CS670: Cryptographic Techniques for Privacy Preservation

Module 2: Secure Memory Access

Adithya Vadapalli

Drawings of Alice, Bob, and other crypto characters by Simon Oya: https://simonoya.com/drawings/

Memory Access

Alice

Bob

Alice has a lot of data.

However, she doesn’t have much space.

Bob has a lot of space, though! (Unfortunately, she does not trust him)

What can she do?

Memory Access on Secret Shares

Alice

$\overrightarrow{D}_0 \quad i^*_0 \quad M_0$

Hold shares of database: $\overrightarrow{D}$

Hold shares of index: i^*

Hold shares of a value: M

Bob

$\overrightarrow{D}_1 \quad i^*_1 \quad M_1$

Their goal: To obtain the shares of $D[\, i^* \,]$

Similarly, do a write operation: To obtain the shares of D'

Such that: $D'[i^*] = D[i^*] + M$

$D'[i] = D[i] \qquad i \neq i^*$

Secure Multiparty Computation

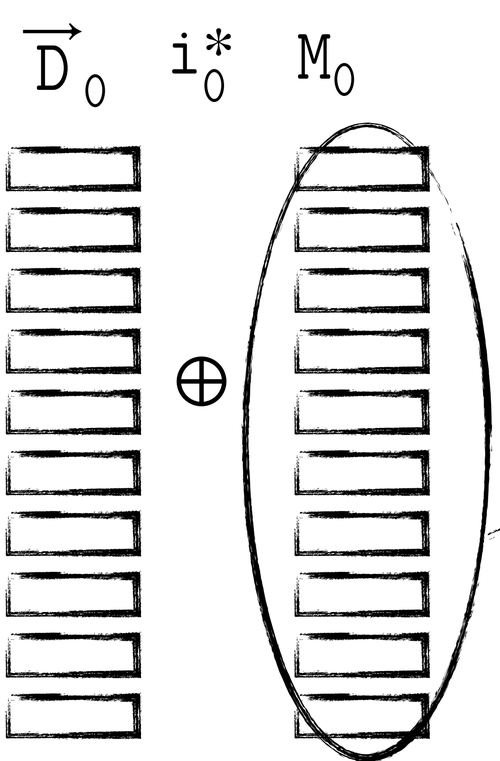
(but on a very specific task!)

Oblivious Write

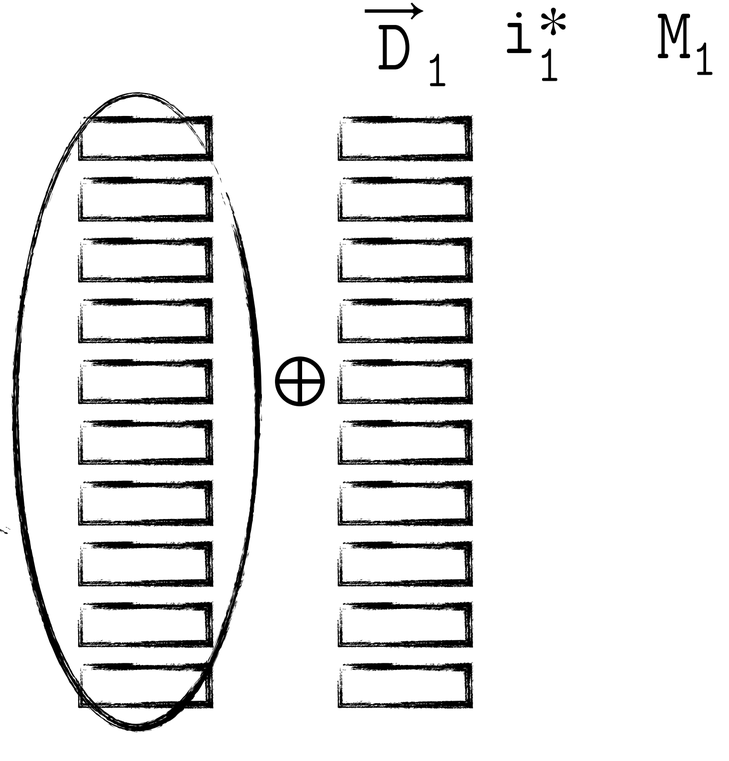
Alice

Bob

Any idea on how to do an oblivious write?



Shares of a standard-basis vector



Oblivious Write

Alice

$\overrightarrow{\mathtt{D}}_0 \quad \mathtt{i}^*_0 \quad \mathtt{M}_0$

So, what does the problem boil down to?

