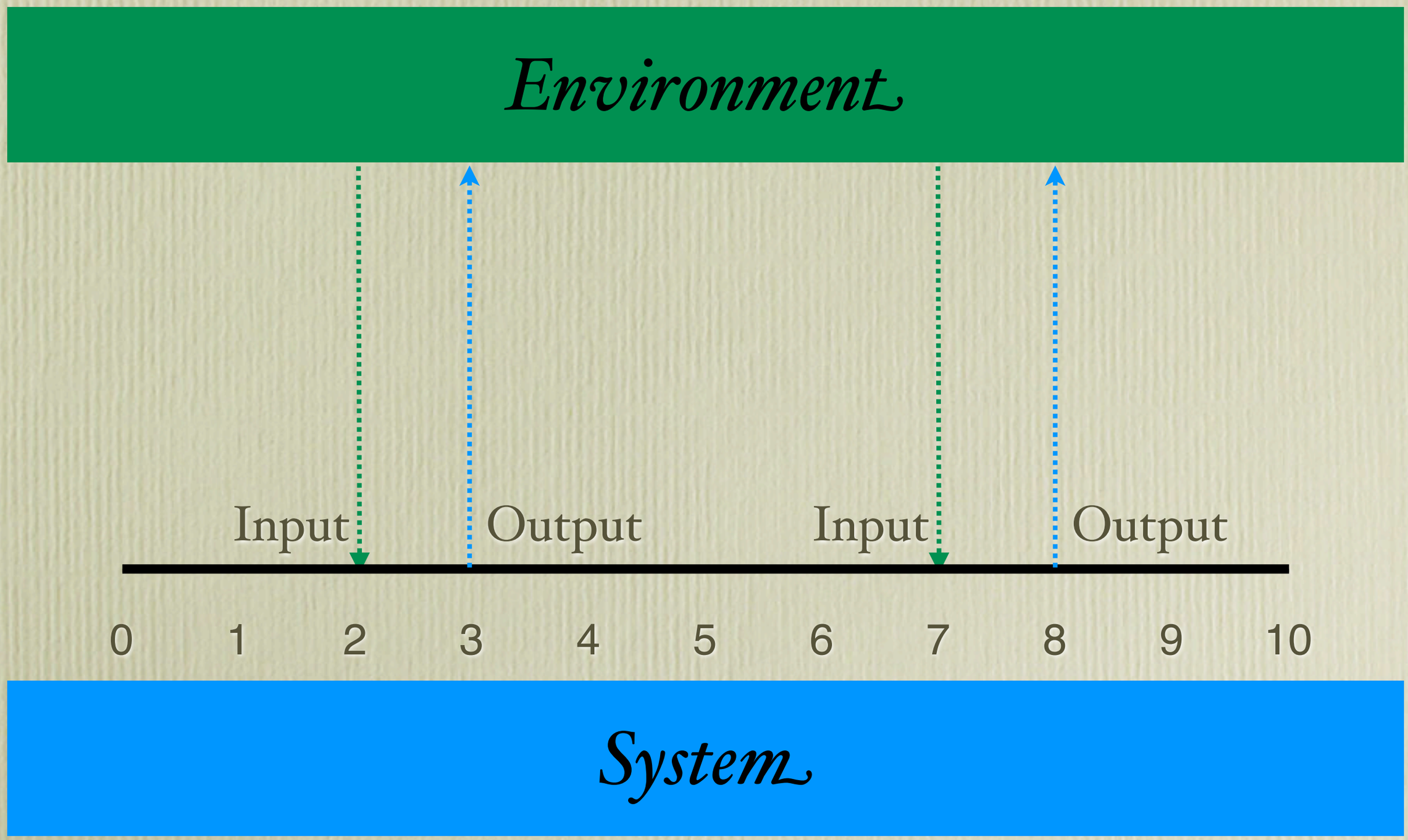
High-Level Programming of Real-Time Software Systems

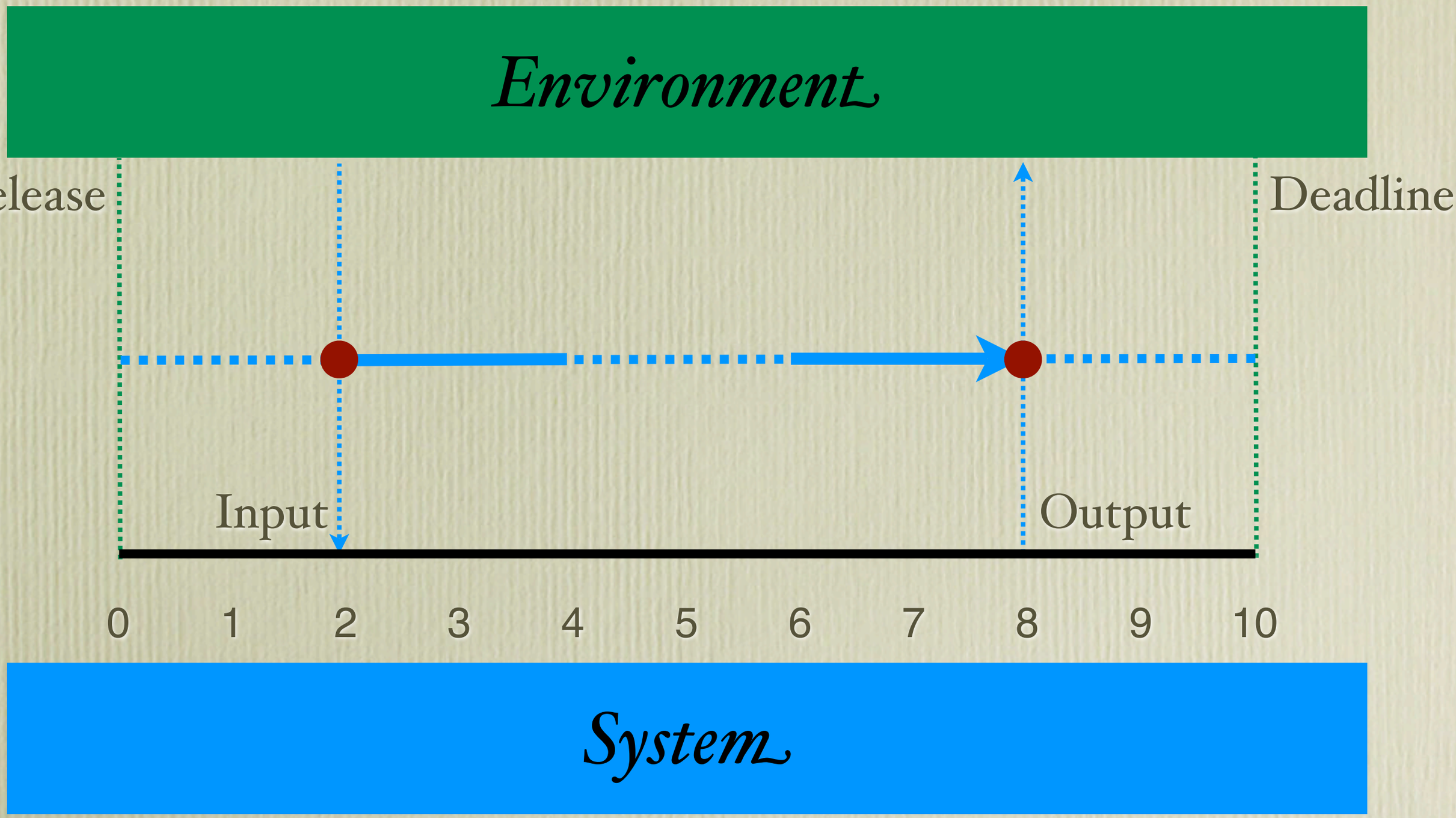
Christoph Kirsch
Universität Salzburg

Università della Svizzera italiana, March 2006

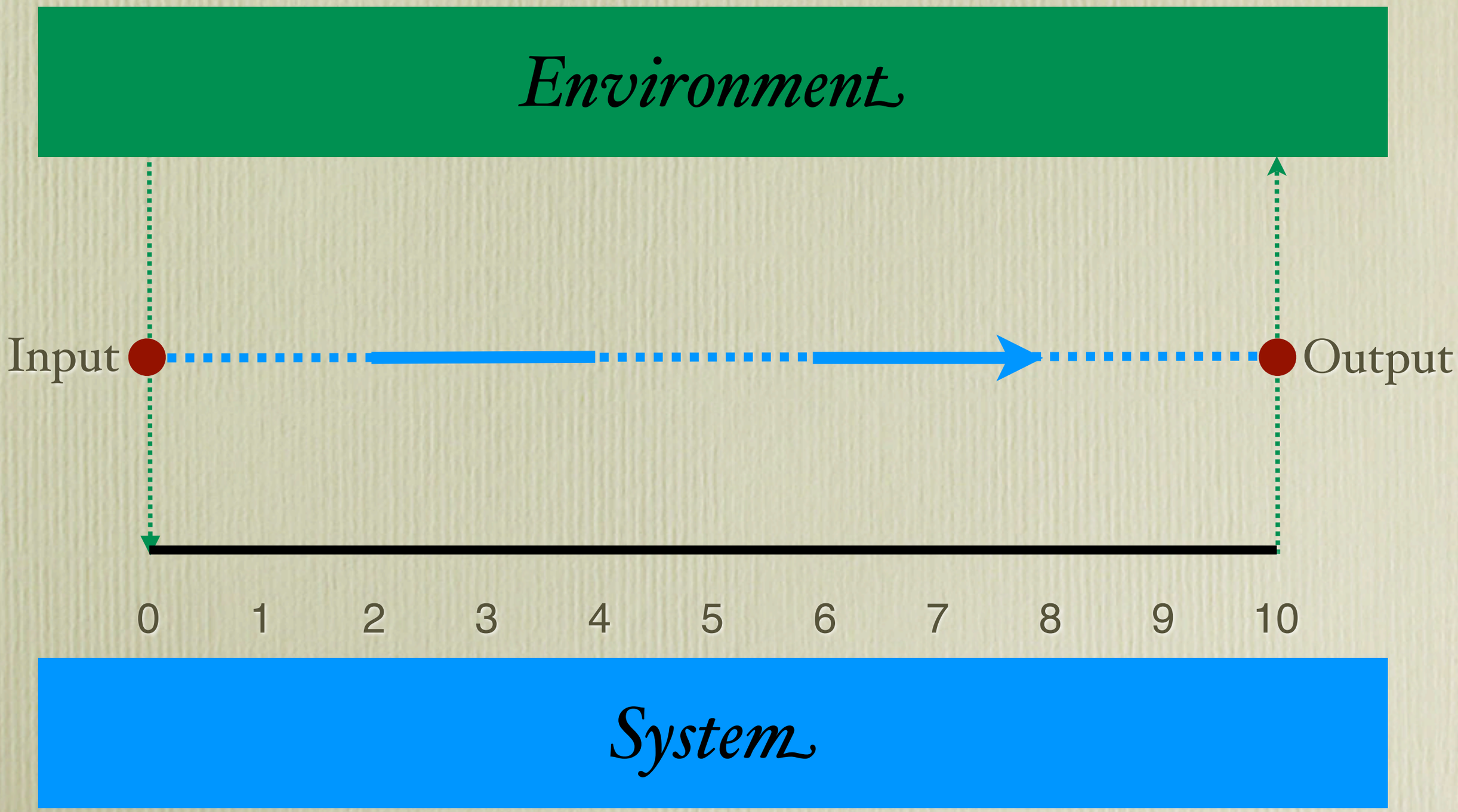
Real-Time Programming



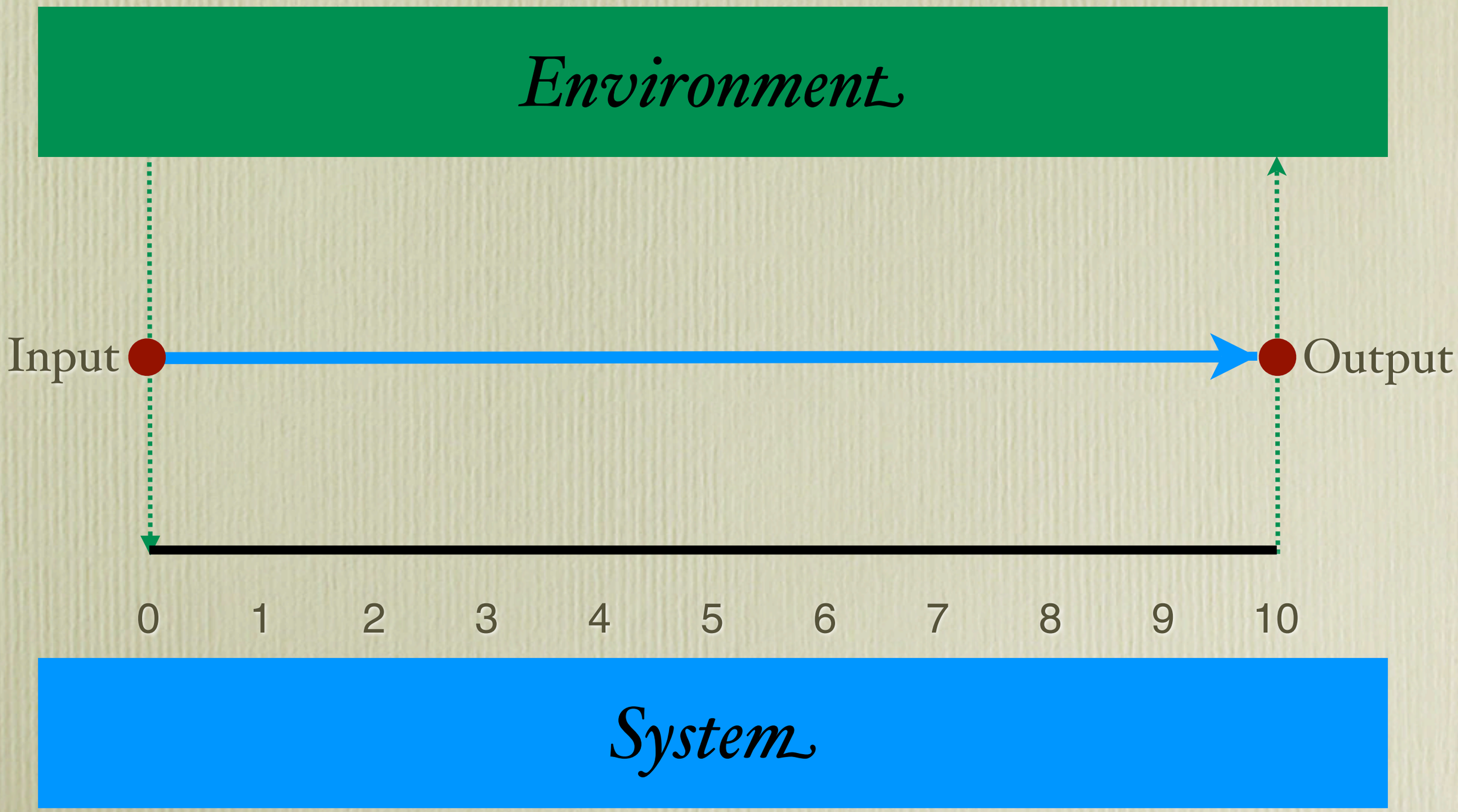
RT Programming Tradition



Logical Execution Time (LET)



