Computational Systems Engineering

Christoph Kirsch
University of Salzburg

3 Unit Graduate Course, Winter 2004/2005
Chapter 1: Introduction

Organization

- Web: www.cs.uni-salzburg.at/~ck/teaching/CSE-Winter-2004
- Mailing list: cst-winter-2004@cs.uni-salzburg.at
- Administration: Petra.Kirchweger@cs.uni-salzburg.at
- Science: Christoph.Kirsch@cs.uni-salzburg.at

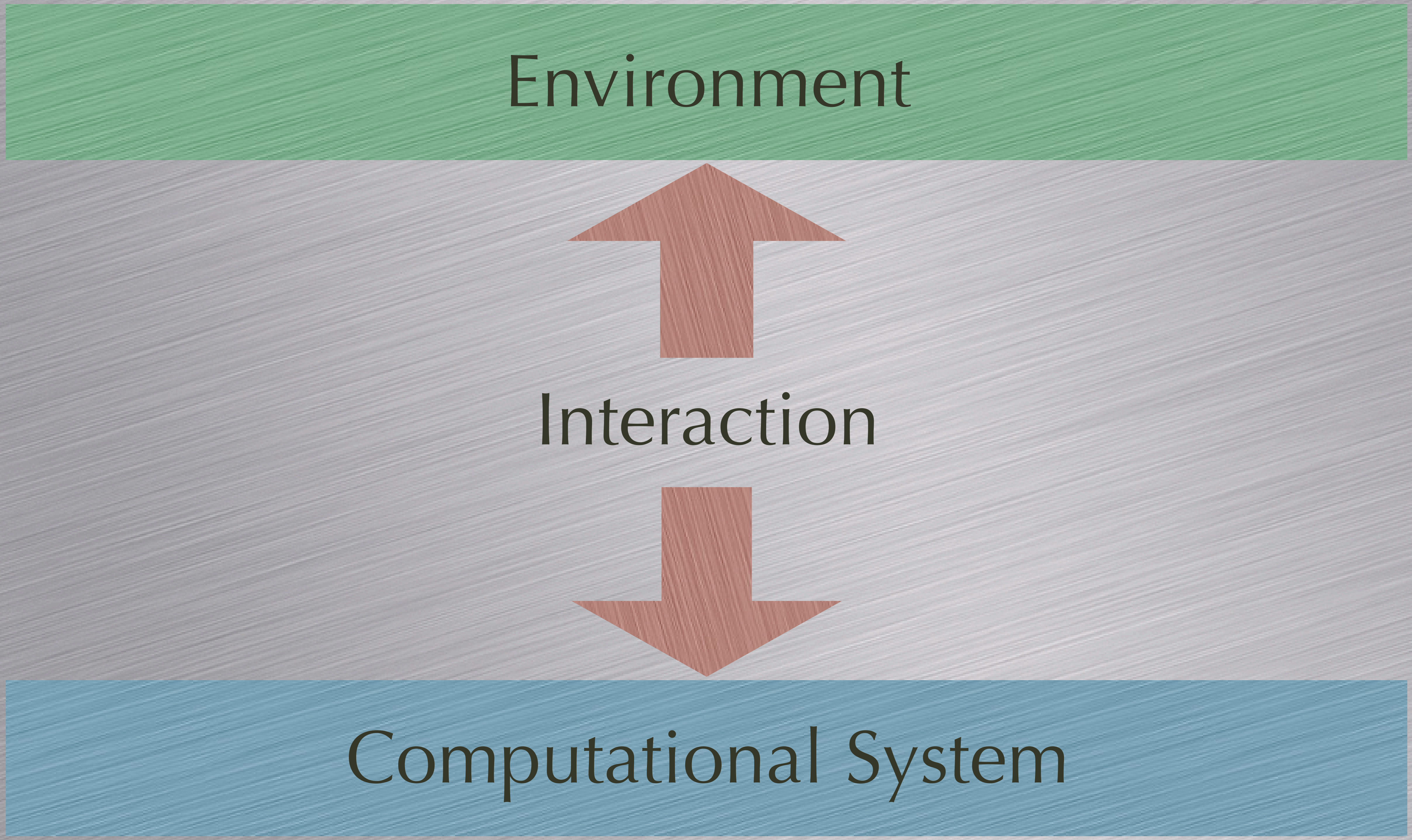
Assignments

- Paper readings: not more than once a week one paper, short 3-4 bullet summary due before next lecture
- Home work: occasional
- Project: form teams of 2-3 students, pick subject, design and implement, write project summary, and present at the end of the semester

Fun

- Shopping: search, compare, propose which hardware to buy
- Install OS and development tools
- Create user accounts, CVS repository, home page (sourceforge!?)
- Read and understand GPL (summary due before next lecture)

Environment vs. System

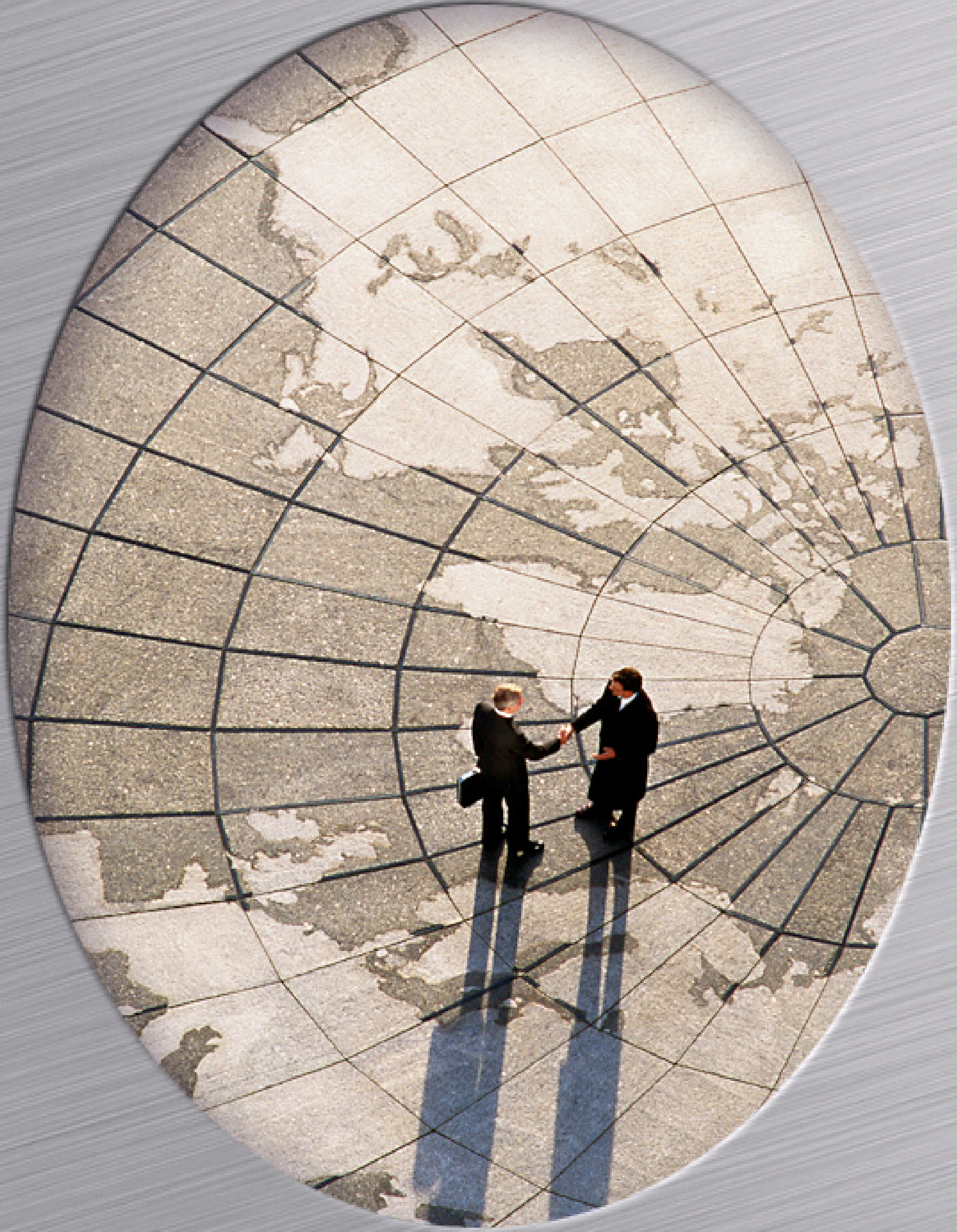


Environment

Interaction

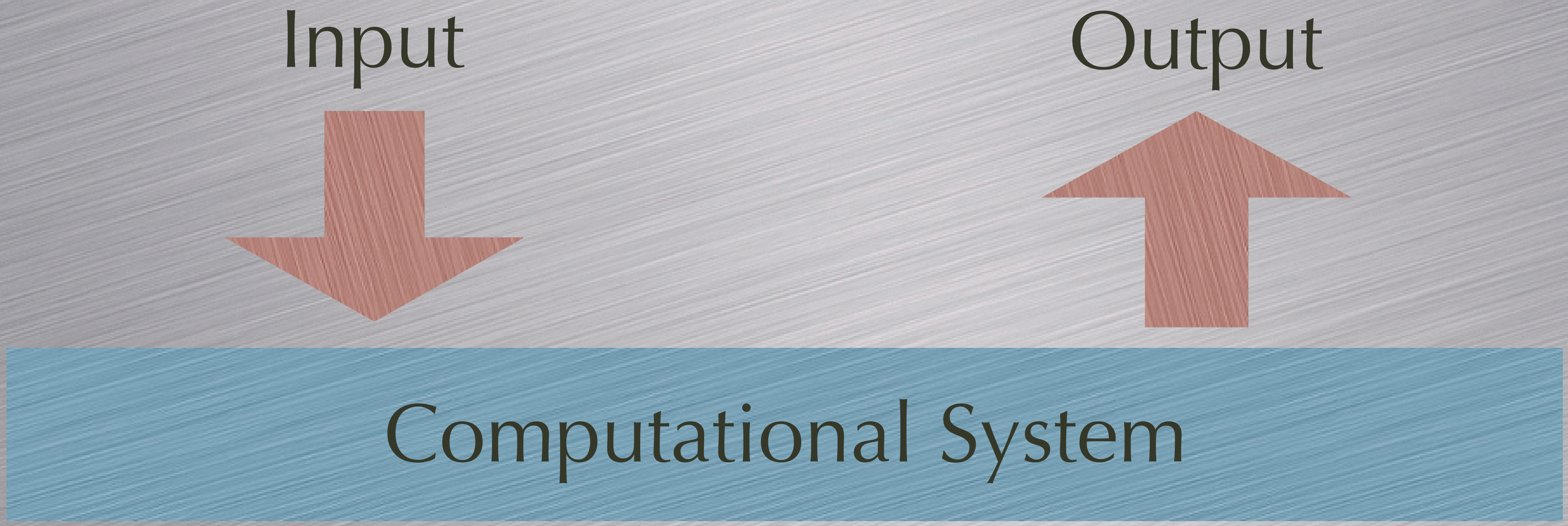
Computational System

Humans



- Humans interact with the physical world
- Humans interact with other humans
- A human is a computational system

