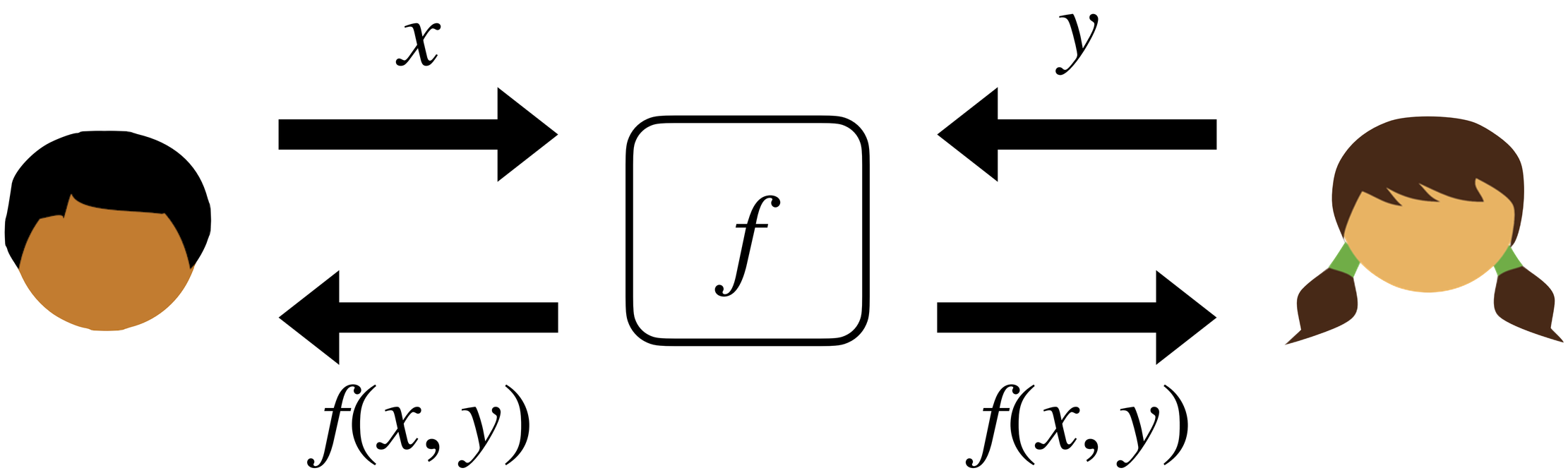
EpiGRAM

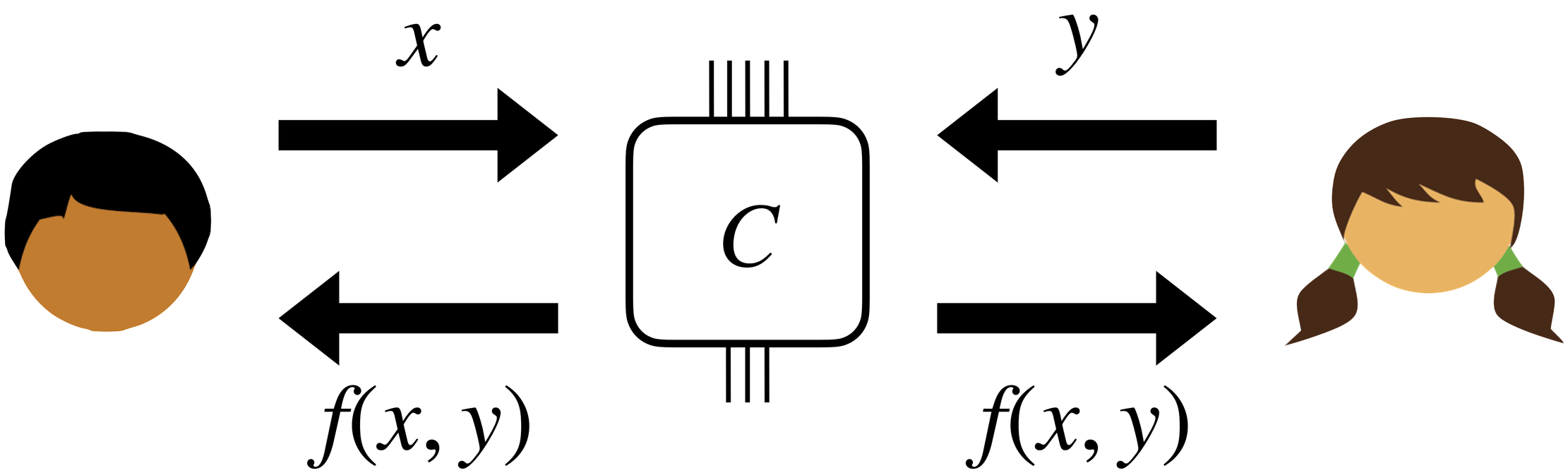
Practical Garbled RAM

David Heath — Georgia Tech ⇒ UIUC
Vladimir Kolesnikov — Georgia Tech
Rafail Ostrovsky — UCLA

Secure 2PC



Garbled Circuits



Garbled Circuits

Constant Round ✓

Garbled Circuits

Constant Round ✓

Symmetric Key Primitives ✓

Garbled Circuits

Constant Round ✓

Symmetric Key Primitives ✓

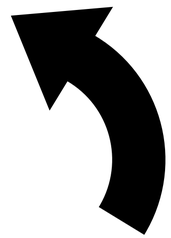
Circuits ✗

Garbled Circuits

Constant Round ✓

Symmetric Key Primitives ✓

Circuits ✗



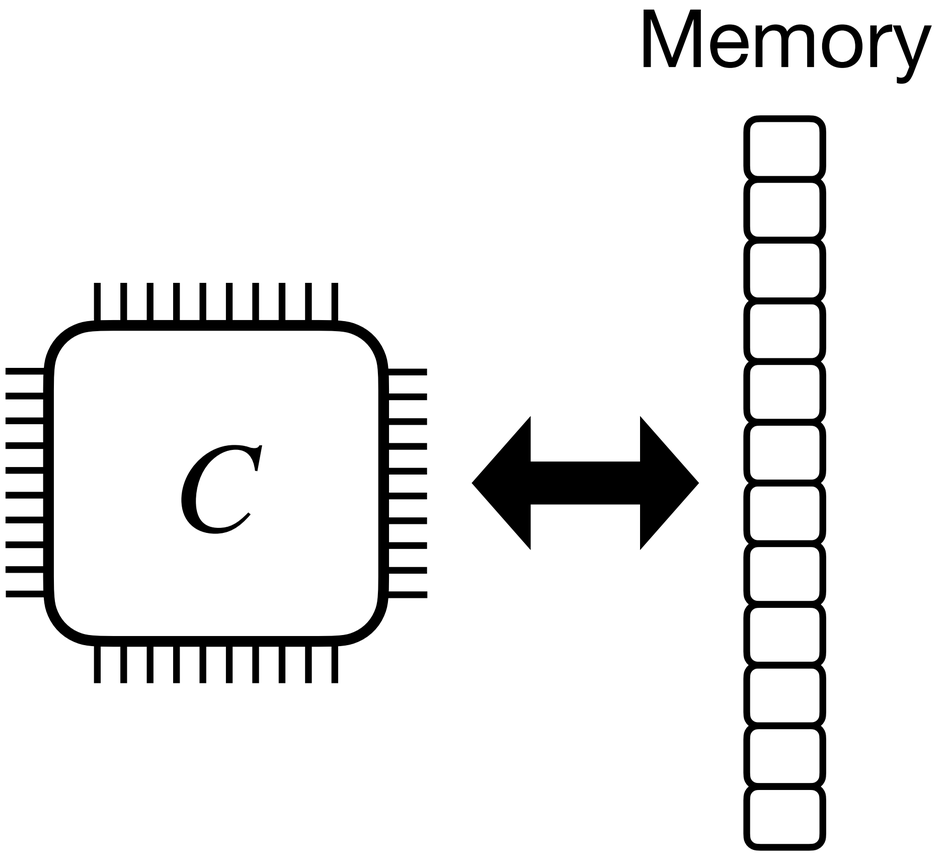
Cost scales w/ # gates

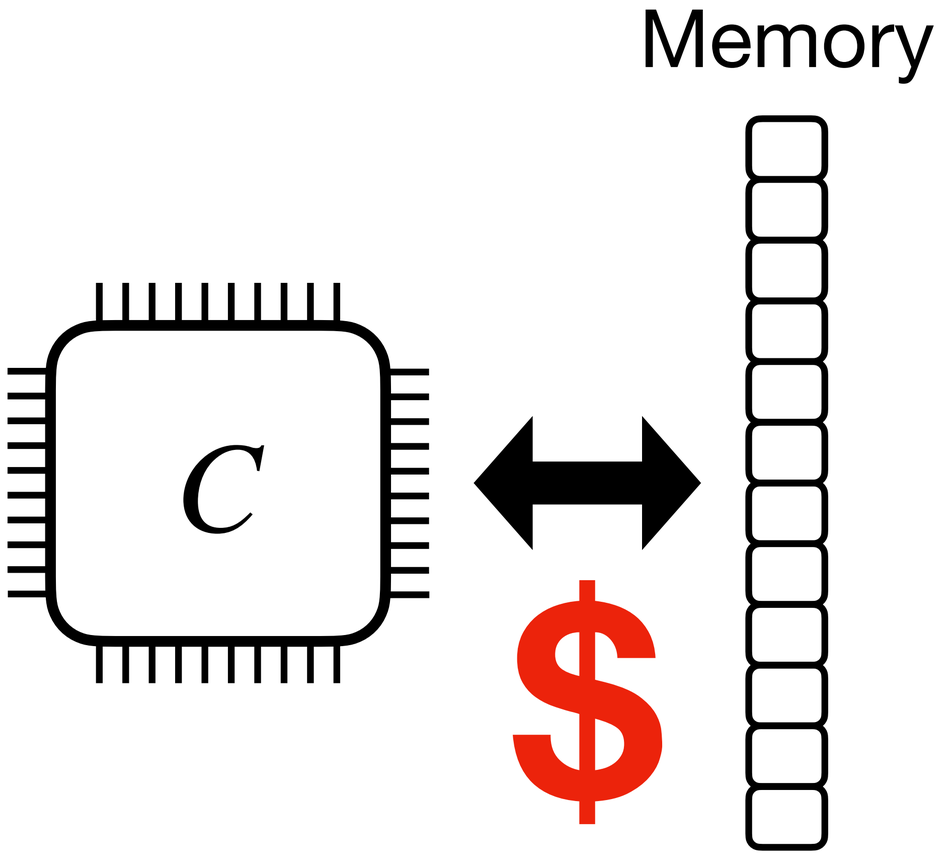
[LO13] Garbled Circuits RAM

Constant Round ✓

Symmetric Key Primitives ✓

Circuits ✗


Memory
C



Goal: Amortized $O(\text{poly}(\log n))$

[LO13] Garbled Circuits RAM

Constant Round ✓

Symmetric Key Primitives ✓

RAM Machines ~~Circuits~~ ✓

[LO13] Garbled Circuits ~~Garbled Circuits~~ RAM

Constant Round ✓

Symmetric Key Primitives ✓

RAM Machines ~~Circuits~~ ✓

Non-BB Crypto

Non-Standard Assumptions ✗

High Concrete Overhead ✗

Garbled Circuits RAM

Constant Round ✓

Symmetric Key Primitives ✓

RAM Machines ~~Circuits~~ ✓

Non-Standard Assumptions ✗

High Concrete Overhead ✗

Non-BB Crypto

[GHL+14, GLO15, ...]

