Embedded Software Engineering

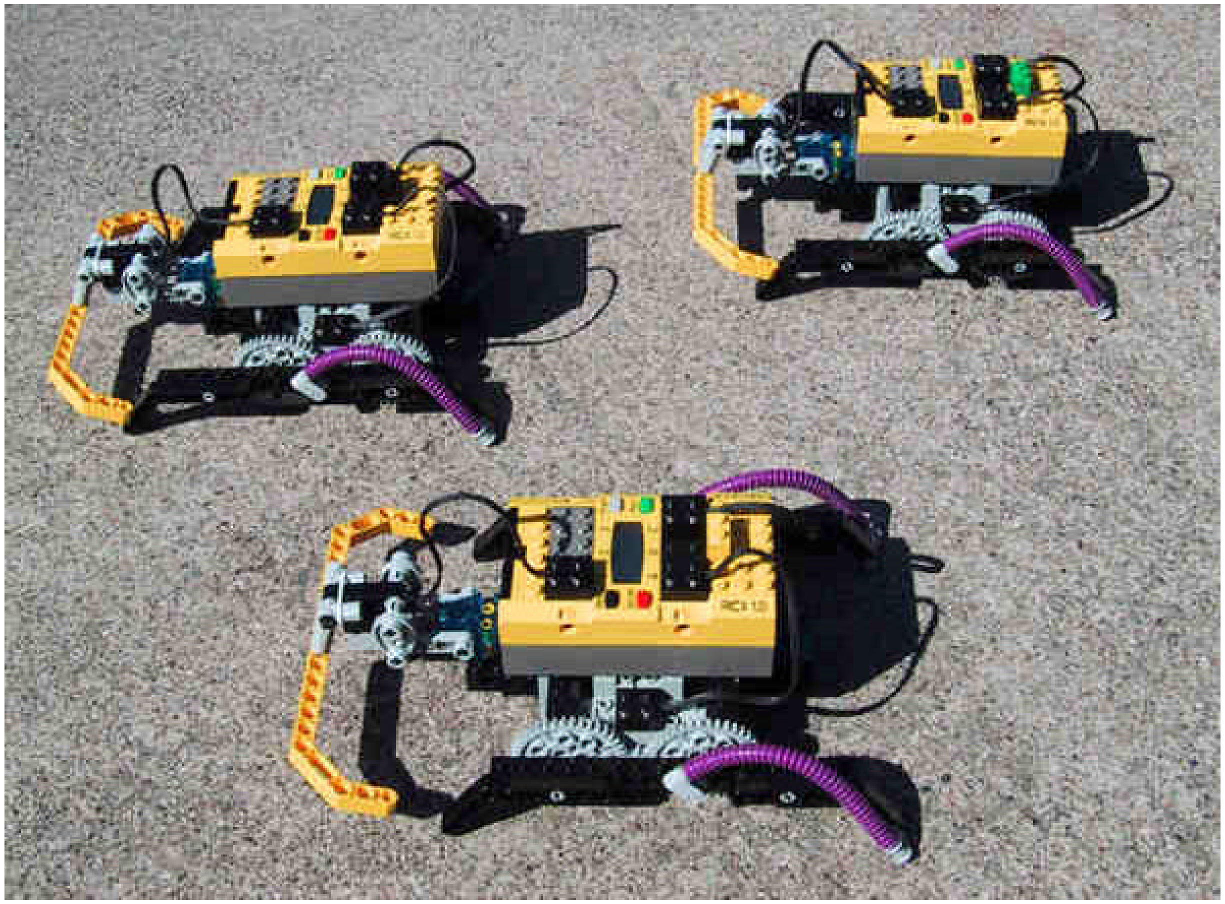
Final Class Project Presentations

EECS Department, UC Berkeley

Christoph Kirsch

www.eecs.berkeley.edu/~fresco/giotto/course

It’s fun

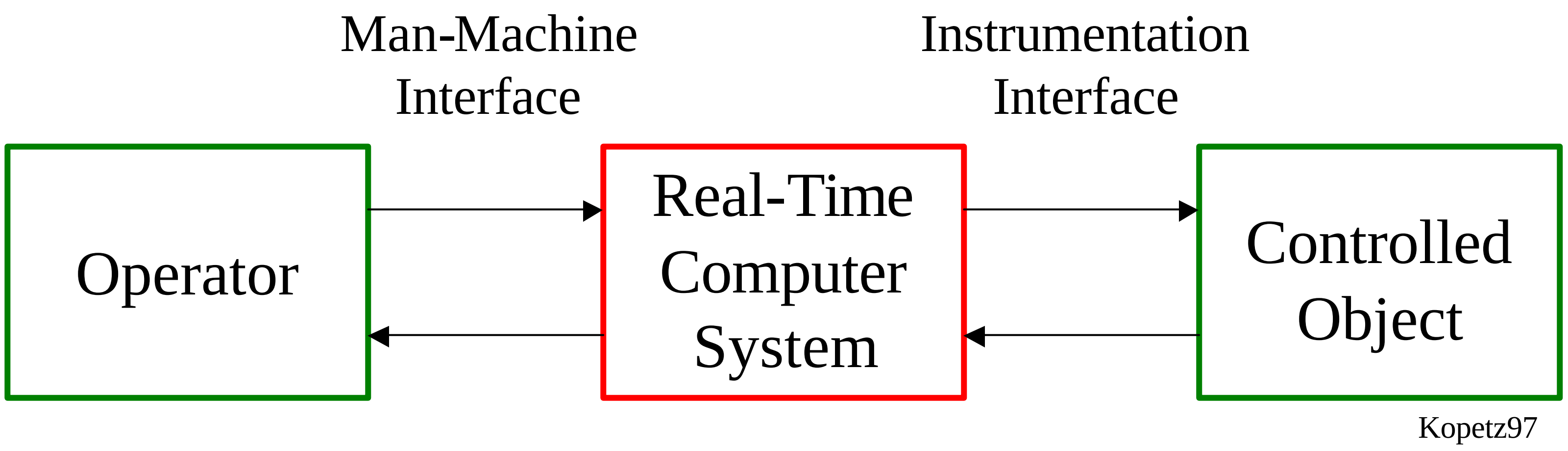


Schedule

- Christoph: Introduction
- Alvin, Daniel: Tutebot
- Carlo, Jeff: Synchronous Computation
- Paul, Jason: Scheduled Computation
- Elaine, Steve: Code Generation

- Ben, Shawn: Time-triggered Machine

Problem



Methodologies for the implementation of embedded real-time applications

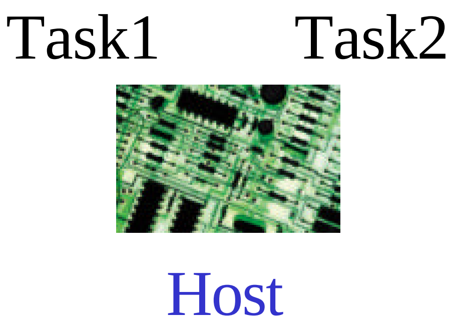
- Methodology: tool-supported, logical, compositional
- Implementation: compositional, scalable, dependable

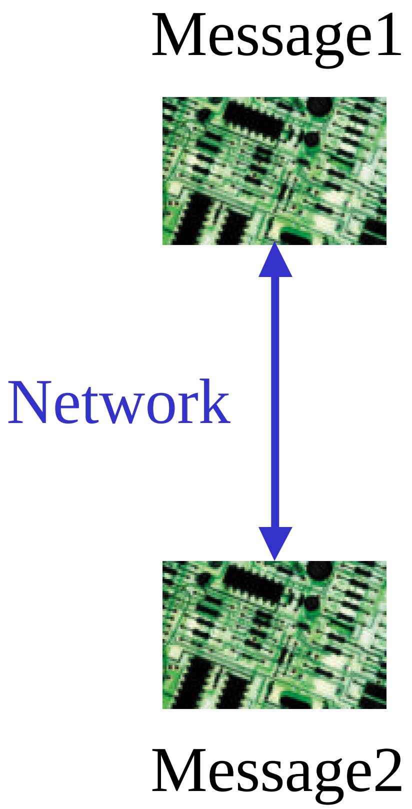
Embedded Programming

…requires the integration of:

1. Real-time scheduling/communication concepts
2. Logical RTOS: The embedded machine
3. Programming language design
4. Compiler design
5. Classical software engineering techniques
6. Formal methods

Concurrency

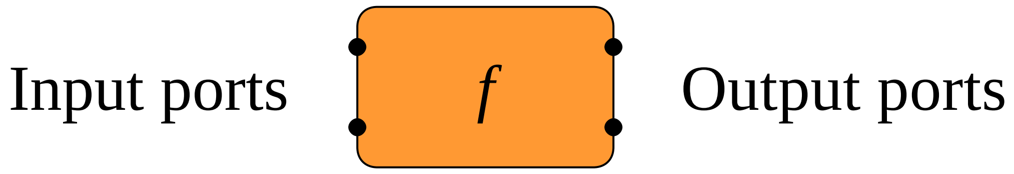




In addition:

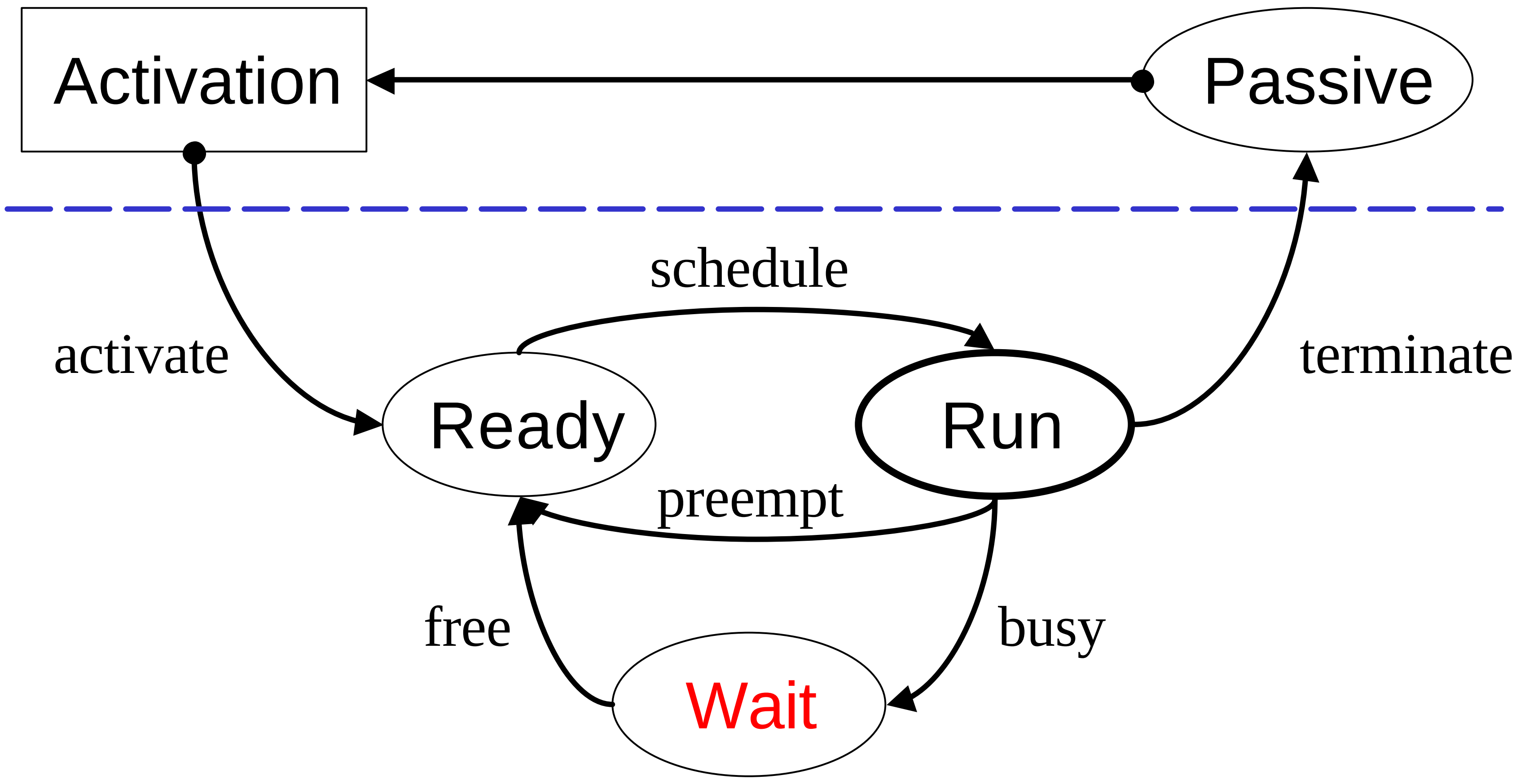
- Other resource constraints
- Time constraints

The Task Model

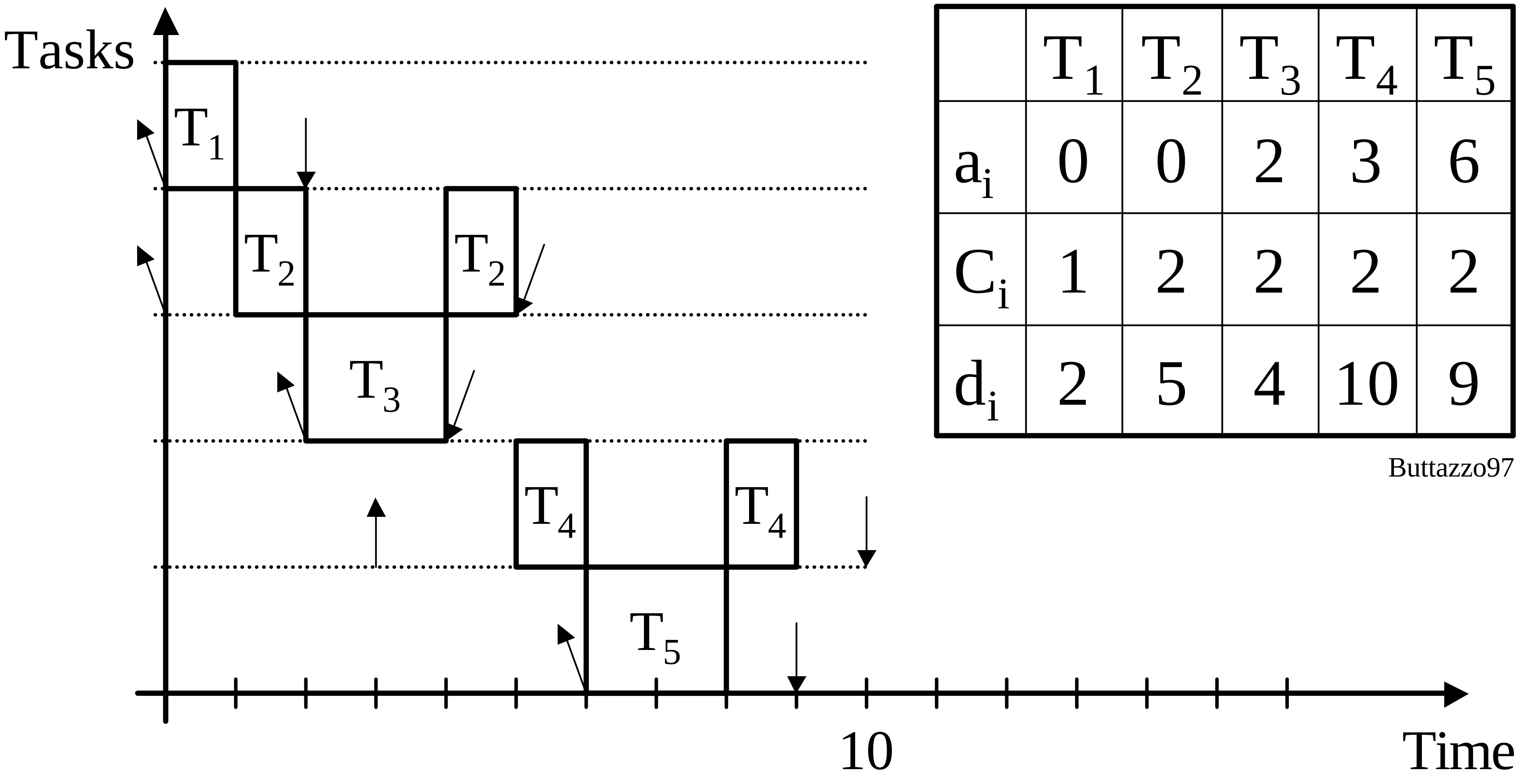


- a task is a subroutine *not* a coroutine [Wirth96]
- runs to completion, possibly preempted
- no synchronization points
- known worst case execution time

RTOS Model


Activation
Passive
schedule
activate
terminate
Ready
Run
preempt
free
busy
Wait

Earliest Deadline First

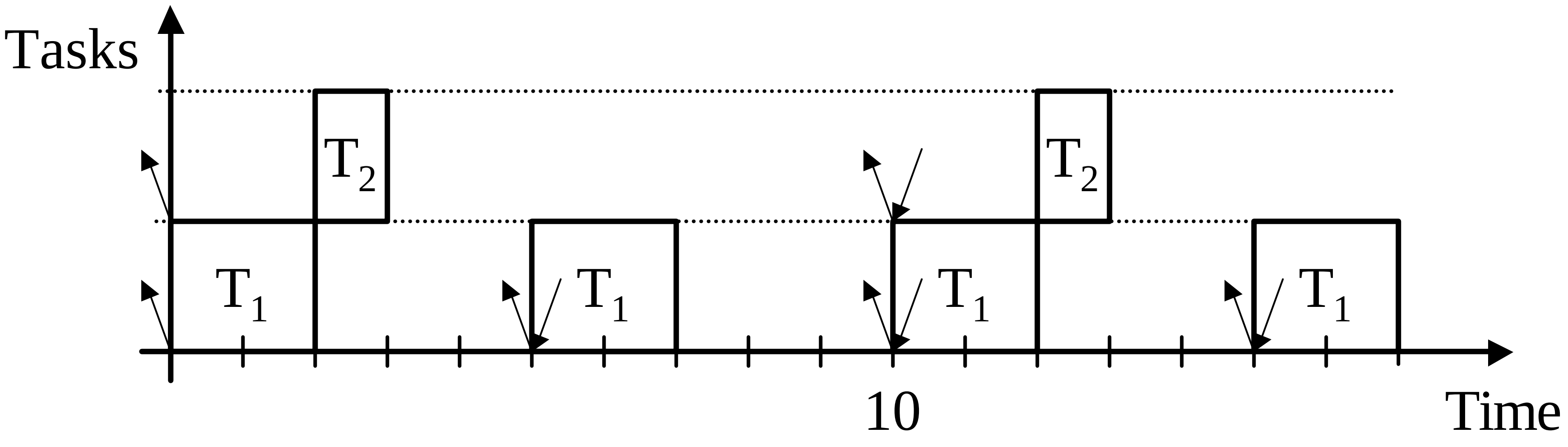


	T_1	T_2	T_3	T_4	T_5
a_i	0	0	2	3	6
C_i	1	2	2	2	2
d_i	2	5	4	10	9

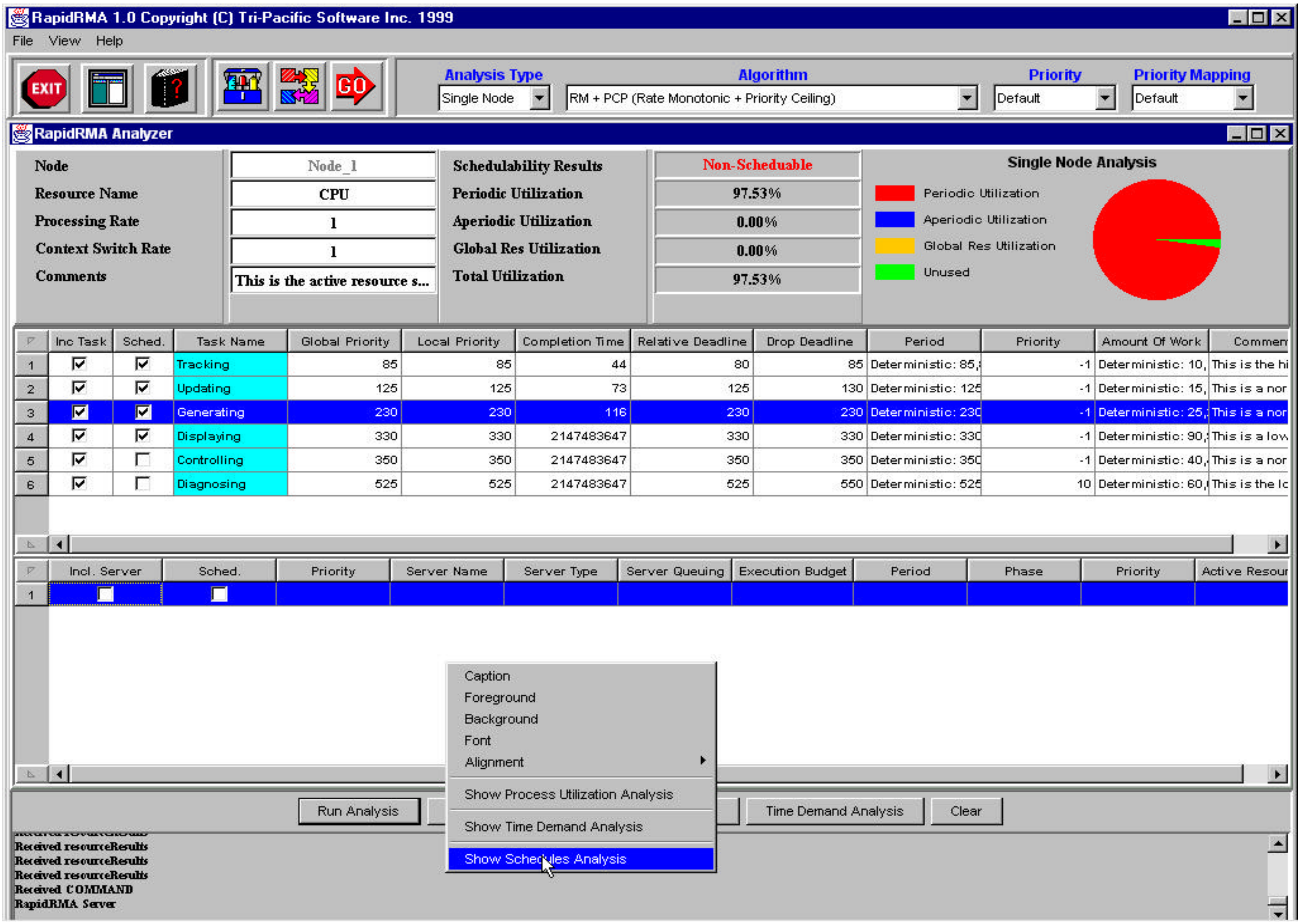
Buttazzo97

Rate Monotonic Analysis

	T_1	T_2
C_i	2	1
p_i	5	10



Elaine: RapidRMA



Analysis Type	Algorithm	Priority	Priority Mapping
Single Node	RM + PCP (Rate Monotonic + Priority Ceiling)	Default	Default

Node	Node_1
Resource Name	CPU
Processing Rate	1
Context Switch Rate	1
Comments	This is the active resource s...

Schedulability Results	Non-Schedulable
Periodic Utilization	97.53%
Aperiodic Utilization	0.00%
Global Res Utilization	0.00%
Total Utilization	97.53%

🟥 Periodic Utilization
🟦 Aperiodic Utilization
🟨 Global Res Utilization
🟩 Unused

	Inc Task	Sched.	Task Name	Global Priority	Local Priority	Completion Time	Relative Deadline	Drop Deadline	Period	Priority	Amount Of Work	Commen
1	✓	✓	Tracking	85	85	44	80	85	Deterministic: 85,	-1	Deterministic: 10,	This is the hi
2	✓	✓	Updating	125	125	73	125	130	Deterministic: 125	-1	Deterministic: 15,	This is a nor
3	✓	✓	Generating	230	230	116	230	230	Deterministic: 230	-1	Deterministic: 25,	This is a nor
4	✓	✓	Displaying	330	330	2147483647	330	330	Deterministic: 330	-1	Deterministic: 90,	This is a low
5	✓	☐	Controlling	350	350	2147483647	350	350	Deterministic: 350	-1	Deterministic: 40,	This is a nor
6	✓	☐	Diagnosing	525	525	2147483647	525	550	Deterministic: 525	10	Deterministic: 60,	This is the lc

	Incl. Server	Sched.	Priority	Server Name	Server Type	Server Queuing	Execution Budget	Period	Phase	Priority	Active Resour
1	☐	☐									

```
Received resourceResults
Received resourceResults
Received resourceResults
Received COMMAND
RapidRMA Server
```

Ben: Deferrable,Sporadic Servers

No slides available

Jeff: Wet Scheduling

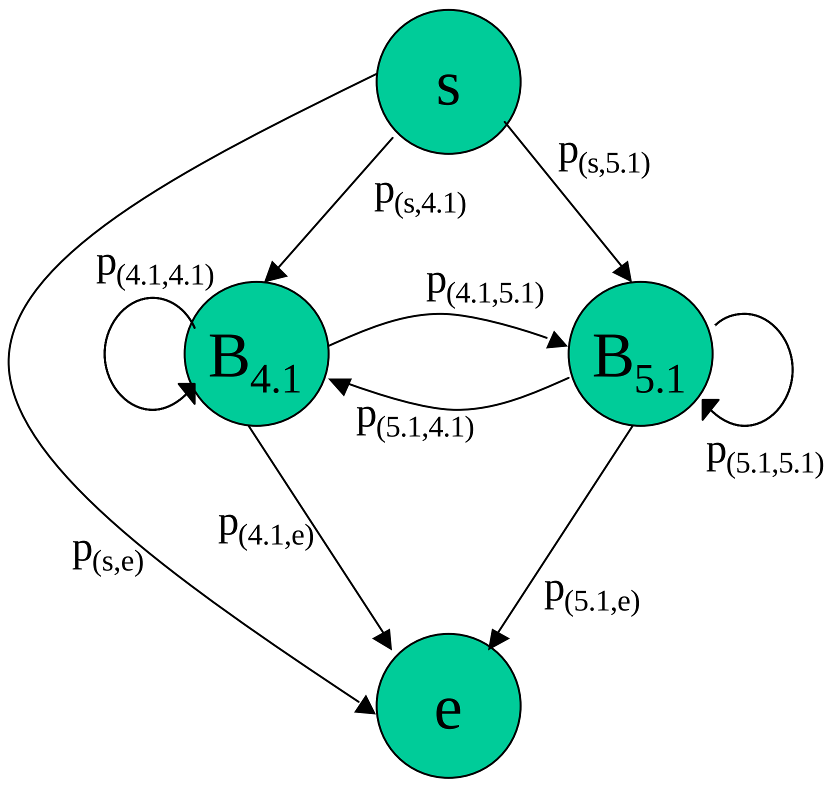
No slides available

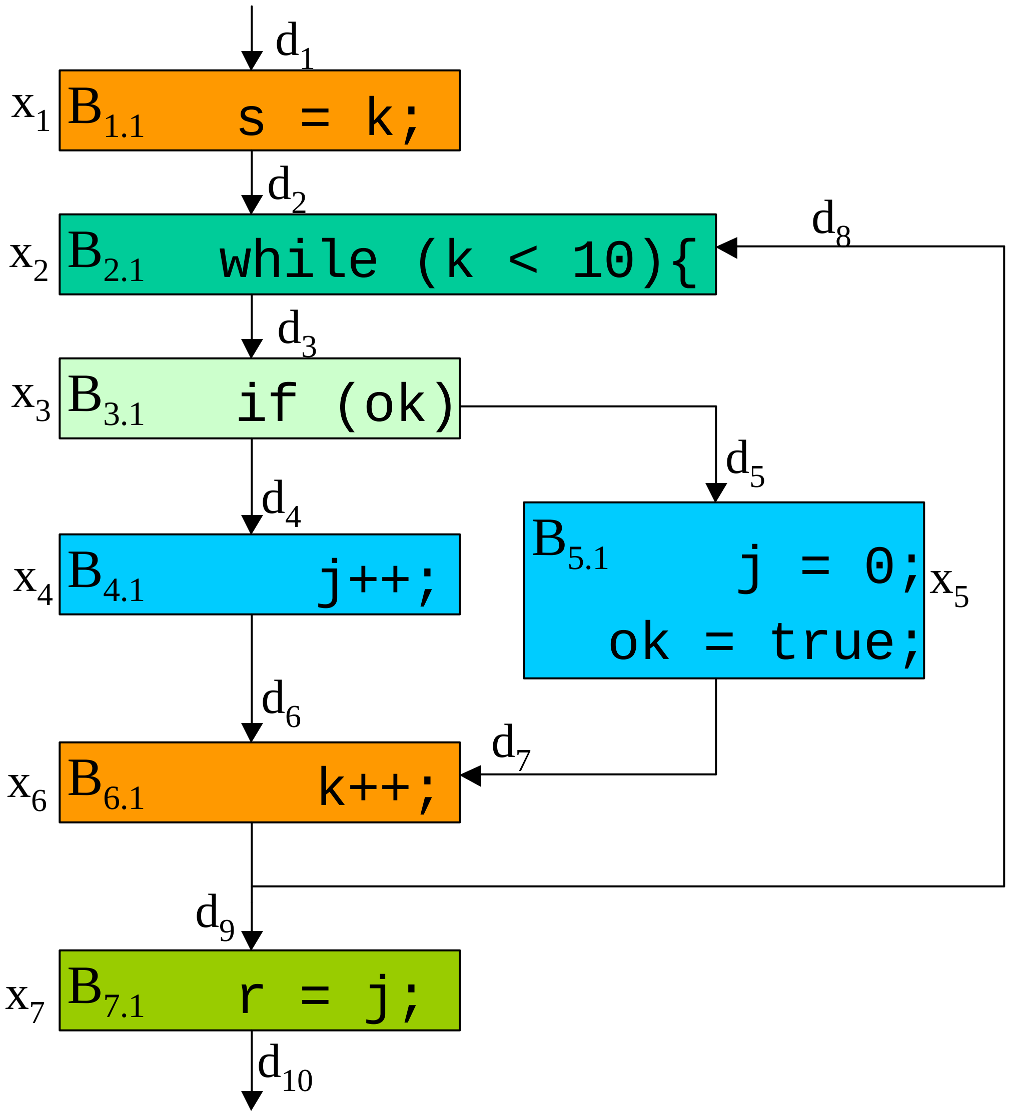
Alvin: Research Operating Systems

Objective: The ability to treat tasks with explicit timing constraints, such as periods and deadlines