Logical Execution Time (LET)



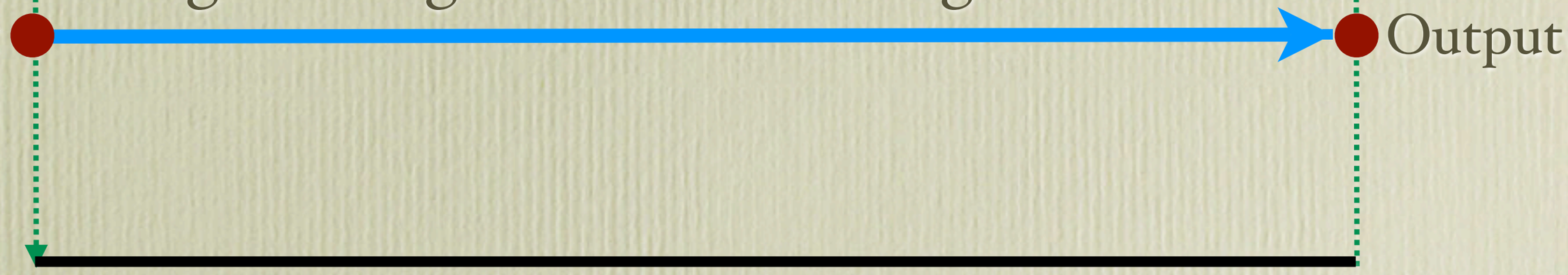
Logical Execution Time (LET)

Environment

Programming as if there is enough CPU time

Input

Output



System

Logical Execution Time (LET)

Environment

Programming as if there is enough CPU time

Input → Output

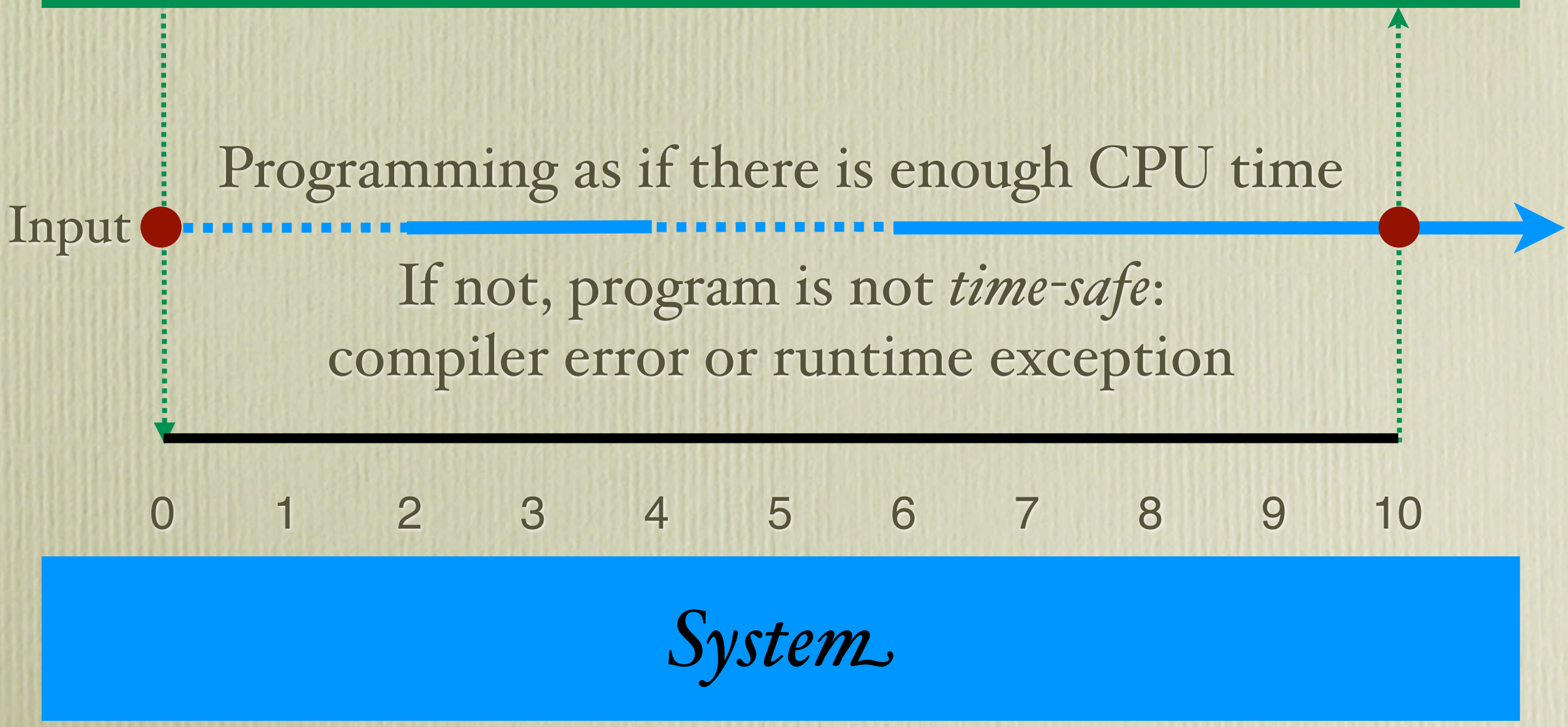
Compiler checks if there is enough CPU time

0 1 2 3 4 5 6 7 8 9 10

System

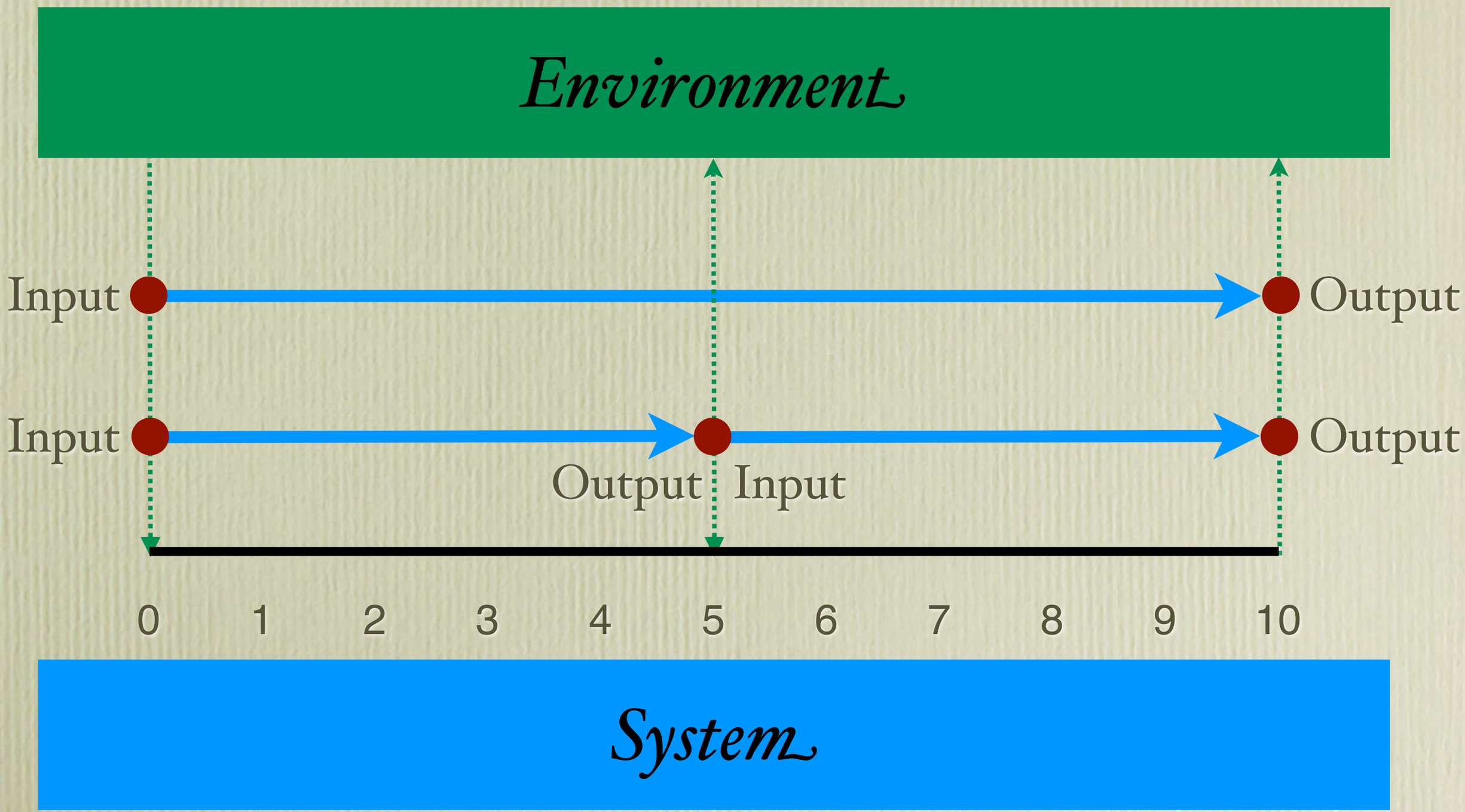
Logical Execution Time (LET)

Environment

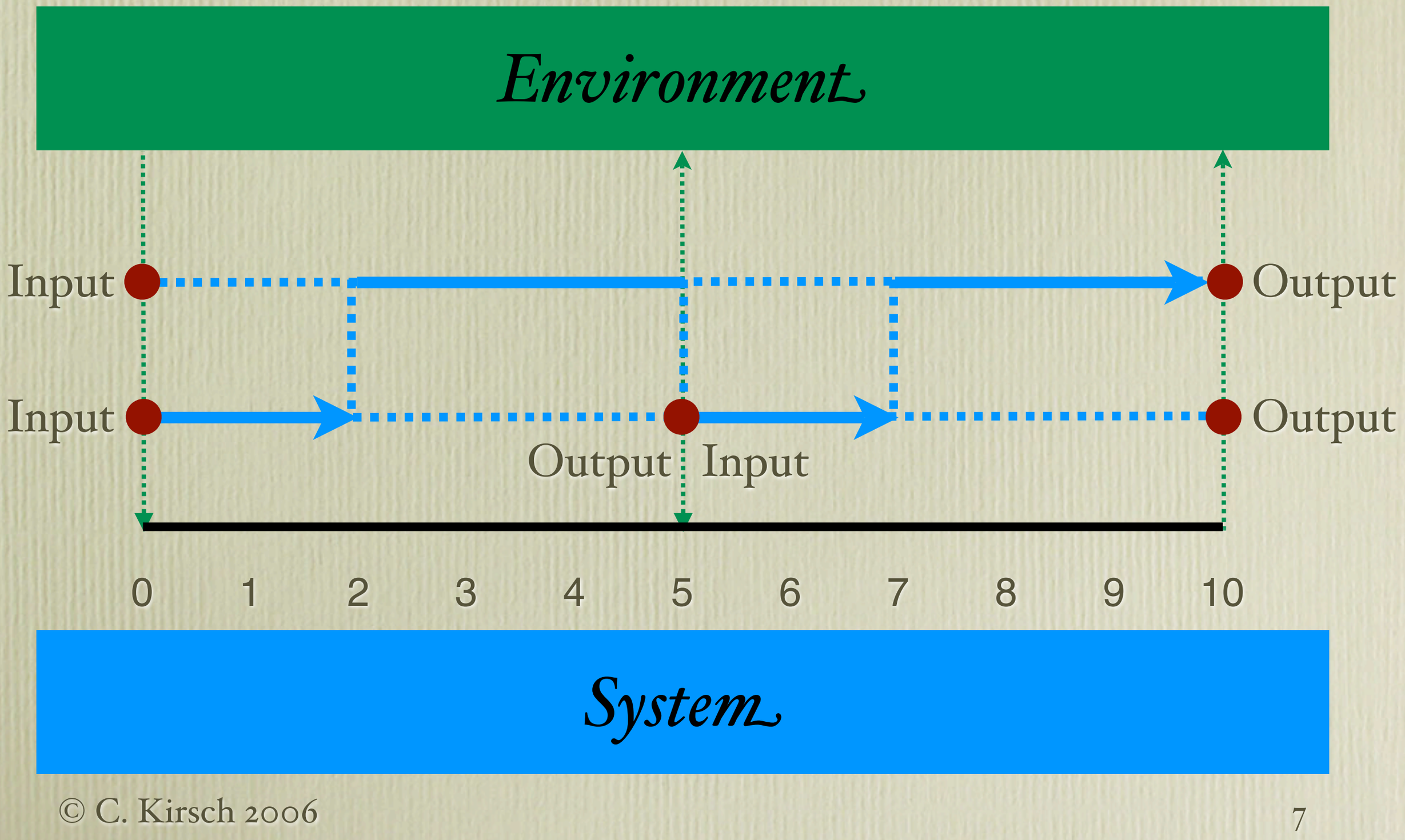


System

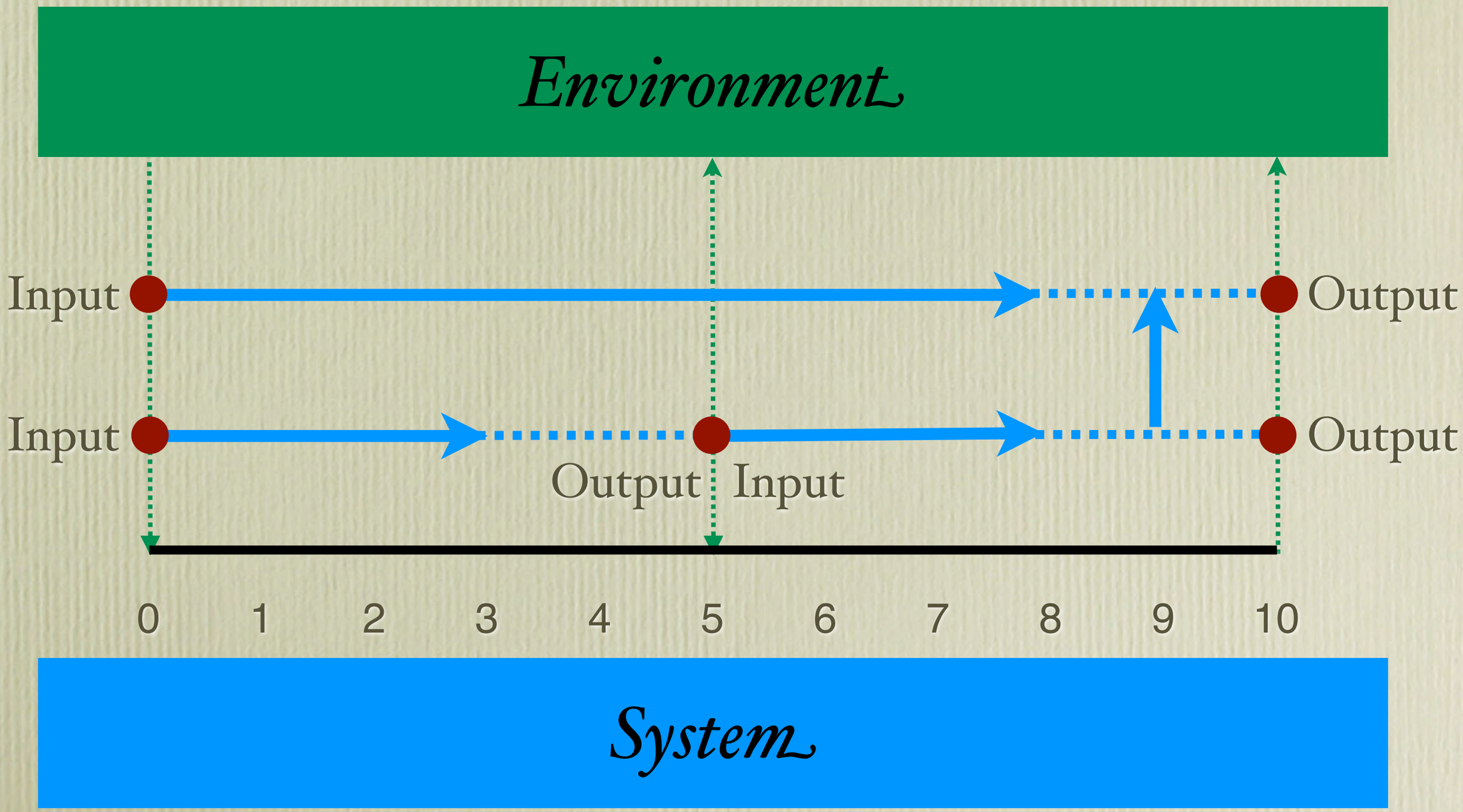
LET Programming



Single CPU, EDF Scheduler



Two CPUs, TDMA Network



Tool Chain

Giotto

Tool Chain

"Giotto: A Time-Triggered Language for Embedded Programming"

