MÉTHODES FORMELLES

MAIS QUOI DONC EST-CE ?

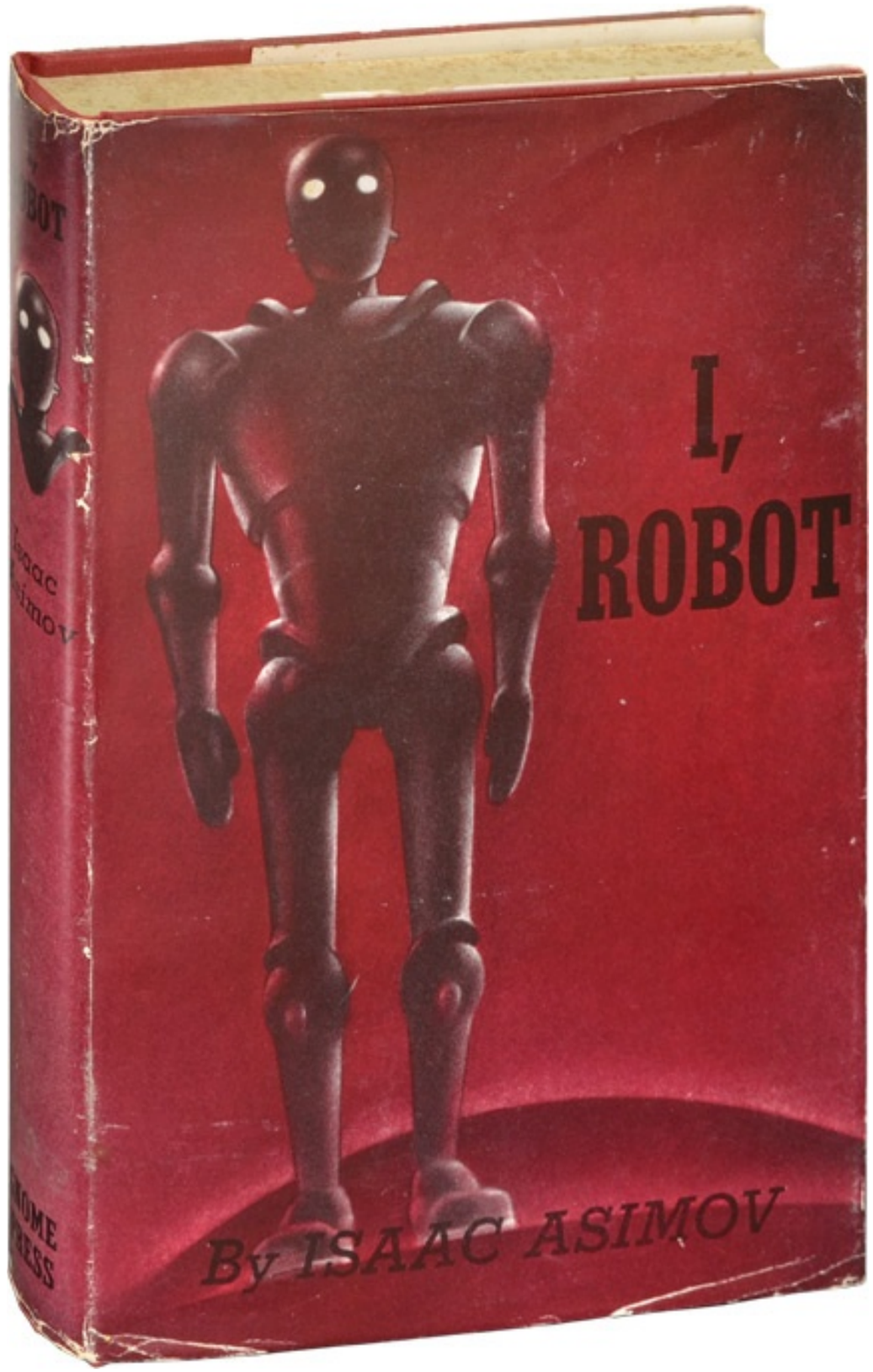
PAS SAGE EN SEINE

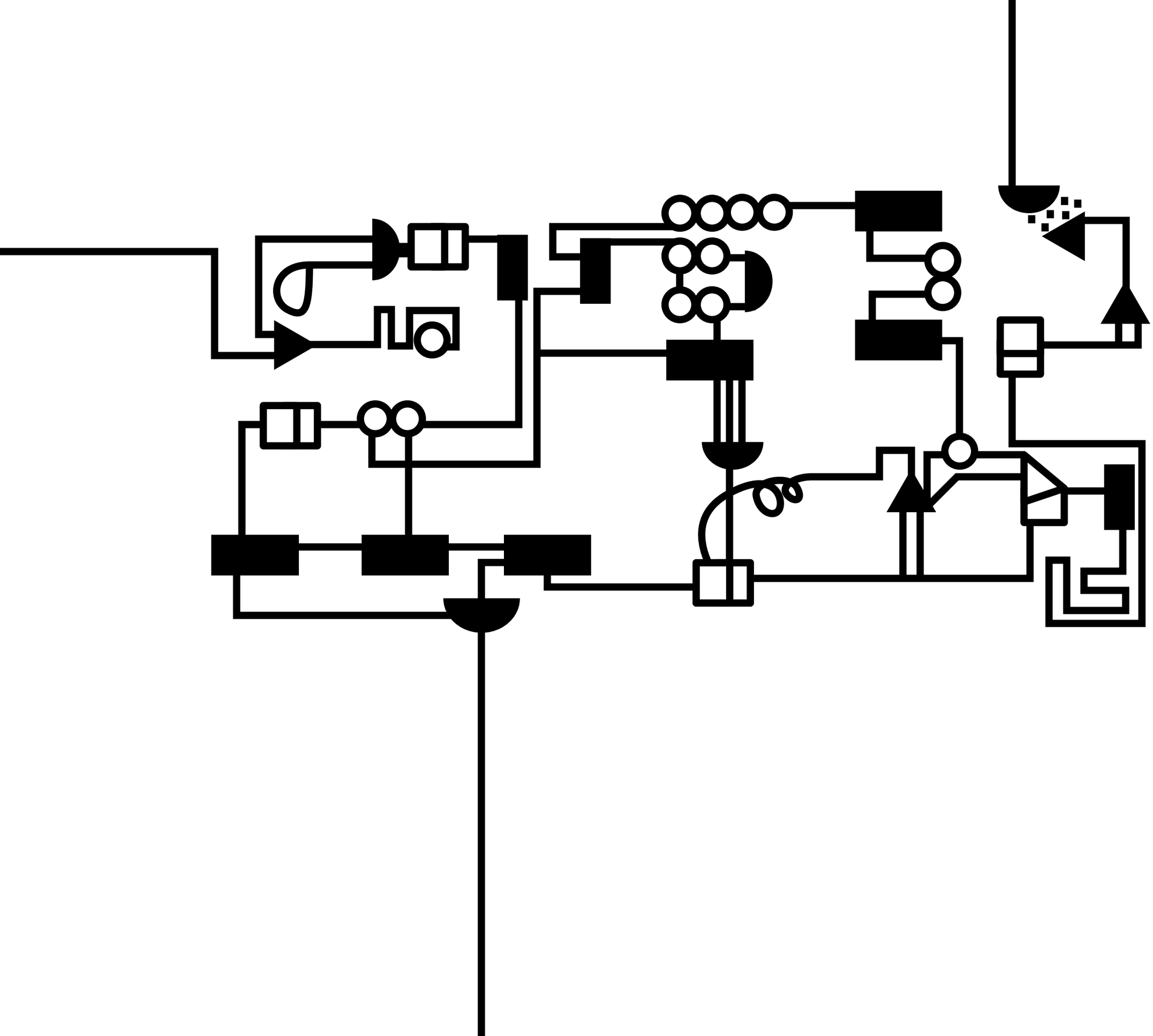
27 · 28 · 29 · 30 Juin

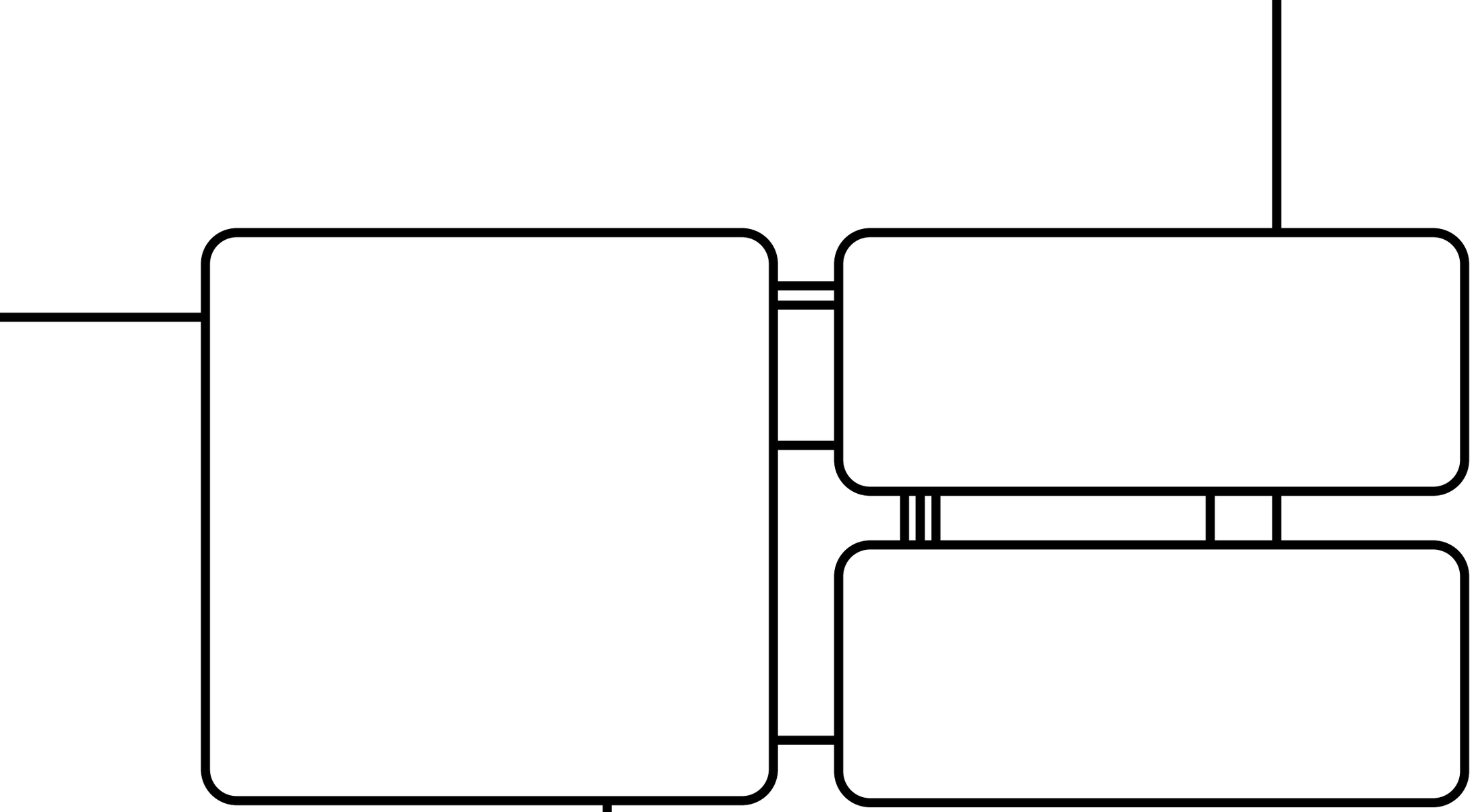
SOCIÉTÉ ■ INTERNET ■ LIBERTÉ

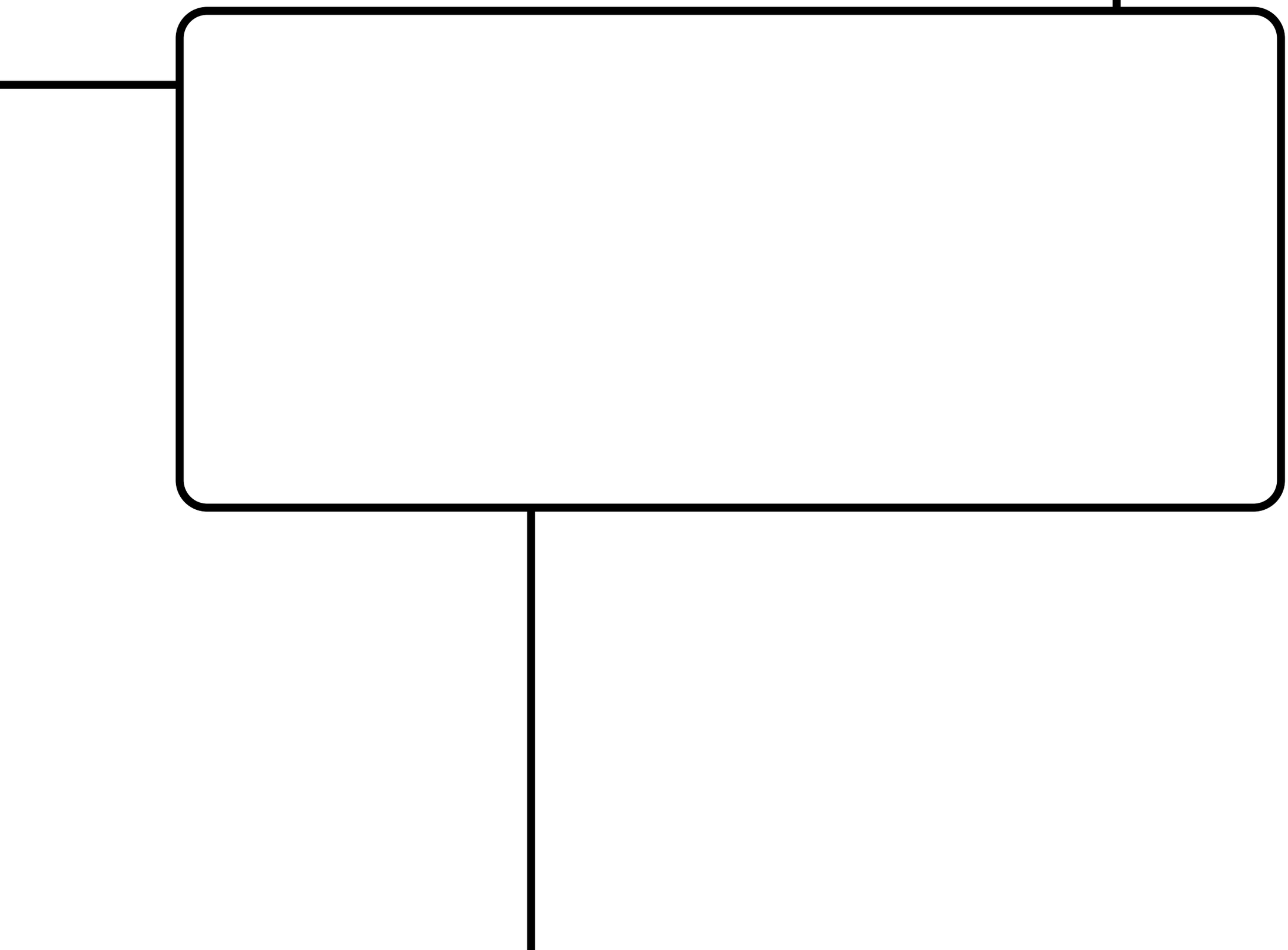
FLORENT & LESLY

JEUDI 27 JUIN 2019


I,
ROBOT
By ISAAC ASIMOV



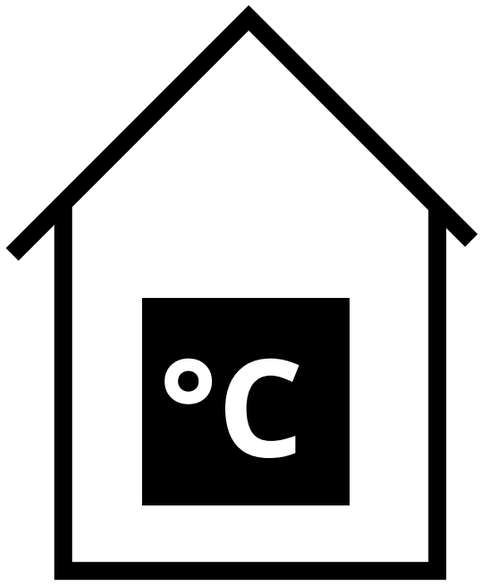


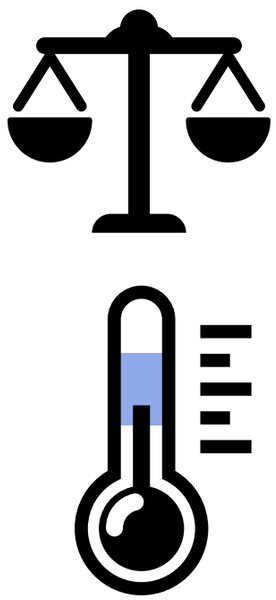


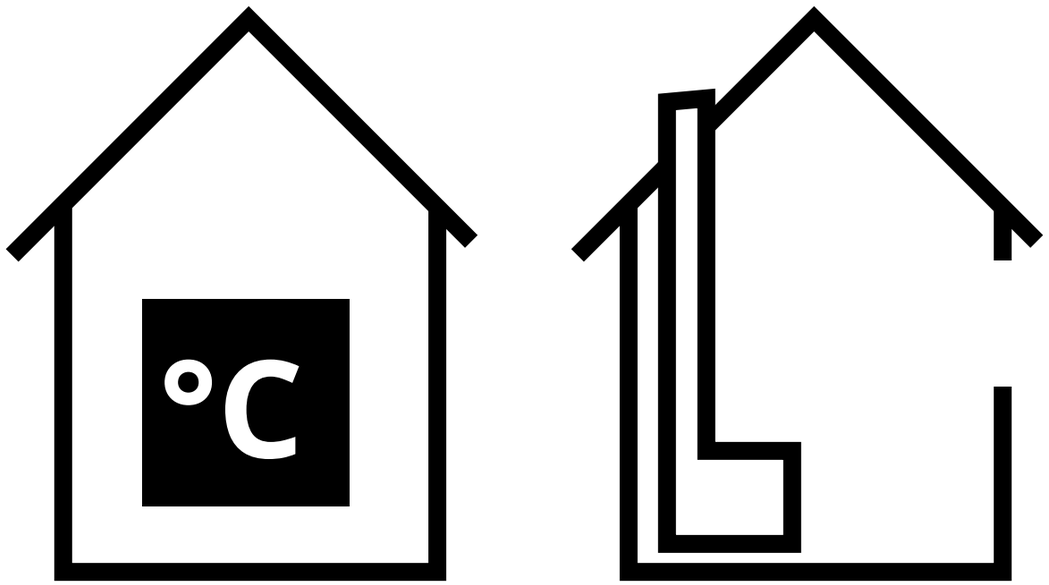
HOME SWEET HOME

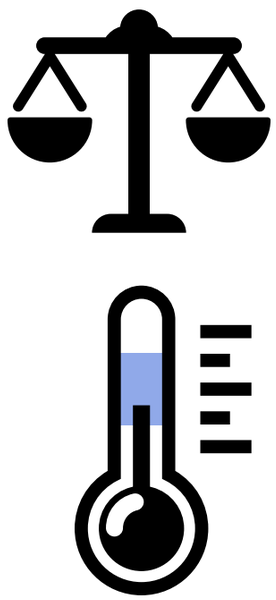
EN ROUTE VERS LE DÉTAIL

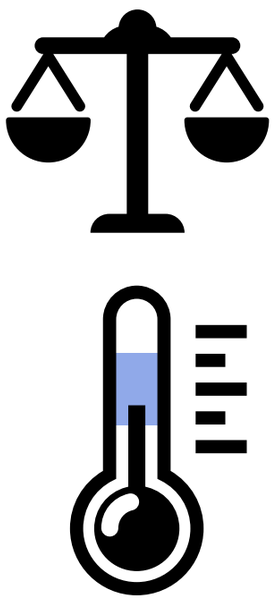
CONCEVOIR DES PROGRAMMES FIABLES

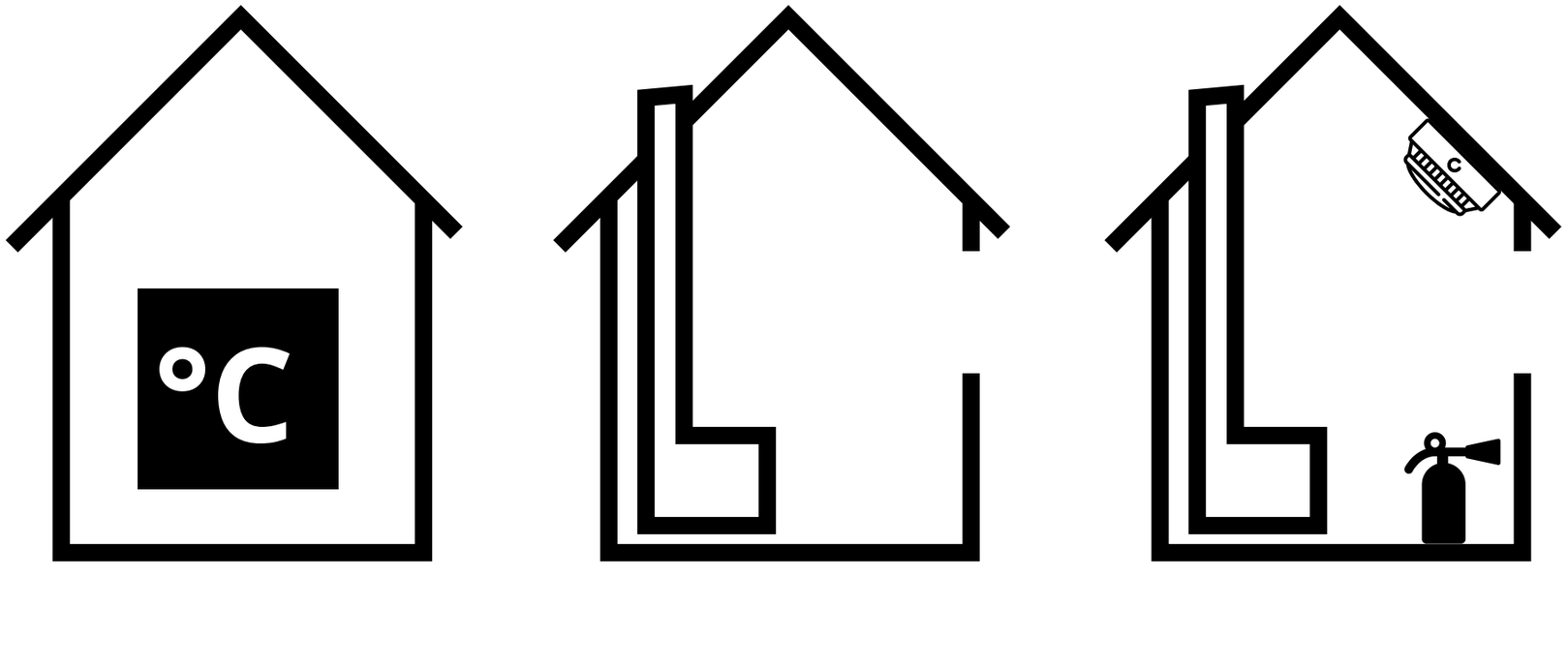

°C

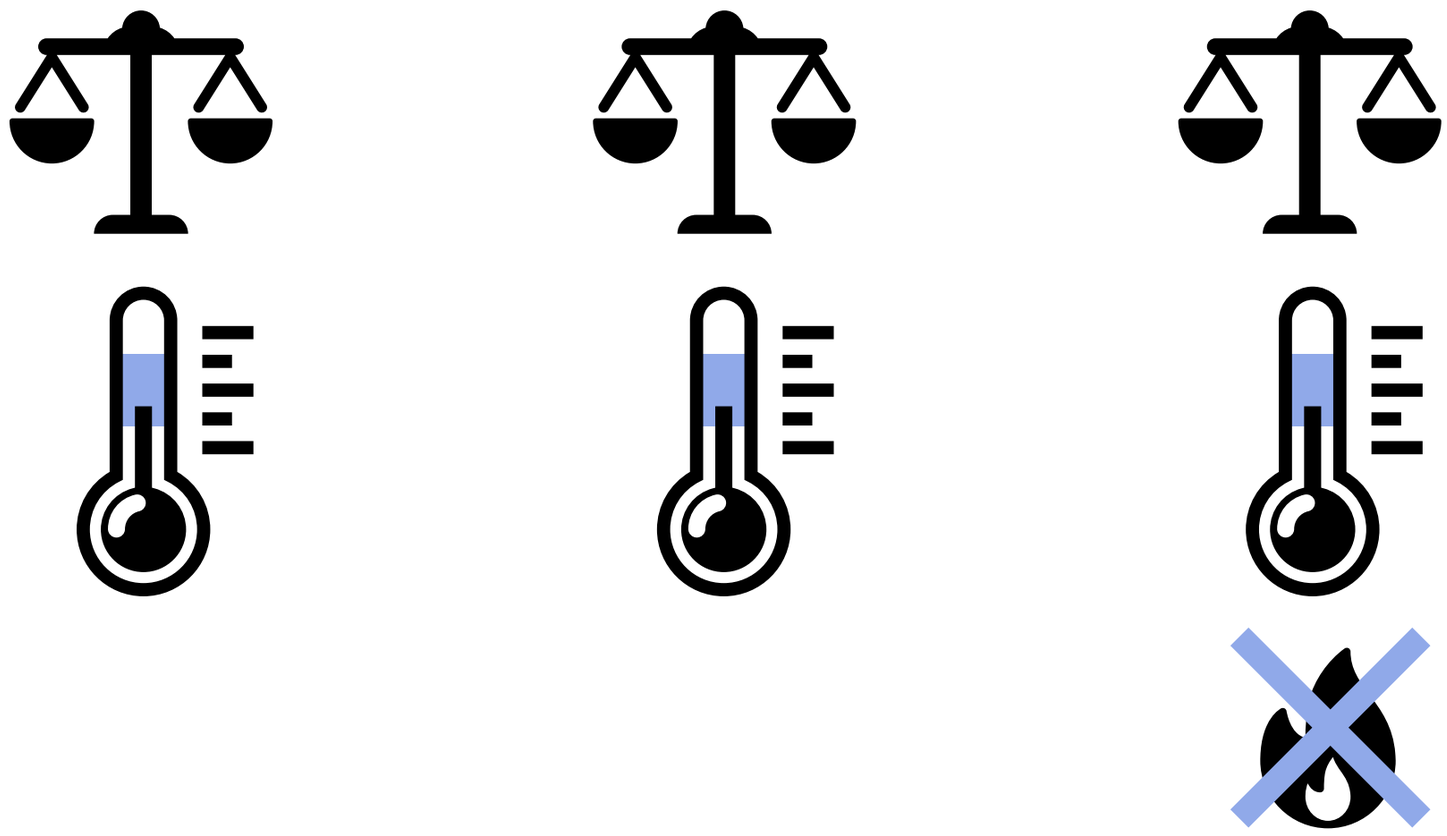


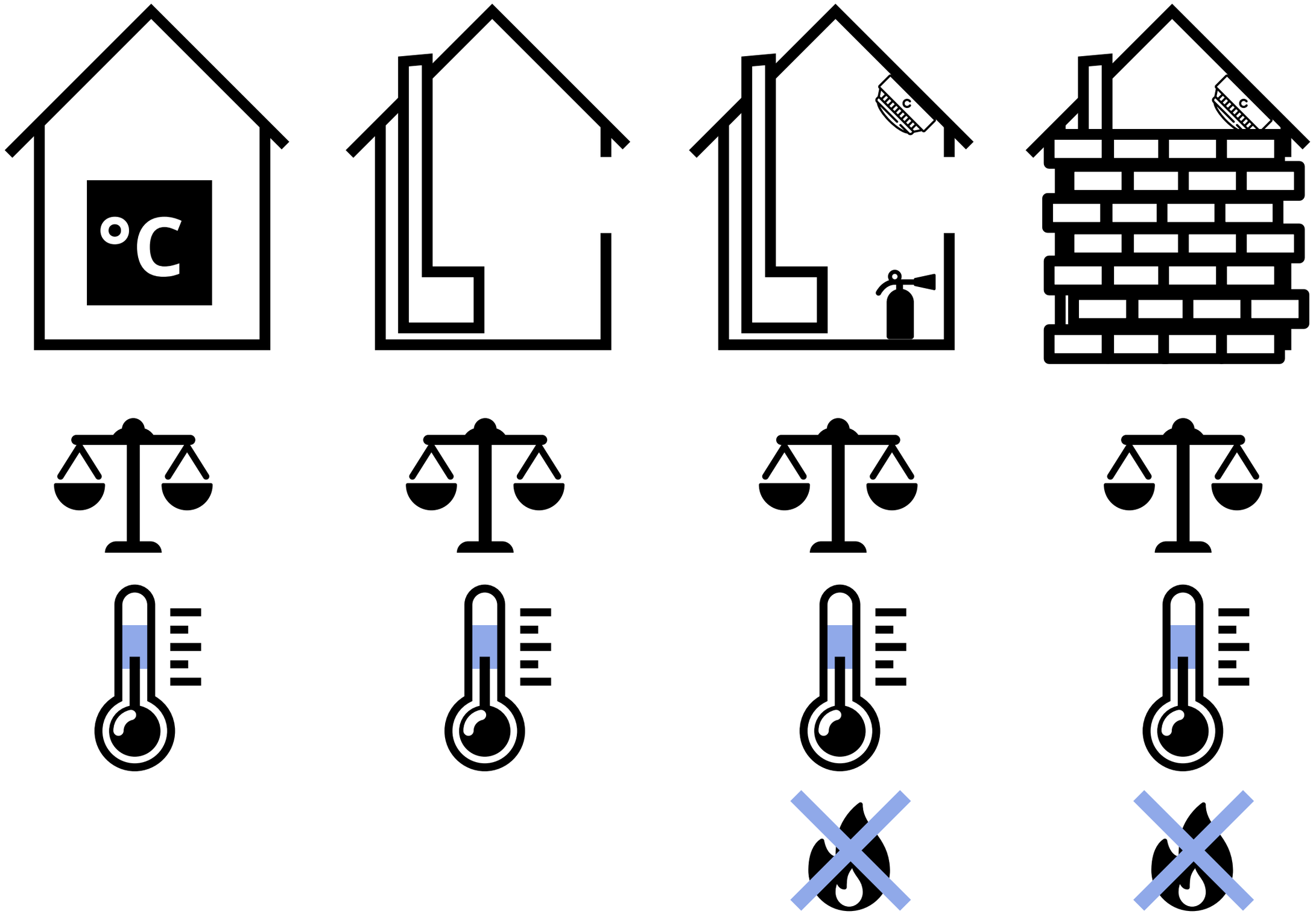

°C

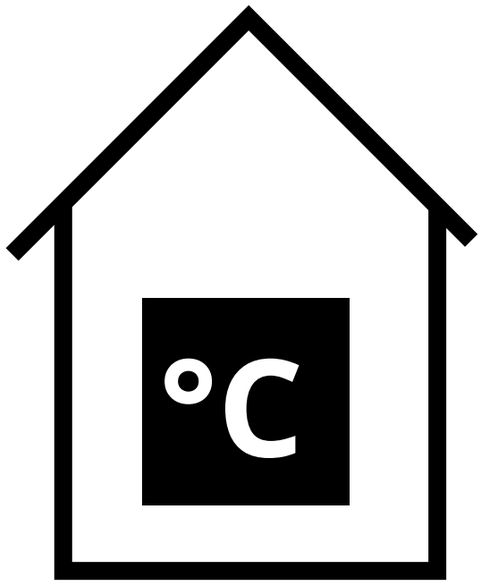


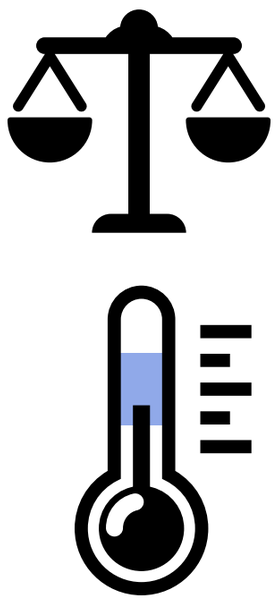


