Lecture 19

6.2.4 Language classes with polynomial complexity 194

Language classes with polynomial complexity

Language classes with polynomial complexity

- **Class of polynomial time complexity *P***

Language classes with polynomial complexity

- **Class of polynomial time complexity P**

$$P = \bigcup_{i \geq 1} \mathrm{DTIME}(\, n^i \,)$$

Language classes with polynomial complexity

- **Class of polynomial time complexity *P***

$$P = \bigcup_{i \geq 1} \mathrm{DTIME}(\, n^i \,)$$

- **Class of nondeterministic polynomial time complexity *NP***

Language classes with polynomial complexity

- **Class of polynomial time complexity *P***

$$P = \bigcup_{i \geq 1} \mathrm{DTIME}(\, n^i \,)$$

- **Class of nondeterministic polynomial time complexity *NP***

$$NP = \bigcup_{i \geq 1} \mathrm{NTIME}(\, n^i \,)$$

Language classes with polynomial complexity

Language classes with polynomial complexity

- **Language L_{sat}**

Language classes with polynomial complexity

- Language L_{sat}
- Logical expression

Language classes with polynomial complexity

- Language L_{sat}
- Logical expression
  - Variables, brackets, logical operator ***and*** ($\wedge$), logical operator ***or*** ($\vee$) and the operator of ***negation*** ($\neg$)

Language classes with polynomial complexity

- Language L_{sat}
- Logical expression
  - Variables, brackets, logical operator ***and*** **(∧),** logical operator ***or*** **(∨) and the operator of** ***negation*** **(¬)**
  - **Variables take values 0 or 1**

Language classes with polynomial complexity

- Language L_{sat}
- Logical expression
  - Variables, brackets, logical operator ***and*** ($\wedge$), logical operator ***or*** ($\vee$) and the operator of ***negation*** ($\neg$)
  - Variables take values **0** or **1**
  - String w – logical expression

Language classes with polynomial complexity

- **Language L_{sat}**
- **Logical expression**
  - **Variables, brackets, logical operator *and* ($\wedge$), logical operator *or* ($\vee$) and the operator of *negation* ($\neg$)**
  - **Variables take values 0 or 1**
  - **String *w* – logical expression**

$$y_1 \wedge y_2$$

Language classes with polynomial complexity

- Language L_{sat}
- Logical expression
  - Variables, brackets, logical operator ***and*** ($\wedge$), logical operator ***or*** ($\vee$) and the operator of ***negation*** ($\neg$)
  - Variables take values **0** or **1**
  - String ***w*** – logical expression

$$y_1 \wedge y_2$$

$$y_1 \wedge \neg y_1$$

Language classes with polynomial complexity

- Language L_{sat}
- Logical expression
  - Variables, brackets, logical operator ***and*** ($\wedge$), logical operator ***or*** ($\vee$) and the operator of ***negation*** ($\neg$)
  - Variables take values **0** or **1**
  - String *w* – logical expression

$$y_1 \wedge y_2$$

$$y_1 \wedge \neg y_1$$

- String *w* belongs to L_{sat} if it is possible to give values **0** or **1** to the variables of the logical expression so that it evaluates to **1**

Language classes with polynomial complexity

Language classes with polynomial complexity

$$y_1 \quad \wedge \quad y_2$$

Language classes with polynomial complexity

$0 \quad . \quad \wedge \quad y_2$

Language classes with polynomial complexity

0 . ^ **0**

Language classes with polynomial complexity

$$0 \, . \;\; \wedge \;\; 0 \;\; \neq 1$$

Language classes with polynomial complexity

$$0 \;.\; \wedge \quad 0 \quad \neq 1$$

$$y_1 \wedge y_2 \notin L_{sat}$$

Language classes with polynomial complexity

$$y_1 \quad \wedge \quad y_2$$

$y_1 \wedge y_2 \notin L_{sat}$

Language classes with polynomial complexity

1 . ∧ y_2

$y_1 \wedge y_2 \notin L_{sat}$

Language classes with polynomial complexity

1 . ∧ 0

$y_1 \wedge y_2 \notin L_{sat}$

Language classes with polynomial complexity

$$1 \; . \; \wedge \; 0 \; \neq 1$$

$$y_1 \wedge y_2 \notin L_{sat}$$

Language classes with polynomial complexity

$$y_1 \quad \wedge \quad y_2$$

$$y_1 \wedge y_2 \notin L_{sat}$$

Language classes with polynomial complexity

$0_{\_} \wedge y_2$

$y_1 \wedge y_2 \notin L_{sat}$

Language classes with polynomial complexity

0 . ^ **1**

$y_1 \wedge y_2 \notin L_{sat}$

Language classes with polynomial complexity

$$0 \; . \quad \wedge \quad 1 \quad \neq 1$$

$y_1 \wedge y_2 \notin L_{sat}$

Language classes with polynomial complexity

$$y_1 \quad \wedge \quad y_2$$

$$y_1 \wedge y_2 \notin L_{sat}$$

Language classes with polynomial complexity

1 . $\wedge$ y_2

$y_1 \wedge y_2 \notin L_{sat}$

Language classes with polynomial complexity

1 . ^ **1**

$y_1 \wedge y_2 \notin L_{sat}$

Language classes with polynomial complexity

$$1 \quad . \quad \wedge \quad 1 \quad = 1$$

$$y_1 \wedge y_2 \notin L_{sat}$$

Language classes with polynomial complexity

$$1 \;.\; \wedge \quad 1 \quad = 1$$

$y_1 \wedge y_2 \in L_{sat}$

Language classes with polynomial complexity

$$1 \; . \; \wedge \; 1 \; = 1$$

$$y_1 \wedge y_2 \in L_{sat}$$

$$y_1 \wedge \neg y_1$$

Language classes with polynomial complexity

$$1 \quad . \quad \wedge \quad 1 \quad = 1$$

$$y_1 \wedge y_2 \in L_{sat}$$

$$0 \quad \wedge \quad \neg \quad y_1$$

Language classes with polynomial complexity

$$1 \quad \land \quad 1 \quad = 1$$

$$y_1 \land y_2 \in L_{sat}$$

$$0 \quad \land \quad \neg \quad 0$$

Language classes with polynomial complexity

$$1 \wedge 1 = 1$$

$$y_1 \wedge y_2 \in L_{sat}$$

$$0 \wedge \neg 0 \neq 1$$

Language classes with polynomial complexity

$$1 \quad \wedge \quad 1 \quad = 1$$

$$y_1 \wedge y_2 \in L_{sat}$$

$$0 \quad \wedge \quad \neg \quad 0 \quad \neq 1$$

$$y_1 \wedge \neg y_1 \notin L_{sat}$$

Language classes with polynomial complexity

$$1 \quad \land \quad 1 \quad = 1$$

$$y_1 \land y_2 \in L_{sat}$$

$$y_1 \quad \land \quad \neg \quad y_1$$

$$y_1 \land \neg y_1 \notin L_{sat}$$

Language classes with polynomial complexity

$$1 \wedge 1 = 1$$

$$y_1 \wedge y_2 \in L_{sat}$$

$$1 \wedge \neg y_1$$

$$y_1 \wedge \neg y_1 \notin L_{sat}$$

Language classes with polynomial complexity

$$1 \;.\; \wedge \quad 1 \quad = 1$$

$$y_1 \wedge y_2 \in L_{sat}$$

$$1 \quad \wedge \quad \neg \quad 1$$

$$y_1 \wedge \neg y_1 \notin L_{sat}$$

Language classes with polynomial complexity

$$1 \wedge 1 = 1$$

$$y_1 \wedge y_2 \in L_{sat}$$

$$1 \wedge \neg 1 \neq 1$$

$$y_1 \wedge \neg y_1 \notin L_{sat}$$

Language classes with polynomial complexity

$$1 \wedge 1 = 1$$

$$y_1 \wedge y_2 \in L_{sat}$$

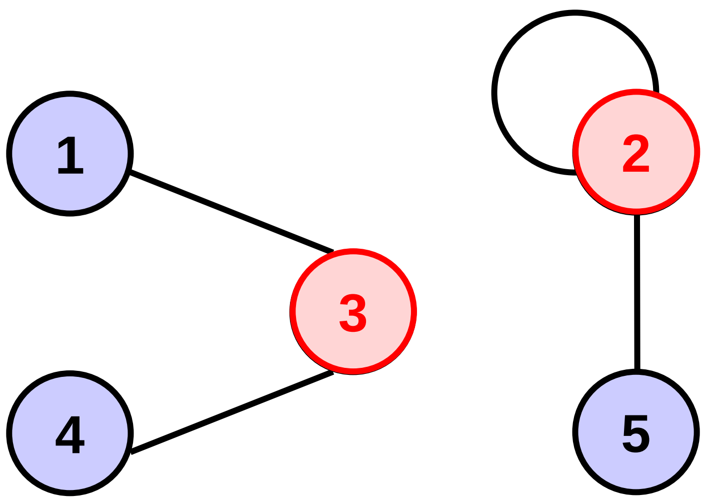
$$1 \wedge \neg 1 \neq 1$$

$$y_1 \wedge \neg y_1 \notin L_{sat}$$

- $L_{sat} \in NP$
  - **TM nondeterministically generates values for all variables of the logical expression and verifies whether the logical expression for the given values evaluates to 1**

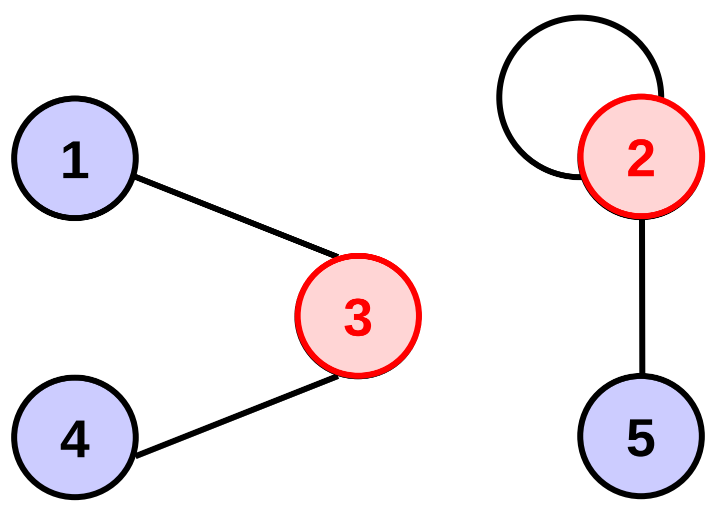
Language classes with polynomial complexity

- ***Language L_{vc}***



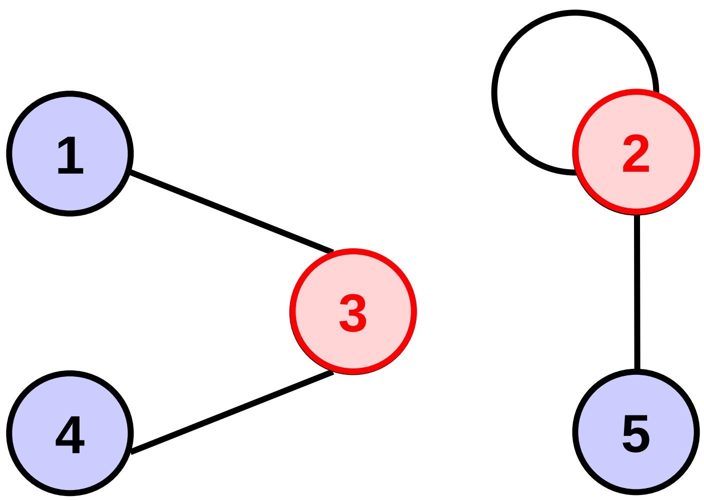
Language classes with polynomial complexity

- ***Language*** L_{vc}
- **G = (V, E) – undirected graph**



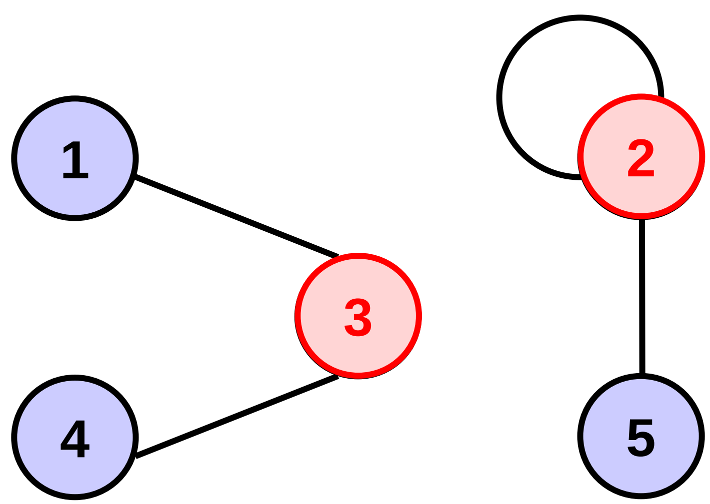
Language classes with polynomial complexity

- ***Language L_{vc}***
- **$G = (V, E)$ – undirected graph**
  - **a set of vertices V and a set of edges E**



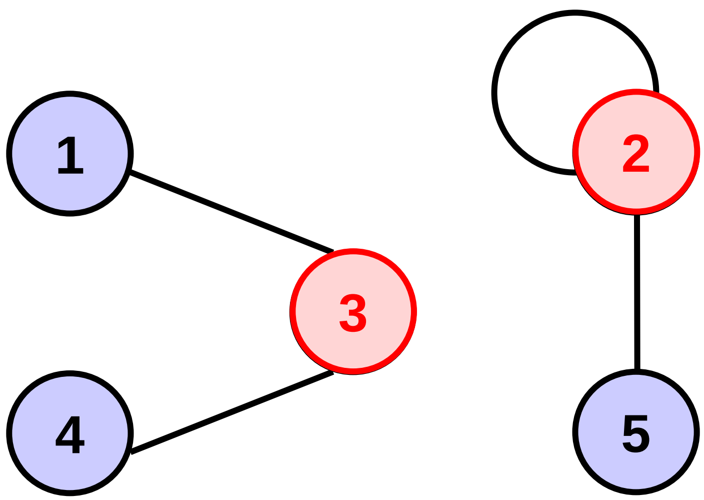
Language classes with polynomial complexity

- *Language L_{vc}*
- $G = (V, E)$ – undirected graph
  - **a set of vertices V and a set of edges E**
- A vertex cover A



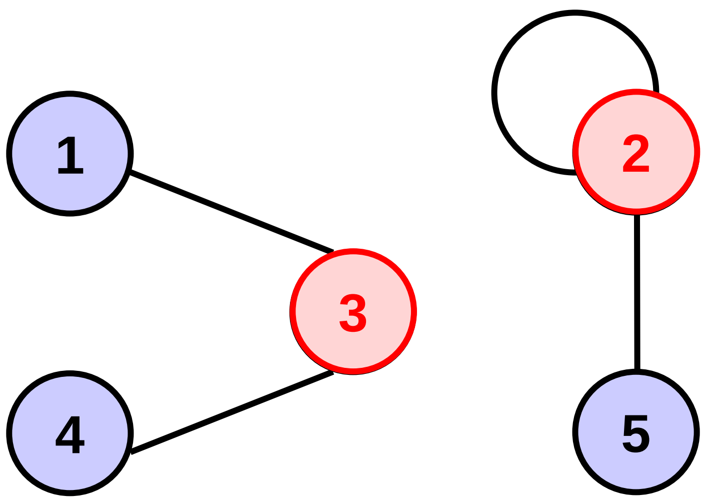
Language classes with polynomial complexity

- *Language* L_{vc}
- $G = (V, E)$ – undirected graph
  - a set of vertices V and a set of edges E
- A vertex cover A
  - Subset of vertices V such that



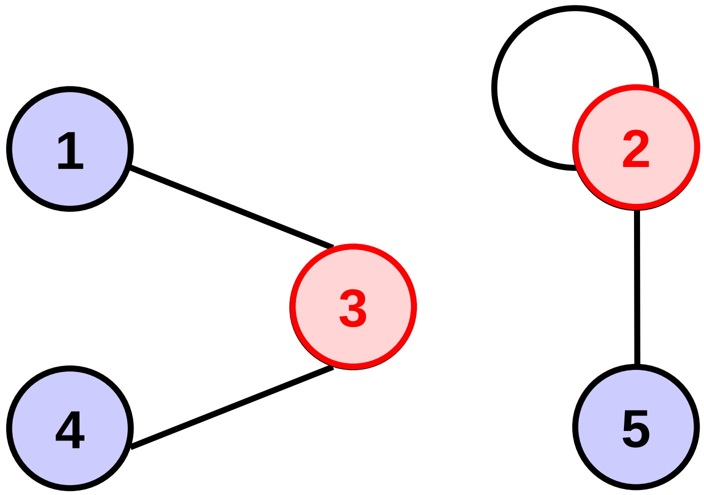
Language classes with polynomial complexity

- *Language* L_{vc}
- $G = (V, E)$ – undirected graph
  - a set of vertices *V* and a set of edges *E*
- A vertex cover *A*
  - Subset of vertices *V* such that
    - — if (x, y) is any edge from *E*



Language classes with polynomial complexity

- *Language L_{vc}*
- $G = (V, E)$ – undirected graph
  - a set of vertices V and a set of edges E
- A vertex cover A
  - Subset of vertices V such that
    - if (x, y) is any edge from E
    - then A contains at least one of the vertices x or y



Language classes with polynomial complexity

Language classes with polynomial complexity

k ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5

Language classes with polynomial complexity

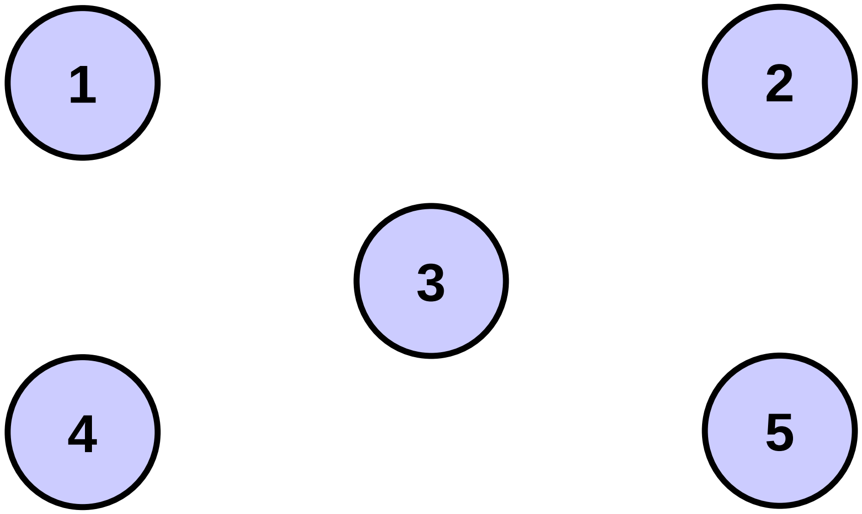
Graph vertices **Graph edges**

k **** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5**

Language classes with polynomial complexity

Graph vertices **Graph edges**

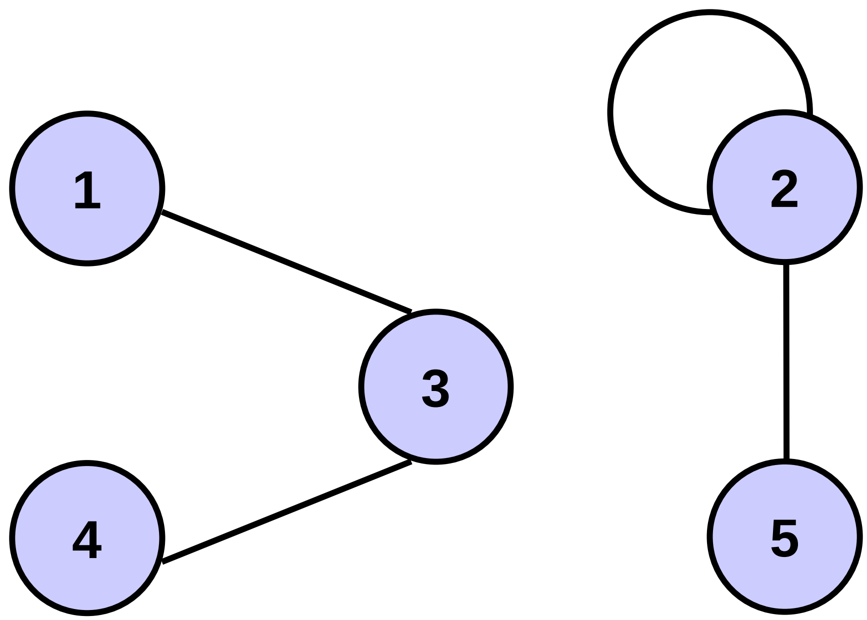
k ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5



Language classes with polynomial complexity

Graph vertices **Graph edges**

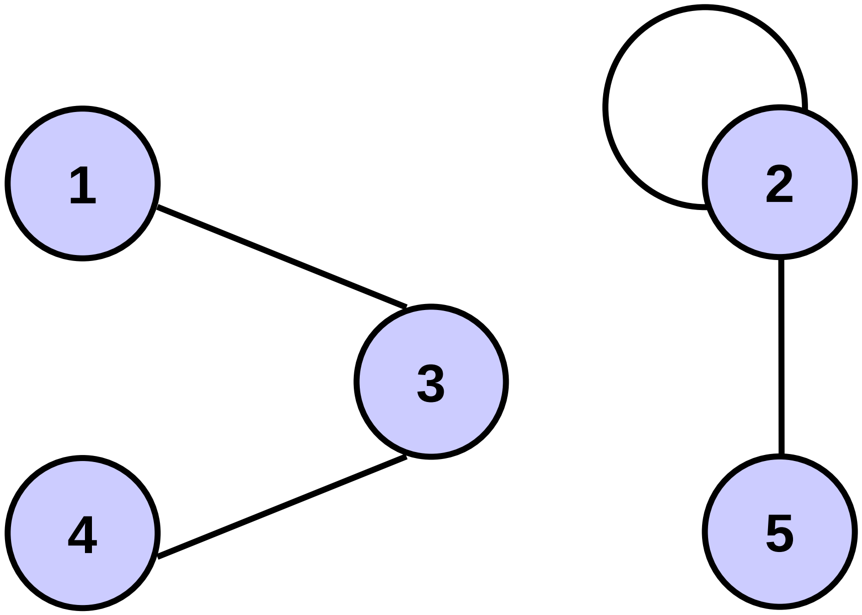
k ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5



Language classes with polynomial complexity

Graph vertices **Graph edges**

k ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5

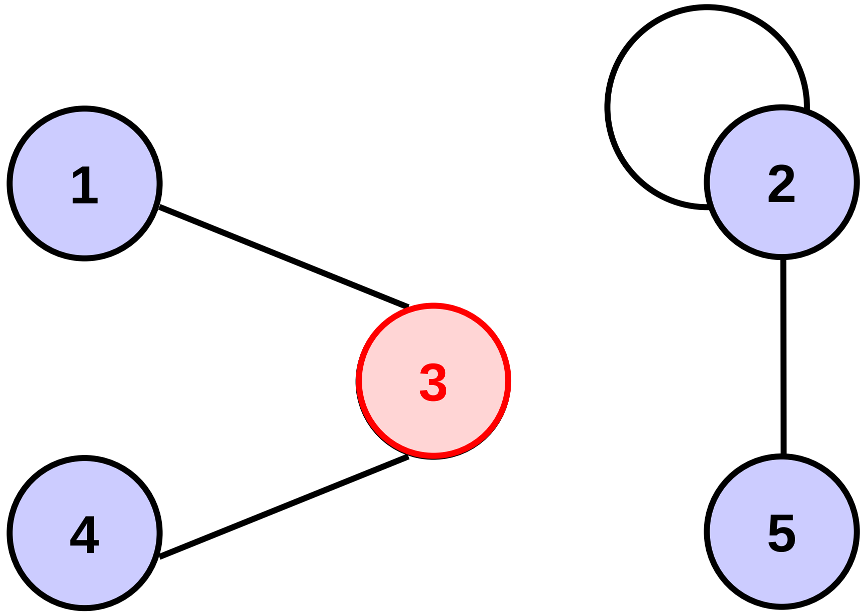


A string belongs to L_{vc} if the corresponding graph G has a vertex cover with k or less vertices

Language classes with polynomial complexity

Graph vertices **Graph edges**

k ** **1 * 2 * 3 * 4 * 5** ** **1,3 * 3,4 * 2,2 * 2,5**

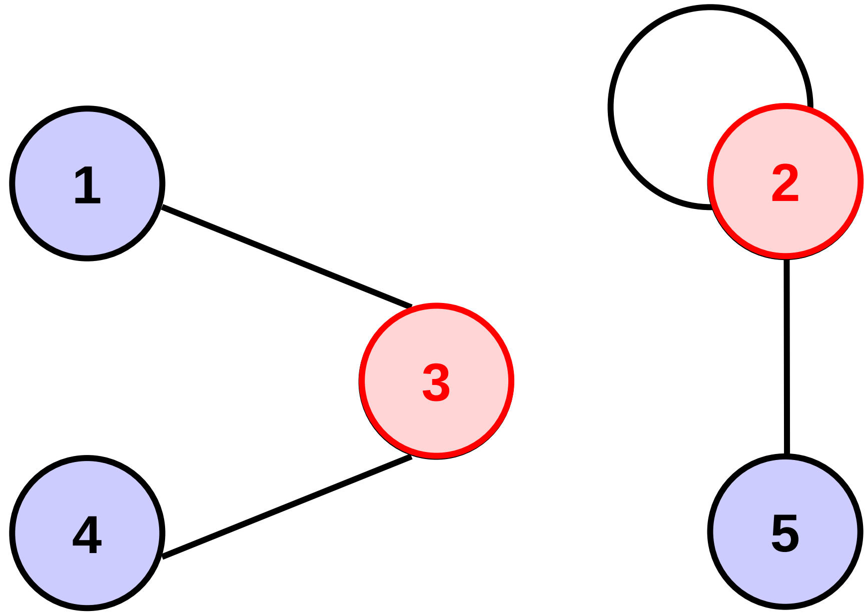


A string belongs to L_{vc} if the corresponding graph *G* has a vertex cover with *k* or less vertices

Language classes with polynomial complexity

Graph vertices **Graph edges**

k ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5

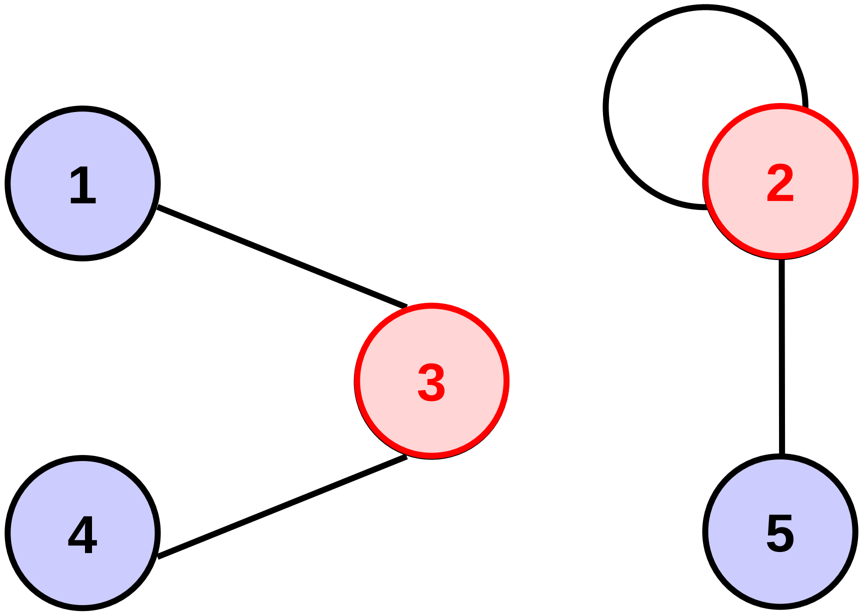


A string belongs to L_{vc} if the corresponding graph *G* has a vertex cover with *k* or less vertices

Language classes with polynomial complexity

Graph vertices **Graph edges**

k ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5



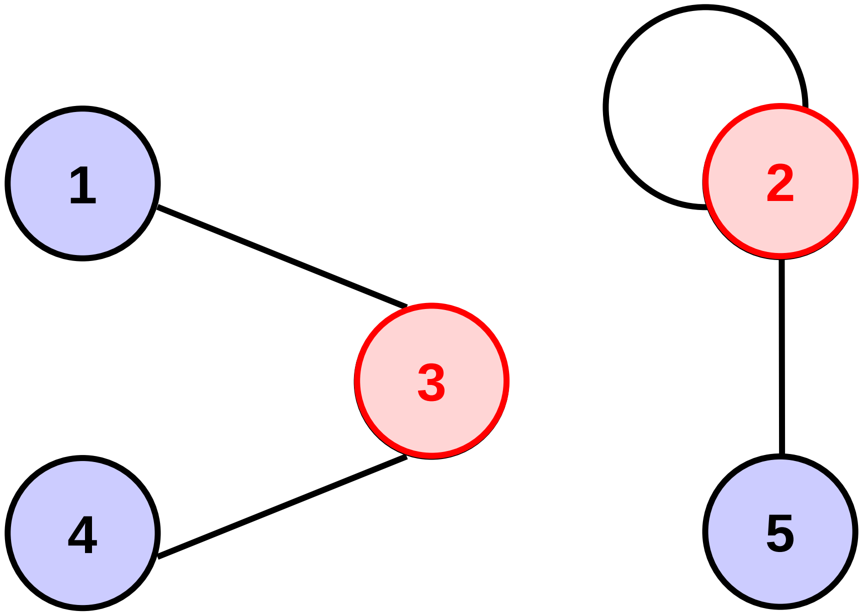
A string belongs to L_{vc} if the corresponding graph G has a vertex cover with k or less vertices

$A = \{2, 3\}$

Language classes with polynomial complexity

Graph vertices **Graph edges**

k **** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5**



A string belongs to L_{vc} if the corresponding graph *G* has a vertex cover with *k* or less vertices

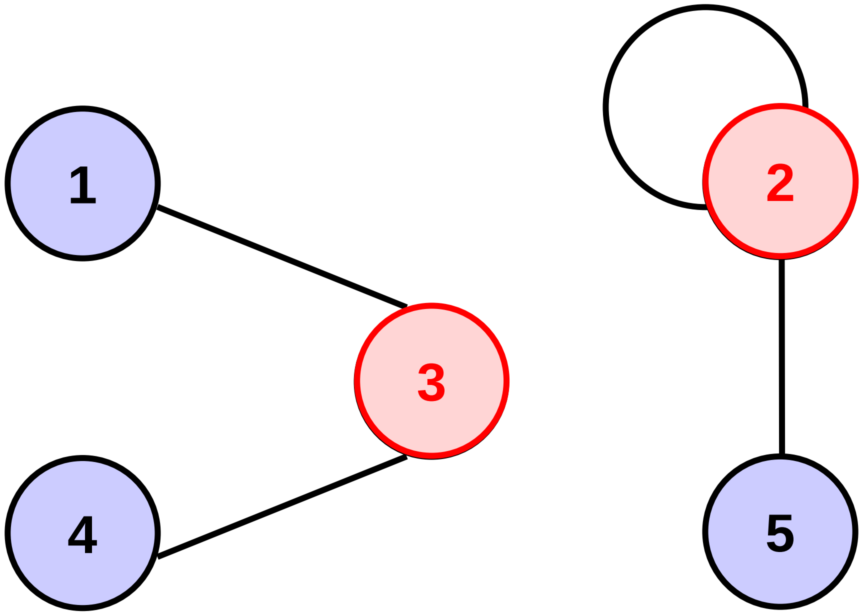
$A = \{2, 3\}$

2 ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5 $\in L_{vc}$

Language classes with polynomial complexity

Graph vertices **Graph edges**

k **** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5**



A string belongs to L_{vc} if the corresponding graph G has a vertex cover with k or less vertices

$A = \{2, 3\}$

2 ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5 $\in L_{vc}$

1 ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5 $\notin L_{vc}$

Language classes with polynomial complexity

Language classes with polynomial complexity

- $L_{vc} \in NP$

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of k vertices, and for each generated set verifies whether it is a vertex cover

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of *k* vertices, and for each generated set verifies whether it is a vertex cover

- **Difference between the complexity of languages in classes *NP* and *P***

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of *k* vertices, and for each generated set verifies whether it is a vertex cover

- **Difference between the complexity of languages in classes *NP* and *P***
  - Harder to verify

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of k vertices, and for each generated set verifies whether it is a vertex cover

- Difference between the complexity of languages in classes *NP* and *P*
  - Harder to verify
    - Is it is possible to give values 0 or 1 to the variables of the logical expression so that it evaluates to 1?