Bob

$\overrightarrow{\mathtt{D}}_1 \quad \mathtt{i}^*_1 \quad \mathtt{M}_1$

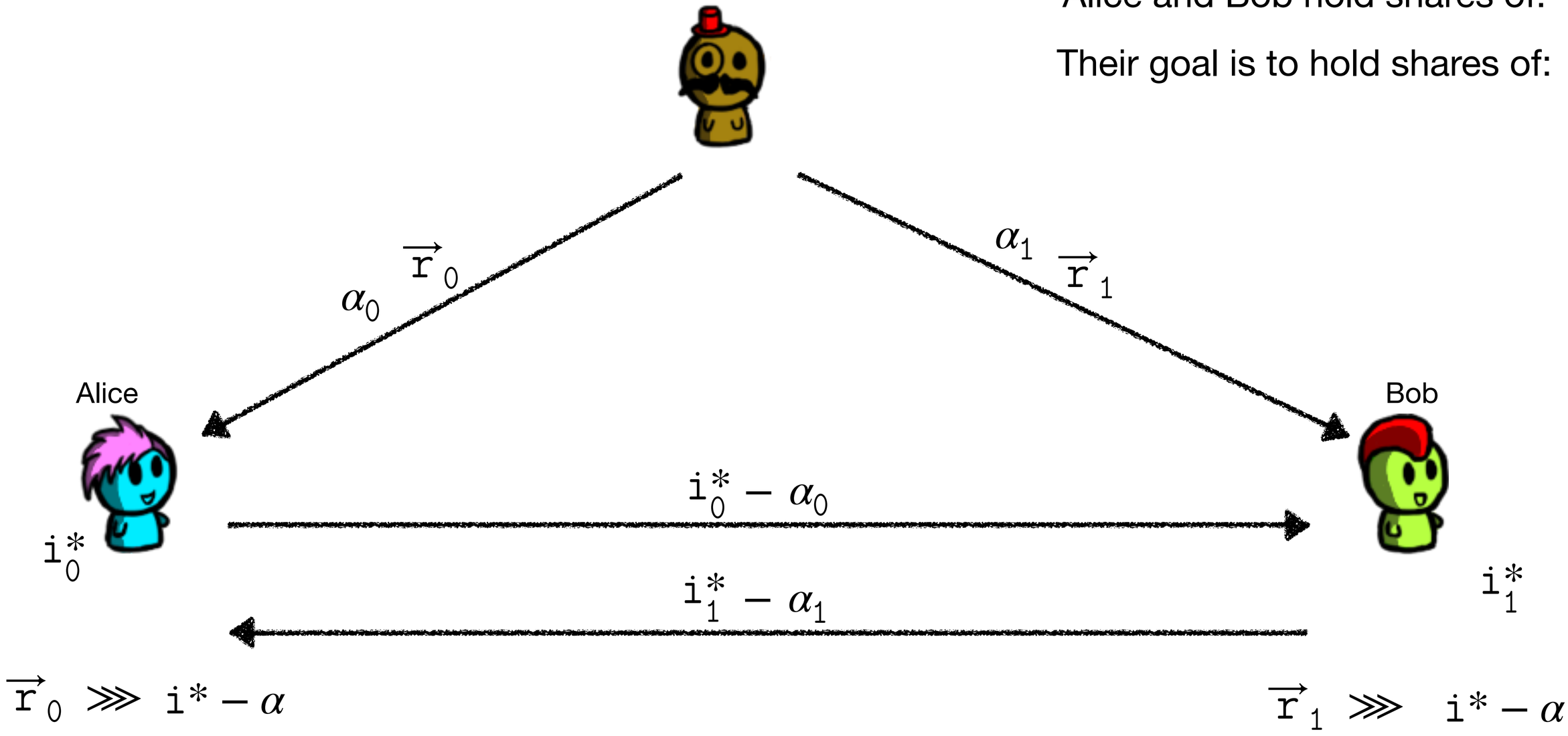
The Rotation Trick

$\vec{r}_0 + \vec{r}_1 = \vec{e}_\alpha$ → *Standard-basis vector at* α

$\alpha_0 + \alpha_1 = \alpha$ → *some random value*

Alice and Bob hold shares of: i^*

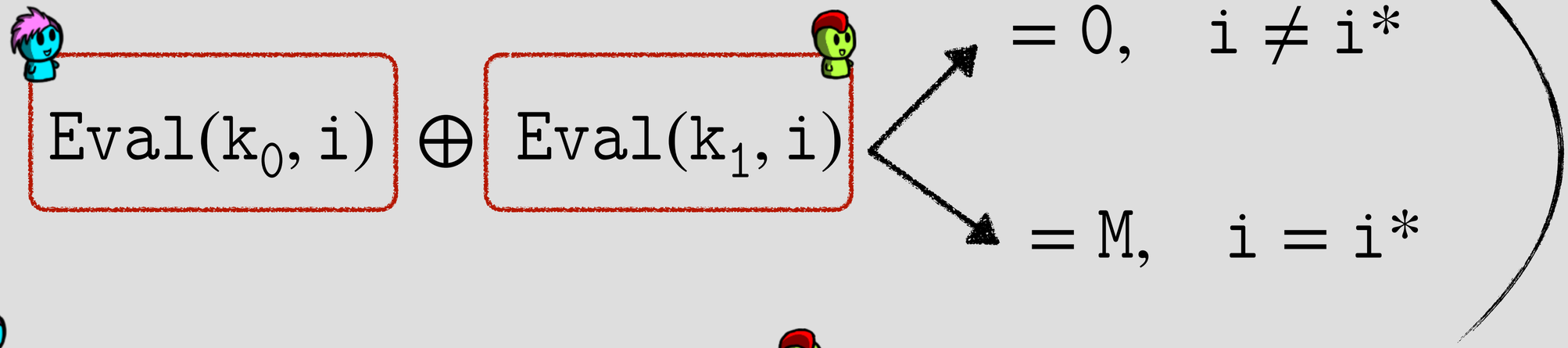
Their goal is to hold shares of: $\vec{e}_{i^*}$



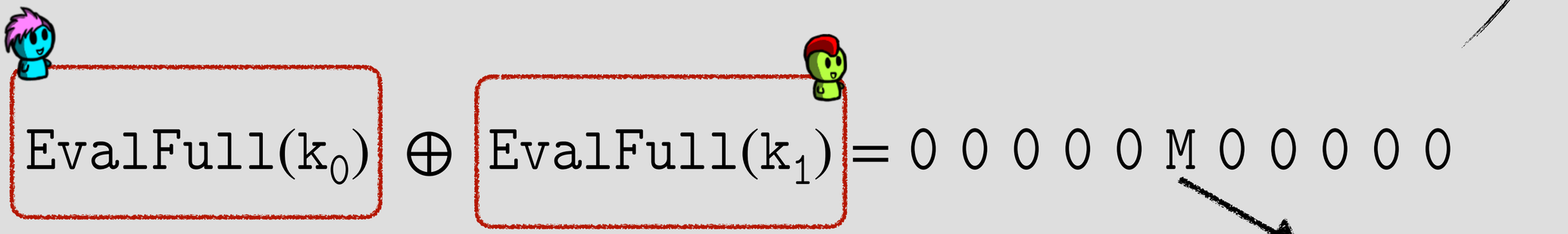
A Detour: Distributed Point Functions

The client (who is trying to distribute the shares)

$$(k_0, k_1) \leftarrow \mathtt{Gen}(i^*, M, 1^\lambda)$$



$$\mathtt{Eval}(k_0, i) \oplus \mathtt{Eval}(k_1, i) = \begin{cases} 0, & i \neq i^* \\ M, & i = i^* \end{cases}$$



$$\mathtt{EvalFull}(k_0) \oplus \mathtt{EvalFull}(k_1) = 0\ 0\ 0\ 0\ 0\ M\ 0\ 0\ 0\ 0\ 0$$

$\mathtt{index} = i^*$

Oblivious Write

Alice

$\overrightarrow{D}_0$ i^*_0 M_0

Bob

$\overrightarrow{D}_1$ i^*_1 M_1

Do you have any ideas on how to solve this problem now?

The parties need to run an MPC on: DPF generation and evaluations

Attendance!

