SHIM: A Deterministic Model for Heterogeneous Embedded Systems

Stephen A. Edwards and Olivier Tardieu

Department of Computer Science,
Columbia University

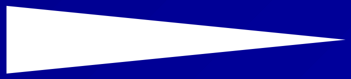
www.cs.columbia.edu/~sedwards

{sedwards,tardieu}@cs.columbia.edu

Definition

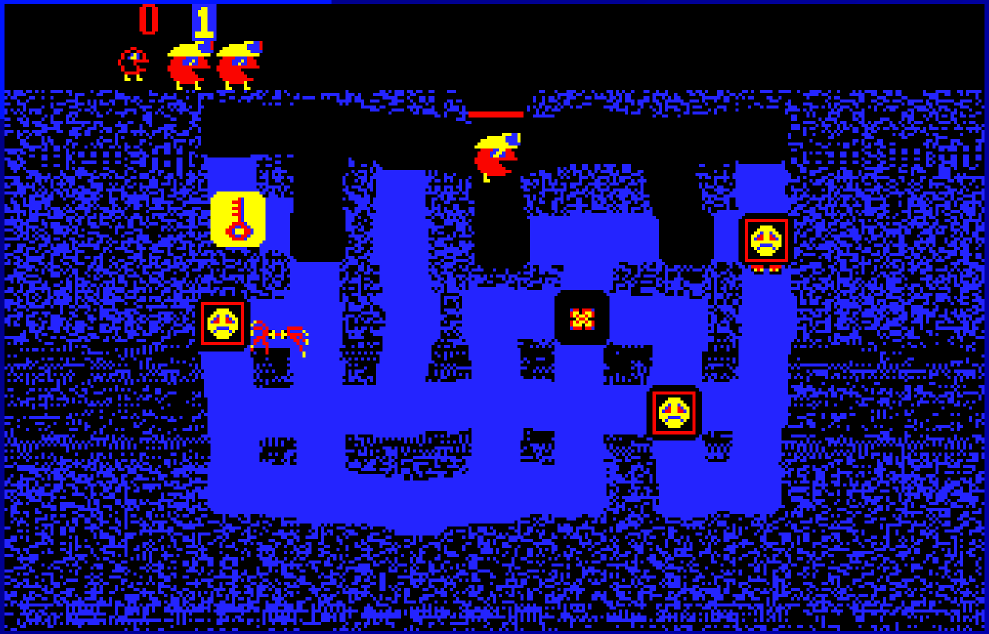
shim \’shim\ *n*

1 : a thin often tapered piece of material (as wood, metal, or stone) used to fill in space between things (as for support, leveling, or adjustment of fit).

