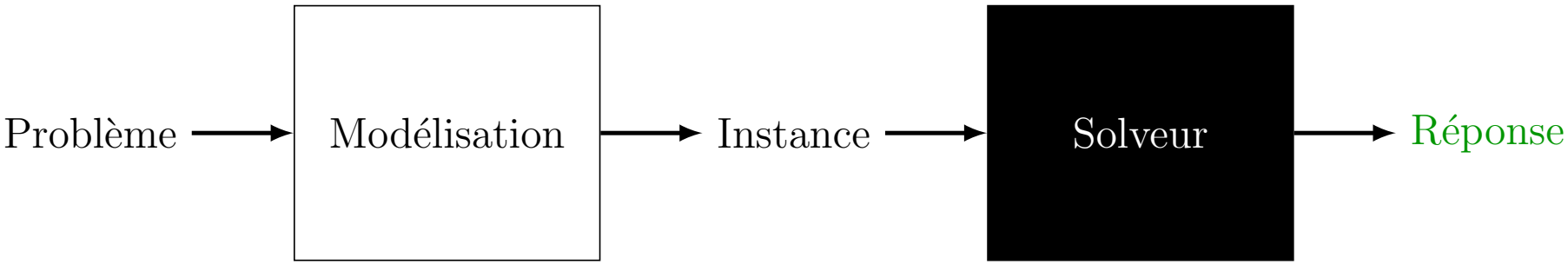
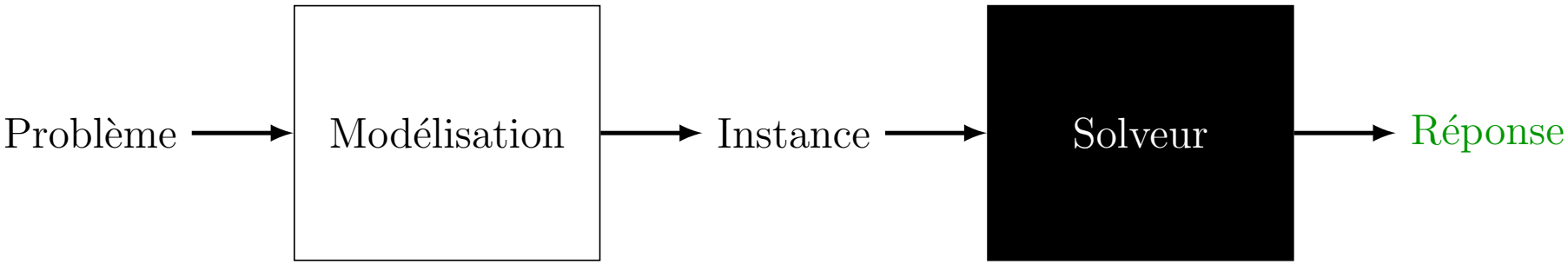
MCSP3 & XCSP3

