On Helping and Stacks

Vitaly Aksenov, INRIA Paris / ITMO University
Petr Kuznetsov, Telecom ParisTech
Anatoly Shalyto, ITMO University

NETYS 2018

Takeaway

- Linearization-based helping [Censor-Hillel et al., 2015]: a process takes a step and fixes the order between two operations.
- Theorem: stack and queue do not have a wait-free help-free implementation using compare&swap and fetch&add.
  - Proved for exact order types.
  - Stack is not exact order type.
  - We give a direct proof for stack.

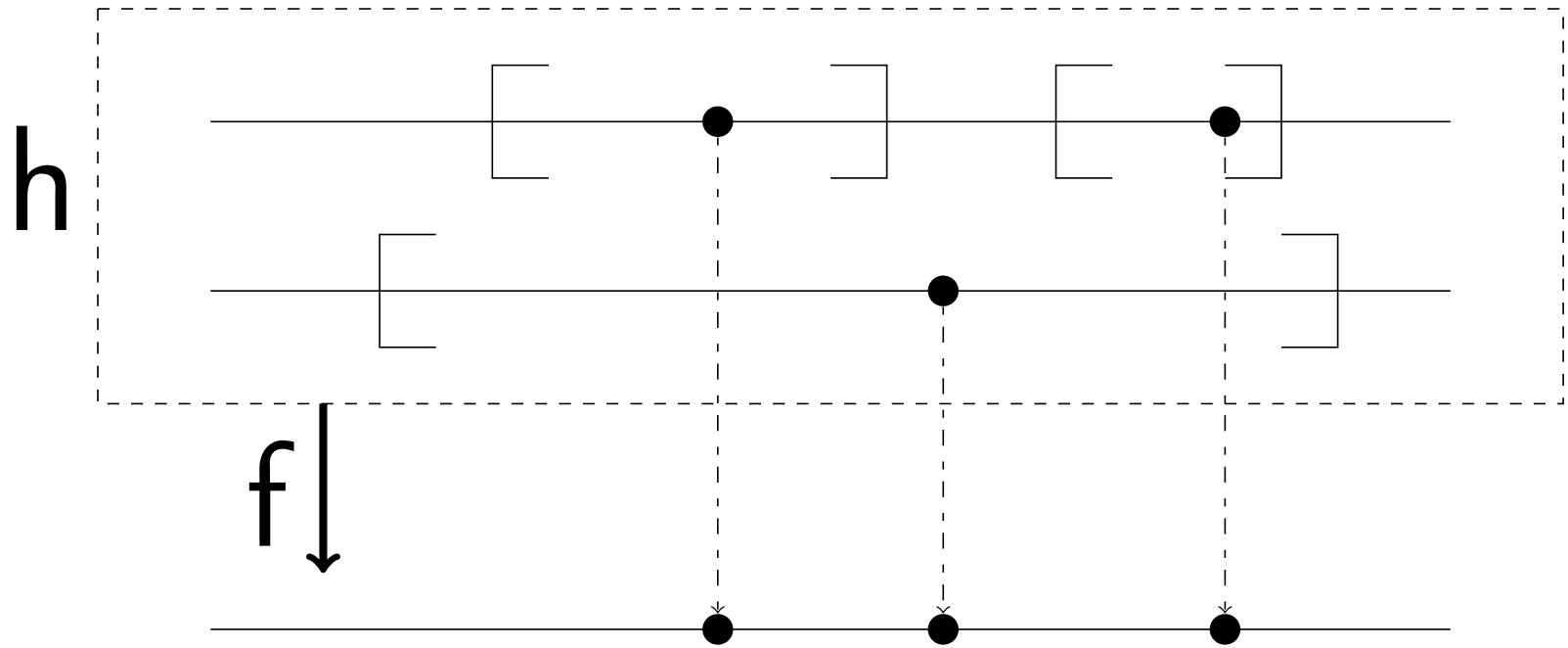
Outline

Outline

What is helping?

- Intuitively: an operation performs some work on behalf of another.
- Often appears in wait-free and lock-free algorithms.
- Examples:
  - Universal wait-free construction [Herlihy, 1991];
  - Lock-free binary search tree [Ellen et al., 2010].
- How to define helping formally?
  - Linearization-based helping.

