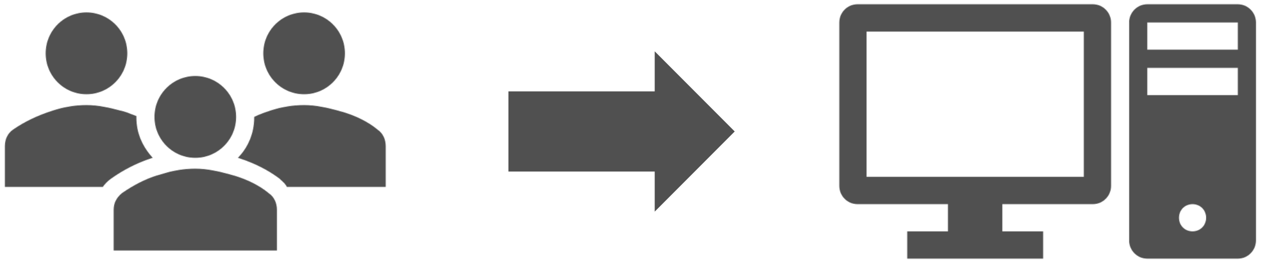
Jukka Suomela
Aalto University

Can we automate our own work

— or show that it is hard?

Computer science: ***what can be automated?***



Computer science: ***what can be automated?***

Today: ***can we automate our own work?***

Focus: ***theory of distributed computing***

Consider a typical theory paper in OPODIS, PODC, DISC…

Abstract

While the relationship of time and space is an estab
plexity theory, this is not the case in distributed com
the time and space complexity of algorithms in a weal
puting. While a constant number of communication
visited during the execution, the other direction is n
exist non-trivial graph problems that are solvable by
a non-constant running time. Somewhat surprisingly
of only cycle and path graphs. Our work provides us
computing and raises interesting questions about th
and space complexity.

1998 ACM Subject Classification F.1.1 Models of
and Classes

Keywords and phrases distributed computing, spa
weak models, Thue–Morse sequence

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2

1 Introduction

In the classical centralised theory of computing, th
helped us with understanding computability and c

1. Constant-space models (finite-state machines) p
ation.
2. Space-limited complexity classes (e.g., PSPAC
limited complexity classes (e.g., NP ⊆ PSPACE

Consider a typical theory paper in OPODIS, PODC, DISC…

How much of the work is done with computers?

Abstract

While the relationship of time and space is an estab
plexity theory, this is not the case in distributed com
the time and space complexity of algorithms in a weal
puting. While a constant number of communication
visited during the execution, the other direction is n
exist non-trivial graph problems that are solvable by
a non-constant running time. Somewhat surprisingly
of only cycle and path graphs. Our work provides us
computing and raises interesting questions about th
and space complexity.

1998 ACM Subject Classification F.1.1 Models of
and Classes

Keywords and phrases distributed computing, spa
weak models, Thue–Morse sequence

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2

1 Introduction

In the classical centralised theory of computing, th
helped us with understanding computability and c

1. Constant-space models (finite-state machines) p
ation.
2. Space-limited complexity classes (e.g., PSPAC
limited complexity classes (e.g., NP ⊆ PSPACE

Consider a typical theory paper in OPODIS, PODC, DISC…

How much of the work is done with computers?

How much of it could be done with computers?

Abstract

While the relationship of time and space is an estab
plexity theory, this is not the case in distributed com
the time and space complexity of algorithms in a weal
puting. While a constant number of communication
visited during the execution, the other direction is n
exist non-trivial graph problems that are solvable by
a non-constant running time. Somewhat surprisingly
of only cycle and path graphs. Our work provides us
computing and raises interesting questions about th
and space complexity.

1998 ACM Subject Classification F.1.1 Models of
and Classes

Keywords and phrases distributed computing, spa
weak models, Thue–Morse sequence

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2

1 Introduction

In the classical centralised theory of computing, th
helped us with understanding computability and c

1. Constant-space models (finite-state machines) p
ation.
2. Space-limited complexity classes (e.g., PSPAC
limited complexity classes (e.g., NP ⊆ PSPACE

Standard process

- **Question:** is there an efficient distributed algorithm for solving task X in model M?

Standard process

- **Question:** is there an efficient distributed algorithm for solving task X in model M?
- **Approach:** find smart people, spend lots of time in front of a whiteboard …

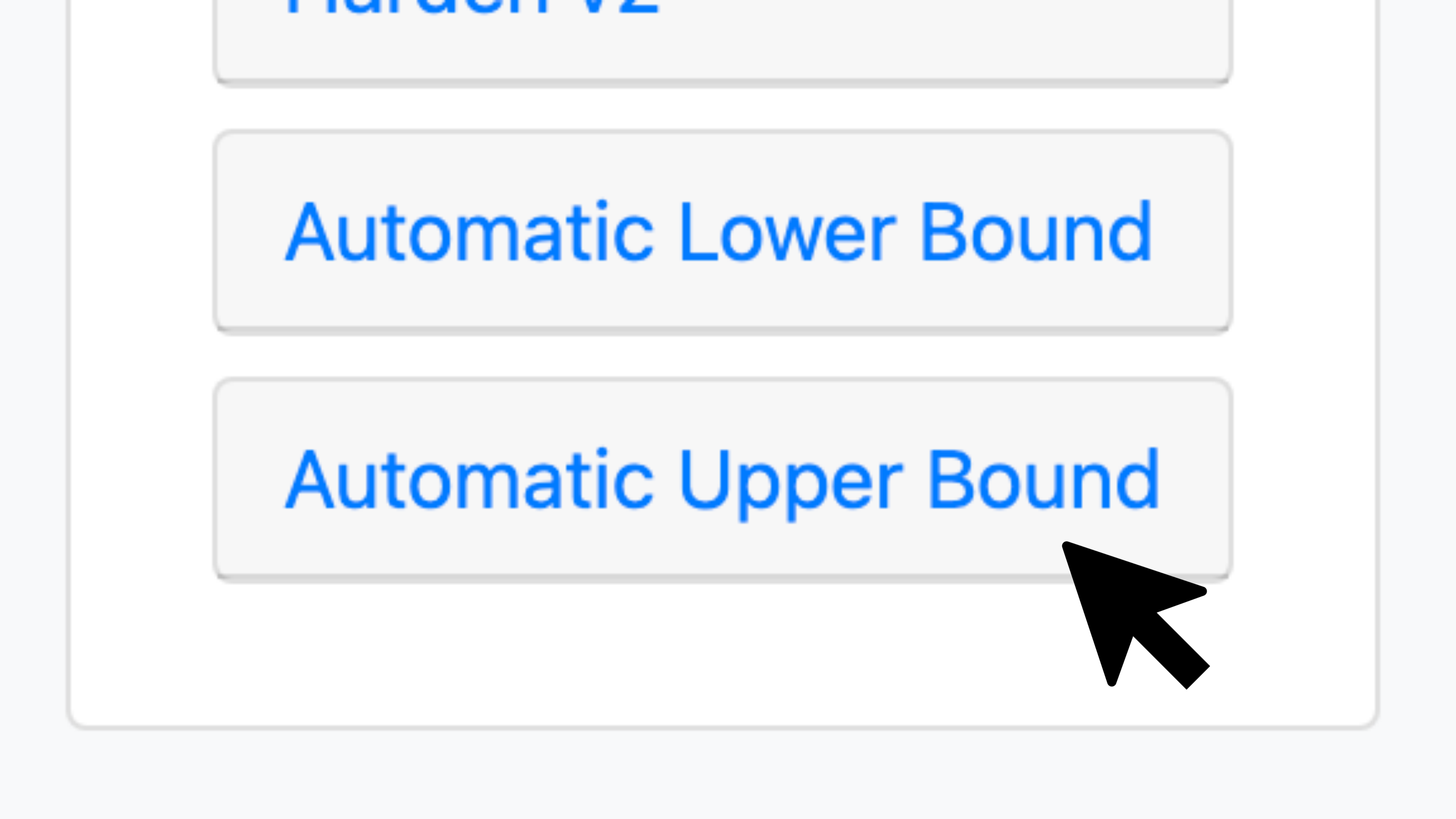
Standard process

- **Question:** is there an efficient distributed algorithm for solving task X in model M?
- **Approach:** find smart people, spend lots of time in front of a whiteboard …
- **End result:** algorithm, algorithm analysis, proof of correctness, lower bound proof …

Standard process

- **Question:** is there an efficient distributed algorithm for solving task *X* in model *M*?
- **Approach:** ~~find smart people, spend lots of time in front of a whiteboard …~~

- **End result:** algorithm, algorithm analysis, proof of correctness, lower bound proof …

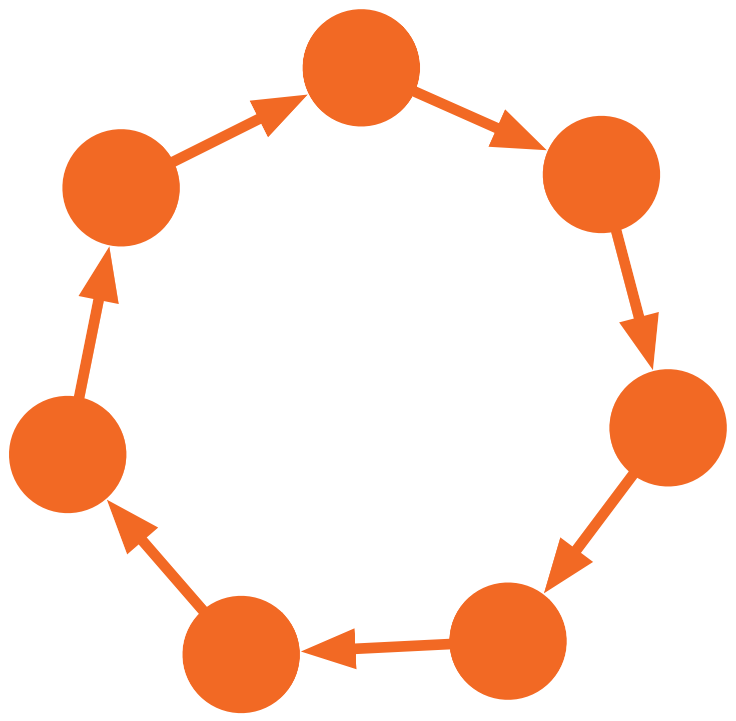

Automatic Lower Bound
Automatic Upper Bound

Lost sanity?

Toy example:
Locally checkable problems in cycles

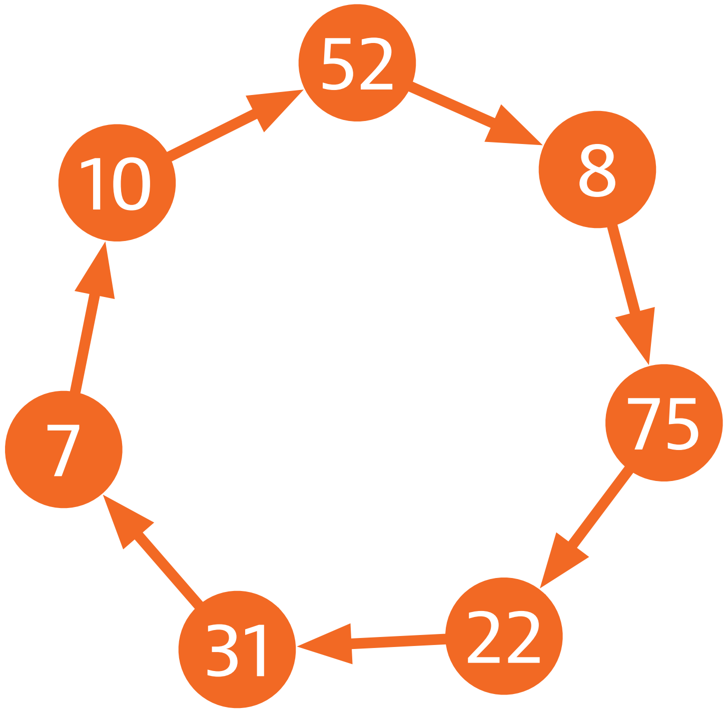
Setting

- ***Computer network:*** **cycle** of *n* computers
  - globally consistent orientation
  - each node has one “successor” and one “predecessor”



Setting

- ***Computer network:*** **cycle** of n computers
- ***Model of computing:*** **LOCAL model**
  - synchronous communication rounds
  - time = number of rounds until all nodes stop
  - unbounded message size
  - unlimited local computation
  - unique identifiers



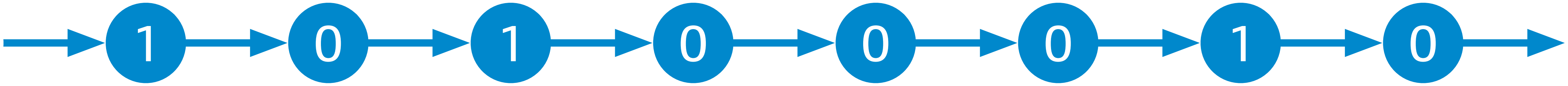
Setting

- ***Computer network:*** **cycle** of *n* computers
- ***Model of computing:*** **LOCAL model**
- ***Problem:*** any discrete problem you can define with **local constraints**
  - finite number of output labels
  - relation that tells which label sequences are valid

Setting

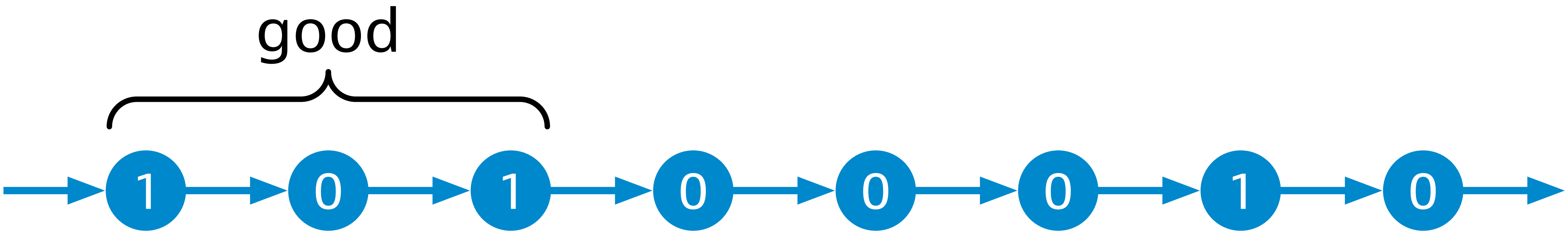
- ***Computer network:*** **cycle** of n computers
- ***Model of computing:*** **LOCAL model**
- ***Problem:*** any discrete problem you can define with **local constraints**

Example: maximal independent set



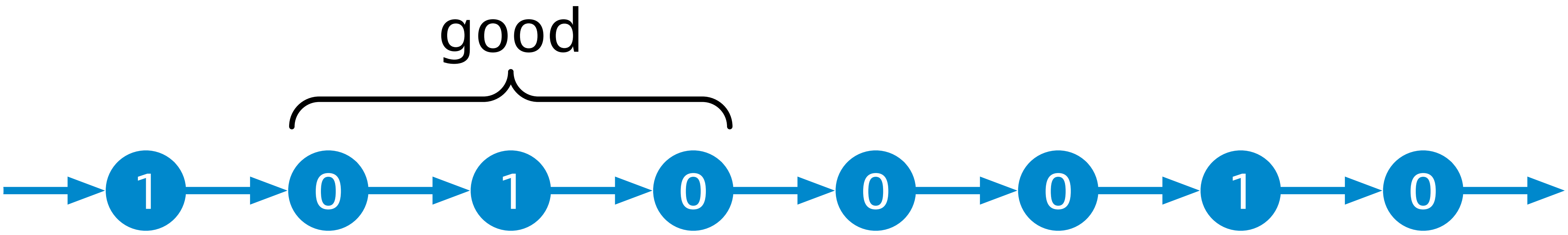
Setting

- ***Computer network:*** **cycle** of n computers
- ***Model of computing:*** **LOCAL model**
- ***Problem:*** any discrete problem you can define with **local constraints**



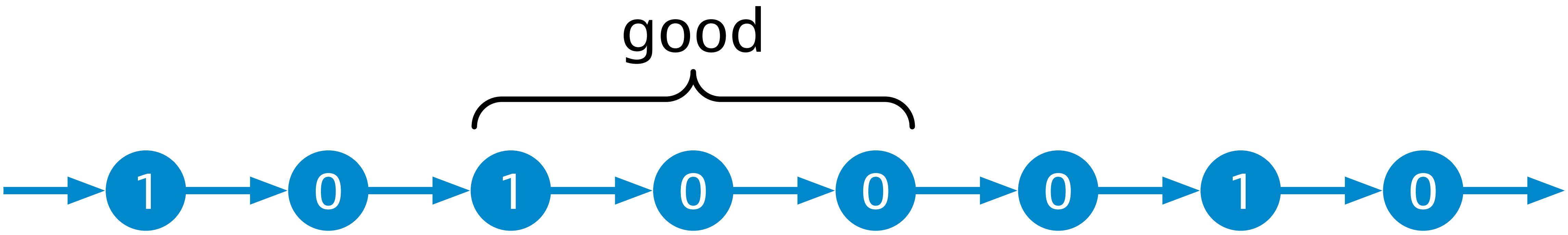
Setting

- ***Computer network:*** **cycle** of n computers
- ***Model of computing:*** **LOCAL model**
- ***Problem:*** any discrete problem you can define with **local constraints**



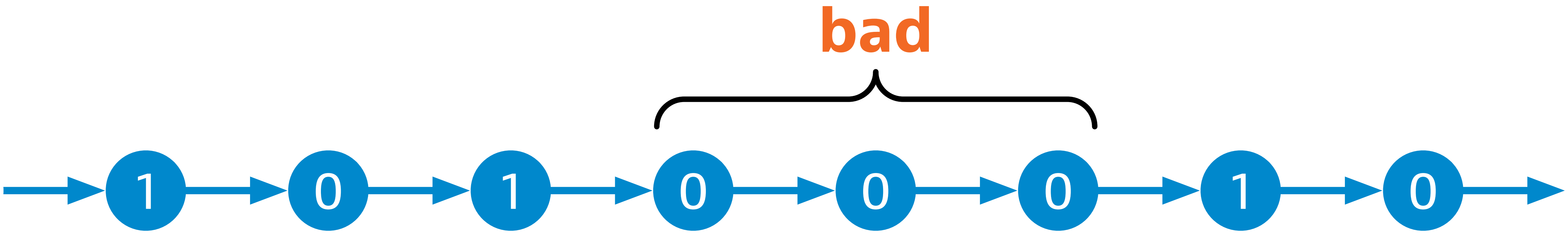
Setting

- ***Computer network:*** **cycle** of n computers
- ***Model of computing:*** **LOCAL model**
- ***Problem:*** any discrete problem you can define with **local constraints**



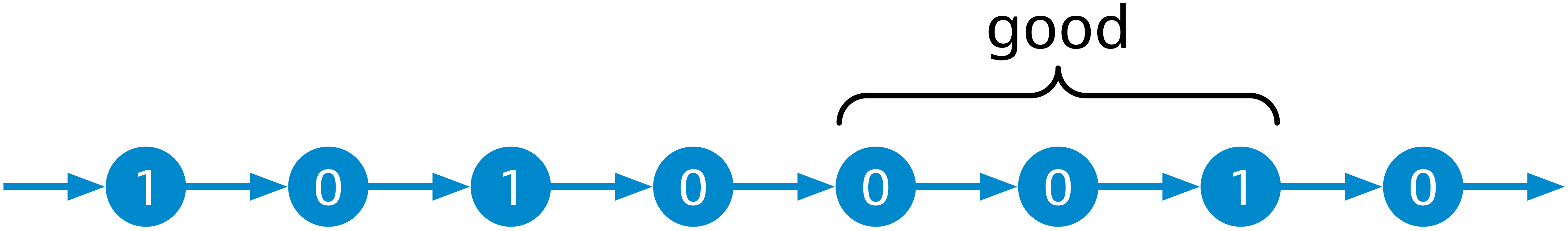
Setting

- ***Computer network:*** **cycle** of *n* computers
- ***Model of computing:*** **LOCAL model**
- ***Problem:*** any discrete problem you can define with **local constraints**



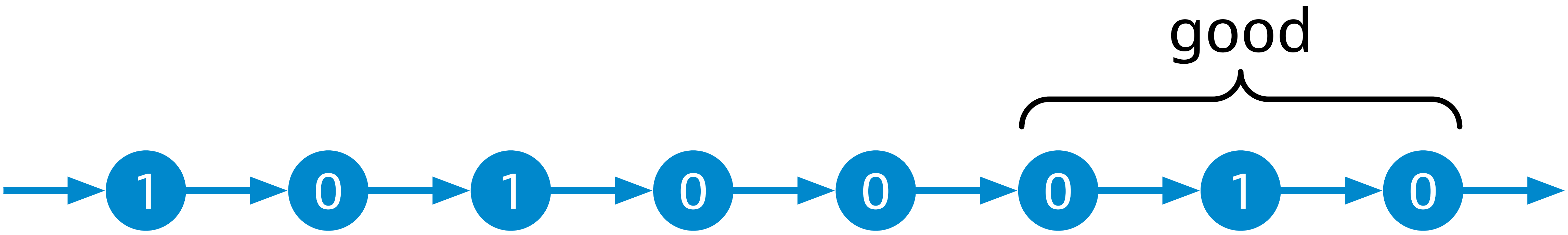
Setting

- ***Computer network:*** **cycle** of n computers
- ***Model of computing:*** **LOCAL model**
- ***Problem:*** any discrete problem you can define with **local constraints**



