stream
- Reset Method
- End-of-segment method
  - with output stream note declaring how many tokens can be produced at EOS
- Local state variables
  - reset at beginning of segment
  - available to end-of-segment method
- Padding streams

Gedae Consideration

- What can we do in future to better support edge processing?
  - Make nearly static dataflow static
    - Requires creating different schedules to handle beginning and ending processing
  - Provide a complete library of functions to handle edges for all data types.
  - Provide example graphs demonstrating different edge handling techniques