Garbled Circuits RAM

Constant Round ✓

Symmetric Key Primitives ✓

RAM Machines ~~Circuits~~ ✓

~~Non-BB Crypto~~ ~~Non-Standard Assumptions~~ ✗

High Concrete Overhead ✗

[GHL+14, GLO15, ...]

Garbled Circuits RAM

Constant Round ✓

Symmetric Key Primitives ✓

RAM Machines ~~Circuits~~ ✓

Standard Assumptions ✓

High Concrete Overhead ✗

[GLO15] $\tilde{O}(\kappa^2 \cdot \log^4 n \cdot (w + \log^2 n))$

Hidden $\mathrm{poly}(\log \log n)$ factors

Analysis due to [PLS22]

Ours $O(\kappa \cdot \log^2 n \cdot (w + \log^2 n))$

κ Security parameter

w Size of RAM elements

n Number of RAM elements

[GLO15] $\tilde{O}(\kappa^2 \cdot \log^4 n \cdot (w + \log^2 n))$

Hidden $\mathrm{poly}(\log \log n)$ factors

Ours $O(\kappa \cdot \log^2 n \cdot (w + \log^2 n))$

κ	Security parameter
w	Size of RAM elements
n	Number of RAM elements

[GLO15] $\tilde{O}(\kappa^2 \cdot \log^4 n \cdot (w + \log^2 n))$

Hidden $\mathrm{poly}(\log \log n)$ factors

Ours $O(\kappa \cdot \log^2 n \cdot (w + \log^2 n))$

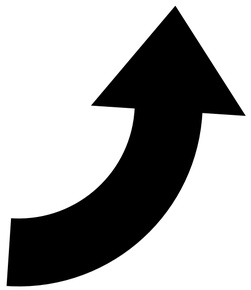
κ Security parameter

w Size of RAM elements

n Number of RAM elements

[GLO15] $\tilde{O}(\kappa^2 \cdot \log^4 n \cdot (w + \log^2 n))$

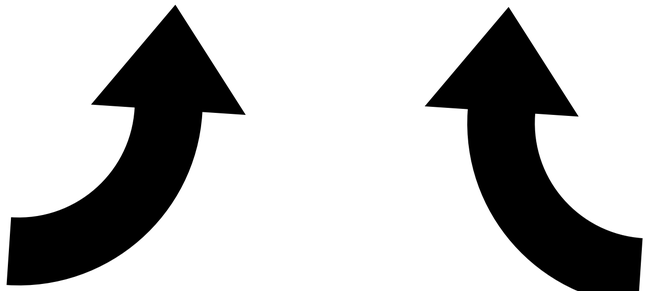
Ours $O(\kappa \cdot \log^2 n \cdot (w + \log^2 n))$



Improved GC
encoding of GC labels

[GLO15] $\tilde{O}(\kappa^2 \cdot \log^4 n \cdot (w + \log^2 n))$

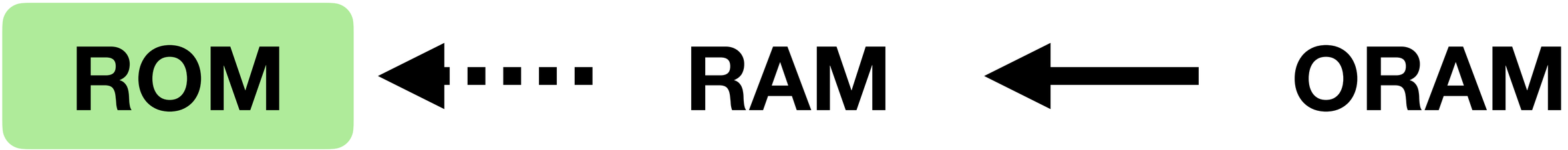
Ours $O(\kappa \cdot \log^2 n \cdot (w + \log^2 n))$



Improved GC encoding of GC labels

Clean GC data structure

Core Technical Challenge



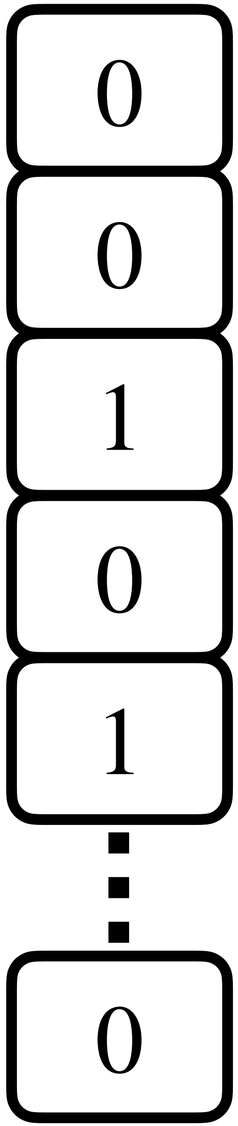
See paper

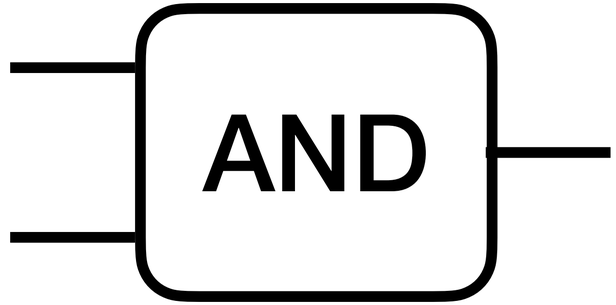
[GO96,…]

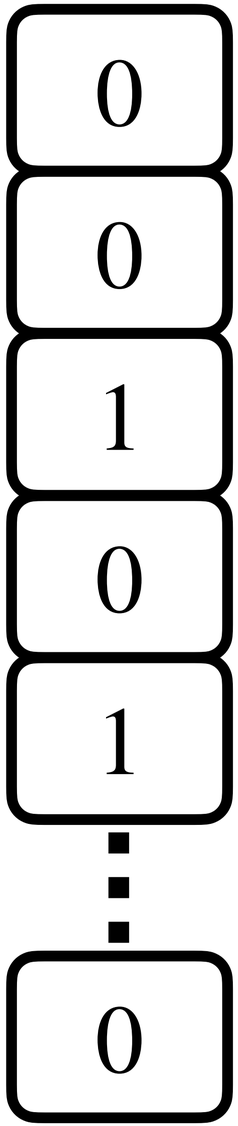
Core Technical Challenge

Read <u>once</u> only memory ◀⋯⋯ **RAM** ◀— **ORAM**

See paper [GO96,…]

