Derandomization of Distributed Graph Algorithms

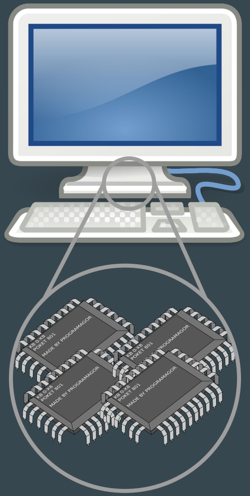
Václav Rozhoň (ETH)

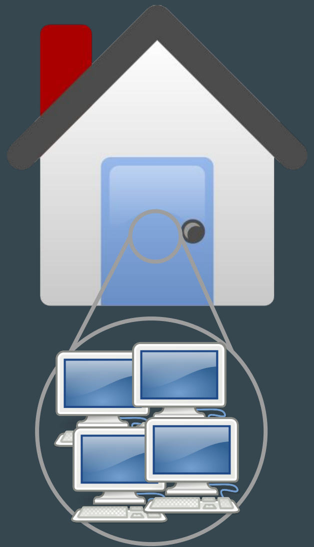
joint work with
Mohsen Ghaffari (ETH)

Models for programming parallel algorithms

Centralized models

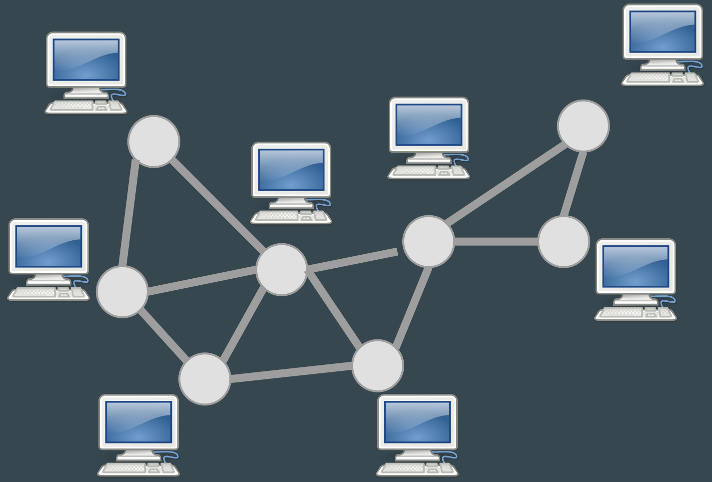
PRAM [Fortune, Wyllie STOC'78]s





MPC [Karloff, Suri, Vassilvitskii SODA'10]
(MapReduce [Dean, Ghemawat OSDI'04)

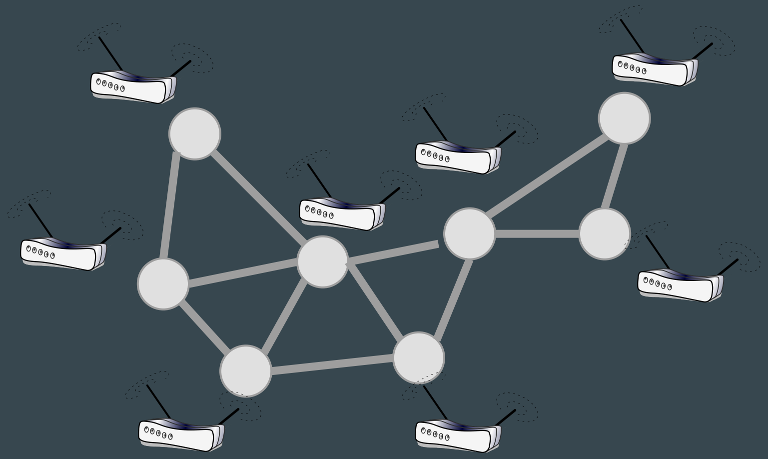
Distributed models



LOCAL model [Linial FOCS'87]

The LOCAL model of distributed graph algorithms

Example: wifi routers want to broadcast on different frequencies.

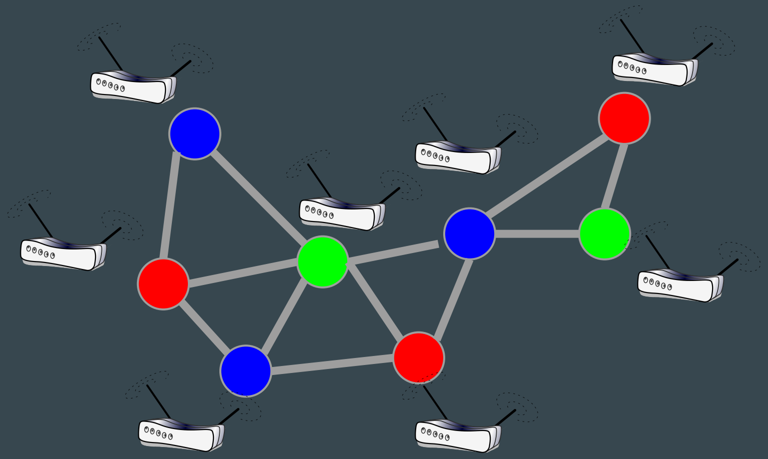


LOCAL model [Linial FOCS'87]

The LOCAL model of distributed graph algorithms

Example: wifi routers want to broadcast on different frequencies.

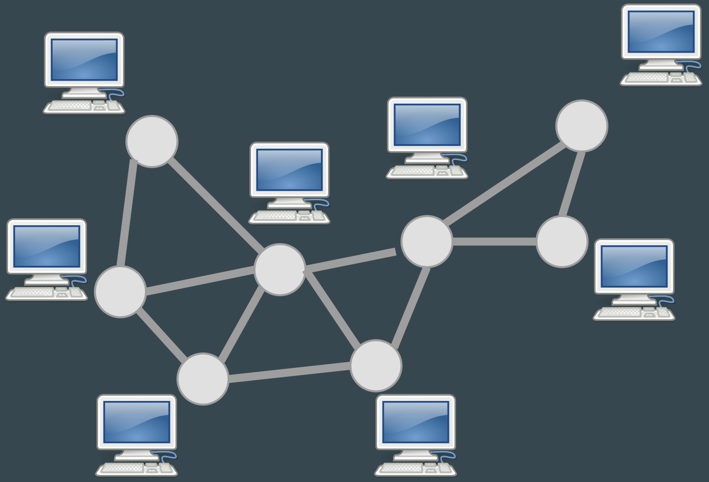
=> coloring problem!



LOCAL model [Linial FOCS'87]

The LOCAL model of distributed graph algorithms

- Undirected graph $G=(V,E)$ with n nodes
- One computer in each node
- Synchronous message passing rounds: in one round node can communicate with its neighbours
- Unbounded message size and computation! (more honest version: CONGEST model)
- Initially, nodes know only (upper bound on) n, optionally max degree Δ and their unique label
- In the end, each node should know its part of output
- Time complexity: number of rounds

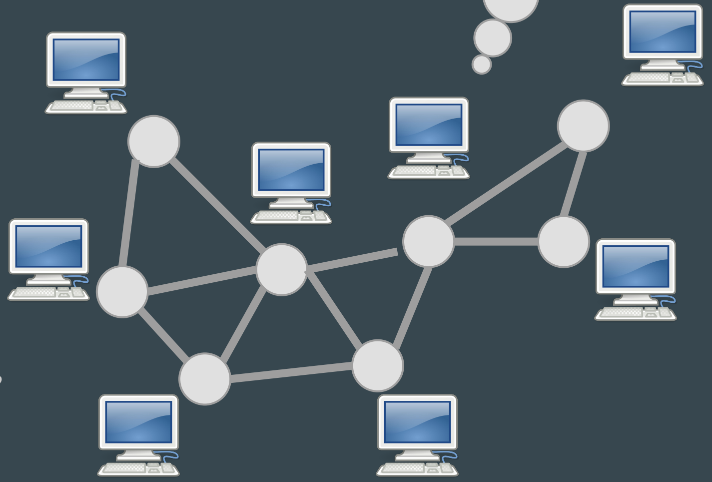


LOCAL model [Linial FOCS'87]

The LOCAL model of distributed graph algorit

We neglect: computation, congestion, asynchronicity, fault-tolerance, security, …

- Undirected graph $G=(V,E)$ with n nodes
- One computer in each node
- Synchronous message passing rounds: in one round node can communicate with its neighbours
- Unbounded message size and computation! (more honest version: CONGEST model)
- Initially, nodes know only (upper bound on) n, optionally max degree Δ and their unique label
- In the end, each node should know its part of output
- Time complexity: number of rounds



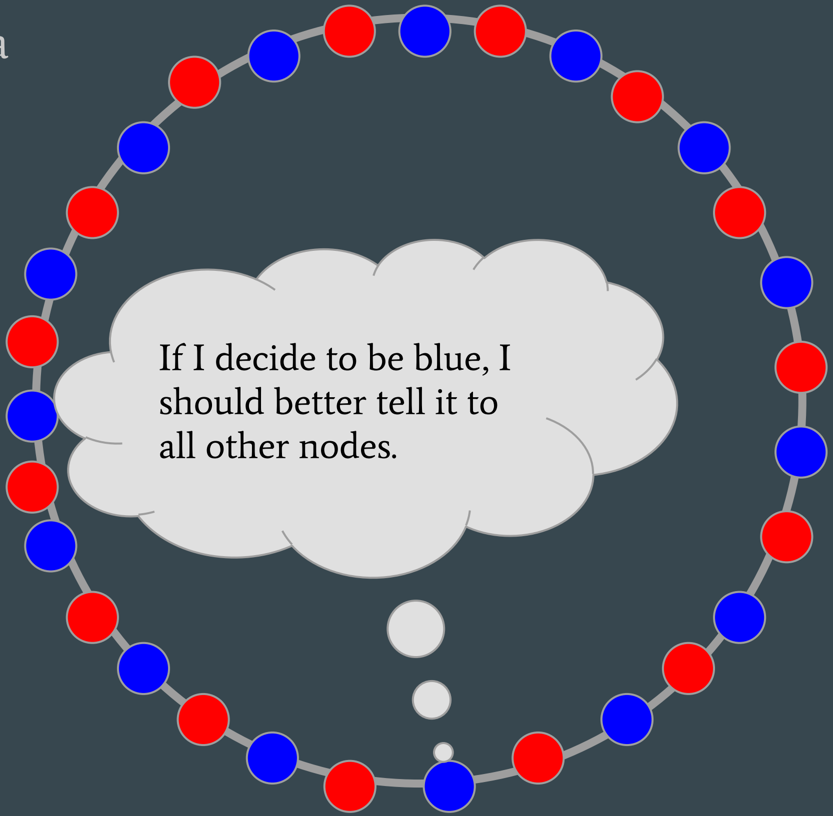
LOCAL model [Linial FOCS'87]

Let’s see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there quick distributed algorithm that finds 2-coloring?

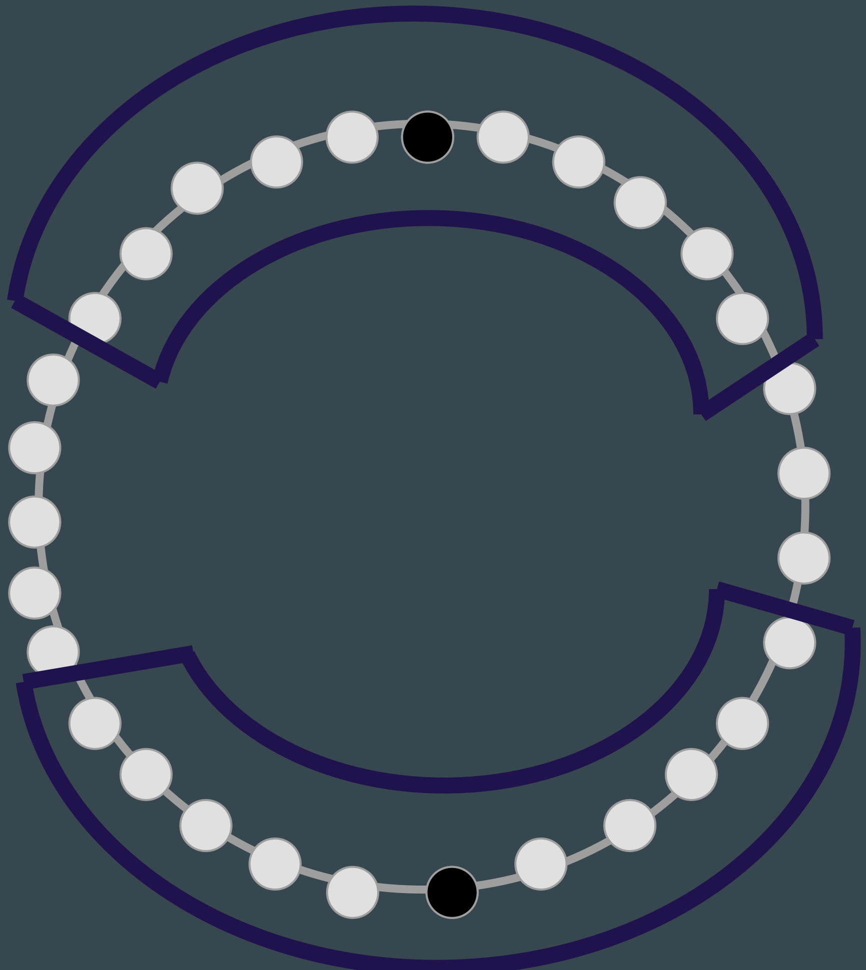
Let’s see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there quick distributed algorithm that finds 2-coloring?



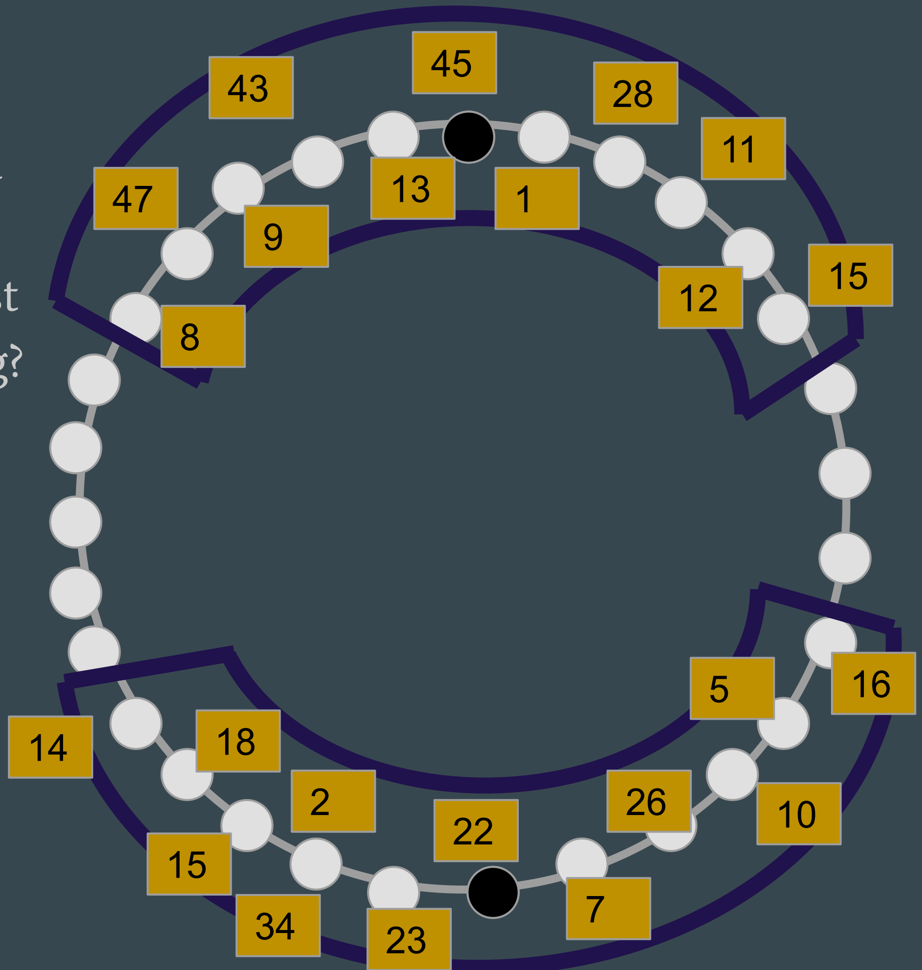
Let's see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there fast distributed algorithm that finds 2-coloring?



