

Compiling Esterel

Stephen A. Edwards

Department of Computer Science,
Columbia University

www.cs.columbia.edu/~sedwards

sedwards@cs.columbia.edu

Outline

Introduction to Esterel and Existing Compilers

My Software Compiler [DAC 2000, TransCAD 2002]

My Hardware Compiler [SLAP 2002, IWLS 2002]

The Esterel Language

Developed by Gérard Berry starting 1983

Originally for robotics applications

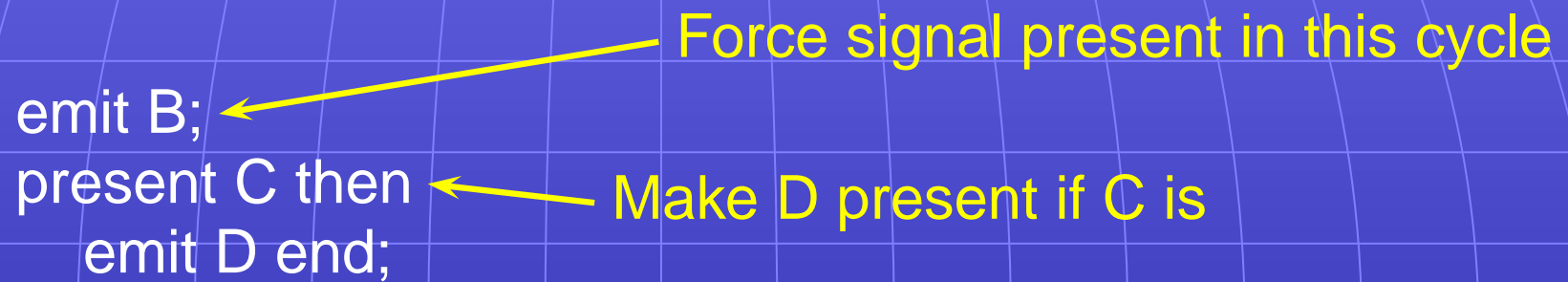
Imperative, textual language

Synchronous model of time like that in digital circuits

Concurrent

An Example

```
emit B; ← Force signal present in this cycle  
present C then ← Make D present if C is  
emit D end;
```

The diagram consists of two yellow arrows pointing from explanatory text to code. The first arrow originates from the text 'Force signal present in this cycle' and points to the code line 'emit B;'. The second arrow originates from the text 'Make D present if C is' and points to the code line 'present C then'. The code is displayed in white text on a blue background with a grid pattern.

An Example

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

Wait for next cycle where A is present

Wait for next cycle

An Example

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

An Example

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

||

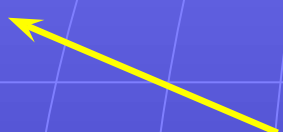
```
loop
  present B then
    emit C end;
  pause
end
```

← 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

Automata Compilers

Esterel is a finite-state language, so build an automata:

```
loop                               switch (s) {
  emit A; await C;                 case 0: A = 1; s = 1; break;
  emit B; pause                    case 1: if (C) { B = 1; s = 0; } break;
end                                 }
```

V1, V2, V3 (INRIA/CMA) [Berry, Gonthier 1992]

Fastest known code; great for programs with few states.

Does not scale; concurrency causes state explosion.

Discrete-Event Based Compilers

SAXO-RT [Weil et al. 2000] Divides Esterel program into event functions dispatched by a fixed scheduler.

```
loop          unsigned curr = 0x1;
              unsigned next = 0;
  emit A; await C;
  emit B; pause
end           }
              static void f1() {
              A = 1;
              curr &= ~0x1; next |= 0x2;
              }
              static void f2() {
              if (!C) return;
              B = 1;
              curr &= ~0x2; next |= 0x1;
              }
              void tick() {
              if (curr & 0x1) f1();
              if (curr & 0x2) f2();
              curr |= next;
              next = 0;
              }
```

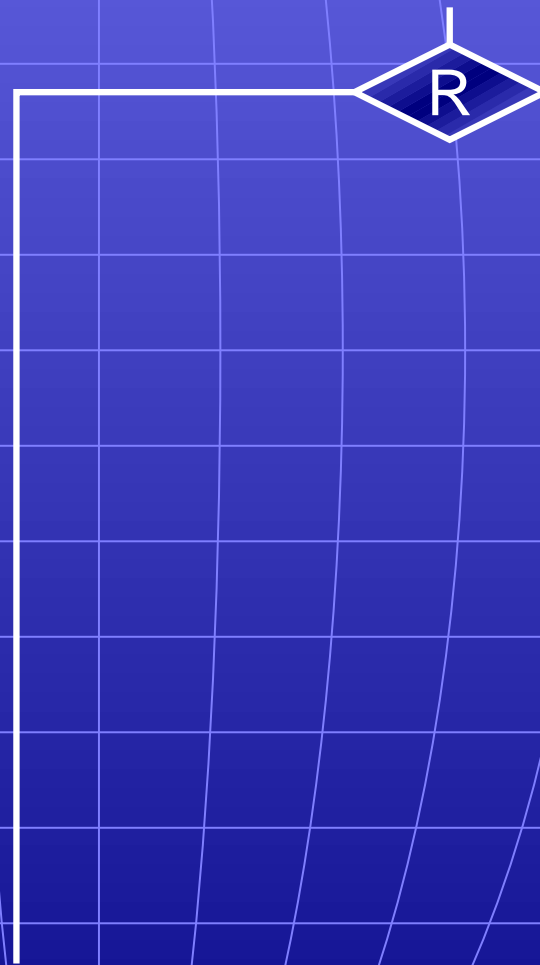
My Esterel Compiler for Software

Presented at DAC 2000 (also TransCAD 2002)

Used inside Synopsys' CoCentric System Studio to
generate control code

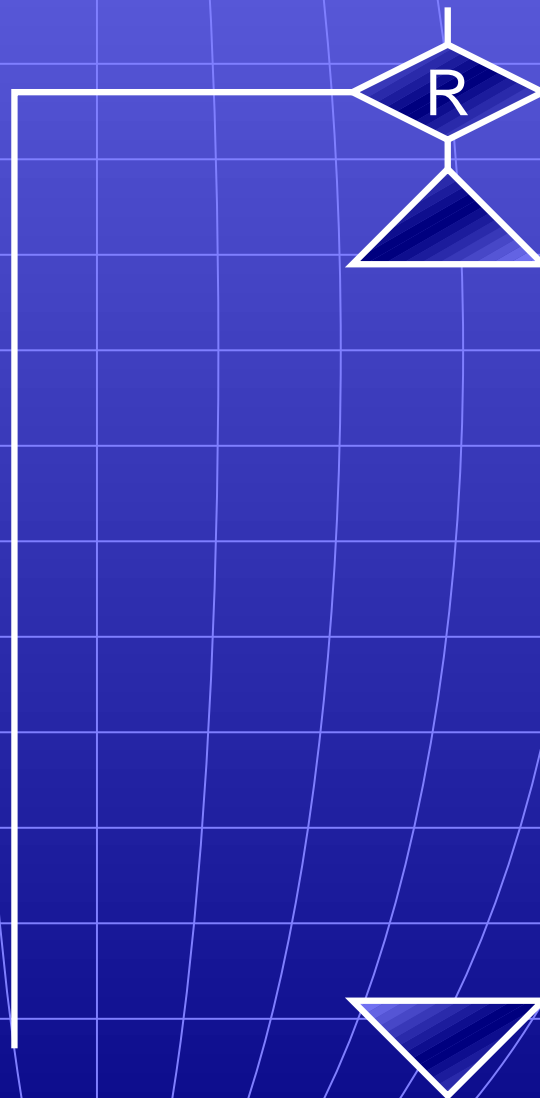
Translate every

```
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
```



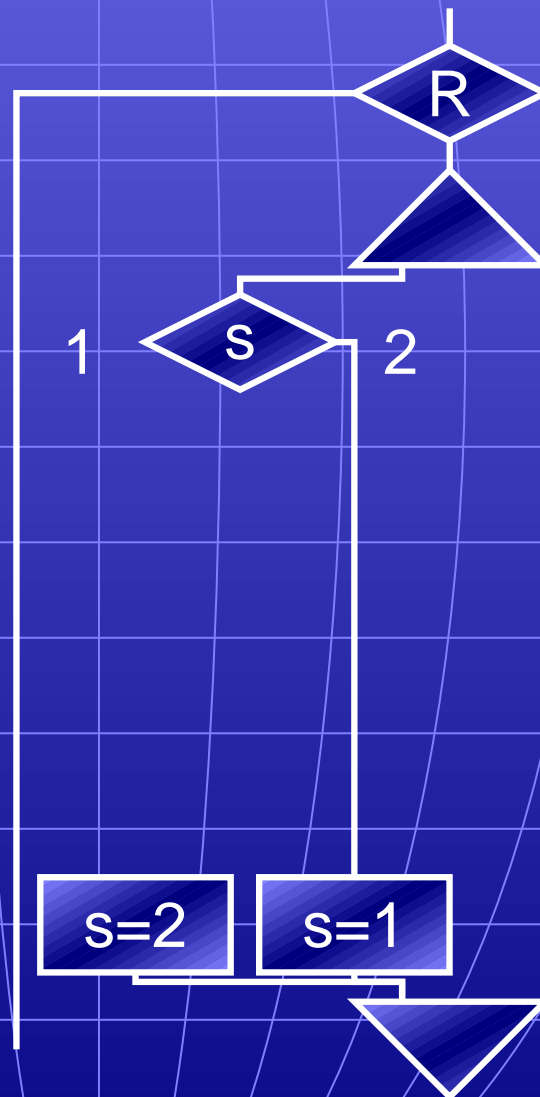
Add Threads

```
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
```



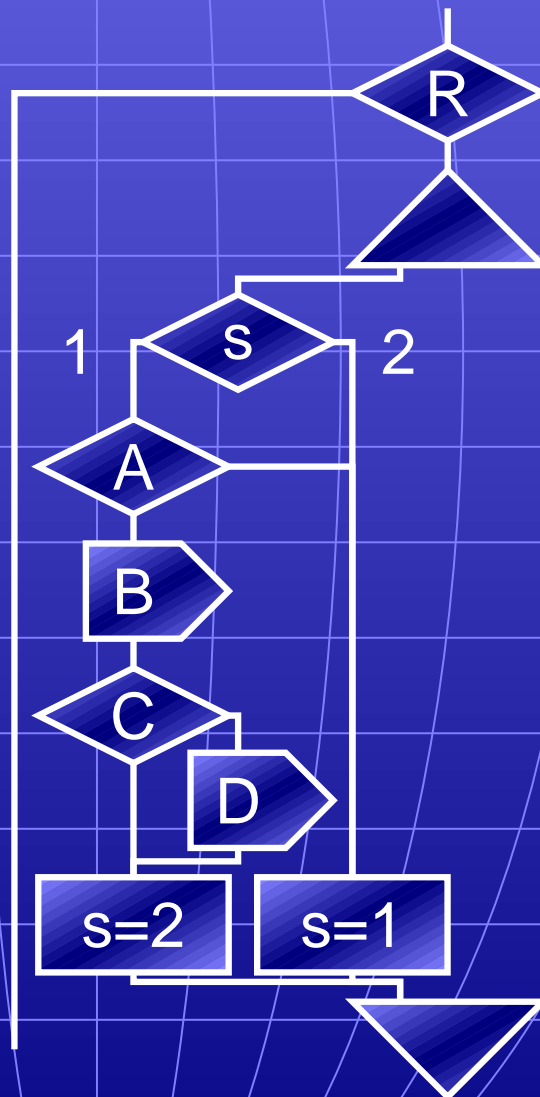
Split at Pauses

```
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
```



