System-on-a-chip and the Coming Design Revolution

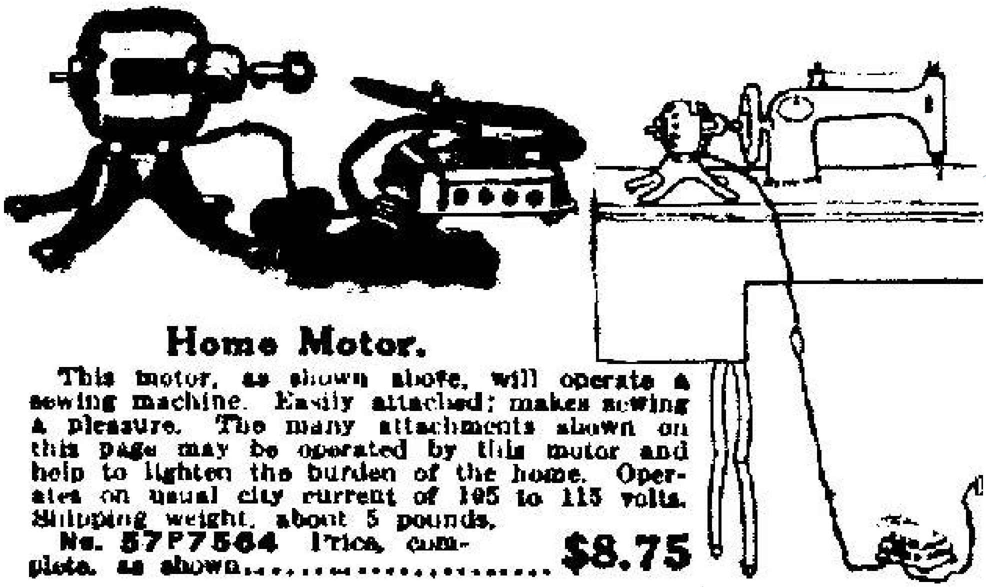
Stephen A. Edwards

Department of Computer Science,
Columbia University

www.cs.columbia.edu/~sedwards

sedwards@cs.columbia.edu

1918 Sears Roebuck Catalog



About $100 in today's dollars.

From Donald Norman, *The Invisible Computer*, 1998.

What happened to Home Motors?

Motors became cheap enough to embed in any appliance that needed them.

How many motors do you own?

2000 MacMall Catalog

How many computers do you own?

What will the SoCs of the future be?

Hint:

Hidden Computers

Casio Camera Watch

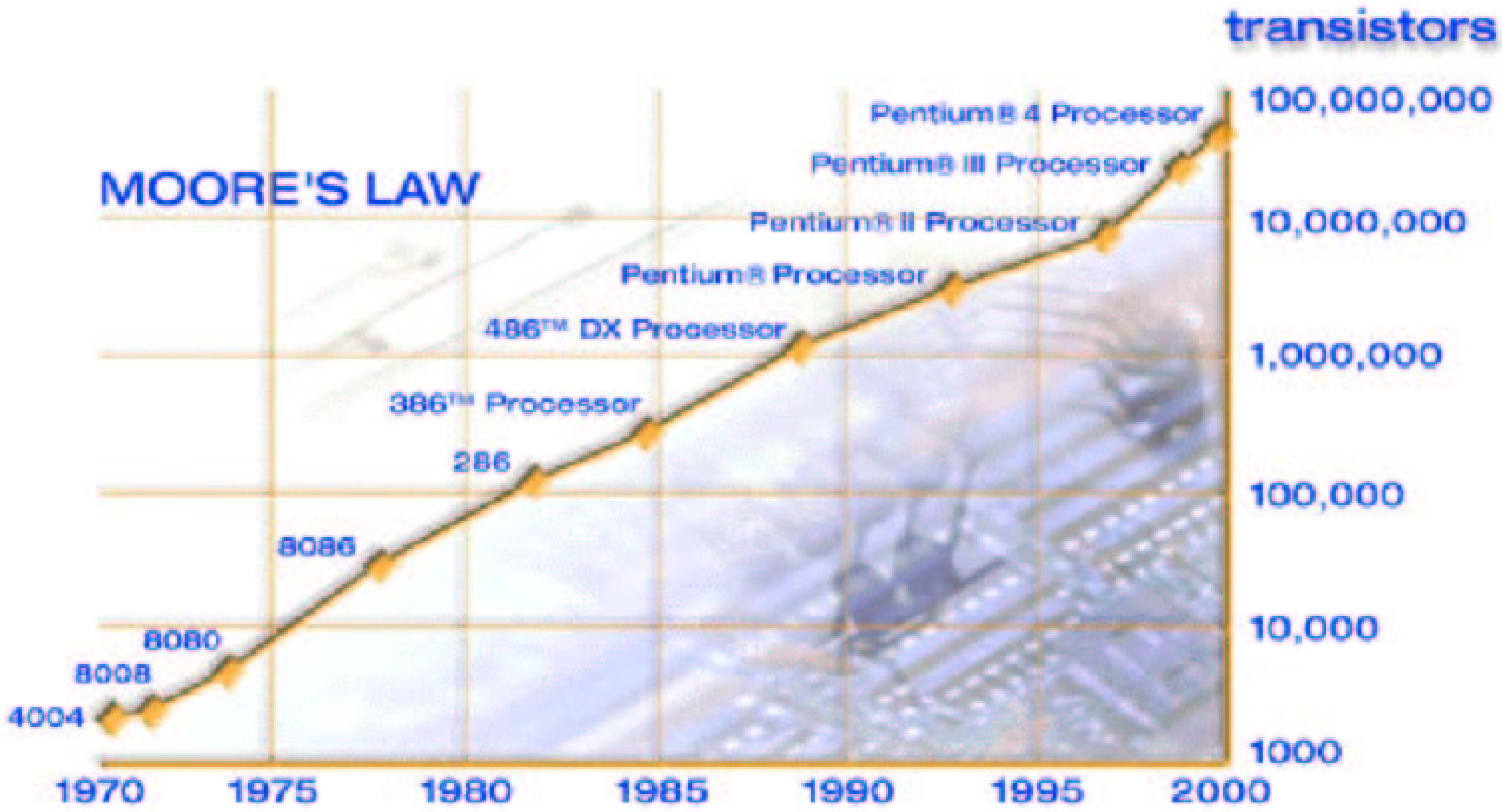
Nokia 7110 Browser Phone

Sony Playstation 2

Philips DVD Player

Philips TiVo Recorder

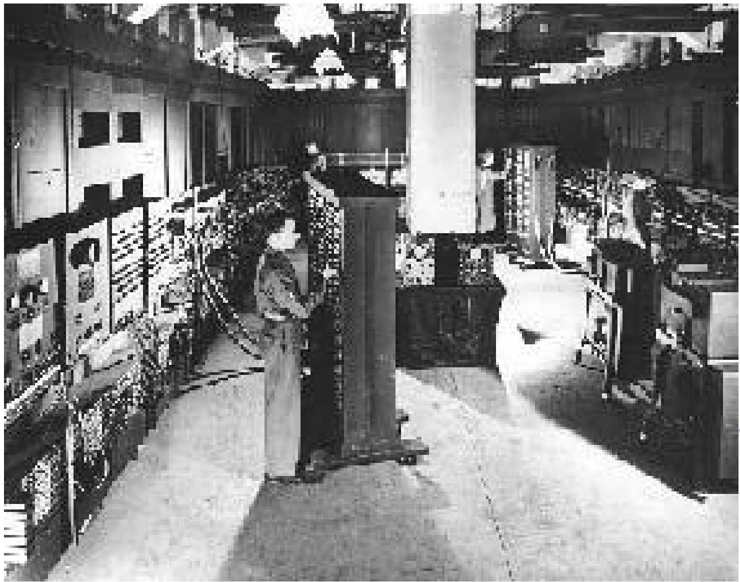
Transistor Cost Continues Plummeting



Each Pentium sold for about $600 initially.

