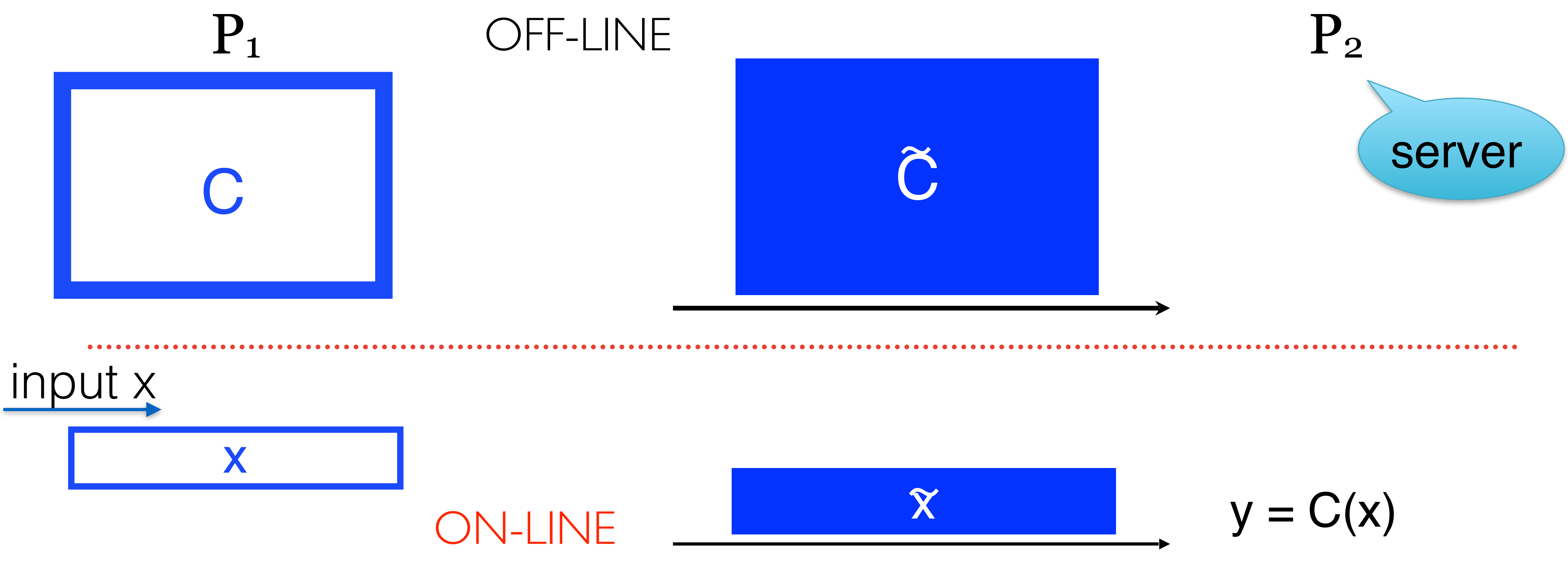
Adaptively Secure Garbled Circuits from one-way functions

Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky,
Alessandra Scafuro, Daniel Wichs

the problem

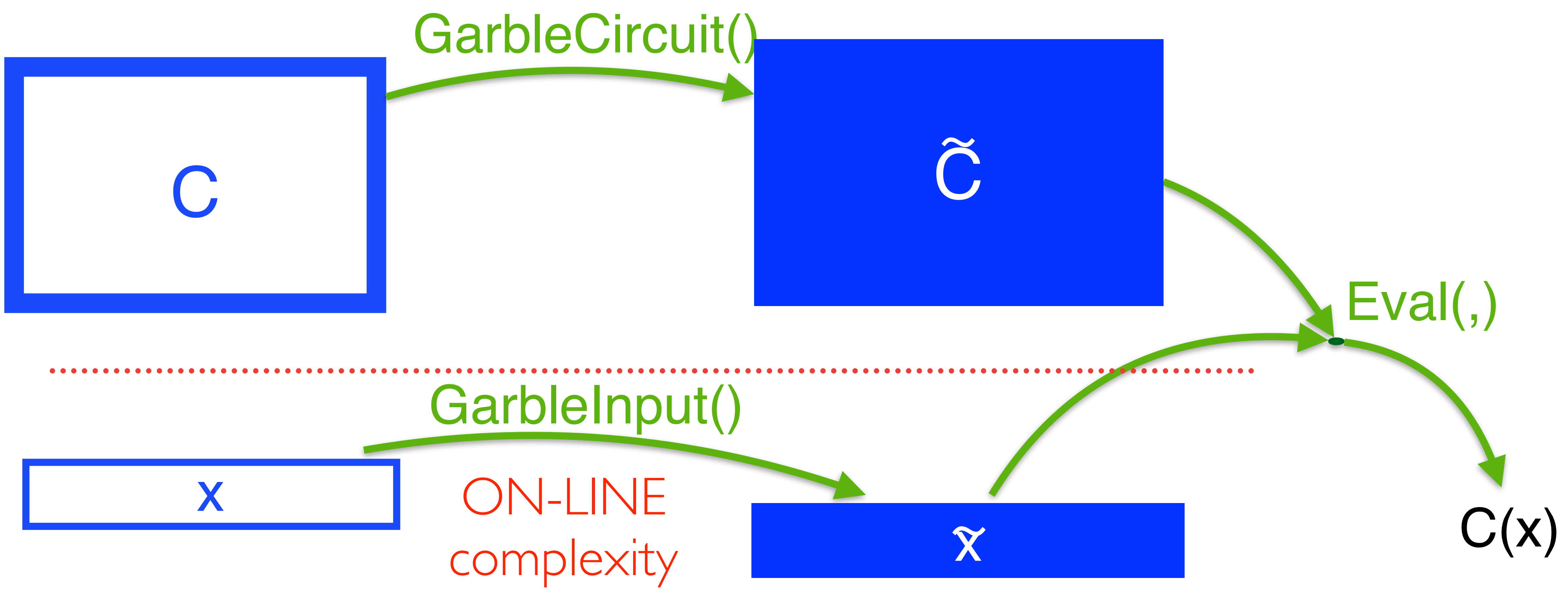
What we want: Server to compute C(x) without learning anything else about C or x.

- correctness
- security
- efficiency

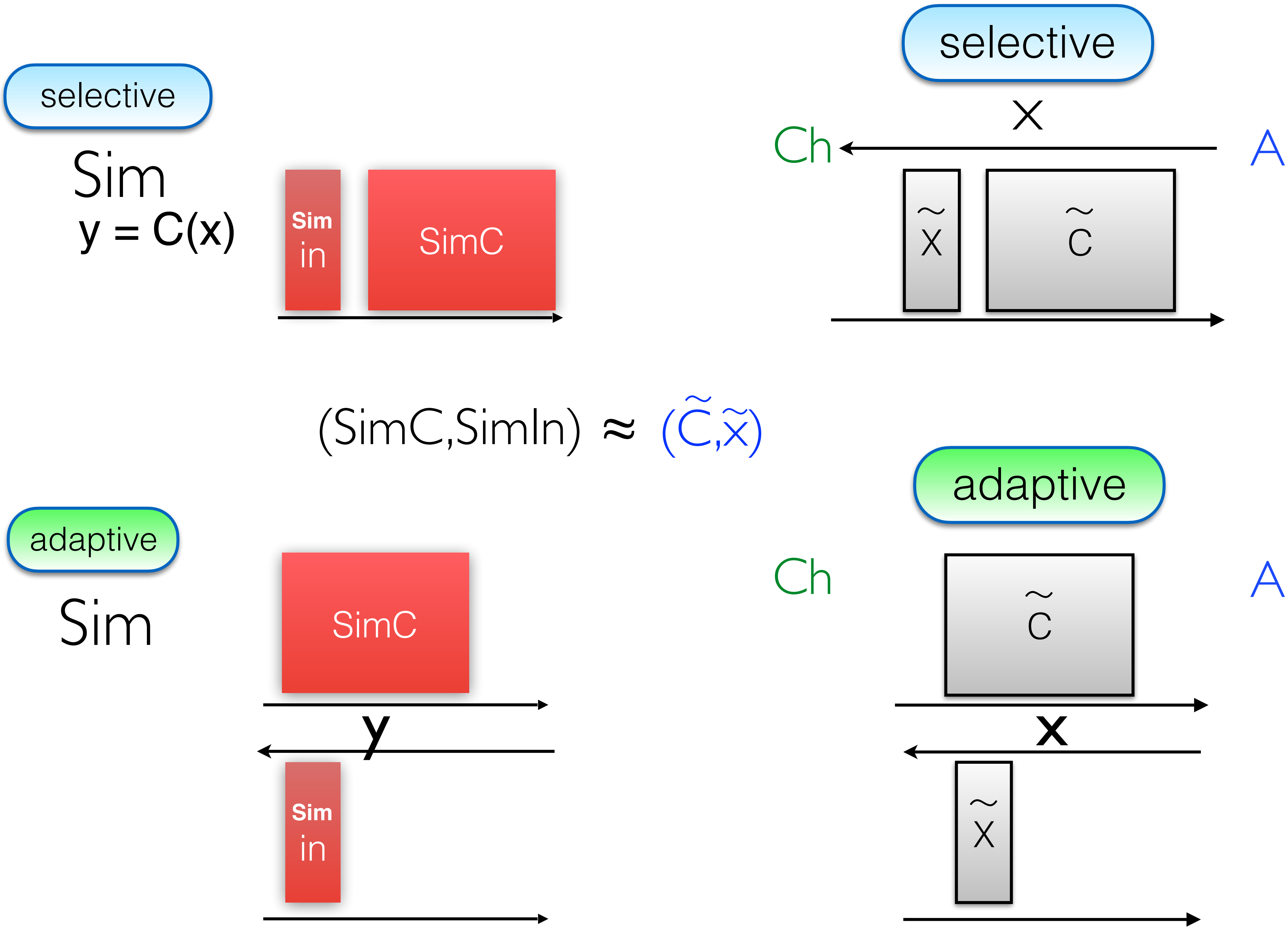


Efficiency ON-LINE complexity smaller than circuit size

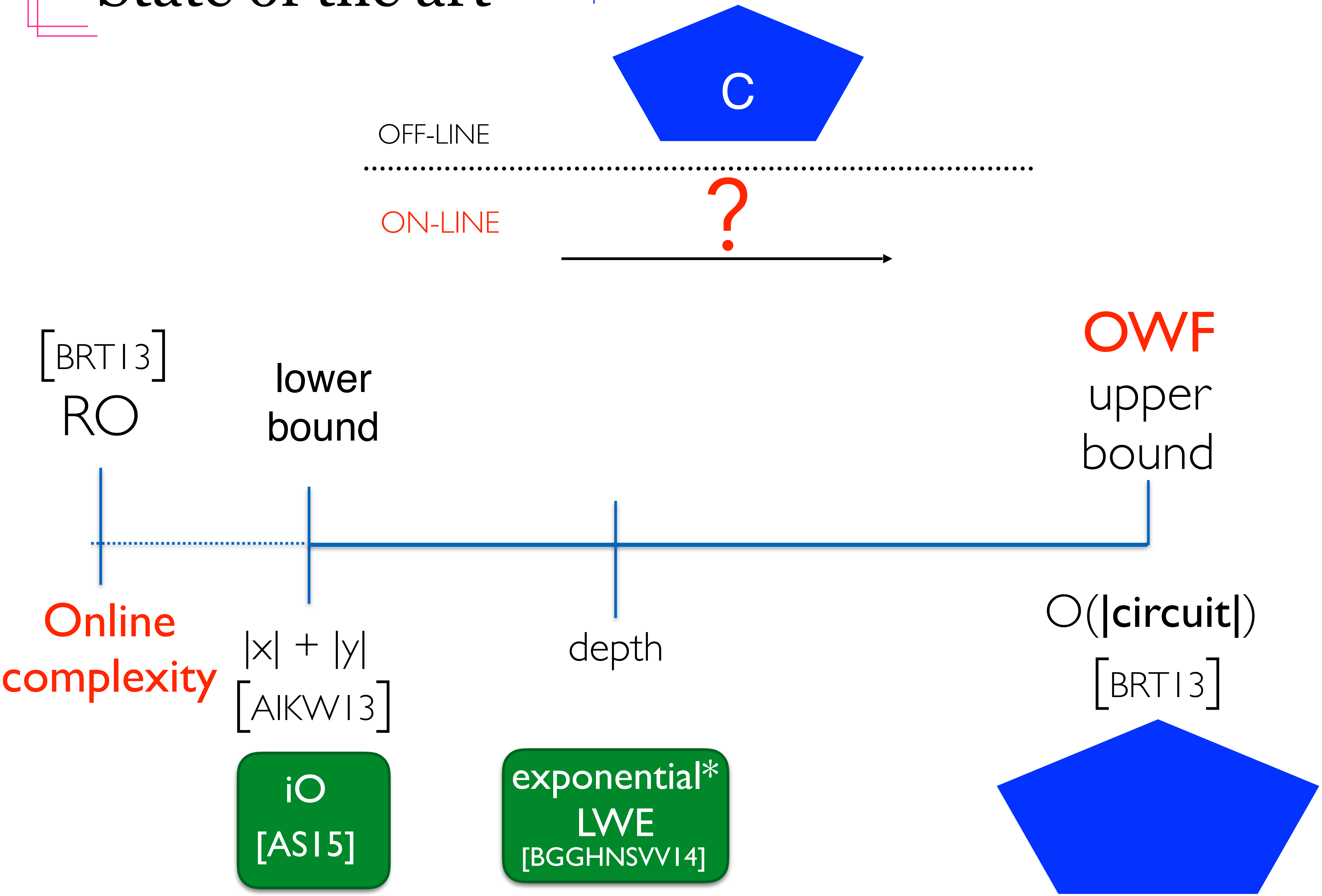
Garbling Scheme :



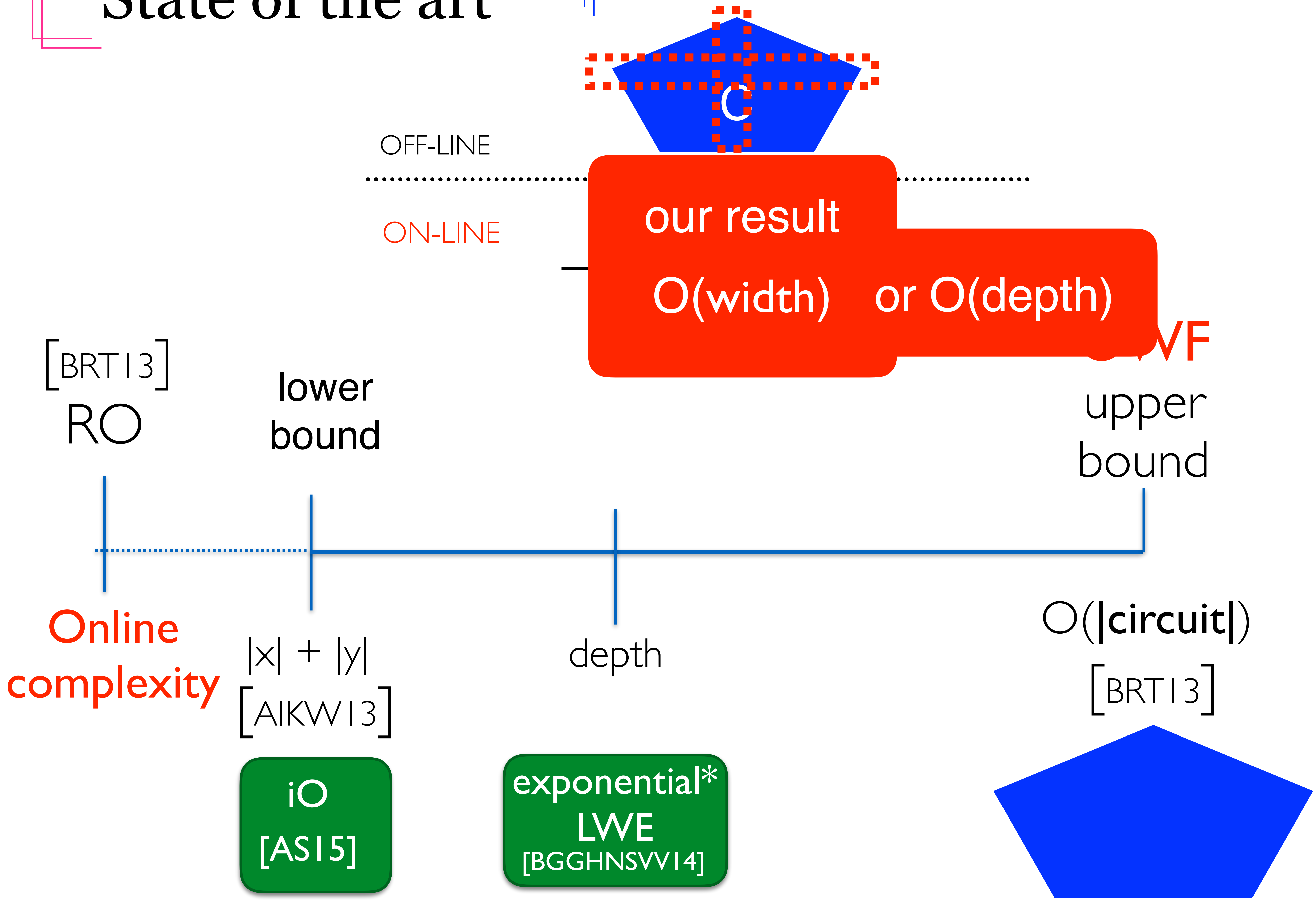
Security


selective
Sim
y = C(x)
Sim in
SimC
selective
x
Ch
A
x̃
C̃
(SimC,SimIn) ≈ (C̃,x̃)
adaptive
Sim
SimC
y
Sim in
adaptive
Ch
A
C̃
x
x̃