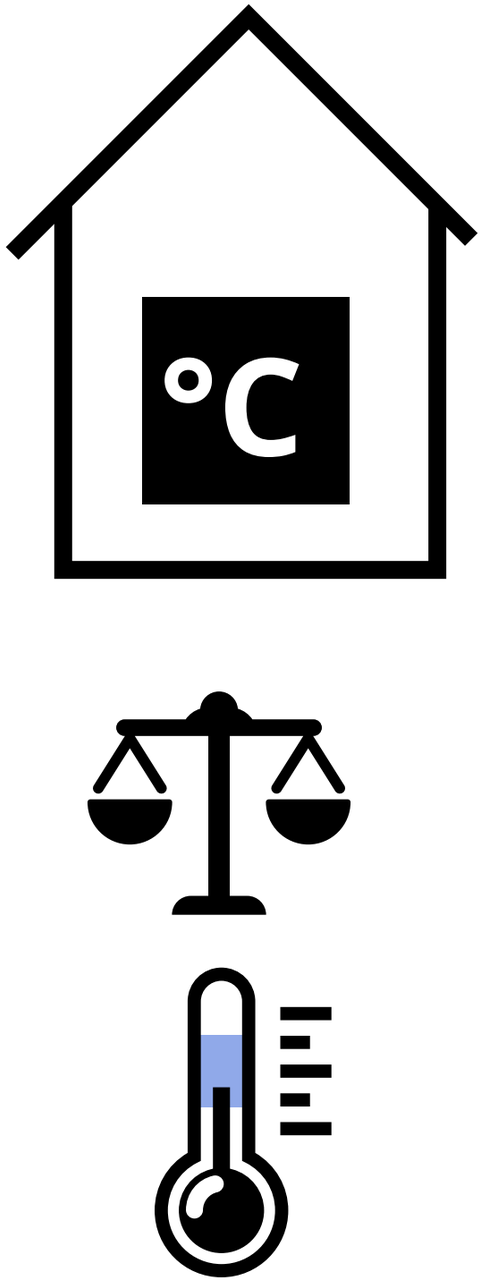

°C

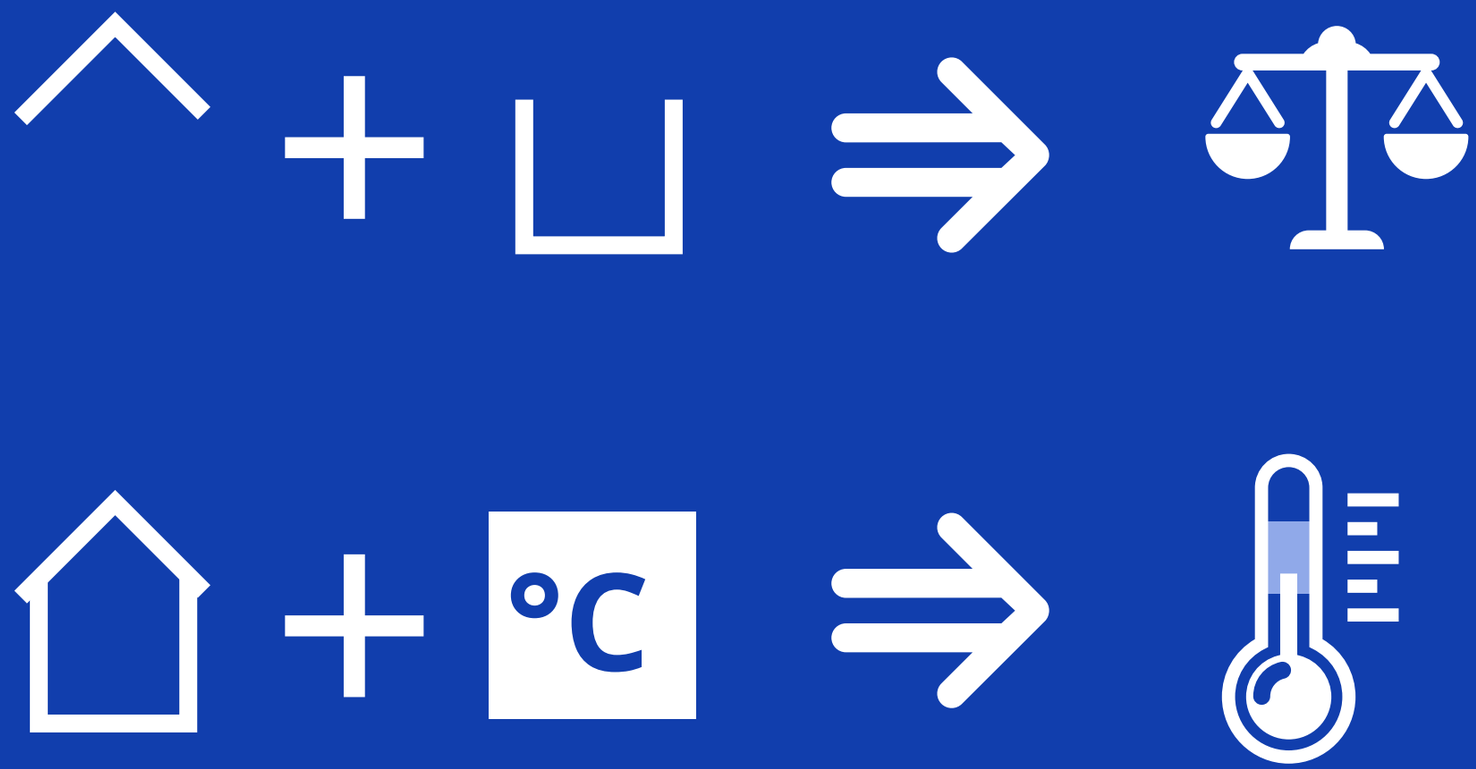


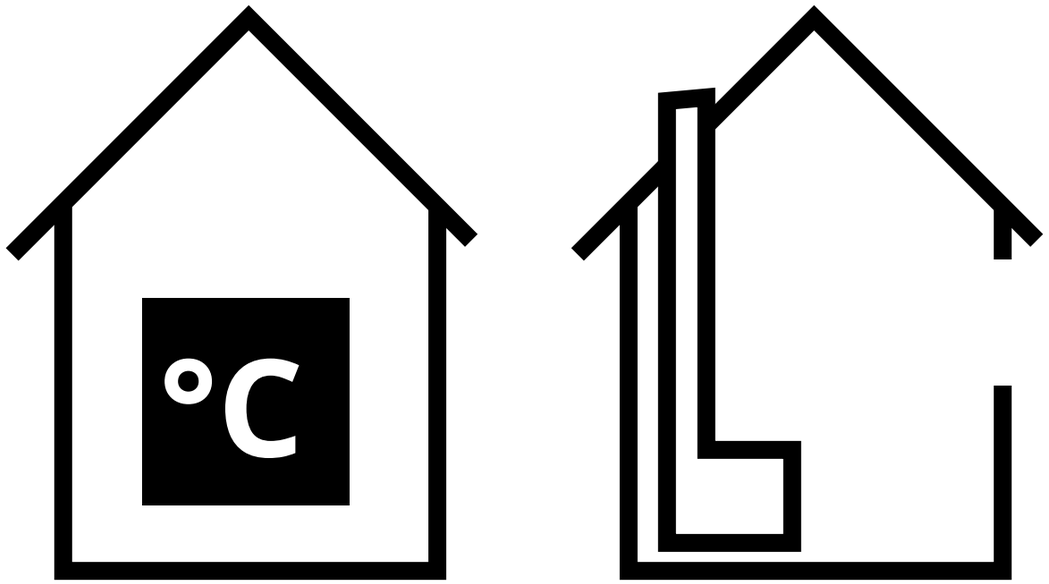

°C

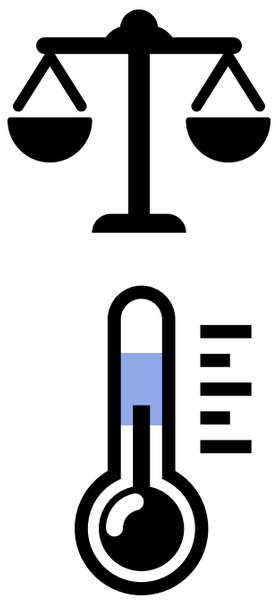

°C

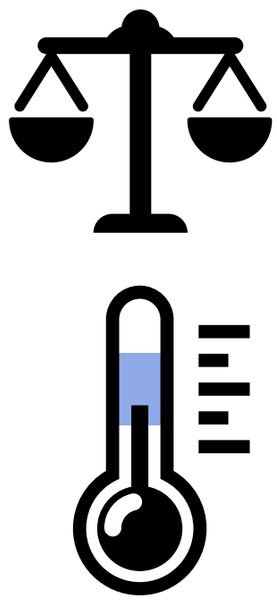


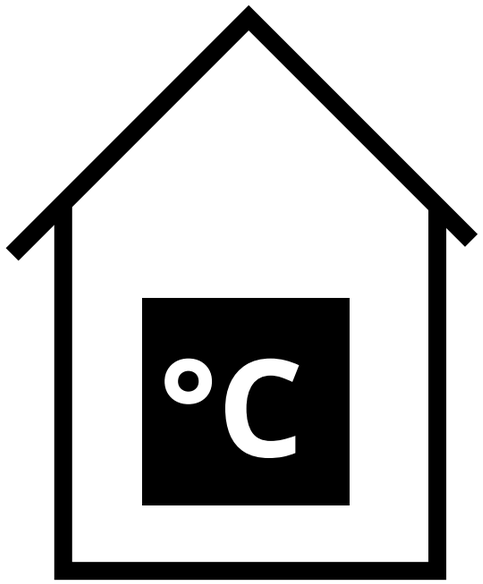

°C

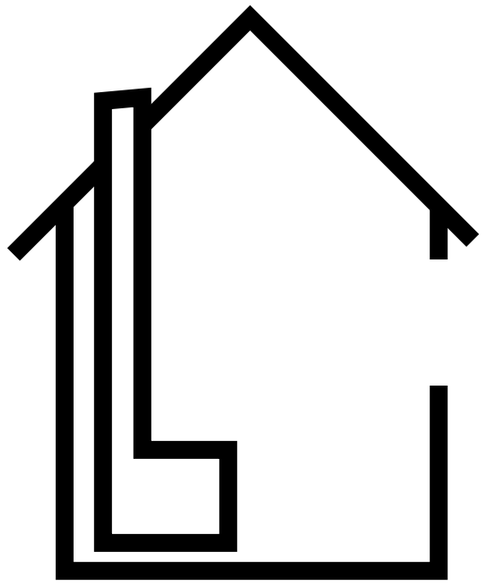

°C

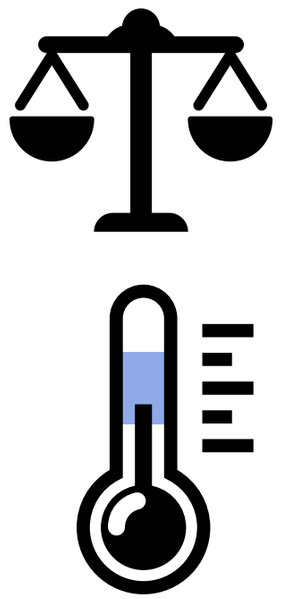

°C

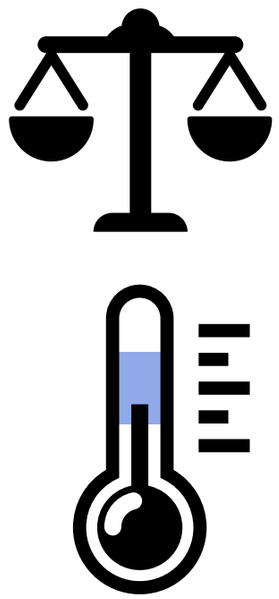


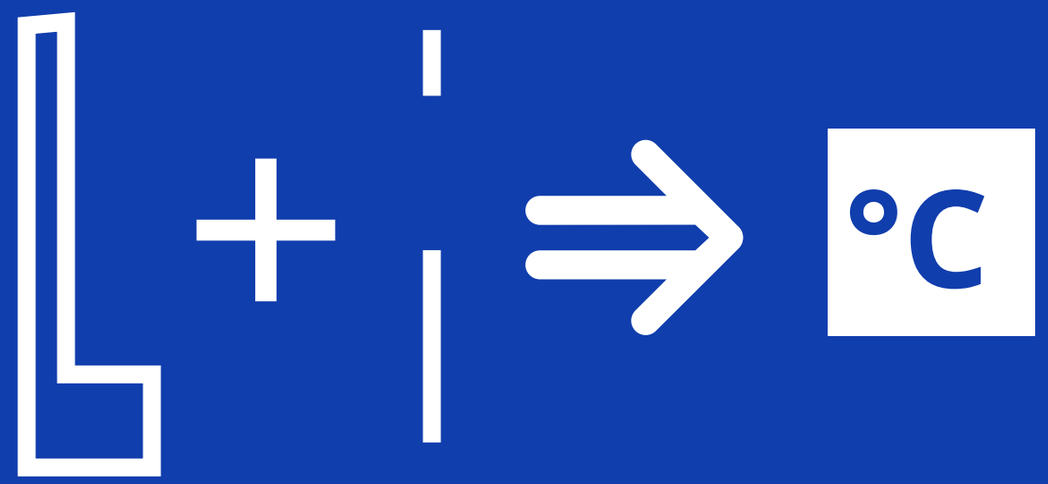


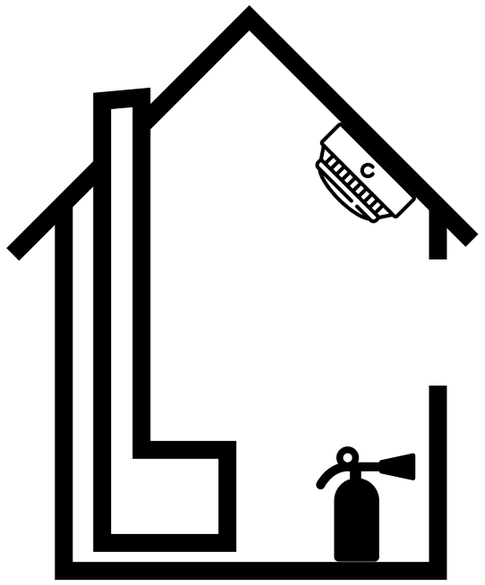

°C

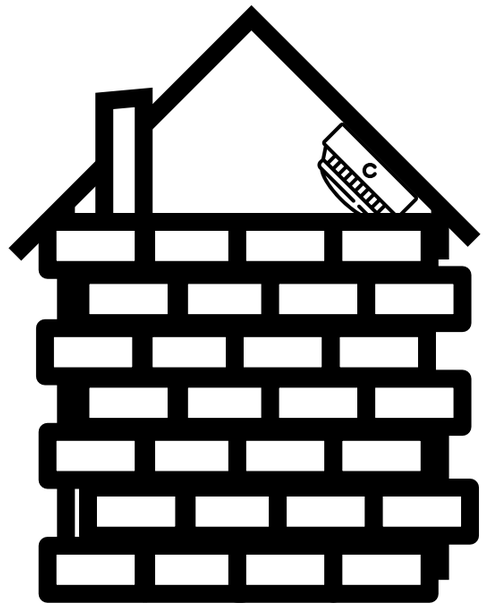


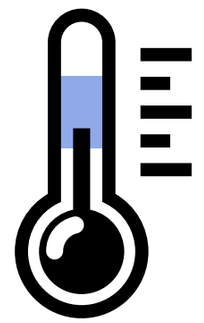


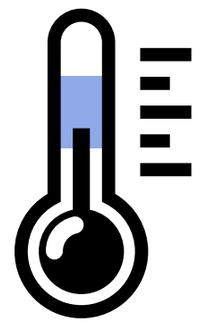



°C





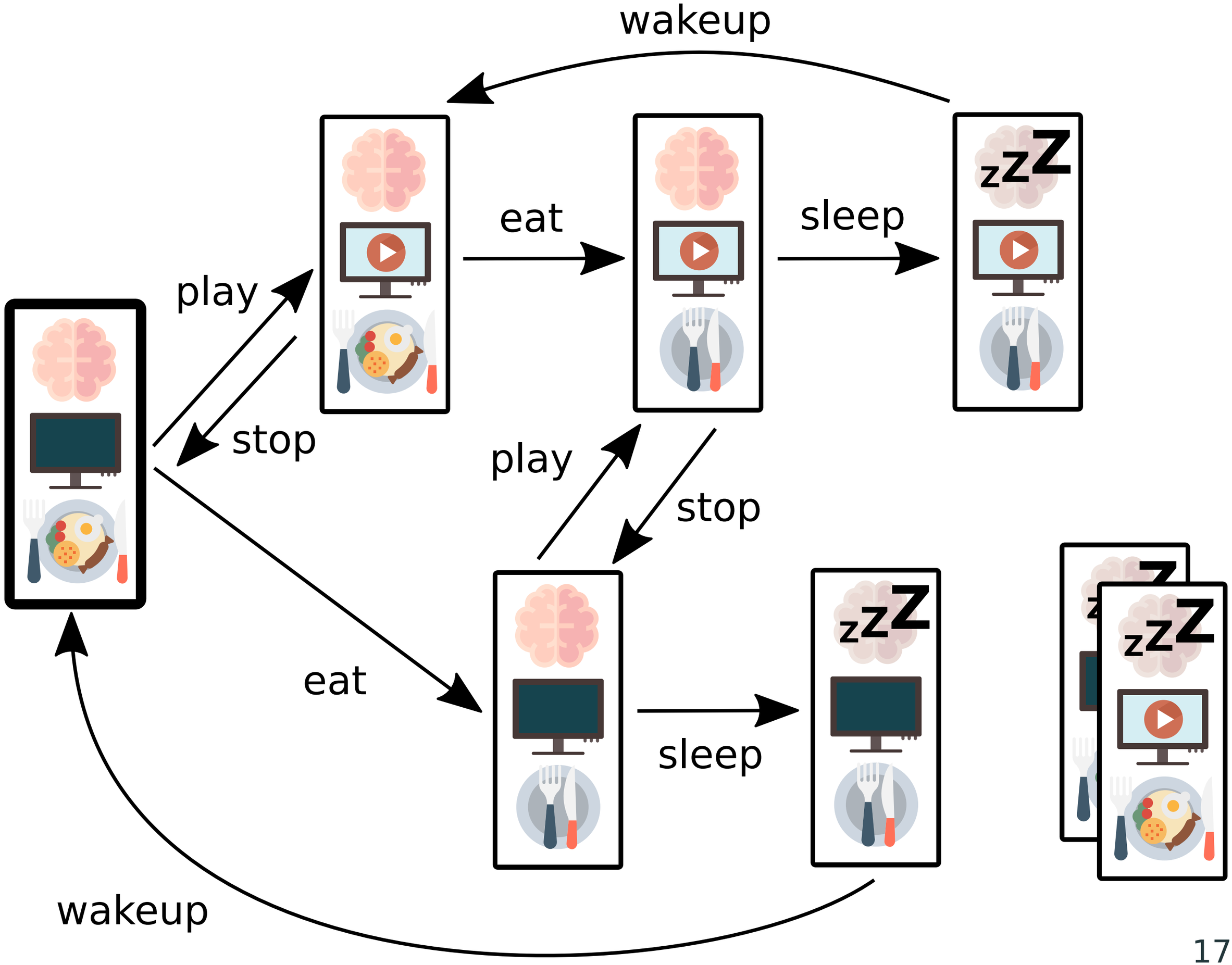




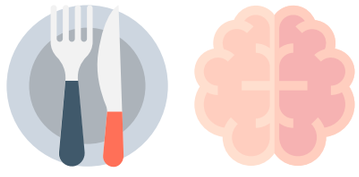
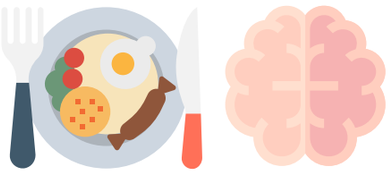
SOIRÉE SÉRIE

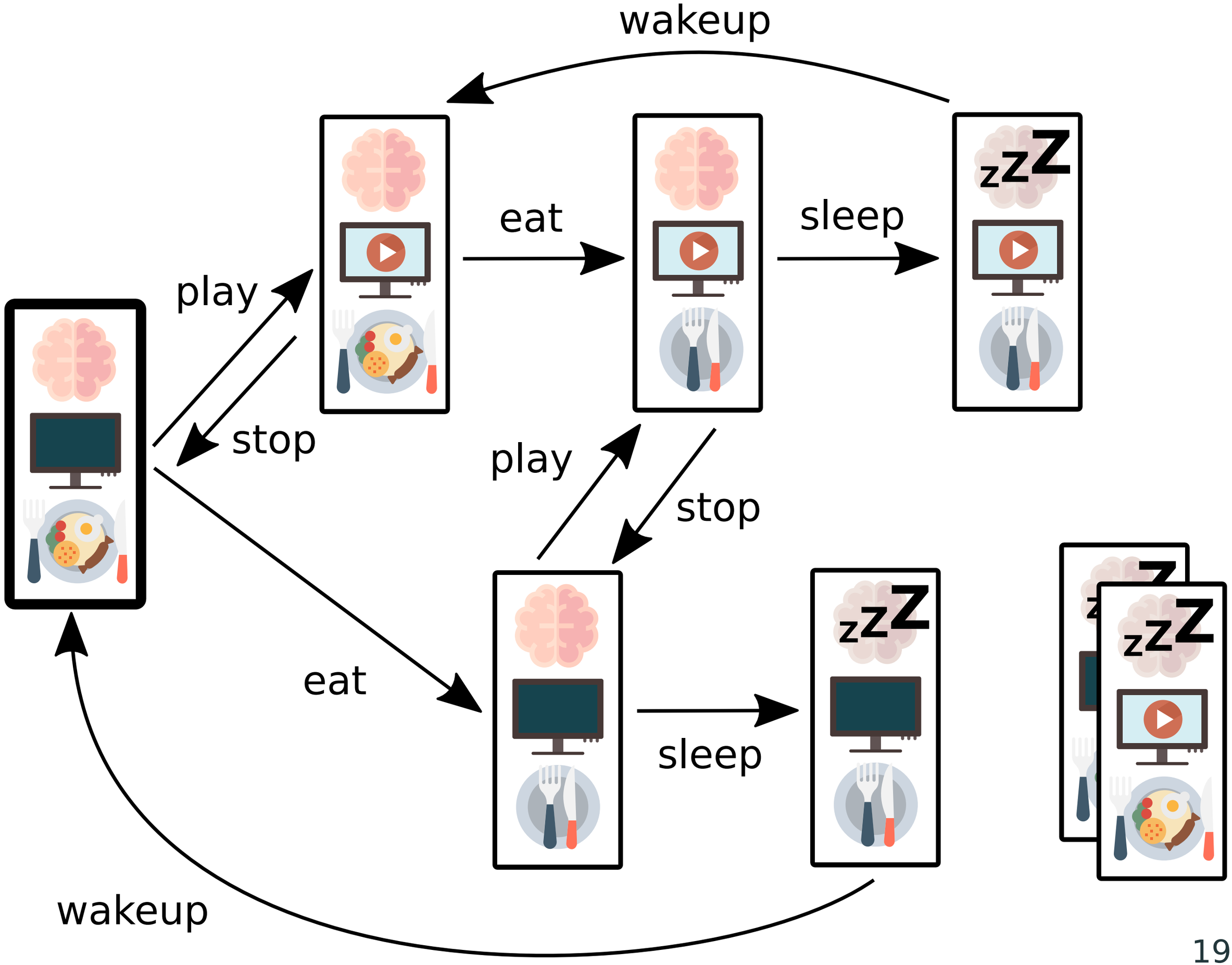
DESCRIPTION ET PREUVE

MAIS ON MANIPULE QUOI AU JUSTE ?