Source: Intel

Computers’ Changing Role



Environment and humans subservient to computer

Simple peripherals

Computers subservient to humans and the environment

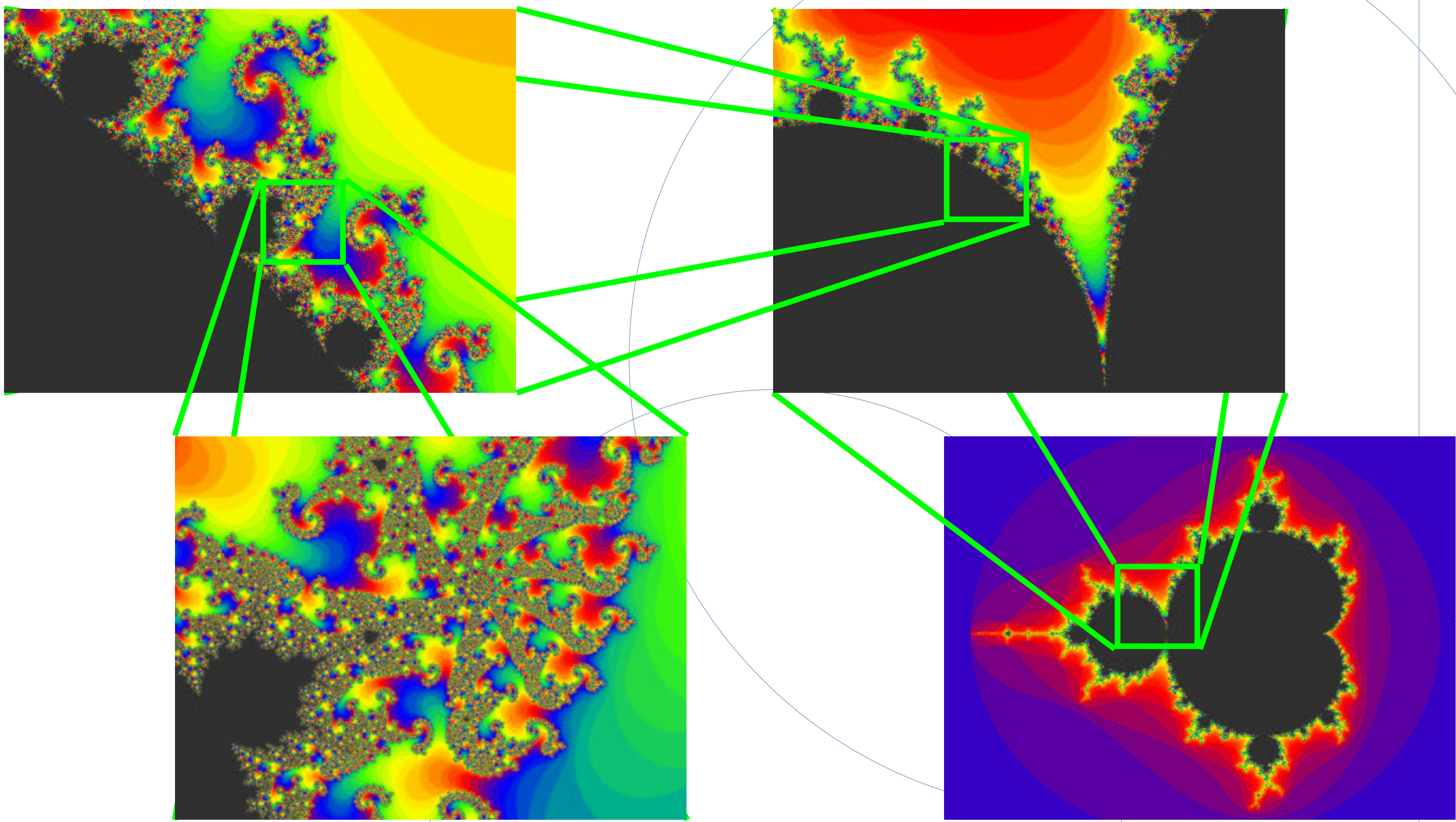
Complex peripherals

Embedded System Challenges

Real-time Deadlines

Embedded System Challenges

Complexity



Software complexity growing

Size of Typical Embedded System

1985	13 kLOC	
1989	21 kLOC	↓ 44 % per year
1998	1 MLOC	
2000	2 MLOC	
2008	16 MLOC	≈ Windows NT 4.0
2010	32 MLOC	≈ Windows 2000

Source: “ESP: A 10-Year Retrospective,” Embedded Systems Programming, November 1998

Written in stone-age languages

"Which of the following programming languages have you used for embedded systems in the last 12 months?"

C	81%
Assembly	70%
C++	39%
Visual Basic	16%
Java	7%

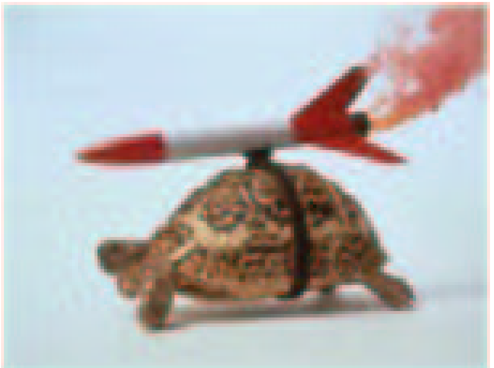
Source: "ESP: A 10-Year Retrospective," Embedded Systems Programming, November 1998

Embedded System Challenges

Concurrency


Photo by Thomas Danoghue

Existing Techniques



...aren't up to the task.

- Existing multi-threaded concurrency models

  ...are completely unstructured

  The "goto" of control

- Most real-time scheduling

  ...ignores communication aspects

We need some alternatives!

An Alternative: Esterel

Domain-specific language for safety-critical, real-time systems.

Uses a synchronous model of time that is deterministic and provides precise control over time.

Timing verification becomes checking a single worst-case-execution-time bound.

Timing

Java

```
class PClock
 implements Runnable {
 public void run() {
  for (;;) {
   java.util.Date now =
      new java.util.Date();
   System.out.
     println(now.toString());
   try {
     Thread.currentThread().
       sleep(1000);
   } catch (IntExcept e) {}
  }
 }
}
public class Clock {
  public static void
   main(String args[]) {
    Thread t =
      new Thread(new PClock());
    t.start();
  }
}
```

```
$ java Clock
Sat Sep 14 13:04:27 EDT 2002
Sat Sep 14 13:04:29 EDT 2002
Sat Sep 14 13:04:30 EDT 2002
Sat Sep 14 13:04:31 EDT 2002
```

A Leap Second?

Esterel

```
every 1000 MS do
   emit SECOND
end
```

Just works

An Example

```
emit B;
present C then
  emit D end;
```

Force signal present in this cycle

Make D present if C is

An Example

Wait for next cycle where A is present

```
await A;
emit B;
present C then
  emit D end;
pause
```

Wait for next cycle

An Example

```
loop
   await A;
   emit B;
   present C then
      emit D end;
   pause
end
```

Infinite Loop

An Example

```
loop
    await A;
    emit B;
    present C then
        emit D end;
    pause
end
||
loop
    present B then
        emit C end;
    pause
end
```

Run Concurrently

An Example

```
every R do
  loop
    await A;
    emit B;
    present C then
      emit D end;
    pause
  end
||
  loop
    present B then
      emit C end;
    pause
  end
end
```

Restart on R

An Example

```
every R do
   loop
      await A;
      emit B;
      present C then
         emit D end
      pause
   end
||
   loop
      present B then
         emit C end;
      pause
   end
end
```

Same-cycle bidirectional communication

An Example

```
every R do
   loop
      await A;
      emit B;
      present C then
         emit D end;
      pause
   end
||
   loop
      present B then
         emit C end;
      pause
   end
end
```

