# Distributed Computing

José Orlando Pereira

Grupo de Sistemas Distribuídos
Departamento de Informática
Universidade do Minho

2009/2010

# Asynchronous systems

- Assume no bounds on:
  - clock drift
  - processing time
  - message passing time

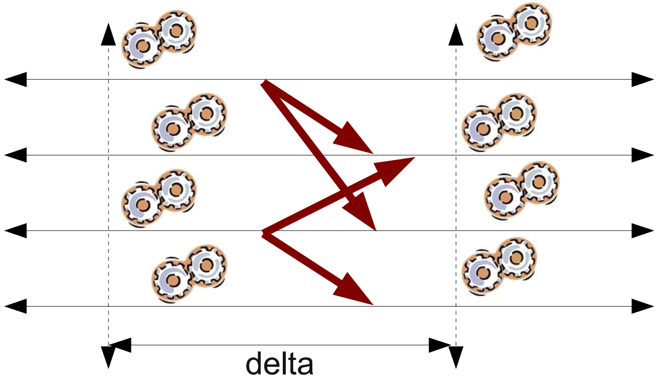
# Goals

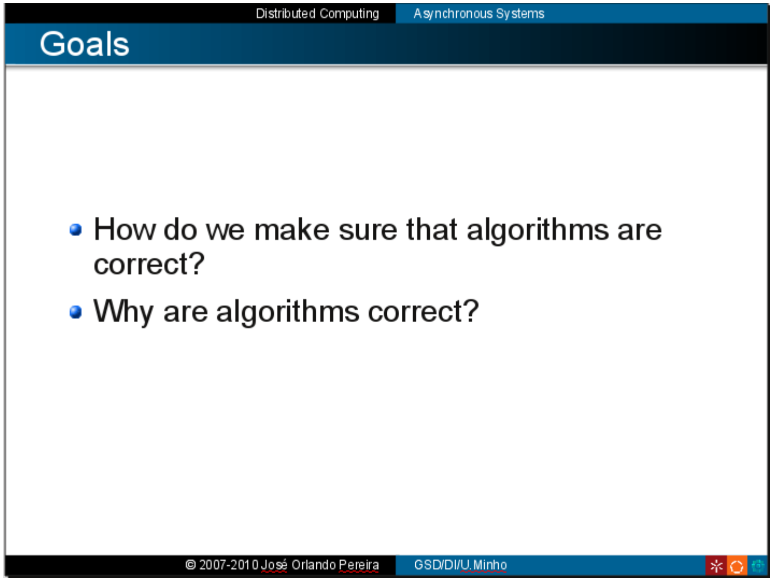
- How do we make sure that algorithms are correct?
- Why are algorithms correct?

# Synchronous System

- With synchronous rounds:
  - Simple proofs by induction
  - Local state easily reflects global state



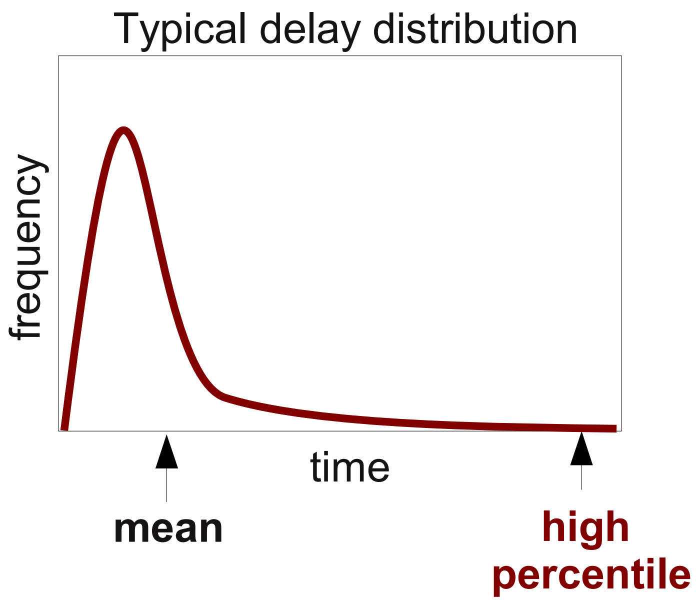



# Synchronous System

- Are solutions obtained with the synchronous system applicable?
- Not really...
  - The practitioner's argument
  - The theoretician's argument

