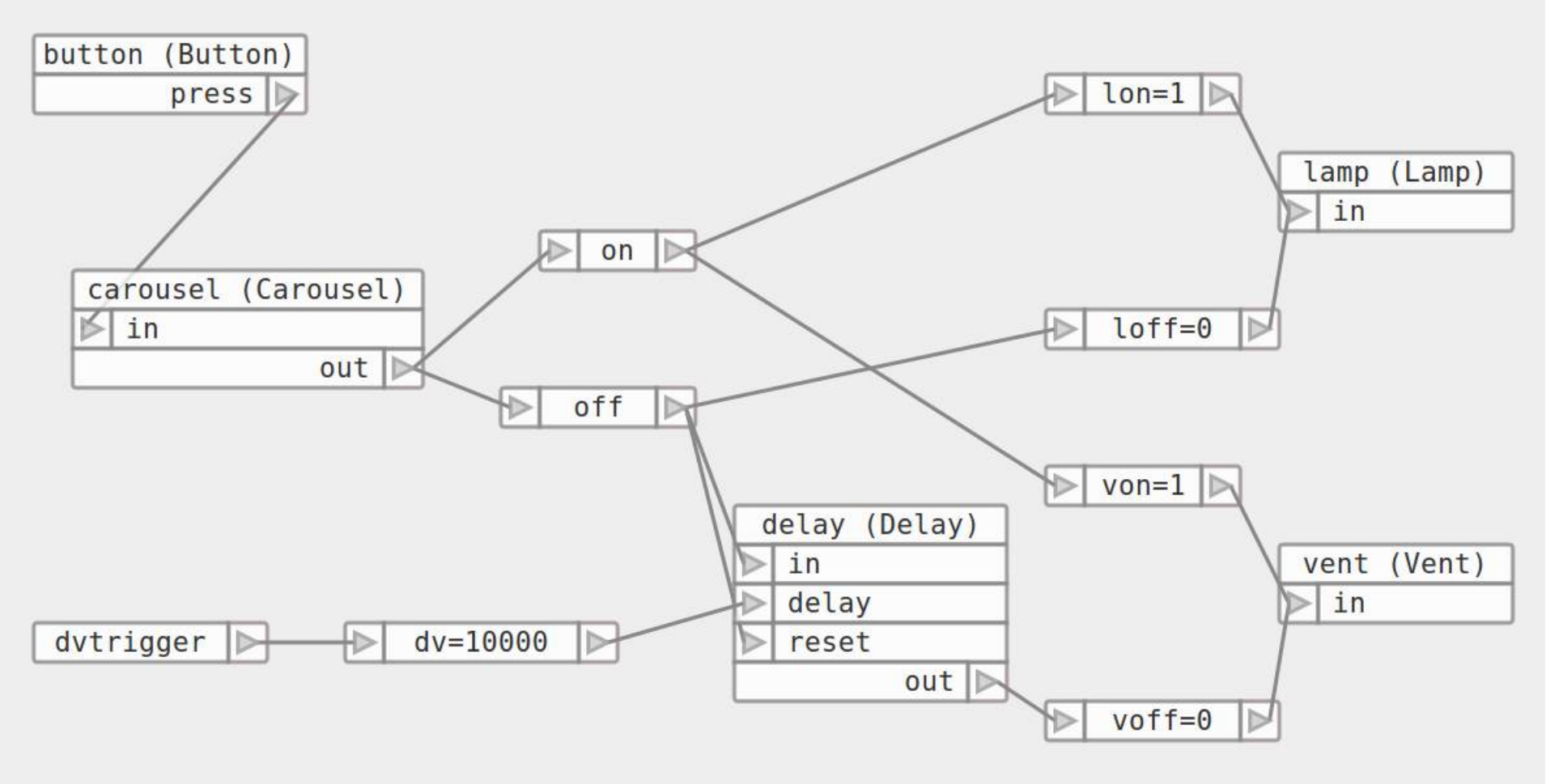
Dataflow Programming



ern0 – http://linkbroker.hu

V4 – 2015.09.26

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Basics

Definition

Component & Port

Data Types

Source, Processor, Sink

Advanced

Component: Native vs Composite

Scheduling: Synchronous vs Asynchronous

Triggering: Push vs Pull

Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

App Creating vs Programming, Component Programming, Application Building

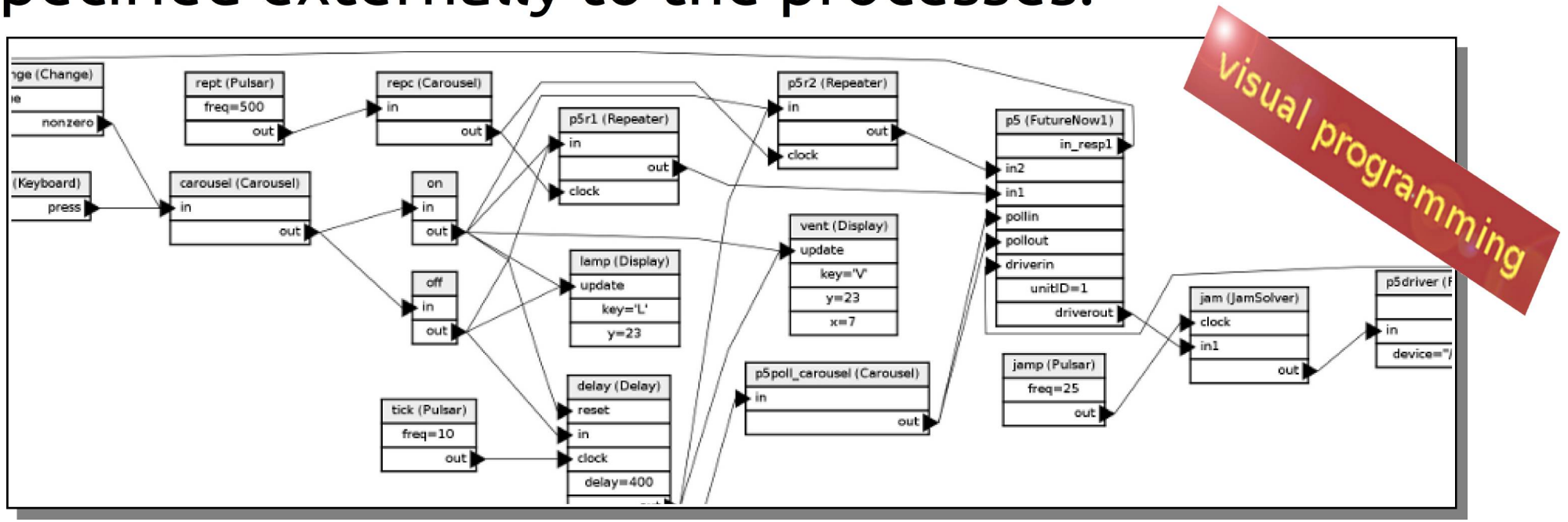
Benefits

Rapid Prototyping, Reusability, Transparency

Definition

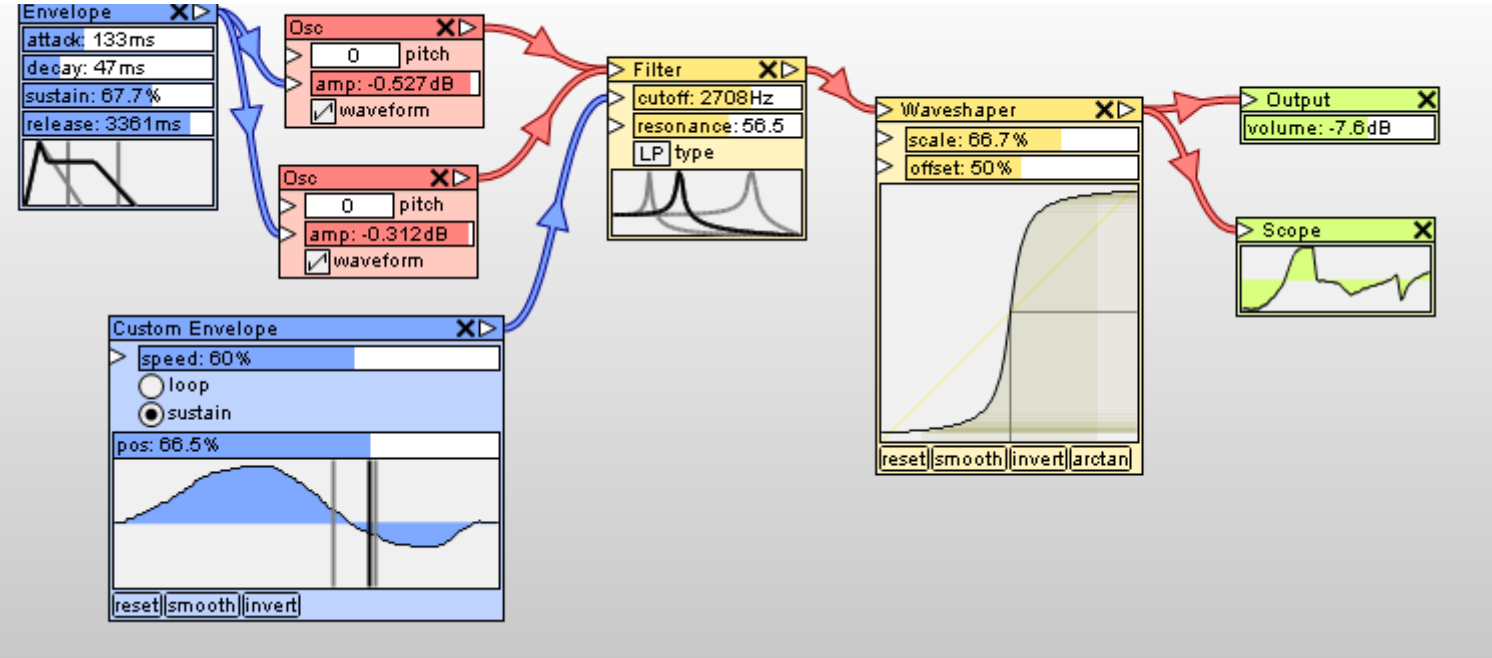
Programming paradigm / software architecture: computation is modelled as a directed graph.

Applications is a network of "black box" processes, which exchange data across predefined connections by message passing, where the connections are specified externally to the processes.



Domains

- Synth/sampler/workstation
- Audio/video processing
- Animation rendering
- Industrial/home automation
- Spreadsheet
- Task automation



Similar, See Also...

Flow Based Programming

Reactive Programming

Functional Programming

Event-Driven Programming

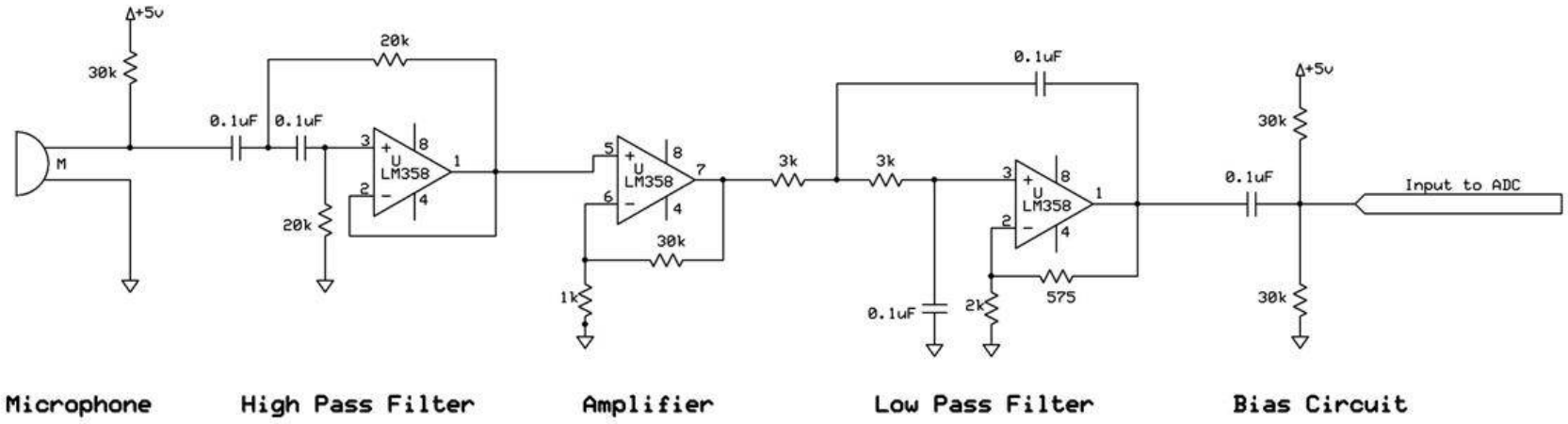
PLC (Ladder Logic, Functional Block Diagram)

Microservices

Kahn Process Networks, Petri Net

Electricity

etc.



Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

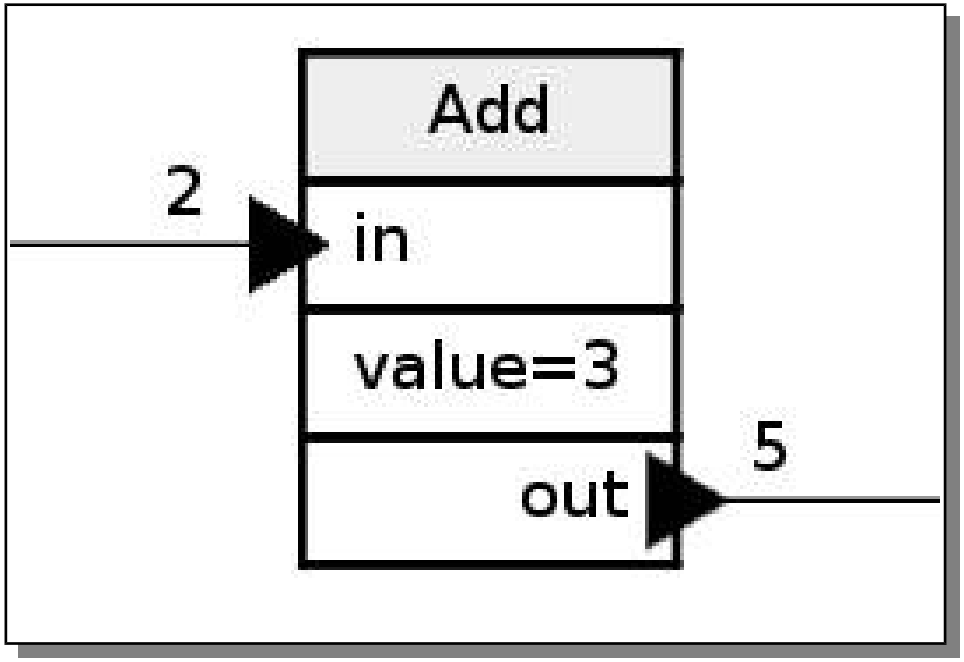
App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Component & Port

- consumer (input)
- property / parameter
- producer (output)



Component library:
platform, “language”

stateful components

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

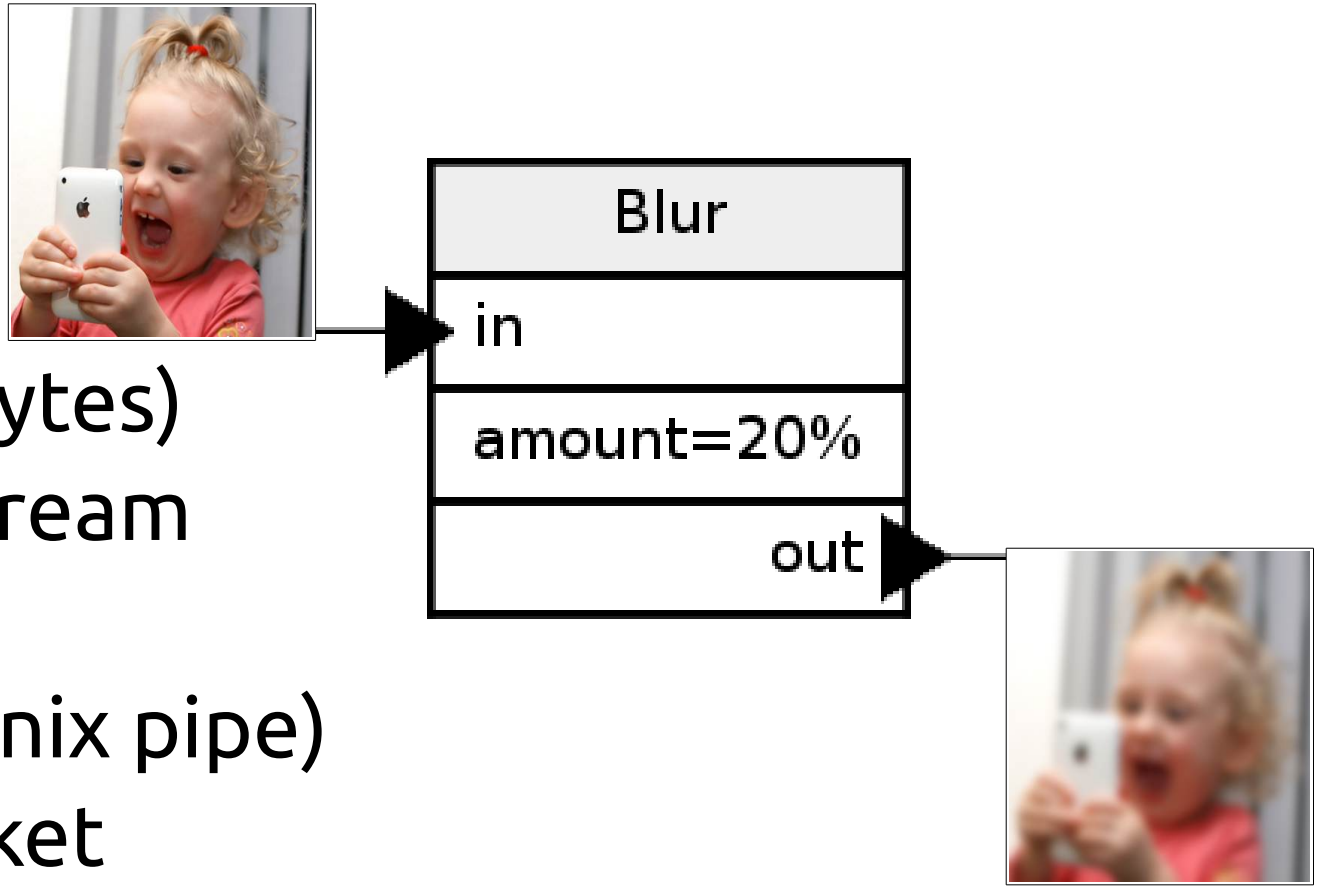
App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Data Types

- Trigger
- Integer
- Packet (some bytes)
- Image, video stream
- Audio stream
- Lines of text (Unix pipe)
- Composite packet



Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

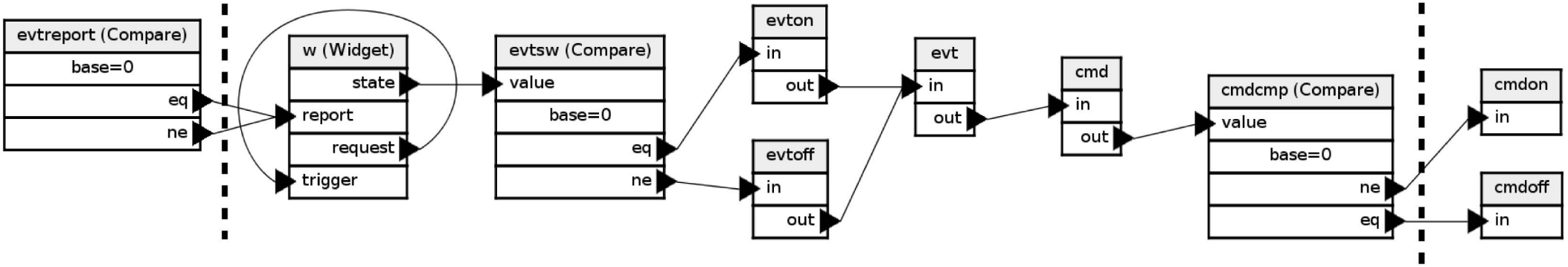
Practice

App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Component Function Types



source

external input
import, feed
network receive

processor

data process
transform
path select
process control

sink

result presentation
export
network send

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

App Creating vs Programming, Component Programming, Application Building

Benefits

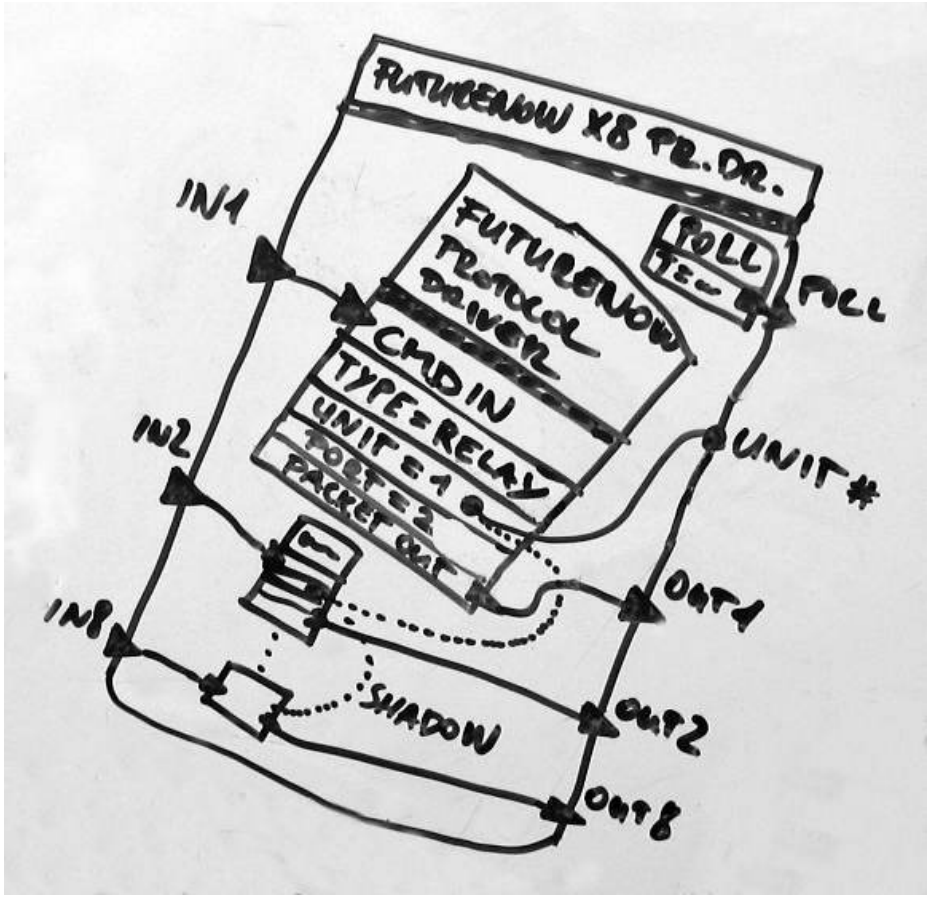
Rapid Prototyping, Reusability, Transparency

Component Implementation Modes

Native

```
class ChangeComponent {

  void messageHandler(Msg* message) {

    int v = message->getValue();
    int l = last->getValue();

    if (v == l) return;
    last->setValue(v);

    changePort->fire(v);

    if (v == 0) {
      zeroPort->fire(v);
    } else {
      nonzeroPort->fire(v);
    }

  } // messageHandler()

} // class
```

