

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



NEW!

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

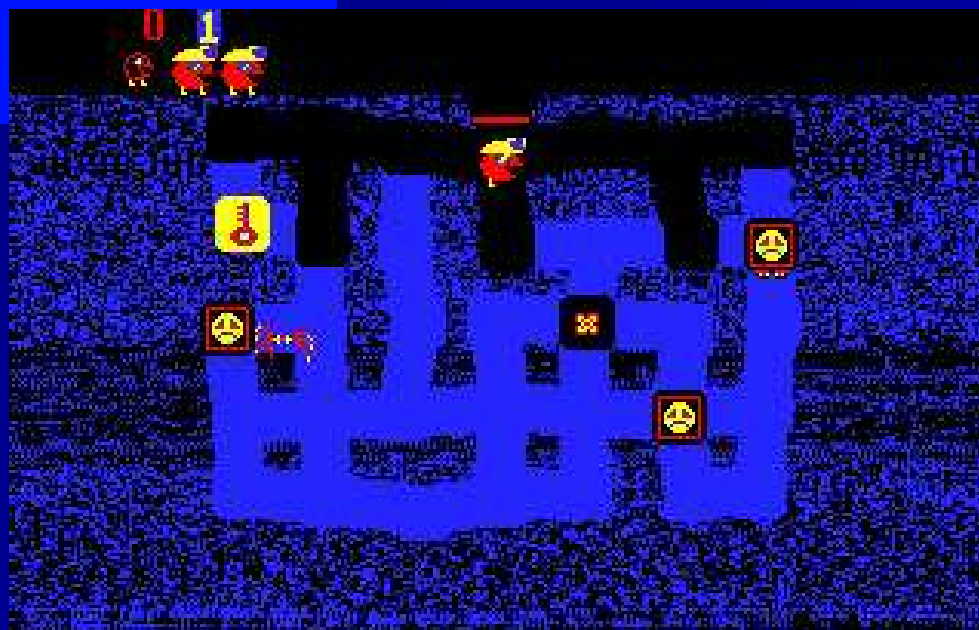
Formal semantics guarantee determinism & boundedness

