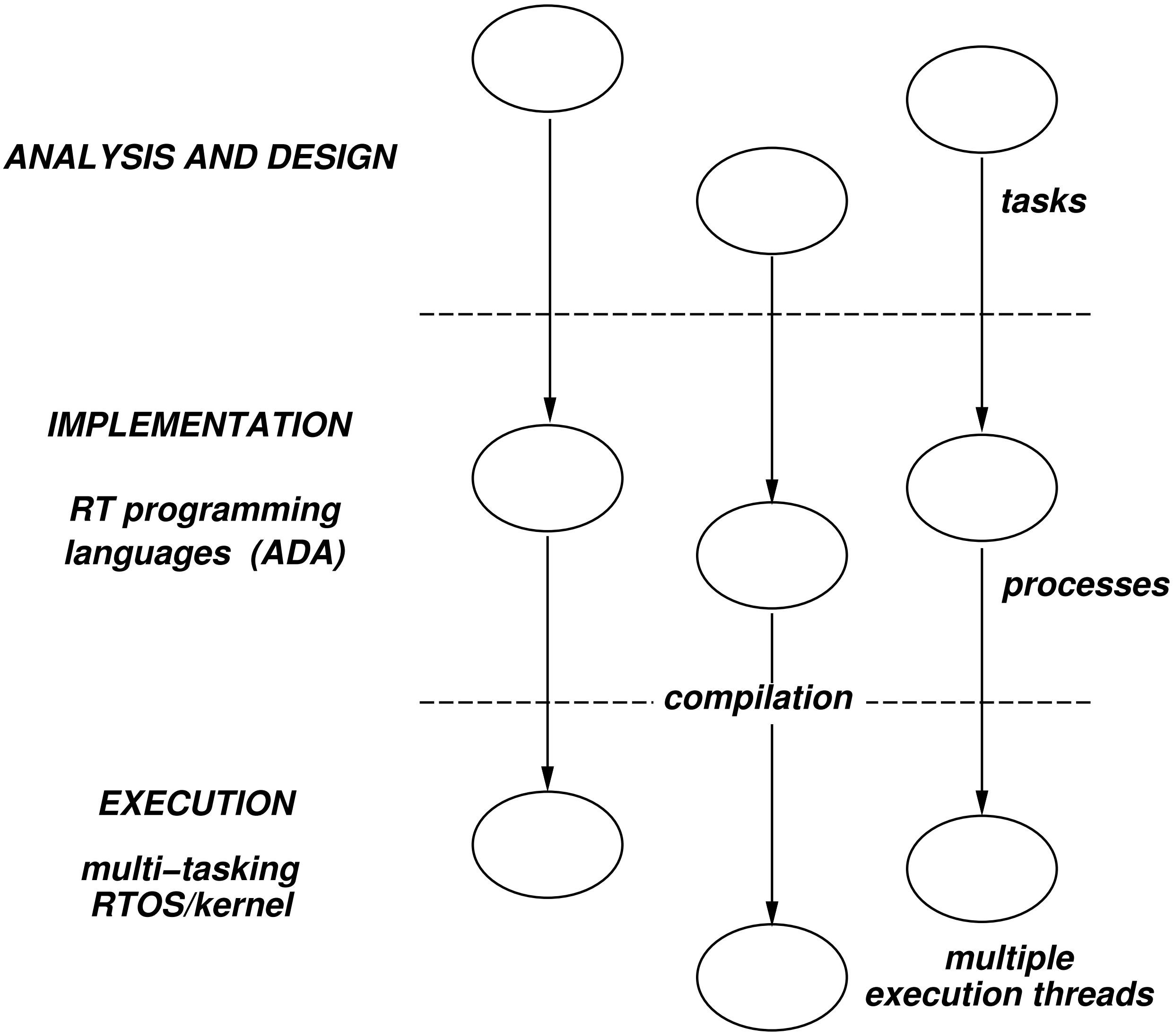
Real-Time Programming

- Concurrent, task-based approach
  - multi-tasking RTOS (VxWorks, Spring)
  - scheduling theory
  - industrial state-of-the-art

- Synchronous approach
  - reactive kernel represented as an automata
  - formal properties
  - special-purpose programming languages - Esterel

