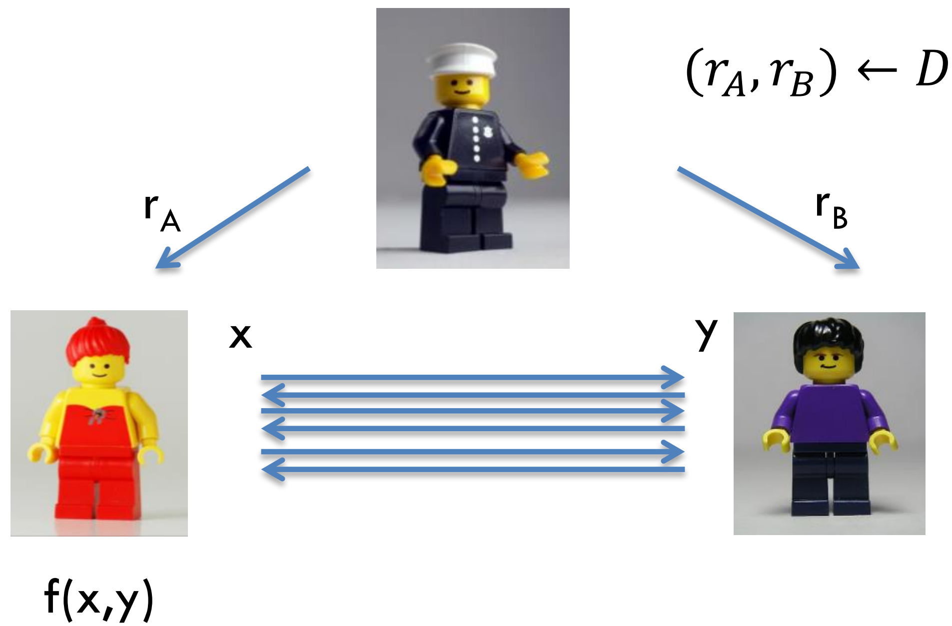
5th Bar-Ilan Winter School on Cryptography
Advances in Practical Multiparty Computation

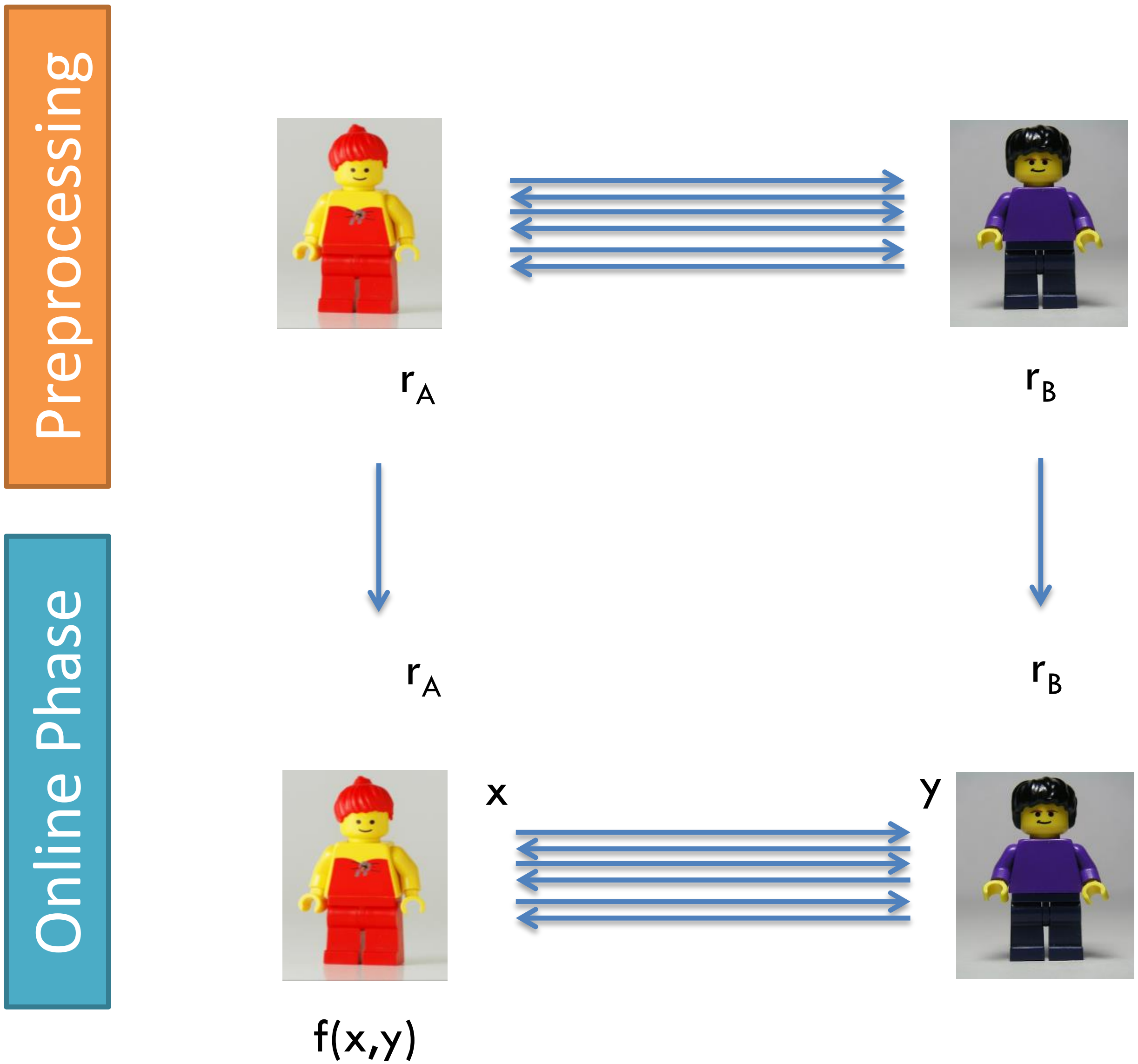
"Tiny OT" – Part 2

A ~~New~~ (4 years old) Approach to Practical Active-Secure Two-Party Computation

Claudio Orlandi, Aarhus University

Trusted Dealer




Preprocessing
Online Phase
r_A
r_B
r_A
r_B
x
y
f(x,y)

TinyOT authenticated bits

- $[x] = ((x_A, k_A, m_A) , (x_B, k_B, m_B))$ s.t.
  - $m_B = k_A + x_B \Delta_A$ (symmetric for m_A)
  - Δ_A, Δ_B is the same for all wires.
  - MACs, keys are k-bit strings.

- Very similar to Oblivious Transfer
  - Sender has two messages u_0, u_1
  - Receiver has a bit b and learns u_b
  - Set $u_0=k$, $u_1=k+\Delta$, $b=x$
    then $u_b=k+x\Delta$

Two probems:

- ***Efficiency:*** OT requires public key primitives, inherently efficient

The Crypto Toolbox

Weaker assumption ← Stronger assumption

OTP >> SKE >> PKE >> FHE >> Obfuscation

More efficient ← Less efficient

Two probems:

- ***Efficiency:*** OT requires public key primitives, inherently efficient

- ***Security:*** If we authenticated more than one bit, how do we make sure Bob uses the same value Δ?

- Two birds with one stone! Next hour:
  Active secure OT extension!

Authenticated Bits

x

$m_x = k_x + x\Delta$

OT

$(k_x,\ k_x+\Delta)$

y

$m_y = k_y + y\Delta$

OT

$(k_y,\ k_y+\Delta)$

“[z]=[x]+[y]”

$z = x + y$

$m_z = m_x + m_y$

$k_z = k_x + k_y$

“z=Open(B,[z])”

z, m_z

$m_z = k_z + z\Delta$

Authenticated Bits

x

$m_x = k_x + x\Delta$

OT

$(k_x, k_x+\Delta)$

y

$m_y = k_y + y\Delta + ey$

OT

$(k_y, k_y+\Delta+e)$

“[z]=[x]+[y]”

$z = x + y$

$m_z = m_x + m_y$

$k_z = k_x + k_y$

“z=Open(

z, m_z

Bob learns y (and therefore x)!
(should only learn XOR)