wakeup
play
eat
sleep
stop
play
stop
eat
sleep
wakeup

ÉTAT	éveil	faim	film
Au départ			

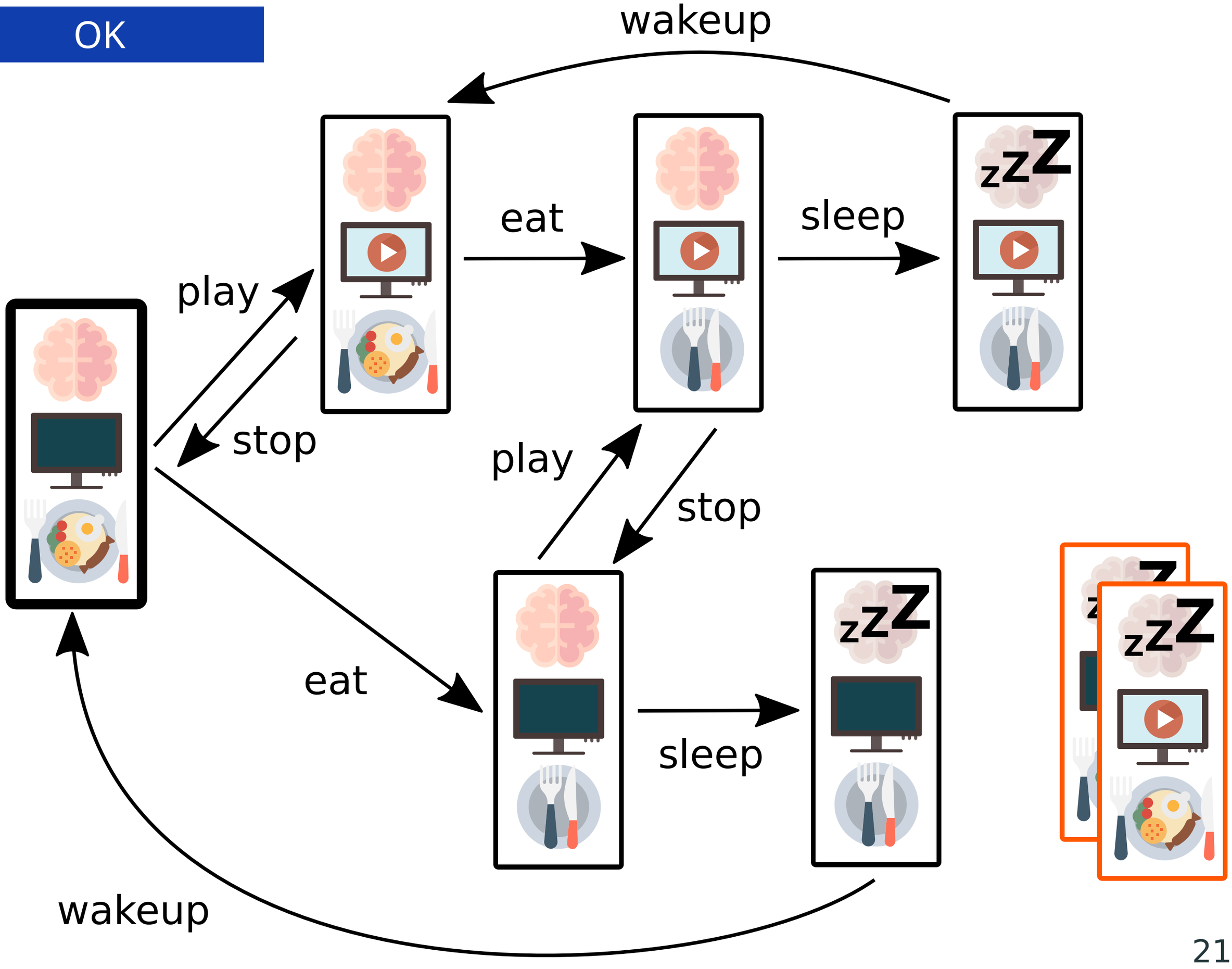
ACTIONS	prérequis	modification
play		
stop		
eat		
sleep		
wake up		


wakeup
play
eat
sleep
stop
play
stop
eat
sleep
wakeup

1

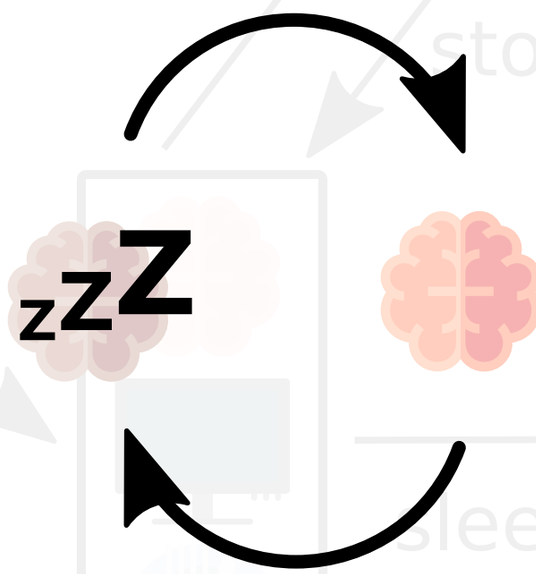
ON NE DOIT PAS DORMIR L'ESTOMAC VIDE

1 OK



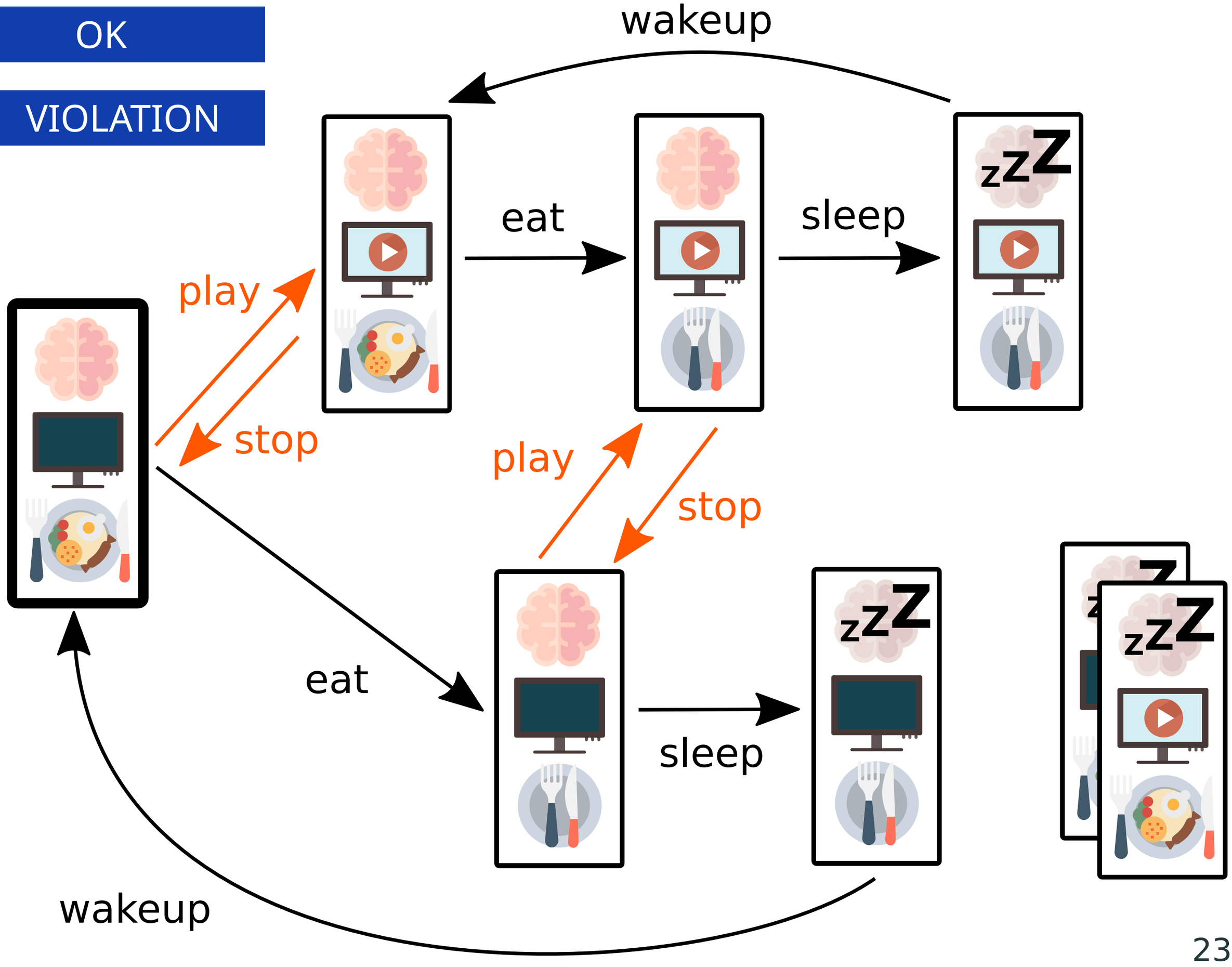
2

IL FAUT ALTERNER L'ÉVEIL ET LE SOMMEIL



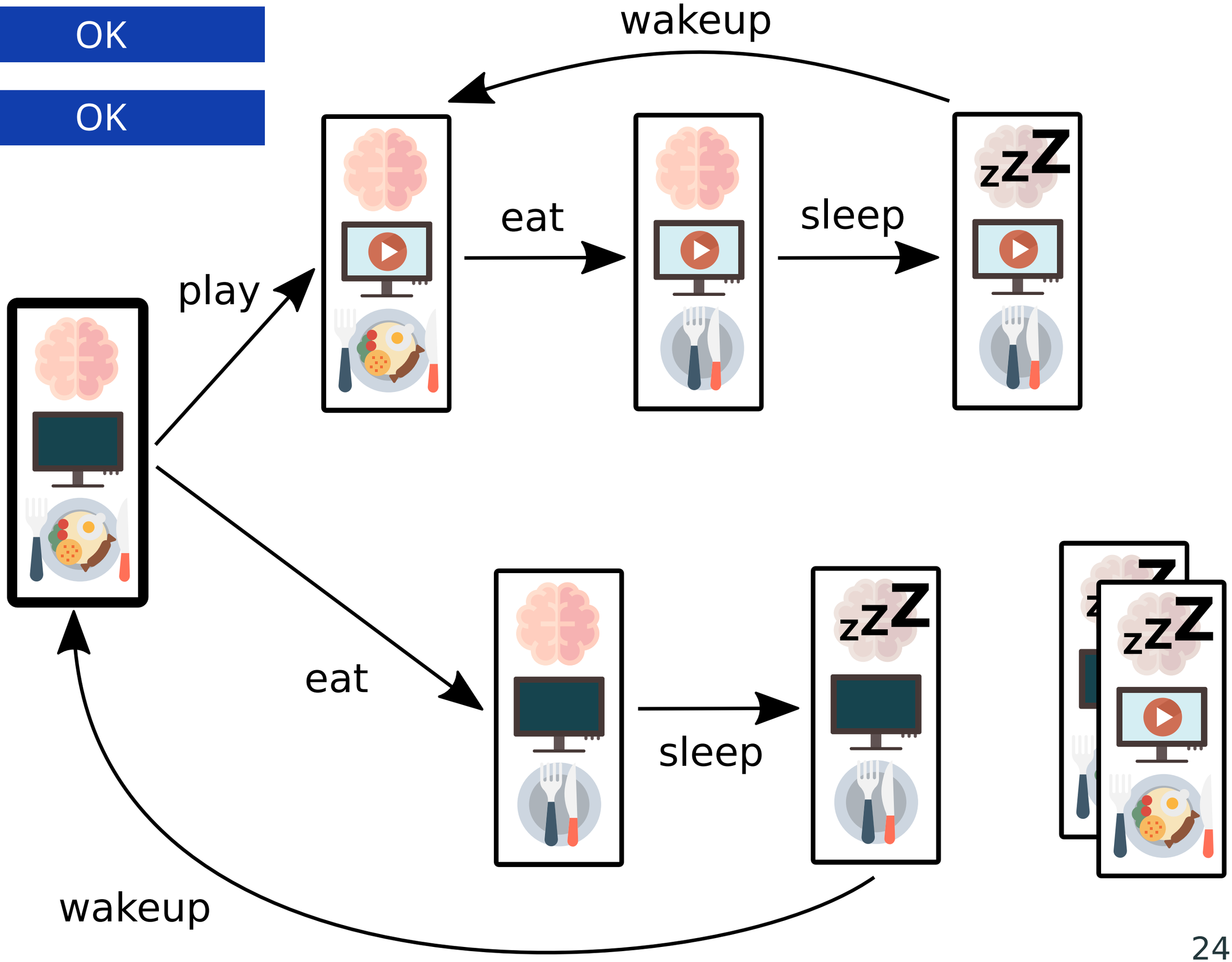
1 OK

2 VIOLATION

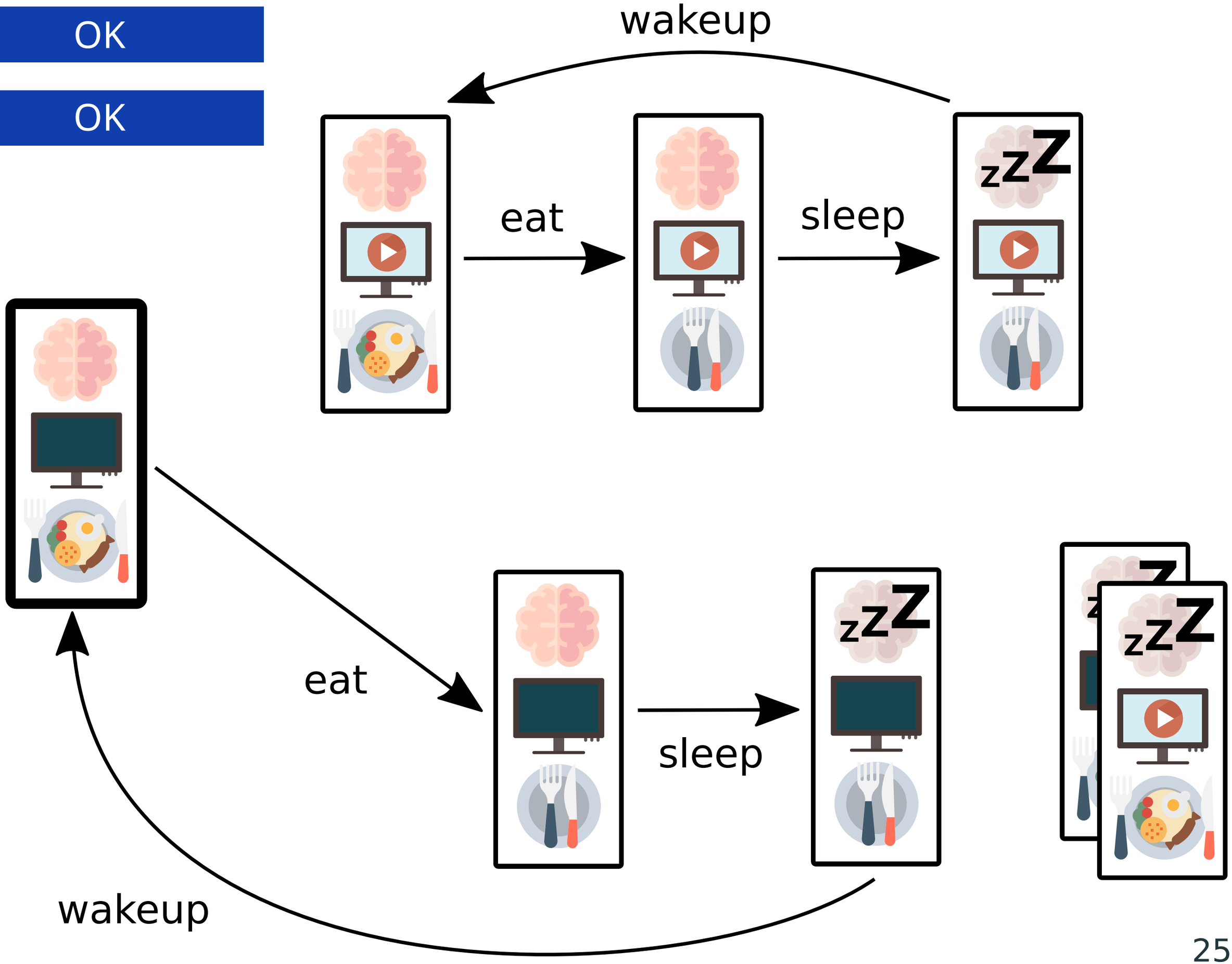


1 OK

2 OK

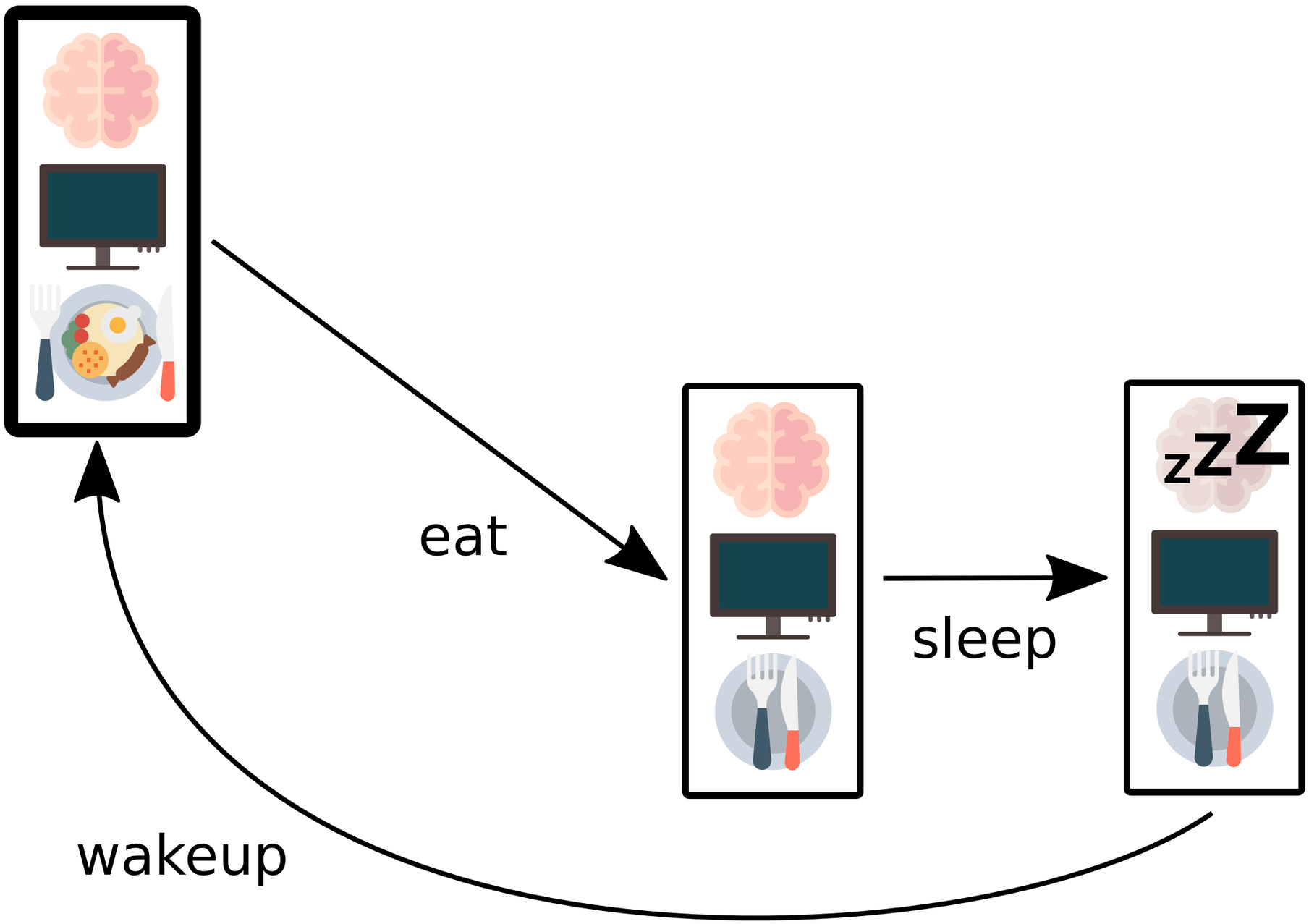


1. OK
2. OK



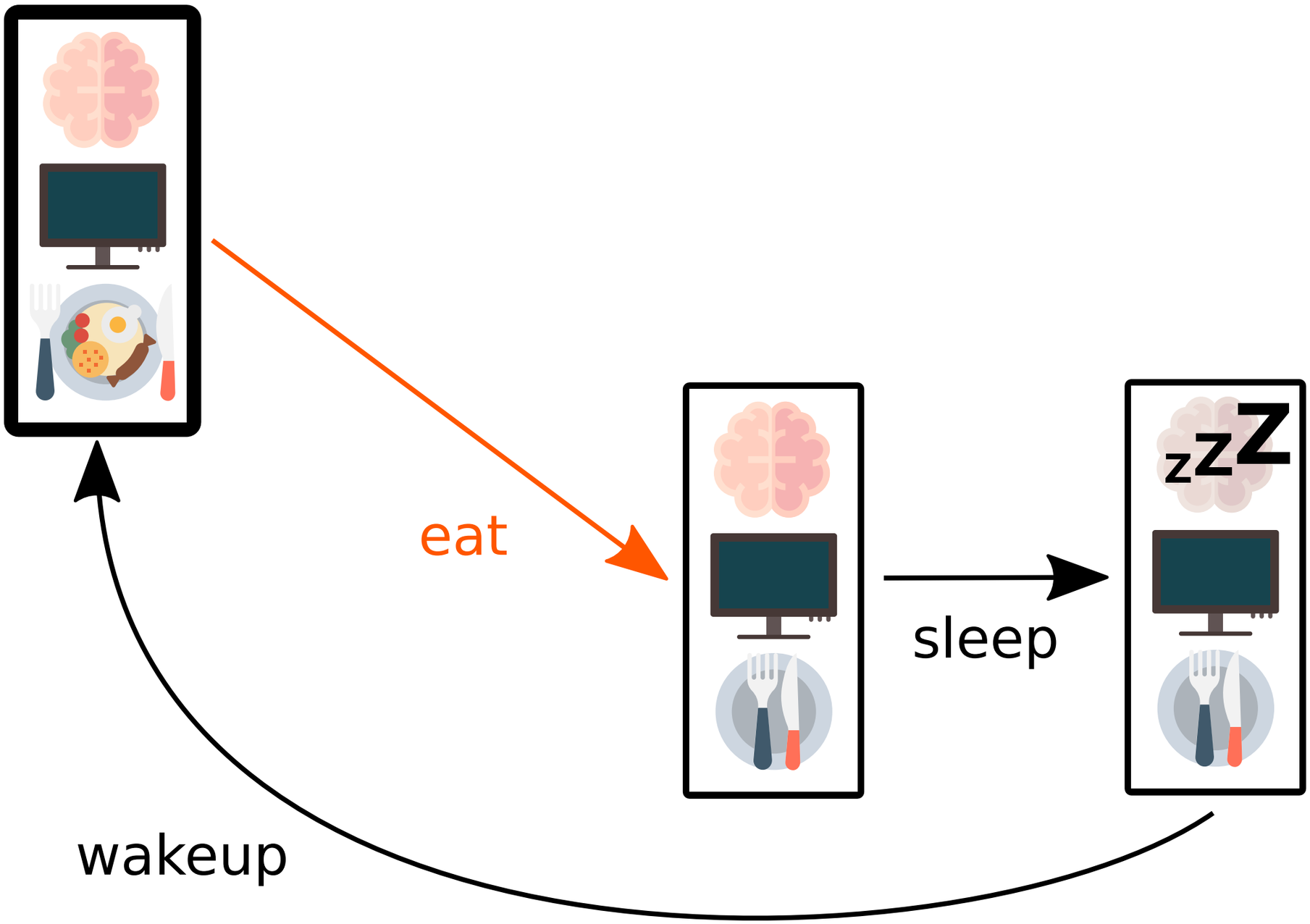
1 OK

2 OK



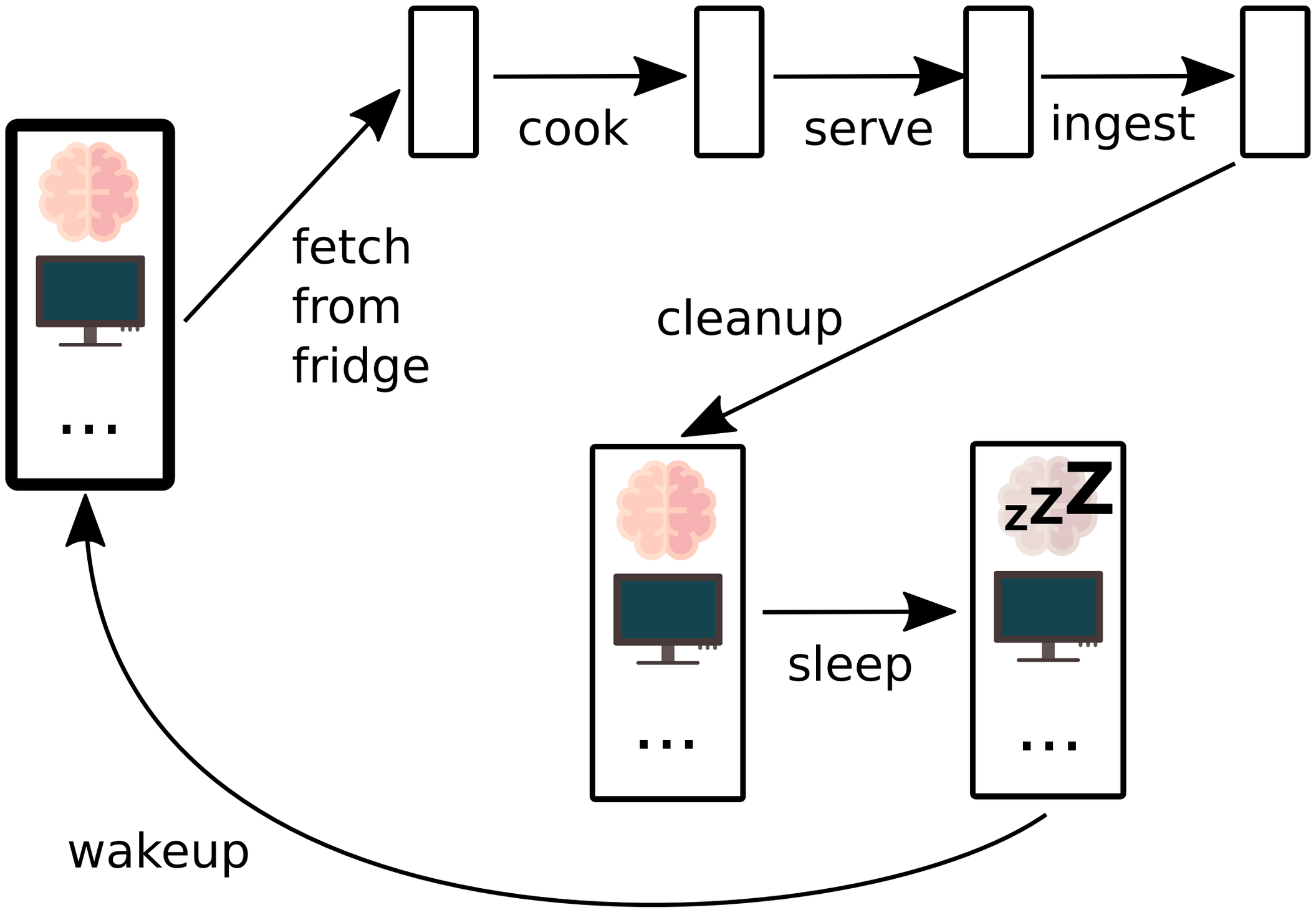
1 OK

2 OK



1 OK

2 OK


cook
serve
ingest
fetch
from
fridge
cleanup
...
...
zzZ
...
sleep
wakeup

File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

Arith.v | Arith_base.v | PeanoNat.v

```
 revert m; induction n; destruct m; simpl; rewrite ?IHn; split; auto; easy.
Qed.

Lemma compare_lt_iff n m : (n ?= m) = Lt <-> n < m.
Proof.
 revert m; induction n; destruct m; simpl; rewrite ?IHn; split; try easy.
 - intros _. apply Peano.le_n_S, Peano.le_0_n.
 - apply Peano.le_n_S.
 - apply Peano.le_S_n.
Qed.

Lemma compare_le_iff n m : (n ?= m) <> Gt <-> n <= m.
Proof.
 revert m; induction n; destruct m; simpl; rewrite ?IHn.
 - now split.
 - split; intros. apply Peano.le_0_n. easy.
 - split. now destruct 1. inversion 1.
 - split; intros. now apply Peano.le_n_S. now apply Peano.le_S_n.
Qed.

Lemma compare_antisym n m : (m ?= n) = CompOpp (n ?= m).
Proof.
 revert m; induction n; destruct m; simpl; trivial.
Qed.

Lemma compare_succ n m : (S n ?= S m) = (n ?= m).
Proof.
 reflexivity.
Qed.


(* BUG: Ajout d'un cas * après preuve finie (deuxième niveau +++*** ) :
    *  --->   Anomaly: Uncaught exception Proofview.IndexOutOfRange(_). Please report. *

(** ** Minimum, maximum *)

Lemma max_l : forall n m, m <= n -> max n m = n.
Proof.
 exact Peano.max_l.
Qed.

Lemma max_r : forall n m, n <= m -> max n m = m.
Proof.
 exact Peano.max_r.
Qed.
```

```
2 subgoals
n : nat
IHn : forall m : nat, (n ?= m) <> Gt <-> n <= m
m : nat
H : n <= m
______________________________________(1/2)
S n <= S m
______________________________________(2/2)
n <= m
```