Easy to synthesize into hardware and software

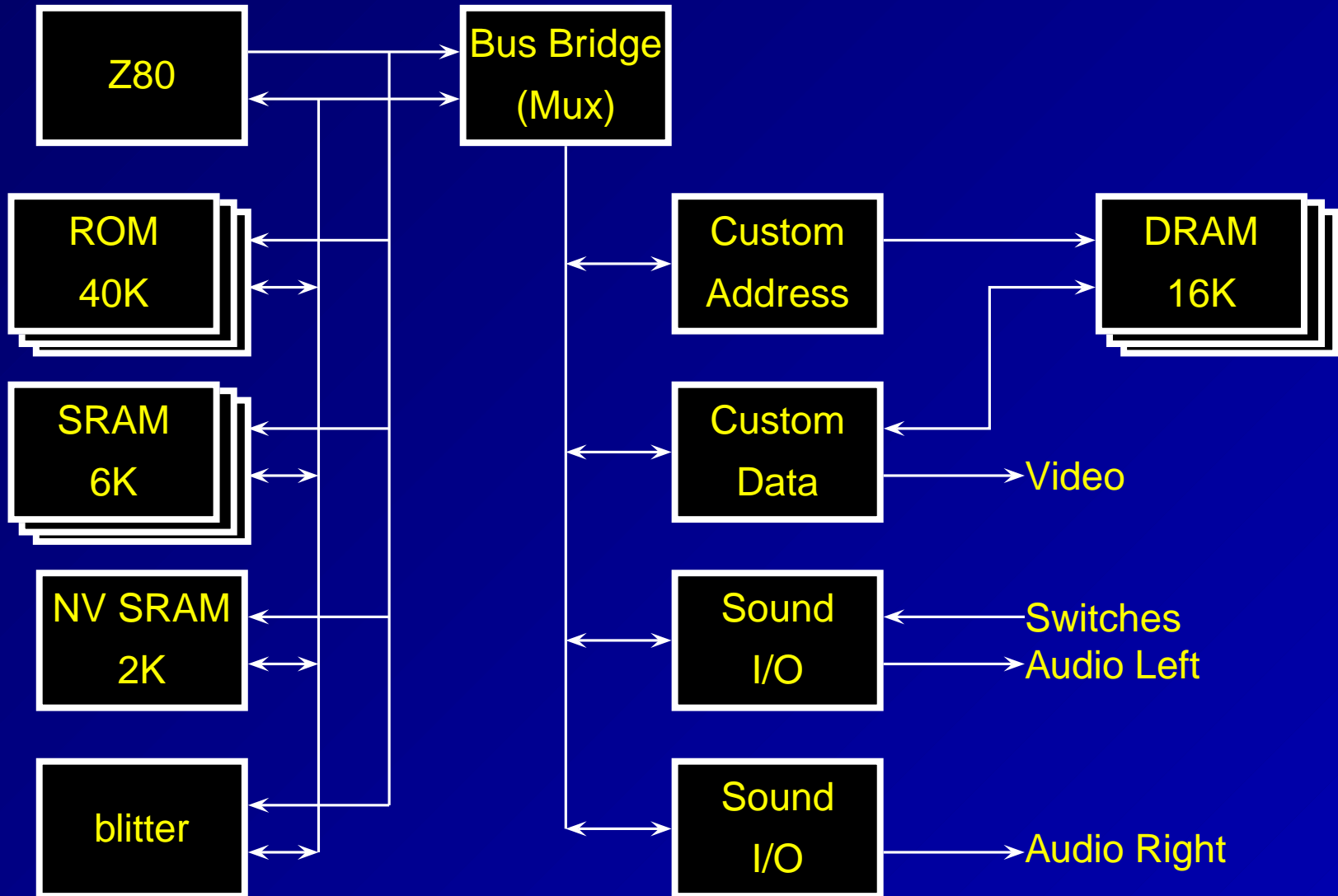
Applicable to large, important class of systems, but not all