$m_z = k_z + z\Delta + ey$

Part 2: Active Secure OT Extension

- **Warmup: OT properties**

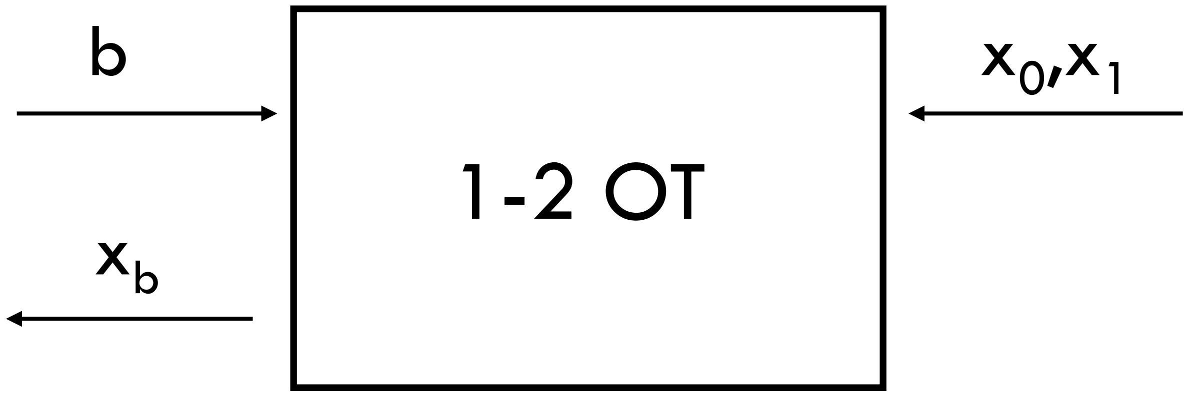
- Recap: Passive Secure OT Extension

- Active Secure OT Extension

OT

Receiver

Sender



- $x_b = x_0 + b(x_0 + x_1)$
- $x_b = (1+b)\, x_0 + b\, x_1$

OT = AND

Receiver

Sender

b

ab + c

1-2 OT

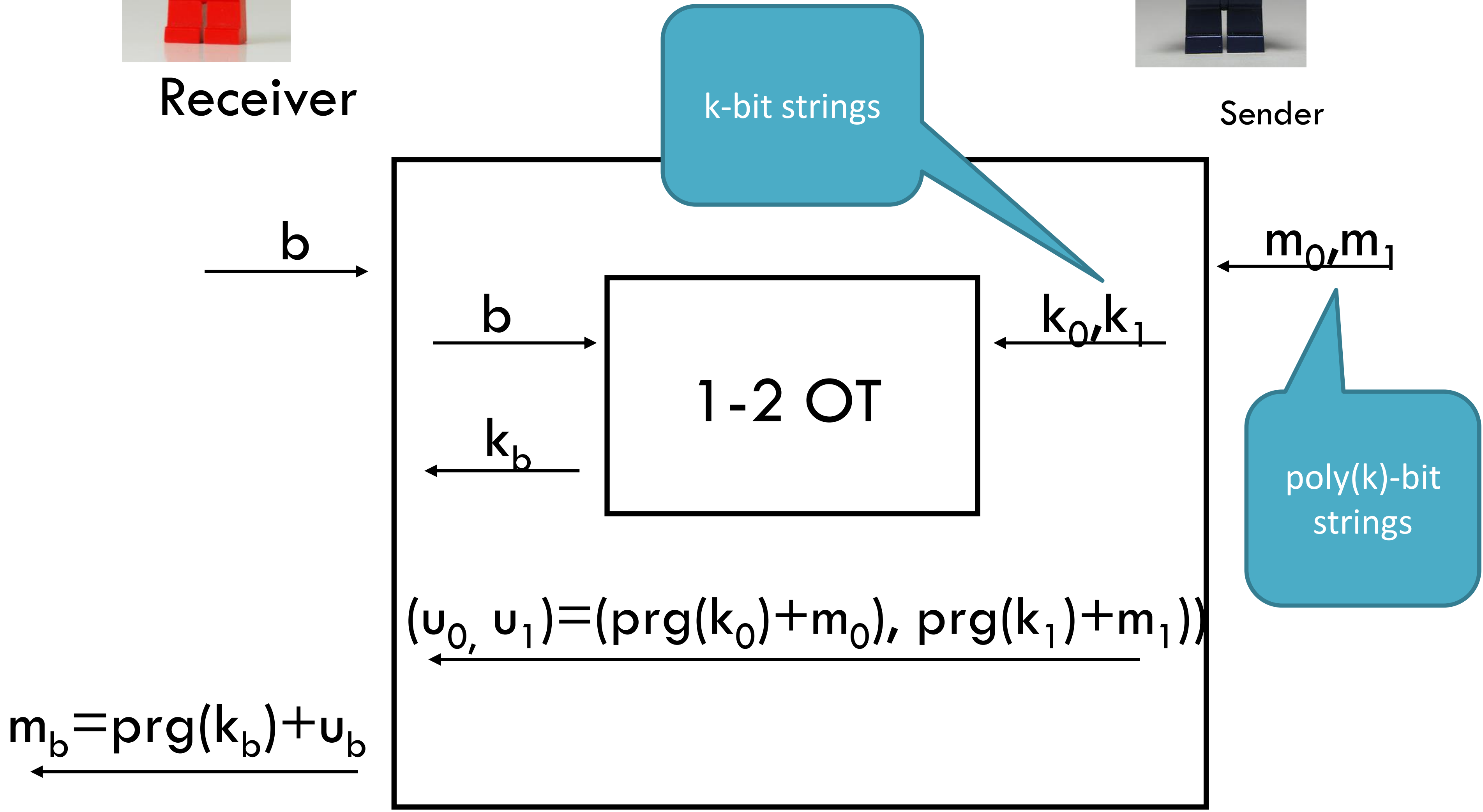
(a,a+c)

Bits

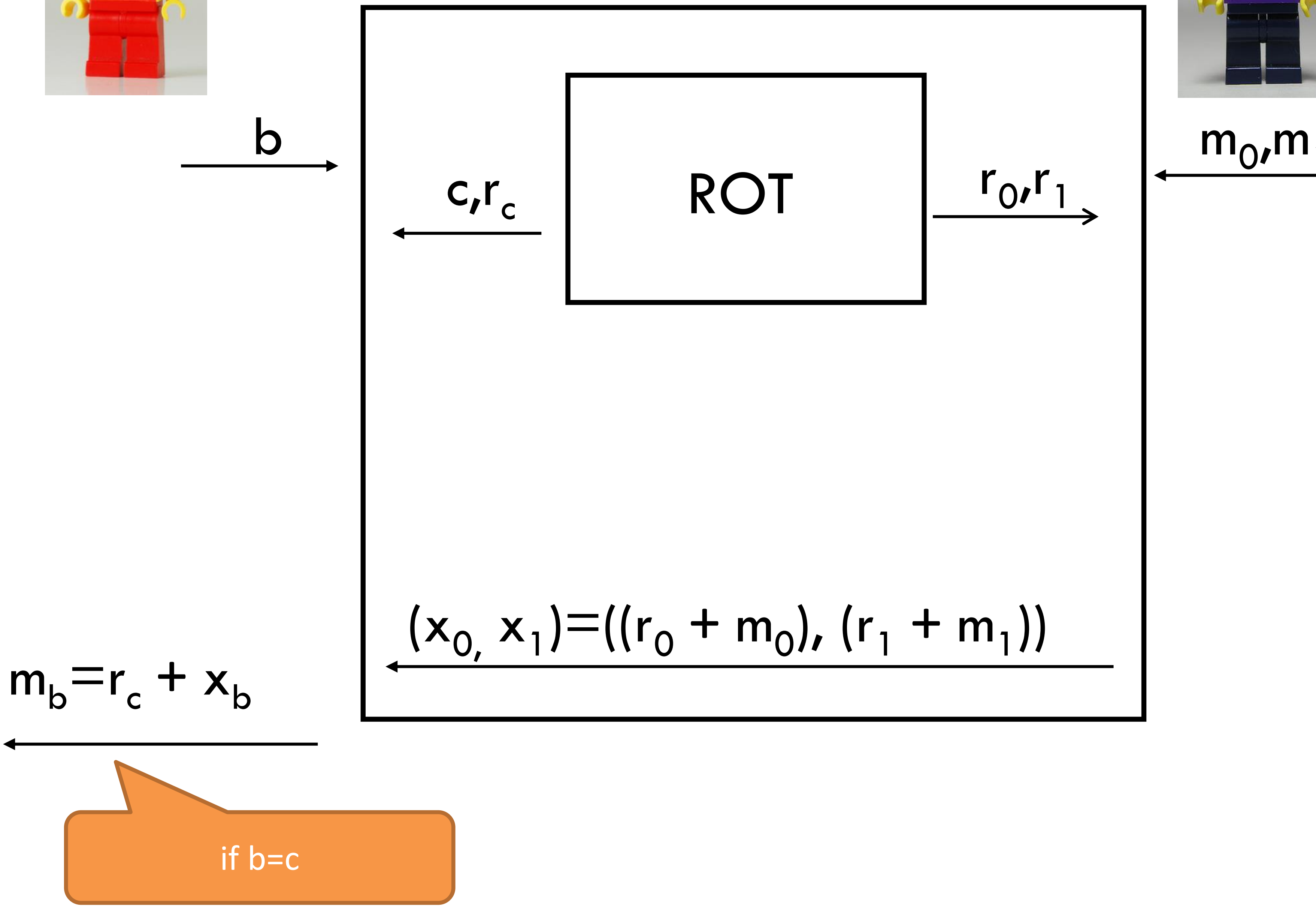
Stretching OT

Receiver

Sender



Random OT = OT



Random OT = OT

b →

m_0, m_1 ←

ROT

c, r_c ←

r_0, r_1 →

$d = b + c$ →

$(x_0, x_1) = (r_{0+d} + m_0), (r_{1+d} + m_1))$ ←

$m_b = r_c + x_b$ ←

Exercise: check that it works!

(R)OT is symmetric

bits

$s_0, s_1 \leftarrow$ ROT $\rightarrow b, y = s_b$

$c = s_0 + s_1$

$z = s_0$

$r_0 = y$

$r_1 = b + r_0$

$\xleftarrow{c,\ z = r_c}$

$\xrightarrow{r_0, r_1}$

No communication!

Exercise: check that it works

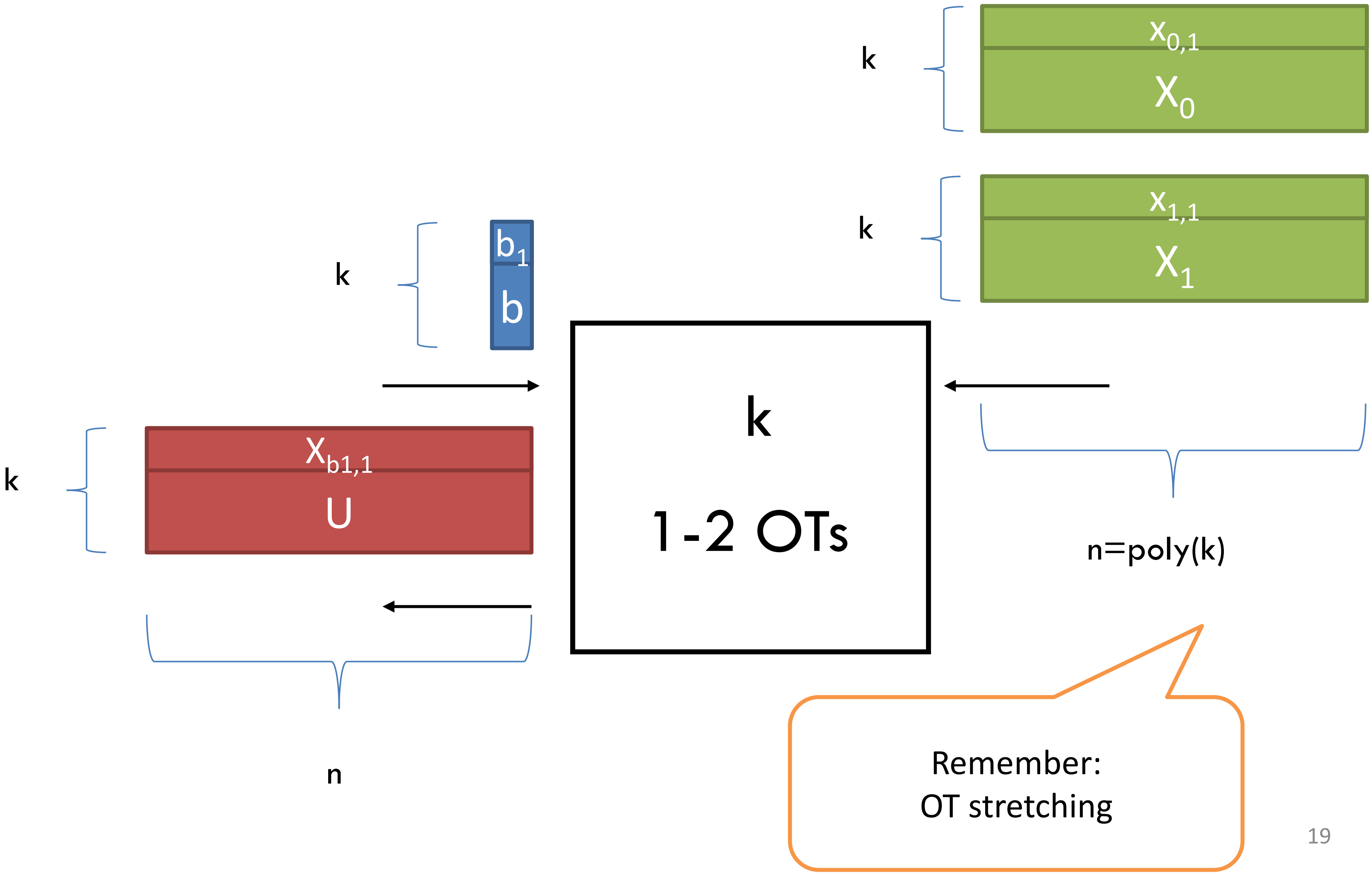
Part 2: Active Secure OT Extension

- Warmup: OT properties
- **Recap: Passive Secure OT Extension**
- Active Secure OT Extension

