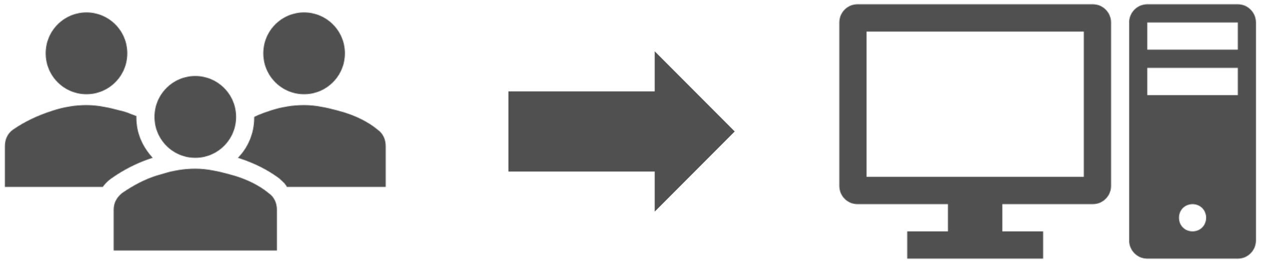
Jukka Suomela
Aalto University · Helsinki · Finland
jukkasuomela.fi

Algorithms that design algorithms?

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



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

Today: ***can we automate the study of distributed computing?***

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

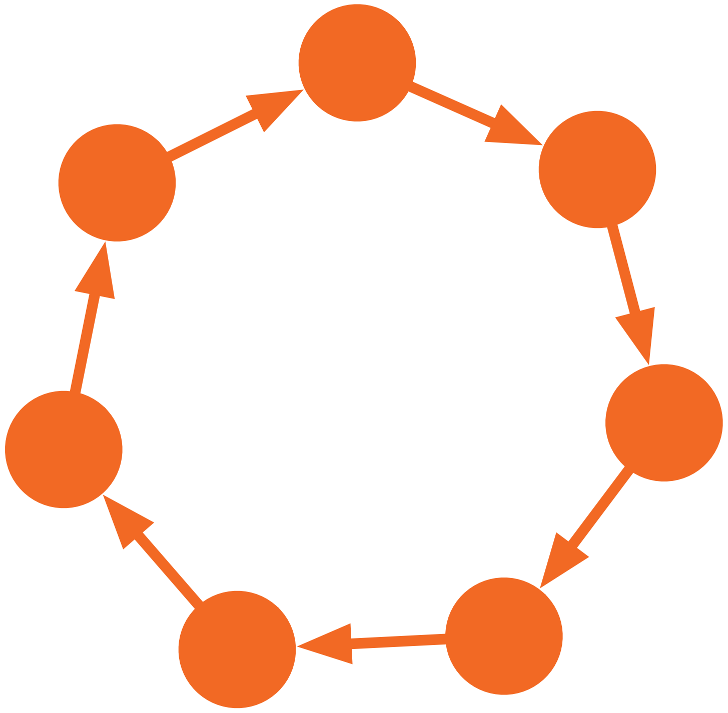
- **Question:** is there an efficient distributed algorithm for solving task *X* in model *M*?
- **Approach:** ~~find smart people, spend lots of time in front of a whiteboard …~~

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

Toy example:
Locally checkable problems in cycles

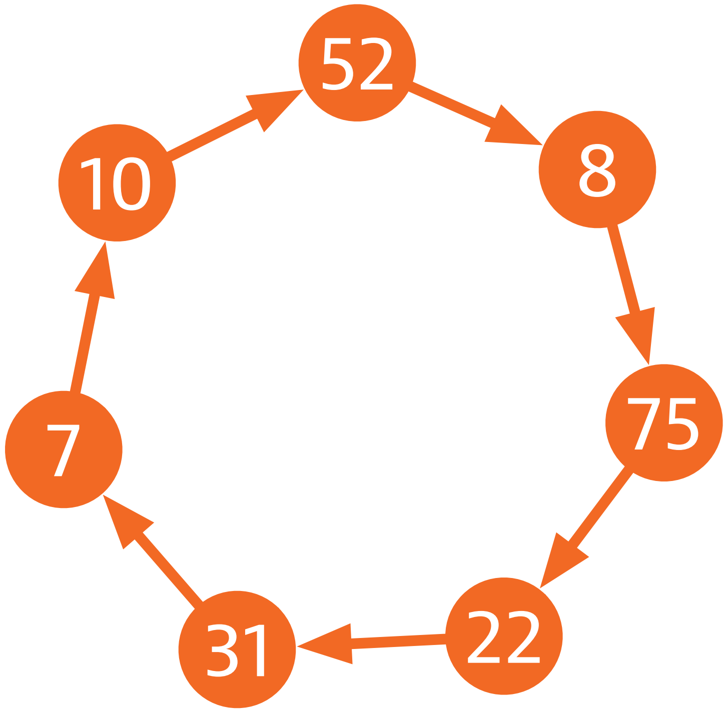
Setting

- ***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

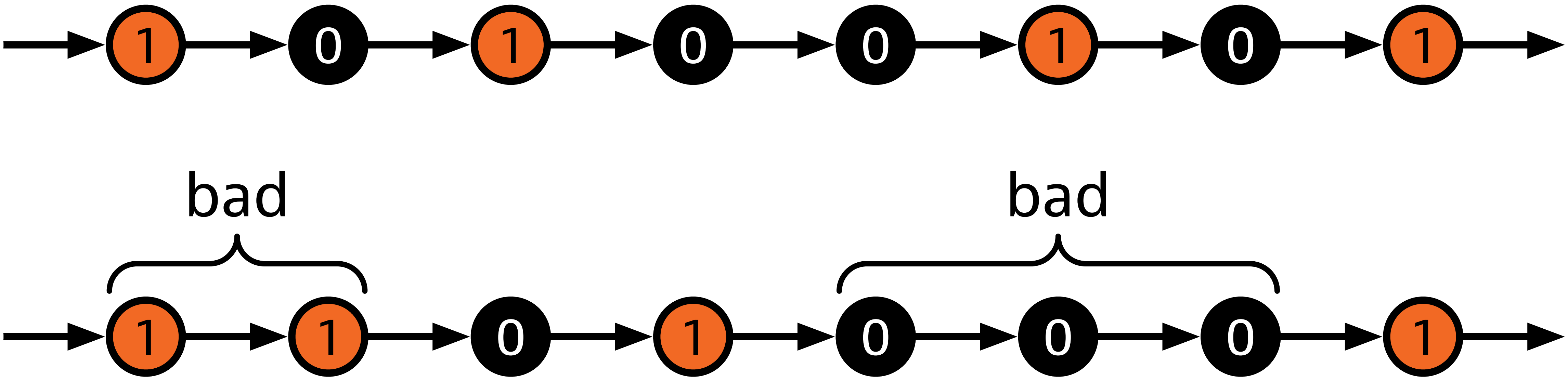


Setting

- ***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

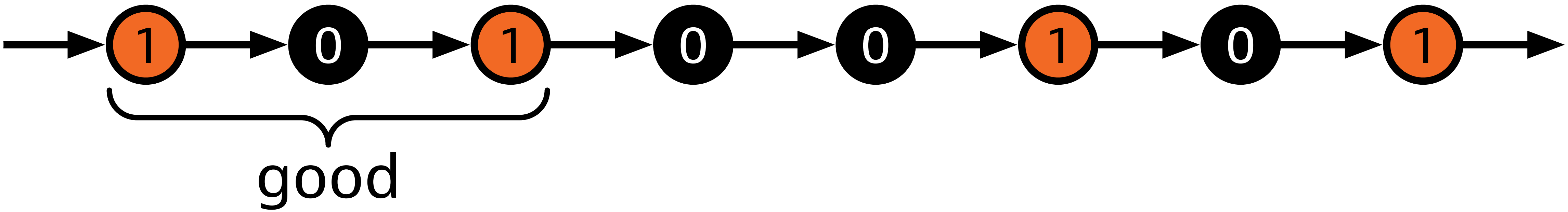
Local problems

- *Example:* **maximal independent set**
  - independent set = no two neighbors selected
  - maximal = cannot greedily add more



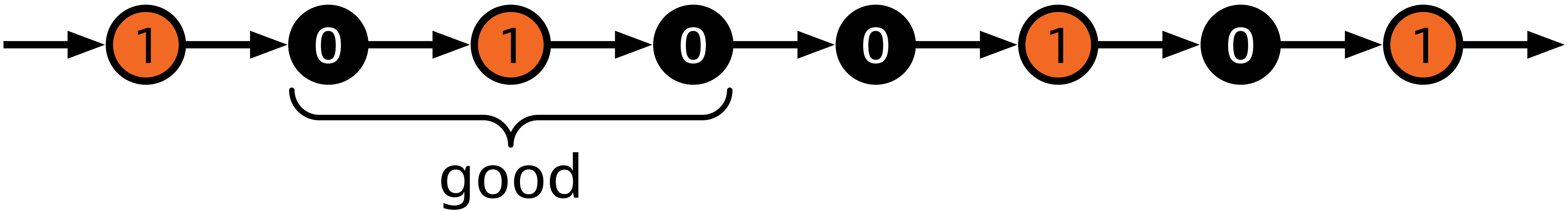
Local problems

- *Example:* **maximal independent set**
  - independent set = no two neighbors selected
  - maximal = cannot greedily add more



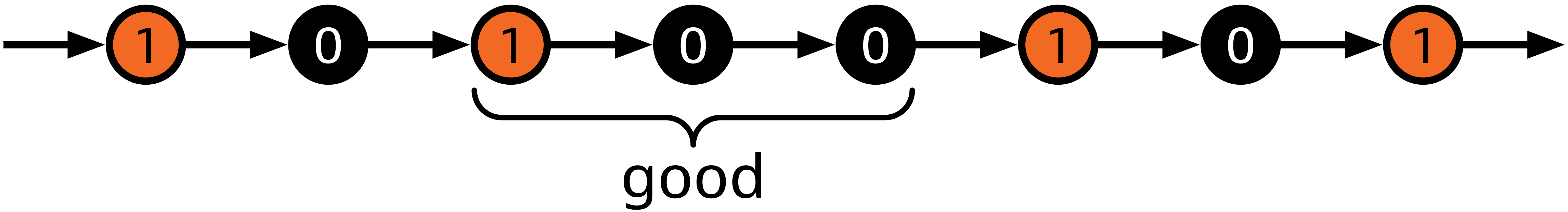
Local problems

- *Example:* **maximal independent set**
  - independent set = no two neighbors selected
  - maximal = cannot greedily add more



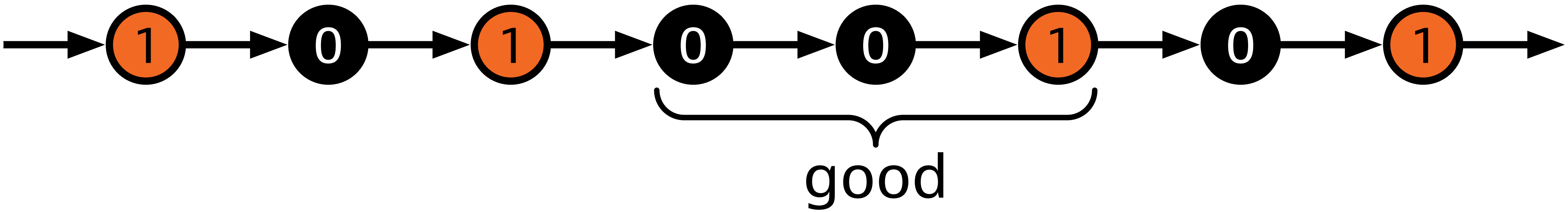
Local problems

- *Example:* **maximal independent set**
  - independent set = no two neighbors selected
  - maximal = cannot greedily add more



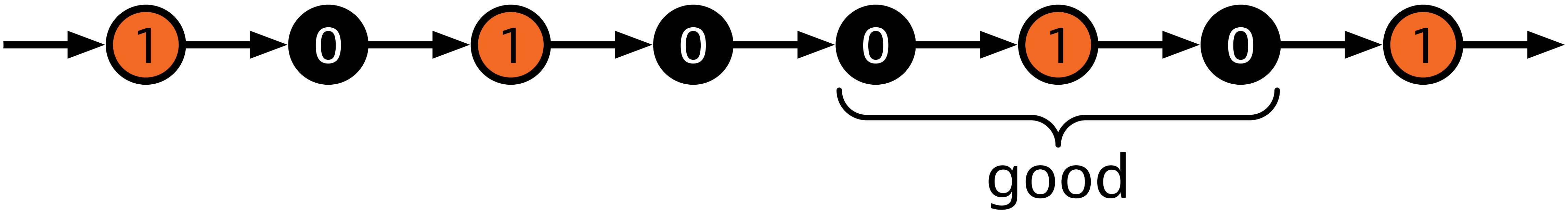
Local problems

- *Example:* **maximal independent set**
  - independent set = no two neighbors selected
  - maximal = cannot greedily add more



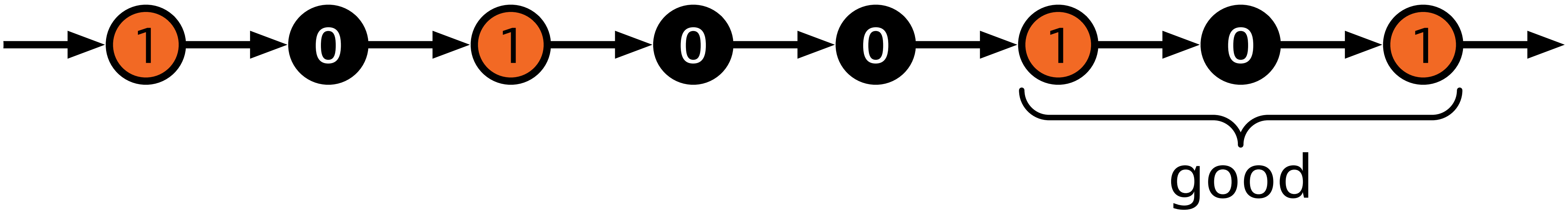
Local problems

- *Example:* **maximal independent set**
  - independent set = no two neighbors selected
  - maximal = cannot greedily add more



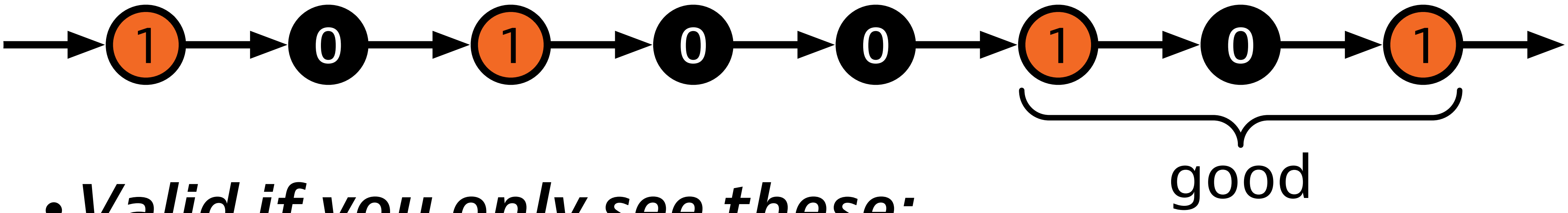
Local problems

- *Example:* **maximal independent set**
  - independent set = no two neighbors selected
  - maximal = cannot greedily add more



Local problems

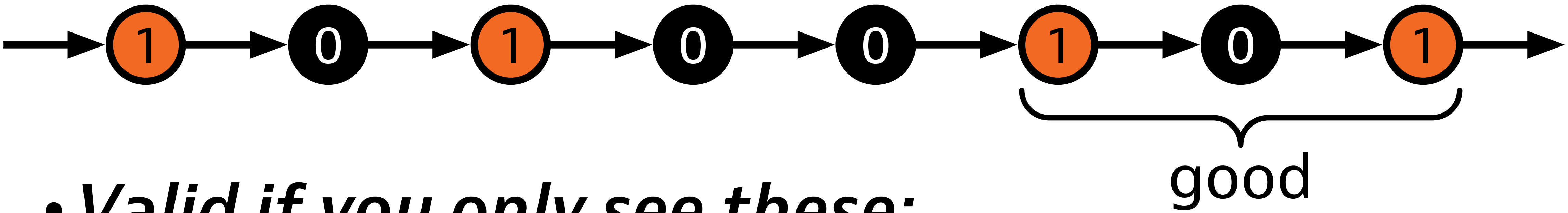
- *Example:* **maximal independent set**
  - independent set = no two neighbors selected
  - maximal = cannot greedily add more



- ***Valid if you only see these:***
  ???, ???, ???, ???

Local problems

- ***Example:*** **maximal independent set**
  - independent set = no two neighbors selected
  - maximal = cannot greedily add more

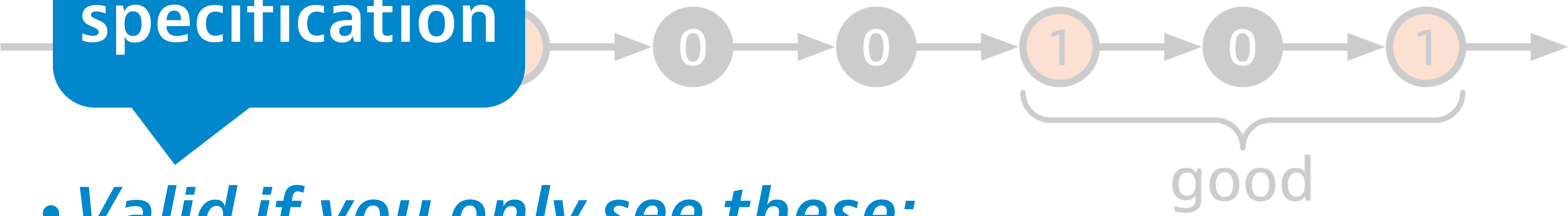


- ***Valid if you only see these:***
  001, 010, 100, 101

Local problems

- *Example:* **maximal independent set**
  - independent set = no two neighbors selected
  - not greedily add more

Problem specification



- ***Valid if you only see these:***
  001, 010, 100, 101

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

X = { 001, 010, 100, 101 }

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

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)

How?