Cyril Terrioux
cyril.terrioux@univ-amu.fr

Contexte

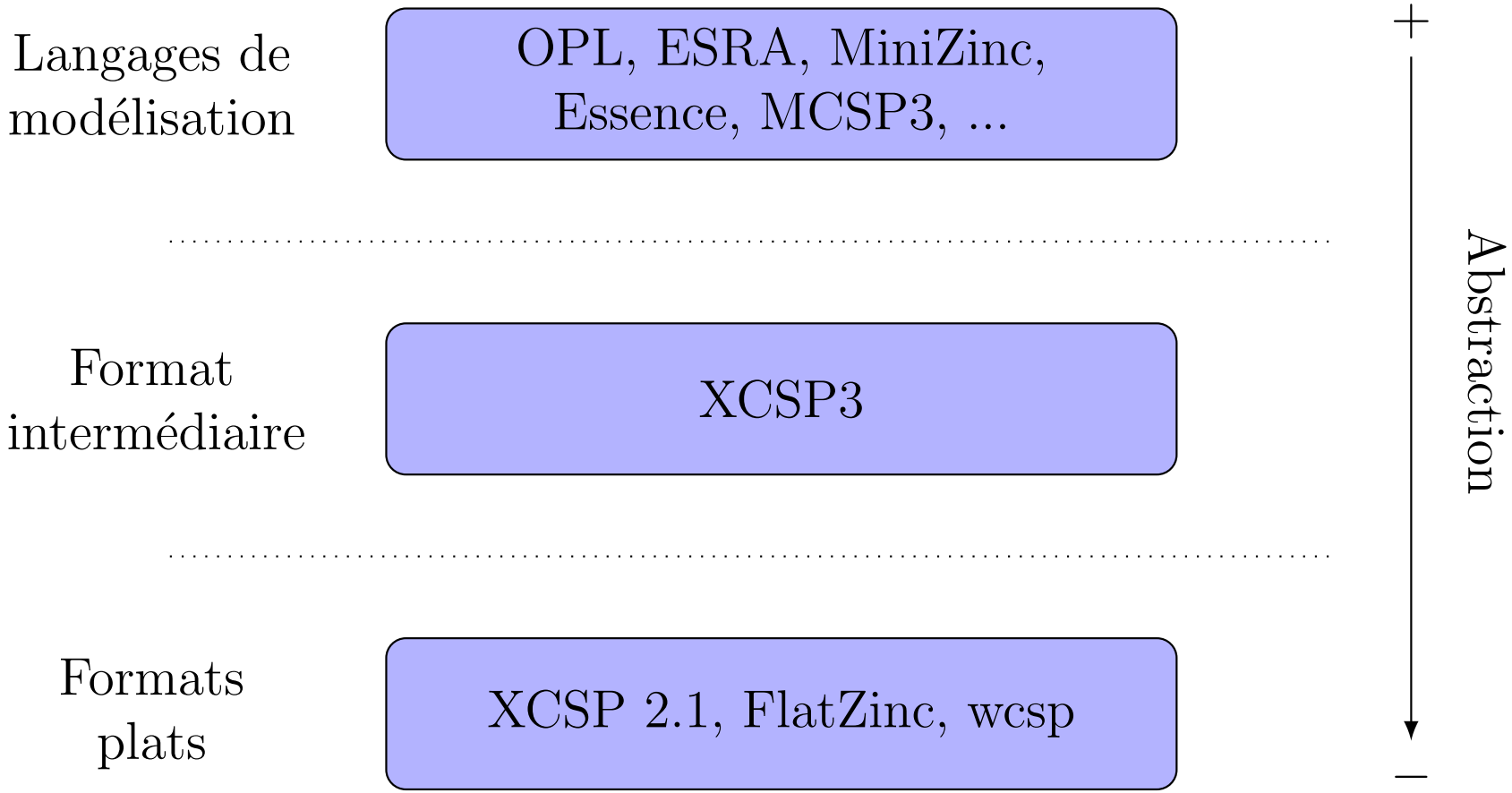


Contexte

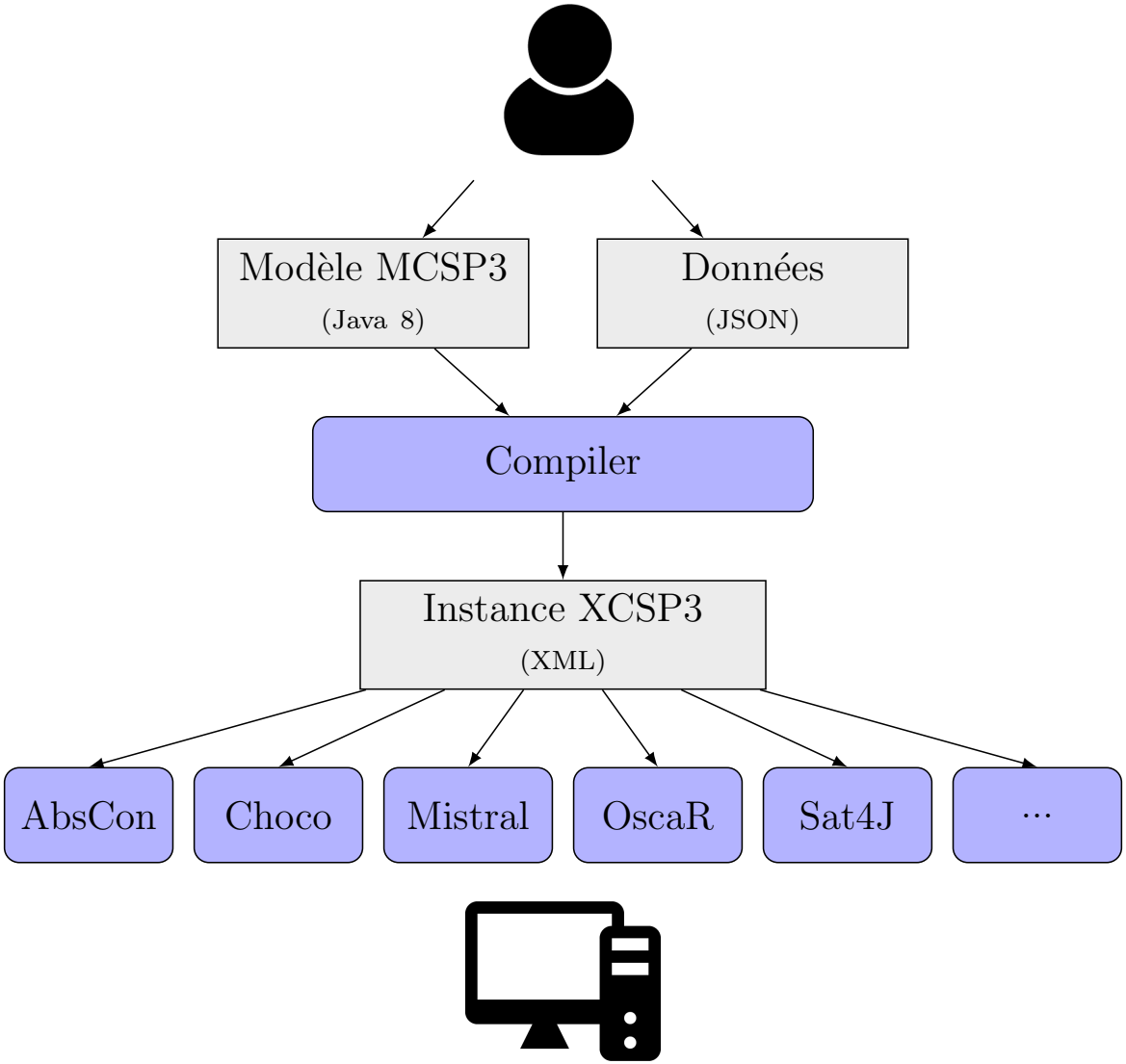


Quel langage / format pour représenter les instances ?

Contexte


Langages de modélisation
OPL, ESRA, MiniZinc, Essence, MCSP3, ...
Format intermédiaire
XCSP3
Formats plats
XCSP 2.1, FlatZinc, wcsp
+
−
Abstraction

Contexte


Modèle MCSP3
(Java 8)
Données
(JSON)
Compiler
Instance XCSP3
(XML)
AbsCon
Choco
Mistral
OscaR
Sat4J
...

Pourquoi MCSP3 et XCSP3 ?

MCSP3 :

- Basé sur Java
- Couplé à JSON $\Rightarrow$ données facilement lisibles et manipulables

Pourquoi MCSP3 et XCSP3 ?

MCSP3 :

- Basé sur Java
- Couplé à JSON $\Rightarrow$ données facilement lisibles et manipulables

XCSP3 :

- Basé sur XML
- Permet de structurer le document
- Préserve la structure de l'instance
- Facilement lisible
- Format très riche (CSP, COP, CSP pondérés)

XCSP3-core

