Concurrent Algorithms and Data Structures

Concurrent Algorithms and Data Structures

Parosh Aziz Abdulla

parosh@it.uu.se

Concurrent Algorithms and Data Structures

Parosh Aziz Abdulla

parosh@it.uu.se

Sarbojit Das

sarbojit.das@it.uu.se

Concurrent Algorithms and Data Structures

Concurrent Algorithms and Data Structures

Examination

Concurrent Algorithms and Data Structures

Examination

- assignments
  - three theoretical
  - three programming
  - re-submission possible: 1-10 April, 1-10 June

Concurrent Algorithms and Data Structures

solved individually

Examination

- assignments
  - three theoretical
  - three programming
  - re-submission possible: 1-10 April, 1-10 June

Concurrent Algorithms and Data Structures

solved individually

Examination

- assignments
  - three theoretical
  - three programming
  - re-submission possible: 1-10 April, 1-10 June

- To pass
  - pass written exam
  - pass all six assignments

Concurrent Algorithms and Data Structures

solved individually

Examination

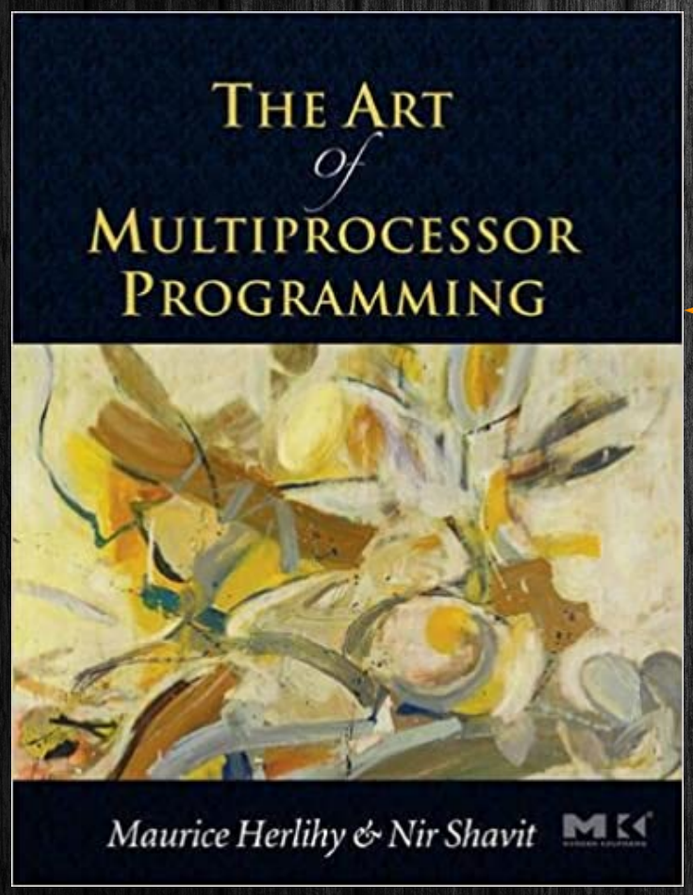
- assignments
  - three theoretical
  - three programming
  - re-submission possible: 1-10 April, 1-10 June

- To pass
  - pass written exam
  - pass all six assignments

- Bonus System:
  - 2% in the final exam, for each passed assignment before the 1st deadline

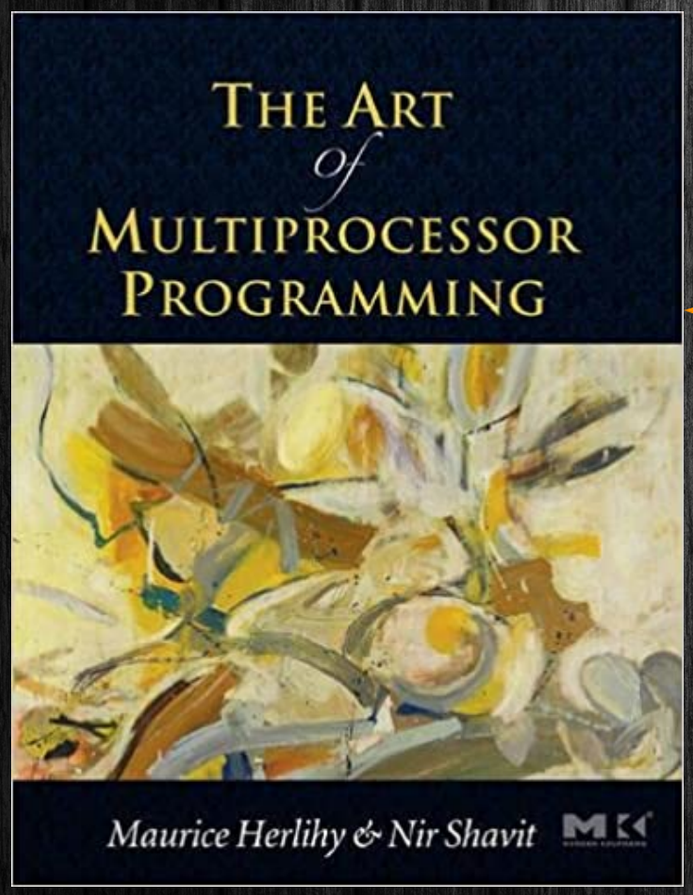
Concurrent Algorithms and Data Structures

Concurrent Algorithms and Data Structures



literature

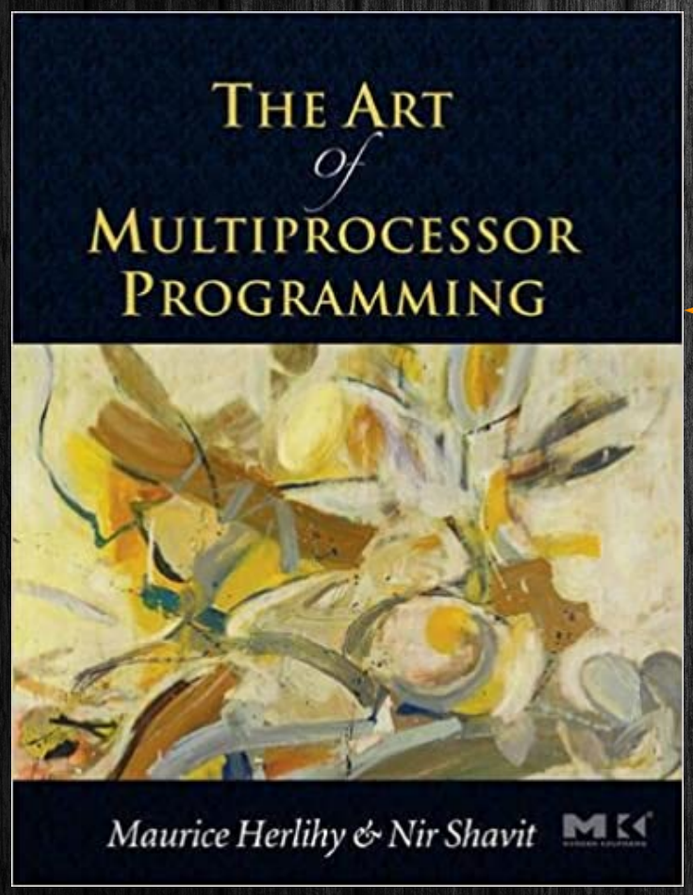
Concurrent Algorithms and Data Structures



literature

Reading list will be provided

Concurrent Algorithms and Data Structures



literature

Reading list will be provided



internet

Concurrent Algorithms and Data Structures

Concurrent Algorithms and Data Structures

Studium

Concurrent Algorithms and Data Structures

course information

- slides
- assignments
- results
- evaluation
- etc.

Studium

Concurrent Algorithms and Data Structures

Outline

Concurrent Algorithms and Data Structures

Outline

- Overview

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists

Set Algorithms

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm

Set Algorithms

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Set Algorithms

Concurrent Algorithms and Data Structures

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Outline

- **Overview**
- **Abstract Data Types**
- **Sequential Programs**
- **Concurrent Programs**
- **Coarse-Grained Locking**
- **Fine-Grained Locking**
- **Lazy Lists**
- **Optimistic Lists**
- **Treiber Stack Algorithm**
- **Michael&Scott Queue Algorithm**

Overview

- Algorithms and Data Structures
- Concurrency
- Efficiency & Correctness
- Concepts
    - Specifications
    - Safety Properties
    - Liveness Properties

Overview

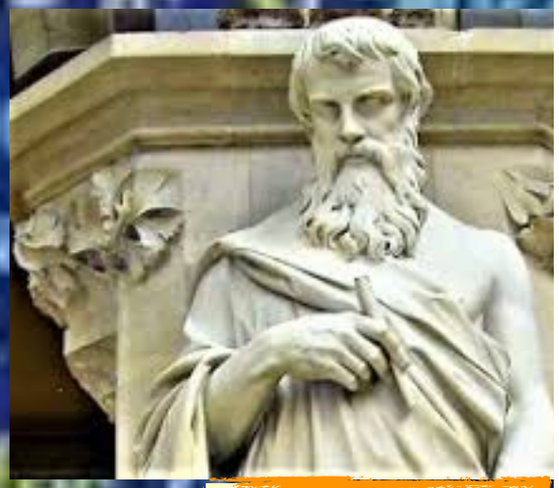
- **Algorithms and Data Structures**
- Concurrency
- Efficiency & Correctness
- Concepts
  - Specifications
  - Safety Properties
  - Liveness Properties

Algorithmics

Algorithmics | **Old Science**

Algorithmics

Old Science



Euclides

1220 mod 516 = 188

516 mod 188 = 140

188 mod 140 = 48

140 mod 48 = 44

48 mod 44 = 4

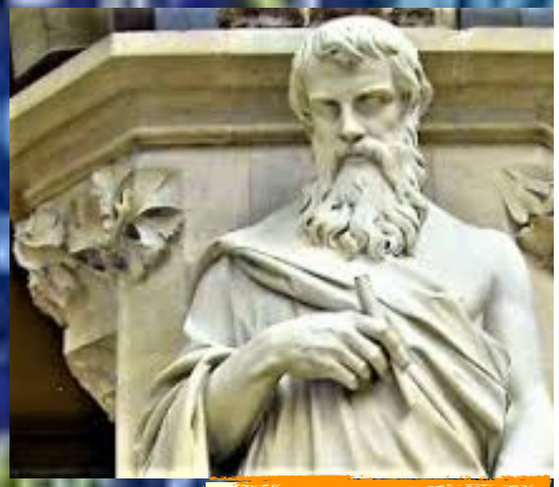
44 mod 4 = 0

4 = GCD

gcd algorithm

Algorithmics

Old Science



Euclides

1220 mod 516 = 188

516 mod 188 = 140

188 mod 140 = 48

140 mod 48 = 44

48 mod 44 = 4

44 mod 4 = 0

4 = GCD

gcd algorithm

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

In other projects
Wikimedia Commons

Languages
العربية
Español

Article Talk

Euclidean algorithm

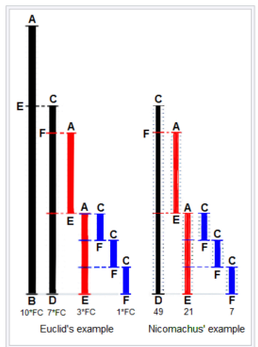
From Wikipedia, the free encyclopedia

This article is about an algorithm for the greatest common divisor. For the mathematics of space, see Euclidean geometry. For other uses of "Euclidean", see Euclidean (disambiguation).

In mathematics, the **Euclidean algorithm**,[note 1] or **Euclid's algorithm**, is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder. It is named after the ancient Greek mathematician Euclid, who first described it in his *Elements* (c. 300 BC). It is an example of an *algorithm*, a step-by-step procedure for performing a calculation according to well-defined rules, and is one of the oldest algorithms in common use. It can be used to reduce fractions to their simplest form, and is a part of many other number-theoretic and cryptographic calculations.

The Euclidean algorithm is based on the principle that the greatest common divisor of two numbers does not change if the larger number is replaced by its difference with the smaller number. For example, 21 is the GCD of 252 and 105 (as 252 = 21 × 12 and 105 = 21 × 5), and the same number 21 is also the GCD of 105 and 252 − 105 = 147. Since this replacement reduces the larger of the two numbers, repeating this process gives successively smaller pairs of numbers until the two numbers become equal. When that occurs, they are the GCD of the original two numbers. By reversing the steps or using the extended Euclidean algorithm, the GCD can be expressed as a linear combination of the two original numbers, that is the sum of the two numbers, each multiplied by an integer (for example, 21 = 5 × 105 + (−2) × 252. The fact that the GCD can always be expressed in this way is known as Bézout's identity.

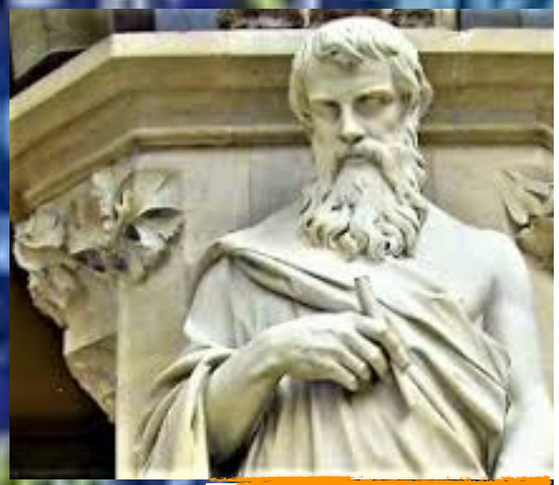
The version of the Euclidean algorithm described above (and by Euclid) can take many subtraction steps to find the GCD when one of the given numbers is much bigger than the other. A more efficient version of the algorithm shortcuts these steps, instead replacing the larger of the two numbers by its remainder when divided by the smaller of the two (with this version, the algorithm stops when reaching a zero remainder). With this improvement, the algorithm never requires more steps than five times the number of digits (base 10) of the smaller integer. This was proven by Gabriel Lamé in 1844, and marks the beginning of computational complexity theory. Additional methods for



Euclid's method for finding the greatest common divisor (GCD) of two starting lengths BA and DC, both defined to be multiples of a common "unit" length. The length DC being shorter, it is used to "measure" BA, but only once because remainder EA is less than DC. EA now measures (twice) the shorter length DC, with remainder FC shorter than EA. Then FC measures (three times) length EA. Because there is no remainder, the process ends with FC being the GCD. On the right Nicomachus's example with numbers 49 and 21 resulting in their

Algorithmics

Old Science



Euclides

1220 mod 516 = 188

516 mod 188 = 140

188 mod 140 = 48

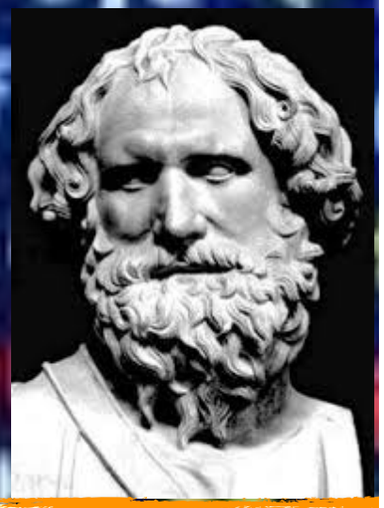
140 mod 48 = 44

48 mod 44 = 4

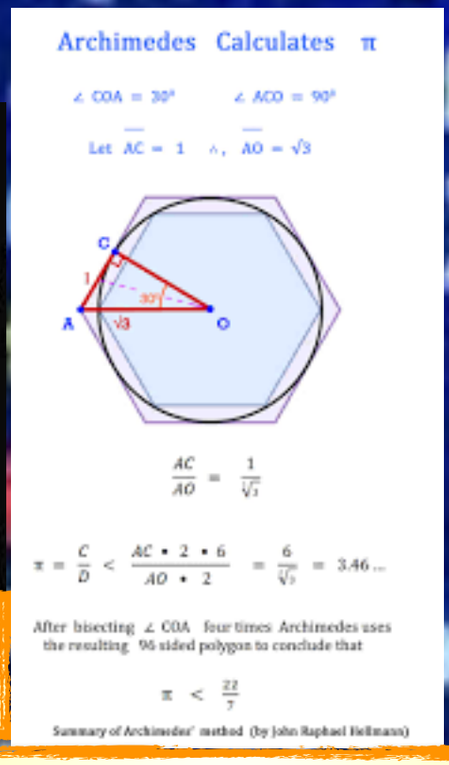
44 mod 4 = 0

4 = GCD

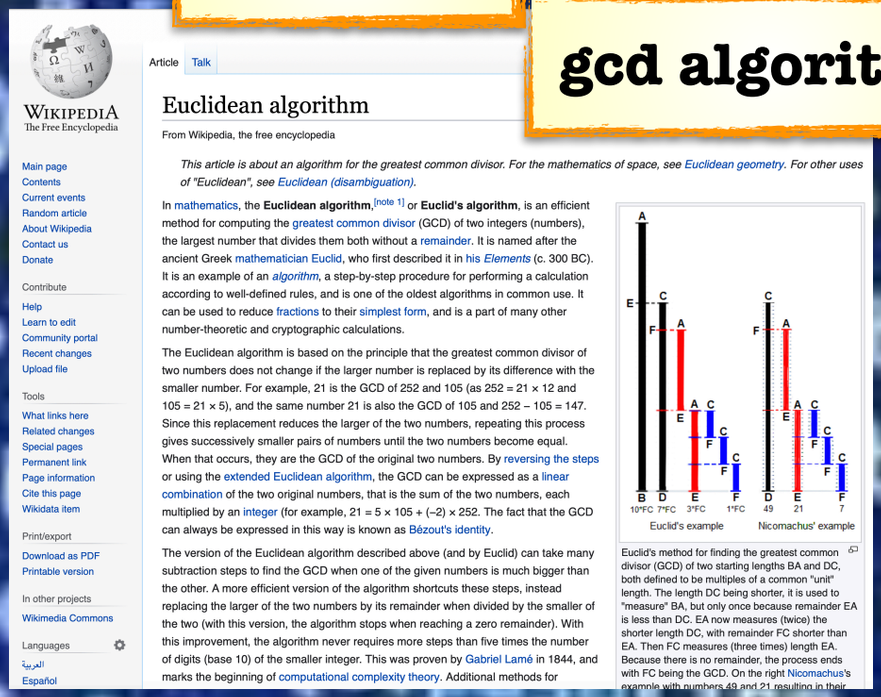
gcd algorithm



Archimedes



π-approximation



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

In other projects
Wikimedia Commons

Languages
العربية
Español

Article Talk

Euclidean algorithm

From Wikipedia, the free encyclopedia

This article is about an algorithm for the greatest common divisor. For the mathematics of space, see Euclidean geometry. For other uses of "Euclidean", see Euclidean (disambiguation).

In mathematics, the **Euclidean algorithm**,[note 1] or **Euclid's algorithm**, is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder. It is named after the ancient Greek mathematician Euclid, who first described it in his *Elements* (c. 300 BC). It is an example of an *algorithm*, a step-by-step procedure for performing a calculation according to well-defined rules, and is one of the oldest algorithms in common use. It can be used to reduce fractions to their simplest form, and is a part of many other number-theoretic and cryptographic calculations.

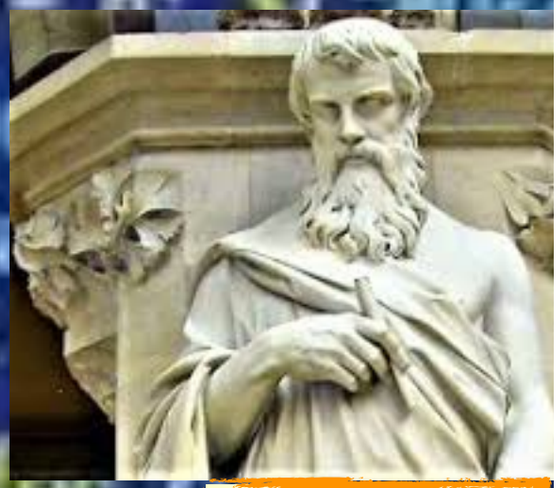
The Euclidean algorithm is based on the principle that the greatest common divisor of two numbers does not change if the larger number is replaced by its difference with the smaller number. For example, 21 is the GCD of 252 and 105 (as 252 = 21 × 12 and 105 = 21 × 5), and the same number 21 is also the GCD of 105 and 252 − 105 = 147. Since this replacement reduces the larger of the two numbers, repeating this process gives successively smaller pairs of numbers until the two numbers become equal. When that occurs, they are the GCD of the original two numbers. By reversing the steps or using the extended Euclidean algorithm, the GCD can be expressed as a linear combination of the two original numbers, that is the sum of the two numbers, each multiplied by an integer (for example, 21 = 5 × 105 + (−2) × 252. The fact that the GCD can always be expressed in this way is known as Bézout's identity.

The version of the Euclidean algorithm described above (and by Euclid) can take many subtraction steps to find the GCD when one of the given numbers is much bigger than the other. A more efficient version of the algorithm shortcuts these steps, instead replacing the larger of the two numbers by its remainder when divided by the smaller of the two (with this version, the algorithm stops when reaching a zero remainder). With this improvement, the algorithm never requires more steps than five times the number of digits (base 10) of the smaller integer. This was proven by Gabriel Lamé in 1844, and marks the beginning of computational complexity theory. Additional methods for

Euclid's method for finding the greatest common divisor (GCD) of two starting lengths BA and DC, both defined to be multiples of a common "unit" length. The length DC being shorter, it is used to "measure" BA, but only once because remainder EA is less than DC. EA now measures (twice) the shorter length DC, with remainder FC shorter than EA. Then FC measures (three times) length EA. Because there is no remainder, the process ends with FC being the GCD. On the right Nicomachus's example with numbers 49 and 21 resulting in their

Algorithmics

Old Science



Euclides

1220 mod 516 = 188

516 mod 188 = 140

188 mod 140 = 48

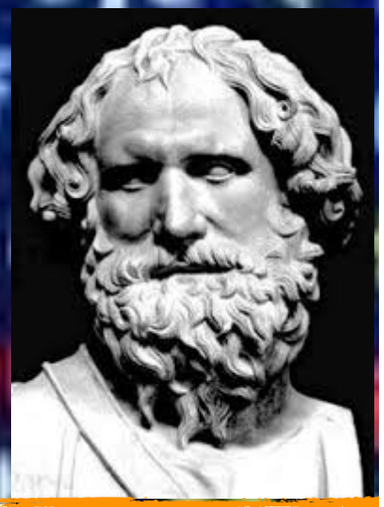
140 mod 48 = 44

48 mod 44 = 4

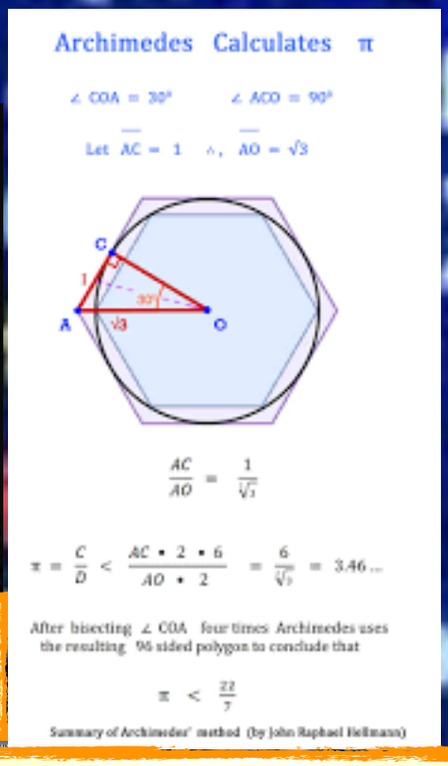
44 mod 4 = 0

4 = GCD

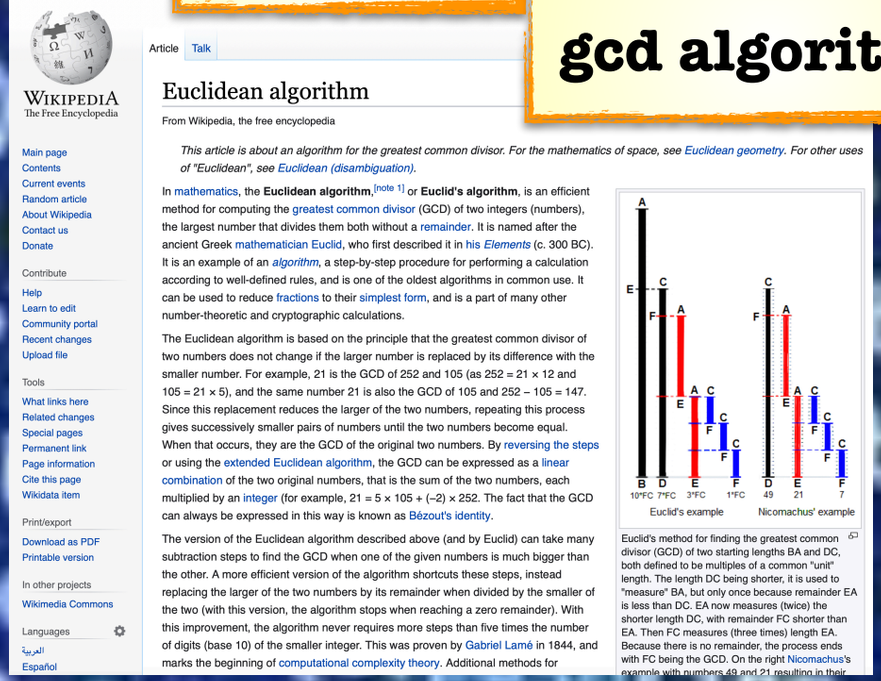
gcd algorithm



Archimedes



π-approximation



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

In other projects
Wikimedia Commons

Languages
العربية
Español

Article Talk

Euclidean algorithm

From Wikipedia, the free encyclopedia

This article is about an algorithm for the greatest common divisor. For the mathematics of space, see Euclidean geometry. For other uses of "Euclidean", see Euclidean (disambiguation).

In mathematics, the **Euclidean algorithm**,[note 1] or **Euclid's algorithm**, is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder. It is named after the ancient Greek mathematician Euclid, who first described it in his *Elements* (c. 300 BC). It is an example of an *algorithm*, a step-by-step procedure for performing a calculation according to well-defined rules, and is one of the oldest algorithms in common use. It can be used to reduce fractions to their simplest form, and is a part of many other number-theoretic and cryptographic calculations.

The Euclidean algorithm is based on the principle that the greatest common divisor of two numbers does not change if the larger number is replaced by its difference with the smaller number. For example, 21 is the GCD of 252 and 105 (as 252 = 21 × 12 and 105 = 21 × 5), and the same number 21 is also the GCD of 105 and 252 − 105 = 147. Since this replacement reduces the larger of the two numbers, repeating this process gives successively smaller pairs of numbers until the two numbers become equal. When that occurs, they are the GCD of the original two numbers. By reversing the steps or using the extended Euclidean algorithm, the GCD can be expressed as a linear combination of the two original numbers, that is the sum of the two numbers, each multiplied by an integer (for example, 21 = 5 × 105 + (−2) × 252. The fact that the GCD can always be expressed in this way is known as Bézout's identity.

The version of the Euclidean algorithm described above (and by Euclid) can take many subtraction steps to find the GCD when one of the given numbers is much bigger than the other. A more efficient version of the algorithm shortcuts these steps, instead replacing the larger of the two numbers by its remainder when divided by the smaller of the two (with this version, the algorithm stops when reaching a zero remainder). With this improvement, the algorithm never requires more steps than five times the number of digits (base 10) of the smaller integer. This was proven by Gabriel Lamé in 1844, and marks the beginning of computational complexity theory. Additional methods for

Euclid's method for finding the greatest common divisor (GCD) of two starting lengths BA and DC, both defined to be multiples of a common "unit" length. The length DC being shorter, it is used to "measure" BA, but only once because remainder EA is less than DC. EA now measures (twice) the shorter length DC, with remainder FC shorter than EA. Then FC measures (three times) length EA. Because there is no remainder, the process ends with FC being the GCD. On the right Nicomachus's example with numbers 49 and 21 resulting in their



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

In other projects
Wikimedia Commons
Wikiquote

Languages
Afrikaans

Article Talk Read View source View history

Pi

From Wikipedia, the free encyclopedia

This article is about the mathematical constant. For the Greek letter, see Pi (letter). For other uses, see PI (disambiguation).

The number **π** (/paɪ/) is a mathematical constant. It is defined as the ratio of a circle's circumference to its diameter, and it also has various equivalent definitions. It appears in many formulas in all areas of mathematics and physics. It is approximately equal to 3.14159. It has been represented by the Greek letter "π" since the mid-18th century, and is spelled out as "**pi**". It is also referred to as **Archimedes' constant**.[1][2][3]

Being an irrational number, π cannot be expressed as a common fraction, although fractions such as 22/7 are commonly used to approximate it. Equivalently, its decimal representation never ends and never settles into a permanently repeating pattern. Its decimal (or other base) digits appear to be randomly distributed, and are conjectured to satisfy a specific kind of statistical randomness.

It is known that π is a transcendental number:[2] it is not the root of any polynomial with rational coefficients. The transcendence of π implies that it is impossible to solve the ancient challenge of squaring the circle with a compass and straightedge.

Ancient civilizations, including the Egyptians and Babylonians, required fairly accurate approximations of π for practical computations. Around 250 BC, the Greek mathematician Archimedes created an algorithm to approximate π with arbitrary accuracy. In the 5th century AD, Chinese mathematics approximated π to seven digits, while Indian mathematics made a five-digit approximation, both using geometrical techniques. The first exact formula for π, based on infinite series, was discovered a millennium later, when in the 14th century the Madhava–Leibniz series was discovered in Indian mathematics.[4][5]

The invention of calculus soon led to the calculation of hundreds of digits of π, enough for all practical scientific computations. Nevertheless, in the 20th and 21st centuries, mathematicians and computer scientists have pursued new approaches that, when combined with increasing computational power, extended the decimal representation of π to many trillions of digits.[6][7] The primary motivation for these computations is as a test case to develop efficient algorithms to calculate numeric series, as well as the quest to break records.[8][9] The extensive calculations involved have also been used to test supercomputers and high-precision multiplication algorithms.

Part of a series of articles on the
mathematical constant π



3.14159 26535 89793 23846 26433...

Uses
Area of a circle · Circumference · Use in other formulae

Properties
Irrationality · Transcendence

Value
Less than 22/7 · Approximations · Memorization

People
Archimedes · Liu Hui · Zu Chongzhi · Aryabhata · Madhava · Ludolph van Ceulen · Seki Takakazu · Takebe Kenko · William Jones · John Machin · William Shanks · Srinivasa Ramanujan · John Wrench · Chudnovsky brothers · Yasumasa Kanada

History
Chronology · Book

In culture
Legislation · Pi Day

Related topics
Squaring the circle · Basel problem · Six nines in π · Other topics related to π

v · t · e

Algorithmics | **Old Science**

Algorithmics

Old Science

- Implement an algorithms as a program
- Run them on a computer (system)

new

Algorithmics

Old Science

- Implement an algorithms as a program
- Run them on a computer (system)

new

basis of computer science

Algorithmics

Old Science

- Implement an algorithms as a program
- Run them on a computer (system)