OT Extension

- OT pro(v/b)ably requires public-key primitivies
  - OT extension ≈ hybrid encryption
  - **Start from k "real" OTs**
  - **Turn them into poly(k) OTs using only few symmetric primitives per OT**

OT Extension, Pictorially



Condition for OT extension

X_0

$\oplus$

X_1

$=$

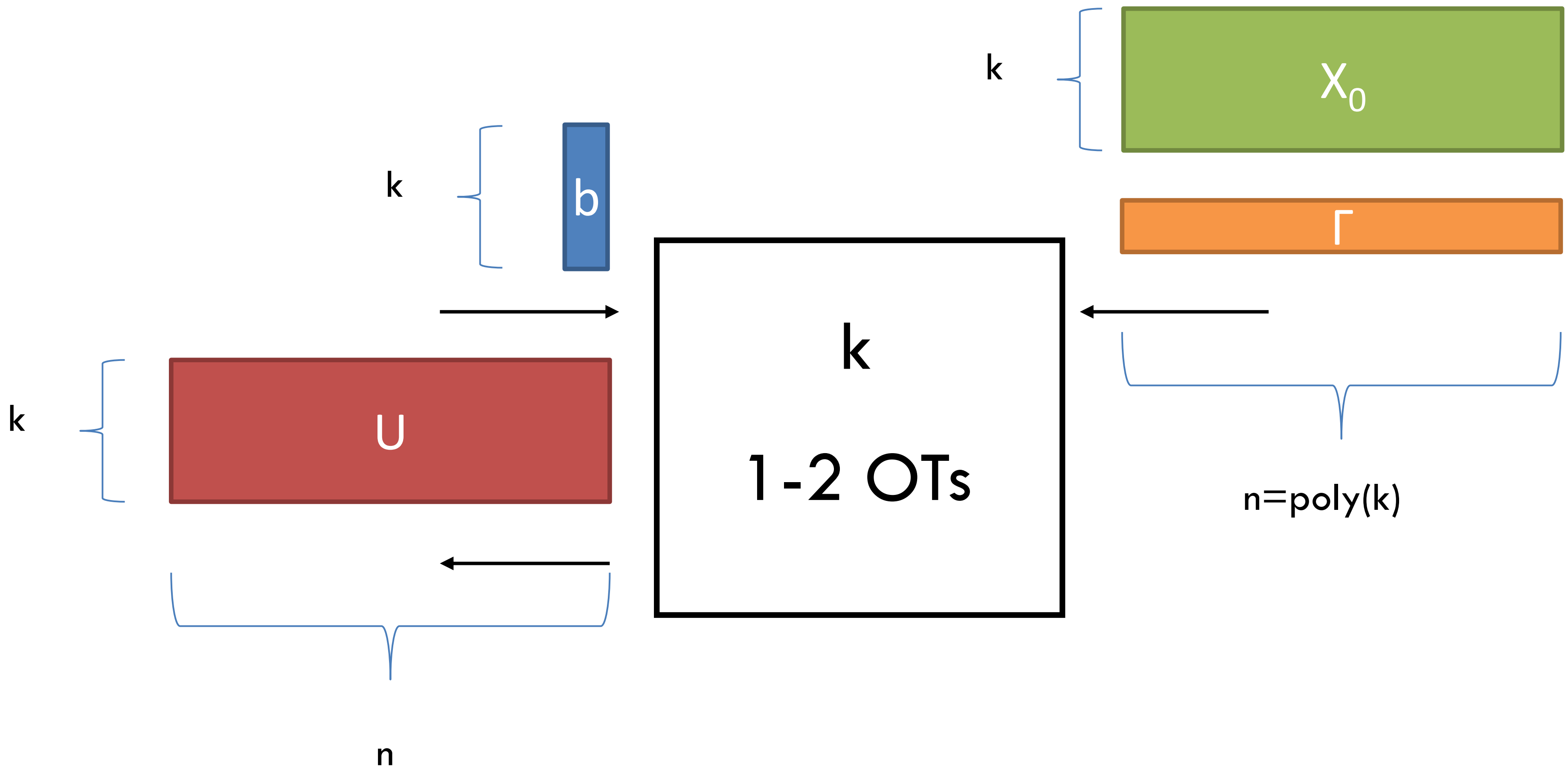
Γ

…

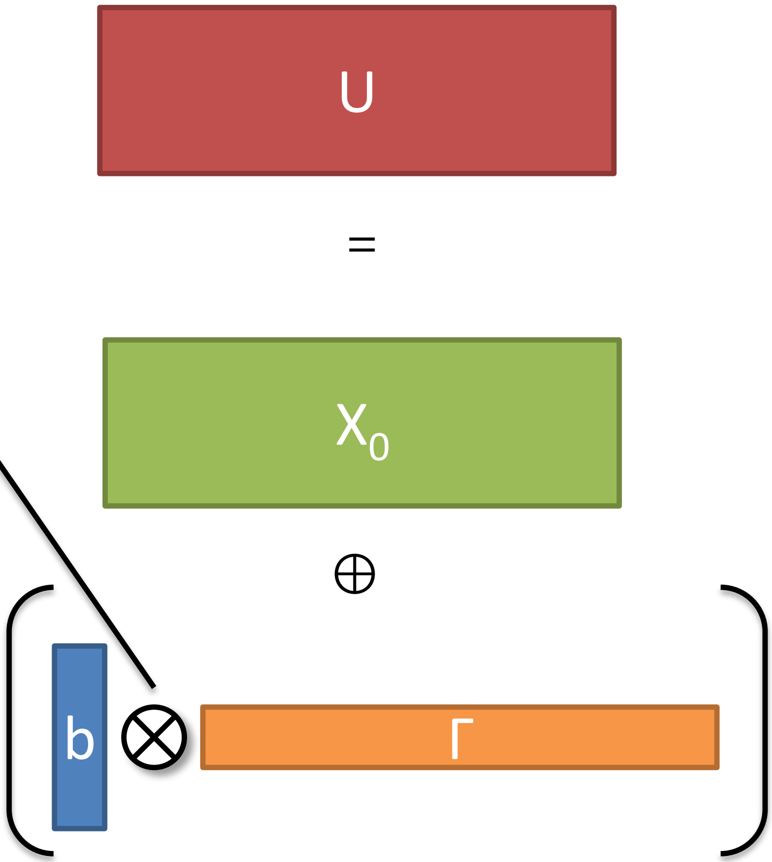
Γ

Problem for active security!

