Modelling with Streams in Daisy
and
The SchemEngine Project

Steven D Johnson
Indiana University Computer Science Department
System Design Methods Laboratory
sjohnson@cs.indiana.edu
www.cs.indiana.edu/~sjohnson

Eric Jeschke
Computer Science Department
The Univerisity of Hawaii, Hilo
jeschke@hawaii.edu
uhhach.uhh.hawaii.edu/~jeschke/

Outline

I. *Background and context*

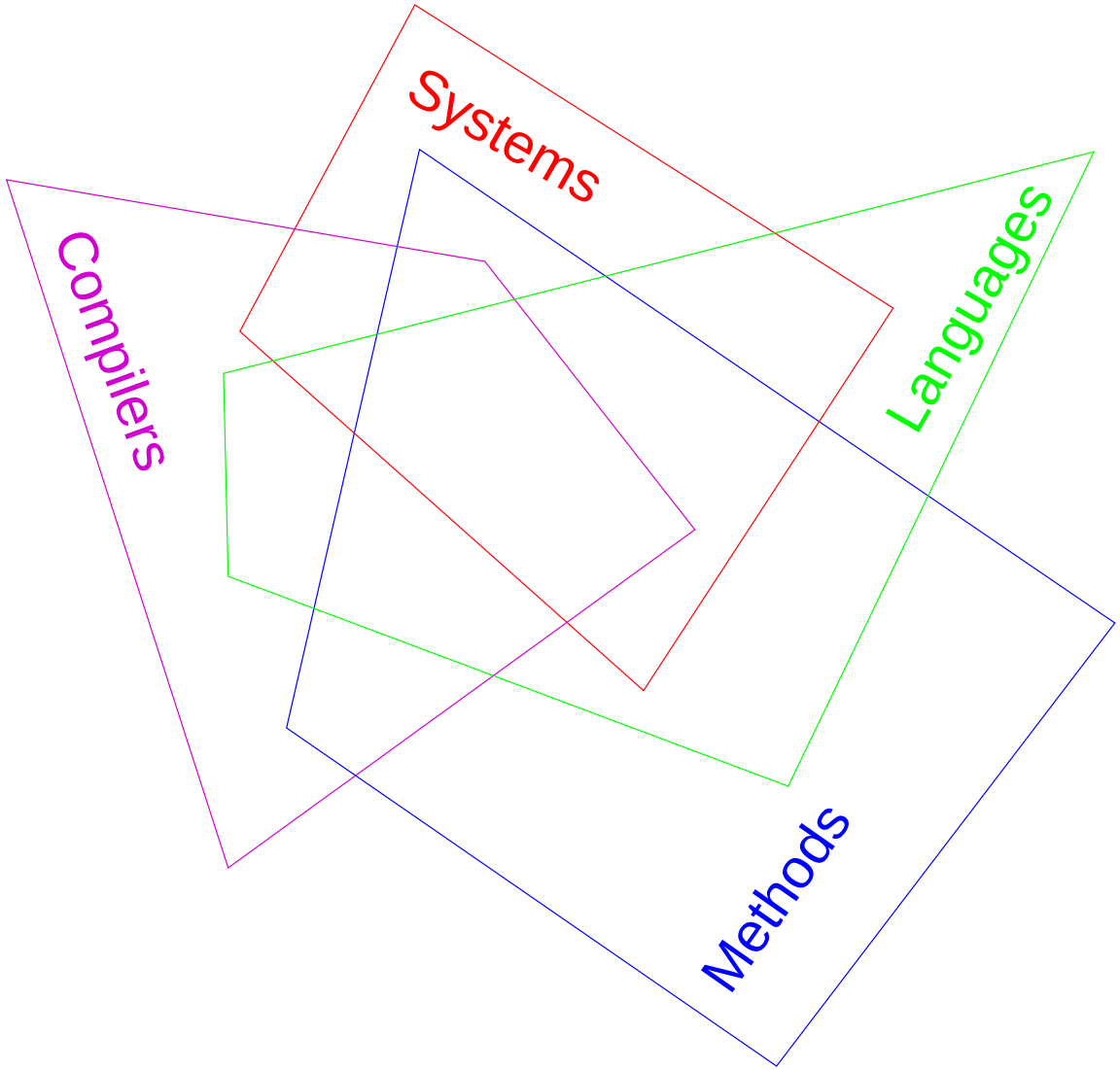
II. *Modeling with streams in Daisy*

- A. Lazy CONS demand-oriented computation, stream I/O, concurrency.
- B. Stream systems.
- C. Some modeling techniques.
- D. Distributed extensions.

III. *The* SchemEngine *Project*

- A. Design derivation
- B. Previous studies, *Schemachine*
- C. *VLISP* [Guttman, Ramsdell, Wand, *L&SC 95]*
- D. *SchemEngine* objectives in integrated formal analysis
- E. Toward an integrated codesign enviornment

I. Background and context, 1975–now



Background (languages)

75-80	Functional programming languages
	Fridman & Wise, "CONS should not evaluate its arguments"
	Applicative programming for systems
	Extensions for indeterminacy, `set:` $[\alpha\ \beta\ \cdots]$
	Suspending construction model [continuations, engines in Scheme]
	Semantics ?!
80-85	Daisy/DSI in Unix
	Stream-based I/O From concurrency to parallelism
	O'Donnell, programming environments, hardware models
85-90	Parallel DSI [Jeschke95]
	Language-driven architecture $\Rightarrow$ design derivation
90-95	Bounded speculation
	Windows on the data space
	Daisy/DSI for small-scale MIMD?
95-00	Distributed demand propagation
	HW models as case studies

Background (methods)

75-80	Functional programming methods
	Prosser & Winkel, structured digital design, ASMs
80-85	Compiler derivation [Wand]
	Combinator factorization $\Rightarrow$ "machine"
	Stream systems and synchronous hardware
	Formalized synthesis
85-90	Digital design derivation
	Functional/algebraic formalism
	GC-PLD, GC-VLSI, SECD
90-95	DDD $\Rightarrow \cdots$ [Bose, Tuna, Rath, W.Hunt]
	Heterogeneous reasoning
	FM8502, FM9001-DDD, Schemachine
	DDD vis-a-vis PVS, coinductive types [Minor]
95-00	Tools
	Behavior tables
	etc.