Messages | Errors | Jobs

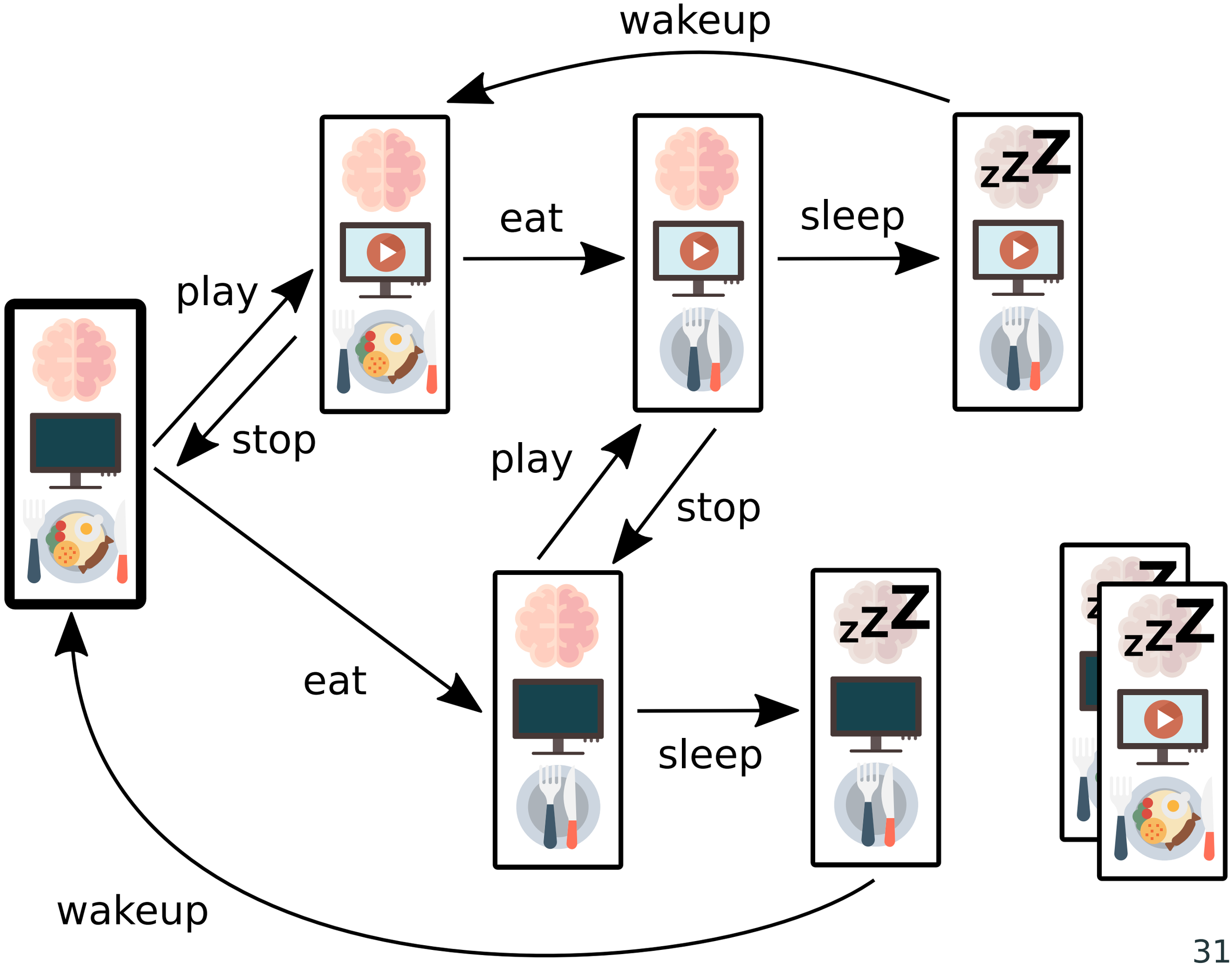
Ready in Nat, proving compare_le_iff | Line: 211 Char: 18 | Coq is ready | 0 / 0

Coq Agda Why3 TLA$^+$ B/EventB

RIEN À PROUVER SOI MÊME

LE MODEL CHECKING

UNE TECHNIQUE ABORDABLE


wakeup
play
eat
sleep
stop
play
stop
eat
sleep
wakeup

ÉTAT

contenu des cabines

ADN

ÉTAT DE DÉPART

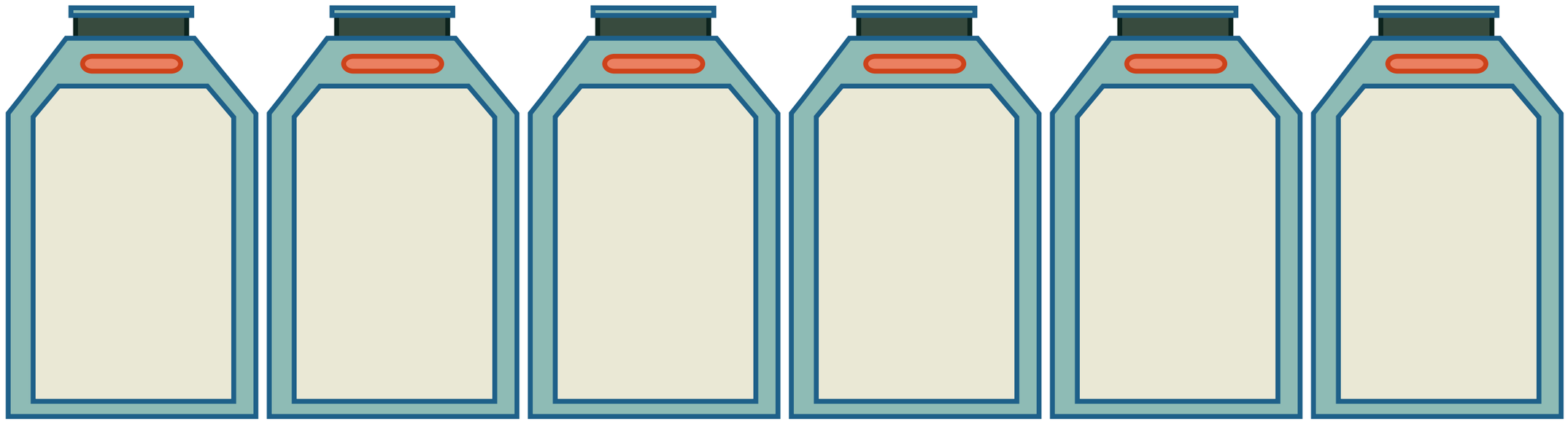
ÉTAT DE DÉPART

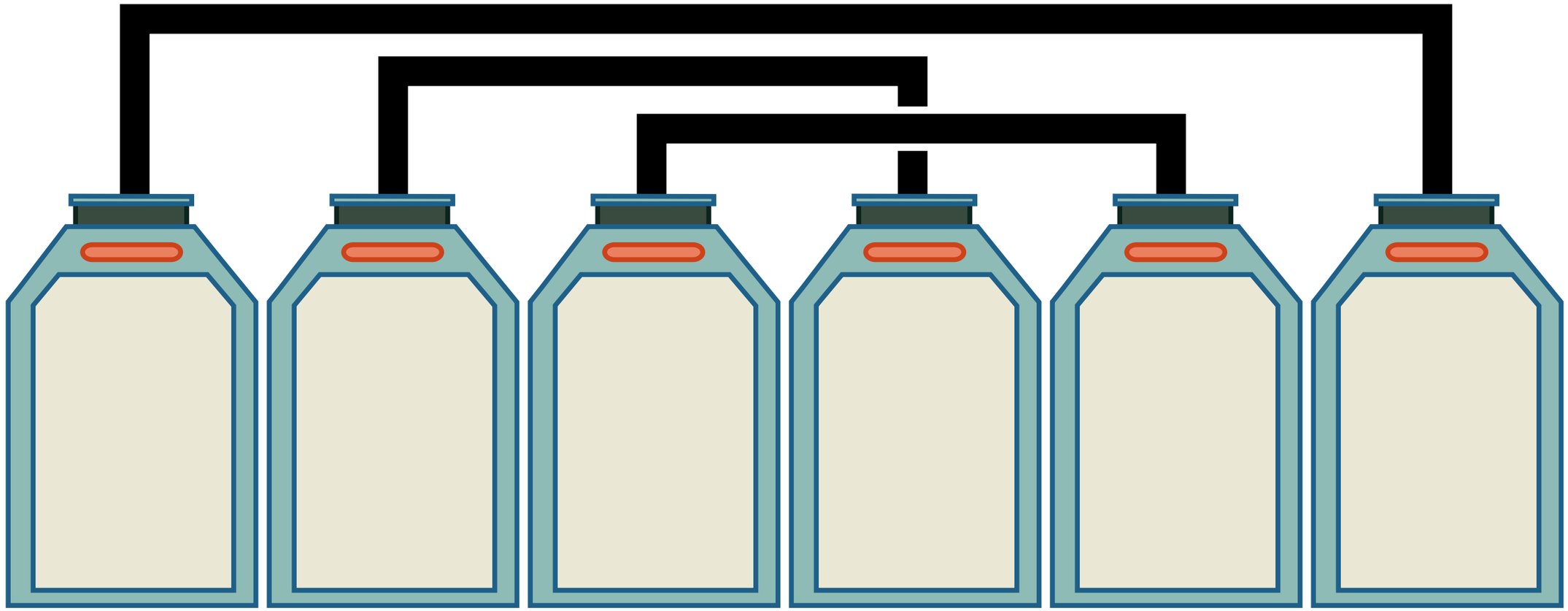
IL NE SE PASSE PAS N'IMPORTE QUOI

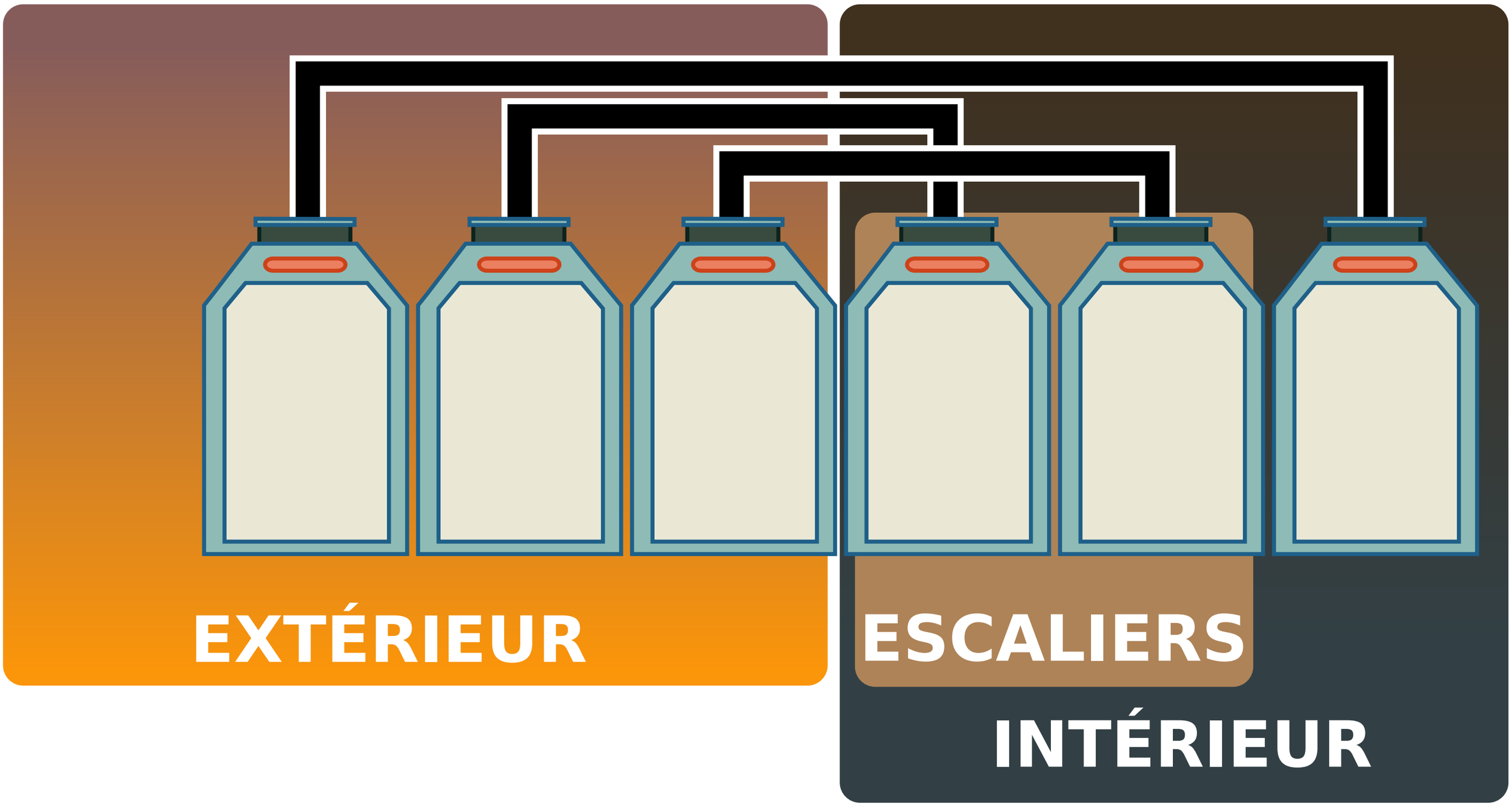
ÉTAT DE DÉPART

IMPOSSIBLE D'ARRIVER AU TÉLÉPORTEUR FINAL SOUS FORME HUMAINE

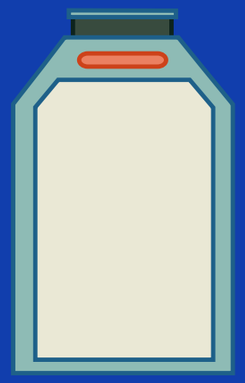

R
R



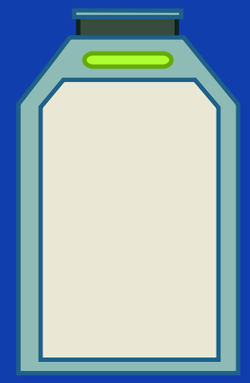



EXTÉRIEUR
ESCALIERS
INTÉRIEUR

ACTION
TÉLÉPORTER

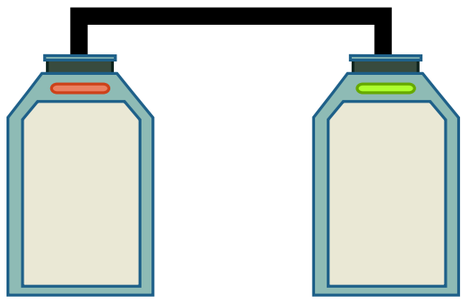


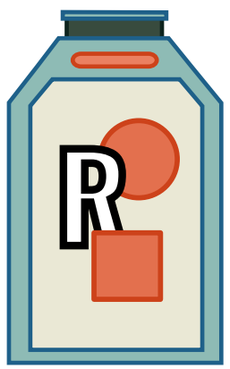
ORIGINE



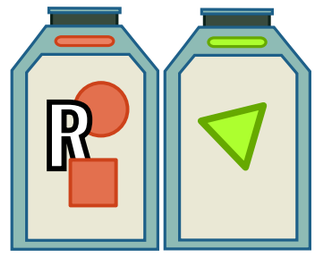
DESTINATION

prérequis

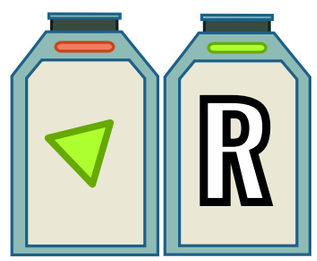


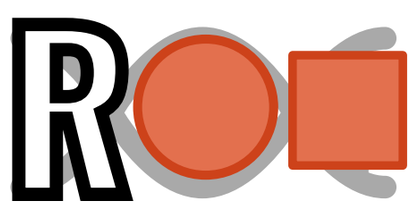


avant

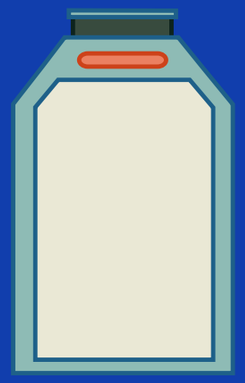


après

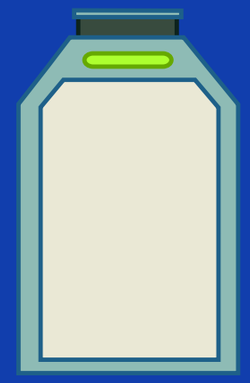




ACTION
PURGER

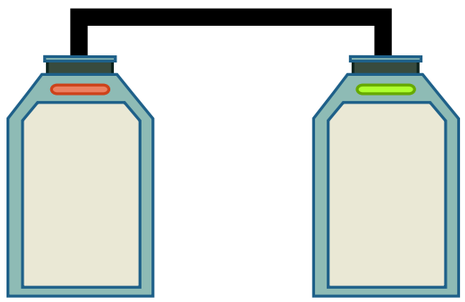


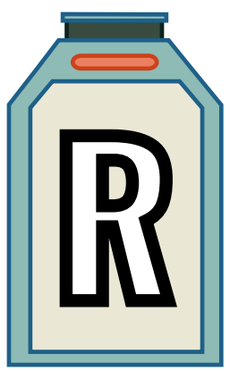
ORIGINE



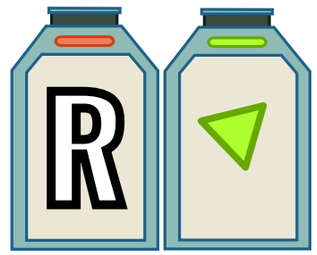
DESTINATION

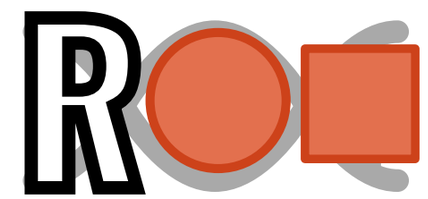
prérequis



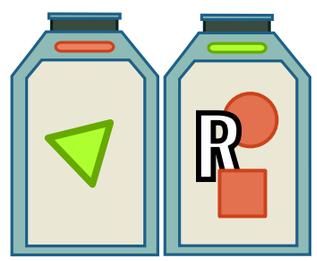


avant

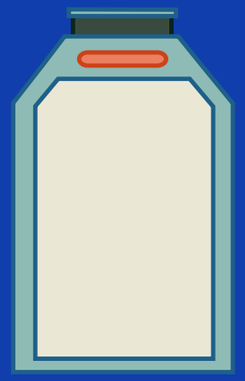




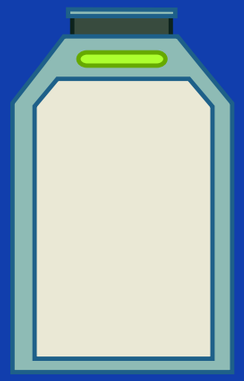
après



ACTION
VOLER



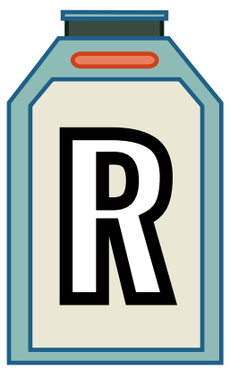
ORIGINE

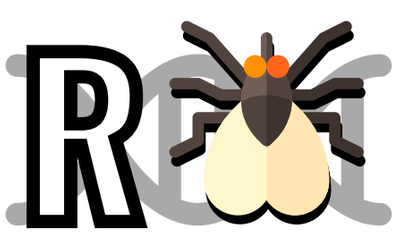


DESTINATION

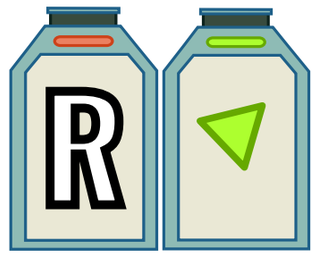
prérequis

OU





avant



après

Vérification de Programme

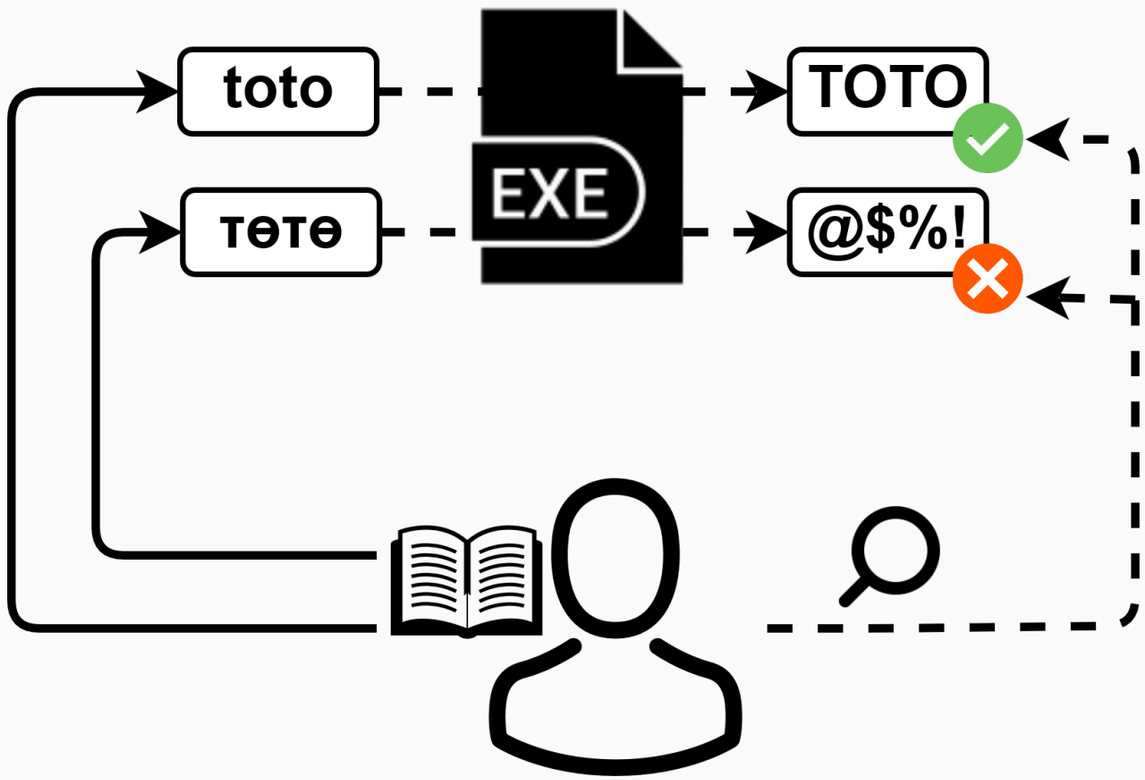
Vérification de Programme

Correct par construction ?

Mais j'ai déjà un programme écrit, qu'est-ce que je peux faire ?

Les méthodes formelles ne vous abandonnent pas.

