Compiling Esterel into Static Discrete-Event Code

Stephen A. Edwards
Columbia University
Computer Science Department
New York, USA
sedwards@cs.columbia.edu

Vimal Kapadia and Michael Halas
IBM
Poughkeepsie
NY USA
vimal@kapadia.us michael@halas.us

Presented by

Michael Halas

SLAP 2004

CEC

- Compiles Esterel into very efficient C code
- Minimizes runtime overhead
  - Compile time
  - Runtime

An Example

Modeling a shared resource

Input I,S ;
Output O,Q;

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

Takes **I**, and passes to group two through **R**

1

```
await I;
weak abort
   sustain R
when immediate A;
emit O
||
loop
   pause; pause;
   present R then
      emit A
   end present
end loop
||
loop
   present R then
      pause; emit Q
   else
      pause
   end present
end loop
end every
```

Responds to **R** with **A**

2

Makes **Q** delayed version of **R**

3

Takes **I**, and passes to group two through **R**

1

```
await I;
weak abort
   sustain R
when immediate A;
emit O
||
```

Responds to **R** with **A**

2

```
loop
   pause; pause;
   present R then
      emit A
   end present
end loop
||
```

Makes **Q** delayed version of **R**

3

```
loop
   present R then
      pause; emit Q
   else
      pause
   end present
end loop
end every
```

Takes **I**, and passes to group two through **R**

1

```
await I;
weak abort
   sustain R
when immediate A;
emit O
||
loop
   pause; pause;
   present R then
      emit A
   end present
end loop
||
loop
   present R then
      pause; emit Q
   else
      pause
   end present
end loop
end every
```

2

Responds to **R** with **A**

3

Makes **Q** delayed version of **R**

1. Takes **I**, and passes to group two through **R**
2. Responds to **R** with **A**
3. Makes **Q** delayed version of **R**

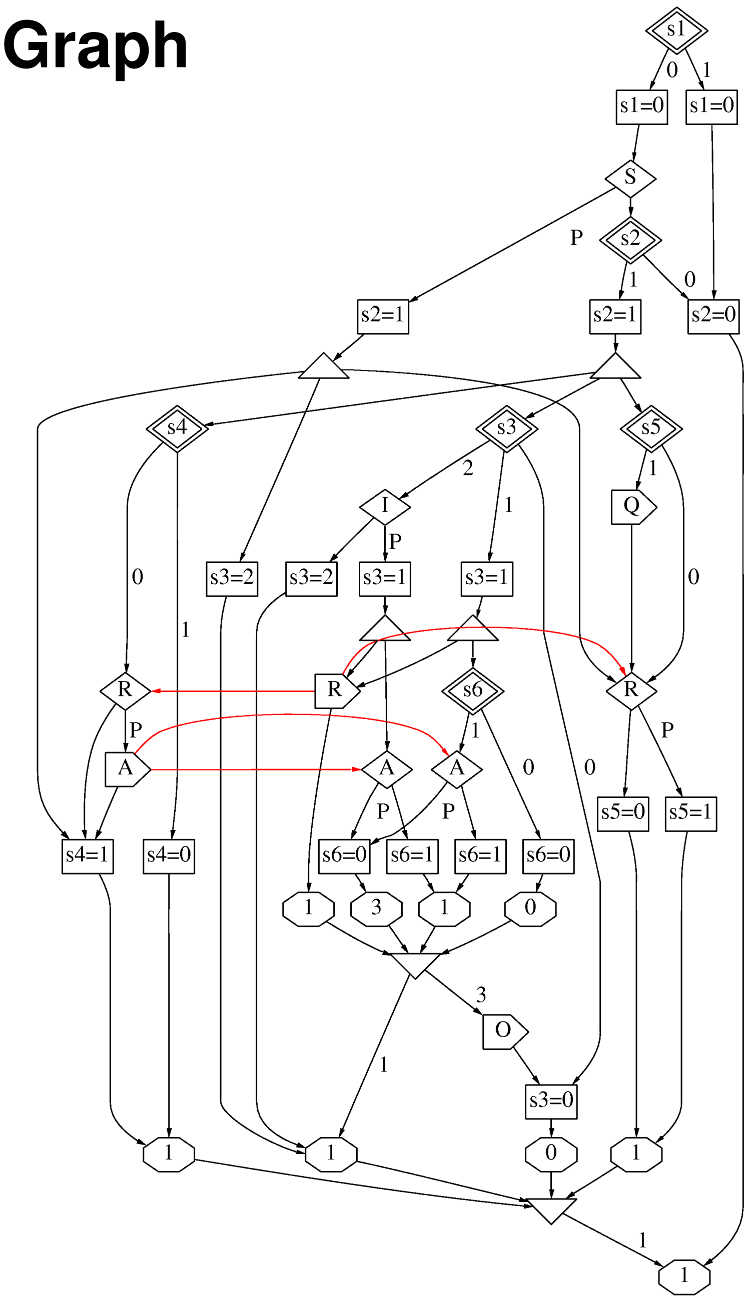
```
await I;
weak abort
    sustain R
when immediate A;
emit O
||
loop
    pause; pause;
    present R then
        emit A
    end present
end loop
||
loop
    present R then
        pause; emit Q
    else
        pause
    end present
end loop
end every
```

The GRC Representation

Developed by Potop-Butucaru

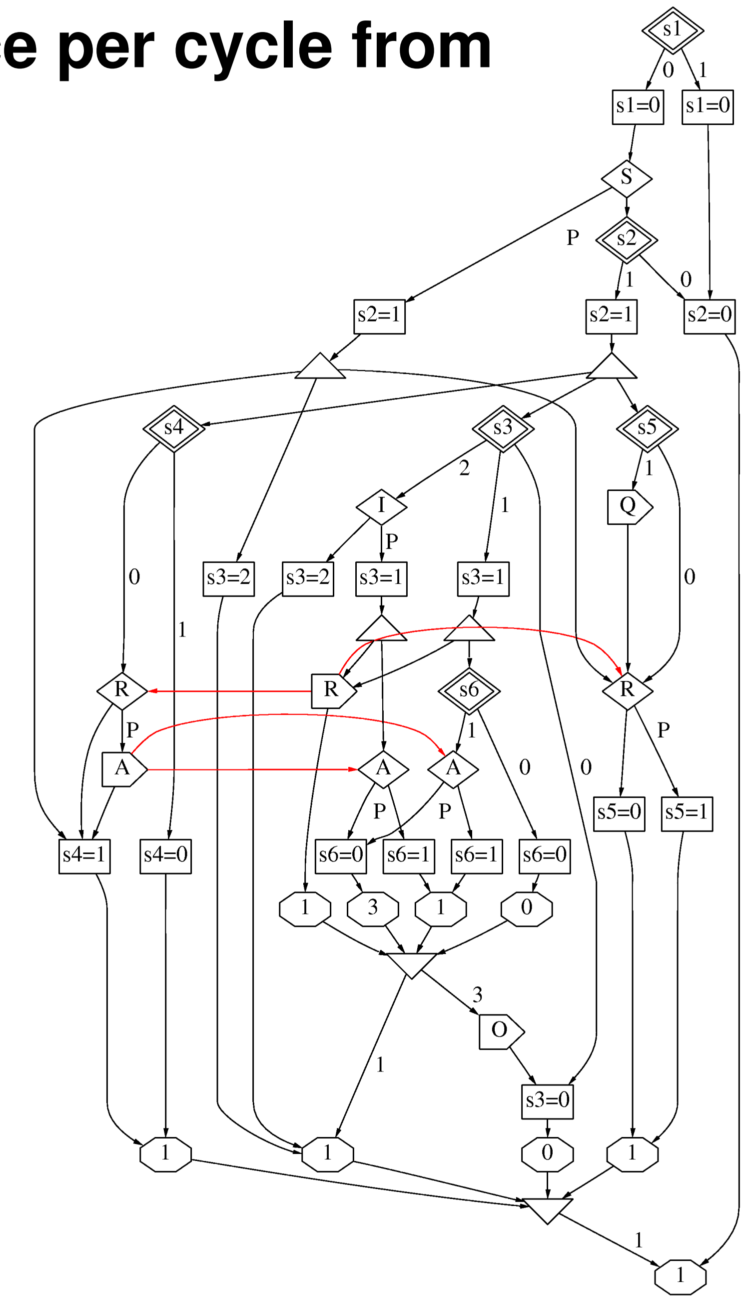
Control Flow Graph

```
Input I,S ;
Output O,Q;
Signal R,A in

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

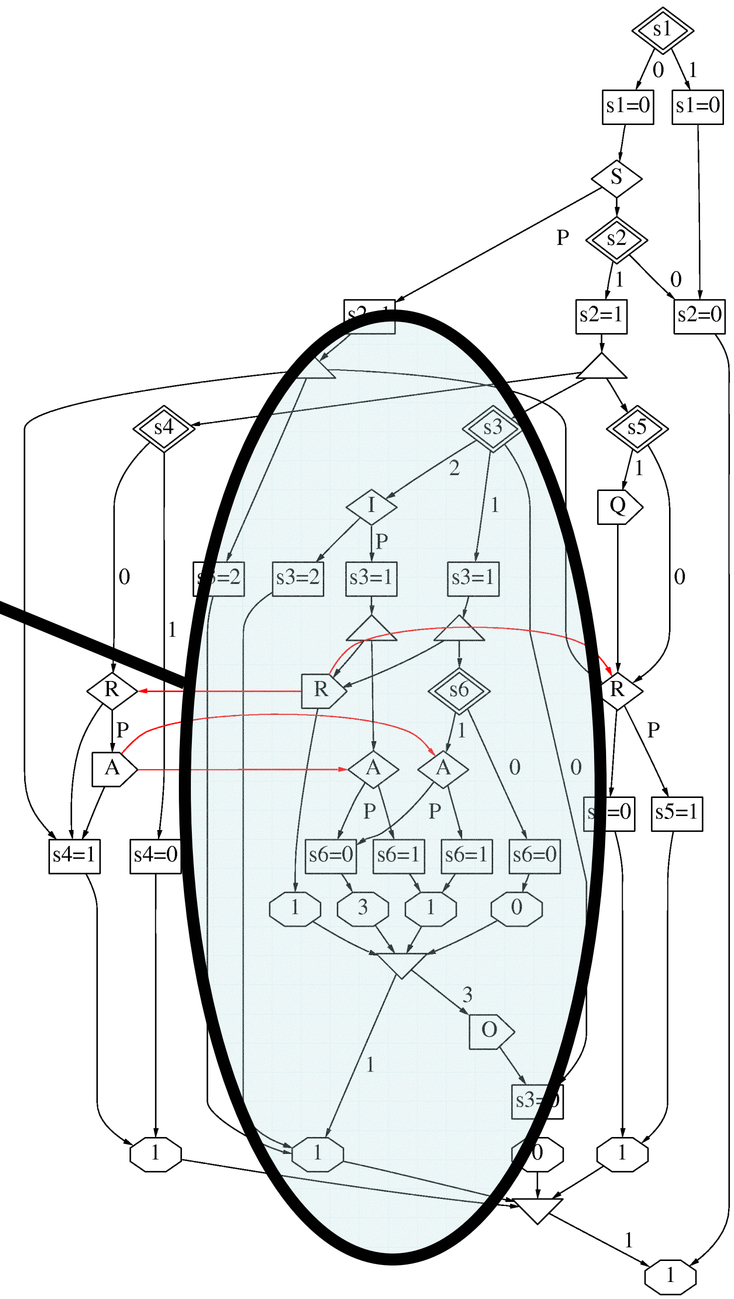


Executes once per cycle from entry to exit

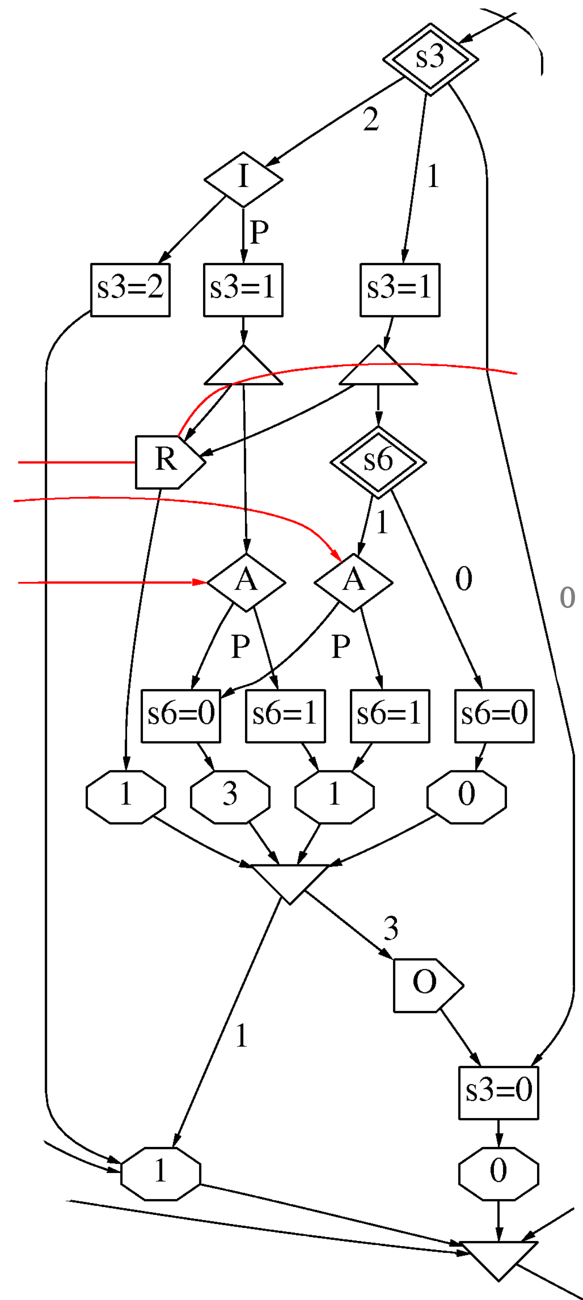
```
Input I,S ;
Output O,Q;
Signal R,A in