State of the art



State of the art



Outline

- Yao's garbling scheme
- Selective → Adaptive Yao: Difficulties
- Our approach

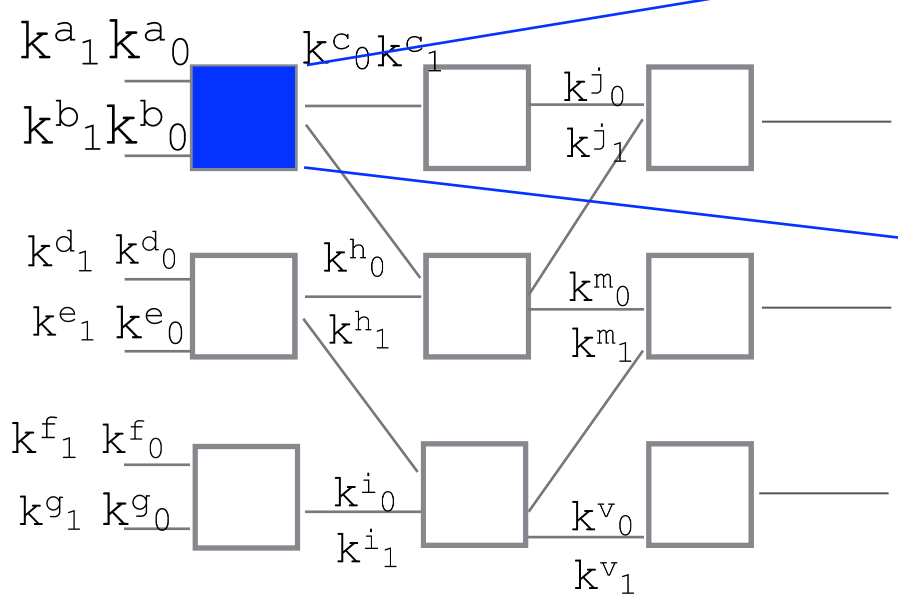
Garbling scheme

Real Garbling

Simulation

Indistinguishability proof

Yao's garbling scheme



Garbled gate

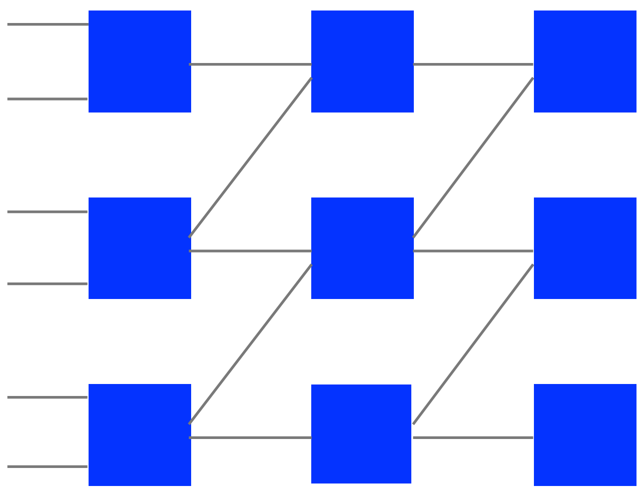
$\mathbf{Enc}_{(k^a_0,k^b_0)}(k^c_0)$
$\mathbf{Enc}_{(k^a_0,k^b_1)}(k^c_0)$
$\mathbf{Enc}_{(k^a_1,k^b_0)}(k^c_0)$
$\mathbf{Enc}_{(k^a_1,k^b_1)}(k^c_1)$

Notation:

$$\mathbf{Enc}_{(k^a_1,k^b_0)}(k^c_1) := \mathbf{Enc}_{k^a_1}(\mathbf{Enc}_{k^b_0}(k^c_1))$$

Yao's garbling scheme

GarbleCircuit(C)



output table

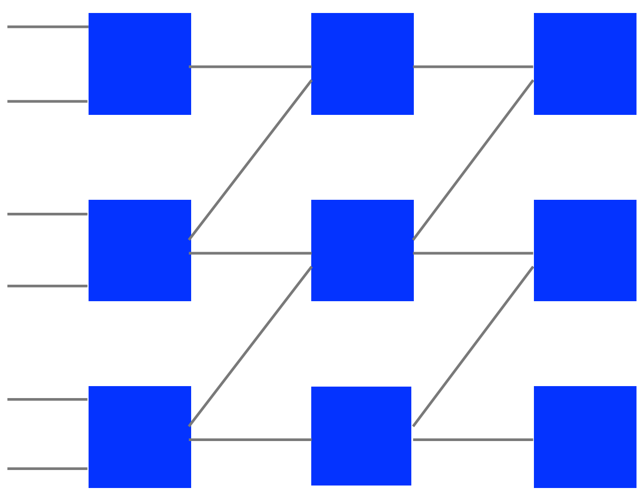
k^g_0	⇢ 0
k^g_1	⇢ 1
k^f_0	⇢ 0
k^f_1	⇢ 1
k^h_0	⇢ 0
k^h_1	⇢ 1

GarbleInput(x)

k^a_1	$\mathbf{k^a_0}$
$\mathbf{k^b_1}$	k^b_0
k^d_1	$\mathbf{k^d_0}$
$\mathbf{k^e_1}$	k^e_0
k^d_1	$\mathbf{k^d_0}$
k^e_1	$\mathbf{k^e_0}$

Yao's garbling scheme

GarbleCircuit(C)



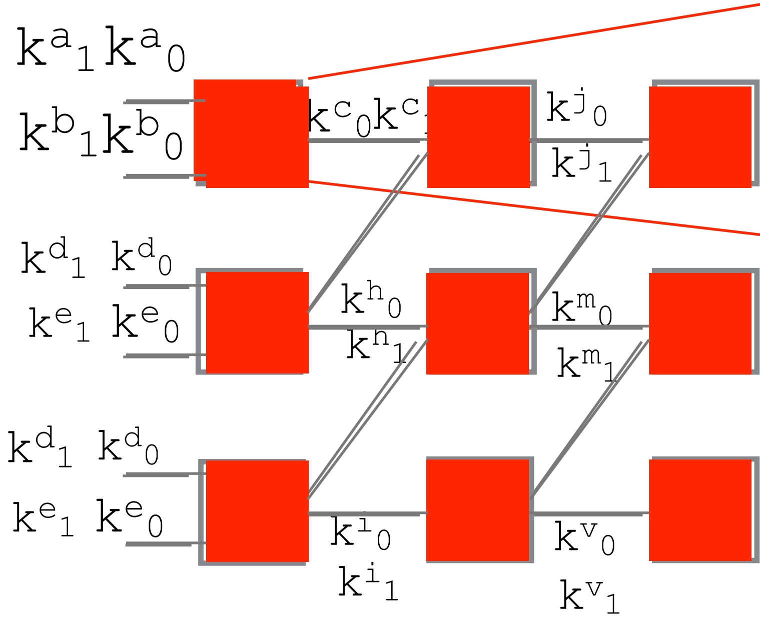
output table

k^g_0 ⇢	0
k^g_1 ⇢	1
k^f_0 ⇢	0
k^f_1 ⇢	1
k^h_0 ⇢	0
k^h_1 ⇢	1

GarbleInput(x)

k^a_1 $\mathbf{k^a_0}$
$\mathbf{k^b_1}$ k^b_0
k^d_1 $\mathbf{k^d_0}$
$\mathbf{k^e_1}$ k^e_0
k^d_1 $\mathbf{k^d_0}$
k^e_1 $\mathbf{k^e_0}$

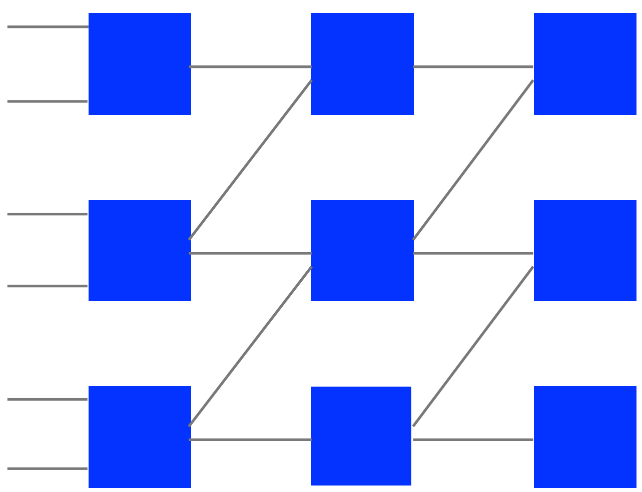
Sim(y)



Enc (ka_0, kb_0) (k^c_0)
Enc (ka_0, kb_1) (k^c_0)
Enc (ka_1, kb_0) (k^c_0)
Enc (ka_1, kb_1) (k^c_0)

Yao's garbling scheme

GarbleCircuit(C)



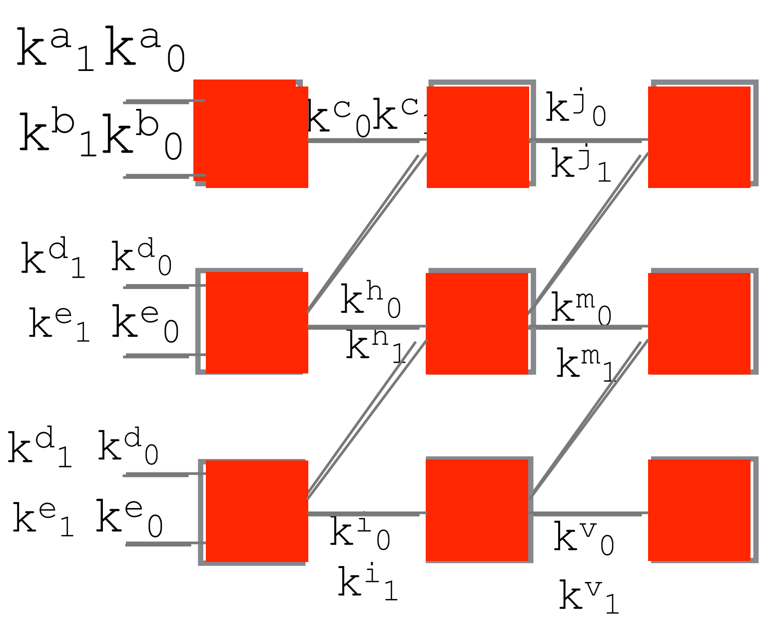
output table

$k^g_0 \dashrightarrow 0$
$k^g_1 \dashrightarrow 1$
$k^f_0 \dashrightarrow 0$
$k^f_1 \dashrightarrow 1$
$k^h_0 \dashrightarrow 0$
$k^h_1 \dashrightarrow 1$

GarbleInput(x)

k^a_1 $\mathbf{k^a_0}$
$\mathbf{k^b_1}$ k^b_0
k^d_1 $\mathbf{k^d_0}$
$\mathbf{k^e_1}$ k^e_0
k^d_1 $\mathbf{k^d_0}$
k^e_1 $\mathbf{k^e_0}$

Sim(y)



output table

$k^g_0 \dashrightarrow \mathbf{y_1}$
$k^g_1 \dashrightarrow 1-y_1$
$k^f_0 \dashrightarrow \mathbf{y_2}$
$k^f_1 \dashrightarrow 1-y_2$
$k^h_0 \dashrightarrow \mathbf{y_3}$
$k^h_1 \dashrightarrow 1-y_3$

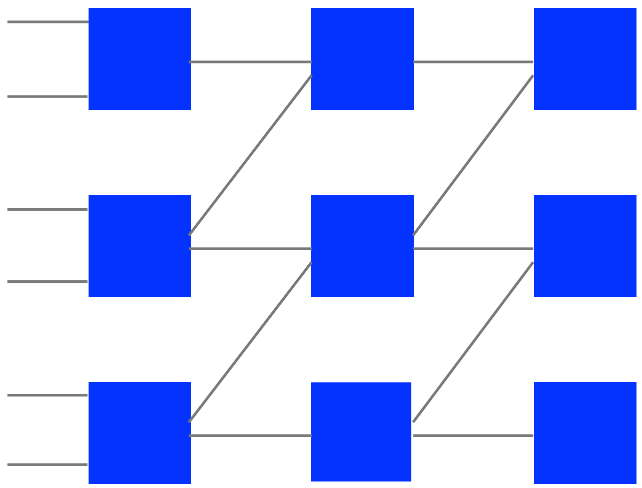
GarbleInput(x)

$\mathbf{k^a_0}$
$\mathbf{k^b_0}$
$\mathbf{k^d_0}$
$\mathbf{k^e_0}$
$\mathbf{k^d_0}$
$\mathbf{k^e_0}$

Yao's garbling scheme

real

GarbleCircuit(C)



output table

$k^g_0 \to 0$

$k^g_1 \to 1$

$k^f_0 \to 0$

$k^f_1 \to 1$

$k^h_0 \to 0$

$k^h_1 \to 1$

GarbleInput(x)

k^a_0

k^b_1

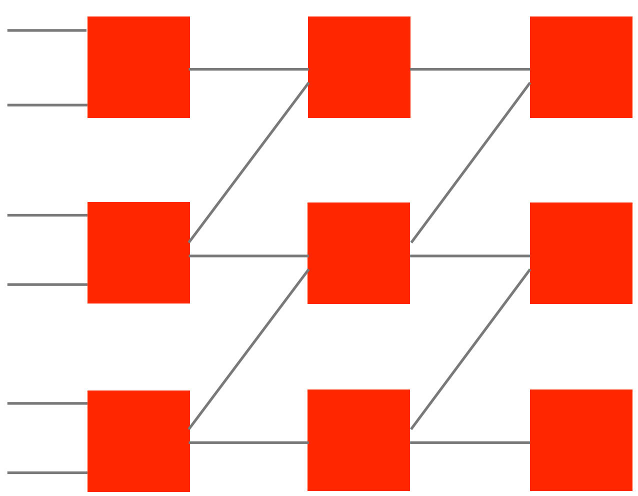
k^d_0

k^e_1

k^d_0

k^e_0

simulated



$k^g_0 \to y_1$

$k^g_1 \to 1-y_1$

$k^f_0 \to y_2$

$k^f_1 \to 1-y_2$

$k^h_0 \to y_3$

$k^h_1 \to 1-y_3$

k^a_0

k^b_0

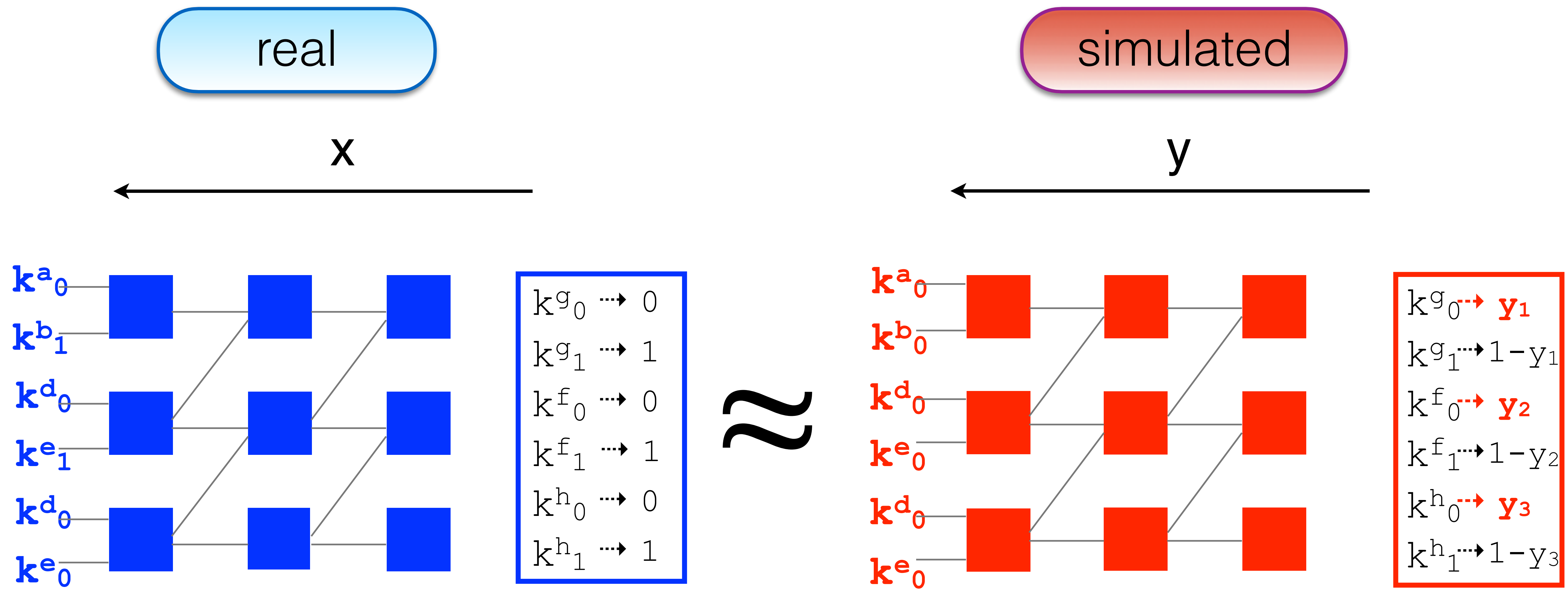
k^d_0

k^e_0

k^d_0

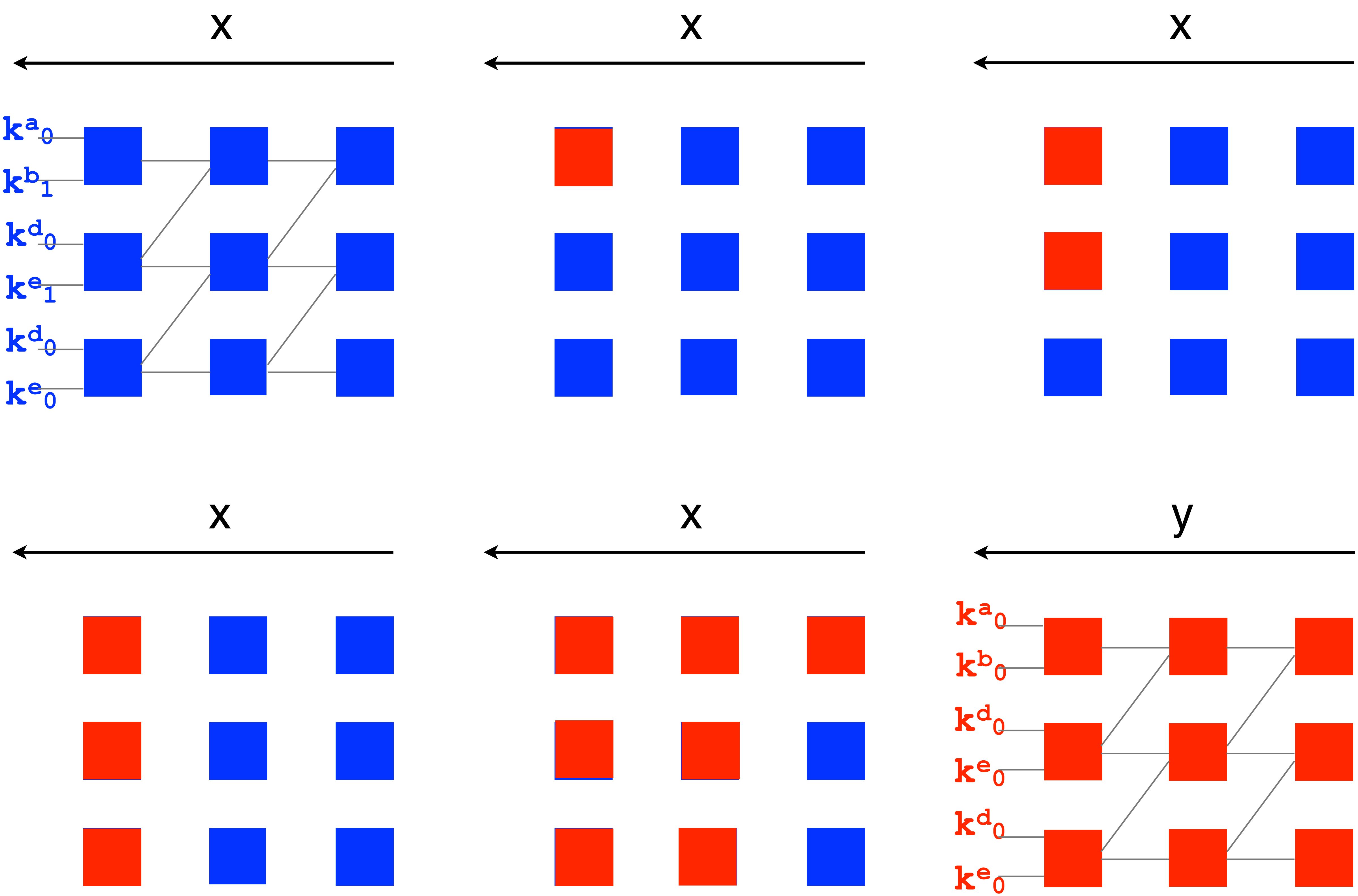
k^e_0

Indistinguishability Proof [Lindell-Pinkas 04]