new

basis of computer science

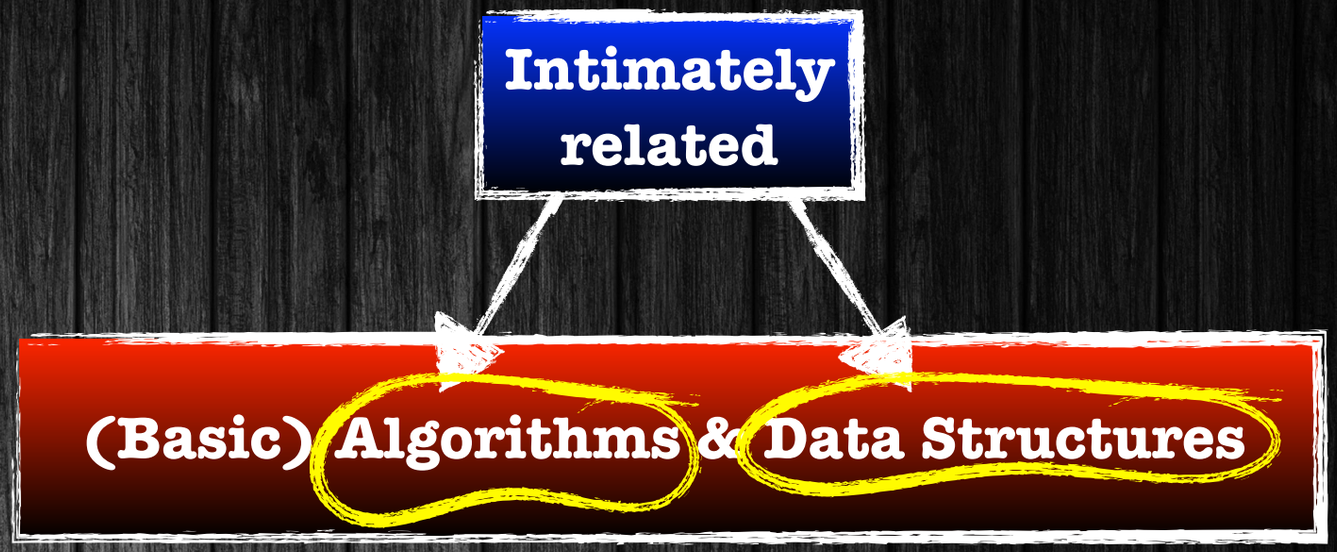
hardware

software

Overview

(Basic) Algorithms & Data Structures

Overview



Overview

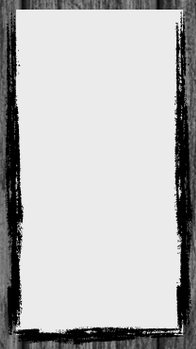
(Basic) Algorithms & Data Structures

Overview

Example | Stack | (Basic) Algorithms & Data Structures

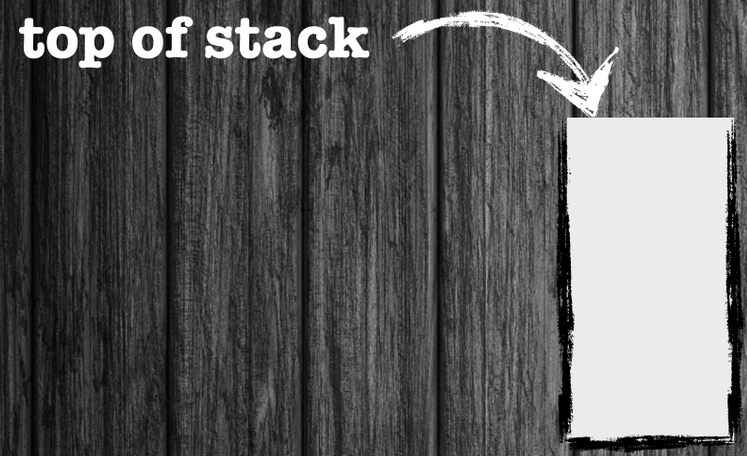
Overview

Example Stack (Basic) Algorithms & Data Structures



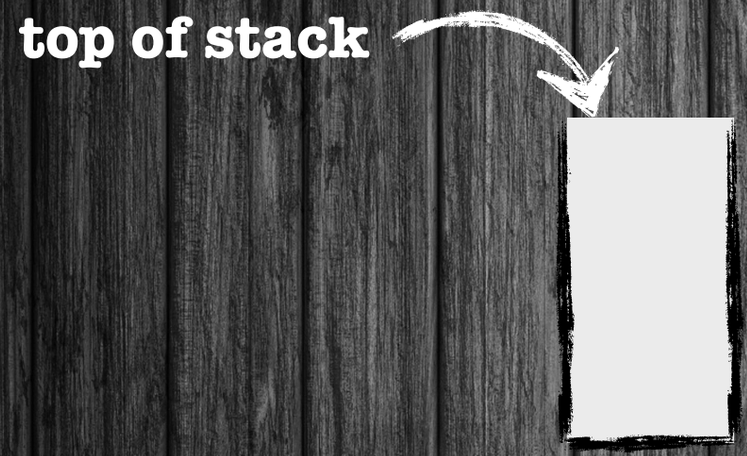
Overview

Example | Stack | (Basic) Algorithms & Data Structures



Overview

Example | Stack | (Basic) Algorithms & Data Structures



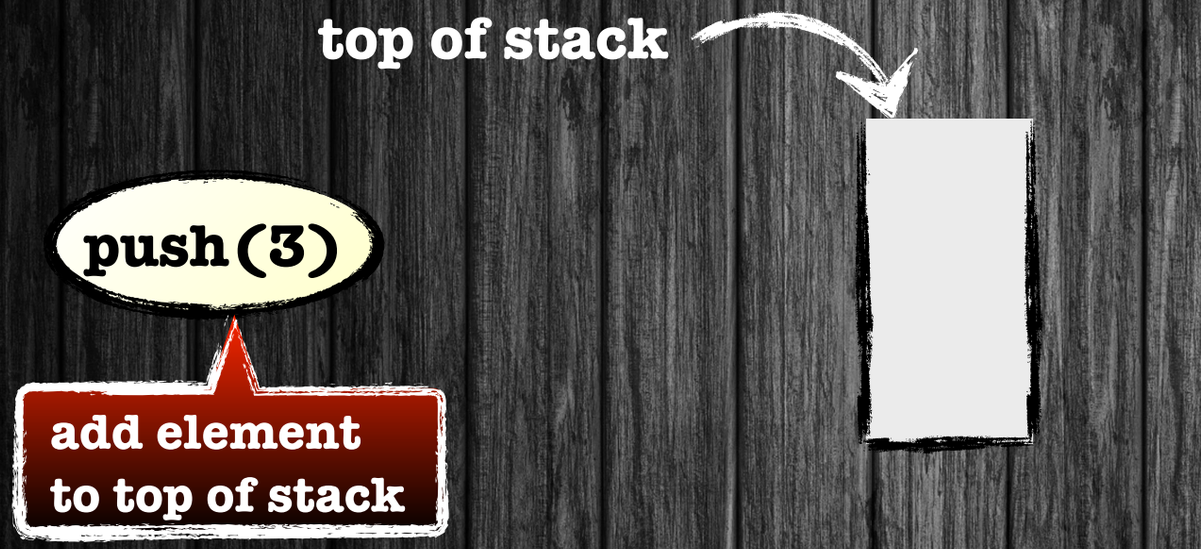
Overview

Example | Stack | (Basic) Algorithms & Data Structures



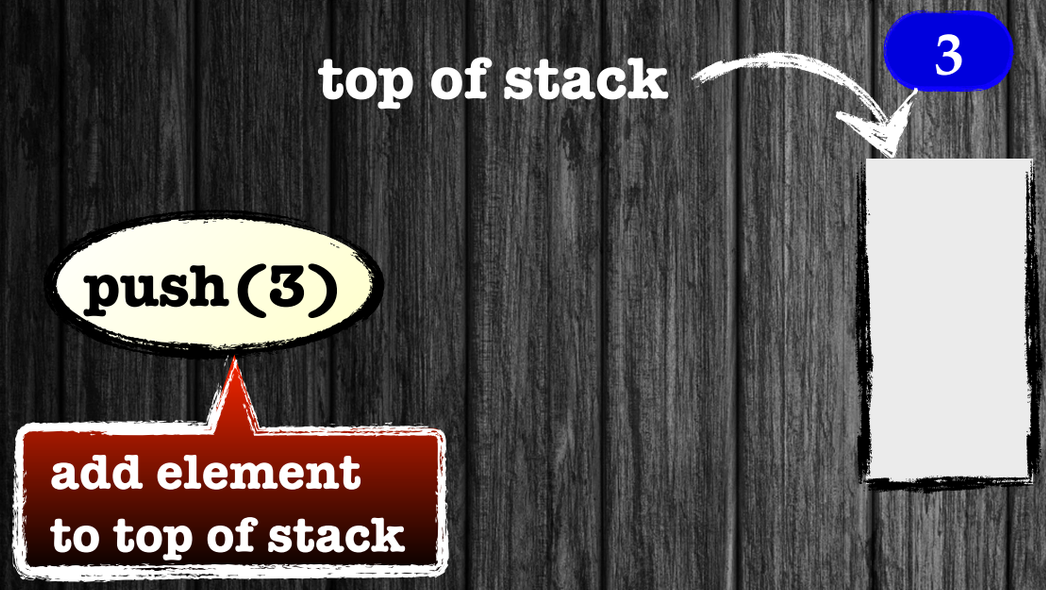
Overview

Example Stack (Basic) Algorithms & Data Structures



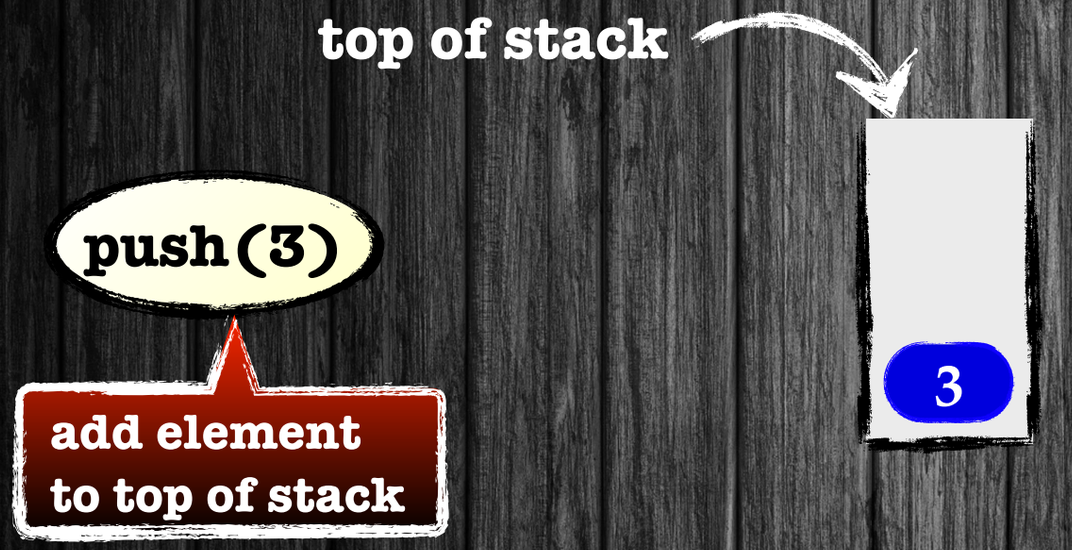
Overview

Example Stack (Basic) Algorithms & Data Structures



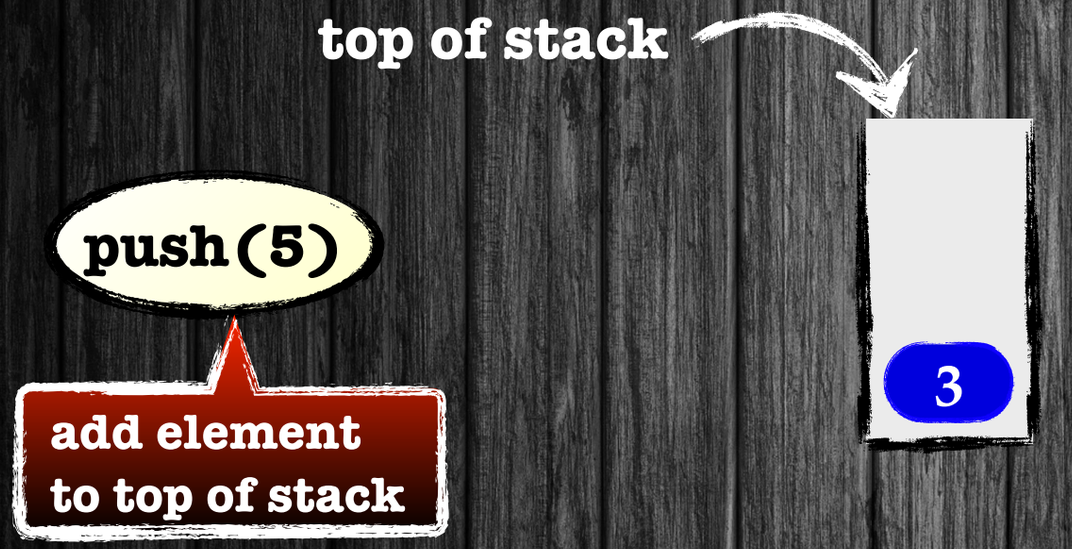
Overview

Example | Stack | (Basic) Algorithms & Data Structures



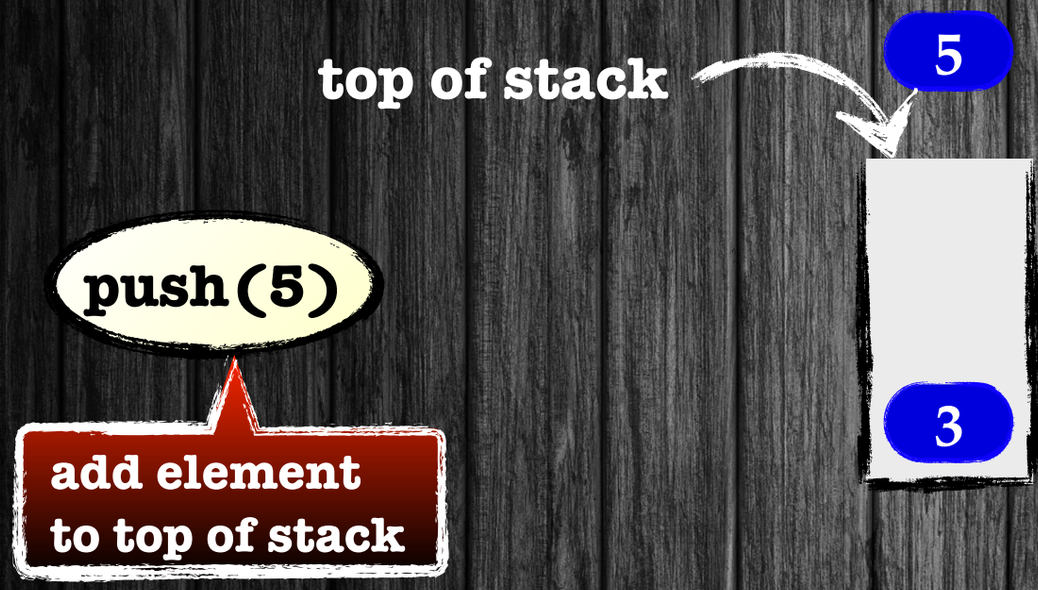
Overview

Example | Stack | (Basic) Algorithms & Data Structures



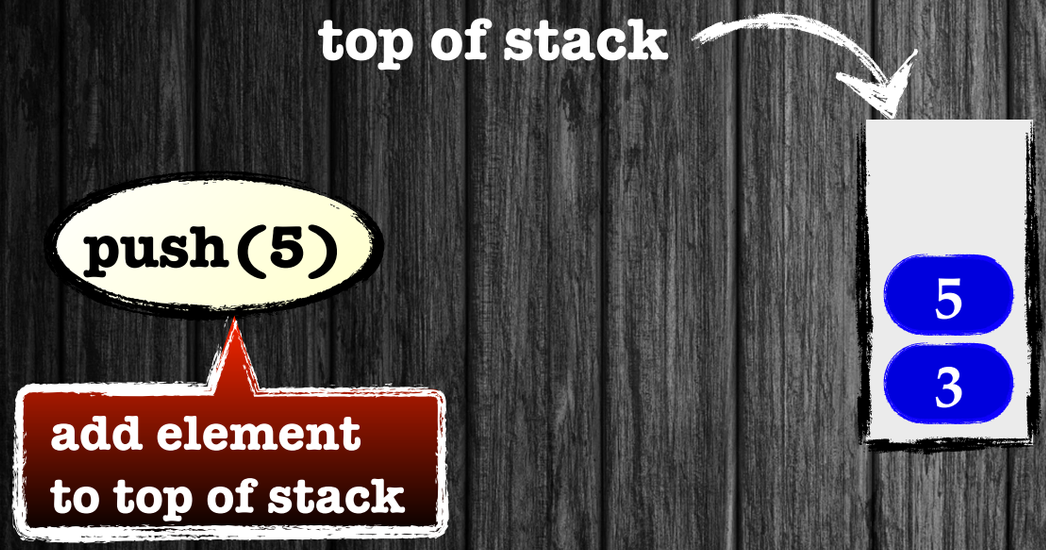
Overview

Example | Stack | (Basic) Algorithms & Data Structures



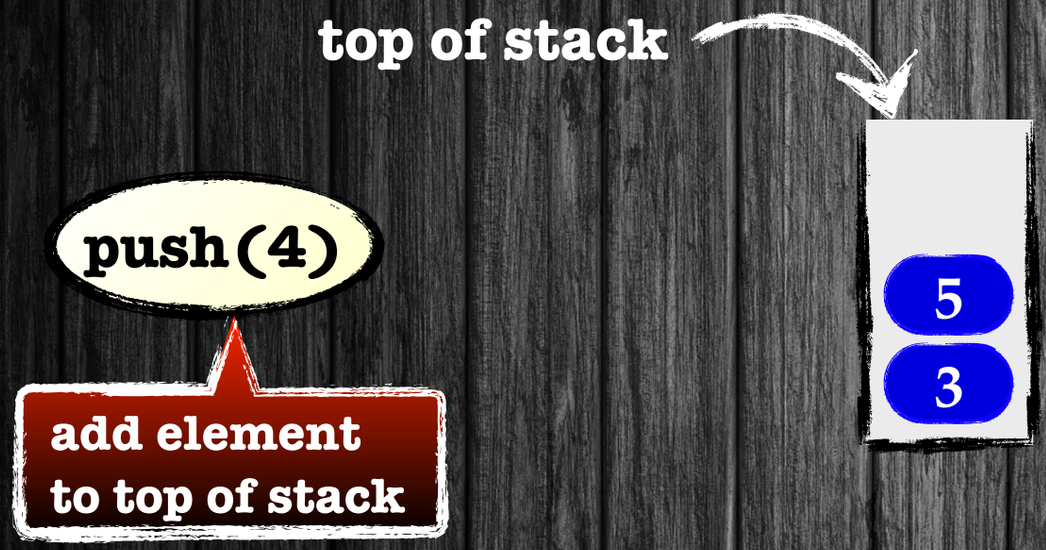
Overview

Example | Stack | (Basic) Algorithms & Data Structures



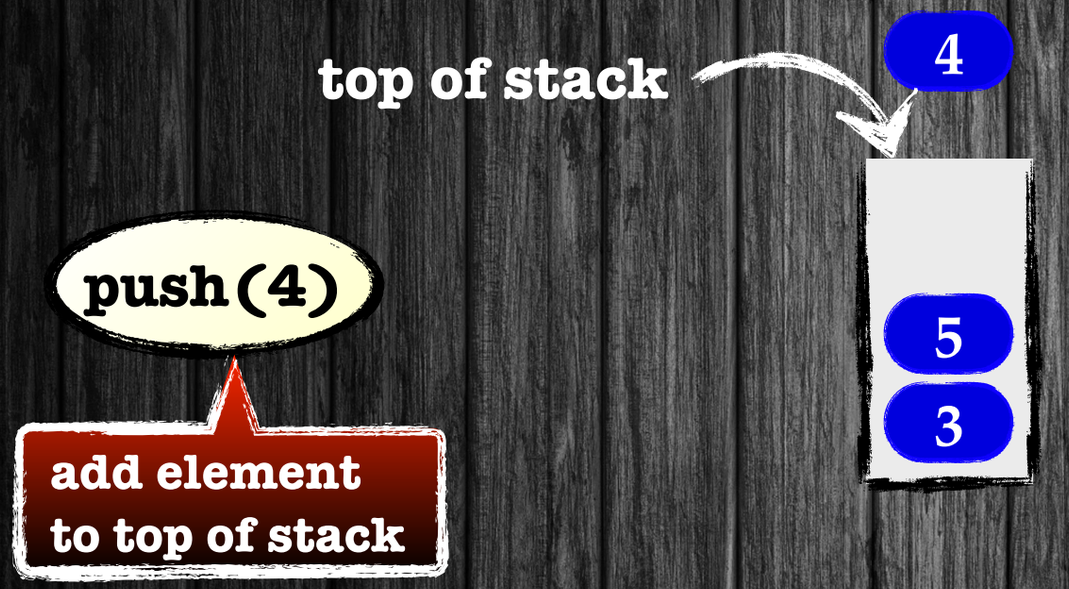
Overview

Example | Stack | (Basic) Algorithms & Data Structures



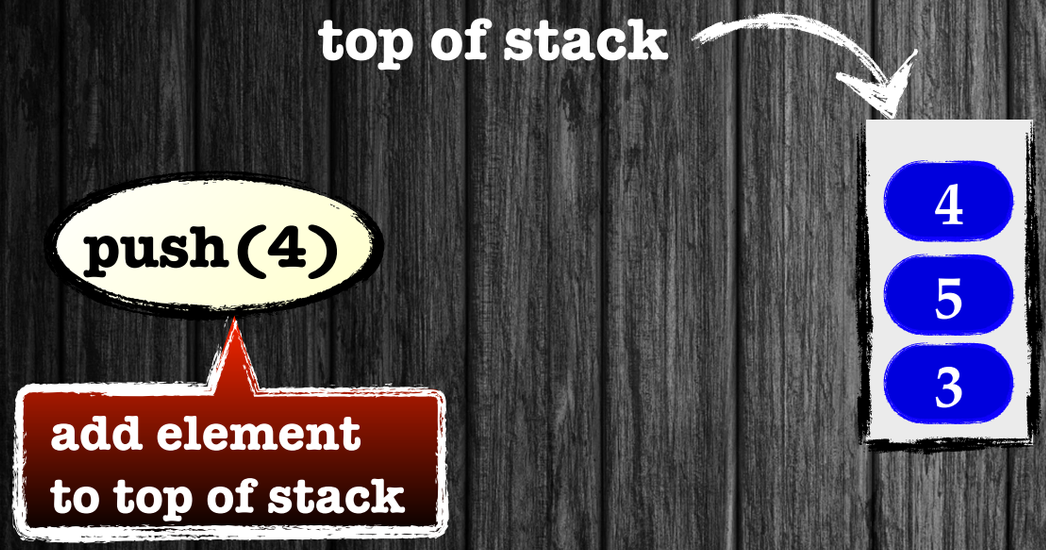
Overview

Example | Stack | (Basic) Algorithms & Data Structures



Overview

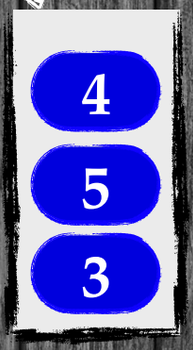
Example Stack (Basic) Algorithms & Data Structures



Overview

Example Stack (Basic) Algorithms & Data Structures

top of stack

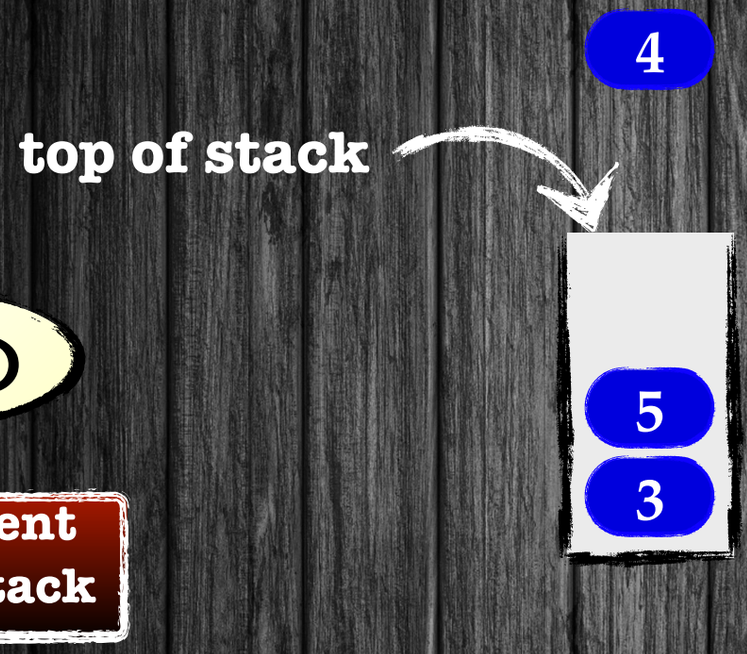


pop(4)

remove element
from top of stack

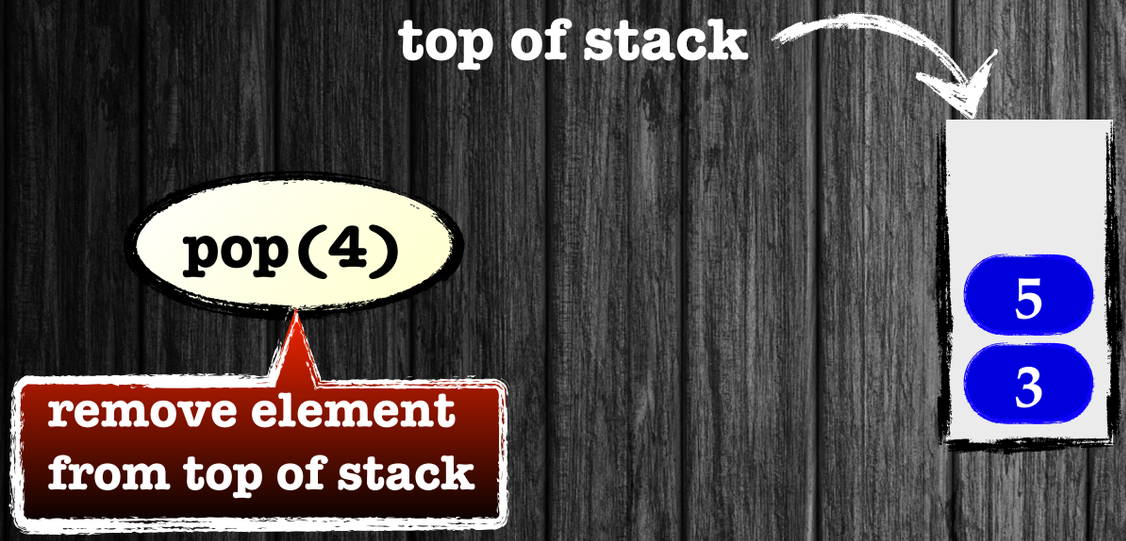
Overview

Example | Stack | (Basic) Algorithms & Data Structures



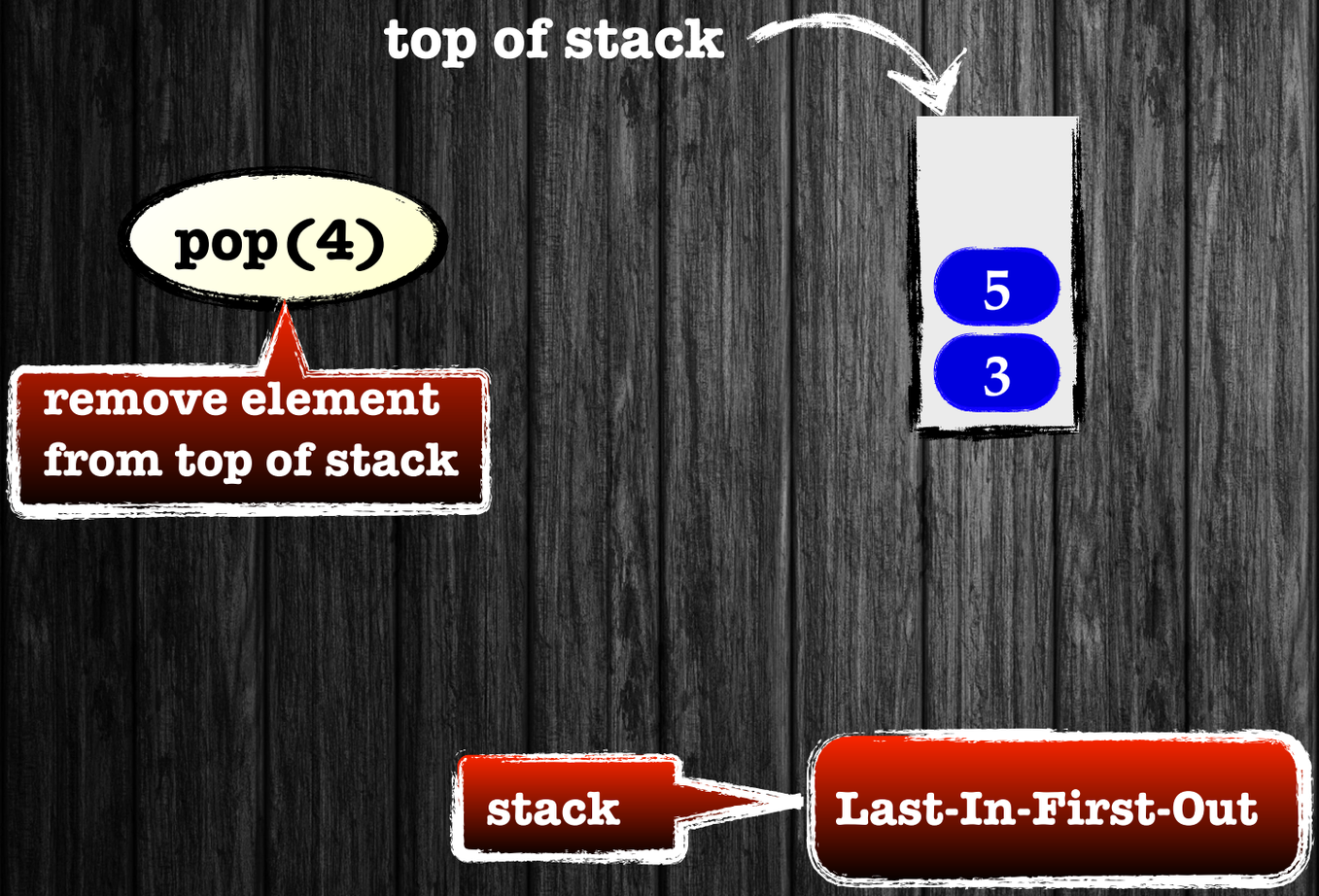
Overview

Example | Stack | (Basic) Algorithms & Data Structures



Overview

Example | Stack | (Basic) Algorithms & Data Structures



Overview

- Algorithms and Data Structures
- **Concurrency**
- Algorithmics
- Efficiency & Correctness
- Concepts
  - Specifications
  - Safety Properties
  - Liveness Properties

