

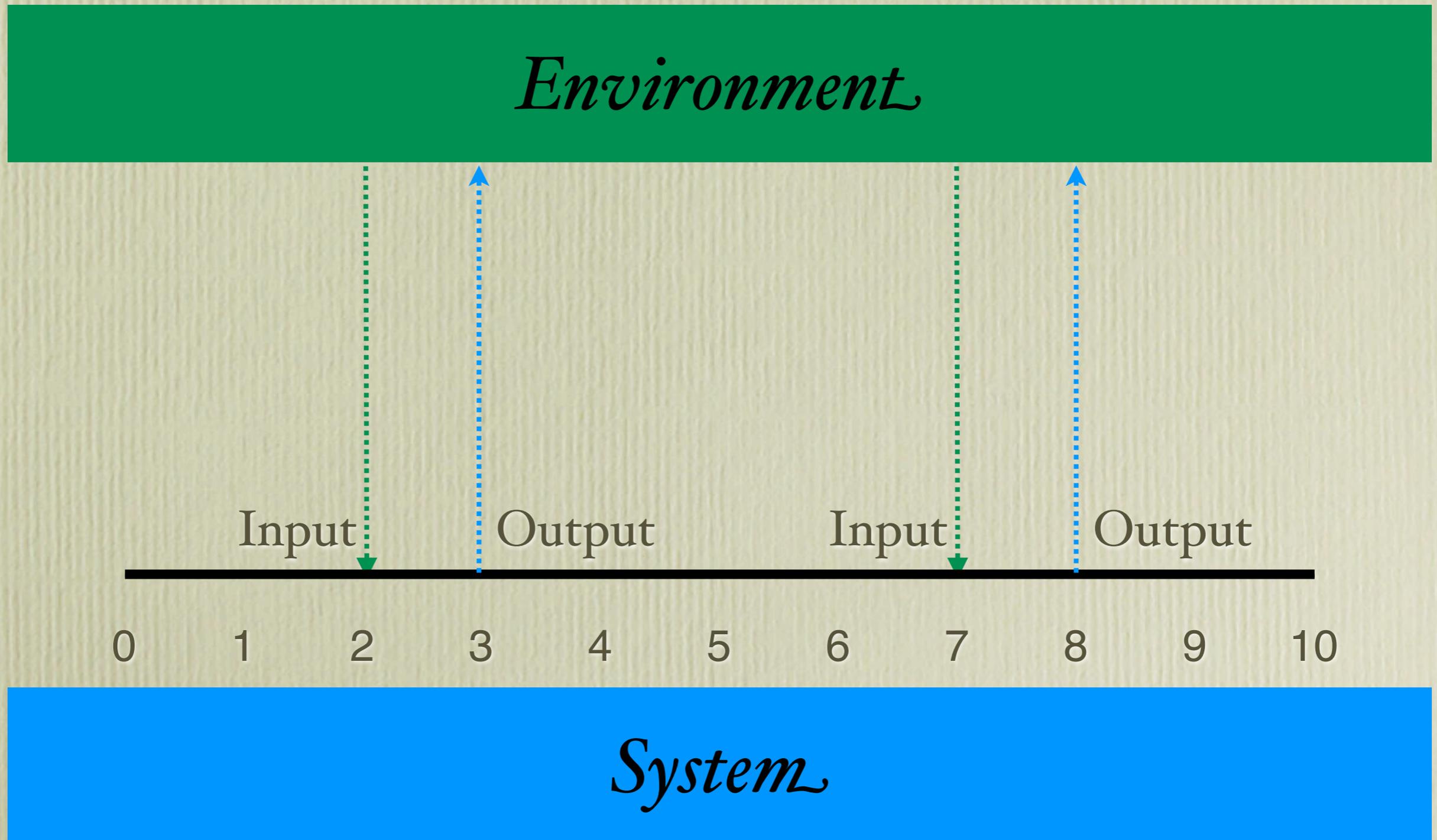
High-Level Programming of Real-Time Software Systems

Christoph Kirsch
Universität Salzburg



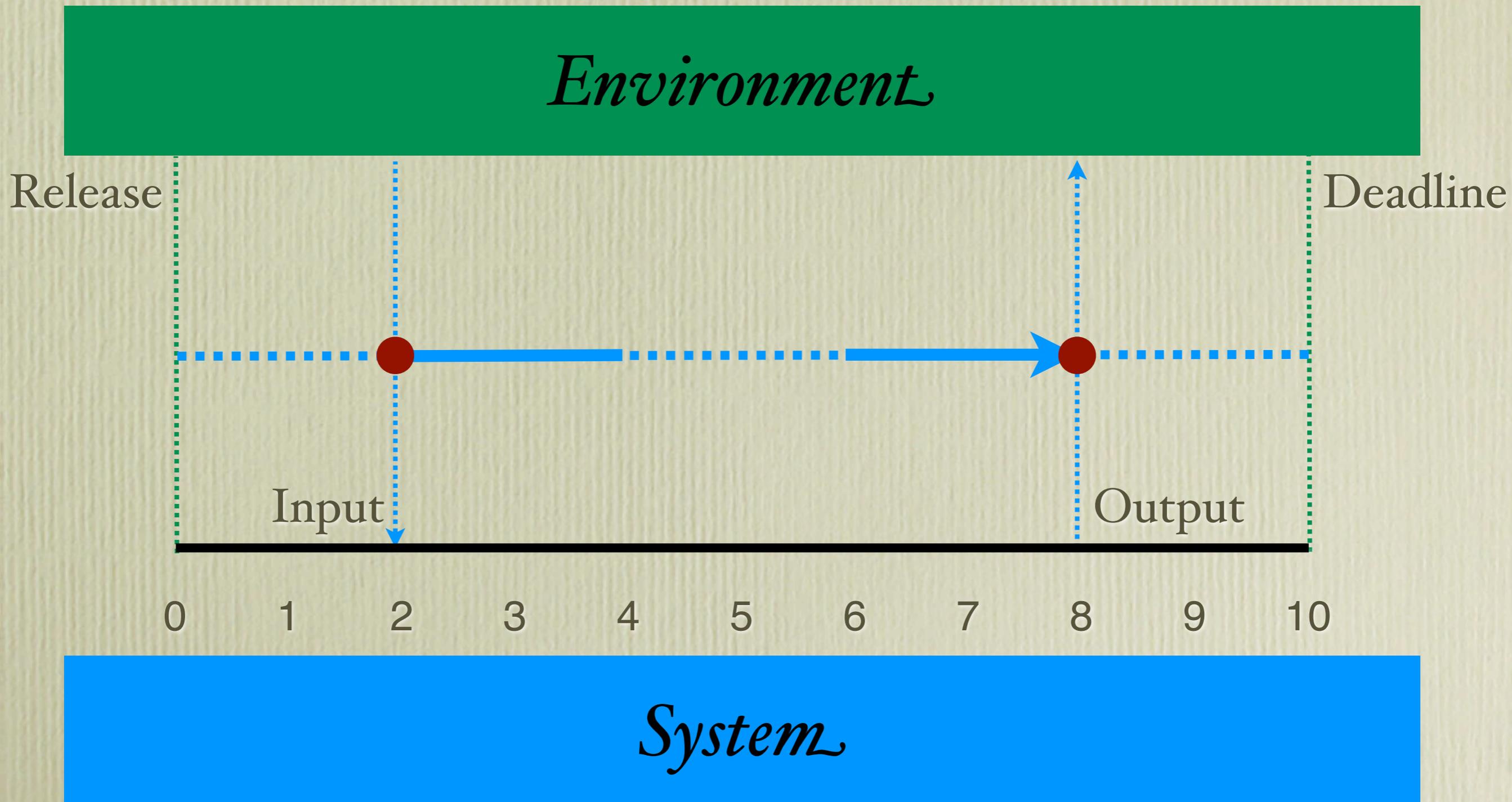
Università della Svizzera italiana, March 2006

