

# A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

GILLES GEERAERTS

`gigeerae@ulb.ac.be`.

Université Libre de Bruxelles – Département  
d'informatique

# Plan of the talk

A Comparison of Various Backward Analyzers for  
Parametrized Concurrent Systems

# Plan of the talk

A Comparison of Various Backward Analyzers for  
Parametrized Concurrent Systems

- What is a parametrized concurrent system ?

# Plan of the talk

## A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

- What is a parametrized concurrent system ?
  - Need for verification, how to formalize...

# Plan of the talk

## A Comparison of Various **Backward Analyzers** for Parametrized Concurrent Systems

- What is a **parametrized concurrent system** ?
  - Need for verification, how to formalize...
- How do we **verify** ?

# Plan of the talk

## A Comparison of Various **Backward Analyzers** for Parametrized Concurrent Systems

- What is a **parametrized concurrent system** ?
  - Need for verification, how to formalize...
- How do we **verify** ?
  - Forward and backward approach, decidability results...

# Plan of the talk

## A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

- What is a parametrized concurrent system ?
  - Need for verification, how to formalize...
- How do we verify ?
  - Forward and backward approach, decidability results...
- What can we compare ?

# Plan of the talk

## A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

- What is a parametrized concurrent system ?
  - Need for verification, how to formalize...
- How do we verify ?
  - Forward and backward approach, decidability results...
- What can we compare ?
  - Performances with different datastructures...



# Motivation – Concurrent systems

- Concurrent system = system with **many processes** interacting and communicating...

# Motivation – Concurrent systems

- Concurrent system = system with **many processes** interacting and communicating...
- ... they can be found everywhere !

# Motivation – Concurrent systems

- Concurrent system = system with **many processes** interacting and communicating. . .
- . . . they can be found everywhere !
  - e.g.: Multi-threaded Java programs for web-based applications.

# Motivation – Concurrent systems

- Concurrent system = system with **many processes** interacting and communicating. . .
- . . . they can be found everywhere !
  - e.g.: Multi-threaded Java programs for web-based applications.
- They often get involved in **safety-critical** environments.

# Motivation – Concurrent systems

- Concurrent system = system with **many processes** interacting and communicating...
- ... they can be found everywhere !
  - e.g.: Multi-threaded Java programs for web-based applications.
- They often get involved in **safety-critical** environments.
  - e.g.: Online secured billing.

# Motivation – Concurrent systems

- Concurrent system = system with **many processes** interacting and communicating...
- ... they can be found everywhere !
  - e.g.: Multi-threaded Java programs for web-based applications.
- They often get involved in **safety-critical** environments.
  - e.g.: Online secured billing.

**We need well-suited verification procedures !**

# Motivation – Parametrized verification

- Most of the time, the number of process is not fixed (it is a **parameter**).

# Motivation – Parametrized verification

- Most of the time, the number of process is not fixed (it is a **parameter**).
- e.g.: How many clients are going to connect to a given web-server ?



# Motivation – Parametrized verification

- Most of the time, the number of process is not fixed (it is a **parameter**).
  - e.g.: How many clients are going to connect to a given web-server ?
- **Classical approach**: try for one process, two processes, three processes...

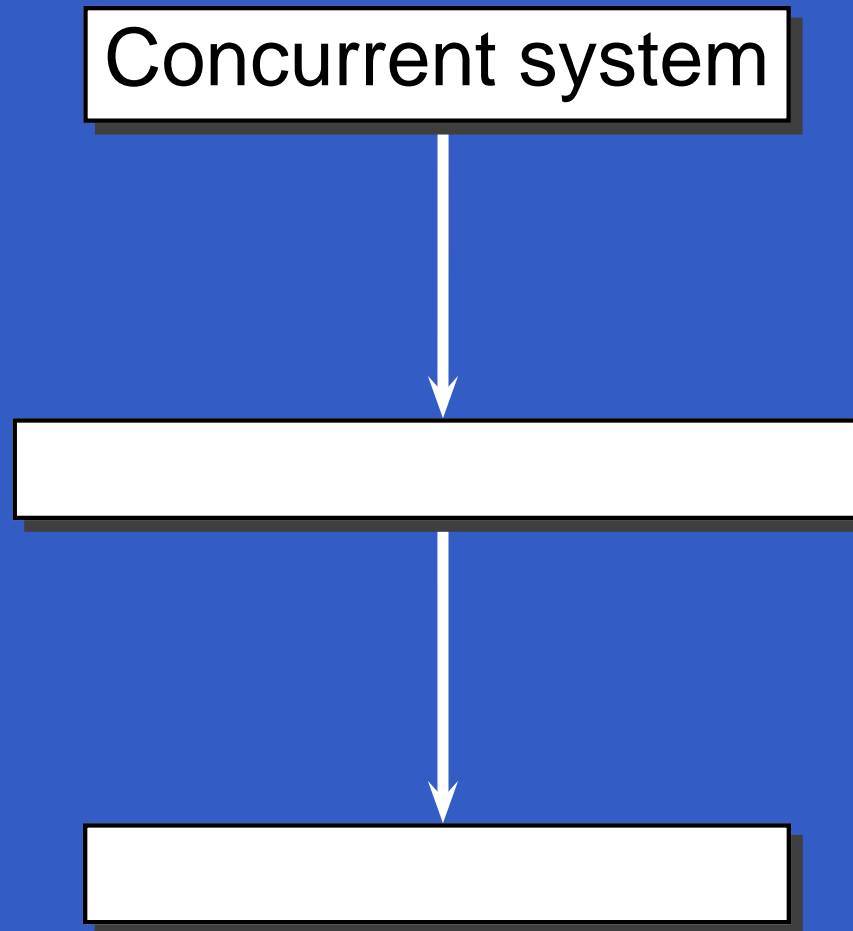
# Motivation – Parametrized verification

- Most of the time, the number of process is not fixed (it is a **parameter**).
  - e.g.: How many clients are going to connect to a given web-server ?
- **Classical approach**: try for one process, two processes, three processes...
- ... and hope the property holds for other values of the parameter !

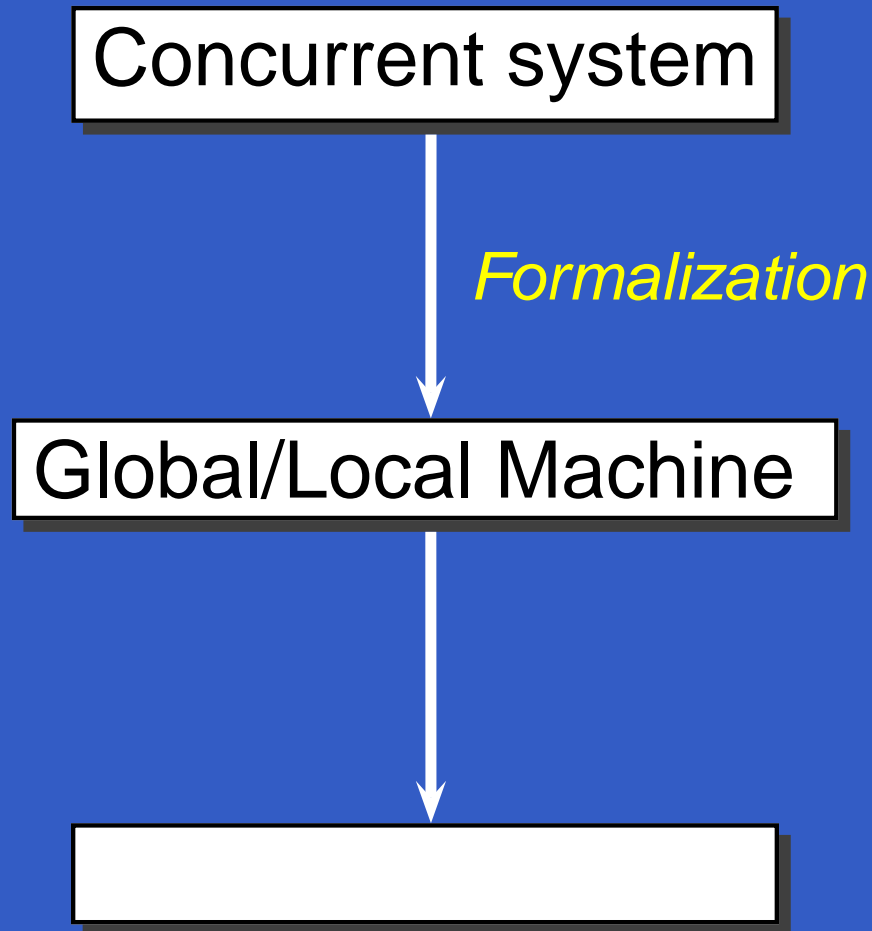
# Motivation – Parametrized verification

- Most of the time, the number of process is not fixed (it is a **parameter**).
  - e.g.: How many clients are going to connect to a given web-server ?
- **Classical approach**: try for one process, two processes, three processes...
- ... and hope the property holds for other values of the parameter !
- **Parametrized approach**: Verify the property for **any value** of the parameter.

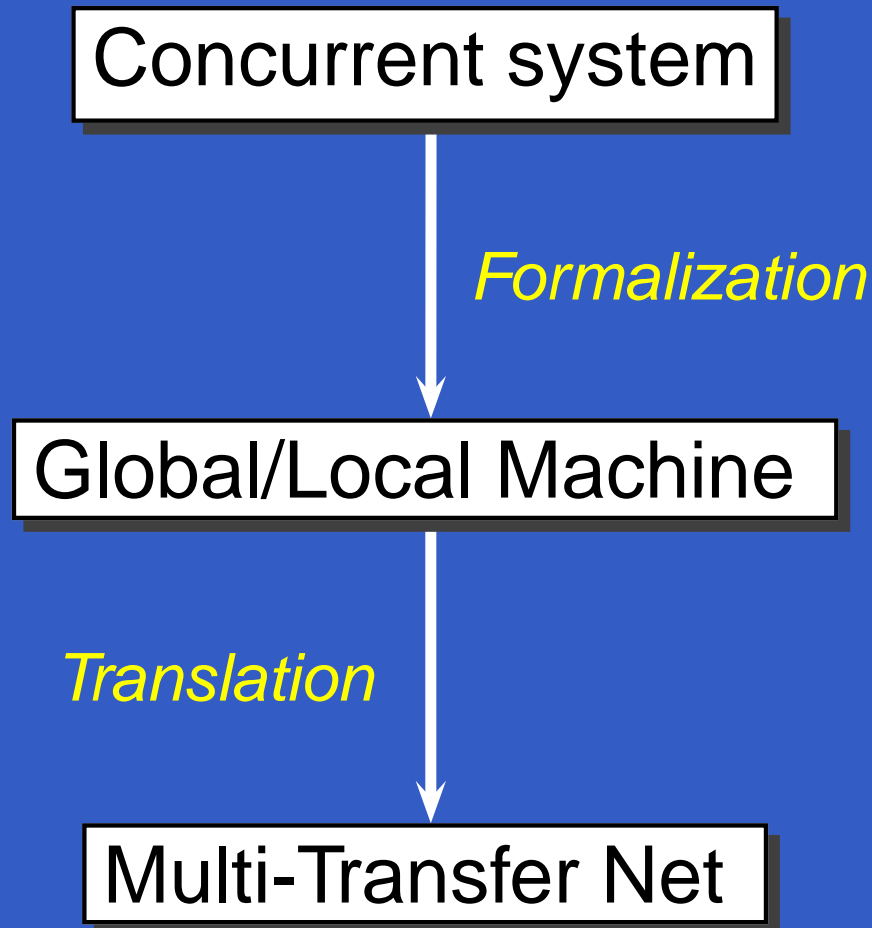
# The verification process



# The verification process



# The verification process



# Global/Local machines

- One **global machine** = collection of several **local machines** + global **boolean variables**.