Algorithmics | **Concurrency**

Algorithmics | Concurrency

motivation

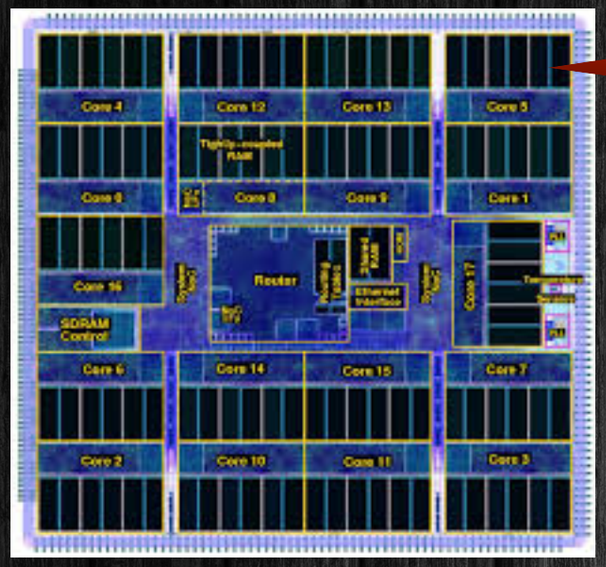
Algorithmics | Concurrency

motivation

Concurrent systems are everywhere

Algorithmics

Concurrency



Multicore architectures

motivation

Concurrent systems are everywhere

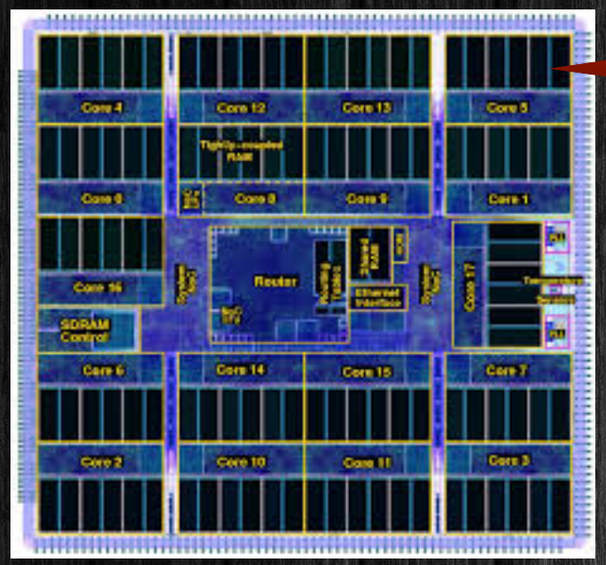
Algorithmics

Concurrency

Concurrent systems are everywhere

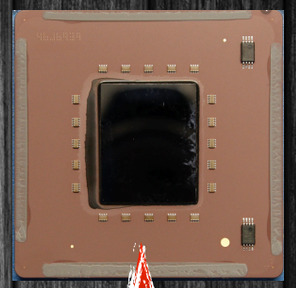
Multicore architectures

motivation













intel

ARM

IBM Power

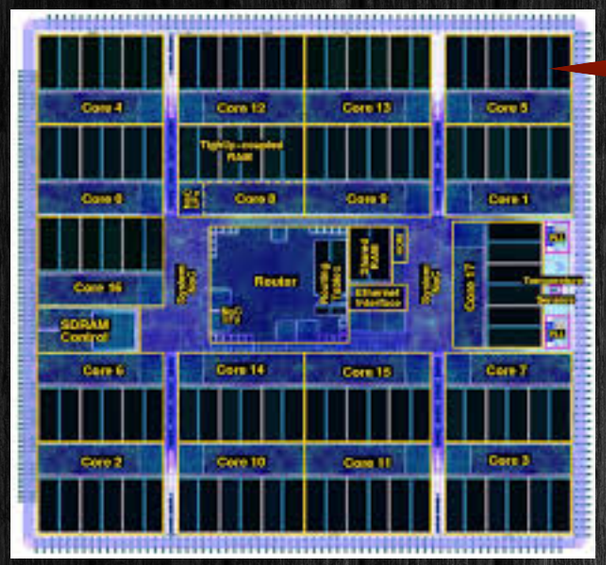
Algorithmics

Concurrency

Concurrent systems are everywhere

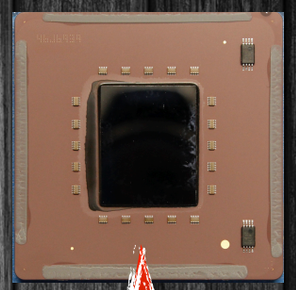
Multicore architectures

motivation









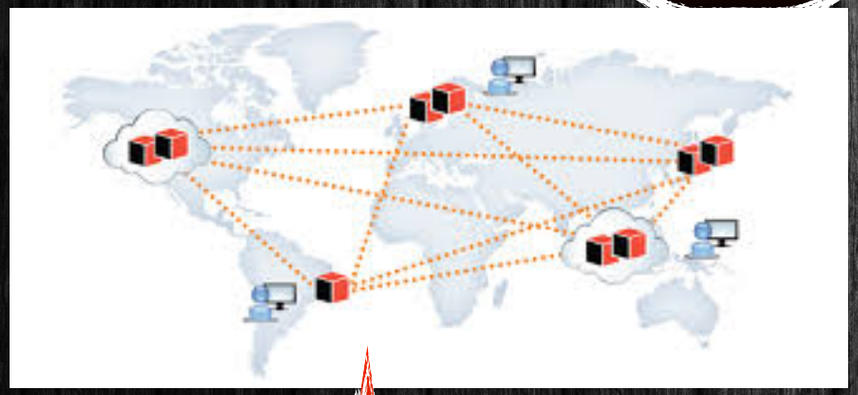




intel

ARM

IBM Power



Distributed databases

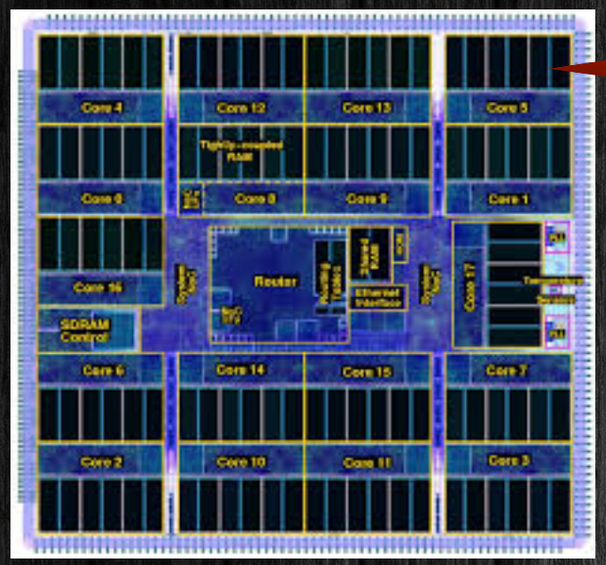
Algorithmics

Concurrency

Concurrent systems are everywhere

Multicore architectures

motivation













intel

ARM

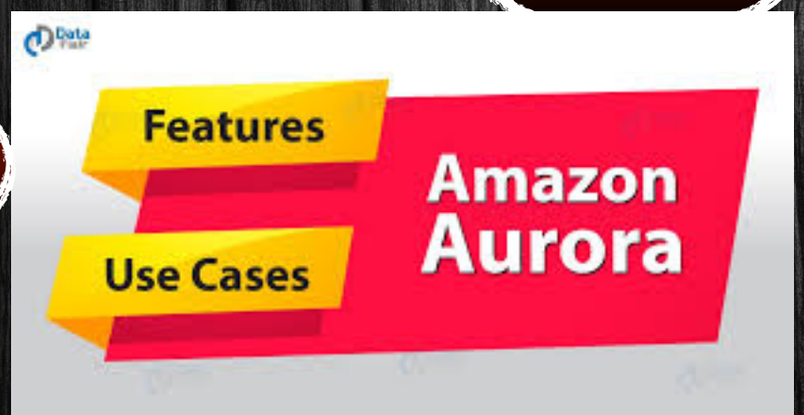
IBM Power



Distributed databases



Facebook



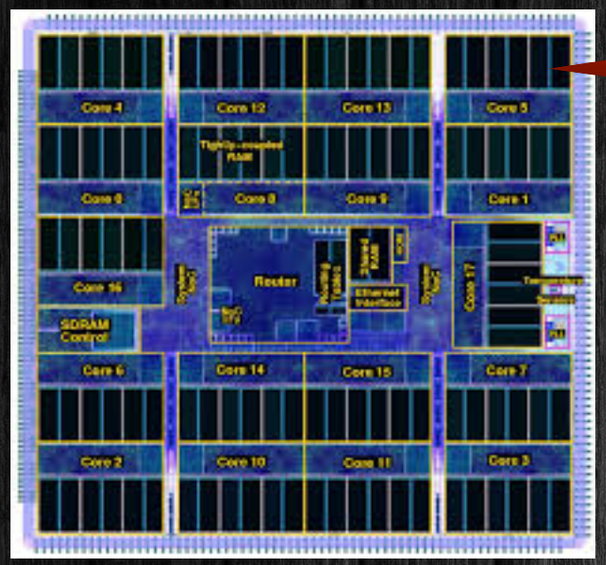
Algorithmics

Concurrency

Concurrent systems are everywhere

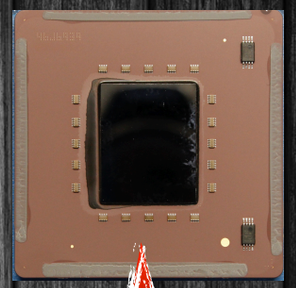
Multicore architectures

motivation









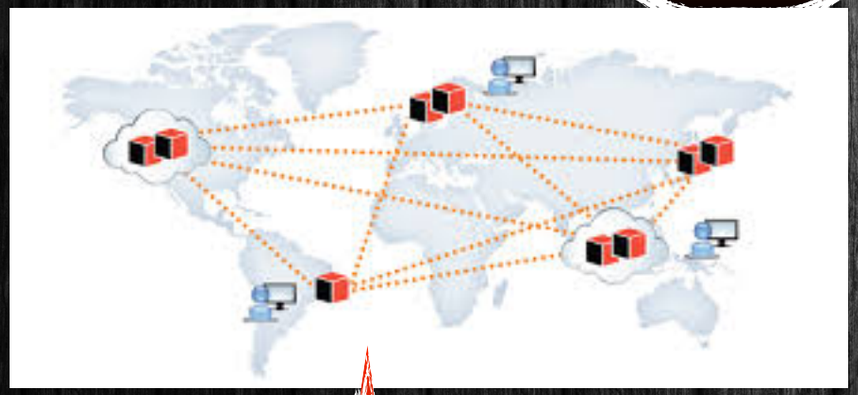




intel

ARM

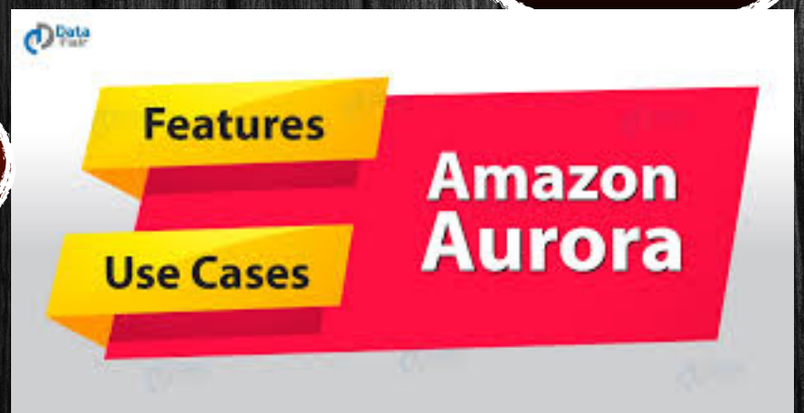
IBM Power



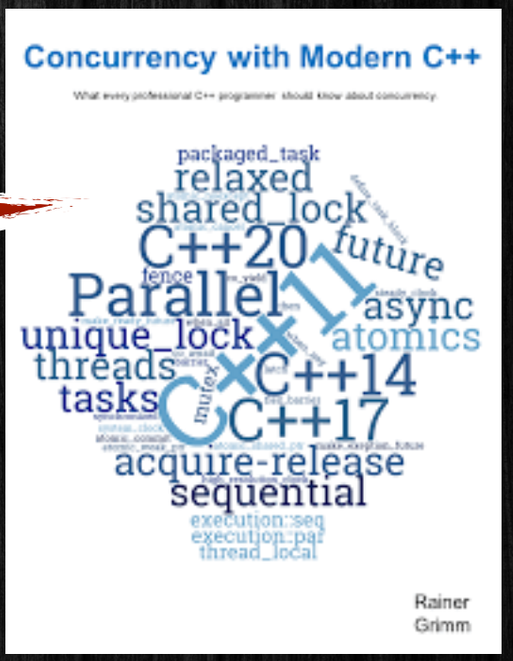
Distributed databases



Facebook

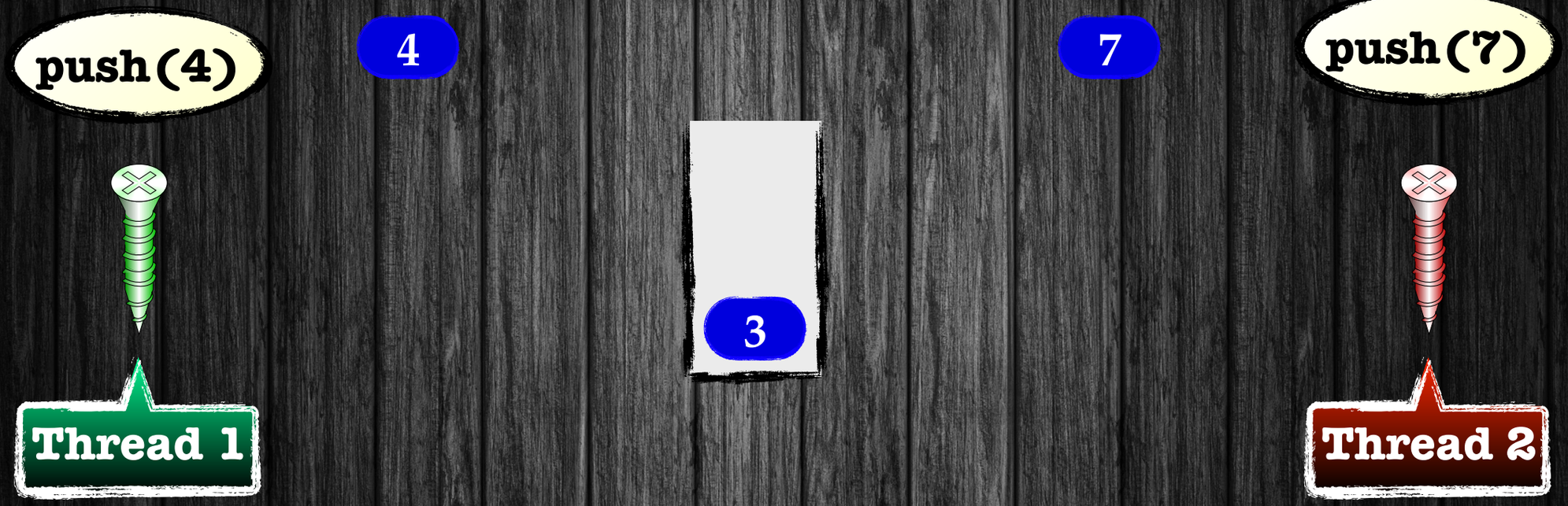


Programming languages



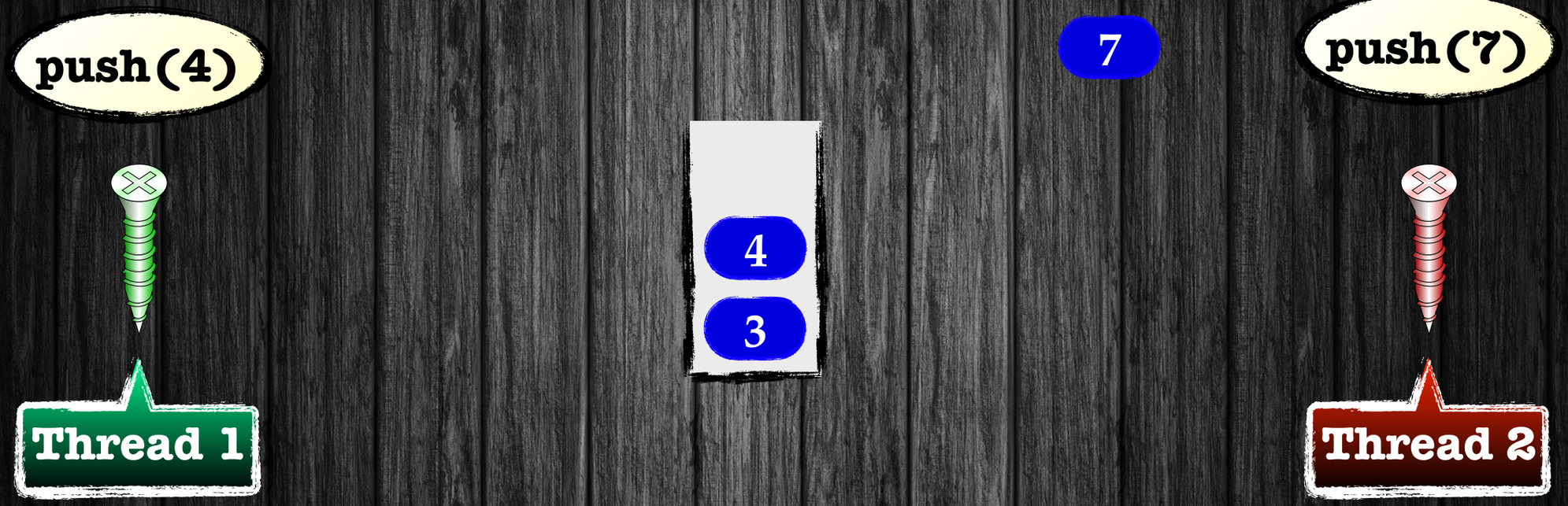
Overview | Goal → concurrent setting

Example | Stack | (Basic) Algorithms & Data Structures



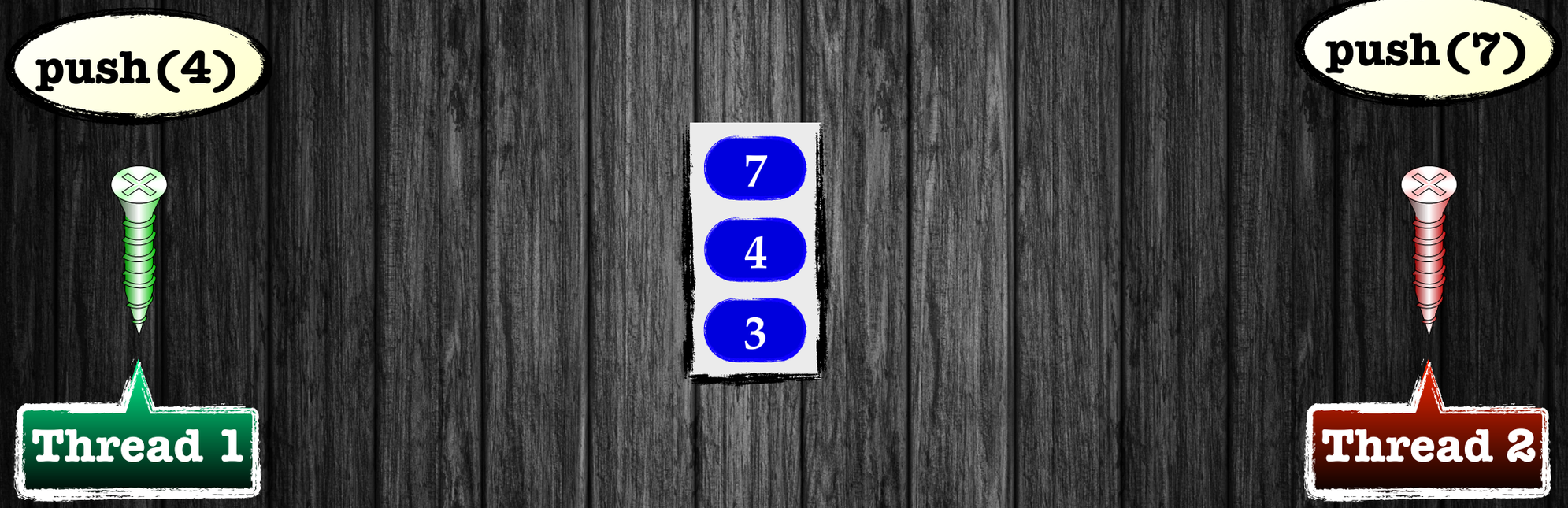
Overview | Goal → concurrent setting

Example | Stack | (Basic) Algorithms & Data Structures



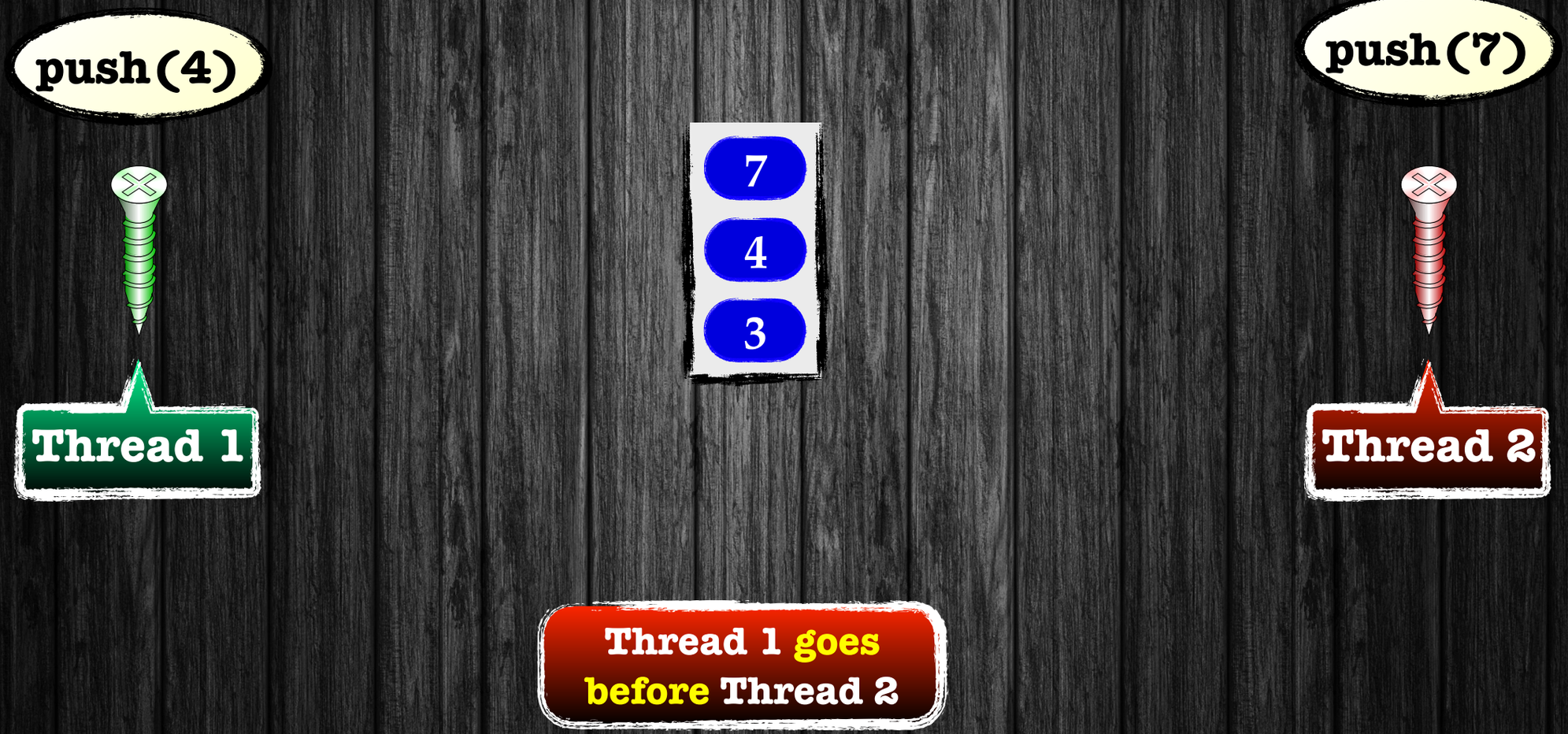
Overview | Goal → concurrent setting

Example | Stack | (Basic) Algorithms & Data Structures



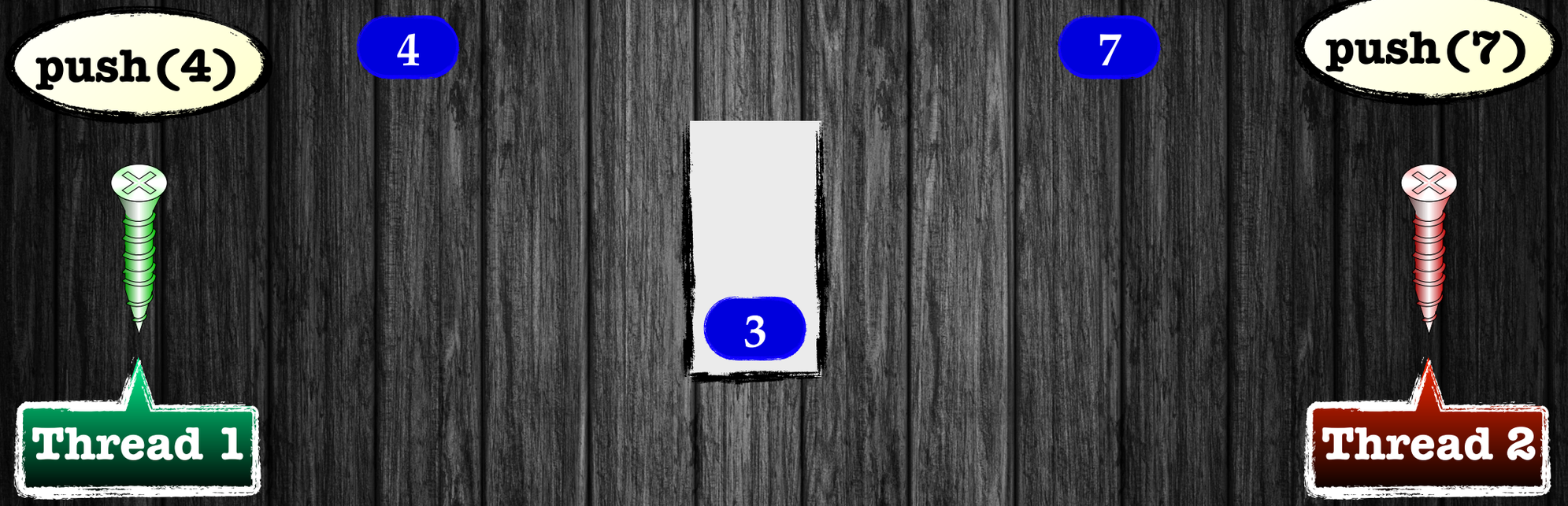
Overview Goal → concurrent setting

Example Stack (Basic) Algorithms & Data Structures



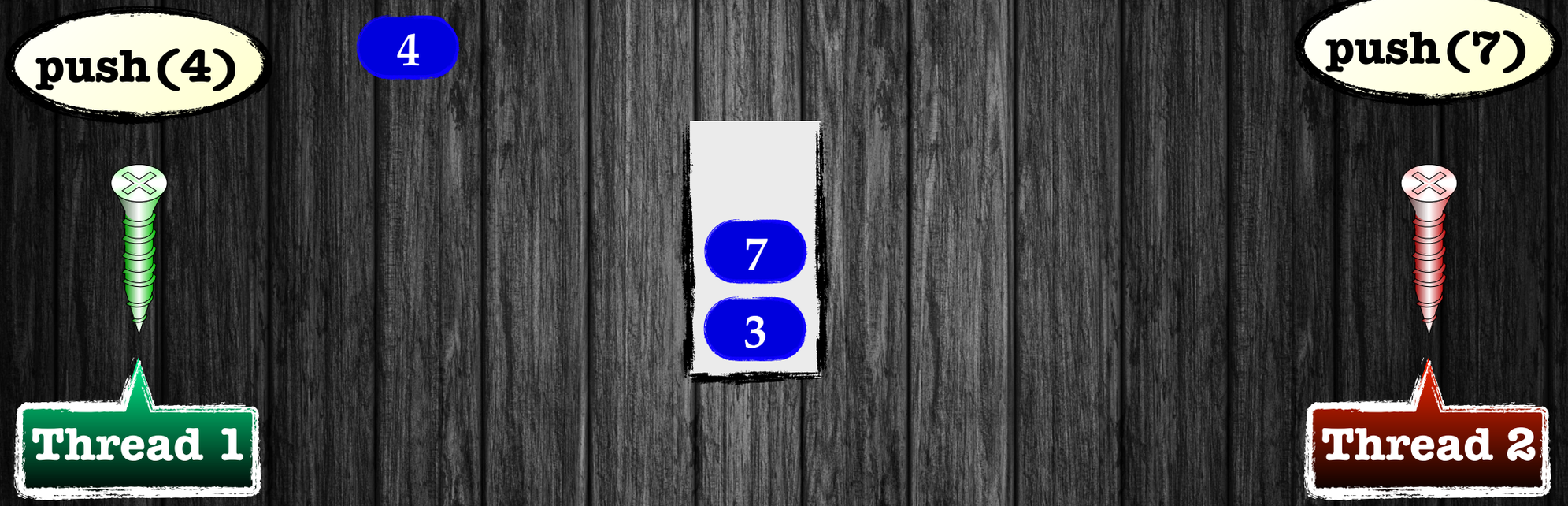
Overview | Goal → concurrent setting

Example | Stack | (Basic) Algorithms & Data Structures



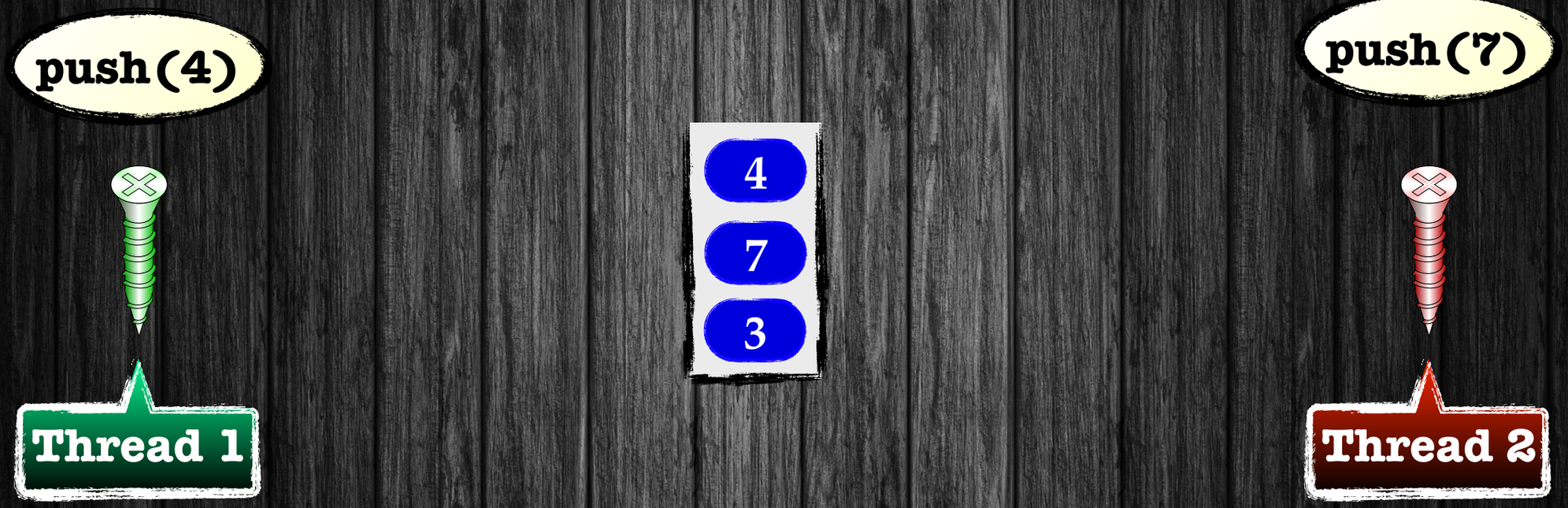
Overview | Goal → concurrent setting

Example | Stack | (Basic) Algorithms & Data Structures



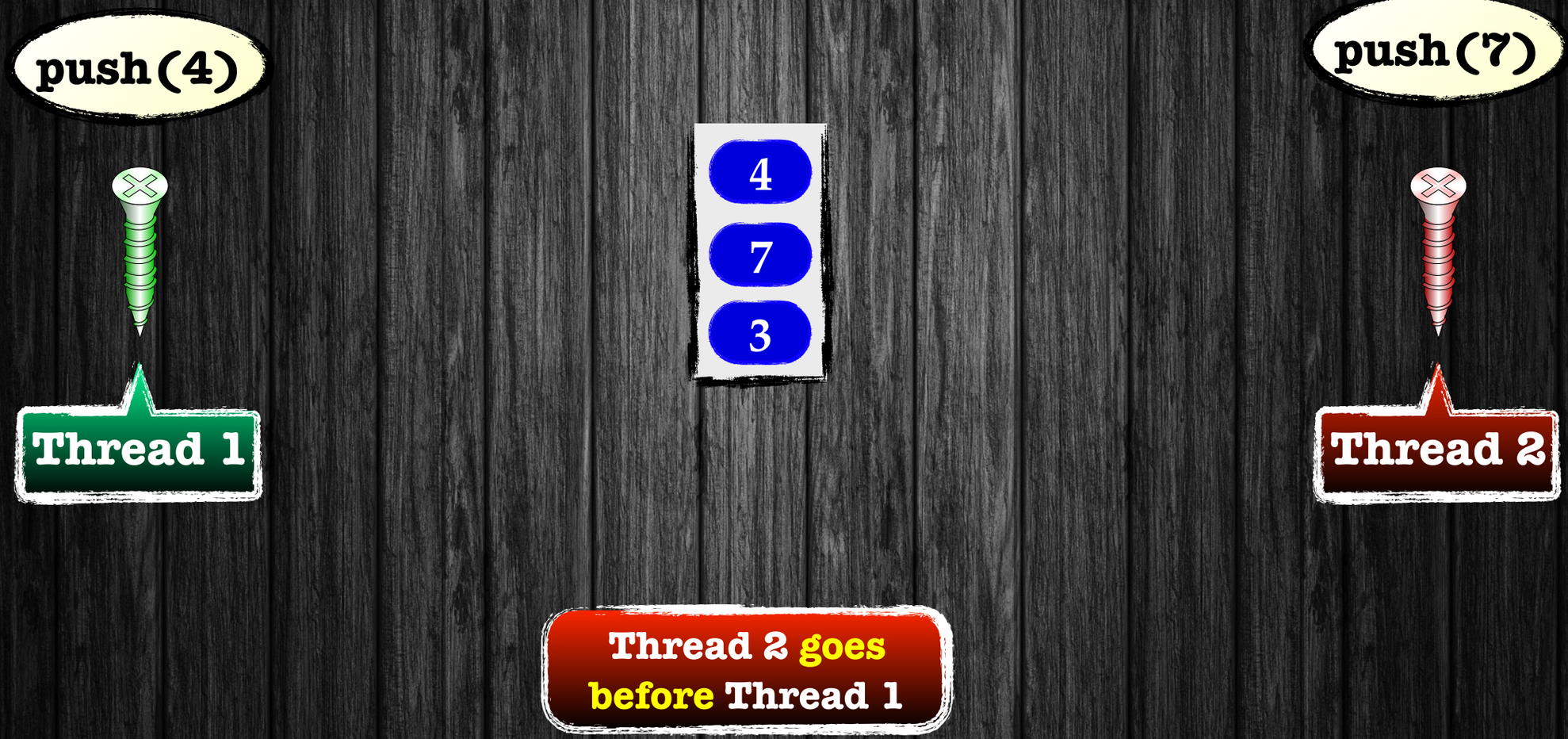
Overview **Goal** → **concurrent setting**

Example **Stack** **(Basic) Algorithms & Data Structures**



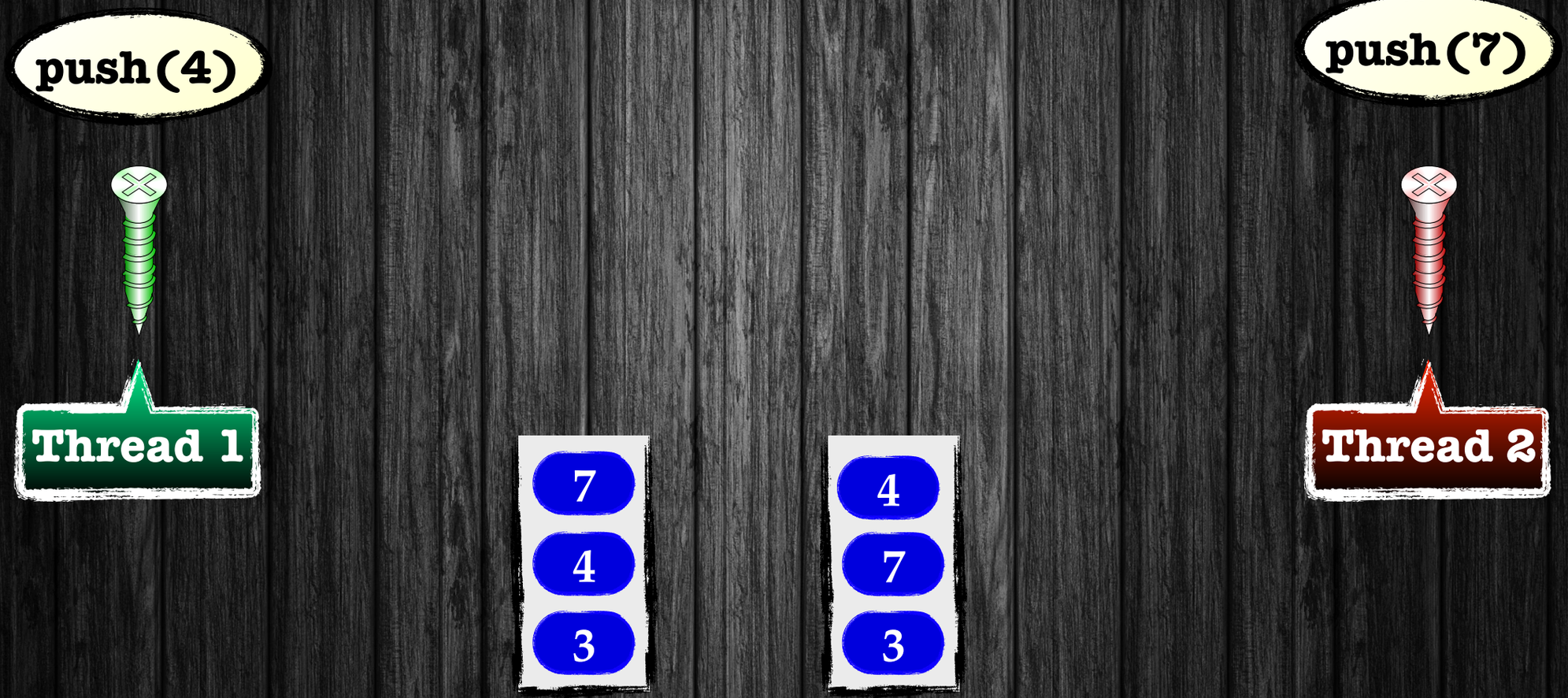
Overview | Goal → concurrent setting

Example | Stack | (Basic) Algorithms & Data Structures



Overview **Goal** → **concurrent setting**

Example **Stack** **(Basic) Algorithms & Data Structures**



Overview | Goal → concurrent setting

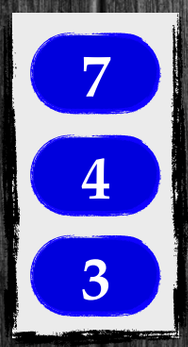
Example | Stack | (Basic) Algorithms & Data Structures

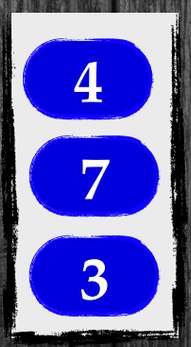
push(4)



push(7)







Which thread goes first?

Overview | Goal → concurrent setting

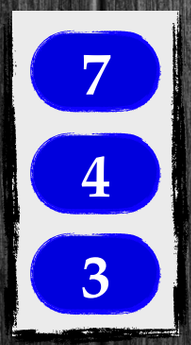
Example | Stack | (Basic) Algorithms & Data Structures

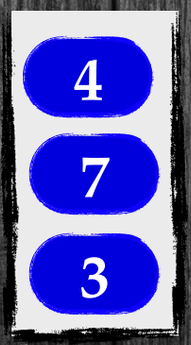
push(4)

push(7)



?







Which thread goes first?

Overview

- Algorithms and Data Structures
- Concurrency
- Algorithmics
- **Efficiency & Correctness**
- Concepts
  - Specifications
  - Safety Properties
  - Liveness Properties

Efficiency & Correctness

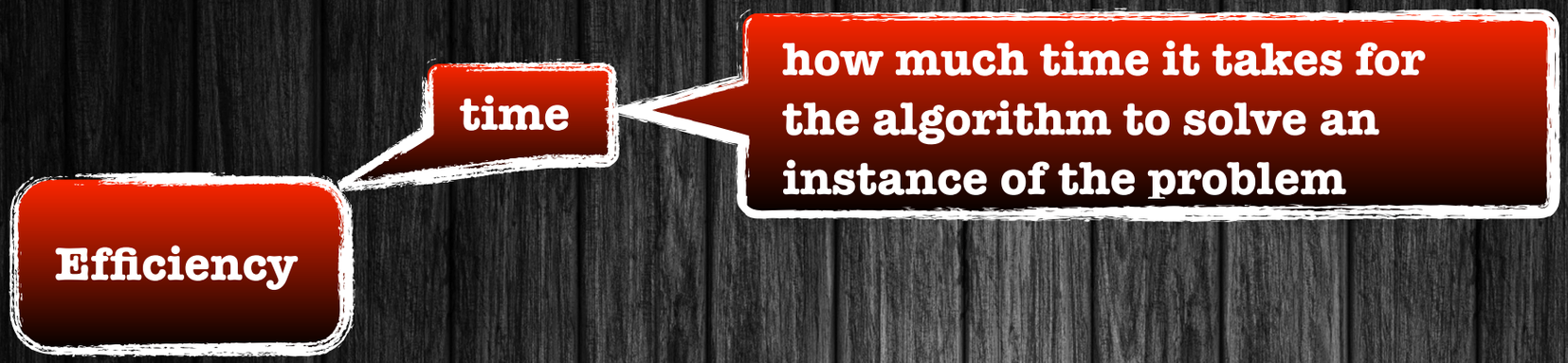
Efficiency & Correctness

Efficiency

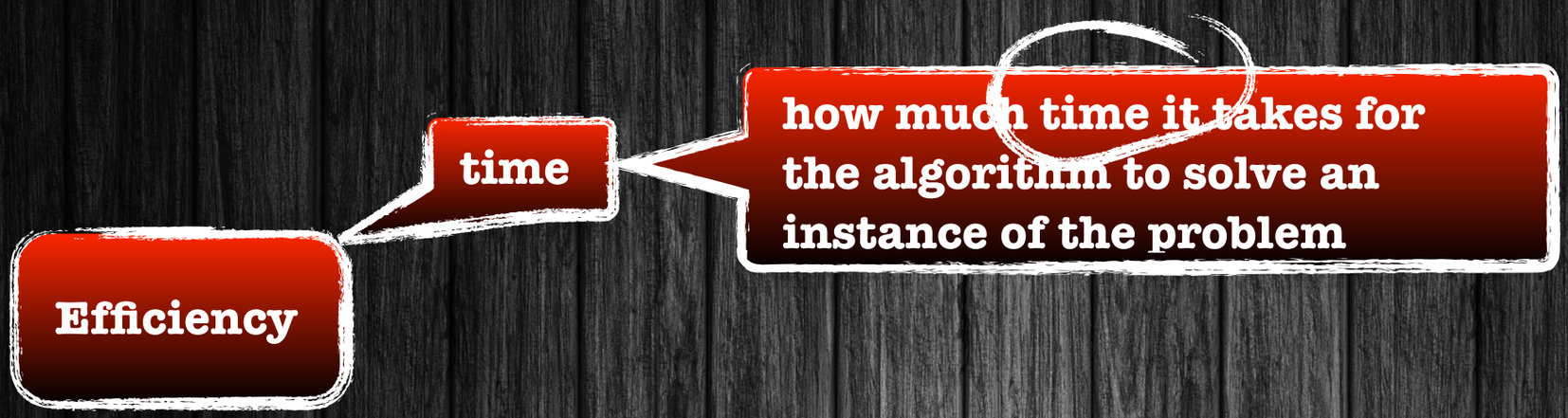
Efficiency & Correctness



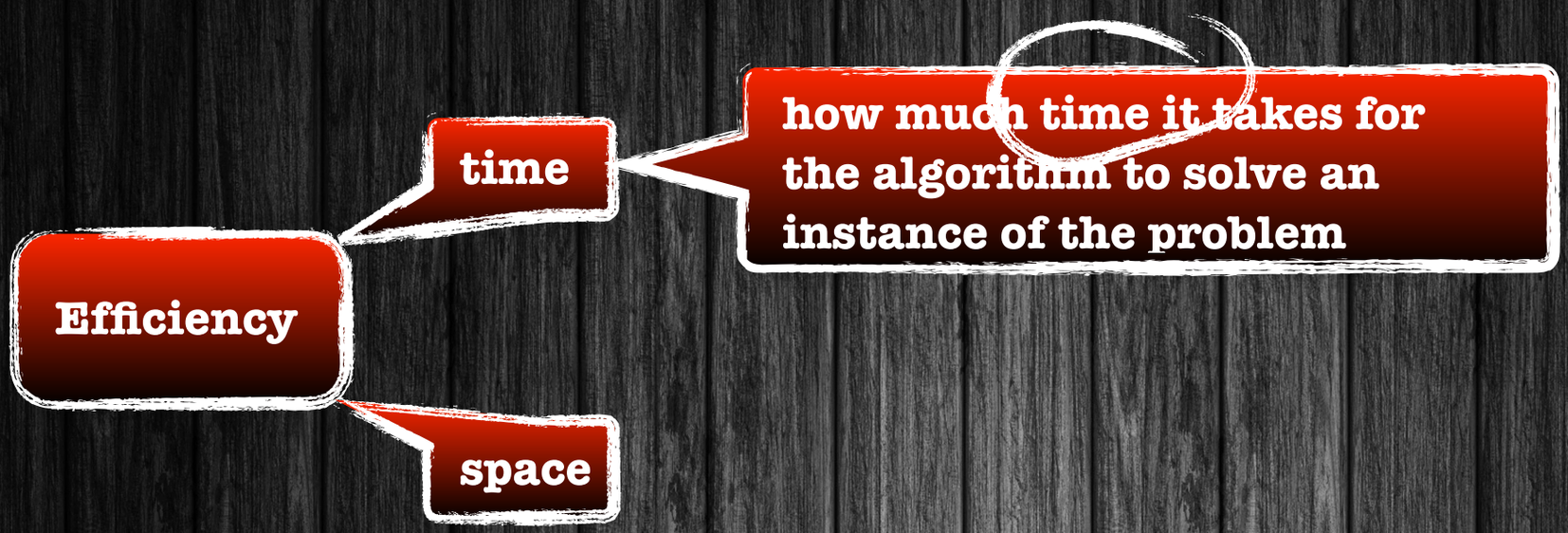
Efficiency & Correctness



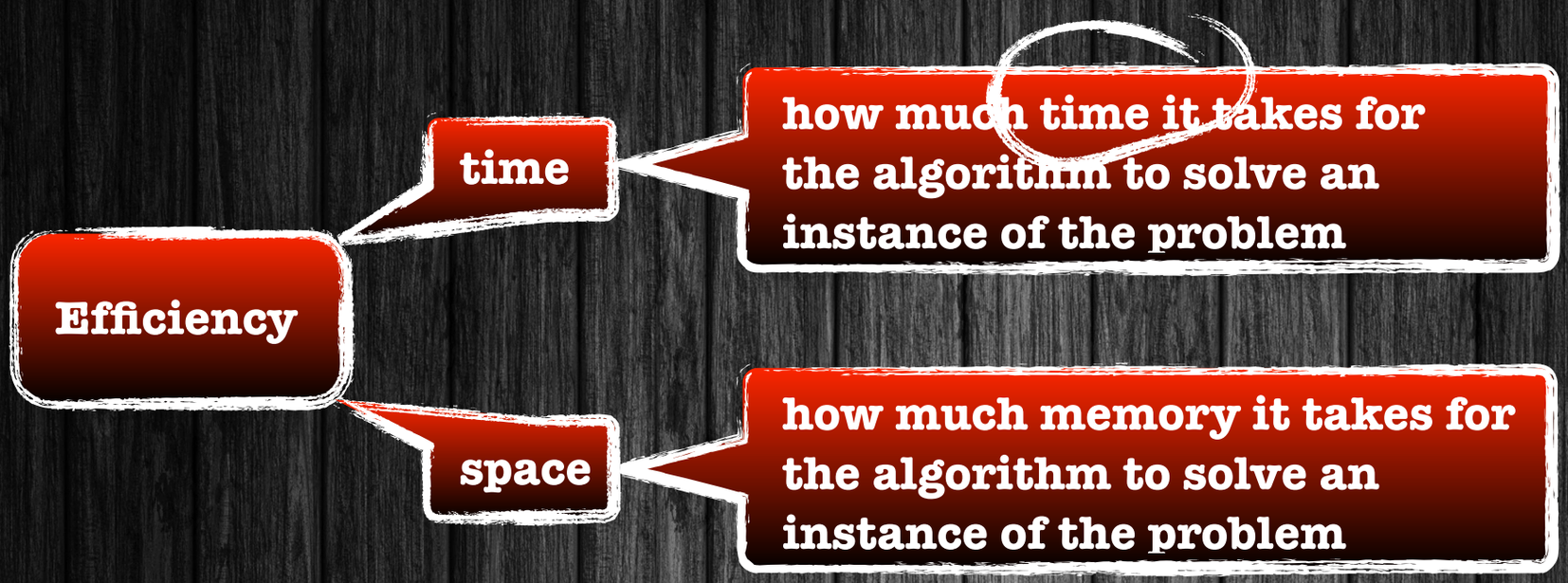
Efficiency & Correctness



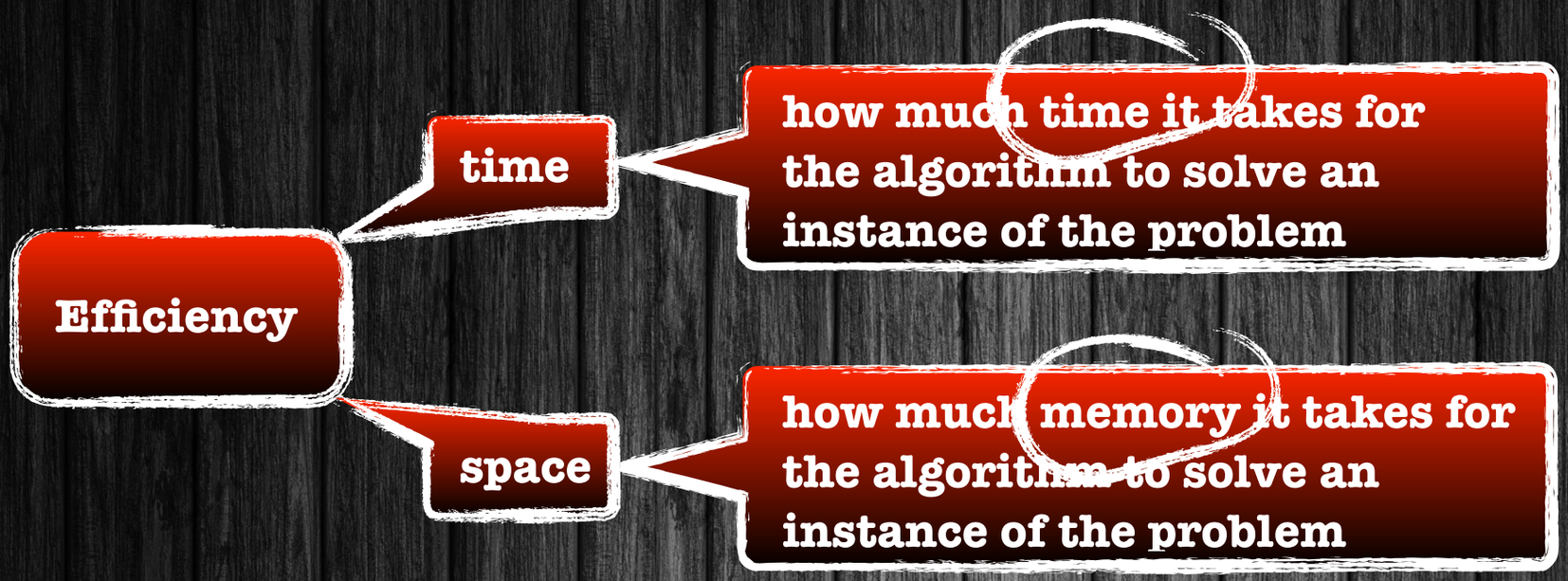
Efficiency & Correctness



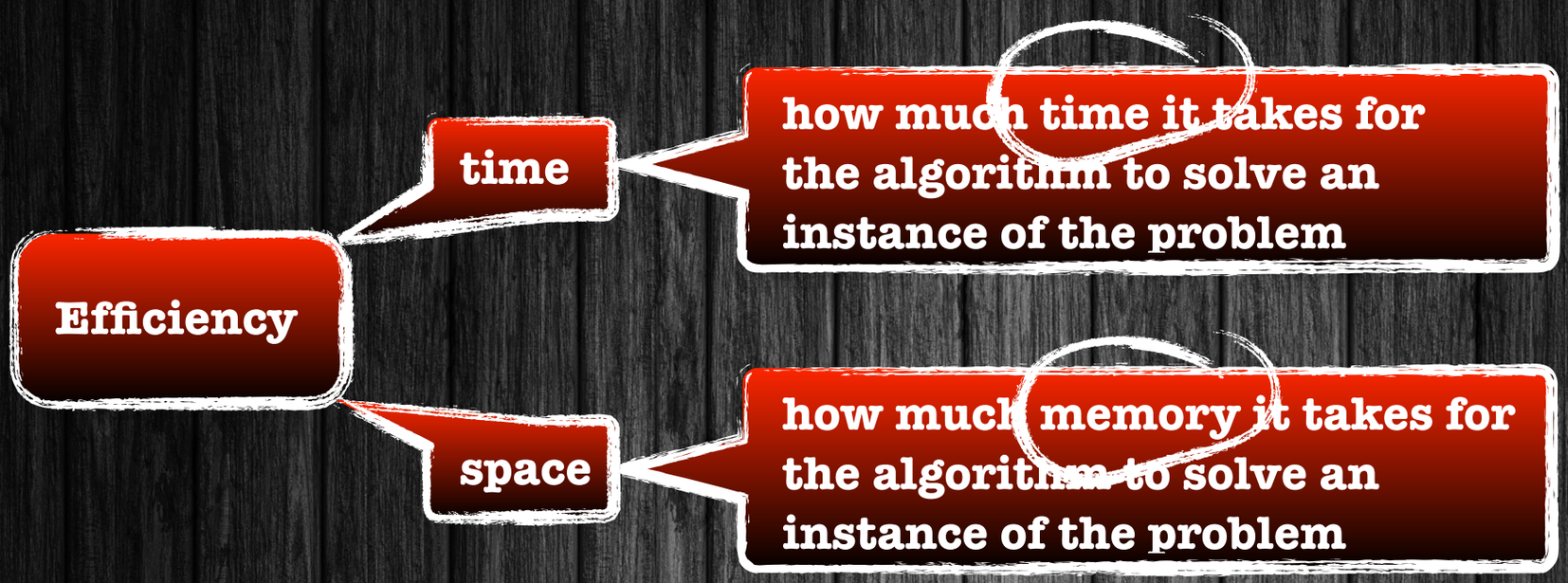
Efficiency & Correctness



Efficiency & Correctness

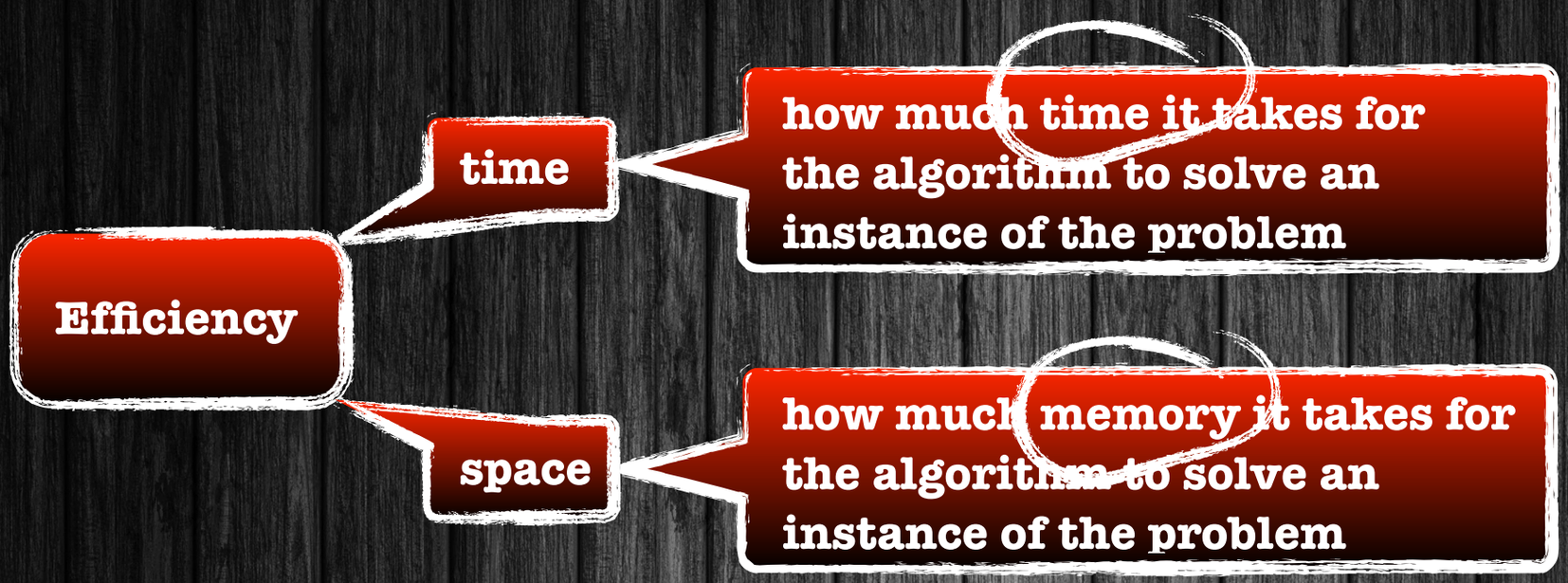


Efficiency & Correctness



Complexity Theory

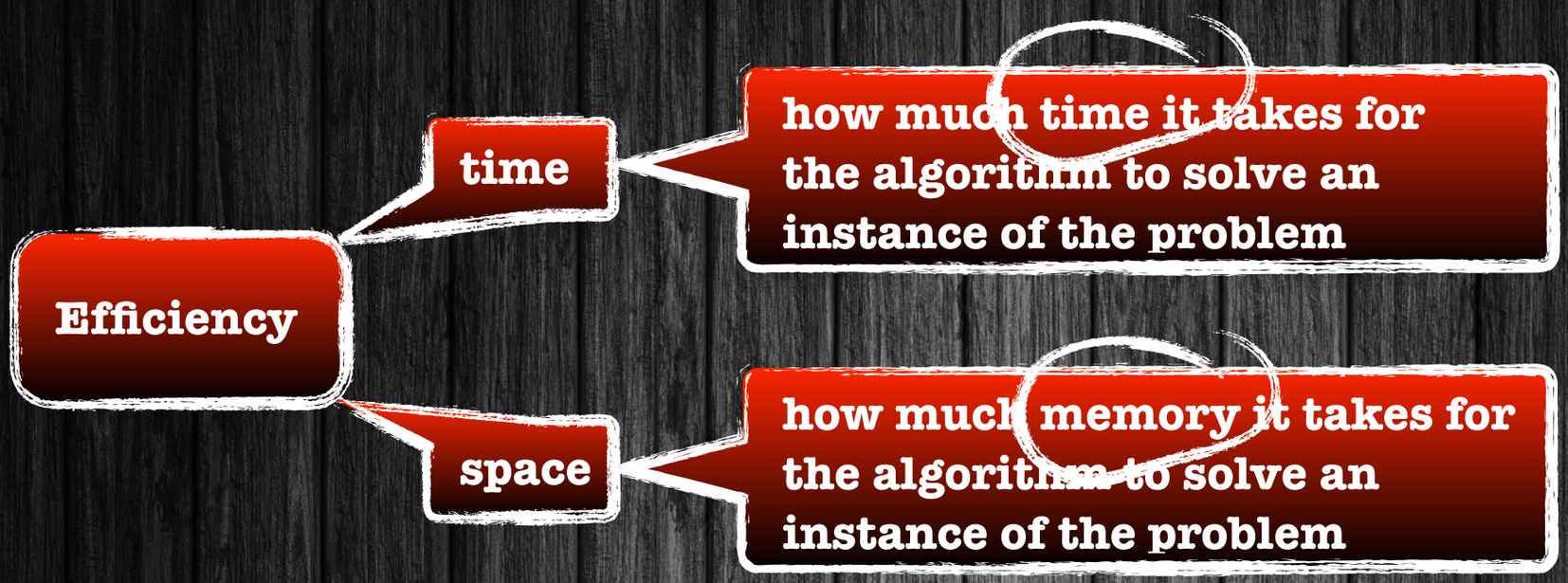
Efficiency & Correctness



Complexity Theory

Reliability

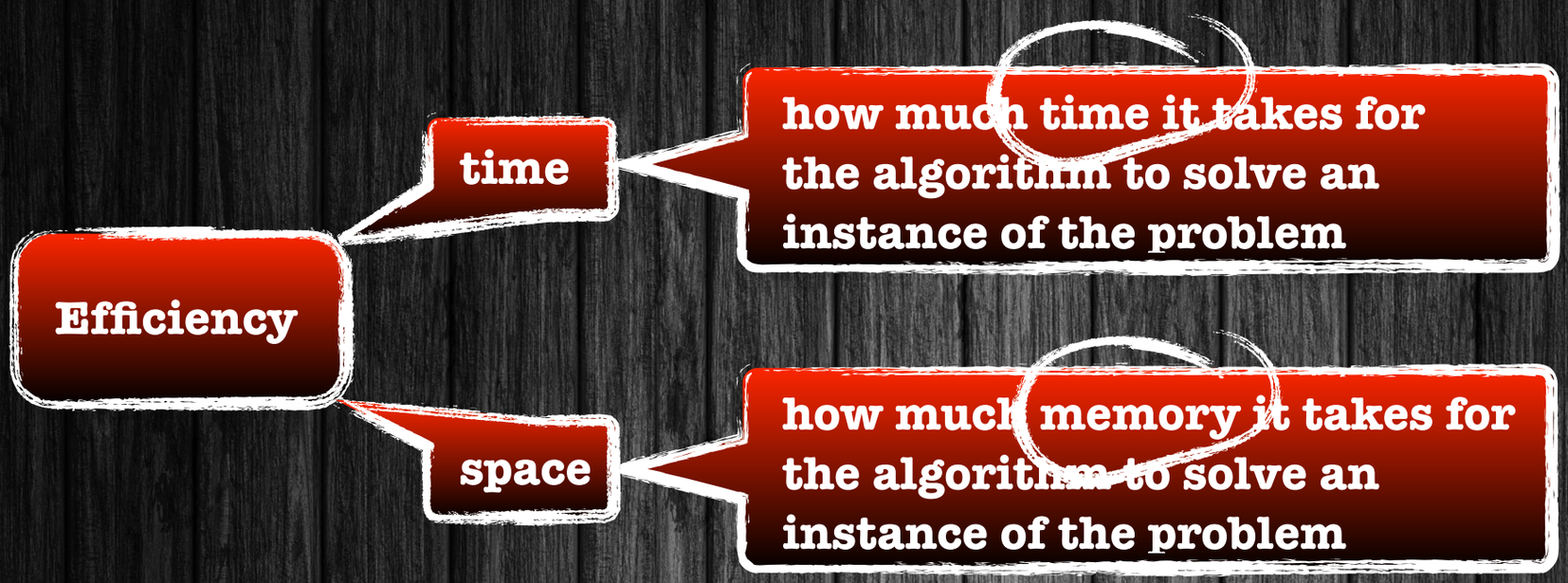
Efficiency & Correctness



Complexity Theory



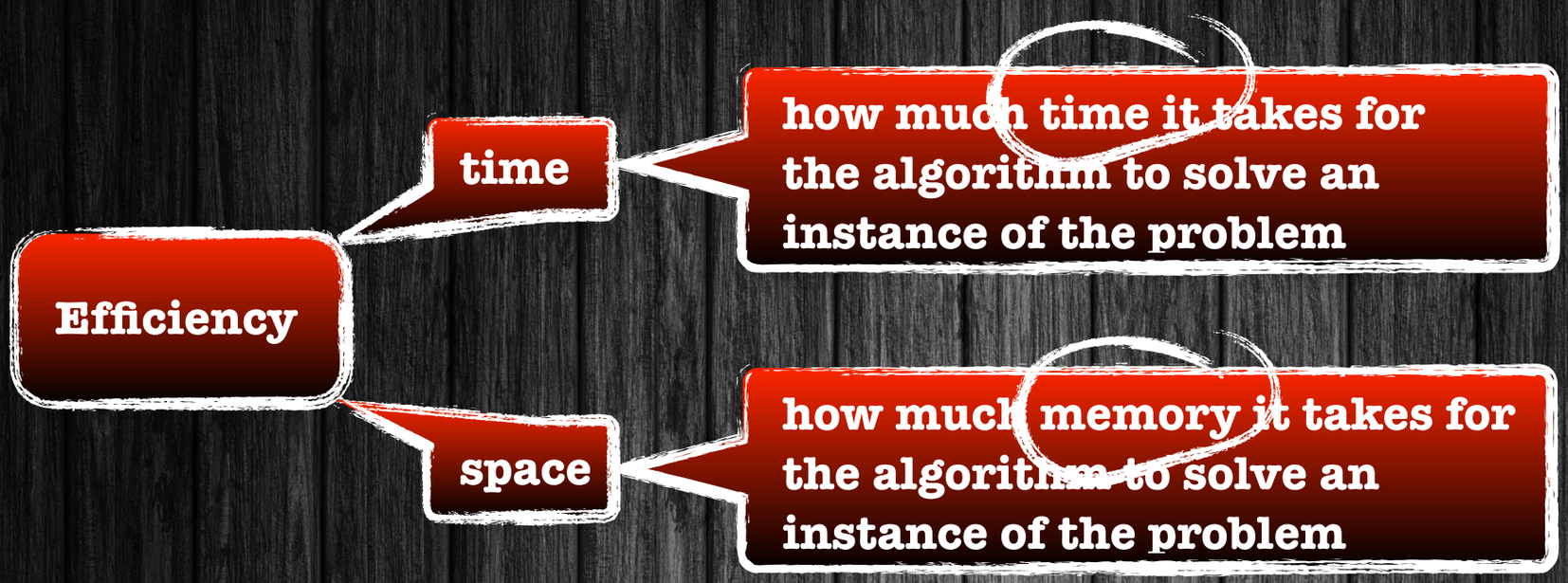
Efficiency & Correctness



Complexity Theory



Efficiency & Correctness



Complexity Theory



- **Testing**
- **Verification**

Overview

- Algorithms and Data Structures
- Concurrency
- Efficiency & Correctness
- **Concepts**
  - **Specifications**
  - **Safety Properties**
  - **Liveness Properties**

