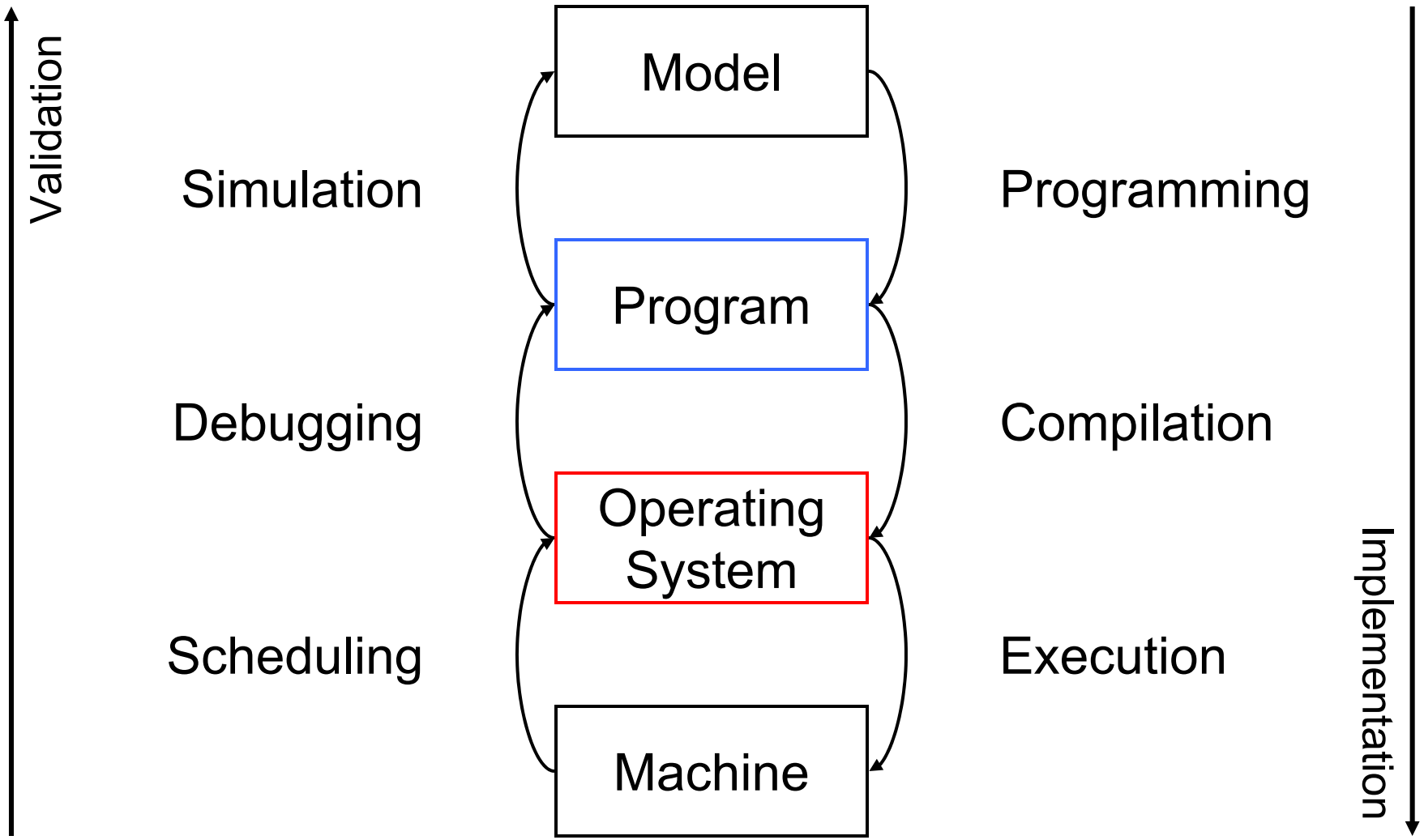
Real-Time Programming Based on Schedule-Carrying Code

Christoph M. Kirsch

UC Berkeley

www.eecs.berkeley.edu/~cm

Outline

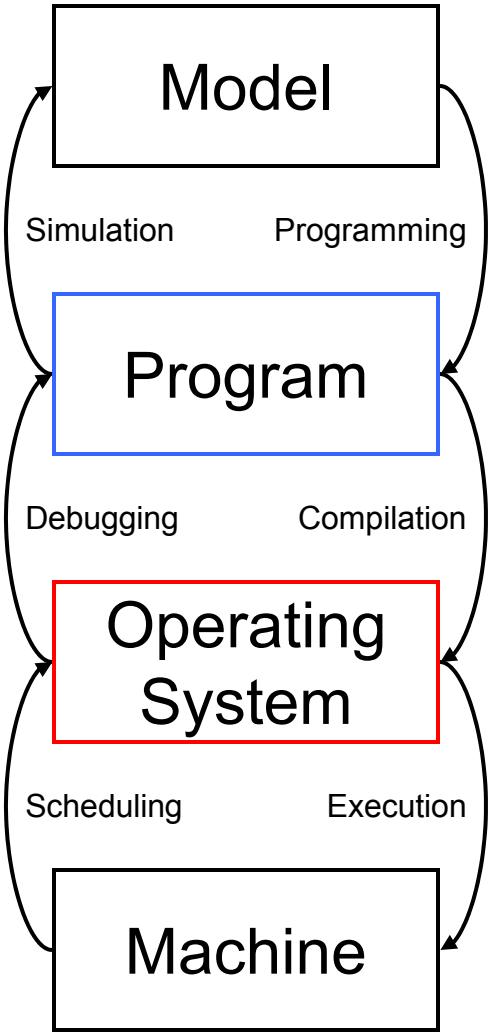

Validation
Simulation
Debugging
Scheduling
Model
Program
Operating System
Machine
Programming
Compilation
Execution
Implementation