Embedded systems should be designed on the SHIM model of computation

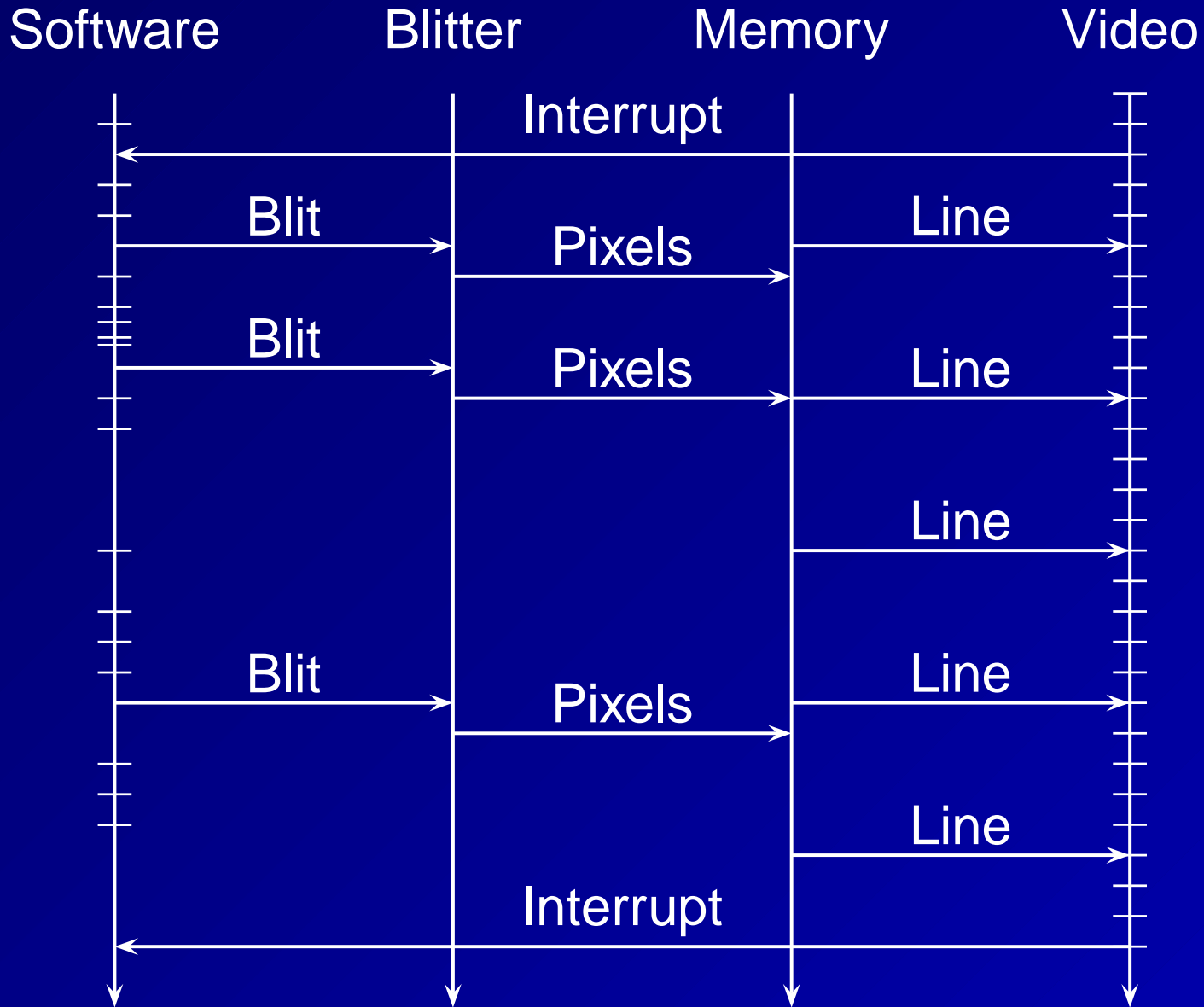
Robby Roto (Bally/Midway, 1981)



Robby Roto Block Diagram



HW/SW Interaction

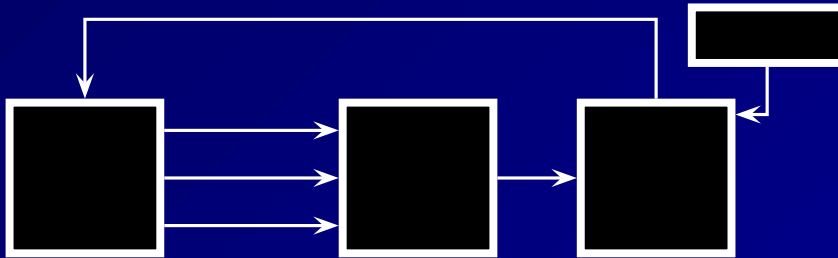


SHIM Wishlist



- *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

Buffers

Interrupts

Synchrony

Synchronous dataflow

Sensors

Arbiters

introduce

Buffer processes

Polling and periodic communication

Clock signals

Buffers

Source processes

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

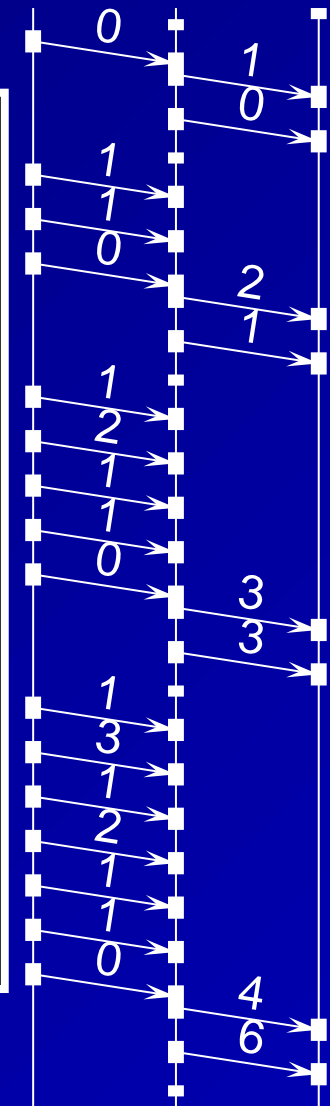
Vision: clock processes impose execution rates, checked through static timing analysis