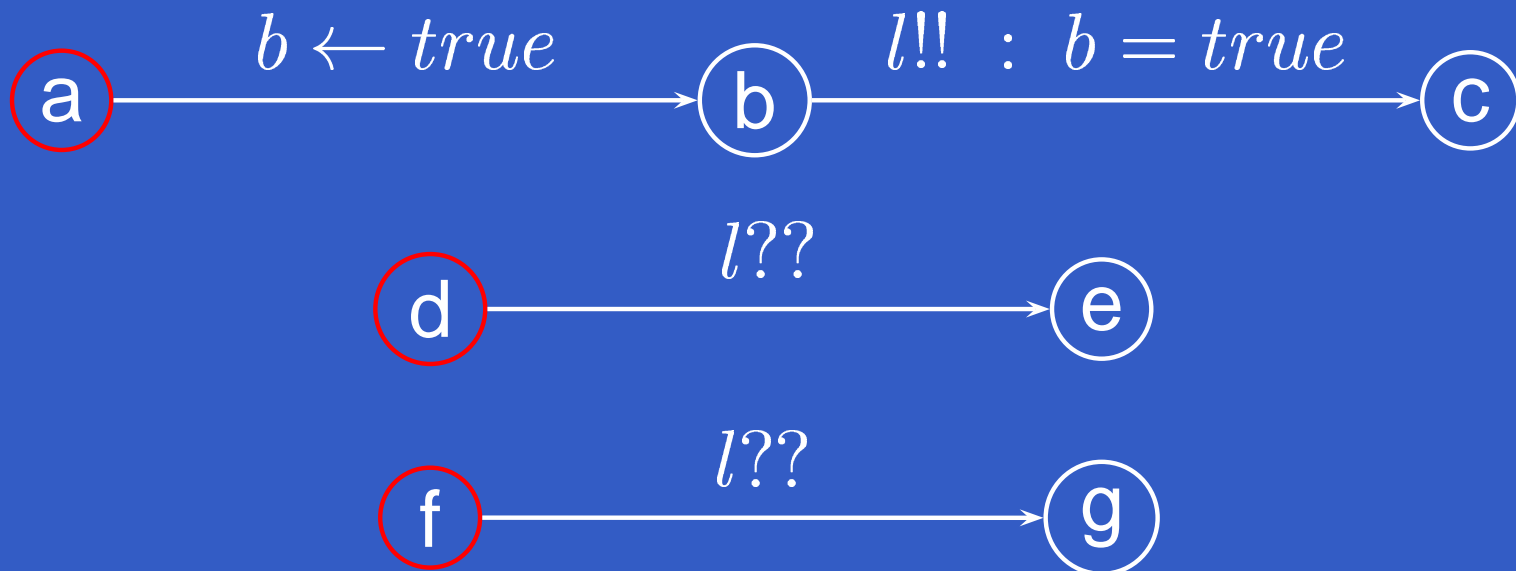
# Global/Local machines

- One **global machine** = collection of several **local machines** + global **boolean variables**.
- The local machines **synchronize** through *rendez-vous*, broadcasts and asynchronous *rendez-vous*.



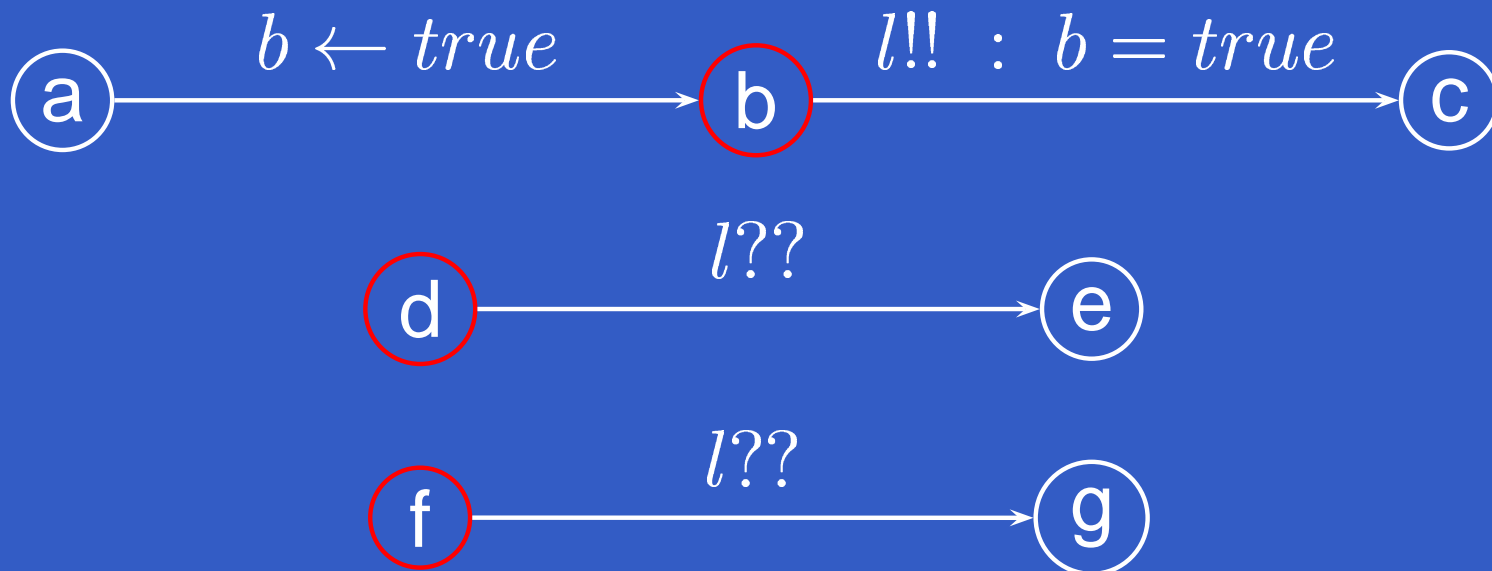
# Global/Local machines

- One **global machine** = collection of several **local machines** + global **boolean variables**.
- The local machines **synchronize** through *rendez-vous*, broadcasts and asynchronous *rendez-vous*.



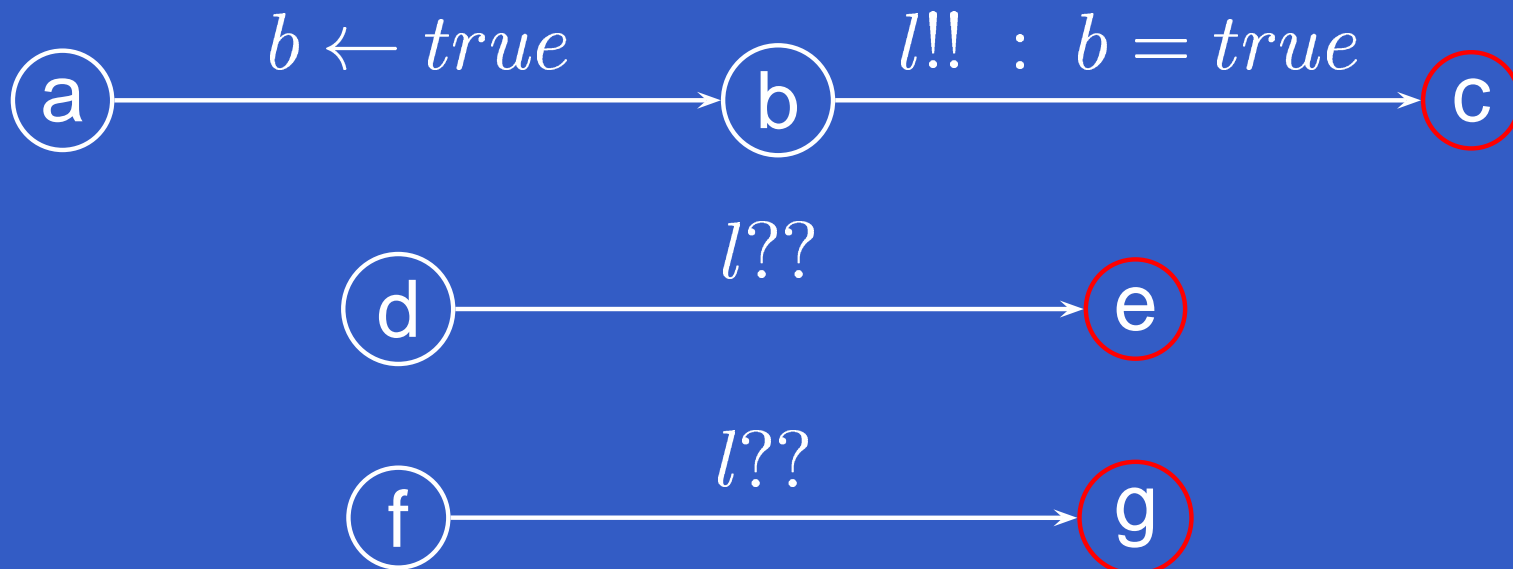
# Global/Local machines

- One **global machine** = collection of several **local machines** + global **boolean variables**.
- The local machines **synchronize** through *rendez-vous*, broadcasts and asynchronous *rendez-vous*.