Copyright ©1996 by Roderic A. Grupen

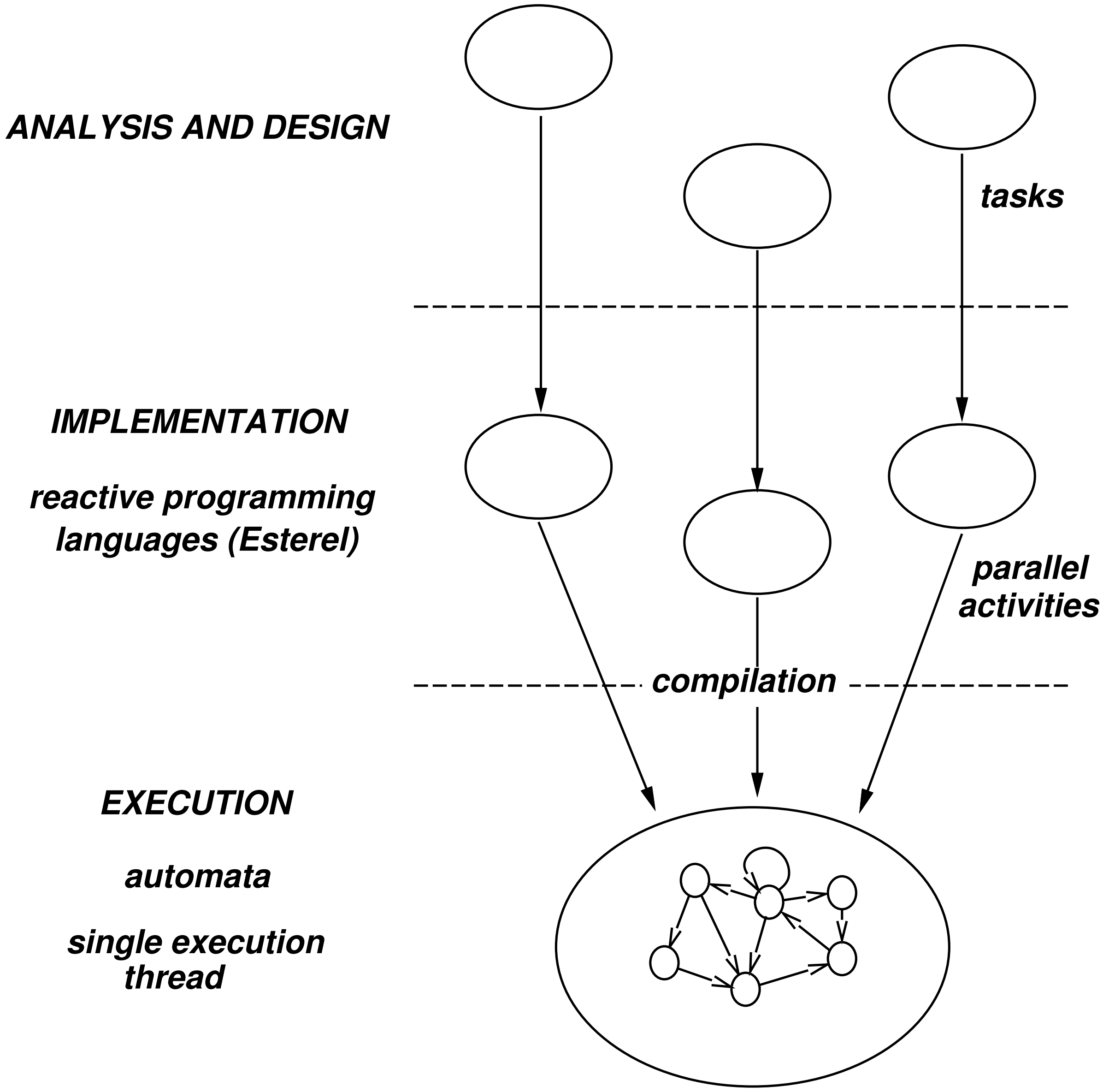
Concurrent Programming



concurrency in control "situations"