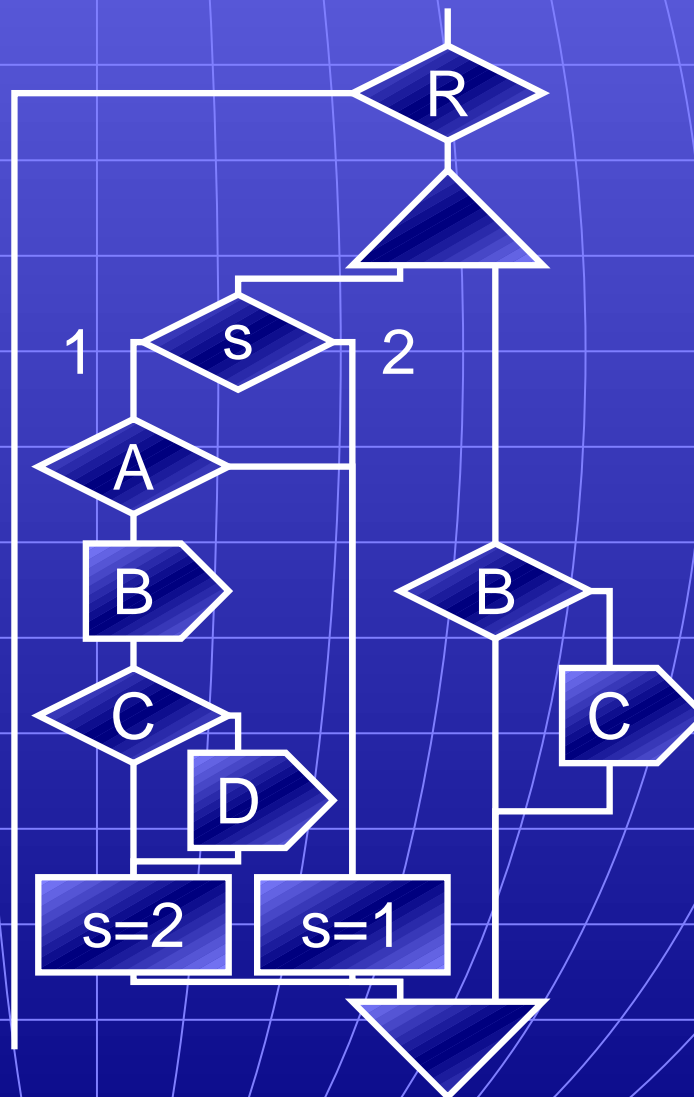
Add Code Between Pauses

```
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
```



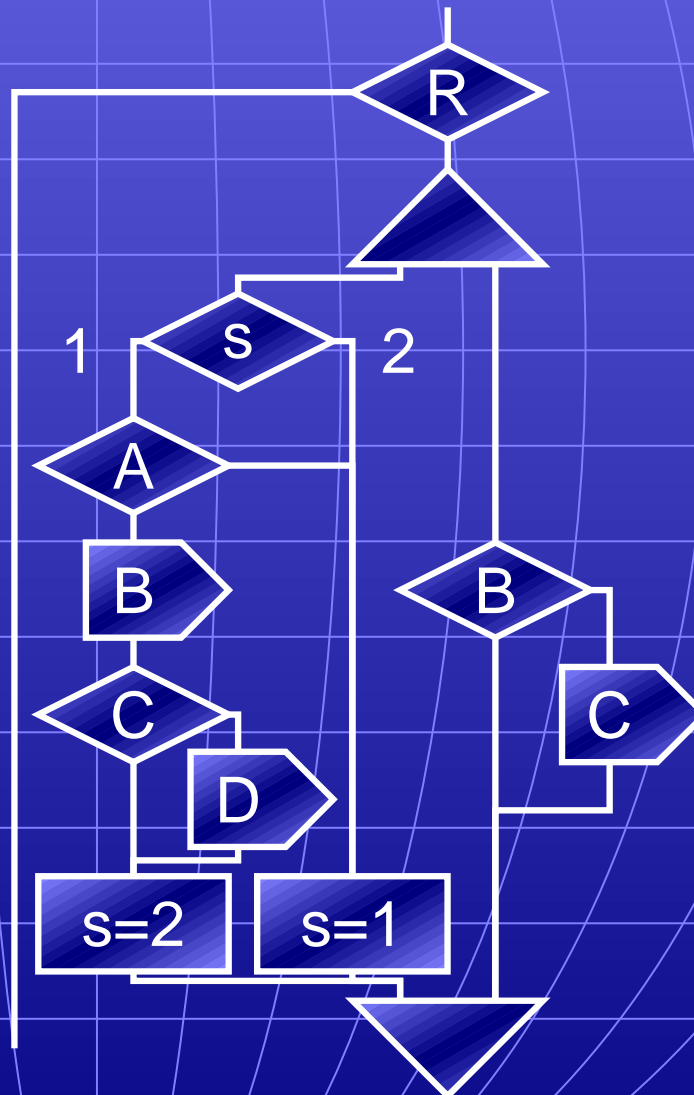
Translate Second Thread

```
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
```



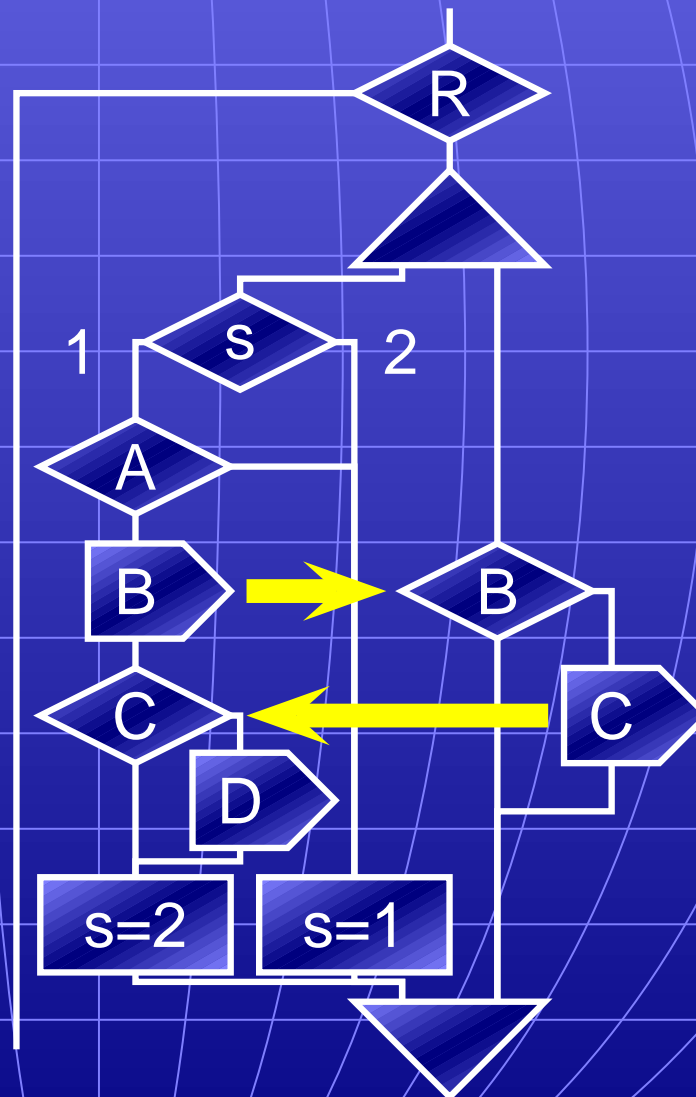
Finished Translating

```
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
```