OT Extension, Pictorially

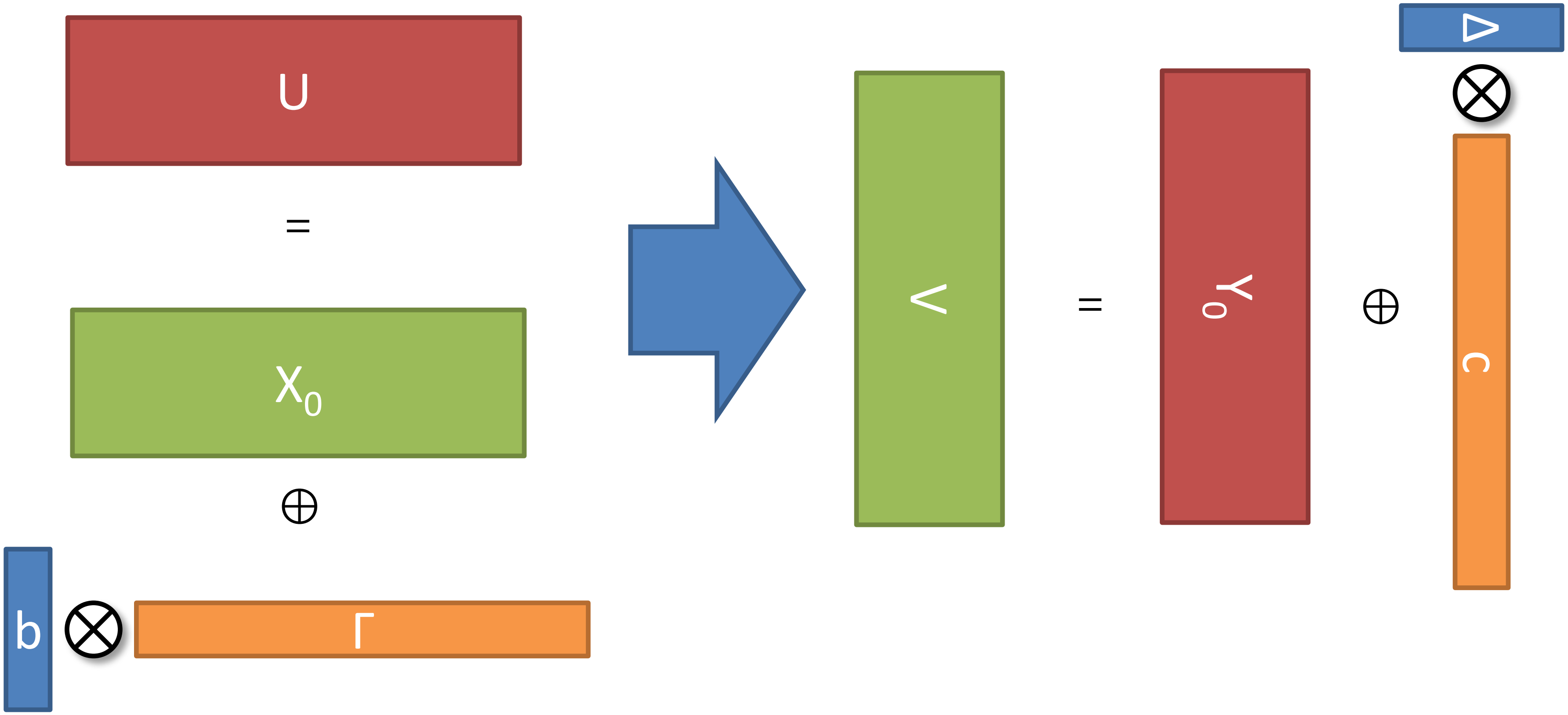


OT Extension, Pictorially

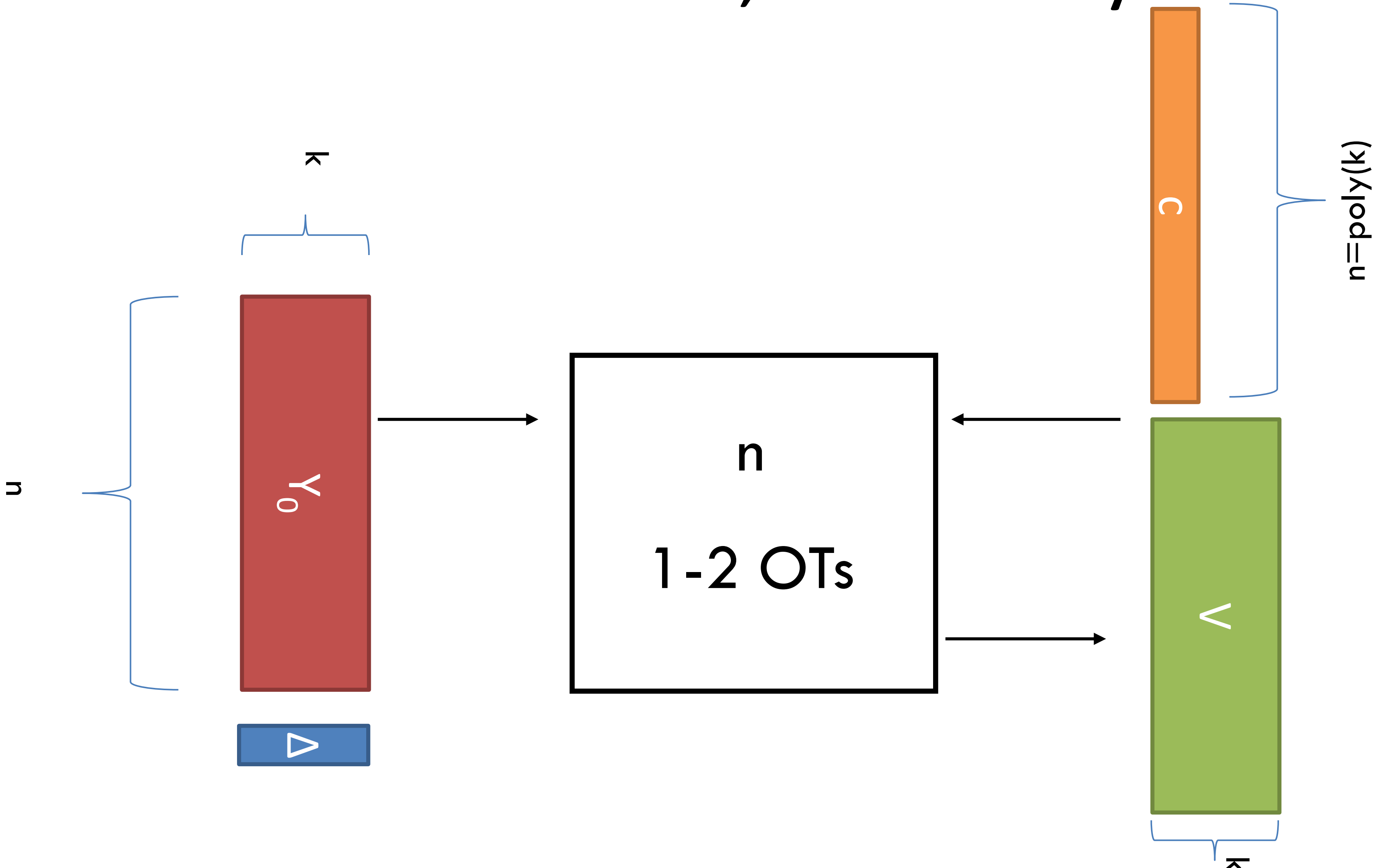


$$(b \otimes \Gamma)_{ij} = b_i \cdot \Gamma_j$$

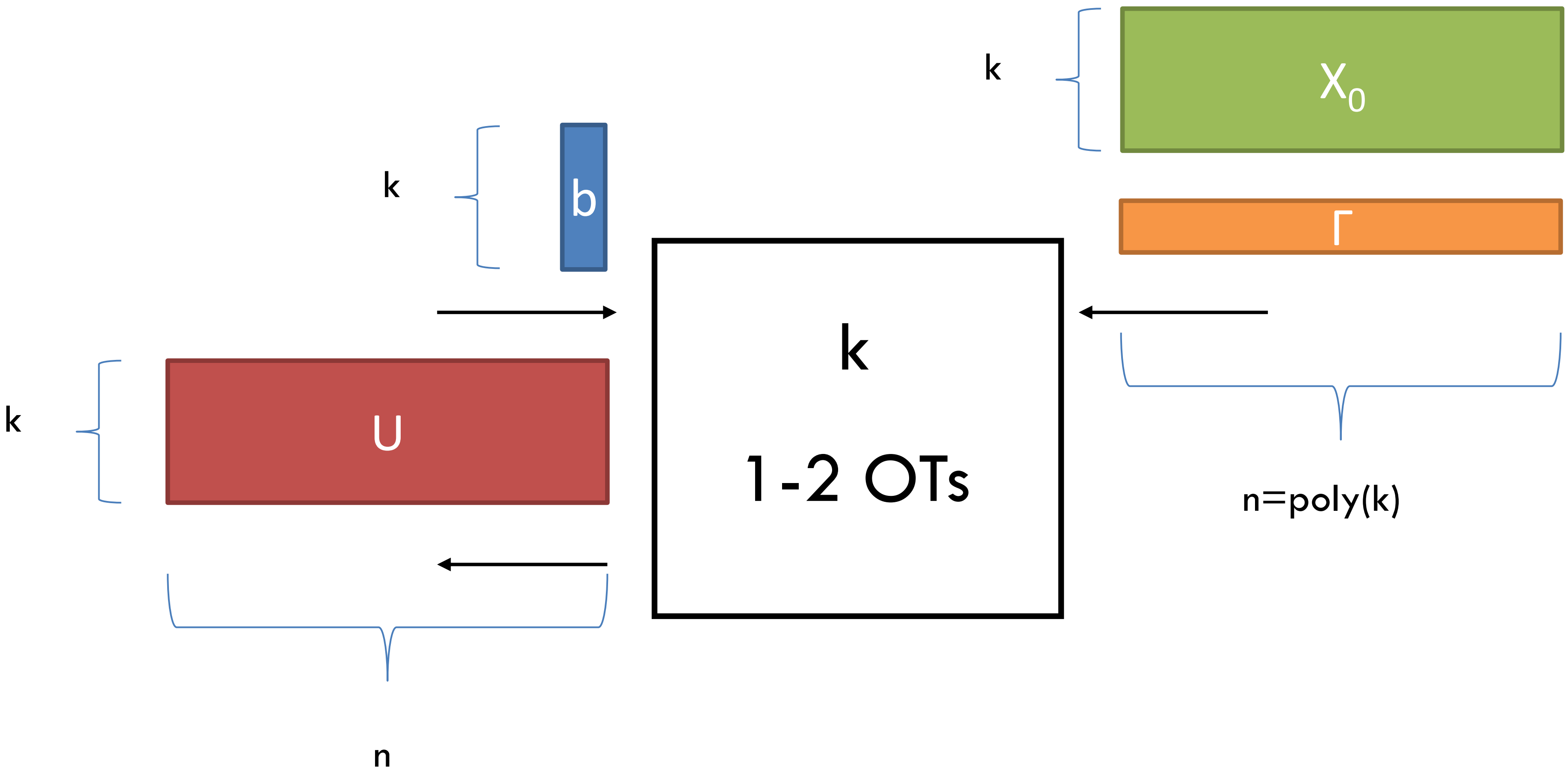
OT Extension, Turn your head!