Version supportée par les solveurs

XCSP3-core

Version supportée par les solveurs

Variables booléennes ou entières

XCSP3-core

Version supportée par les solveurs

Variables booléennes ou entières

Contraintes :

- `intension`
- `extension`
- `regular`
- `mdd`
- `allDifferent`
- `allEqual`
- `ordered`
- `lex`
- `sum`
- `count`
- `nValues`
- `cardinality`
- `maximum`
- `minimum`
- `element`
- `channel`
- `noOverlap`
- `cumulative`
- `instantiation`
- `circuit`
- `slide`

Problème des n-dames

Objectif : Placer n dames sur un échiquier $n \times n$

Problème des n-dames

Objectif : Placer n dames sur un échiquier $n \times n$

Modélisation :

- $X = \{x_1, \ldots, x_n\}$
- $D = \{d_{x_1}, \ldots, d_{x_n}\}$ avec $\forall i,\ d_{x_i} = \{1, \ldots, n\}$
- $C = \{c_{ij} | i < j\}$ avec :
  - $S(c_{ij}) = \{x_i, x_j\}$
  - $R(c_{ij}) = \{(v_i, v_j) \in d_{x_i} \times d_{x_j} | v_i \neq v_j \wedge |v_i - v_j| \neq |i - j|\}$

