Stream Languages and Programming Models

Saman Amarasinghe and William Thies

Massachusetts Institute of Technology

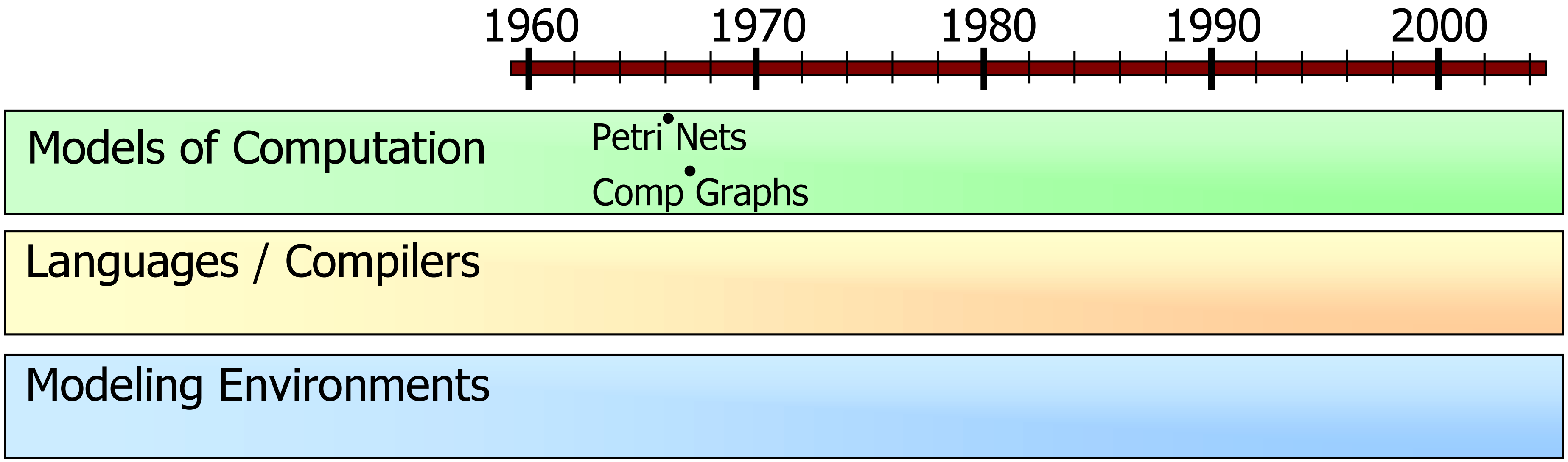
PACT 2003

September 27, 2003

Schedule

1:30-1:40	Overview (Saman)
1:40-2:20	Stream Architectures (Saman)
2:20-3:00	Stream Languages (Bill)
3:00-3:30	Break
3:30-3:55	Stream Compilers (Saman)
3:55-4:20	Domain-specific Optimizations (Saman)
4:20-5:00	Scheduling Algorithms (Bill)

Timeline: 1960's



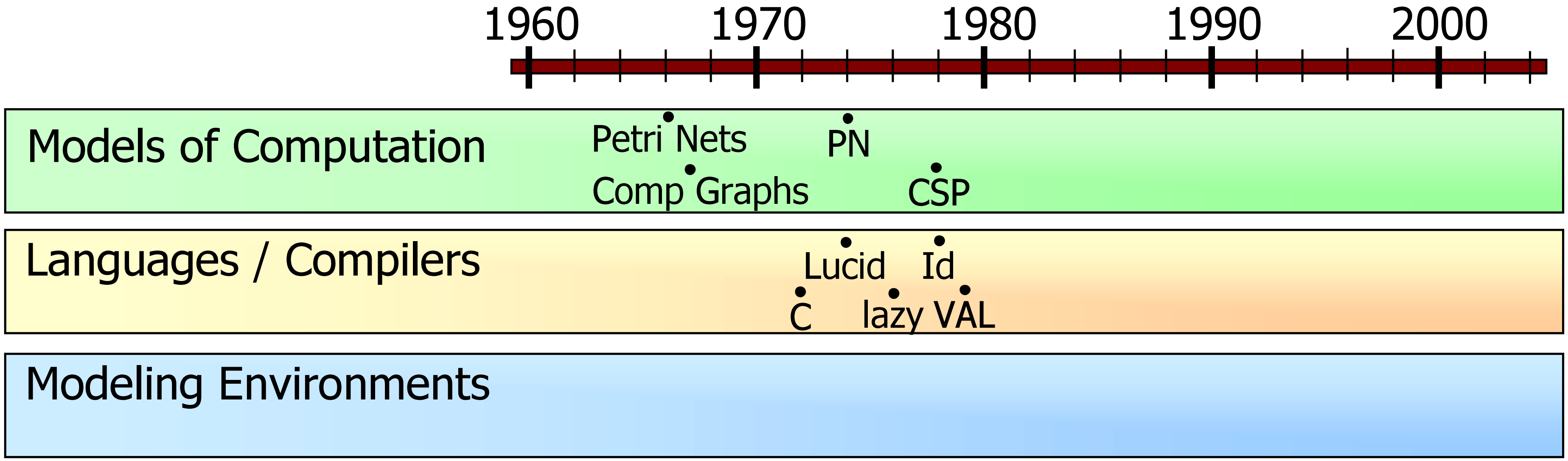
Models of Computation

Languages / Compilers

Modeling Environments

- "Stream" (P.J. Landin) – 1960
  - Linking Algol 60 and lambda calculus, used for loop histories
- Petri Nets (C.A. Petri) – 1966
  - Places, transitions, tokens
- Computation Graphs (Karp, Miller) – 1967
  - Graph with firing actors, minimal firing requirements
  - Formulate determinancy, termination, queuing properties

Timeline: 1970's



1960 1970 1980 1990 2000

Models of Computation

Petri Nets

PN

Comp Graphs

CSP

Languages / Compilers

Lucid

Id

C

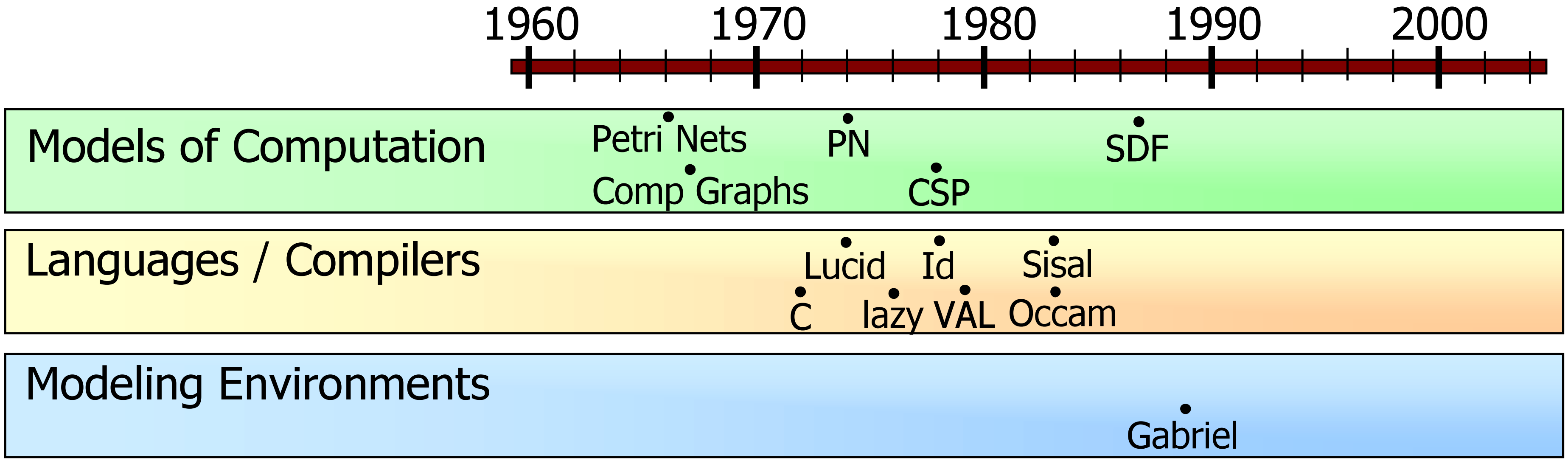
lazy

VAL

Modeling Environments

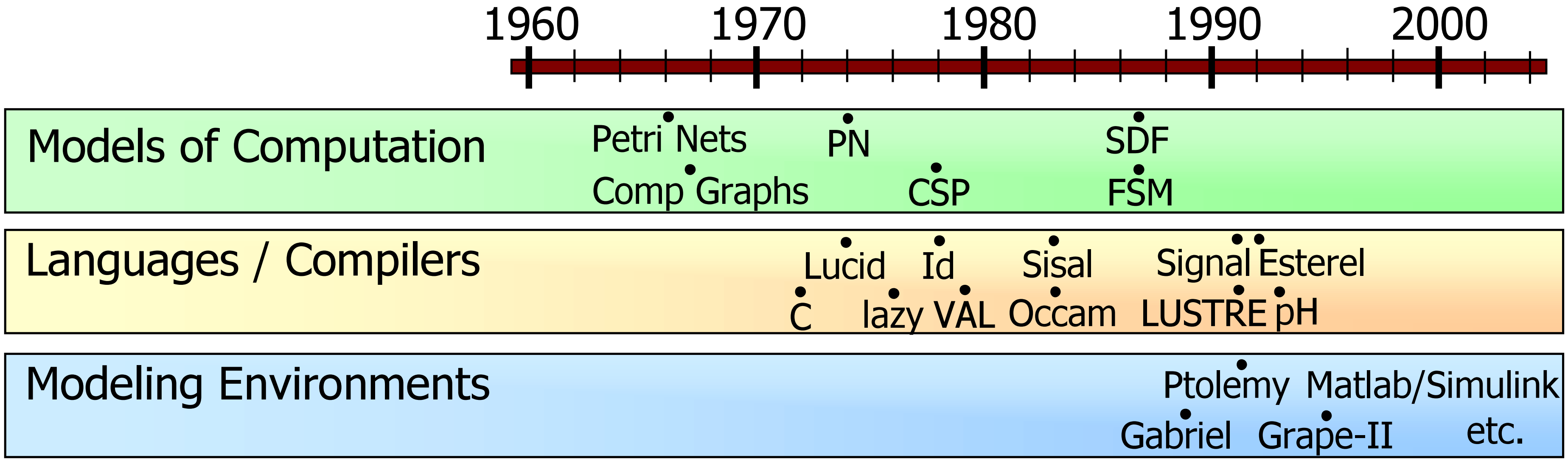
- Process Networks (Kahn) – 1974
  - Sequential threads communicate with unbounded FIFO's
  - Deterministic
- CSP: Communicating Sequential Processes (Hoare) – 1978
  - Sequential threads communicate with rendezvous message-passing
  - Non-deterministic due to guards
- Dataflow languages
  - First version dataflow procedure langauge (Dennis)
  - Lucid (Ashcroft, Wadge), Id (Arvind, Gostelow), VAL (Dennis)
- Functional languages with lazy evaluation for streams
  - lazy evaluator (Henderson, Morris); Sieve of Eratosthenes (Friedman, Wise)

Timeline: 1980's



- SDF: Synchronous Dataflow (Lee, Messerschmitt) – 1987
  - Actors have static, non-uniform rates; firing is atomic and data-driven
  - Allows static scheduling
- Sisal: Streams and Iteration in a Single Assignment Language – 1983
  - Adds recursion, finite streams to VAL
  - Implementations on many parallel machines
  - IF1 intermediate format
- Occam – 1983
  - Strongly typed procedural language
  - Practical implementation of CSP
- More work on dataflow and functional languages (e.g., M. Broy)

Timeline: 1990's



1960 1970 1980 1990 2000

Models of Computation

Petri Nets PN SDF

Comp Graphs CSP FSM

Languages / Compilers

Lucid Id Sisal Signal Esterel

C lazy VAL Occam LUSTRE pH

Modeling Environments

Ptolemy Matlab/Simulink

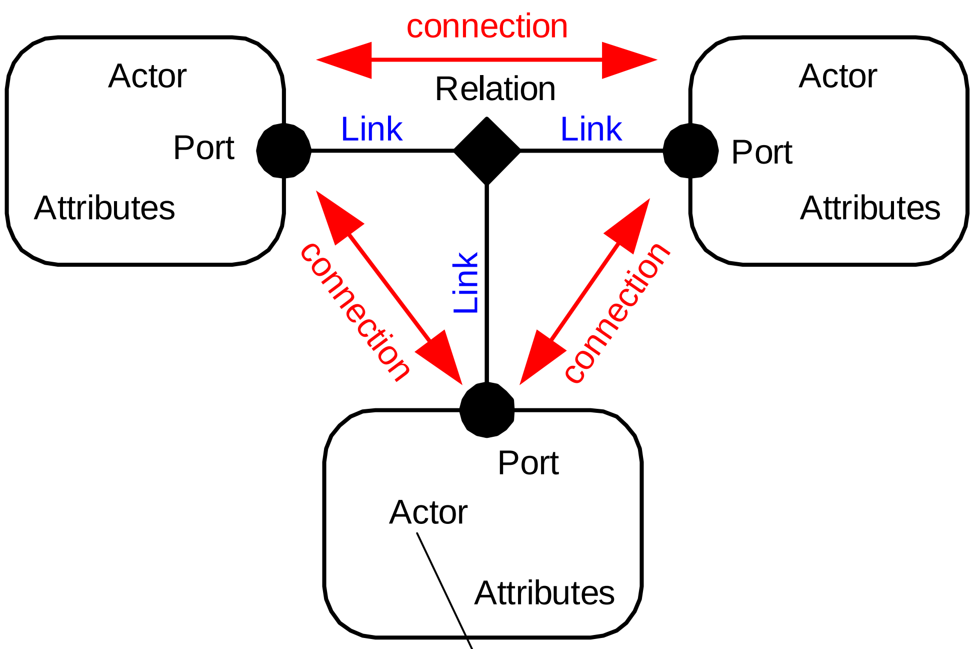
Gabriel Grape-II etc.

- Synchronous Languages: Signal, LUSTRE, etc.
  - Designed for expressiveness, verification moreso than high performance
- Esterel
  - For reactive programming; event-driven and control-oriented
  - Often implemented in either hardware or software
- pH: Parallel Haskell (Nikhil, Arvind, et al.)
  - Combines lazy functional and dataflow philosophies for high performance
- Ptolemy: Heterogeneous Modeling Environment (Lee et al.)
  - Many contributions to formalisms, scheduling, graph-level optimization
- Commercial Environments (Matlab, SPW, COSSAP, ADS, etc.)
  - Becoming increasingly prevalent

Stream Programming Models

- Prototyping environments
- Conventional languages
  - Object Oriented
  - Procedural
  - Assembly
- Stream languages
  - StreamIt
  - Brook
  - Cg

Actor-Oriented Design in the Ptolemy Project (UC Berkeley)



called a "kernel," "step," …

Model of Computation:

- Messaging schema
- Flow of control
- Concurrency

Examples:

- Dataflow
- Process networks
- Synchronous
- Time triggered
- Discrete-event systems
- Publish & subscribe

Most Ptolemy II models of computation are "actor oriented." But the precise semantics depends on the selected "director," which implements a model of computation.

Focus on Dataflow (a few variants)

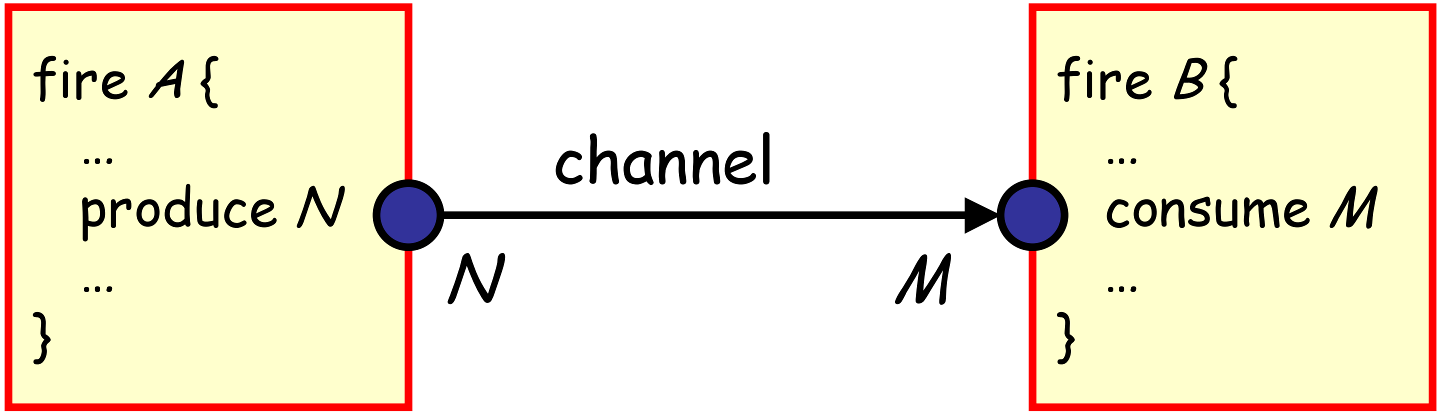
- Computation graphs [Karp & Miller - 1966]
- Process networks [Kahn - 1974]
- Static dataflow [Dennis - 1974]
- Dynamic dataflow [Arvind, 1981]
- K-bounded loops [Culler, 1986]
- Synchronous dataflow [Lee & Messerschmitt, 1986]
- Structured dataflow [Kodosky, 1986]
- PGM: Processing Graph Method [Kaplan, 1987]
- Synchronous languages [Lustre, Signal, 1980's]
- Well-behaved dataflow [Gao, 1992]
- Boolean dataflow [Buck and Lee, 1993]
- Multidimensional SDF [Lee, 1993]
- Cyclo-static dataflow [Lauwereins, 1994]
- Integer dataflow [Buck, 1994]
- Bounded dynamic dataflow [Lee and Parks, 1995]
- Heterochronous dataflow [Girault, Lee, & Lee, 1997]
- …

Many tools, software frameworks, and hardware architectures have been built to support one or more of these.

This slide compliments of Edward A. Lee, UC Berkeley

Synchronous Dataflow (SDF)

Fixed Production/Consumption Rates



```
fire A {
   …
   produce N
   …
}
```

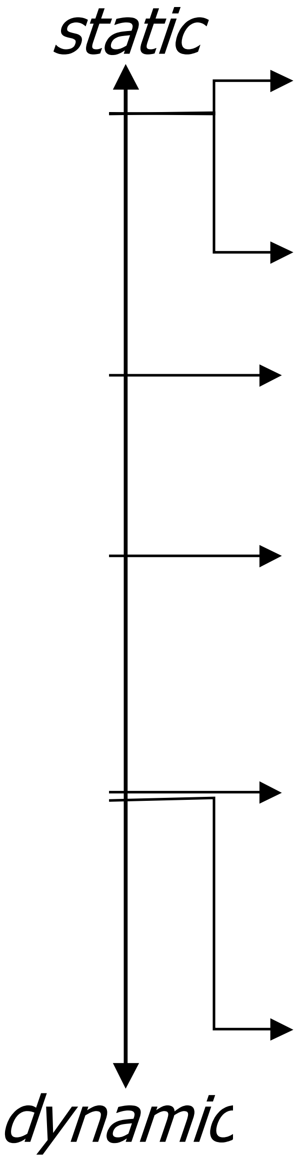
```
fire B {
   …
   consume M
   …
}
```

- Schedulable statically
- Decidable:
  - buffer memory requirements
  - deadlock

➡ Will address in detail in section on scheduling

This slide adapted from Edward A. Lee, UC Berkeley

Selected Generalizations



Multidimensional Synchronous Dataflow (1993)

- Arcs carry multidimensional streams
- One balance equation per dimension per arc

Cyclo-Static Dataflow (Lauwereins, et al., 1994)

- Periodically varying production/consumption rates

Heterochronous Dataflow (1997)

- Combines state machines with SDF graphs
- Very expressive, yet decidable

Boolean & Integer Dataflow (1993/4)

- Balance equations are solved symbolically
- Permits data-dependent routing of tokens
- Heuristic-based scheduling (undecidable)

Dynamic Dataflow (1981-)

- Firings scheduled at run time
- Challenge: maintain bounded memory, deadlock freedom, liveness
- Demand driven, data driven, and fair policies all fail

Kahn Process Networks (1974-)

- Replace discrete firings with process suspension
- Challenge: maintain bounded memory, deadlock freedom, liveness

This slide compliments of Edward A. Lee, UC Berkeley

Other Stream-Like Models of Computation
(all implemented in Ptolemy II)

- Push/Pull
  - dataflow with disciplined nondeterminism
  - e.g. Click (Kohler, 2001)
- Discrete events
  - data tokens have time stamps
  - e.g. NS
- Continuous time
  - streams are a continuum of values
  - e.g. Simulink
- Synchronous languages
  - sequence of values, one per clock tick
  - fixed-point semantics
  - e.g. Esterel
- Time triggered
  - similar, but no fixed-point semantics
  - e.g. Giotto
- Modal models
  - state machines + stream-like MoCs, hierarchical
  - e.g. Hybrid systems

all of these include a logical notion of time

Software Legacy of the Ptolemy Project

- Gabriel (1986-1991)
  - Written in Lisp
  - Aimed at signal processing
  - Synchronous dataflow (SDF) block diagrams
  - Parallel schedulers
  - Code generators for DSPs
  - Hardware/software co-simulators
- Ptolemy Classic (1990-1997)
  - Written in C++
  - Multiple models of computation
  - Hierarchical heterogeneity
  - Dataflow variants: BDF, DDF, PN
  - C/VHDL/DSP code generators
  - Optimizing SDF schedulers
  - Higher-order components
- Ptolemy II (1996-2022)
  - Written in Java
  - Domain polymorphism
  - Multithreaded
  - Network integrated and distributed
  - Modal models
  - Sophisticated type system
  - CT, HDF, CI, GR, etc.

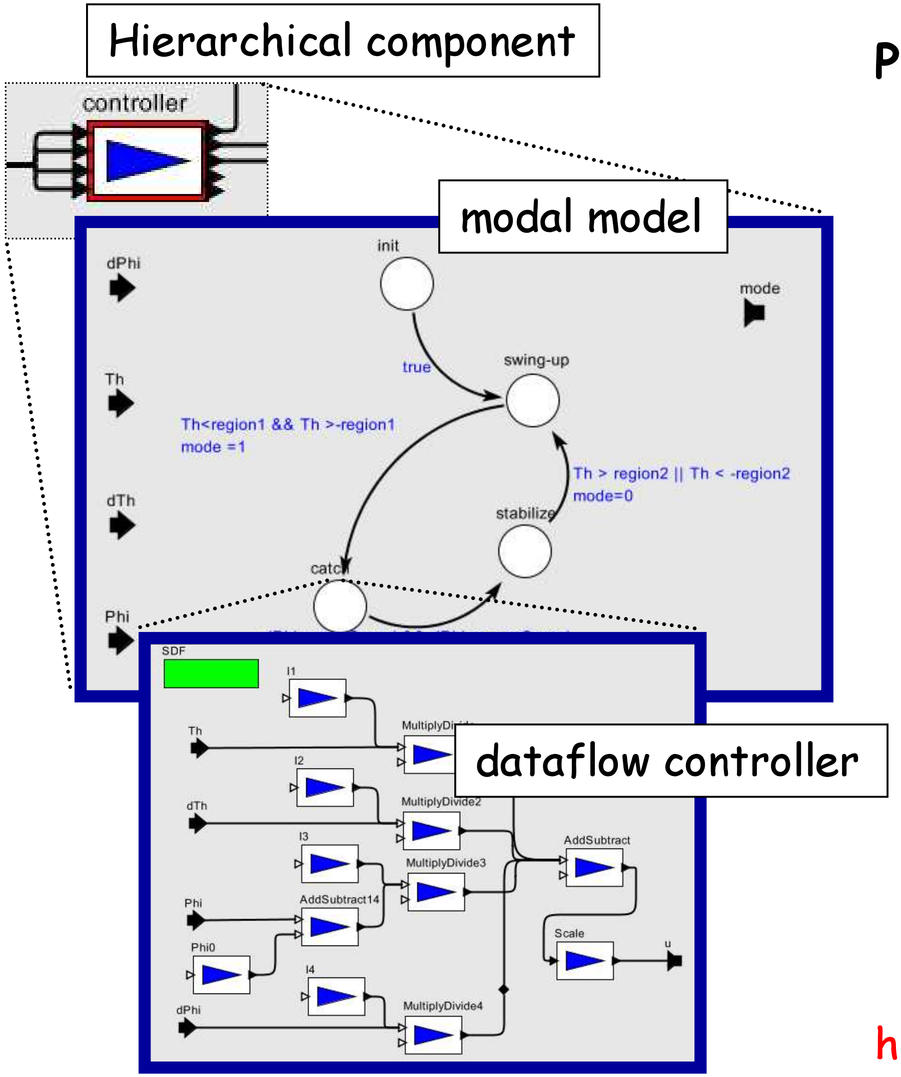
Each of these served, first-and-foremost, as a laboratory for investigating design.

- PtPlot (1997-??)
  - Java plotting package
- Tycho (1996-1998)
  - Itcl/Tk GUI framework
- Diva (1998-2000)
  - Java GUI framework

Focus has always been on embedded software.