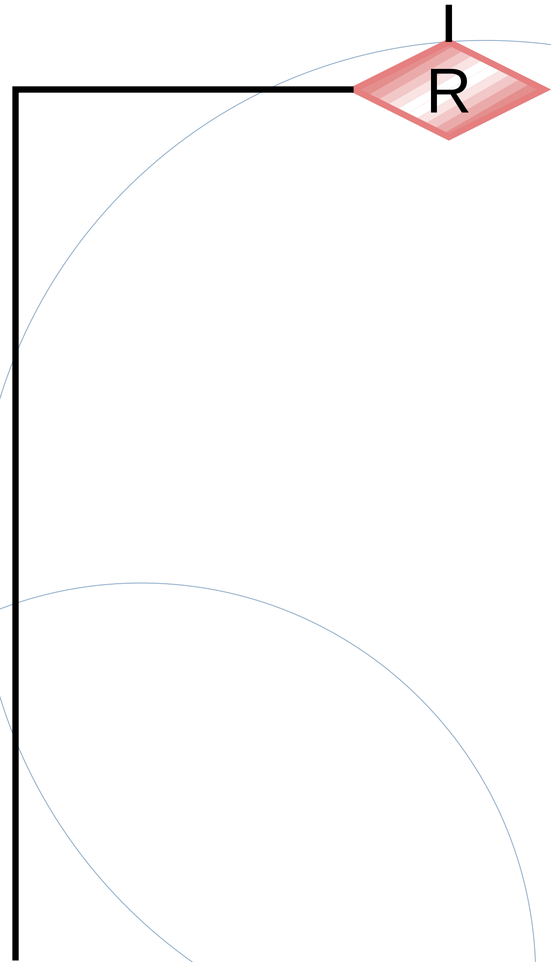
Good for hierarchical FSMs

Bad at manipulating data

Hardware Esterel variant proposed to address this

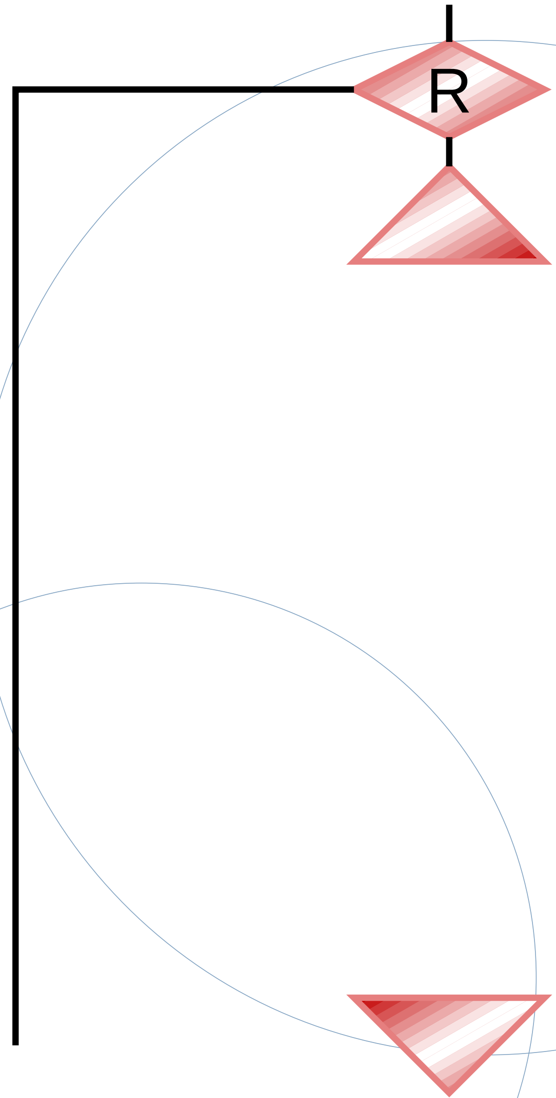
Translate every

```
every R do
  loop
    await A;
    emit B;
    present C then
      emit D end;
    pause
  end
||
  loop
    present B then
      emit C end;
    pause
  end
end
```



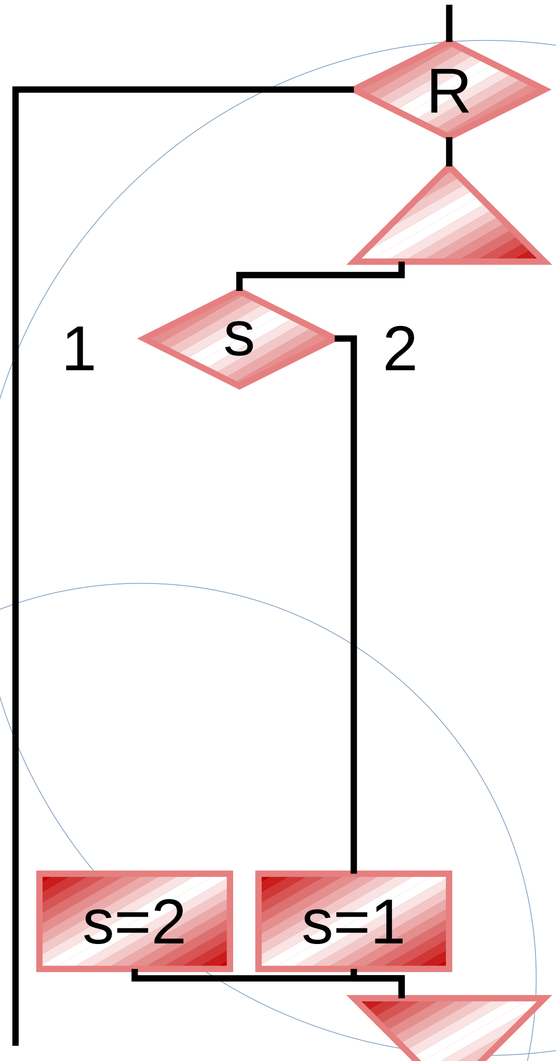
Add Threads

```
every R do
   loop
      await A;
      emit B;
      present C then
         emit D end;
      pause
   end
||
   loop
      present B then
         emit C end;
      pause
   end
end
```



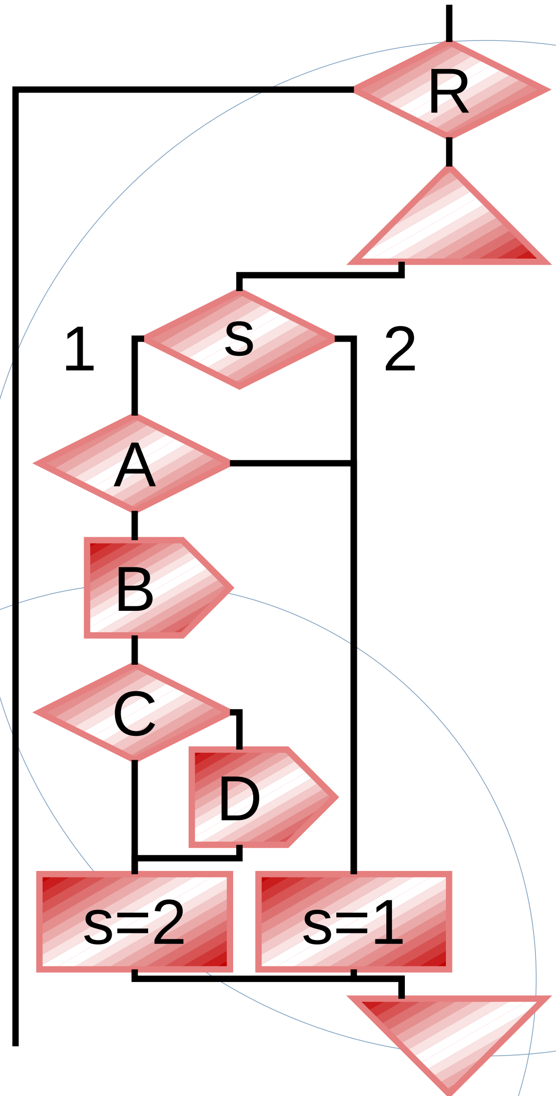
Split at Pauses

```
every R do
   loop
      await A;
      emit B;
      present C then
         emit D end;
      pause
   end
||
   loop
      present B then
         emit C end;
      pause
   end
end
```