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of k vertices, and for each generated set verifies whether it is a vertex cover

- Difference between the complexity of languages in classes *NP* and *P*
  - Harder to verify
    - Is it is possible to give values 0 or 1 to the variables of the logical expression so that it evaluates to 1?
  - Easier to verify

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of *k* vertices, and for each generated set verifies whether it is a vertex cover

- Difference between the complexity of languages in classes *NP* and *P*
  - Harder to verify
    - Is it is possible to give values 0 or 1 to the variables of the logical expression so that it evaluates to 1?
  - Easier to verify
    - Does the logical expression for the given variable values evaluates to 1?

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of k vertices, and for each generated set verifies whether it is a vertex cover

- Difference between the complexity of languages in classes *NP* and *P*
  - Harder to verify
    - Is it is possible to give values 0 or 1 to the variables of the logical expression so that it evaluates to 1?
  - Easier to verify
    - Does the logical expression for the given variable values evaluates to 1?

  - Harder to verify

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of *k* vertices, and for each generated set verifies whether it is a vertex cover

- Difference between the complexity of languages in classes *NP* and *P*
  - Harder to verify
    - Is it is possible to give values 0 or 1 to the variables of the logical expression so that it evaluates to 1?
  - Easier to verify
    - Does the logical expression for the given variable values evaluates to 1?

  - Harder to verify
    - Does the given graph contain a vertex cover with *k* or less vertices?

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of *k* vertices, and for each generated set verifies whether it is a vertex cover

- Difference between the complexity of languages in classes *NP* and *P*
  - Harder to verify
    - Is it is possible to give values 0 or 1 to the variables of the logical expression so that it evaluates to 1?
  - Easier to verify
    - Does the logical expression for the given variable values evaluates to 1?

  - Harder to verify
    - Does the given graph contain a vertex cover with *k* or less vertices?
  - Easier to verify

Language classes with polynomial complexity

- $L_{vc} \in NP$
  - TM nondeterministically generates all sets of k vertices, and for each generated set verifies whether it is a vertex cover

- Difference between the complexity of languages in classes *NP* and *P*
  - Harder to verify
    - Is it is possible to give values 0 or 1 to the variables of the logical expression so that it evaluates to 1?
  - Easier to verify
    - Does the logical expression for the given variable values evaluates to 1?

  - Harder to verify
    - Does the given graph contain a vertex cover with k or less vertices?
  - Easier to verify
    - Is the given set of vertices a vertex cover of the given graph?

Language classes with polynomial complexity

- **Class of polynomial time complexity *P***

$$P = \bigcup_{i \geq 1} \mathrm{DTIME}(\, n^i \,)$$

- **Class of nondeterministic polynomial time complexity *NP***

$$NP = \bigcup_{i \geq 1} \mathrm{NTIME}(\, n^i \,)$$

Language classes with polynomial complexity

- **Class of polynomial time complexity *P***

$$P = \bigcup_{i \geq 1} \text{DTIME}(\, n^i \,)$$

- **Class of nondeterministic polynomial time complexity *NP***

$$NP = \bigcup_{i \geq 1} \text{NTIME}(\, n^i \,)$$

- **Classes of languages with respect to space complexity**

Language classes with polynomial complexity

- Class of polynomial time complexity ***P***

$$P = \bigcup_{i \geq 1} \mathrm{DTIME}(n^i)$$

- Class of nondeterministic polynomial time complexity ***NP***

$$NP = \bigcup_{i \geq 1} \mathrm{NTIME}(n^i)$$

- Classes of languages with respect to space complexity

$$\mathrm{PSPACE} = \bigcup_{i \geq 1} \mathrm{DSPACE}(n^i)$$

Language classes with polynomial complexity

- **Class of polynomial time complexity *P***

$$P = \bigcup_{i \geq 1} \mathrm{DTIME}(n^i)$$

- **Class of nondeterministic polynomial time complexity *NP***

$$NP = \bigcup_{i \geq 1} \mathrm{NTIME}(n^i)$$

- **Classes of languages with respect to space complexity**

$$\mathrm{PSPACE} = \bigcup_{i \geq 1} \mathrm{DSPACE}(n^i)$$

$$\mathrm{NPSPACE} = \bigcup_{i \geq 1} \mathrm{NSPACE}(n^i)$$

Language classes with polynomial complexity

Language classes with polynomial complexity

$$\textbf{NSPACE}\,(\, f(n) \,) \subseteq \textbf{DSPACE}\,(\, f(n)^2 \,)$$

Language classes with polynomial complexity

$$\mathbf{NSPACE}\ (\ f(n)\ ) \subseteq \mathbf{DSPACE}\ (\ f(n)^2\ )$$

We put $f(n) = n^i$

Language classes with polynomial complexity

NSPACE ($f(n)$) $\subseteq$ DSPACE ($f(n)^2$)

We put $f(n) = n^i$

NSPACE (n^i) $\subseteq$ DSPACE (n^{2i})

Language classes with polynomial complexity

$$\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f(n)^2)$$

We put $f(n) = n^i$

$$\text{NSPACE}(n^i) \subseteq \text{DSPACE}(n^{2i})$$

$$\text{PSPACE} = \bigcup_{i \geq 1} \text{DSPACE}(n^i)$$

Language classes with polynomial complexity

$$\text{NSPACE}(\, f(n) \,) \subseteq \text{DSPACE}(\, f(n)^2 \,)$$

We put $f(n) = n^i$

$$\text{NSPACE}(\, n^i \,) \subseteq \text{DSPACE}(\, n^{2i} \,)$$

$$\text{PSPACE} = \bigcup_{i \geq 1} \text{DSPACE}(\, n^i \,)$$

$$\text{NPSPACE} = \bigcup_{i \geq 1} \text{NSPACE}(\, n^i \,)$$

Language classes with polynomial complexity

$$\text{NSPACE}(\ f(n)\ ) \subseteq \text{DSPACE}(\ f(n)^2\ )$$

We put $f(n) = n^i$

$$\text{NSPACE}(\ n^i\ ) \subseteq \text{DSPACE}(\ n^{2i}\ )$$

$$\text{PSPACE} = \bigcup_{i \geq 1} \text{DSPACE}(\ n^i\ )$$

$$\text{NPSPACE} = \bigcup_{i \geq 1} \text{NSPACE}(\ n^i\ )$$

PSPACE = NPSPACE

Language classes with polynomial complexity

$$\mathrm{NSPACE}(\, f(n) \,) \subseteq \mathrm{DSPACE}(\, f(n)^2 \,)$$

We put $f(n) = n^i$

$$\mathrm{NSPACE}(\, n^i \,) \subseteq \mathrm{DSPACE}(\, n^{2i} \,)$$

$$\mathrm{PSPACE} = \bigcup_{i \geq 1} \mathrm{DSPACE}(\, n^i \,)$$

$$\mathrm{NPSPACE} = \bigcup_{i \geq 1} \mathrm{NSPACE}(\, n^i \,)$$

$$P \subseteq NP \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE}$$

Language reduction

Language reduction

TM M_R

Language reduction

TM M_R

$x \in L_1 \Rightarrow y = R(x) \in L_2$

Language reduction

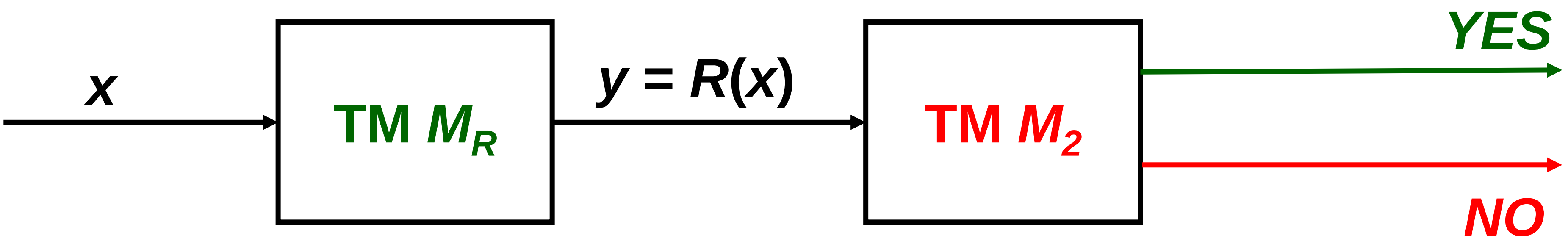
TM M_R

$x \in L_1 \Rightarrow y = R(x) \in L_2$

Language reduction

TM M_R

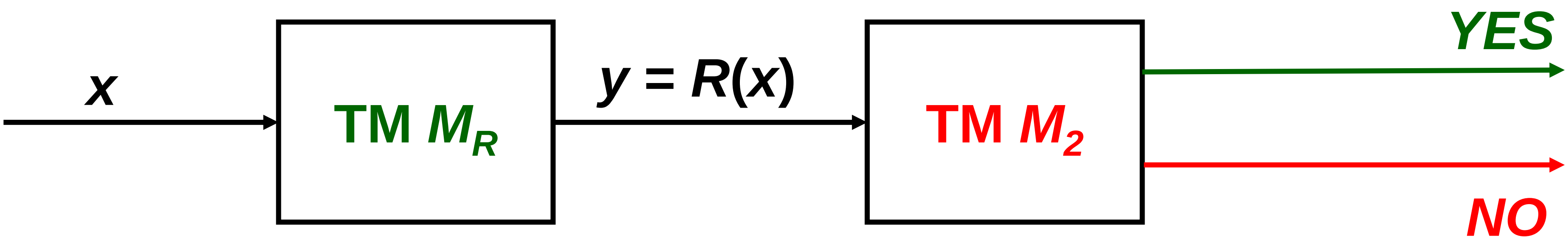
$x \in L_1 \Rightarrow y = R(x) \in L_2$