Research Areas

Model Validation

System Validation

Scheduling Theory
Program Analysis

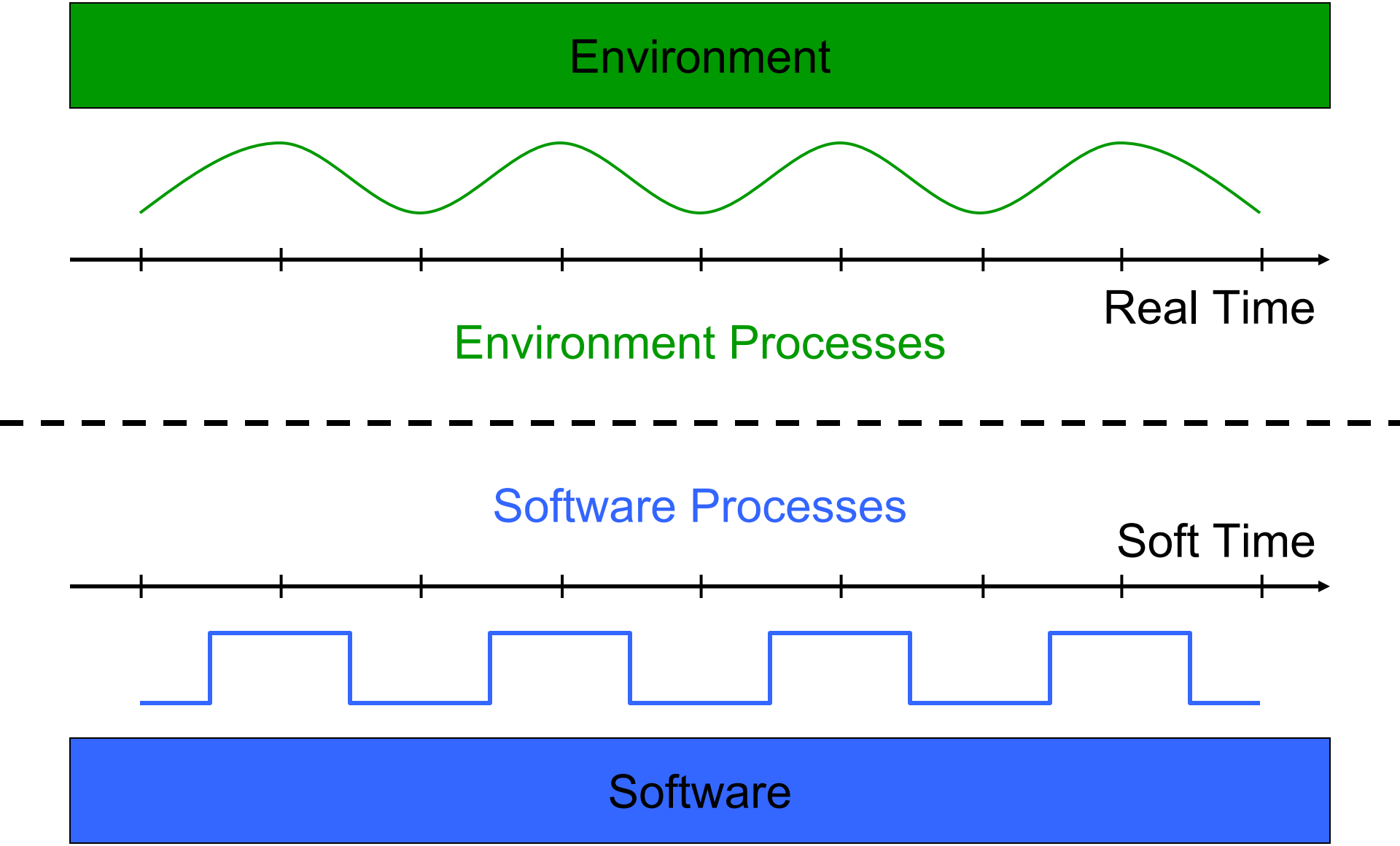


Programming Languages

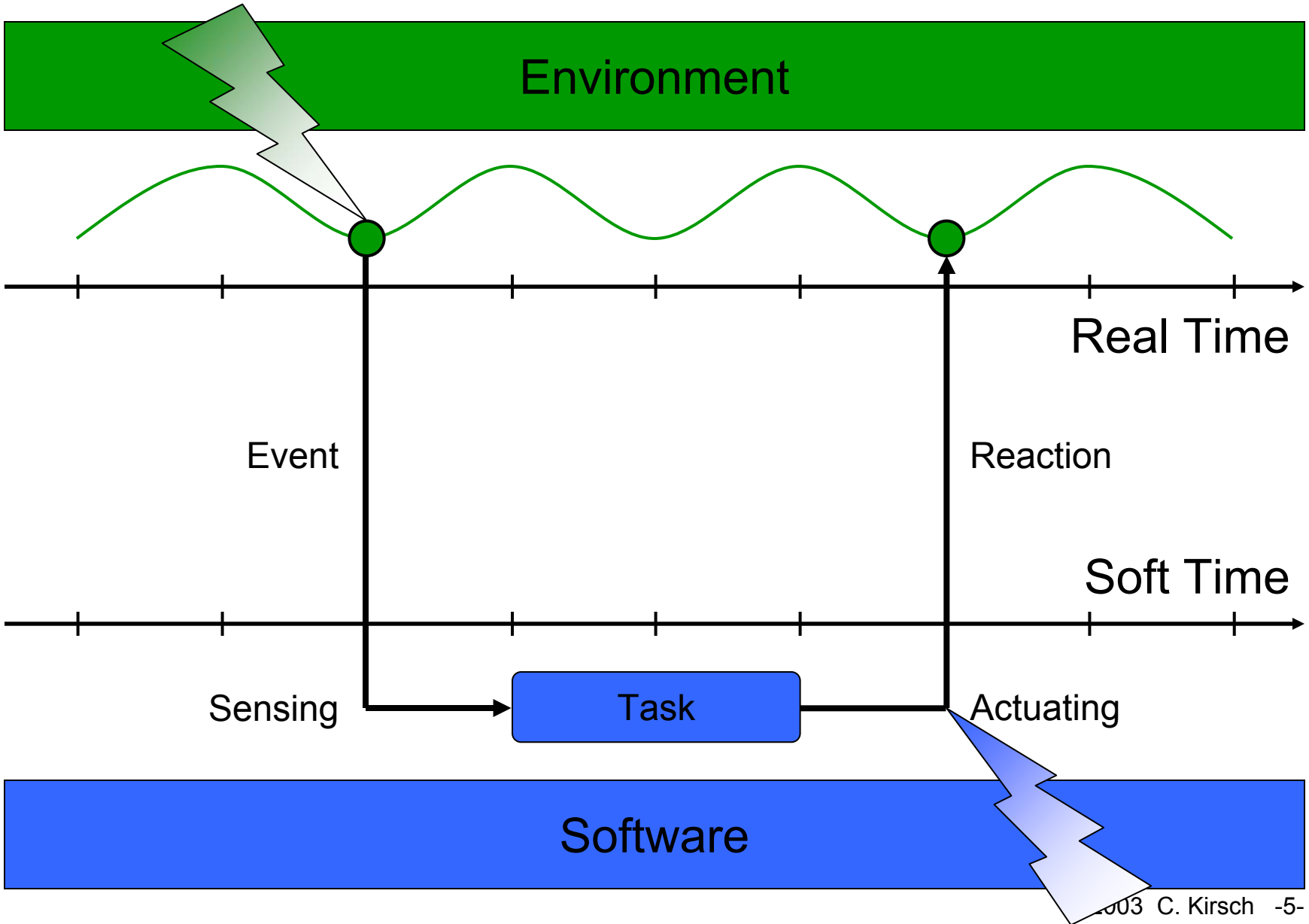
Compiler Design

Operating Systems
Distributed Systems

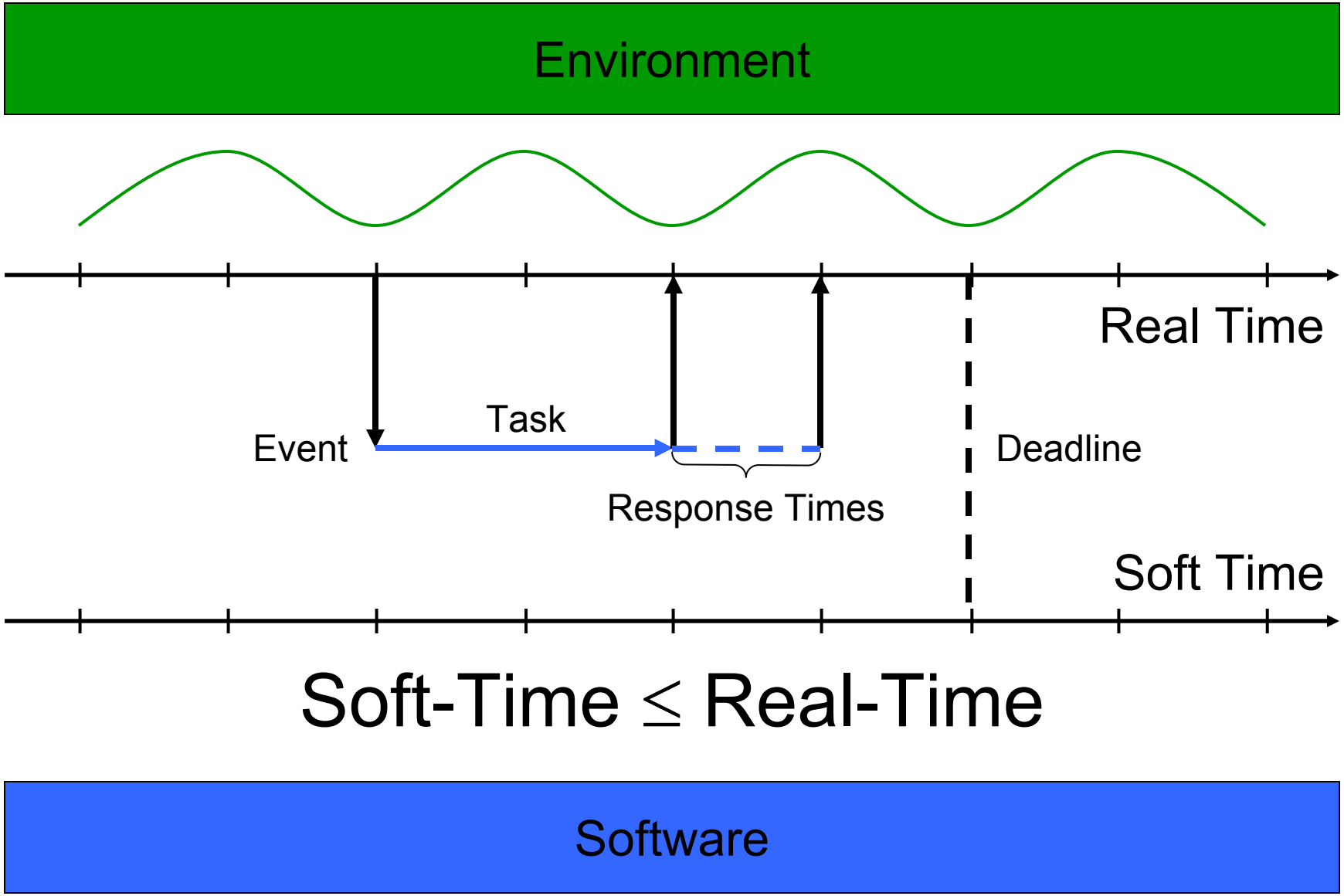
Real Time vs. Soft Time



Sensing, Computing, Actuating

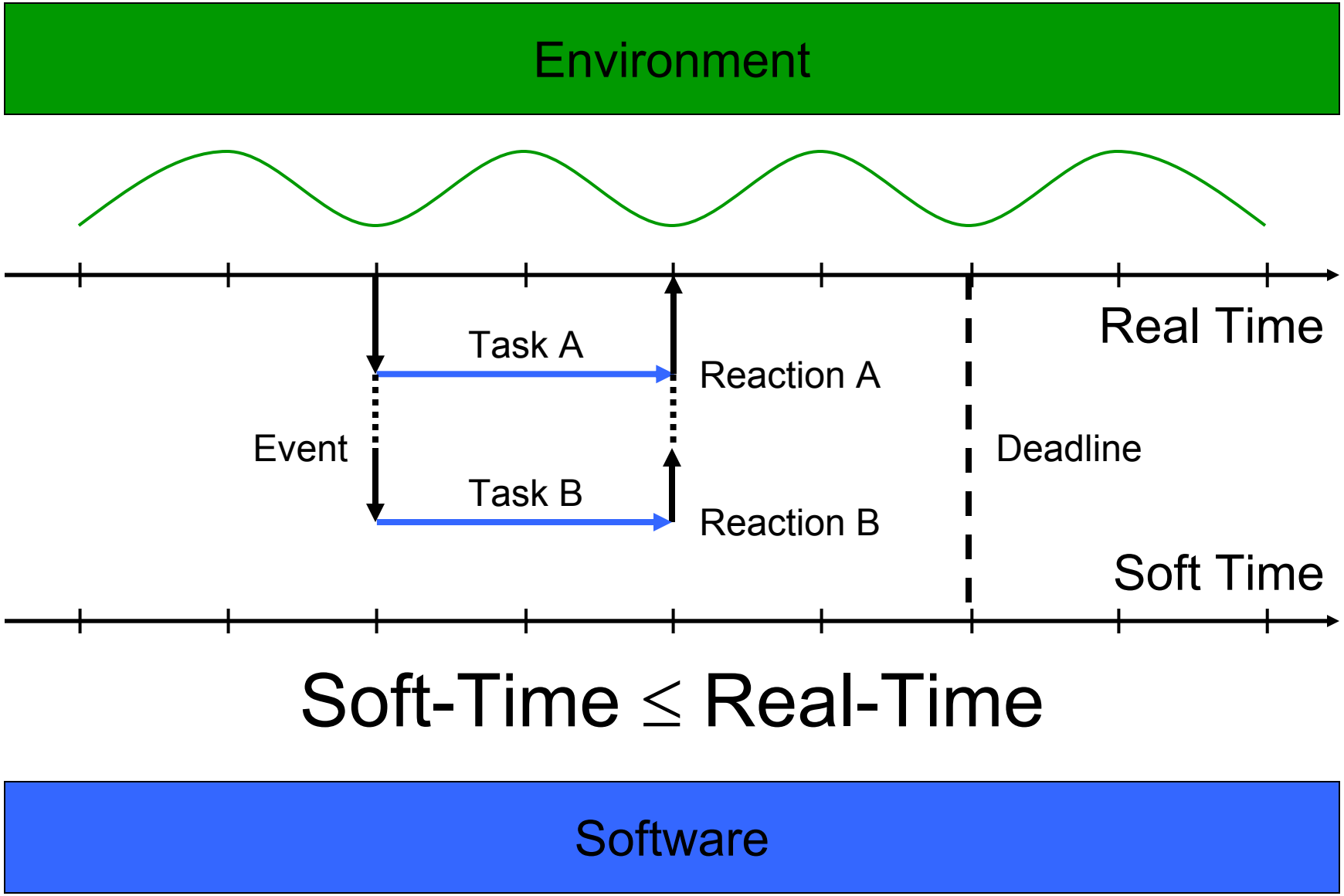


Real-Time Programming



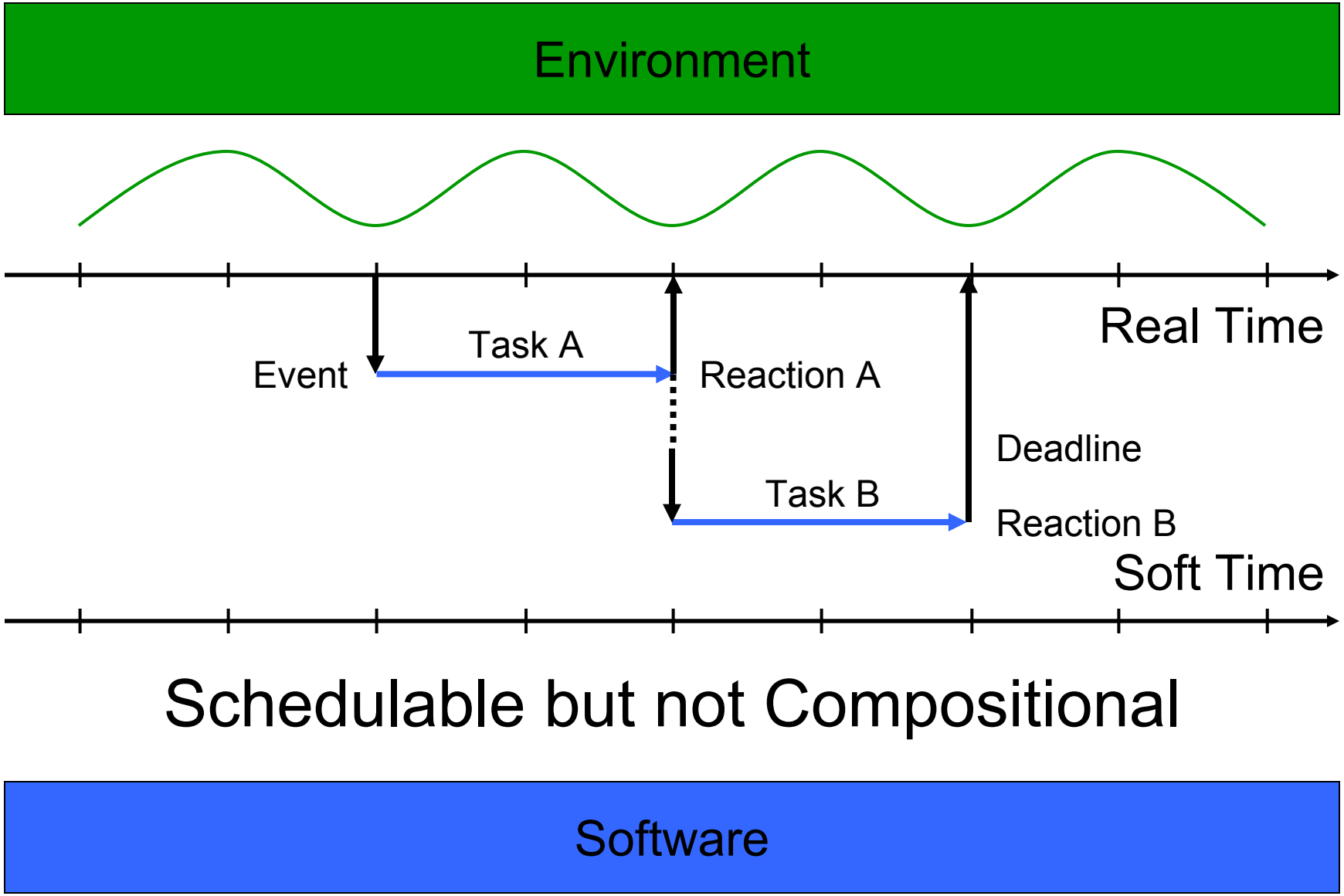
Soft-Time ≤ Real-Time

Concurrency



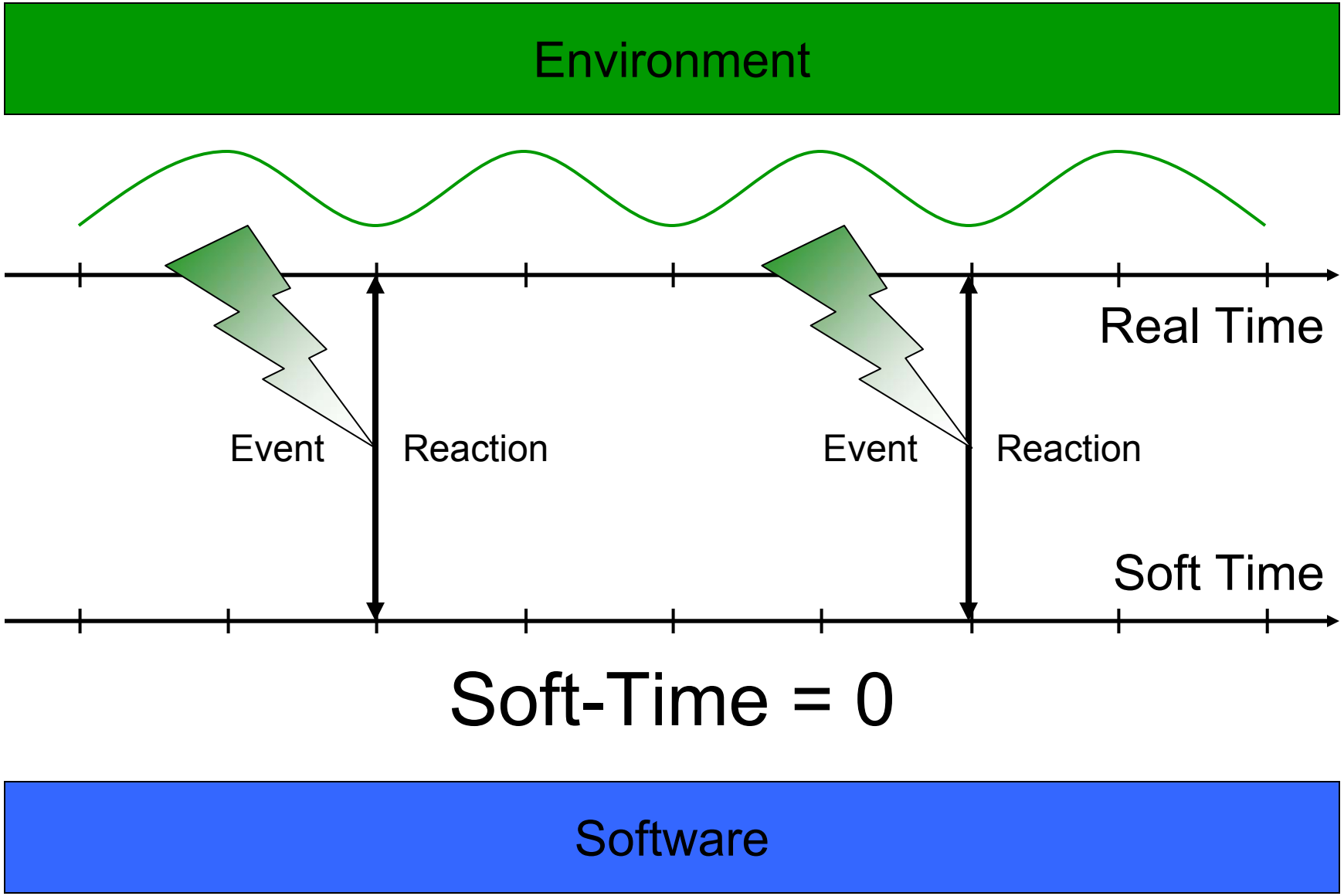
Soft-Time ≤ Real-Time

The Problem

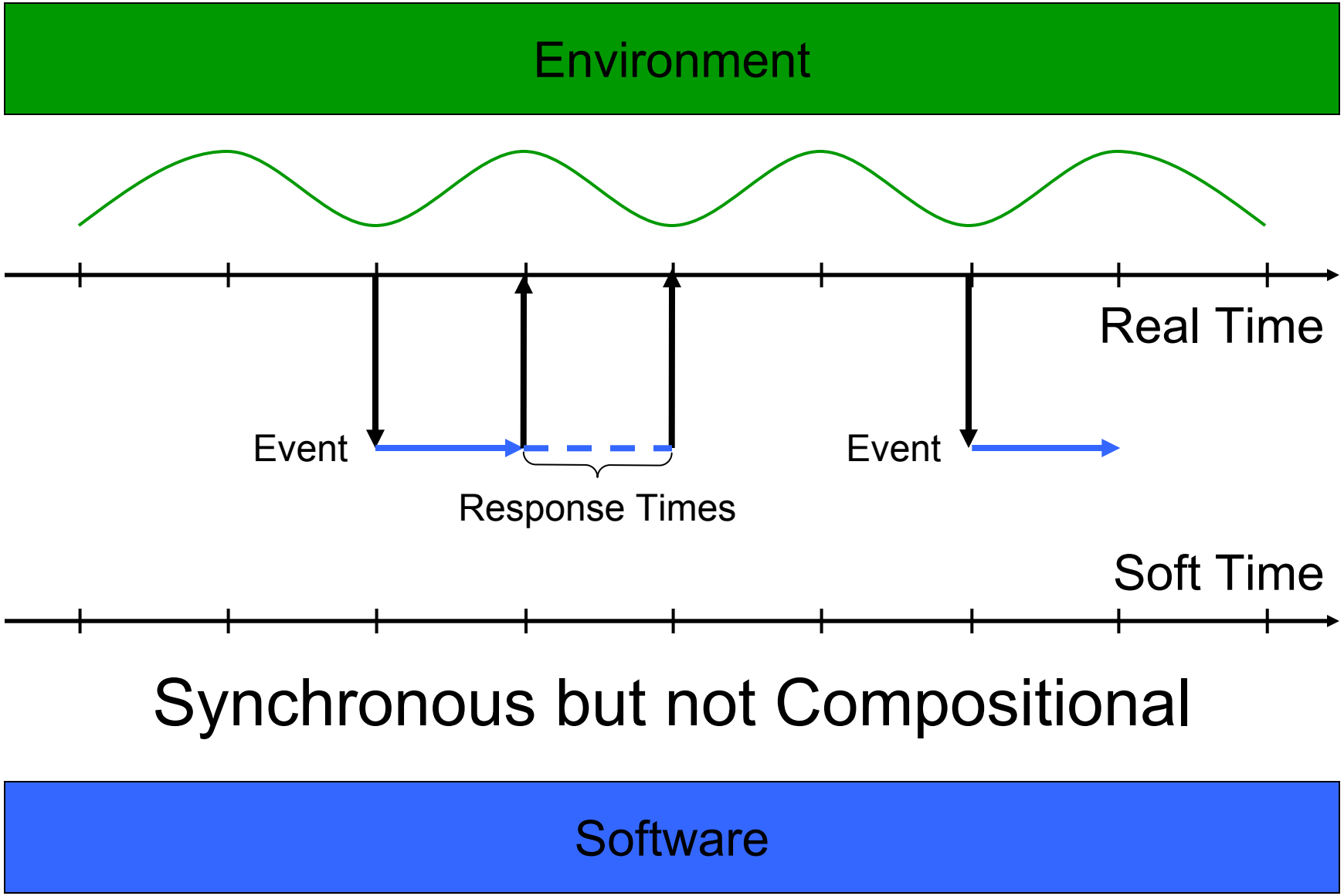


Schedulable but not Compositional

The Synchronous Model

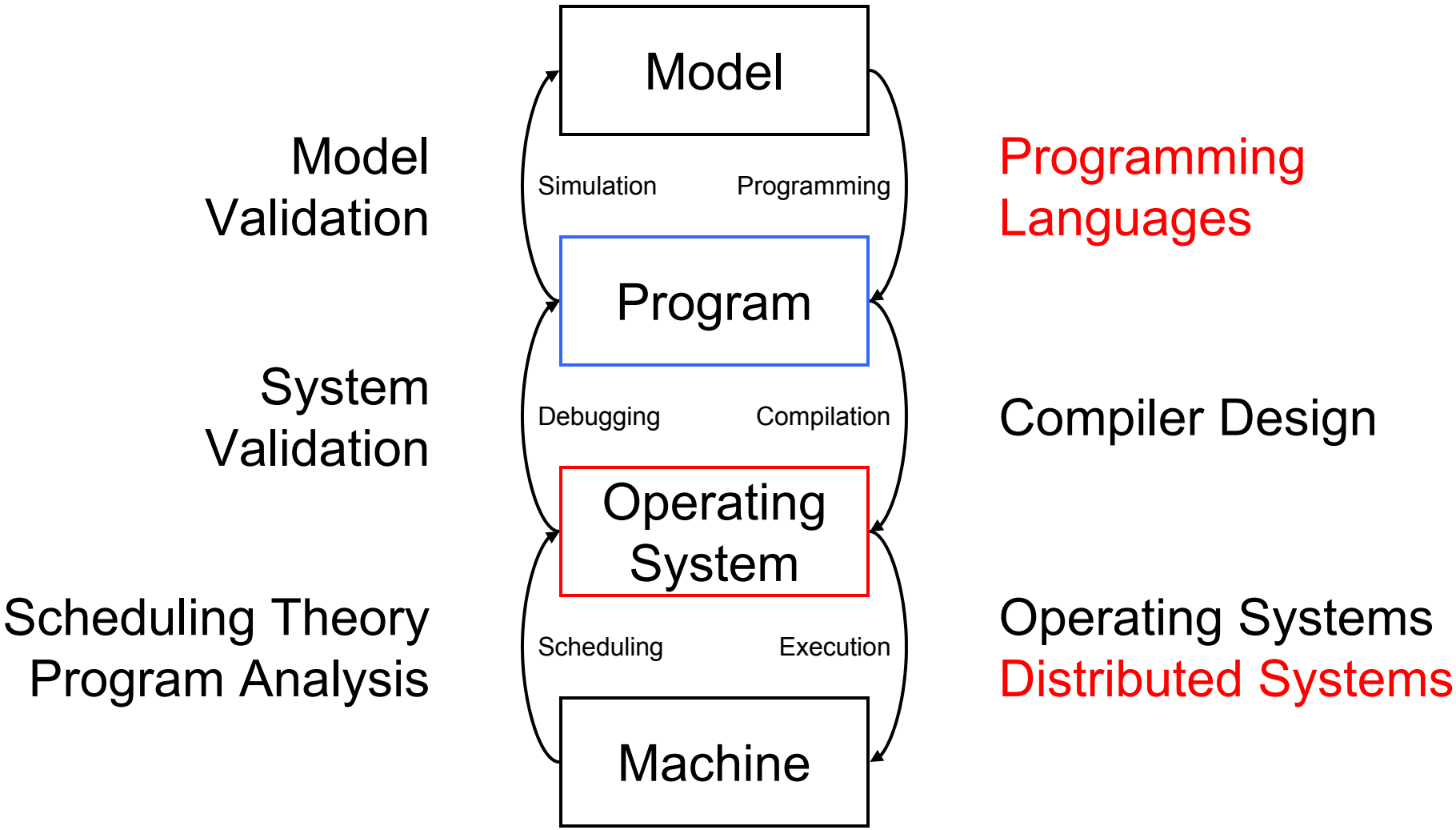


A Synchronous Implementation

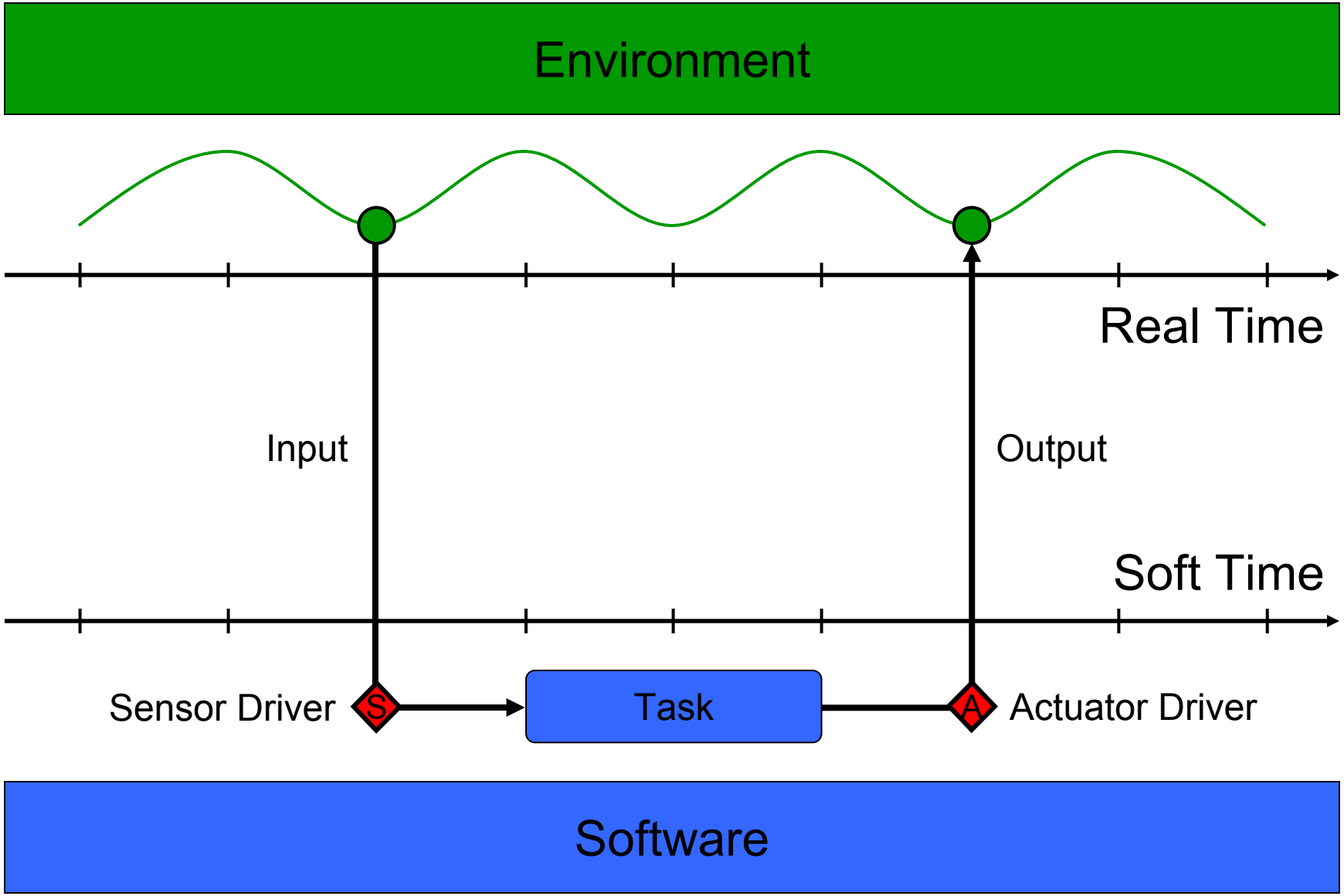


Synchronous but not Compositional