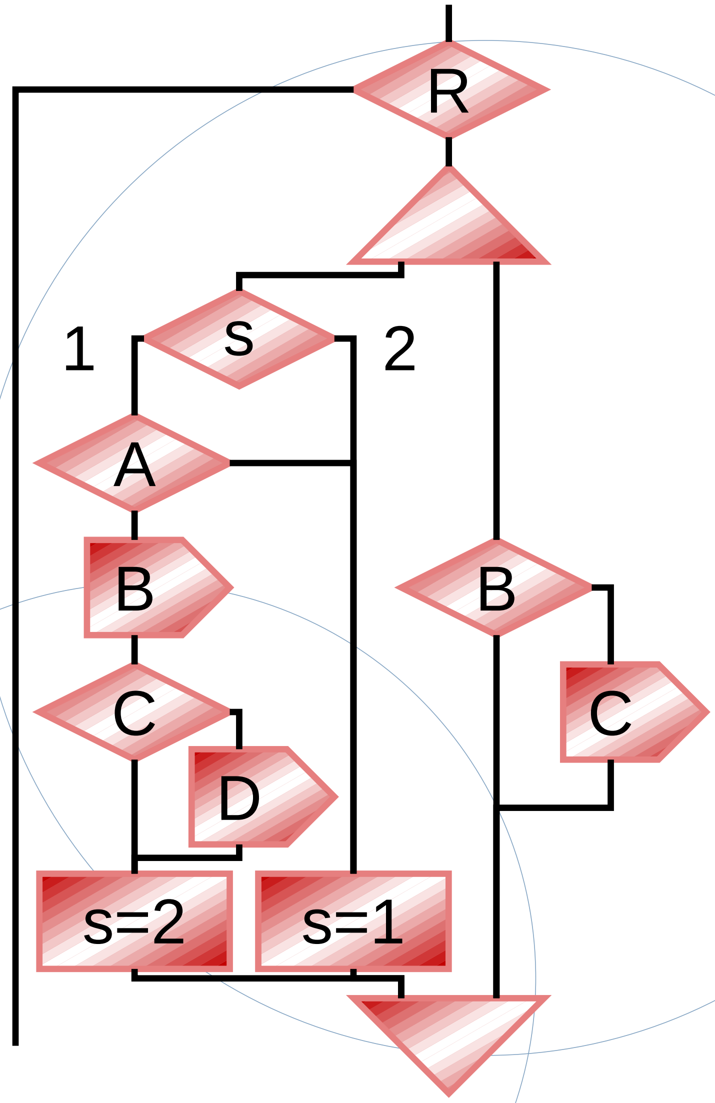
Add Code Between Pauses

```
every R do
  loop
    await A;
    emit B;
    present C then
      emit D end;
    pause
  end
||
  loop
    present B then
      emit C end;
    pause
  end
end
```



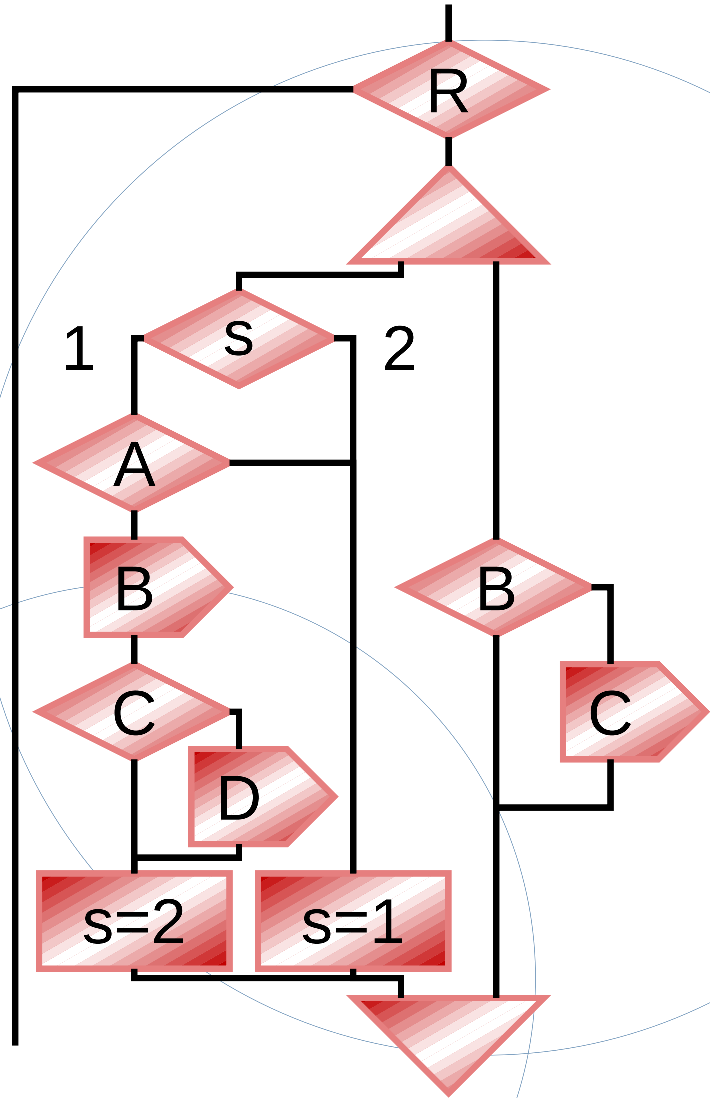
Translate Second Thread

```
every R do
    loop
        await A;
        emit B;
        present C then
            emit D end;
        pause
    end
||
    loop
        present B then
            emit C end;
        pause
    end
end
```



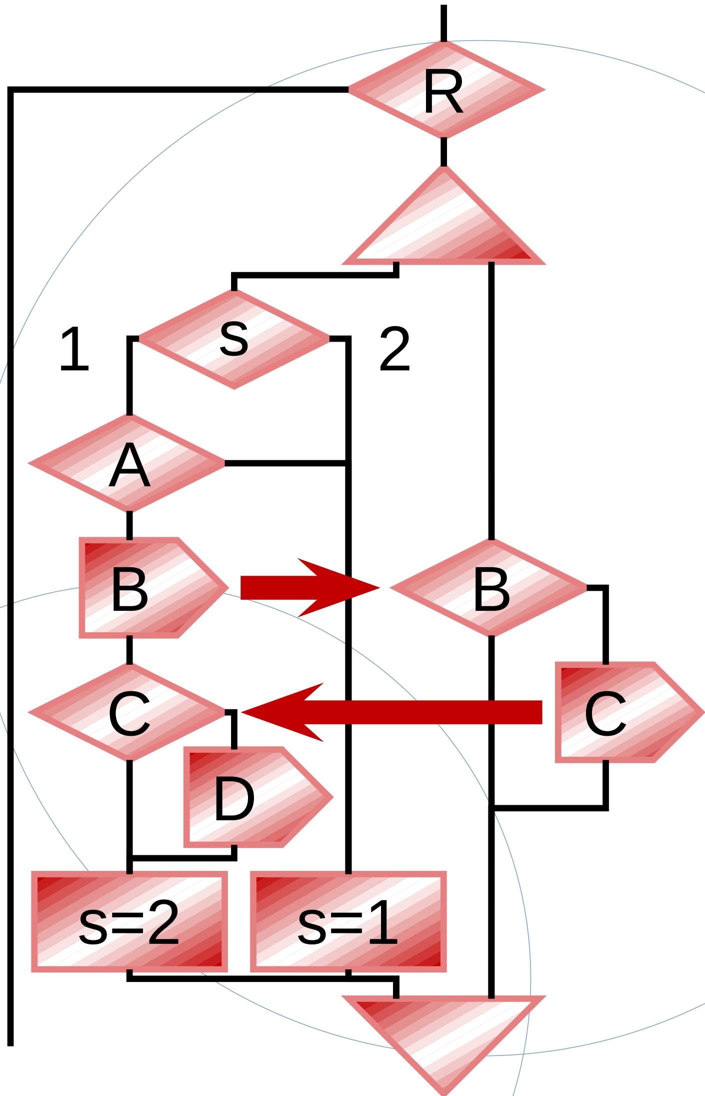
Finished Translating

```
every R do
   loop
      await A;
      emit B;
      present C then
         emit D end;
      pause
   end
||
   loop
      present B then
         emit C end;
      pause
   end
end
```