Language reduction

TM M_R

$x \in L_1 \Rightarrow y = R(x) \in L_2$

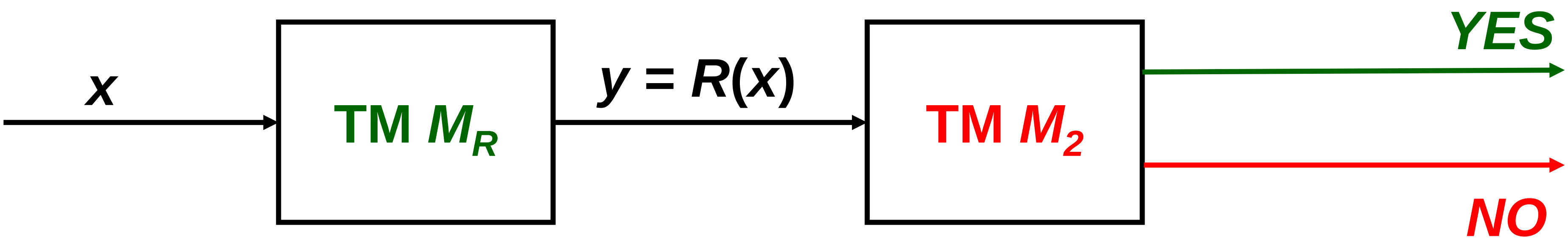


TM M_2

Language reduction

TM M_R

$x \in L_1 \Rightarrow y = R(x) \in L_2$



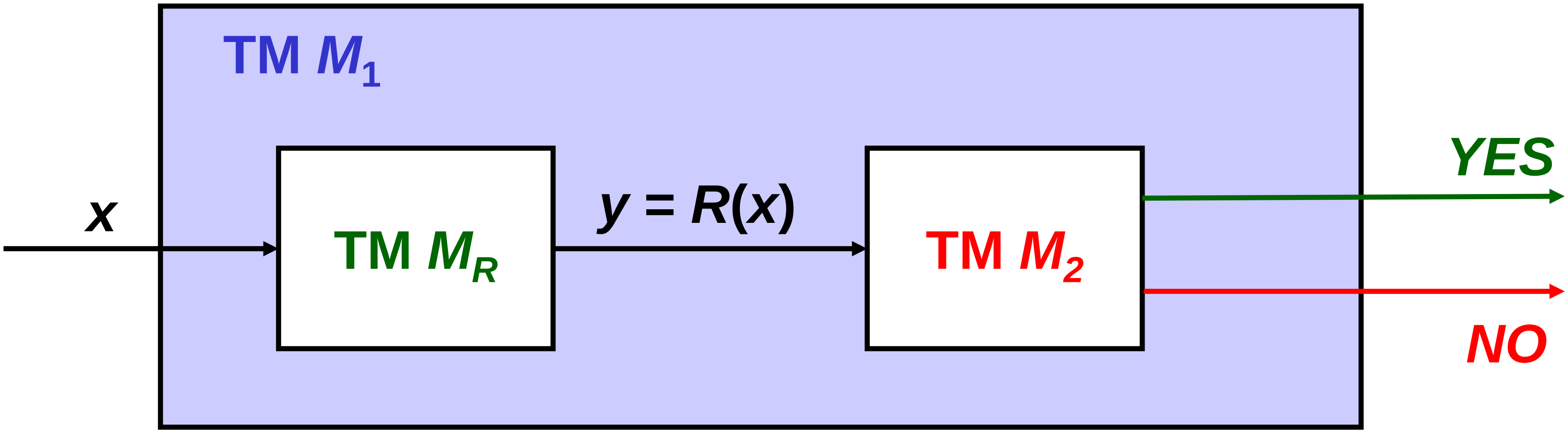
TM M_2

$L(M_2) = L_2$

Language reduction

TM M_R

$x \in L_1 \Rightarrow y = R(x) \in L_2$



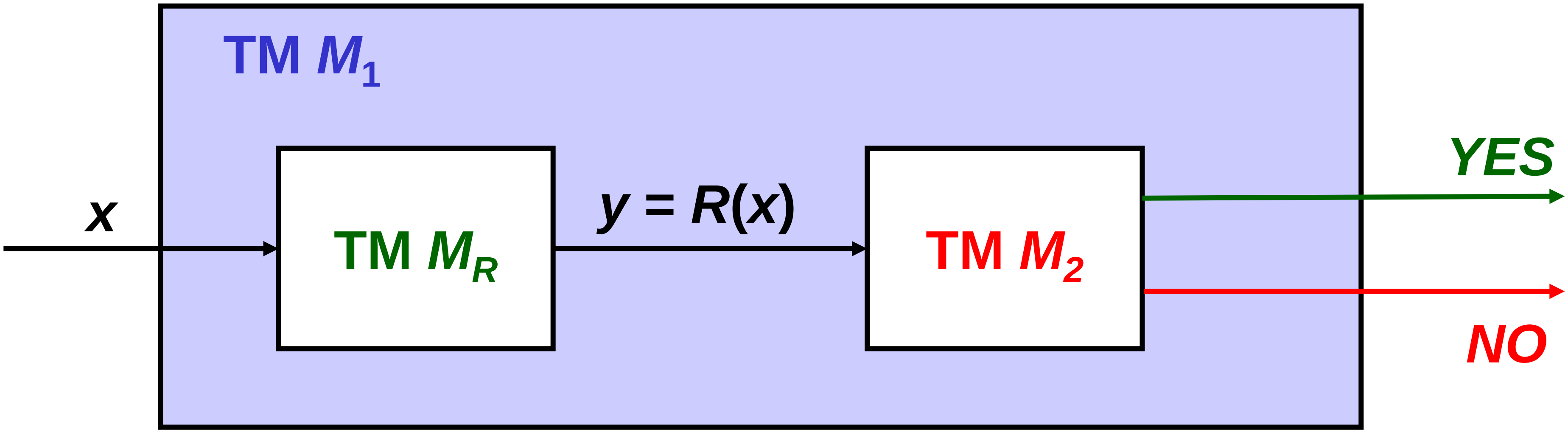
TM M_2

$L(M_2) = L_2$

Language reduction

TM M_R

$x \in L_1 \Rightarrow y = R(x) \in L_2$



TM M_1

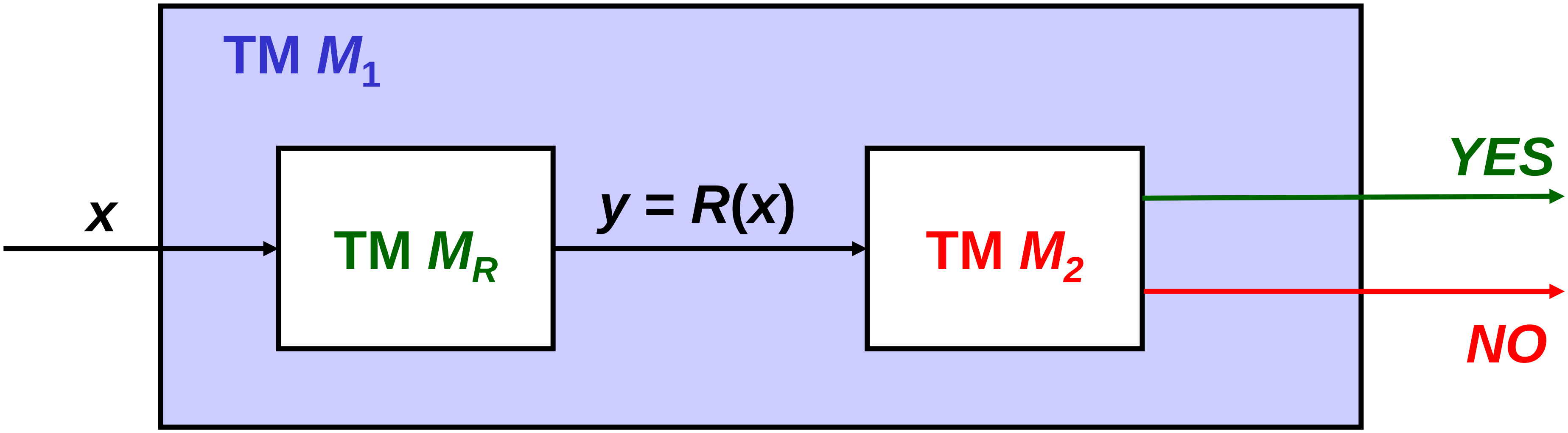
TM M_2

$L(M_2) = L_2$

Language reduction

TM M_R

$x \in L_1 \Rightarrow y = R(x) \in L_2$



TM M_1

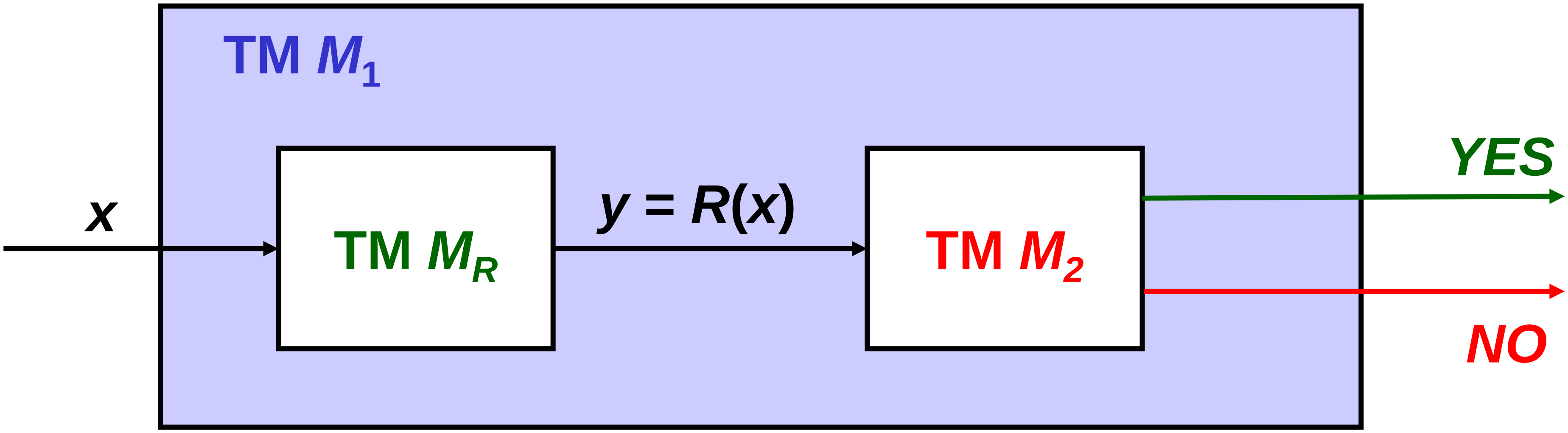
$L(M_1) = L_1$

TM M_2

$L(M_2) = L_2$

Language reduction

TM M_R
$x \in L_1 \Rightarrow y = R(x) \in L_2$

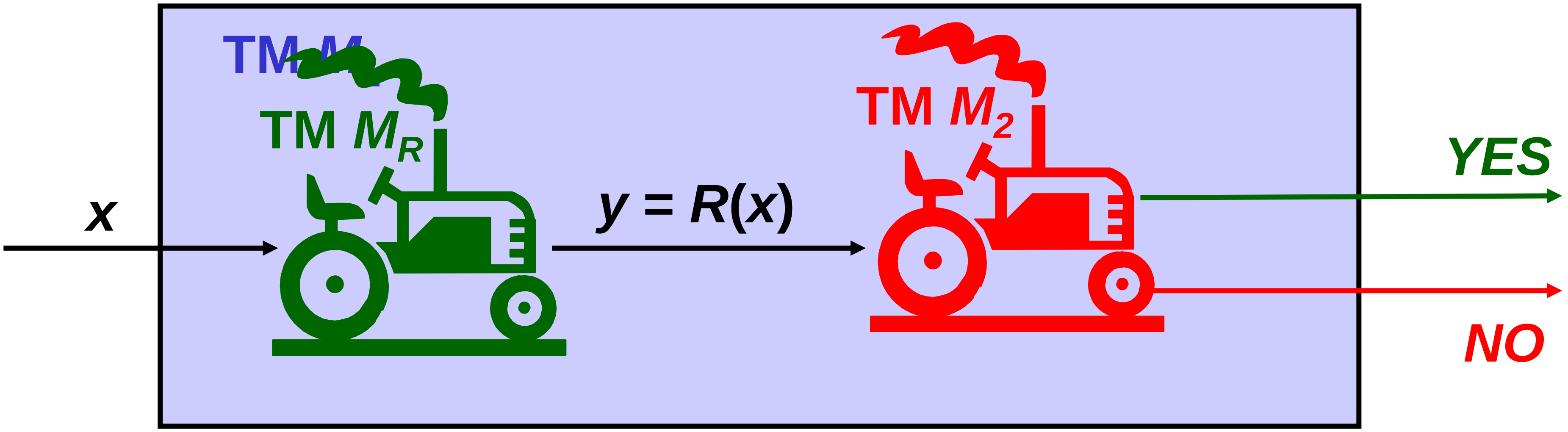


TM M_1
$L(M_1) = L_1$

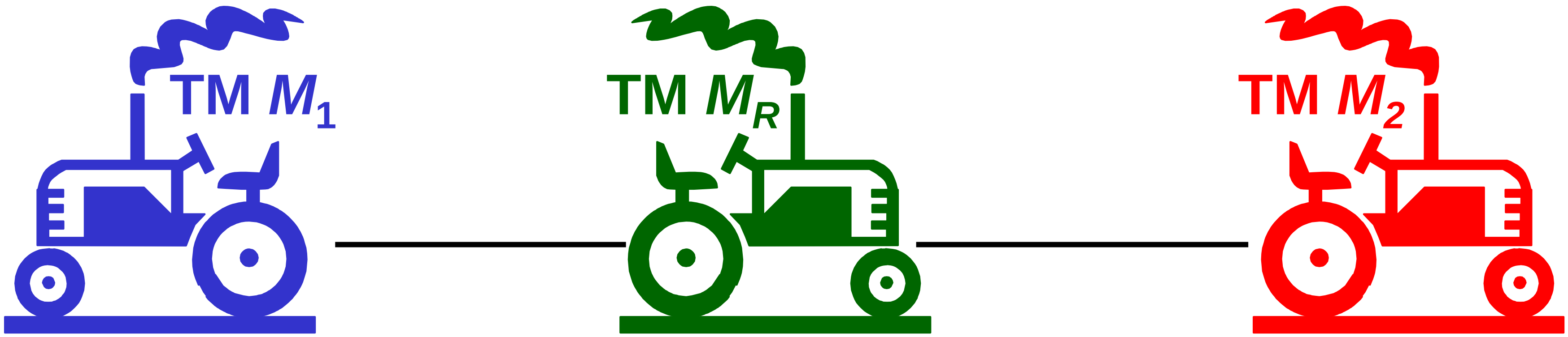
TM M_2
$L(M_2) = L_2$

Language L_1 is reduced to language L_2

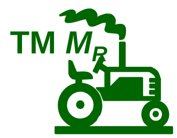
Language reduction



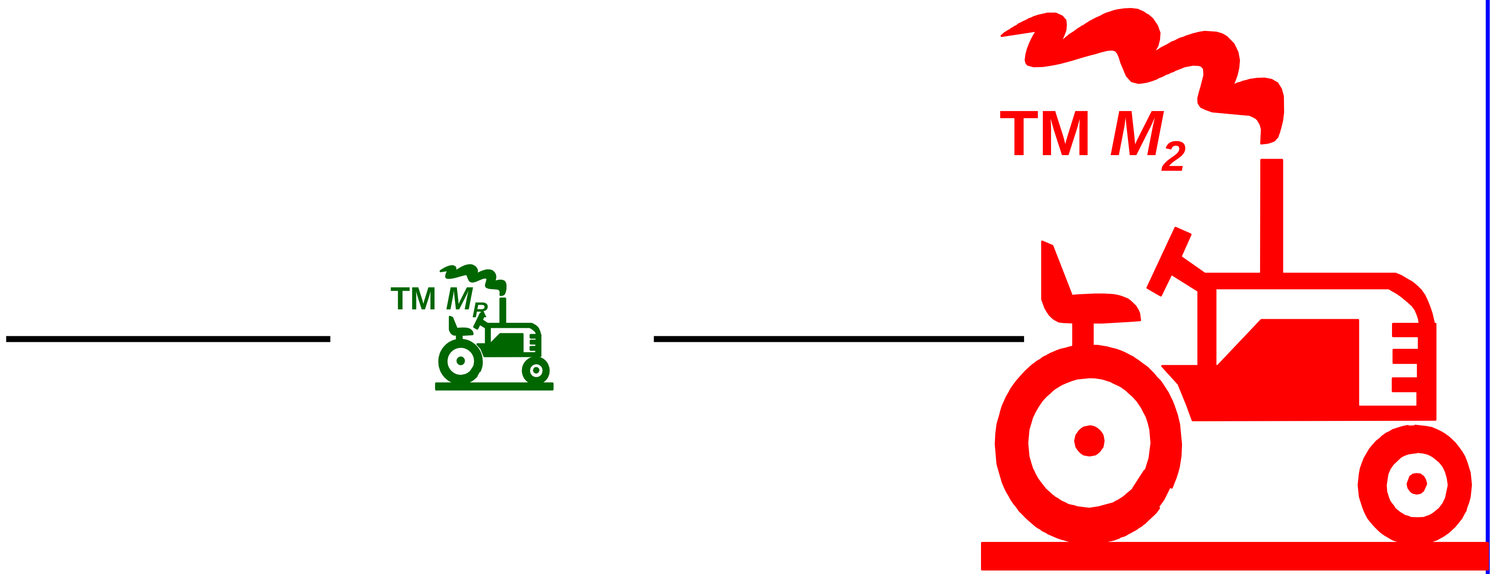
Language reduction



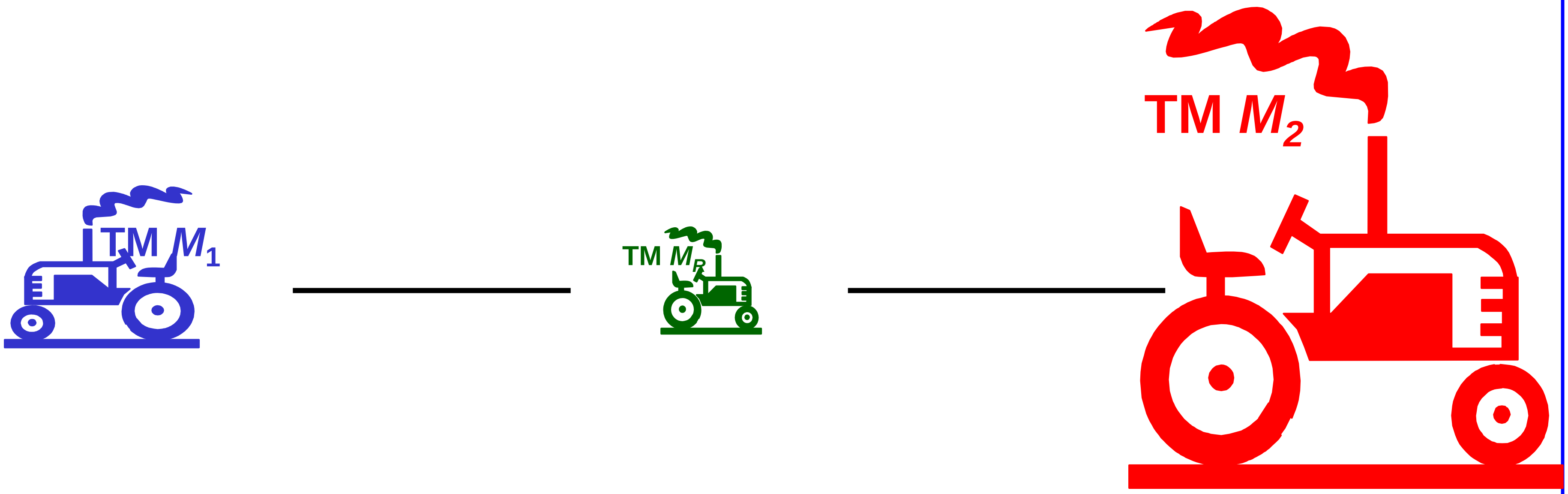
Language reduction



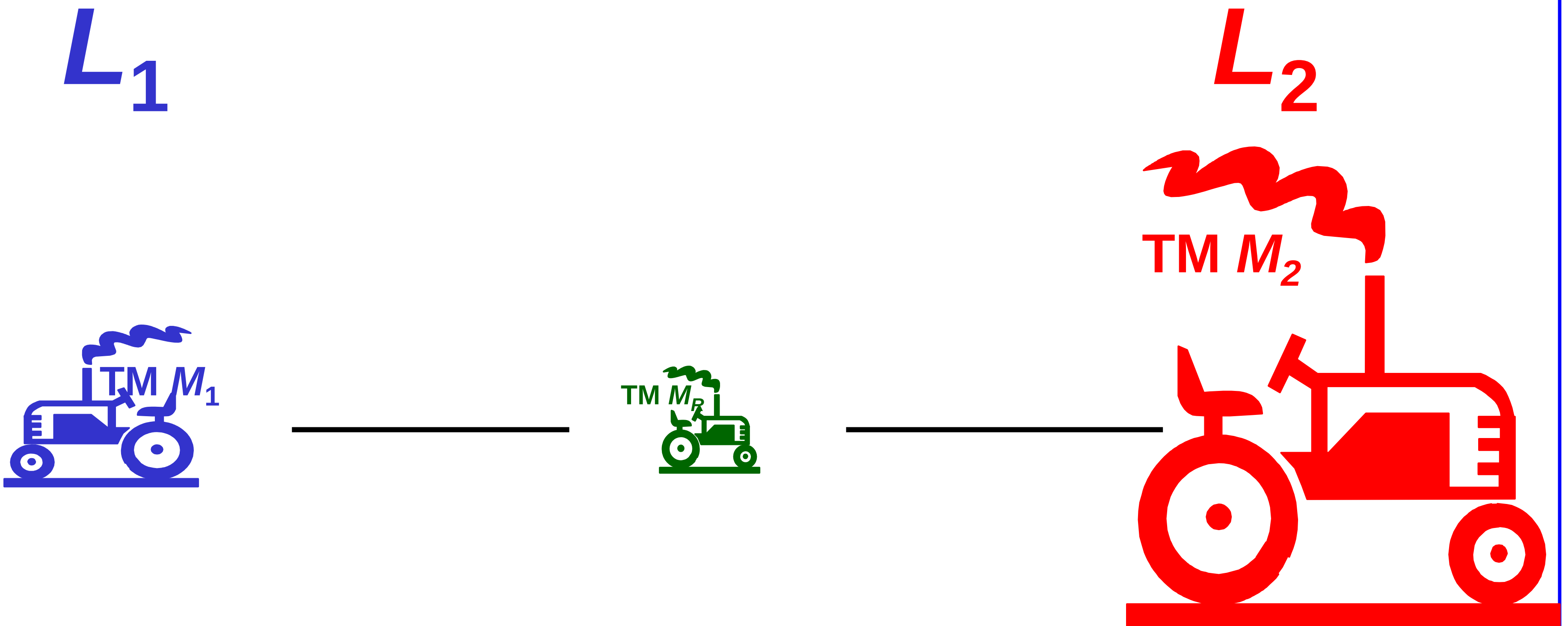
Language reduction



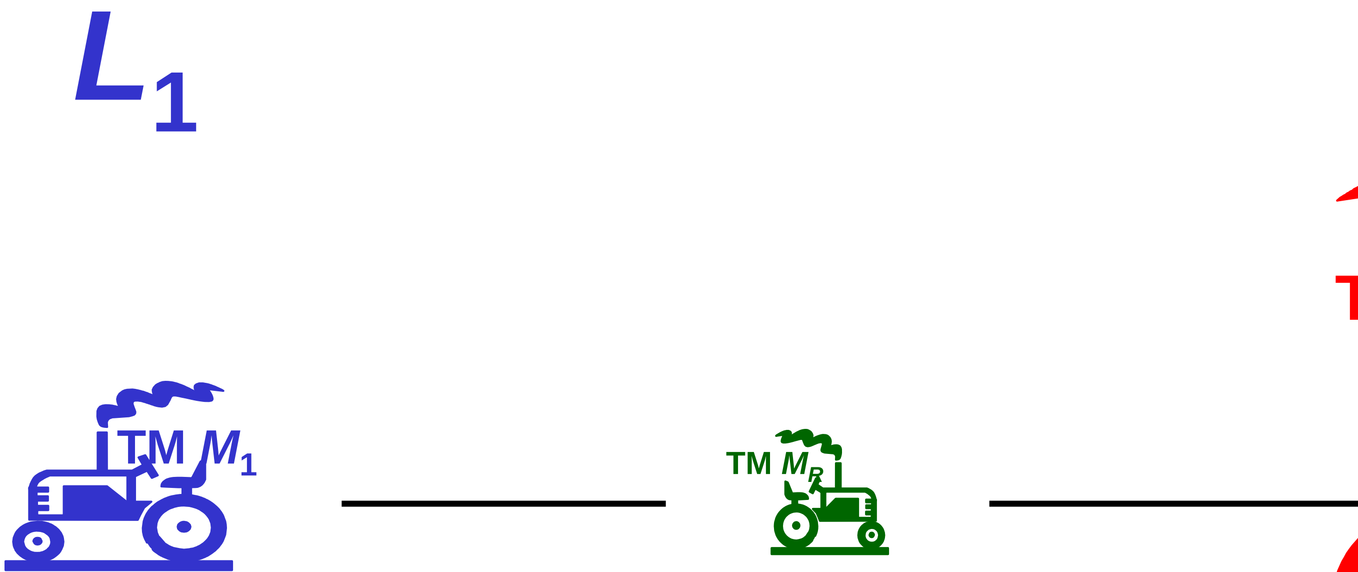
Language reduction

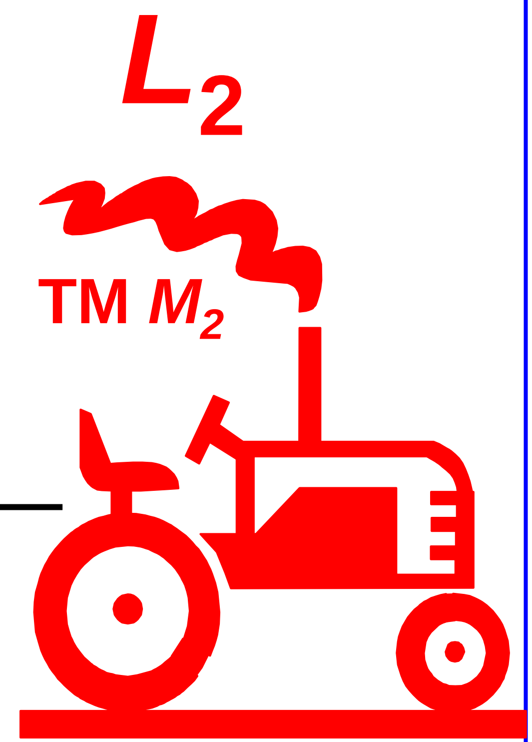


Language reduction



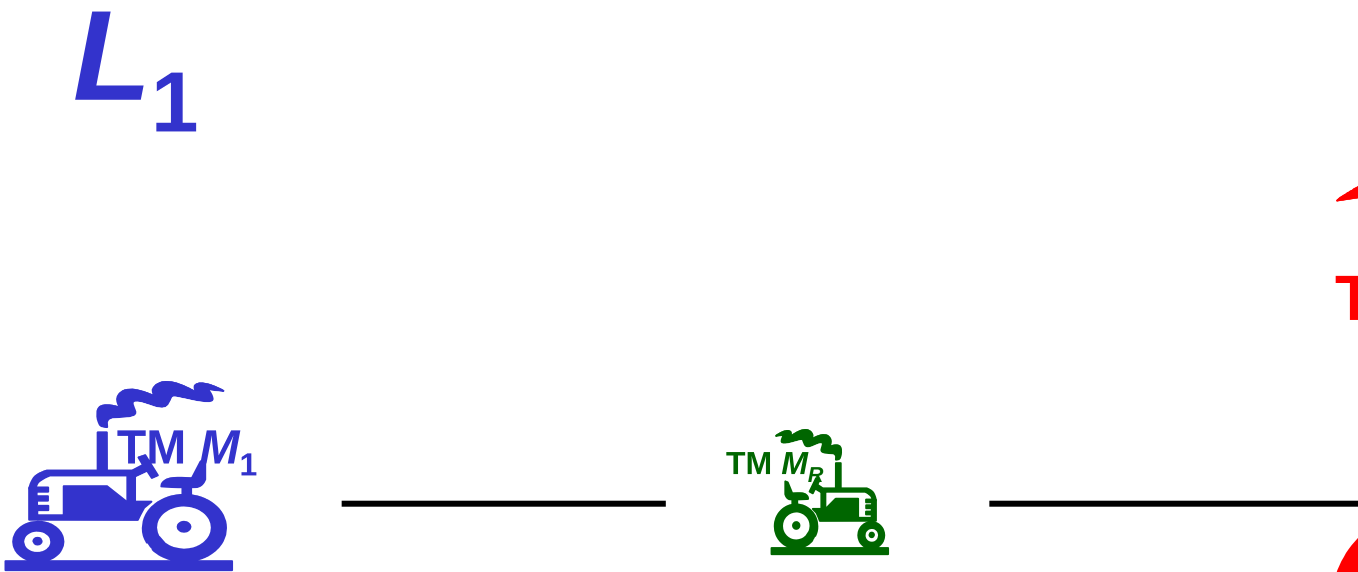
Language reduction

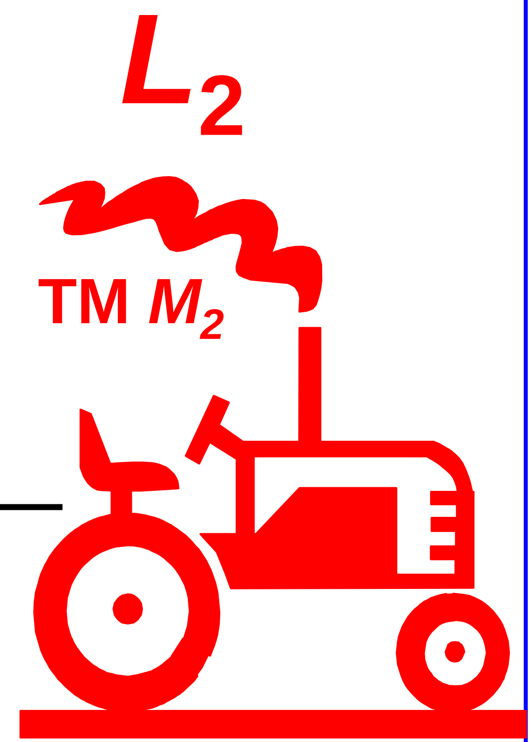




- **If complexity of TM M_R is small**

Language reduction

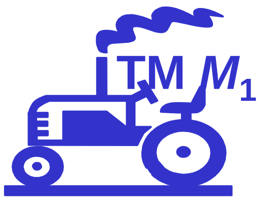


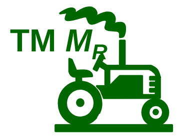


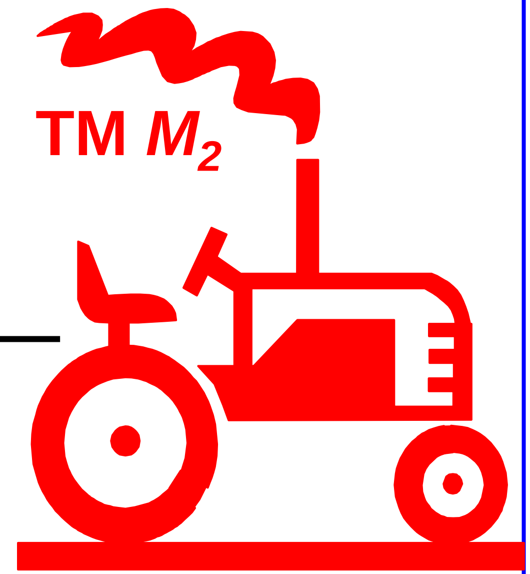
- If complexity of TM M_R is small
  - If we can reduce L_1 to L_2

Language reduction

L_1 L_2



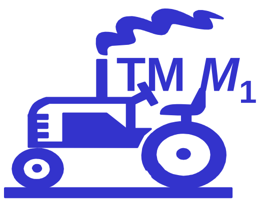


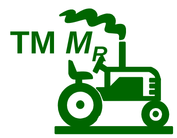


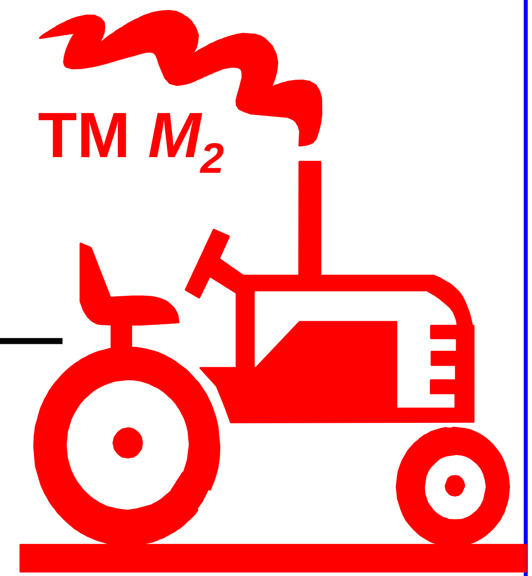
- **If complexity of TM M_R is small**
  - **If we can reduce L_1 to L_2**
  - **then language L_2 has equal or higher complexity than L_1**

Language reduction

L_1 L_2

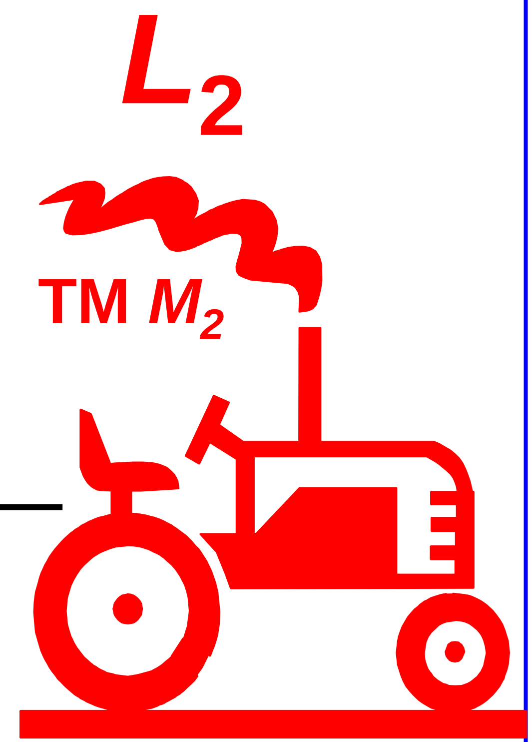






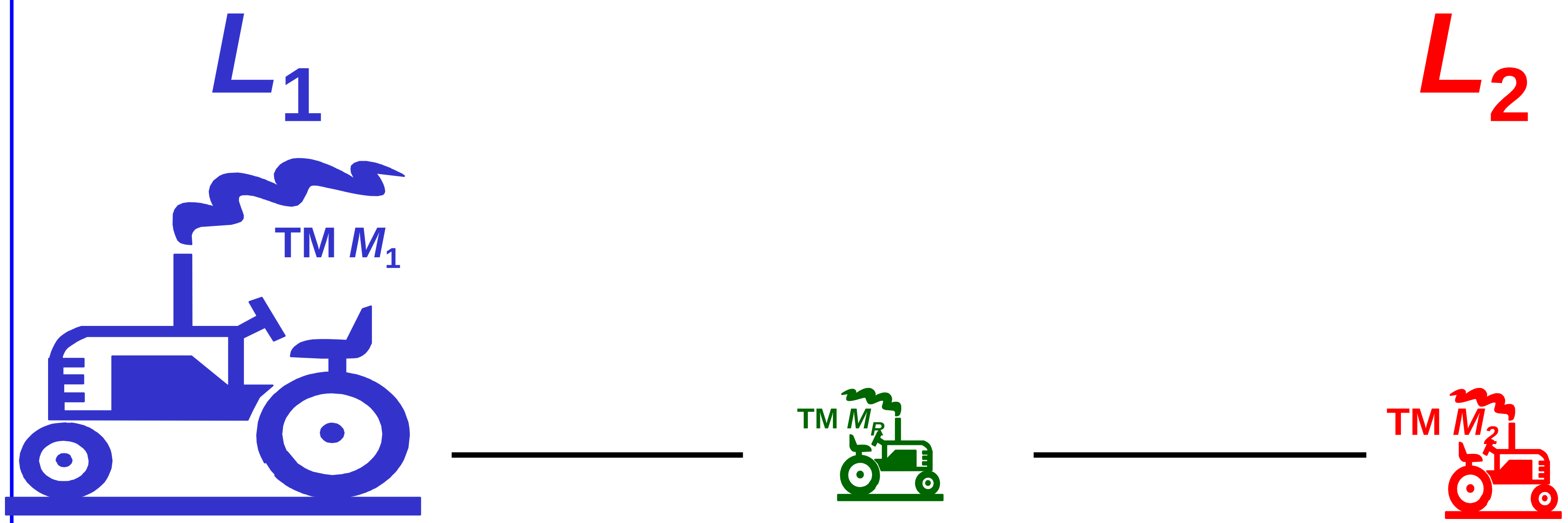
- **If complexity of TM M_R is small**
  - **If we can reduce L_1 to L_2**
  - **then language L_2 has equal or higher complexity than L_1**
  - **Example**

Language reduction

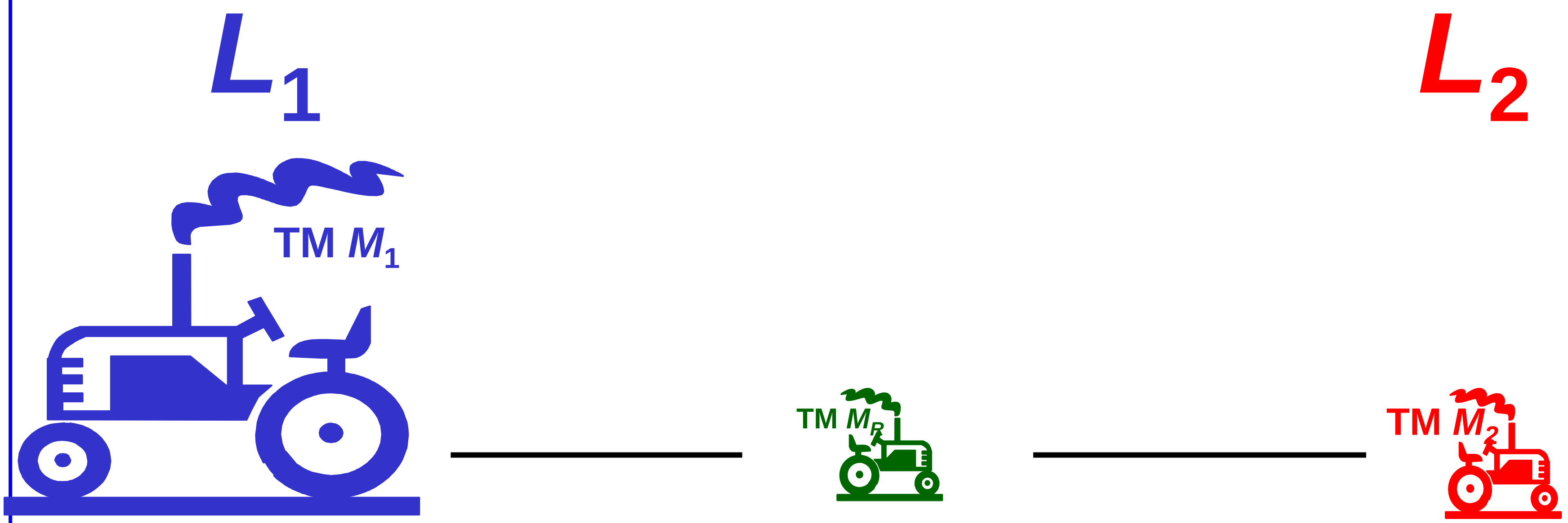


- If complexity of TM M_R is small
  - If we can reduce L_1 to L_2
  - then language L_2 has equal or higher complexity than L_1
  - Example
    - language L_1 with polynomial time complexity can be reduced in polynomial time to language L_2 with exponential or polynomial time complexity

Language reduction

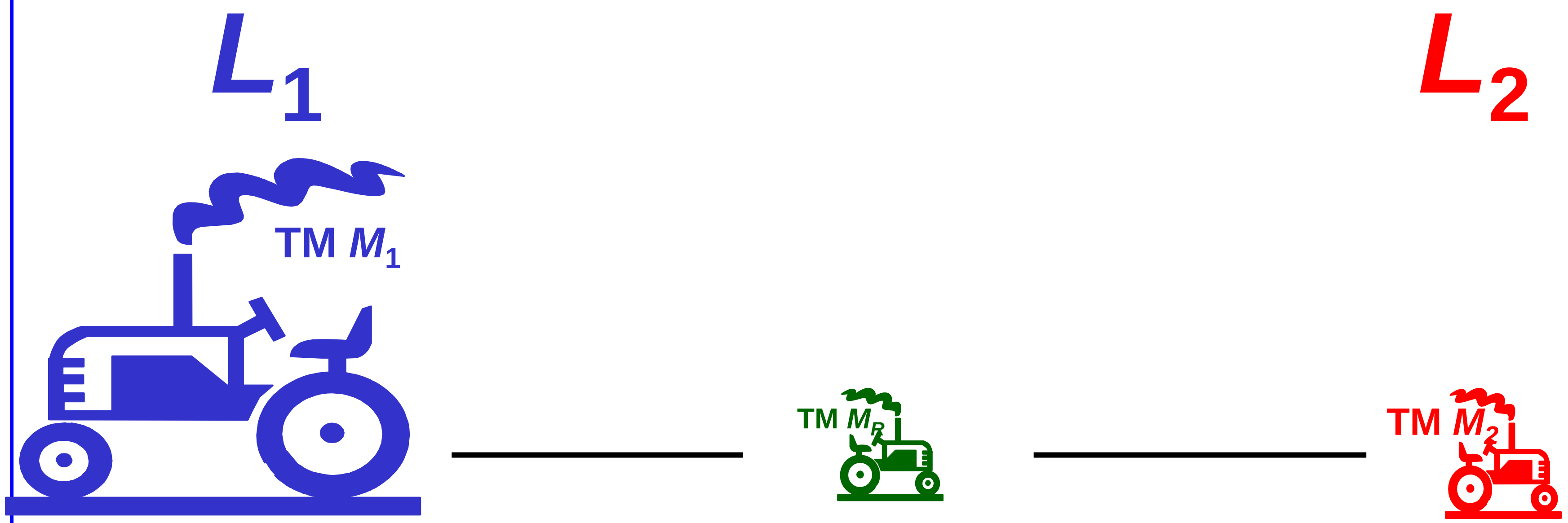


Language reduction



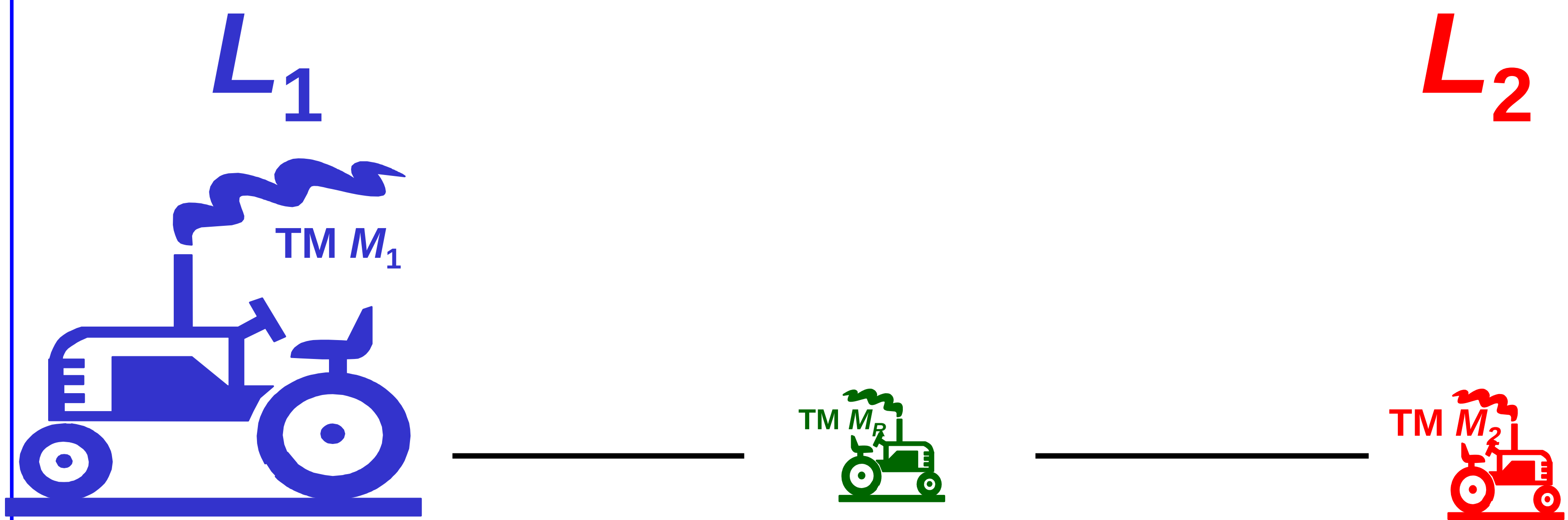
- It is not possible to

Language reduction



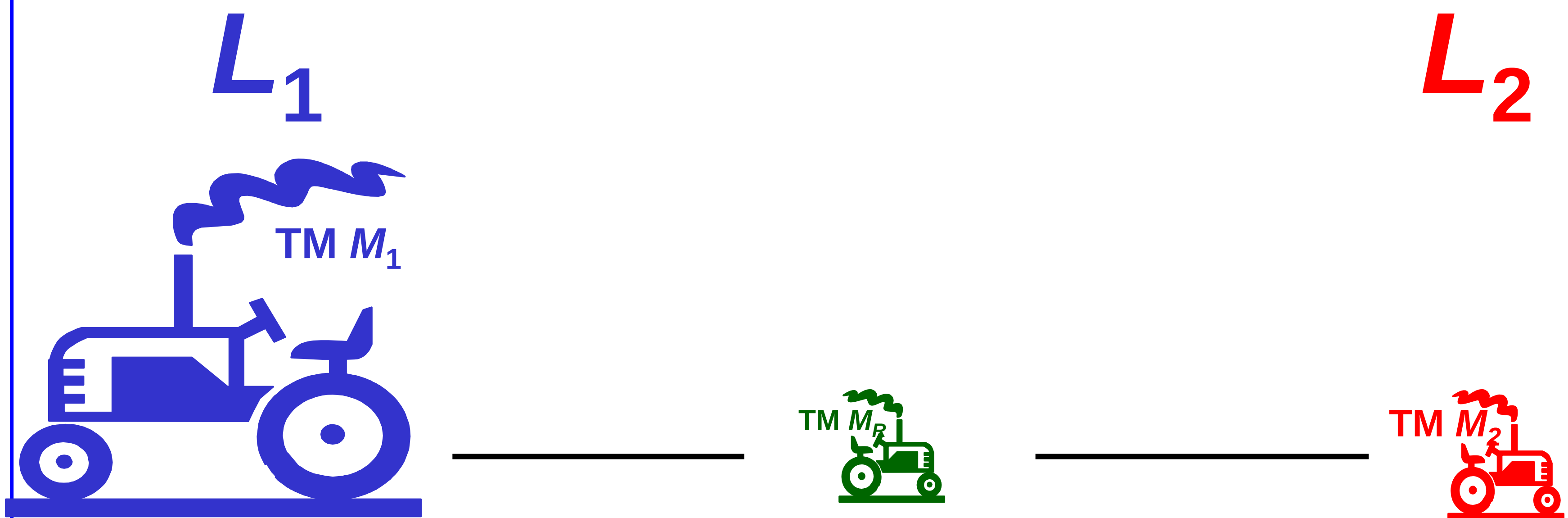
- It is not possible to
  - language L_1 with exponential time complexity

Language reduction



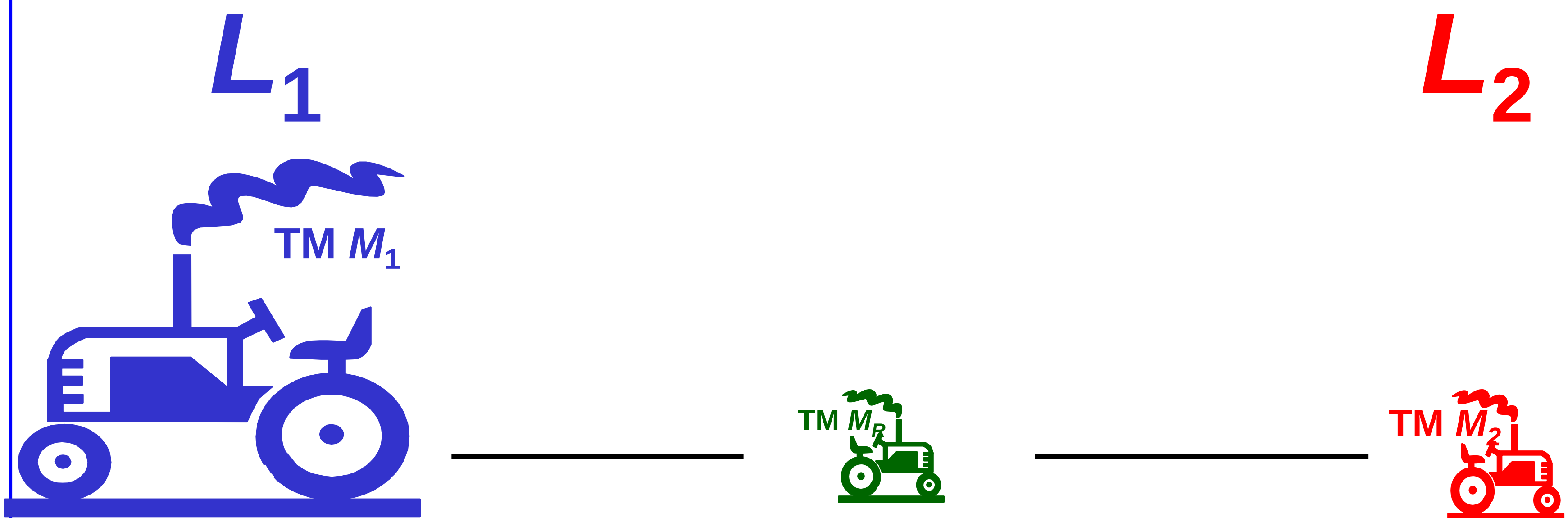
- It is not possible to
  - language L_1 with exponential time complexity
  - reduce in polynomial time to

Language reduction



- It is not possible to
  - language L_1 with exponential time complexity
  - reduce in polynomial time to
  - language L_2 with polynomial time complexity

Language reduction



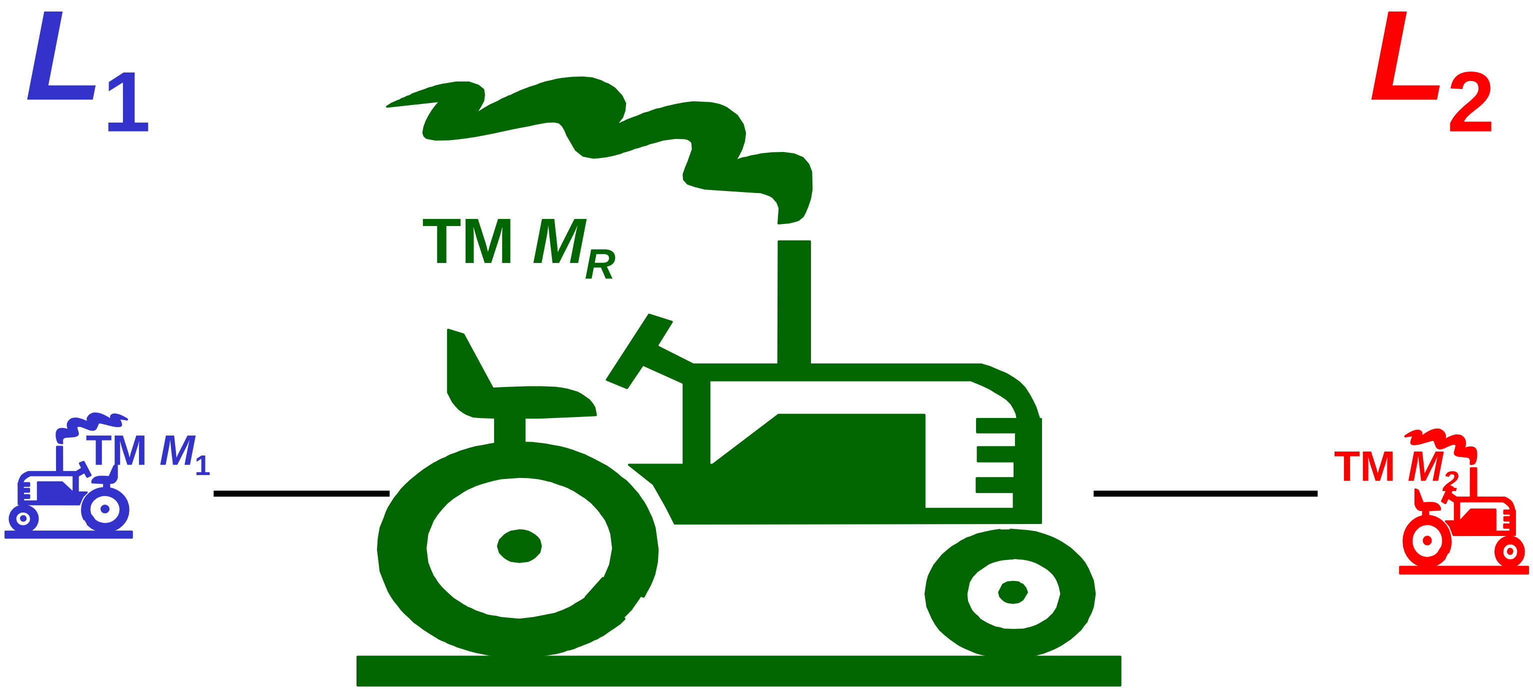
- It is not possible to
  - language L_1 with exponential time complexity
  - reduce in polynomial time to
  - language L_2 with polynomial time complexity
    - — If we can reduce language L_1 in polynomial time to language L_2 with polynomial time complexity, then language L_1 also has polynomial time complexity, not exponential