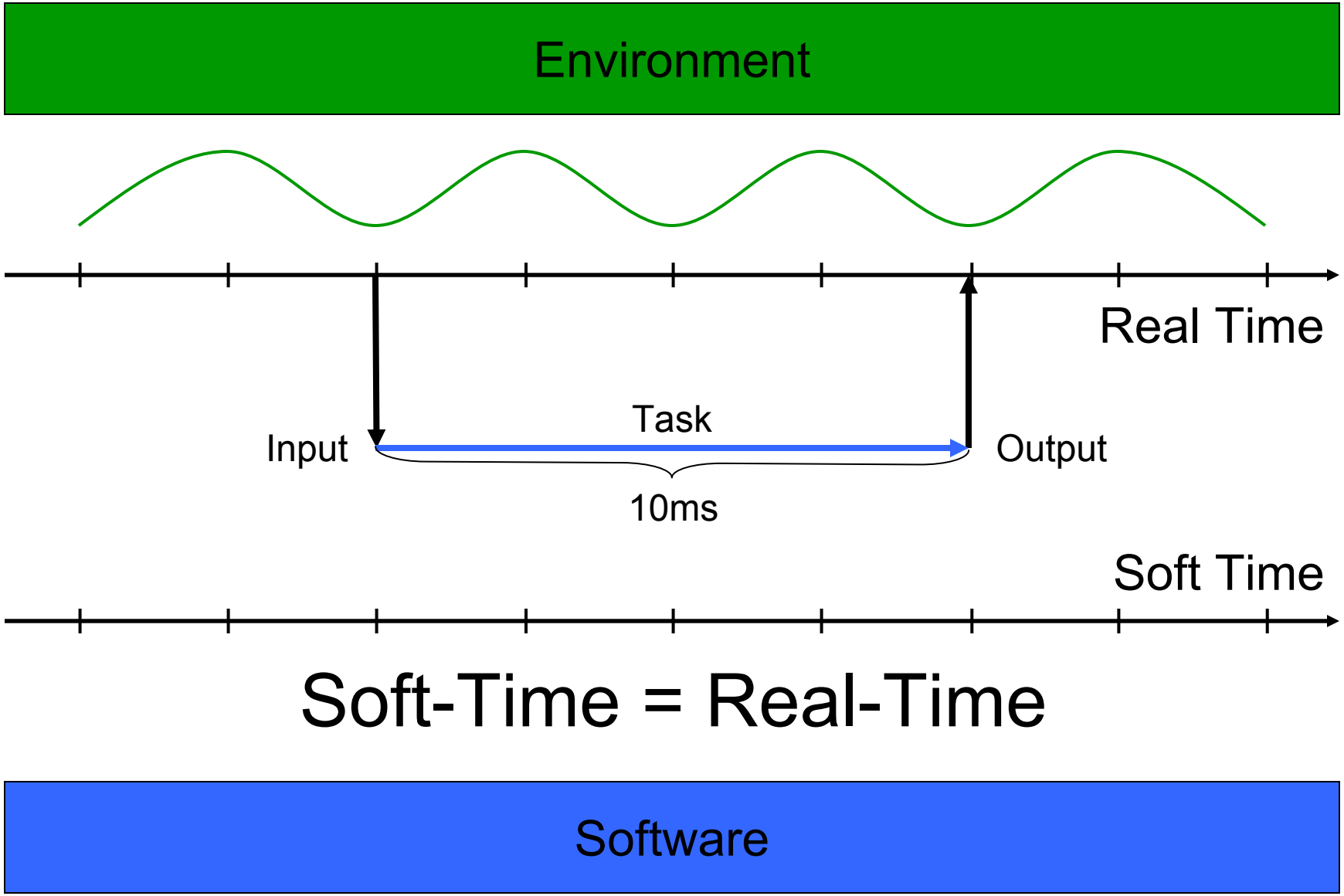
Embedded Programming Language: Giotto



Sensing, Computing, Actuating

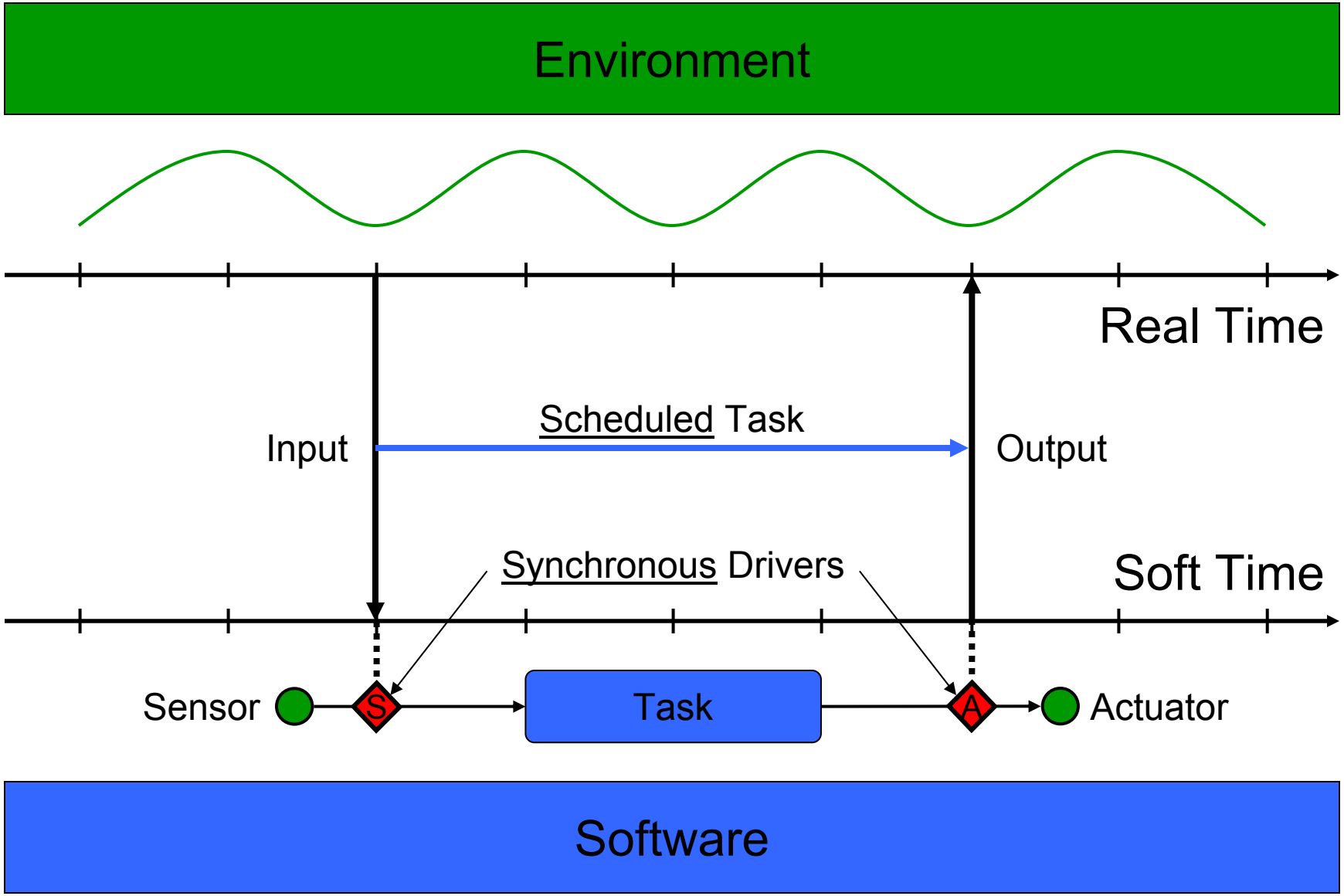


Giotto's Programming Abstraction

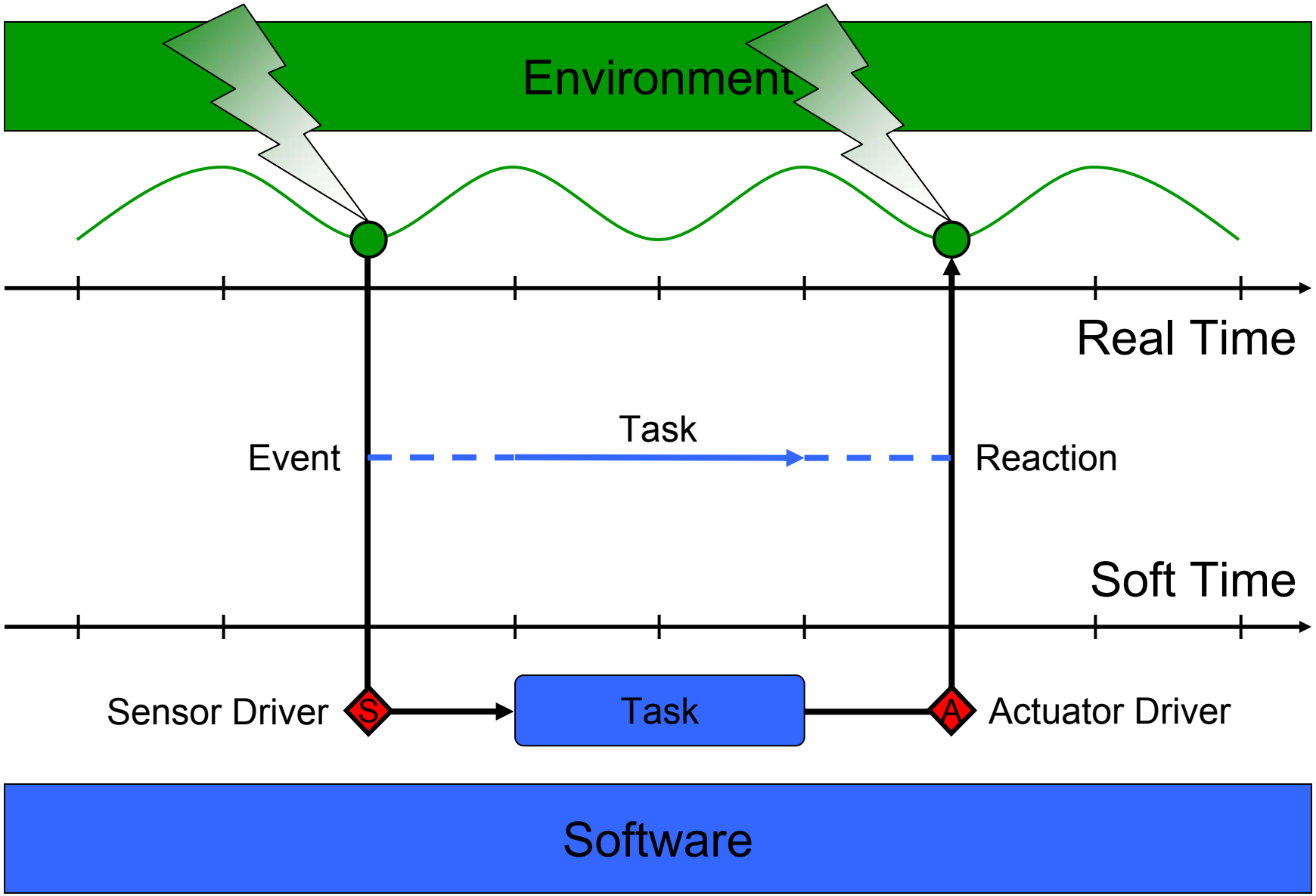


Soft-Time = Real-Time

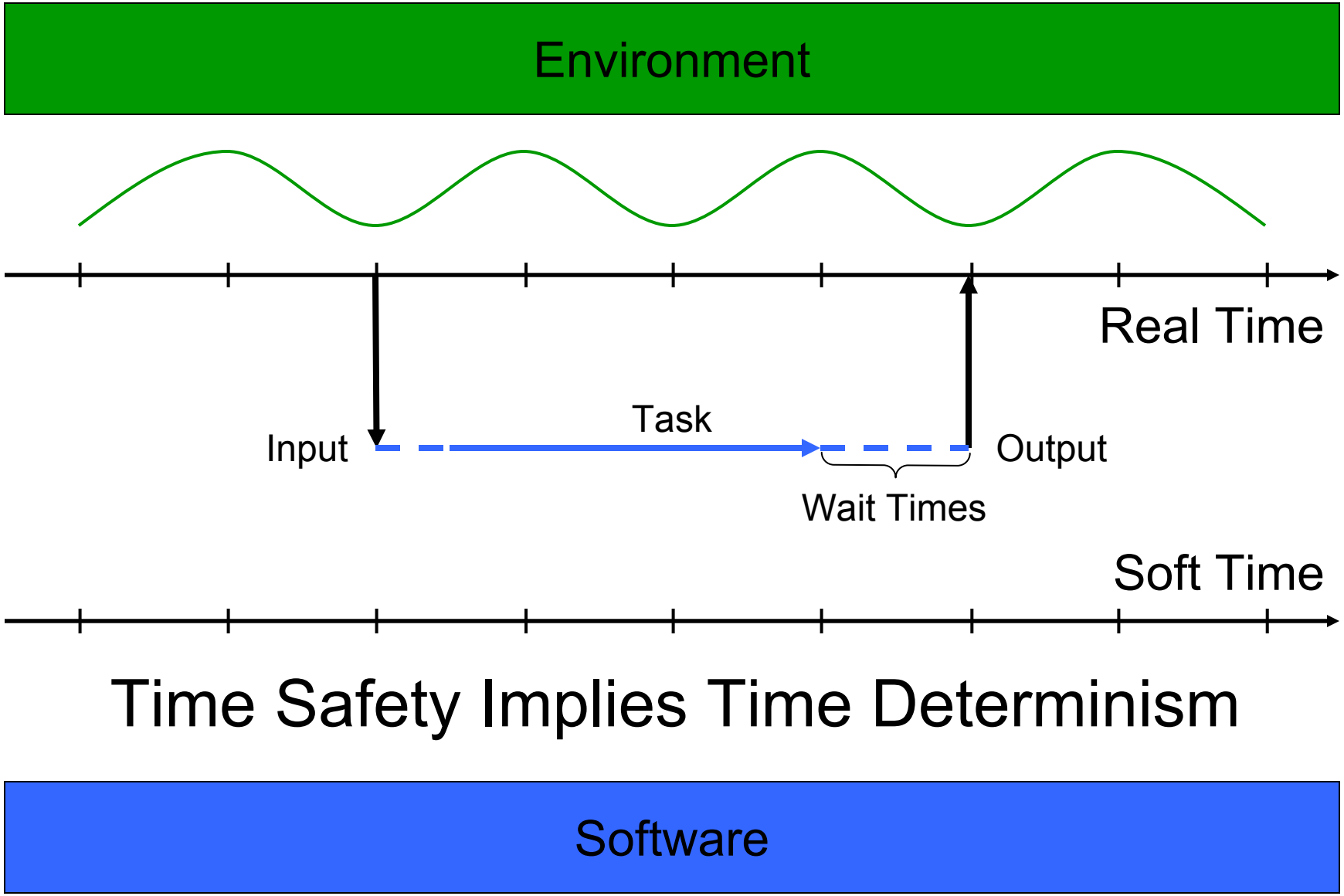
Synchronous vs. Scheduled Computation



Environment-triggered Programs

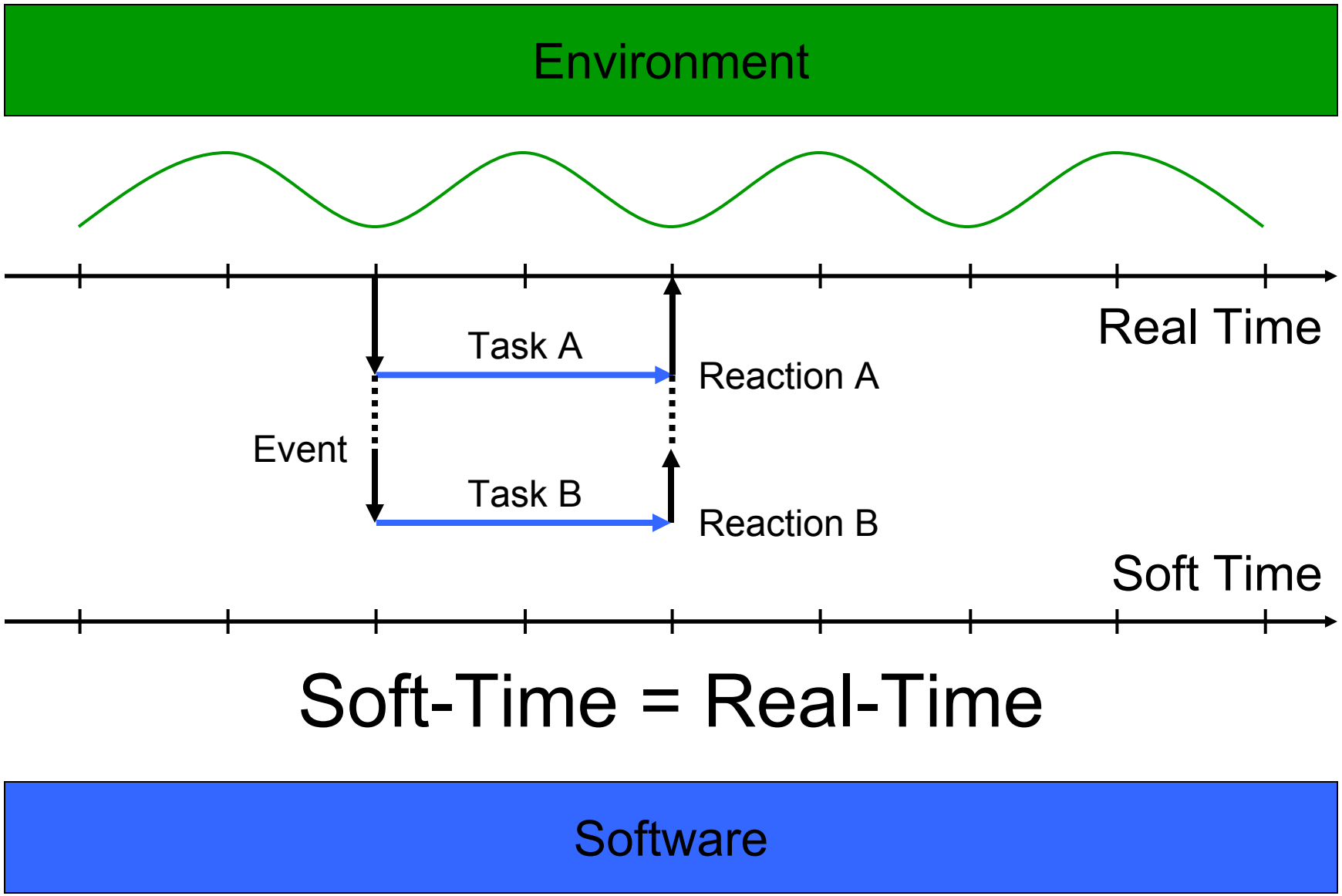


A Time-Safe Implementation



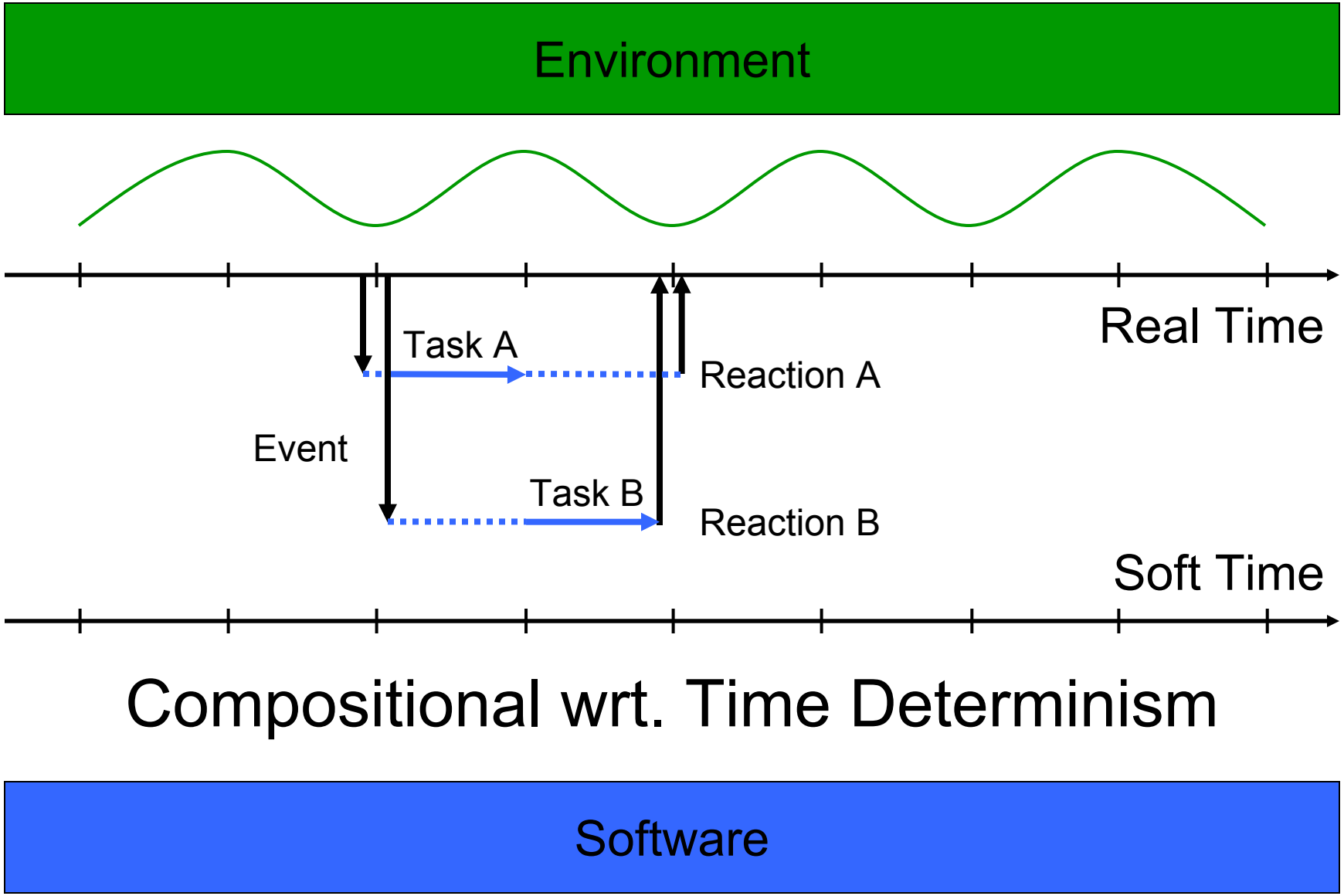
Time Safety Implies Time Determinism

Abstraction



Soft-Time = Real-Time

Implementation



Compositional wrt. Time Determinism

Giotto: A Time-triggered Language for Embd. Programming
(Henzinger, Horowitz, Kirsch in the Proceedings of the IEEE, Jan 2003)

Giotto is a time-triggered programming language that supports the development of embedded control systems