Overview

- Algorithms and Data Structures
- Concurrency
- Efficiency & Correctness
- **Concepts**
  - **Specifications**
  - **Safety Properties**
  - **Liveness Properties**

Specification

Specification

“abstract description of the expected behavior of the program”

Specification

“abstract description of the expected behavior of the program”

typically the computations performed by the program

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

output

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

[3,0,5,8,7]

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

[3,0,5,8,7]

[0,8,5,3,7]

permutation

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

output

- b is a permutation of a
- b is sorted

$b_1 \leq b_2 \leq \ldots \leq b_n$

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

$b_1 \leq b_2 \leq \ldots \leq b_n$

[3,0,5,8,7]

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

$b_1 \leq b_2 \leq \ldots \leq b_n$

[3,0,5,8,7]

[0,3,5,7,8]

sorted permutation

Specification

"abstract description of the expected behavior of the program"

typically the computations performed by the program

algorithms satisfying the "sorting" specification

- Quick Sort
- Merge Sort
- Heap Sort
- etc

Example

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

output

- b is a permutation of a
- b is sorted

$b_1 \leq b_2 \leq \ldots \leq b_n$

[3,0,5,8,7]

[0,3,5,7,8]

sorted permutation

Overview

- Algorithms and Data Structures
- Concurrency
- Efficiency & Correctness
- **Concepts**
  - **Specifications**
  - **Safety Properties**
  - **Liveness Properties**

Overview

- Algorithms and Data Structures
- Concurrency
- Efficiency & Correctness
- **Concepts**
  - **Specifications**
  - **Safety Properties**
  - **Liveness Properties**

Overview

- Algorithms and Data Structures
- Concurrency
- Efficiency & Correctness
- **Concepts**
  - **Specifications**
  - **Safety Properties**
  - **Liveness Properties**



Not logged in Talk Contributions Create account Log in

Article Talk | Read Edit View history | Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

Languages

Safety property

From Wikipedia, the free encyclopedia

In distributed computing, **safety properties** informally require that "something bad will never happen" in a distributed system or distributed algorithm.[1][2] In a database system, a promise to never return data with null fields is an example of a safety guarantee.[3] Another example is that of deadlock freedom—it should never occur that all processes or a distributed system are unable to continue because they are waiting for action from another process.[4]

Safety properties are types of linear time properties studied in the area of model checking, along with liveness properties.[4] Unlike liveness properties, safety properties can be violated by a finite execution of a distributed system. All properties can be expressed as the intersection of safety and liveness properties.[3]

References [edit]

1. ^ Rodrigues, Christian Cachin; Rachid Guerraoui; Luís (2010). *Introduction to reliable and secure distributed programming* (2. ed.). Berlin: Springer Berlin. pp. 22–24. ISBN 978-3-642-15259-7.
2. ^ Lamport, L. (1977). "Proving the Correctness of Multiprocess Programs". *IEEE Transactions on Software Engineering* (2): 125–143. CiteSeerX 10.1.1.137.9454. doi:10.1109/TSE.1977.229904.
3. ^ a b Alpern, B.; Schneider, F. B. (1987). "Recognizing safety and liveness". *Distributed Computing.* **2** (3): 117. CiteSeerX 10.1.1.20.5470. doi:10.1007/BF01782772.
4. ^ a b Baier, Christel; Katoen, Joost-Pieter (2008). *Principles of Model Checking.* MIT Press. p. 104. ISBN 9780262026499.

This computing article is a stub. You can help Wikipedia by expanding it.

Categories: Computing stubs | Distributed computing | Model checking

This page was last edited on 4 October 2020, at 23:40 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Specification

- safety properties
- liveness properties

> “abstract description of the expected behavior of the program”

Safety Properties

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers

Specification

safety properti

liveness properti

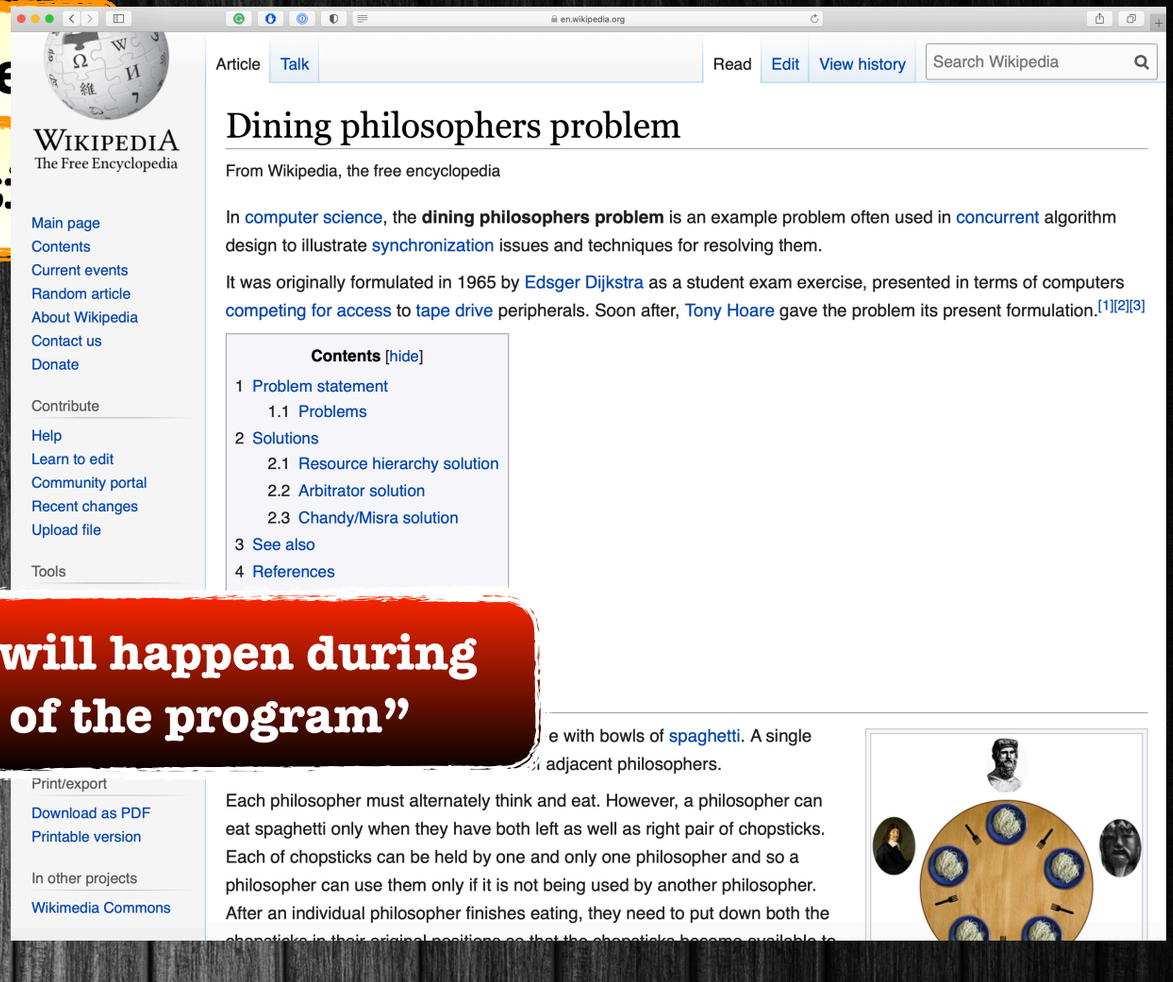
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



Specification

safety properti

liveness properti

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers





Specification

safety properti

liveness properti

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers







Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

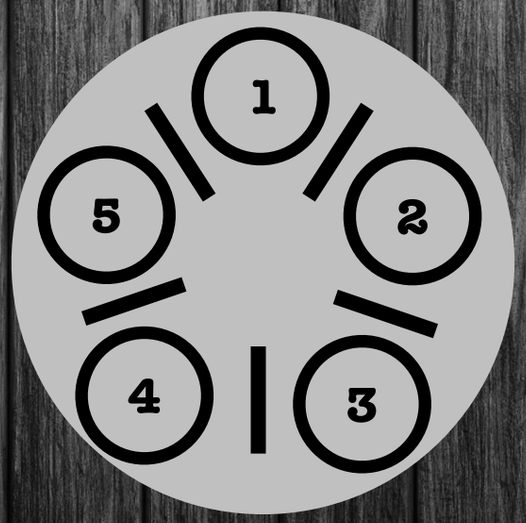
Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers





Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

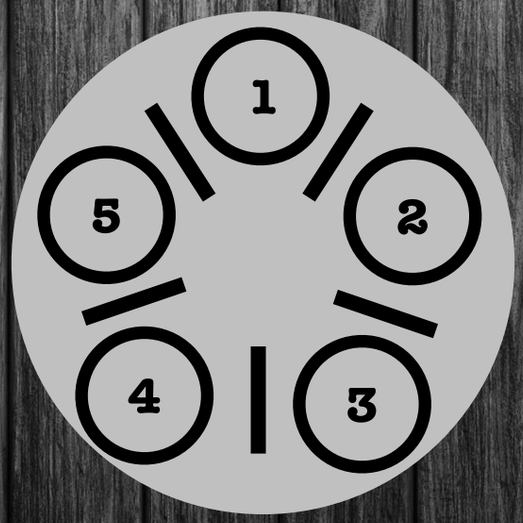
Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers





Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

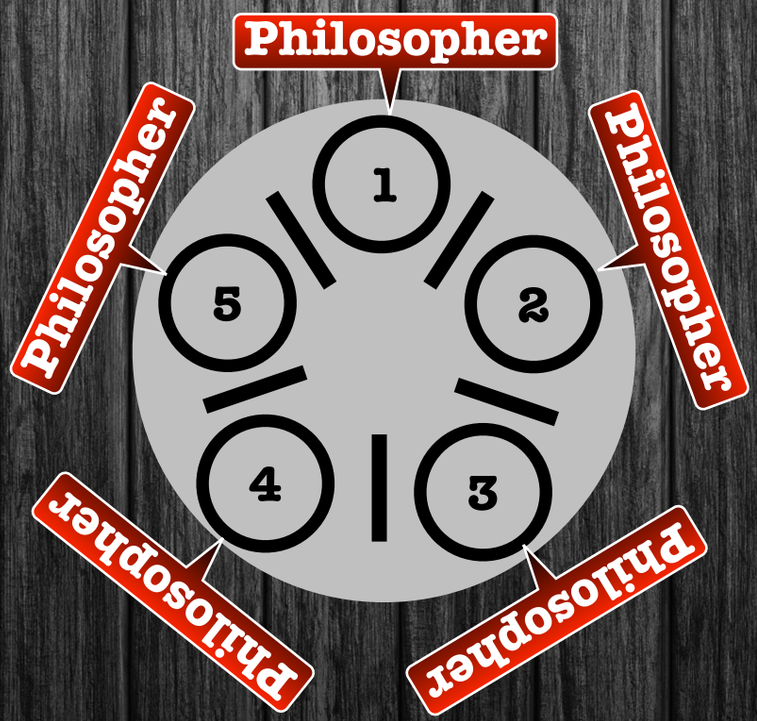
Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers





Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

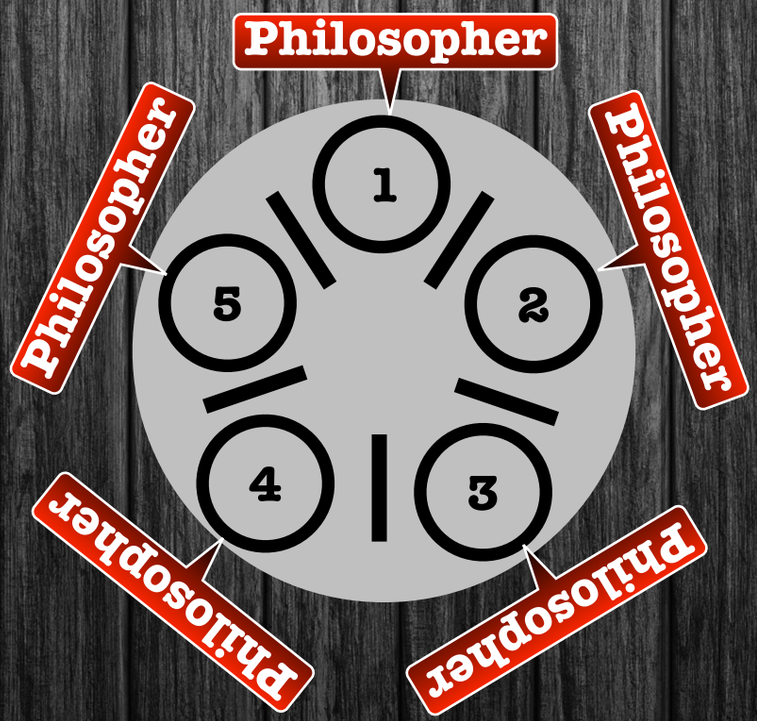
Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers





Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

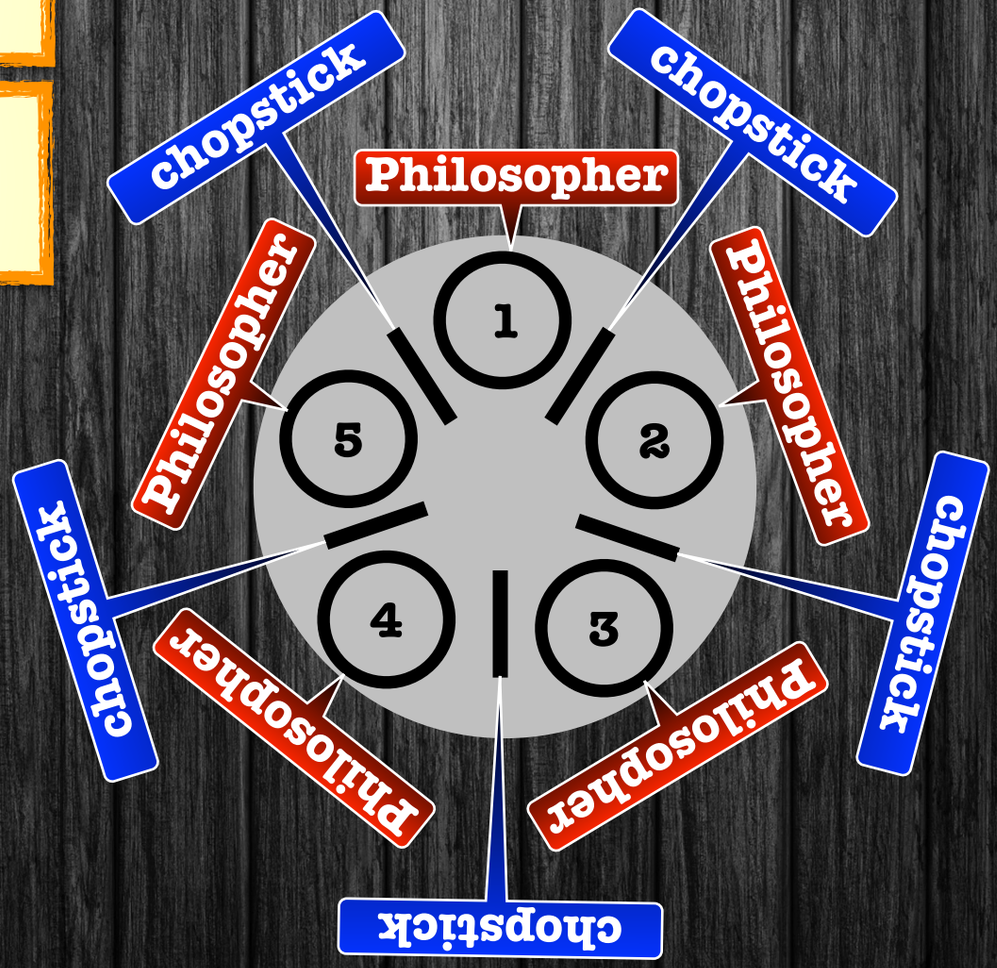
Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers





Specification

- safety properties
- liveness properties

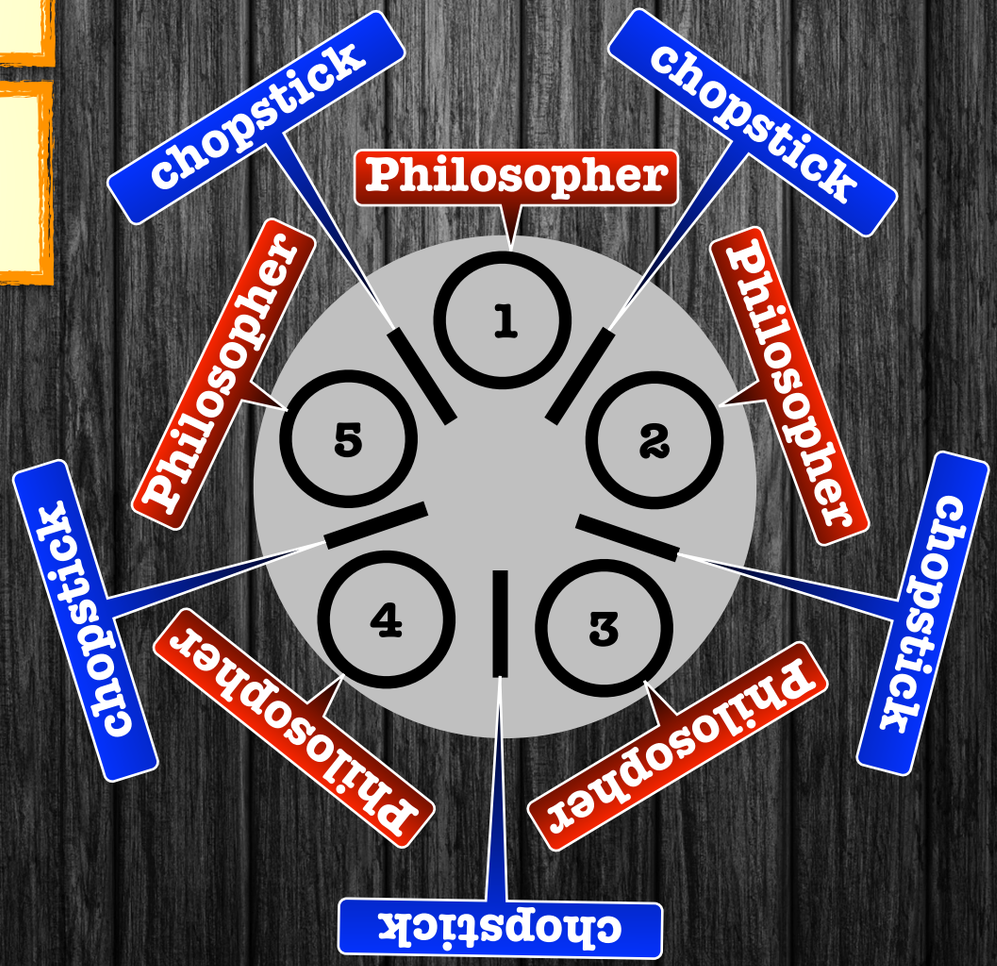
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



Specification

- safety properties
- liveness properties

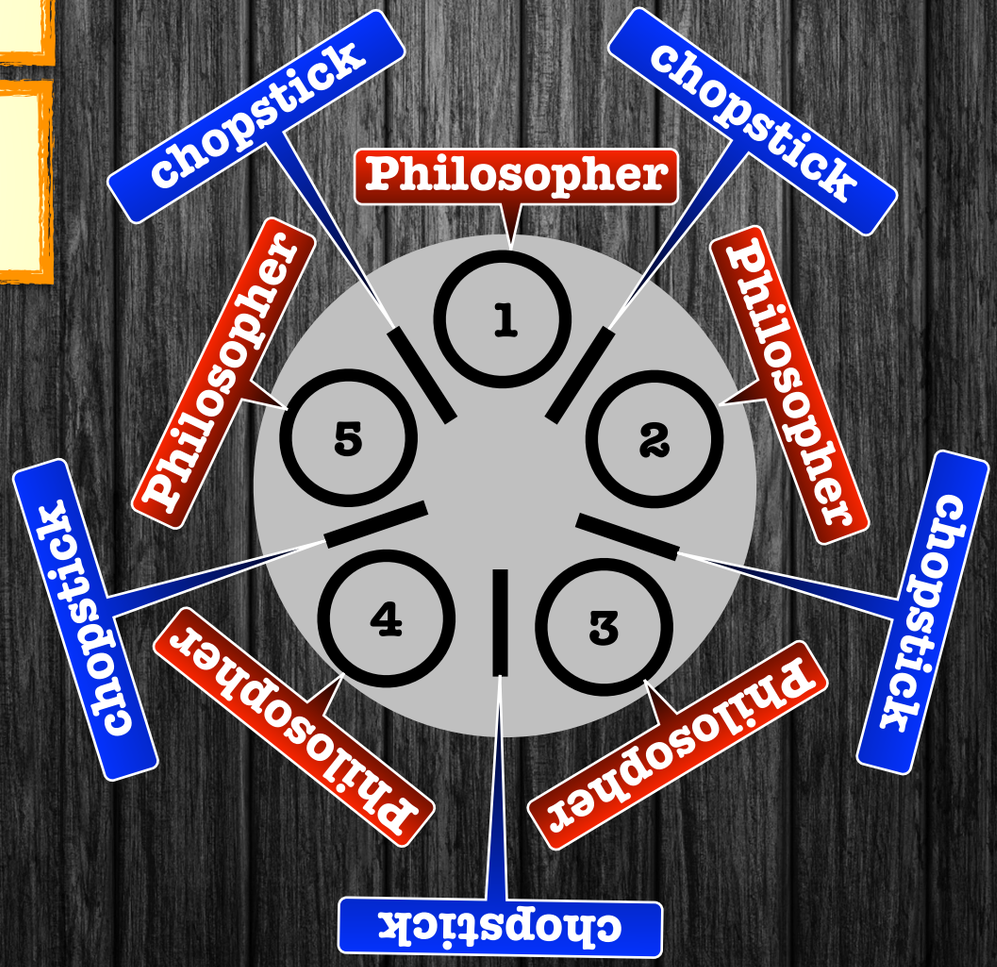
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states

Specification

- safety properties
- liveness properties

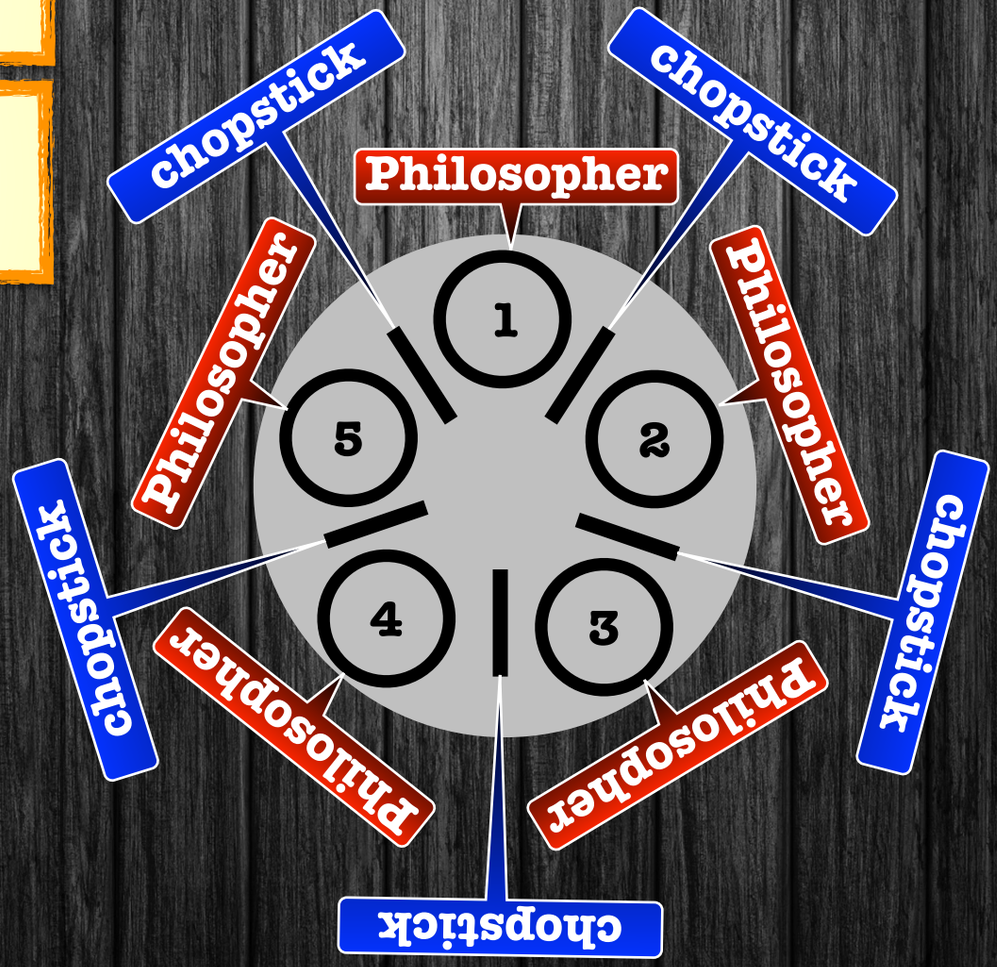
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states

- thinking

Specification

- safety properties
- liveness properties

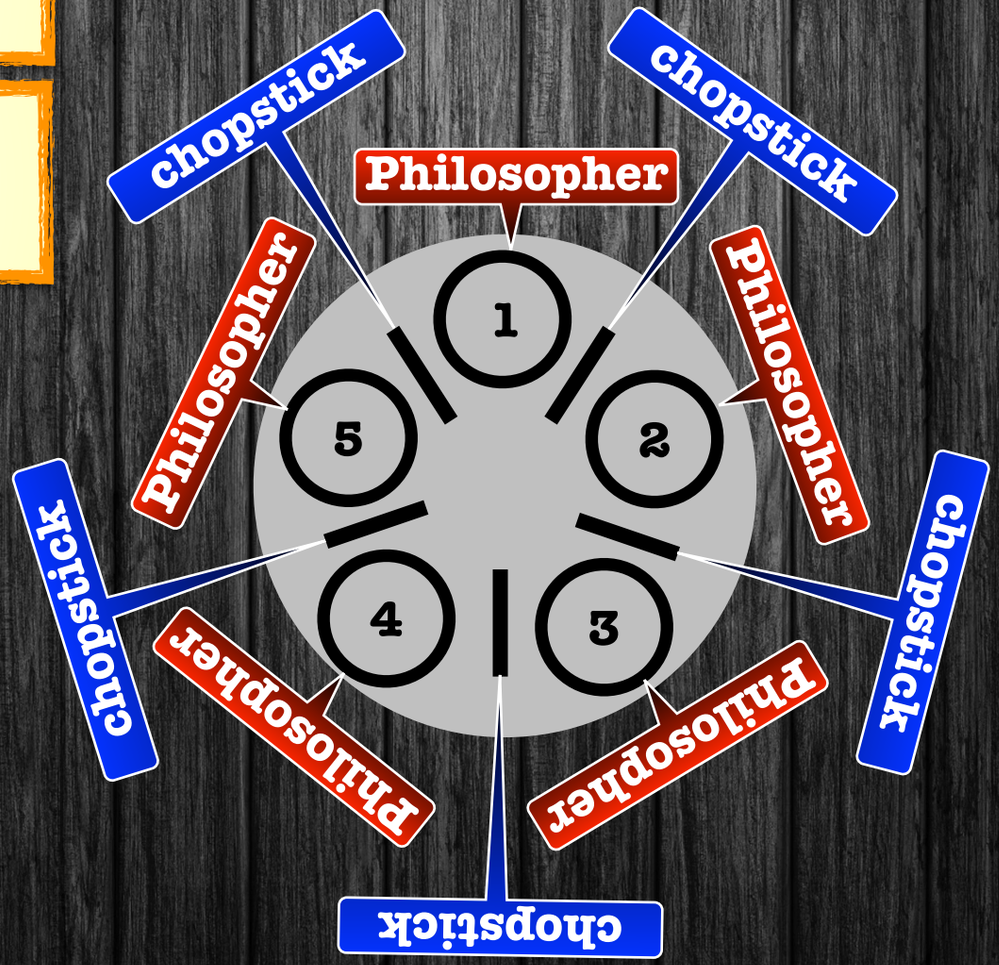
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states

- thinking
- hungry

Specification

- safety properties
- liveness properties

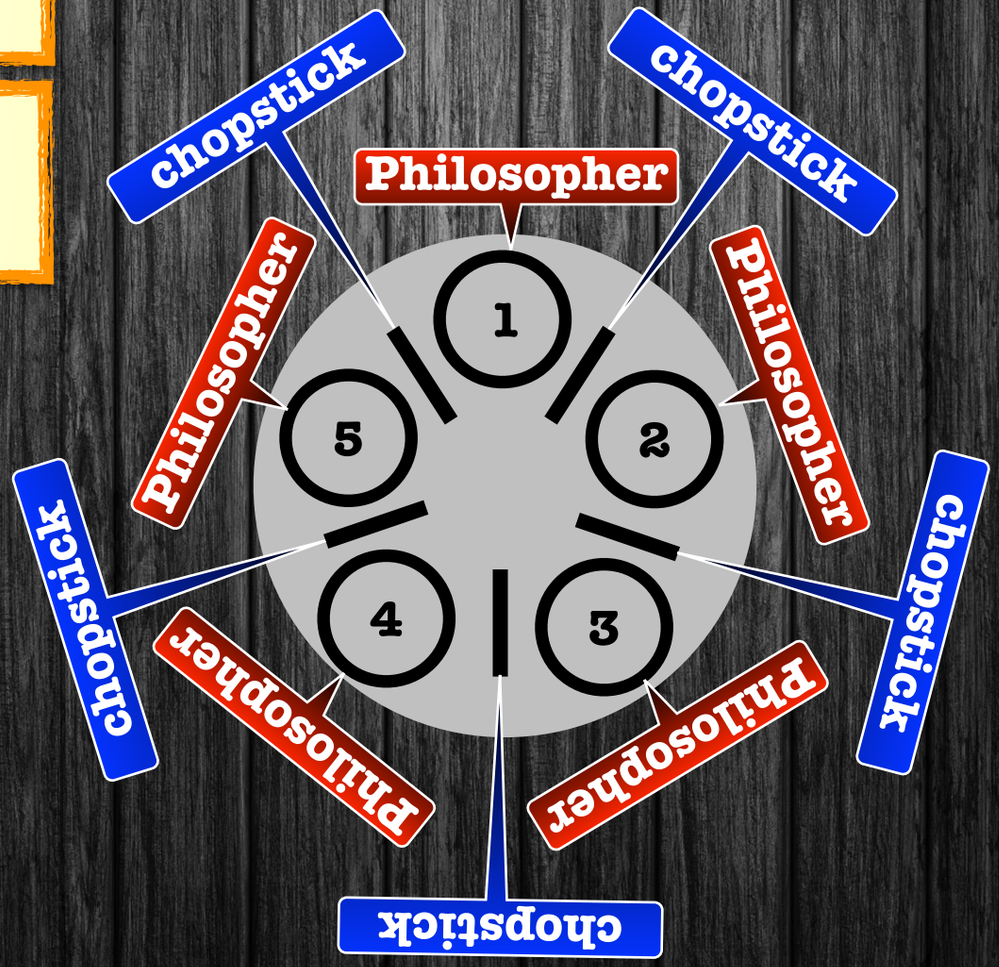
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states

- thinking
- hungry
- eating

Specification

- safety properties
- liveness properties

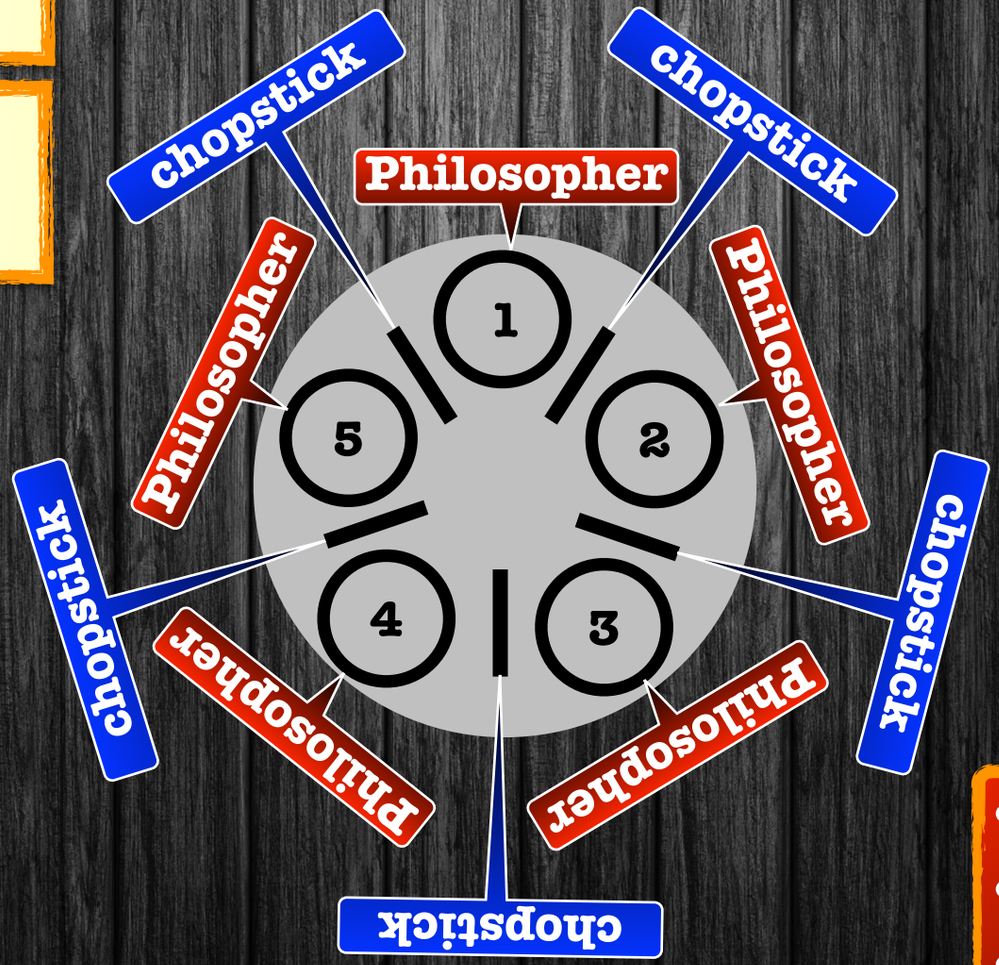
"abstract description of the expected behavior of the program"

Safety Properties

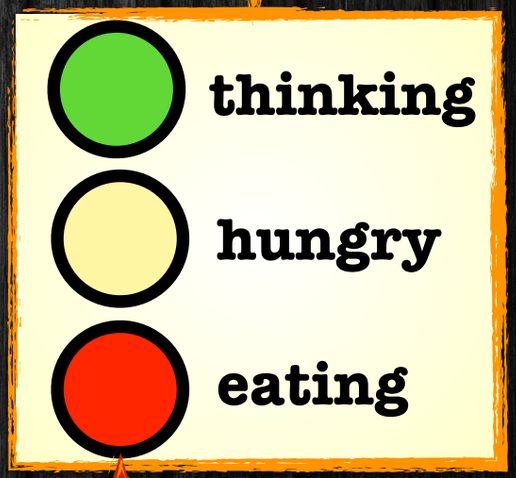
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

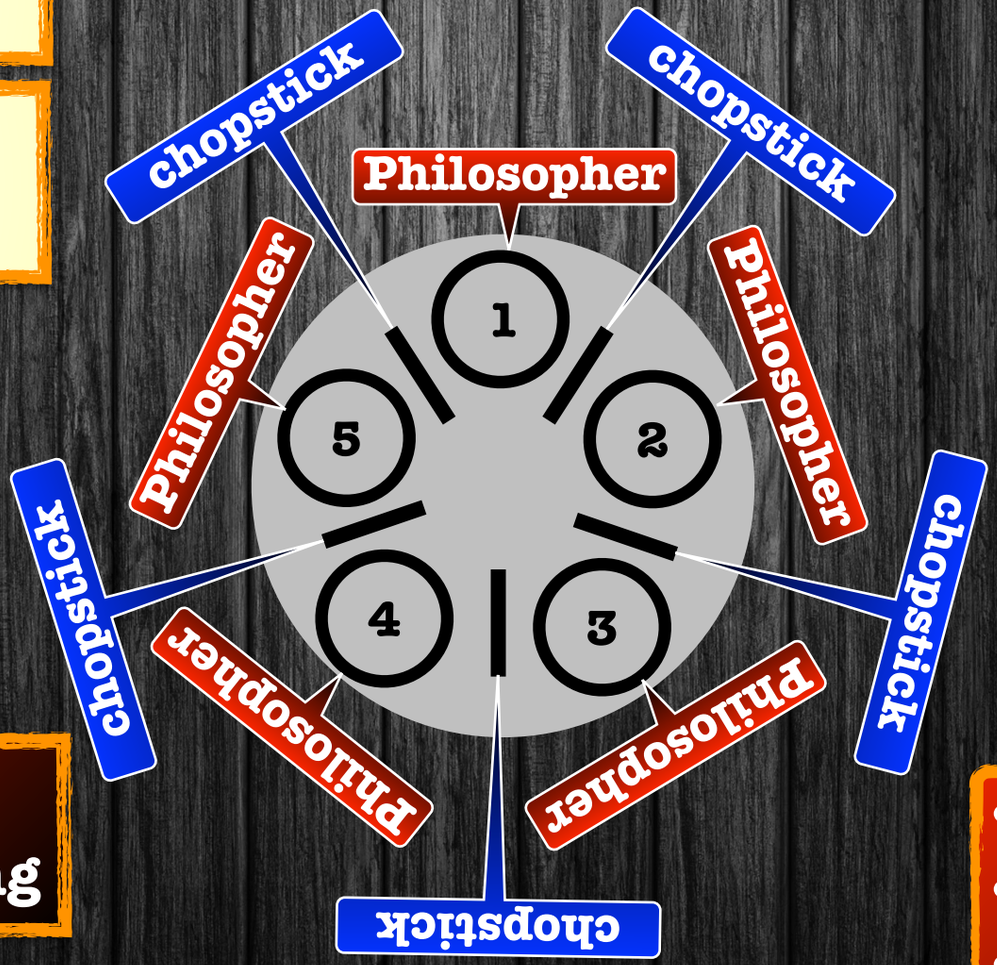
"abstract description of the expected behavior of the program"