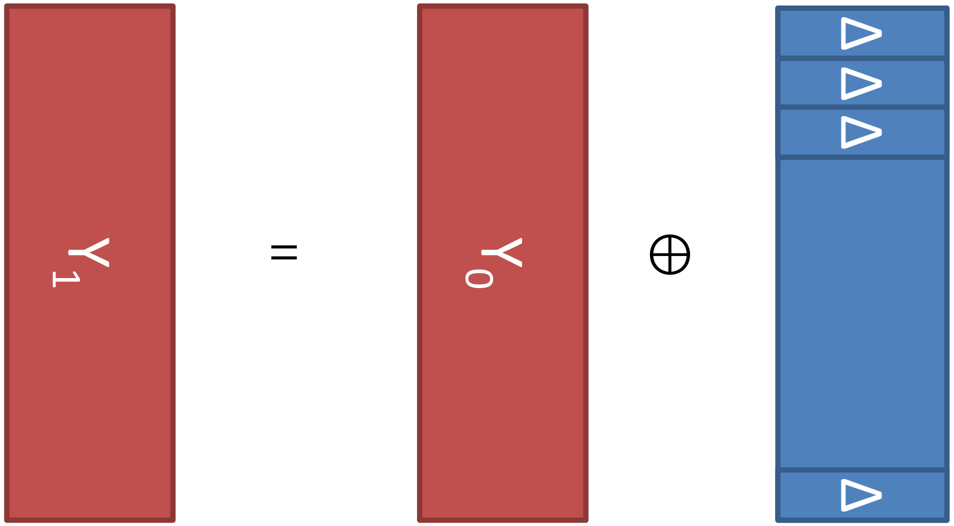
OT Extension, Pictorially



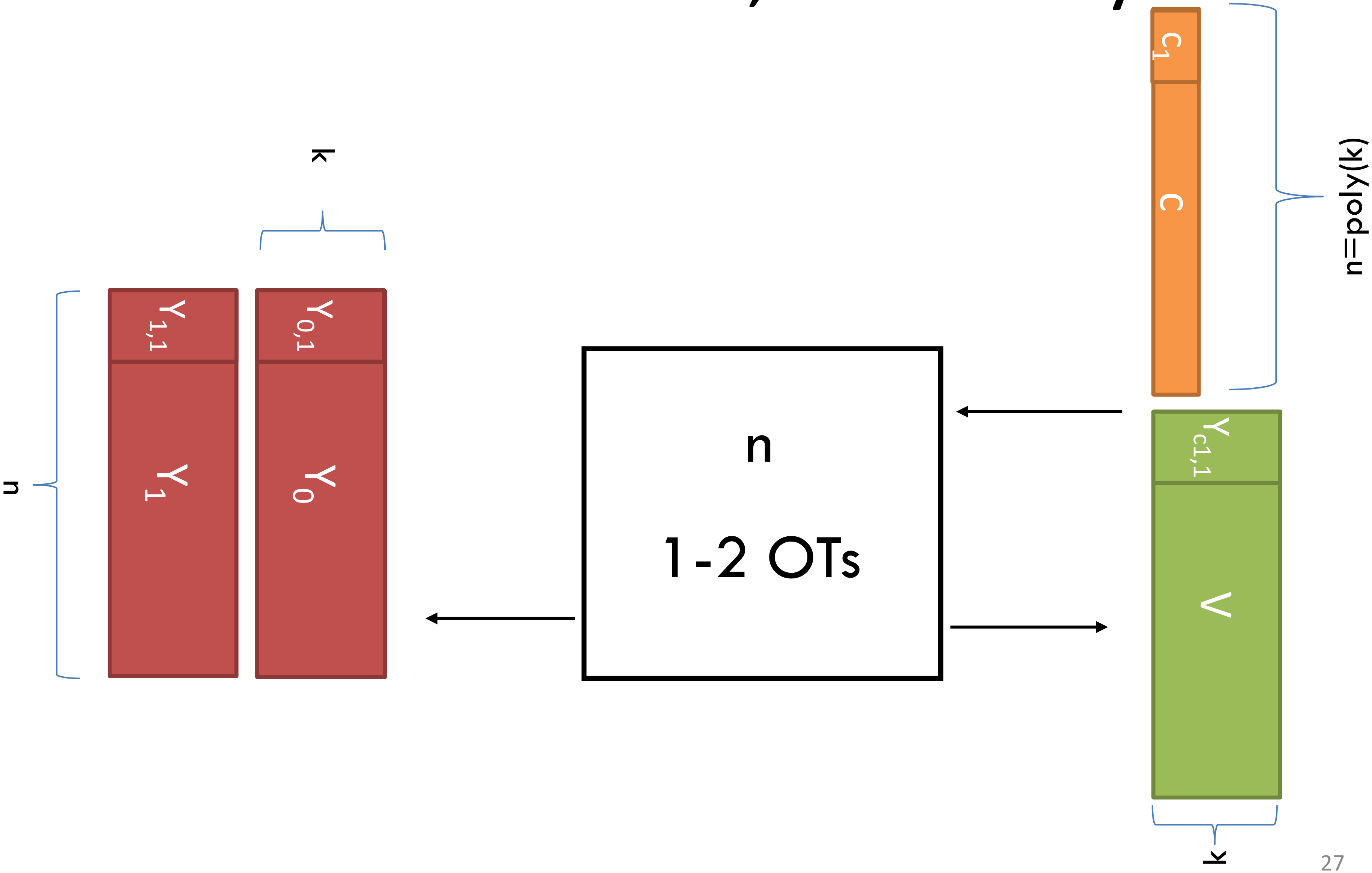
OT Extension, Pictorially



Defining Y_1



OT Extension, Pictorially



Finishing Up

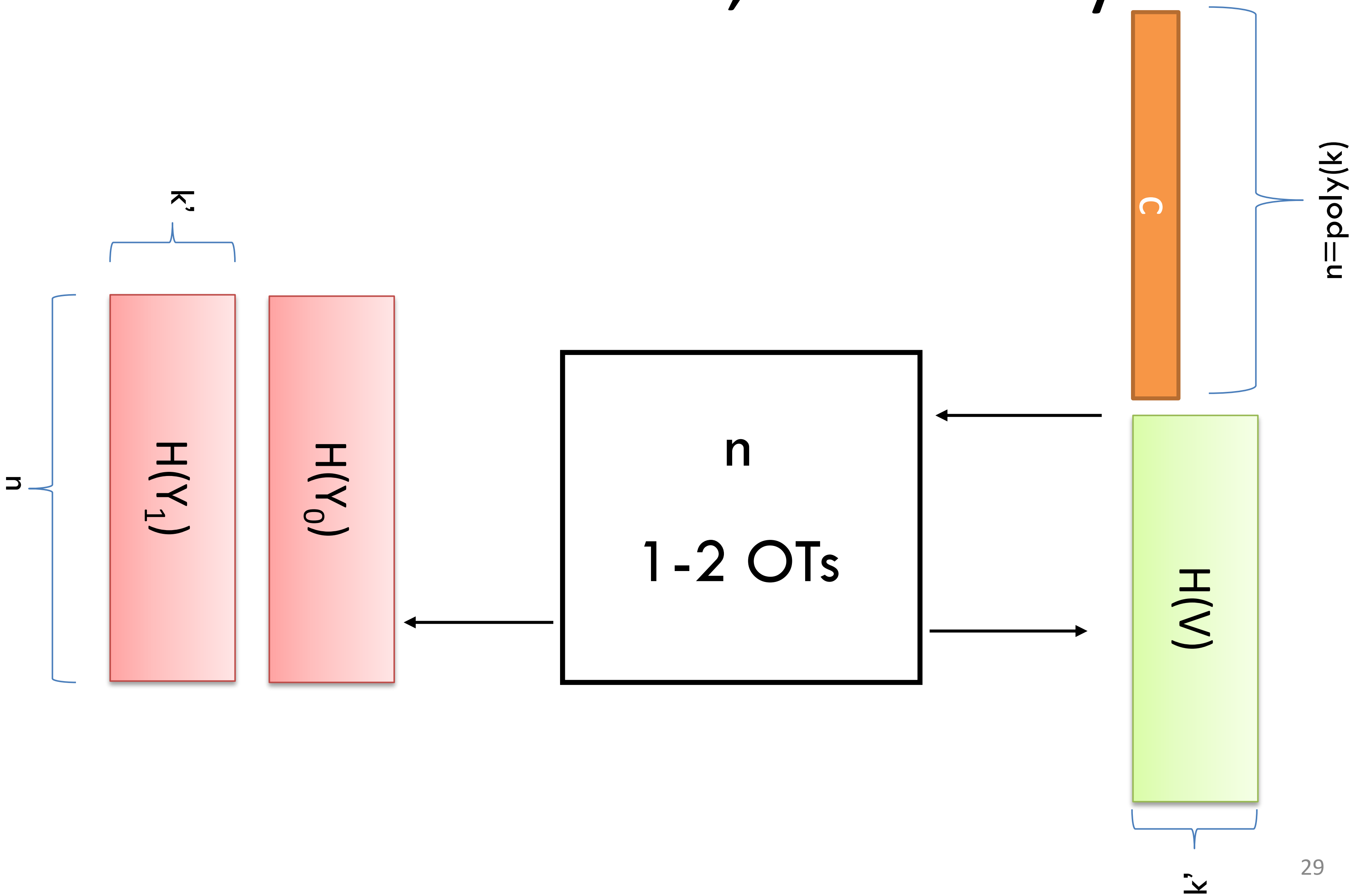
- **Problem:** (Y_0, Y_1) not random!

- **Solution**: just hash each row
  - $Y'_0 = H(Y_0)$
  - $Y'_1 = H(Y_1)$

- Using a **correlation robust hash function** H s.t.
  1. $\{a_0, \ldots, a_n, H(a_0+ \Delta), \ldots, H(a_n+ \Delta)\}$
  2. $\{a_0, \ldots, a_n, b_0, \ldots, b_n\}$ // (a_i's,b_i's random)

  are ***computationally indistinguishable***

OT Extension, Pictorially



Recap

0. Strech **k OTs** from *k- to poly(k)=n-bitlong strings*
1. Set each pair of messages x^i_0, x^i_1 s.t., $\mathbf{x^i_0 \oplus x^i_1 = \Gamma}$
2. **Turn your head** (S/R swap roles)
3. The bits of $\mathbf{c=\Gamma}$ are the new **choice bits**
4. The new messages are of the form $\mathbf{y^j_0, y^j_1=y^j_0 \oplus \Delta}$
5. Break the correlation: $\mathbf{y'^j_0=H(y^j_0), y'^j_1=H(y^j_1)}$

- **Not secure against active adversaries**

Part 2: Active Secure OT Extension

- Warmup: OT properties
- Recap: Passive Secure OT Extension
- **Active Secure OT Extension**

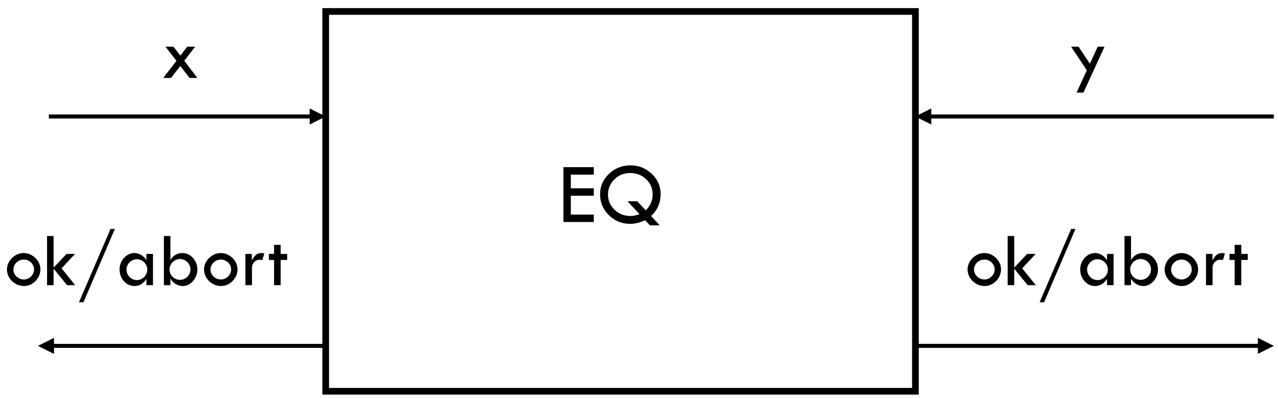
Active Security

1. Set each pair of messages x^i_0, x^i_1 s.t., $\mathbf{x^i_0 \oplus x^i_1 = \Gamma}$

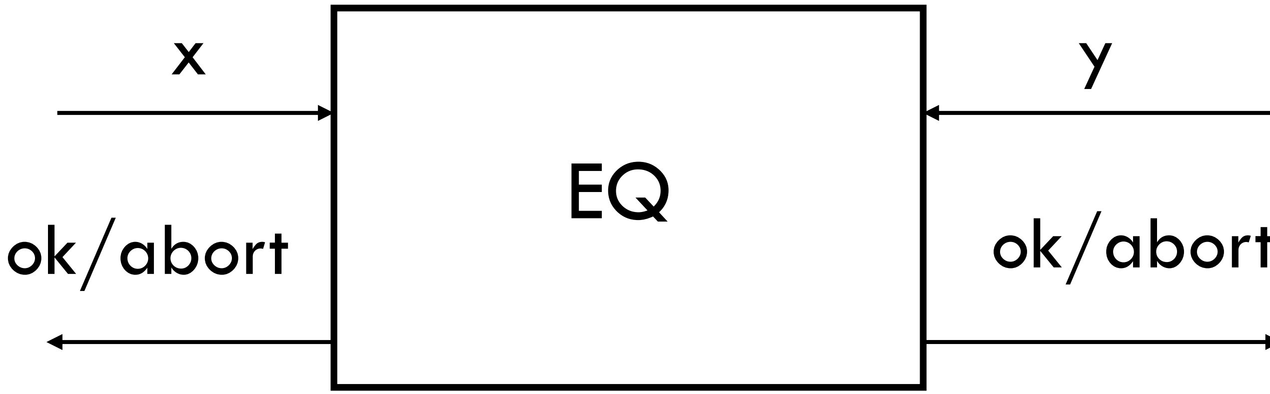
- **How to force Bob to use same value?**
- **“Cut-and-choose”**
  - Start with ≈2k OTs
  - Pair them at random *(destroys half)*
  - Check if the same **Γ** was used
  - **abort** otherwise