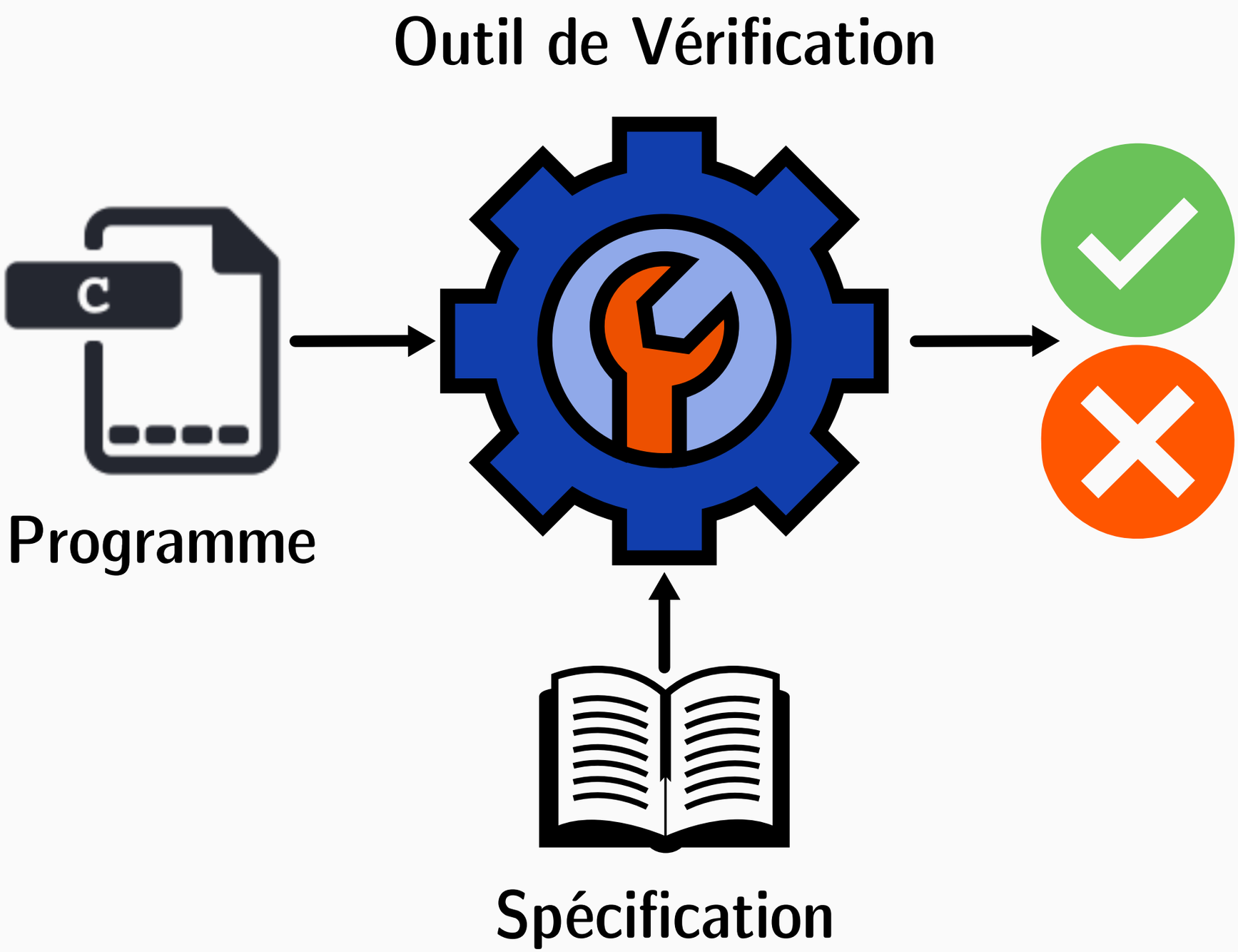
Test Manuel



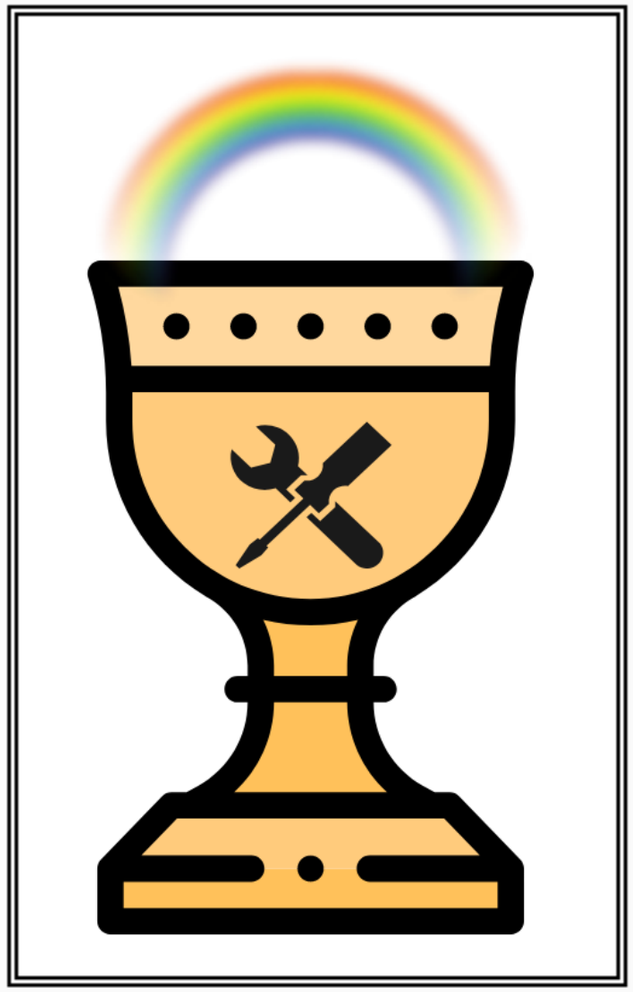
Test Manuel

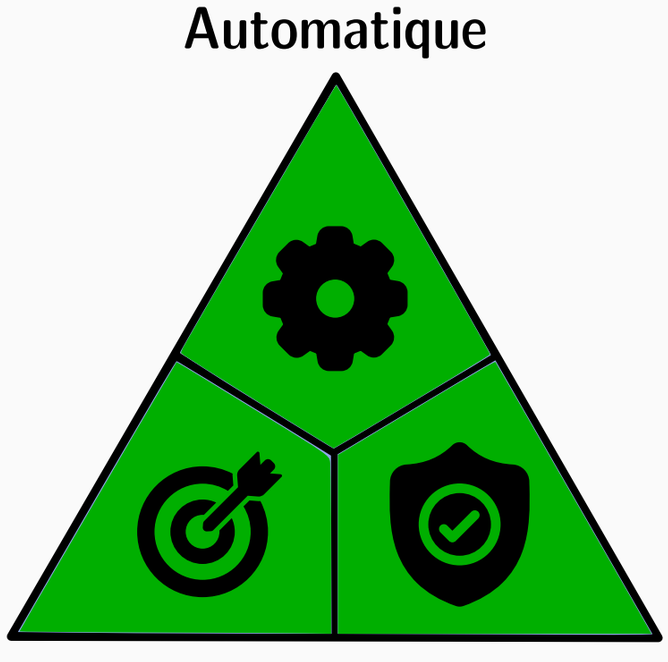
- Trouve des Bugs ✓
- Preuve ✗
- Automatique ✗

Vérification de Programme

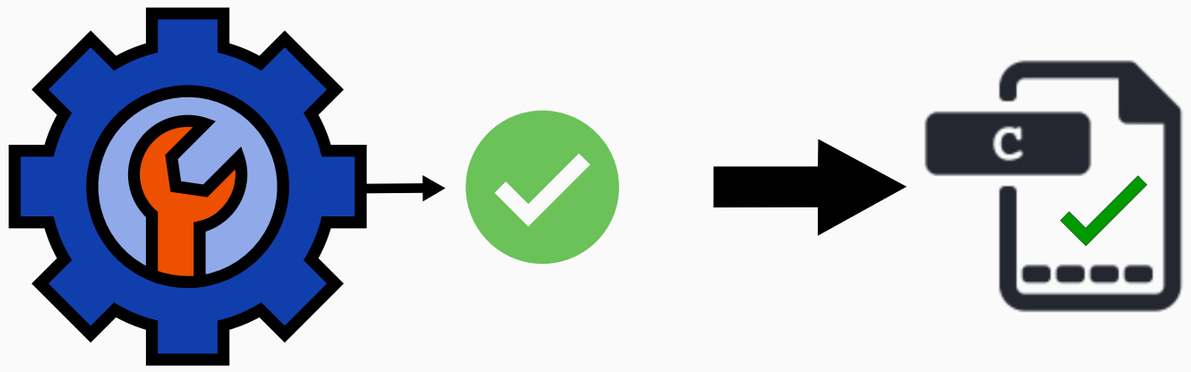


Outil de Vérification Idéal

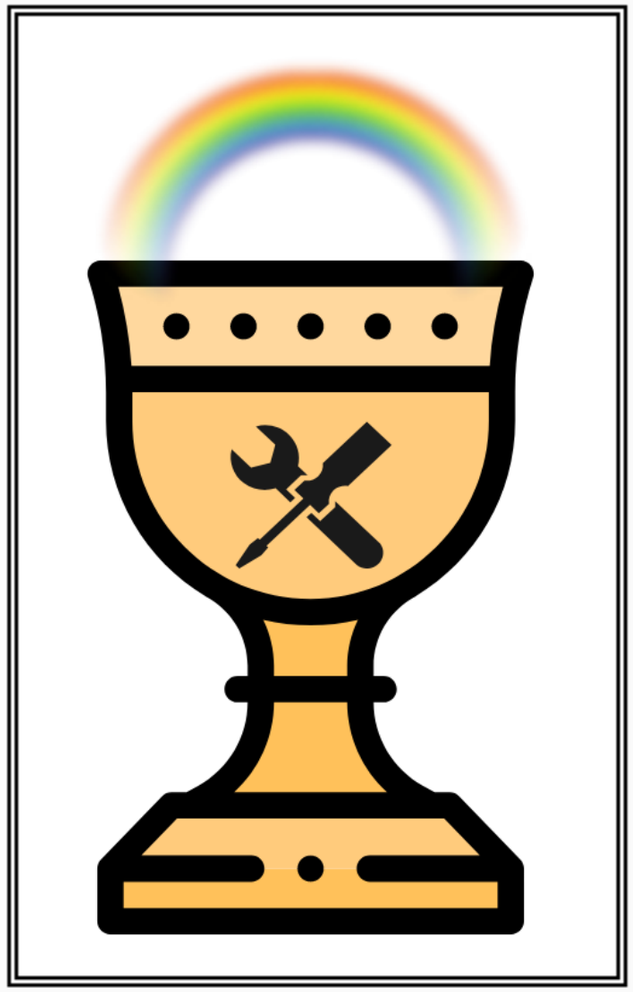


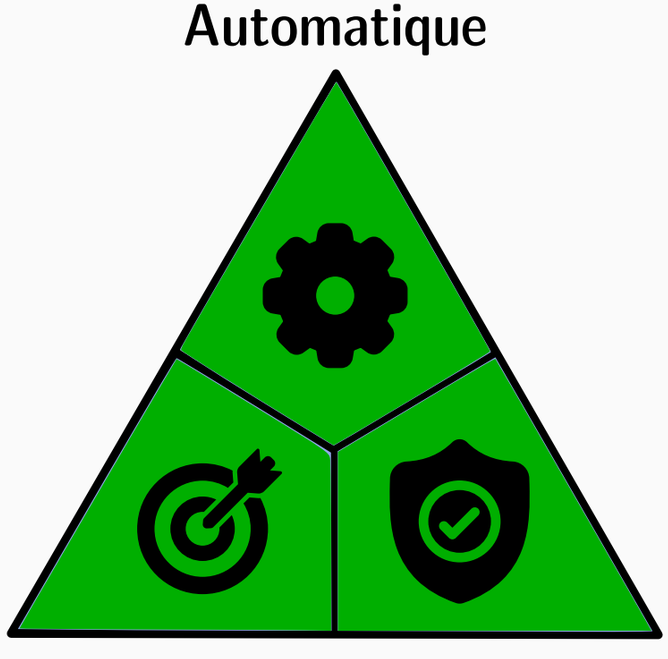


Correct :

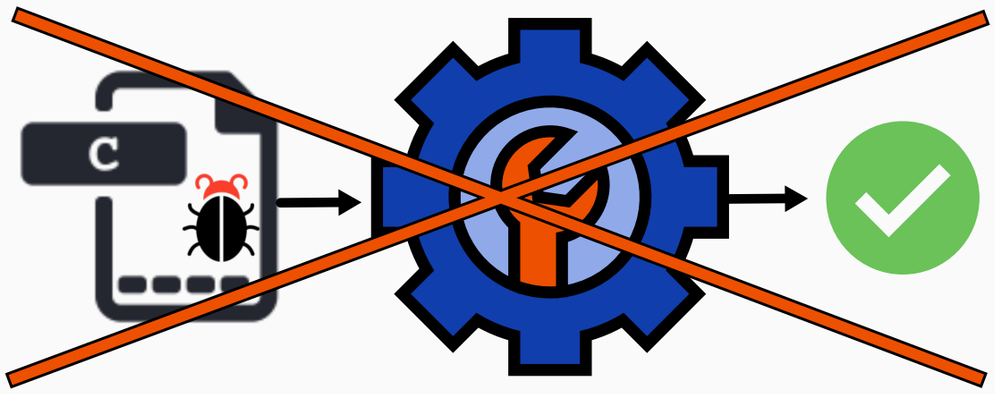


Outil de Vérification Idéal

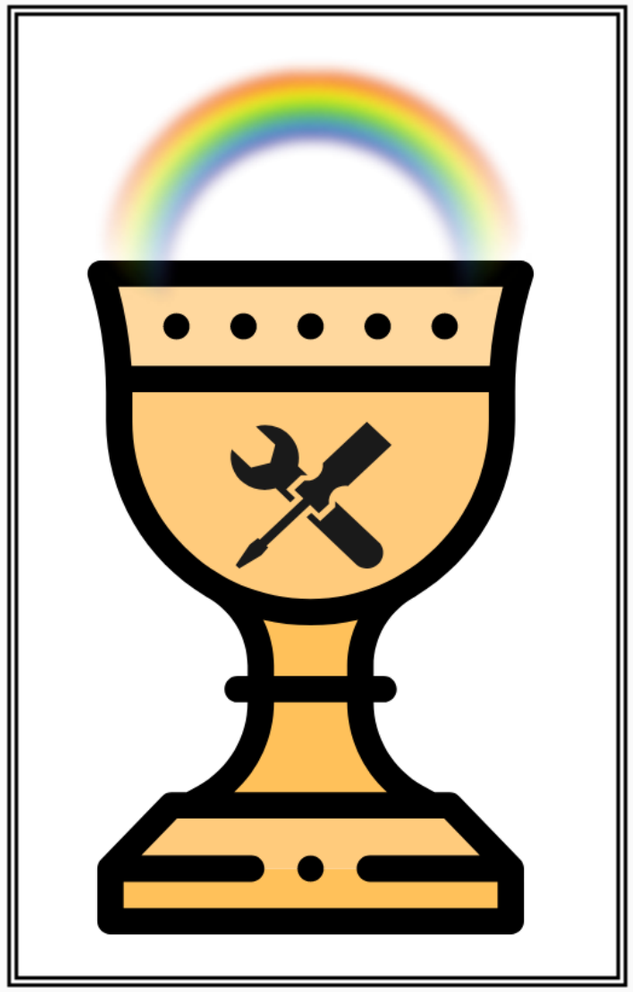


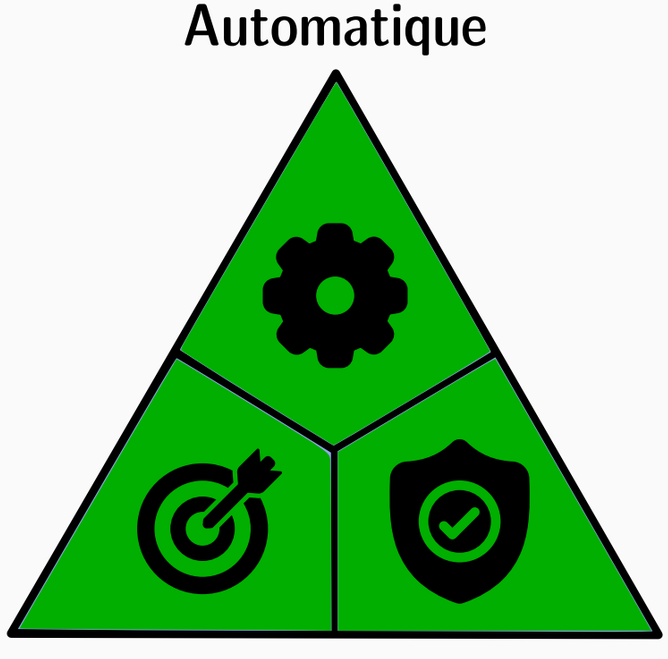


Correct :

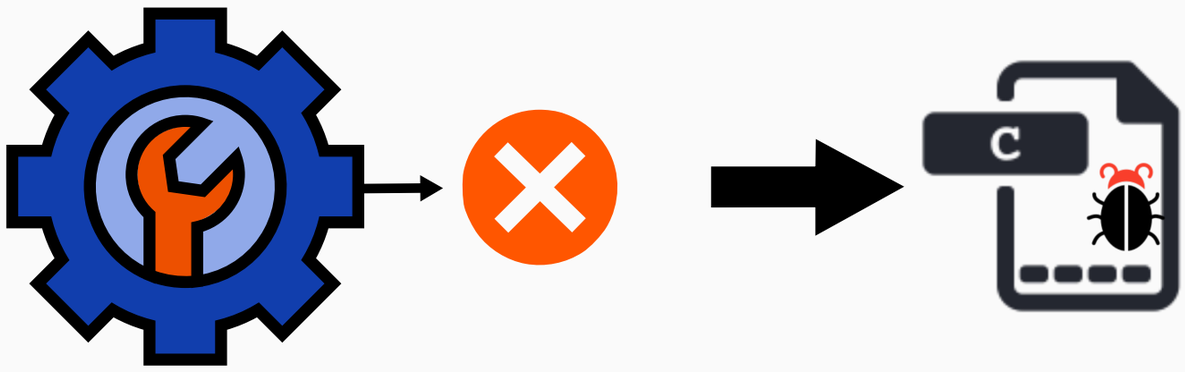


Outil de Vérification Idéal

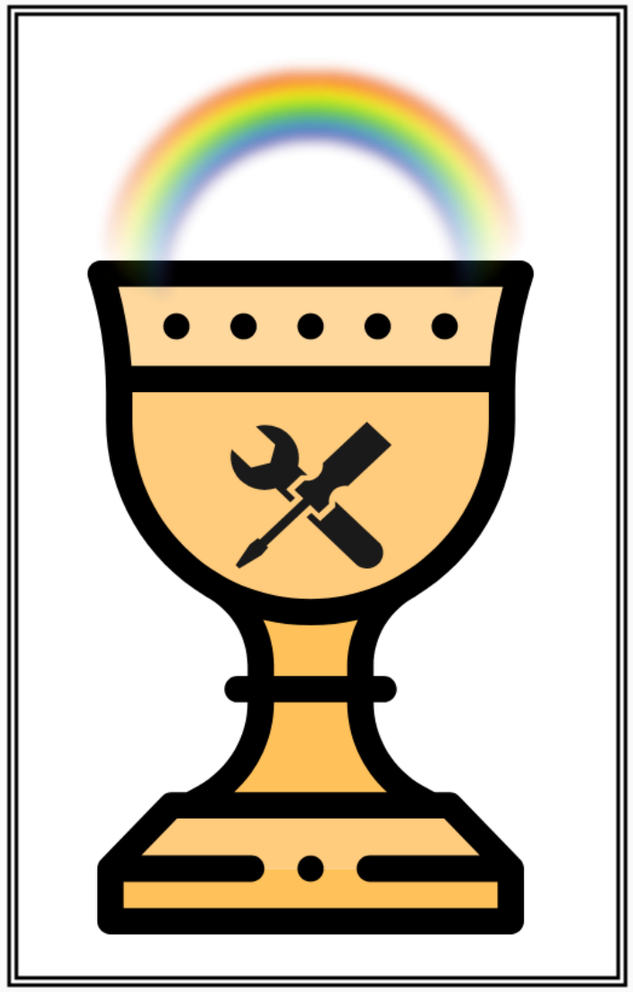


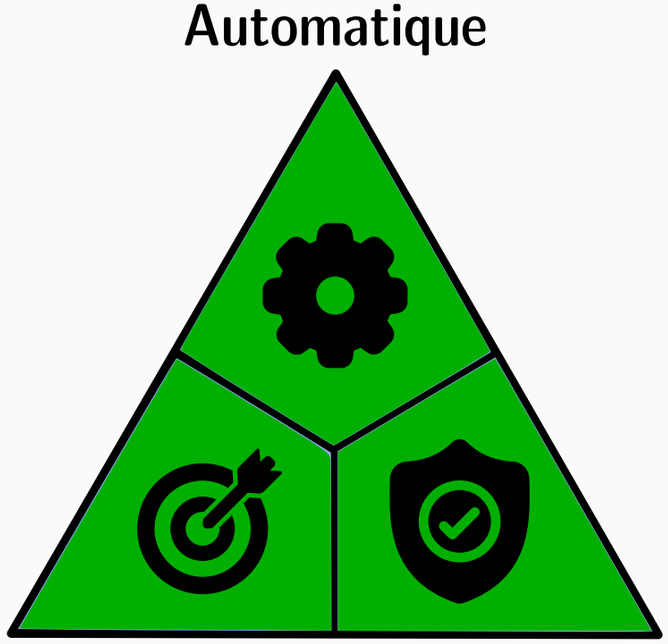


Précis :

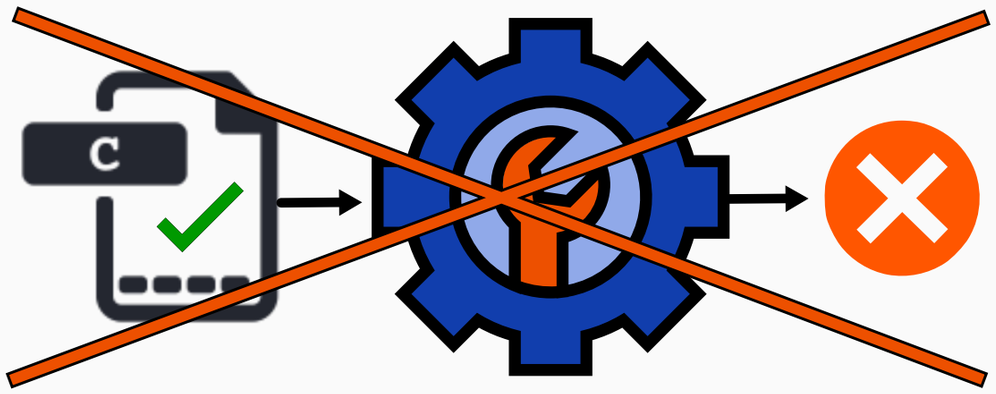


Outil de Vérification Idéal

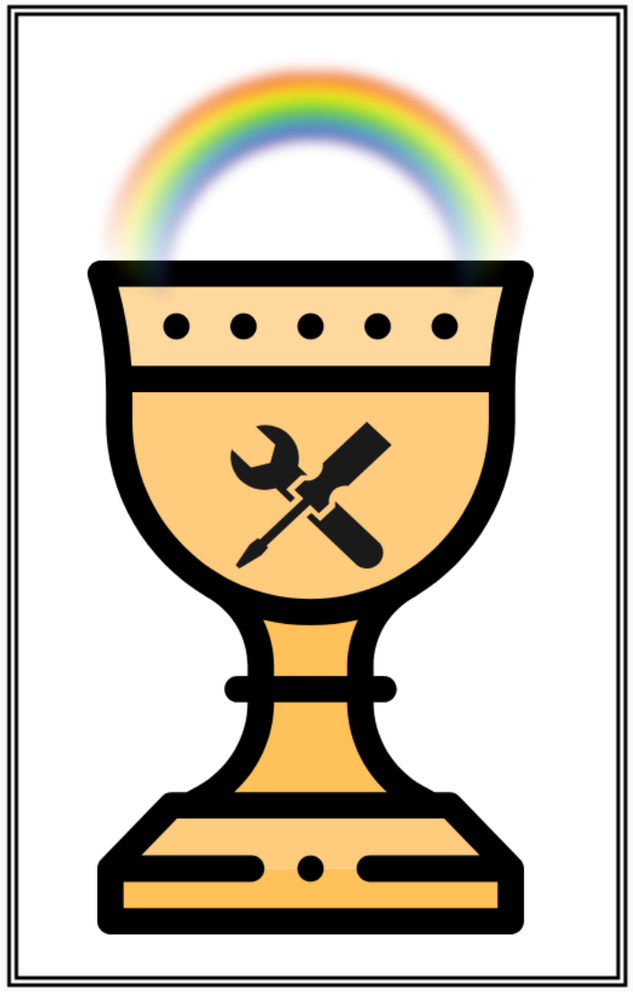


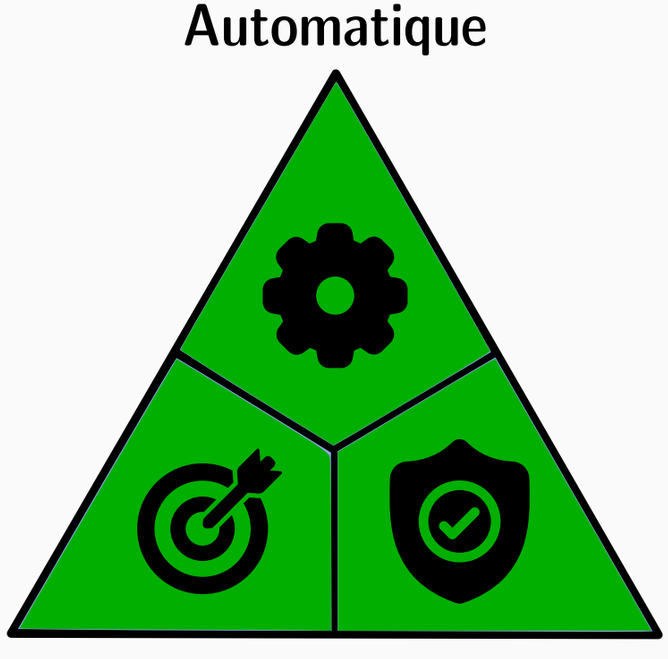


Précis :