This slide compliments of Edward A. Lee, UC Berkeley

Ptolemy II


Hierarchical component

modal model

dataflow controller

example Ptolemy II model: hybrid control system

Ptolemy II:

Our current framework for experimentation with actor-oriented design, concurrent semantics, visual syntaxes, and hierarchical, heterogeneous design.

http://ptolemy.eecs.berkeley.edu

Implementing High-Performance Streaming Applications

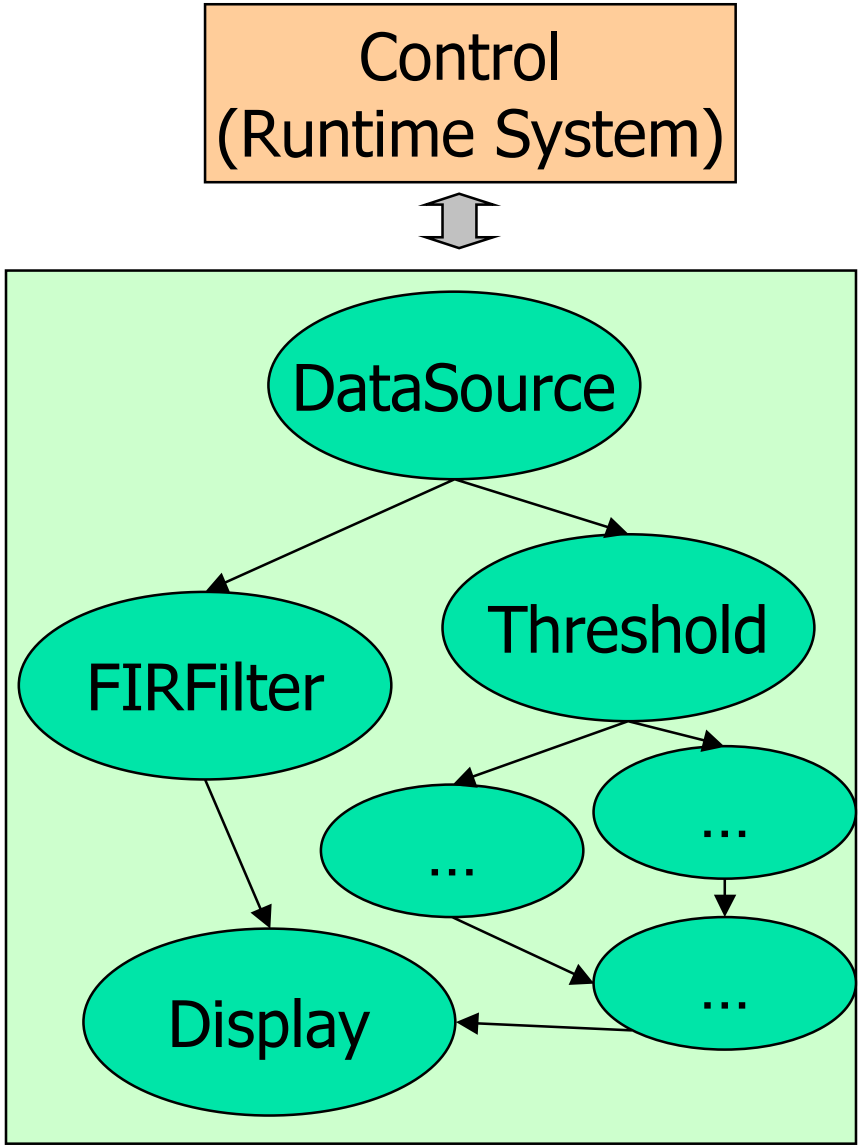
- Modeling environments are good for prototyping, algorithmic optimizations
- However, embedded systems have tight resource constraints:
  - Real-time requirements (throughput, latency)
  - Limited battery life (power)
  - Limited instruction and data memory
- Current practice: re-implement stream algorithm in high-performance language
  - C / assembly
  - C++ runtime system (e.g., Spectrumware)
- New class of “stream languages” aim to raise abstraction level, provide unified development environment
  - StreamIt
  - Brook
  - Cg

Stream Programming Models

- Prototyping environments
- Conventional languages
  - Object Oriented
  - Procedural
  - Assembly
- Stream languages
  - StreamIt
  - Brook
  - Cg

Streaming in Object Oriented Style

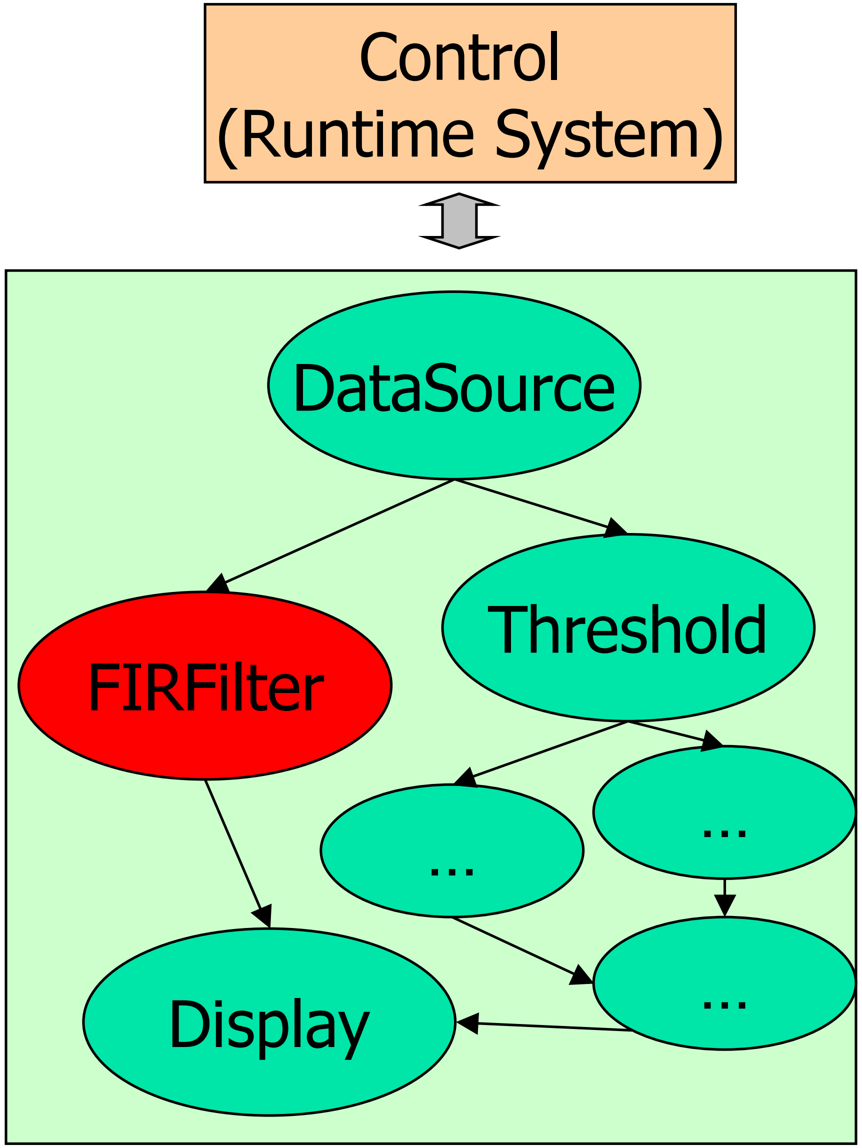
- Each actor is an object
- Scheduled by pull model



Streaming in Object Oriented Style

- Each actor is an object
- Scheduled by pull model

```
class FIRFilter extends Stream {
    int N;
    float[] input;
    void getData(float[] output,
                   int offset, int length) {
            if (input==null) {
                input = new float[MAX_LENGTH];
                source.getData(input, 0, N+length);
            } else
                source.getData(input, N, length);
            for (int i=0; i<length; i++) {
                float sum = 0;
                for (int j=0; j<N; j++)
                        sum = sum + data1[i+j]*h[j][N];
                output[i+offset] = sum;
            }
            for (int i=0; i<N; i++)
                input[i] = input[i+length];
    }
}
```



Control
(Runtime System)

DataSource

FIRFilter

Threshold

…

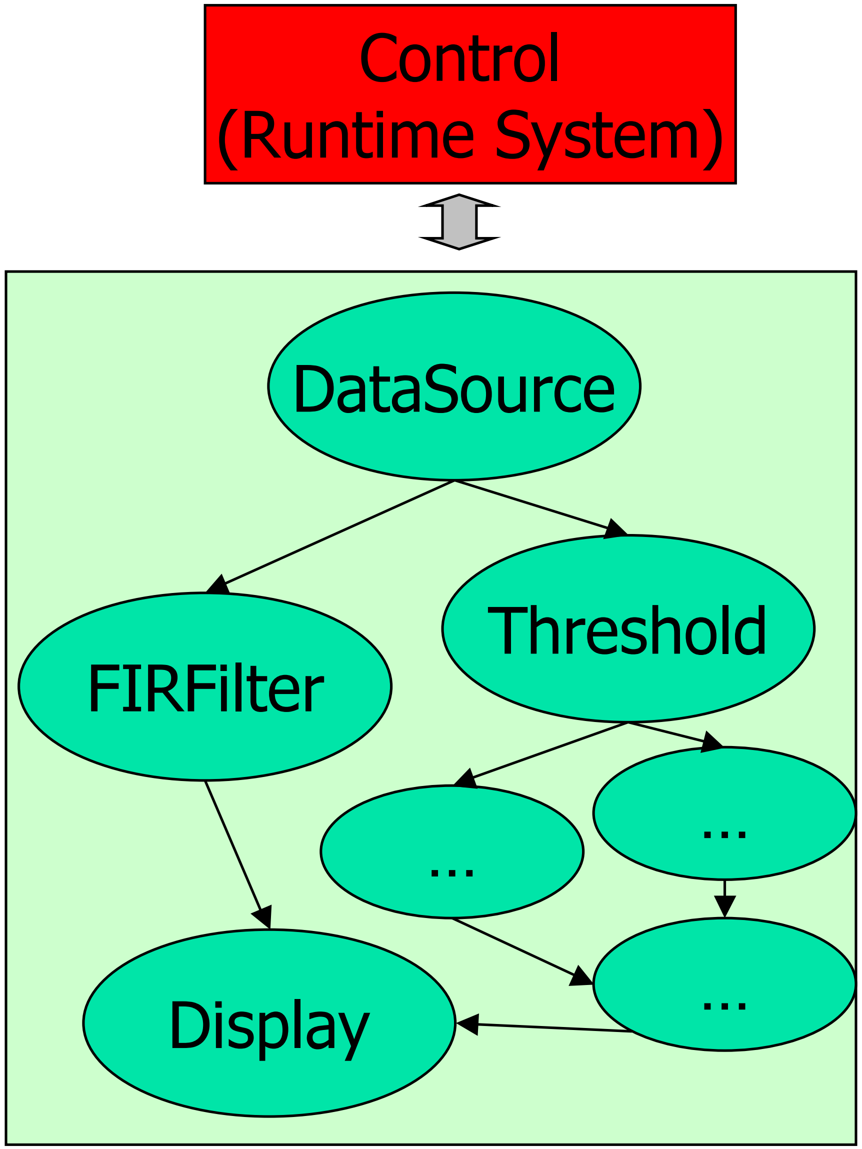
…

…

Display

Streaming in Object Oriented Style

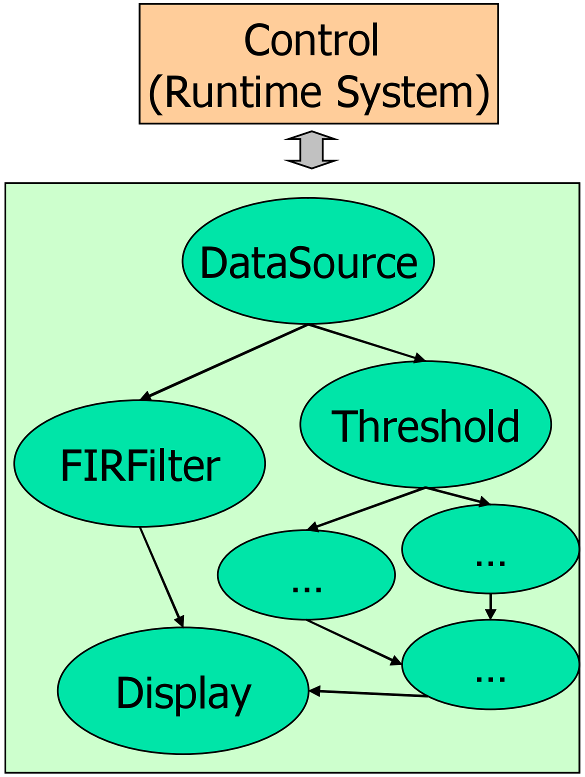
- Each actor is an object
- Scheduled by pull model



```
void main() {
    DataSource datasource = new DataSource();
    FIRFilter filter = new FIRFilter(5);
    Display display = new Display();
    filter.source = datasource;
    display.source = filter;
    display.run();
}
```

Streaming in Object Oriented Style

- Pro:
  - Modular
  - Shows structure of graph
  - Automatic scheduling
- Con:
  - Overhead of objects
    - Communication is static; don't need virtual dispatch
  - Coarse-grained communication
    - Block size is architecture-dependent
    - Obscures fine-grained algorithm
  - Overhead of run-time scheduler
    - Lots of method calls
    - Impossible to keep persistent data in registers
  - Compiler can't optimize across module boundaries



Stream Programming Models

- Prototyping environments
- Conventional languages
  - Object Oriented
  - Procedural
  - Assembly
- Stream languages
  - StreamIt
  - Brook
  - Cg

Streaming in Procedural Style

```
int N = 5;
int BLOCK_SIZE = 100;

void main() {
    float input[] = new float[N];
    float output[] = new float[BLOCK_SIZE];
    int i, j;
    for (i=0; i<N; i++)
              input[i] = getData();
    while (true) {
              for (out=0; i<N; i++, j++)
                  step(input, output, i, j);
              int wholeSteps = (BLOCK_SIZE-j)/N;
              for (int k=0; k<wholeSteps; k++)
                  for (i=0; i<N; i++, j++)
                            step(input, output, i, j);
              for (i=0; j<BLOCK_SIZE; i++, j++)
                  step(input, output, i, j);
              displayBlock(output);
    }
}
```

- Complicated loop nest
  - Statements in loops represent actors
  - Circular buffers for data items
  - Scheduling done by hand
  - Loop bounds adjusted for cache size

```
void step(float[] input, float[] output, int i, int j)
{
    float sum = 0;
    for (int k=0; k<i; k++)
              sum = sum + input[k]*h[k+i][N];
    for (int k=i; k<N; k++)
              sum = sum + input[k]*h[k-i][N];
    output[j] = sum;
    input[i] = getData();
}
```

Streaming in Procedural Style

```
int N = 5;
int BLOCK_SIZE = 100;

void main() {
    float input[] = new float[N];
    float output[] = new float[BLOCK_SIZE];
    int i, j;
    for (i=0; i<N; i++)
              input[i] = getData();
    while (true) {
              for (out=0; i<N; i++, j++)
                  step(input, output, i, j);
              int wholeSteps = (BLOCK_SIZE-j)/N;
              for (int k=0; k<wholeSteps; k++)
                  for (i=0; i<N; i++, j++)
                            step(input, output, i, j);
              for (i=0; j<BLOCK_SIZE; i++, j++)
                  step(input, output, i, j);
              displayBlock(output);
    }
}
```

- Pro:
  - Better performance than object-oriented style
- Con:
  - Obscures parallelism and communication patterns
  - Scheduling and buffer management done by hand
    - Difficult get it right
    - Hard to maintain
    - Impossible for compiler to optimize for given resources
  - No modularity
    - Actors are mixed with global variables and control flow
  - Hard to visualize computation

Stream Programming Models

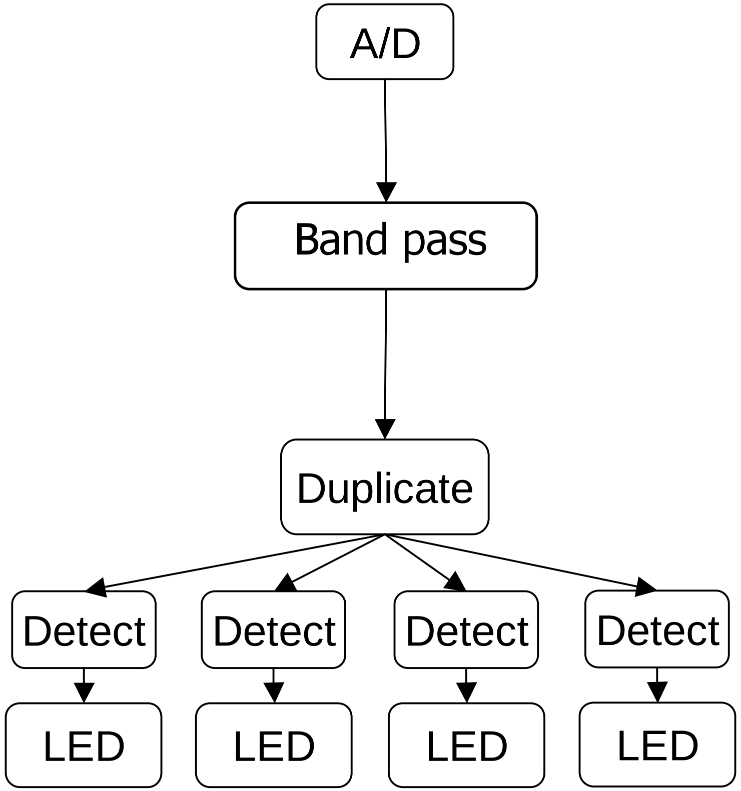
- Prototyping environments
- Conventional languages
  - Object Oriented
  - Procedural
  - Assembly
- Stream languages
  - StreamIt
  - Brook
  - Cg

Streaming in Assembly Code

- Example: Freq band detection

- Used in…
  - metal detector
  - garage door opener
  - spectrum analyzer

Source:
Application Report SPRA414
Texas Instruments, 1999



DSP Implementation

```
;********************************************************************
; File Name: FIR0.ASM
; Originator: Digital control systems Apps group - Houston
; Target Sys: 'C24x Evaluation Board
;
; Description: FIR bandpass filter which detects the presence of a
; 500Hz signal. If the tone is detected an LED is
; lit by using the output port. Sampling Frequency
; forced to be 4kHz.
;
; Last Update: 9 June 1997
;
;********************************************************************
.include f240regs.h
;--------------------------------------------------------------------
; I/O Mapped EVM Registers
;--------------------------------------------------------------------
DAC0 .set 0000h ;Input data register for DAC0
DAC1 .set 0001h ;Input data register for DAC1
DAC2 .set 0002h ;Input data register for DAC2
DAC3 .set 0003h ;Input data register for DAC3
DACUPDATE .set 0004h ;DAC Update Register
;--------------------------------------------------------------------
; Variable Declarations for B2
;--------------------------------------------------------------------
.bss GPR0,1 ;General Purpose Register
.bss DAC0VAL,1 ;DAC0 Channel Value
.bss DAC1VAL,1 ;DAC1 Channel Value
.bss DAC2VAL,1 ;DAC2 Channel Value
.bss DAC3VAL,1 ;DAC3 Channel Value
;--------------------------------------------------------------------
; Vector address declarations
;--------------------------------------------------------------------
.sect ".vectors"
RSVECT B START ; Reset Vector
INT1 B PHANTOM ; Interrupt Level 1
INT2 B FIR_ISR ; Interrupt Level 2
INT3 B PHANTOM ; Interrupt Level 3
INT4 B PHANTOM ; Interrupt Level 4
INT5 B PHANTOM ; Interrupt Level 5
INT6 B PHANTOM ; Interrupt Level 6
RESERVED B PHANTOM ; Reserved
SW_INT8 B PHANTOM ; User S/W Interrupt
SW_INT9 B PHANTOM ; User S/W Interrupt
SW_INT10 B PHANTOM ; User S/W Interrupt
SW_INT11 B PHANTOM ; User S/W Interrupt
SW_INT12 B PHANTOM ; User S/W Interrupt
SW_INT13 B PHANTOM ; User S/W Interrupt
SW_INT14 B PHANTOM ; User S/W Interrupt
SW_INT15 B PHANTOM ; User S/W Interrupt
SW_INT16 B PHANTOM ; User S/W Interrupt
TRAP B PHANTOM ; Trap vector
NMINT B PHANTOM ; Non-maskable Interrupt
EMU_TRAP B PHANTOM ; Emulator Trap
SW_INT20 B PHANTOM ; User S/W Interrupt
SW_INT21 B PHANTOM ; User S/W Interrupt
SW_INT22 B PHANTOM ; User S/W Interrupt
SW_INT23 B PHANTOM ; User S/W Interrupt
```

```
;====================================================================
; M A I N C O D E - starts here
;====================================================================
.text
NOP
START: SETC INTM ;Disable interrupts
SPLK #0002h,IMR ;Mask all core interrupts
; except INT2
LACC IFR ;Read Interrupt flags
SACL IFR ;Clear all interrupt flags
CLRC SXM ;Clear Sign Extension Mode
CLRC OVM ;Reset Overflow Mode
CLRC CNF ;Config Block B0 to Data mem
;-----------------------------------
; Set up PLL Module
;-----------------------------------
LDP #00E0h ;DP = 224; Address for
;7000h - 707Fh
;The following line is necessary if a previous program set the PLL
;to a different setting than the settings which the application
;uses. By disabling the PLL, the CKCR1 register can be modified
;so that the PLL can run at the new settings when it is re-enabled.
SPLK #0000000001000001b,CKCR0 ;CLKMD=PLL Disable
;SYSCLK=CPUCLK/2
; 5432109876543210
SPLK #0000000010111011b,CKCR1
;CLKIN(OSC)=10MHz,CPUCLK=20MHz
;CKCR1 - Clock Control Register 1
;Bits 7-4 (1011)CKINF(3)-CKINF(0) - Crystal or Clock-In Frequency
; Frequency = 10MHz
;Bit 3 (1) PLLDIV(2) - PLL divide by 2 bit
; Divide PLL input by 2
;Bits 2-0 (011) PLLFB(2)-PLLFB(0) - PLL multiplication ratio
; PLL Multiplication Ration = 4
; 5432109876543210
SPLK #0000000011000011b,CKCR0
;CLKMD=PLL Enable, SYSCLK=CPUCLK/2
;CKCR0 - Clock Control Register 0
;Bits 7-6 (11) CLKMD(1),CLKMD(0) - Operational mode of Clock
; Module
; PLL Enabled; Run on CLKIN on exiting low power mode
;Bits 5-4 (00) PLLOCK(1),PLLOCK(0) - PLL Status. READ ONLY
;Bits 3-2 (00) PLLPM(1),PLLPM(0) - Low Power Mode
; LPM0
;Bit 1 (0) ACLKENA - 1MHz ACLK Enable
; ACLK Enabled
;Bit 0 (1) PLLPS - System Clock Prescale Value
; f(sysclk)=f(cpuclk)/2
; 5432109876543210
SPLK #0100000011000000b,SYSCR ;CLKOUT=CPUCLK
;SYSCR - System Control Register
;Bit 15-14 (01) RESET1,RESET0 - Software Reset Bits
; No Action
;Bits 13-8 (000000) Reserved
;Bit 7-6 (11) CLKSRC1,CLKSRC0 - CLKOUT-Pin Source Select
; CPUCLK: CPU clock output mode
;Bit 5-0 (000000) Reserved
SPLK #006Fh, WDCR ;Disable WD if VCCP=5V (JP5 in pos. 2-3)
KICK_DOG ;Reset Watchdog
```

```
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;- Event Manager Module Reset
;*
;-This section resets all of the Event Manager Module Registers.
;*This is necessary for silicon revsion 1.1; however, for
;-silicon revisions 2.0 and later, this is not necessary
;*
;-
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
LDP #232 ;DP=232 Data Page for the Event
;Manager
SPLK #0000h,GPTCON ;Clear General Purpose Timer Control
SPLK #0000h,T1CON ;Clear GP Timer 1 Control
SPLK #0000h,T2CON ;Clear GP Timer 2 Control
SPLK #0000h,T3CON ;Clear GP Timer 3 Control
SPLK #0000h,COMCON ;Clear Compare Control
SPLK #0000h,ACTR ;Clear Full Compare Action Control
;Register
SPLK #0000h,SACTR ;Clear Simple Compare Action Control
;Register
SPLK #0000h,DBTCON ;Clear Dead-Band Timer Control
;Register
SPLK #0FFFFh,EVIFRA;Clear Interrupt Flag Register A
SPLK #0FFFFh,EVIFRB;Clear Interrupt Flag Register B
SPLK #0FFFFh,EVIFRC;Clear Interrupt Flag Register C
SPLK #0000h,CAPCON ;Clear Capture Control
SPLK #0000h,EVIMRA ;Clear Event Manager Mask Register A
SPLK #0000h,EVIMRB ;Clear Event Manager Mask Register B
SPLK #0000h,EVIMRC ;Clear Event Manager Mask Register C
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
; End of RESET section for silicon revision 1.1 *
;*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;-----------------------------------
; Set up Event Manager Module
;-----------------------------------
T1COMPARE .set 2500
T1PERIOD .set 5000 ;Sets up period for 4kHz frequency
LDP #232 ;DP=232, Data Page for Event Manager
Addresses
SPLK #T1COMPARE,T1CMPR;Compare value for 50% duty cycle
; 2109876543210
SPLK #0000001010101b,GPTCON
;GPTCON - GP Timer Control Register
;Bit 15 (0) T3STAT - GP Timer 3 Status. READ ONLY
;Bit 14 (0) T2STAT - GP Timer 2 Status. READ ONLY
;Bit 13 (0) T1STAT - GP Timer 1 Status. READ ONLY
;Bits 12-11 (00) T3TOADC - ADC start by event of GP Timer 3
; No event starts ADC
;Bits 10-9 (00) T2TOADC - ADC start by event of GP Timer 2
; No event starts ADC
;Bits 8-7 (00) T1TOADC - ADC start by event of GP Timer 1
; No event starts ADC
;Bit 6 (1) TCOMPOE - Compare output enable
; Enable all three GP timer compare outputs
;Bits 5-4 (01) T3PIN - Polarity of GP Timer 3 compare output
; Active Low
;Bits 3-2 (01) T2PIN - Polarity of GP Timer 2 compare output
; Active Low
;Bits 1-0 (01) T1PIN - Polarity of GP Timer 1 compare output
; Active Low
SPLK #T1PERIOD,T1PR ;Period value for 2kHz signal
SPLK #0000h,T1CNT ;Clear GP Timer 1 Counter
SPLK #0000h,T2CNT ;Clear GP Timer 2 Counter
SPLK #0000h,T3CNT ;Clear GP Timer 3 Counter
; 5432109876543210
SPLK #0001000000000010b,T1CON
```