# In practice

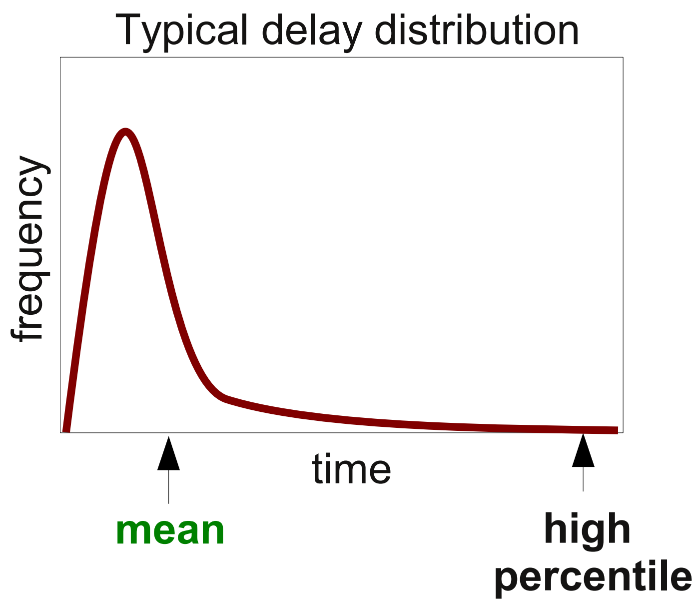
- Tight synchronous limits are dangerous:
  - Round time proportional to mean delay
  - Low coverage or expensive systems




- Large synchronous limits are not useful:
  - Round time proportional to high percentile delay
  - Taking advantage of synchrony causes a very large performance penalty

# In practice

- Solutions for asynchronous systems might have better performance:
  - Round time proportional to mean delay
  - Even if more message exchanges are necessary



# In theory

- Start with a synchronous reliable fully connected network
- Relax the system model:
  - Unbounded message loss
  - Large/unknown graph diameter
  - Dynamic graph
- Example: Leader election