Language reduction

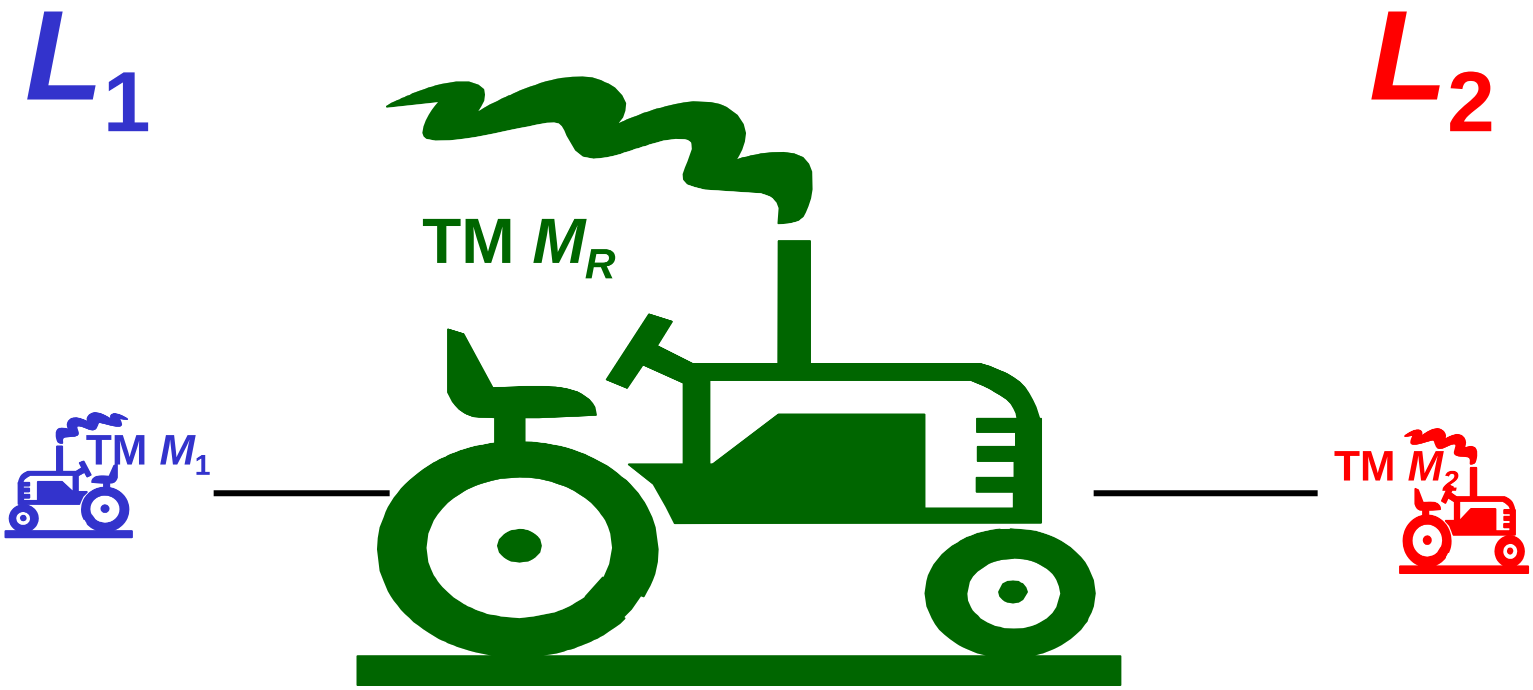
L_1 L_2

- It is not possible to
  - language L_1 with exponential time complexity
  - reduce in polynomial time to
  - language L_2 with polynomial time complexity
    - — If we can reduce language L_1 in polynomial time to language L_2 with polynomial time complexity, then language L_1 also has polynomial time complexity, not exponential

Language reduction

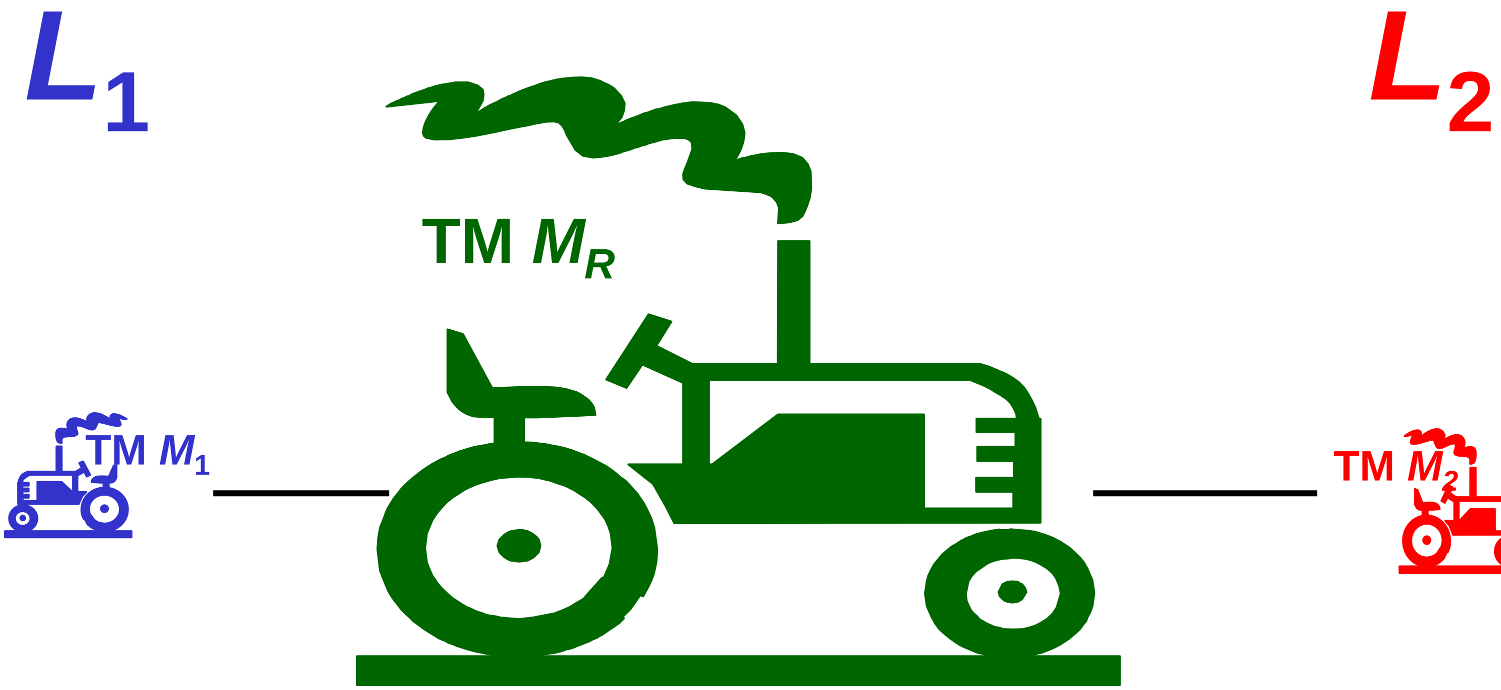


Language reduction



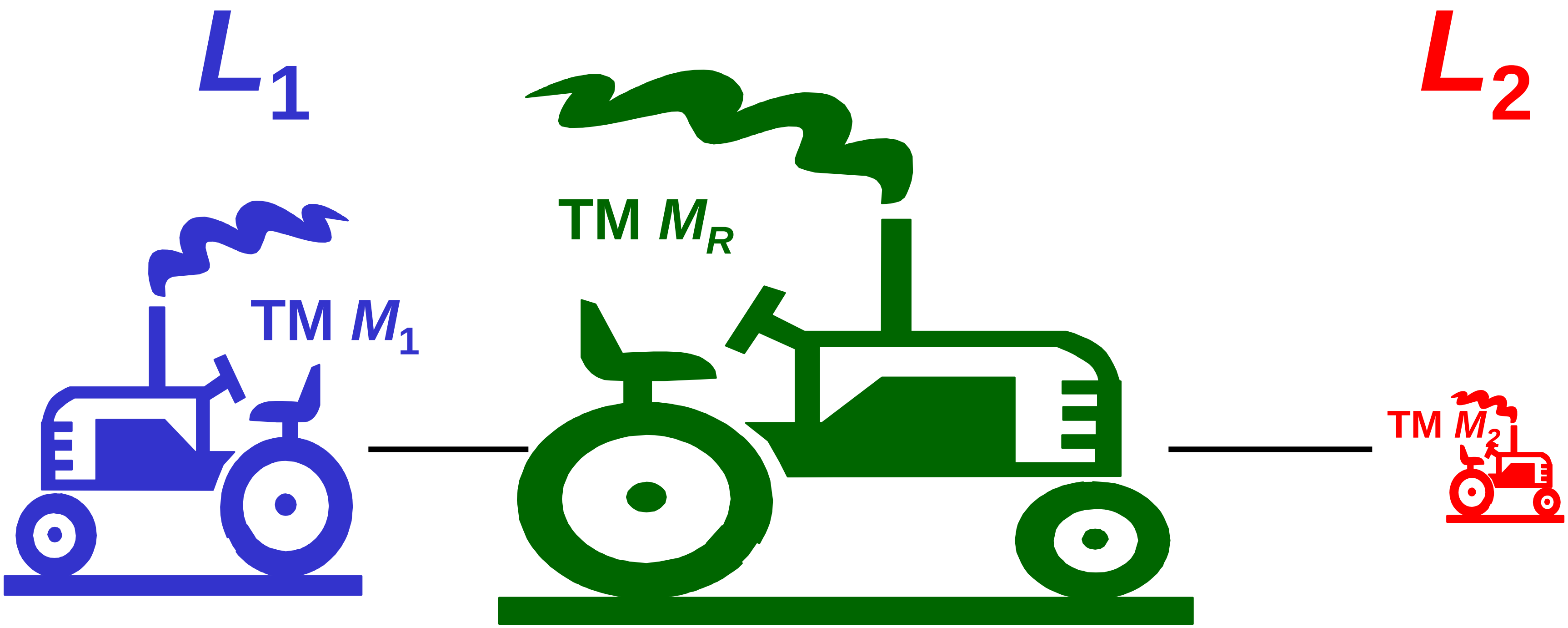
- Large complexity of TM M_R

Language reduction



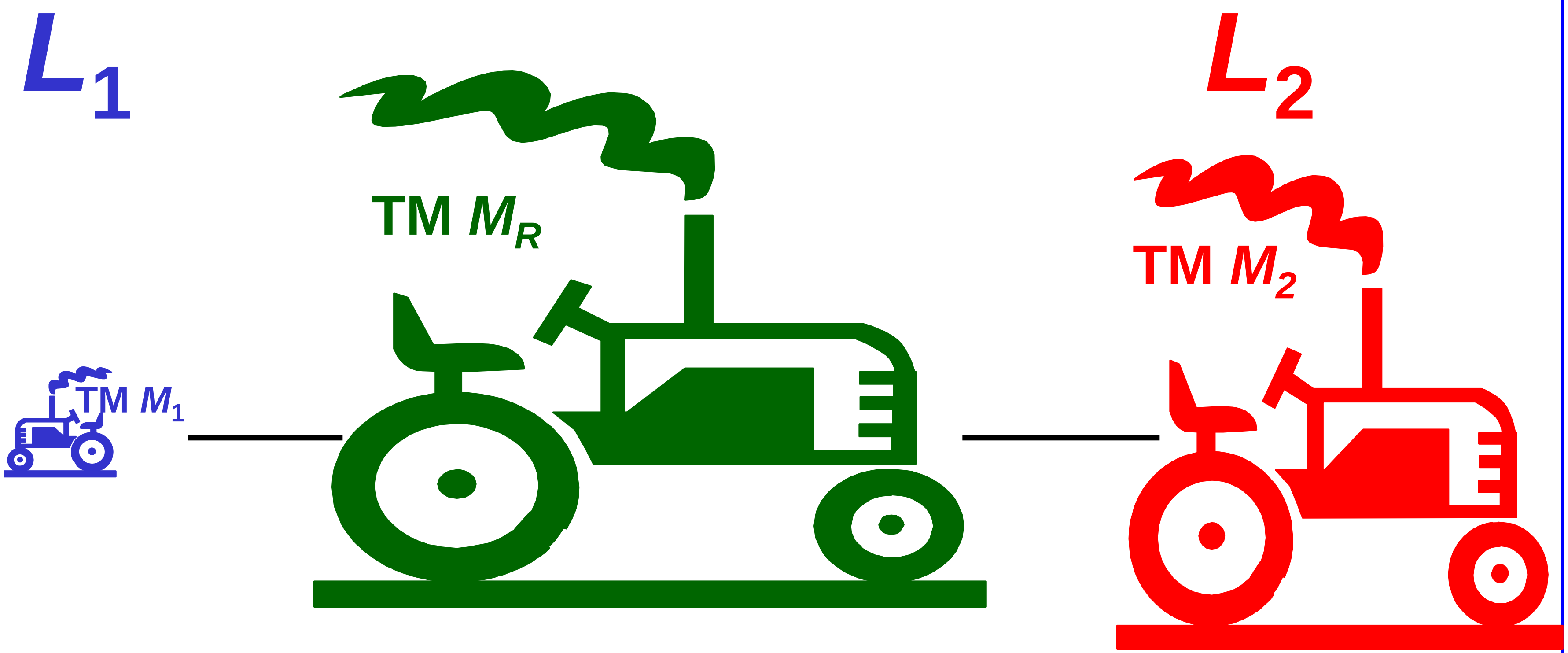
- Large complexity of TM M_R
  - Impossible to estimate complexities of L_1 and L_2

Language reduction



- Large complexity of TM M_R
  - Impossible to estimate complexities of L_1 and L_2

Language reduction



- Large complexity of TM M_R
  - Impossible to estimate complexities of L_1 and L_2

Language reduction

Language reduction

- **Efficient language reduction**

Language reduction

- **Efficient language reduction**
  - **Polynomial time complexity TM M_R**

Language reduction

- **Efficient language reduction**

  - **Polynomial time complexity TM M_R**
    - **Language L_1 is *polynomial-time reduced* to language L_2 if there is a deterministic TM M_R with polynomial time complexity which generates an output string $y=R(x)$ from language L_2 if and only if the input string x belongs to language L_1**

Language reduction

- **Efficient language reduction**

  - **Polynomial time complexity TM M_R**
    — **Language L_1 is *polynomial-time reduced* to language L_2 if there is a deterministic TM M_R with polynomial time complexity which generates an output string $y=R(x)$ from language L_2 if and only if the input string x belongs to language L_1**

  - **Logarithmic space complexity TM M_R**

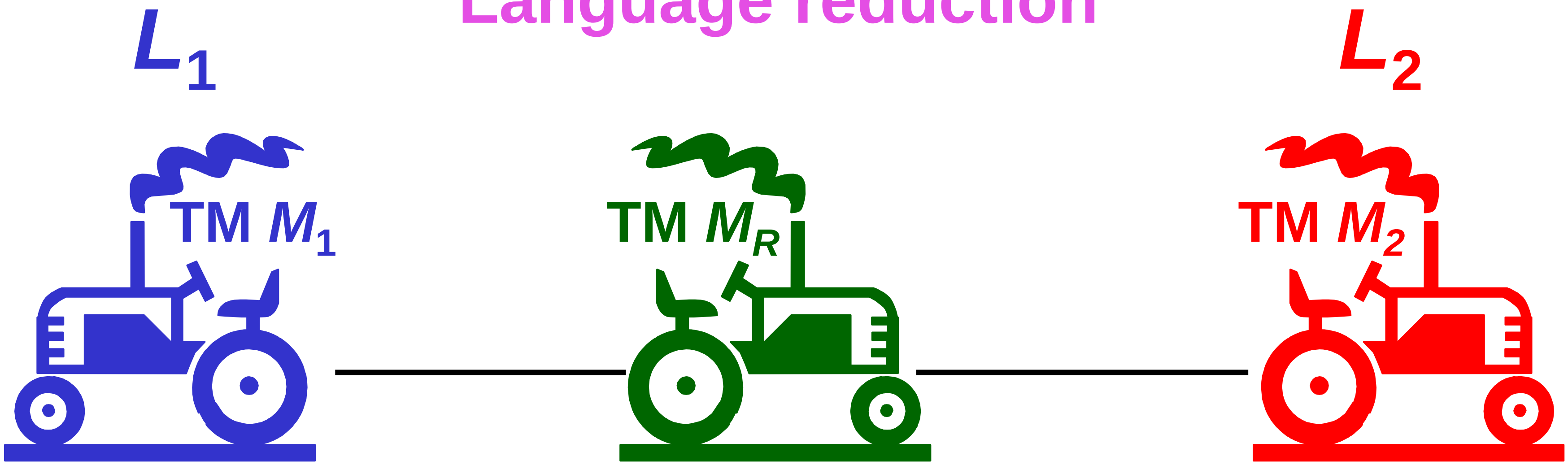
Language reduction

- **Efficient language reduction**

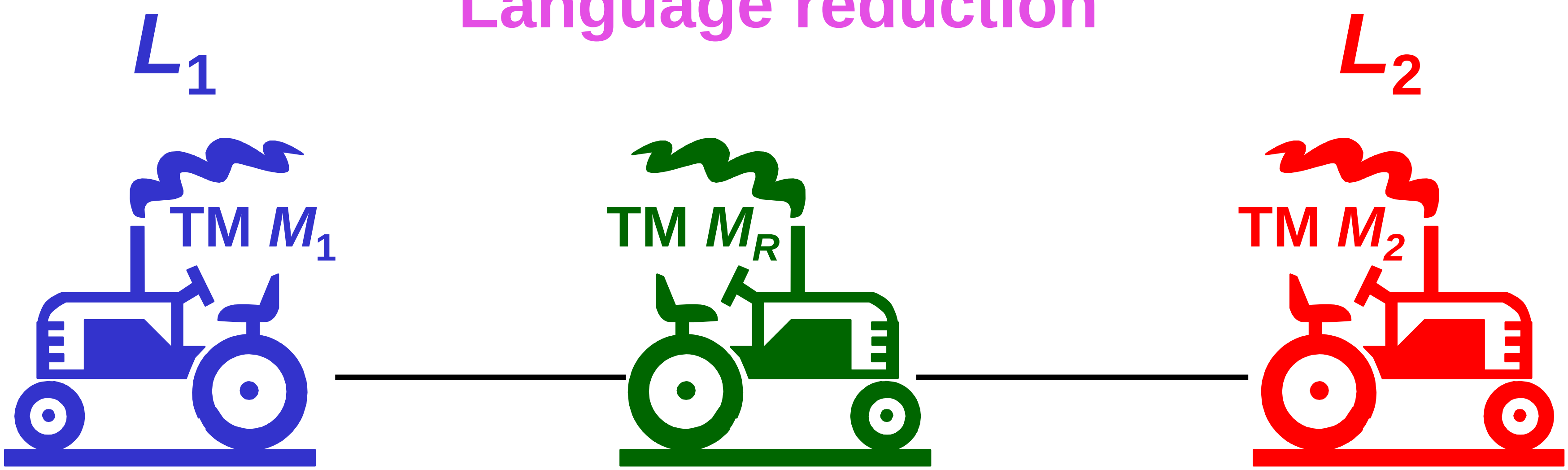
- **Polynomial time complexity TM M_R**
  - Language L_1 is *polynomial-time reduced* to language L_2 if there is a deterministic TM M_R with polynomial time complexity which generates an output string $y=R(x)$ from language L_2 if and only if the input string x belongs to language L_1

- **Logarithmic space complexity TM M_R**
  - Language L_1 is *log-space reduced* to language L_2 if there is a deterministic TM M_R with logarithmic space complexity which generates an output string $y=R(x)$ from language L_2 if and only if the input string x belongs to language L_1

Language reduction

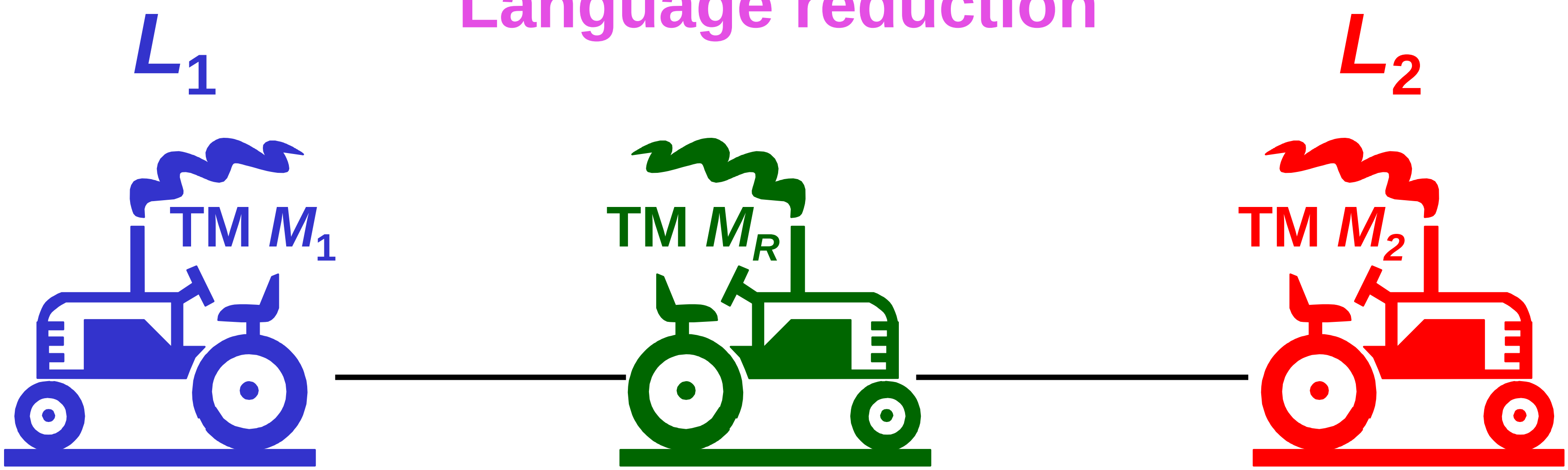


Language reduction



- **Language L_1** is **polynomial-time** reduced to **language L_2**

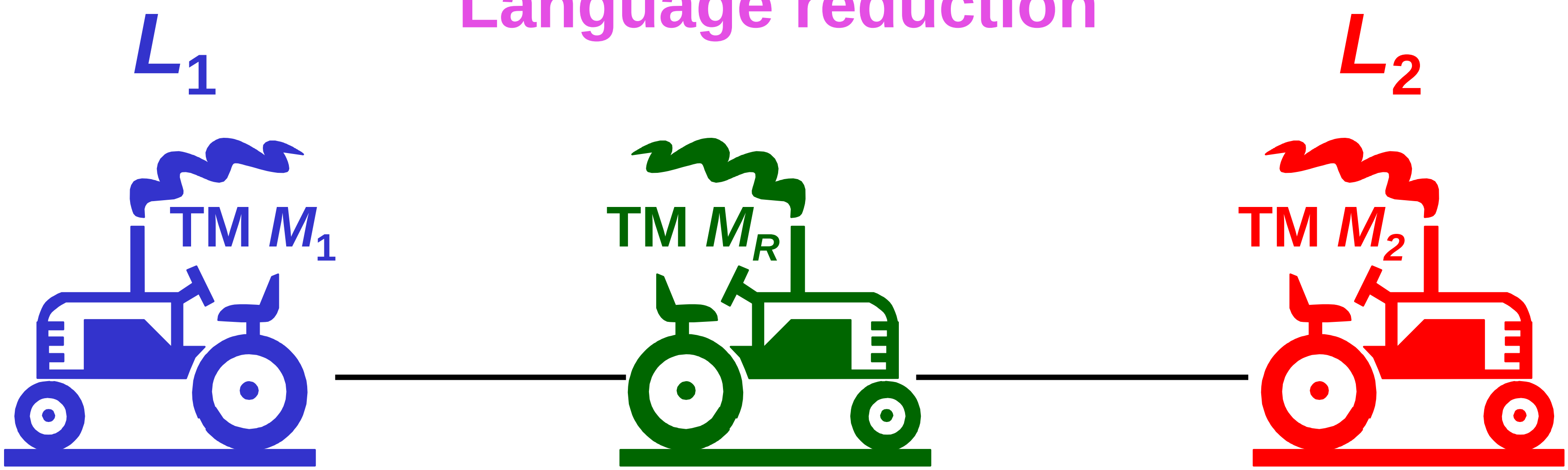
Language reduction



- **Language L_1** is **polynomial-time** reduced to **language L_2**

  a) If **language L_2** is in class ***P***, then **language L_1** is also in class ***P***

Language reduction

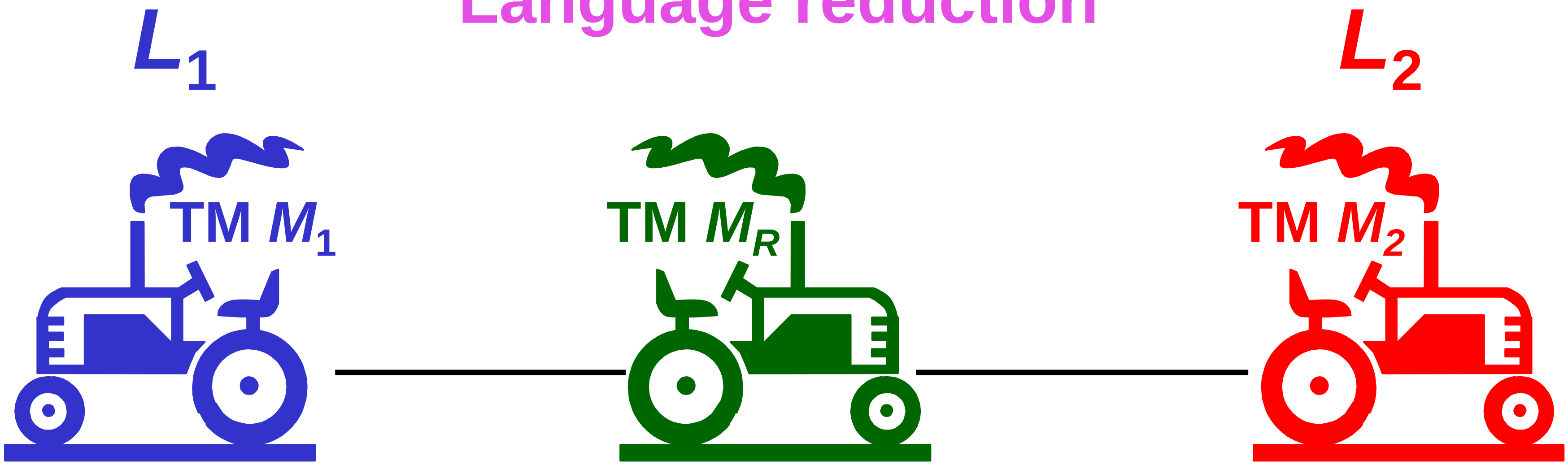


- **Language L_1** is **polynomial-time** reduced to **language L_2**

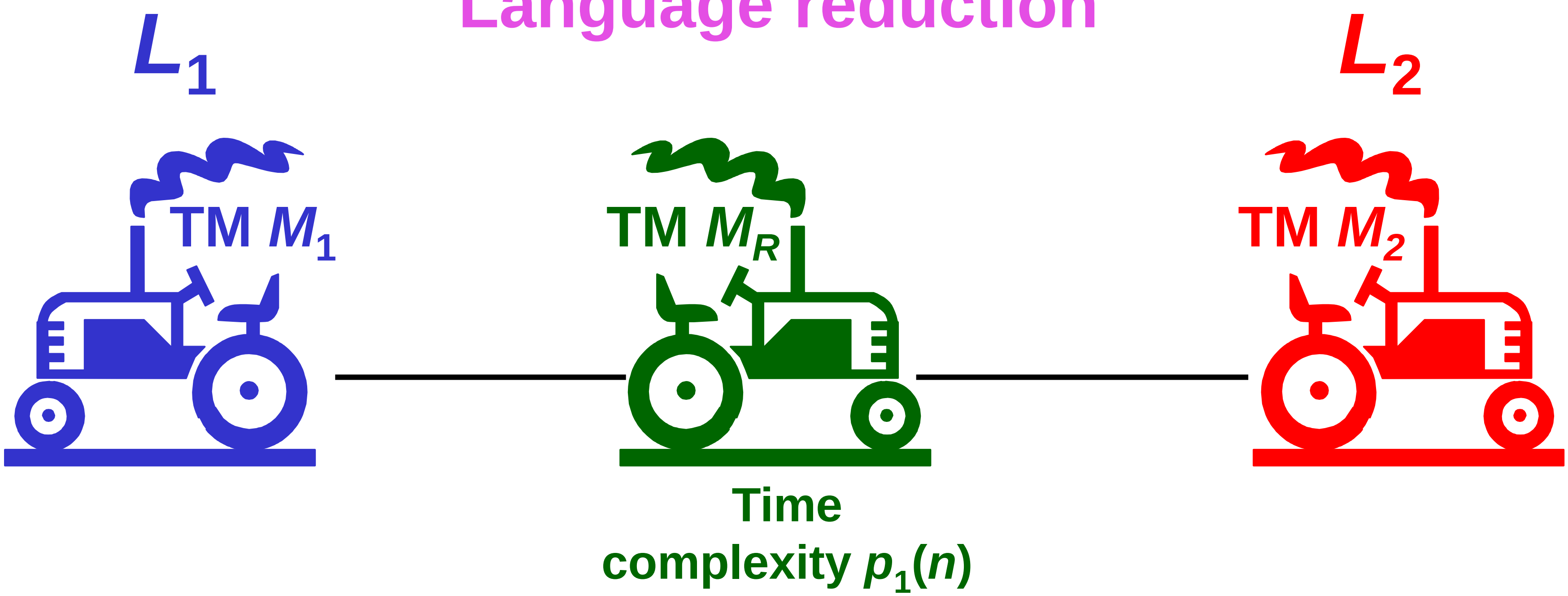
a) If **language L_2** is in class ***P***, then **language L_1** is also in class ***P***

b) If **language L_2** is in class ***NP,*** then **language L_1** is also in class ***NP***

