Even for cheaters who look at their cards

Valia Mitsou
Hungarian Academy of Sciences

Joint with

Jean-Francois Baffier[1,2], Man-Kwun Chiu[1,2], Yago Diez[3], Matias Korman[3], André van Renssen[1,2], Marcel Roeloffzen[1,2], and Yushi Uno[4]

1. National Institute of Informatics (NII), Tokyo.
2. JST, ERATO, Kawarabayashi Large Graph Project.
3. Tohoku University, Sendai, Japan.
4. Osaka Prefecture University.

Hanabi: characteristics

- Collaborative
- Information-sharing

Spiel des Jahres 2013 KRITIKERPREIS

Hanabi: characteristics

- Collaborative
- Information-sharing

  Previous works: different strategies so that players can share information and collectively play as many cards as possible.

Hanabi: characteristics

- Collaborative
- Information-sharing

  Previous works: different strategies so that players can share information and collectively play as many cards as possible.

This paper: *Even with one player and perfect information, the game is intractable.*

Our model

- Single player
- Perfect information
- No hints
- Slightly different set of moves than Hanabi

Our model

Cards

- v: number of values;
- c: number of colors.

Our model

- Given stream of N cards.

  (Identical cards might *repeat up to r times*)

- Moves:
  - Discard;
  - Play;
  - Store.
- Decide whether stacks 1 ... v can be formed for all c *colors* (given that at any time we can be *storing up to h cards*).

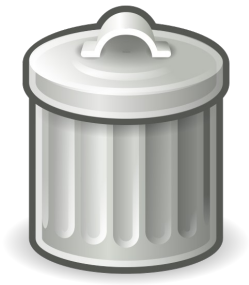
Example

$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$

3	1	2	1	2	3	2	3	1	2	3	3

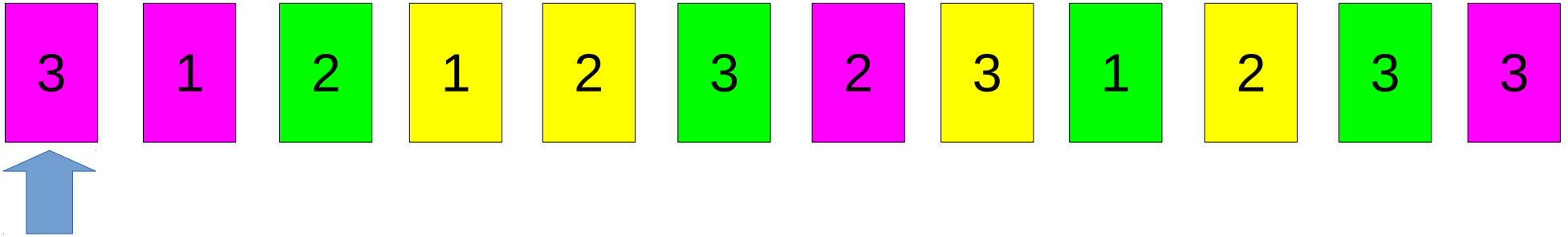


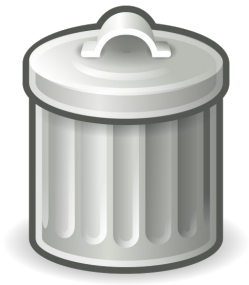
Example

$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$



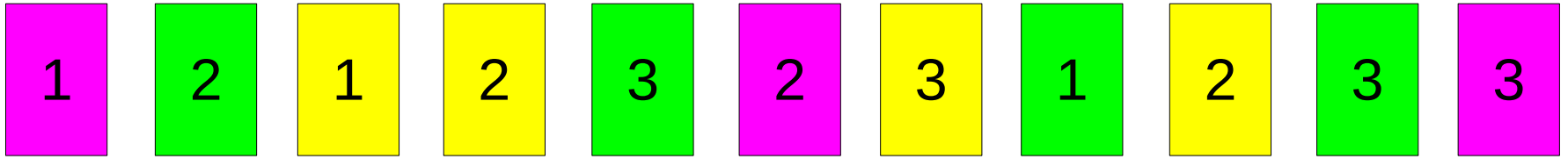


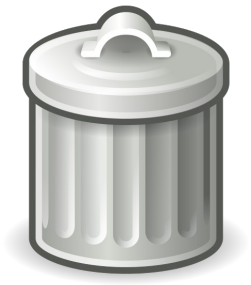
Example

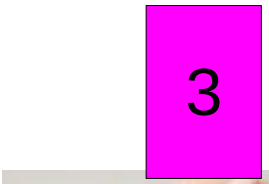
$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$





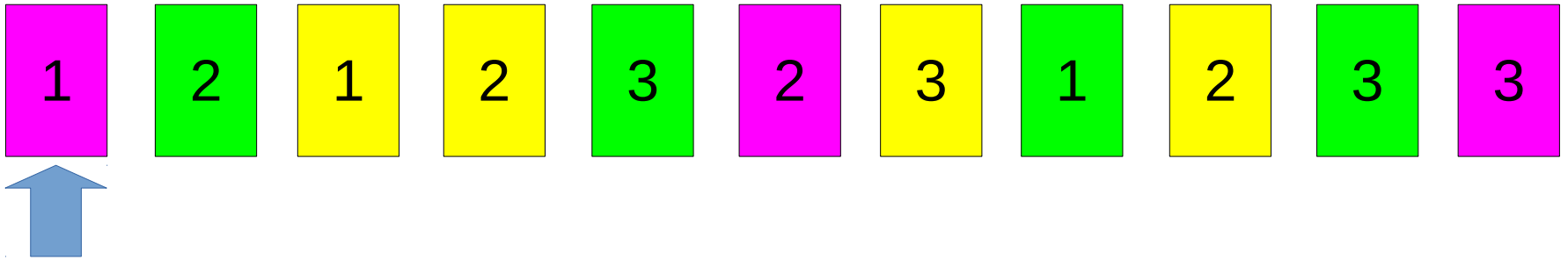


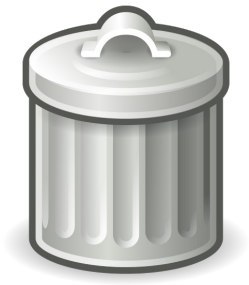
Example

$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$



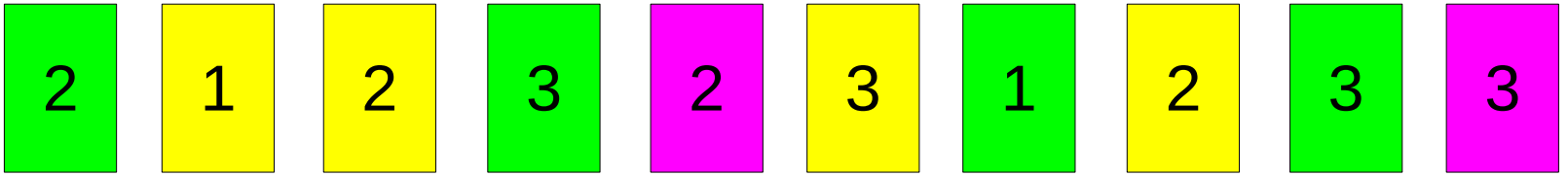


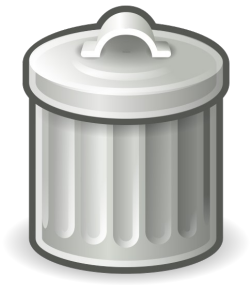
Example

$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$



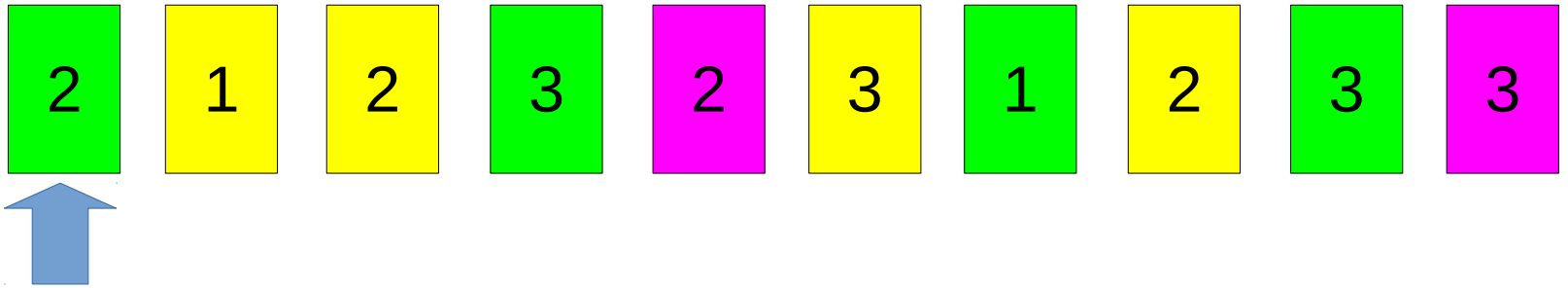


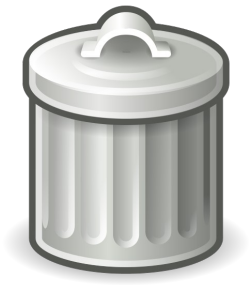
Example

$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$



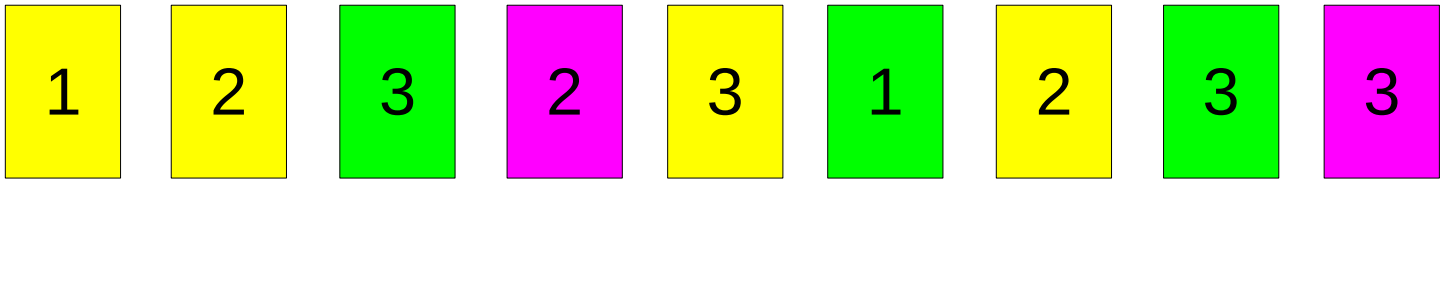


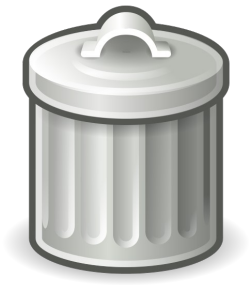
Example

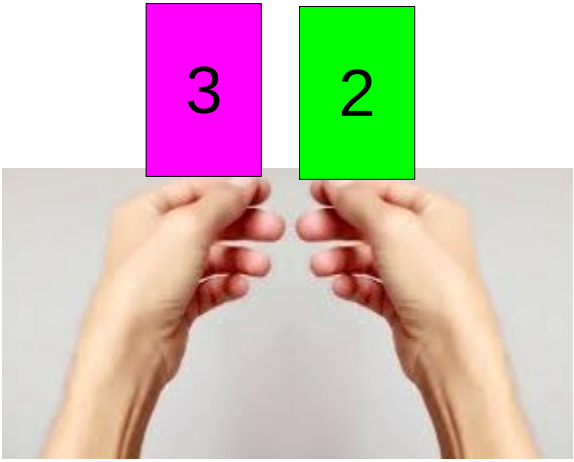
$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$





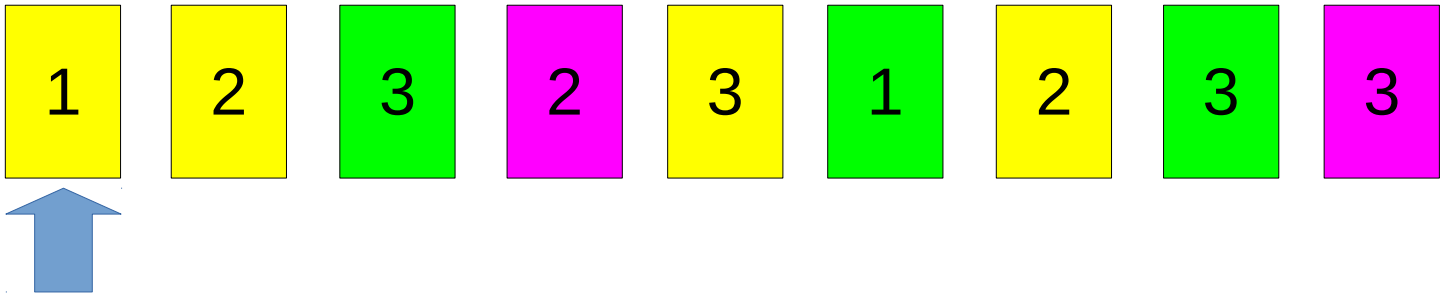


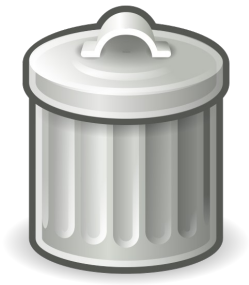
Example

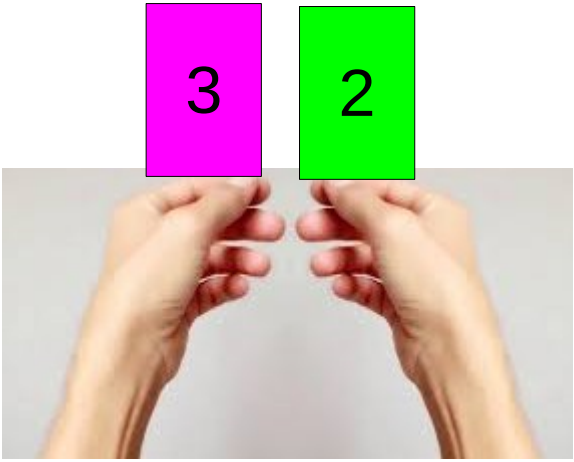
$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$