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



```
Input I,S ;
Output O,Q;
Signal R,A in

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



await I;
weak abort
sustain R
when immediate A;
emit O



Clustering

1. Group the GRC nodes into clusters that can run without interruption
2. Assign levels – Partial Ordering

   Levels execute in order

   Clusters within the same level can execute in any order

Clustering

1. Group the GRC nodes into clusters that can run without interruption

2. Assign levels – Partial Ordering

   Levels execute in order – Compile Time

   Clusters within the same level can execute in any order

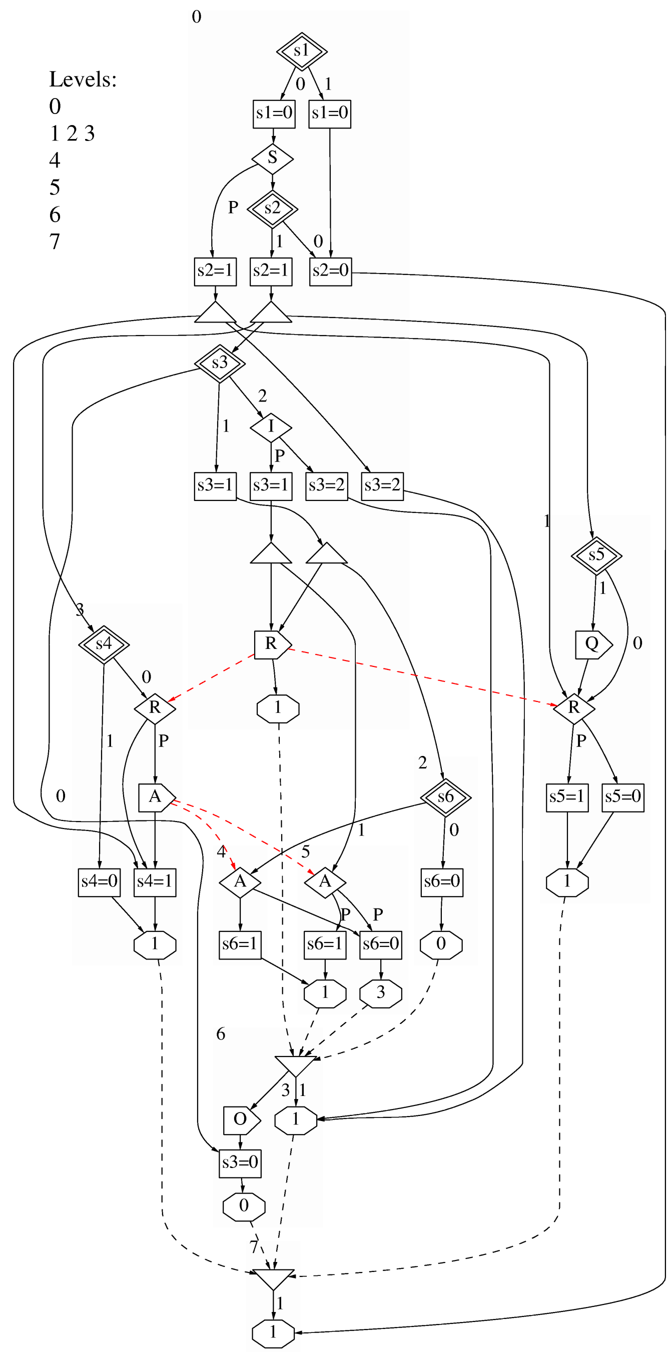
Clustering

1. Group the GRC nodes into clusters that can run without interruption
2. Assign levels – Partial Ordering

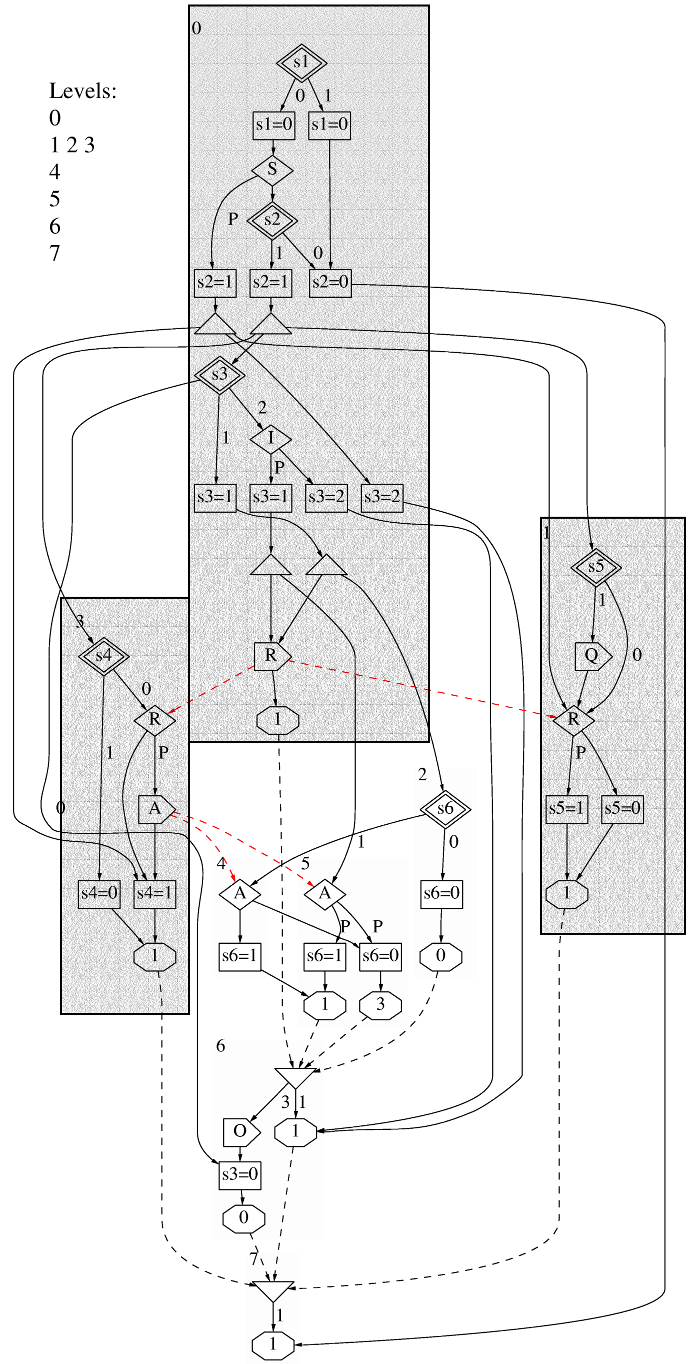
   Levels execute in order – Compile Time

   Clusters within the same level can execute in any order - Runtime

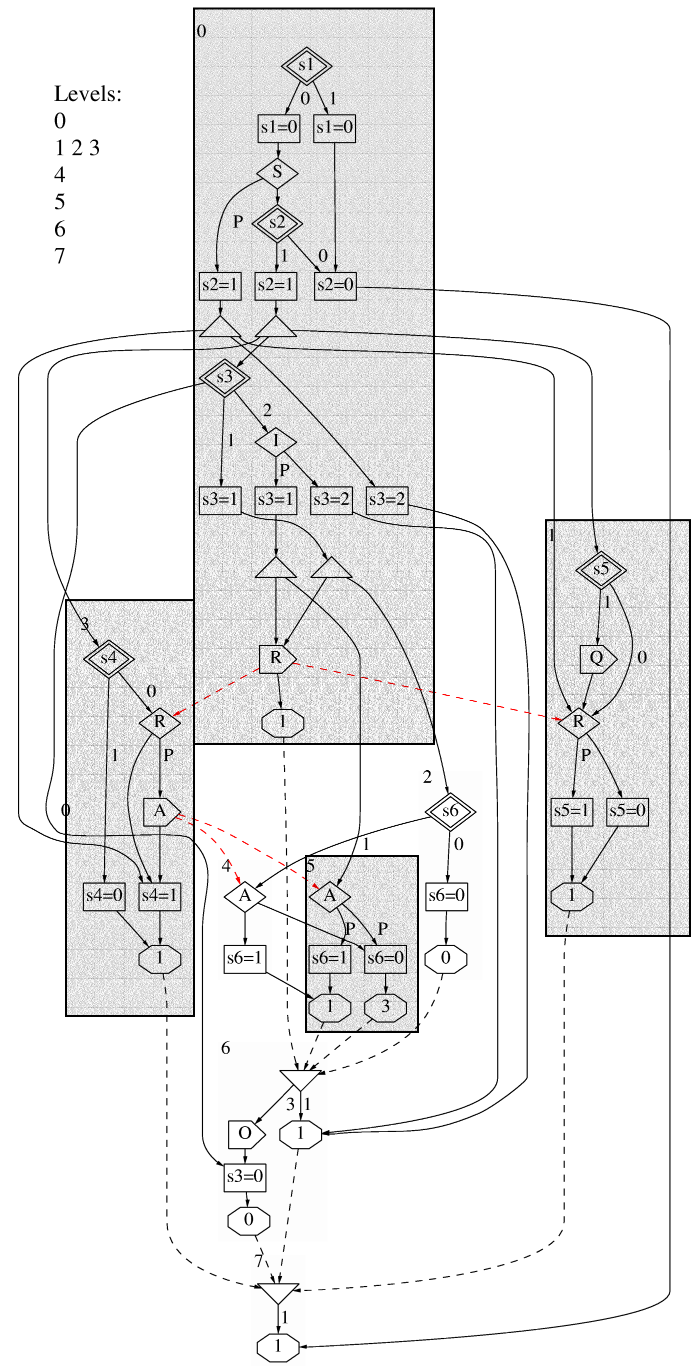
```
every S do
   await I;
   weak abort
      sustain R
   when immediate A;
   emit O
||
loop
   pause; pause;
   present R then
      emit A
   end present
end loop
||
loop
   present R then
      pause; emit Q
   else
      pause
   end present
end loop
end every
```



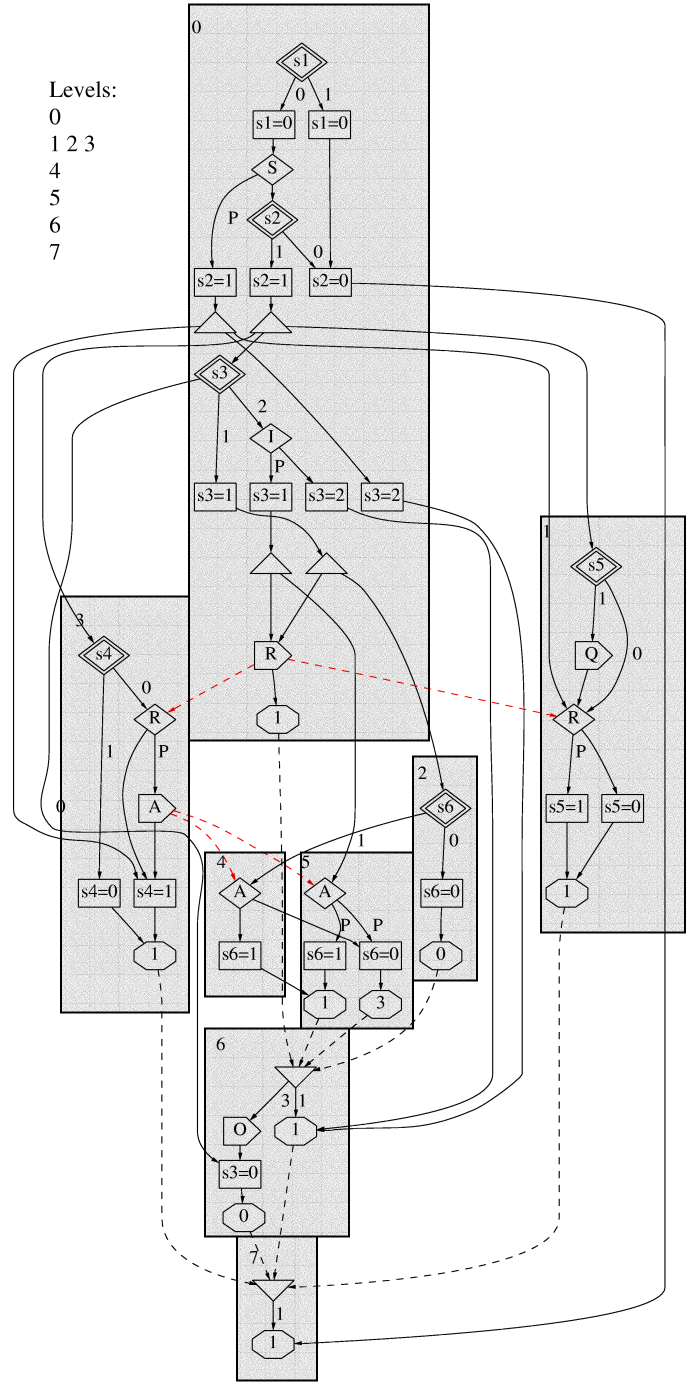
```
every S do