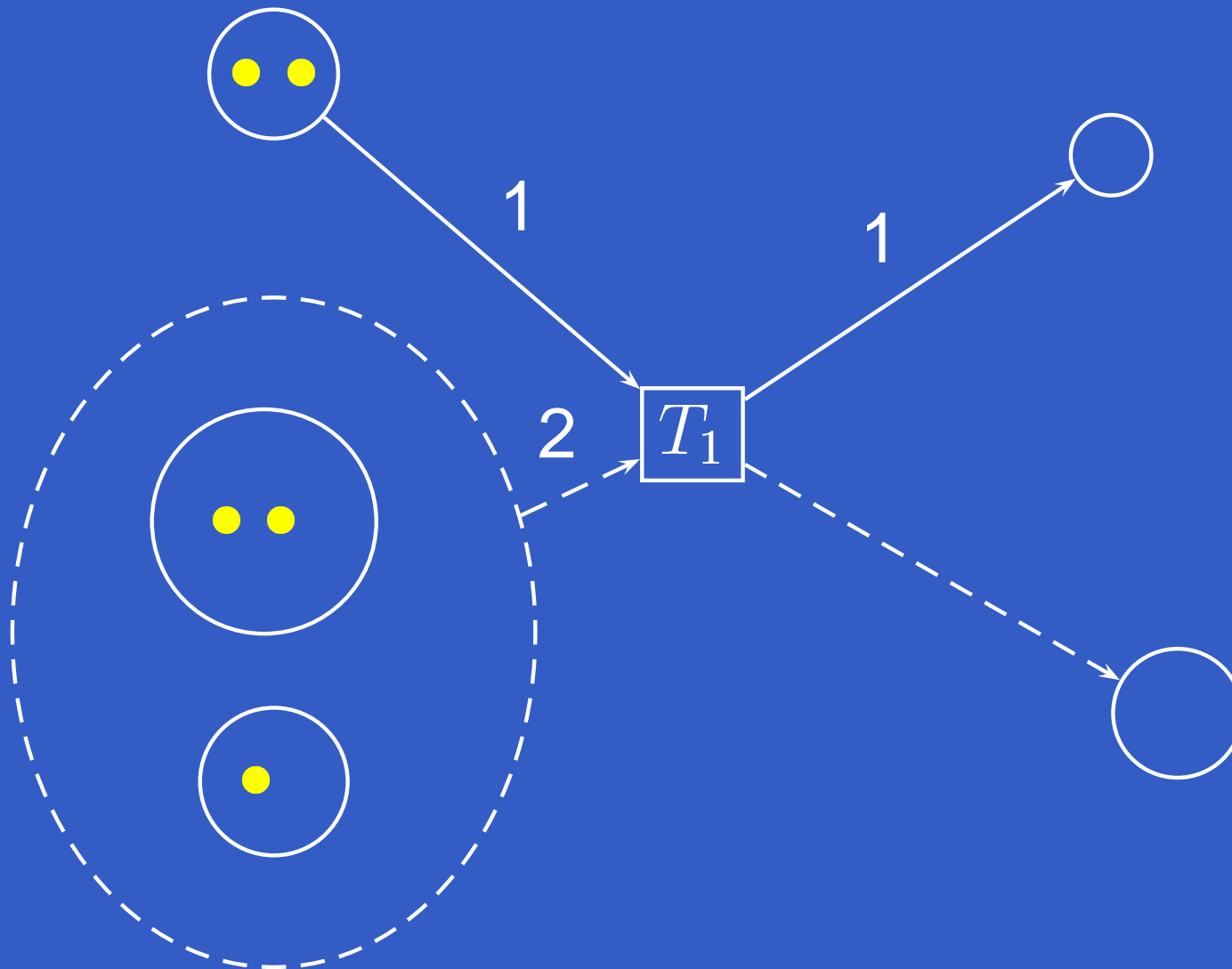


# Global/Local machines

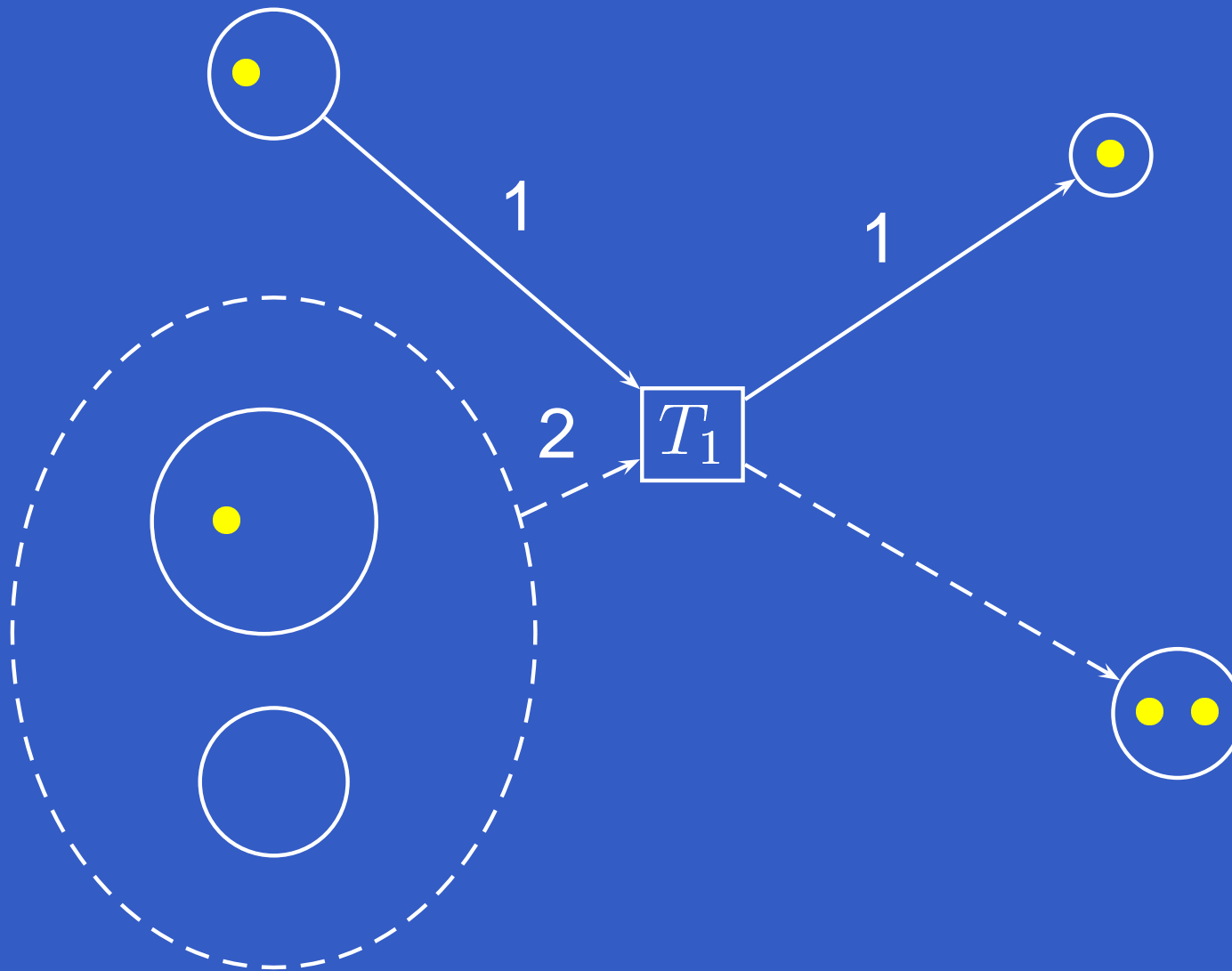
- One **global machine** = collection of several **local machines** + global **boolean variables**.
- The local machines **synchronize** through *rendez-vous*, broadcasts and asynchronous *rendez-vous*.



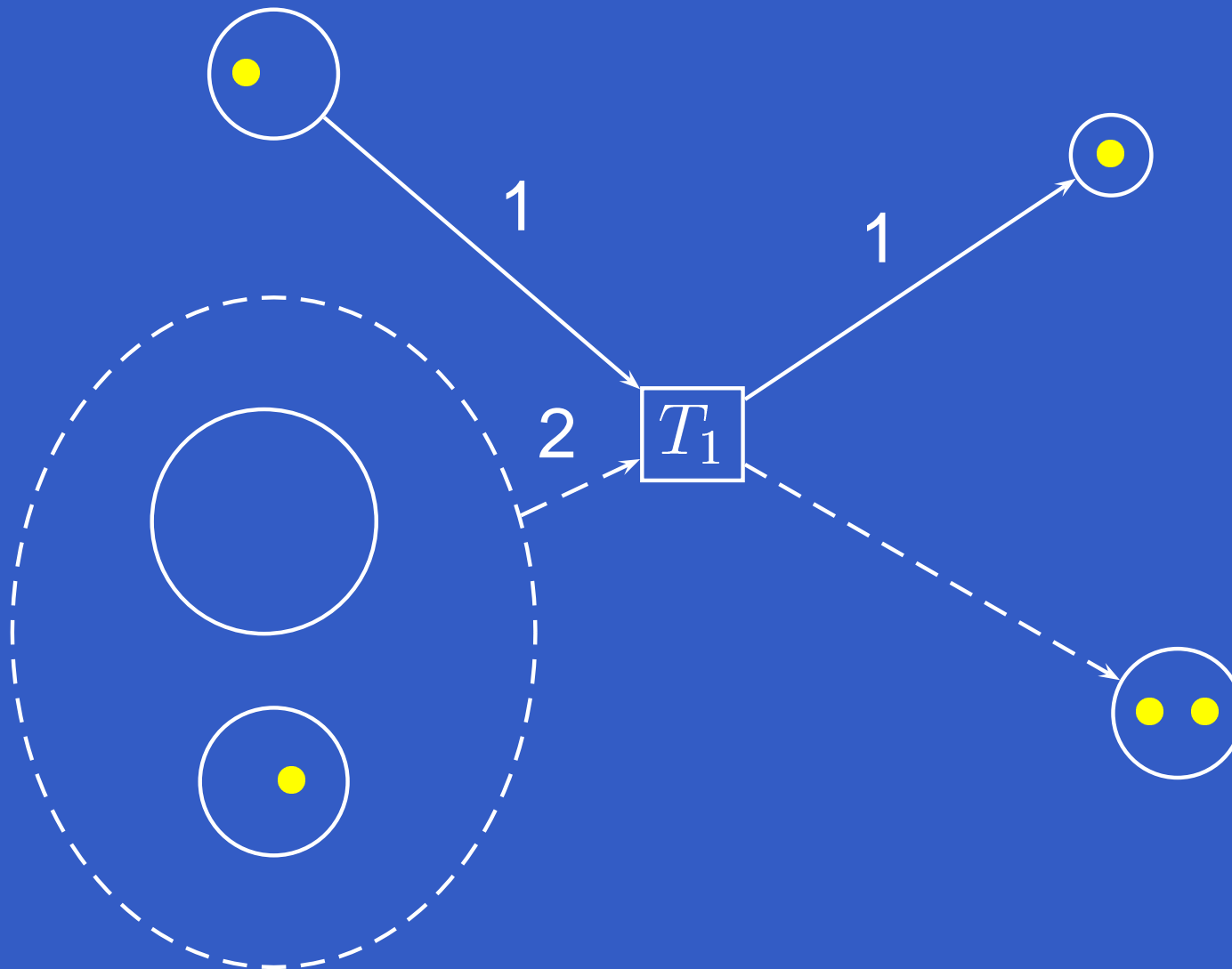
# Multi-transfer nets



# Multi-transfer nets



# Multi-transfer nets



# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?

# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:



# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:

Forward



# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:

Forward



# Forward v.s. Backward

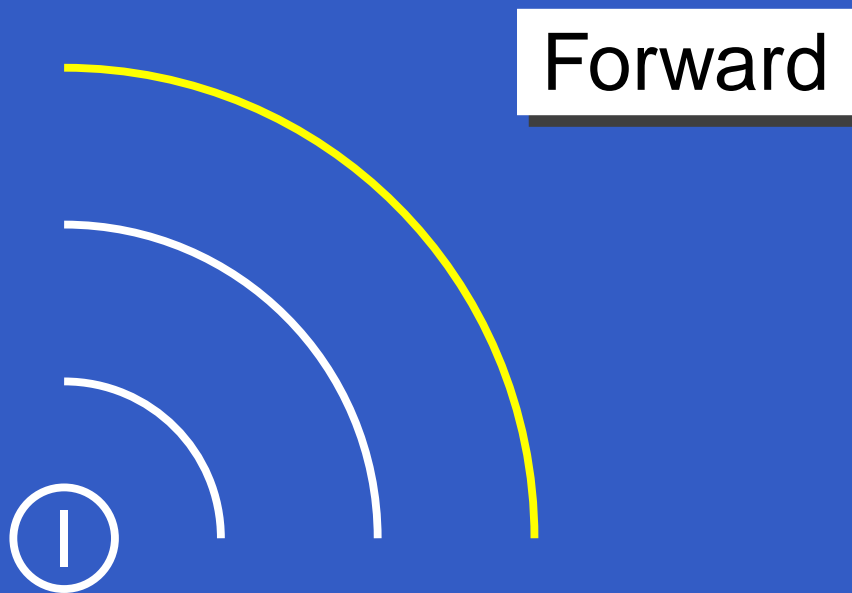
- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:

Forward



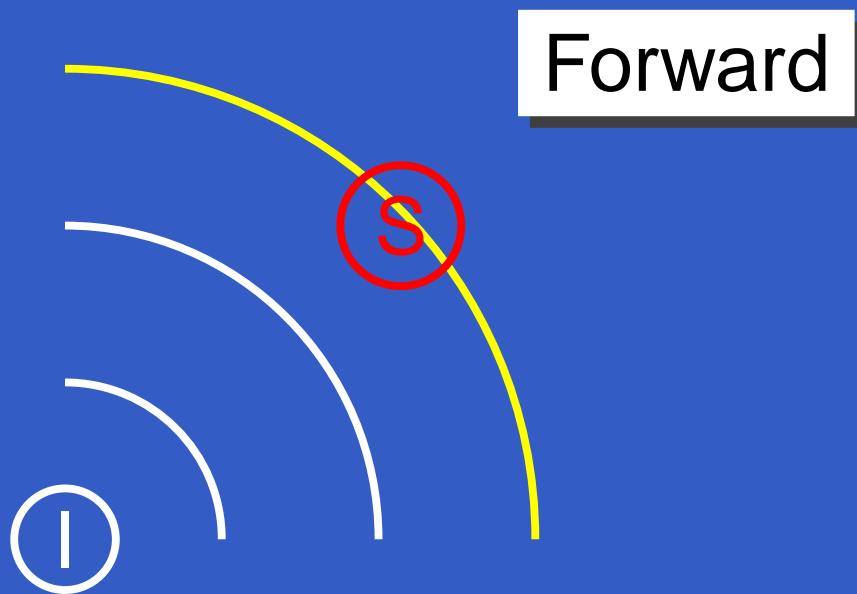
# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:



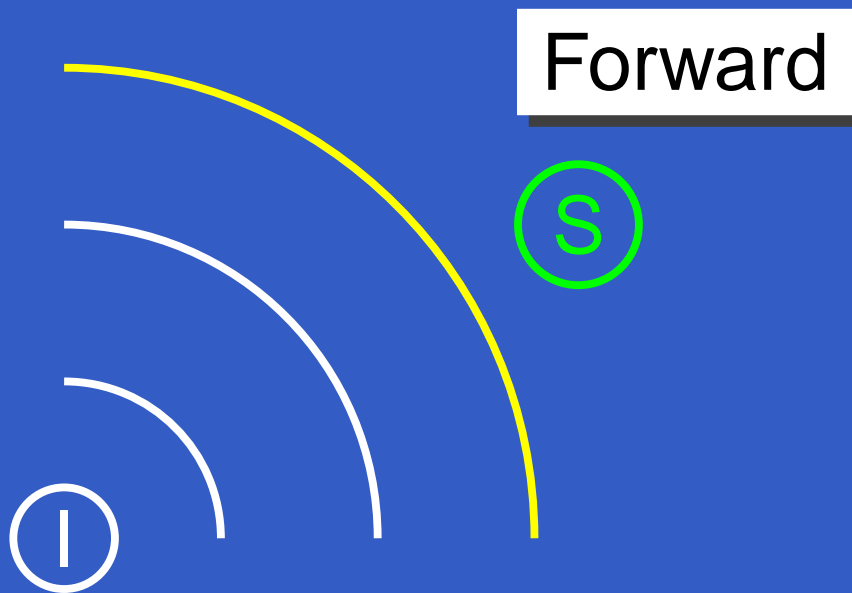
# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:



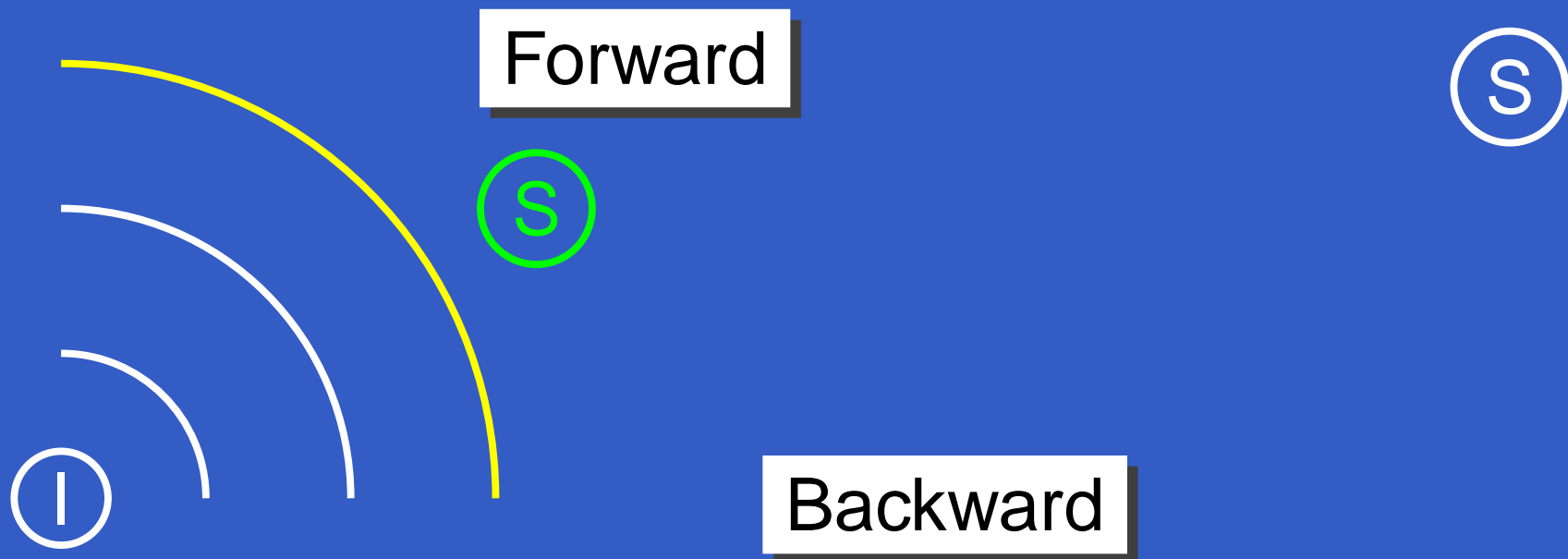
# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:



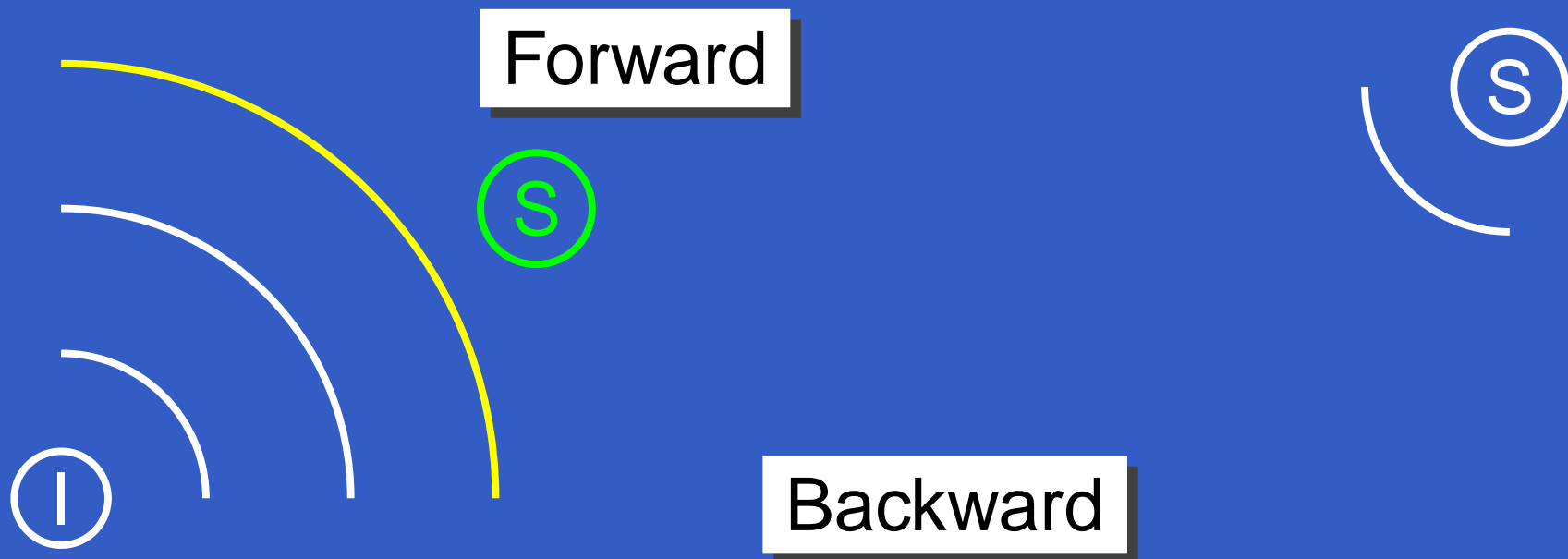
# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:



# Forward v.s. Backward

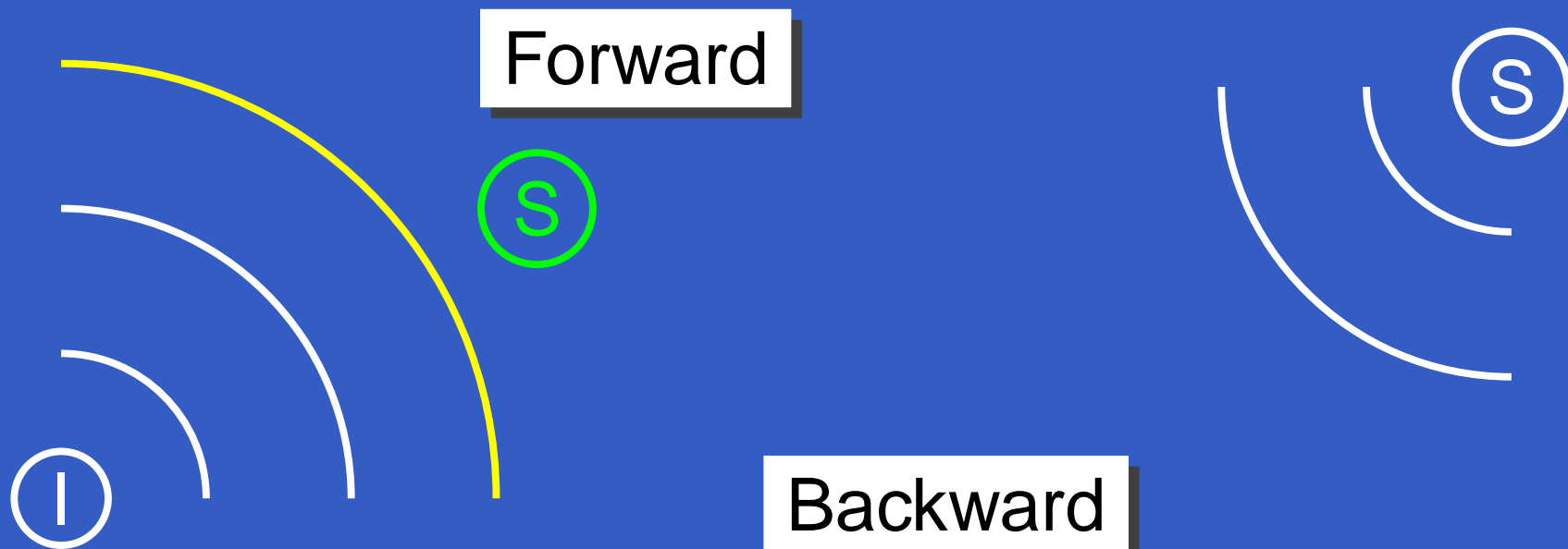
- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:





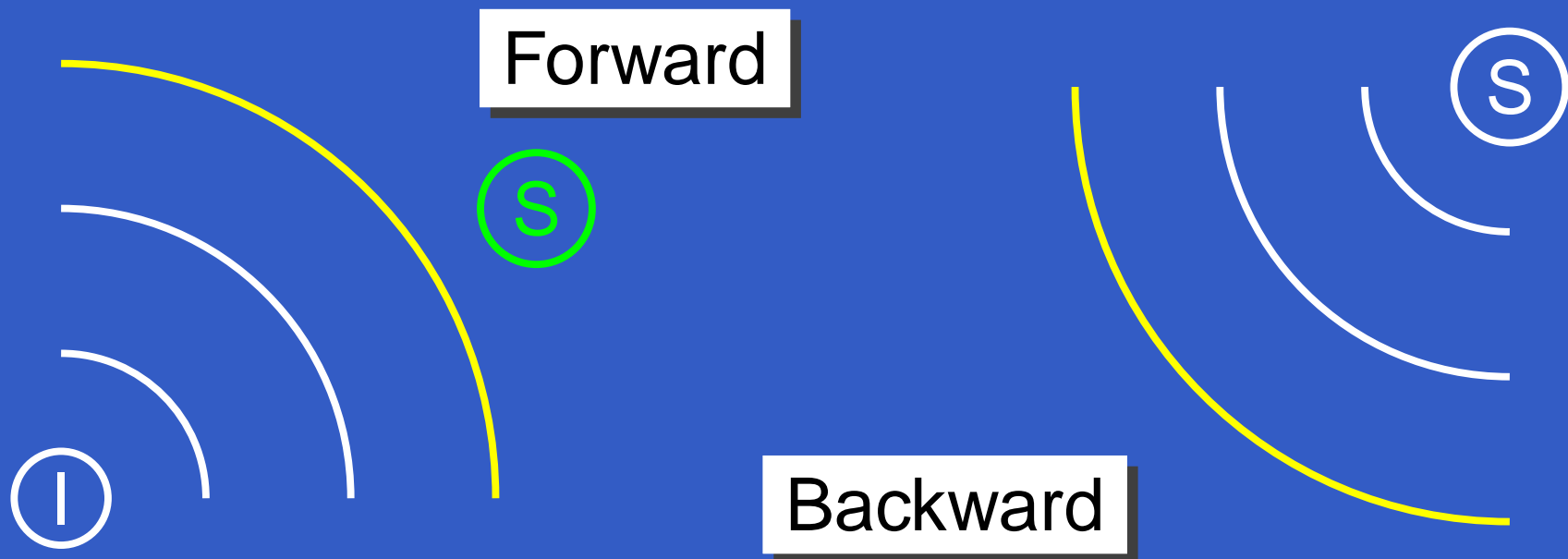
# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:



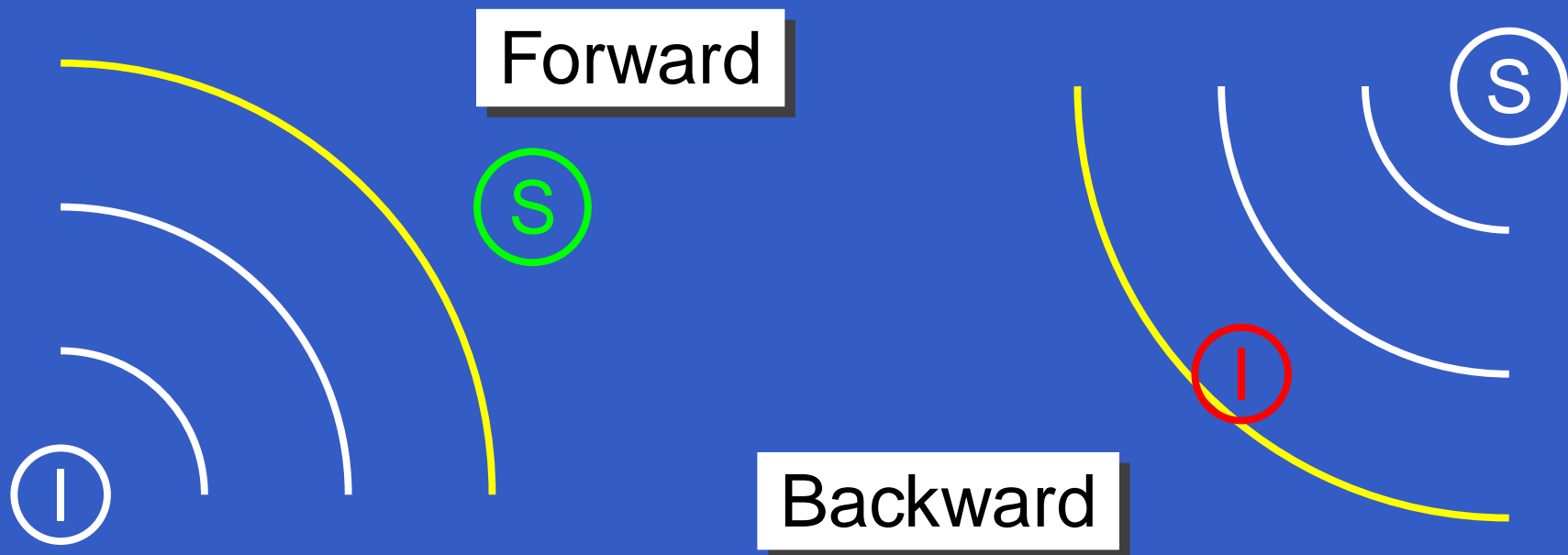
# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:



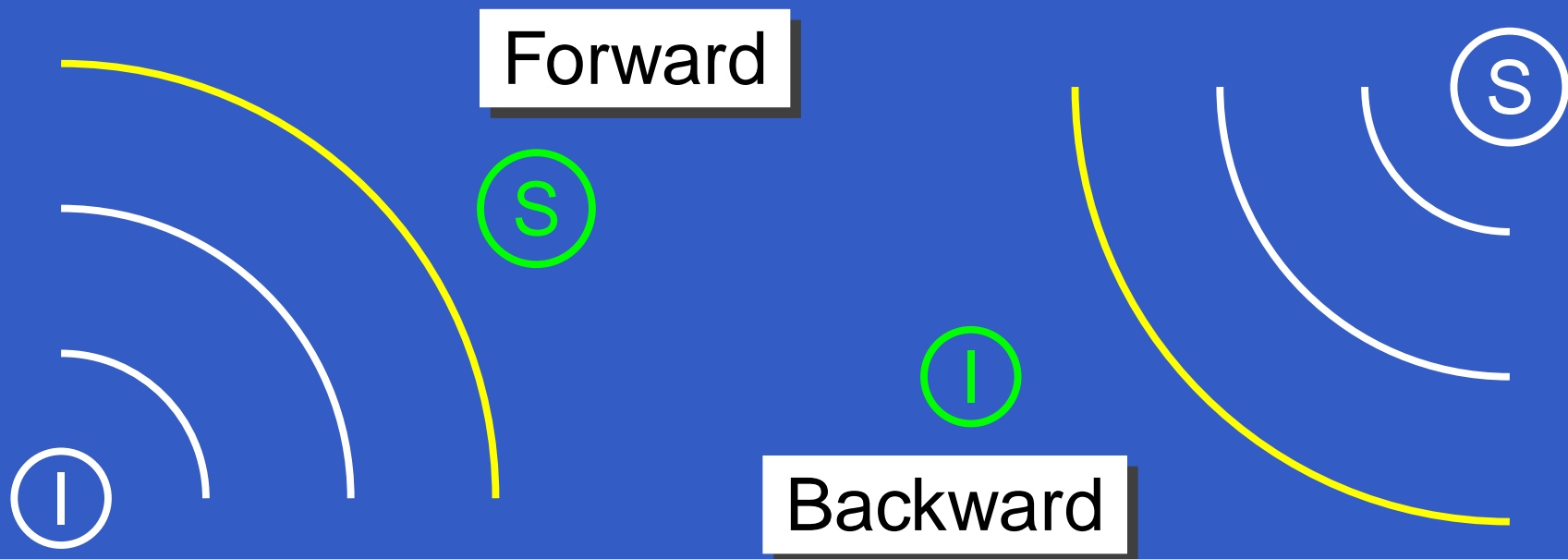
# Forward v.s. Backward

- Verification of a **safety property** → is a MTN marking **reachable** ?
- Two approaches:



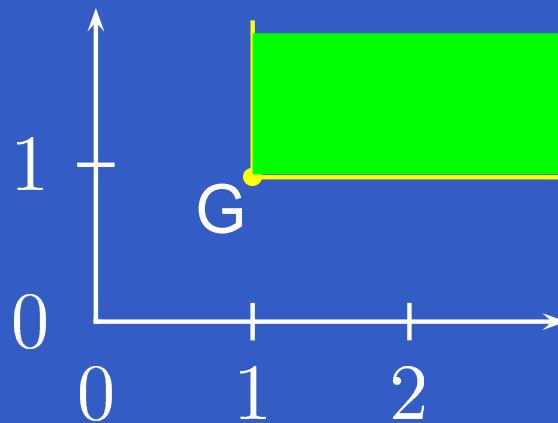
# Forward v.s. Backward

- Verification of a **safety property**  $\rightarrow$  is a MTN marking **reachable** ?
- Two approaches:



# Decidability

- The fixed-point algorithm working backwards will finish if the set of unsafe points is **upward-closed** [Abdulla, Cerans, ...].
- An upward-closed set of points (markings) is characterized by its **generator**.



# Datastructures

- To store the set of reachable markings, we need **efficient datastructures** .

# Datastructures

- To store the set of reachable markings, we need **efficient datastructures** .

Which one is best-suited ?

# Datastructures

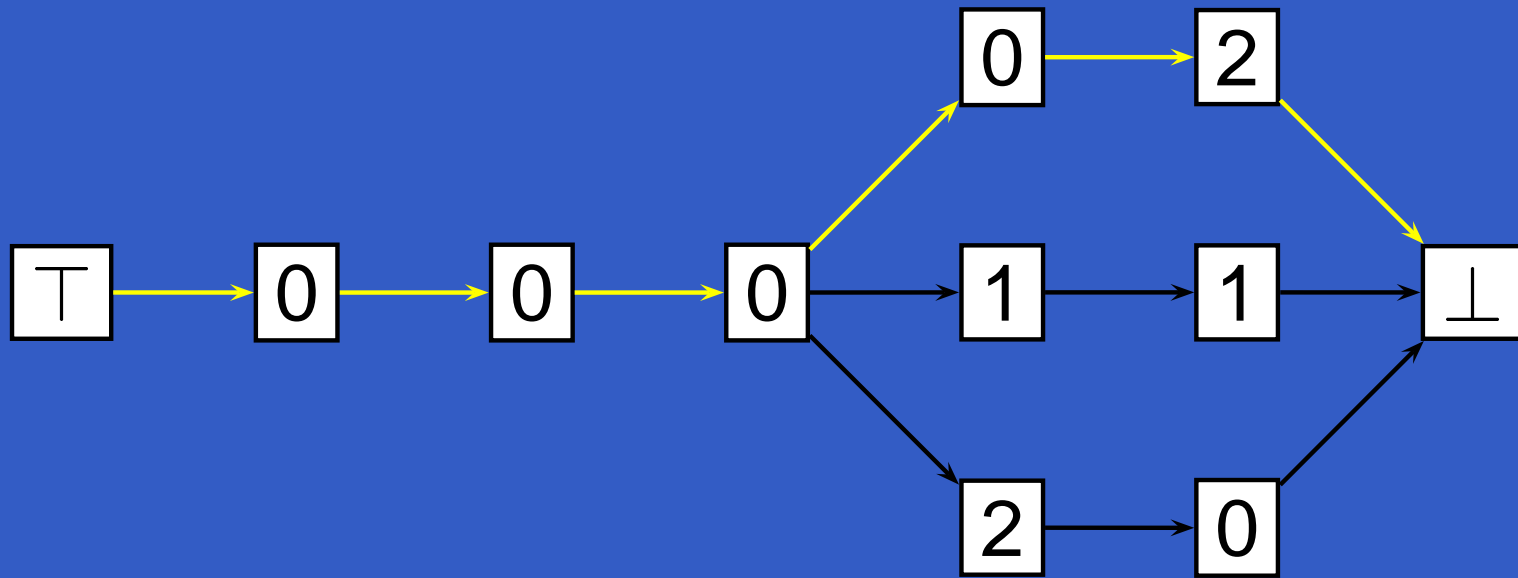
- To store the set of reachable markings, we need **efficient datastructures** .

Which one is best-suited ?

- Let's compare the practical performances of four of them: **CST, IST, DDD, NDD** !



# Covering Sharing Trees



$$(p_1 \geq 0 \wedge p_2 \geq 0 \wedge p_3 \geq 0 \wedge p_4 \geq 0 \wedge p_5 \geq 2)$$

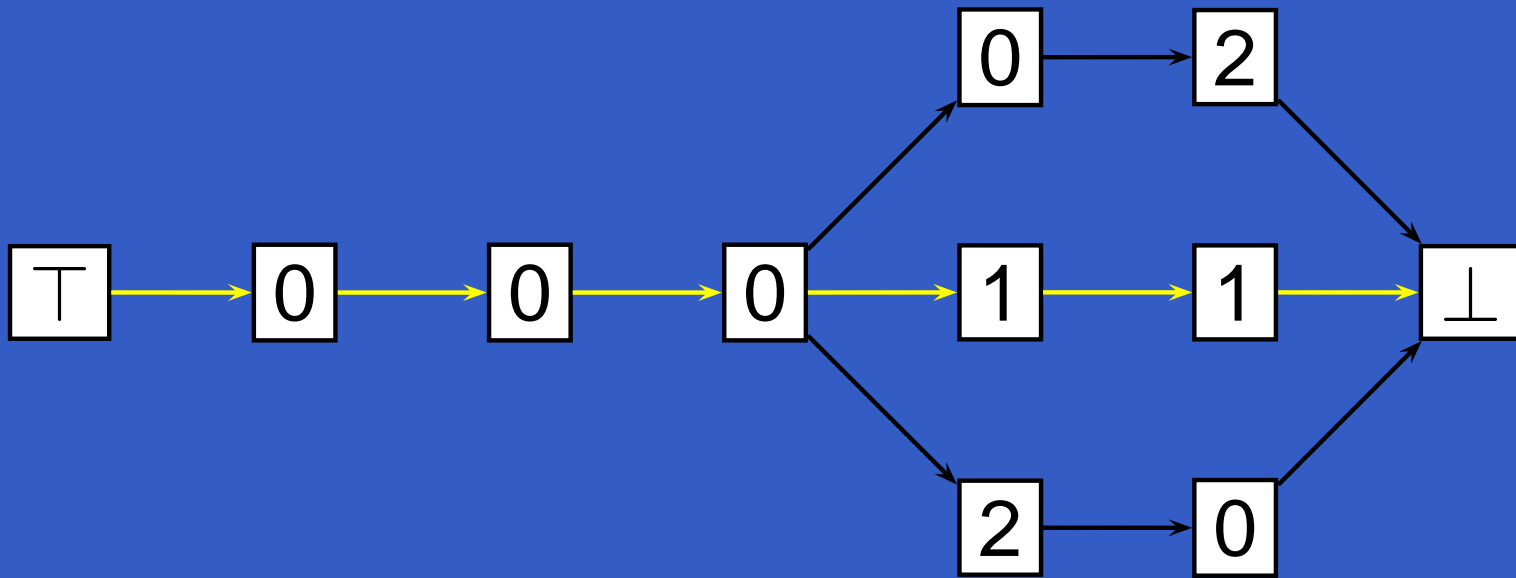
$$\vee$$

$$(p_1 \geq 0 \wedge p_2 \geq 0 \wedge p_3 \geq 0 \wedge p_4 \geq 1 \wedge p_5 \geq 1)$$

$$\vee$$

$$(p_1 \geq 0 \wedge p_2 \geq 0 \wedge p_3 \geq 0 \wedge p_4 \geq 2 \wedge p_5 \geq 0)$$

# Covering Sharing Trees



$$(p_1 \geq 0 \wedge p_2 \geq 0 \wedge p_3 \geq 0 \wedge p_4 \geq 0 \wedge p_5 \geq 2)$$

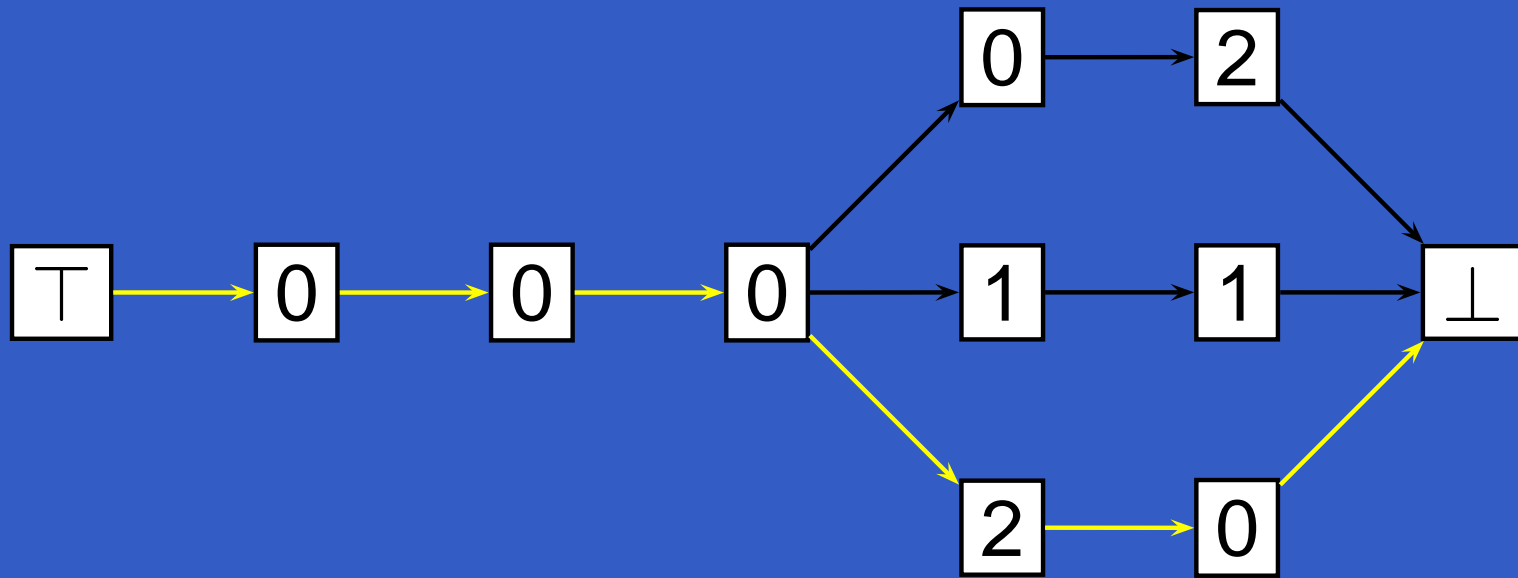
∨

$$(p_1 \geq 0 \wedge p_2 \geq 0 \wedge p_3 \geq 0 \wedge p_4 \geq 1 \wedge p_5 \geq 1)$$

∨

$$(p_1 \geq 0 \wedge p_2 \geq 0 \wedge p_3 \geq 0 \wedge p_4 \geq 2 \wedge p_5 \geq 0)$$