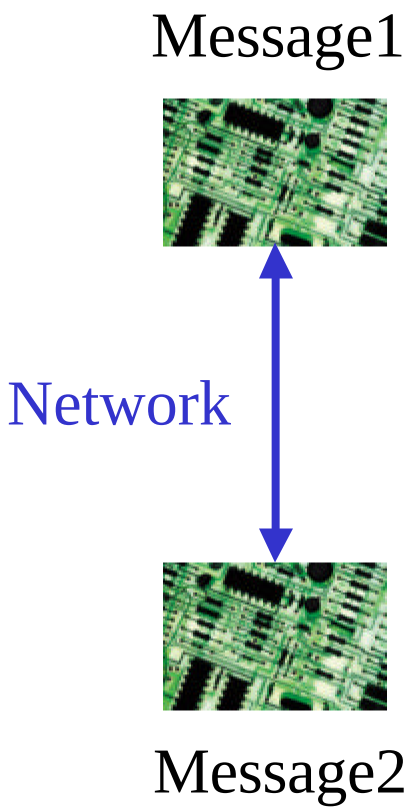
Characteristics:
- Scheduling guarantee mechanisms
- Characterize tasks with additional parameters
- Avoidance of nondeterministic blocking time

Shawn: WCET Analysis

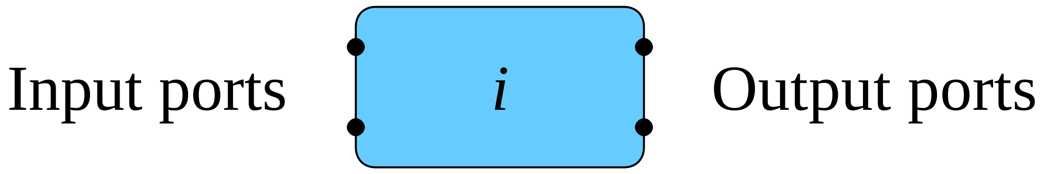




Real-Time Communication

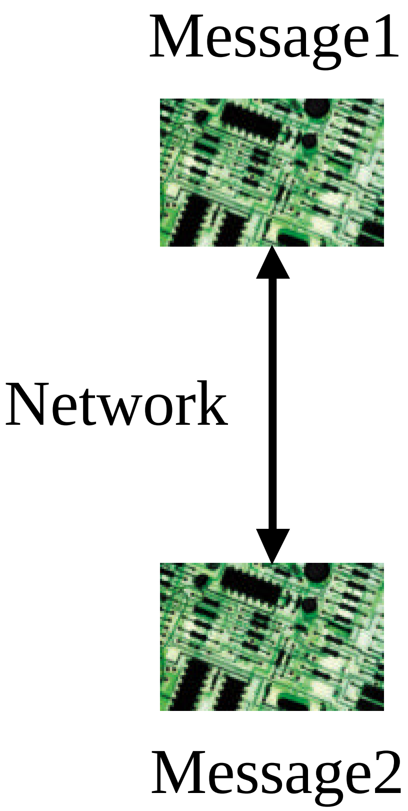


The Communication Model



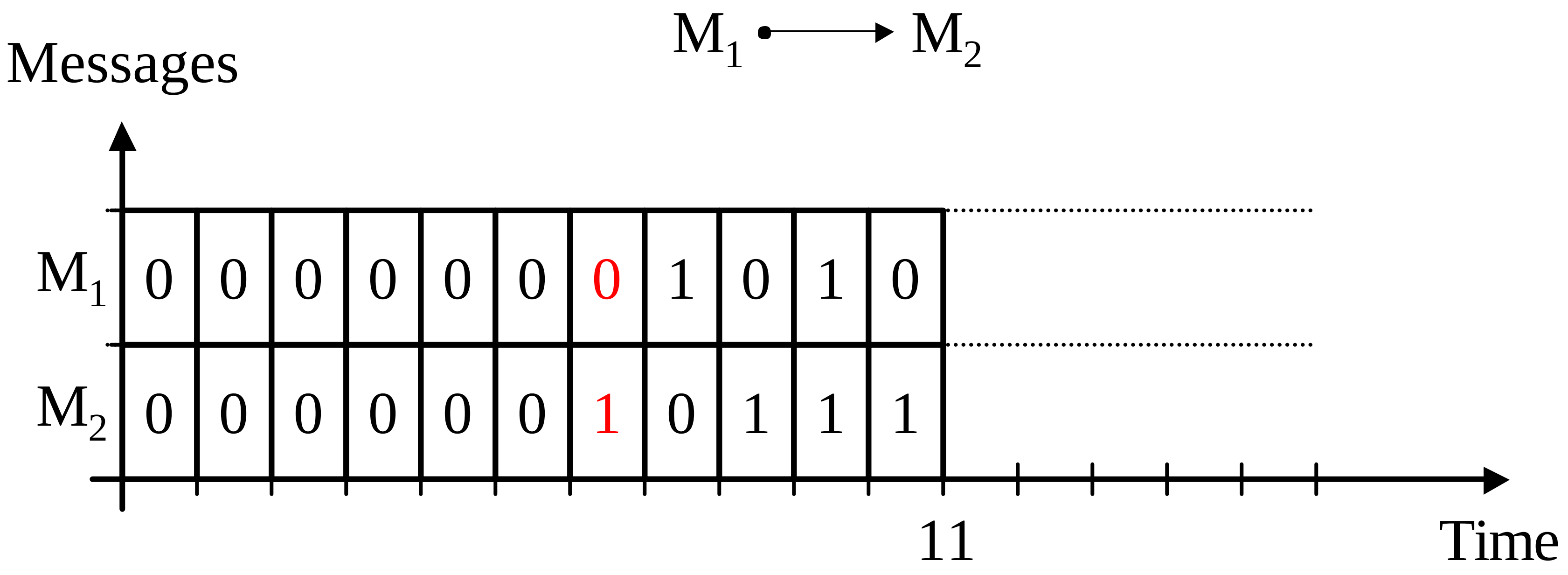
- a connection is a function from input to output ports
- a message is a valuation of the input ports
- no predefined protocol, preemption possible
- known worst case latency

Explicit Flow Control



- Send time not known a priori
- Sender can detect errors

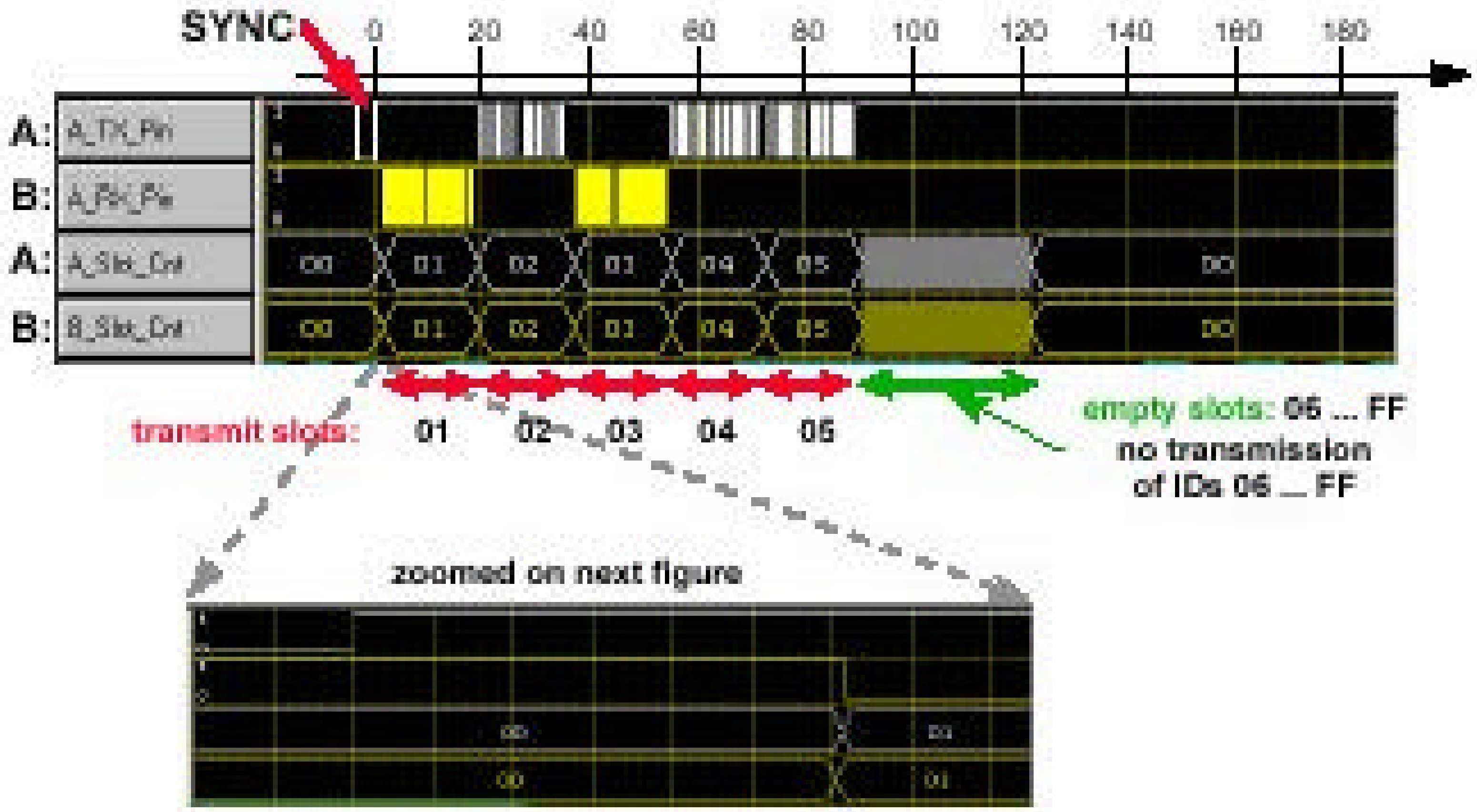
Control Area Network



Paul: Strengths and Weaknesses of CAN

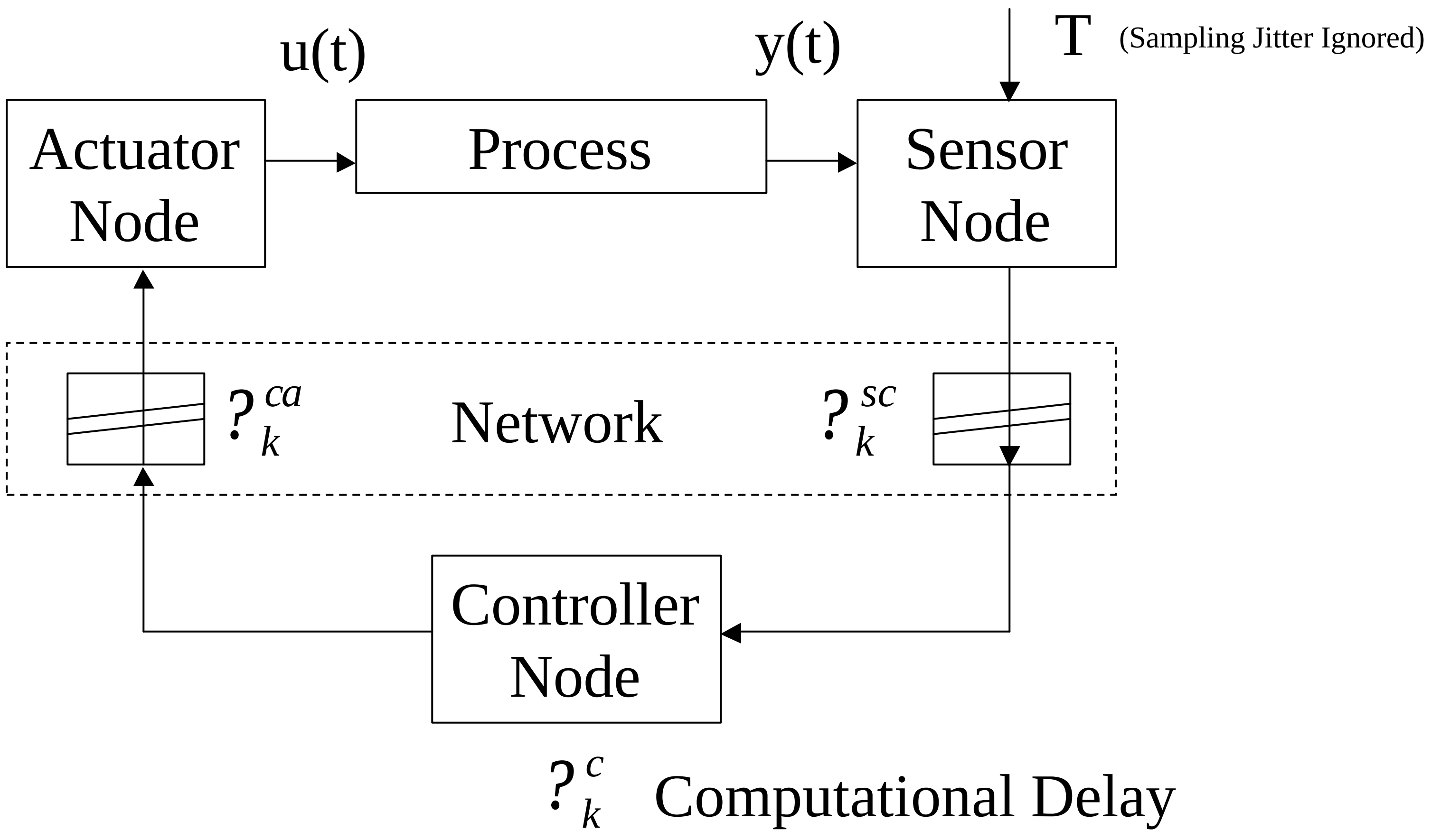
- Widely accepted standard
- Robust
  - Handles extreme conditions (does not exhibit thrashing)
  - Simple to configure
  - Good error detection
  - Two wire fault detection
- Lots of hardware and software that support CAN

Jason: Byteflight

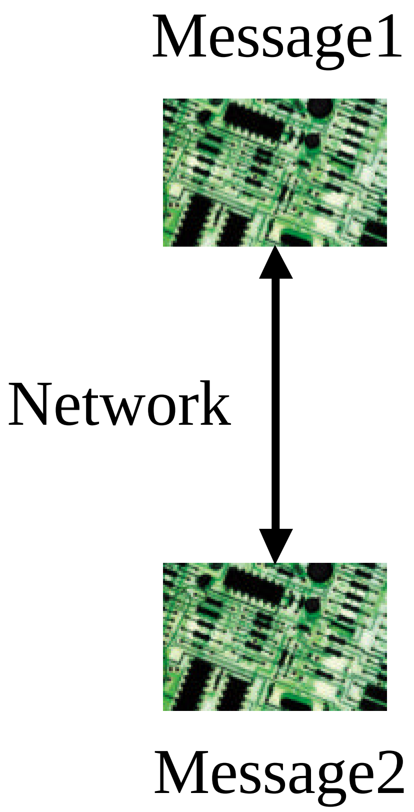


Carlo: Ethernet & Fieldbus War

- What causes the variable latency?



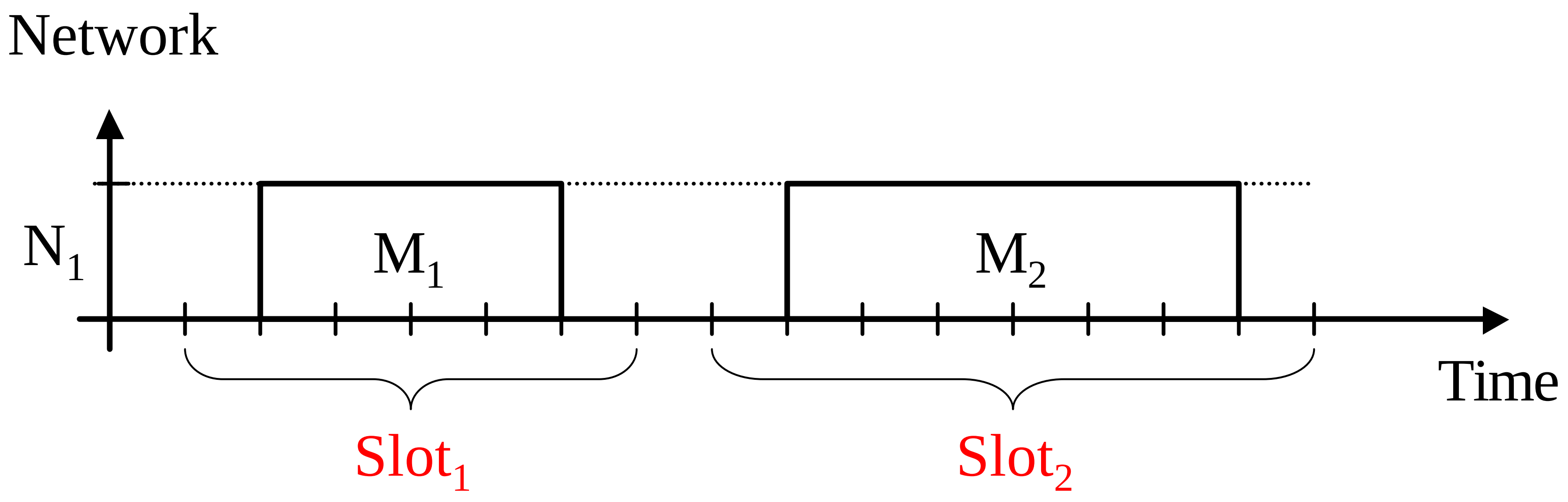
Implicit Flow Control



- Send time is known a priori
- Receiver can detect errors

Time-Triggered Protocol

$$\begin{tikzcd} M_1 \arrow[r, "\bullet" description, start anchor=west] & M_2 \end{tikzcd}$$

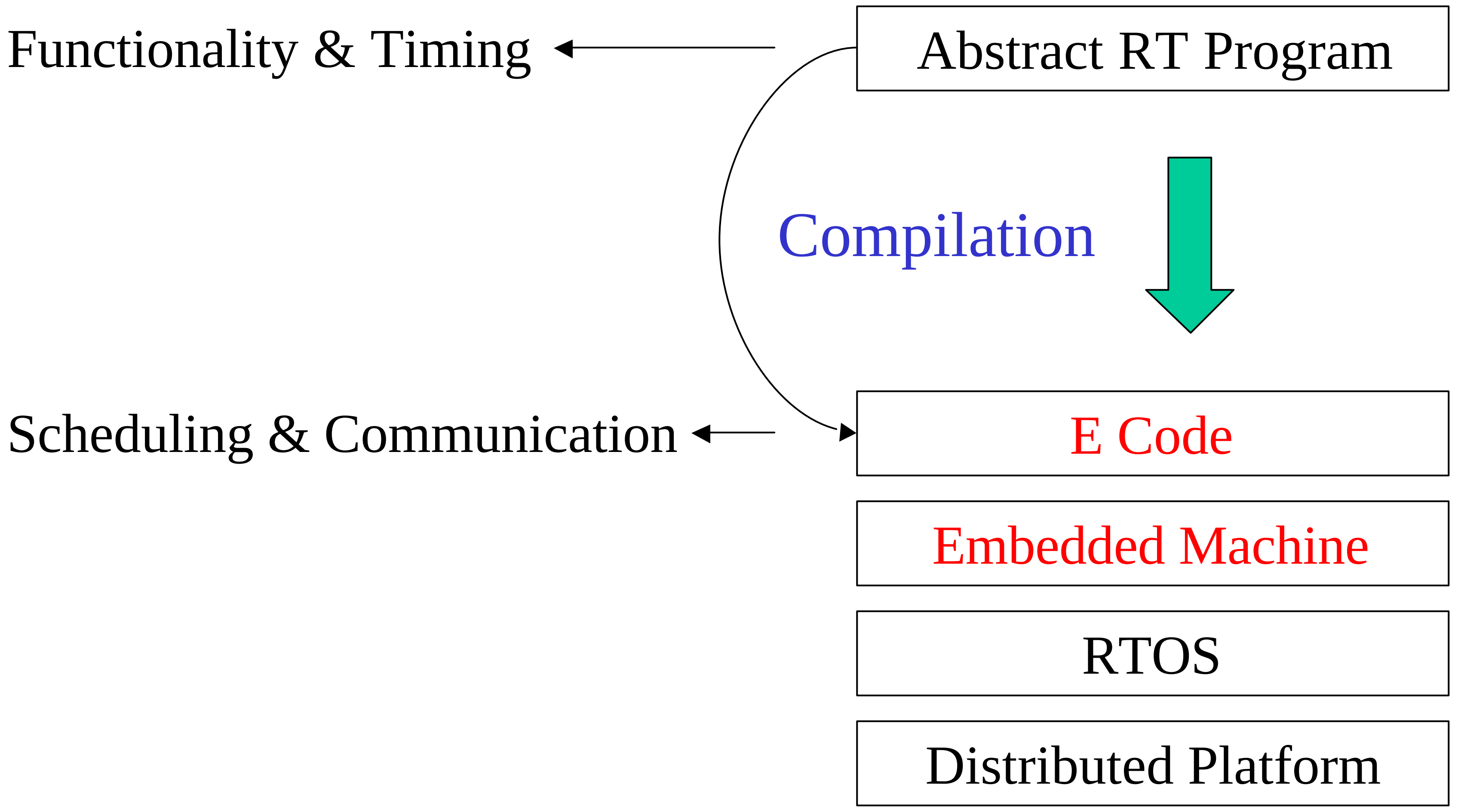


Embedded Programming

…requires the integration of:

1. Real-time scheduling/communication concepts
2. Logical RTOS: The embedded machine
3. Programming language design
4. Compiler design
5. Classical software engineering techniques
6. Formal methods

The Embedded Machine



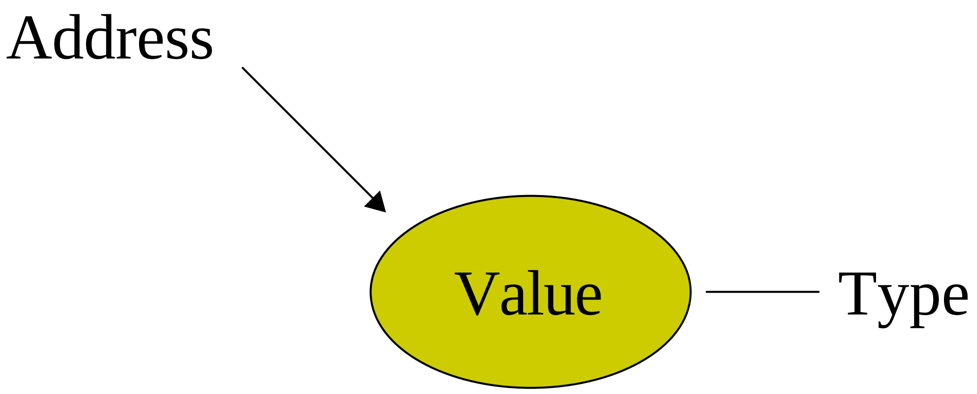
The E Machine

- The embedded machine or E machine is a virtual scheduling machine

- The E machine has:
  - internal memory, external interface
  - an instruction set similar to machine code
  - a stack used for arguments and return addresses

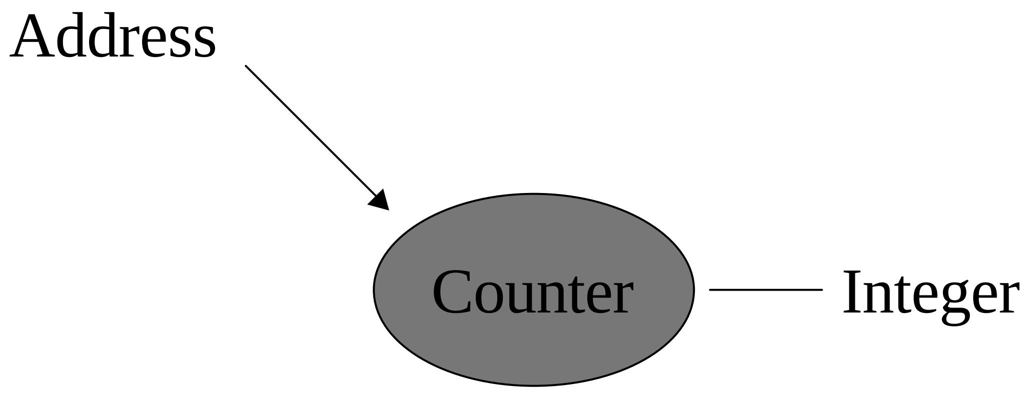
- The E machine provides a platform for generating distributed real-time schedules