Problème des *n*-dames (MCSP3)

```
class Queens implements ProblemAPI {
  int n; // number of queens

  public void model() {
    Var[] q = array("q", size(n), dom(range(n)),
      "q[i] is the column where is put the ith queen (at row i)");

    forall(range(n).range(n), (i, j) -> {
      if (i < j)
        intension(and(ne(q[i], q[j]), ne(dist(q[i], q[j]), dist(i, j))));
    });
  }
}
```

Problème des *n*-dames (MCSP3)

```
class Queens implements ProblemAPI {
  int n; // number of queens

  public void model() {
    Var[] q = array("q", size(n), dom(range(n)),
      "q[i] is the column where is put the ith queen (at row i)");

    forall(range(n).range(n), (i, j) -> {
      if (i < j)
        intension(and(ne(q[i], q[j]), ne(dist(q[i], q[j]), dist(i, j))));
    });
  }
}
```

```
$ java org.xcsp.modeler.Compiler Queens -data=4
```

Problème des *n*-dames (XCSP3)

```
<instance format="XCSP3" type="CSP">
  <variables>
    <array id="q" note="q[i] is the column where is put the ith queen (at row i)" size="[4]"
          >
      0..3
    </array>
  </variables>
  <constraints>
    <group>
      <intension> and(ne(%0,%1),ne(dist(%0,%1),dist(%2,%3))) </intension>
      <args> q[0] q[1] 0 1 </args>
      <args> q[0] q[2] 0 2 </args>
      <args> q[0] q[3] 0 3 </args>
      <args> q[1] q[2] 1 2 </args>
      <args> q[1] q[3] 1 3 </args>
      <args> q[2] q[3] 2 3 </args>
    </group>
  </constraints>
</instance>
```

Problème des *n*-dames (MCSP3)

```
$ java org.xcsp.modeler.Compiler Queens -data=queens.json
```

```
{
  "n":4
}
```

Problème des n-dames

Deuxième Modélisation :

- $X = \{x_1, \ldots, x_n\}$
- $D = \{d_{x_1}, \ldots, d_{x_n}\}$ avec $\forall i,\ d_{x_i} = \{1, \ldots, n\}$
- $C = \{\mathit{All\text{-}different}(X)\} \cup \{c_{ij} | i < j\}$ avec :
  - $S(c_{ij}) = \{x_i, x_j\}$
  - $R(c_{ij}) = \{(v_i, v_j) \in d_{x_i} \times d_{x_j} || v_i - v_j| \neq |i - j|\}$

Problème des *n*-dames (MCSP3)

```
class Queens implements ProblemAPI {
  int n; // number of queens

  public void model() {
    Var[] q = array("q", size(n), dom(range(n)));

    if (isModel("m1")) {
      forall(range(n).range(n), (i, j) -> {
        if (i < j)
          intension(and(ne(q[i], q[j]), ne(dist(q[i], q[j]), dist(i, j))));
      });
    }
    if (isModel("m2")) {
      allDifferent(q);
      forall(range(n).range(n), (i, j) -> {
        if (i < j)
          notEqual(dist(q[i], q[j]), dist(i, j));
      });
    }
  }
}
```

Problème des *n*-dames (MCSP3)

```
class Queens implements ProblemAPI {
  int n; // number of queens

  public void model() {
    Var[] q = array("q", size(n), dom(range(n)));

    if (isModel("m1")) {
      forall(range(n).range(n), (i, j) -> {
        if (i < j)
          intension(and(ne(q[i], q[j]), ne(dist(q[i], q[j]), dist(i, j))));
      });
    }
    if (isModel("m2")) {
      allDifferent(q);
      forall(range(n).range(n), (i, j) -> {
        if (i < j)
          notEqual(dist(q[i], q[j]), dist(i, j));
      });
    }
  }
}
```

```
$ java org.xcsp.modeler.Compiler Queens -data=4 -model=m2
```

Problème des n-dames (XCSP3)

```
<instance format="XCSP3" type="CSP">
  <variables>
    <array id="q" size="[4]"> 0..3 </array>
  </variables>
  <constraints>
    <allDifferent> q[] </allDifferent>
    <group>
      <intension> ne(dist(%0,%1),dist(%2,%3)) </intension>
      <args> q[0] q[1] 0 1 </args>
      <args> q[0] q[2] 0 2 </args>
      <args> q[0] q[3] 0 3 </args>
      <args> q[1] q[2] 1 2 </args>
      <args> q[1] q[3] 1 3 </args>
      <args> q[2] q[3] 2 3 </args>
    </group>
  </constraints>
</instance>
```