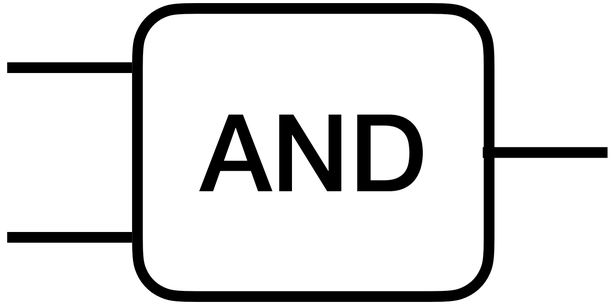

0
0
1
0
1
0

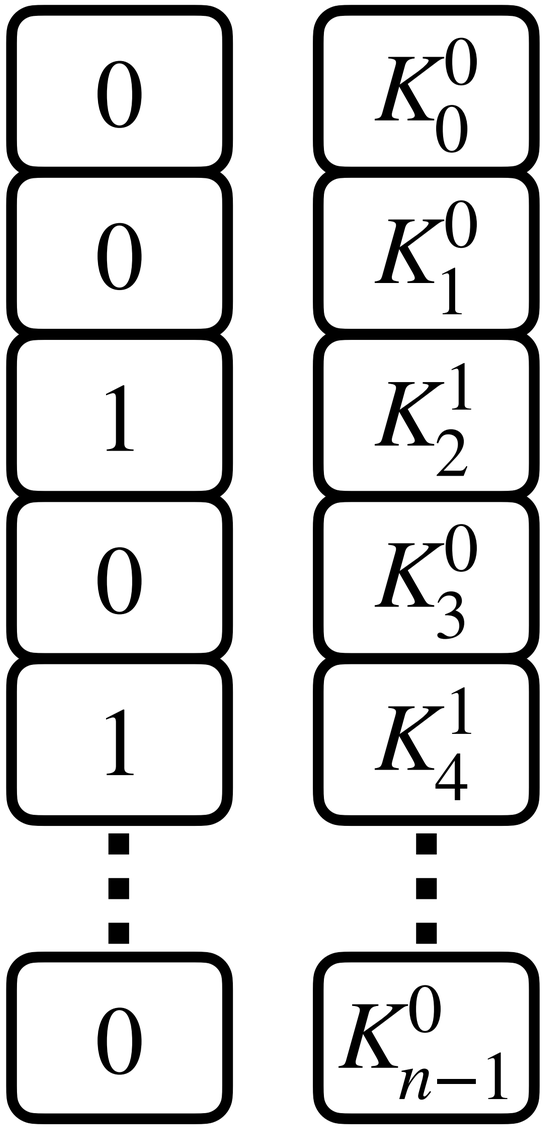

AND



Language

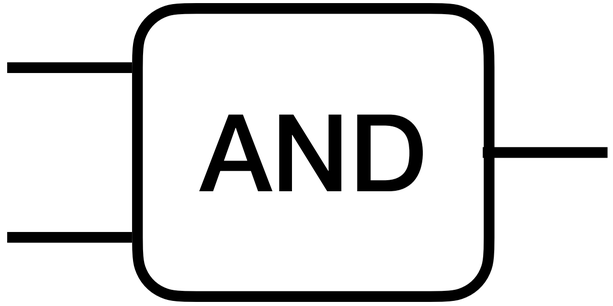
K^0, K^1

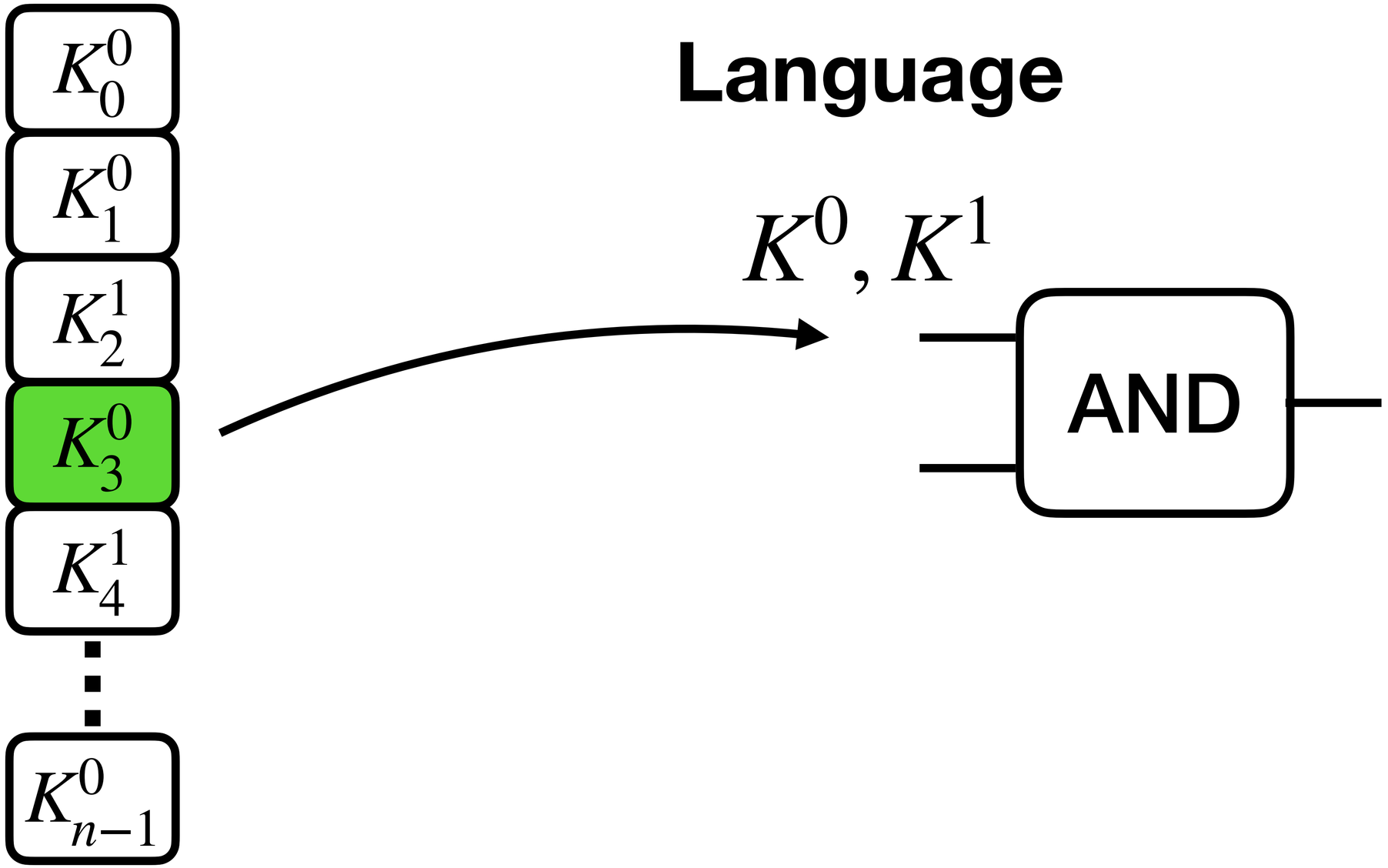


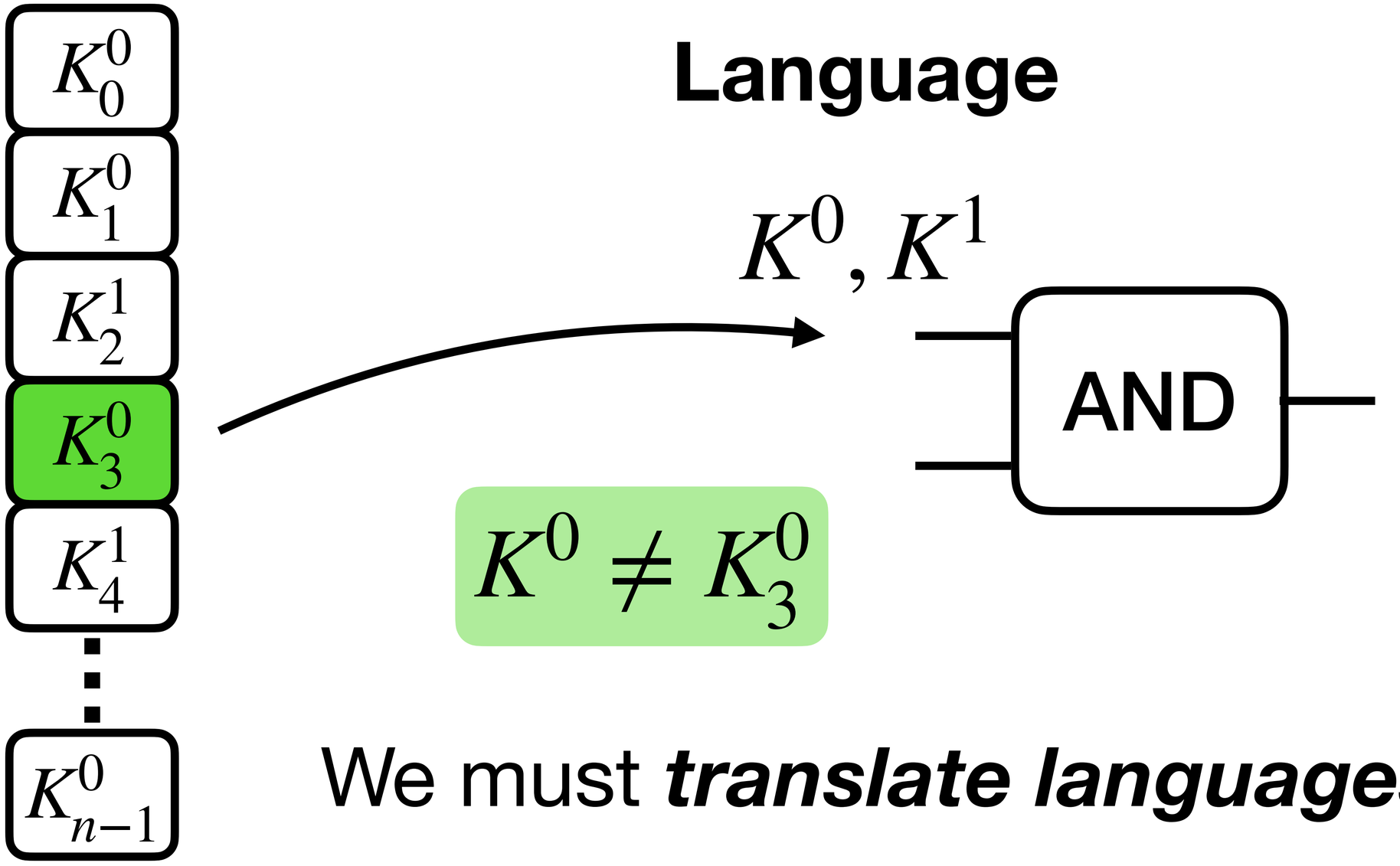


Language

K^0, K^1



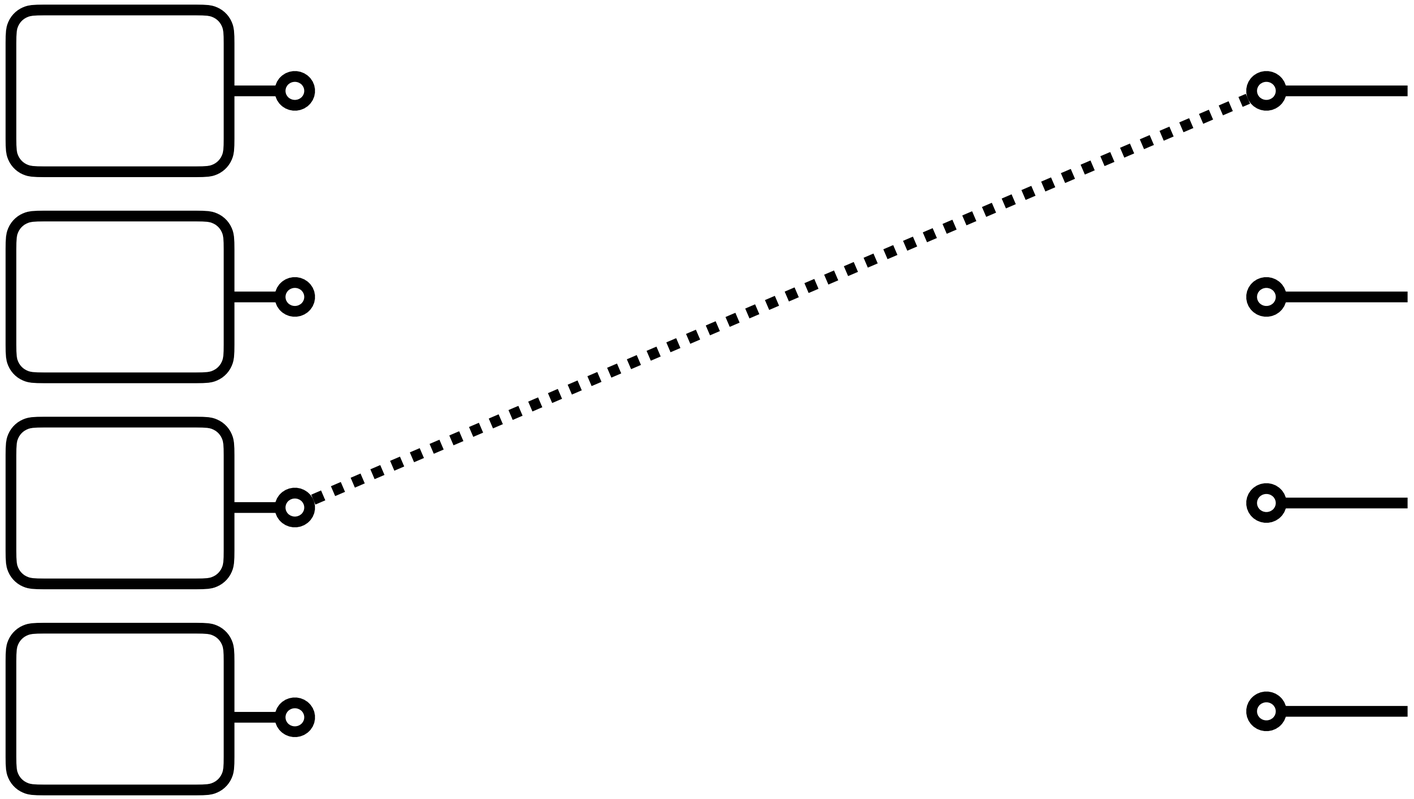

