SHIM: A Deterministic Model for Heterogeneous Embedded Systems

Stephen A. Edwards and Olivier Tardieu

Department of Computer Science,
Columbia University

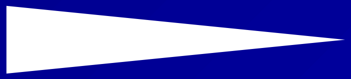
www.cs.columbia.edu/~sedwards

{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

Conclusions

SHIM is an effective model of computation for embedded hardware/software systems

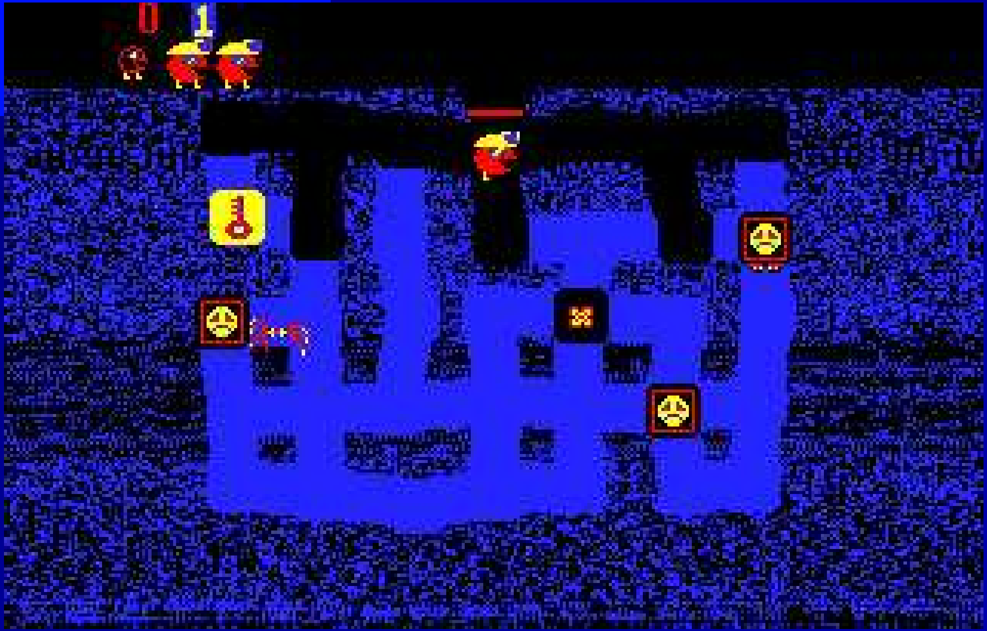
Formal semantics guarantee determinism & boundedness