Interaction and System

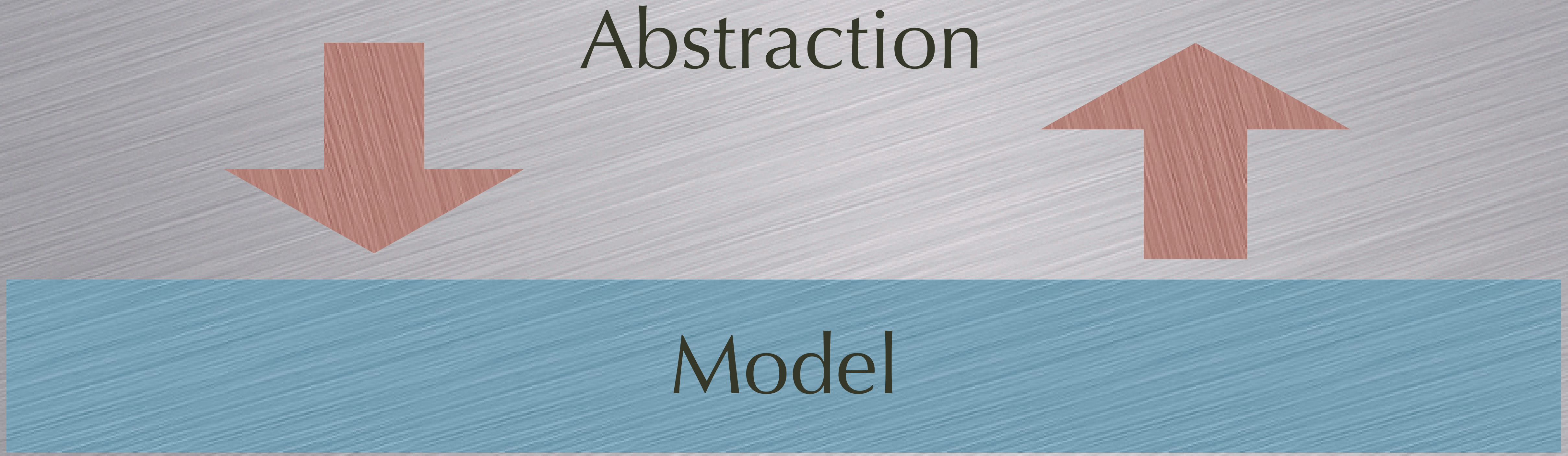


Input

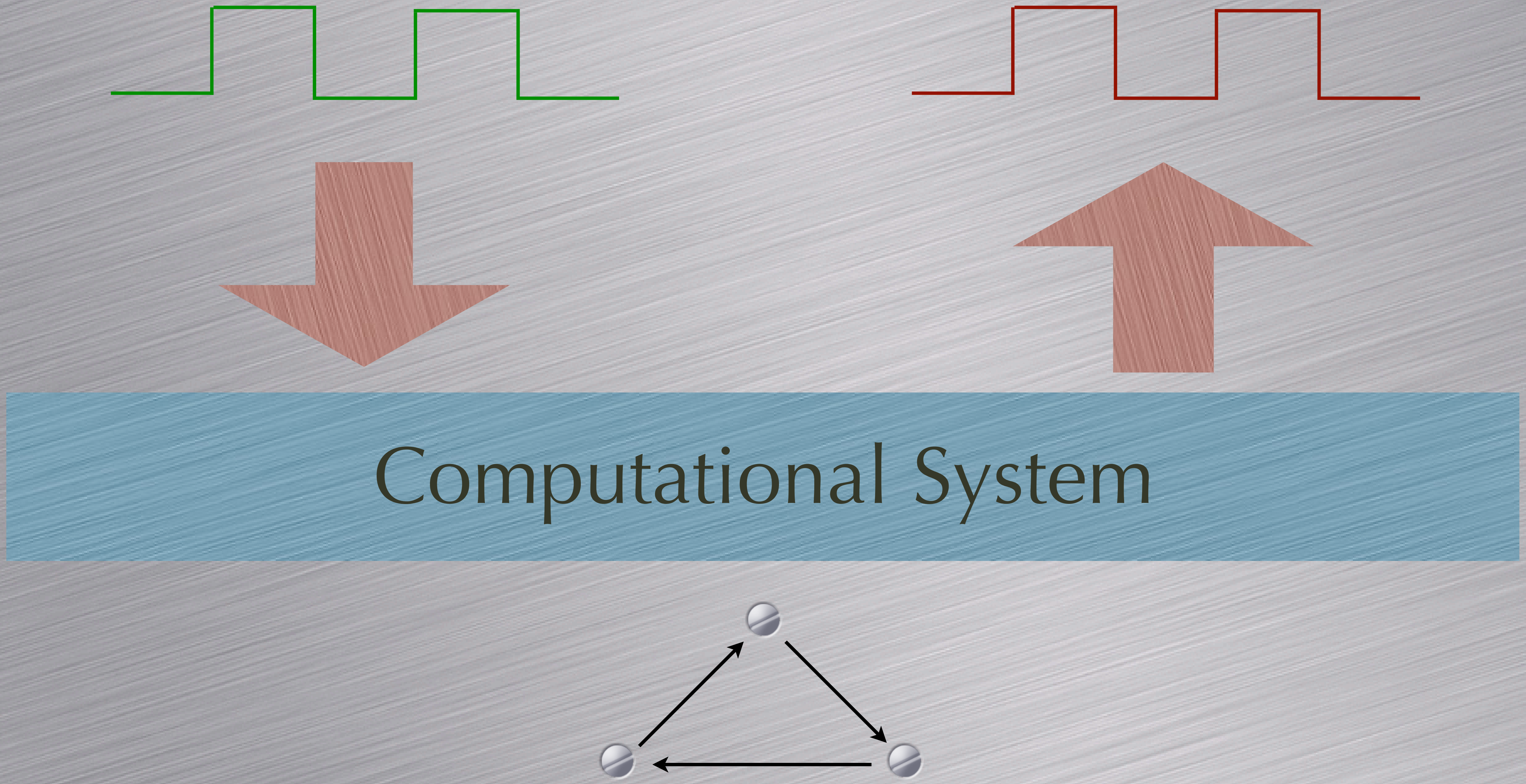
Output

Computational System

Model and Abstraction



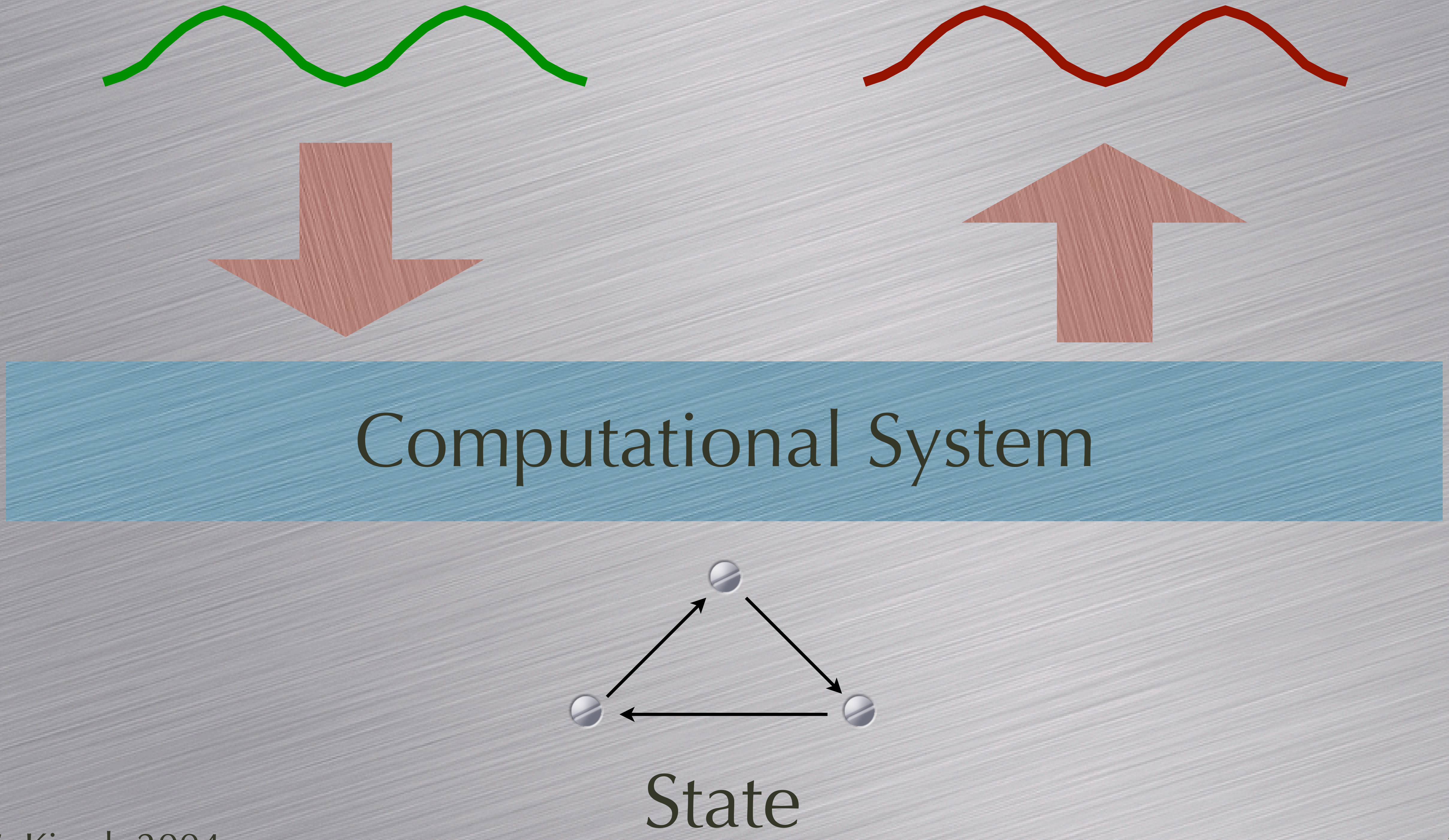
Behavior



Computational System

State

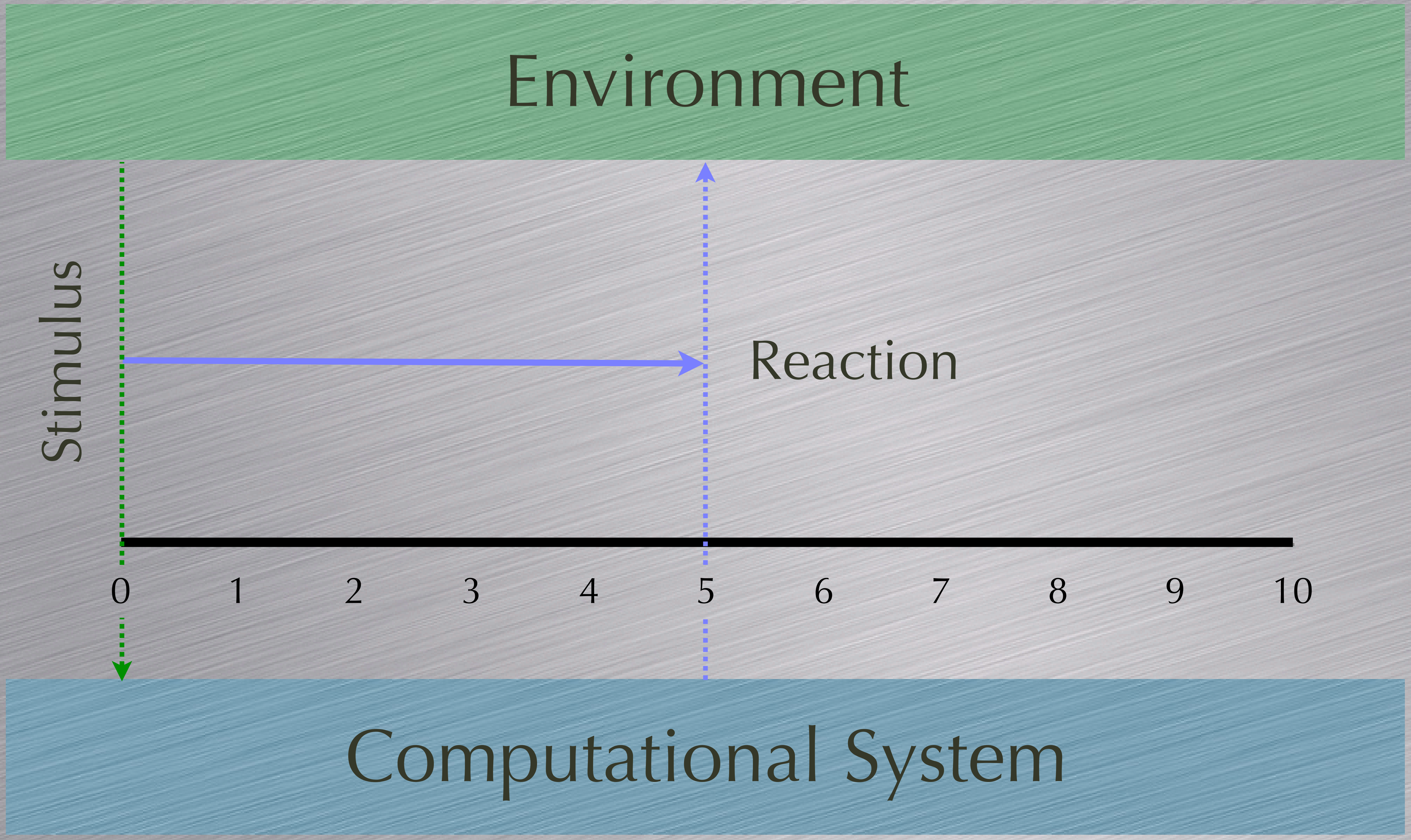
Speed



Computational System

State

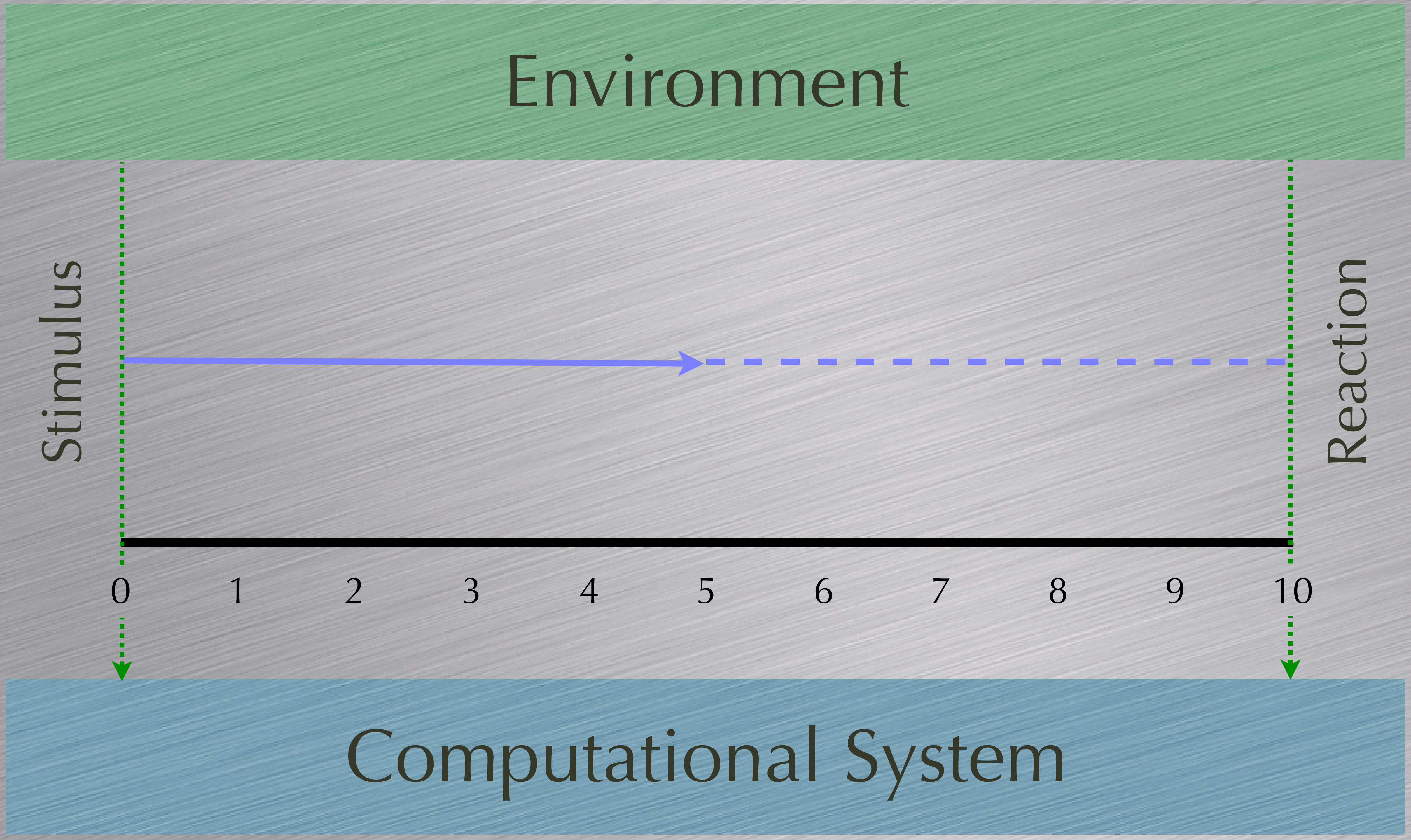
Interactive System

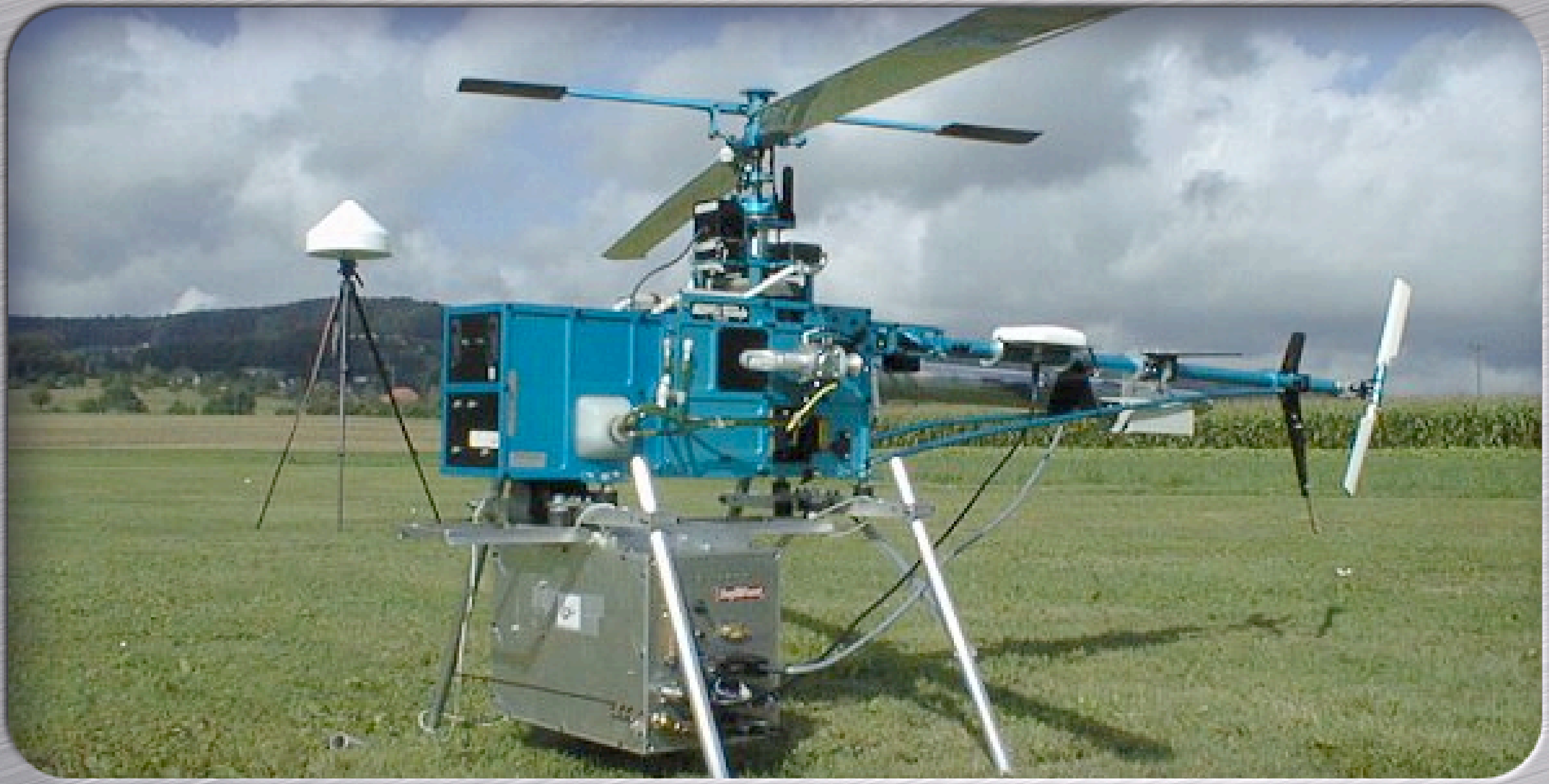


© C. Kirsch 2004

Desktop Computer