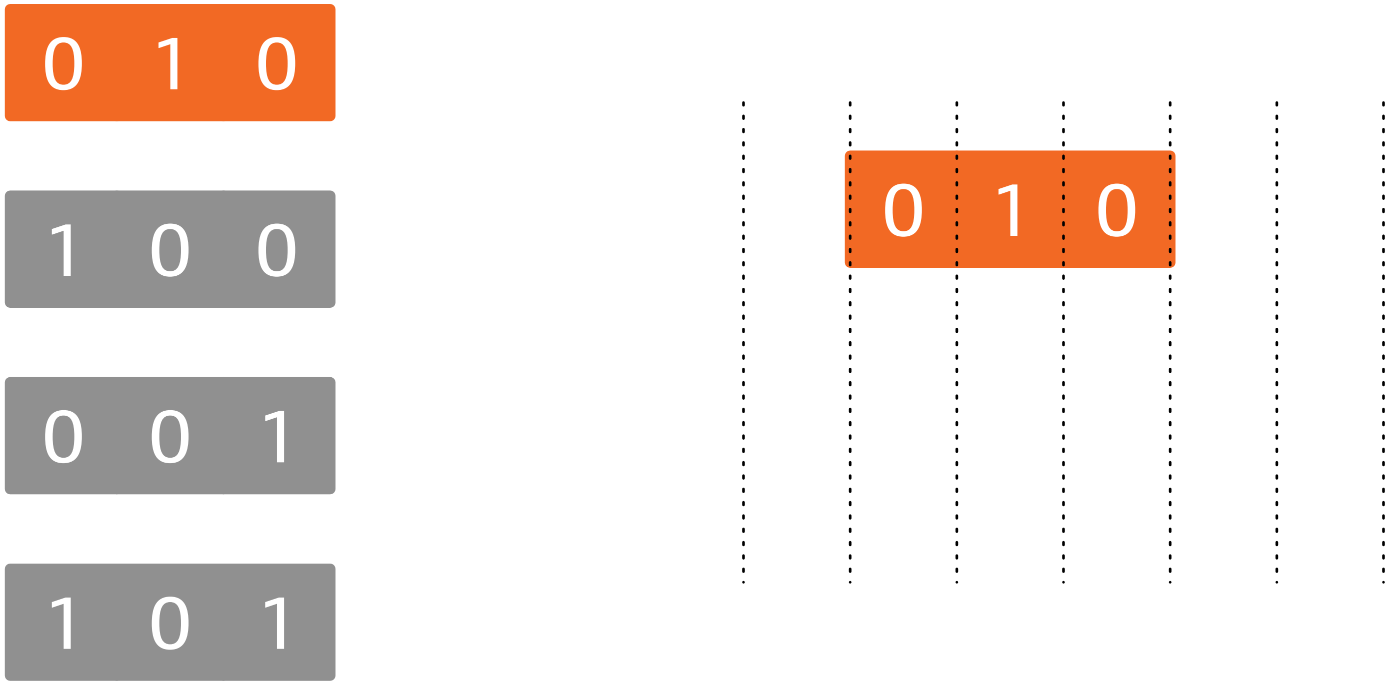
0 1 0

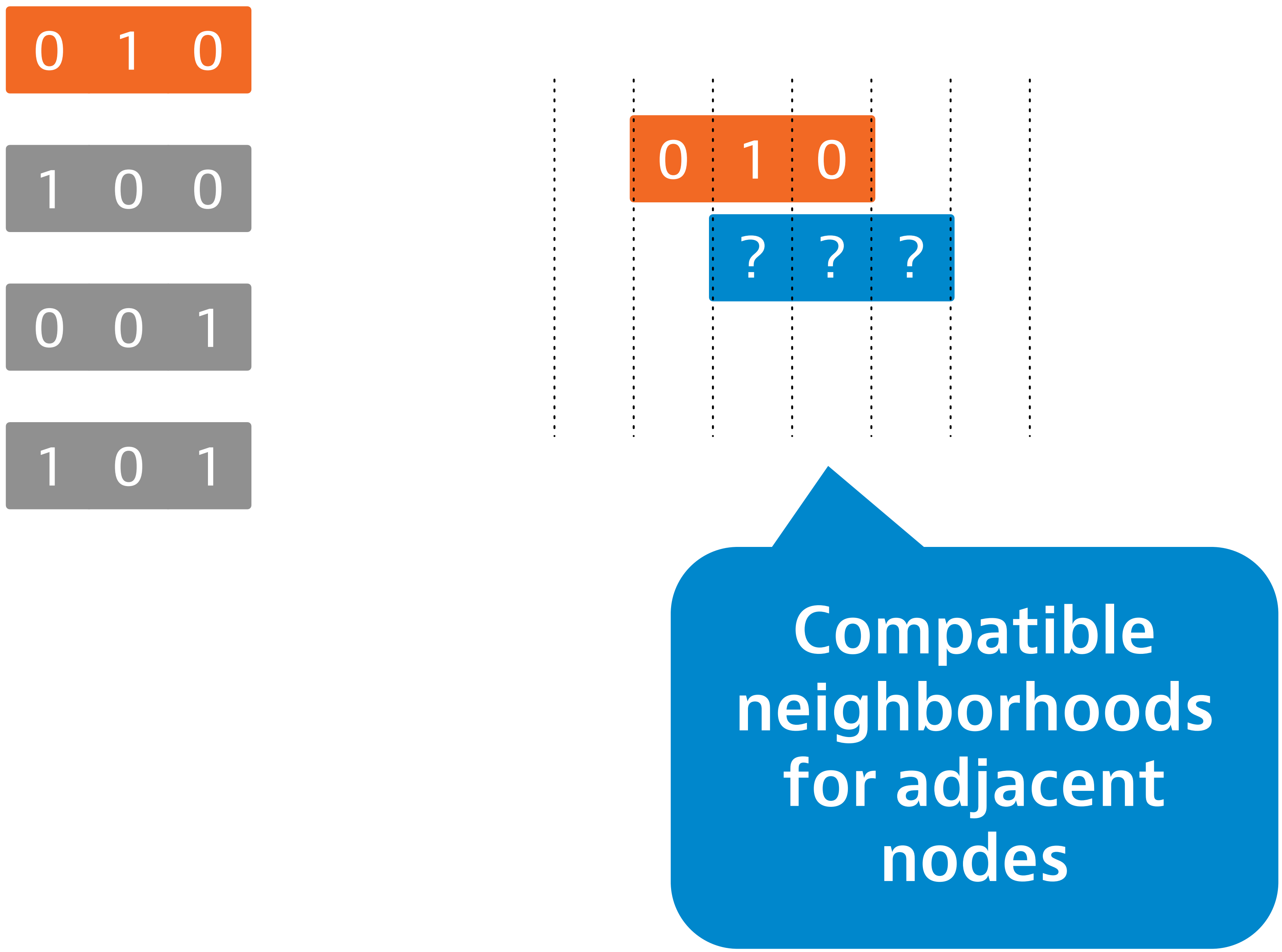
1 0 0

0 0 1

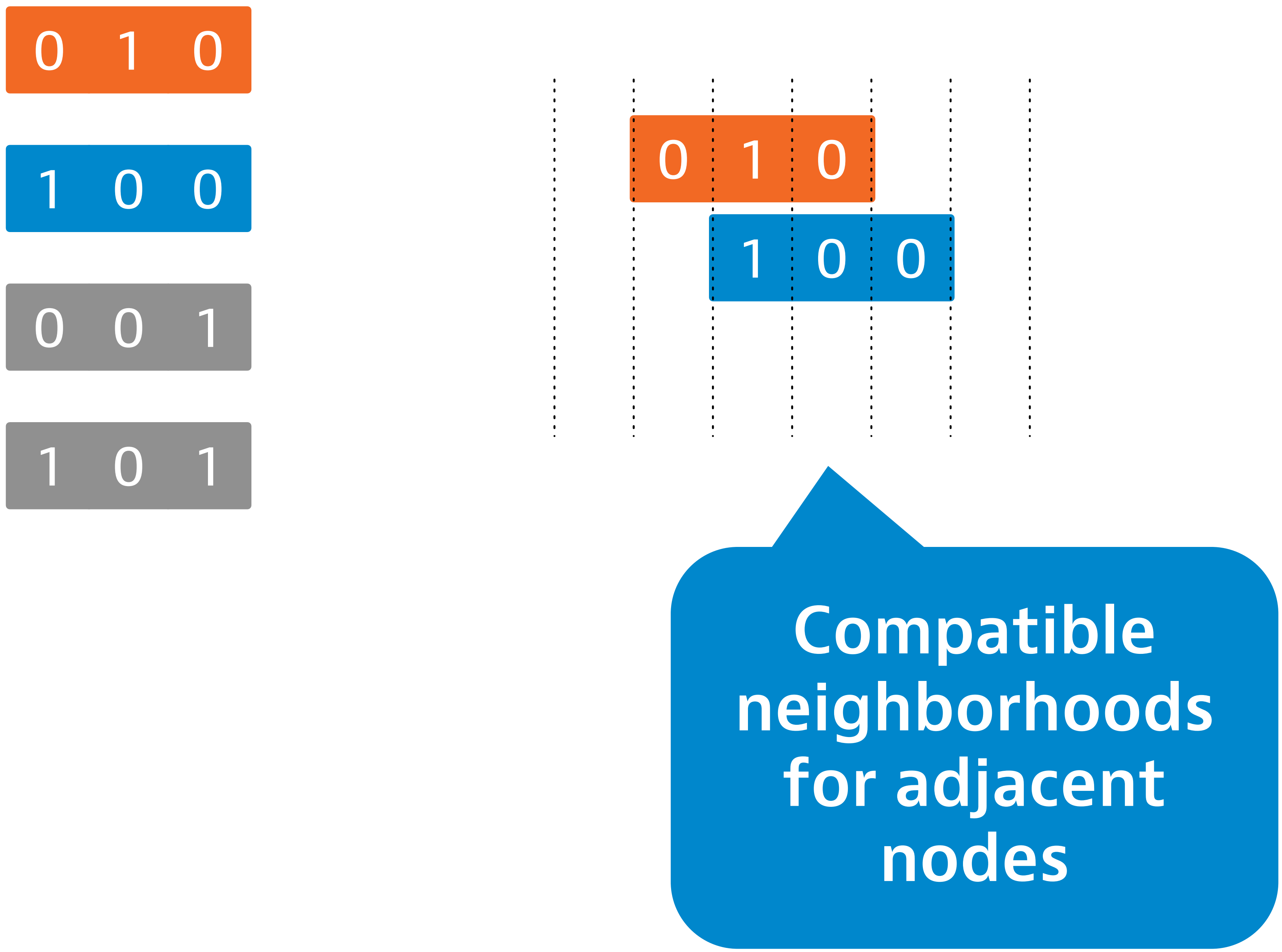
1 0 1

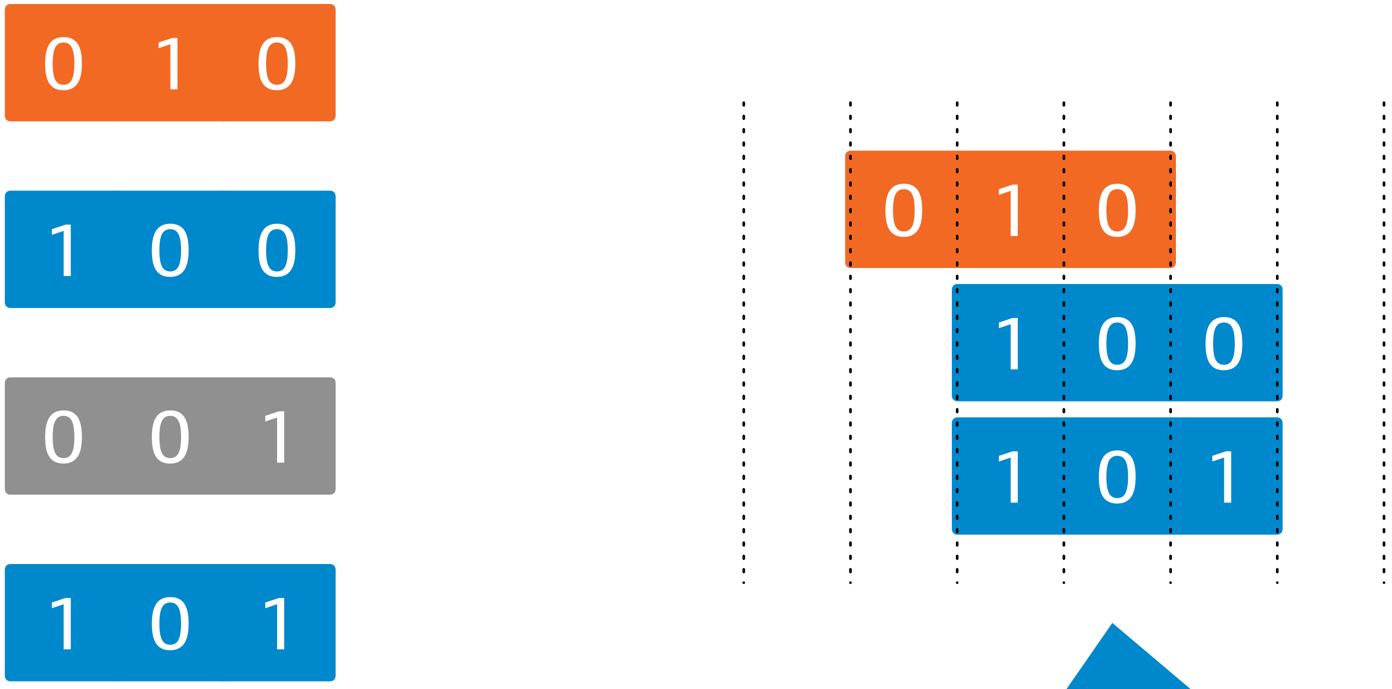
Example:
X = maximal independent set problem


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

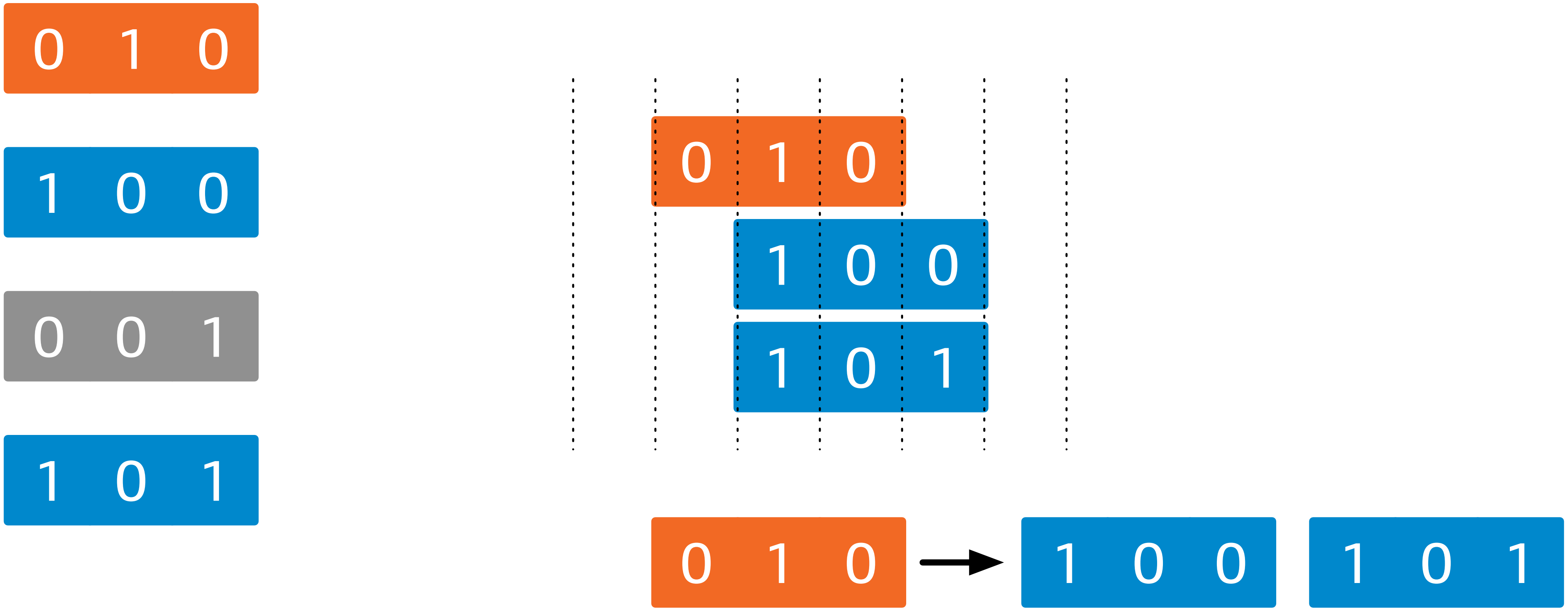


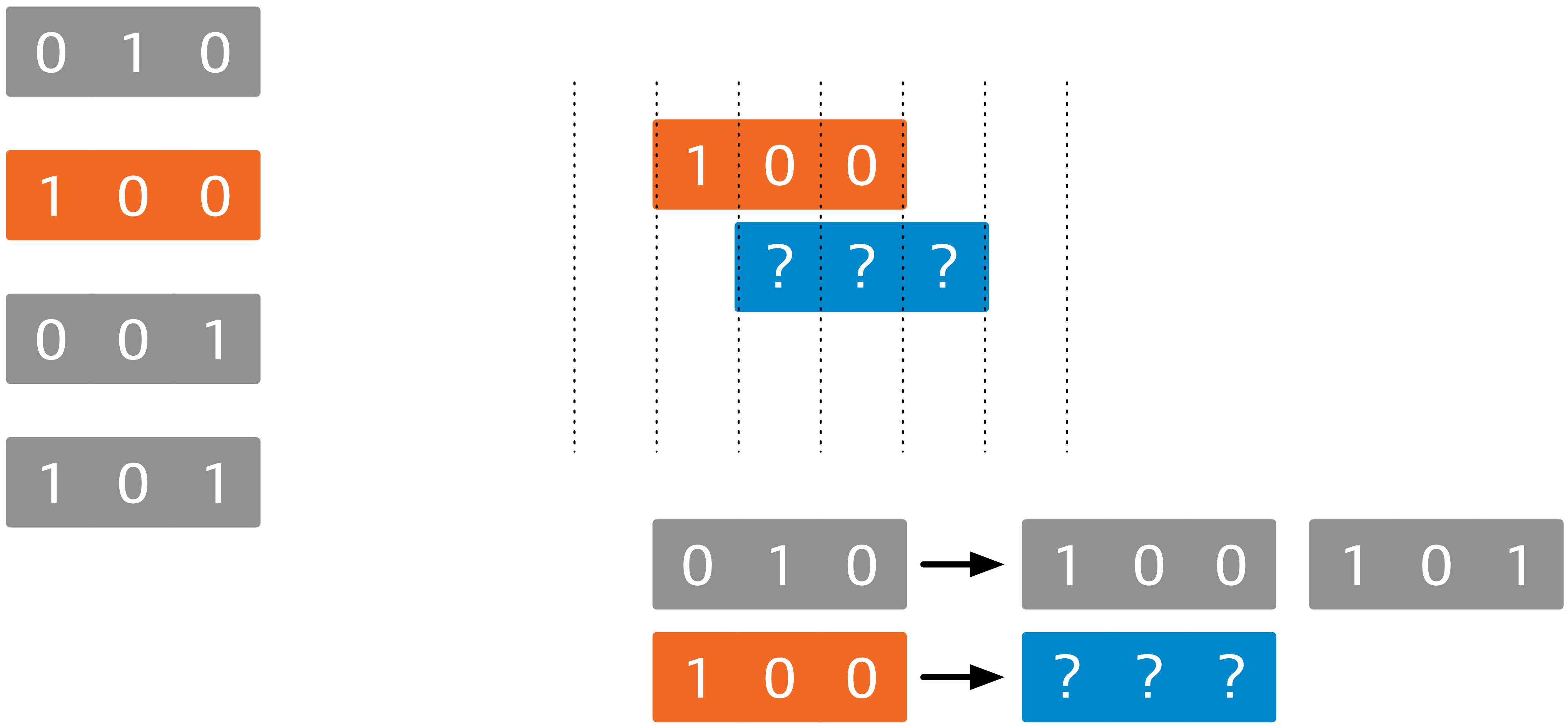
0 1 0
1 0 0
0 0 1
1 0 1
0 1 0
? ? ?
Compatible neighborhoods for adjacent nodes

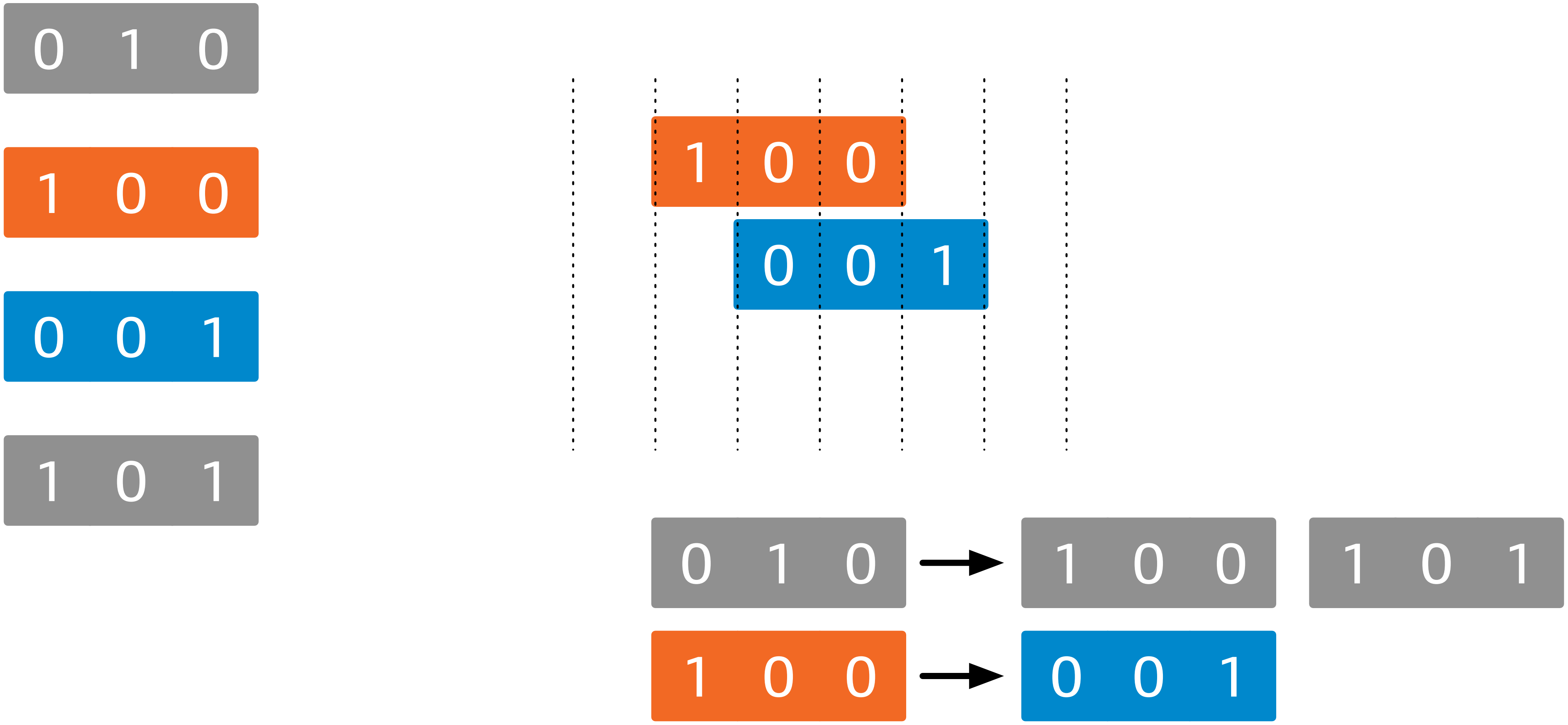

0 1 0
1 0 0
0 0 1
1 0 1
0 1 0
1 0 0
Compatible neighborhoods for adjacent nodes

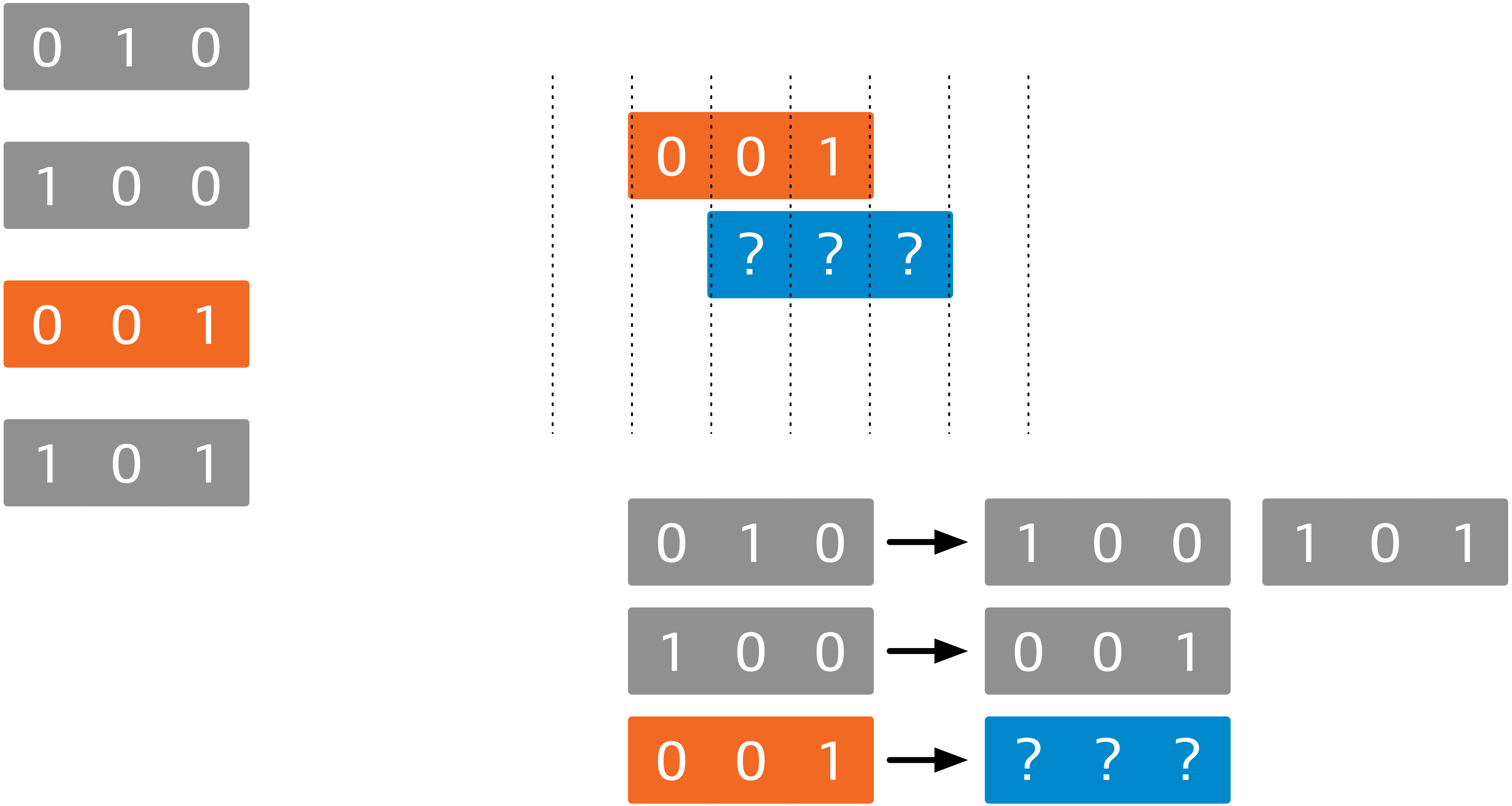


Compatible neighborhoods for adjacent nodes

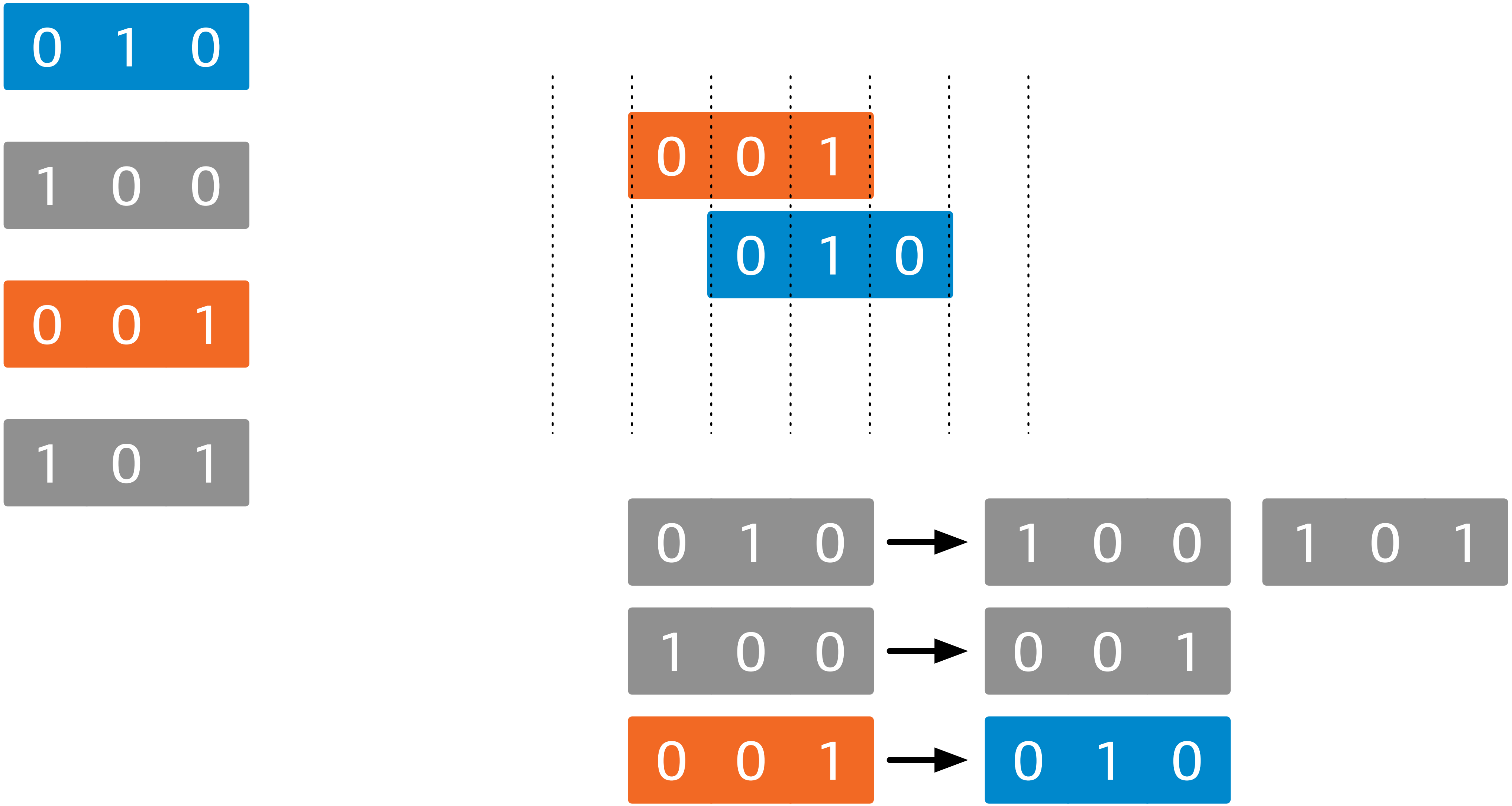

0 1 0
1 0 0
0 0 1
1 0 1
0 1 0
1 0 0
1 0 1
0 1 0
1 0 0
1 0 1

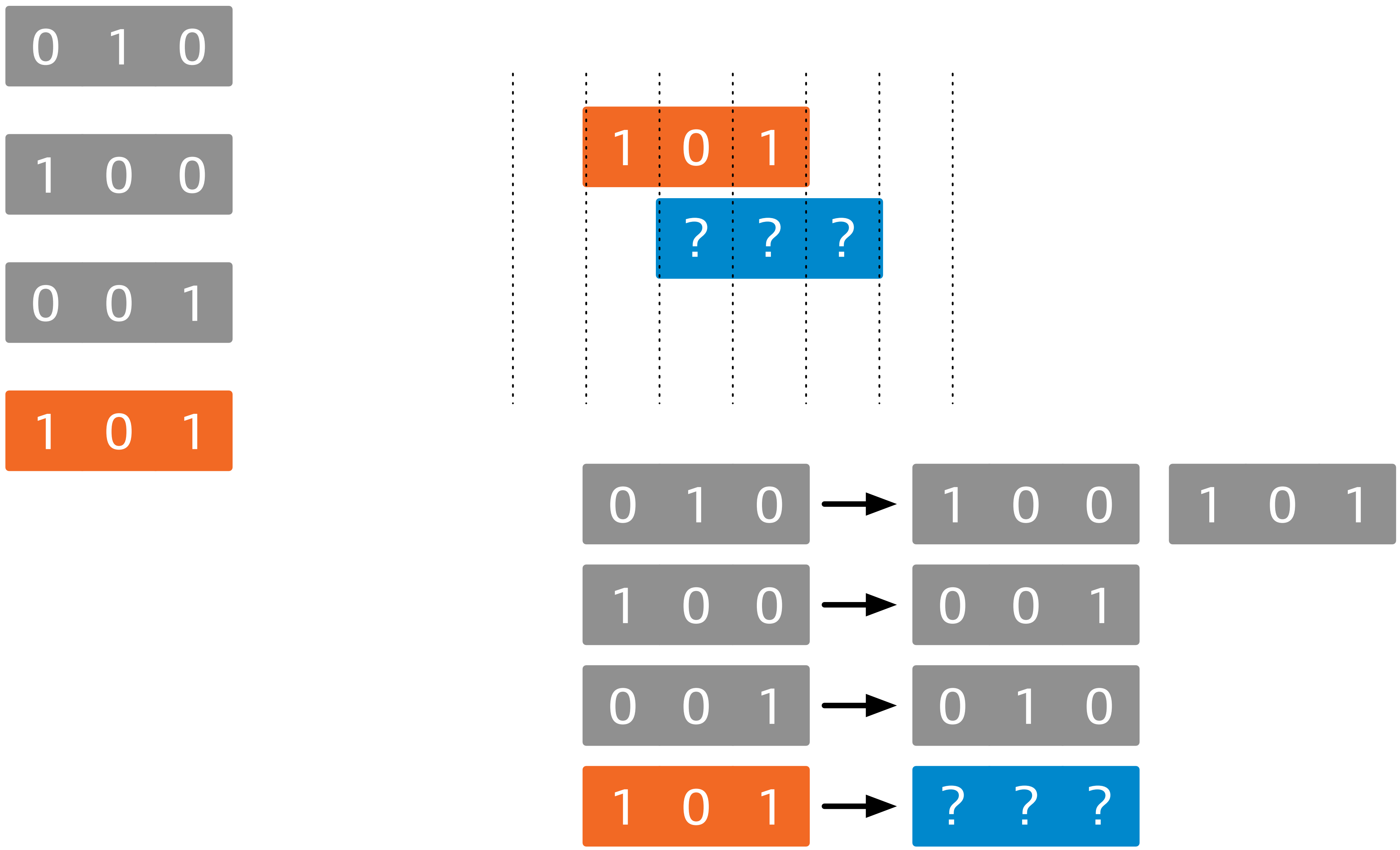

0 1 0
1 0 0
0 0 1
1 0 1
1 0 0
? ? ?
0 1 0
1 0 0
1 0 1
1 0 0
? ? ?