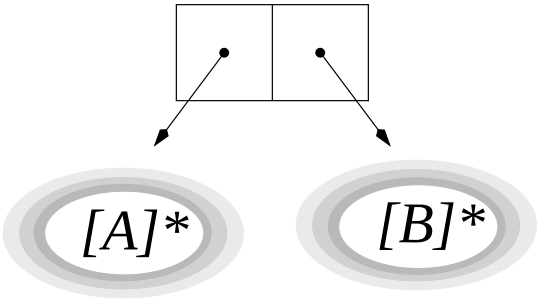
II. Modeling with streams in Daisy

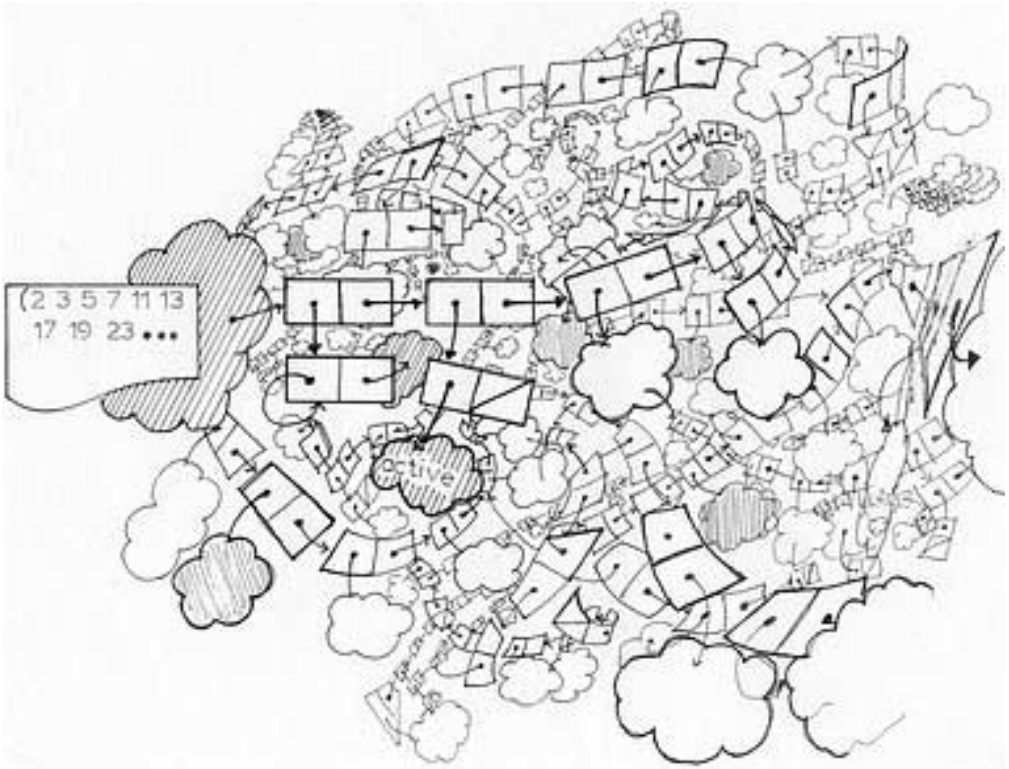
Animation is a key aspect of functional formalism.

- Suspending CONS, demand oriented computation.
- List (stream) representation of I/O
- Concurrency construct, `set`
- Windowing support

Suspending Cons

Delays? No.
Futures? No.
Engines? Almost.
Demand driven computation? No.
Demand oriented computation ···
bounded speculation

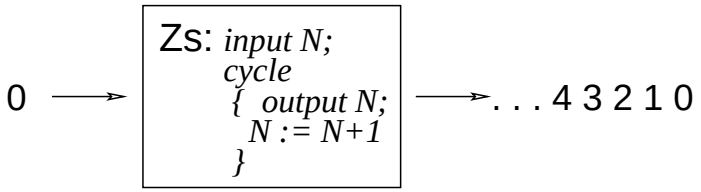


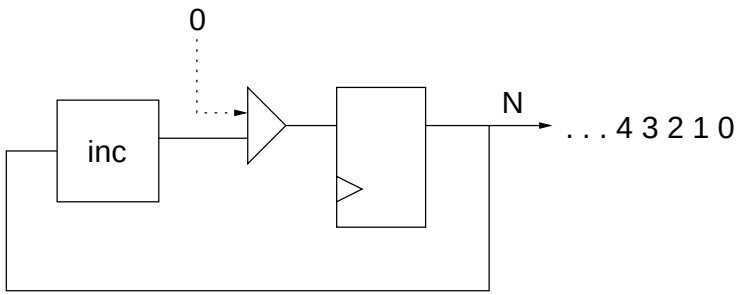


DSI:

- ◇ Heap based
  symbolic multiprocessing
- ◇ Transparent
  process management

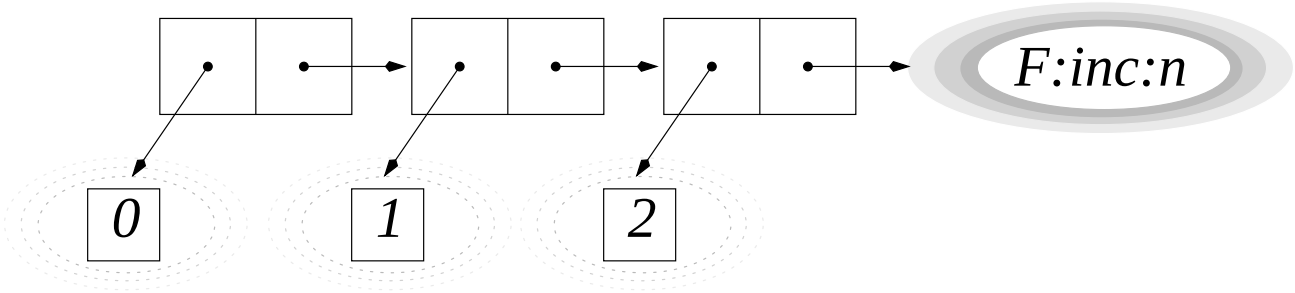
Processes as streams





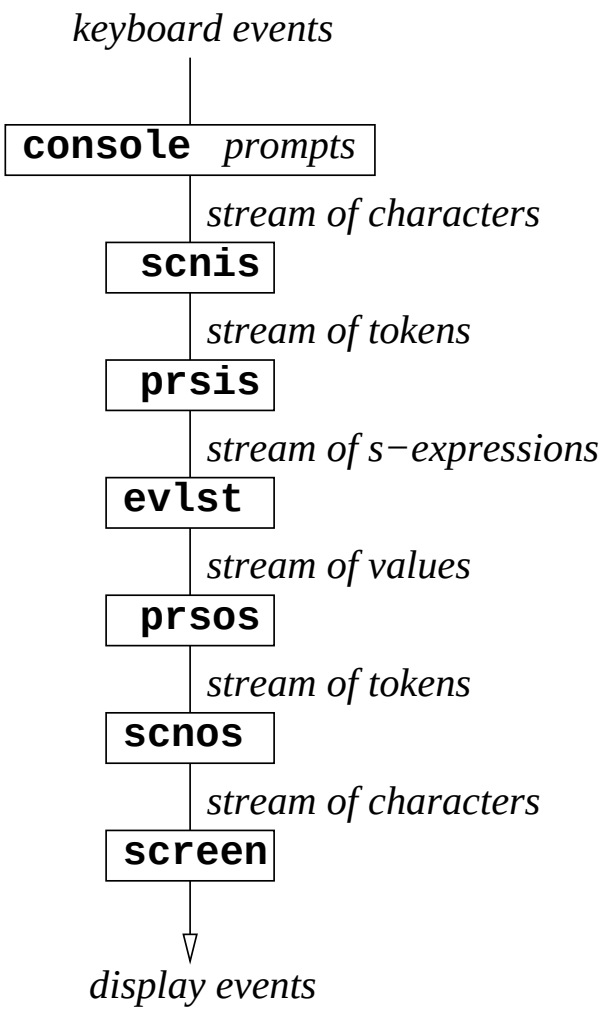

```
F:n = [n !  F:inc:n]
```

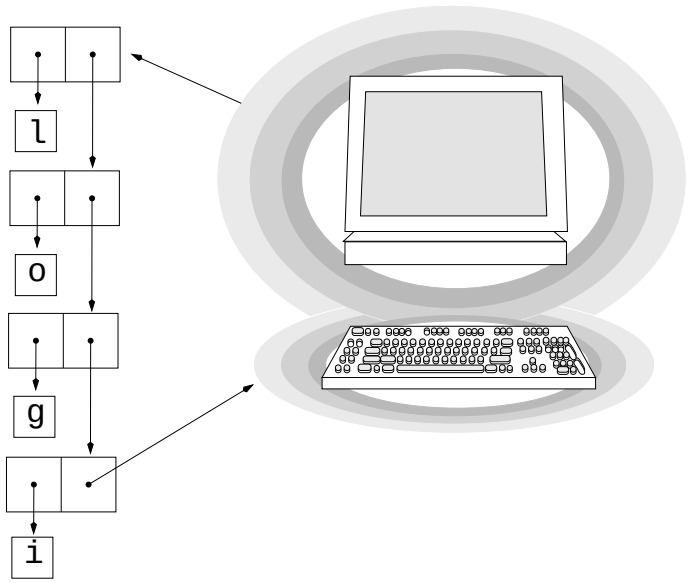
```
N = [0 !  (map:inc):N]
```



Stream (i.e. lazy-list) based I/O

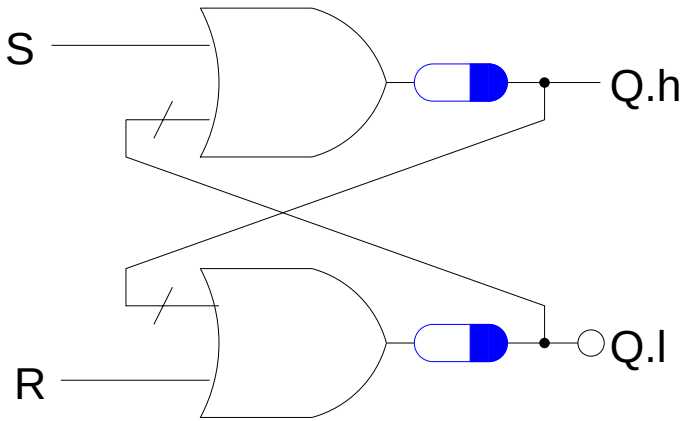
I/O synchronization and suspension coercion use the same synchronization mechanism (e.g. a *presence bit*)





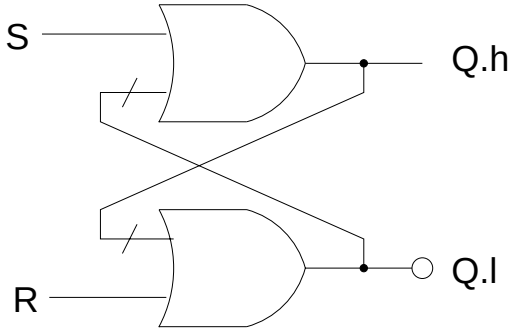
`screen:console:`*prompt*

```
NOT = (map:not)
OR = (mapxps:or)
```




```
RSFF = \[S R].
 rec
   Qh = [0 ! OR:[S NOT:Ql]]
   Ql = [1 ! OR:[S NOT:Qh]]
  in
   [Qh Ql]


bit-filter = \[C ! Cs].
   if:[ same?:[C "0"] [0 ! bit-filter:Cs]
        same?:[C "1"] [1 ! bit-filter:Cs]
                      bit-filter:Cs
       ]


xps:
RSFF:[bit-filter:console:"NL S: "
      bit-filter:console:"NL R: "]
```

```
& xps:
RSFF:[bit-filter:console:"NL S: "
      bit-filter:console:"NL R: "]
```



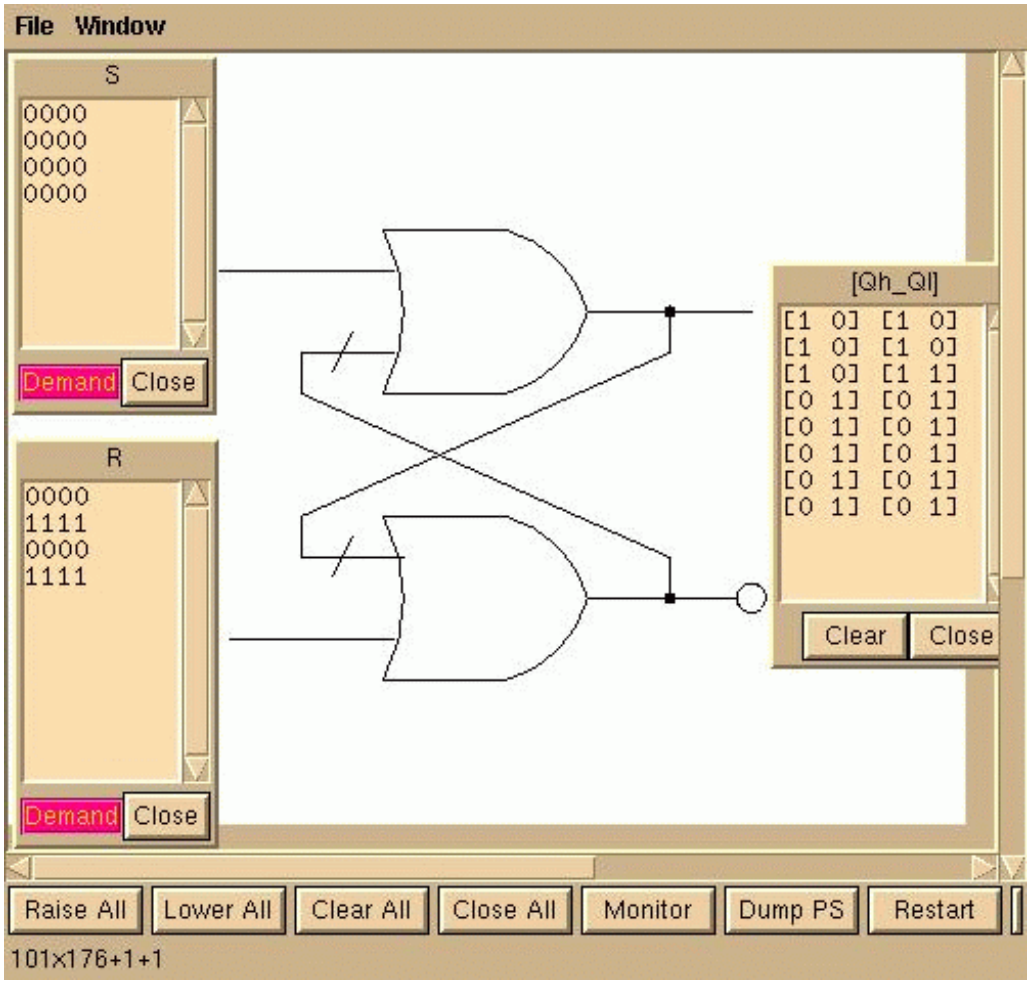

```
[
S: 0 0 0 0 0 0 0 0
[1 0] [
R: 0 0 0 1 1 1 0 0
 0] [1 0] [1 0] [1 1] [0 1] [0 1] [0 1]
S: 0 0 0 1 1 1 0 0 0
 [0 1] [0
R: 0 0 0 0 0 0 0 0 0
 1] [0 1] [0 1] [1 1] [1 0] [1 0] [1 0] [1 0]
S:
```