Composite



unlimited depth

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

App Creating vs Programming, Component Programming, Application Building

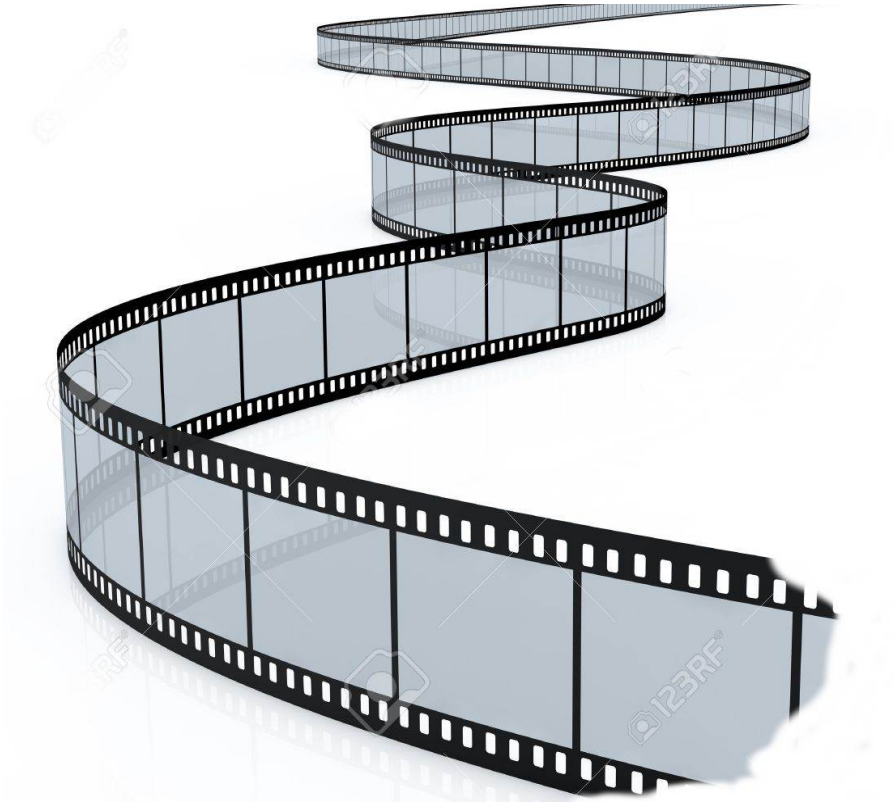
Benefits

Rapid Prototyping, Reusability, Transparency

Scheduling Modes

Synchronous

system clock



Asynchronous

trigger

	Variables	
5	Future Value = FV =	
6	Present Value = PV =	
7	Regular Payment Made at Regular Time Intervals = PMT =	$ 250.00
8	Annual (Year) Rate = i =	6.00%
9	Number of Compounding Periods per Year =n =	12
10	Years = x =	5
11	Period Rate = i/n =	0.0050
12	Total Number of Periods = n*x =	=B9*B10
13	Ordinary Annuity (PMT at end) = 0; Annuity Due (PMT at beg) = 1	

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Triggering Modes

Push	Pull
data/event driven	demand driven
active source component	passive source component
overload, unneeded messages	response delay, improper sampling

buffering

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

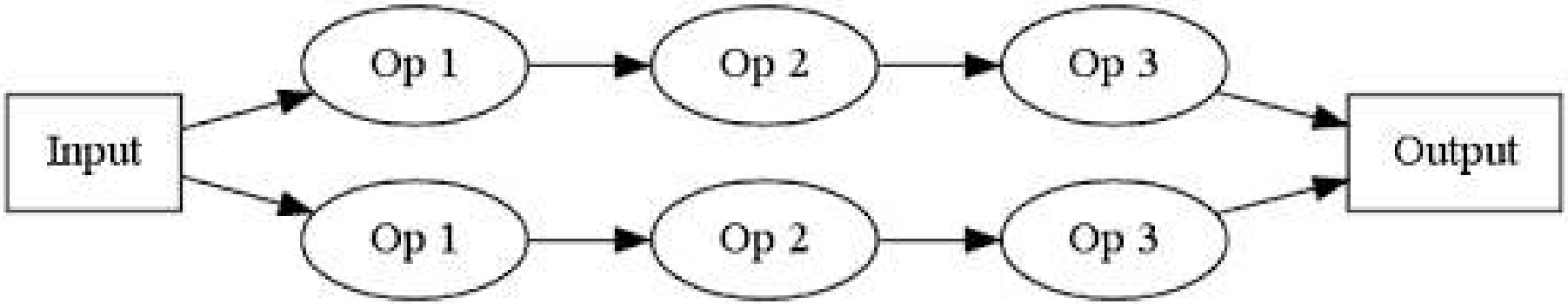
Practice

App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

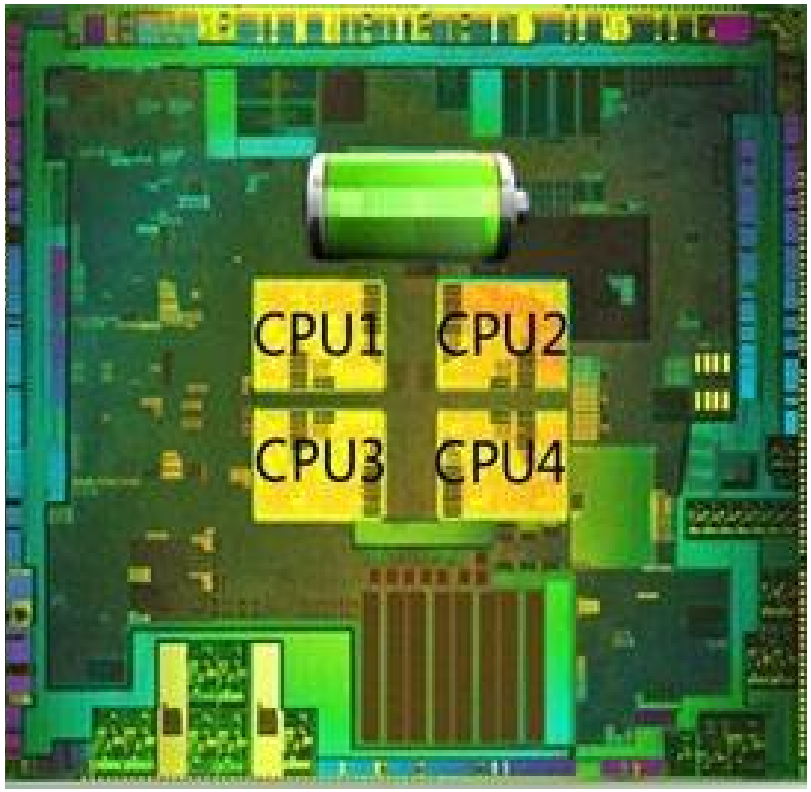
Parallel Execution



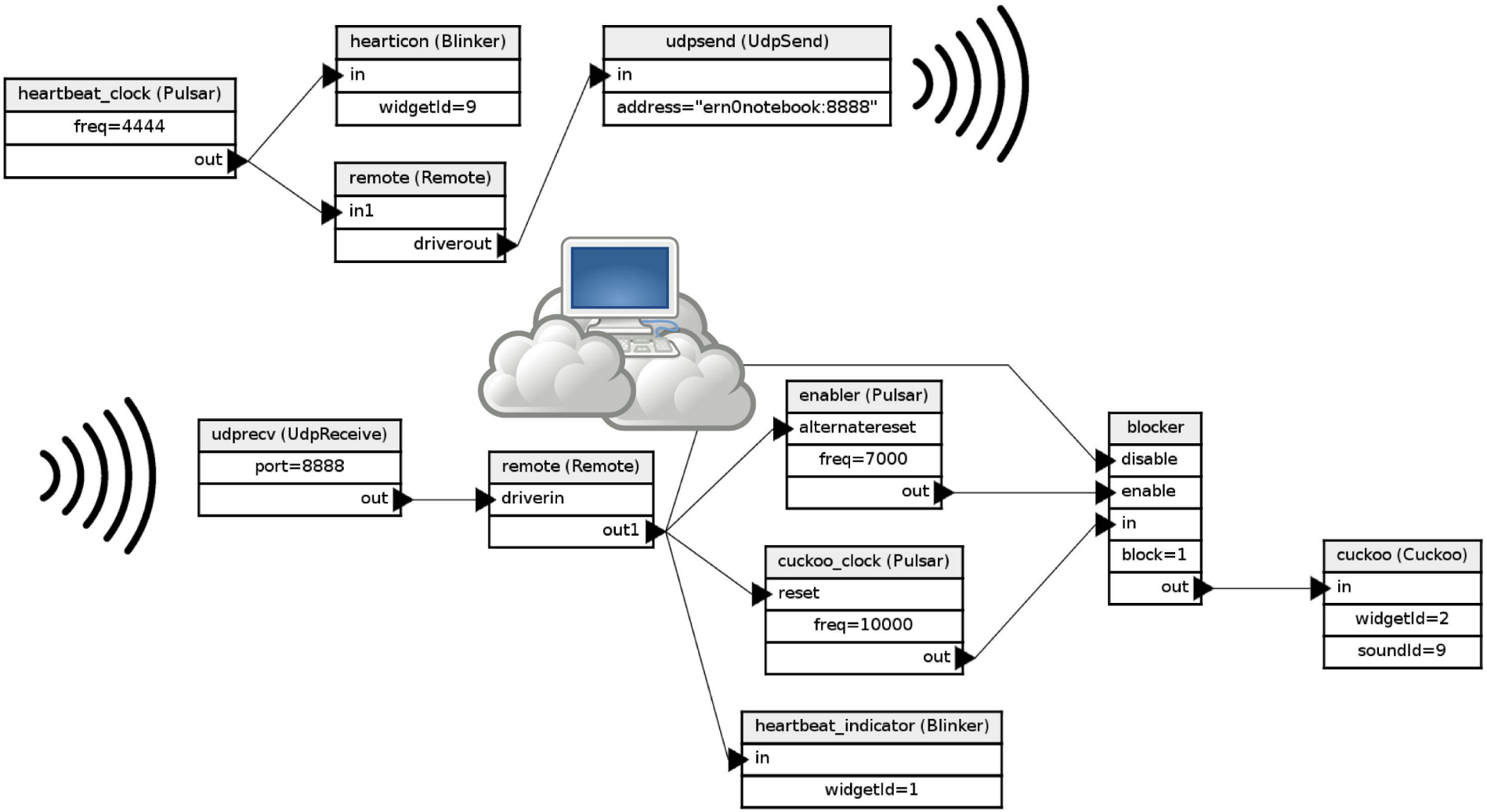
Converts single-threaded algorythms to multi-threaded

Load balancing, merging problems

Utilizes multi-core CPUs



Multi-host Application



Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

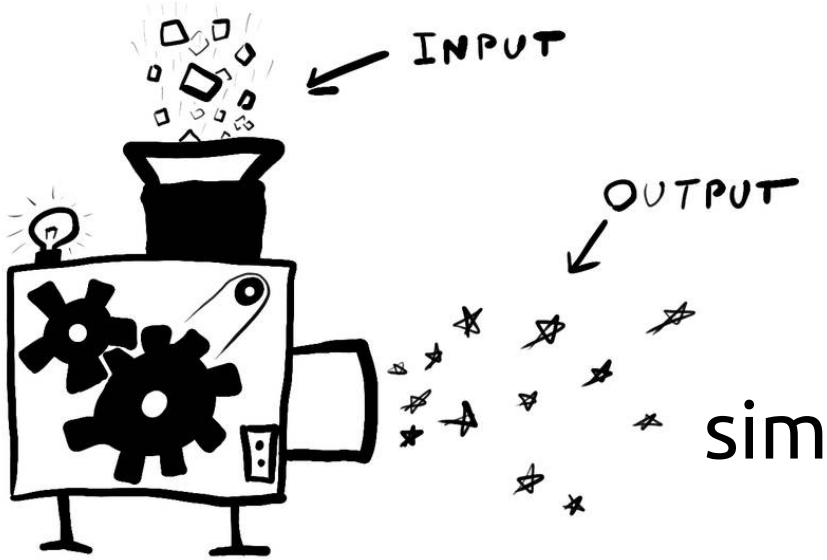
App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Unix Pipe

- All the commands are components by default
- One, universal data type: lines of text
- Restricted graph: 1-in-1-out (+ files)
- No editor required, CLI syntax (c1 | c2 | c3)
- Parallel execution (check it: ps)
  (MS-DOS: single, using tmp files)



/bin/cat
/usr/bin/tee

simple architecture

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, **Spreadsheet**, Make etc.