Safety Properties

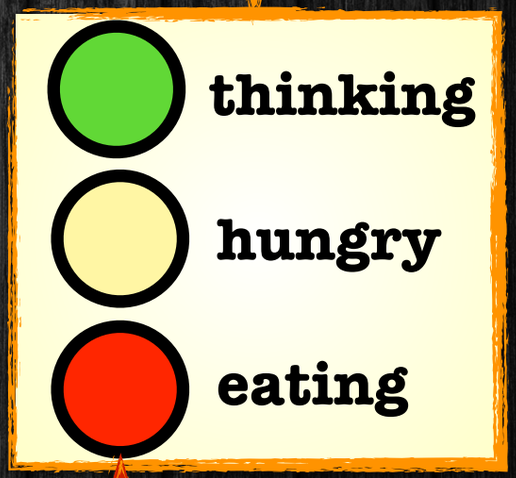
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Initially:
all philosophers thinking

Specification

- safety properties
- liveness properties

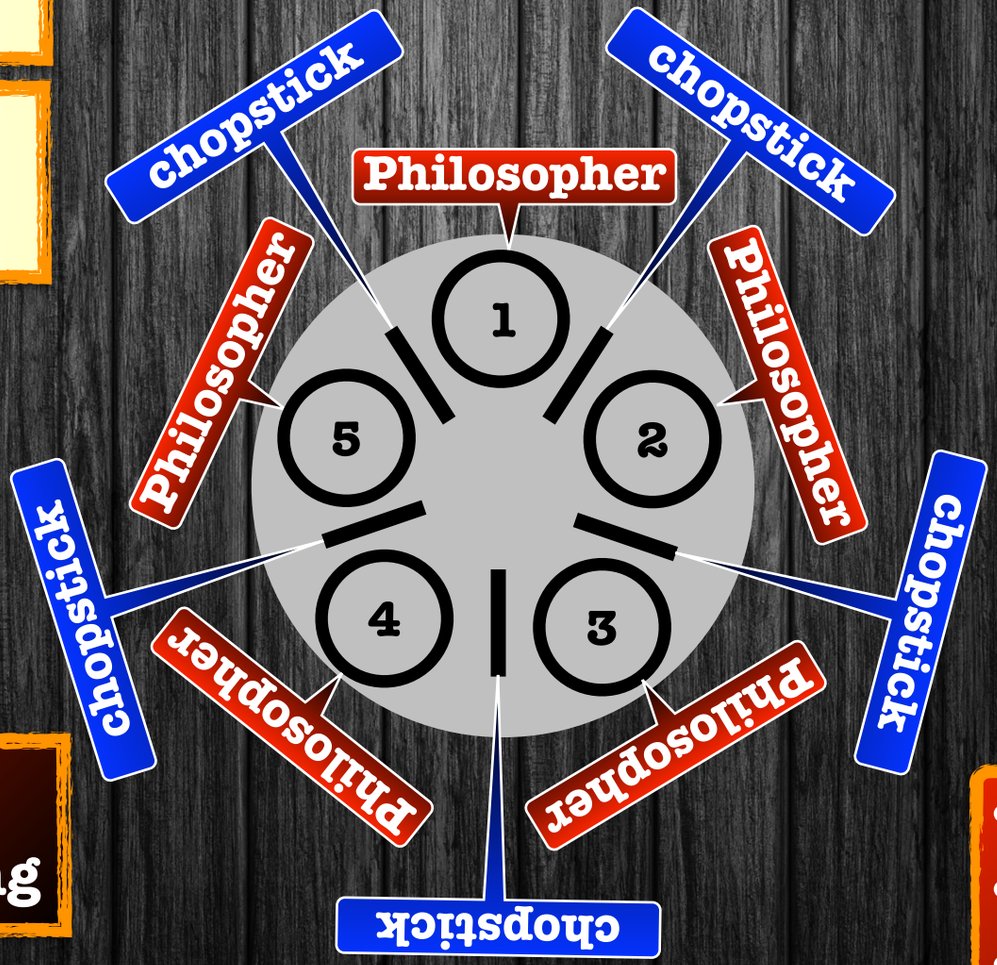
"abstract description of the expected behavior of the program"

Safety Properties

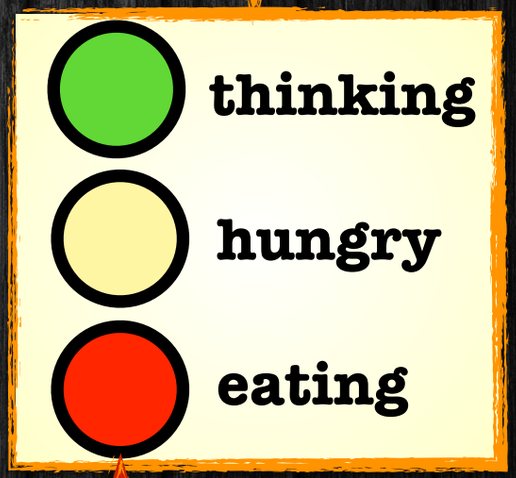
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Initially:
all philosophers thinking

Specification

- safety properties
- liveness properties

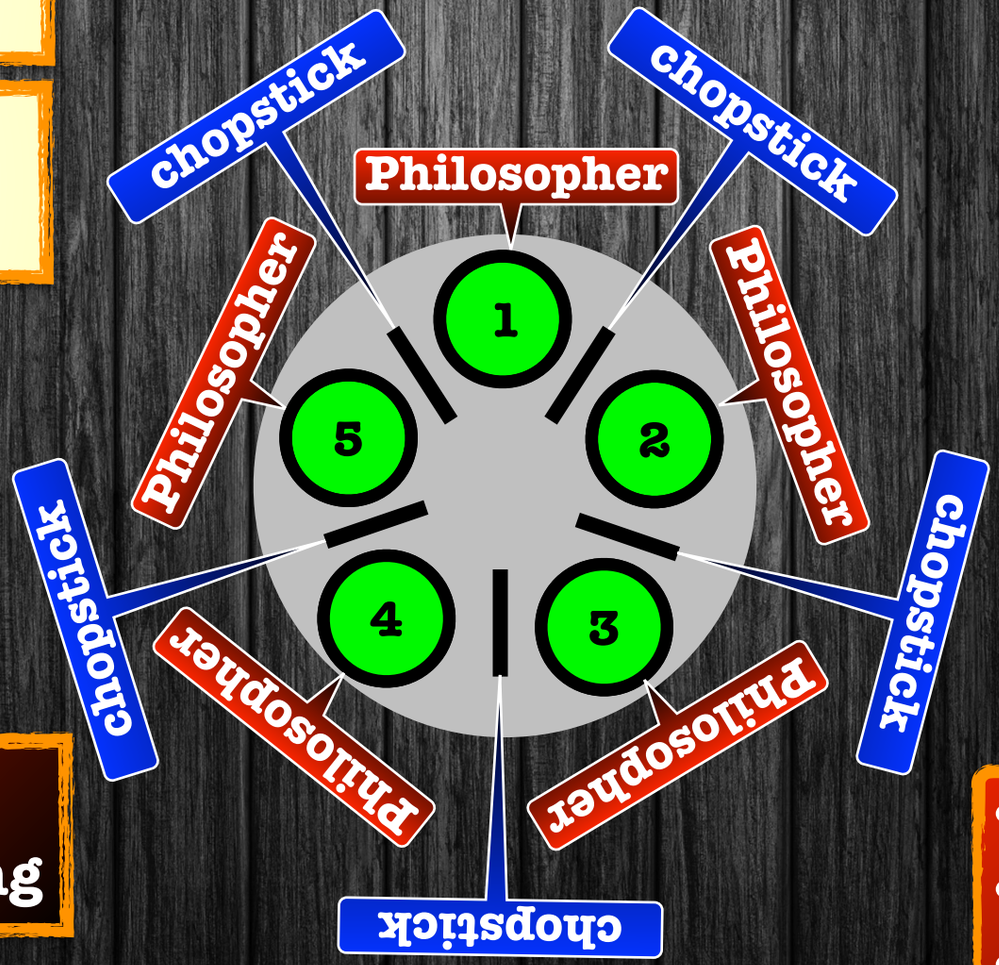
"abstract description of the expected behavior of the program"

Safety Properties

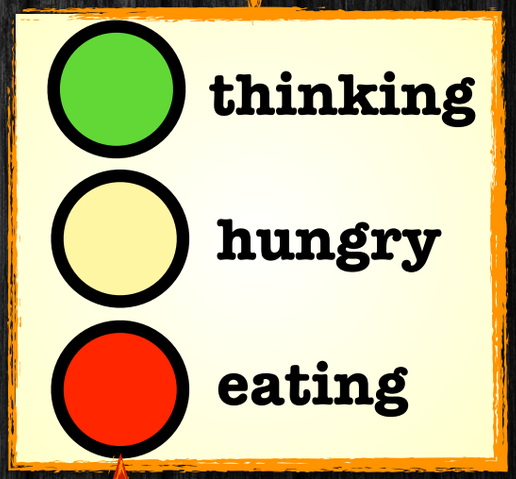
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Initially:
all philosophers thinking

Specification

- safety properties
- liveness properties

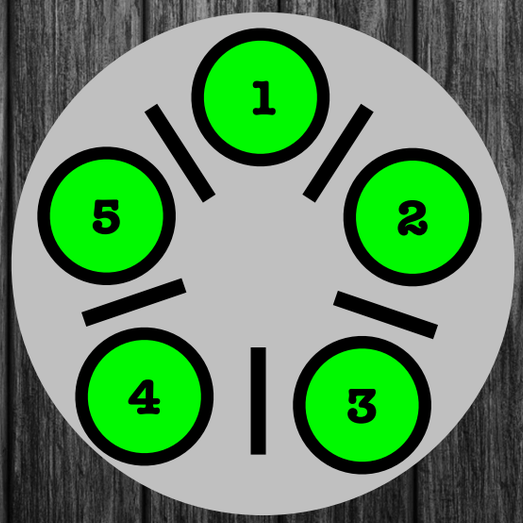
"abstract description of the expected behavior of the program"

Safety Properties

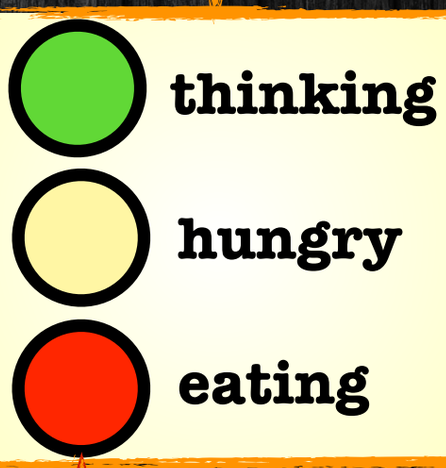
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

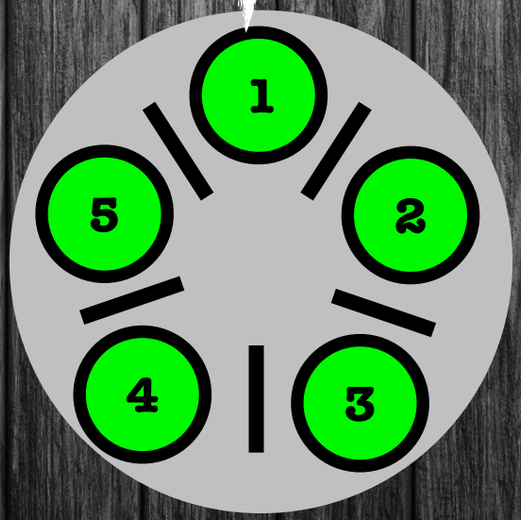
Safety Properties

"nothing bad will happen during the execution of the program"

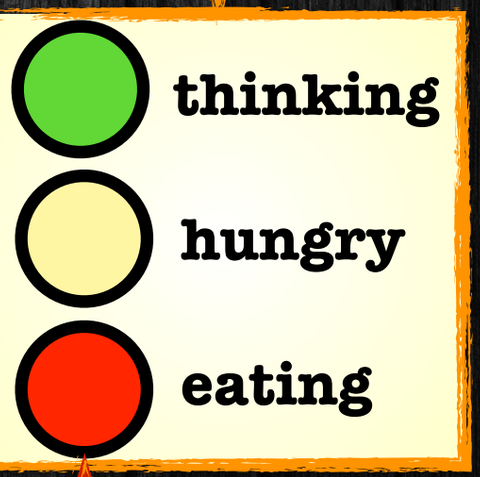
Examples

The Dining Philosophers

becomes hungry



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

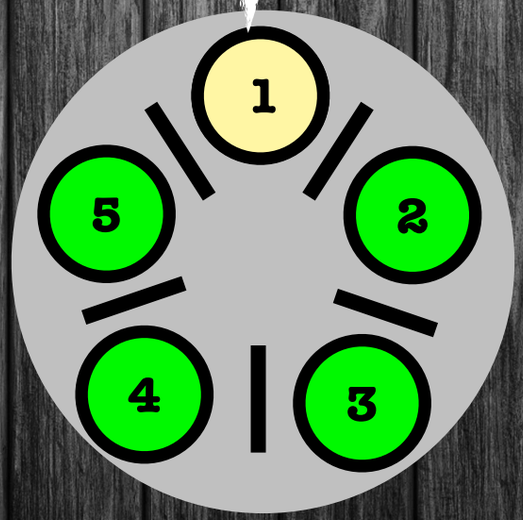
Safety Properties

"nothing bad will happen during the execution of the program"

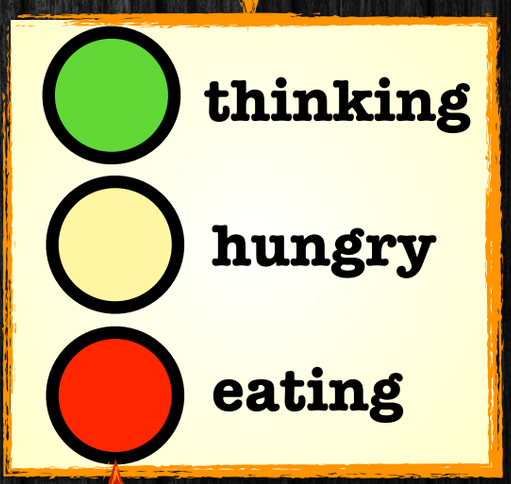
Examples

The Dining Philosophers

becomes hungry



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

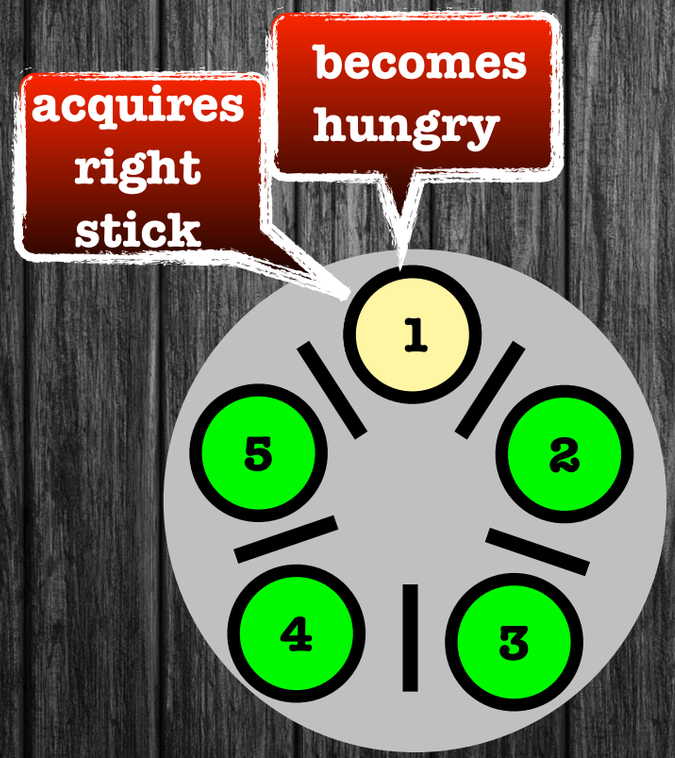
"abstract description of the expected behavior of the program"

Safety Properties

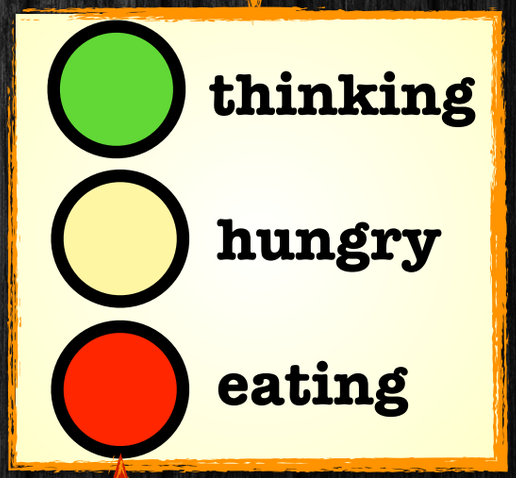
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

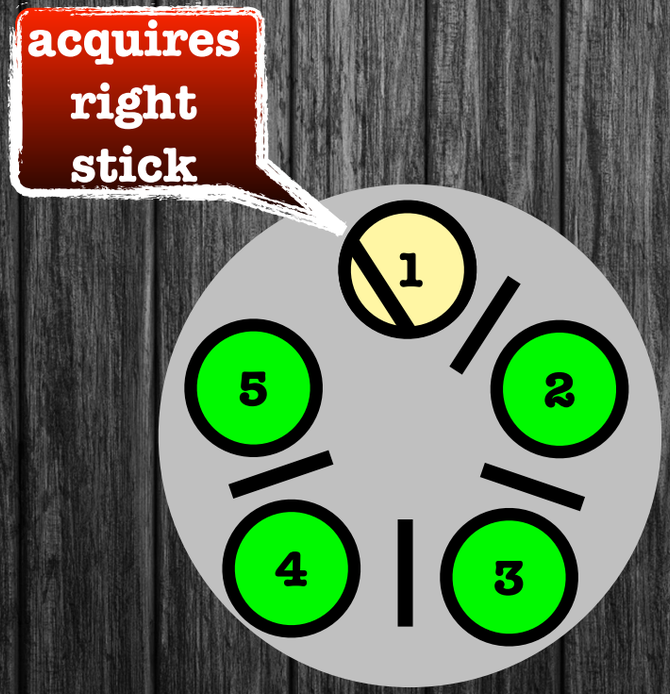
"abstract description of the expected behavior of the program"

Safety Properties

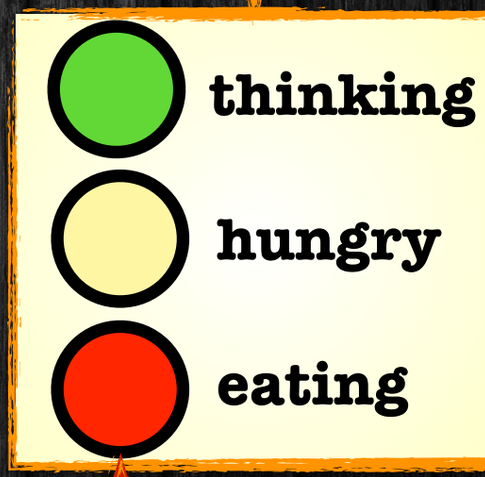
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

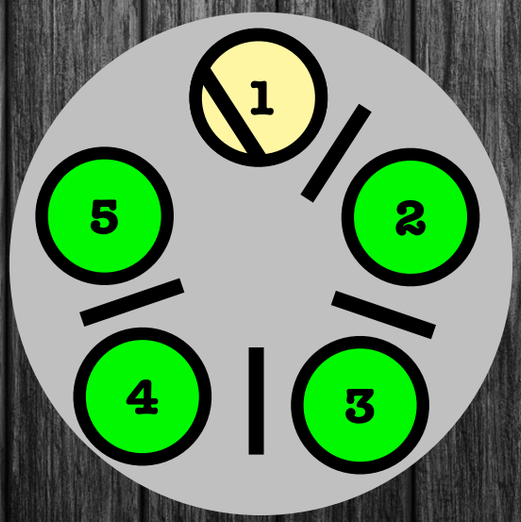
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states

- thinking
- hungry
- eating

- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

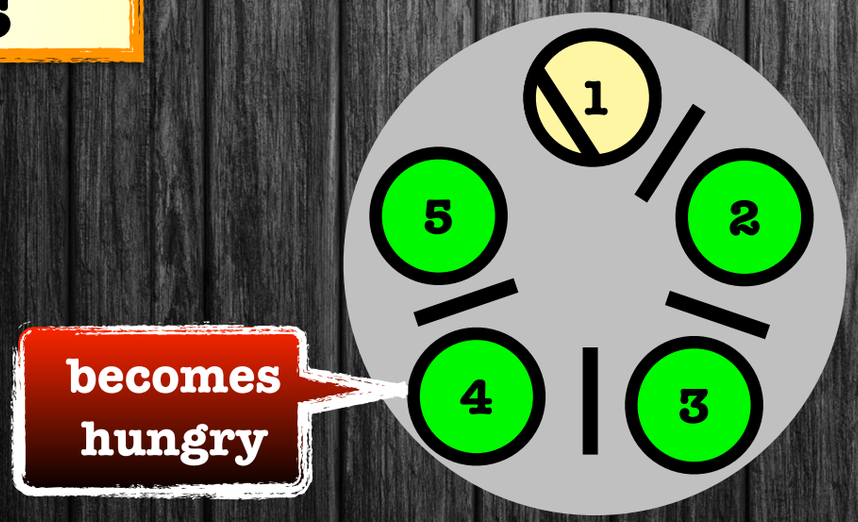
"abstract description of the expected behavior of the program"

Safety Properties

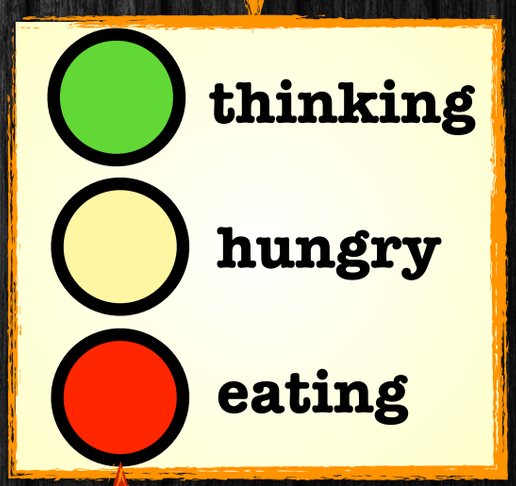
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

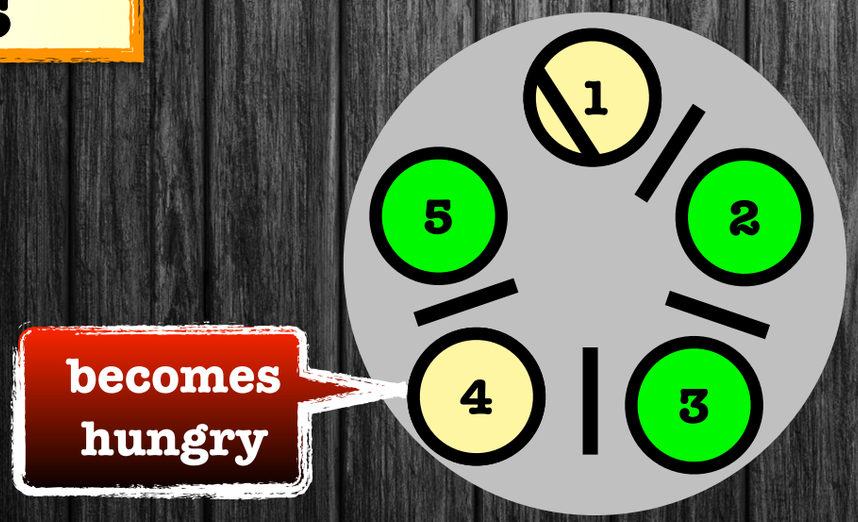
"abstract description of the expected behavior of the program"

Safety Properties

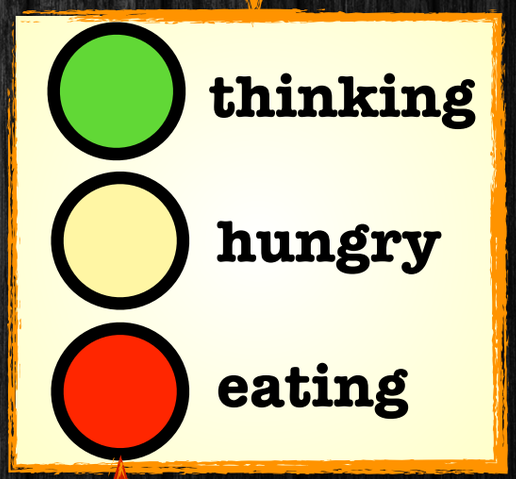
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

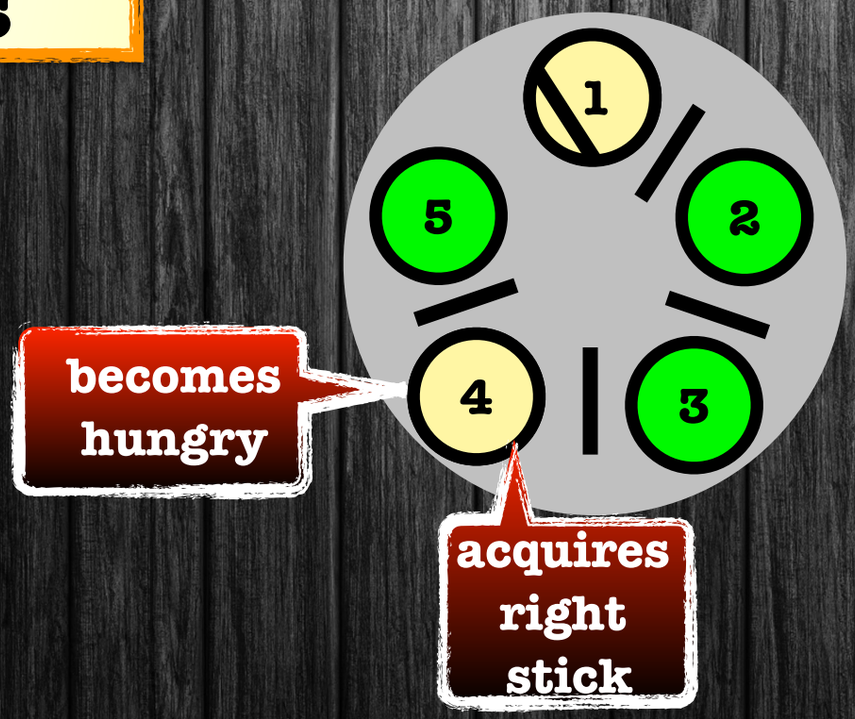
"abstract description of the expected behavior of the program"

Safety Properties

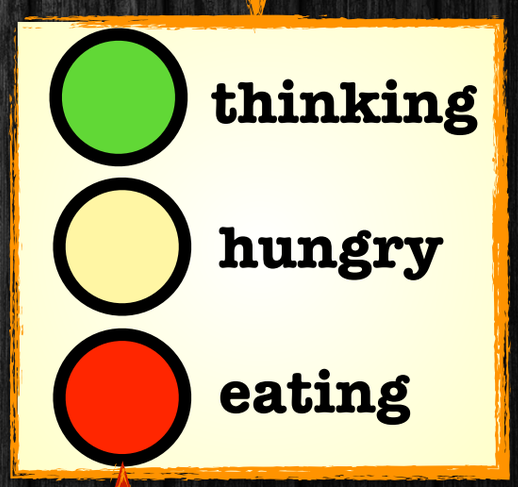
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

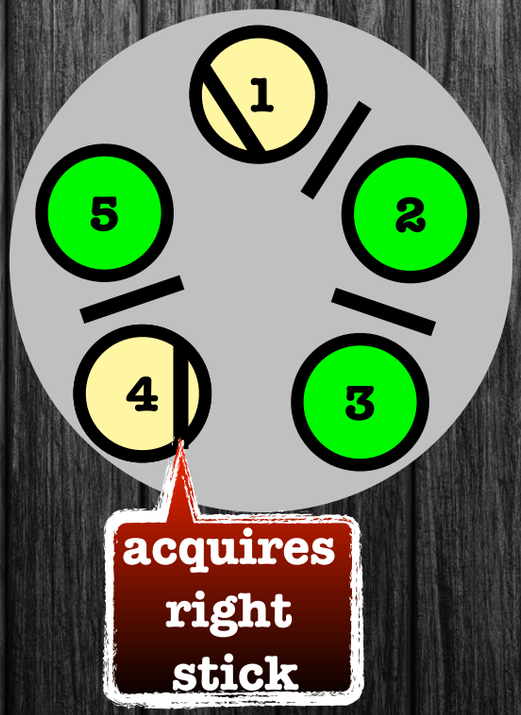
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states

- thinking
- hungry
- eating

- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

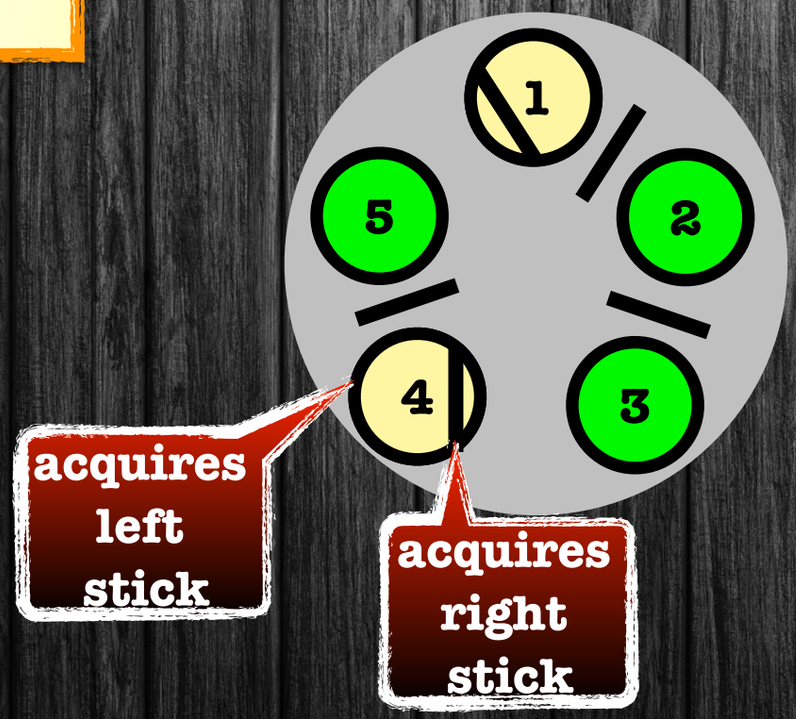
"abstract description of the expected behavior of the program"

Safety Properties

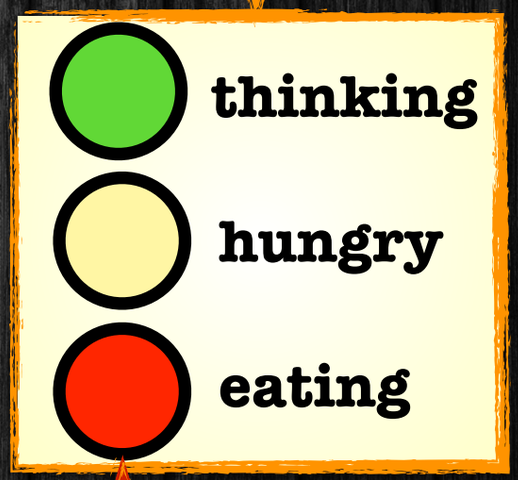
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

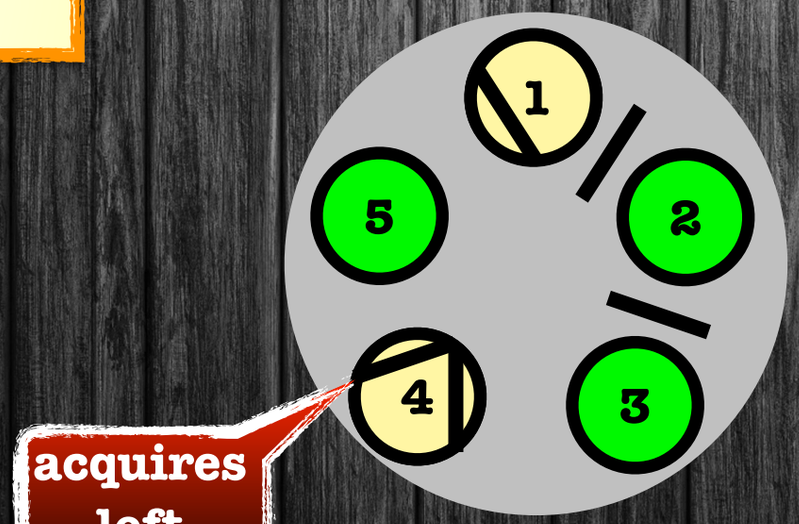
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

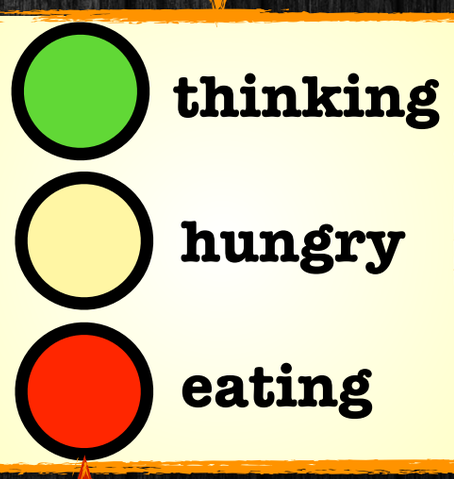
Examples

The Dining Philosophers



acquires left stick

philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

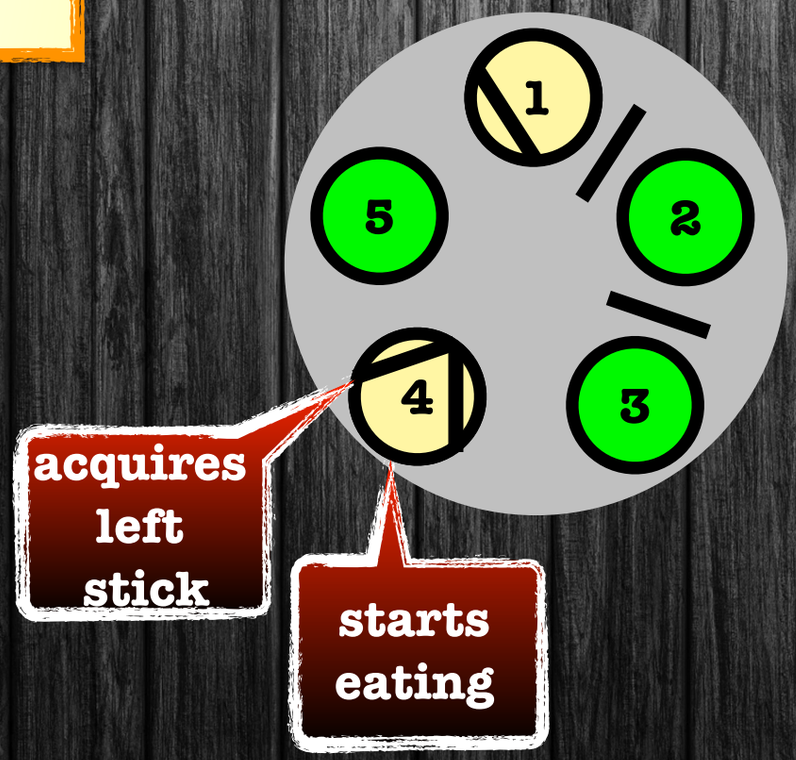
"abstract description of the expected behavior of the program"

Safety Properties

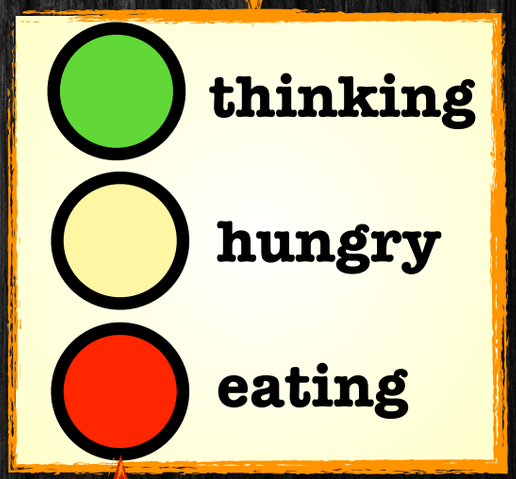
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

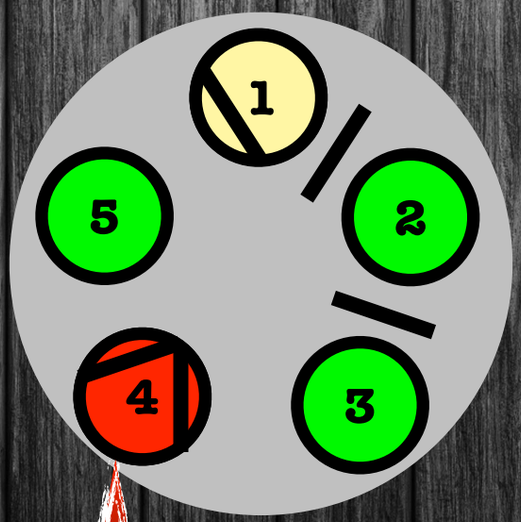
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

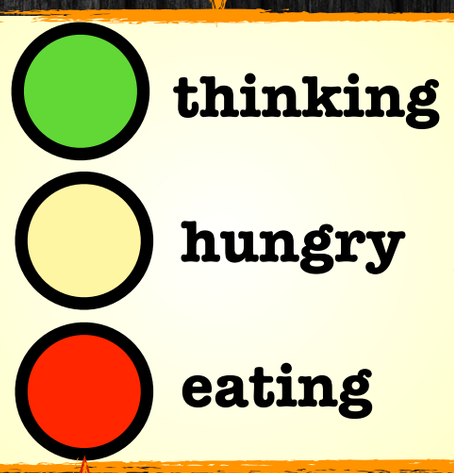
Examples

The Dining Philosophers



starts eating

philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

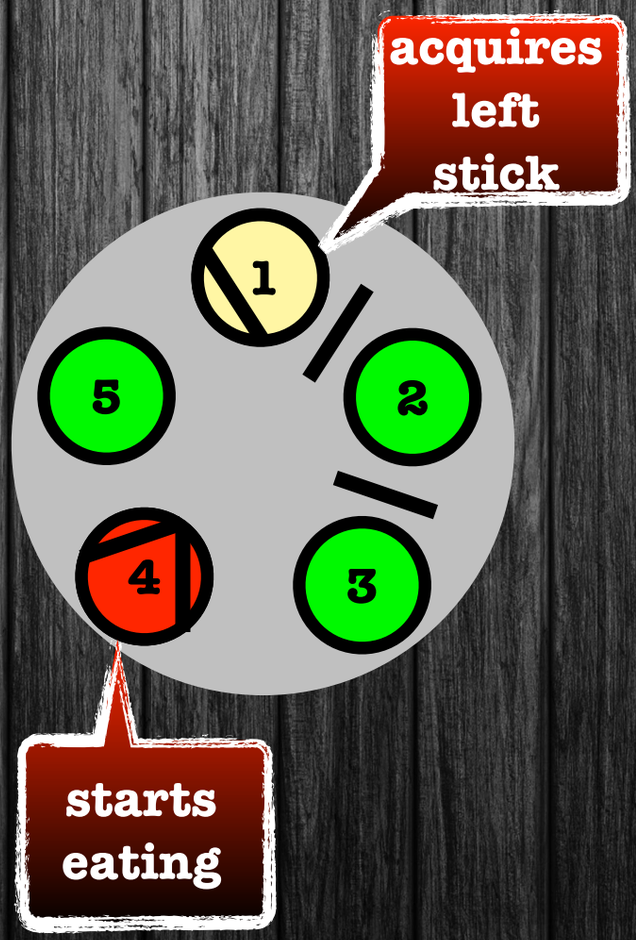
"abstract description of the expected behavior of the program"

Safety Properties

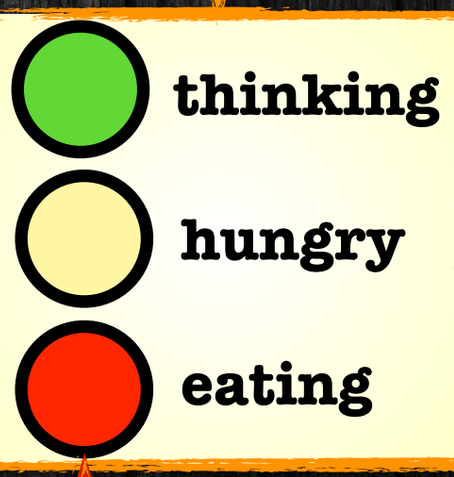
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