Source: Application Report SPRA414, Texas Instruments, 1999

Cont.

```
;T1CON - GP Timer 1 Control Register
;Bits 15-14(00) FREE,SOFT - Emulation Control Bits
; Stop immediately on emulation suspend
;Bits 13-11(010) TMODE2-TMODE0 - Count Mode Selection
; Continuous-Up Count Mode
;Bits 10-8 (000) TPS2-TPS0 - Input Clock Prescaler
; Divide by 1
;Bit 7 (0) Reserved
;Bit 6 (0) TENABLE - Timer Enable
; Disable timer operations
;Bits 5-4 (00) TCLKS1,TCLKS0 - Clock Source Select
; Internal Clock Source
;Bits 3-2 (00) TCLD1,TCLD0 - Timer Compare Register Reload
; Condition
; When counter is 0
;Bit 1 (1) TECMPR - Timer compare enable
; Enable timer compare operation
;Bit 0 (0) Reserved
; 5432109876543210
SPLK #0000000000000000b,T2CON
;GP Timer 2 - Not Used
;T2CON - GP Timer 2 Control Register
;Bits 15-14(00) FREE,SOFT - Emulation Control Bits
; Stop immediately on emulation suspend
;Bits 13-11(000) TMODE2-TMODE0 - Count Mode Selection
; Stop/Hold
;Bits 10-8 (000) TPS2-TPS0 - Input Clock Prescaler
; Divide by 1
;Bit 7 (0) TSWT1 - GP Timer 1 timer enable bit
; Use own TENABLE bit
;Bit 6 (0) TENABLE - Timer Enable
; Disable timer operations
;Bits 5-4 (00) TCLKS1,TCLKS0 - Clock Source Select
; Internal Clock Source
;Bits 3-2 (00) TCLD1,TCLD0 - Timer Compare Register Reload
; Condition
; When counter is 0
;Bit 1 (0) TECMPR - Timer compare enable
; Disable timer compare operation
;Bit 0 (0) SELT1PR - Period Register select
; Use own period register
; 5432109876543210
SPLK #0000000000000000b,T3CON
;GP Timer 3 - Not Used
;T3CON - GP Timer 3 Control Register
;Bits 15-14(00) FREE,SOFT - Emulation Control Bits
; Stop immediately on emulation suspend
;Bits 13-11(000) TMODE2-TMODE0 - Count Mode Selection
; Stop/Hold
;Bits 10-8 (000) TPS2-TPS0 - Input Clock Prescaler
; Divide by 1
;Bit 7 (0) TSWT1 - GP Timer 1 timer enable bit
; Use own TENABLE bit
;Bit 6 (0) TENABLE - Timer Enable
; Disable timer operations
;Bits 5-4 (00) TCLKS1,TCLKS0 - Clock Source Select
; Internal Clock Source
;Bits 3-2 (00) TCLD1,TCLD0 - Timer Compare Register Reload
; Condition
; When counter is 0
;Bit 1 (0) TECMPR - Timer compare enable
; Disable timer compare operation
;Bit 0 (0) SELT1PR - Period Register select
; Use own period register
```

```
;-----------------------------------
; Set up Digital I/O Port
;-----------------------------------
LDP #225 ;DP=225, Data Page to Configure OCRA
; 5432109876543210
SPLK #0011100000001111b,OCRA
;OCRA - Output Control Register A
;Bit 15 (0) CRA.15 - IOPB7
;Bit 14 (0) CRA.14 - IOPB6
;Bit 13 (1) CRA.13 - T3PWM/T3CMP
;Bit 12 (1) CRA.12 - T2PWM/T2CMP
;Bit 11 (1) CRA.11 - T1PWM/T1CMP
;Bit 10 (0) CRA.10 - IOPB2
;Bit 9 (0) CRA.9 - IOPB1
;Bit 8 (0) CRA.8 - IOPB0
;Bits 7-4 (0000)Reserved
;Bit 3 (1) CRA.3 - ADCIN8
;Bit 2 (1) CRA.2 - ADCIN9
;Bit 1 (1) CRA.1 - ADCIN1
;Bit 0 (1) CRA.0 - ADCIN0
;-----------------------------------
; Set up ADC Module
;-----------------------------------
LDP #224
; 5432109876543210
SPLK #1000100100000000b,ADCTRL1
;ADCTRL1 - ADC Control Register 1
;Bit 15 (1) Suspend-SOFT -
; Complete Conversion before halting emulator
;Bit 14 (0) Suspend-FREE -
; Operations is determined by Suspend-SOFT
;Bit 13 (0) ADCIMSTART - ADC start converting immediately
; No Action
;Bit 12 (0) ADC2EN - Enable/Disable ADC2
; Disable ADC2
;Bit 11 (1) ADC1EN - Enable/Disable ADC1
; Enable ADC1
;Bit 10 (0) ADCCONRUN - ADC Continuous Conversion Mode
; Disable Continuous Conversion
;Bit 9 (0) ADCINTEN - Enable ADC Interrupt
; Mask ADC Interrupt
;Bit 8 (1) ADCINTFLAG - ADC Interrupt Flag
; Clear Interrupt Flag Bit
;Bit 7 (0) ADCEOC - End of Conversion Bit READ ONLY
;Bits 6-4 (000) ADC2CHSEL - ADC2 Channel Select
; Channel 8
;Bits 3-1 (000) ADC1CHSEL - ADC1 Channel Select
; Channel 0
;Bit 0 (0) ADCSOC - ADC Start of conversion bit
; No Action
; 5432109876543210
SPLK #0000000000000101b,ADCTRL2
;ADCTRL2 - ADC Control Register 2
;Bits 15-11 (00000)Reserved
;Bit 10 (0) ADCEVSOC - Event Manager SOC mask bit
; Mask ADCEVSOC
;Bit 9 (0) ADCEXTSOC - External SOC mask bit
; Mask ADCEXTSOC
;Bit 8 (0) Reserved
;Bits 7-6 (00) ADCFIFO1 - Data Register FIFO1 Status READ
ONLY
;Bit 5 (0) Reserved
;Bits 4-3 (00) ADCFIFO2 - Data Register FIFO2 Status READ
ONLY
;Bits 2-0 (101) ADCPSCALE - ADC Input Clock Prescaler
; Prescale Value 16
; SYSCLK Period = 0.1usec
; 0.1usec x 16 x 6 = 9.6 usec >= 6usec
```

```
;-----------------------------------
; Set up DAC Module
;-----------------------------------
;The DAC module requires that wait states be generated for proper
;operation.
LDP #0000h ;Set Data Page Pointer to 0000h, Block B2
SPLK #4h,GPR0 ;Set Wait State Generator for
OUT GPR0,WSGR ;Program Space, 0WS
;Date Space, 0WS
;I/O Space, 1WS
;-------------------------------------------------------------------
; MAIN LINE
;-------------------------------------------------------------------
.sect ".blk0"
XVALUE .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.sect ".blk1"
VALUEIN .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
VALUEOUT .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.data
;Coefficients for 500Hz Bandpass filter for 4kHz Sampling Frequency
BCOEFF .word 0000h,0002h,0002h,0001h
.word 0000h,0000h,0000h,0FFFFh
.word 0FFFFh,0000h,0000h,0002h
.word 0002h,0FFFFh,0FFF9h,0FFF7h
.word 0000h,0013h,0025h,0021h
.word 0FFFBh,0FFBCh,0FF90h,0FFA7h
.word 0011h,00A4h,00FDh,00BFh
.word 0FFDEh,0FEC3h,0FE2Ah,0FEA7h
.word 0033h,0206h,02F1h,0220h
.word 0FFC2h,0FD19h,0FBD6h,0FD05h
.word 003Ch,03B8h,054Ah,03C5h
.word 0FFD4h,0FBB4h,0F9EAh,0FBADh
.word 0010h,0484h,0660h,0484h
.word 0010h,0FBADh,0F9EAh,0FBB4h
.word 0FFD4h,03C5h,054Ah,03B8h
.word 003Ch,0FD05h,0FBD6h,0FD19h
.word 0FFC2h,0220h,02F1h,0206h
.word 0033h,0FEA7h,0FE2Ah,0FEC3h
.word 0FFDEh,00BFh,00FDh,00A4h
.word 0011h,0FFA7h,0FF90h,0FFBCh
.word 0FFFBh,0021h,0025h,0013h
.word 0000h,0FFF7h,0FFF9h,0FFFFh
.word 0002h,0002h,0000h,0000h
.word 0FFFFh,0FFFFh,0000h,0000h
.word 0000h,0001h,0002h,0002h
.word 0000h
LEDS .set 000Ch ;I/O Address for LEDS register
WINDOW .set 500 ;Number of smpls to check before
;reset’g MAX values
.bss LEDSOUT,1 ;Variable for which LEDS to light
.bss MAXIN,1 ;Maximum value input value
.bss MAXOUT,1 ;Maxumum FIR result value
.bss DIFFIN,1 ;Maximum Input Value - DC Offset
;(7ffh)
.bss DIFFOUT,1 ;Maximum Output value - DC Offset
;(7ffh)
.bss THRESHOLD1,1 ;Threshold value for 1st LED
.bss THRESHOLD2,1 ;Threshold value for 2nd LED
.bss THRESHOLD3,1 ;Threshold value for 3rd LED
.bss THRESHOLD4,1 ;Threshold value for 4th LED
.bss THRESHOLD5,1 ;Threshold value for 5th LED
.bss THRESHOLD6,1 ;Threshold value for 6th LED
.bss THRESHOLD7,1 ;Threshold value for 7th LED
.bss THRESHOLD8,1 ;Threshold value for 8th LED
.bss RESET_MAX,1 ;Counter to determine when to
```

Source: Application Report SPRA414, Texas Instruments, 1999

Cont. Cont.

```
;reset MAX values
.bss TEMP,1 ;Variable for temporary storage
;of values
.text
MAIN LAR AR1,#ADCFIFO1 ; AR1 = ADCFIFO1 address
LAR AR2,#ADCTRL1 ; AR2 = ADCTRL1 address
LAR AR3,#BCOEFF ; AR3 = BCOEFF address
LAR AR5,#LEDS ; AR5 = LEDS Output
LDP #232
LACC EVIFRA ;ACC = Event Module Type A Interrupt
;Flags
SACL EVIFRA ;EVIFRA = ACC; Clears the current
;set flags
SPLK #0080h,EVIMRA ; Enable Timer 1 Period
; Interrupt
MAR *,AR2 ;ARP = AR2
LACC * ;ACC = ADCTRL1
ADD #1 ;SET BIT FOR SINGLE CONVERSION
SACL *,0,AR1 ;STARTS ADC CONVERSION
SBIT1 T1CON,B6_MSK ;Sets Bit 6 of T1CON; Starts
;the timer
LDP #0 ;DP = 0; Addresses 0000h - 007Fh
SPLK #0000h,LEDSOUT ;Clear the LEDS
OUT LEDSOUT,LEDS
SPLK #0E38h,THRESHOLD1;Q15 value for 1/9
SPLK #1C71h,THRESHOLD2;Q15 value for 2/9
SPLK #2AAAh,THRESHOLD3;Q15 value for 3/9
SPLK #38E3h,THRESHOLD4;Q15 value for 4/9
SPLK #471Ch,THRESHOLD5;Q15 value for 5/9
SPLK #5555h,THRESHOLD6;Q15 value for 6/9
SPLK #638Eh,THRESHOLD7;Q15 value for 7/9
SPLK #71C7h,THRESHOLD8;Q15 value for 8/9
SPLK #0000h,MAXIN ;Initialize Maxmimum input
;value
SPLK #0000h,MAXOUT ;Initialize Maximum FIR output
;value
SPLK #WINDOW,RESET_MAX ;Initialize the maximum
;reset counter
CLRC INTM ;Enable Interrupts
WAIT B WAIT ;Wait for interrupt
;-------------------------------------------------------------------
; INTERRUPT SERVICE ROUTINES FOR FIR FILTER
;-------------------------------------------------------------------
FIR_ISR LAR AR4,#XVALUE+100 ;AR4 = DATA ADDRESS
MAR *,AR1 ;ARP = AR1 = ADCFIFO1
LACC *,0,AR4 ;ACC = ADCFIFO1; ARP =
AR4
LDP #0 ;DP = 0
;Addresses 0000h - 007Fh
SACL DAC1VAL ;DAC1VAL = ADCFIFO1
RPT #7 ;Shift ADC value 8 places
; - Reduce to 8 bit value
SFR ;Larger bit values produced
;large results
SUB #7Fh ;Subtract the equivalent 8 bit
;DC offset
LDP #04h ;DP = 4; Address 0200h - 027Fh
SACL XVALUE ;XVALUE = ADCFIFO1 / 256;
LACC #0h ;Initialize the ACCUMULATOR
MPY #0h ;Initialize the PROD REG
RPT #100 ;Calculate Y
MACD BCOEFF,*- ;Multiply X with B, and add
APAC ;final accumulation
LDP #0
RPT #7 ;Shift the result 8 places to left
SFL
SACH DAC0VAL,1 ;DAC0VAL = Y * 2; shift to
;remove extra sign bit
;FIR result to output
```

```
;Multiply the values by 5/4 because the maximum gain is 4/5
LT DAC0VAL ;TREG = DAC0VAL
MPY #5 ;PREG = DAC0VAL * 5
PAC ;ACC = PREG = DAC0VAL * 5
SFR ;ACC = DAC0VAL * 5 / 2
SFR ;ACC = DAC0VAL * 5 / 4
SACL DAC0VAL ;DAC0VAL = DAC0VAL * 5/4
LACC DAC0VAL ;ACC = DAC0VAL
RPT #3 ;Shift right 4 times
; = 16 bit value to
SFR ;12 bit value because
;DAC is 12bits
ADD #7FFh ;Add DC offset
AND #0FFFh ;Ensure 12 bits
SACL DAC0VAL ;Store value for output on the DAC
LDP #7 ;DP=7; Address for 0380h to 03FFh
SACL VALUEOUT ;Store value to find maximum
;value of the output values
LAR AR6,#(VALUEOUT+127-1) ;AR6 = End of VALUE OUT
;buffer
LAR AR7,#126 ;AR7 = 127 - 1; Number of
;values to move
MAR *,AR6 ;ARP = AR6
SHIFT1 DMOV *-,AR7 ;Move all of the values in
;the VALUEOUT
BANZ SHIFT1,*-,AR6 ;Data Buffer to the next
;higher address
LDP #0 ;DP = 0; Addresses 0000h - 007Fh
LACC DAC1VAL ;ACC = DAC1VAL = Input Value
RPT #3 ;Shift the value to the
;right 4 times
SFR ;Convert the value from 16
;bits to 12 bits
SACL DAC1VAL ;DAC1VAL = 12 bit value for DAC
LDP #6 ;DP = 6; Addresses 0300h - 037Fh
SACL VALUEIN ;VALUEIN = DAC1VAL
LAR AR6,#(VALUEIN+127-1);AR6 = End of VALUE IN
;buffer
LAR AR7,#126 ;AR7 = 127 - 1; Number of
;values to move
MAR *,AR6 ;ARP = AR6
SHIFT2 DMOV *-,AR7 ;Move all of the values in
;the VALUEIN
BANZ SHIFT2,*-,AR6 ;Data Buffer to the next
;higher address
;Outputs the FIR results and the original value
;DAC0 has the FIR results and DAC1 has the original value
LDP #0
OUT DAC0VAL,DAC0 ;DAC0 = DAC0VAL; FIR result on
;DAC channel 0
OUT DAC1VAL,DAC1 ;DAC1 = DAC1VAL; Input value
;on DAC channel 1
OUT DAC0VAL,DACUPDATE
;Update the values on the DAC
;Find the maximum value among VALUEIN and VALUEOUT for the LEDs
LACC RESET_MAX ;ACC = RESET_MAX
; Max Reset Counter
SUB #1 ;Decrement by 1
SACL RESET_MAX ;Store new value for RESET_MAX
BCND NO_RESET,GT ;If not WINDOWth value, don’t
;reset counter
SPLK #WINDOW,RESET_MAX
;Else reset the max reset counter
SPLK #0000h,MAXIN ;Reset the MAXIN value
SPLK #0000h,MAXOUT ;Reset the MAXOUT value
NO_RESET LAR AR6,#VALUEIN ;AR6 = VALUEIN; Beginning of
```

```
;Data In Buffer
LAR AR7,#127 ;AR7 = 128 - 1; Counter to find
;max value in
MAR *,AR6 ;ARP = AR6
FIND_MAXIN LACC *+,0,AR7 ;ACC = Value pointed by AR6
SUB MAXIN ;Subtract MAXIN
BCND RESUME1,LEQ ;If the value results in a
;value less than 0,
;then the value is smaller
;than MAXIN, else the
;value is larger than MAXIN
ADD MAXIN ;ACC = Value pointed by AR6
SACL MAXIN ;Store new MAXIN value
RESUME1 BANZ FIND_MAXIN,*-,AR6 ;If smaller than MAXIN,
;decrement loop counter
;(AR7), move to next value in
;buffer
LAR AR7,#127 ;Since VALUEIN buffer is
;adjacent to
;VALUEOUT buffer, only AR7
;needs to be reset
;ARP is already AR6
FIND_MAXOUT LACC *+,0,AR7 ;ACC = Value pointed by AR6
SUB MAXOUT ;Subtract MAXOUT
BCND RESUME2,LEQ ;If the value results in a
;value less than 0,
;then the value is smaller than
;MAXOUT, els
;the value is larger than
;MAXOUT
ADD MAXOUT ;ACC = Value pointed by AR6
SACL MAXOUT ;Store new MAXOUT value
RESUME2 BANZ FIND_MAXOUT,*-,AR6 ;If smaller than MAXOUT,
;dec loop counter (AR7),
;move to next value in buffer
;The following section determines if the value meets the threshold
;requirement
LDP #0 ;DP = 0; Addresses 0000h to 007Fh
;All variables used are in B2
;Need to remove the DC offset because if the FIR result is 0 it will
;equal 7ffh which is already 50% of the maximum input value
LACC MAXIN ;ACC = MAXIN
SUB #7FFh ;Subtract the DC offset
SACL DIFFIN ;DIFFIN = MAXIN - 7ffh
LACC MAXOUT ;ACC = MACOUT
SUB #7FFh ;Subtract the DC offset
SACL DIFFOUT ;DIFFOUT = MAXOUT - 7ffh
;Check if the output exceeds the middle threshold value, THRESHOLD4
LT DIFFIN ;TREG = DIFFIN
TH4 MPY THRESHOLD4 ;PREG = DIFFIN * THRESHOLD4
PAC ;ACC = PREG
SACH TEMP,1 ;TEMP = ACC*2; Shift to remove
;extra sign bit
LACC TEMP ;ACC = TEMP
SUB DIFFOUT ;Subtract DIFFOUT
BCND ABOVE4,LT ;If DIFFOUT is greater than
;TEMP, then the FIR result is
;greater than VALUEIN * THRESHOLD4,
;else, it is below THRESHOLD4 value
;Output is below THRESHOLD4. Check if above THRESHOLD2
BELOW4 LT DIFFIN
TH2 MPY THRESHOLD2
PAC
SACH TEMP,1
LACC TEMP
SUB DIFFOUT
BCND ABOVE2,LT
;Output is below THRESHOLD4 & THRESHOLD2. Check if above THRESHOLD1
BELOW2 LT DIFFIN
TH1 MPY THRESHOLD1
PAC
SACH TEMP,1
LACC TEMP
SUB DIFFOUT
BCND ABOVE1,LT
```

Source: Application Report SPRA414, Texas Instruments, 1999

Cont. Cont. Cont.

```
;Output is below THRESHOLD4, THRESHOLD2, & THRESHOLD1. Turn off LEDS
BELOW1 SPLK #0000h,LEDSOUT
B OUTLEDS
;Output is below THRESHOLD4, THRESHOLD2, but above THRESHOLD1. Turn
;on DS1
ABOVE1 SPLK #0001h,LEDSOUT
B OUTLEDS
;Output is below THRESHOLD4, but above THRESHOLD2. Check if above
;THRESHOLD3
ABOVE2 LT DIFFIN
TH3 MPY THRESHOLD3
PAC
SACH TEMP,1
LACC TEMP
SUB DIFFOUT
BCND ABOVE3,LT
;Output is below THRESHOLD4 and THRESHOLD3, but above THRESHOLD2.
;Turn on DS1-DS2
BELOW3 SPLK #0003h,LEDSOUT
B OUTLEDS
;Output is below THRESHOLD4, but above THRESHOLD3 and THRESHOLD2.
;Turn on DS1-DS3
ABOVE3 SPLK #0007h,LEDSOUT
B OUTLEDS
;Output is above THRESHOLD4. Check if above THRESHOLD6
ABOVE4 LT DIFFIN
TH6 MPY THRESHOLD6
PAC
SACH TEMP,1
LACC TEMP
SUB DIFFOUT
BCND ABOVE6,LT
;Output is above THRESHOLD4, but below THRESHOLD6. Check if above
;THRESHOLD5.
BELOW6 LT DIFFIN
TH5 MPY THRESHOLD5
PAC
SACH TEMP,1
LACC TEMP
SUB DIFFOUT
BCND ABOVE5,LT
;Output is above THRESHOLD4, but below THRESHOLD6 & THRESHOLD5. Turn
;on DS1-DS4
BELOW5 SPLK #000Fh,LEDSOUT
B OUTLEDS
;Output is above THRESHOLD4 & THRESHOLD5, but below THRESHOLD6.
;Turn on DS1-DS5
ABOVE5 SPLK #001Fh,LEDSOUT
B OUTLEDS
;Output is above THRESHOLD4 & THRESHOLD6. Check if above THRESHOLD8.
ABOVE6 LT DIFFIN
TH8 MPY THRESHOLD8
PAC
SACH TEMP,1
LACC TEMP
SUB DIFFOUT
BCND ABOVE8,LT
;Output is above THRESHOLD4 & THRESHOLD6, but below THRESHOLD8.
;Check if above THRESHOLD7.
BELOW8 LT DIFFIN
TH7 MPY THRESHOLD7
PAC
SACH TEMP,1
LACC TEMP
SUB DIFFOUT
BCND ABOVE7,LT
```

```
;Output is above THRESHOLD4 & THRESHOLD6, but below THRESHOLD8 &
;THRESHOLD7. Turn on DS1-DS6
BELOW7 SPLK #003Fh,LEDSOUT
B OUTLEDS
;Output is above THRESHOLD4, THRESHOLD6, & THRESHOLD7, but below
;THRESHOLD8. Turn on ;DS1-DS7
ABOVE7 SPLK #007Fh,LEDSOUT
B OUTLEDS
;Output is above THRESHOLD4, THRESHOLD6, & THRESHOLD8. Turn on
;DS1-DS8
ABOVE8 SPLK #00FFh,LEDSOUT
OUTLEDS OUT LEDSOUT,LEDS ;Turn on the LEDS
RESTART_ADC MAR *,AR2 ;ARP = AR2
LACC * ;ACC = ADCTRL1
ADD #1h ;Set bit to restart the ADC
SACL * ;Start converting next value
LDP #232
LACC EVIFRA ;Clear the flag register of
;Event Manager
SACL EVIFRA
CLRC INTM ;ENABLE INTERRUPTS
RET ;Return to main line
;===================================================================
; I S R - PHANTOM
;
; Description: Dummy ISR, used to trap spurious interrupts.
;
; Modifies:
;
; Last Update: 16-06-95
;===================================================================
PHANTOM B PHANTOM
```

Source: Application Report SPRA414, Texas Instruments, 1999

DSP Implementation (Excerpt)

```
;The following section determines if the value meets the threshold
;requirement
LDP #0 ;DP = 0; Addresses 0000h to 007Fh
;All variables used are in B2
;Need to remove the DC offset because if the FIR result is 0 it will
;equal 7ffh which is already 50% of the maximum input value
LACC MAXIN ;ACC = MAXIN
SUB #7FFh ;Subtract the DC offset
SACL DIFFIN ;DIFFIN = MAXIN - 7ffh
LACC MAXOUT ;ACC = MACOUT
SUB #7FFh ;Subtract the DC offset
SACL DIFFOUT ;DIFFOUT = MAXOUT - 7ffh
;Check if the output exceeds the middle threshold value, THRESHOLD4
LT DIFFIN ;TREG = DIFFIN
TH4 MPY THRESHOLD4 ;PREG = DIFFIN * THRESHOLD4
PAC ;ACC = PREG
SACH TEMP,1 ;TEMP = ACC*2; Shift to remove
;extra sign bit
LACC TEMP ;ACC = TEMP
SUB DIFFOUT ;Subtract DIFFOUT
BCND ABOVE4,LT ;If DIFFOUT is greater than
;TEMP, then the FIR result is
;greater than VALUEIN * THRESHOLD4,
;else, it is below THRESHOLD4 value
;Output is below THRESHOLD4. Check if above THRESHOLD2
BELOW4 LT DIFFIN
TH2 MPY THRESHOLD2
PAC
…
```

Source: Application Report SPRA414, Texas Instruments, 1999

Streaming in Assembly Code

- Pro: Fast!
- Con:
  - Extremely tedious, costly, and error-prone
  - Not portable between architectures
  - Very hard to maintain
    - Move center frequency from 500 Hz to 1200 Hz?
    - According to TI, in the conventional design flow:
      - Redesign filter in MATLAB
      - Cut-and-paste values to EXCEL
      - Recalculate the coefficients
      - Update the assembly code
    - Will address this issue again later today, in section on Domain Specific Optimizations

Stream Languages to the Rescue

- Goals of a stream language:
  - Expose parallelism
  - Expose communication patterns
  - Encapsulate common idioms
    - Autonomous filters
    - Circular buffers

➡ Improve BOTH performance and programmer productivity

- Vision:

  A unified, high-level programming environment that achives the performance of hand-coded assembly

Stream Programming Models

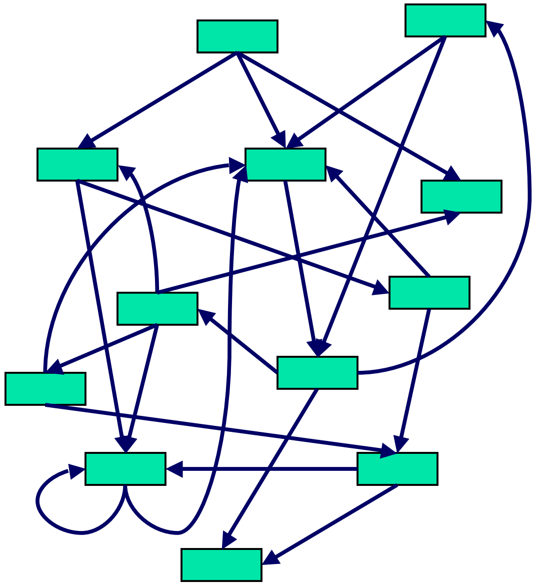
- Prototyping environments
- Conventional languages
  - Object Oriented
  - Procedural
  - Assembly
- Stream languages
  - StreamIt
  - Brook
  - Cg

The StreamIt Language

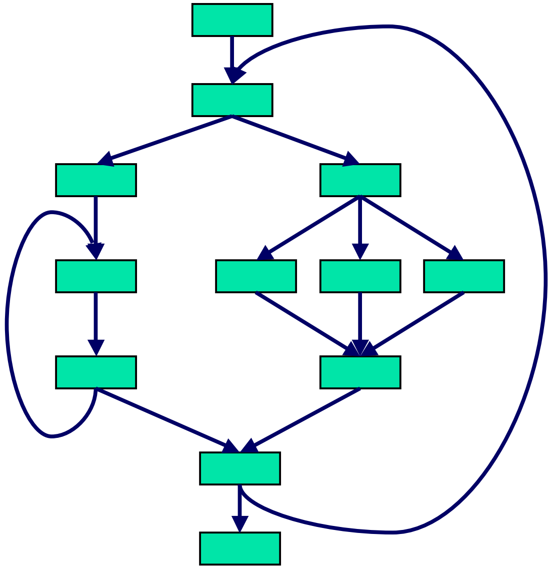
- A high-level, architecture-independent language for streaming applications
- Current focus domain: Synchronous Dataflow
- Contributions
  - Language Design, Structured Streams, Buffer Management *(CC 2002)*
  - Exploiting Wire-Exposed Architectures *(ASPLOS 2002)*
  - Scheduling of Static Dataflow Graphs *(LCTES 2003)*
  - Domain Specific Optimizations *(PLDI 2003)*

Representing Streams

- Conventional wisdom: streams are graphs
  - Graphs have no simple textual representation
  - Graphs are difficult to analyze and optimize
- Insight: stream programs have structure