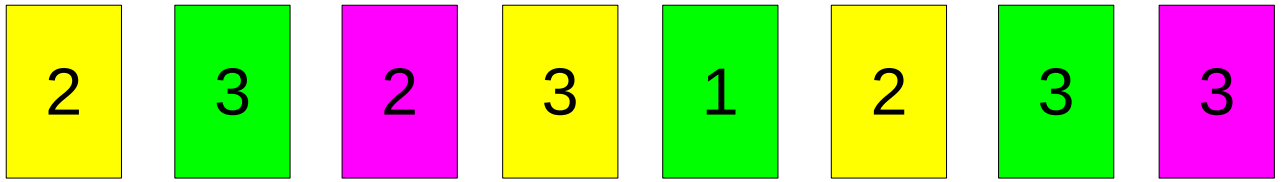


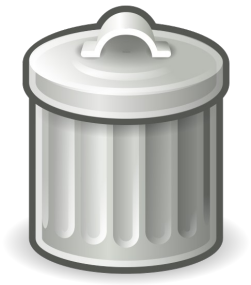
$N = 12$

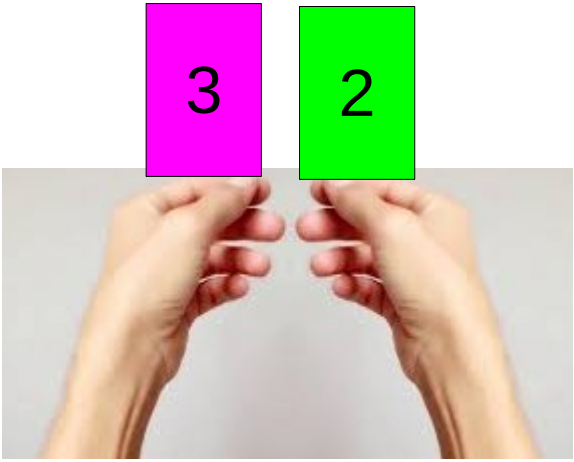
$v = 3$ $r = 2$

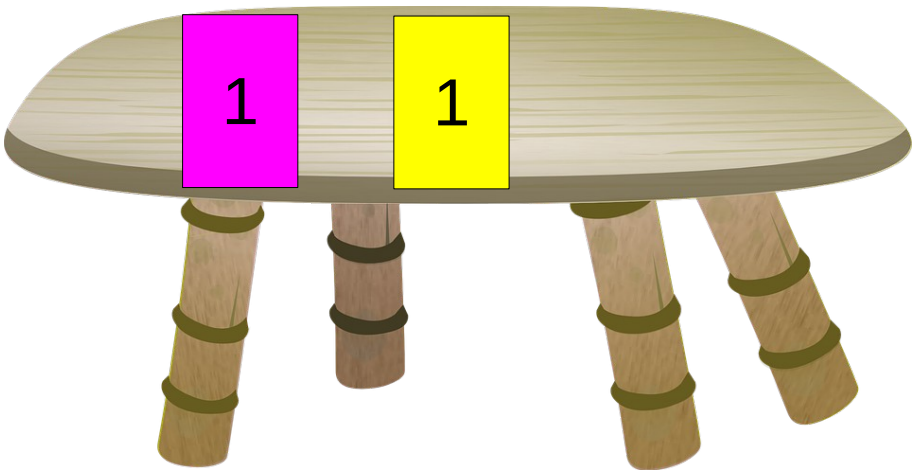
$c = 3$ $h = 2$

Example







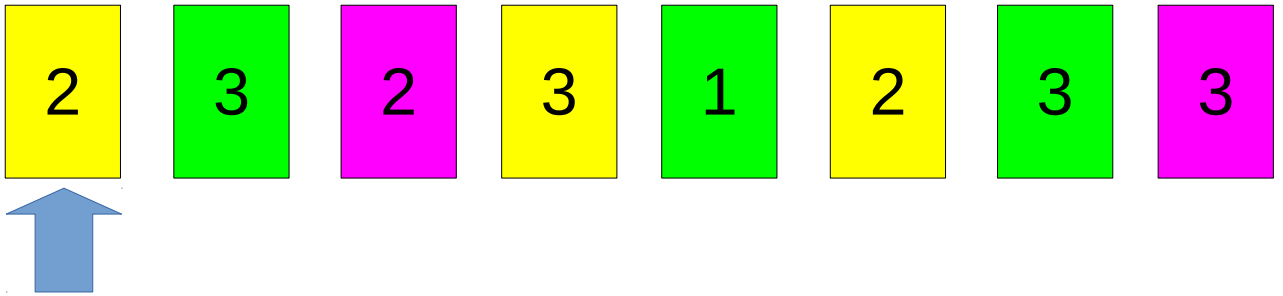


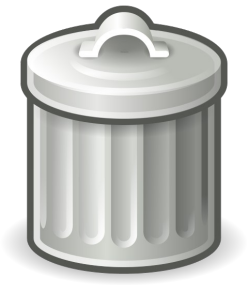
$N = 12$

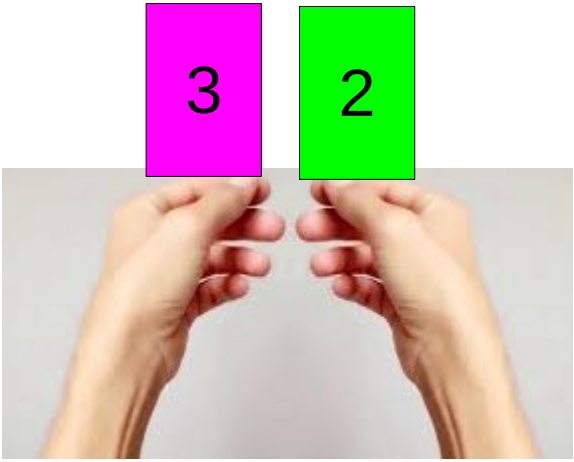
$v = 3$ $r = 2$

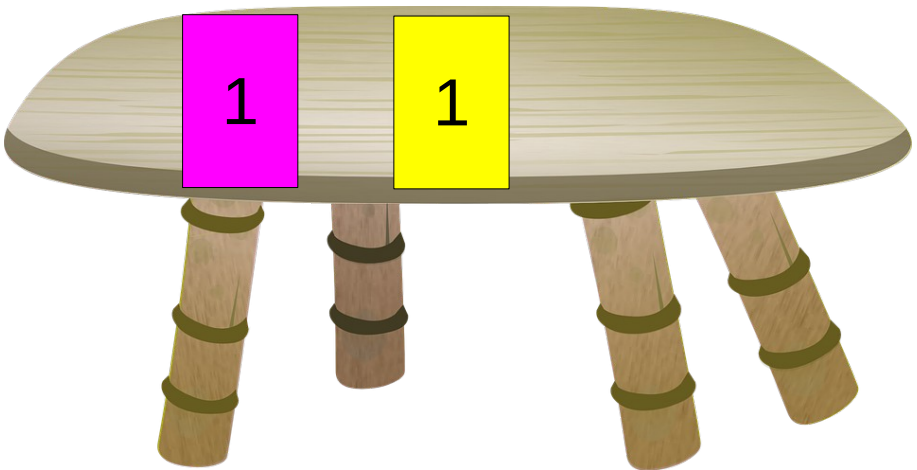
$c = 3$ $h = 2$

Example







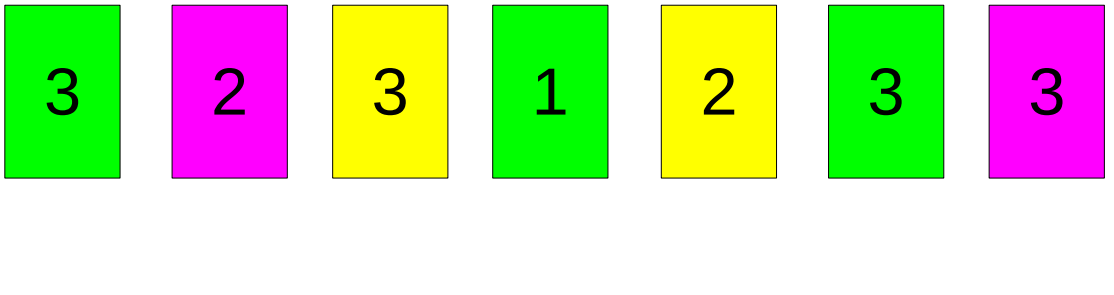


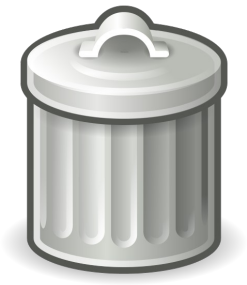
$N = 12$

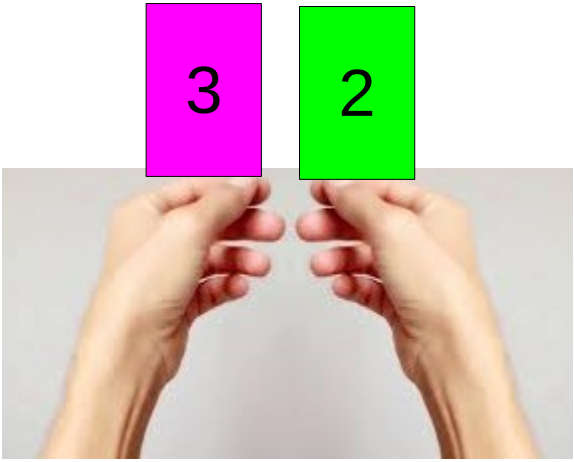
$v = 3$ $r = 2$

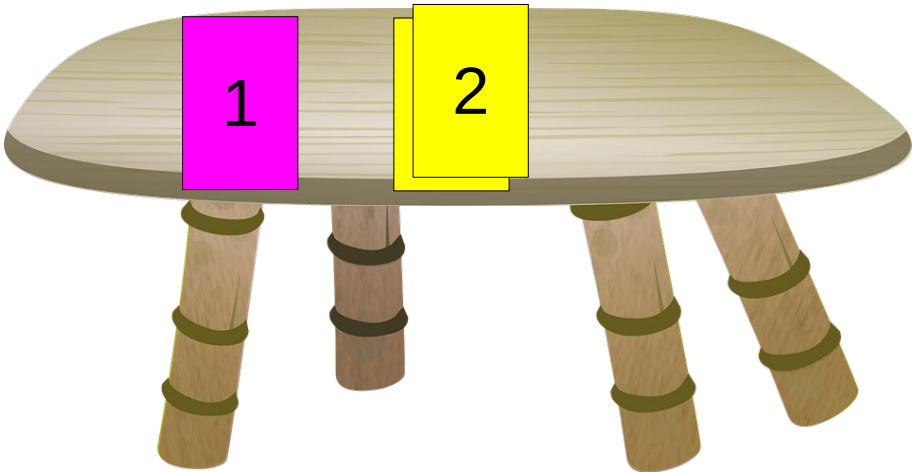
$c = 3$ $h = 2$

Example







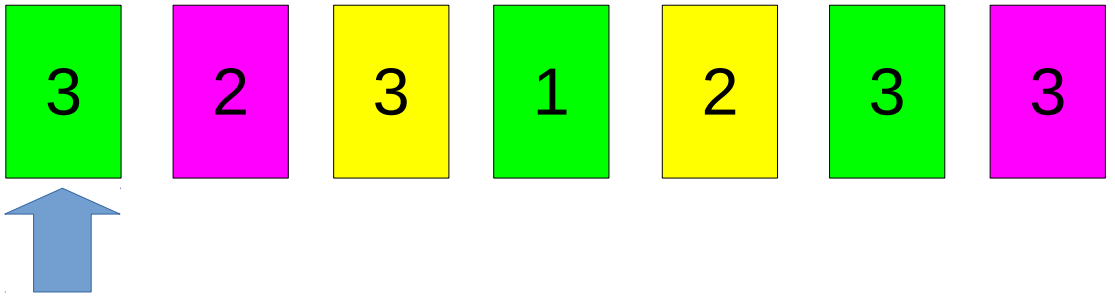


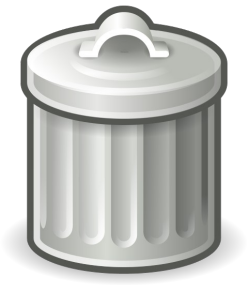
Example

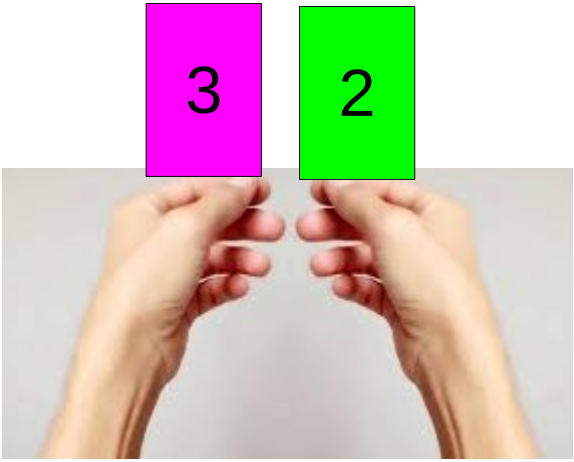
$N = 12$

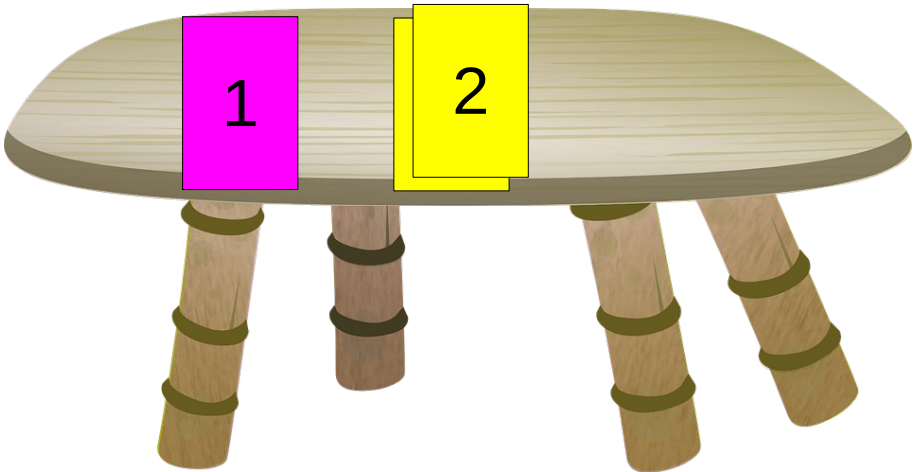
$v = 3 \qquad r = 2$

$c = 3 \qquad h = 2$







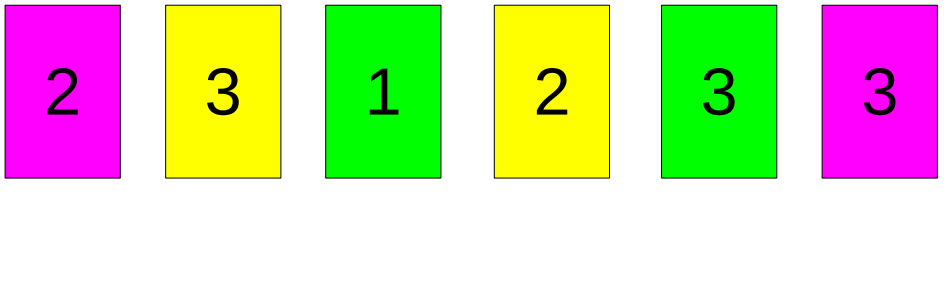


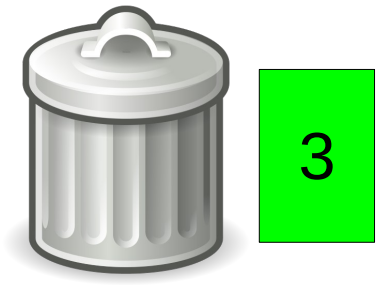
Example

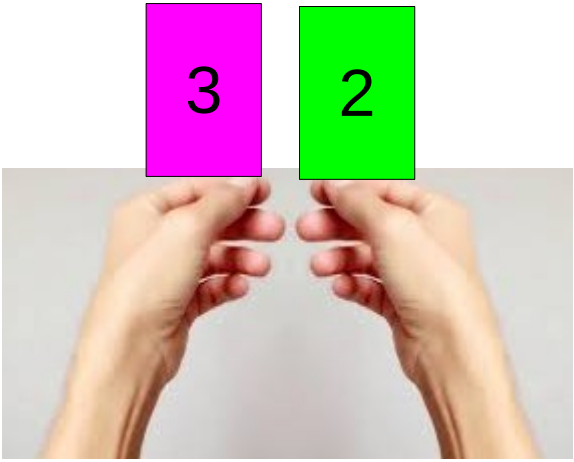
$N = 12$

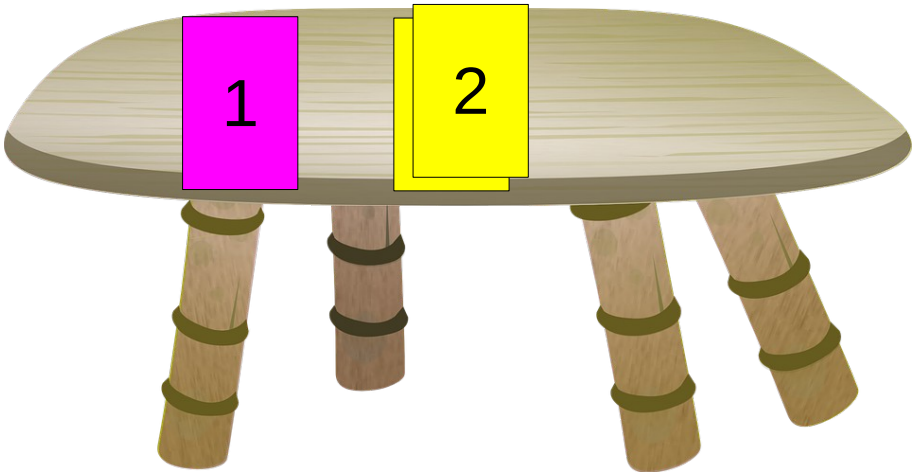
$v = 3$ $r = 2$

$c = 3$ $h = 2$







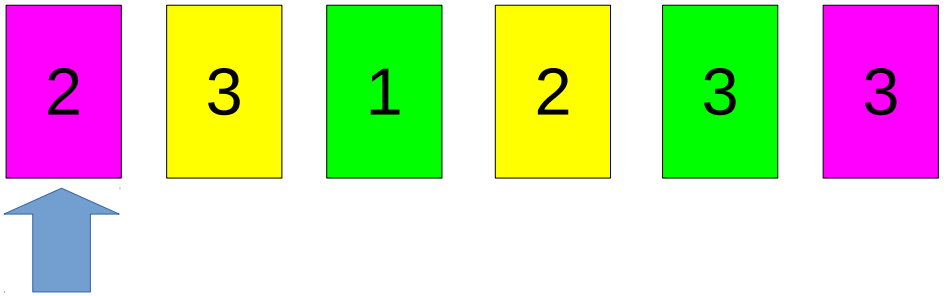


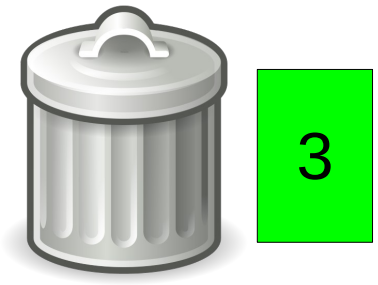
Example

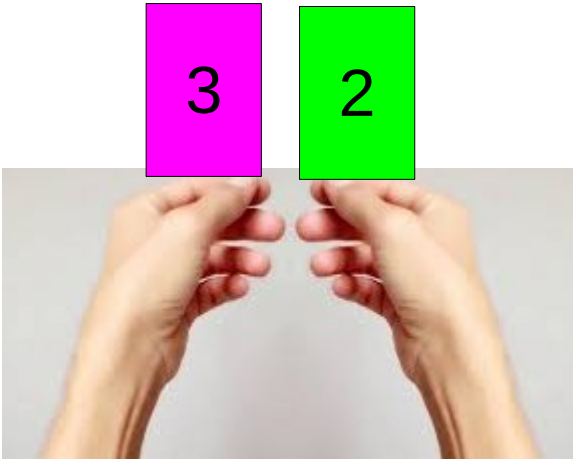
$N = 12$

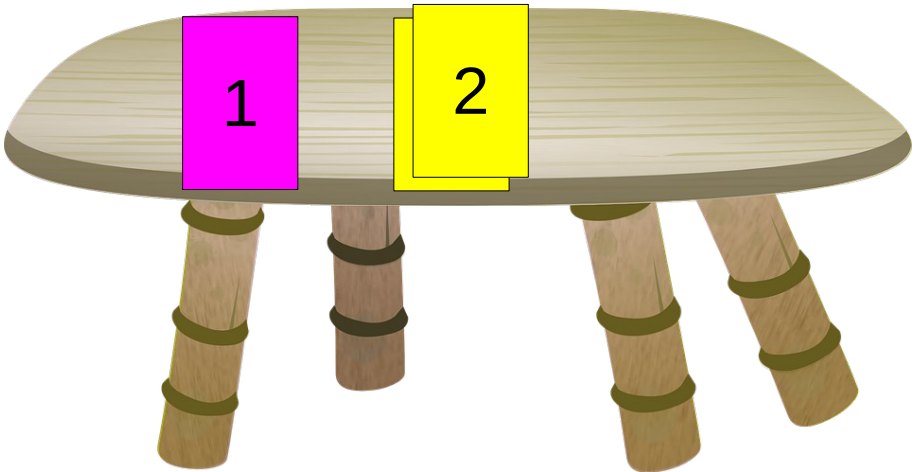
$v = 3$ $r = 2$

$c = 3$ $h = 2$







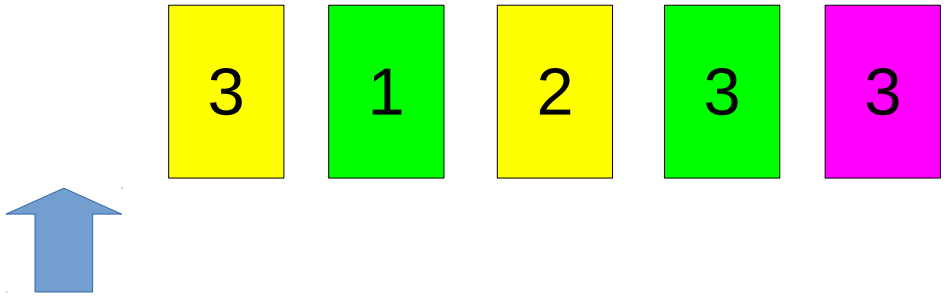