Real-Time Programming

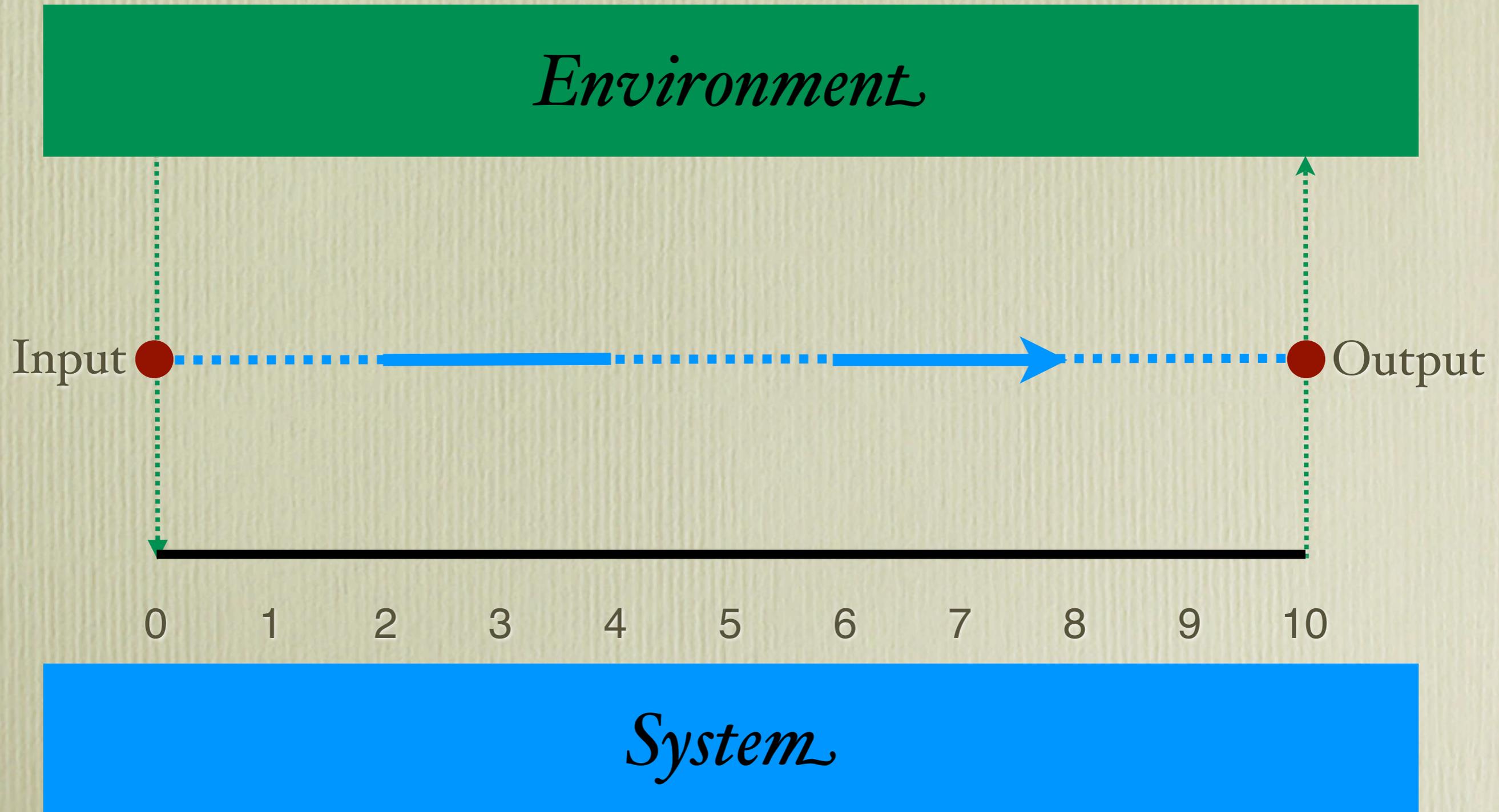




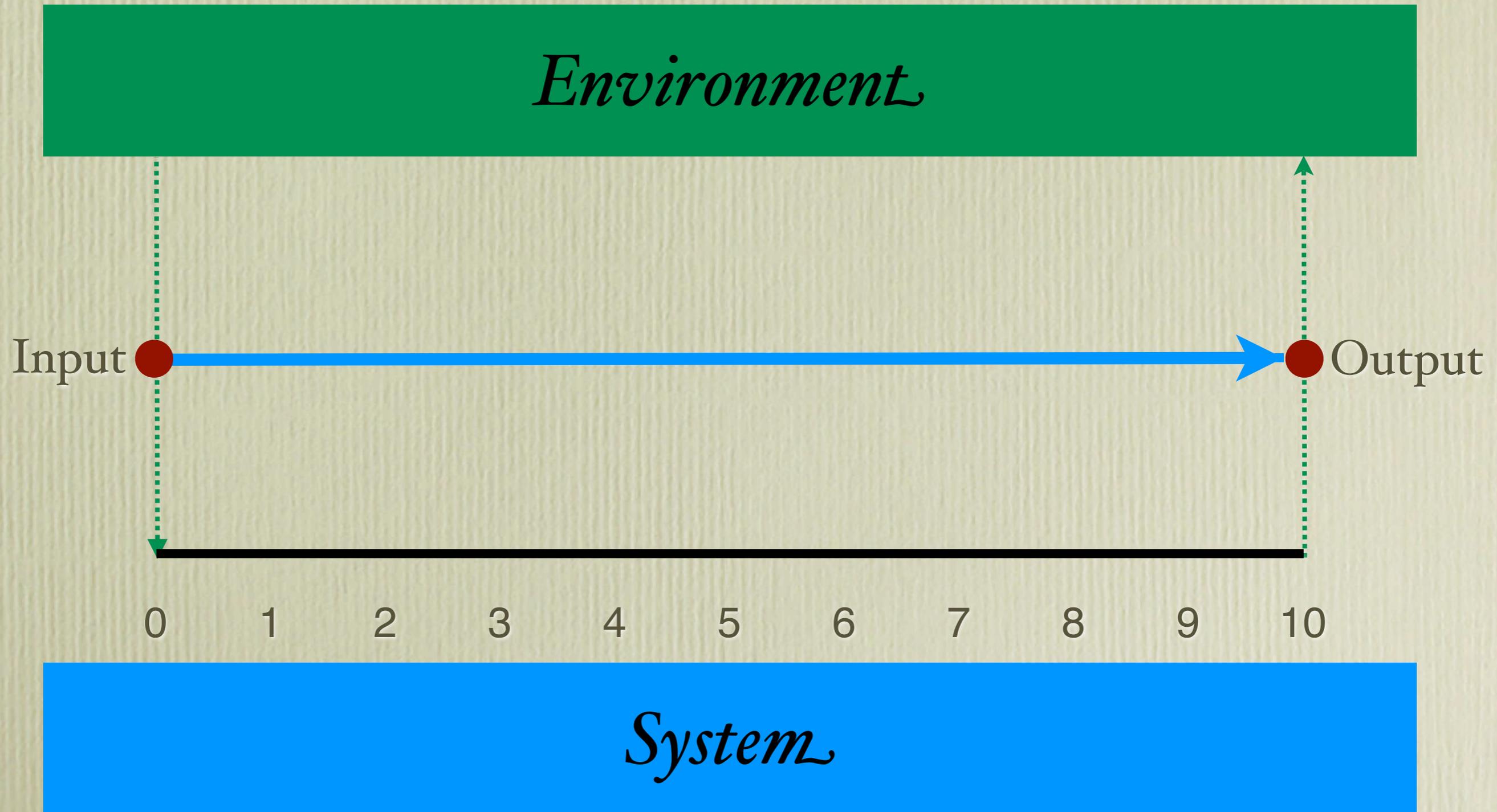
RT Programming Tradition



Logical Execution Time (LET)



Logical Execution Time (LET)



Logical Execution Time (LET)

Environment

Input ● Programming as if there is enough CPU time ● Output

0 1 2 3 4 5 6 7 8 9 10

System

Logical Execution Time (LET)

Environment

Input ● Programming as if there is enough CPU time → ● Output

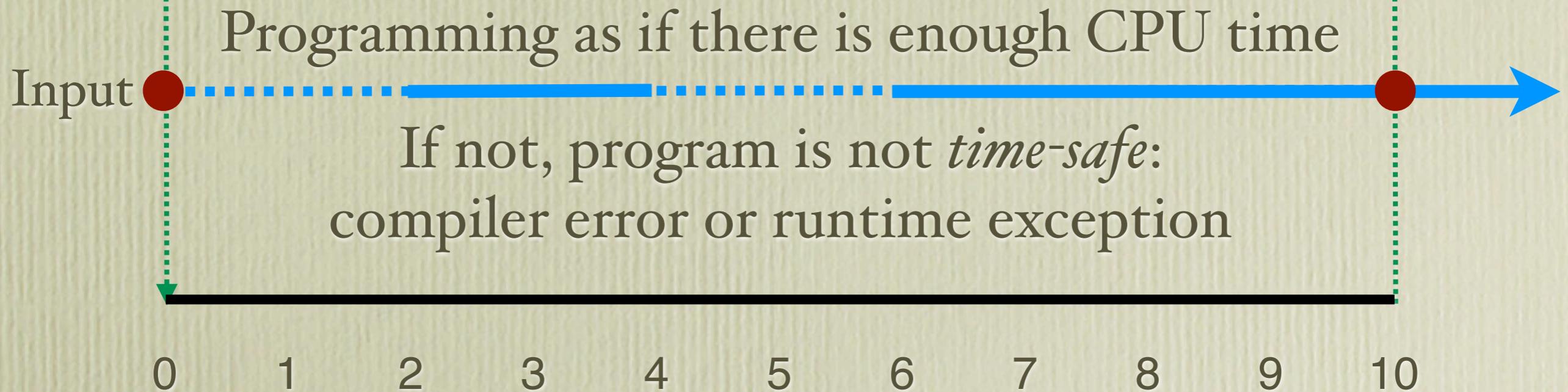
Compiler checks if there is enough CPU time

0 1 2 3 4 5 6 7 8 9 10

System

Logical Execution Time (LET)