K_0^0
K_1^0
K_2^1
K_3^0
K_4^1
K_{n-1}^0
Language
K^0, K^1
AND



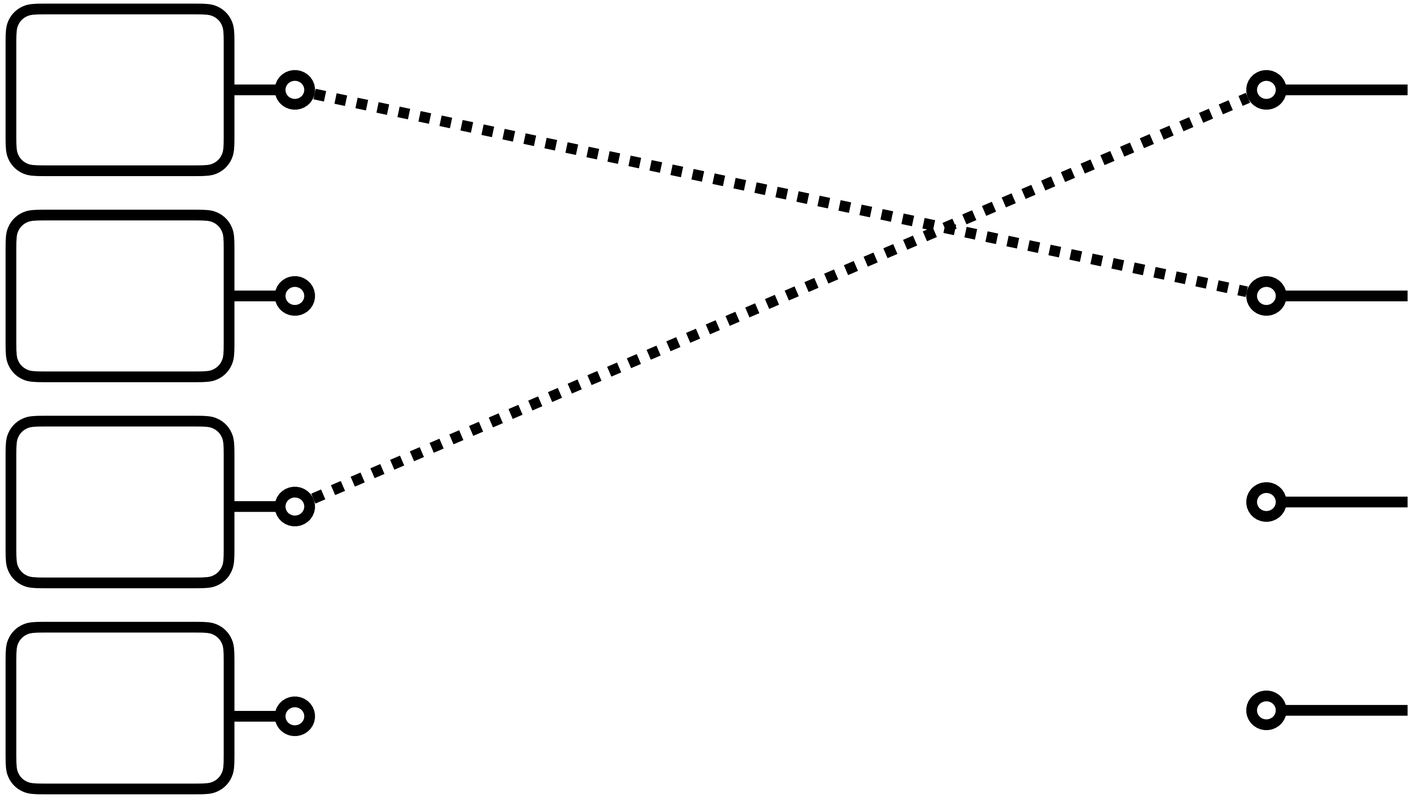
We must ***translate languages***

Lazy Permutation

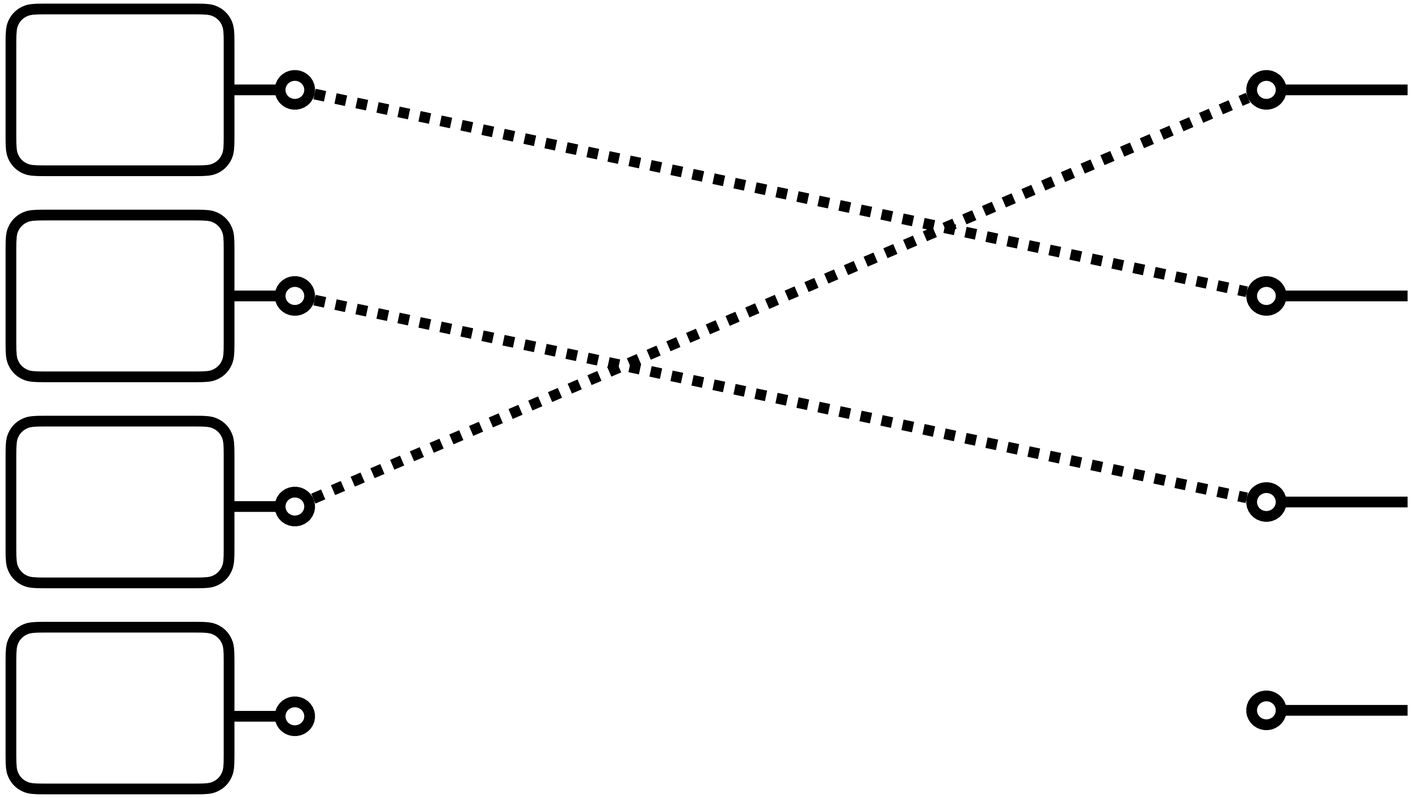
Lazy Permutation



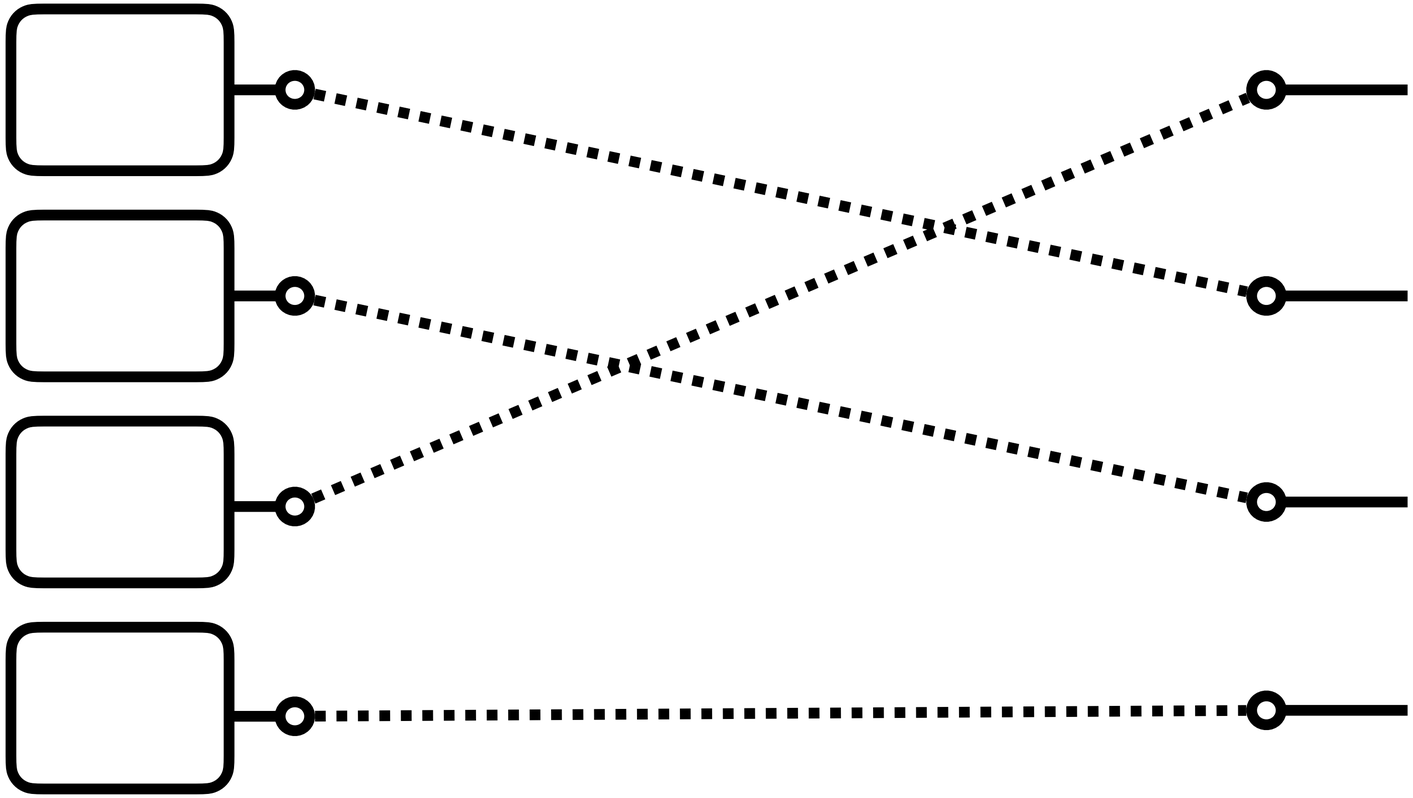
Lazy Permutation