    await I;
    weak abort
        sustain R
    when immediate A;
    emit O
||
loop
    pause; pause;
    present R then
        emit A
    end present
end loop
||
loop
    present R then
        pause; emit Q
    else
        pause
    end present
end loop
end every
```



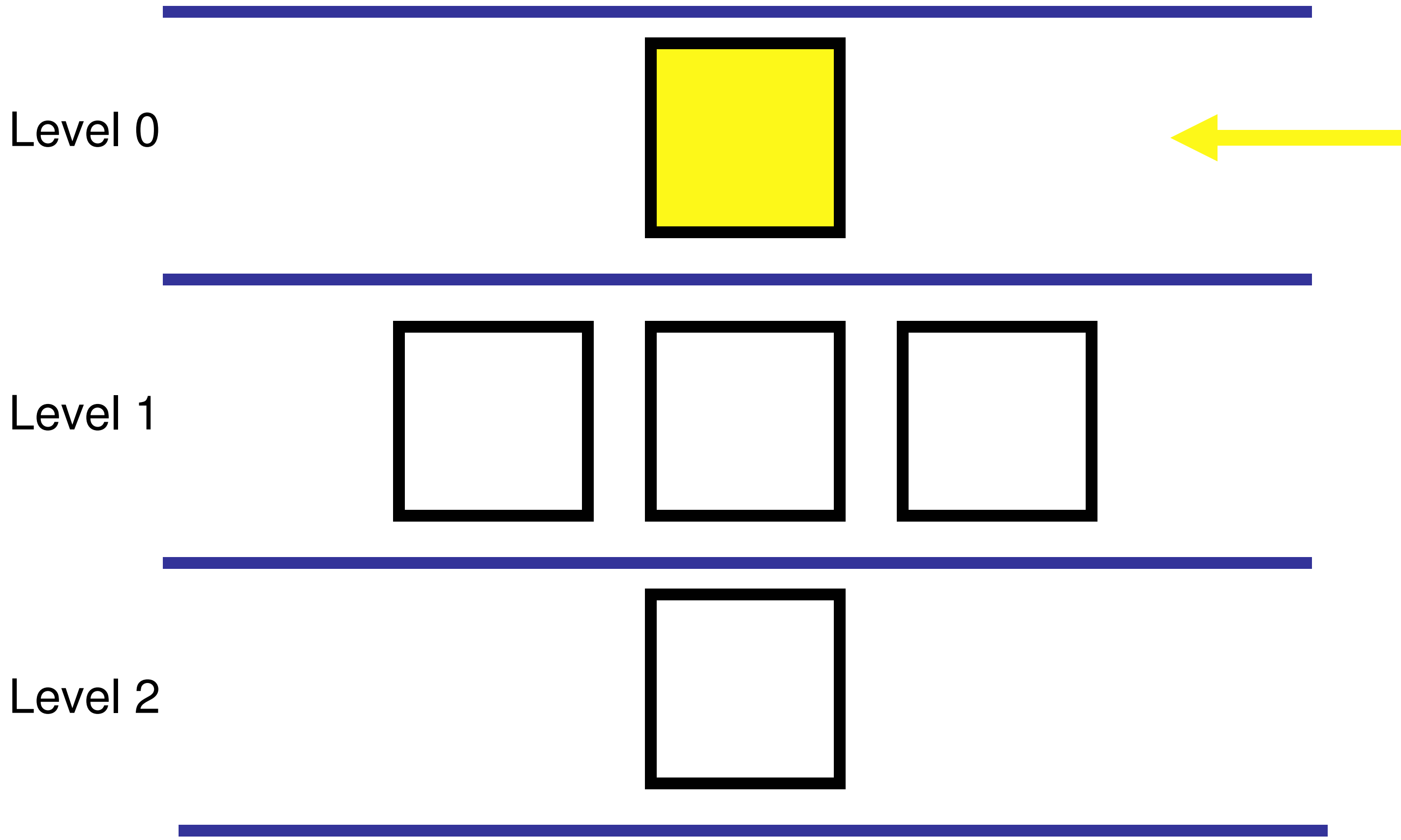
```
every S do
    await I;
    weak abort
        sustain R
    when immediate A;
    emit O
||
loop
    pause; pause;
    present R then
        emit A
    end present
end loop
||
loop
    present R then
        pause; emit Q
    else
        pause
    end present
end loop
end every
```



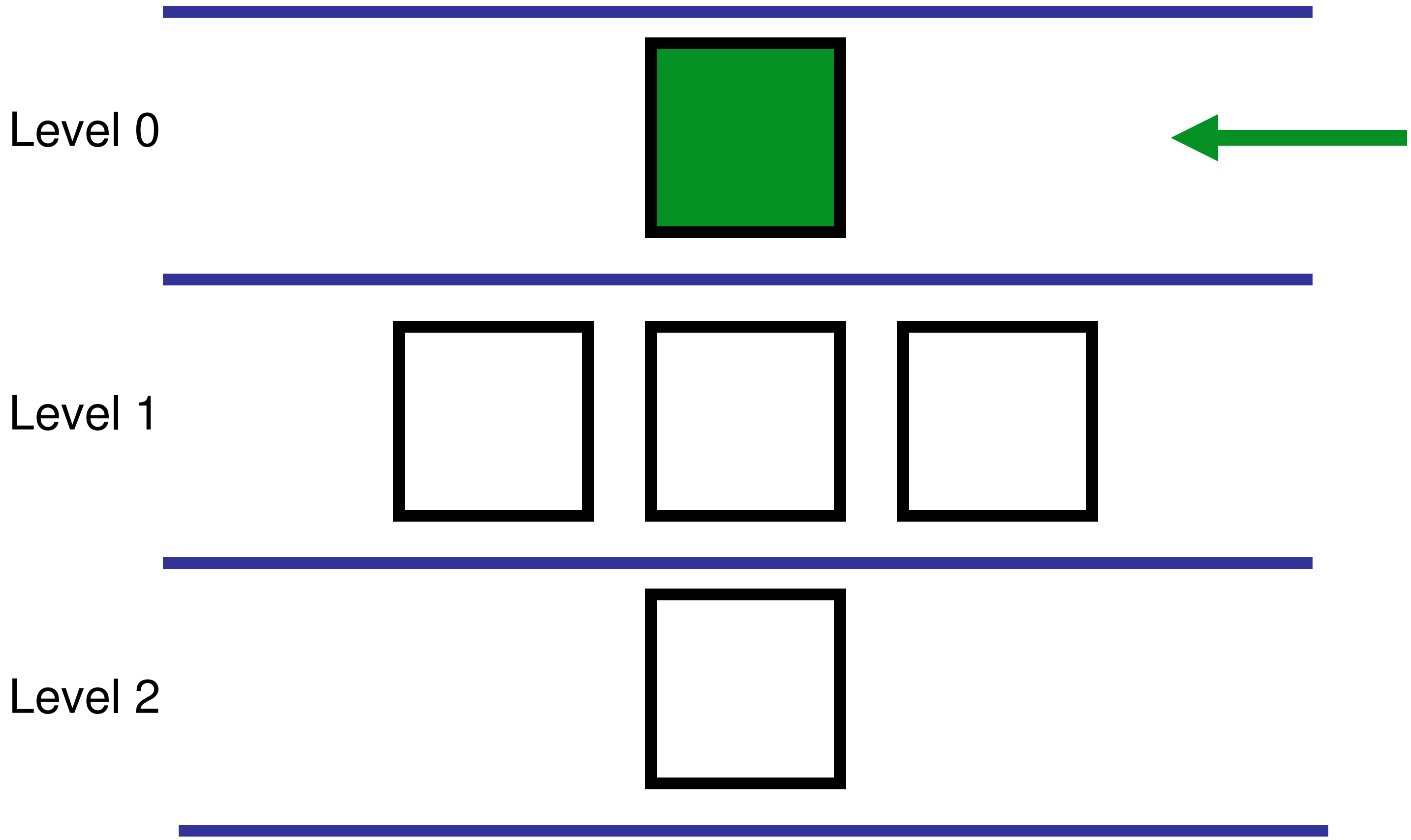
```
every S do