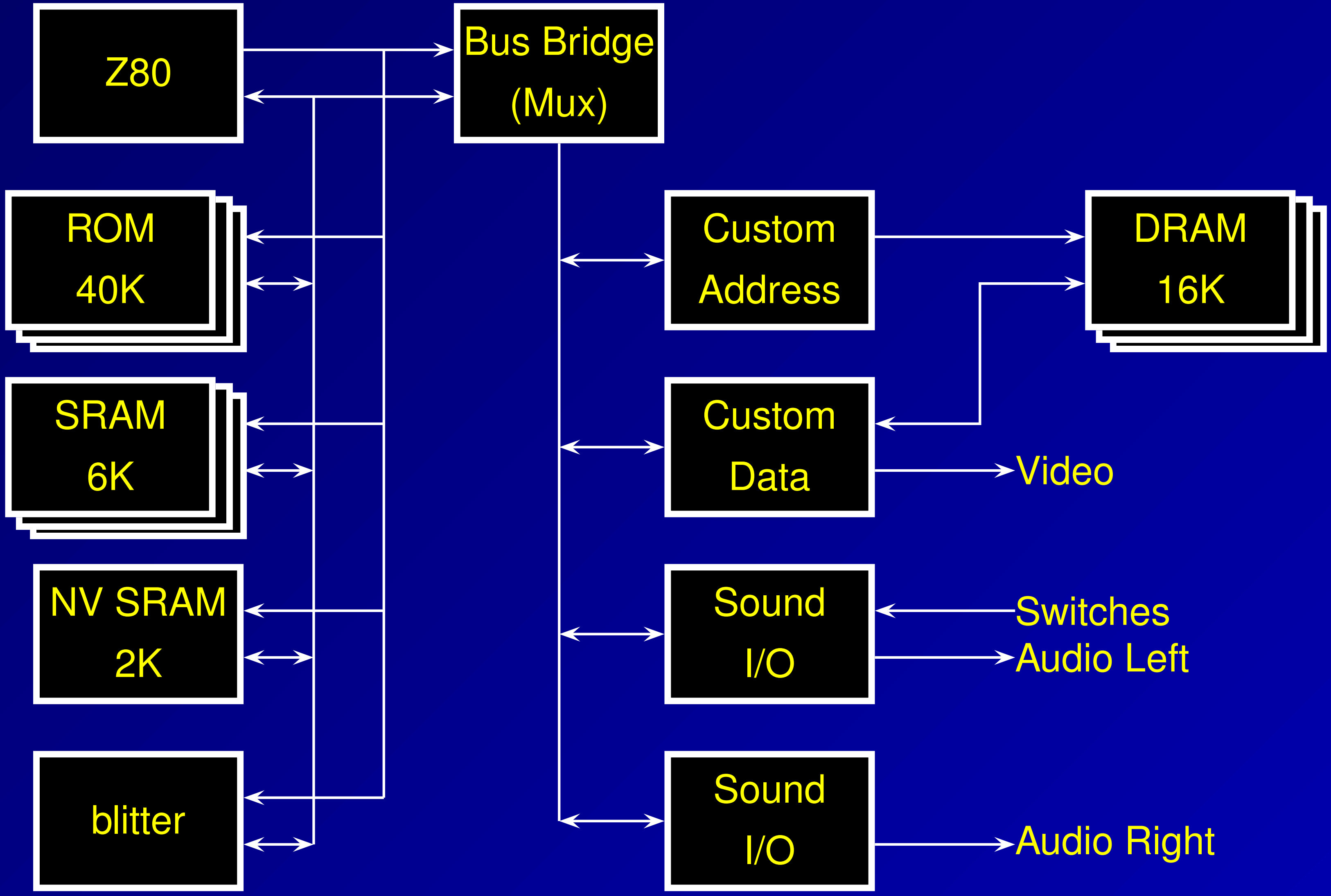


2 : *Software/Hardware Integration Medium*, a model for describing hardware/software systems

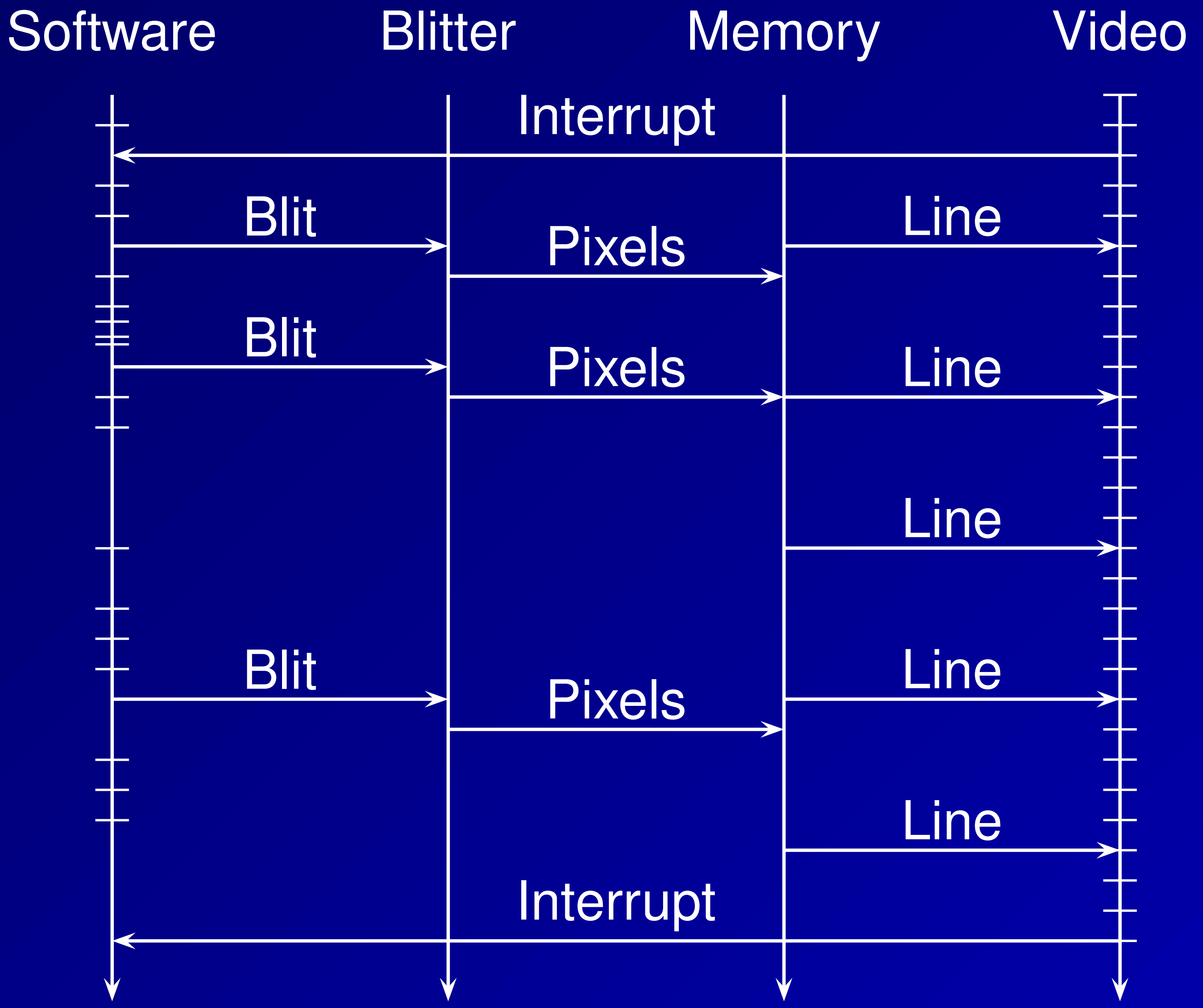
Robby Roto (Bally/Midway, 1981)