Time-safe Giotto programs are:

- *predictable* – deterministic real-time code
- *platform-independent* – runs on distributed systems
- *multi-modal* – supports mode switching
- *composable* – supports modular compilation

Giotto is available for:

- Linux, Windows, OSEKWorks, HelyOS
- Java

Giotto on the ETH Zürich Helicopter

(Kirsch, Sanvido, Henzinger, Pree in Proc. of EMSOFT 2002)

6 degrees of freedom, 1 processor (StrongARM 200Mhz)

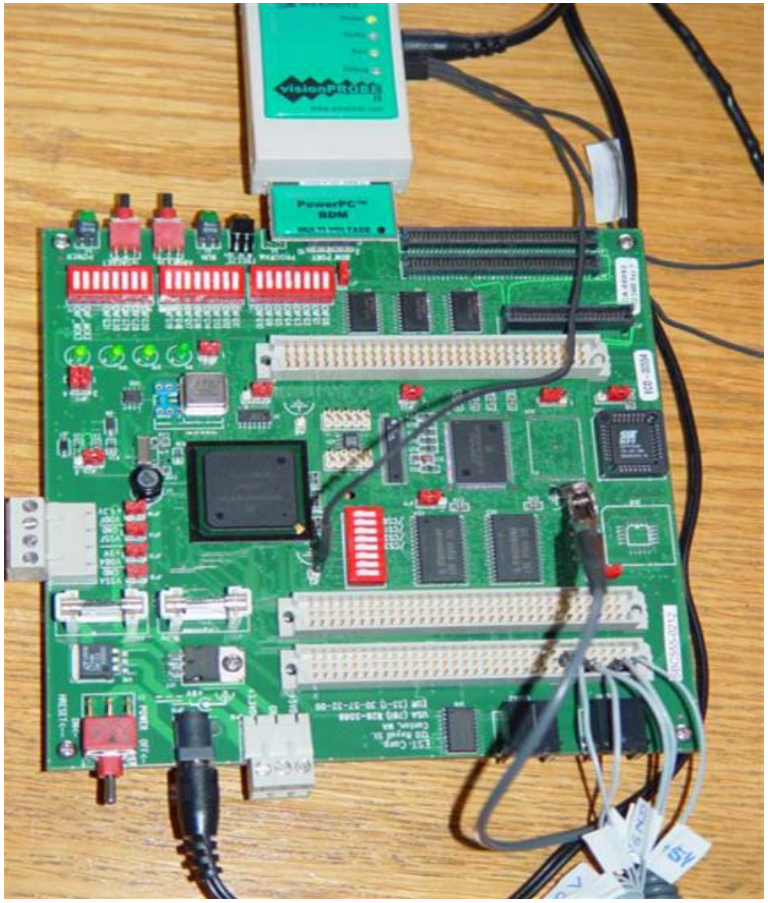
Giotto on the UC Berkeley Helicopter

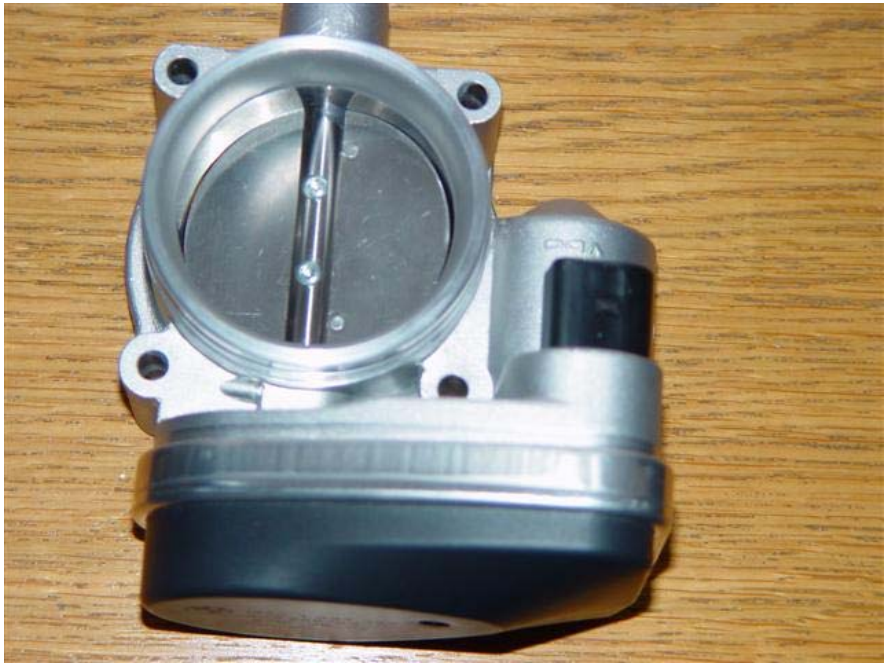
(Part of the DARPA SEC Project with Boeing and Honeywell)

6 degrees of freedom, 3 processors (Intel x86)

Giotto for a Drive-By-Wire BMW Throttle

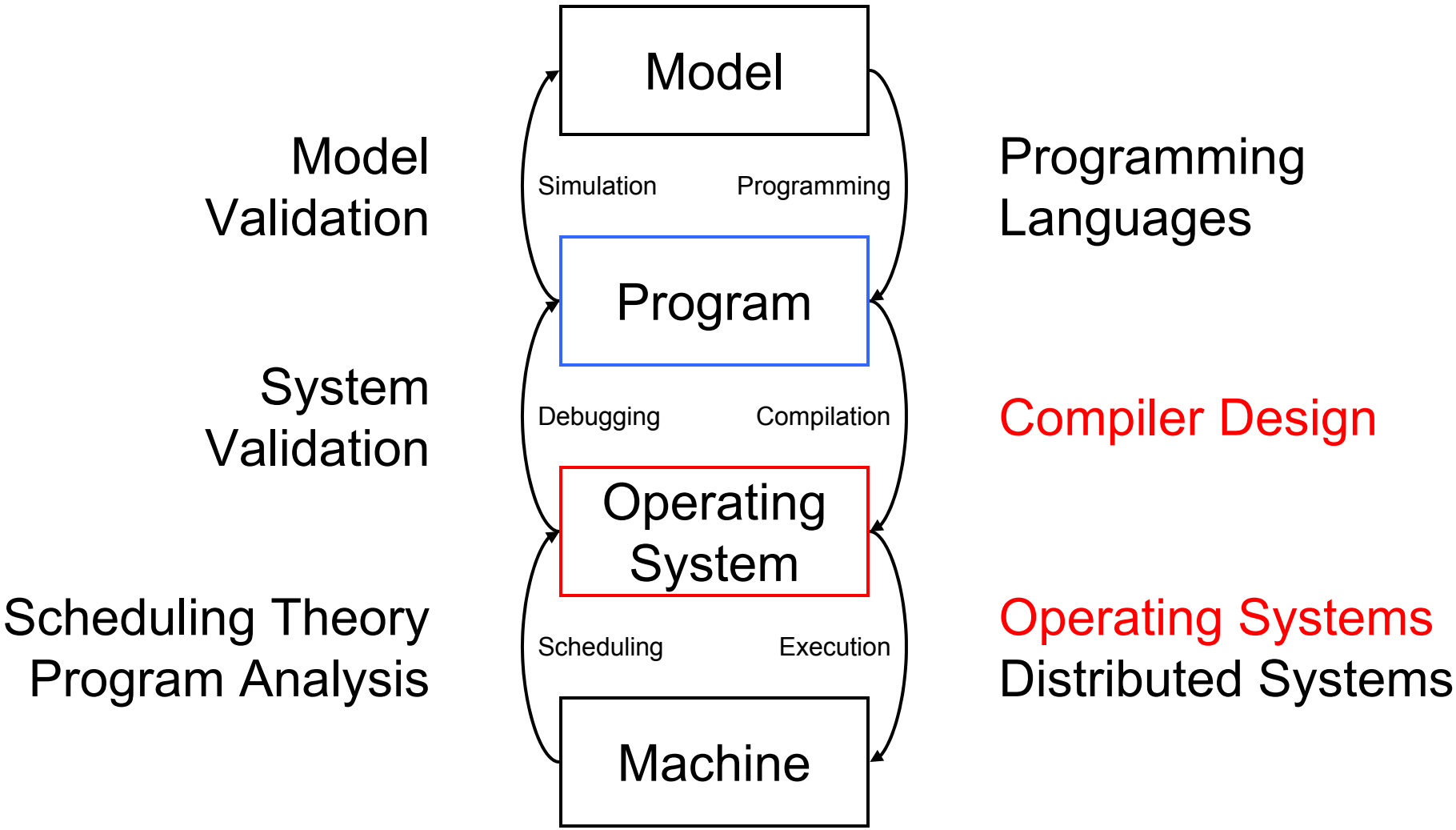
(Part of the DARPA MoBIES Project continued at Universität Salzburg)