    await I;
    weak abort
        sustain R
    when immediate A;
    emit O
||
loop
    pause; pause;
    present R then
        emit A
    end present
end loop
||
loop
    present R then
        pause; emit Q
    else
        pause
    end present
end loop
end every
```



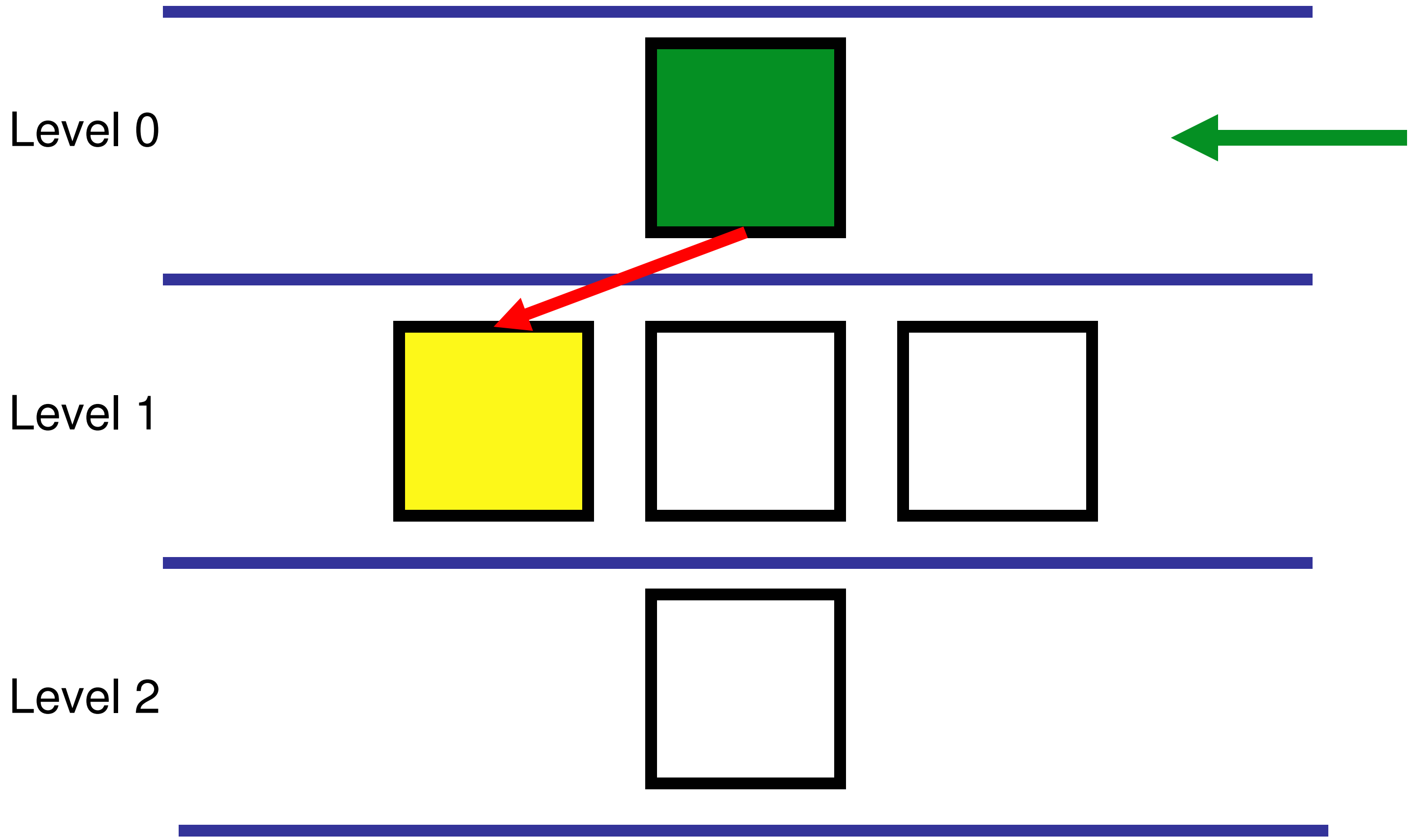
Running A Cycle



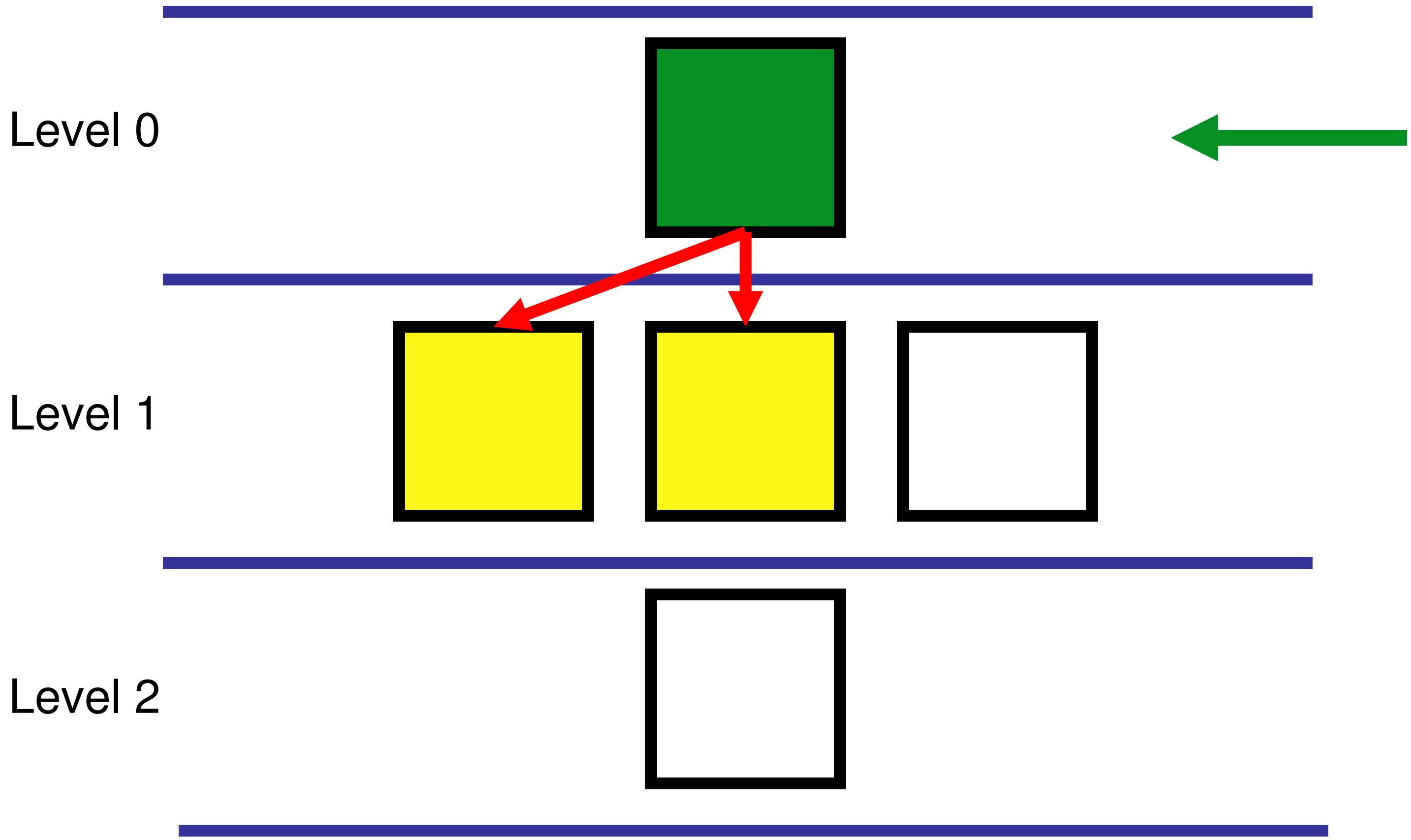
Running A Cycle



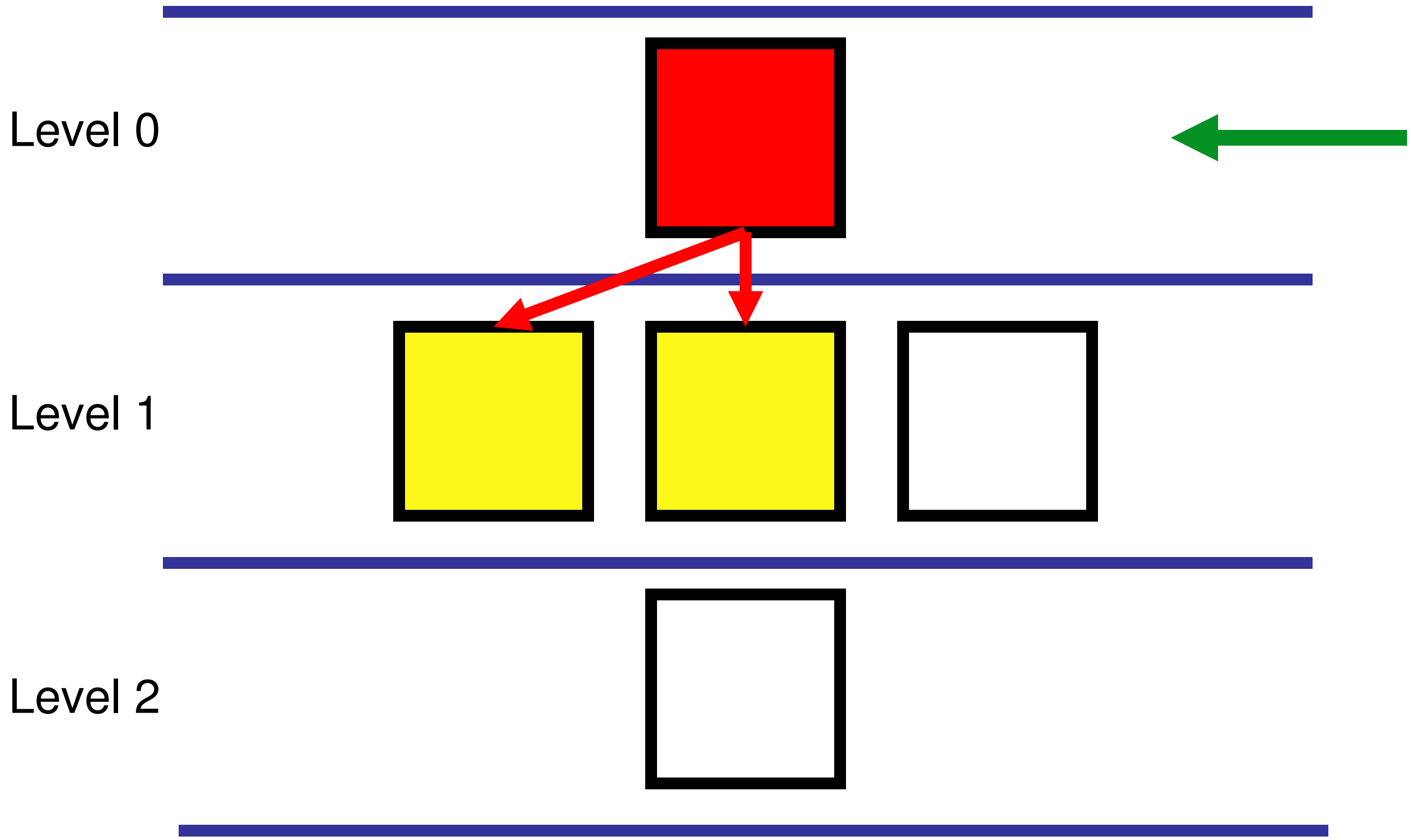
Running A Cycle



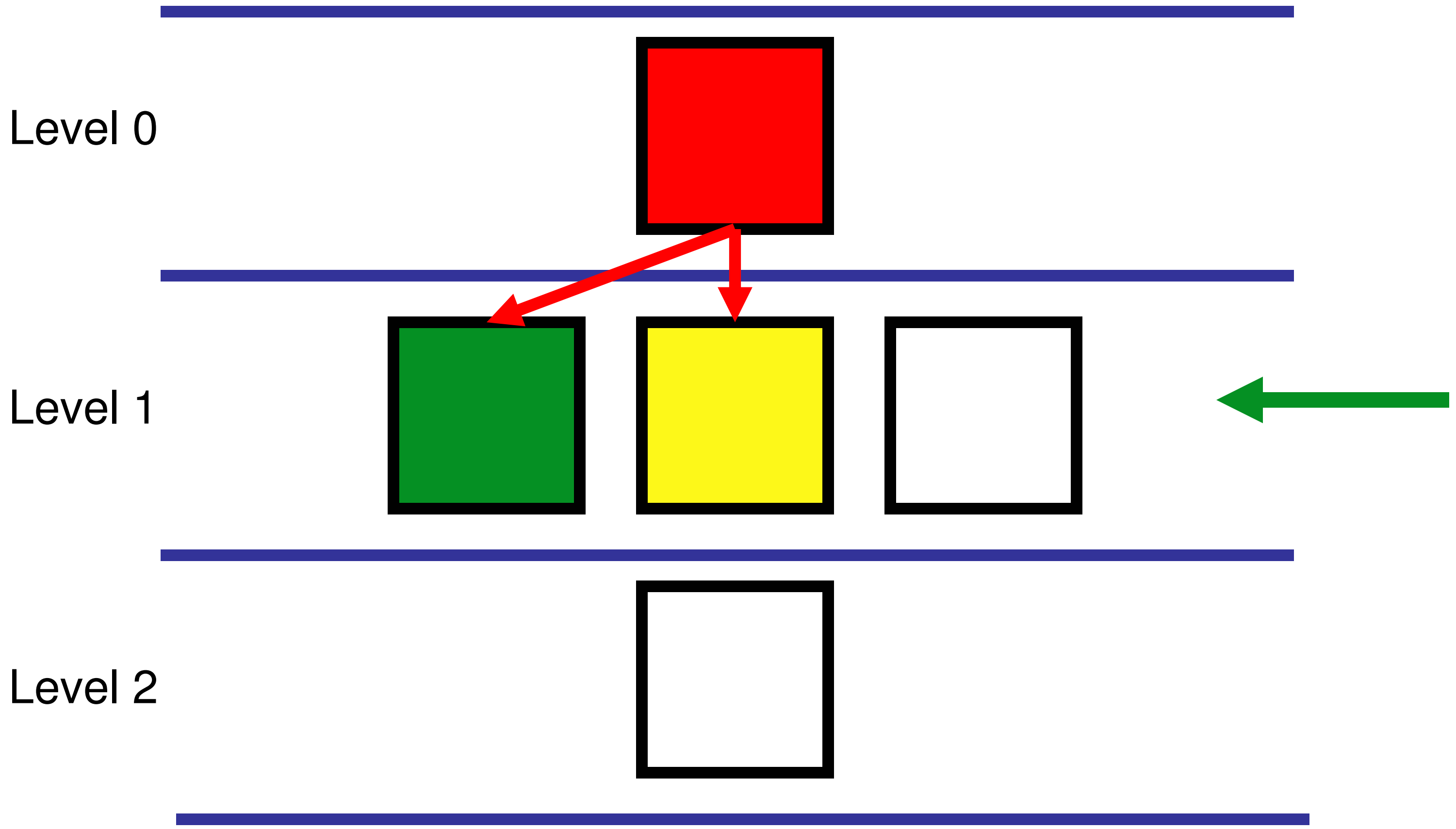
Running A Cycle



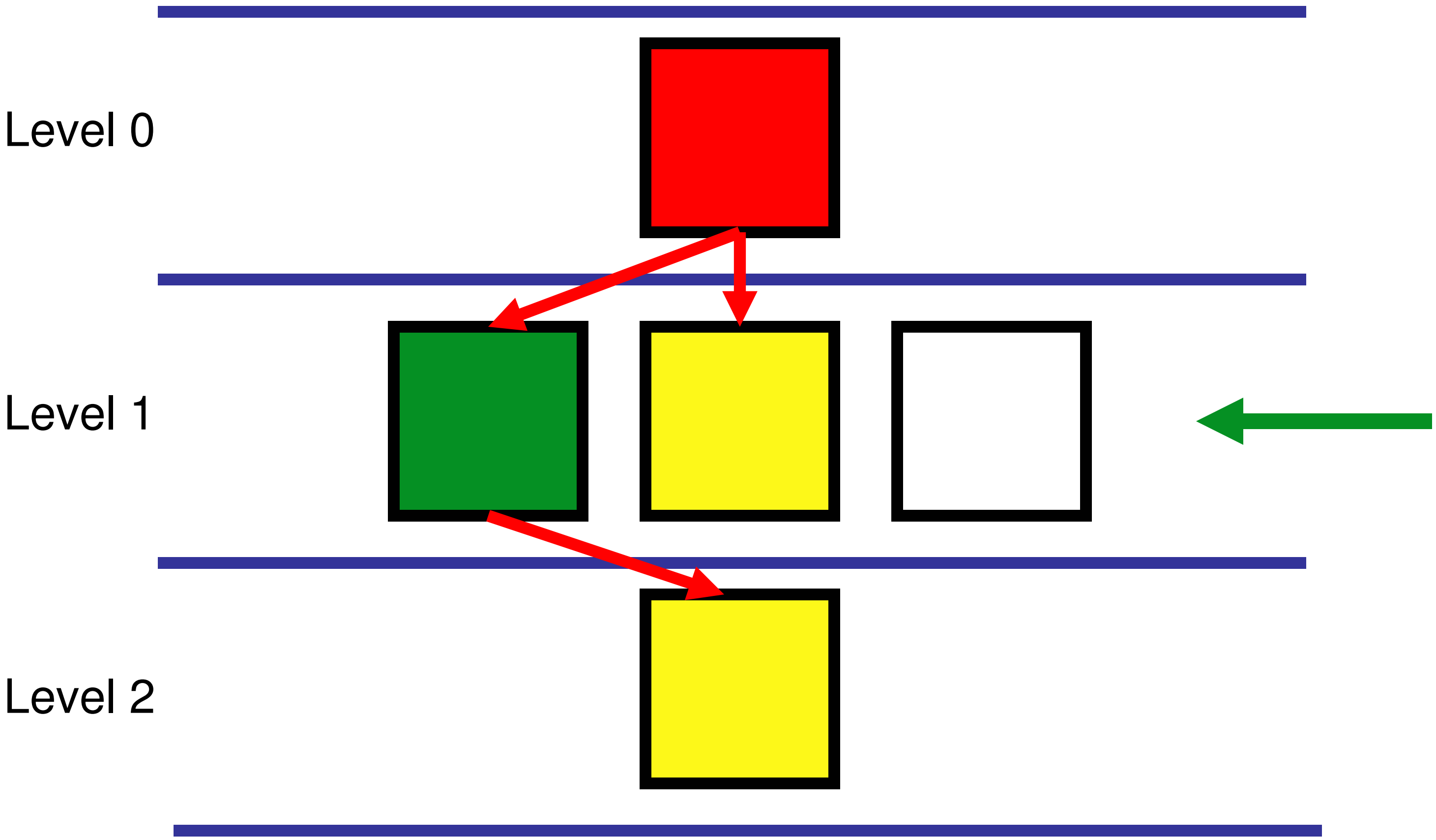
Running A Cycle



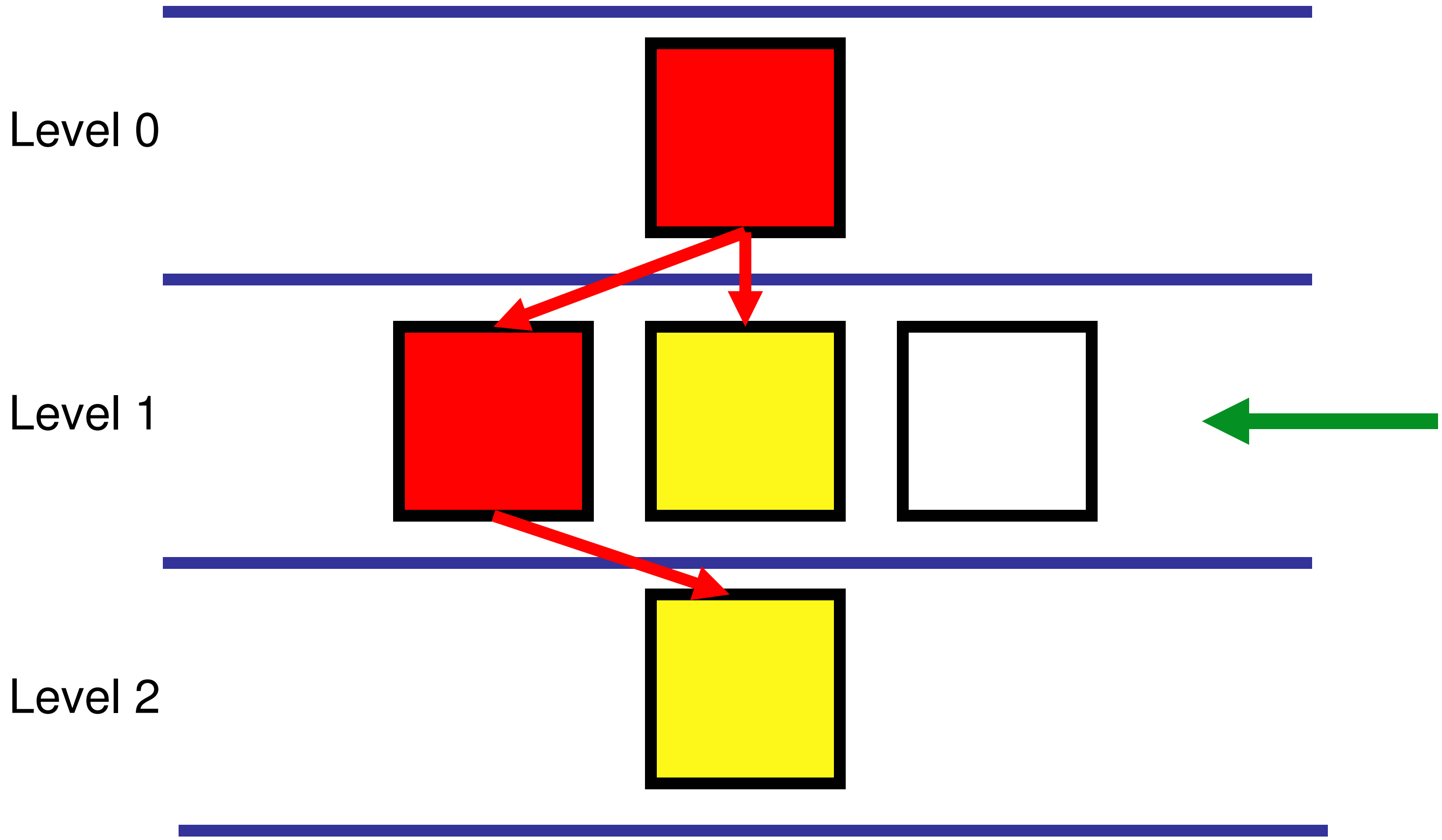
Running A Cycle



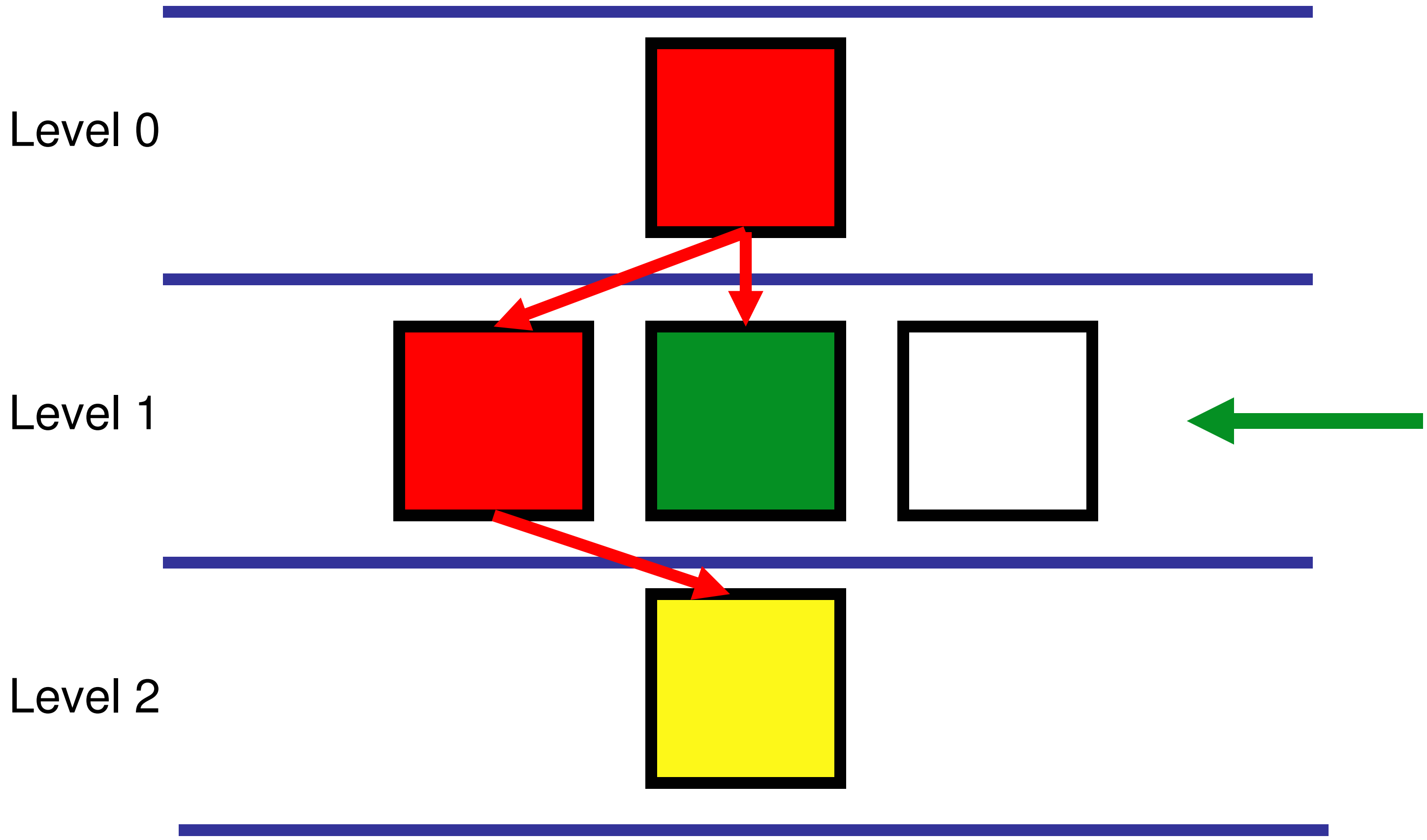