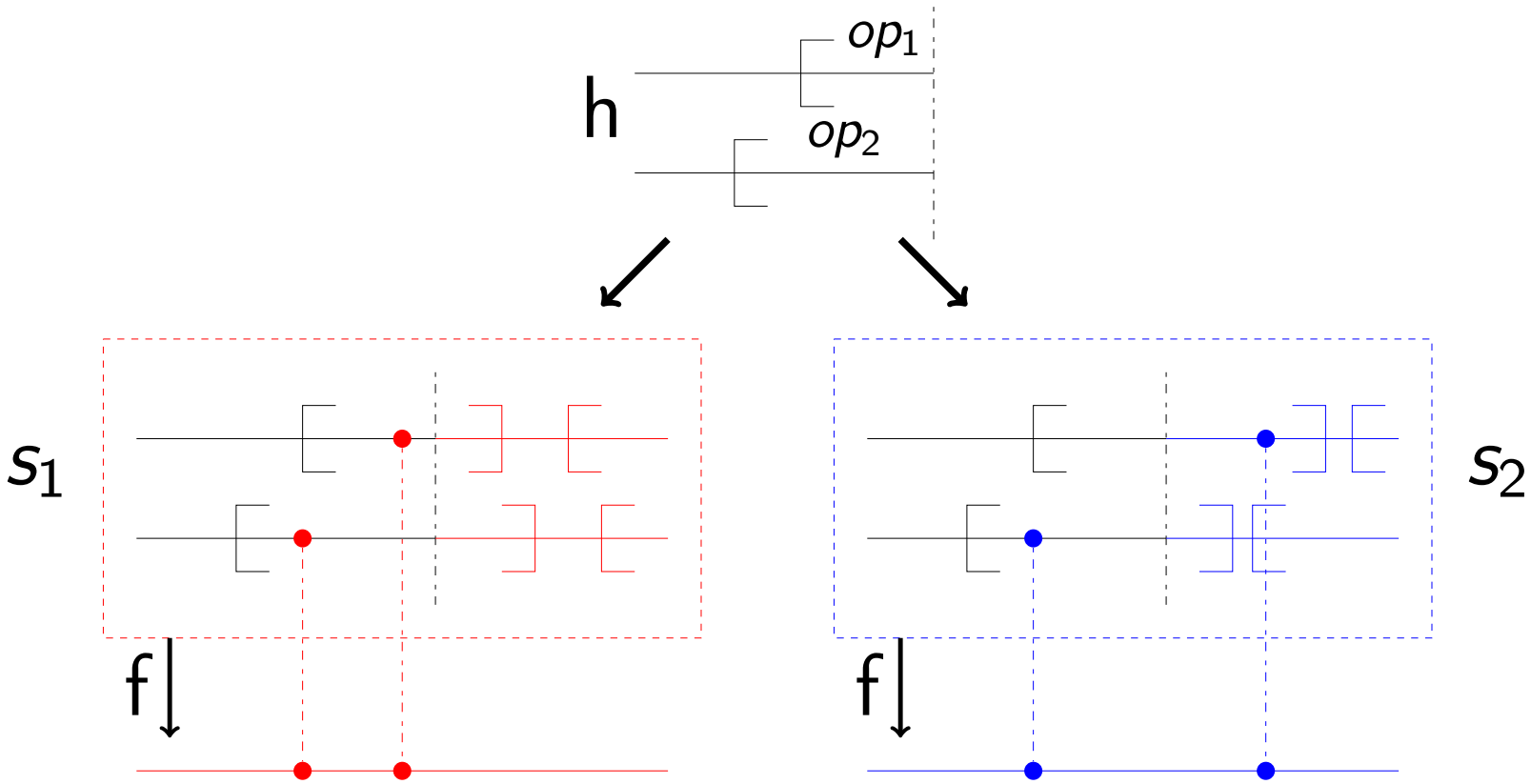
Linearization function



Helping. Decided before

Definition 1

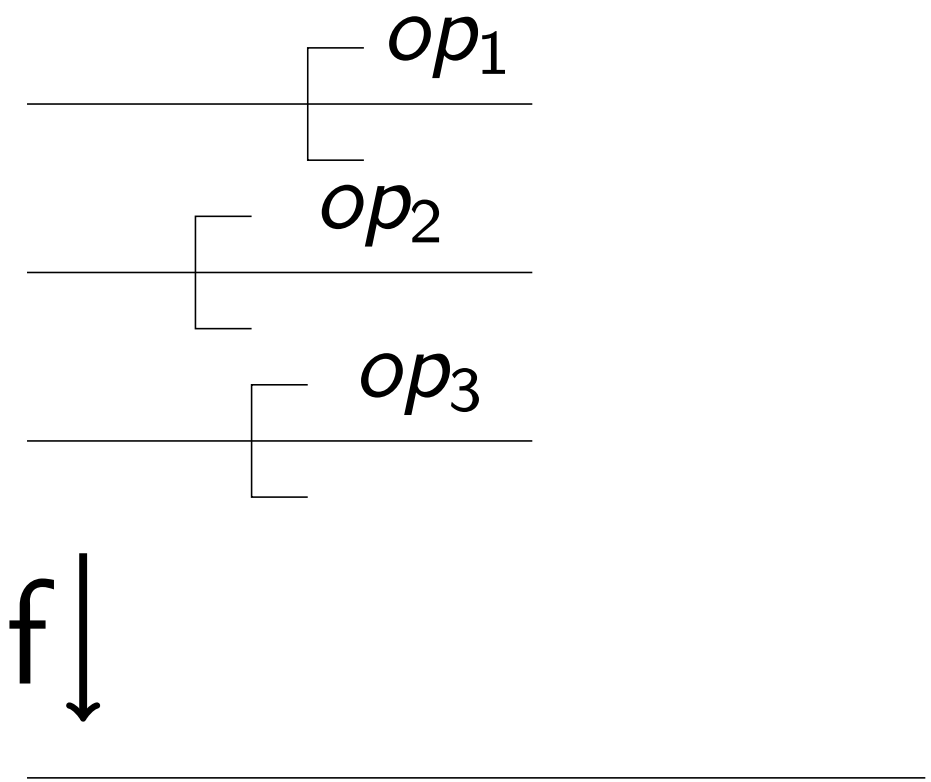
We say that op_1 *is decided before* op_2 *in* h with respect to f, if there exists no extension s of h such that $op_2 \prec_{f(s)} op_1$.



Helping. Definition

Definition 2

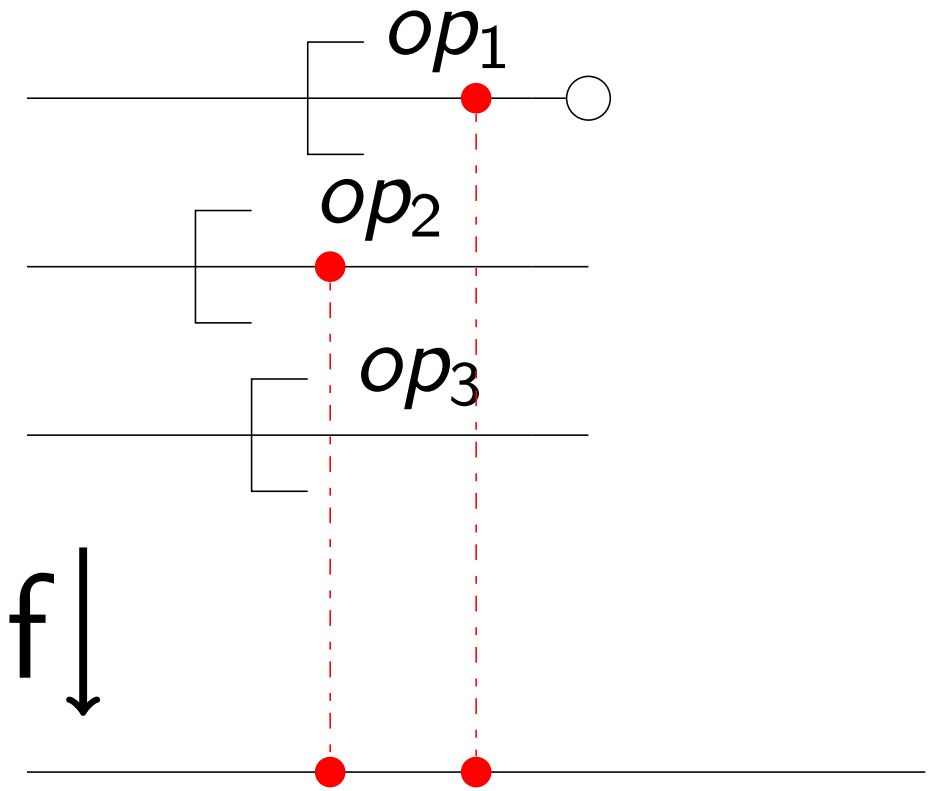
An implementation has helping if op_1 by p_1 is decided before op_2 by p_2 after a step of p_3.