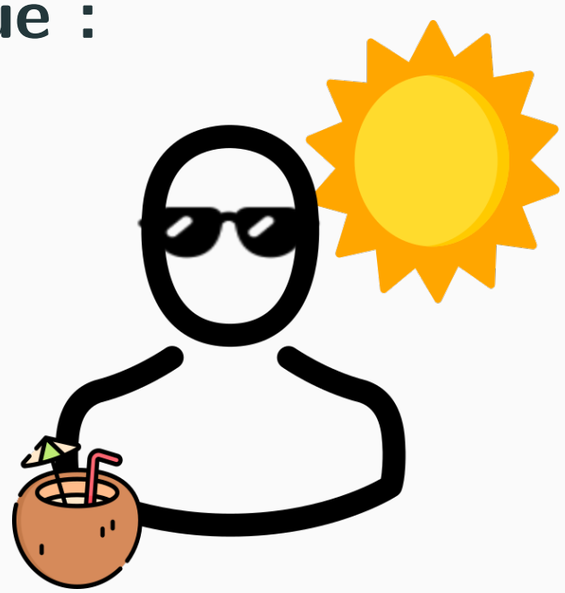


Outil de Vérification Idéal

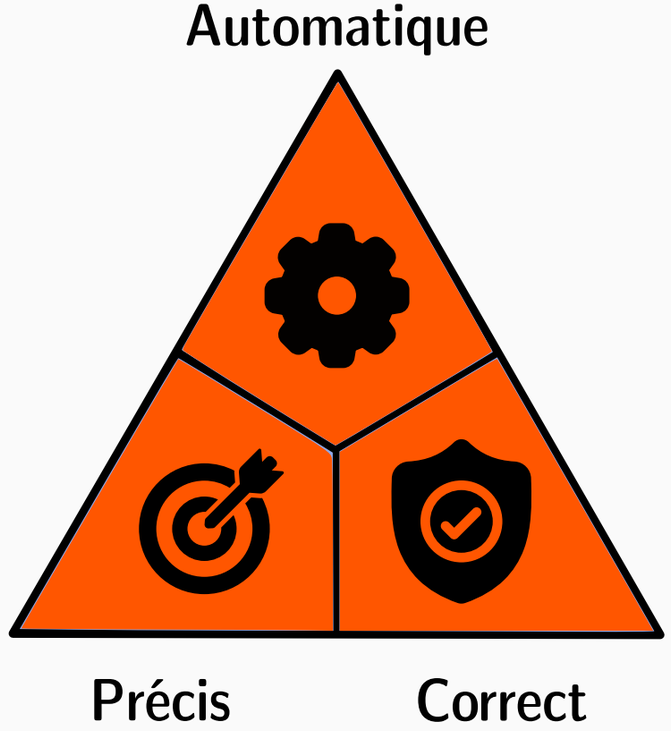




Automatique :



Outil de Vérification Idéal

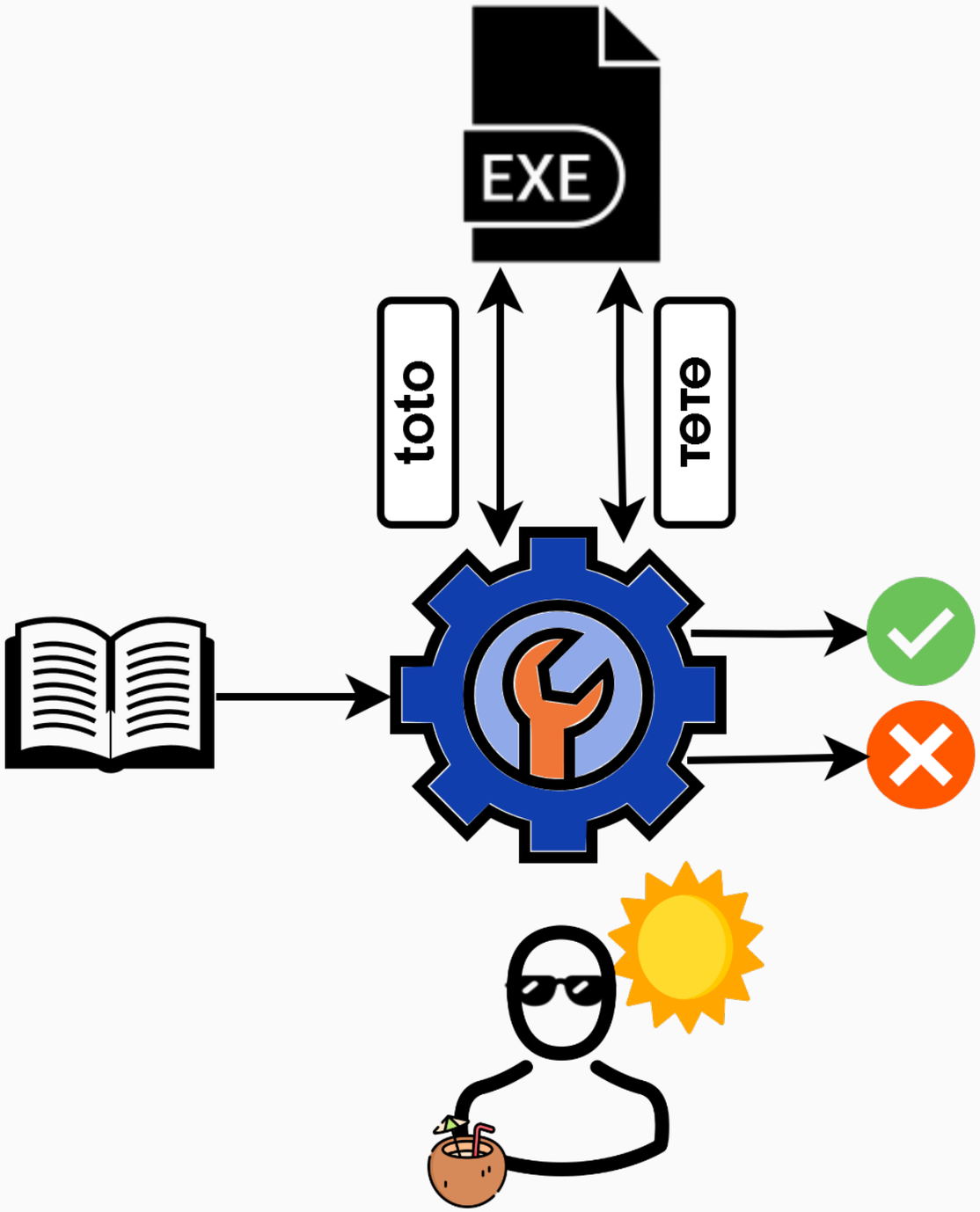


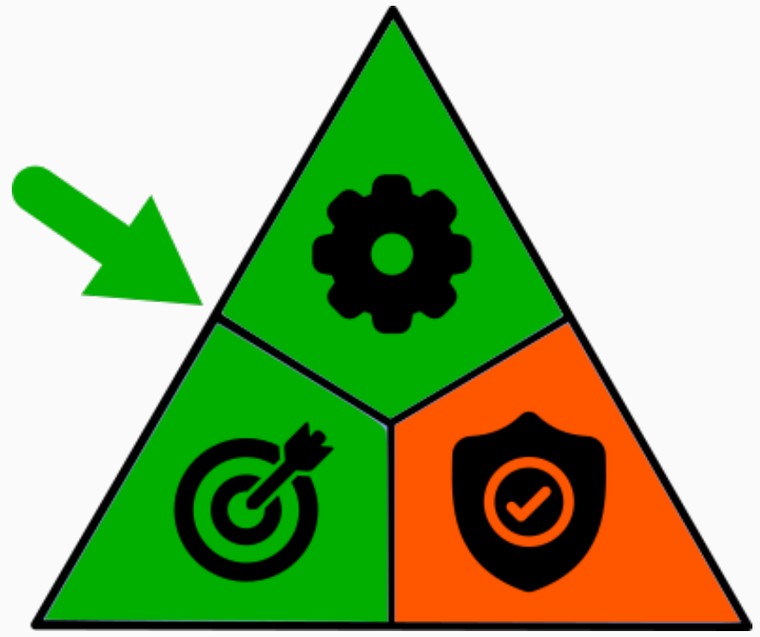
Théorème de Rice (1953)

Impossible de construire un tel outil.

Génération de Tests Automatique

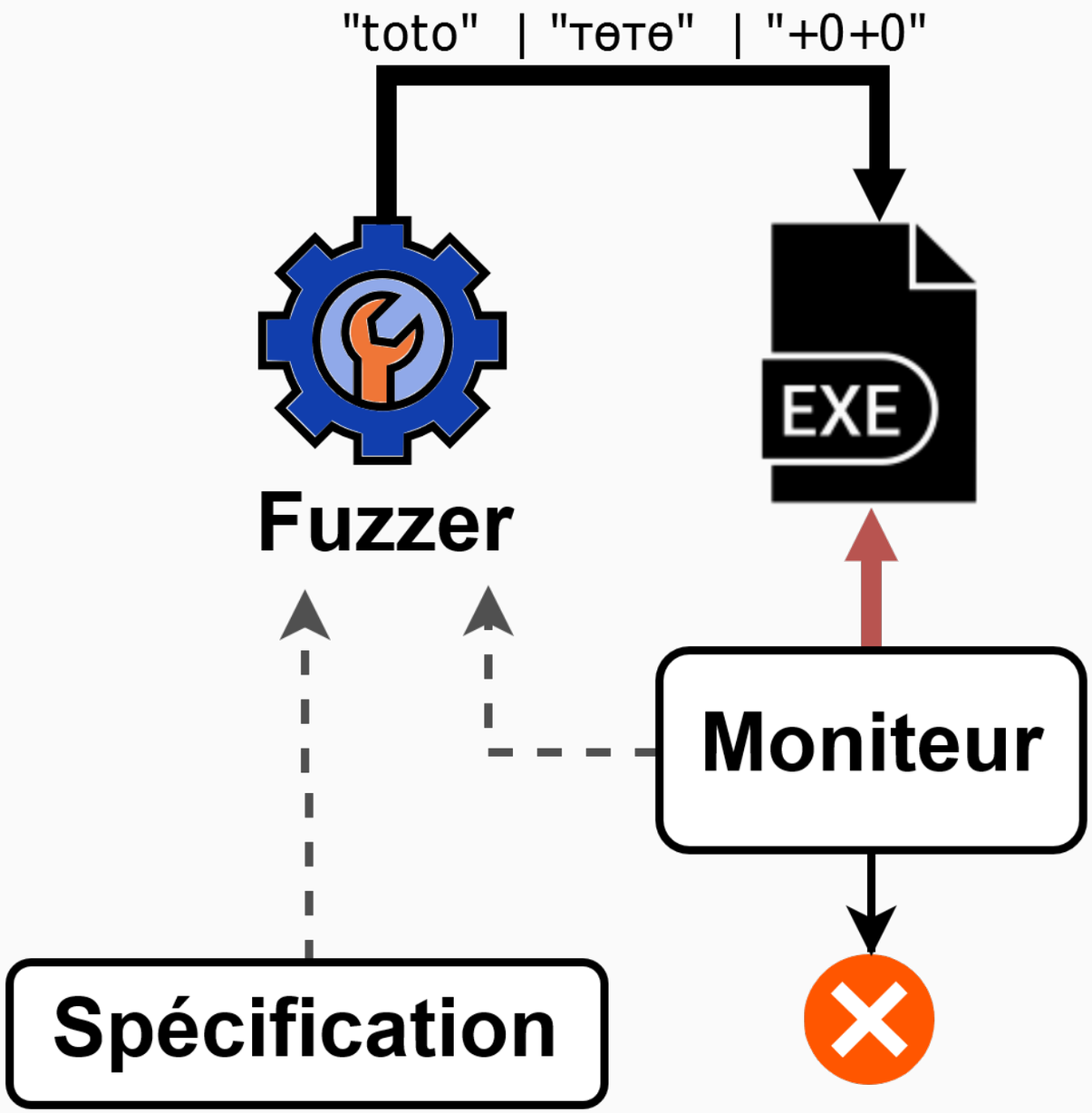
Test Générés Automatiquement





- Automatique & Précis
- Plus efficace que le test manuel
- Pas correct

Fuzzing



Principe

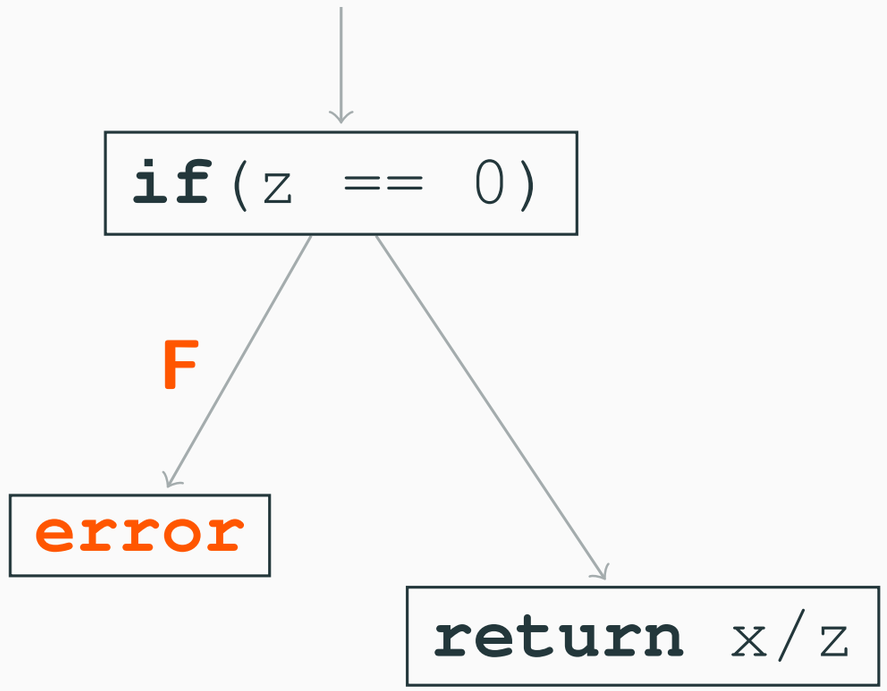
- Test automatique intensif
- Entrées générées aléatoirement

Problème du Fuzzing

```
int main(int x,int y){
   int z = x * y - 48;
   if (z == 0)
      error();
   else
      return x / z;
}
```

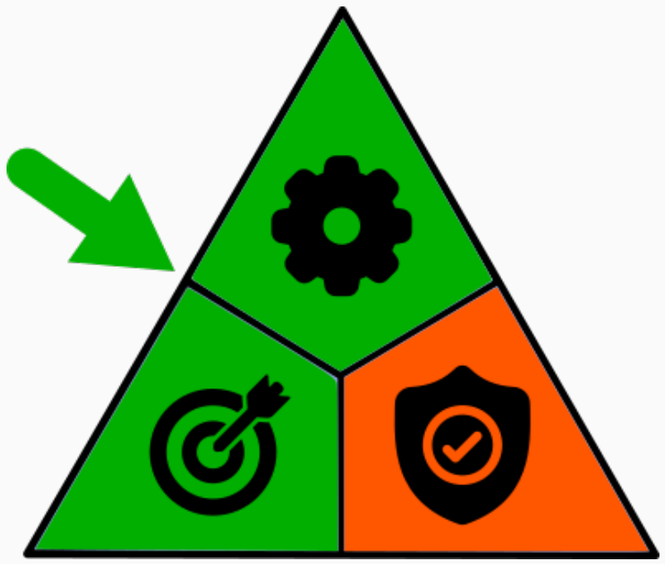
Alternative: Exécution Symbolique

```
int main(int x,int y){
  int z = x * y - 48;
  if (z == 0)
    error();
  else
    return x / z;
}
```



Trouver un bug :

Récap: Recherche de Bugs



Fuzzing

- Génère plein d'entrées rapidement
- Peut manquer des chemins compliqués

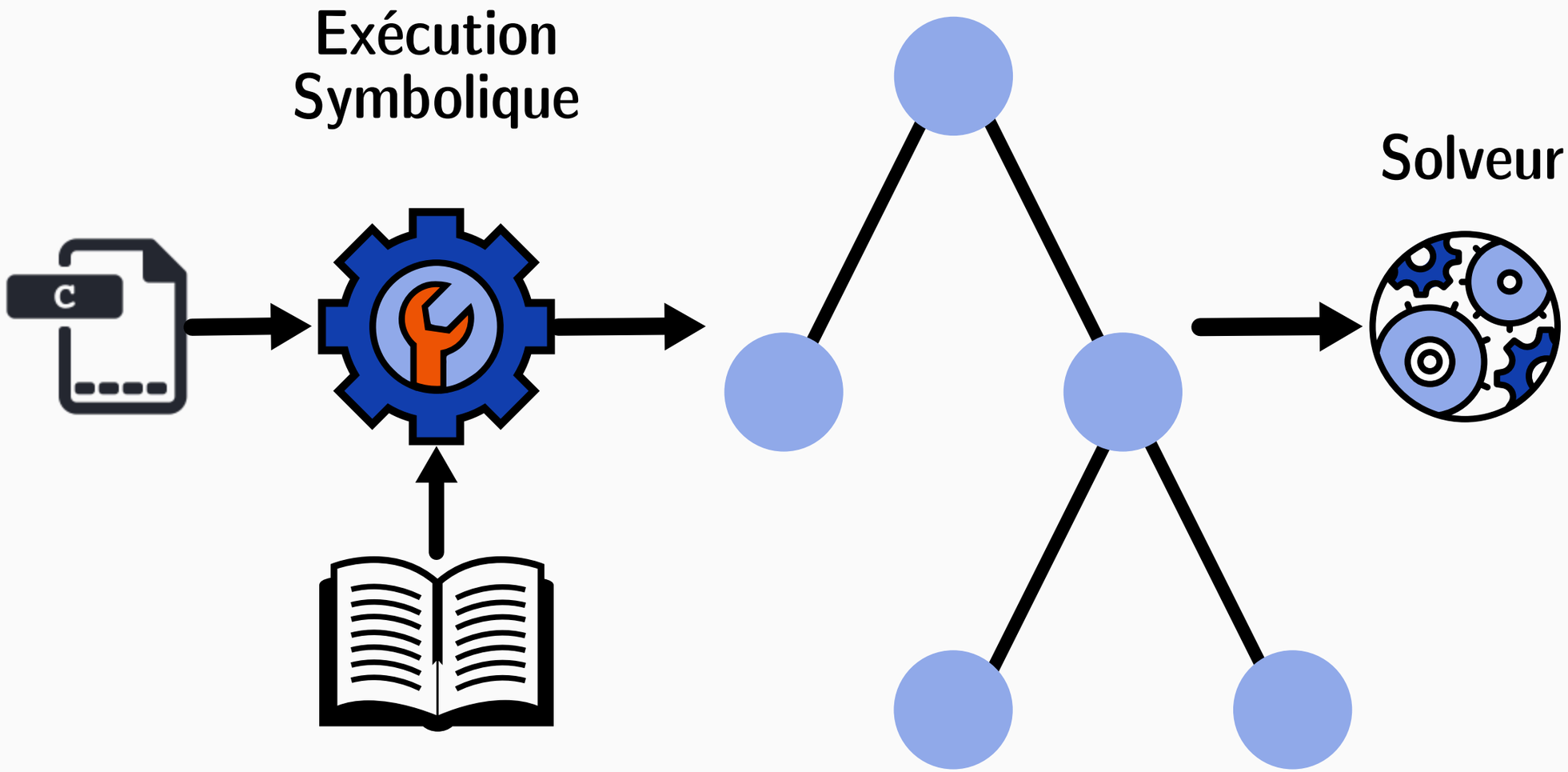
Exécution Symbolique

- Meilleure couverture des chemins compliqués
- Pas aussi rapide que le fuzzing

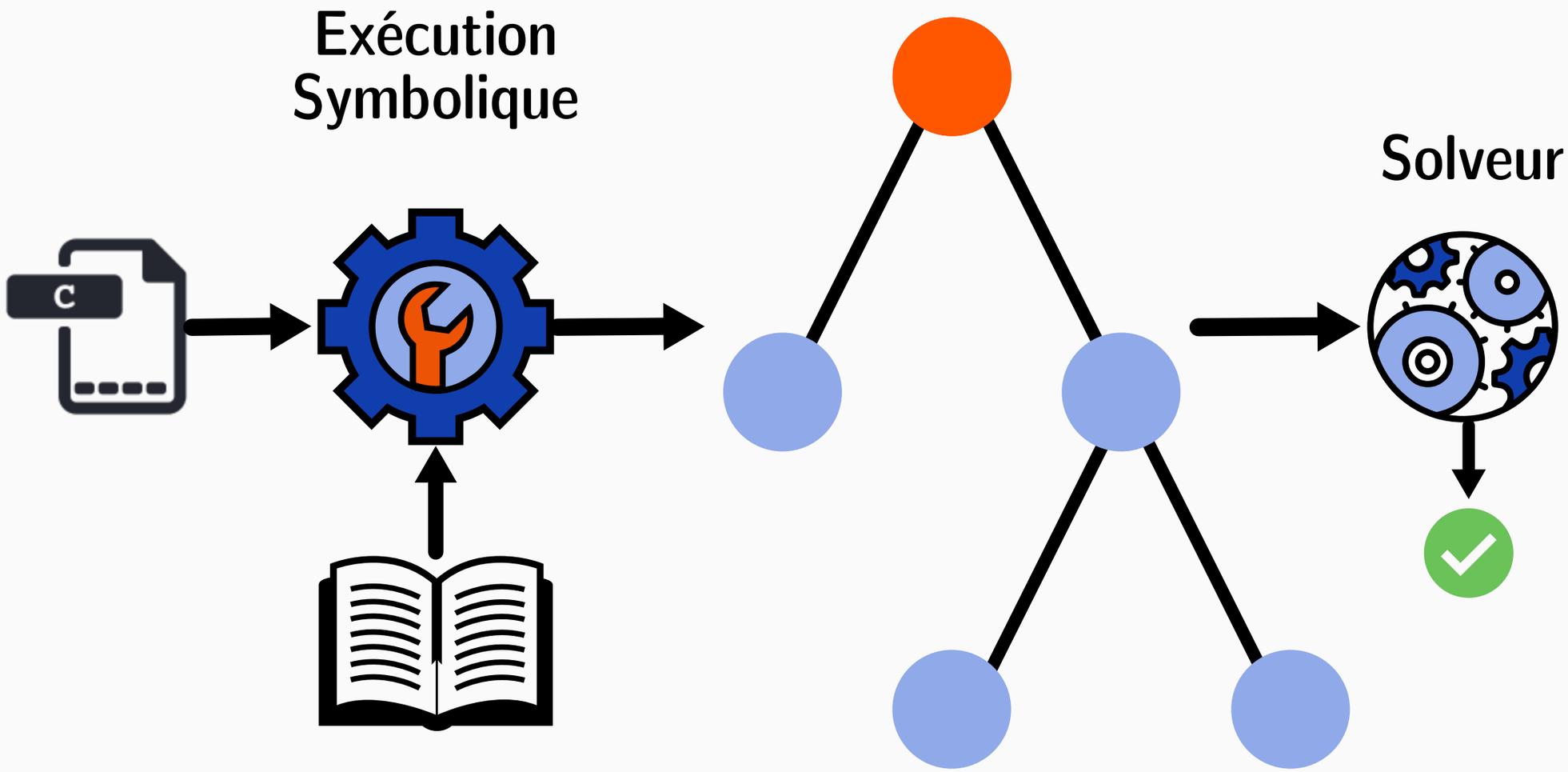
On peut combiner les deux pour plus d'efficacité