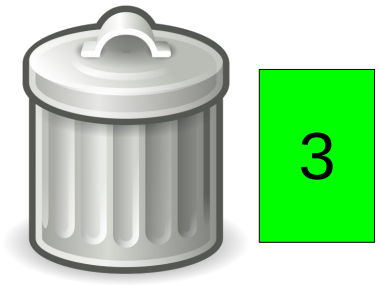
Example

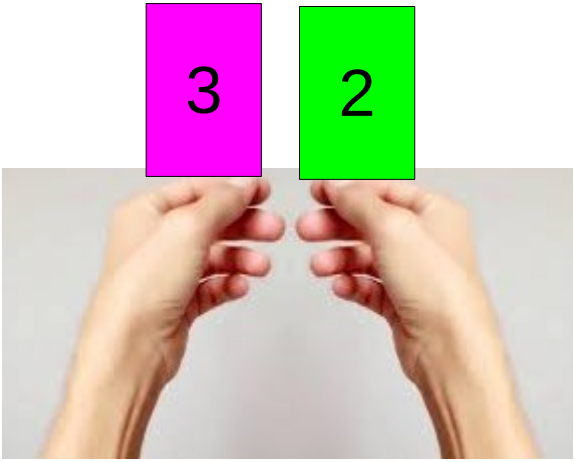
$N = 12$

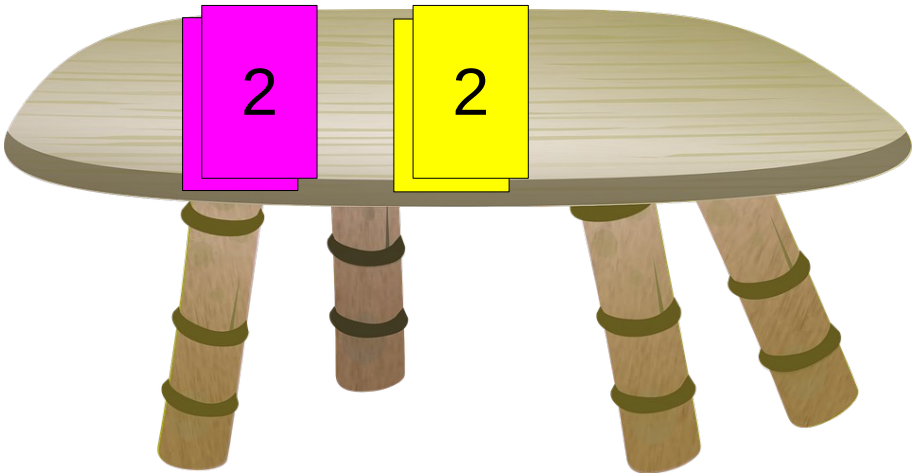
$v = 3 \qquad r = 2$

$c = 3 \qquad h = 2$







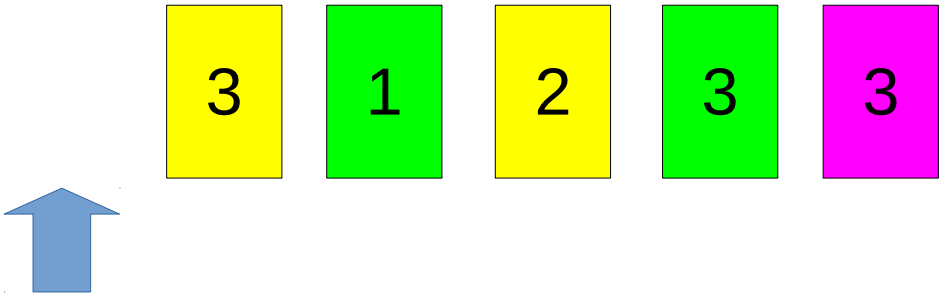


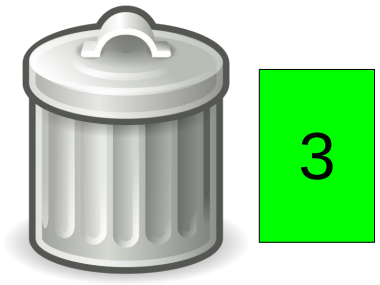
Example

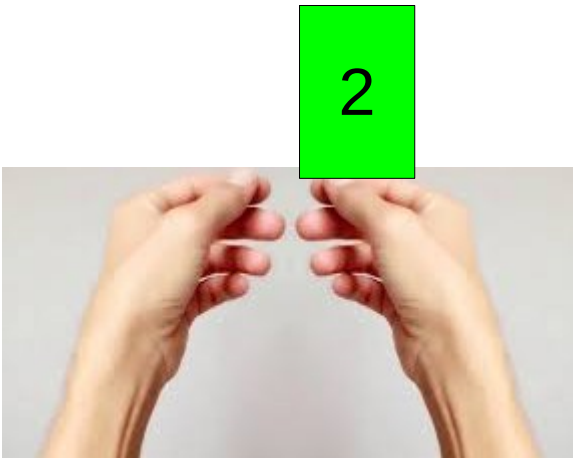
$N = 12$

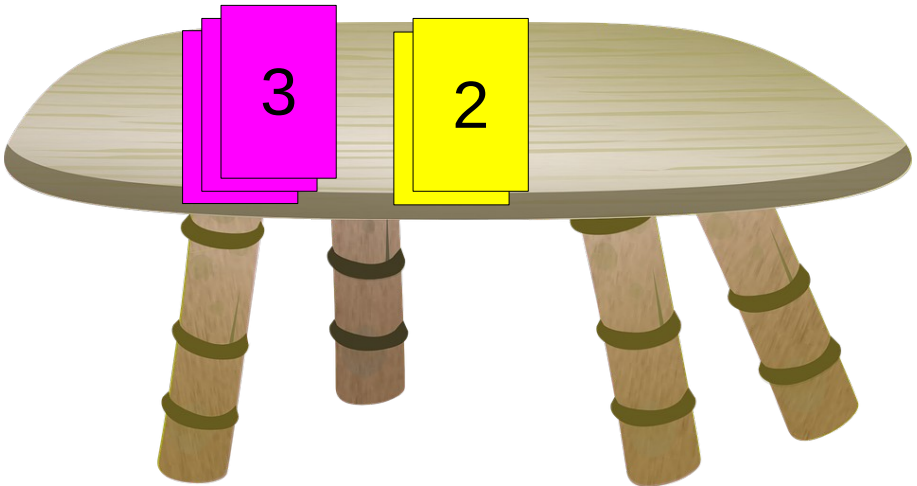
$v = 3$ $r = 2$

$c = 3$ $h = 2$







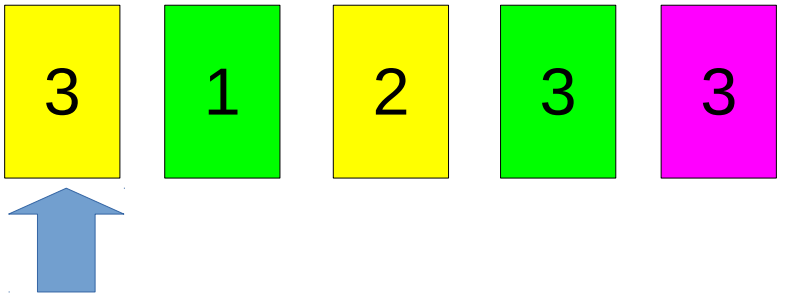


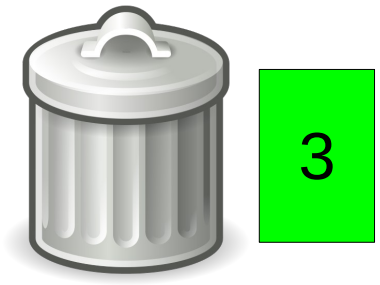
Example

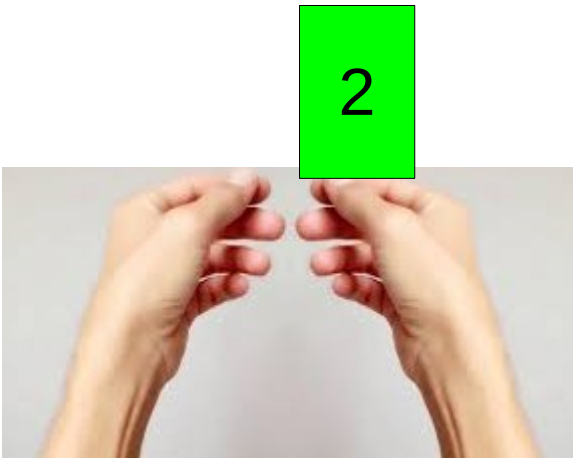
$N = 12$

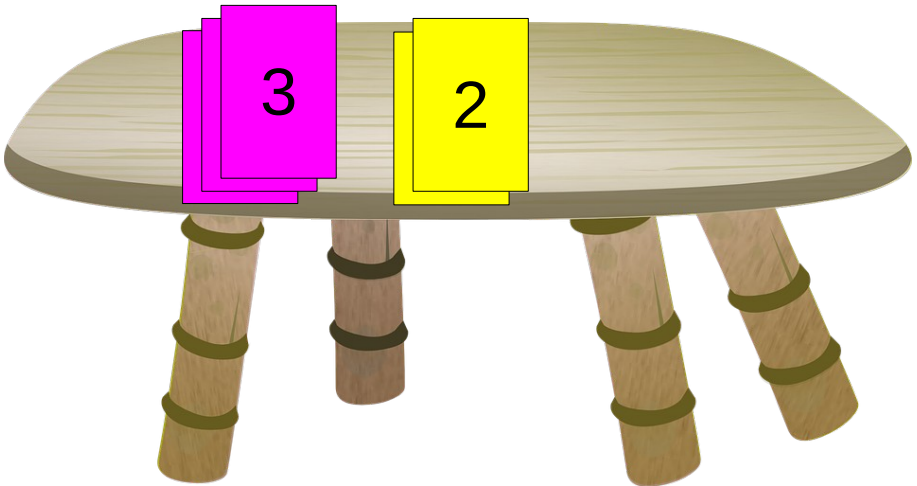
$v = 3$ $r = 2$

$c = 3$ $h = 2$







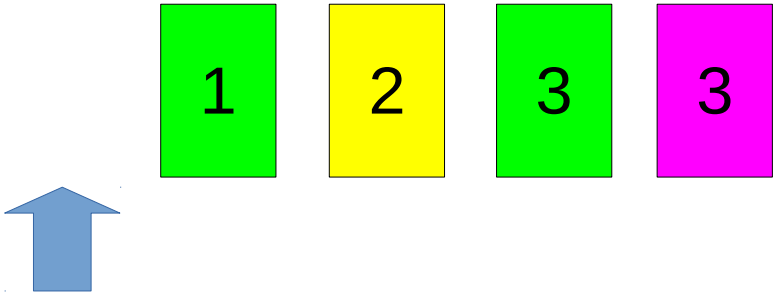


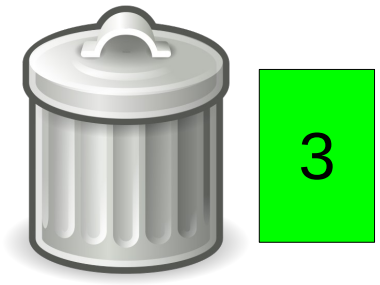
Example

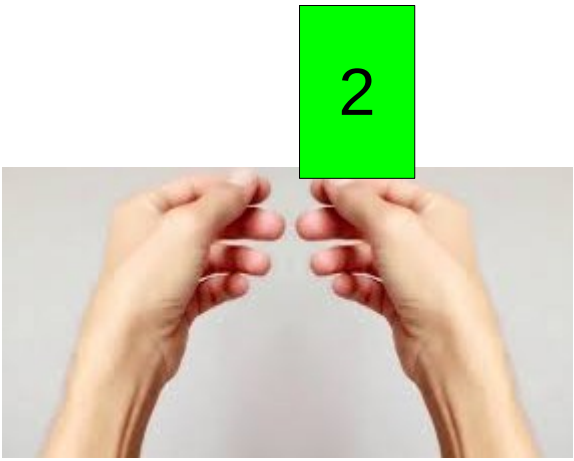
$N = 12$

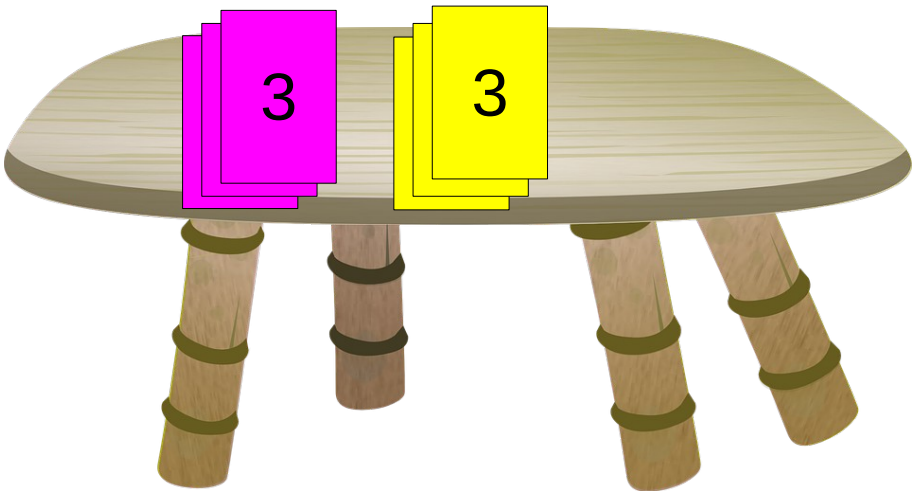
$v = 3$ $r = 2$

$c = 3$ $h = 2$







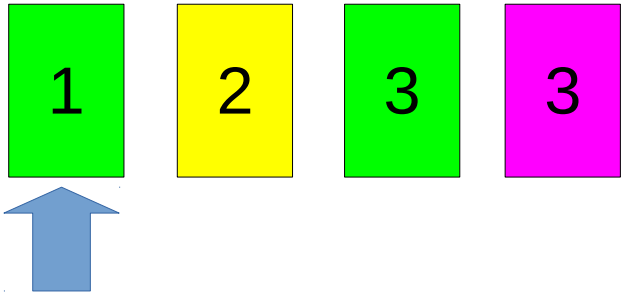


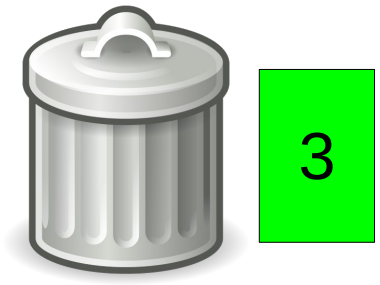
Example

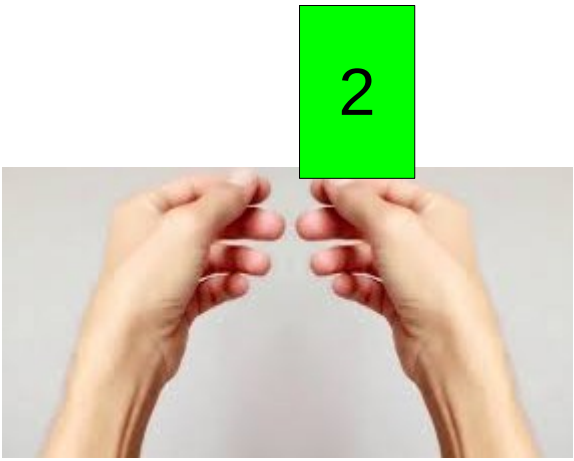
$N = 12$

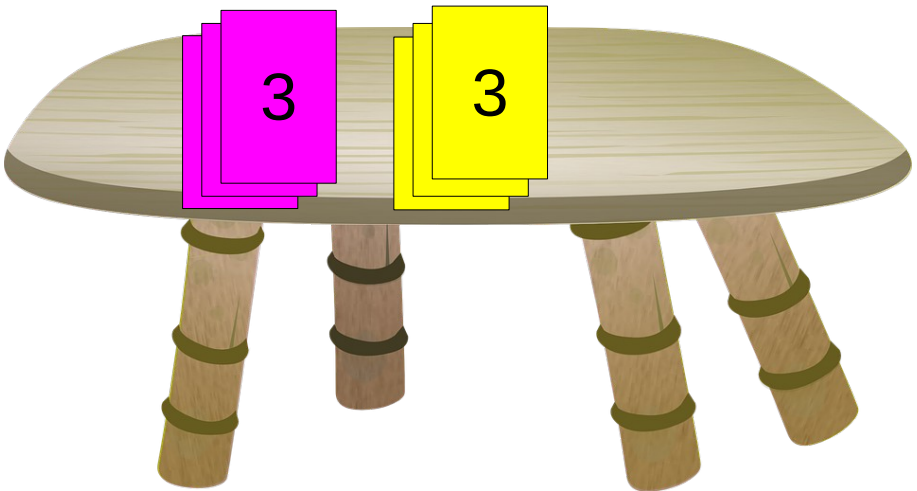
$v = 3 \qquad r = 2$

$c = 3 \qquad h = 2$







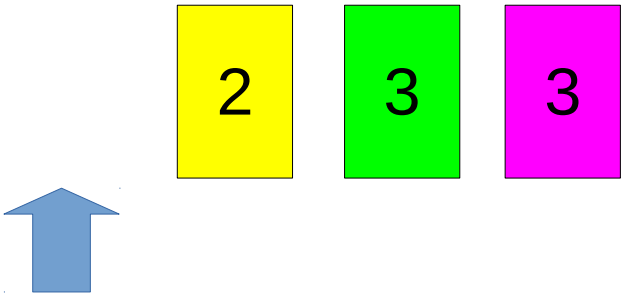


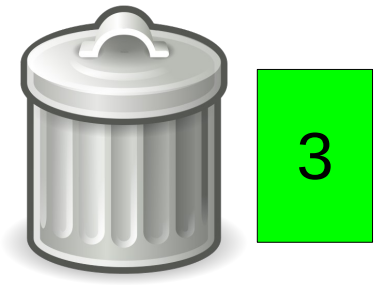
Example

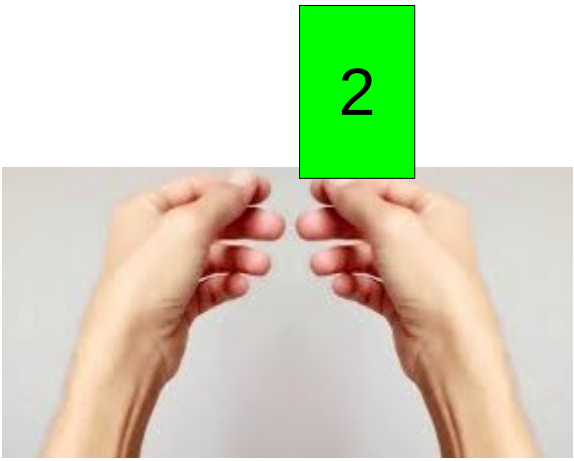
$N = 12$

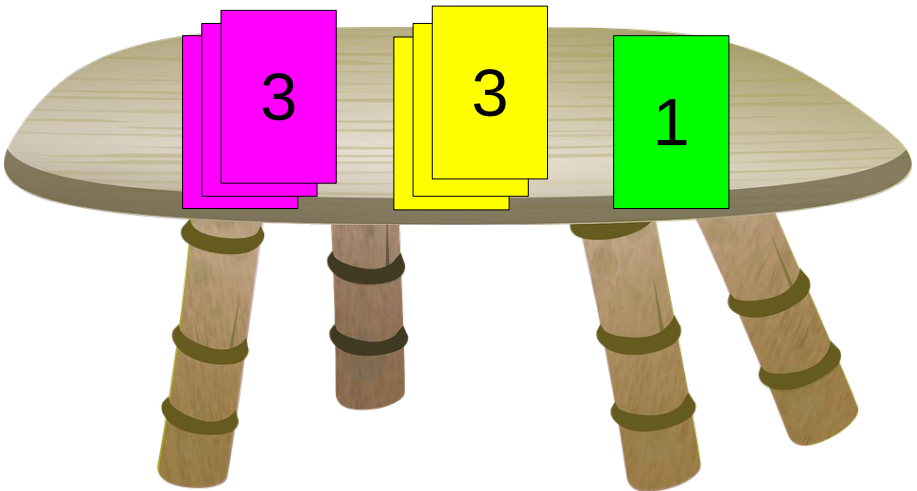
$v = 3 \qquad r = 2$

$c = 3 \qquad h = 2$







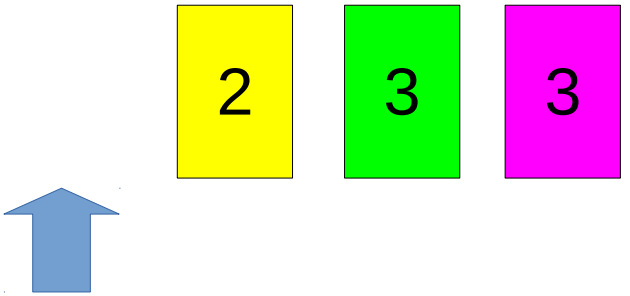