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

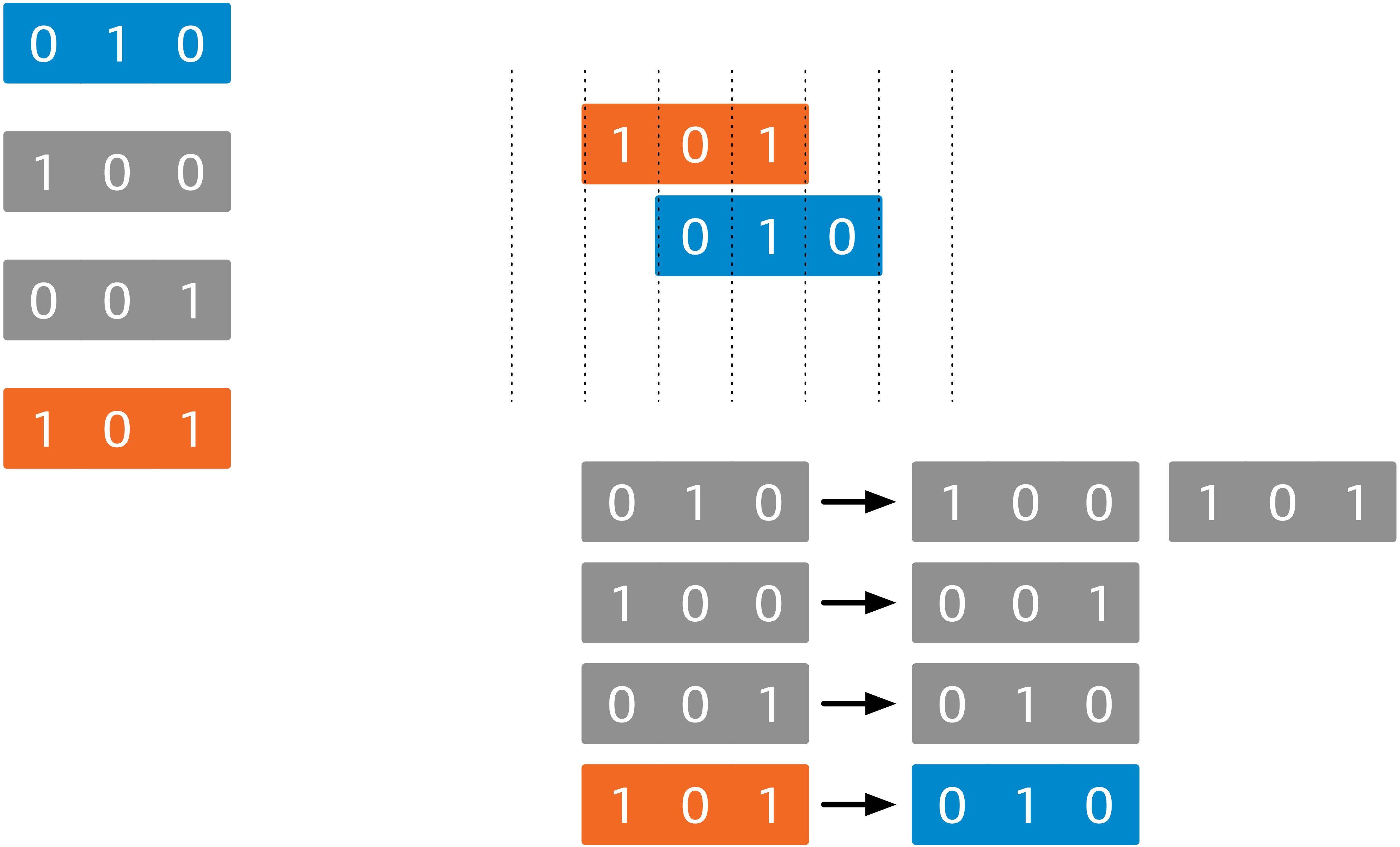


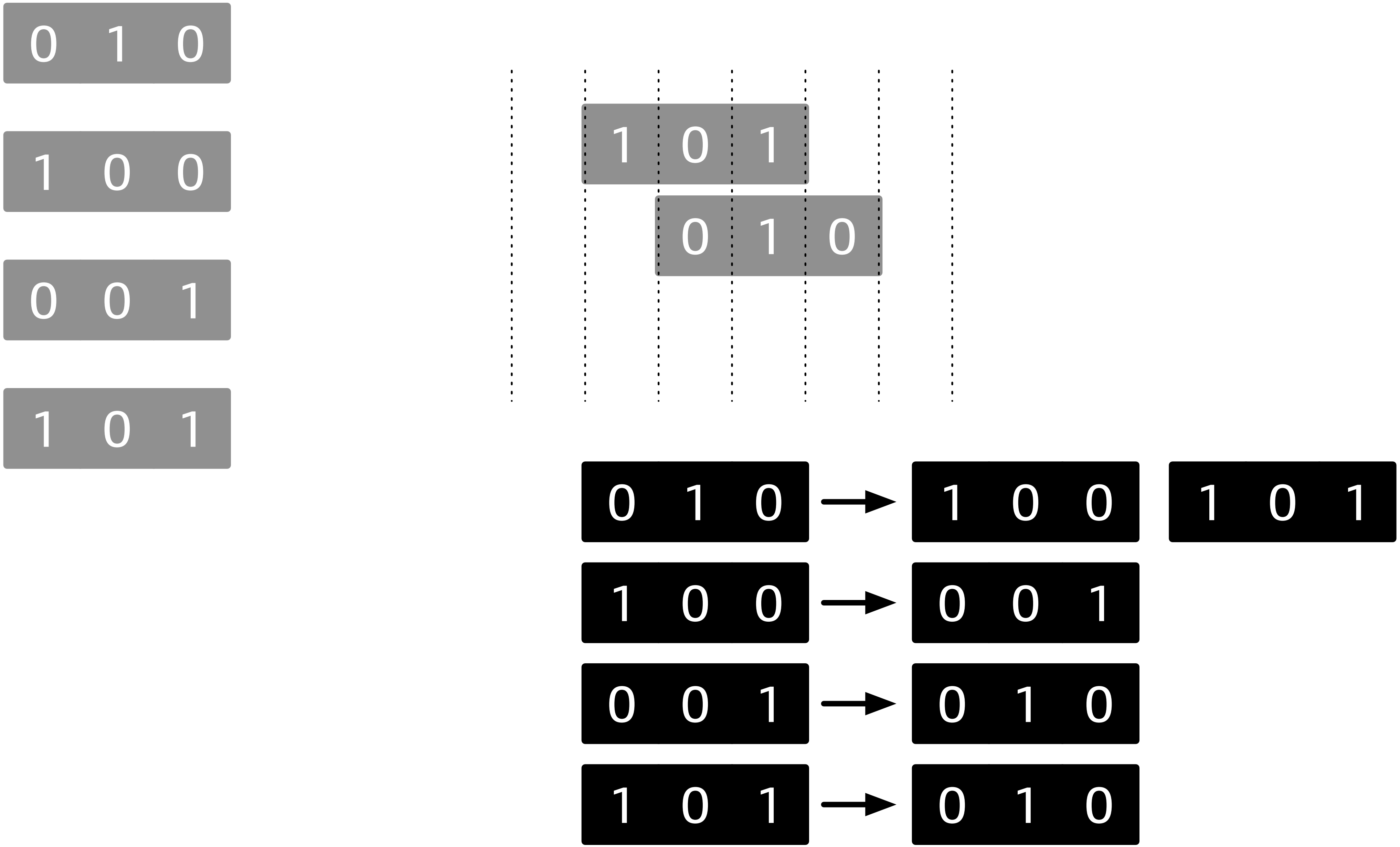
0 1 0
1 0 0
0 0 1
1 0 1
0 0 1
? ? ?
0 1 0
1 0 0
1 0 1
1 0 0
0 0 1
0 0 1
? ? ?

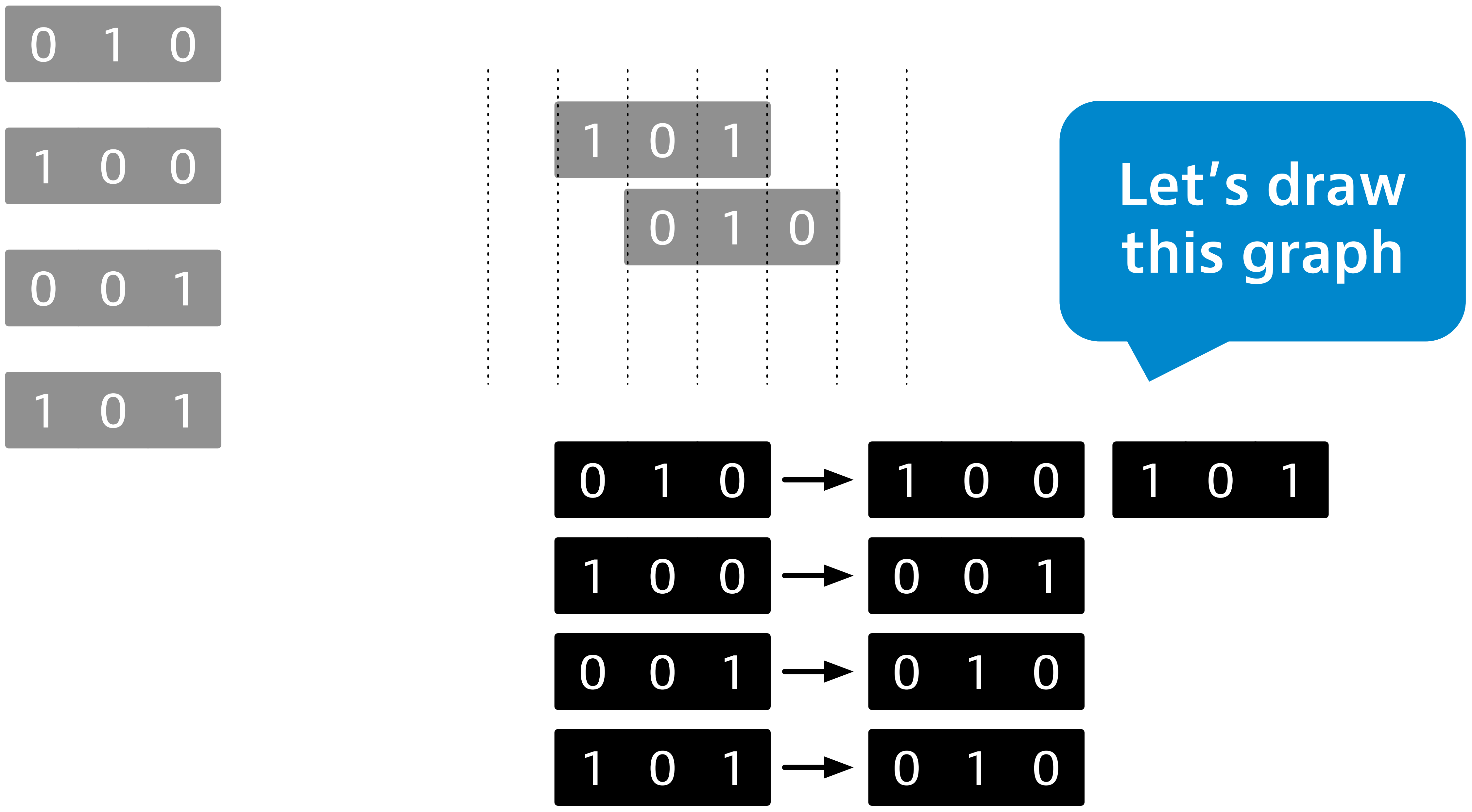

0 1 0
1 0 0
0 0 1
1 0 1
0 0 1
0 1 0
0 1 0
1 0 0
1 0 1
1 0 0
0 0 1
0 0 1
0 1 0

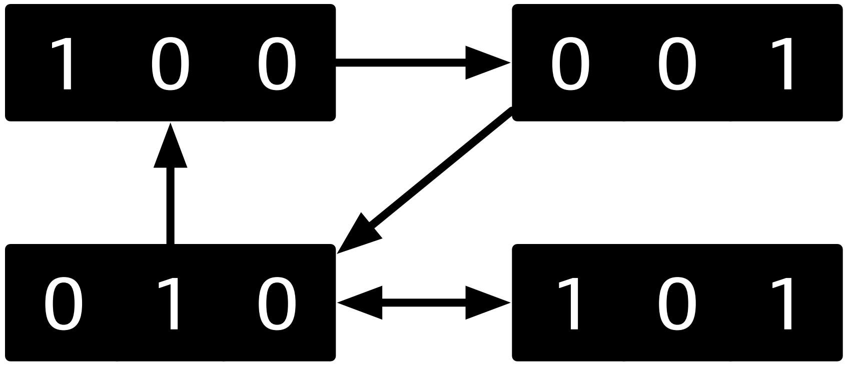


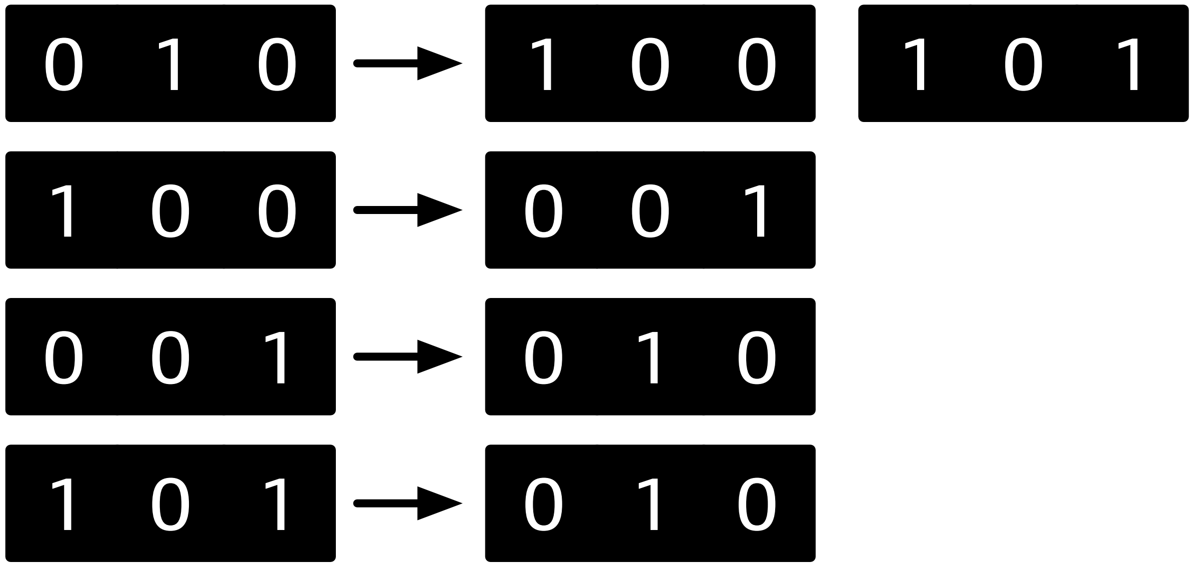
0 1 0
1 0 0
0 0 1
1 0 1
1 0 1
? ? ?
0 1 0
1 0 0
1 0 1
1 0 0
0 0 1
0 0 1
0 1 0
1 0 1
? ? ?

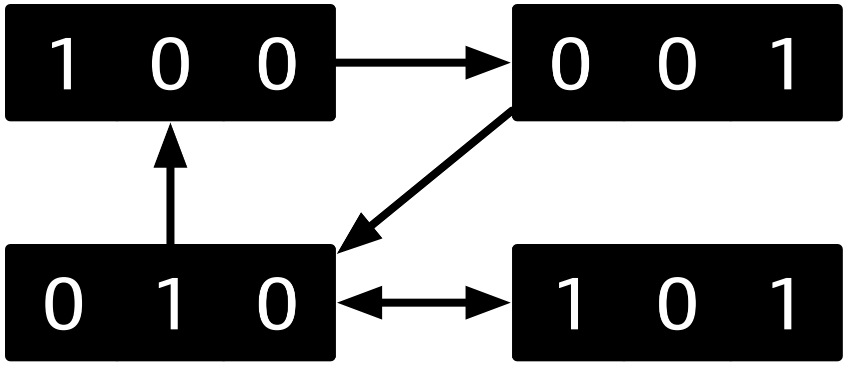

0 1 0
1 0 0
0 0 1
1 0 1
1 0 1
0 1 0
0 1 0
1 0 0
1 0 1
1 0 0
0 0 1
0 0 1
0 1 0
1 0 1
0 1 0


0 1 0
1 0 0
0 0 1
1 0 1
1 0 1
0 1 0
0 1 0 → 1 0 0
1 0 1
1 0 0 → 0 0 1
0 0 1 → 0 1 0
1 0 1 → 0 1 0

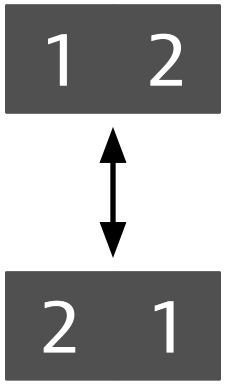

0 1 0
1 0 0
0 0 1
1 0 1
1 0 1
0 1 0
Let's draw this graph
0 1 0 → 1 0 0
1 0 0 → 0 0 1
0 0 1 → 0 1 0
1 0 1 → 0 1 0
1 0 1


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

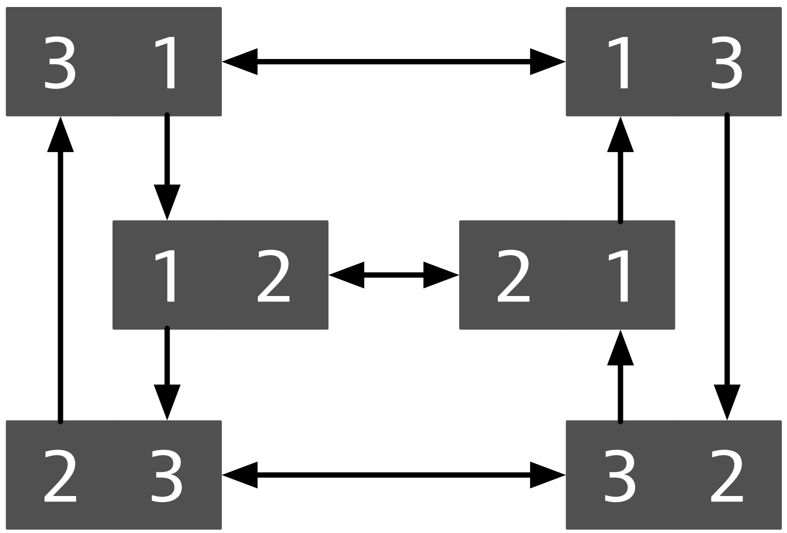

0 1 0
1 0 0
1 0 1
1 0 0
0 0 1
0 0 1
0 1 0
1 0 1
0 1 0



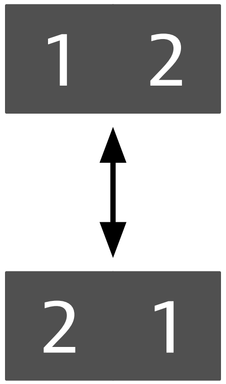
This graph is all that we need!