Back to the Detour

$$(\mathtt{k}_0, \mathtt{k}_1) \leftarrow \mathtt{Gen}(\mathtt{i}*, \mathtt{M}, 1^\lambda)$$

Some ideas on the Gen Function?

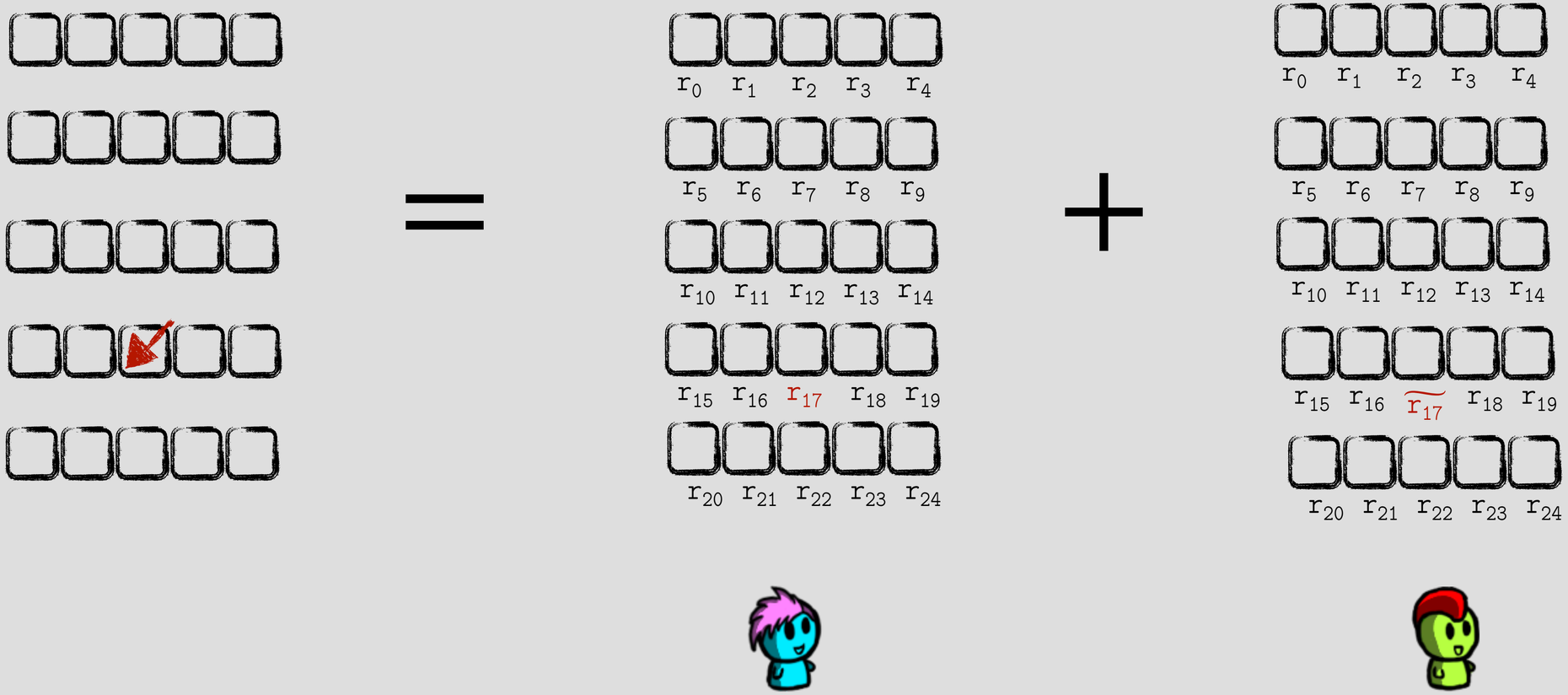
Any naive idea?

Can we get better than a naive

=

+

Can we get better than a naive



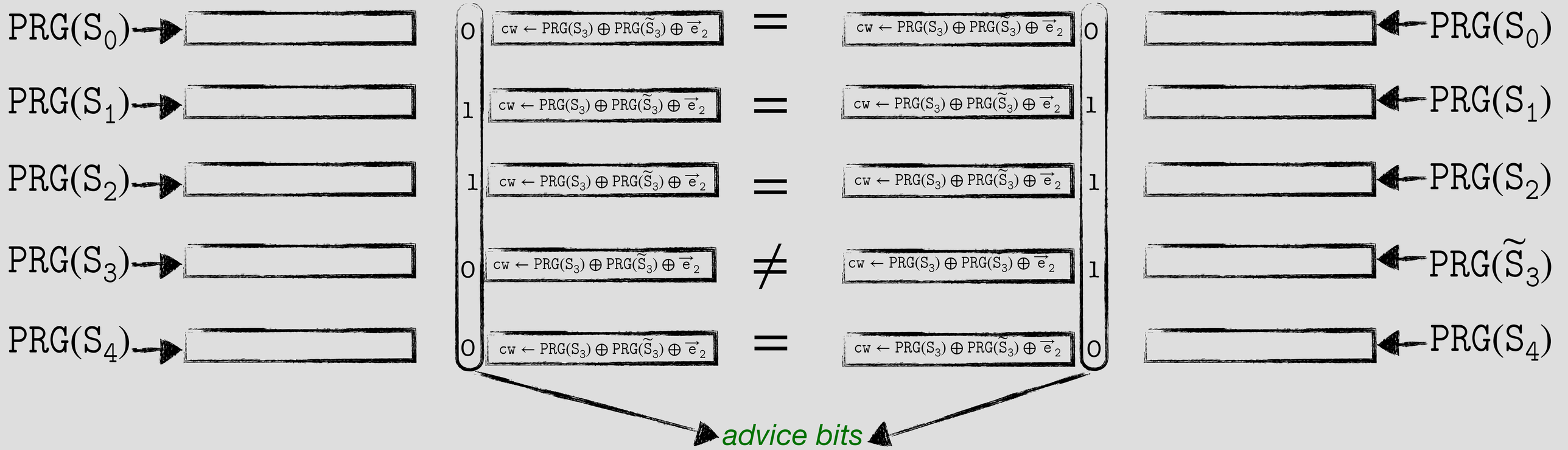
Back to the Detour

$$(k_0, k_1) \leftarrow \mathtt{Gen}(i^*, M, 1^\lambda)$$

This *almost* gives us what we want!

Why almost?