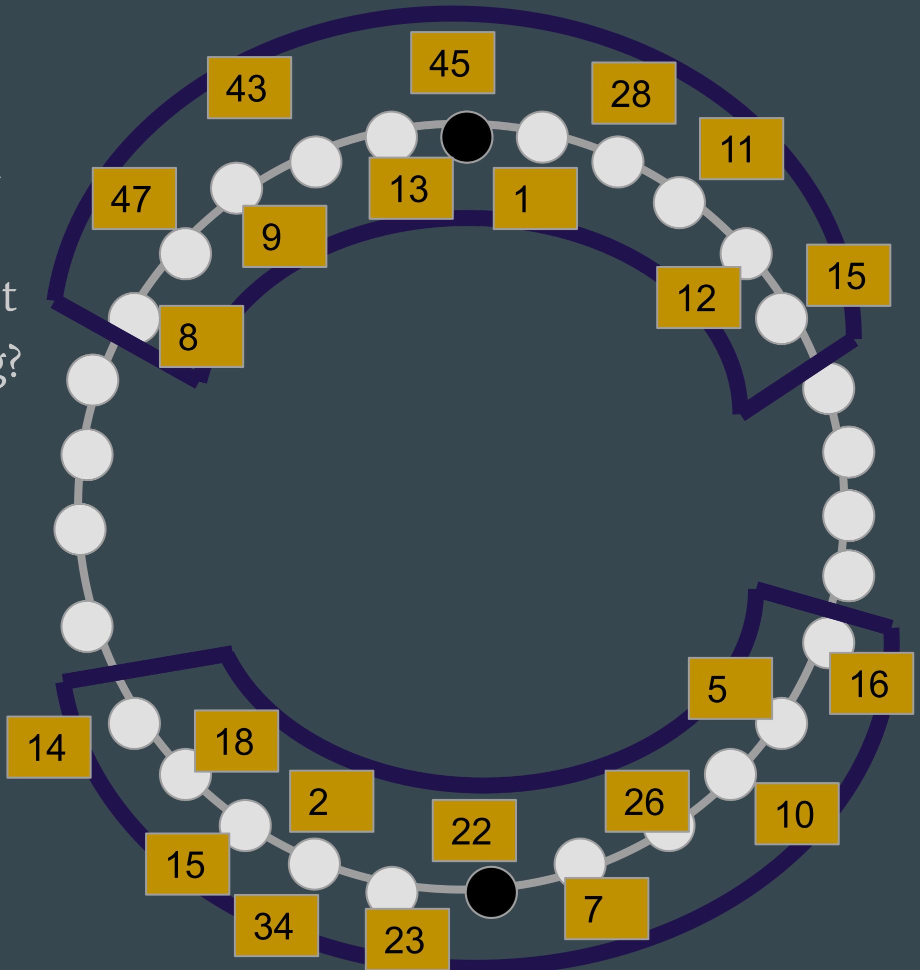
Let’s see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there fast distributed algorithm that finds 2-coloring?



Let's see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there fast distributed algorithm that finds 2-coloring?

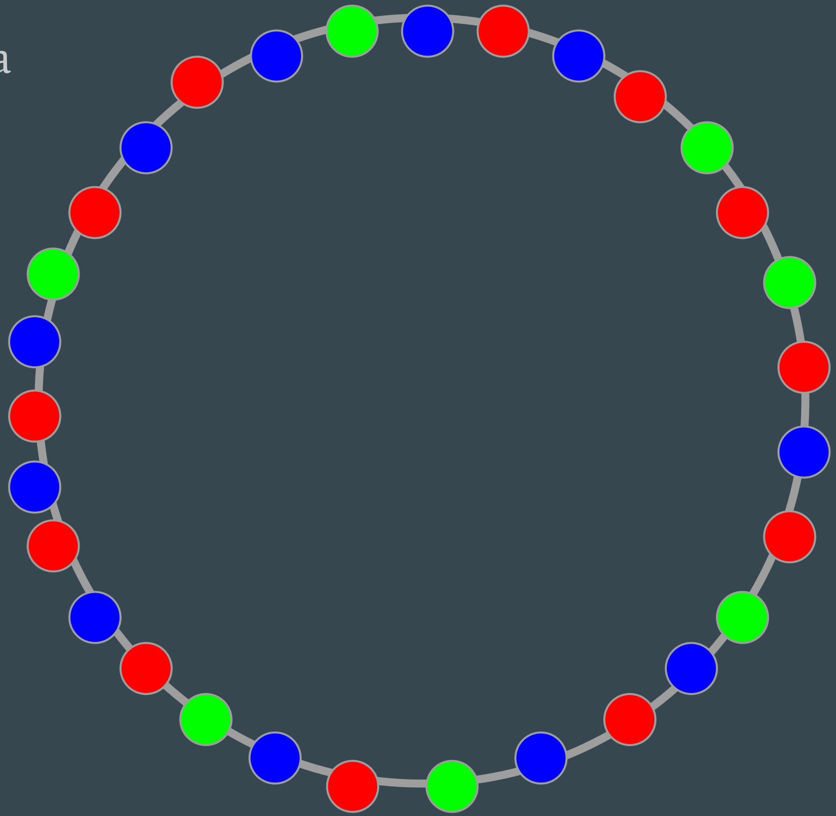


Let’s see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there quick distributed algorithm that finds 2-coloring? No!
- What about 3-coloring?

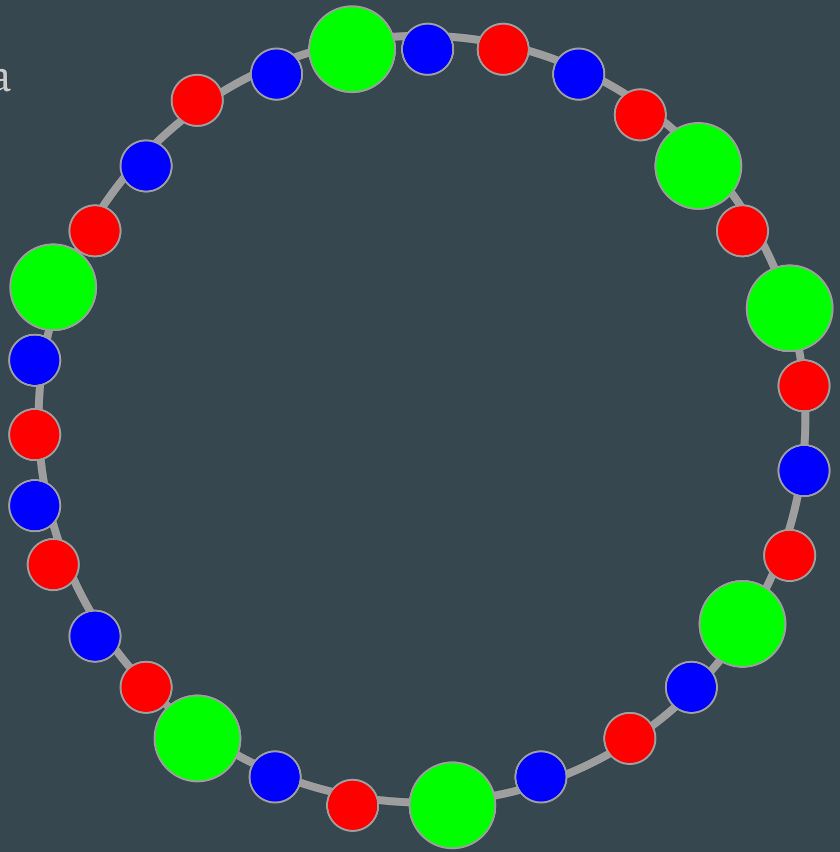
Let’s see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there quick distributed algorithm that finds 2-coloring? No!
- What about 3-coloring?



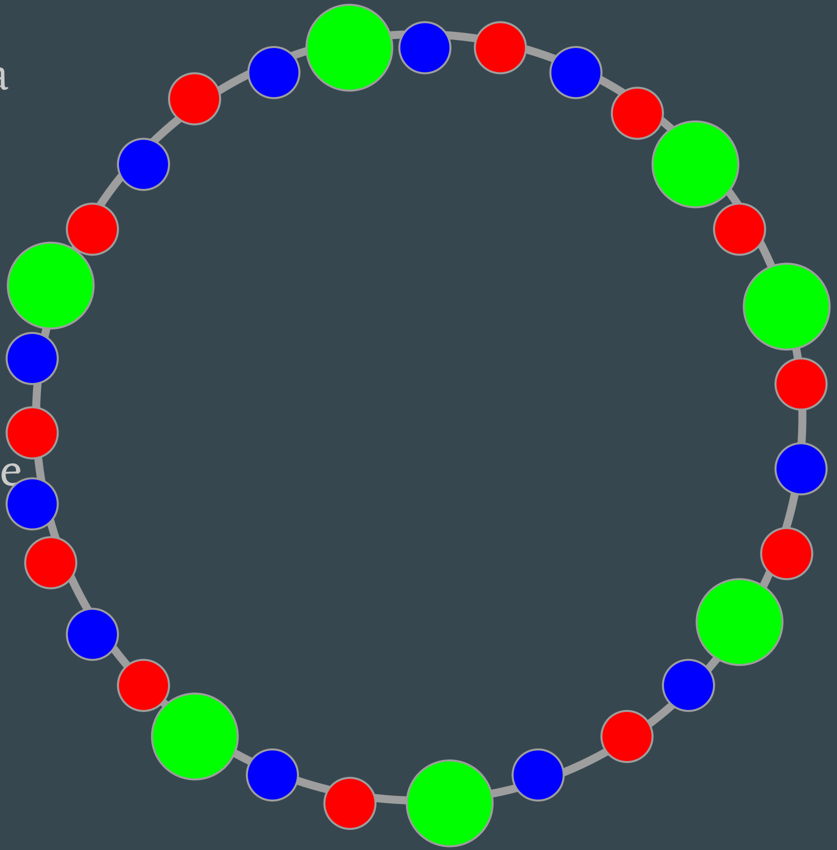
Let's see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there quick distributed algorithm that finds 2-coloring? No!
- What about 3-coloring?



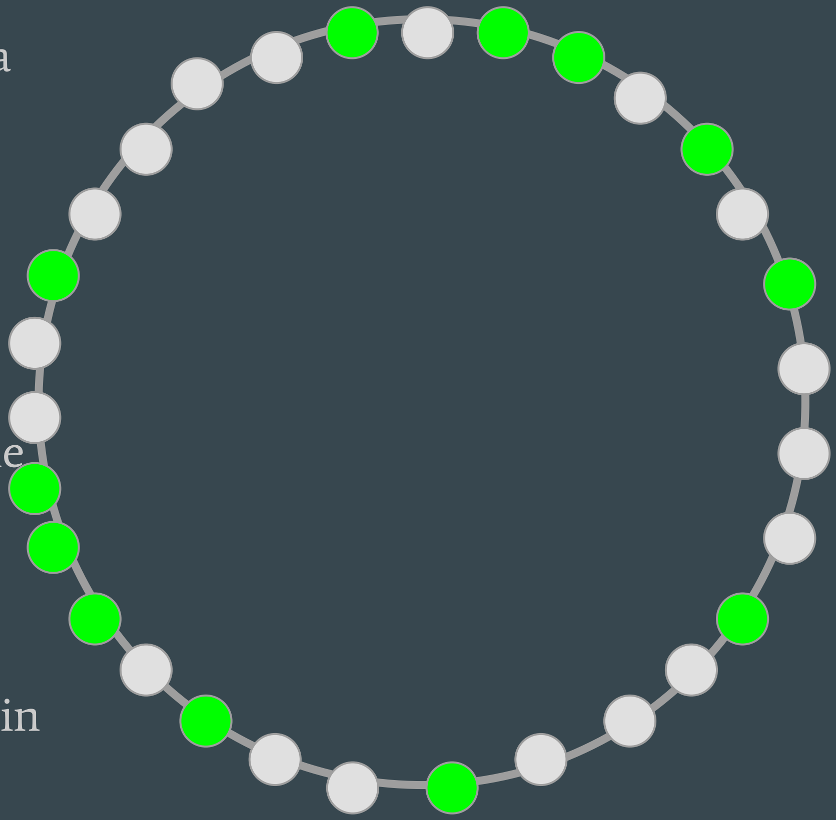
Let’s see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there quick distributed algorithm that finds 2-coloring? No!
- What about 3-coloring?
- Let’s start with first colour which splits the graph in small chains.
- These will then be two-coloured in time proportional to their length.



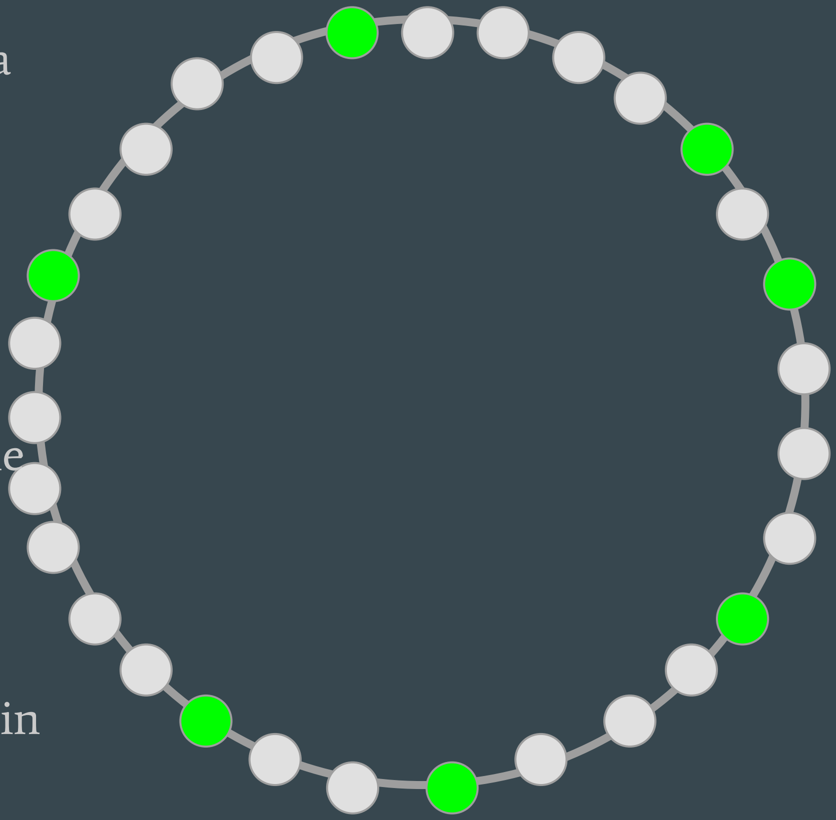
Let’s see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there quick distributed algorithm that finds 2-coloring? No!
- What about 3-coloring?
- Let’s start with first colour which splits the graph in small chains.
- These will then be two-coloured in time proportional to their length.
- Random coloring + cleaning does the job in O(log n) rounds w.h.p.