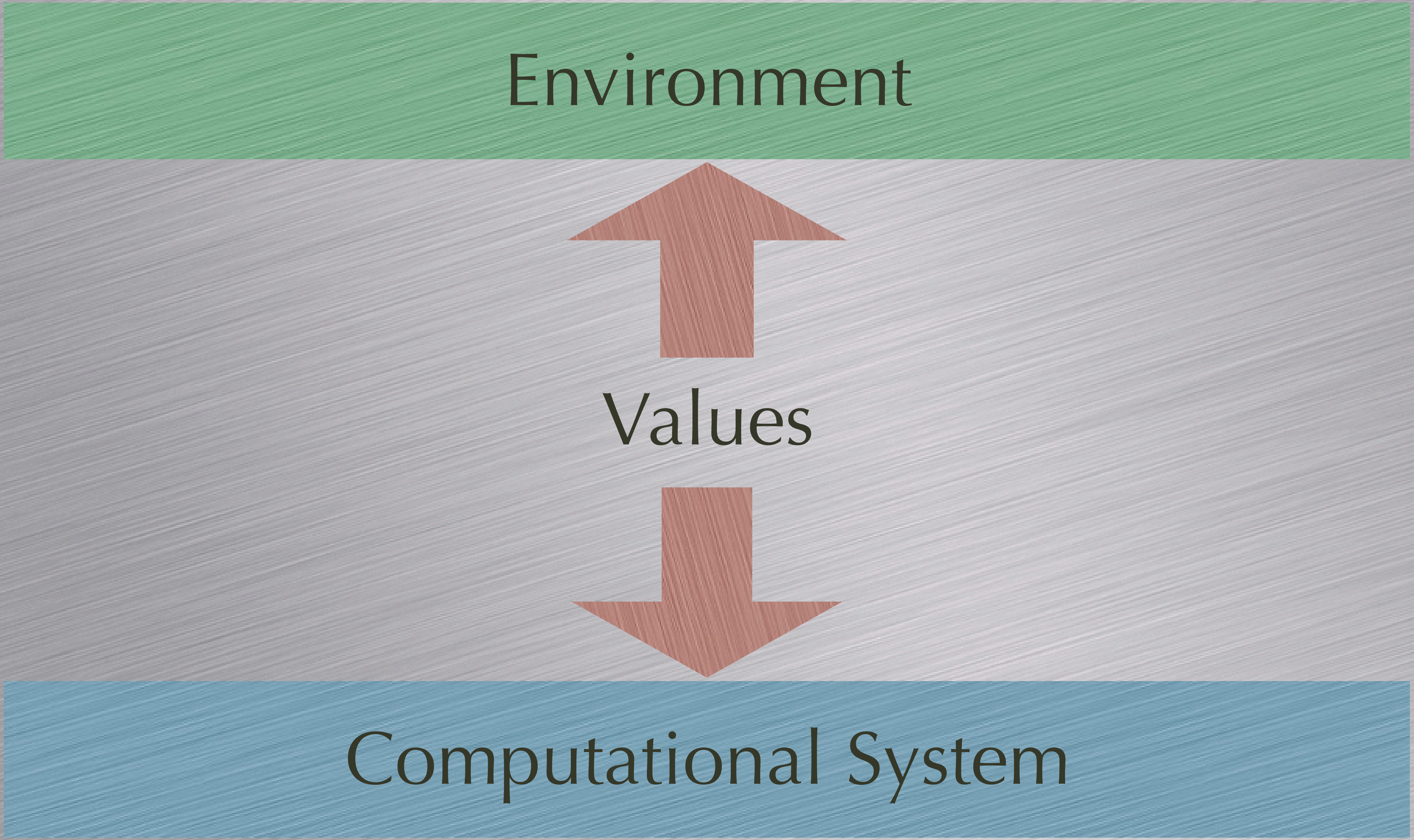
Reactive System





Control Computer

Data

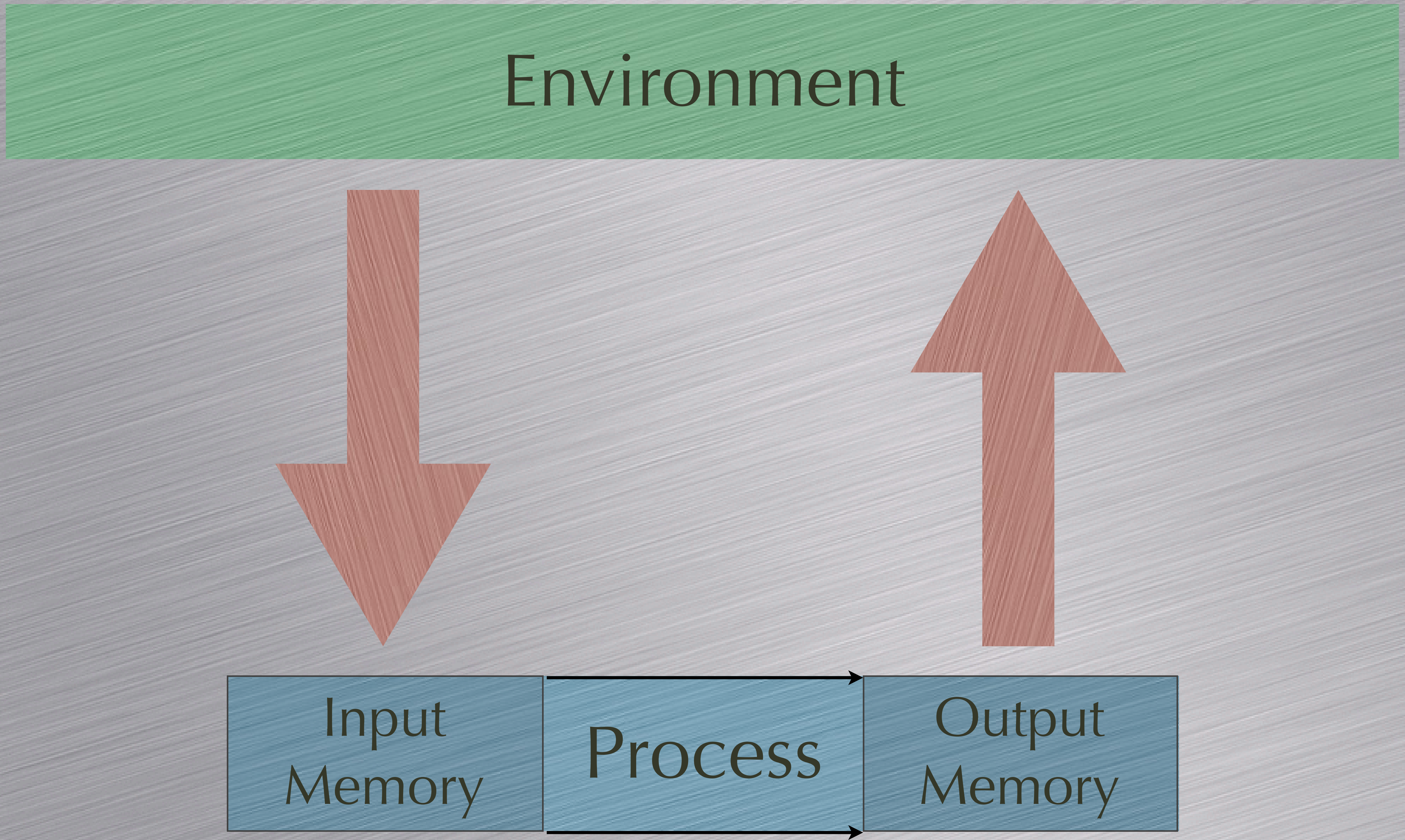


Environment

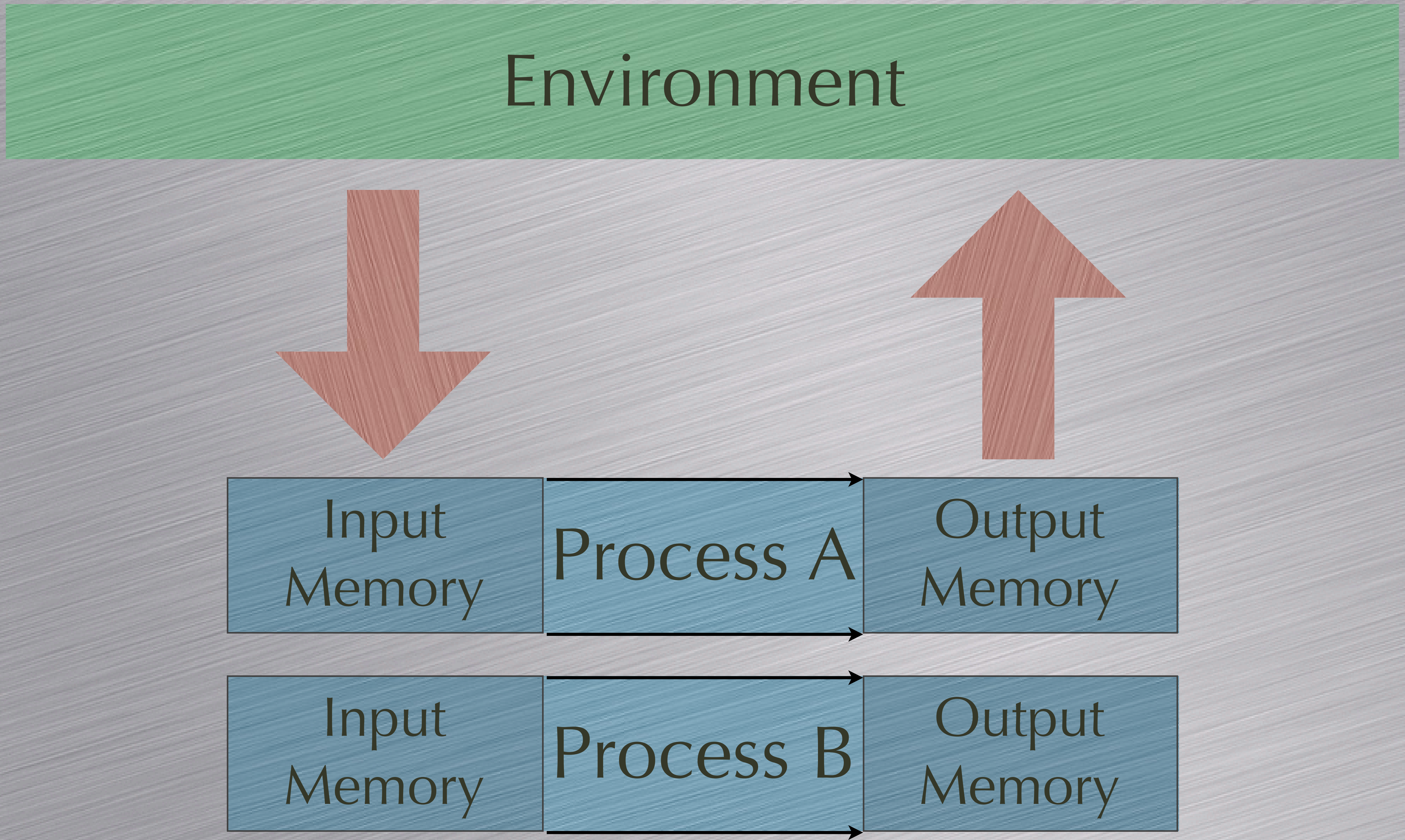
Values

Computational System

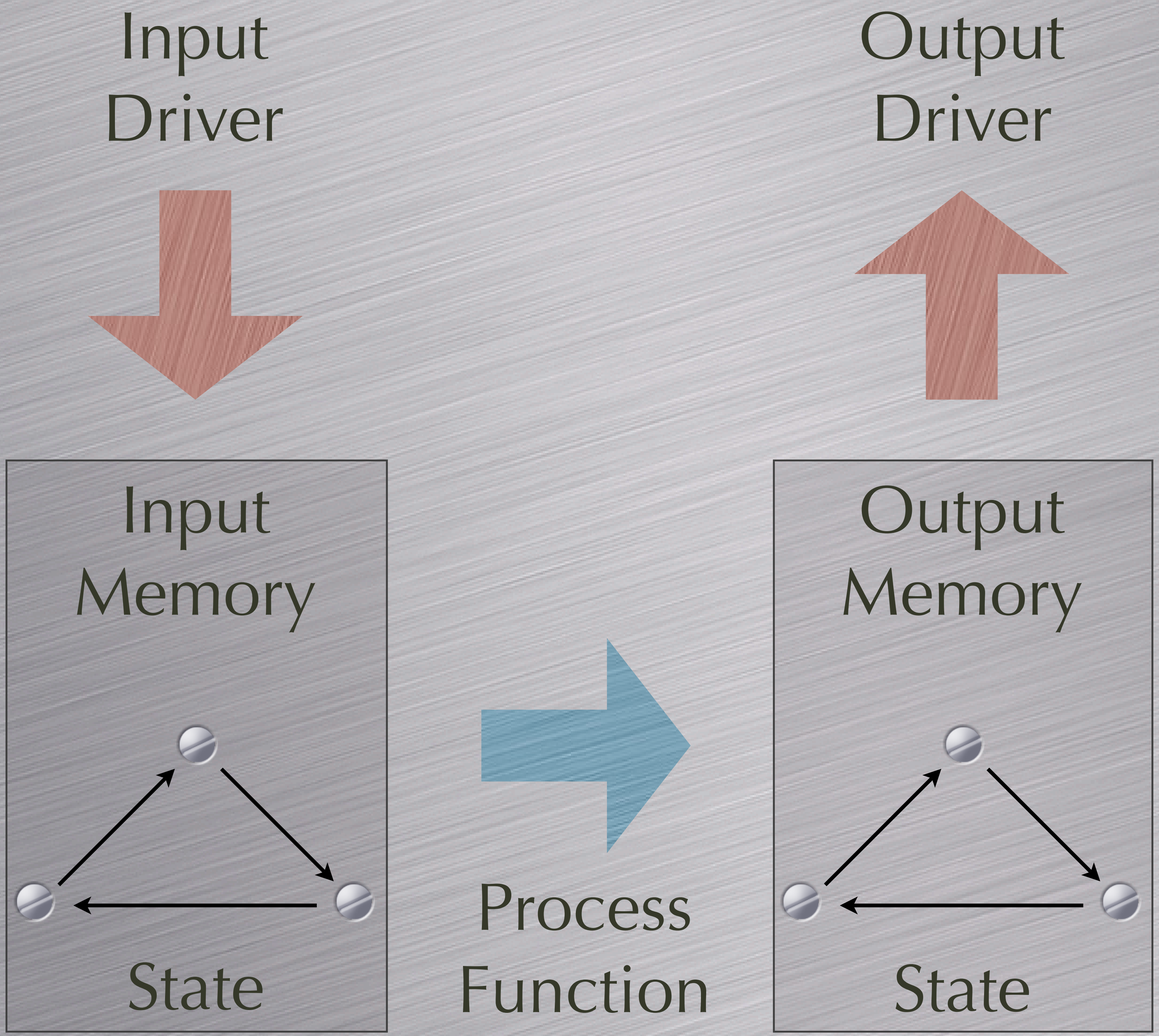
Memory



Concurrency



Process Structure



Input Driver

Output Driver

Input Memory

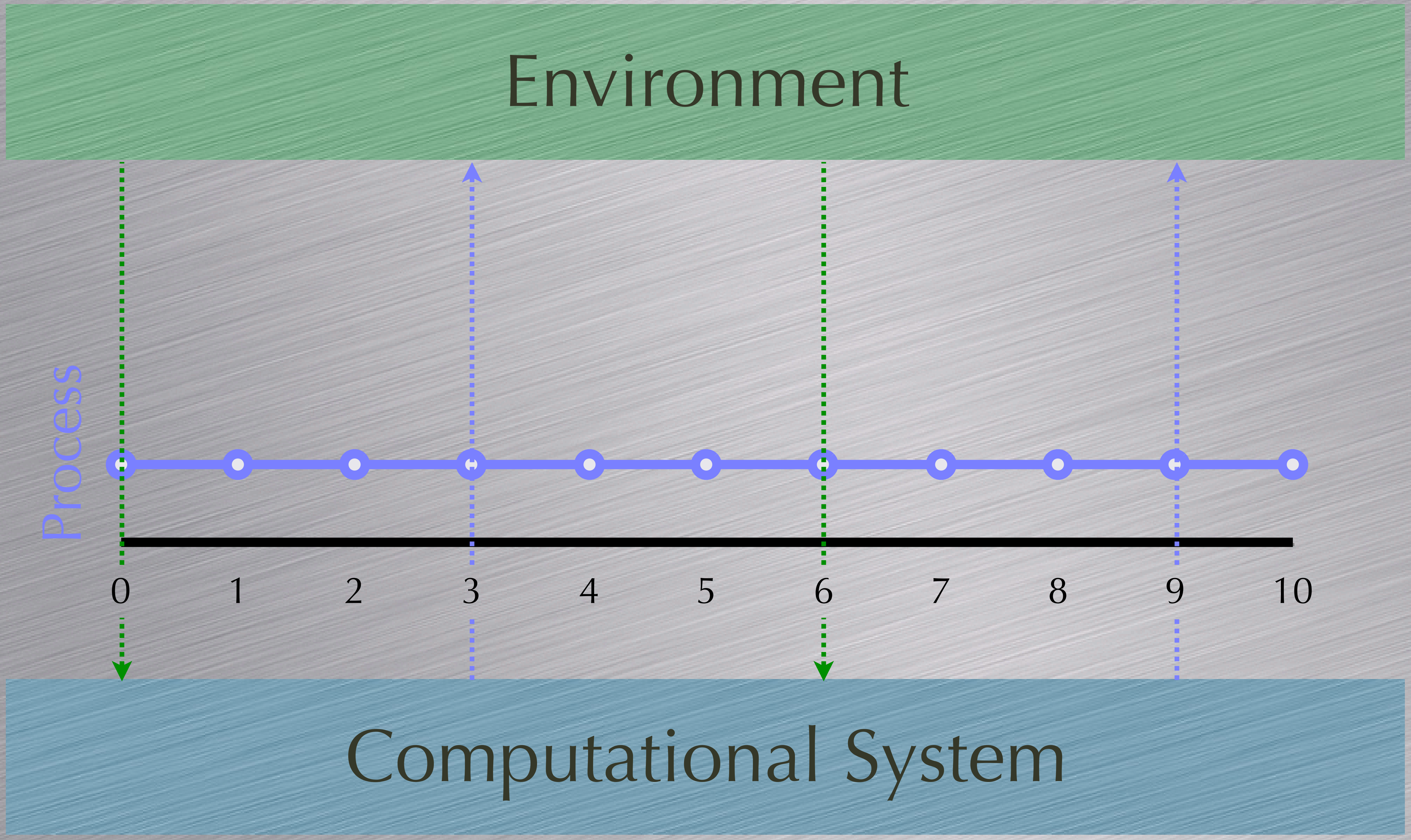
State

Process Function

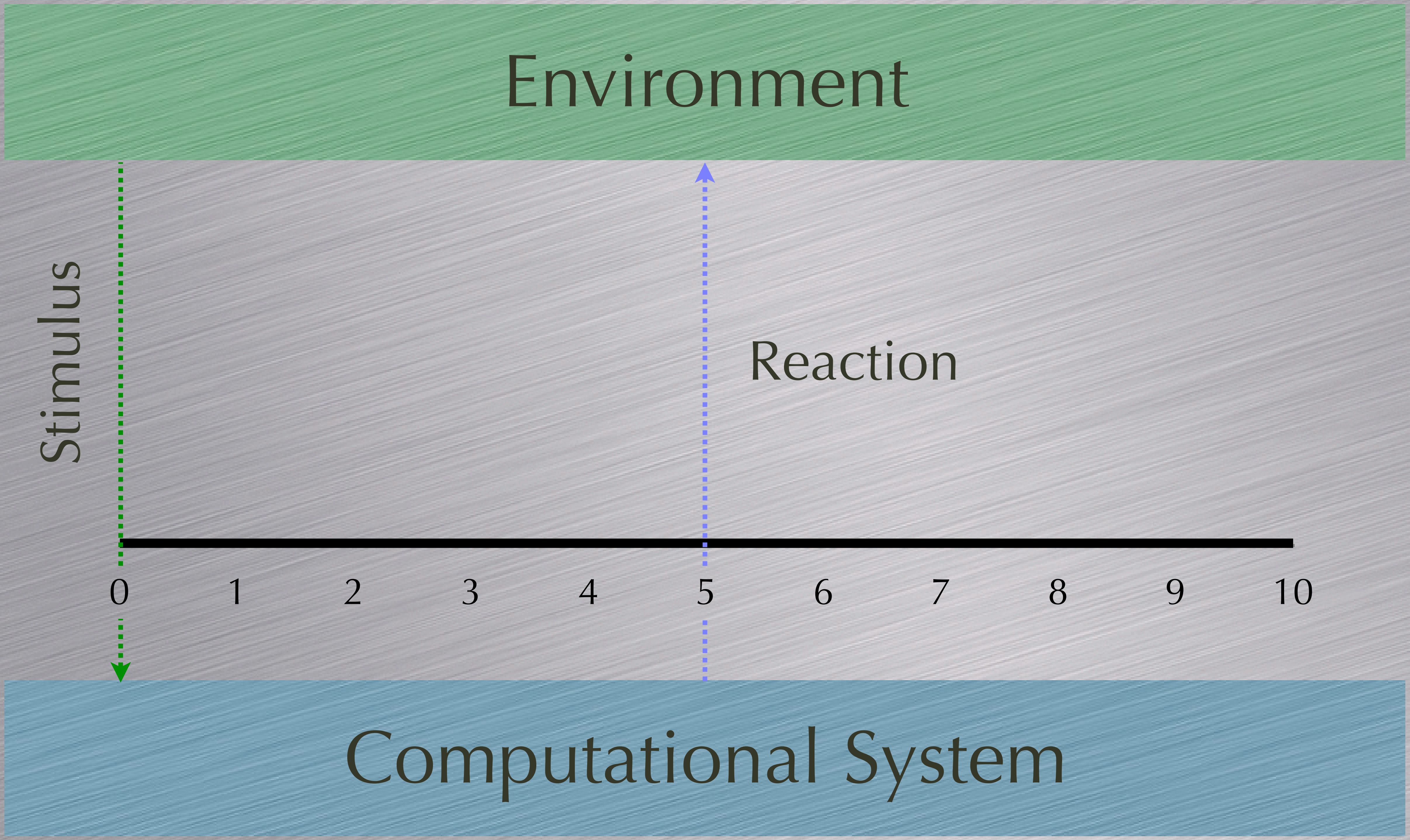
Output Memory

State

Process Behavior



Control