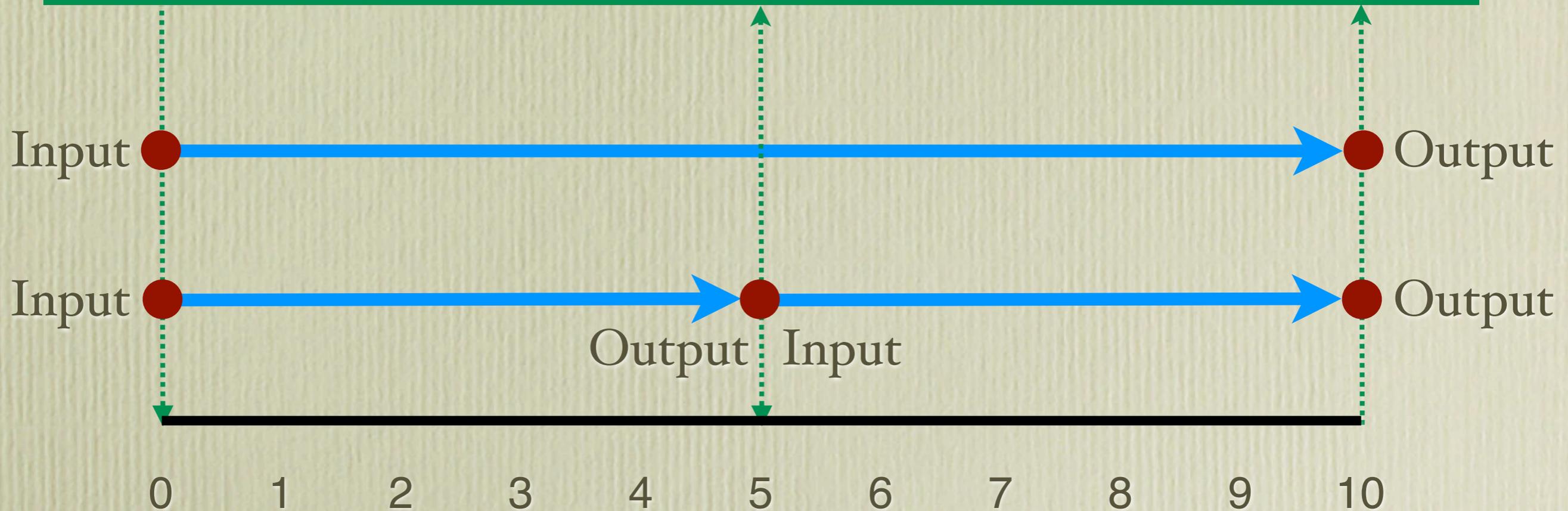
Environment



System

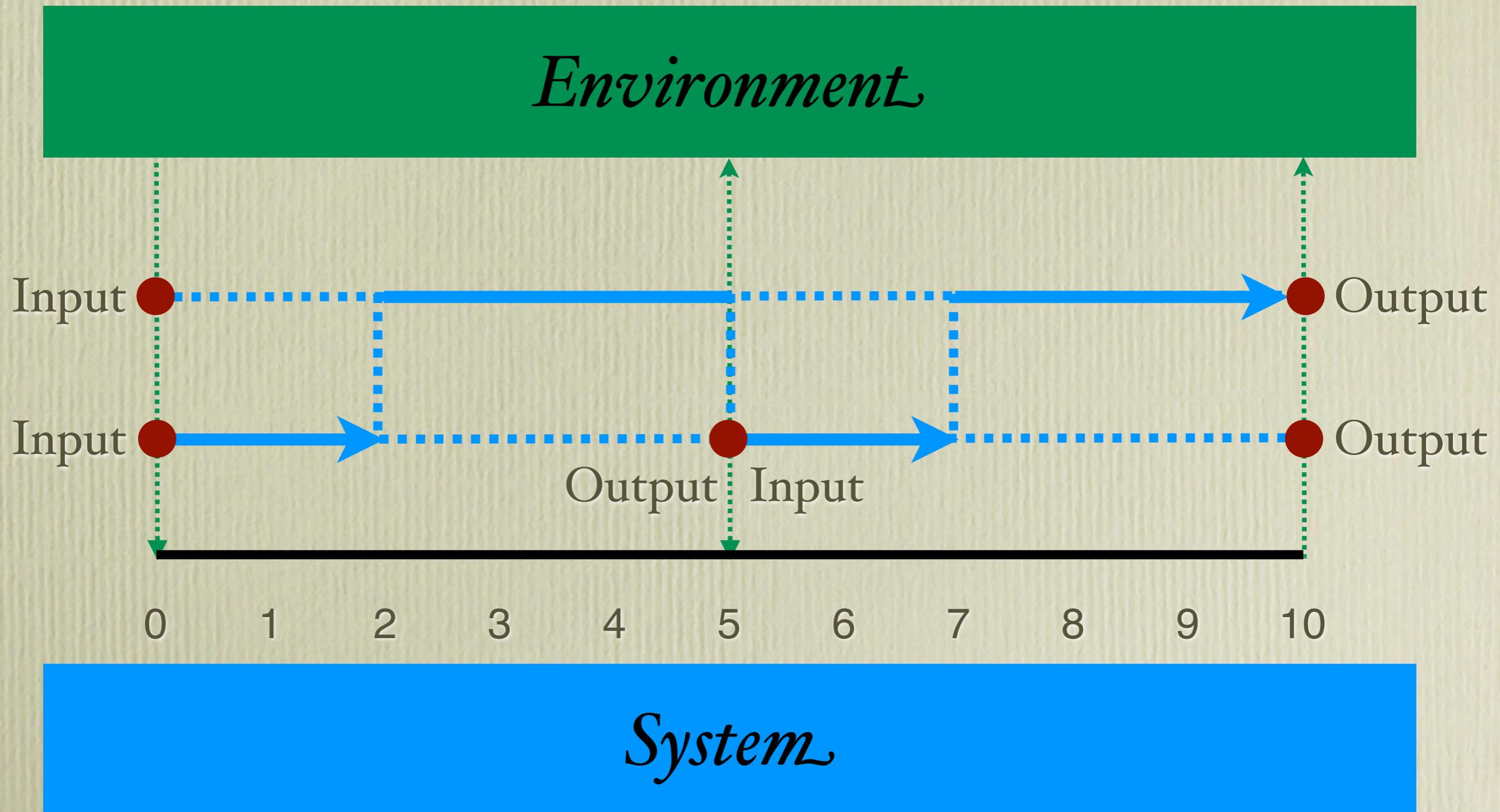
LET Programming

Environment

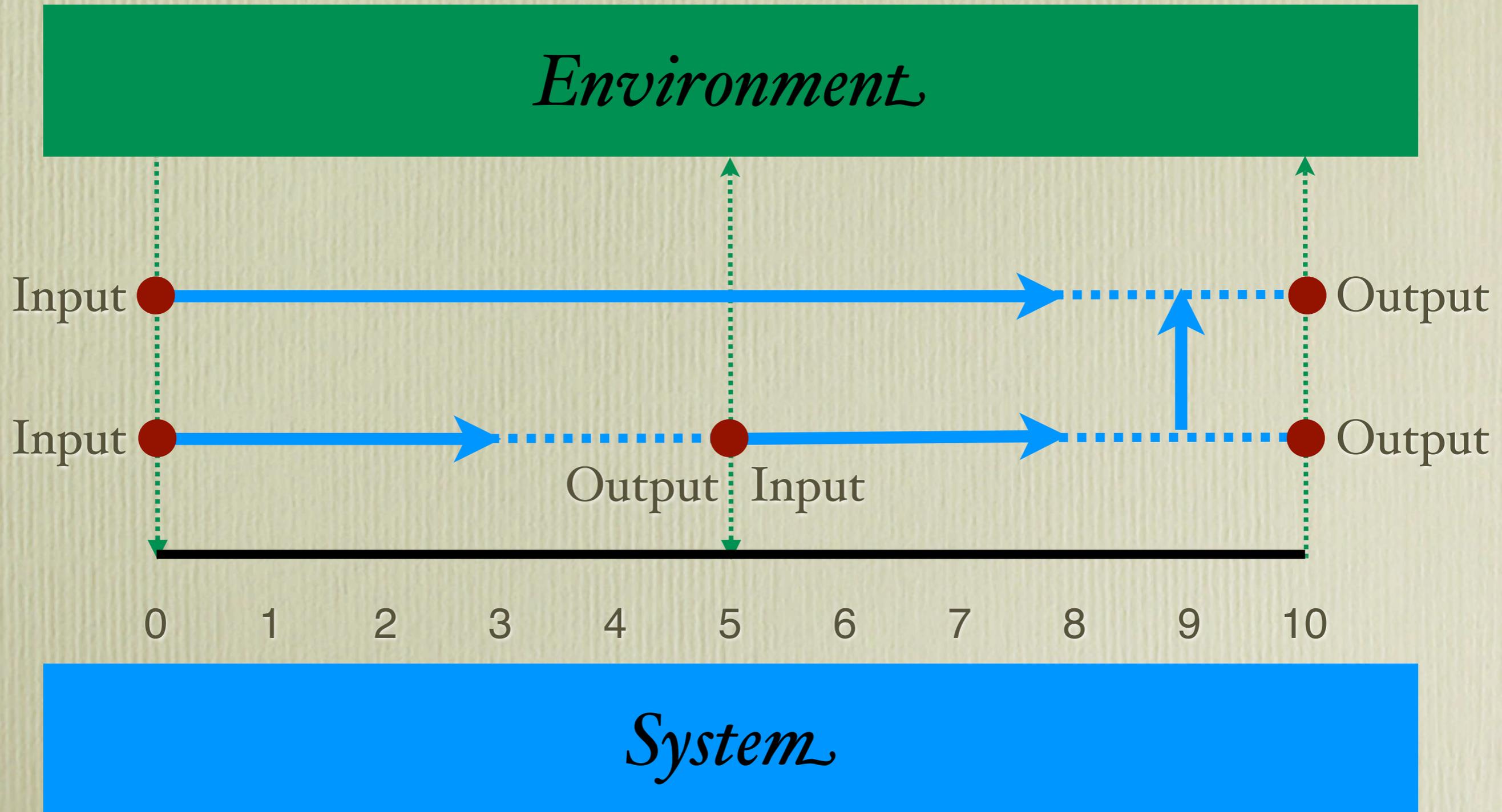


System

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

Simulink

“From Control Models
to Real-Time Code”

{IEEE CSM, 2003}

Giotto

Runtime System



Tool Chain

Simulink

Giotto

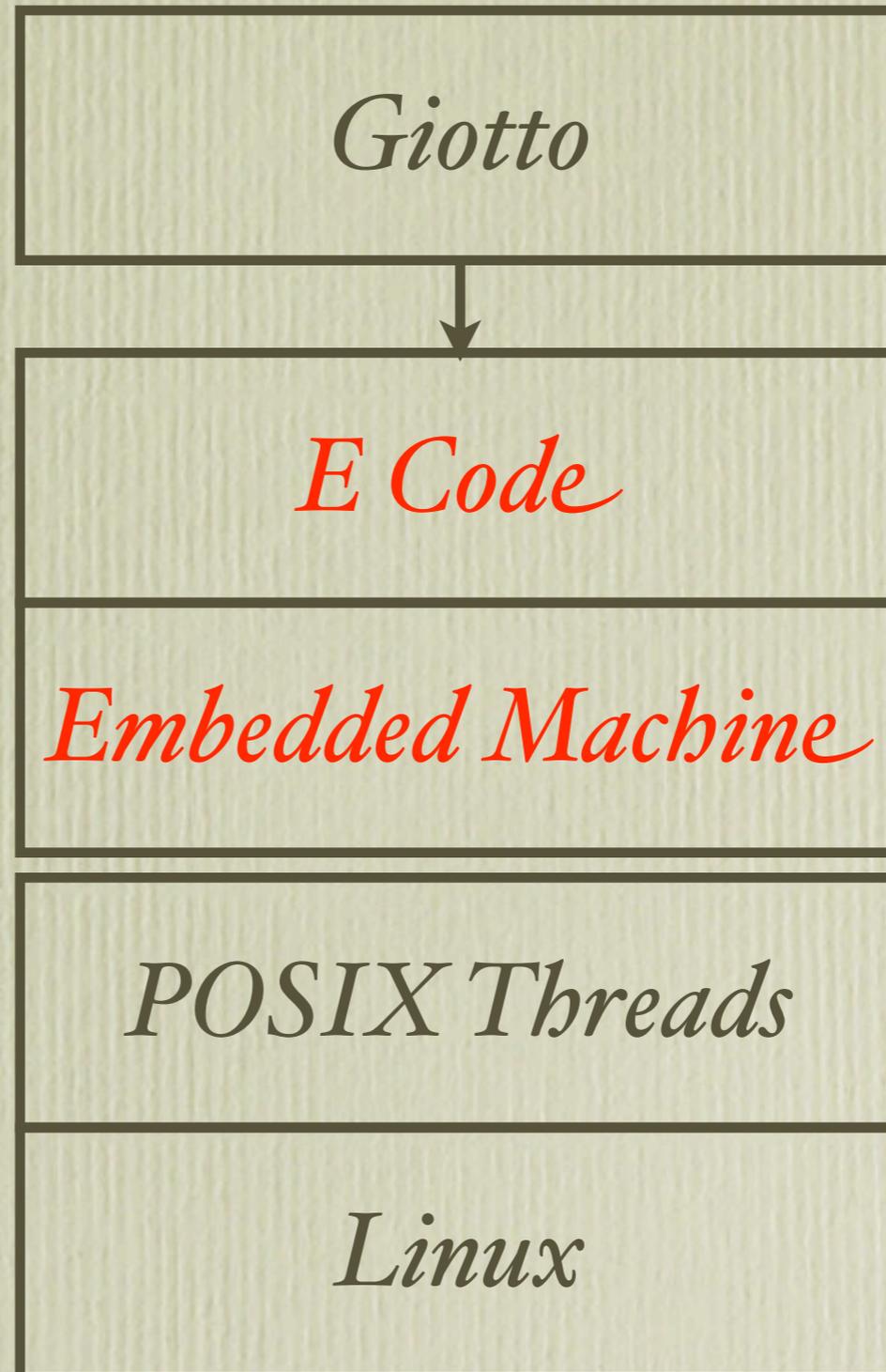
“A Giotto-Based Helicopter
Control System”

[EMSOFT, 2002]

Runtime System



Runtime System

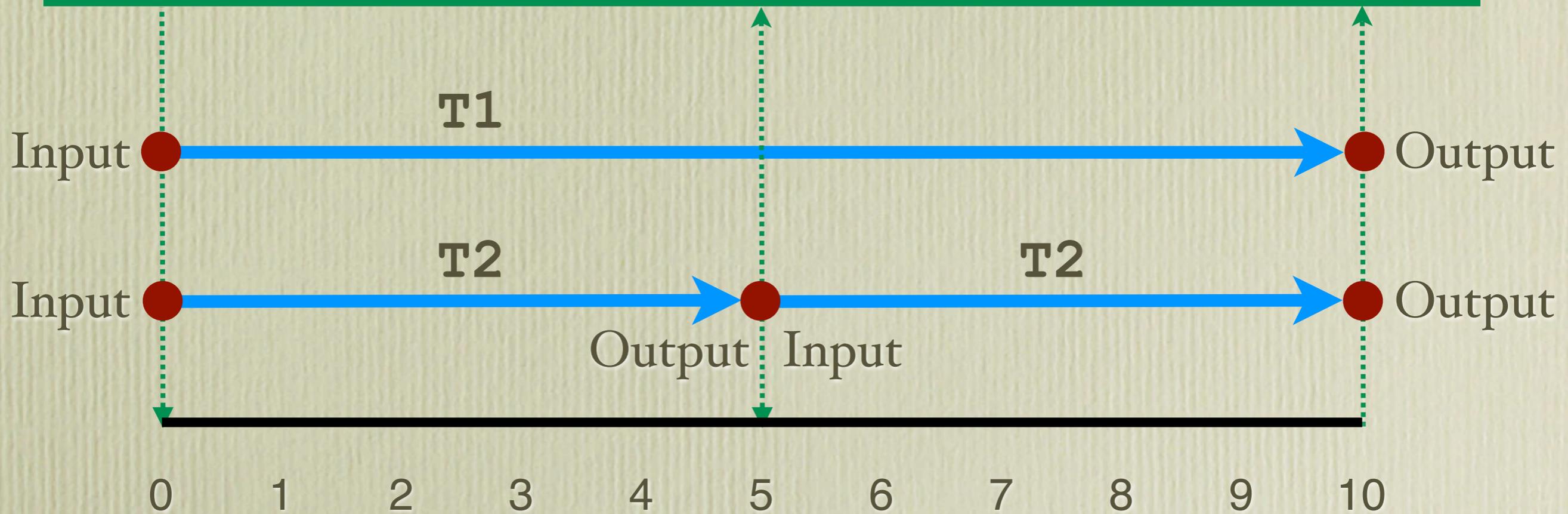


“The Embedded Machine:
Predictable, Portable
Real-Time Code”

[PLDI, 2002]