# Example: Leader election

Static known participants

Synchronous
Reliable static

Synchronous
Reliable dynamic

Synchronous
Reliable clique

Synchronous
Unreliable clique

Synchronous
Bounded unreliable
Clique

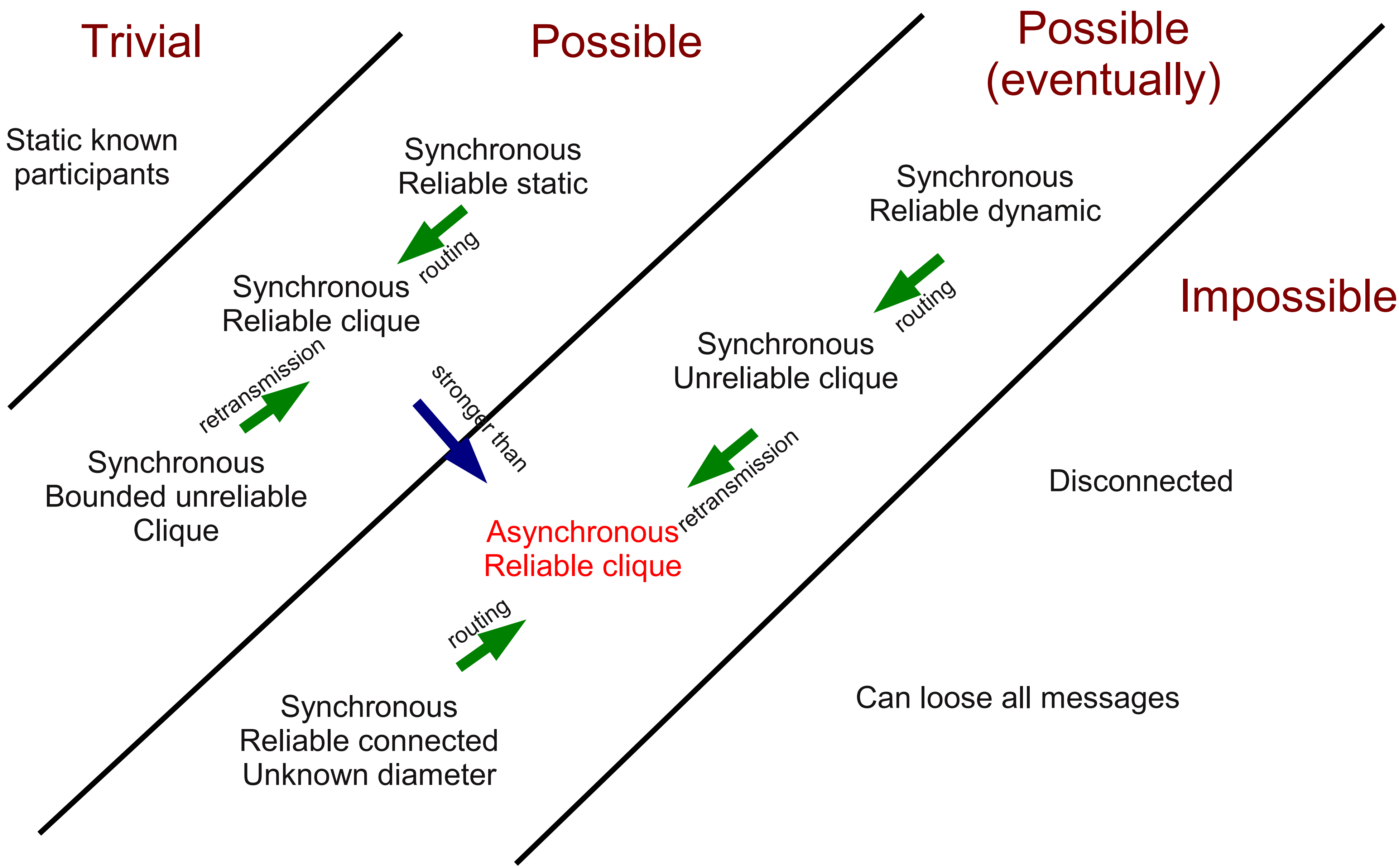
Disconnected

Asynchronous
Reliable clique

Synchronous
Reliable connected
Unknown diameter

Can loose all messages

# Example: Leader election



# In theory

- Asynchronism subsumes:
  - Heterogeneity
  - Dynamics
  - Uncertainty
- Much simpler than handling them explicitly
- Often considered an Universal model:
  - Widely applicable solutions