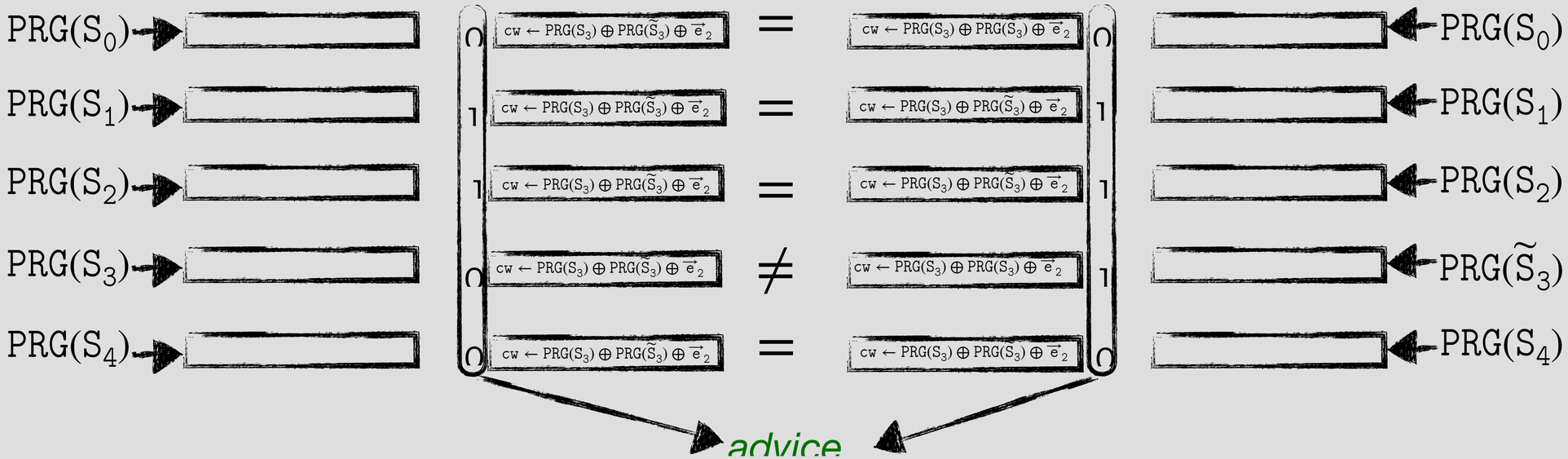
How can we fix it?



$$\mathtt{cw} \leftarrow \mathrm{PRG}(S_3) \oplus \mathrm{PRG}(\widetilde{S}_3) \oplus \vec{e}_2$$

Back to the Detour

$(k_0, k_1) \leftarrow \mathrm{Gen}(i^*, M, 1^\lambda)$



$\mathrm{cw} \leftarrow \mathrm{PRG}(S_3) \oplus \mathrm{PRG}(\widetilde{S}_3) \oplus \vec{e}_2$

$k_0 = (s_0, s_1, s_2, s_3, s_4, \mathrm{cw}, b_0, b_1, b_2, b_3, b4)$

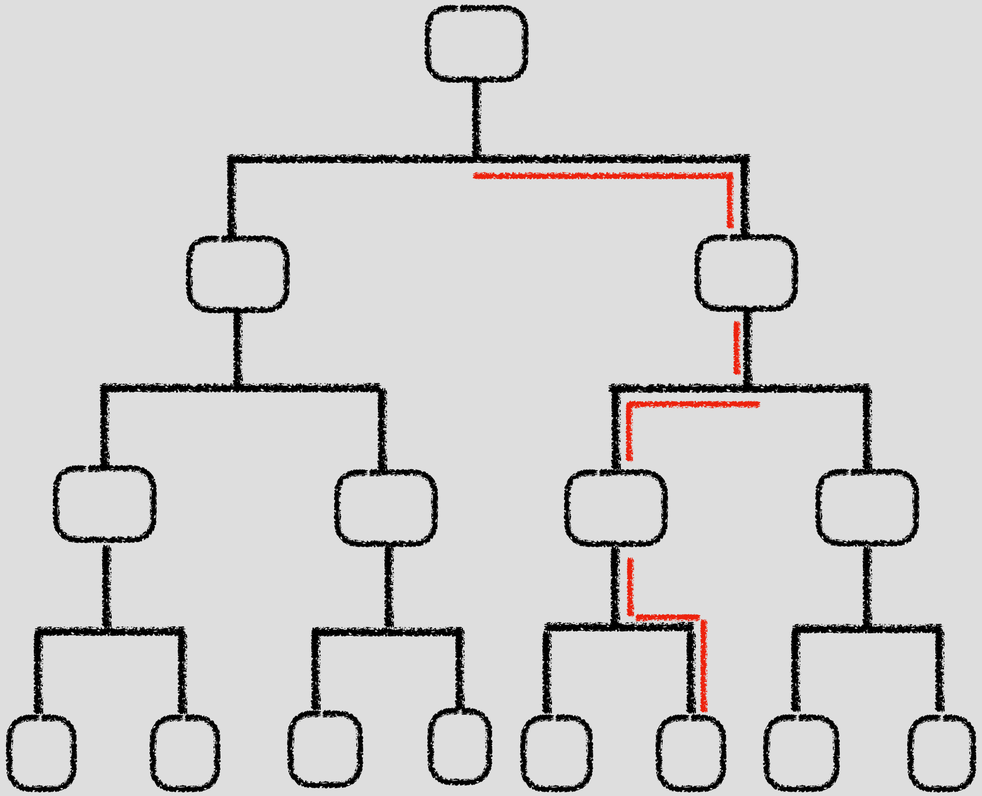
$k_1 = (s_0, s_1, s_2, \widetilde{s_3}, s_4, \mathrm{cw}, b_0, b_1, b_2, \widetilde{b_3}, b4)$