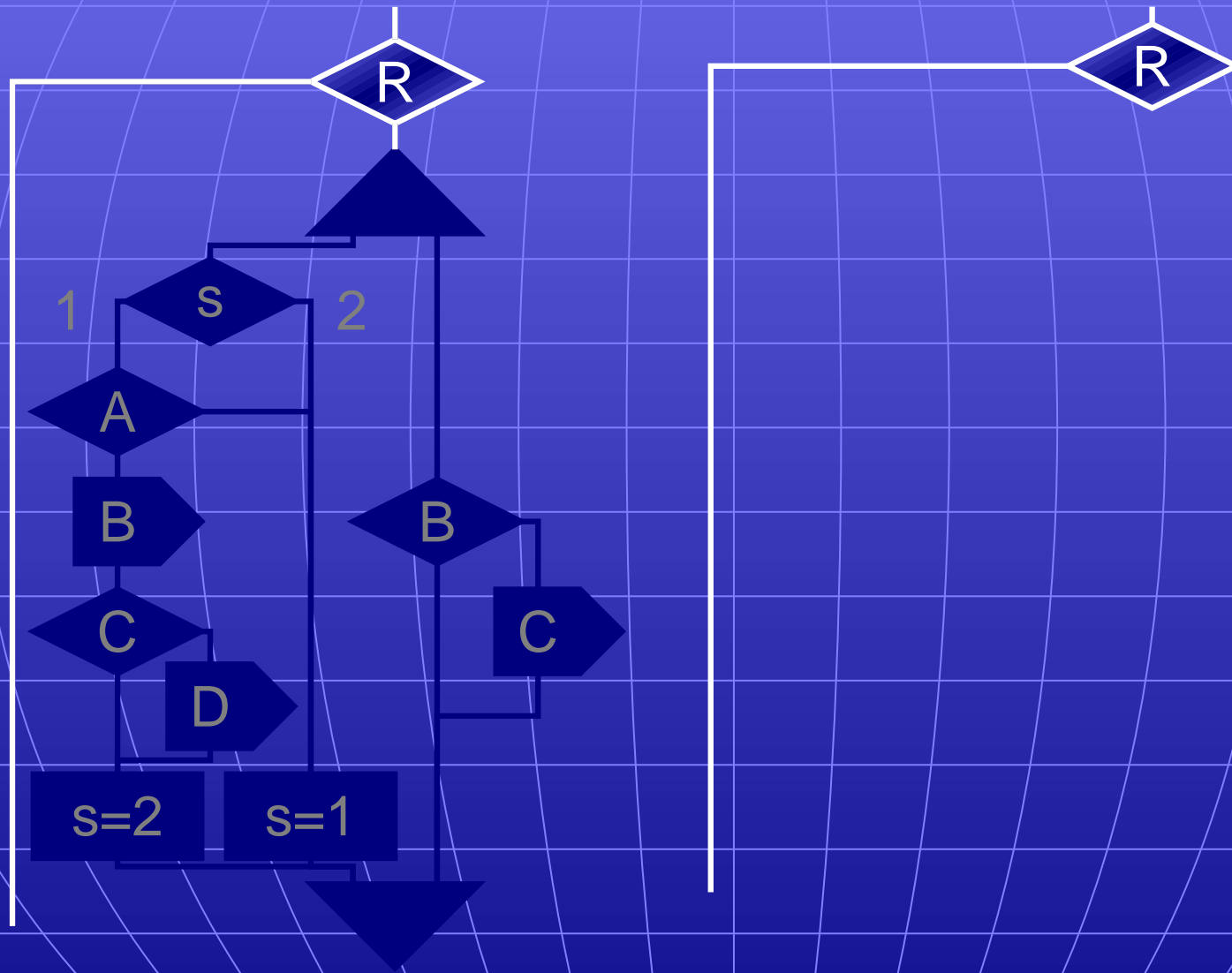


Add Dependencies and Schedule

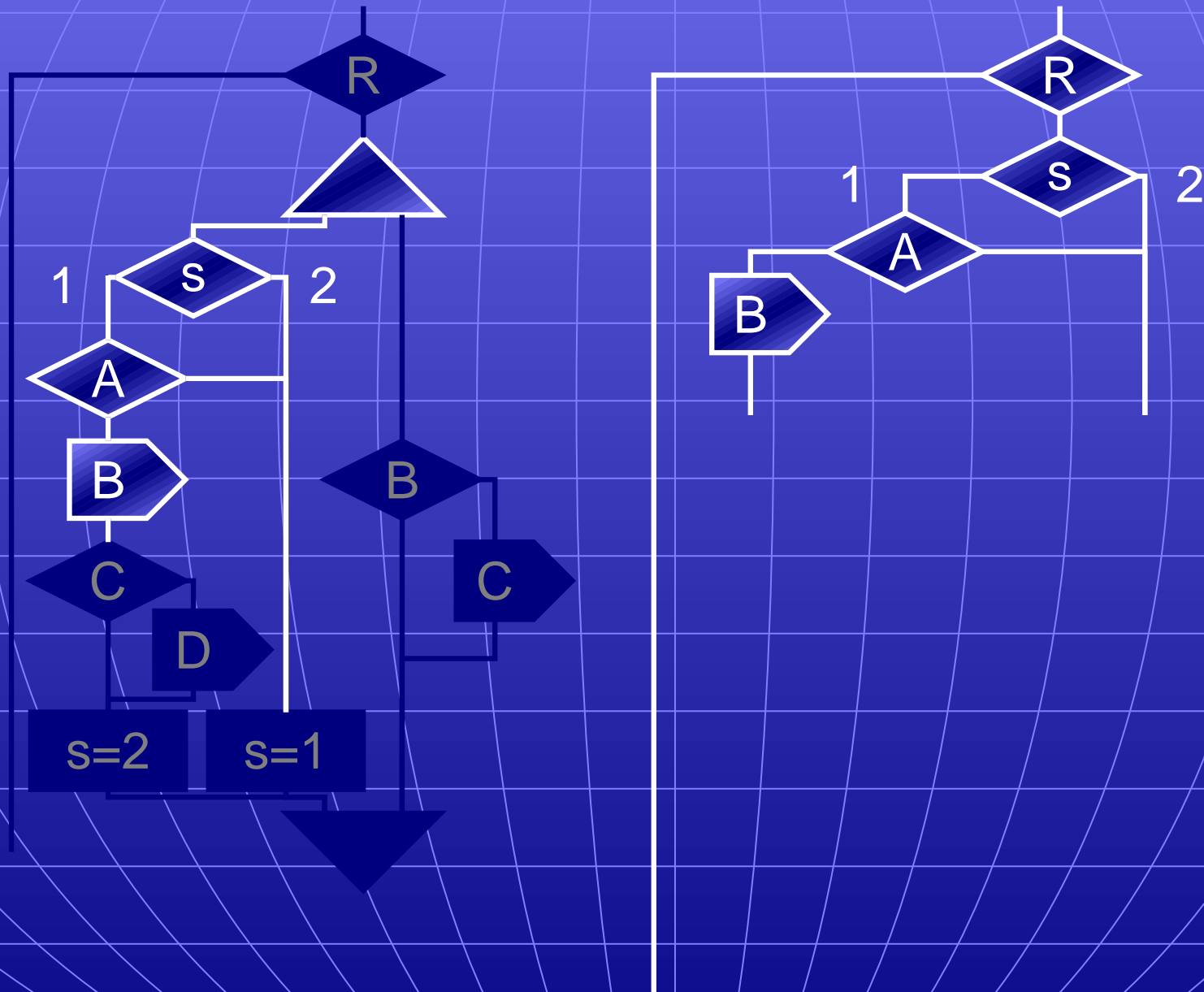
```
every R do
  loop
    await A;
    emit B;
    present C then
      emit D end;
    pause
  end
||
loop
  present B then
    emit C end;
  pause
end
end
```



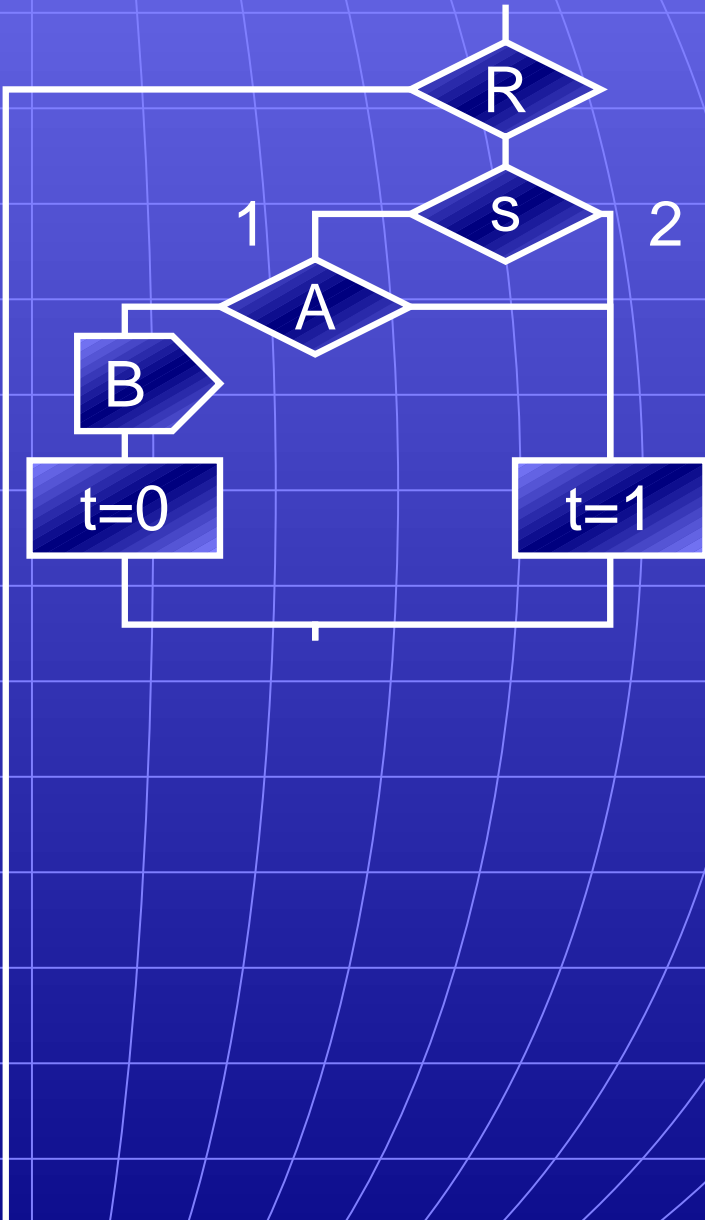
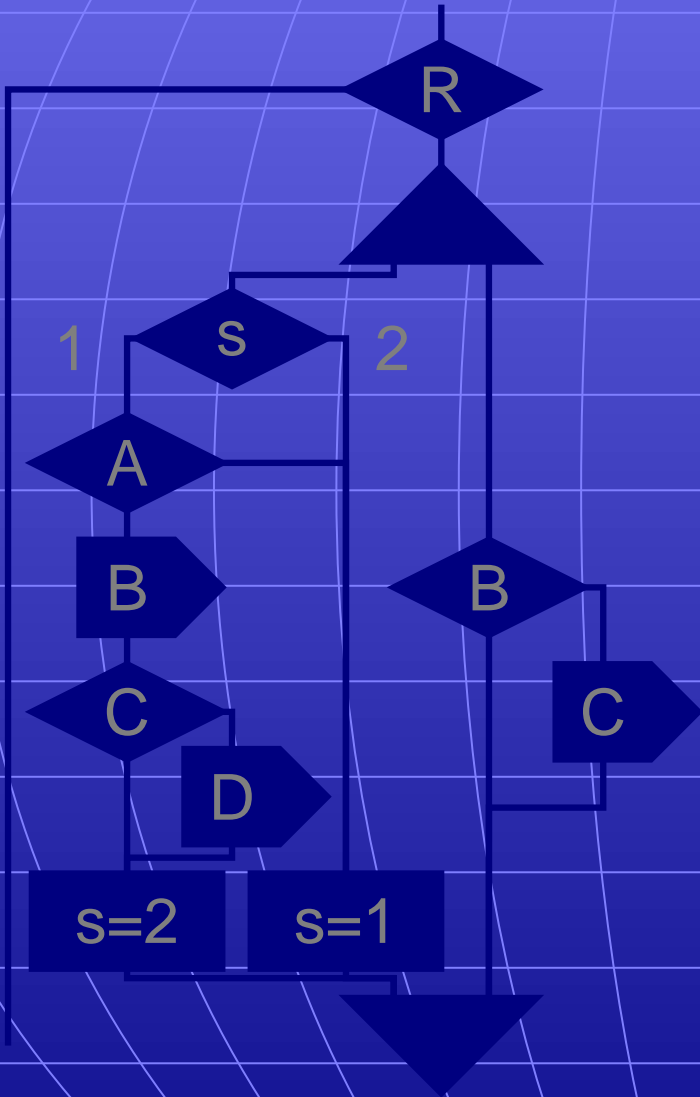
Run First Node



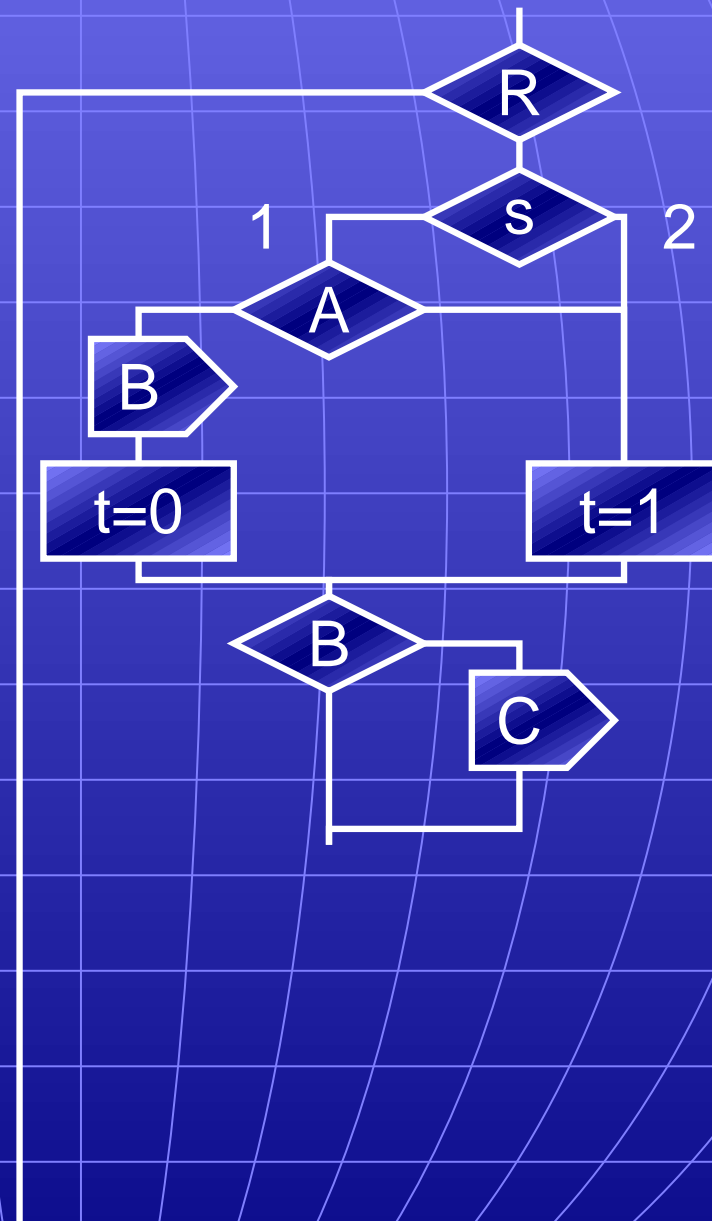
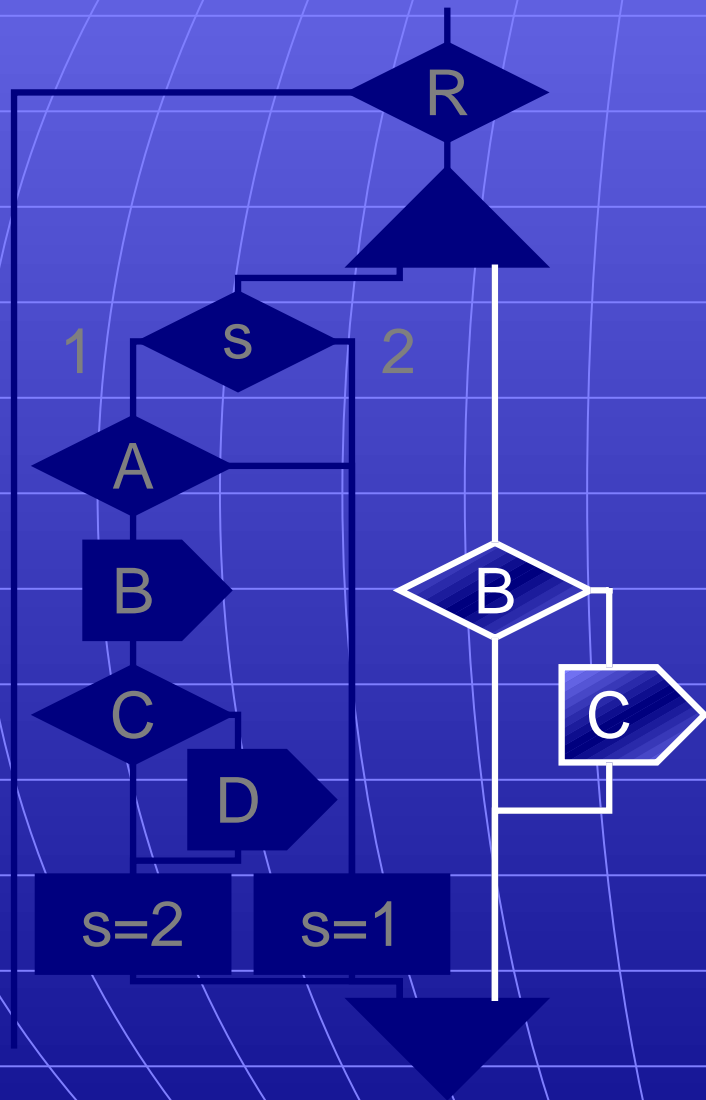
Run First Part of Left Thread