Widget devices:

`wndi`: $name \rightarrow char^*$

`wndo`: $[name, char^*] \rightarrow$ `[]`

Also: `filei/o`, `socketi/o`*, `execi/o`*, ...




```
wndo:[ "[Qh Ql]"
       prsos:
        scnos:
         xps:
          RSFF:[ bitfilter:wndi:"S"
                 bitfilter:wndi:"R" ]]
```

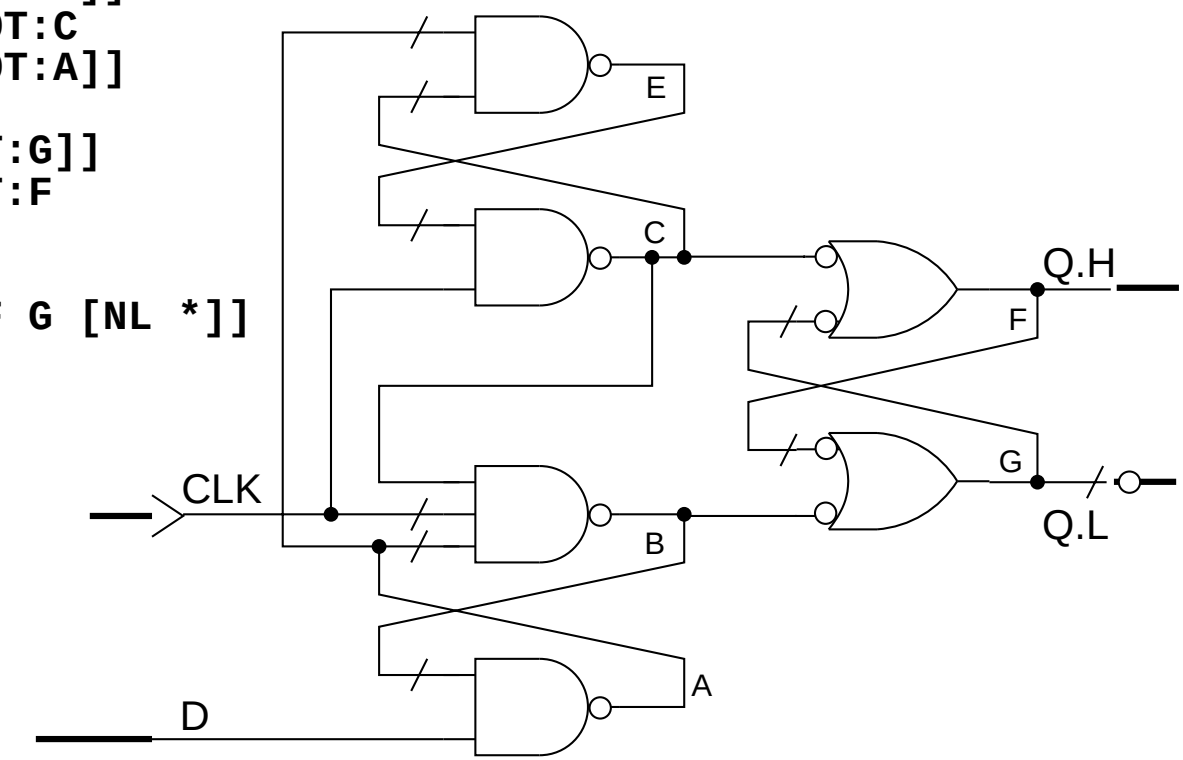
Concurrency

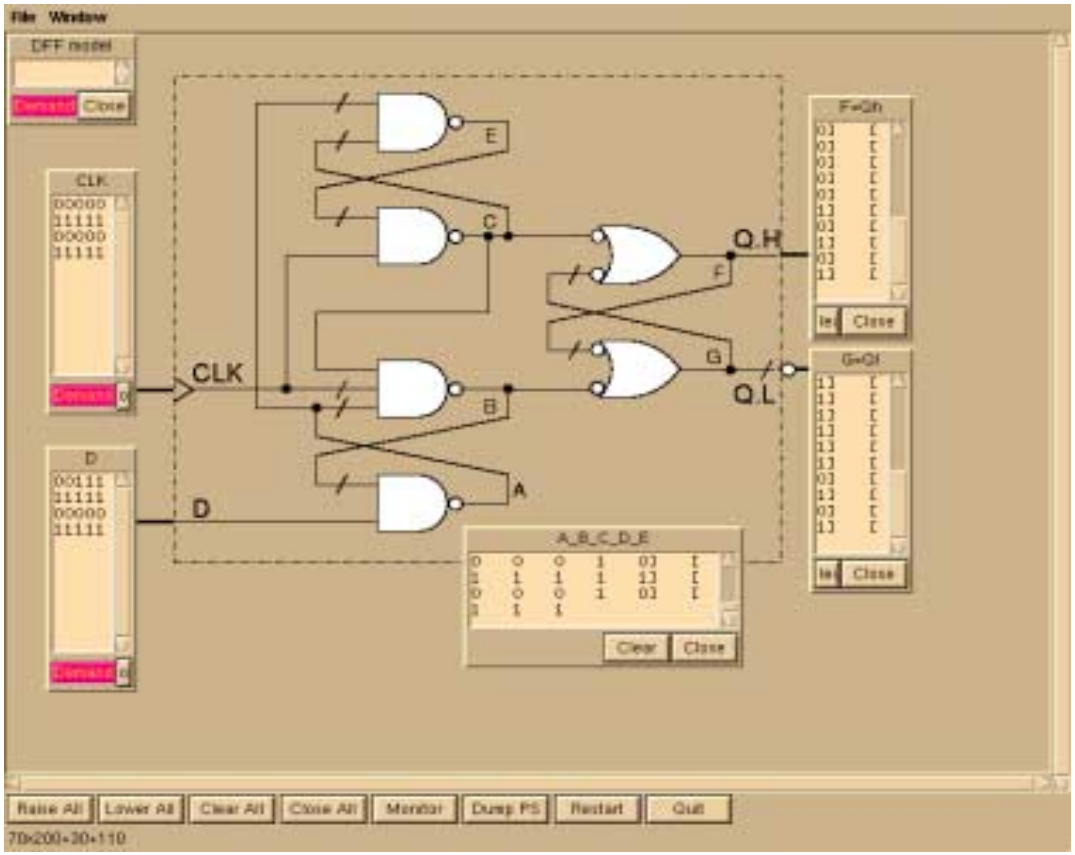
Implicit through bounded speculation

Explicit through (constructs such as) `set`

- Notation: expression A, computation α, result a.
- `set`: $[A\ \ B\ \ C]$ constructs list object $L = [\alpha\ \ \beta\ \ \gamma]$
- L becomes manifest as $[a\ \ b\ \ c]$, or $[b\ \ a\ \ c]$, or $[b\ \ c\ \ a]$, *etc.*,
- There is an imprecise operational relationship with computational effort.
- *Semantics (?!)*

```
DFF = \[a b c e f g].
      \[CLK D].
 rec
  A = [a ! AND:[D NOT:B]]
  B = [b ! AND:[CLK
                AND:[NOT:A
                     NOT:C]]]
  C = [c ! AND:[CLK
                NOT:E]]
  E = [e ! AND:[NOT:C
                NOT:A]]
  F = [f ! OR:[C
               NOT:G]]
  G = [g ! OR:[NOT:F
               B]]
 in
  [CLK D A B C E F G [NL *]]
```