Example Processes

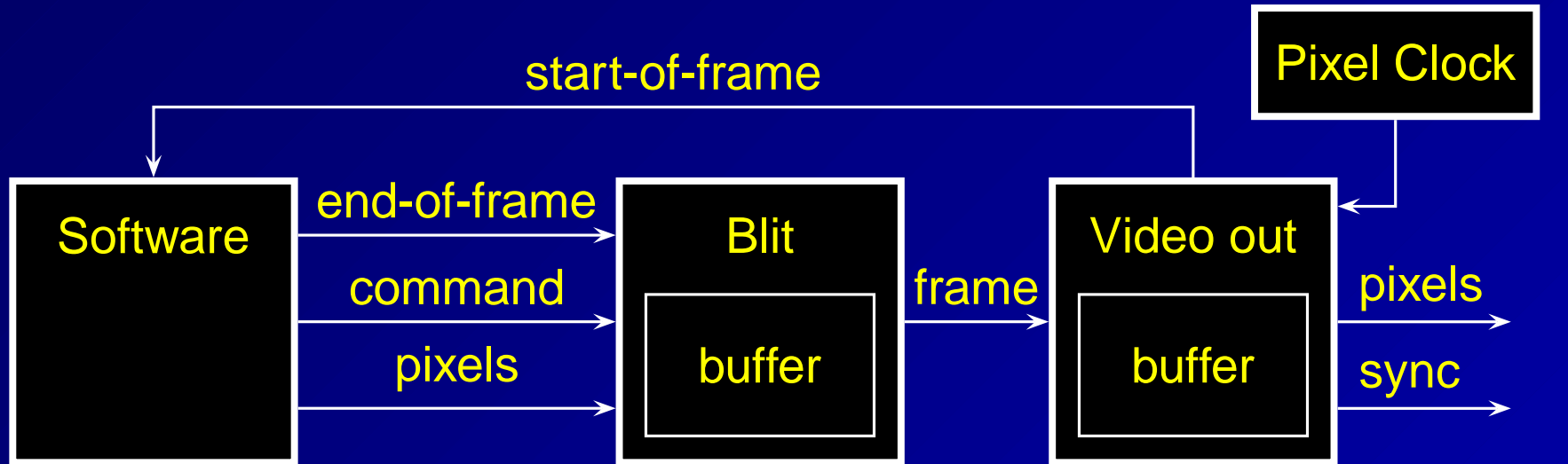
```
process sender(uint32 &C) {  
  for (int d = 0 ; d < 4 ;  
      ++d) {  
    for ( int e = d ;  
          e > 0 ; --e ) {  
      C = 1, C = e;  
    }  
    C = 0;  
  }  
}
```

```
process sink(uint32 &D) {  
  for (;;) { int v = D; }  
}
```

```
process receiver(  
  uint32 C, uint32 &D) {  
  int b = 0;  
  for (;;) {  
    int a = 0;  
    int r = 1;  
    for (;;) {  
      r = C;  
      if (r == 0) break;  
      int v = C;  
      a = a + v;  
    }  
    ++b;  
    D = b, D = a;  
  }  
}
```



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

Implementing Tiny-SHIM in Software

Each process becomes a C function with static variable that can resume itself.

Main scheduler takes runnable process from head of list and calls its function.

Processes mark themselves blocked, can schedule their communication partner.