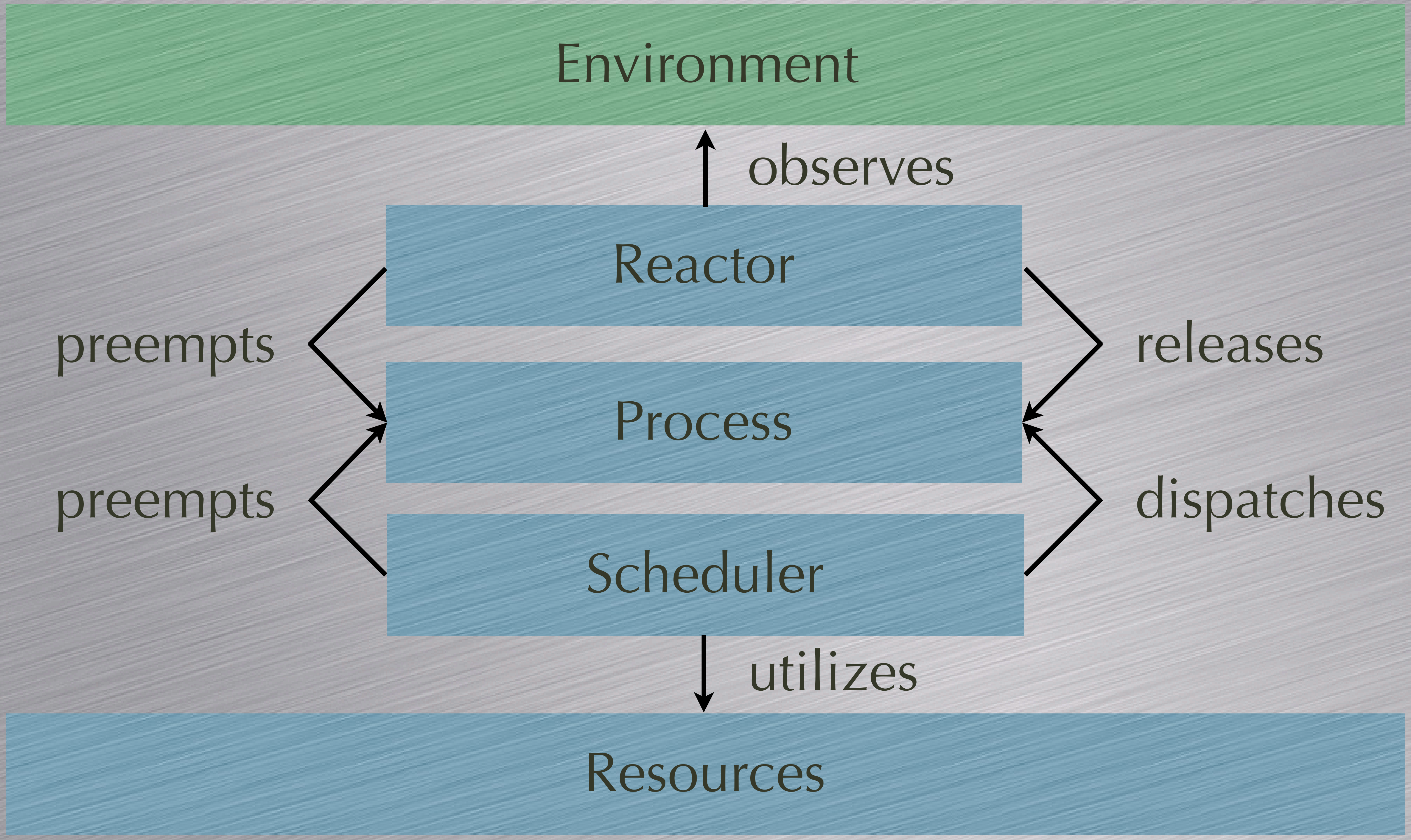
Environment

Stimulus

Reaction

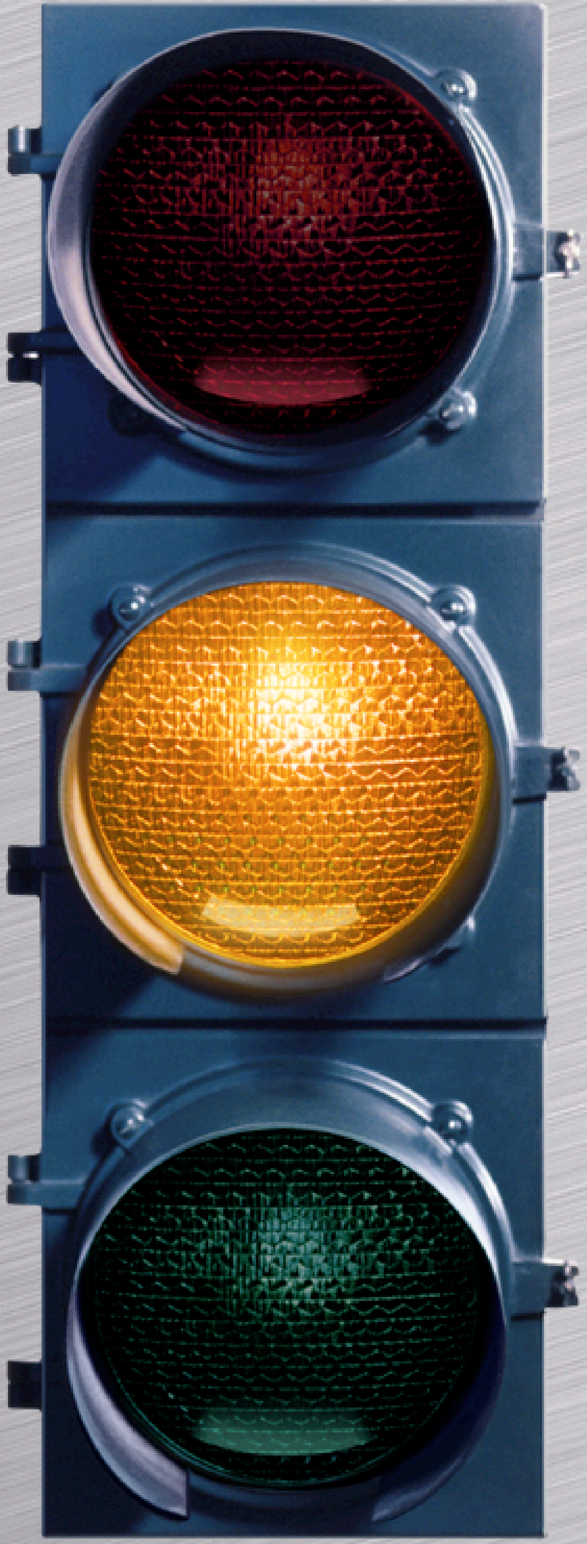
Computational System

System Structures



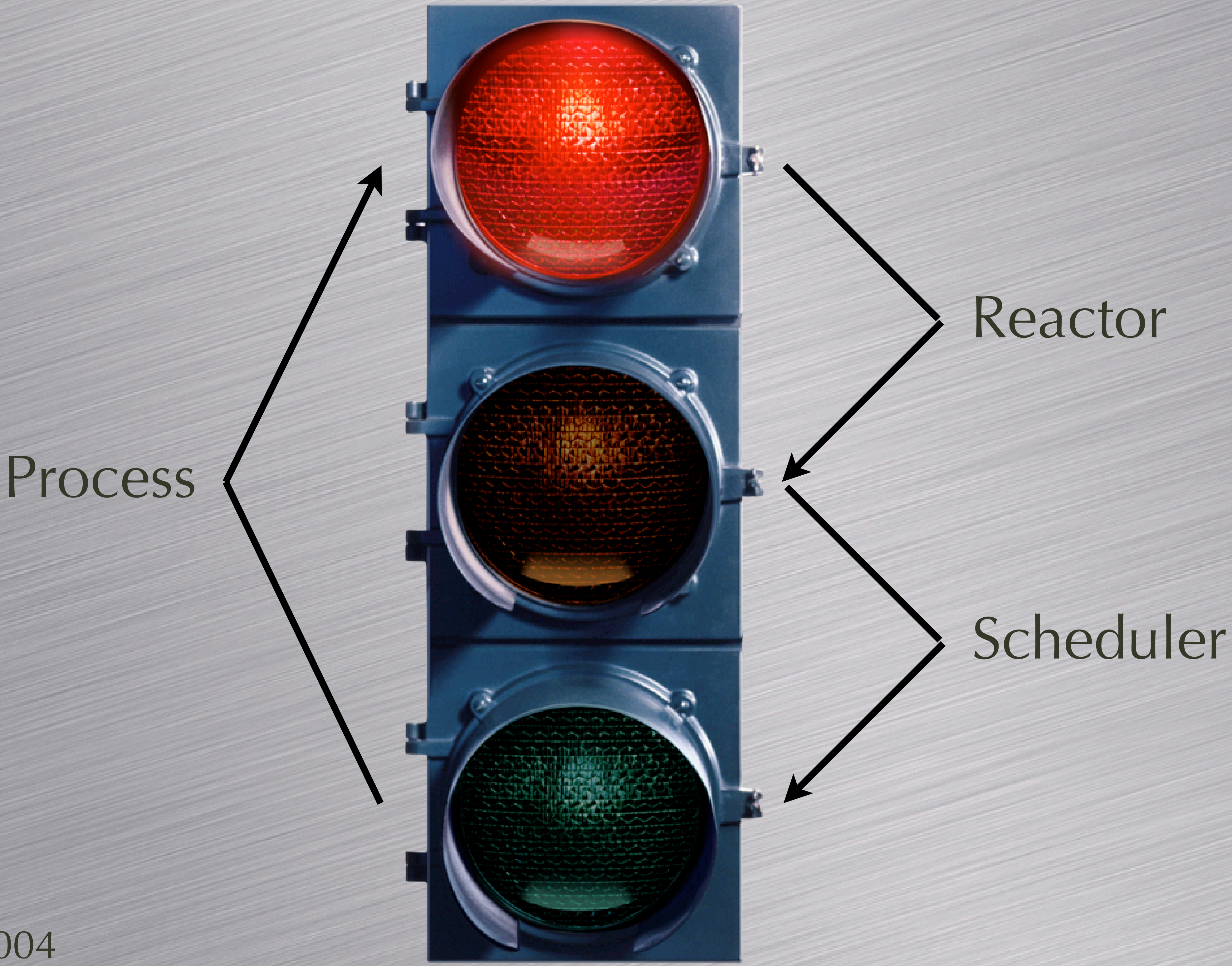
Blocked Process

Released Process

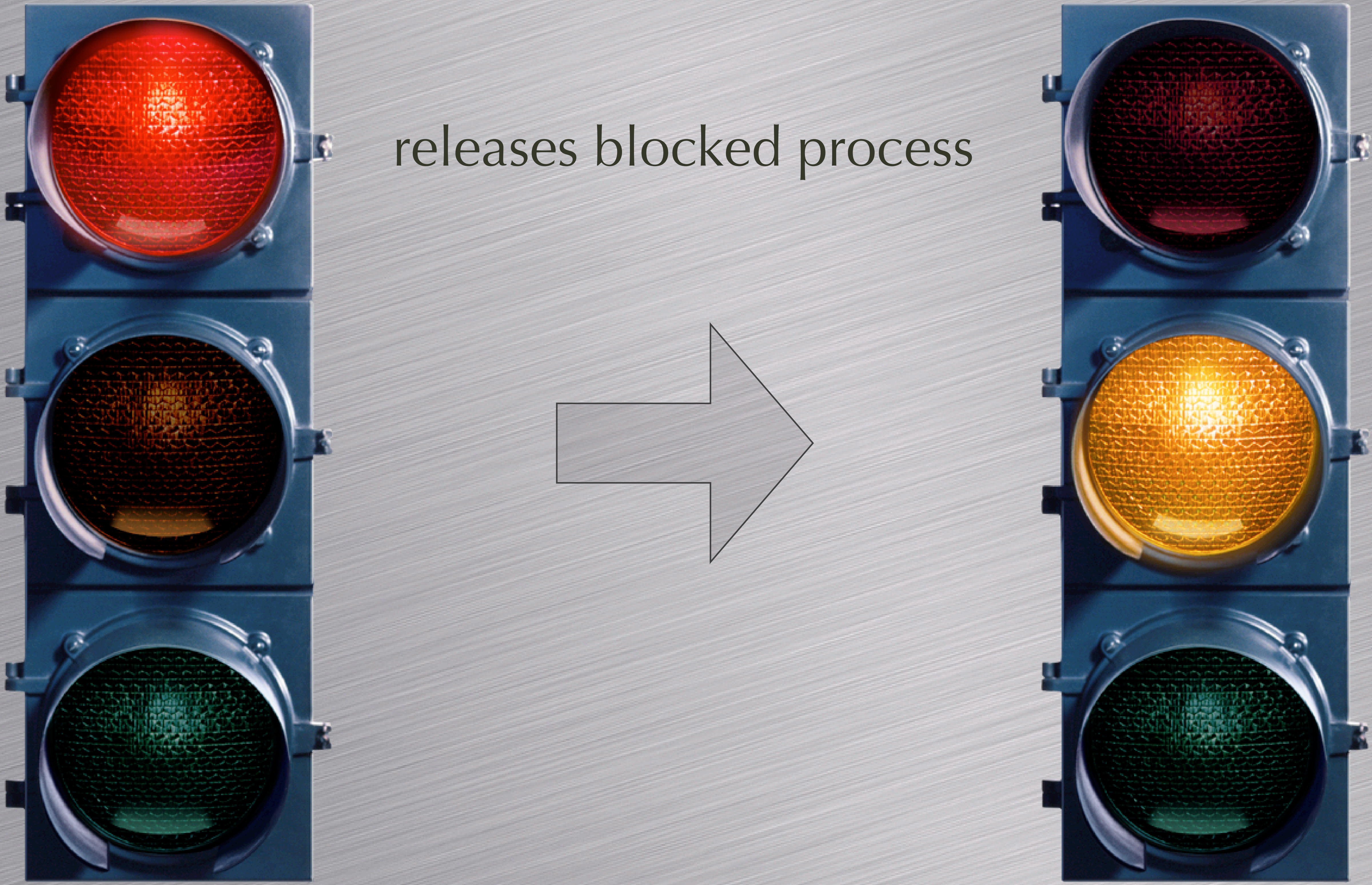


Running Process

State Transitions



Reactor

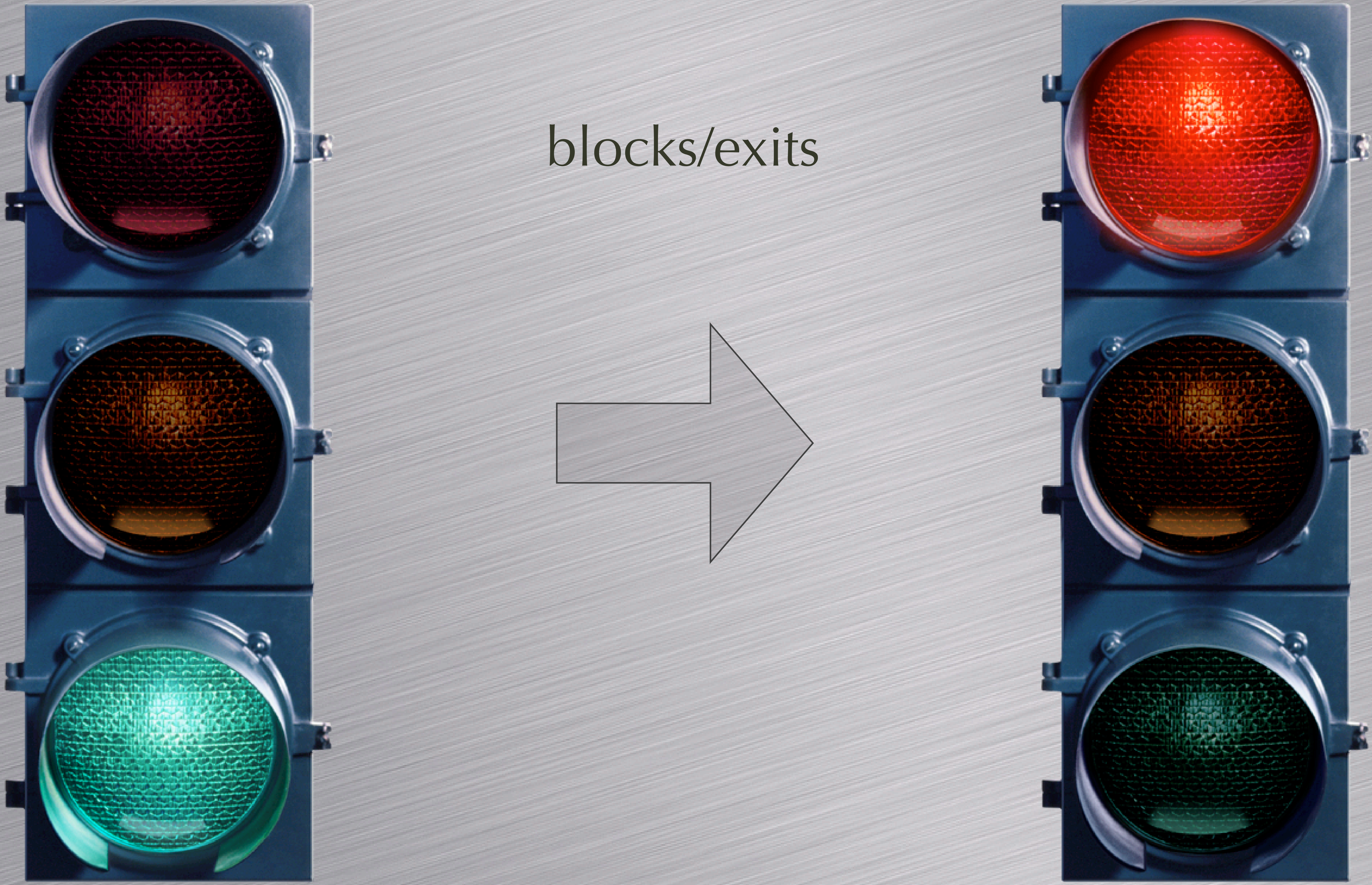


releases blocked process

Scheduler

runs released process

Process

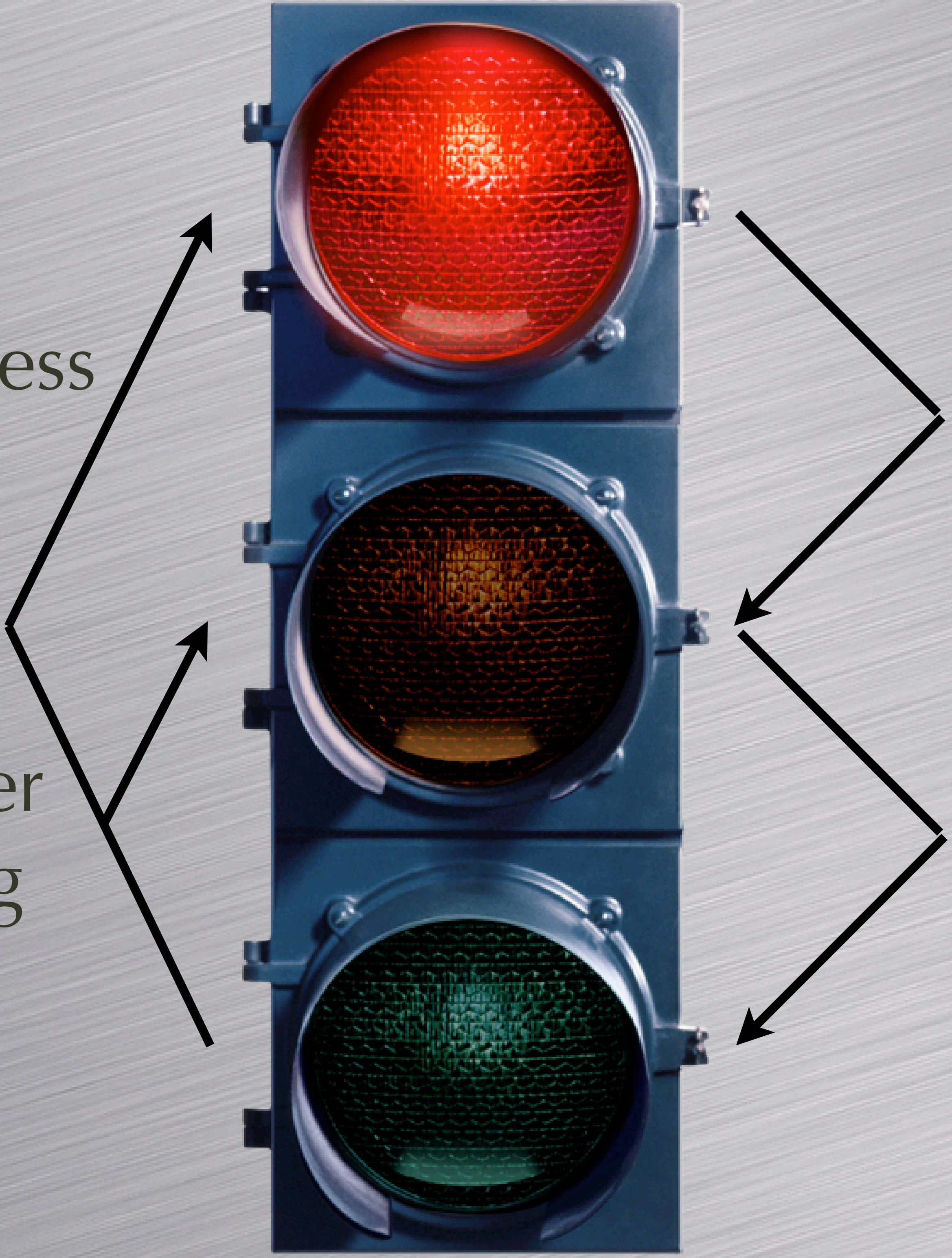


blocks/exits