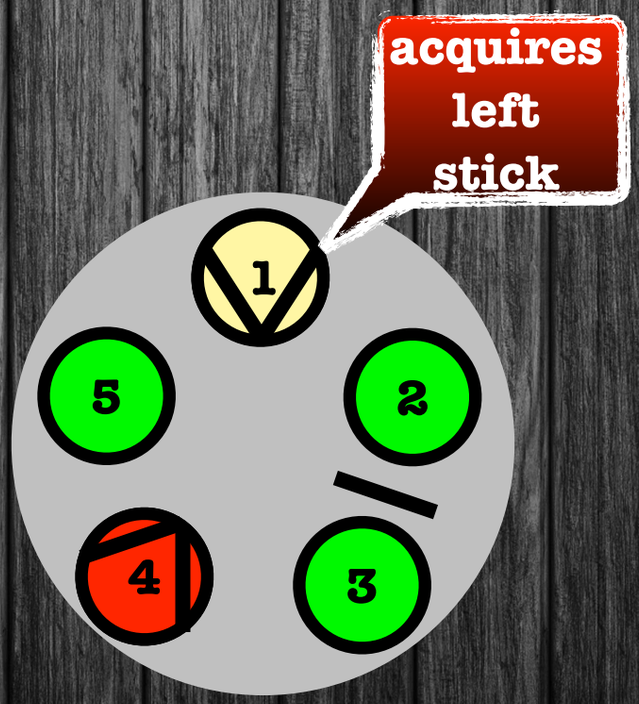
"abstract description of the expected behavior of the program"

Safety Properties

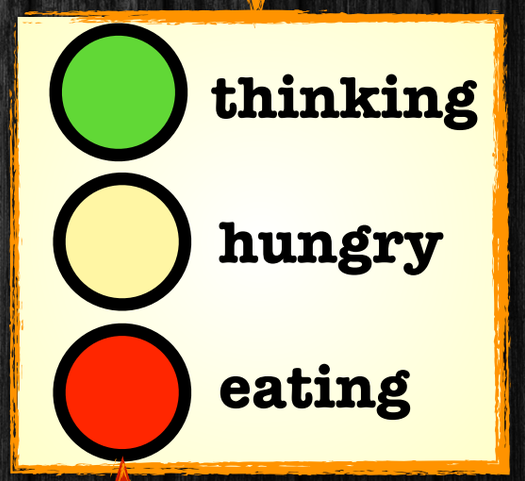
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

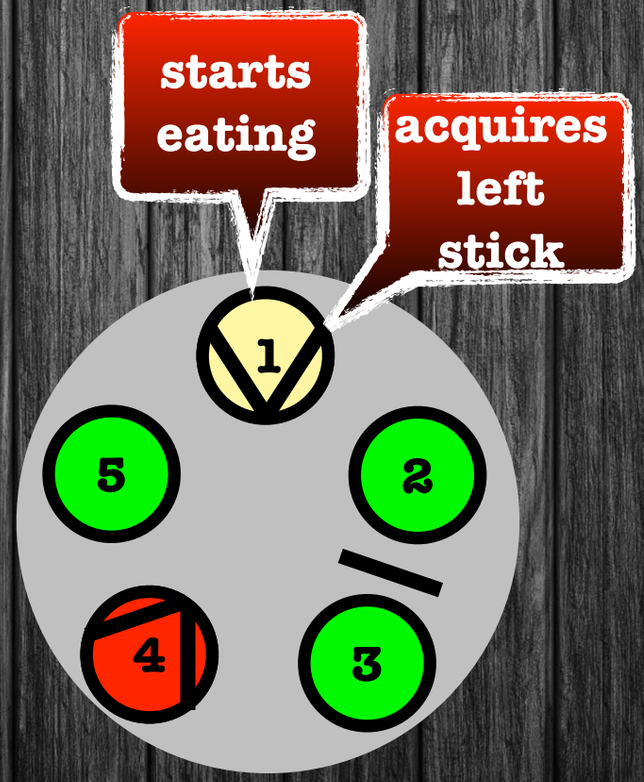
"abstract description of the expected behavior of the program"

Safety Properties

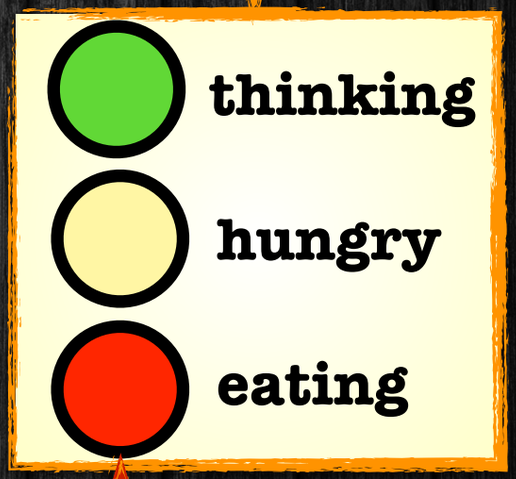
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

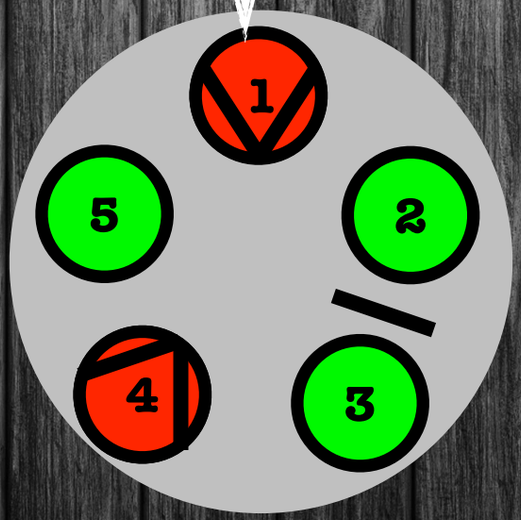
"abstract description of the expected behavior of the program"

Safety Properties

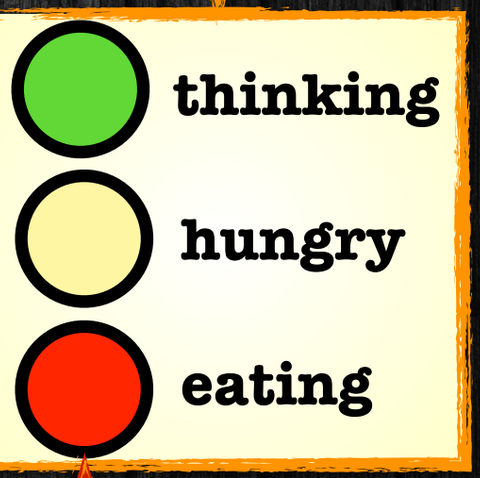
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

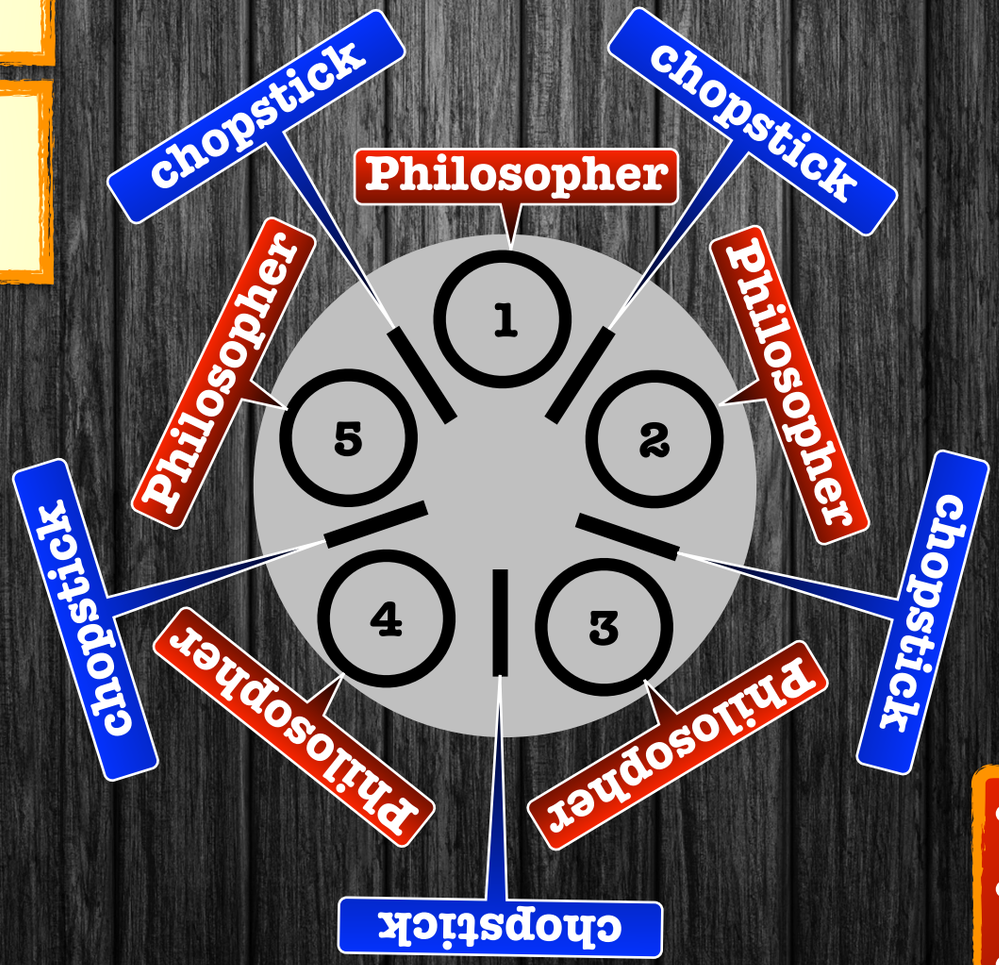
"abstract description of the expected behavior of the program"

Safety Properties

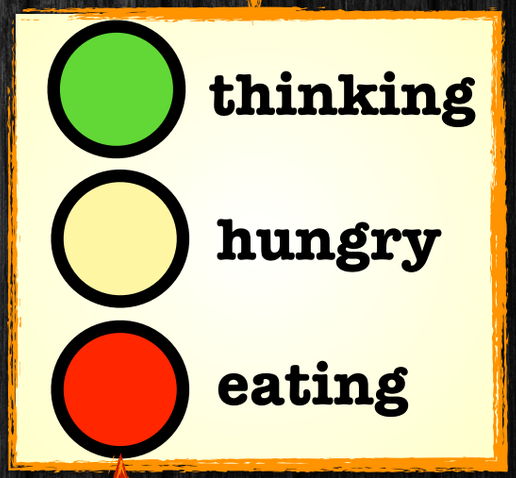
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

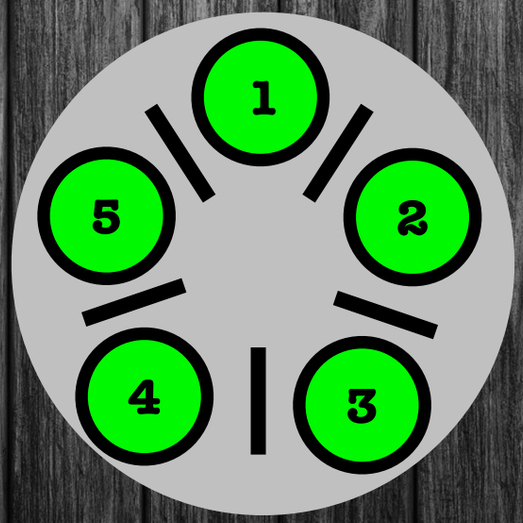
"abstract description of the expected behavior of the program"

Safety Properties

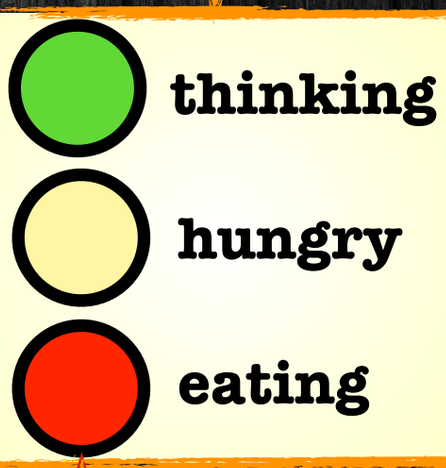
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

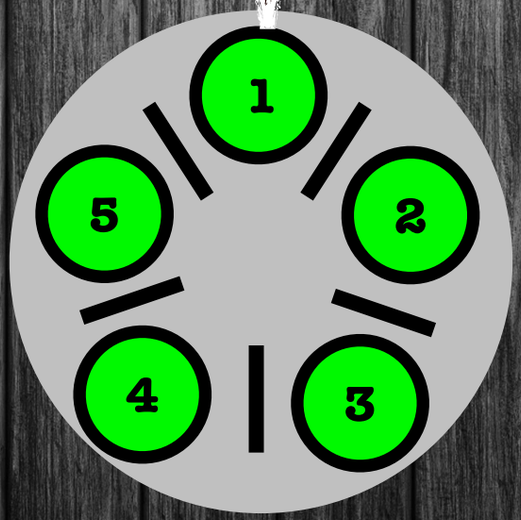
Safety Properties

"nothing bad will happen during the execution of the program"

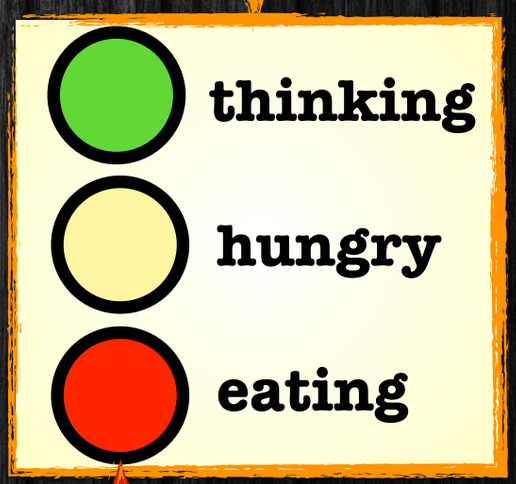
Examples

The Dining Philosophers

becomes hungry



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

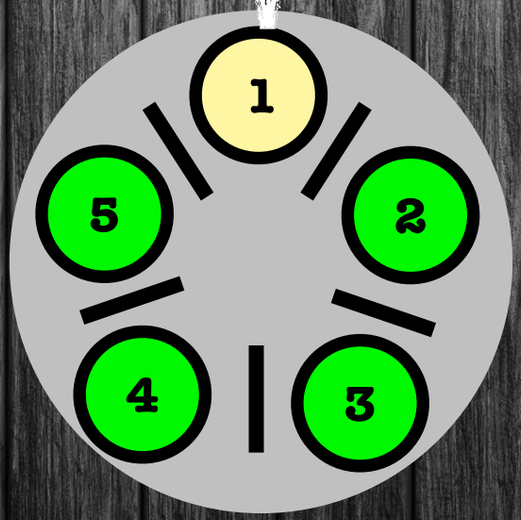
Safety Properties

"nothing bad will happen during the execution of the program"

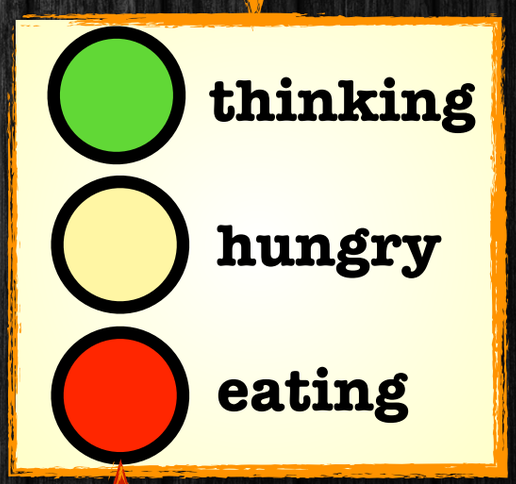
Examples

The Dining Philosophers

becomes hungry



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

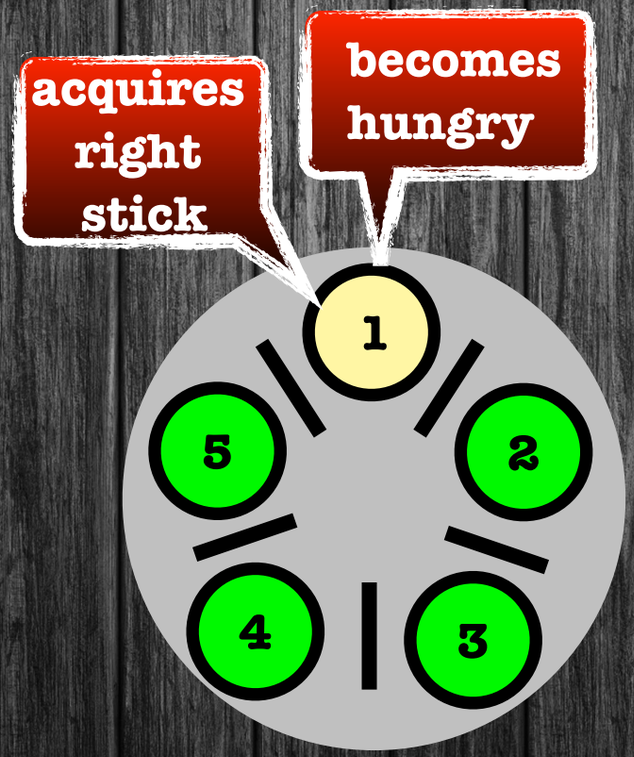
"abstract description of the expected behavior of the program"

Safety Properties

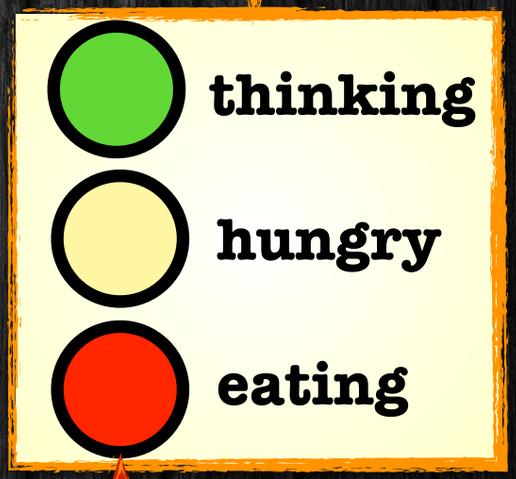
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

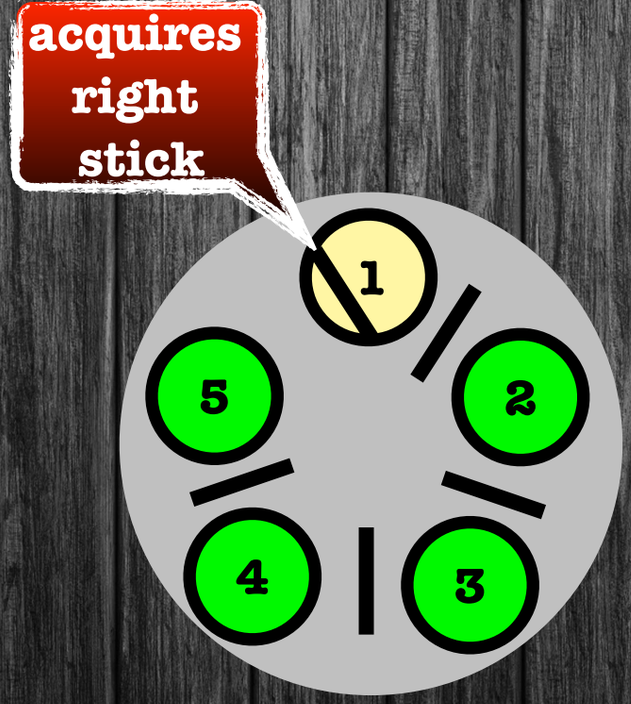
"abstract description of the expected behavior of the program"

Safety Properties

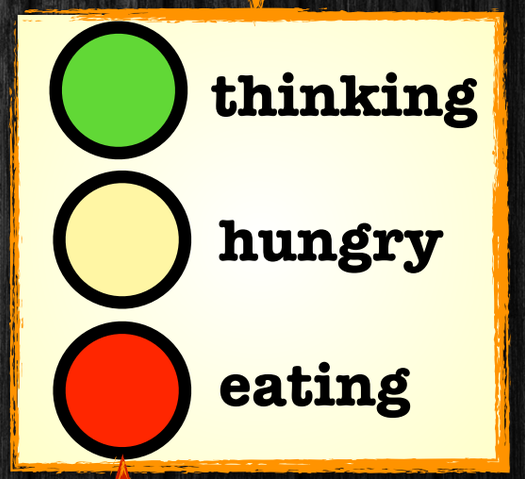
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

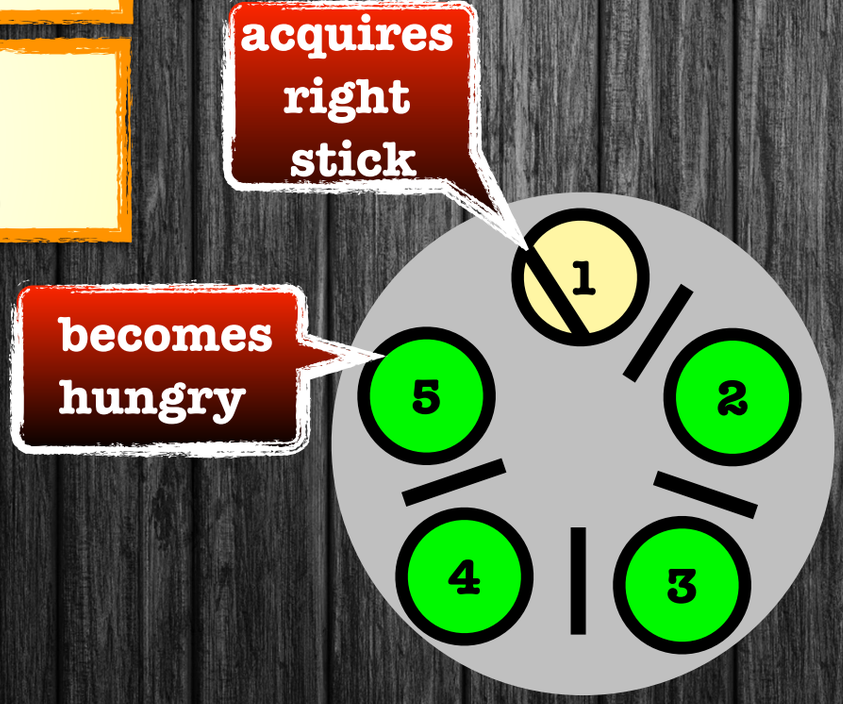
"abstract description of the expected behavior of the program"

Safety Properties

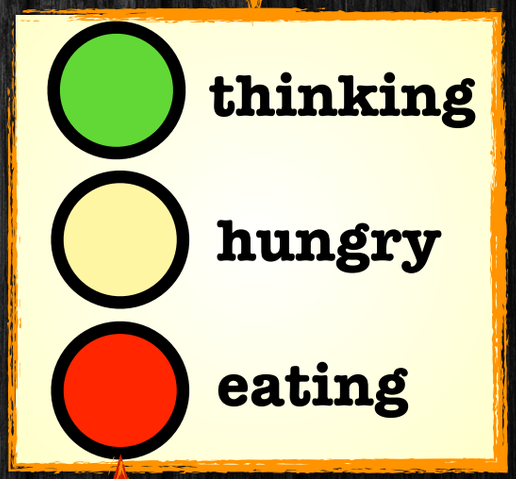
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

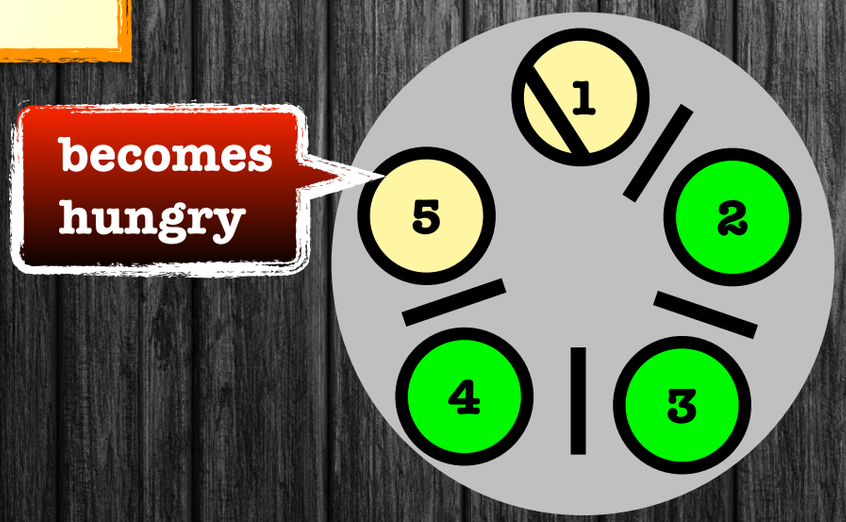
"abstract description of the expected behavior of the program"

Safety Properties

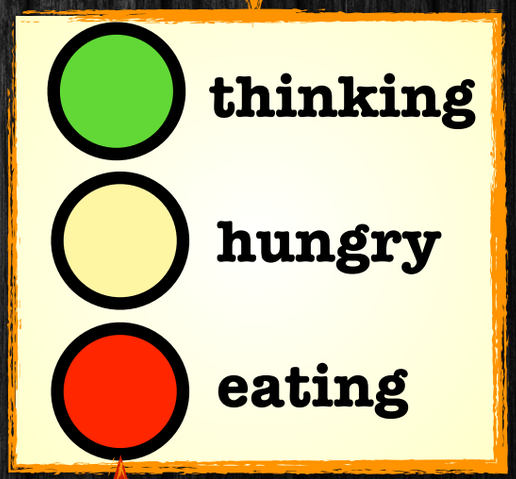
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

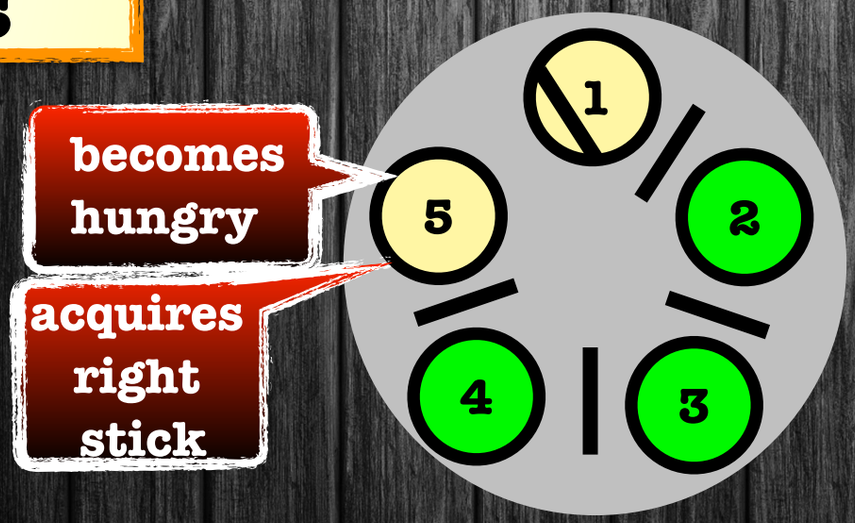
"abstract description of the expected behavior of the program"

Safety Properties

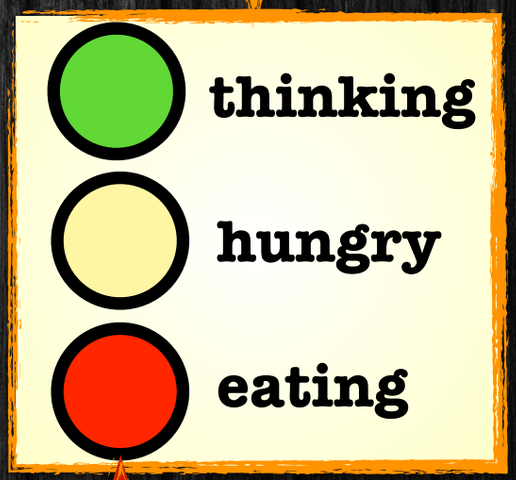
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

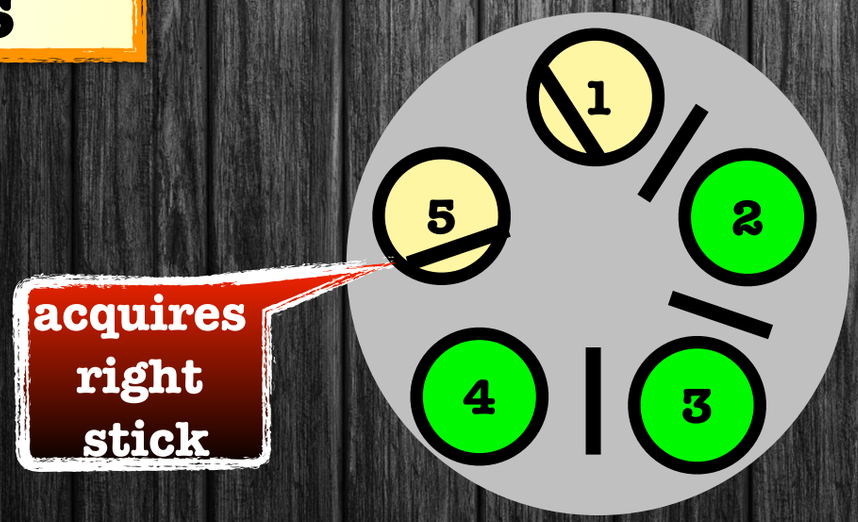
"abstract description of the expected behavior of the program"

Safety Properties

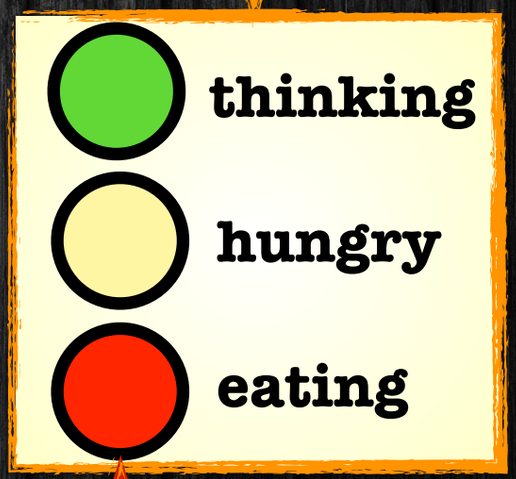
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

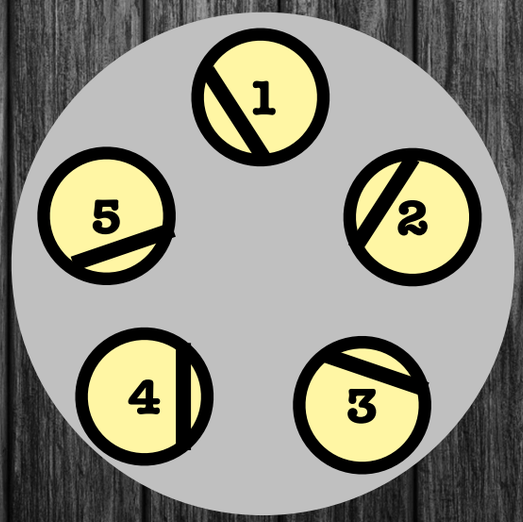
"abstract description of the expected behavior of the program"

Safety Properties

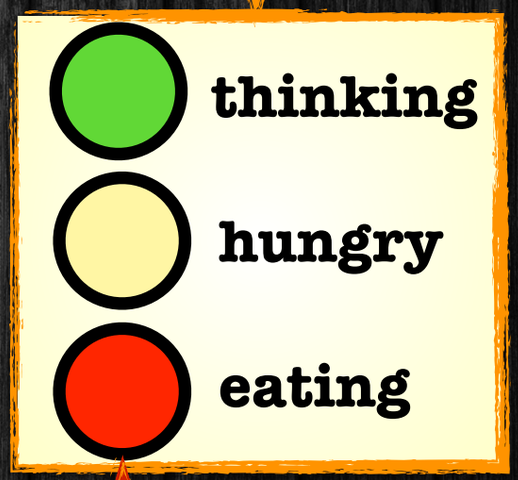
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

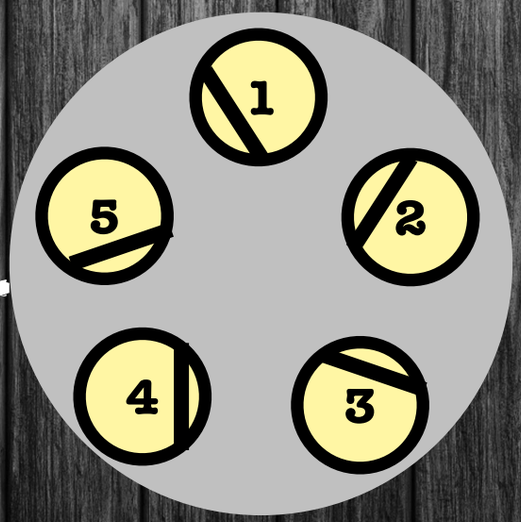
Safety Properties

"nothing bad will happen during the execution of the program"

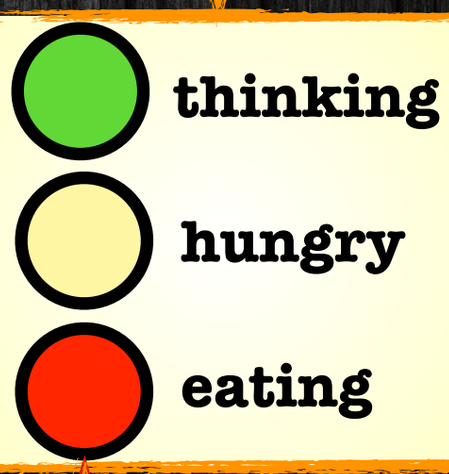
Examples

The Dining Philosophers

each philosopher waiting for the others



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

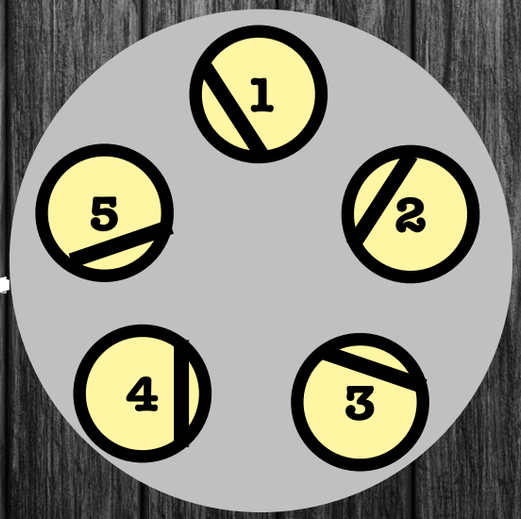
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

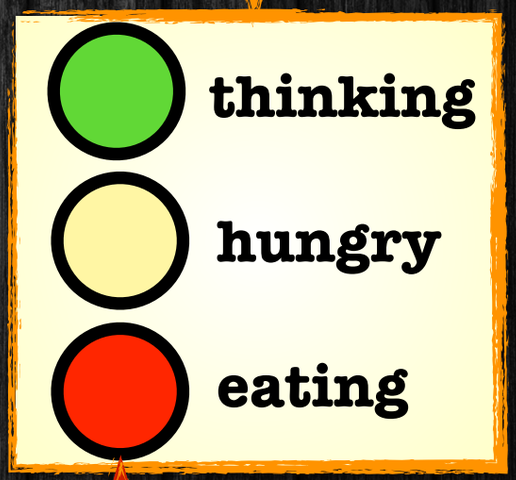
The Dining Philosophers



each philosopher waiting for the others

system cannot progress

philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

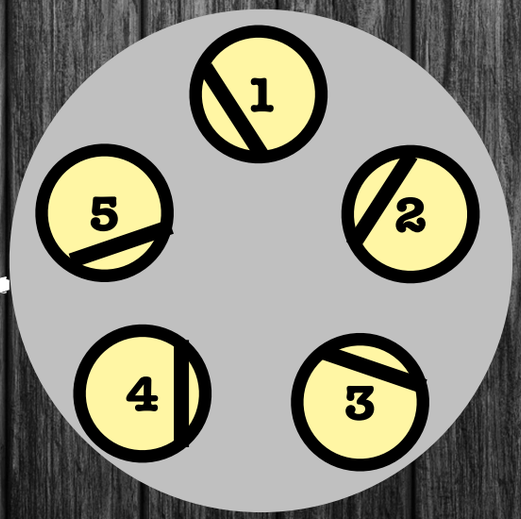
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers

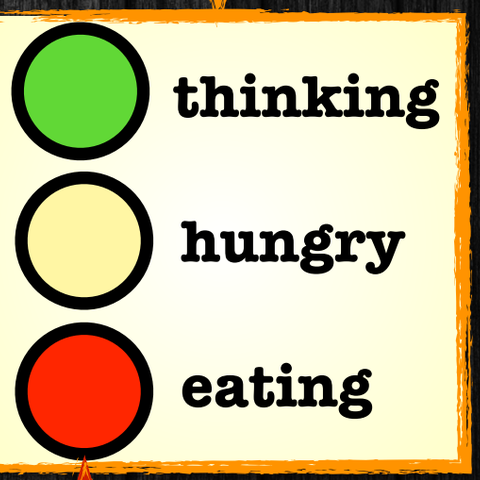


each philosopher waiting for the others

system cannot progress

Deadlock!

philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

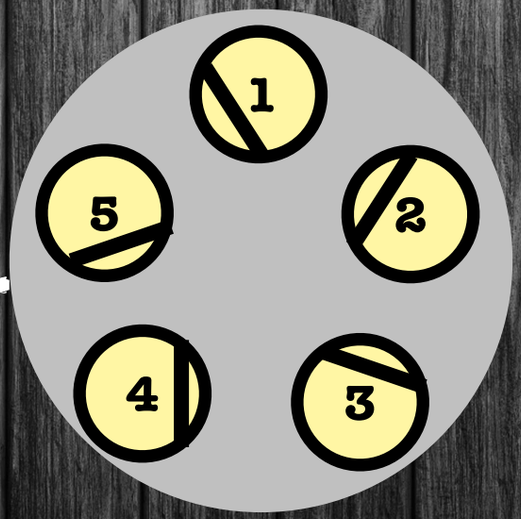
"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers

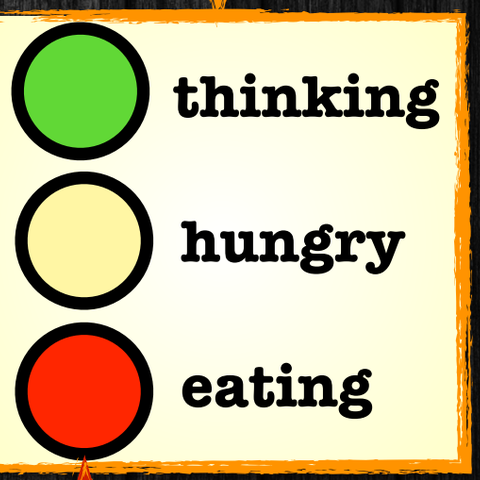


each philosopher waiting for the others

system cannot progress

Deadlock!

philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

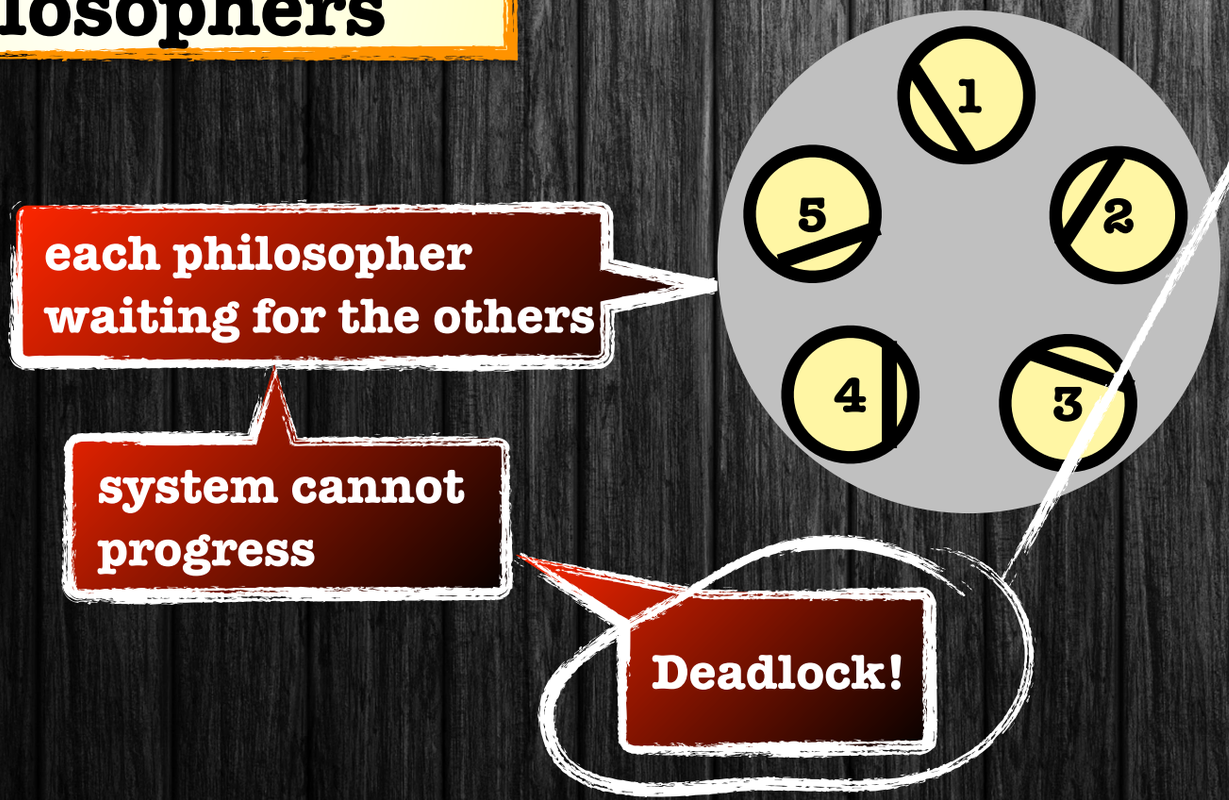
No Deadlock

Examples

The Dining Philosophers

philosopher states

- acquire right stick
- acquire left stick
- start eating



each philosopher waiting for the others

system cannot progress

Deadlock!