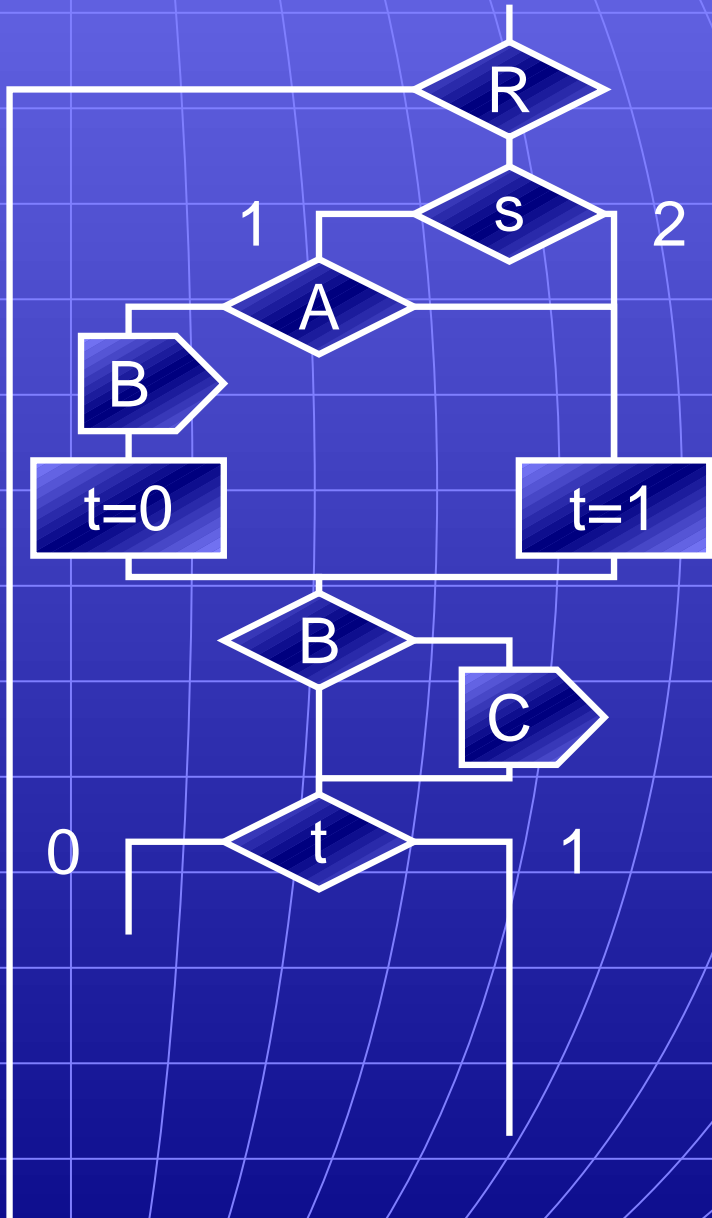
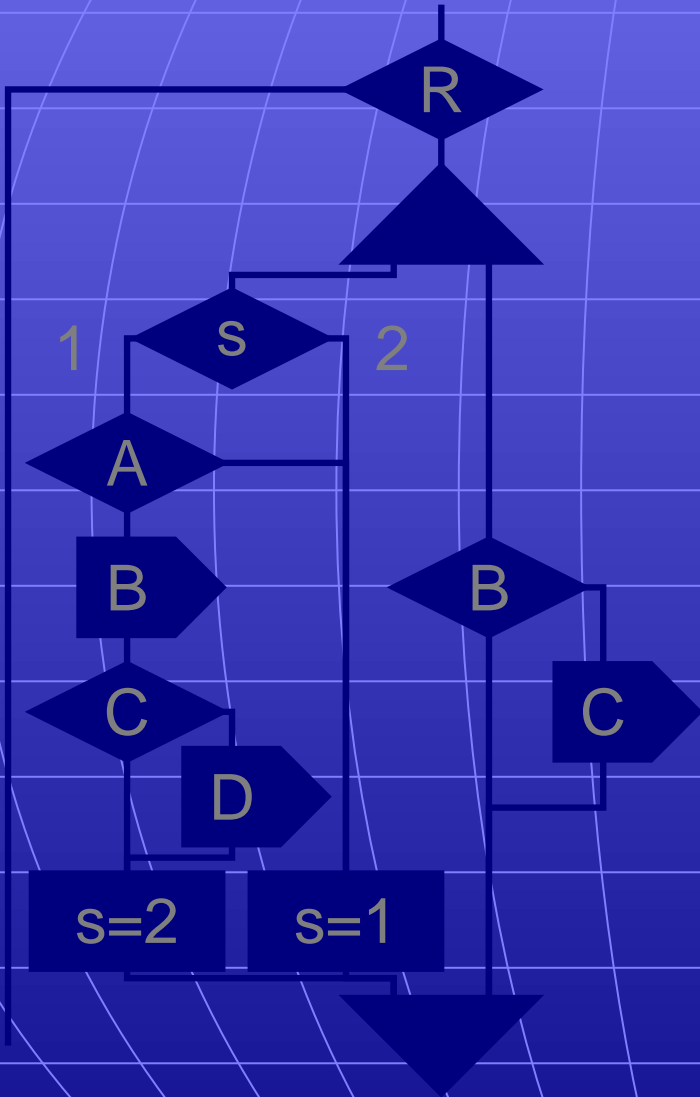
Context Switch



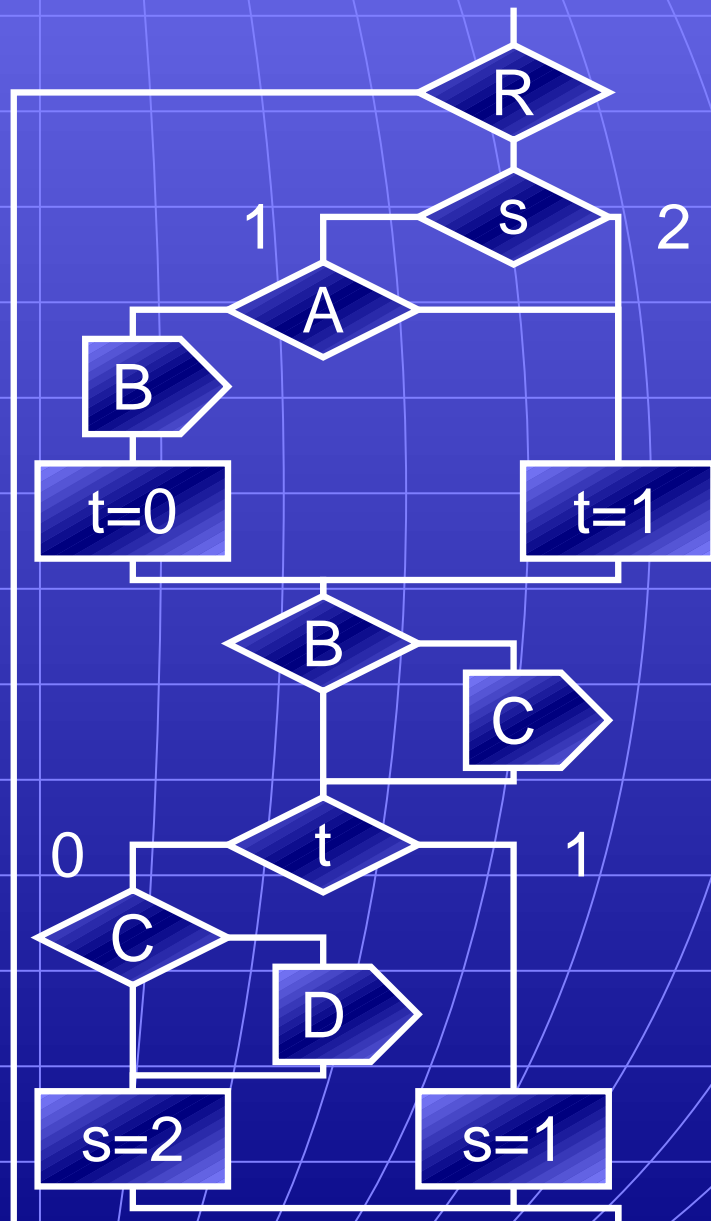
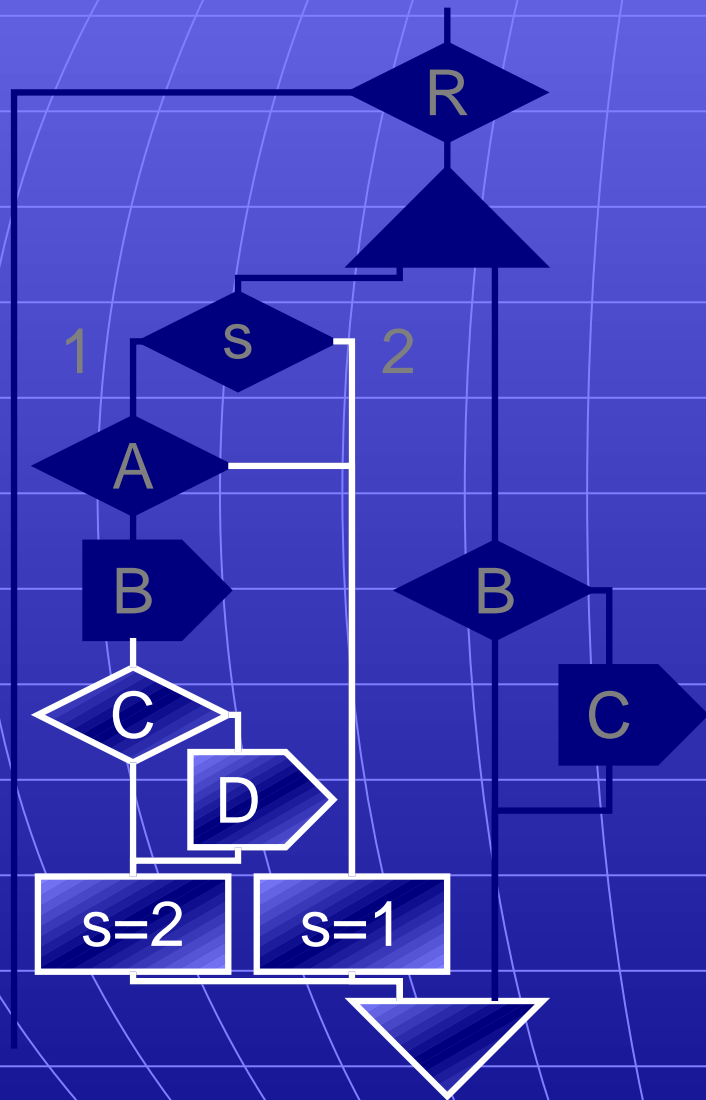
Run Right Thread