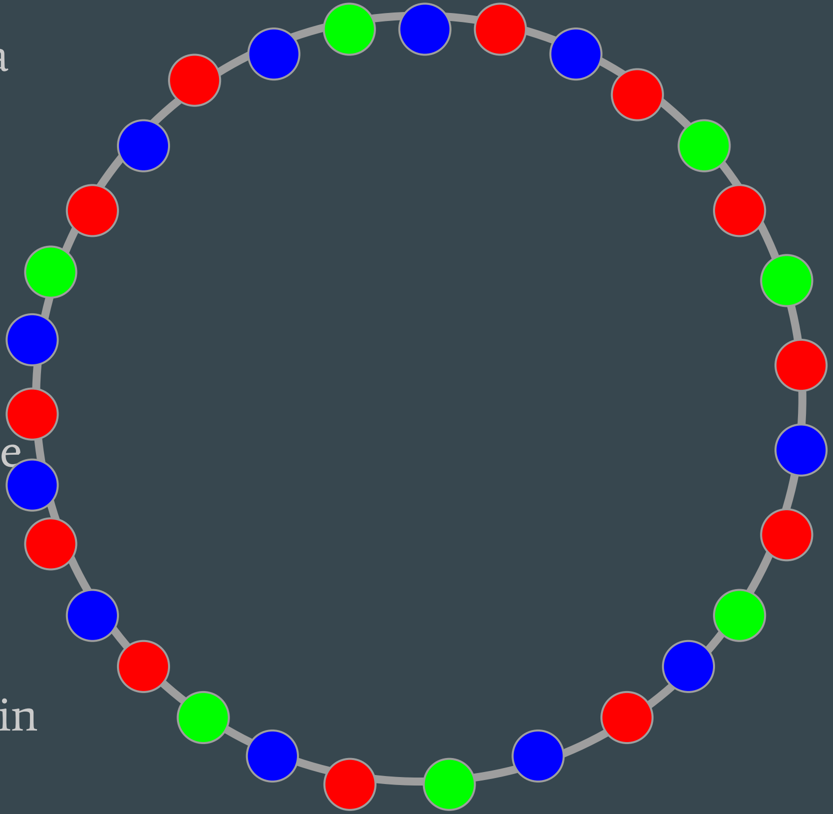
Let’s see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there quick distributed algorithm that finds 2-coloring? No!
- What about 3-coloring?
- Let’s start with first colour which splits the graph in small chains.
- These will then be two-coloured in time proportional to their length.
- Random coloring + cleaning does the job in O(log n) rounds w.h.p.



Let’s see how it works!

- Can we color at least a special graph, e.g. a cycle?
- If it has even number of nodes, is there quick distributed algorithm that finds 2-coloring? No!
- What about 3-coloring?
- Let’s start with first colour which splits the graph in small chains.
- These will then be two-coloured in time proportional to their length.
- Random coloring + cleaning does the job in O(log n) rounds w.h.p.

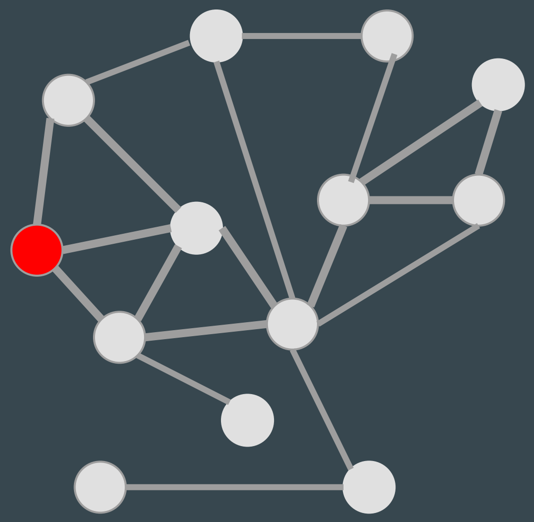


More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.

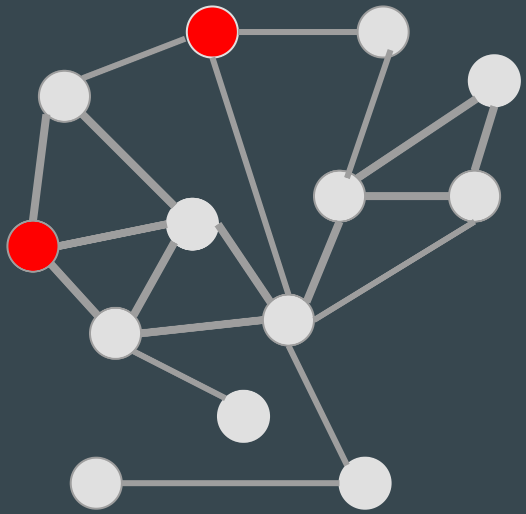
More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.



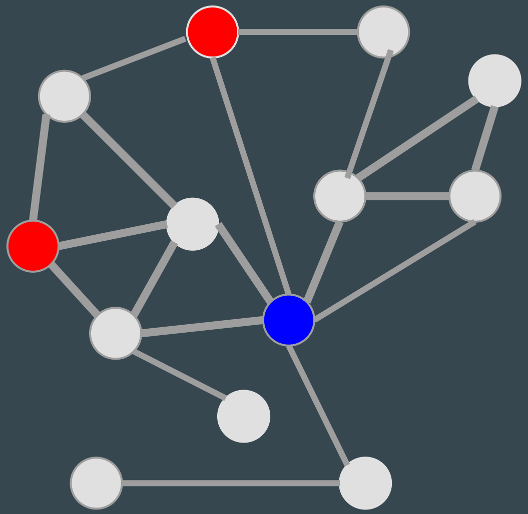
More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.



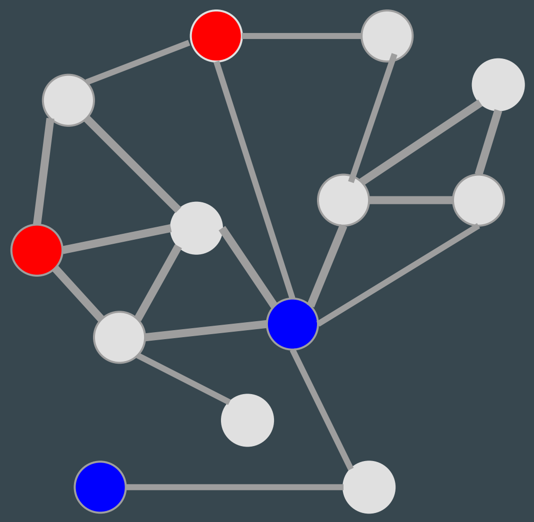
More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.



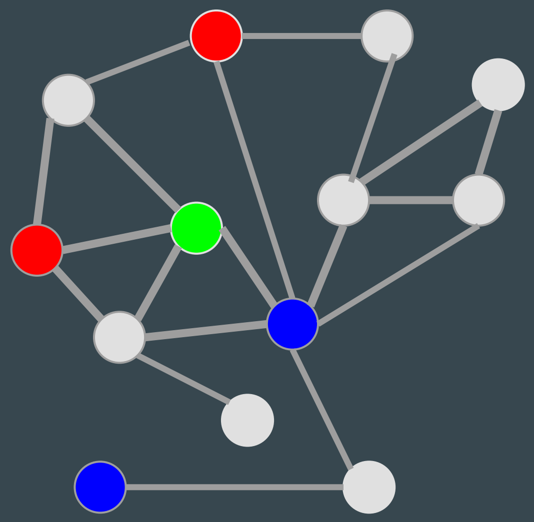
More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.



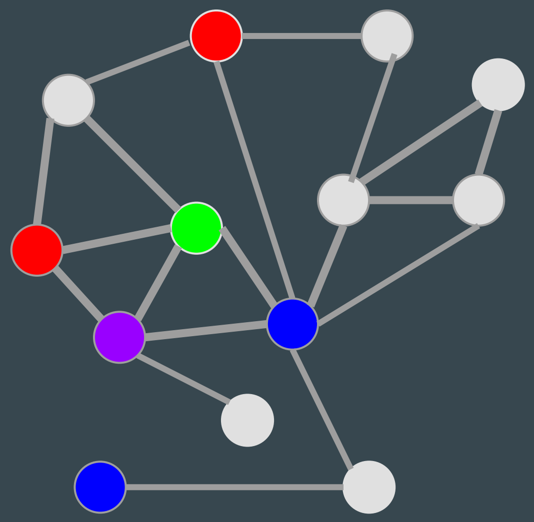
More general examples

Previous example is a special case of the *Δ+1* **coloring problem**.



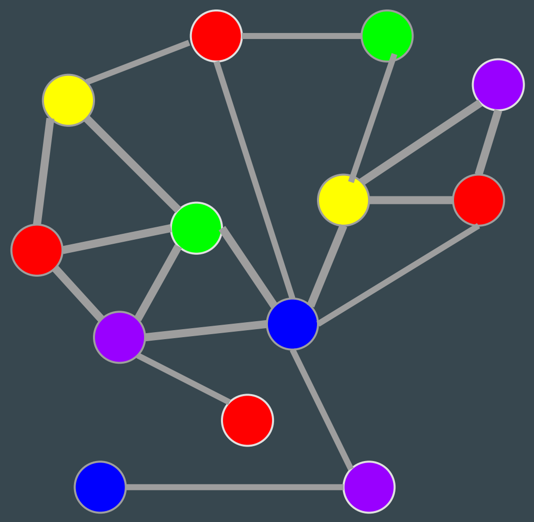
More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.



More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.



More general examples

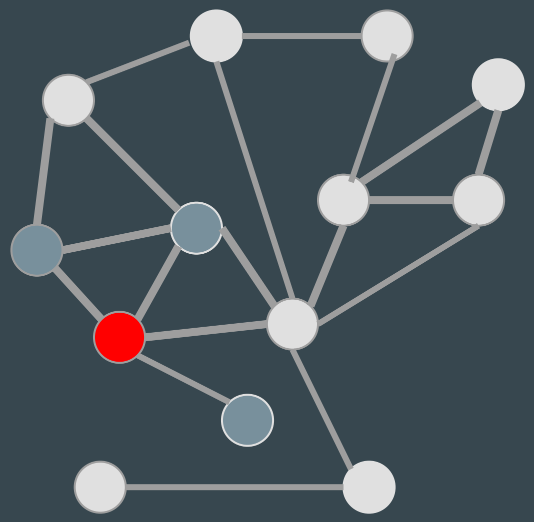
Previous example is a special case of the ***Δ+1* coloring problem**.

Maximal independent set is another example of a problem that is easy to solve sequentially.

More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.

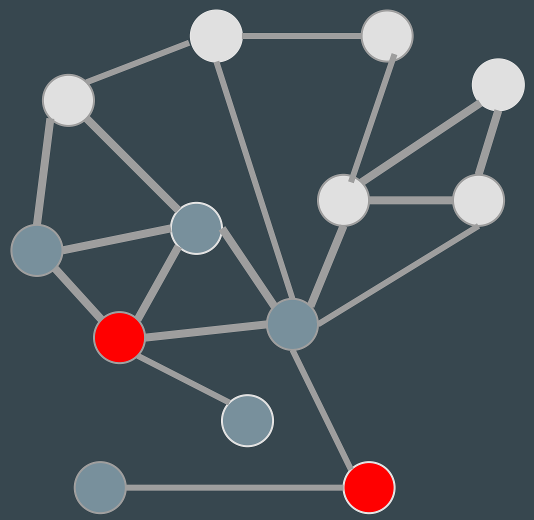
Maximal independent set is another example of a problem that is easy to solve sequentially.



More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.

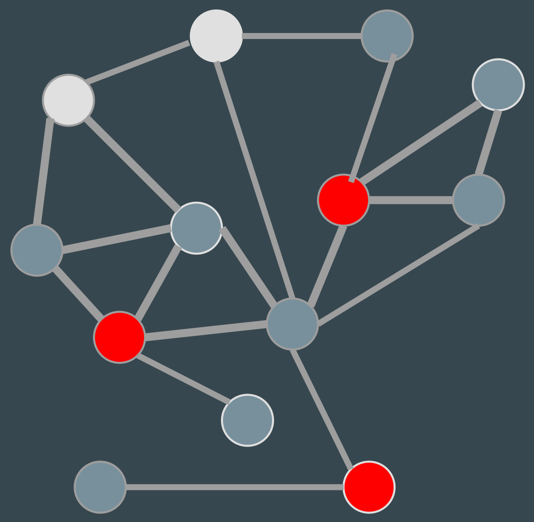
Maximal independent set is another example of a problem that is easy to solve sequentially.



More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.

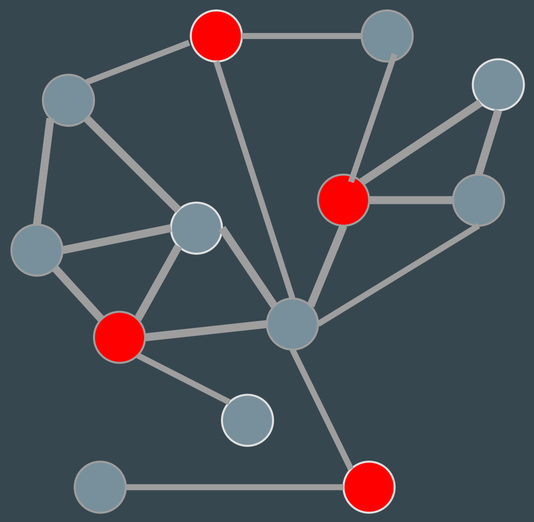
Maximal independent set is another example of a problem that is easy to solve sequentially.



More general examples

Previous example is a special case of the ***Δ+1* coloring problem**.

Maximal independent set is another example of a problem that is easy to solve sequentially.



More general examples

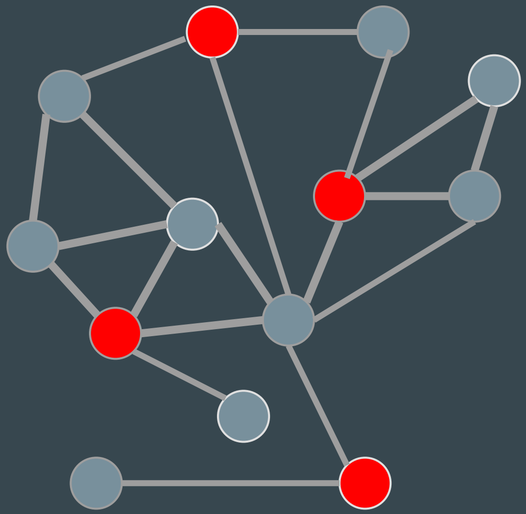
But how do we solve these problems distributedly?

Truly simple algorithm [Luby STOC'85]:

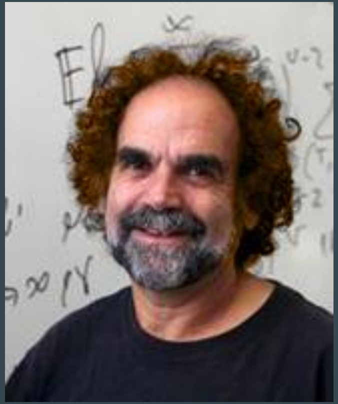
In one round

- each node picks random real number from [0,1]
- local minima join MIS and are removed together with their neighbours

Finishes in O(log n) rounds w.h.p.



Deterministic maximal independent set



Q: Is there also an efficient (polylogarithmic round) deterministic algorithm for MIS? [Linial FOCS'87]

A: Yes! [RG19+]

And also for *Δ+1* coloring, maximal matching and many other problems (approximation of maximum independent set, hypergraph splitting,...)

For some problems an efficient algorithm was known (e.g. maximal matching [Hanckowiak, Karonski, Panconesi SODA'98 & PODC'99], [Fischer DISC'17], but they are unfortunately very clever!

Principled approach

What we really seek is a **principled approach for distributing sequential algorithms** (think of maximal independent set and *Δ+1* coloring)

This will now lead us to the problem of **network decomposition** (think of it as a complete problem for this task).

Let's keep maximal independent set as our running example (more complicated problems may have sequential algorithms with larger radius or consist of several sequential passes but this does not affect the framework).

Principled approach

Easy case for our quest: the underlying graph has small (polylogarithmic) diameter.

Principled approach

Easy case for our quest: the underlying graph has small (polylogarithmic) diameter.

I am the lowest id node. I will collect the topology of the whole graph. Then I internally simulate the sequential algorithm and send the solution to other nodes.