Robby Roto Block Diagram



HW/SW Interaction



SHIM Wishlist

- *Mixes synchronous and asynchronous styles*

  Need multi-rate for hardware/software systems

- *Delay-insensitive (Deterministic)*

  Want simulated behavior to reflect reality

  Verify functionality and performance separately

- *Only requires bounded resources*

  Hardware resources fundamentally bounded

- *Formal semantics*

  Do not want arguments about what something means

Deterministic, Concurrent MoCs

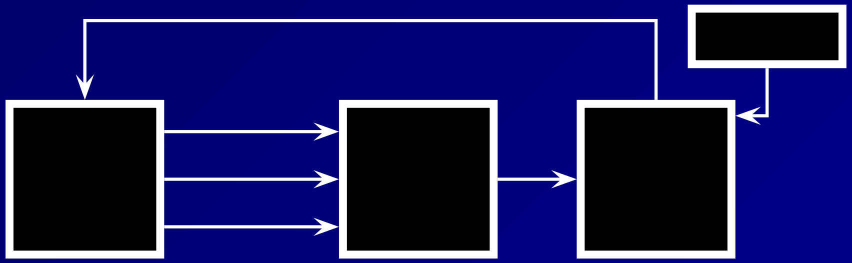
Not too many:

The Synchronous Model	Bad for multi-rate and asynchronous behavior
The Lambda Calculus	Unbounded in general, not obvious in hardware
Kahn Networks	Unbounded in general, difficult to schedule

Idea: Restrict Kahn to be bounded.