Running A Cycle



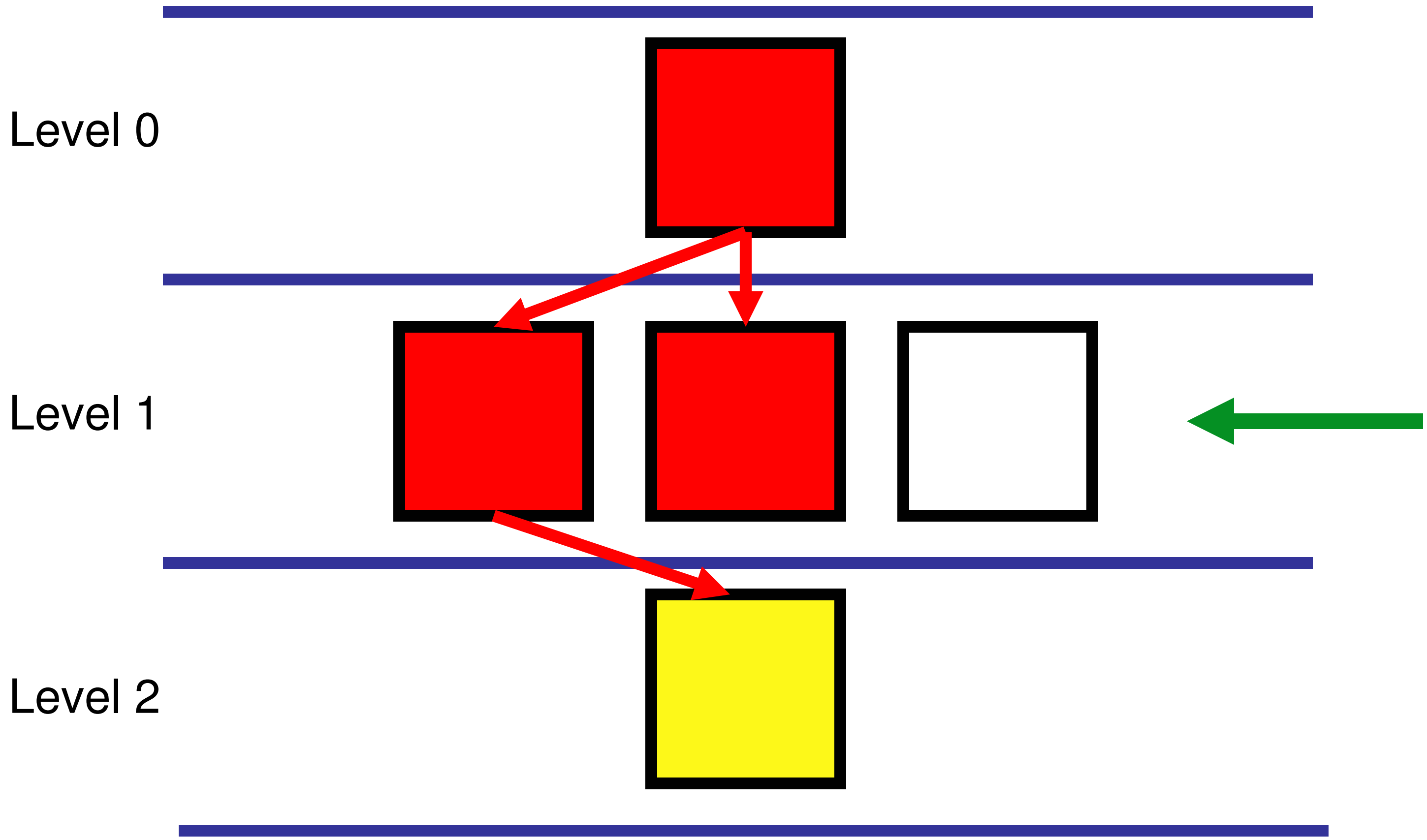
Running A Cycle



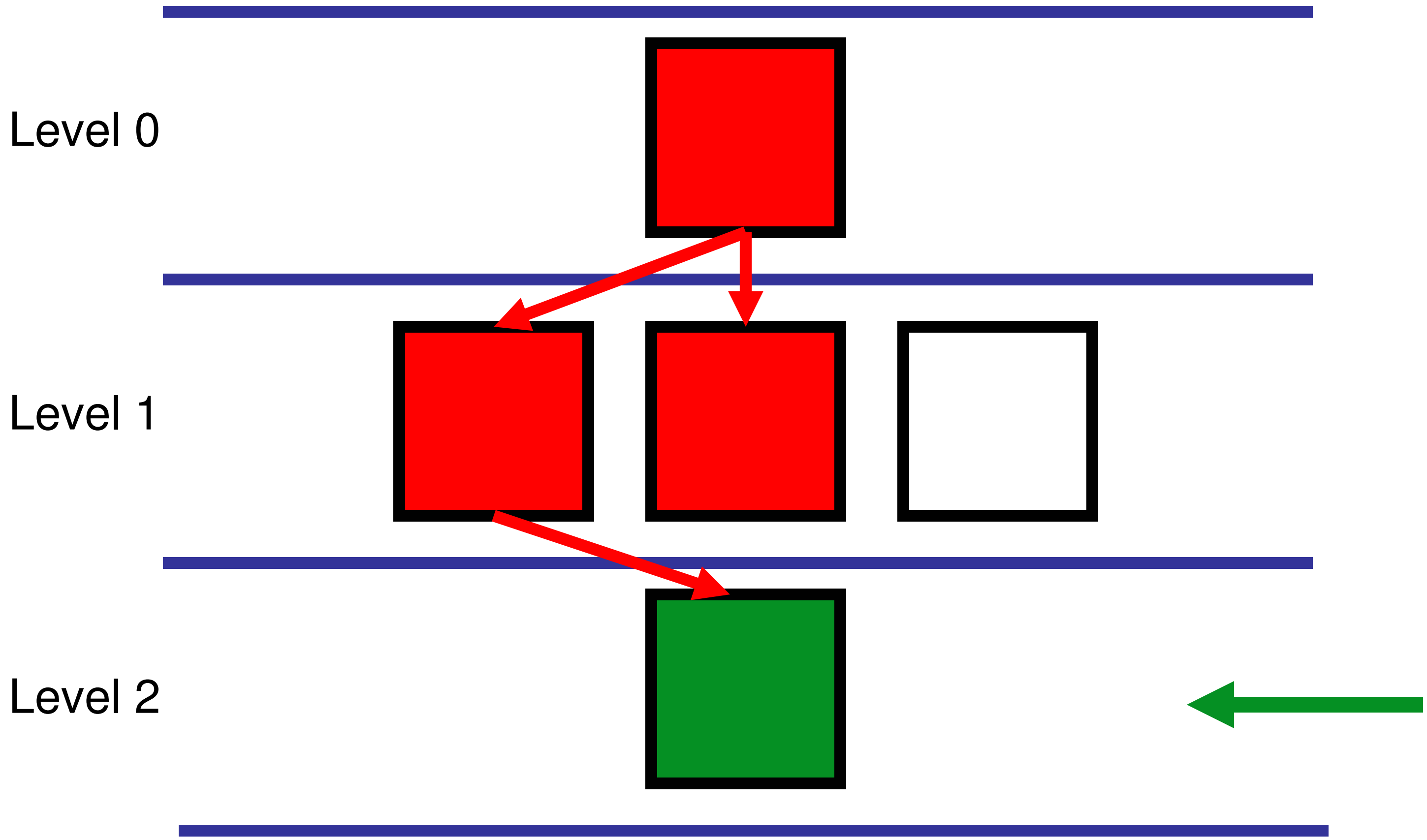
Running A Cycle



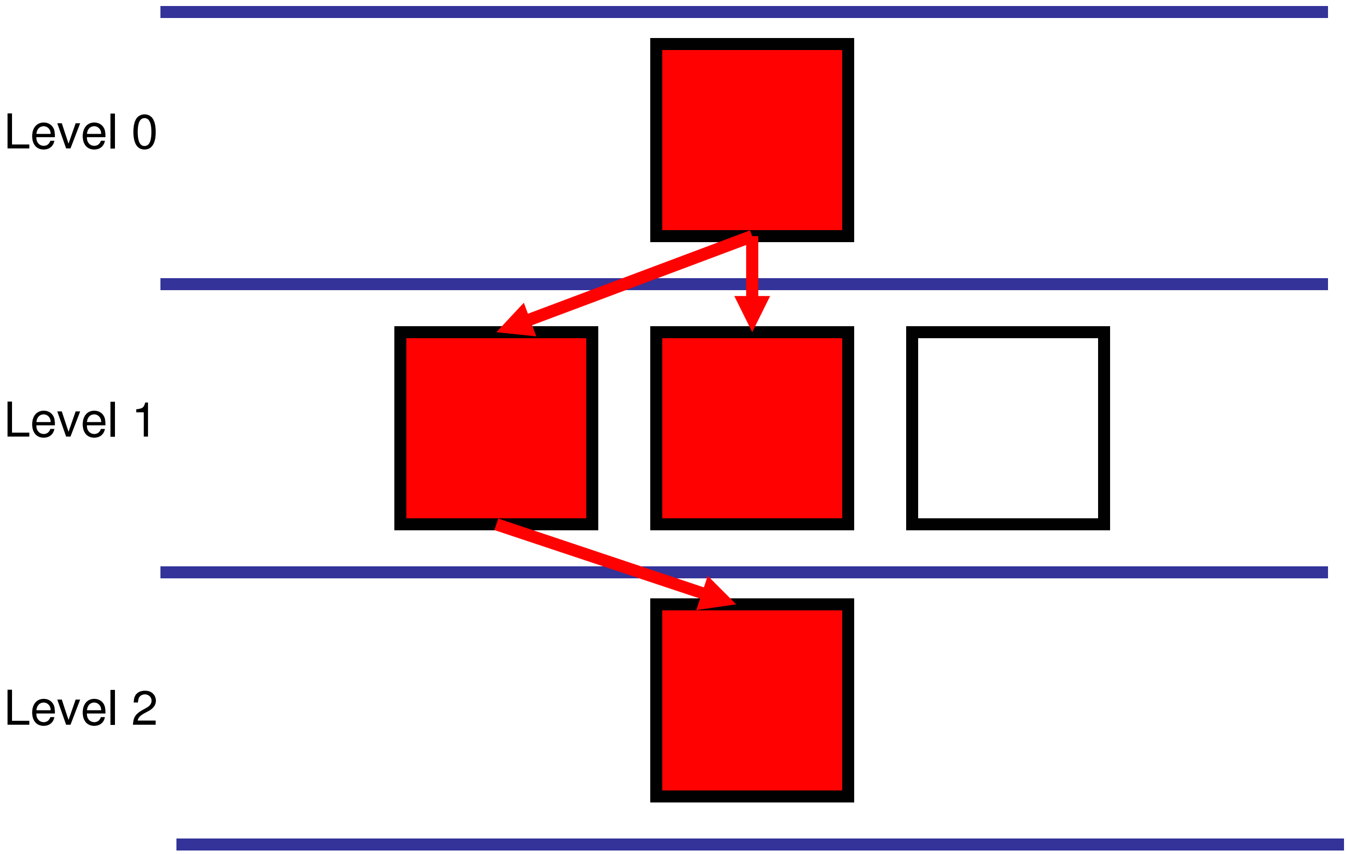
Running A Cycle



Running A Cycle

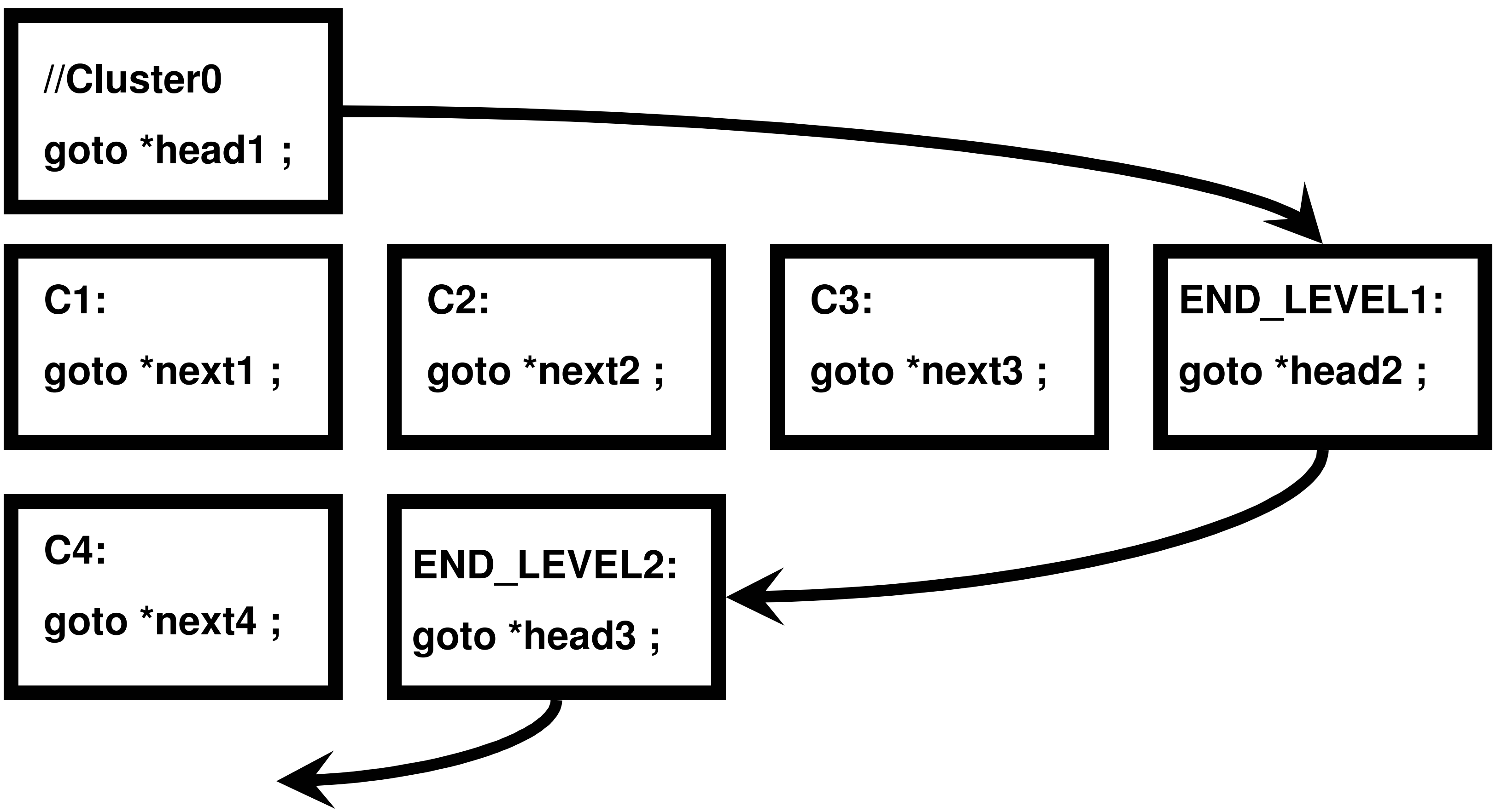


Running A Cycle



Running A Cycle

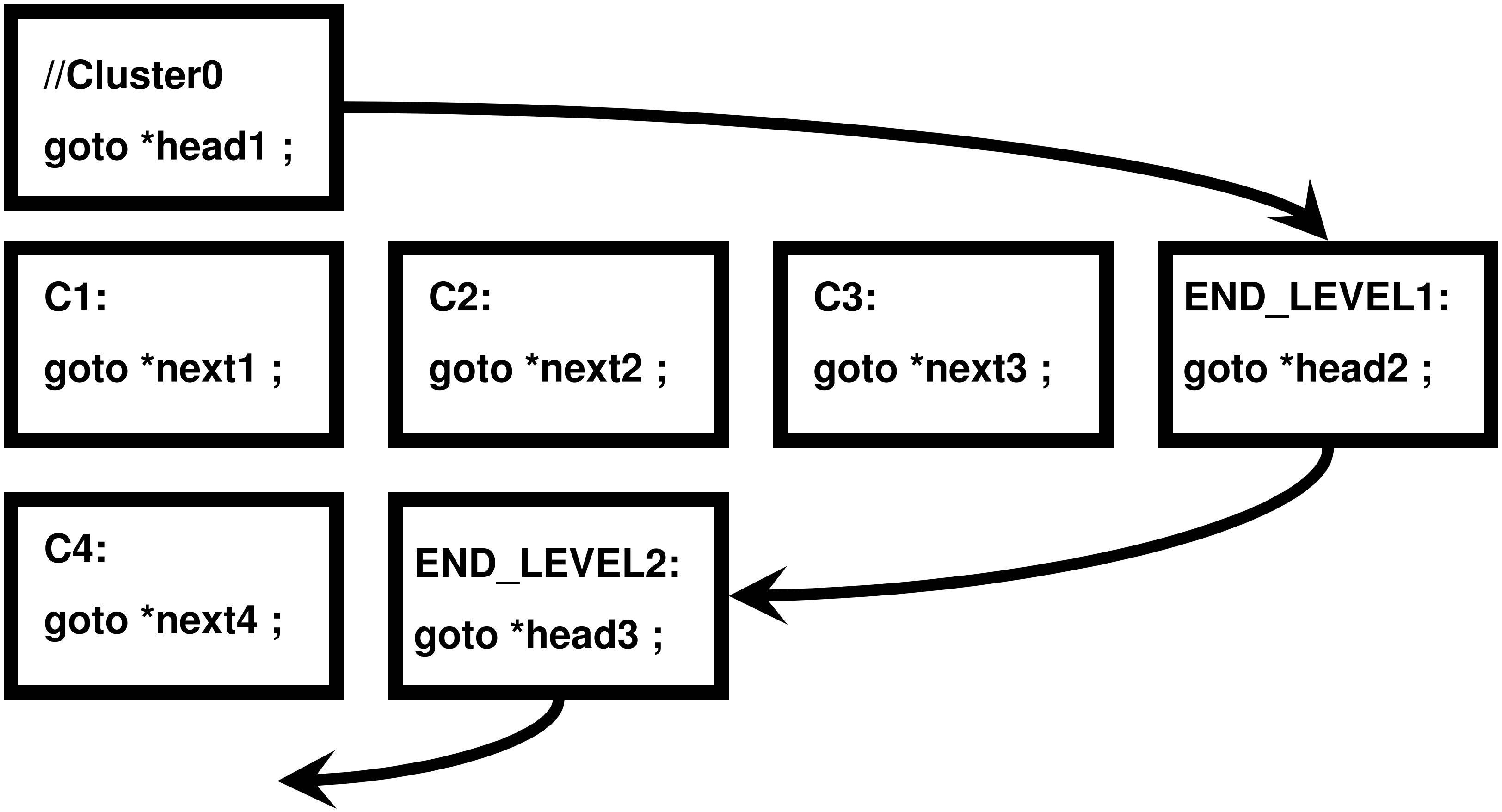
Linked list structure with nothing scheduled



Only have to run cluster 0 and jump to each level

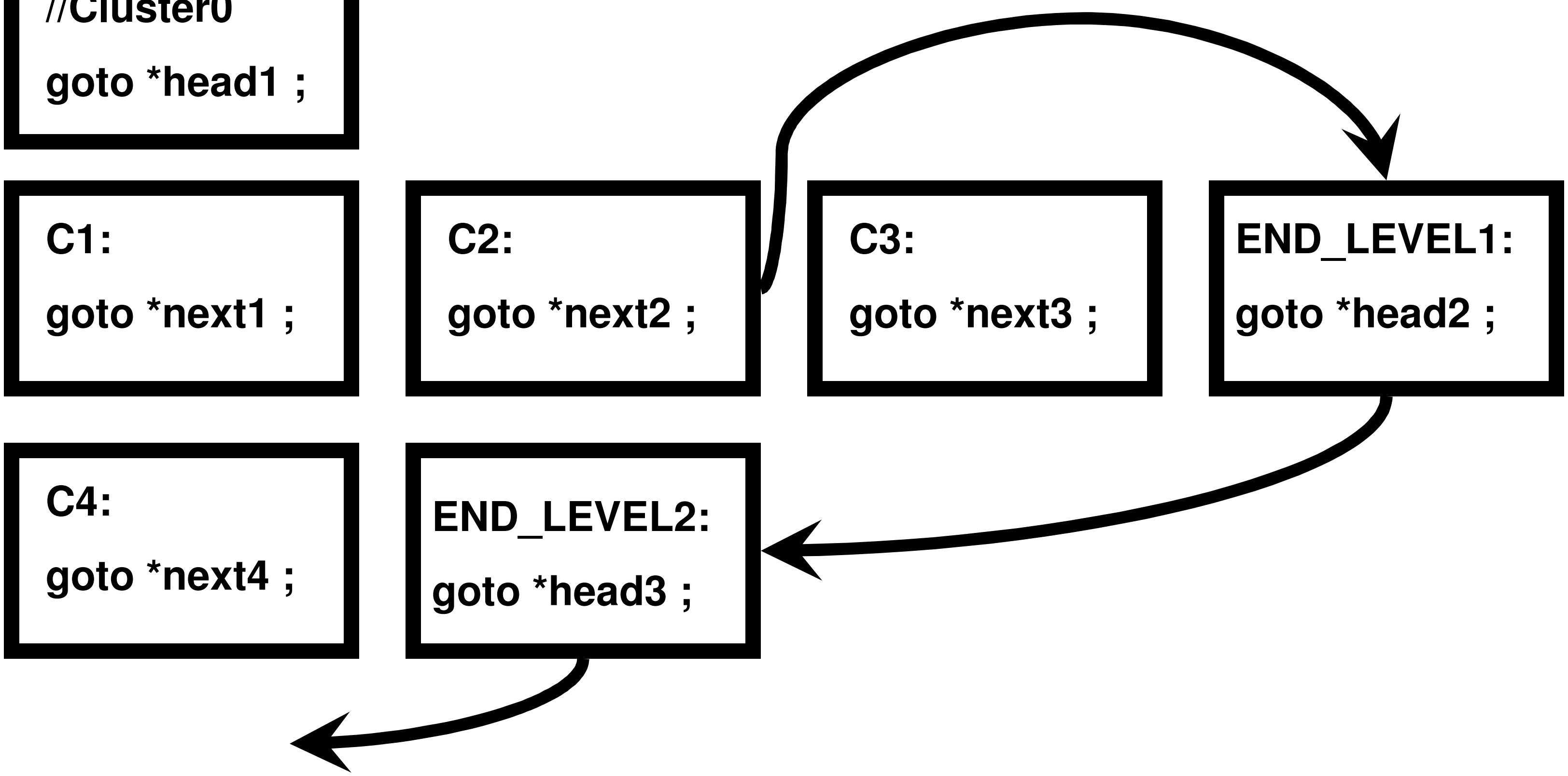
Schedule cluster 2 in the empty structure

next2 = head1, head1 = &&C2



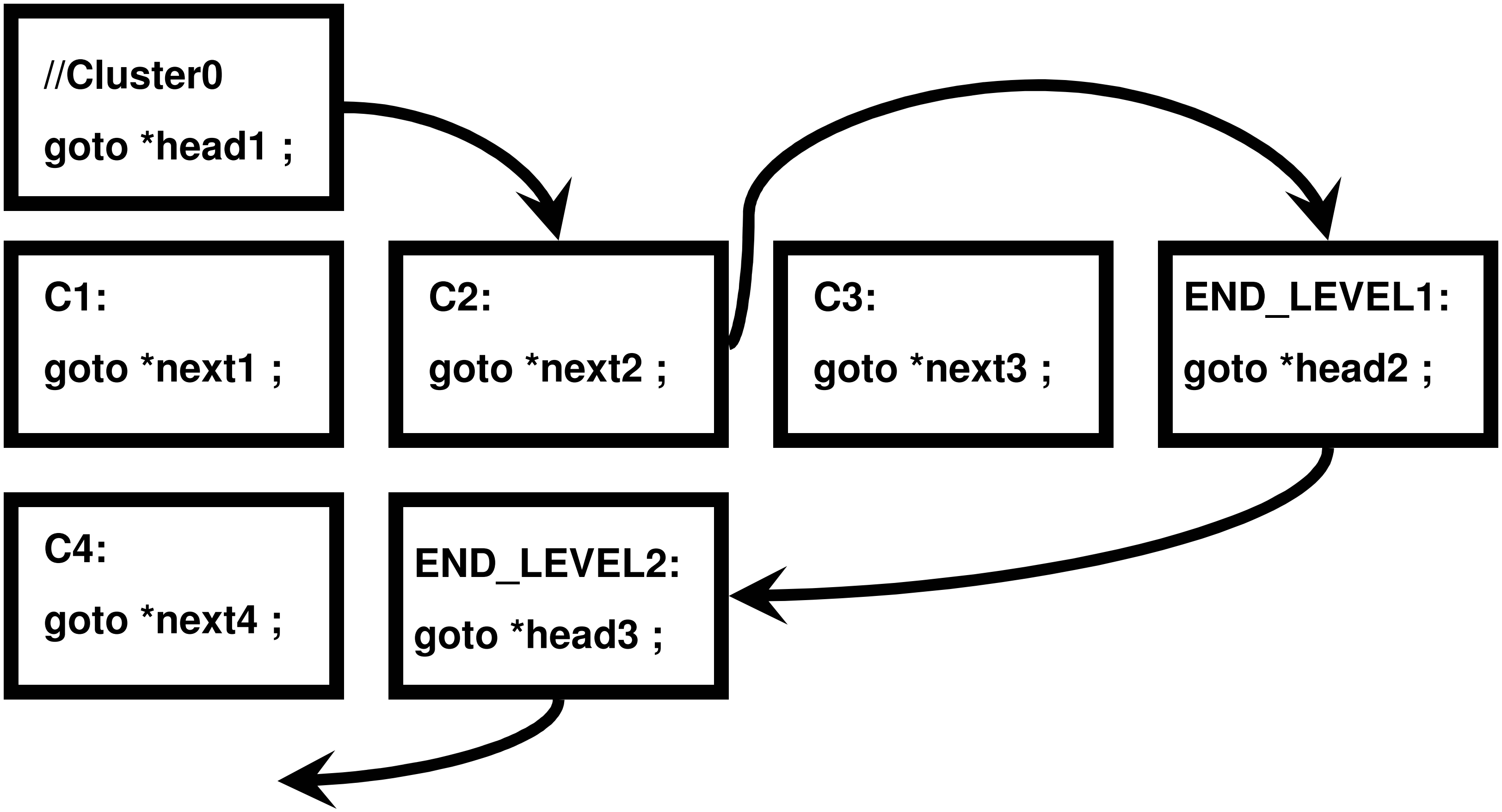
Schedule cluster 2 to the empty structure