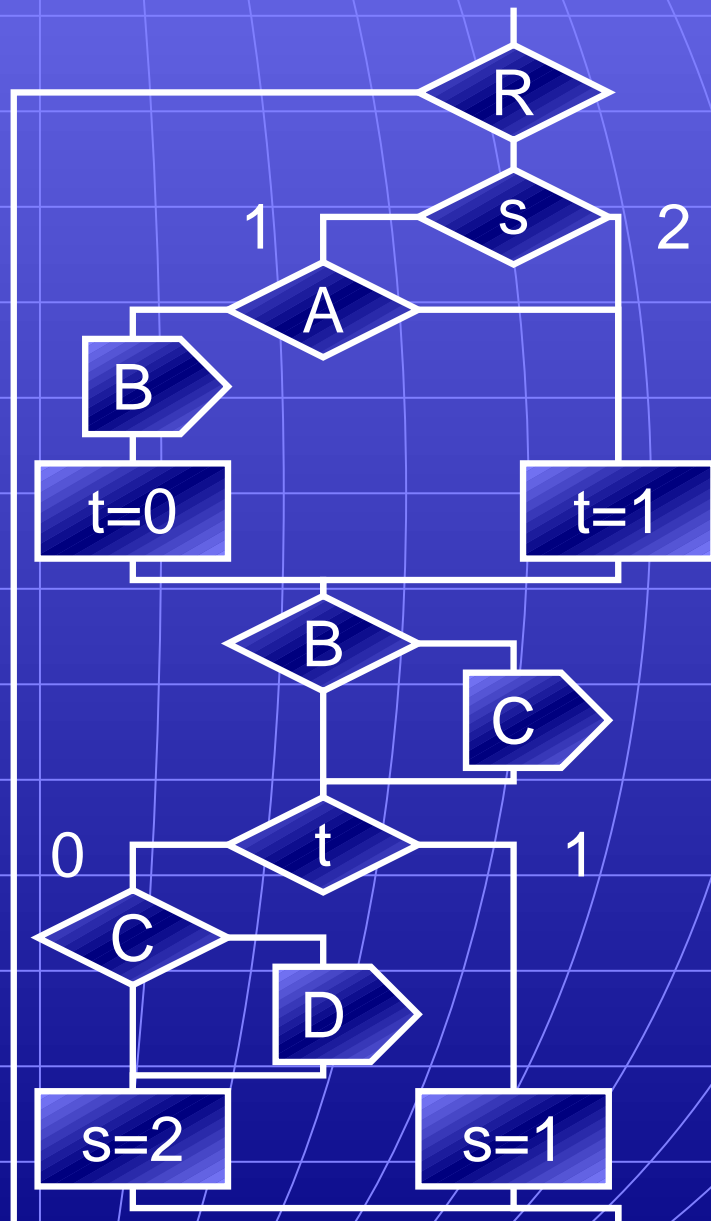
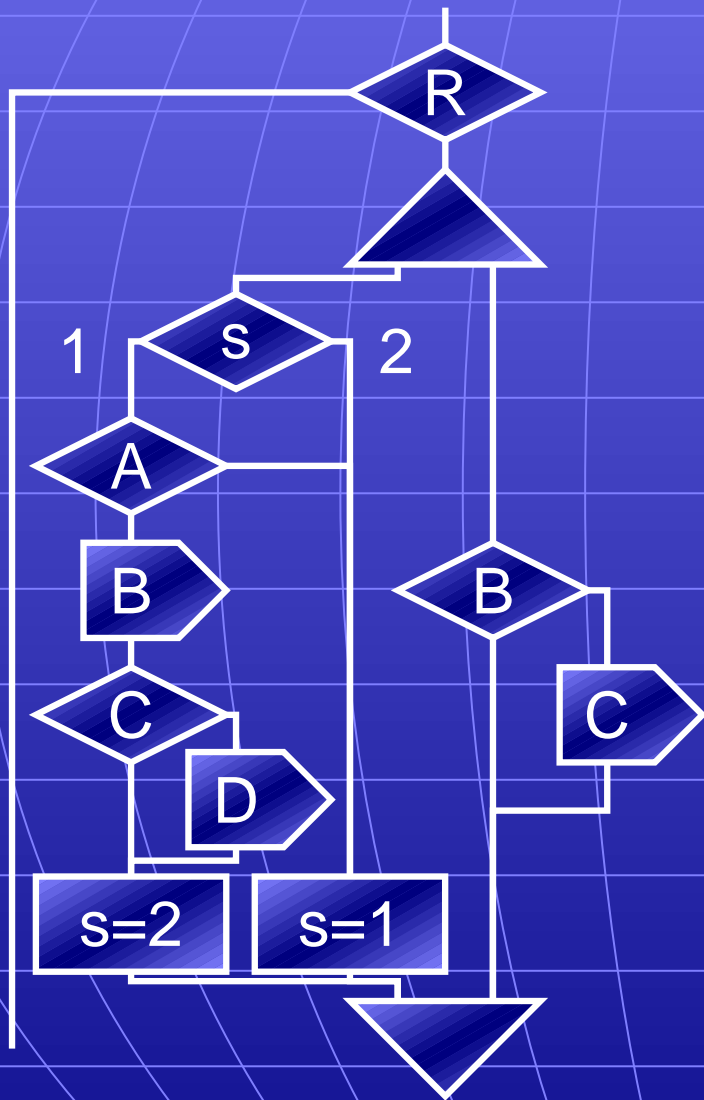
Context Switch