The SHIM Model

Kahn networks with rendezvous communication



Sequential processes

Unbuffered point-to-point communication channels exchange data tokens

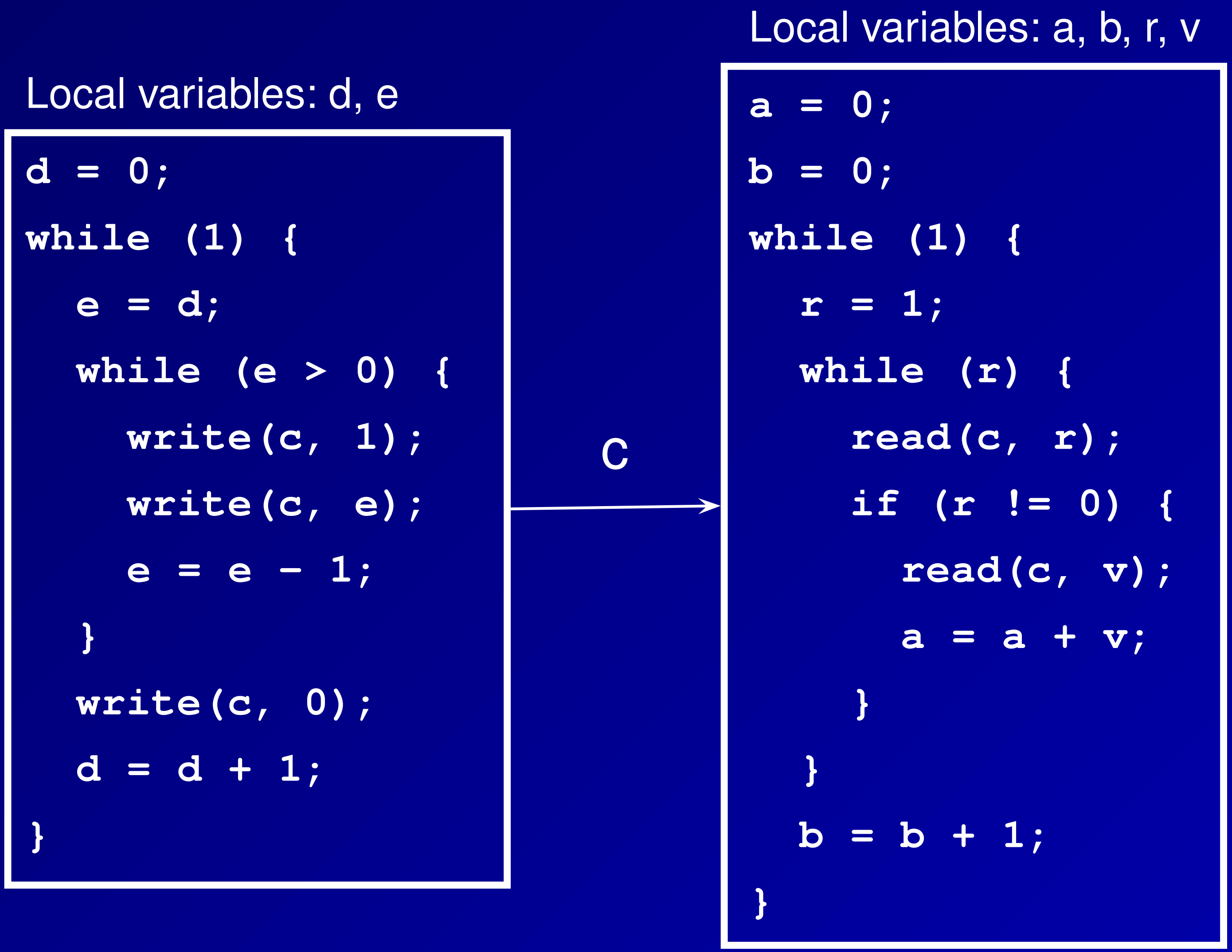
Fixed communication topology

Fundamentally asynchronous

Each communication event is synchronous (like a clock)

Delay-insensitive: sequence of data through any channel is independent of scheduling policy (the Kahn principle)

Tiny-SHIM Processes



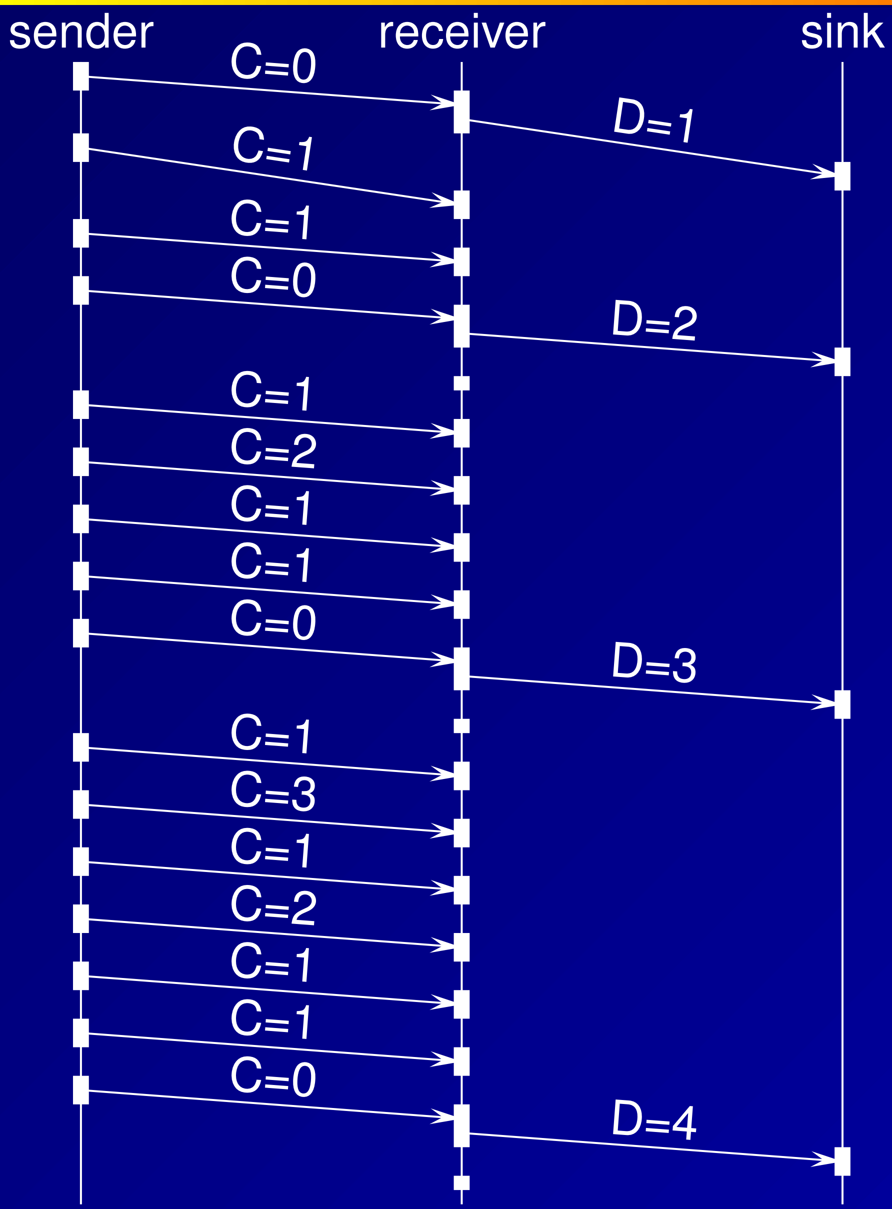
Local variables: d, e

```
d = 0;
while (1) {
  e = d;
  while (e > 0) {
    write(c, 1);
    write(c, e);
    e = e – 1;
  }
  write(c, 0);
  d = d + 1;
}
```