Helping. Definition

Definition 2

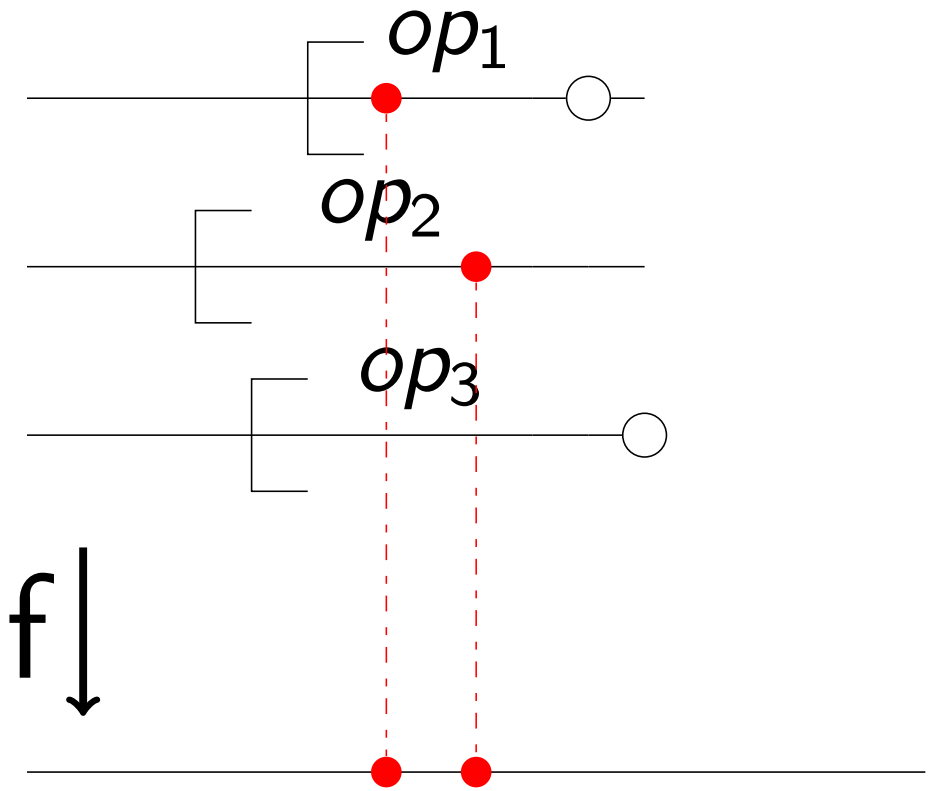
An implementation has helping if op_1 by p_1 is decided before op_2 by p_2 after a step of p_3.



Helping. Definition

Definition 2

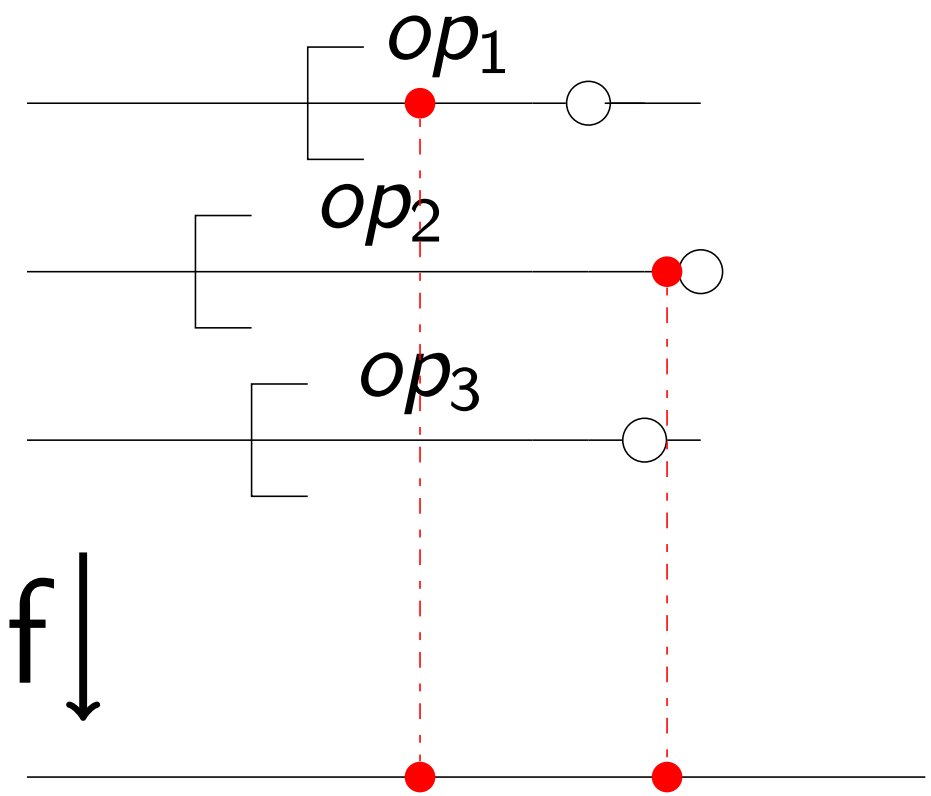
An implementation has helping if op_1 by p_1 is decided before op_2 by p_2 after a step of p_3.



Helping. Definition

Definition 2

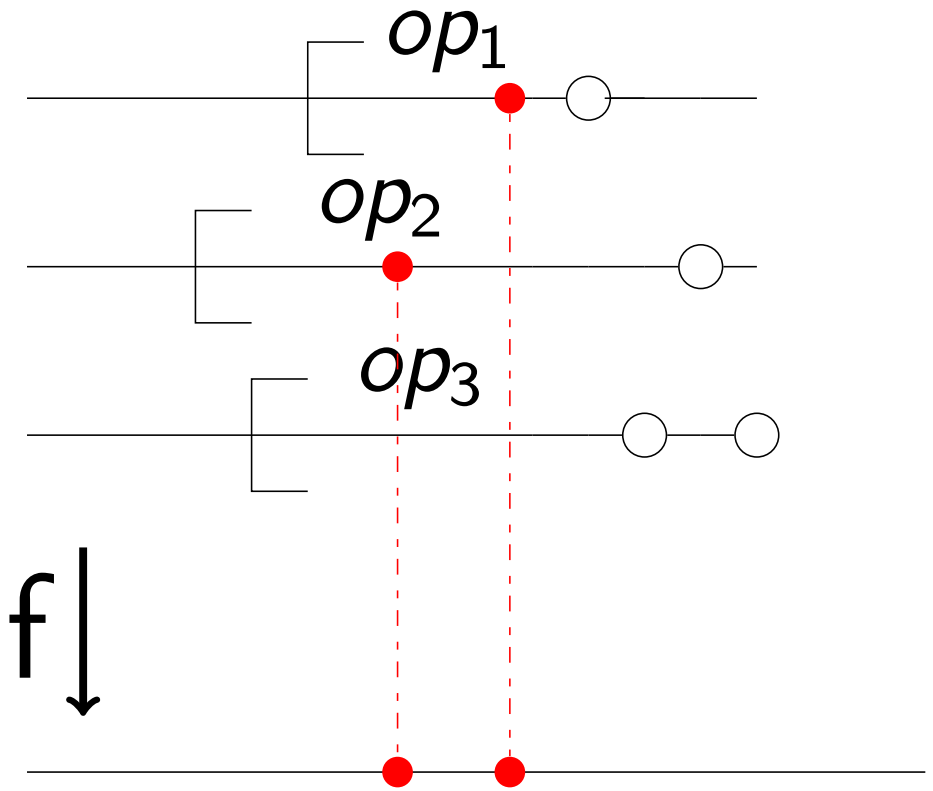
An implementation has helping if op_1 by p_1 is decided before op_2 by p_2 after a step of p_3.