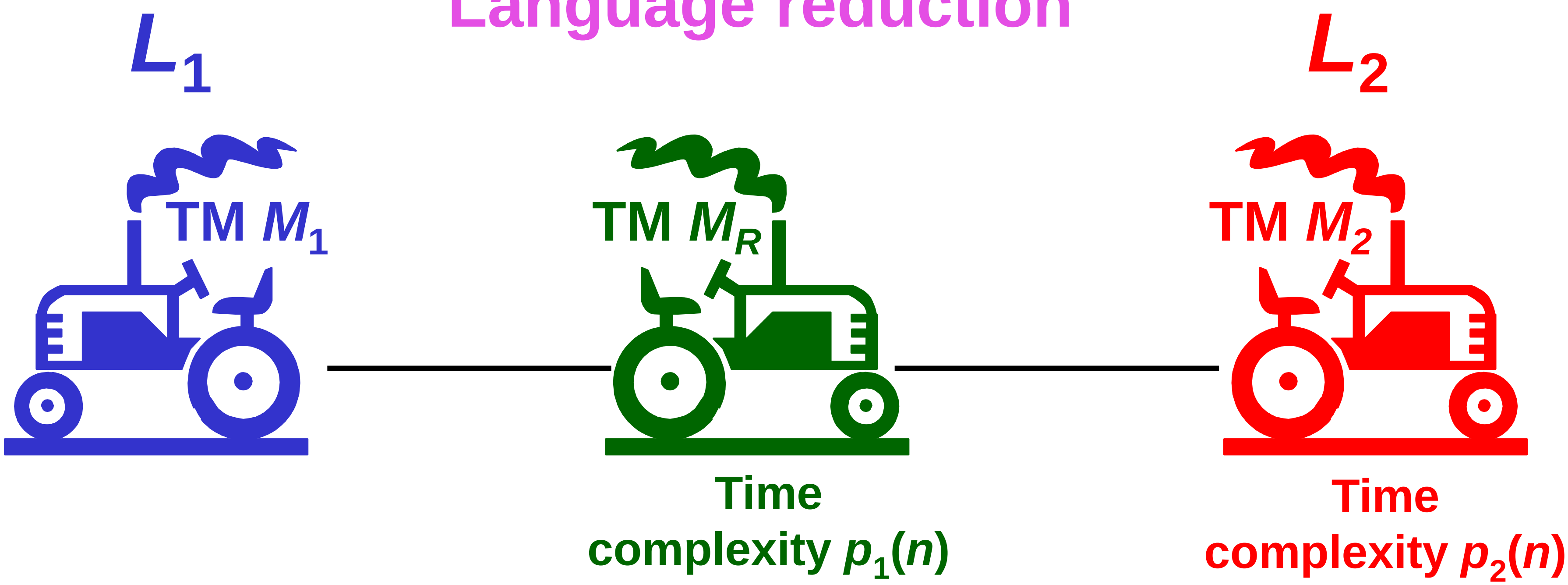
Language reduction



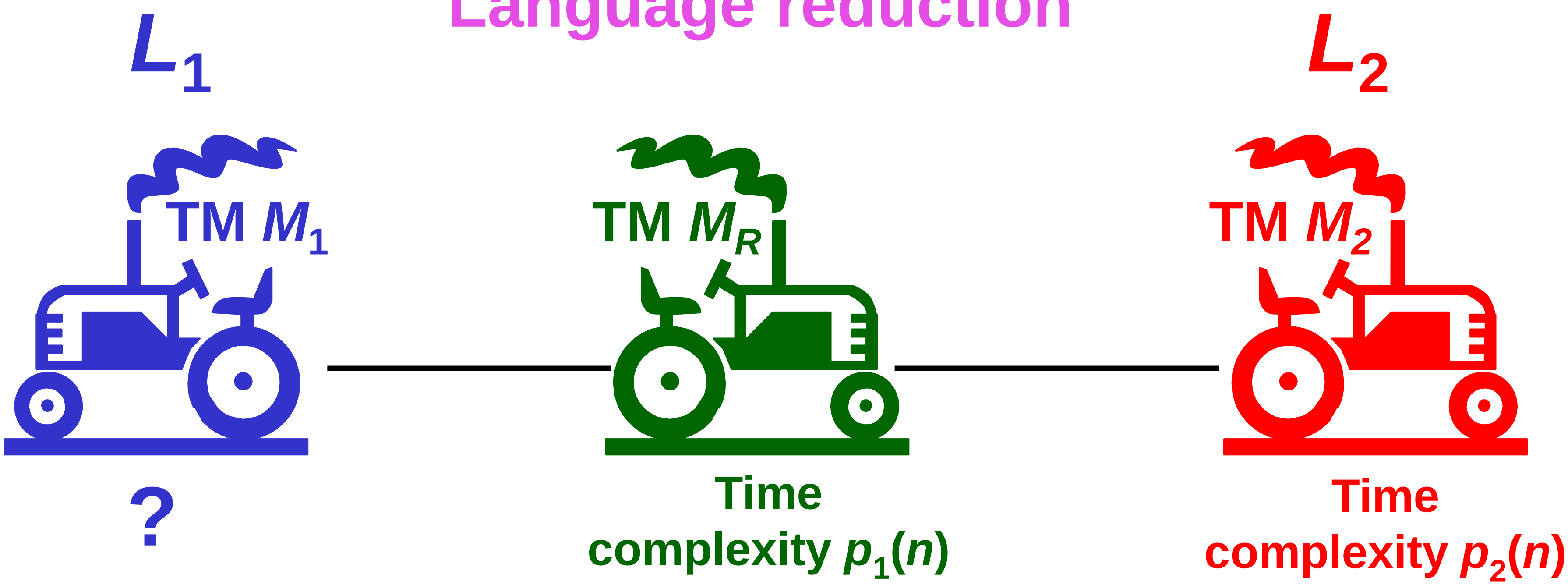
Language reduction



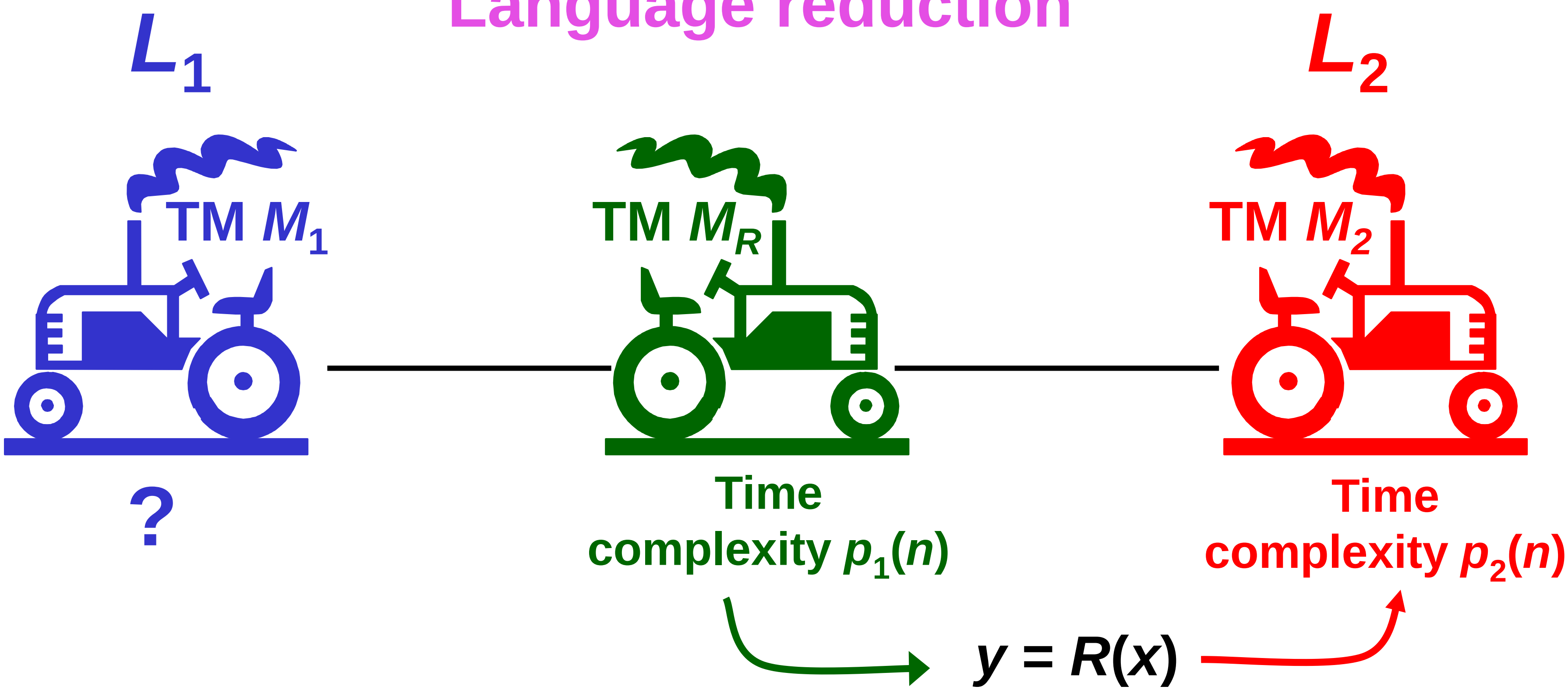
Language reduction



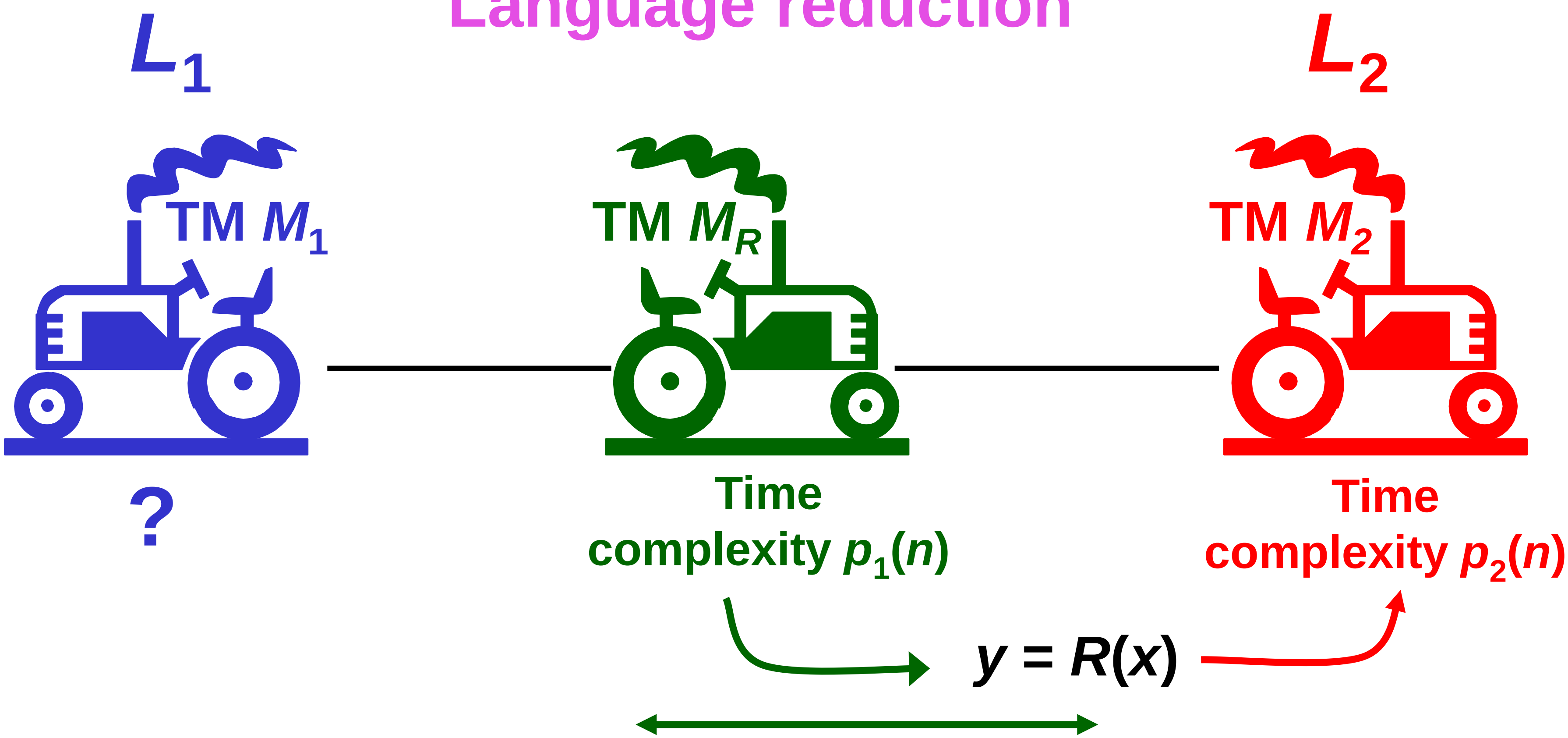
Language reduction



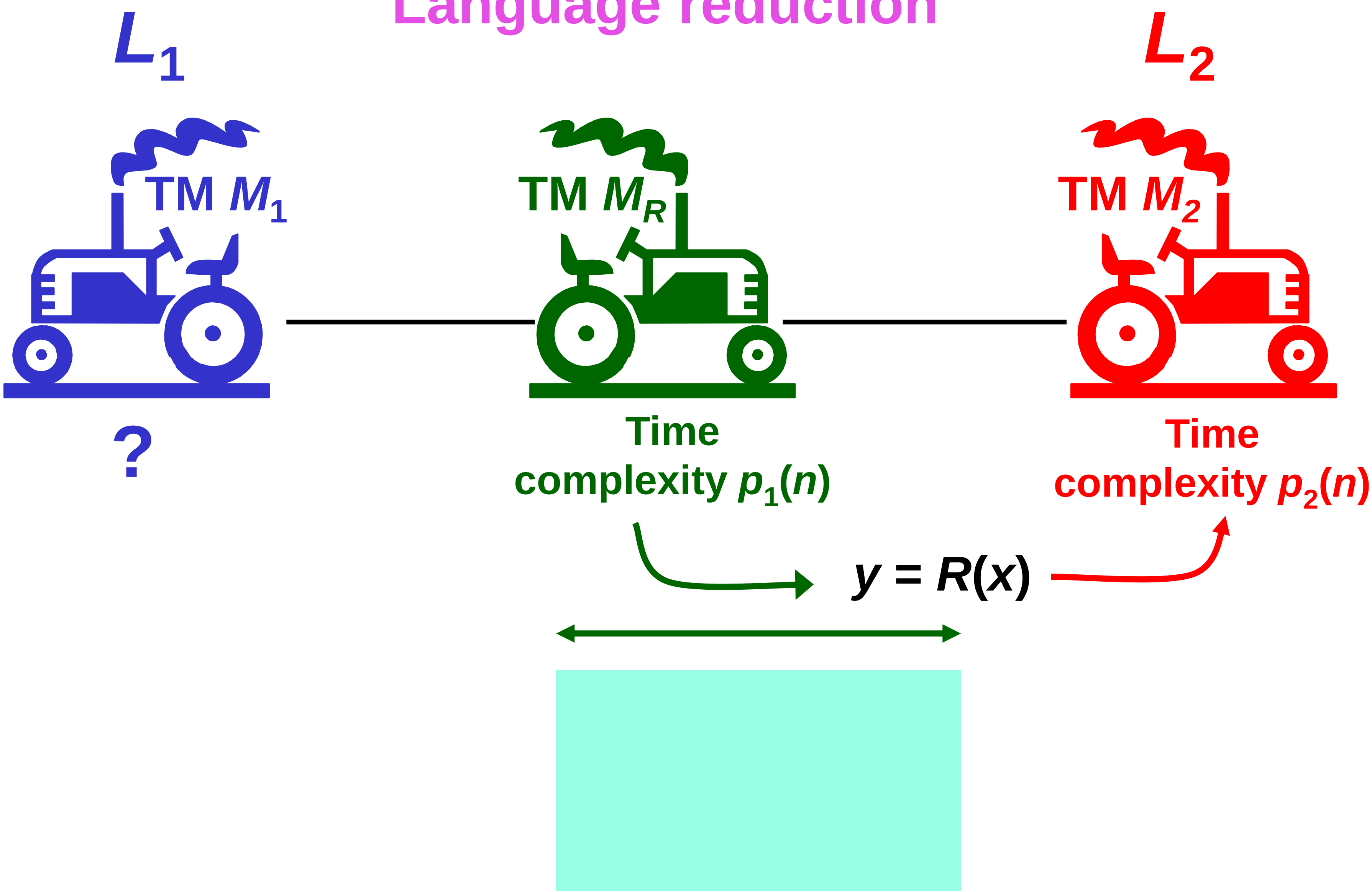
Language reduction



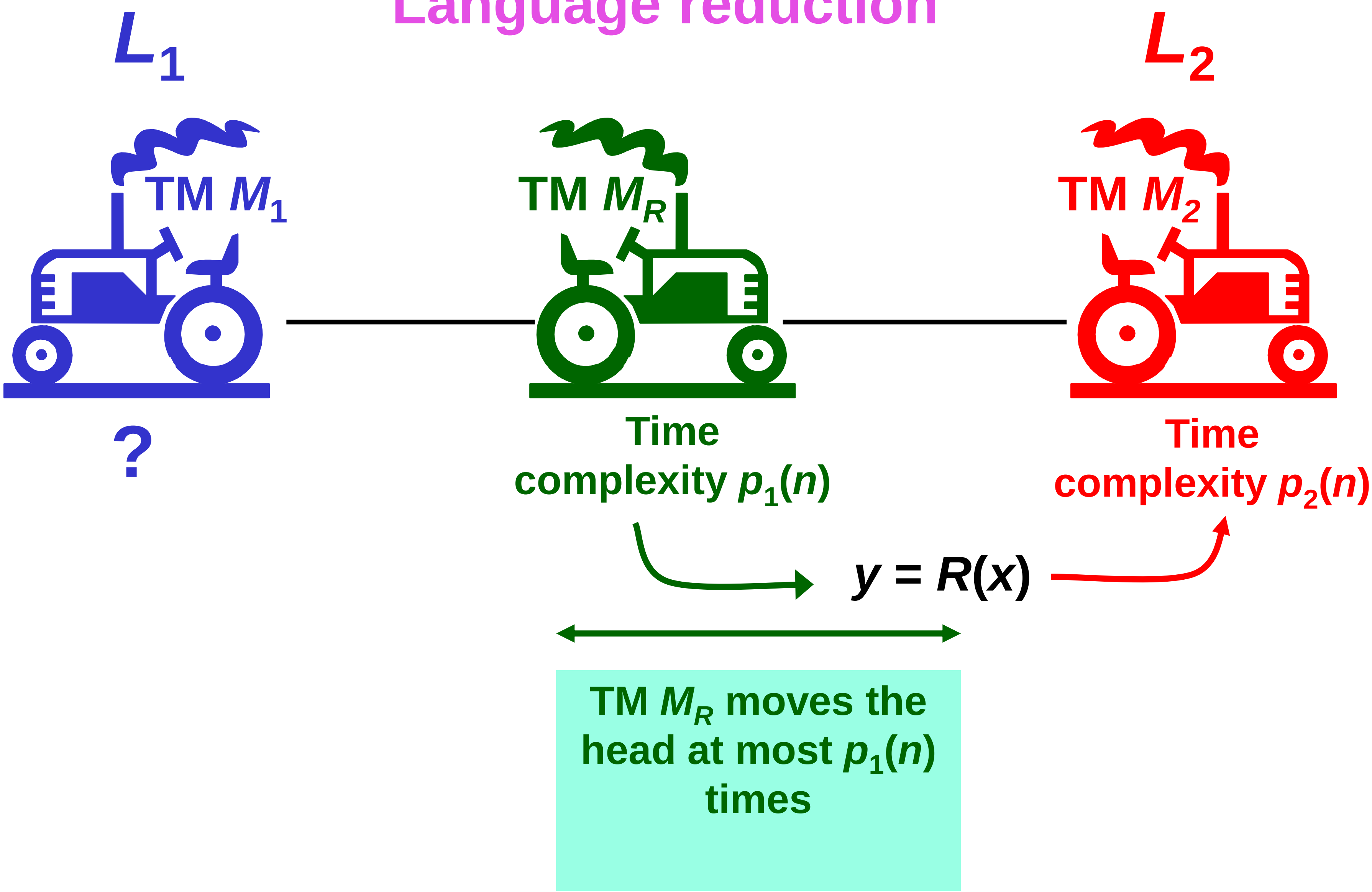
Language reduction



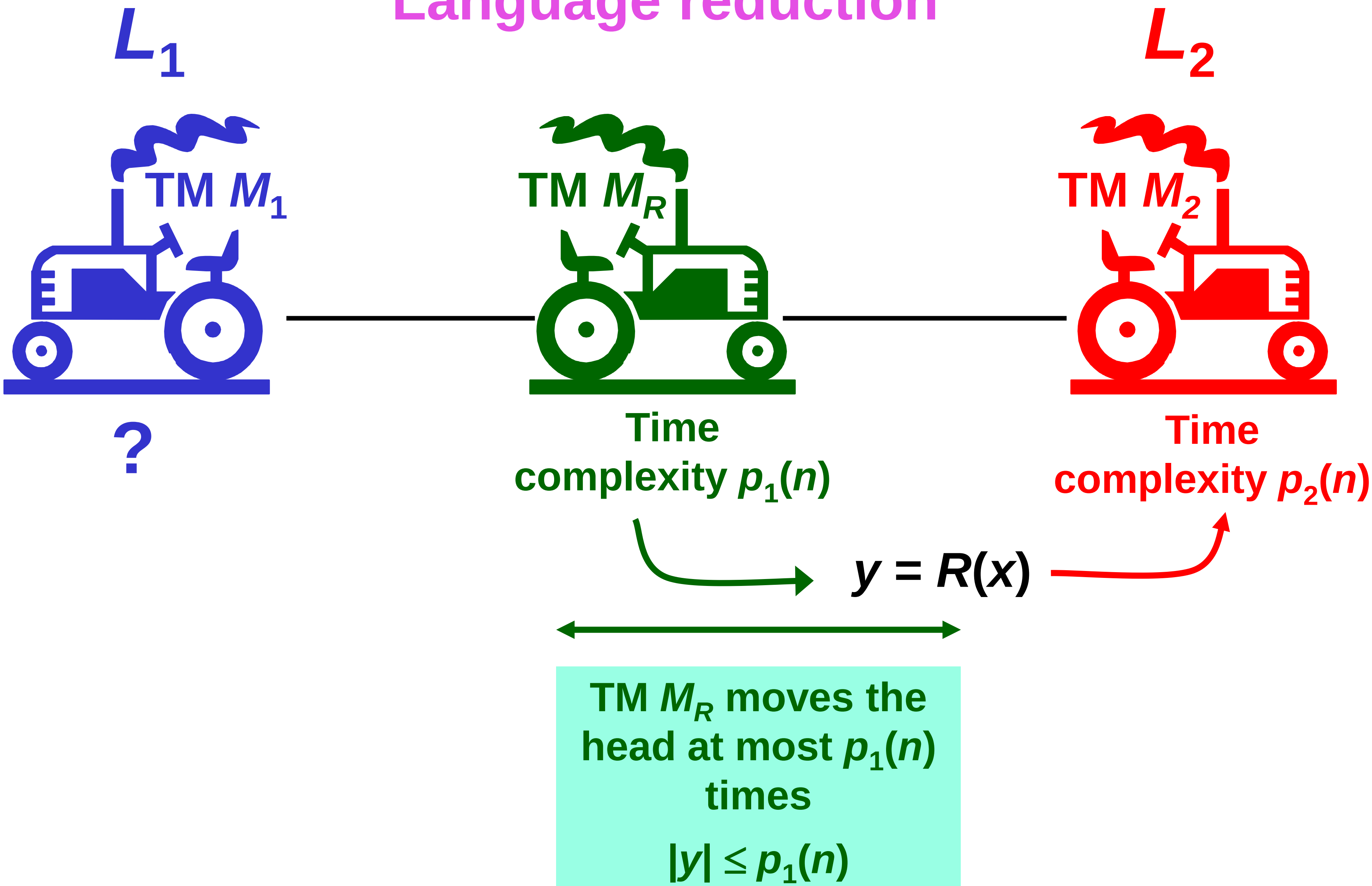
Language reduction



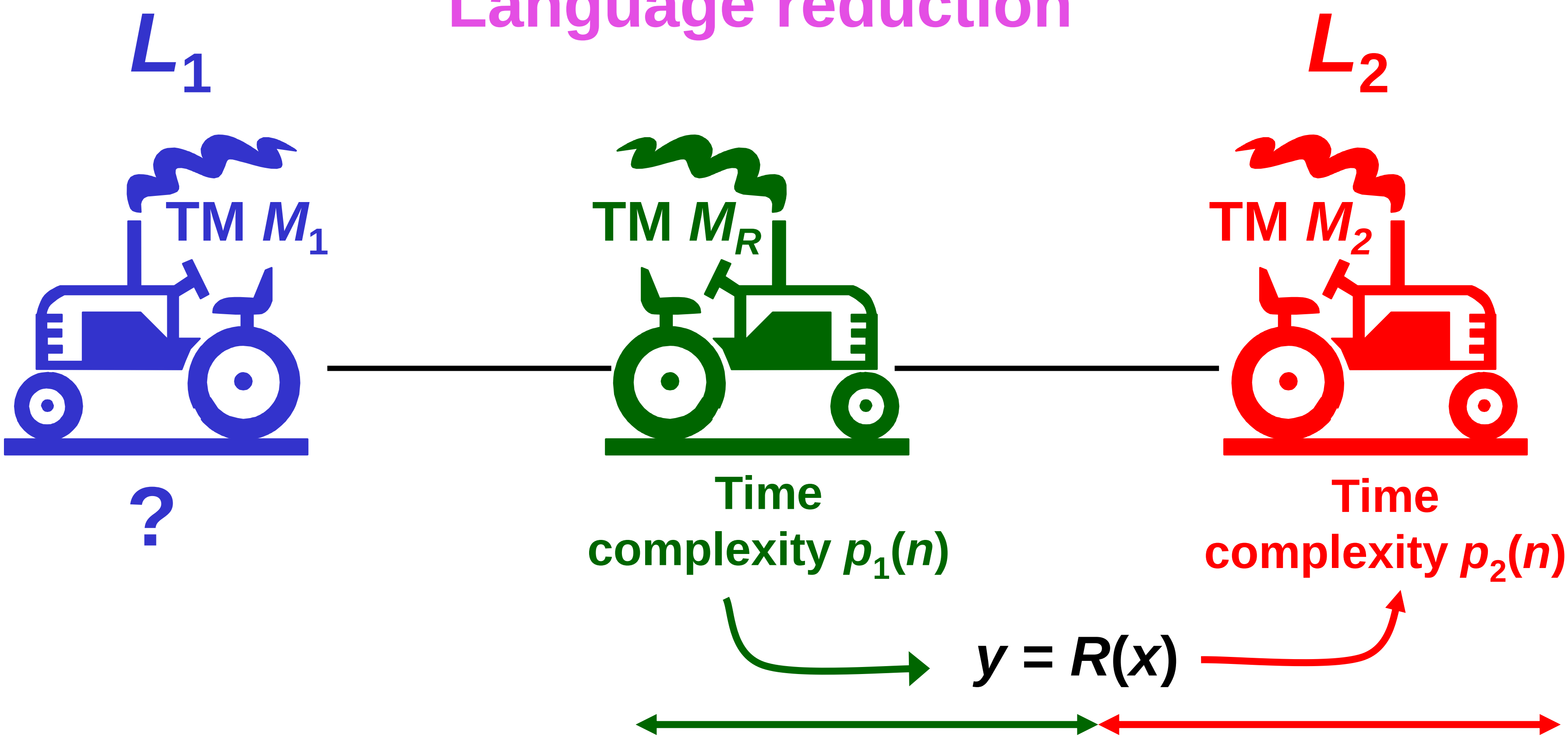
Language reduction



Language reduction



Language reduction

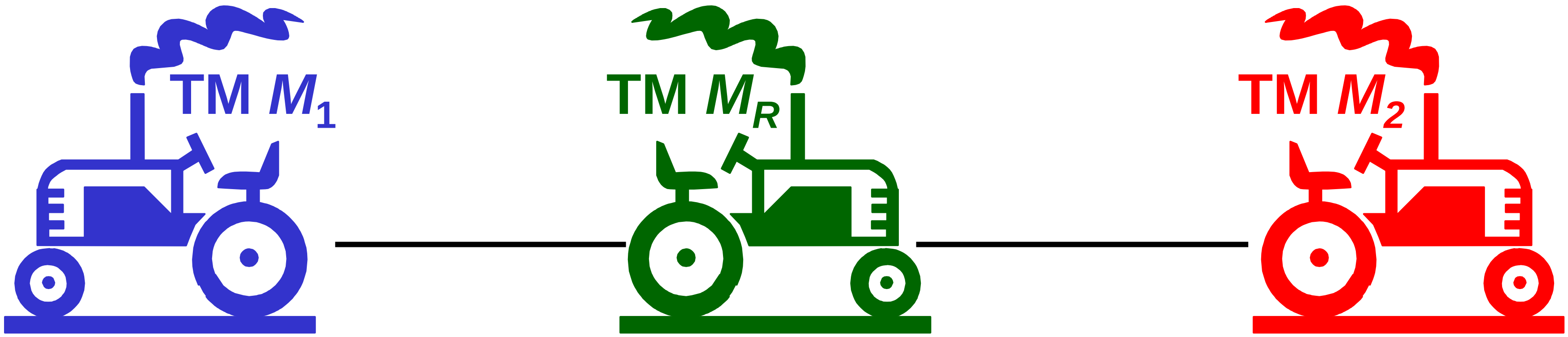


TM M_R moves the head at most $p_1(n)$ times

$|y| \leq p_1(n)$

Language reduction

L_1 L_2



?