Reactor/Scheduler

preempt running process

Transitions Revisited



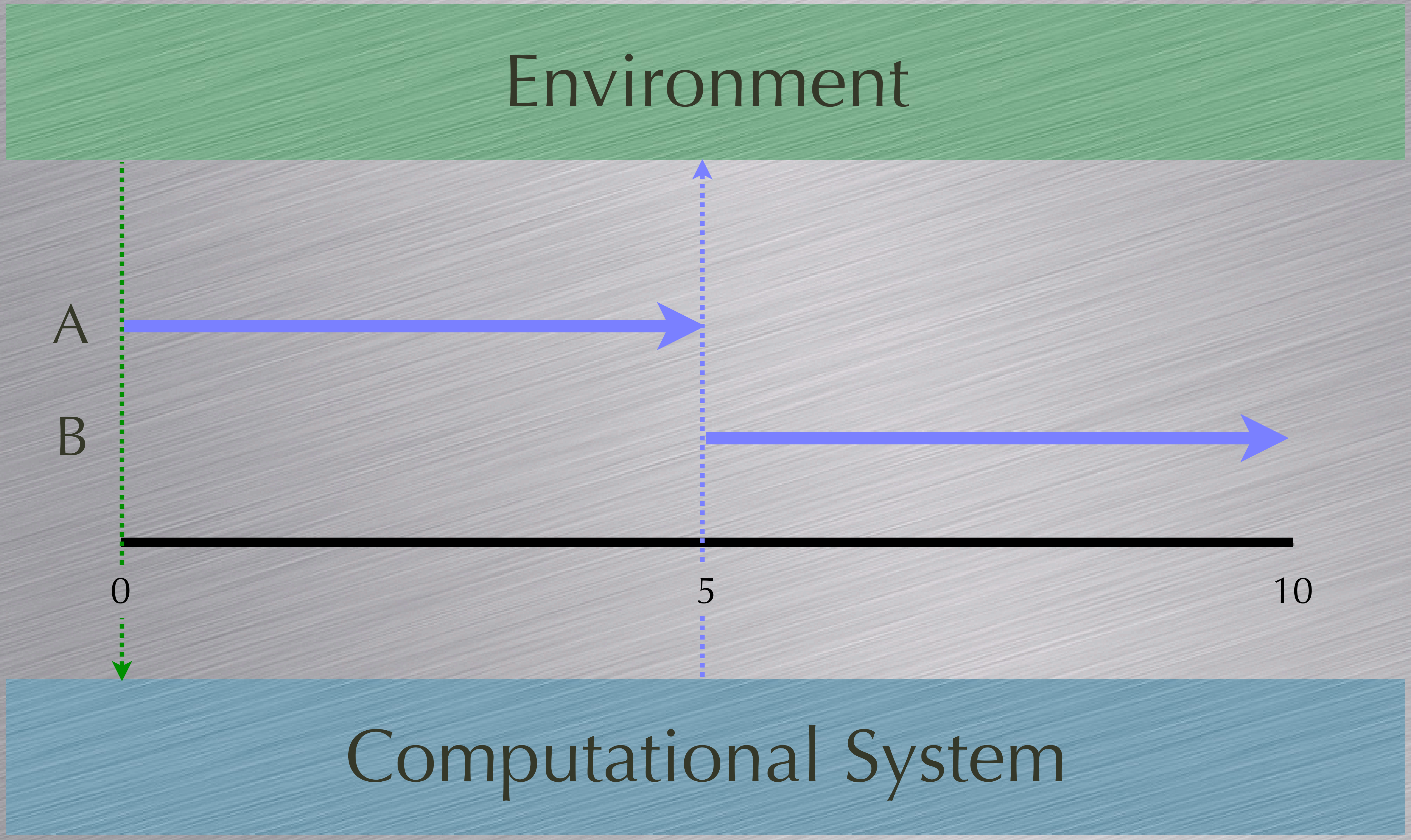
Running process blocks/exits

Reactor releases blocked process

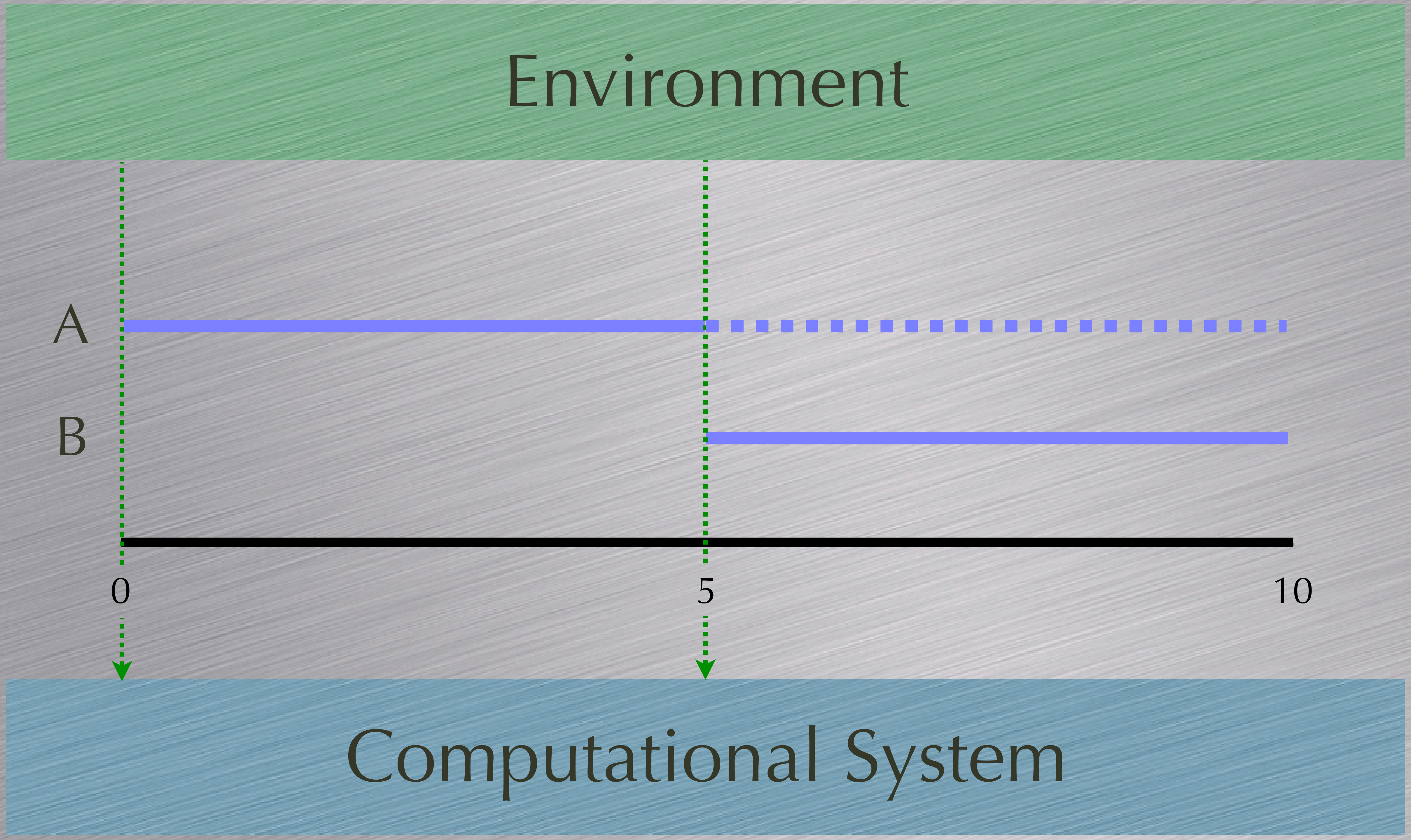
Reactor/Scheduler preempts running process

Scheduler runs released process

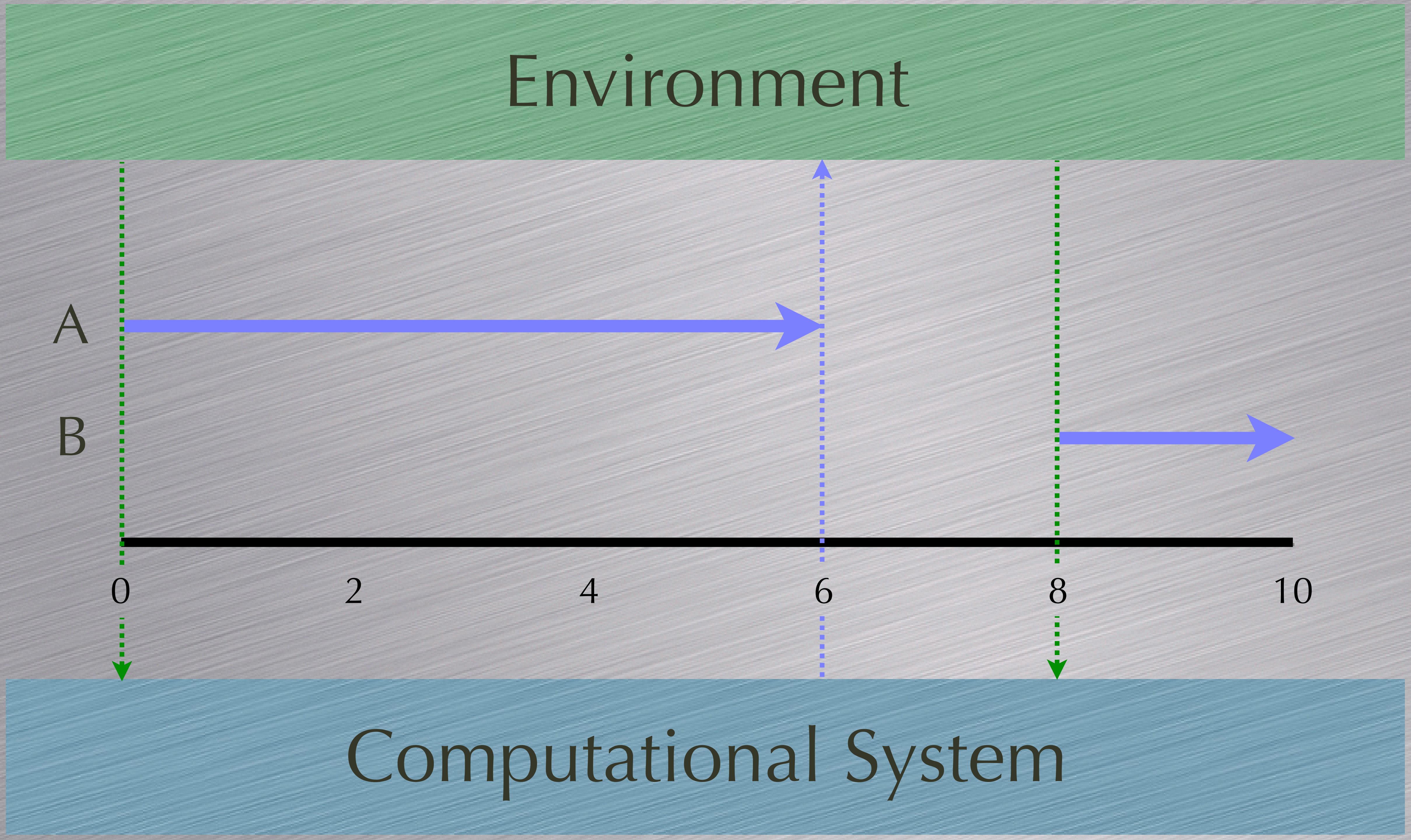
Cooperation



Preemption



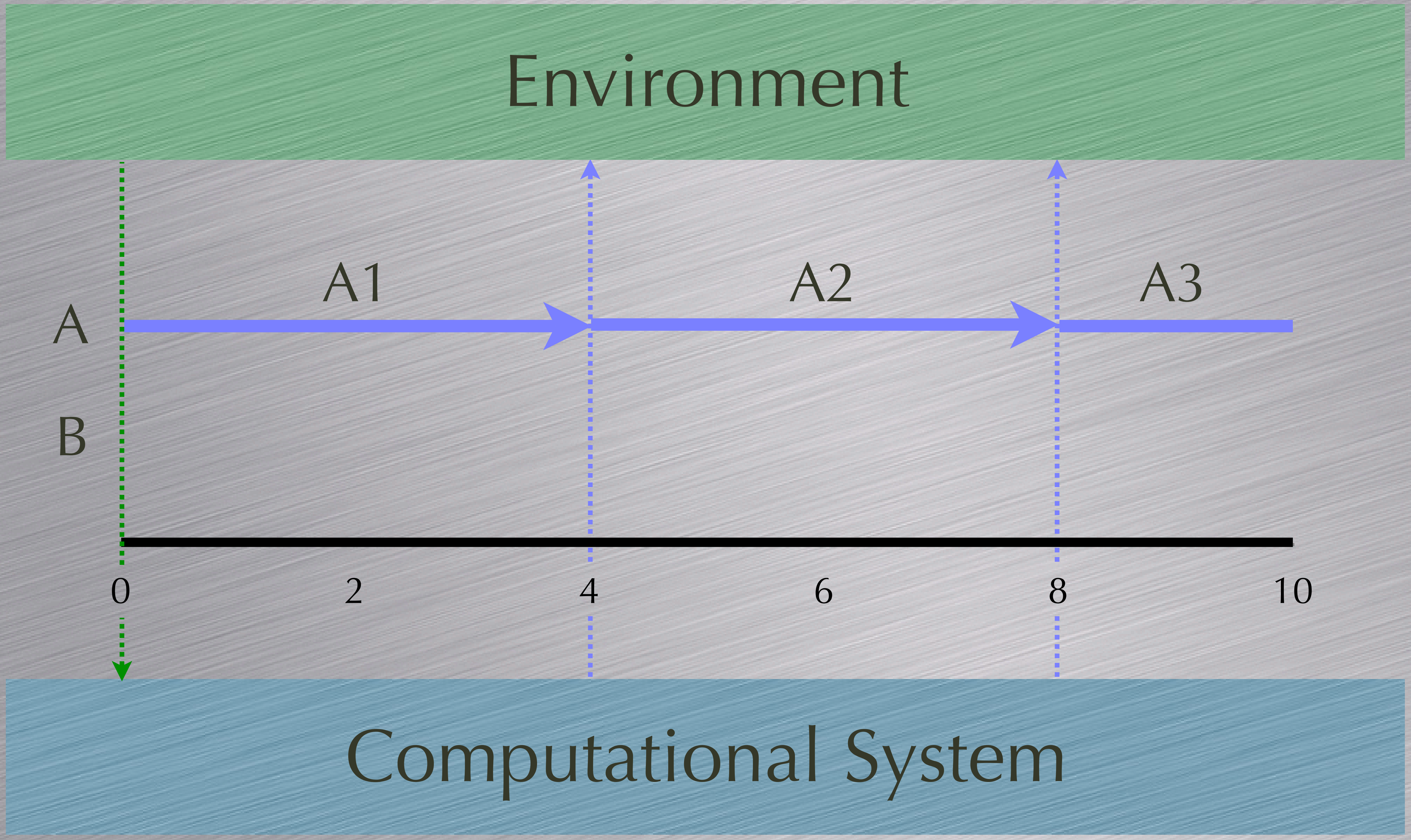
Cooperative Example



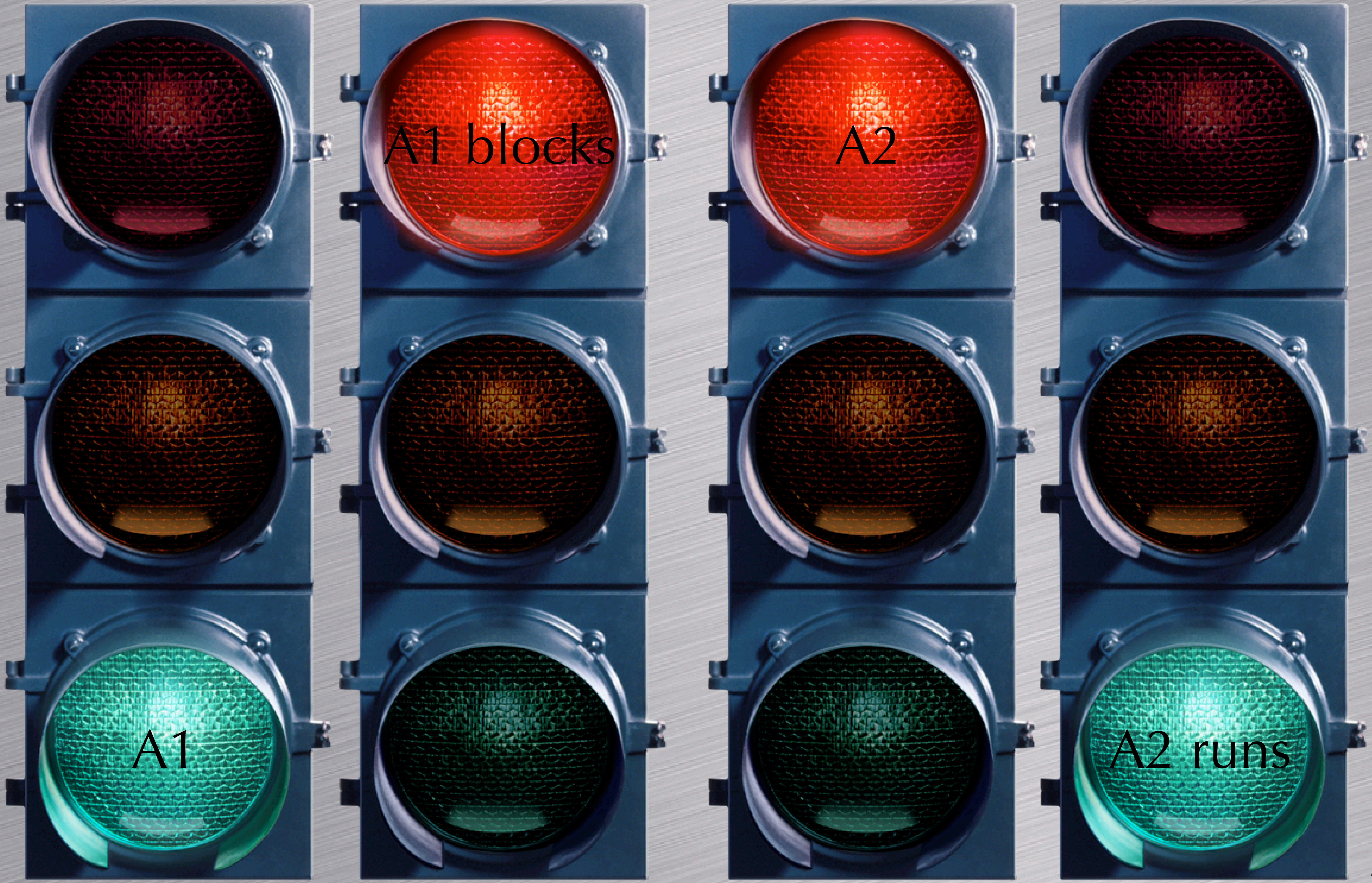
No Scheduler!