Setting

- ***Computer network:*** **cycle** of n computers
- ***Model of computing:*** **LOCAL model**
- ***Problem:*** any discrete problem you can define with **local constraints**



Valid label sequences

- *2-coloring:* 12, 21
- *3-coloring:* 12, 21, 13, 31, 23, 32
- *Independent set:* 01, 10, 00
- *Maximal independent set:* 001, 010, 100, 101
- *Distance-2 coloring with 3 colors:*
  123, 132, 213, 231, 312, 321

Valid label sequences

- *2-coloring:* 12, 21
- *3-coloring:* 12, 21, 13, 31, 23, 32
- *Independent set:* 01, 10, 00
- *Maximal independent set:* 001, 010, 100, 101
- *Distance-2 coloring with 3 colors:* 123, 132, 213, 231, 312, 321

All possible output labelings in a window of size *k*

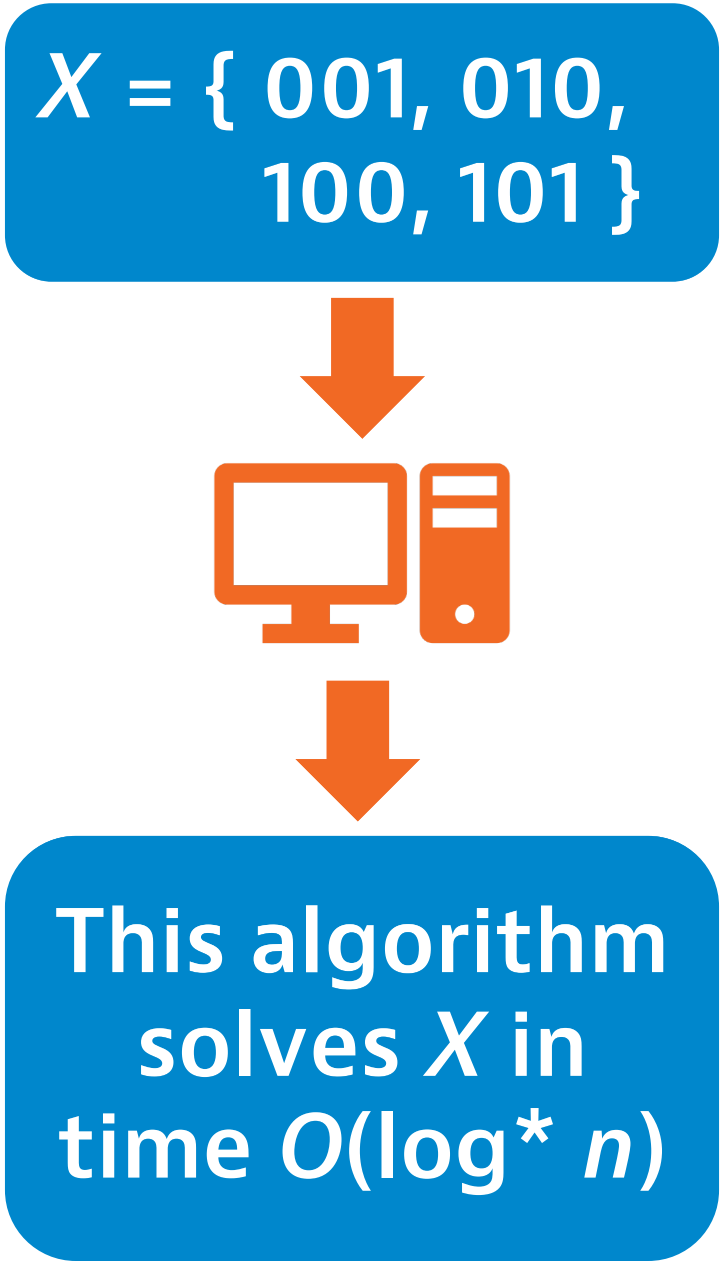
Fully automatic

X = { 001, 010, 100, 101 }

- Write down the specification of *any locally checkable problem X*

Fully automatic

- Write down the specification of ***any locally checkable problem X***
- Then you can ***find efficiently***
  - distributed round complexity of X
  - asymptotically optimal distributed algorithm for X



Fully automatic

- Write down the specification of ***any locally checkable problem X***
- Then you can ***find efficiently***
  - distributed round complexity of X
  - asymptotically optimal distributed algorithm for X

Polynomial time (in the size of problem description)

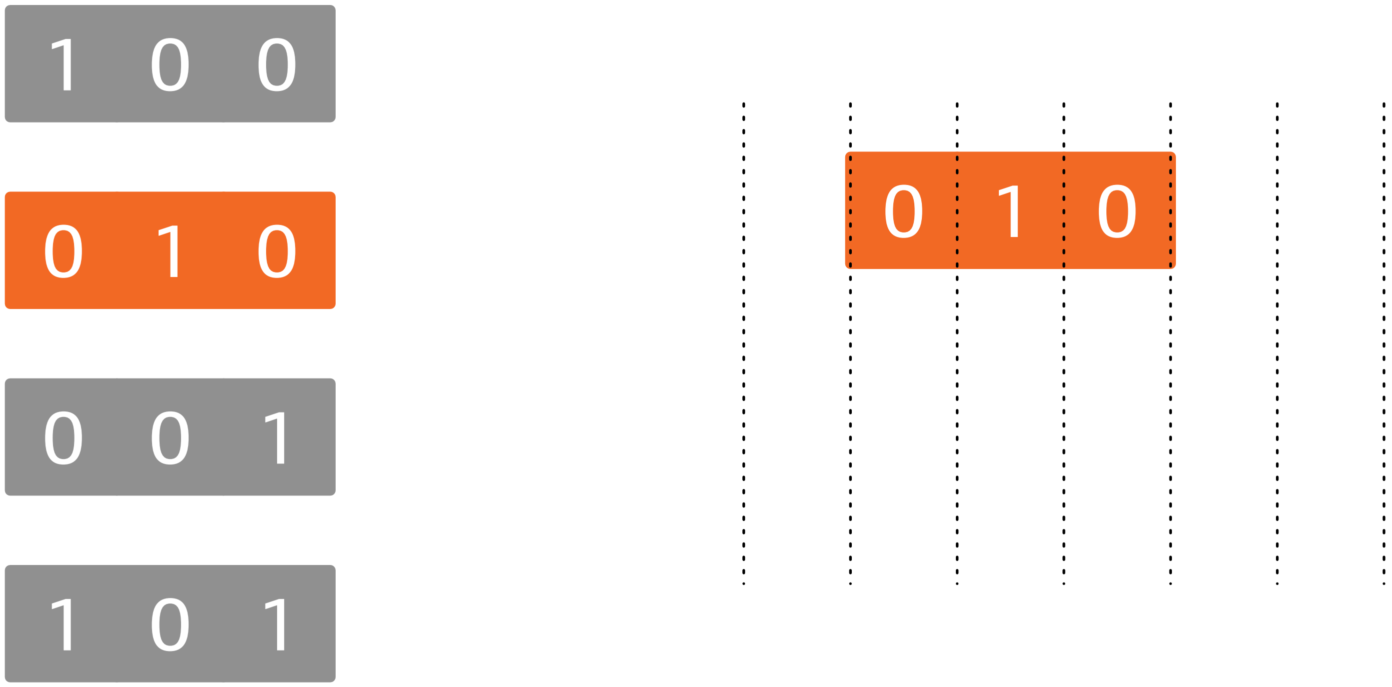
1 0 0

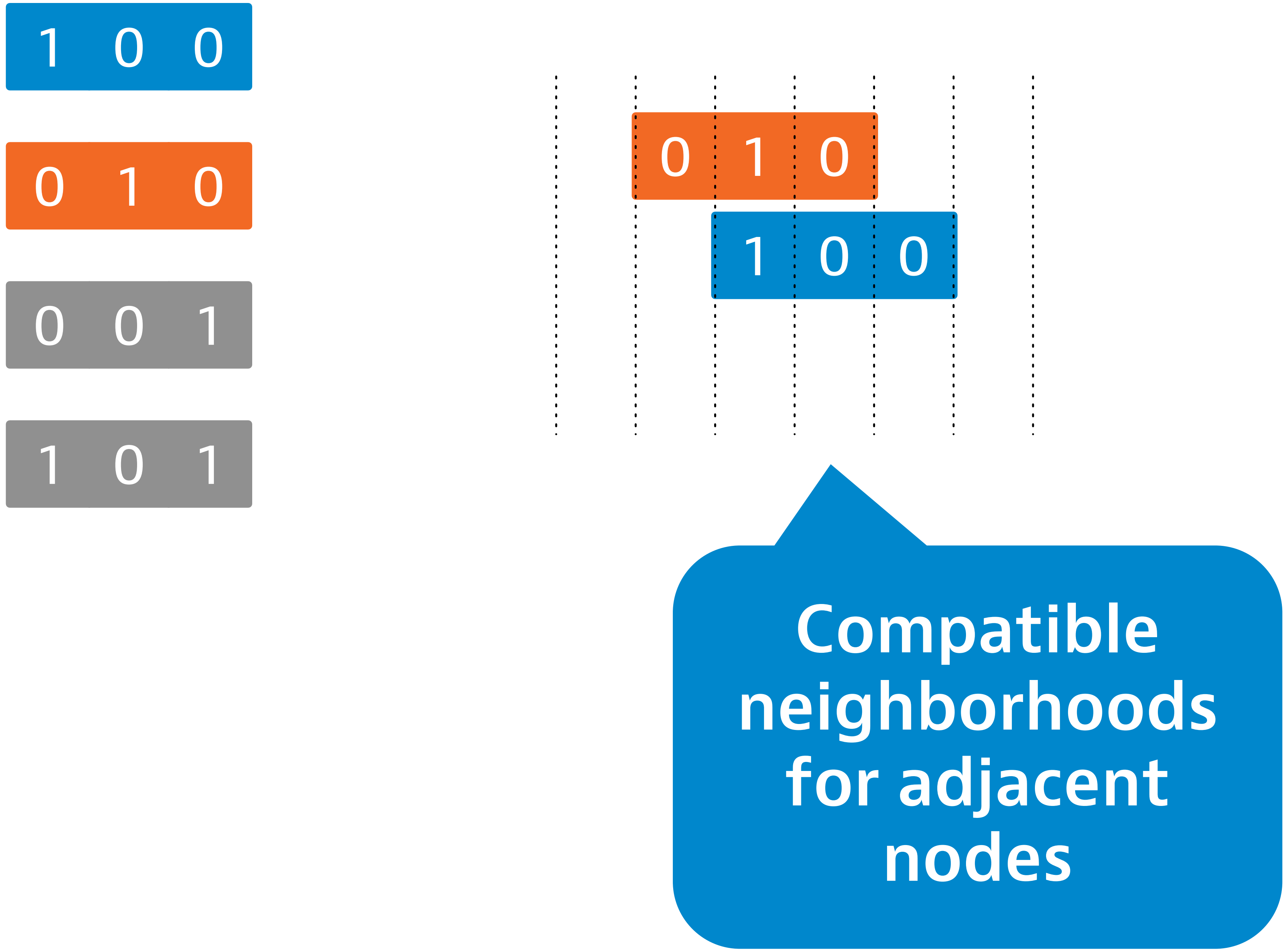
0 1 0

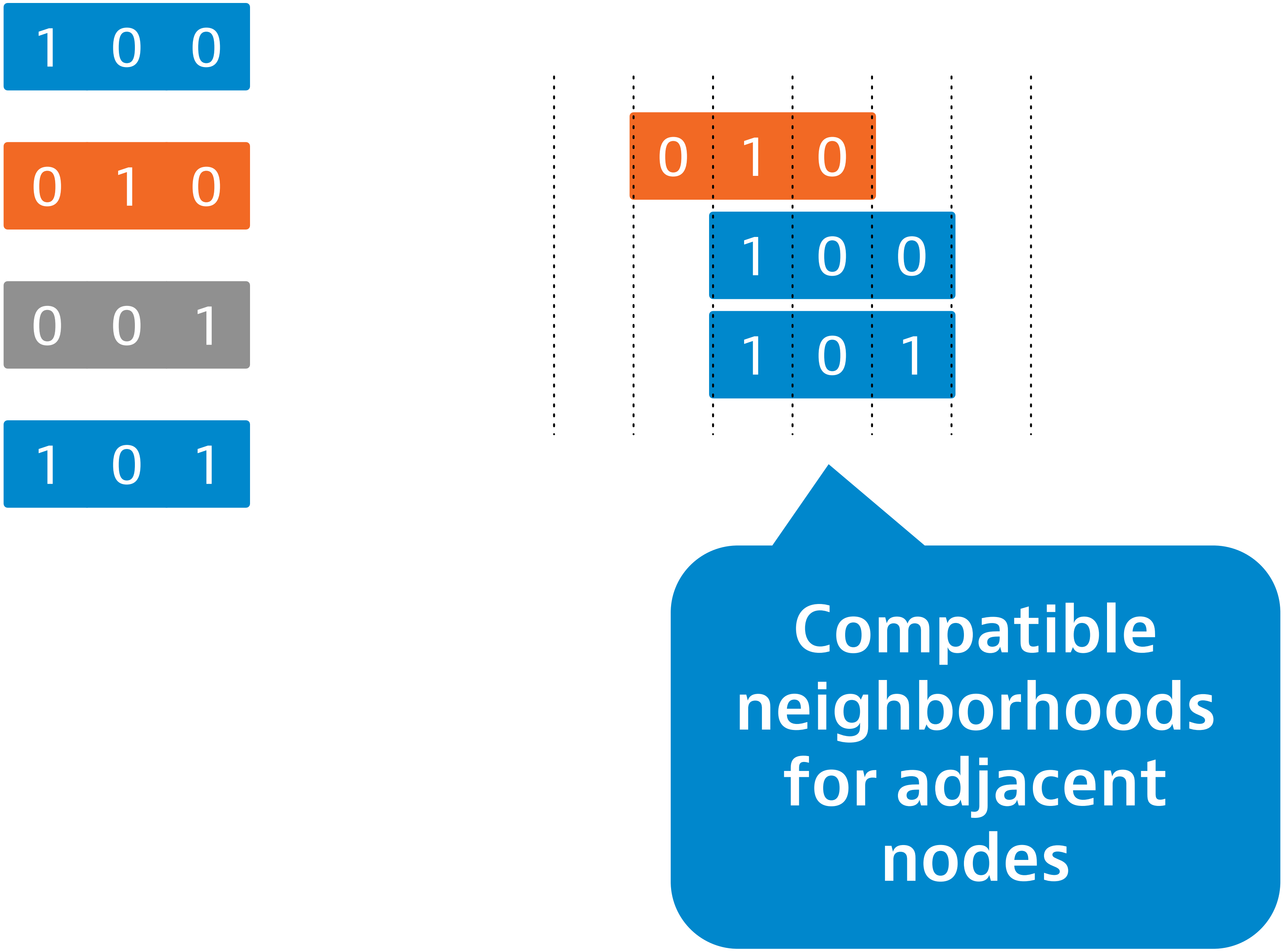
0 0 1

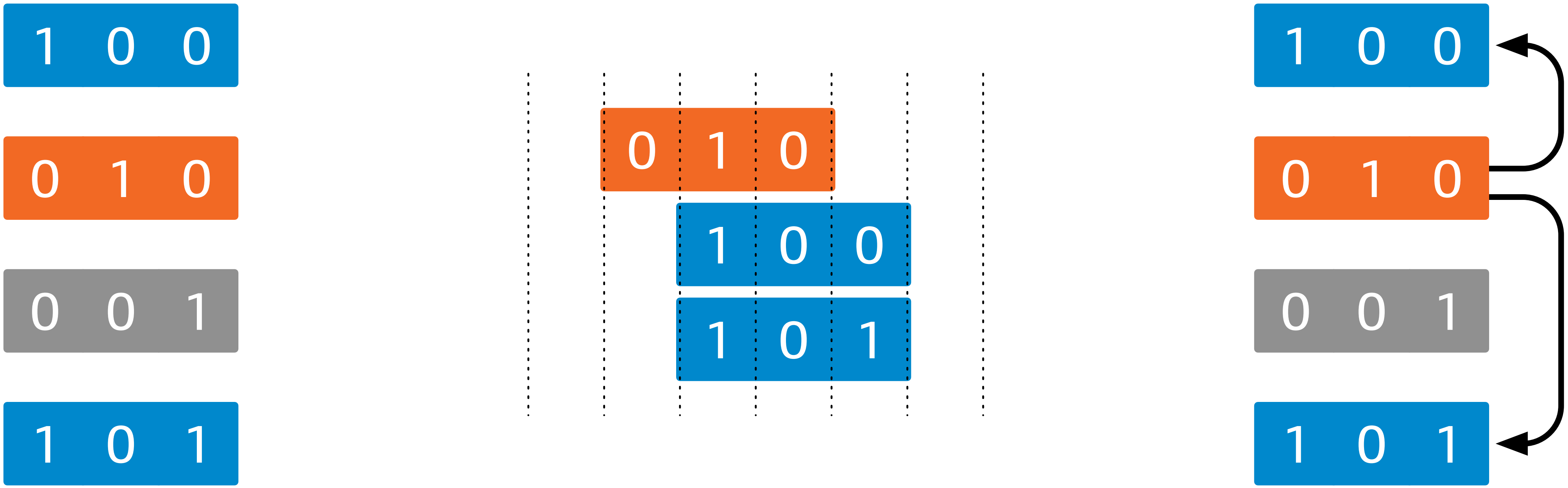
1 0 1

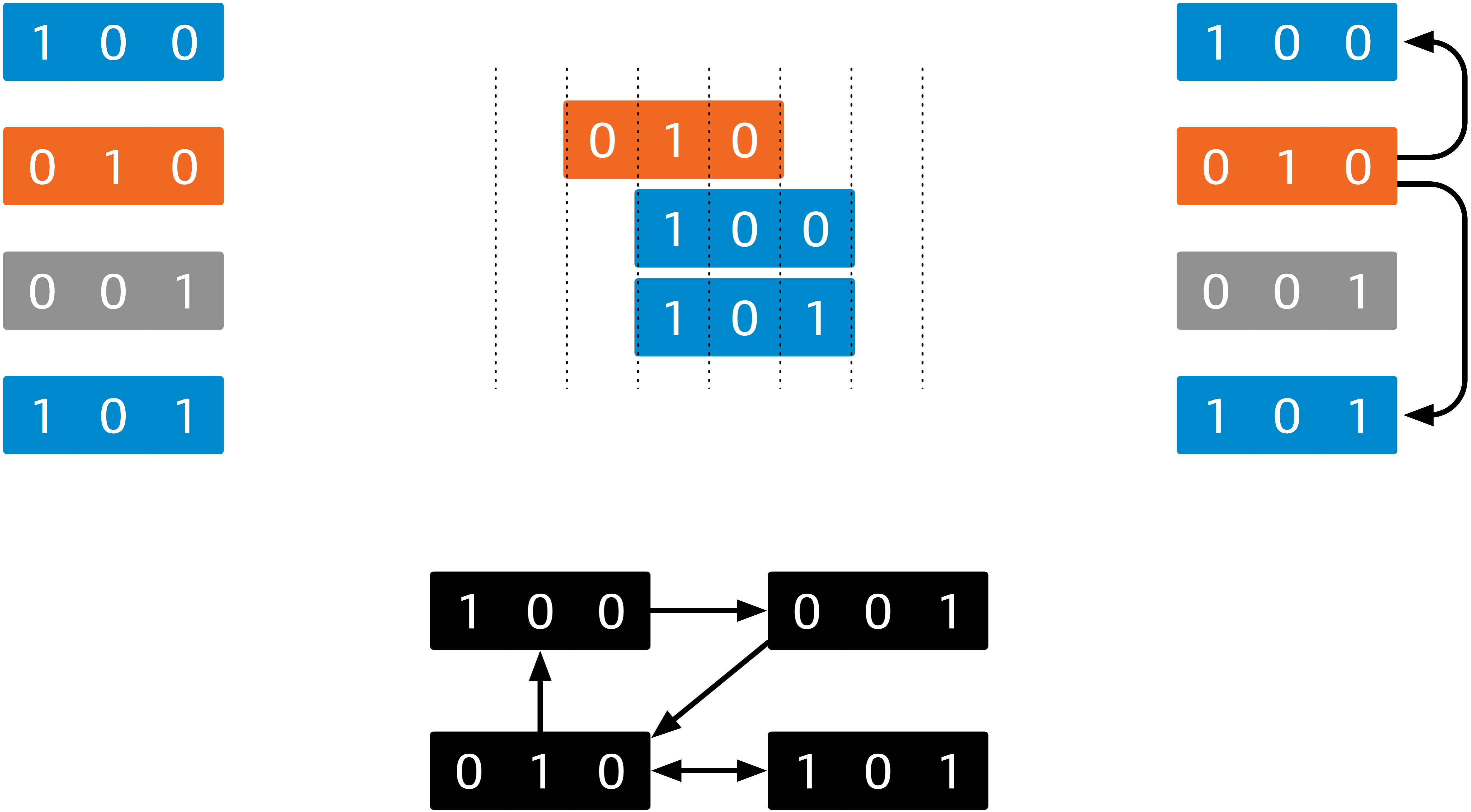
Example:
X = maximal independent set problem