Time complexity $p_1(n)$

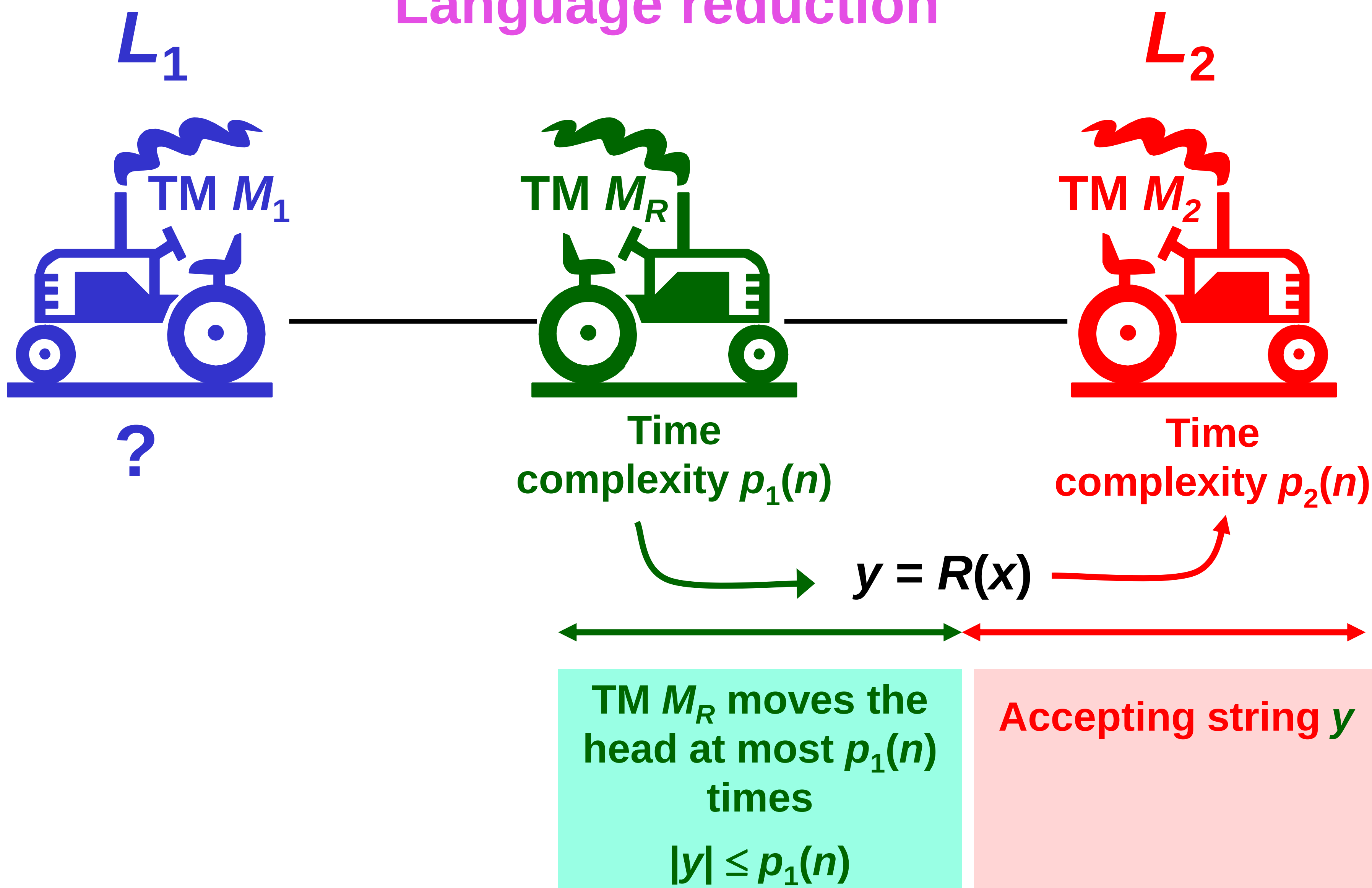
Time complexity $p_2(n)$

$y = R(x)$

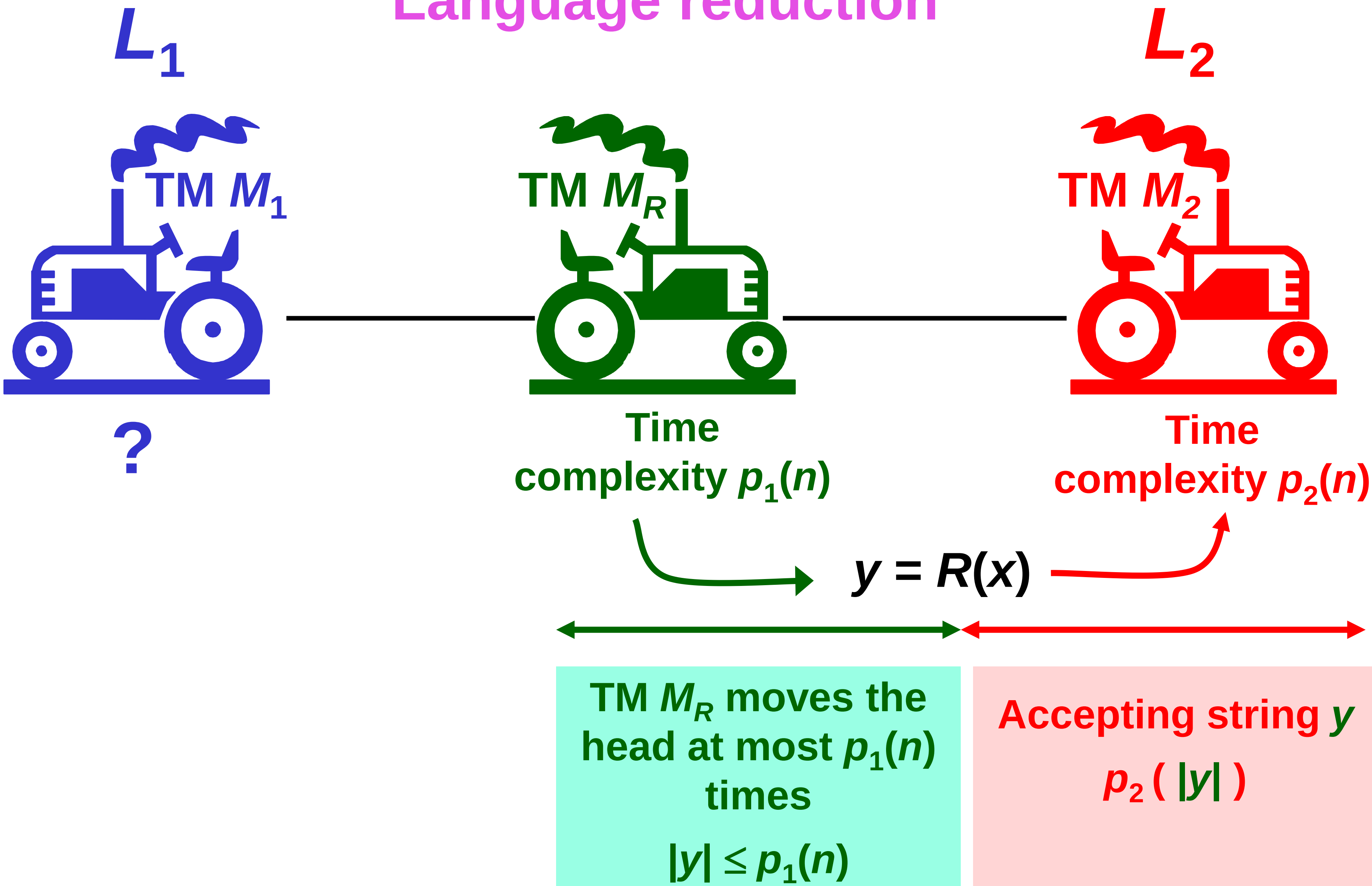
TM M_R moves the head at most $p_1(n)$ times

$|y| \leq p_1(n)$

Language reduction

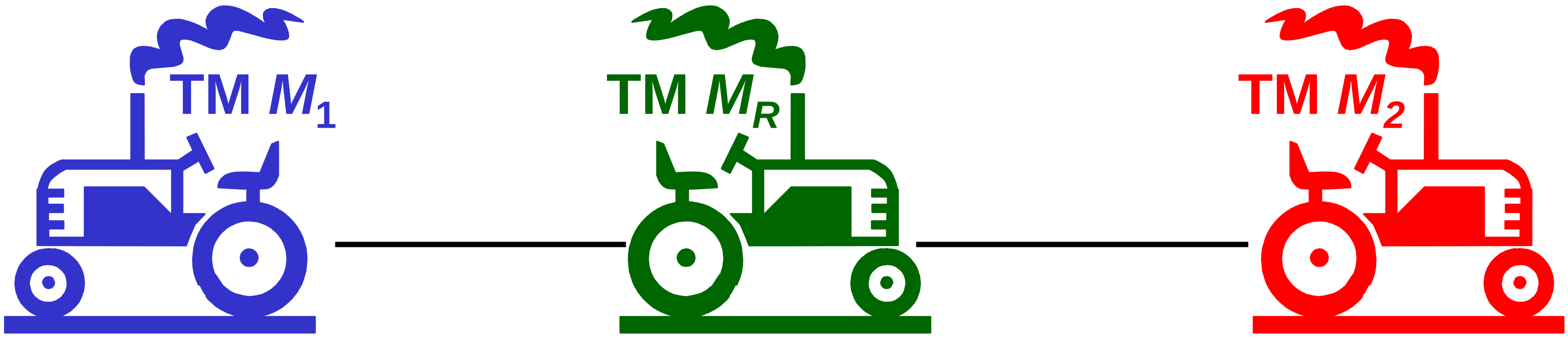


Language reduction



Language reduction

L_1 $\qquad$ L_2



?

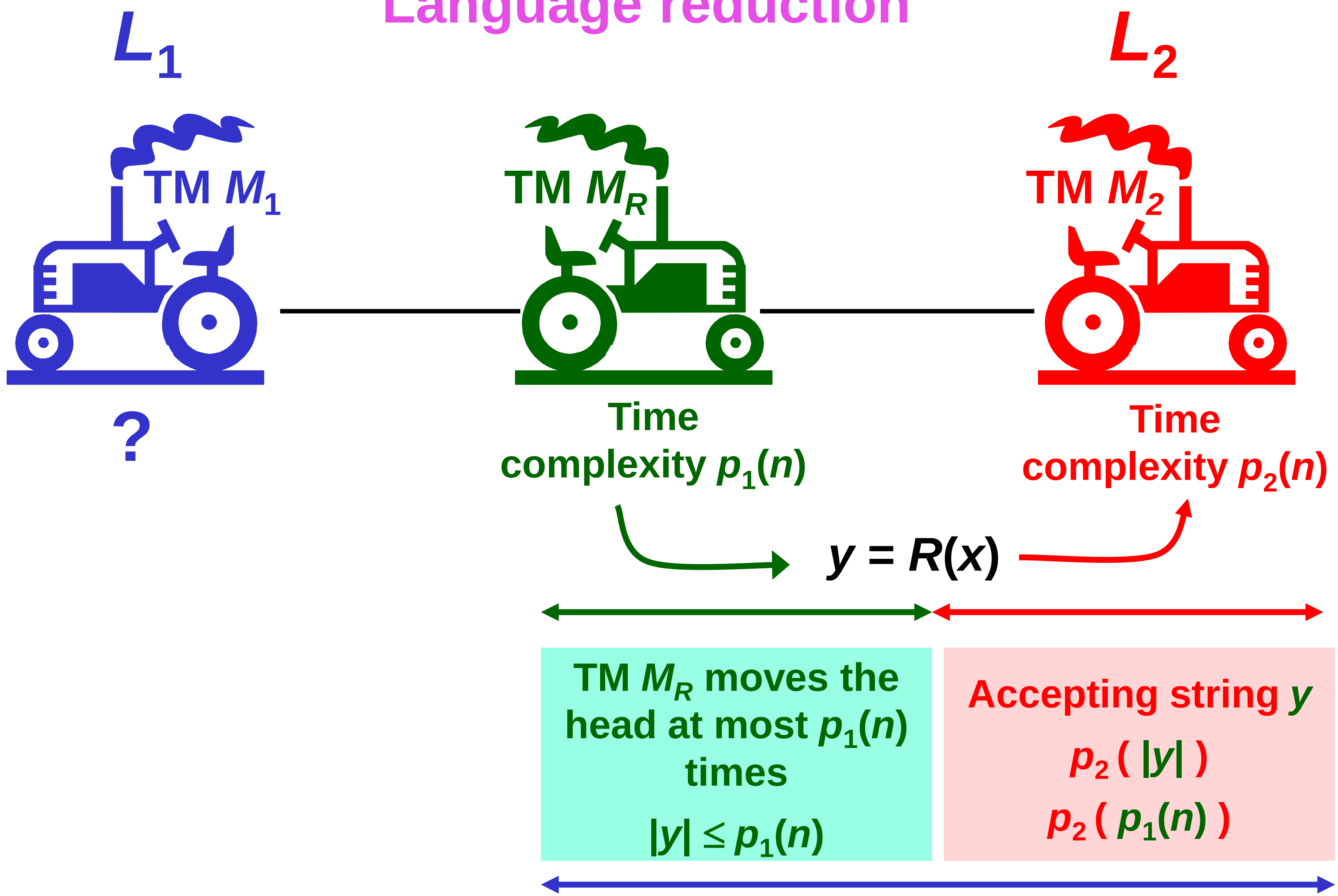
Time complexity $p_1(n)$

Time complexity $p_2(n)$

$y = R(x)$

TM M_R moves the head at most $p_1(n)$ times $\lvert y \rvert \leq p_1(n)$	Accepting string y $p_2(\ \lvert y \rvert\ )$ $p_2(\ p_1(n)\ )$

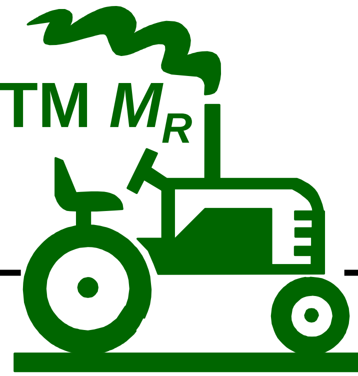
Language reduction



Language reduction

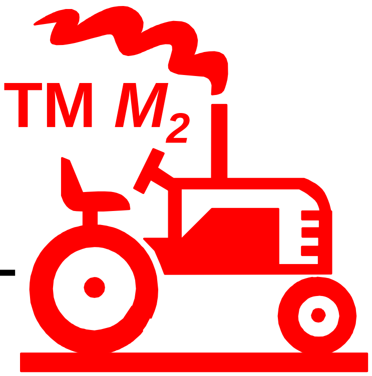
L_1

?



Time complexity $p_1(n)$

L_2



Time complexity $p_2(n)$

$y = R(x)$

TM M_R moves the head at most $p_1(n)$ times

$|y| \leq p_1(n)$

Accepting string y

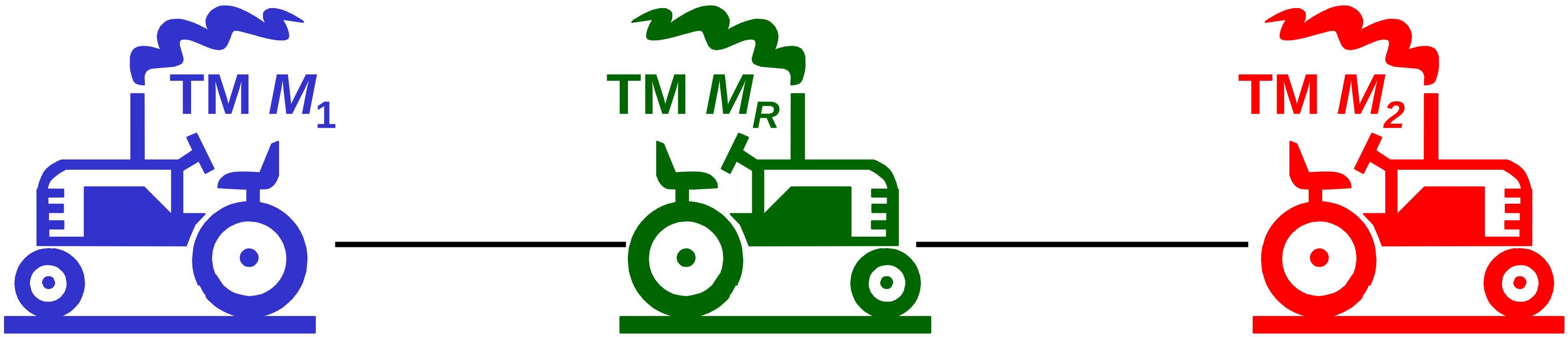
$p_2(\,|y|\,)$

$p_2(\,p_1(n)\,)$

$p_1(n) + p_2(\,p_1(n)\,)$

Language reduction

L_1 L_2



Time complexity $p_1(n) + p_2(p_1(n))$

Time complexity $p_1(n)$

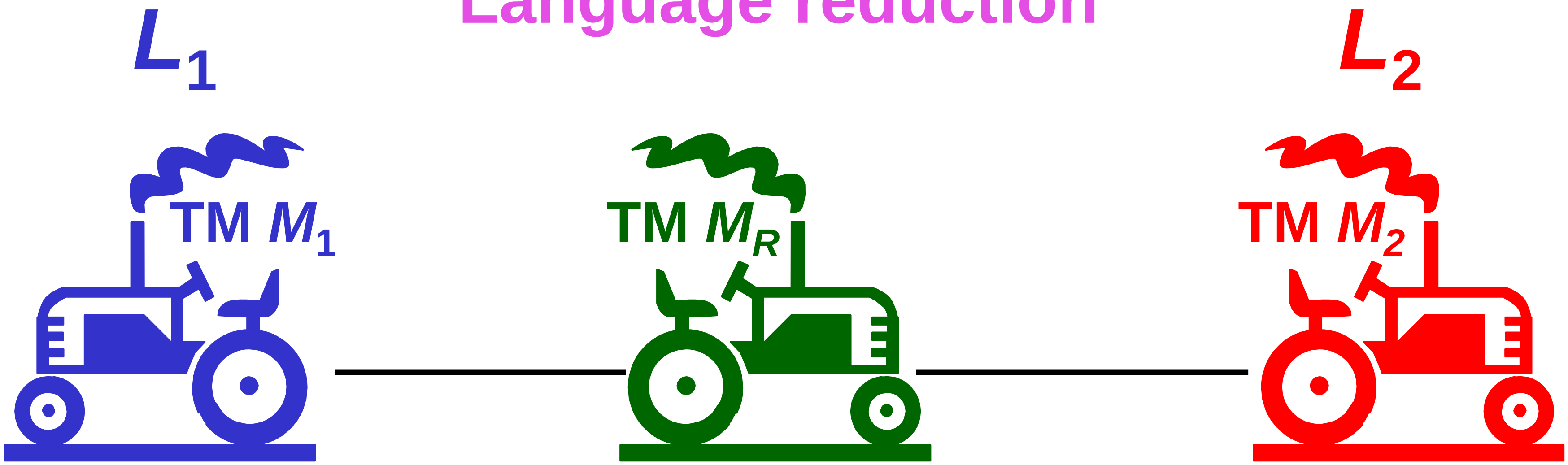
Time complexity $p_2(n)$

$y = R(x)$

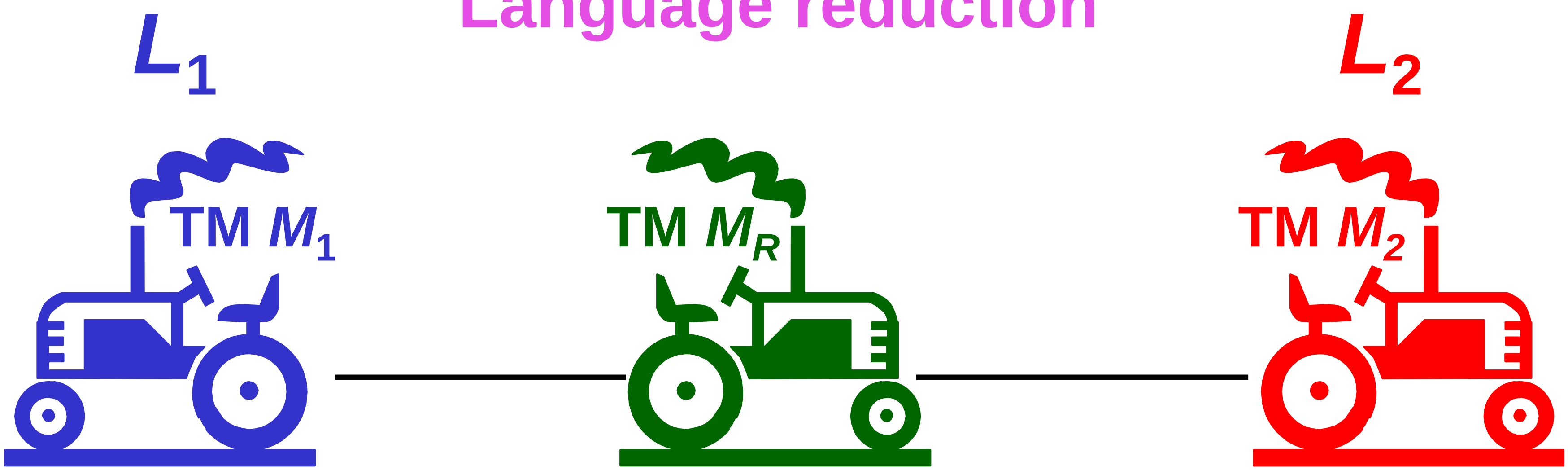
TM M_R moves the head at most $p_1(n)$ times $\lvert y \rvert \le p_1(n)$	Accepting string y $p_2(\lvert y \rvert)$ $p_2(p_1(n))$

$p_1(n) + p_2(p_1(n))$

Language reduction

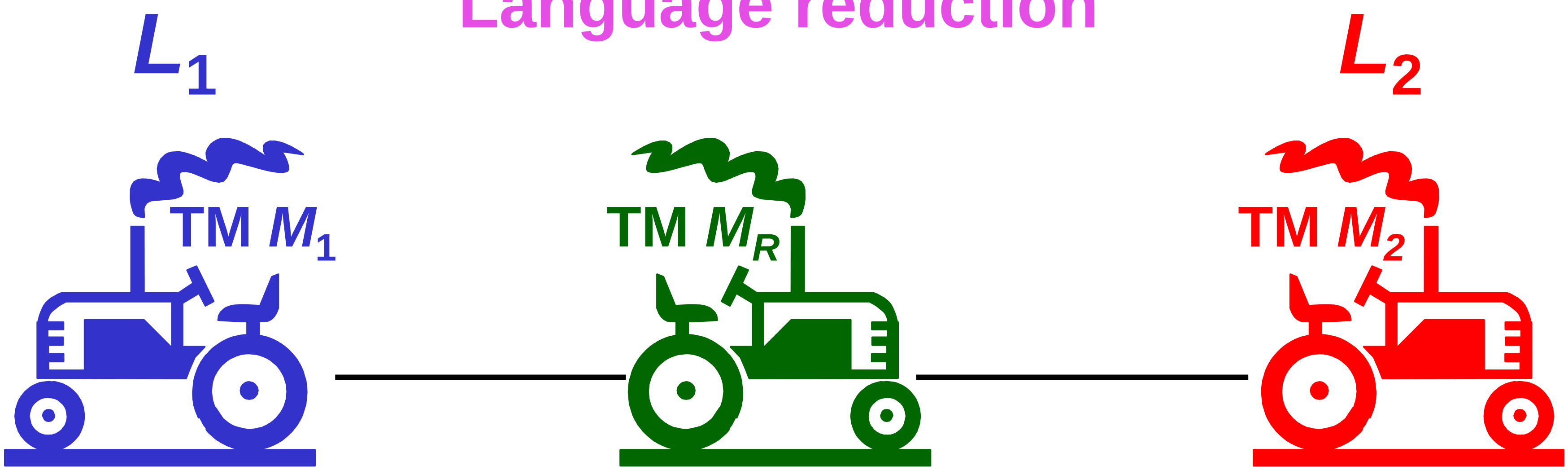


Language reduction



- **Language L_1** is **log-space** reduced to **language L_2**

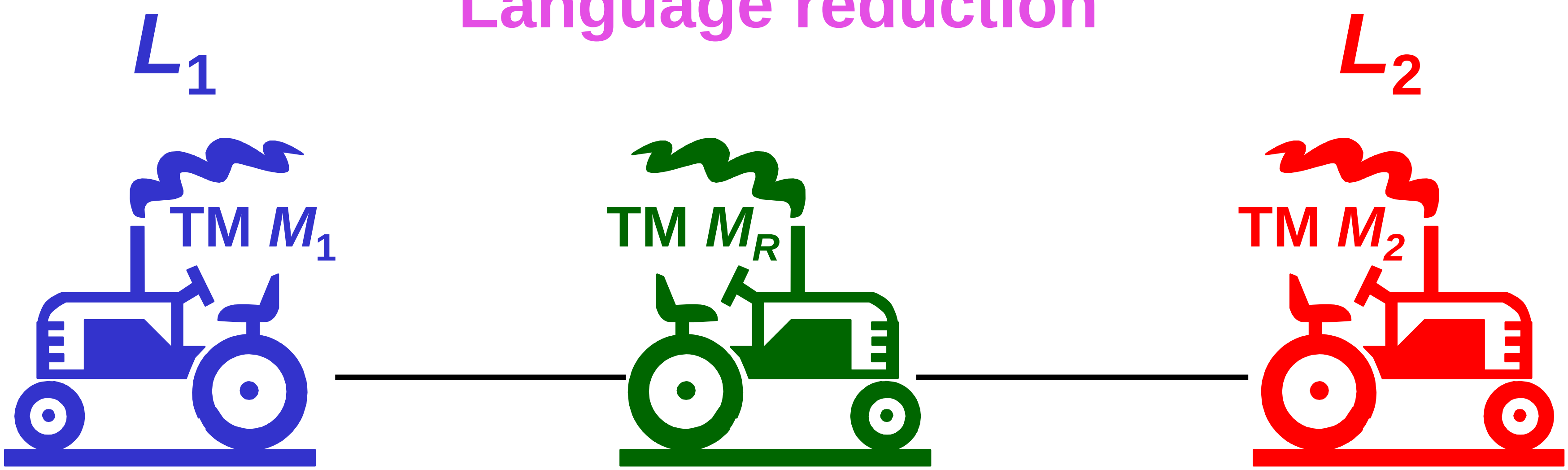
Language reduction



- **Language L_1** is **log-space** reduced to **language L_2**

c) If **language L_2** is in class $\boldsymbol{P}$, then **language L_1** is also in class $\boldsymbol{P}$

Language reduction

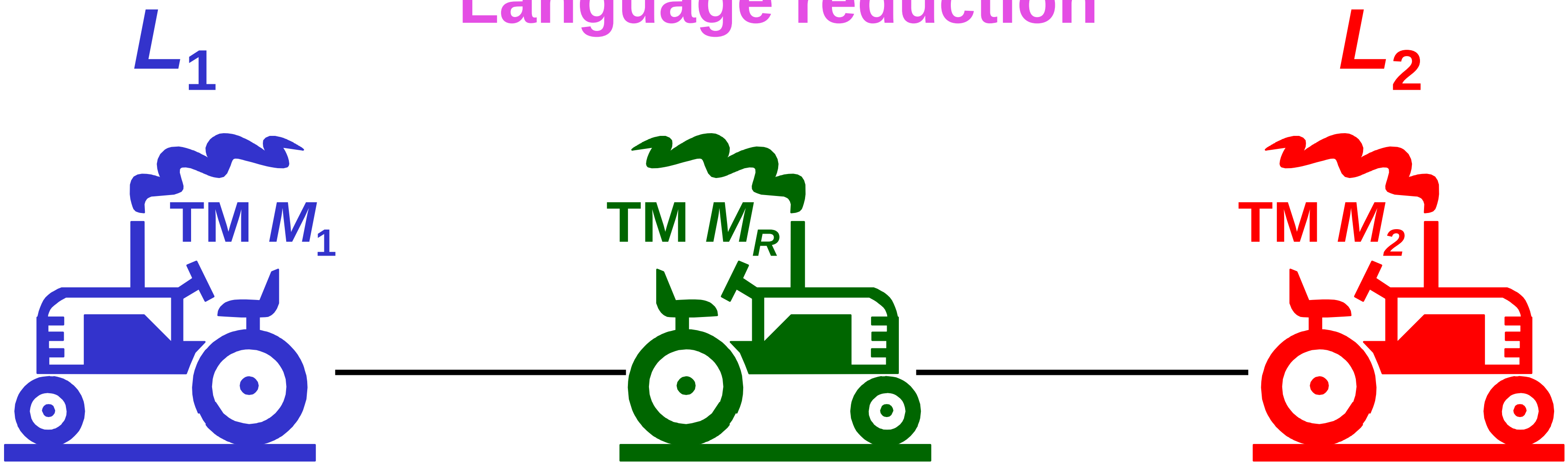


- **Language L_1** is **log-space** reduced to **language L_2**

c) If **language L_2** is in class ***P***, then **language L_1** is also in class ***P***

d) If **language L_2** *is in class* **NSPACE($\log^k n$)**, then **language L_1** is also in class **NSPACE($\log^k n$)**

Language reduction



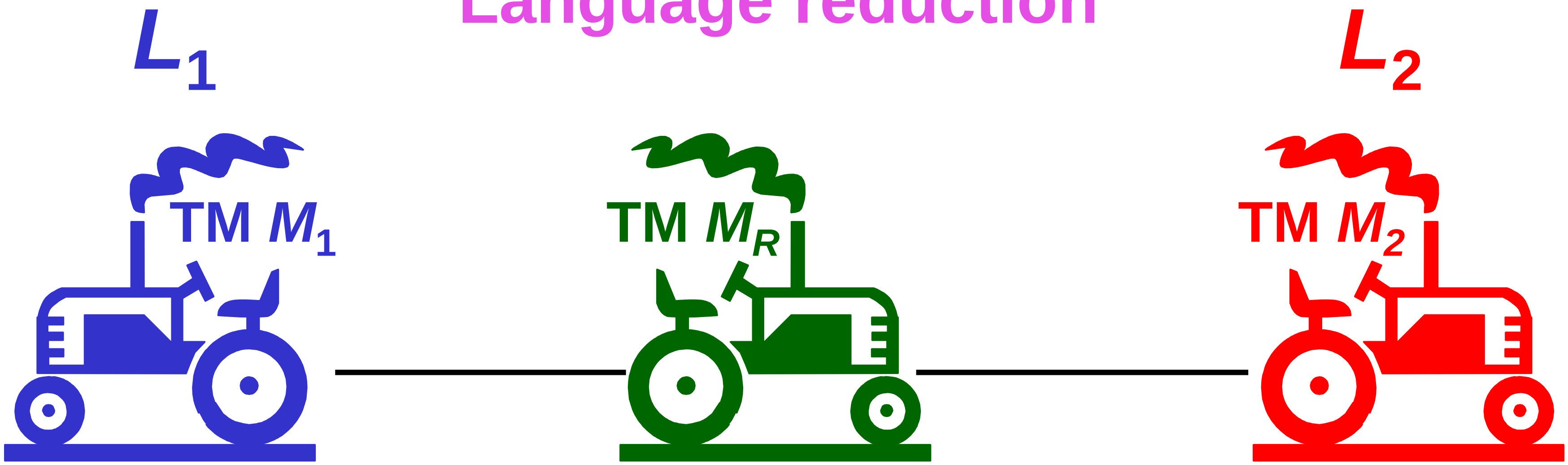
- **Language** L_1 is **log-space** reduced to **language** L_2

c) If **language** L_2 is in class ***P***, then **language** L_1 is also in class ***P***

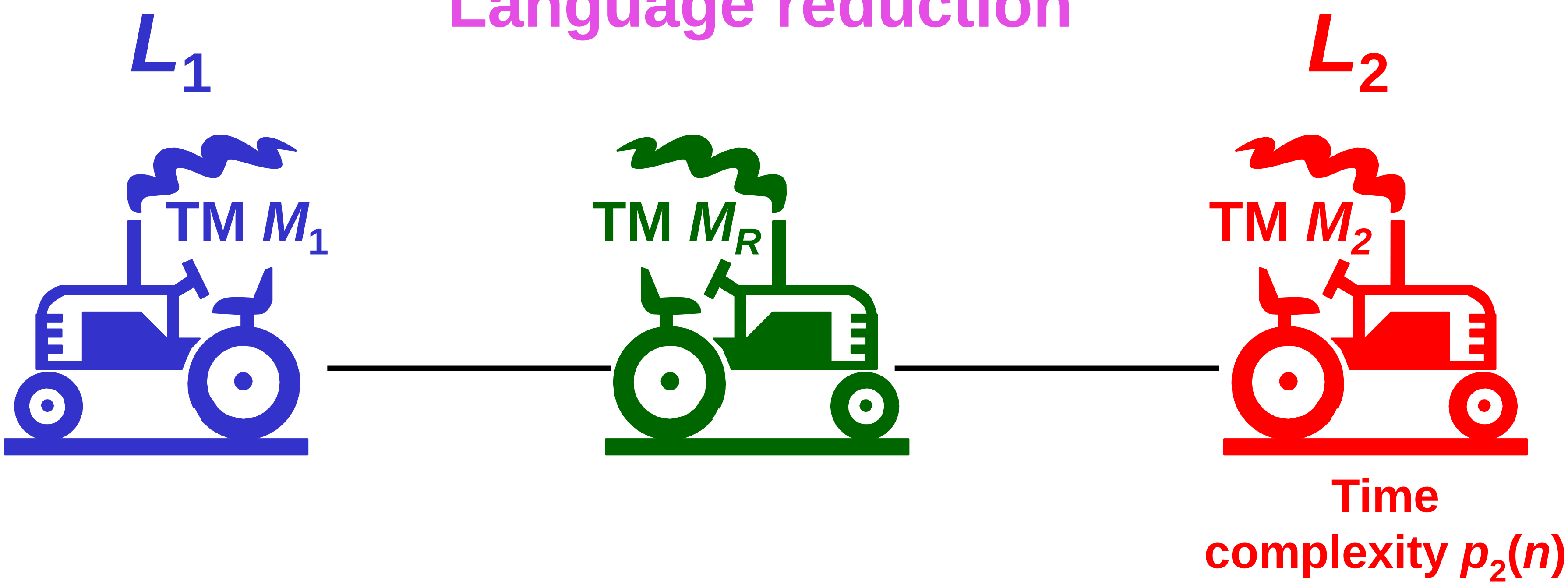
d) If **language** L_2 *is in class* **NSPACE(log^k*n*)**, then **language** L_1 is also in class **NSPACE(log^k*n*)**

e) If **language** L_2 *is in class* **DSPACE(log^k*n*)**, then **language** L_1 is also in class **DSPACE(log^k*n*)**

