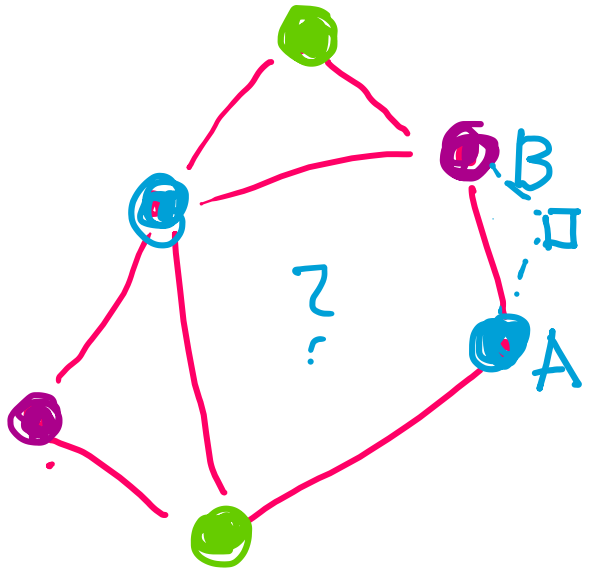
Graph coloring

Mku, PWr 2021

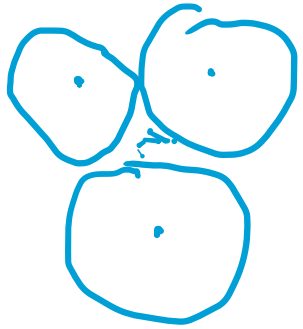
input: graph (V, E)

output: each vertex has a color
no edge with endpoints of the same color

- applications:
  1) telecom.
  2) ....
- node ID's?
  → helpful
  → choose at random
- number of colors?
  - degree
  - star graph

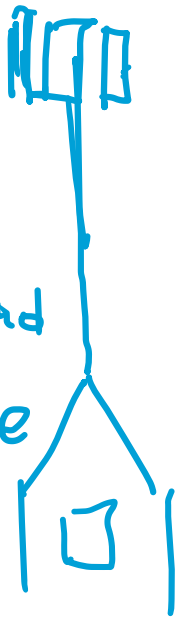


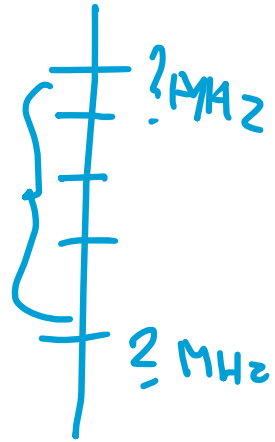
Base Station



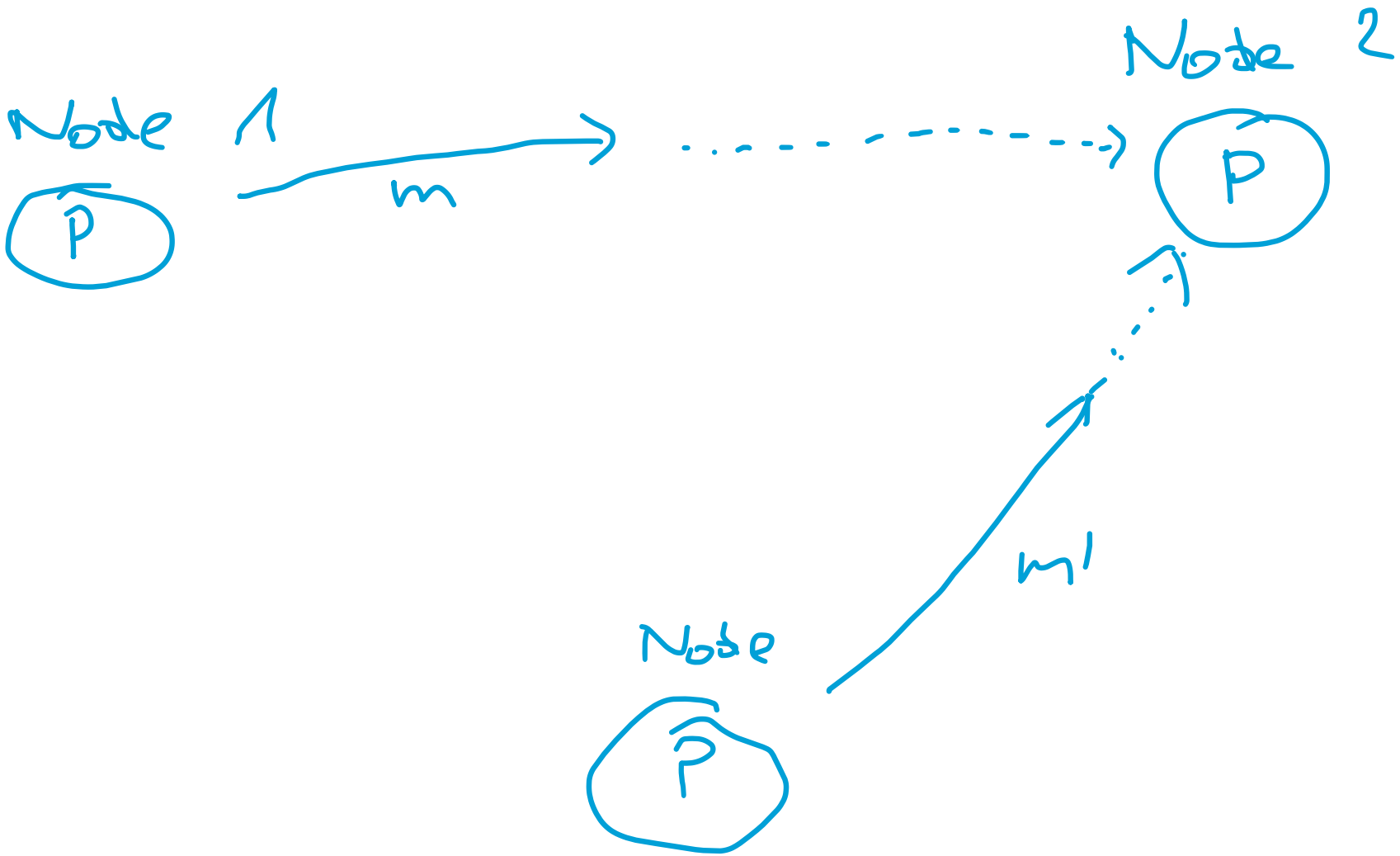
Small # of colors:

1) pay for freq. band
2) subbands possible
narrow → speed low

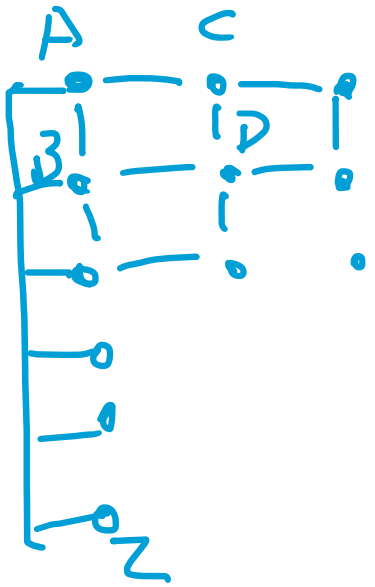




Distributed computing

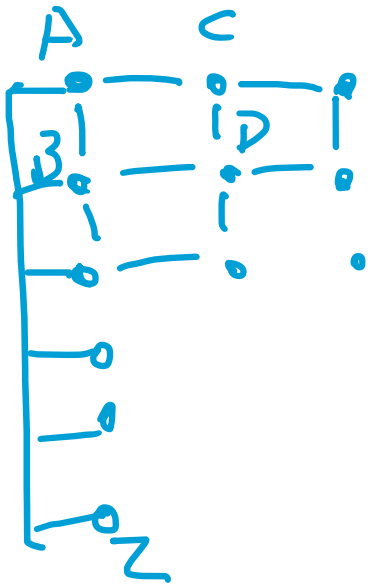


parallel comp. :
- latency low
- messages arive
- some synchrony



distributed comp.:
- latency high
- unreliable comm.
- no global clock
- nodes come & go
- Byzantine nodes - malicious, conspire

parallel comp. :
- latency low
- messages arive
- some synchrony



distributed comp.:
- latency high
- unreliable comm.
- no global clock
- nodes come & go
- Byzantine nodes - malicious, conspire

Data:

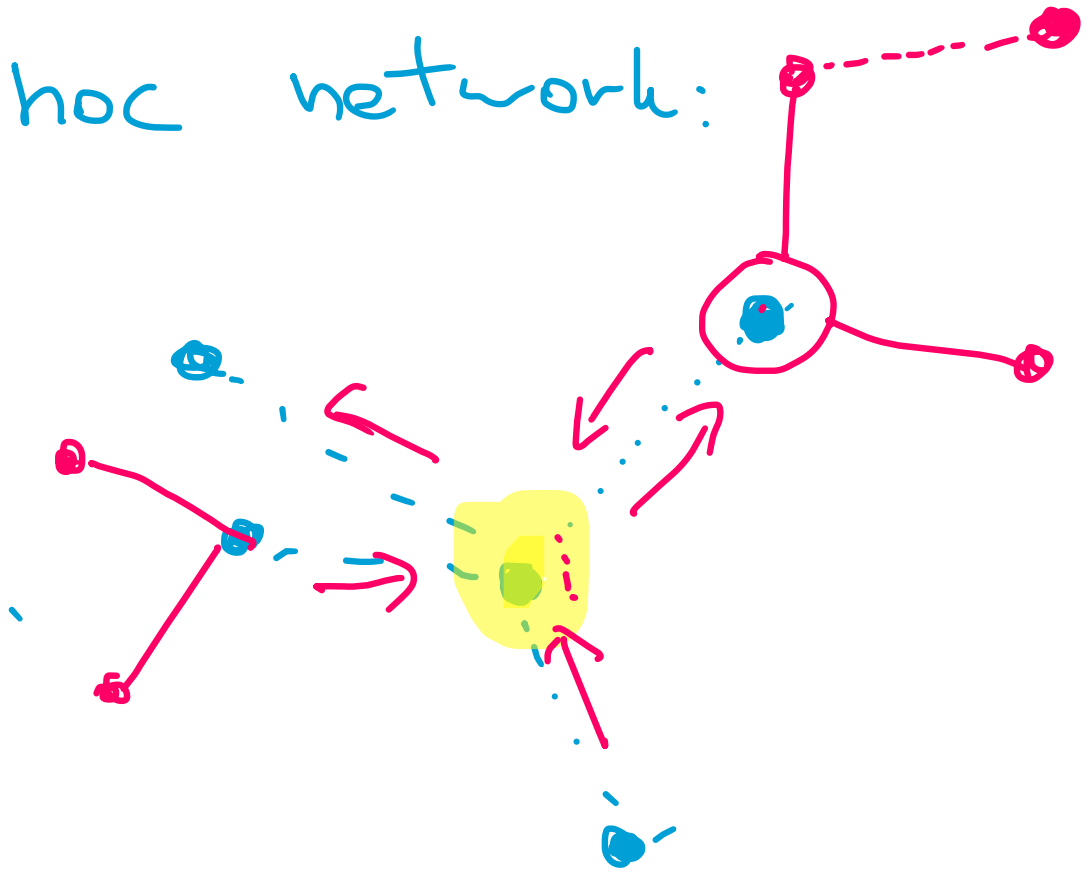
input is distributed

Ad hoc network:

Data:

input is distributed

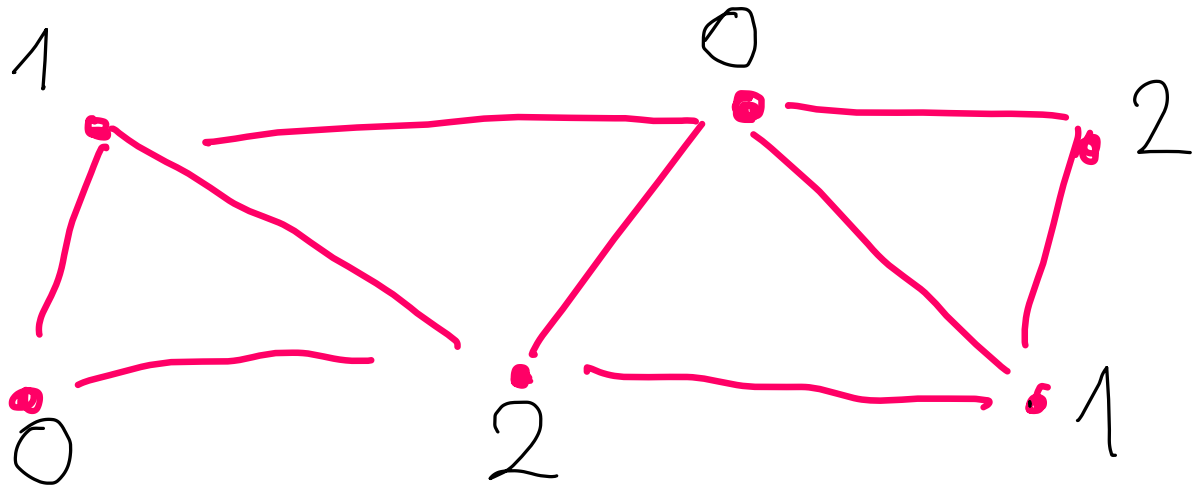
Ad hoc network:



Greedy algorithm

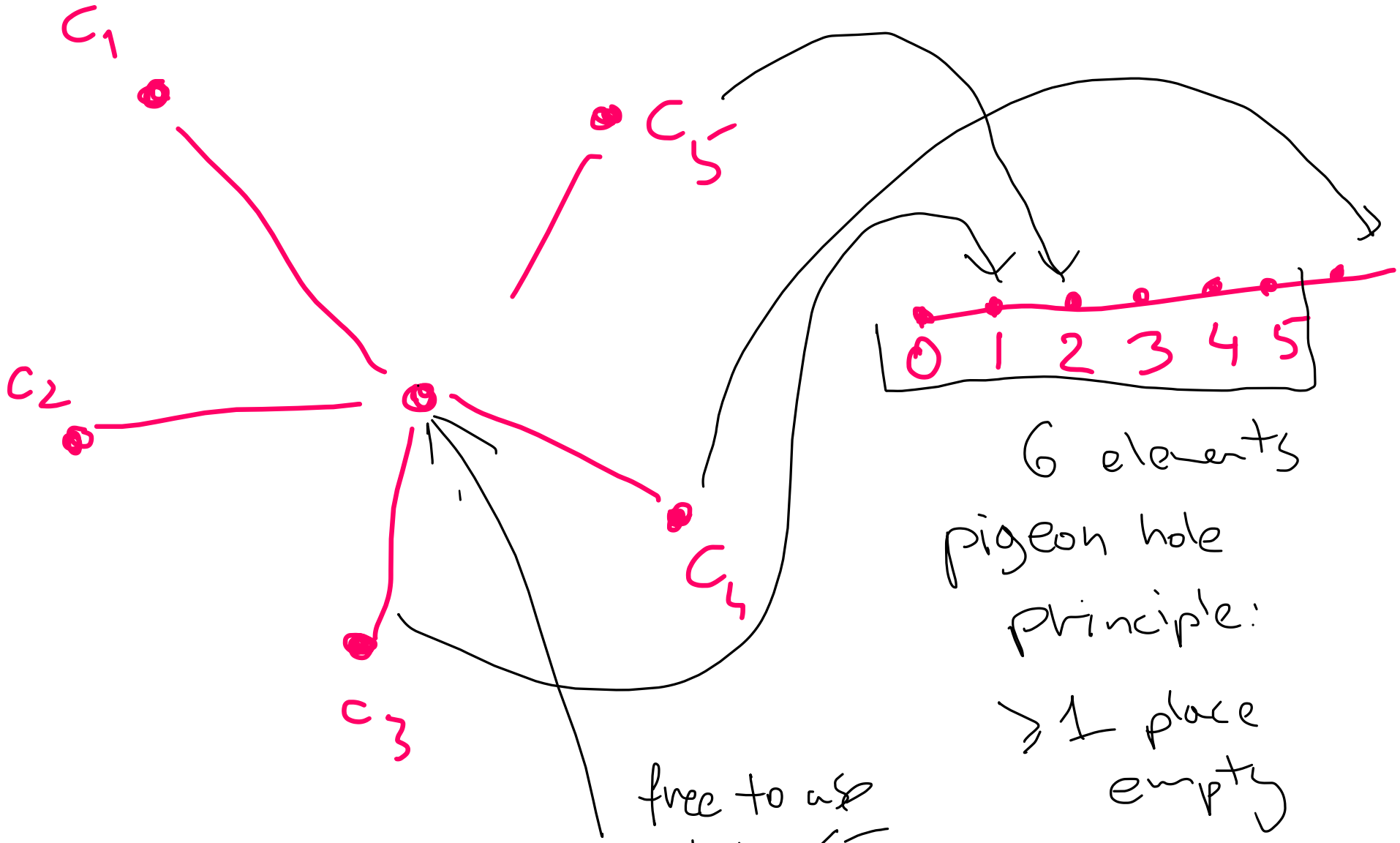
Algorithm 1.5 Greedy Sequential

1: **while** there is an uncolored vertex v **do**
2: color v with the minimal color (number) that does not conflict with the already colored neighbors
3: **end while**