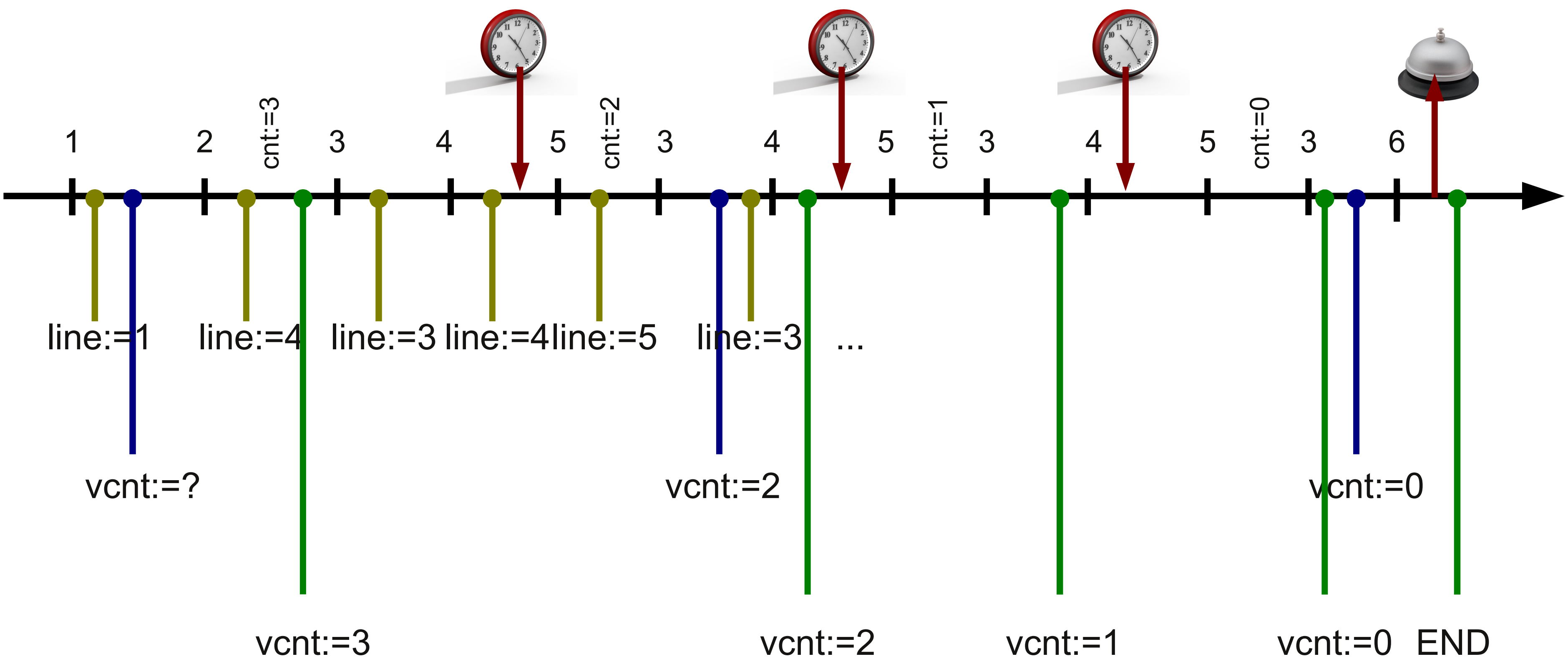
# Sample computation

- An alarm clock program:

```
main:                    // line 1
    cnt:=3               // line 2
    while cnt>0:         // line 3
        sleep 1s         // line 4
        cnt := cnt-1     // line 5
    ring                 // line 6
```