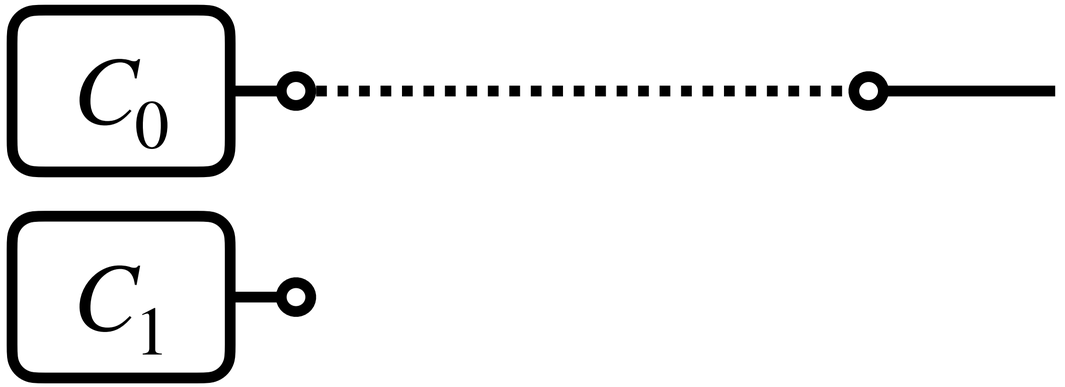
Lazy Permutation



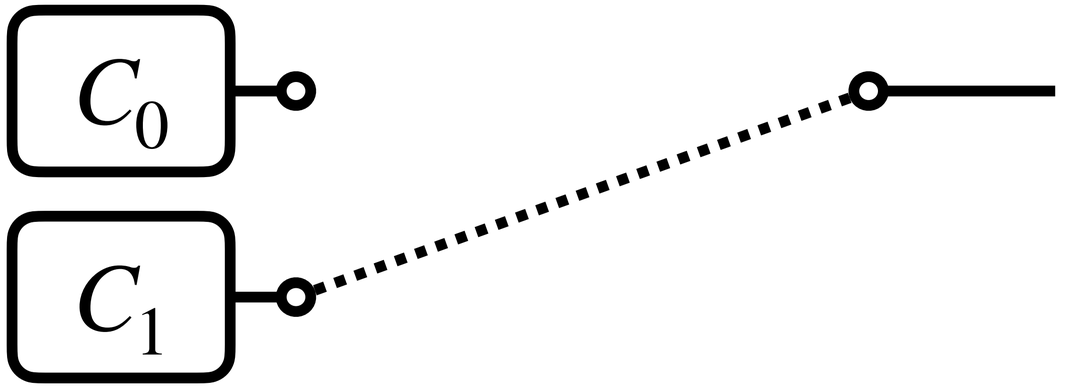
Lazy Permutation



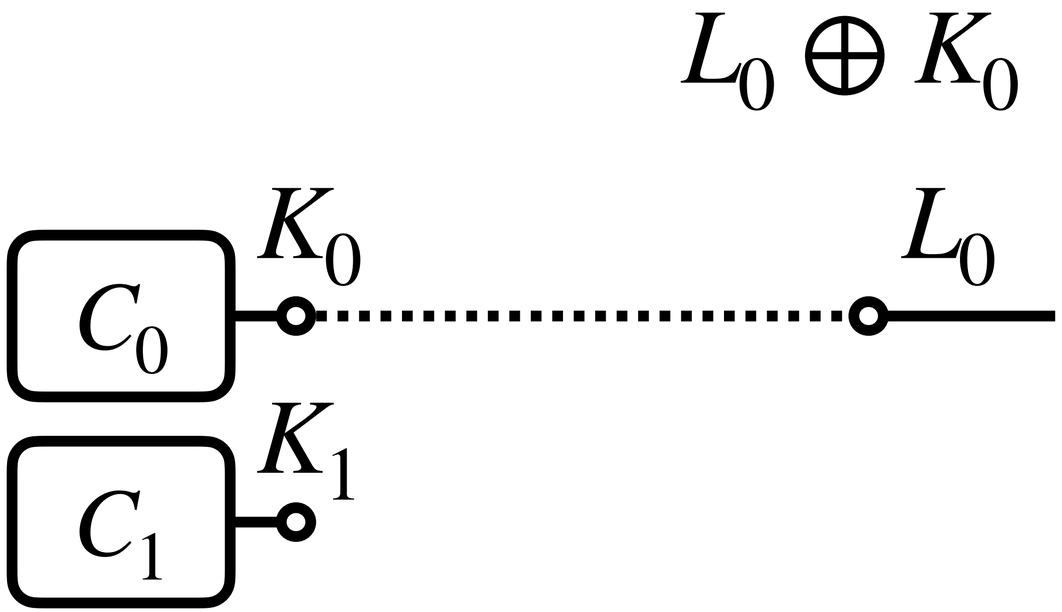
Dynamic Soldering and Evaluation



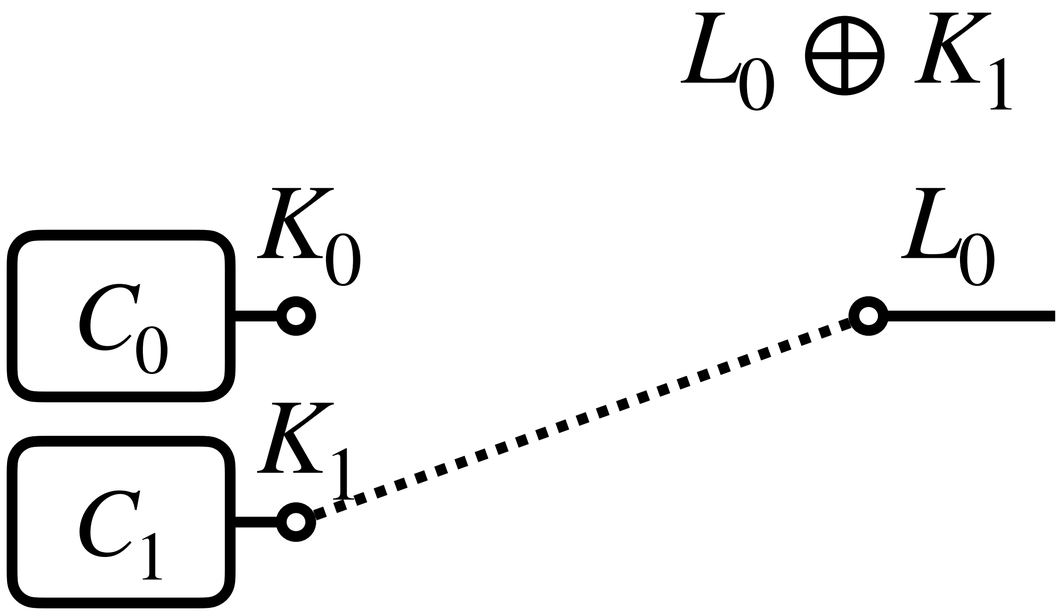
Dynamic Soldering and Evaluation



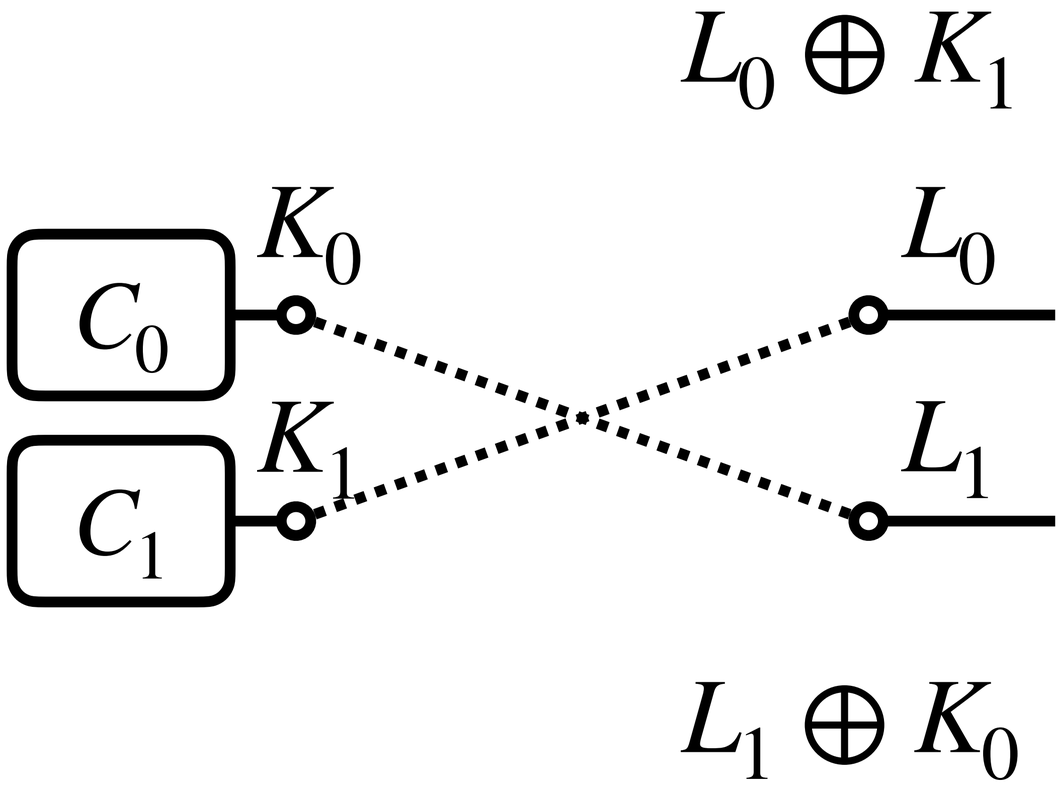