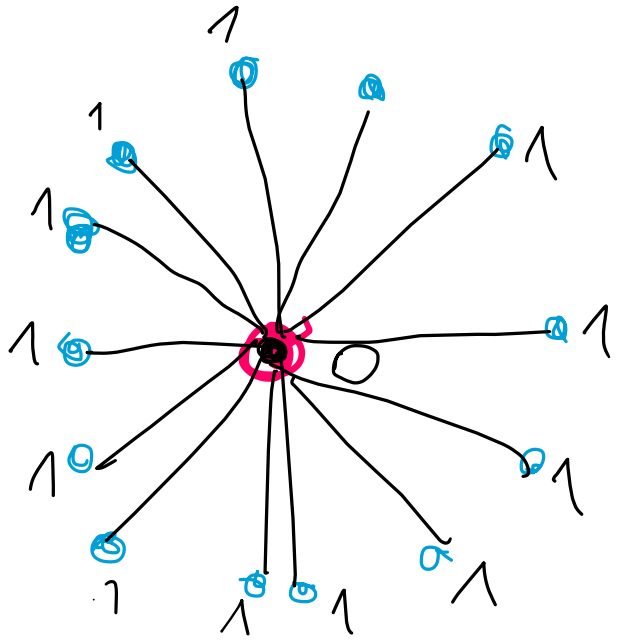


Δ = degree max

number of colors: $\leq \Delta + 1$


c_1
c_2
c_3
c_4
c_5
0 1 2 3 4 5
6 elements
pigeon hole
principle:
≥ 1 place
empty
free to use
color ≤ 5

example: star graph



Δ of center: huge

$\Delta+1$ huge

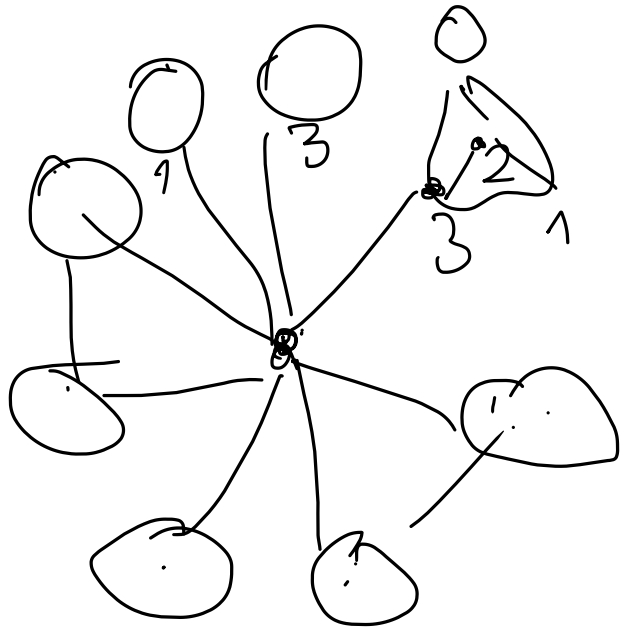
coloring: 2 colors

red in the middle

blue: elsewhere

Problem:

we do not know what to color first

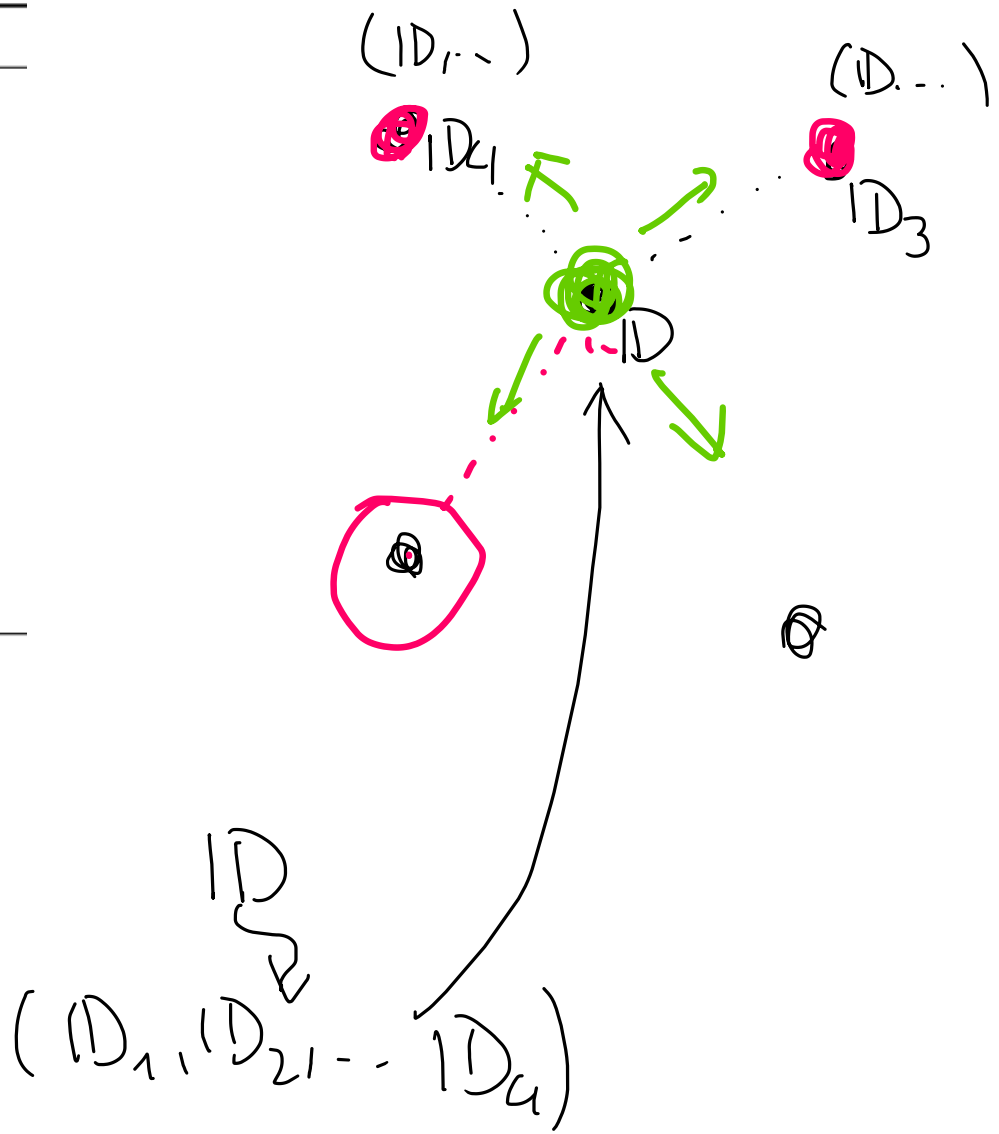

1
3
2
3
1

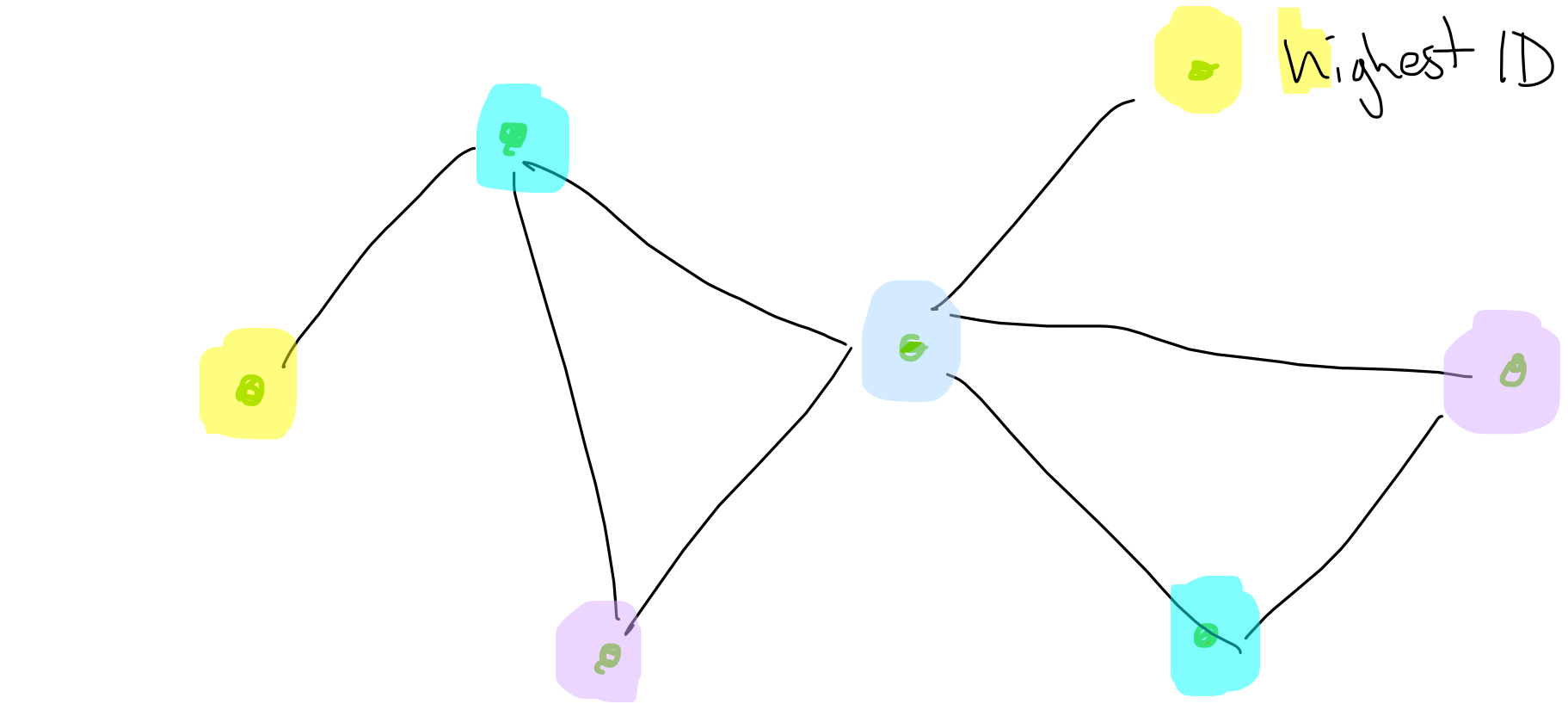
Distributed version

Algorithm 1.9 Reduce

1: Assume that initially all nodes have IDs
2: **Each node** v executes the following code:
3: node v sends its ID to all neighbors
4: node v receives IDs of neighbors
5: **while** node v has an uncolored neighbor with higher ID **do**
6: node v sends "undecided" to all neighbors
7: node v receives new decisions from neighbors
8: **end while**
9: node v chooses the smallest admissible free color
10: node v informs all its neighbors about its choice

Stop





time

0 1 2 3 4 5 6 7

rounds time: $\leq n$

n = # vertices

Time complexity:

number of steps

1) total
2) max per node

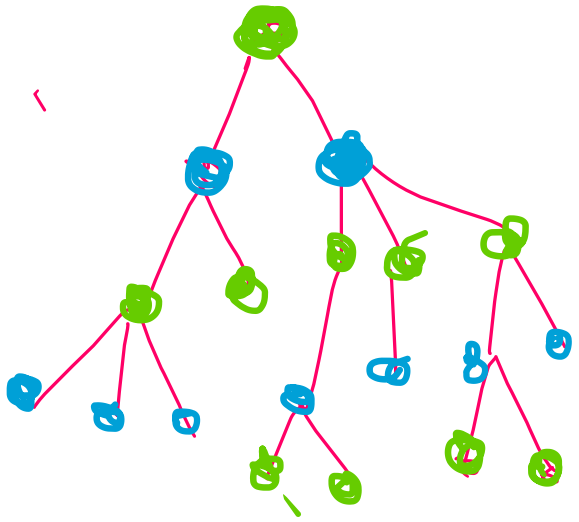
Message complexity:

keep small!