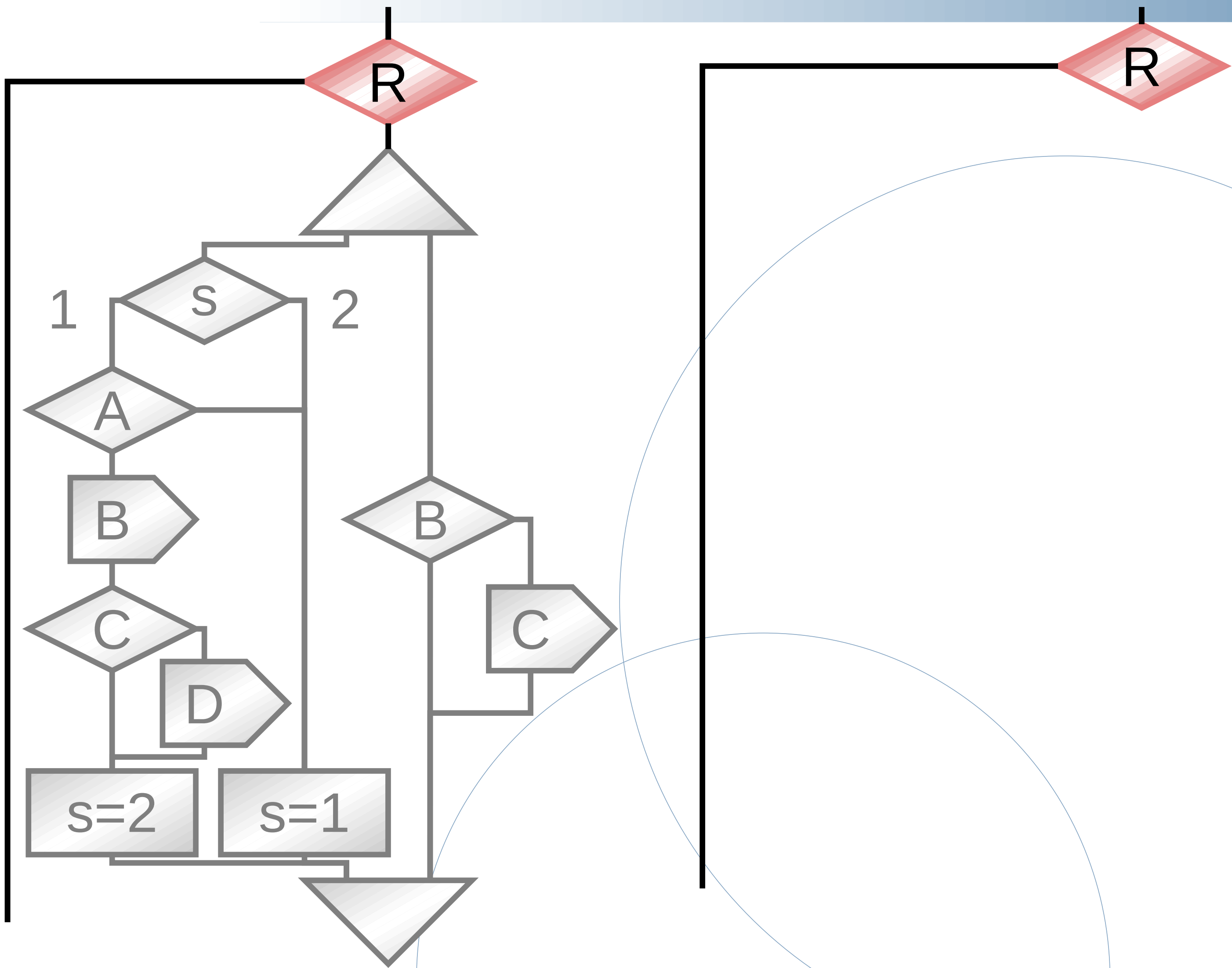


Add Dependencies and Schedule

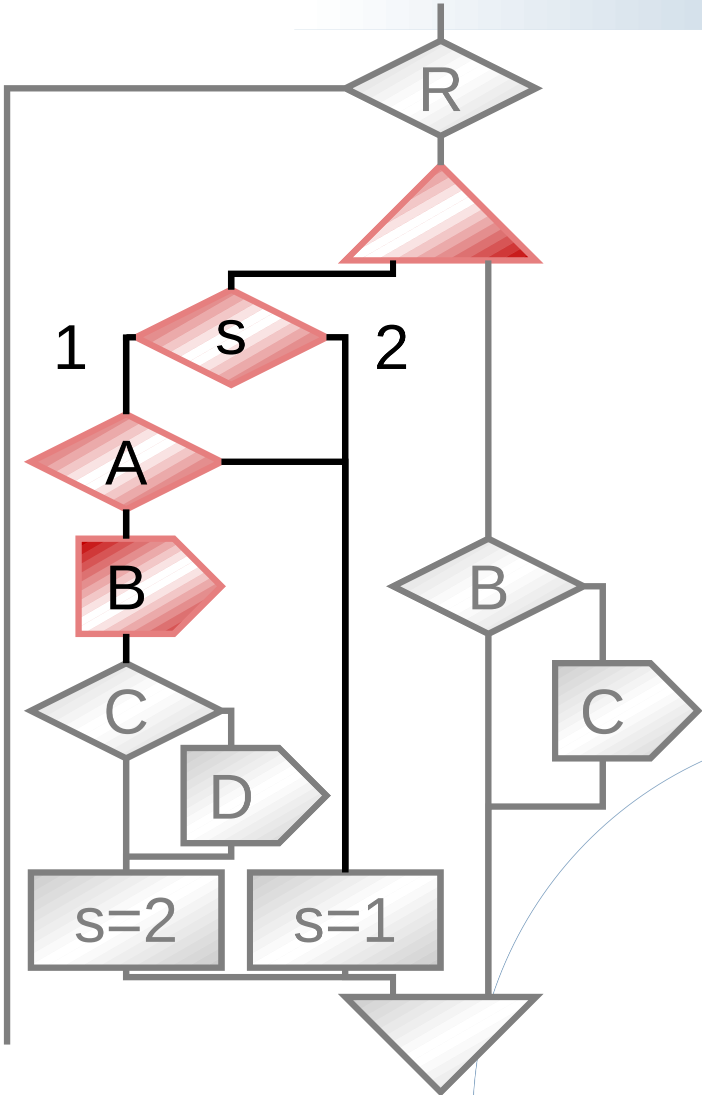
```
every R do
   loop
      await A;
      emit B;
      present C then
         emit D end;
      pause
   end
||
   loop
      present B then
         emit C end;
      pause
   end
end
```