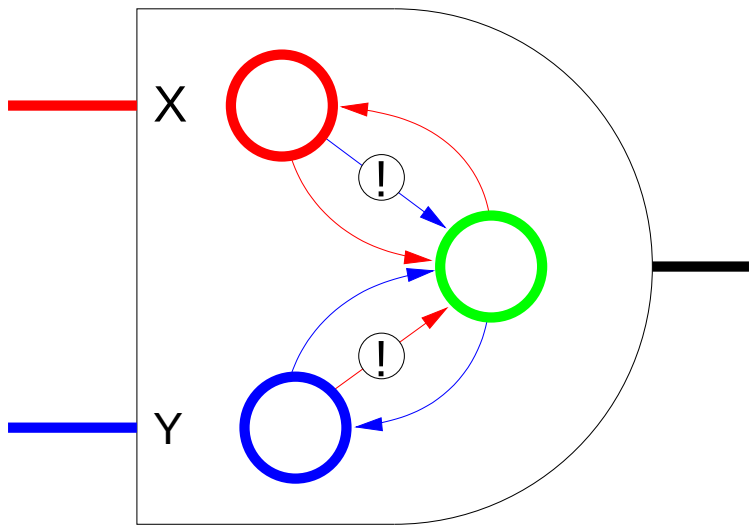

```
let
  [CLK D A B C E Ql Qh NLs]
  = (DFF:[2 2 2 2 1 0]):
      [ f-t:wndi:"CLK"
        f-t:wndi:"D"
       ]
in
  head:
  set:
   [ consume:wndi:"DFF model"
     wndo:["A_B_C_D_E" s-p-x:[NLs A B C D E]]
     wndo:["F=Qh"      s-p-x:[NLs Qh]]
     wndo:["G=Ql"      s-p-x:[NLs Ql]]
    ]

consume = \Cs. if:⌈ nil?:Cs ⌈⌉ consume:tail:Cs ⌋
```

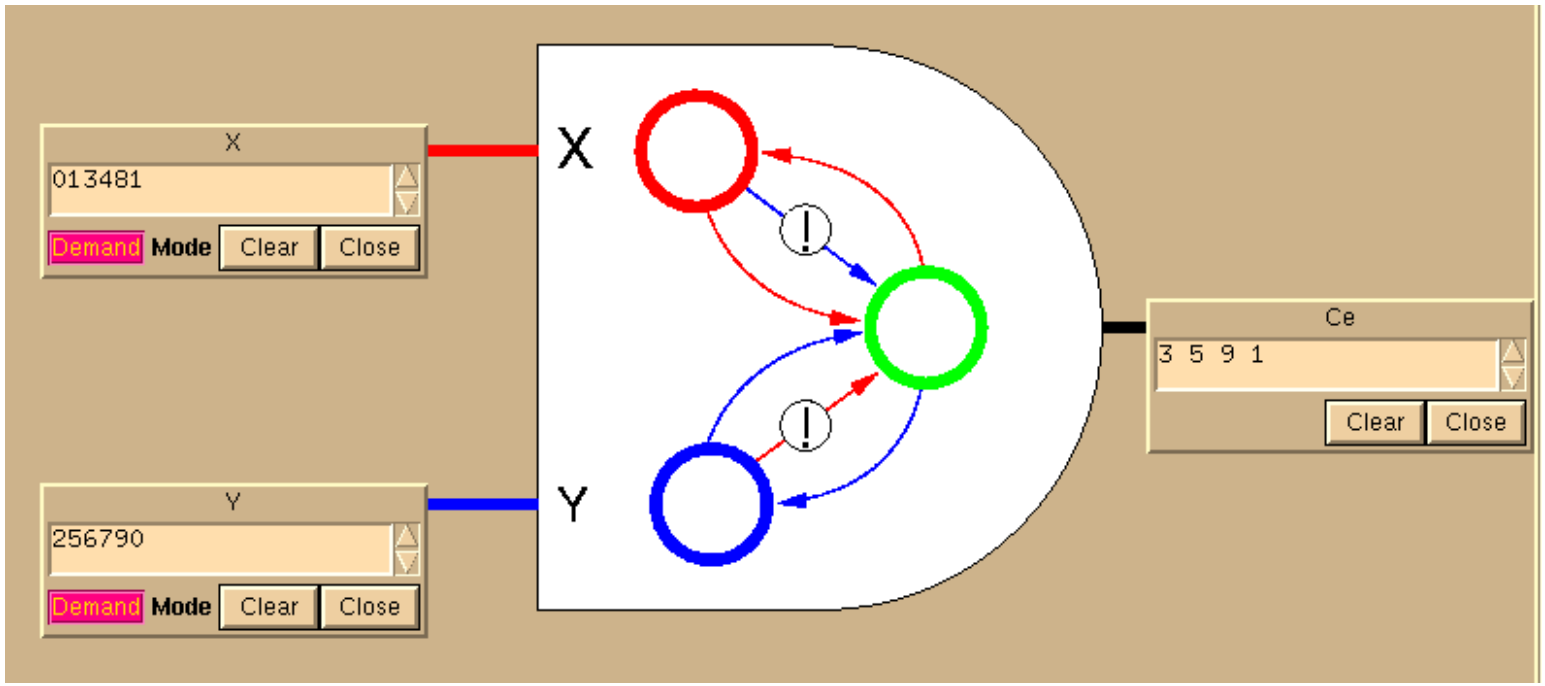
Asynchronous interactions

The pseudo-function `gif` (*"guarded"* `if`) takes a list of guarded expressions, `[ ... [`g_k v_k`] ... ]` and returns one of the values, v_j, whose guard is *true*.

```
   | FAIL: a distinguished value
   | guard:[Bool  Value] -->  Value + %FAIL%
   | gif:{[Bool , Value] ... [Bool, Value]} --> Value + %FAIL%

def gif = \Guards.
  rec
   GUARD = \[Test Result].
              if Test then [Result] else FAIL

   LOOP = \Guards = [G ! Gs].
     if      nil?:Guards
      then    FAIL
     else if fail?:G
      then    LOOP:Gs
      else    head:G
  in
   LOOP:set:(map:GUARD):Guards
```

```
Ce = rec
    Se = \[ [Xh ! Xt] = Xs [Yh ! Yt] = Ys ].
          gif:[ [present?:Xh Sy:[Xt Ys]]
                [present?:Yh Sx:[Xs Yt]]

    Sx = \[ [Xh ! Xt] = Xs [Yh ! Yt] = Ys ].
          gif:[ [present?:Xh [Xh ! Se:[Xt Ys]]]
                [present?:Yh Se:[Xs Yt]]

    SY = \[ [Xh ! Xt] = Xs [Yh ! Yt] = Ys ].
          gif:[ [present?:Xh Se:[Xt Ys]]
                [present?:Yh [Yh ! Se:[Xs Yt]]]
   in Se
```