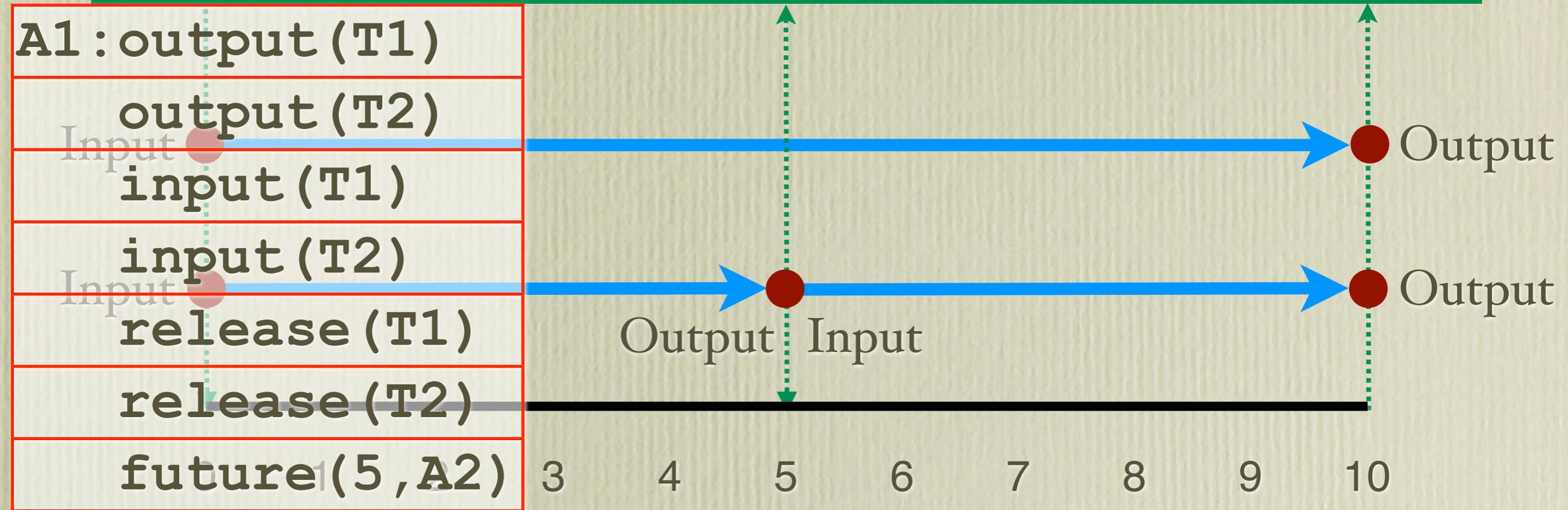
E Code

Environment



E Code

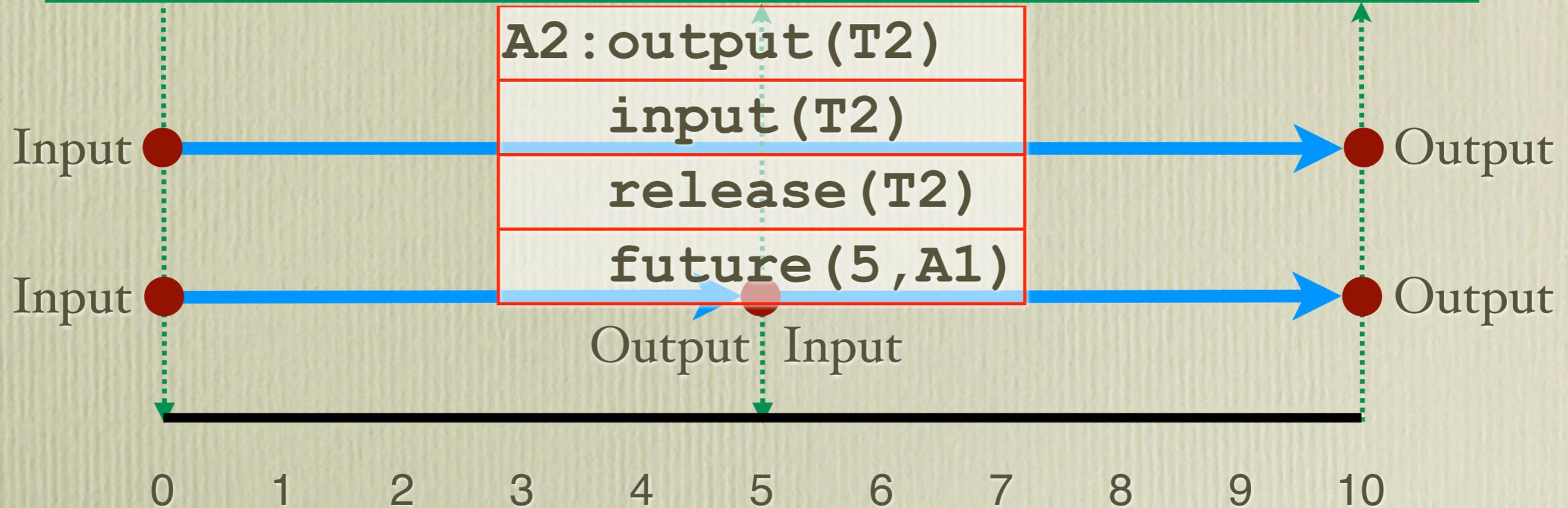
Environment



System

E Code

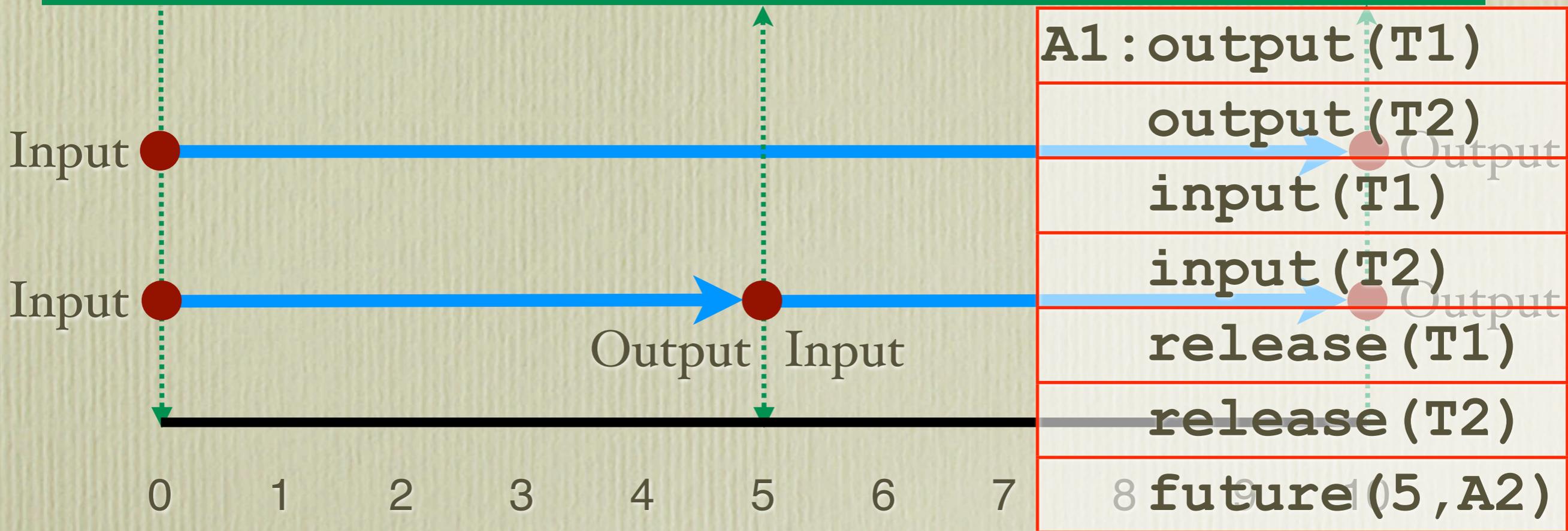
Environment



System

E Code

Environment





Schedule-Carrying Code

*Schedule-Carrying
Code*

E+S Machine

POSIX Threads

Linux

{EMSOFT, 2003}

*Schedule-Carrying
Code*

E+S Machine

Microkernel

StrongARM

{VEE, 2005}

*Schedule-Carrying
Code*

E+S Machine

RT Ethernet

RT Linux

{LCTES, 2005}



Current Projects



{w/ EPF Lausanne
& UC Berkeley}



{w/ IBM T.J. Watson}



{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)