Dynamic Soldering and Evaluation



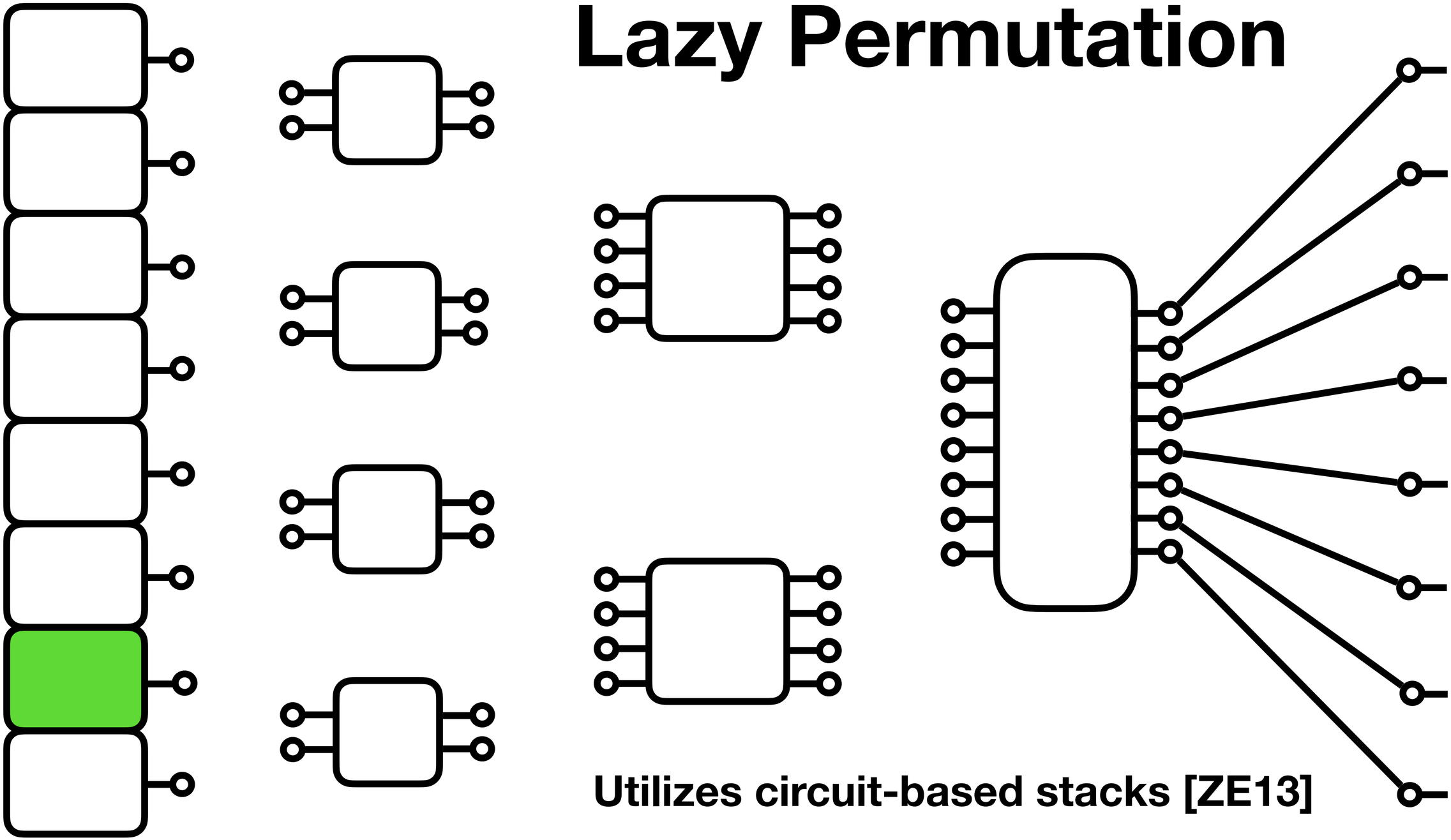
Dynamic Soldering and Evaluation



Dynamic Soldering and Evaluation

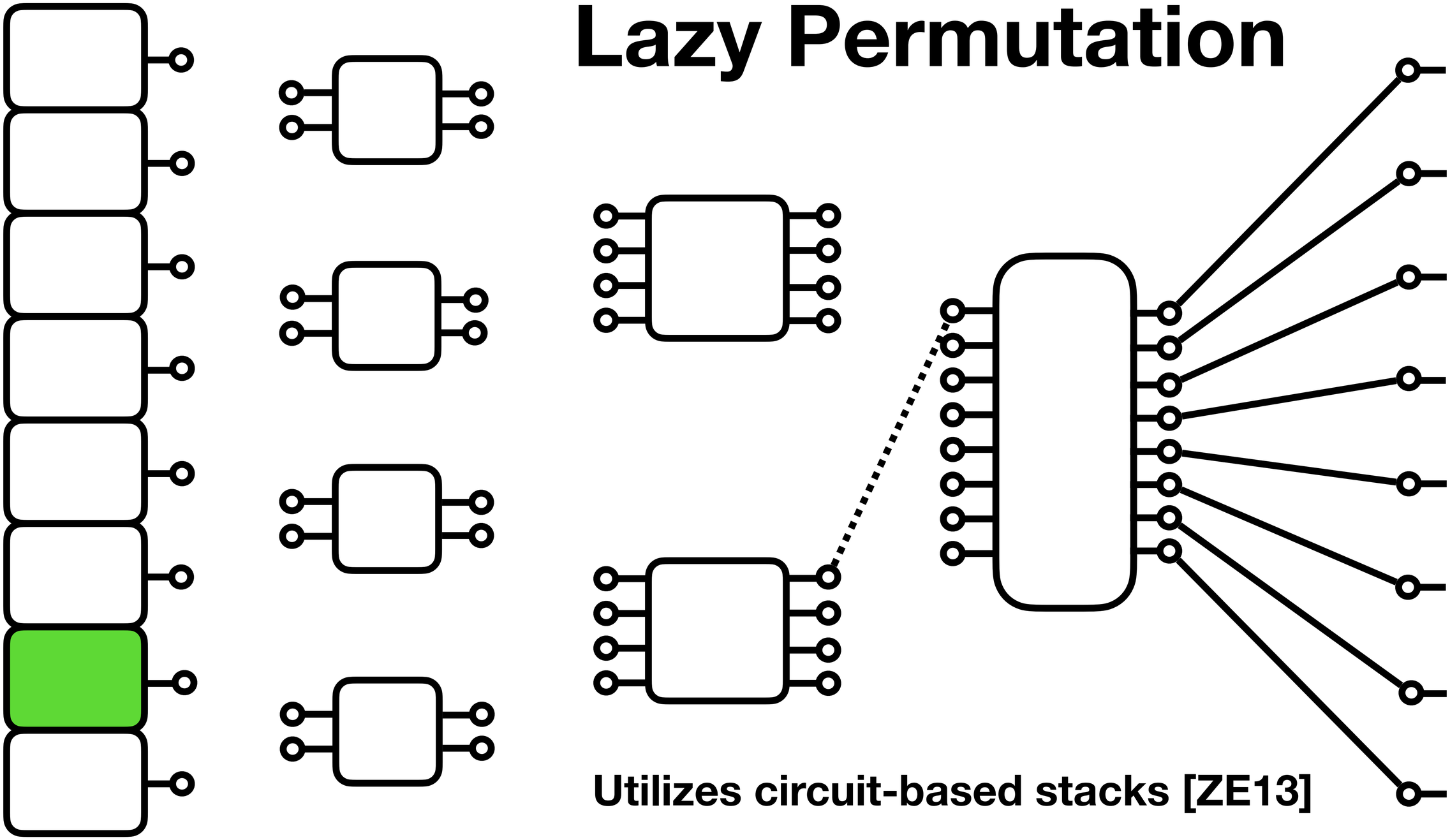


Lazy Permutation



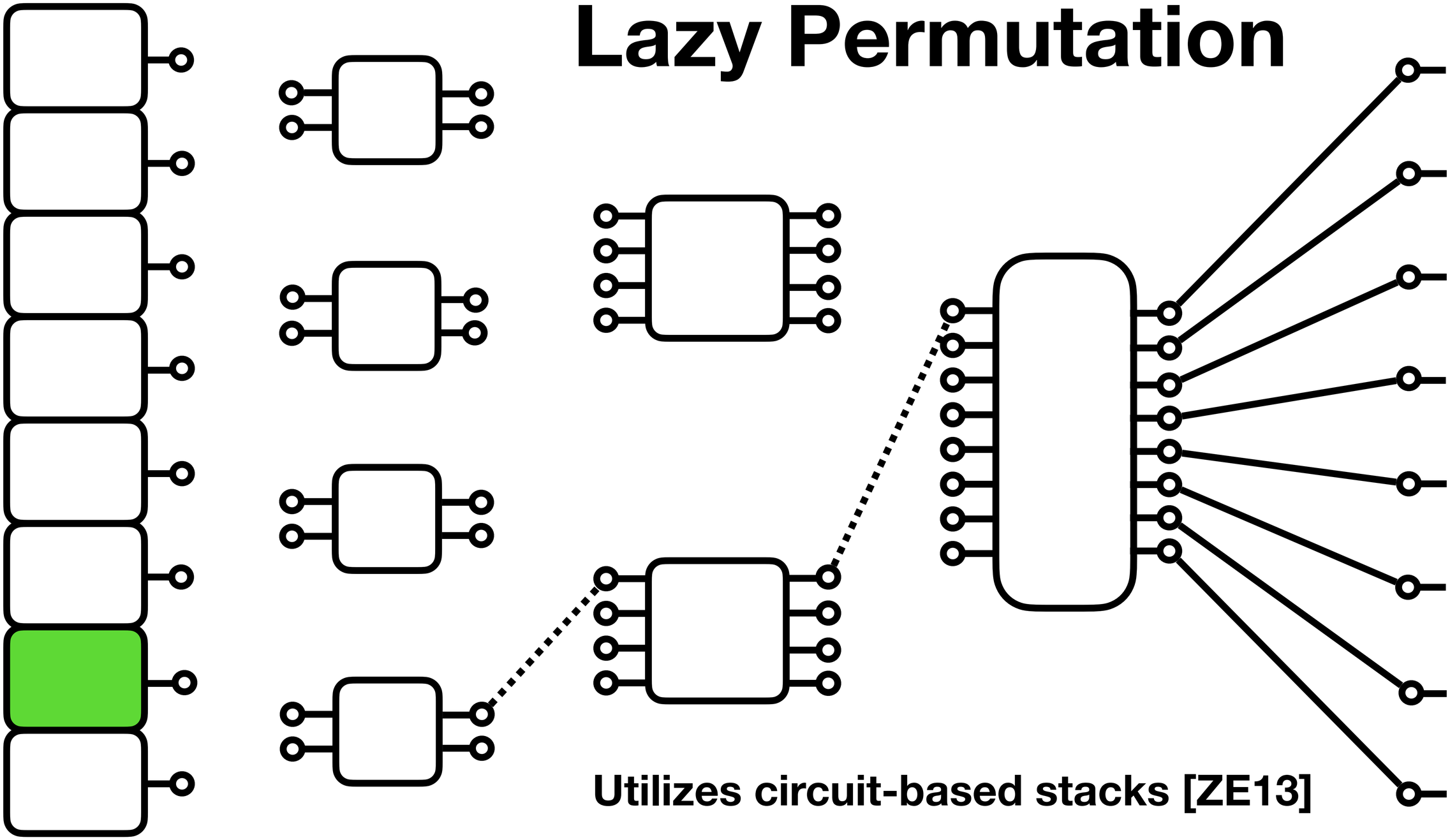
Utilizes circuit-based stacks [ZE13]