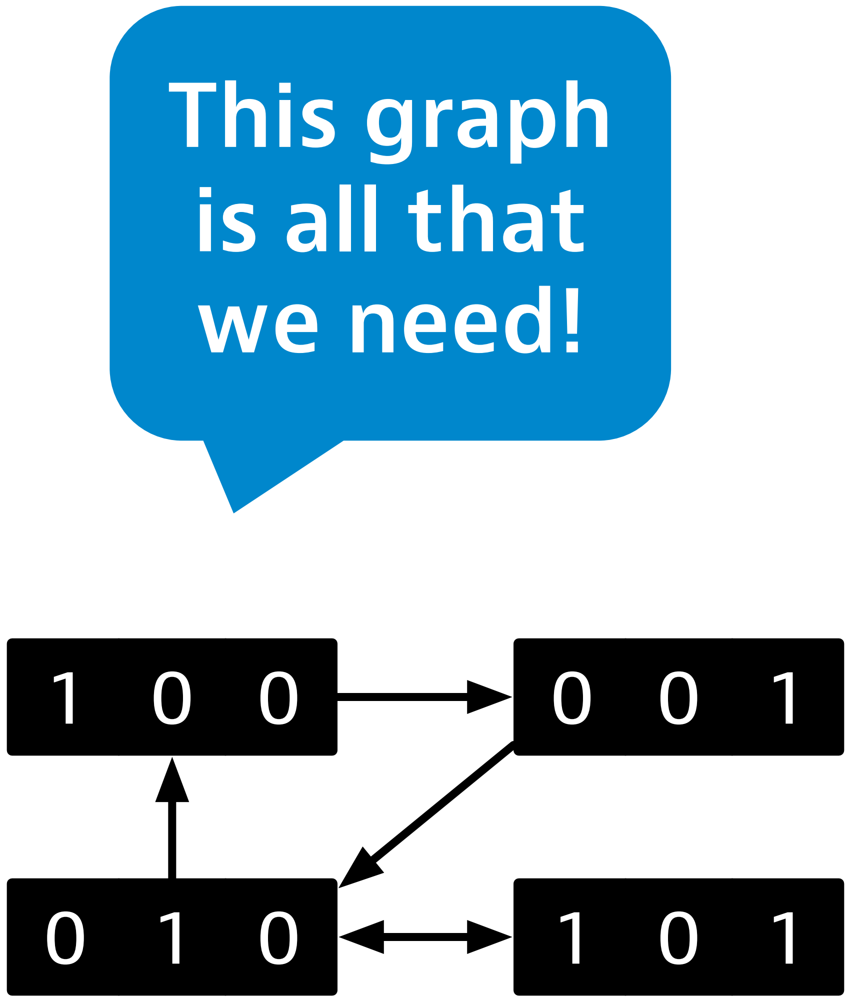

1 0 0
0 1 0
0 0 1
1 0 1
0 1 0

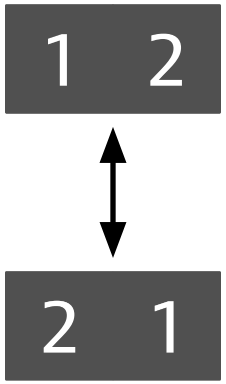

1 0 0
0 1 0
0 0 1
1 0 1
0 1 0
1 0 0
Compatible neighborhoods for adjacent nodes


1 0 0
0 1 0
0 0 1
1 0 1
0 1 0
1 0 0
1 0 1
Compatible neighborhoods for adjacent nodes

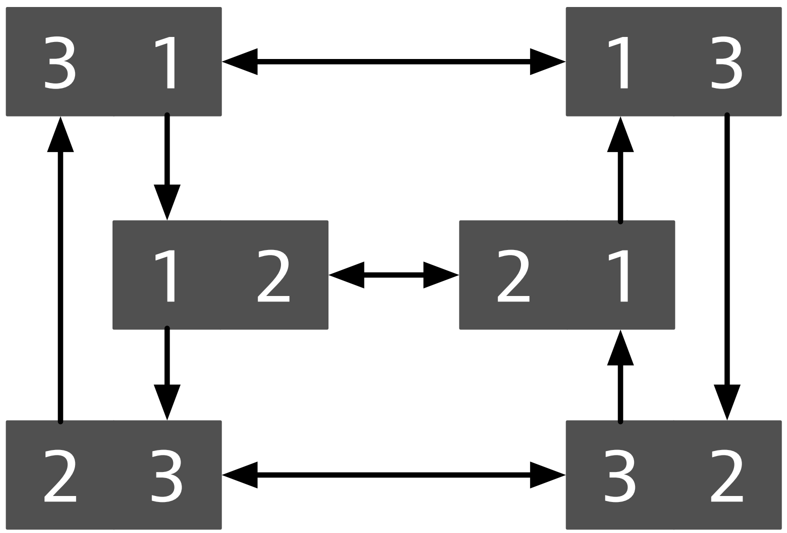

1 0 0
0 1 0
0 0 1
1 0 1
0 1 0
1 0 0
1 0 1
1 0 0
0 1 0
0 0 1
1 0 1


1 0 0
0 1 0
0 0 1
1 0 1
0 1 0
1 0 0
1 0 1
1 0 0
0 1 0
0 0 1
1 0 1
1 0 0
0 0 1
0 1 0
1 0 1

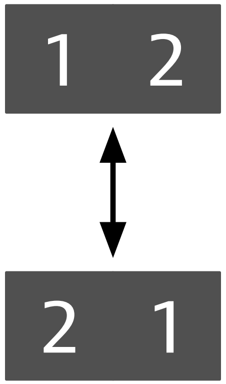

This graph is all that we need!
1 0 0
0 0 1
0 1 0
1 0 1



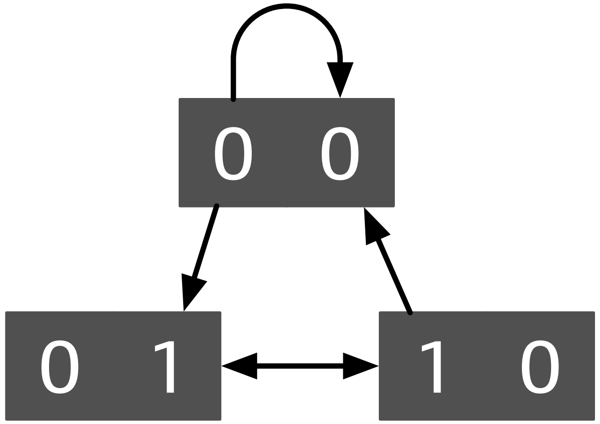
2-coloring



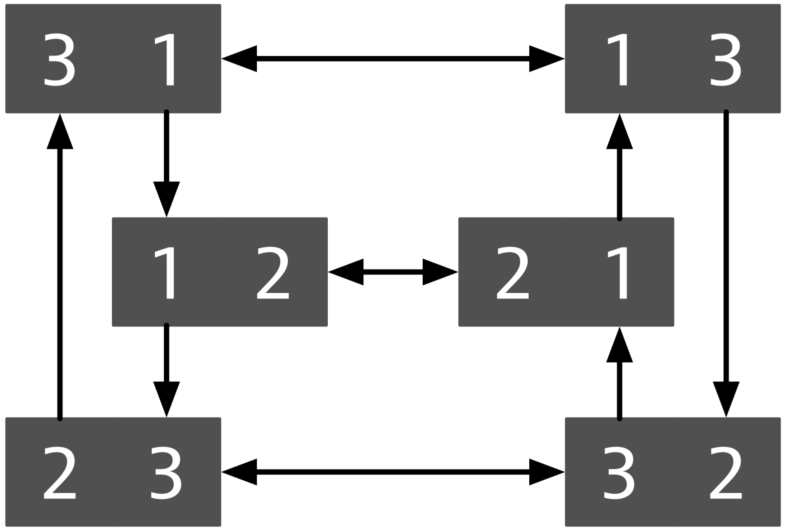
3-coloring



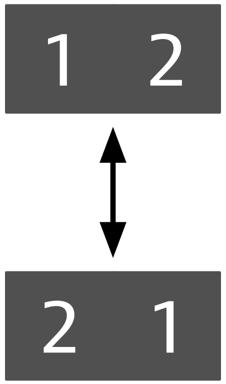
2-coloring



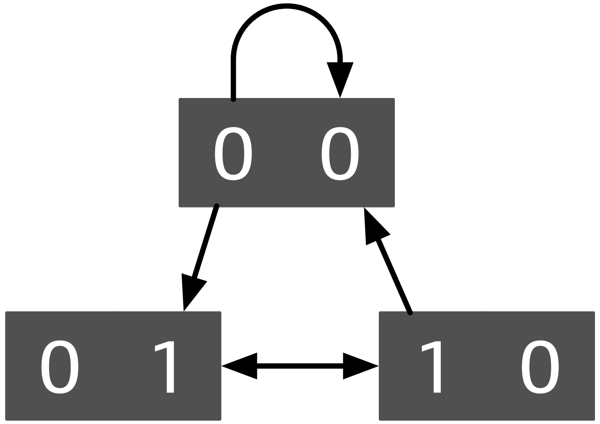
independent set



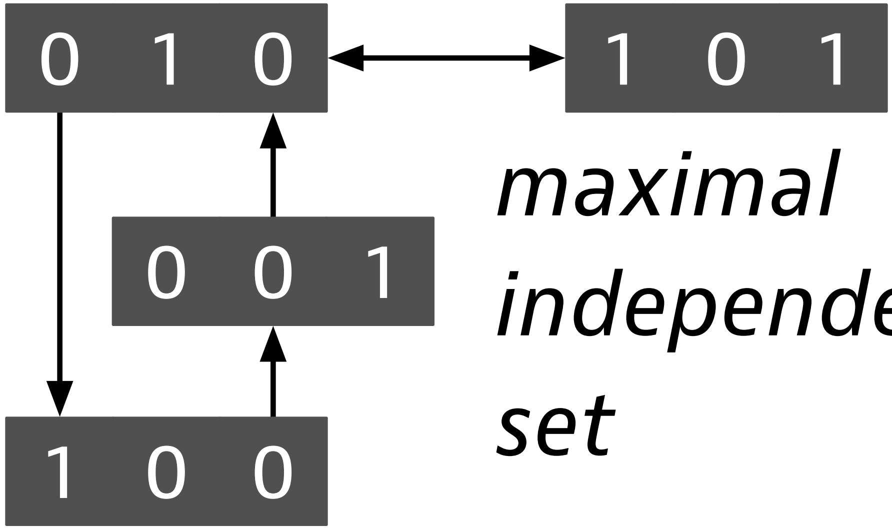
3-coloring



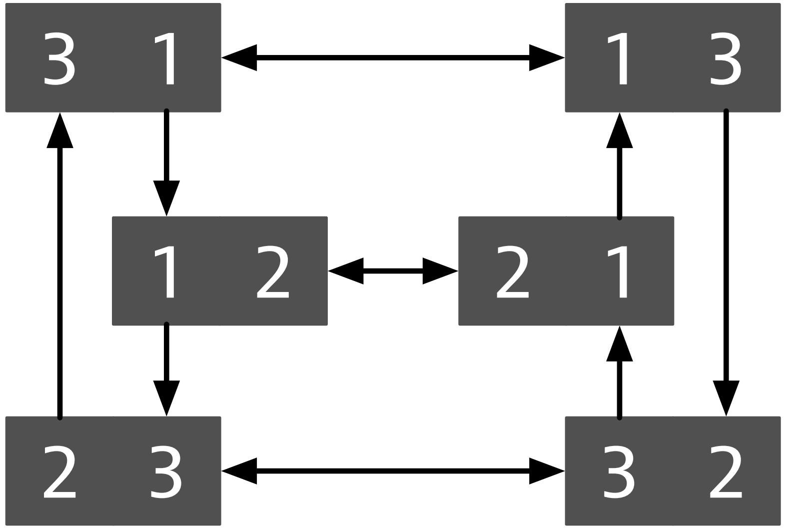
2-coloring



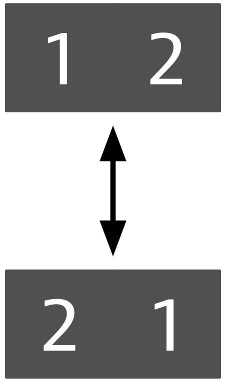
independent set



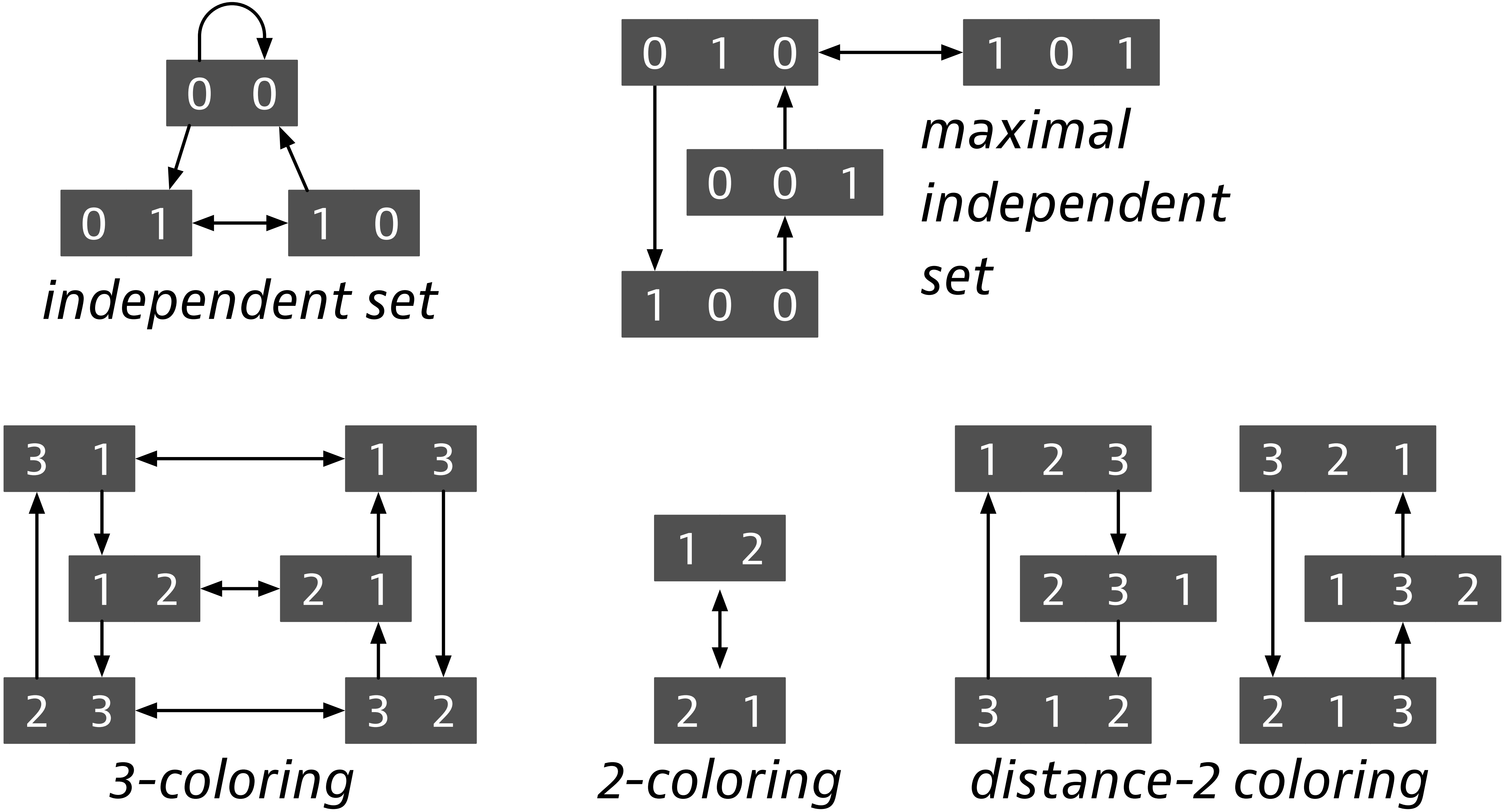
maximal independent set



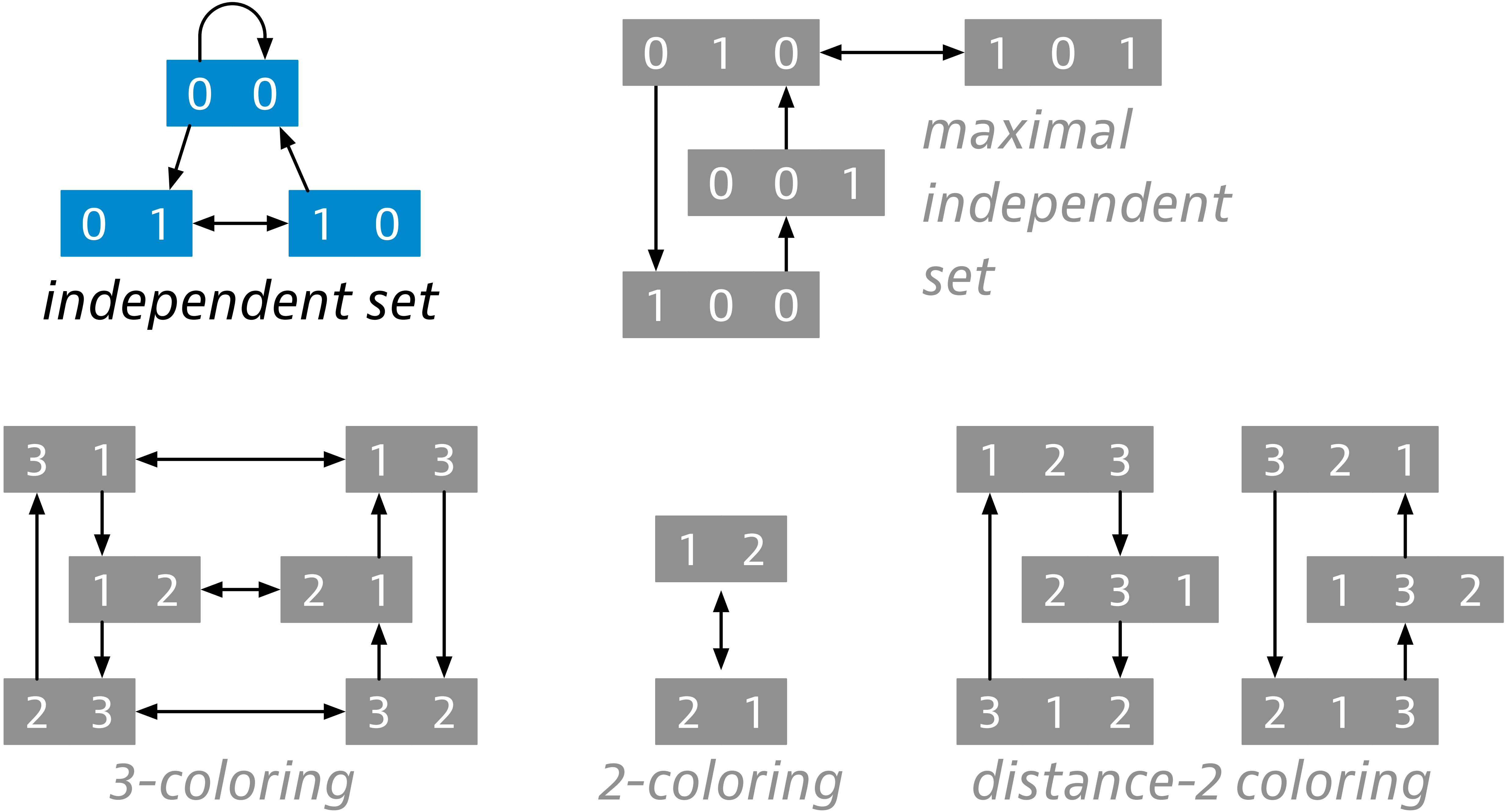
3-coloring



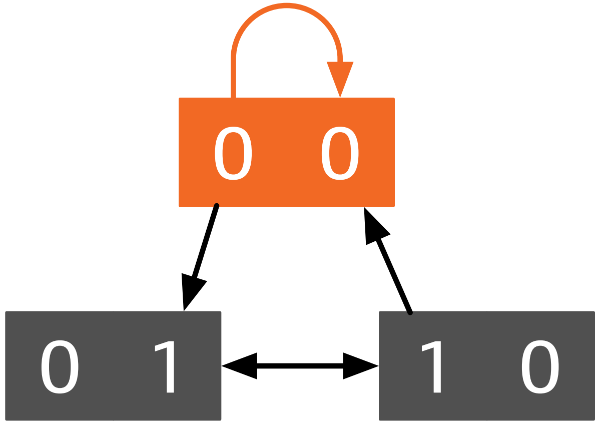
2-coloring



0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring

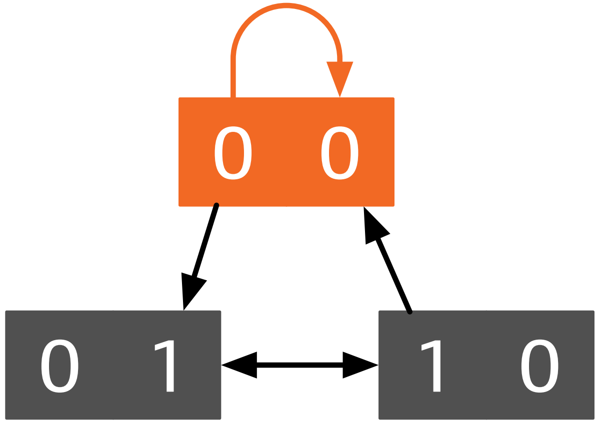


0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring



independent set

self-loop

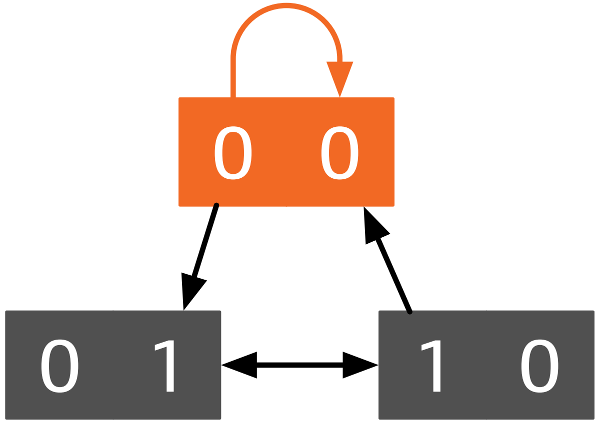


independent set

self-loop

solvable
in $O(1)$ rounds

Algorithm:
Constant output
(e.g. here all-0)