C Translation Example

```
process p1 {  
    output C;  
    write(C, 42);  
}
```

```
process p2 {  
    input C;  
    int v;  
    read(C, v);  
}
```


C Declarations

```
typedef struct process_struct {  
    void (*process)(void);  
    struct process_struct *next;  
} process_t;
```

Linked list of runnable processes
Function of process
Next runnable process

```
typedef struct {  
    int value;  
    process_t *waiting;  
} channel_t;
```

Channel datatype
Value being transferred
Blocked process, if any

```
channel_t C = { 0, 0 };
```

Definition of channel C

```
int p1_state = 0;  
int p2_state = 0;  
void p1_function(void);  
void p2_function(void);
```

State of each process
Process functions
(forward declarations)

```
process_t p1 = { p1_function, 0 };  
process_t p2 = { p2_function, &p1 };  
process_t *head_process = &p2;
```

Linked List
of runnable
processes

The (Trivial) Scheduler

```
int main()
{
    process_t *running_process;
    while (head_process) {
        running_process = head_process;
        head_process = running_process->next;
        (*(running_process->process))();
    }
    return 0;
}
```

The scheduler

Remove head

Run it

Everything terminated or deadlocked

The Writing Process

```
void p1_function() {  
    switch (p1_state) {  
        case 1: goto L1;  
        case 0: goto L0;  
    }  
}
```

Resume at current state

```
L0:  
    C.value = 42;  
    if (C.waiting) {  
        (C.waiting)->next = head_process;  
        head_process = C.waiting;  
    }  
    C.waiting = &p1;  
    p1_state = 1;  
    return;  
L1:  
    ;  
}
```

write(C, 42)

Schedule
reading process

```
process p1 {  
    output C;  
    write(C, 42);  
}
```

Suspend

The Reading Process

```
void p2_function() {  
    static int v;  
    switch (p2_state) {  
        case 0: goto L0;  
        case 1: goto L1;  
    }  
}
```

resume at current state

```
L0:  
    if (!C.waiting) {  
        C.waiting = &p2;  
        p2_state = 1;  
        return;  
    }  
L1:  
    v = C.value;  
    (C.waiting)->next = head_process;  
    head_process = C.waiting;  
    C.waiting = 0;  
}
```

```
process p2 {  
    input C;  
    int v;  
    read(C, v);  
}
```

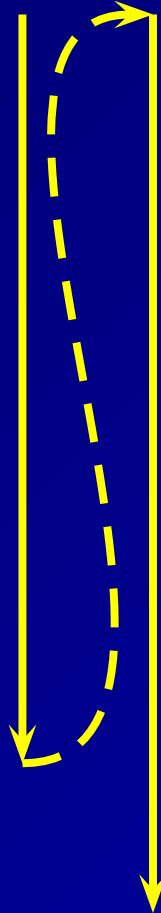
read(C, v)

Suspend

Schedule
writing process

Write before Read

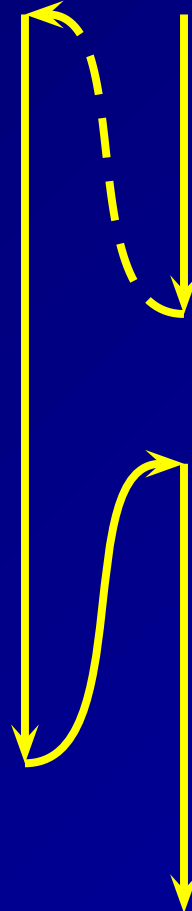
```
/* write(C, 42) */
C.value = 42;
if (C.waiting) {
    (C.waiting)->next =
        head_process;
    head_process =
        C.waiting;
}
C.waiting = &p1;
p1_state = 1;
return;
L1:
```



```
/* read(C, v) */
if (!C.waiting) {
    C.waiting = &p2;
    p2_state = 1;
    return;
}
L1:
v = C.value;
(C.waiting)->next =
    head_process;
head_process =
    C.waiting;
C.waiting = 0;
```