# Covering Sharing Trees



$$(p_1 \geq 0 \wedge p_2 \geq 0 \wedge p_3 \geq 0 \wedge p_4 \geq 0 \wedge p_5 \geq 2)$$

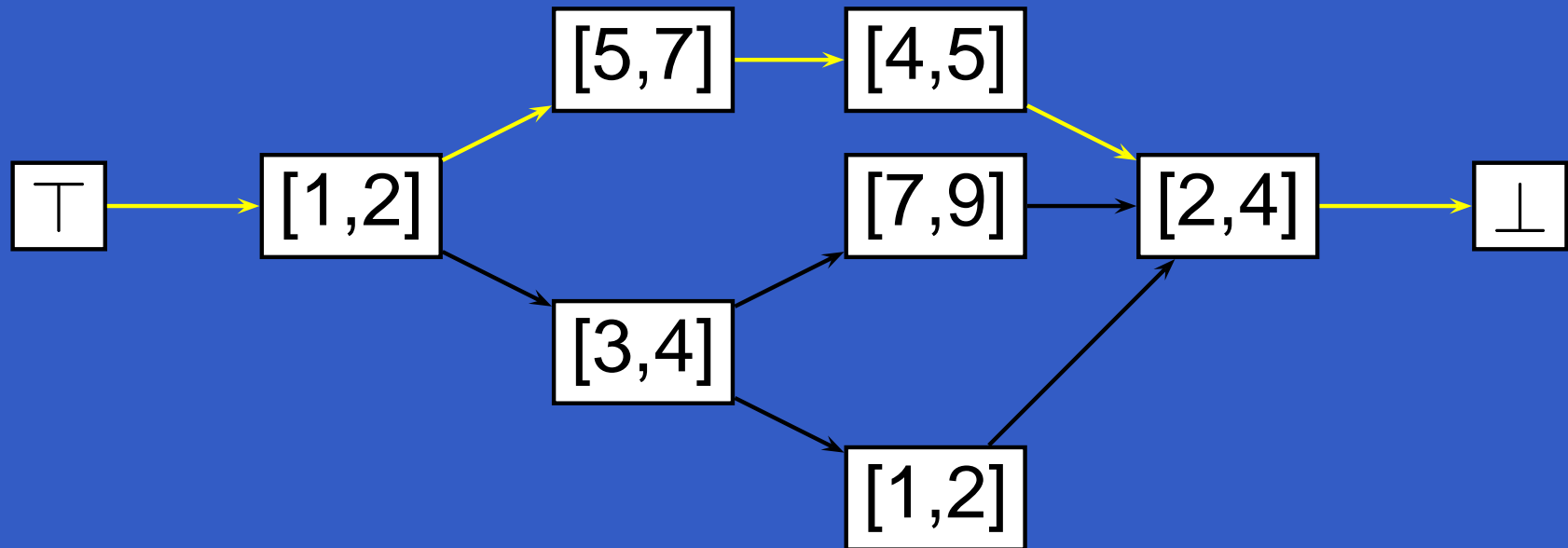
$$\vee$$

$$(p_1 \geq 0 \wedge p_2 \geq 0 \wedge p_3 \geq 0 \wedge p_4 \geq 1 \wedge p_5 \geq 1)$$

$$\vee$$

$$(p_1 \geq 0 \wedge p_2 \geq 0 \wedge p_3 \geq 0 \wedge p_4 \geq 2 \wedge p_5 \geq 0)$$

# Interval Sharing Trees



$$(1 \leq m_1 \leq 2) \wedge (5 \leq m_2 \leq 7) \wedge (4 \leq m_3 \leq 5) \wedge (2 \leq m_4 \leq 4)$$

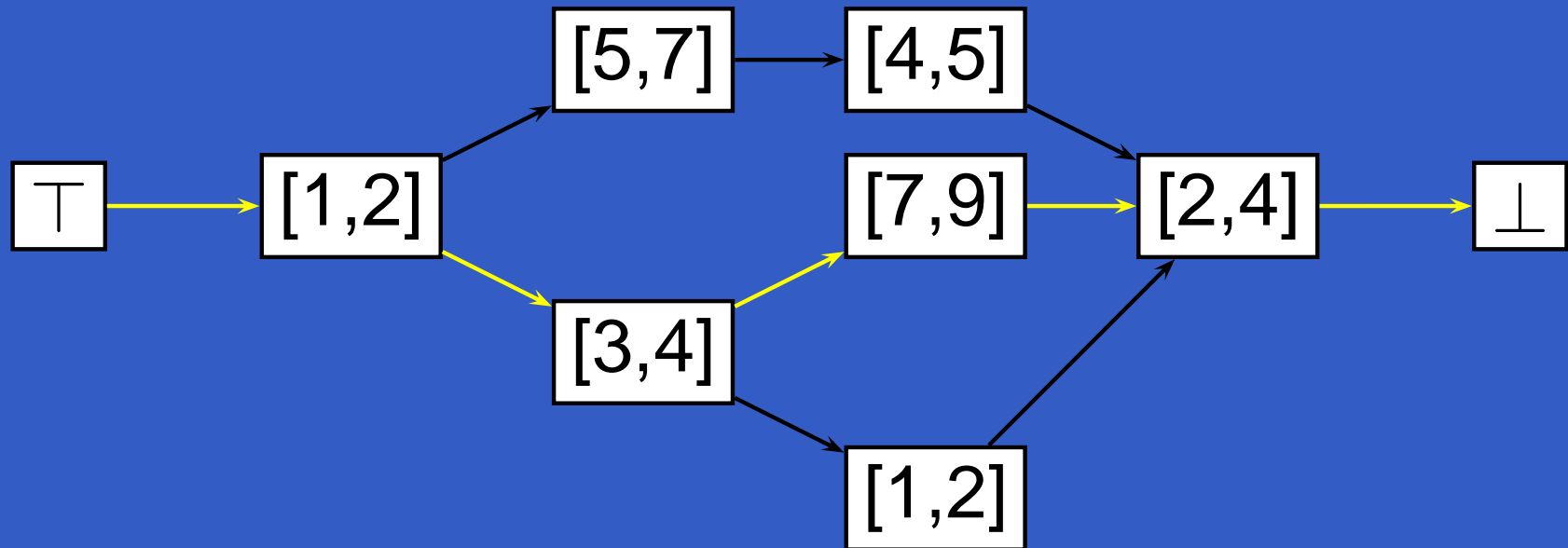
∨

$$(1 \leq m_1 \leq 2) \wedge (3 \leq m_2 \leq 4) \wedge (7 \leq m_3 \leq 9) \wedge (2 \leq m_4 \leq 4)$$

∨

$$(1 \leq m_1 \leq 2) \wedge (3 \leq m_2 \leq 4) \wedge (1 \leq m_3 \leq 2) \wedge (2 \leq m_4 \leq 4)$$

# Interval Sharing Trees



$$(1 \leq m_1 \leq 2) \wedge (5 \leq m_2 \leq 7) \wedge (4 \leq m_3 \leq 5) \wedge (2 \leq m_4 \leq 4)$$

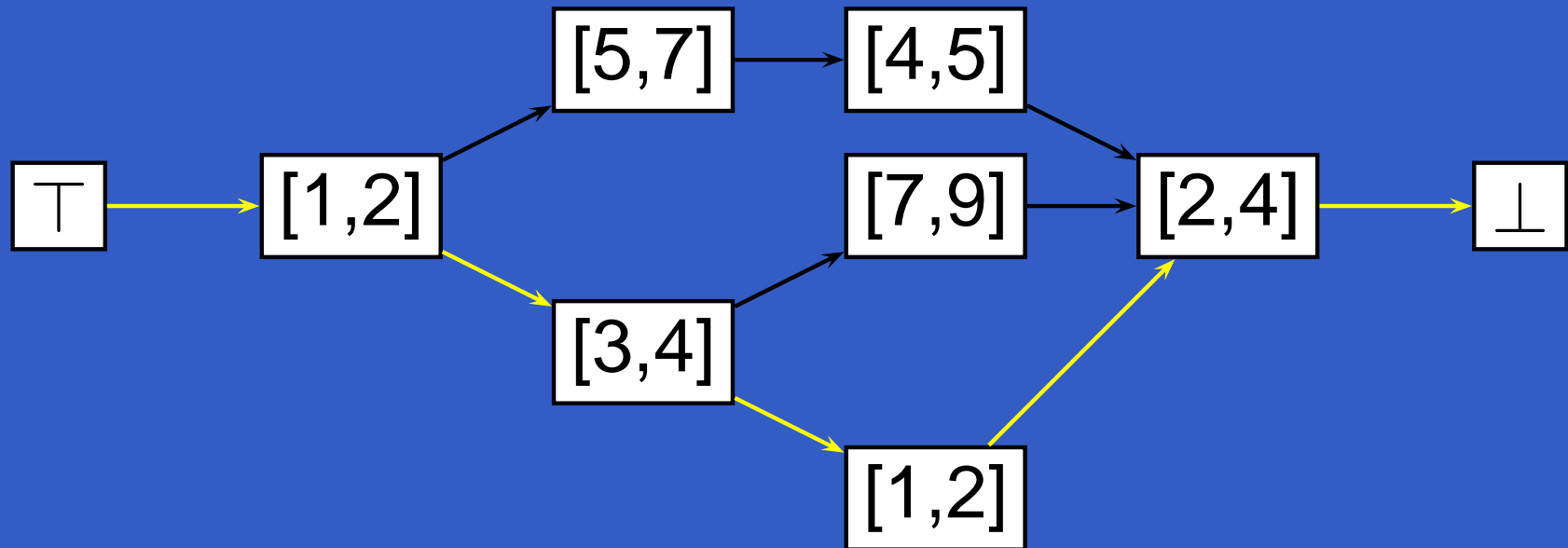
∨

$$(1 \leq m_1 \leq 2) \wedge (3 \leq m_2 \leq 4) \wedge (7 \leq m_3 \leq 9) \wedge (2 \leq m_4 \leq 4)$$

∨

$$(1 \leq m_1 \leq 2) \wedge (3 \leq m_2 \leq 4) \wedge (1 \leq m_3 \leq 2) \wedge (2 \leq m_4 \leq 4)$$

# Interval Sharing Trees



$$(1 \leq m_1 \leq 2) \wedge (5 \leq m_2 \leq 7) \wedge (4 \leq m_3 \leq 5) \wedge (2 \leq m_4 \leq 4)$$

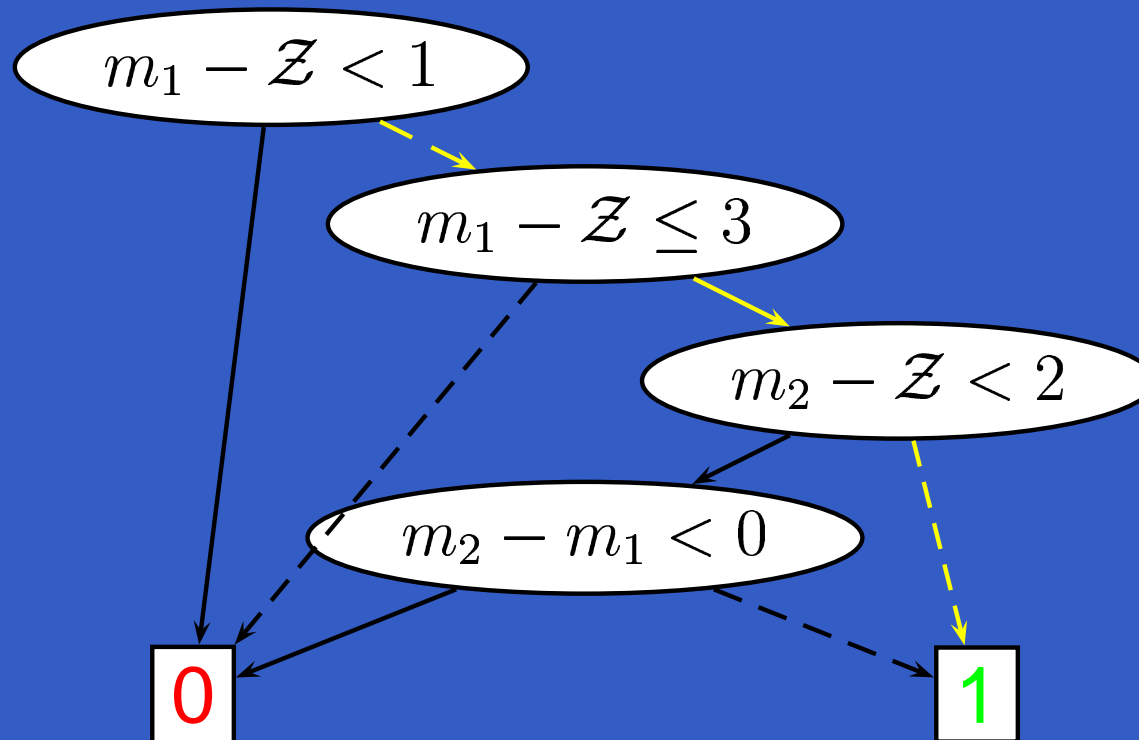
∨

$$(1 \leq m_1 \leq 2) \wedge (3 \leq m_2 \leq 4) \wedge (7 \leq m_3 \leq 9) \wedge (2 \leq m_4 \leq 4)$$