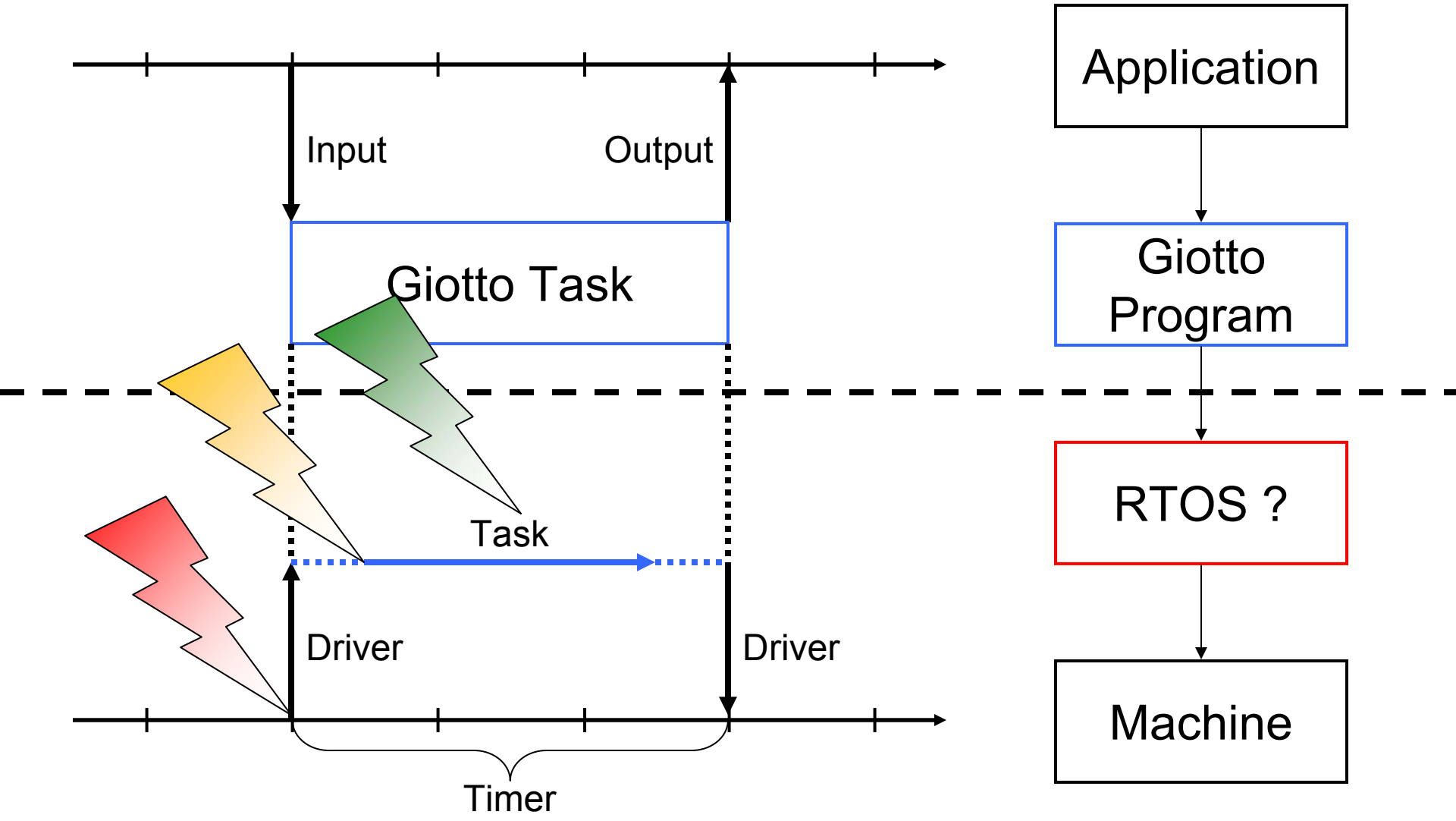


OSEKWorks RTOS, 1 processor (Motorola MPC555 40Mhz)

Operating Systems and Compiler Design: The Embedded Machine and E Code

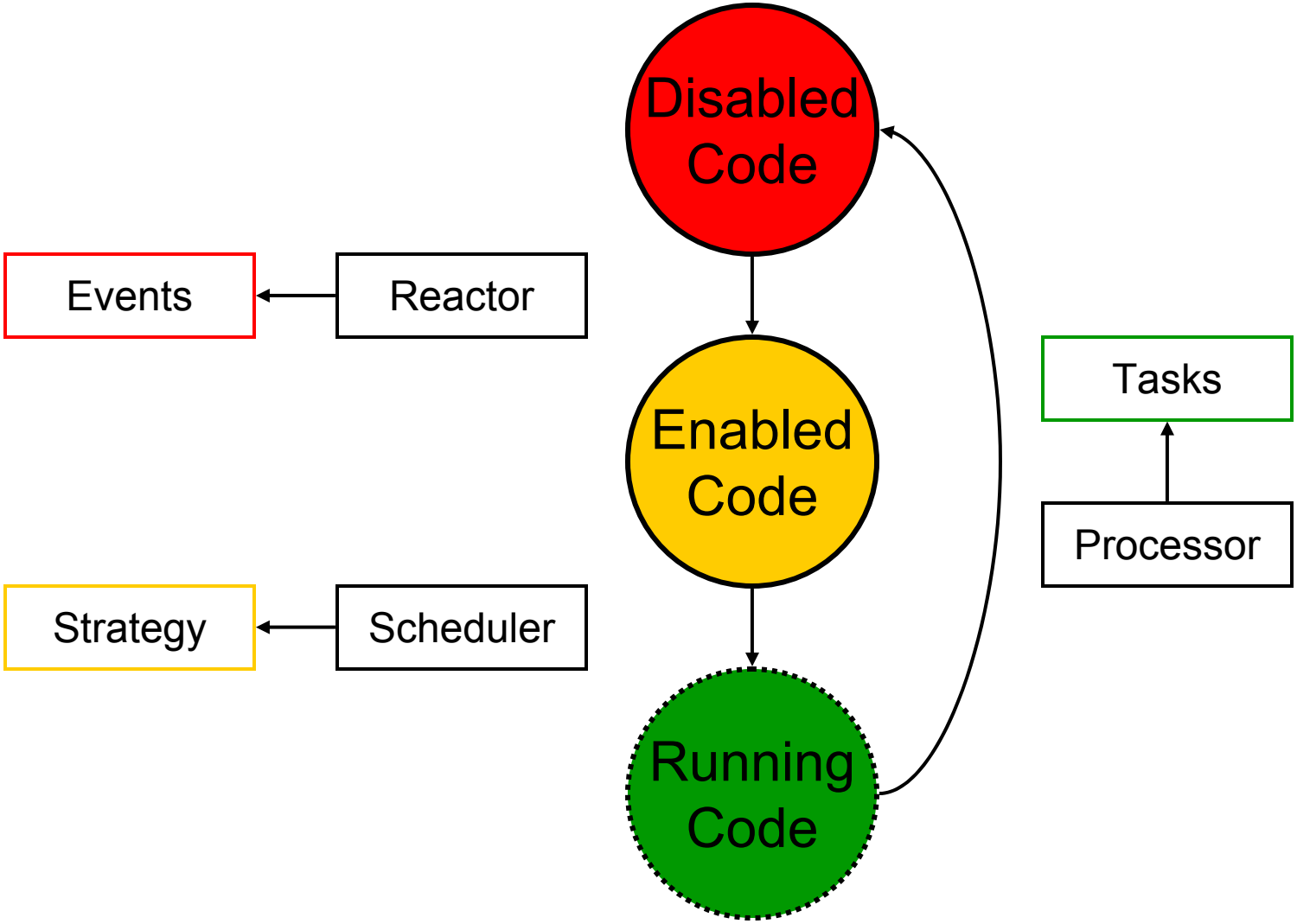


Triggering & Scheduling & Computing

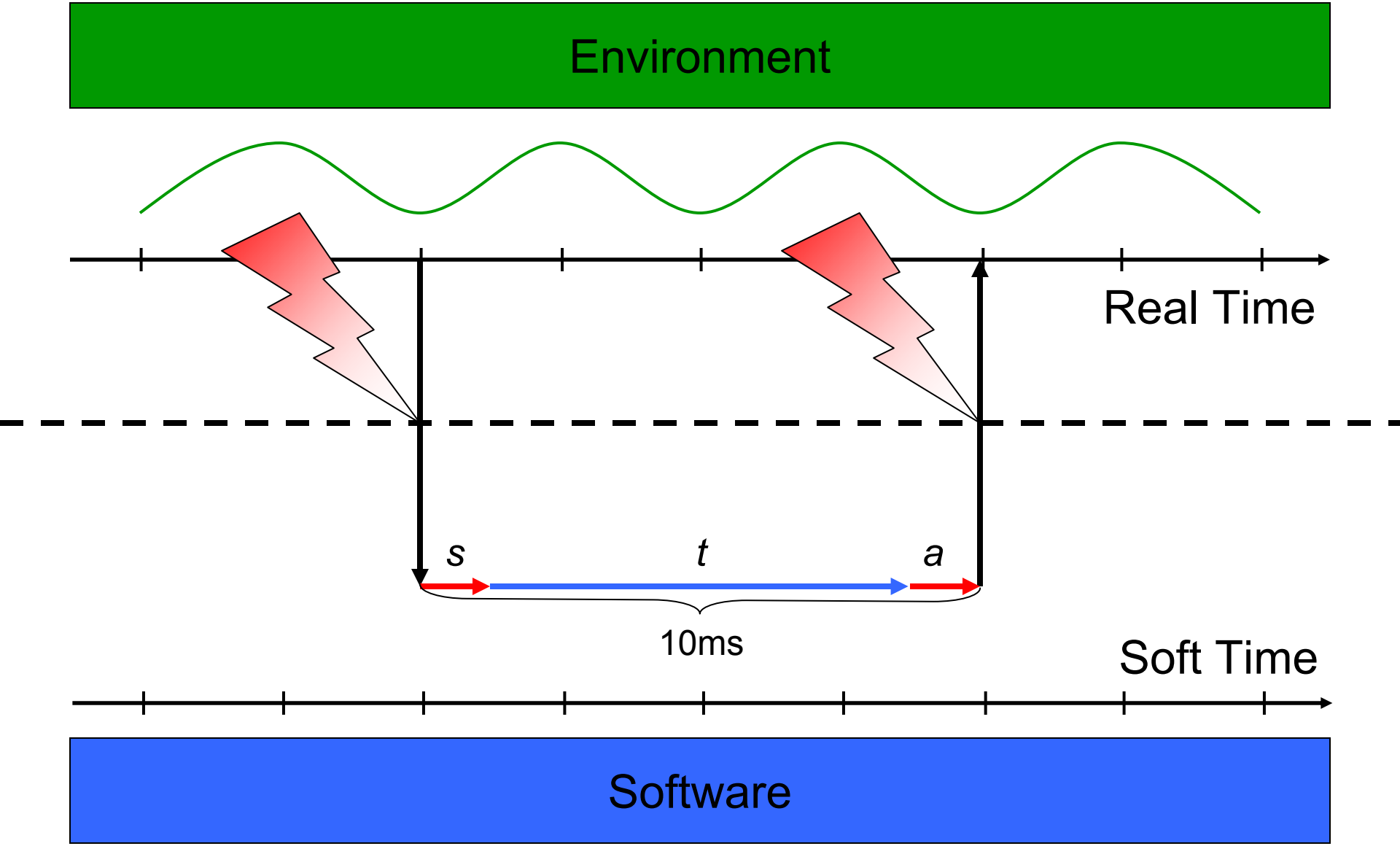


Reactor vs. Scheduler vs. Processor

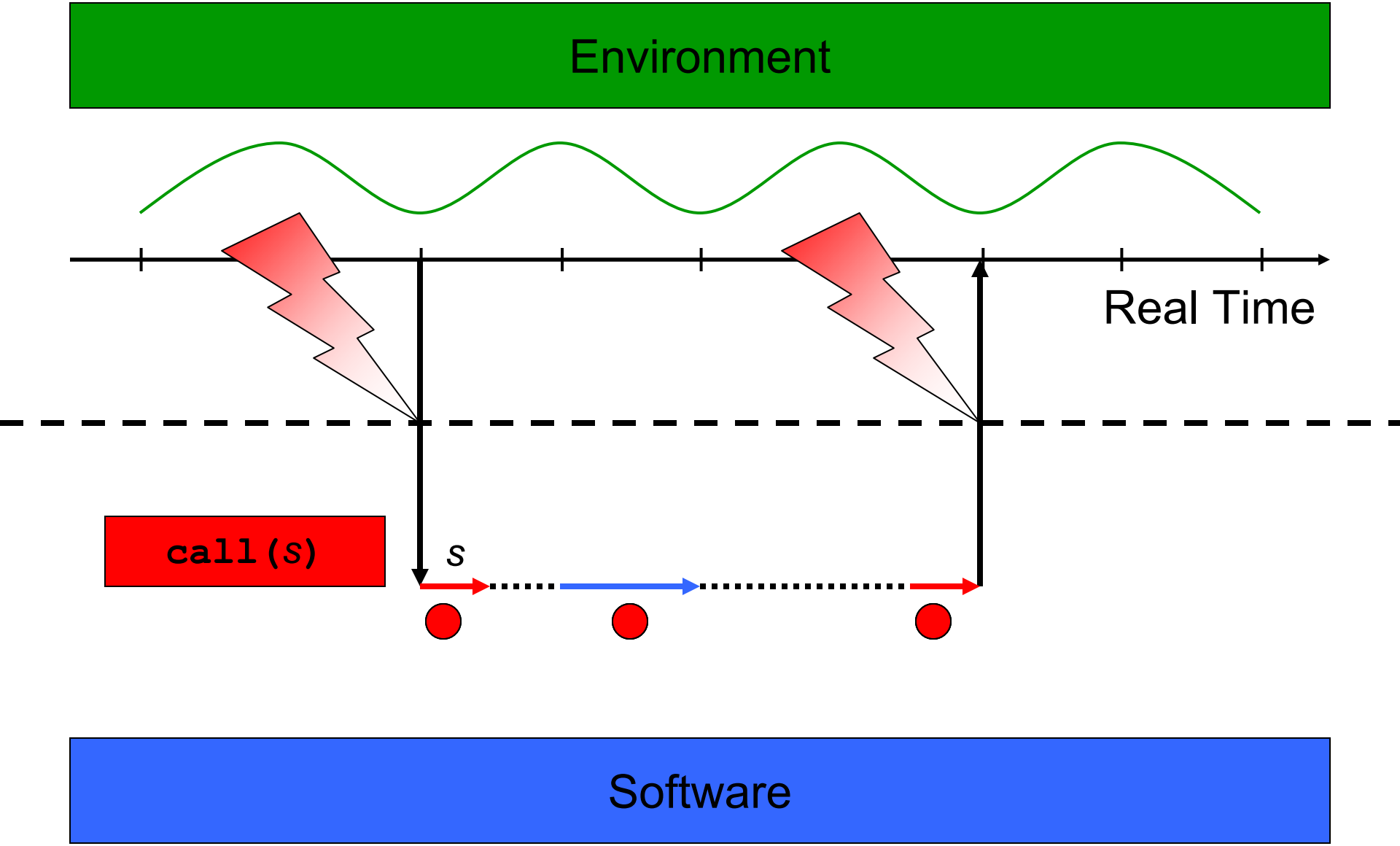
(Kirsch in the Proceedings of EMSOFT 2002)