A
A runs
B
B runs

Completion Event: Chaining

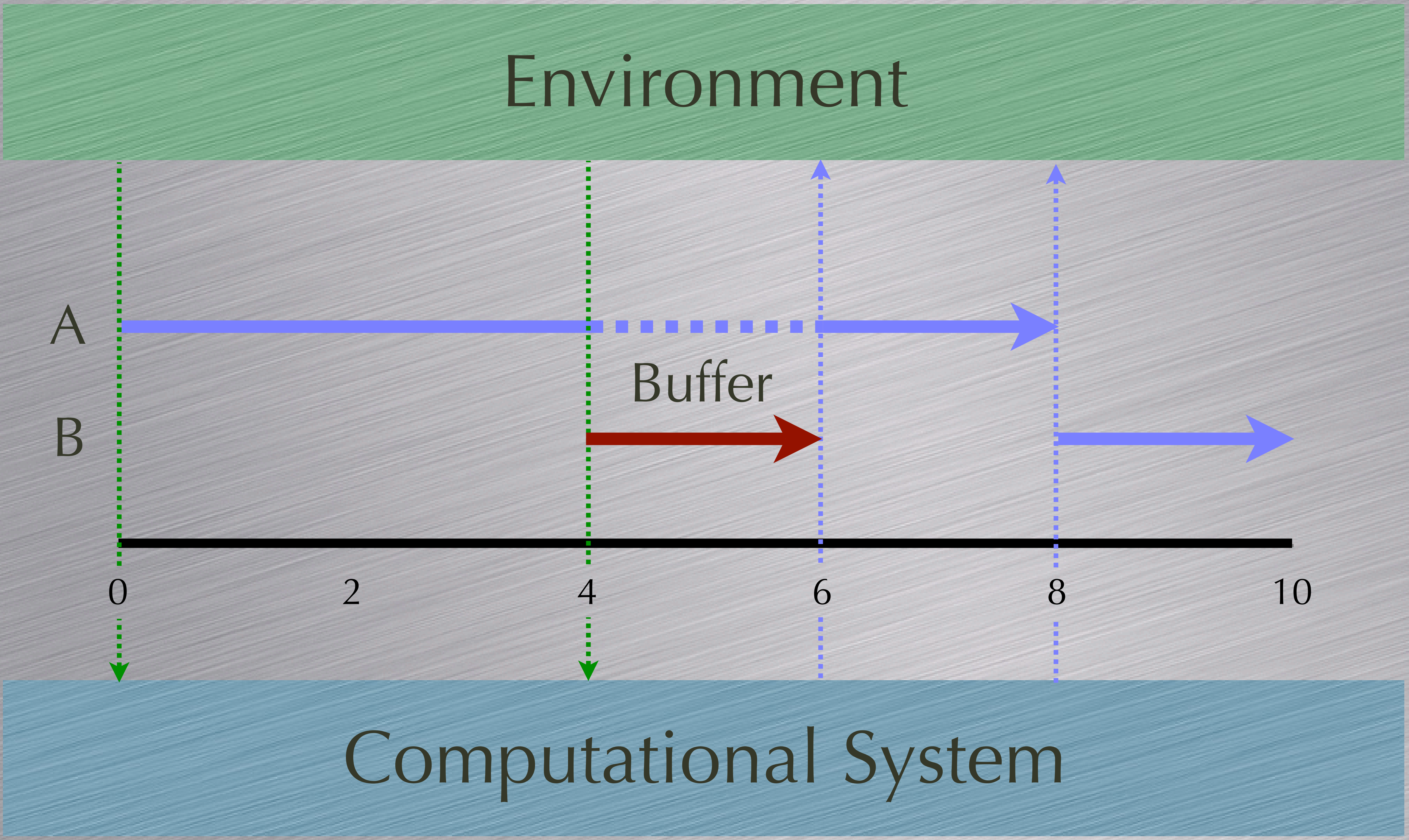


Chaining



© C. Kirsch 2004

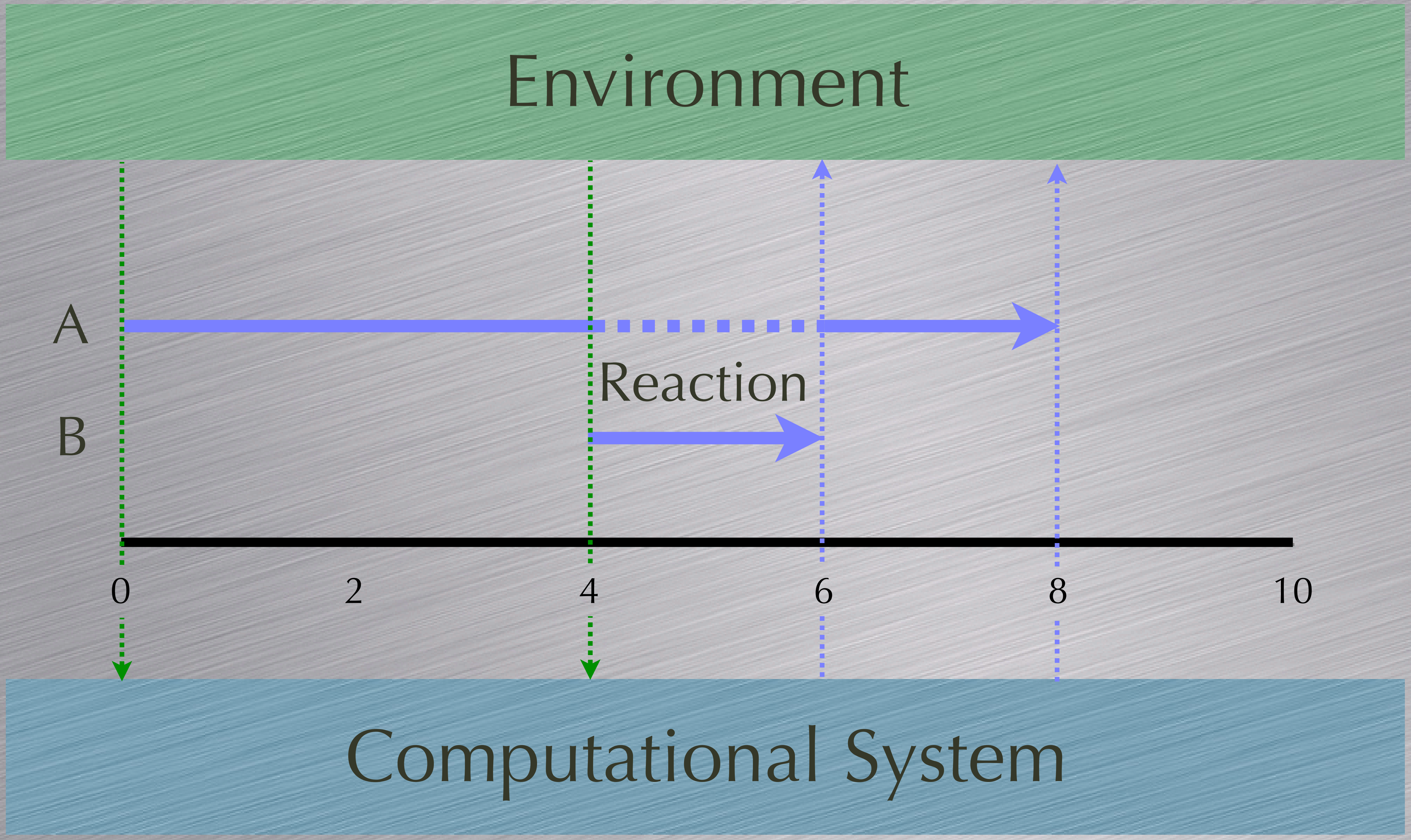
Preemptive Cooperation



Reactor vs. Scheduler

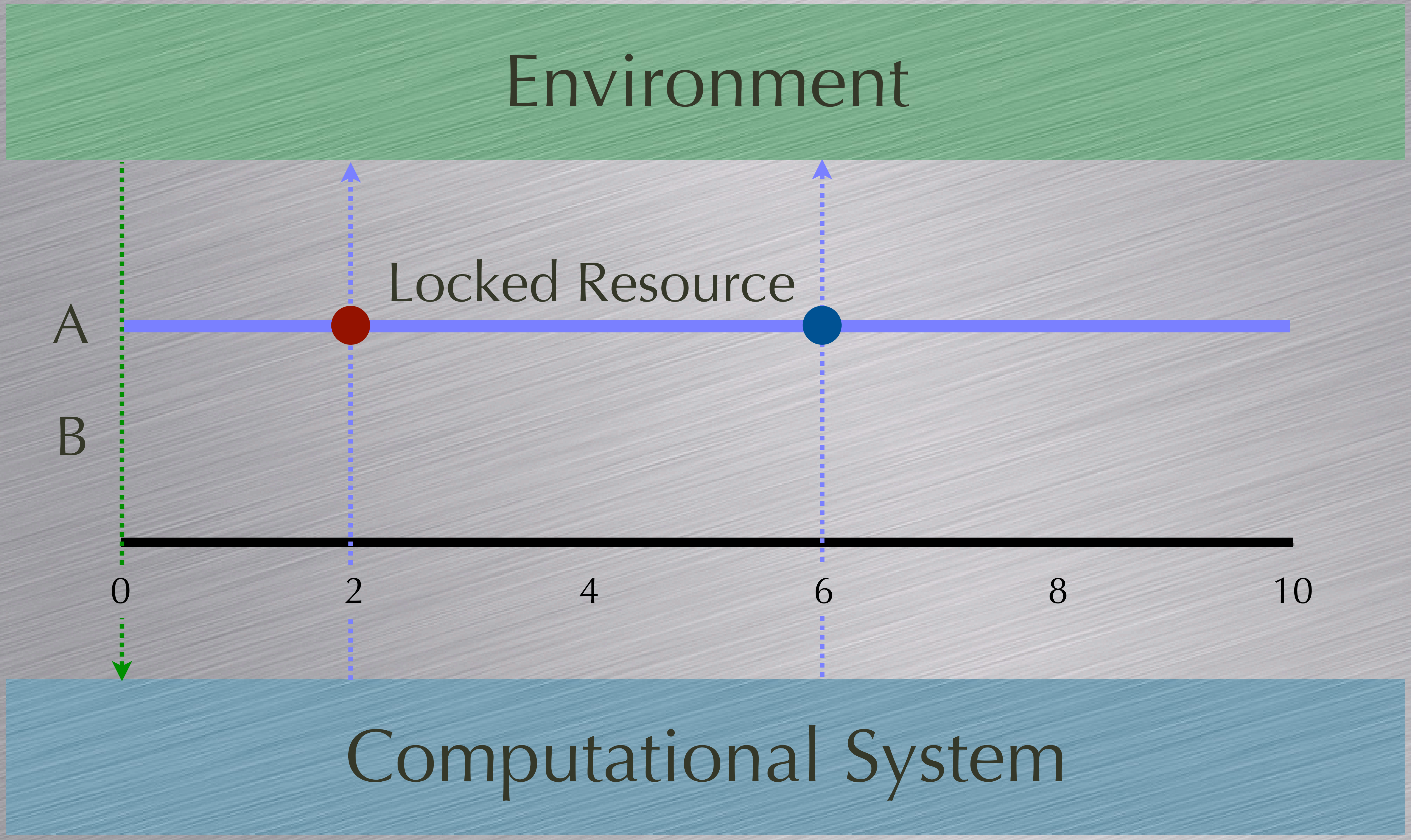
- Reactor-based: queue events and release at most one process (ex: event-driven state machine)
- Scheduler-based: release more than one process but run processes until completion (ex: state threads)

Why Full Preemption?



© C. Kirsch 2004

Locking

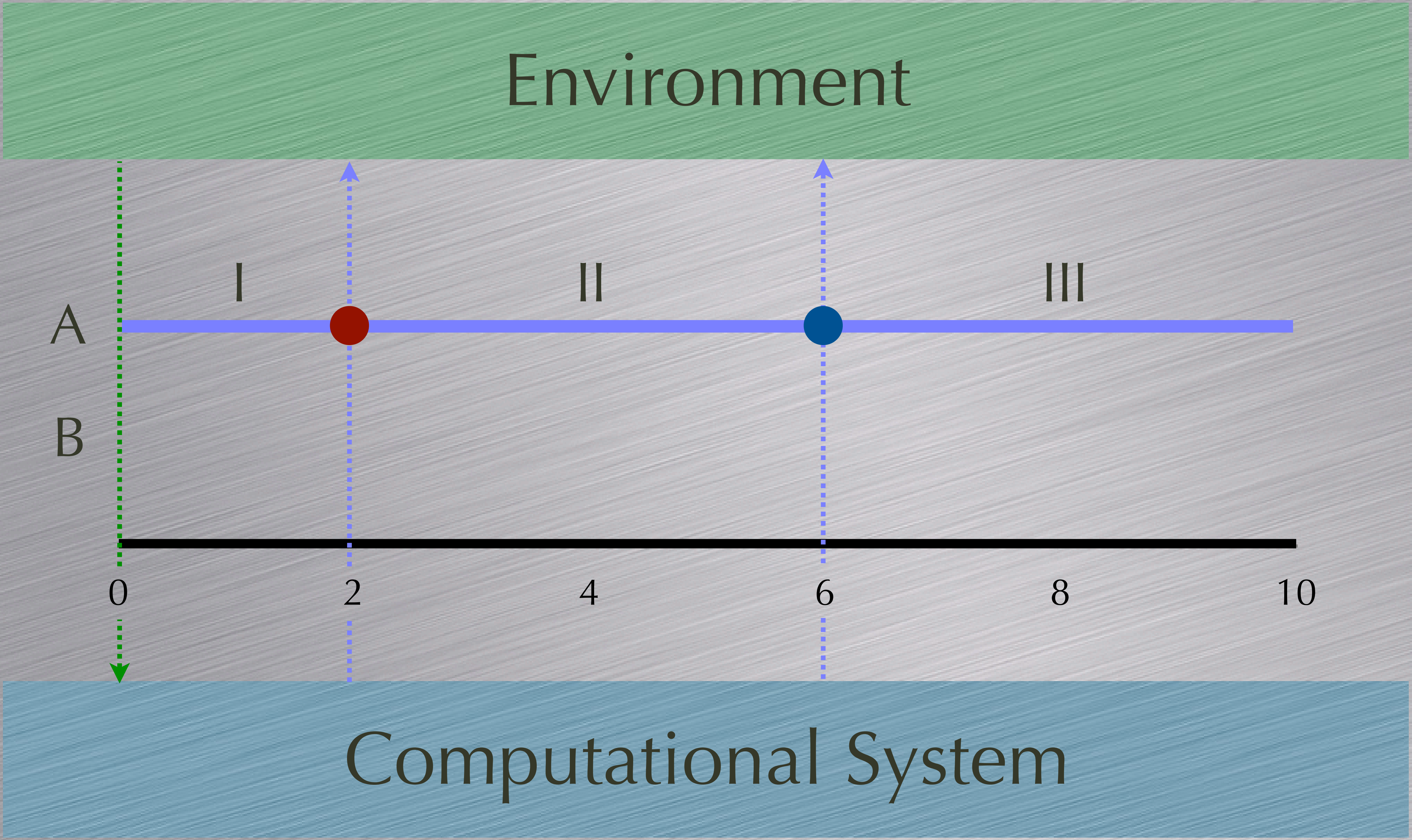

Environment
Locked Resource
A
B
0
2
4
6
8
10
Computational System

Lock Synchronization

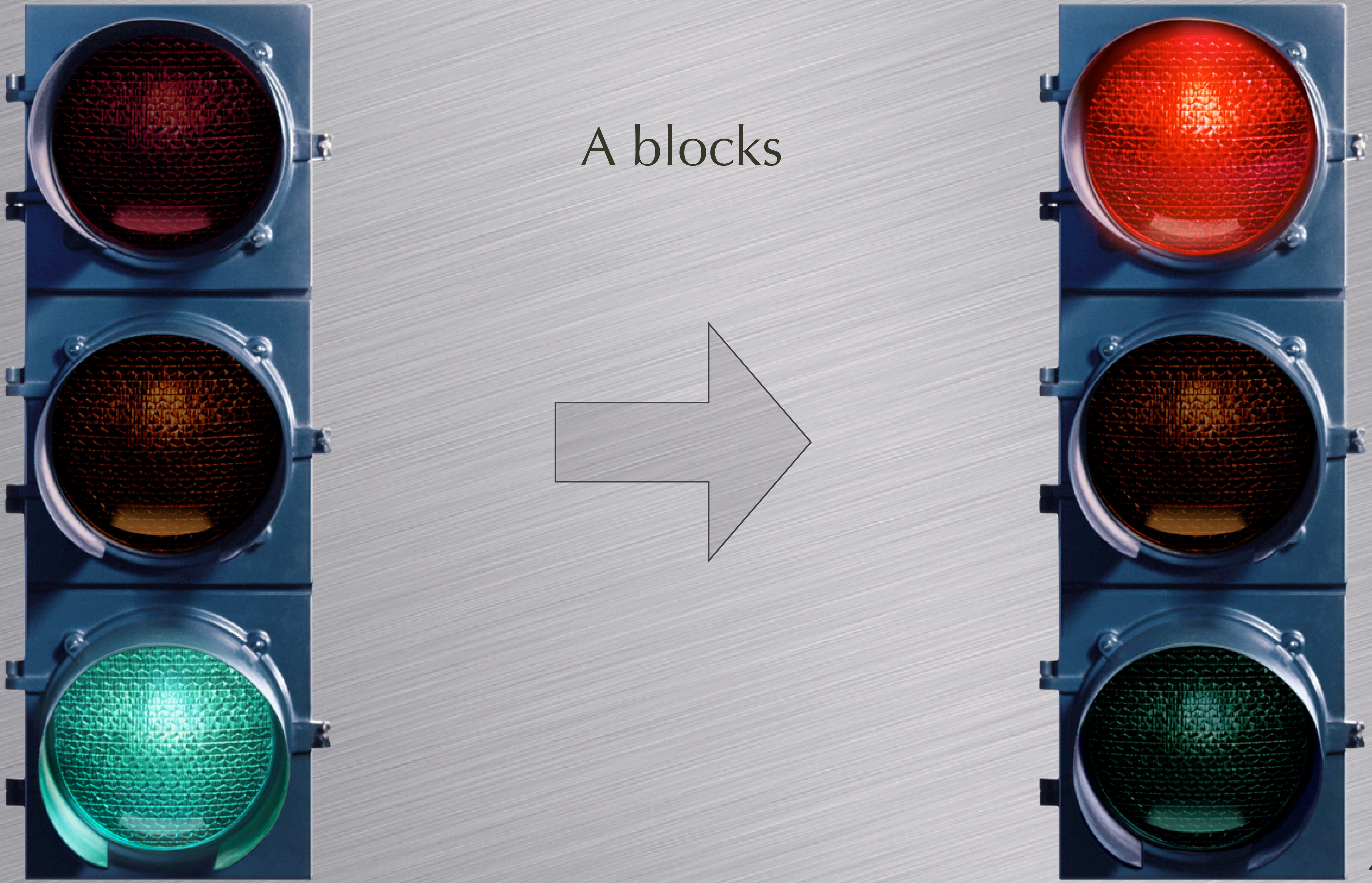
- Thread A attempts to acquire lock
- A gets the lock (uncontended case)

- Lock is owned by thread B (contended case)
- A is blocked and waits until lock is available