Helping. Definition

Definition 2

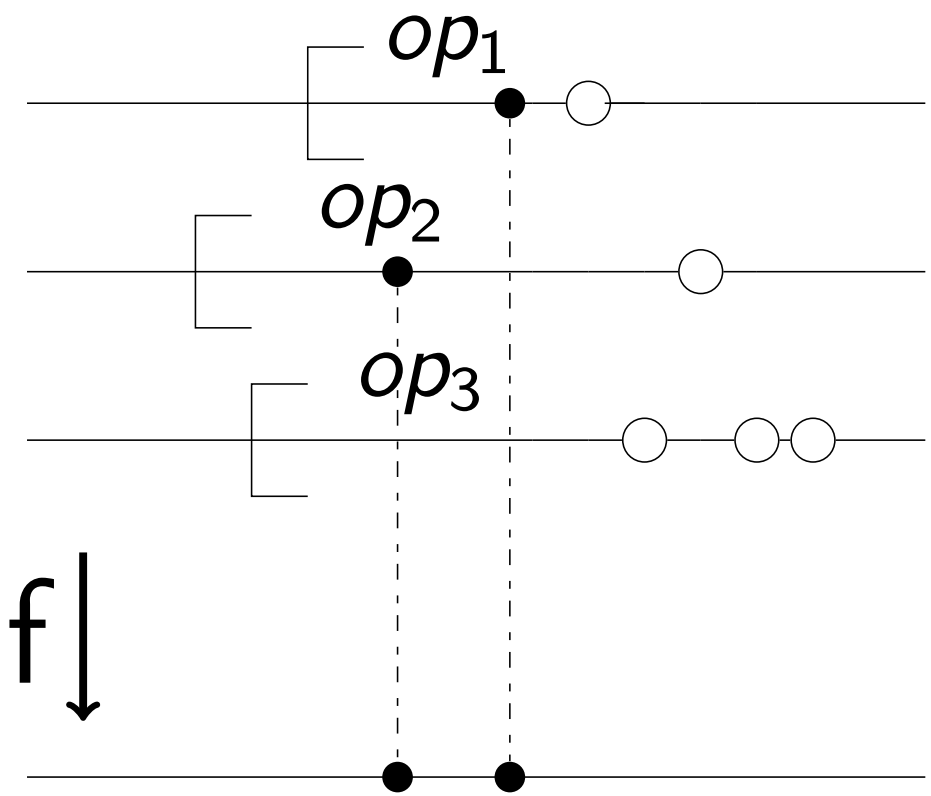
An implementation has helping if op_1 by p_1 is decided before op_2 by p_2 after a step of p_3.



Helping. Definition

Definition 2

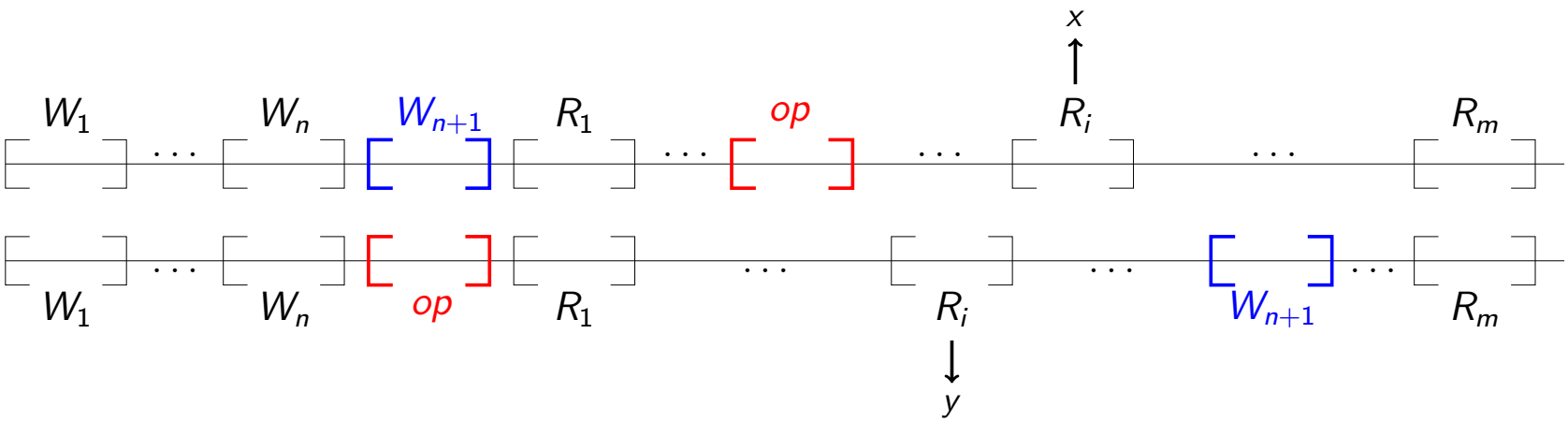
An implementation has helping if op_1 by p_1 is decided before op_2 by p_2 after a step of p_3.