Giotto

[Proc. IEEE, 2003]
[EMSOFT, 2001]

Tool Chain

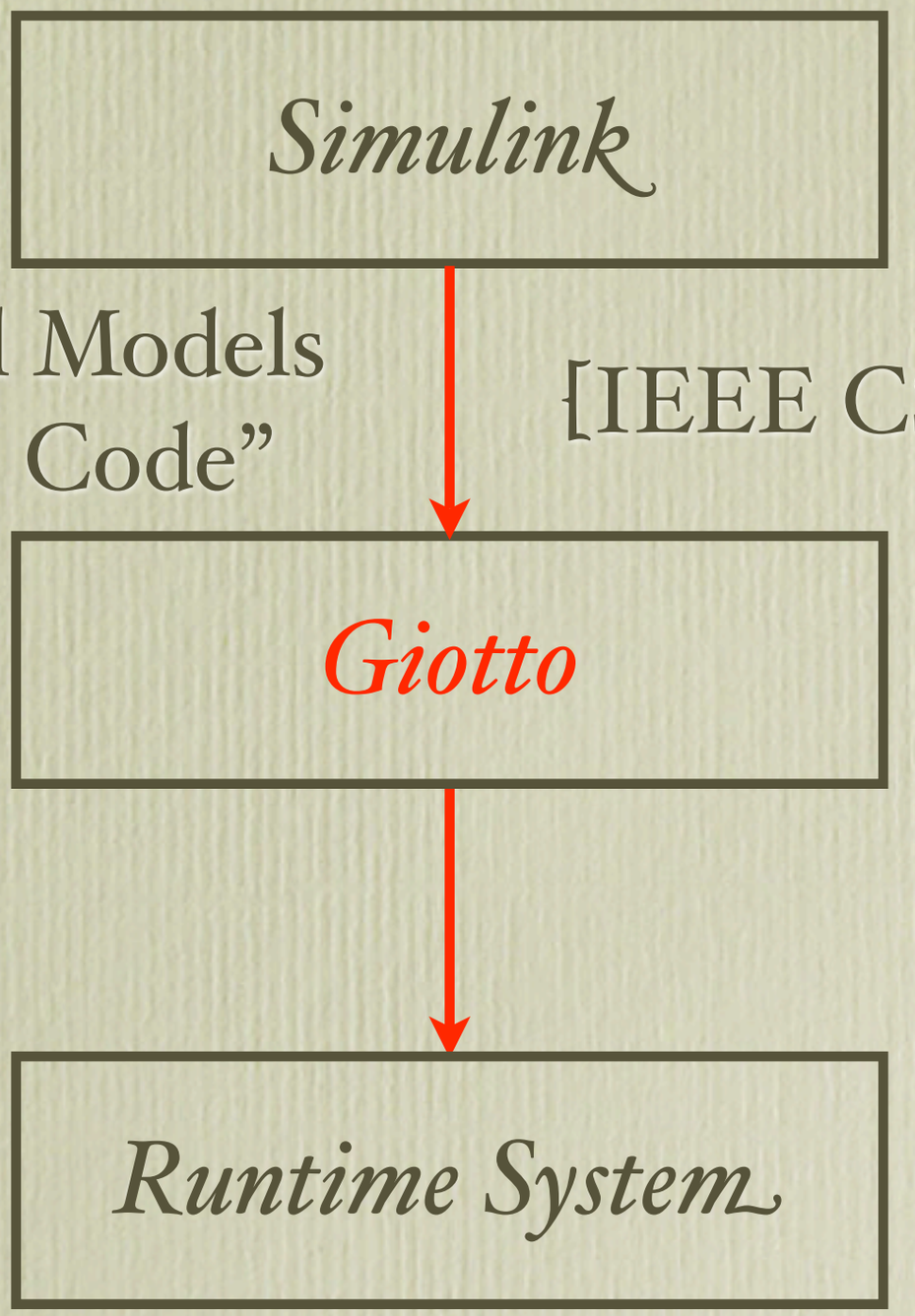
Giotto

"Time-Safety Checking for Embedded Programs"

[EMSOFT, 2002]

Runtime System

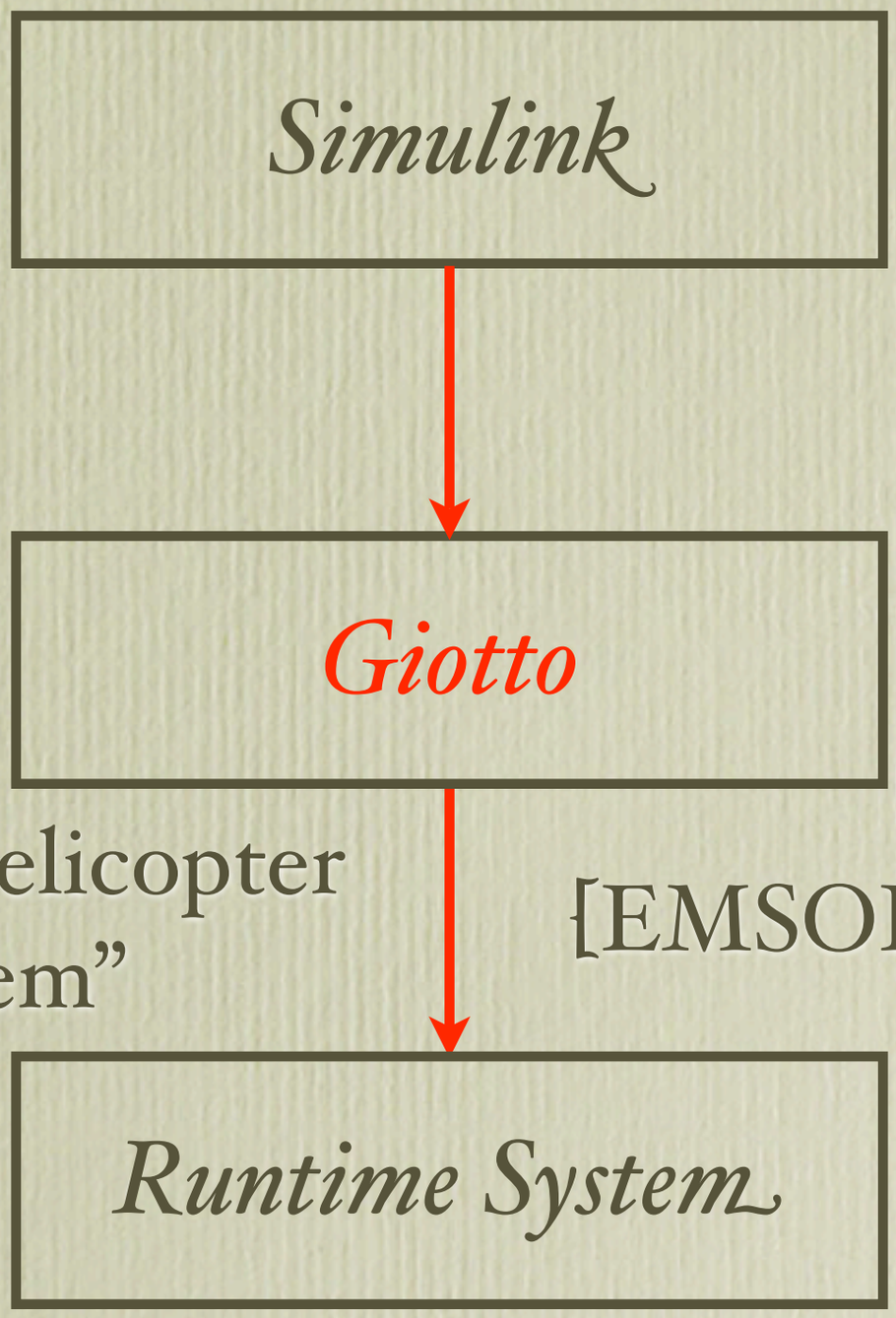
Tool Chain



"From Control Models to Real-Time Code"

[IEEE CSM, 2003]

Tool Chain

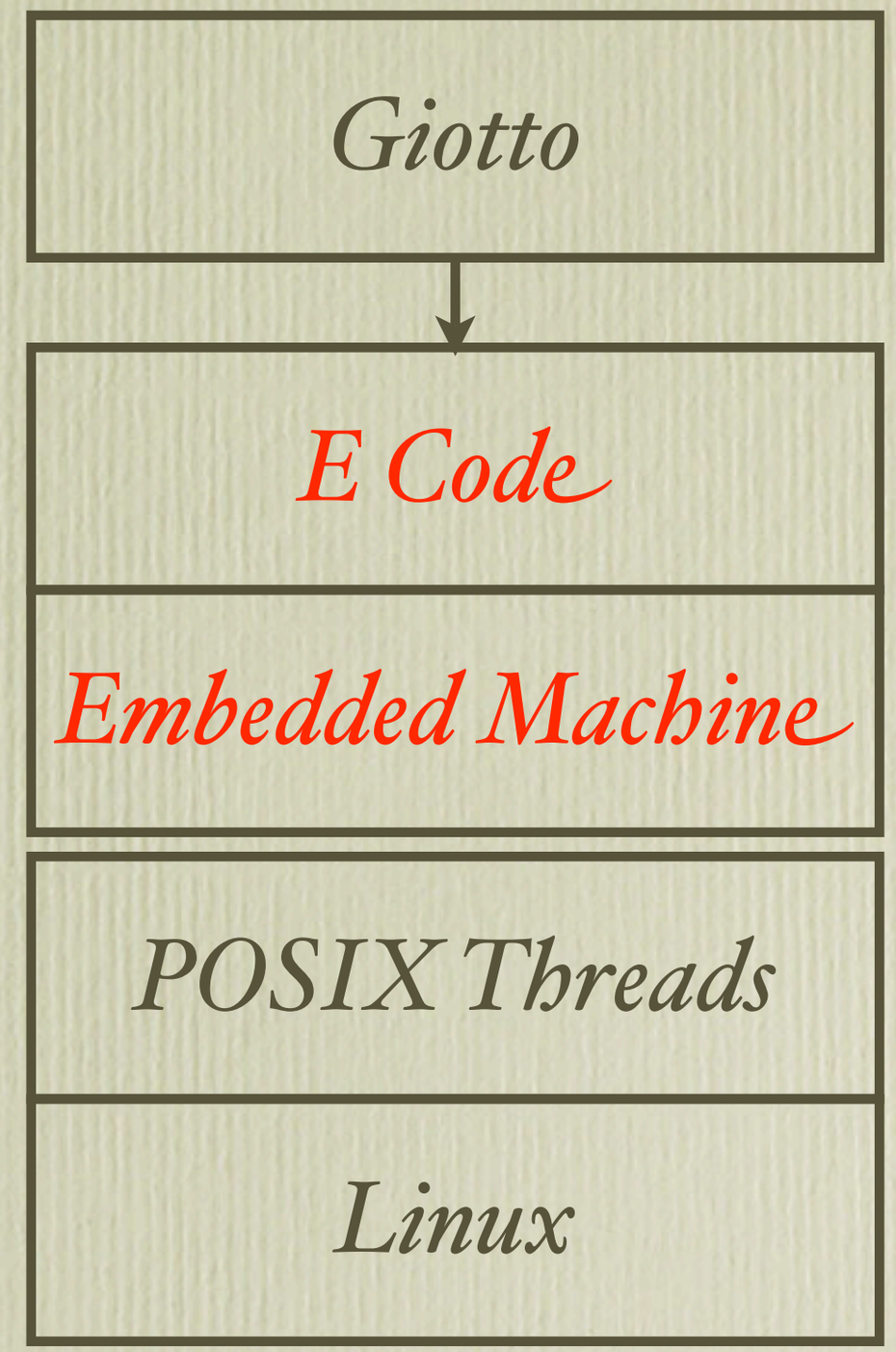


"A Giotto-Based Helicopter Control System"

[EMSOFT, 2002]