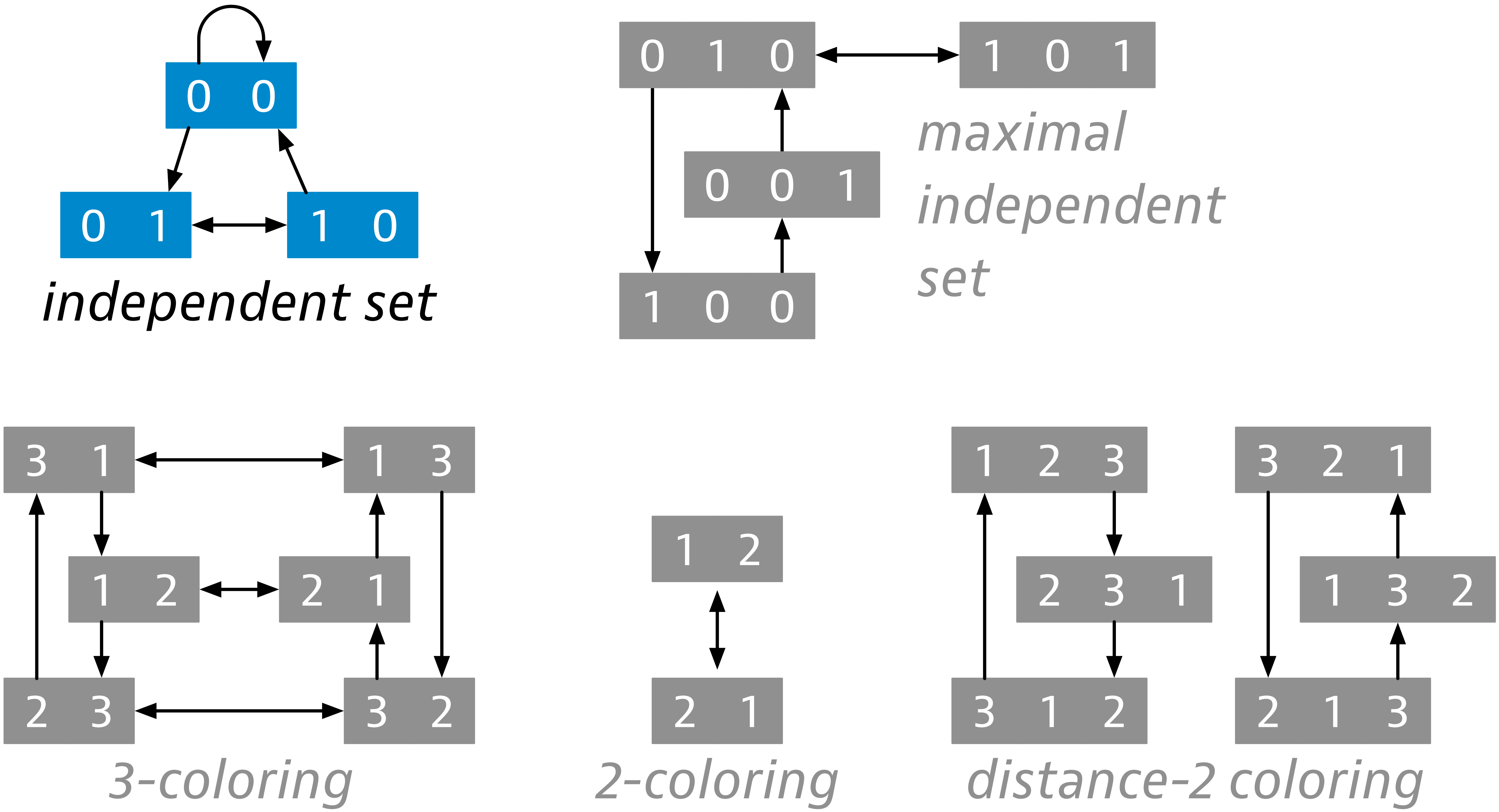
independent set

self-loop

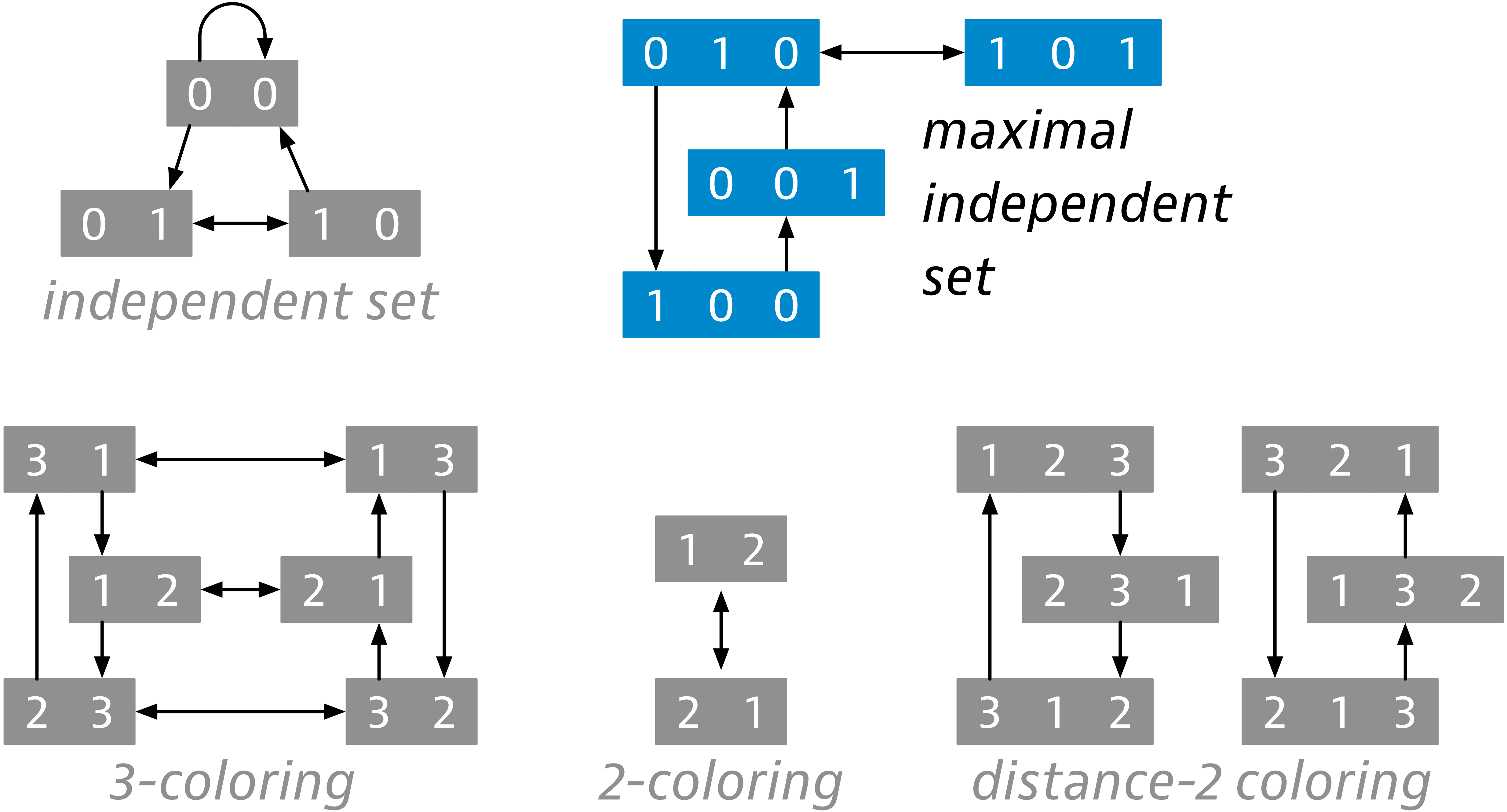
↓ ↑

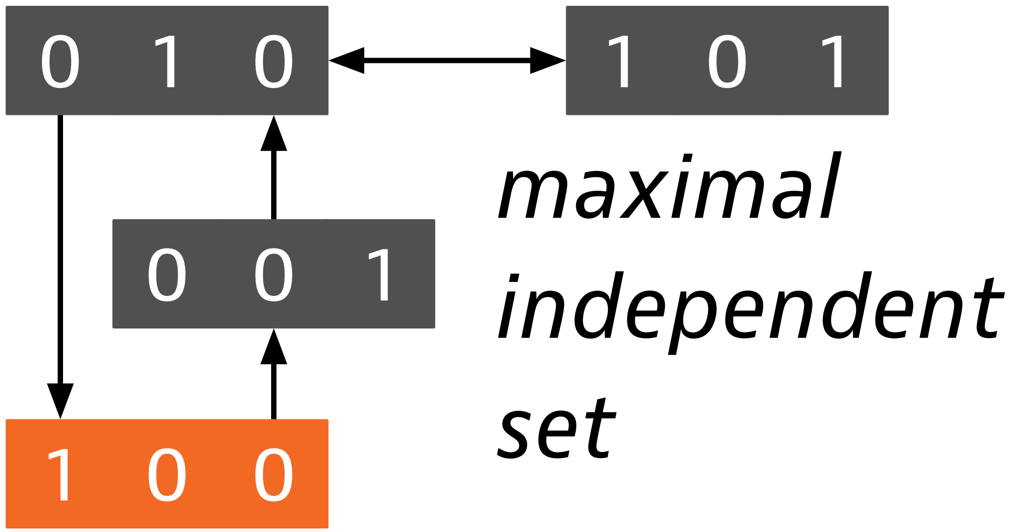
solvable in $O(1)$ rounds

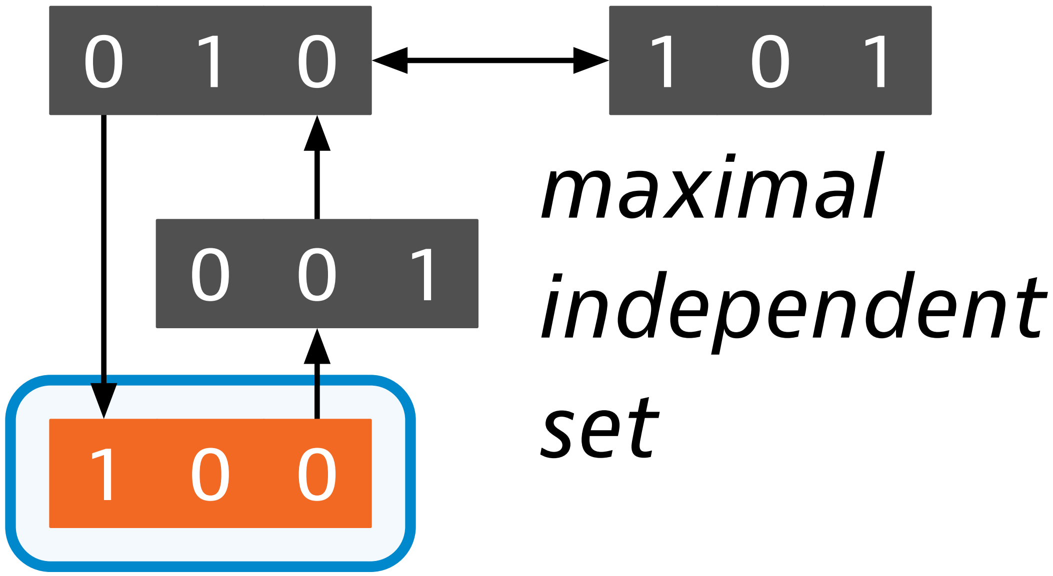
Proof: No self-loop
→ any solution breaks symmetry everywhere
→ can be used to find 3-coloring
→ not possible in $o(\log^* n)$ rounds



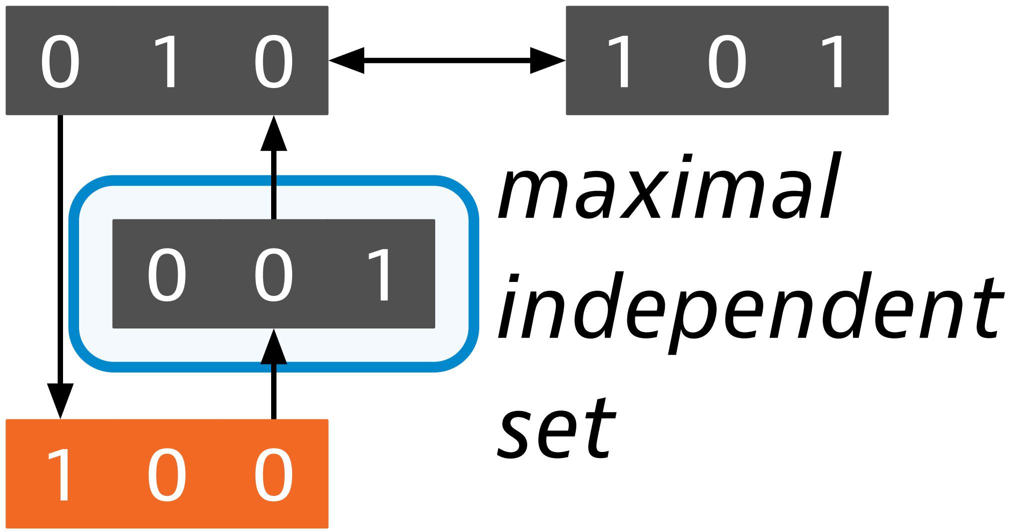
0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring


0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring

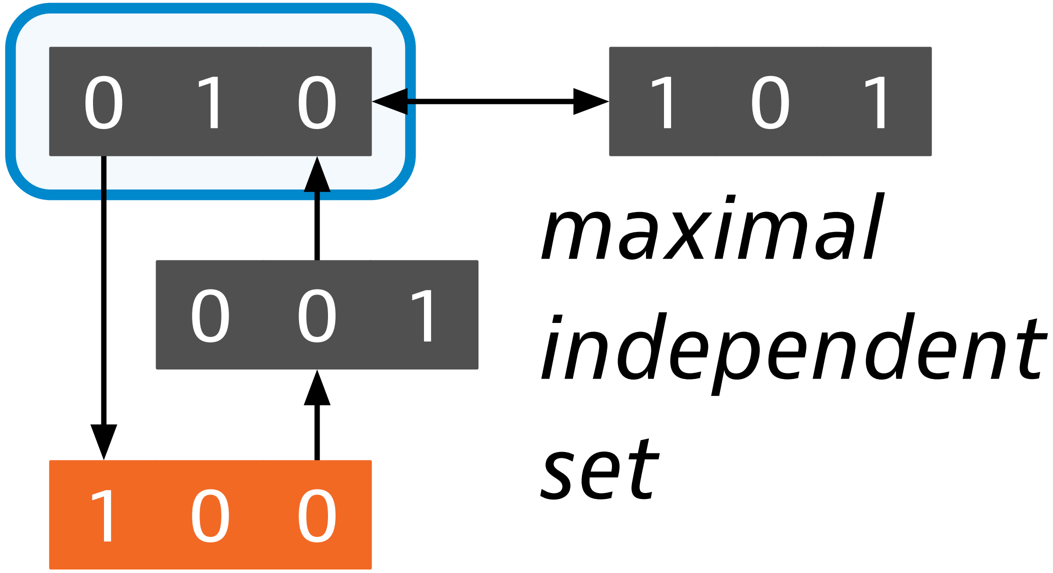

0 1 0
1 0 1
0 0 1
1 0 0
maximal
independent
set



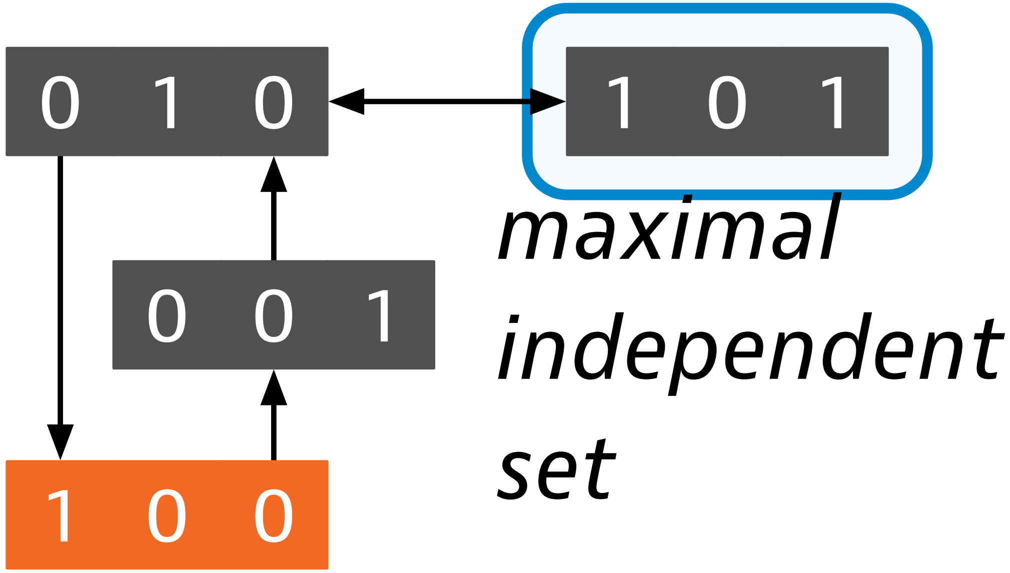
0



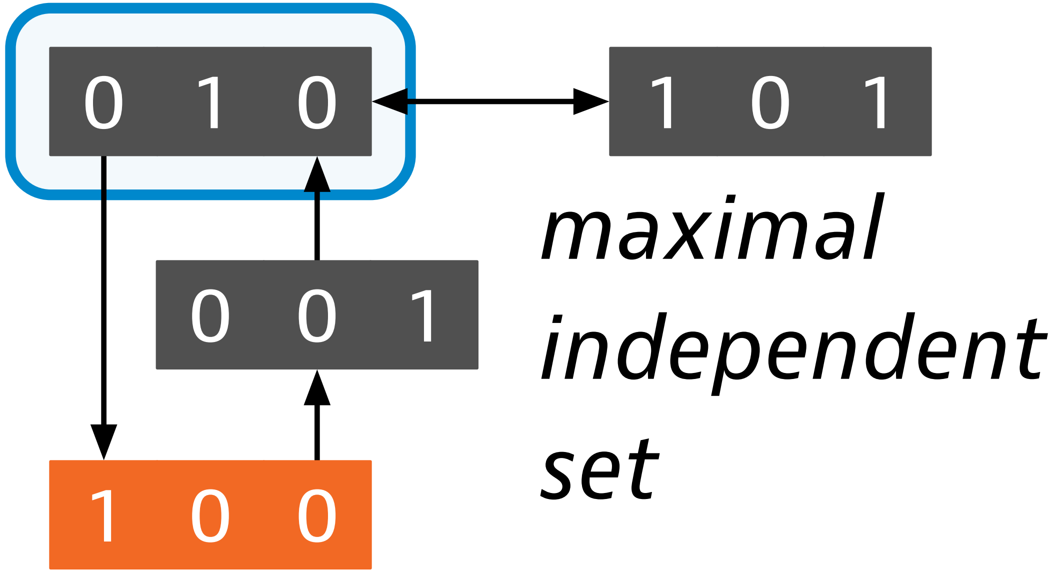
1



2

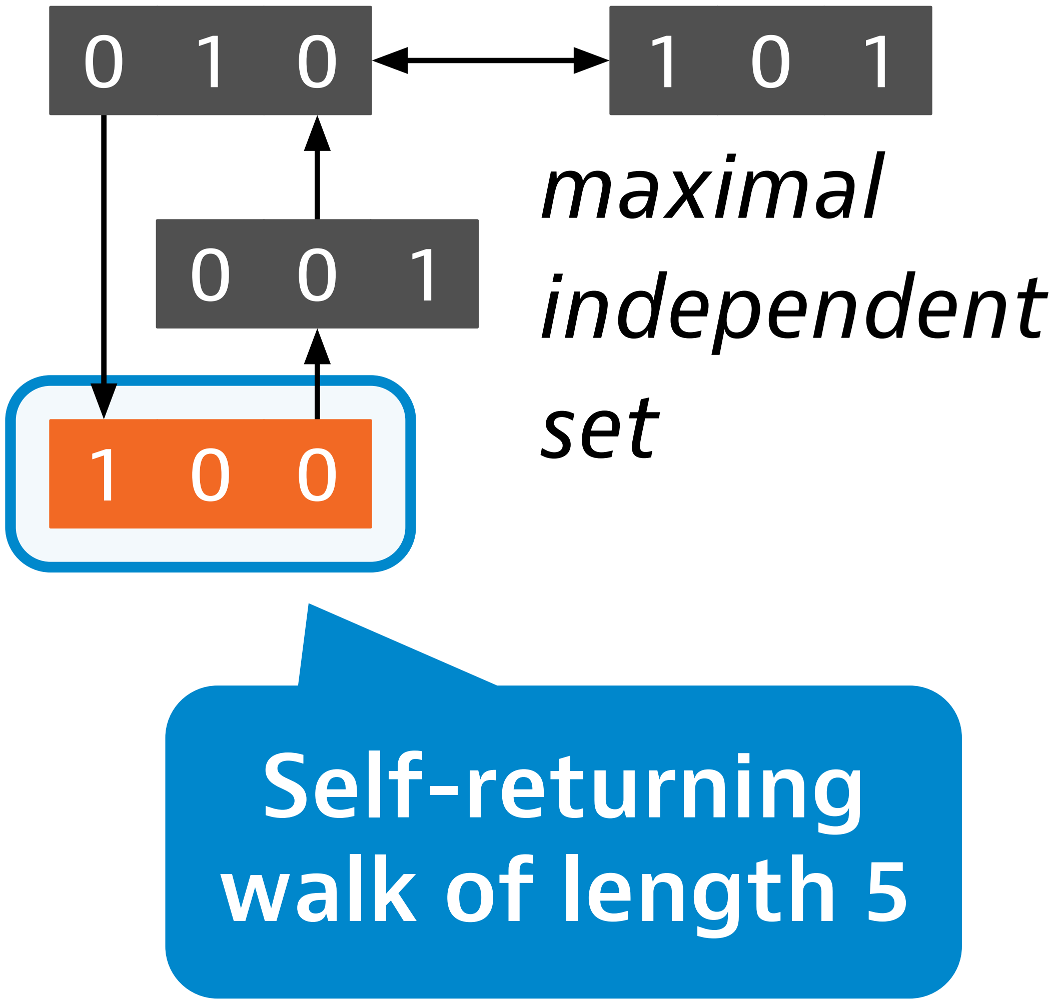


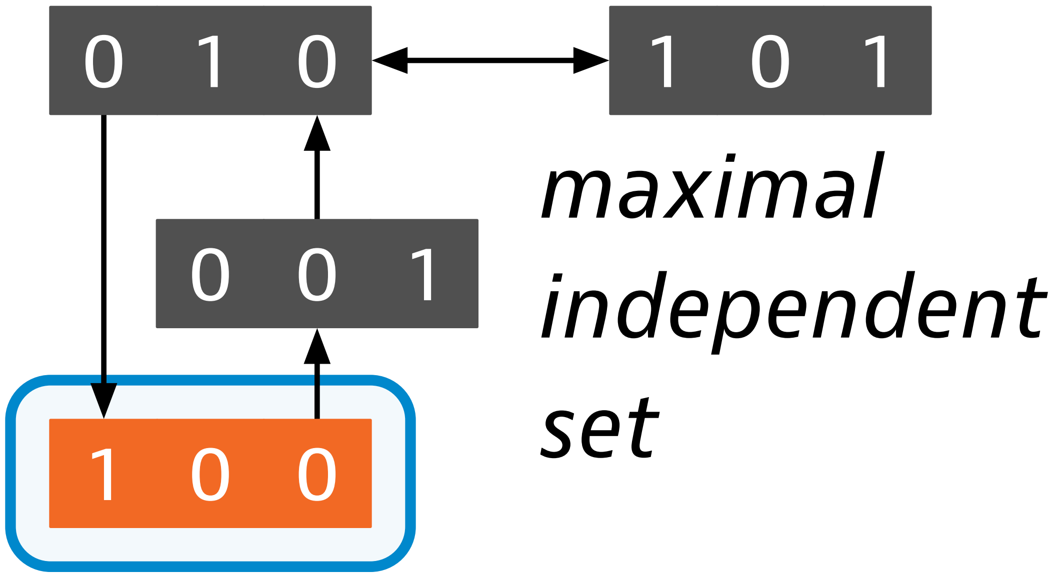
3



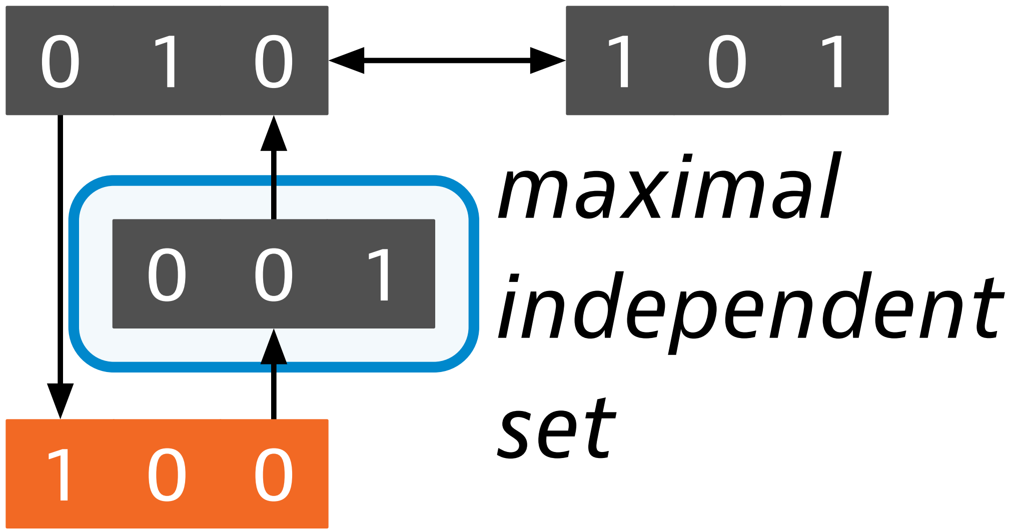
4

5

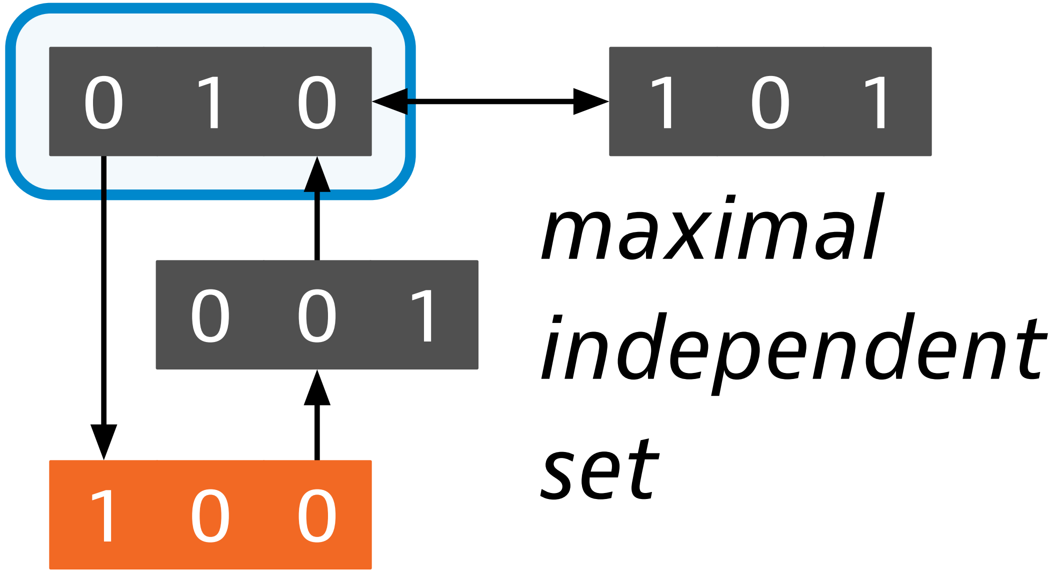

0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
Self-returning walk of length 5



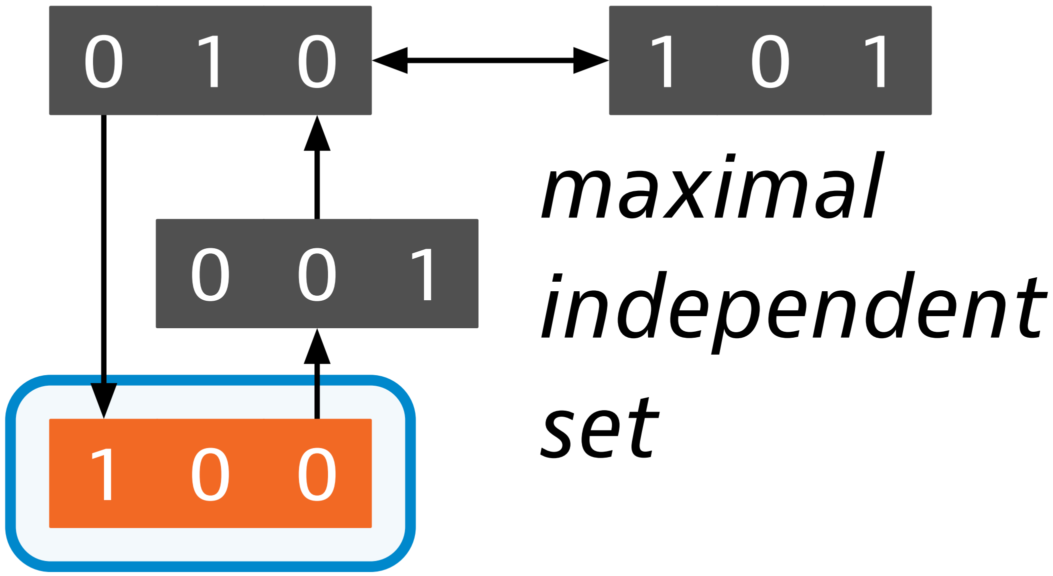
0



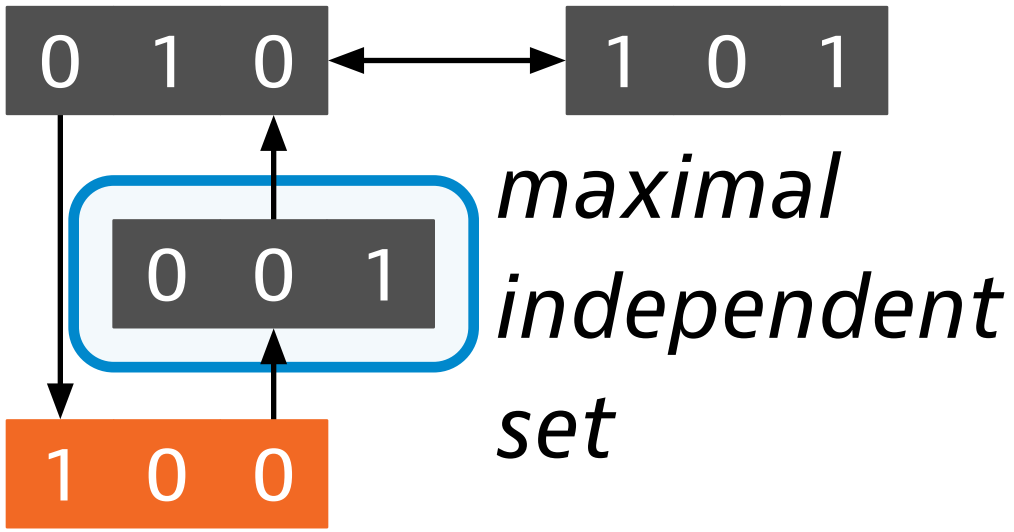
1



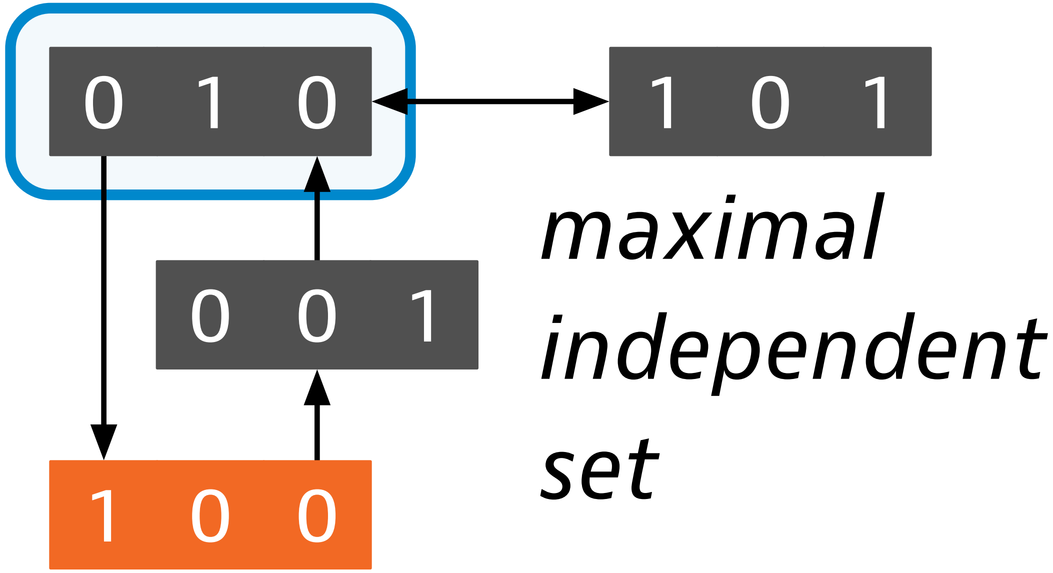
2



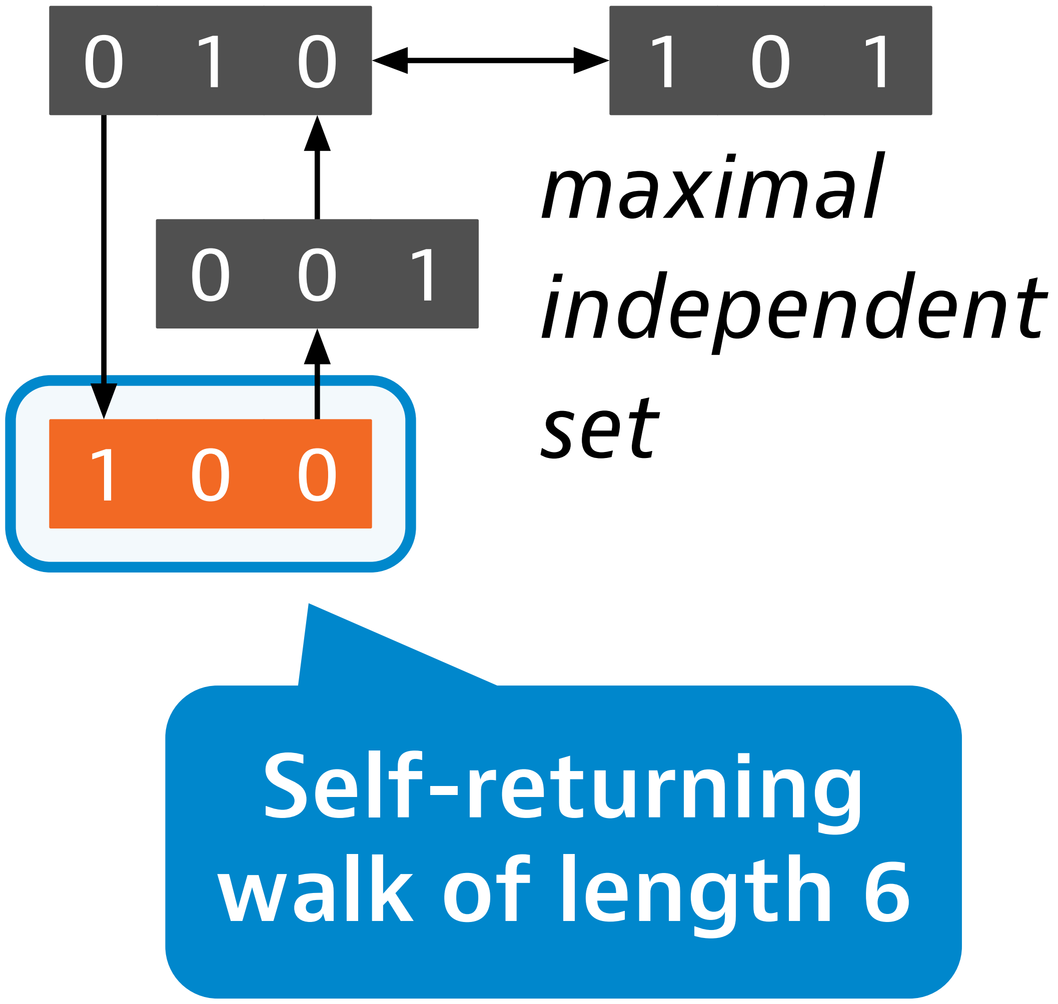
3



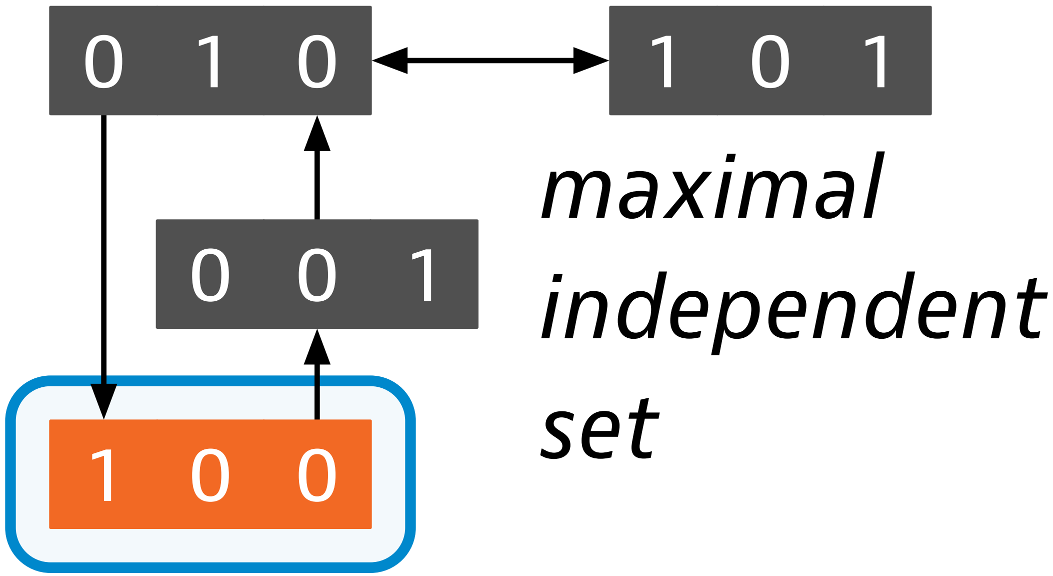
4



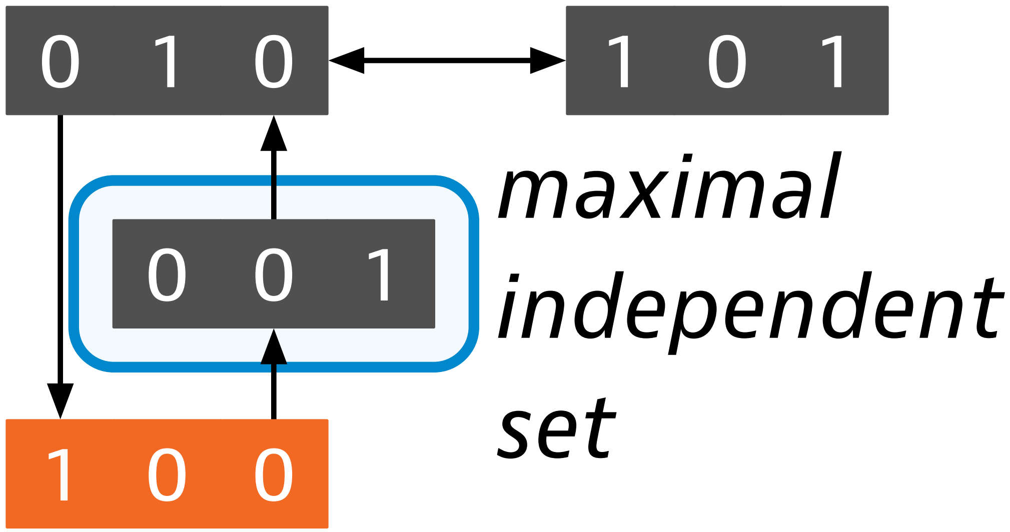
5


0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
Self-returning walk of length 6