Vérification Symbolique et Preuve de Théorème

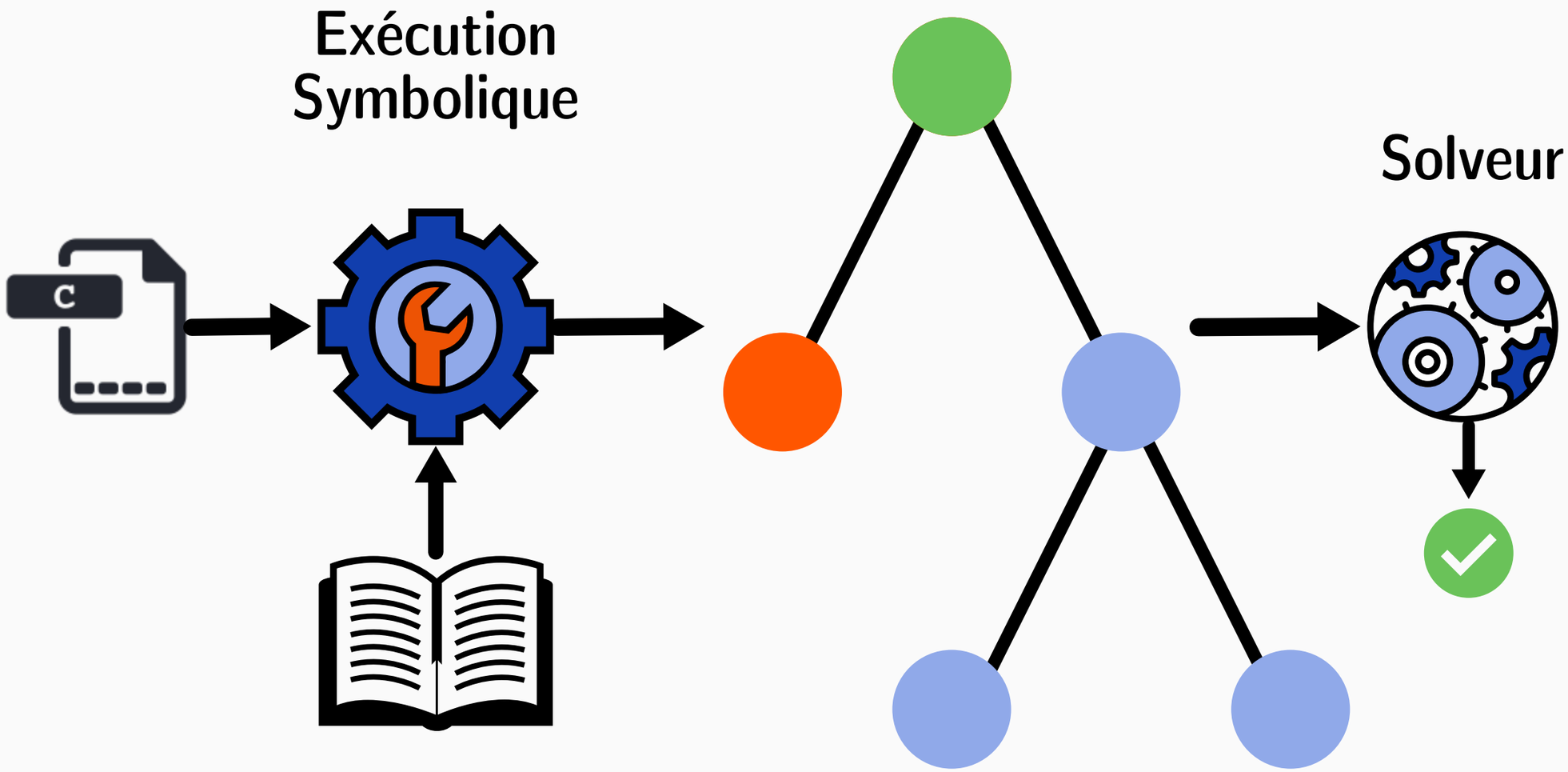
Vérification Symbolique



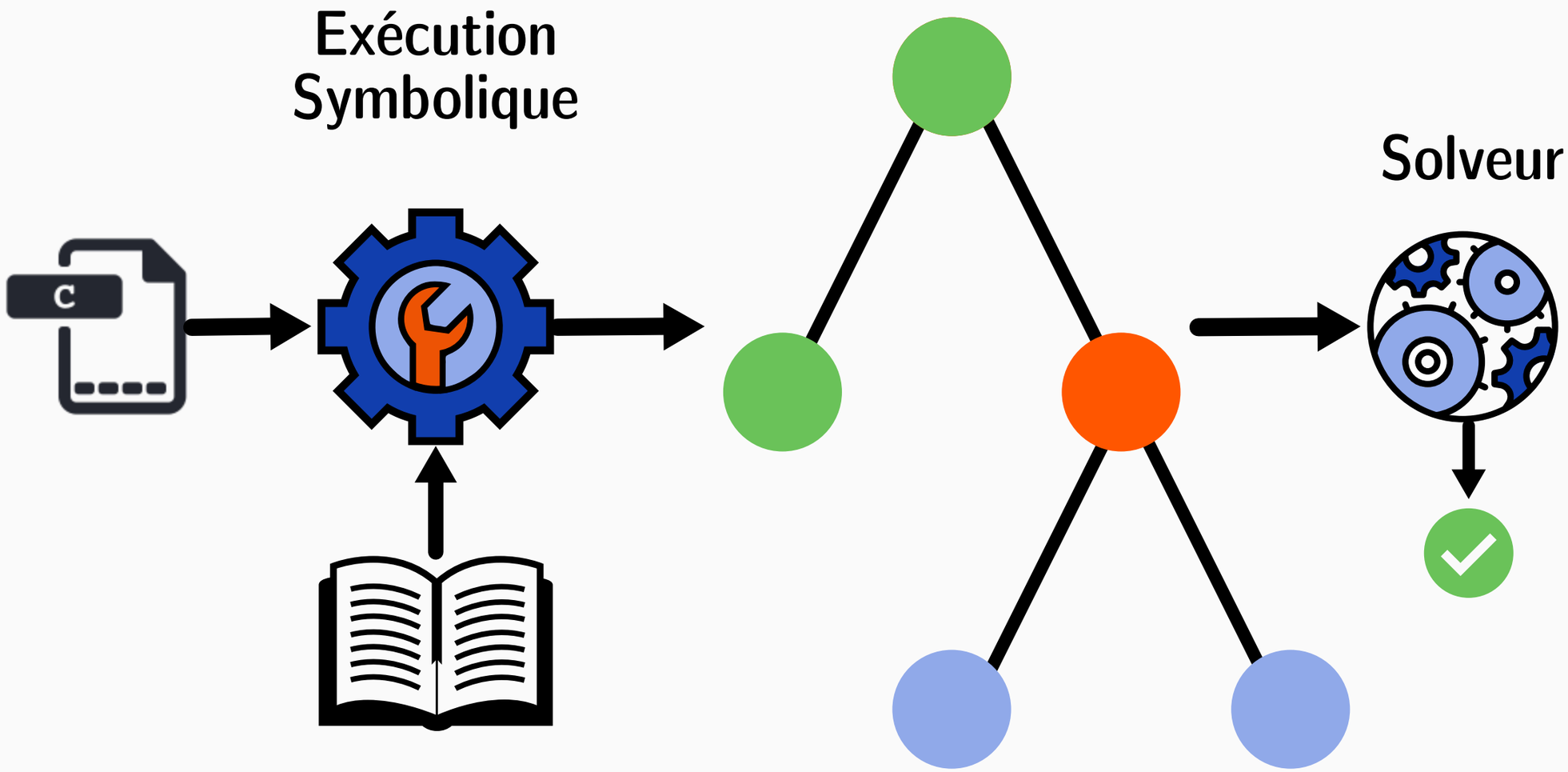
Vérification Symbolique



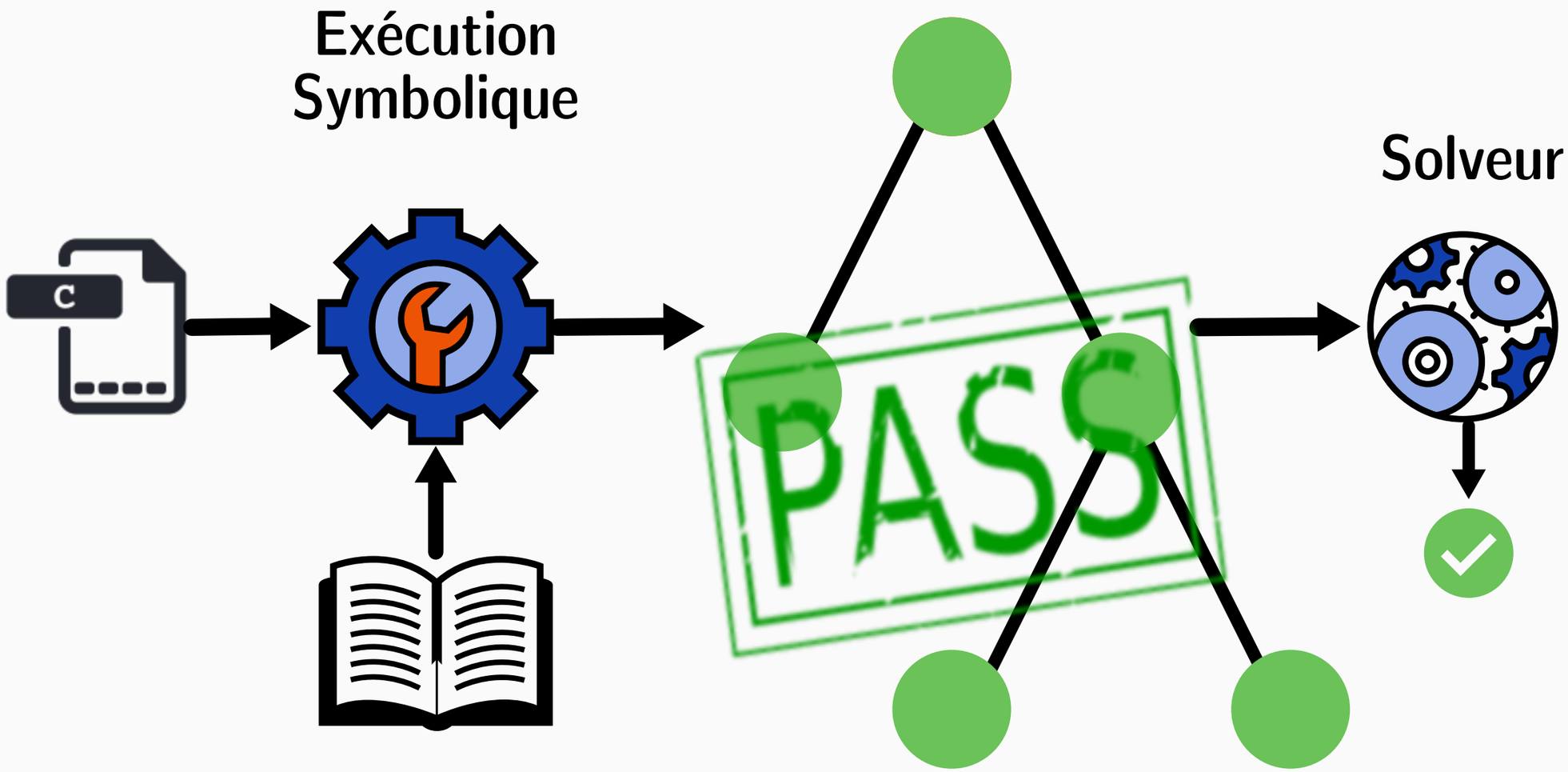
Vérification Symbolique



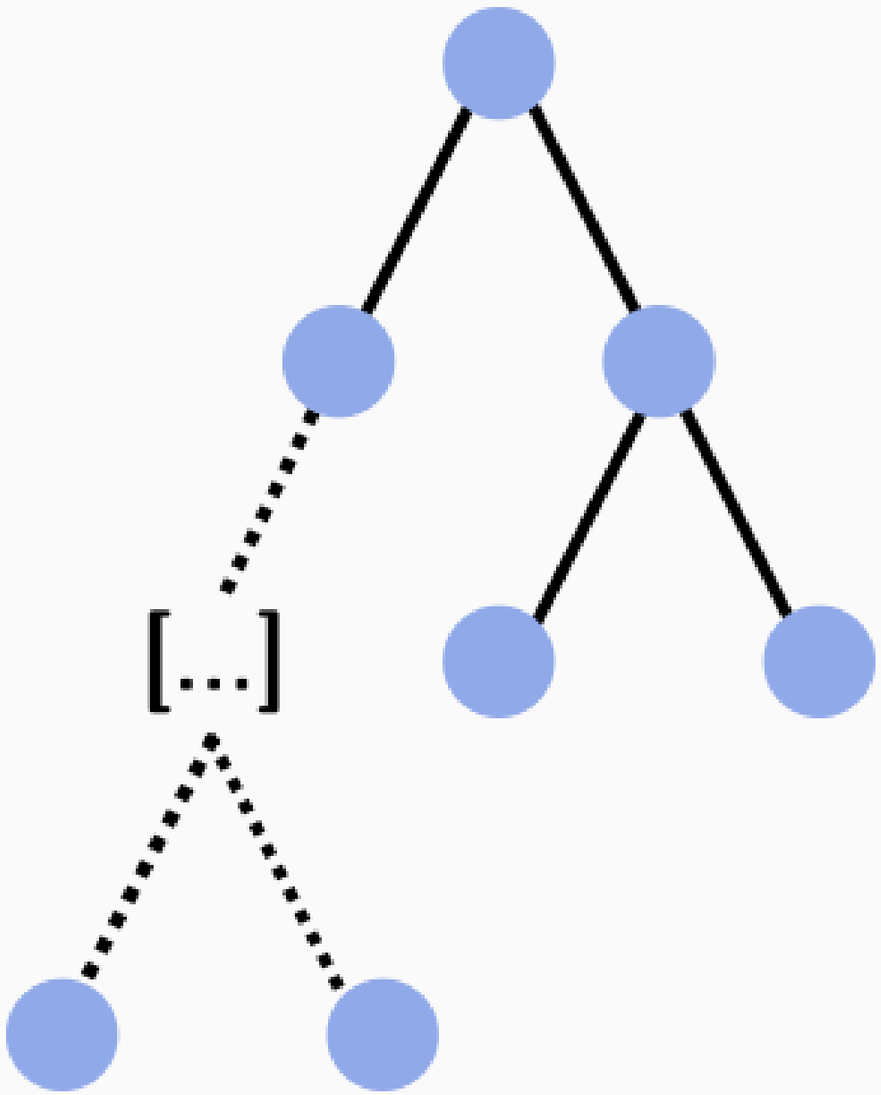
Vérification Symbolique



Vérification Symbolique



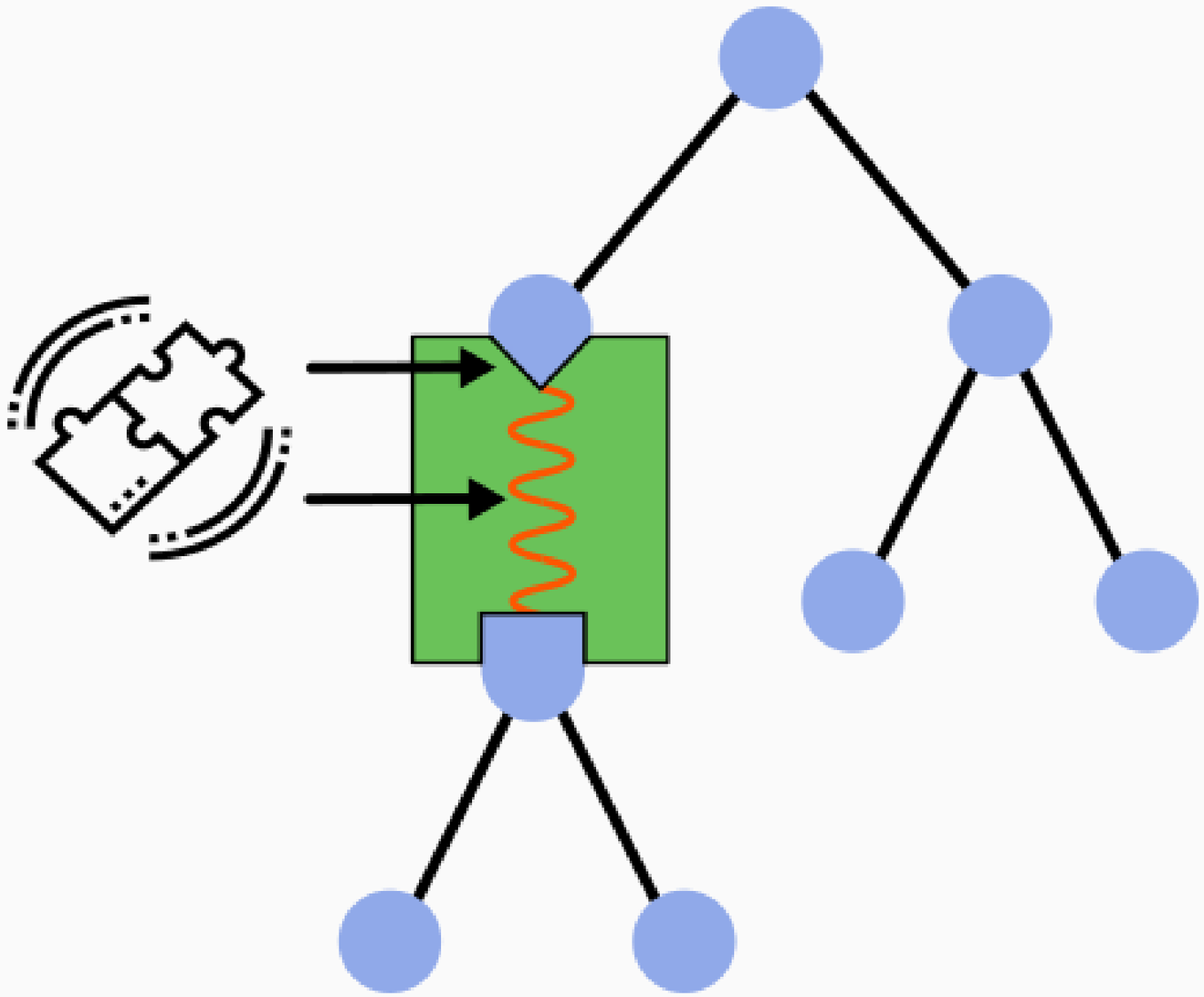
Problème: Programmes Trop Complexes



Cas difficiles

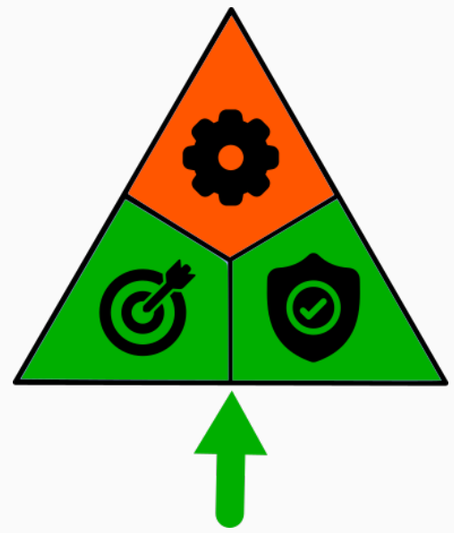
- Boucles non bornées
- Fonctions récursives

Solution: Abstraire des Parties de Programme avec des Théorèmes



On doit prouver

- Précondition
- Transition pré/post

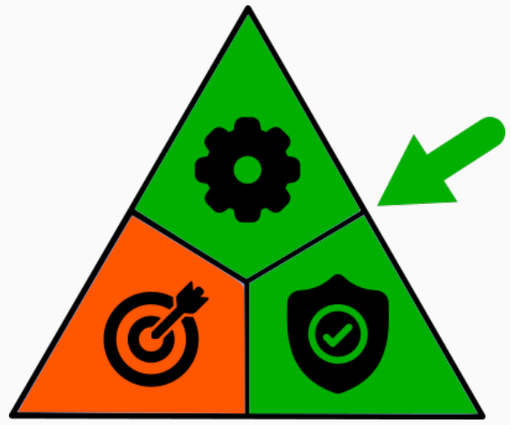


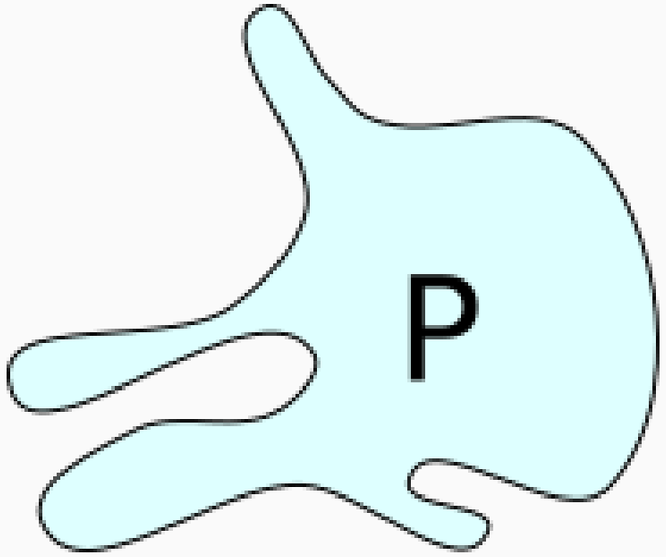
Preuve Automatique de Programme

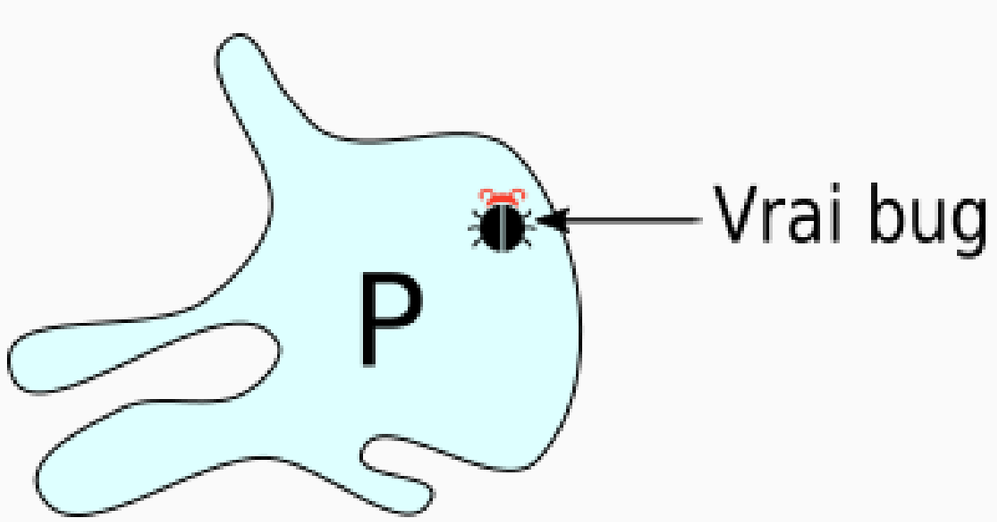
Interprétation Abstraite

But

Surapproximation du programme pour la preuve



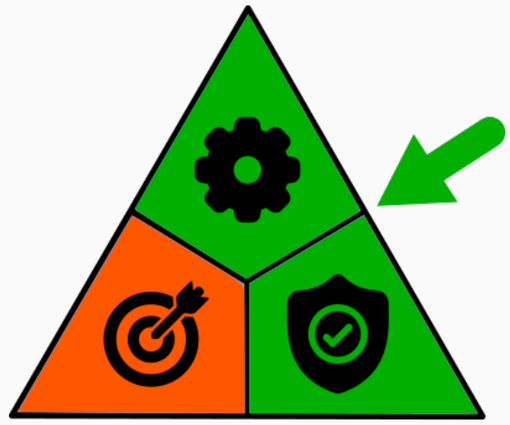


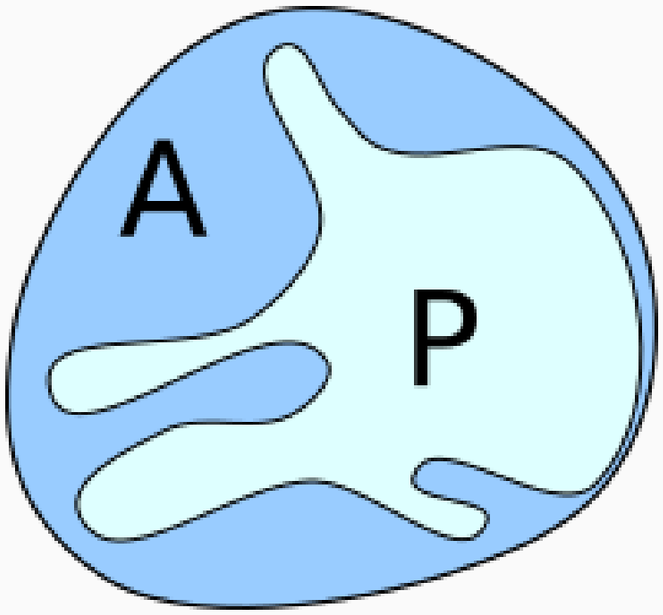


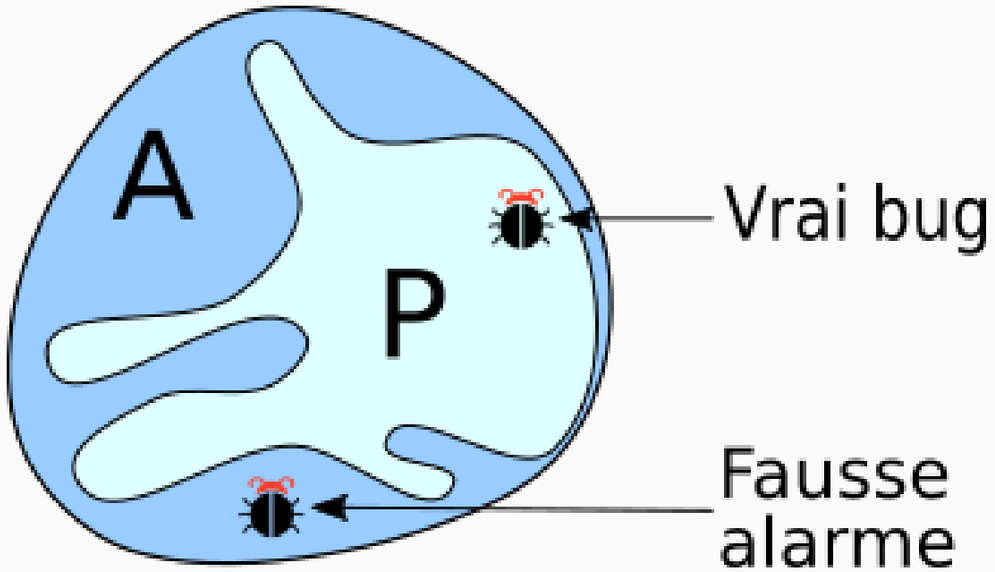
Interprétation Abstraite

But

Surapproximation du programme pour la preuve





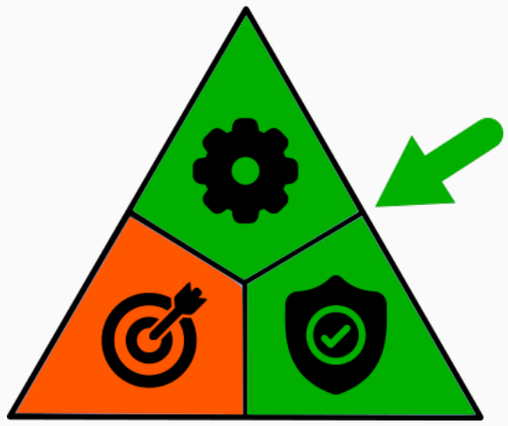


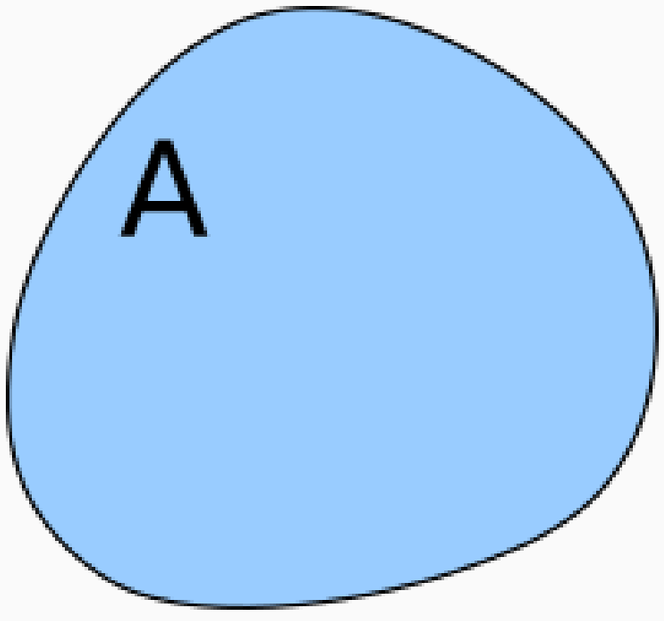
✓(A) $\Longrightarrow$ ✓(P)

Interprétation Abstraite

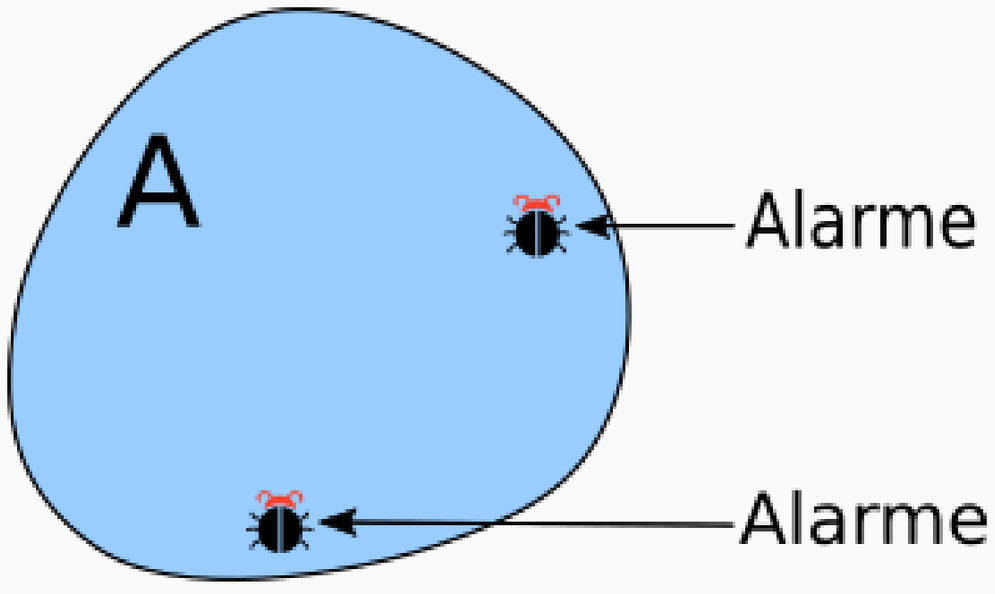
But

Surapproximation du programme pour la preuve





✓(A) $\Longrightarrow$ ✓(P)