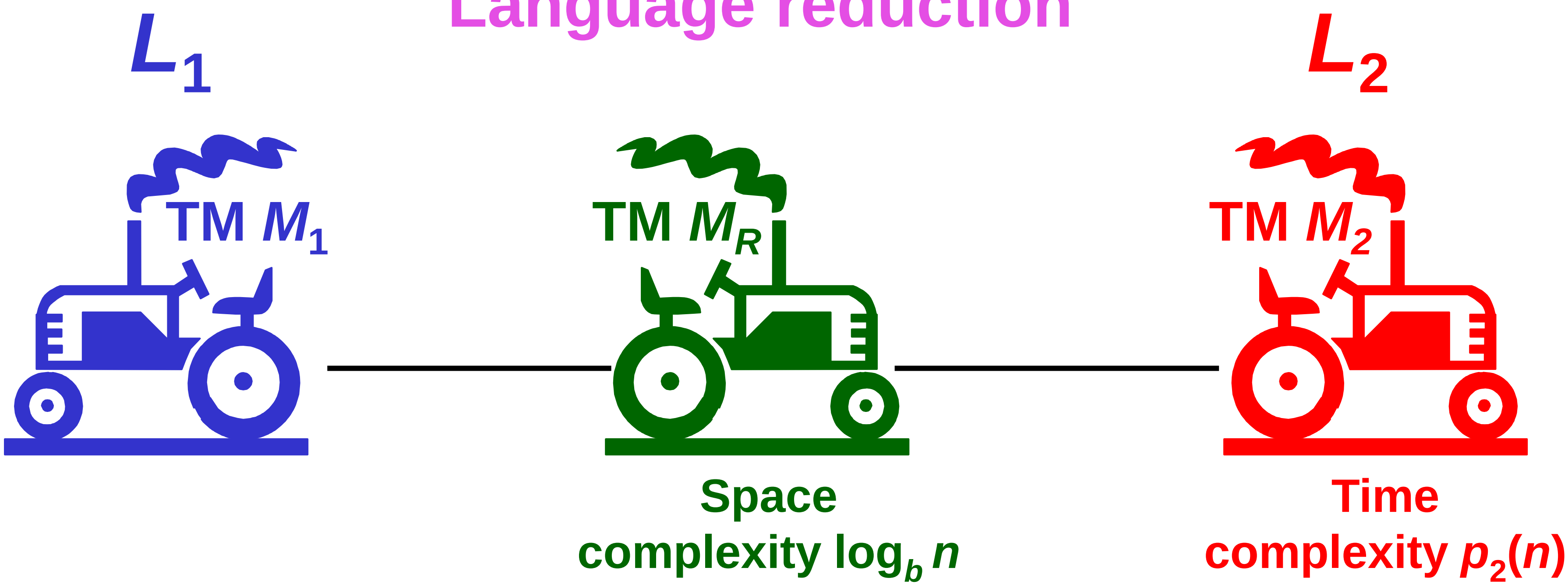
Language reduction



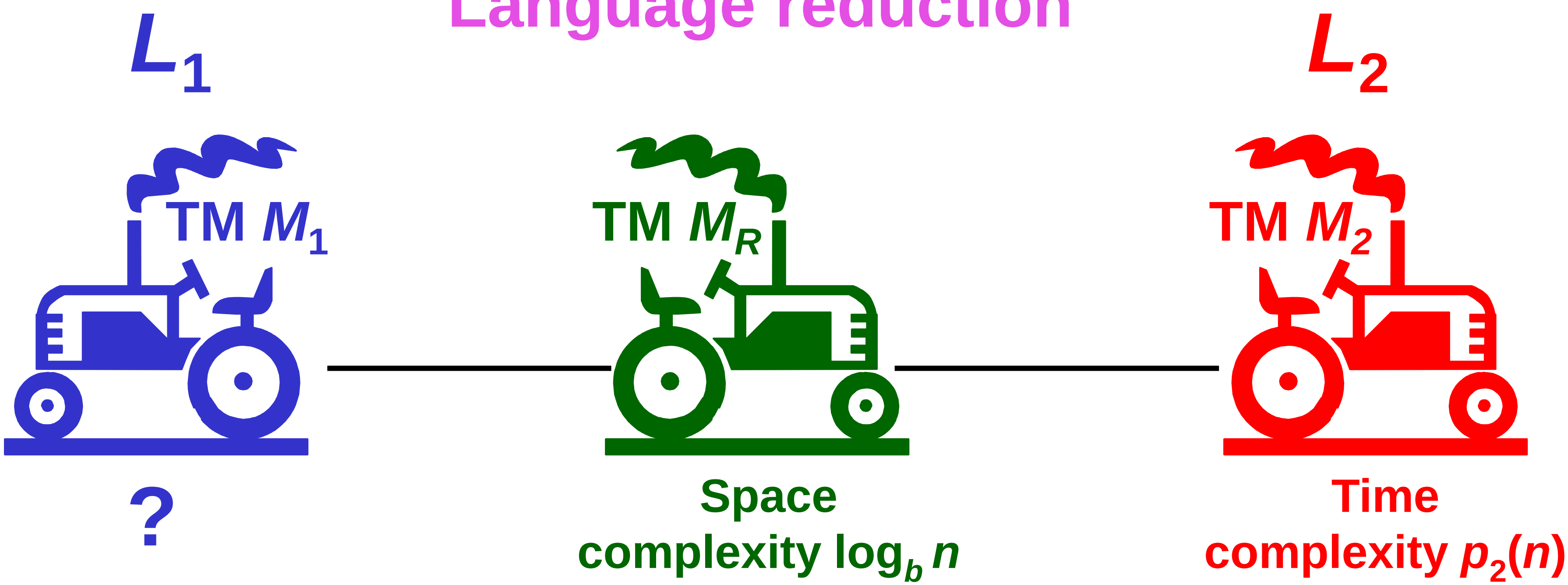
Language reduction



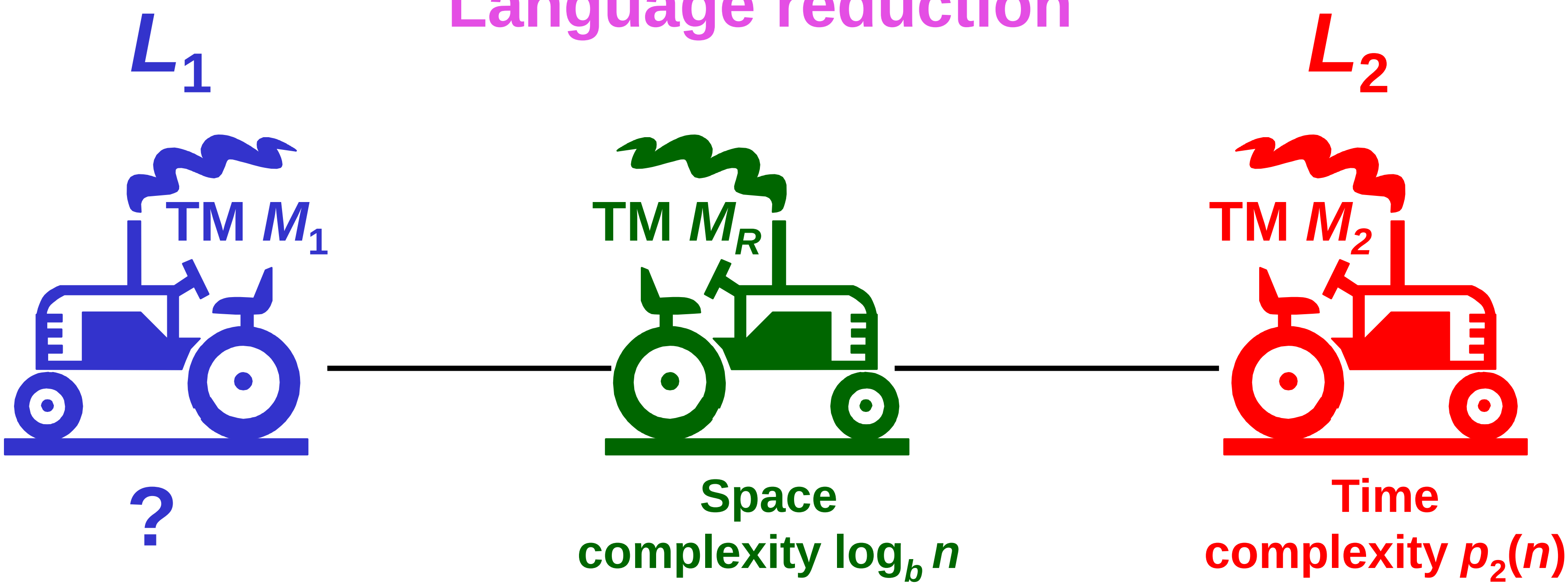
Language reduction



Language reduction

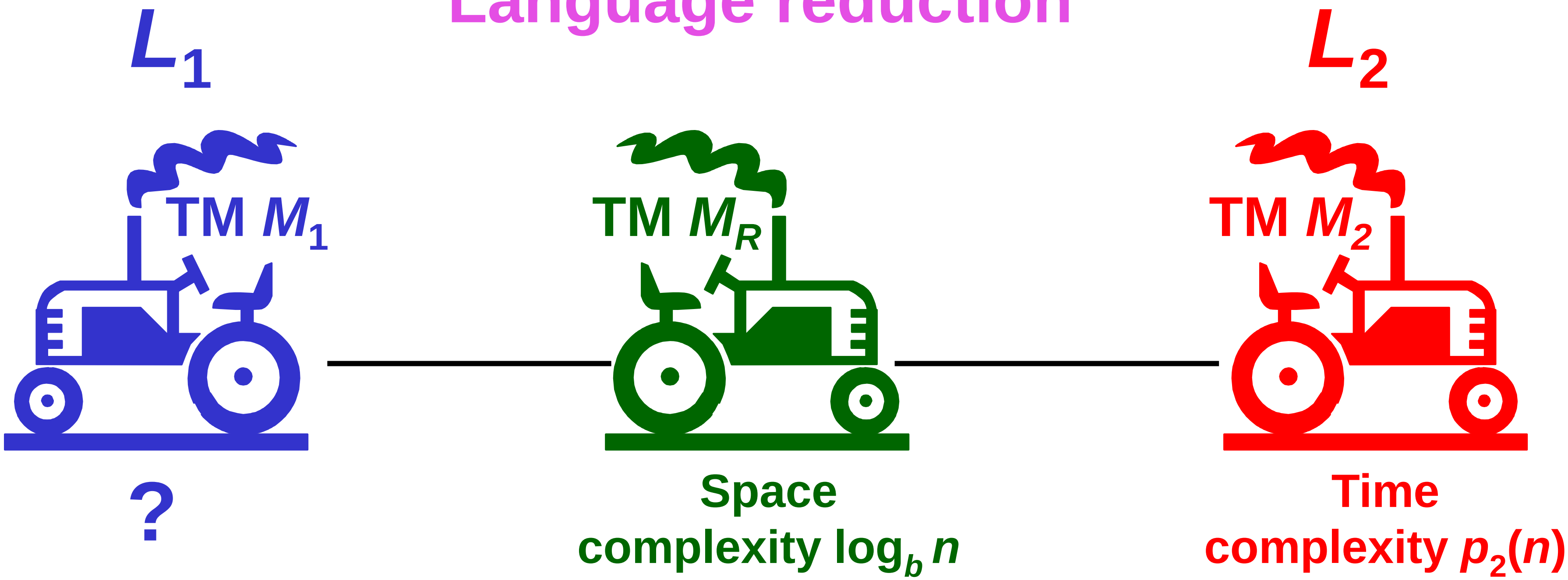


Language reduction



Case (*c*):

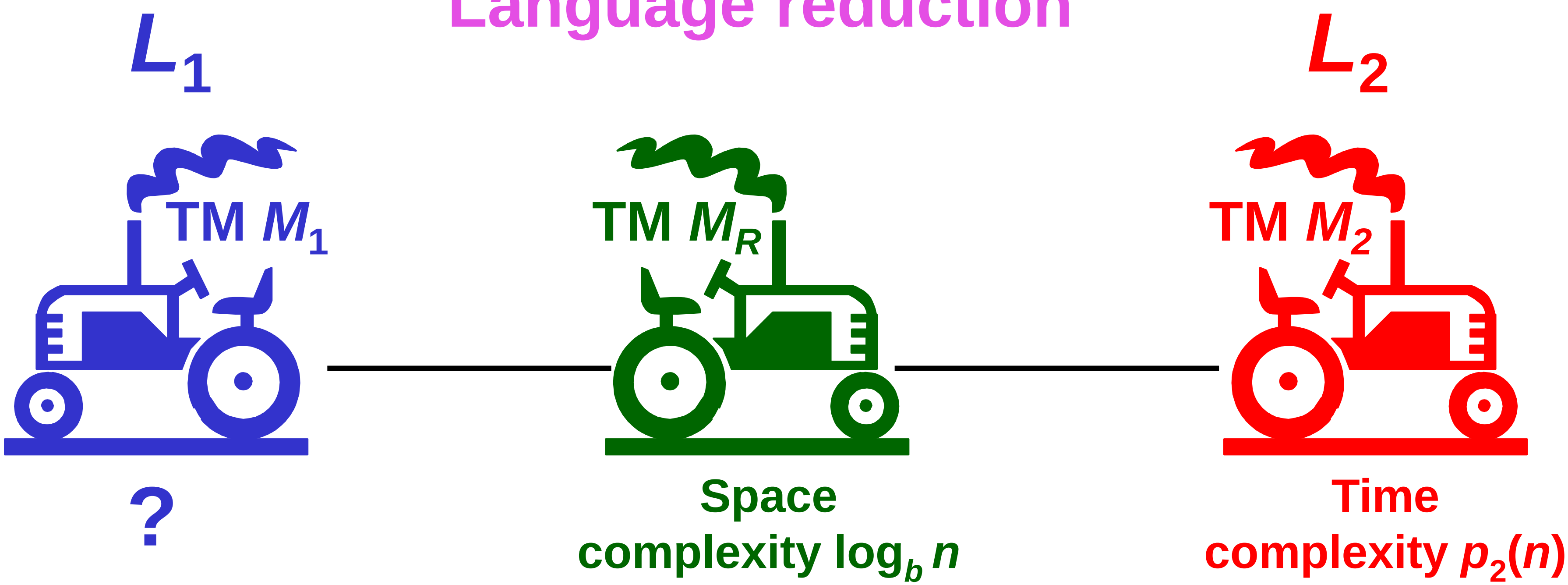
Language reduction



Case (*c*):

$L \in \mathrm{DSPACE}(f(n)) \Rightarrow L \in \mathrm{DTIME}(c^{f(n)})$

Language reduction

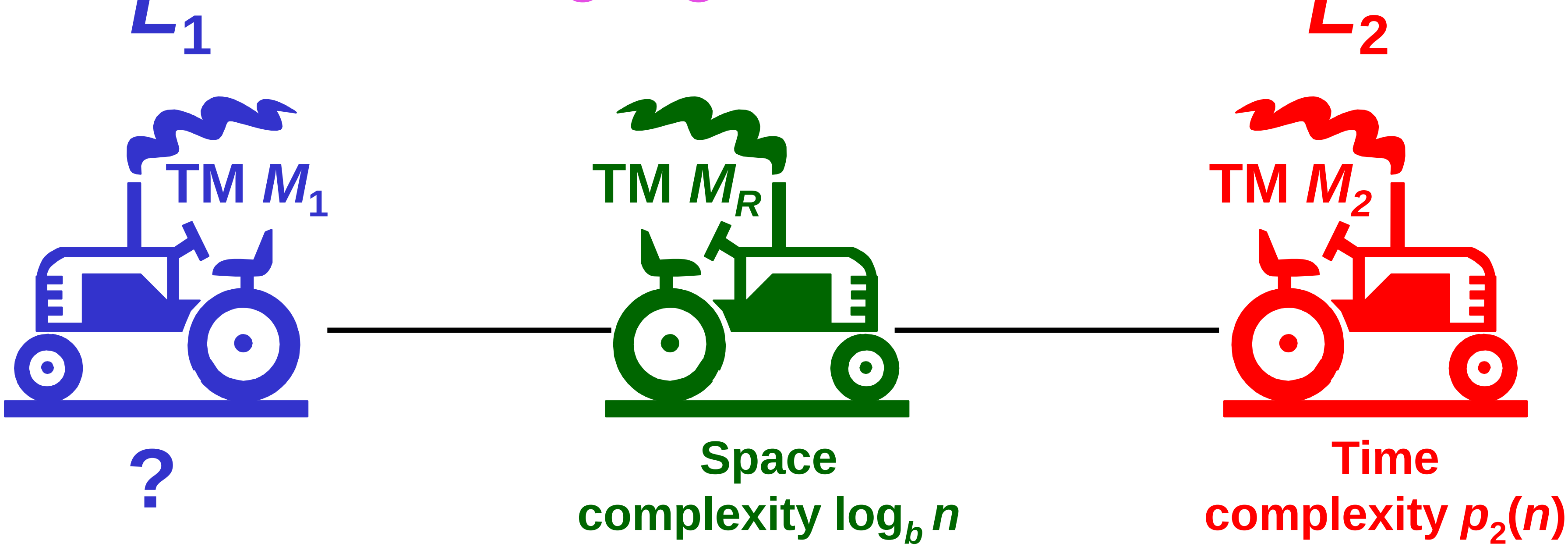


Case (*c*):

$L \in \mathrm{DSPACE}(f(n)) \Rightarrow L \in \mathrm{DTIME}(c^{f(n)})$

$f(n) = \log_b n$

Language reduction



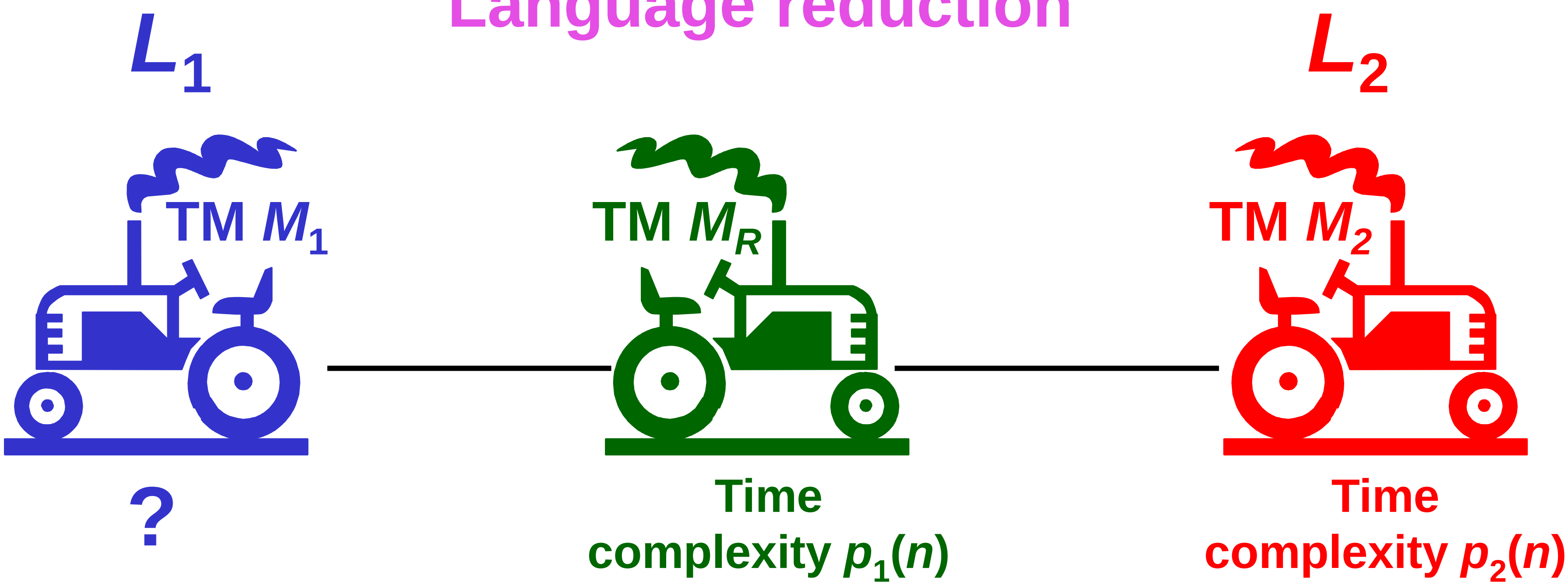
Case (*c*):

$L \in \text{DSPACE}(f(n)) \Rightarrow L \in \text{DTIME}(c^{f(n)})$

$f(n) = \log_b n$

$c^{f(n)} = c^{\log_b n} \square (b^k)^{\log_b n} = b^{k \log_b n} = (b^{\log_b n})^k = n^k$

Language reduction



Case (c):

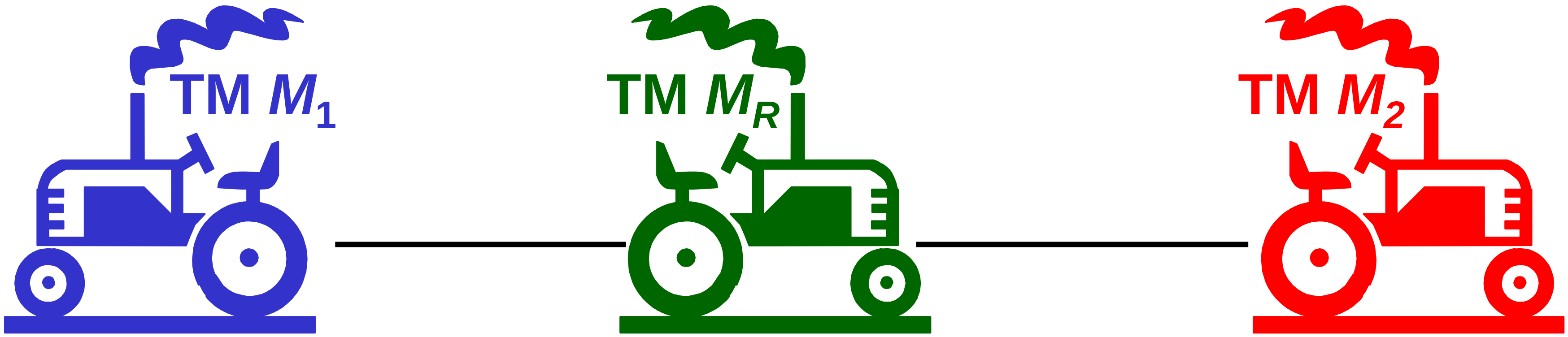
$L \in \mathrm{DSPACE}(f(n)) \Rightarrow L \in \mathrm{DTIME}(c^{f(n)})$

$f(n) = \log_b n$

$c^{f(n)} = c^{\log_b n} \square (b^k)^{\log_b n} = b^{k \log_b n} = (b^{\log_b n})^k = n^k$

Language reduction

L_1 L_2



?

Time complexity $p_1(n)$

Time complexity $p_2(n)$

$y = R(x)$

Case (*c*):

$L \in \mathrm{DSPACE}(f(n)) \Rightarrow L \in \mathrm{DTIME}(c^{f(n)})$

$f(n) = \log_b n$

TM M_R moves the head at most $p_1(n)$ times

$|y| \leq p_1(n)$

Accepting string y

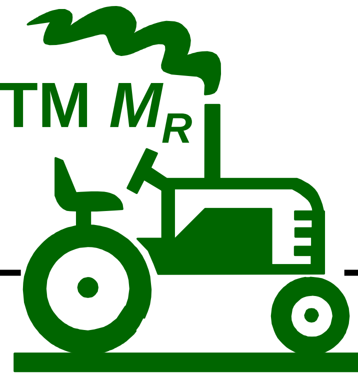
$p_2(\,|y|\,)$

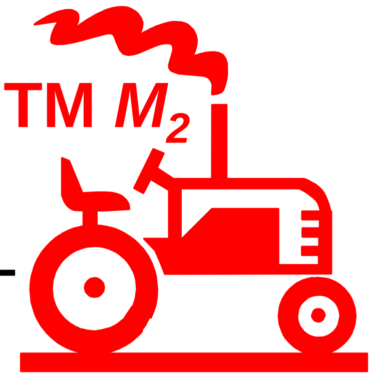
$p_2(\,p_1(n)\,)$

$c^{f(n)} = c^{\log bn} \;\Box\; (b^k)^{\log bn} = b^{k \log bn} = (b^{\log bn})^k = n^{\,k}$ $p_1(n) + p_2(\,p_1(n)\,)$

Language reduction

L_1 L_2





Time complexity $p_1(n) + p_2(p_1(n))$

Time complexity $p_1(n)$

Time complexity $p_2(n)$

$y = R(x)$

Case (*c*):

$L \in \mathrm{DSPACE}(f(n)) \Rightarrow L \in \mathrm{DTIME}(c^{f(n)})$

$f(n) = \log_b n$

TM M_R moves the head at most $p_1(n)$ times	Accepting string y
$\lvert y \rvert \le p_1(n)$	$p_2(\lvert y \rvert)$
	$p_2(p_1(n))$

$c^{f(n)} = c^{\log_b n}$ □ $(b^k)^{\log_b n} = b^{k \log_b n} = (b^{\log_b n})^k = n^k$ $p_1(n) + p_2(p_1(n))$

Complete and hard languages with respect to a language class

Complete and hard languages with respect to a language class

- **Transitivity of language reductions enables classification of languages in tightly connected langauge classes**

Complete and hard languages with respect to a language class

- **Transitivity of language reductions enables classification of languages in tightly connected langauge classes**

  - **If language *<u>L is in class K</u>* and all languages from *K* are *<u>polynomial-time</u>* reducible to language *L*, then language *L* is complete with respect to class *K* and polynomial-time reduction**

Complete and hard languages with respect to a language class

- **Transitivity of language reductions enables classification of languages in tightly connected langauge classes**

  - **If language *<u>L is in class K</u>* and all languages from *K* are *<u>polynomial-time</u>* reducible to language *L*, then language *L* is complete with respect to class *K* and polynomial-time reduction**

  - **If language *<u>L is in class K</u>* and all languages from *K* are *<u>log-space</u>* reducible to language *L*, then language *L* is complete with respect to class *K* and log-space reduction**

Complete and hard languages with respect to a language class

- **Transitivity of language reductions enables classification of languages in tightly connected langauge classes**

  - **If language *__L is in class K__* and all languages from *K* are *__polynomial-time__* reducible to language *L*, then language *L* is complete with respect to class *K* and polynomial-time reduction**

  - **If language *__L is in class K__* and all languages from *K* are *__log-space__* reducible to language *L*, then language *L* is complete with respect to class *K* and log-space reduction**

  - **If all languages from class *K* are *__polynomial-time__* reductible to language *L*, and language *L* *__is not necessarily in class K__*, then language *L* is hard with respect to class *K* and polynomial-time reduction**

Complete and hard languages with respect to a language class

- **Transitivity of language reductions enables classification of languages in tightly connected langauge classes**

  - **If language *__L is in class K__* and all languages from *K* are *__polynomial-time__* reducible to language *L*, then language *L* is complete with respect to class *K* and polynomial-time reduction**

  - **If language *__L is in class K__* and all languages from *K* are *__log-space__* reducible to language *L*, then language *L* is complete with respect to class *K* and log-space reduction**

  - **If all languages from class *K* are *__polynomial-time__* reductible to language *L*, and language *L* *__is not necessarily in class K__*, then language *L* is hard with respect to class *K* and polynomial-time reduction**

  - **If all languages from class *K* are *__log-space__* reducible to language L, and language *L* *__is not necessarily in class K__*, then language *L* is hard with respect to class *K* and log-space reduction**

Complete and hard languages with respect to a language class

Complete and hard languages with respect to a language class

- **Language *L* is *NP* -complete (*NP* -hard) if and only if *L* is complete (hard) with respect to class *NP* and polynomial-time reduction**

Complete and hard languages with respect to a language class

- Language *L* is ***NP*** **-complete** (***NP*** **-hard**) if and only if *L* is **complete** (**hard**) with respect to class ***NP*** and polynomial-time reduction

- Language *L* is ***NP*** **-complete** (***NP*** **-hard**) if and only if *L* is **complete** (**hard**) with respect to class ***NP*** and log-space reduction

Complete and hard languages with respect to a language class

- Language *L* is *NP*-complete (*NP*-hard) if and only if *L* is complete (hard) with respect to class *NP* and polynomial-time reduction

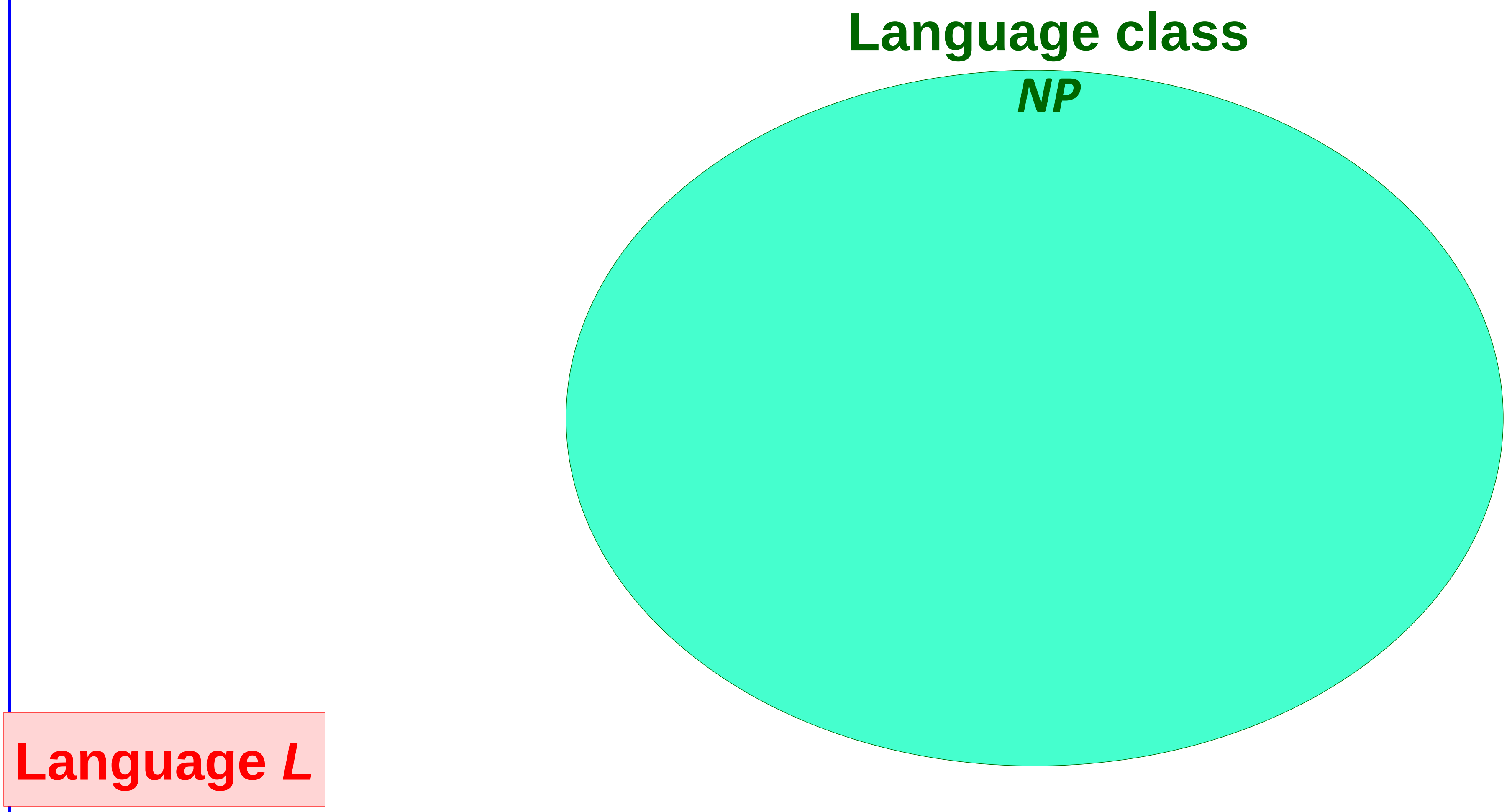
- Language *L* is *NP*-complete (*NP*-hard) if and only if *L* is complete (hard) with respect to class *NP* and log-space reduction