Coloring trees

Algorithm 1.14 Slow Tree Coloring

1: Color the root 0, root sends 0 to its children
2: **Each node** v concurrently executes the following code:
3: **if** node v receives a message c_p (from parent) **then**
4: node v chooses color $c_v = 1 - c_p$
5: node v sends c_v to its children (all neighbors except parent)
6: **end if**



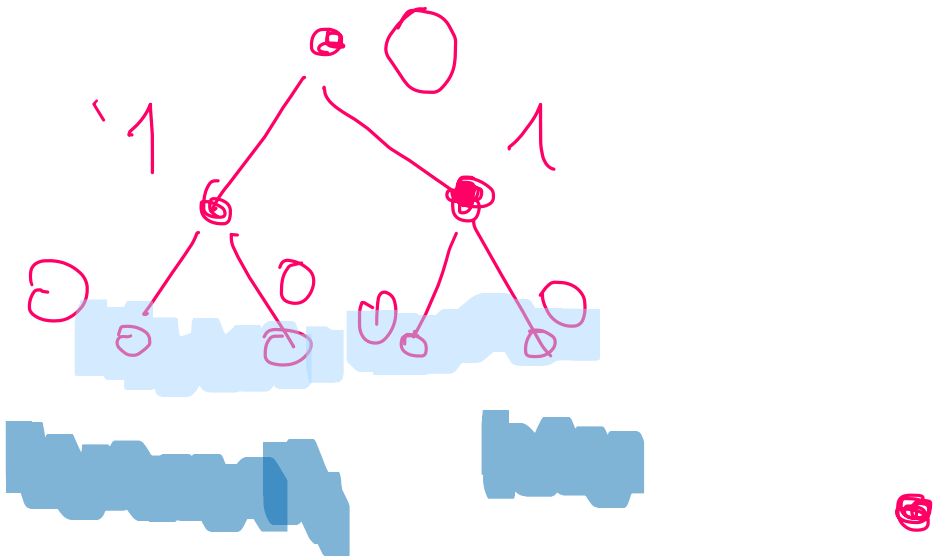
unknown topology
- distance to root
mod 2 = color

Coloring trees

Algorithm 1.14 Slow Tree Coloring

1: Color the root 0, root sends 0 to its children
2: **Each node** v concurrently executes the following code:
3: **if** node v receives a message c_p (from parent) **then**
4: node v chooses color $c_v = 1 - c_p$
5: node v sends c_v to its children (all neighbors except parent)
6: **end if**

time ≈ depth of tree



Clever $\log^*$ algorithm

$\log(\log(\log(n)))$

$\min_i \log^{(i)}(n) < 2 = \log^* n$

extremely slowly growing f-n

in practice $\leq$ const

2^{80}

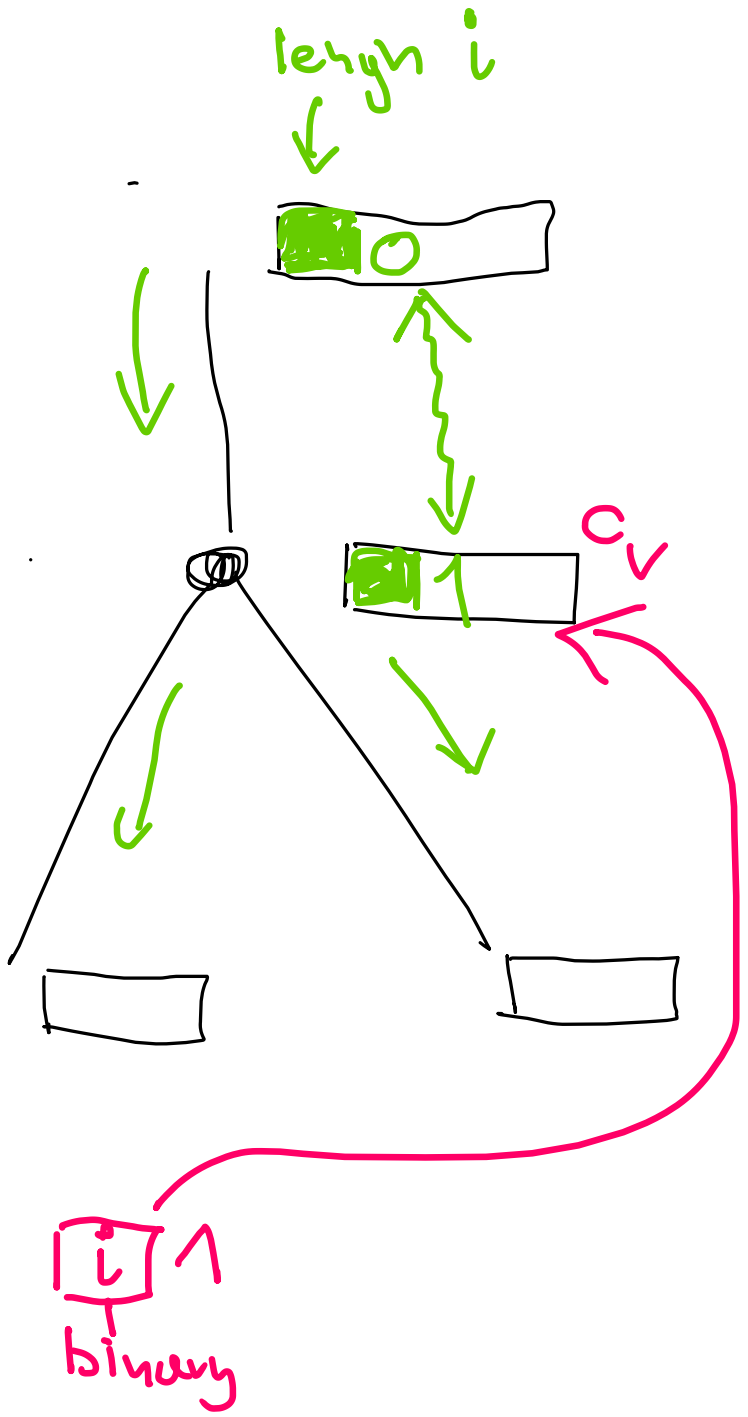
Algorithm 1.17 "6-Color"