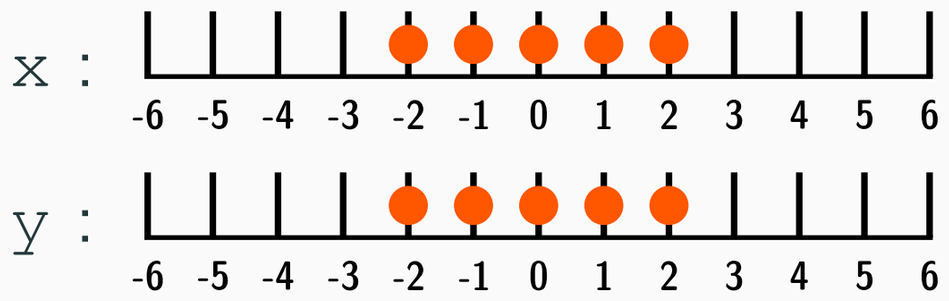


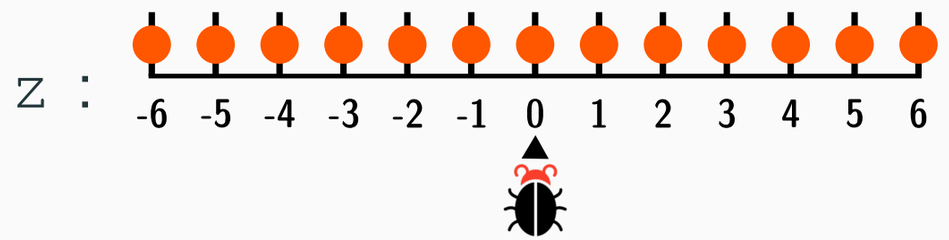
✗(A) $\Longrightarrow$?(P)

Exemple: Intervalles

```
/*————————————————————*/
/*  Pre: x ∈ [−2;2]   */
/*       y ∈ [−2;2]   */
/*————————————————————*/
int foo(int x, int y) {
  int z = 2 * x + y;
  assert z ≠ 0;
  return x / z;
}
```

Exécutions concrètes

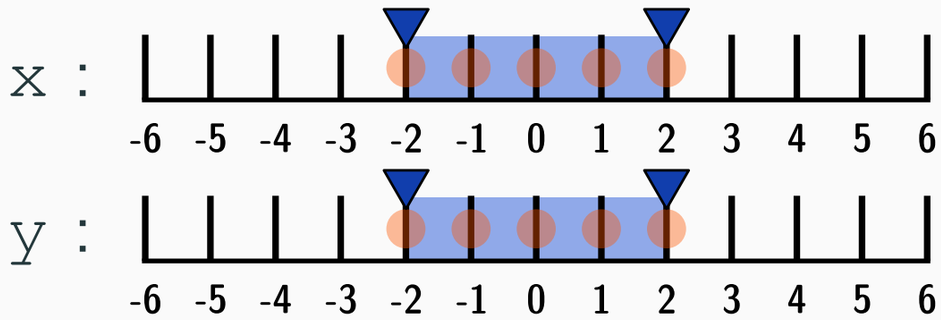


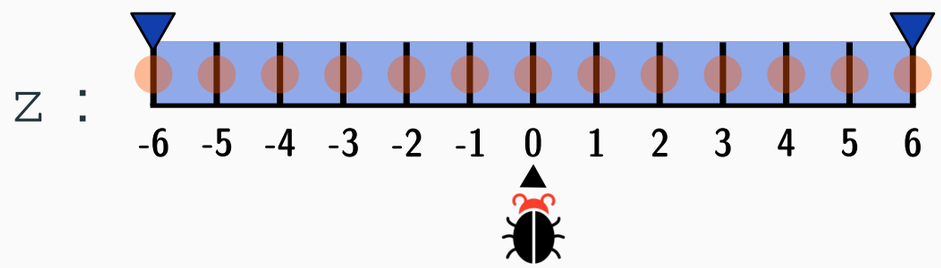


Exemple: Intervalles

```
/*————————————————————*/
/*  Pre: x ∈ [−2;2]   */
/*       y ∈ [−2;2]   */
/*————————————————————*/
int foo(int x, int y) {
  int z = 2 * x + y;
  assert z ≠ 0;
  return x / z;
}
```

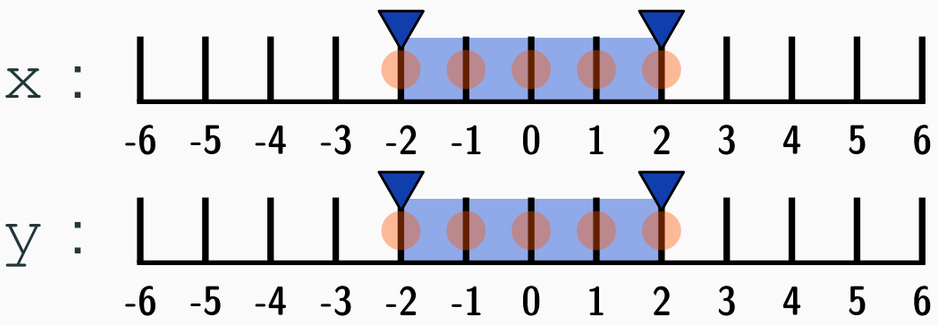
Représentation Abstraite



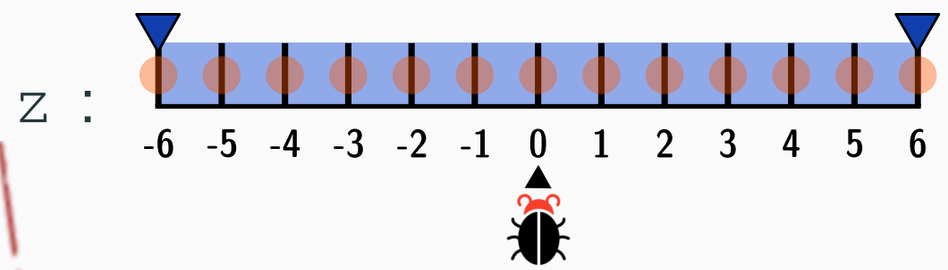


Exemple: Intervalles

```
/*————————————————————*/
/* Pre: x ∈ [−2;2]    */
/*      y ∈ [−2;2]    */
/*————————————————————*/
int foo(int x, int y) {
  int z = 2 * x + y;
  assert z ≠ 0;
  return x / z;
}
```



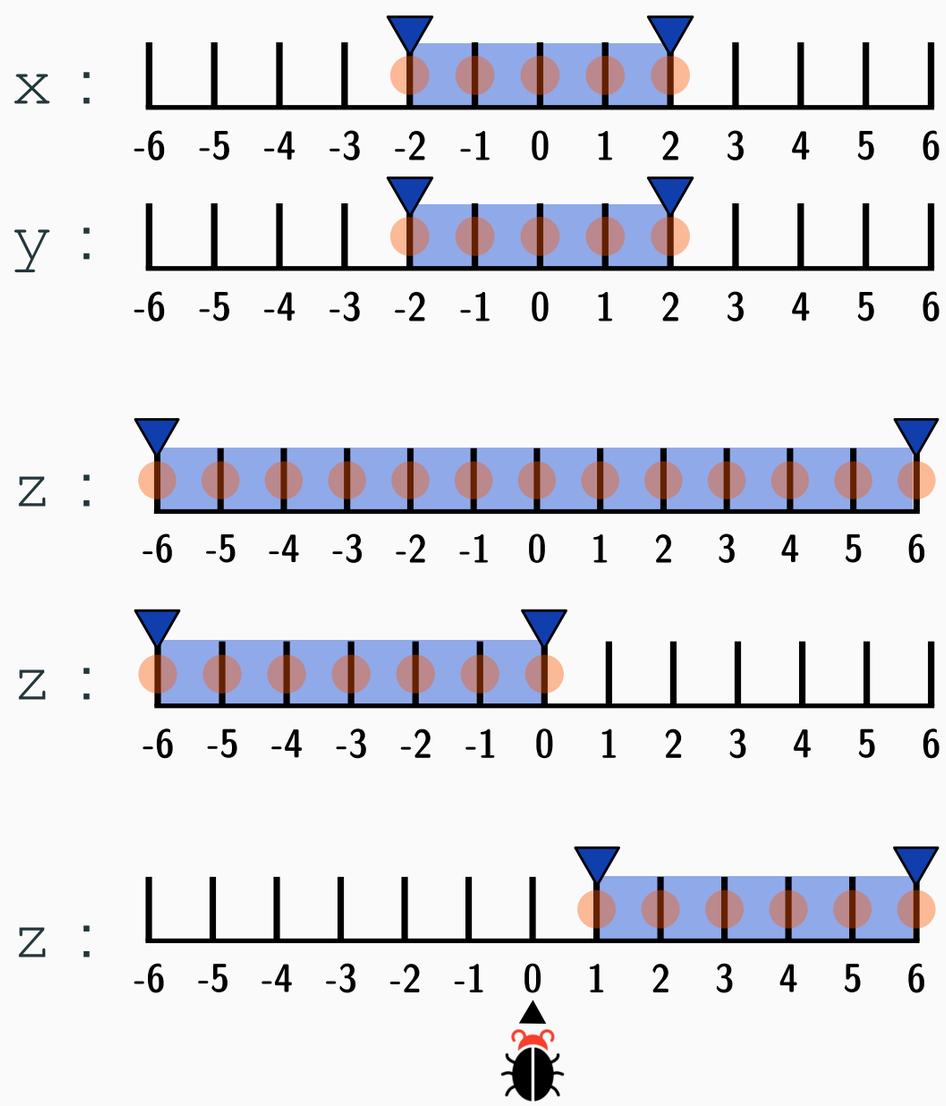
Représentation Abstraite



Exemple: Prouver l'Absence de Division par 0

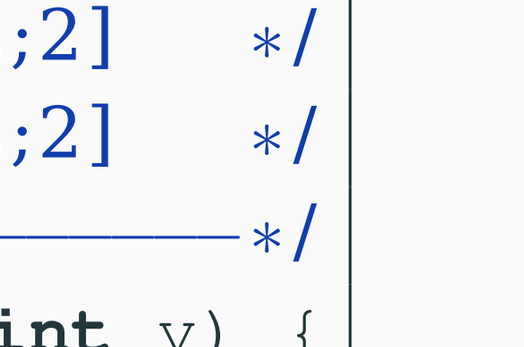
```
/*-------------------------------*/
/* Pre: x ∈ [-2;2]               */
/*      y ∈ [-2;2]               */
/*-------------------------------*/
int foo(int x, int y) {
  int z = 2 * x + y;
  if (z ≤ 0)
    return -1;
  else
    assert z ≠ 0;
    return x / z;
}
```

Intervalles

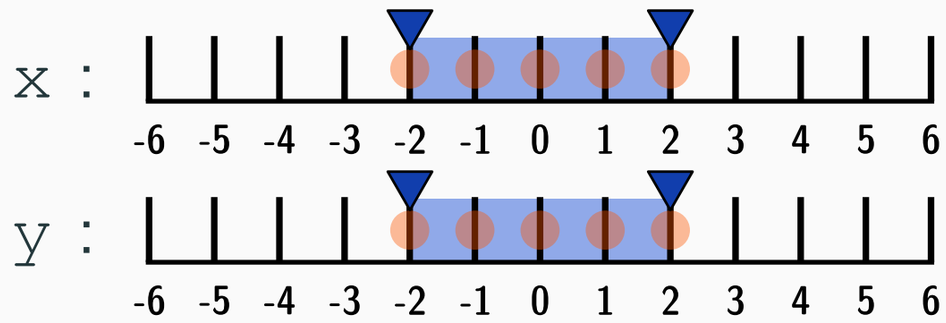


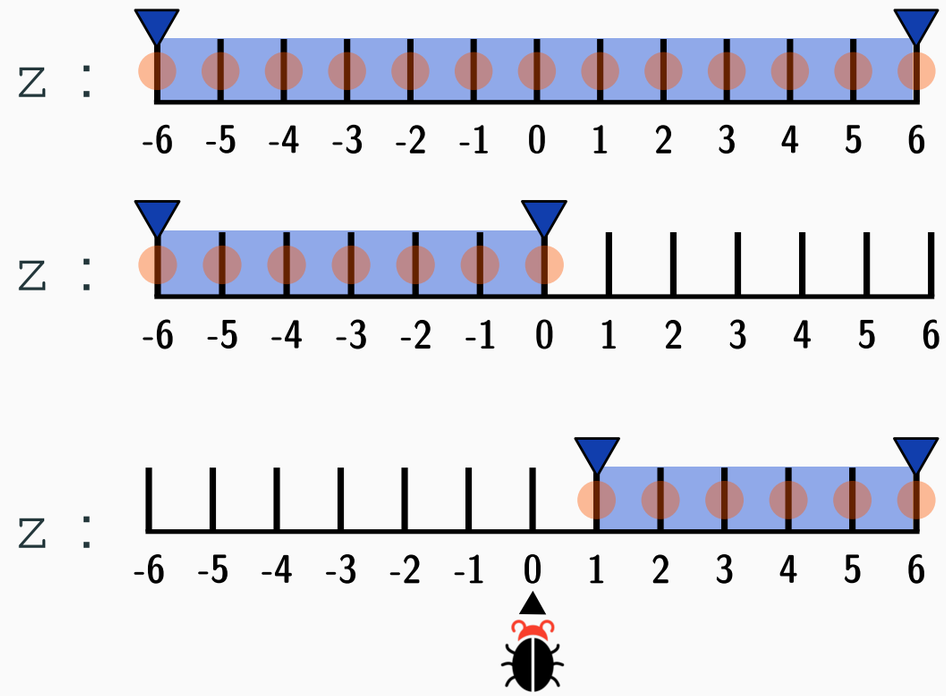
Exemple: Prouver l'Absence de Division par 0

```
/*-------------------------------*/
/* Pre: x ∈ [-2;2]               */
/*      y ∈ [-2;2]               */
/*-------------------------------*/
int foo(int x, int y) {
  int z = 2 * x + y;
  if (z ≤ 0)
    return -1;
  else
    assert z ≠ 0;
    return x / z;
}
```



Intervalles

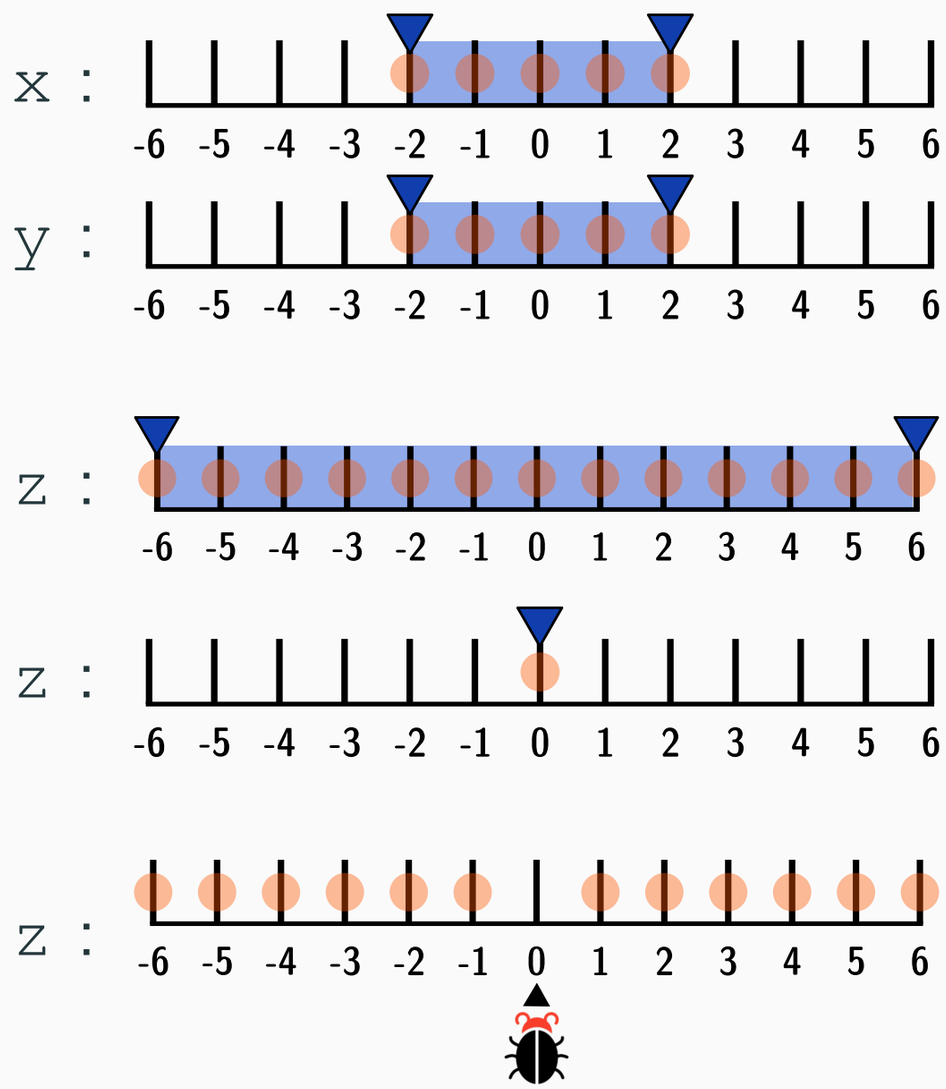




Exemple: Fausse Alarme

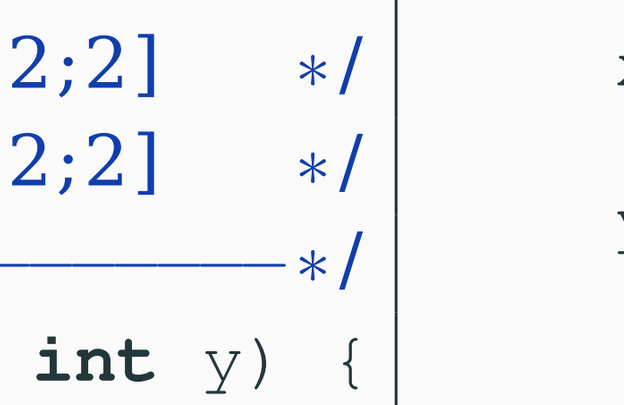
```
/*————————————————————*/
/* Pre: x ∈ [−2;2]    */
/*      y ∈ [−2;2]    */
/*————————————————————*/
int foo(int x, int y) {
  int z = 2 * x + y;
  if (z == 0)
    return −1;
  else
    assert z ≠ 0;
    return x / z;
}
```

Intervalles

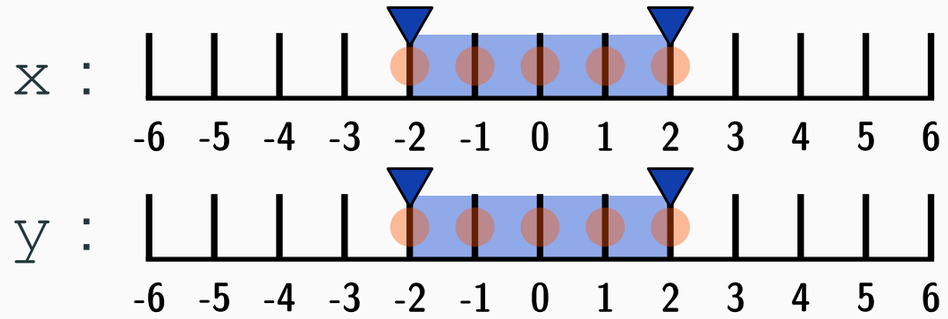


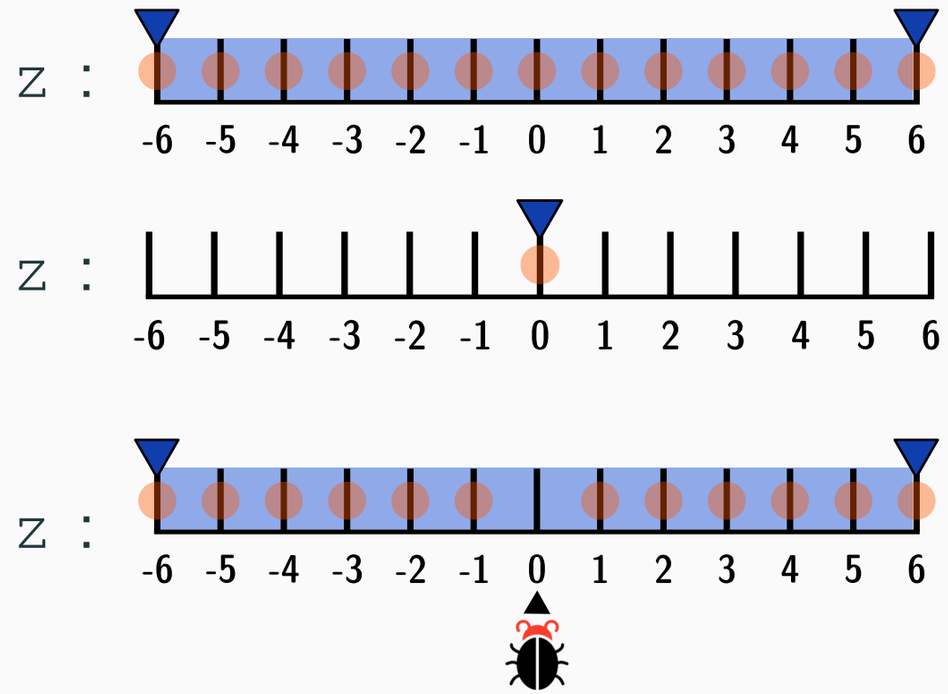
Exemple: Fausse Alarme

```
/*-------------------------------*/
/* Pre: x ∈ [-2;2]               */
/*      y ∈ [-2;2]               */
/*-------------------------------*/
int foo(int x, int y) {
  int z = 2 * x + y;
  if (z == 0)
    return -1;
  else
    assert z ≠ 0;
    return x / z;
}
```



Intervalles





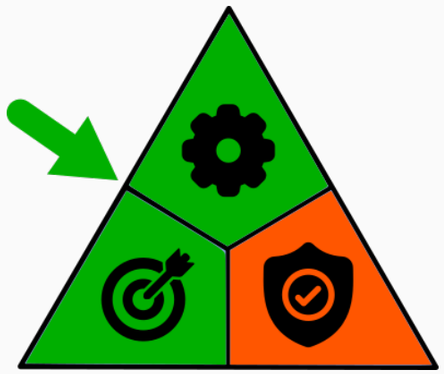
Conclusion

Approche correcte par construction

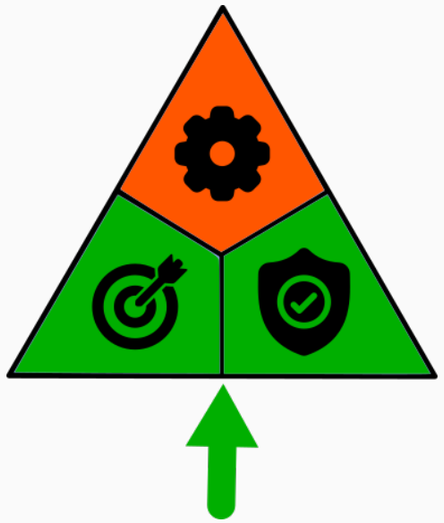
Raisonner sur des modèles pour construire des systèmes plus sûrs.

Vérification de Programme

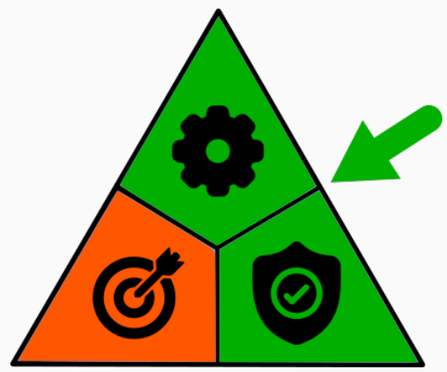
Test Automatique
(Fuzzing et Ex. Sym.)



Vérification Symbolique
et Preuve de Théorème



Interprétation Abstraite



Méthodes Formelles: De quoi on n'a pas parlé

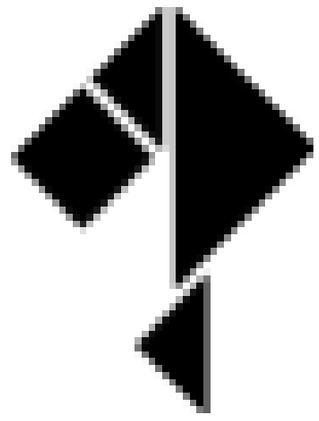
- Démontration automatique de théorèmes
- Des systèmes de types
- De la programmation par contraintes
- De la compilation certifiée...

Quelques outils

Fuzzing / Exécution Symbolique: AFL, KLEE

Preuve de Théorèmes:

Why3




```
method DutchFlag(a: array<Color>)
  requires a ≠ null modifies a
» ensures ∀ i,j · 0 ≤ i < j < a.Length ⟹ Ordered(a[i], a[j])
  ensures multiset(a[..]) == old(multiset(a[..]))
{
  var r, w, b := 0, 0, a.Length;
» while w ≠ b
    invariant 0 ≤ r ≤ w ≤ b ≤ a.Length;
    invariant ∀ i · 0 ≤ i < r⟹a[i] == Red
    invariant multiset(a[..]) == old(multiset(a[..]))
  {   match a[w]
      case Red ⇒
        a[r], a[w] := a[w], a[r];
        r, w := r + 1, w + 1;
      case White ⇒
        w := w + 1;
      case Blue ⇒
        b := b - 1;
```

Interprétation Abstraite

frama C

Software Analyzers

Astrée (AbsInt)

AbsInt

VERASCO

Questions

Thank you for your attention.

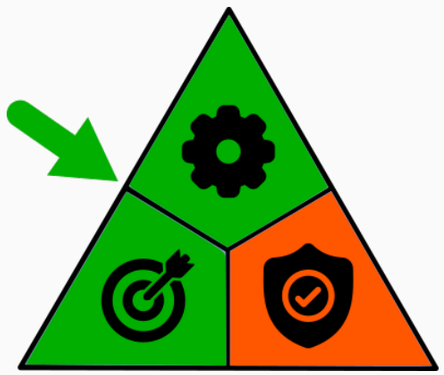
Conclusion

Approche correcte par construction

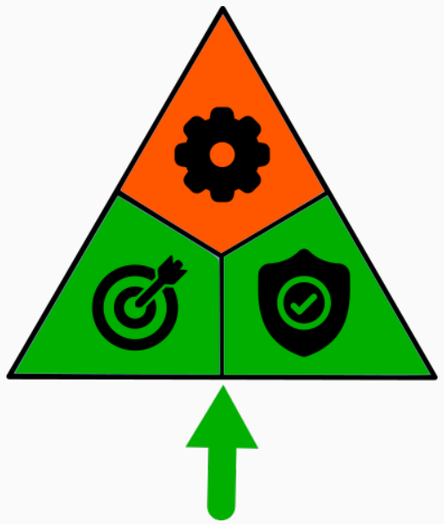
Raisonner sur des modèles pour construire des systèmes plus sûrs.

Vérification de Programme

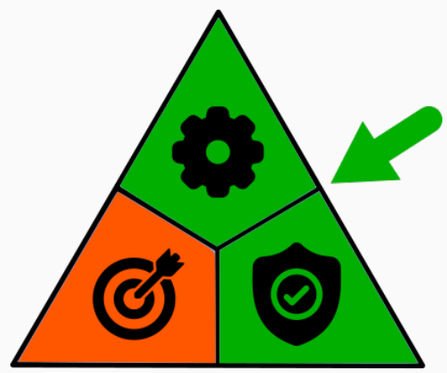
Test Automatique
(Fuzzing et Ex. Sym.)



Vérification Symbolique
et Preuve de Théorème



Interprétation Abstraite



Remerciements

- Julien G. pour son aide pour les slides
- Olivier N. et Dara L. pour leur implication dans le projet
- Olivier D. pour son aide sur quelques schémas
- À tous ceux qui ont donné des retours (David B., Valentin P., Julien S., ...)