- Language *L* is PSPACE-complete (PSPACE-hard) if and only if *L* is complete (hard) with respect to class PSPACE and polynomial-time reduction

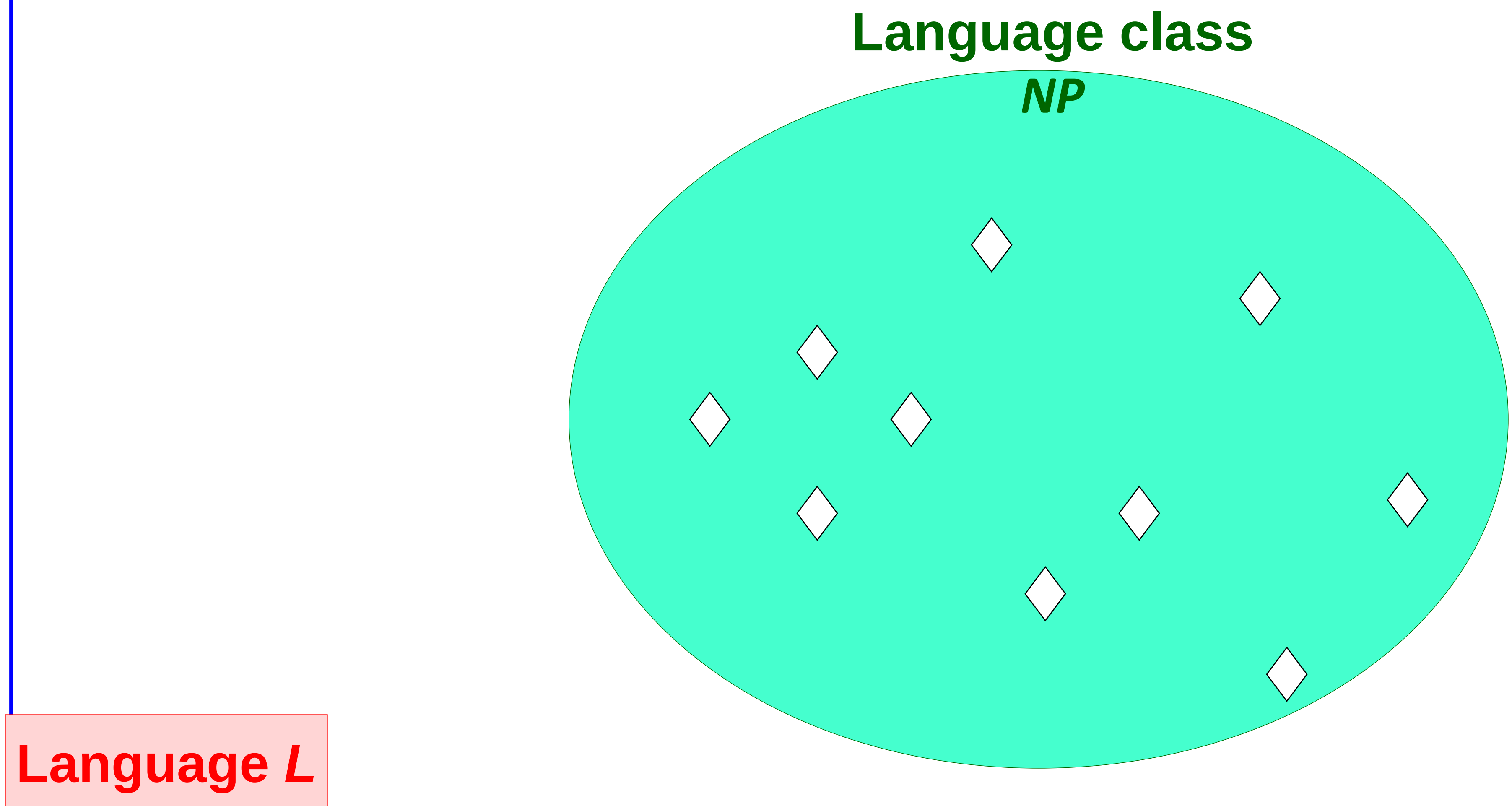
Complete and hard languages with respect to a language class

Language *L*

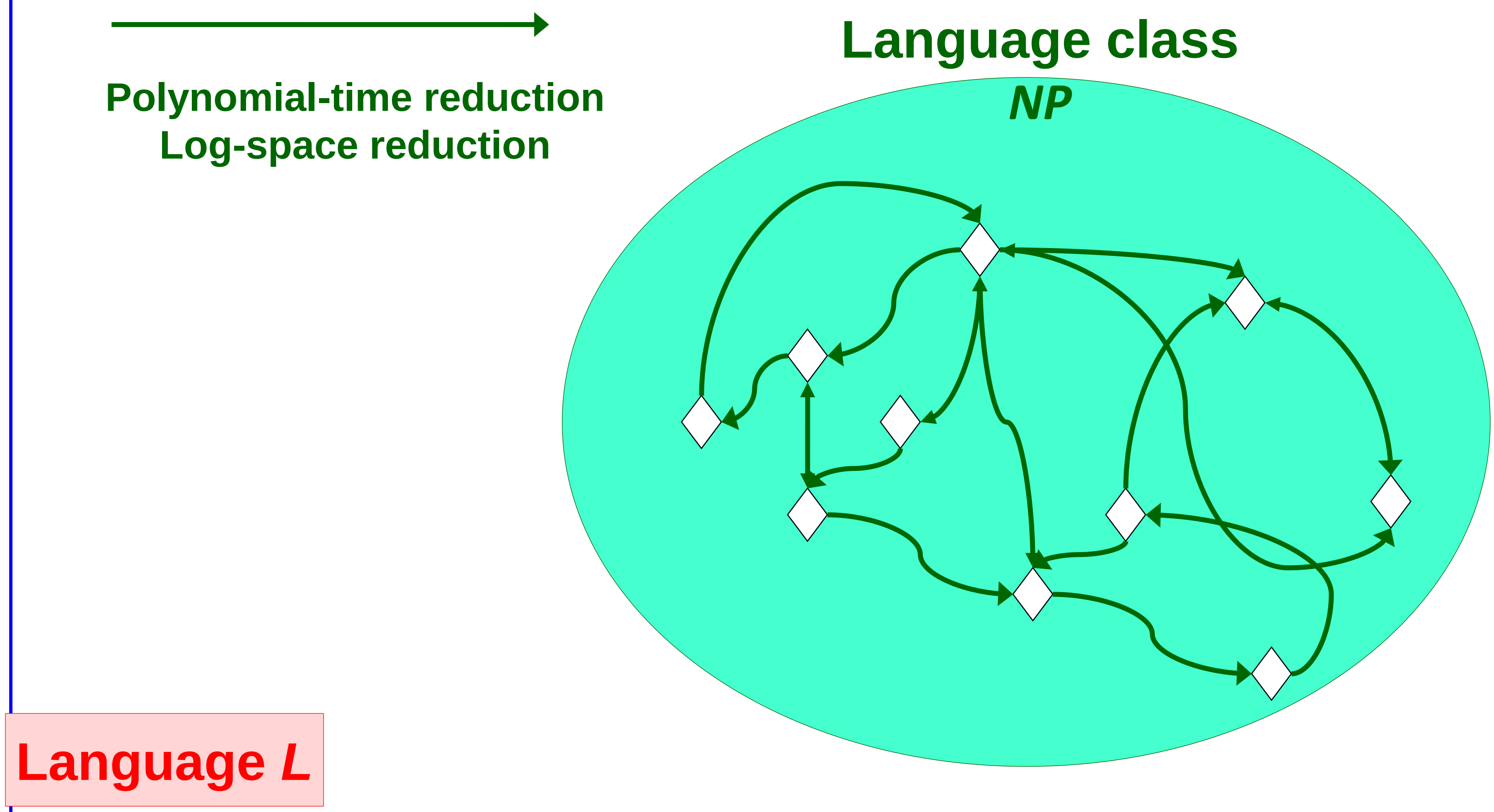
Complete and hard languages with respect to a language class



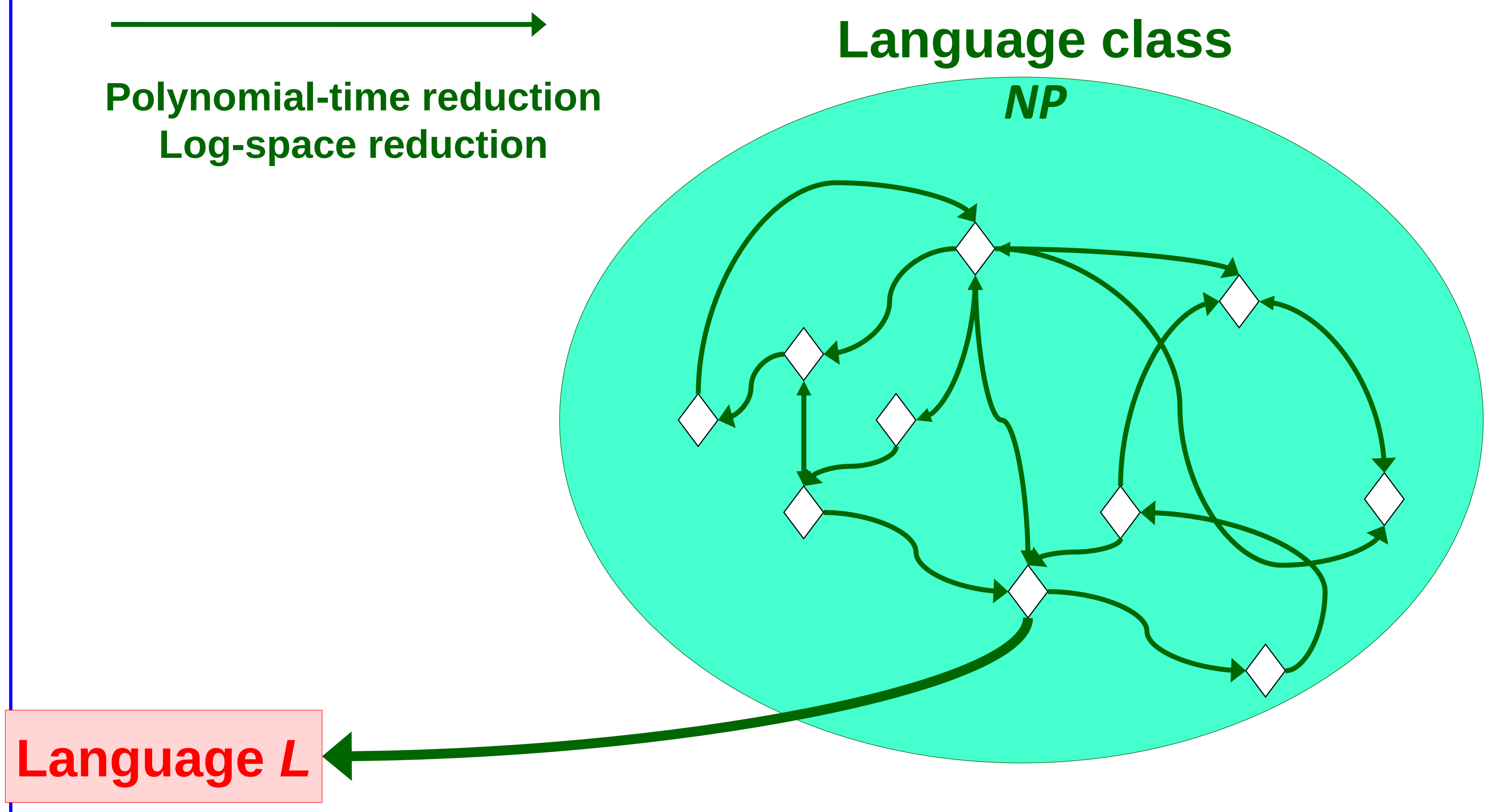
Complete and hard languages with respect to a language class



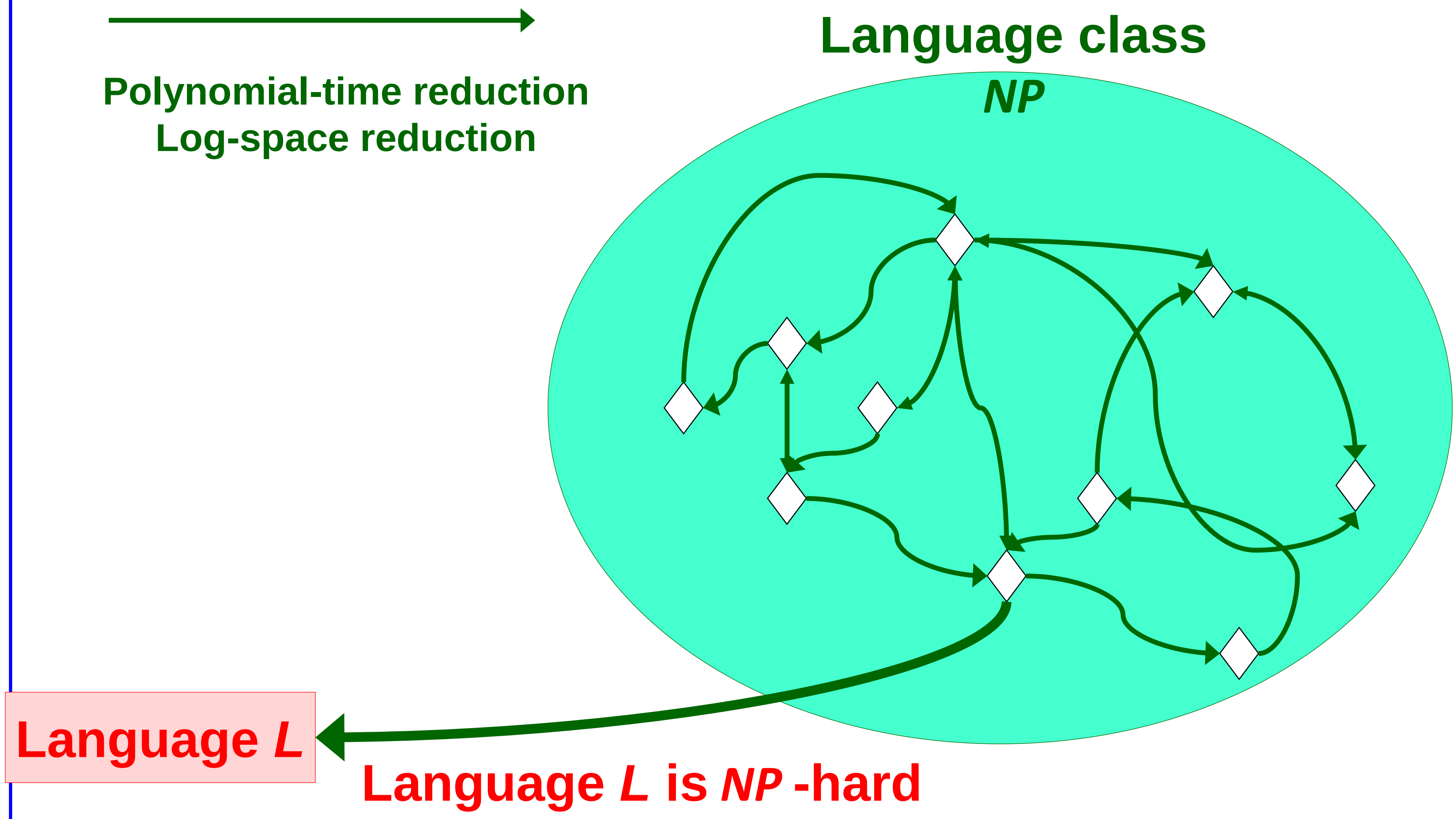
Complete and hard languages with respect to a language class



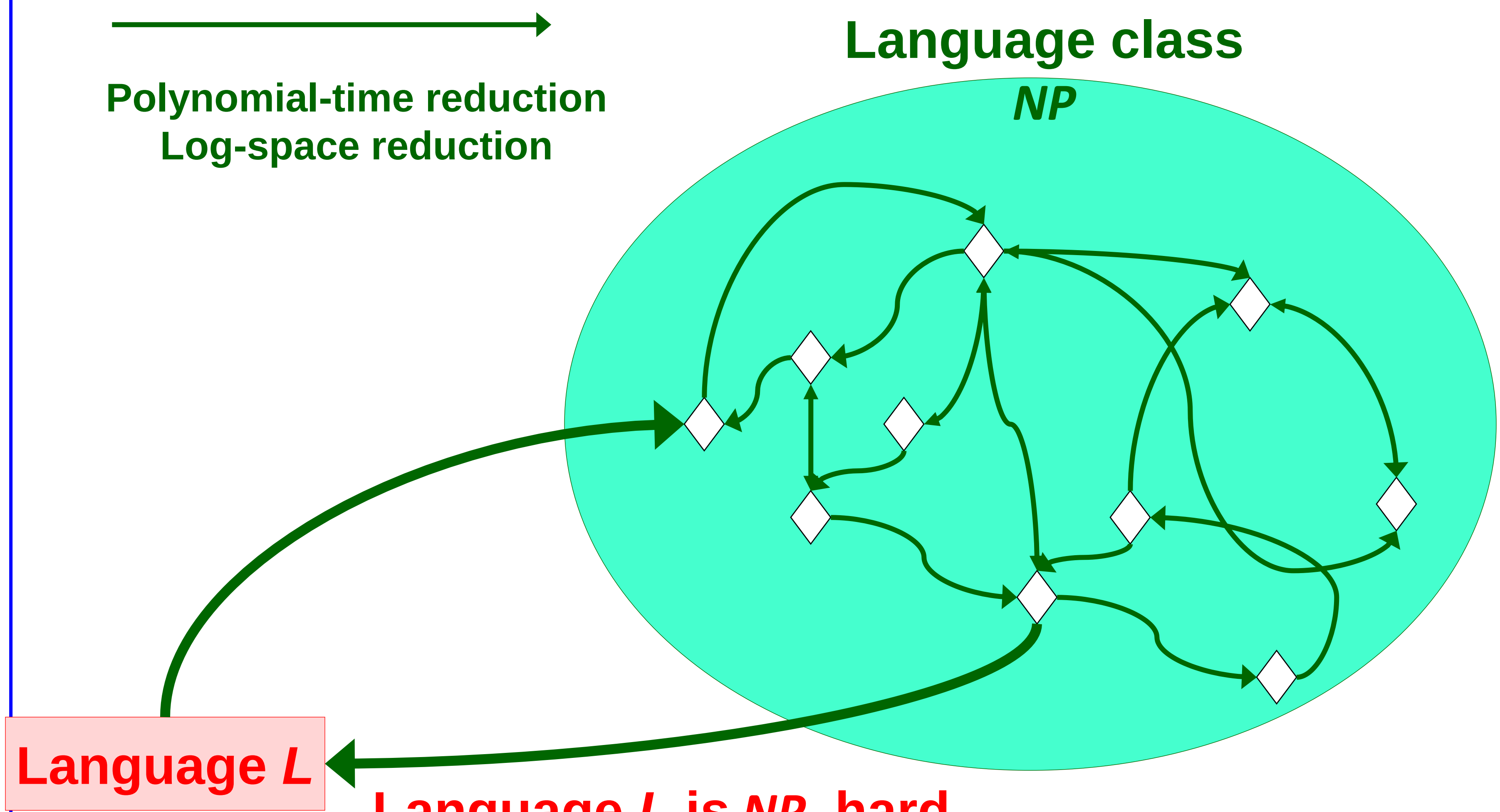
Complete and hard languages with respect to a language class



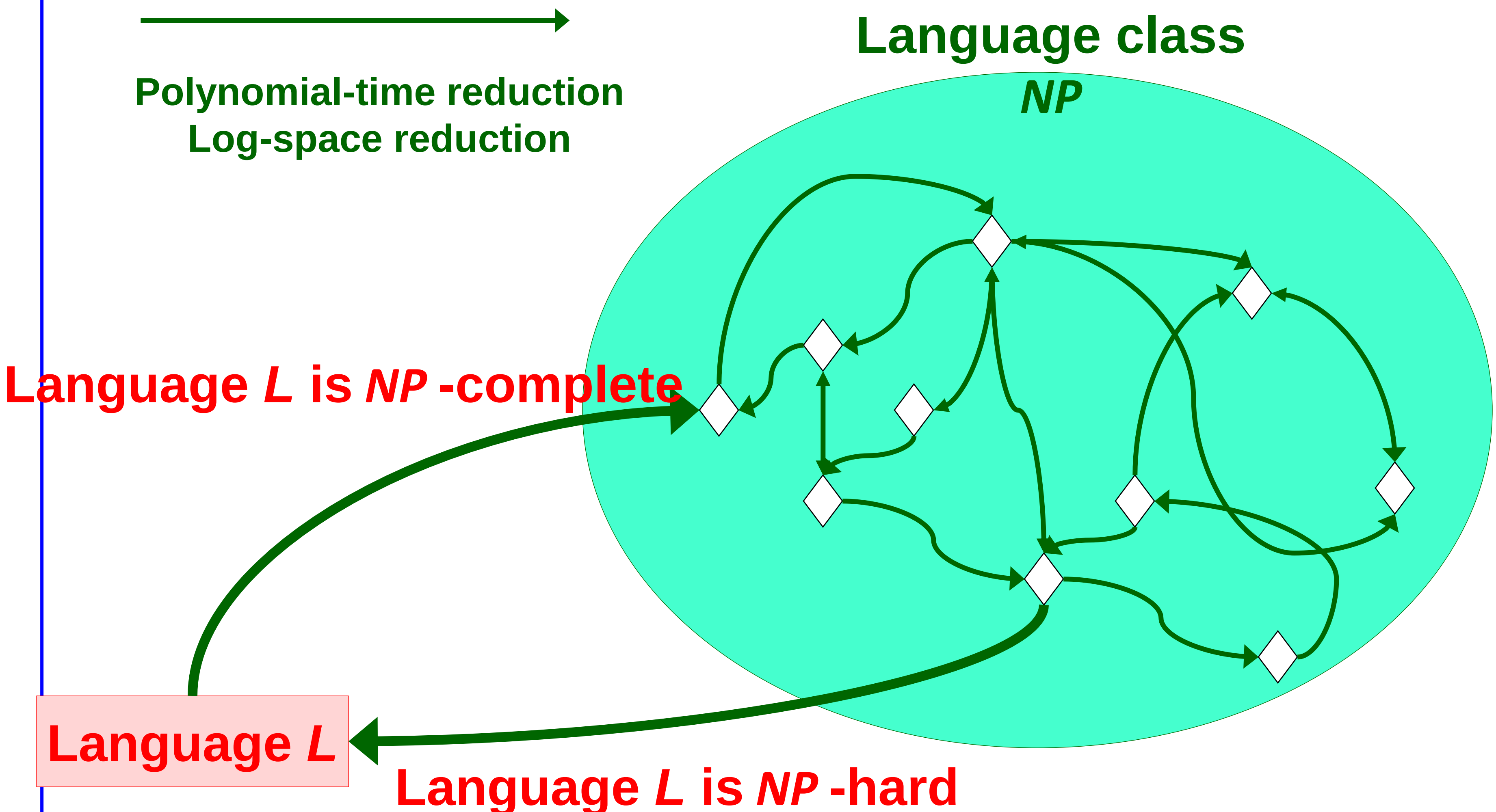
Complete and hard languages with respect to a language class



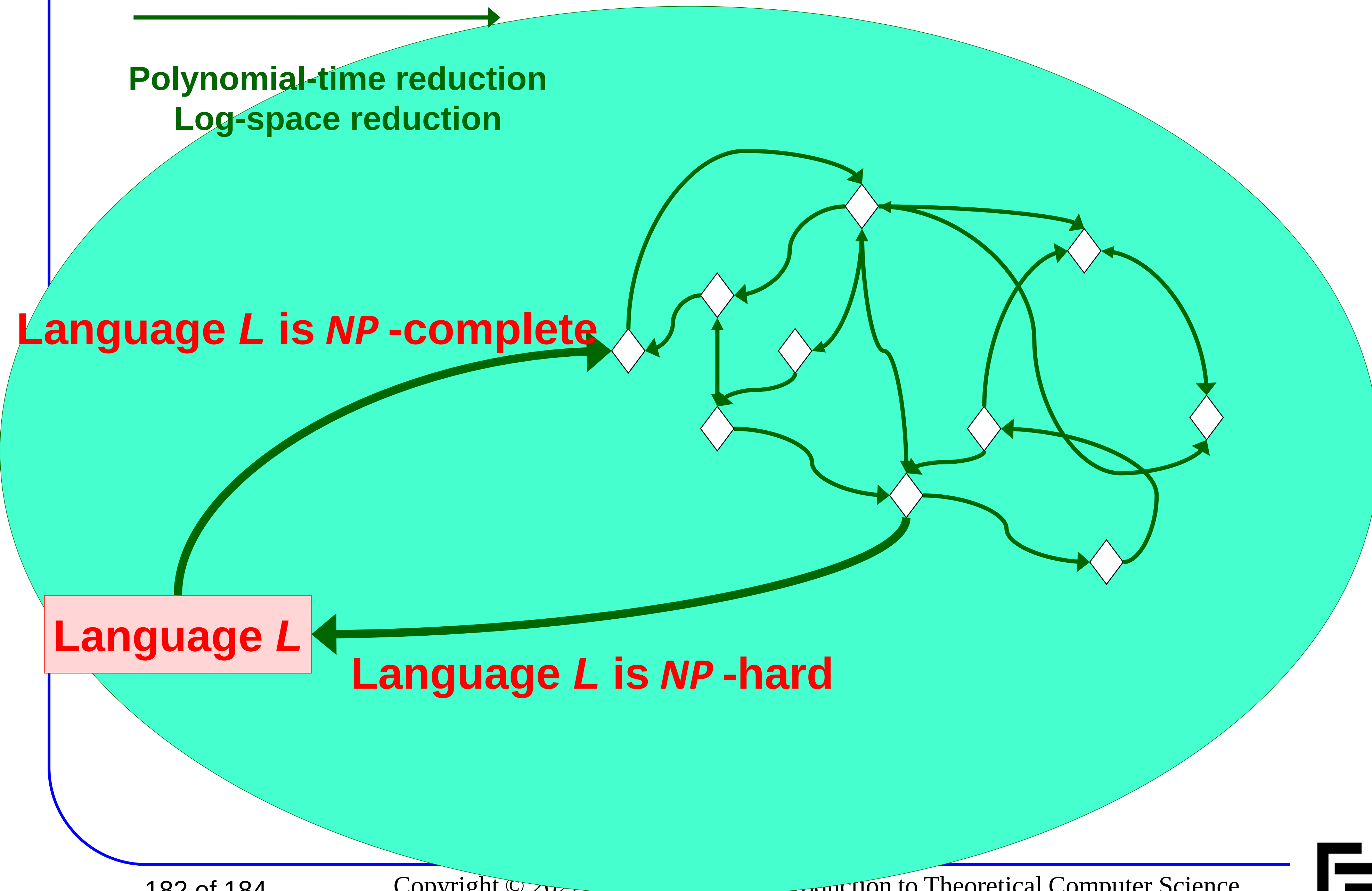
Complete and hard languages with respect to a language class



Complete and hard languages with respect to a language class



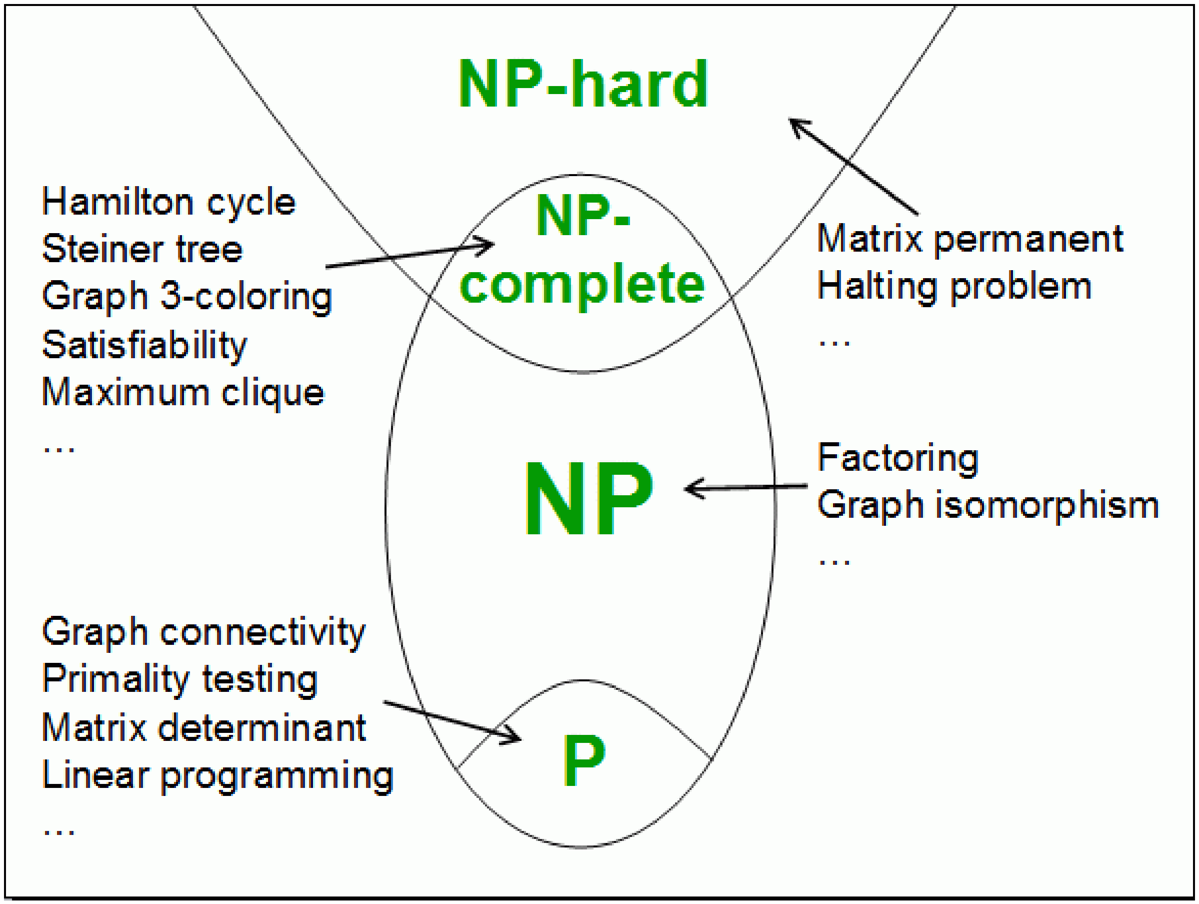
Complete and hard languages with respect to a language class



Informal definitions

- **P**
  - problems that can be solved in polynomial time
- **NP**
  - problems whose solution can be verified in polynomial time, and found in exponential time by generating all potential solutions
- **NP complete**
  - the most difficult problems in the NP class
  - they can be used to solve all other problems in NP
- **PSPACE**
  - problems that require a polynomial amount of memory

Some examples



Source: *http://naveenkandwal.blogspot.com/2015/01/p-np-np-complete-np-hard.html*