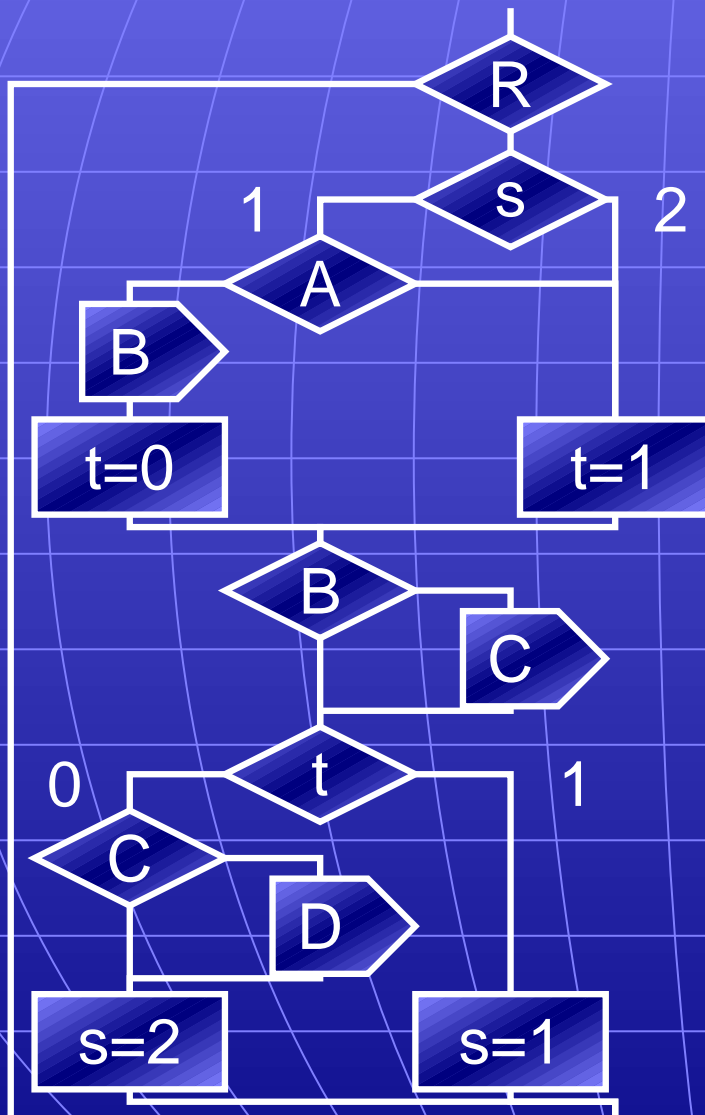
Finish Left Thread



Completed Example

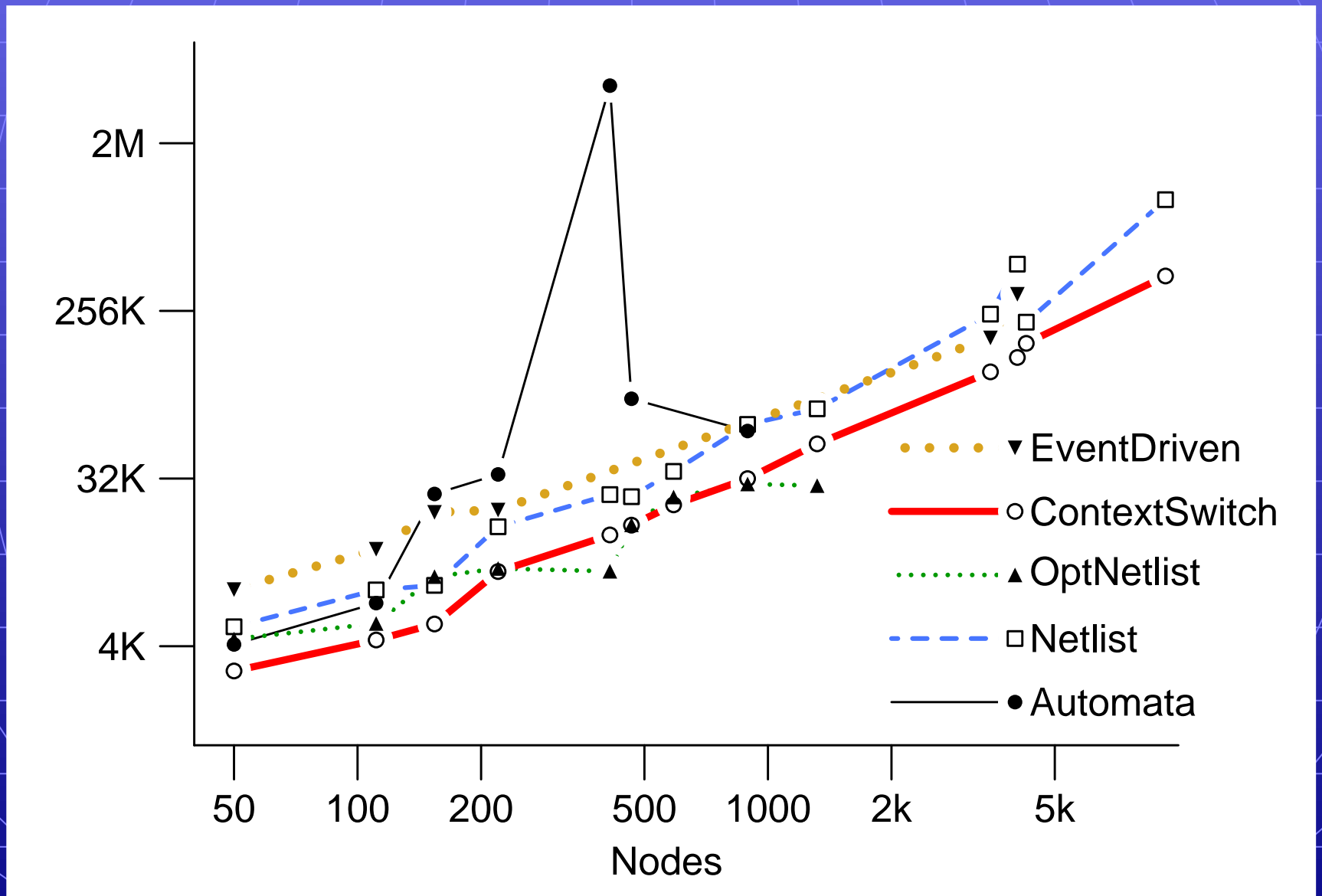


Generated Code

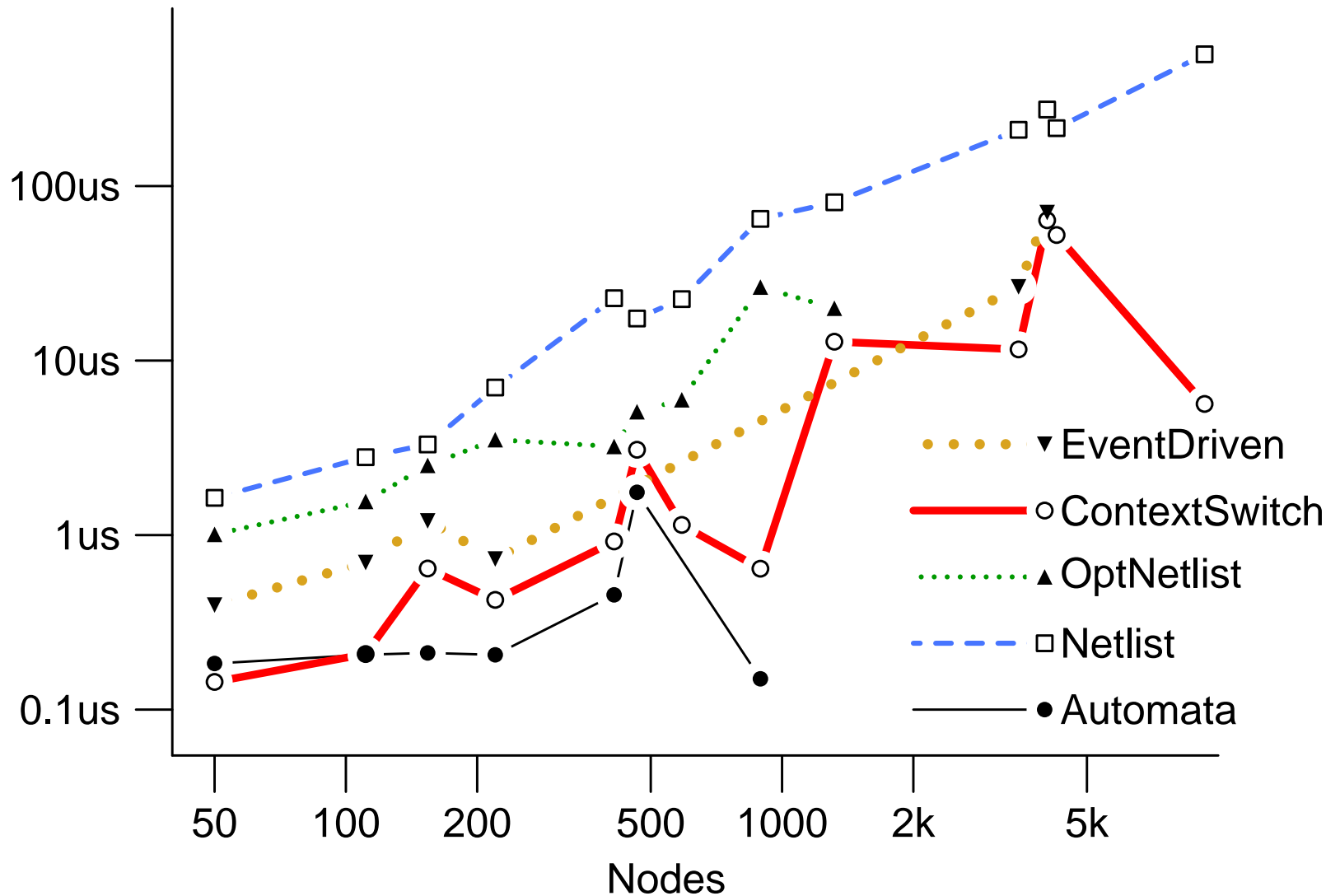


```
if (!R) {
    if (s == 1 && A) {
        B = 1;
        t = 0;
    } else {
        t = 1;
    }
    if (B) C = 1;
    if (t == 0) {
        if (C) D = 1;
        s = 2;
    } else {
        s = 1;
    }
}
```

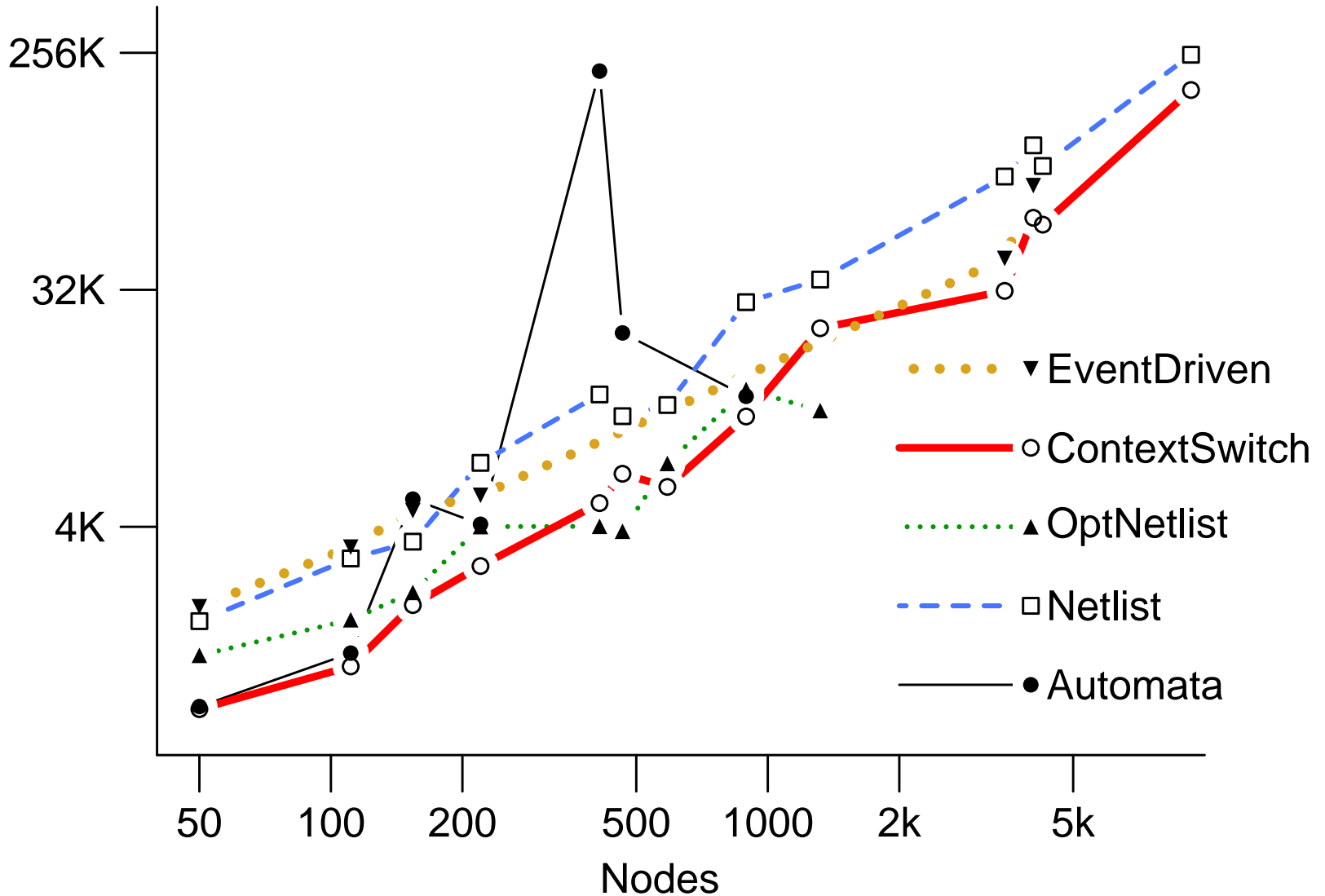
Generated Code Size (UltraSparc-II)



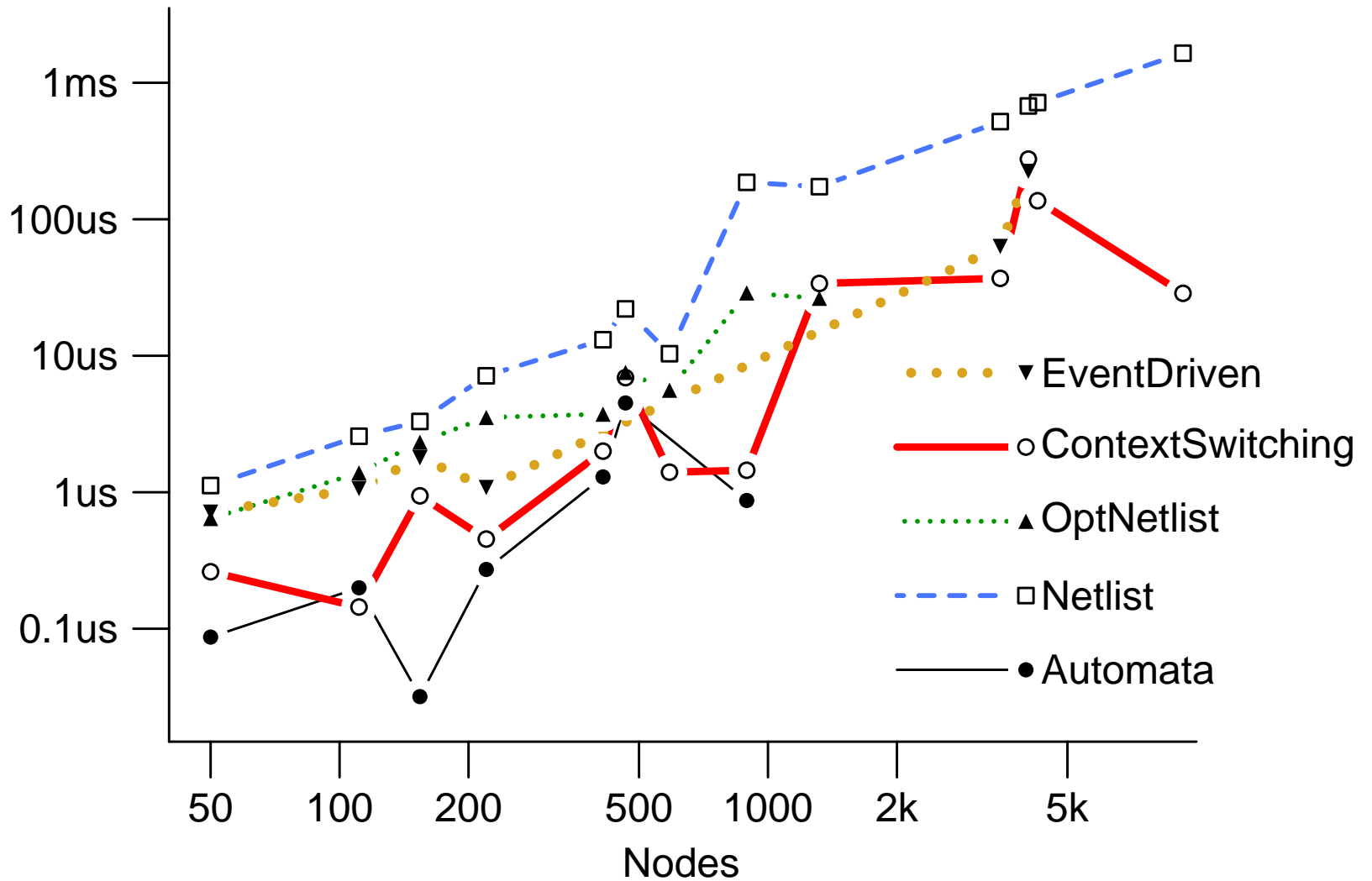
Average Cycle Times (UltraSparc-II)