Phases



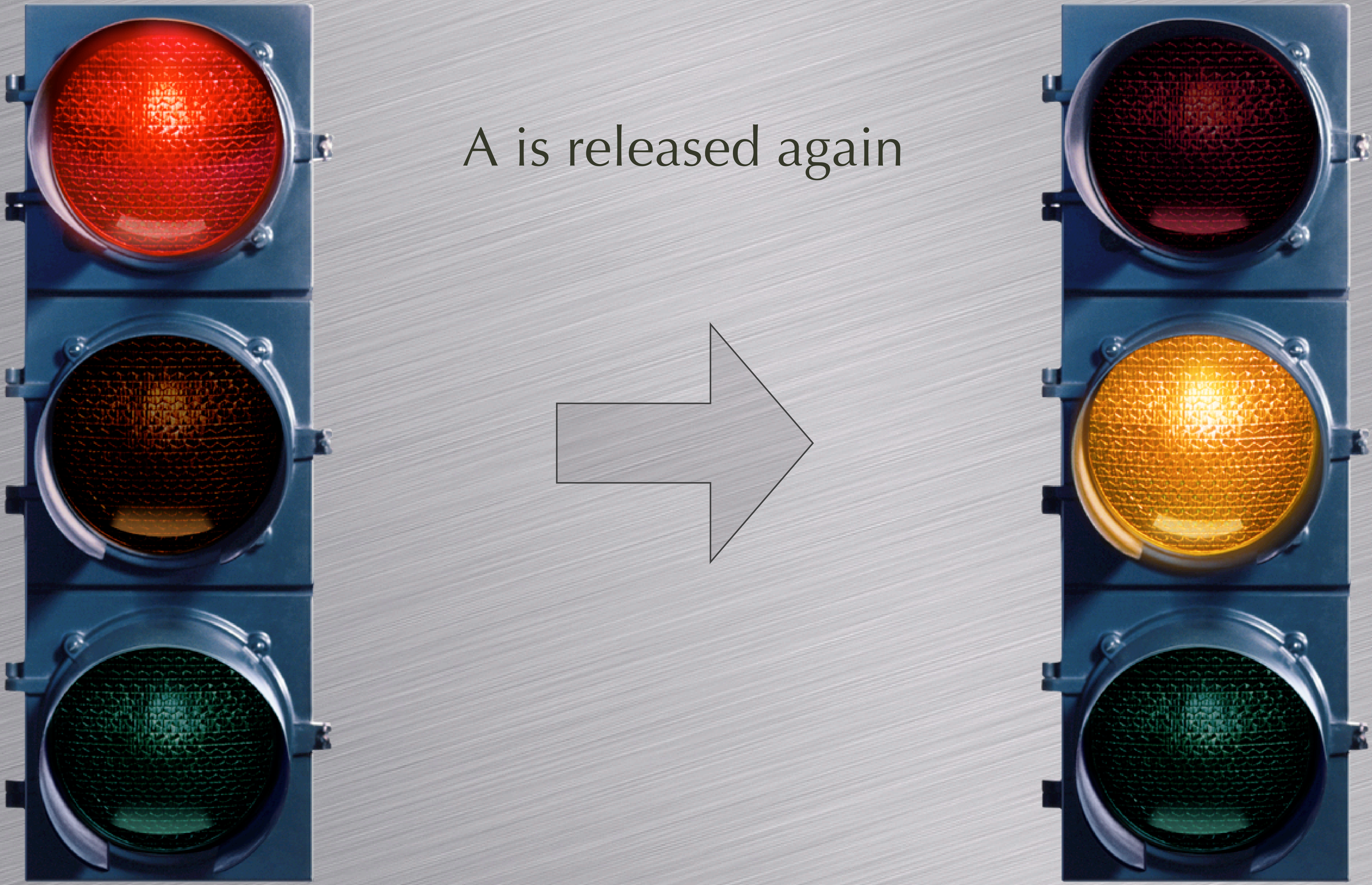
Phase I



A blocks

© C. Kirsch 2004

Still Phase I



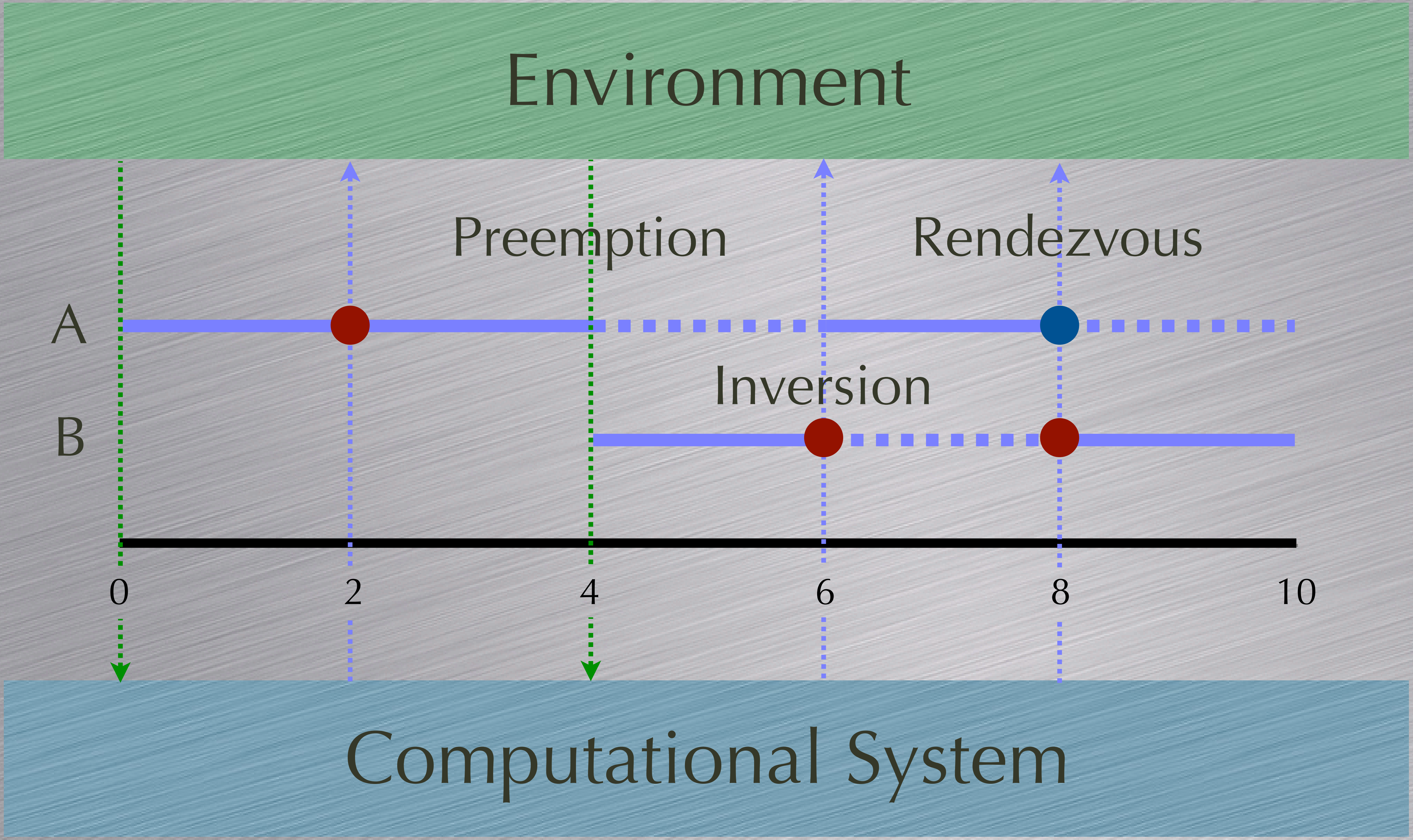
A is released again

Still, Still Phase I

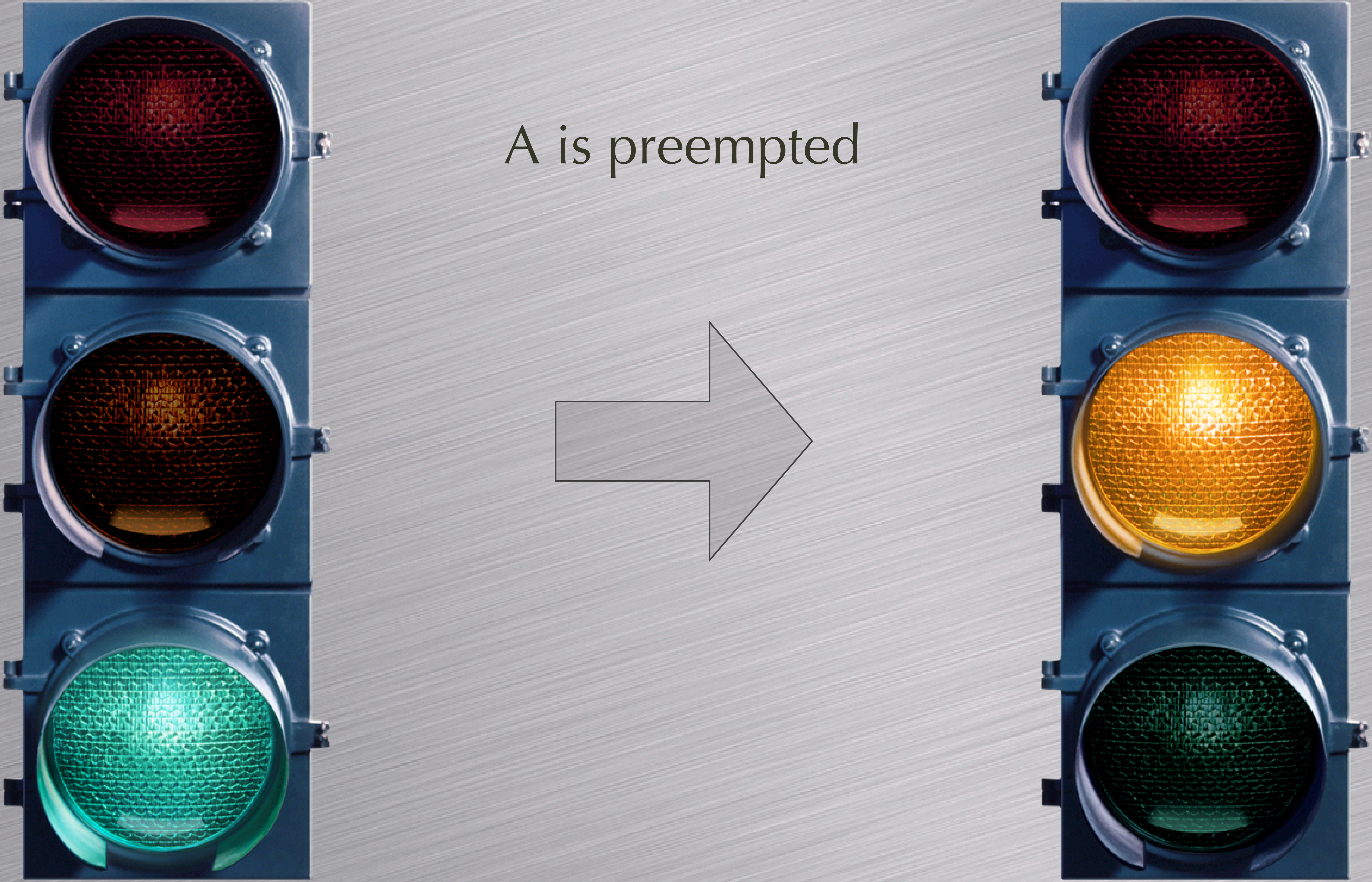
A is chosen to run

Phase II

Synchronization



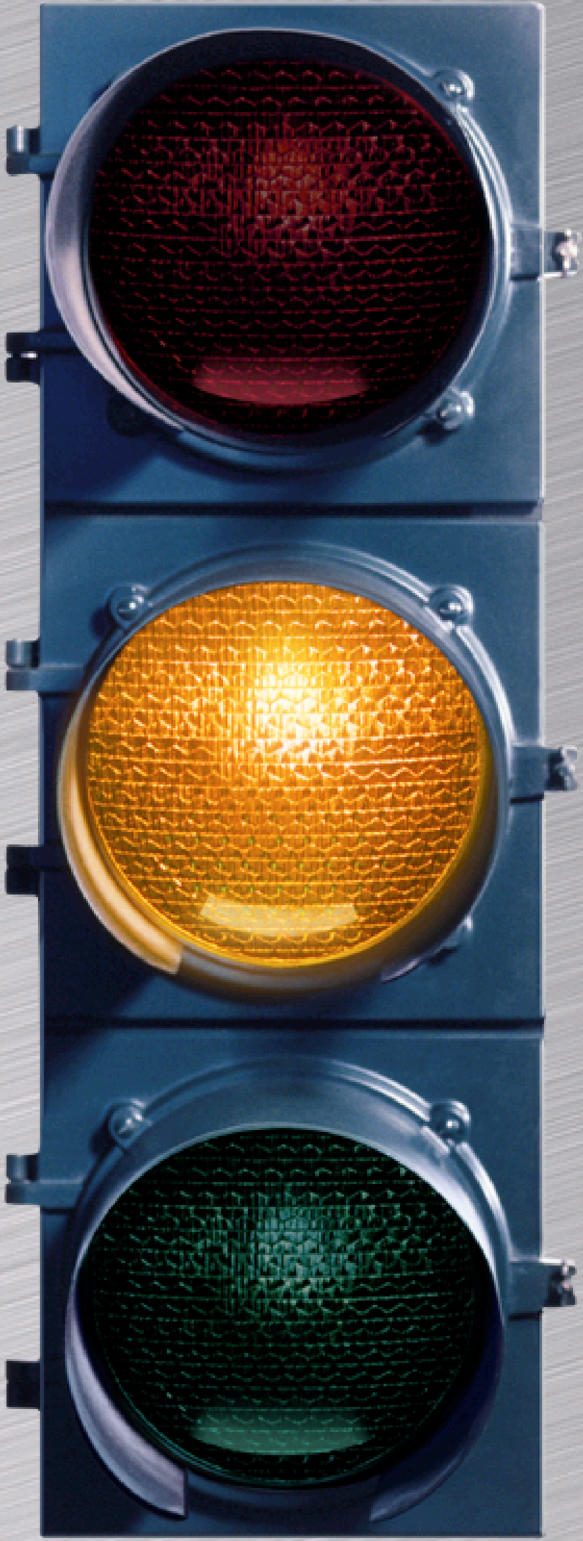
Preemption



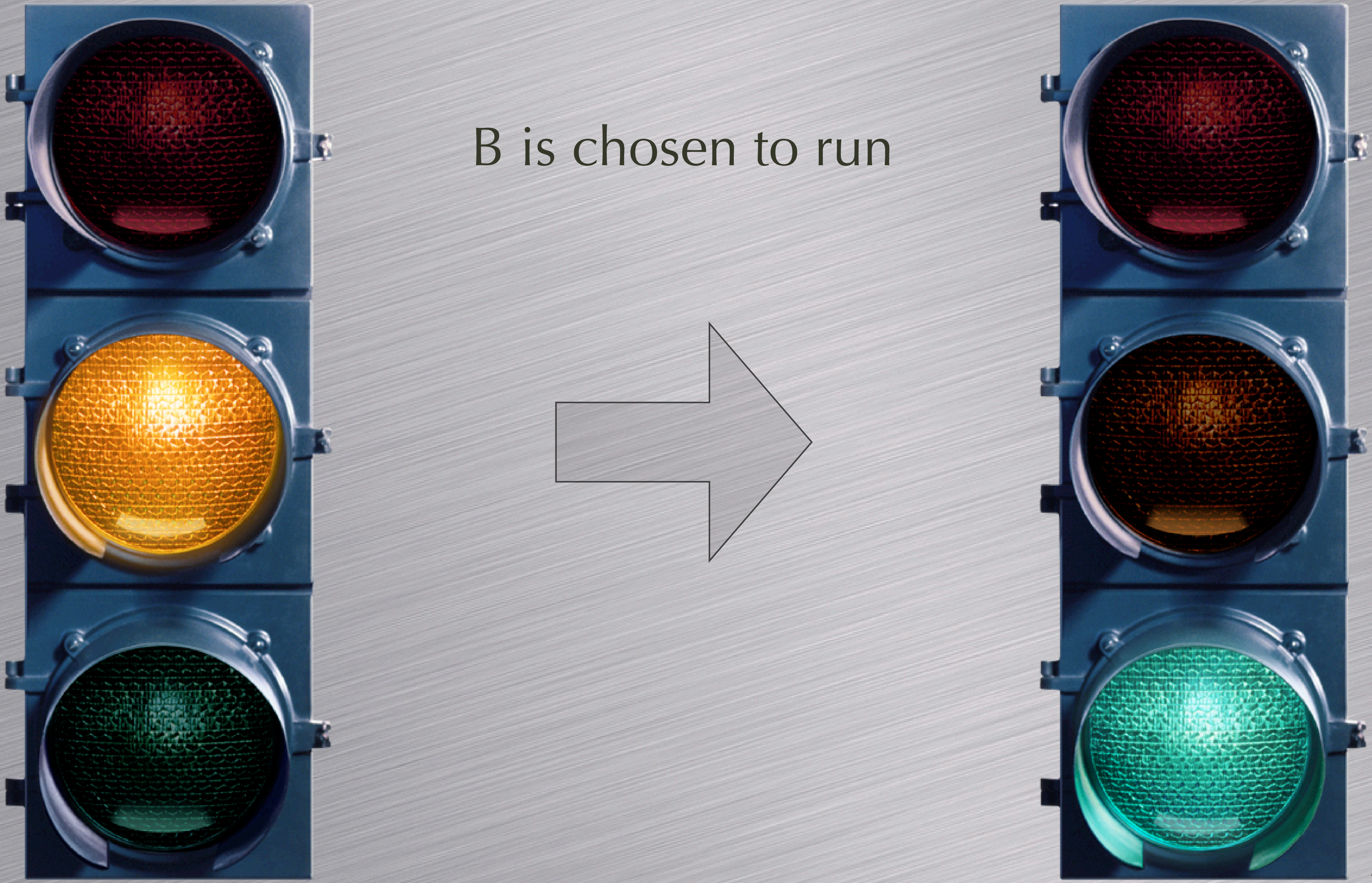
A is preempted

Still Preemption

B is released

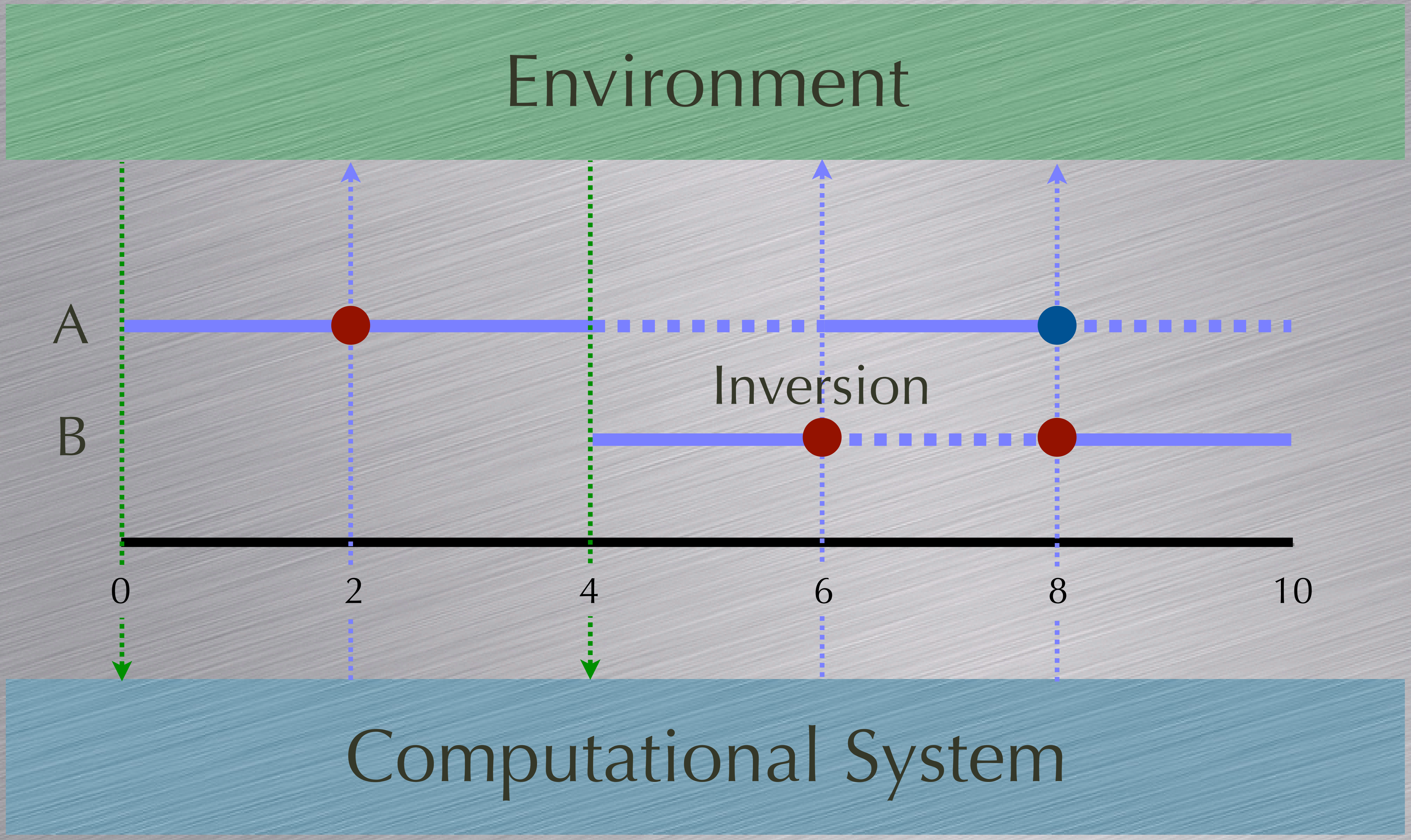


Still, Still Preemption

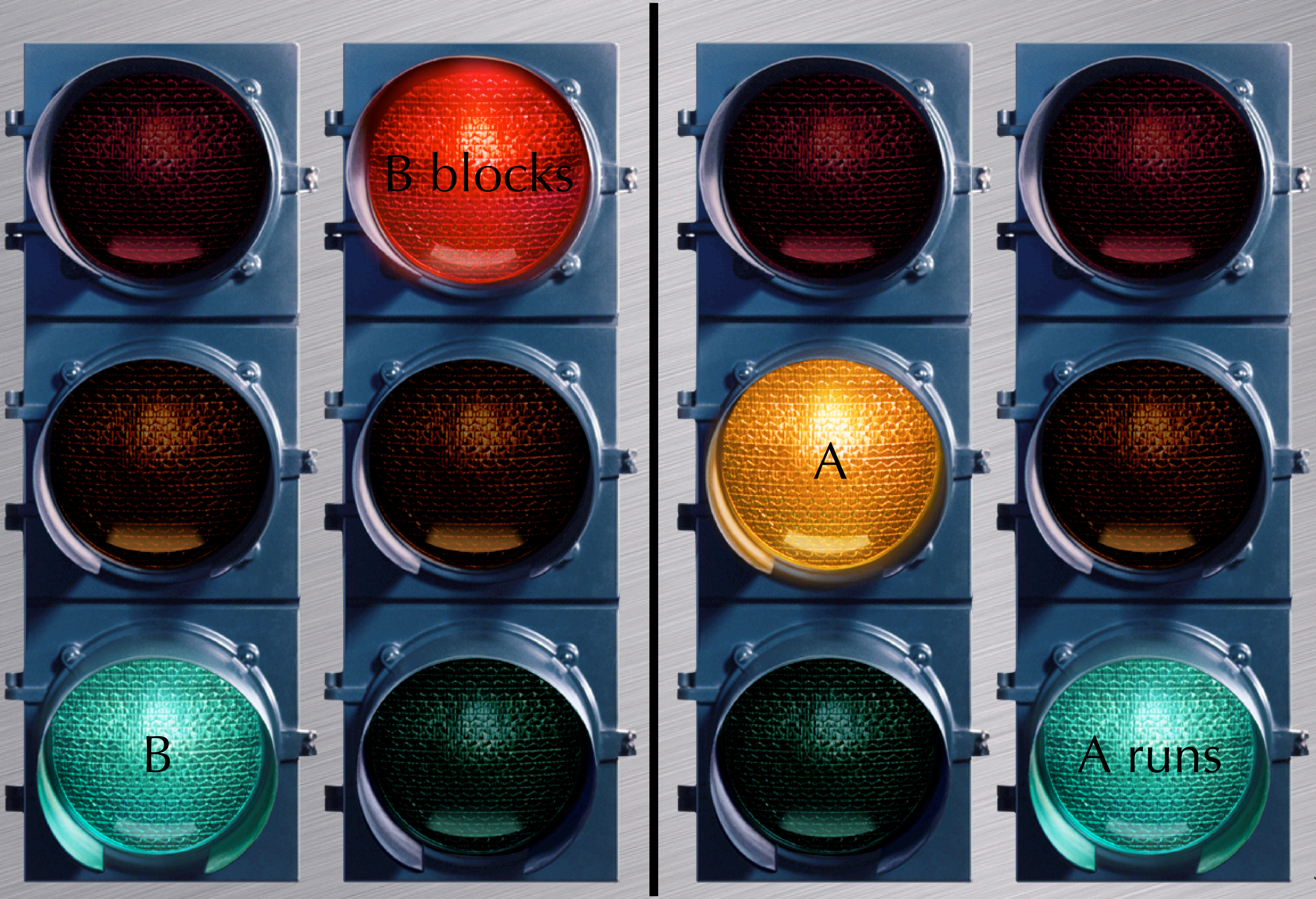


B is chosen to run

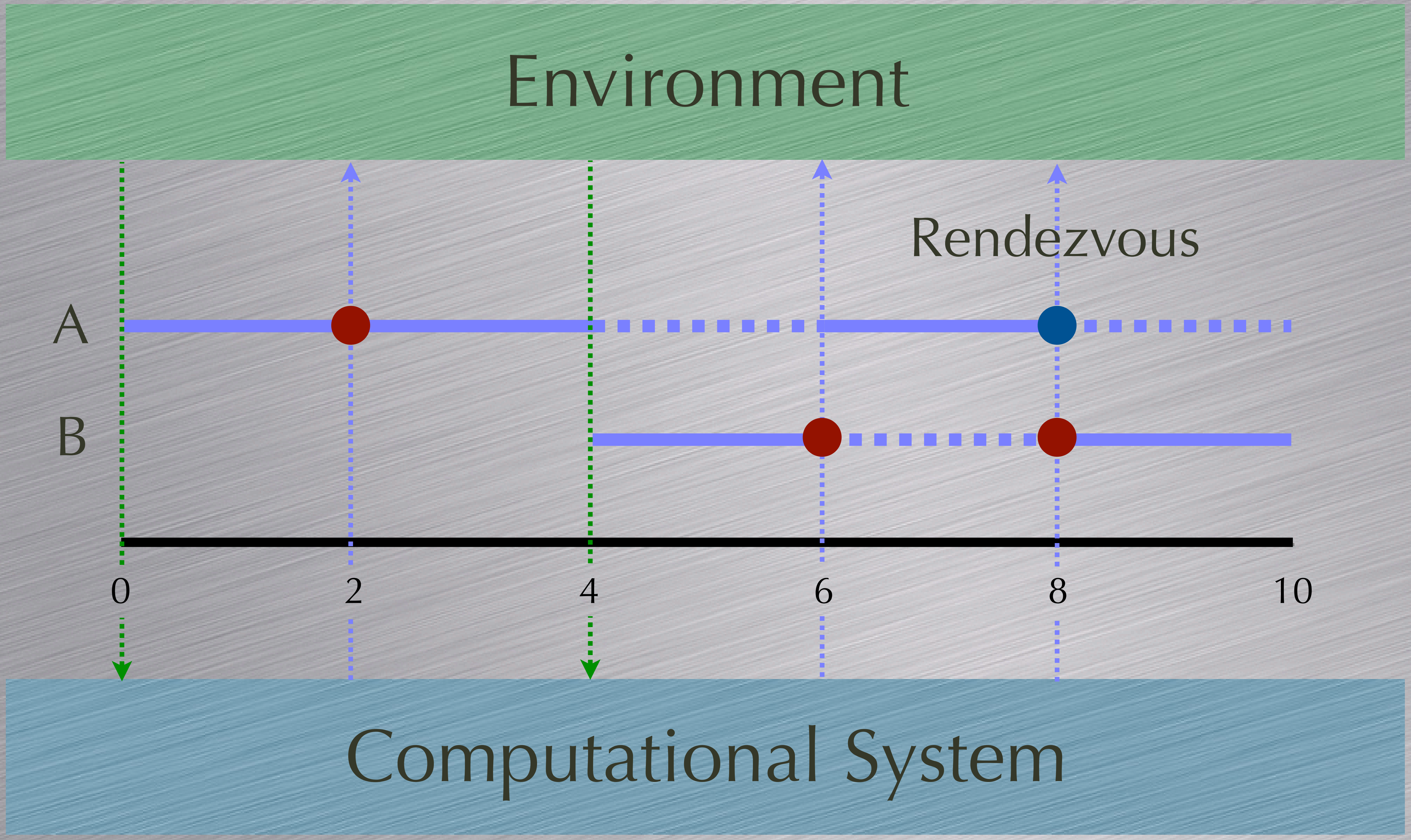
Inversion



Inversion



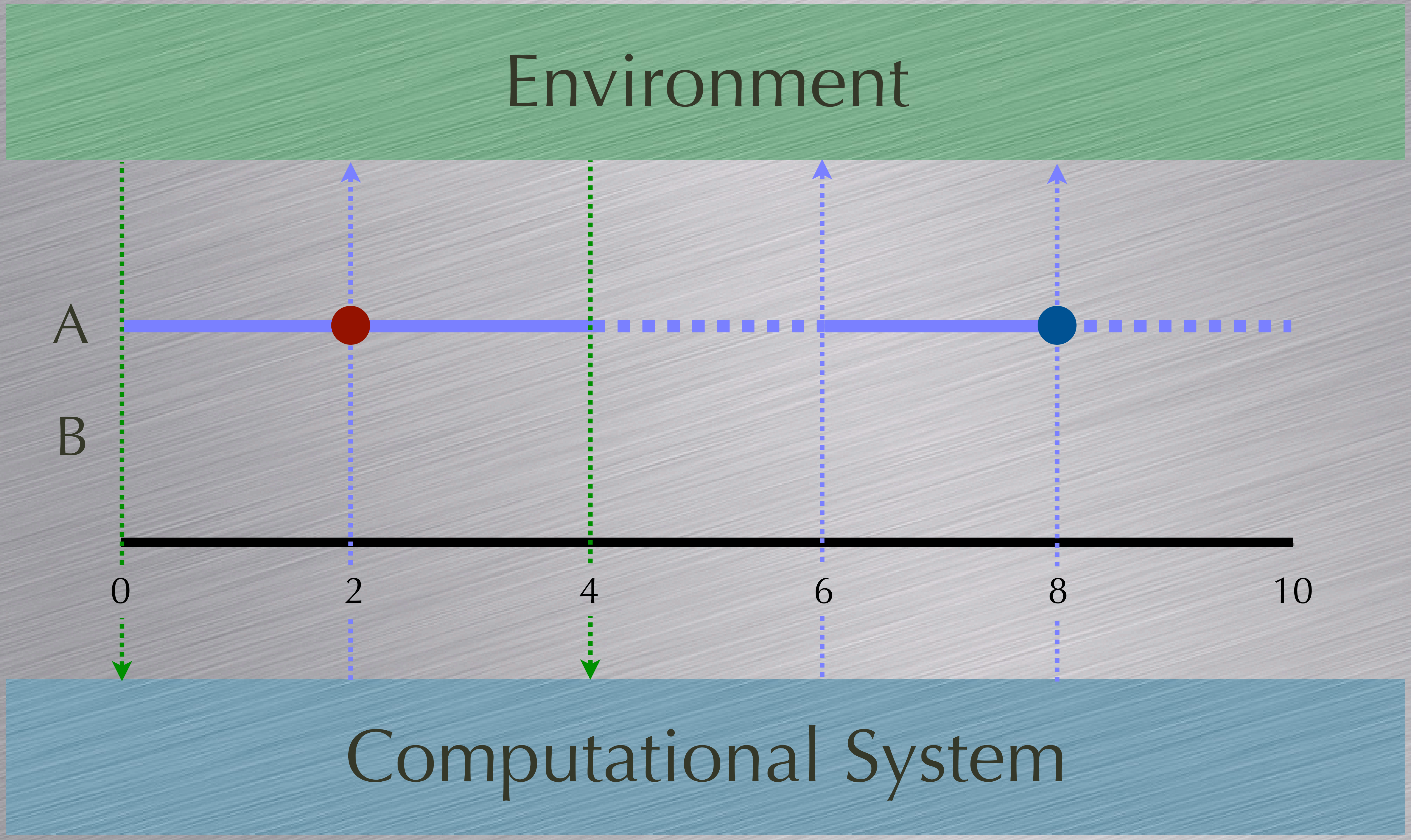
Rendezvous



Rendezvous A

Rendezvous B

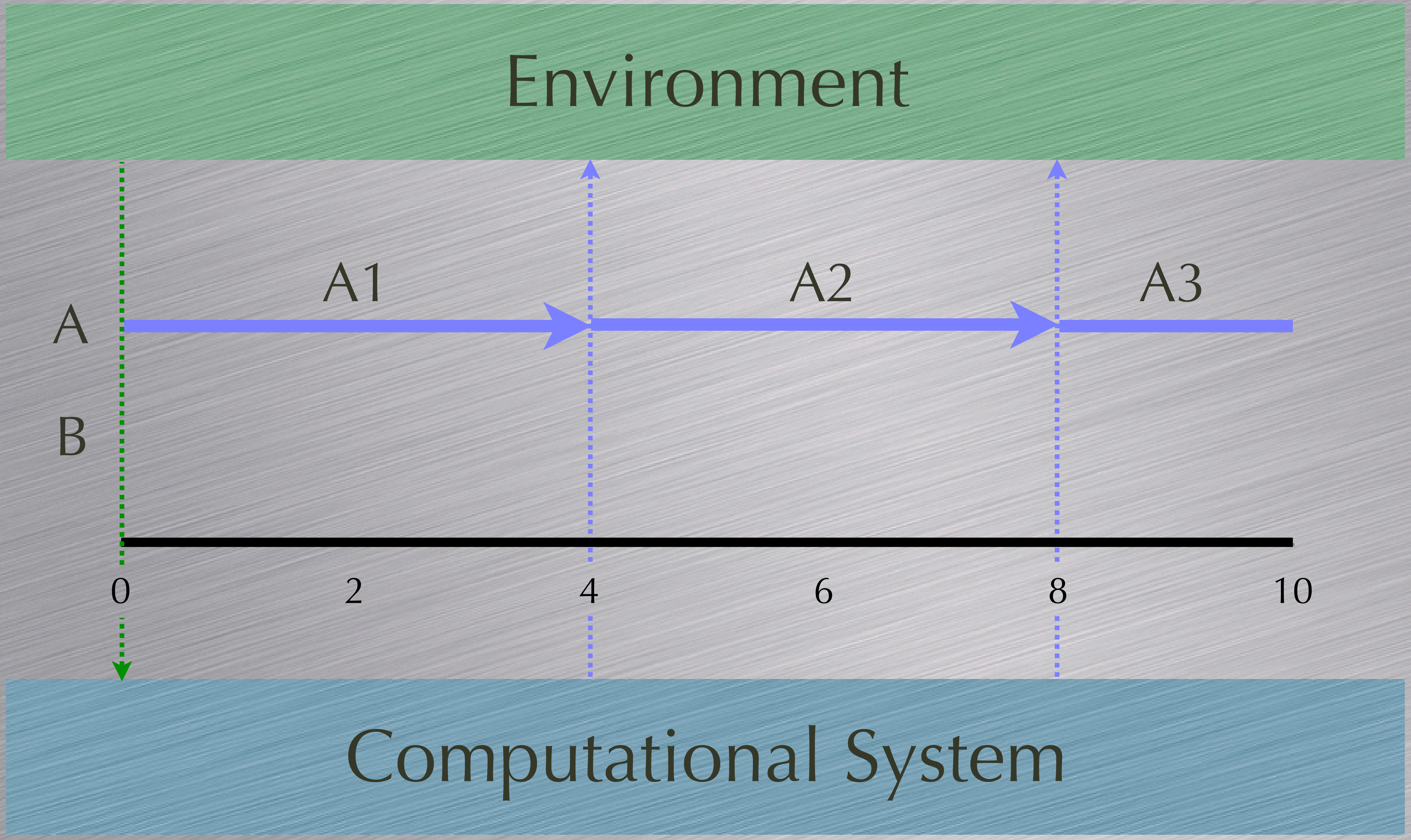
Preemptive Yet Atomic Access



Event-Driven Model

- Event queue
- Event handler table
- Callbacks (event handlers)
- Share memory on heap
- Manual stack management
- Cooperative (but could be preemptive)
- No synchronization required

Unrolling the Stack