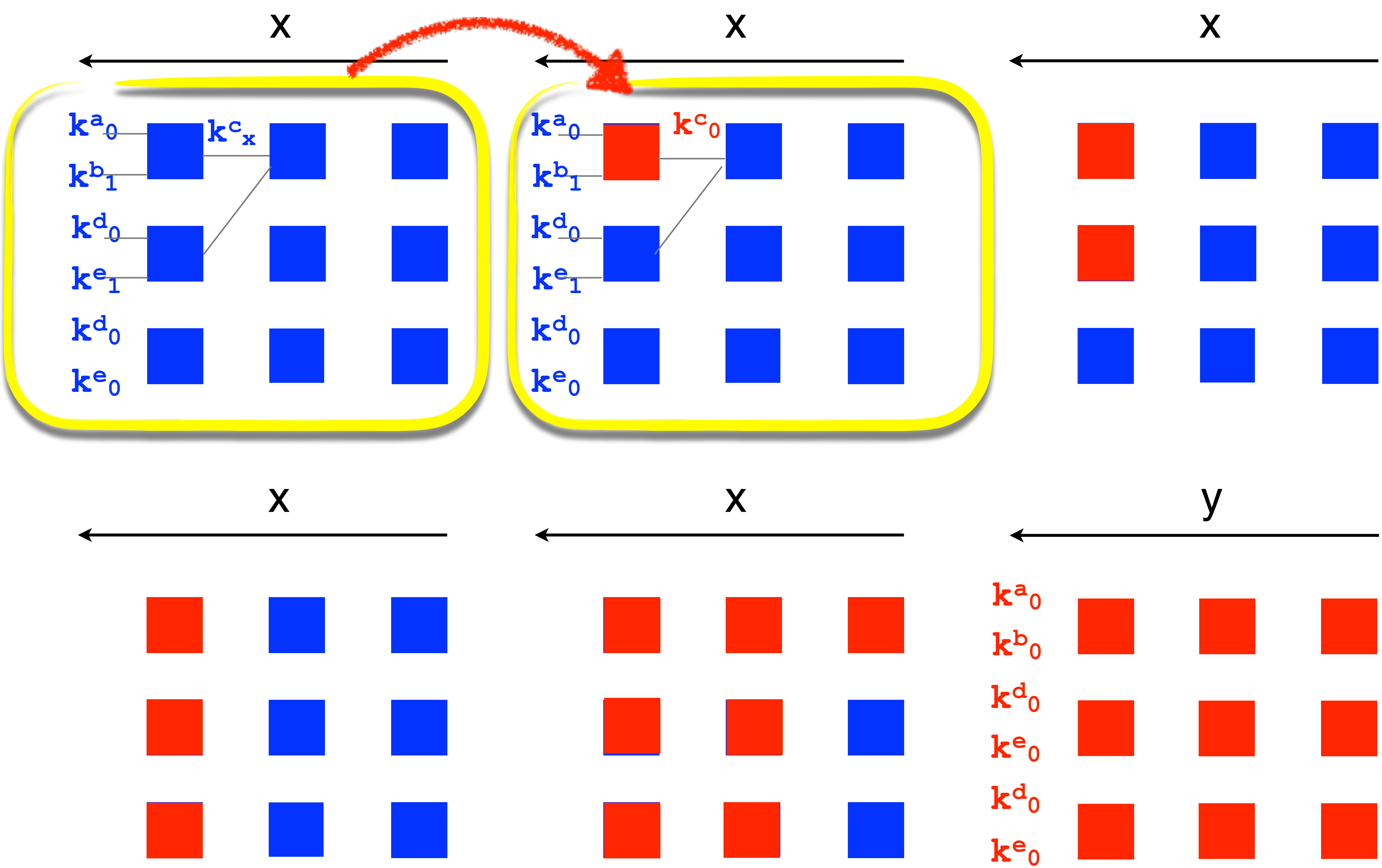
A distinguishes the two ⇒ A' breaks CPA security

Hybrid distributions

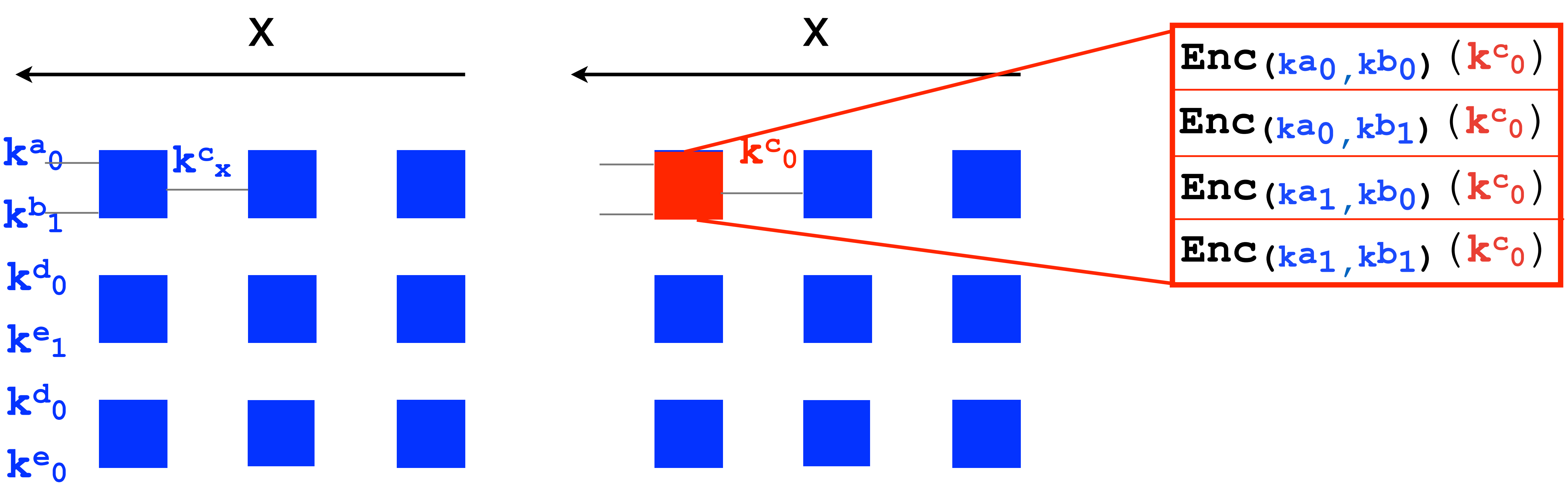


Hybrid distributions

computationally indistinguishable?



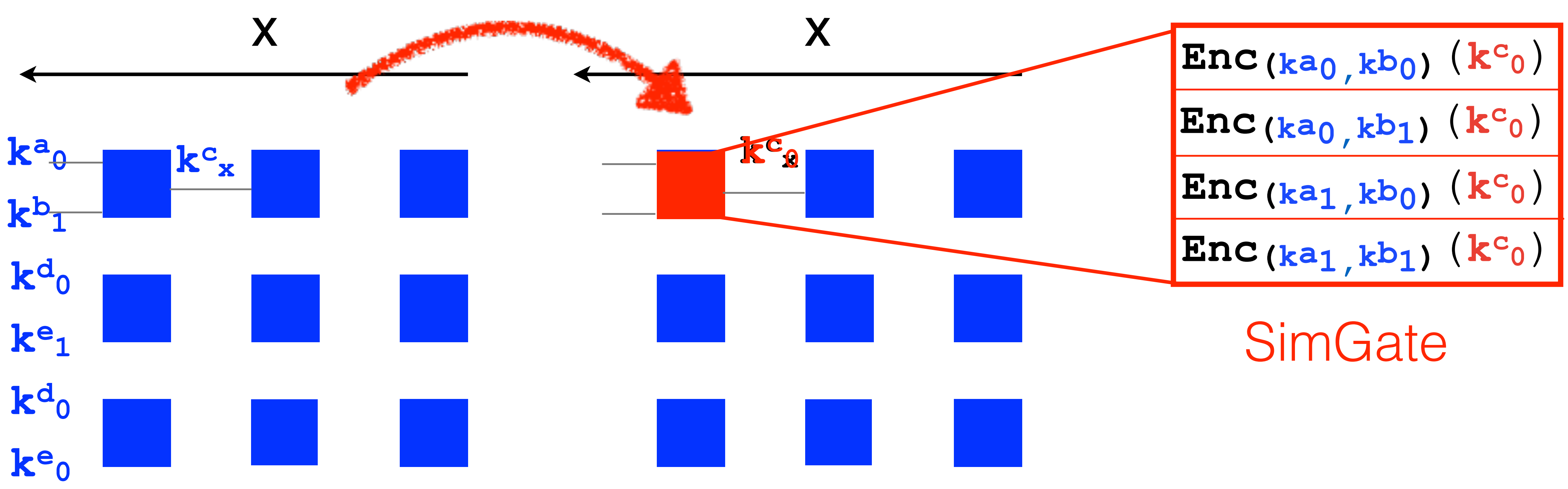
Hybrid distributions



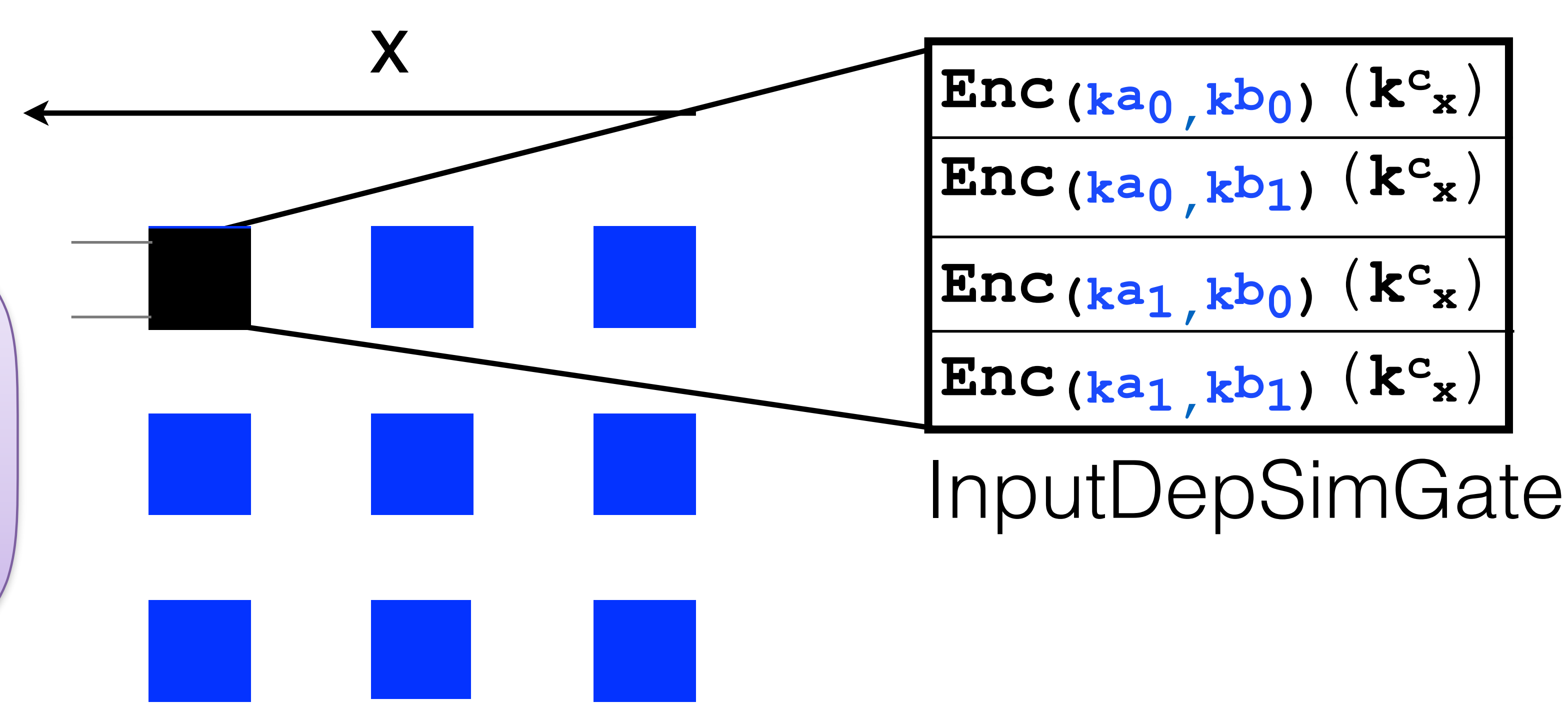
We know the input x, before creating the GarbledCircuit

Hybrid distributions

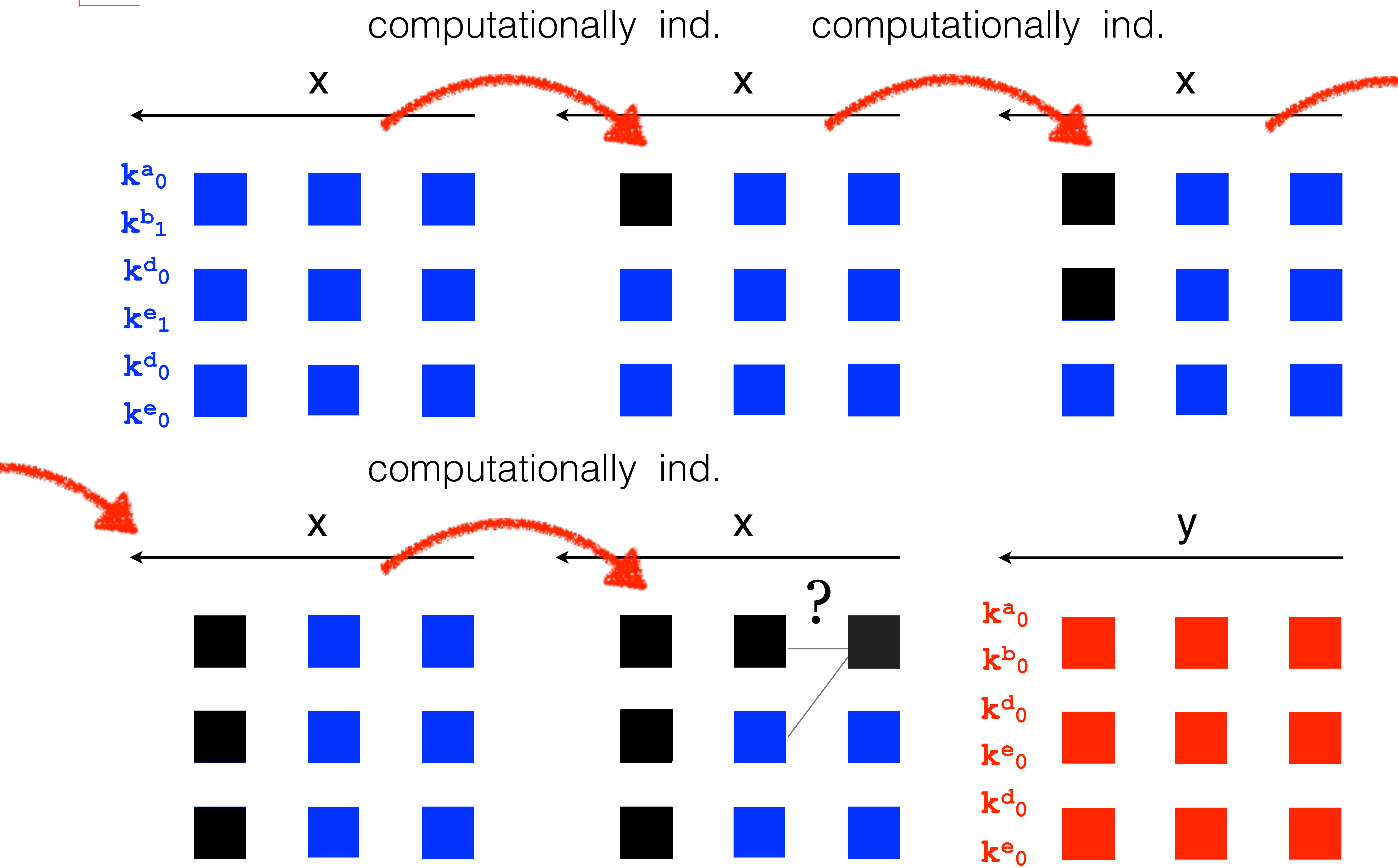
computationally indistinguishable!