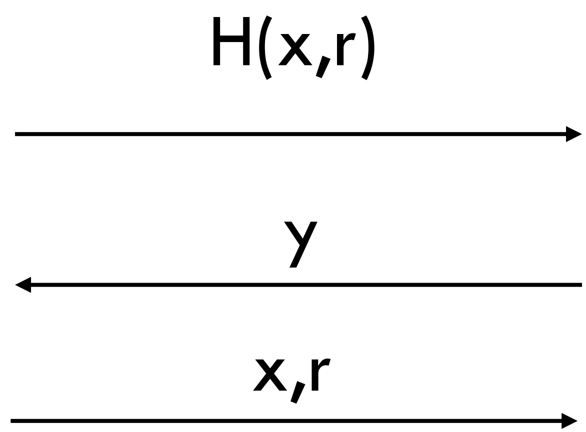
The Equality BOX

- **Output ok if equal**
- **abort/reveal all if different**

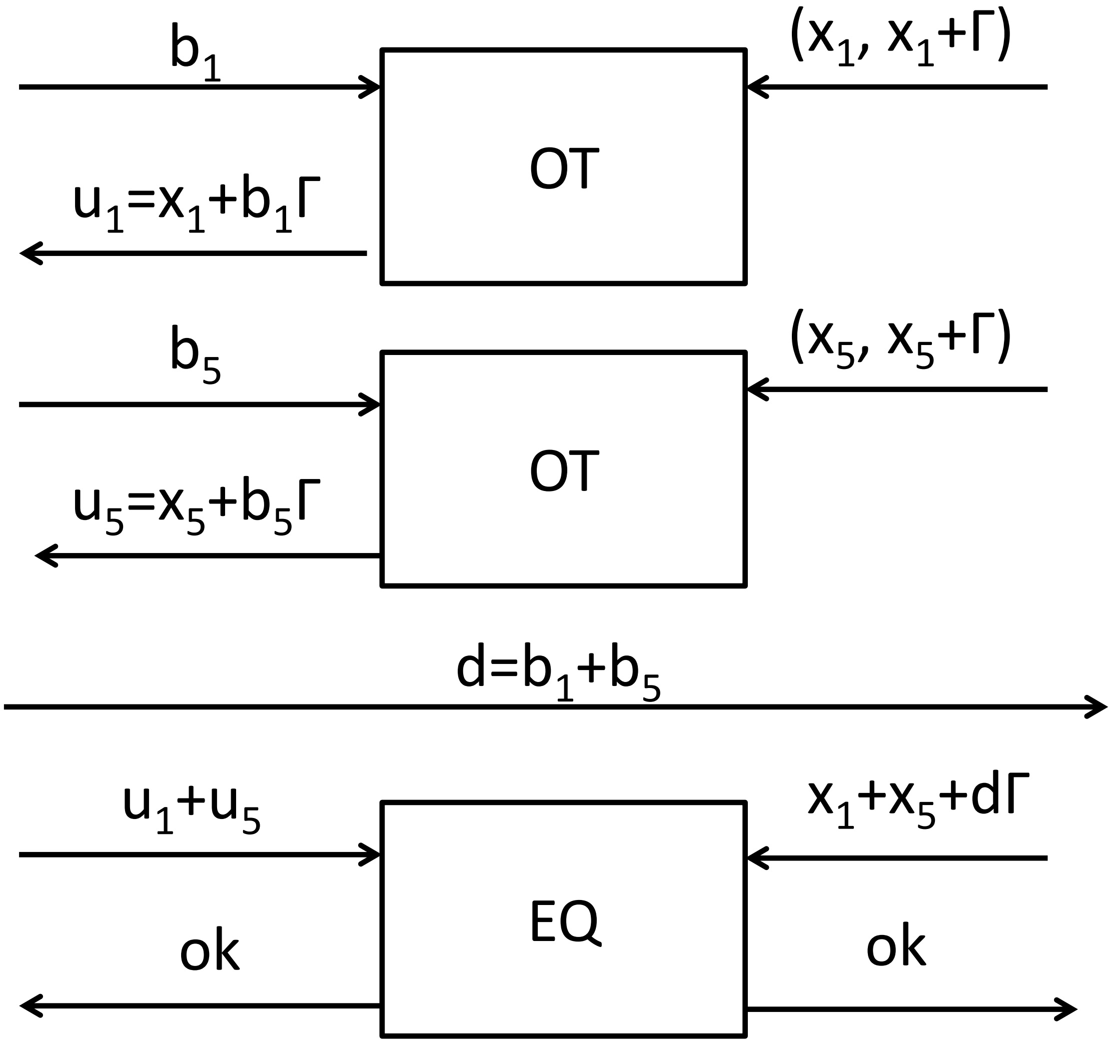


The Equality BOX





Pair and check

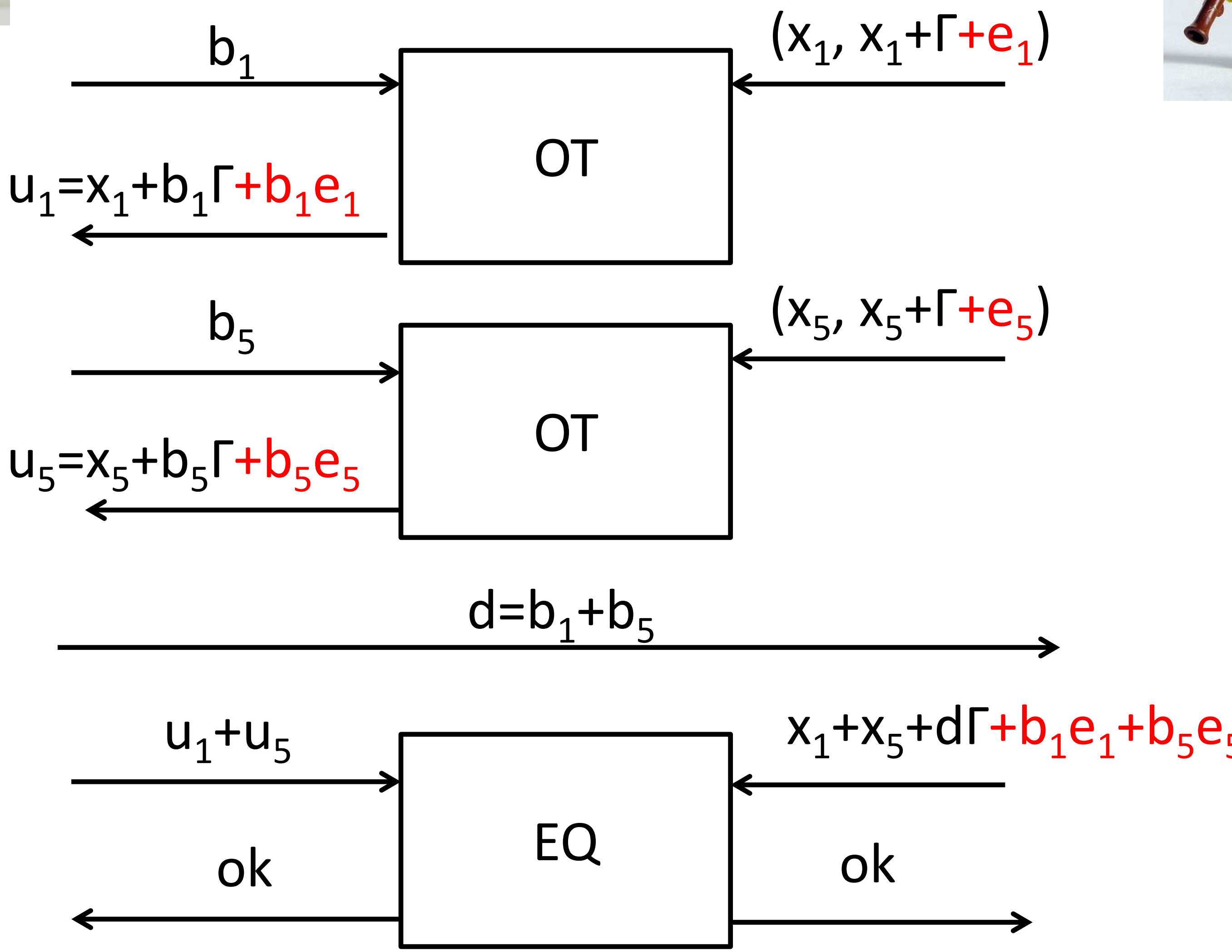


Analysis

- **Ok if both honest**
  - $u_i = x_i + b_i \Gamma_{\mathrm{i}}$
  - $u_i + u_i = x_i + x_j + \left(b_i + b_j\right) \Gamma$ if $\Gamma_i = \Gamma_j = \Gamma$
  - Throw away OT *j* and keep *i* for later use

- **Why use EQ?**
  - Alice needs to prove d is correct too!
  - **Else**: corrputed Alice sends $\mathrm{d} = 1 + b_i + b_j$…
  - …learns two MACs with same key
  - …learns Γ
  - …protocol brekas down completely