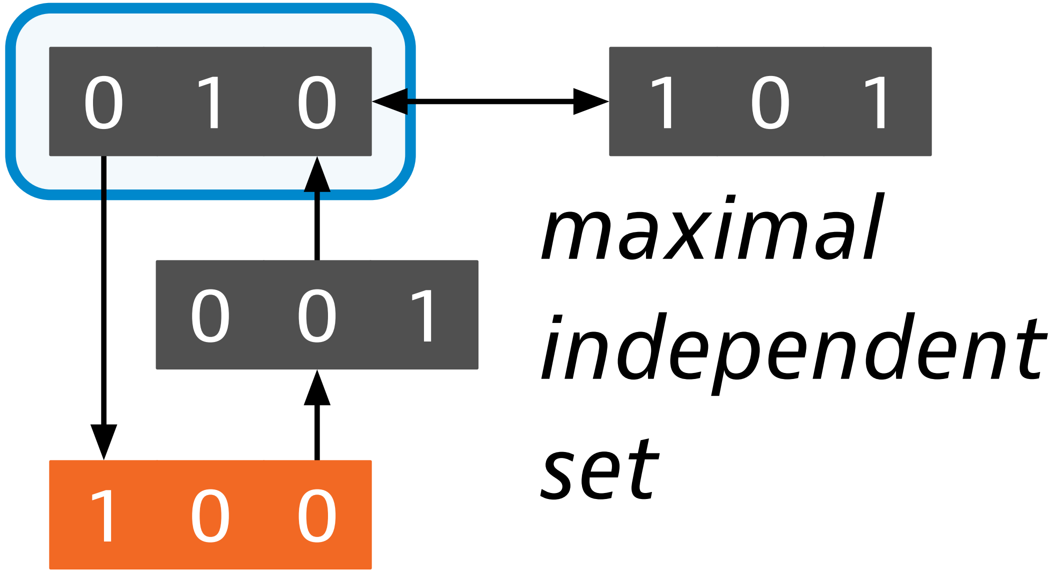
6



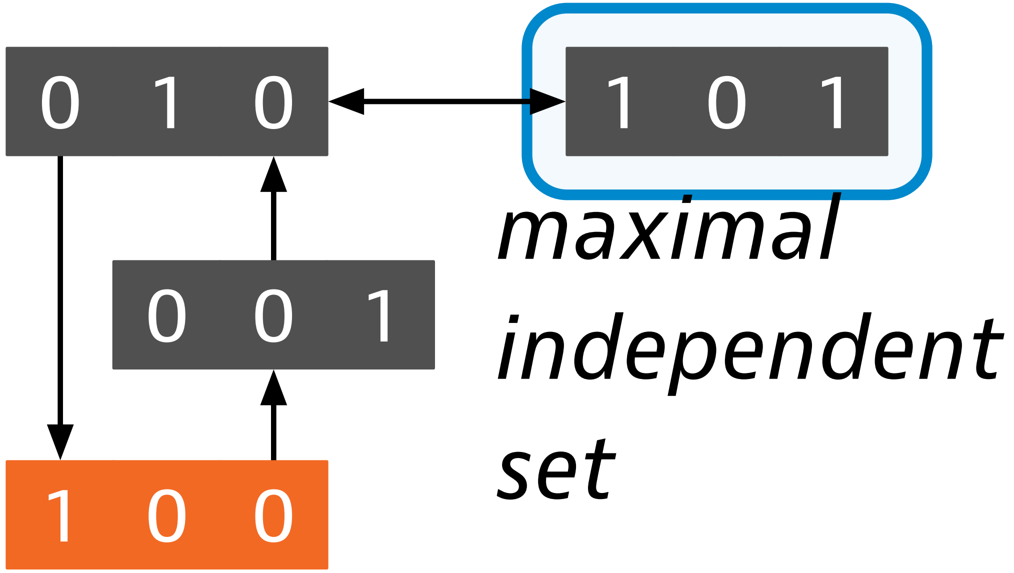
0



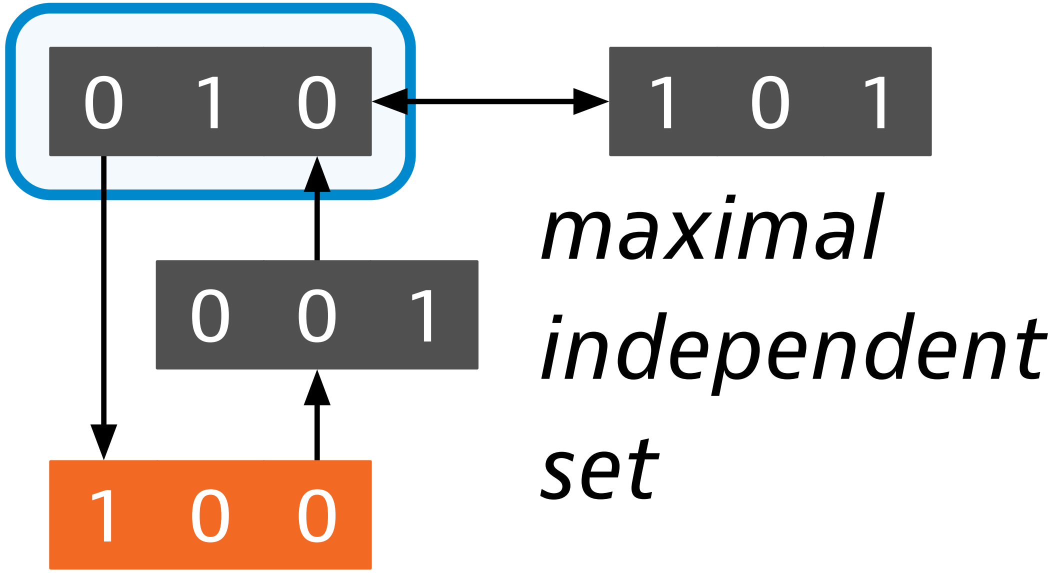
1



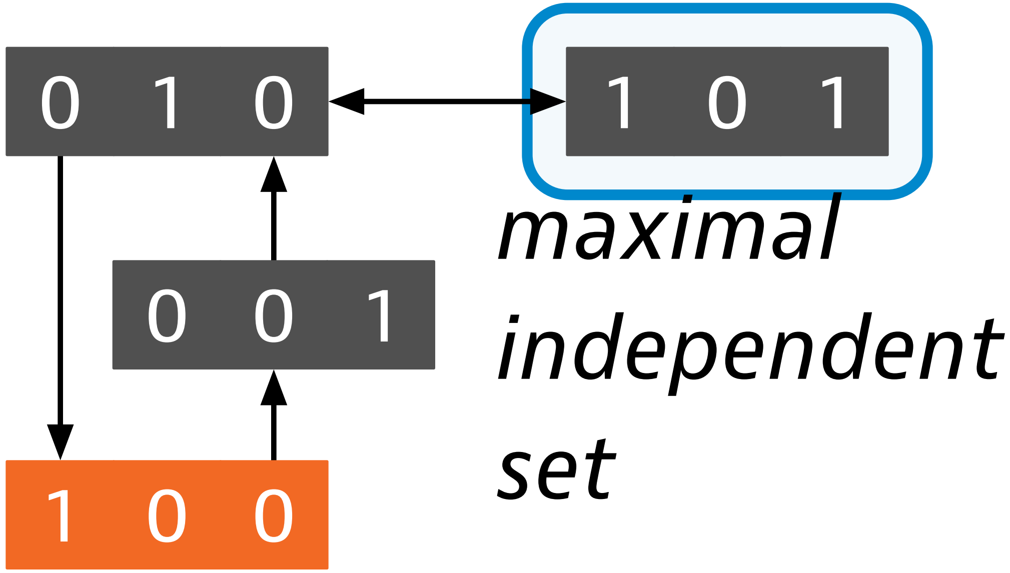
2



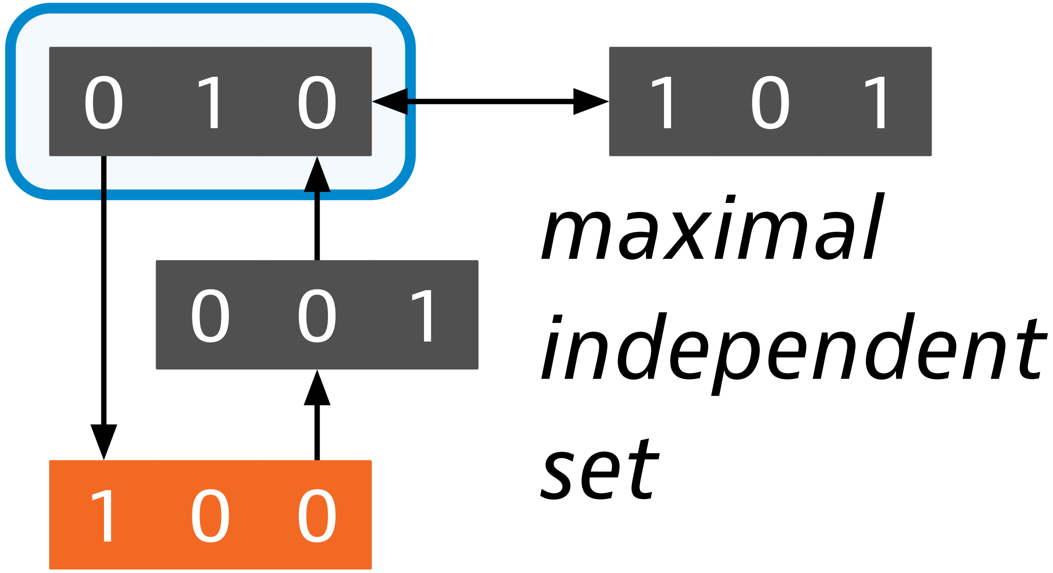
3



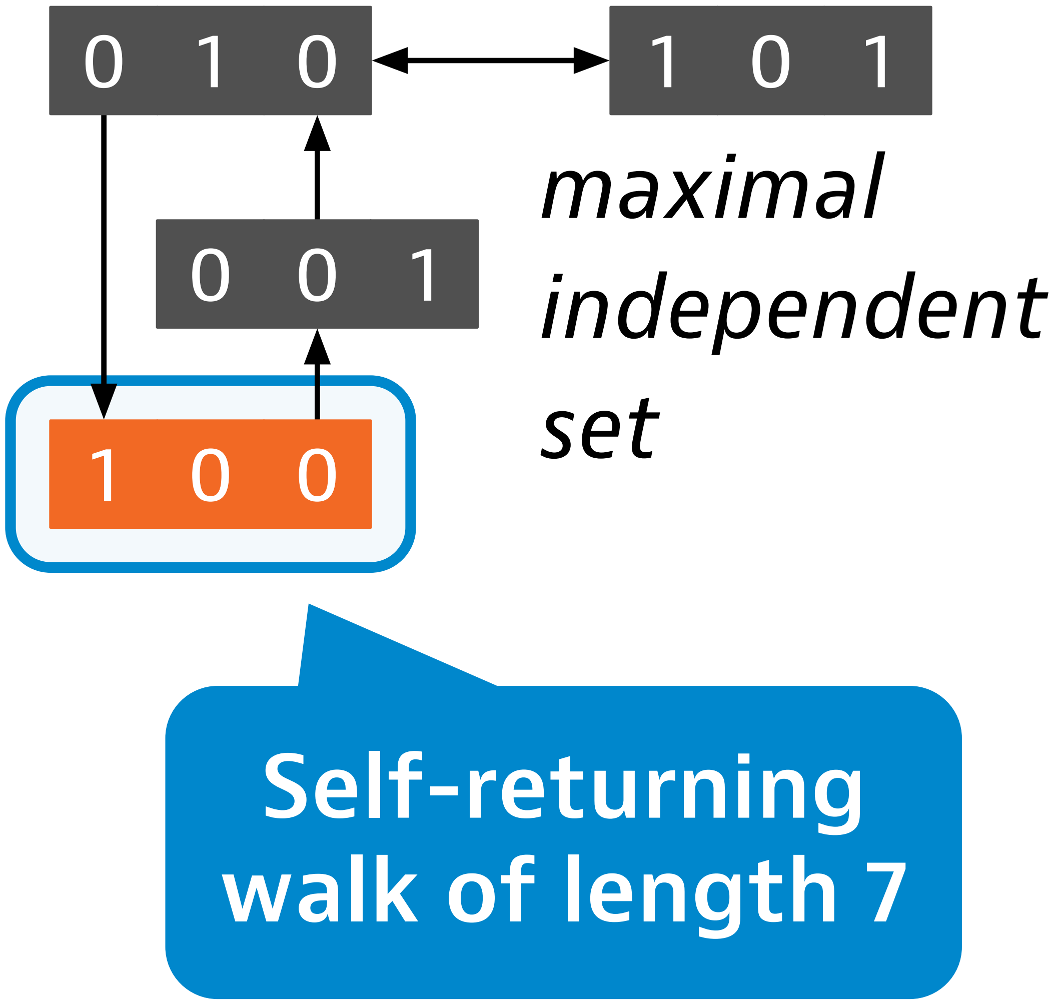
4



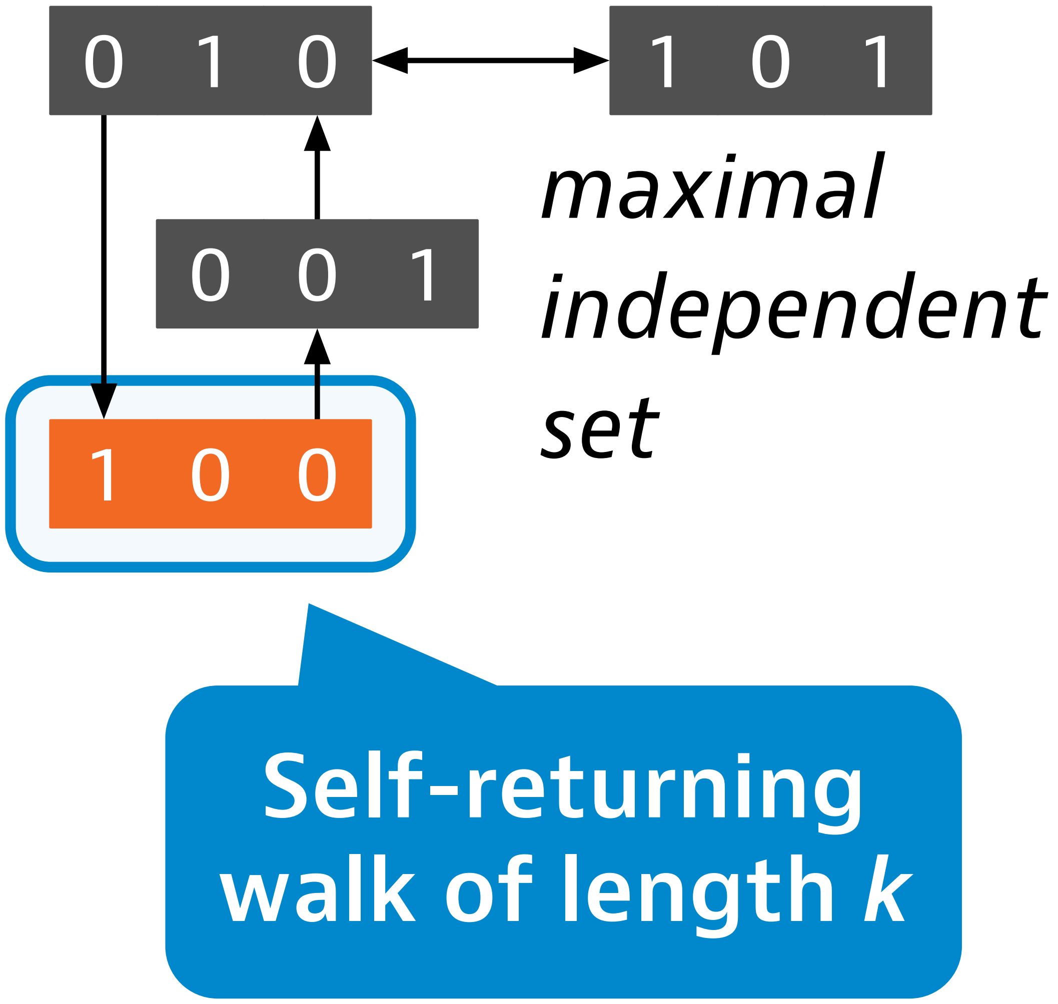
5



6

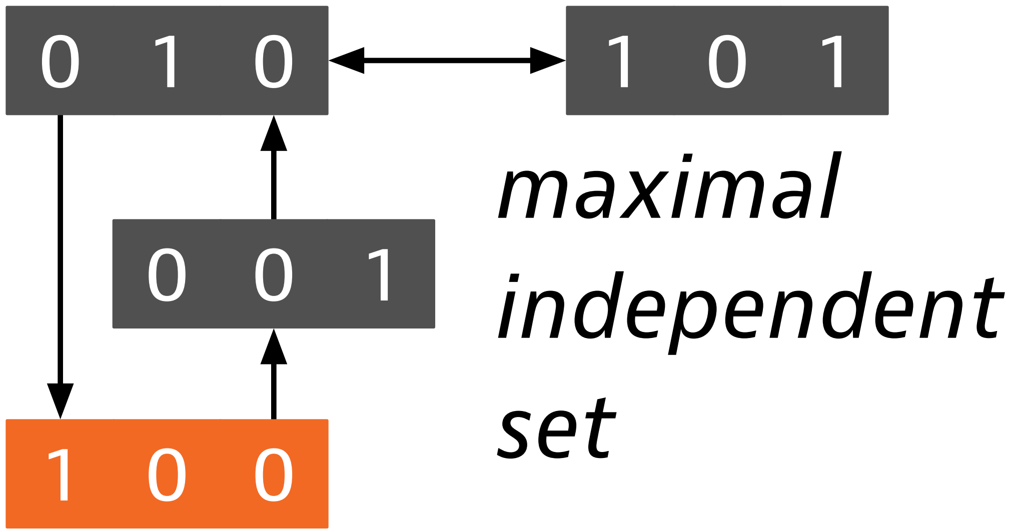

0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
Self-returning walk of length 7

7

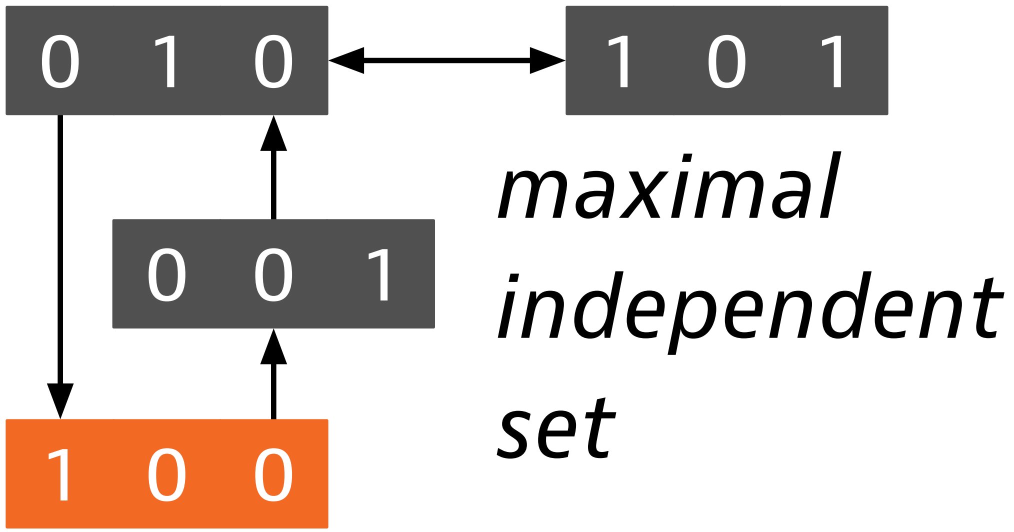

0 1 0
1 0 1
0 0 1
1 0 0
maximal
independent
set
Self-returning
walk of length *k*