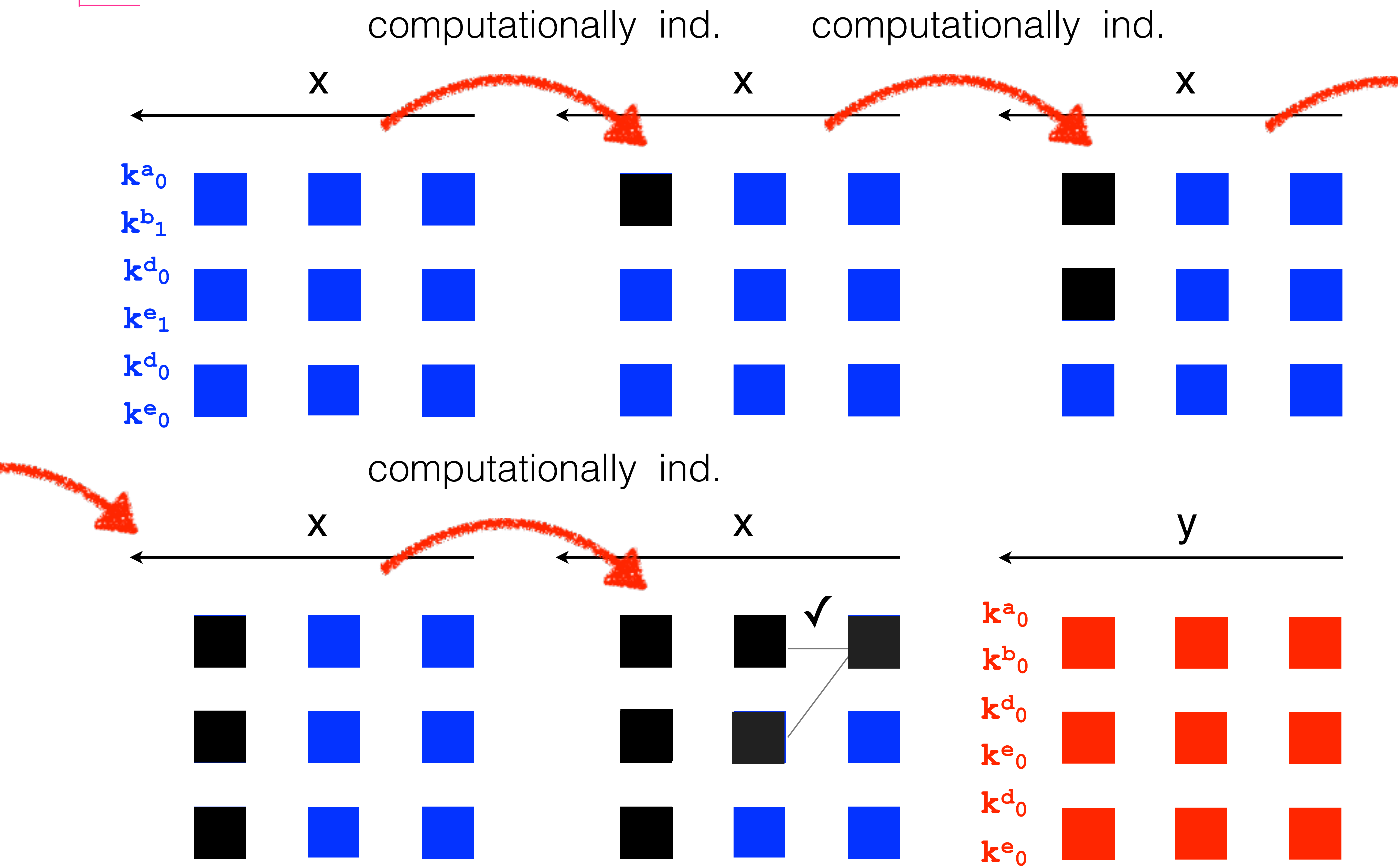
We know the input x, before creating the GarbledCircuit



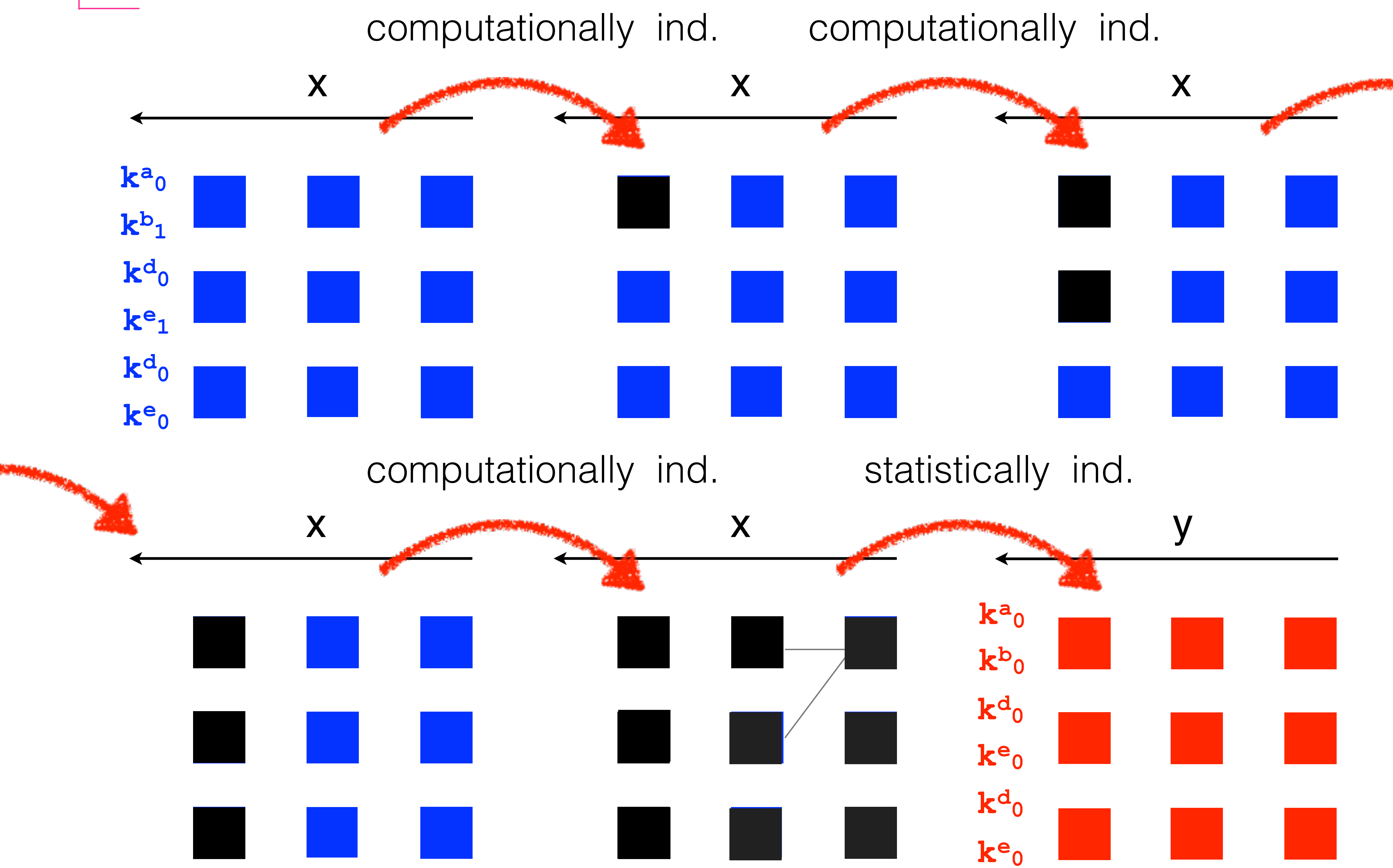
Hybrid distributions



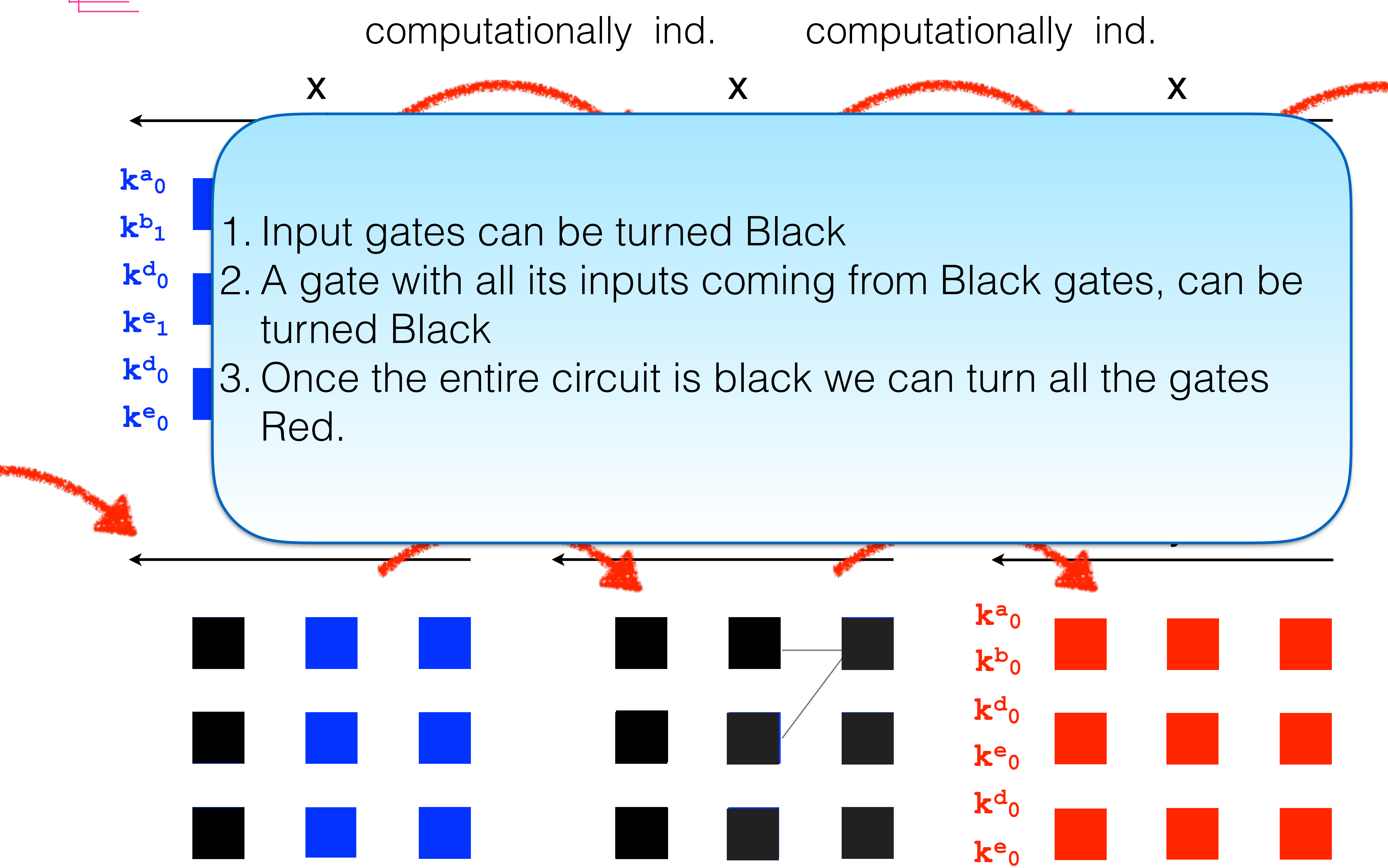
Hybrid distributions



Hybrid distributions



Hybrid distributions



1. Input gates can be turned Black
2. A gate with all its inputs coming from Black gates, can be turned Black
3. Once the entire circuit is black we can turn all the gates Red.

Outline

- Yao's garbling scheme
- Selective → Adaptive Yao: Difficulties
- Our approach

Selective to Adaptive

Real Garbling

Simulation

Indistinguishability proof

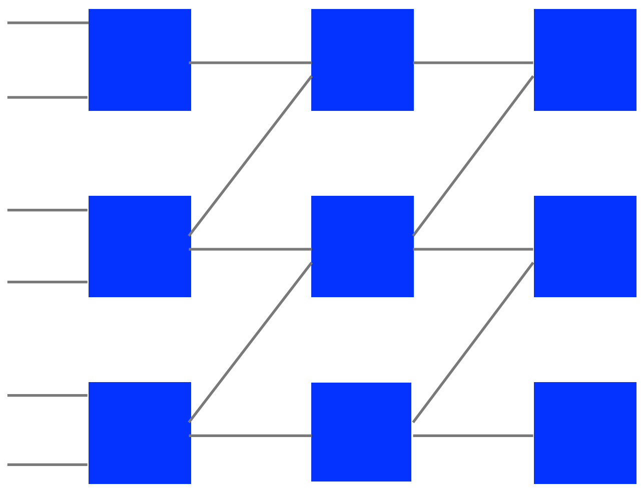
on-line complexity is at least
input size+output size

Modified Yao's garbling scheme

OFF-LINE | ON-LINE

real

GarbleCircuit(C)



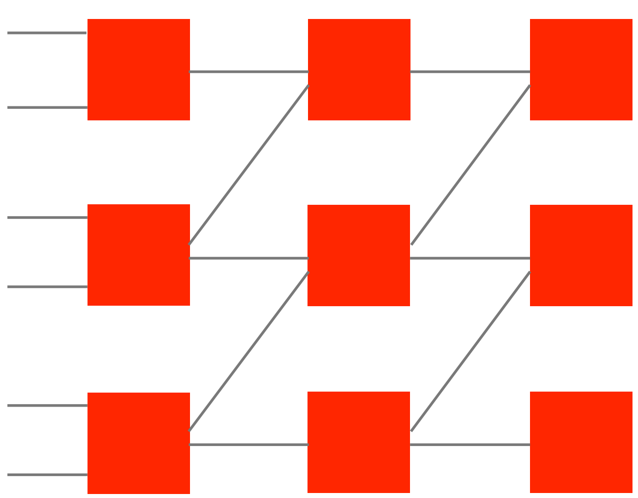
GarbleInput(x)

output table

$k^g_0 \dashrightarrow 0$
$k^g_1 \dashrightarrow 1$
$k^f_0 \dashrightarrow 0$
$k^f_1 \dashrightarrow 1$
$k^h_0 \dashrightarrow 0$
$k^h_1 \dashrightarrow 1$

$\mathbf{k^a_0}$
$\mathbf{k^b_1}$
$\mathbf{k^d_0}$
$\mathbf{k^e_1}$
$\mathbf{k^d_0}$
$\mathbf{k^e_0}$

simulated



GarbleInput(x)

output table

$k^g_0 \dashrightarrow \mathbf{y_1}$
$k^g_1 \dashrightarrow 1-y_1$
$k^f_0 \dashrightarrow \mathbf{y_2}$
$k^f_1 \dashrightarrow 1-y_2$
$k^h_0 \dashrightarrow \mathbf{y_3}$
$k^h_1 \dashrightarrow 1-y_3$

$\mathbf{k^a_0}$
$\mathbf{k^b_0}$
$\mathbf{k^d_0}$
$\mathbf{k^e_0}$
$\mathbf{k^d_0}$
$\mathbf{k^e_0}$