Principled approach

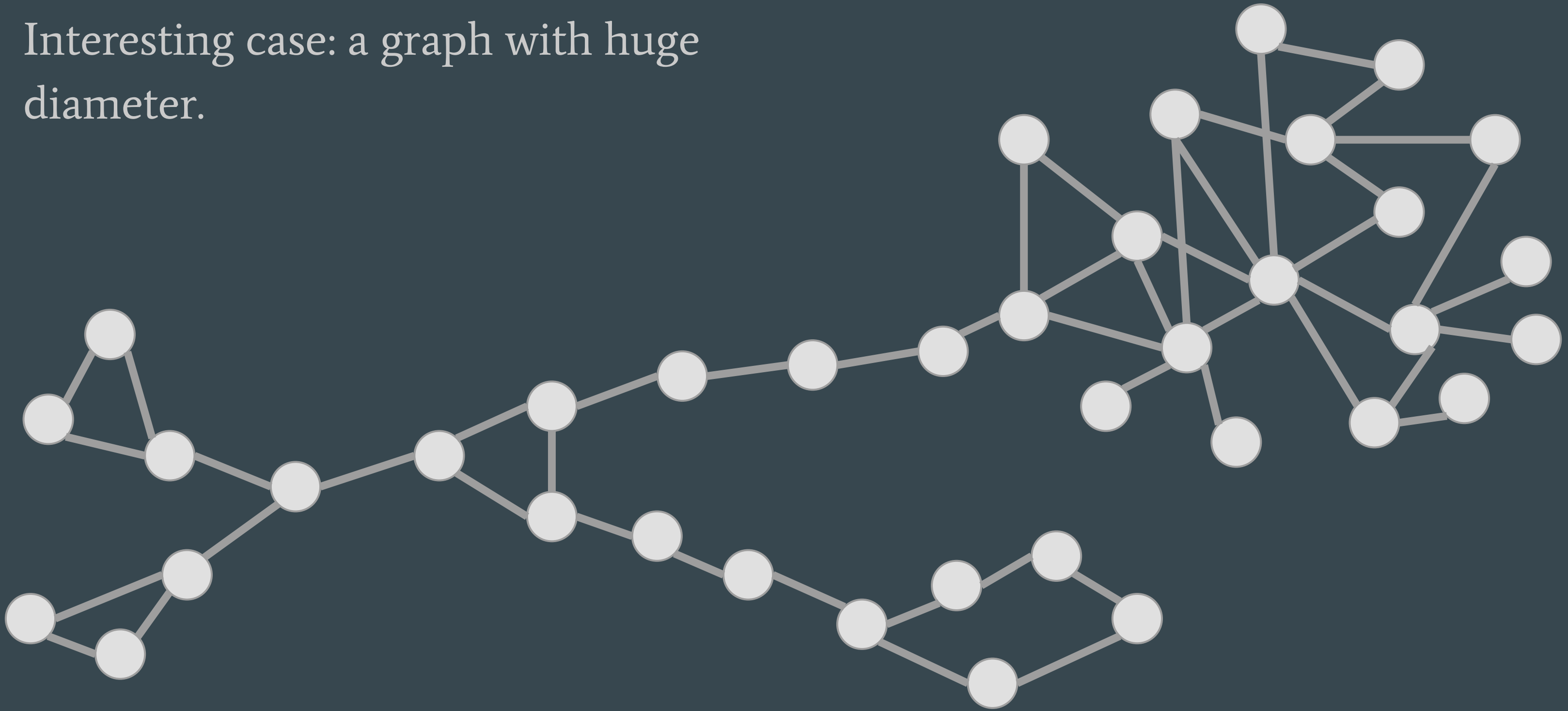
Easy case for our quest: the underlying graph has small (polylogarithmic) diameter.

I am the lowest id node. I will collect the topology of the whole graph. Then I internally simulate the sequential algorithm and send the solution to other nodes.

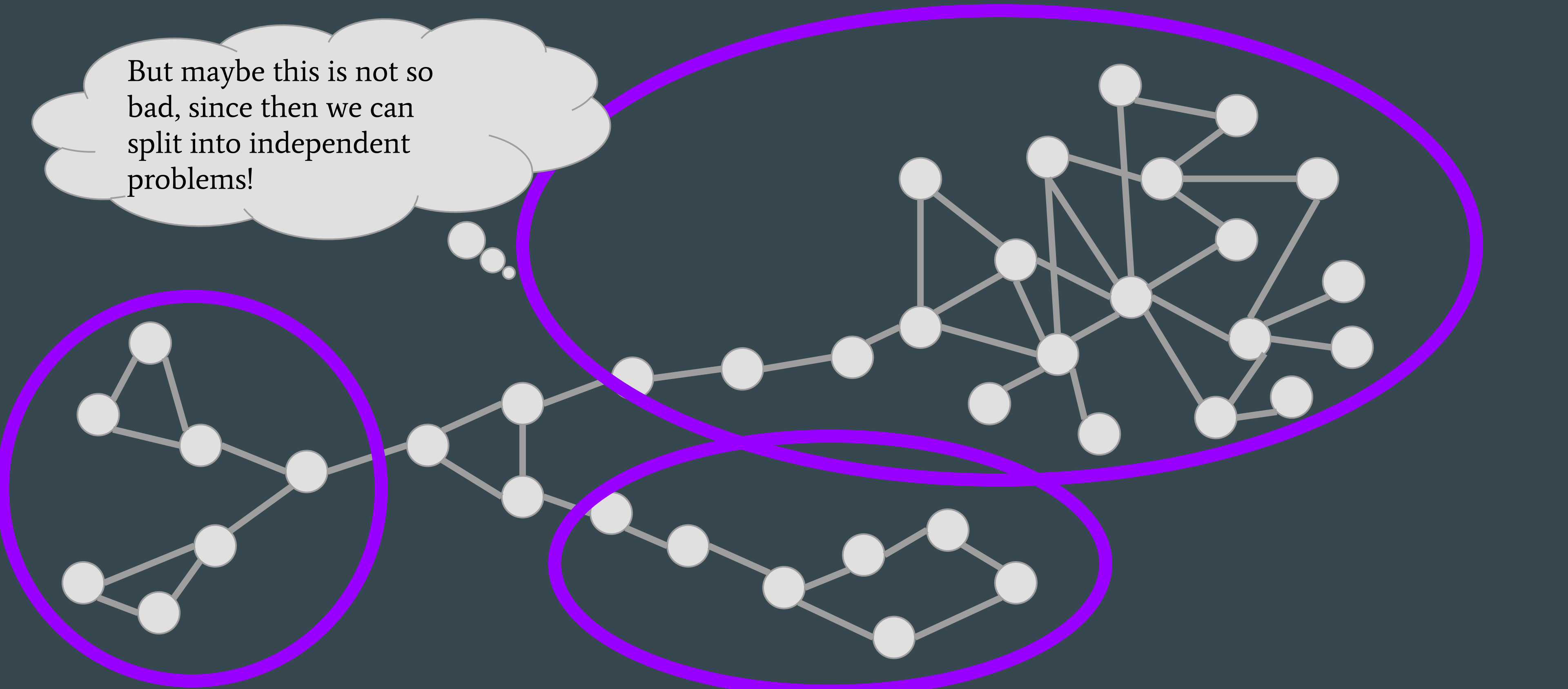
This really works for any problem. Huge diameter graphs are what we attempt to fight in the LOCAL model.

Principled approach

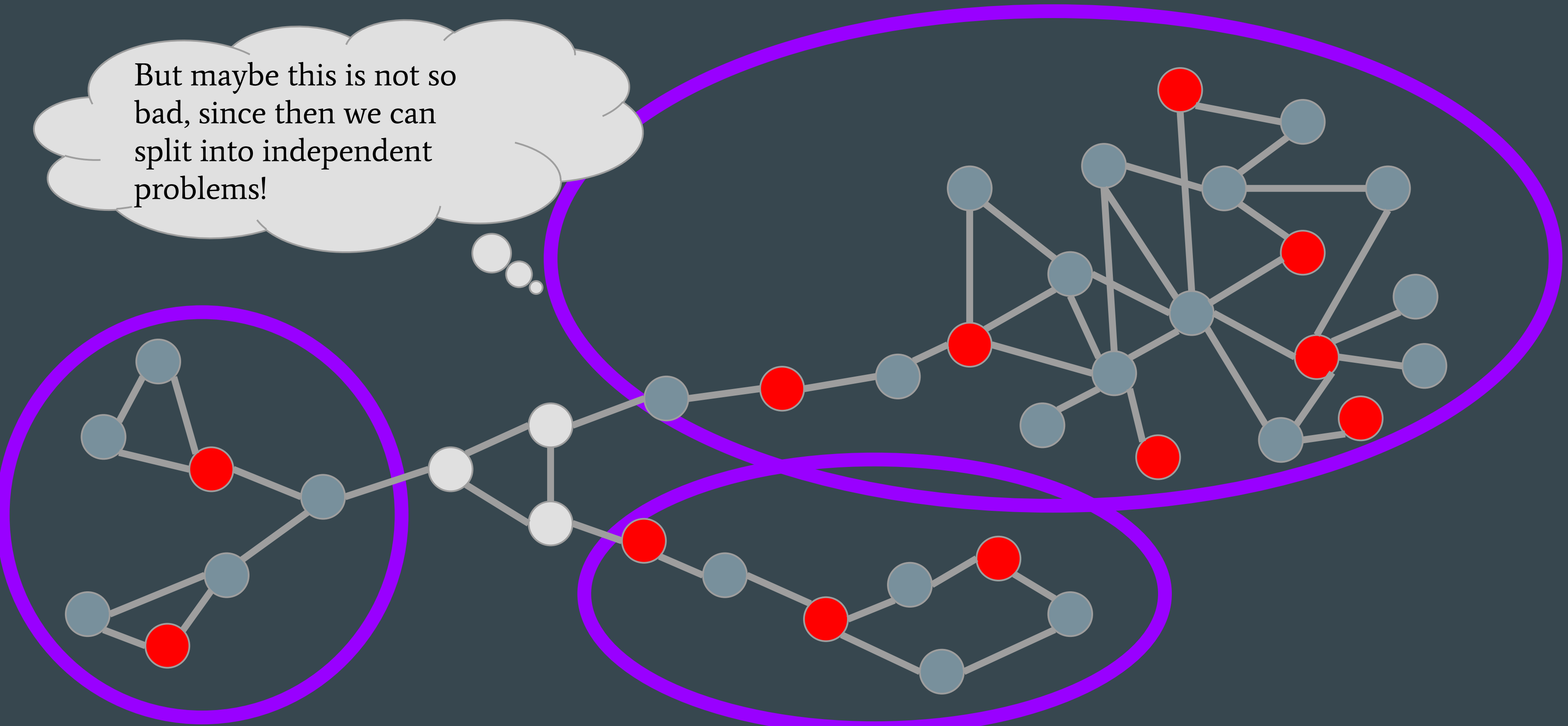
Interesting case: a graph with huge diameter.



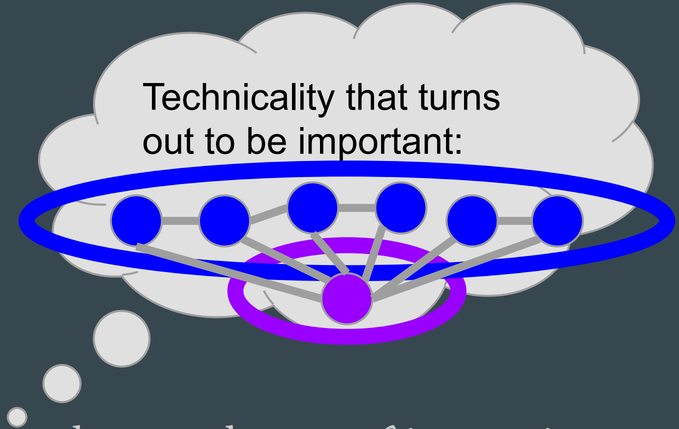
Principled approach



Principled approach

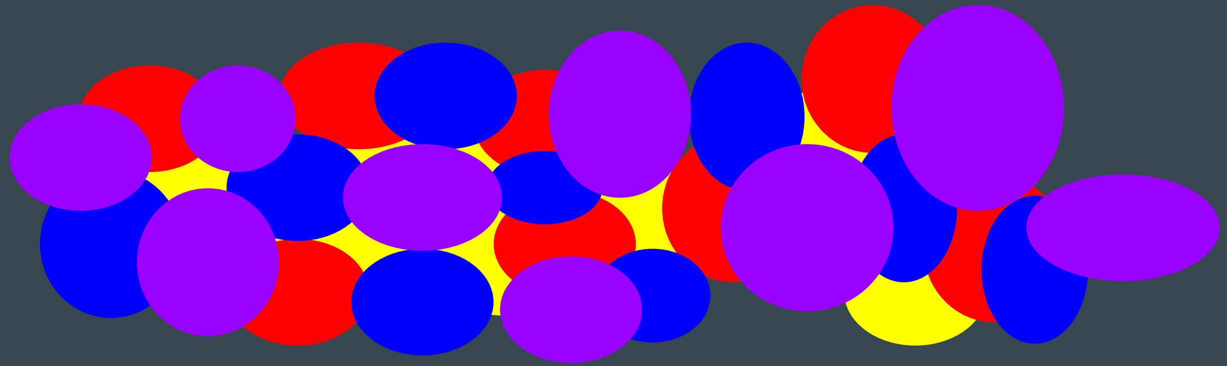


Network decomposition

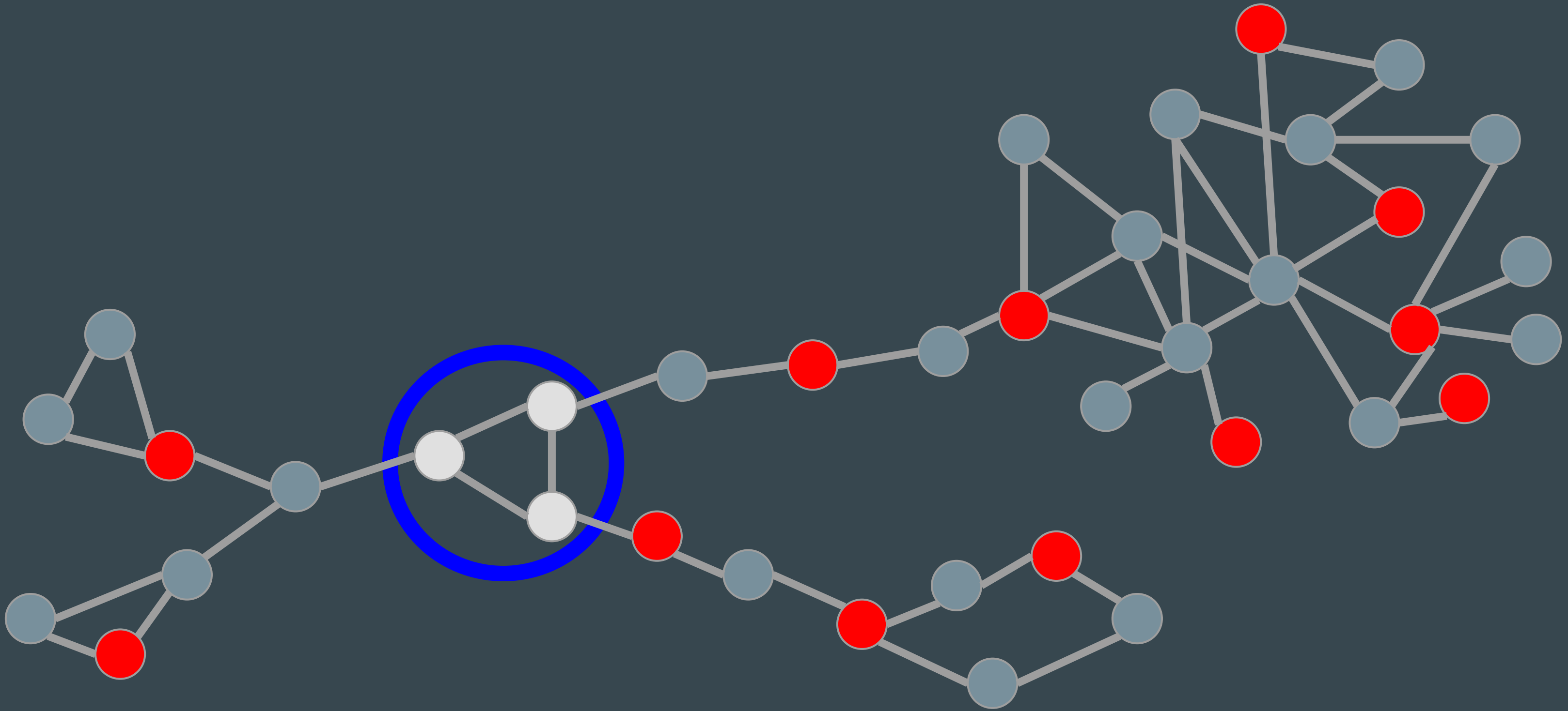


Color vertices of the underlying graph so that:

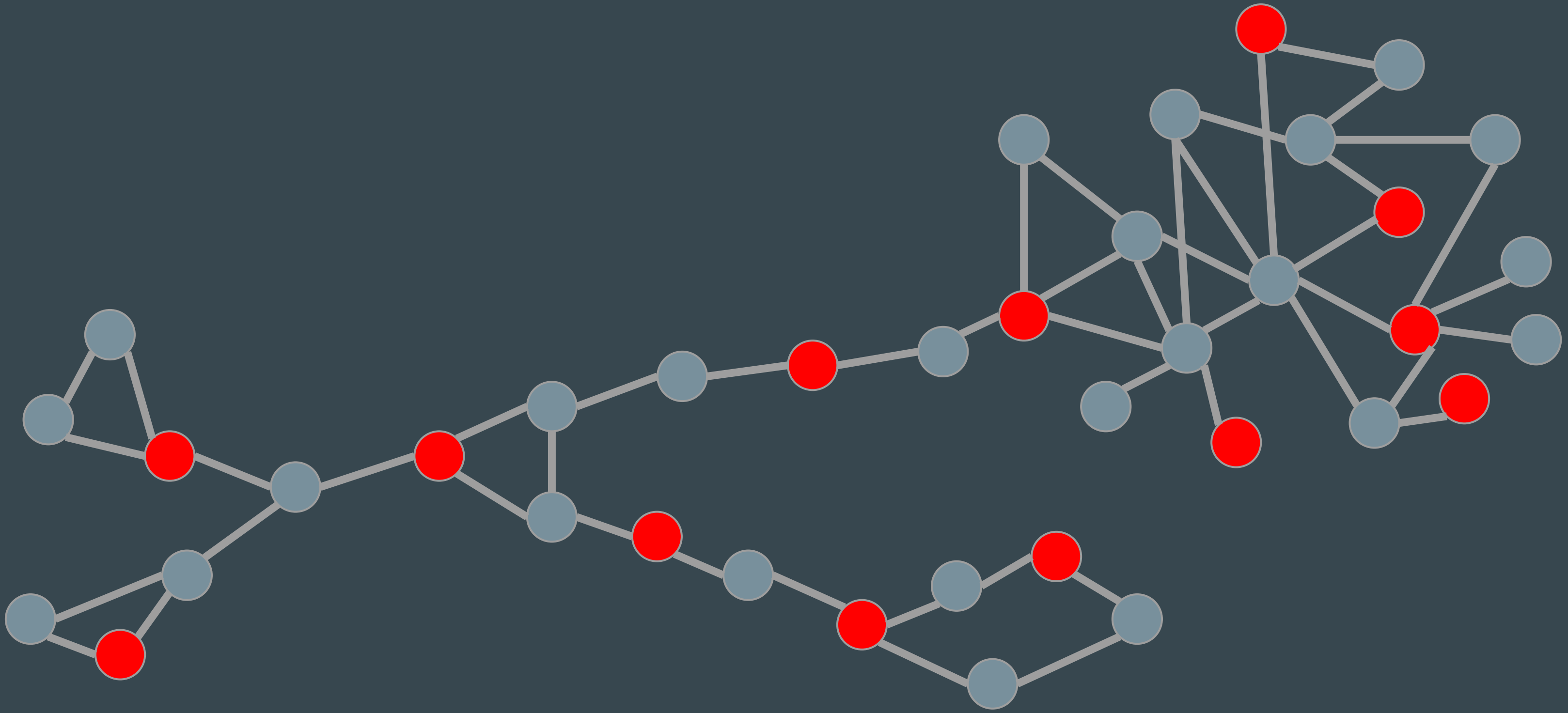
- there are polylogarithmic number of colors
- if we fix one component of vertices of the same color, then each two of its vertices are close in the underlying graph



Principled approach



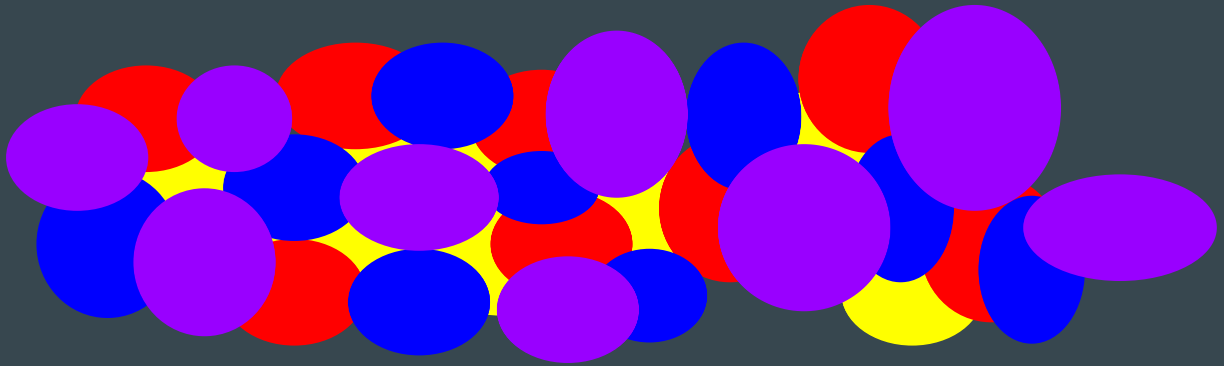
Principled approach



Network decomposition

Color vertices of the underlying graph so that:

- there are polylogarithmic number of colors
- if we fix one component of vertices of the same color, then each two of its vertices are close in the underlying graph



Sequential algorithm

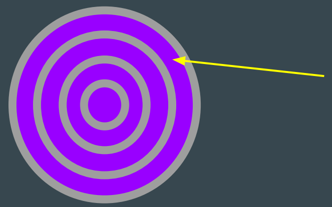
Let's start with just the first color class.

We should try to color at least half of the vertices, since then we have O(log n) colors.

Sequential algorithm

Sequential algorithm

Sequential algorithm



How many new vertices are there in the next layer?

A) At most the same as the number of vertices we have already seen.

B) At least the same as the number of vertices we have already seen.

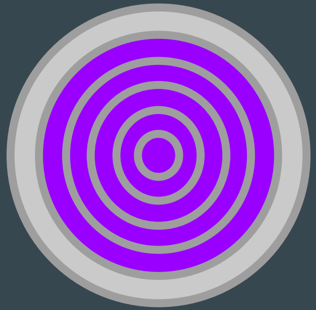
Sequential algorithm

How many new vertices are there in the next layer?

A) At most the same as the number of vertices we have already seen. Cool! Let’s delete the boundary and make a new cluster.

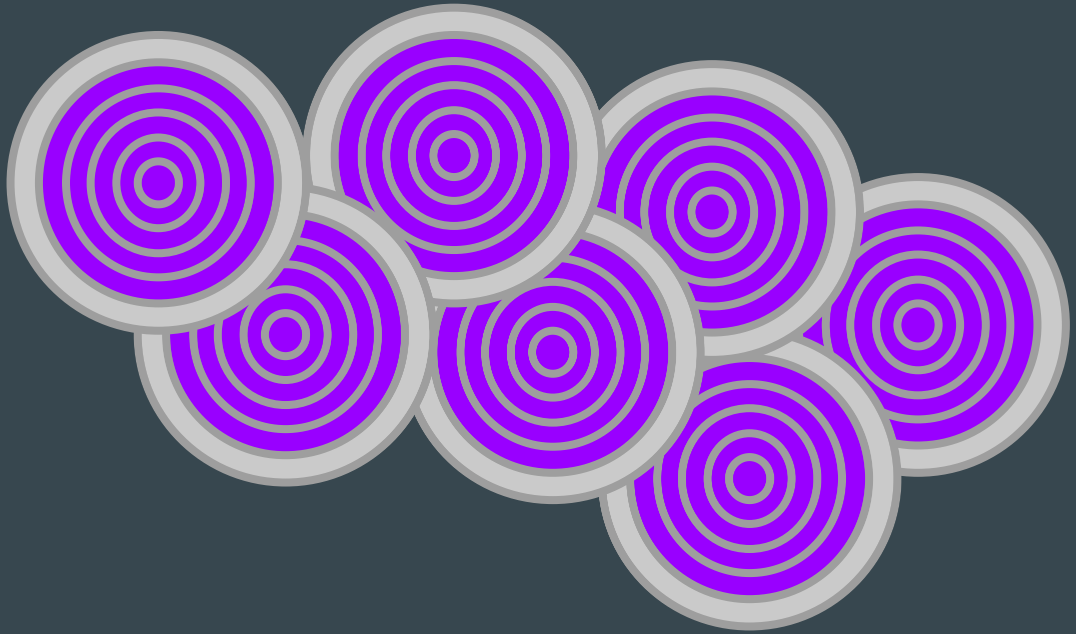
B) At least the same as the number of vertices we have already seen. Not bad! There are at most log n such steps!

Sequential algorithm



Sequential algorithm

Sequential algorithm



When the whole graph is processed, at most half of the vertices remained uncolored.

Hence, we color all nodes in $\log n$ iterations of this procedure.

Network decomposition

In some sense, the network decomposition problem is a complete problem for our quest for distributing sequential algorithms!

There is a long line of work on understanding network decomposition:

[Awerbuch, Goldberg, Luby, Plotkin FOCS'89] $2^{O(\sqrt{\log n \log\log n})}$-round det. algorithm

[Linial, Saks '90] poly(log n)-round randomized algorithm

[Panconesi, Srinivasan '96] $2^{O(\sqrt{\log n})}$-round det. algorithm

[Ghaffari, Kuhn, Maus STOC'17], [Ghaffari, Harris, Kuhn FOCS'18] our narrative

[Rozhoň, Ghaffari '19+] poly(log n)-round deterministic algorithm

Derandomization

Via method of conditional expectation [GHK FOCS'18], we get general derandomization theorem:

"For all problems that allow polylogarithmic-round randomized algorithm*, there is also a polylogarithmic-round **deterministic** algorithm. "

*whose solution can be checked deterministically in polylogarithmic time

Details on the blackboard if time allows.

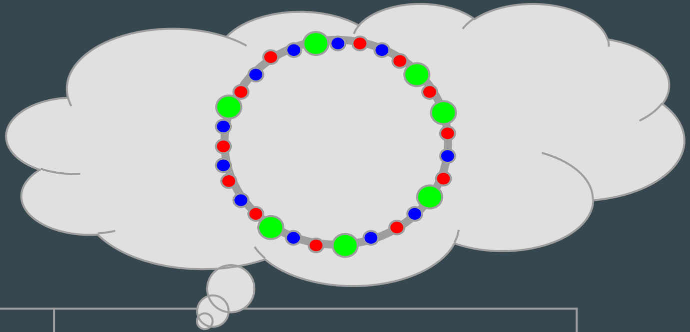
Randomized algorithms

	deterministic	randomized
maximal independent set	$2^{O(\sqrt{\log n})}$ [PS] poly(log n) [RG]	log n [Luby]

Randomized algorithms

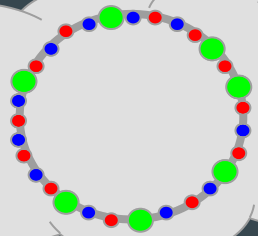
	deterministic	randomized
maximal independent set	$2^{O(\sqrt{\log n})}$ [PS] poly(log n) [RG]	log n [Luby] log Δ + $2^{O(\sqrt{\log\log n})}$ [Ghaffari]

Randomized algorithms



	deterministic	randomized
maximal independent set	$2^{O(\sqrt{\log n})}$ [PS] poly(log n) [RG]	log n [Luby] log Δ + $2^{O(\sqrt{\log \log n})}$ [Ghaffari]

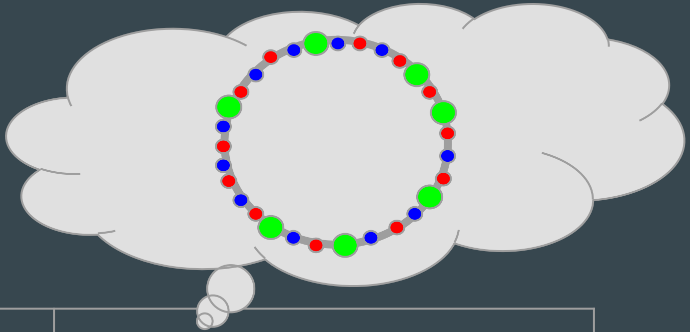
Randomized algorithms



	deterministic	randomized
maximal independent set	$2^{O(\sqrt{\log n})}$ [PS] poly(log n) [RG]	log n [Luby] $\log \Delta + 2^{O(\sqrt{\log \log n})}$ [Ghaffari]

Randomized algorithms cannot be more than exponentially faster than deterministic counterparts!!!
[Chang, Kopelowitz, Pettie FOCS'16]

Randomized algorithms



	deterministic	randomized
maximal independent set	$2^{O(\sqrt{\log n})}$ [PS] poly(log n) [RG]	log n [Luby] log *Δ* + $2^{O(\sqrt{\log \log n})}$ [Ghaffari] ⇒ log *Δ* + *poly(log log n)* [RG]

Current bounds are matching lower bounds in the first order sense.

The same principle works for *Δ+1* coloring problem and other problems!

The deterministic network decomposition algorithm

Detour: ruling set (weaker version of maximal ind. set)

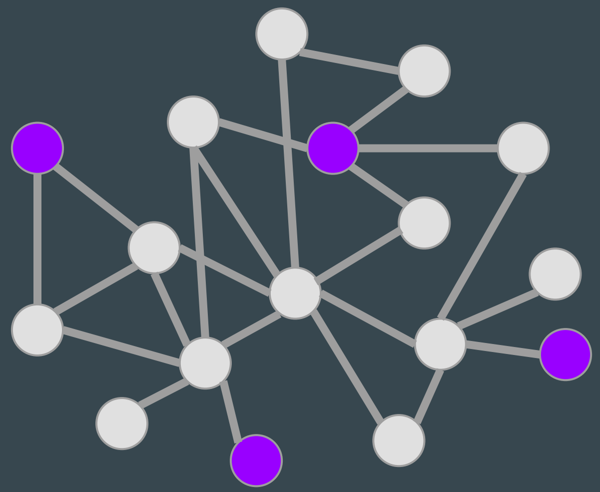
This is a subroutine used by previous algorithms (they would like to construct maximal independent set but how would they get it?)

The deterministic network decomposition algorithm

Detour: ruling set (weaker version of maximal ind. set)

This is a subroutine used by previous algorithms (they would like to construct maximal independent set but how would they get it?)

We show how to construct an independent set such that each node outside the set is at distance at most O(log n) from some node in the set.