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

No Deadlock

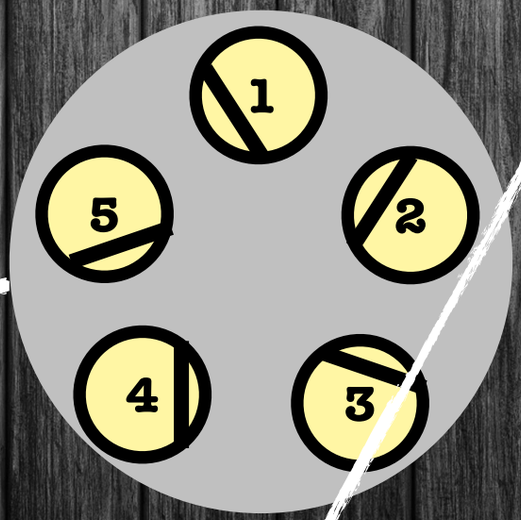
Examples

The Dining Philosophers

philosopher states

- thinking
- hungry
- eating

- acquire right stick
- acquire left stick
- start eating



each philosopher waiting for the others

system cannot progress

Deadlock!

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

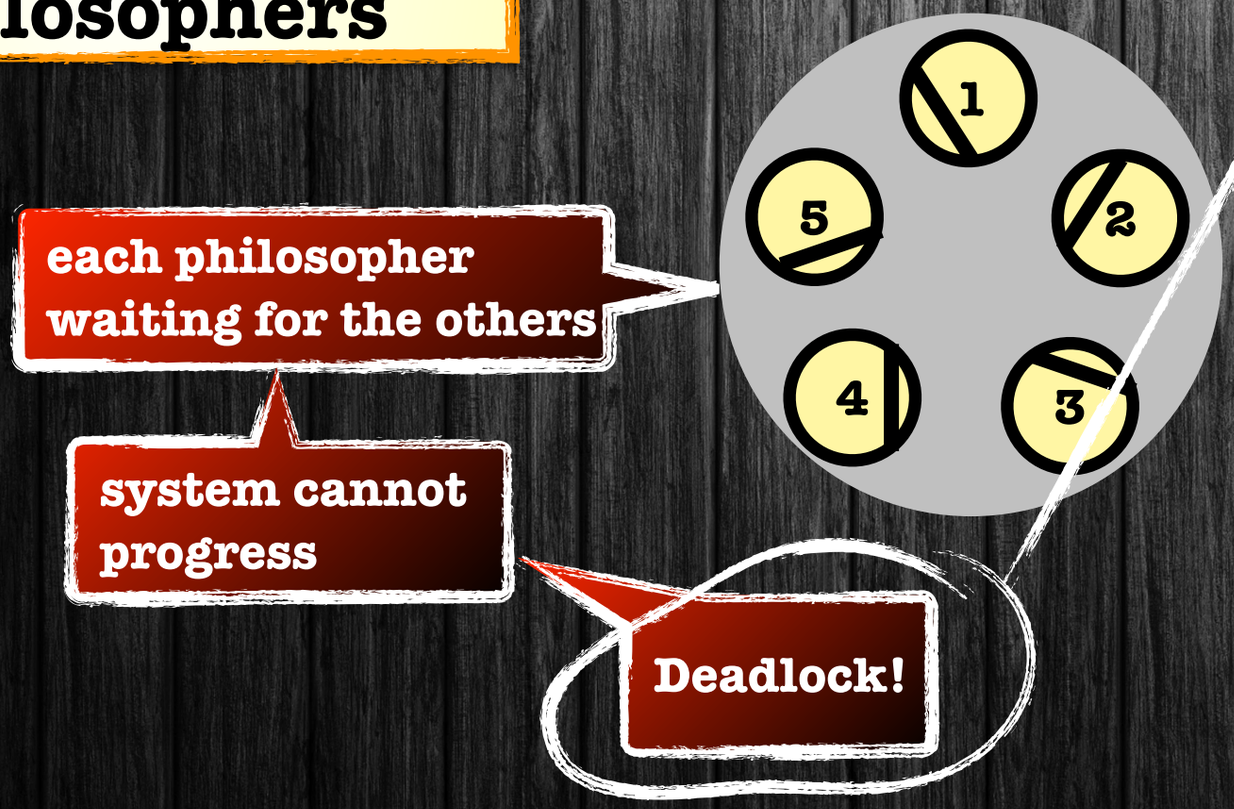
"nothing bad will happen during the execution of the program"

No Deadlock

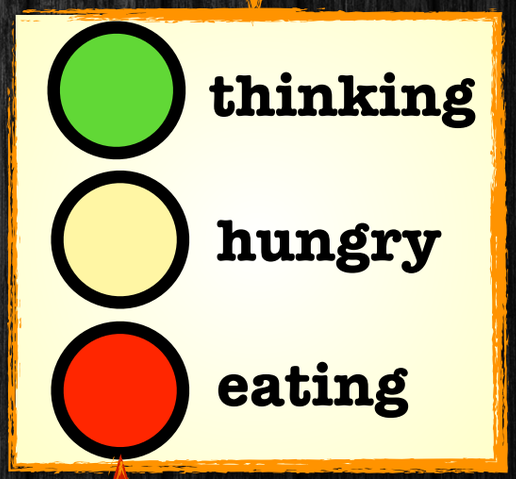
Examples

The Dining Philosophers

Reachability of bad states



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

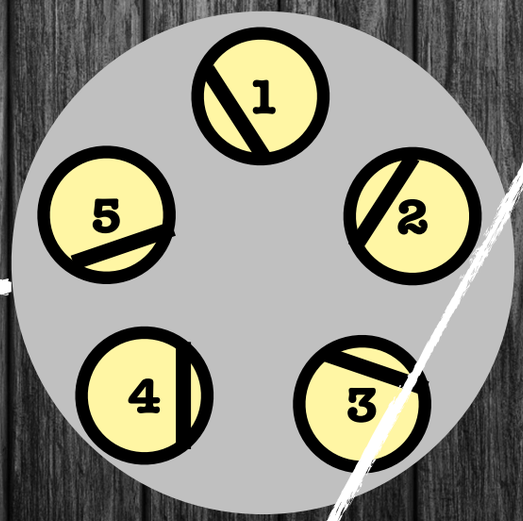
"nothing bad will happen during the execution of the program"

No Deadlock

Examples

The Dining Philosophers

Reachability of bad states

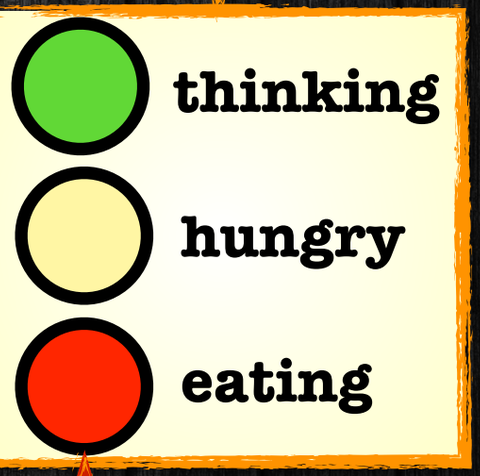


each philosopher waiting for the others

system cannot progress

Deadlock!

philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

"we will not reach a deadlock state"

No Deadlock

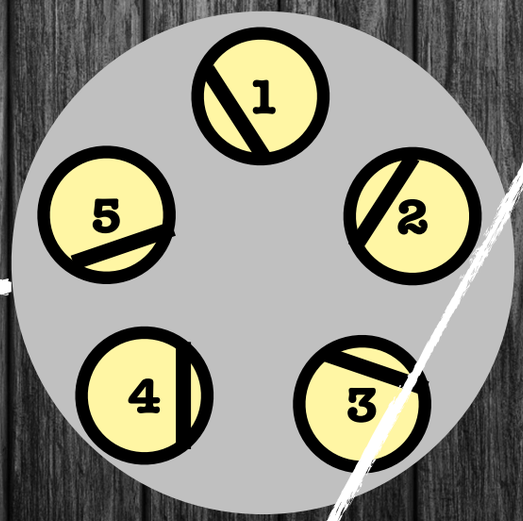
"nothing bad will happen during the execution of the program"

Safety Properties

Examples

The Dining Philosophers

Reachability of bad states



each philosopher waiting for the others

system cannot progress

Deadlock!

philosopher states

- thinking
- hungry
- eating

- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

"we will not reach a deadlock state"

No Deadlock

"nothing bad will happen during the execution of the program"

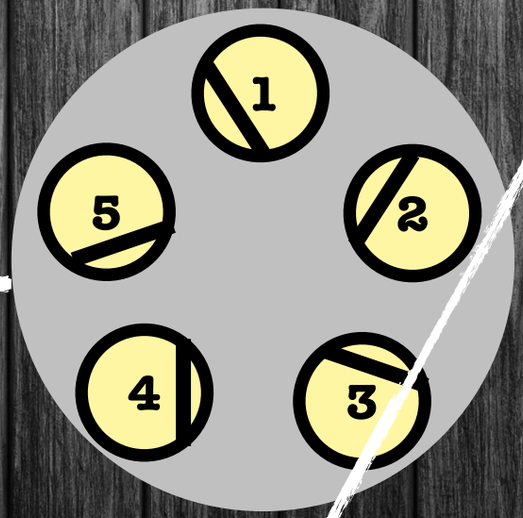
Safety Properties

Examples

The Dining Philosophers

Reachability of bad states

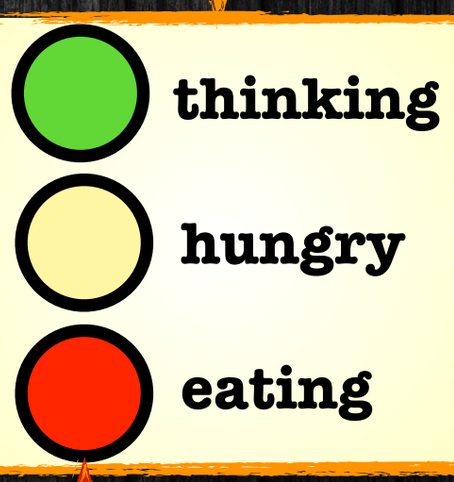
philosopher states



each philosopher waiting for the others

system cannot progress

Deadlock!



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

"we will not reach a deadlock state"

No Deadlock

"nothing bad will happen during the execution of the program"

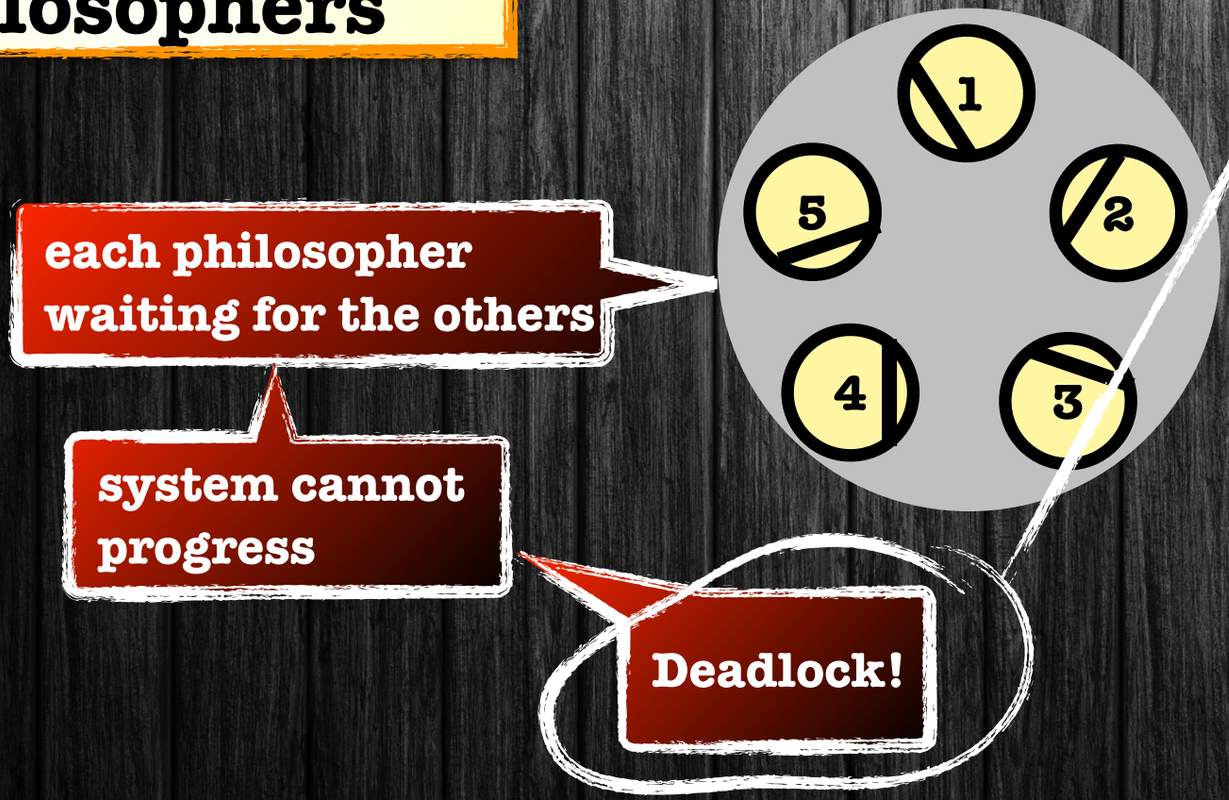
Safety Properties

Examples

The Dining Philosophers

Reachability of bad states

philosopher states



each philosopher waiting for the others

system cannot progress

Deadlock!

- thinking
- hungry
- eating

- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

"we will not reach a deadlock state"

"nothing bad will happen during the execution of the program"

No Deadlock

concurrent setting

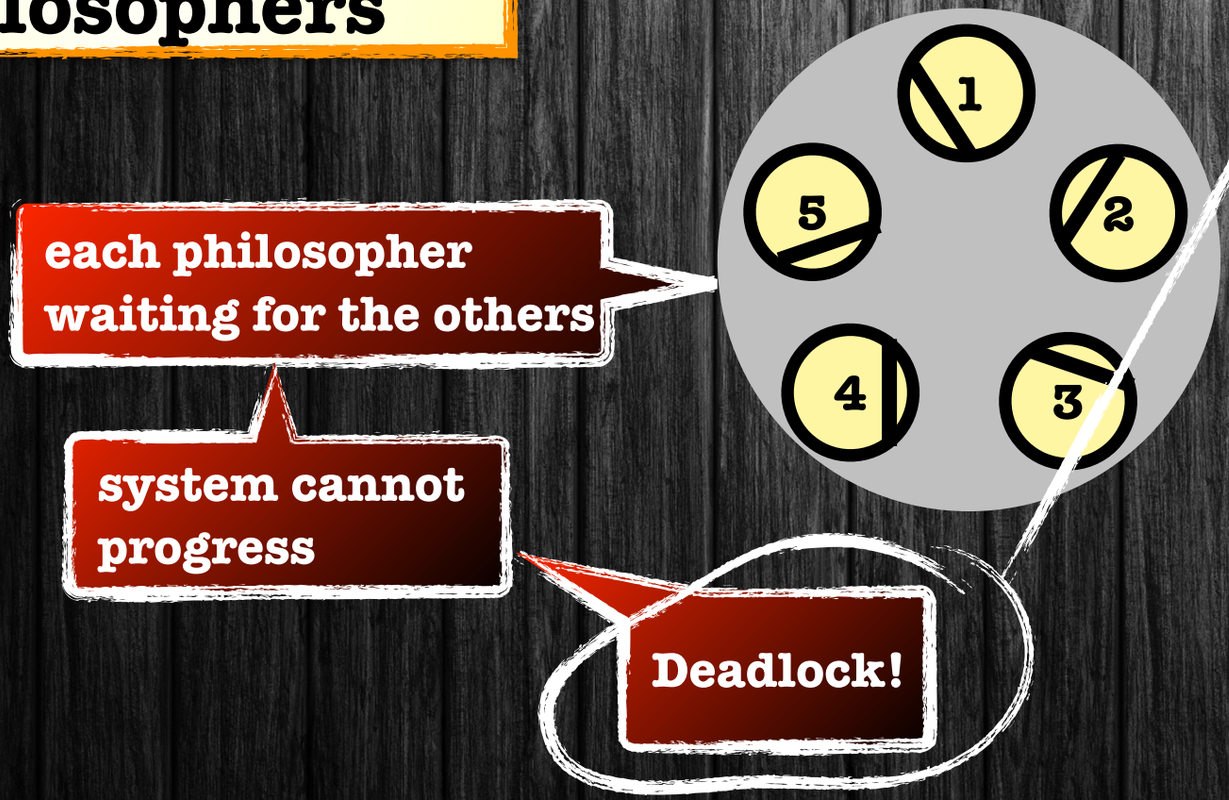
Safety Properties

Examples

The Dining Philosophers

Reachability of bad states

philosopher states



each philosopher waiting for the others

system cannot progress

Deadlock!

- thinking
- hungry
- eating

- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Sorting

Reachability of bad states

list $a=[a_1,a_2,\ldots,a_n]$ of integers — input

list $b=[b_1,b_2,\ldots,b_n]$ of integers — output

- b is a permutation of a
- b is sorted

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

input: [3,7,8,5,1]

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

Reachability of bad states

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

- b is a permutation of a
- b is sorted

output

Reachability of bad states

input

[3,7,8,5,1]

[1,3,8,5,7]

output

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

output

- b is a permutation of a
- b is sorted

Reachability of bad states

input

[3,7,8,5,1]

[1,3,8,5,7]

output

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers — input

list $b=[b_1,b_2,\ldots,b_n]$ of integers — output

- b is a permutation of a
- b is sorted

Reachability of bad states

input: [3,7,8,5,1]

output: [1,3,8,5,7]

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers — input

list $b=[b_1,b_2,\ldots,b_n]$ of integers — output

- b is a permutation of a
- b is sorted

Reachability of bad states

input: [3,7,8,5,1]

output: [1,3,8,5,7]

sequential setting

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Reachability of bad states

Specification

- safety properties
- liveness properties

concurrent setting

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Reachability of bad states

Specification

- safety properties
- liveness properties

concurrent setting

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Reachability of bad states

Mutual Exclusion

Specification

- safety properties
- liveness properties

concurrent setting

"abstract description of the expected behavior of the program"

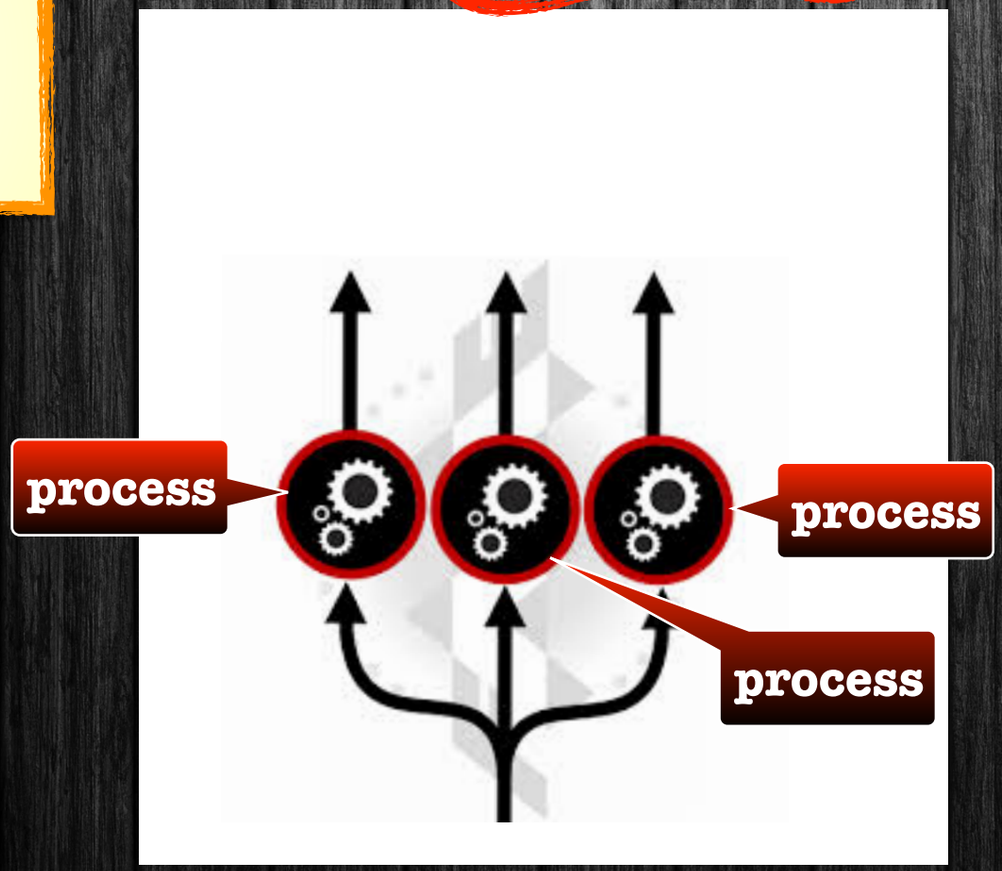
Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Reachability of bad states

Mutual Exclusion



Specification

- safety properties
- liveness properties

concurrent setting

"abstract description of the expected behavior of the program"

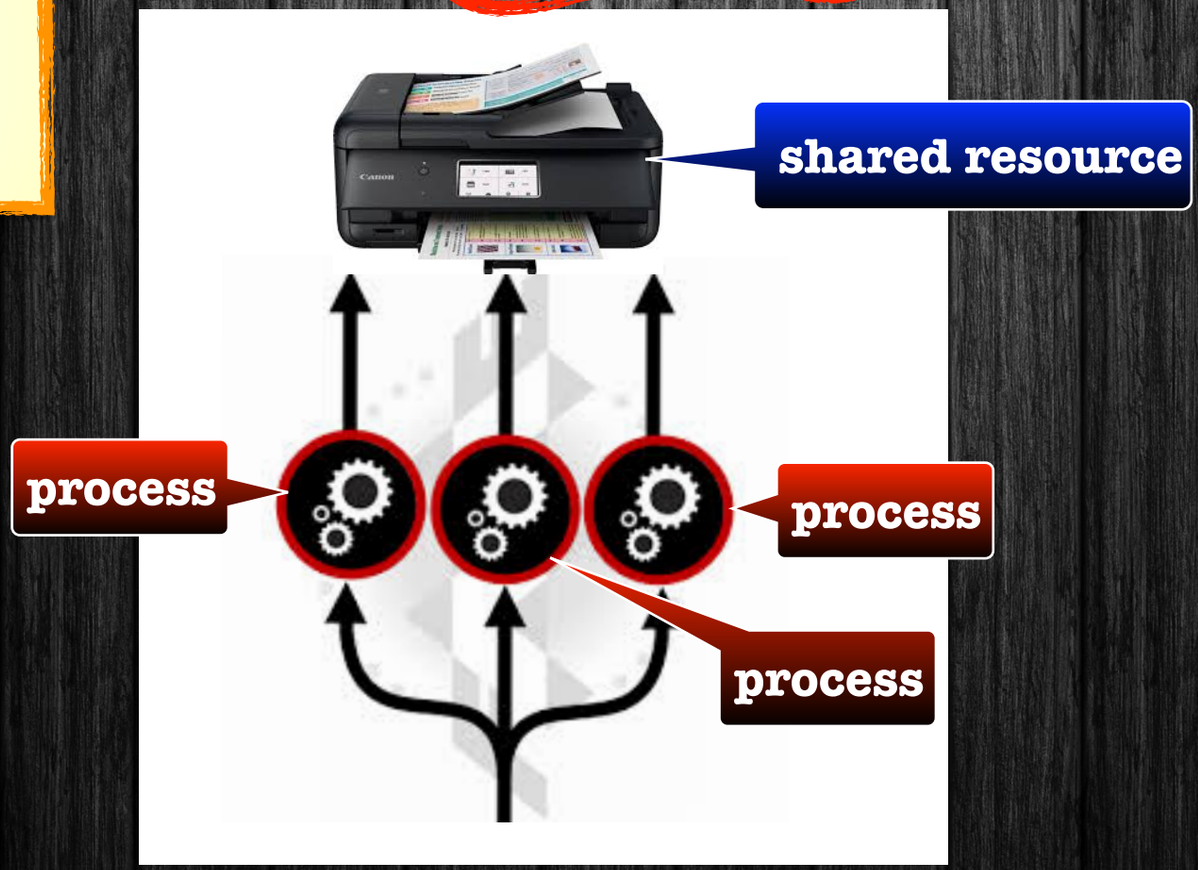
Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Mutual Exclusion

Reachability of bad states



Specification

- safety properties
- liveness properties

concurrent setting

"abstract description of the expected behavior of the program"

"at most one process can access the shared resource at any point"

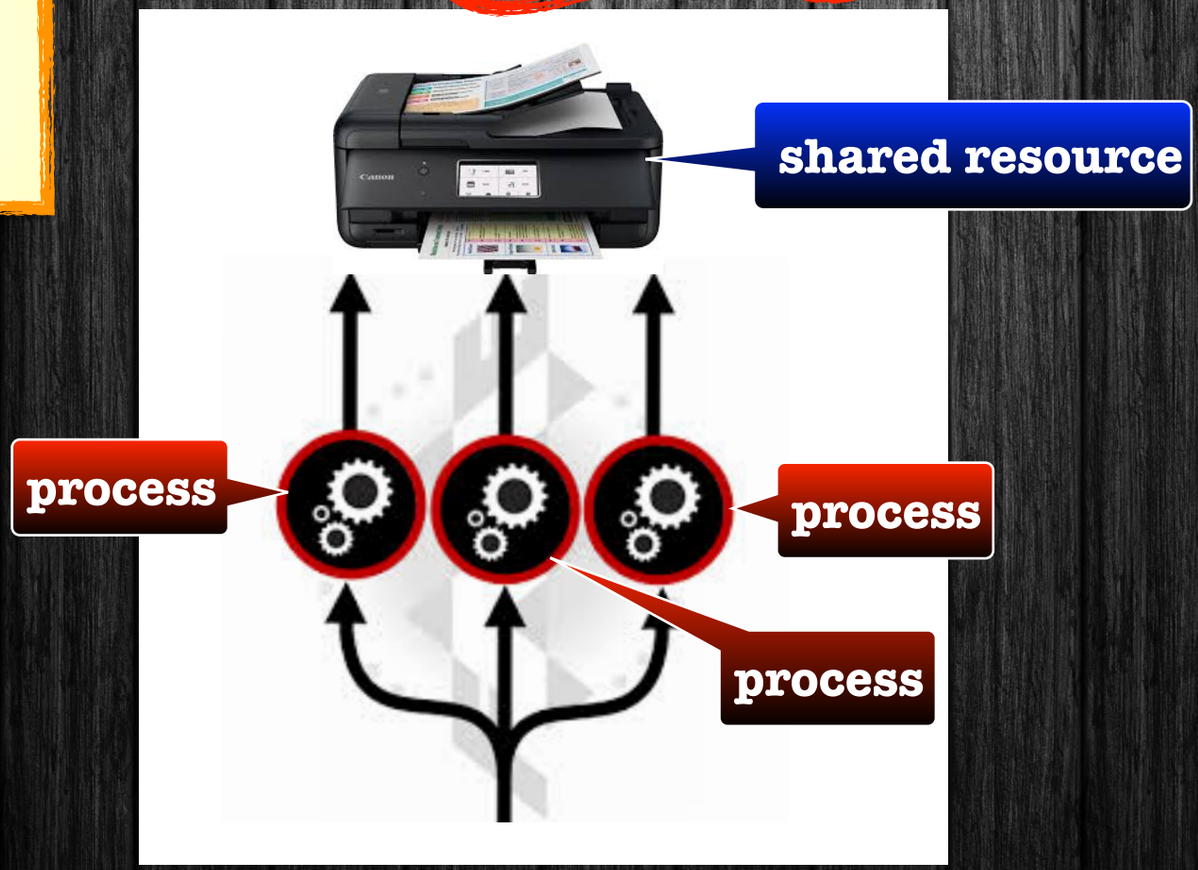
Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Reachability of bad states

Mutual Exclusion



Specification

- safety properties
- liveness properties

concurrent setting

"abstract description of the expected behavior of the program"

"at most one process can access the shared resource at any point"

Safety Properties

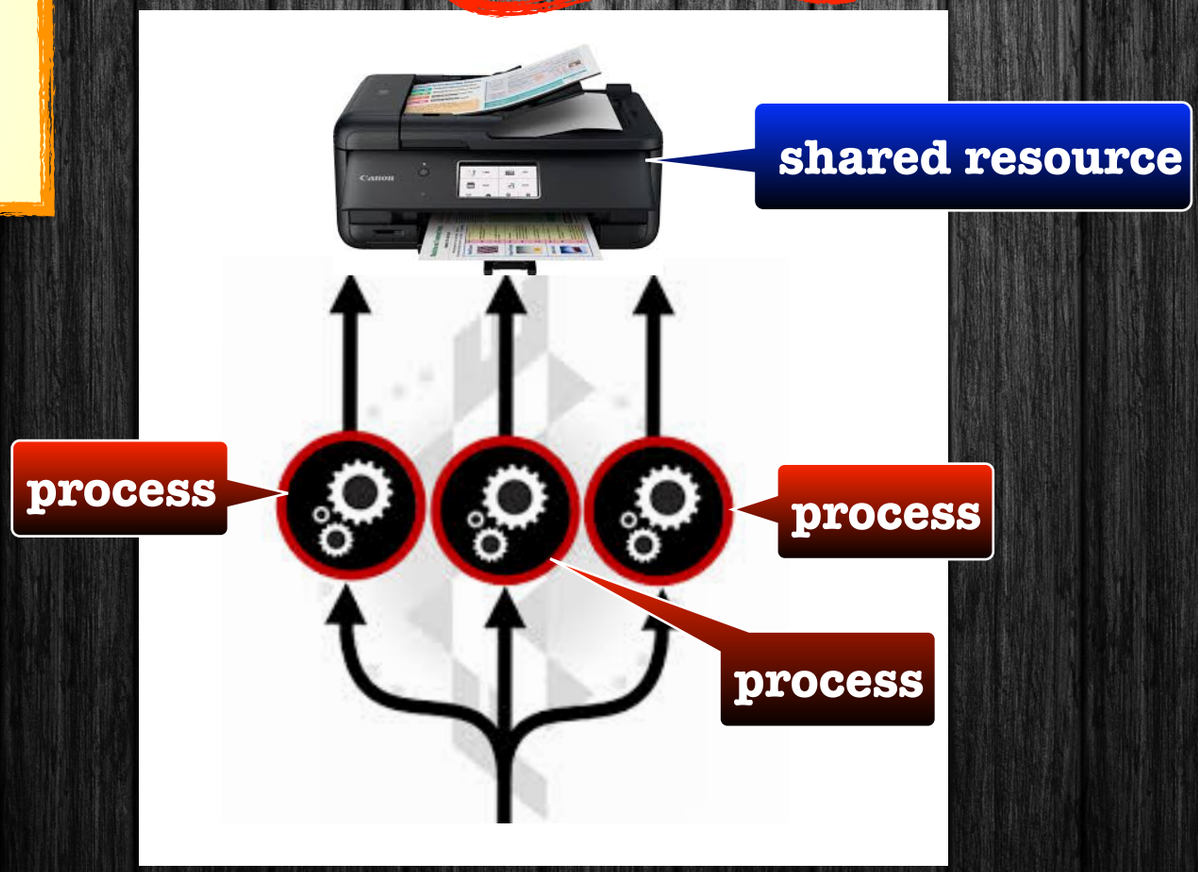
"nothing bad will happen during the execution of the program"

Examples

Mutual Exclusion

Reachability of bad states

"two or more processes access the shared resource at some point"



Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Reachability of bad states

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

concurrent setting

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Reachability of bad states

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Reachability of bad states

concurrent setting

this course

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

concurrent setting

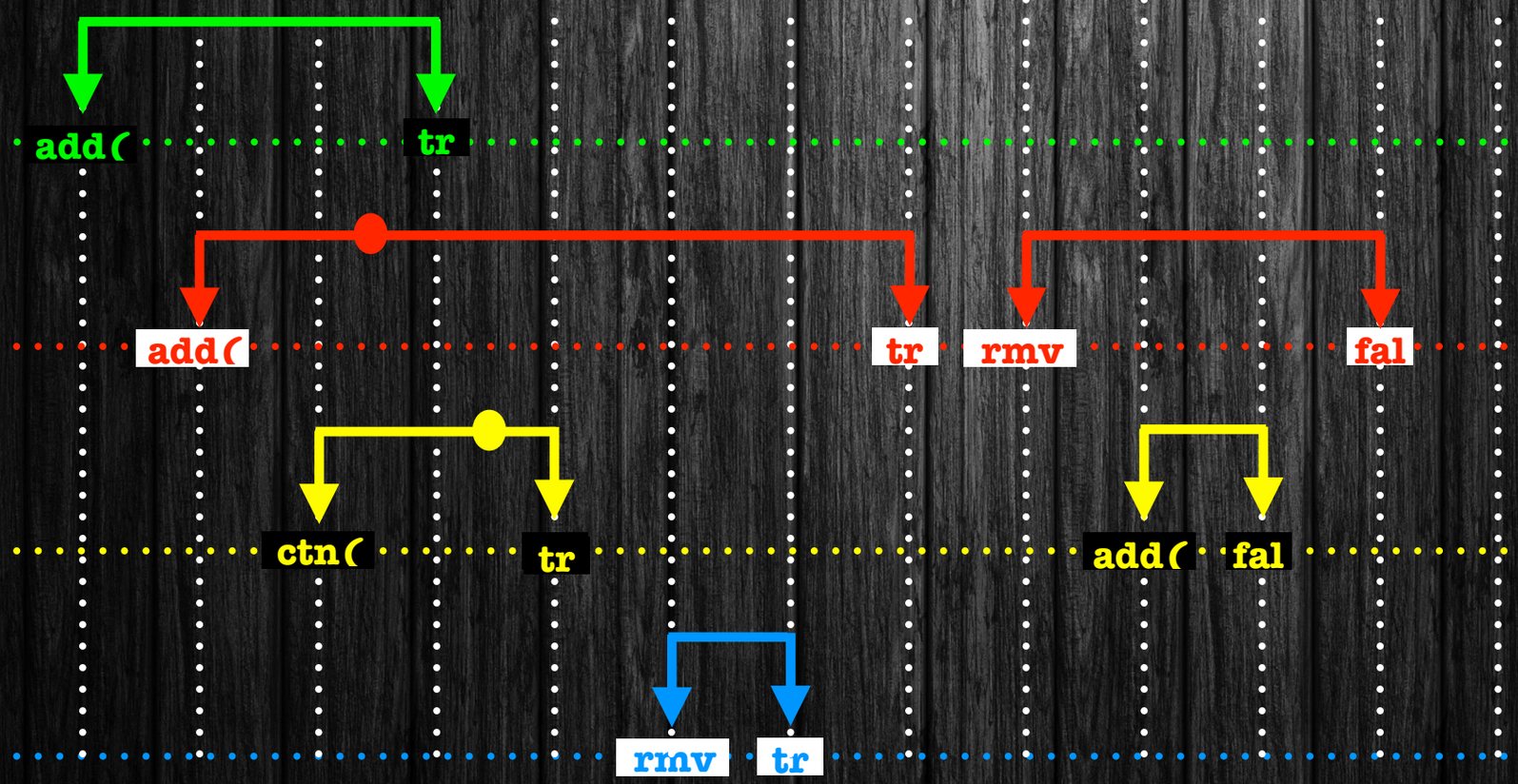
this course

Safety Properties

"nothing bad will happen during the execution of the program"

Examples

Reachability of bad states



Linearizability

Overview

- Algorithms and Data Structures
- Concurrency
- Efficiency & Correctness
- **Concepts**
  - **Specifications**
  - **Safety Properties**
  - **Liveness Properties**

Overview

- Algorithms and Data Structures
- Concurrency
- Efficiency & Correctness
- **Concepts**
  - **Specifications**
  - **Safety Properties**
  - **Liveness Properties**

Overview

- Algorithms and Data St
- Concurrency
- Efficiency & Correctness
- **Concepts**
  - **Specifications**
  - **Safety Properties**
  - **Liveness Properties**

Not logged in Talk Contributions Create account Log in

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

Languages
Français
한국어
Edit links

Liveness

From Wikipedia, the free encyclopedia

In concurrent computing, **liveness** refers to a set of properties of concurrent systems, that require a system to make progress despite the fact that its concurrently executing components ("processes") may have to "take turns" in critical sections, parts of the program that cannot be simultaneously run by multiple processes.[1] Liveness guarantees are important properties in operating systems and distributed systems.[2]

More generally, a liveness property states that "something good will eventually occur", contrasting a safety property which states that "something bad does not occur". If a safety property is violated there is always a finite execution that shows the violation (the "bad" event occurring), but a liveness property cannot be violated in a finite execution of a distributed system because the "good" event might still occur at some later time. Eventual consistency is an example of a liveness property.[3] All linear time properties can be expressed as the intersection of safety and liveness properties.[4] Whereas the violation of a given safety property admits a finite witness, violation of liveness properties may be harder to establish as no finite witness can be used as proof.[5]

Contents [hide]
1 Forms of liveness
2 Liveness and safety
  2.1 Formal distinction
  2.2 Bounded bypass and bounded overtaking
3 See also
4 References

Forms of liveness [edit]

Several forms of liveness are recognized. The following ones are defined in terms of a multi-process system that has a critical section, protected by some mutual exclusion (mutex) device. All processes are assumed to correctly use the mutex; progress is defined as finishing execution of the critical section.