Local variables: a, b, r, v

```
a = 0;
b = 0;
while (1) {
  r = 1;
  while (r) {
    read(c, r);
    if (r != 0) {
      read(c, v);
      a = a + v;
    }
  }
  b = b + 1;
}
```

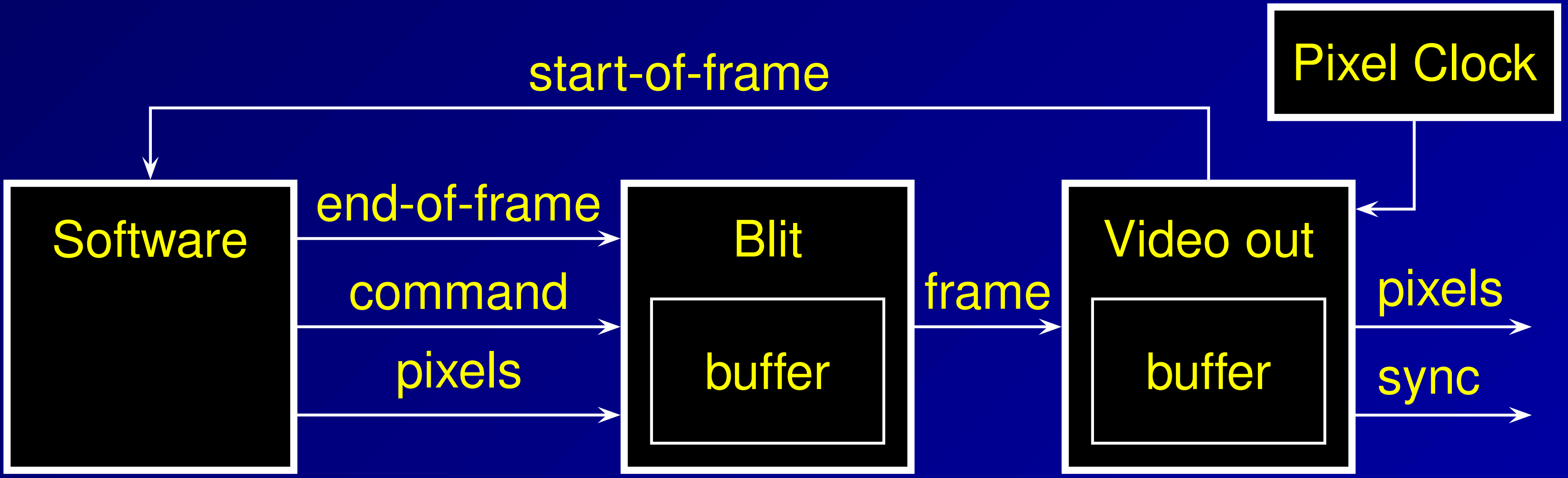
Behavior of the Processes



```
d = 0;
while (1) {
   e = d;
   while (e > 0) {
      write(c, 1);
      write(c, e);
      e = e - 1;
   }
   write(c, 0);
   d = d + 1;
}
```

```
a = 0;
b = 0;
while (1) {
   r = 1;
   while (r) {
      read(c, r);
      if (r != 0) {
         read(c, v);
         a = a + v;
      }
   }
   b = b + 1;
}
```

Robby Roto in SHIM: Block Diagram



```
while the player is alive do
    Wait for start-of-frame
    ...game logic...
    Write "false" to end-of-frame
    Write to the blitter
    ...game logic...
    Write "true" to end-of-frame
```

```
while 1 do
    while not end-of-frame do
        Read blit command
        Write pixels to memory
    Write frame
```

```
while 1 do
    Write start-of-frame
    for each line do
        Emit line timing signals
        for each pixel do
            Wait for pixel clock
            Read pixel from memory
            Send pixel to display
    Read next frame
```

The Syntax of Tiny-SHIM

$e ::=$	L	(literal)
$\mid$	V	(variable)
$\mid$	*op* e	(unary op)
$\mid$	e *op* e	(binary op)
$\mid$	`(` e `)`	(paren)

$s ::=$	V `=` e	(assignment)
$\mid$	`if (` e `)` s `else` s	(conditional)
$\mid$	`while (` e `)` s	(loop)
$\mid$	s `;` s	(sequencing)
$\mid$	`read(` C, V `)`	(blocking read)
$\mid$	`write(` C, e `)`	(blocking write)
$\mid$	$\{$ s $\}$	(grouping)