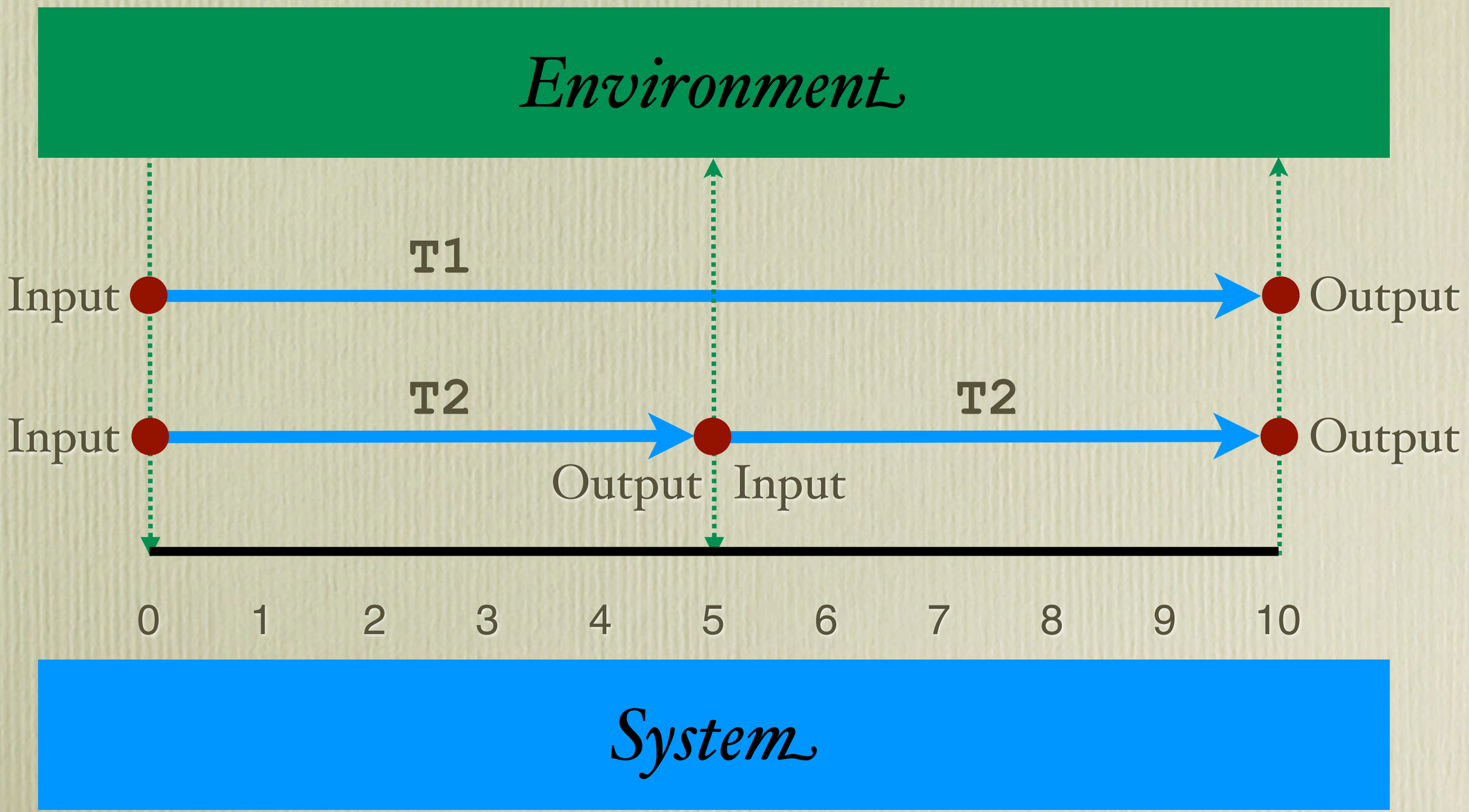
Runtime System

"The Embedded Machine: Predictable, Portable Real-Time Code"

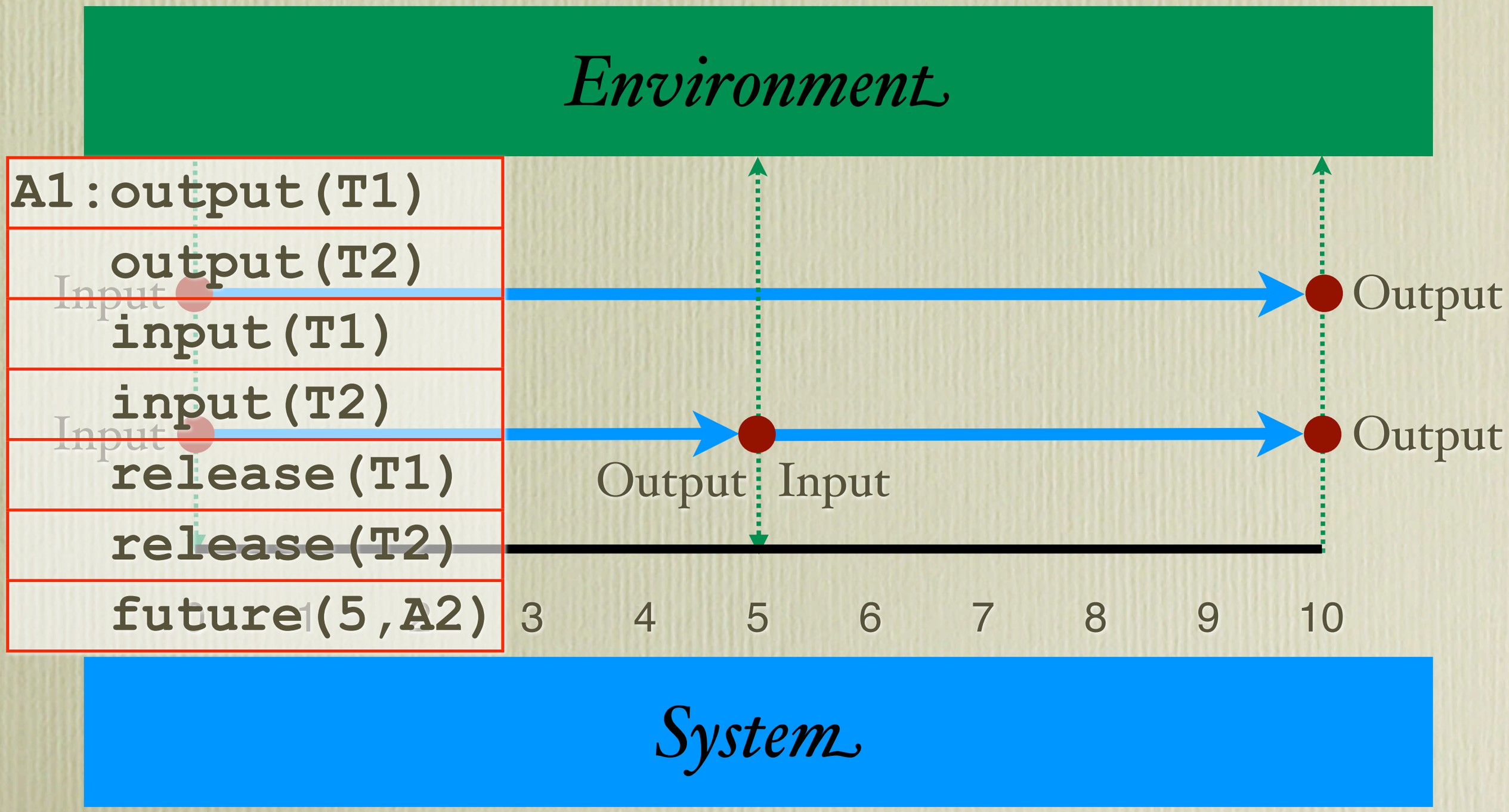


[PLDI, 2002]

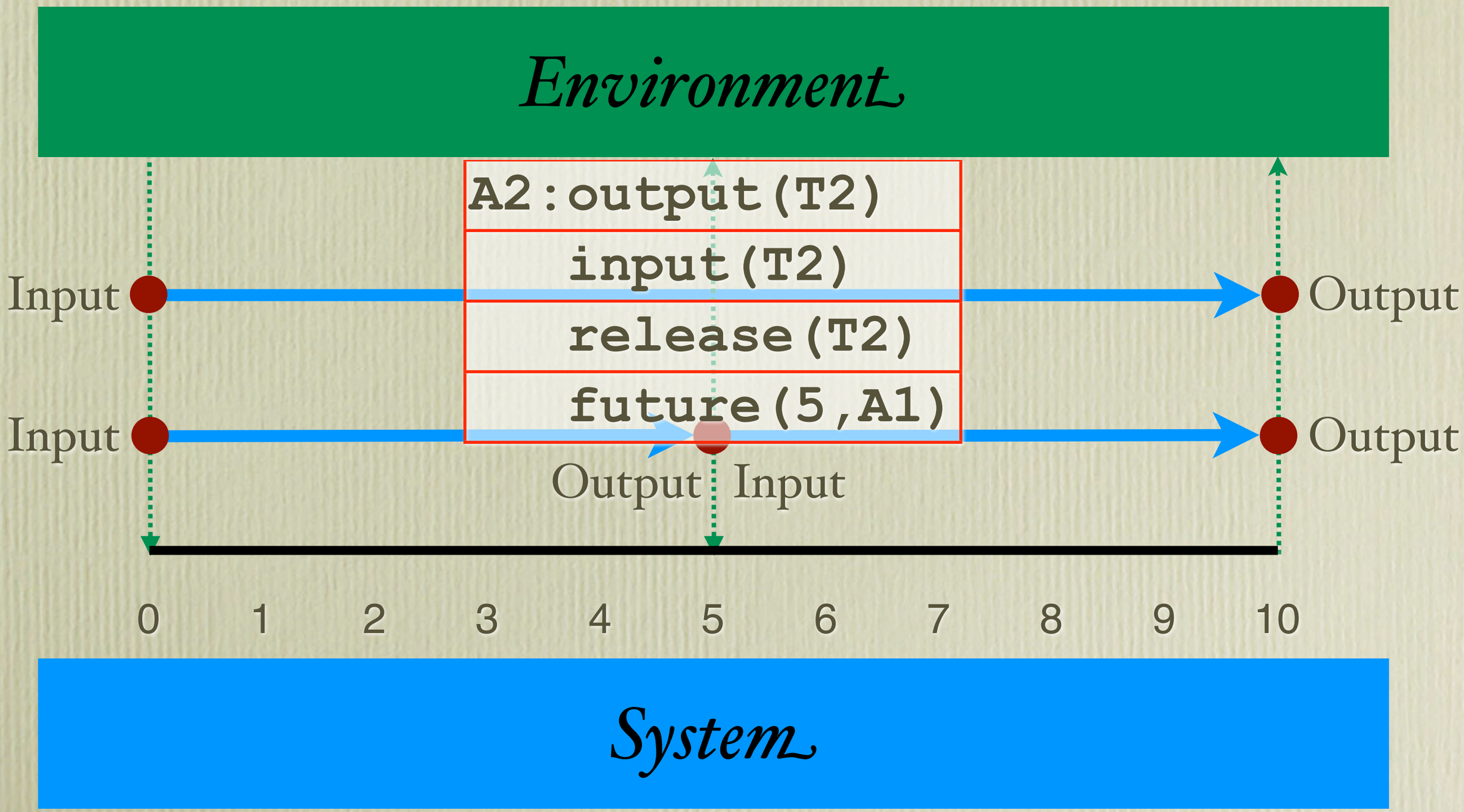
E Code



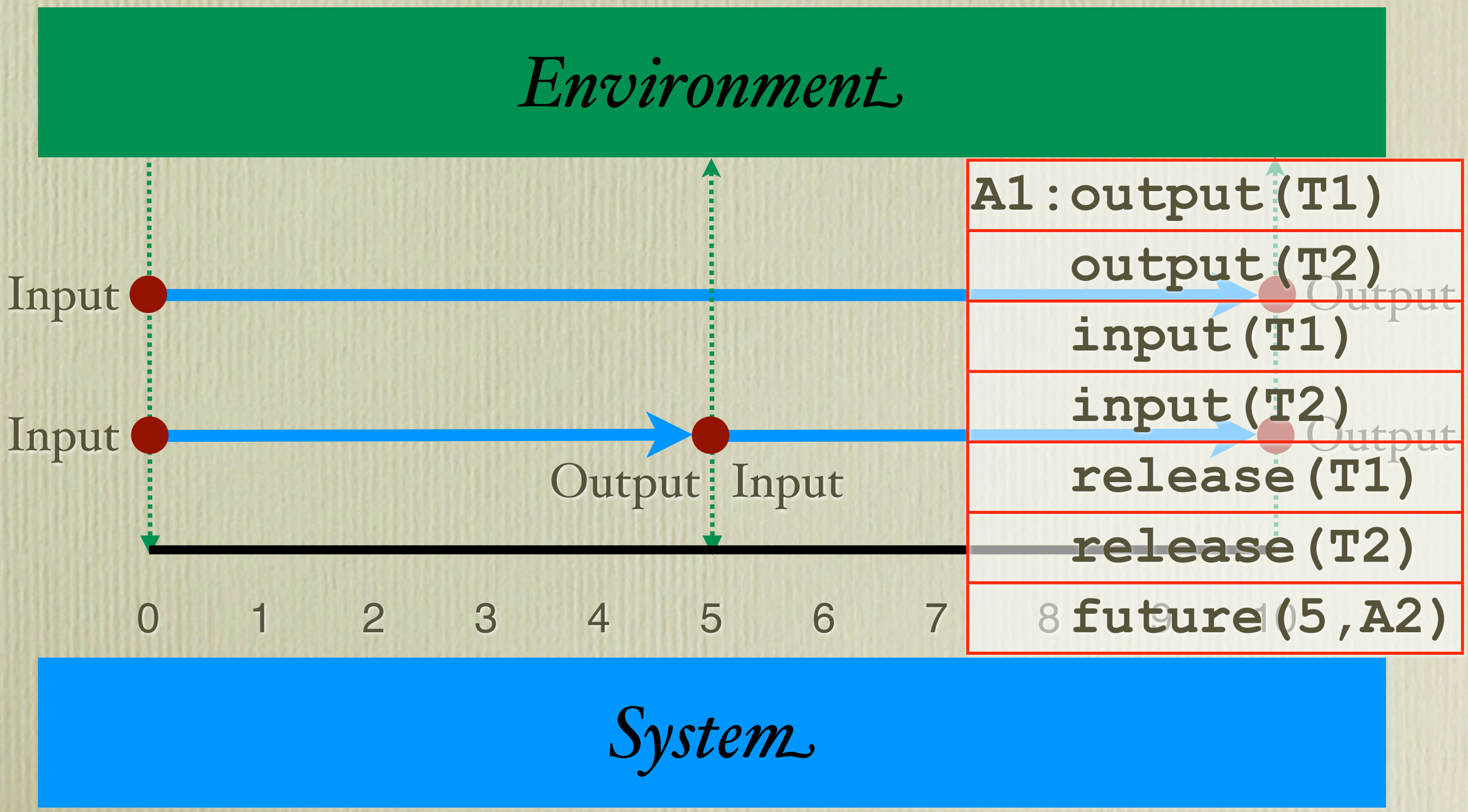
E Code



E Code



E Code



Schedule-Carrying Code

Schedule-Carrying Code	*Schedule-Carrying Code*	*Schedule-Carrying Code*
E+S Machine	*E+S Machine*	*E+S Machine*
POSIX Threads	*Microkernel*	*RT Ethernet*
Linux	*StrongARM*	*RT Linux*
[EMSOFT, 2003]	[VEE, 2005]	[LCTES, 2005]

Current Projects

TAP Project

LET

HTL Project

[w/ EPF Lausanne & UC Berkeley]

LET and Java

[w/ IBM T.J. Watson]

JAviator Project

AirJava Project

[w/ UC Berkeley]

The HTL Project

htl.cs.uni-salzburg.at

- Goal:

  ➡ enable *compositional* real-time programming of distributed control systems

- Solution:

  ➡ HTL programs are extensible in two dimensions without changing their timing behavior: new program modules can be *added* and individual program tasks can be *refined*

Collaboration

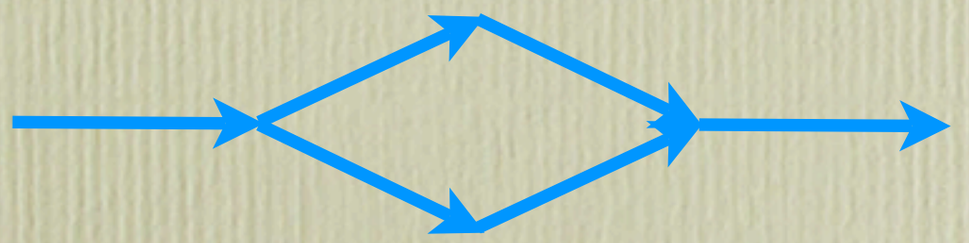
- UC Berkeley (A. Ghosal, PhD student with A. Sangiovanni-Vincentelli)
- Politehnica Univ. of Timisoara (D. Iercan, PhD student)
- EPFL (T. Henzinger)
- Univ. of Salzburg (Myself: looking for students)