Value Ports



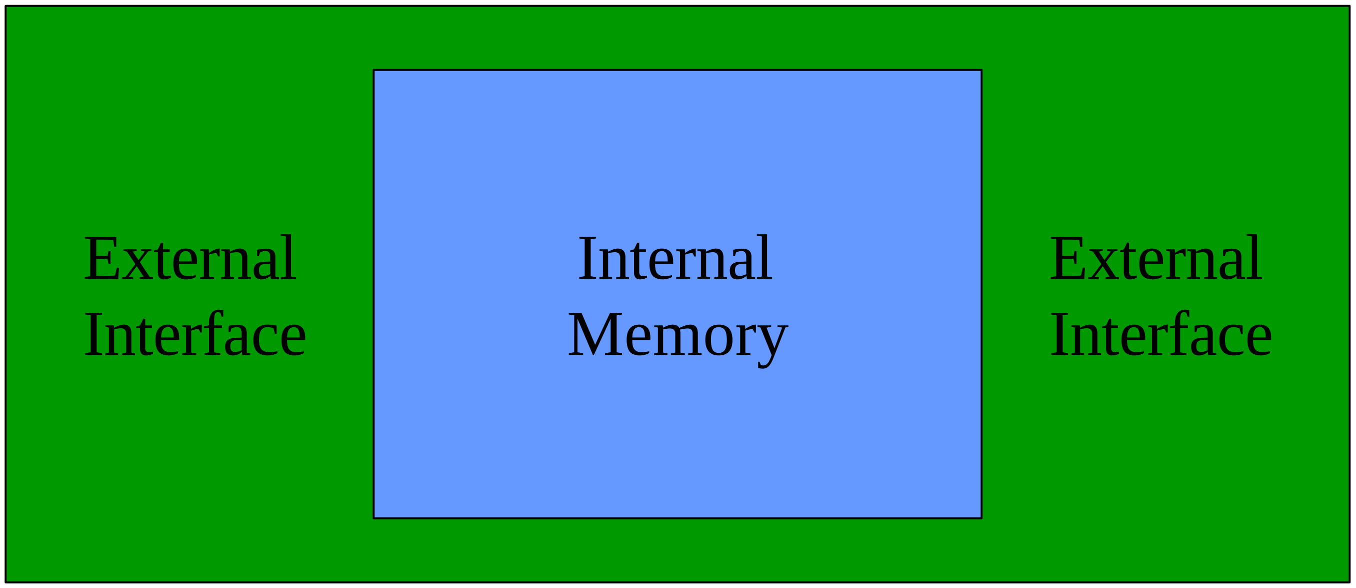
Example: Sensor Value

Signal Ports



Example: Absolute Time

Memory and Interfaces

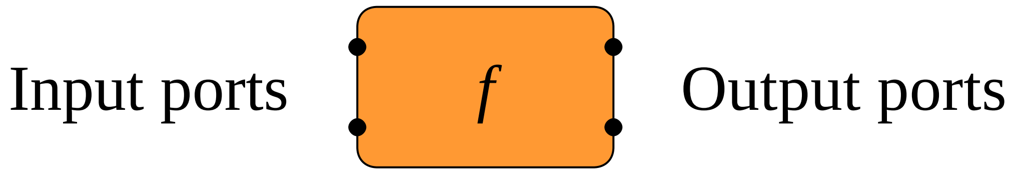


- Sensors
- Clocks

- Task Communication

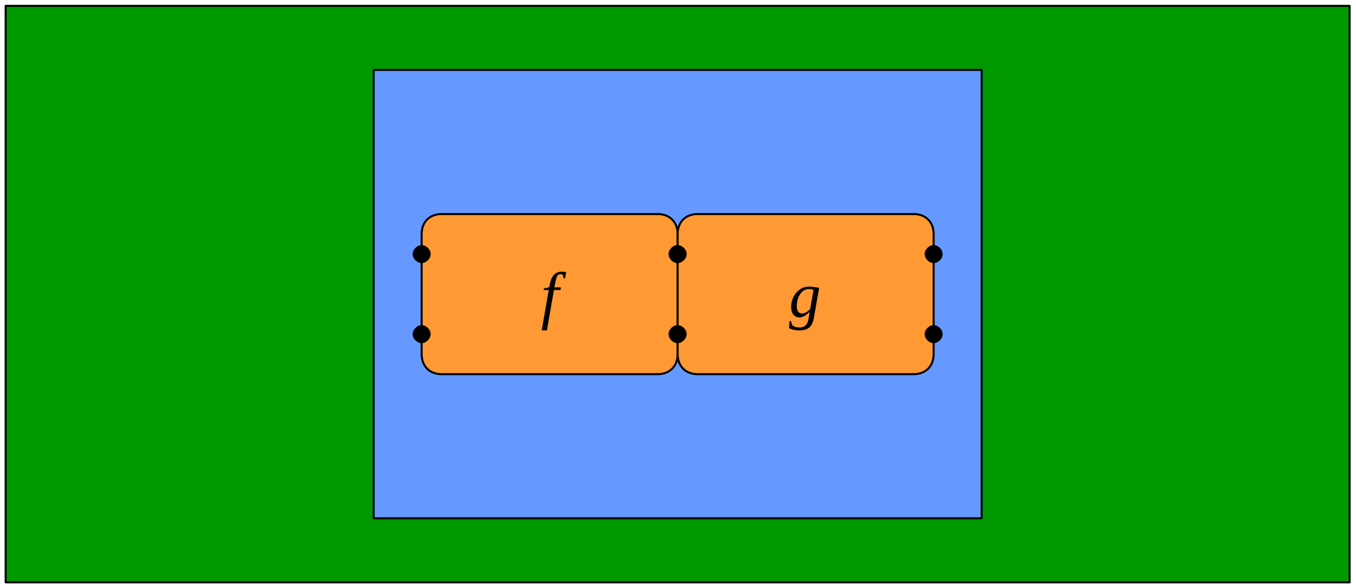
- Actuators
- Networks

The Task Model



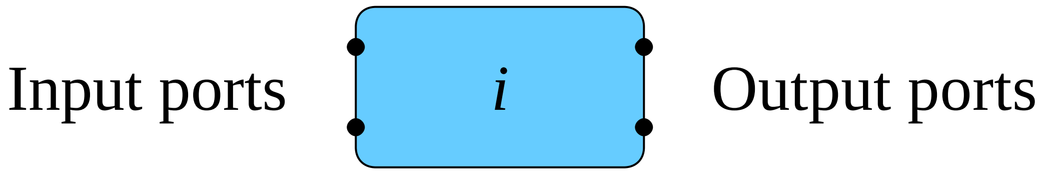
- a task is a subroutine *not* a coroutine [Wirth96]
- runs to completion, possibly preempted
- no synchronization points
- known worst case execution time

The Task Model



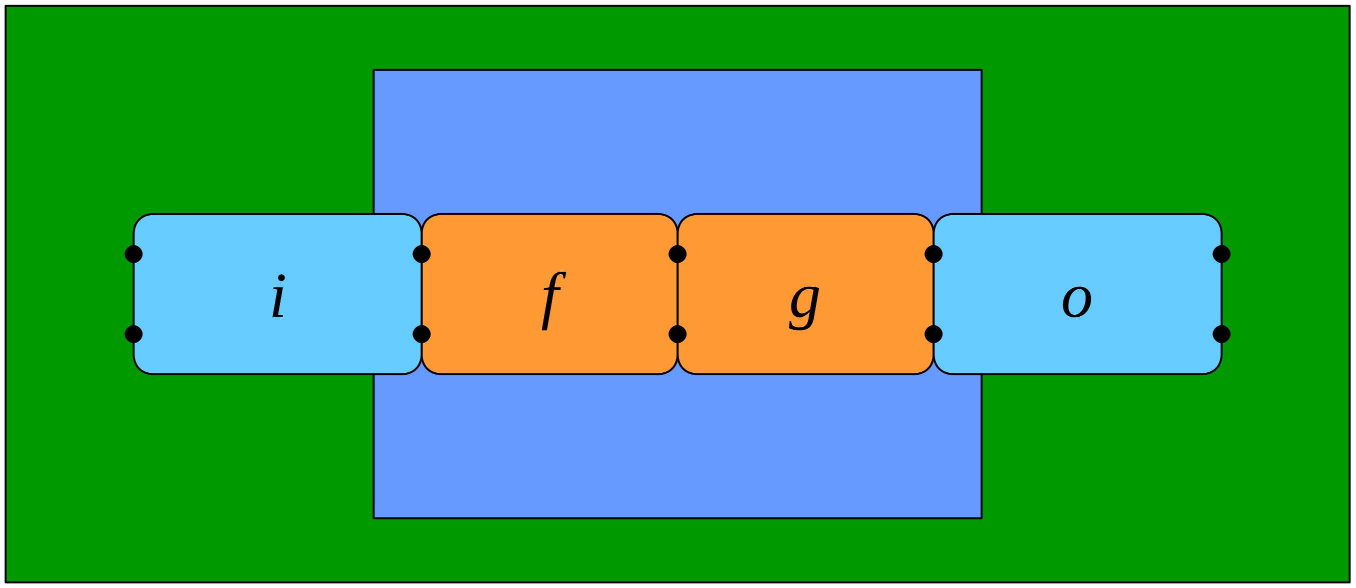
- Task Communication

The Communication Model



- a connection is a function from input to output ports
- a message is a valuation of the input ports
- no predefined protocol, preemption possible
- known worst case latency

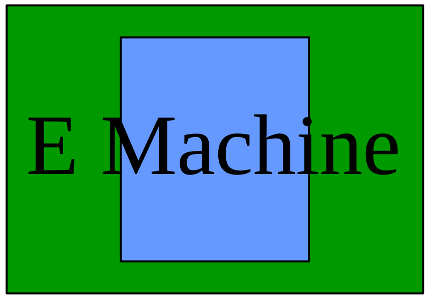
The Communication Model



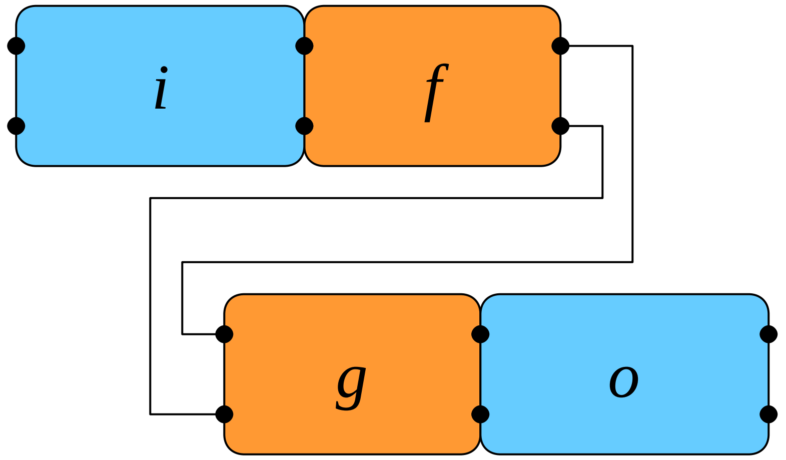
- Sensors
- Clocks

- Actuators
- Networks

E Machine Scheduling

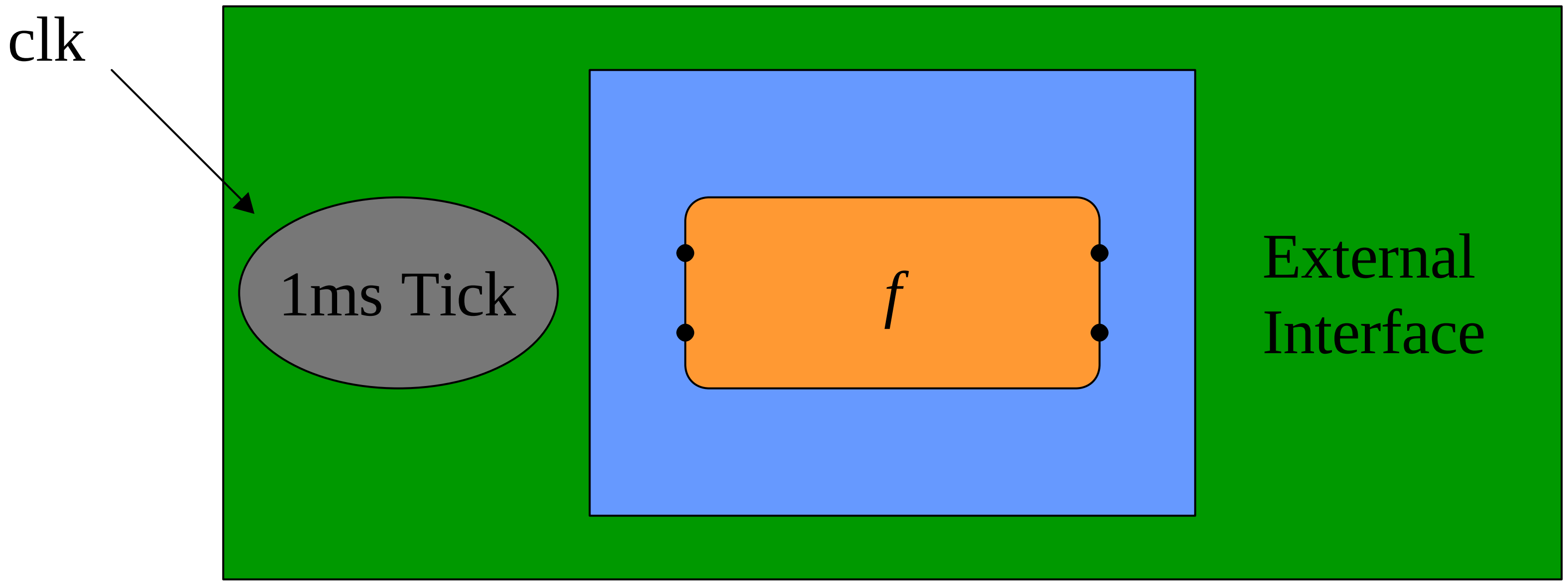


schedules



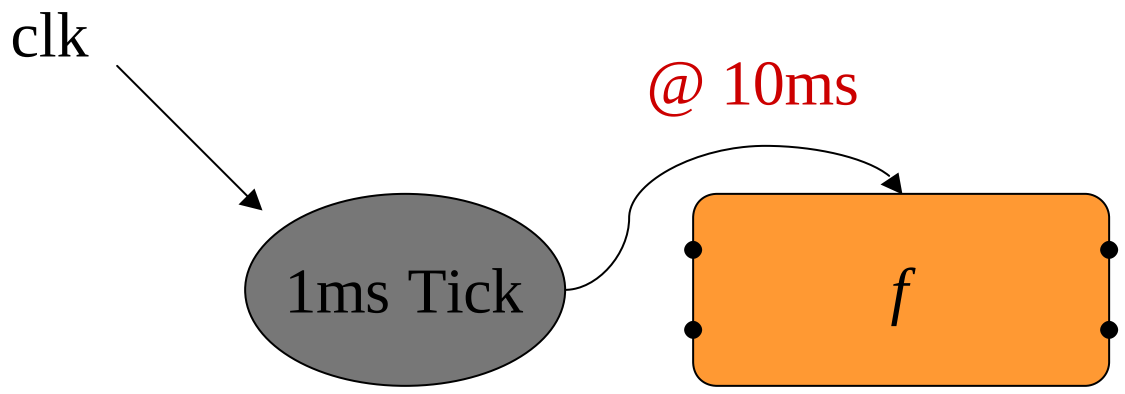
- Tasks
- Connections

A Time-Triggered Task



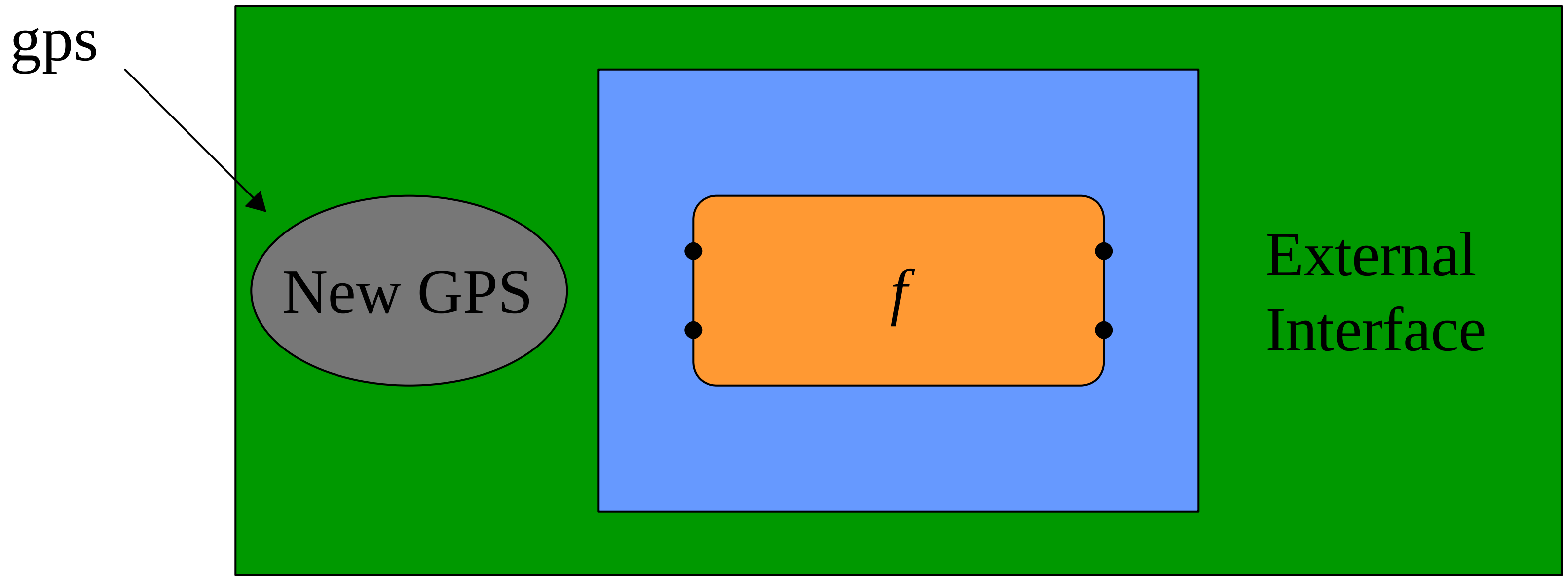
- RT Clock

A Trigger



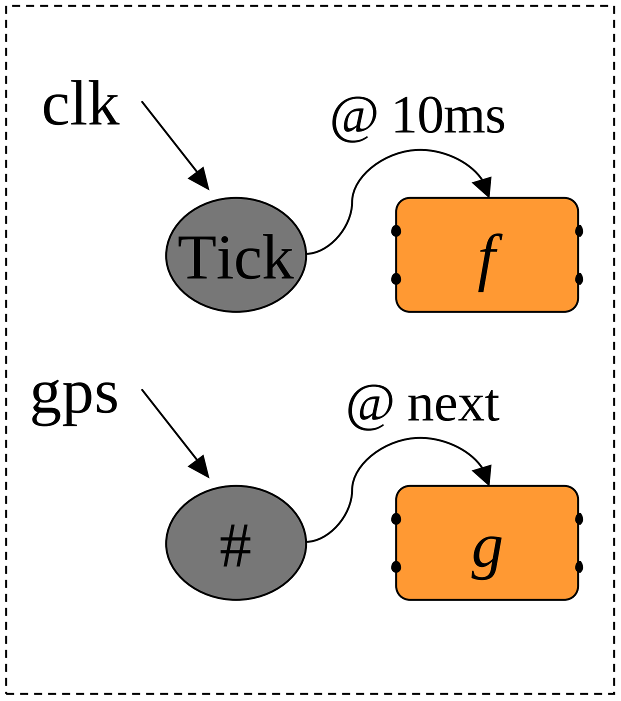
• an E machine schedule is a set of triggers

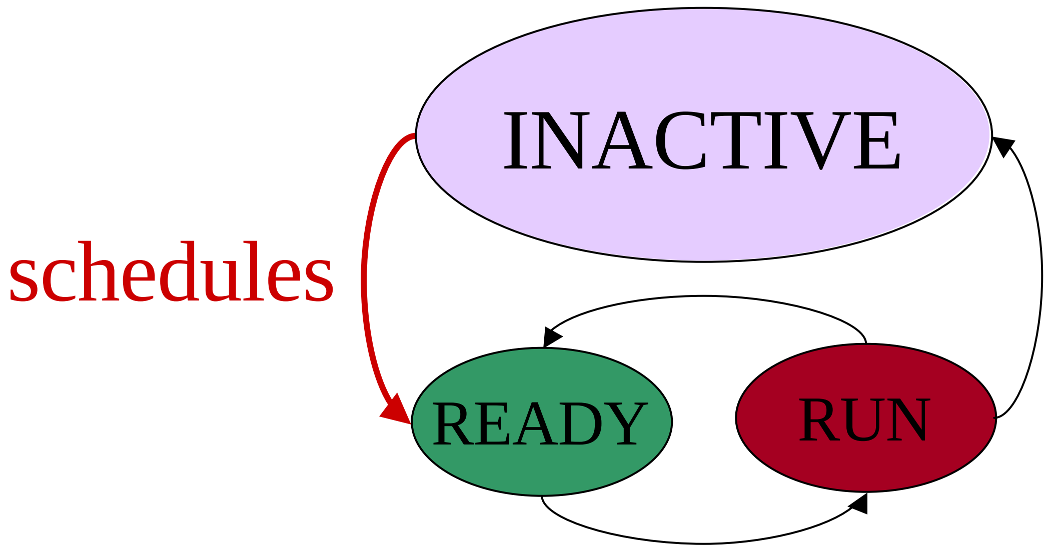
An Event-Triggered Task



- GPS Receiver

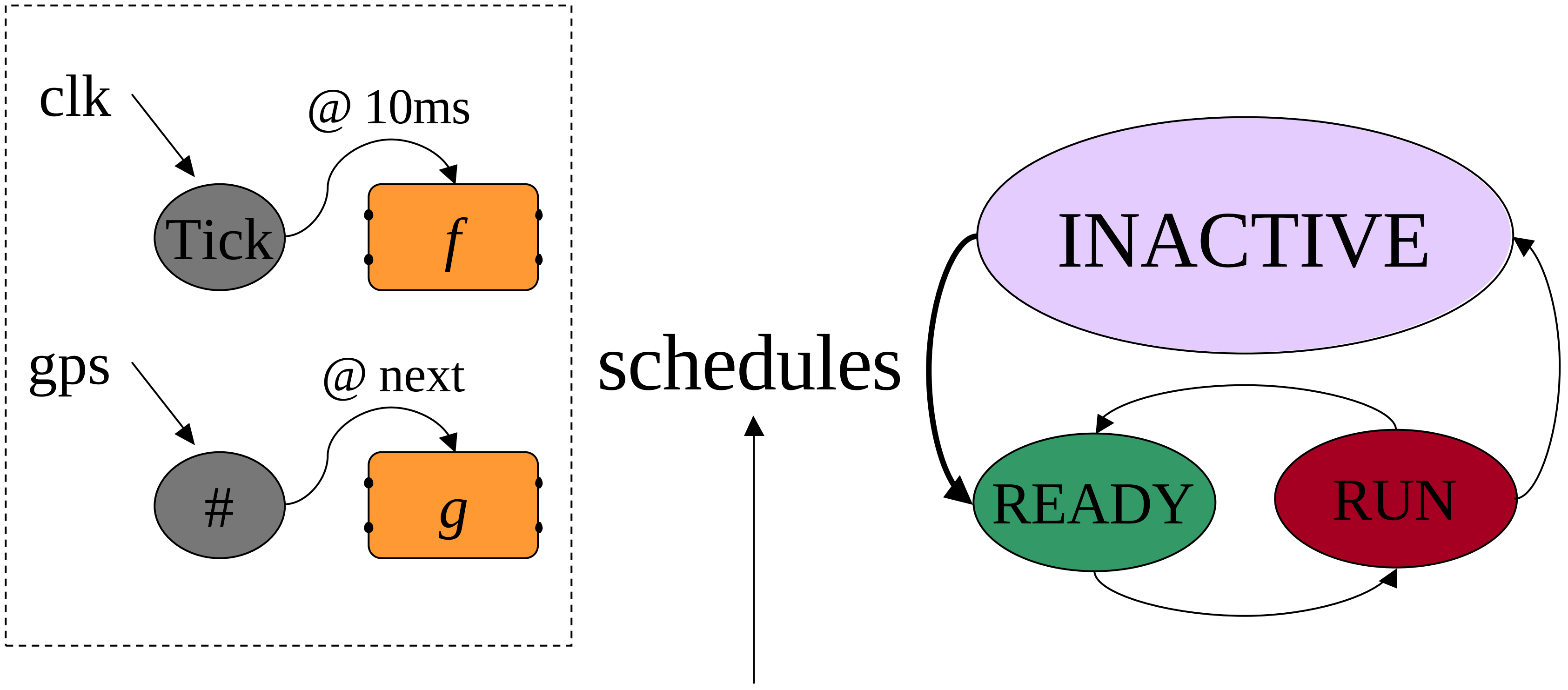
The E Machine Scheduler





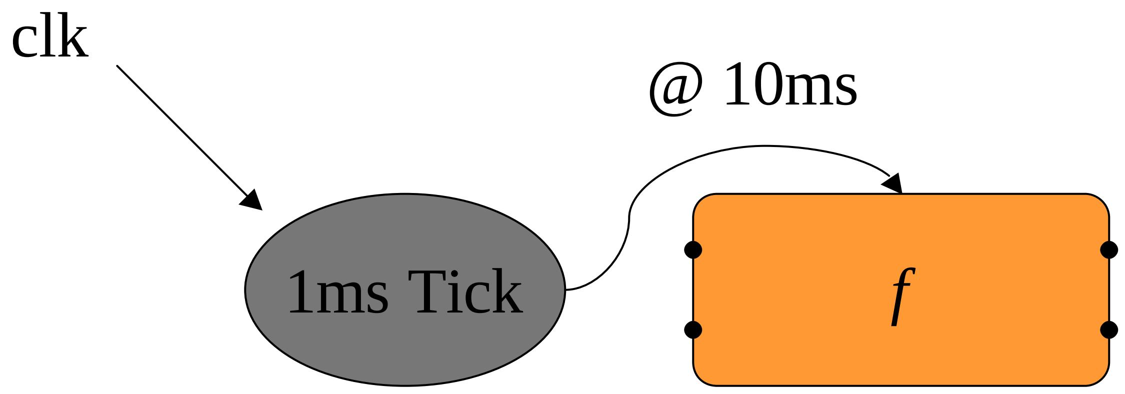
- E Machine Scheduler
- Dispatcher

Scheduling Algorithm



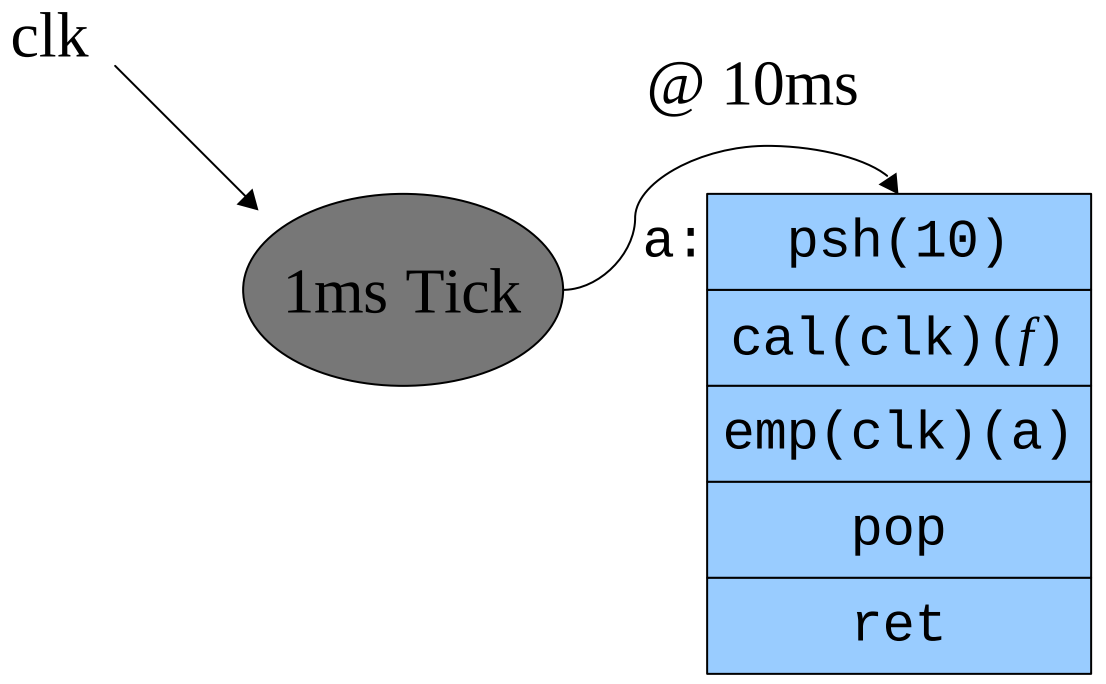
• The scheduling algorithm, e.g, EDF is a parameter of E code

A Trigger



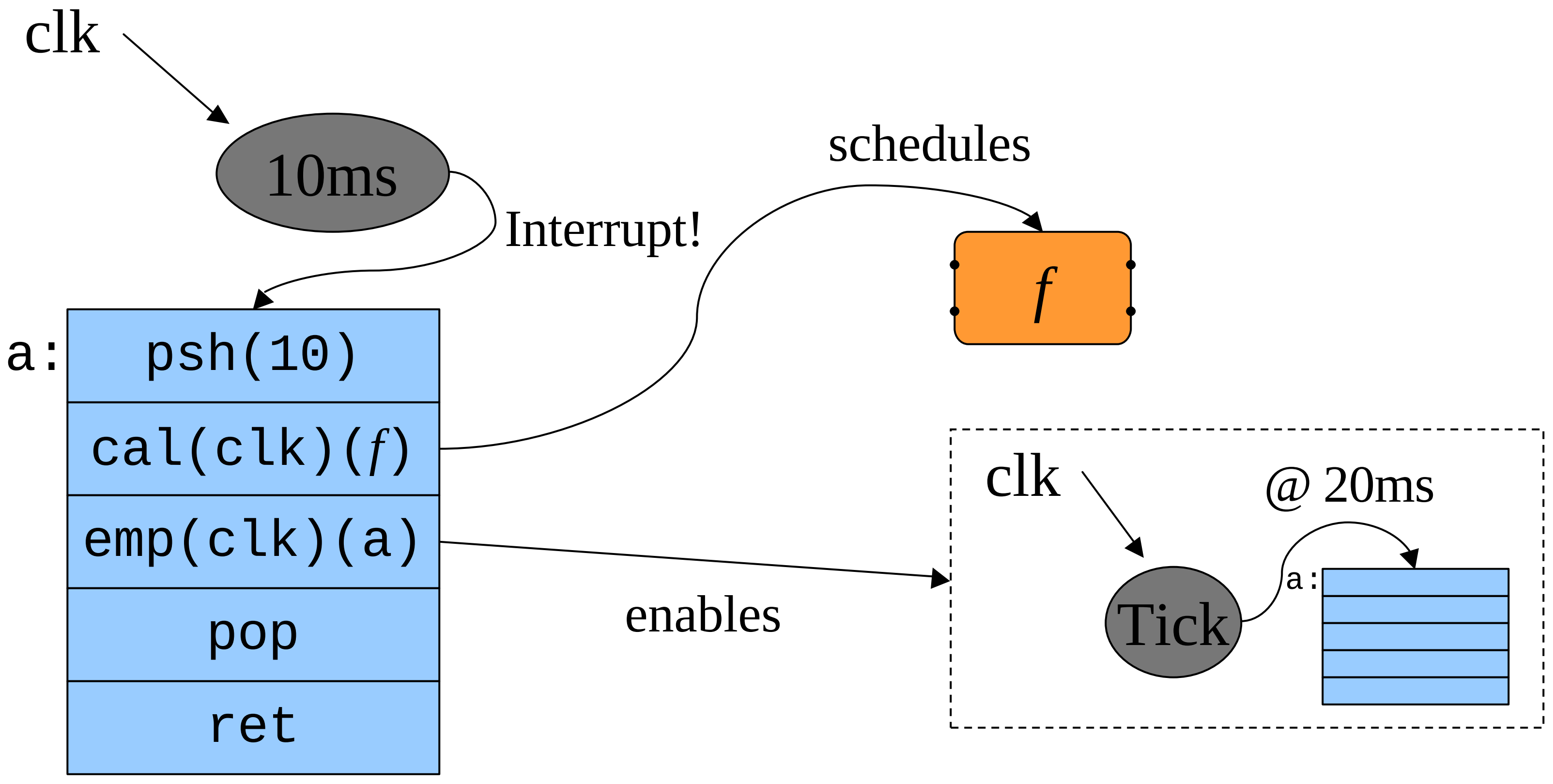
- How can we generalize triggers?