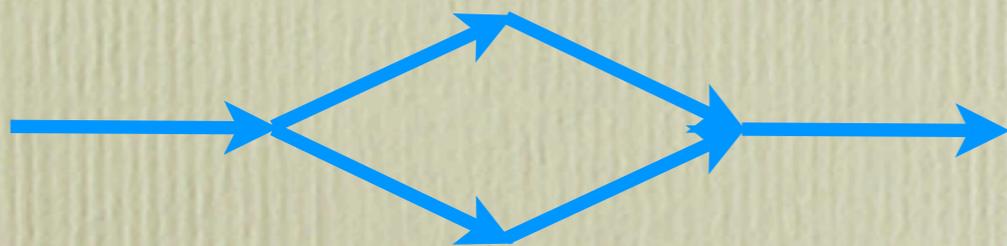


Parallel Composition

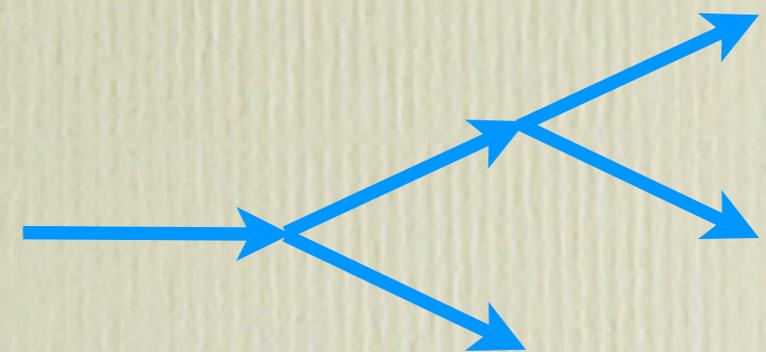
Program Module M₁



Program Module M₂

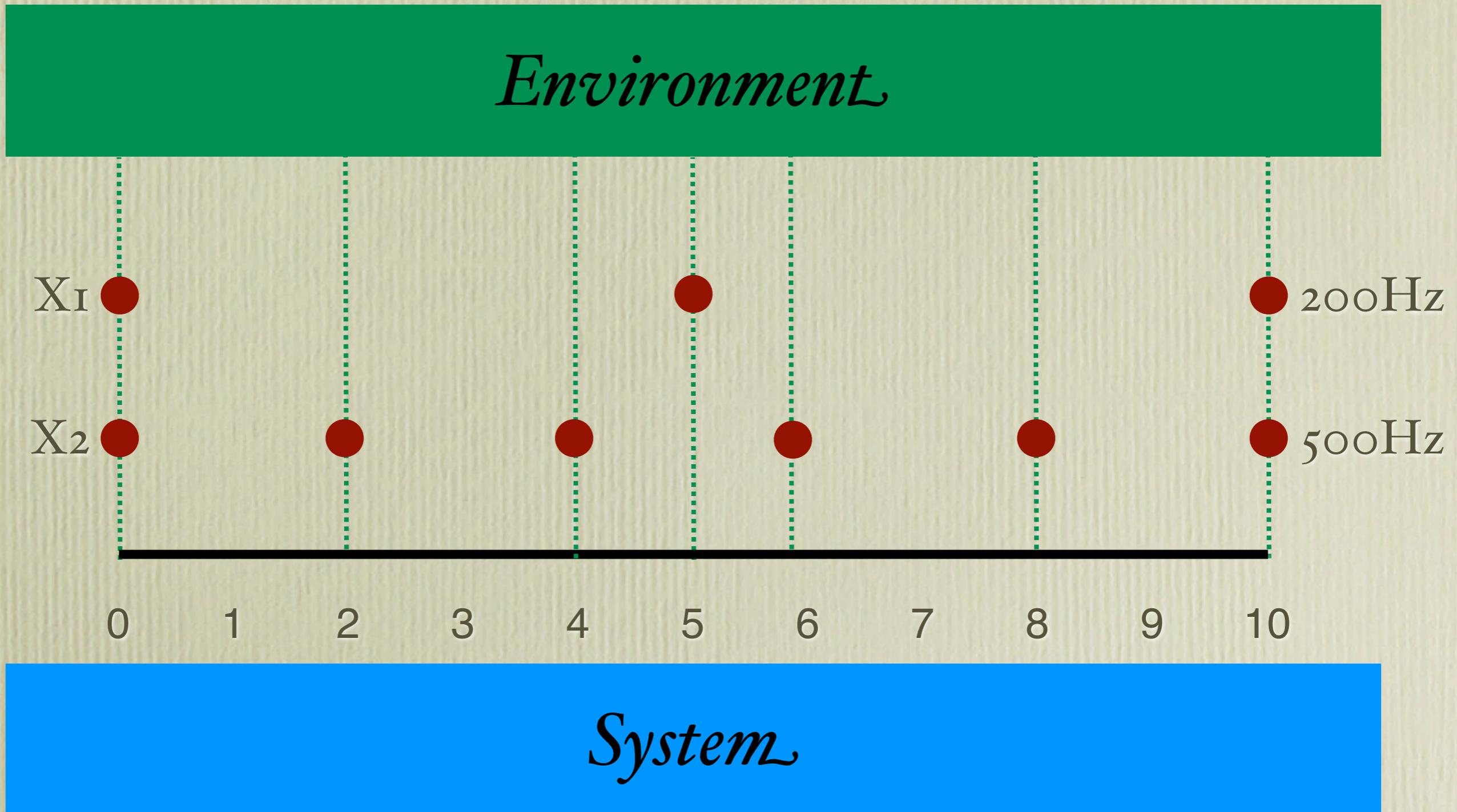


100Hz

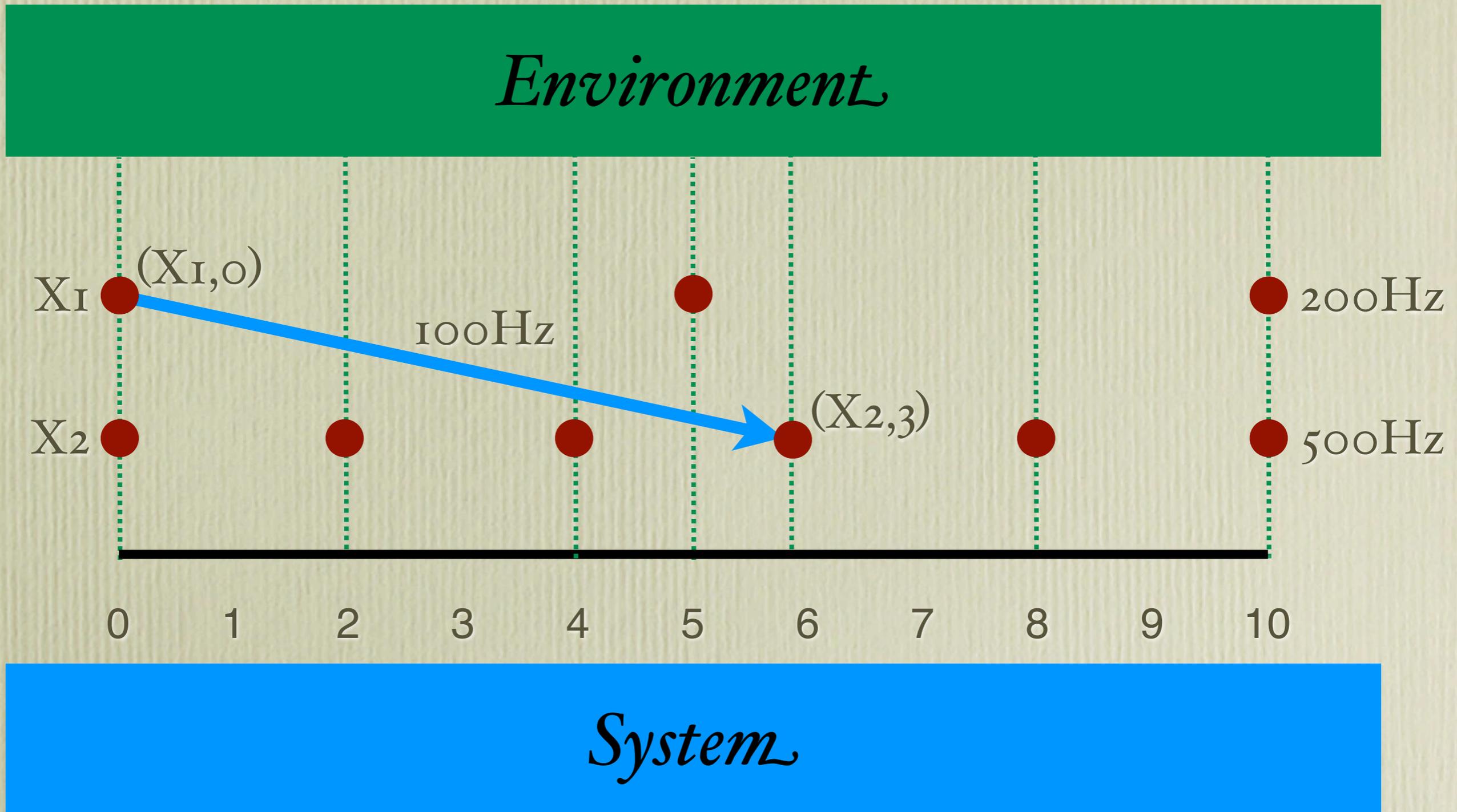


200Hz

Timed Variables

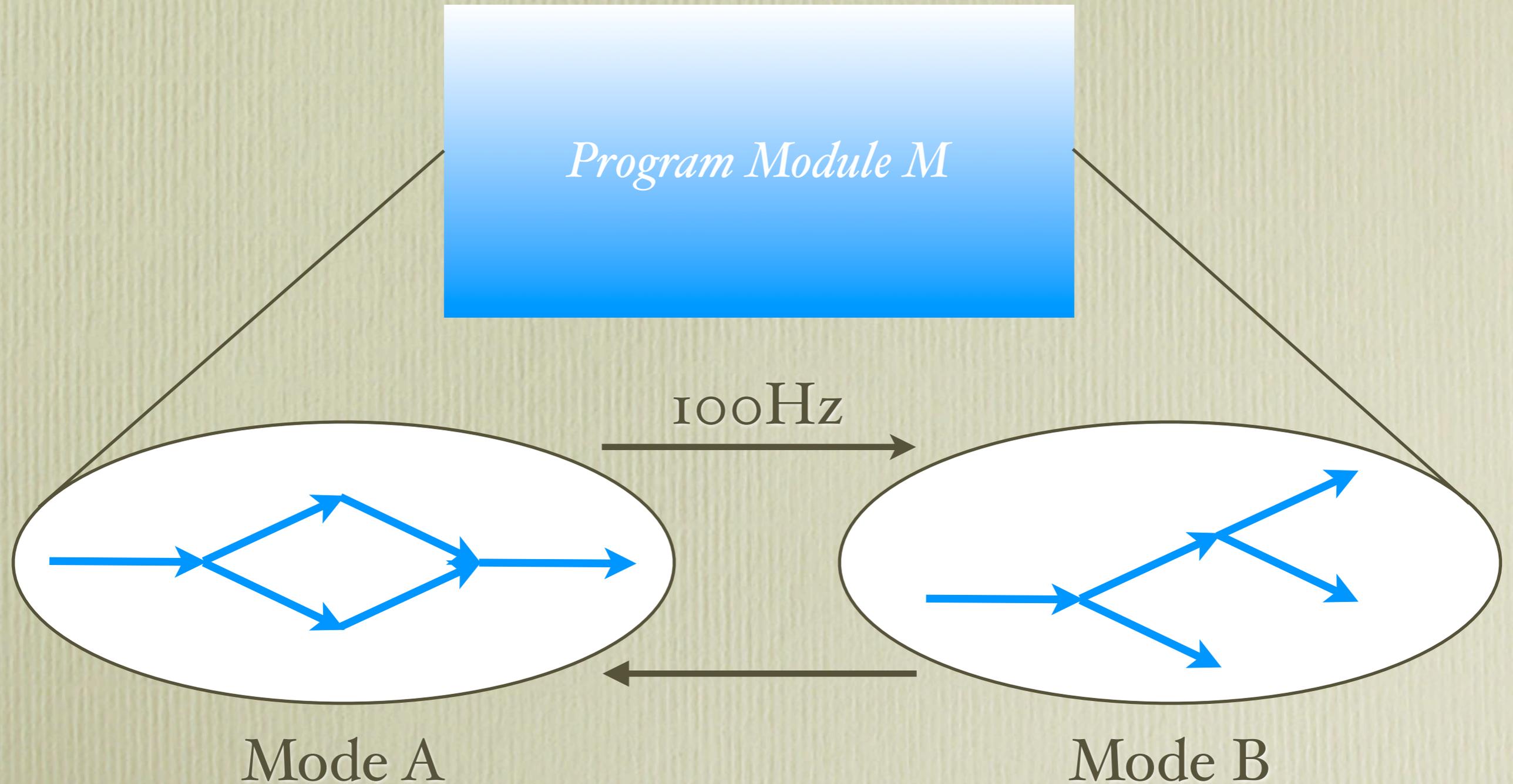


Timed Variables



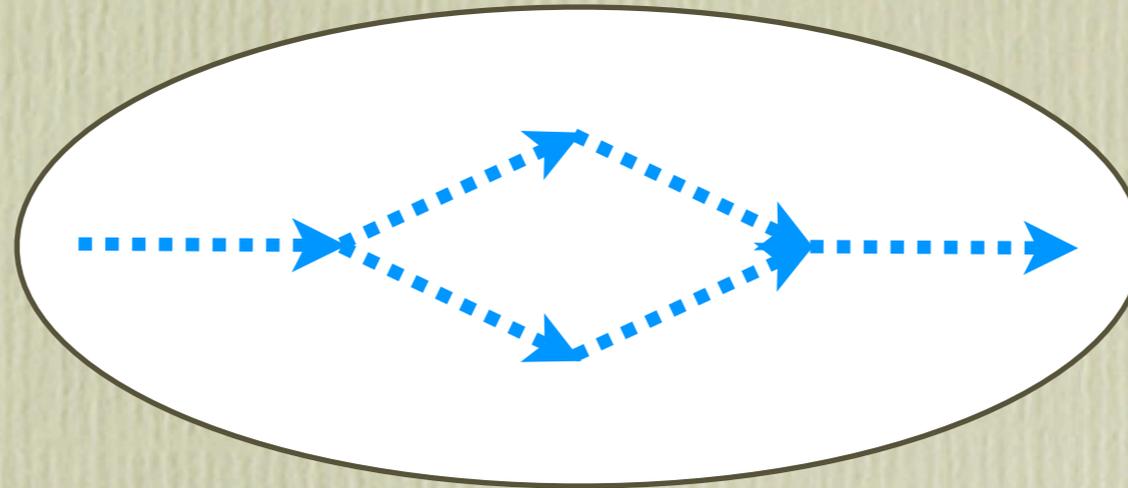


Sequential Composition

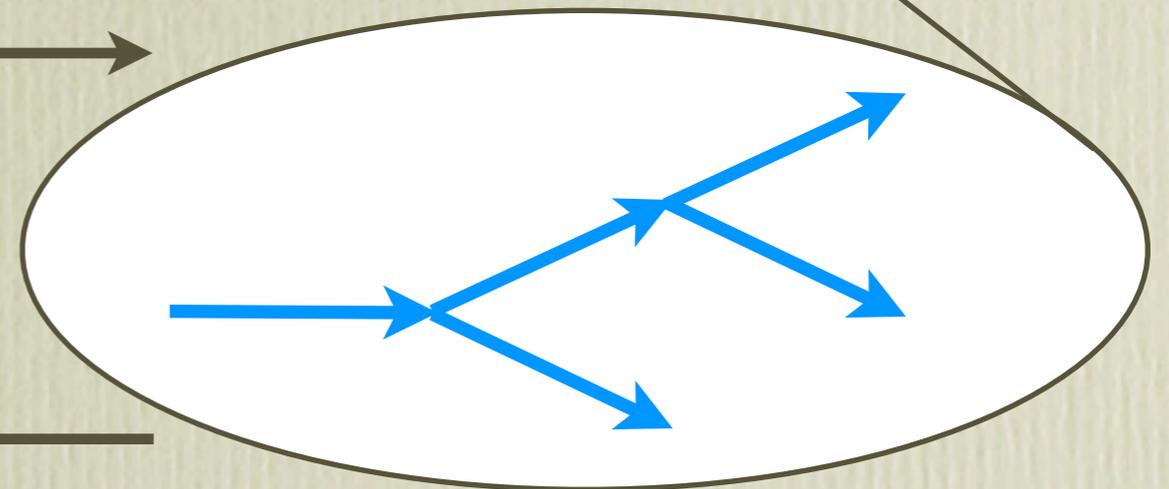
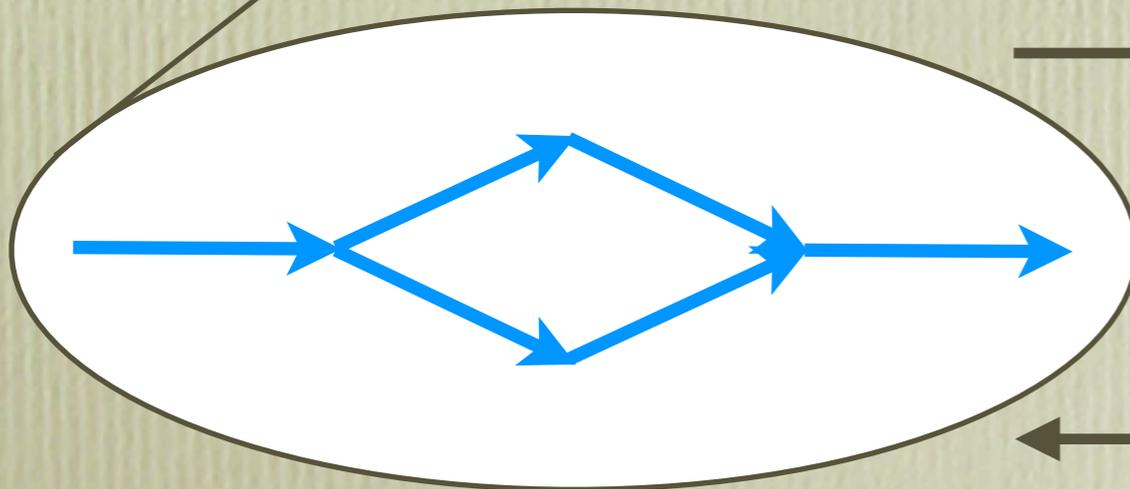