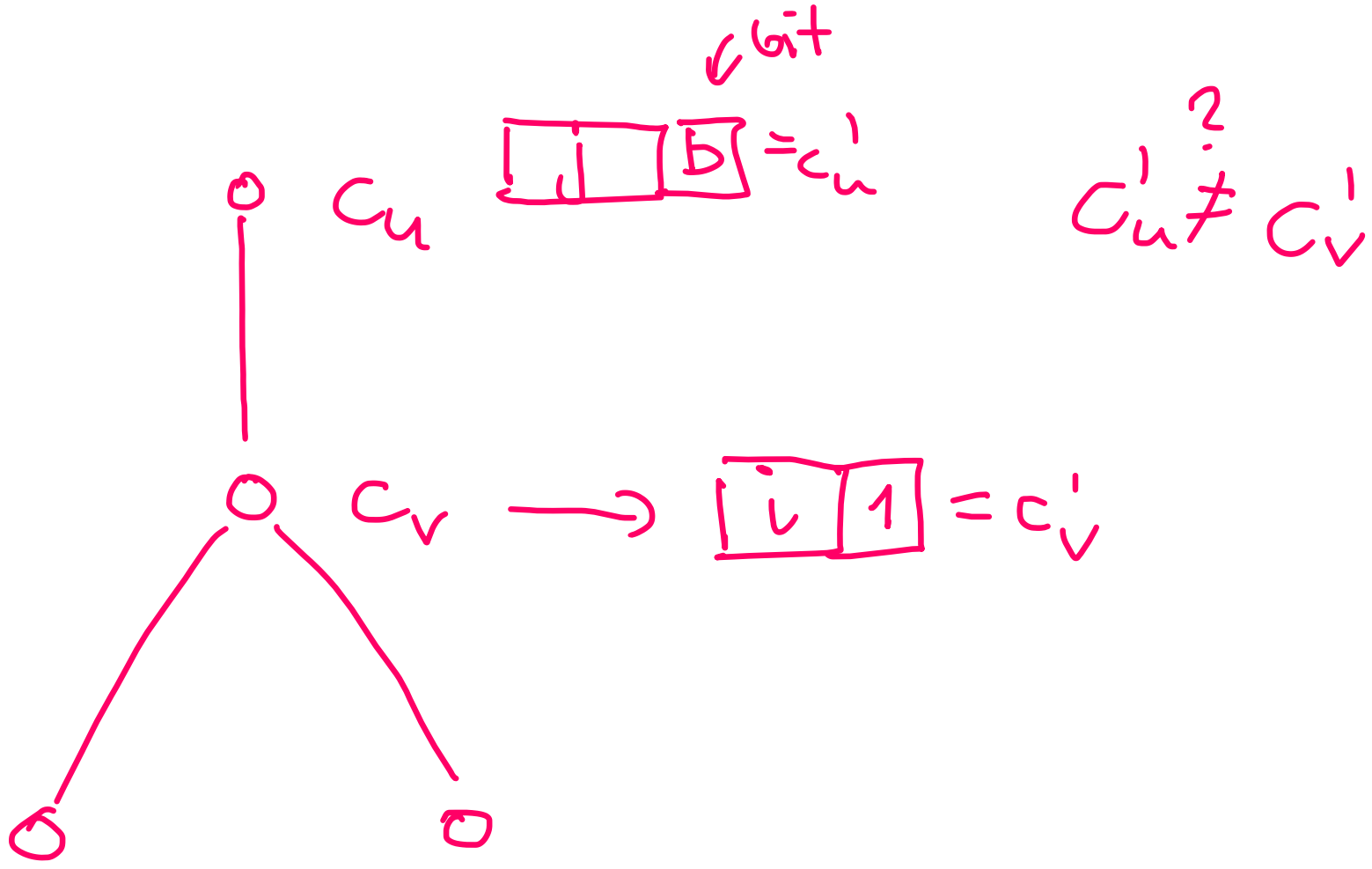
1: Assume that initially the nodes have IDs of size $\log n$ bits
2: The root assigns itself the label 0
3: **Each** other **node** v executes the following code
4: send own color c_v to all children
5: **repeat**
6: receive color c_p from parent
7: interpret c_v and c_p as bit-strings
8: let i be the index of the smallest bit where c_v and c_p differ
9: the new label is i (as bitstring) followed by the i^{th} bit of c_v
10: send c_v to all children
11: **until** $c_w \in \{0, \ldots, 5\}$ for all nodes w

Clever $\log^* n$ algorithm

Algorithm 1.17 "6-Color"

1: Assume that initially the nodes have IDs of size $\log n$ bits
2: The root assigns itself the label 0
3: **Each** other **node** v executes the following code
4: send own color c_v to all children
5: **repeat**
6: receive color c_p from parent
7: interpret c_v and c_p as bit-strings
8: let i be the index of the smallest bit where c_v and c_p differ
9: the new label is i (as bitstring) followed by the i^{th} bit of c_v
10: send c_v to all children
11: **until** $c_w \in \{0, \ldots, 5\}$ for all nodes w





1) $i \neq j$, then $(\text{length } c_u' \neq \text{length } c_v')$ ∨ (equal and diff)
ok ok

2) $i = j$ $\boxed{i \mid b}$ $\boxed{i \mid 1}$ $b = 0$, colors diff.