Practice

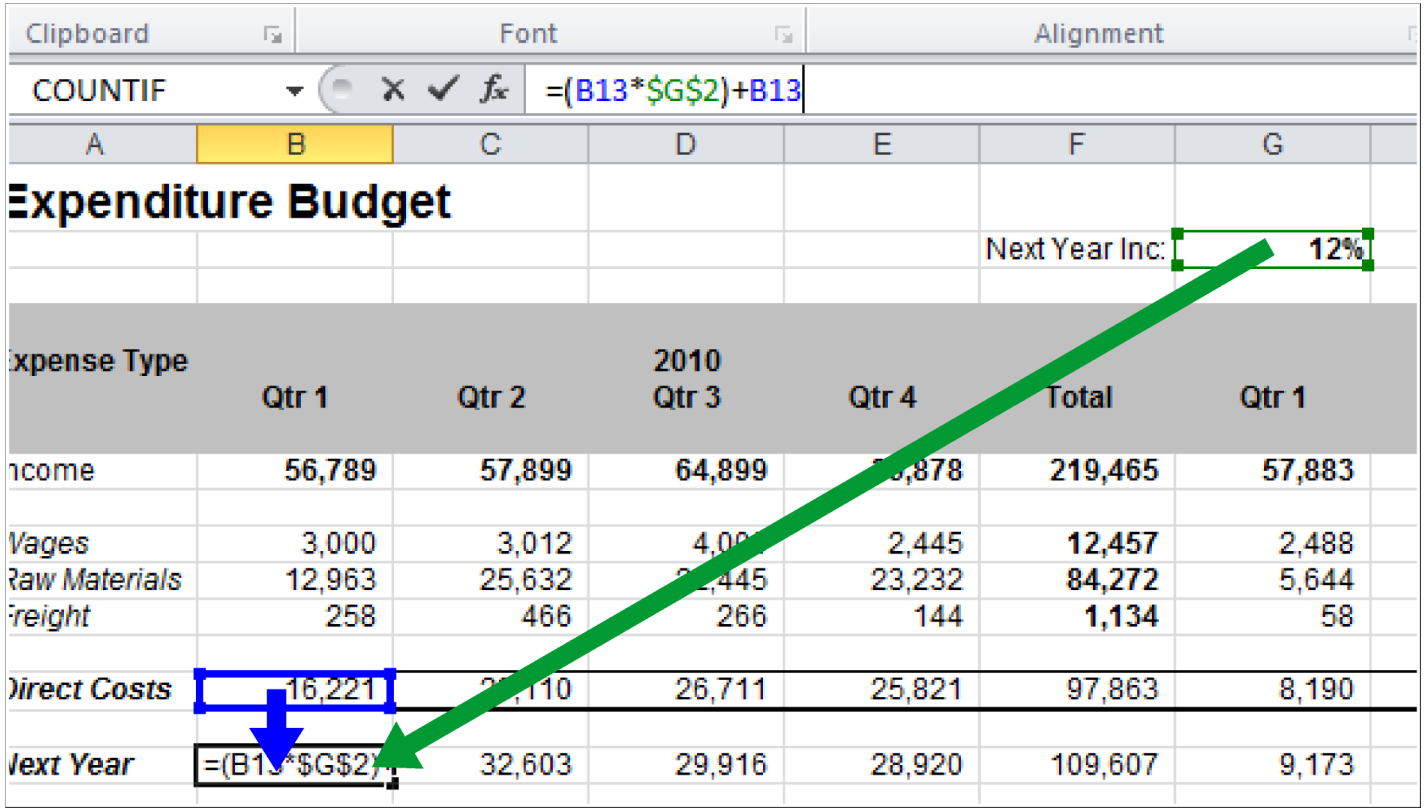
App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Spreadsheet

- Formula components (issue: no repository)
- Data types: numeric, date, string
- Graph defined by 2D/3D cell coordinate references



Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, **Make** etc.

Practice

App Creating vs Programming, Component Programming, Application Building

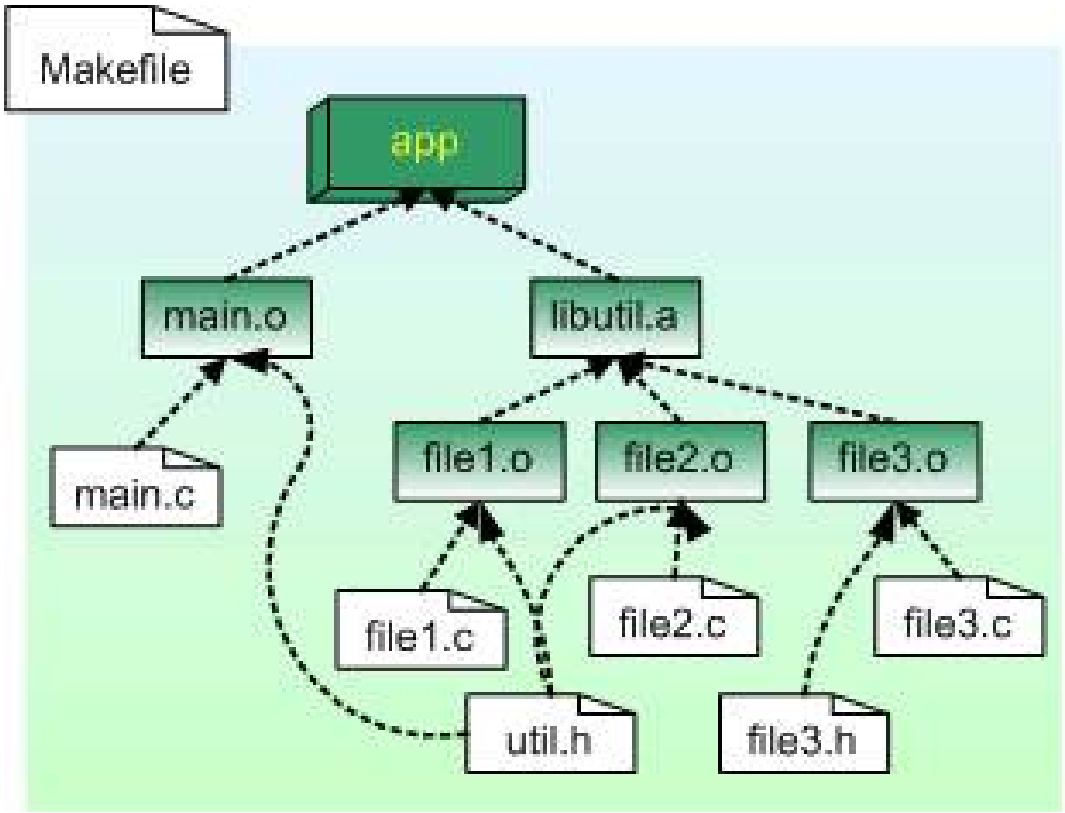
Benefits

Rapid Prototyping, Reusability, Transparency

Make

- Component: job (compiler script)
- Data: file (sources, objects, executable)
- Dependency tree
- Parallel execution

  make -j



Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make **etc.**

Practice

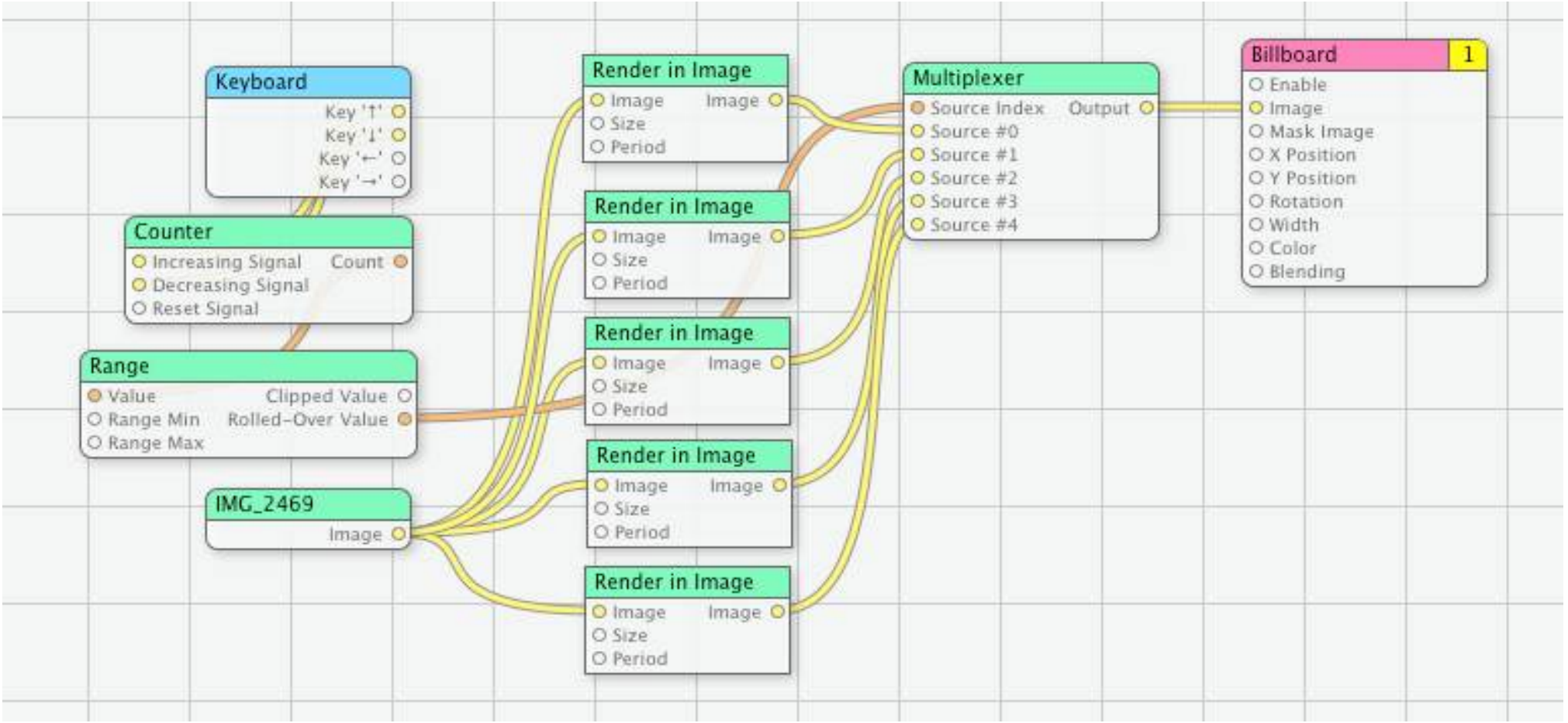
App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