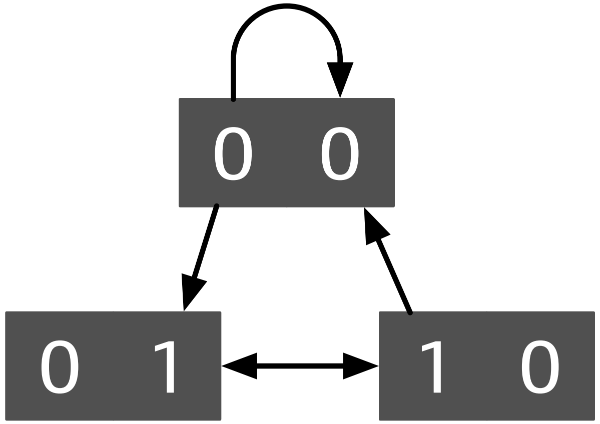
2-coloring



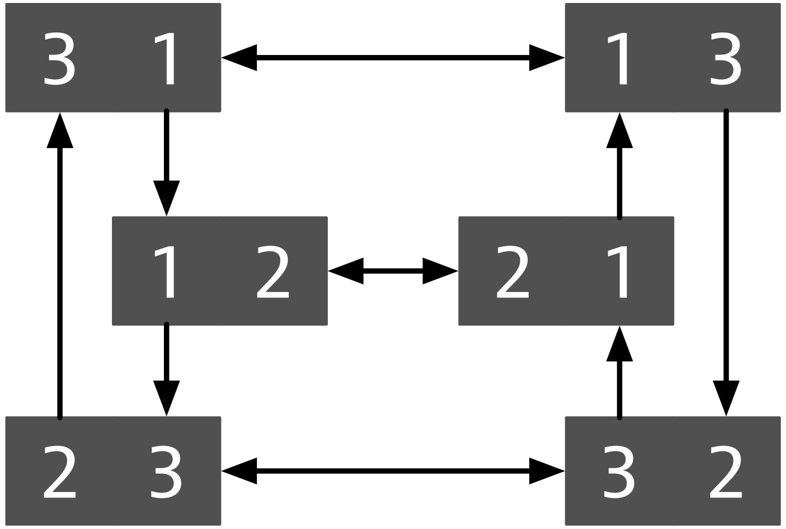
3-coloring



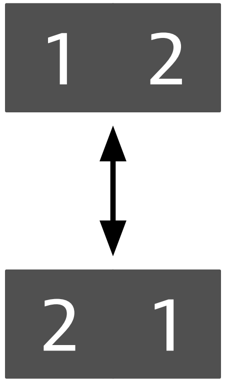
2-coloring



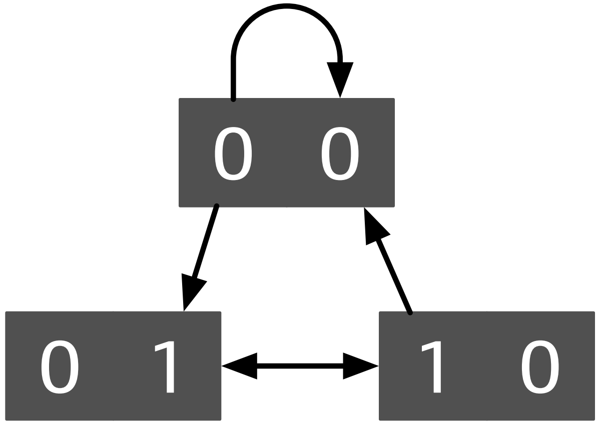
independent set



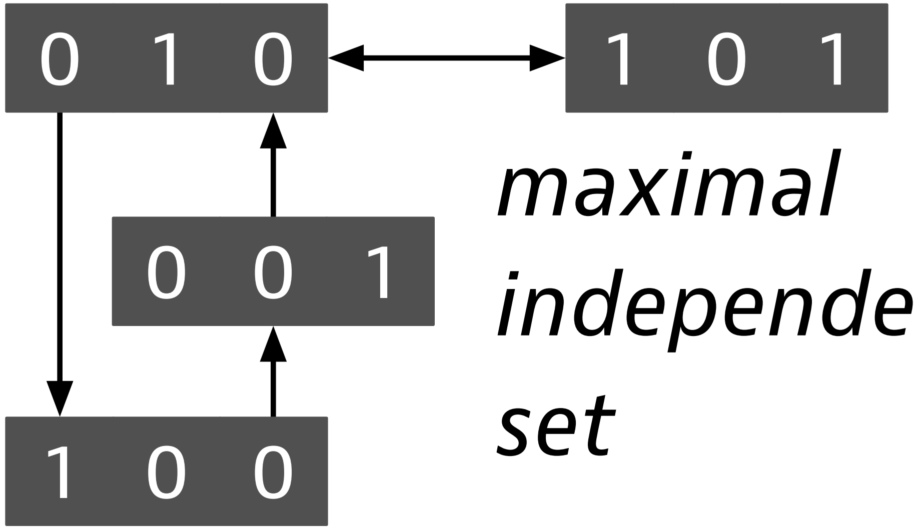
3-coloring



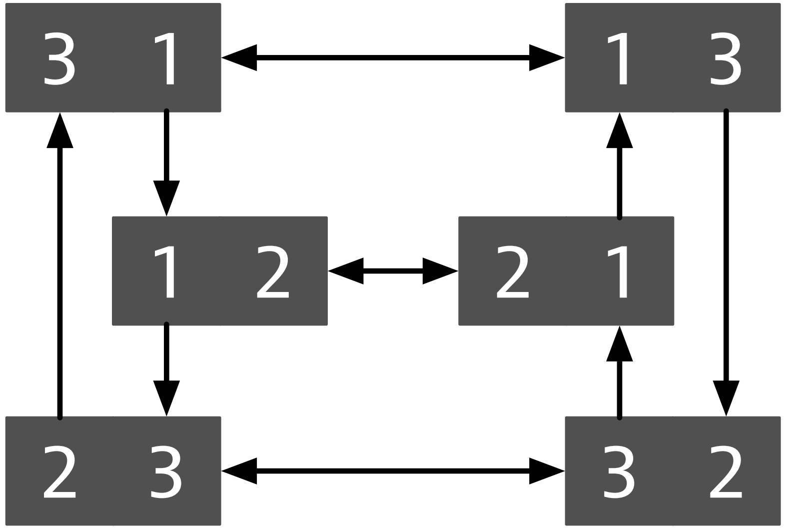
2-coloring



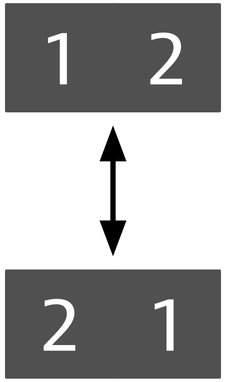
independent set



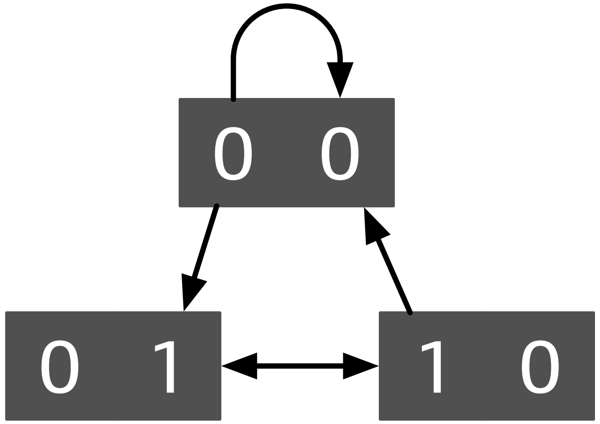
maximal independent set



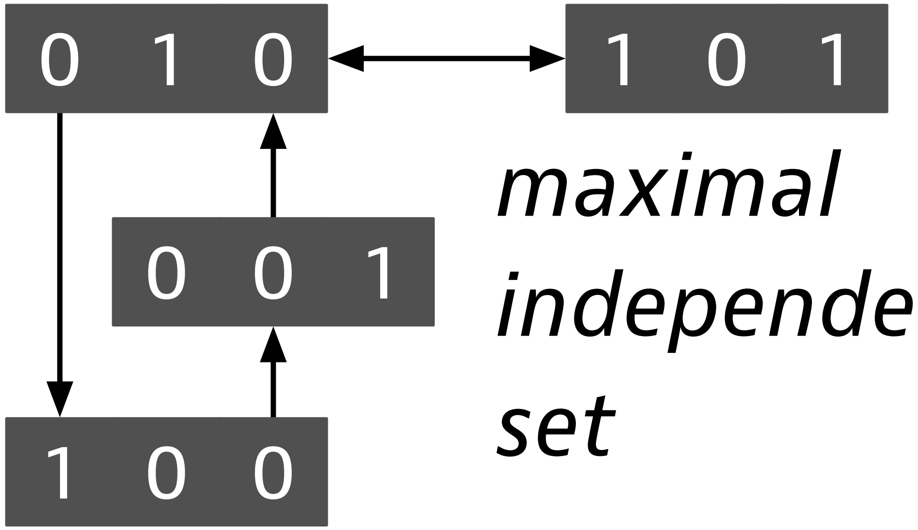
3-coloring



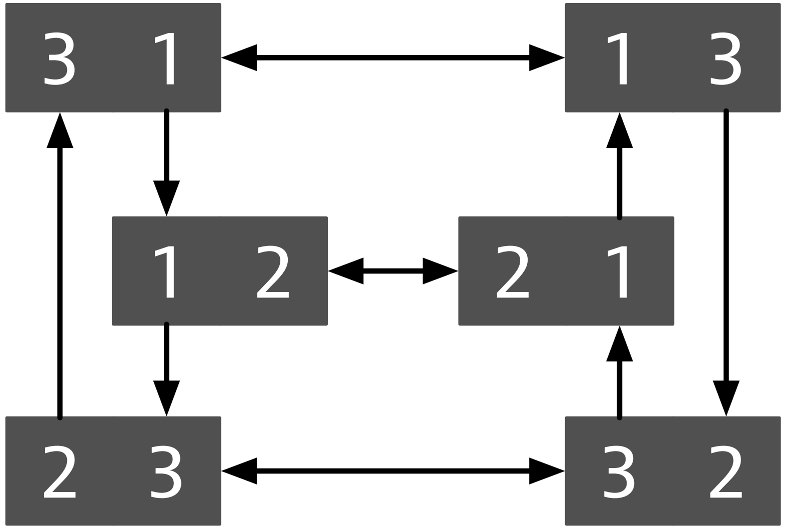
2-coloring



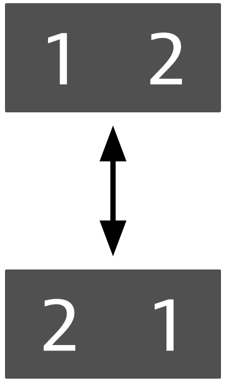
independent set



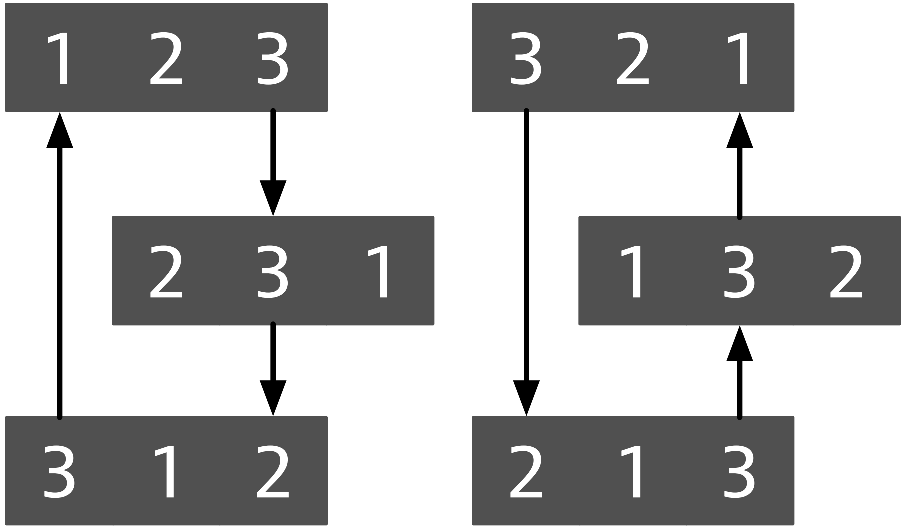
maximal independent set



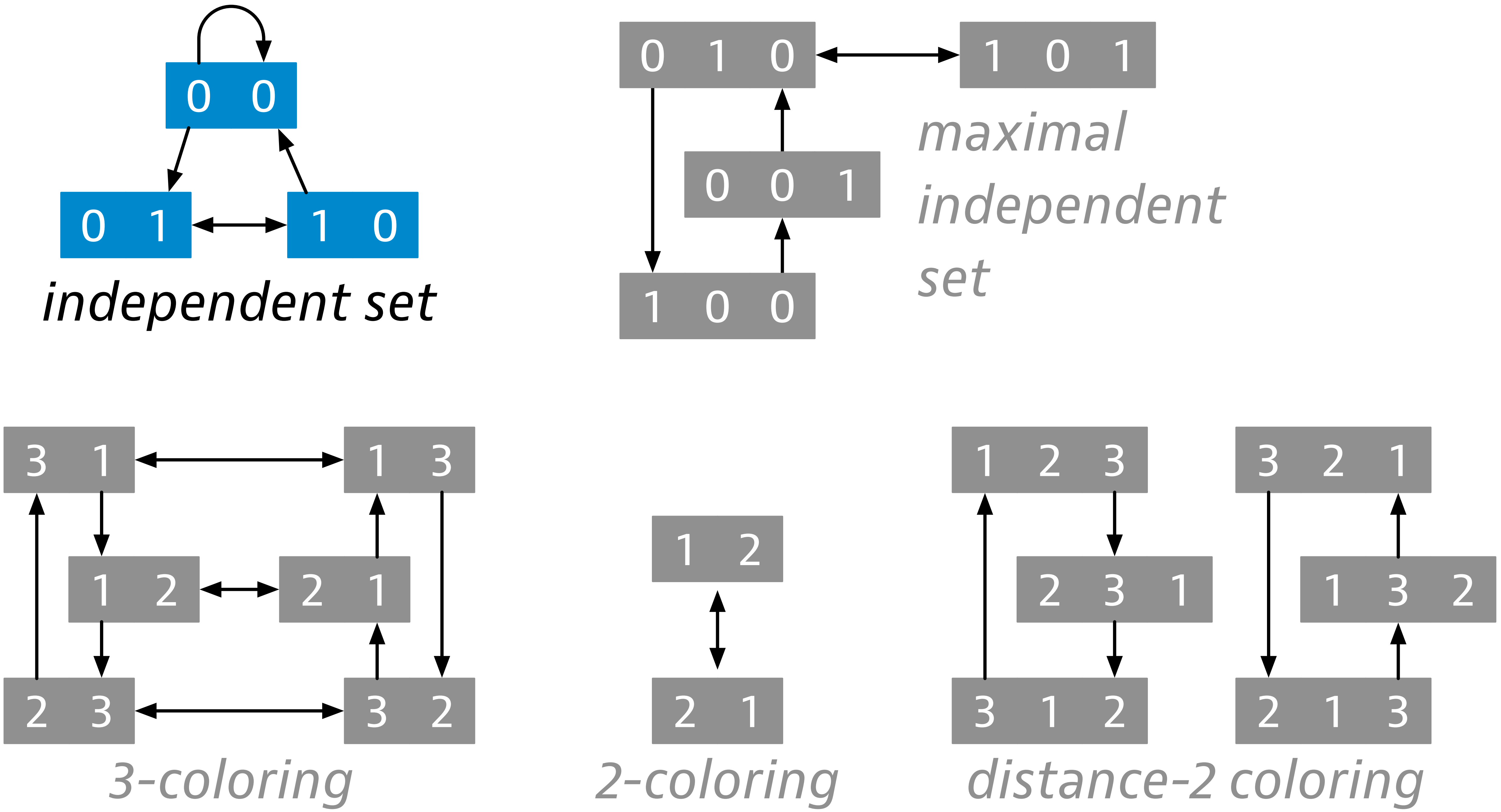
3-coloring

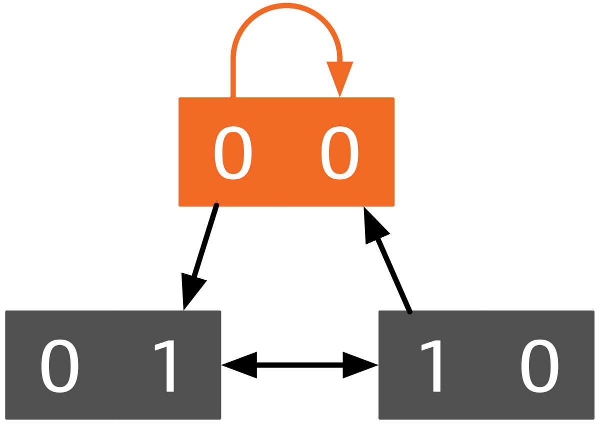


2-coloring



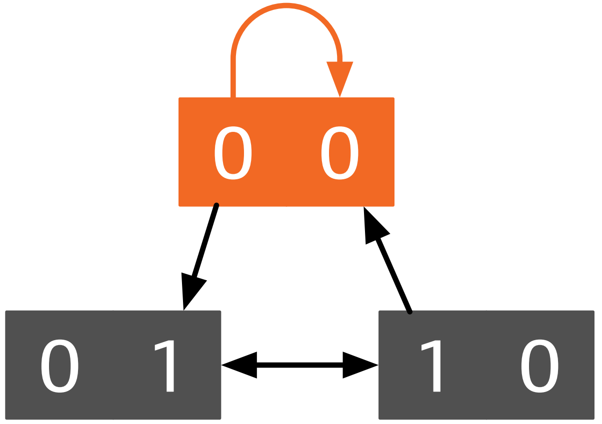
distance-2 coloring


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

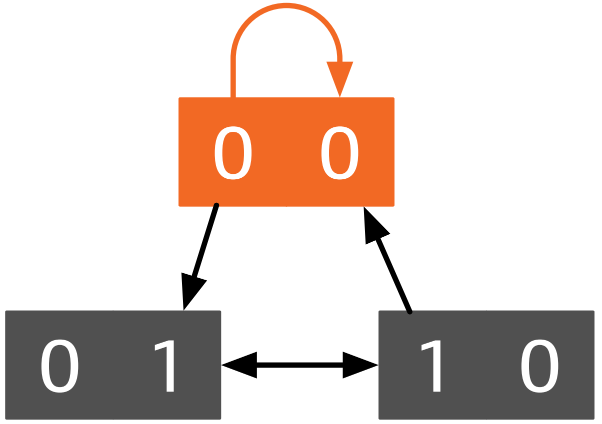


independent set

self-loop

solvable in $O(1)$ rounds

Algorithm:
?

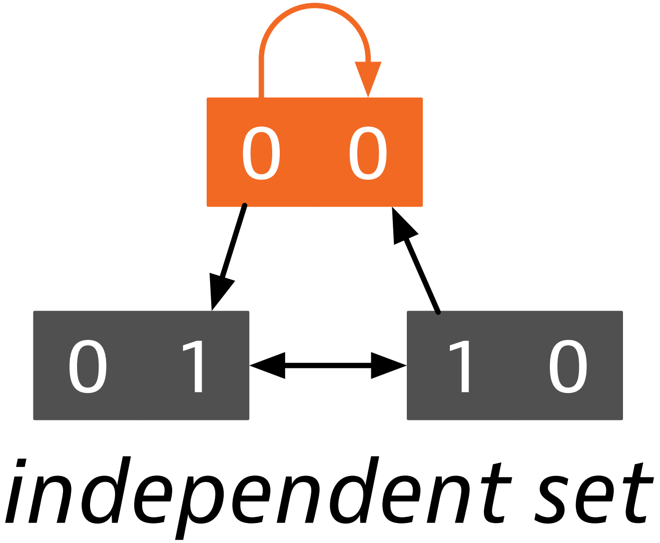


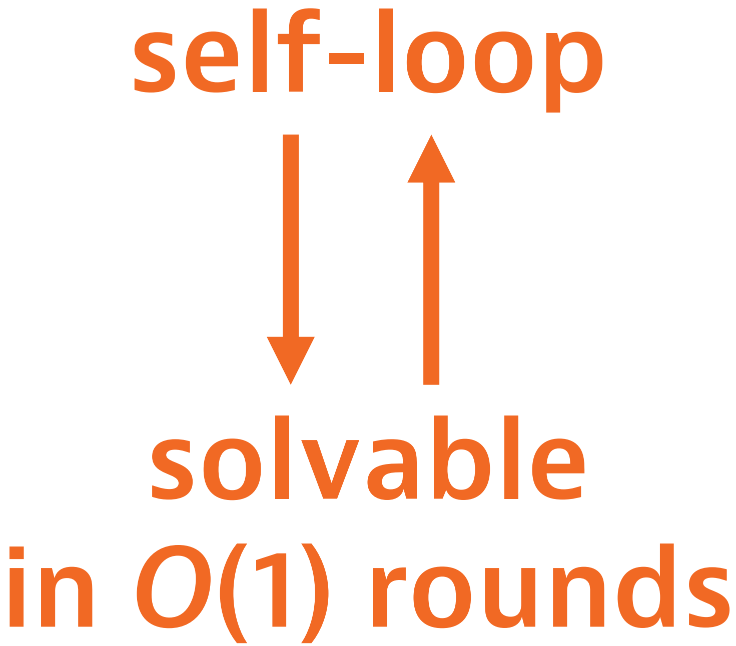
independent set

self-loop

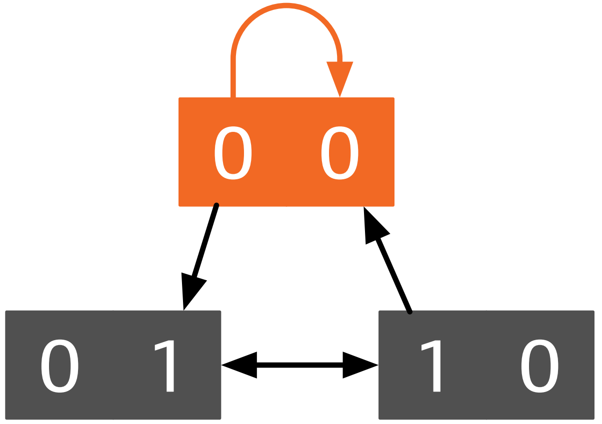
solvable in $O(1)$ rounds

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





Proof: ?



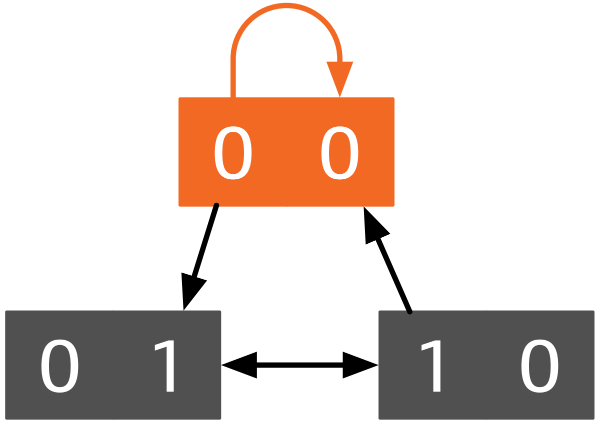
independent set

self-loop

↓ ↑

solvable
in $O(1)$ rounds

Proof: No self-loop
→ any solution breaks symmetry everywhere



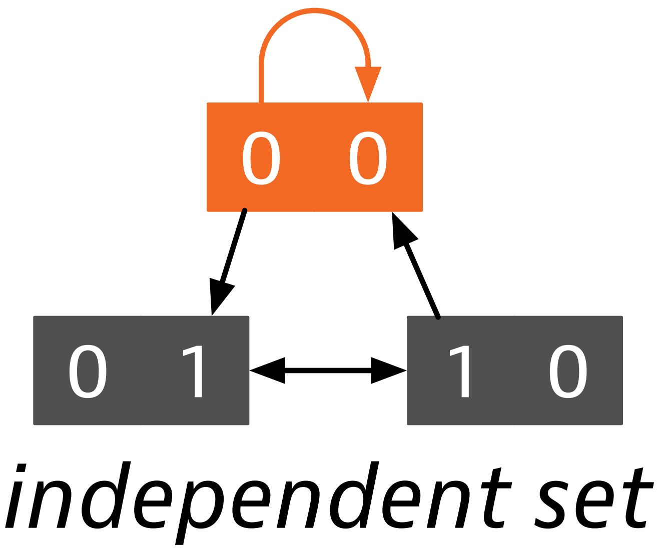
independent set

self-loop

↓↑

solvable in $O(1)$ rounds

Proof: No self-loop
→ any solution breaks symmetry everywhere
→ can be used to find 3-coloring