Parallel Composition

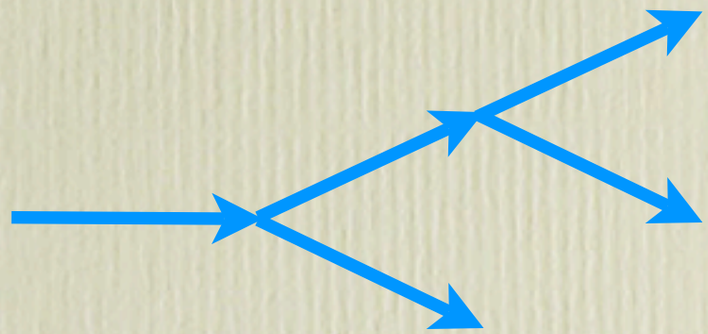
Program Module M1

||

Program Module M2

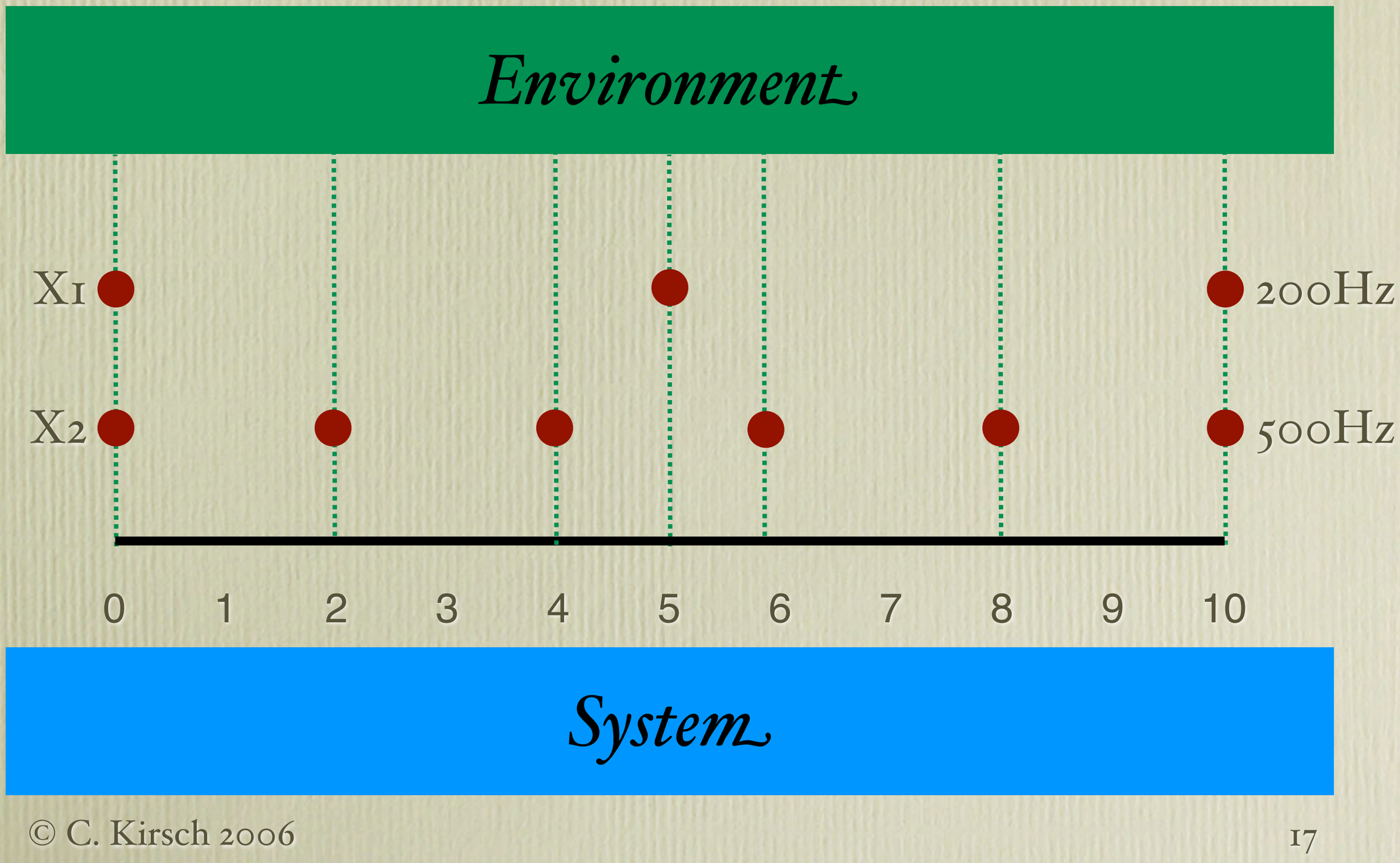


100Hz

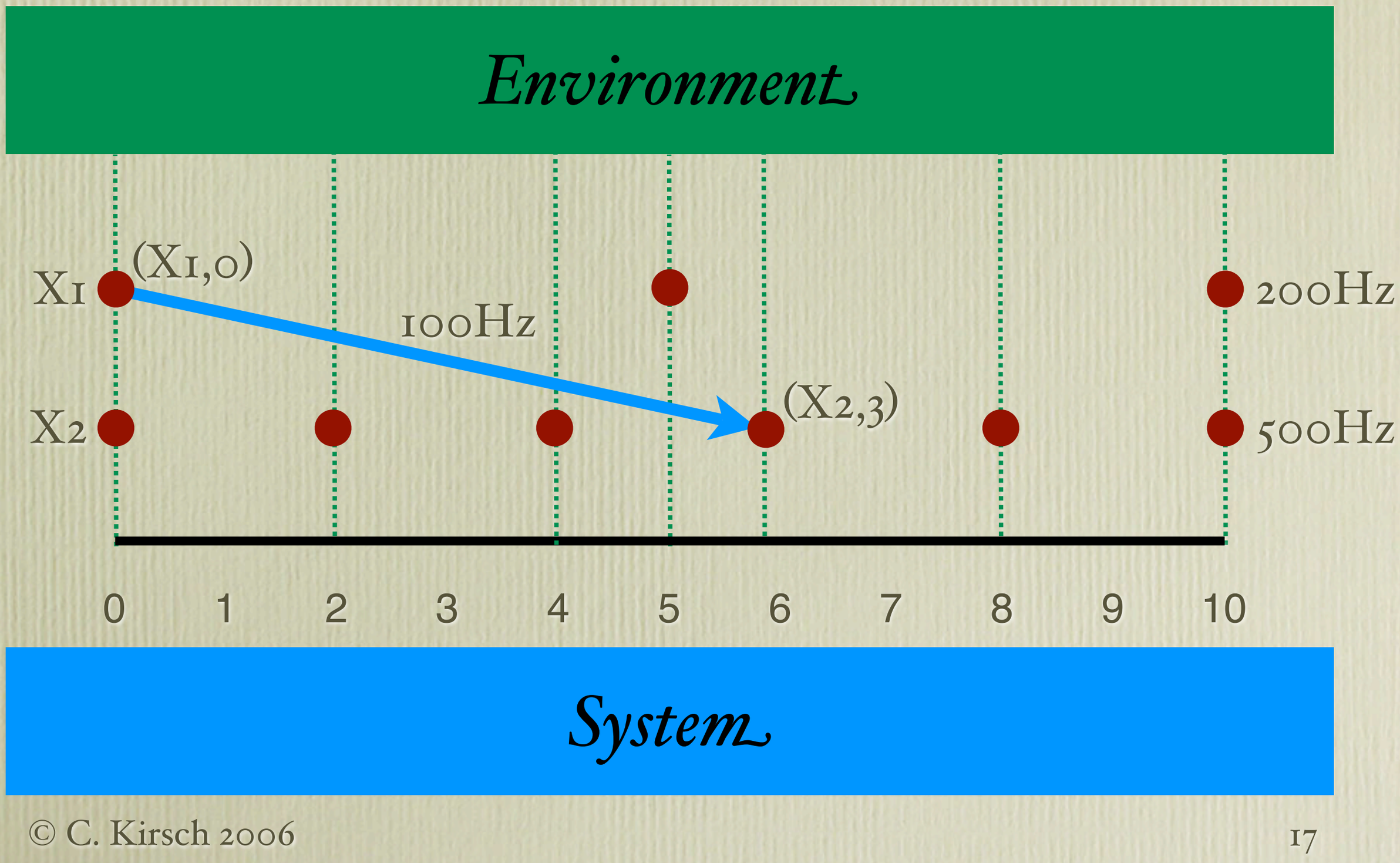


200Hz

Timed Variables

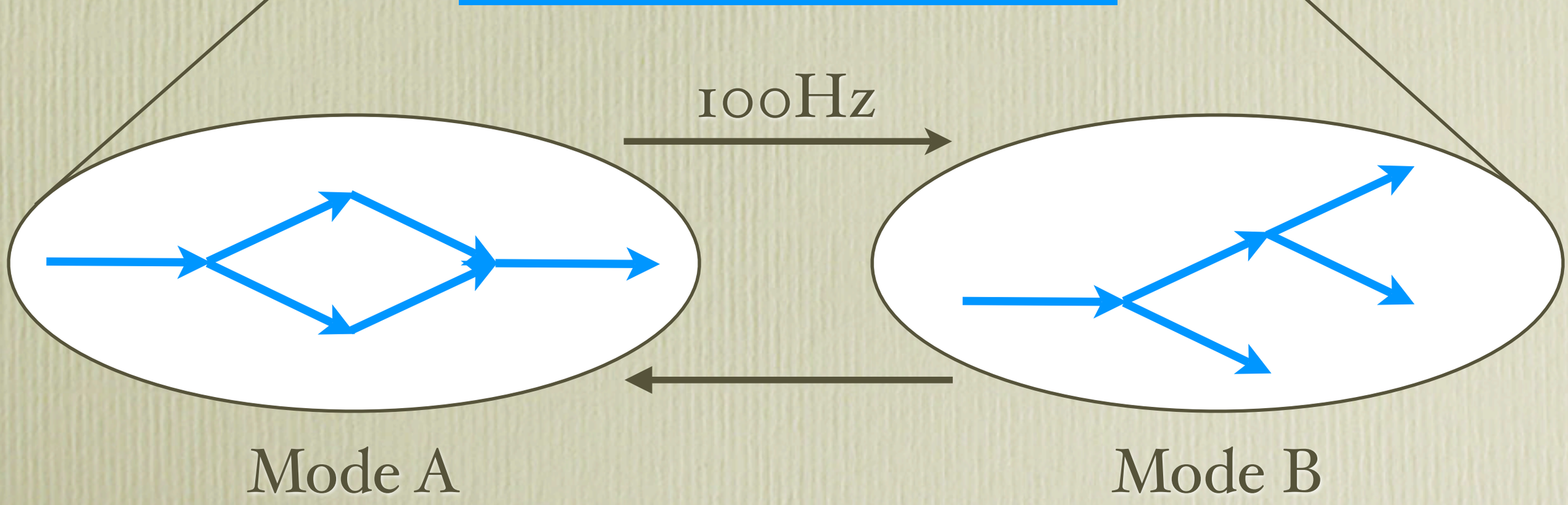


Timed Variables

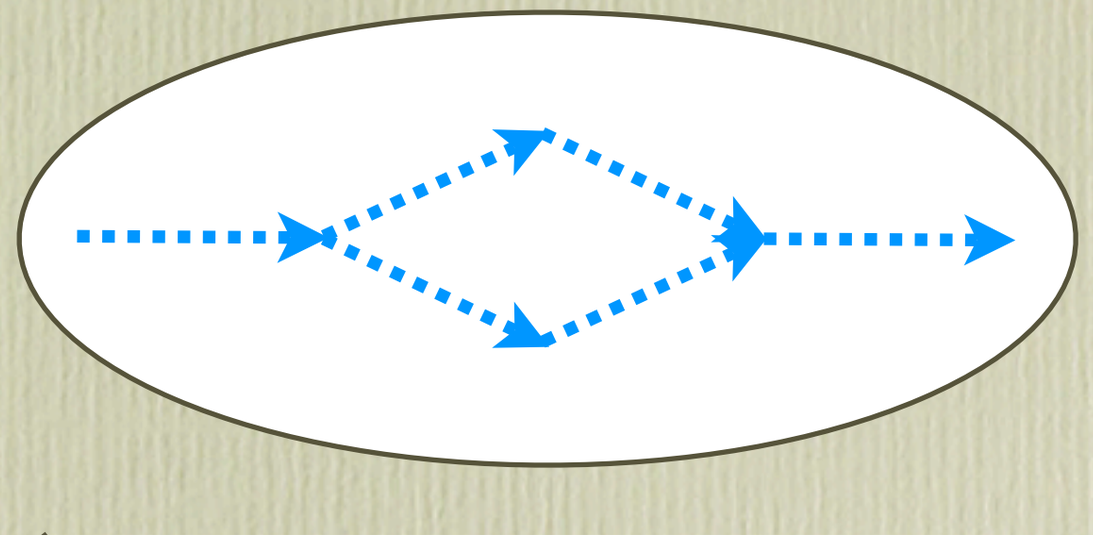


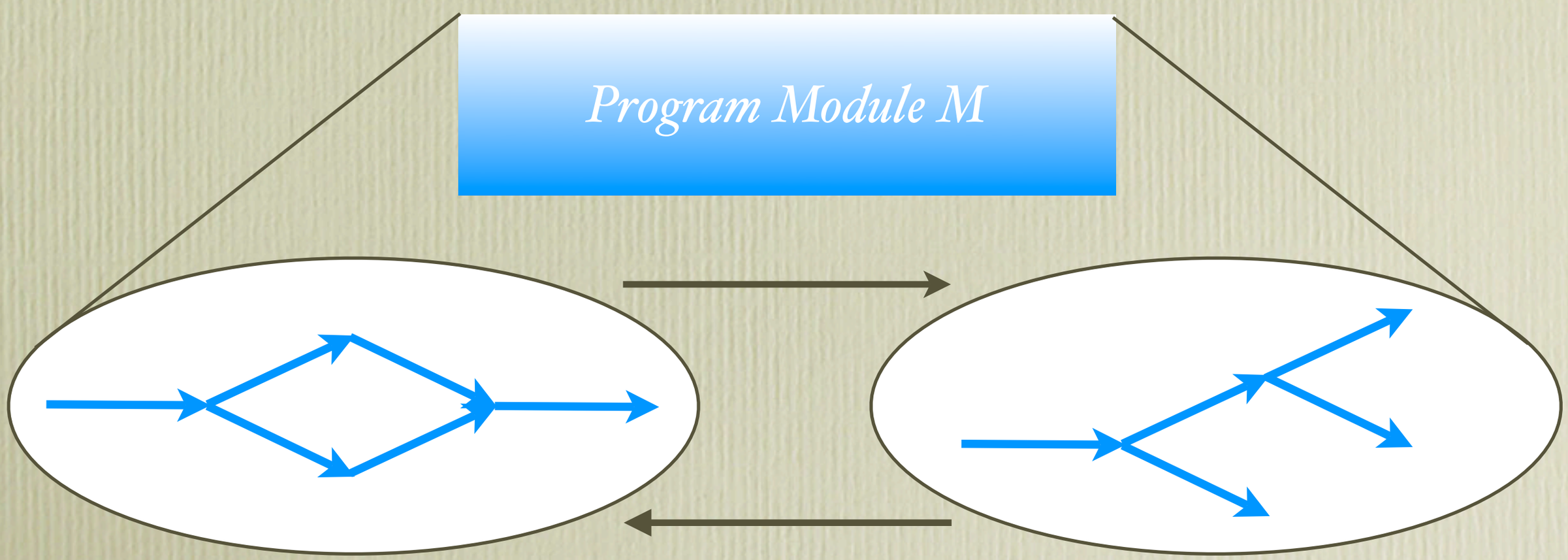
Sequential Composition

Program Module M

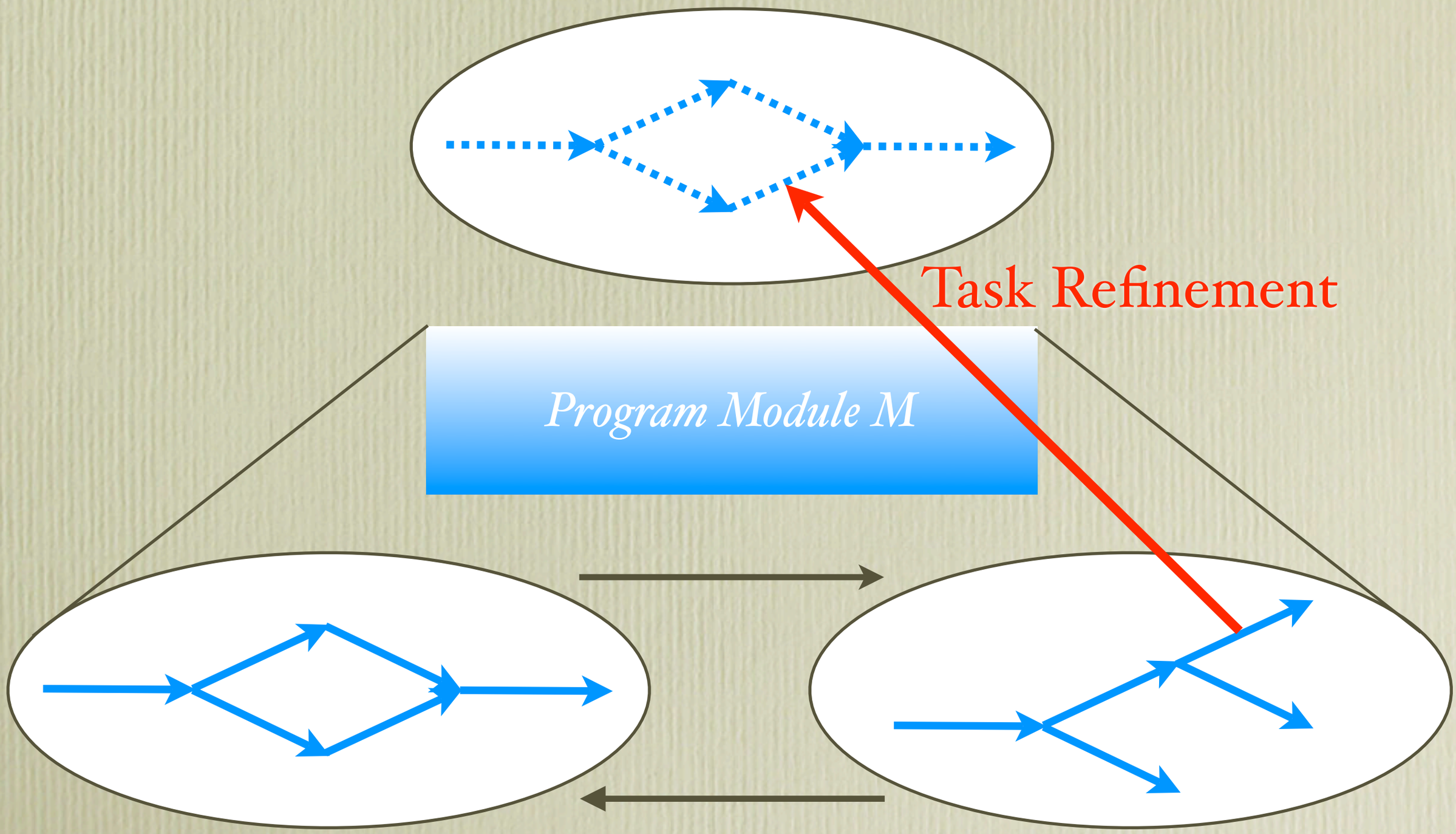


Program Refinement

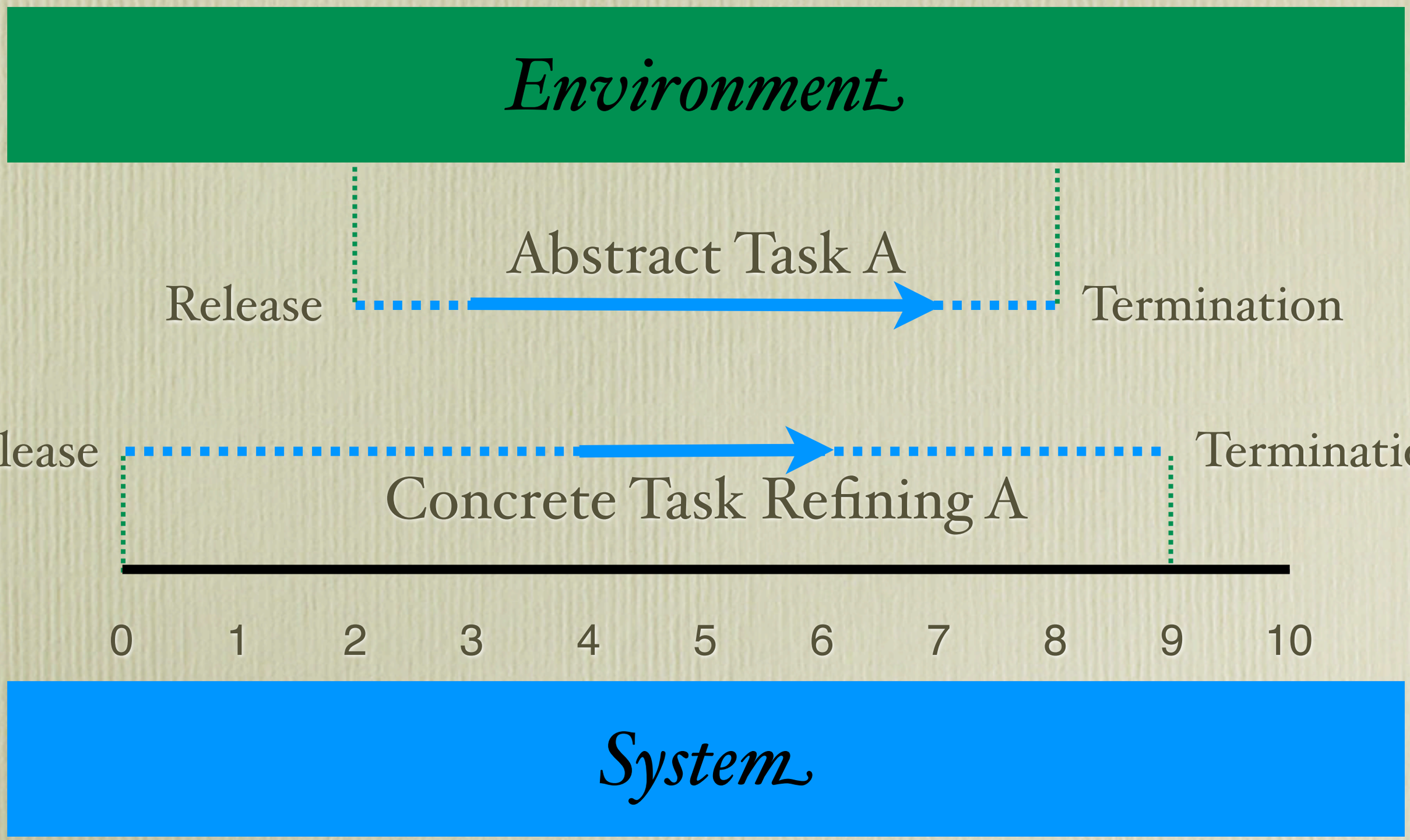




Program Refinement



Task Refinement



Compositional Real-Time Programming in HTL

Submitted to LCTES 2006

If there is a time-safe execution trace of an (abstract) HTL program A, then there is a time-safe execution trace for any (concrete) HTL program that refines A.

CHESS

Center for Hybrid & Embedded Software Systems

COLLEGE OF ENGINEERING, UC BERKELEY

Hierarchical Timing Language

Arkadeb Ghosal
Thomas A. Henzinger[1]
Daniel Iercan[2]
Christoph Kirsch[3]
Alberto Sangiovanni-Vincentelli

(1) EPFL
(2) Politehnica U. of Timisoara
(3) University of Salzburg

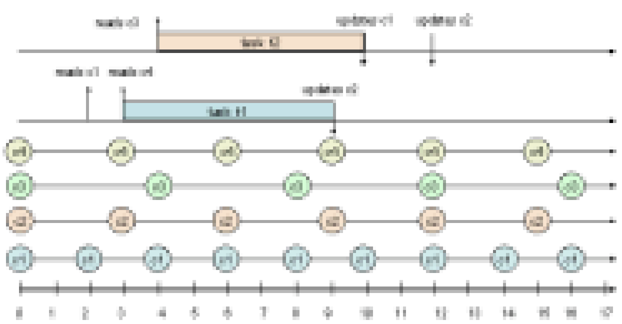
http://chess.eecs.berkeley.edu/

Hierarchical Timing Language (HTL) ....

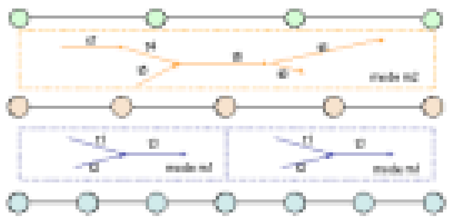
... is a new programming language for hard real-time systems. Critical timing constraints are specified within the language, and ensured by the compiler. The main novel feature of the language is that programs are extensible in two dimensions without changing their timing behavior: new program modules can be added, and individual program task can be refined. The mechanism that supports time invariance under parallel composition is that different