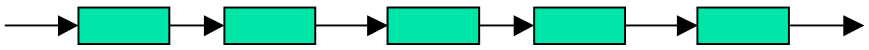
unstructured



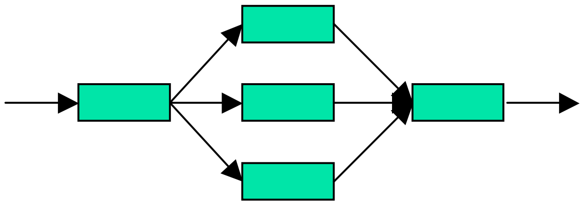
structured

Structured Streams

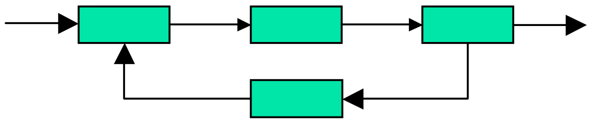
- Hierarchical structures:
  - Pipeline



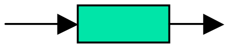
  - SplitJoin



  - Feedback Loop

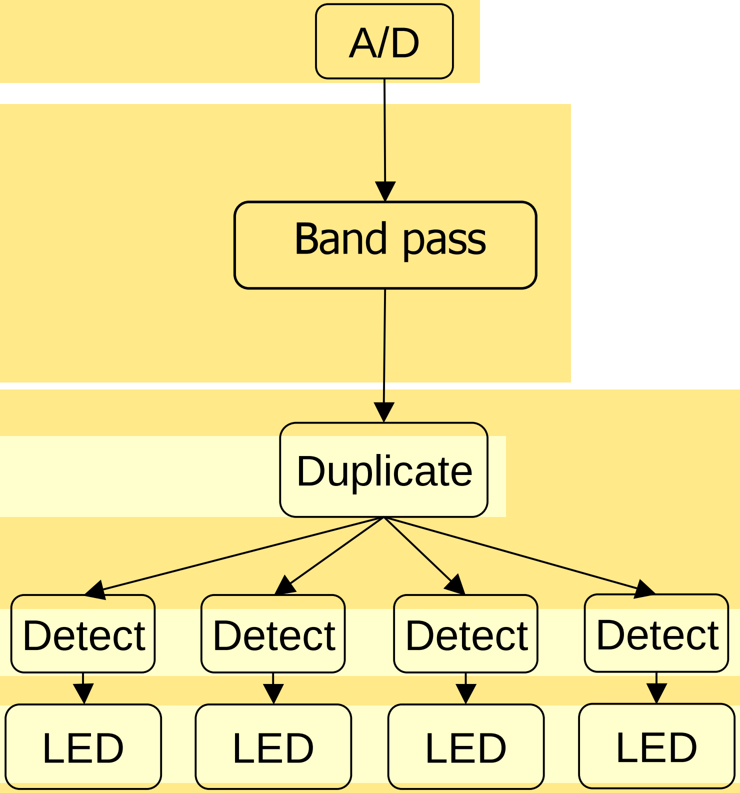


- Basic programmable unit: Filter



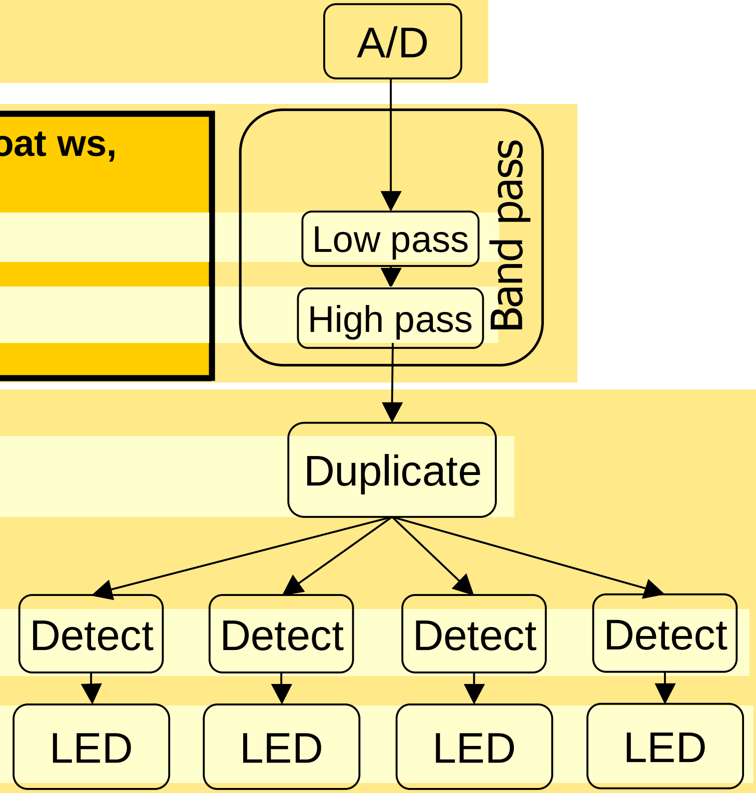
Freq band detection in StreamIt

```
void->void pipeline FrequencyBand {
  float sFreq = 4000;
  float cFreq = 500/(sFreq*2*pi);
  float  wFreq = 100/(sFreq*2*pi);
  add D2ASource(sFreq);

  add BandPassFilter(1, cFreq-wFreq,
                        cFreq+wFreq, 100);

  add splitjoin {
    split duplicate;
    for (int i=0; i<4; i++) {
      add pipeline {
        add Detector(i/4);

        add LEDOutput(i);
      }
    }
    join roundrobin(0);
  }
}
```



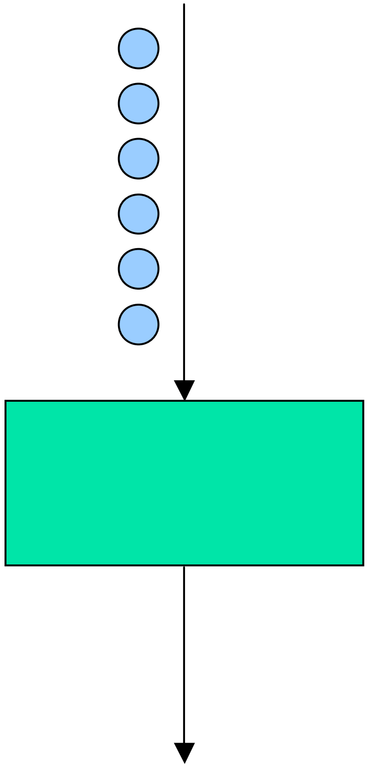
Freq band detection in StreamIt

```
void->void pipeline FrequencyBand {
  float sFreq = 4000;
  float cFreq = 500/(sFreq*2*pi);
  float  wFreq = 100/(sFreq*2*pi);
  add D2ASource(sFreq);

  float->float pipeline BandPassFilter(float gain, float ws,
                                       float wp, int num) {
        add LowPassFilter(1, wp, num);
        add HighPassFilter(gain, ws, num);
  }

  add splitjoin {
    split duplicate;
    for (int i=0; i<4; i++) {
      add pipeline {
        add Detector(i/4);
        add LEDOutput(i);
      }
    }
    join roundrobin(0);
  }
}
```



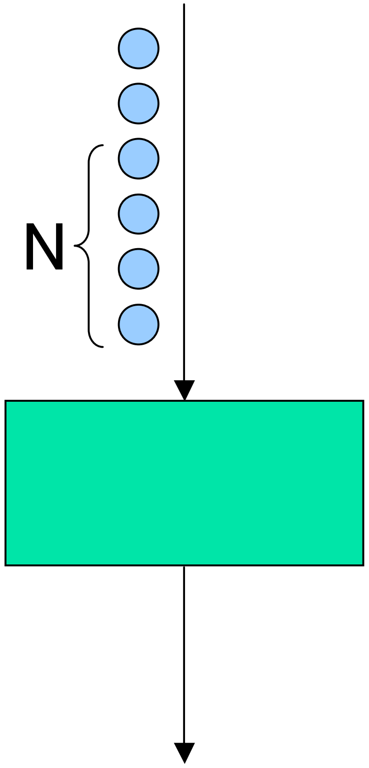
Filter Example: LowPassFilter

```
float->float filter LowPassFilter (int N, float freq) {
    float[N] weights;

    init {
        weights = calcWeights(N, freq);
    }

    work push 1 pop 1 peek N {
        float result = 0;
        for (int i=0; i<weights.length; i++) {
            result += weights[i] * peek(i);
        }
        push(result);
        pop();
    }
}
```



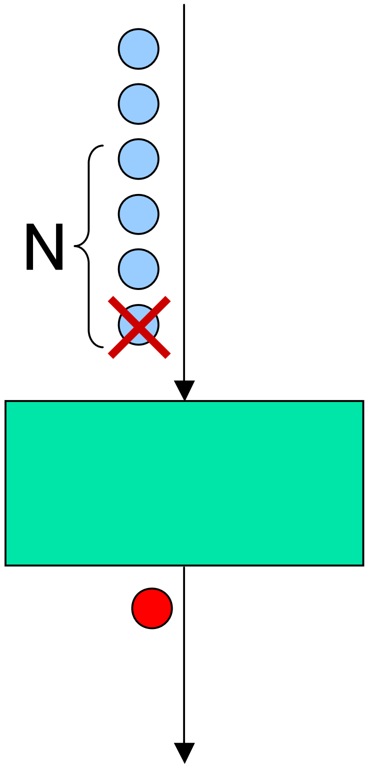
Filter Example: LowPassFilter

```
float->float filter LowPassFilter (int N, float freq) {
    float[N] weights;

    init {
        weights = calcWeights(N, freq);
    }

    work push 1 pop 1 peek N {
        float result = 0;
        for (int i=0; i<weights.length; i++) {
            result += weights[i] * peek(i);
        }
        push(result);
        pop();
    }
}
```



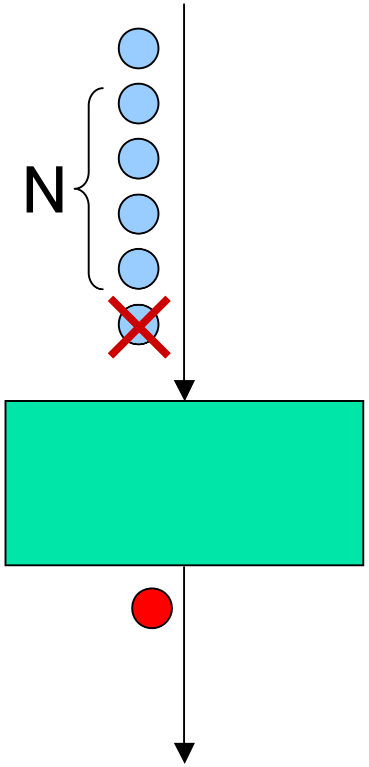
Filter Example: LowPassFilter

```
float->float filter LowPassFilter (int N, float freq) {
   float[N] weights;

   init {
      weights = calcWeights(N, freq);
   }

   work push 1 pop 1 peek N {
      float result = 0;
      for (int i=0; i<weights.length; i++) {
         result += weights[i] * peek(i);
      }
      push(result);
      pop();
   }
}
```



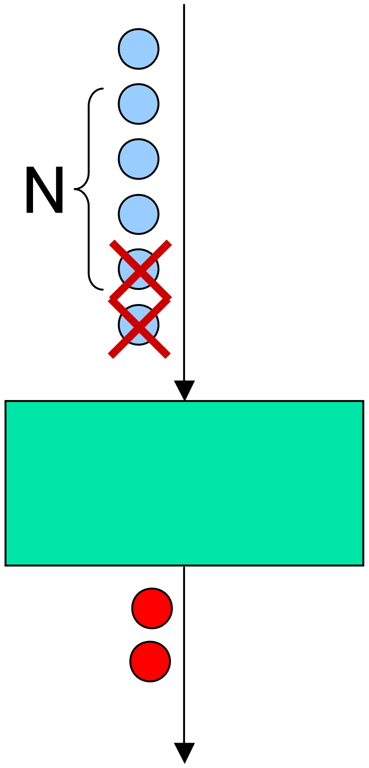
Filter Example: LowPassFilter

```
float->float filter LowPassFilter (int N, float freq) {
    float[N] weights;

    init {
        weights = calcWeights(N, freq);
    }

    work push 1 pop 1 peek N {
        float result = 0;
        for (int i=0; i<weights.length; i++) {
            result += weights[i] * peek(i);
        }
        push(result);
        pop();
    }
}
```



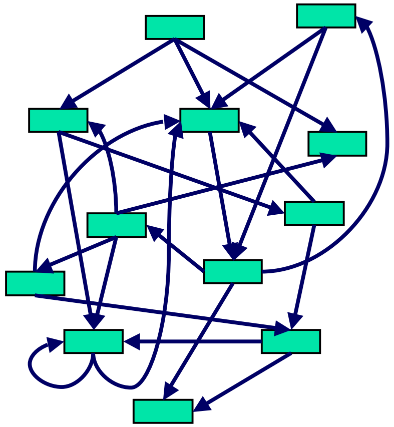
Filter Example: LowPassFilter

```
float->float filter LowPassFilter (int N, float freq) {
    float[N] weights;

    init {
        weights = calcWeights(N, freq);
    }

    work push 1 pop 1 peek N {
        float result = 0;
        for (int i=0; i<weights.length; i++) {
            result += weights[i] * peek(i);
        }
        push(result);
        pop();
    }
}
```

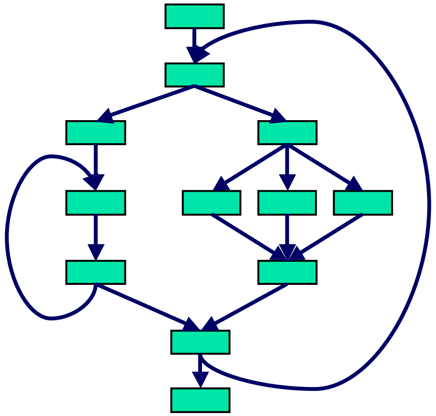


Why Structured Streams?

- Compare to structured control flow



GOTO statements



If / else / for statements

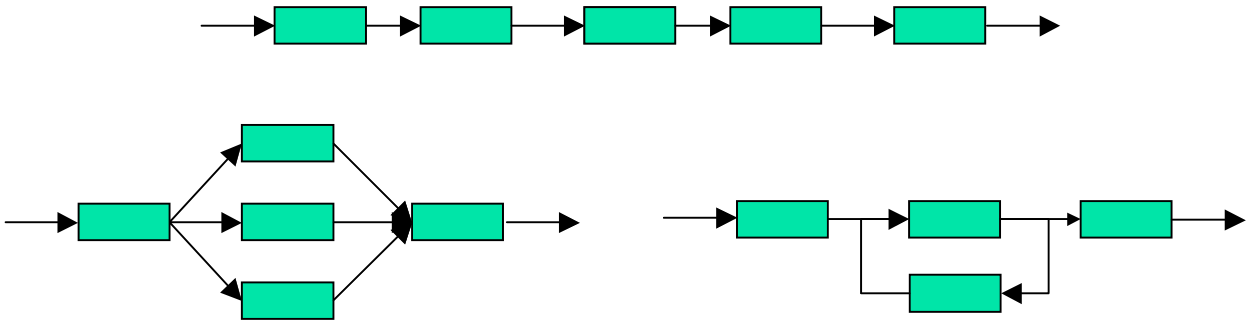
- Tradeoff:

  PRO: - more robust - more analyzable

  CON: - “restricted” style of programming

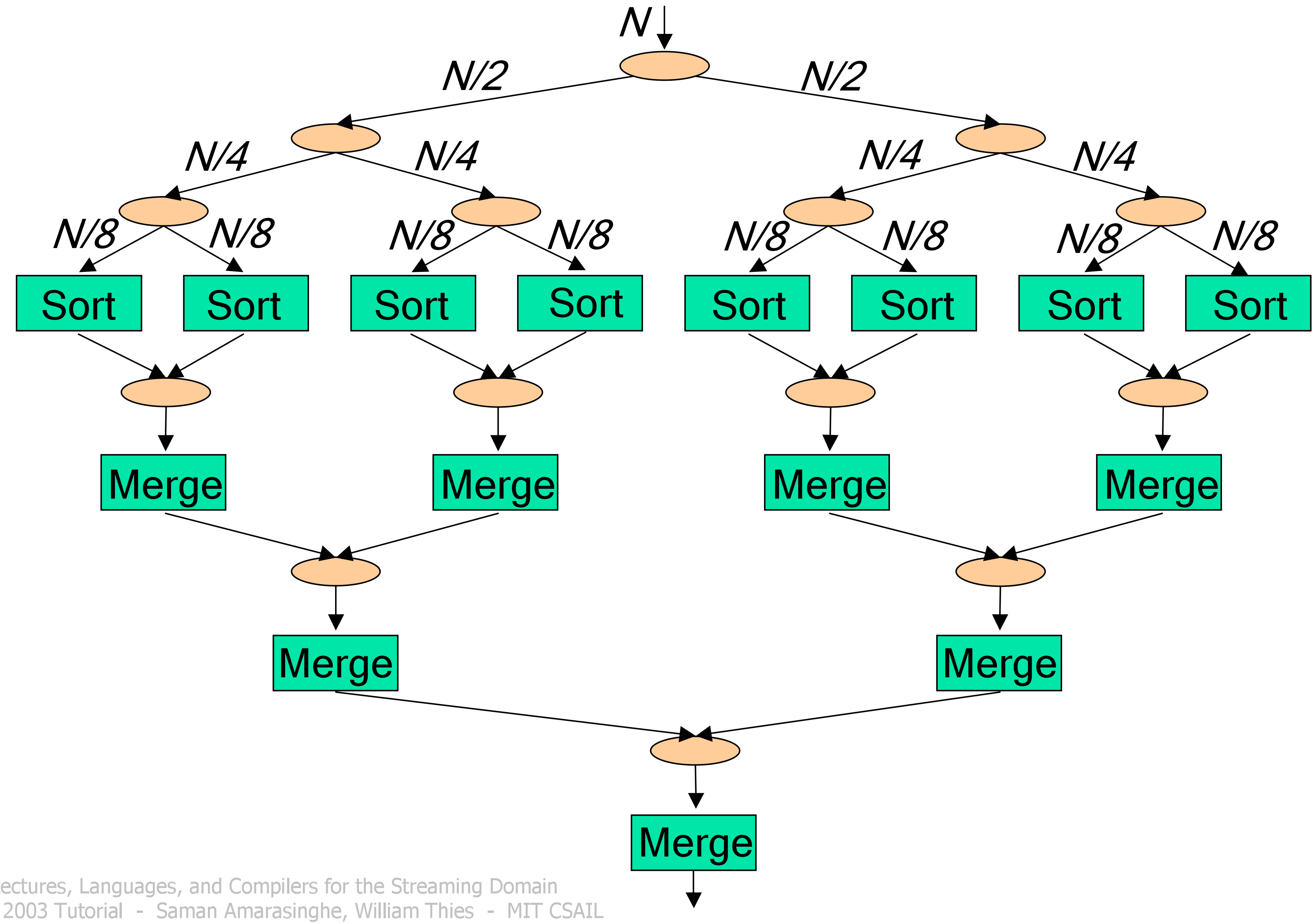
Structure Helps Programmers

- Modules are hierarchical and composable
  - Each structure is single-input, single-output



- Encapsulates common idioms
- Good textual representation
  - Enables parameterizable graphs

N-Element Merge Sort (3-level)

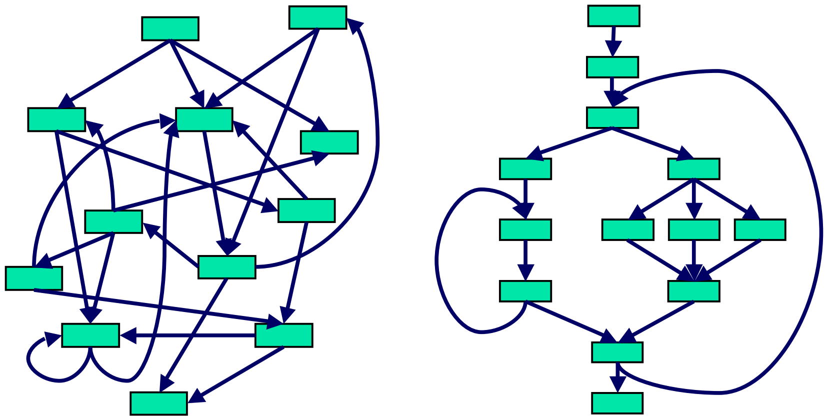


N-Element Merge Sort (K-level)

```
pipeline MergeSort (int N, int K) {
    if (K==1) {
        add Sort(N);
    } else {
        add splitjoin {
            split roundrobin;
            add MergeSort(N/2, K-1);
            add MergeSort(N/2, K-1);
            joiner roundrobin;
        }
    }
    add Merge(N);
  }
}
```

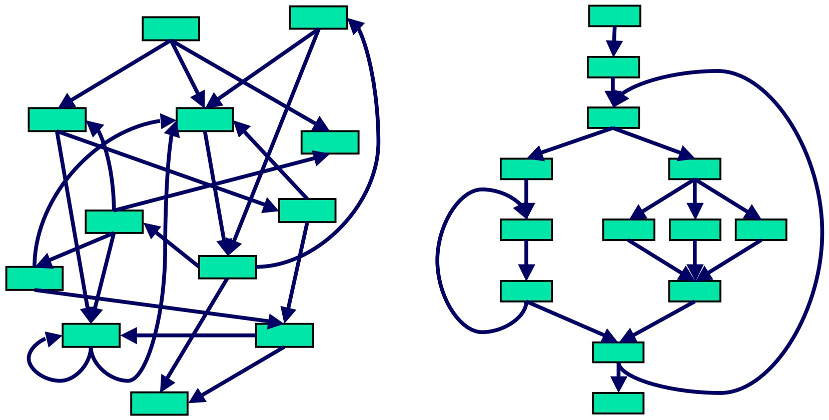
Structure Helps Compilers

- Enables local, hierarchical analyses
  - Scheduling
  - Optimization
  - Parallelization
  - Load balancing

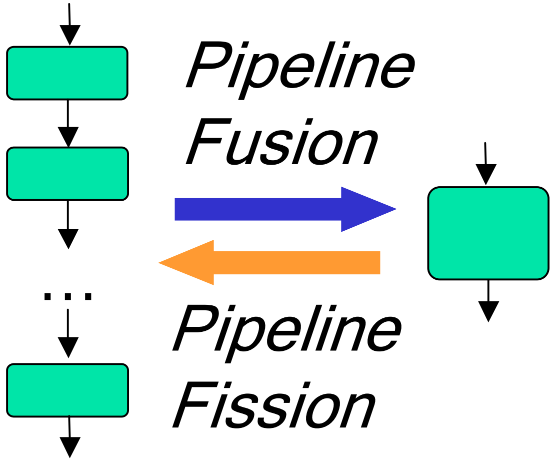


Structure Helps Compilers

- Enables local, hierarchical analyses
  - Scheduling
  - Optimization
  - Parallelization
  - Load balancing

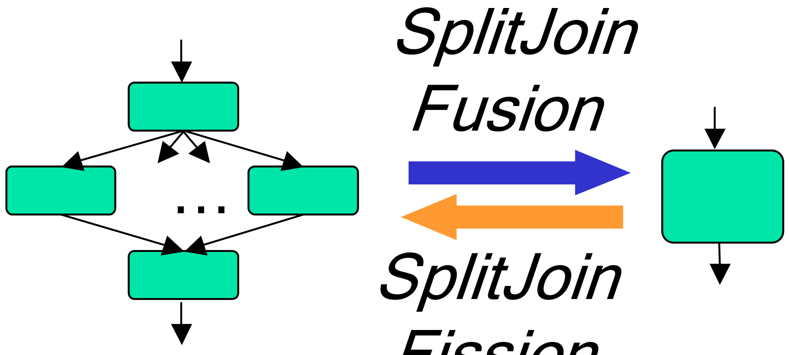


- Examples:



Pipeline Fusion

Pipeline Fission

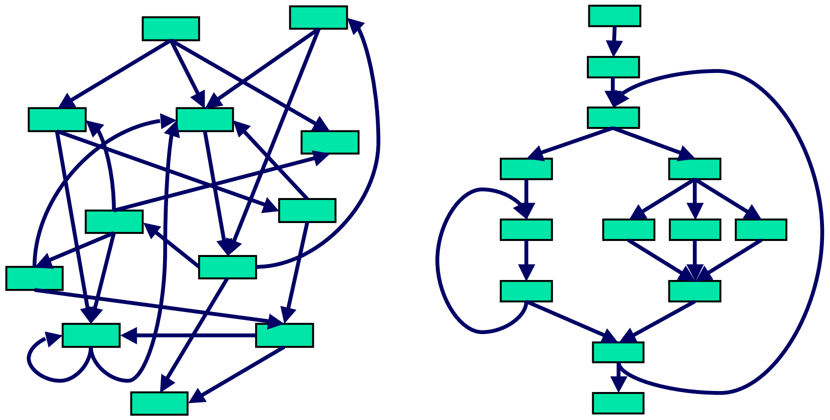


SplitJoin Fusion

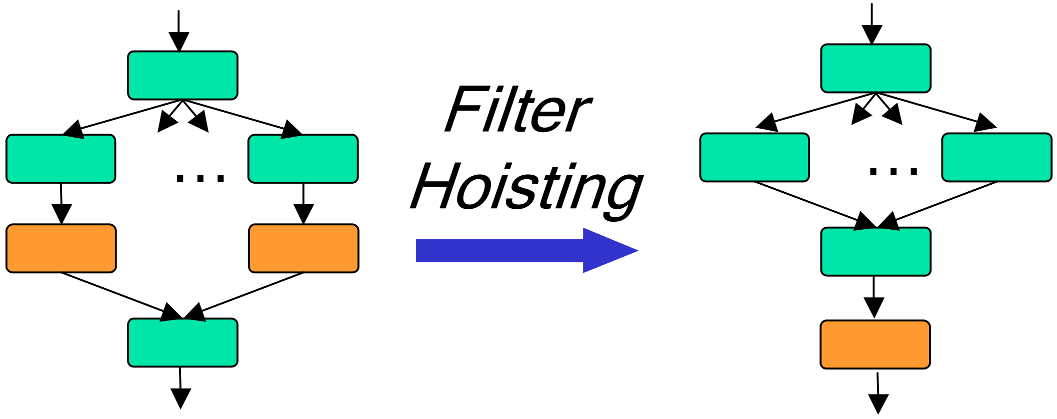
SplitJoin Fission

Structure Helps Compilers

- Enables local, hierarchical analyses
  - Scheduling
  - Optimization
  - Parallelization
  - Load balancing

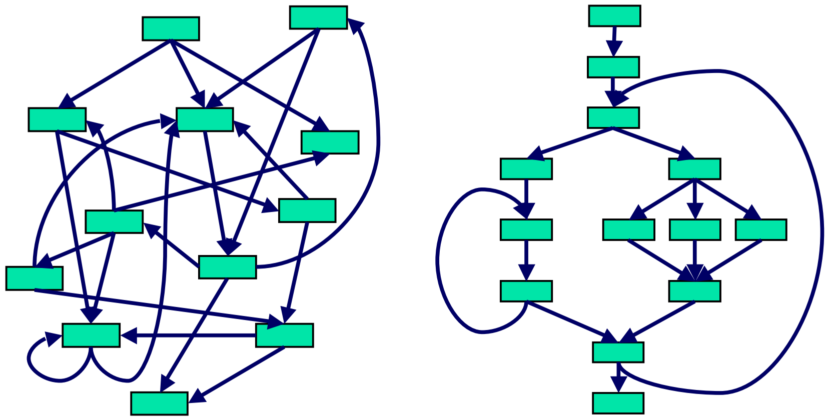


- Examples:



Structure Helps Compilers

- Enables local, hierarchical analyses
  - Scheduling
  - Optimization
  - Parallelization
  - Load balancing



- Disallows non-sensical graphs
- Simplifies separate compilation
  - All blocks single-input, single-output

CON: Restricts Coding Style

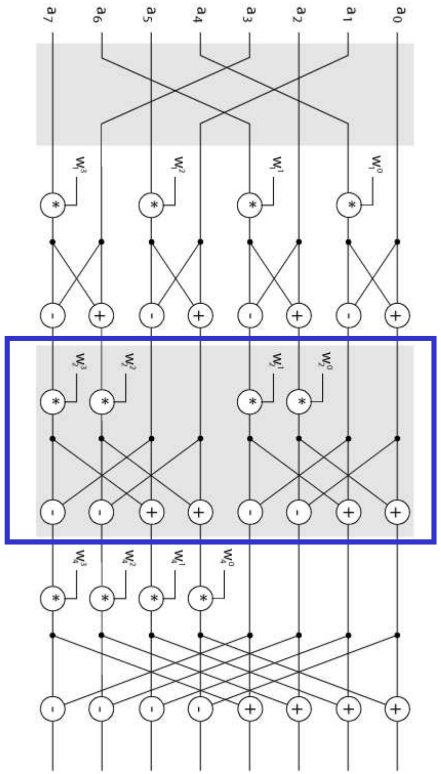
- Some graphs need to be re-arranged
- Example: FFT

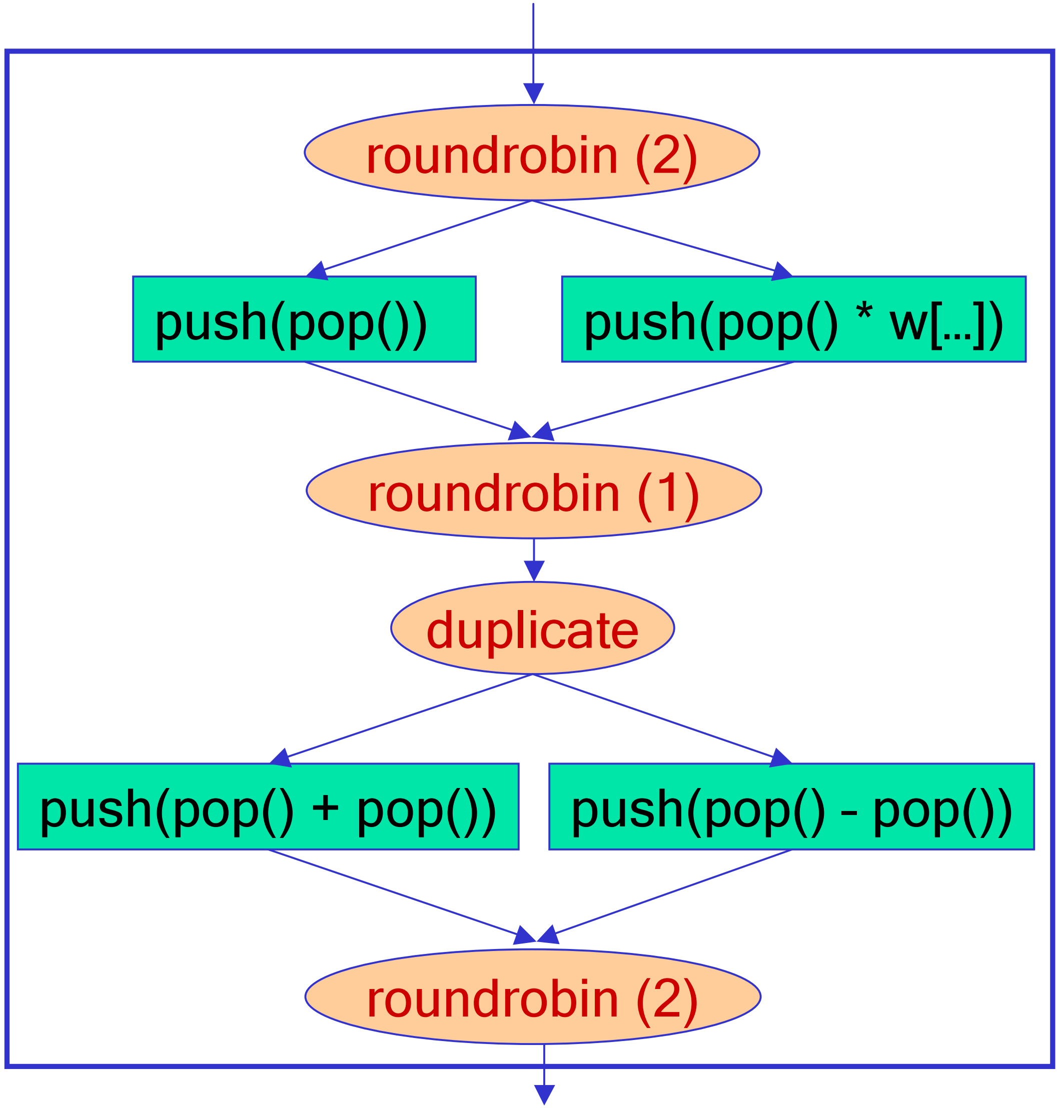
Bit-reverse order

Butterfly (2 way)

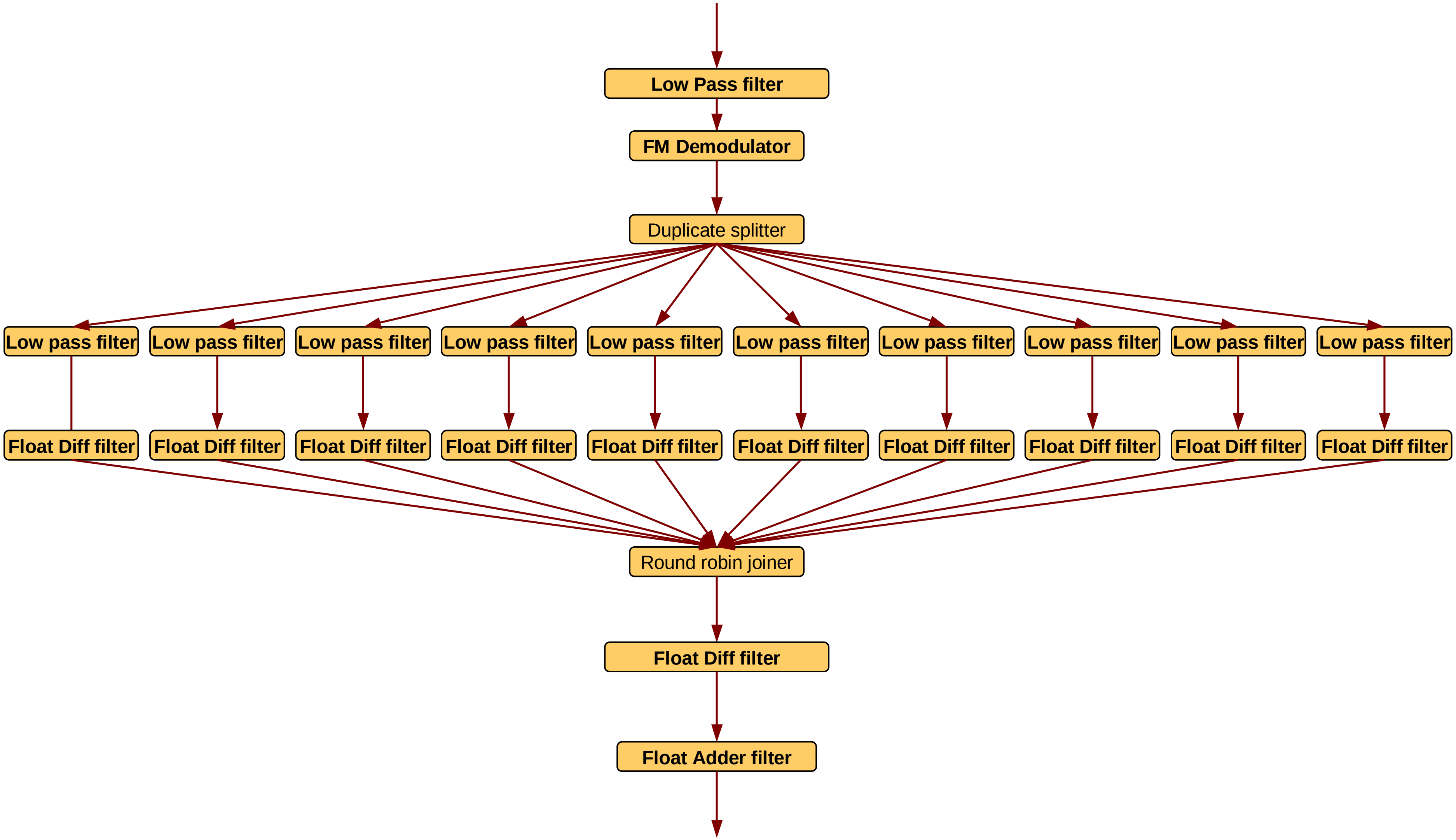
Butterfly (4 way)

Butterfly (8 way)

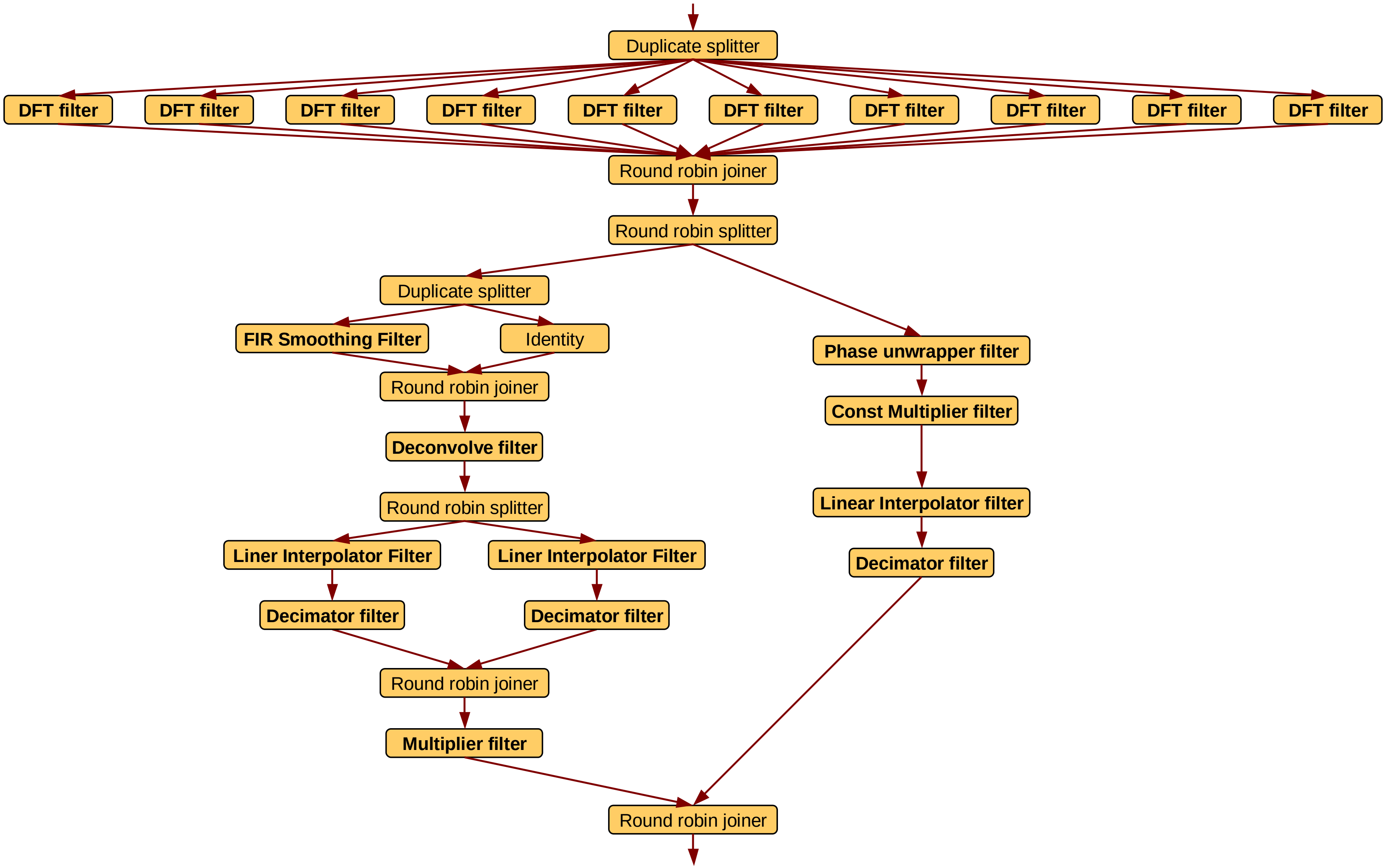




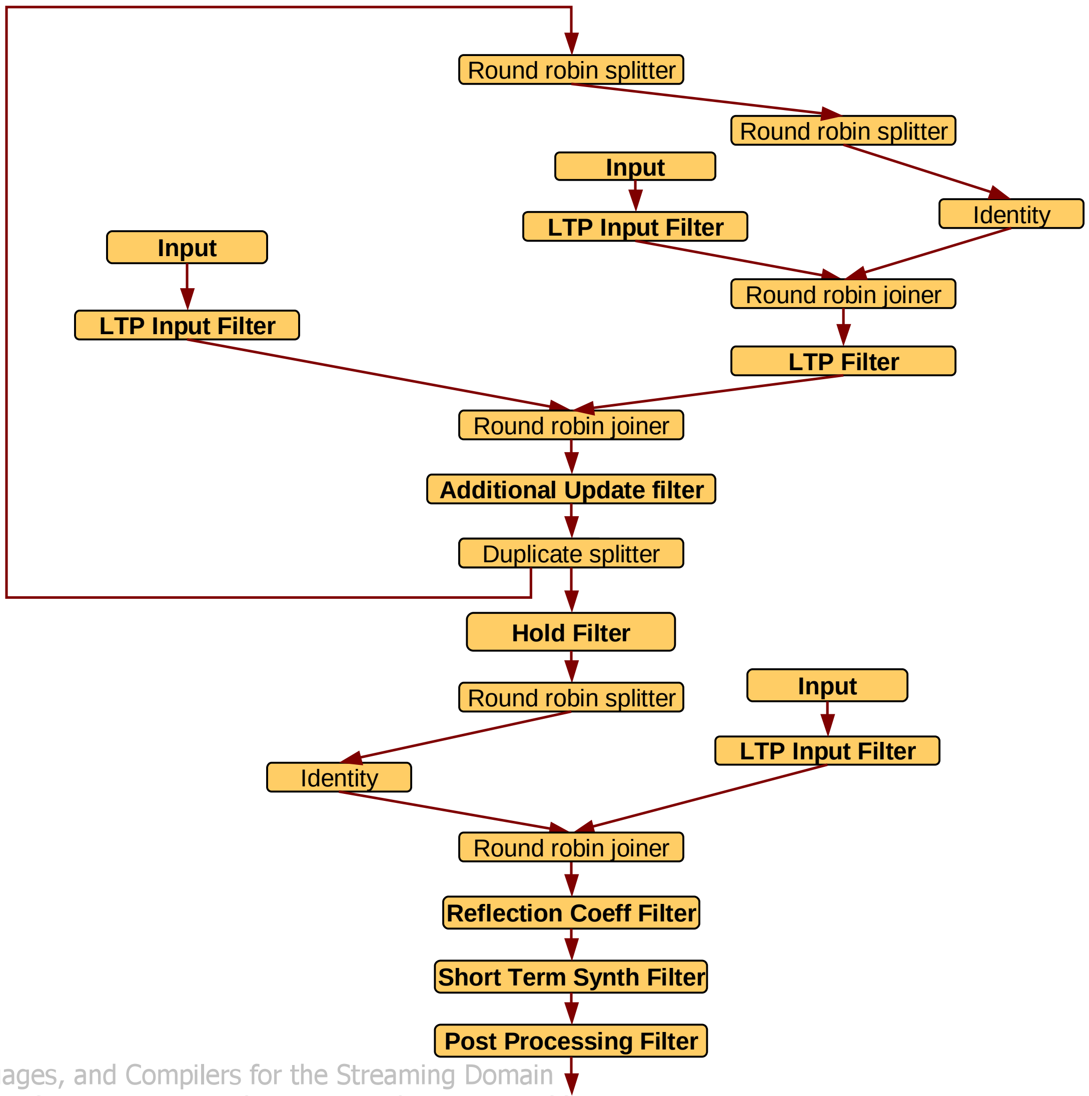
Example: FM Radio with Equalizer



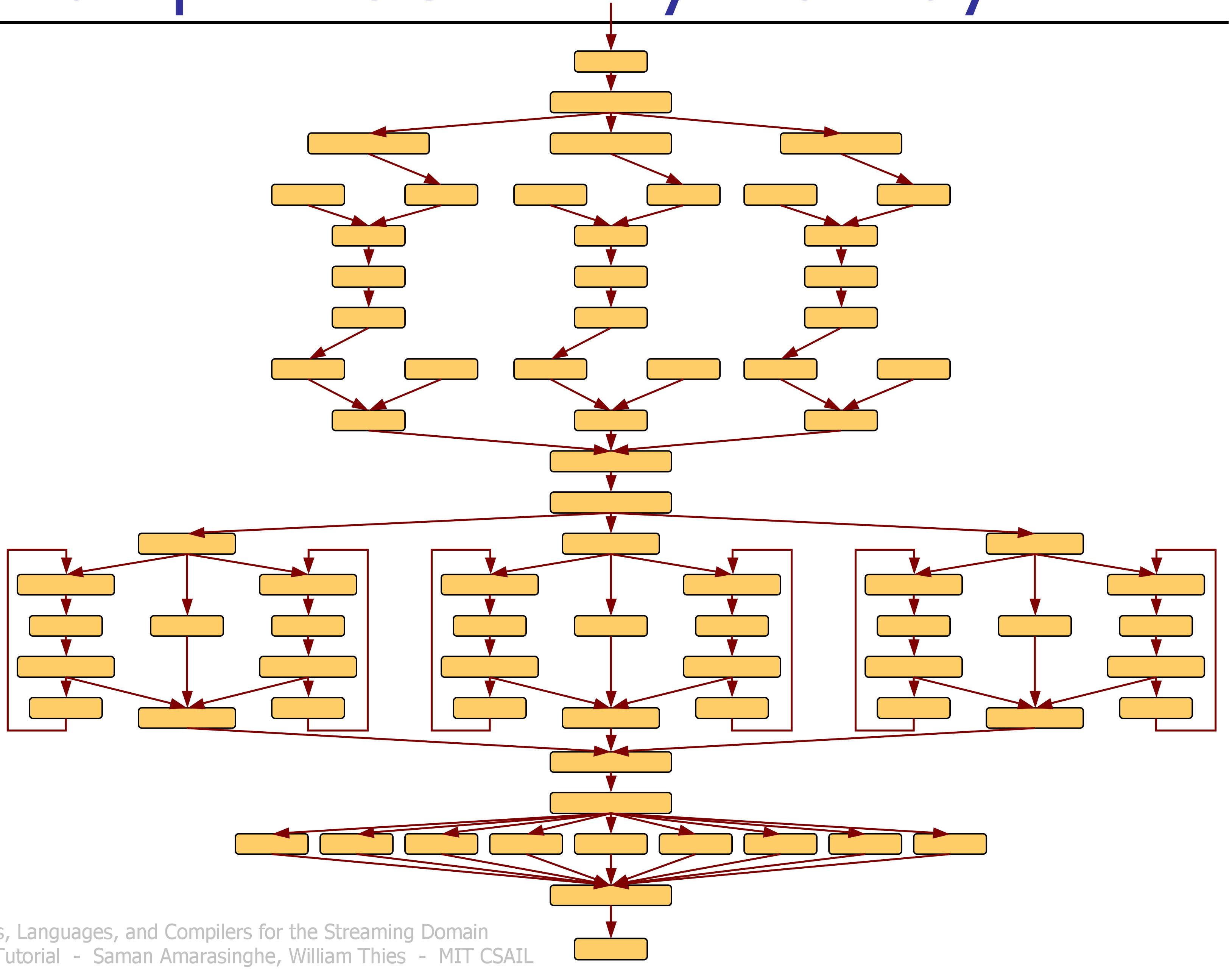
Example: Vocoder



Example: GSM decoder

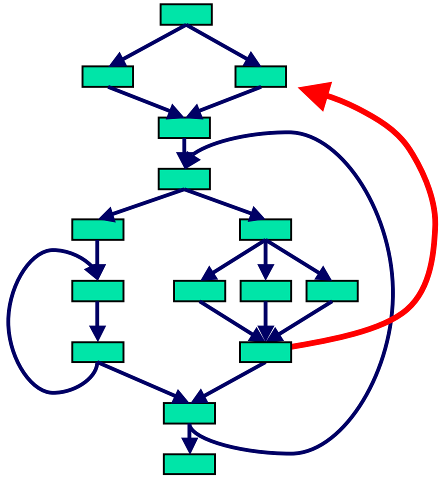


Example: 3GPP Physical Layer



Control Messages

- Structures for regular, high-bandwidth data
- But also need a control mechanism for irregular, low-bandwidth events



  - Change volume on a cell phone
  - Initiate handoff of stream
  - Adjust network protocol

Supporting Control Messages

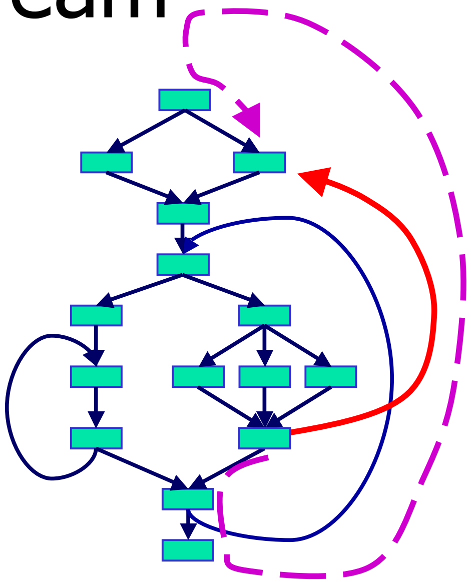
- Option 1: Embed message in stream

  PRO: - message arrives with data

  CON: - complicates filter code

  - complicates structure

  - runtime overhead

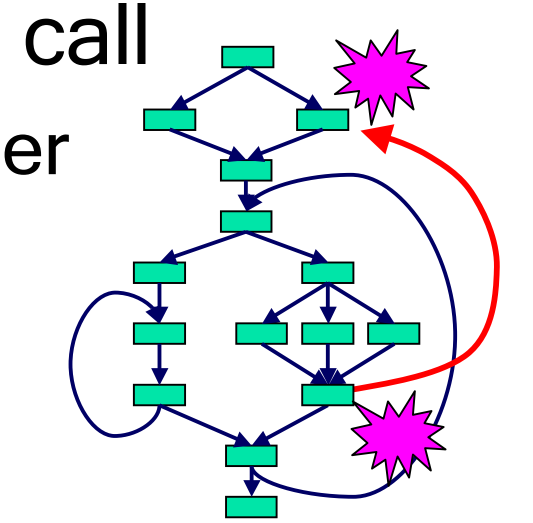


- Option 2: Synchronous method call

  PRO: - delivery transparent to user

  CON: - timing is unclear

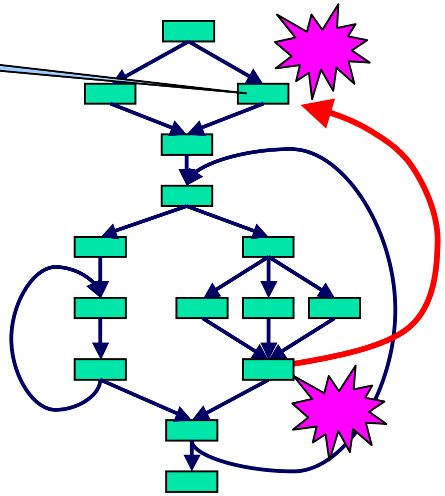
  - limits parallelism



StreamIt Messaging System

- Looks like method call, but semantics differ

```
void raiseVolume(int v)
  myVolume += v;
}
```

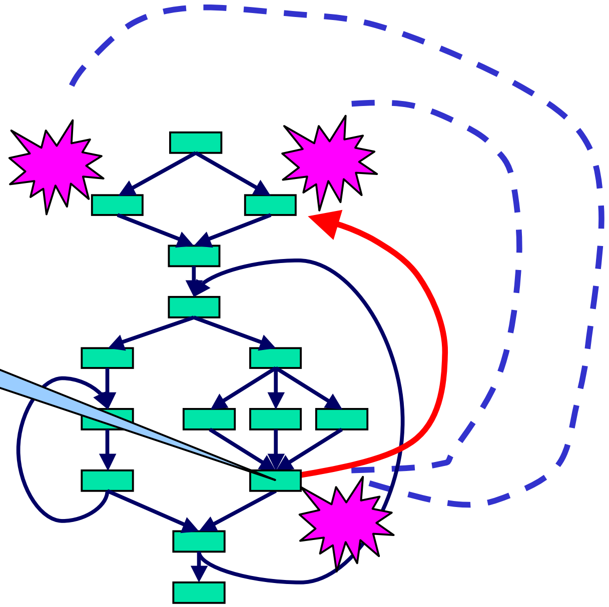


  - No return value
  - Asynchronous delivery
  - Can broadcast to multiple targets

StreamIt Messaging System

- Looks like method call, but semantics differ

```
TargetFilter x;
work {
  …
  if (lowVolume())
     x.raiseVolume(10) at 100;
}
```