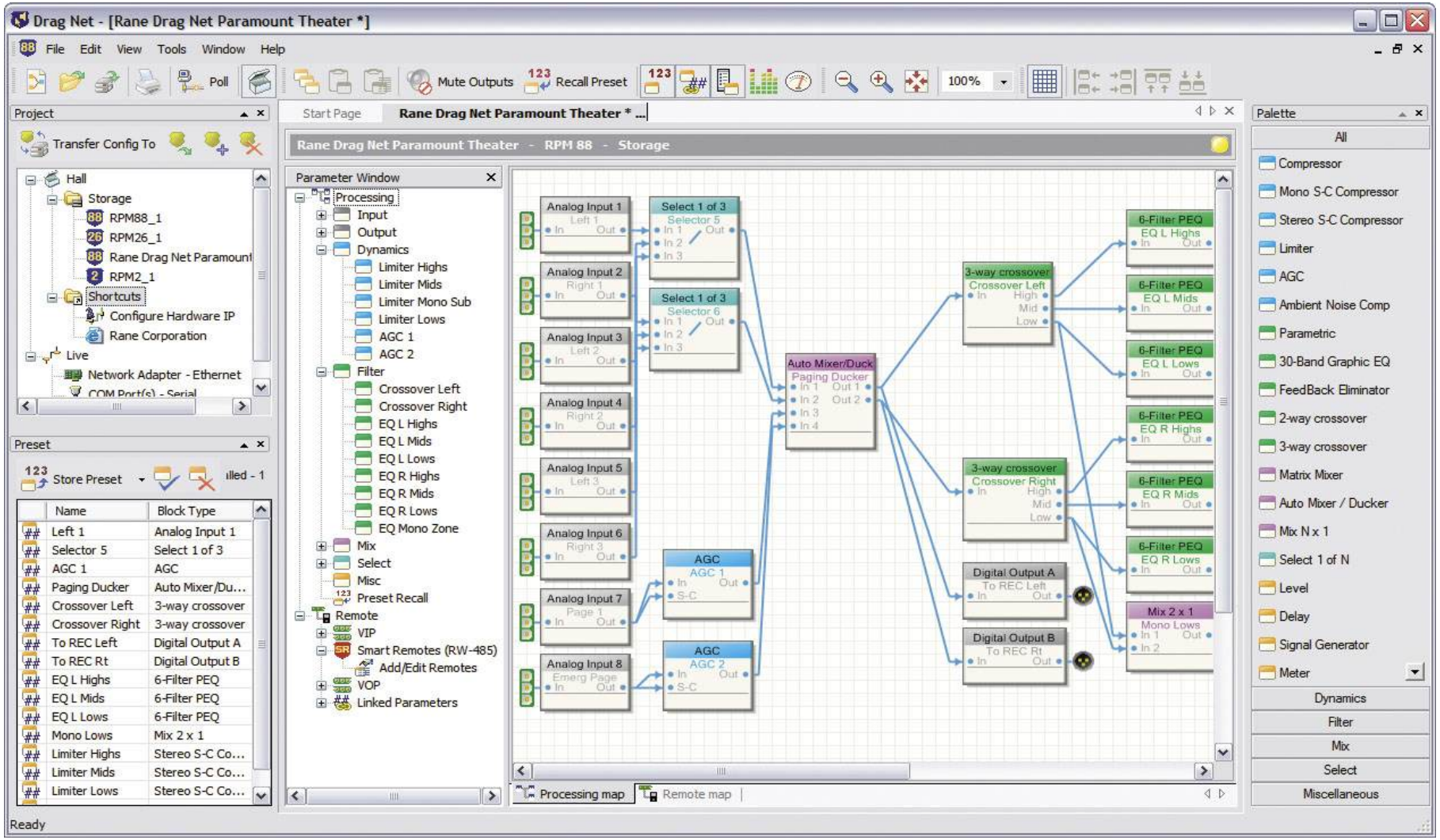
Quartz Composer

- Graphics purpose
- Comes with Mac OS X



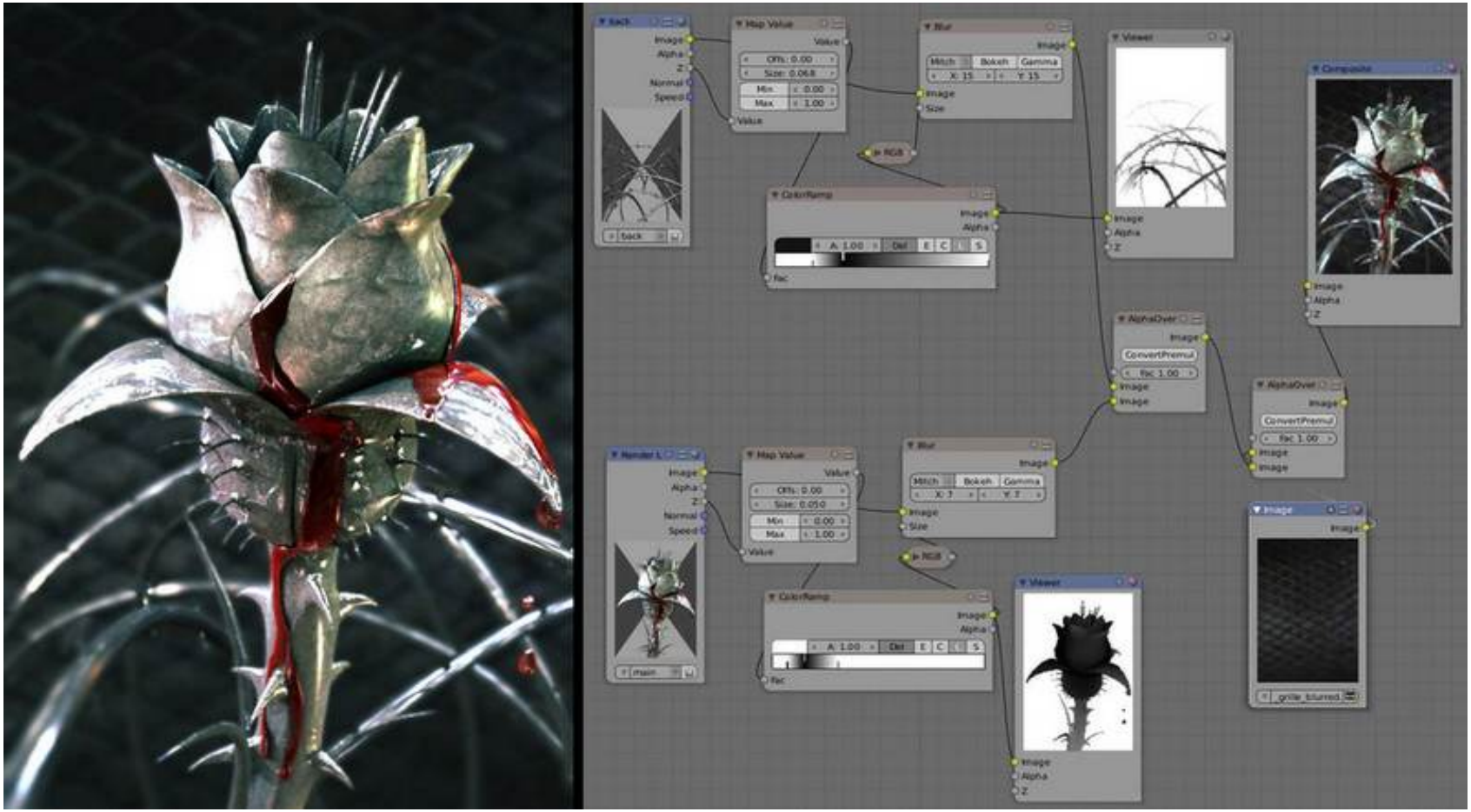
Rane DragNet

Audio system



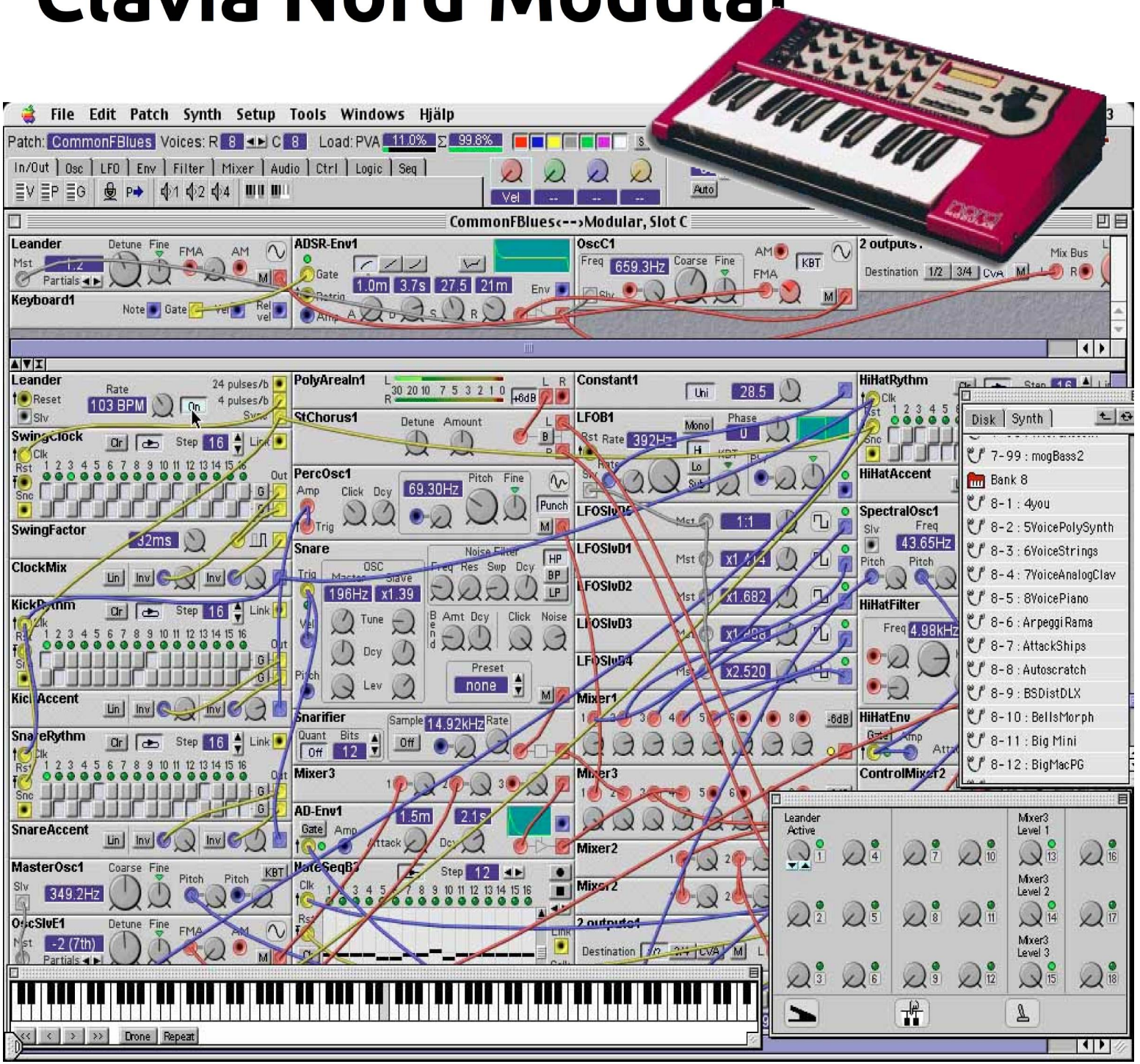
Blender

- Video system
- Open source



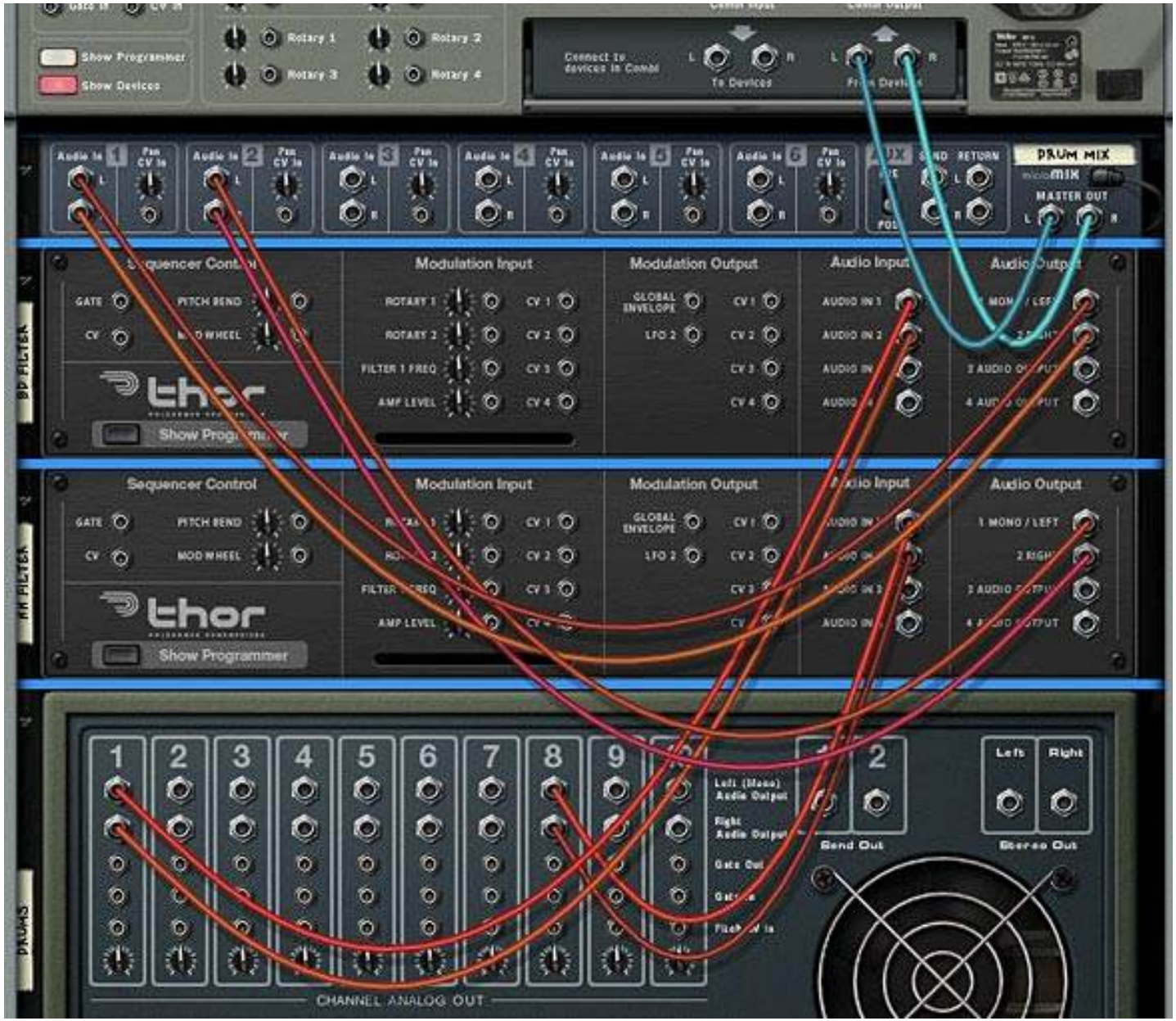
Clavia Nord Modular

- Music
- Win32 editor



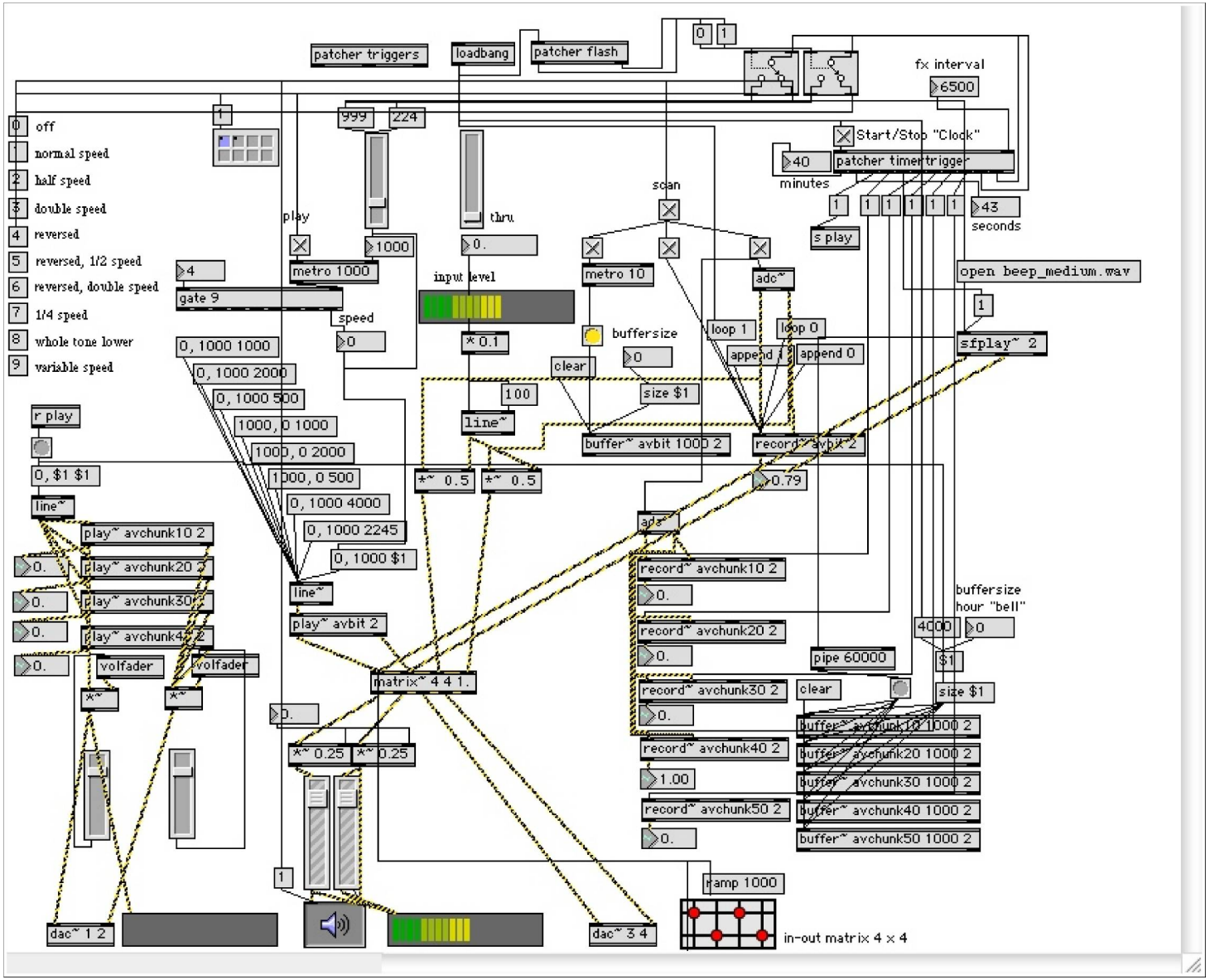
Propellerhead Reason

- Audio workstation
- Rack+wire metaphor



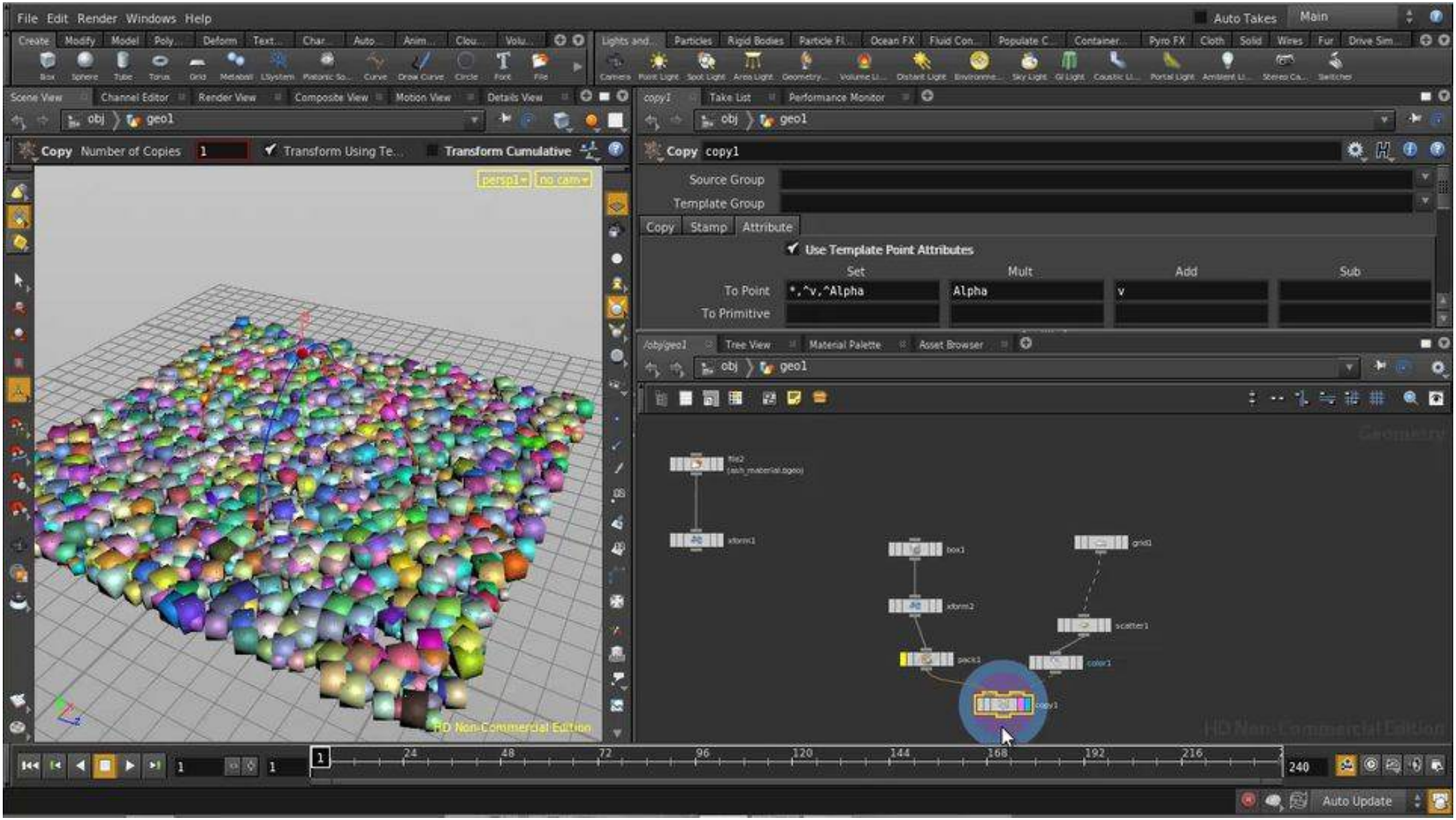
MaxMSP

Audio/video magic



Houdini

3D Animation



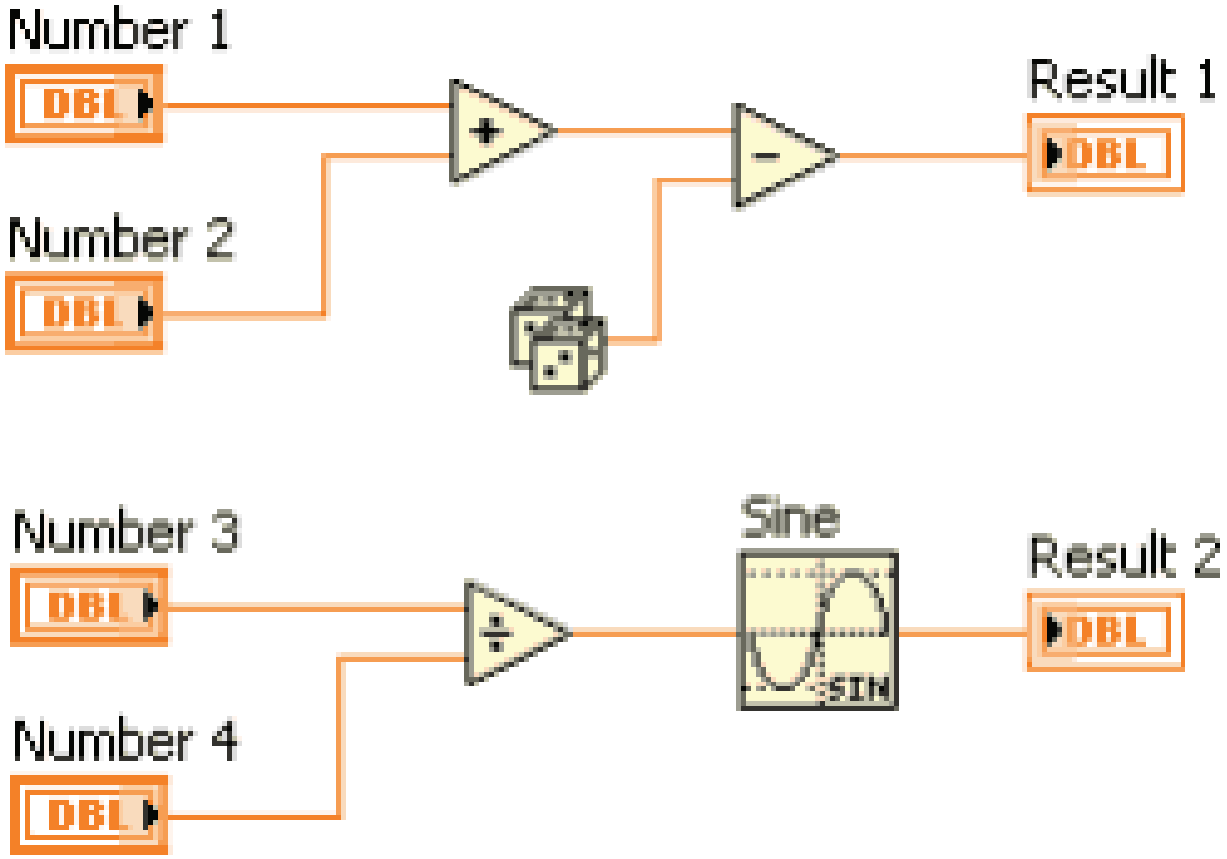
TinyOS

Embedded Systems


```
// CounterSounder
Main.StdControl -> CounterSounderM.StdControl;

// TimerC
CounterSounderM.Timer -> TimerC.Timer[unique("Timer")];
Main.StdControl -> TimerC.StdControl;

// LedsC