Selective to Adaptive

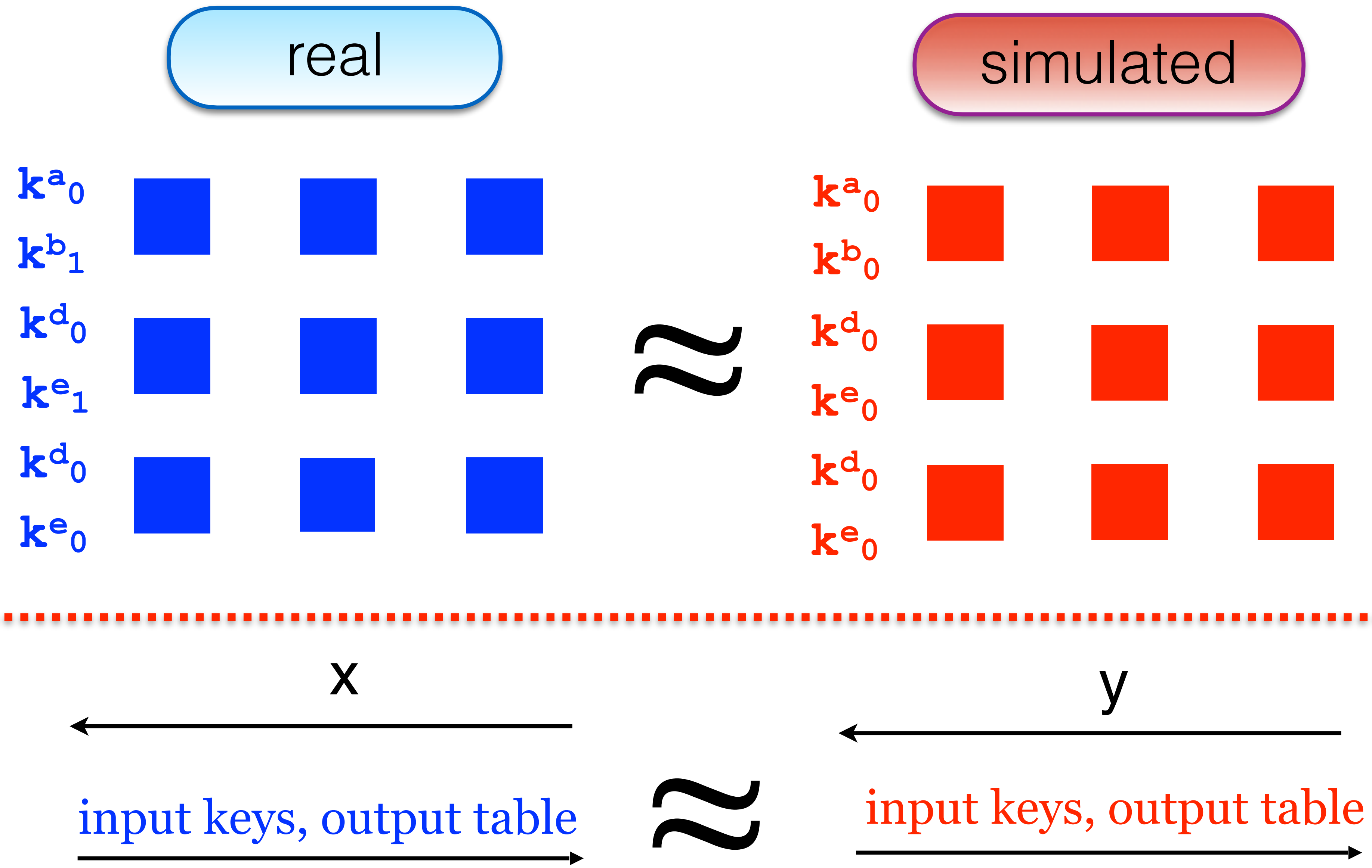
Real Garbling

Simulation

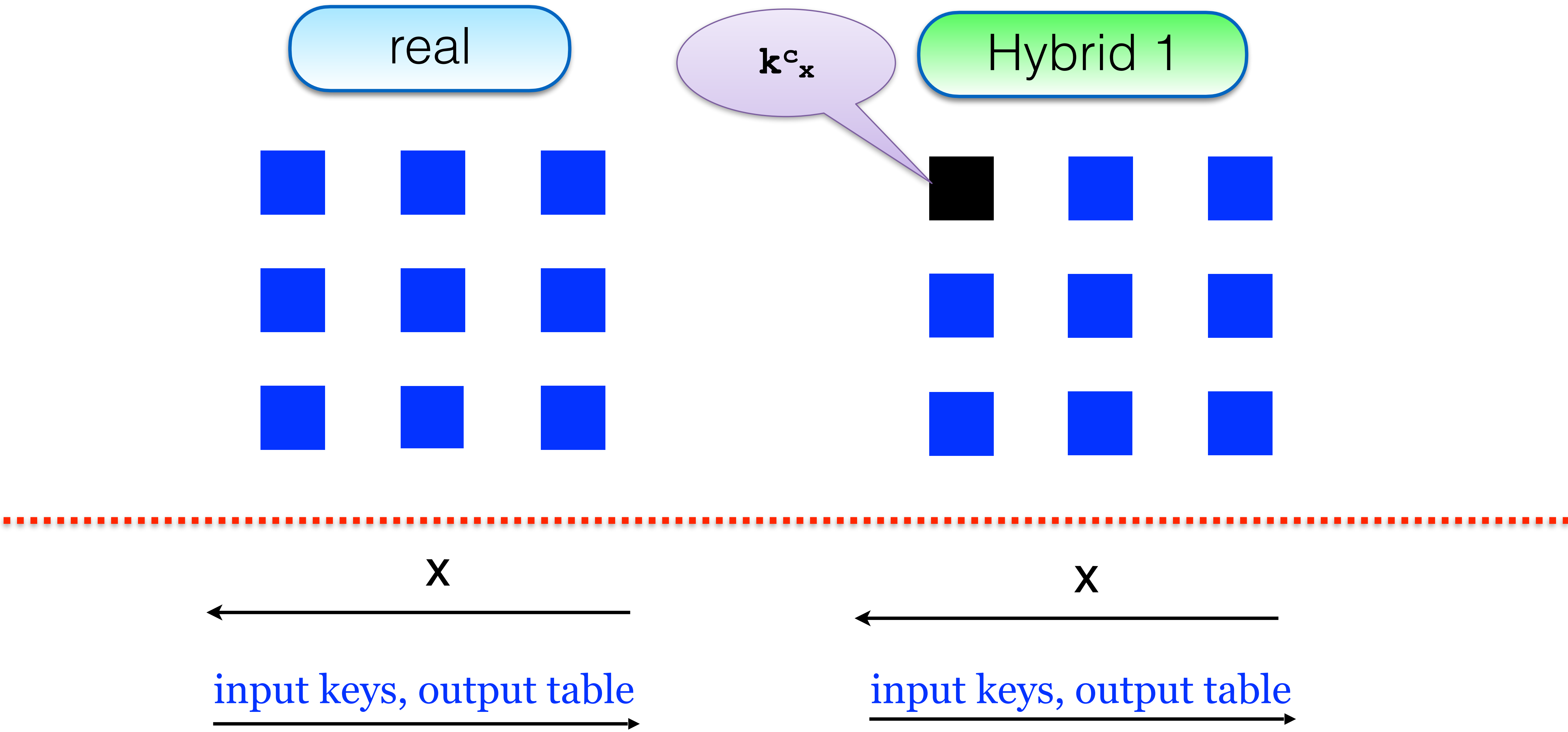
Indistinguishability proof

on-line complexity is at least
input size+output size

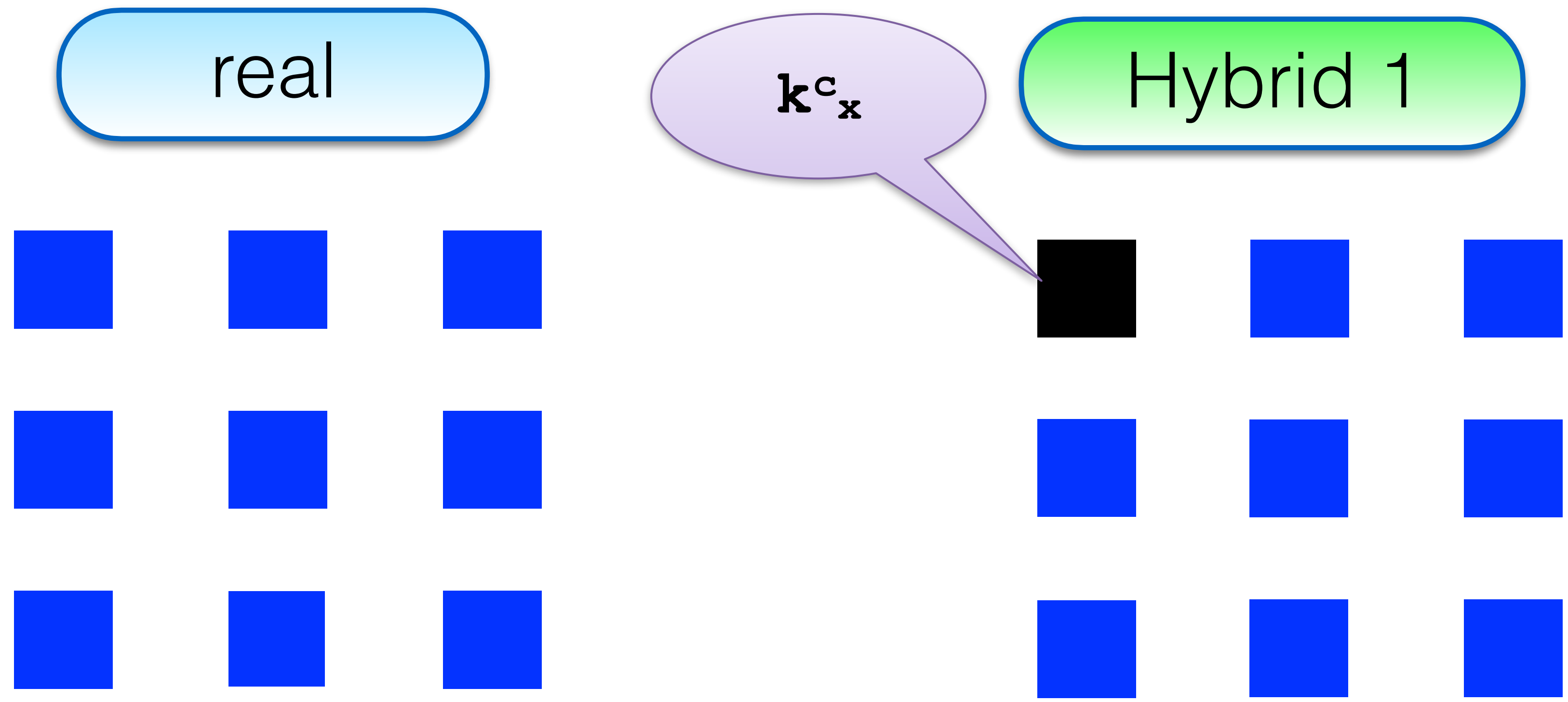
Indistinguishability Proof



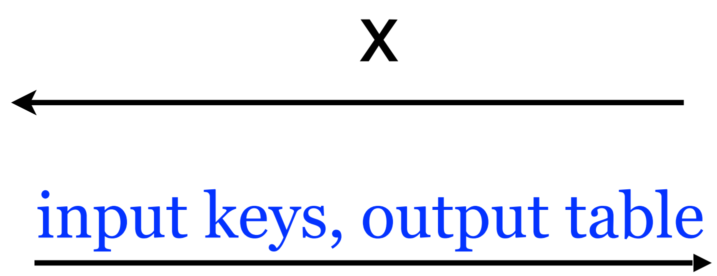
Hybrid distributions



Hybrid distributions



We don't know the input x,
The hybrid game is not
well-defined



Selective to Adaptive

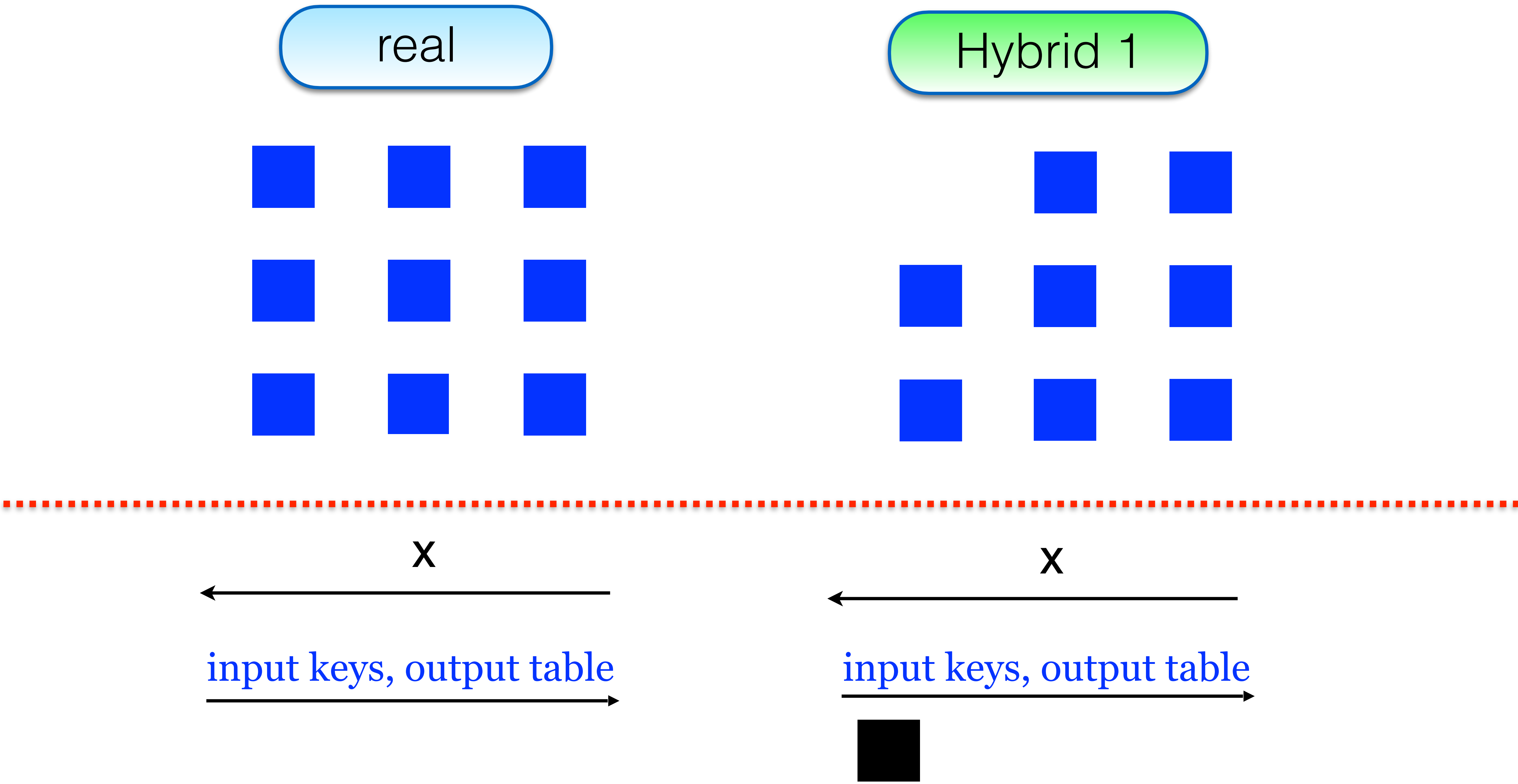
Real Garbling

Simulation

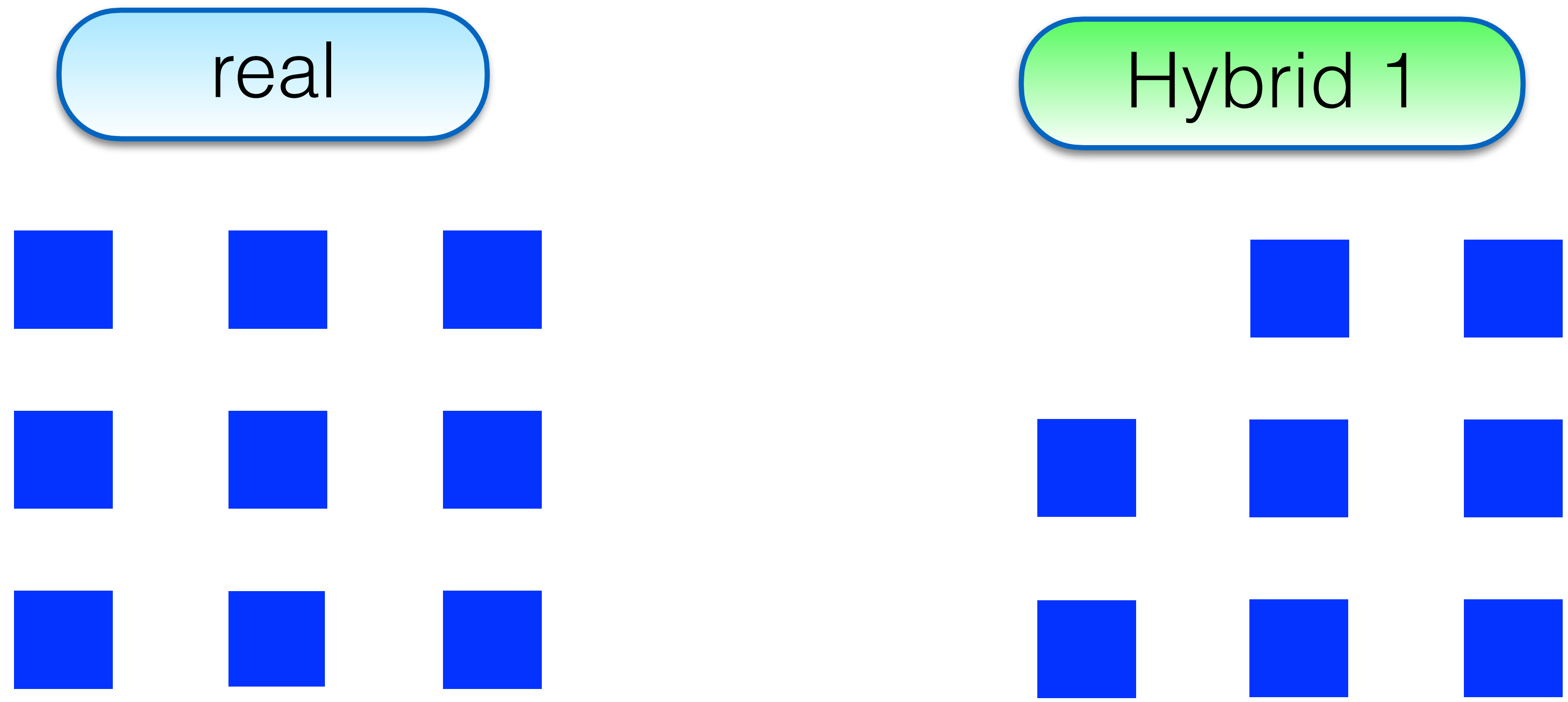
~~Indistinguishability proof~~

on-line complexity is at least
input size+output size

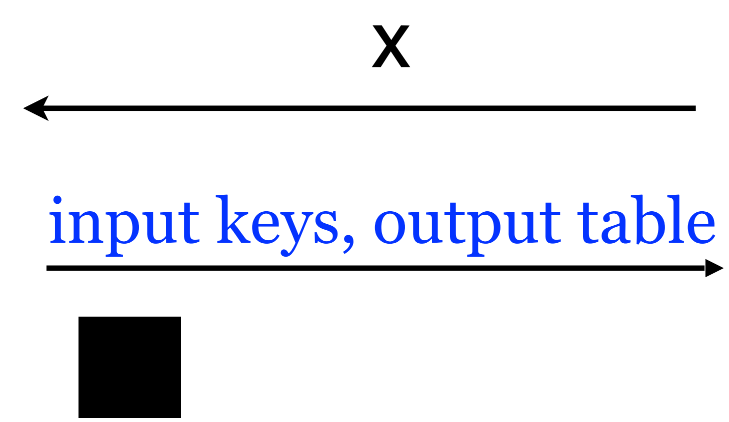
Hybrid distributions



Hybrid distributions



It's well-defined, but at the end, you're sending the entire circuit when online



Outline

- Yao's garbling scheme
- Selective → Adaptive Yao: Difficulties
- Our approach

Ideas

Find a way to define hybrids with InputDepSimGate

Be able to garble a gate after seeing the input

Somewhere Equivocal Encryption

Keep the number of InputDepSimGate as small as possible

Find a way to turn some InputDepSimGate into SimGate

Smarter Hybrid Arguments

Somewhere Equivocal Encryption

OWF
Boyle, Gilboa, Ishai
Distributed Point function

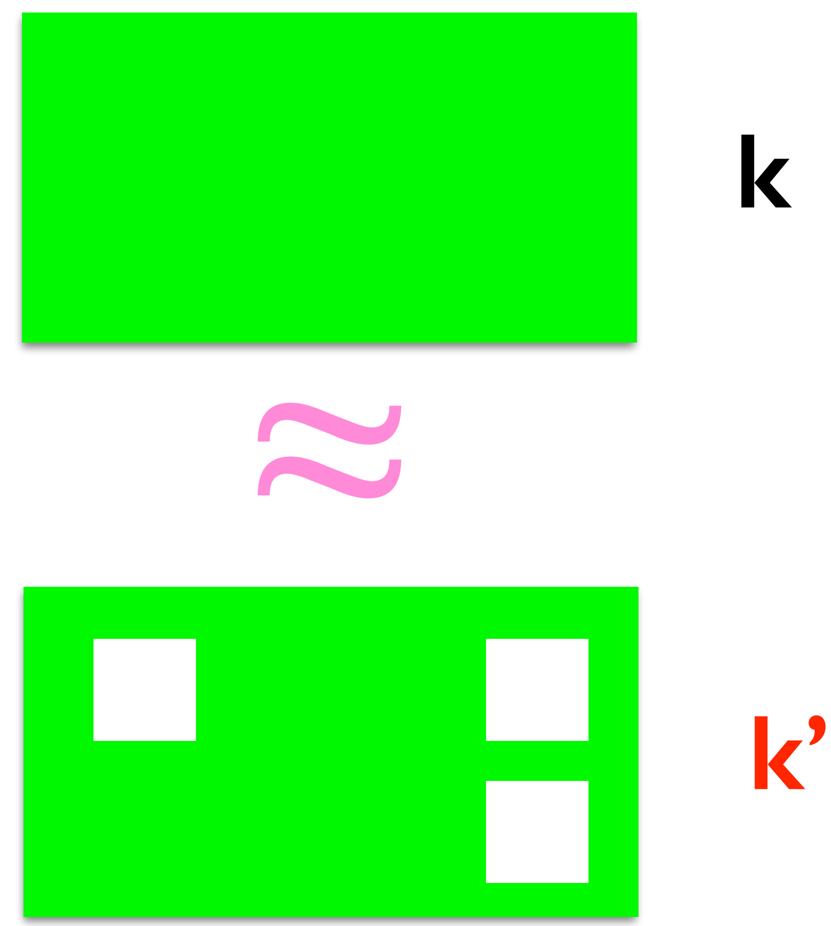
$\overline{\mathbf{m}} = m_1, m_2, m_3, m_4, m_5, m_6$