A Revised Trigger



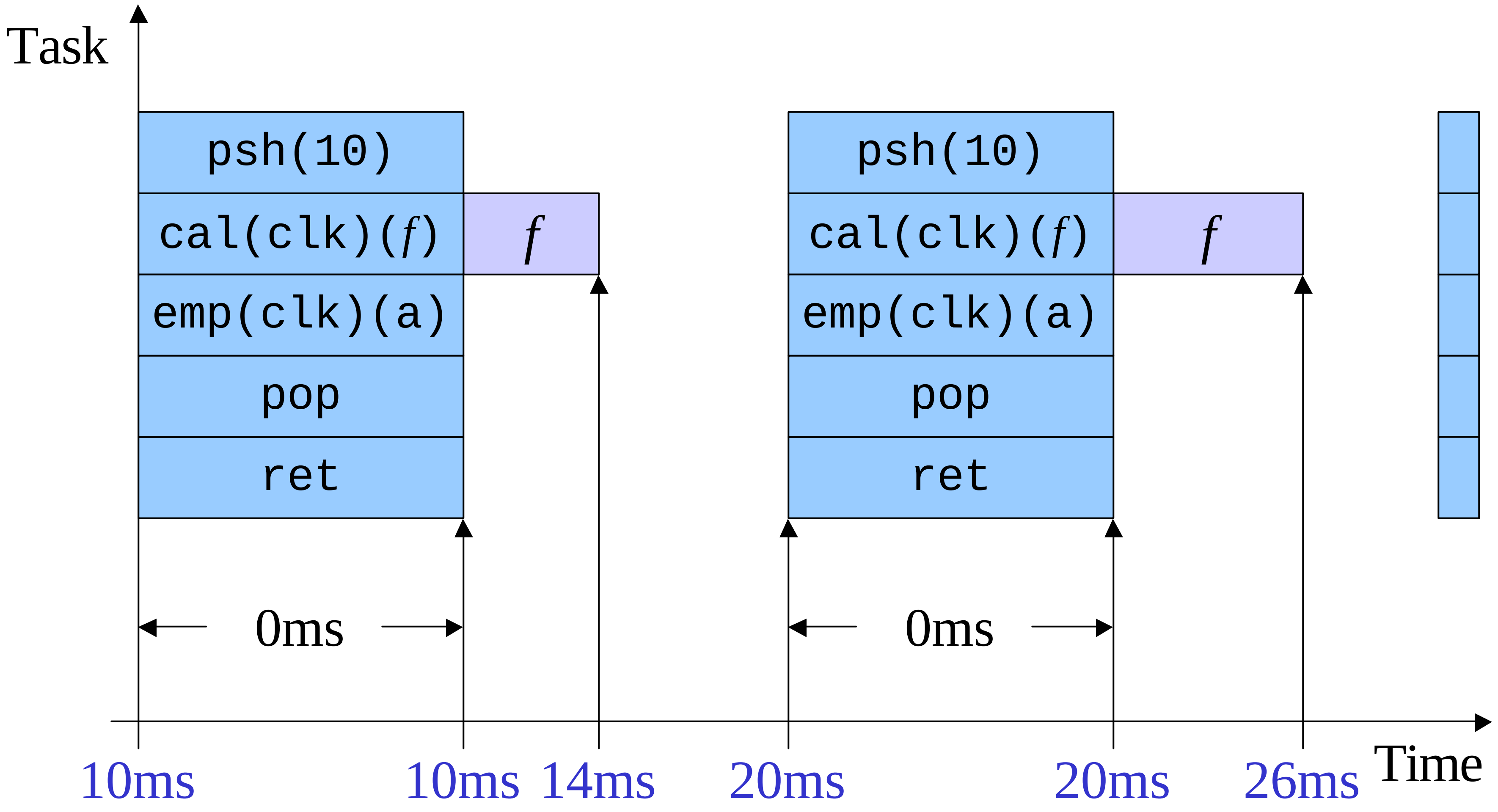
- E code for the E machine

E Code



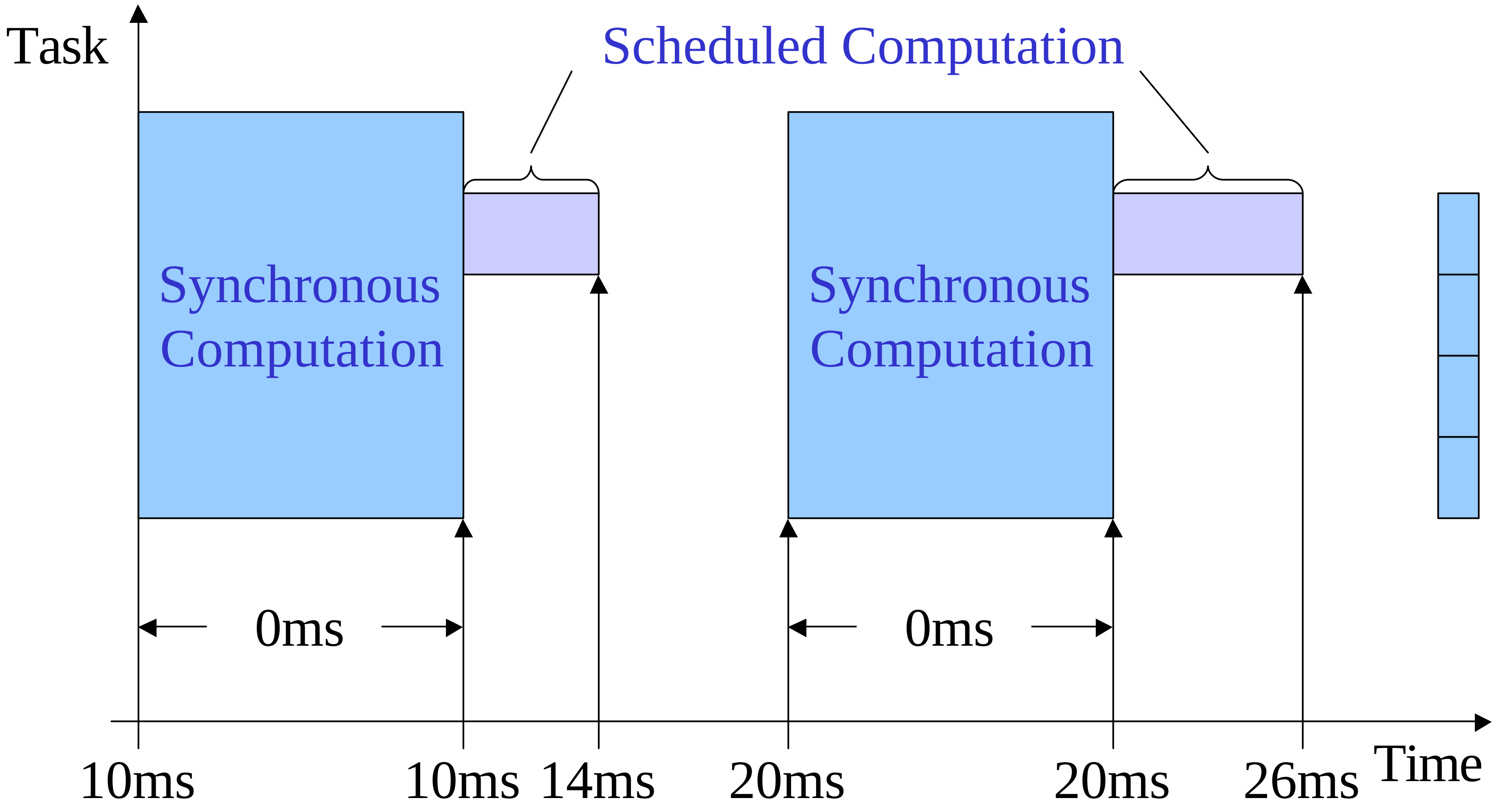
- `cal(clk)(`*f*`)` assigns priority to *f*
- `emp(clk)(a)` jumps to `a:` 10ms in the future

Timing

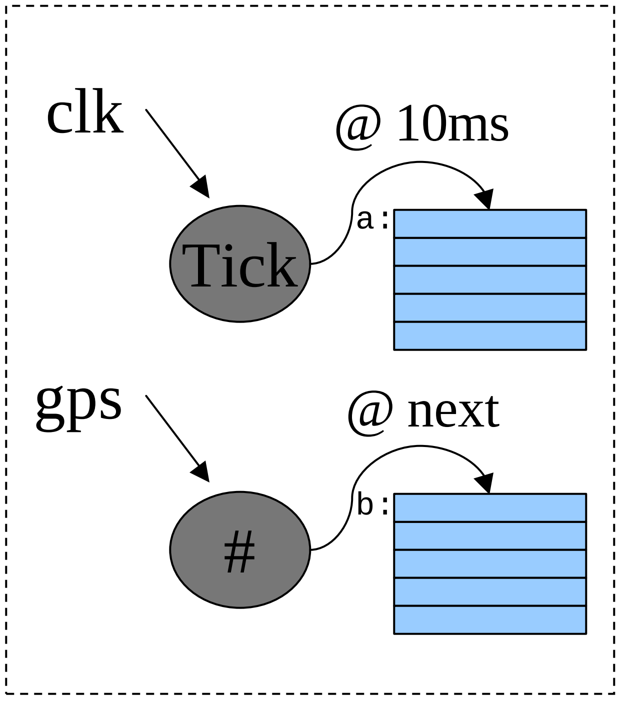


Task
psh(10)
cal(clk)(f)
emp(clk)(a)
pop
ret
f
0ms
psh(10)
cal(clk)(f)
emp(clk)(a)
pop
ret
f
0ms
Time
10ms
10ms
14ms
20ms
20ms
26ms

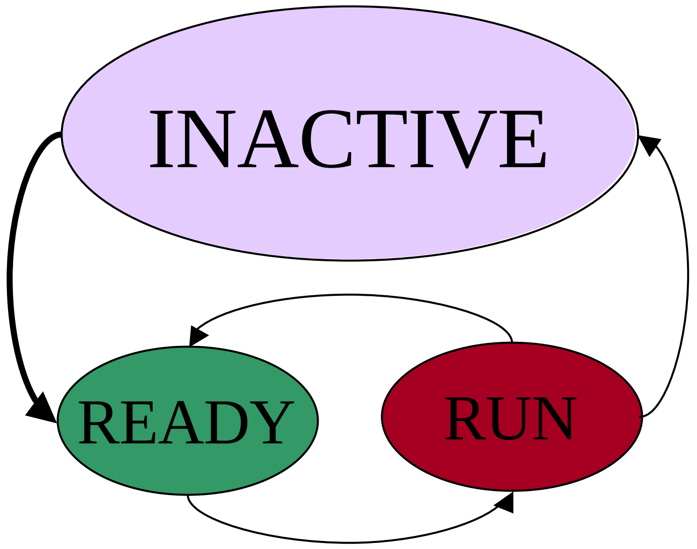
Synchronous vs. Scheduled Computation



Synchronous vs. Scheduled Computation



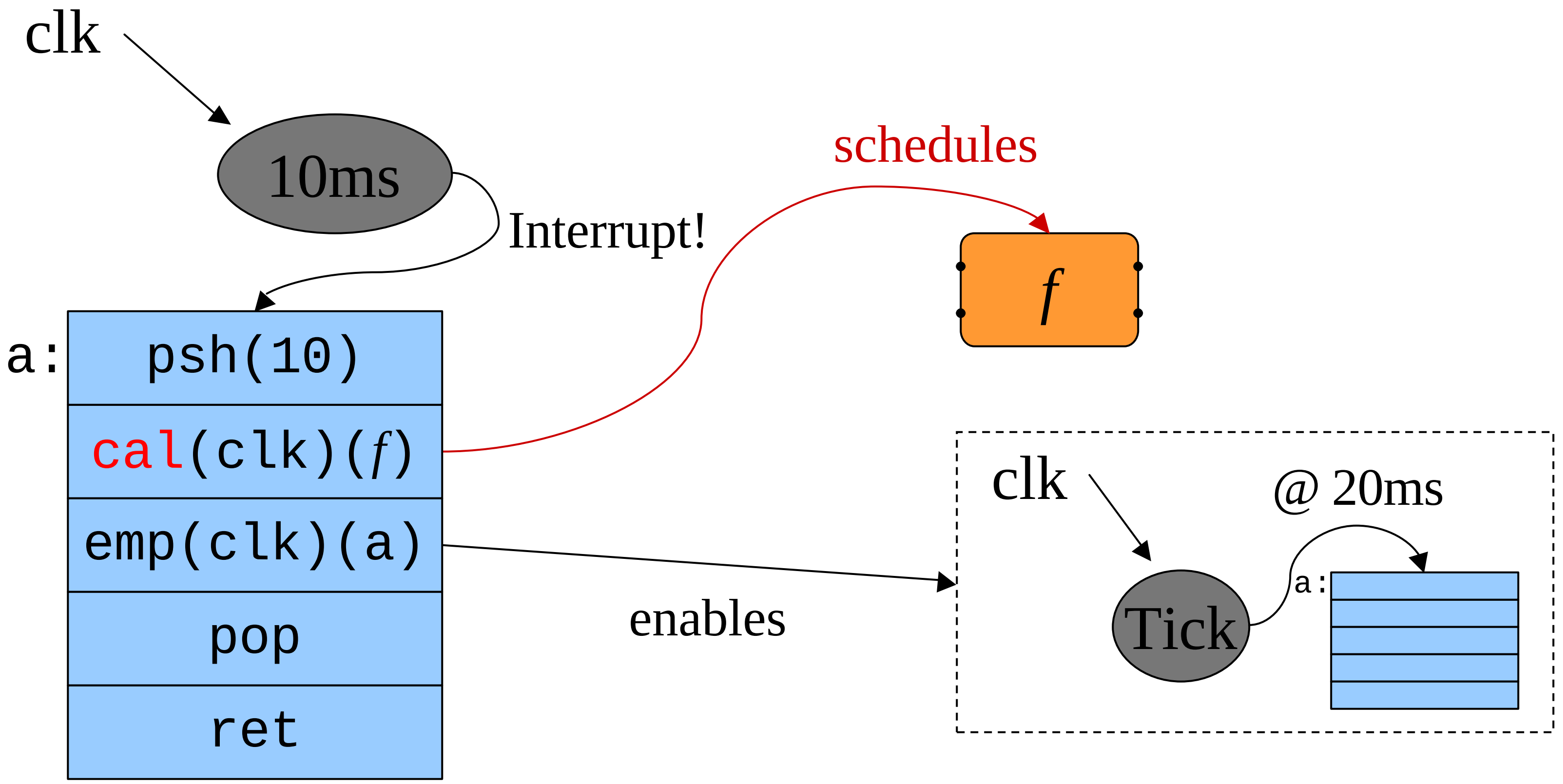
schedules



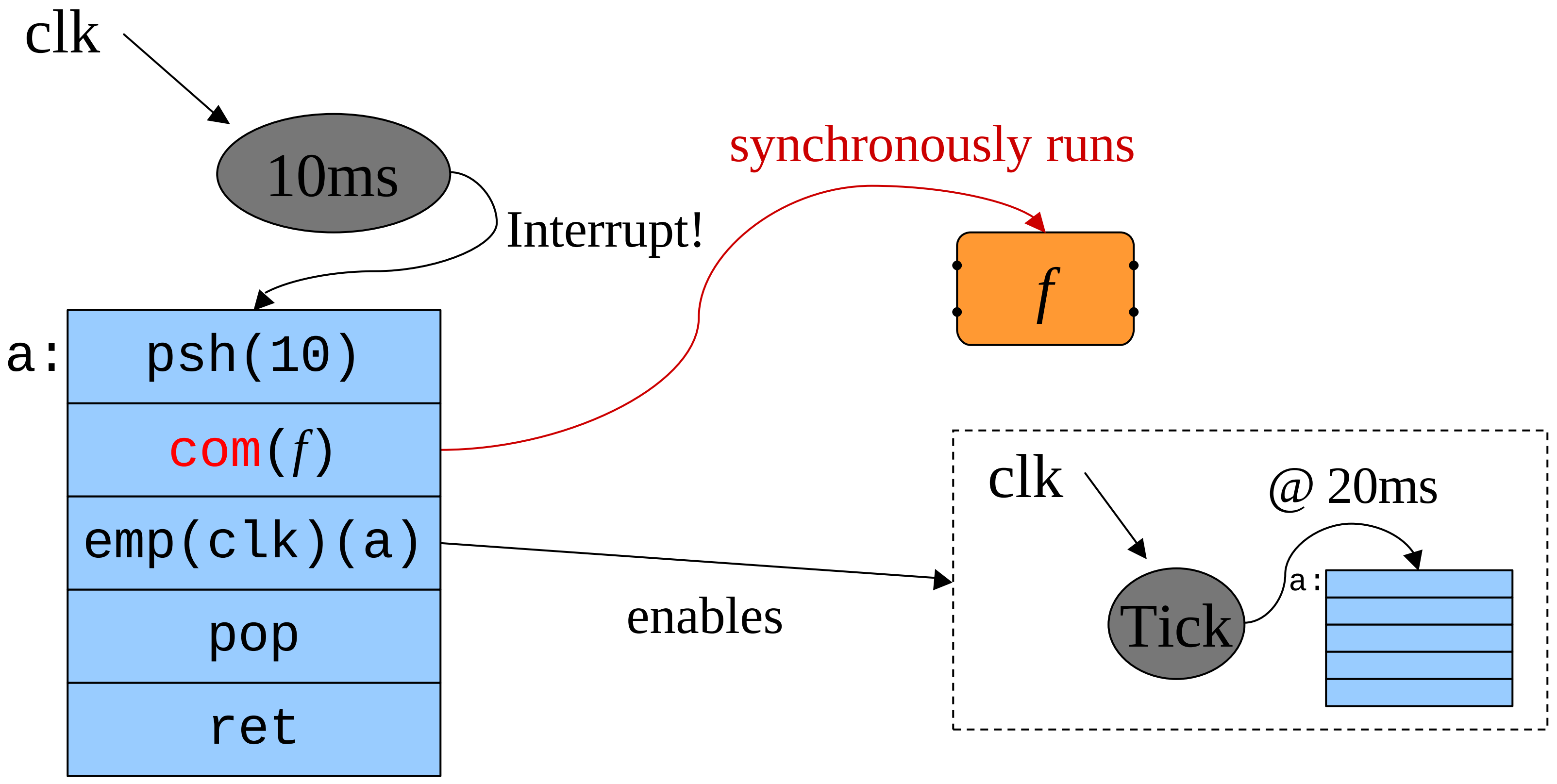
- Synchronous computation
- Kernel context
- Trigger related interrupts disabled

- Scheduled computation
- User context

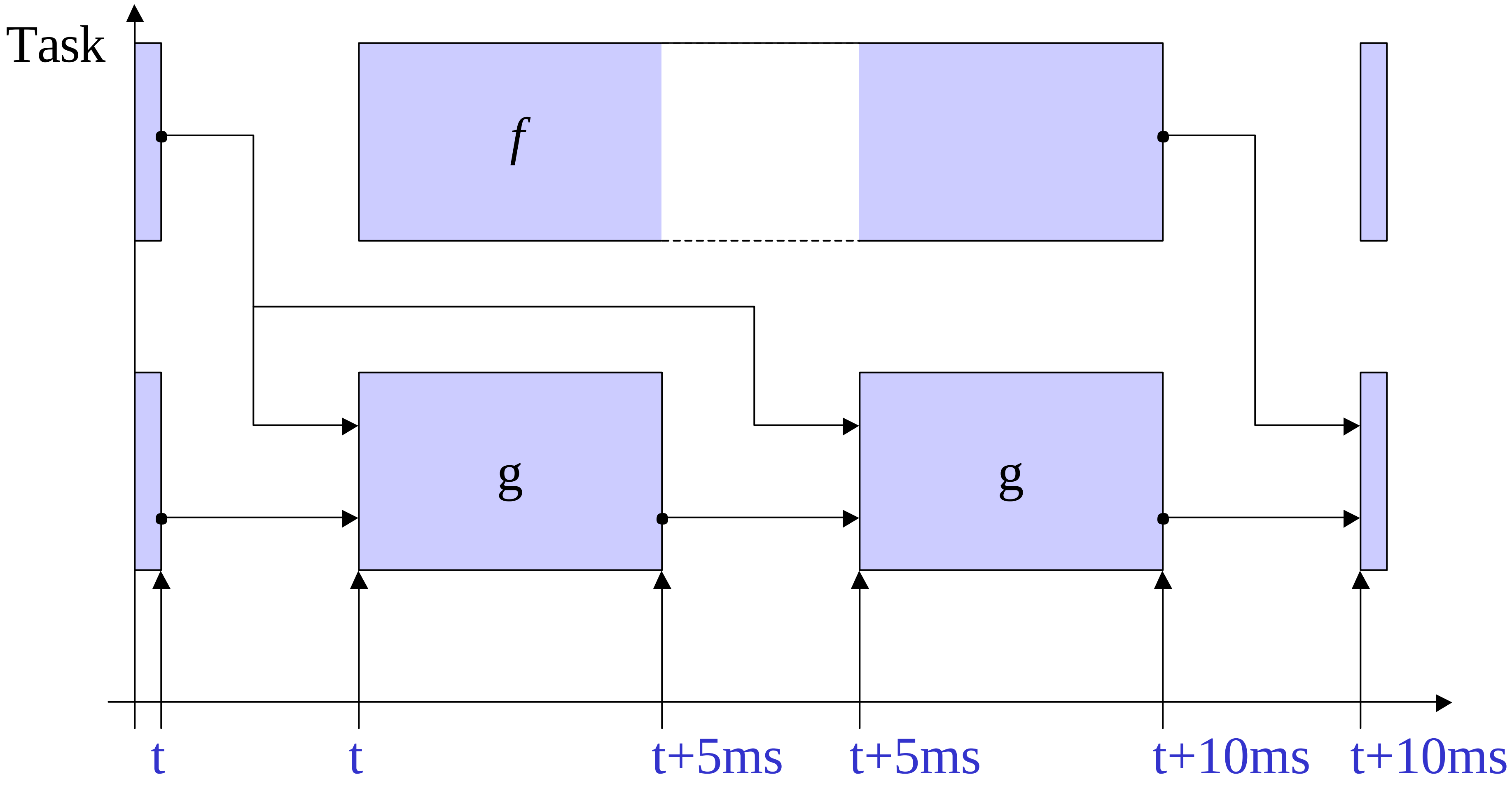
More E Code


clk
10ms
Interrupt!
schedules
f
a:
psh(10)
cal(clk)(*f*)
emp(clk)(a)
pop
ret
enables
clk
@ 20ms
Tick
a:

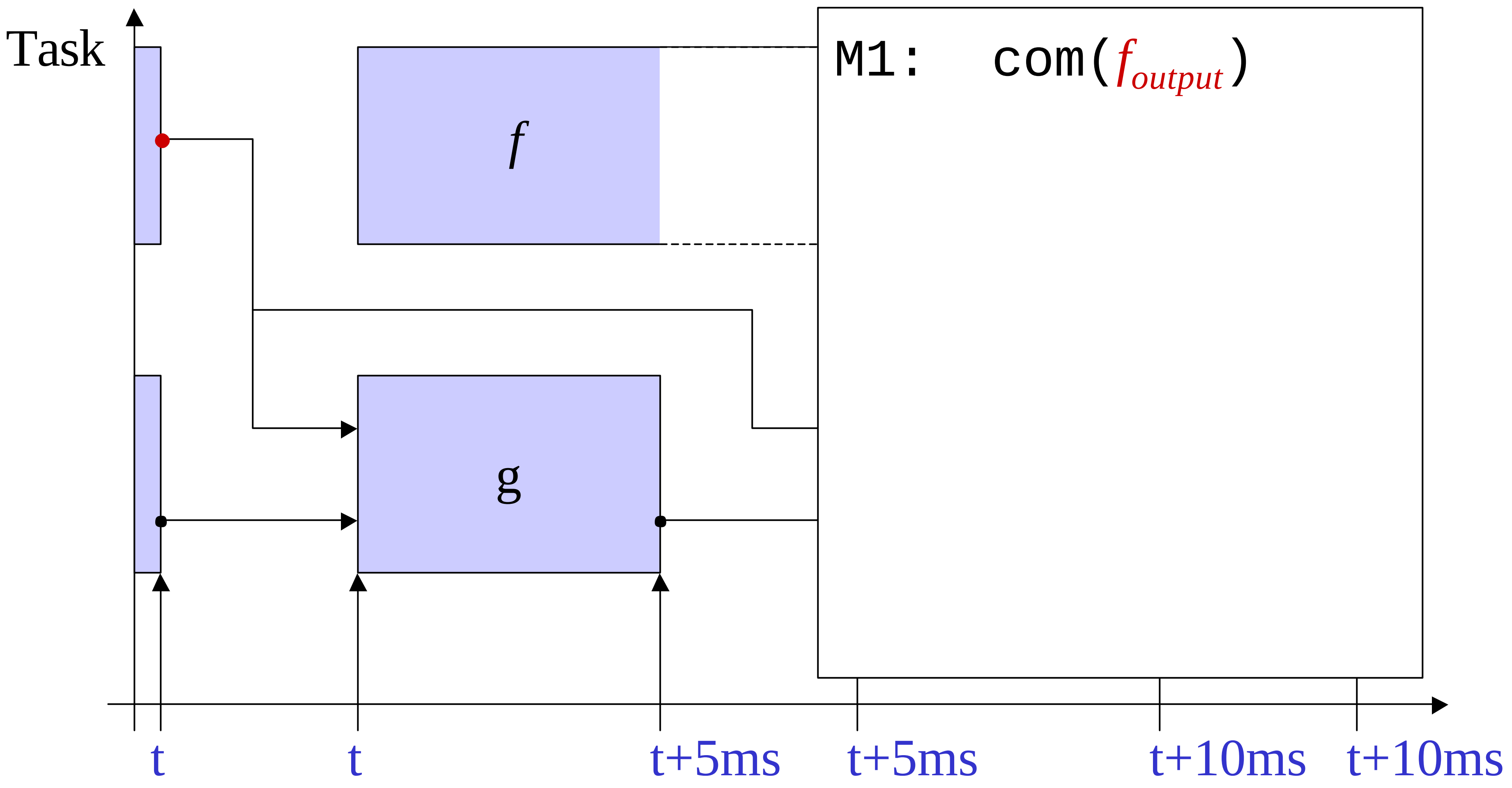
More E Code


clk
10ms
Interrupt!
synchronously runs
f
a:
psh(10)
com(f)
emp(clk)(a)
pop
ret
enables
clk
@ 20ms
Tick
a:

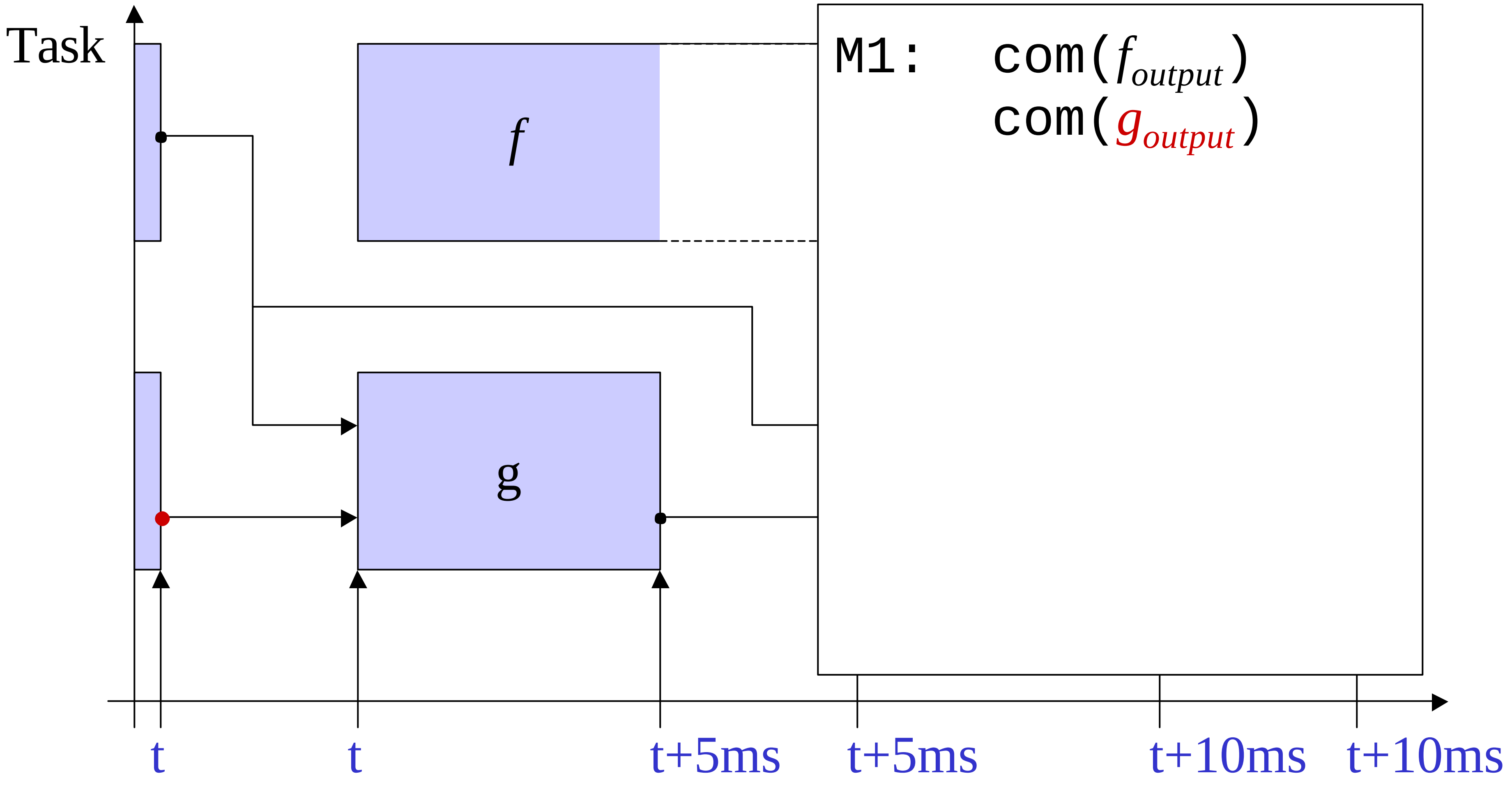
Giotto Example



Update *f*'s Output Port

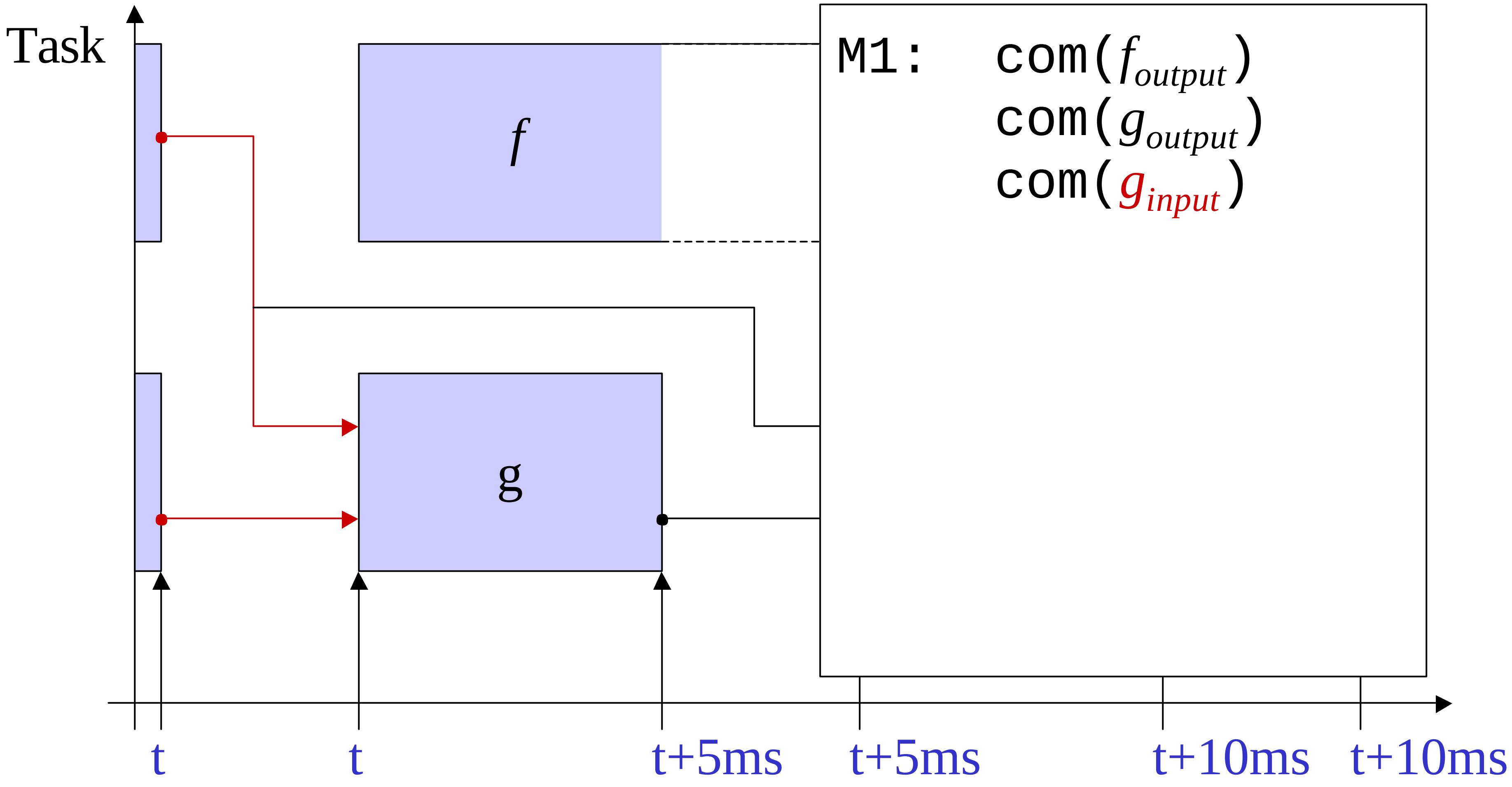


Update *g*'s Output Port

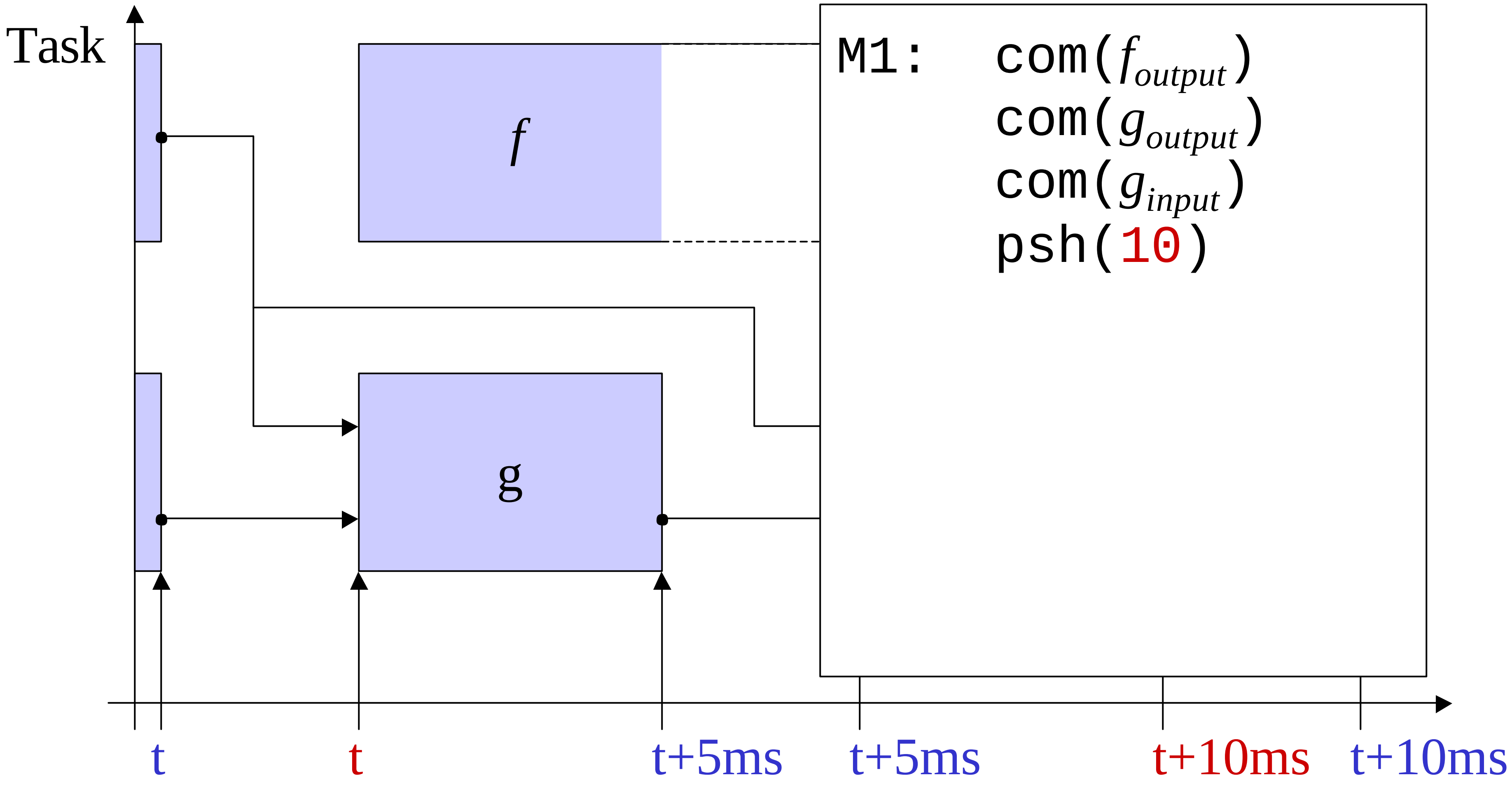


```
M1:   com(f_output)
      com(g_output)
```

Load *g*'s Input Ports

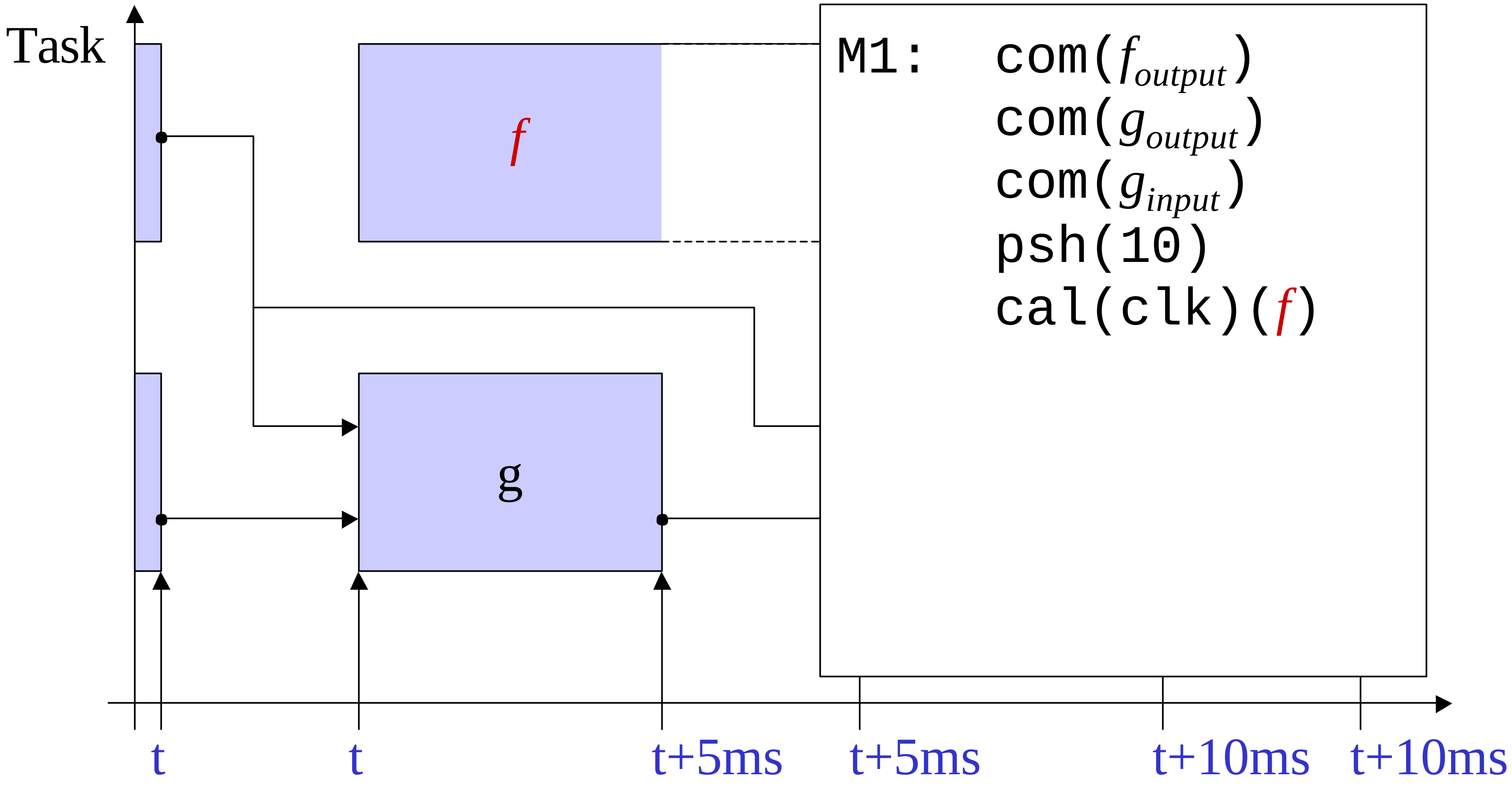


f's Deadline



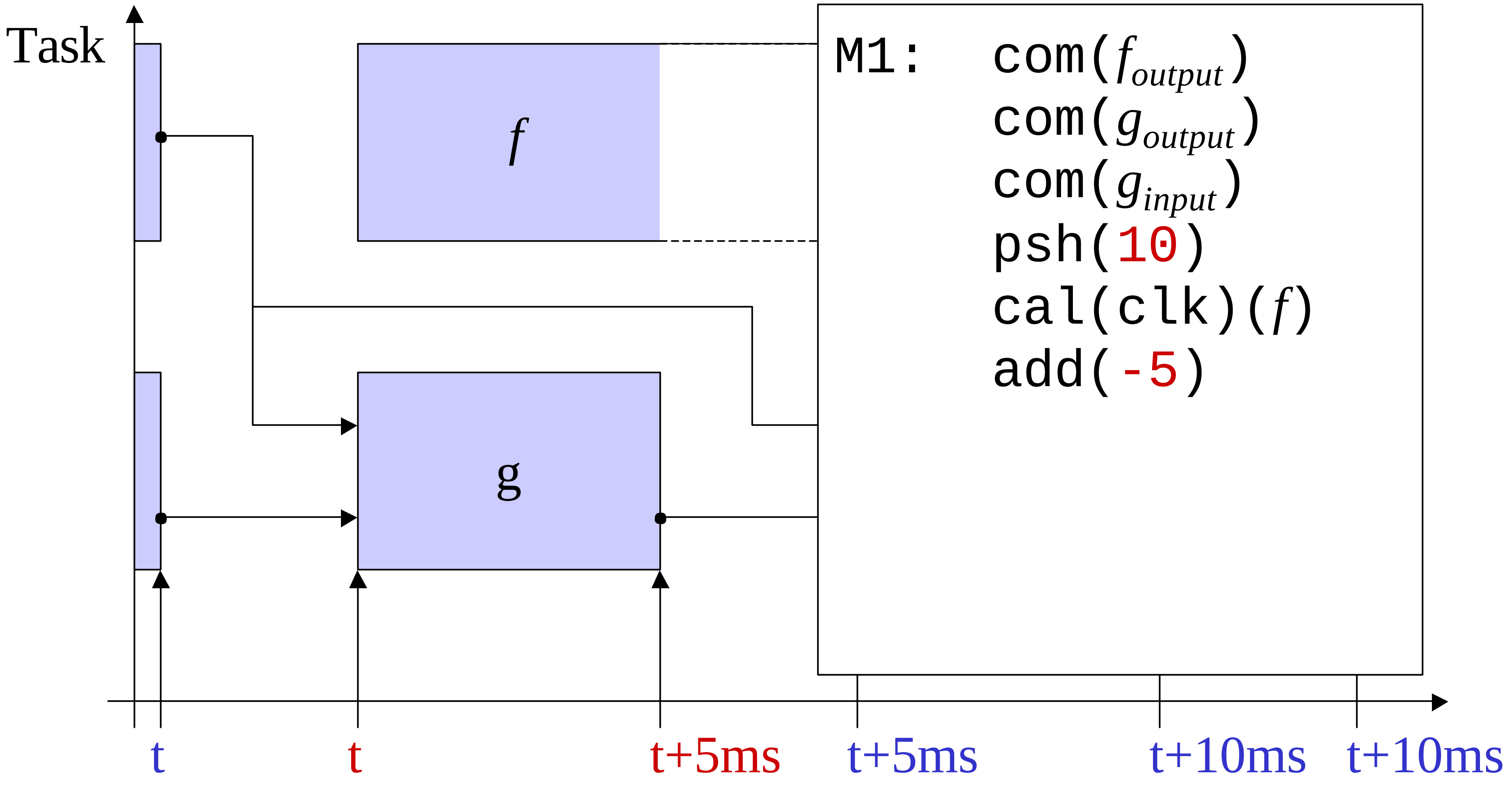
```
M1:   com($f_{output}$)
      com($g_{output}$)
      com($g_{input}$)
      psh(10)
```

Schedule *f*



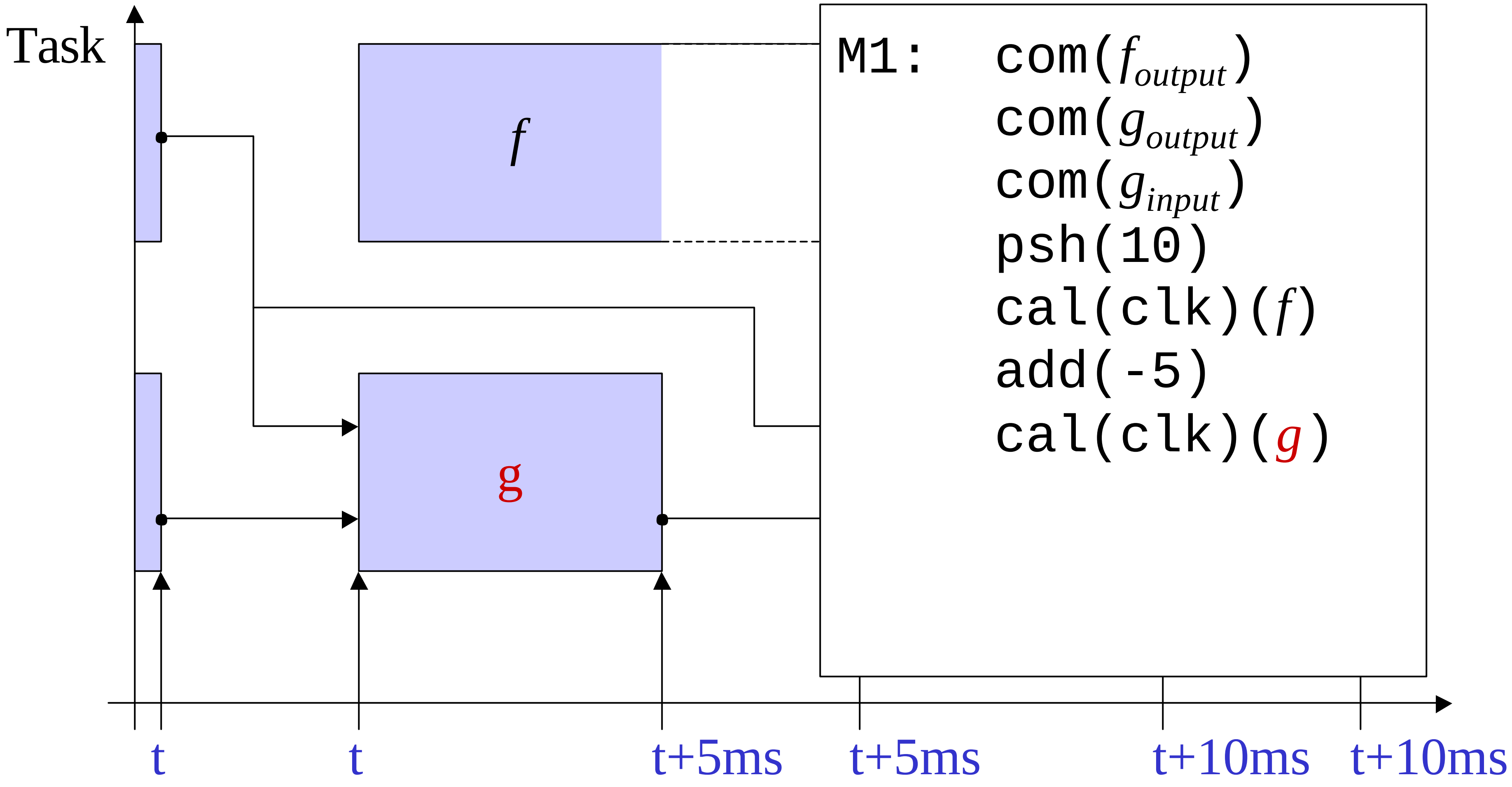
```
M1:  com(f_output)
     com(g_output)
     com(g_input)
     psh(10)
     cal(clk)(f)
```

g's Deadline



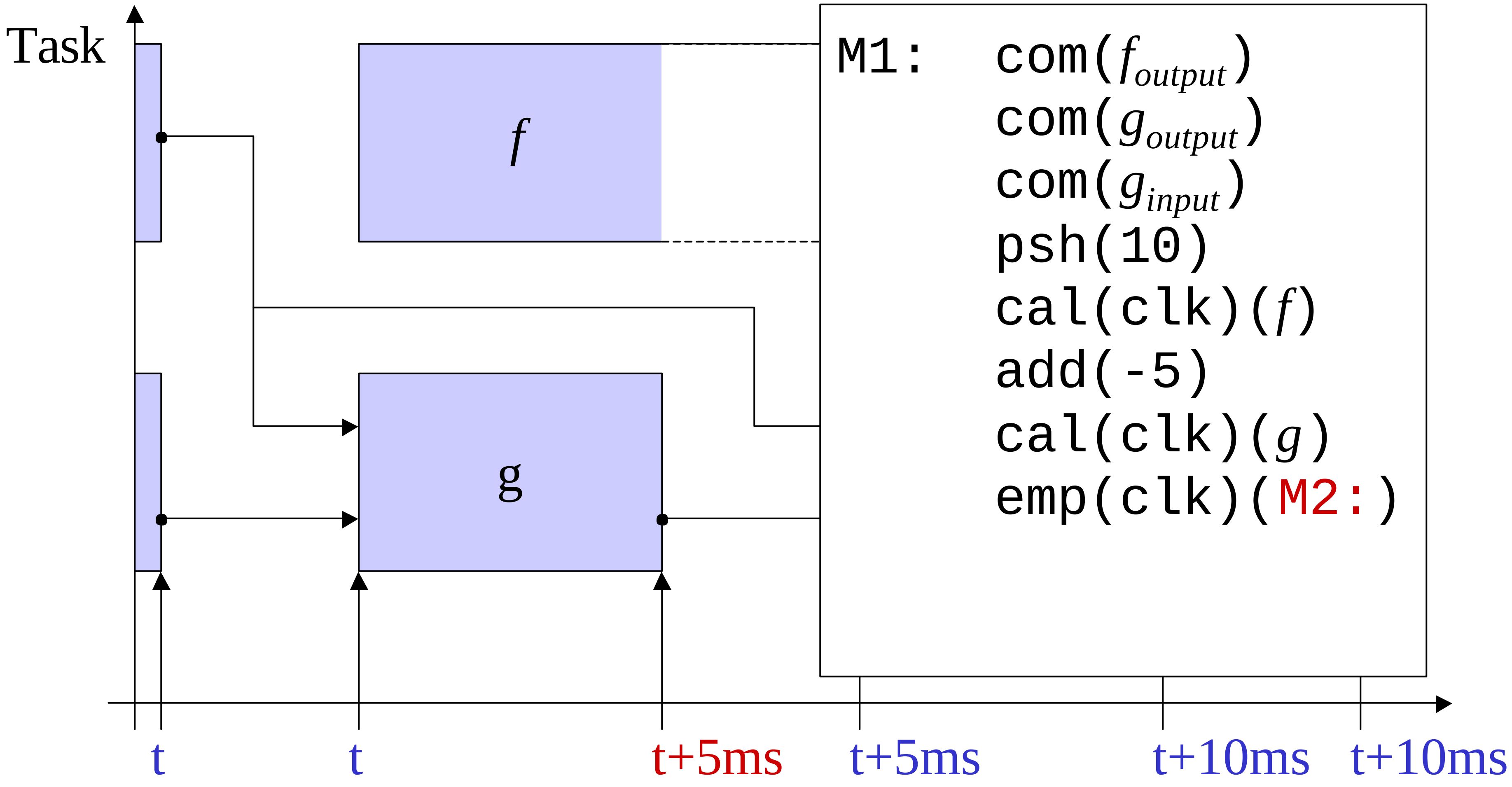
```
M1:  com(f_output)
     com(g_output)
     com(g_input)
     psh(10)
     cal(clk)(f)
     add(-5)
```