Copyright ©1996 by Roderic A. Grupen

Synchronous Programming



discrete event dynamic systems — DEDS

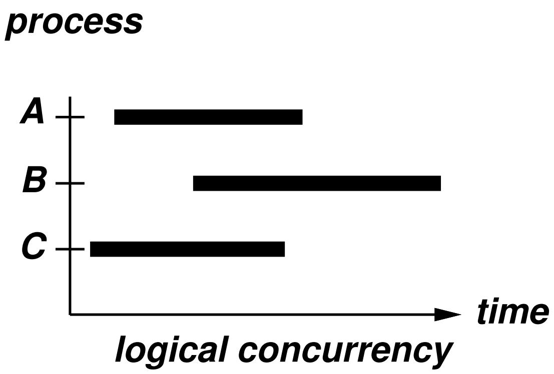
Copyright ©1996 by Roderic A. Grupen

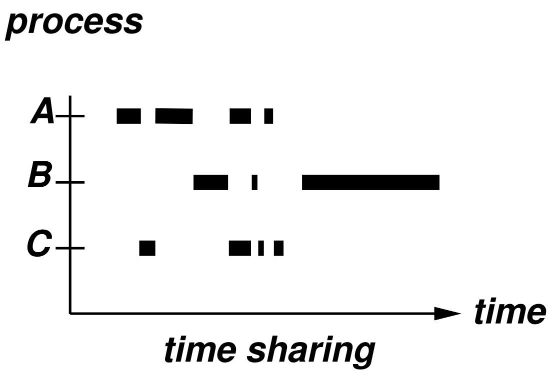
Concurrent Programming

A concurrent program can be viewed as a collection of autonomous sequential processes executing (logically) in parallel. Each process has a single thread of control.

Processes can either:

1. multiplex their operation on a single processor;
2. multiplex their execution on a multi-processor with shared memory; or
3. multiplex their execution on several processors without shared memory (distributed systems).

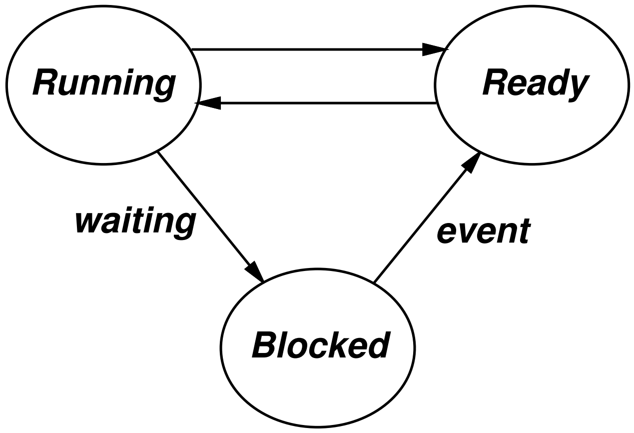




Copyright ©1996 by Roderic A. Grupen

Processes

A process (task) can be in one of the following process states:



Ready: Processes that are ready to execute. Stored in a ReadyQueue (a linked list)

Running: The currently executing process (the first process in the ReadyQueue)

Blocked: Processes waiting for an event, e.g.,

- an external event ($Temp < Tmin$)
- a temporal event

The switch from one running process to another is called a **context switch**

A wait operation forces a context switch.

Copyright ©1996 by Roderic A. Grupen

Processes (cont.)

During the context switch, the state of the running process is saved and the state of the process that will run next is restored.

The process state consists of:

- program counter,
- process status information (priority, enabled interrupts), and
- the contents of the programmable registers.

Process administration is handled by the RT kernel

Copyright ©1996 by Roderic A. Grupen

Semaphores — review

...simple mechanism for mutual exclusion and conditional (asymmetric) and symmetric synchronization — Dijkstra (1968)

Definition: a non-negative integer acted upon by two atomic procedures (obtained by disabling interrupts): *wait()* and *signal()*

```
wait(s) {
    if (s == 0) {
        change process state
        from Running to Blocked
    }
    else s = s − 1
}
```

```
signal(s) {
    if {Blocked} is not empty {
        insert the first Blocked
        process into ReadyQueue
    }
    else s = s + 1
}
```


Copyright ©1996 by Roderic A. Grupen

Semaphores — review (cont.)

Mutual Exclusion

$s_{init} = 1$

P1:	P2:
wait(s)	wait(s)
$<$ *common resource* $>$	$<$ *common resource* $>$
signal(s)	signal(s)

Copyright ©1996 by Roderic A. Grupen

Semaphores — review (cont.)

Conditional Synchronization/Asymmetric Synchronization

$s_{init} = 0$

P1:	P2:
wait(s)	$\vdots$
$\vdots$	signal(s)

P1 can never run ahead of P2

Symmetric Synchronization

$s1_{init} = s2_{init} = 0$

P1:	P2:
$\vdots$	wait($s1$)
signal($s1$)	$\vdots$
wait($s2$)	signal($s2$)
$\vdots$	

ADA rendezvous is implemented this way

Copyright ©1996 by Roderic A. Grupen