|key| grows with # holes

honest procedure

KeyGen → **k**

$\text{Enc}_{\mathbf{k}}(\overline{\mathbf{m}}) \rightarrow c$

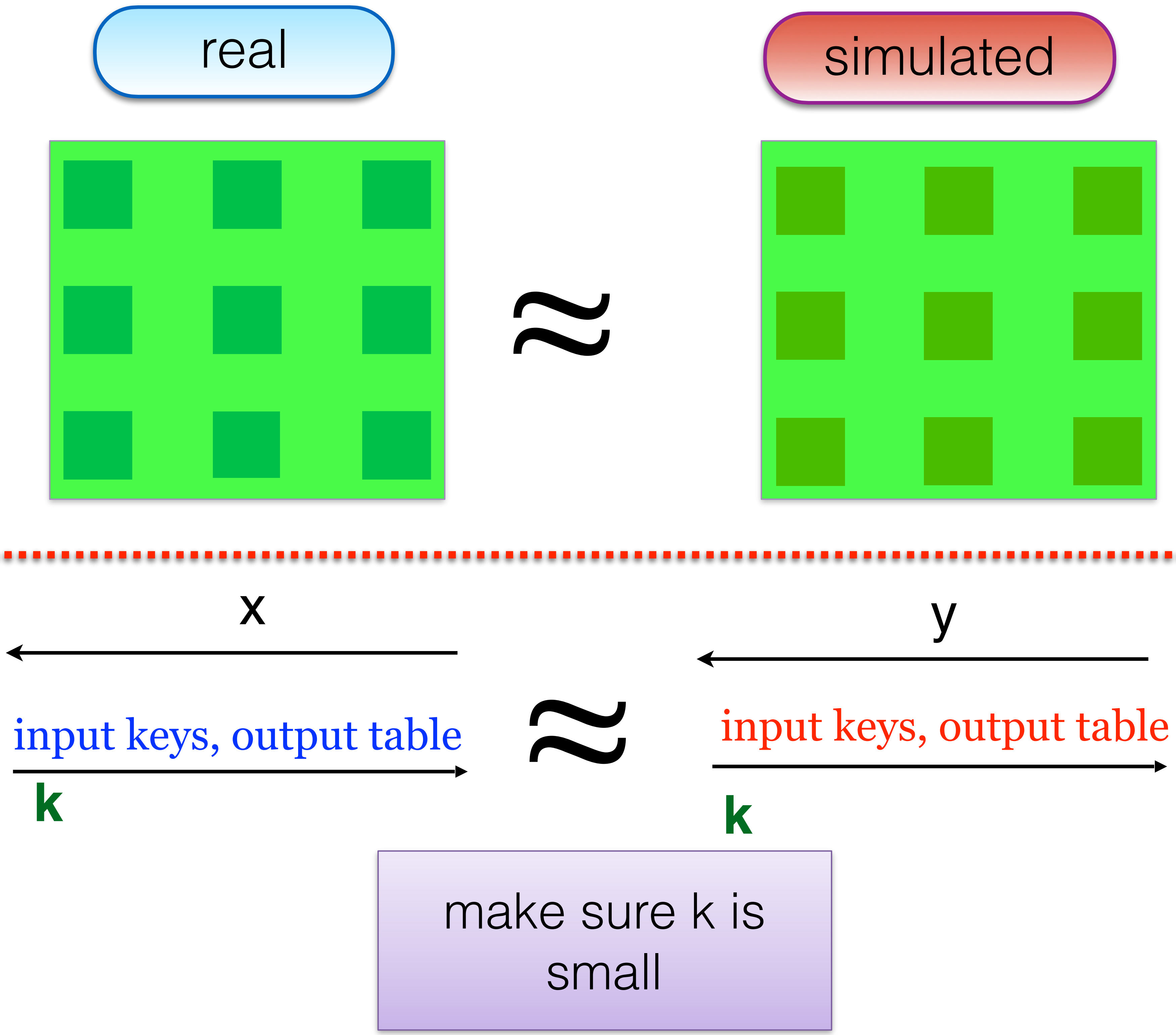


simulated procedure

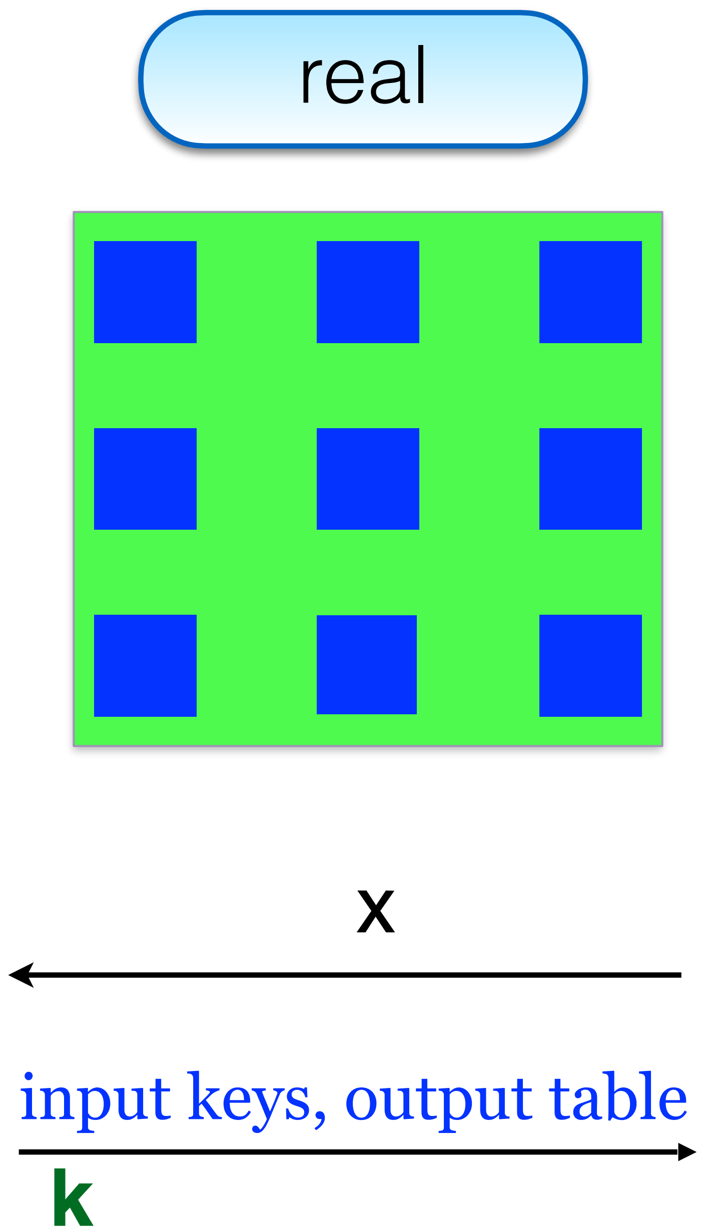
$\text{SimEnc}(m_1, *, *, m_4, *, m_6) \rightarrow (c, s)$

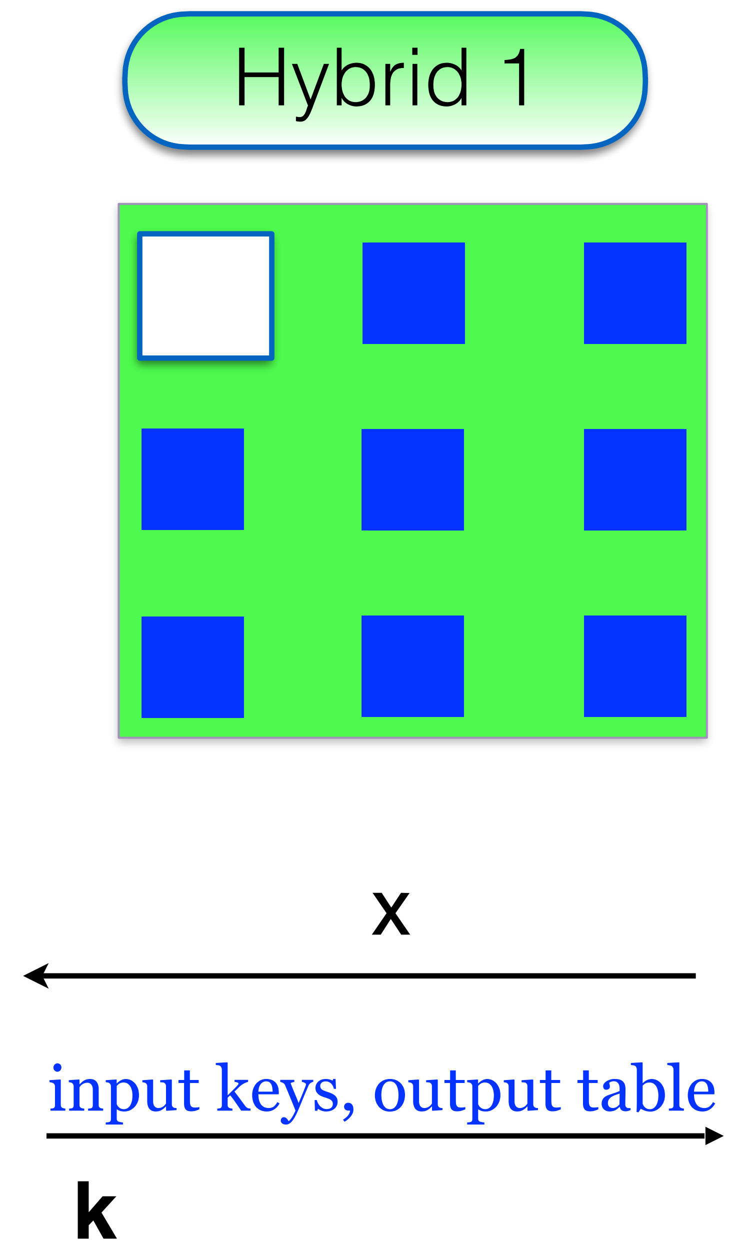
$\text{SimKey}(m_2, m_3, m_{5,s}) \rightarrow$ **k'**