Schedule *g*



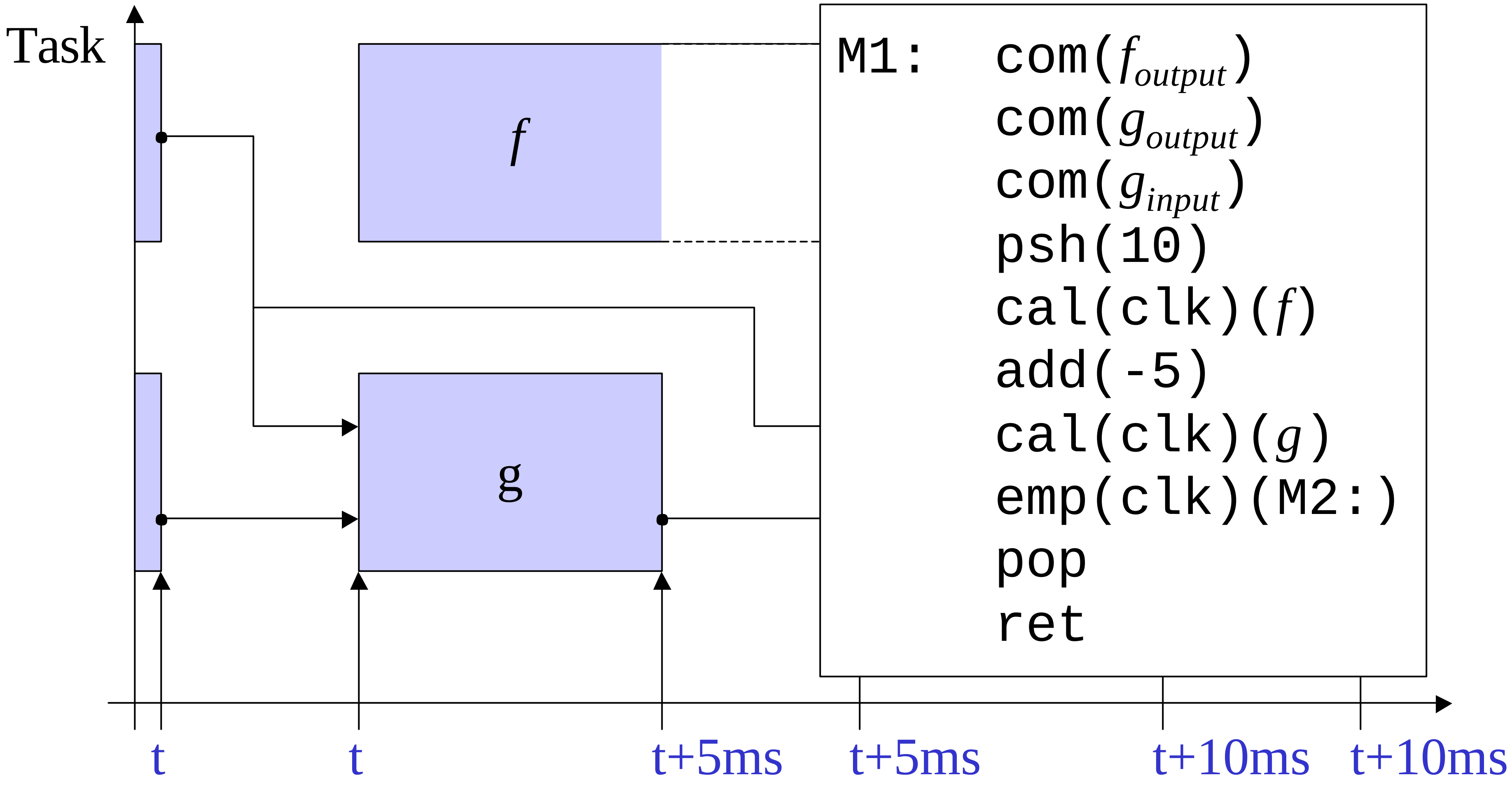
```
M1:   com(f_output)
      com(g_output)
      com(g_input)
      psh(10)
      cal(clk)(f)
      add(-5)
      cal(clk)(g)
```

Schedule Myself



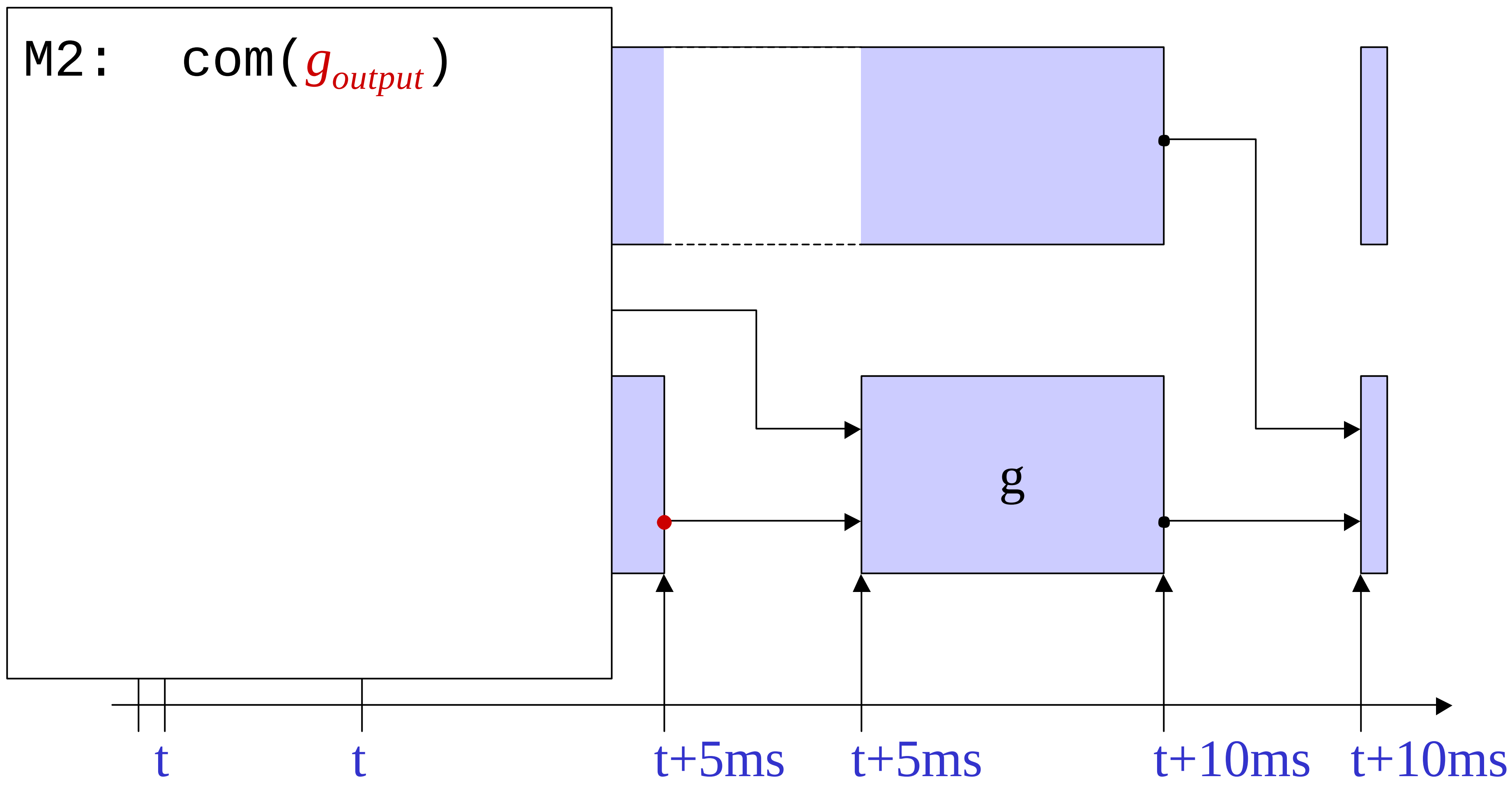
```
M1:  com(f_output)
     com(g_output)
     com(g_input)
     psh(10)
     cal(clk)(f)
     add(-5)
     cal(clk)(g)
     emp(clk)(M2:)
```

Exit

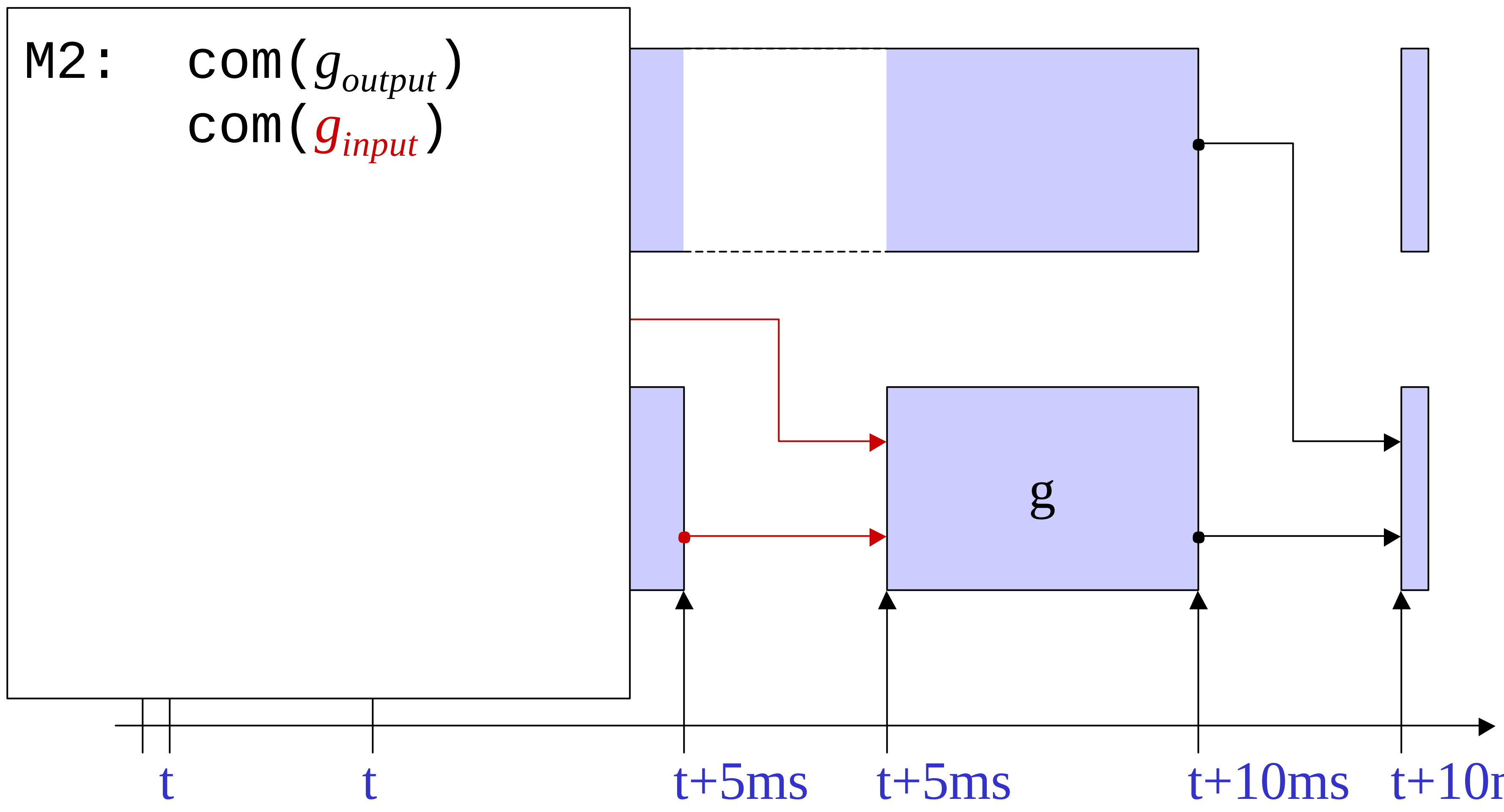


```
M1:  com(f_output)
     com(g_output)
     com(g_input)
     psh(10)
     cal(clk)(f)
     add(-5)
     cal(clk)(g)
     emp(clk)(M2:)
     pop
     ret
```

Update *g*'s Output Port



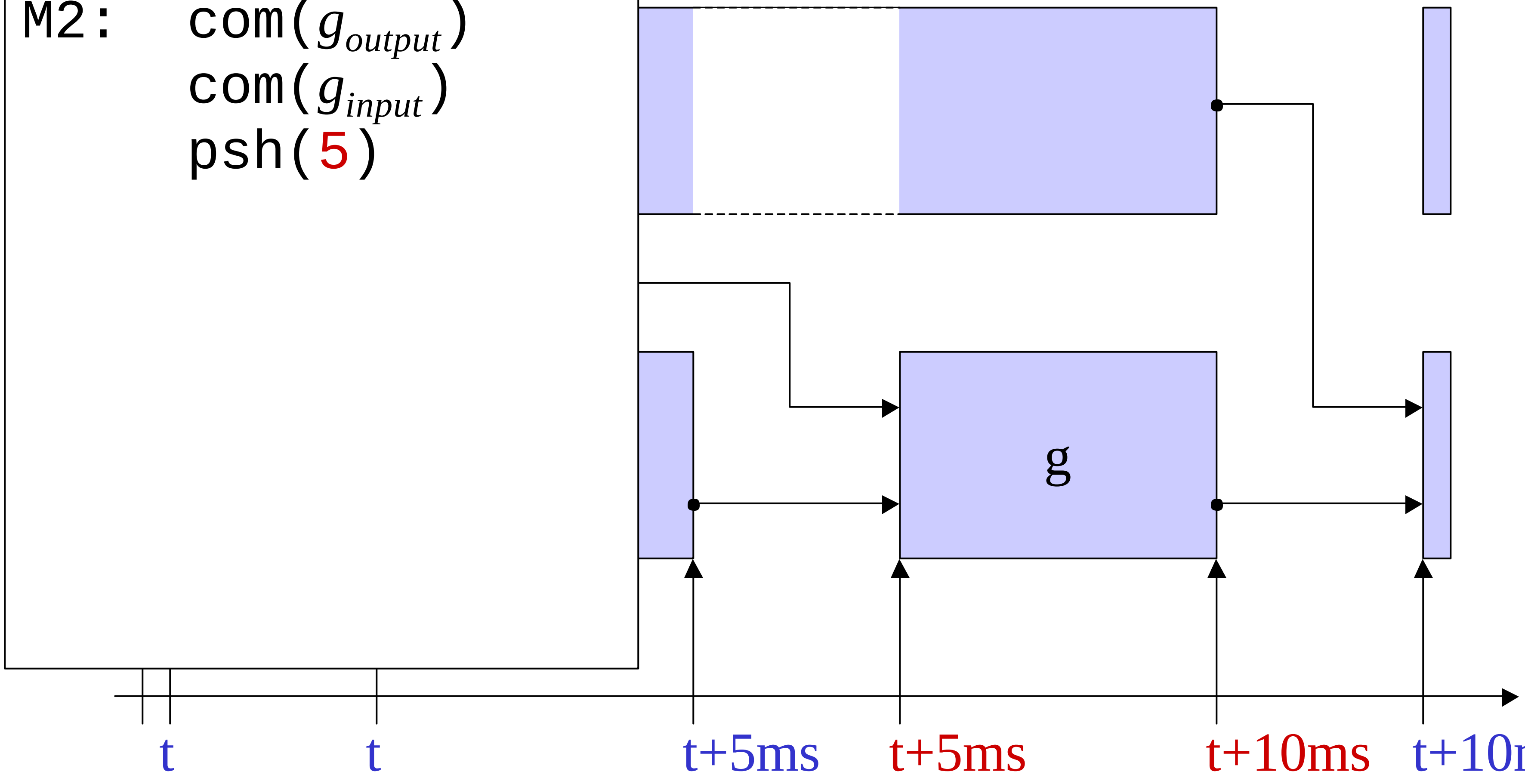
Load *g*'s Input Ports



```
M2:   com(goutput)
      com(ginput)
```

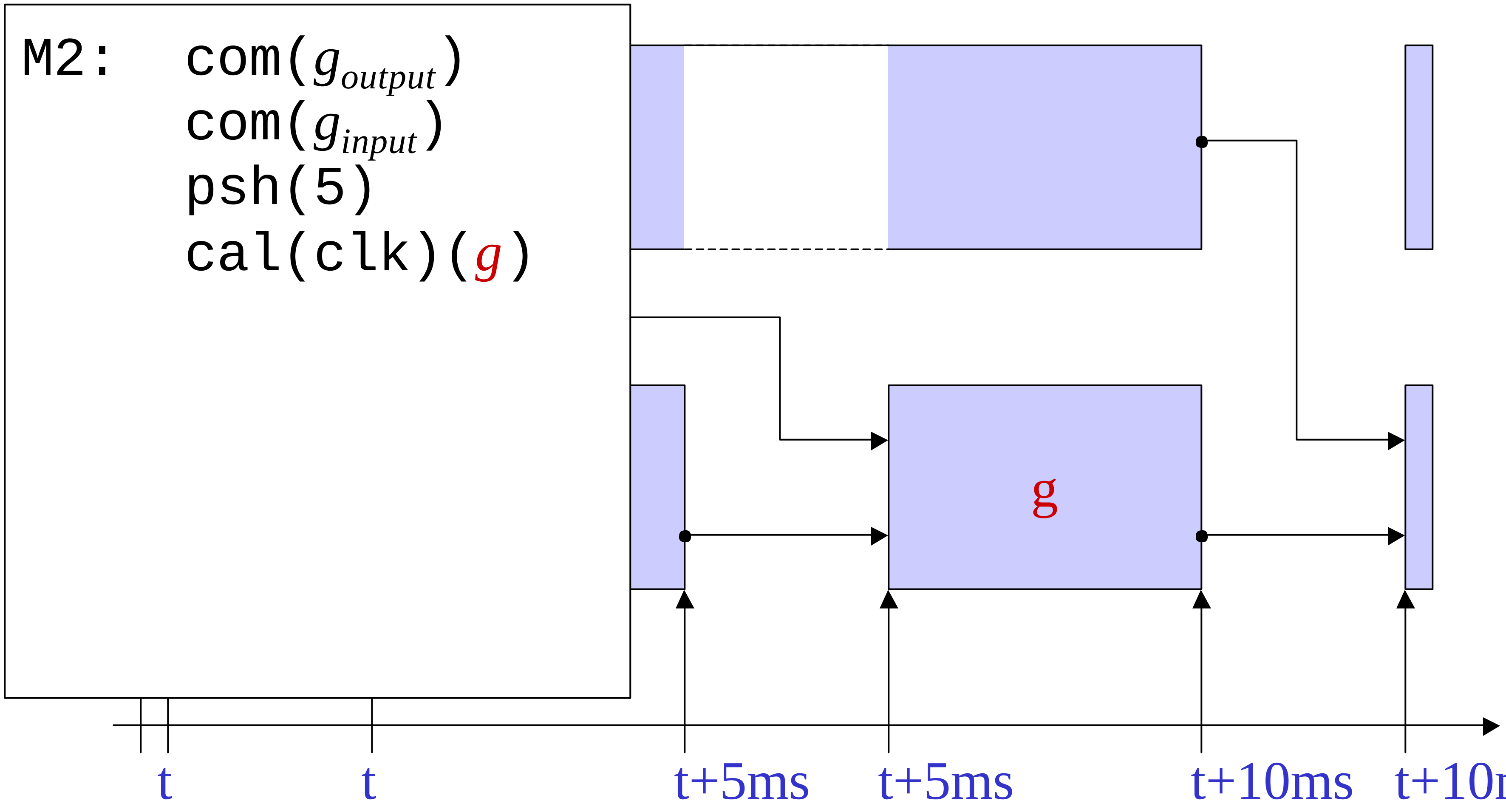
g's Deadline

```
M2:   com(g_output)
      com(g_input)
      psh(5)
```



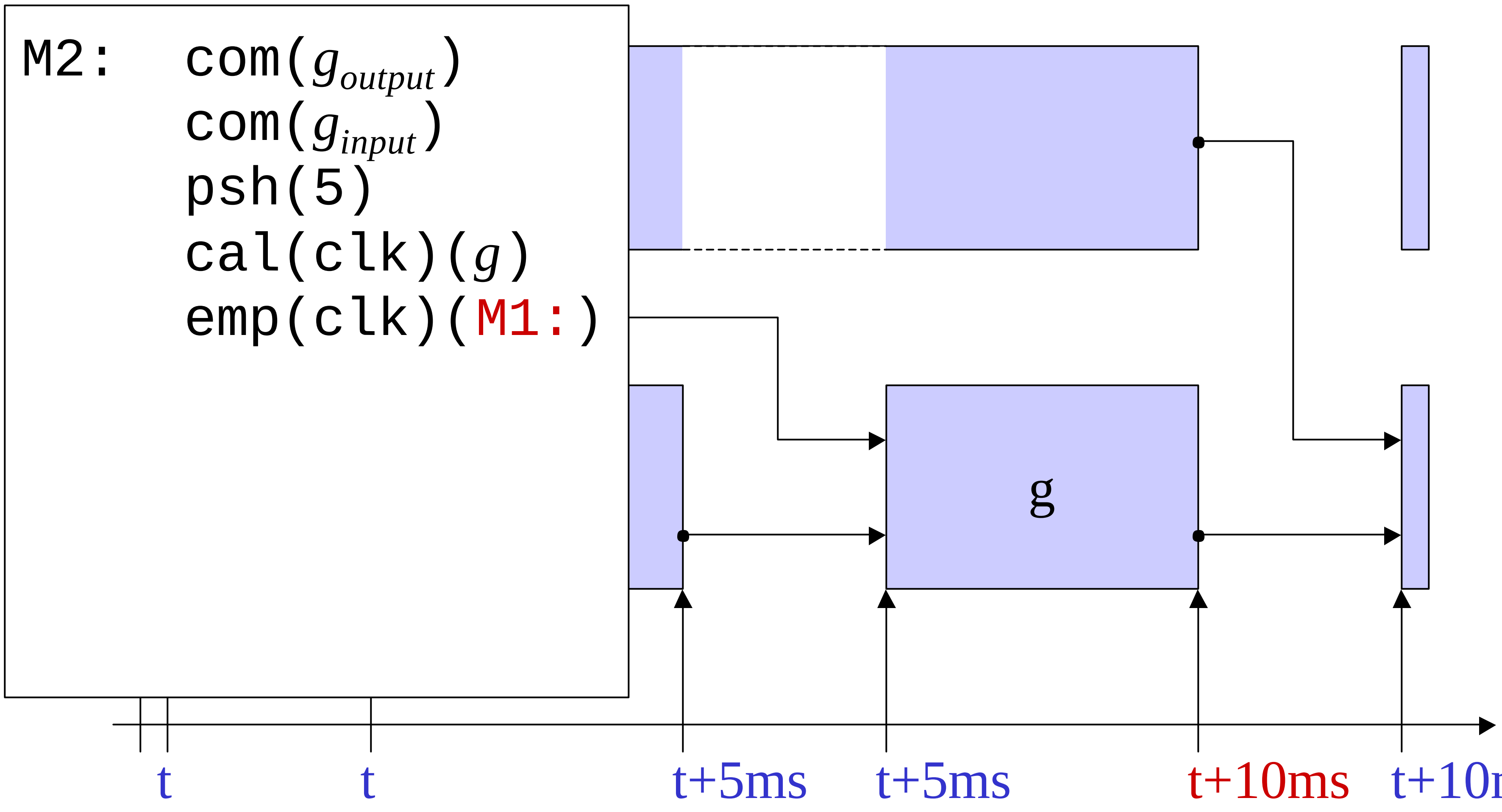
Schedule g

```
M2:   com(g_output)
      com(g_input)
      psh(5)
      cal(clk)(g)
```

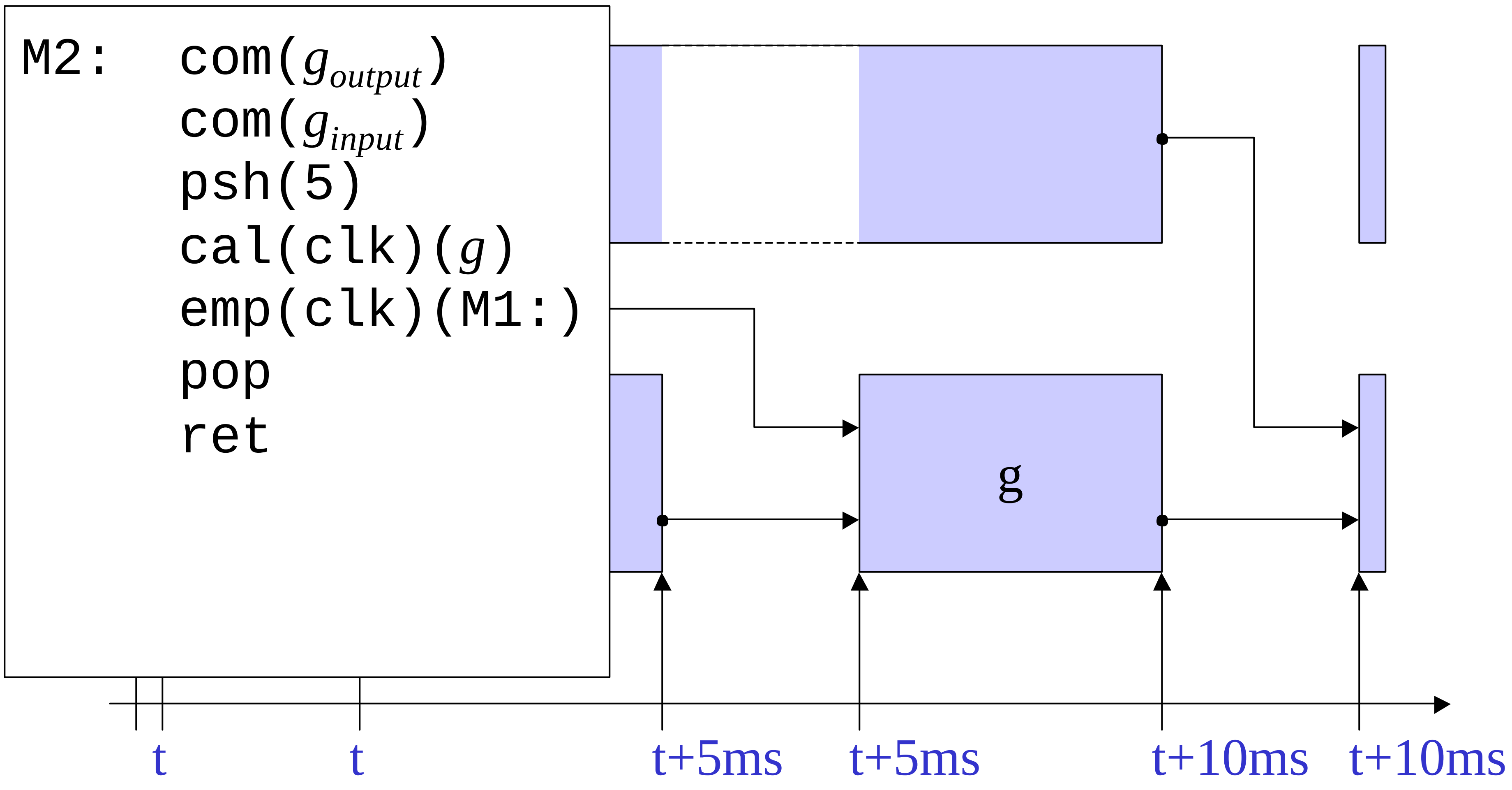


Schedule Myself

```
M2:   com(g_output)
      com(g_input)
      psh(5)
      cal(clk)(g)
      emp(clk)(M1:)
```



Exit



M2: com(g_{output})
com(g_{input})
psh(5)
cal(clk)(g)
emp(clk)(M1:)
pop
ret

Embedded Programming

…requires the integration of:

1. Real-time scheduling/communication concepts
2. Logical RTOS: The embedded machine
3. Programming language design
4. Compiler design
5. Classical software engineering techniques
6. Formal methods