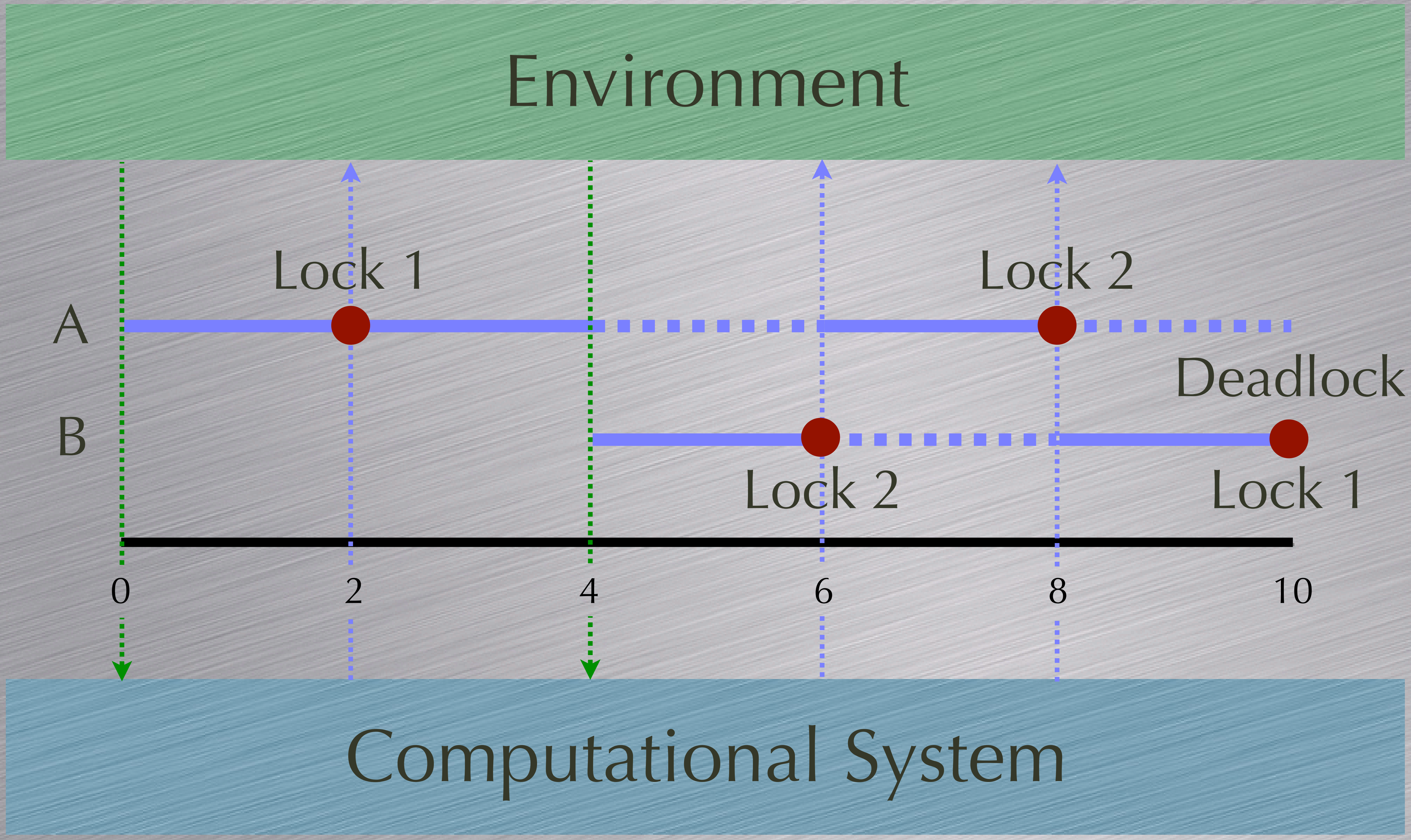


© C. Kirsch 2004

Threads

- Procedures + stack + shared heap
- Process - own heap: lightweight process
- Share memory on heap
- Automatic stack management
- Preemptive (but could be cooperative)
- Requires synchronization
- Deadlock, Race Conditions

Deadlock


Environment
Lock 1
Lock 2
A
Deadlock
B
Lock 2
Lock 1
0
2
4
6
8
10
Computational System

Context Switch

1. Interrupt or yield
2. Save stack
3. Do something (reactor)
4. Do something (scheduler)
5. Restore stack
6. Switch

setjmp/longjmp

- `int setjmp (jmp_buf env)`
  saves context in `env`

-

- `int longjmp(jmp_buf env, int val)`
  restores context from `env` previously saved by `setjmp`

Example

```
#include<setjmp.h>

main() {
   jmp_buf env;
   int i;

   i=setjmp(env);
   printf("i= %d\n",i);
   if(i==0)
     printf("I am in if ..\n");
   else {
     printf("I am in else too...\n");

     exit(0);
   }
   longjmp(env,2);
   printf("Grrr... why am i not getting printed\n");
}
```