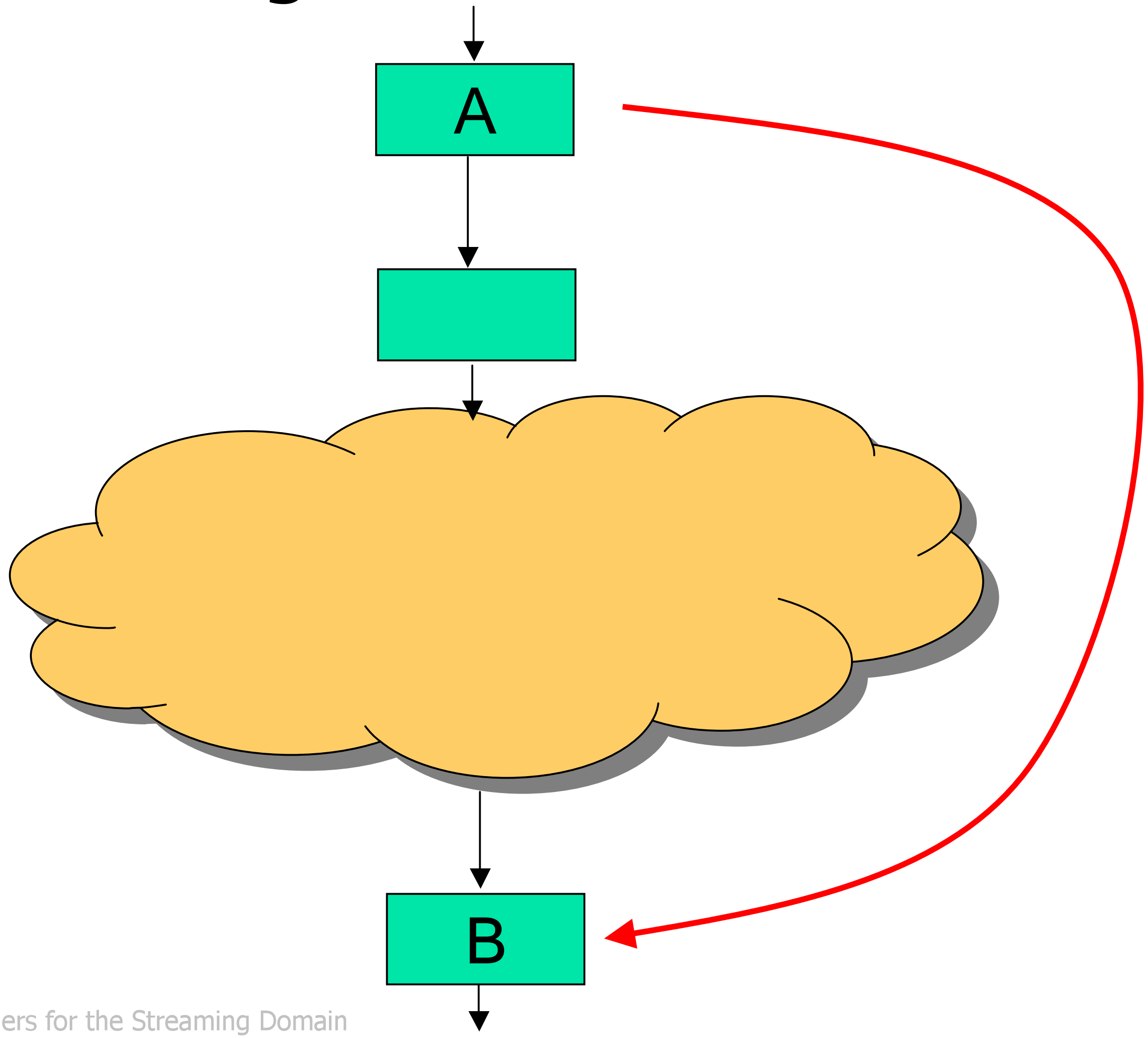


  - No return value
  - Asynchronous delivery
  - Can broadcast to multiple targets