advice bits

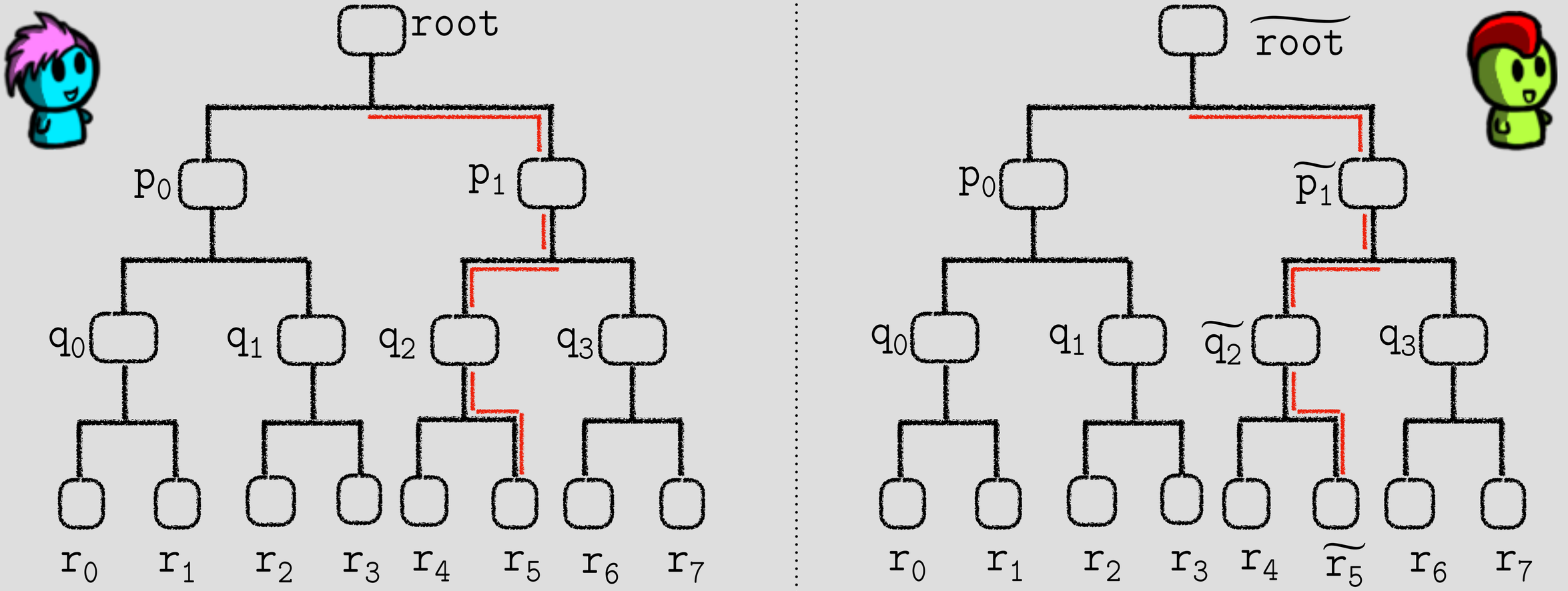
Point Function as binary tree

`101` = target point

A function that evaluates to 0 everywhere excpet at a special point.



Distributing a Point Function among two parties



101 = target point

The more efficient DPF

101 = target point

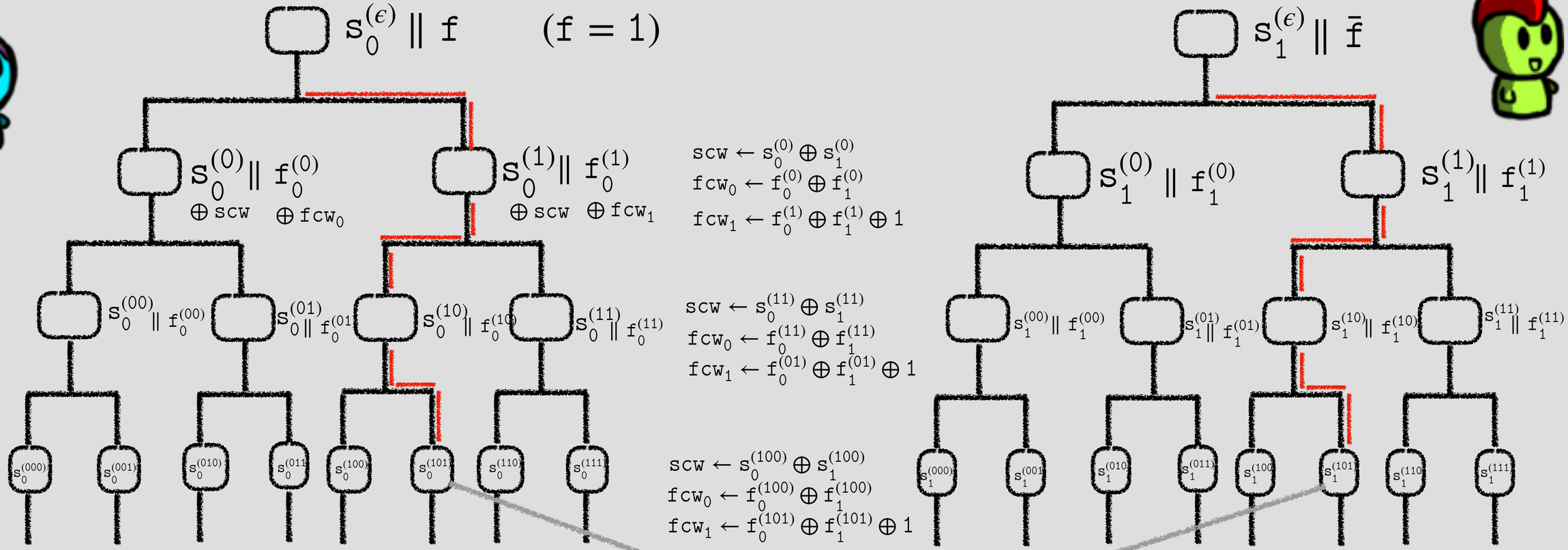
$scw \leftarrow s_0^{(0)} \oplus s_1^{(0)}$
$fcw_0 \leftarrow f_0^{(0)} \oplus f_1^{(0)}$
$fcw_1 \leftarrow f_0^{(1)} \oplus f_1^{(1)} \oplus 1$

$scw \leftarrow s_0^{(11)} \oplus s_1^{(11)}$
$fcw_0 \leftarrow f_0^{(11)} \oplus f_1^{(11)}$
$fcw_1 \leftarrow f_0^{(01)} \oplus f_1^{(01)} \oplus 1$

$scw \leftarrow s_0^{(100)} \oplus s_1^{(100)}$
$fcw_0 \leftarrow f_0^{(100)} \oplus f_1^{(100)}$
$fcw_1 \leftarrow f_0^{(101)} \oplus f_1^{(101)} \oplus 1$