Program Refinement

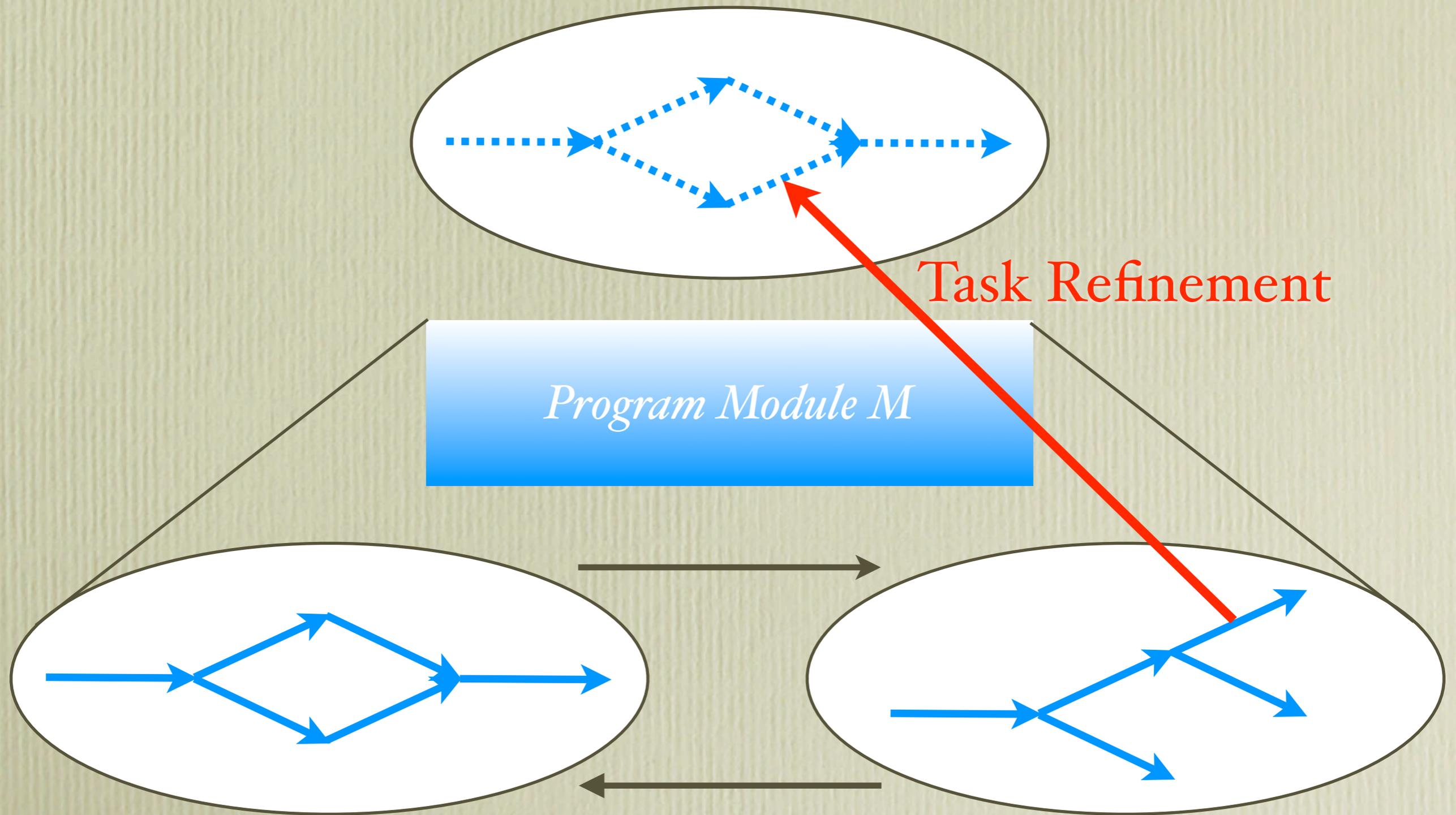


Program Module M

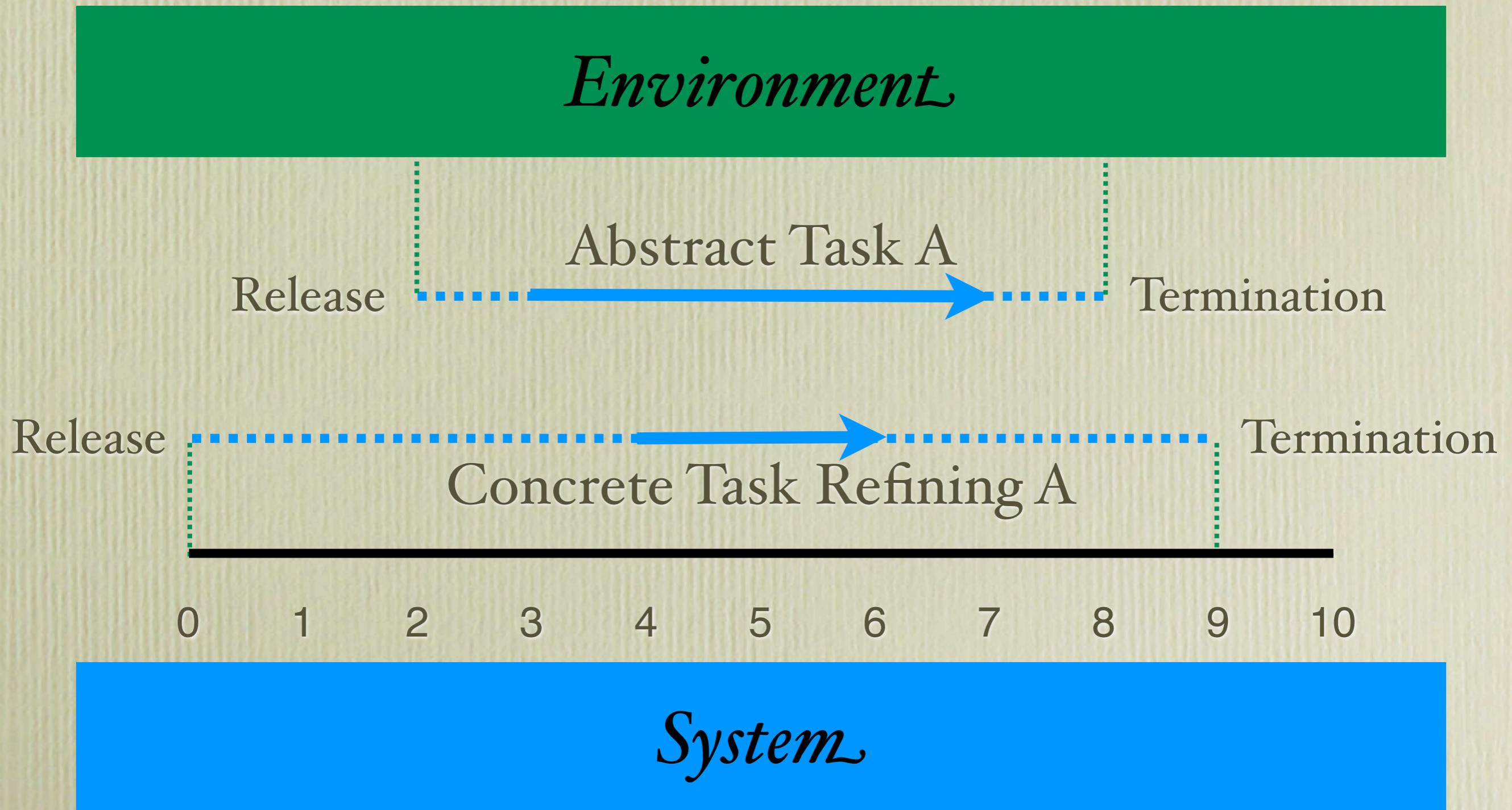




Program Refinement



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 .*



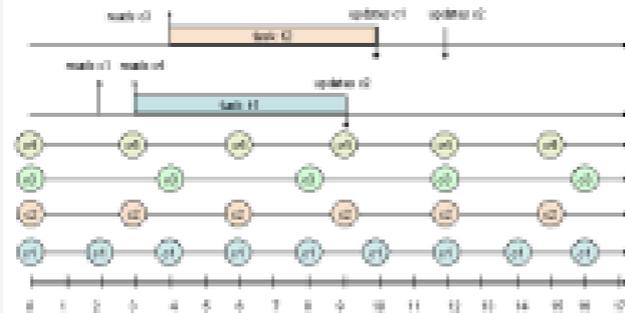
Hierarchical Timing Language (HTL) ...

... 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 schedulable in two dimensions without changing their timing behavior: new program modules can be added, and individual program tasks 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 at 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://chess.eecs.berkeley.edu/>

Communications and Tasks

A communication is a variable (with a fixed datatype) and can be accessed (i.e. read from and write to) only at specific time instances. It is updated by the time instance through a communication port. A task is a block of sequential code (without any internal synchronization) that gives certain inputs positive outputs. A task starts from certain instances of some communication and updates certain instances of the same or other communication.

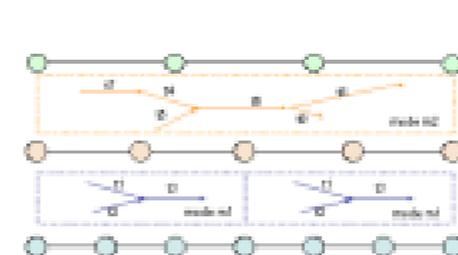


The figure shows four communication variables, c_1, c_2 , and c_3 and of task periods 1, 2, 3, 4 and corresponding activation times t_1 and t_2 . The instances of access for the communication have been shown along the time line. Task 1 reads c_1 and updates c_2 and c_3 . Task 2 reads c_1 and updates c_2 and c_3 . Task 3 reads c_1 and updates c_2 and c_3 . Task 4 reads c_1 and updates c_2 and c_3 . The read and write instances of the communication specify an activation window for the tasks. Read and write instances of c_1 and c_2 and c_3 are shown in the figure. Task 1 reads c_1 and updates c_2 and c_3 at time t_1 and t_2 respectively. Task 2 reads c_1 and updates c_2 and c_3 at time t_3 and t_4 respectively. Task 3 reads c_1 and updates c_2 and c_3 at time t_5 and t_6 respectively. Task 4 reads c_1 and updates c_2 and c_3 at time t_7 and t_8 respectively.

Core Program Structure

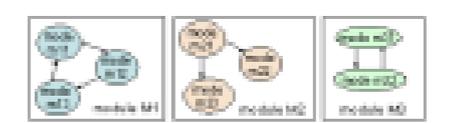
An HTL program is a set of communication and a set of groups of tasks. Tasks are grouped (hierarchical) relative to an **abstract** task because there can be procedure (hierarchical) tasks. The task procedure is expressed through **ports** (i.e. variables with a fixed datatype). If a task procedure contains tasks, then the second task must read a port updated by the first task. The ports are not bound to time instances (i.e. at some at a port is updated by a task, another task (from the same mode) may read the port).

A mode is a periodicity of tasks in a group repeat with the periodicity of the containing mode. Tasks of the mode can only communicate through communication. A task in a mode specifies the ports and the instances of communication (relative to the mode period) it reads from and writes to.

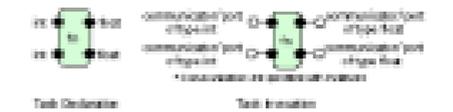


Extending core structure

HTL allows mode and abstract task mode can define to other mode based upon a condition specified as a predicate on ports of the mode and communication. Each a network of mode in a module. Any instance, task of almost one mode of a module may be executing. In the extended model, an HTL program specifies a set of modes and a set of communication. The modes are composed in parallel with mode in a module are composed sequentially.



Abstract Task and Concrete Task



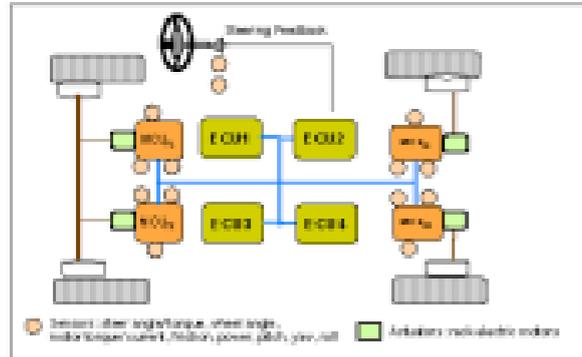
Abstract tasks have to function definition, but only interface definition. Concrete tasks have both function definition and interface definition.

Refinement

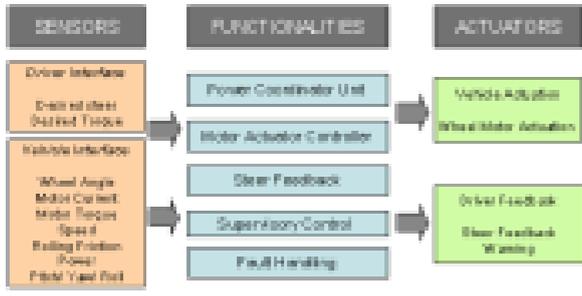


- period of m_2 is same as that of mode m_1 .
- every task invocation of m_2 maps to an unique task invocation (defined as a subtask) of m_1 .
- parent task should be an abstract task.
- latest communication port of a task should be equal to or earlier than that of the parent task.
- earliest communication port of a task should be later or later than that of the parent task.
- WCET and WCTT of a task should be less than or equal to the respective number for the parent task.
- and, precedence in m_2 should be contained in m_1 .

STEER BY WIRE



A steer-by-wire (SBW) control system replaces mechanical linkage between steering wheel and car wheels by a set of steering wheel angle sensors, electric motors that control the wheel angle, and a controller that computes the required wheel motor activities. To maintain a realistic road condition feel for the driver, a force feedback actuator is placed on the steering wheel. The specific architecture that has been used here is a simplified steer-by-wire model used by General Motors for their prototype hydrogen fuel-cell car F100.



The SBW case study has been implemented on eight AMD64 (dual-core) laptops with SBW hardware supported by a custom hardware board. The case study is written in C and compiled to Linux. The hardware is configured to run at 100 MHz and the software is configured to run at 100 MHz. The hardware is configured to run at 100 MHz and the software is configured to run at 100 MHz. The hardware is configured to run at 100 MHz and the software is configured to run at 100 MHz.

modules for steer-by-wire implementation



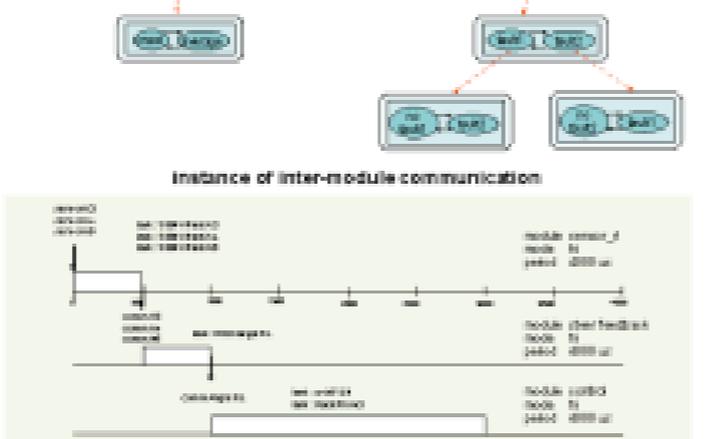
modes for steer-by-wire implementation



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 approximation of the mode 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 if compiler extensibility.

Distribution. HTL allows a programmer to specify the distribution of an HTL program. Different modes can run on different hosts. The programmer retains 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.