Generated Code Size (Pentium)



Average Cycle Times (Pentium)



Generating Fast Circuits

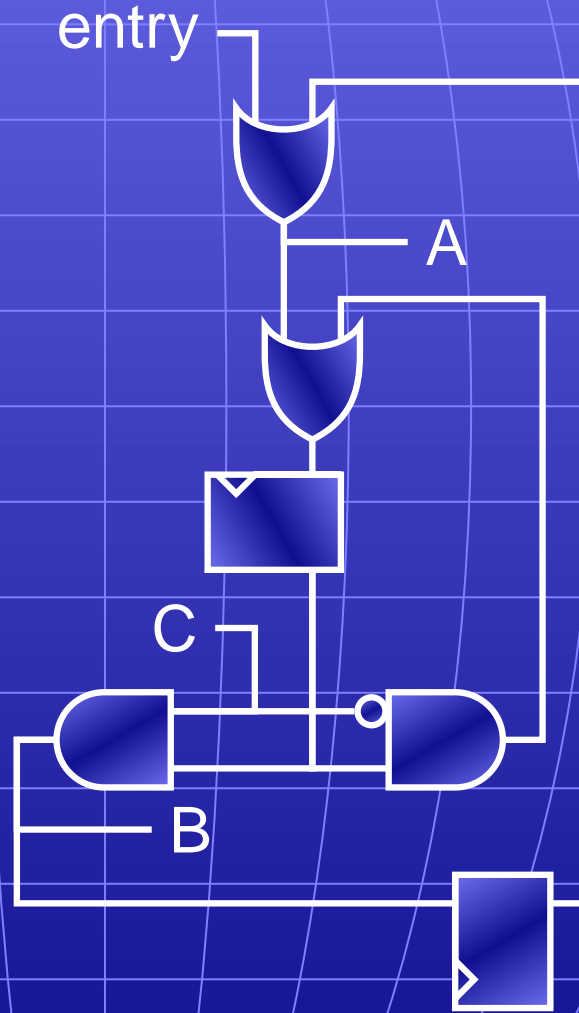
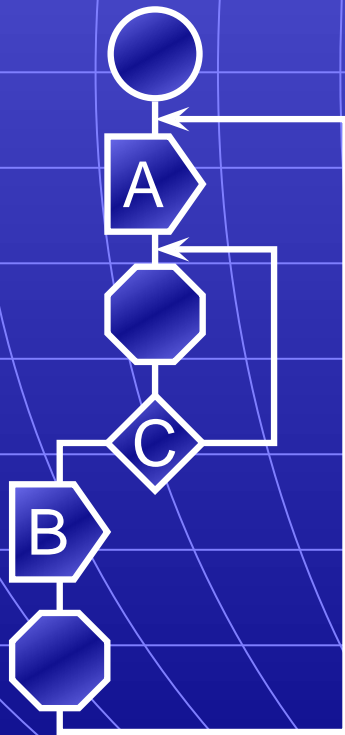
Basic Circuit Generation

loop

emit A; await C;

emit B; pause

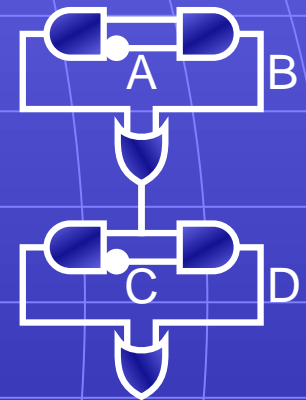
end



Basic Circuit Generation

Berry's technique [1992] works, but is fairly inefficient:

- Many combinational redundancies. E.g.,
`present A then emit B end;`
`present C then emit D end`
produces two redundant OR gates
- Many sequential redundancies



One flop per pause can be very wasteful

Touati, Toma, Sentovich, and Berry [1993–1997] proposed techniques to eliminate many, but requires reachable state space and only works on circuit.

Generating Fast Circuits

Esterel's semantics match hardware. Translation is straightforward.

Nice feature: state space is well-defined and hierarchical (e.g., due to abort and concurrency).

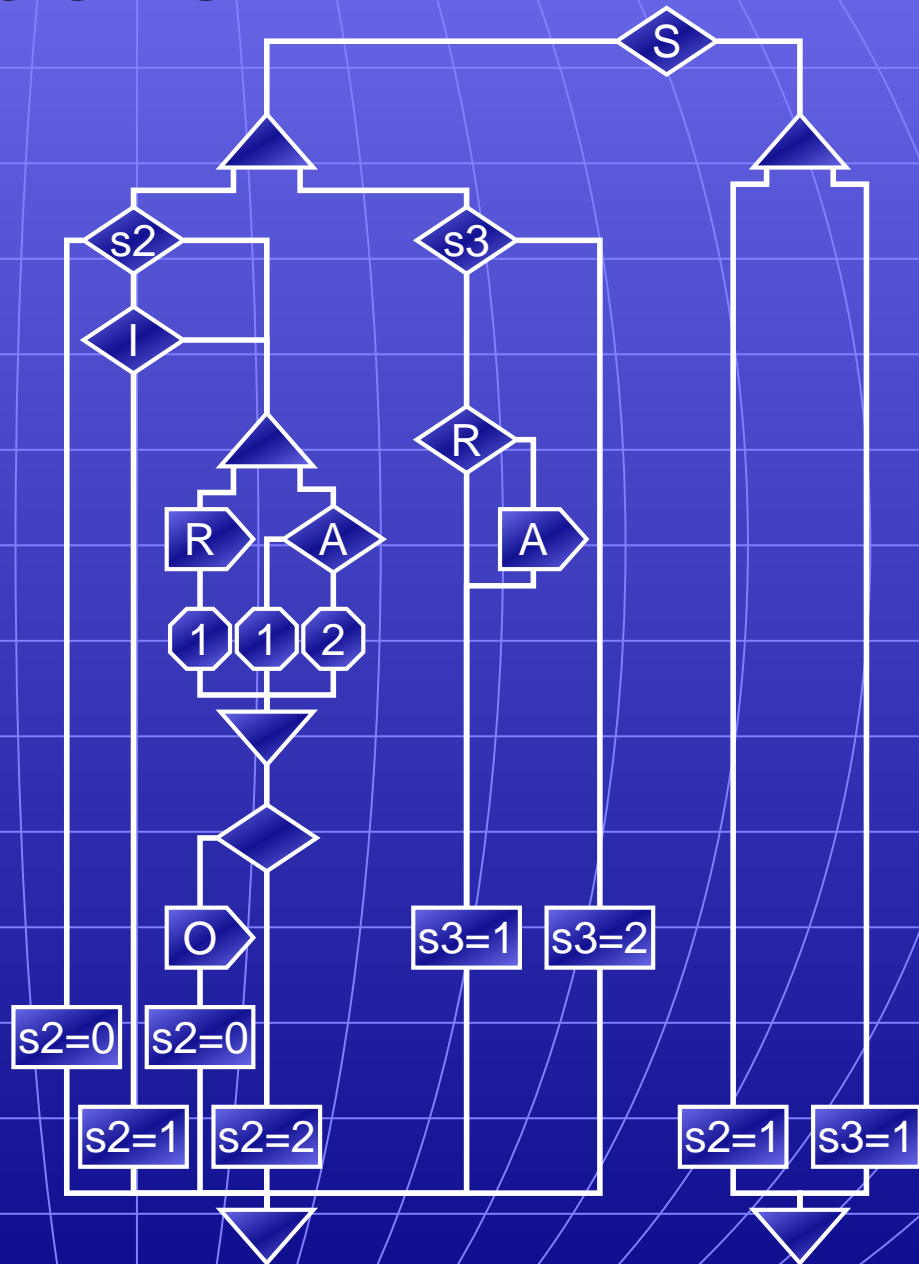
Enables a hierarchical state assignment/synthesis procedure.

Translation to CCFG

```

every S do
  loop
    await I;
    weak abort
    sustain R
    when immediate A;
    emit O
  end
  ||
  loop
    pause; pause;
    present R then
      emit A
    end
  end
end
end

```



A State Assignment Example

```
abort
  [
    await A; await B
    ||
    await C
  ]
when D;
emit E;
pause;
[
  await F
  ||
  await G
]
```

Hierarchical States

abort

[

await A;

await B

||

await C

]

when D;

emit E;

pause;