Easy to synthesize into hardware and software

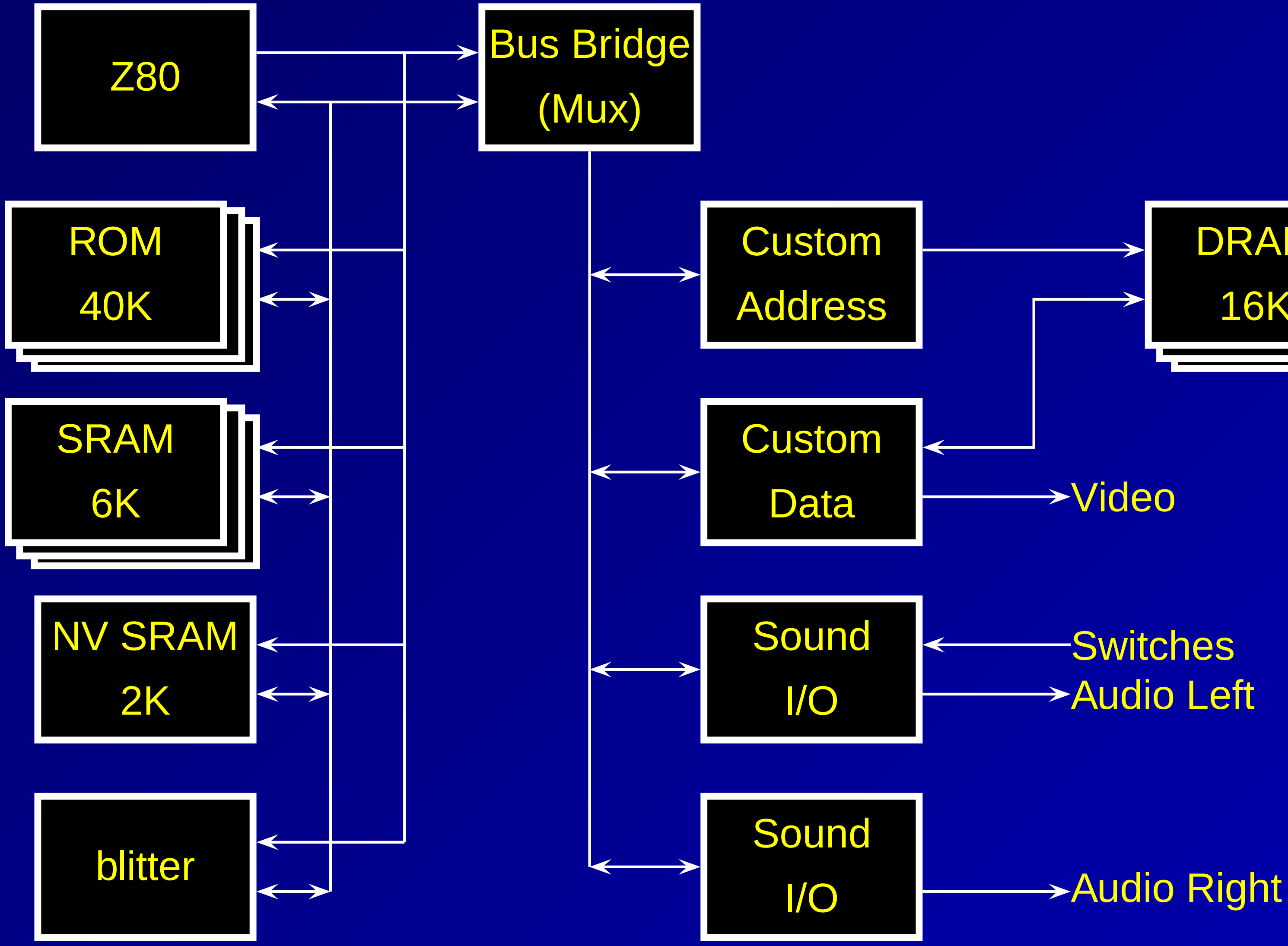
Applicable to large, important class of systems, but not all