The SOS Semantics of Tiny-SHIM

σ	Process memory state	p	Process code
$\langle \sigma, p \rangle$	Process p in state σ	$\langle \sigma \rangle$	Terminated in state σ
$\xrightarrow{a}$	Single-process rule	$\Rightarrow$	System rule
$\mathcal{E}(\sigma, e)$	Value of e in σ		

$$\frac{\mathcal{E}(\sigma, e) = n}{\langle \sigma, v \texttt{ = } e \rangle \rightarrow \langle \sigma[v \leftarrow n] \rangle} \qquad \text{(assign)}$$

$$\frac{\mathcal{E}(\sigma, e) \neq 0}{\langle \sigma, \texttt{if (}e\texttt{)}\ p\ \texttt{else}\ q \rangle \rightarrow \langle \sigma, p \rangle} \qquad \text{(if-true)}$$

$$\frac{\mathcal{E}(\sigma, e) = 0}{\langle \sigma, \texttt{if (}e\texttt{)}\ p\ \texttt{else}\ q \rangle \rightarrow \langle \sigma, q \rangle} \qquad \text{(if-false)}$$

Semantics of Looping & Sequencing

$$\frac{\mathcal{E}(\sigma, e) \neq 0}{\langle \sigma, \texttt{while (}e\texttt{)}\ p \rangle \rightarrow \langle \sigma, p\ \texttt{;}\ \texttt{while (}e\texttt{)}\ p \rangle}$$ (while-true)

$$\frac{\mathcal{E}(\sigma, e) = 0}{\langle \sigma, \texttt{while (}e\texttt{)}\ p \rangle \rightarrow \langle \sigma \rangle}$$ (while-false)

$$\frac{\langle \sigma, p \rangle \xrightarrow{a} \langle \sigma', p' \rangle}{\langle \sigma, p\ \texttt{;}\ q \rangle \xrightarrow{a} \langle \sigma', p'\ \texttt{;}\ q \rangle}$$ (seq)

$$\frac{\langle \sigma, p \rangle \xrightarrow{a} \langle \sigma' \rangle}{\langle \sigma, p\ \texttt{;}\ q \rangle \xrightarrow{a} \langle \sigma', q \rangle}$$ (seq-term)

Communication and Concurrency

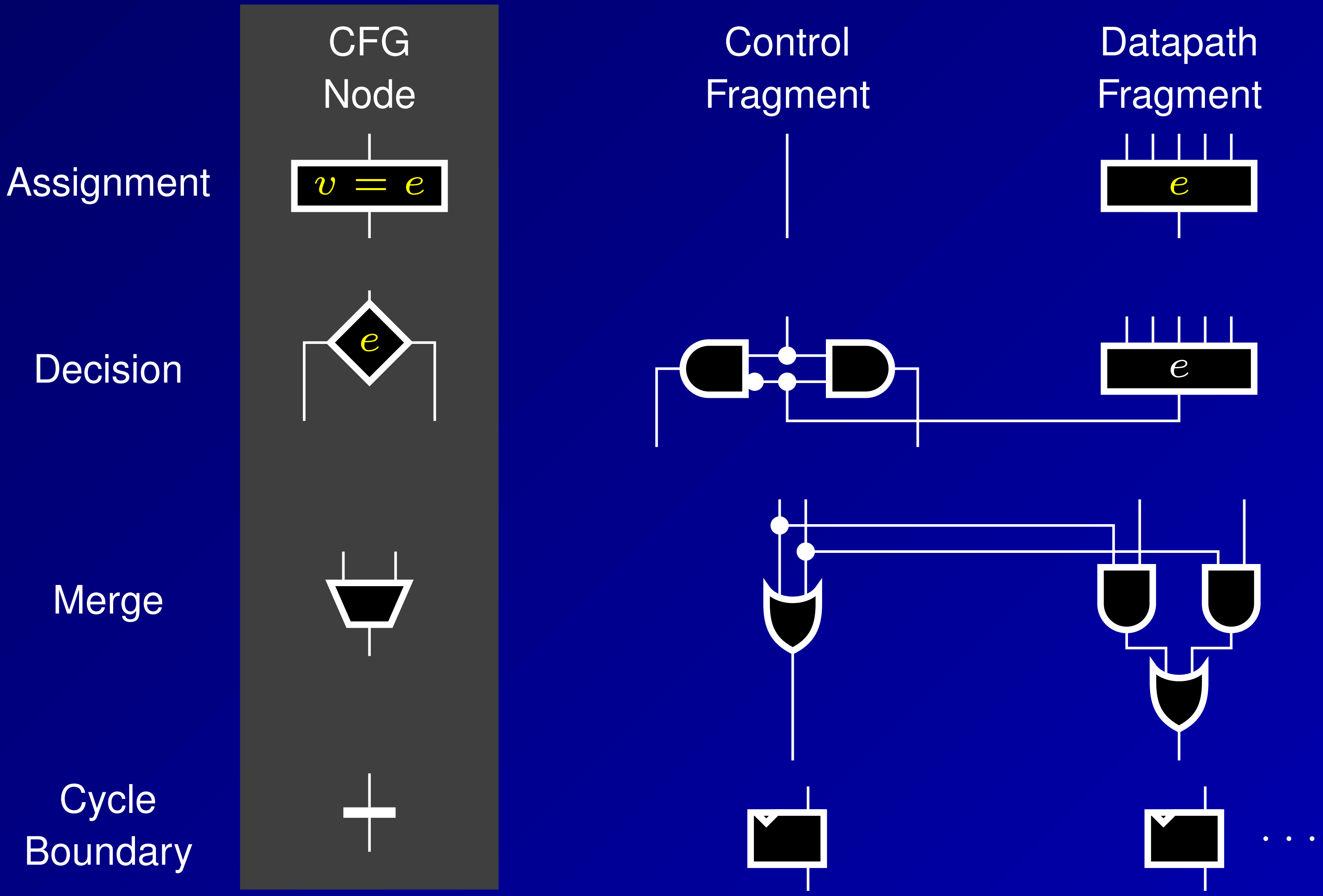
$$\langle \sigma, \texttt{read}(c, v) \rangle \xrightarrow{c \text{ get } n} \langle \sigma[v \leftarrow n] \rangle \qquad \text{(read)}$$