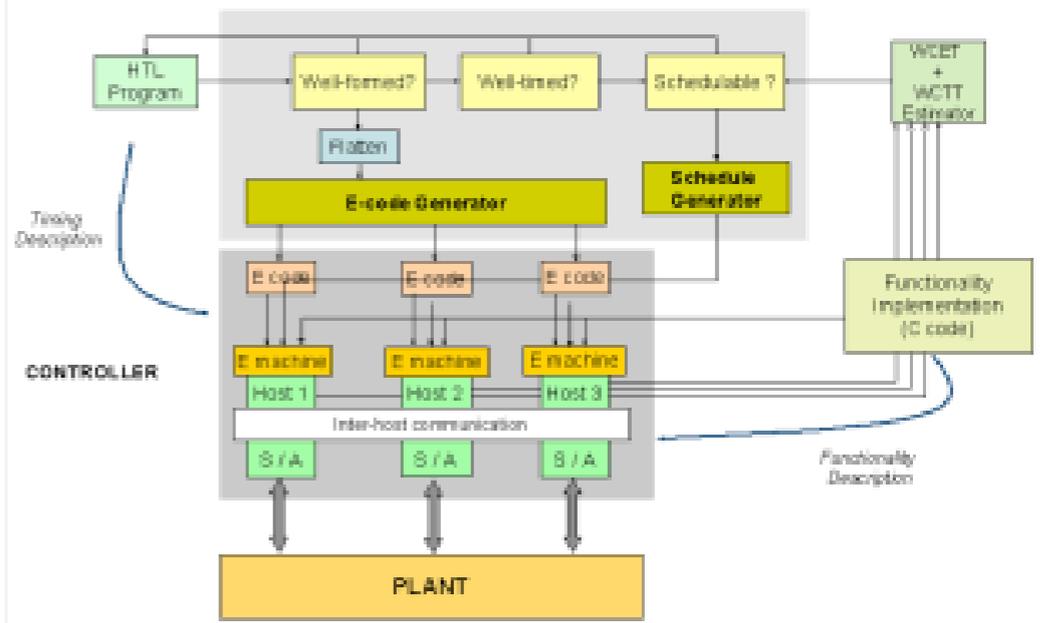
Composing Modules (Horizontal Extension)

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

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 leads to a constructive algorithm for the top-level of an HTL program. The problem is constructive algorithm, and proofs 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



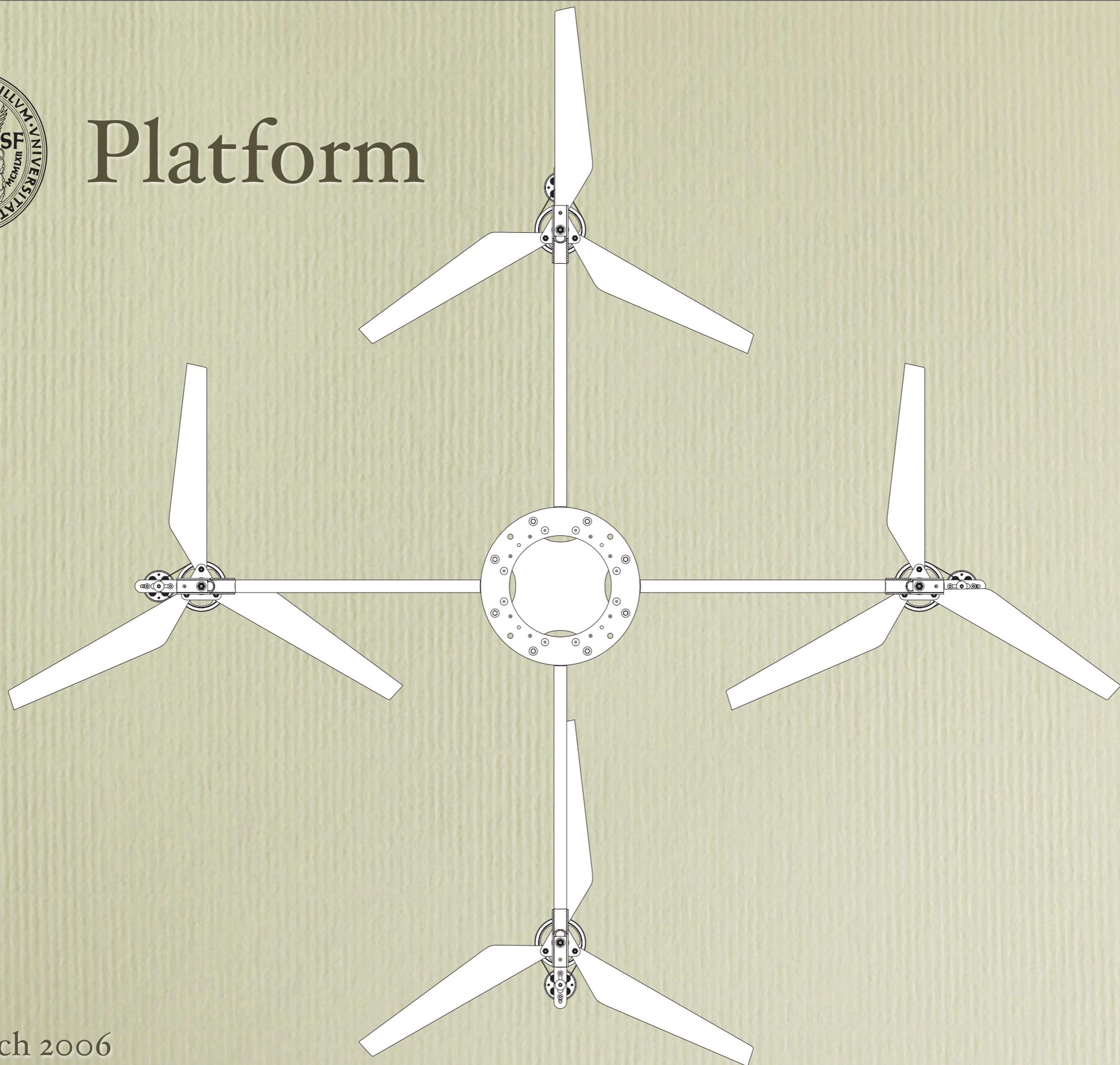
Collaboration

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

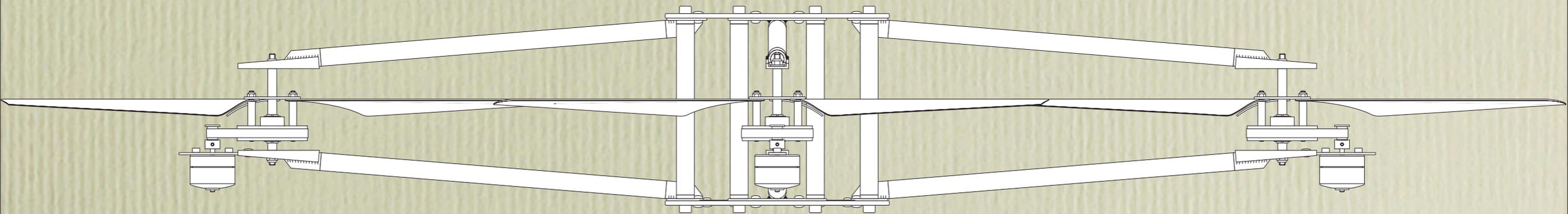


Platform





It's a 'Bicycle Wheel'