Concurrency

$\boxed{\text{Parallel Composition}}$

$\text{Task1} \parallel \text{Task2}$

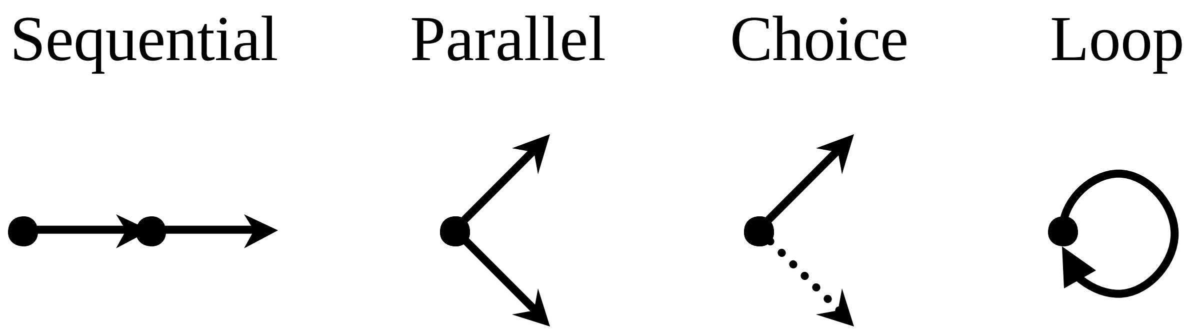
Control

$\boxed{\text{I/O Decomposition}}$

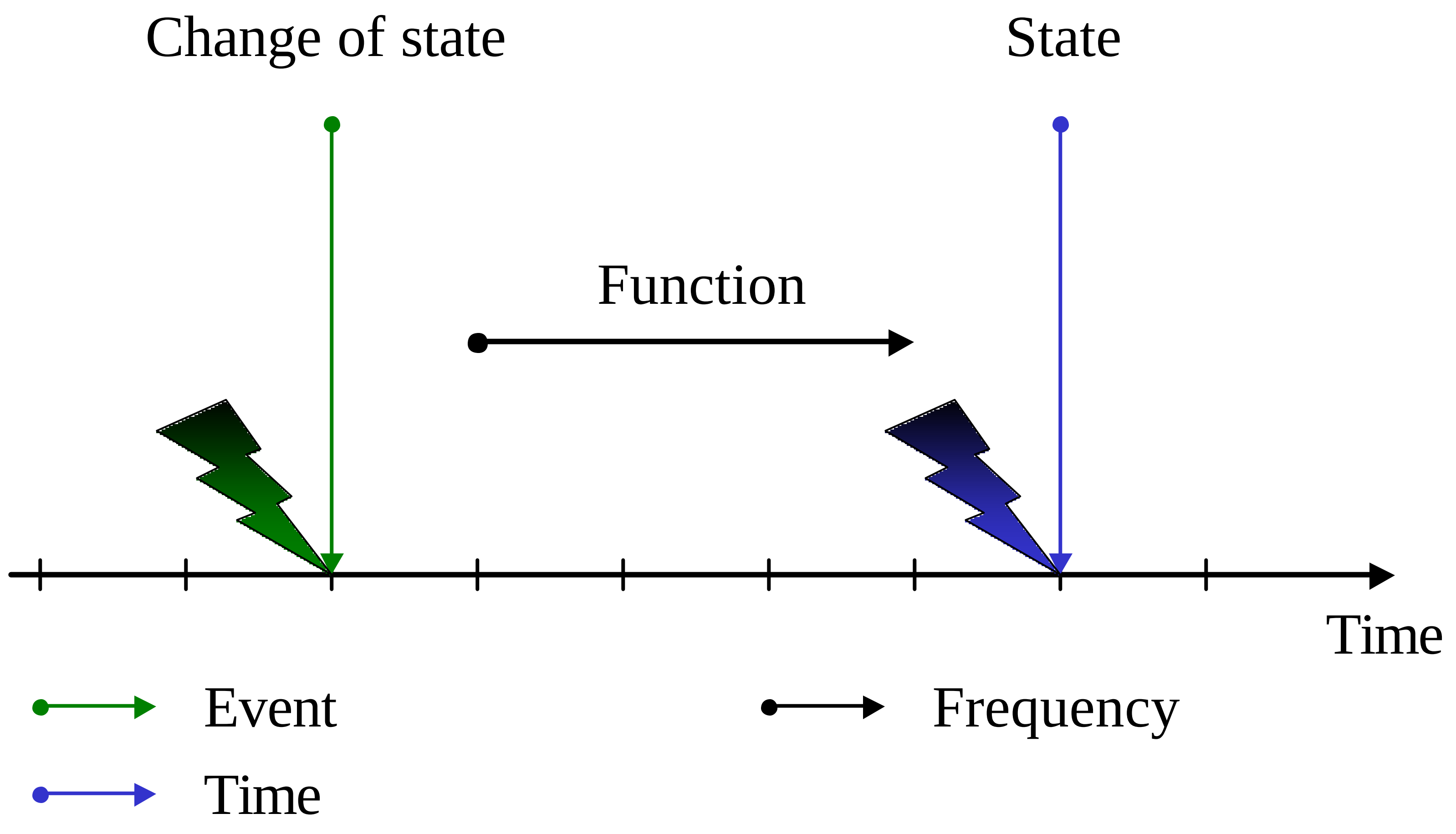
$\text{Task1} \longleftrightarrow \text{Task2}$

Data

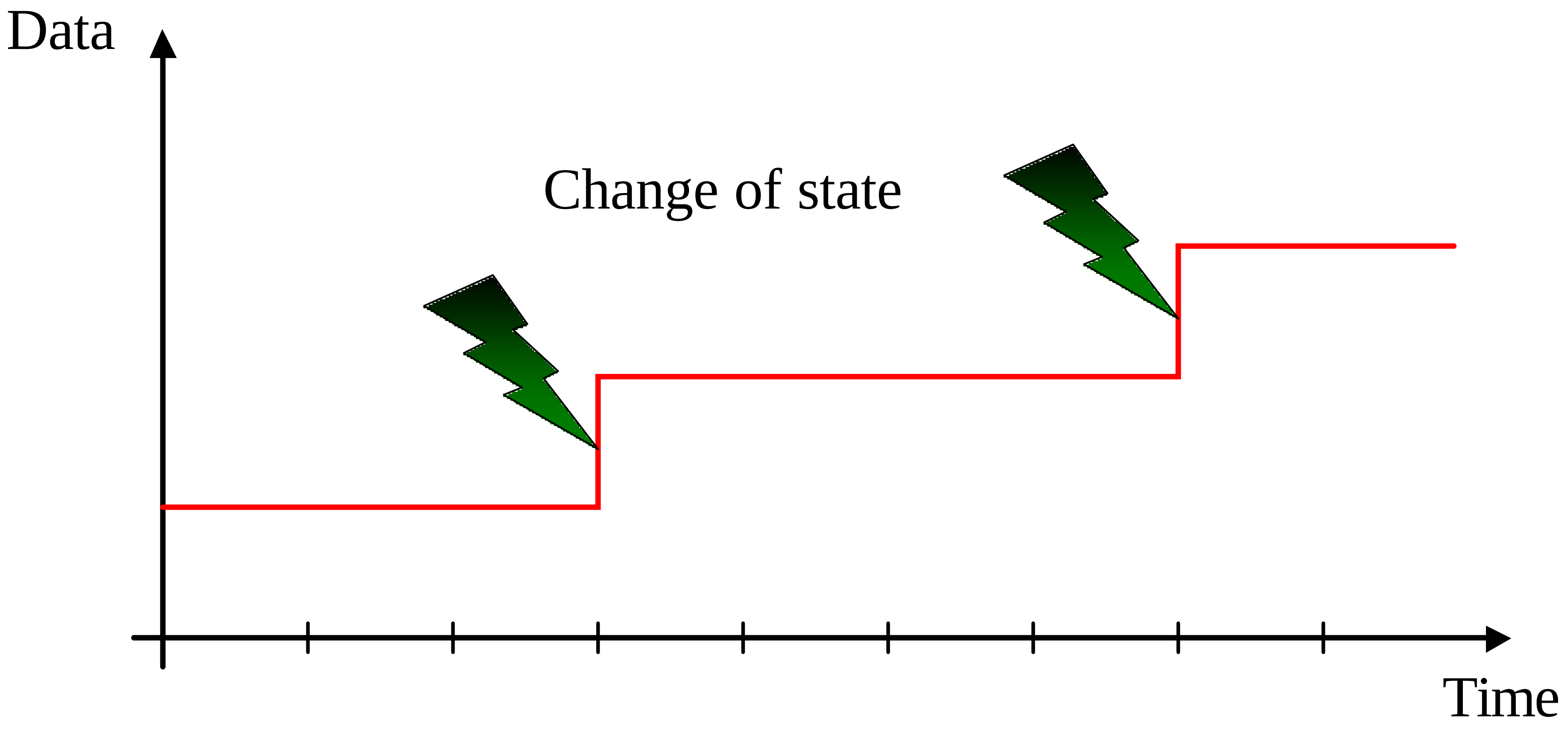
Control Operators



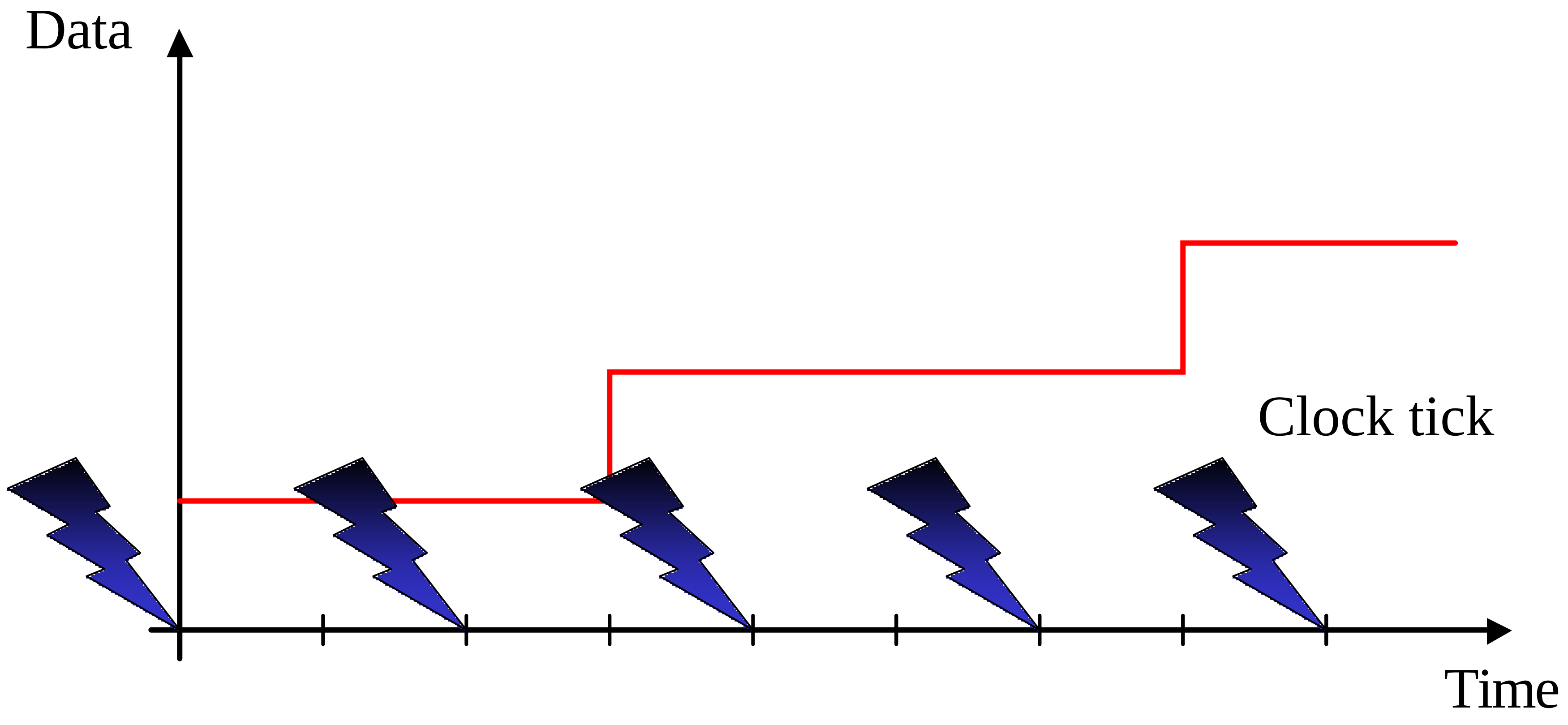
Real-Time



Event-Triggered (ET) System



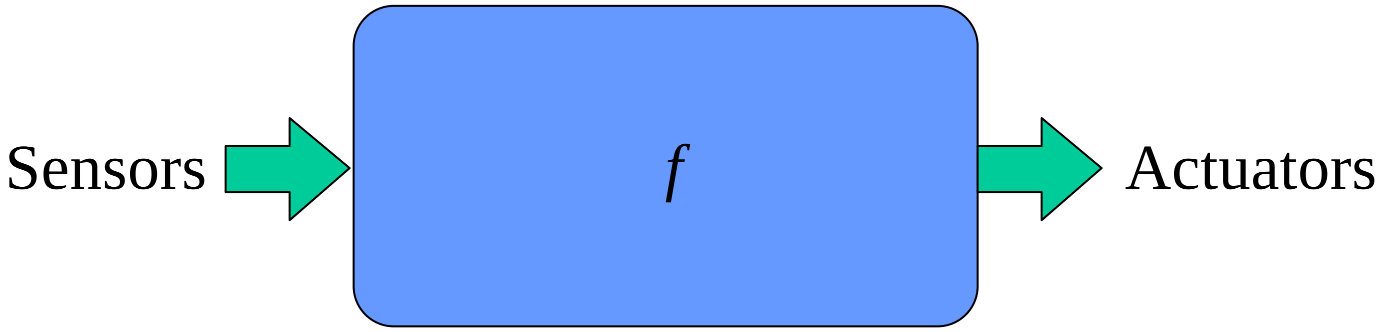
Time-Triggered (TT) System



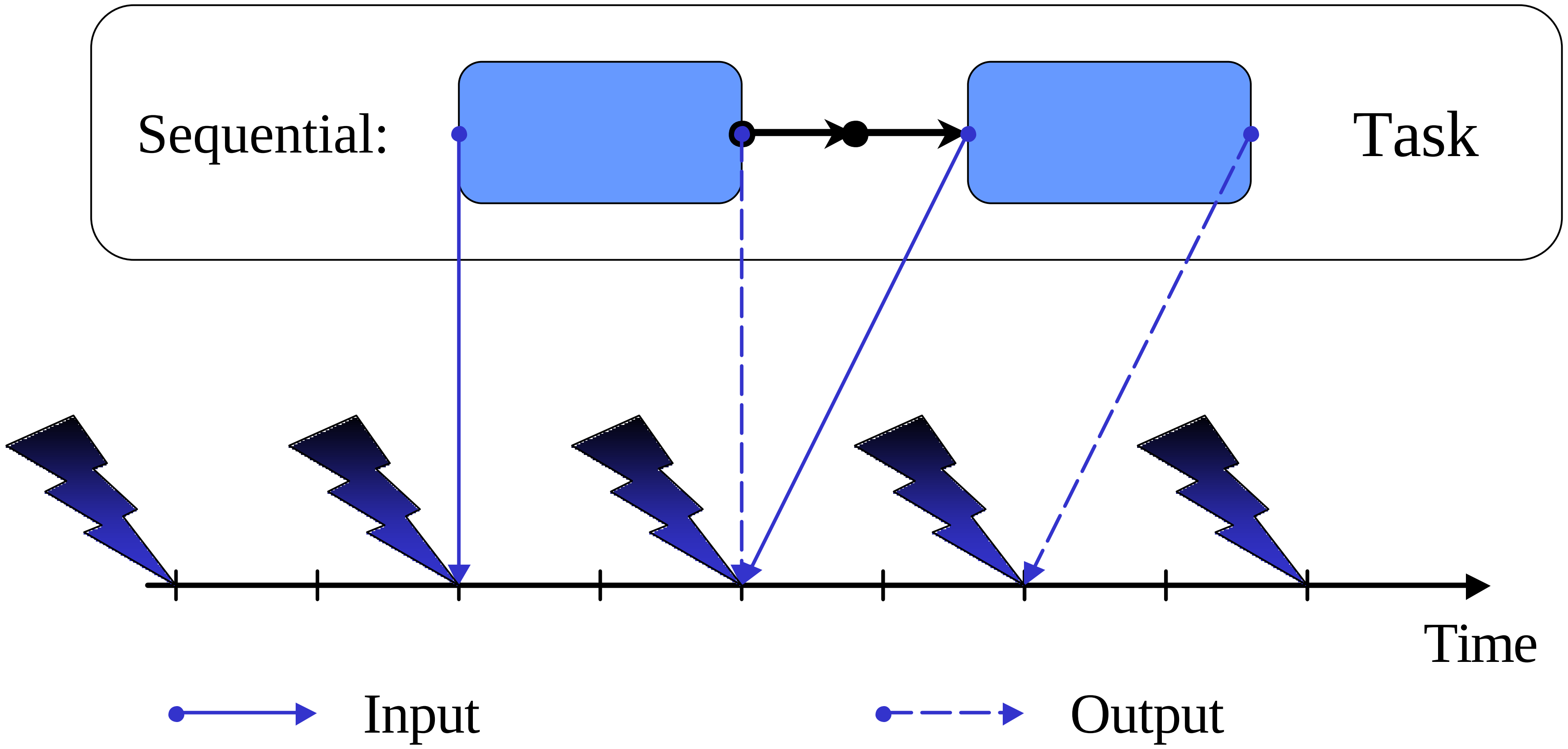
Esterel - Giotto

- Esterel:
  - Synchronous reactive language
  - Event-triggered semantics
- Giotto:
  - Time-triggered semantics
  - Distributed platforms

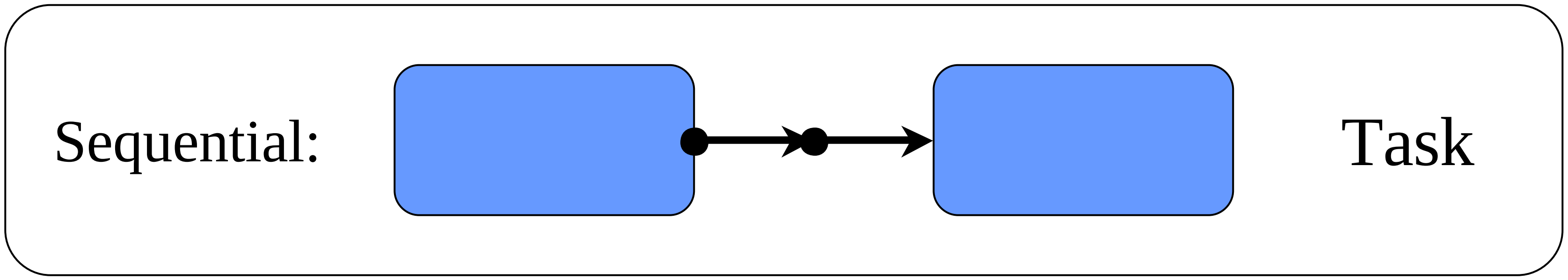
Sensor - Control Law - Actuator

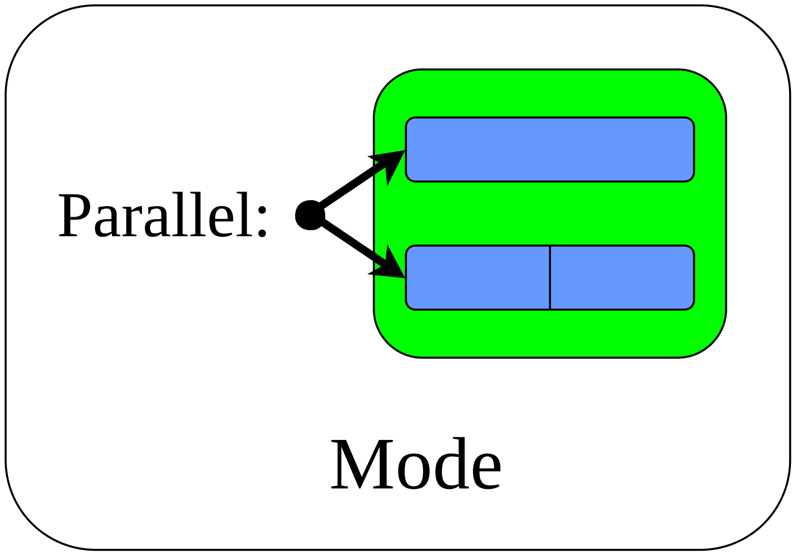


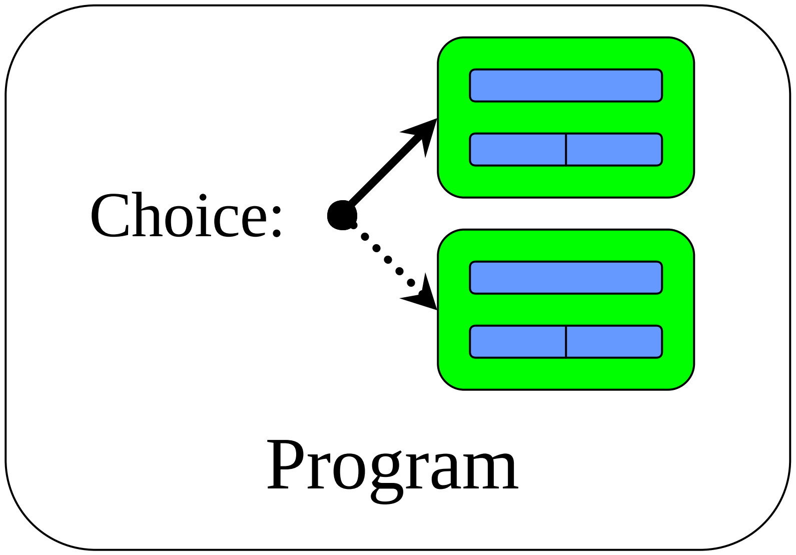
Giotto: Time



Giotto: Operators



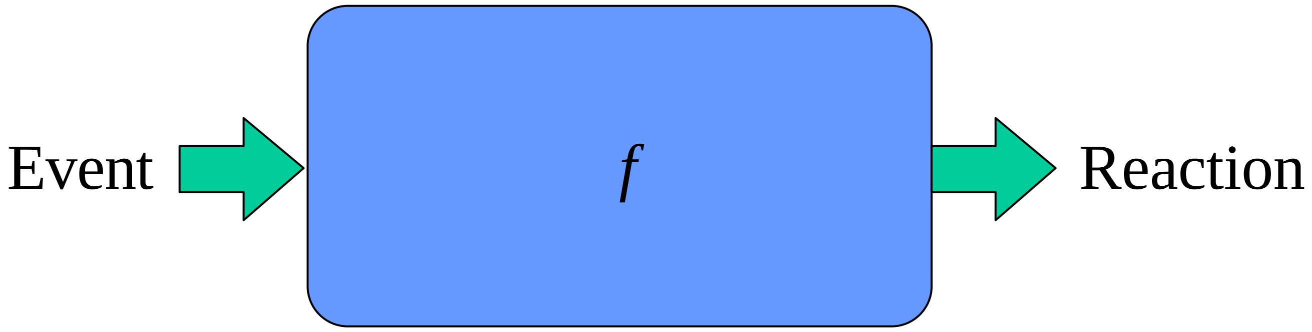




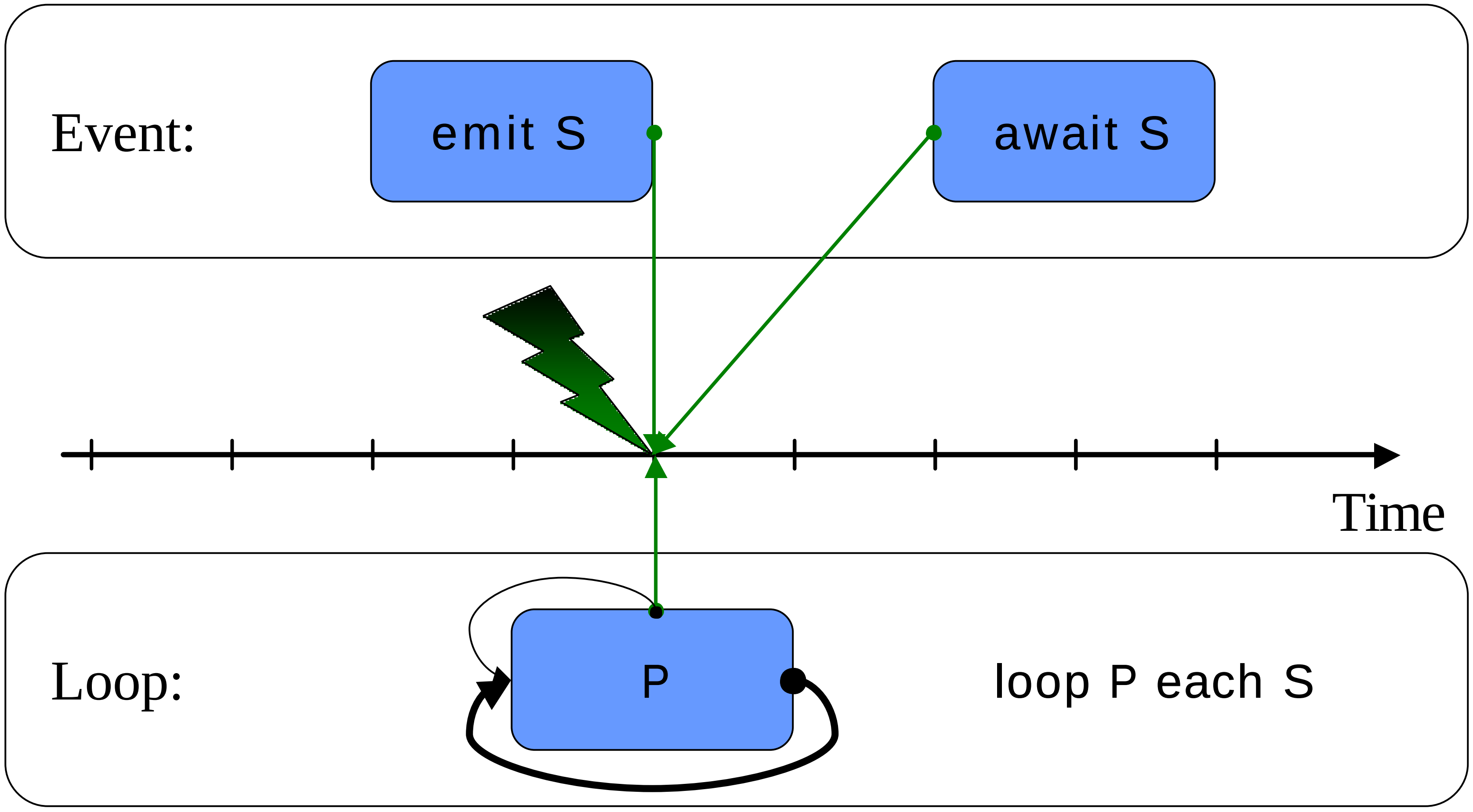
Giotto: Helicopter Control

```
mode normal ( ) period 20ms
  {
      taskfreq 1 do servo  = Control ( position ) ;
      taskfreq 4 do position  = Navigation ( GPS, position ) ;
  }
```

Event - Reaction

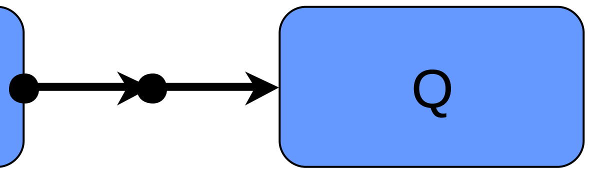


Esterel: Event



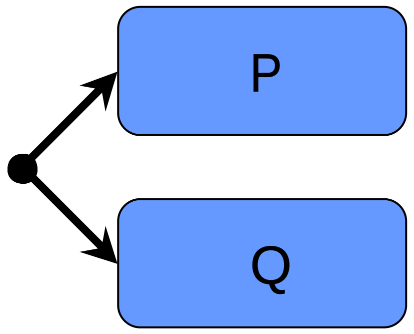
Esterel: Operators

Sequential:



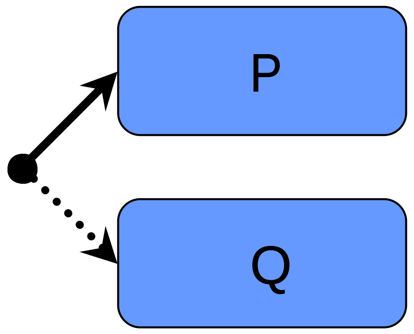
P ; Q

Parallel:



P || Q

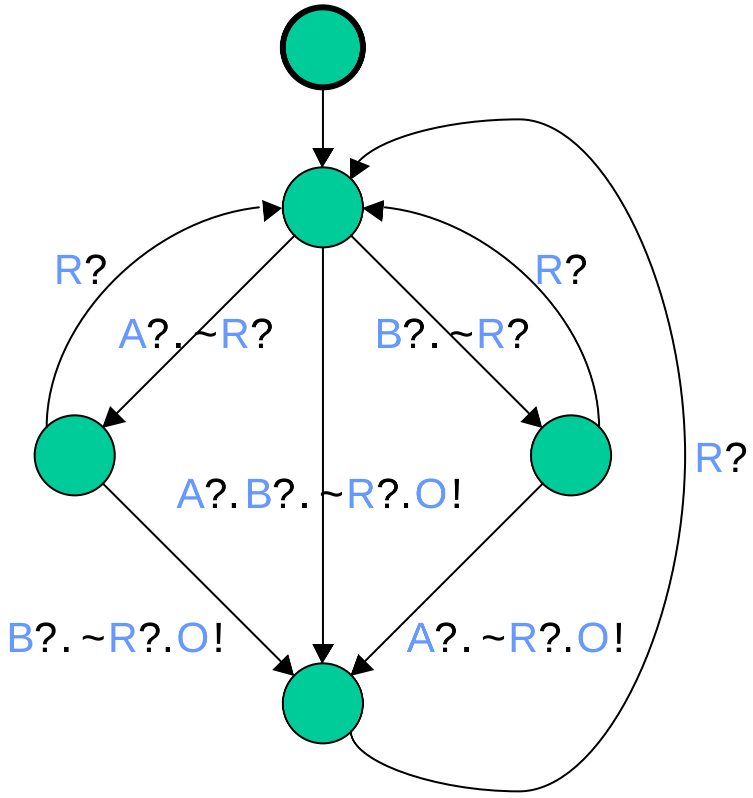
Choice:



present S then P else Q

Esterel: Controller

```
module normal:
  input A, B, R;
  output O;
    loop
      [ await A || await B ];
      emit O
    each R
end module
```



Embedded Programming

…requires the integration of:

1. Real-time scheduling/communication concepts
2. Logical RTOS: The embedded machine
3. Programming language design
4. Compiler design
5. Classical software engineering techniques
6. Formal methods