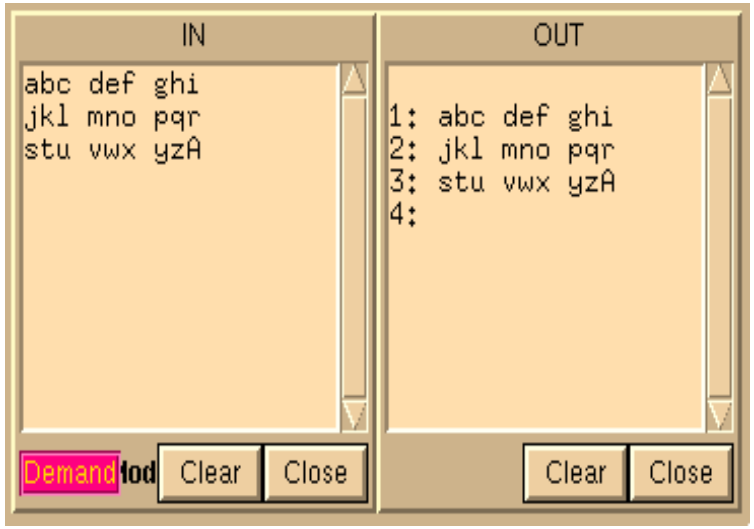


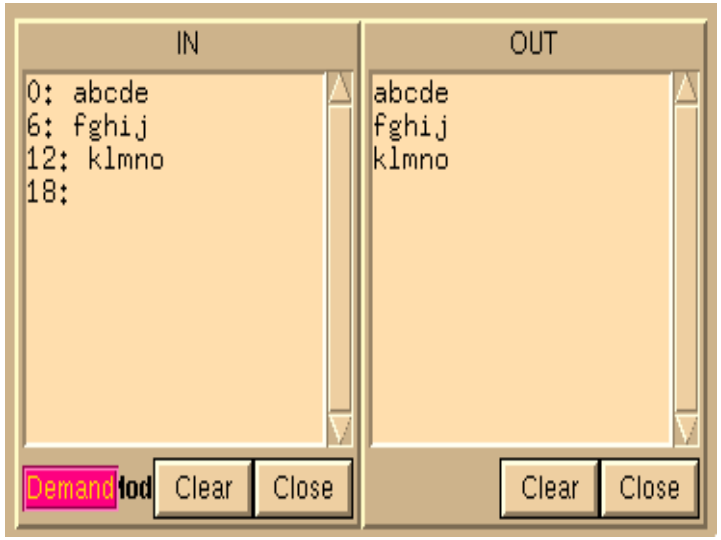

X
013481
Demand
Mode
Clear
Close
Y
256790
Demand
Mode
Clear
Close
X
Y
Ce
3 5 9 1
Clear
Close

Interaction refinements

```
wndo:[
 "OUT"
 interleave_prompts:[
     [NL ! wndi:"IN"]
     ints:1]]
```

```
rec
  Cs = wnd:["IN"
            insert_prompts:
              [NL ! Cs]
              ints:0]]
in
  wndo:["OUT" Cs]
```

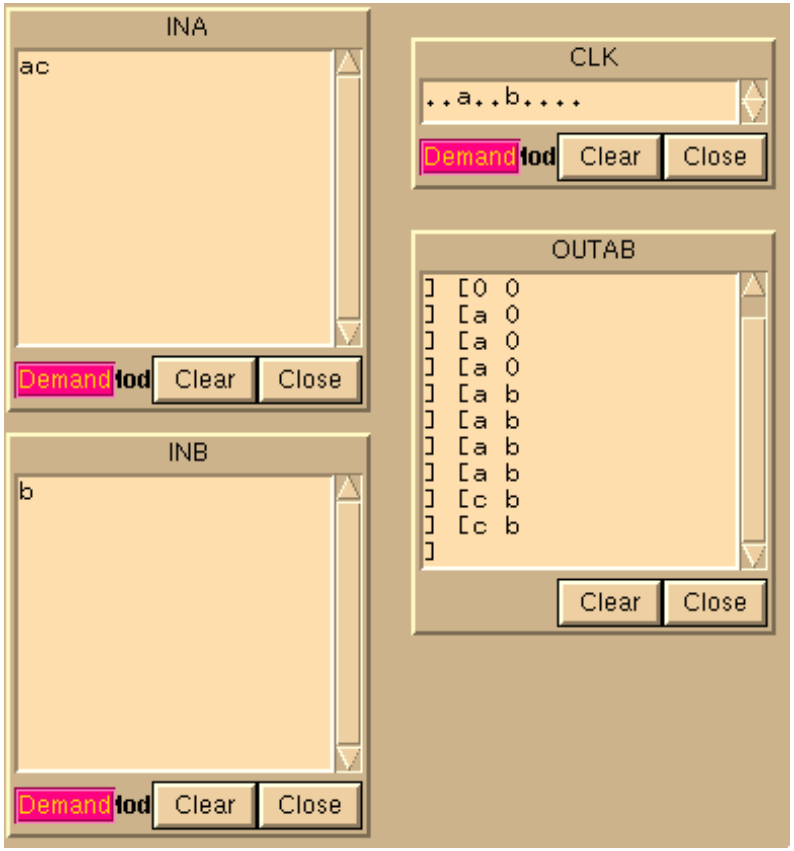




Refinements

```
source = \[D Vs CLKs].
    gif:[
         [ present?:head:Vs
             [head:Vs ! source:[head:Vs  tail:Vs  CLKs]]]
         [present?:head:CLKs
             [D       ! source:[D  Vs  tail:CLKs]]]] ]
```

```
rec
  CLK = wndi:"CLK"
  INA = source:["?" wndi:"INA" CLK]
  INB = source:["?" wndi-lg:"INB" CLK]
in
  wndo:["OUTAB" s-p-x:[INA INB NLs]]
```



Modeling systems with Daisy, conclusions

Daisy/DSI is not a production language. It is an experimental vehicle.

A central motivation of the work has been to explore implications of demand-oriented computation on architecture.

We are currently looking at distributed modeling and *demand propagation across networks*.

We would like to contribute to a modeling methodology based on functional expressions and streams.

One topic of interest is the proper abstraction of *devices* (given that they have *effects*).

output: a (unit?) function that

merges all streams it is applied to?

input: an (nullary?) function that *splits* on demand?

Is *bidirectionality* basic (*sockets*, *dialogues* [O'Donnell])?