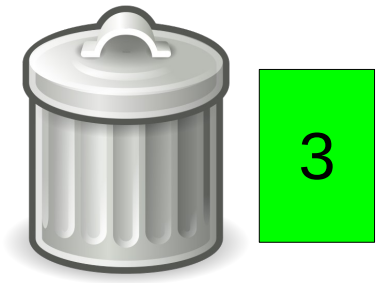
Example

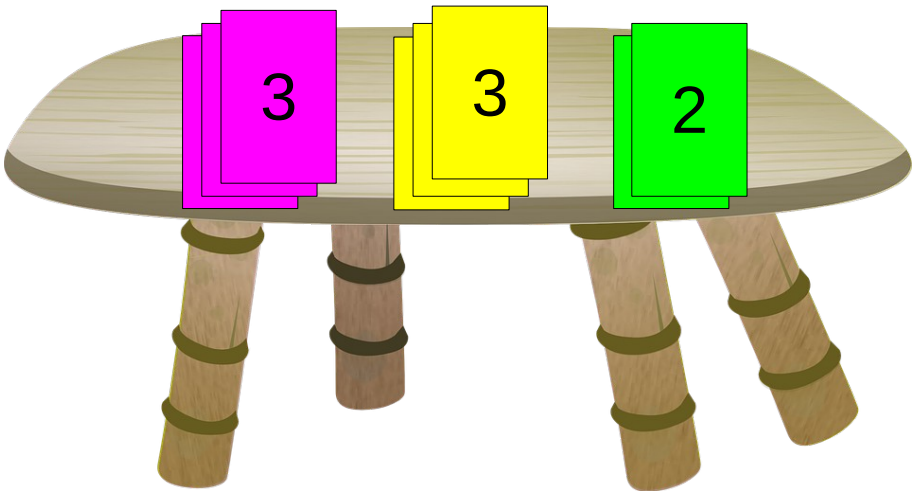
$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$





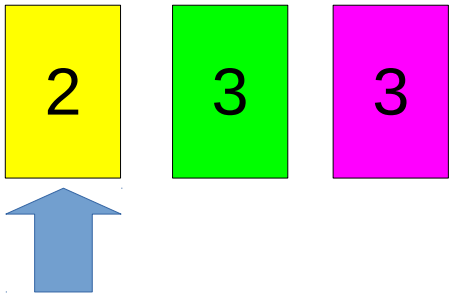


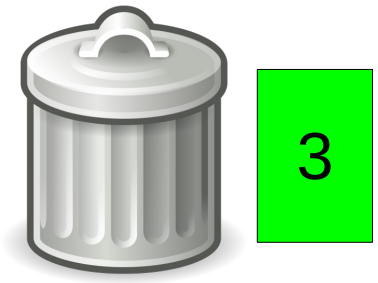
Example

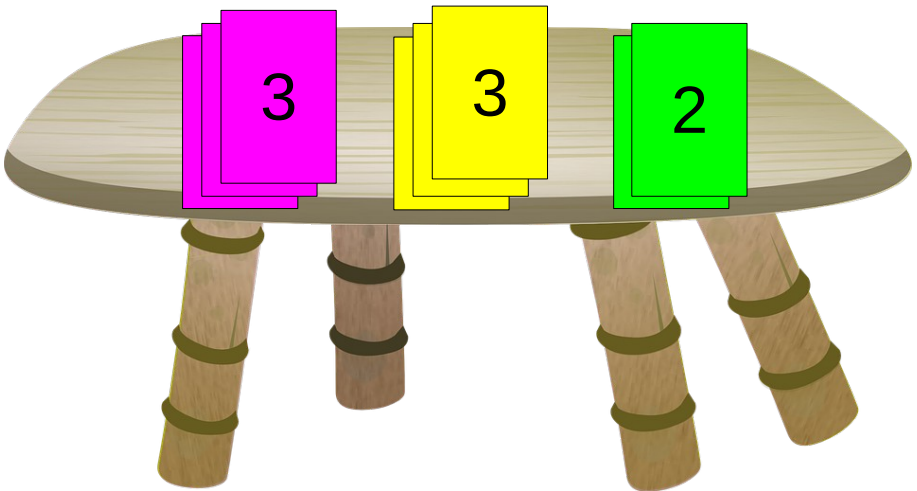
$N = 12$

$v = 3$ $r = 2$

$c = 3$ $h = 2$





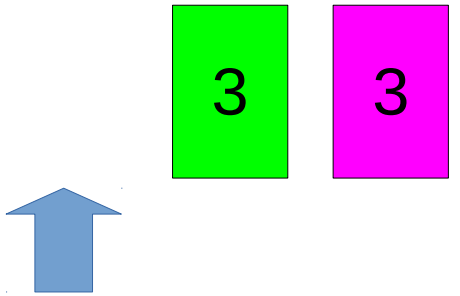


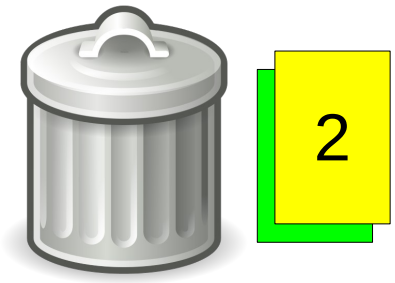
$N = 12$

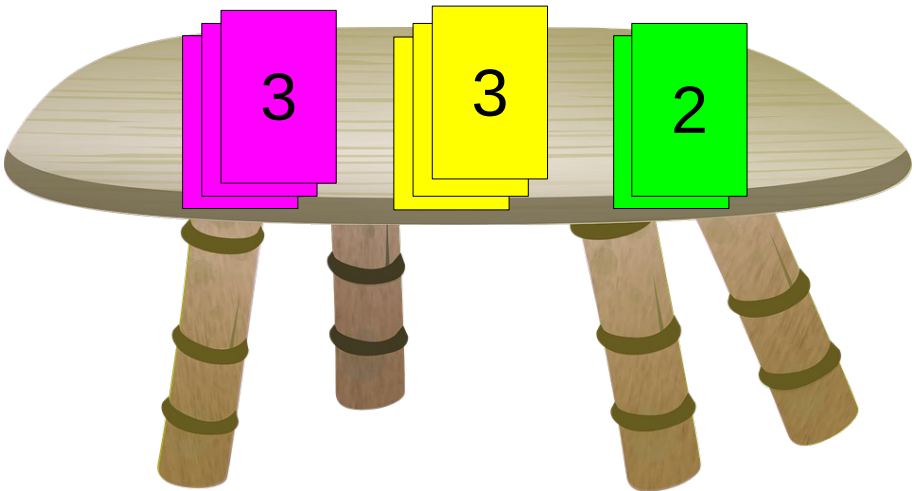
$v = 3$ $r = 2$

$c = 3$ $h = 2$

Example





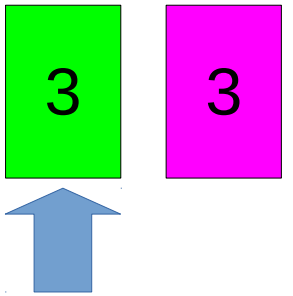


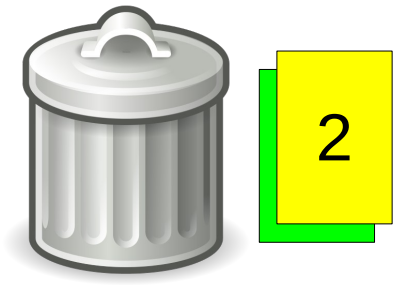
$N = 12$

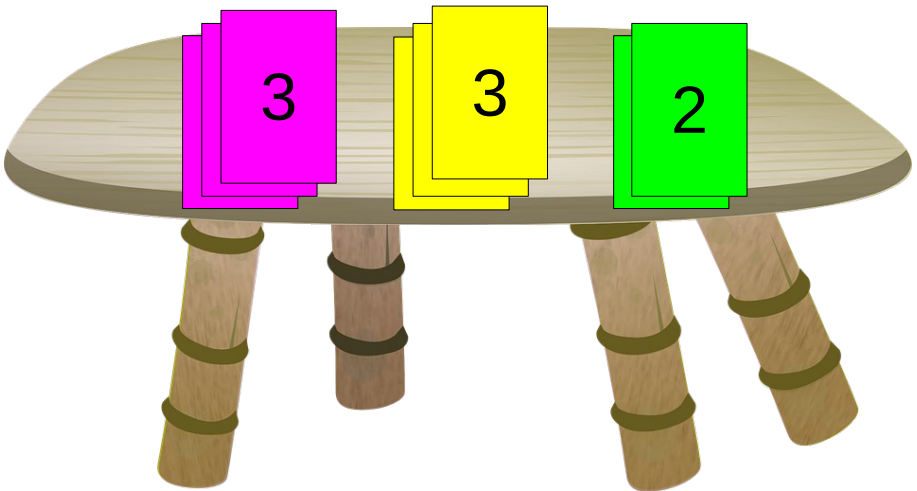
$v = 3$ $r = 2$

$c = 3$ $h = 2$

Example





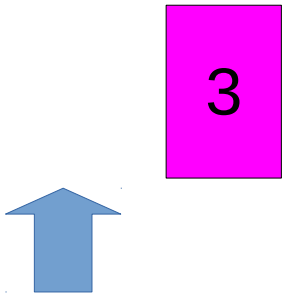


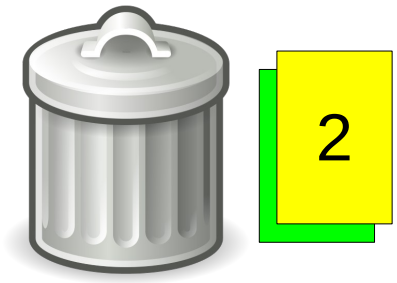
Example

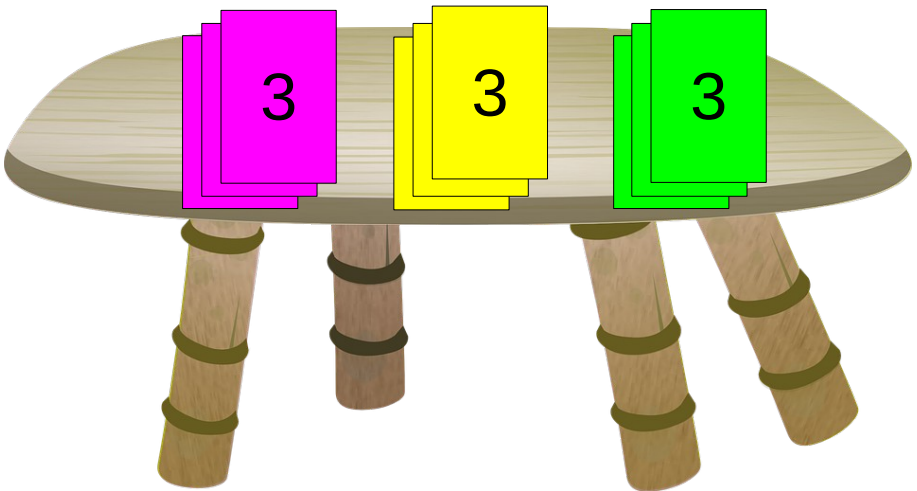
$N = 12$

$v = 3 \qquad r = 2$

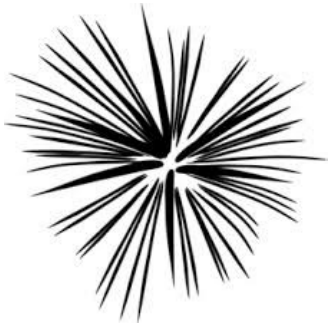
$c = 3 \qquad h = 2$







Results



Case Studied	Result
$h \geq 2,\ r \geq 2$	NP-complete
$r = 1$	$\mathrm{O}(N)$
$c = 1$	$\mathrm{O}(N + n \cdot \log h)$
General case	$\mathrm{O}(N \cdot h \cdot c^h \cdot n^{h+\mathrm{c}-1})$

NP-hardness

Reduction: From 3-SAT to Hanabi:

- $h,r = 2$
- c,v unbounded

$\rightarrow$ Construct a sequence of cards:

sequence playable $\leftrightarrow$ formula satisfiable

NP-hardness

Hand-restructuring gadgets: use some dummy color with unique appearances of values.