The Final Correction Word

101 = target point



\+

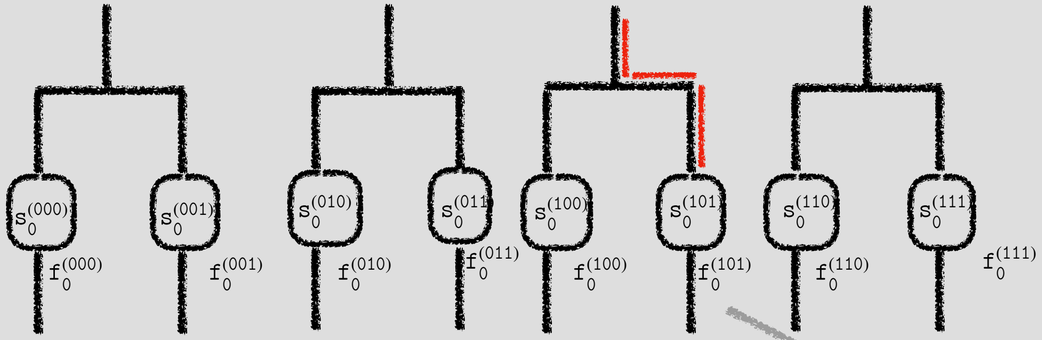
something random

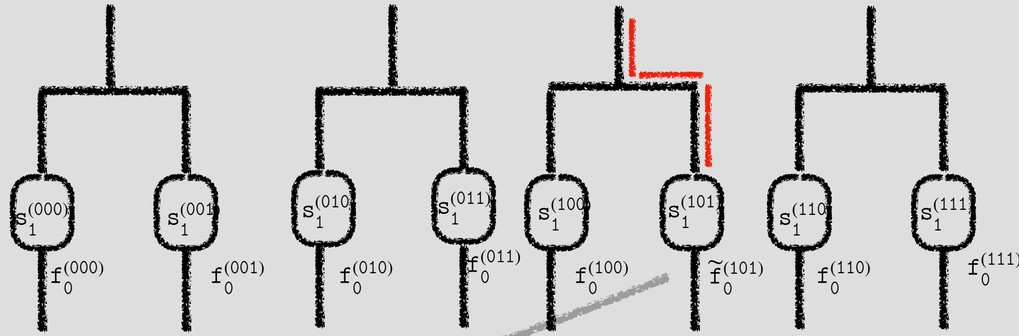
But, we do not want something random!

We want: M

The Final Correction Word

101 = target point





something random

+

But, we do not want something random! We want: M

Will this help?

The leaf layers also have advice bits!

$$\mathrm{FCW} \leftarrow s_0^{(101)} \oplus s_1^{(101)} \oplus M$$

Oblivious Write

Alice

$\overrightarrow{D}_0$ i^*_0 M_0

Bob

$\overrightarrow{D}_1$ i^*_1 M_1

So, now what is the problem we need to sove.

The parties need to run an MPC on: DPF generation and evaluations

 &  hold shares of i^*

Alice Bob

Quiz 1 Statistics

Mean: 6.23 Median: 6.0 Maximum: 17.5

Review Requests Open Until the End of Sunday (only on gradescope)

MPC DPF Generation


Alice

&


Bob

hold shares of `i*`

In other words,


Alice

&

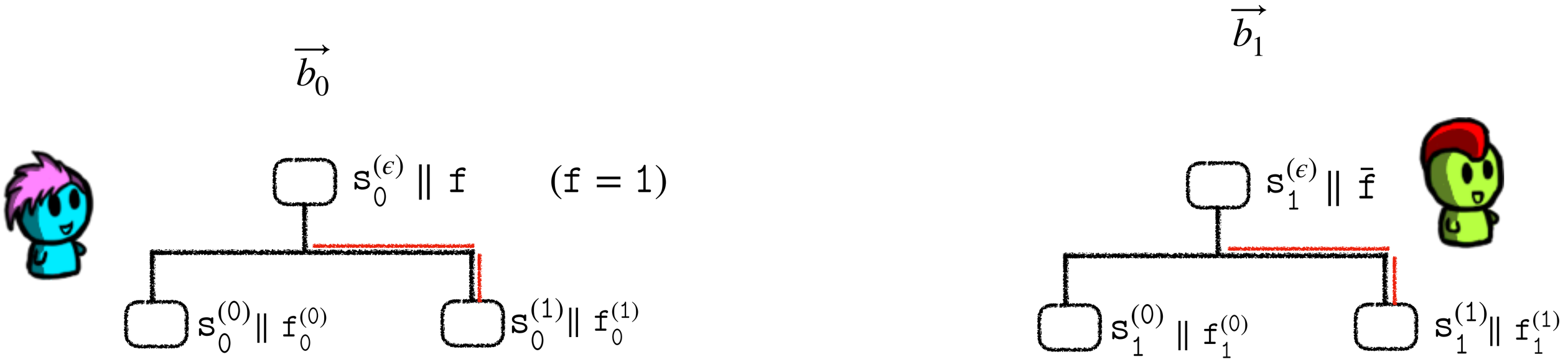

Bob

hold $\overrightarrow{b_0}$ *and* $\overrightarrow{b_1}$

$\vec{b} \leftarrow \vec{b}_0 \oplus \vec{b}_1$

What does the binary representation of `i*` mean?

binary representation of: `i*`



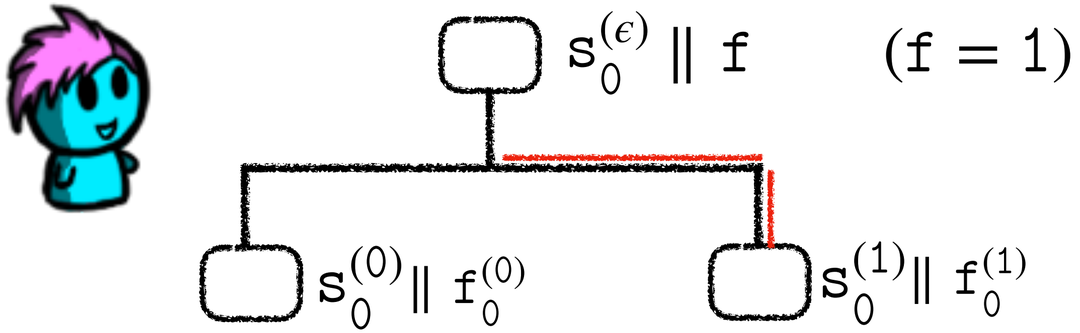
What does $\vec{b}_0[0] \oplus \vec{b}_1[0]$ represent?