Corrupted Bob

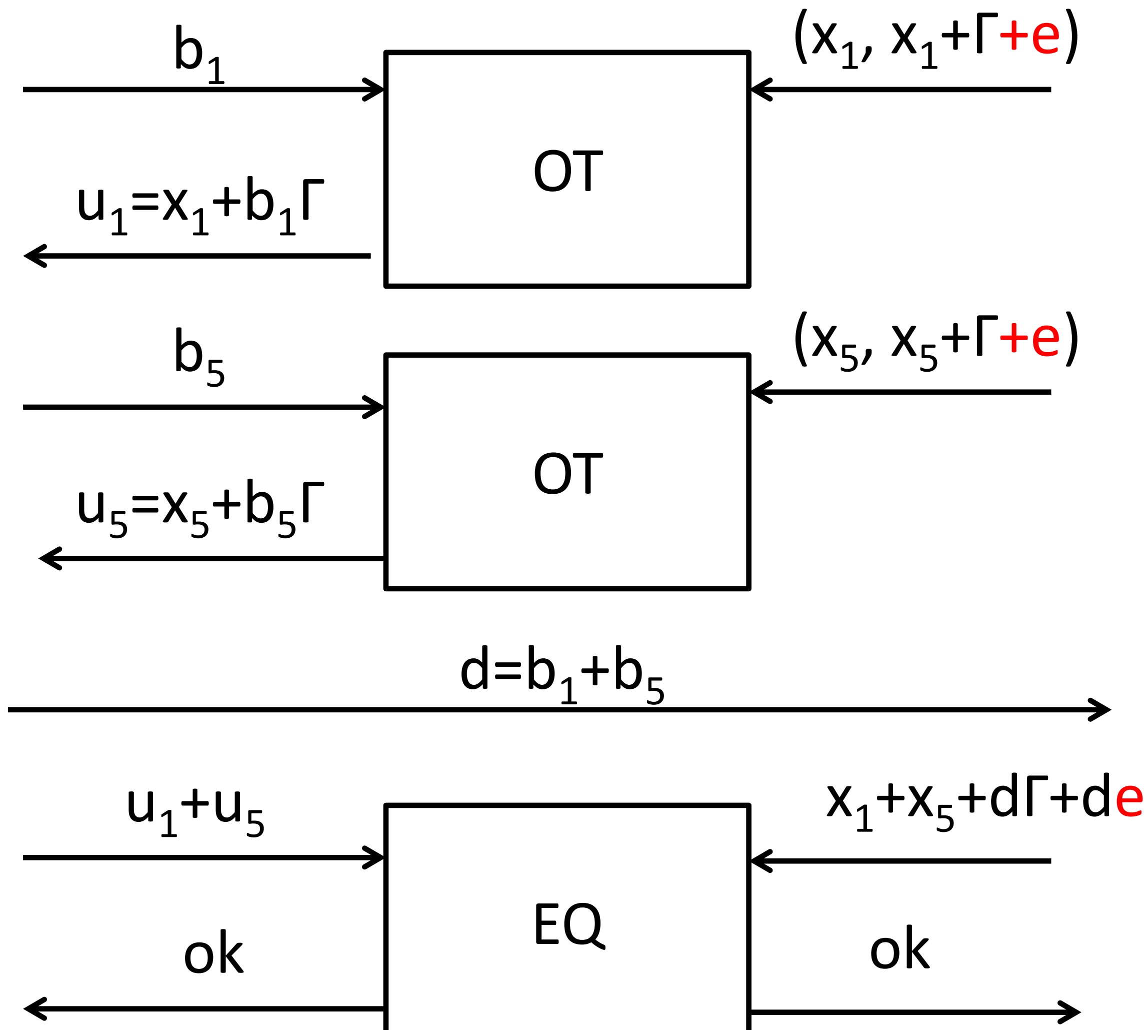


Three cases

- **No error:** $e_i = e_j = 0$
  - Bob always pass the check and learns nothing ☺
- **One error:** $e_i \neq 0, e_j = 0$
  - Bob pass the test if guess b_i correctly
  - 50% abort, 50% Bob learns b_i 😐
- **Canceling errors**: $e_i = e_j \neq 0$
  - Bob always pass the test
  - Can be simulated by leaking bit b_i ☹

For simplicity
$\forall\, i \;\; e_i \in \{0, e^*\}$

Simulating ☹



b_1
$(x_1, x_1+\Gamma+e)$
OT
$u_1=x_1+b_1\Gamma$
b_5
$(x_5, x_5+\Gamma+e)$
OT
$u_5=x_5+b_5\Gamma$
$d=b_1+b_5$
u_1+u_5
$x_1+x_5+d\Gamma+de$
EQ
ok
ok

Simulating ☹

b_1 → OT ← $({\color{red}x'_1}, {\color{red}x'_1}+\Gamma)$

$u_1=x_1+b_1\Gamma$ ←

b_5 → OT ← $({\color{red}x'_5}, {\color{red}x'_5}+\Gamma)$

$u_5=x_5+b_5\Gamma$ ←

Where $x'_i = x_i + b_i e$

$d=b_1+b_5$ →

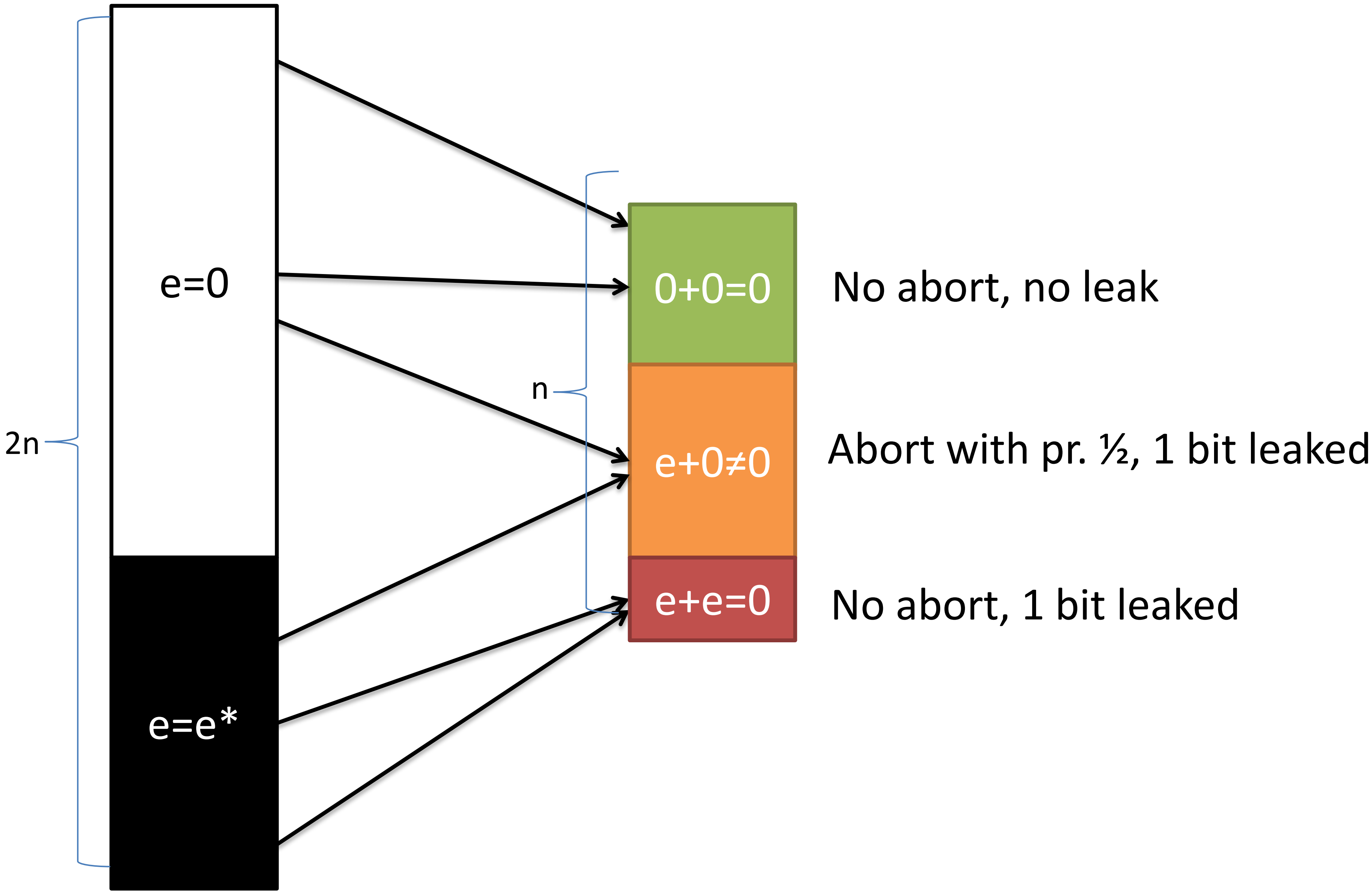
u_1+u_5 → EQ ← ${\color{red}x'_1+x'_5}+d\Gamma$

ok ← EQ → ok

Three cases

- **No error:** $e_i = e_j = 0$
  - Bob always pass the check and learns nothing ☺
- **One error:** $e_i \neq 0, e_j = 0$
  - Bob pass the test if guess b_i correctly
  - 50% abort, 50% Bob learns b_i 😐
- **Canceling errors**: $e_i = e_j \neq 0$
  - Bob always pass the test
  - Can be simulated by leaking bit b_i ☹

For simplicity
$\forall\, i \;\; e_i \in \{0, e^*\}$

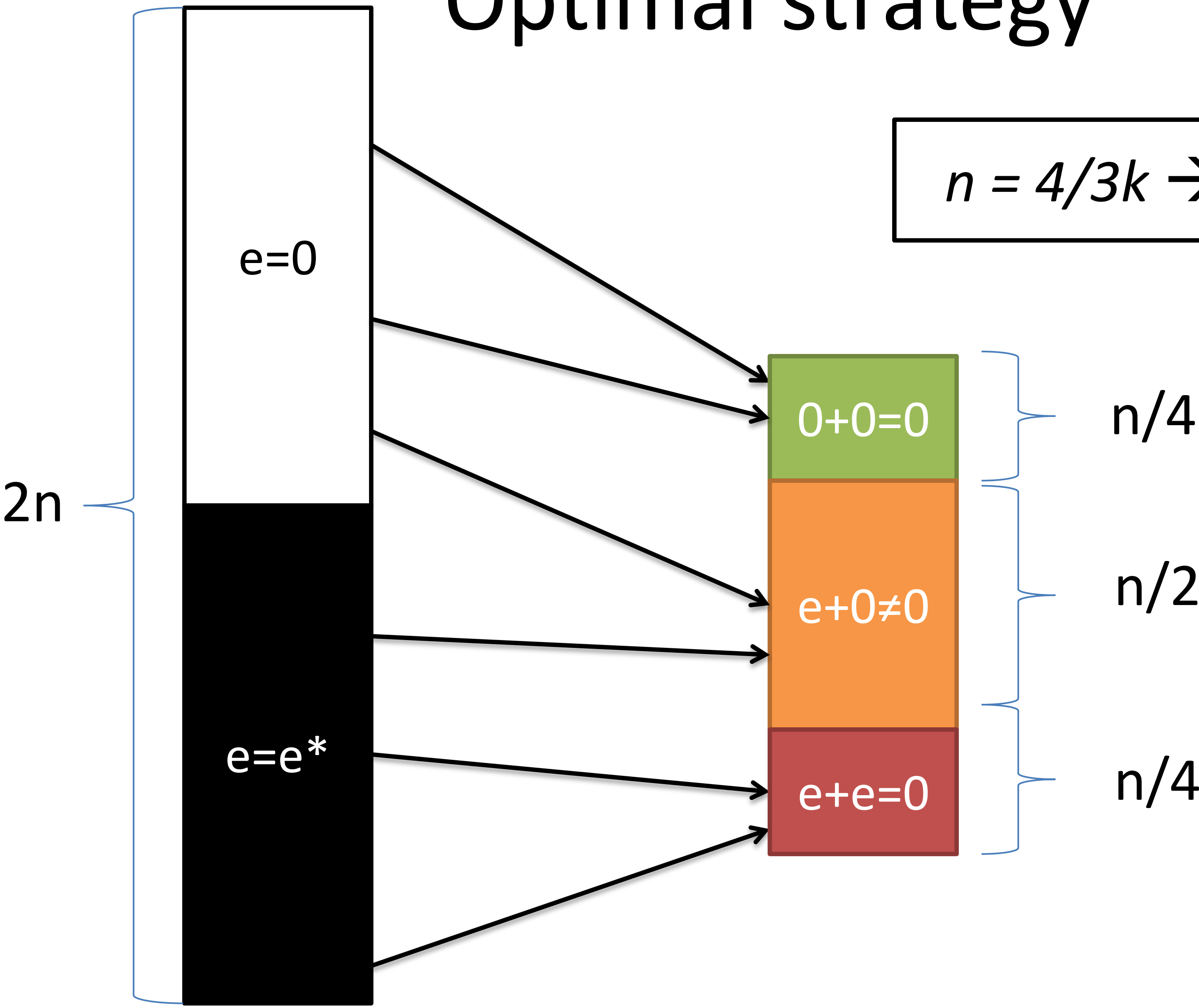

2n
e=0
e=e*
n
0+0=0
e+0≠0
e+e=0
No abort, no leak
Abort with pr. ½, 1 bit leaked
No abort, 1 bit leaked

How many bits does Bob learn?