Outline

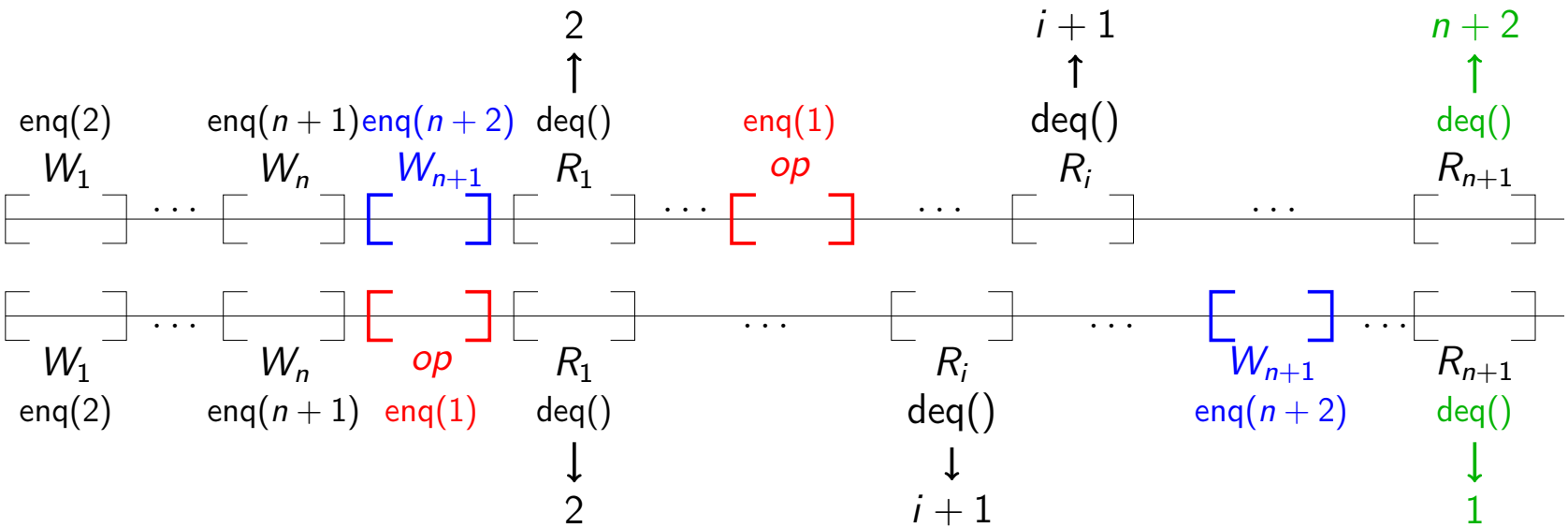
Exact Order Type

There exist an operation op, an infinite sequence W and a sequence R such that for any n there exists m...



Exact Order Type. Queue

- $op = \mathrm{enq}(1)$, $W_i = \mathrm{enq}(i+1)$ and $R_i = \mathrm{deq}()$.
- Given n, we choose $m = n + 1$.



Exact Order Type

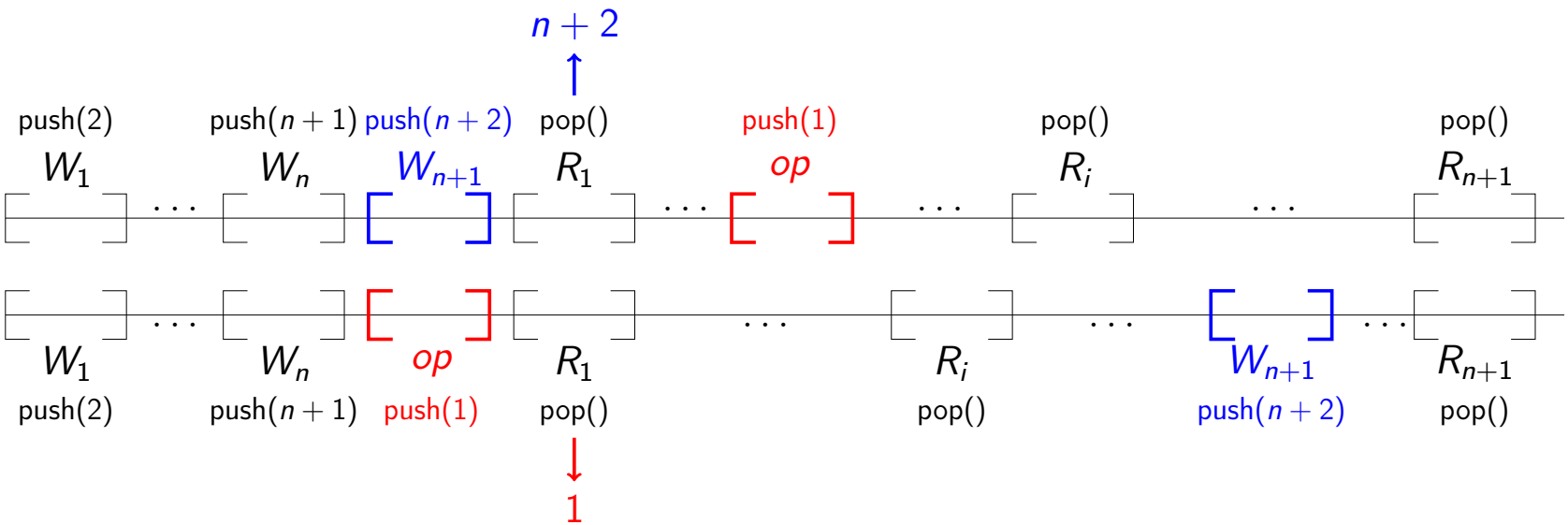
Theorem 3 (Censor-Hillel et al., 2015)

Any exact order type does not have a wait-free help-free implementation in systems with compare&swap and fetch&add primitives.