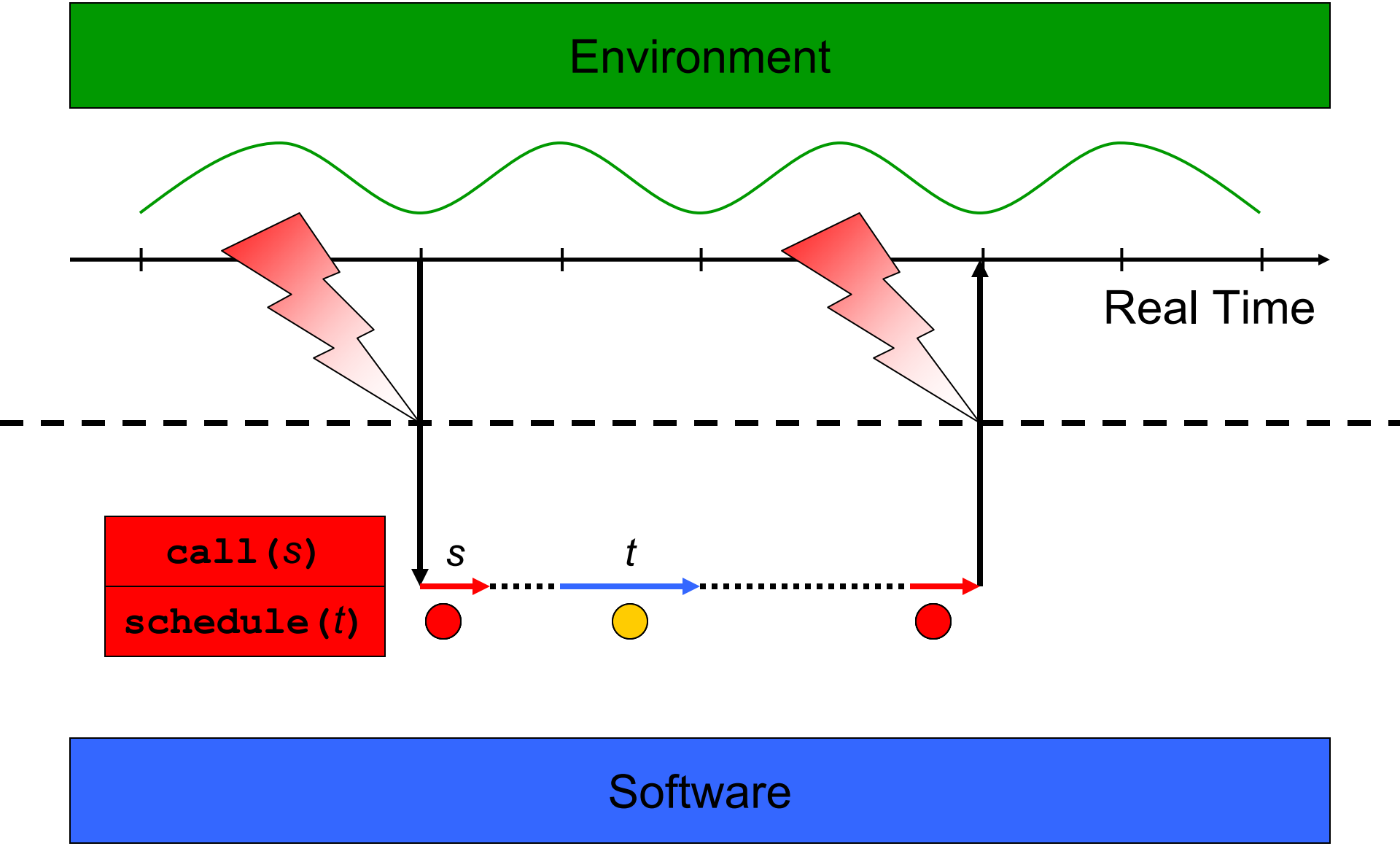
Sensing, Computing, Actuating



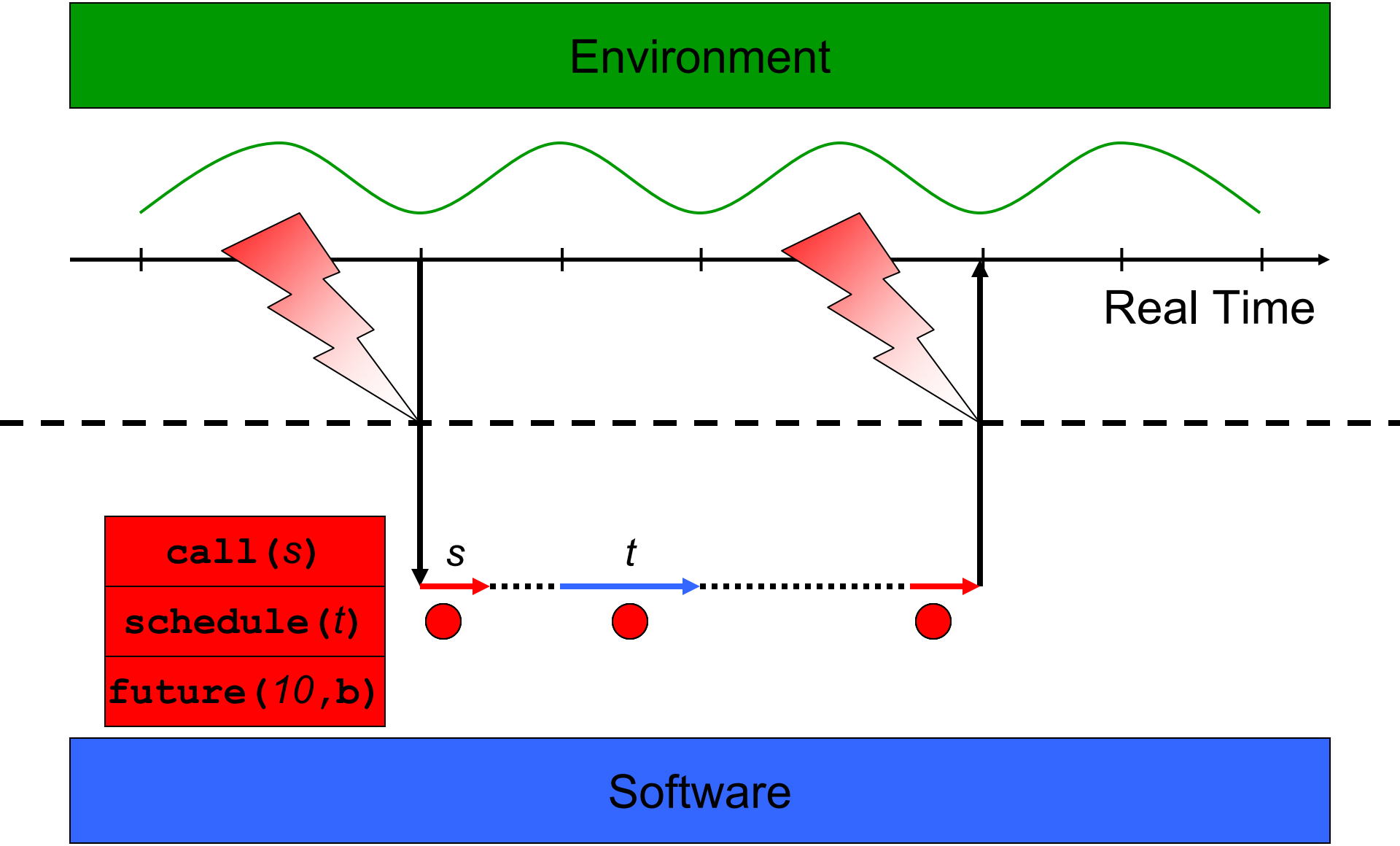
E Code



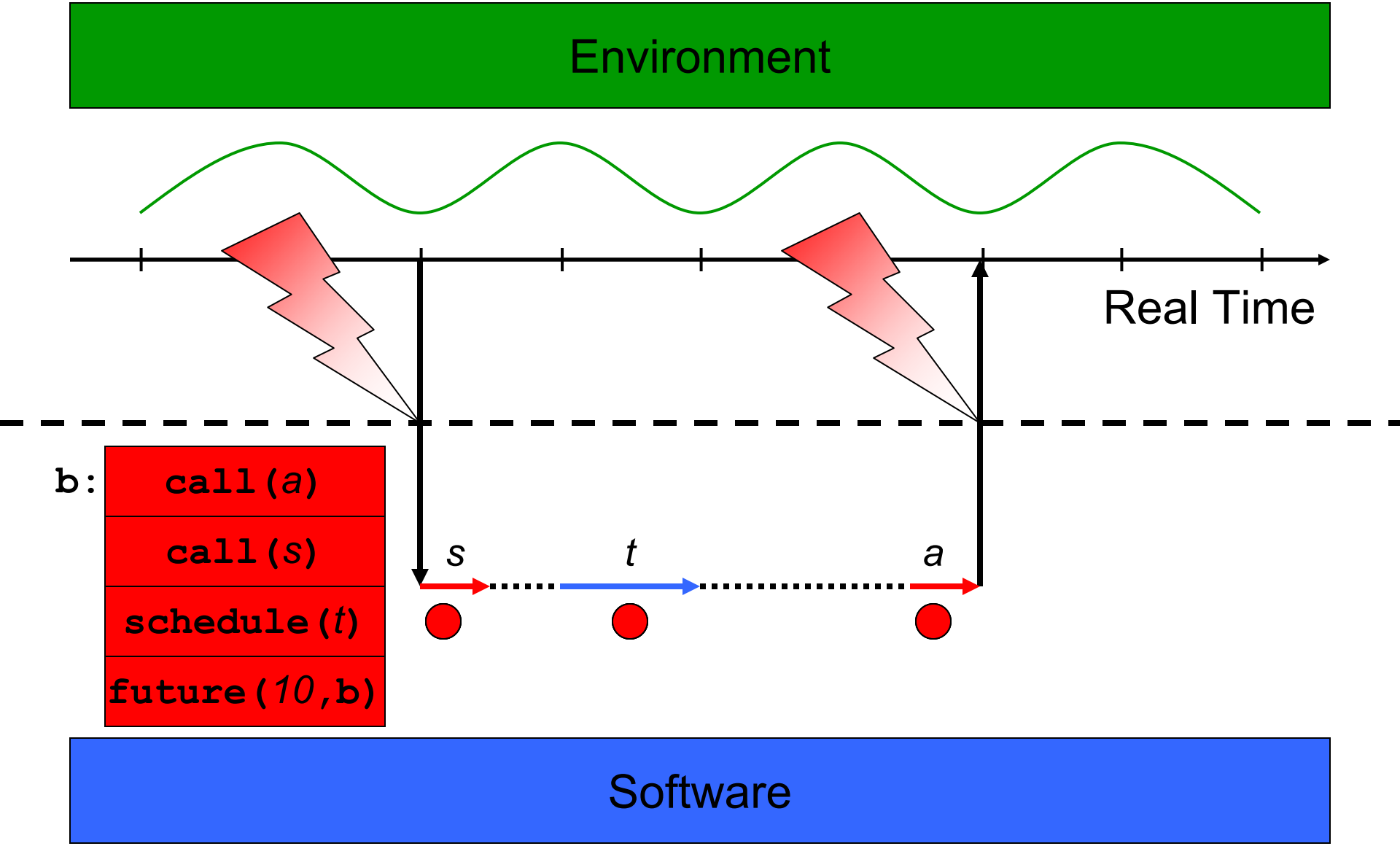
E Code



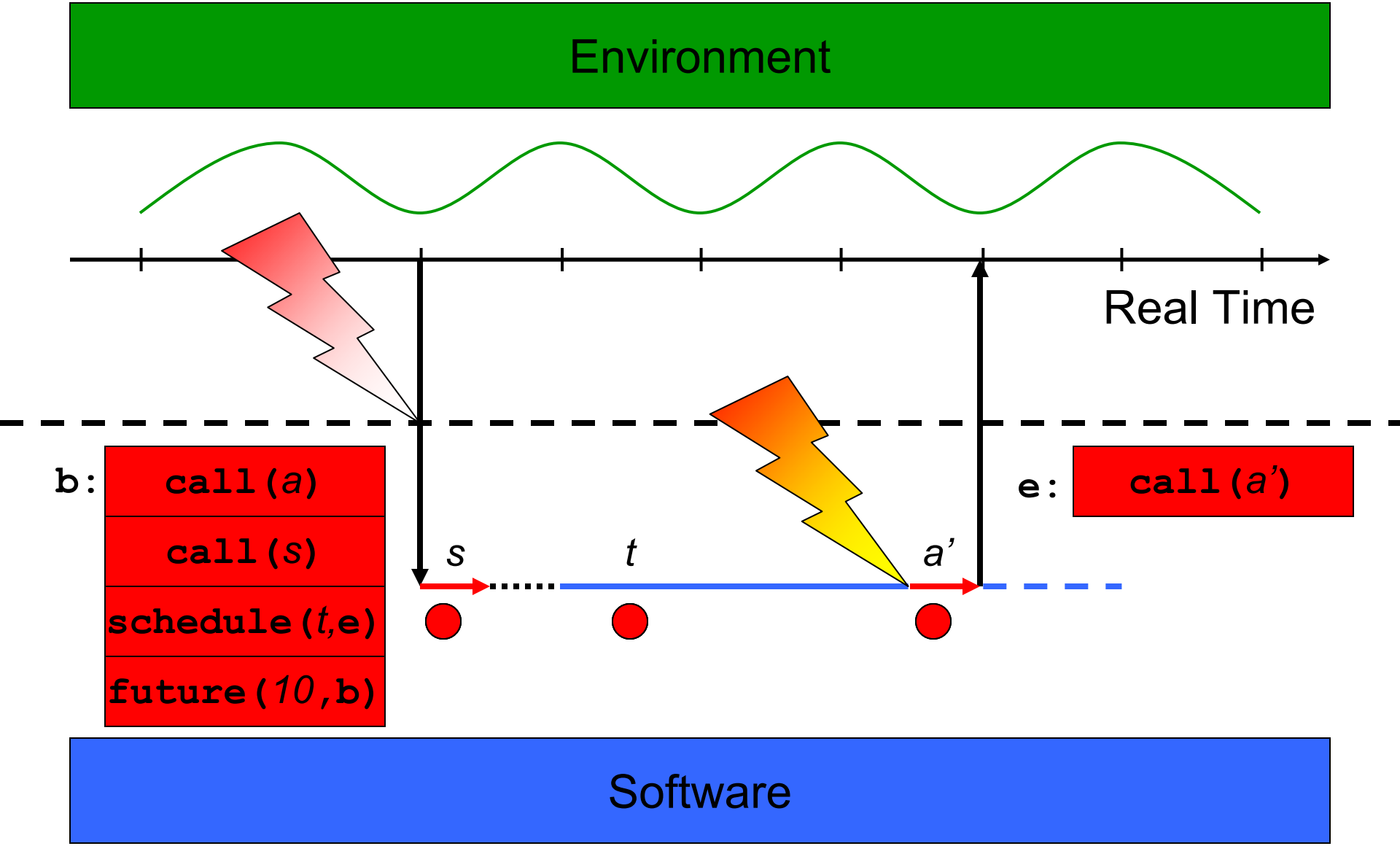
E Code


Environment
Real Time
call(s)
schedule(t)
future(10,b)
s
t
Software

E Code


Environment
Real Time
b:
call(a)
call(s)
schedule(t)
future(10,b)
s
t
a
Software

Time Safety or Runtime Exception



The Embedded Machine

(Henzinger, Kirsch in the Proceedings of PLDI 2002)

The Embedded Machine is a virtual machine that triggers the execution of software tasks wrt. events

Time-safe, environment-triggered E code is:

- *portable* – mobile real-time code
- *predictable* – deterministic real-time code
- *composable* – supports:
    - modular/incremental compilation
    - dynamic linking/patching

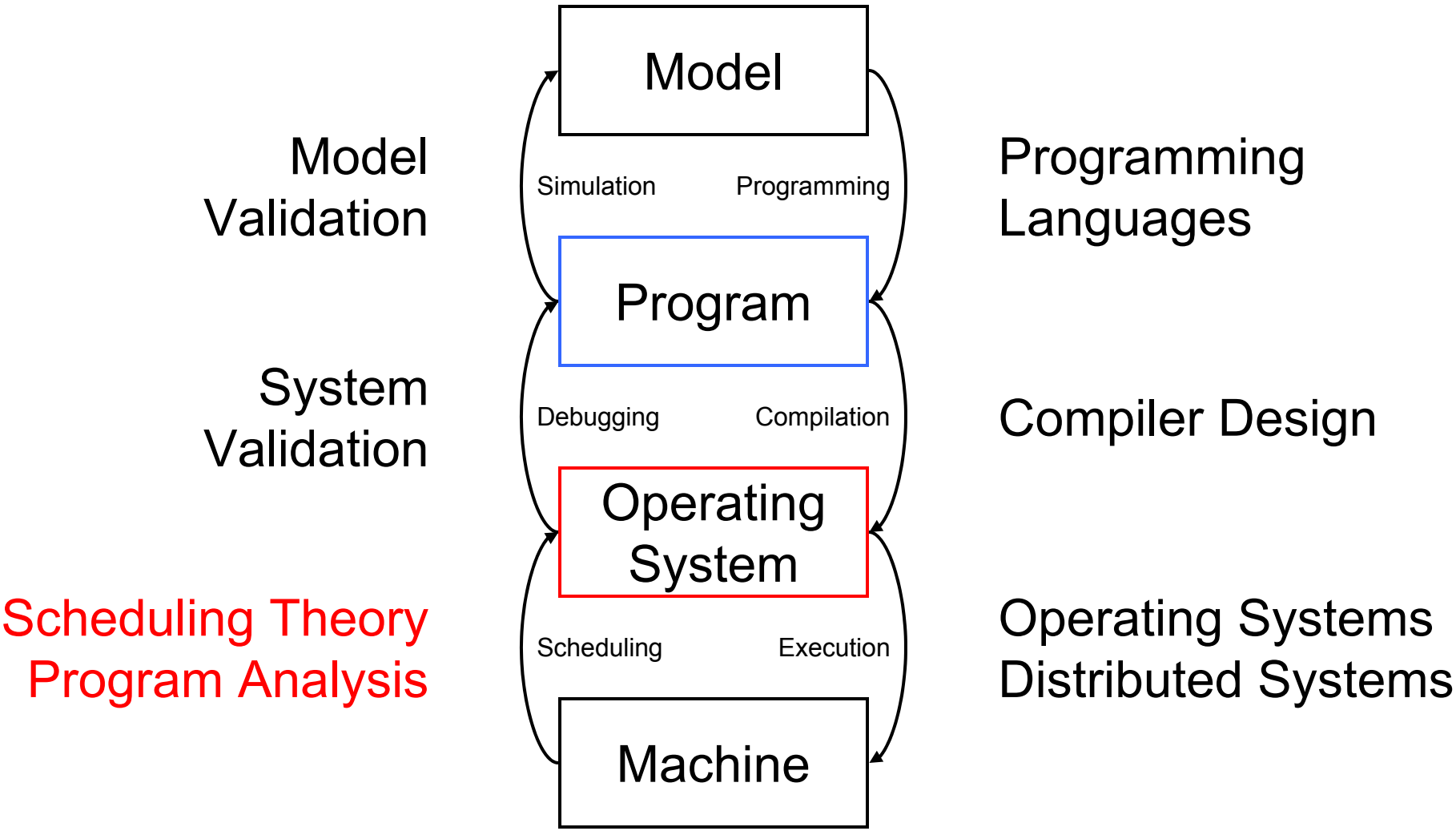
The Embedded Machine is available for:

- Linux, Windows, OSEKWorks, HelyOS
- Java (incl. E code debugger)

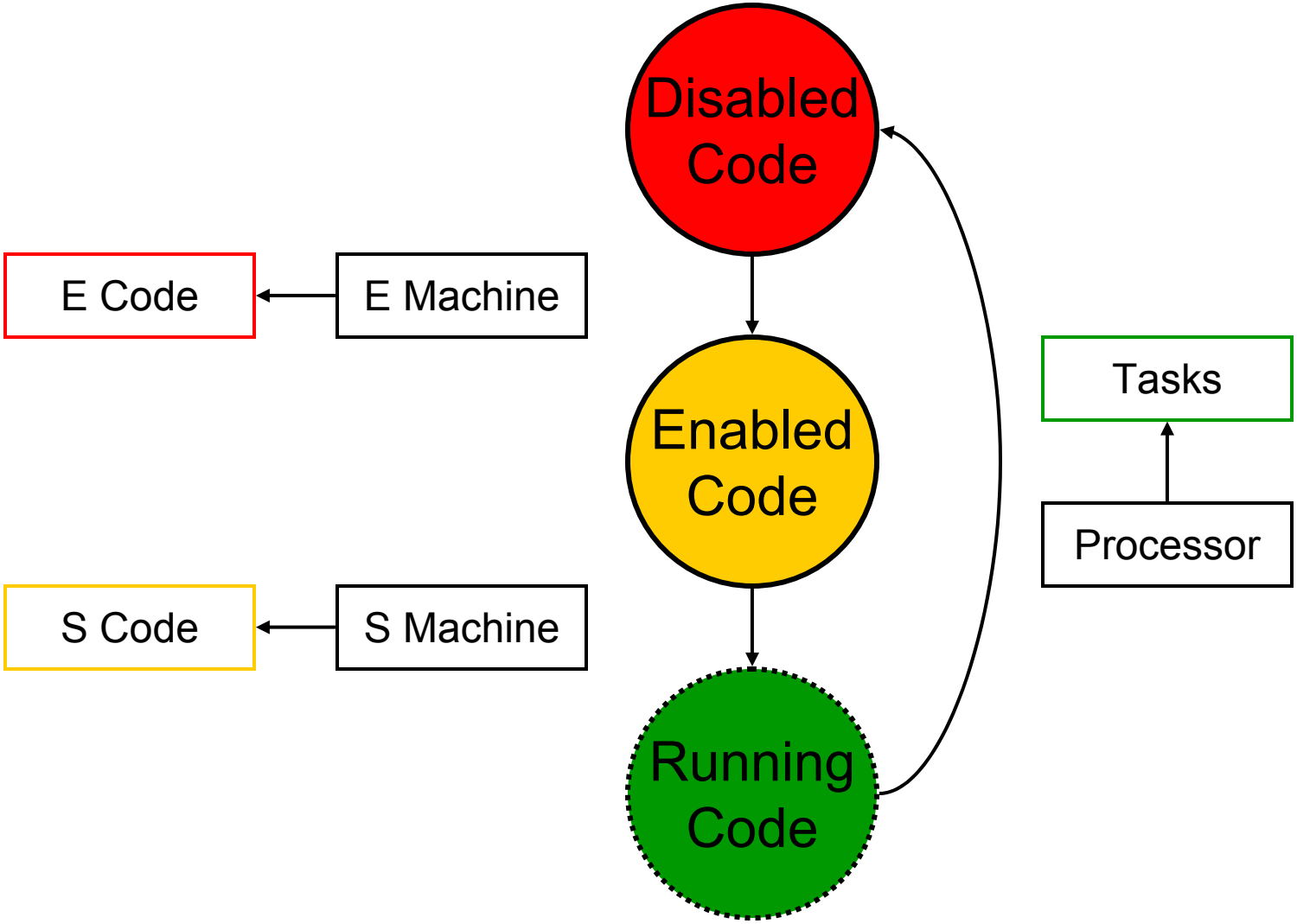
Distributed Embedded Machines

(Part of CHESS, $13m NSF Project at UC Berkeley)

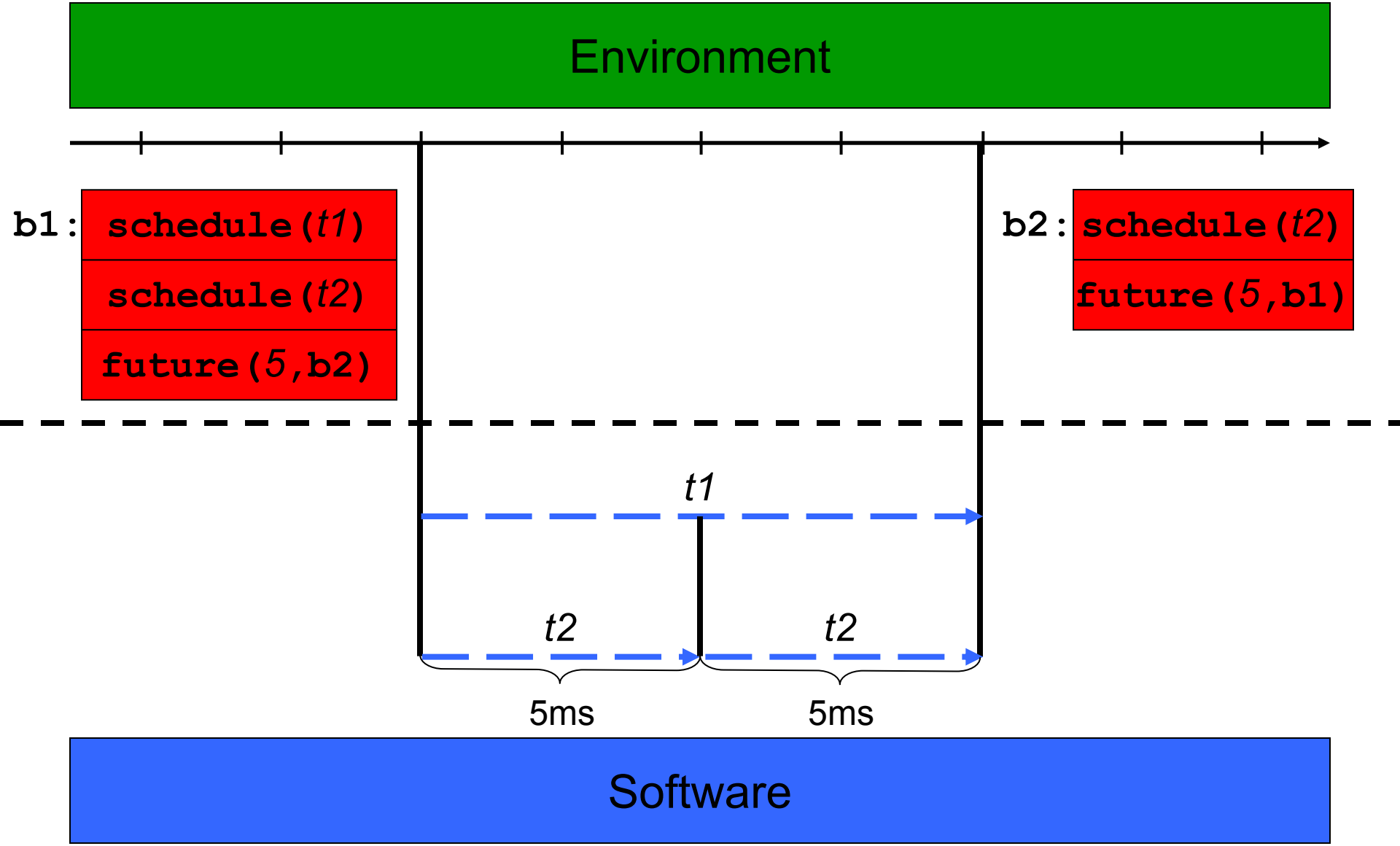
Scheduling Theory and Program Analysis: Schedule-Carrying Code (SCC)