Concurrency

$\boxed{\text{Parallel Composition}}$

$\text{Task1} \parallel \text{Task2}$

$\Downarrow$

$\text{Task1} \; ; \; \text{Task2}$

$\text{Task2} \; ; \; \text{Task1}$

$\boxed{\text{I/O Decomposition}}$

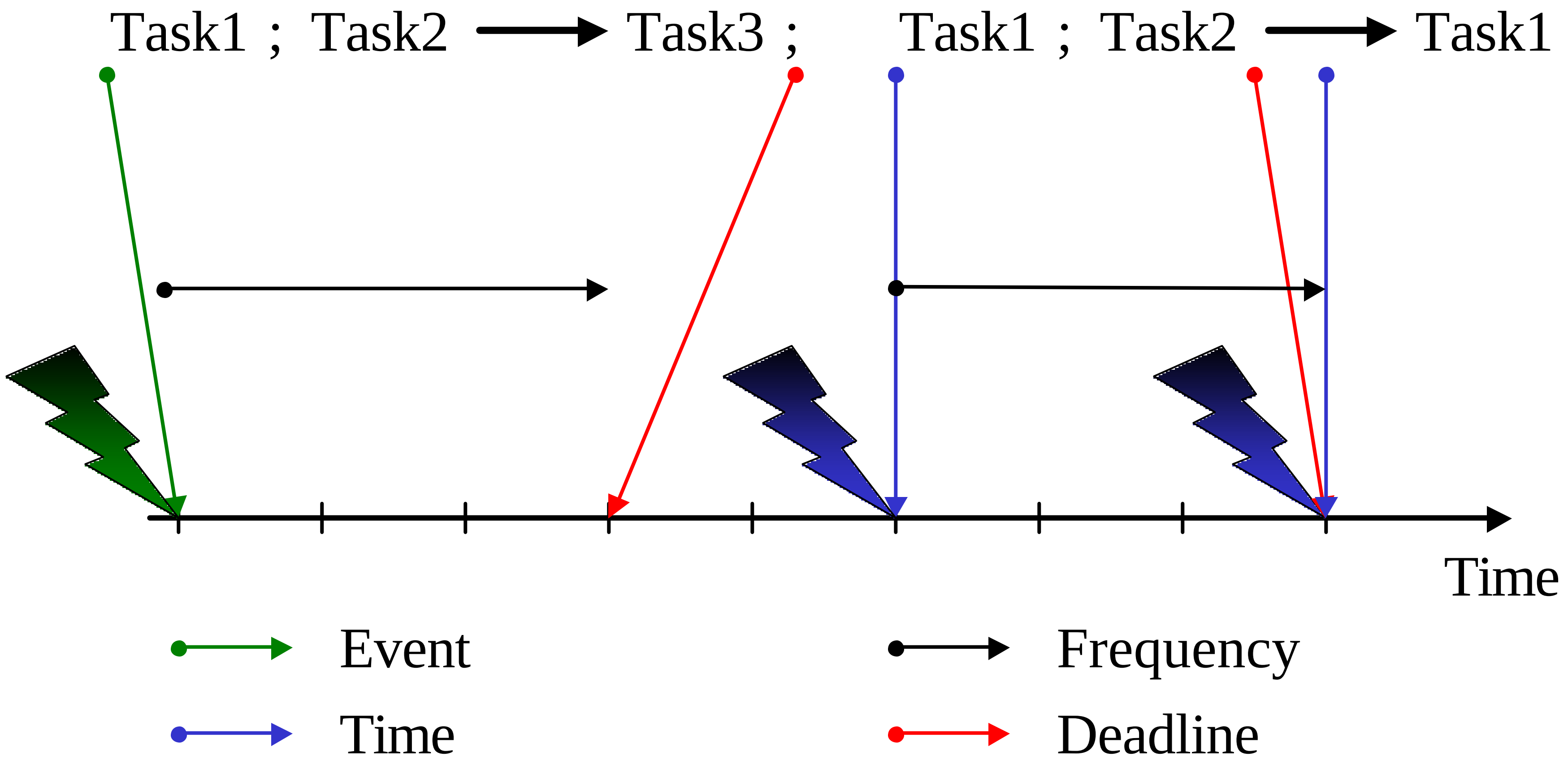
$\text{Task1} \longleftrightarrow \text{Task2}$

$\Downarrow$

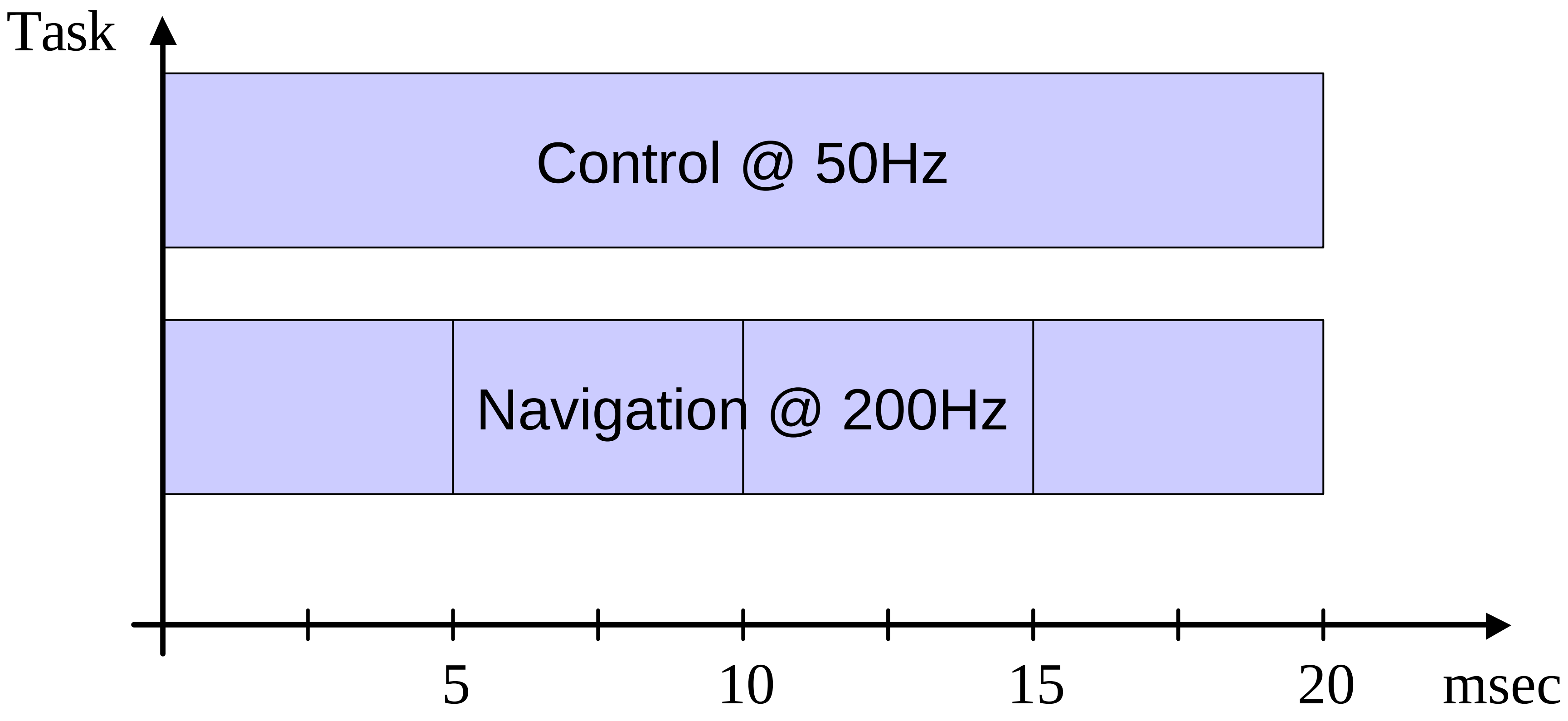
$\text{Task1} \longrightarrow \text{Task2}$

$\text{Task2} \longrightarrow \text{Task1}$

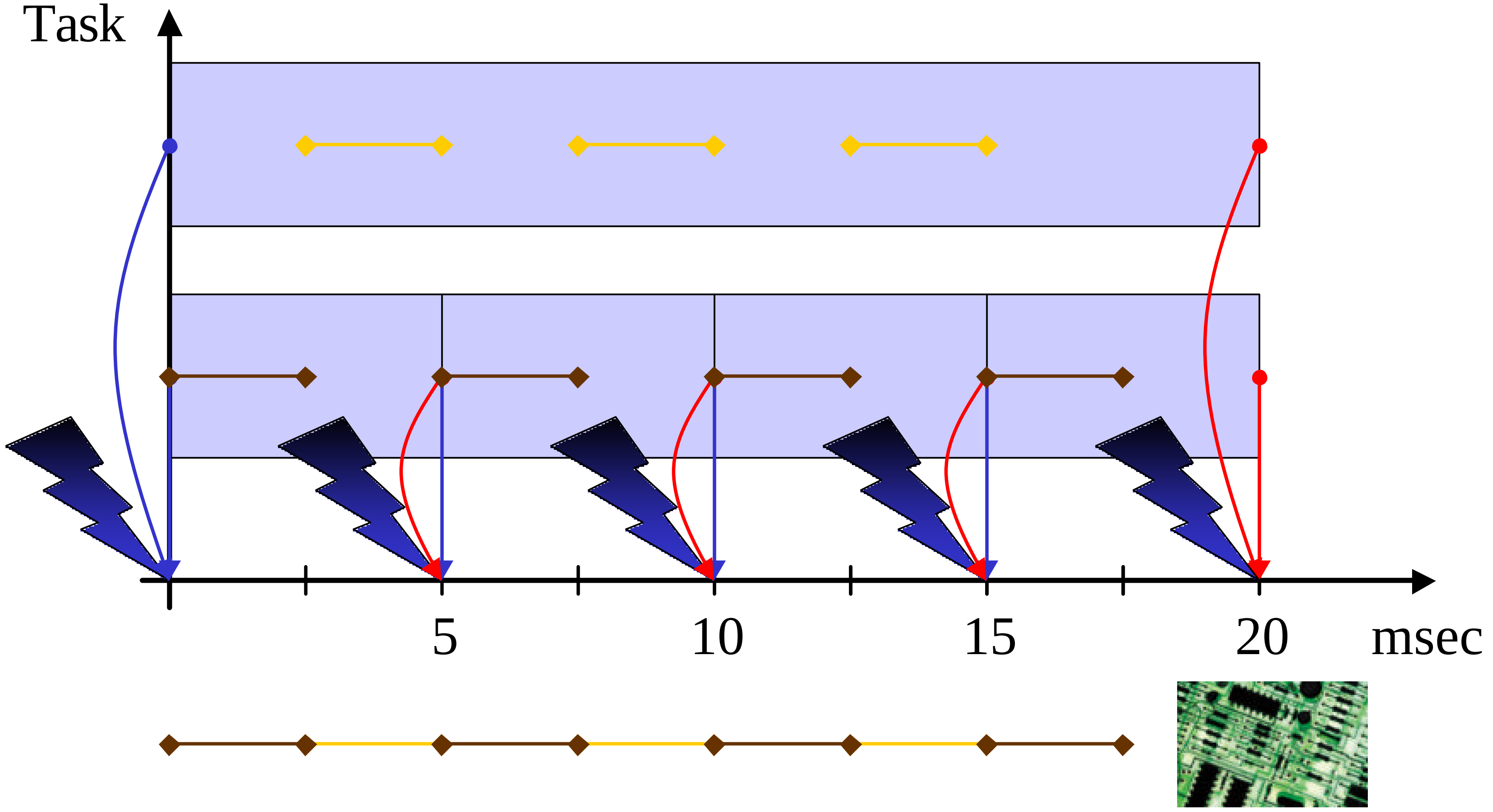
Real-Time


Task1 ; Task2
Task3 ;
Task1 ; Task2
Task1
Time
Event
Time
Frequency
Deadline

Helicopter Control



Code

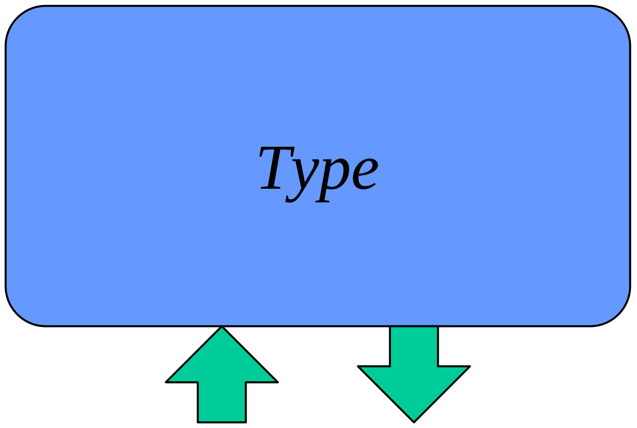


Embedded Programming

…requires the integration of:

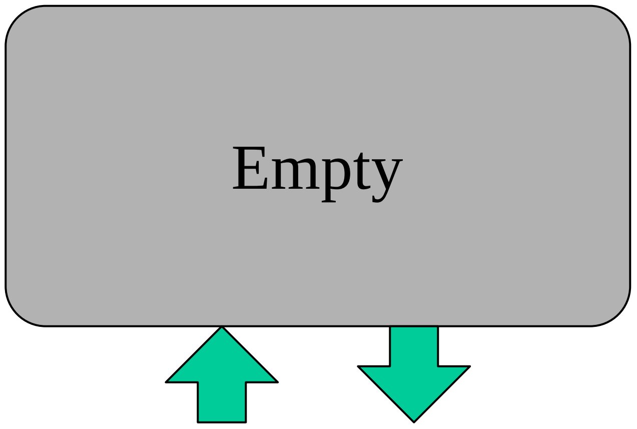
1. Real-time scheduling/communication concepts
2. Logical RTOS: The embedded machine
3. Programming language design
4. Compiler design
5. Classical software engineering techniques
6. Formal methods

Abstract Data Type



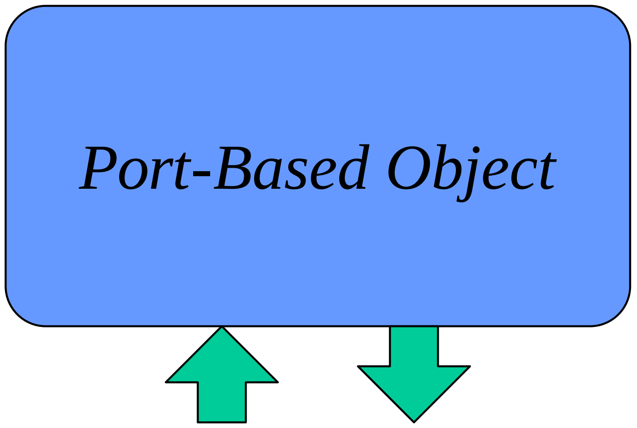
Interface: Set of methods

Abstract Interface



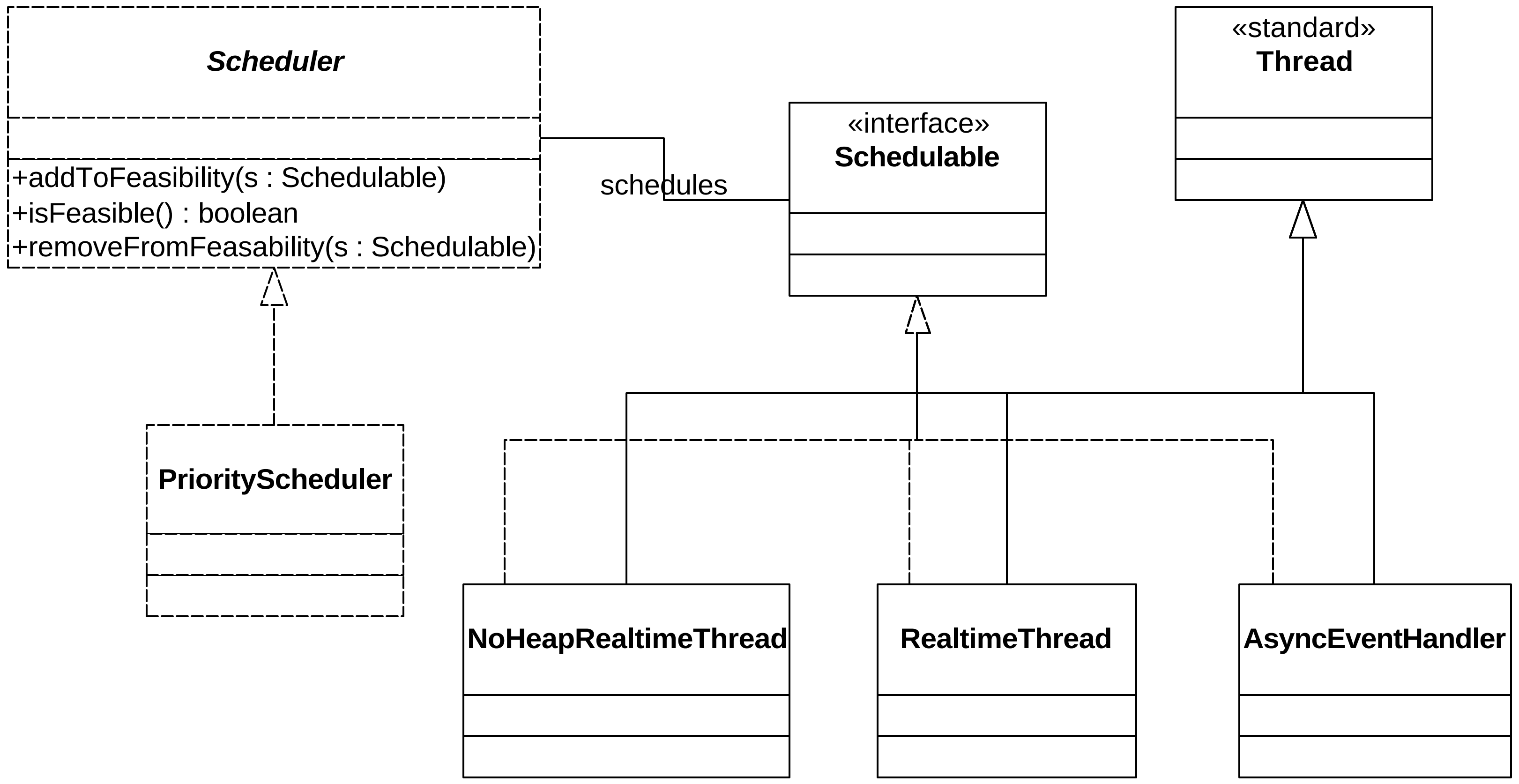
Interface: Set of methods

Object-Based vs. Object-Oriented



Interface: Ports + Control Methods

Steve: Real-Time Java

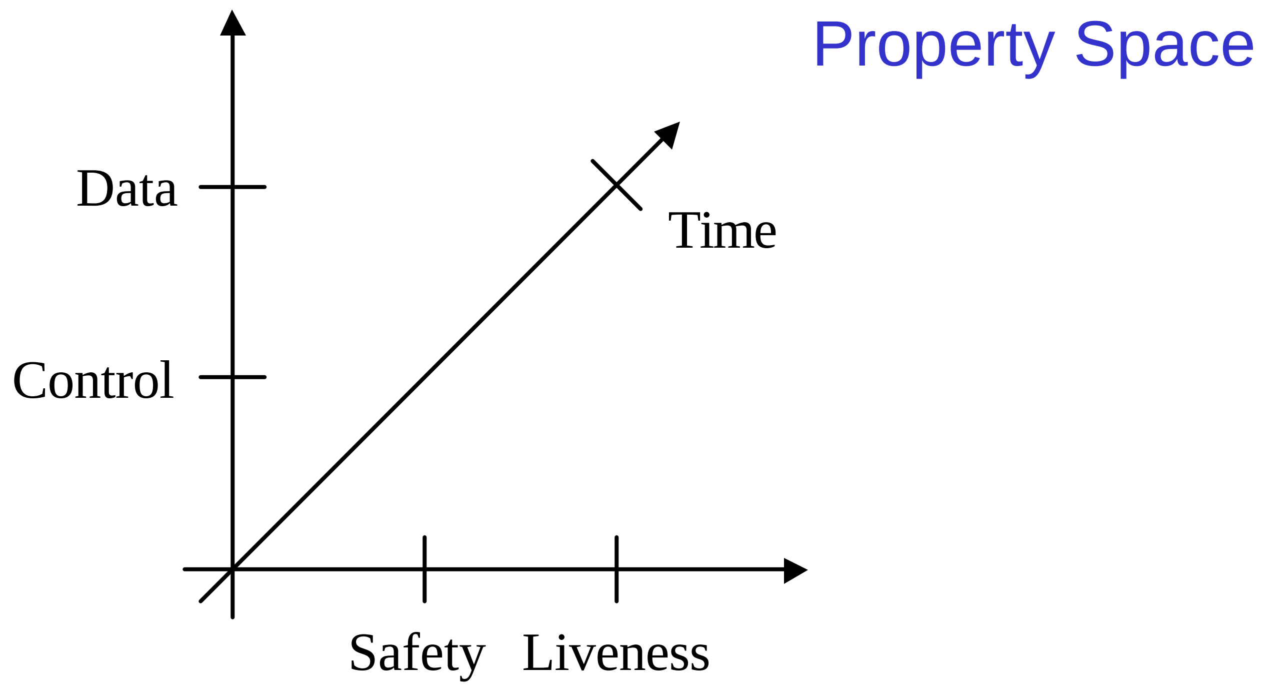


Embedded Programming

…requires the integration of:

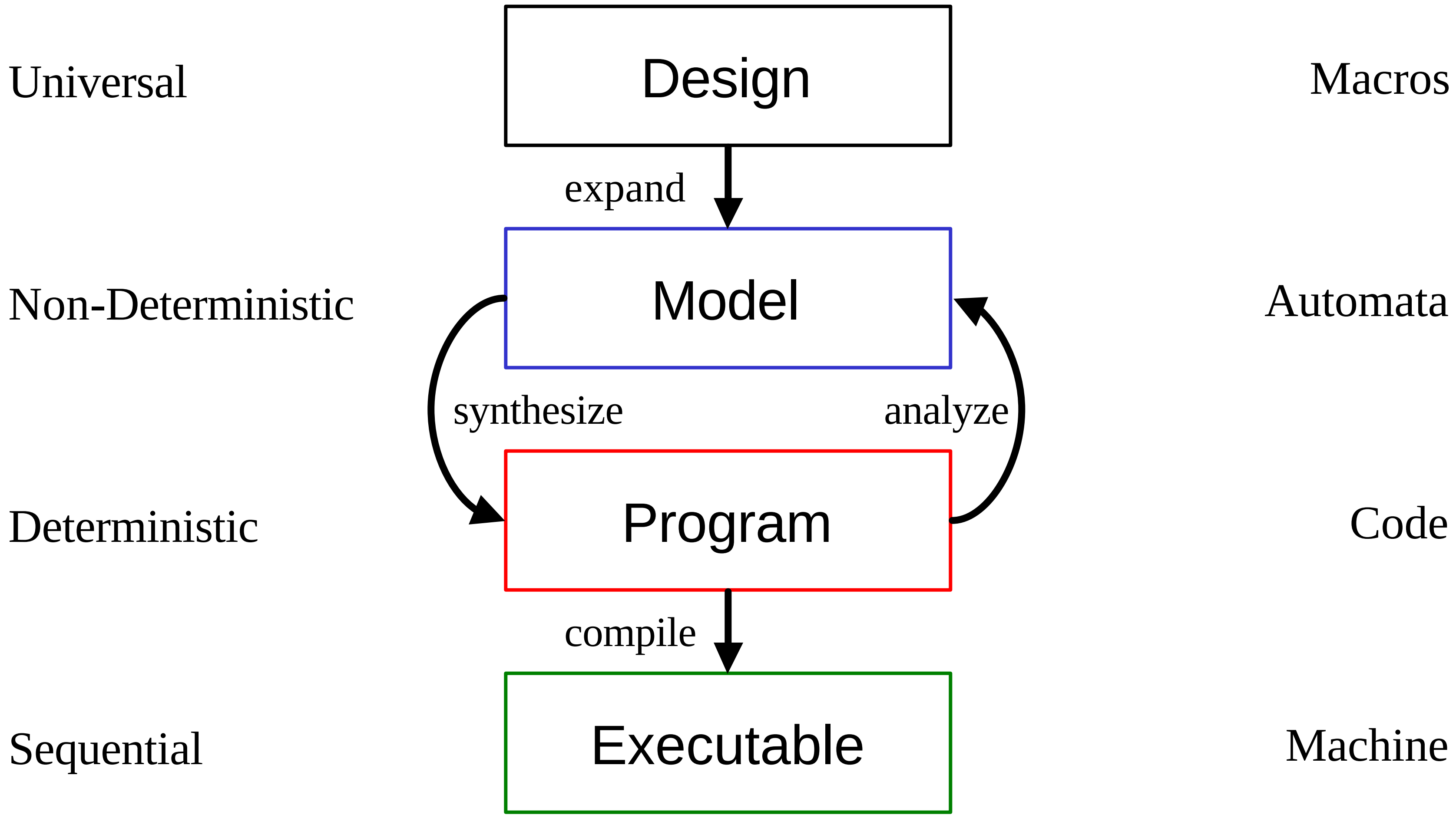
1. Real-time scheduling/communication concepts
2. Logical RTOS: The embedded machine
3. Programming language design
4. Compiler design
5. Classical software engineering techniques
6. Formal methods

Formal Verification

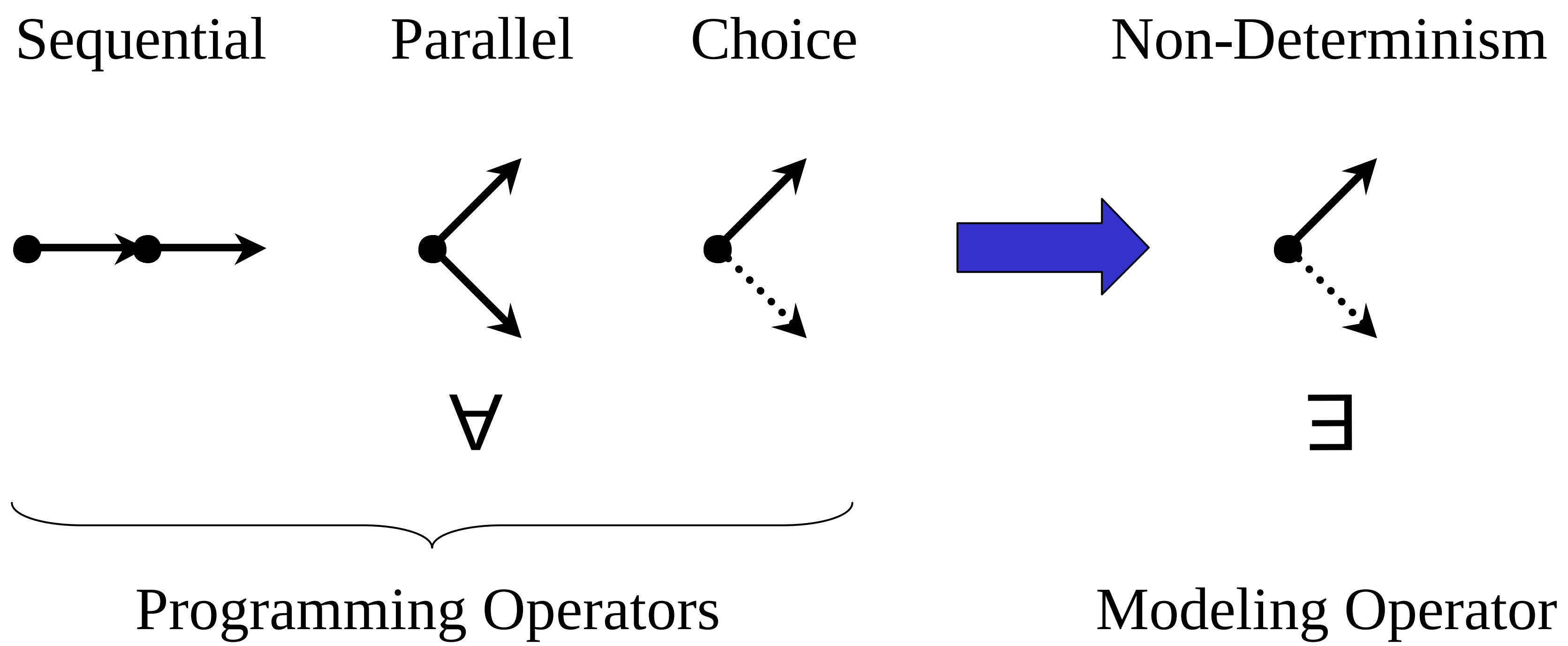


- Safety: Wrong things never happen!
- Liveness: Something useful will happen eventually!

Language Hierarchy



Non-Determinism



Helicopter…

Helicopter…

The Mindstorm Machine