1D reduction

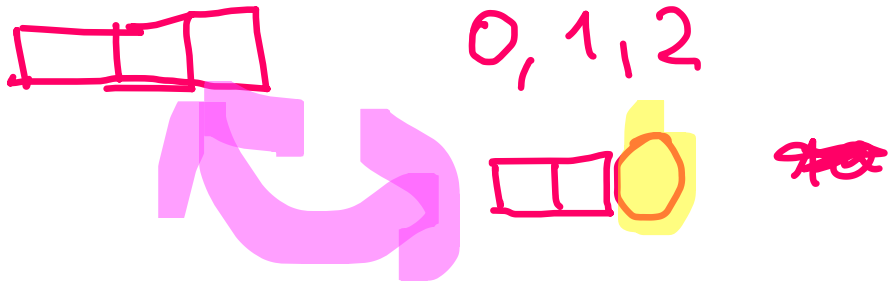
from length k

$\rightsquigarrow \lceil \log_2 k \rceil + 1$

$\approx \log^* n$

0-5 colors:

length 3 - 8 colors



time:

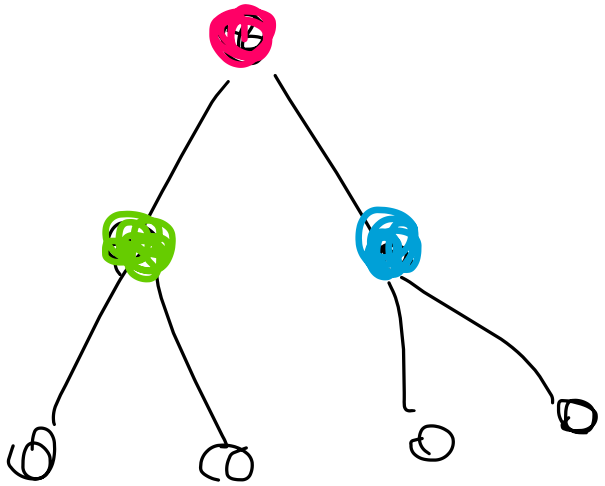
$\approx \log^*(n)$ colors $0, \ldots, 5$

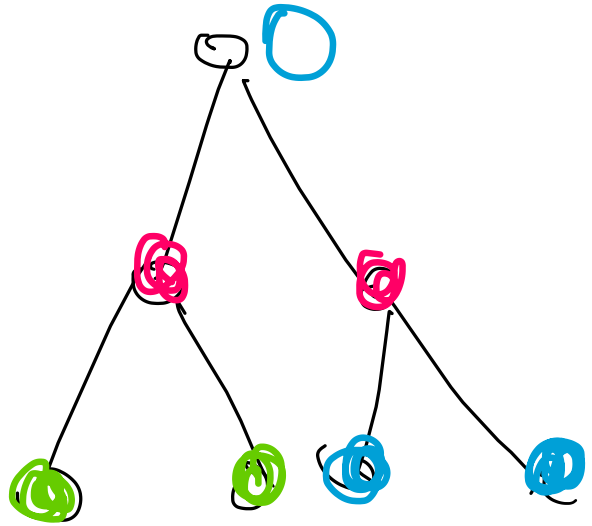
before:
time n (worst)
$\log n$ (best)

Reduction to 3 colors

Algorithm 1.19 Shift Down

1: **Each** other **node** v concurrently executes the following code:
2: Recolor v with the color of parent
3: Root chooses a new (different) color from $\{0, 1, 2\}$

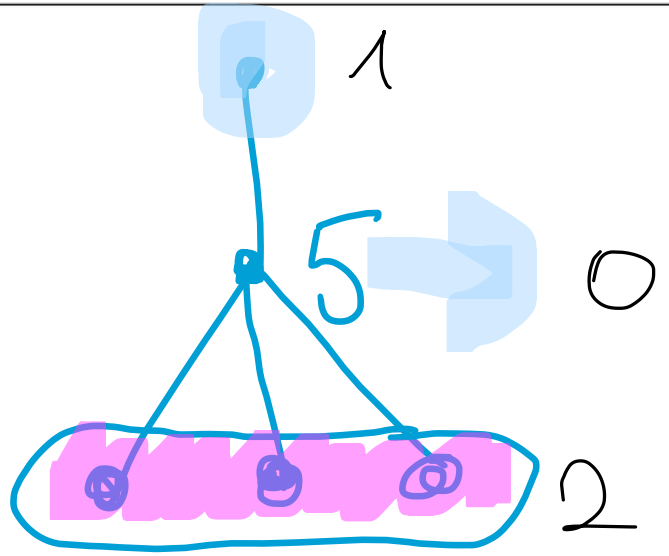




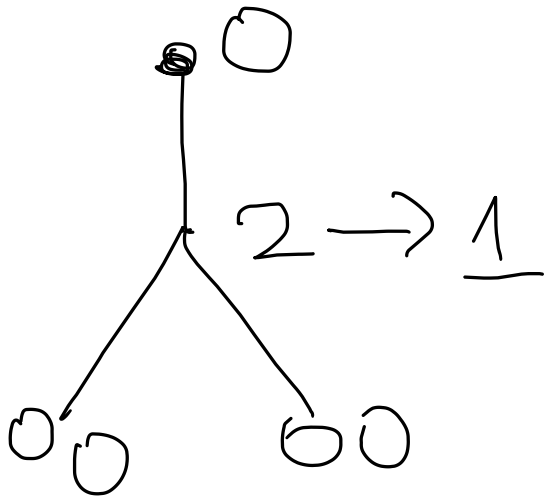
Eliminating colors 5, 4, 3

Algorithm 1.21 Six-2-Three

1: **Each node** v concurrently executes the following code:
2: **for** $x = 5, 4, 3$ **do**
3: Perform subroutine Shift down (Algorithm 1.19)
4: **if** $c_v = x$ **then**
5: choose the smallest admissible new color $c_v \in \{0, 1, 2\}$
6: **end if**
7: **end for**



few steps
local!



END