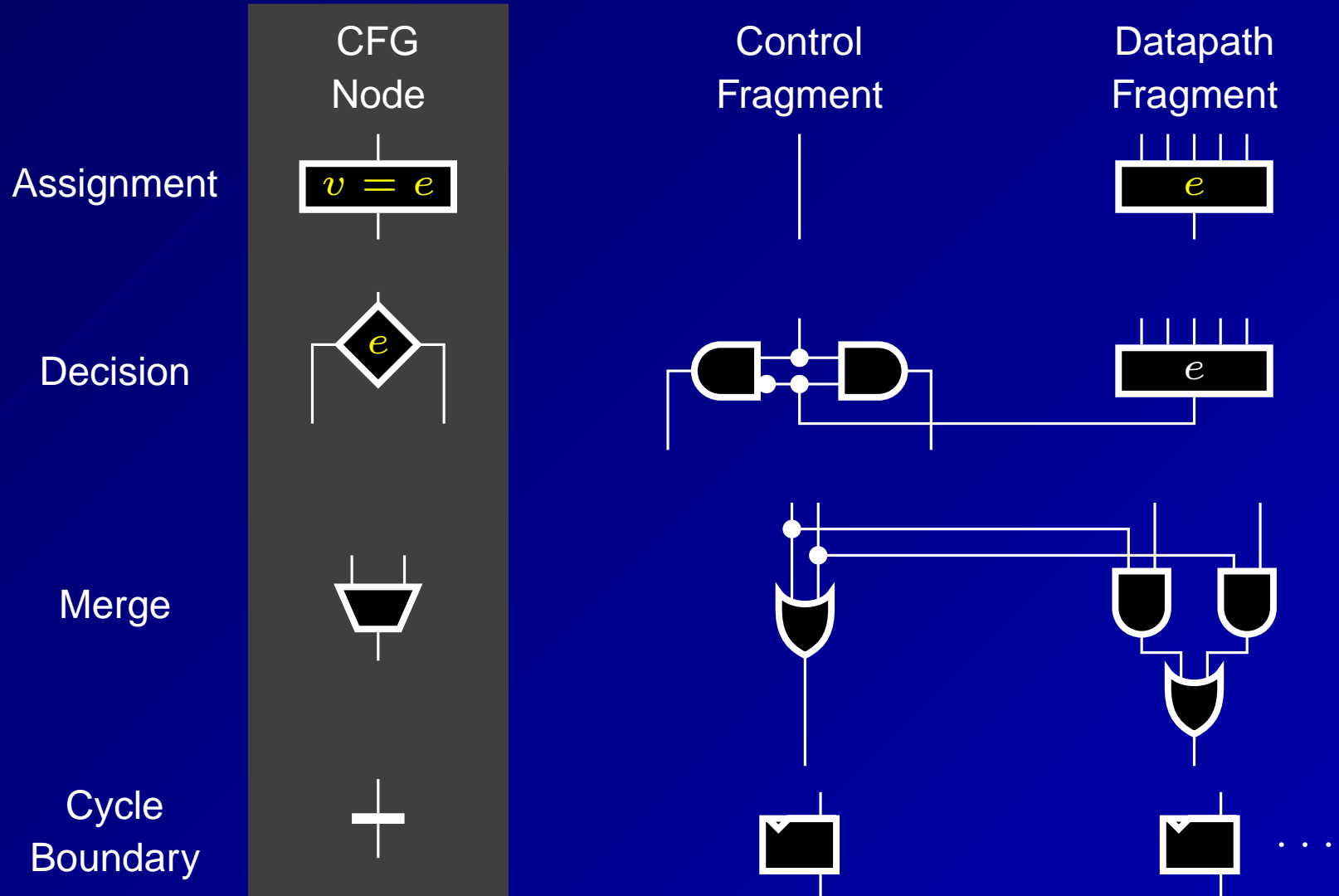
Read before Write

```
/* write(C, 42) */  
C.value = 42;  
if (C.waiting) {  
    (C.waiting)->next =  
        head_process;  
    head_process =  
        C.waiting;  
}  
C.waiting = &p1;  
p1_state = 1;  
return;  
L1:
```