Consider the sequence $2 \circ 3 \circ \ldots \circ i \circ 1$.

$\rightarrow$ In order to play all blue cards, we need to keep $i - 1$ spots of the hand occupied.

NP-hardness

Hand-restructuring gadgets: use some dummy color with unique appearances of values.

Consider the sequence $2 \circ 3 \circ \ldots \circ i \circ 1$.

$\rightarrow$ In order to play all blue cards, we need to keep $i - 1$ spots of the hand occupied.

- *Hand-dump* gadget D: guarantees that we have no leftovers from previous gadgets.
- *Hand-reduction* gadget $R(i)$: reduces the size of the hand by i.

NP-hardness

Reduction is from 3-SAT:

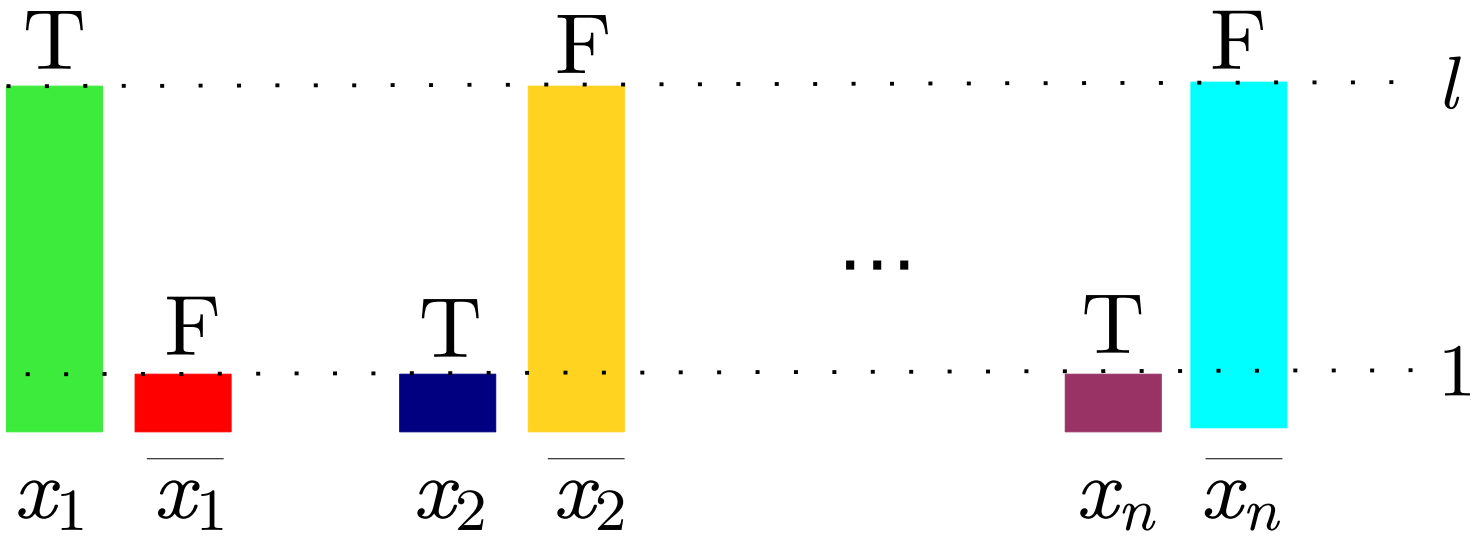
- Each variable corresponds to 2 colors.

  Variable gadget: $R(1)$ 2 2 1 3 ... l 1 3 ... l D

- We can store at most one card of value 2, playing all the way up to l in this color, but only value 1 in the other.
- Define (and maintain) two levels: high (l) and low (1), indicating the truth assignment.

NP-hardness

After assignment phase:

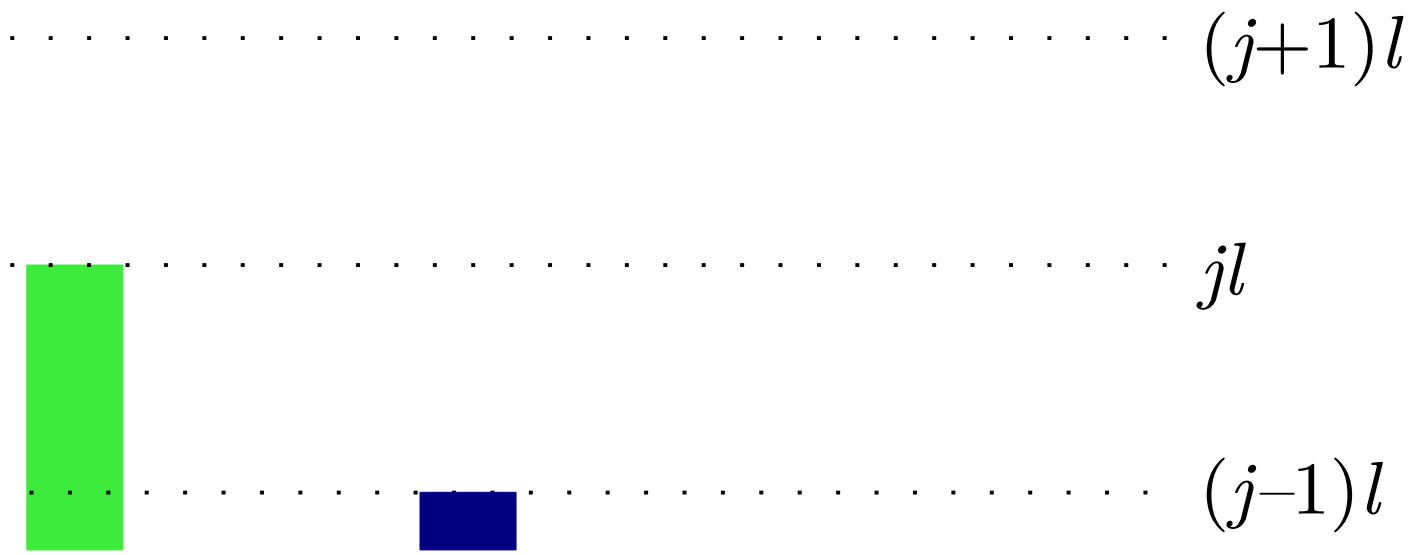


$x_1 = \mathrm{T}, \quad x_2 = \mathrm{F}, \quad \ldots, \quad x_n = \mathrm{F}$

NP-hardness

Satisfaction phase:

For each literal not in C_j we add:

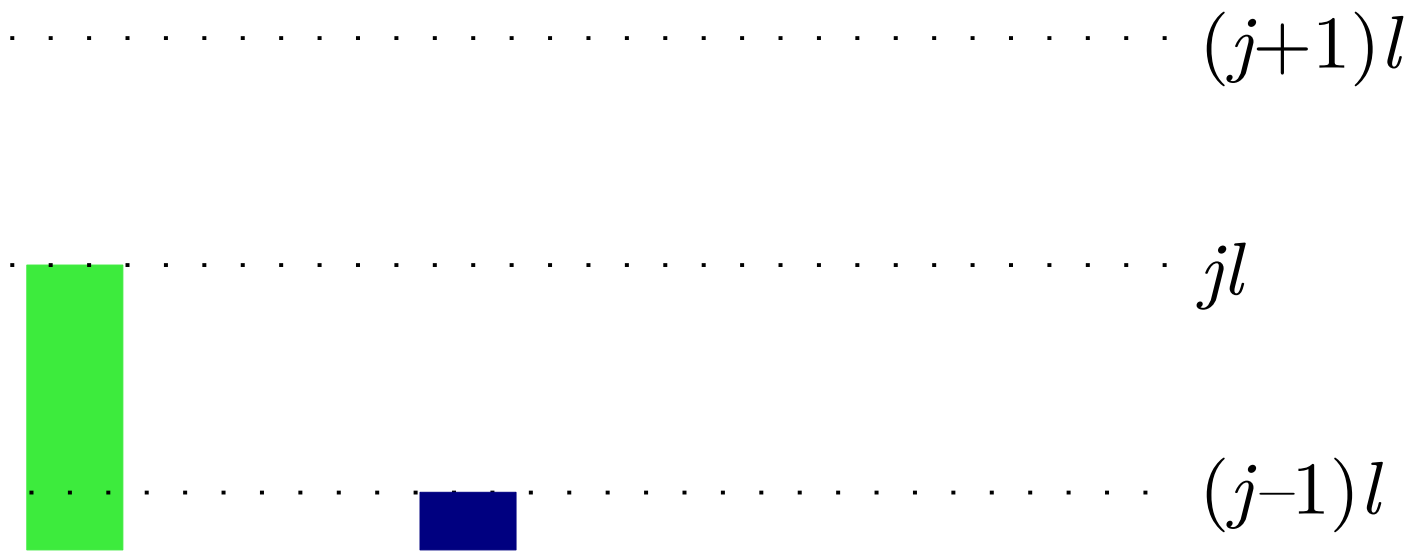


NP-hardness

Satisfaction phase:

For each literal not in C_j we add:

$R(2)$

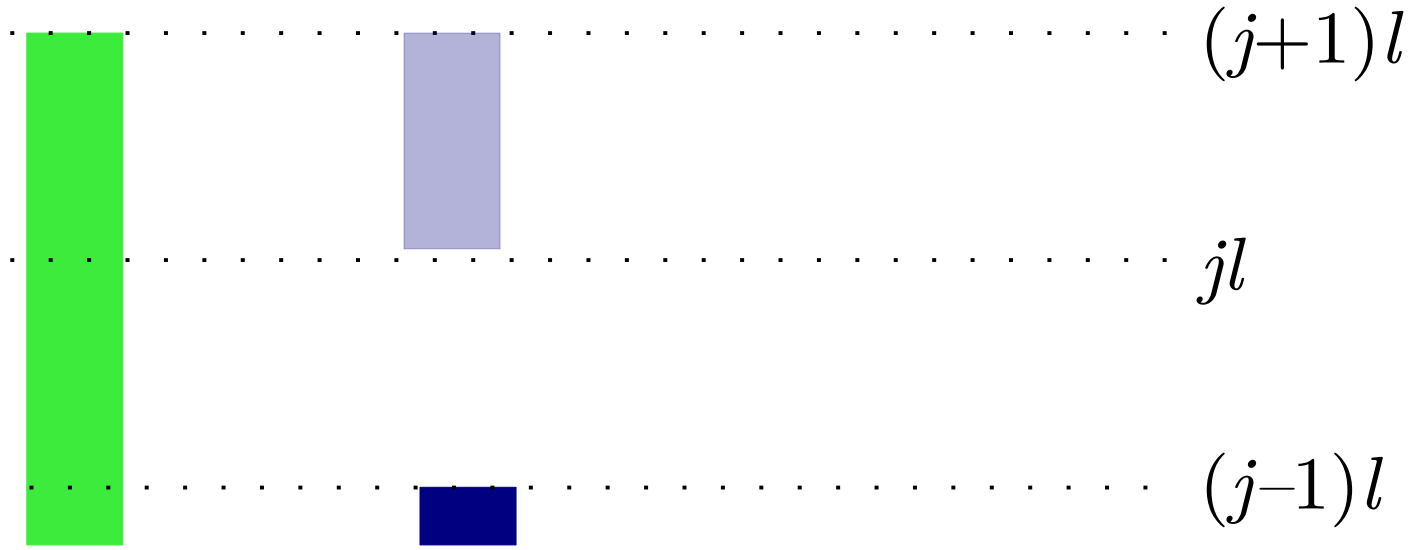


NP-hardness

Satisfaction phase:

For each literal not in C_j we add:

$R(2)$ jl+1 ... (j+1)l

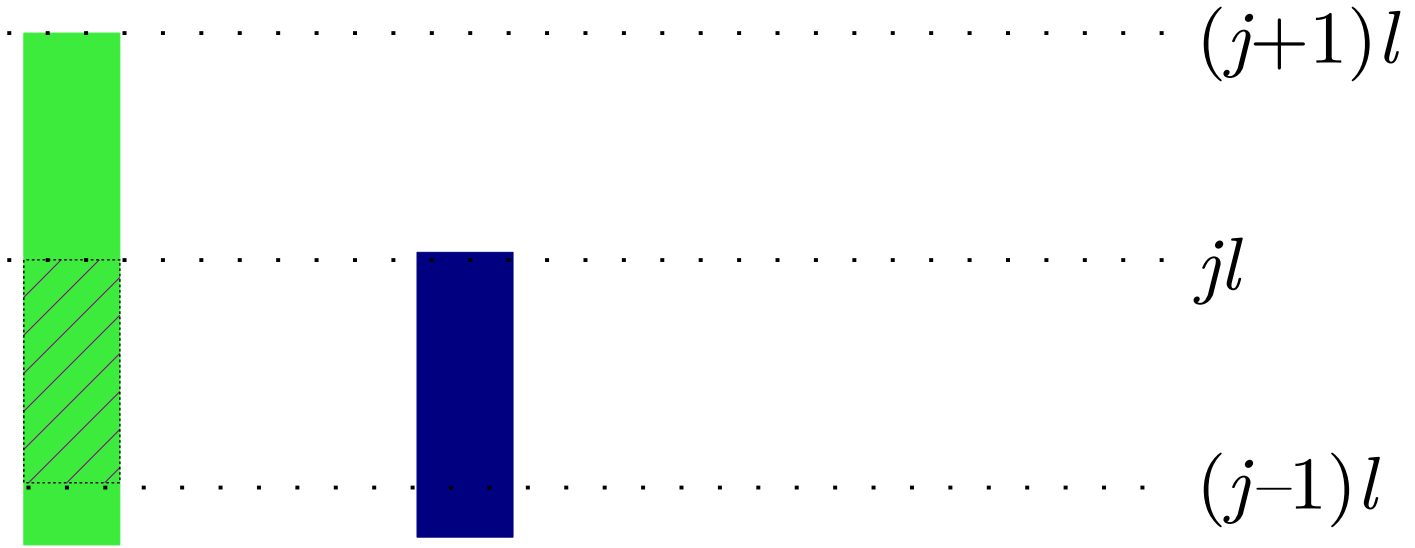


NP-hardness

Satisfaction phase:

For each literal not in C_j we add:

$R(2)$ jl+1 ... $(j$+1$)l$ $(j$–1$)l$+1 ... jl

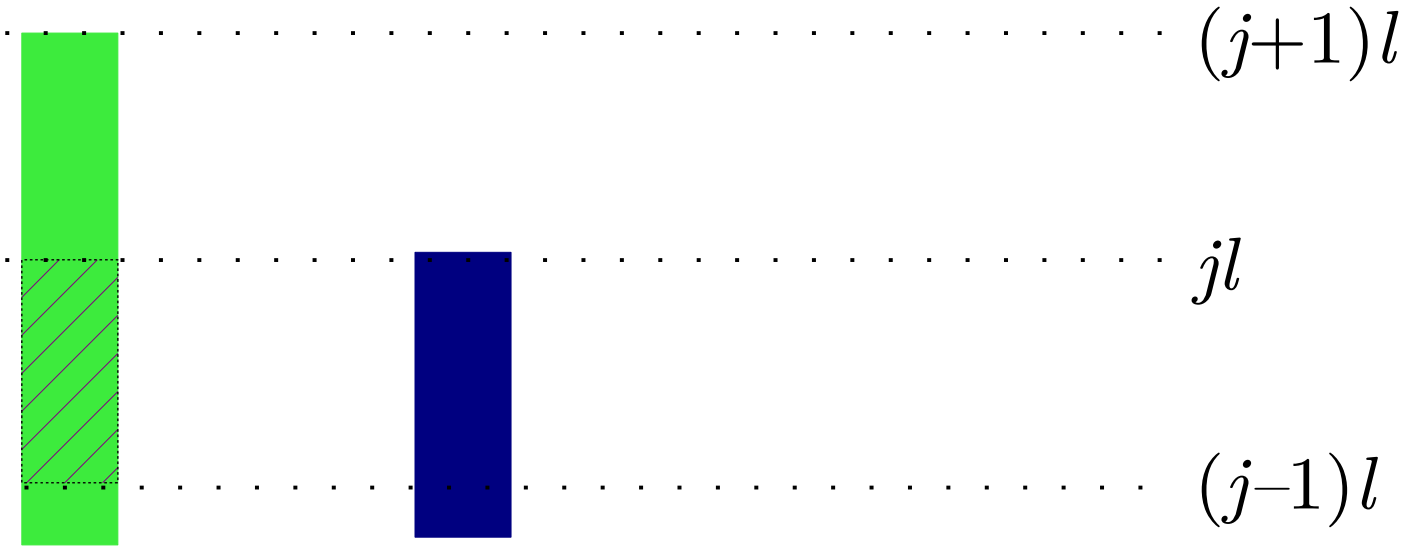


NP-hardness

Satisfaction phase:

For each literal not in C_j we add:

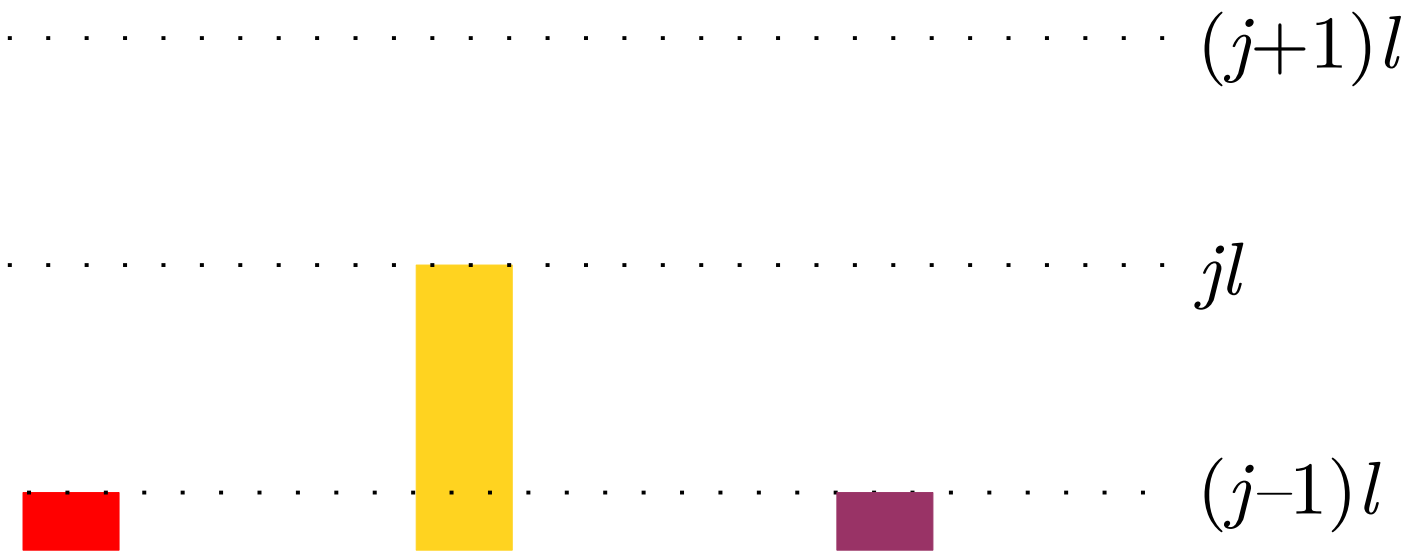
$R(2)$ jl+1 ... $(j$+1$)l$ $(j$–1$)l$+1 ... jl D



NP-hardness

Satisfaction phase:

For the three literals of C_j (r,y,p) add:

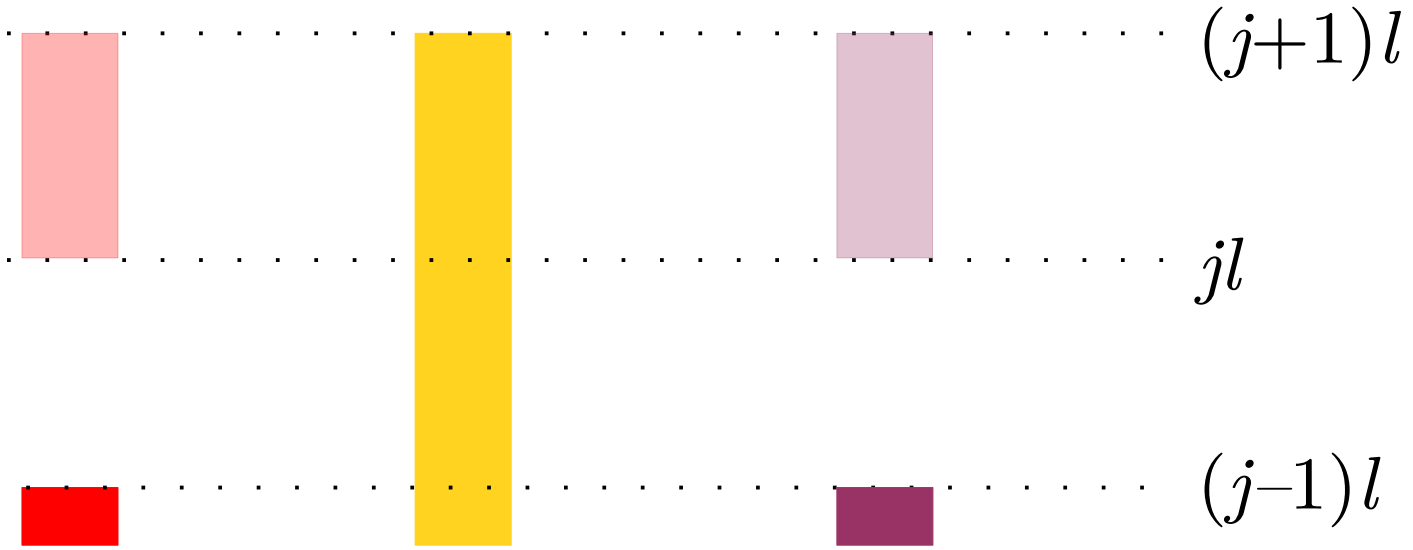


NP-hardness

Satisfaction phase:

For the three literals of C_j (r,y,p) add:

$jl{+}1$ ... $(j{+}1)l$ $jl{+}1$ ... $(j{+}1)l$ $jl{+}1$ ... $(j{+}1)l$ D

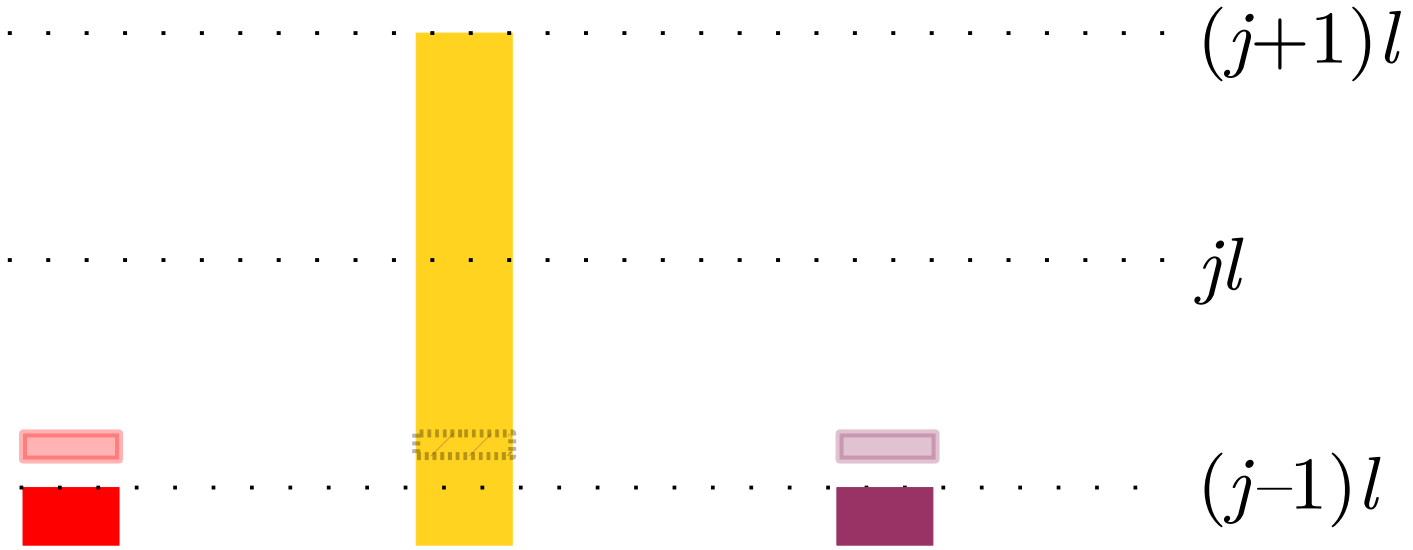


NP-hardness

Satisfaction phase:

For the three literals of C_j (r,y,p) add:

$(j–1)l+2$ $(j–1)l+2$ $(j–1)l+2$

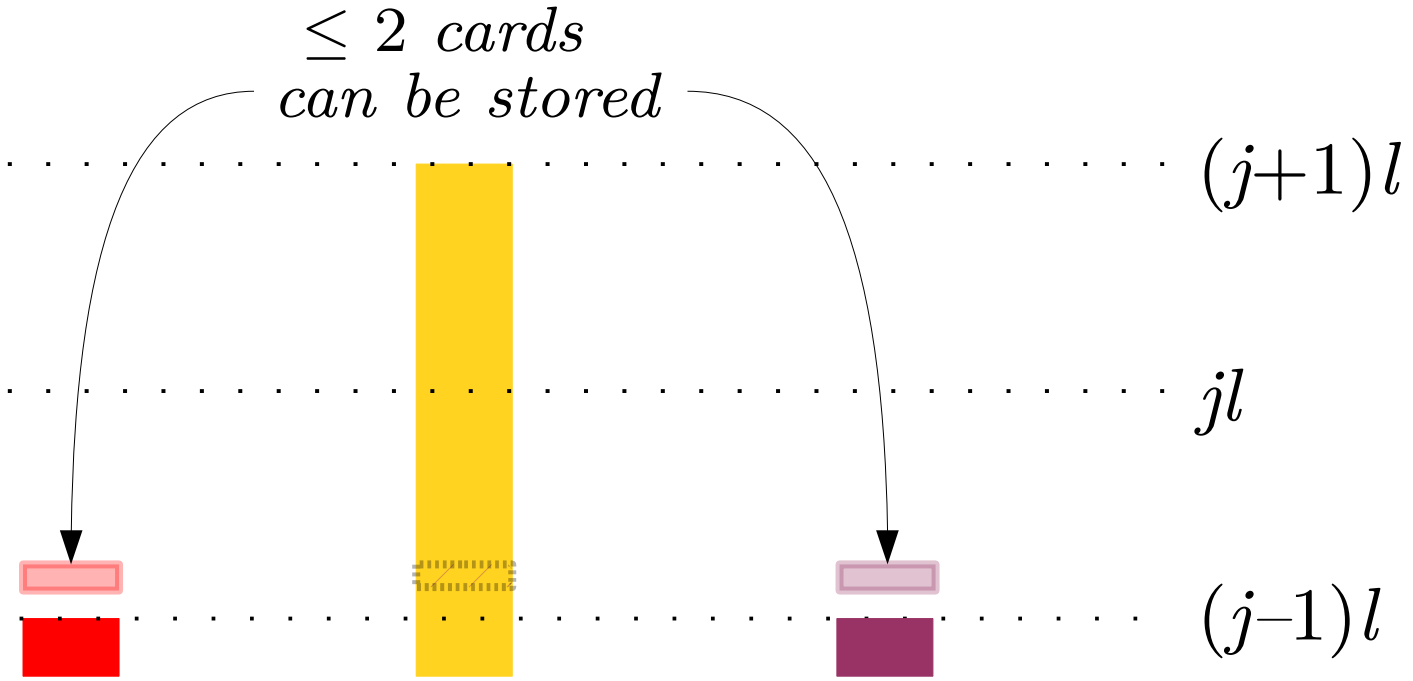


NP-hardness

Satisfaction phase:

For the three literals of C_j (r,y,p) add:

$(j–1)l+2$ $(j–1)l+2$ $(j–1)l+2$

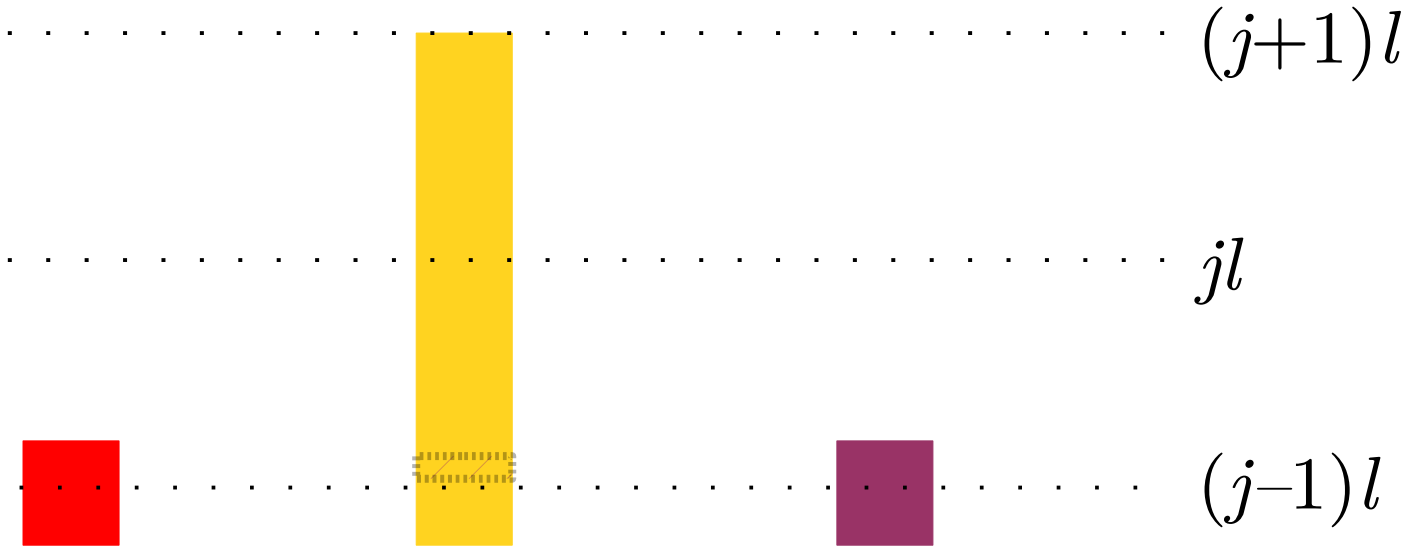


NP-hardness

Satisfaction phase:

For the three literals of C_j (r,y,p) add:

$(j–1)l+1$ $(j–1)l+1$ $(j–1)l+1$

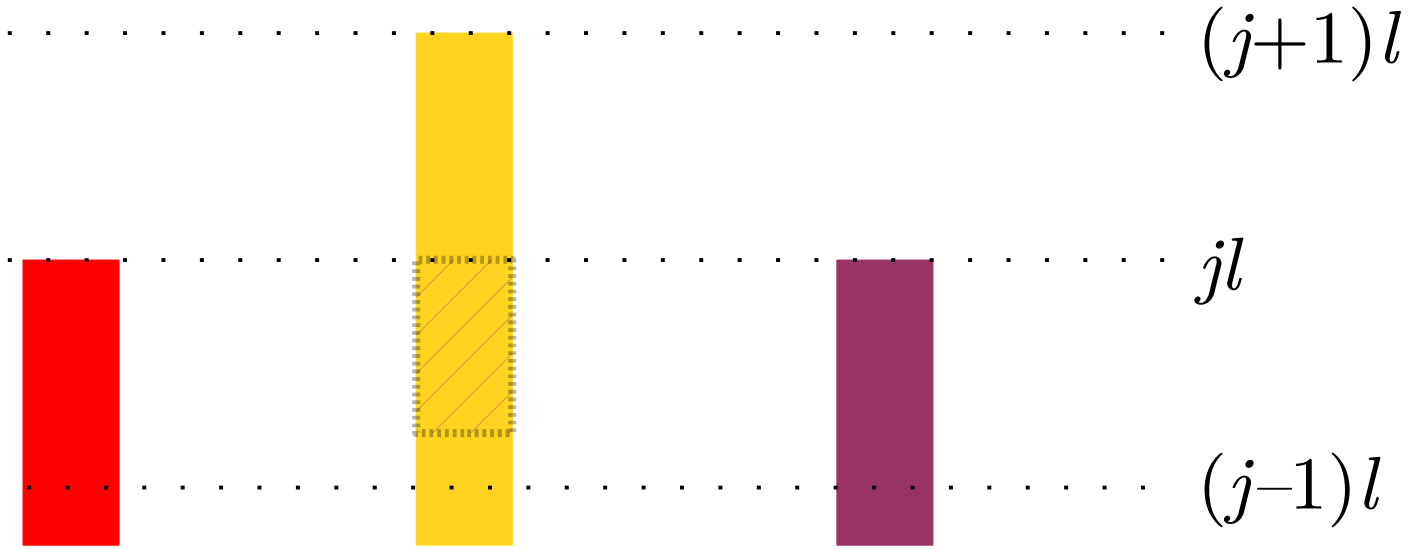


NP-hardness

Satisfaction phase:

For the three literals of C_j (r,y,p) add:

$(j\text{–}1)l\text{+}3 \ldots 2j \quad (j\text{–}1)l\text{+}3 \ldots 2j \quad (j\text{–}1)l\text{+}3 \ldots 2j \; D$



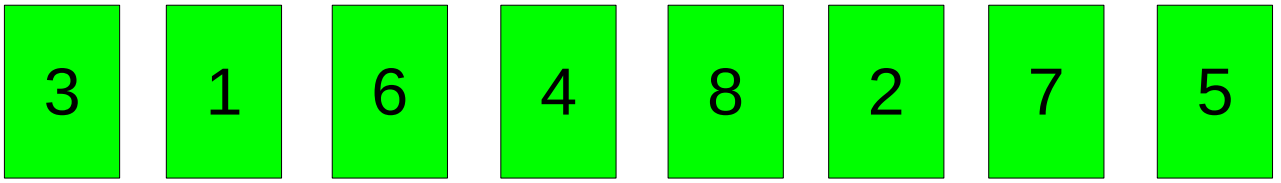
NP-hardness

Reduction works for any $h,r \geq 2$:

- for $r > 2$, we add, for each card, $r - 1$ identical copies next to it;
- for $h > 2$, we use a *hand-reduction* gadget:

  $I' = R(h-2)I$.