$$\frac{\mathcal{E}(\sigma, e) = n}{\langle \sigma, \texttt{write}(c, e) \rangle \xrightarrow{c \text{ put } n} \langle \sigma \rangle} \qquad \text{(write)}$$

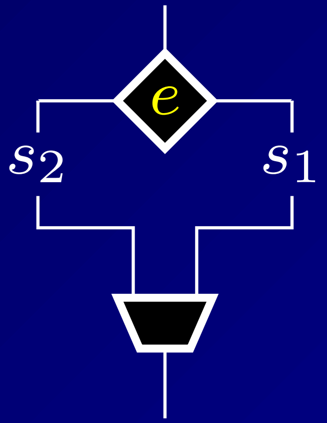
$$\frac{\langle \sigma, p \rangle \longrightarrow s}{\{\langle \sigma, p \rangle\} \uplus S \Rightarrow \{s\} \uplus S} \qquad \text{(step)}$$

$$\frac{\langle \sigma, p \rangle \xrightarrow{c \text{ put } n} s \qquad \langle \sigma', p' \rangle \xrightarrow{c \text{ get } n} s'}{\{\langle \sigma, p \rangle, \langle \sigma', p' \rangle\} \uplus S \Rightarrow \{s, s'\} \uplus S} \qquad \text{(sync)}$$