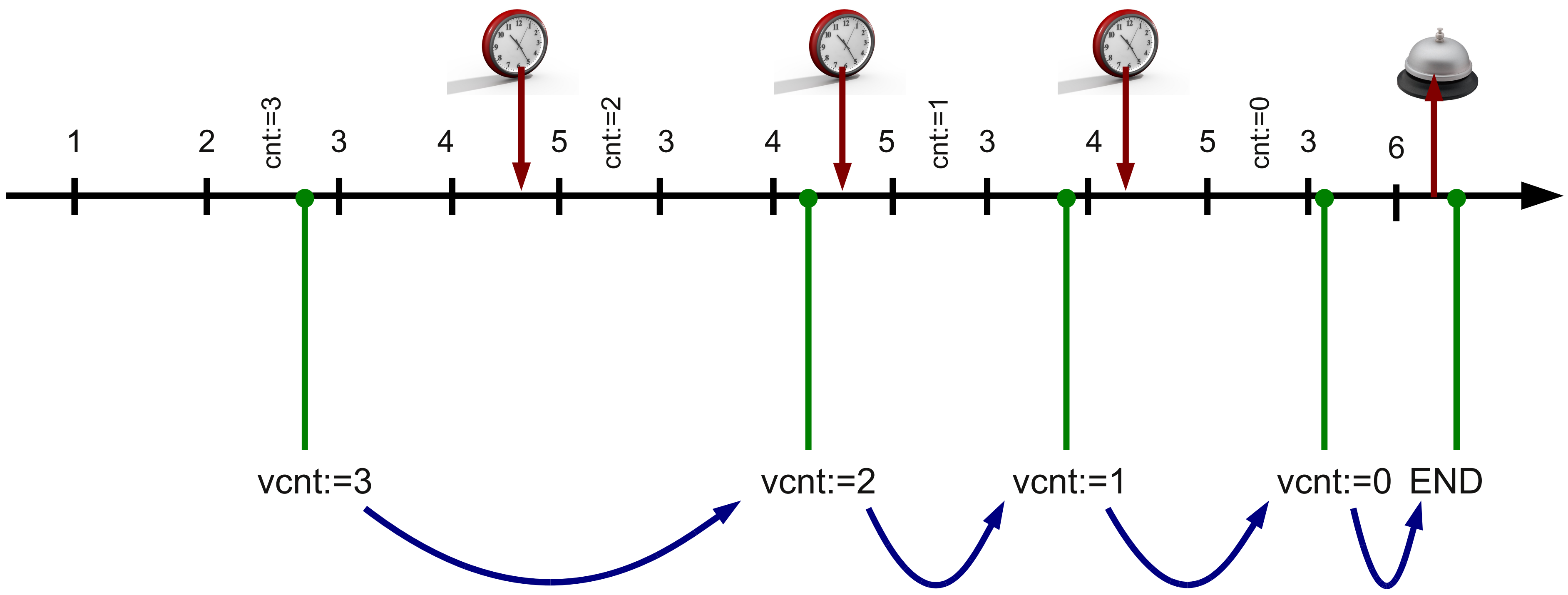
# Observation

- Select model variables and periodically observe the system:



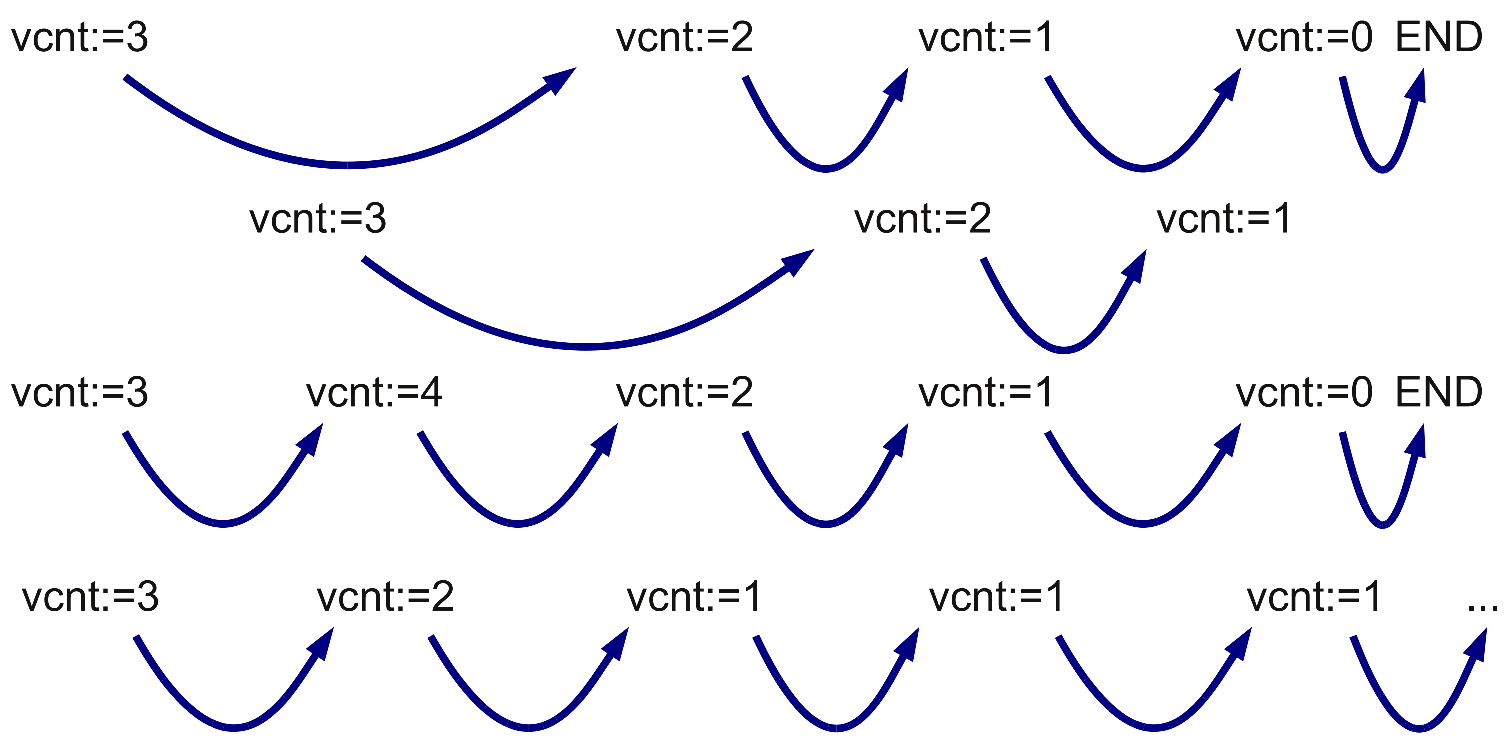
# Abstraction

- Choose observation that allows reasoning on the desired properties:



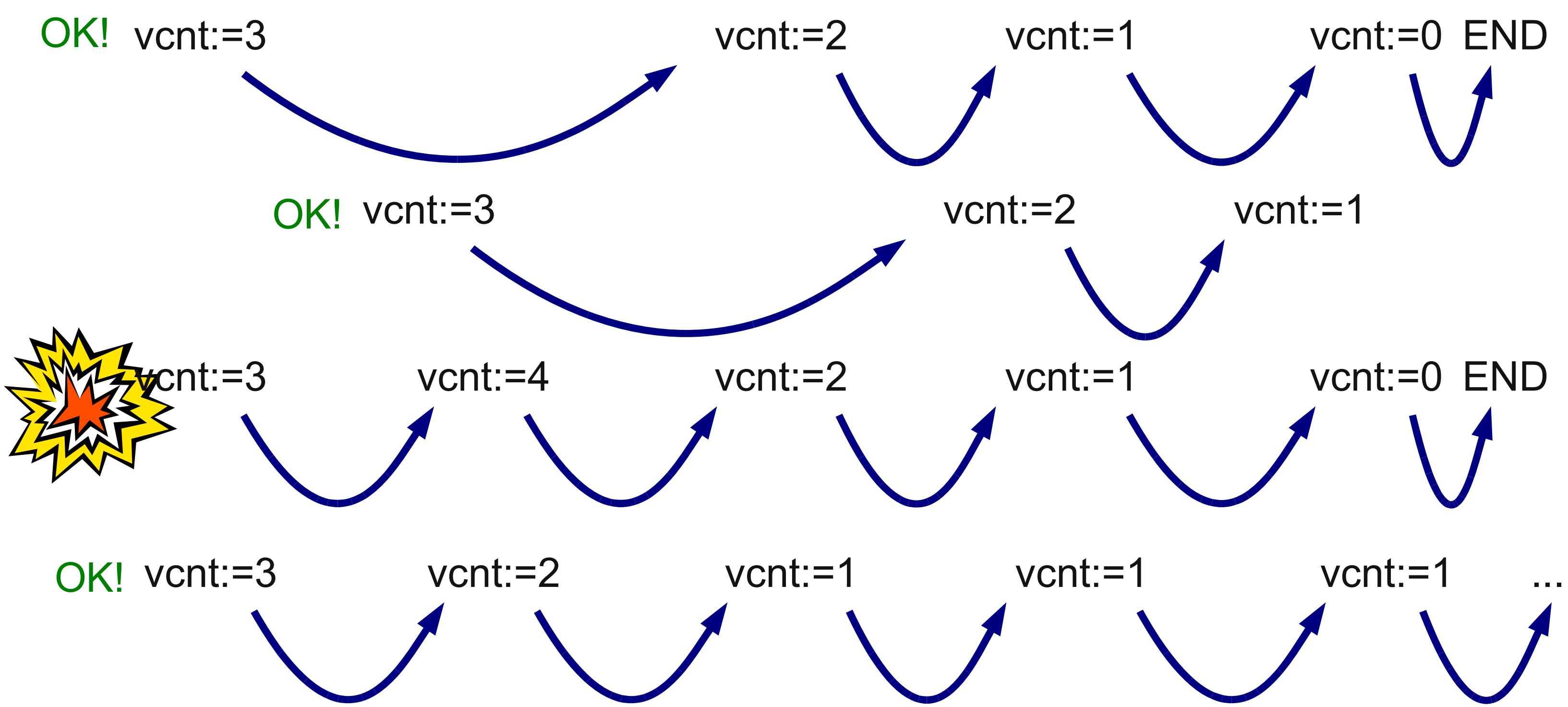
# Behaviors/Executions

- Consider all possible sequences of chosen atomic actions:



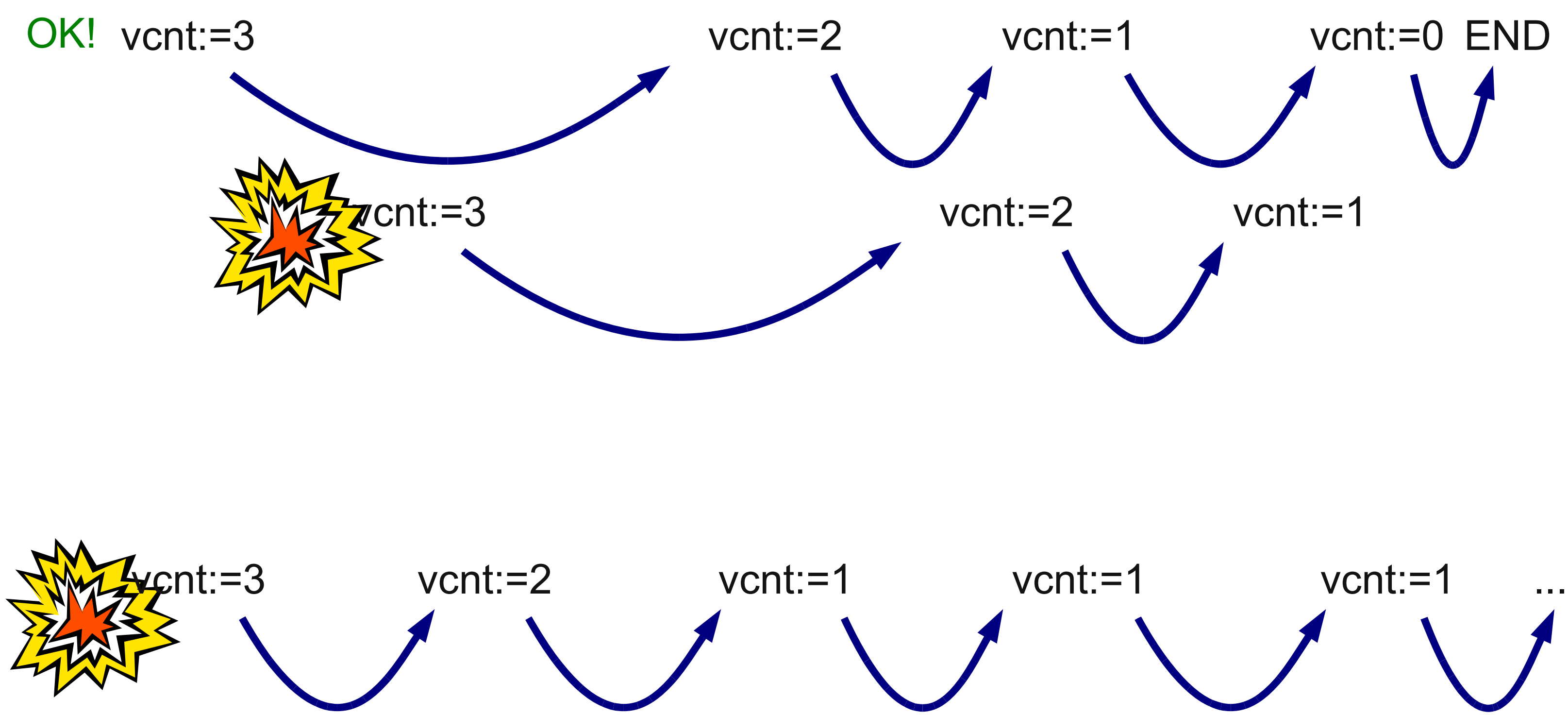
# Safety properties

- Nothing bad ever happens:



# Liveness properties

- Something good eventually(*) happens:




(*) eventually = inevitavelmente ≠ eventualmente

# Specification

- Specification is a set of allowable behaviors:

$$S=\left\{ \text{vcnt:=3} \xrightarrow{\text{timeout}} \text{vcnt:=2} \xrightarrow{\text{timeout}} \text{vcnt:=1} \xrightarrow{\text{timeout}} \text{vcnt:=0} \xrightarrow{\text{ring}} \text{END} \right\}$$

# Goal 1: Is it correct?

- Is there a convenient representation for specification sets?
  - Compact
  - Practical
- How to prove safety and liveness properties?