Whether we go left or right to get to the target index

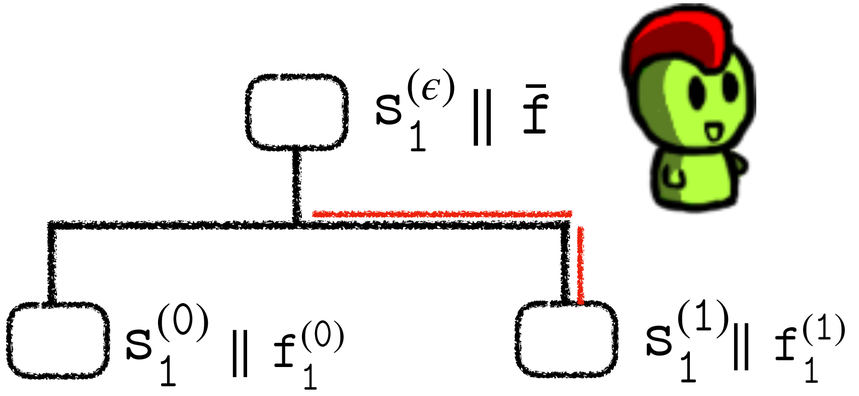
OK, if we want to compute the DPF, what is it that we need to compute in MPC?

correction words and flag corrections!

$\overrightarrow{b_0}$



$\overrightarrow{b_1}$



So how do we compute the correction word?

Let us answer the following question first: What are the possible values of the CW?

$s_0^{(0)} \oplus s_1^{(0)}$ $\quad \vec{b}_0[0] \oplus \vec{b}_1[0] = 1$

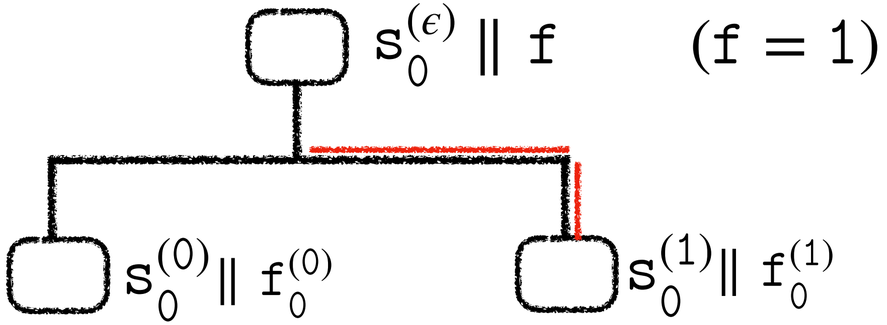
$s_0^{(1)} \oplus s_1^{(1)}$ $\quad \vec{b}_0[0] \oplus \vec{b}_1[0] = 0$

$$\vec{b}_0[0] \oplus \vec{b}_1[0] = b$$

$$\mathtt{cw} \leftarrow \mathtt{b} \cdot (s_0^{(0)} \oplus s_1^{(0)}) \oplus (1 - \mathtt{b}) \cdot (s_0^{(1)} \oplus s_1^{(1)})$$

This can give us shares of the correction word

$\overrightarrow{b_0}$



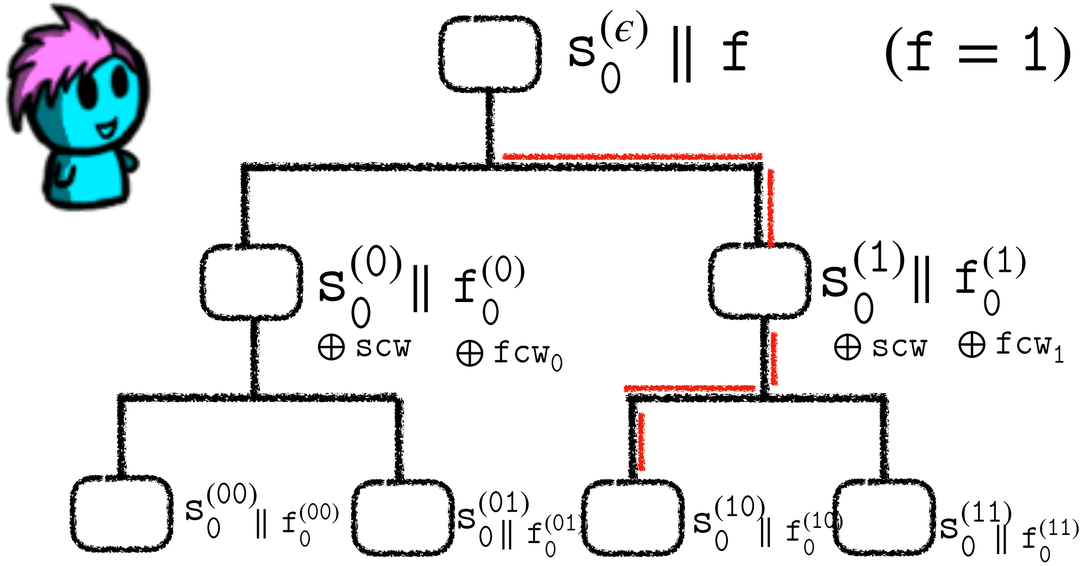
$\overrightarrow{b_1}$

$\vec{b}_0[0] \oplus \vec{b}_1[0] = b$

$\mathtt{cw} \leftarrow \mathtt{b} \cdot (\mathtt{s}_0^{(0)} \oplus \mathtt{s}_1^{(0)}) \oplus (1 - \mathtt{b}) \cdot (\mathtt{s}_0^{(1)} \oplus \mathtt{s}_1^{(1)})$