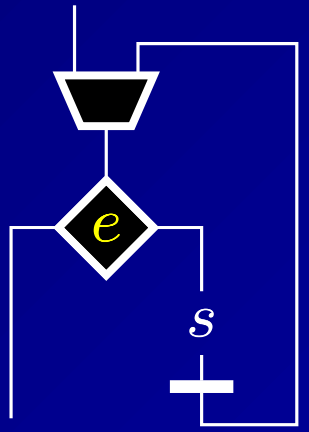
Translating Tiny-SHIM to Hardware



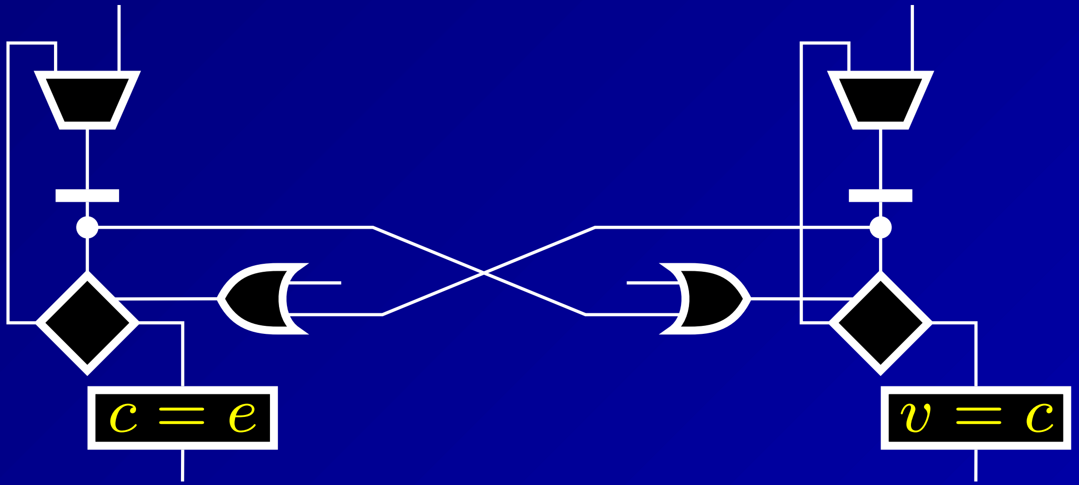
Translation Patterns



if (e) s_1 **else** s_2



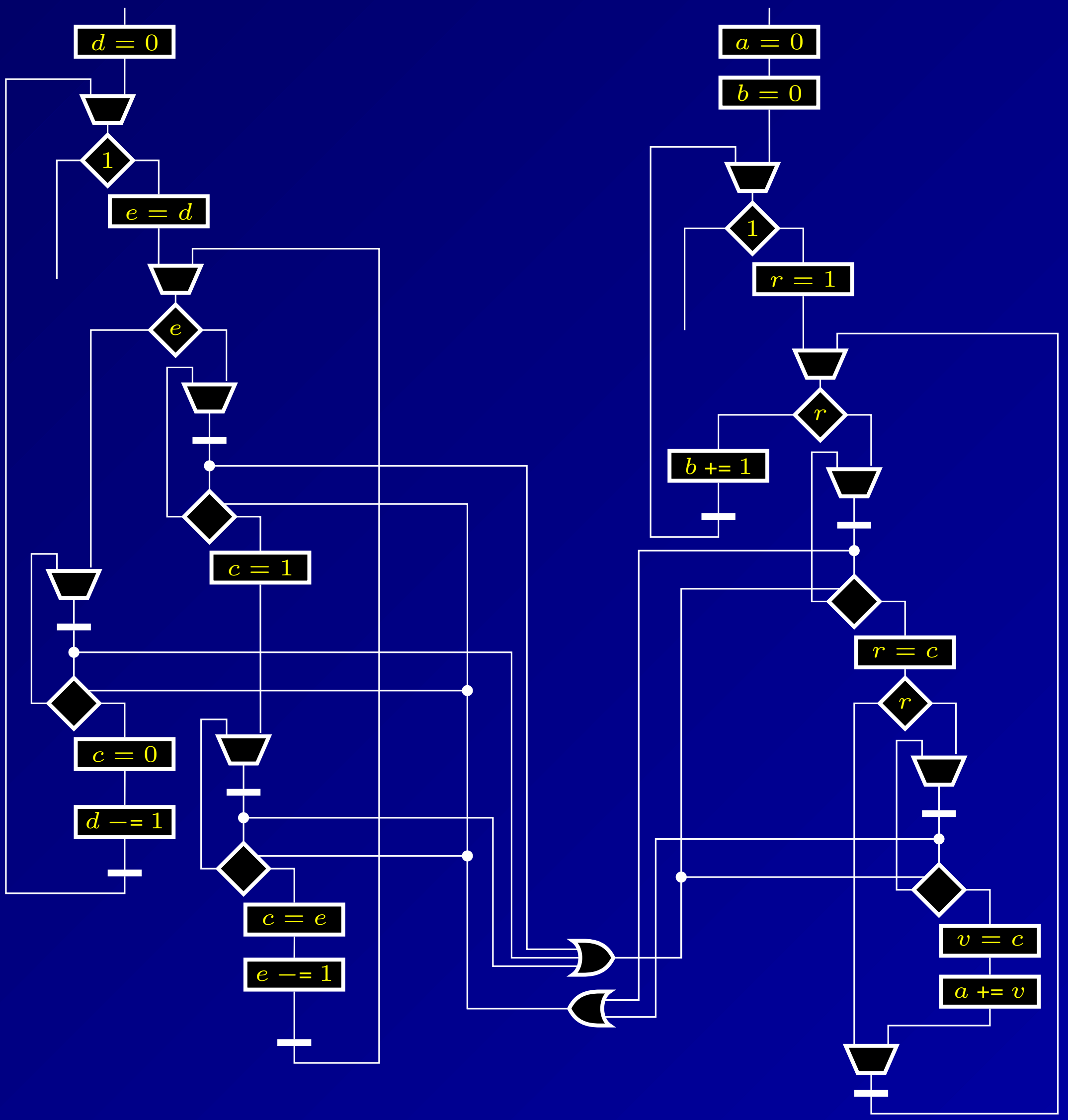
while (e) s



write(c, e)

read(c, v)

Translation



```
d = 0;
while (1) {
  e = d;
  while (e > 0) {
    write(c, 1);
    write(c, e);
    e = e - 1;
  }
  write(c, 0);
  d = d + 1;
}
```

```
a = 0;
b = 0;
while (1) {
  r = 1;
  while (r) {
    read(c, r);
    if (r != 0) {
      read(c, v);
      a = a + v;
    }
  }
  b = b + 1;
}
```

Summary

- SHIM: A delay-insensitive (deterministic) model of computation that supports synchrony and asynchrony
- Tiny-SHIM: A little language that embodies the model
- Formal operational semantics of Tiny-SHIM
- A procedure for translating Tiny-SHIM into hardware

Ongoing Work

- Translation into software
- Relaxation of block-on-single-channel rule
- Complete hardware/software design language
- Translation optimization for hardware and software