CounterSounderM.Leds -> LedsC.Leds;

// Sounder
CounterSounderM.SounderControl -> Sounder.StdControl;
```

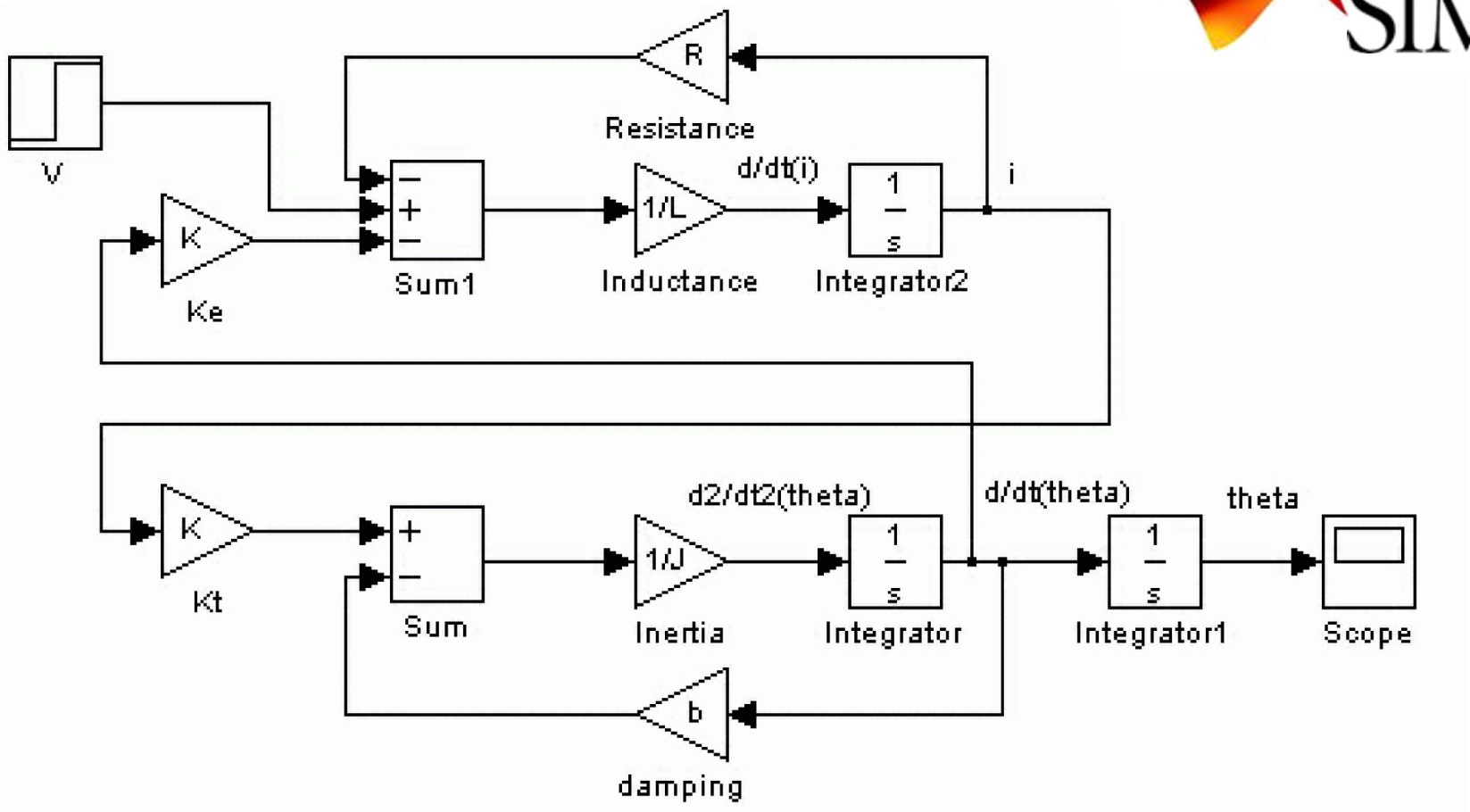
LabView

Math magic



MatLab & SimuLink

Math magic



Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

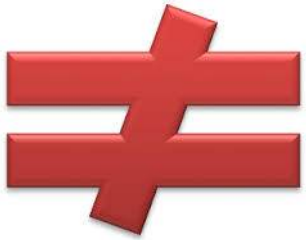
Practice

App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Good News

creating application  programming

People Are Different

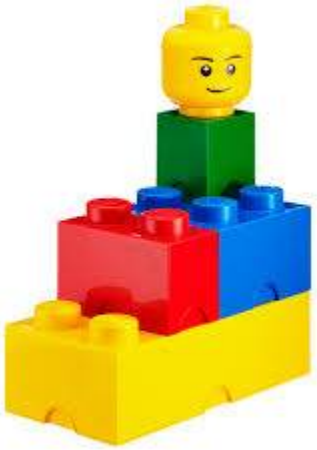
another image, pls

People Are Different

creating application ≠ programming

application builder

domain knowledge



user contact

customization

integration

maintenance

programmer

programming

supporting app builder

separating roles

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