$k = 5, 6, 7, 8, \ldots$

“Flexible”:
for all $k \geq k_0$
there is a self-returning walk
of length k



"Flexible": for all $k \geq k_0$ there is a self-returning walk of length k

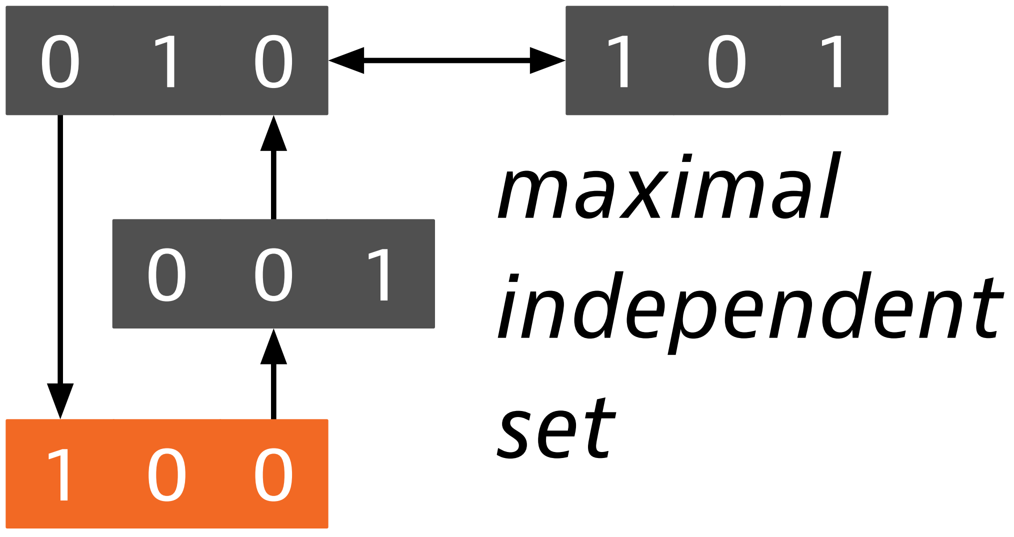


Decidable in polynomial time

"Flexible":
for all $k \geq k_0$
there is a self-returning walk
of length k

↓

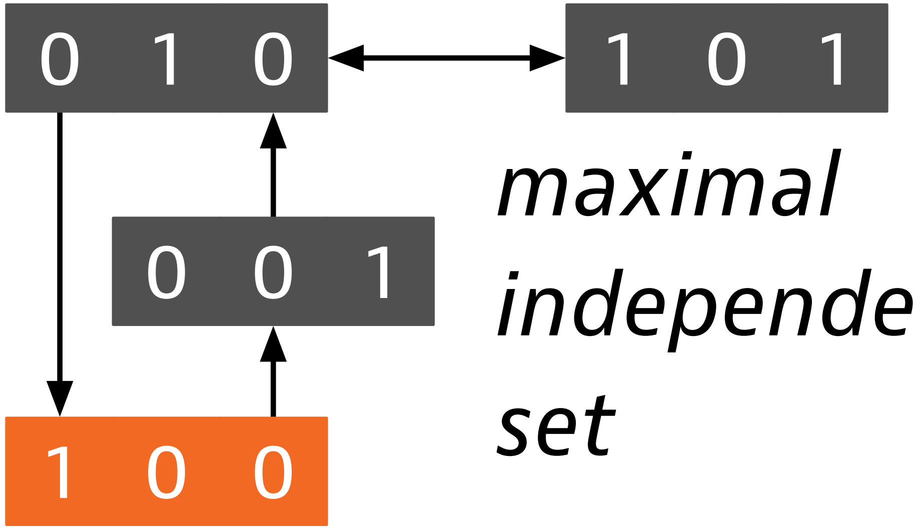
solvable in
$O(\log^* n)$ rounds



Algorithm:

- split in blocks of length $\geq k_0$
- use the flexible configuration at each block boundary
- fill in between boundaries by following a self-returning walk

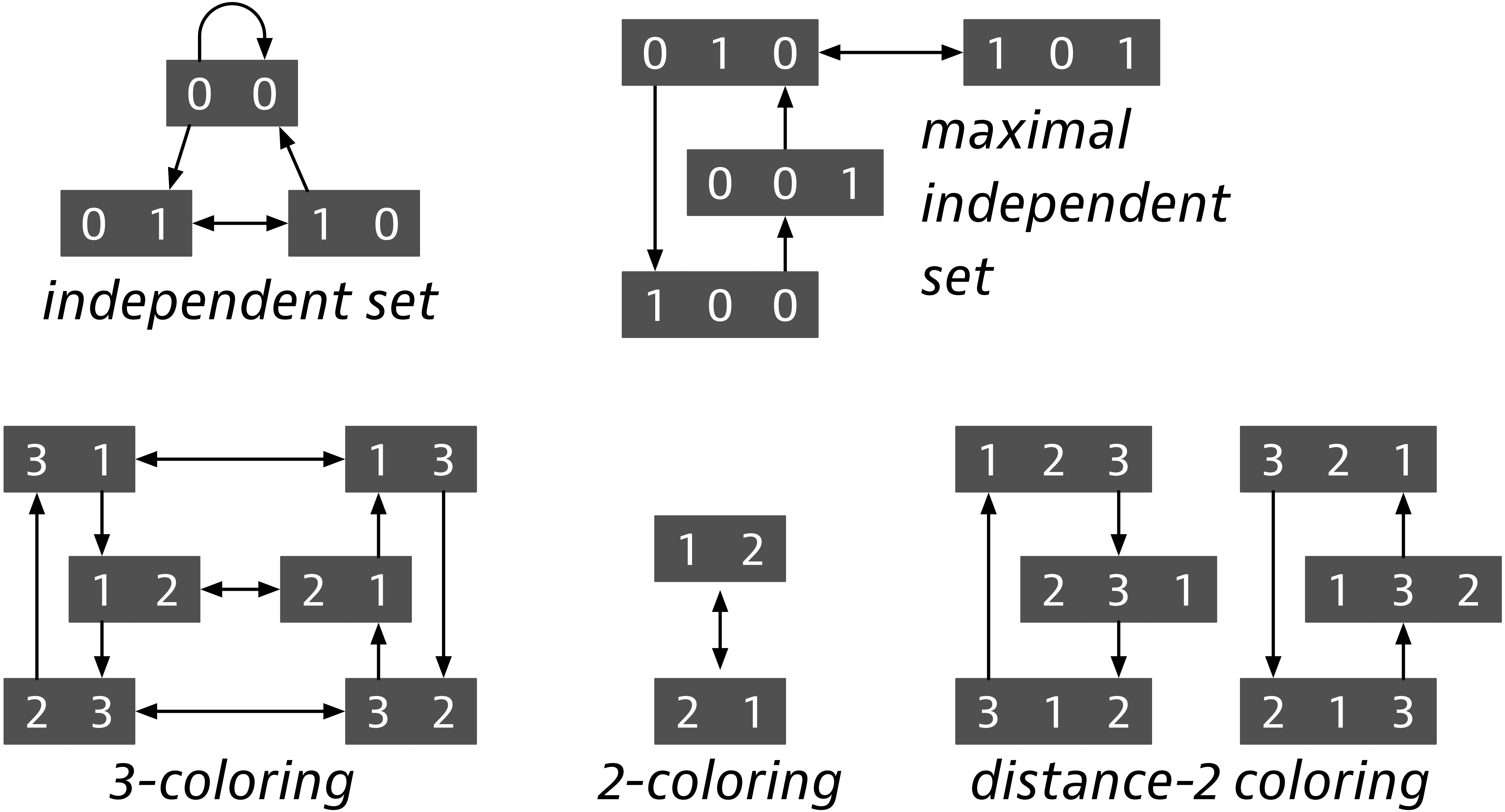
"Flexible":
for all $k \geq k_0$
there is a self-returning walk
of length k



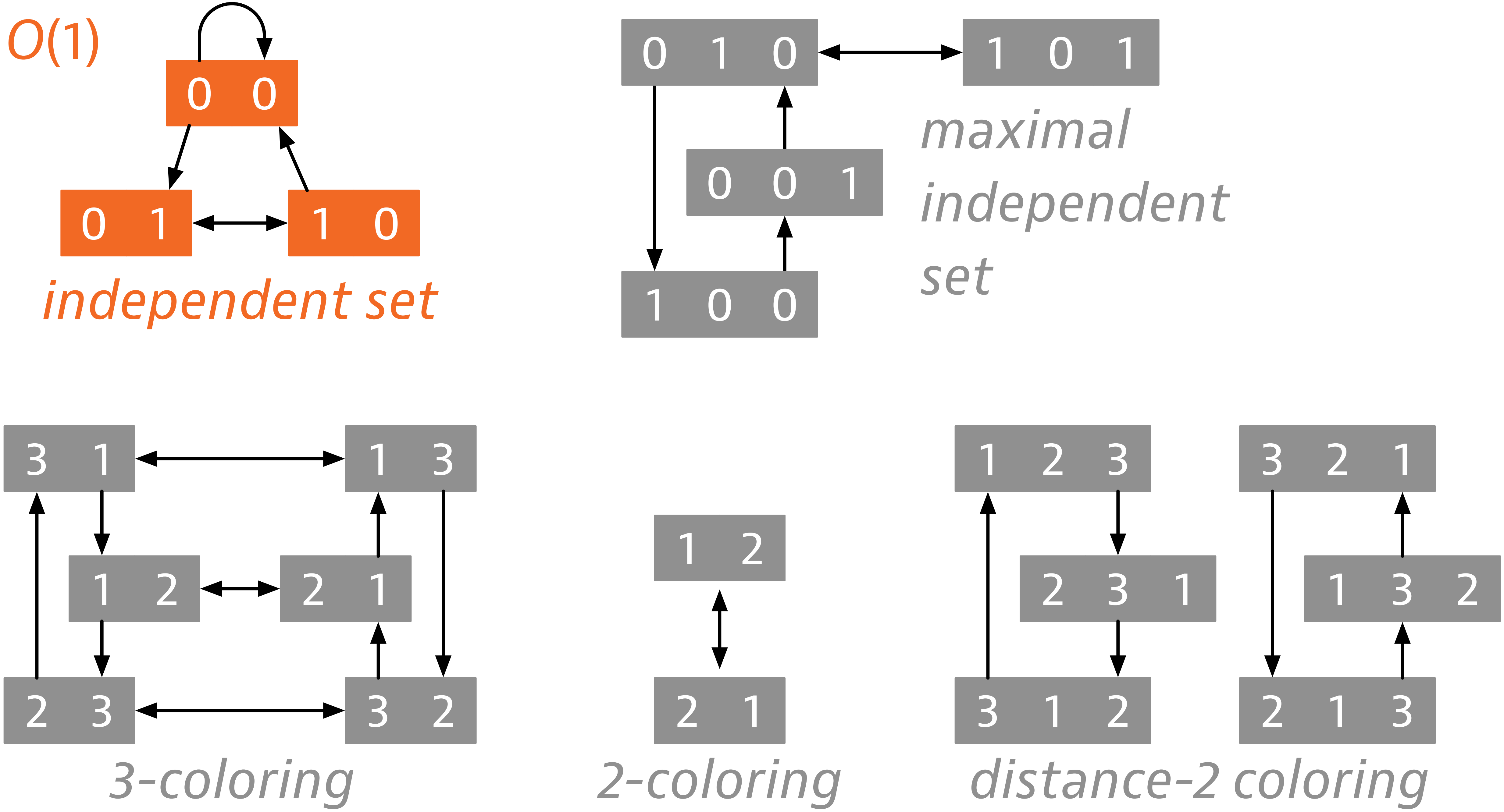
maximal independent set

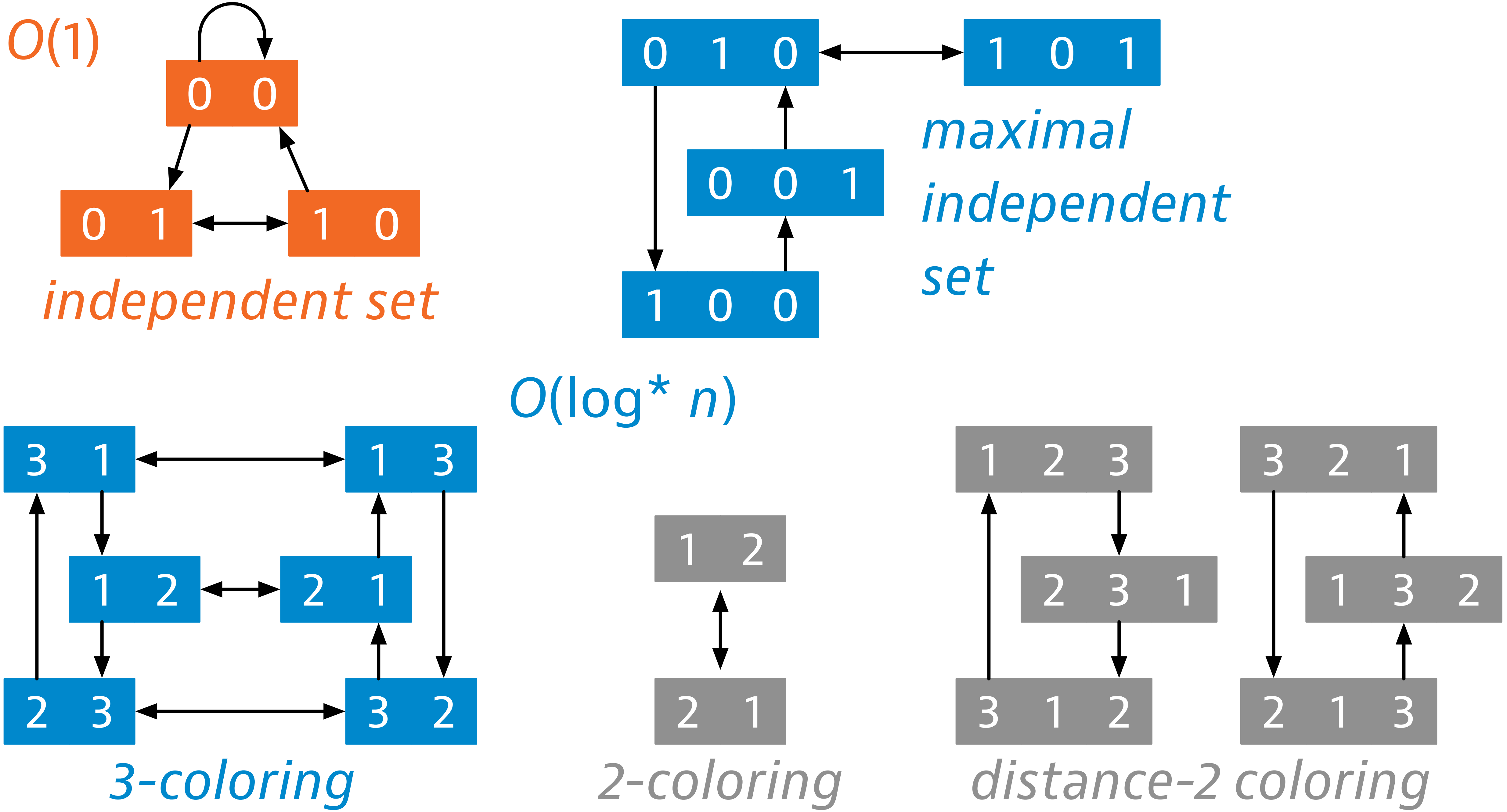
solvable in
$O(\log^* n)$ rounds

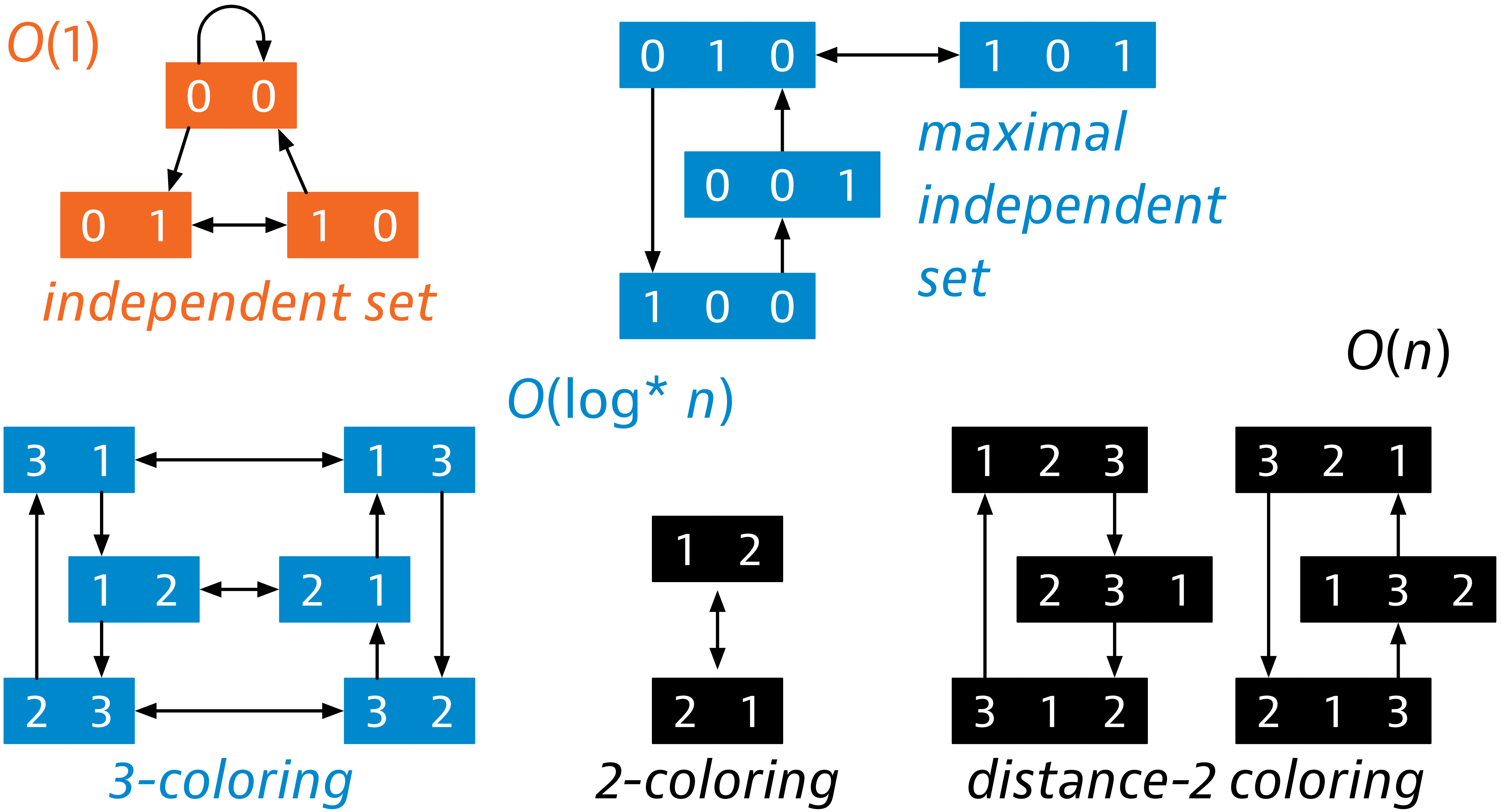
Proof: Not flexible → must use the same non-flexible configuration at least twice far from each other; not compatible for all distances
→ global coordination needed
→ not possible in ***o*(*n*)** rounds



0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring


O(1)
0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring


O(1)
0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
O(log* n)
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring



O(1)
0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
O(n)
O(log* n)
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring

Fully automatic

- Write down the specification of ***any locally checkable problem X***
- Then you can ***find efficiently***
  - distributed round complexity of X
  - asymptotically optimal distributed algorithm for X

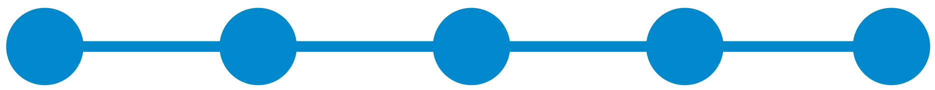
X = { 001, 010, 100, 101 }

This algorithm solves X in time $O(\log^* n)$

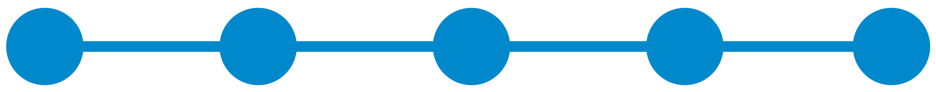
"Oh but doing it for ***this*** *case is of course trivial…"*

But what are **other cases** in which algorithm design & lower-bound proofs can be automated?