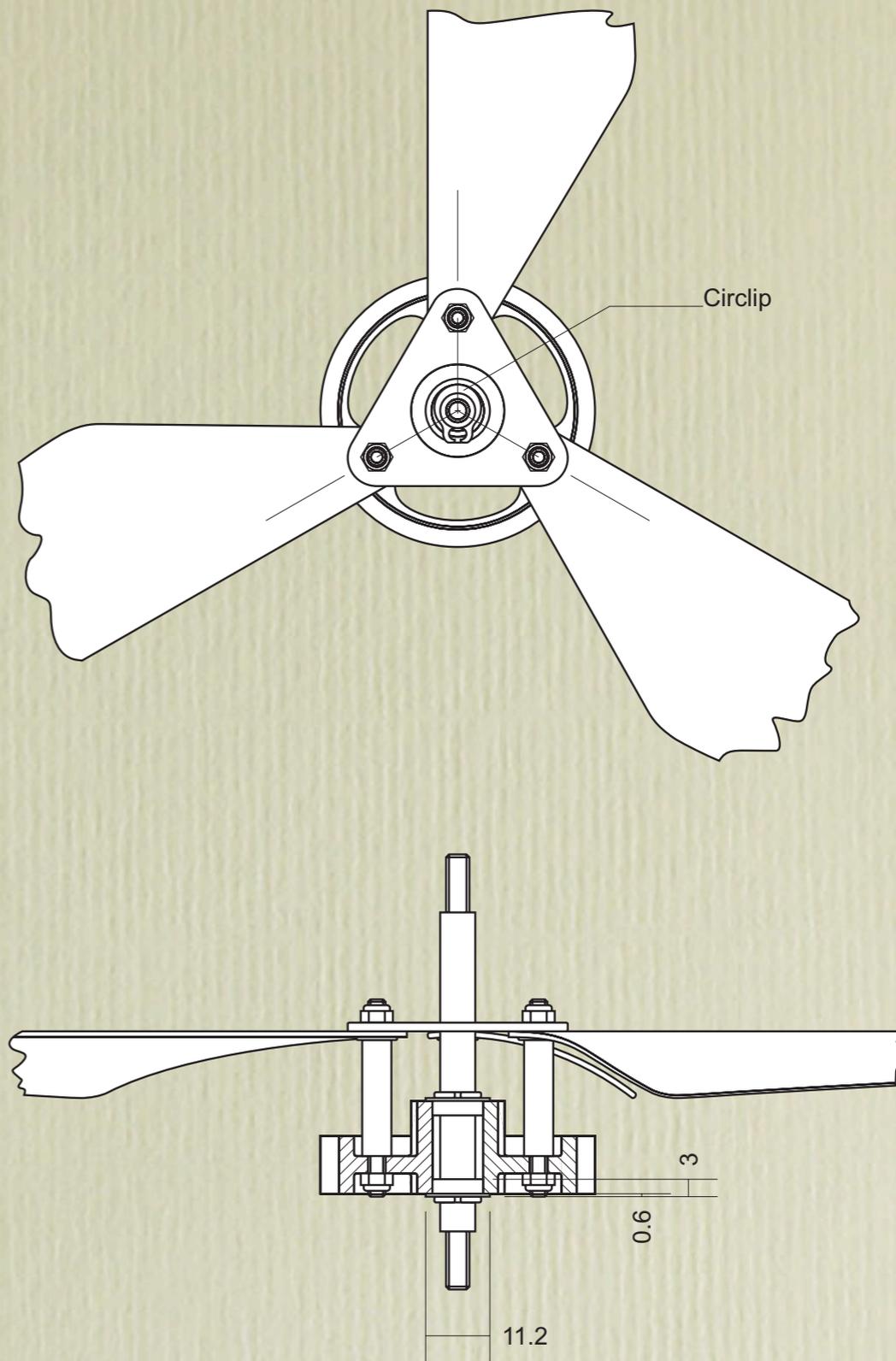




Design

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

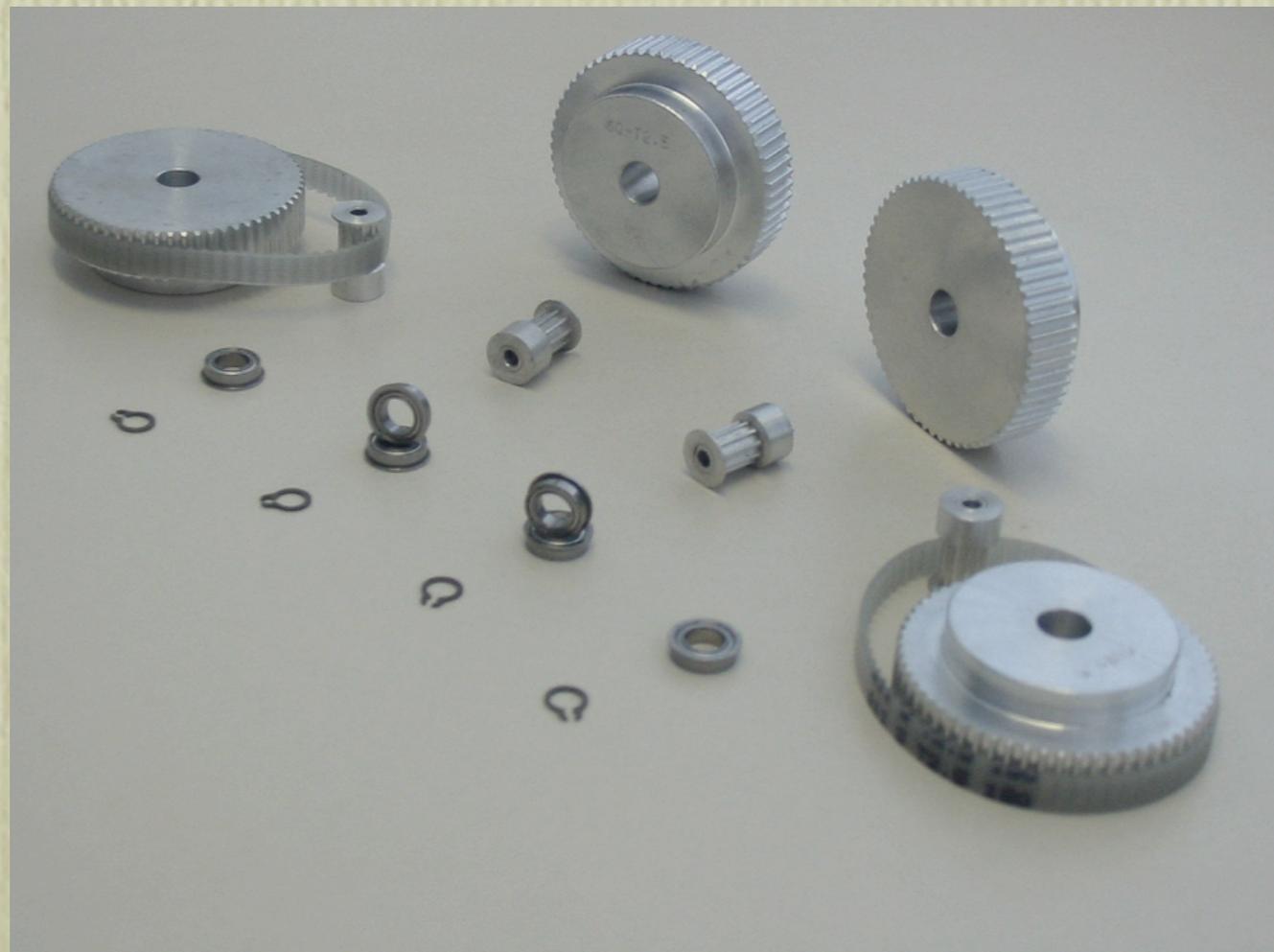
javiator.cs.uni-salzburg.at



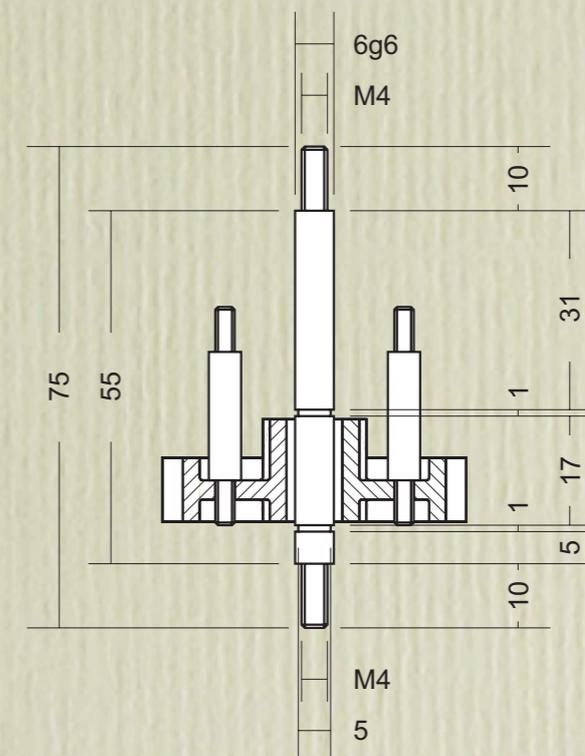
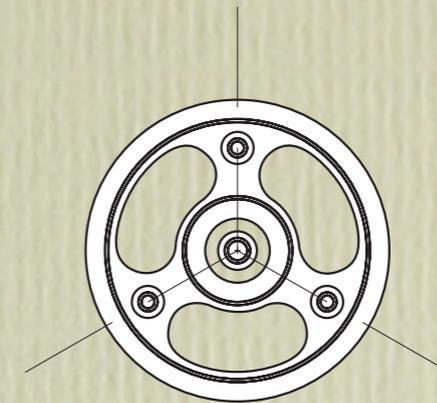
Drawing	Rotor Bearings DDLF-1060	Engineer	Rainer Trummer
Material	Stainless-Steel Alloy	Company	University of Salzburg
Units	Scale	Millimeters	1:1
Project	JAviator Quadrotor	Department	Computer Science
Created	01/07/2006	Copyright	(c) 2006 Rainer Trummer
Released	mm/dd/2006	Disclaimer	All Liability Claims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



Weight..less



gear transmission ratio: 6:1
 max. rotor speed: 1850 rpm



Drawing	Rotor Axle		Engineer	Rainer Trummer
Material	Titan Alloy TiAl6V4		Company	University of Salzburg
Units	Scale	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



Brushless Motors

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





3 Gyros, 3 Accelerometers, and 3 Magnetometers



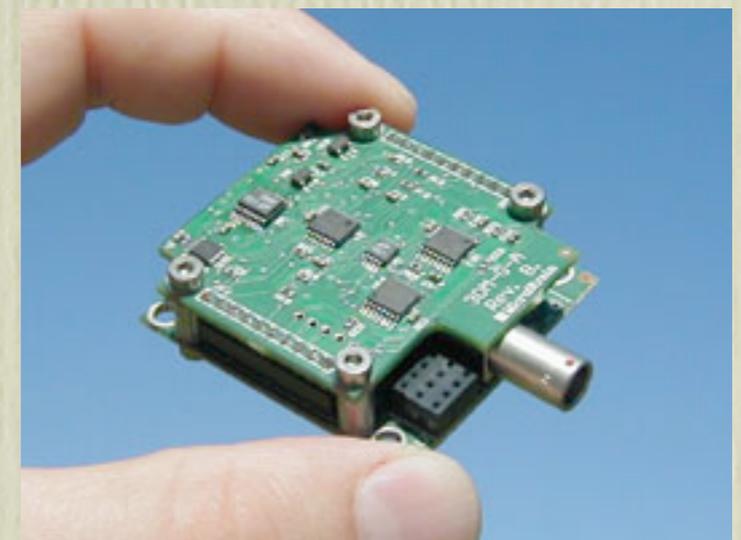
Microstrain 3DM-GX1

Dynamic orientation: gyros

Static orientation: accs, mags

Fusion: onboard programmable filter

I/O: RS-232, RS-485, analog output





IO Ultrasonic Sensors

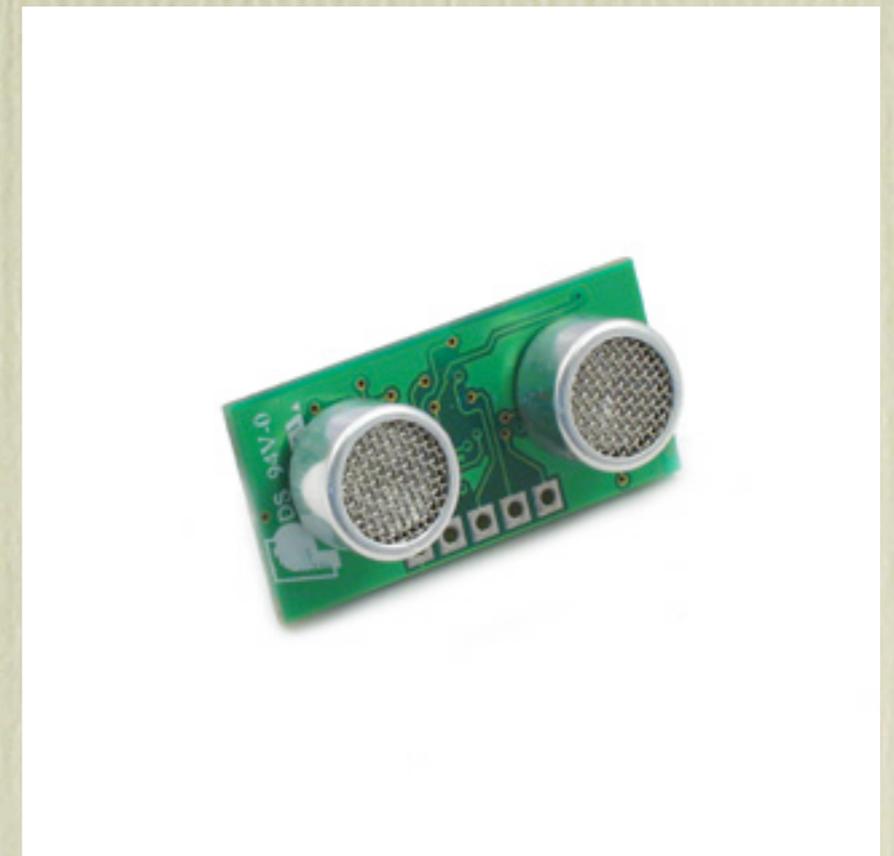
Devantech SRF10 Sonar Ranger

Frequency: 40KHz

Range: 3cm-6m

I/O: I₂C Bus

...but what about lasers?





Processor Board

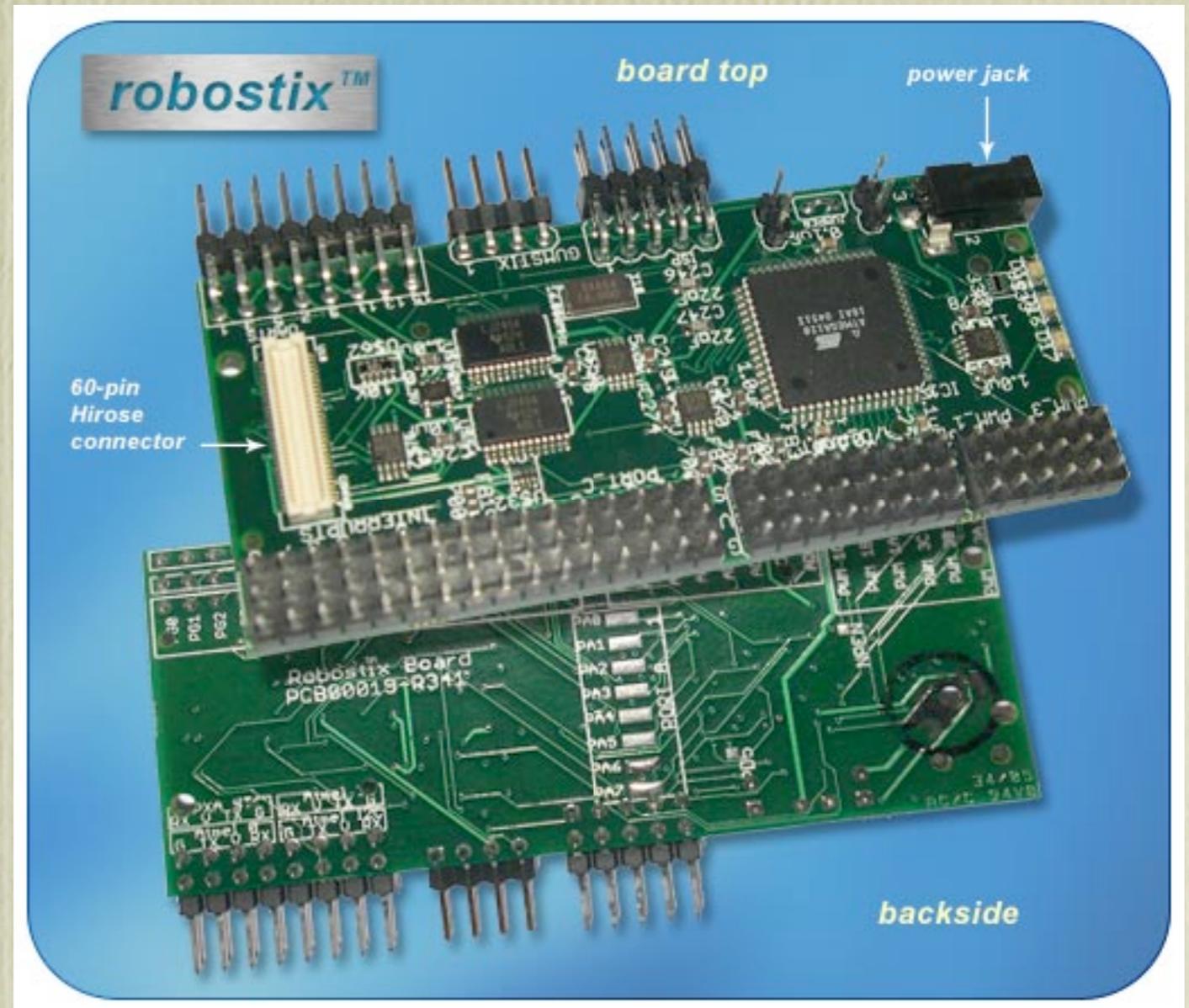
Board: Gumstix
CPU: XScale 400MHz
RAM: 64MB
Flash: 16MB
Network: Bluetooth
OS: Linux 2.6





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