Embedded systems should be designed on the SHIM model of computation

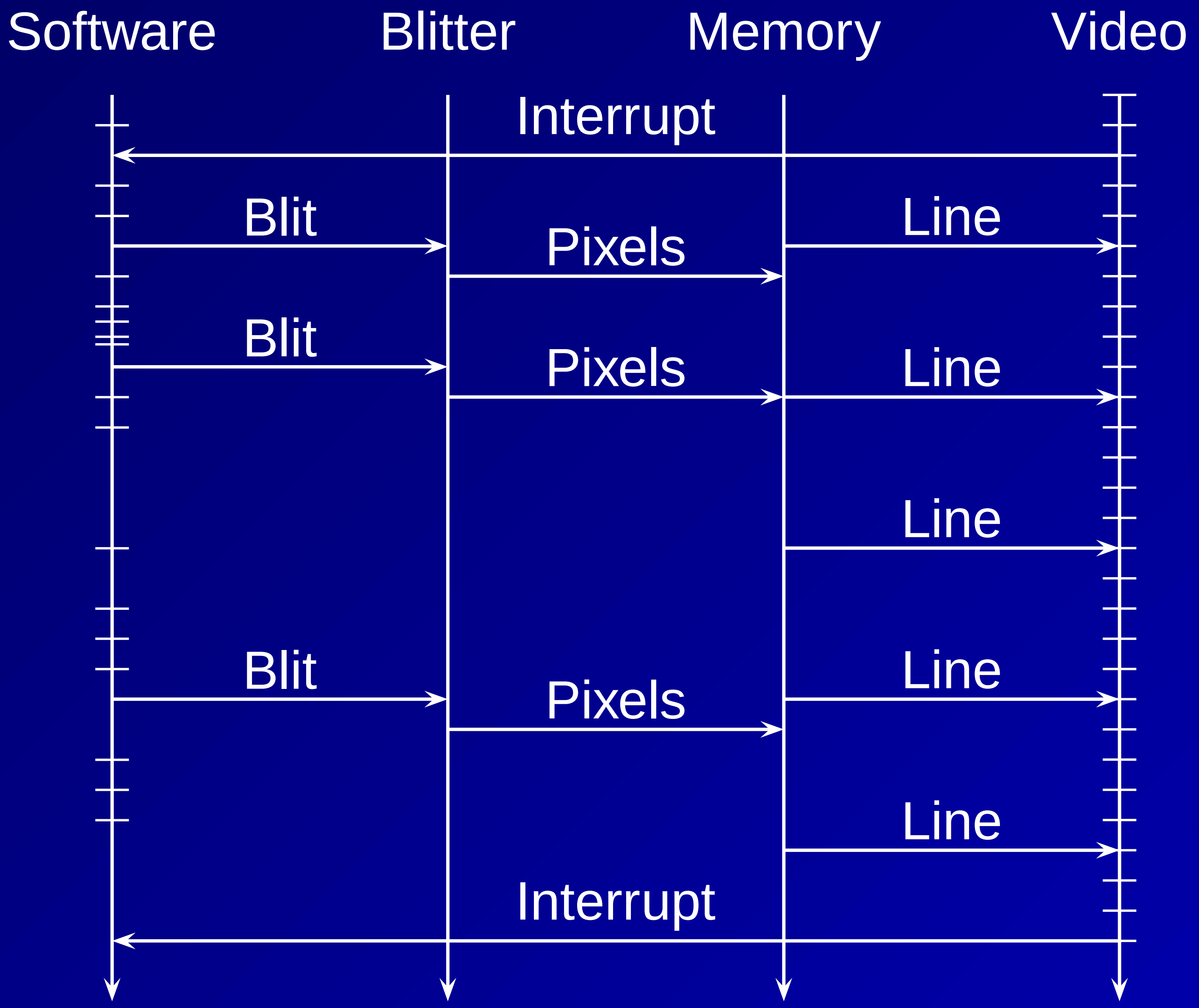
Robby Roto (Bally/Midway, 1981)



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

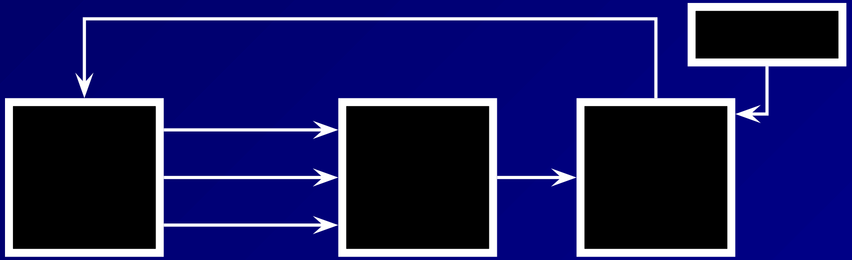
- *Only requires bounded resources*

  Hardware resources fundamentally bounded

- *Formal semantics*

  Do not want arguments about what something means

The SHIM Model



Sequential processes

Unbuffered point-to-point
communication channels
exchange data tokens

Fixed topology

Asynchronous

Synchronous communication events

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

“Kahn networks with rendezvous communication”