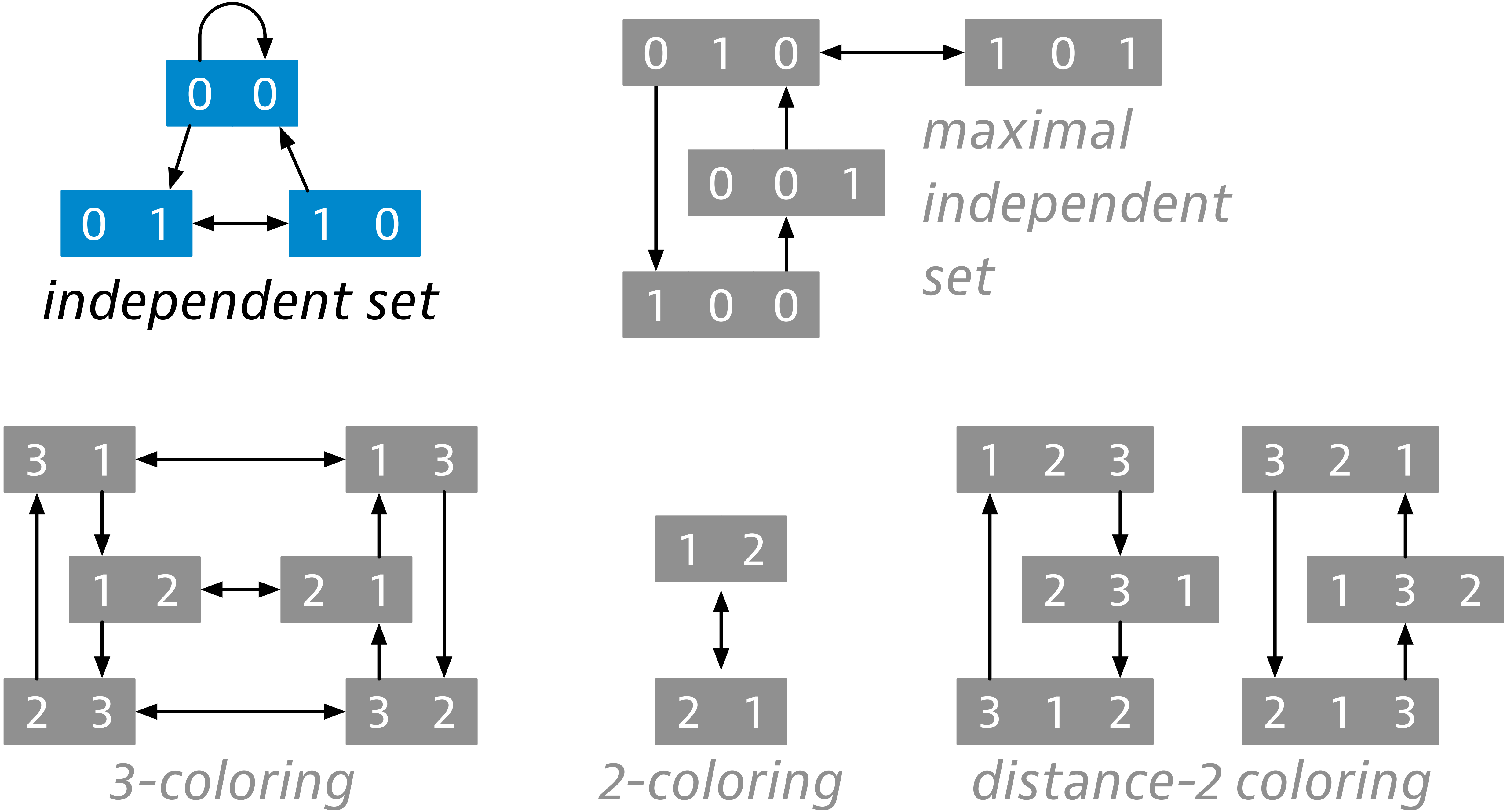


self-loop

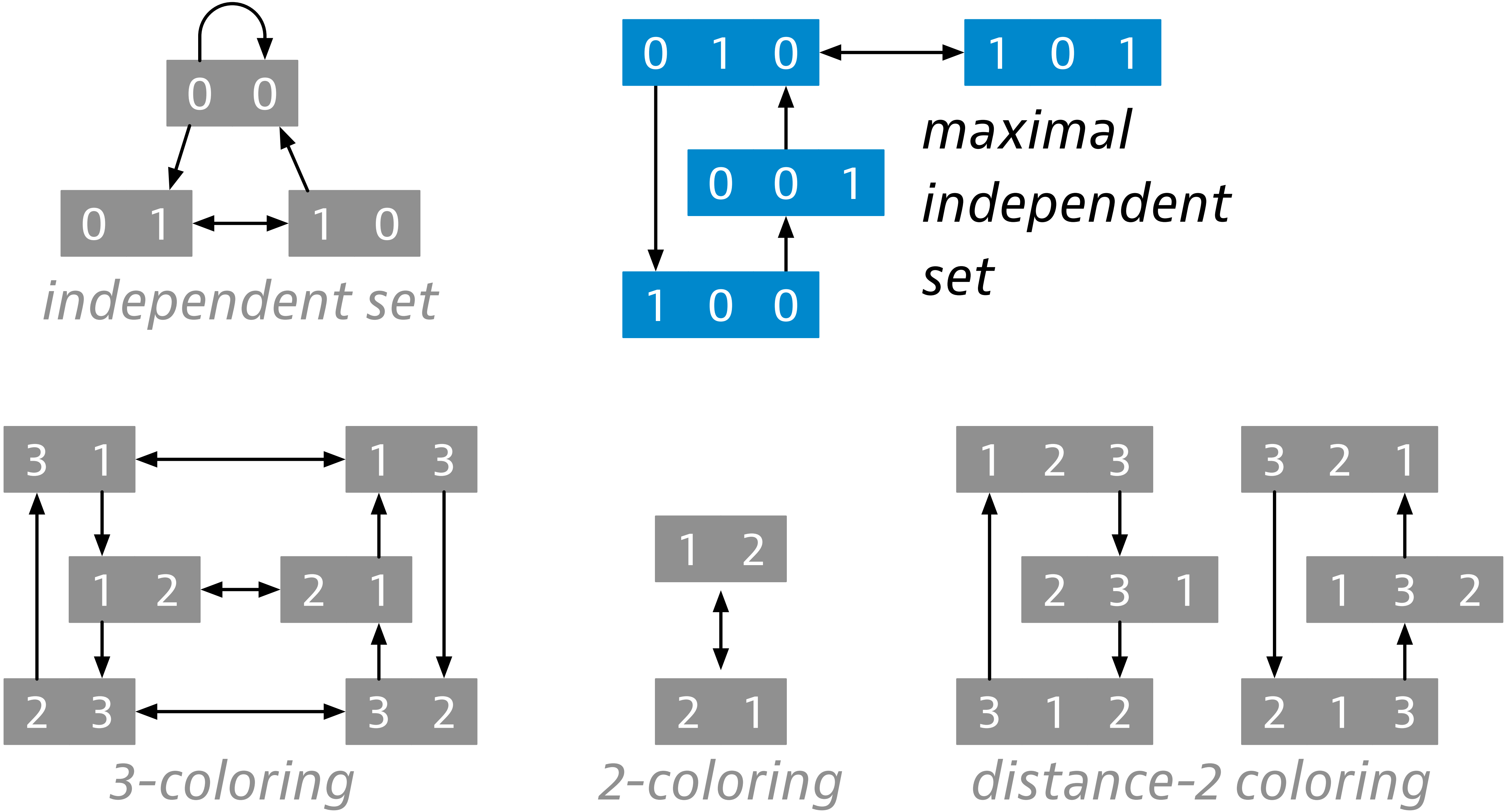
↓ ↑

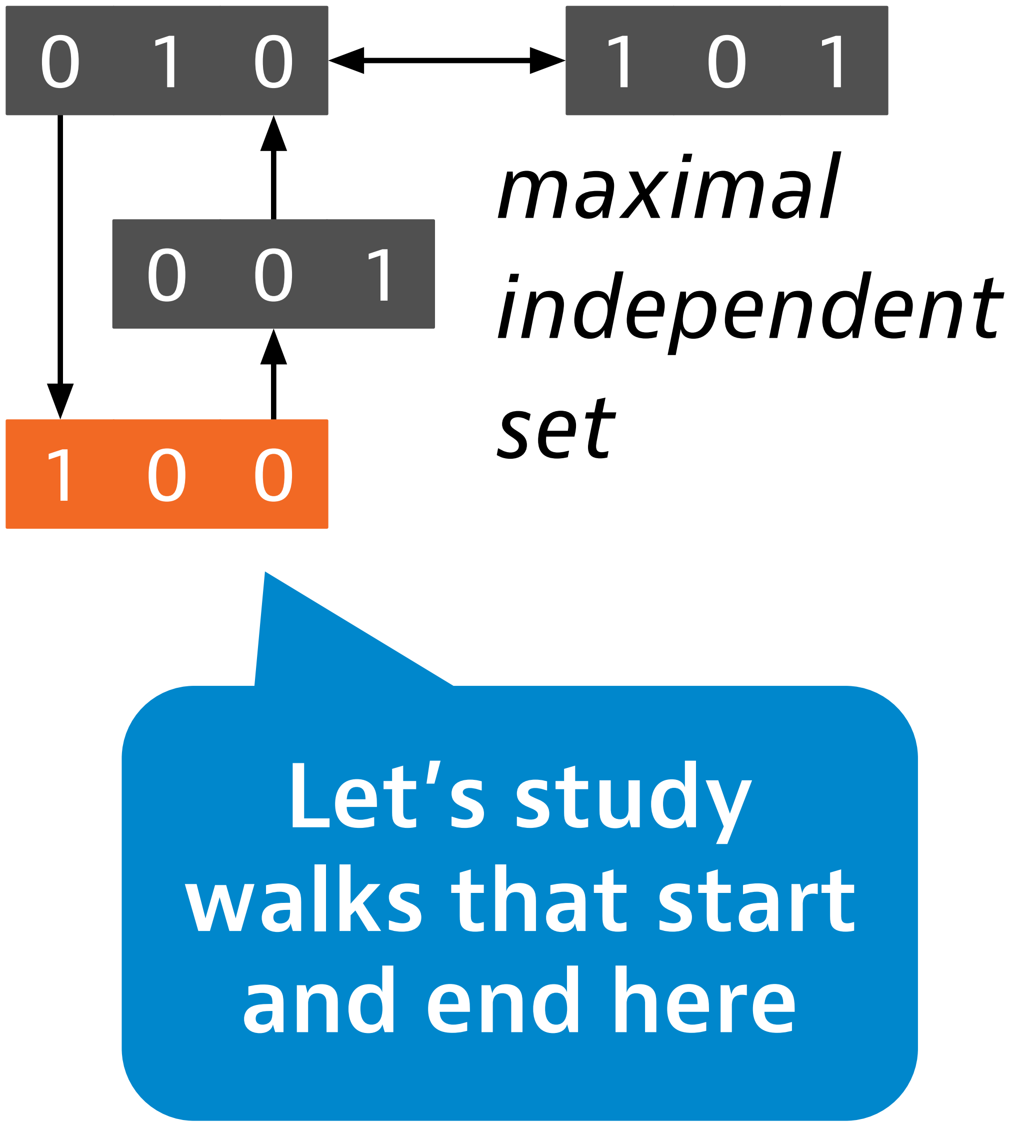
solvable in $O(1)$ rounds

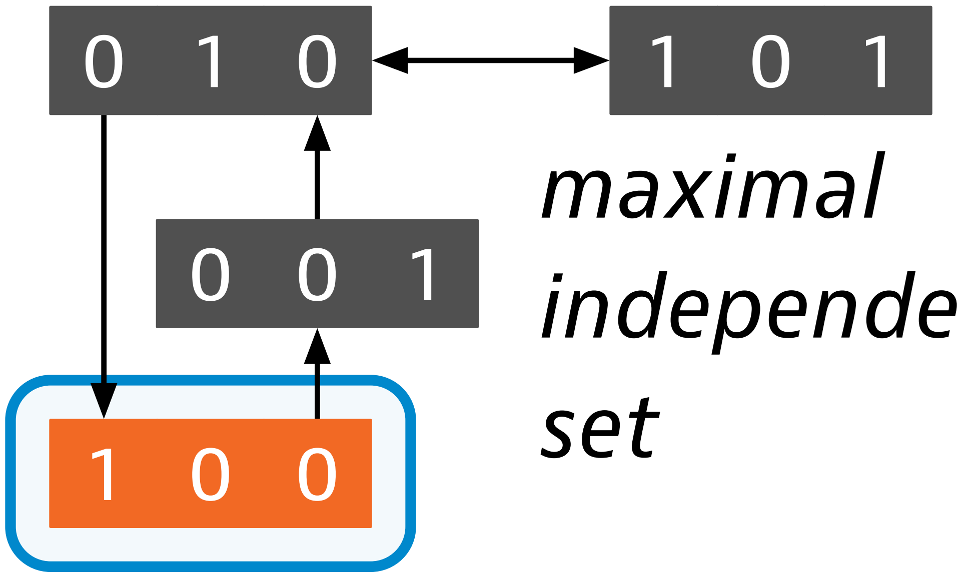
Proof: No self-loop
→ any solution breaks symmetry everywhere
→ can be used to find 3-coloring
→ not possible in $o(\log^* n)$ rounds



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

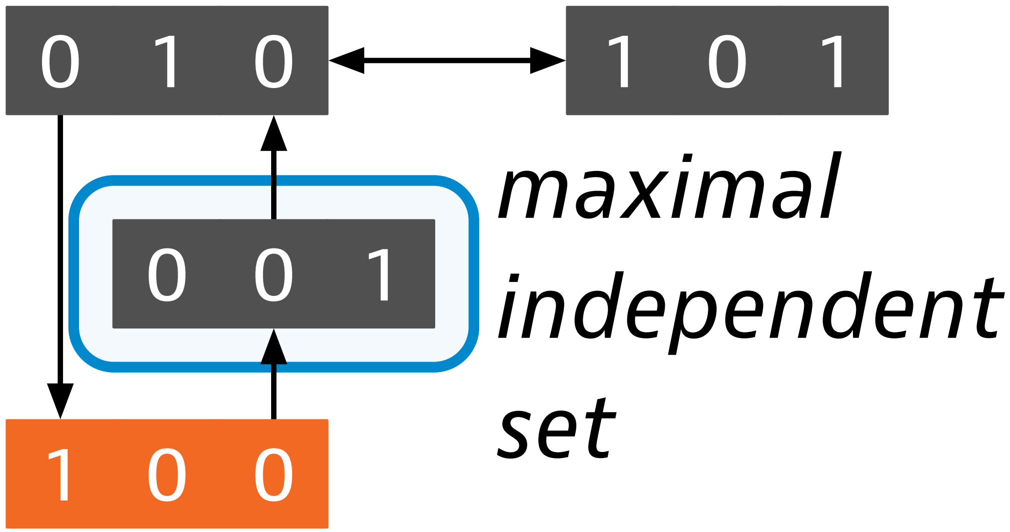

0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring


0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
Let's study walks that start and end here