Our Construction

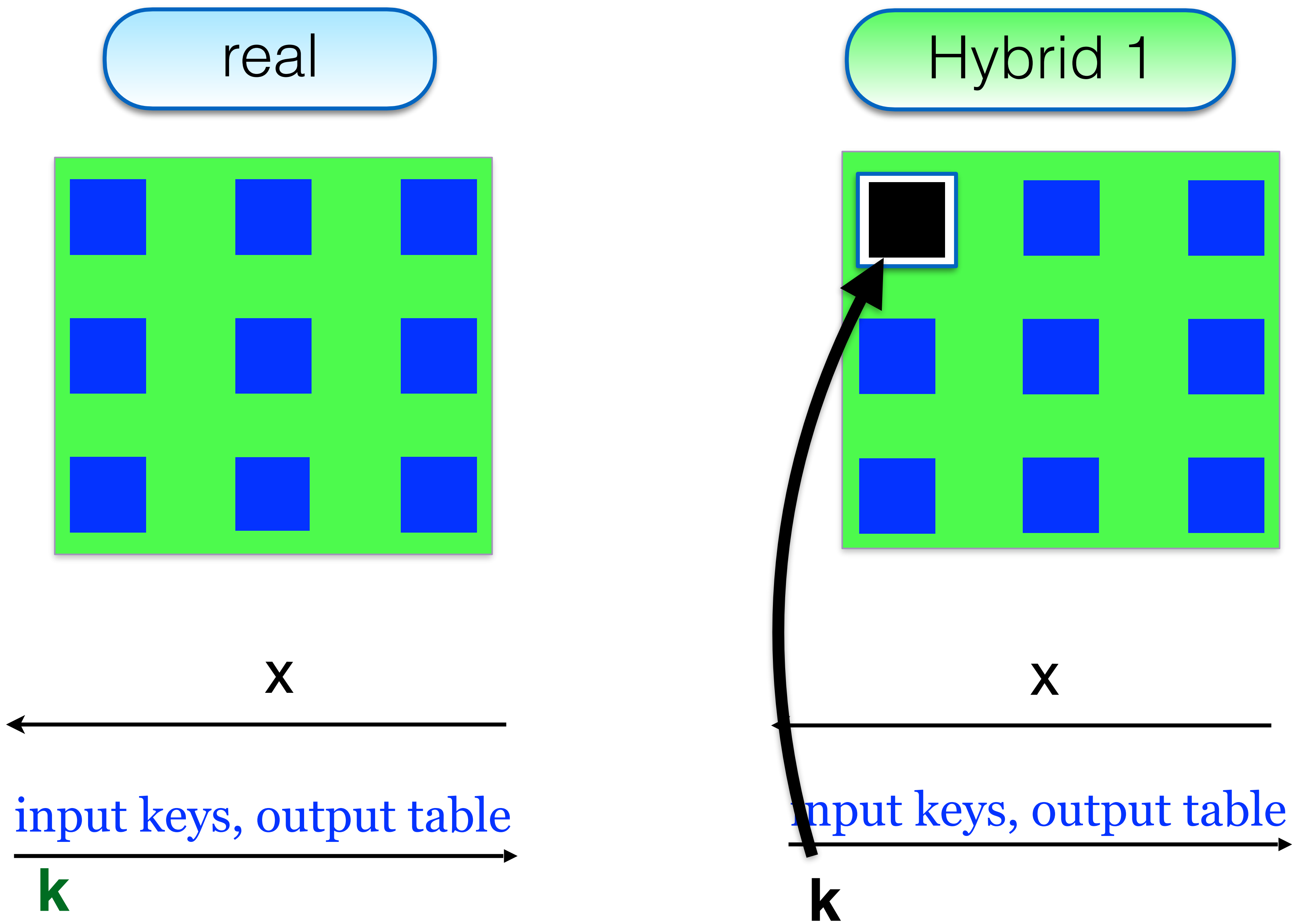


Hybrids

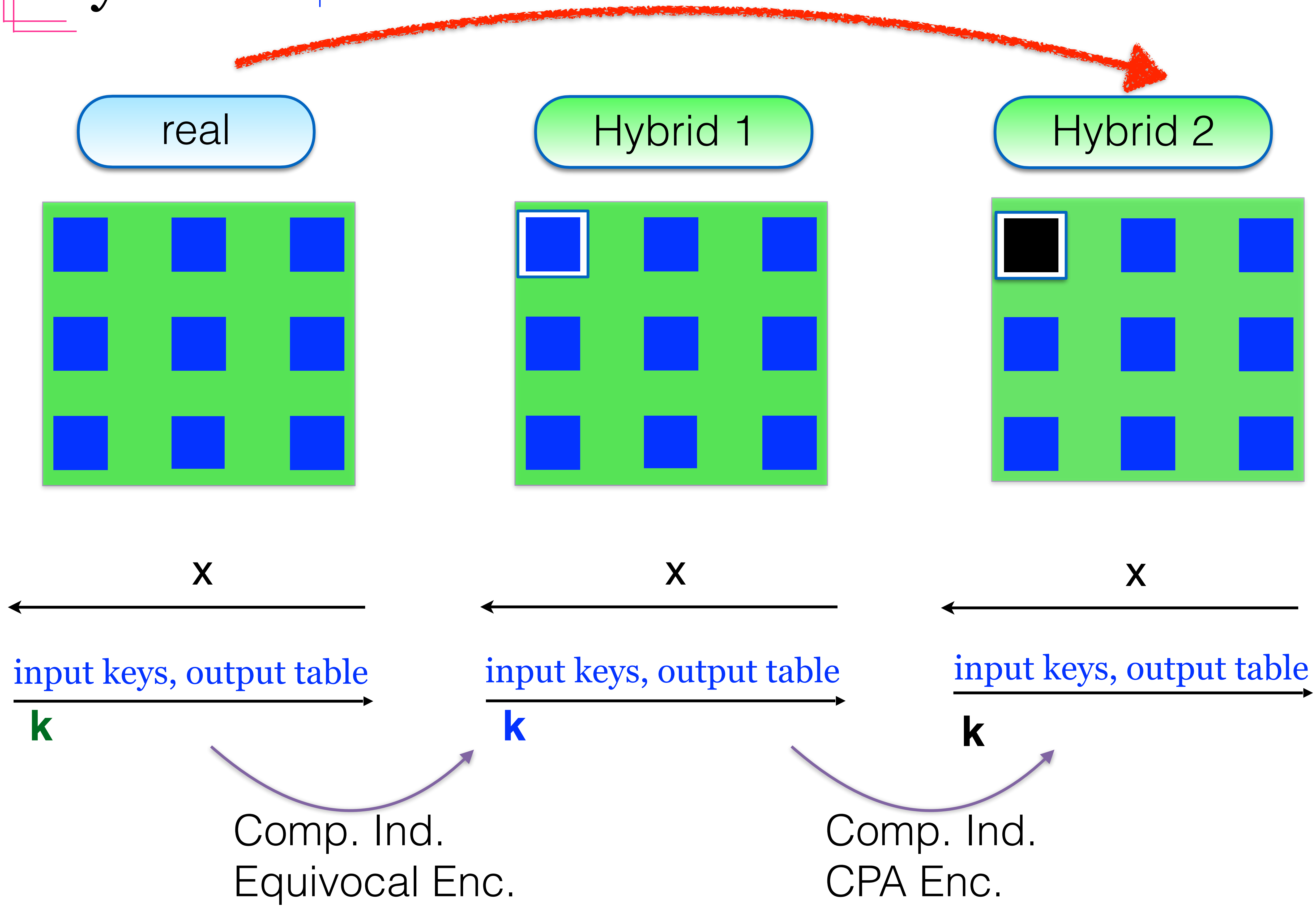

real
x
input keys, output table
k


Hybrid 1
x
input keys, output table
k

Hybrids


real
Hybrid 1
x
x
input keys, output table
input keys, output table
k
k

Hybrids

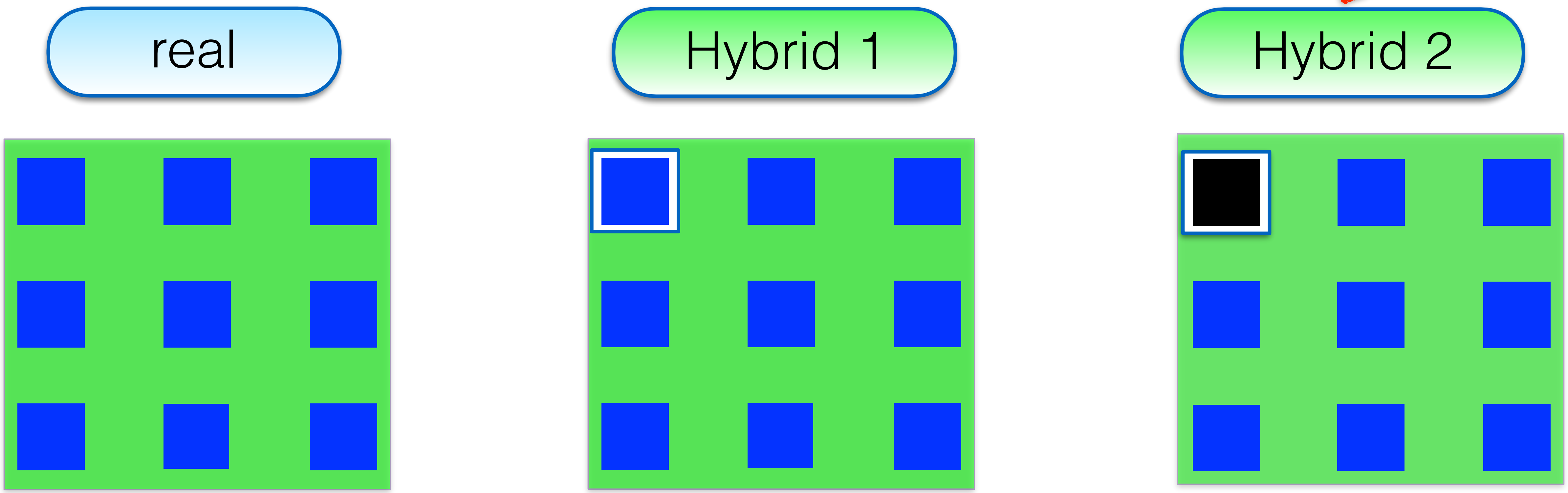


real
Hybrid 1
Hybrid 2
x
x
x
input keys, output table
input keys, output table
input keys, output table
k
k
k
Comp. Ind.
Equivocal Enc.
Comp. Ind.
CPA Enc.

Hybrids

Can we keep going, same as the selective security hybrids?

we need to follow the rules

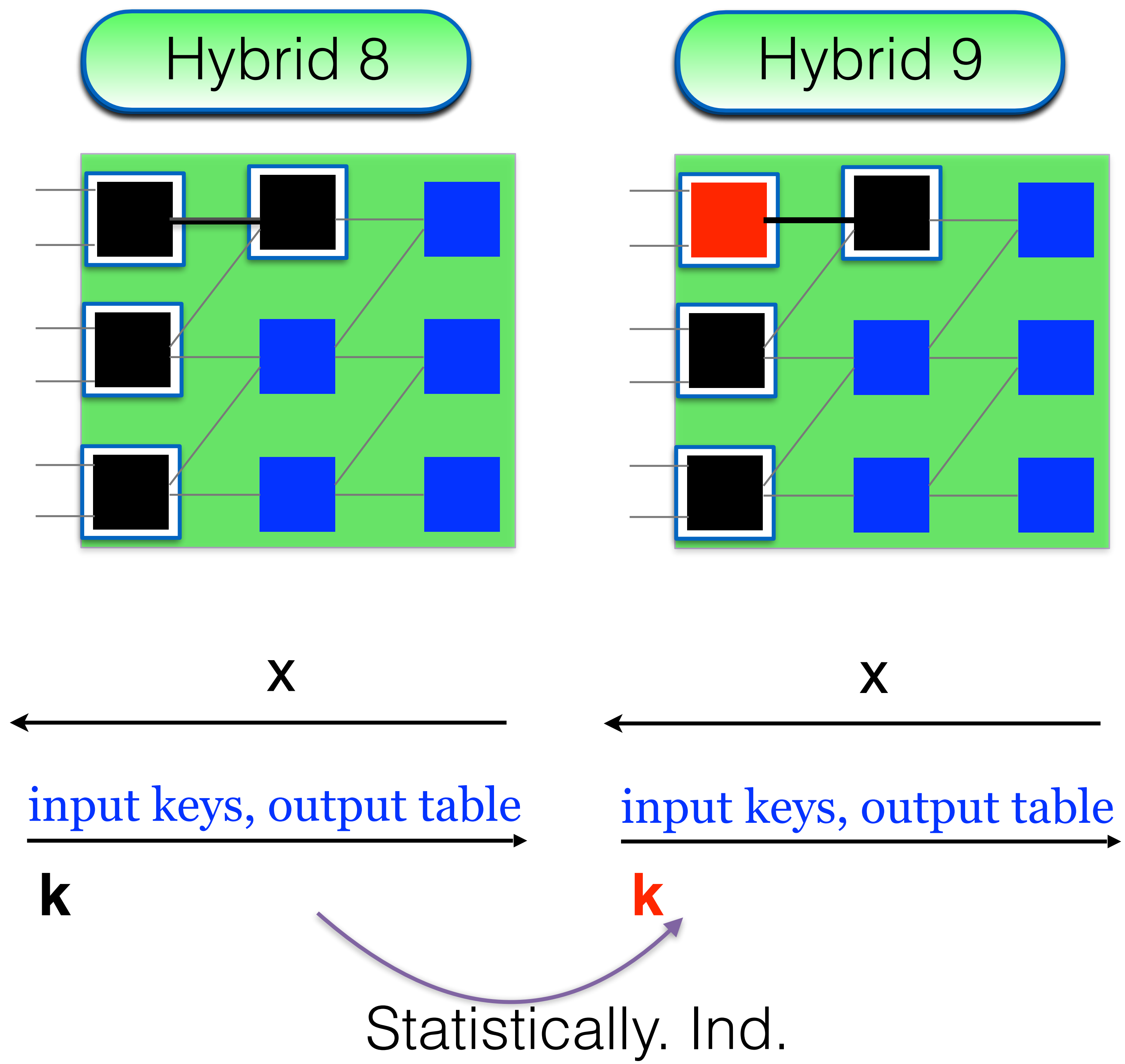


0. Every **Black** gate needs a hole!
1. Input gates can be turned **Black**
2. A gate with all its inputs coming from **Black** gates, can be turned **Black**
3. Once the entire circuit is **Black** we can turn all the gates **Red**.

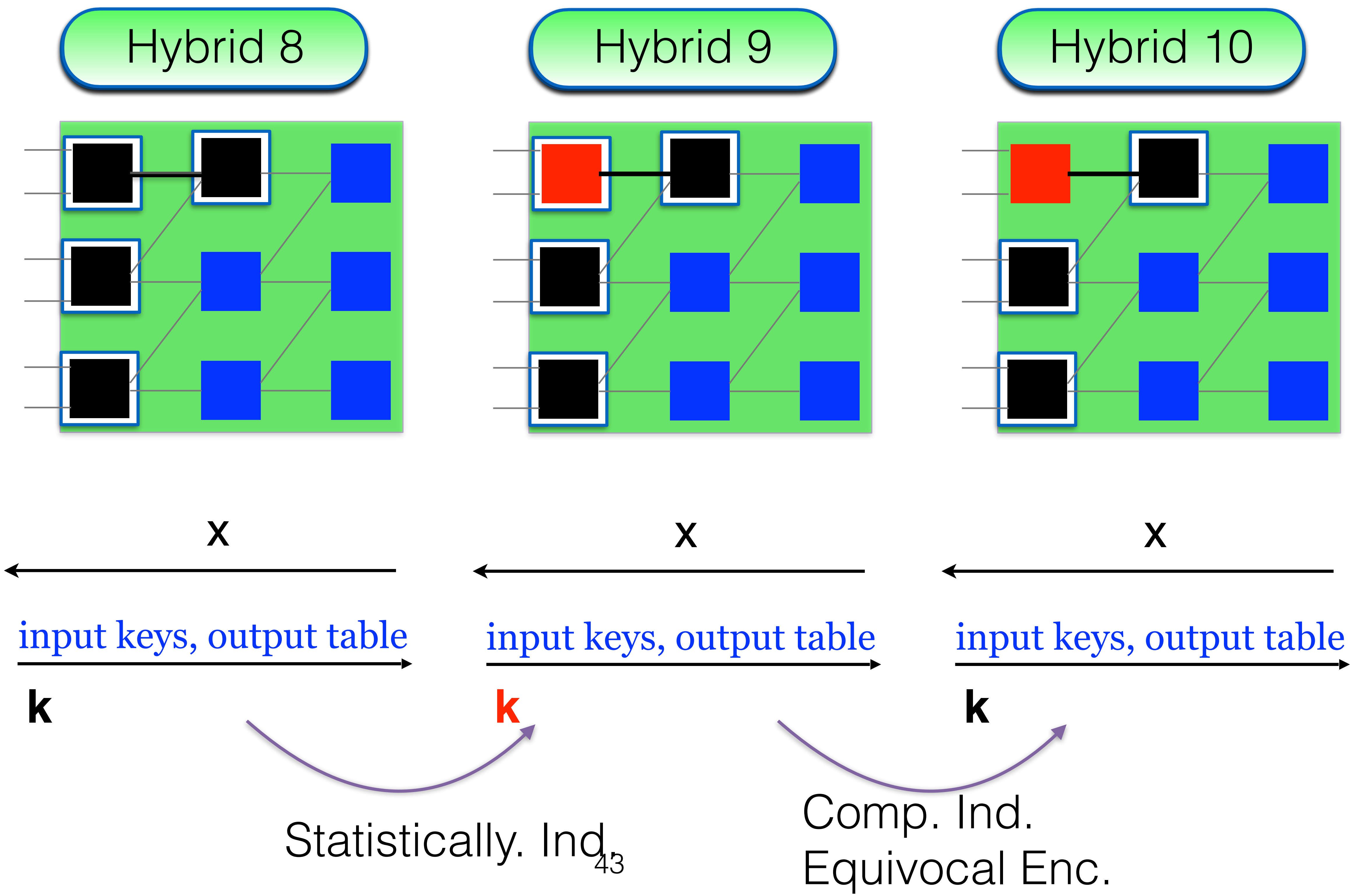
We refine the rules

0. Every **Black** gate needs a hole!
1. Input gates can be turned **Black**
2. A gate with all its inputs coming from **Black** gates, can be turned **Black**
3. Once the entire circuit is **Black** we can turn all the gates **Red**.

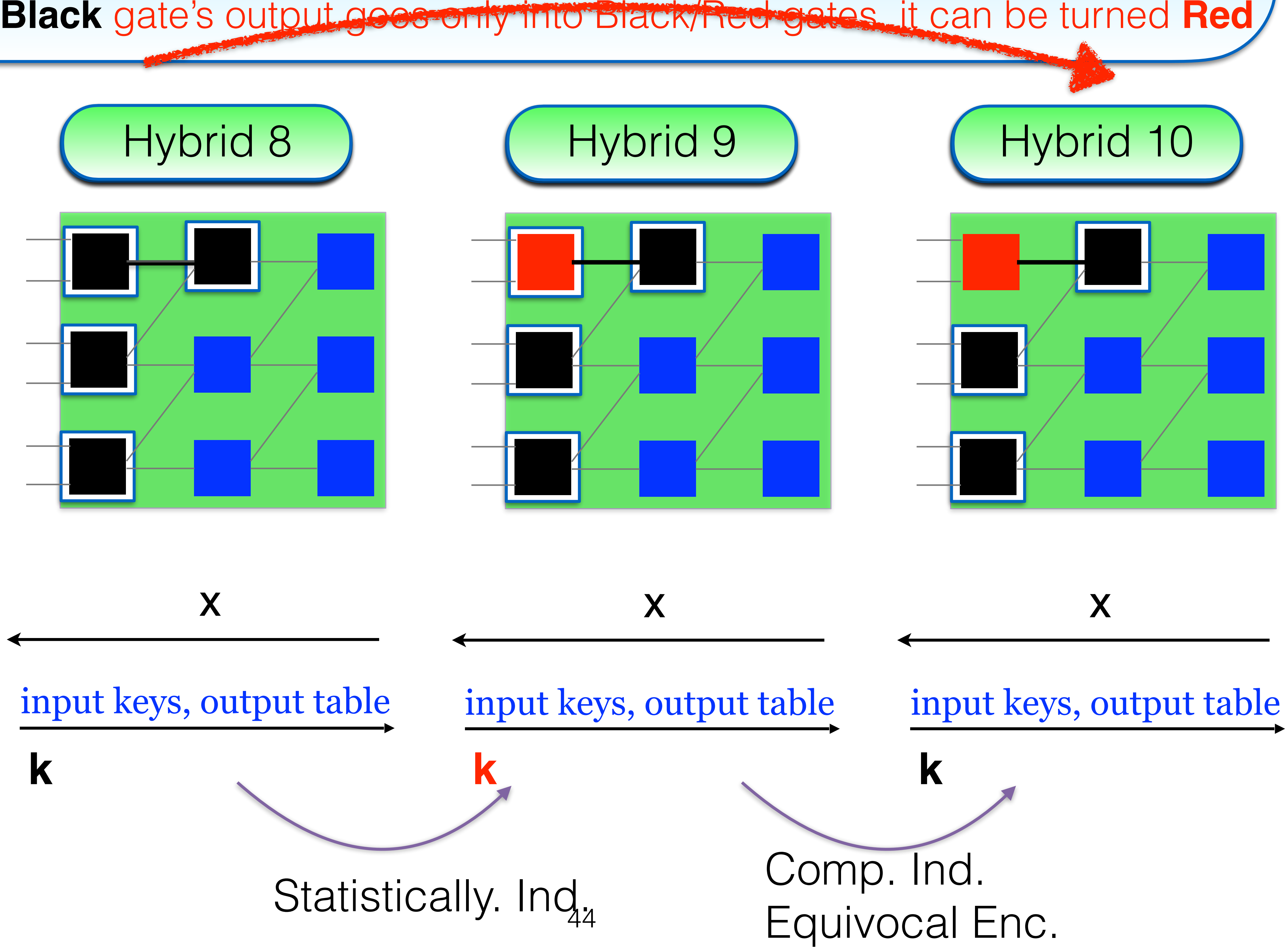
When can we turn Black into Red?