Lazy Permutation



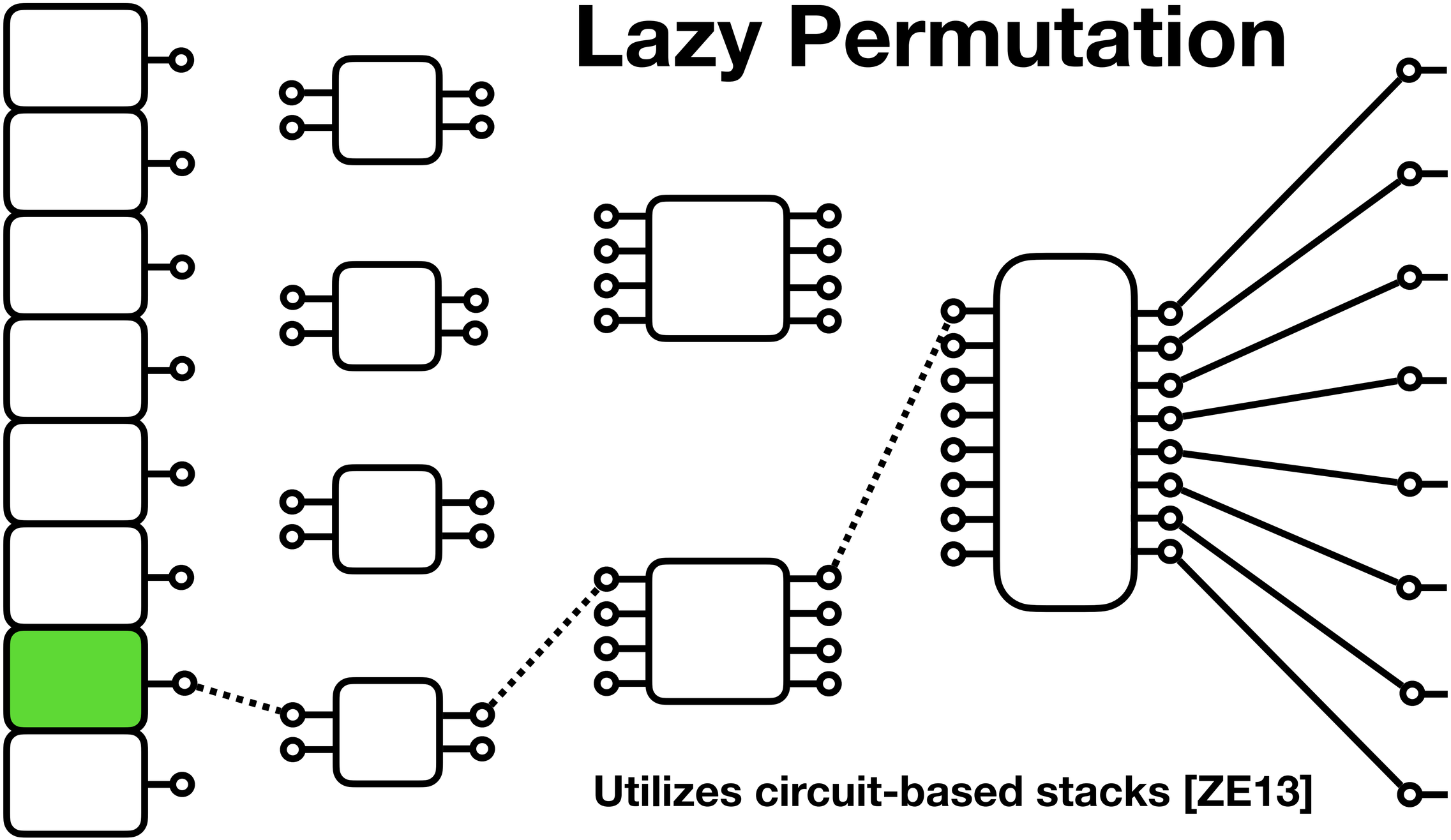
Utilizes circuit-based stacks [ZE13]

Lazy Permutation



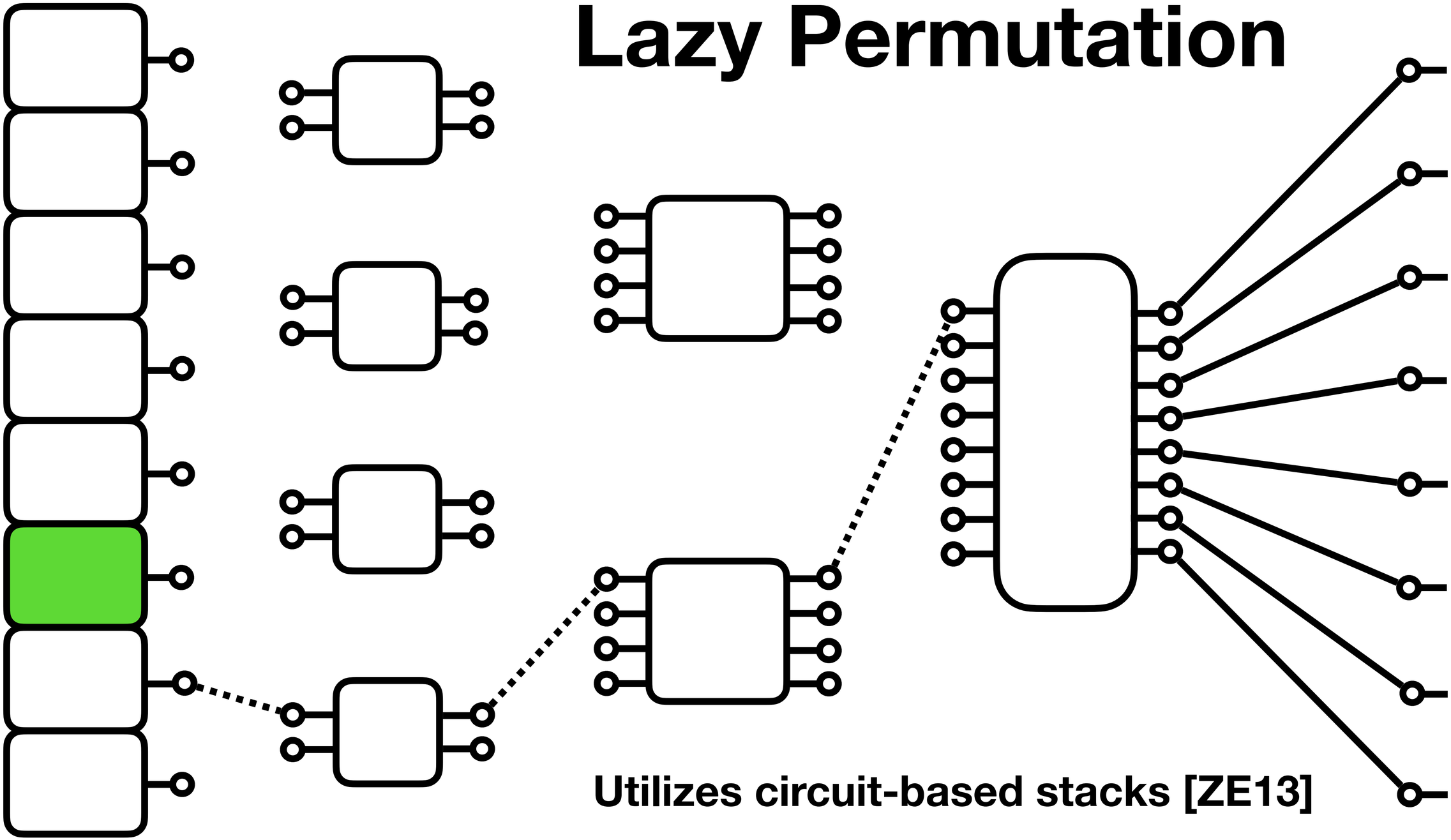
Utilizes circuit-based stacks [ZE13]

Lazy Permutation



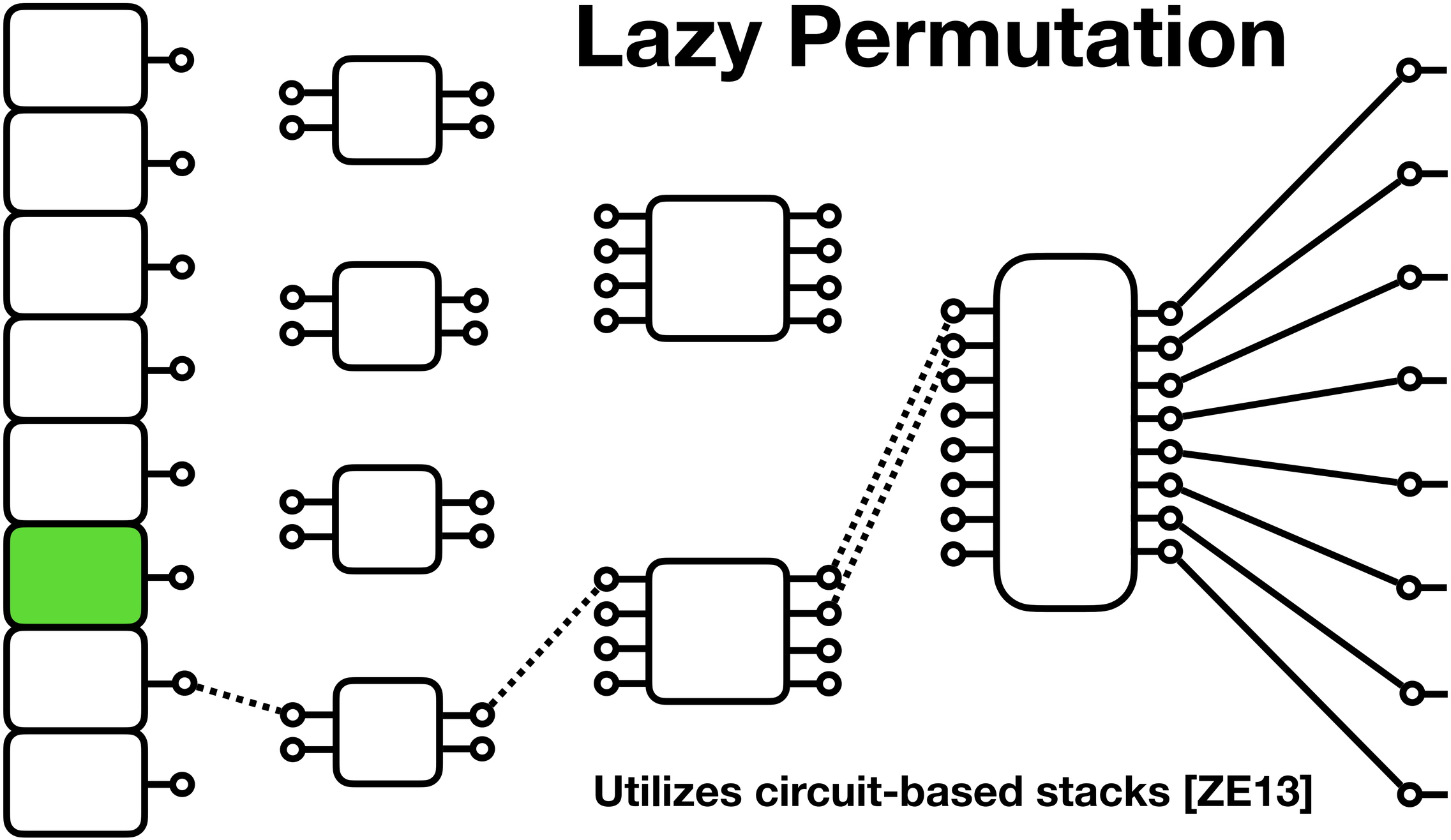
Utilizes circuit-based stacks [ZE13]