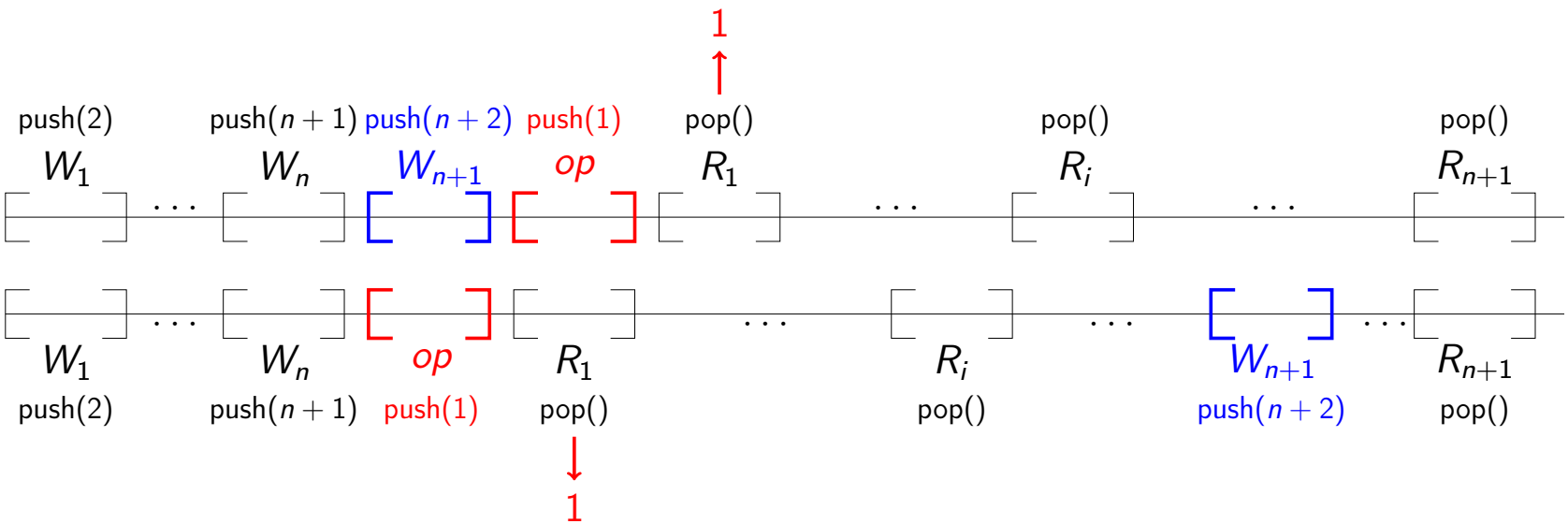
Theorem 4

*Stack is **not** exact order type.*

Stack is not Exact Order Type

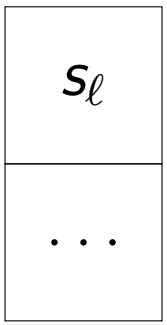


Stack is not Exact Order Type

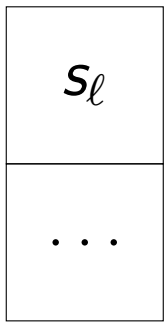


Stack is not Exact Order Type. Construction

$W(n)$



$W(n)$



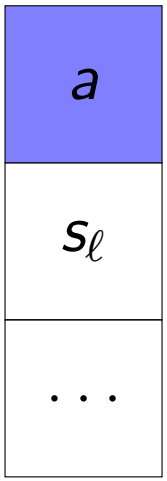
Stack is not Exact Order Type. Construction

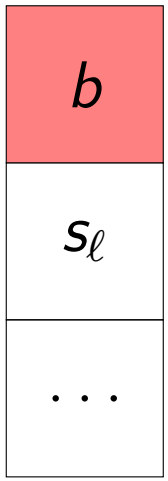
$\text{push}(a)$

$W(n) \circ W_{n+1}$

$\text{push}(b)$

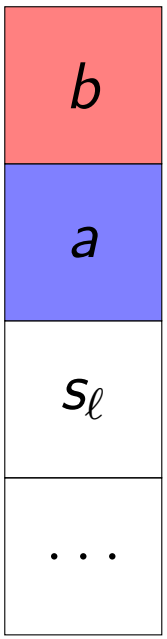
$W(n) \circ op$



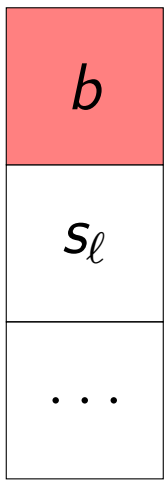


Stack is not Exact Order Type. Construction

$$W(n) \circ W_{n+1} \circ \overset{\text{push}(b)}{op}$$

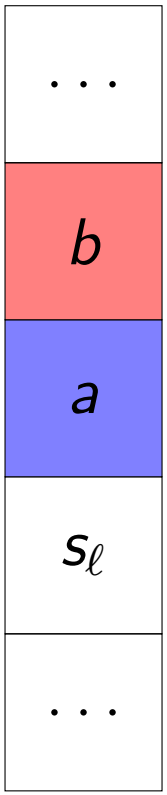


$$W(n) \circ op$$

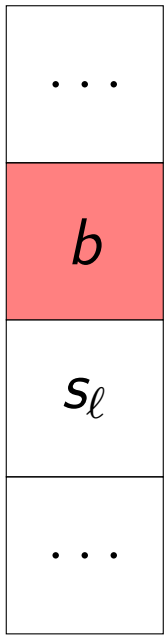


Stack is not Exact Order Type. Construction

$W(n) \circ W_{n+1} \circ op \circ R(x)$

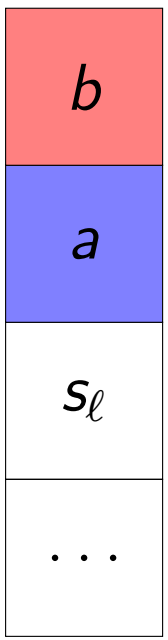


$W(n) \circ op \circ R(x)$

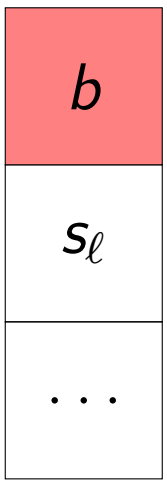


Stack is not Exact Order Type. Construction

$W(n) \circ W_{n+1} \circ op \circ R(y-1)$



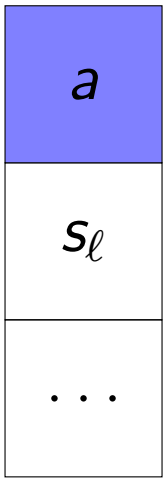
$W(n) \circ op \circ R(y-1)$

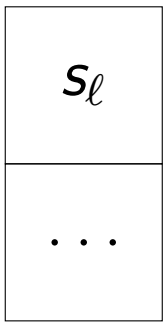


Stack is not Exact Order Type. Construction

$W(n) \circ W_{n+1} \circ op \circ R(y)$

$W(n) \circ op \circ R(y)$



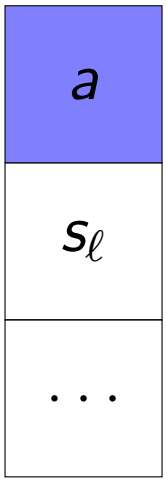


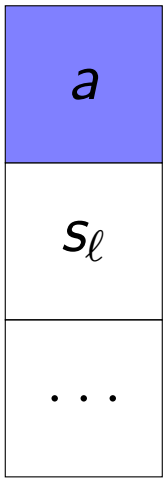
Stack is not Exact Order Type. Construction

$W(n) \circ W_{n+1} \circ op \circ R(y)$

$W(n) \circ op \circ R(y) \circ W_{n+1}$

push(a)





Outline

Proof

Theorem 5

Stack does not have a wait-free help-free implementation in a system with at least three processes and with compare&swap primitive.

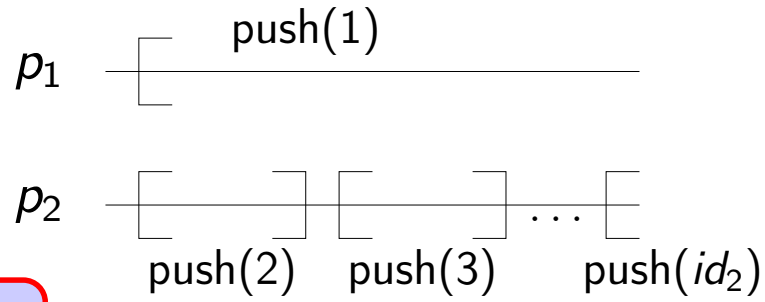
- Programs:
  - first process — $\texttt{push}(1)$
  - second process — $\texttt{push}(2) \circ \texttt{push}(3) \circ \ldots$
  - third process — $\texttt{pop}() \circ \texttt{pop}() \circ \ldots$
- We build a history in which the first process makes infinite number of steps, but never finishes.

Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5    op2 ← push(id2)
6    while true:
7      if op1 is not decided before op2 in h ∘ p1:
8        h ← h ∘ p1
9        continue
10     if op2 is not decided before op1 in h ∘ p2:
11       h ← h ∘ p2
12       continue
13     break
14   h ← h ∘ p2
15   h ← h ∘ p1
16   while op2 is not completed:
17     h ← h ∘ p2
18   id2 ← id2 + 1
```

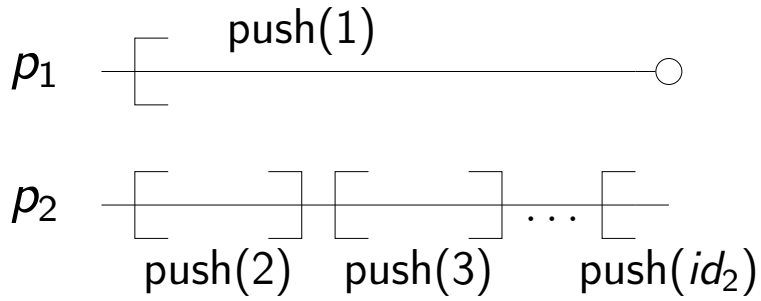
Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5    op2 ← push(id2)
6    while true:
7      if op1 is not decided before op2 in h ∘ p1:
8        h ← h ∘ p1
9        continue
10     if op2 is not decided before op1 in h ∘ p2:
11       h ← h ∘ p2
12       continue
13     break
14   h ← h ∘ p2
15   h ← h ∘ p1
16   while op2 is not completed:
17     h ← h ∘ p2
18   id2 ← id2 + 1
```



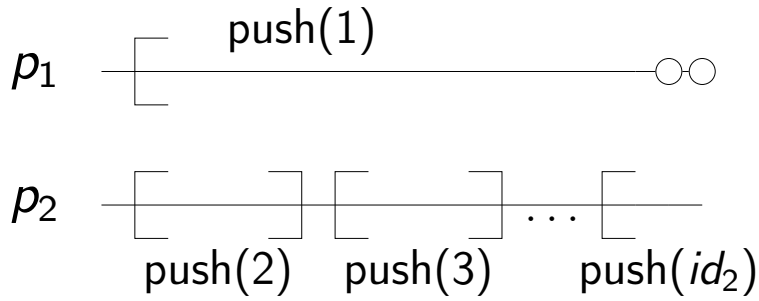
Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5    op2 ← push(id2)
6    while true:
7      if op1 is not decided before op2 in h ∘ p1:
8        h ← h ∘ p1
9        continue
10     if op2 is not decided before op1 in h ∘ p2:
11       h ← h ∘ p2
12       continue
13     break
14   h ← h ∘ p2
15   h ← h ∘ p1
16   while op2 is not completed:
17     h ← h ∘ p2
18   id2 ← id2 + 1
```



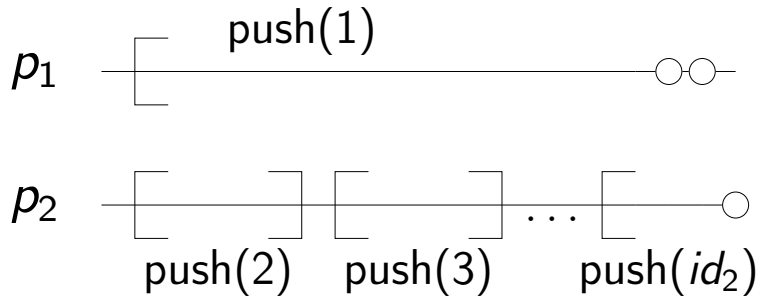
Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5    op2 ← push(id2)
6    while true:
7      if op1 is not decided before op2 in h ∘ p1:
8        h ← h ∘ p1
9        continue
10     if op2 is not decided before op1 in h ∘ p2:
11       h ← h ∘ p2
12       continue
13     break
14   h ← h ∘ p2
15   h ← h ∘ p1
16   while op2 is not completed:
17     h ← h ∘ p2
18   id2 ← id2 + 1
```



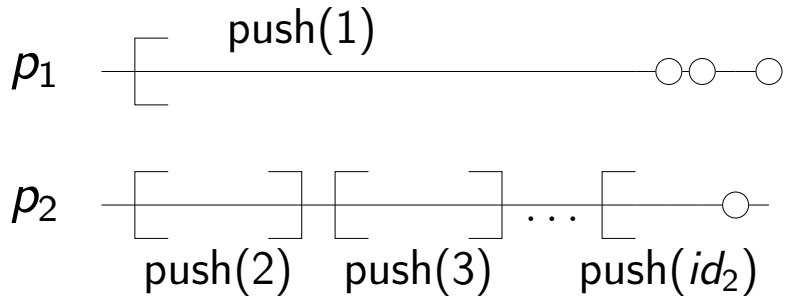
Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5    op2 ← push(id2)
6    while true:
7      if op1 is not decided before op2 in h ∘ p1:
8        h ← h ∘ p1
9        continue
10     if op2 is not decided before op1 in h ∘ p2:
11       h ← h ∘ p2
12       continue
13     break
14   h ← h ∘ p2
15   h ← h ∘ p1
16   while op2 is not completed:
17     h ← h ∘ p2
18   id2 ← id2 + 1
```



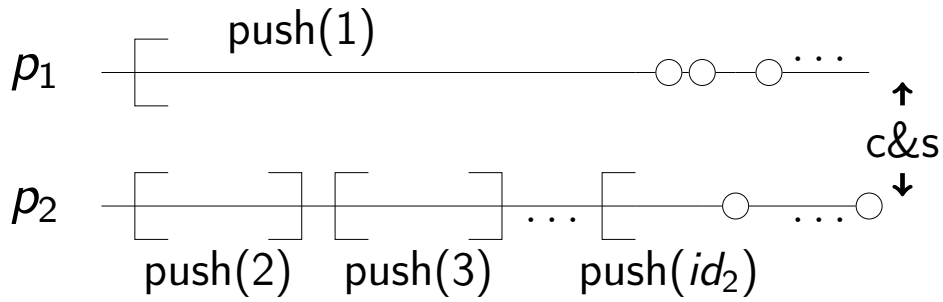
Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5    op2 ← push(id2)
6    while true:
7      if op1 is not decided before op2 in h ∘ p1:
8        h ← h ∘ p1
9        continue
10     if op2 is not decided before op1 in h ∘ p2:
11       h ← h ∘ p2
12       continue
13     break
14   h ← h ∘ p2
15   h ← h ∘ p1
16   while op2 is not completed:
17     h ← h ∘ p2
18   id2 ← id2 + 1
```