∨

$$(1 \leq m_1 \leq 2) \wedge (3 \leq m_2 \leq 4) \wedge (1 \leq m_3 \leq 2) \wedge (2 \leq m_4 \leq 4)$$

# Difference Decision Diagrams

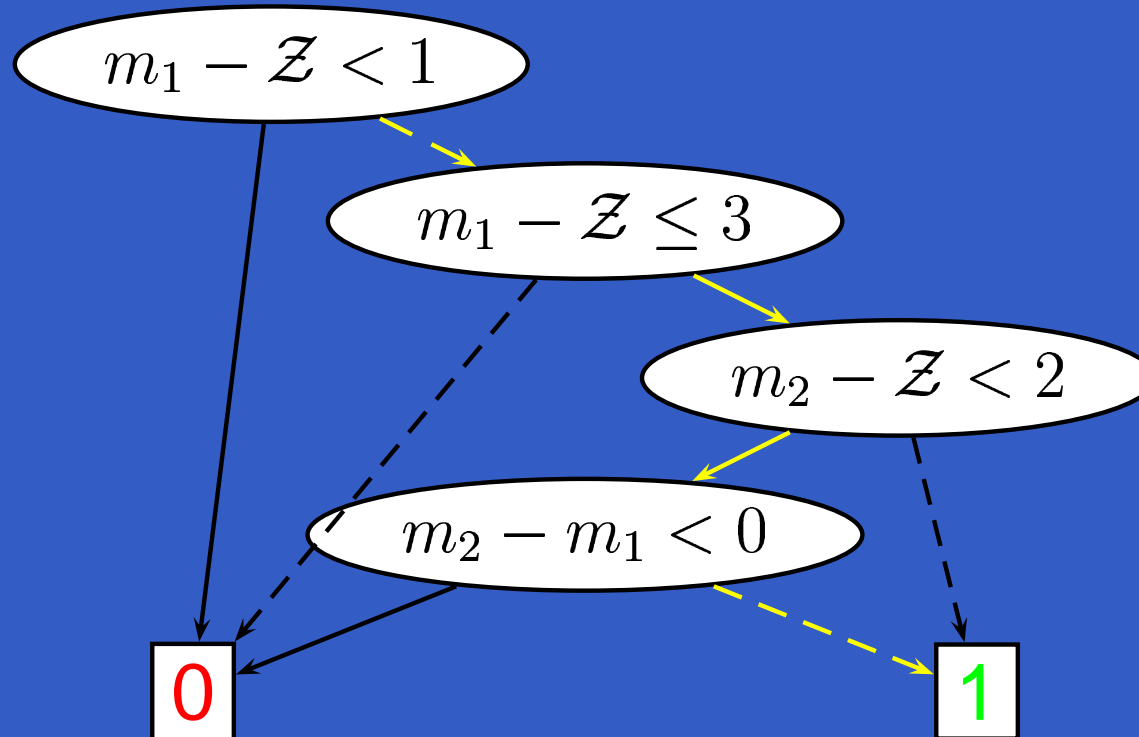


$$\neg(m_1 < 1) \wedge (m_1 \leq 3) \wedge \neg(m_2 < 2)$$

$\vee$

$$\neg(m_1 < 1) \wedge (m_1 \leq 3) \wedge (m_2 < 2) \wedge \neg(m_2 - m_1 < 0)$$

# Difference Decision Diagrams



$$\neg(m_1 < 1) \wedge (m_1 \leq 3) \wedge \neg(m_2 < 2)$$

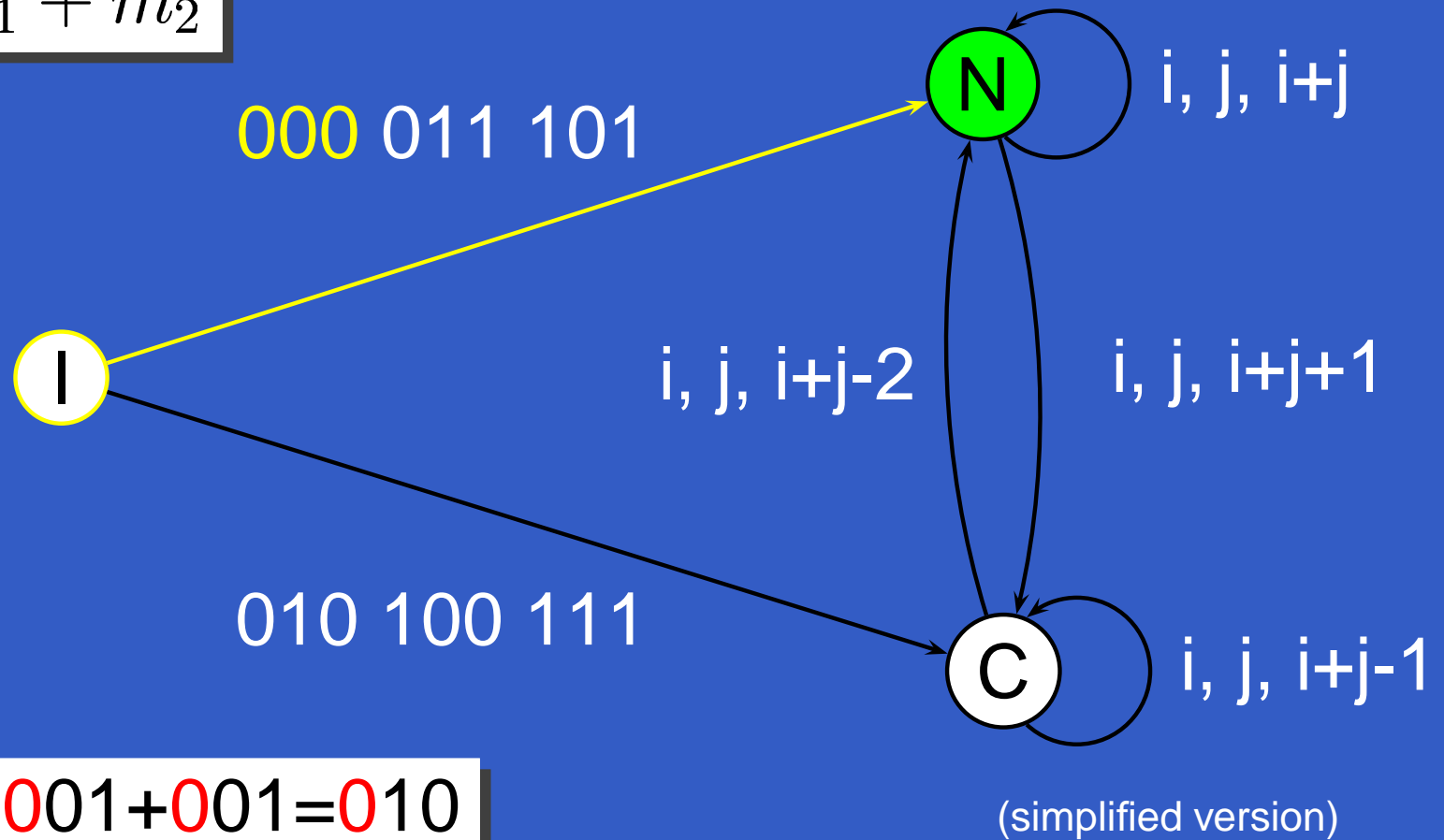
$\vee$

$$\neg(m_1 < 1) \wedge (m_1 \leq 3) \wedge (m_2 < 2) \wedge \neg(m_2 - m_1 < 0)$$