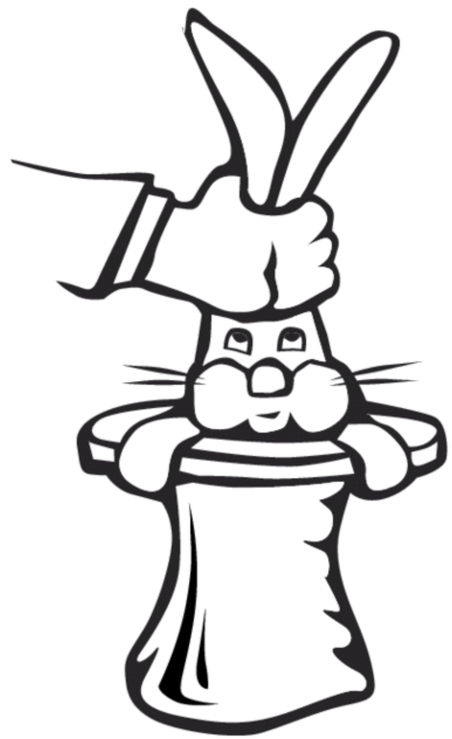
- **Define game:**
  - Choose how many $e \neq 0$. Abort → loses
  - Receive b_i for all i in ***yellow*** and ***red***
  - Guess ***entire vector b.*** Wrong guess → loses
- **Define leak** $L < n + \log($*pr. Bob wins the game*$)$
  - Win = ***not abort*** + ***correct guess***
  - Pr(***not abort***) = $2^{-\#yellow}$
  - Pr(***correct guess***) = $2^{-\#green}$
- $L = n - \#yellow - \#green = \#red$

Optimal strategy

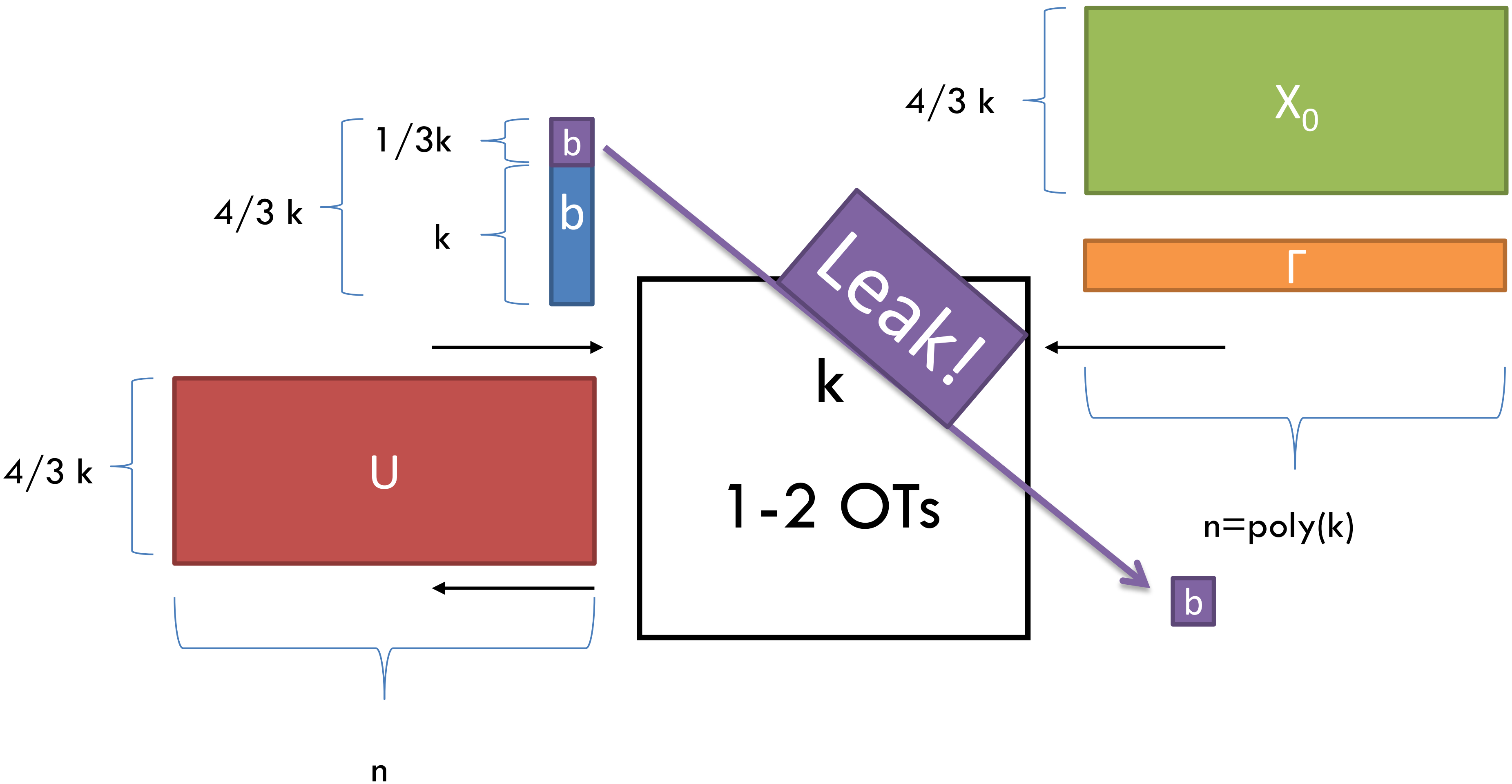
$n = 4/3k \rightarrow L < k/3$



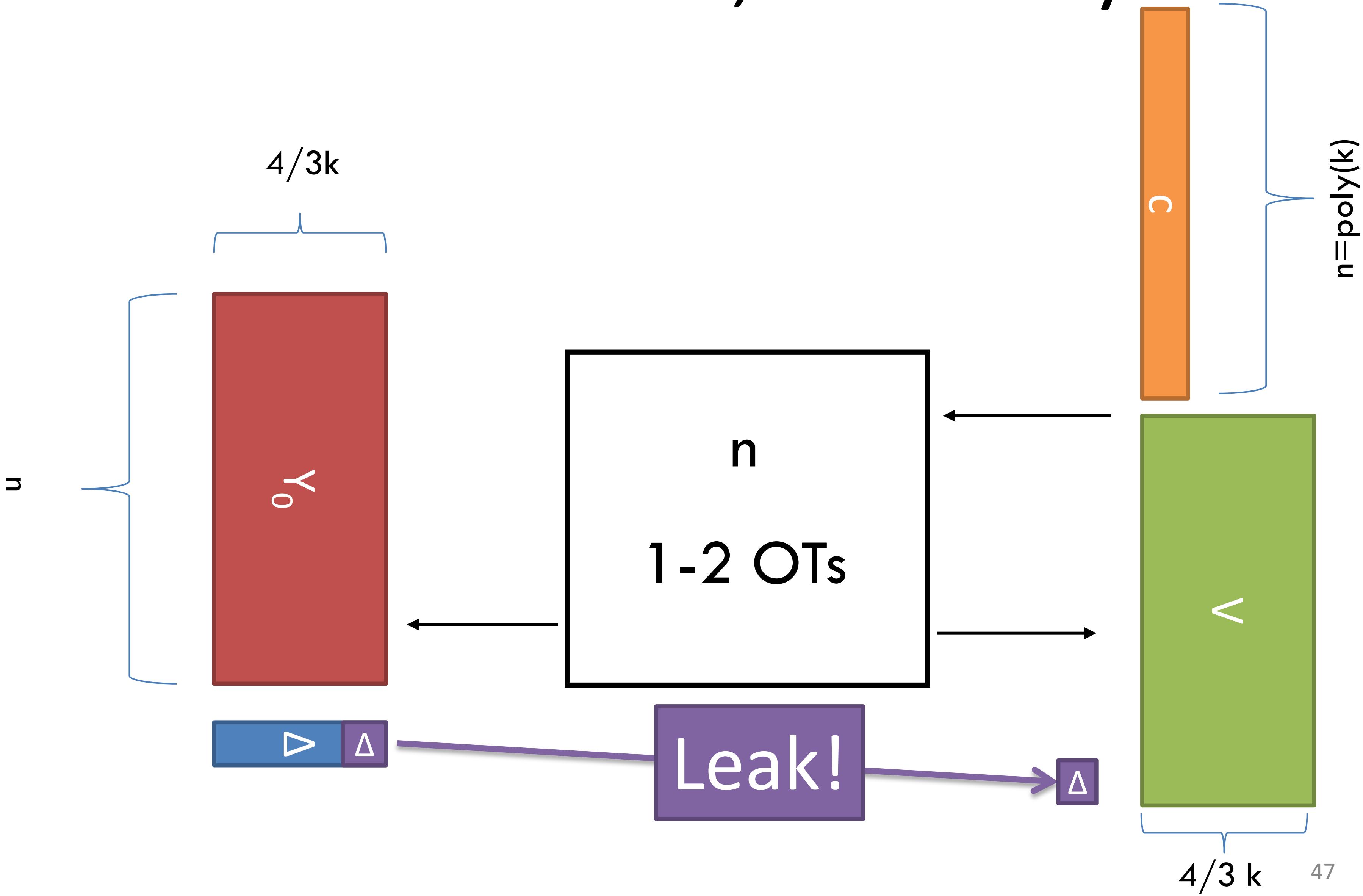
Finishing up…



OT Extension, Pictorially



OT Extension, Pictorially



Solutions

- ***OT Extension:***
  - ***Hash the leak away!***

- ***Authenticated Bits (need linear relation)***
  - ***Universal hash…***
    (multiply with random matrix A)
  - ***…or do nothing!***
    (MAC still secure with ***k*** unknown bits!)

TinyOT authenticated bits

- $[x] = ((x_A, k_A, m_A) , (x_B, k_B, m_B))$ s.t.
  - $m_B = k_A + x_B \Delta_A$ (symmetric for m_A)
  - Δ_A, Δ_B is the same for all wires
    (where the adversary knows at most L bit).
  - MACs, keys are k-bit strings.

Authenticated Bits/OT Extension

1. Run ***(2+2μ)*n OTs** with constant difference ***Γ***
2. ***Cut-and-choose*** and throw away half **OT**s
3. ***Turn your head*** (OT extension)

Authenticated Bits

4. ***Deal with μ-leaked bits with universal hash (or don't).***

OT Extension

4. ***Deal with μ-leaked bits with cryptographic hash.***