- Timed relative to data
  - User gains precision; compiler gains flexibility

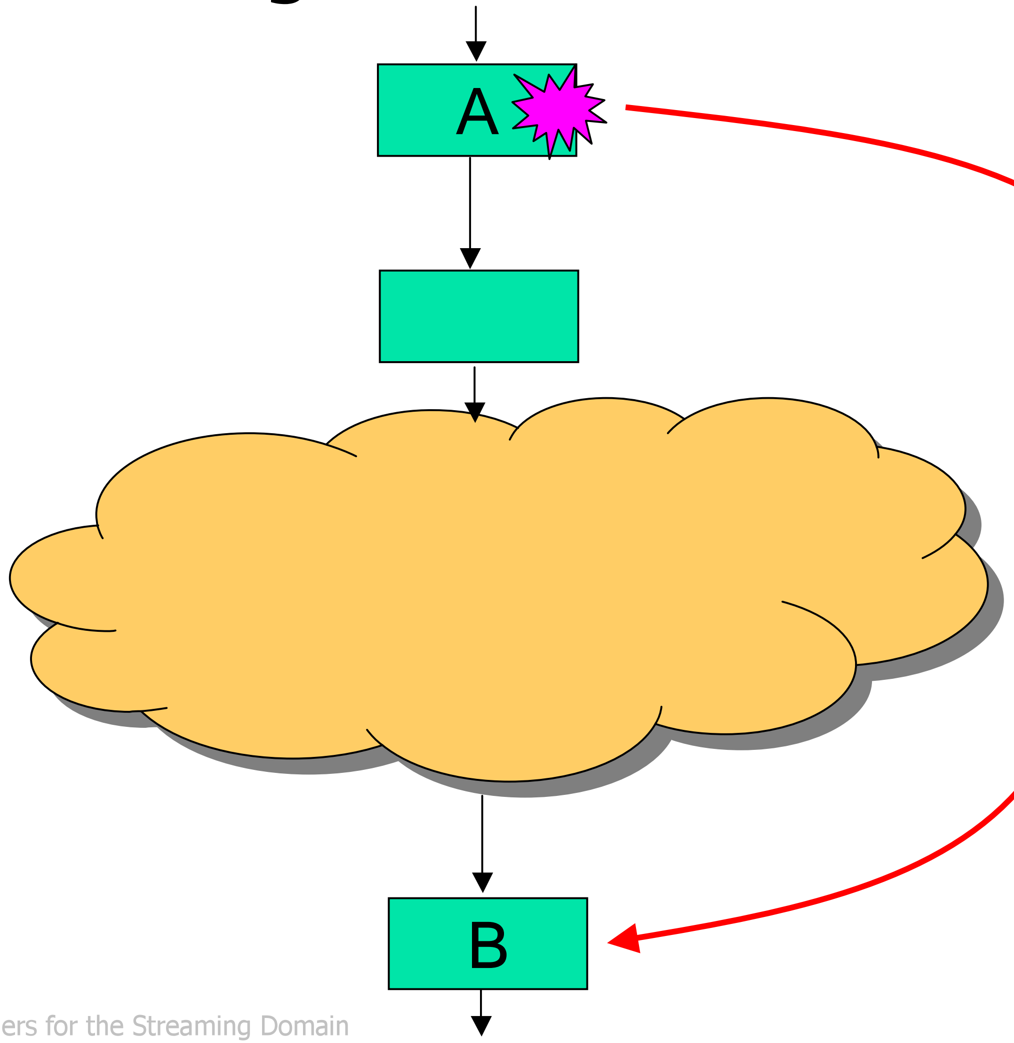
Message Timing

- A sends message to B with zero latency



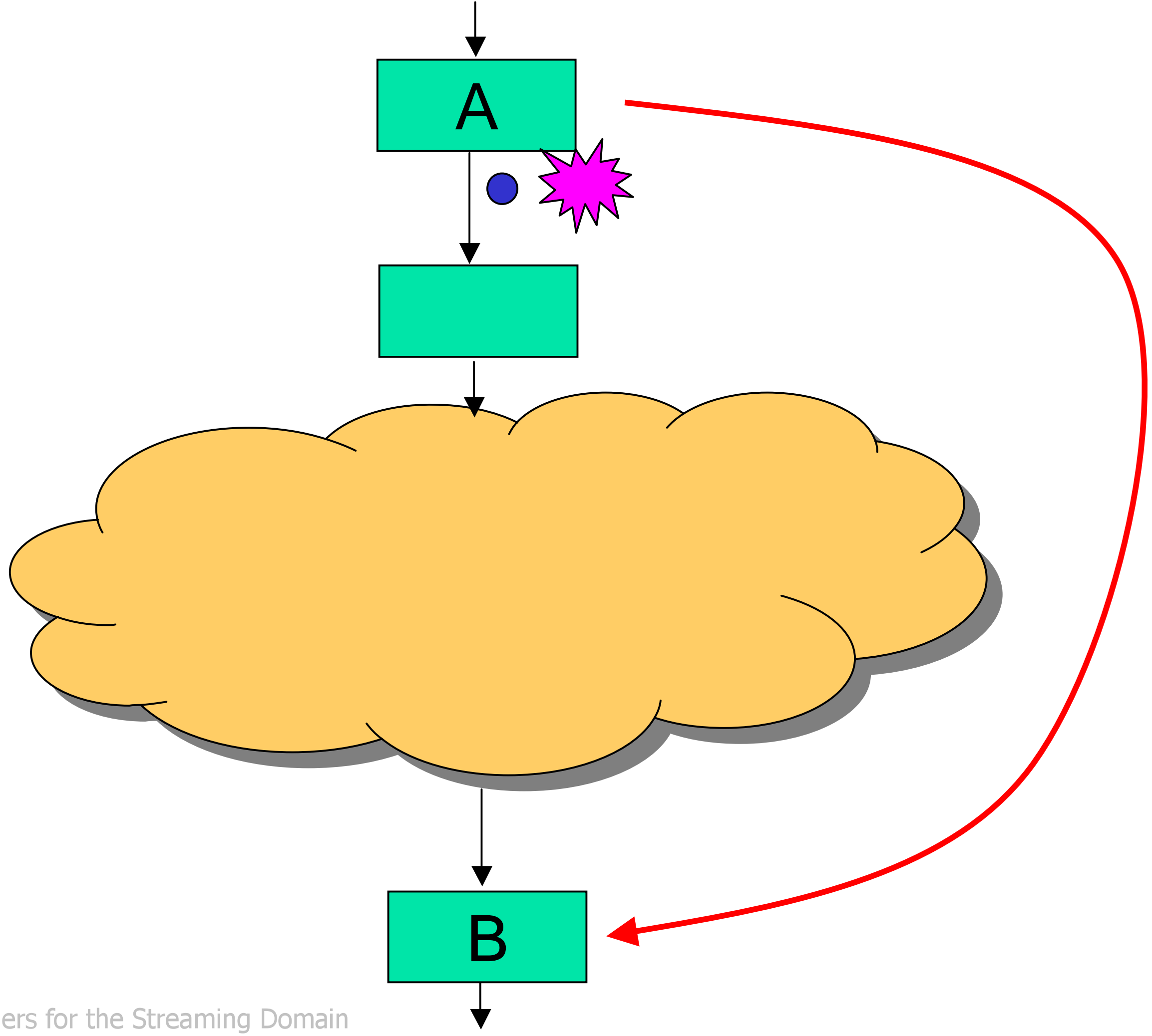
Message Timing

- A sends message to B with zero latency



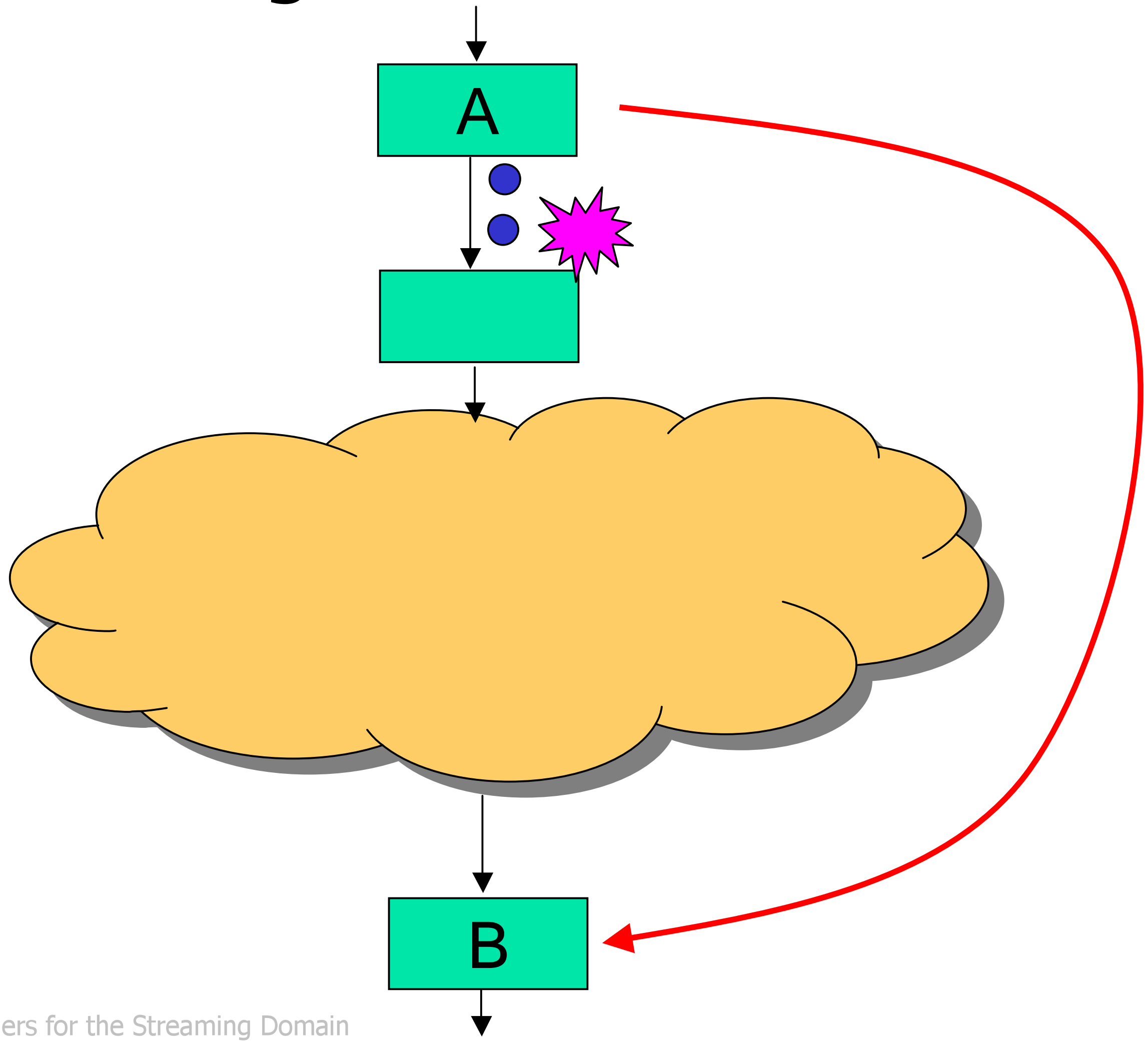
Message Timing

- A sends message to B with zero latency



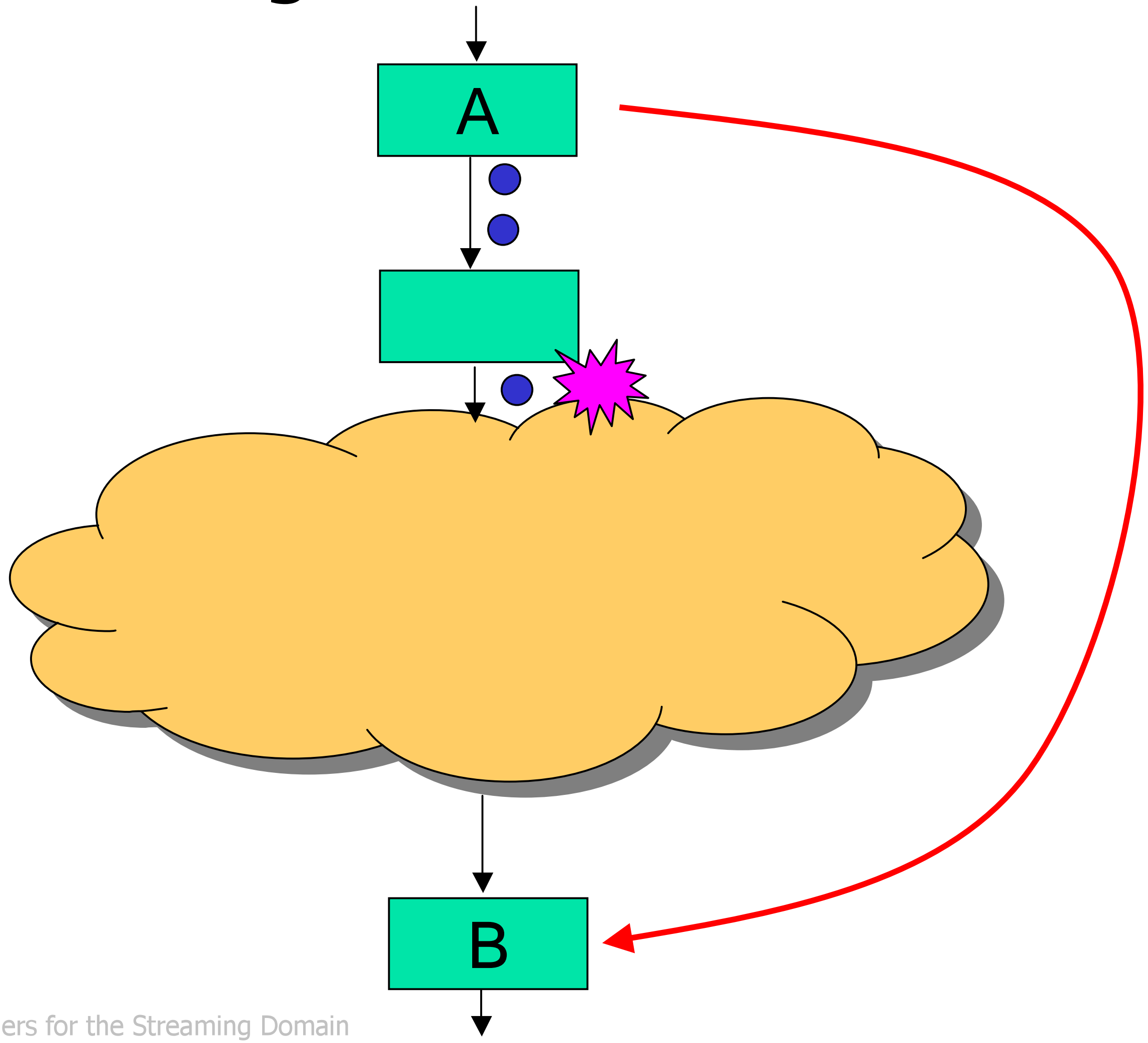
Message Timing

- A sends message to B with zero latency



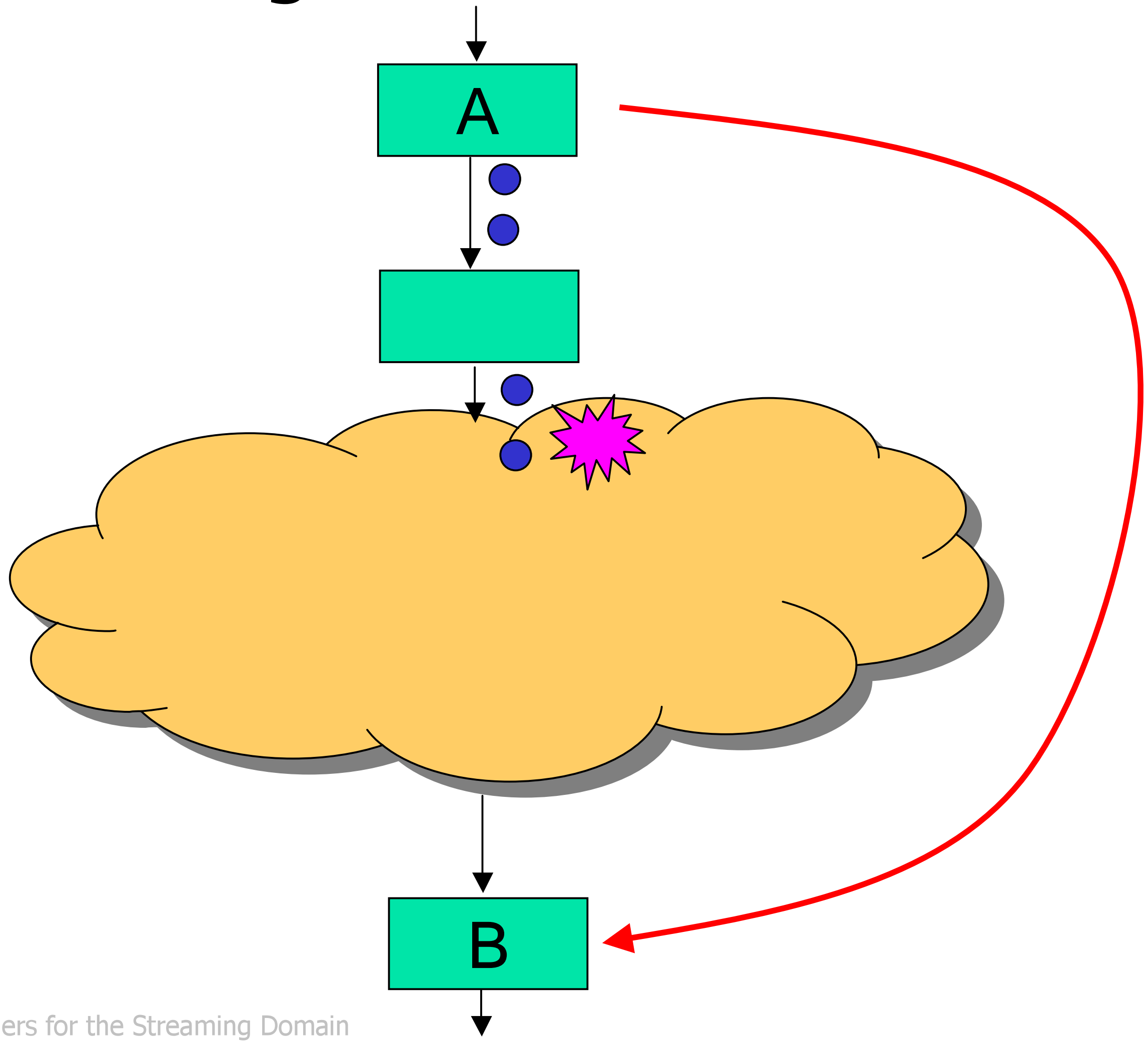
Message Timing

- A sends message to B with zero latency



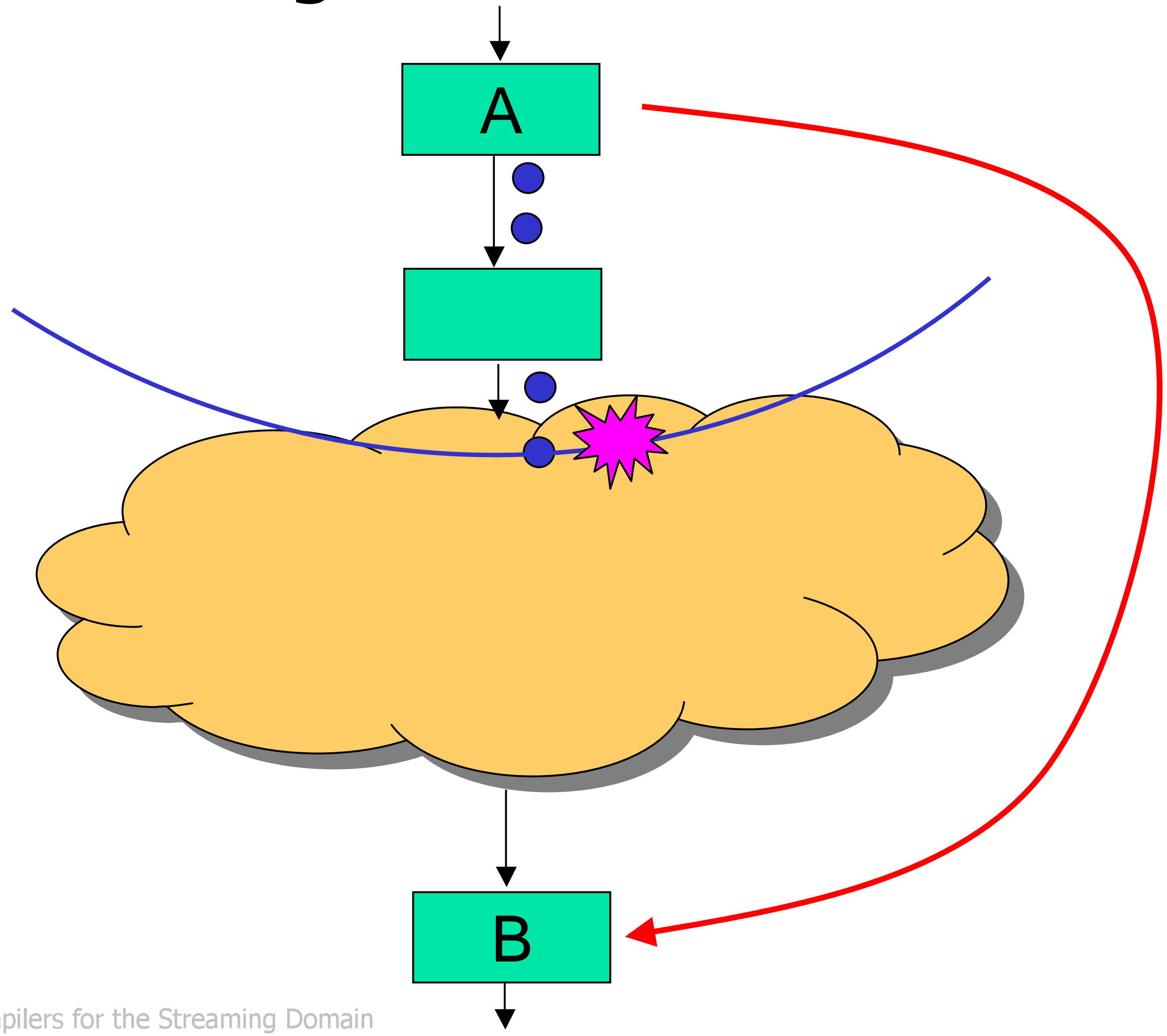
Message Timing

- A sends message to B with zero latency



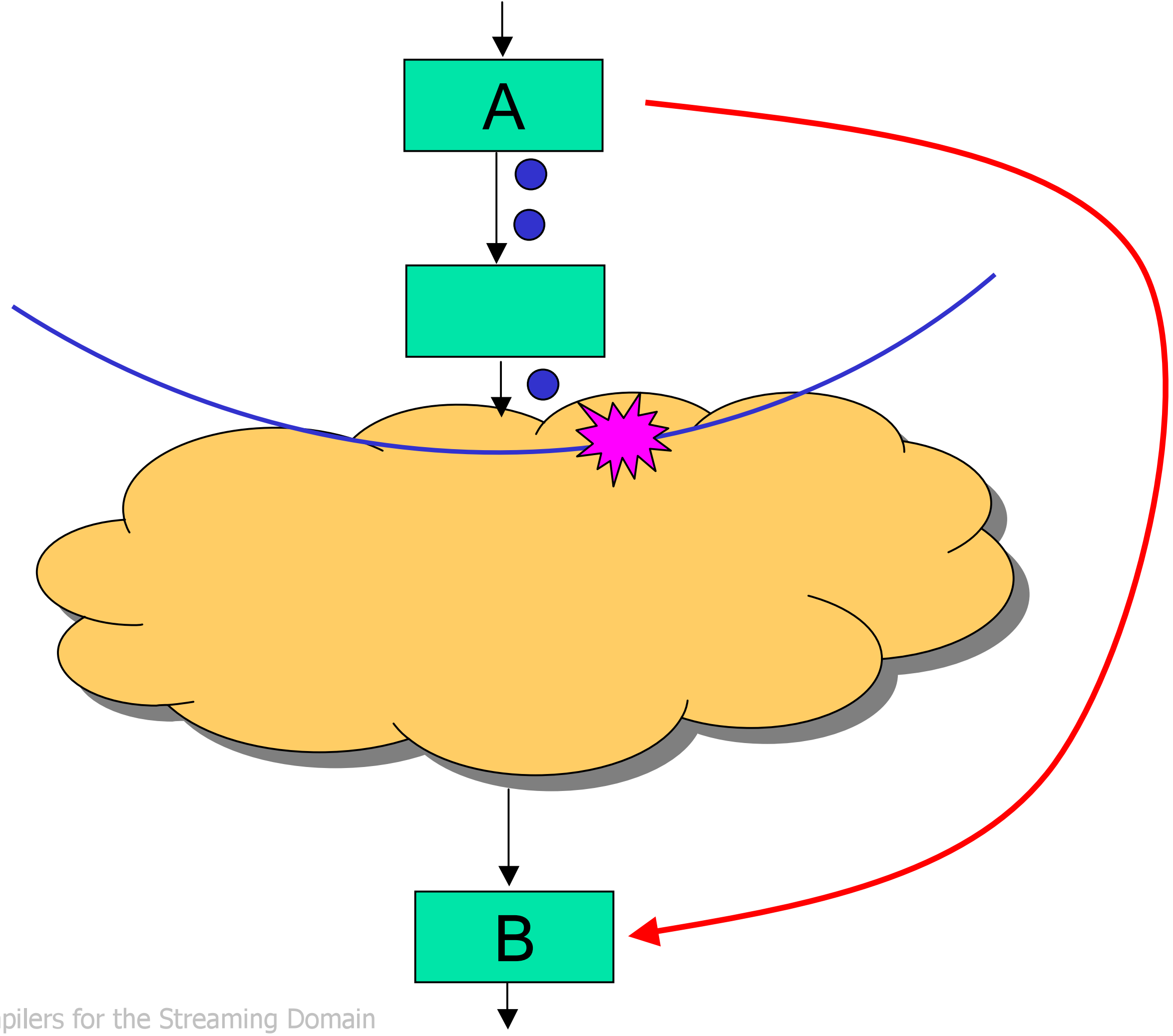
Message Timing

- A sends message to B with zero latency



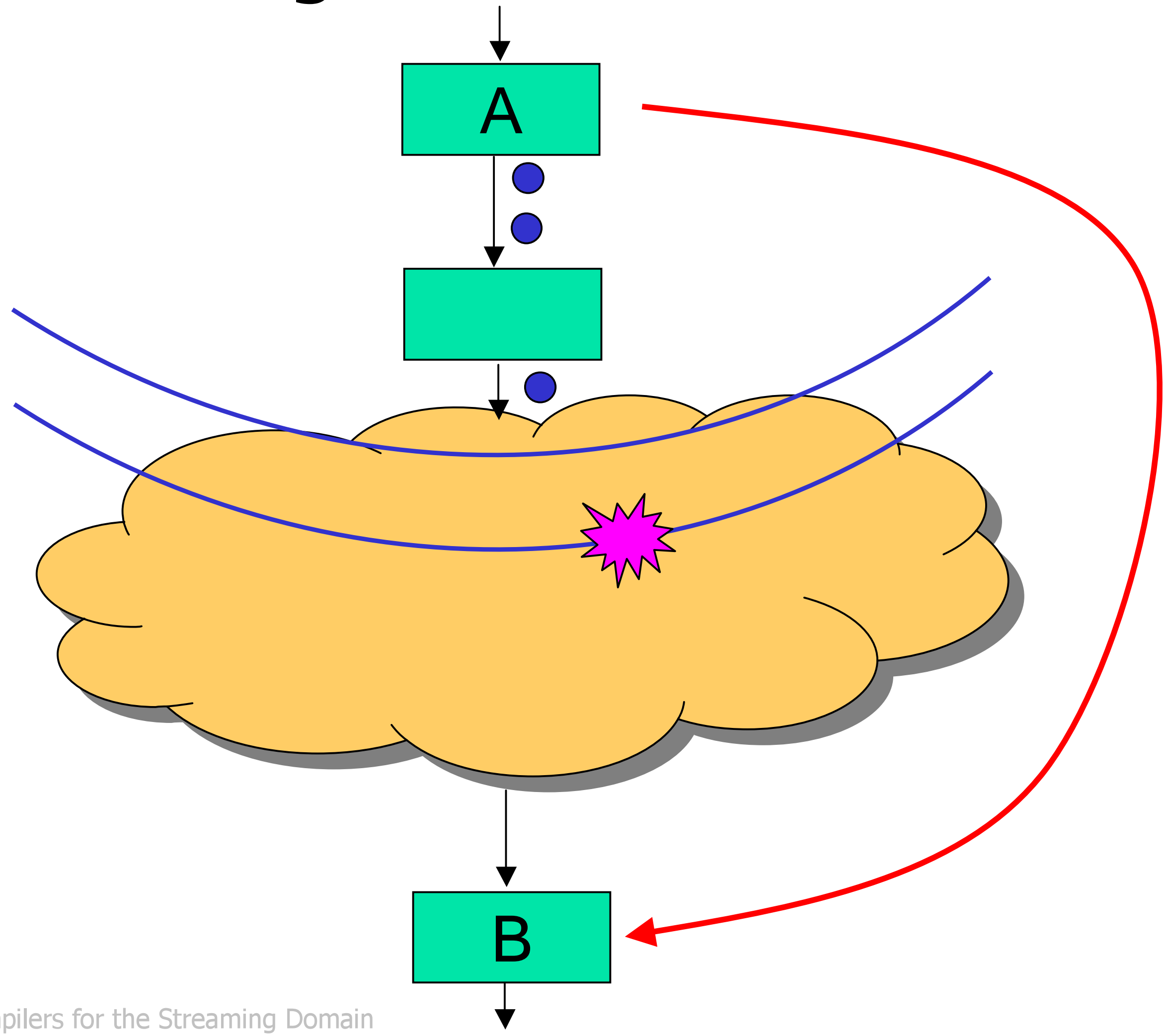
Message Timing

- A sends message to B with zero latency



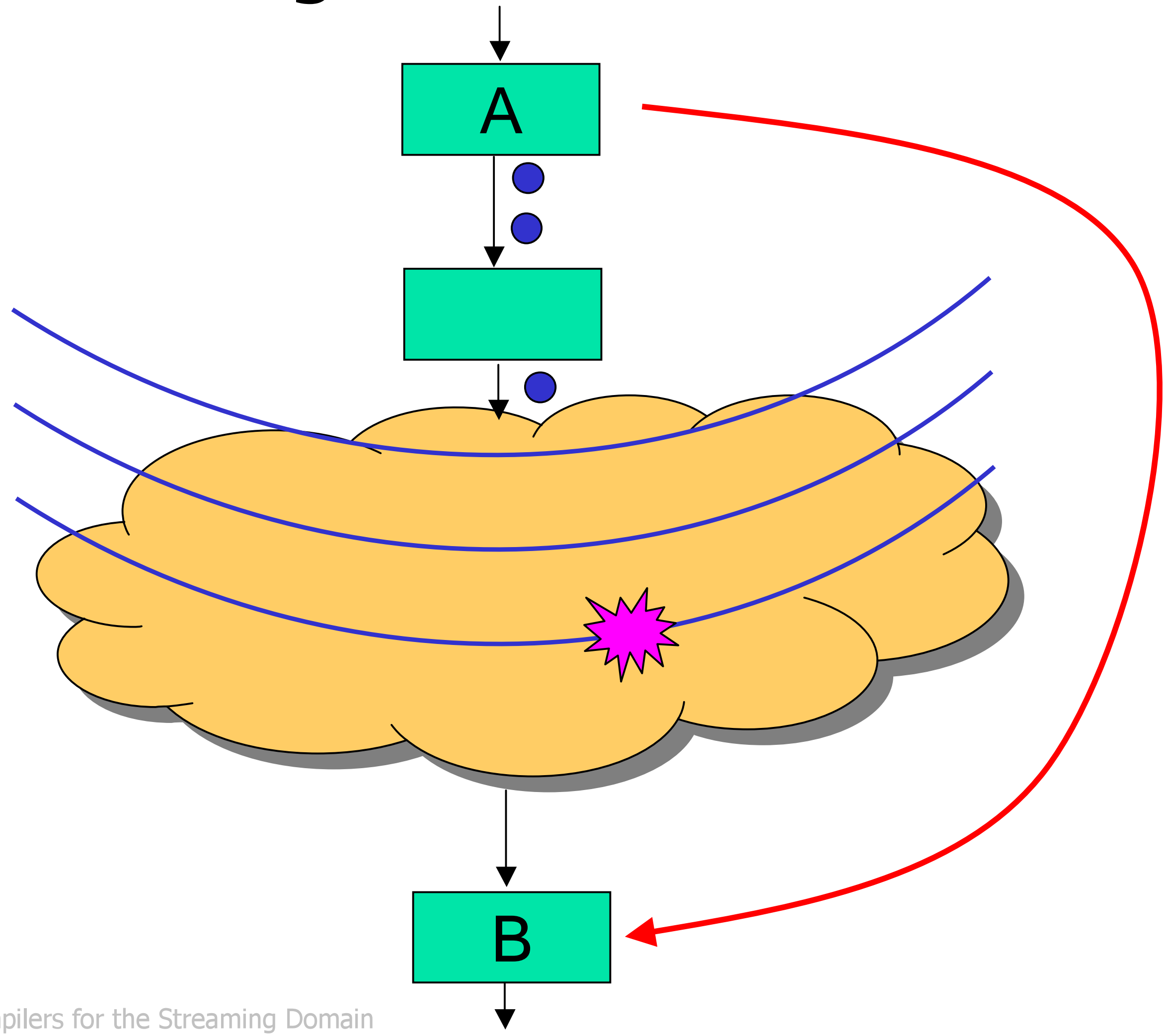
Message Timing

- A sends message to B with zero latency



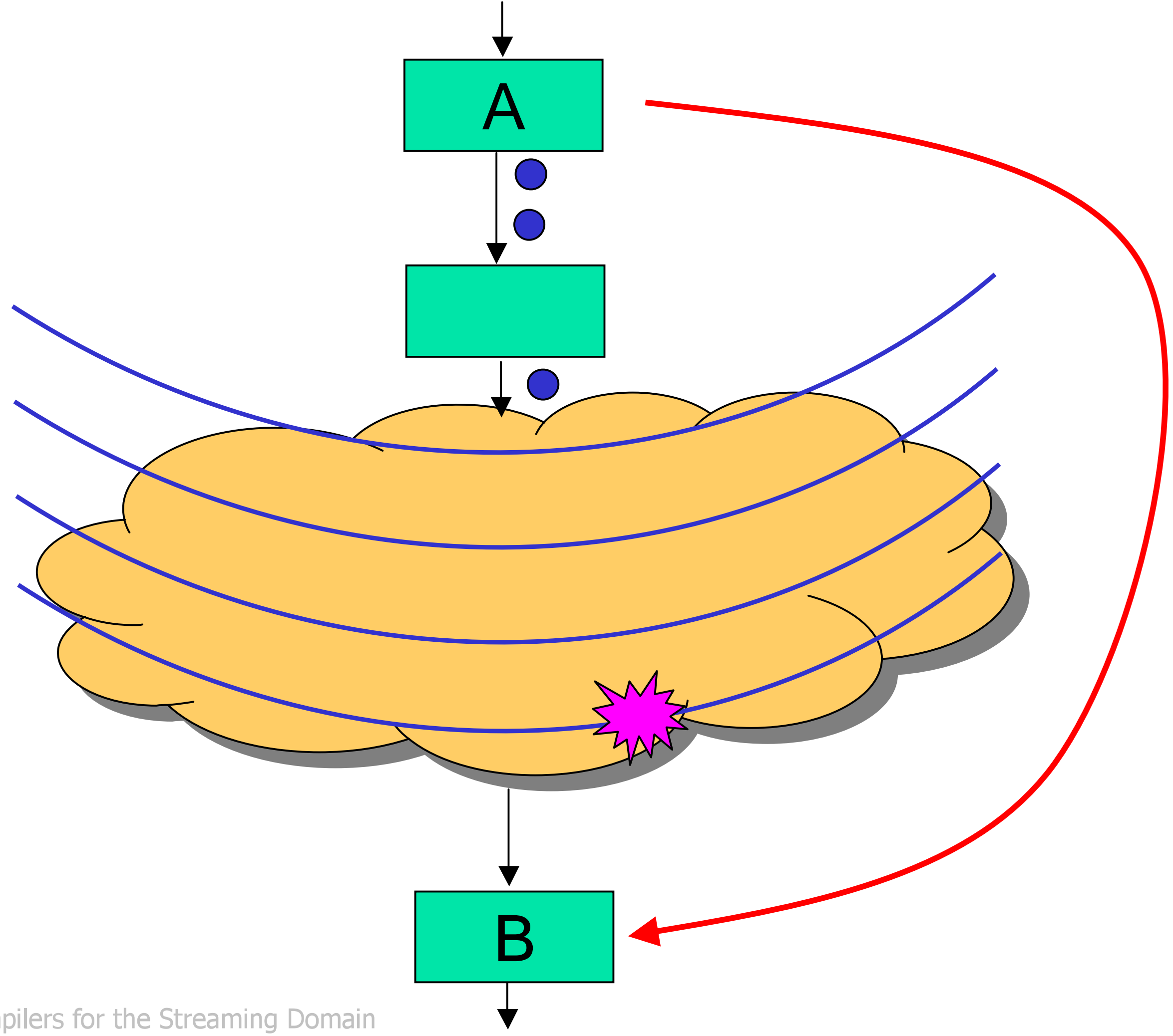
Message Timing

- A sends message to B with zero latency



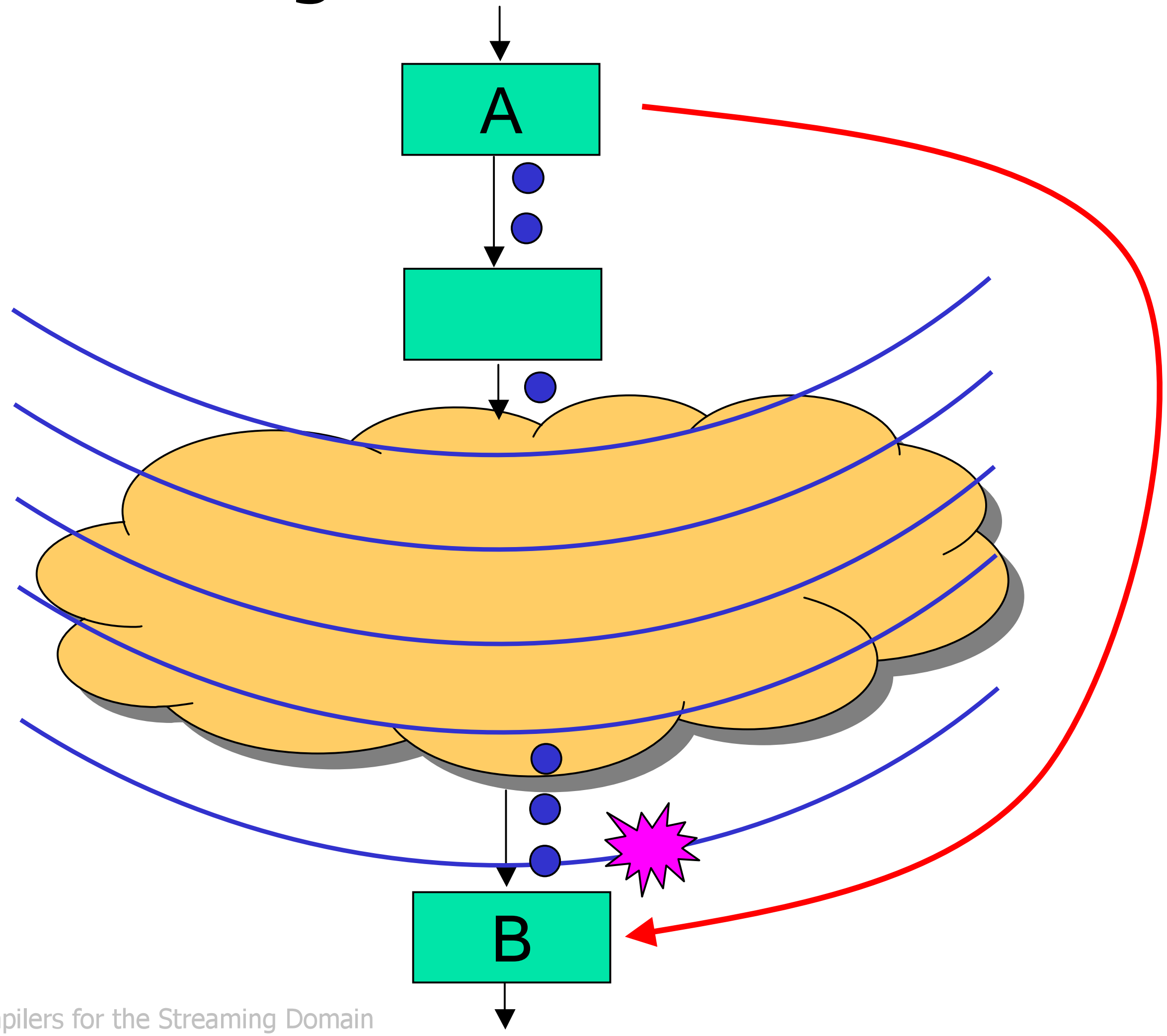
Message Timing

- A sends message to B with zero latency



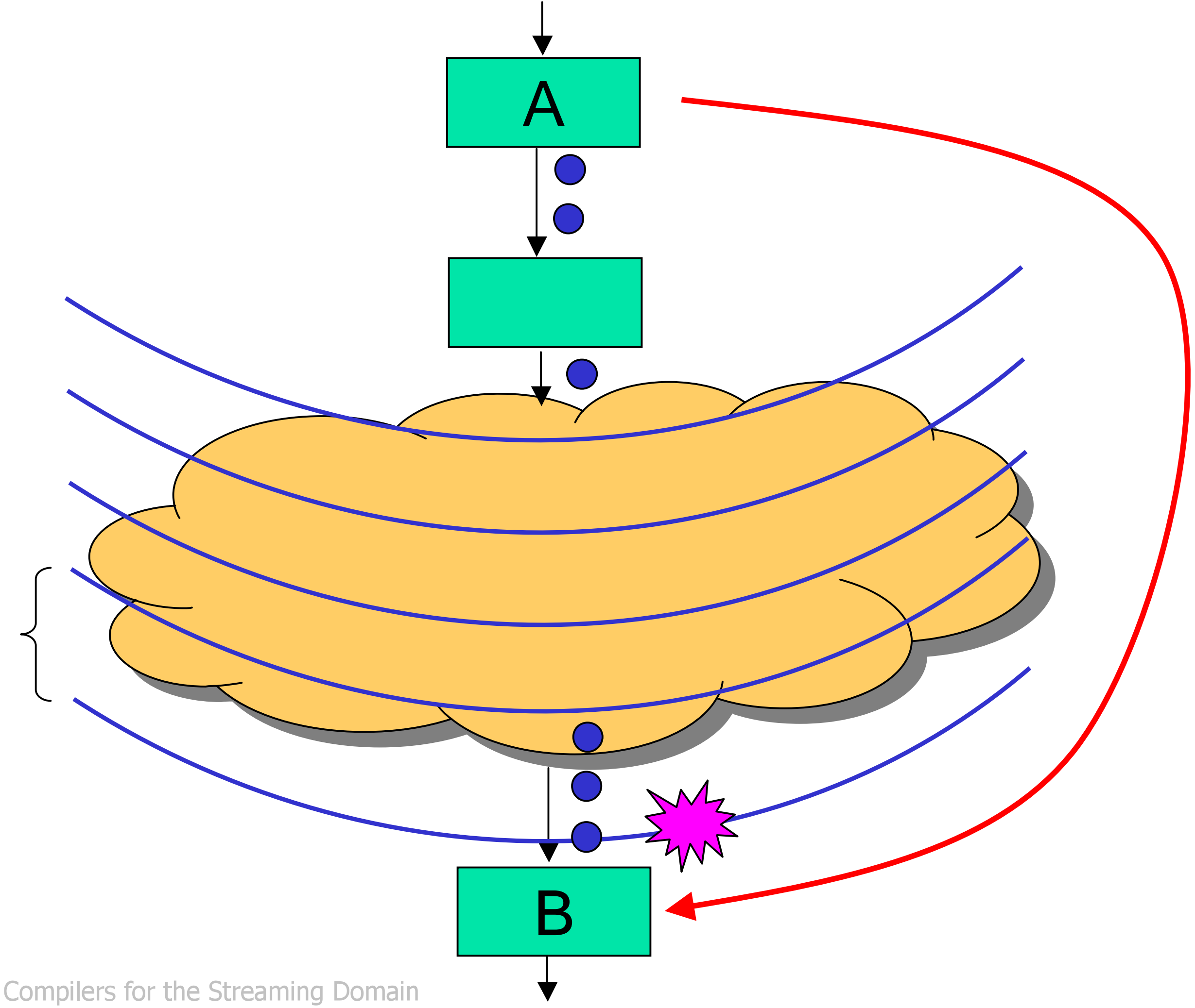
Message Timing

- A sends message to B with zero latency



Message Timing

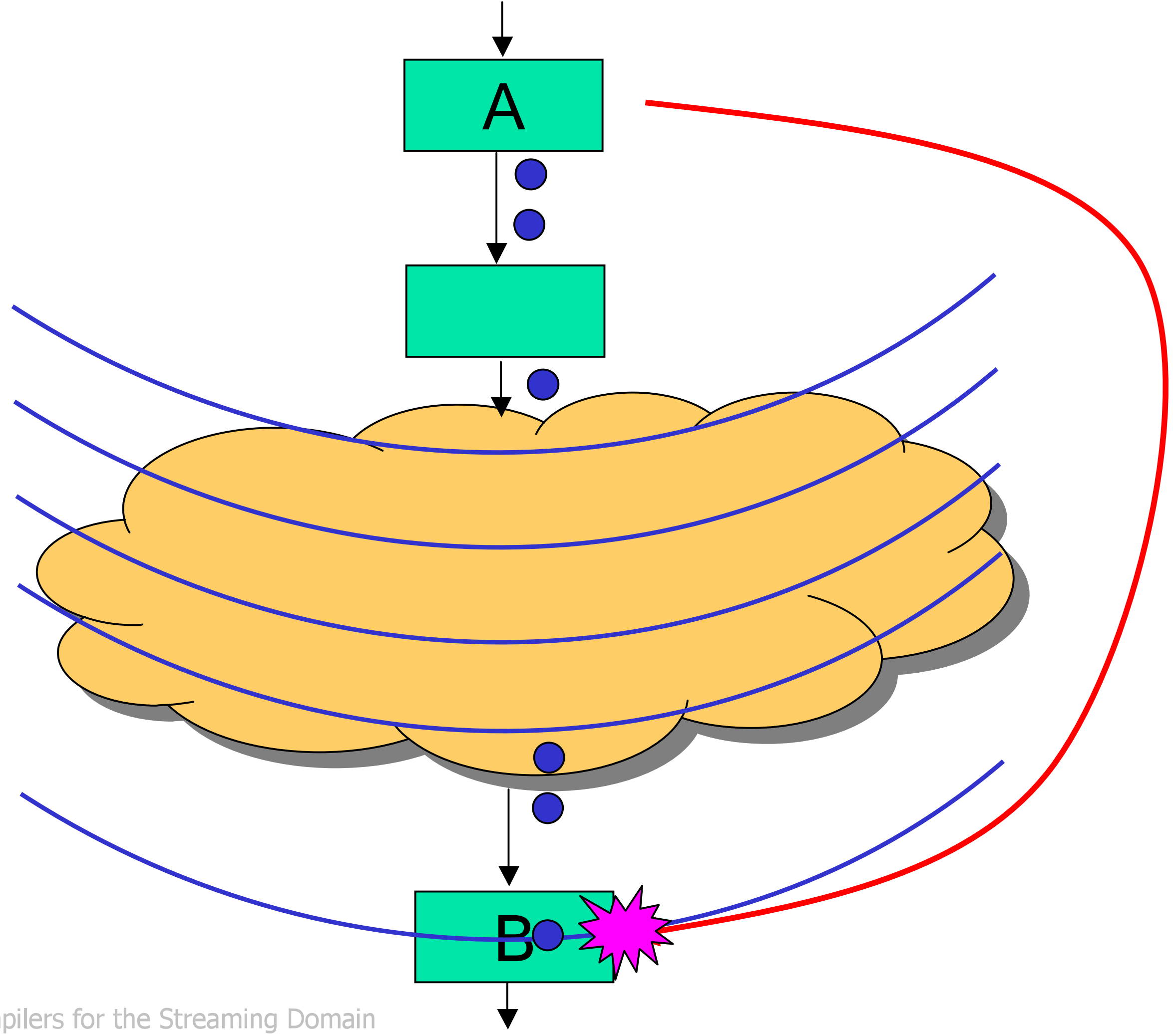
- A sends message to B with zero latency