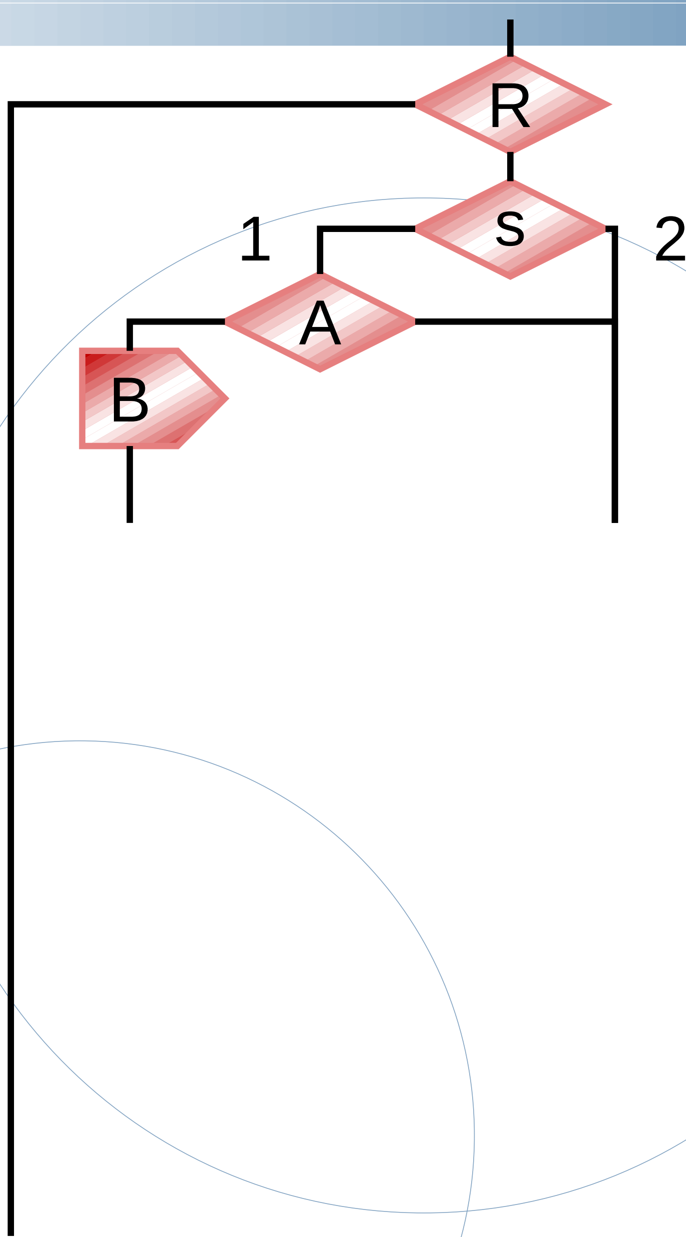


Run First Node

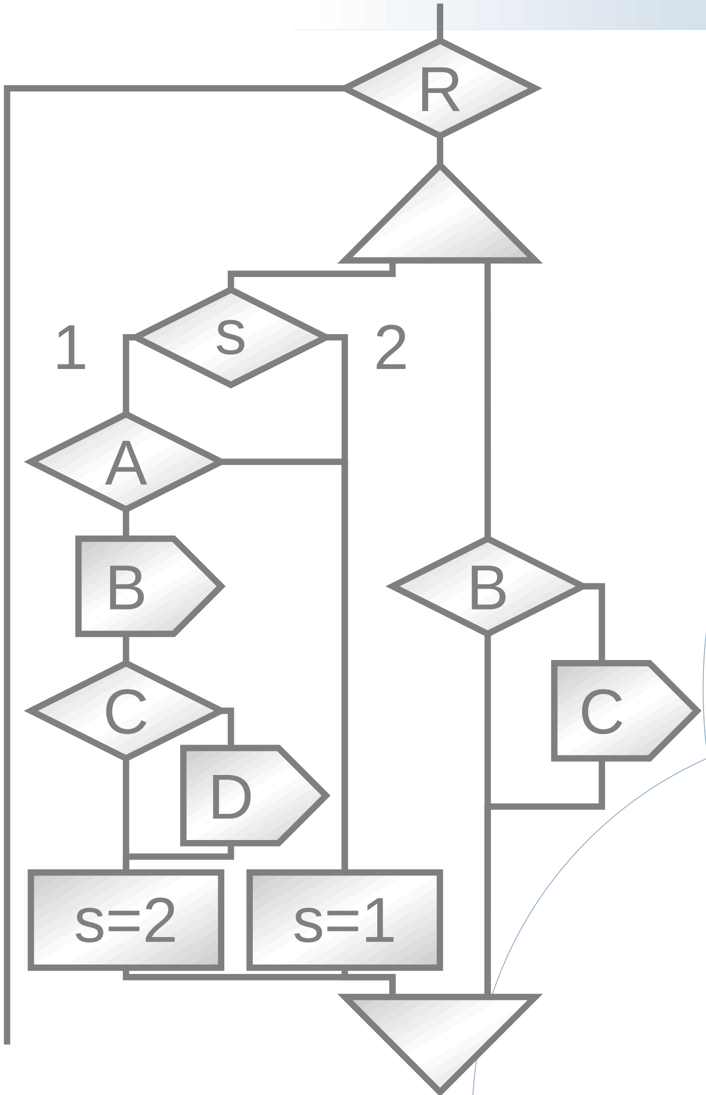


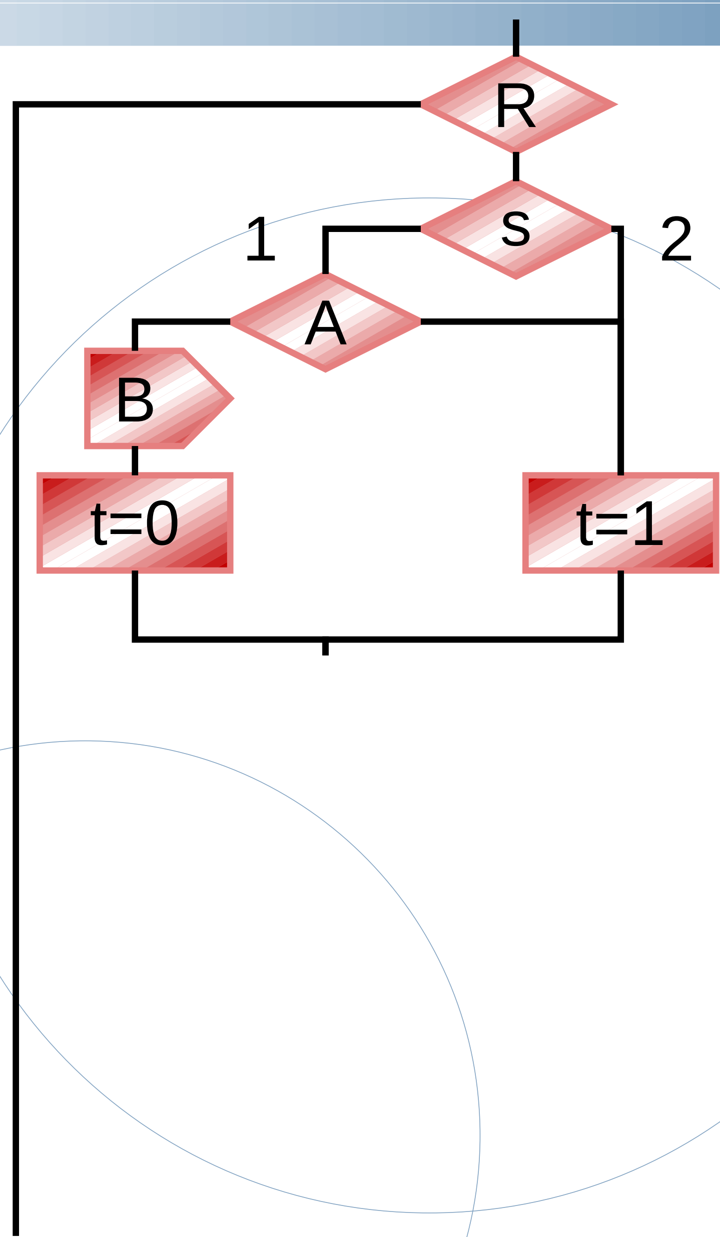
Run First Part of Left Thread



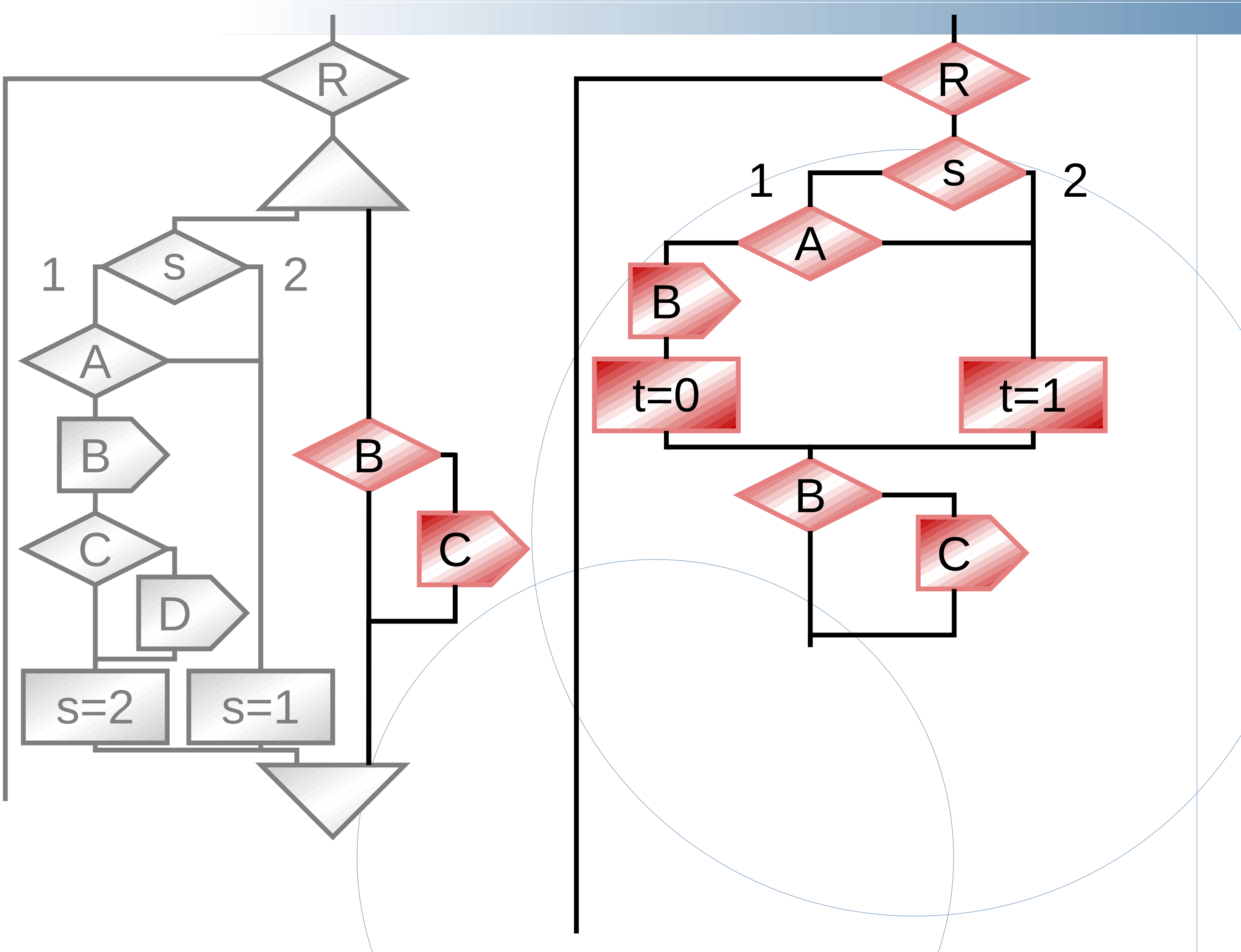


Context Switch

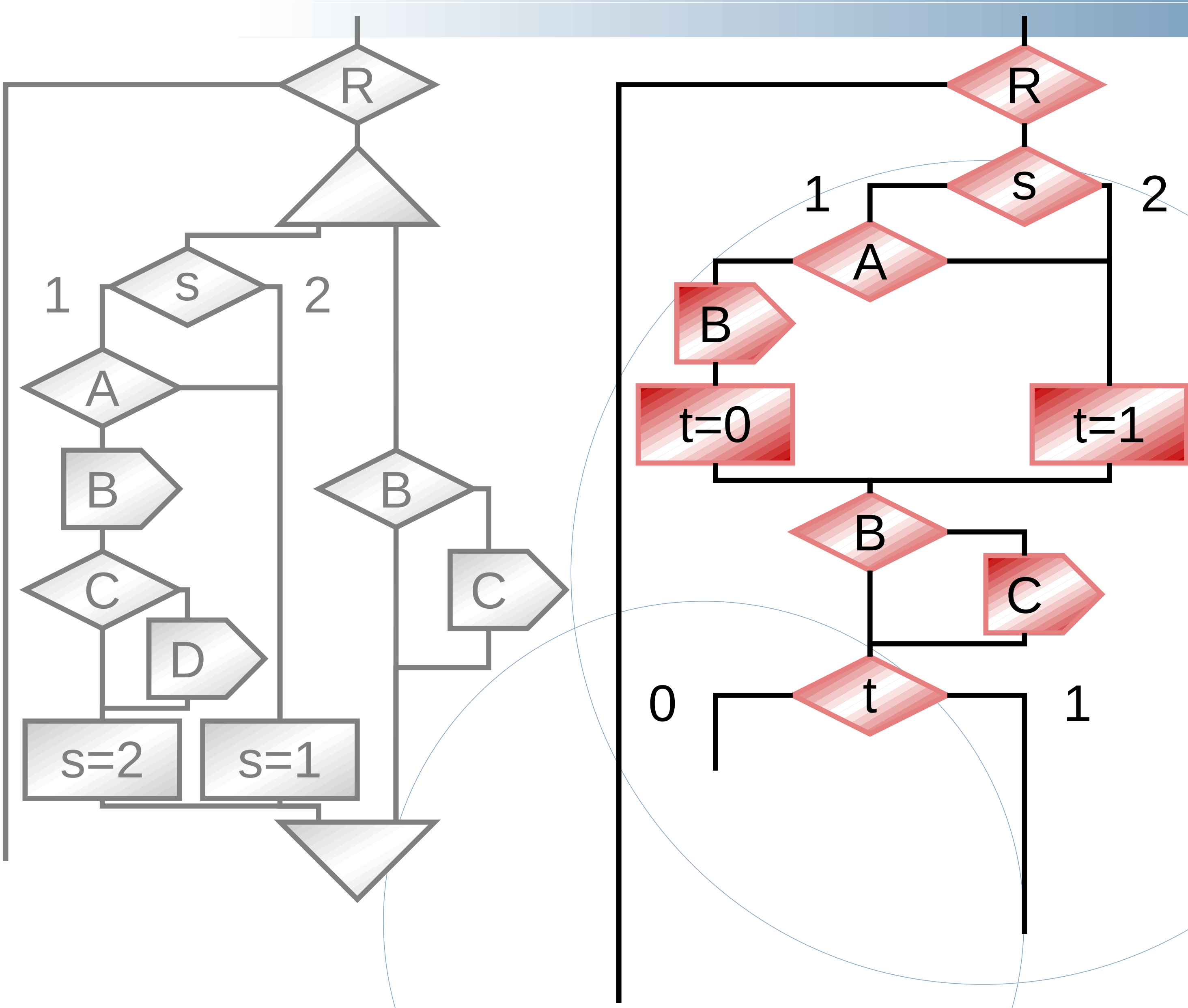




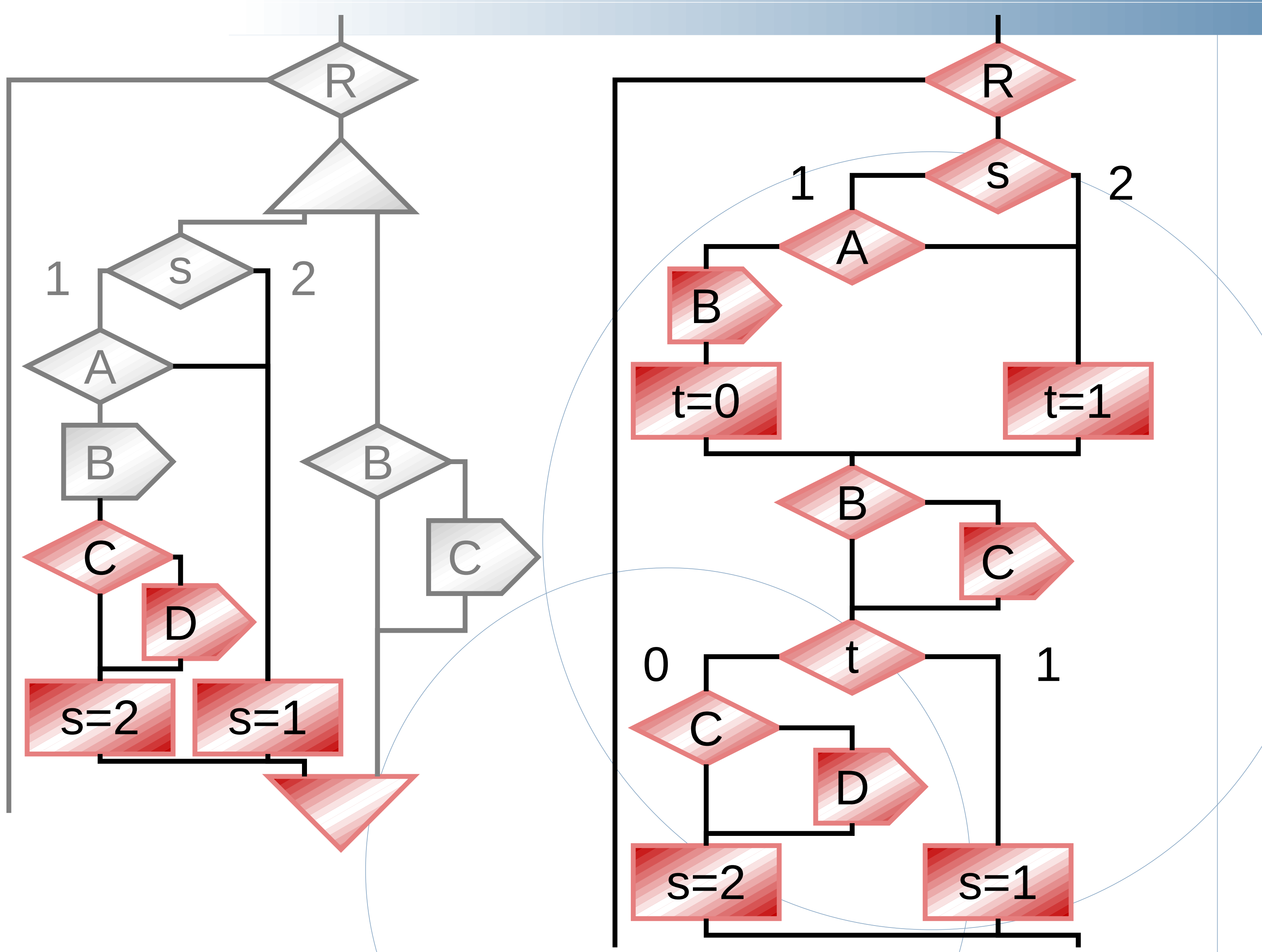
Run Right Thread



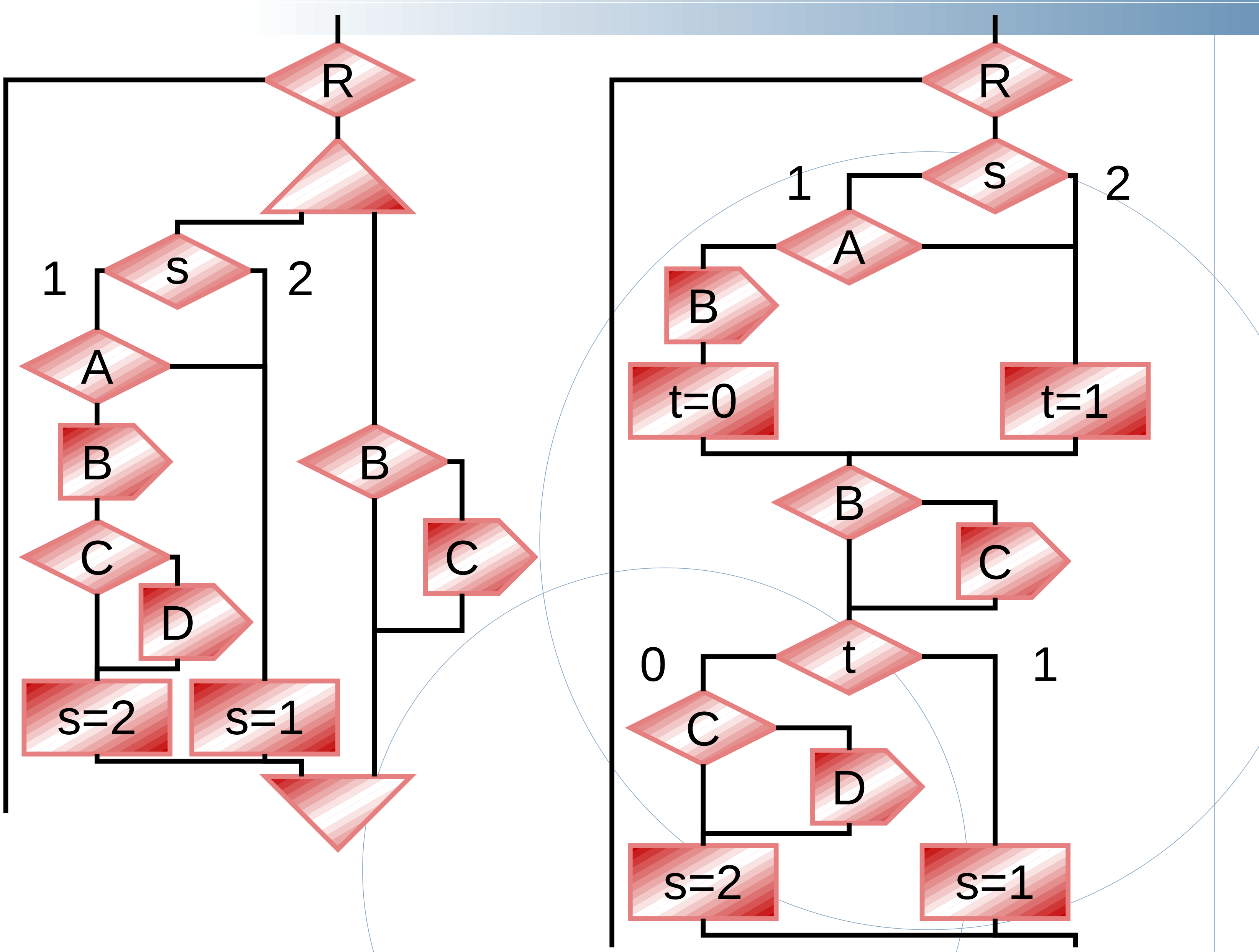
Context Switch