SHIM vs. Other Models

	SHIM	CSP	Kahn	SDF	Haste	Sync	Petri
Deterministic	√		√	√		√	
Blocking Communication	√	√			√	√	√
Bounded Buffers	√	√		√	√	√	
Multi-Rate	√	√	√	√	√		√
Data-Dependent Rates	√	√	√		√		√
Easy-To-Schedule	√	√		√	√	√	√
Static Scheduling				√		√	

Modeling in SHIM

To model	**introduce**
Buffers	Buffer processes
Interrupts	Polling and periodic communication
Synchrony	Clock signals
Synchronous dataflow	Buffers
Sensors	Source processes
Arbiters	A deterministic algorithm

Modeling Time in SHIM

SHIM is timing-independent

Philosophy: separate functional requirements from performance requirements

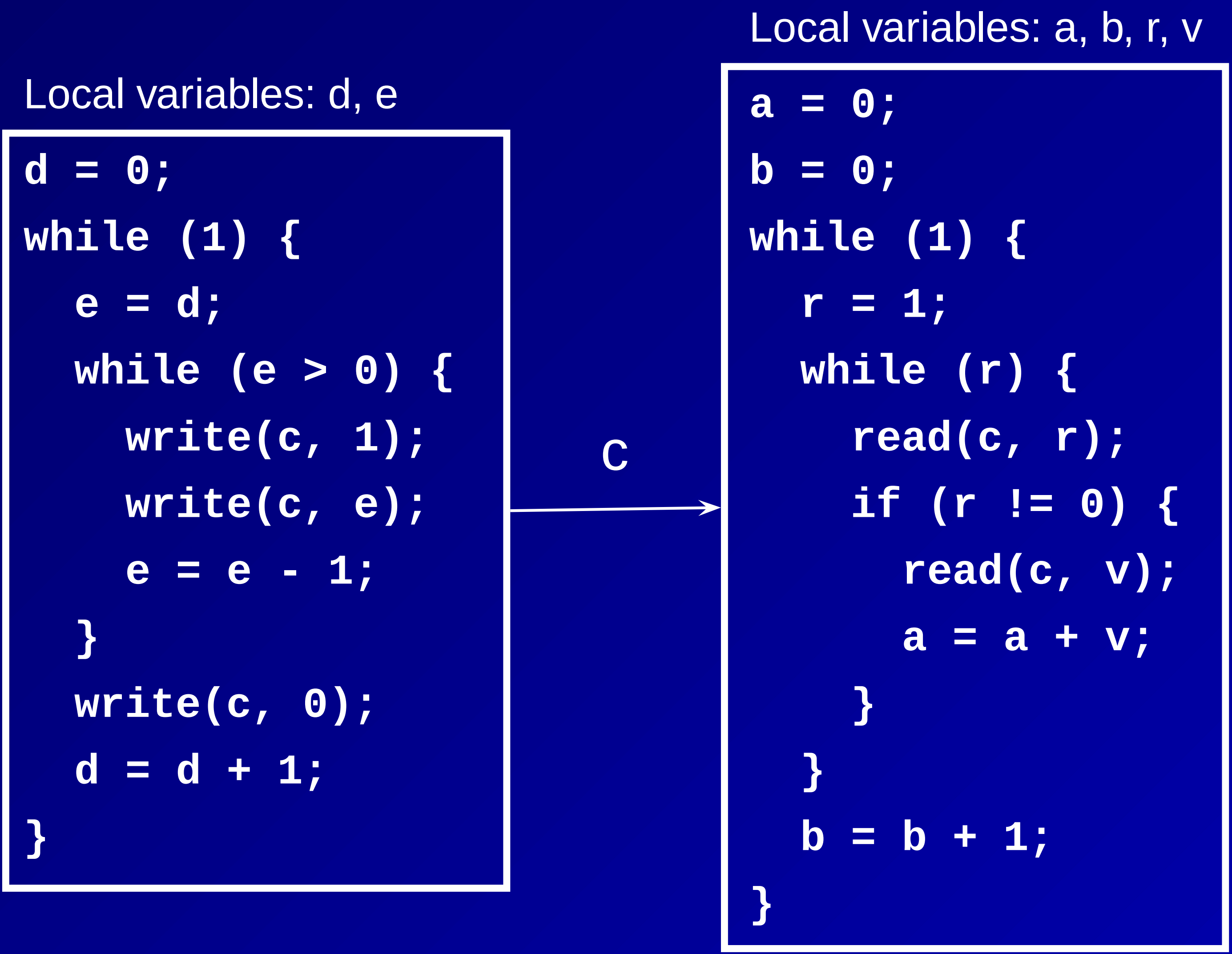
Like synchronous digital logic: establish correct function independent of timing, then check and correct performance errors