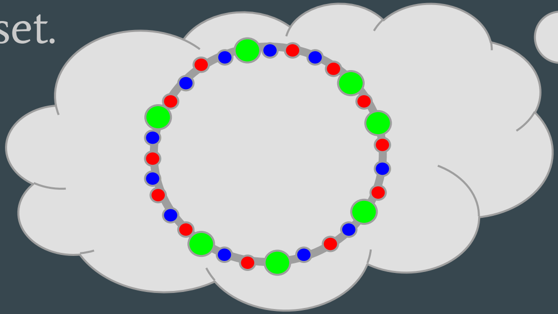
The deterministic network decomposition

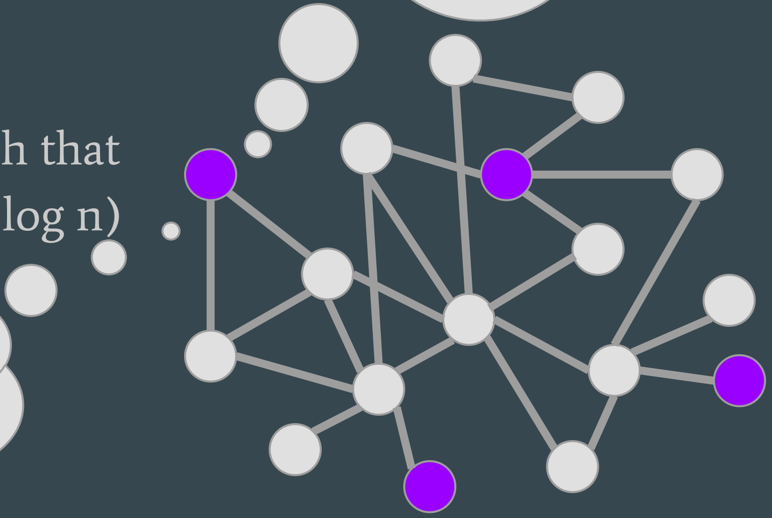
Recall, we know how to construct such a set on a cycle with randomization. But now we miss randomness and work with general graphs!

Detour: ruling set (weaker version of maximal ind. set)

This is a subroutine used by previous algorithms (they would like to construct maximal independent set but how would they get it?)

We show how to construct an independent set such that each node outside the set is at distance at most O(log n) from some node in the set.



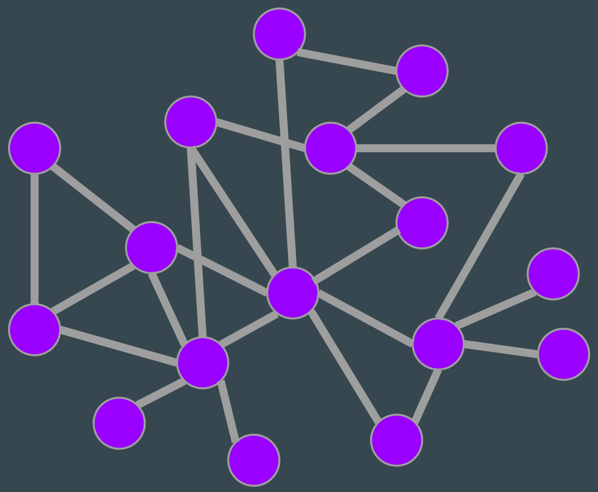


Ruling sets:

Let's think about the problem decrementally: we start with all vertices in the set.
Then, we will gradually remove some vertices, until we obtain an independent set.

Ruling sets:

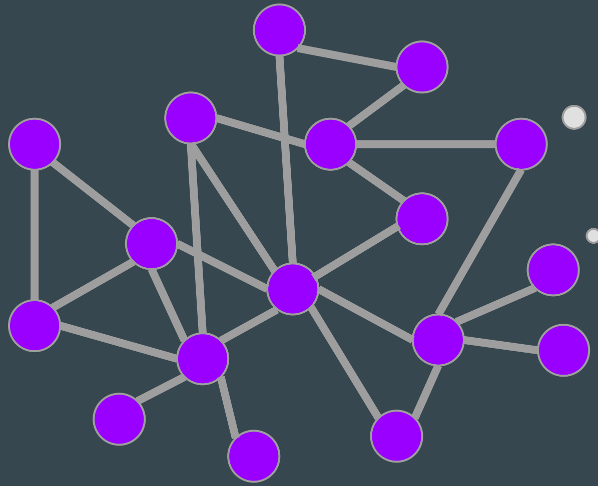
Let's think about the problem decrementally: we start with all vertices in the set.
Then, we will gradually remove some vertices, until we obtain an independent set.



Ruling sets:

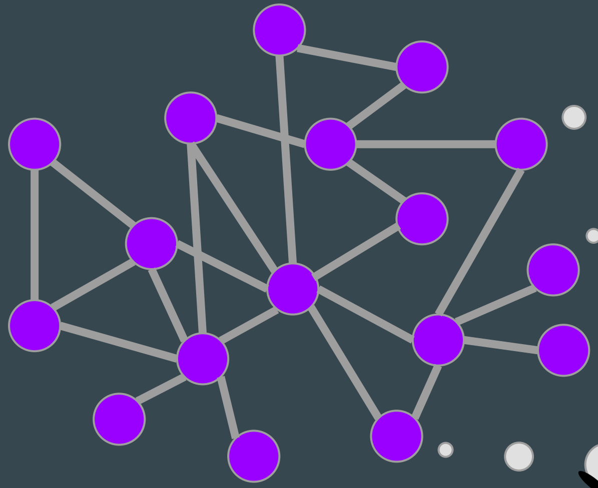
Let's think about the problem decrementally: we start with all vertices in the set.
Then, we will gradually remove some vertices, until we obtain an independent set.

The labels of the vertices have to somehow help us.



Ruling sets:

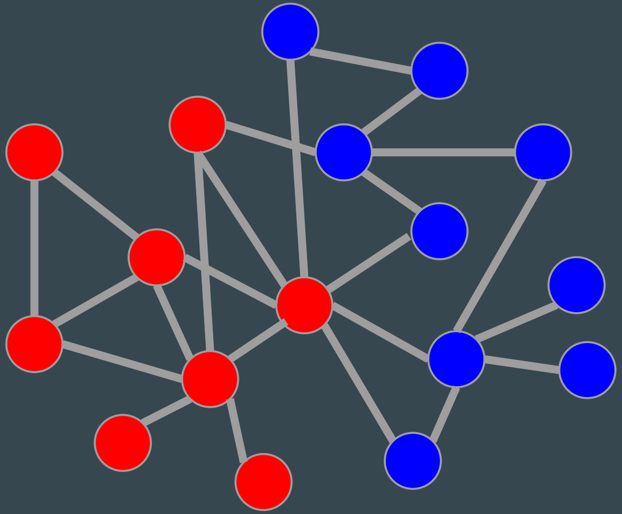
Let’s think about the problem decrementally: we start with all vertices in the set.
Then, we will gradually remove some vertices, until we obtain an independent set.



The labels of the vertices have to somehow help us.

Let’s write them down in binary!
[AGLP ‘89]

Ruling sets:

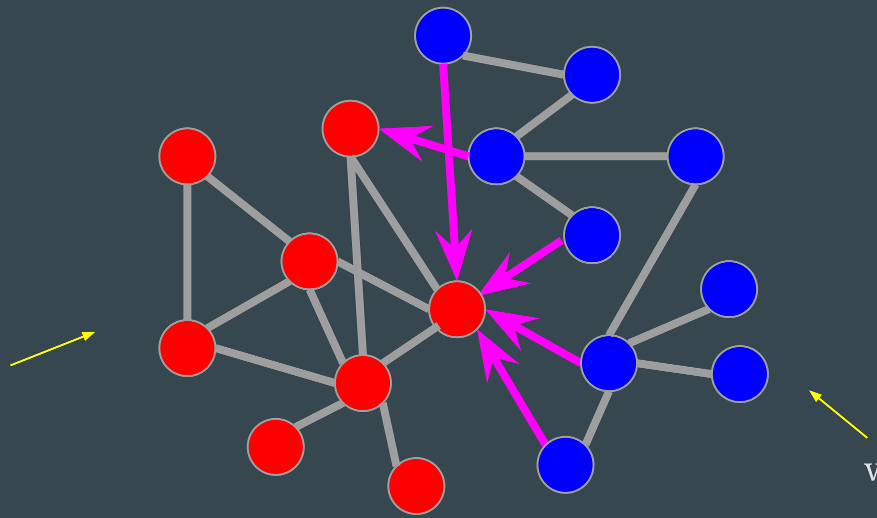


vertices with last bit **0**

vertices with last bit **1**

1 graph with **16** vertices

Ruling sets:



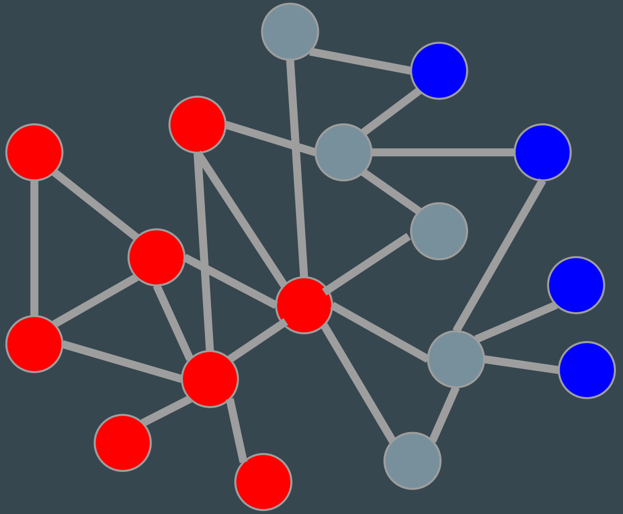
vertices with last bit **0**

vertices with last bit **1**

1 graph with **16** vertices

Ruling sets:

vertices with last bit **0**



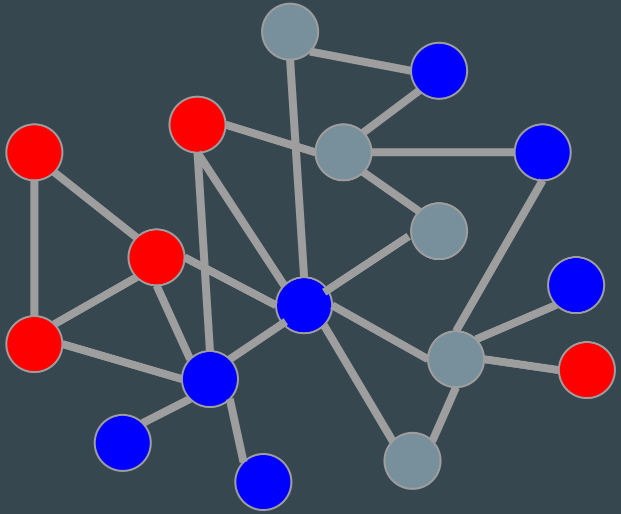
vertices with last bit **1**

Red and blue vertices are now separated!

2 graphs with ≤ **8** vertices

Ruling sets:

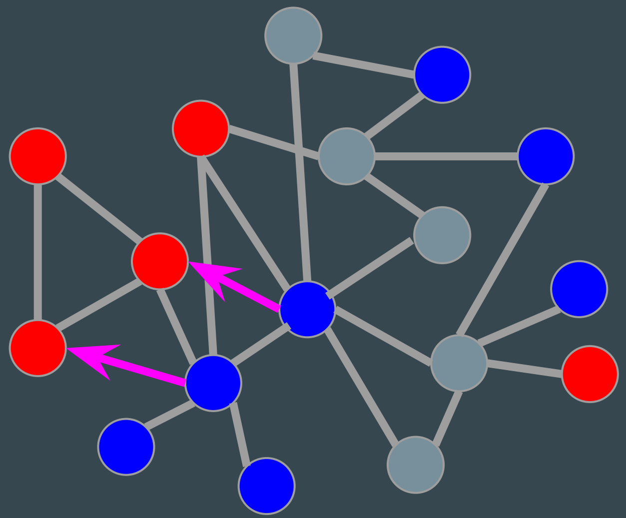
Now let's look at the penultimate bit:



2 graphs with ≤ **8** vertices

Ruling sets:

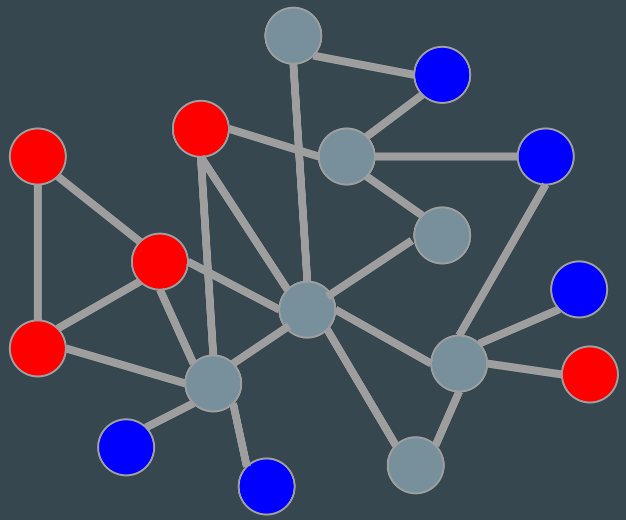
Now let's look at the penultimate bit:



2 graphs with ≤ **8** vertices

Ruling sets:

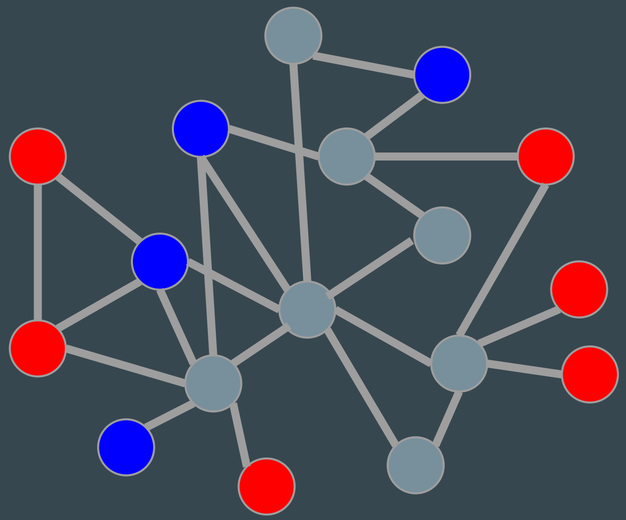
Now let’s look at the penultimate bit:



4 graphs with ≤ 4 vertices

Ruling sets:

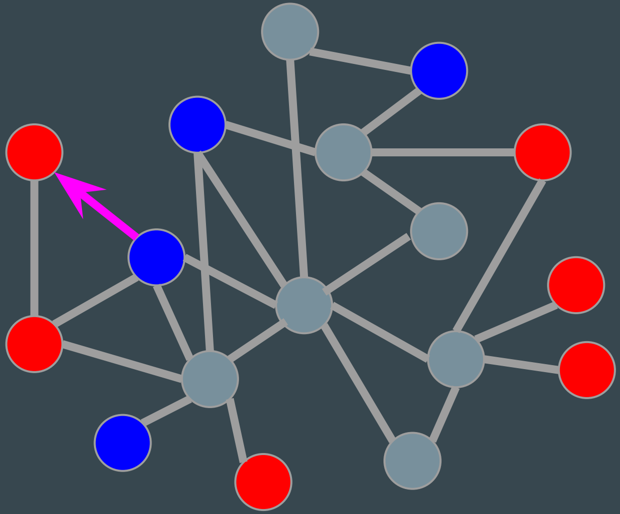
And we go on:



4 graphs with ≤ 4 vertices

Ruling sets:

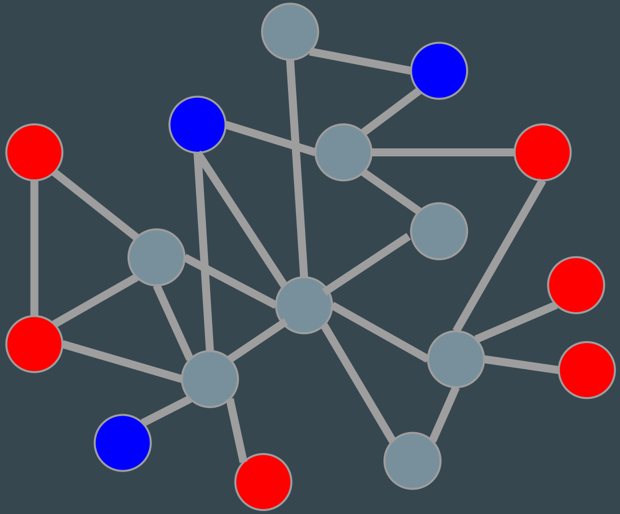
And we go on:



4 graphs with ≤ 4 vertices

Ruling sets:

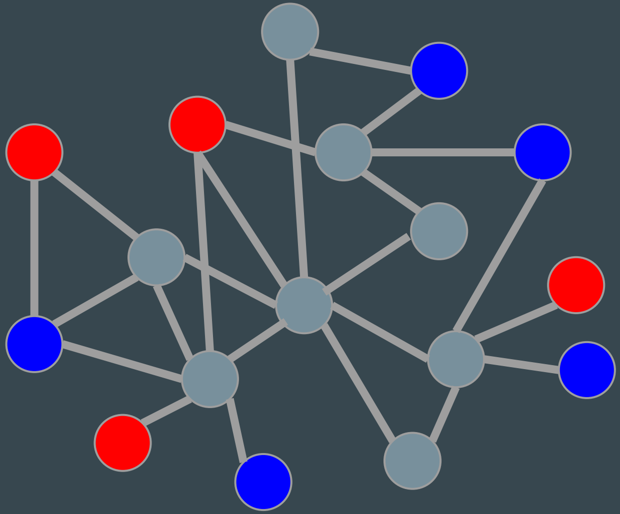
And we go on:



8 graphs with ≤ **2** vertices

Ruling sets:

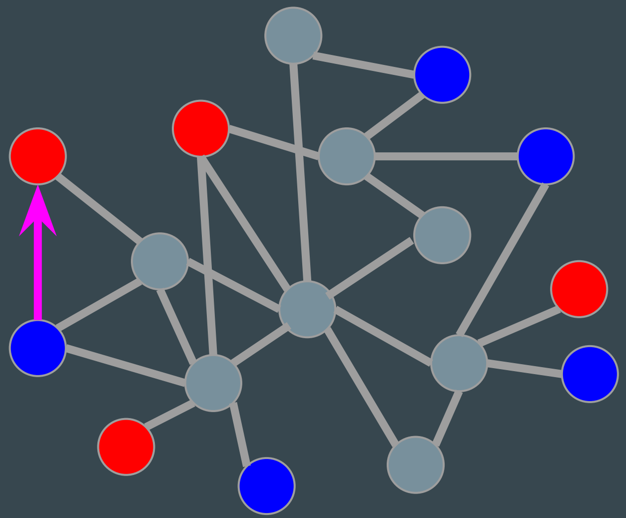
And we go on:



8 graphs with **≤ 2** vertices

Ruling sets:

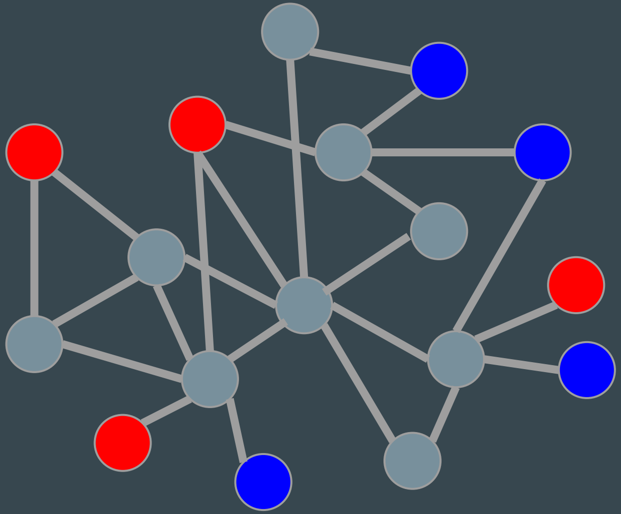
And we go on:



8 graphs with **≤ 2** vertices

Ruling sets:

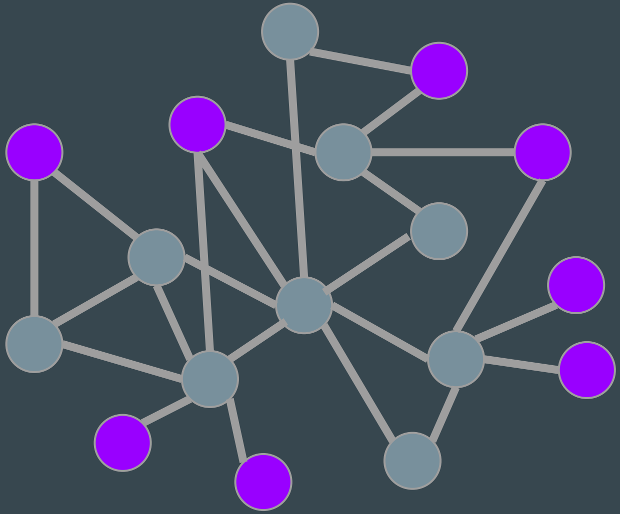
And we go on:



Done!

Ruling sets:

Final set:

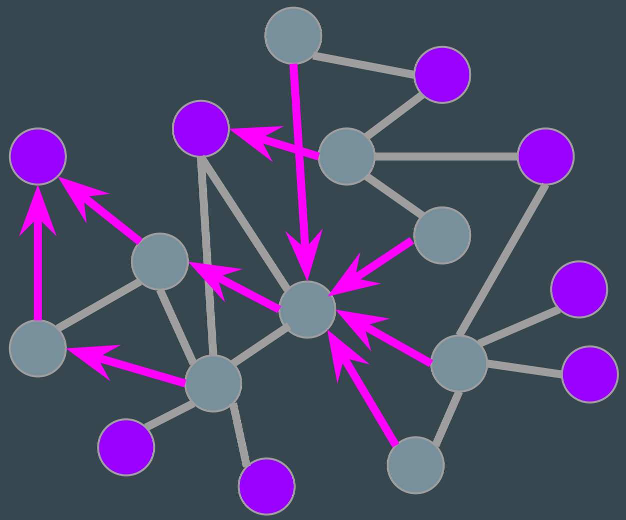


Done!

Ruling sets:

Final set:

Let's trace the algorithm to understand, why all vertices are close to some node in the set.



Done!

Network decomposition:

Distributed network decomposition = ruling set algorithm + sequential network decomposition

Ruling set:

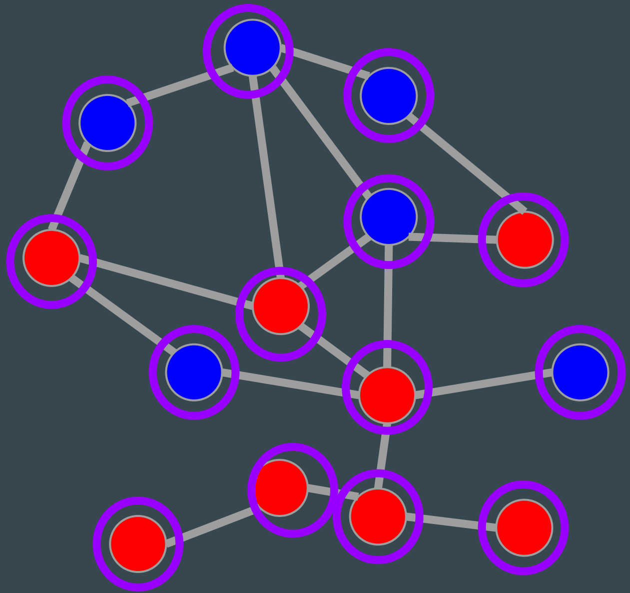
1) independent set of vertices
2) deleted vertices are still close to some node in the set
3) After the i-th phase no two vertices in the set with the same i-th bit are neighbouring

Network decomposition:

1) independent set of small-diameter cluster
2) at most half of the vertices is deleted
3) After the i-th phase no two clusters with the same i-th bit are neighbouring

Network decomposition:

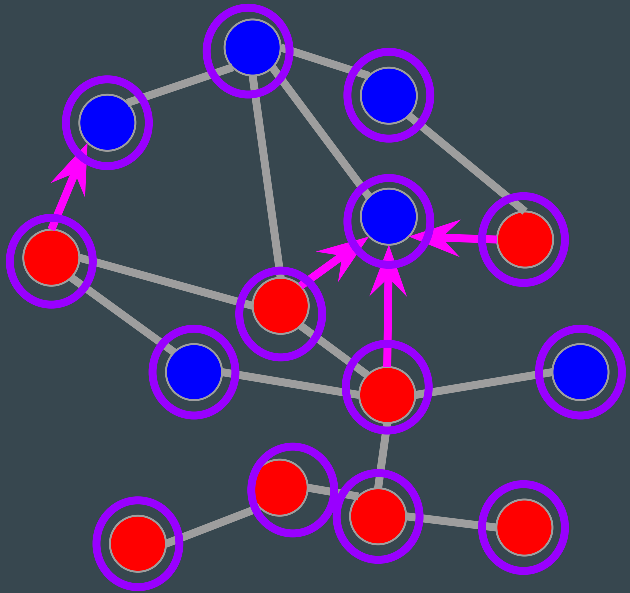
Distributed network decomposition = ruling set algorithm + sequential network decomposition



1st bit $\Rightarrow$ 1 graph

Network decomposition:

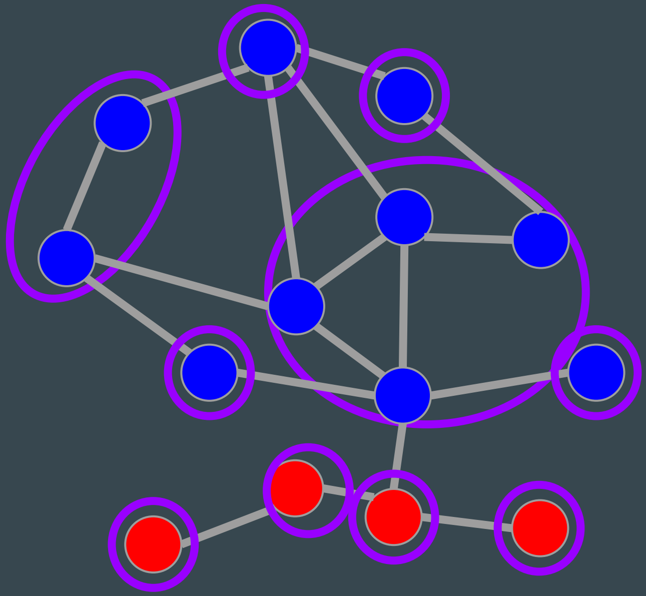
Distributed network decomposition = ruling set algorithm + sequential network decomposition



1st bit ⇒ 1 graph

Network decomposition:

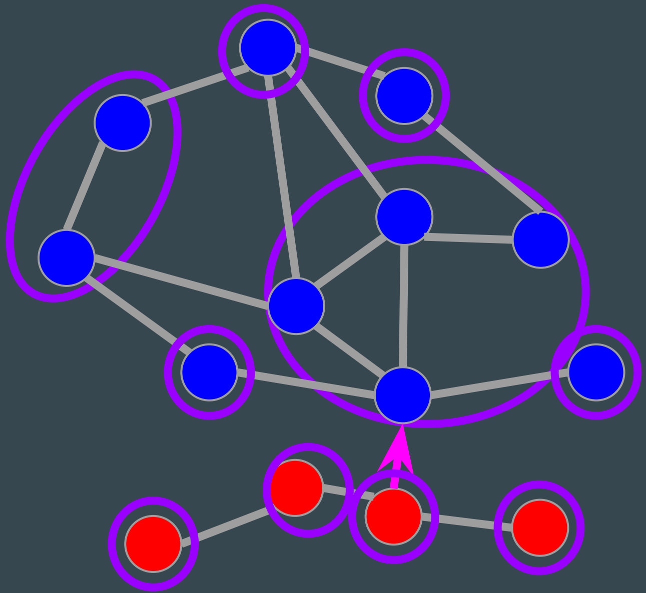
Distributed network decomposition = ruling set algorithm + sequential network decomposition



1st bit ⇒ 1 graph

Network decomposition:

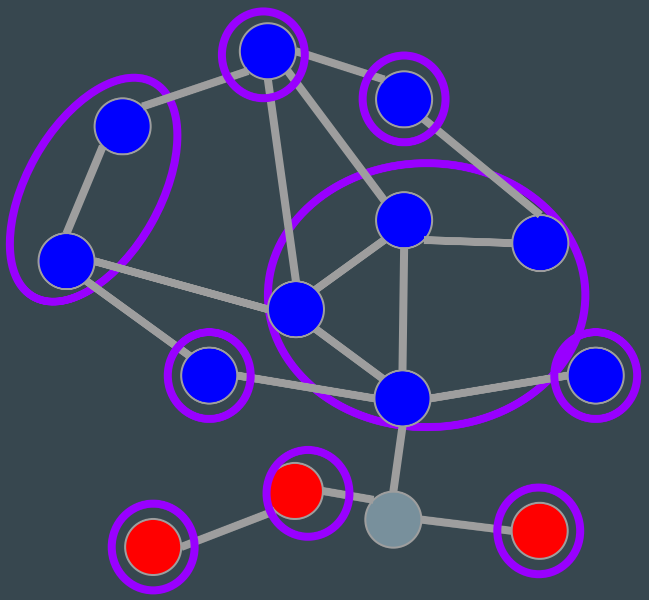
Distributed network decomposition = ruling set algorithm + sequential network decomposition



1st bit ⇒ 1 graph

Network decomposition:

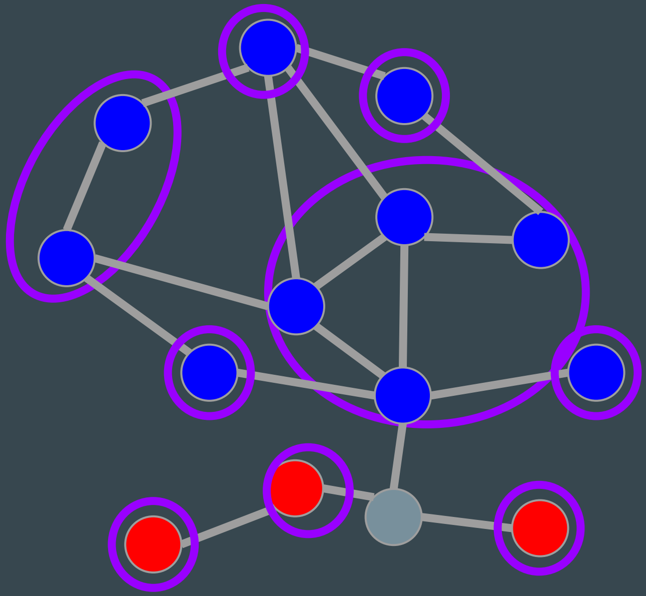
Distributed network decomposition = ruling set algorithm + sequential network decomposition



1st bit ⇒ 1 graph

Network decomposition:

Distributed network decomposition = ruling set algorithm + sequential network decomposition

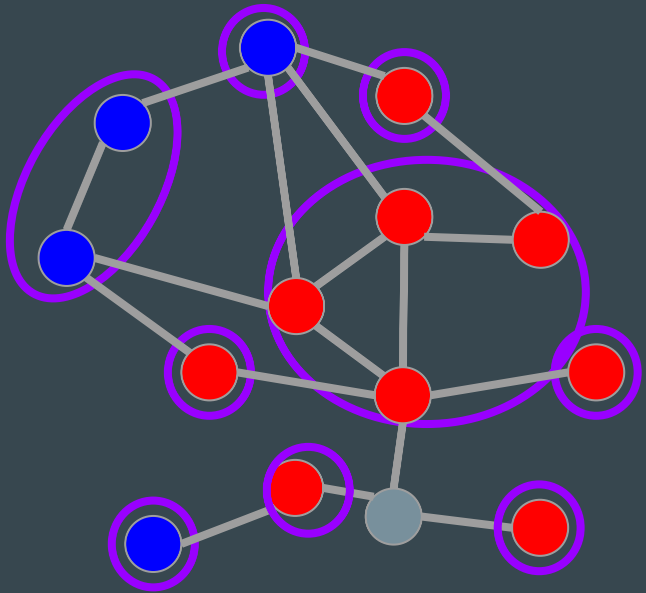


This whole process correspond to the first step of the ruling step algorithm.
We managed to disconnect red and blue vertices, while not destroying structure of clusters by much!