# Number Decision Diagrams

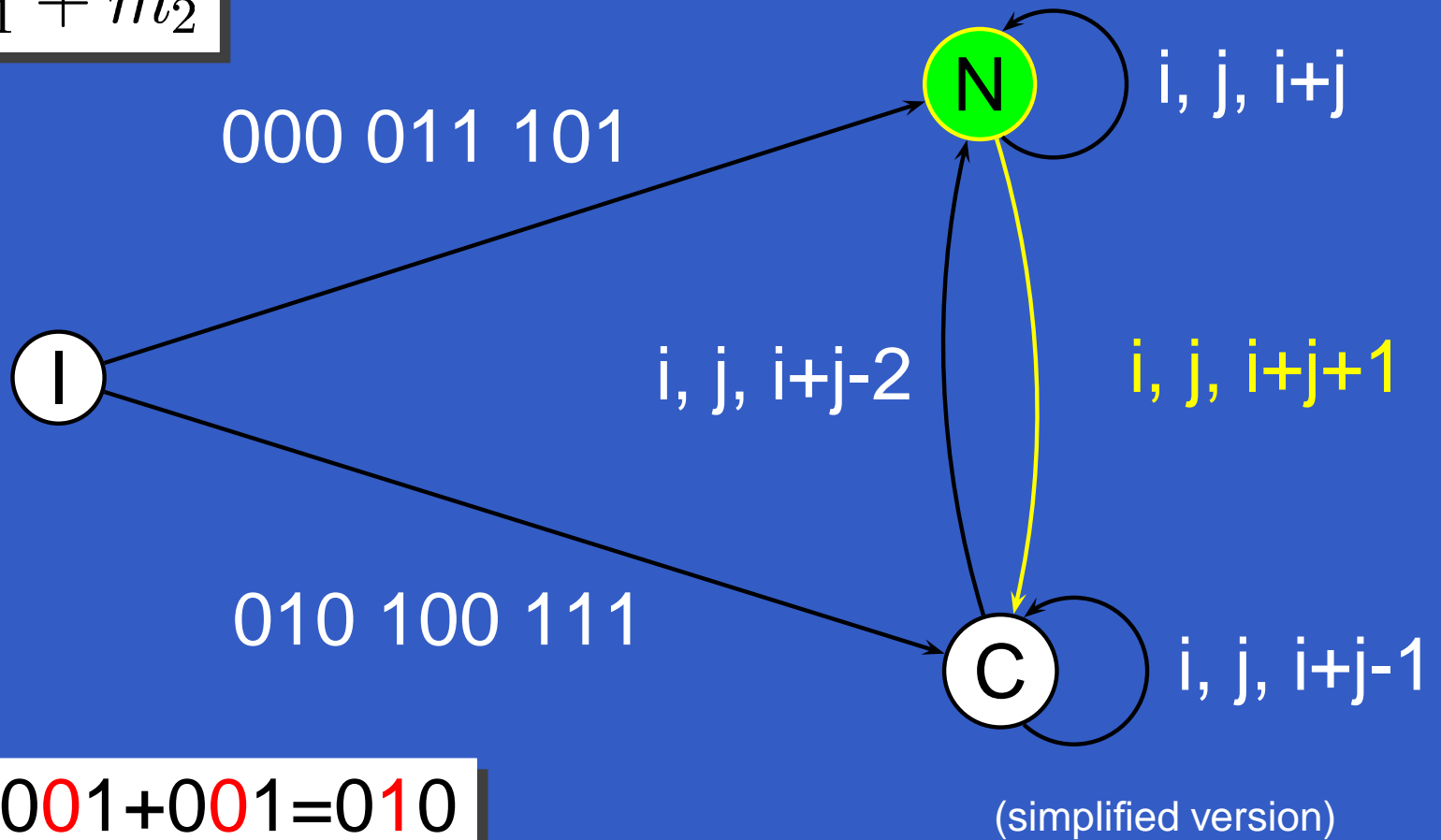
$$m_3 = m_1 + m_2$$



$$1+1=2: 001+001=010$$

# Number Decision Diagrams

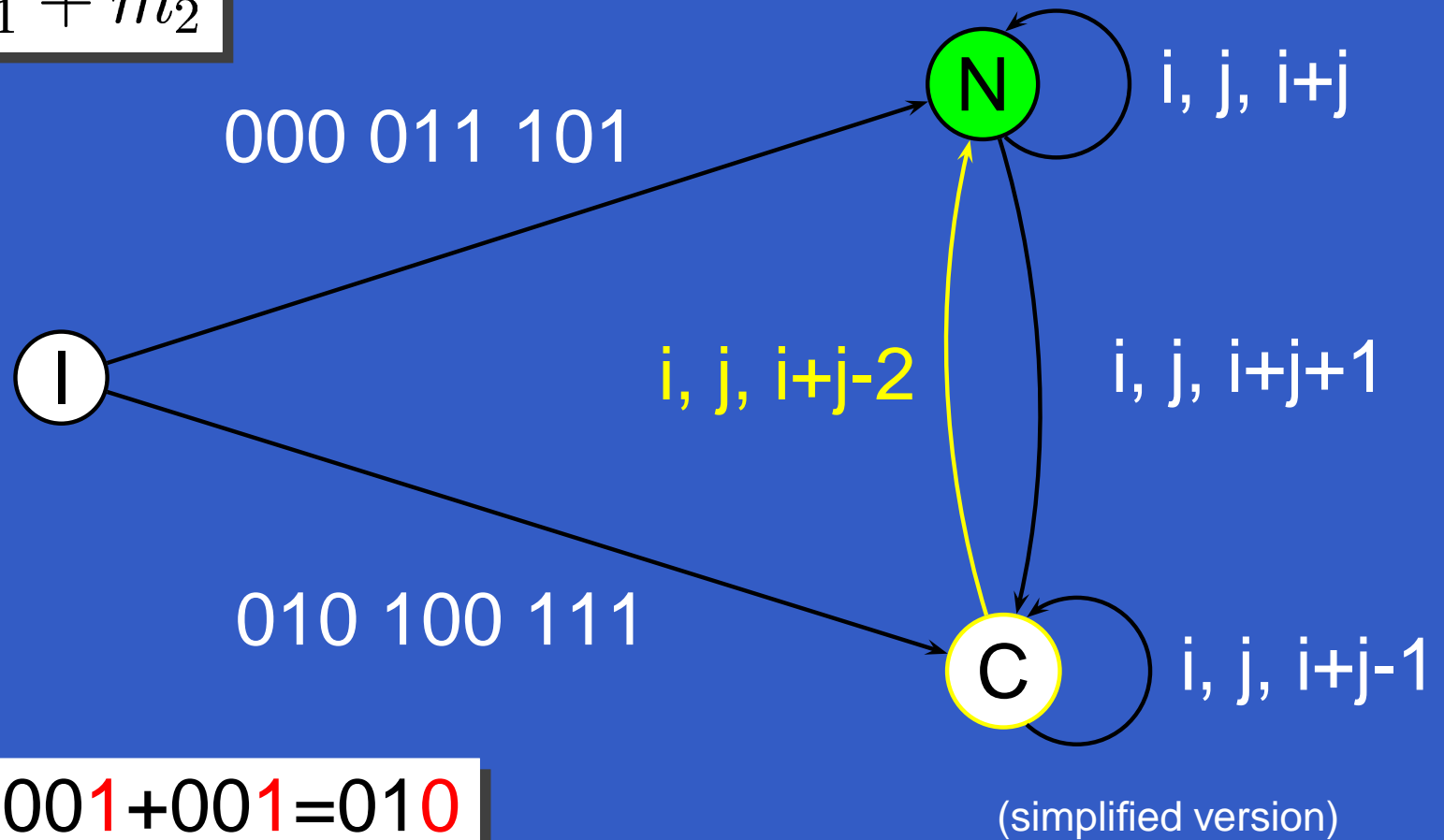
$$m_3 = m_1 + m_2$$



$$1+1=2: 001+001=010$$

# Number Decision Diagrams

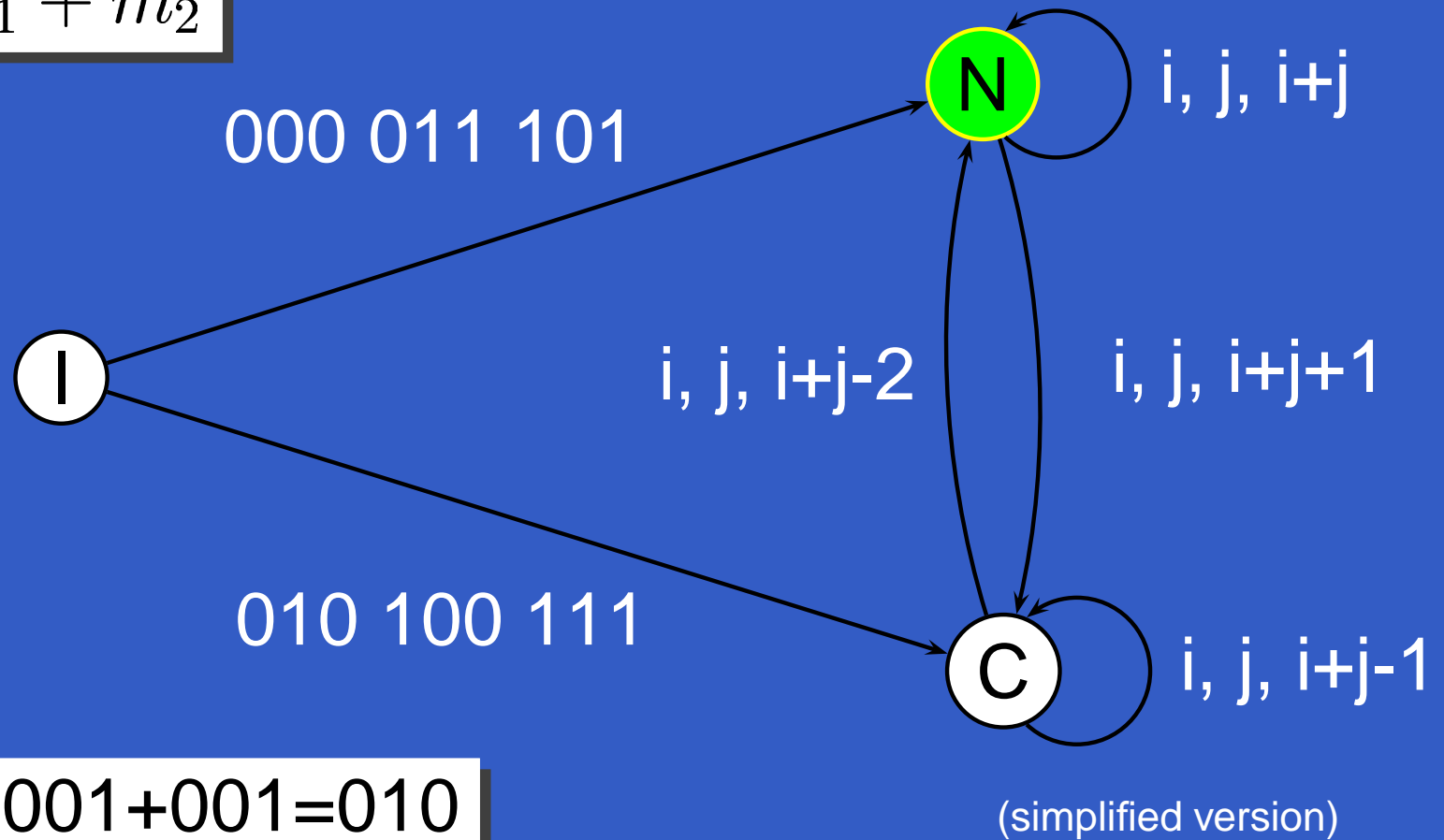
$$m_3 = m_1 + m_2$$



$$1+1=2: 001+001=010$$

# Number Decision Diagrams

$$m_3 = m_1 + m_2$$



$$1+1=2: 001+001=010$$

# The comparison

- First, we need a good set of examples:

# The comparison

- First, we need a good set of examples:
  - Bounded or unbounded Petri nets;

# The comparison

- First, we need a good set of examples:
  - Bounded or unbounded Petri nets;
  - Cache coherency protocol;

# The comparison

- First, we need a good set of examples:
  - Bounded or unbounded Petri nets;
  - Cache coherency protocol;
  - Multi-threaded Java programs.



# The comparison

- First, we need a good set of examples:
  - Bounded or unbounded Petri nets;
  - Cache coherency protocol;
  - Multi-threaded Java programs.
- Then, select the set of parameters:

# The comparison

- First, we need a good set of examples:
  - Bounded or unbounded Petri nets;
  - Cache coherency protocol;
  - Multi-threaded Java programs.
- Then, select the set of parameters:
  - Execution time (User/System);

# The comparison

- First, we need a good set of examples:
  - Bounded or unbounded Petri nets;
  - Cache coherency protocol;
  - Multi-threaded Java programs.
- Then, select the set of parameters:
  - Execution time (User/System);
  - Memory consumption (Resident/Data/Total/... );

# The comparison

- First, we need a good set of examples:
  - Bounded or unbounded Petri nets;
  - Cache coherency protocol;
  - Multi-threaded Java programs.
- Then, select the set of parameters:
  - Execution time (User/System);
  - Memory consumption (Resident/Data/Total/...);
  - Bottleneck operations;

# The comparison

- First, we need a good set of examples:
  - Bounded or unbounded Petri nets;
  - Cache coherency protocol;
  - Multi-threaded Java programs.
- Then, select the set of parameters:
  - Execution time (User/System);
  - Memory consumption (Resident/Data/Total/...);
  - Bottleneck operations;
  - ...

# A twofold comparison – First phase

- **BABYLON**: an unified model-checker.

# A twofold comparison – First phase

- **BABYLON**: an unified model-checker.
- The datastructures are seen as **constraint solvers**;

# A twofold comparison – First phase

- **BABYLON**: an unified model-checker.
- The datastructures are seen as **constraint solvers**;
- The three model-checking algorithms are **shared** among the datastructures;



# A twofold comparison – First phase

- **BABYLON**: an unified model-checker.
- The datastructures are seen as **constraint solvers**;
- The three model-checking algorithms are **shared** among the datastructures;
- Only Petri nets.

# A twofold comparison – First phase

- **BABYLON**: an unified model-checker.
- The datastructures are seen as **constraint solvers**;
- The three model-checking algorithms are **shared** among the datastructures;
- Only Petri nets.

```
class Set {  
    virtual Set * Union (const Set * S) = 0;  
    virtual Set * Intersection (const Set * S) = 0 ;  
    virtual Set * Difference (const Set * S) = 0 ;  
    virtual bool IsEmpty() = 0 ;  
    virtual Set * Pre(void) = 0 ; virtual Set * Pre(int i) = 0 ;  
    virtual void EmptySet() = 0 ; [ . . . ] }  
}
```

# Results – Second Phase

Execution times (sec.) – Algorithm 3

Example	CST	IST	DDD	NDD
Peterson	0.54	0.34	0.33	2'172.19
Lamport	0.14	0.1	0.13	139.19
Multipool	14.19	9.36	3.04	>3 hours
Mesh3x2	466.31	513.62	195.99	>3 hours

(...)

# A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.

# A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part → LASH library (ULg);

# A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part → LASH library (ULg);
  - DDD part → DDD library (Møller) + new extensions.

# A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part → LASH library (ULg);
  - DDD part → DDD library (Møller) + new extensions.
- Other tools already exist for CST and IST.

# A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part → LASH library (ULg);
  - DDD part → DDD library (Møller) + new extensions.
- Other tools already exist for CST and IST.
- The algorithms are **peculiar** to the datastructures.



# A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part → LASH library (ULg);
  - DDD part → DDD library (Møller) + new extensions.
- Other tools already exist for CST and IST.
- The algorithms are **peculiar** to the datastructures.
- Many **optimizations (invariants)** have been used.

# A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part → LASH library (ULg);
  - DDD part → DDD library (Møller) + new extensions.
- Other tools already exist for CST and IST.
- The algorithms are **peculiar** to the datastructures.
- Many **optimizations (invariants)** have been used.
- **New optimizations** techniques have been developed for DDD and NDD.

# A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part → LASH library (ULg);
  - DDD part → DDD library (Møller) + new extensions.
- Other tools already exist for CST and IST.
- The algorithms are **peculiar** to the datastructures.
- Many **optimizations (invariants)** have been used.
- **New optimizations** techniques have been developed for DDD and NDD.
- Large set of examples.

# Results – Second Phase

## Execution times (sec.)

Example	CST	IST	DDD	NDD
Peterson	0.88	0.2	0.31	691.12
Multipool	3.39	5.44	0.49	1'309.12
Client/Server	0.27	0.09	0.44	3.34
Client/Server (Ex I)	–	–	0.04	0.9
Client/Server (He I)	0.04	0	6.28	–
Illinois	–	0	0.04	0.66

(...)

# Conclusion

- NDD are definitely too slow. (Next version ?)

# Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the **same performances**:

# Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the **same performances**:
  - DDD are quicker (more powerful implementation)...

# Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the **same performances**:
  - DDD are quicker (more powerful implementation)...
  - ...but have poor memory consumption.



# Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the **same performances**:
  - DDD are quicker (more powerful implementation)...
  - ...but have poor memory consumption.
  - Their increased expressivity is not interesting here.

# Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the **same performances**:
  - DDD are quicker (more powerful implementation)...
  - ...but have poor memory consumption.
  - Their increased expressivity is not interesting here.
- IST and CST thus seem **best-suited**.

# Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the **same performances**:
  - DDD are quicker (more powerful implementation)...
  - ...but have poor memory consumption.
  - Their increased expressivity is not interesting here.
- IST and CST thus seem **best-suited**.
- These experiments could be **largely refined**.

# Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the **same performances**:
  - DDD are quicker (more powerful implementation)...
  - ...but have poor memory consumption.
  - Their increased expressivity is not interesting here.
- IST and CST thus seem **best-suited**.
- These experiments could be **largely refined**.
- Other datastructures ?

# Personal contributions

- Development (with Giorgio) of the methodology.

# Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.

# Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.

# Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:



# Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:
  - for the invariant-based optimization;

# Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:
  - for the invariant-based optimization;
  - to let it handle transfers (MTN).

# Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:
  - for the invariant-based optimization;
  - to let it handle transfers (MTN).
- Implementation of BABYLON (with Pierre and Laurent)

# Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:
  - for the invariant-based optimization;
  - to let it handle transfers (MTN).
- Implementation of BABYLON (with Pierre and Laurent)
- Benchmarks and collection of the results.