program modules communicate at specified instances of time. Time invariance under refinement is achieved by conservative scheduling of the top level. The language assembles real-time tasks within a hierarchical module structure with timing constraints and is a coordination language (i.e. individual tasks can be implemented in other languages).

HTL compiler, examples and case-study implementations are available at http://htl.cs.uni-salzburg.at/

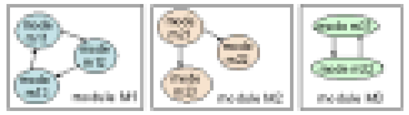
Communicators and Tasks



Core Program Structure.



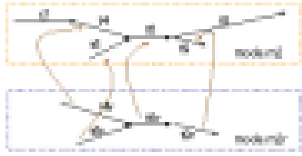
Extending core structure.



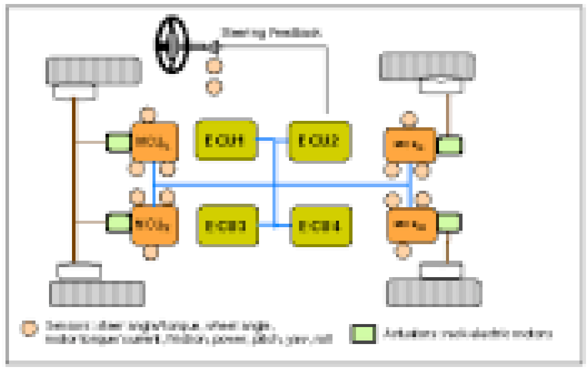
Abstract Task and Concrete Task.

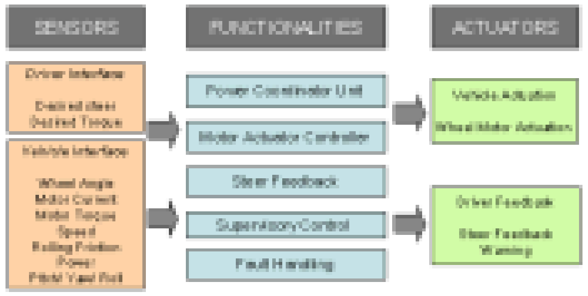
Abstract tasks have no function definition but only interface definition. Concrete tasks have both function definition and interface definition.

Refinement.



STEER BY WIRE

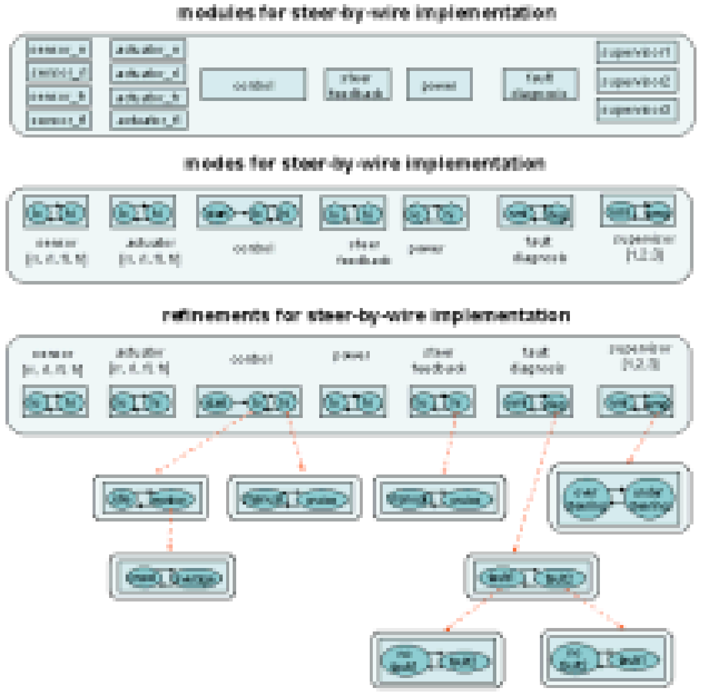




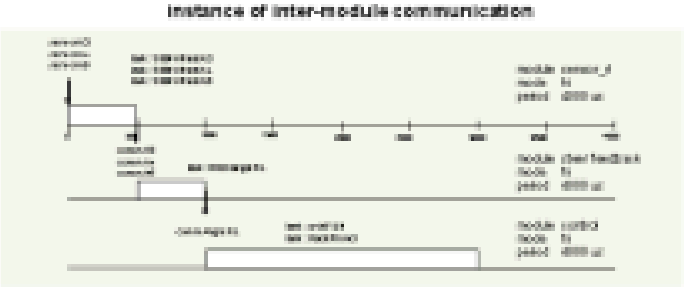
modules for steer-by-wire implementation

modes for steer-by-wire implementation

refinements for steer-by-wire implementation



instance of inter-module communication



Refinement (Vertical Extension).

A mode in a program can be replaced by another HTL program. This does not add expressiveness of the model; in fact a HTL program with arbitrary levels of refinement can be translated into one with no refinement. However the feature allows compact representation without overloading analysis; particularly it simplifies schedulability.

Composing Modules (Horizontal Extension).

Parallel modules can be appended to the implementation without changing the timing behavior of the implementation.