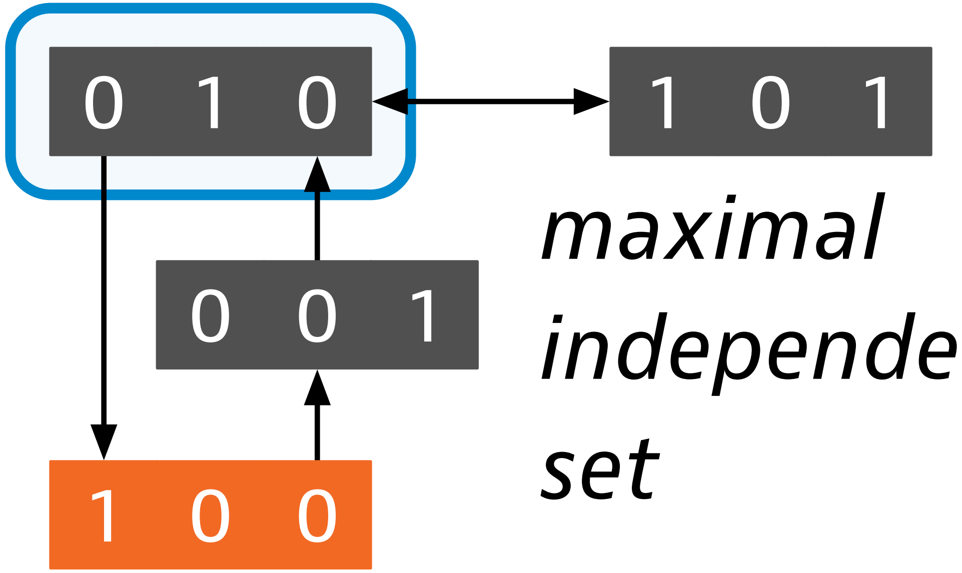


maximal independent set

0

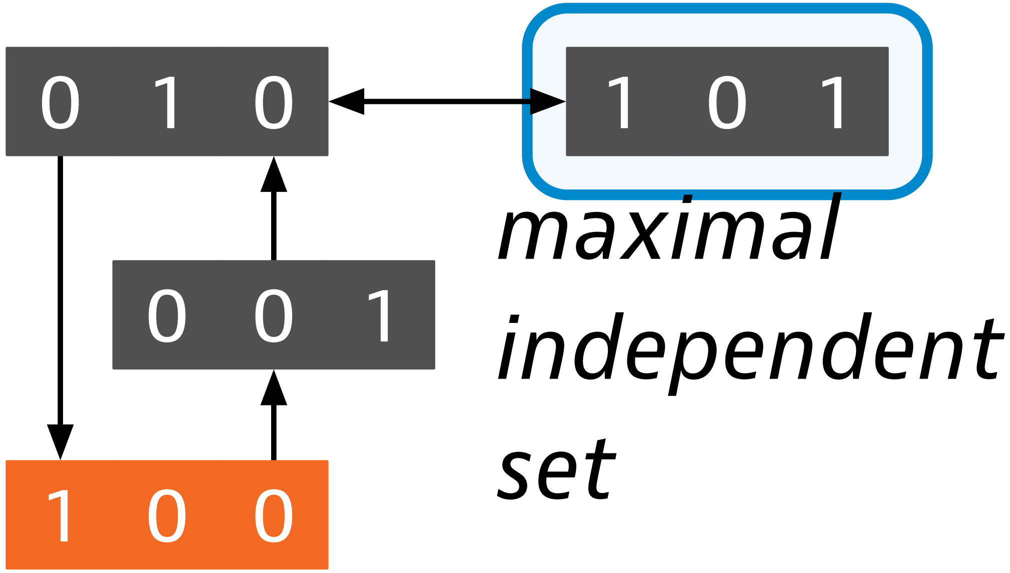


1

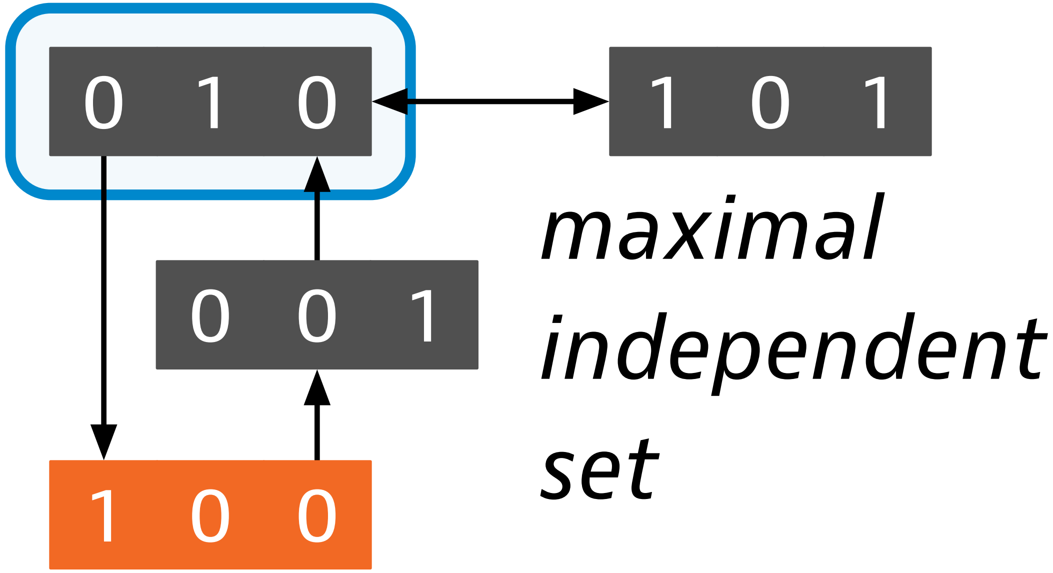


maximal independent set

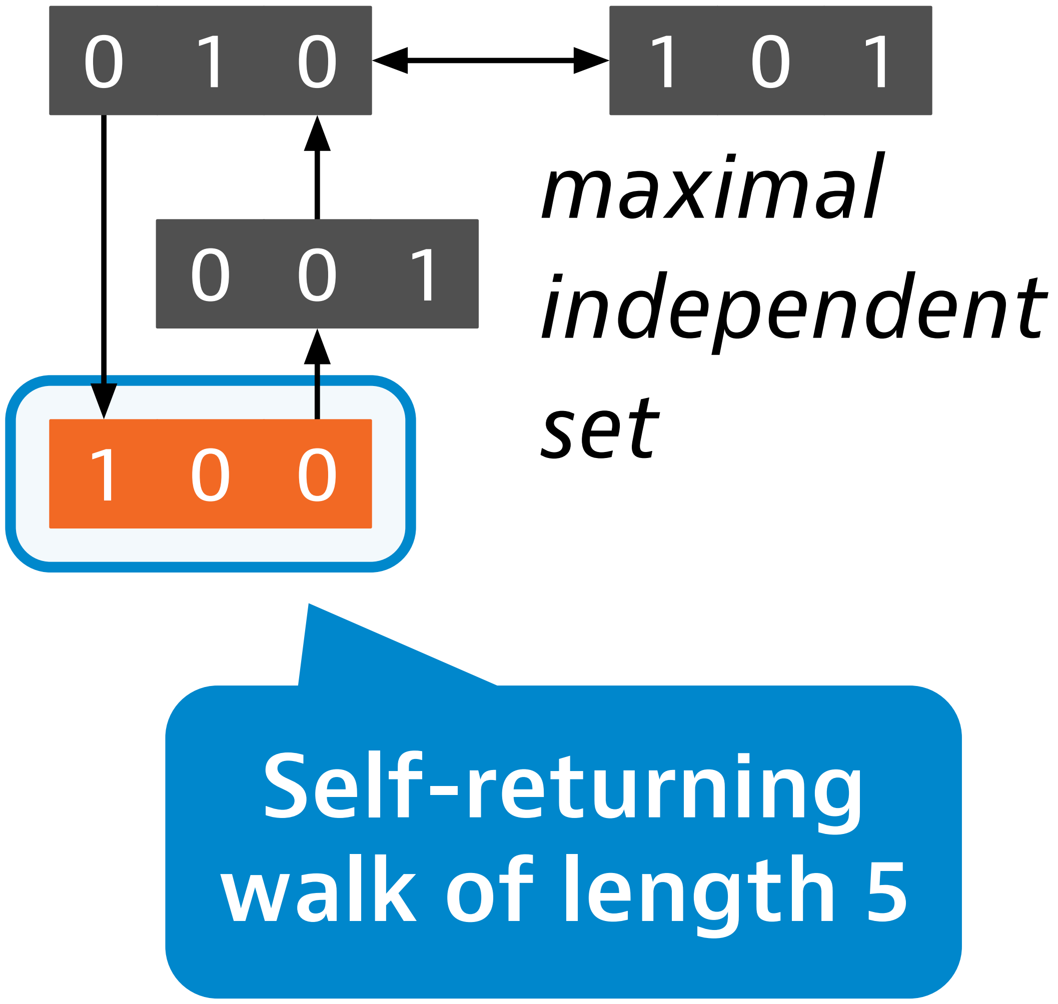
2



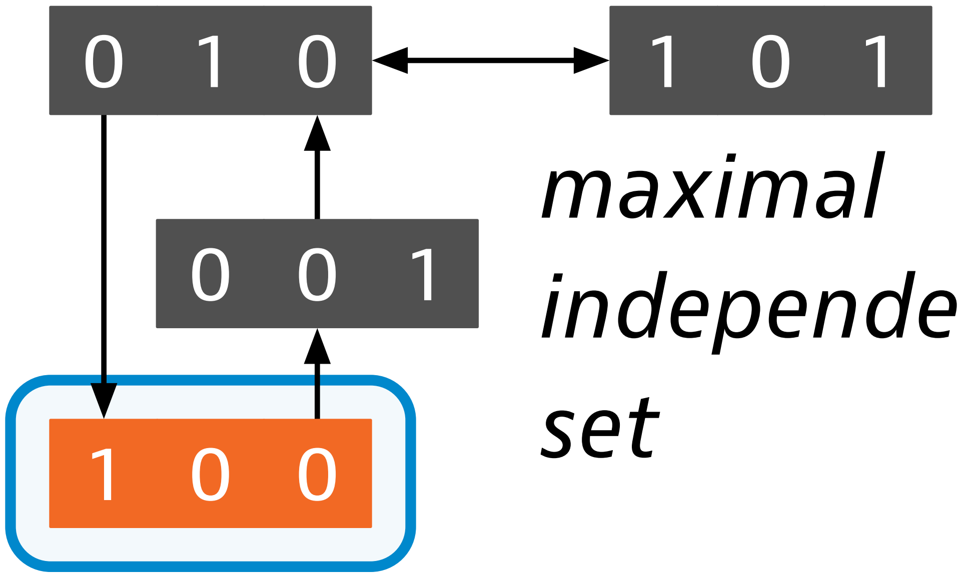
3



4

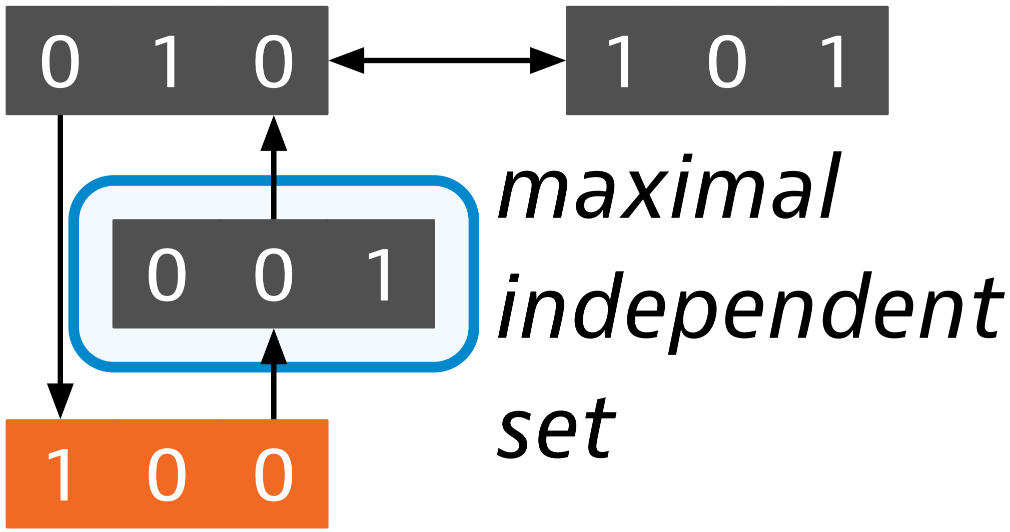

0 1 0
1 0 1
0 0 1
1 0 0
maximal independent set
Self-returning walk of length 5

5

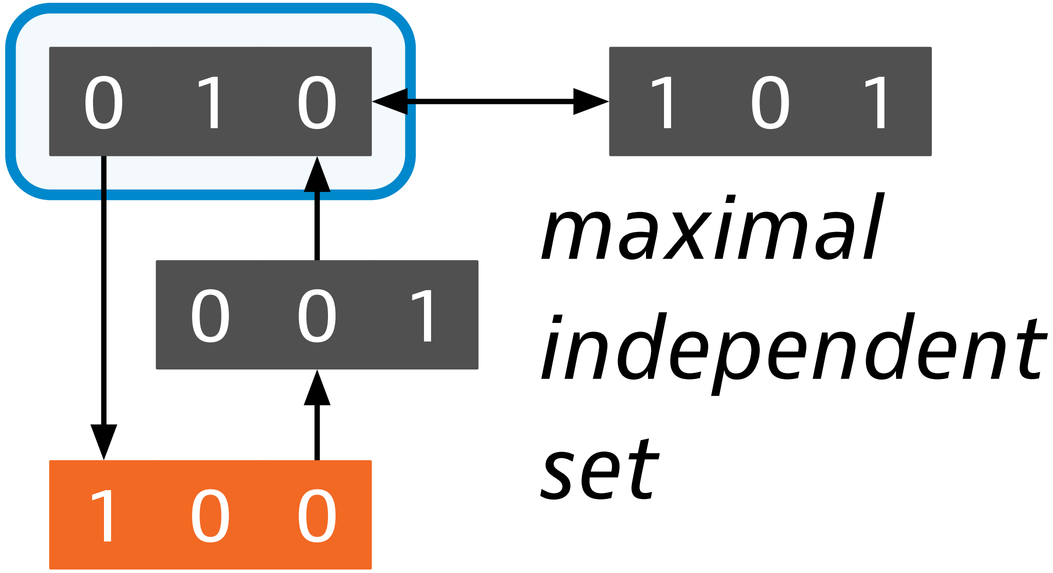


maximal independent set

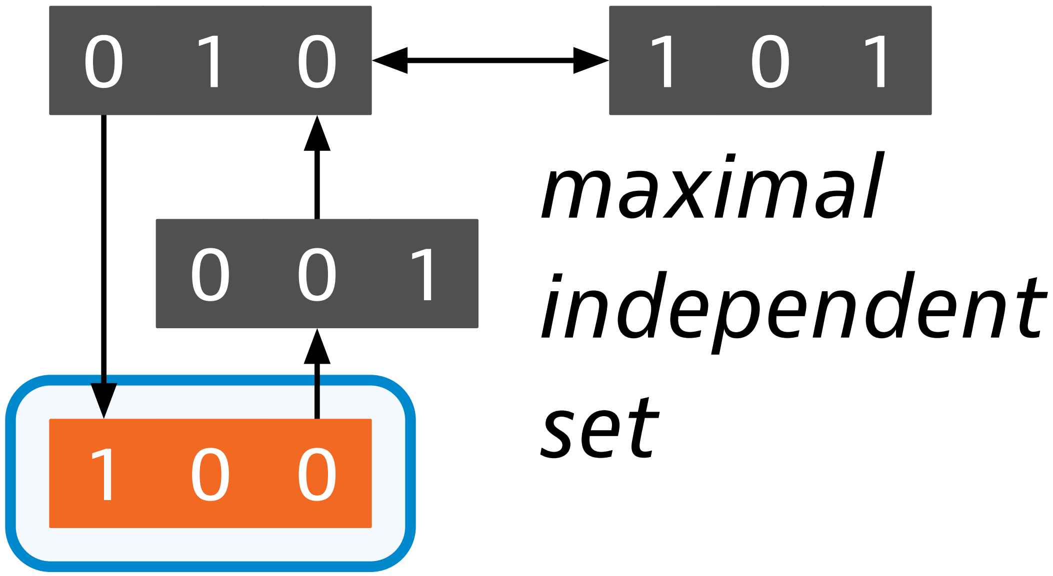
0



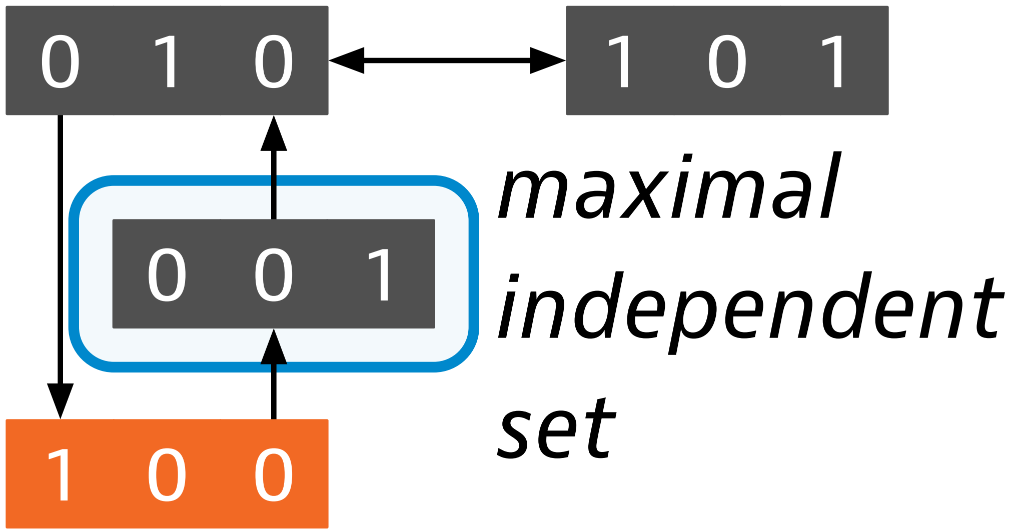
1



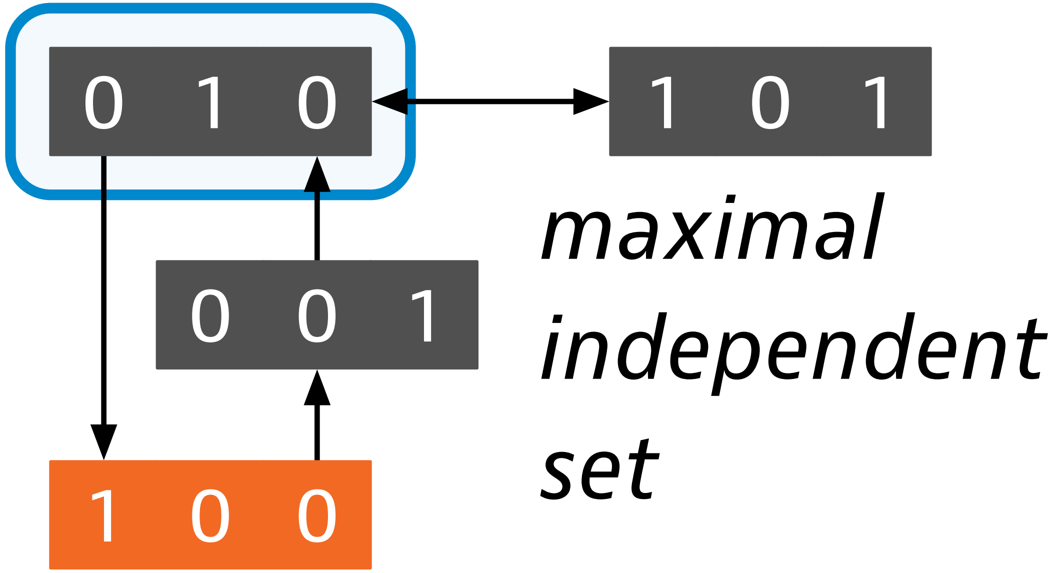
2



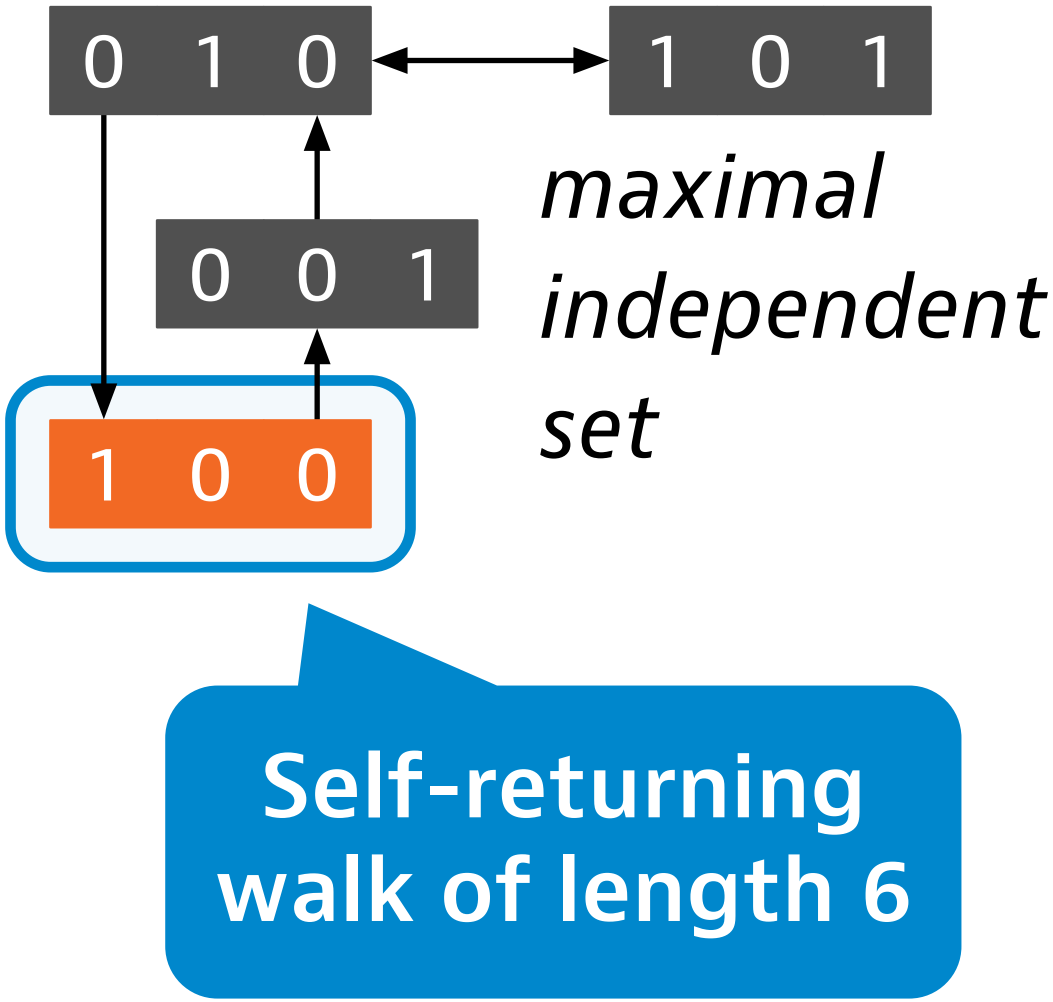
3



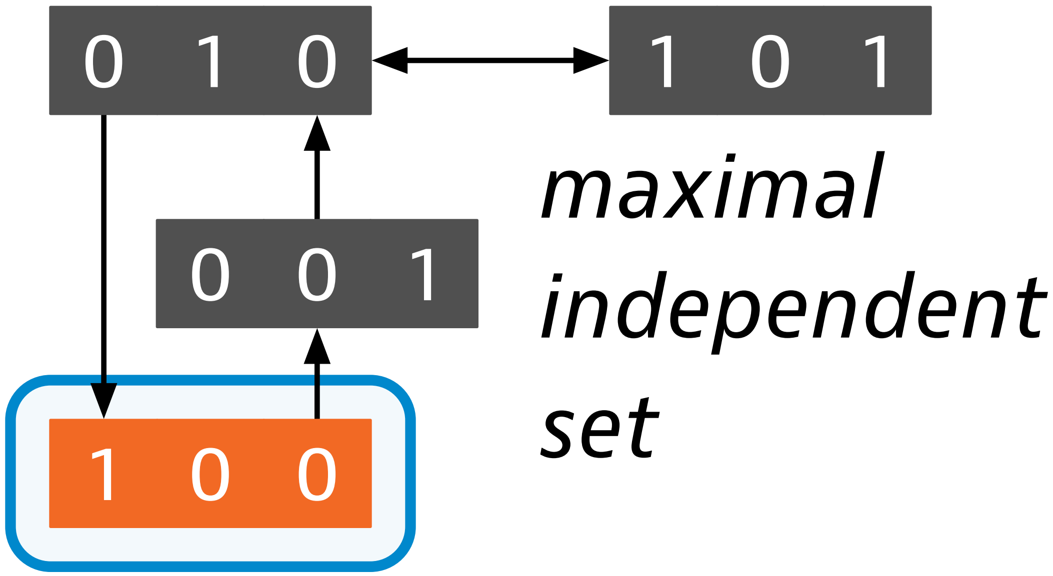
4



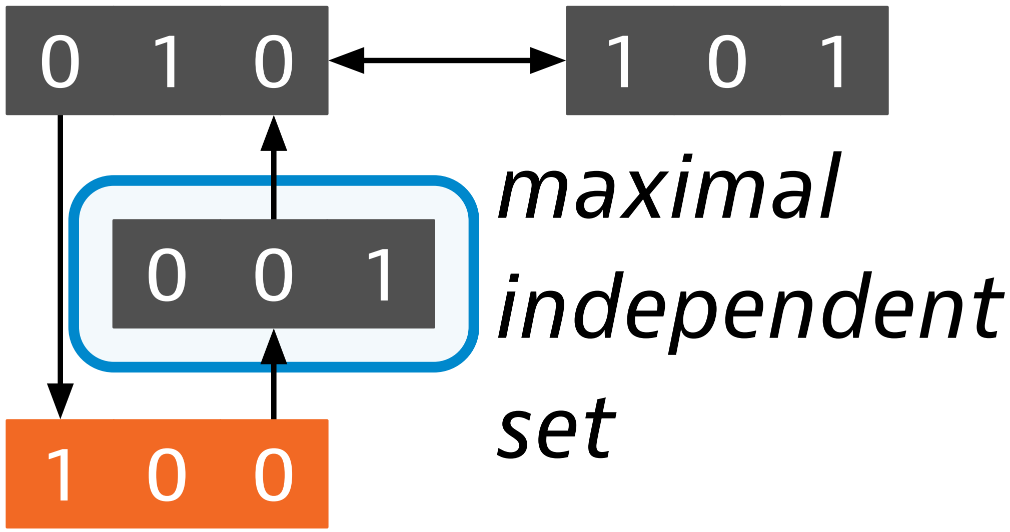
5


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

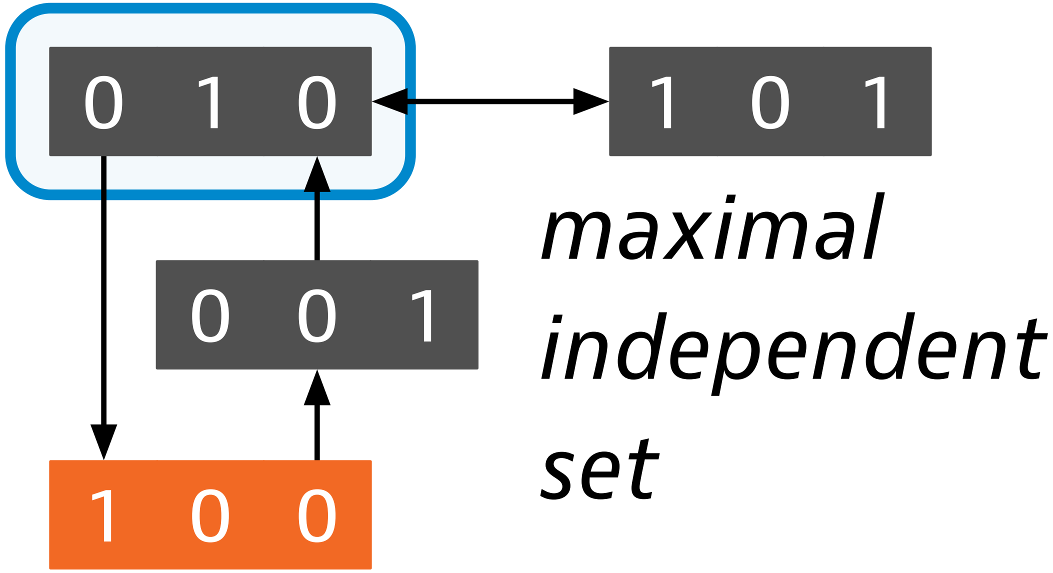
6



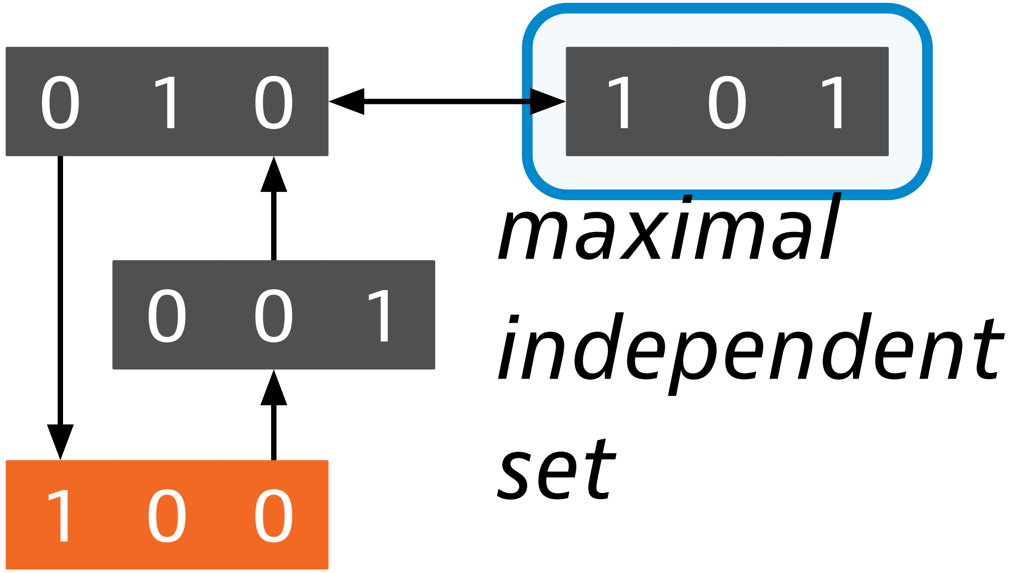
0



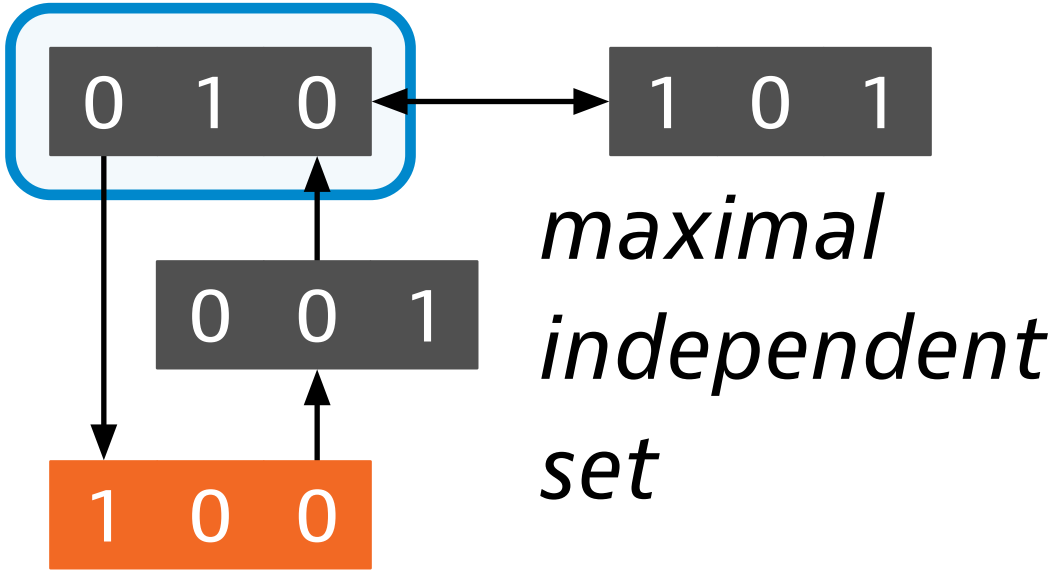
1



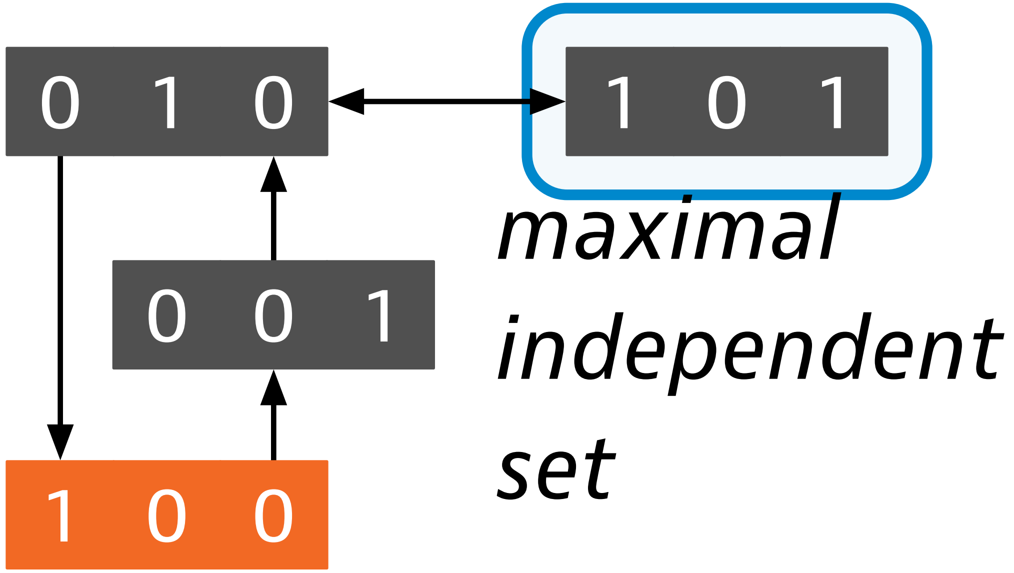
2



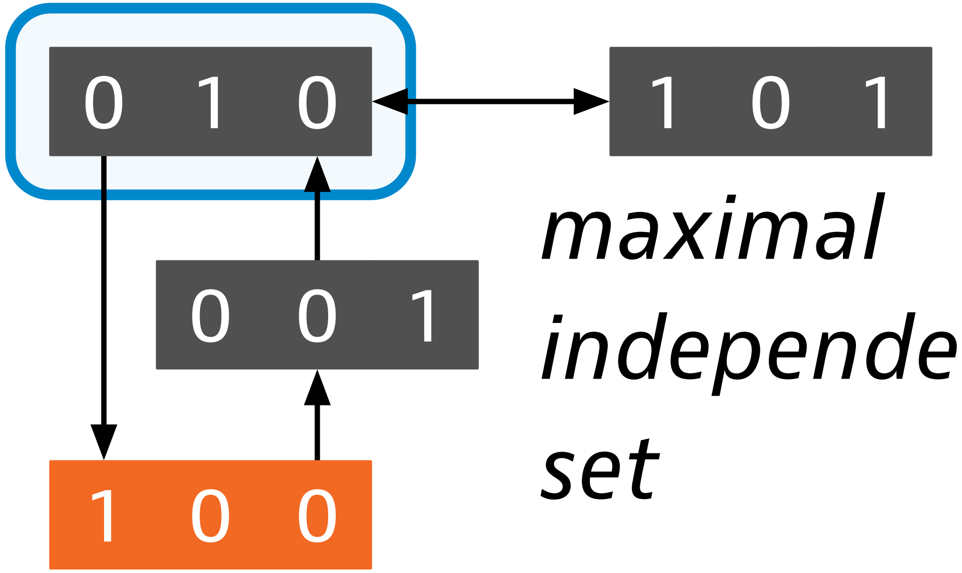
3



4

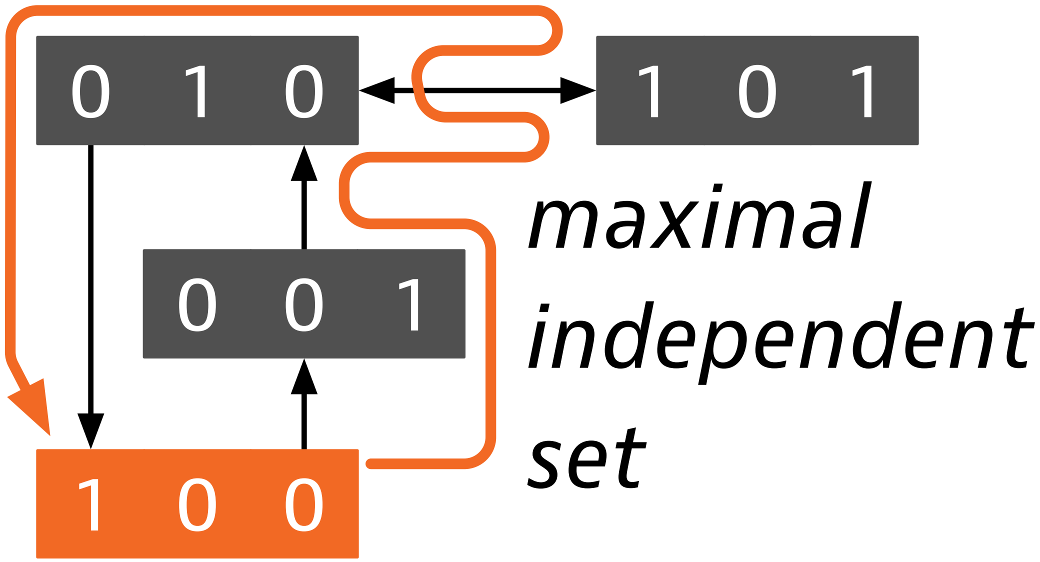


5



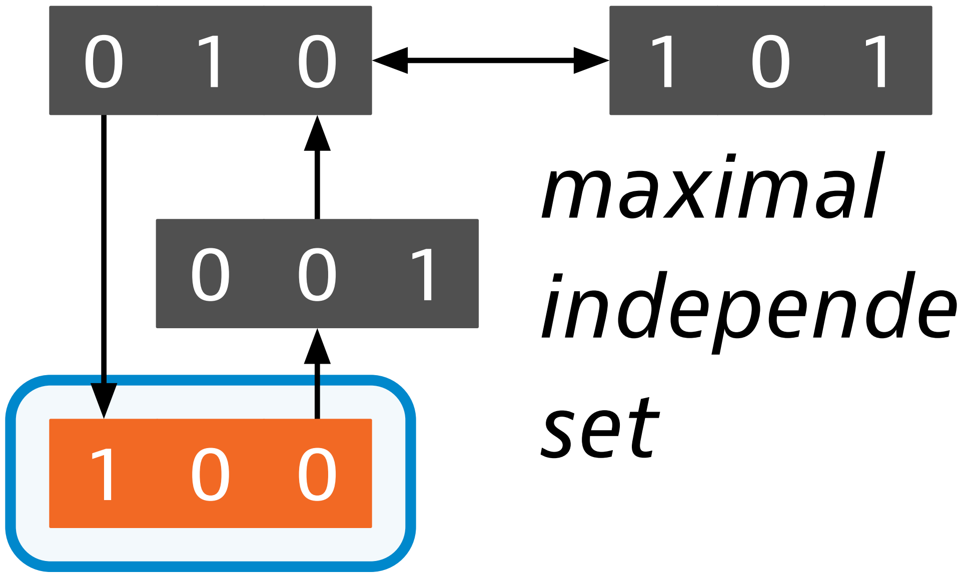
maximal independent set

6



Self-returning walk of length 7

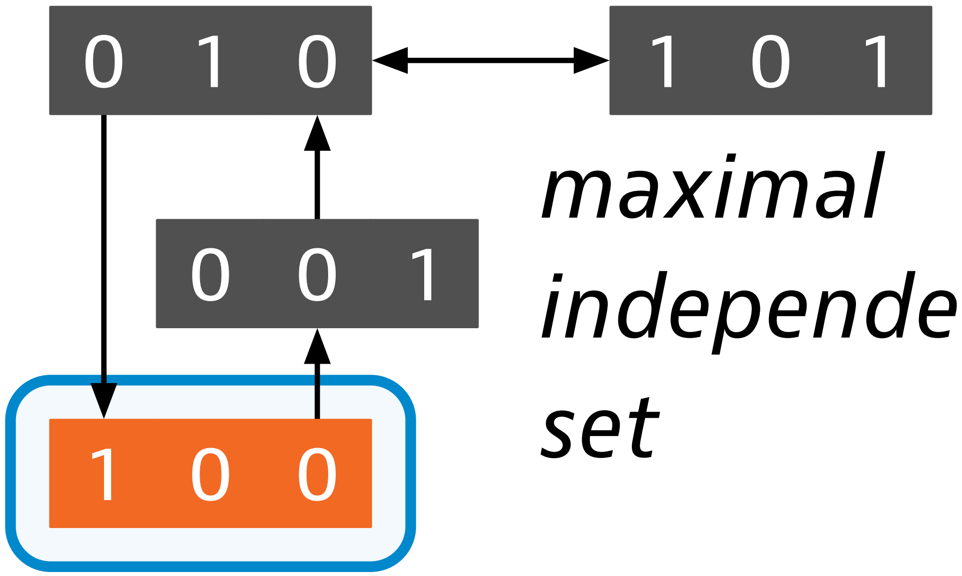
7



maximal independent set

8?

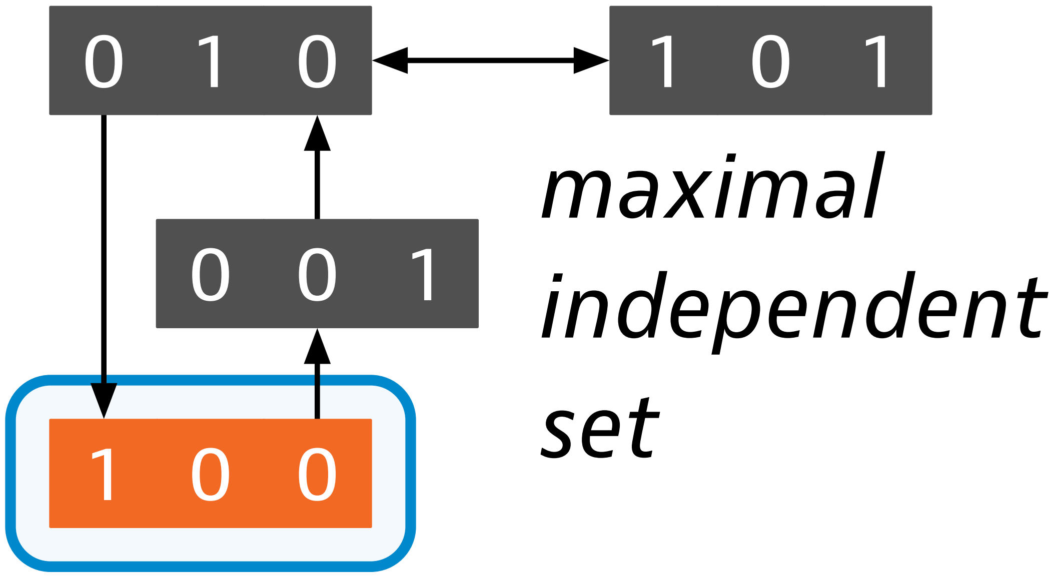
Can you find a self-returning walk of length 8?



maximal independent set

9?