No repetitions ($r = 1$)

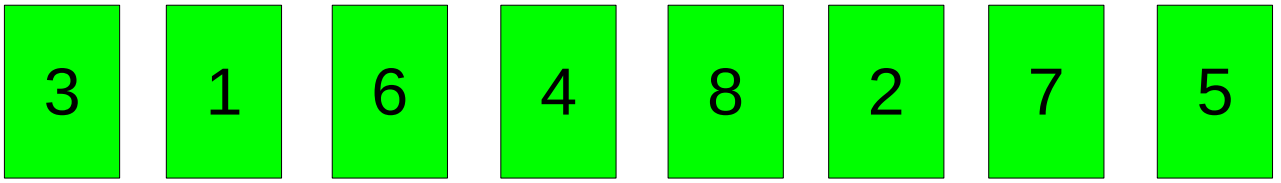


Interesting observations:

- Interval thickness approach
- Bubblesort approach

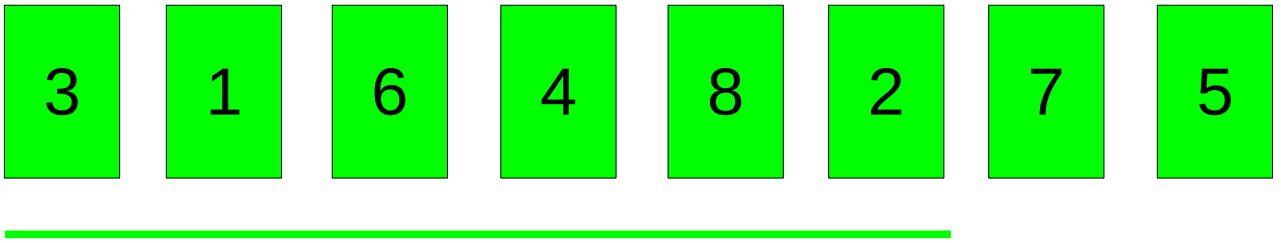
both compute min possible h.

No repetitions ($r = 1$)



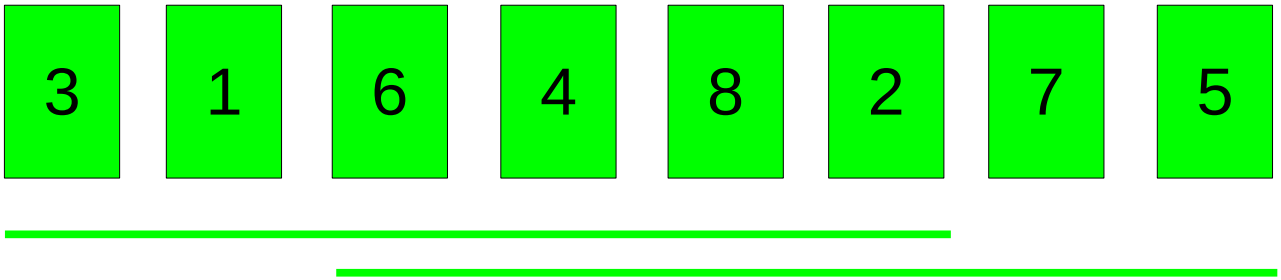
- Interval thickness approach

No repetitions ($r = 1$)



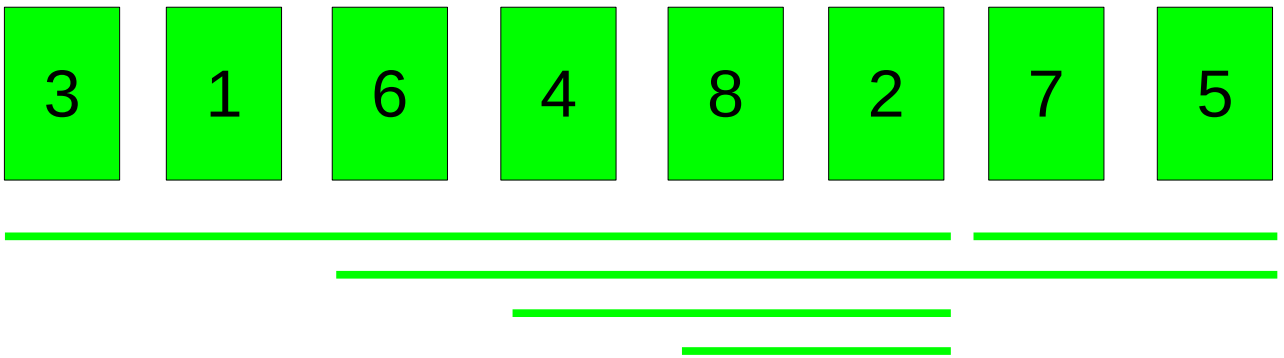
Draw an interval from every card to its rightmost card with same color but smaller value.

No repetitions ($r = 1$)



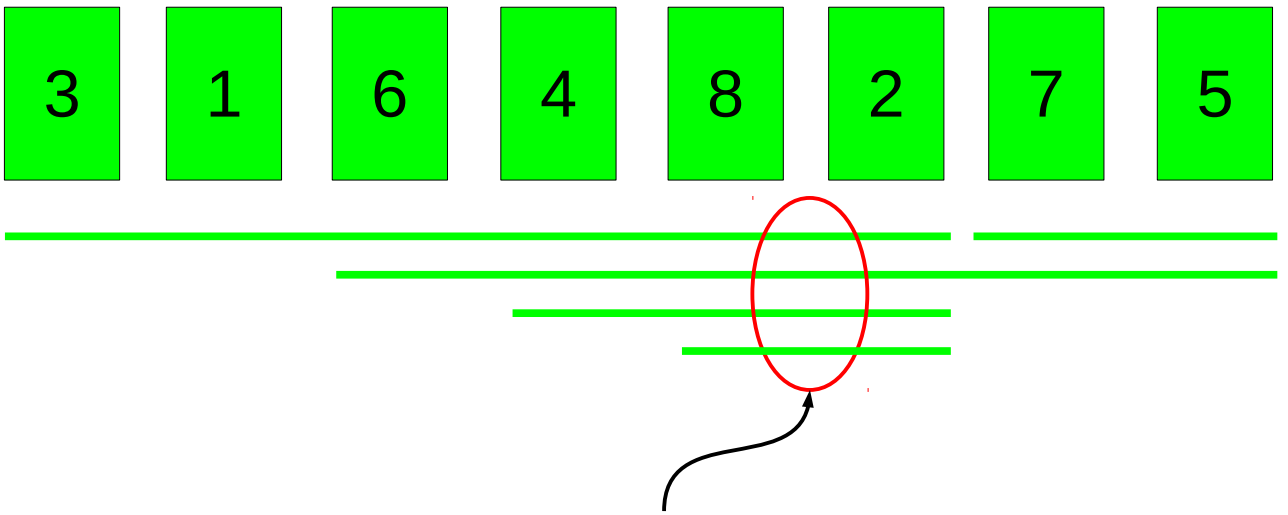
Draw an interval from every card to its rightmost card with same color but smaller value.

No repetitions ($r = 1$)



Draw an interval from every card to its rightmost card with same color but smaller value.

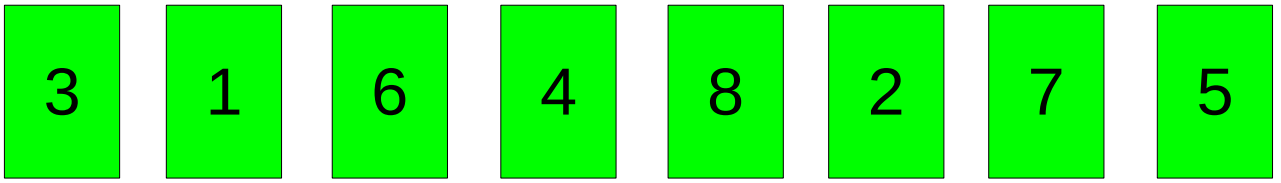
No repetitions ($r = 1$)



$Max\ thickness\ w = 4$

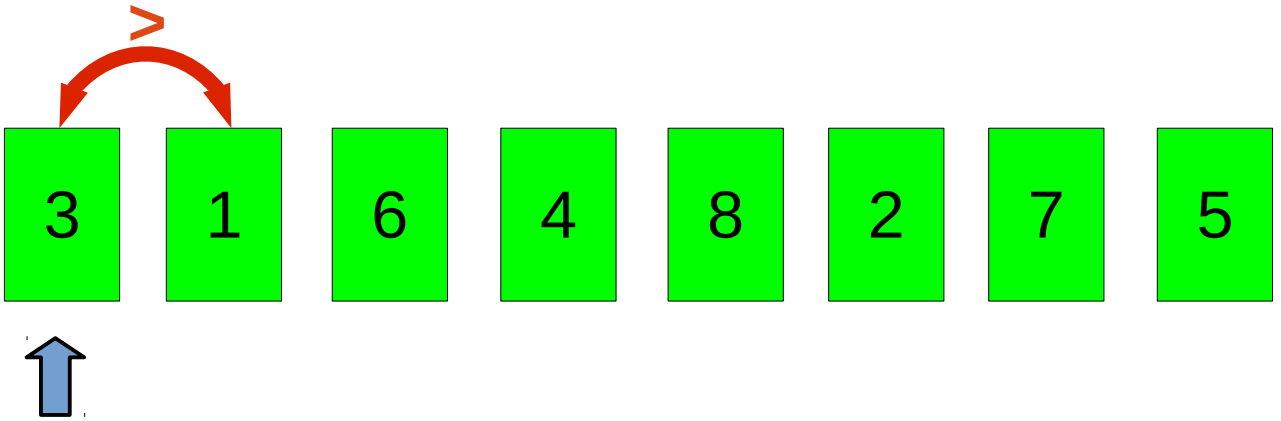
$If\ h \leq w\ then\ the\ instance\ is\ solvable.$

No repetitions ($r = 1$)



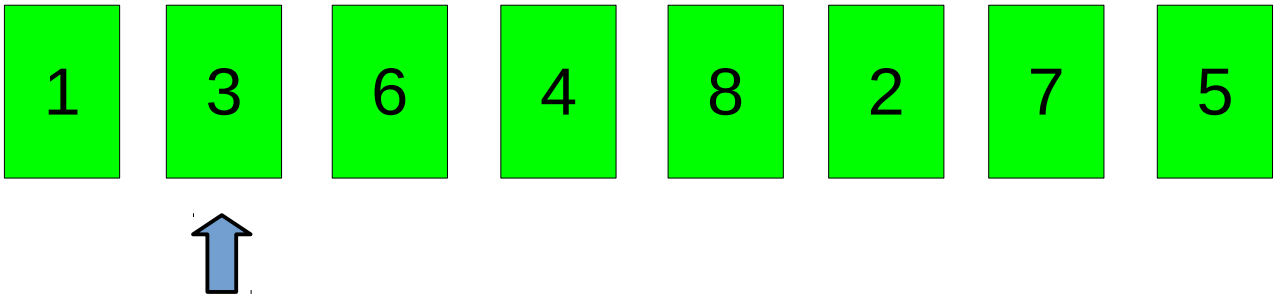
- Bubblesort approach

No repetitions ($r = 1$)



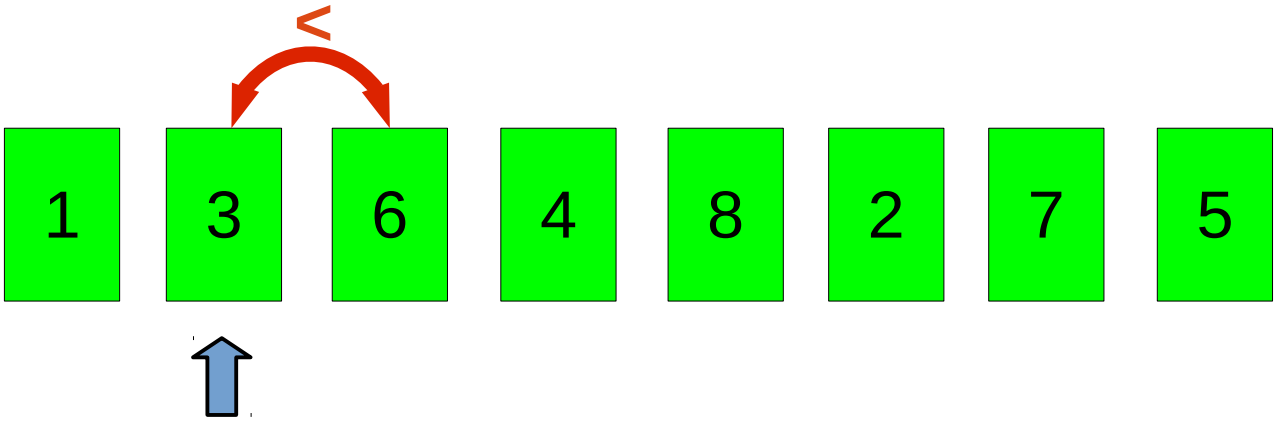
1st pass

No repetitions ($r = 1$)



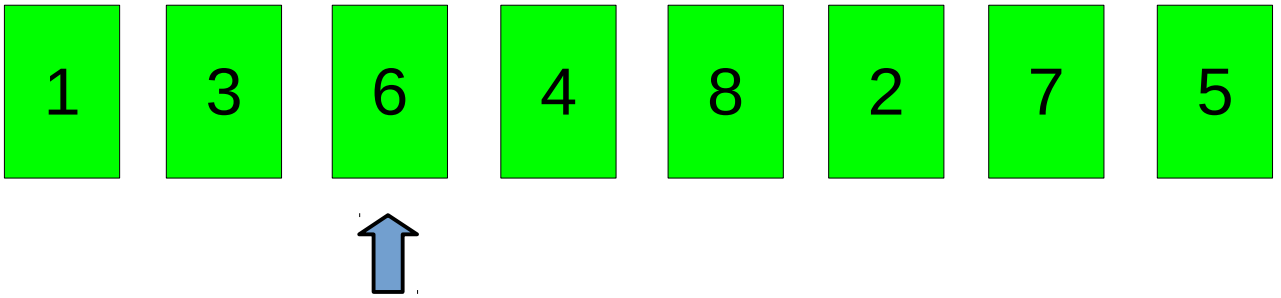
1st pass

No repetitions ($r = 1$)



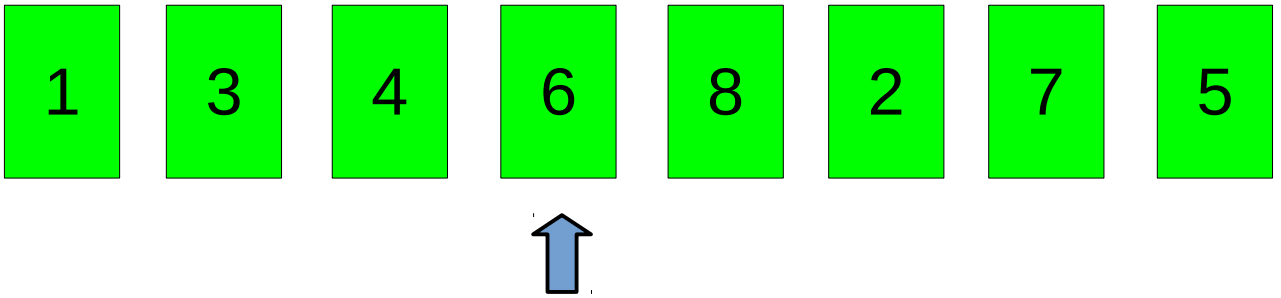
1st pass

No repetitions ($r = 1$)



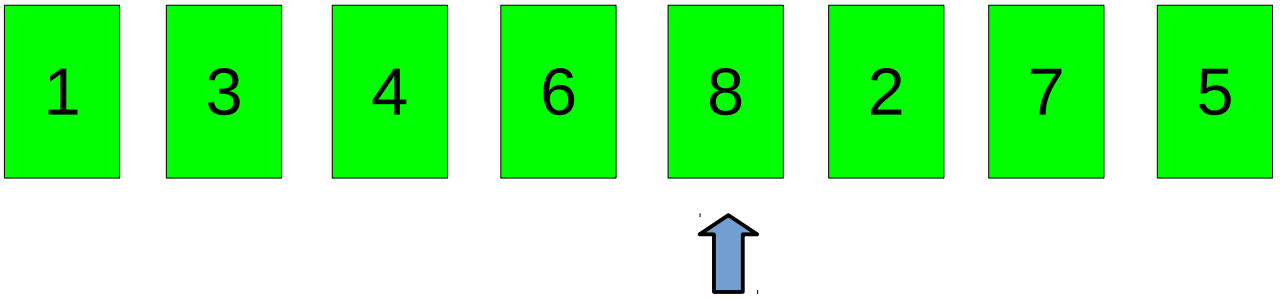
1st pass

No repetitions ($r = 1$)



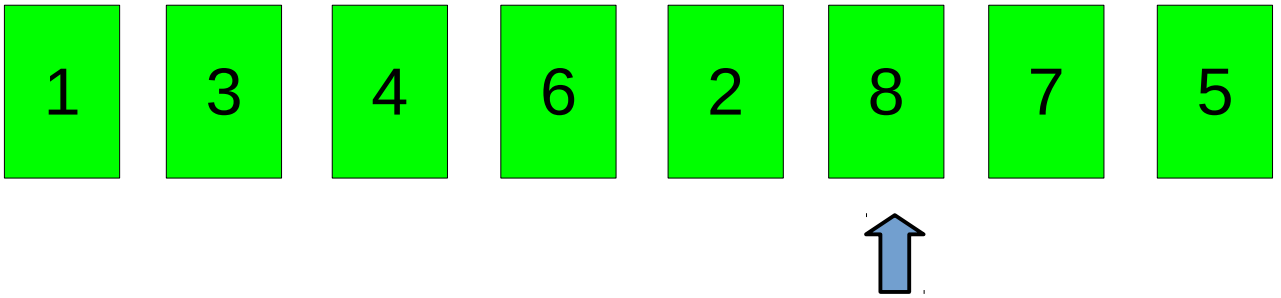
1st pass

No repetitions ($r = 1$)