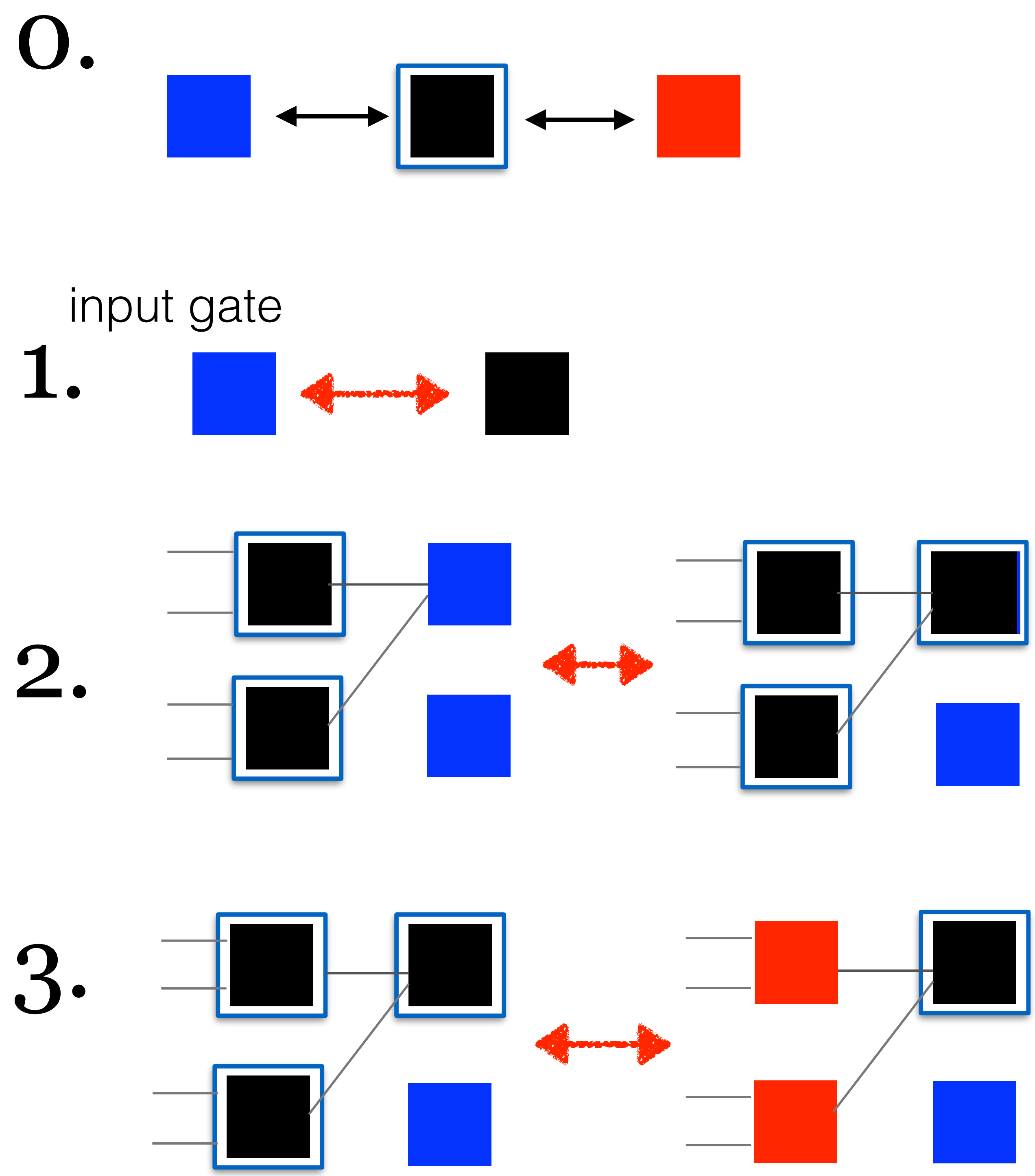
0. Every **Black** gate needs a hole!
1. Input gates can be turned **Black**
2. A gate with all its inputs coming from **Black** gates, can be turned **Black**
3. If **Black** gate's output goes only into Black/Red gates, it can be turned **Red**



0. Every **Black** gate needs a hole!
1. Input gates can be turned **Black**
2. A gate with all its inputs coming from **Black** gates, can be turned **Black**
3. If **Black** gate's output goes only into Black/Red gates, it can be turned **Red**

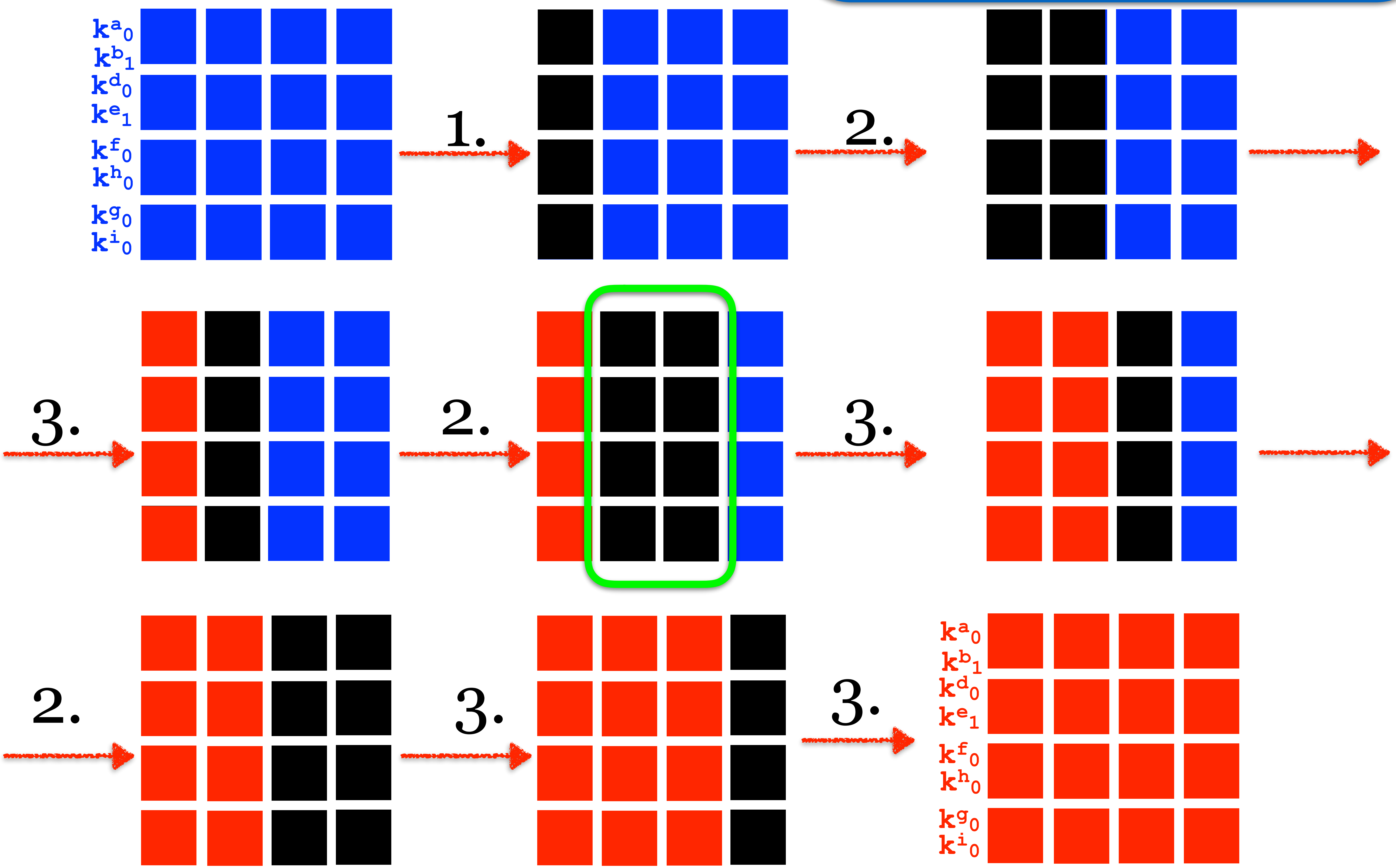


0. Every **Black** gate needs a hole!
1. Input gates can be turned **Black**
2. A gate with all its inputs coming from **Black** gates, can be turned **Black**
3. If **Black** gate's output goes only into Black/Red gates, it can be turned **Red**



Hybrid distributions

Smarter Hybrid Arguments

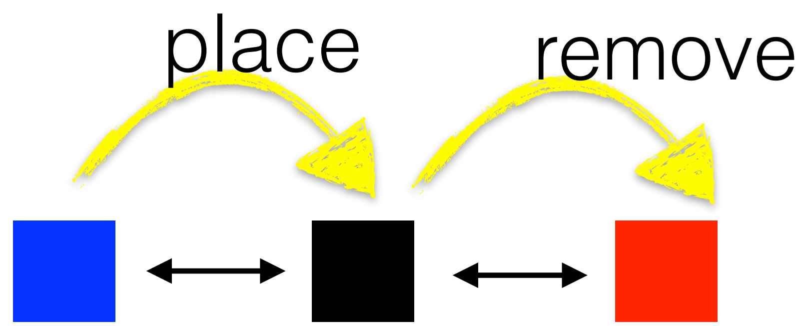


Smarter Hybrid Arguments

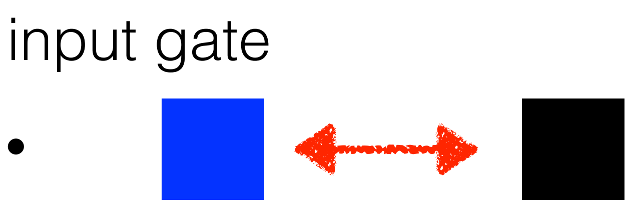
That's one strategy, that gives us a hybrid argument with
1. Number of holes O(width)
2. Number of hybrids O(#gates)

We can generalize this strategy. We take advantage of pebbling games

0.

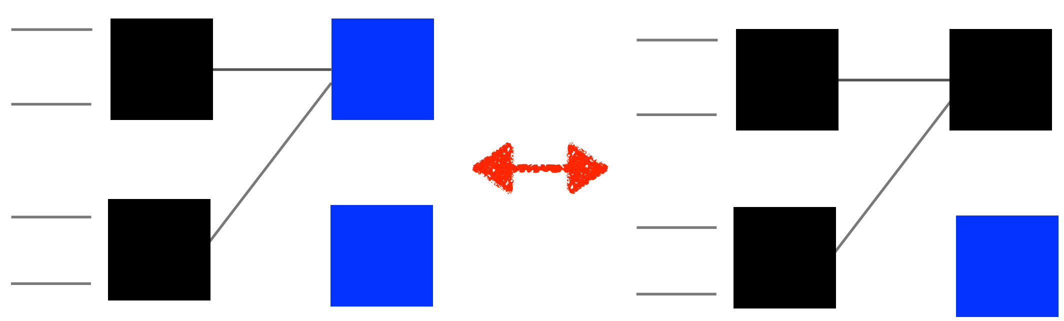


1.



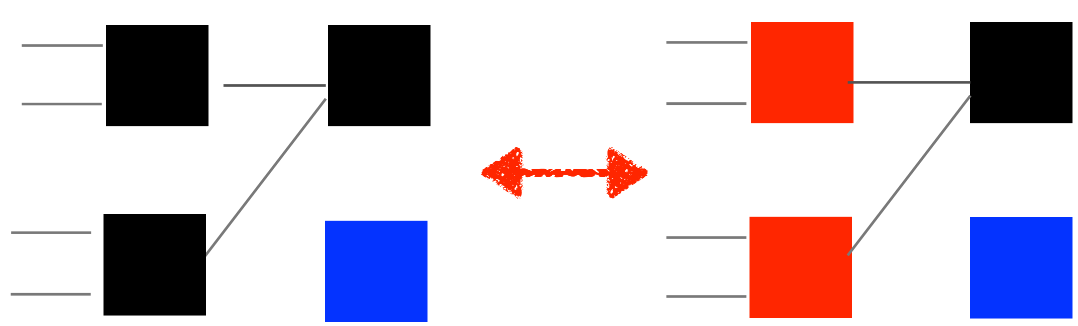
a pebble can be placed on an input-node

2.

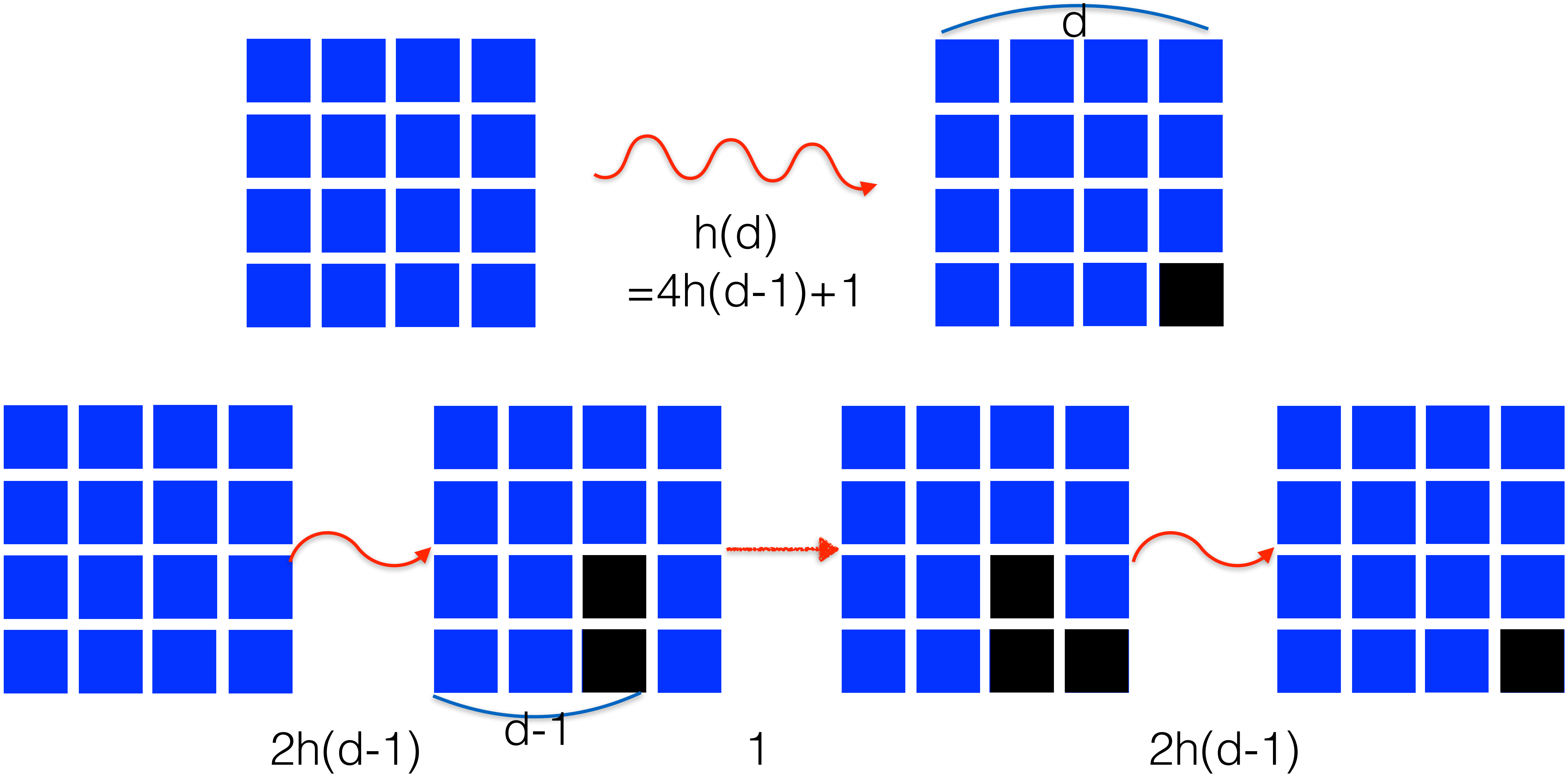


a pebble can be placed on a node if its predecessors already have pebbles

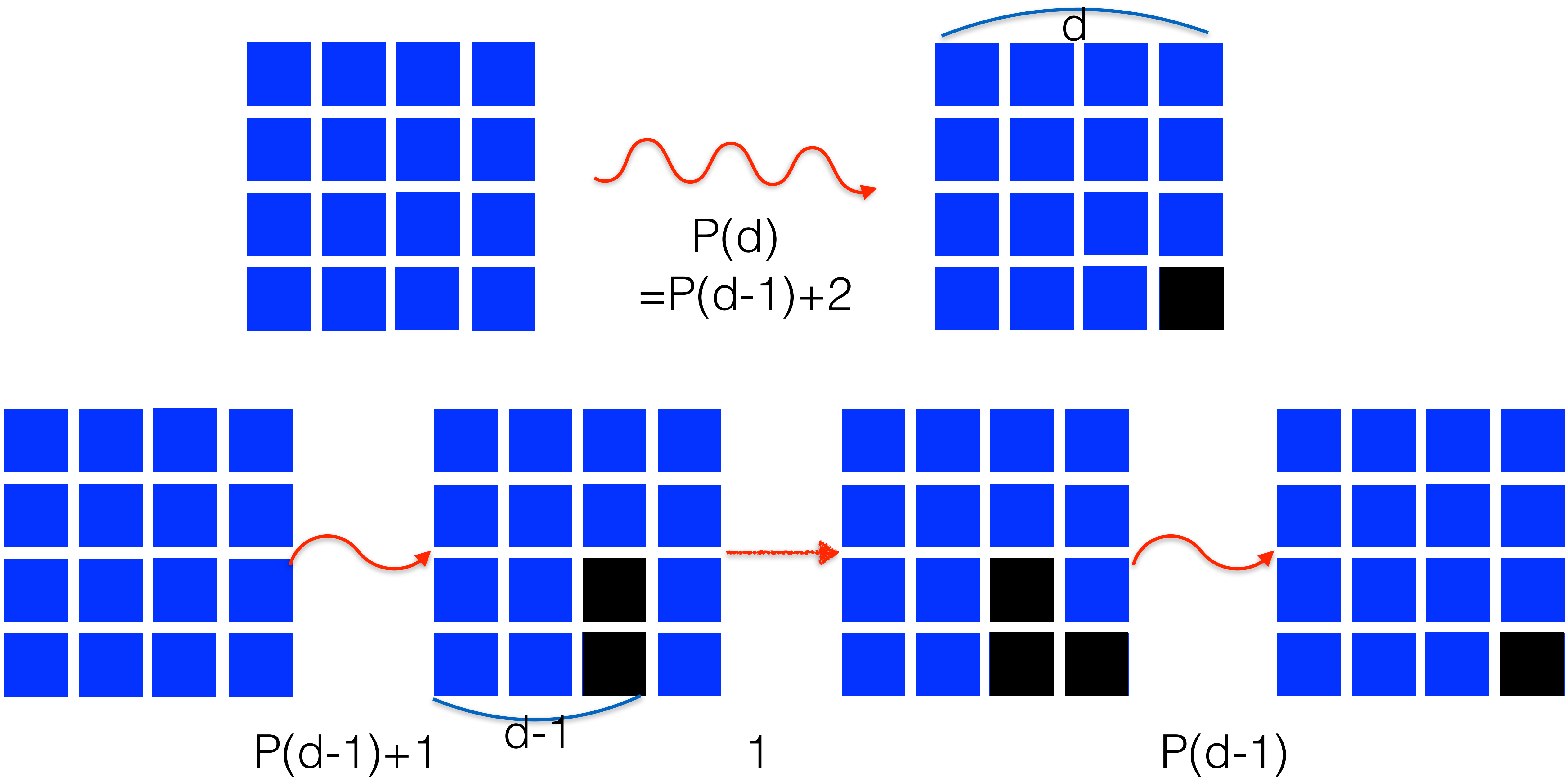
3.



A pebble can be removed if its successors have pebbles



$$h(d)=O(2^{2d})$$



$h(d)=O(2^{2d})$
$P(d)=O(d)$

Smarter Hybrid Arguments

That's one strategy, that gives us a hybrid argument with

1. Number of holes O(width)
2. Number of hybrids O(#gates)

That's another strategy, that gives us a hybrid argument with

1. Number of holes O(depth)
2. Number of hybrids O($2^{2(depth)|}$ |C|)

There can be other pebbling strategies that are more efficient for a specific class of circuits.

1. The security parameter grows with #hybrids, $\lambda > poly(\log(h))$
2. The size of the key grows with #pebbles. $k = poly(\lambda)(|x|+|y|+P)$

Summary

- We show the first adaptive scheme with O(width) or O(depth) online complexity
    - We recast Yao's proof as pebbling game
    - We introduce a encryption scheme for somewhere equivocation
    - Our framework allows different strategies/ different parameter

Thank you!