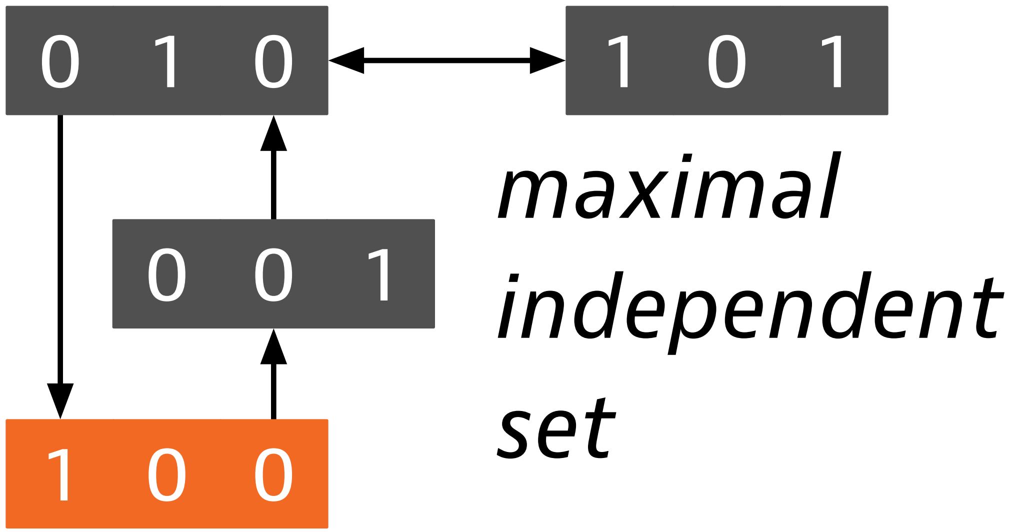
Can you find a self-returning walk of length 9?



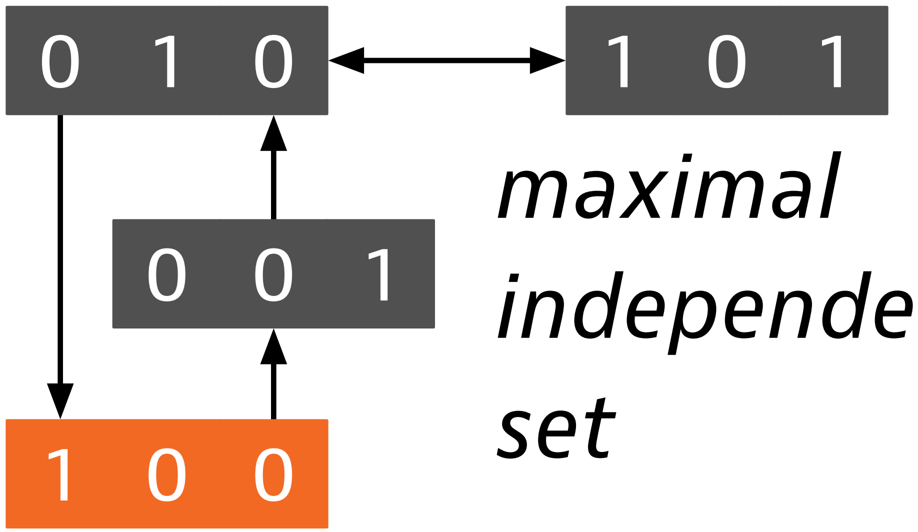
Self-returning walk of length *k*

$$k = 5, 6, 7, 8, 9, \ldots$$

"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



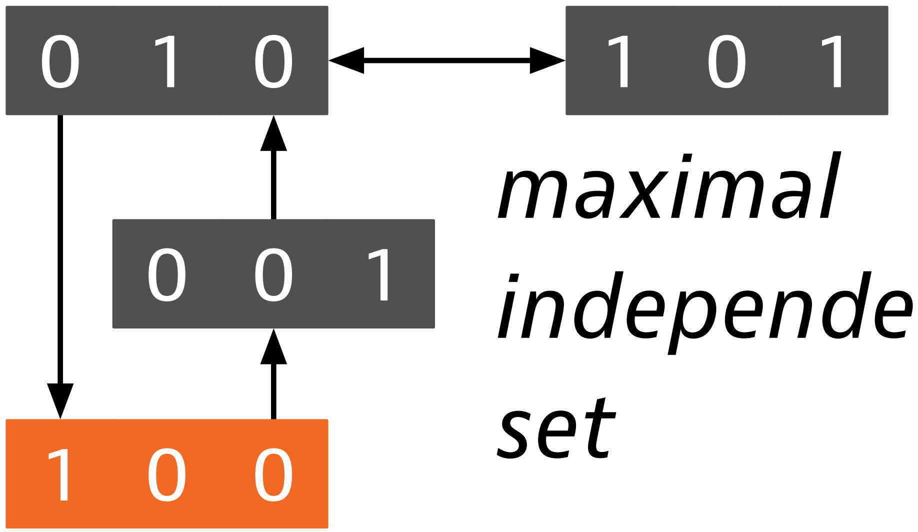
maximal independent set

Decidable in polynomial time (how?)

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

↓

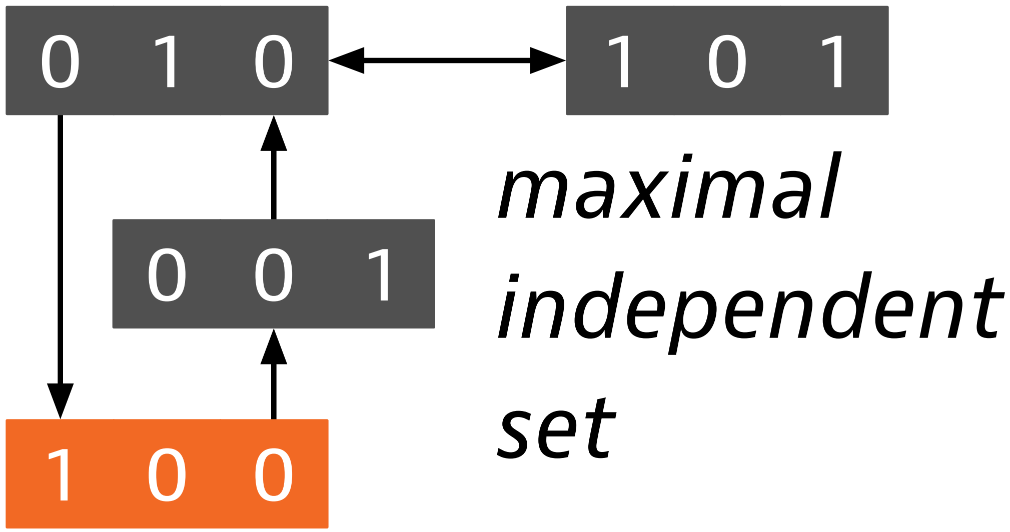
solvable in
$O(\log^* n)$ rounds



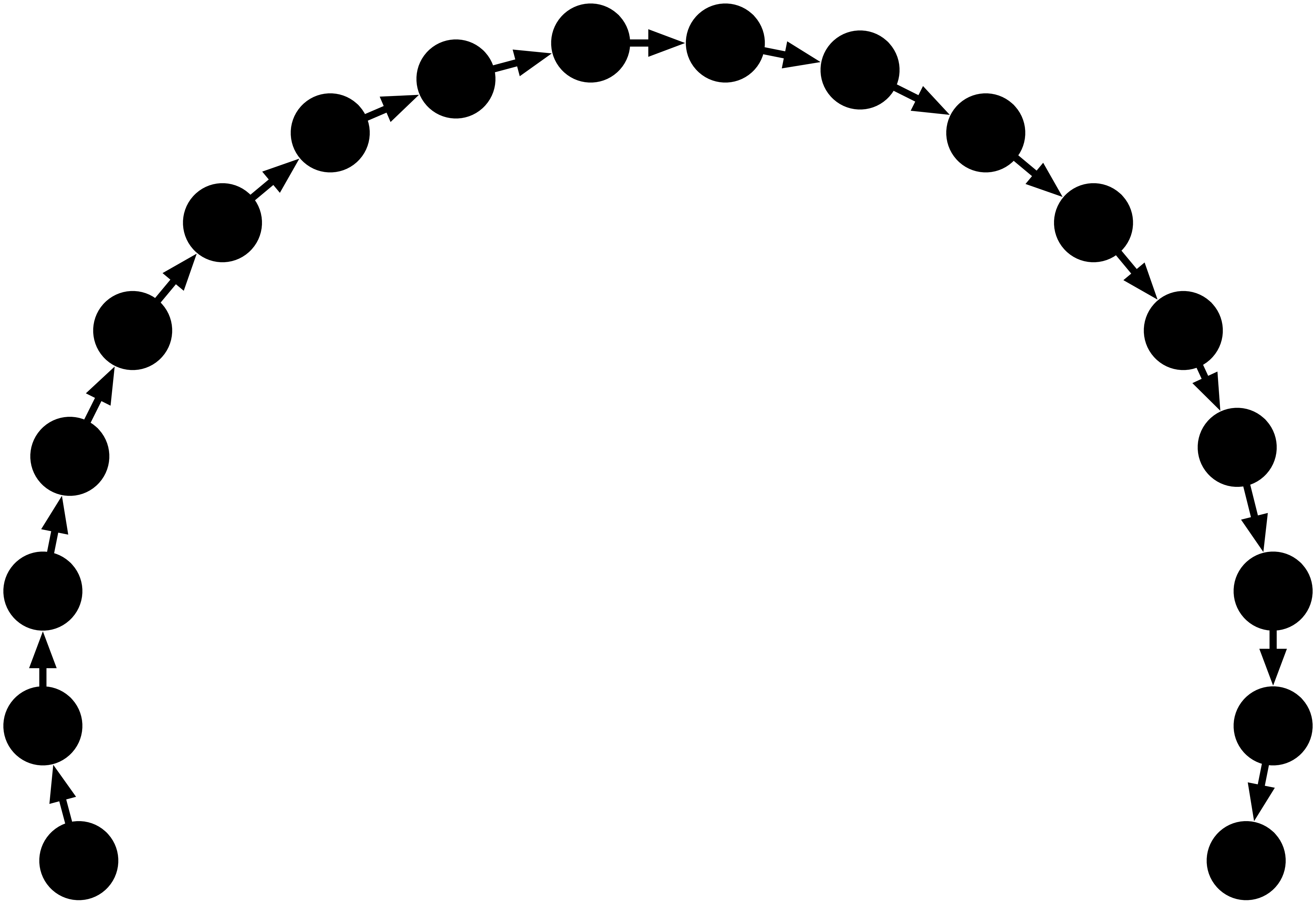
maximal independent set

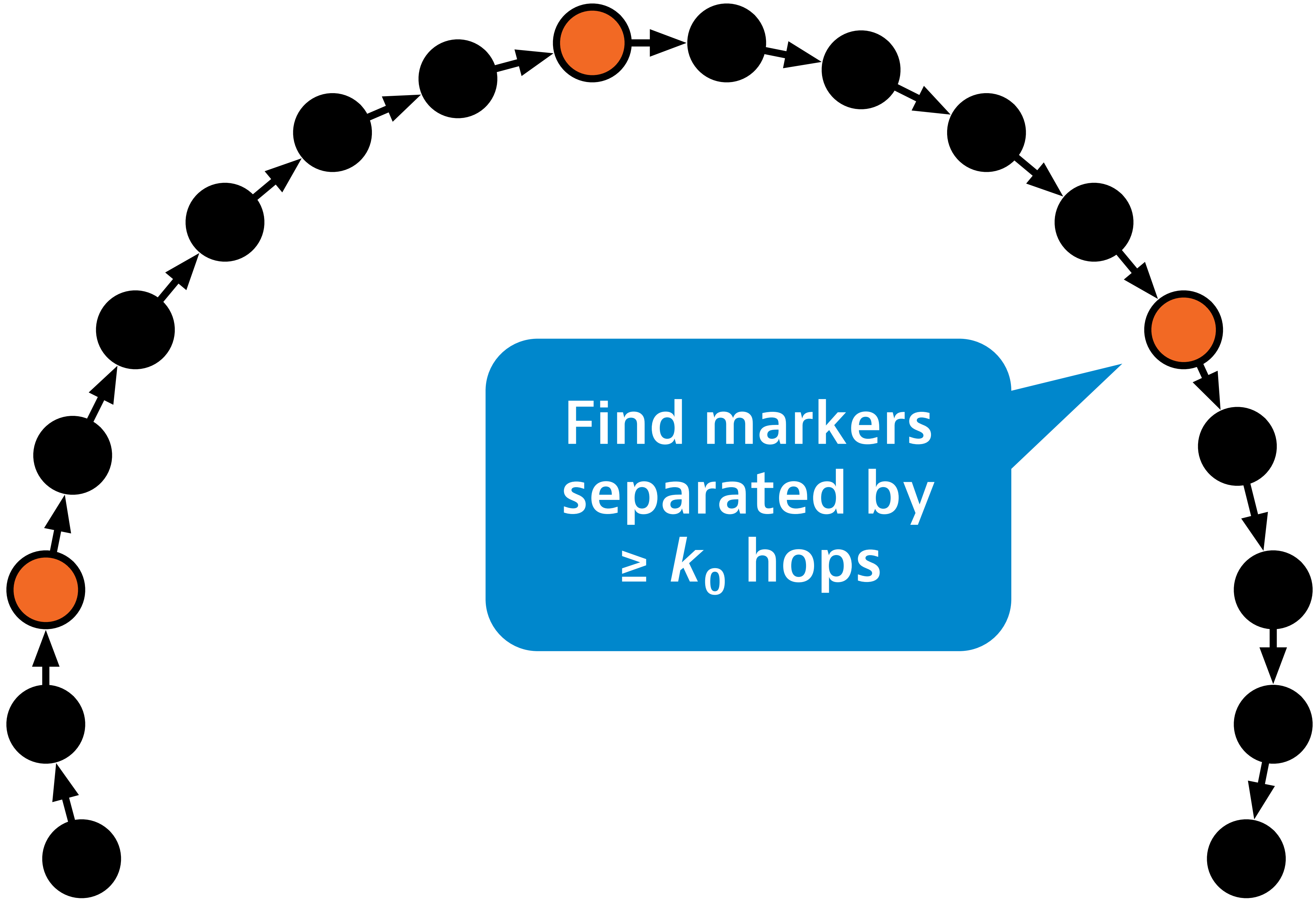
“Flexible”:
for all $k \geq k_0$
there is a self-returning walk
of length k

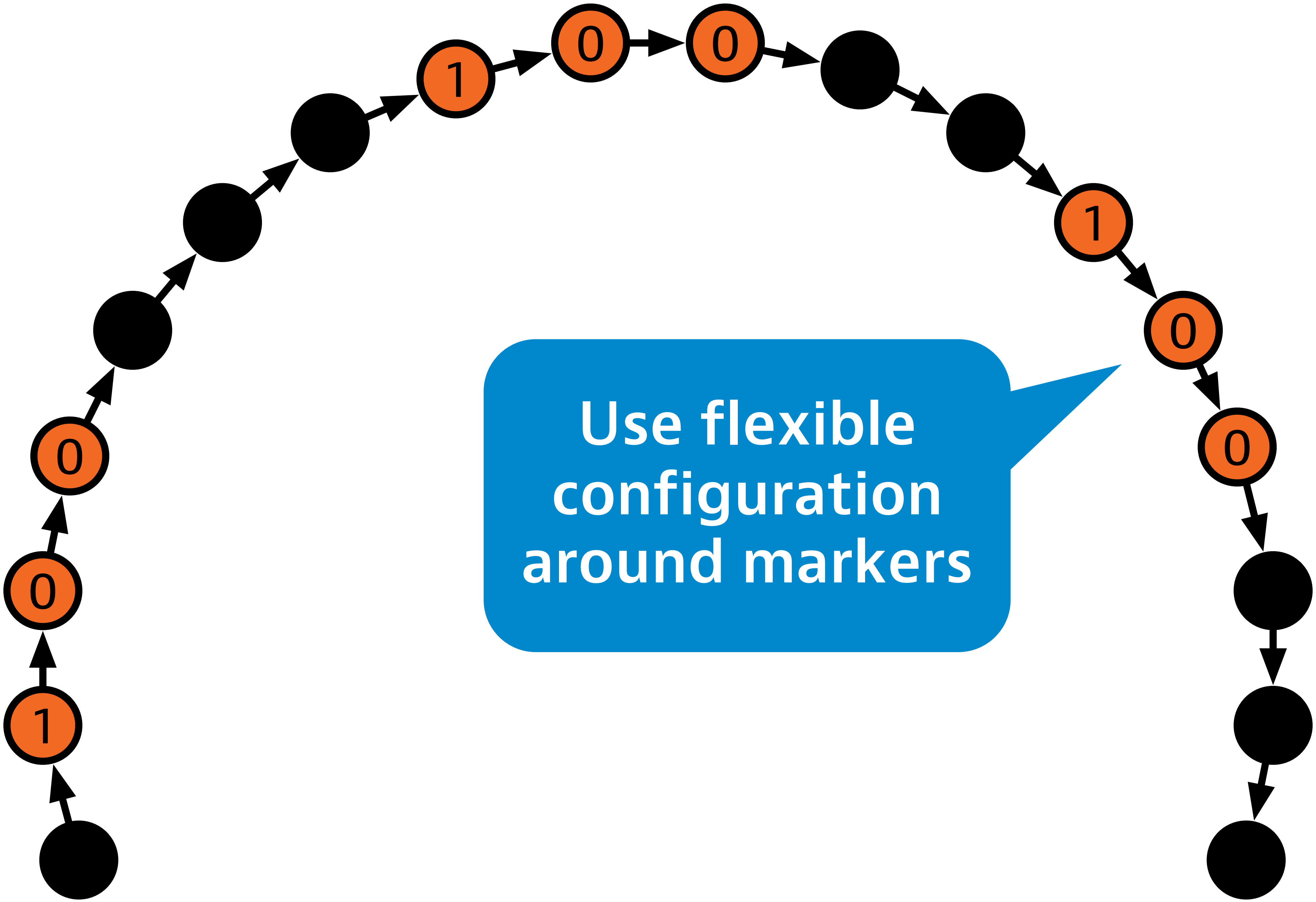
solvable in
$O(\log^* n)$ rounds

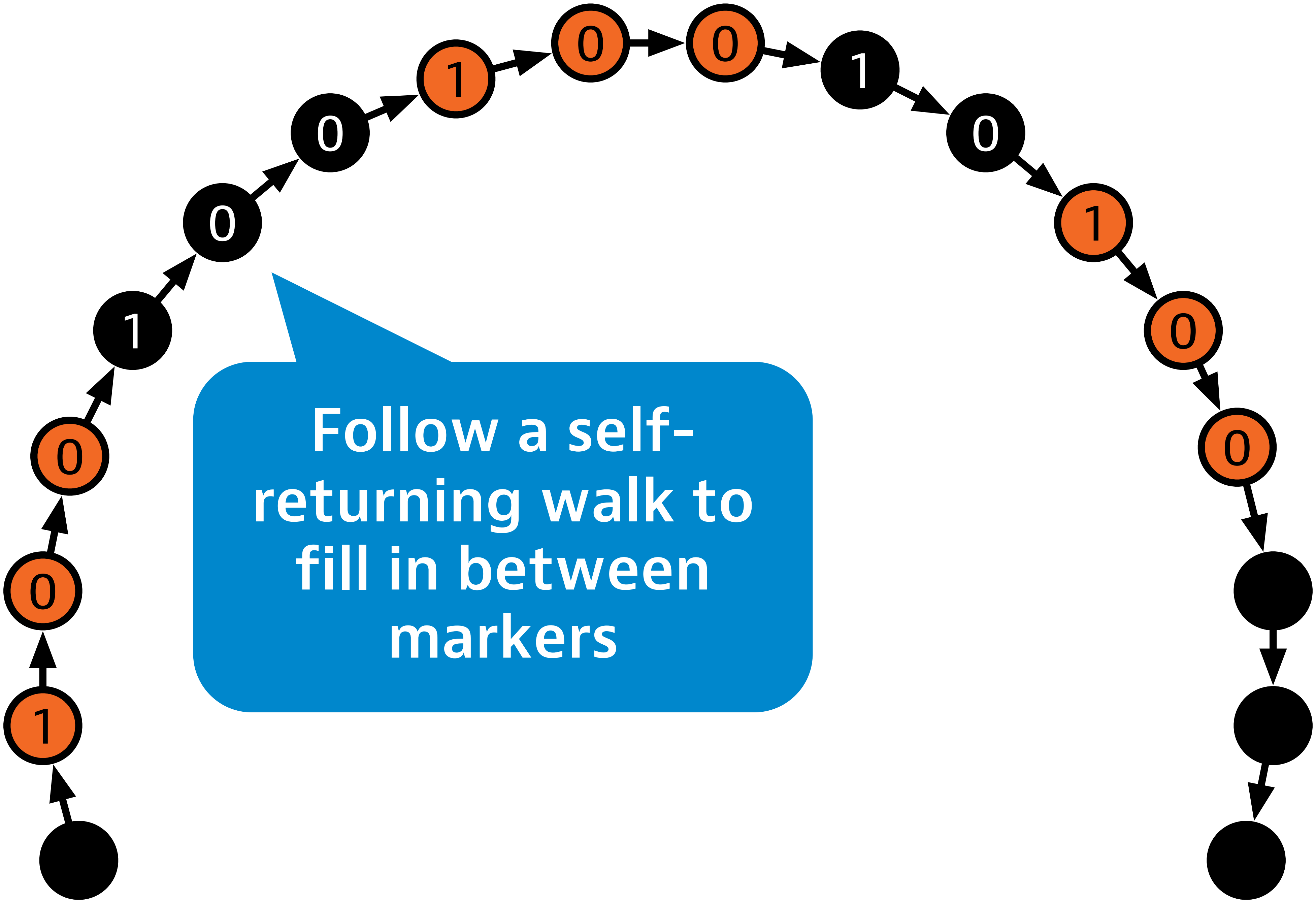


Algorithm: ???



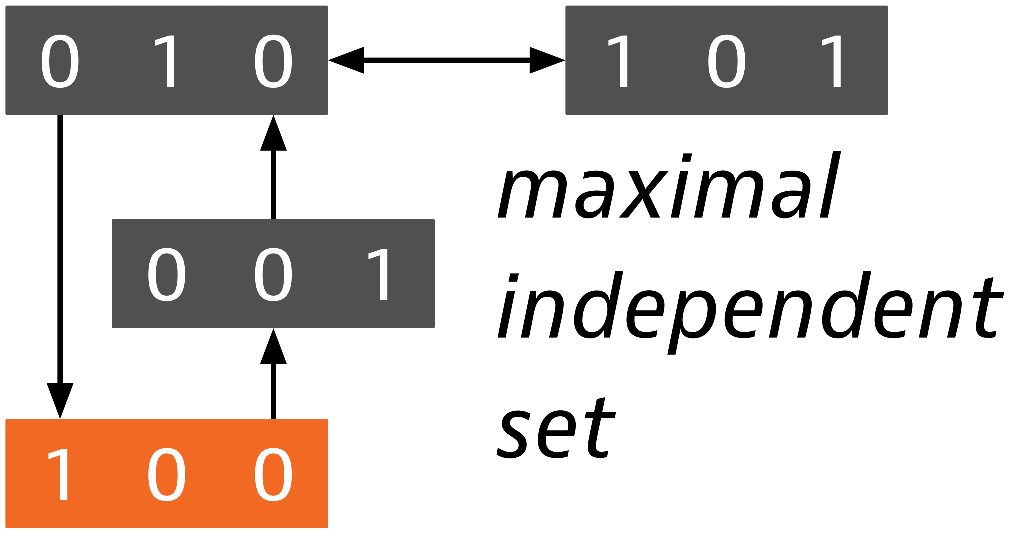

Find markers
separated by
≥ k_0 hops


1
0
0
1
0
0
1
0
0
Use flexible configuration around markers


Follow a self-returning walk to fill in between markers

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

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

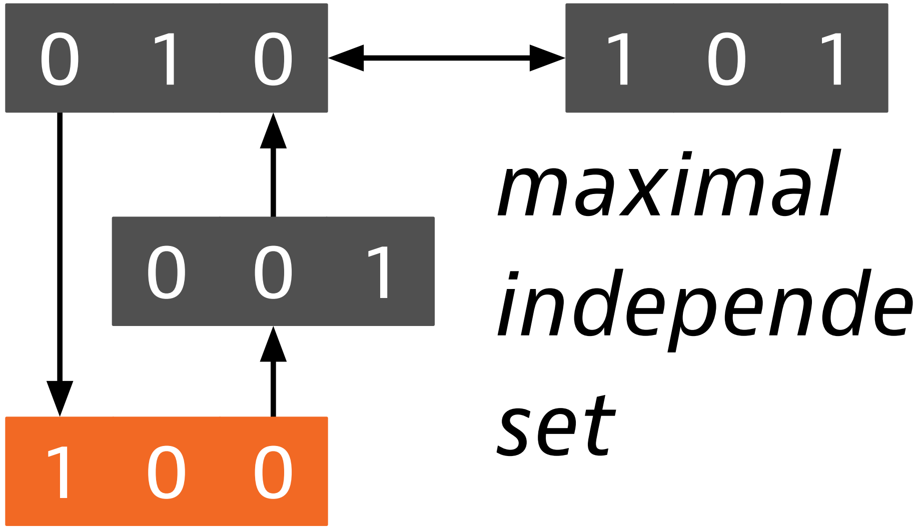


Algorithm:

- 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

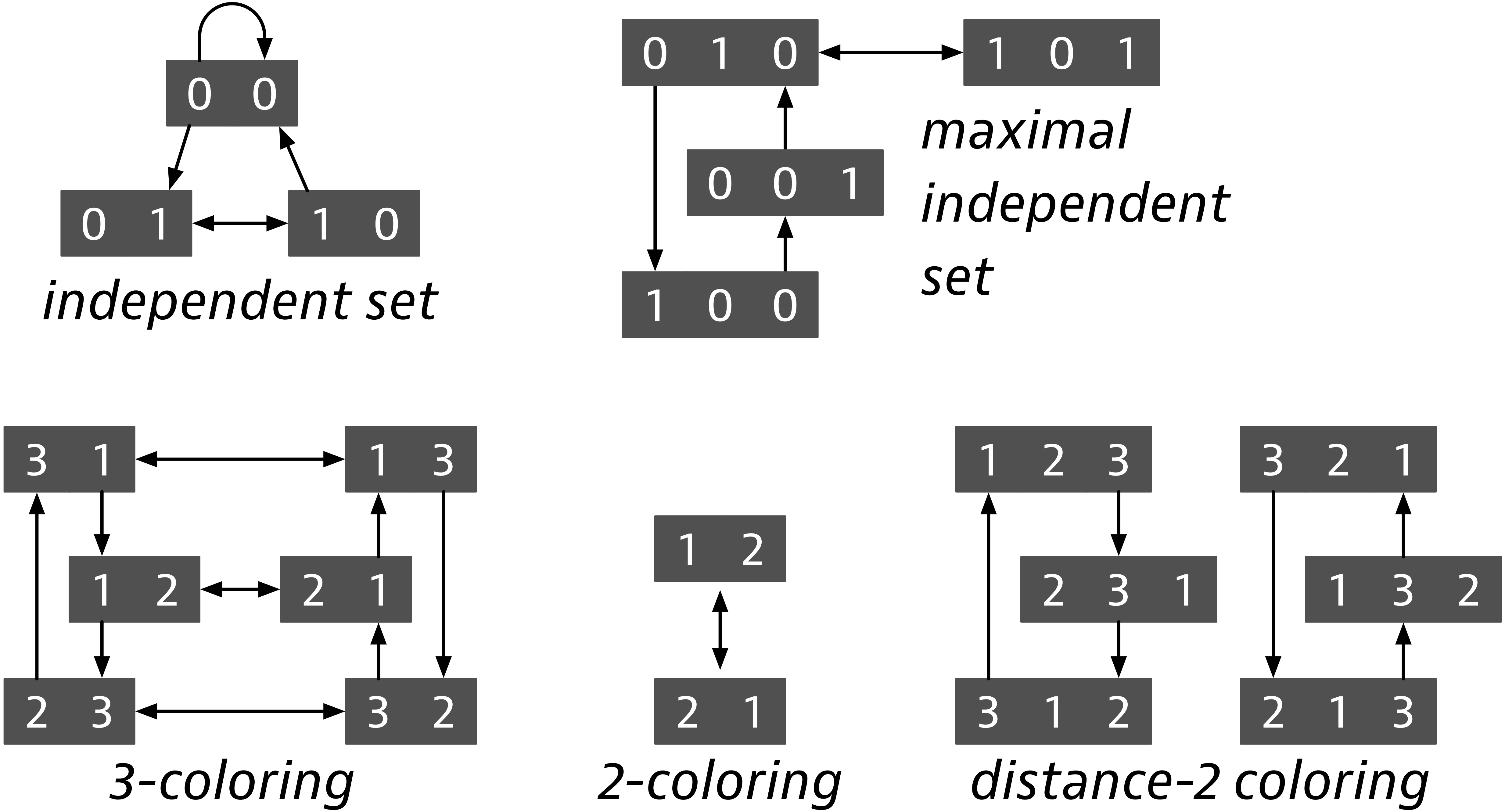
"Flexible":
for all $k \geq k_0$ there is a self-returning walk of length k

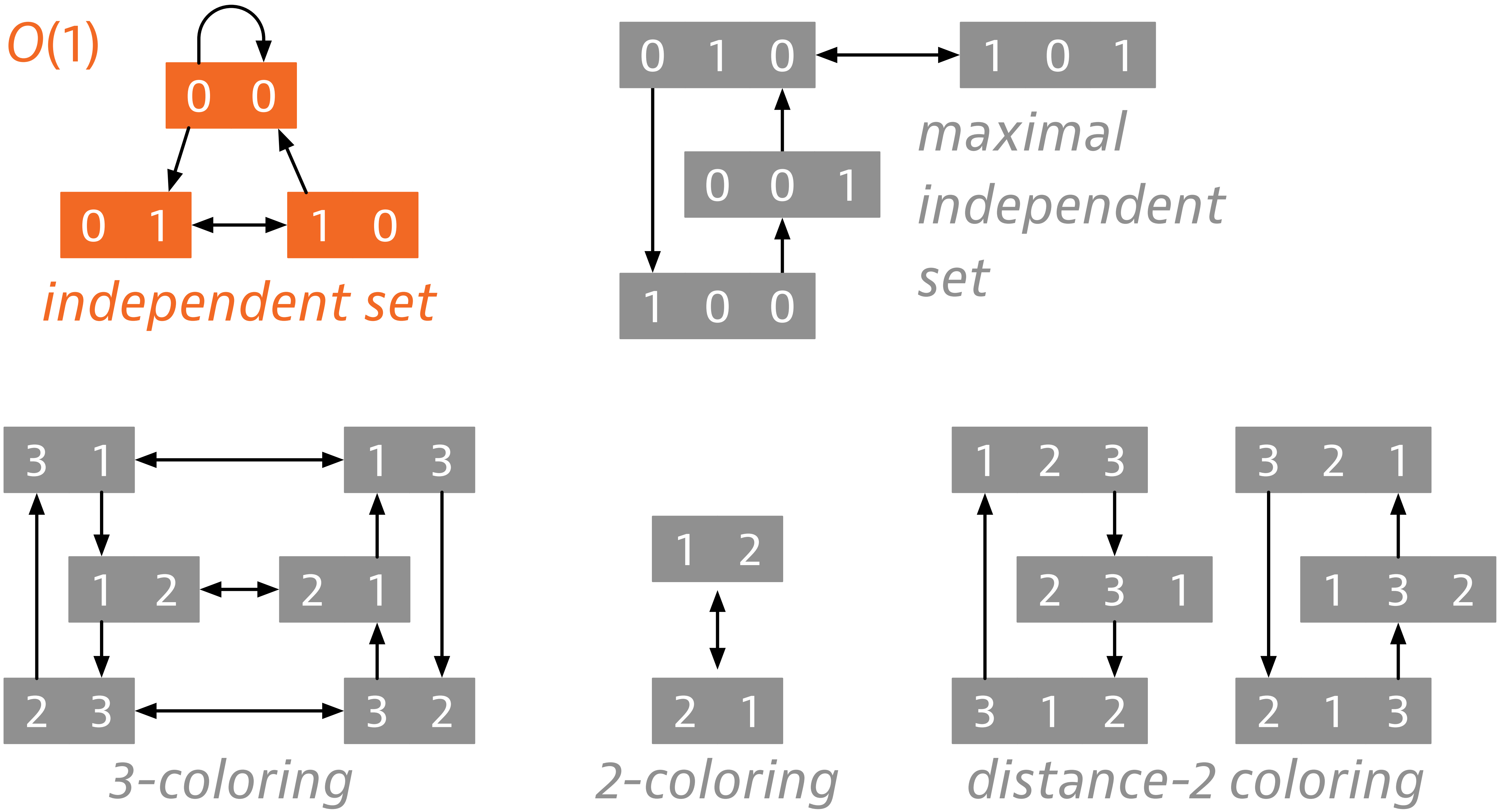
solvable in $O(\log^* n)$ rounds

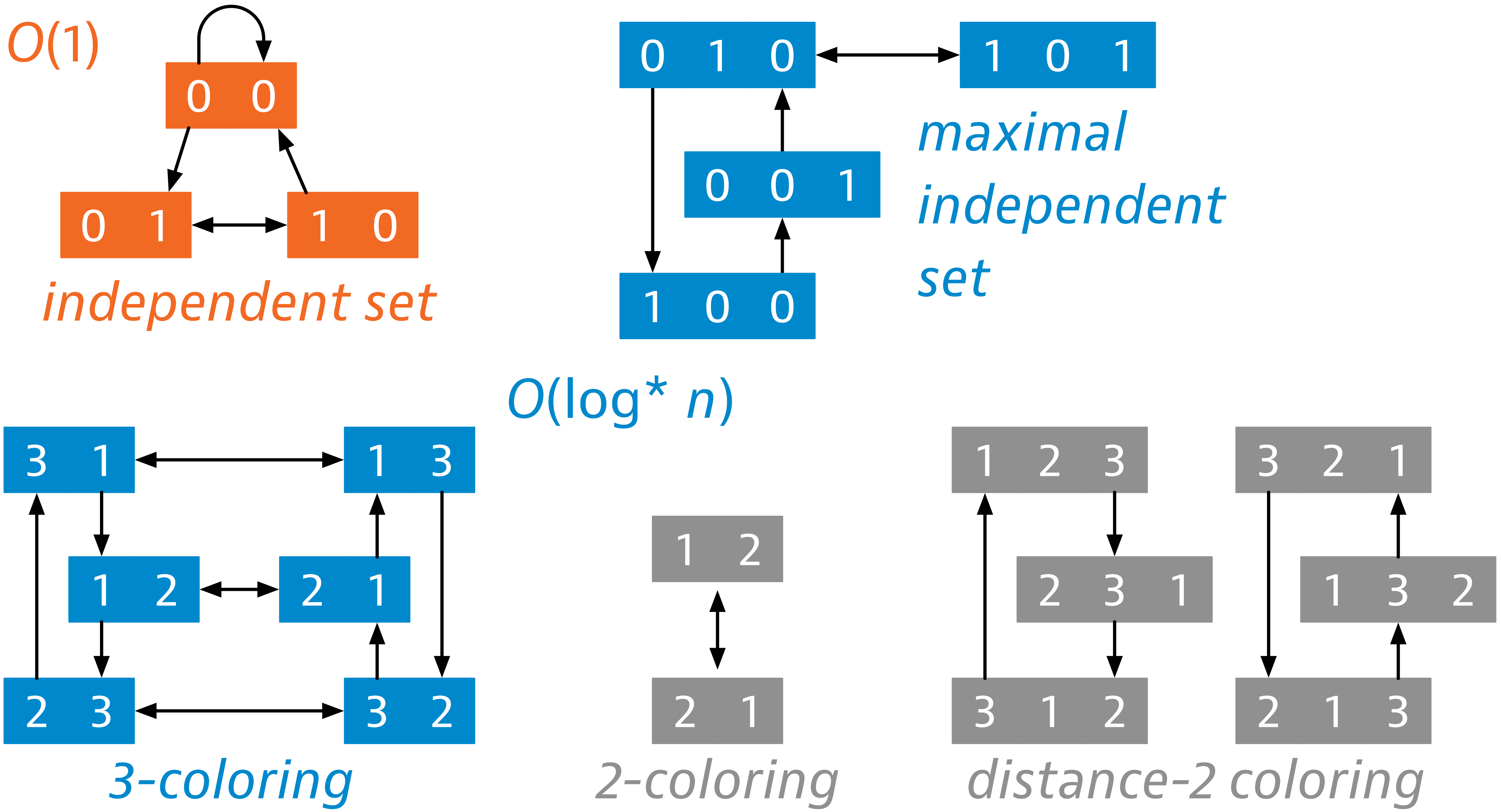


maximal independent set

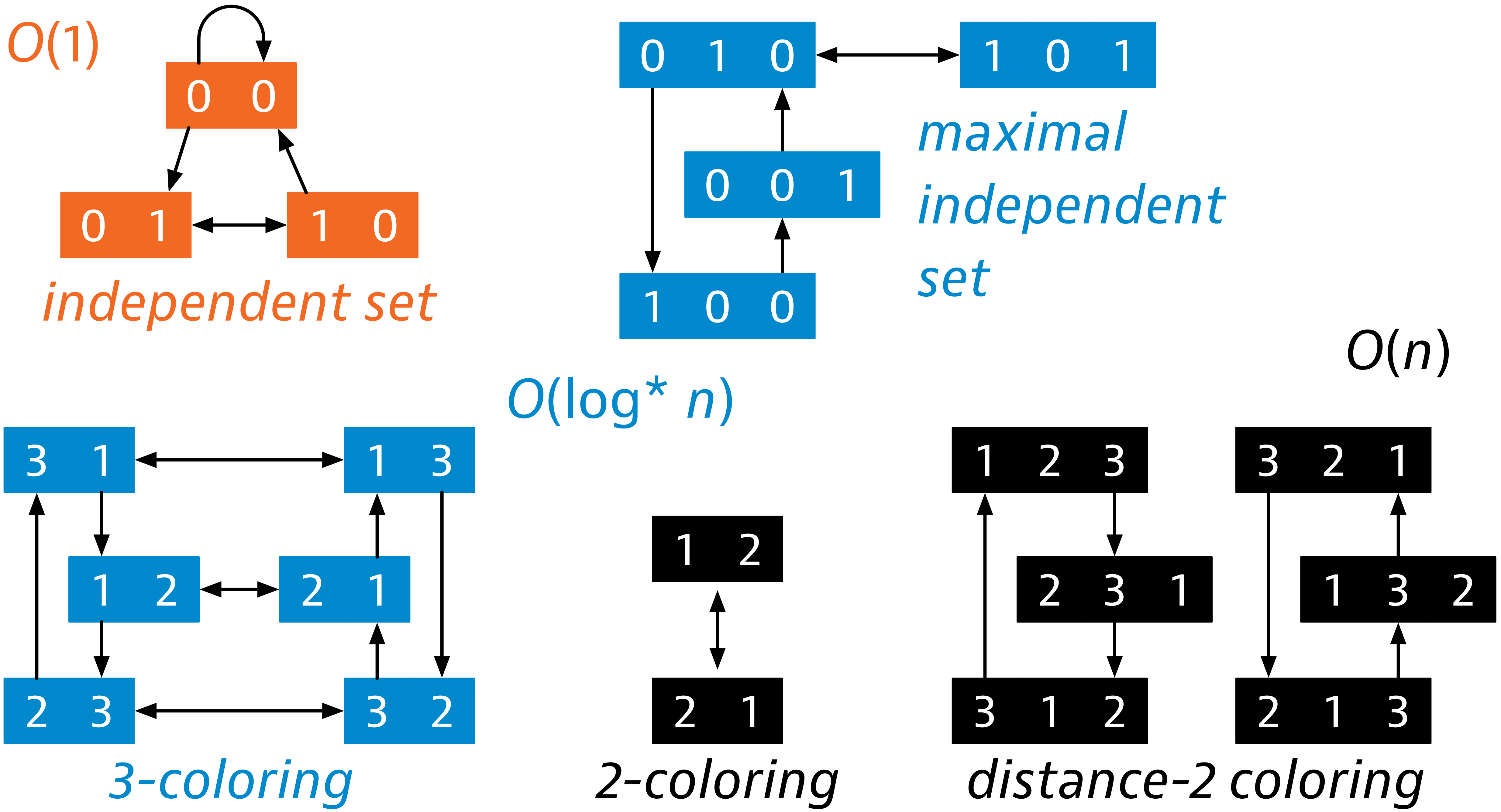
Proof: Not flexible → must use the same non-flexible configuration at least twice far from each other; not compatible for all distances
→ global coordination needed
→ not possible in $o(n)$ rounds


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
2 3 1
3 1 2
3 2 1
1 3 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

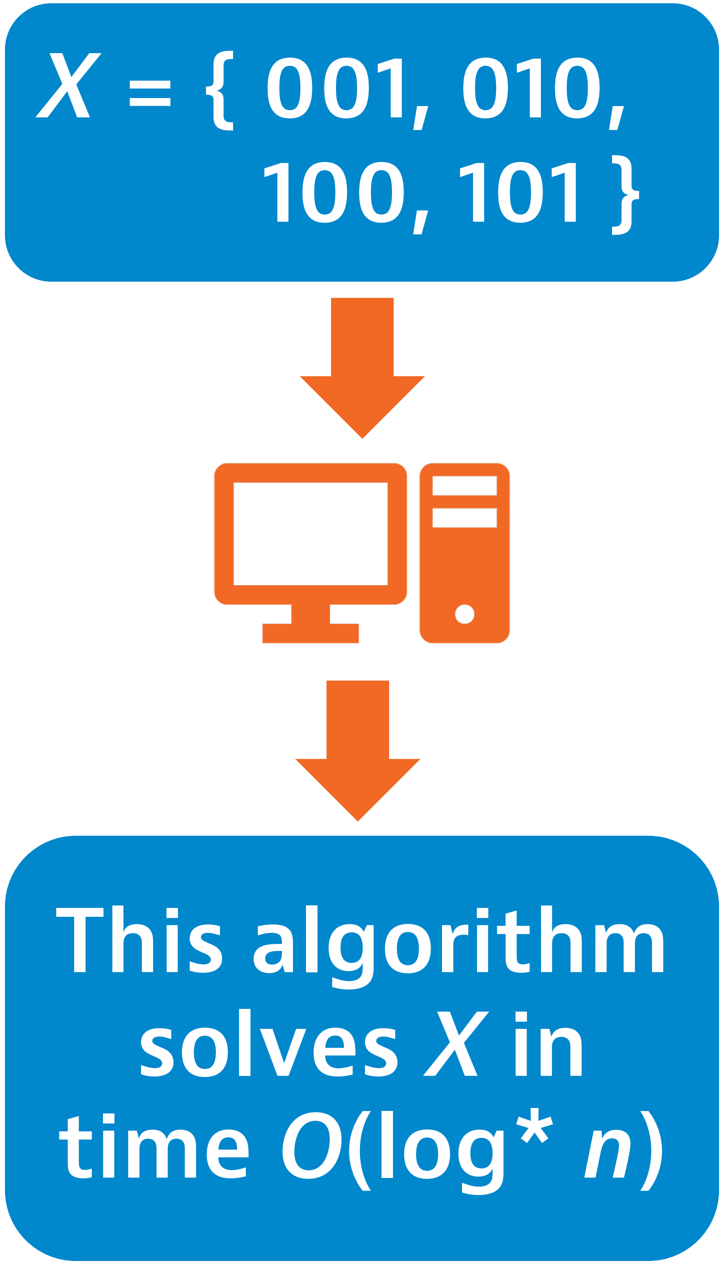


O(1)
0 0
0 1
1 0
independent set
0 1 0
1 0 1
0 0 1
1 0 0
maximal
independent
set
O(log* n)
3 1
1 3
1 2
2 1
2 3
3 2
3-coloring
1 2
2 1
2-coloring
1 2 3
3 2 1
2 3 1
1 3 2
3 1 2
2 1 3
distance-2 coloring


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

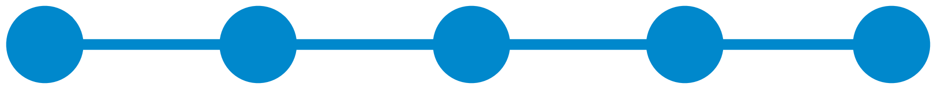
Fully automatic

- 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*

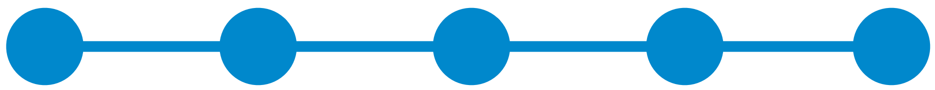


Can we generalize beyond directed cycles?

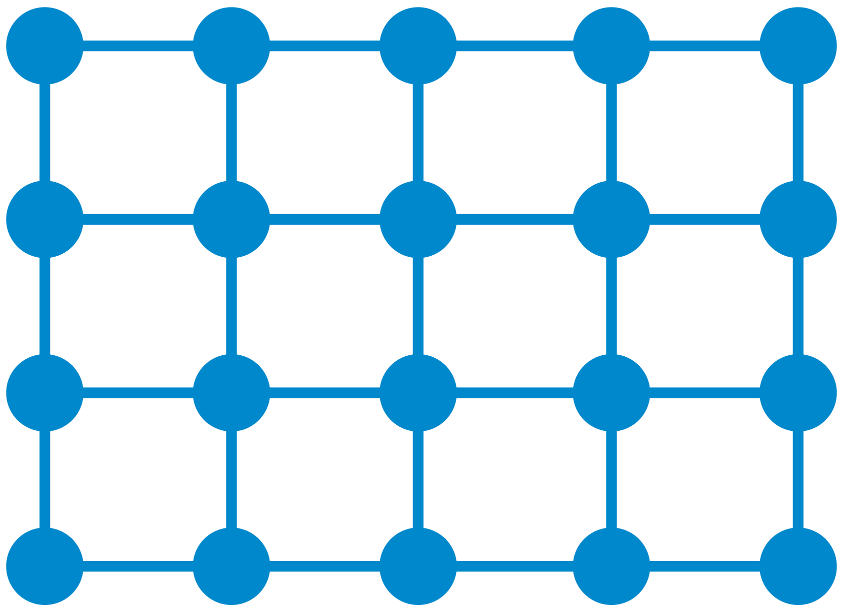
Cycles, paths



Cycles, paths



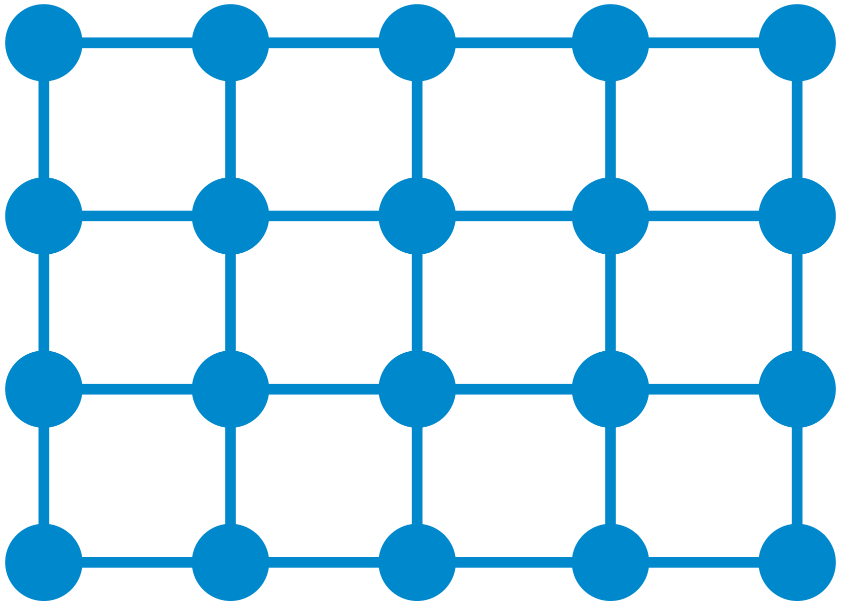
Grids



Cycles, paths

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

Grids

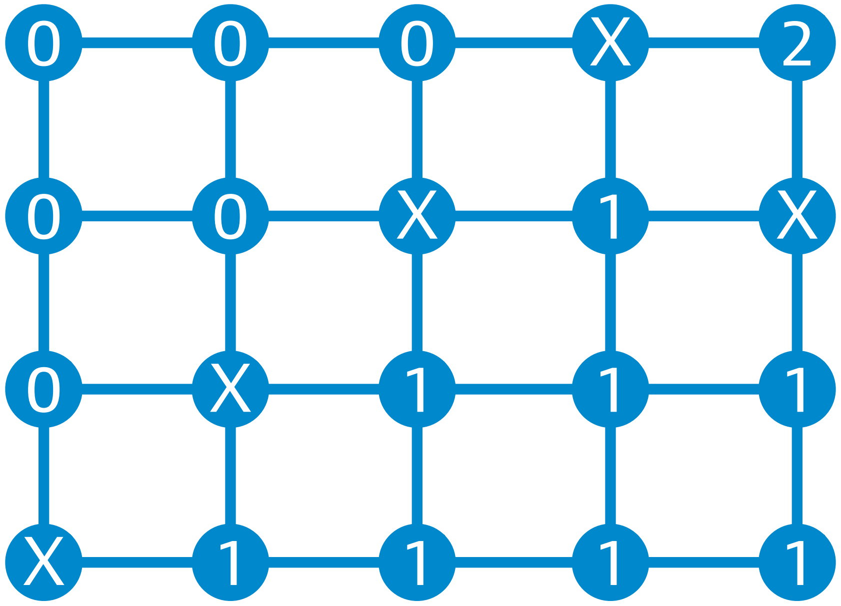


Cycles, paths

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 $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C}_k$

- C_k = distance-k coloring
- 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 $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C_k}$

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

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

- check all possible candidate functions $\boldsymbol{B}$
- 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 $\boldsymbol{k}$ = 1, 2, 3, …:

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

Normal forms

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

- C_k =
- B = ... plored neig... s

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

- check all possible candidate functions $\boldsymbol{B}$
- 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 $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C}_k$

ite function that maps colored
nei orhoods to local outputs

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

- check all possible candidate functions $\boldsymbol{B}$
- 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 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!

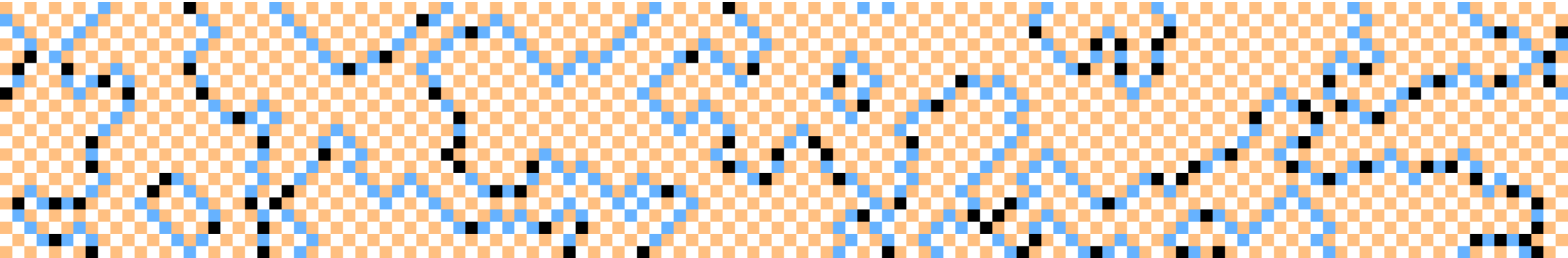
- Natural problems often solvable with a ***small k***
- We can make it more feasible in practice:
  - more ***“compact” normal forms***,
    e.g. distance-*k* coloring → ruling set

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



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?

Big picture:
meta-computational
questions and
algorithms synthesis

Meta questions

- **Designing algorithms that design algorithms?**
- Studying the computational complexity of studying computational complexity?
- Using computation (in practice) to understand computation (in theory)?

…

Verification & synthesis

- **Algorithm verification:**
  - given problem *P* and algorithm *A*
  - does *A* solve *P*?
- **Algorithm synthesis:**
  - given problem *P*
  - find an algorithm *A* that solves *P*?

Verification & synthesis

- **Algorithm verification often hard**
  - recall: halting problem
- **Algorithm synthesis can be easier!**
  - verification must handle arbitrary algorithms
  - synthesis can produce “nice” algorithms

Conclusions

Take-home messages

- **Algorithm design can be made systematic and mechanical, even computers can do it!**
  - we need the right ***representations*** for computational problems & algorithms
  - this is ***not machine learning*** — but is this AI?
- **Key concepts:**
  - ***meta-computational*** problems
  - algorithm ***verification*** vs. algorithm ***synthesis***