Distance between wavefronts might have changed

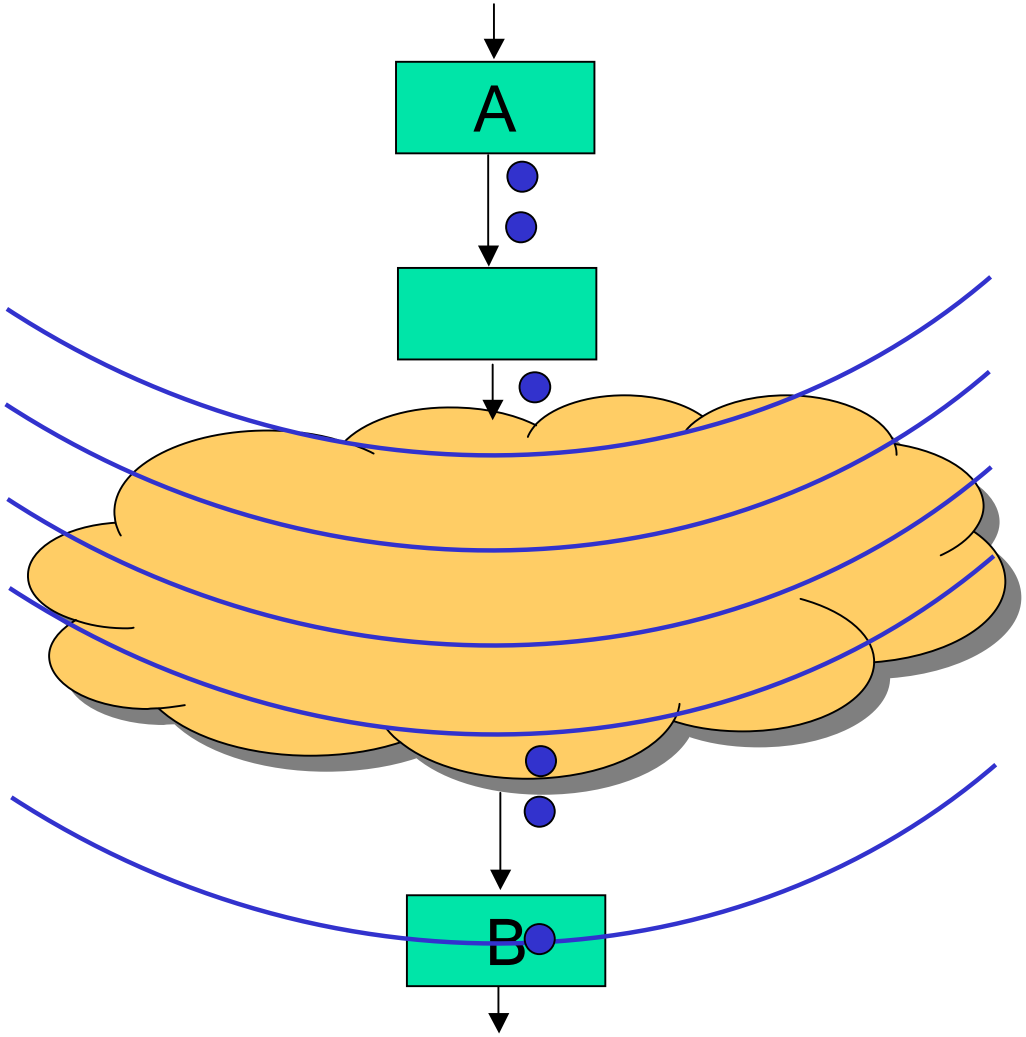
Message Timing

- A sends message to B with zero latency



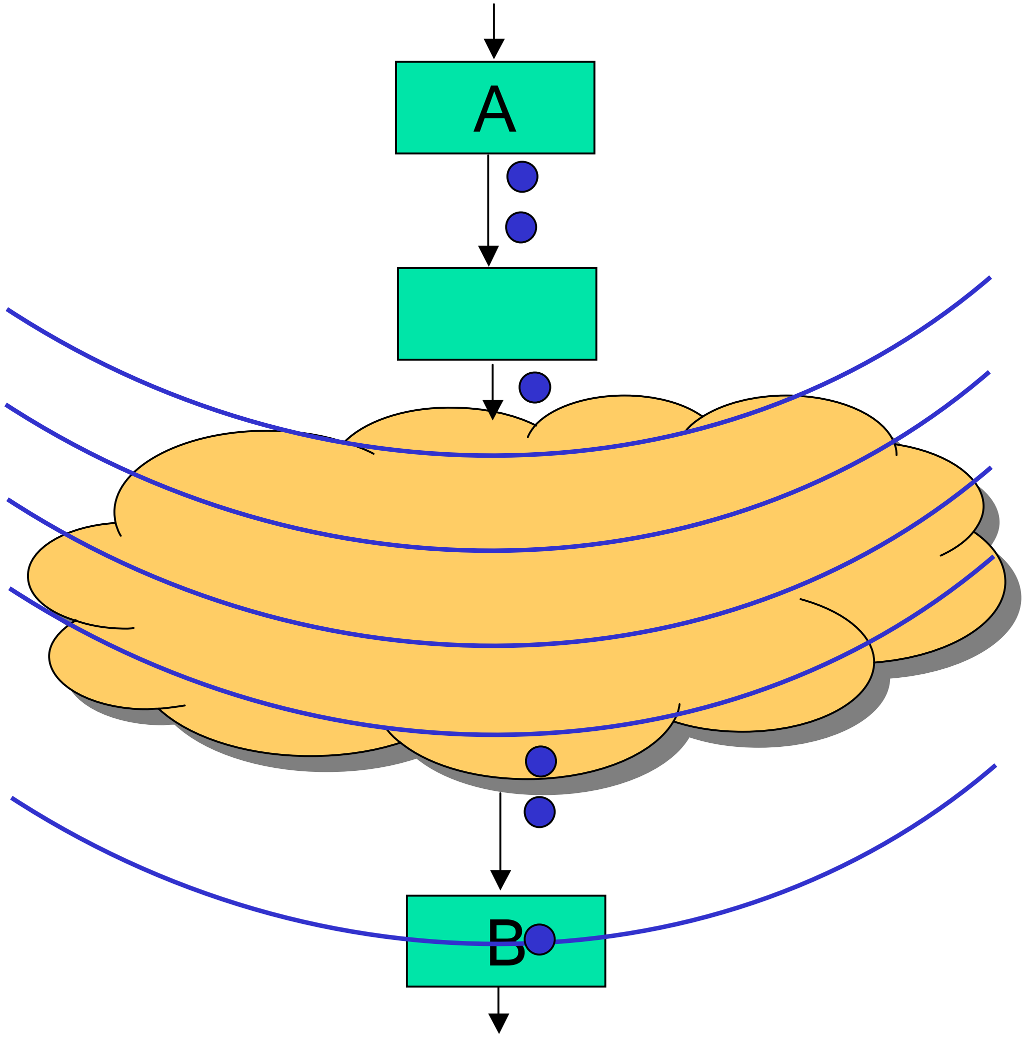
General Message Timing

- Latency of N means:
  - Message attached to wavefront that *sender* sees in N executions



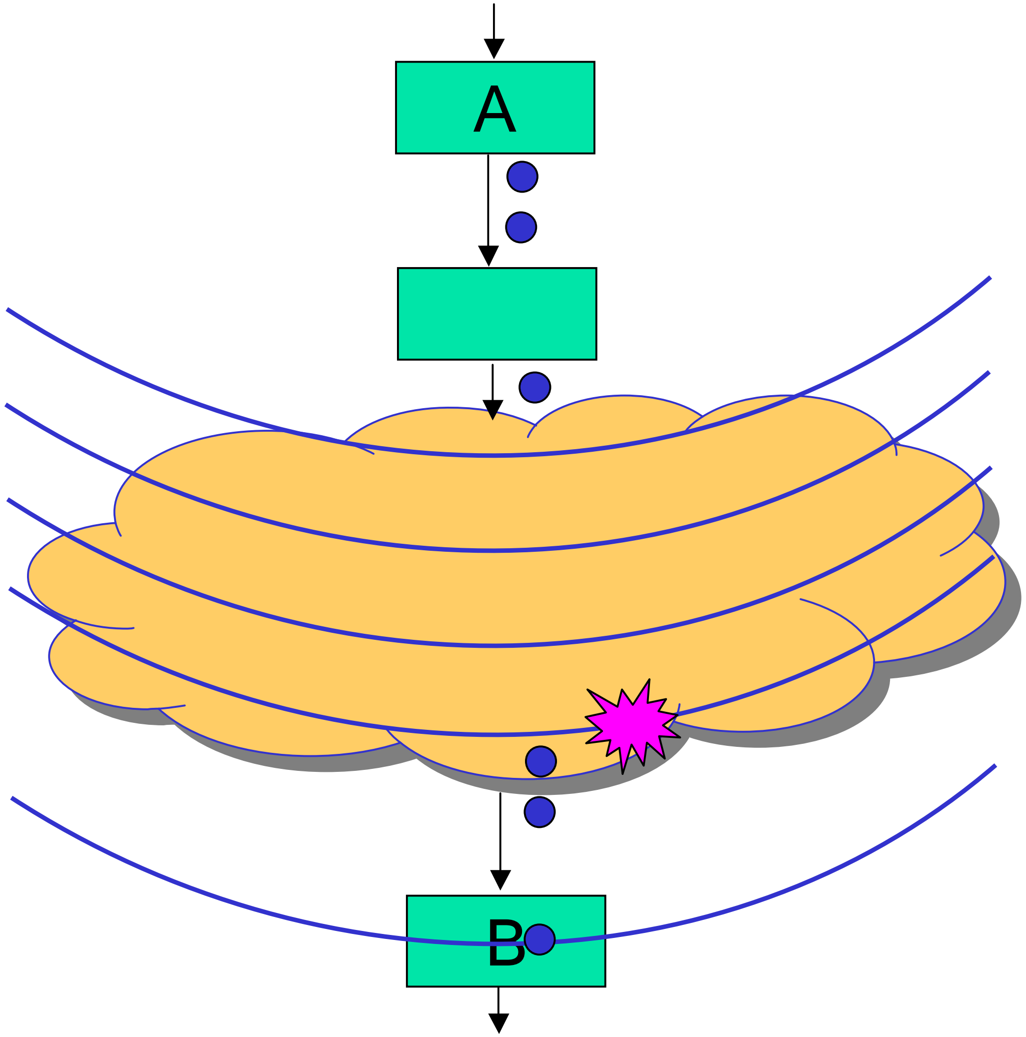
General Message Timing

- Latency of N means:
  - Message attached to wavefront that *sender* sees in N executions
- Examples:
  - A → B, latency 1



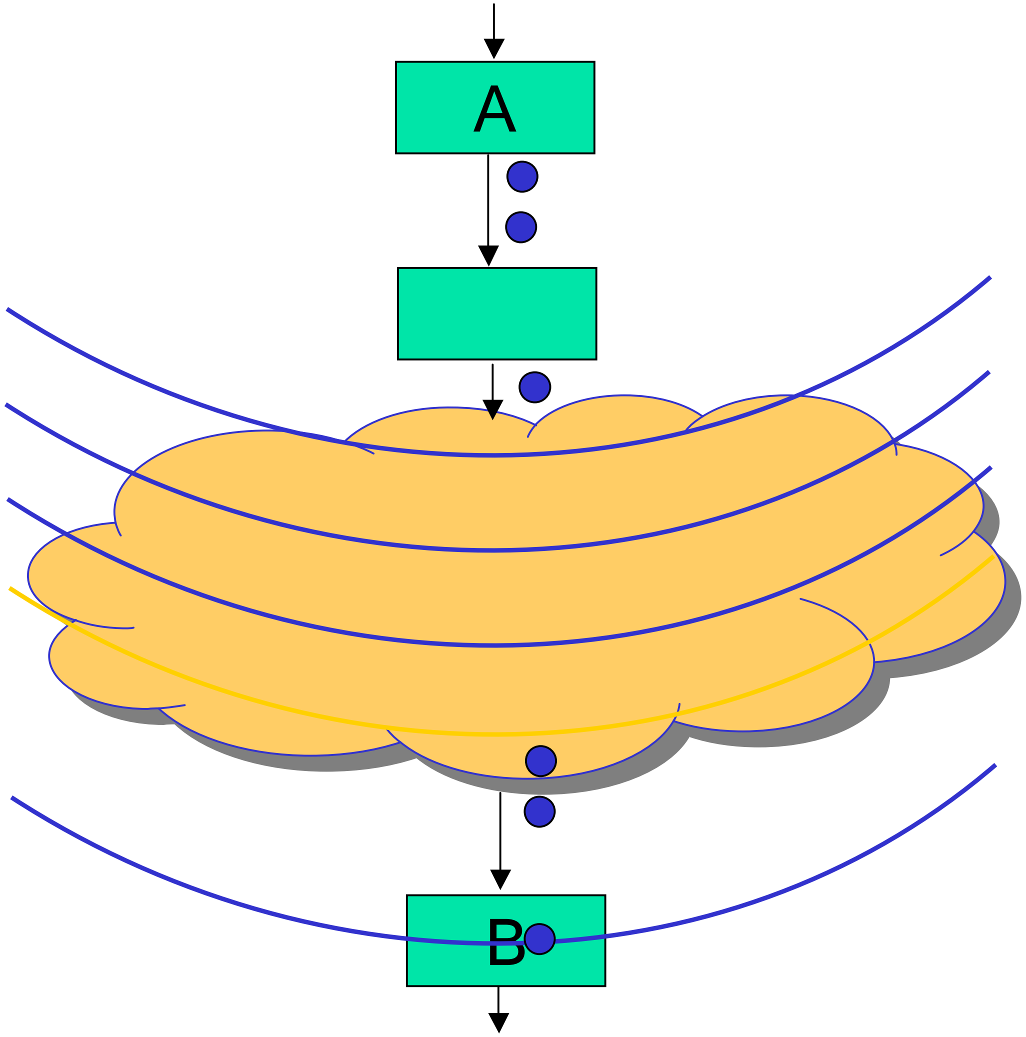
General Message Timing

- Latency of N means:
  - Message attached to wavefront that *sender* sees in N executions
- Examples:
  - A → B, latency 1



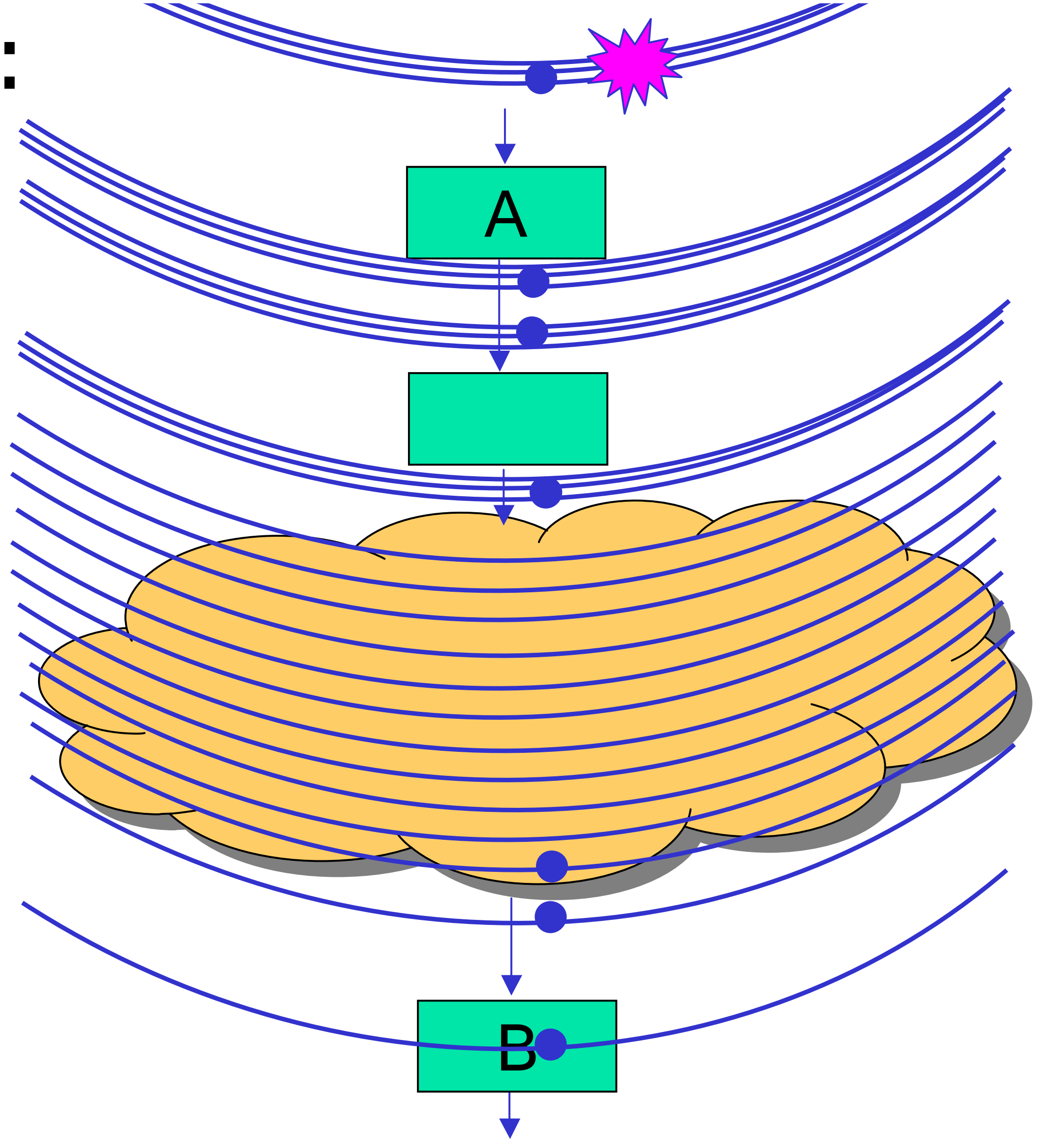
General Message Timing

- Latency of N means:
  - Message attached to wavefront that *sender* sees in N executions
- Examples:
  - A → B, latency 1
  - B → A, latency 25



General Message Timing

- Latency of N means:
  - Message attached to wavefront that *sender* sees in N executions
- Examples:
  - A → B, latency 1
  - B → A, latency 25



Rationale

- Better for the programmer
  - Simplicity of method call
  - Precision of embedding in stream
- Better for the compiler
  - Program is easier to analyze
    - No code for timing / embedding
    - No control channels in stream graph
  - Can reorder filter firings, respecting constraints
  - Implement in most efficient way

StreamIt Language Summary

- High-level, machine-independent stream language
  - Structured streams for high-bandwidth dataflow
  - Messaging system for control
  - Working on new dynamic constructs
- Compiler-conscious language design can improve both programmability and performance

Stream Programming Models

- Prototyping environments
- Conventional languages
  - Object Oriented
  - Procedural
  - Assembly
- Stream languages
  - StreamIt
  - Brook
  - Cg

The Brook Language (Stanford)

- Also an architecture-independent stream language
  - Evolved out of StreamC / KernelC, which targets Imagine

StreamIt	**Brook/StreamC**
Think structured synchronous data flow	Think pointer-less C, with embedded dataflow graphs instead of loop nests
Single stream graph	Multiple stream graphs, surrounded by C-subset
Streams are infinite length	Streams are finite length
Static rates	Dynamic rates
"Filters" can have state, may require sequential processing	"Kernels" must be state-less, allow parallel processing
Designed by compiler people, clean but more constrained	Designed by application and architecture people, rough but more expressive

This slide adapted from Peter Mattson, Reservoir Labs

Stream Programming Models

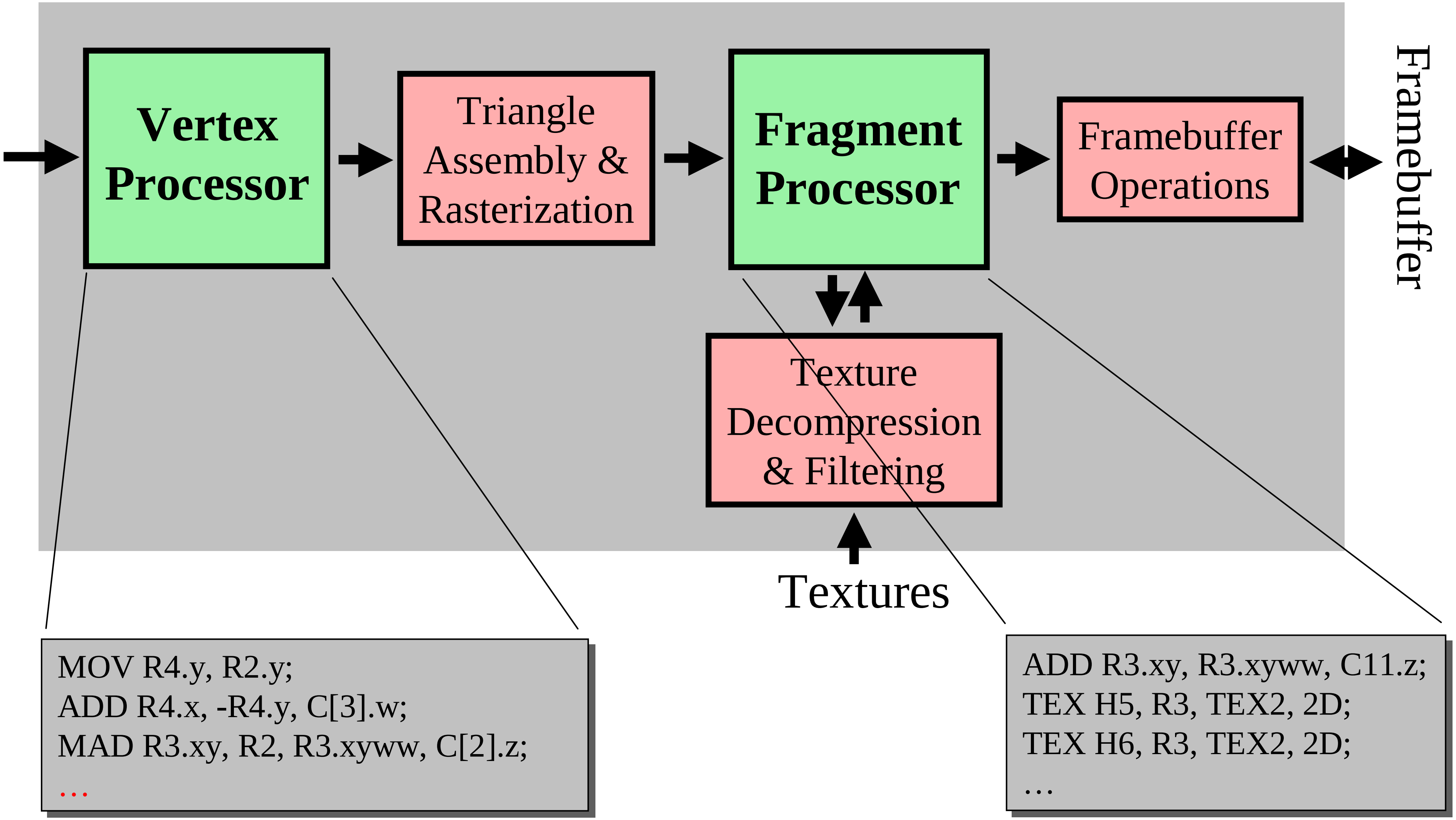
- Prototyping environments
- Conventional languages
  - Object Oriented
  - Procedural
  - Assembly
- Stream languages
  - StreamIt
  - Brook
  - Cg

The Cg language (NVIDIA)

- Cg is both a language and a system
  - Cg language is for writing stream kernels
  - Cg system targets graphics hardware
- Developed by NVIDIA
  - In collaboration with Microsoft
  - Runs on lots of hardware (not just NVIDIA’s)
- Widely deployed
  - Shipping for over a year
  - Anyone can download it
- Lots of information available:
  - Cg language specification – via download
  - Cg tutorial – buy on amazon.com
  - Paper in SIGGRAPH 2003

This slide courtesy of Bill Mark, UT Austin

GPUs are now programmable

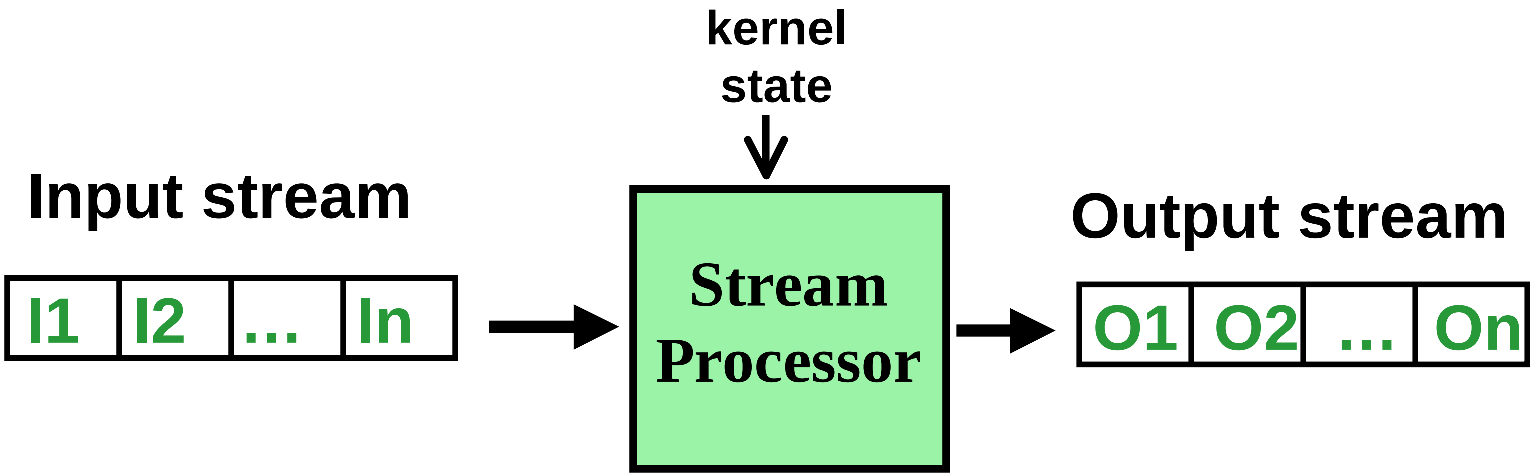


```
MOV R4.y, R2.y;
ADD R4.x, -R4.y, C[3].w;
MAD R3.xy, R2, R3.xyww, C[2].z;
…
```

```
ADD R3.xy, R3.xyww, C11.z;
TEX H5, R3, TEX2, 2D;
TEX H6, R3, TEX2, 2D;
…
```

This slide courtesy of Bill Mark, UT Austin

Programmable units in GPUs are stream processors



- The programmable unit executes a **computational kernel** for **each** input element
- Streams consist of ordered elements

This slide courtesy of Bill Mark, UT Austin

Design Decisions in Cg

- Cg: C for graphics
  - Like C, directly map to underlying hardware
  - General purpose (not just a shading language)
- A program for each pipeline stage
  - Alternative: write one program and have compiler do the partitioning
  - Chose to separate at programmer level to guarantee valid mapping; e.g., for outer-level control dependences
- A language for expressing stream kernels
  - Unlike StreamIt/Brook, does not express high-level connections in stream graph
  - Instead, write kernels for hardware resources and use connections of hardware
  - Use auxiliary namespace (bind-by-name) as dataflow interface between vertex and fragment processors
  - Use Cg runtime API to control kernel execution

Cg language example

```
void simpleTransform(float4   objectPosition : POSITION,
                     float4   color          : COLOR,
                     float4   decalCoord     : TEXCOORD0,
                 out float4   clipPosition   : POSITION,
                 out float4   Color          : COLOR,
                 out float4   oDecalCoord    : TEXCOORD0,
             uniform float    brightness,
             uniform float4x4 modelViewProjection)
{
  clipPosition = mul(modelViewProjection, objectPosition);
  oColor = brightness * color;
  oDecalCoord = decalCoord;
}
```

This slide courtesy of Bill Mark, UT Austin

How should system support different levels of HW?

NV20 R300 NV40 NV50

R200 NV30 R400 … ?

- HW capabilities change each generation
  - Data types
  - Support for branch instructions, …
- We expect this problem to persist
  - Future GPUs will have new features
- Mandate exactly one feature set?
  - Must strand older HW or limit newer HW

This slide courtesy of Bill Mark, UT Austin

Two options for handling HW differences

- Emulate missing features?
  - Too slow on GPU
  - Too slow on CPU, especially for fragment HW
- Expose differences to programmer?
  - They chose this option
  - Differences exposed via **subsetting**
  - A **profile** is a named subset
  - Cg supports function overloading by profile

This slide courtesy of Bill Mark, UT Austin

Cg is closely related to other recent languages

- Microsoft HLSL
  - Largely compatible with Cg
  - NVIDIA and Microsoft collaborated
- OpenGL ARB shading language
- All three languages are similar
  - Overlapping development
  - Extensive cross-pollination of ideas
  - Designers mostly agreed on right approach
- Systems are different

This slide courtesy of Bill Mark, UT Austin

Summary

- There are many prototyping environments for streaming applications
- However, industry still relies on C, C++, and assembly code for implementations
  - Tedious, costly, error-prone
- Stream languages have potential to improve both performance and programmability
  - Expose communication patterns
  - Expose parallelism
  - Encapsulate common idioms
- Examples: StreamIt, Brook, Cg