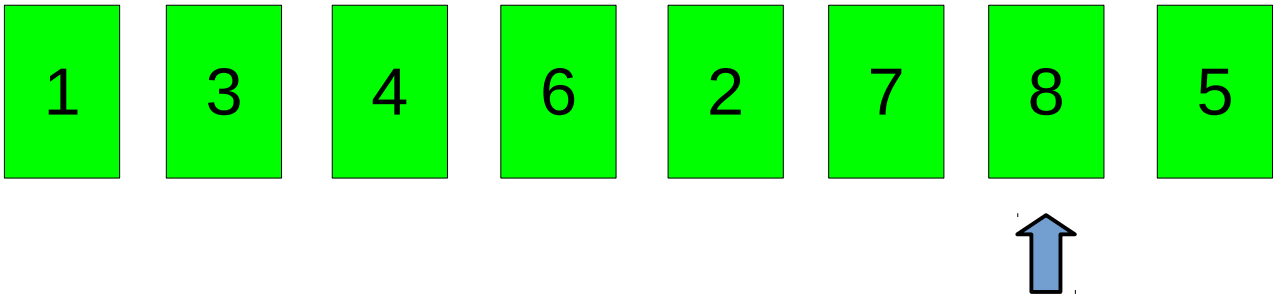
1st pass

No repetitions ($r = 1$)



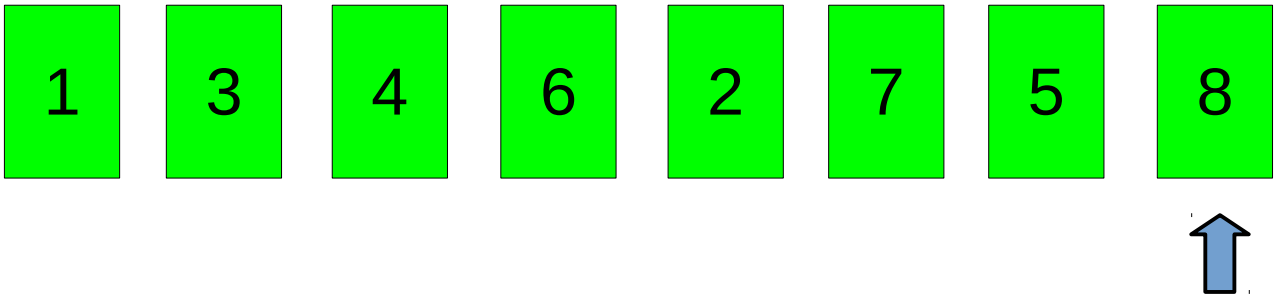
1st pass

No repetitions ($r = 1$)



1st pass

No repetitions ($r = 1$)



1st pass

No repetitions ($r = 1$)

1	3	4	2	6	5	7	8

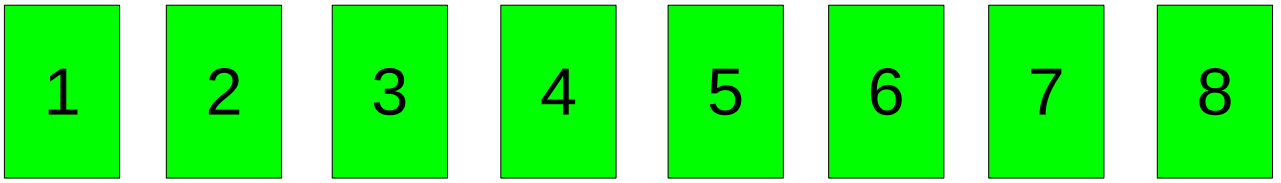
2nd pass

No repetitions ($r = 1$)

1	3	2	4	5	6	7	8

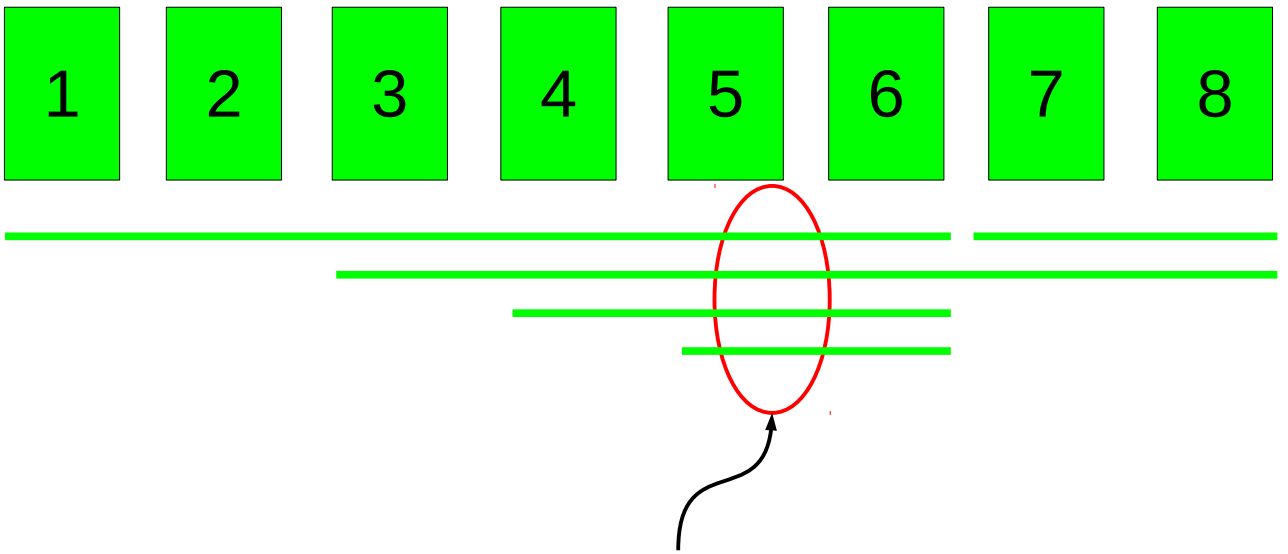
3rd pass

No repetitions ($r = 1$)



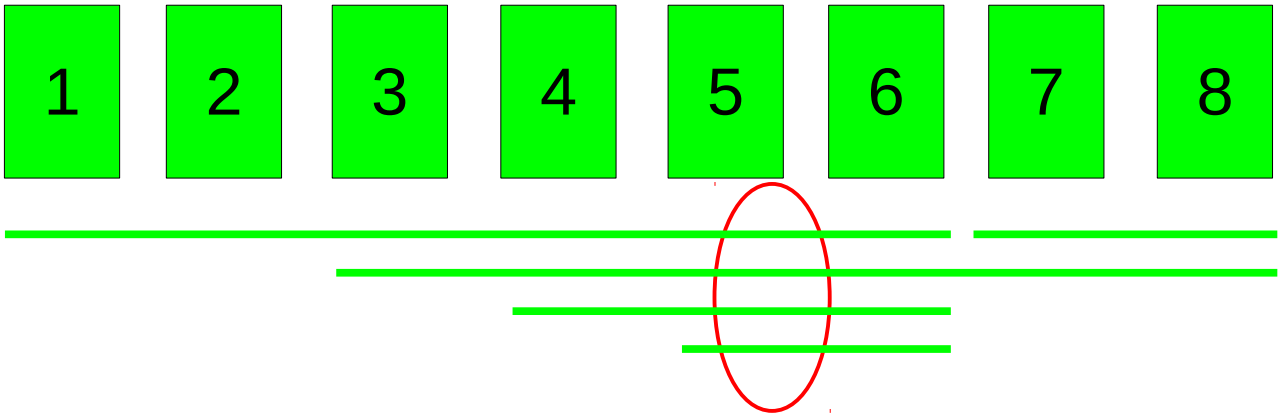
4th pass

No repetitions ($r = 1$)



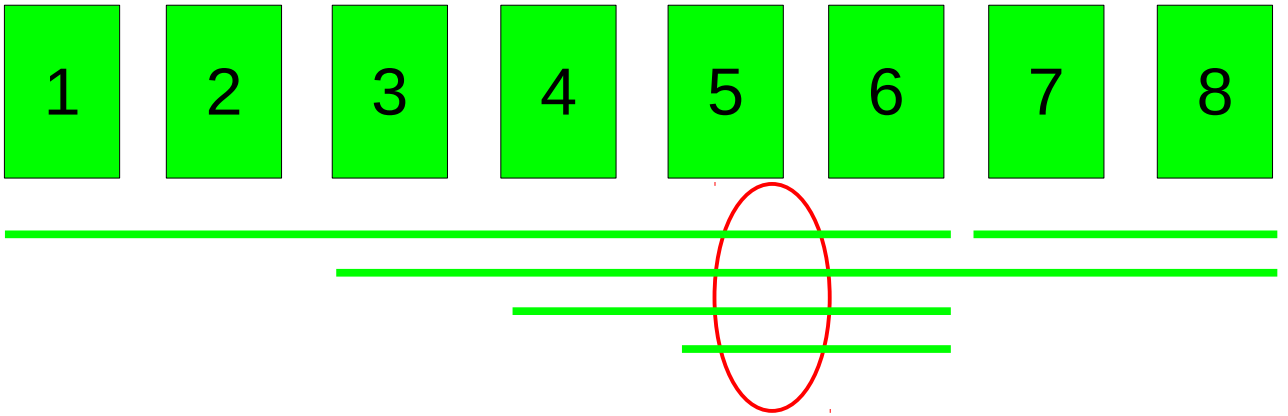
- *Max thickness* $w = 4$
- # of bubblesort passes $= 4$

No repetitions ($r = 1$)



$$\Rightarrow \; Max\ thickness = \#\ of\ bubblesort\ passes = min\ h$$

No repetitions ($r = 1$)



$\Rightarrow$ *Max thickness* $=$ *# of bubblesort passes* $=$ *min* h

Complexity: $N \cdot h$

Eager algorithm playing cards as fast as possible: O(N)

All 3 approaches work for $c > 1$

One color ($c = 1$)

- We describe a lazy algorithm that plays cards as late as possible.

One color ($c = 1$)

- We describe a lazy algorithm that plays cards as late as possible.
  - We store the Last Playable Index for each value.

One color ($c = 1$)

- We describe a lazy algorithm that plays cards as late as possible.
  - We store the Last Playable Index for each value.

Main Lemma

A card with at least $h+1$ larger values having smaller LPIs cannot be played.

One color ($c = 1$)

- We describe a lazy algorithm that plays cards as late as possible.
  - We store the Last Playable Index for each value.

Main Lemma

A card with at least $h+1$ larger values having smaller LPIs cannot be played.

  - *We update the LPI for the value of this card with its previous appearance in the sequence.*

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

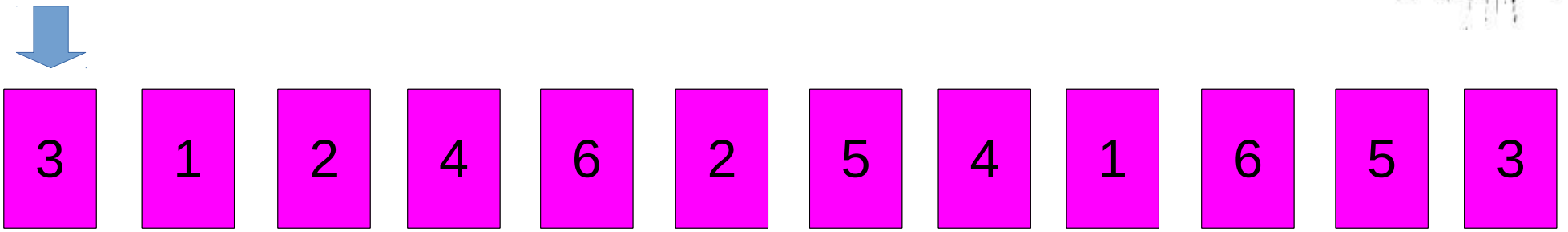
One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Preprocess the instance to create a table P:

Values	1	2	3	4	5	6
Positions						

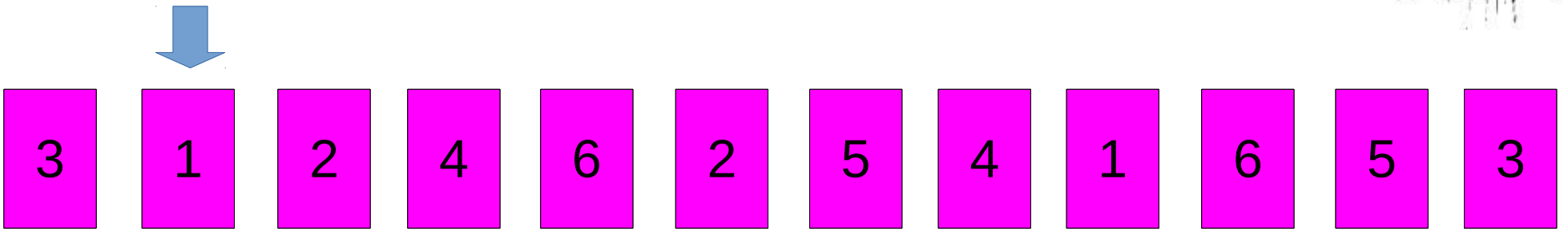
One color ($c = 1$)



- Preprocess the instance to create a table P:

Values	1	2	3	4	5	6
Positions			1			

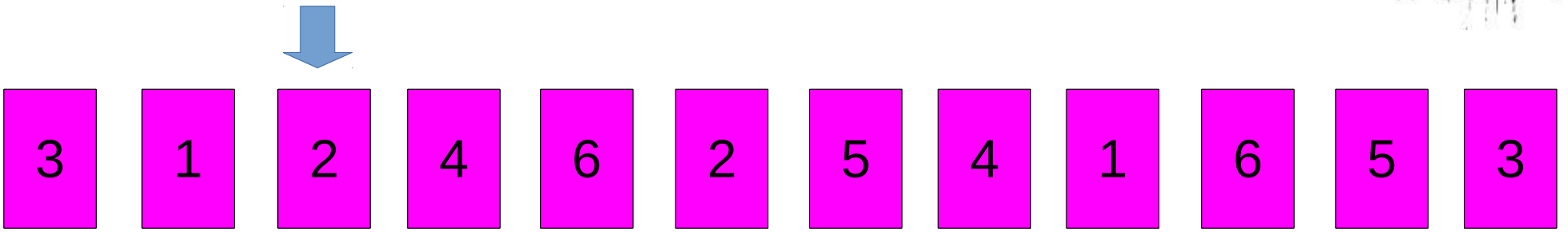
One color ($c = 1$)



- Preprocess the instance to create a table P:

Values	1	2	3	4	5	6
Positions	2		1			

One color ($c = 1$)



- Preprocess the instance to create a table P:

Values	1	2	3	4	5	6
Positions	2	3	1			

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Preprocess the instance to create a table P:

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Use a heap H of size $h+1$, storing the smallest LPIs of larger values (sorted).

($h=2$)

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Initialization of H:

H	10	

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Initialization of H:

H	11	10	

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Initialization of H:

H	11	10	8

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Compare max-value of H to current value's LPI.

H	11	10	8

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Compare max value of H to current value's LPI.
  - If larger, update current value's LPI.

H	11	10	8

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Compare max value of H to current value's LPI.
  - If larger, update current value's LPI.

H	11	10	8

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Compare max value of H to current value's LPI.
  - If smaller, insert it in H (keeping it sorted).

H	11	10	8

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Compare max value of H to current value's LPI.
  - If smaller, insert it in H (keeping it sorted).

H	10	8	1

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Repeat with smaller values

H	10	8	1

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Repeat with smaller values

H	8	6	1

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Repeat with smaller values

H	8	6	1

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Repeat with smaller values

H	8	6	1

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- Repeat with smaller values

H	6	2	1

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

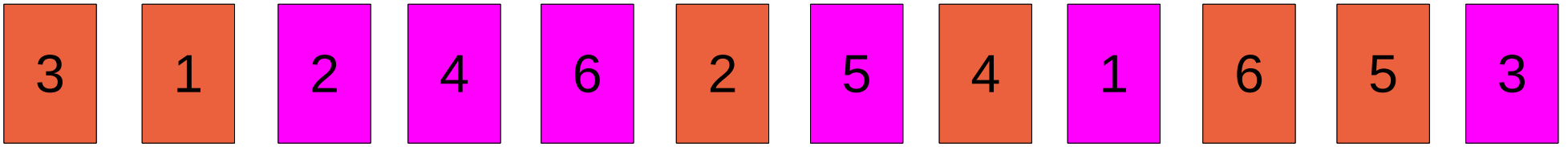
One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- If process terminates then the instance is solvable.

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)



- If process terminates then the instance is solvable.

  Use, for each value, the index the process specified.

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

3	1	2	4	6	2	5	4	1	6	5	3

- If some value runs out of available indices then the instance is not solvable.

Values	1	2	3	4	5	6
Positions	2	3	1	4	7	5
	9	6	12	8	11	10

One color ($c = 1$)

Complexity

- Pre-processing: $O(N)$;
- Heap operations (insertions): $O(\log h)$.

Total running time: $O(N + v \cdot \log h)$

Future Work

- Parameterized complexity?
  - r or h are parameters: paraNP-complete.
  - Dynamic program runs in time exponential in c. $\rightarrow$ W-hard parameterized by c?
  - v is parameter: FPT (it has an exponential kernel running in Nnc). $\rightarrow$ Polynomial kernel?
- Multiplayer game?


Thank you!