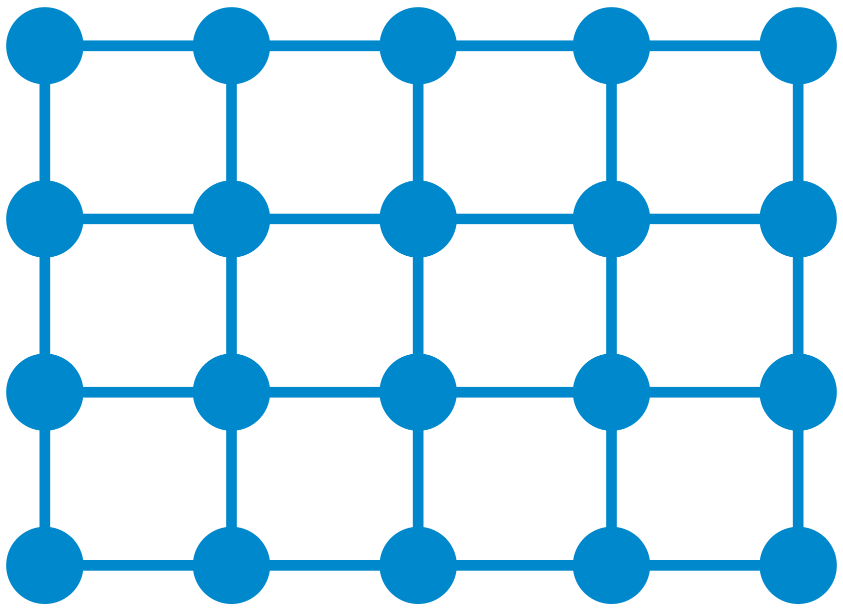
Cycles, paths



Cycles, paths



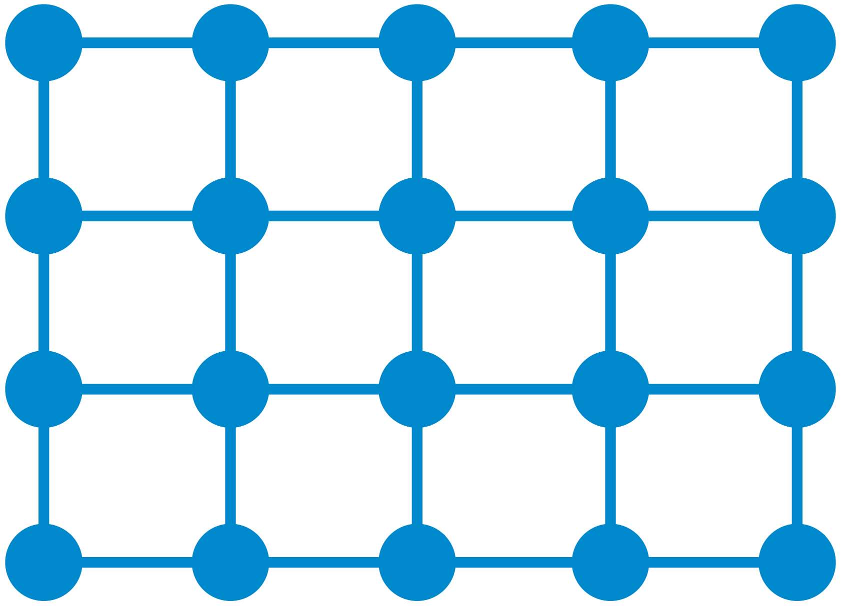
Grids



Cycles, paths

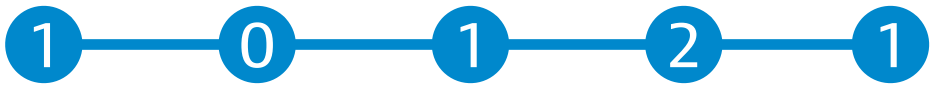
solution ≈
execution history of
a **finite automaton**

Grids



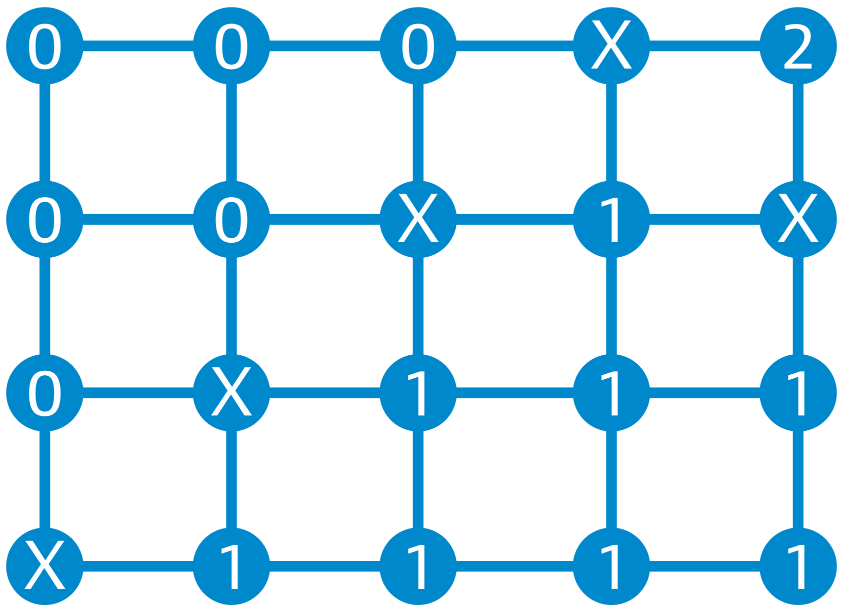
Cycles, paths

solution ≈
execution history of
a **finite automaton**



Grids

solution ≈
execution history of
a **Turing machine**



Cycles, paths

solution ≈
execution history of
a **finite automaton**

Many questions
(efficiently)
decidable

Grids

solution ≈
execution history of
a **Turing machine**

Cycles, paths

solution ≈
execution history of
a **finite automaton**

Many questions (efficiently) decidable

Grids

solution ≈
execution history of
a **Turing machine**

Many questions undecidable

Undecidable

$\neq$

hopeless

Normal forms

Any algorithm $\boldsymbol{A}$ that solves a locally checkable problem X fast can be written as $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C_k}$

- $\boldsymbol{C_k}$ = distance-k coloring
- $\boldsymbol{B}$ = finite function that maps colored neighborhoods to local outputs

Normal forms

Any algorithm $\boldsymbol{A}$ that solves a locally checkable problem X fast can be written as $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C}_k$

- $\boldsymbol{C}_k$ = distance-k coloring
- $\boldsymbol{B}$ = finite function that maps colored neighborhoods to local outputs

"Fast" = e.g. $O(\log^* n)$

Normal forms

Any algorithm $\boldsymbol{A}$ that solves a locally checkable problem X fast can be written as $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C_k}$

- $\boldsymbol{C_k}$ = distance-k coloring
- $\boldsymbol{B}$ = finite function that maps colored neighborhoods to local outputs

Proof idea: Coloring ≈ locally unique identifiers. If A fails with such fake identifiers, it also fails in some small graph with some real identifiers.

Normal forms

Any algorithm $\boldsymbol{A}$ that solves a locally checkable problem X fast can be written as $\boldsymbol{A} = B \circ C_k$

- C_k = distance-k coloring
- B = finite function that maps colored neighborhoods to local outputs

For each k = 1, 2, 3, ...:

- check all possible candidate functions B
- if any of them is good → fast algorithm found!

Normal forms

Any algorithm $\boldsymbol{A}$ that solves a locally checkable problem X fast can be written as $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C_k}$

- $\boldsymbol{C_k}$ = distance-k coloring
- $\boldsymbol{B}$ = finite function that maps colored neighborhoods to local outputs

For each k = 1, 2, 3, ...:

- check all possible candidate functions $\boldsymbol{B}$
- if any of them is good → fast algorithm found!

Normal forms

Any algorithm ***A*** that solves a locally checkable problem X fast can be written as ***A*** = ***B*** ∘ C_k

- C_k =
- ***B*** = ... explored neighbourhoods

For each *k* = 2, 3, ...:

- check all possible candidate functions ***B***
- if any of them is good → fast algorithm found!

Finite computation for a given candidate *B*:
no worries about the halting problem

Normal forms

Undecidability:
don't know when to stop if fast algorithms don't exist

ves a locally checkable
ritten as ***A*** = ***B*** ∘ ***C_k***

ite function that maps colored
neig orhoods to local outputs

For each *k* = 1, 2, 3, ...:

- check all possible candidate functions ***B***
- if any of them is good → fast algorithm found!

Normal forms

Any algorithm $\boldsymbol{A}$ that solves a locally checkable problem X fast can be written as $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C_k}$

- $\boldsymbol{C_k}$ = distance-k
- $\boldsymbol{B}$ = finite func
  neighborhoods

Computational complexity:
typically doubly-exponential in k

For each $\boldsymbol{k}$ = 1, 2, 3, ...

- check all possible candidate functions $\boldsymbol{B}$
- if any of them is good → fast algorithm found!

Sometimes doable!

- Natural problems often solvable with a ***small k***

Sometimes doable!

- Natural problems often solvable with a ***small k***
- We can make it more feasible in practice:
  - more ***“compact” normal forms***,
    e.g. distance-*k* coloring → ruling set

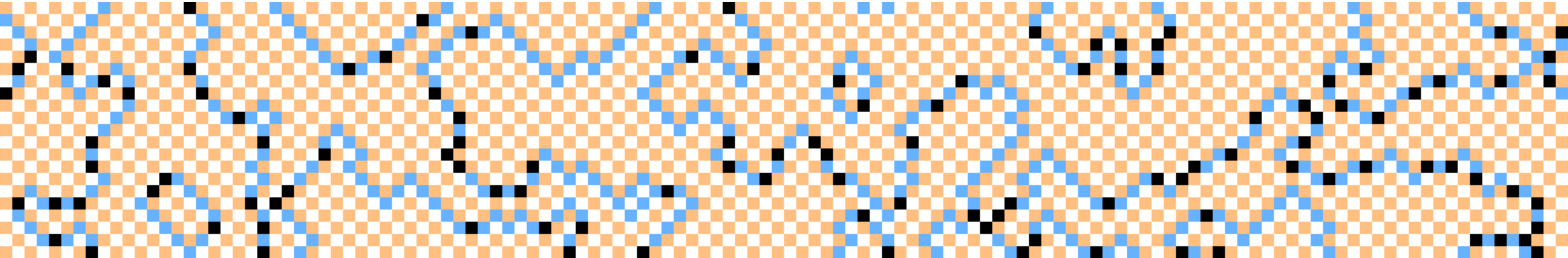
Sometimes doable!

- Natural problems often solvable with a ***small k***
- We can make it more feasible in practice:
    - more ***“compact” normal forms***,
      e.g. distance-*k* coloring → ruling set
    - represent *“candidate B is good for this value of k”*
      as a Boolean formula and use modern ***SAT solvers***
      to find such a *B*

Sometimes doable!

- Example: ***4-coloring in grids***
- Computers were much faster than human beings in figuring out that this is solvable in $O(\log^* n)$ rounds

[Brandt et al., PODC 2017]



Cycles, paths

solution ≈
execution history of
a **finite automaton**

Many questions
(efficiently)
decidable

Grids

solution ≈
execution history of
a **Turing machine**

Many questions
undecidable
(but there is hope!)

Cycles, paths

solution ≈
execution history of
a **finite automaton**

Grids + beyond

solution ≈
execution history of
a **Turing machine**

Bad news apply to
any graph family that
contains large grids

Cycles, paths

solution ≈
execution history of
a **finite automaton**

Grids + beyond

solution ≈
execution history of
a **Turing machine**

What is here between paths and grids?

Cycles, paths

solution ≈
execution history of
a **finite automaton**

Grids + beyond

solution ≈
execution history of
a **Turing machine**

Trees
Bounded treewidth
High girth

Cycles, paths

solution ≈
execution history of
a **finite automaton**

Grids + beyond

solution ≈
execution history of
a **Turing machine**

lots of open questions,
no known obstacles!

- **Trees**
- **Bounded treewidth**
- **High girth**

Cycles, paths

solution ≈
execution history of
a **finite automaton**

Grids + beyond

solution ≈
execution history of
a **Turing machine**

some positive results
already known
(ask me, happy to tell more!)

{ **Trees**
Bounded treewidth
High girth

Big picture: **towards meta-computational research questions**

Meta questions

- **Traditional questions:** what is the best distributed algorithm for solving problem X ?
- **Meta-computational questions:** can we design an (efficient) ***meta-algorithm*** that finds the best distributed algorithm for ***any problem X*** in some problem family F ?

Classification

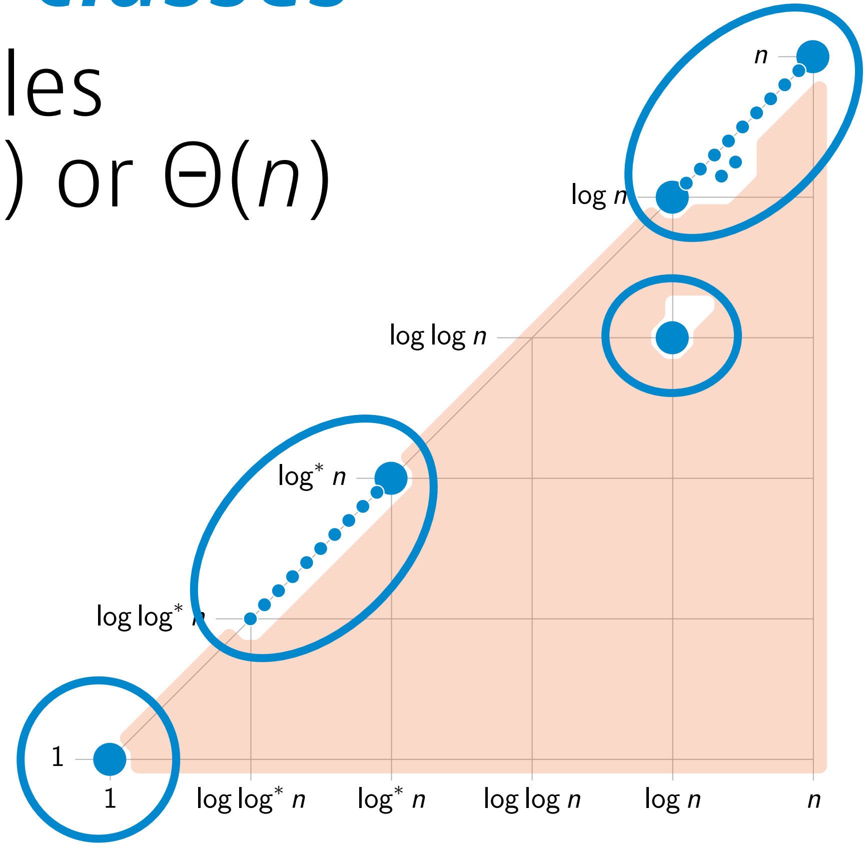
- **Classification:** *"Any problem in this family belongs to one of these classes"*

Classification

- **Classification:** *"Any problem in this family belongs to one of these classes"*
  - locally checkable problems in cycles have complexity $O(1)$ or $\Theta(\log^* n)$ or $\Theta(n)$

Classification

- **Classification:** *"Any problem in this family belongs to one of these classes"*
  - locally checkable problems in cycles have complexity $O(1)$ or $\Theta(\log^* n)$ or $\Theta(n)$
  - locally checkable problems in general graphs belong to one of four broad classes



Classification

- **Classification:** *"Any problem in this family belongs to one of these classes"*
  - locally checkable problems in cycles have complexity $O(1)$ or $\Theta(\log^* n)$ or $\Theta(n)$
- **Meta-algorithms:** *"Here is an efficient algorithm for determining the class of any given problem"*

Classification

- **Classification:** *"Any problem in this family belongs to one of these classes"*
  - locally checkable problems in cycles have complexity $O(1)$ or $\Theta(\log^* n)$ or $\Theta(n)$
- **Meta-algorithms:** *"Here is an efficient algorithm for determining the class of any given problem"*

Classification

- Computers can help with classification, too!

Classification

- Computers can help with classification, too!
- Classify a ***finite sub-family of problems***, automate as much work as possible
  - e.g. bounded alphabet size, bounded degree

Classification

- Computers can help with classification, too!
- Classify a ***finite sub-family of problems***, automate as much work as possible
  - e.g. bounded alphabet size, bounded degree
- Identify interesting nontrivial problems
  - e.g. where computers fail

Classification

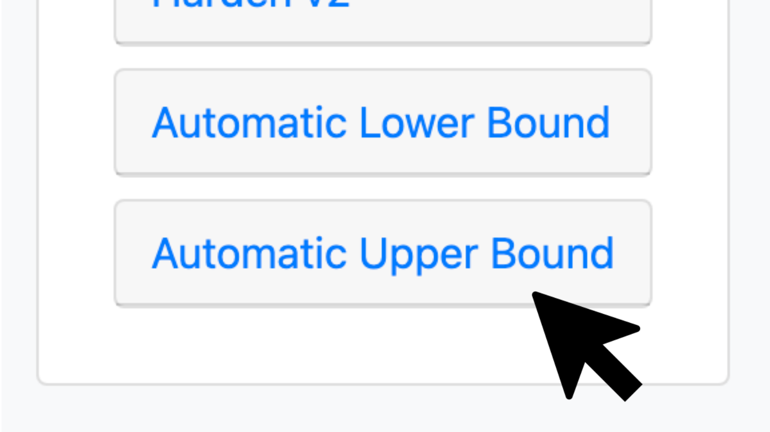
- Computers can help with classification, too!
- Classify a ***finite sub-family of problems***, automate as much work as possible
  - e.g. bounded alphabet size, bounded degree
- Identify interesting nontrivial problems
  - e.g. where computers fail
- Detect patterns, generalize

Classification

- Computers can help with classification, too!
- Classify a ***finite sub-family of problems***, automate as much work as possible
  - e.g. bounded alphabet size, bounded degree
- Identify inter… nontrivial problems
  - e.g. where cor… fail
- Detect p…

How? *Are there general techniques we can apply without much thinking?*

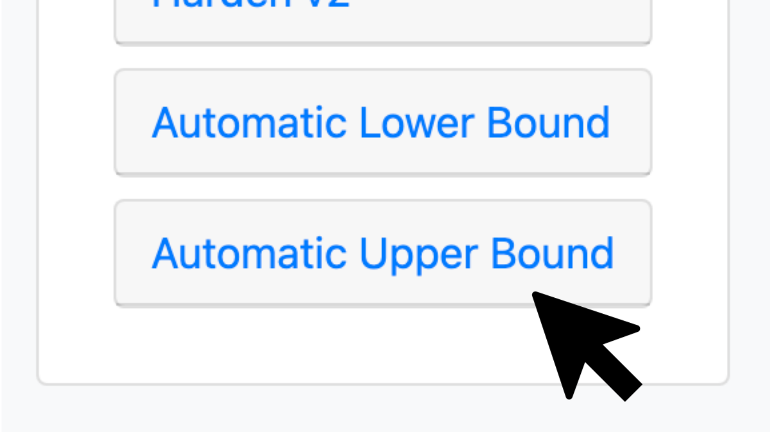
General techniques



- Example: ***round elimination*** technique
  - github.com/olidennis/round-eliminator
  - applicable to any locally checkable problem

[Brandt, PODC 2019]
[Olivetti, PODC 2020]

General techniques



- Example: ***round elimination*** technique
  - github.com/olidennis/round-eliminator
  - applicable to any locally checkable problem
- Does not always work — but when it works, you get algorithms and/or lower bound proofs for free!

[Brandt, PODC 2019]
[Olivetti, PODC 2020]

Success stories

- Lower bound for maximal matching and maximal independent set
- ***Six people and one computer program***
  - enabled rapid hypothesis testing and exploration of possible proof strategies

[Brandt et al., FOCS 2019]

Conclusions

Take-home messages

- *Can we automate our own work?*

Take-home messages

- ***Can we automate our own work?***
  - ***yes*** — some questions on theory of distributed computing can be solved automatically!

Take-home messages

- ***Can we automate our own work?***
  - ***yes*** — some questions on theory of distributed computing can be solved automatically!
  - known obstacles, tons of open questions

Take-home messages

- ***Can we automate our own work?***
  - ***yes*** — some questions on theory of distributed computing can be solved automatically!
  - known obstacles, tons of open questions
- ***Opportunities for human–computer collaboration!***

Take-home messages

- ***Can we automate our own work?***
  - ***yes*** — some questions on theory of distributed computing can be solved automatically!
  - known obstacles, tons of open questions
- ***Opportunities for human–computer collaboration!***
  - theory researchers who can write programs are going to have a competitive edge!