App Creating vs Programming, **Component Programming**, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Component Programming

- Simple, small code (100 – 1000 lines)

  Homeaut.com component sizes (C++, w/o headers):

JamSolver:	497 lines
Scheduler:	628 lines
SimpleSequencer:	815 lines

- Loose coupling: default (Hollywood principle etc.)
- Ready for unit testing
- No customer demands
- No legacy code to learn and modify

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

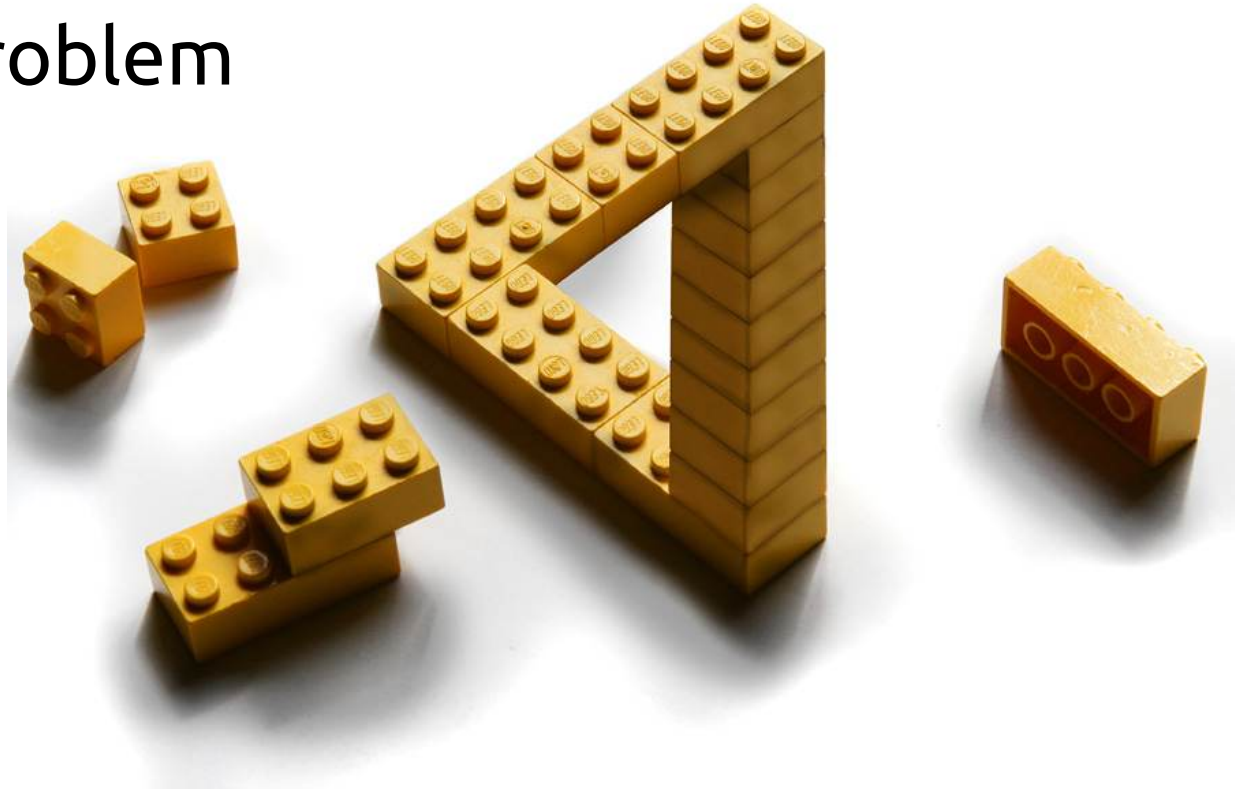
App Creating vs Programming, Component Programming, **Application Building**

Benefits

Rapid Prototyping, Reusability, Transparency

Application Building

- No programming skills required
- Even the user can create apps
- Visual programming
- Convert patterns to composite components
- Focusing on the problem
- Different world



Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

App Creating vs Programming, Component Programming, Application Building

Benefits

Rapid Prototyping, Reusability, Transparency

Rapid Prototyping

- No programming required
- Mock missing components
- Mock missing resources (data source, user input etc.)
- Discover missing components to be implemented

Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

App Creating vs Programming, Component Programming, Application Building

Benefits

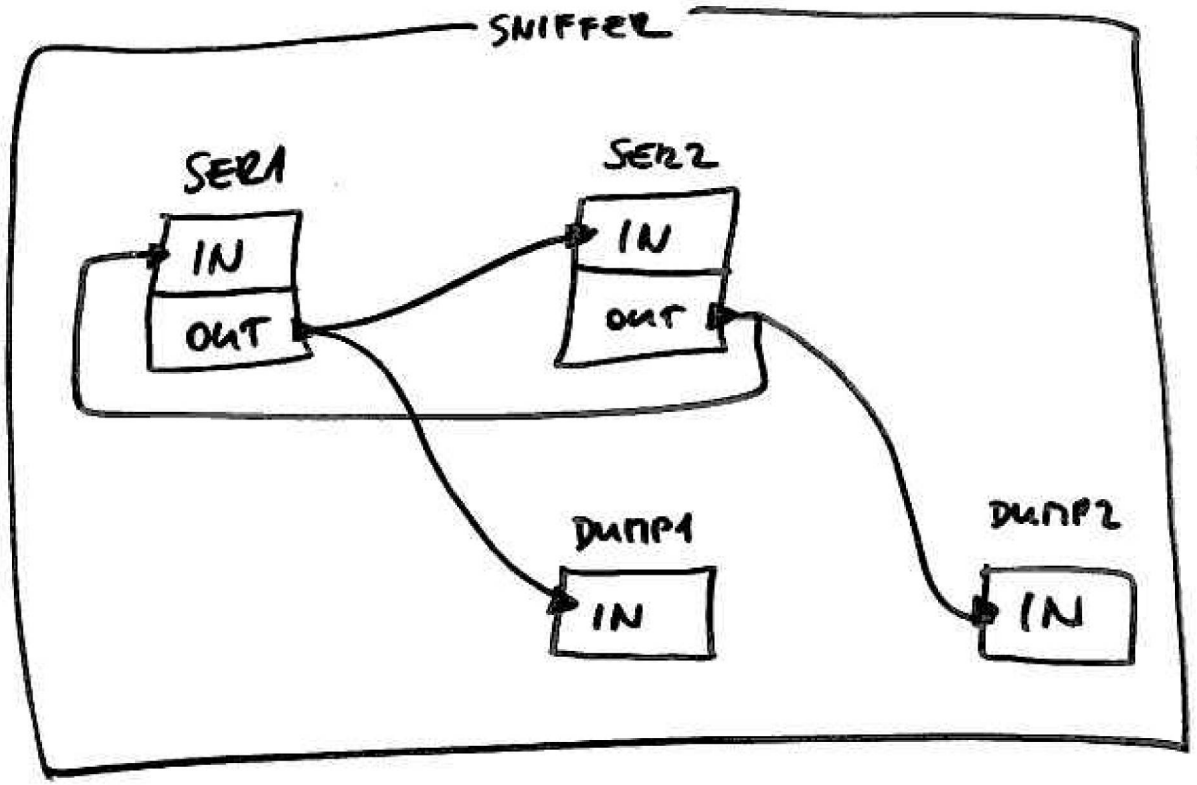
Rapid Prototyping, **Reusability**, Transparency

Reuse. Really.

OOP promised reusability.
It was a lie.

Reusability Example

Serial sniffer with home automation components



Basics

Definition
Component & Port
Data Types
Source, Processor, Sink

Advanced

Component: Native vs Composite
Scheduling: Synchronous vs Asynchronous
Triggering: Push vs Pull
Execution: Parallel, Multi Host

Dataflow Systems

Unix Pipe, Spreadsheet, Make etc.

Practice

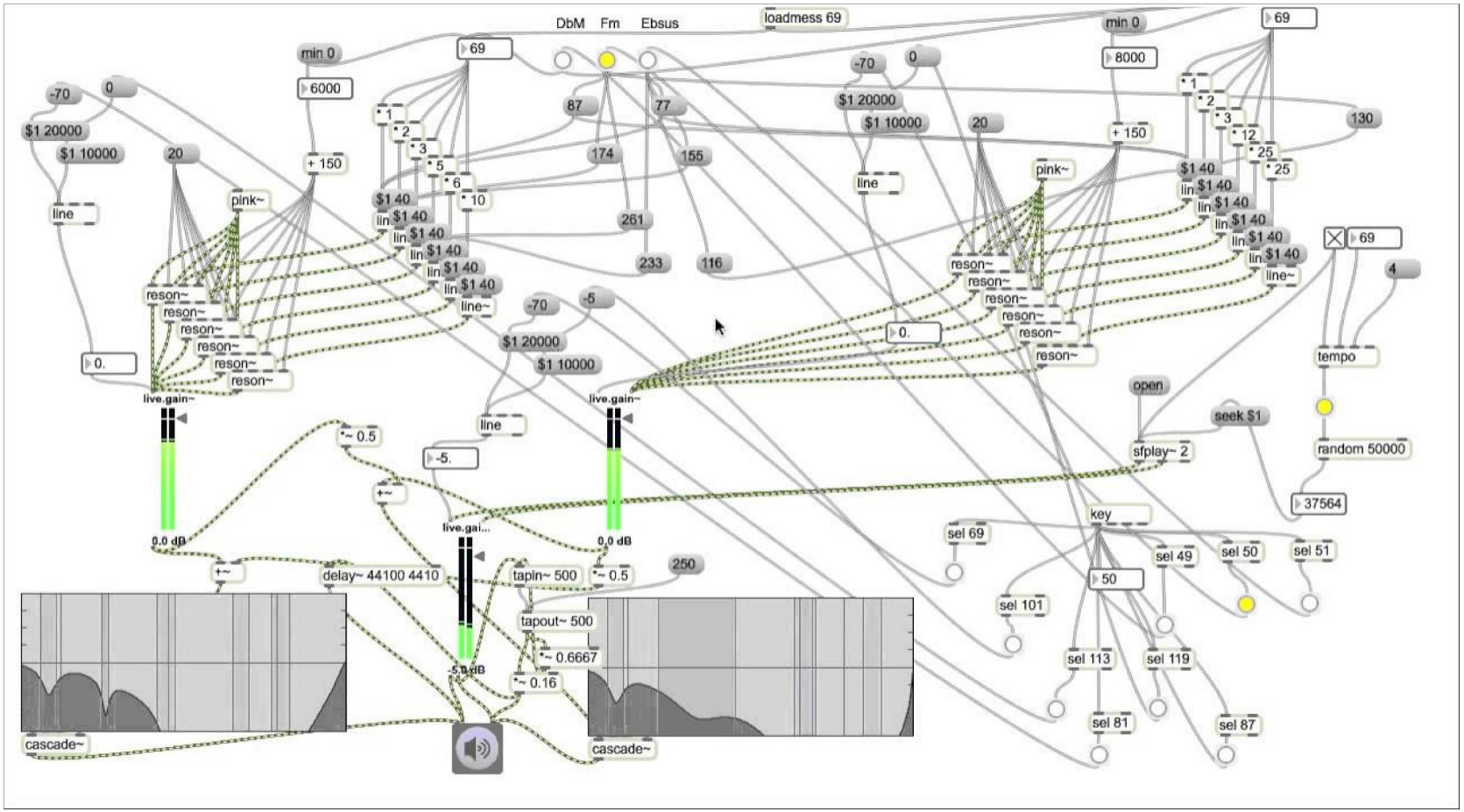
App Creating vs Programming, Component Programming, Application Building

Benefits

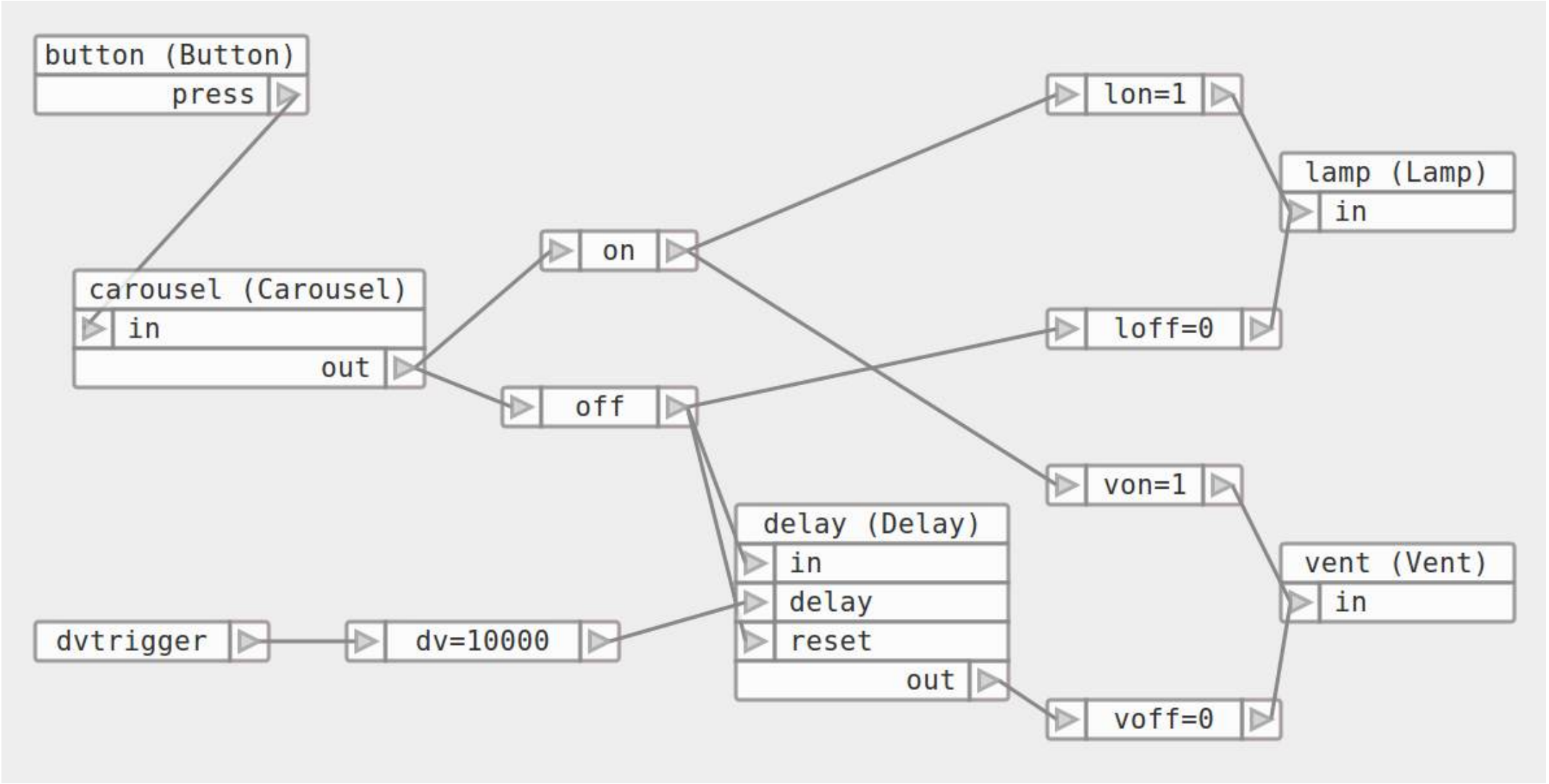
Rapid Prototyping, Reusability, **Transparency**

Transparency

- Automatic documentation of the application
- Well-separated layers



THE END



My favourite application. Can you find the bug?