Lazy Permutation



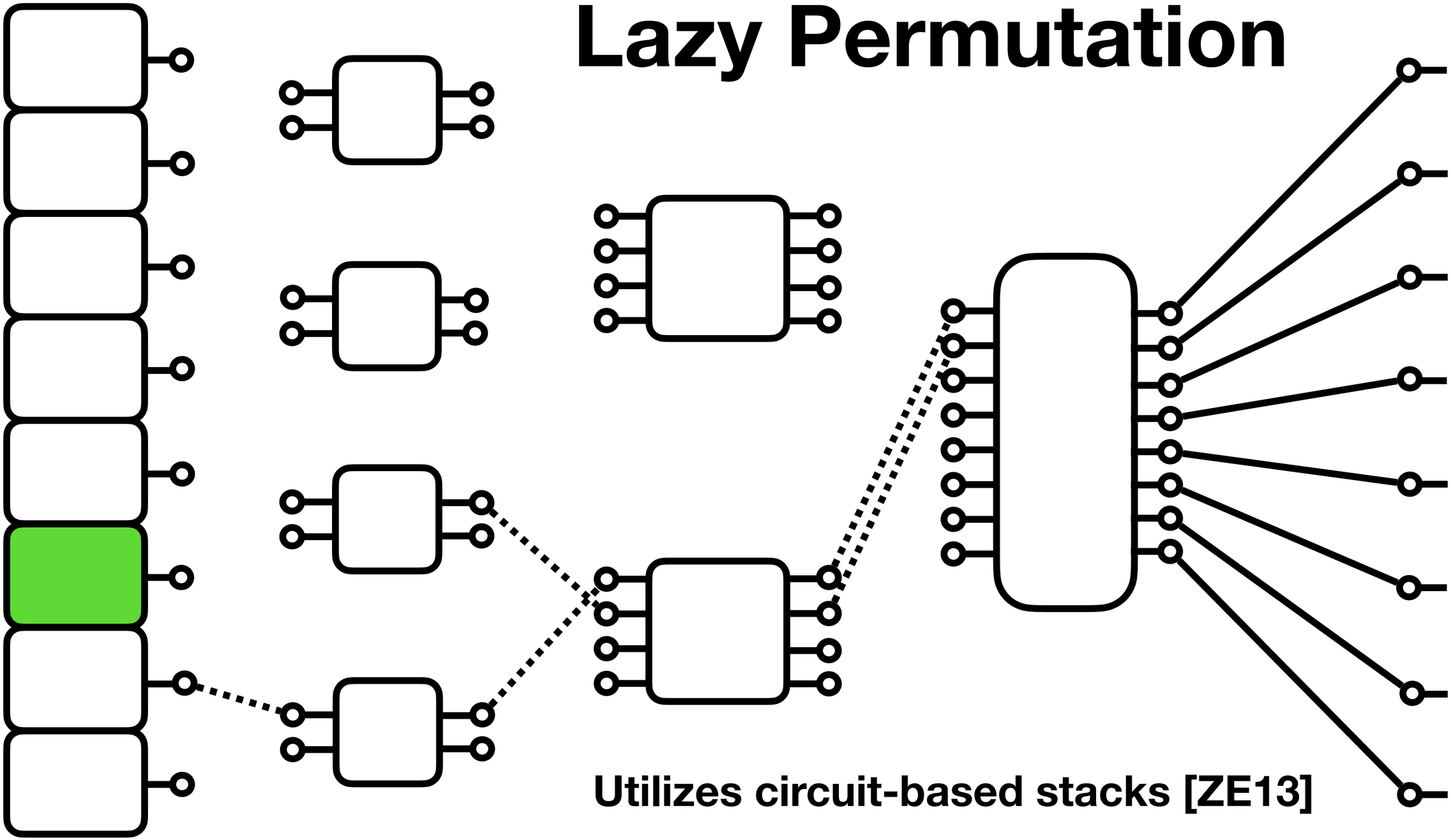
Utilizes circuit-based stacks [ZE13]

Lazy Permutation



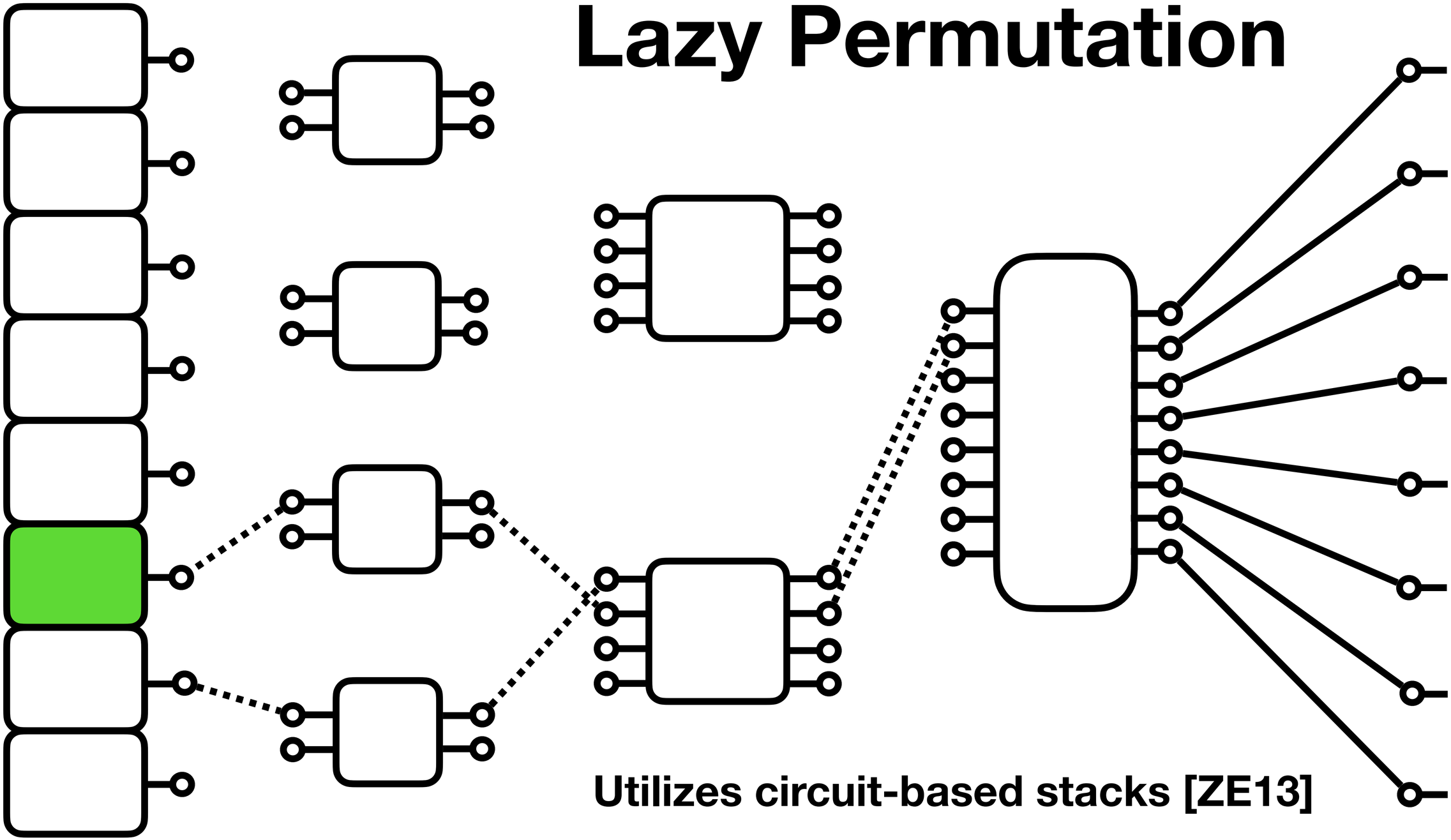
Utilizes circuit-based stacks [ZE13]

Lazy Permutation



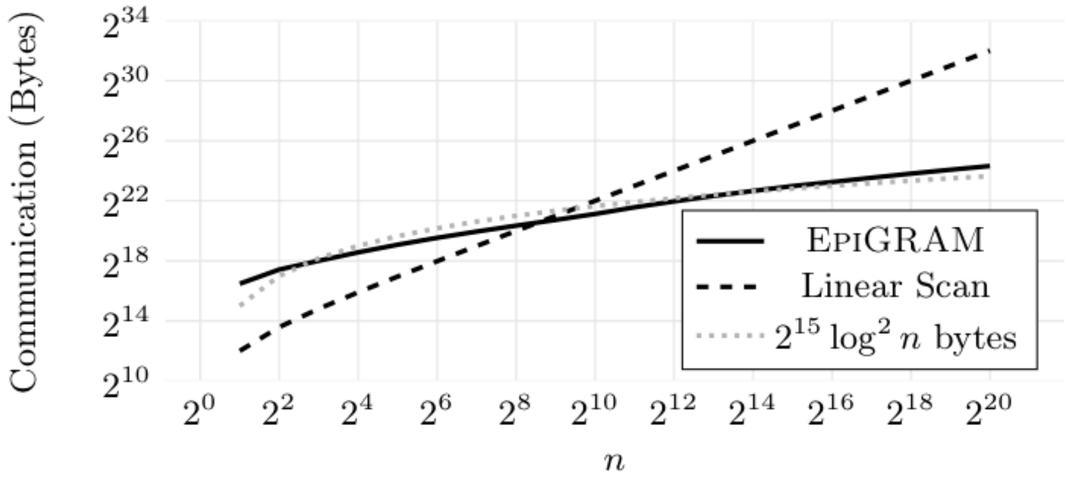
Utilizes circuit-based stacks [ZE13]

Lazy Permutation



Utilizes circuit-based stacks [ZE13]

EpiGRAM



$$O(\kappa \cdot \log^2 n \cdot (w + \log^2 n))$$

Achieves $O(\log^2 n)$ overhead

Opens the door to handling high level programs inside GC