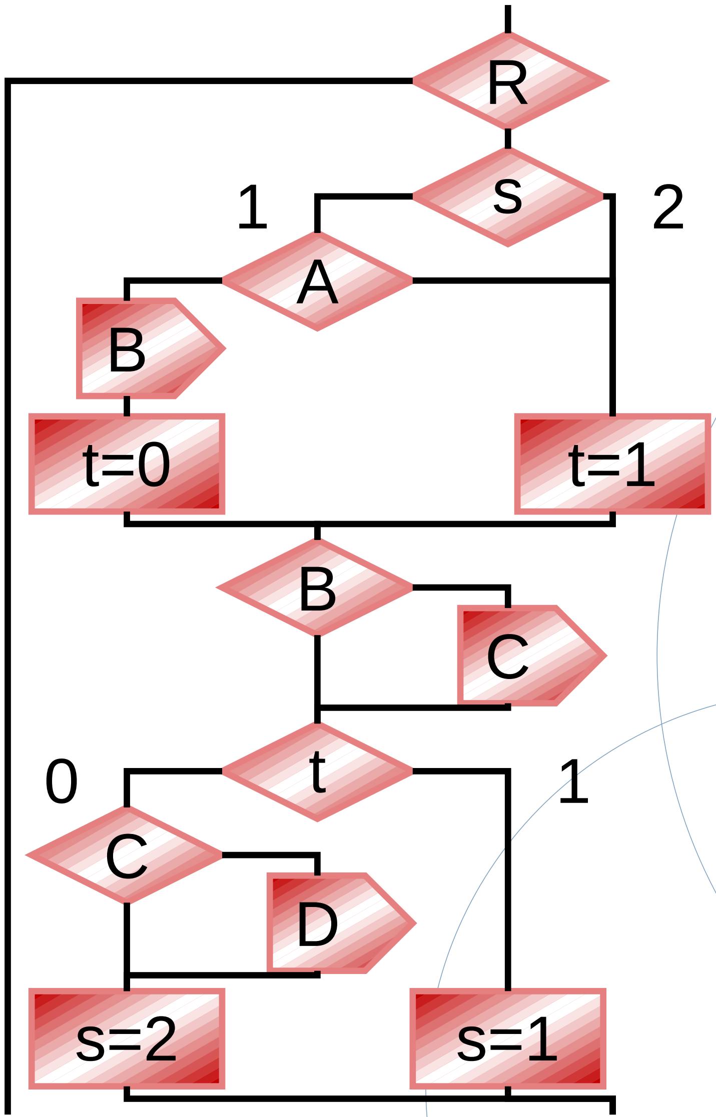
Finish Left Thread



Completed Example



Generated Code




```
if (!R) {
  if (s == 1 && A) {
    B = 1;
    t = 0;
  } else {
    t = 1;
  }
  if (B) C = 1;
  if (t == 0) {
    if (C) D = 1;
    s = 2;
  } else {
    s = 1
  }
}
```

Summary

Plummeting transistor cost is making it practical to put more, smaller computer systems everwhere.

Implemented with SoC technology, these embedded systems will be dominated by software.

Embedded system challenges:

- Real-time issues
- Concurrency
- Software complexity and reliability

Summary

Esterel and the synchronous paradigm solve some problems

- Synchronous model provides deterministic concurrency
- Finite state permits automatic model checking
- Execution time verification provides timing assurance
- Efficient compilation scheme eliminates OS overhead