Vision: clock processes impose execution rates, checked through static timing analysis

The Syntax of Tiny-SHIM

Expressions		**Statements**	
$e ::= L$	(literal)	$s ::= V$ **=** e	(assignment)
$\mid$ V	(variable)	$\mid$ **if (** e **)** s **else** s	(conditional)
$\mid$ *op* e	(unary op)	$\mid$ **while (** e **)** s	(loop)
$\mid$ e *op* e	(binary op)	$\mid$ s **;** s	(sequencing)
$\mid$ **(** e **)**	(paren)	$\mid$ $\{$ s $\}$	(grouping)
		$\mid$ **read(** C, V **)**	(blocking read)
		$\mid$ **write(** C, e **)**	(blocking write)

Example Processes



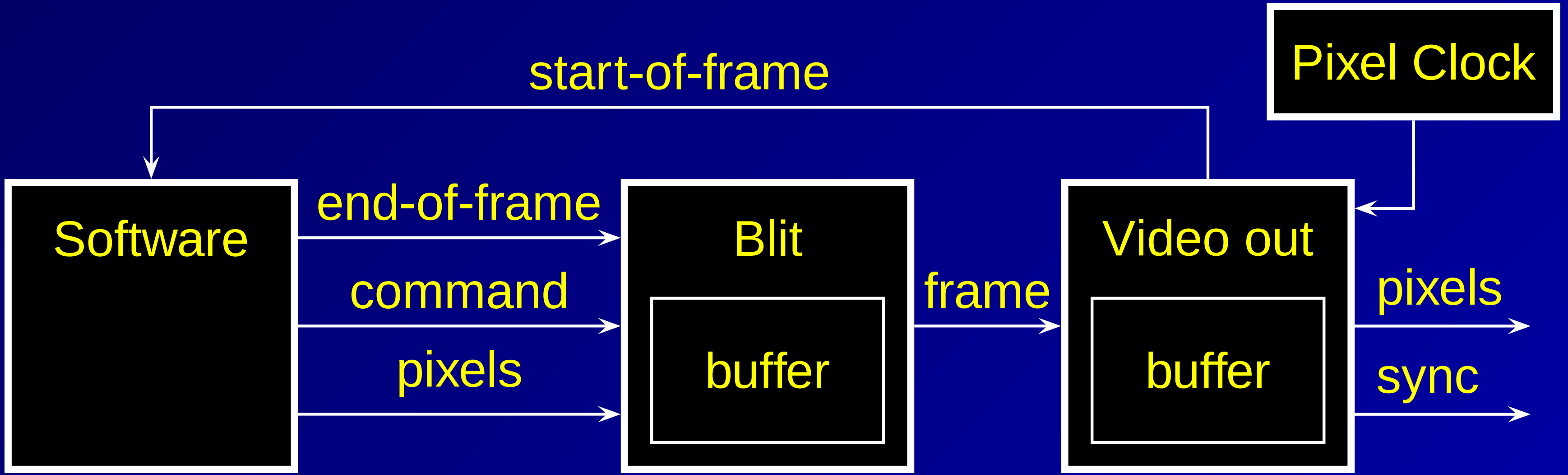
Local variables: d, e

```
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;
}
```