next2 = head1, *head1 = &&C2*



Schedule cluster 2 to the empty structure

next2 = head1, ***head1 = &&C2***



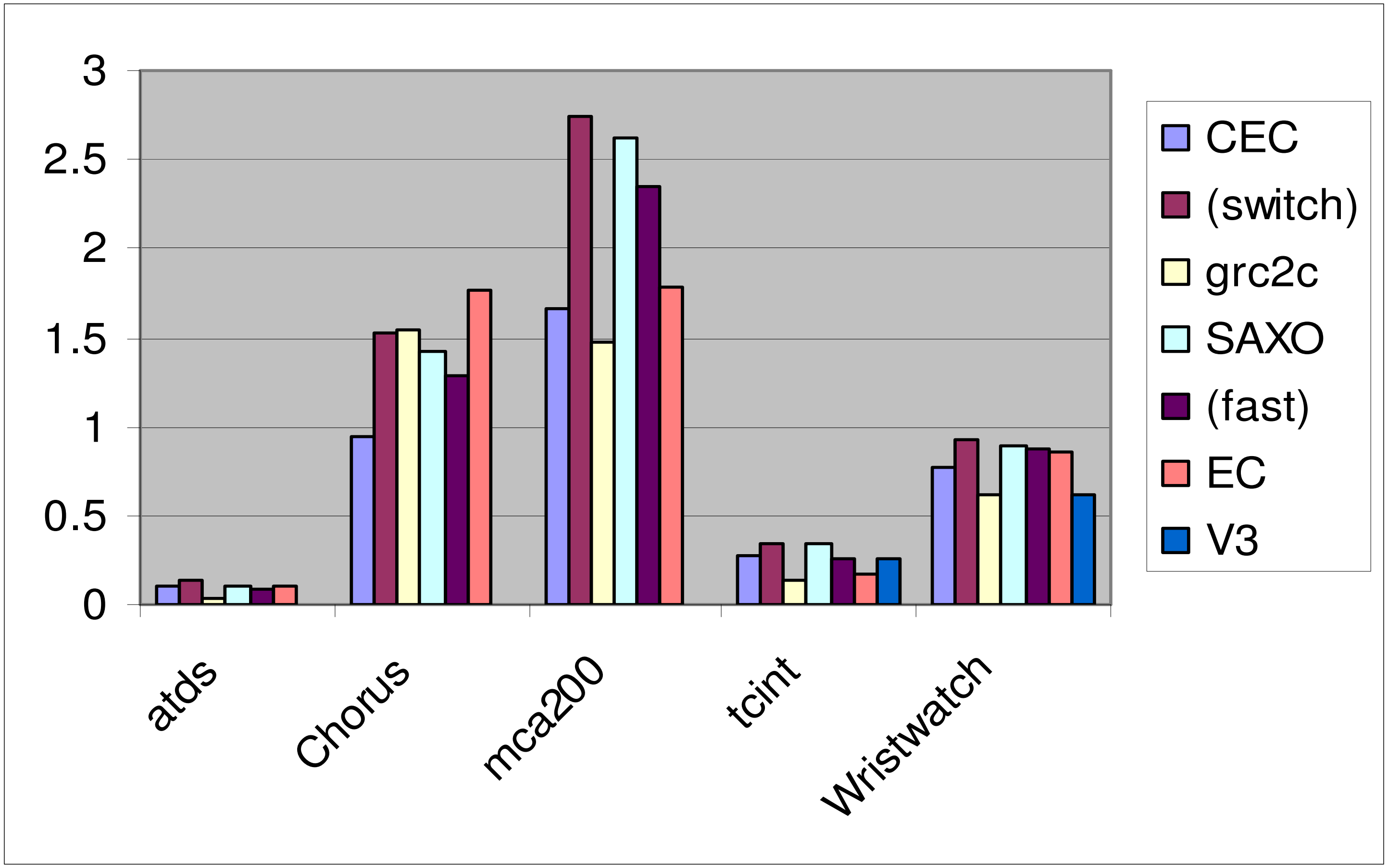
Experimental Results

Five medium sized examples

- Potop-Butucaru's grc2c
  - Beats us on four of the five examples
  - We are substantially faster on the largest example
- SAXO-RT compiler
  - We are faster on the three largest examples

- Most closely resembles SAXO-RT
  - Basic blocks
    - Sorted topologically
    - Executed based on run-time scheduling decisions
  - Two main differences:
    - Only schedule blocks within the current cycle
    - Linked list that eliminates conditional test instead of a scoreboard