2 graphs

Network decomposition:

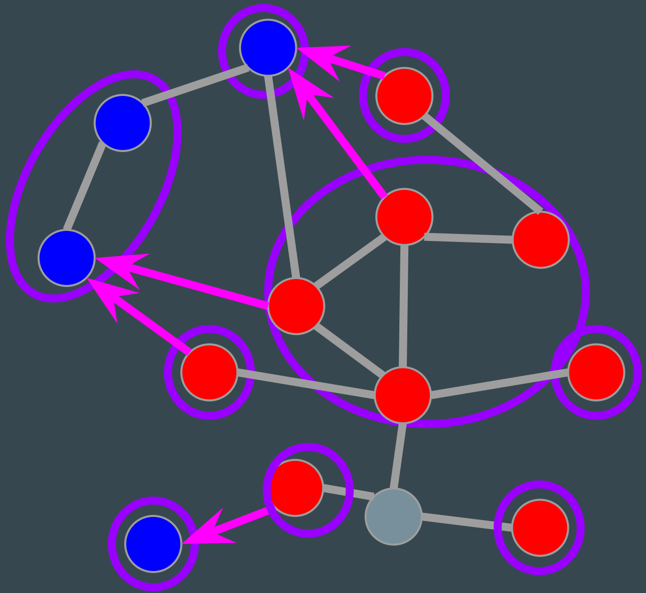
Distributed network decomposition = ruling set algorithm + sequential network decomposition



2nd bit ⇒ **2** graphs

Network decomposition:

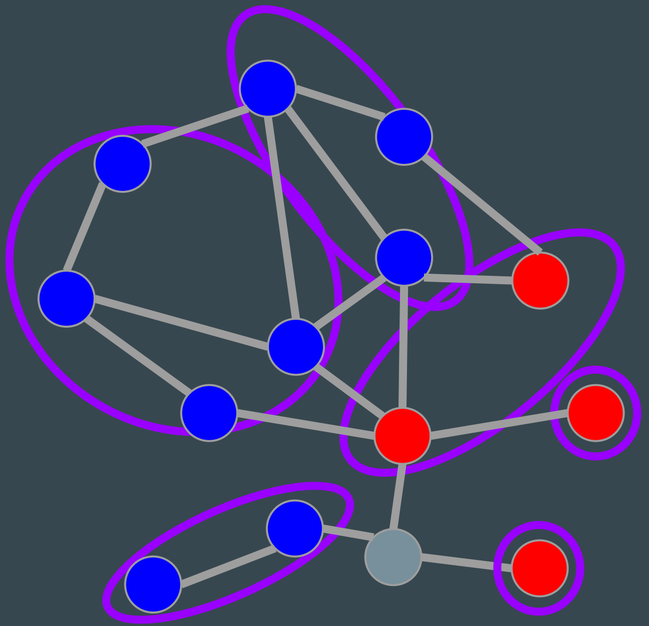
Distributed network decomposition = ruling set algorithm + sequential network decomposition



2nd bit ⇒ **2** graphs

Network decomposition:

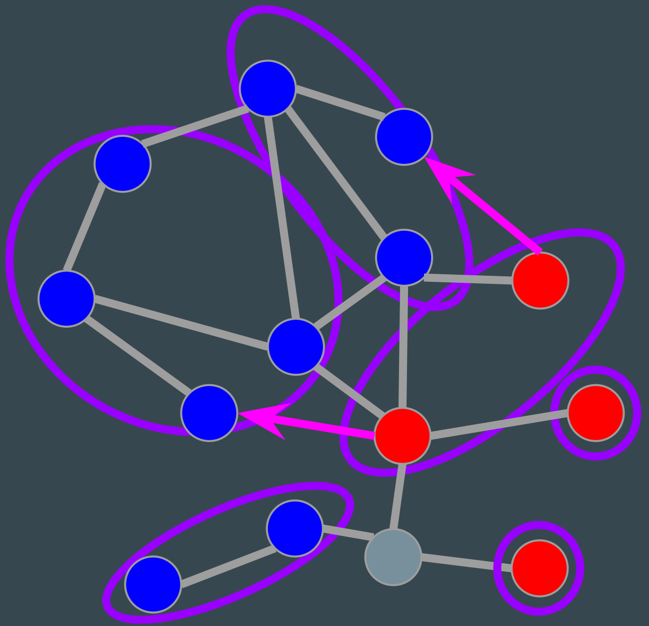
Distributed network decomposition = ruling set algorithm + sequential network decomposition



2nd bit ⇒ **2** graphs

Network decomposition:

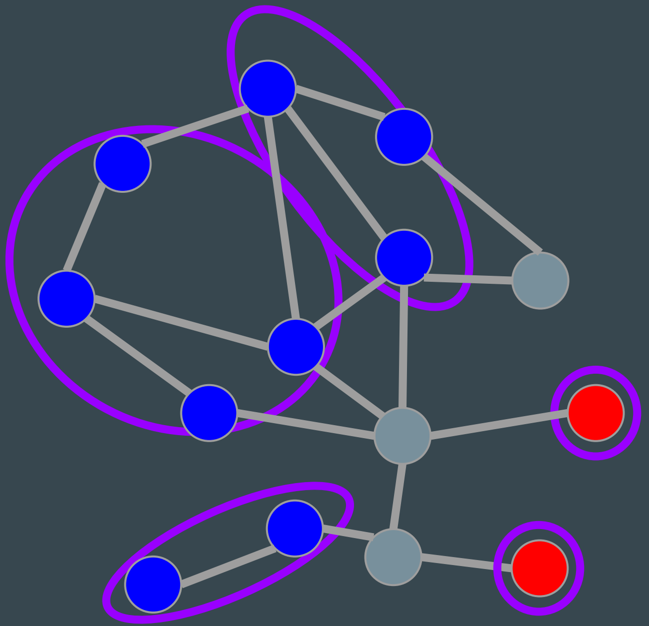
Distributed network decomposition = ruling set algorithm + sequential network decomposition



2nd bit ⇒ **2** graphs

Network decomposition:

Distributed network decomposition = ruling set algorithm + sequential network decomposition

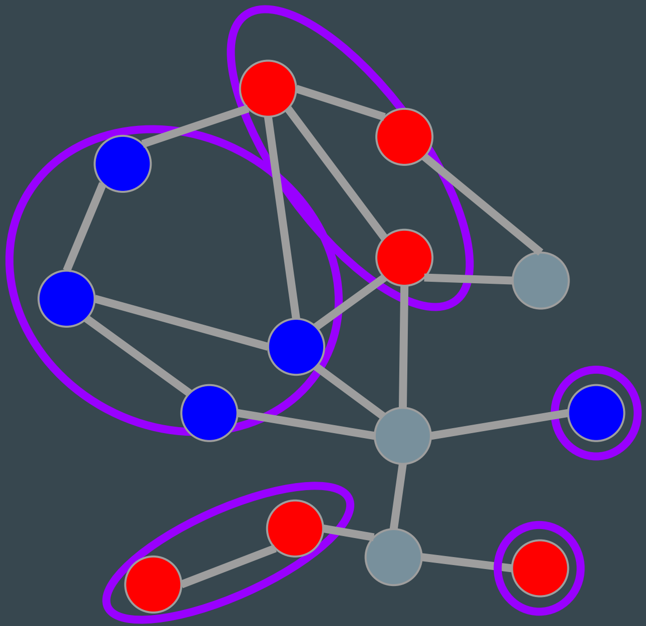


With second bit, we managed to split even further into **four** components!

4 graphs

Network decomposition:

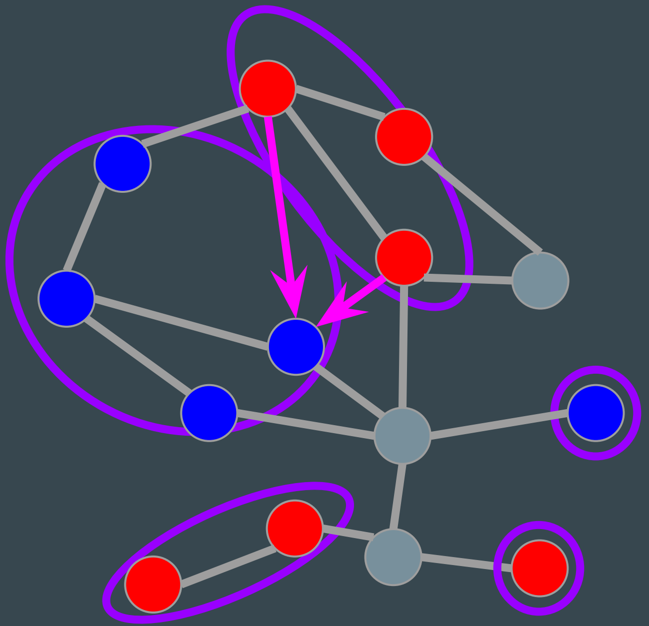
Distributed network decomposition = ruling set algorithm + sequential network decomposition



3rd bit ⇒ 4 graphs

Network decomposition:

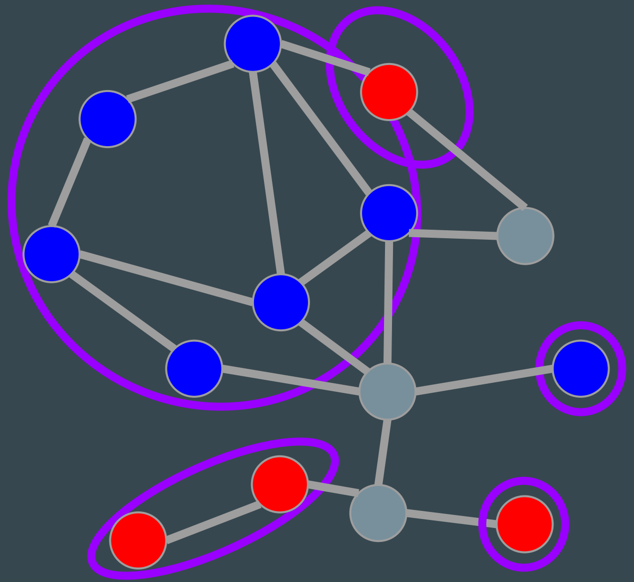
Distributed network decomposition = ruling set algorithm + sequential network decomposition



3rd bit $\Rightarrow$ 4 graphs

Network decomposition:

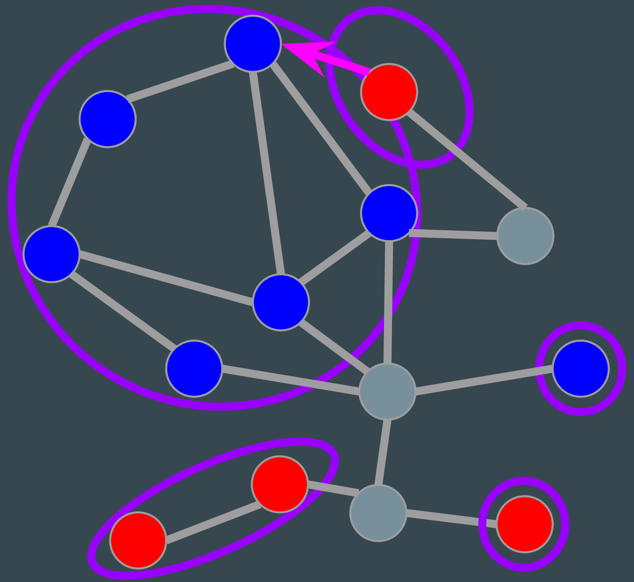
Distributed network decomposition = ruling set algorithm + sequential network decomposition



3rd bit $\Rightarrow$ 4 graphs

Network decomposition:

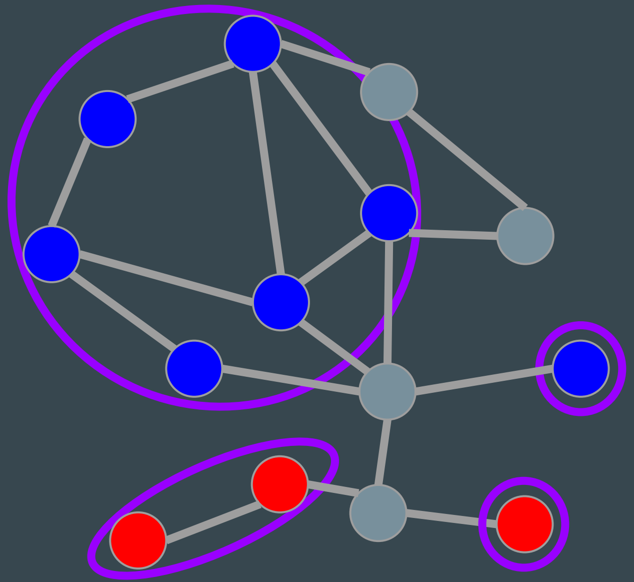
Distributed network decomposition = ruling set algorithm + sequential network decomposition



3rd bit ⇒ 4 graphs

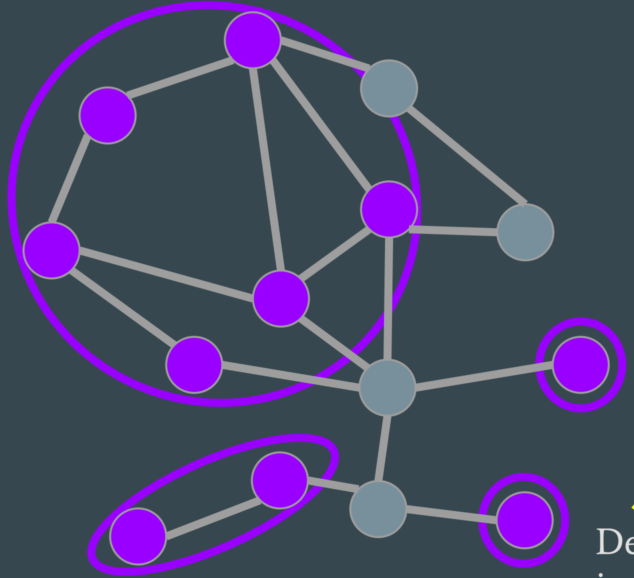
Network decomposition:

Distributed network decomposition = ruling set algorithm + sequential network decomposition



3rd bit $\Rightarrow$ 4 graphs

Network decomposition:



Since we have log n phases in total, we can delete at most O(1/log n) fraction of vertices in one phase. .

Hence, in one step, we either delete boundary, or grow by multiplicative factor 1+O(1/log n).

This implies round complexity of

$$\log^7 n = \log n \ * \ \log n \ * \ \log^2 n \ * \ \log^3 n$$

Number of bits we iterate through

In one step of the algorithm we have to collect information from the boundary of each cluster

Described algorithm computes just first colour class

Number of steps until one cluster grows to size of the whole graph

Outlook: CONGEST model

LOCAL model allows us to 'cheat'; often, we moreover require that messages sent are of logarithmic size (CONGEST model)

CONGEST adds several challenges, but it turns out that we can handle them due to the simplicity of the algorithm.

This has still further implications: e.g. polylogarithmic-round deterministic algorithm for maximal independent set in the CONGEST model (this is not obvious!)

Utility of network decomposition even goes beyond the distributed models: e.g. the best algorithm for *Δ+1* coloring in the MPC model goes from $\sqrt{\log\log n}$ to $\mathrm{poly}(\log\log\log n)$ (and there is again conditional lower bound)