III. The SchemEngine Project

Objectives [Johnson, IUTR 544]

- Advancing design derivation
- System-level formal analysis
- From semantics to hardware
- Heterogeneous reasoning
- Embedded applications (!?)
- Foundations for high-confidence

DDD studies in language-driven architecture

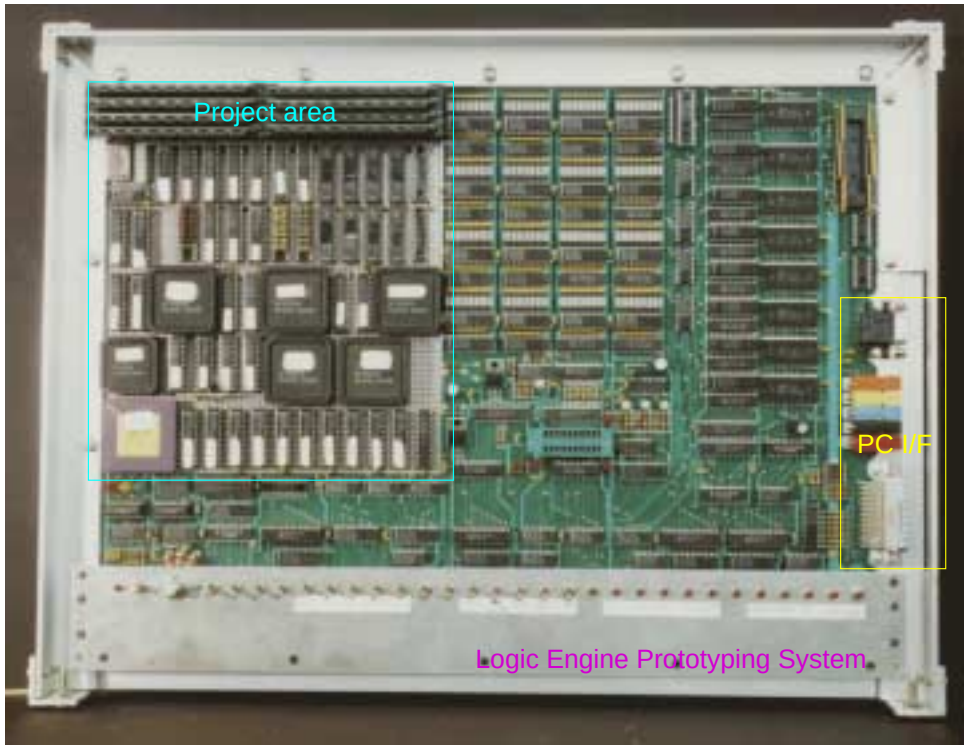
`www.cs.indiana.edu/hmg/`

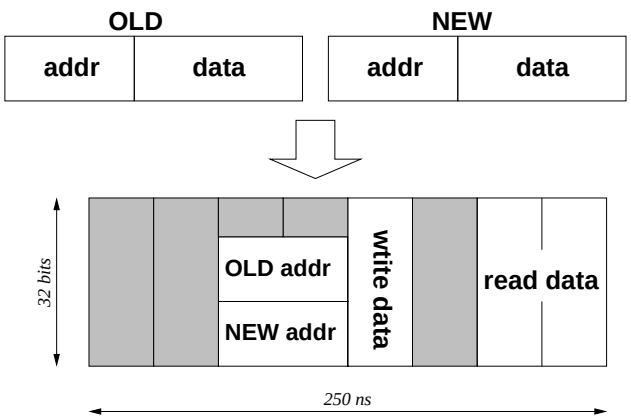
- Garbage collectors in PLDs, VLSI, FPGAs [Boyer 86-90]
- SECD computer [Wehrmeister 89]
- Schemachine [Burger 94]

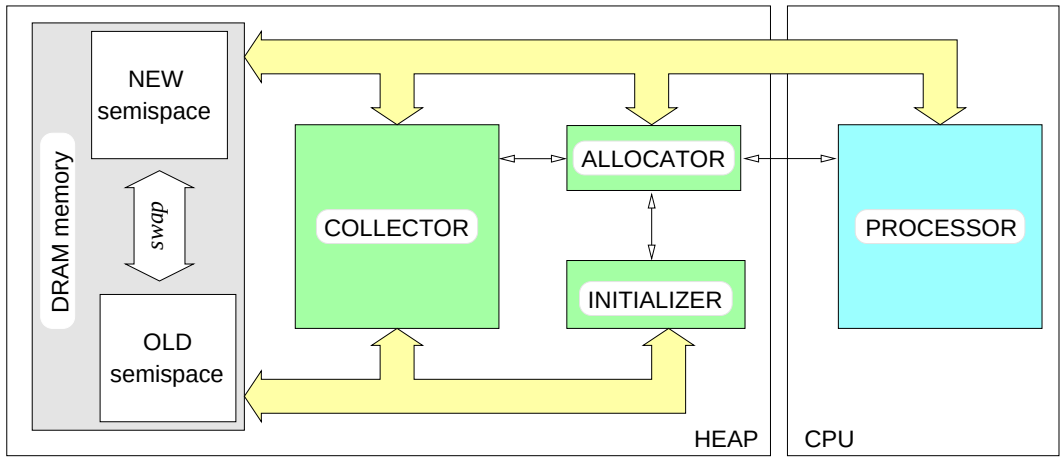
The bigger picture

- Compiler correctness [Wand, Clinger 80-85]
- Compiler *derivation* [Wand 80-85]
- Scheme based methodology and pedagogy
- Codesign tools

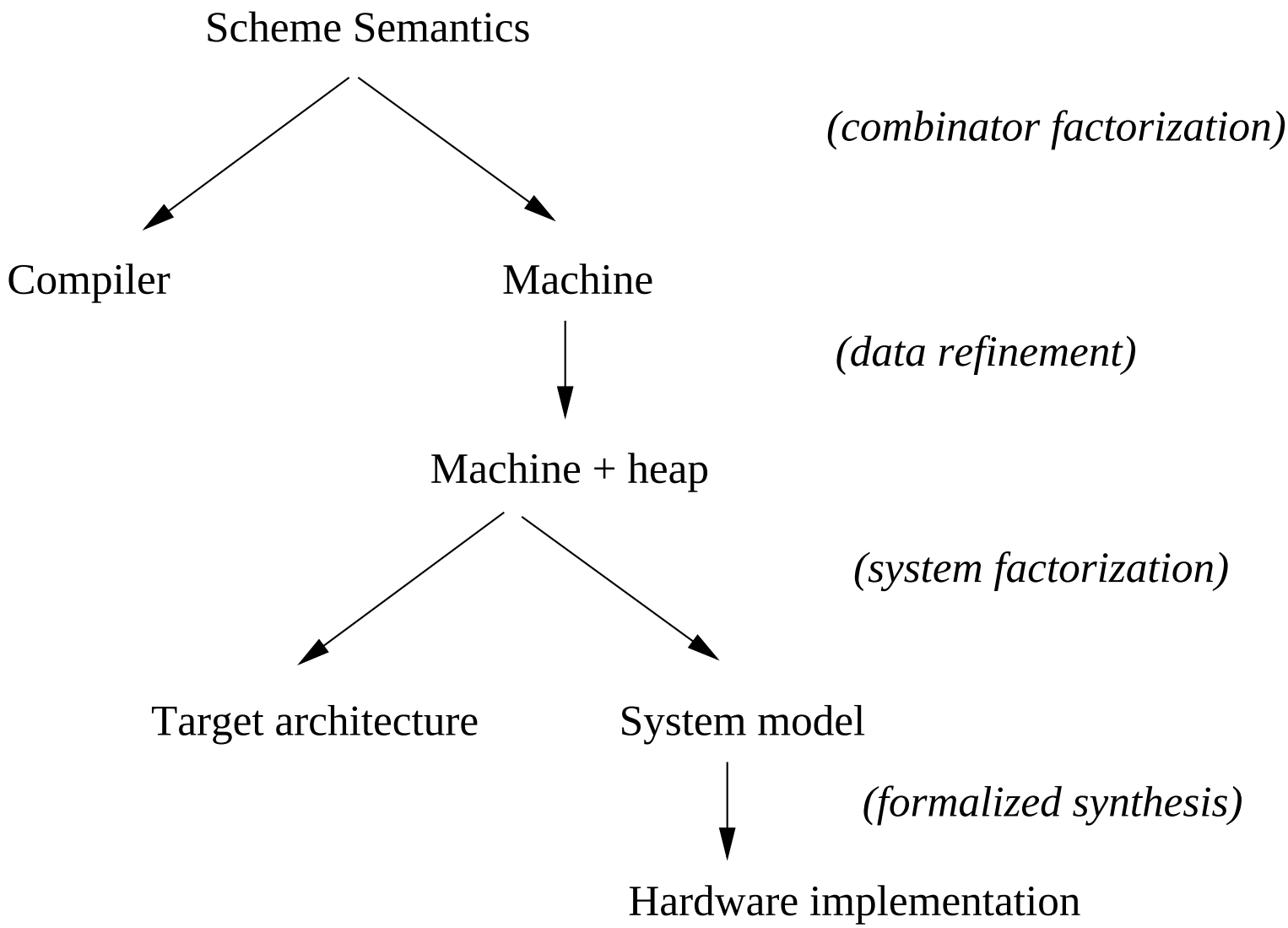
- CPU, GC, ALLOC, INIT derived with DDD
- CPU is naive
- Memory system tuned to GC
- PLDs, mux-based FPGAs, DRAM simms.
- Codesign using Scheme and Logic Engine
  - Spec. models
  - Derived models
  - Staging to hardware