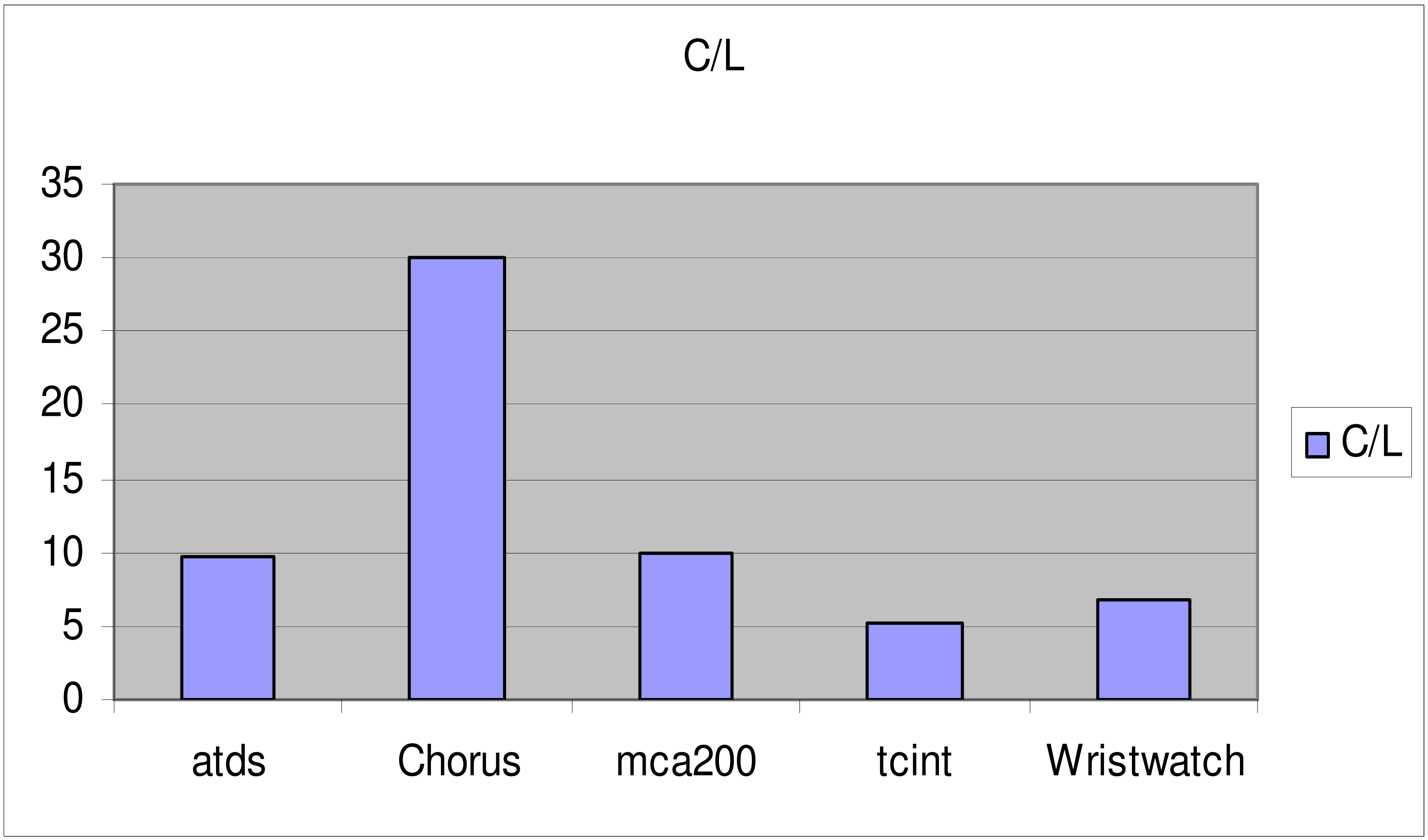
Time in seconds to execute 1 000 000 iterations of the generated code on a 1.7 GHz Pentium 4.



The height of the bars indicates the time in seconds. (Shorter is better)

C/L: Clusters Per Level

The higher C/L the better



Conclusion

- Results in improved running times over an existing compiler that uses a similar technique (SAXO-RT)
- Faster than the fastest-known compiler in the largest example (Potop-Butucaru's)

Source and object code for the compiler described in this presentation is freely available as part of the Columbia Esterel Compiler distribution available from:

http://www.cs.columbia.edu/~sedwards/cec/