[

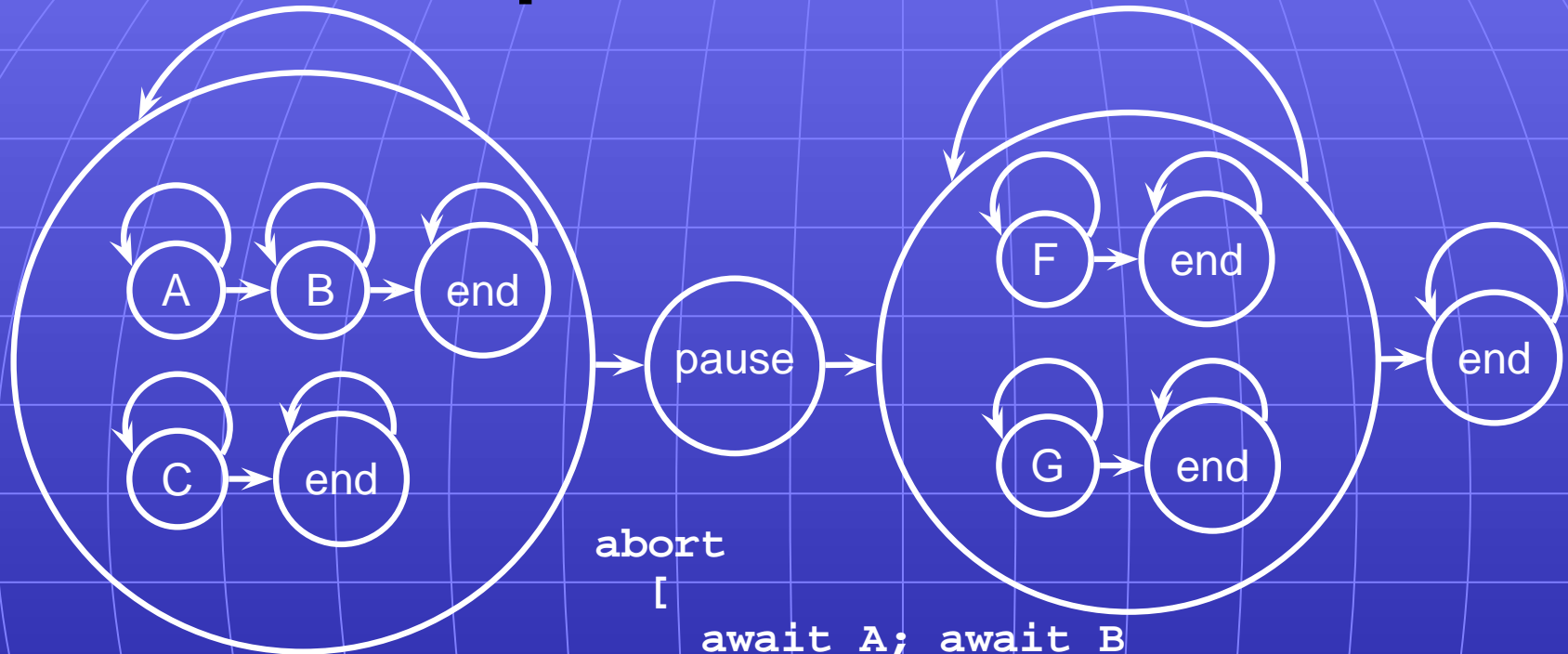
await F

||

await G

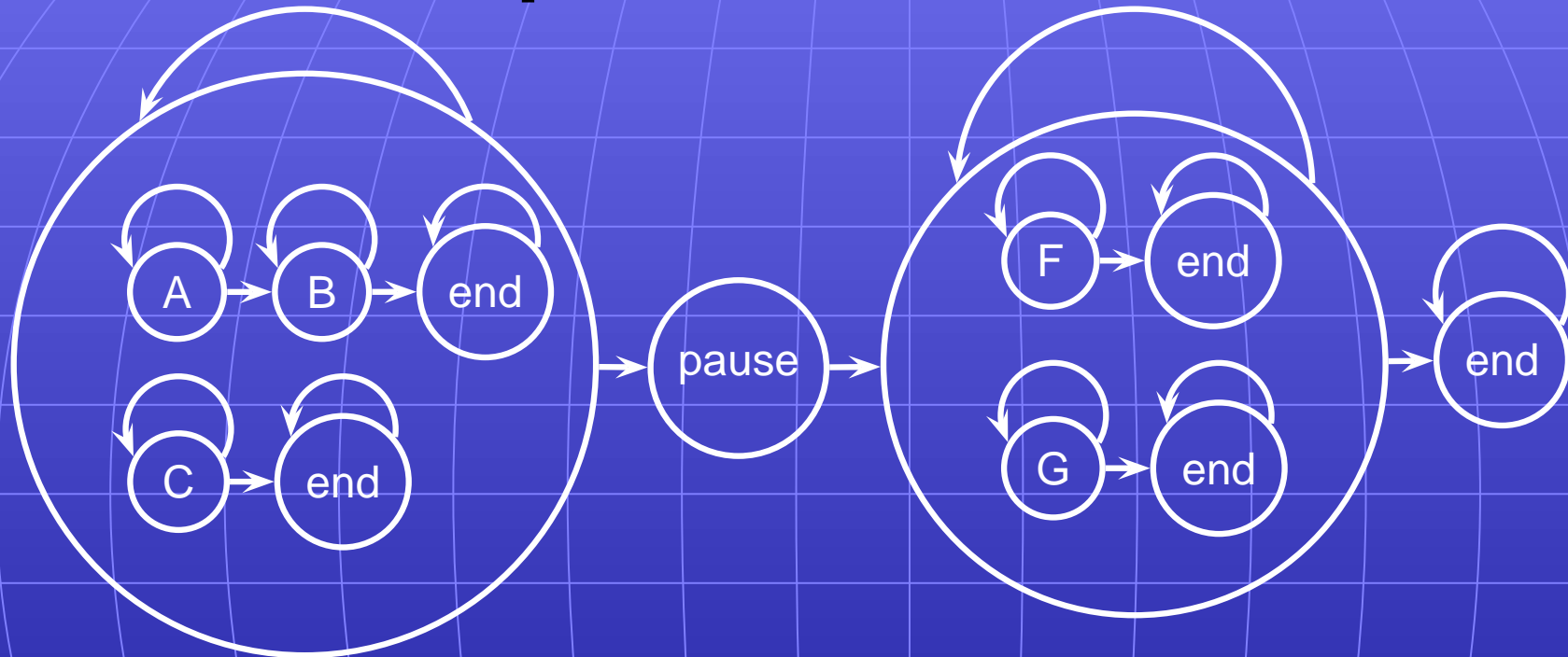
]

Five Simple FSMs



```
abort
[
  await A; await B
  ||
  await C
]
when D;
emit E;
pause;
[
  await F
  ||
  await G
]
```

Five Simple FSMs

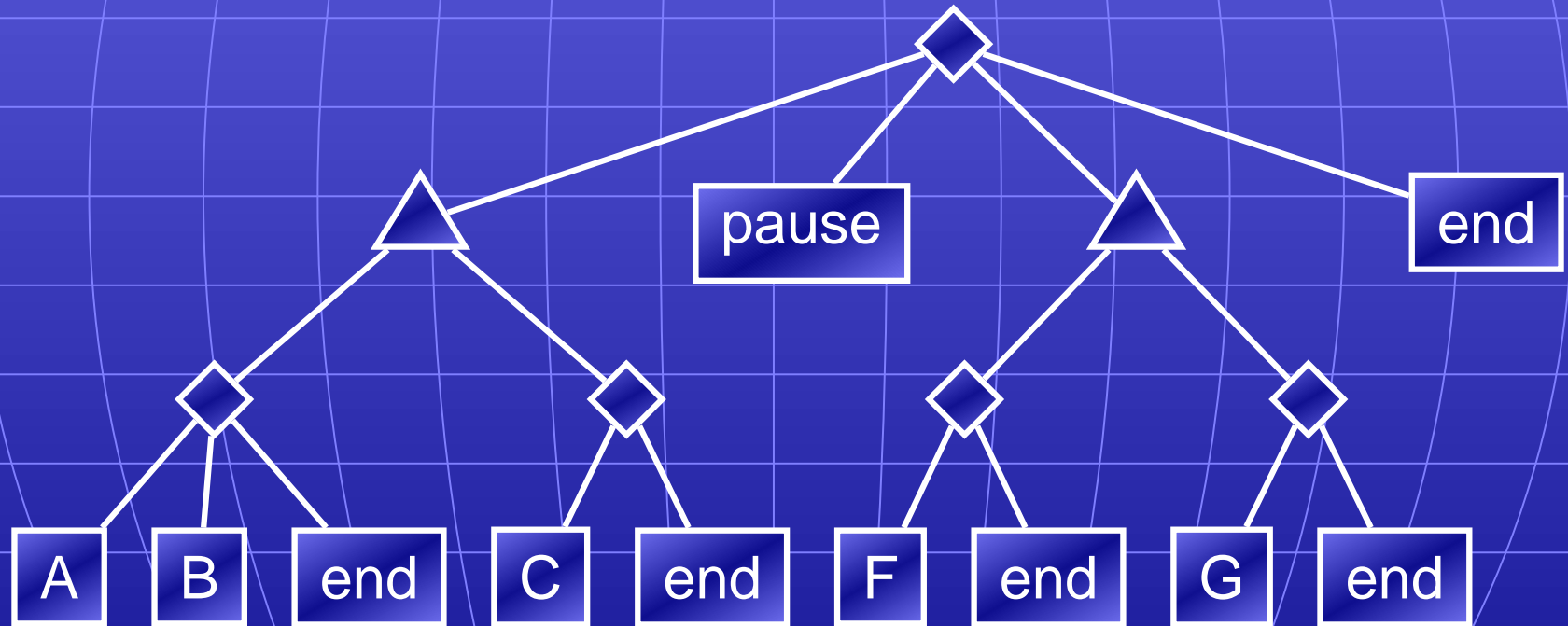


Obvious questions:

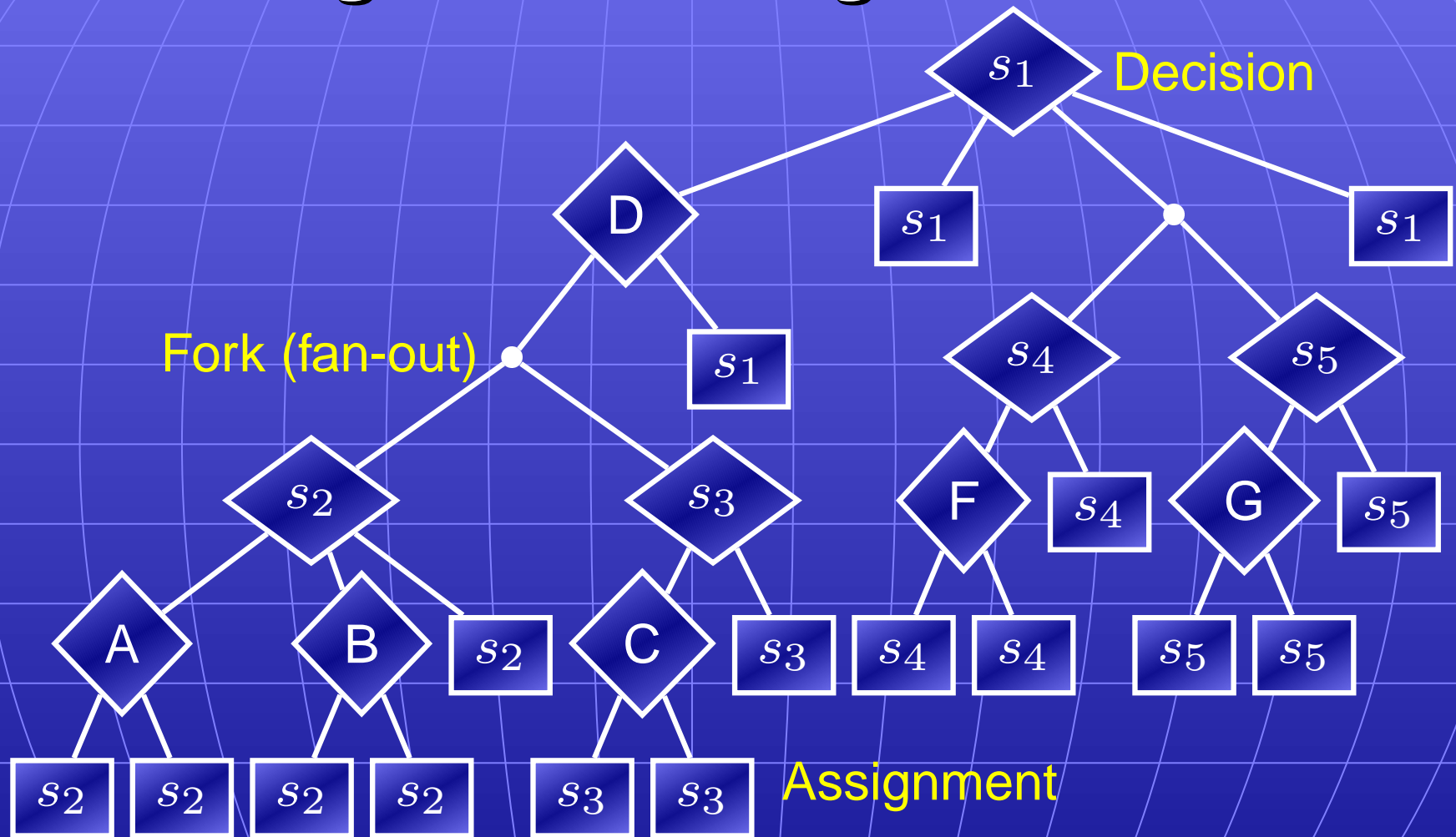
- How should each state machine be encoded?
- Should state be shared between the AB/F and C/G machines?

General Problem Statement

States in an Esterel program are an arbitrary tree of sequential and parallel state machines.



Choosing an Encoding



- How should s_1, \dots, s_4 be encoded?
- Should s_2 or s_3 be shared with s_4 or s_5 ?

Choosing a Good Encoding

Goal: The smallest circuit that meets a timing constraint

1. Start with largest, fastest circuit (one-hot, no sharing)
2. Estimate the slack at each state decision point by estimating how much the delay could be increased at that point while still meeting the timing requirement
3. Attempt to share states at the lowest decision point with the largest slack or reencode the widest-fanout decision point with sufficient slack.
4. Repeat steps 2–3 until no further gain possible

Summary

Introduction to Esterel and Existing Compilers

Synchronous, Concurrent, Textual Language

Automata, Netlist, and Control-based compilers

My Software Compiler [DAC 2000, TransCAD 2002]

Translate to Concurrent CFG, schedule, then
synthesize Sequential CFG

My Hardware Compiler: [SLAP 2002, IWLS 2002]

Translate CCFG to Program Dependence Graph

Trivially translate PDG to circuitry

State assignment problem: heuristic algorithm