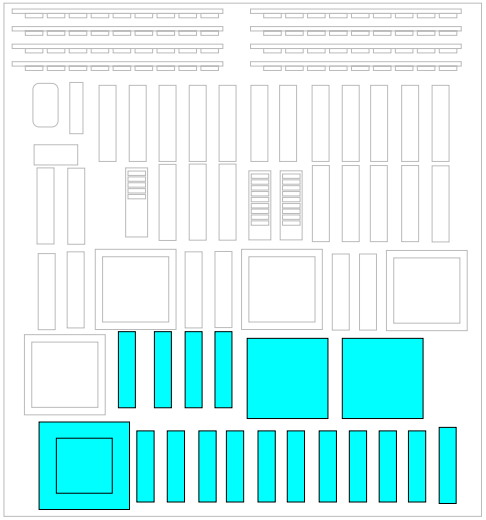


Relationshop to VLISP [Guttman, Ramsdell, Wand, L&SC 95]

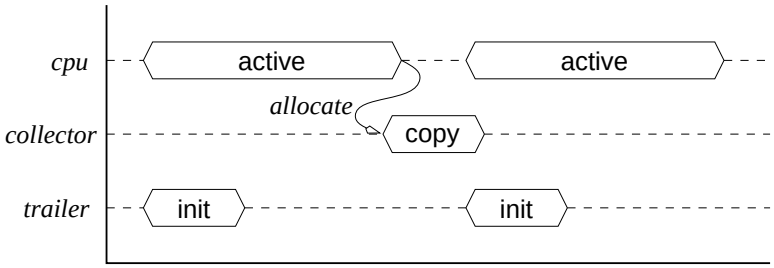


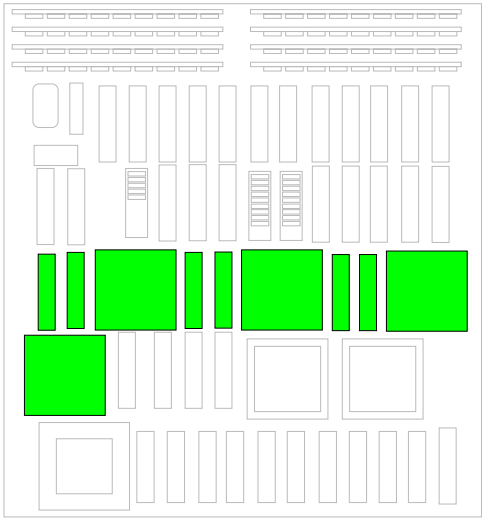
Pending issues

CPU: The architecture is naive; it needs retiming. The specification could be at a much higher level, closer to VLISP/*EoPL*.



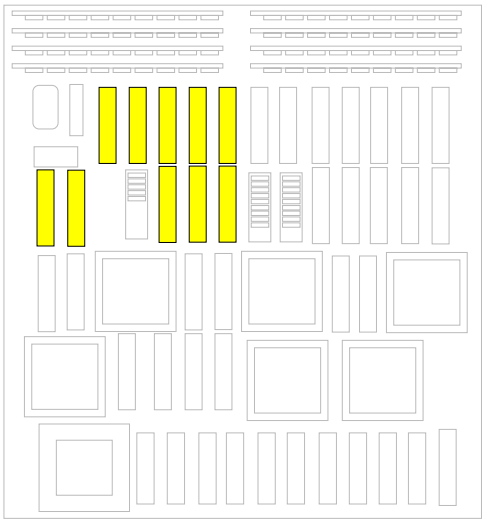
GC: Non-interference verification at the system, algorithm verificaiton. Investigation of caching and incremental variants.

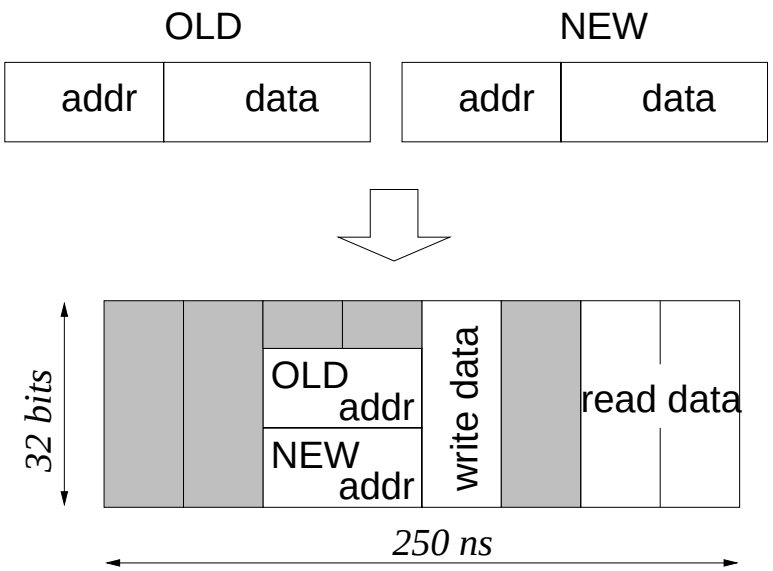




Bus timing: Redesign needed for current memory architectures. Could model checking have found the killer bug?

	read(NEW, a)	‖	write(OLD, b, c)
Large	write(NEW, a, b)	‖	read(OLD, c)
	read(NEW, a)	‖	mark-next(OLD)
	write(NEW, a, b)	‖	mark-next(OLD)





Synchronization: Hierarchical clocking was used.

Other pending issues

- Retargetting Schemachine (Virtex, SDRAM?), cores
- Behavior table case study
- Interface abstraction
- Software factorization
- Integrating **modeling**, derivation, and co-design
- Close with VLISP
- Embedded resource management
- . . . Java (?!)