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 memo
            Send pixel to display
    Read next frame
```

The SOS Semantics of Tiny-SHIM

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

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

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

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

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

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

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

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

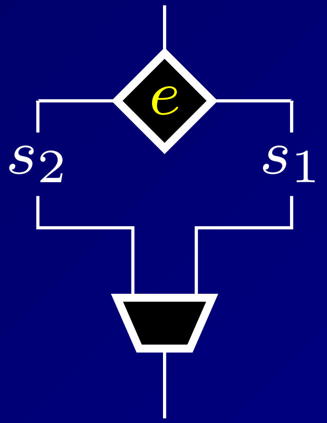
$$\langle\sigma, \texttt{read(}c, v\texttt{)}\rangle \xrightarrow{c\ \mathrm{get}\ n} \langle\sigma[v \leftarrow n]\rangle \quad \text{(read)}$$

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

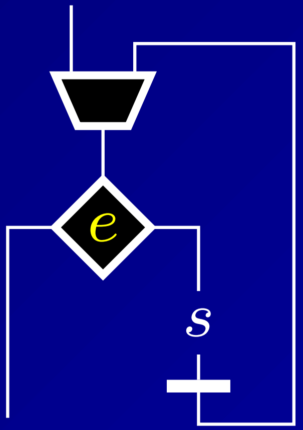
$$\frac{\langle\sigma, p\rangle \to s}{\{\langle\sigma, p\rangle\} \uplus S \Rightarrow \{s\} \uplus S} \quad \text{(step)}$$

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

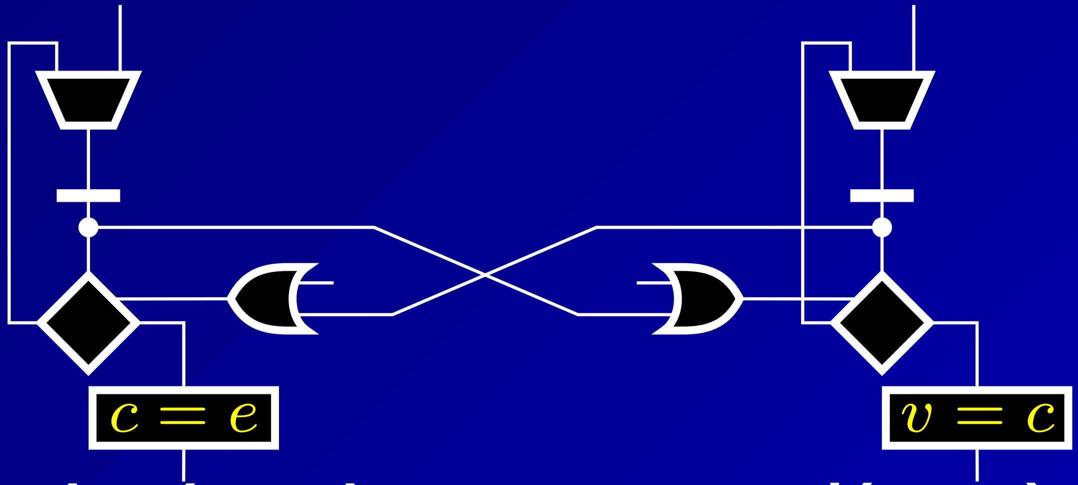
Syntax-Directed HW Translation



if (e) s_1 **else** s_2



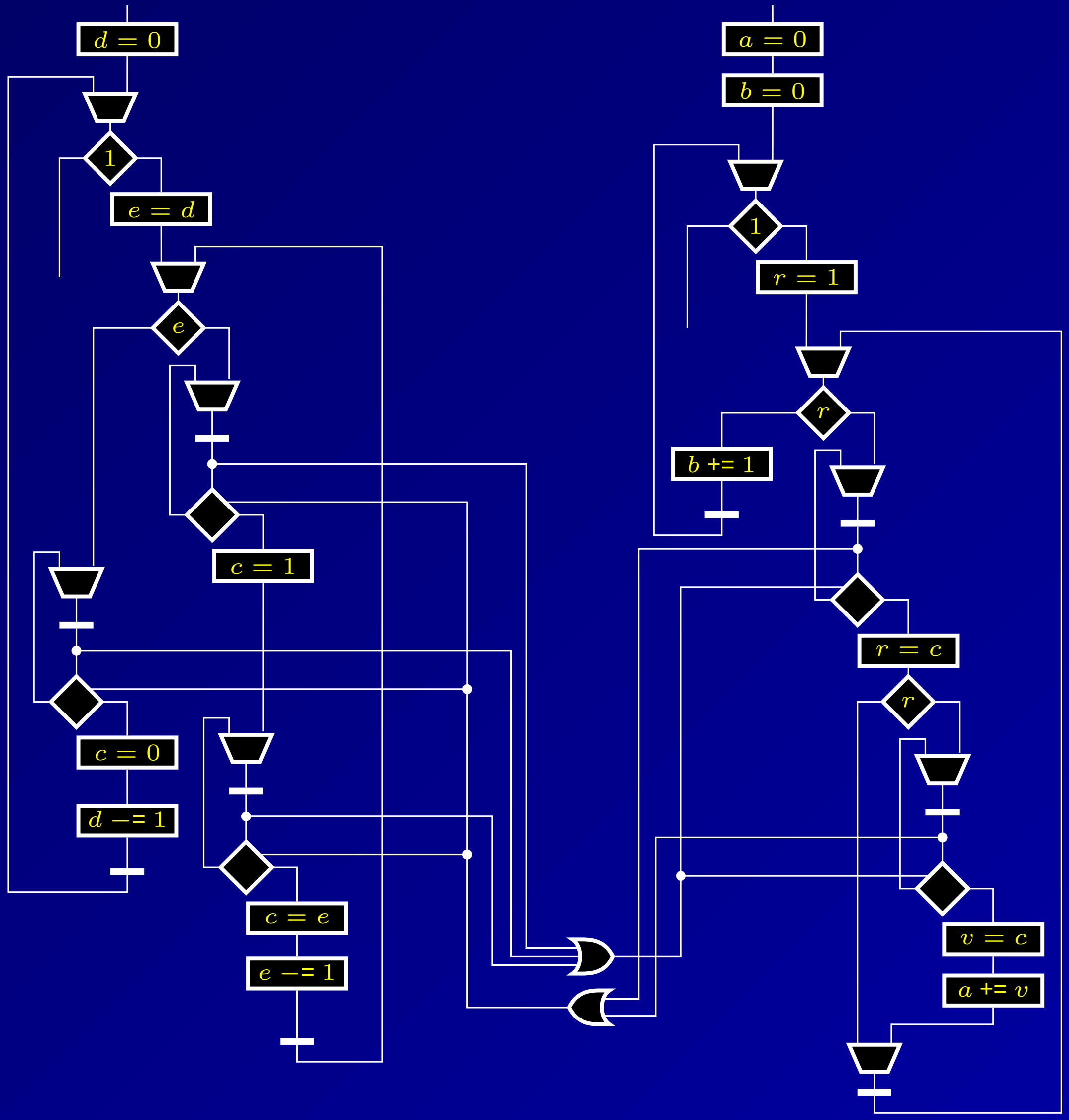
while (e) s



write(c, e)

read(c, v)

Hardware Translation Example



```
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;
}
```

Ongoing Work

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