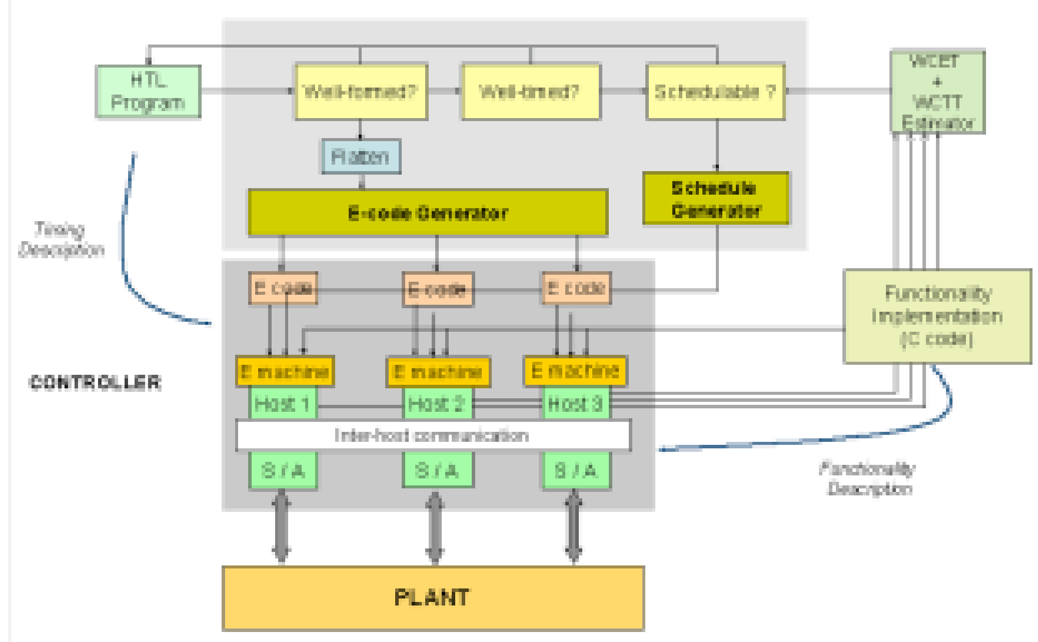
Distribution.

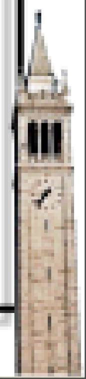
HTL allows a programmer to specify the distribution of an HTL program; different modules can run on different hosts. The semantics remain the same as if they were running on a single host; however code generation and analysis take the distribution into account. Distribution is currently done manually; in the future we would like to automate the mapping taking into account program structure and platform characteristics.

Schedulability.

Each task refinement is constrained in such a way that if the task is schedulable, then the more detailed replacement group of tasks is schedulable as well. As a consequence, schedulability needs to be checked only for the top level of an HTL program. This avoids a combinatorial explosion, and permits scheduling to be performed by the HTL compiler.

HTL COMPILER





The JAviator Project

javiator.cs.uni-salzburg.at

- Goal:

  ➡ enable high-performance real-time code, e.g., flight control software, to be written *entirely* in Java

- Challenge:

  ➡ enable *submillisecond, predictable* real-time behavior while maintaining as much *original* Java semantics as possible

Collaboration

see also [EMSOFT 2005]

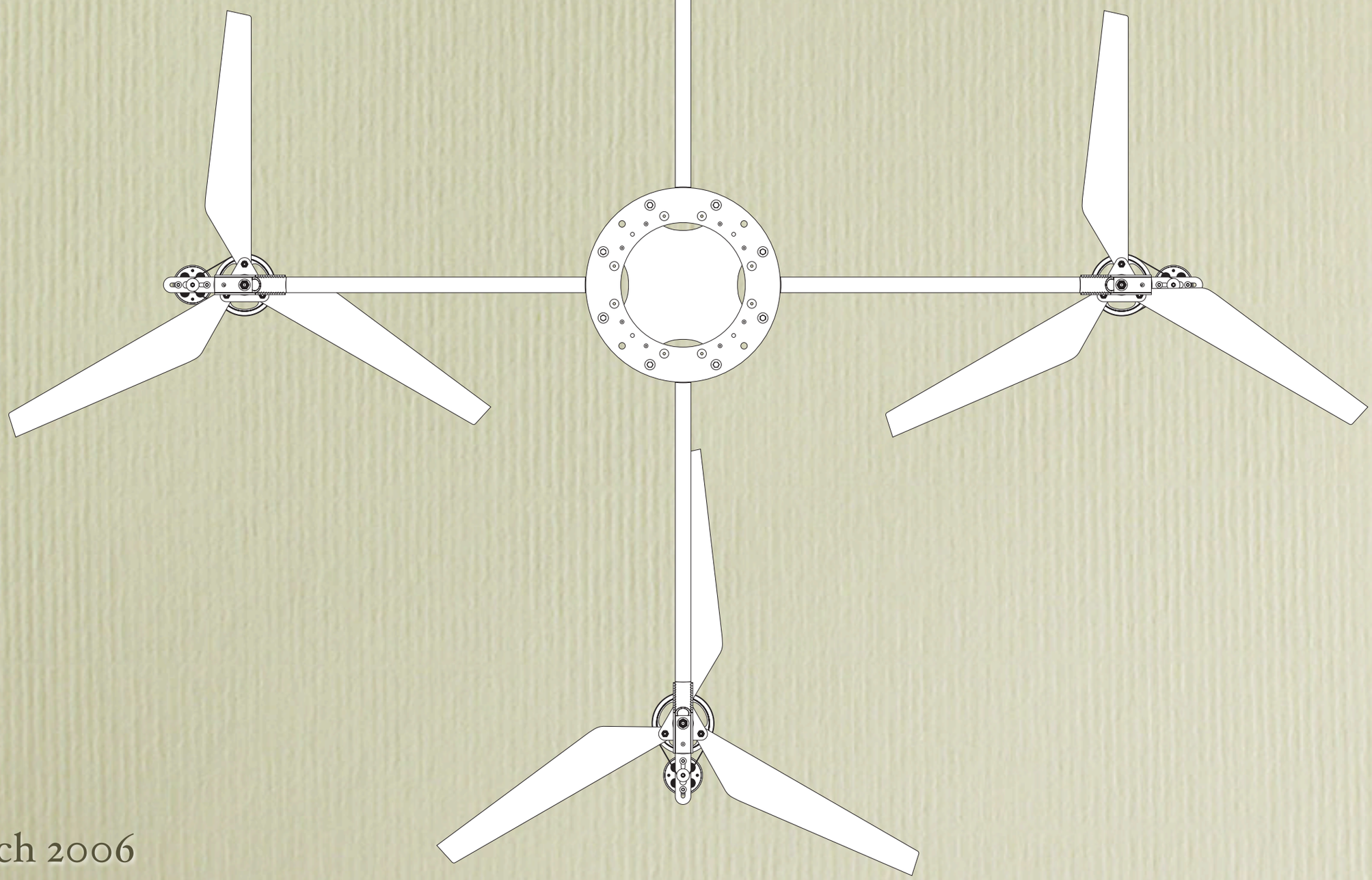
- IBM (2 staff researchers, J.Auerbach, D.Bacon):

  ➡ design and implementation of high-performance real-time garbage collection (Metronome)

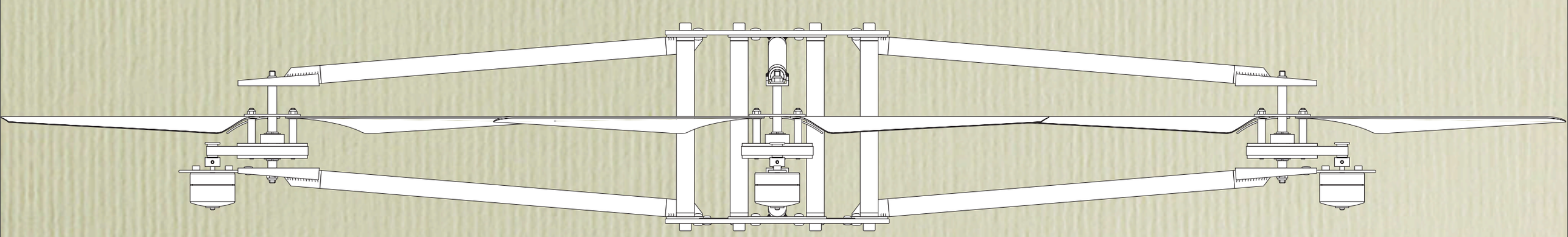
- Our team (3 PhD, 3 Masters students):

  ➡ design and implementation of a LET-based concurrency model that extends Java's notion of "write-once-run-anywhere" to the temporal domain

Platform



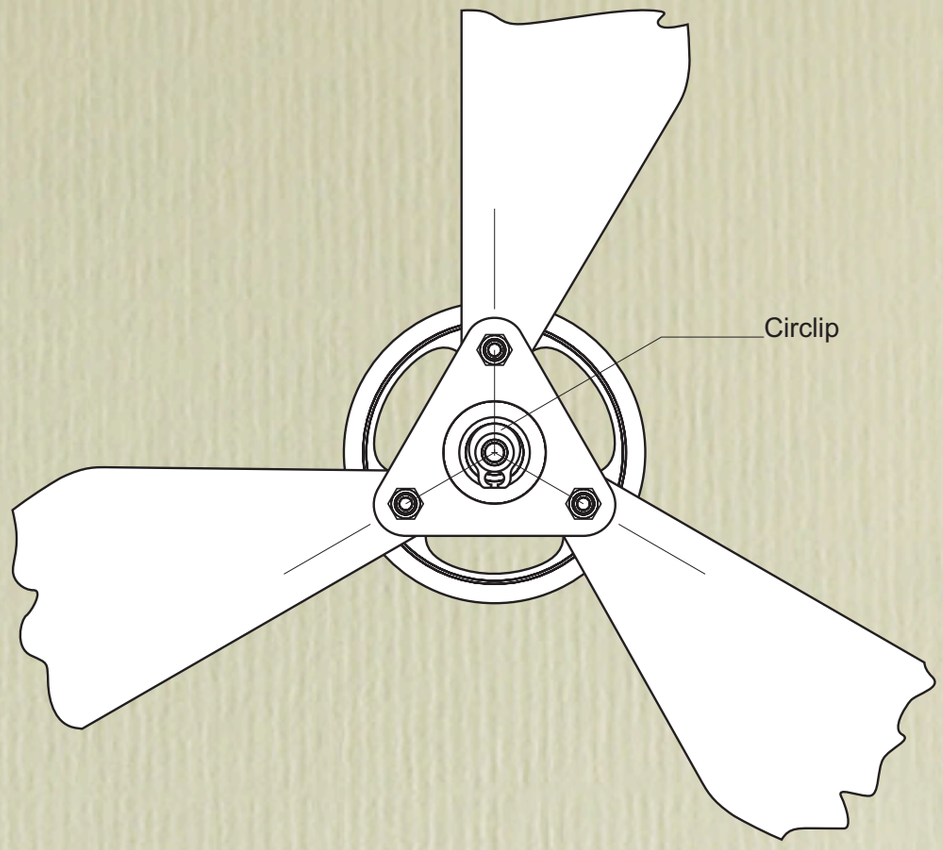
It’s a ‘Bicycle Wheel’

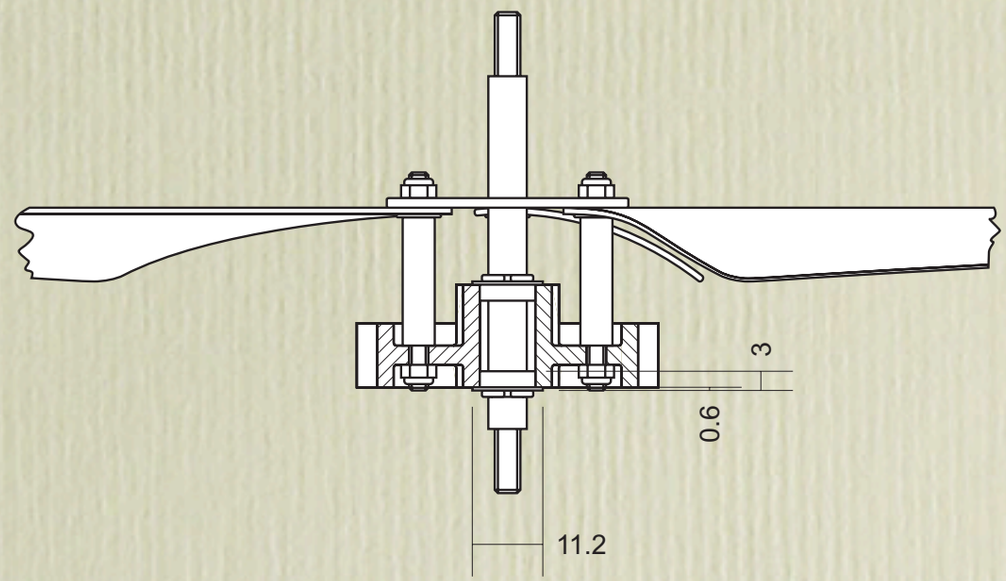


Design

The hardware design including all blueprints will be made available at:

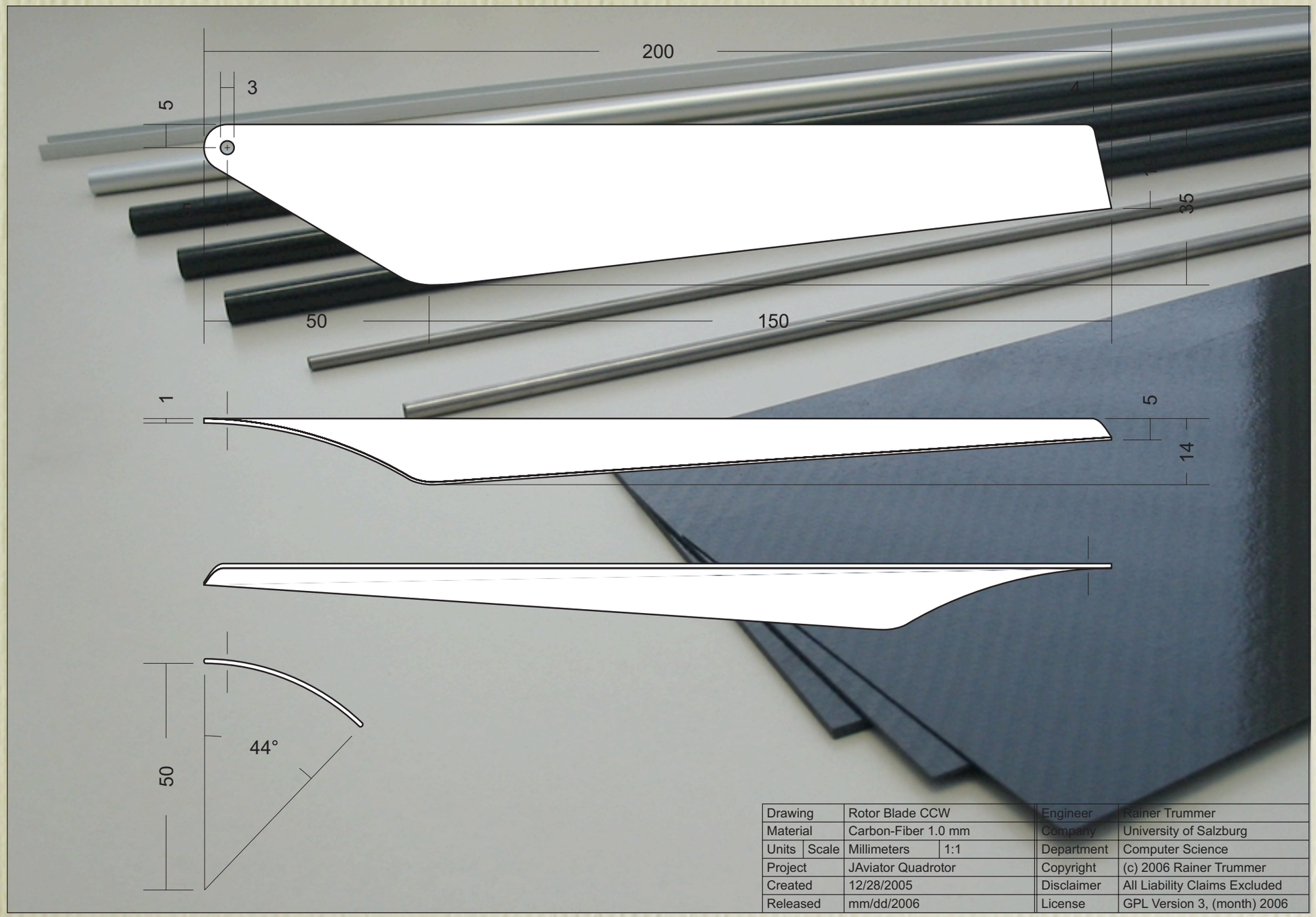
javiator.cs.uni-salzburg.at





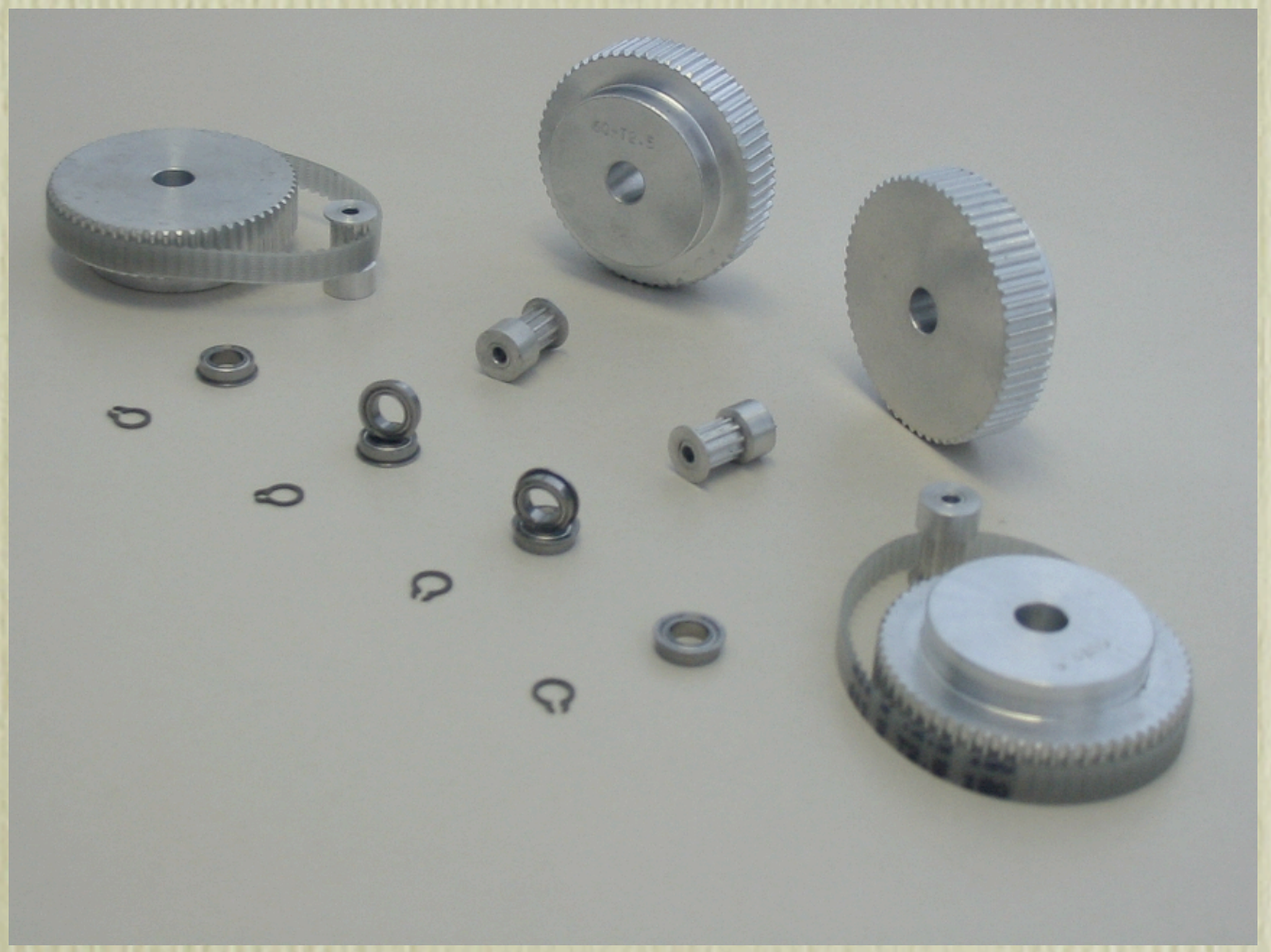
Drawing		Rotor Bearings DDLF-1060		Engineer	Rainer Trummer
Material		Stainless-Steel Alloy		Company	University of Salzburg
Units	Scale	Millimeters	1:1	Department	Computer Science
Project		JAviator Quadrotor		Copyright	(c) 2006 Rainer Trummer
Created		01/07/2006		Disclaimer	All Liability Claims Excluded
Released		mm/dd/2006		License	GPL Version 3, (month) 2006

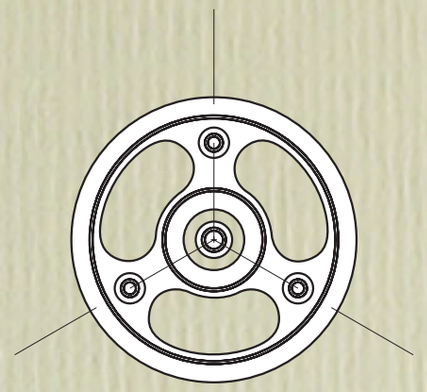
Ouch: Carbon Fiber Blades

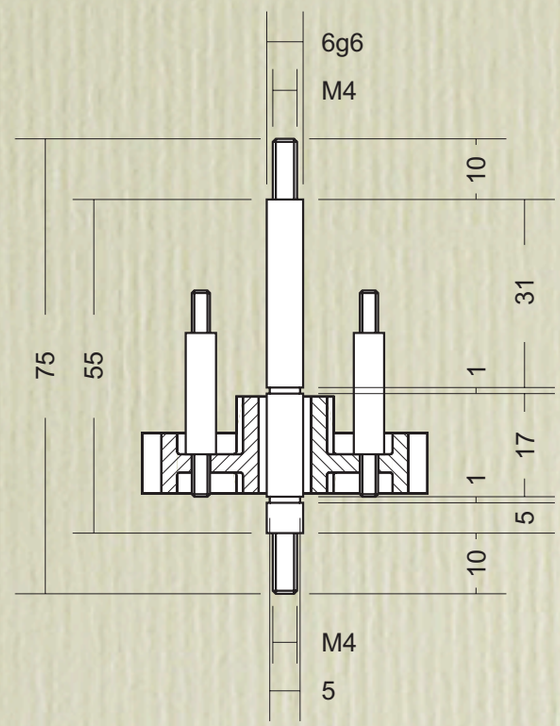


Drawing		Rotor Blade CCW		Engineer	Rainer Trummer
Material		Carbon-Fiber 1.0 mm		Company	University of Salzburg
Units	Scale	Millimeters	1:1	Department	Computer Science
Project		JAviator Quadrotor		Copyright	(c) 2006 Rainer Trummer
Created		12/28/2005		Disclaimer	All Liability Claims Excluded
Released		mm/dd/2006		License	GPL Version 3, (month) 2006

Weight..less







Drawing		Rotor Axle		Engineer	Rainer Trummer
Material		Titan Alloy TiAl6V4		Company	University of Salzburg
Units	Scale	Millimeters	1:1	Department	Computer Science
Project		JAviator Quadrotor		Copyright	(c) 2006 Rainer Trummer
Created		01/07/2006		Disclaimer	All Liability Claims Excluded
Released		mm/dd/2006		License	GPL Version 3, (month) 2006

gear transmission ratio: 6:1

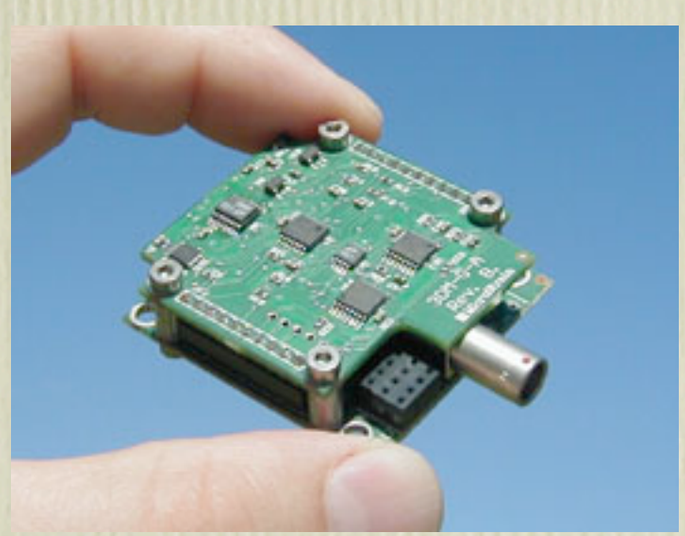
max. rotor speed: 1850 rpm

Brushless Motors

Power: 100W
Weight: 26g
Thrust: 600g

3 Gyros, 3 Accelerometers, and 3 Magnetometers

Microstrain 3DM-GX1
Dynamic orientation: gyros
Static orientation: accs, mags
Fusion: onboard programmable filter
I/O: RS-232, RS-485, analog output



10 Ultrasonic Sensors

Devantech SRF10 Sonar Ranger
Frequency: 40KHz
Range: 3cm-6m
I/O: I2C Bus

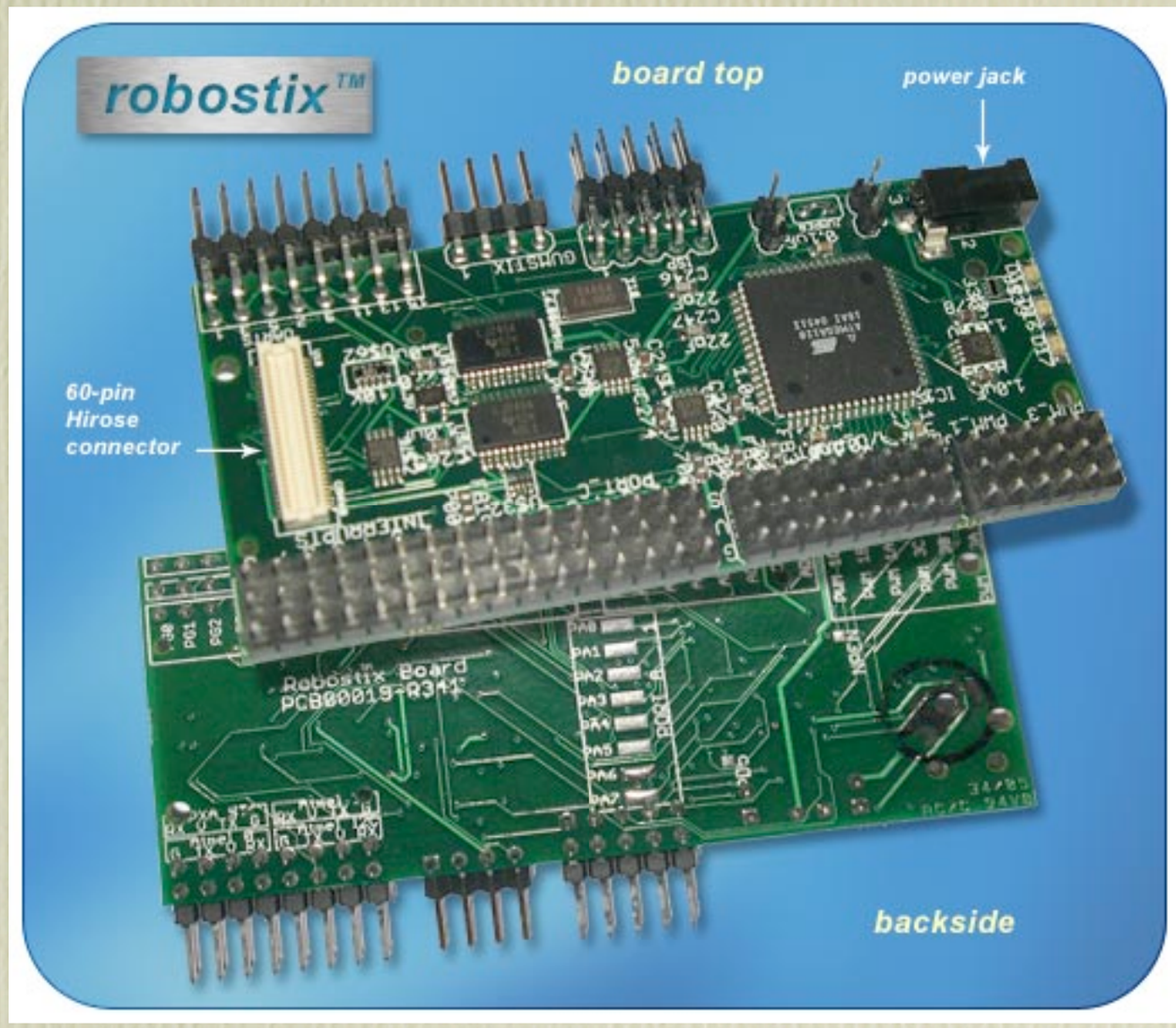
...but what about lasers?

Processor Board

Board: Gumstix
CPU: XScale 400MHz
RAM: 64MB
Flash: 16MB
Network: Bluetooth
OS: Linux 2.6

I/O Board

Board: Robostix
Bus: I2C
I/O: 6 PWM,
8 A/D,
25 GPIO,
2 UART (Atmega)



Concurrency Model: Exotasks

- *exotasks* are individually garbage-collected software tasks that communicate by message passing through so-called *pods*

- each exotask has its own private heap and fully preemptable garbage collector

- exotasks may allocate memory and mutate their pointer structures

- exotasks may neither observe global mutable state nor their mutable state may be observed

Implementation: Real-Time GC + E Code

- exotasks will be compiled into E code (the timing part) and dynamically scheduled and garbage collected (the functional part)
- exotasks with LETs may also be compiled into *G code* (schedule-carrying code extended by garbage-collecting instructions [M. Harringer, MSc Thesis, University of Salzburg, 2005])

Thank you