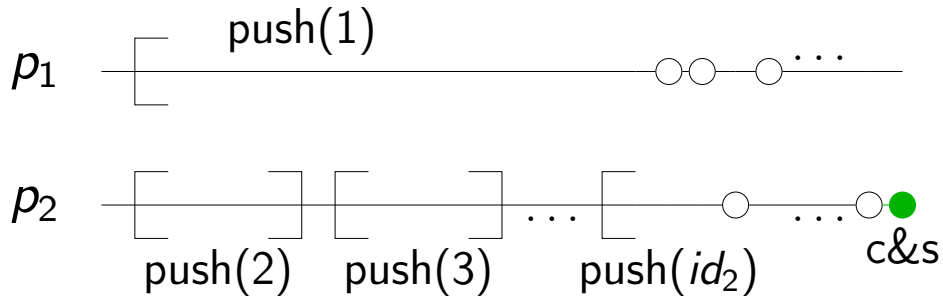
Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5    op2 ← push(id2)
6    while true:
7      if op1 is not decided before op2 in h ∘ p1:
8        h ← h ∘ p1
9        continue
10     if op2 is not decided before op1 in h ∘ p2:
11       h ← h ∘ p2
12       continue
13     break
14   h ← h ∘ p2
15   h ← h ∘ p1
16   while op2 is not completed:
17     h ← h ∘ p2
18   id2 ← id2 + 1
```



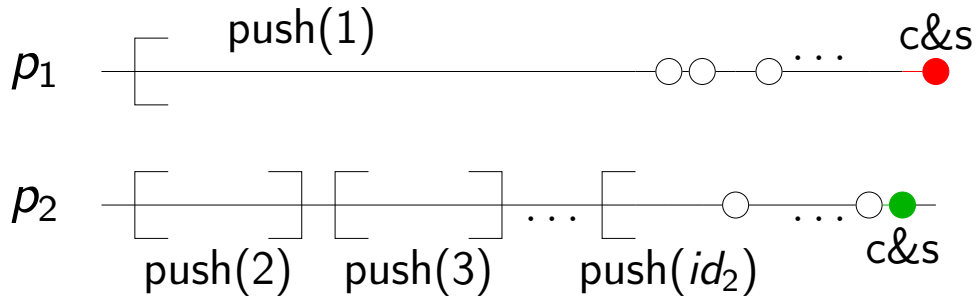
Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5     op2 ← push(id2)
6     while true:
7        if op1 is not decided before op2 in h ∘ p1:
8           h ← h ∘ p1
9           continue
10       if op2 is not decided before op1 in h ∘ p2:
11          h ← h ∘ p2
12          continue
13       break
14    h ← h ∘ p2
15    h ← h ∘ p1
16    while op2 is not completed:
17       h ← h ∘ p2
18    id2 ← id2 + 1
```



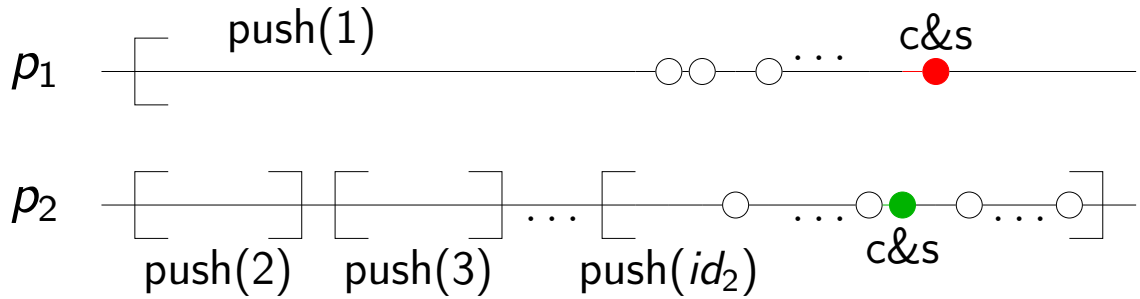
Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5     op2 ← push(id2)
6     while true:
7        if op1 is not decided before op2 in h ∘ p1:
8           h ← h ∘ p1
9           continue
10       if op2 is not decided before op1 in h ∘ p2:
11          h ← h ∘ p2
12          continue
13       break
14    h ← h ∘ p2
15    h ← h ∘ p1
16    while op2 is not completed:
17       h ← h ∘ p2
18    id2 ← id2 + 1
```



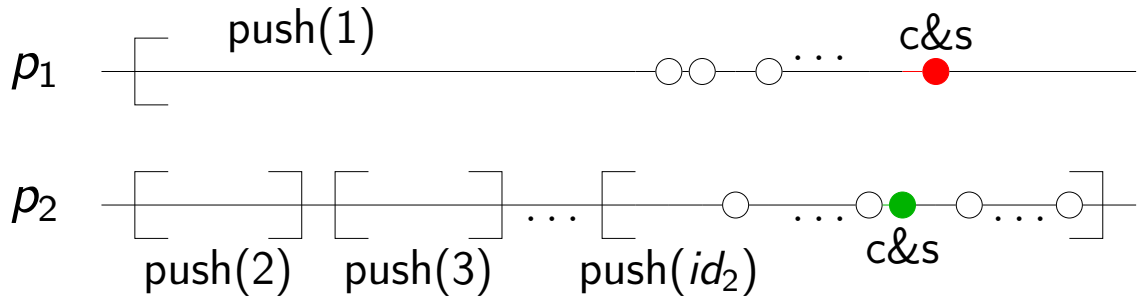
Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5    op2 ← push(id2)
6    while true:
7      if op1 is not decided before op2 in h ∘ p1:
8        h ← h ∘ p1
9        continue
10     if op2 is not decided before op1 in h ∘ p2:
11       h ← h ∘ p2
12       continue
13     break
14   h ← h ∘ p2
15   h ← h ∘ p1
16   while op2 is not completed:
17     h ← h ∘ p2
18   id2 ← id2 + 1
```



Proof. History Construction

```
1  h ← ϵ
2  op1 ← push(1)
3  id2 ← 2
4  while true:
5    op2 ← push(id2)
6    while true:
7      if op1 is not decided before op2 in h ∘ p1:
8        h ← h ∘ p1
9        continue
10     if op2 is not decided before op1 in h ∘ p2:
11       h ← h ∘ p2
12       continue
13     break
14   h ← h ∘ p2
15   h ← h ∘ p1
16   while op2 is not completed:
17     h ← h ∘ p2
18   id2 ← id2 + 1
```



Outline

Conclusion

Theorem 6

There does not exist a help-free wait-free implementation of stack in systems:

- *with at least three processes and compare&swap primitive;*
- *with at least four processes, and compare&swap and fetch&add primitives.*

- We showed that the undirect proof of the theorem provided by Censor-Hillel et al. does not work for stack.
- We provide the direct proof of the theorem for stack.

Related work

- Common2 class of objects. Common2 contains stack. What about queue?
- Valency-based helping by [Attiya et al., 2016].
- Is there any other useful helping formalizations?

Questions

Thank you for attention!