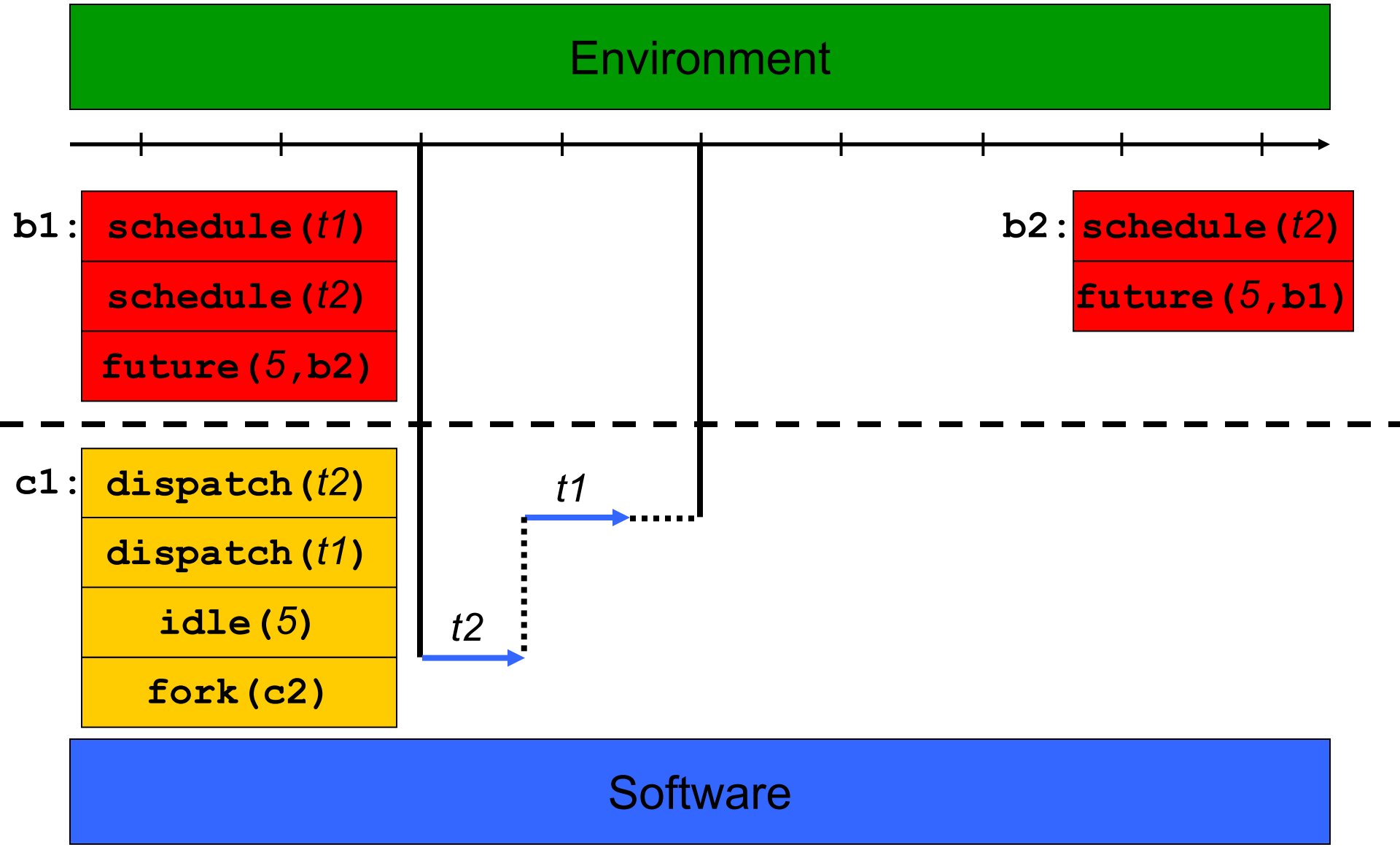
Triggering & Scheduling & Computing



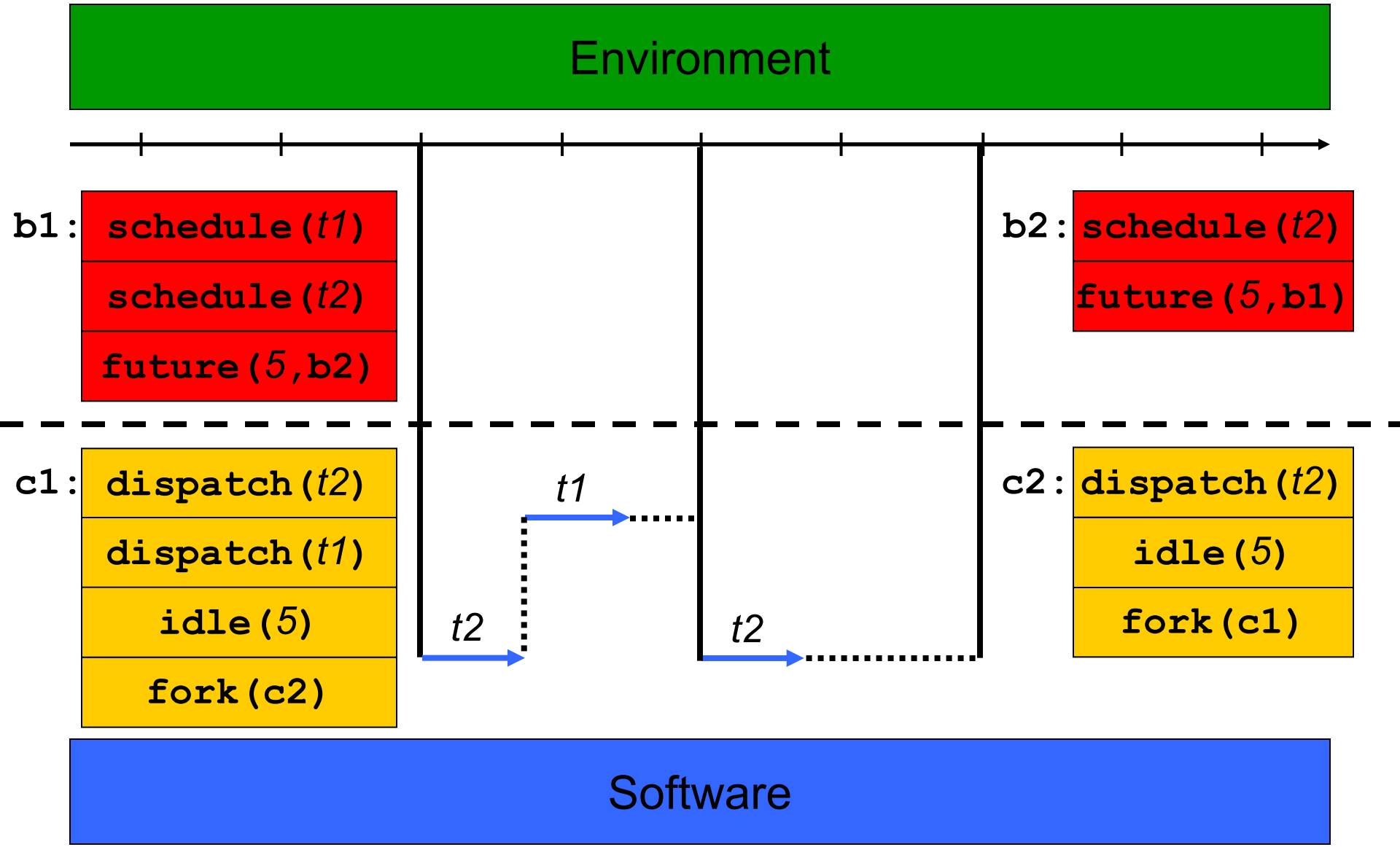
Two Tasks, Different Frequency



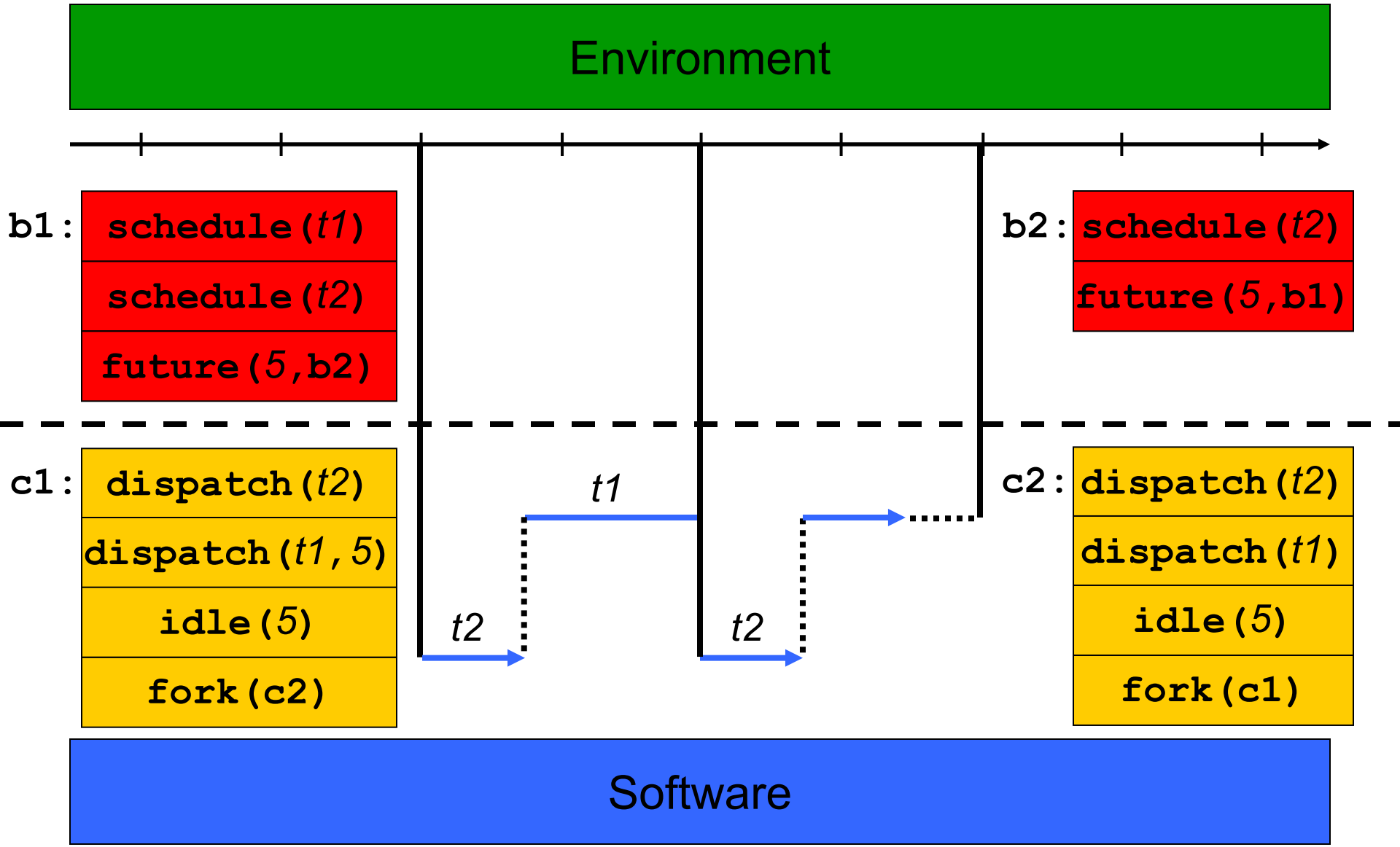
S Code


Environment
b1:
schedule(t1)
schedule(t2)
future(5,b2)
b2:
schedule(t2)
future(5,b1)
c1:
dispatch(t2)
dispatch(t1)
idle(5)
fork(c2)
t1
t2
Software

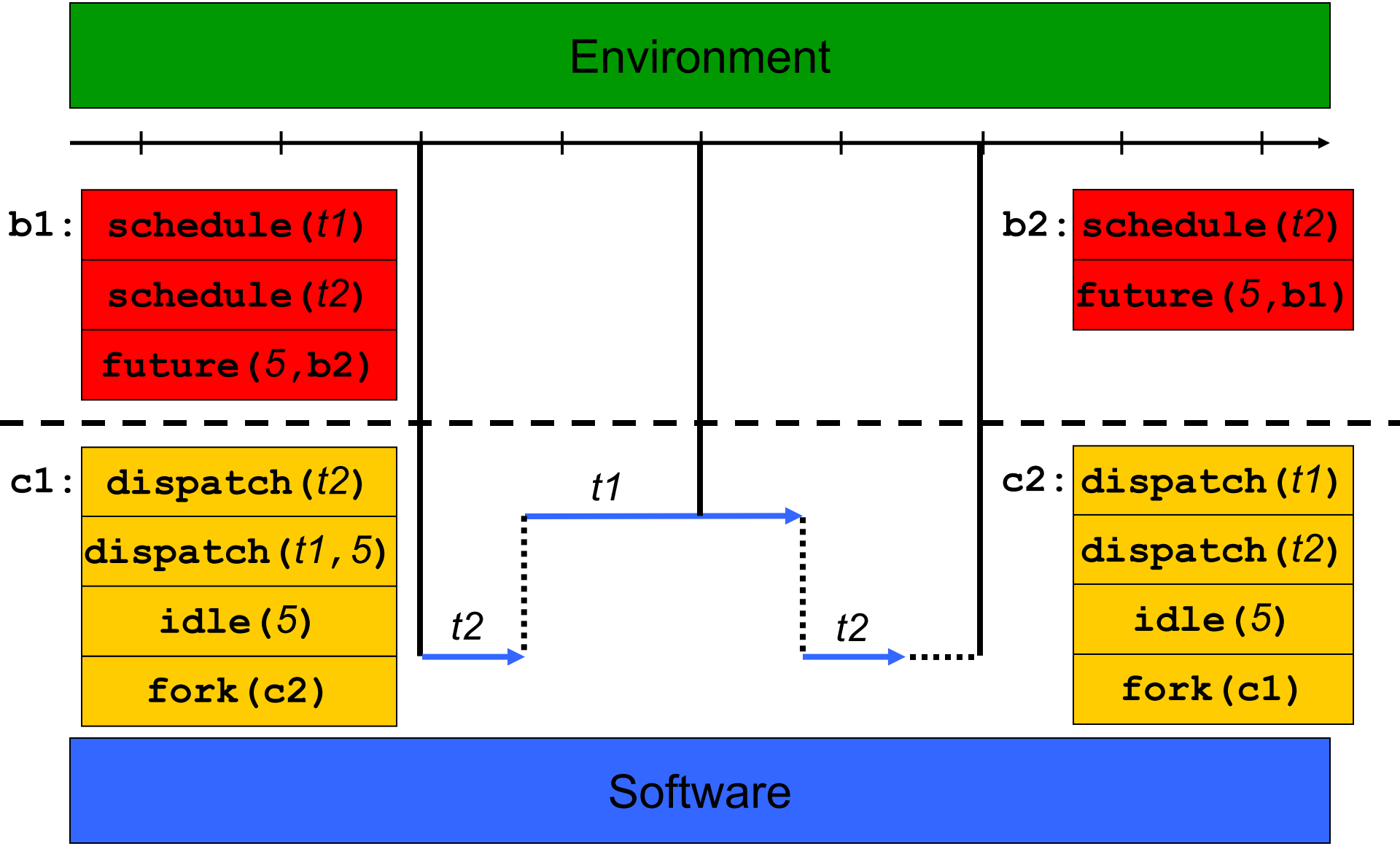
S Code


Environment
b1:
schedule(t1)
schedule(t2)
future(5,b2)
b2:
schedule(t2)
future(5,b1)
c1:
dispatch(t2)
dispatch(t1)
idle(5)
fork(c2)
t1
t2
t2
c2:
dispatch(t2)
idle(5)
fork(c1)
Software

Preemption


Environment
b1:
schedule(t1)
schedule(t2)
future(5,b2)
b2:
schedule(t2)
future(5,b1)
c1:
dispatch(t2)
dispatch(t1,5)
idle(5)
fork(c2)
t1
t2
t2
c2:
dispatch(t2)
dispatch(t1)
idle(5)
fork(c1)
Software

Non-Preemptive Scheduling



Time Safety

Environment

`schedule(`*t1*`)`
`schedule(`*t2*`)`
`future(`*5*`,b2)`

Checking time safety of E code (set of periodic tasks) with a non-preemptive scheduler is *NP-hard*

`schedule(`*t2*`)`
`future(`*5*`,b1)`

Software

Schedule-Carrying Code

(Henzinger, Kirsch, Matic, in Proc. of EMSOFT 2003)

Environment

`schedule(`*t1*`)`
`schedule(`*t2*`)`
`future(`*5*`,b2)`

Checking time safety of E code (set of periodic tasks) with a non-preemptive scheduler is *NP-hard*

`schedule(`*t2*`)`
`future(`*5*`,b1)`

`dispatch(`*t2*`)`
`dispatch(`*t1*`,`*5*`)`
`idle(`*5*`)`
`fork(c2)`

Checking time safety of E code (set of periodic tasks) + non-preemptive S code is *linear* (in E code size)

`dispatch(`*t1*`)`
`dispatch(`*t2*`)`
`idle(`*5*`)`
`fork(c1)`

Software

Time Safety Checking for Embedded Programs:

(Henzinger, Kirsch, Majumdar, Matic in Proc. of EMSOFT 2002)

Scheduling Constraints \ Triggering Complexity	Periodic Tasks	Giotto	?	Arbitrary
Multiprocessor	NP-Complete [Hardness: GJ79]	NP-Hard	NP-Hard	EXPTIME-Complete
Non-Preemptive	NP-Complete [Hardness: JSM91]	NP-Hard	NP-Hard	EXPTIME-Complete
Preemptive	Linear in E Code	Linear in E Code	?	EXPTIME-Complete

The Scheduling Machine

The Scheduling Machine is a virtual machine that orders the execution of software tasks

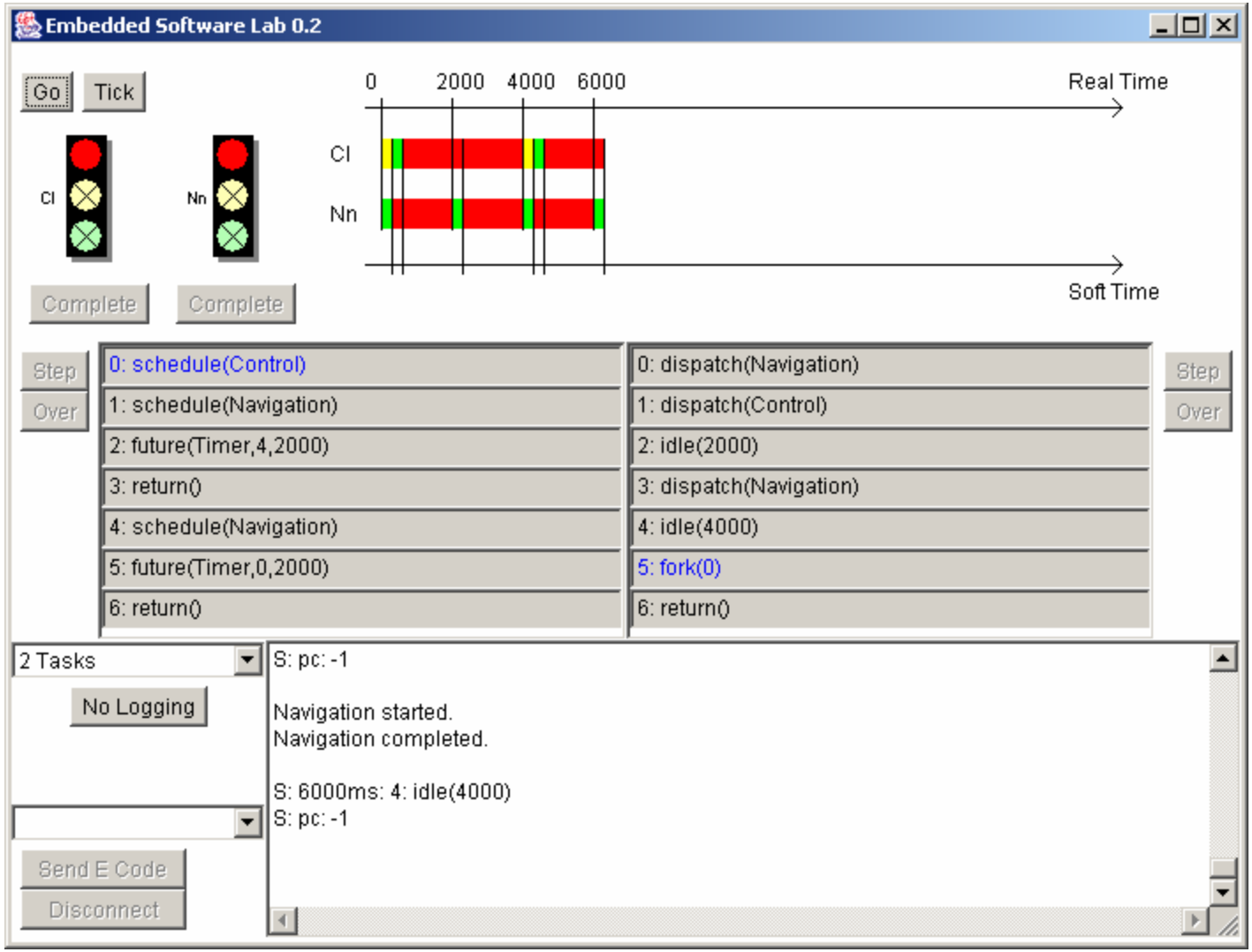
S code (+ *control-flow* instructions) is:

- *universal* – any scheduling strategy
- *verifiable* – fast time safety checking
- *distributed* – can schedule:
  - computation
  - communication

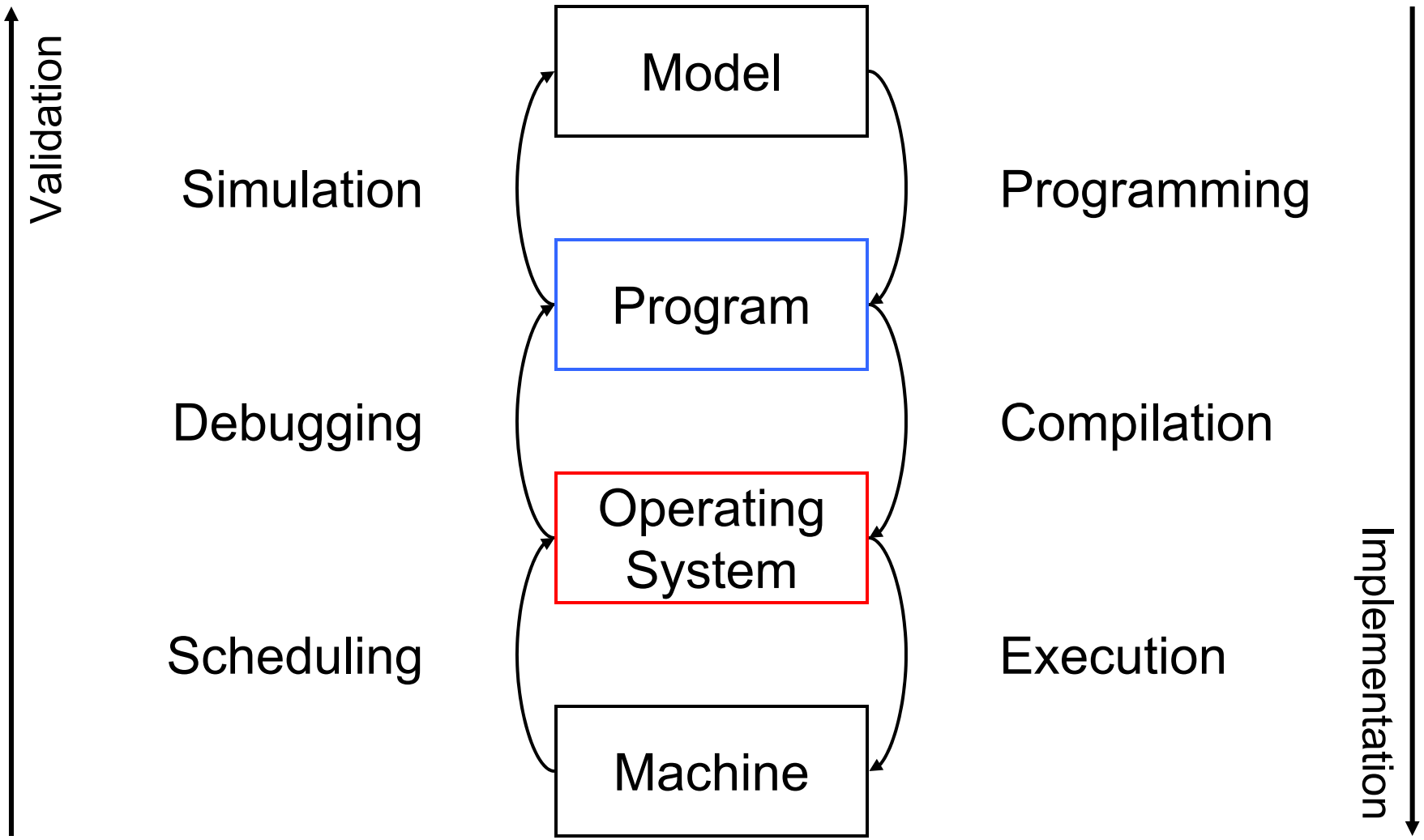
The Scheduling Machine is available for:

- Java (incl. S code debugger)

The E and S Machine in Class



Summary


Validation
Simulation
Debugging
Scheduling
Model
Program
Operating System
Machine
Programming
Compilation
Execution
Implementation