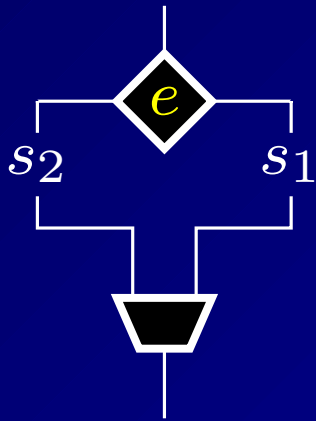


```
/* read(C, v) */  
if (!C.waiting) {  
    C.waiting = &p2;  
    p2_state = 1;  
    return;  
}  
L1:  
v = C.value;  
(C.waiting)->next =  
    head_process;  
head_process =  
    C.waiting;  
C.waiting = 0;
```

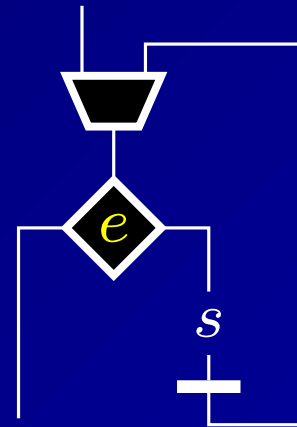
Translating Tiny-SHIM to Hardware



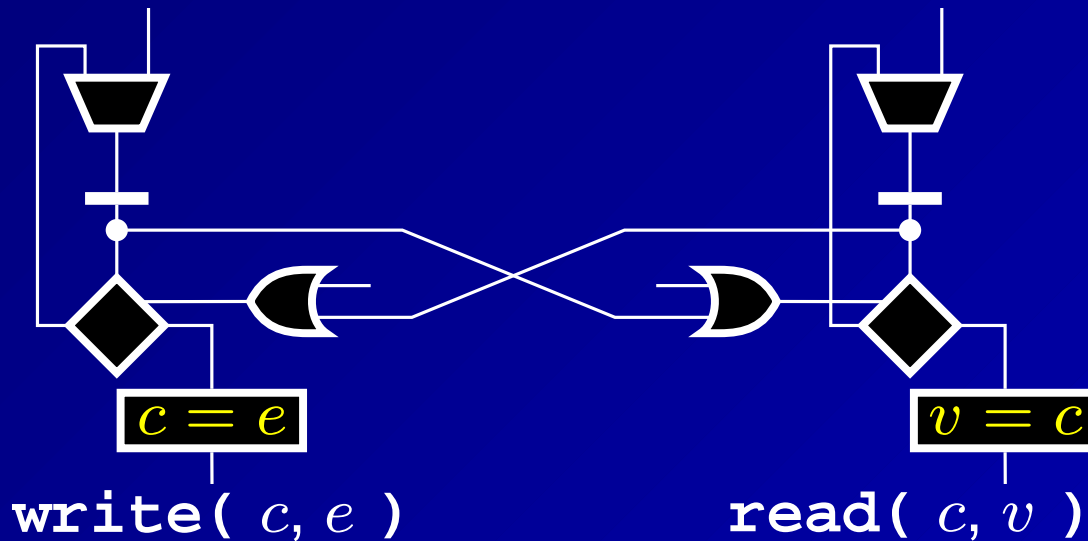
Translation Patterns



`if (e) s1 else s2`



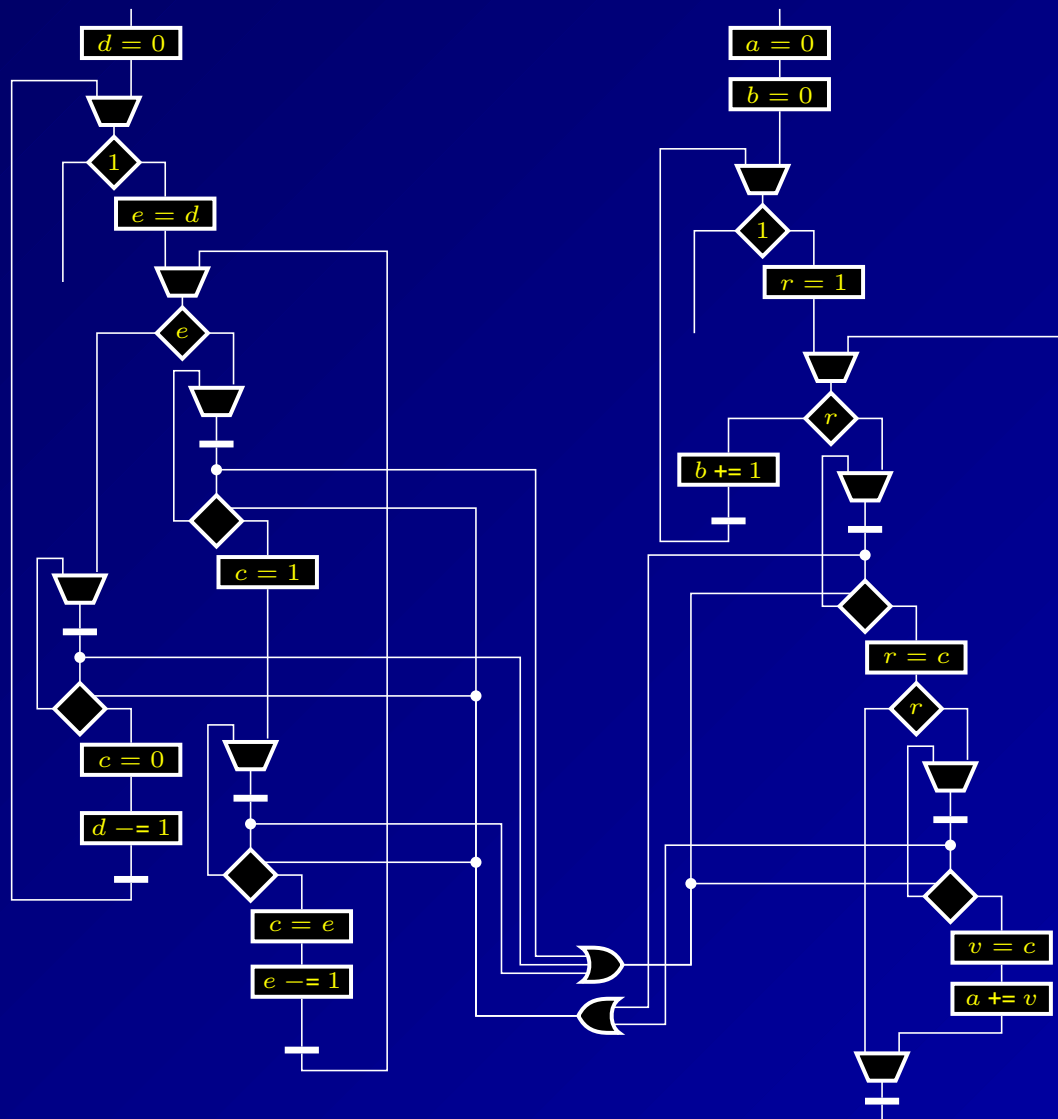
`while (e) s`



`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;
}

```

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		

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

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

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

Semantics of Looping & Sequencing

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

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

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

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

Communication and Concurrency

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

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

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

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

Summary

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

Ongoing Work

- Complete hardware/software design language
One novel aspect: exact integer arithmetic. Ranges inferred at compile time.
- Relaxation of block-on-single-channel rule
There is a way to do this without breaking determinism.
- Static analysis of deadlock
Decidable: our language is not Turing-complete
- Translation optimization for hardware and software