- Freedom from deadlock is a form of liveness, although a weak one. Consider a system with multiple processes and a single critical section, protected by some mutual exclusion device. Such a system is said to be deadlock-free if, when a group of processes is competing for access to the critical section at some point in time, then *some* process eventually makes progress at a later point in time. That process need not belong to the aforementioned group; it might have gained access at an earlier or even later moment.[6]
- Freedom from starvation (or "finite bypass") is a stronger liveness guarantee than deadlock-freedom. It states that *all* processes vying for access to the critical region eventually make progress. Any starvation-free system is also deadlock-free.[6]
- Stronger still is the requirement of bounded bypass. This means that, if n processes are competing for access to the critical region, then

Specification

- safety properties
- liveness properties

Liveness Properties

Specification

- safety properties
- liveness properties

Liveness Properties

Specification

- safety properties
- liveness properties

Liveness Properties

"something good will eventually happen during the execution of the program"

Specification

- safety properties
- liveness properties

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Specification

- safety properties
- liveness properties

sequential setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Specification

- safety properties
- liveness properties

sequential setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Termination

Specification

- safety properties
- liveness properties

sequential setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Termination

algorithms satisfying the "sorting" specification

- Quick Sort
- Merge Sort
- Heap Sort
- etc

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

output

- b is a permutation of a
- b is sorted

$b_1 \leq b_2 \leq \ldots \leq b_n$

[3,0,5,8,7]

[0,3,5,7,8]

sorted permutation

Specification

- safety properties
- liveness properties

sequential setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Termination

algorithms satisfying the "sorting" specification

"algorithm should terminate"

- Quick Sort
- Merge Sort
- Heap Sort
- etc

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

output

- b is a permutation of a
- b is sorted

$b_1 \leq b_2 \leq \ldots \leq b_n$

[3,0,5,8,7]

[0,3,5,7,8]

sorted permutation

Specification

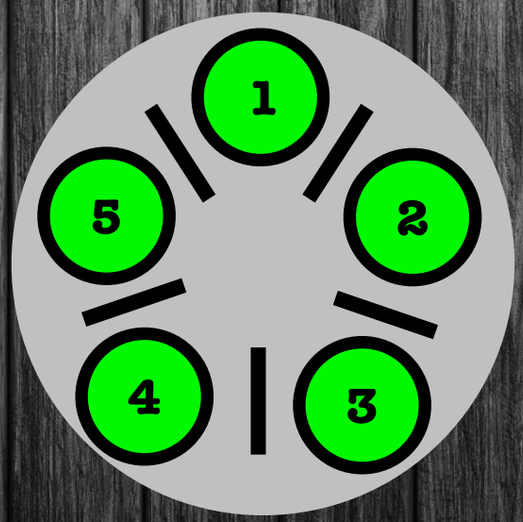
- safety properties
- liveness properties

concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples



Specification

- safety properties
- liveness properties

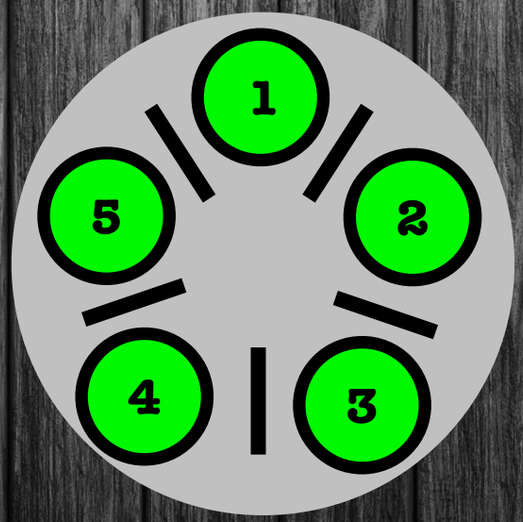
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

concurrent setting

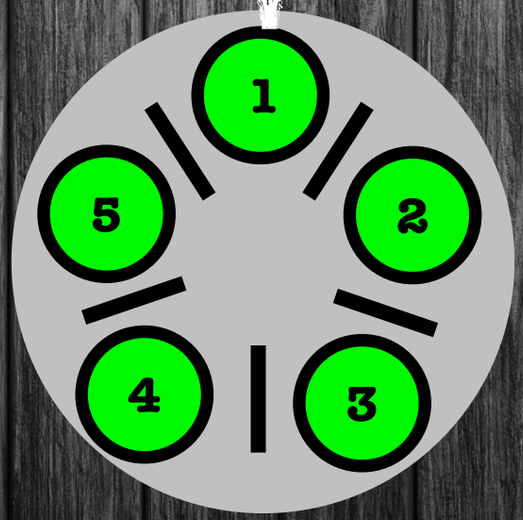
Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom

becomes hungry



Specification

- safety properties
- liveness properties

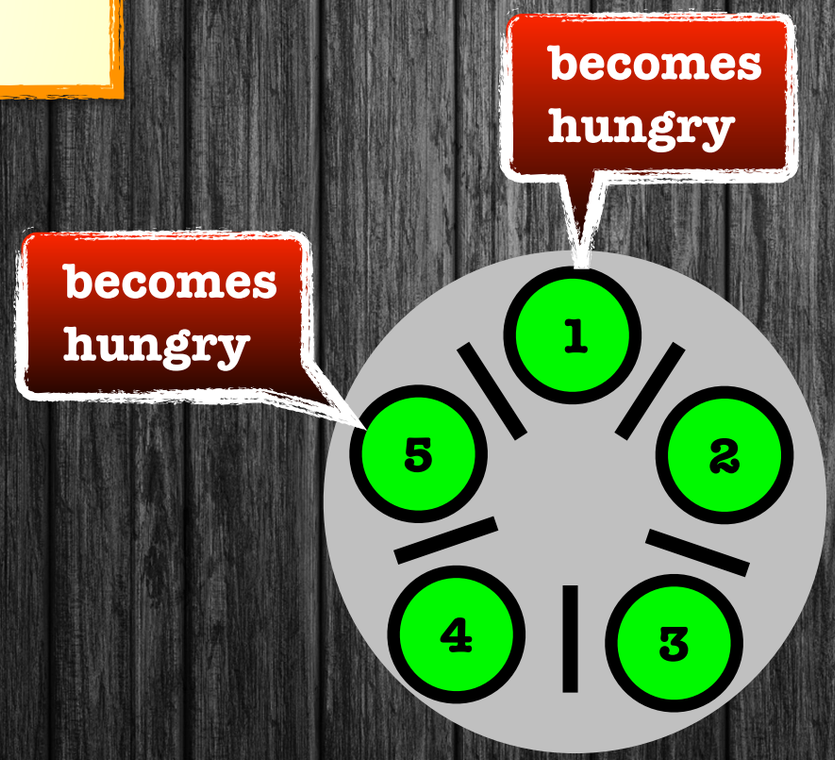
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

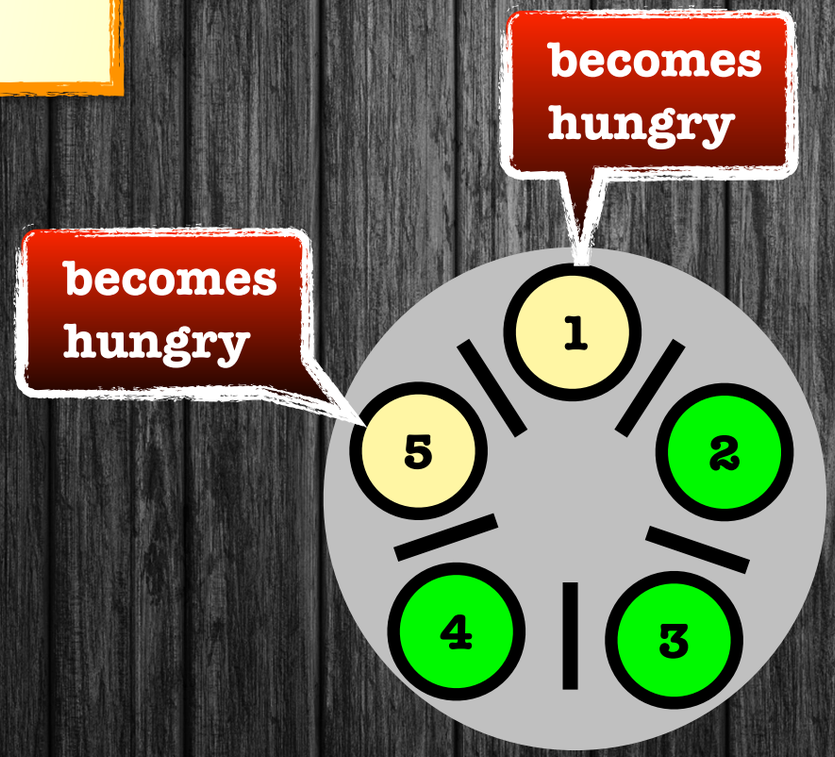
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

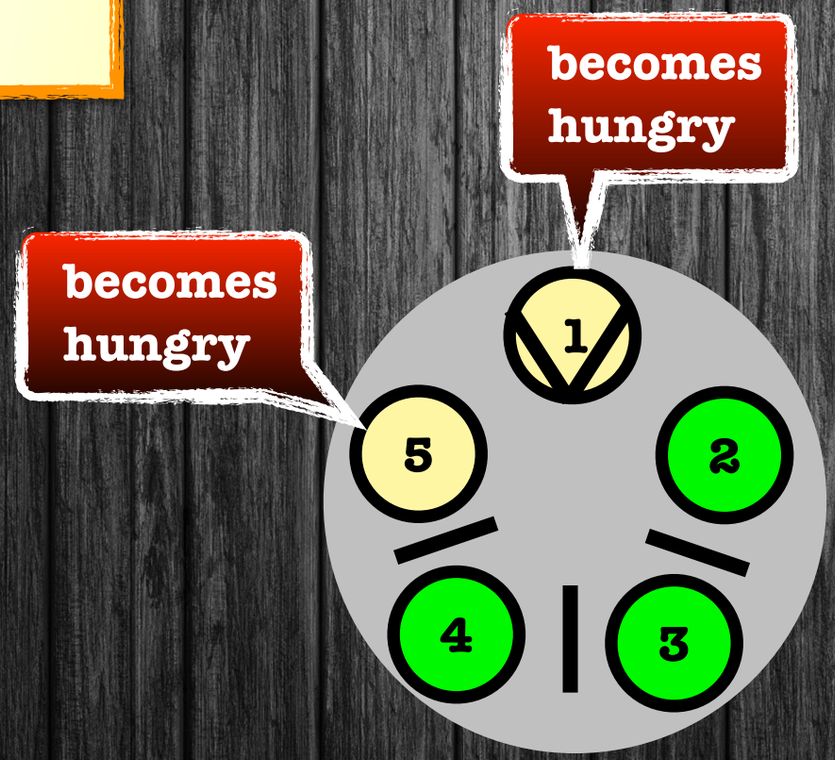
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

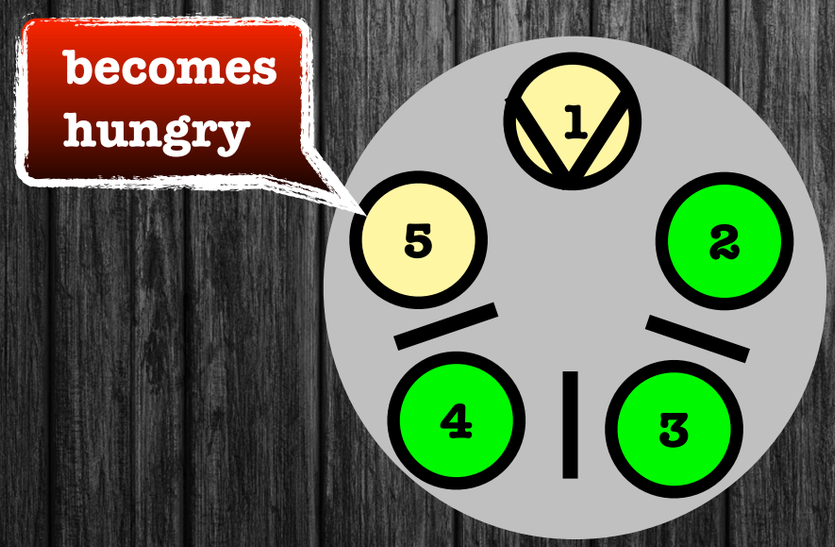
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

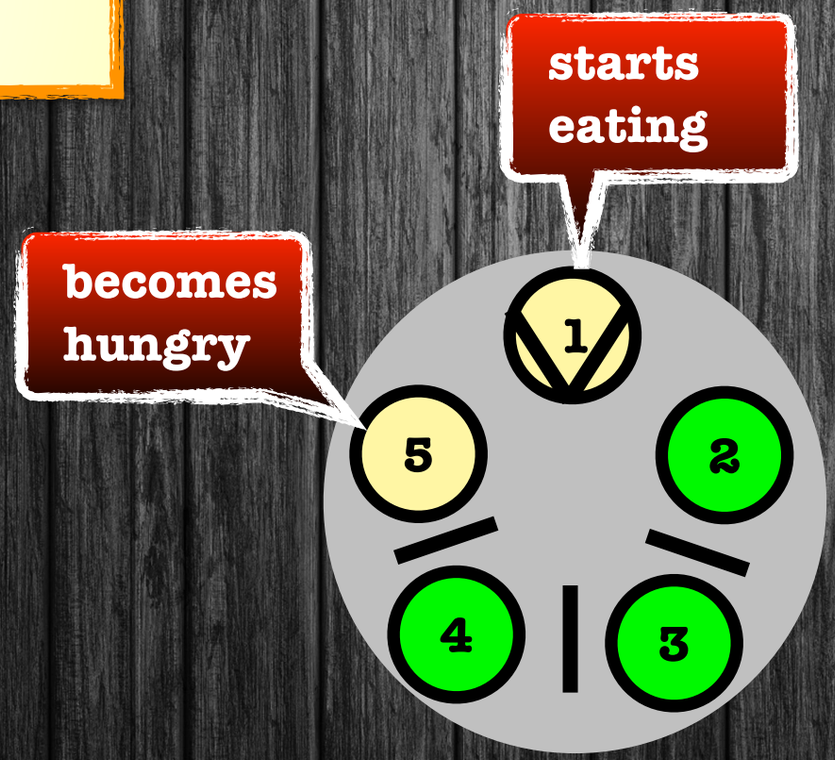
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

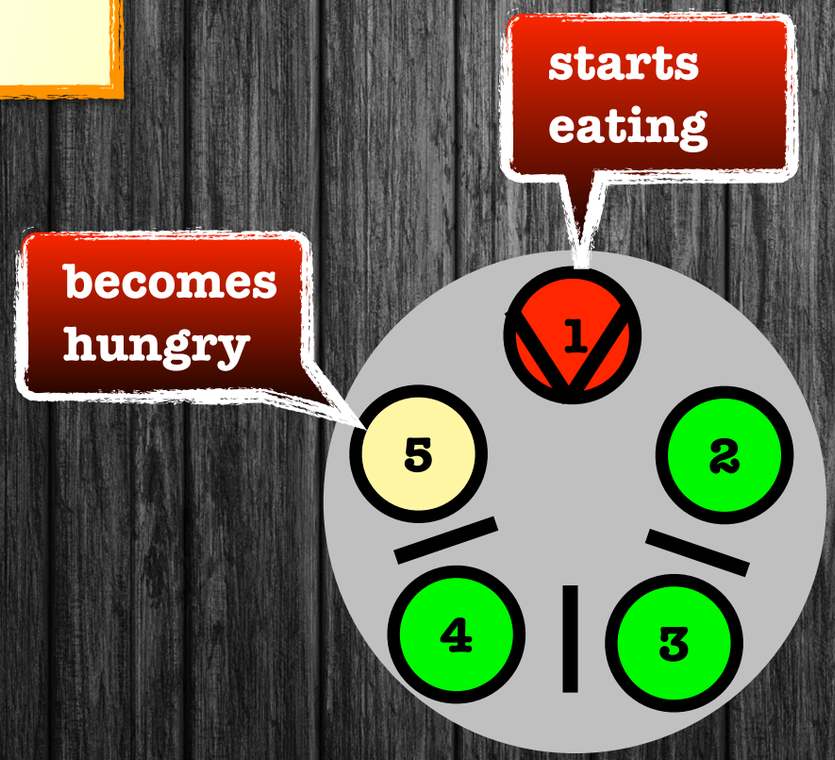
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

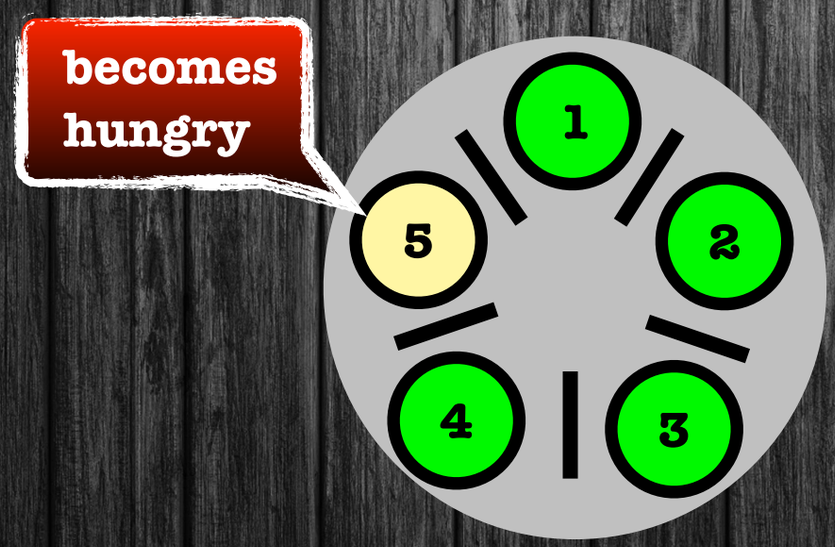
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

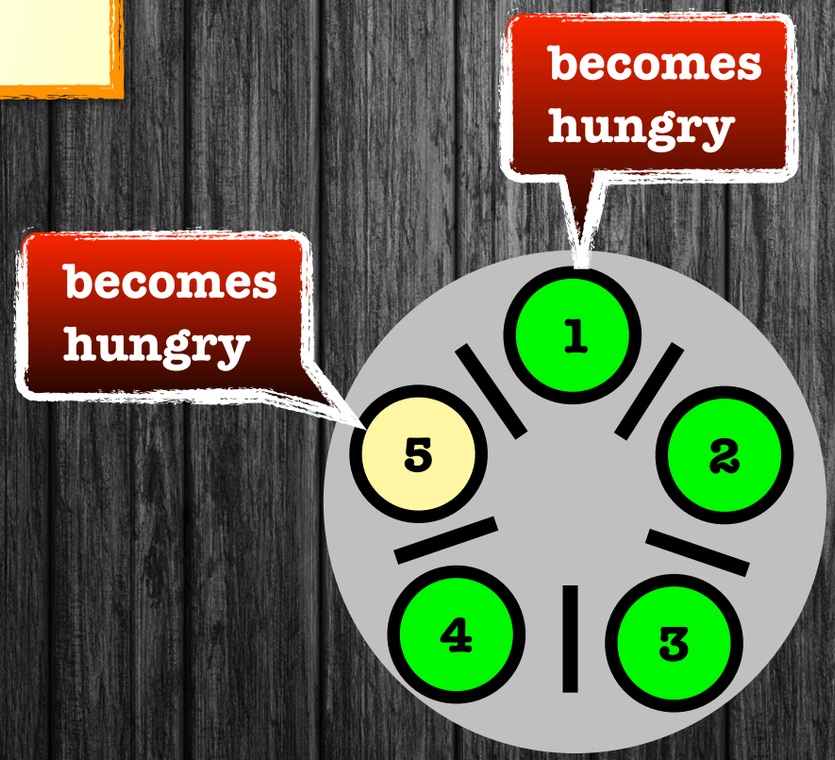
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

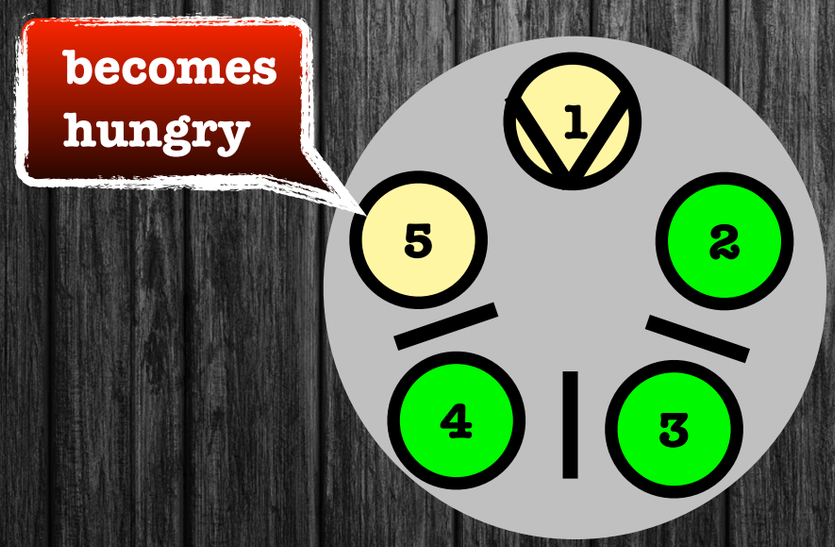
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

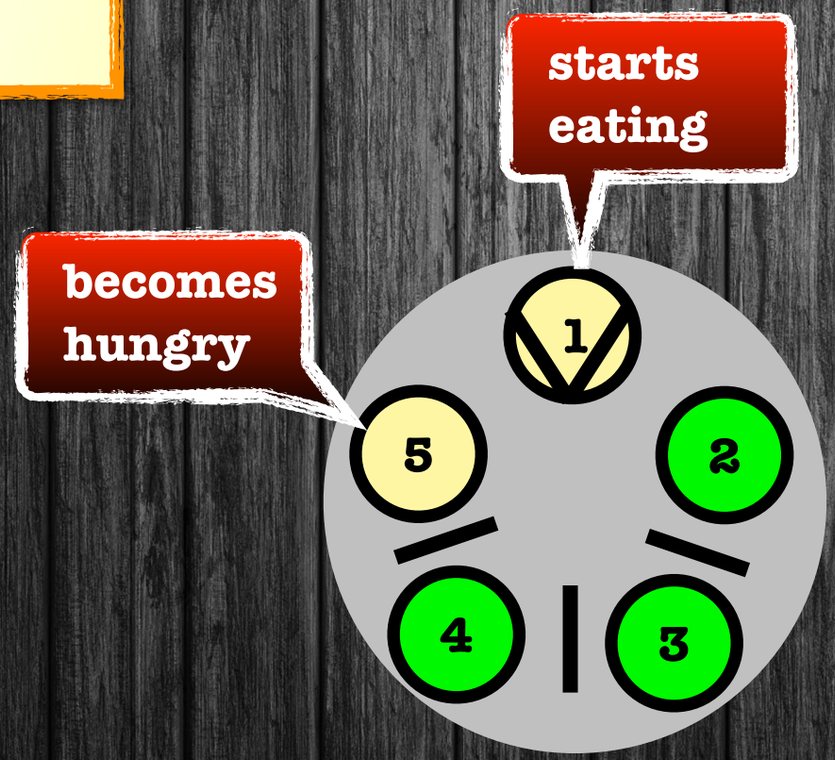
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

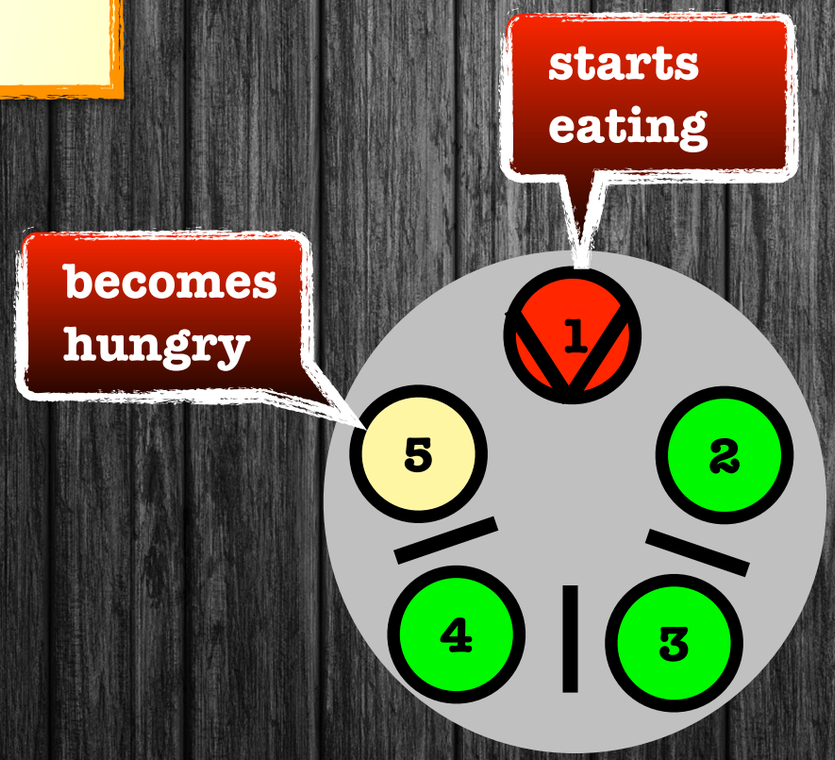
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

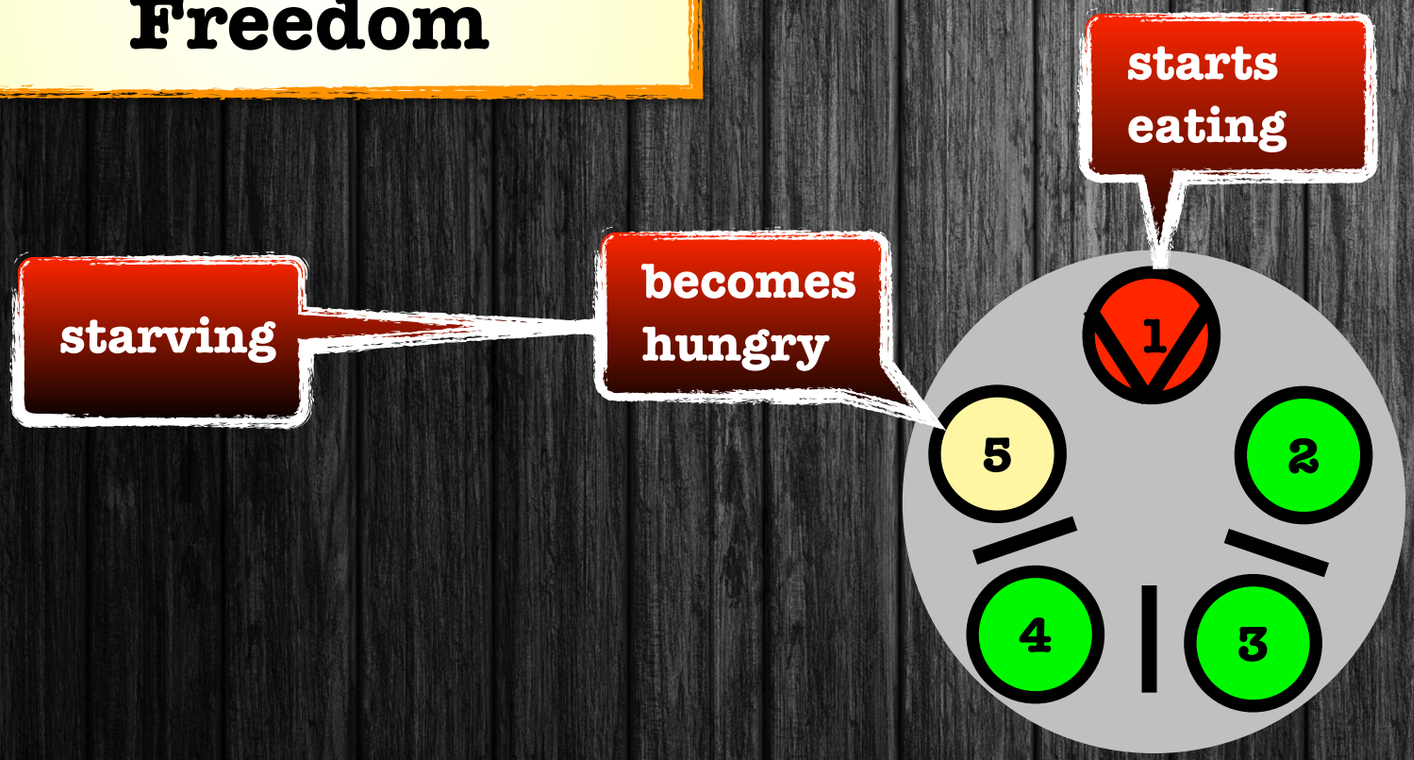
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom

"a philosopher who becomes hungry will eventually eat"



Specification

- safety properties
- liveness properties

concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom

Specification

- safety properties
- liveness properties

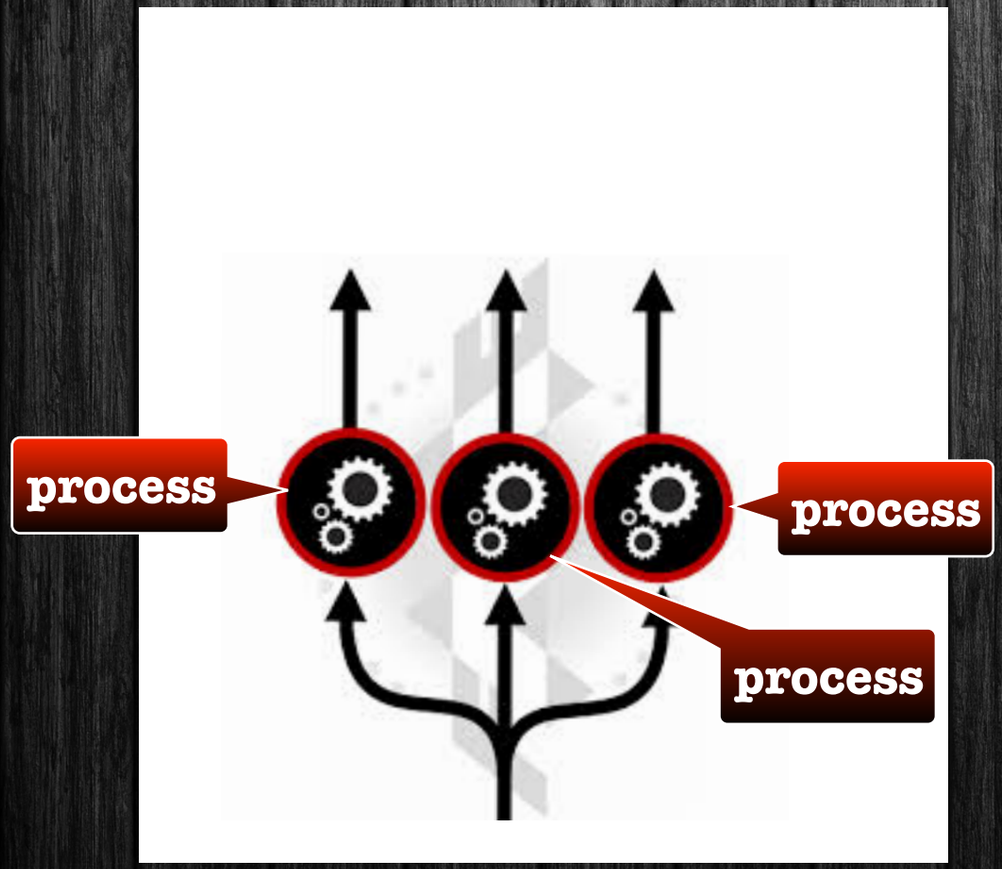
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

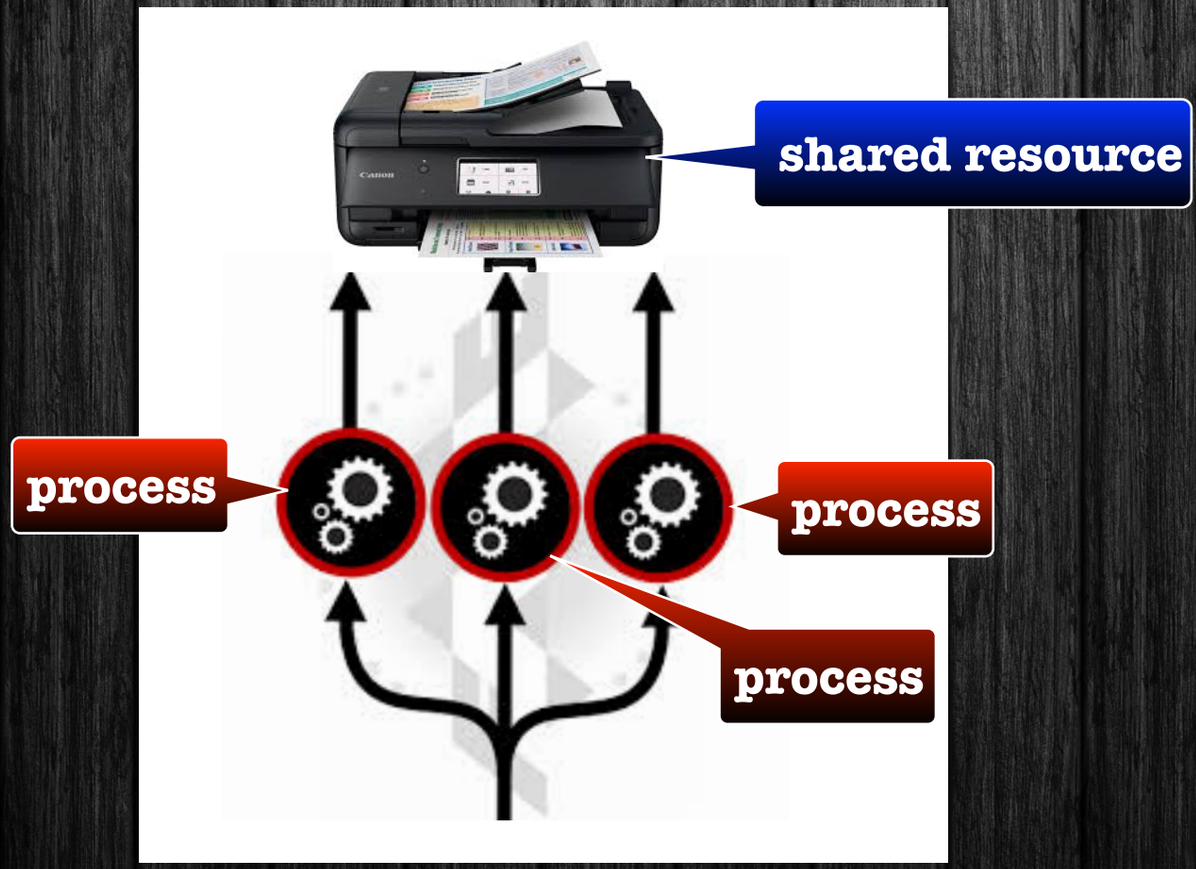
concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom



Specification

- safety properties
- liveness properties

concurrent setting

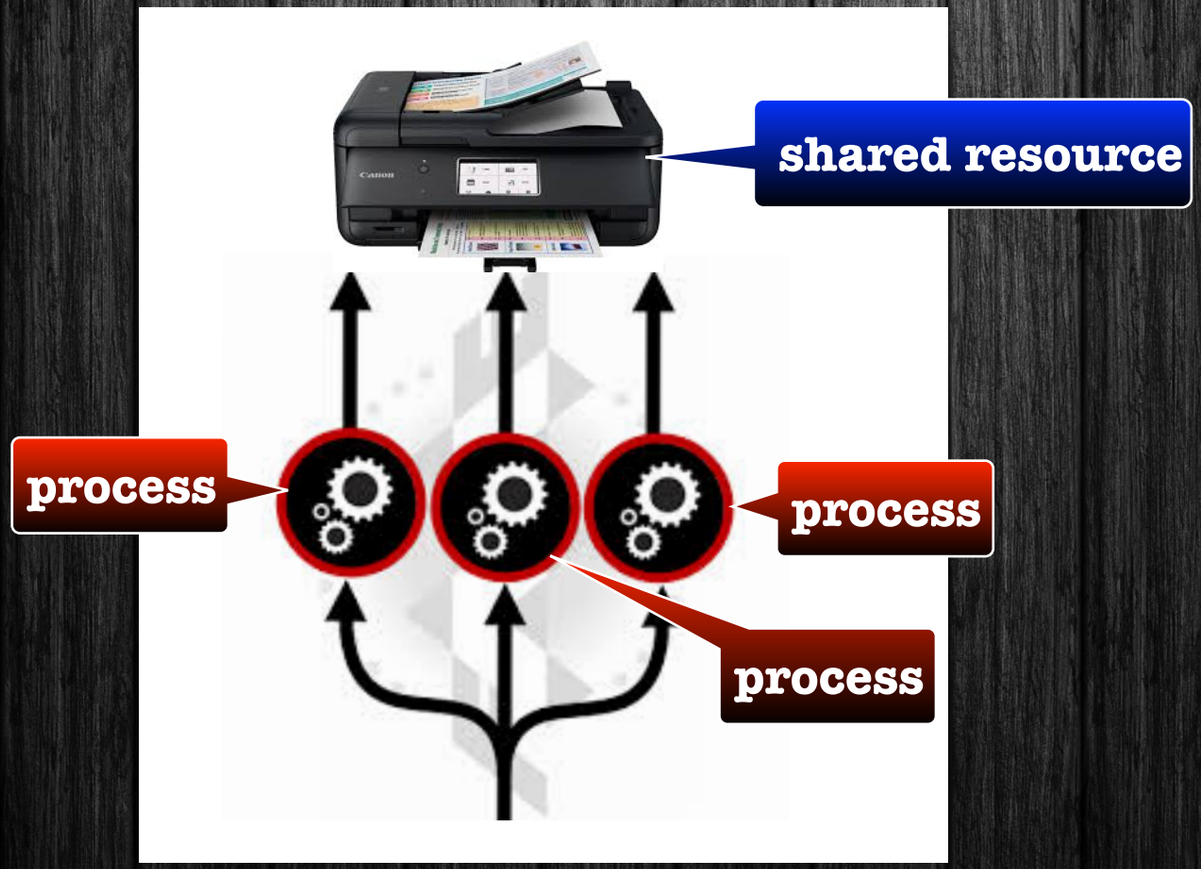
Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Starvation-Freedom

"a process which requests the printer will eventually get access"



Specification

- safety properties
- liveness properties

concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

Specification

- safety properties
- liveness properties

concurrent setting

Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

this course

Specification

- safety properties
- liveness properties

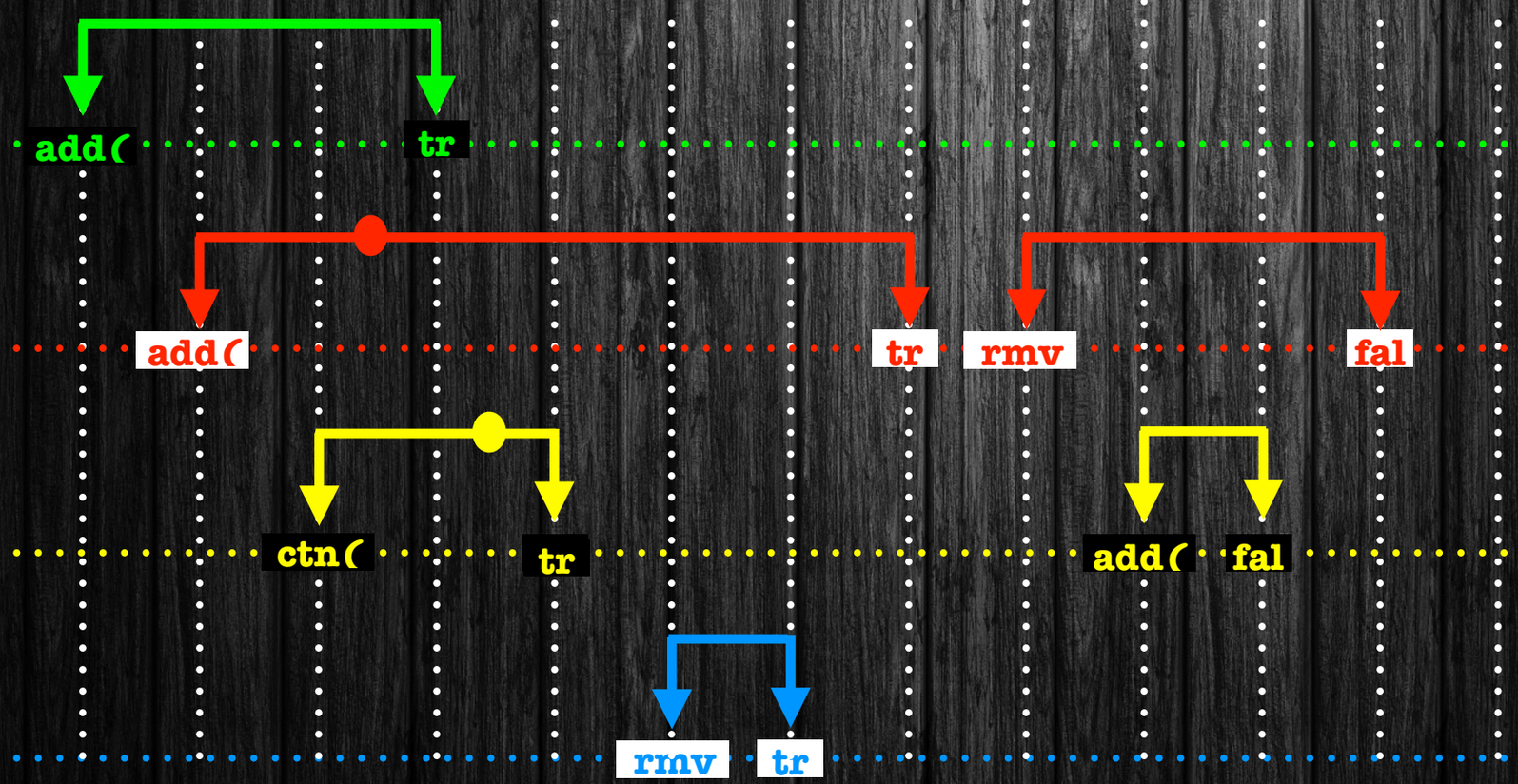
Liveness Properties

Examples

"something good will eventually happen during the execution of the program"

concurrent setting

this course



Linearizability

Specification

- safety properties
- liveness properties

concurrent setting

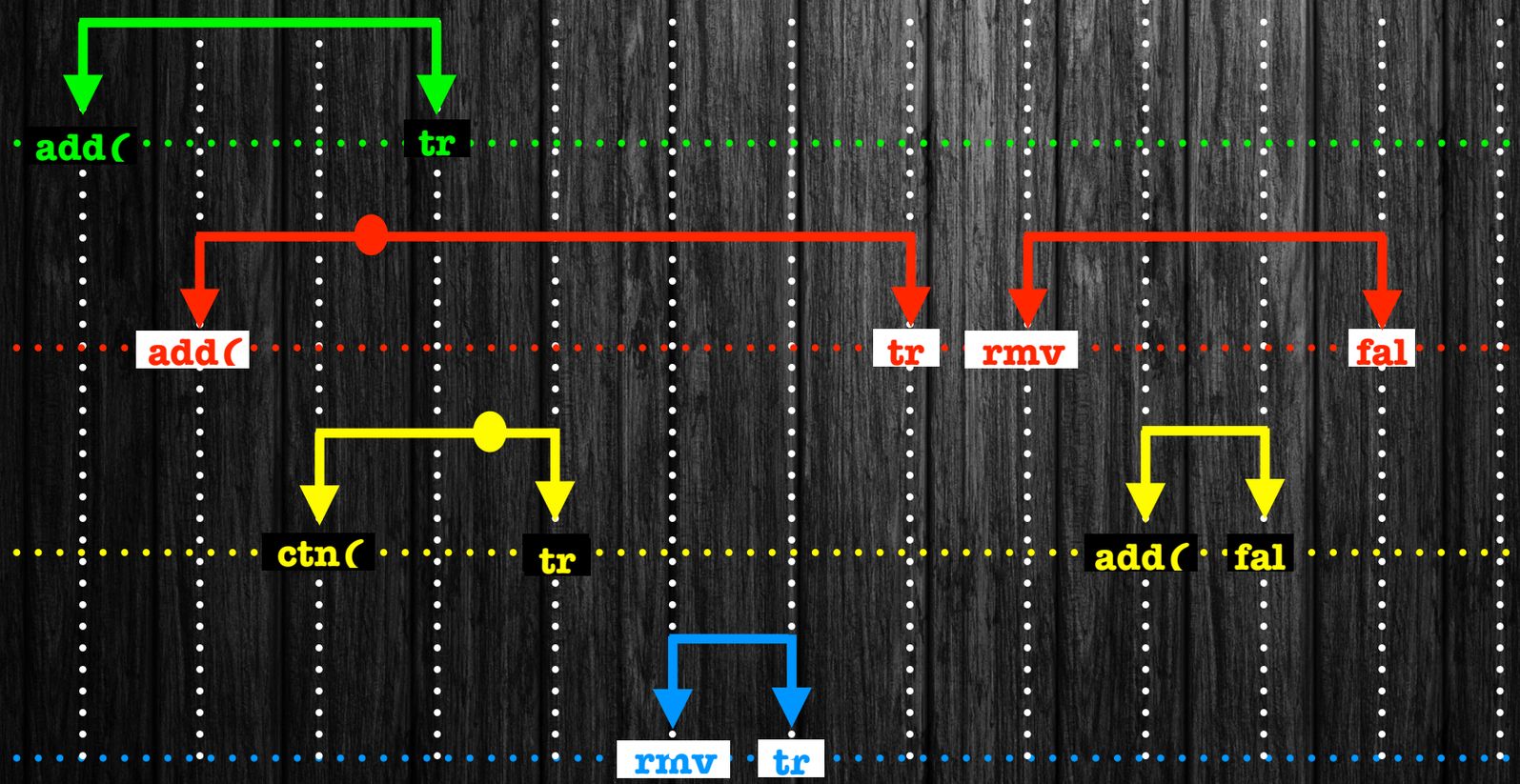
Liveness Properties

"something good will eventually happen during the execution of the program"

Examples

- non-blocking
- wait-freedom
- starvation-freedom

this course



Linearizability

Specification

- safety properties
- liveness properties

Specification

- safety properties
- liveness properties

Safety Properties

"nothing bad will happen during the execution of the program"

Reachability of bad states

Specification

- safety properties
- liveness properties

Safety Properties

"nothing bad will happen during the execution of the program"

Reachability of bad states

Specification

- safety properties
- liveness properties

Safety Properties

"nothing bad will happen during the execution of the program"

Reachability of bad states

Liveness Properties

"something good will eventually happen during the execution of the program"

Repeated Reachability of good states

Specification

- safety properties
- liveness properties

Safety Properties

"nothing bad will happen during the execution of the program"

Reachability of bad states

Liveness Properties

"something good will eventually happen during the execution of the program"

Repeated Reachability of good states