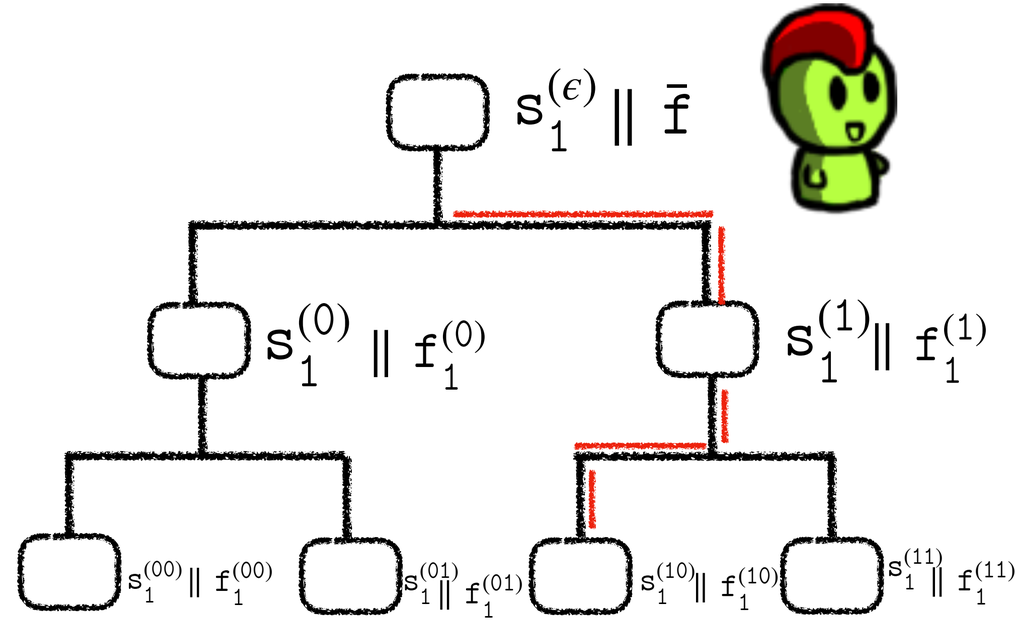
This can give us shares of the correction word

How do we move the next layer?



$$scw \leftarrow s_0^{(1)} \oplus s_1^{(1)}$$
$$fcw_0 \leftarrow f_0^{(0)} \oplus f_1^{(0)}$$
$$fcw_1 \leftarrow f_0^{(1)} \oplus f_1^{(1)} \oplus 1$$

$$scw \leftarrow s_0^{(10)} \oplus s_1^{(10)}$$
$$fcw_0 \leftarrow f_0^{(11)} \oplus f_1^{(11)}$$
$$fcw_1 \leftarrow f_0^{(01)} \oplus f_1^{(01)} \oplus 1$$



Why is this layer different from the previous layer? *four choices rather than two*

$$L_0 \leftarrow s_0^{(00)} \oplus s_0^{(10)}$$
$$R_0 \leftarrow s_0^{(01)} \oplus s_0^{(11)}$$

Does this help us?

$$L_1 \leftarrow s_1^{(00)} \oplus s_1^{(10)}$$
$$R_1 \leftarrow s_1^{(01)} \oplus s_1^{(11)}$$

$$scw \leftarrow s_0^{(1)} \oplus s_1^{(1)}$$
$$fcw_0 \leftarrow f_0^{(0)} \oplus f_1^{(0)}$$
$$fcw_1 \leftarrow f_0^{(1)} \oplus f_1^{(1)} \oplus 1$$

$$scw \leftarrow s_0^{(10)} \oplus s_1^{(10)}$$
$$fcw_0 \leftarrow f_0^{(11)} \oplus f_1^{(11)}$$
$$fcw_1 \leftarrow f_0^{(01)} \oplus f_1^{(01)} \oplus 1$$

$$L_0 \leftarrow s_0^{(00)} \oplus s_0^{(10)}$$
$$R_0 \leftarrow s_0^{(01)} \oplus s_0^{(11)}$$

Does this help us?

$$L_1 \leftarrow s_1^{(00)} \oplus s_1^{(10)}$$
$$R_1 \leftarrow s_1^{(01)} \oplus s_1^{(11)}$$

$$\vec{b}_0[1] \oplus \vec{b}_1[1] = b$$

$$cw \leftarrow b \cdot (L_0 \oplus L_1) \oplus (1 - b) \cdot (R_0 \oplus R_1)$$

Oblivious Single Function Memory

What is a Single Function Memory?

Read Only Memory:

How would you optimize your protocol if we have Read Only Memory.

Write Only Memory:

How would you optimize your protocol if we have Write Only Memory.

Oblivious Read Only Memory

What would be the best way to Oblivious Read Only Memory?

The first question to answer is how we would store the memory?

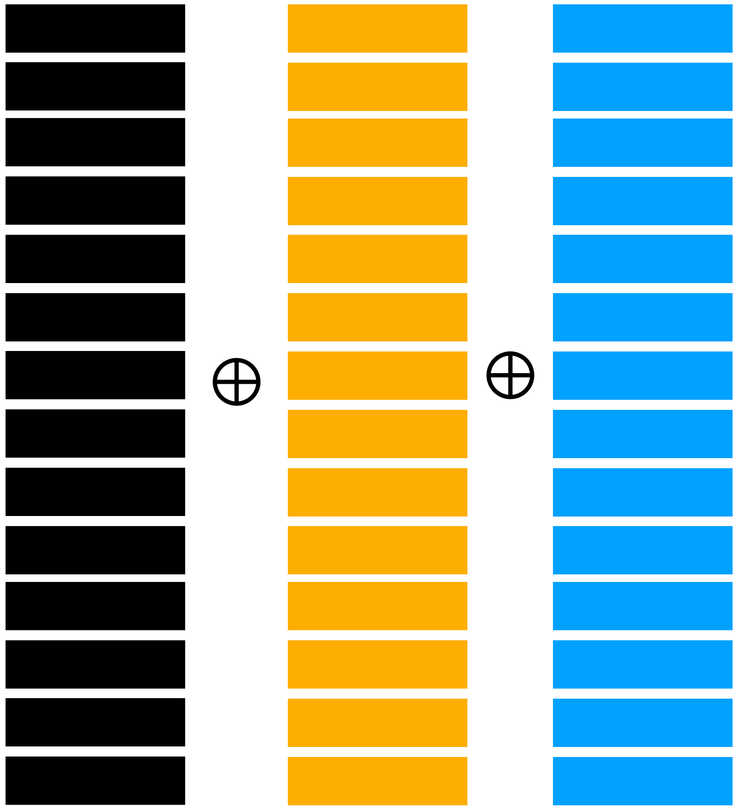
Is secret sharing a good option?

Any other suggestions?

Oblivious Read Only Memory

Alice $\overrightarrow{r_0}$

Bob $\overrightarrow{r_1}$



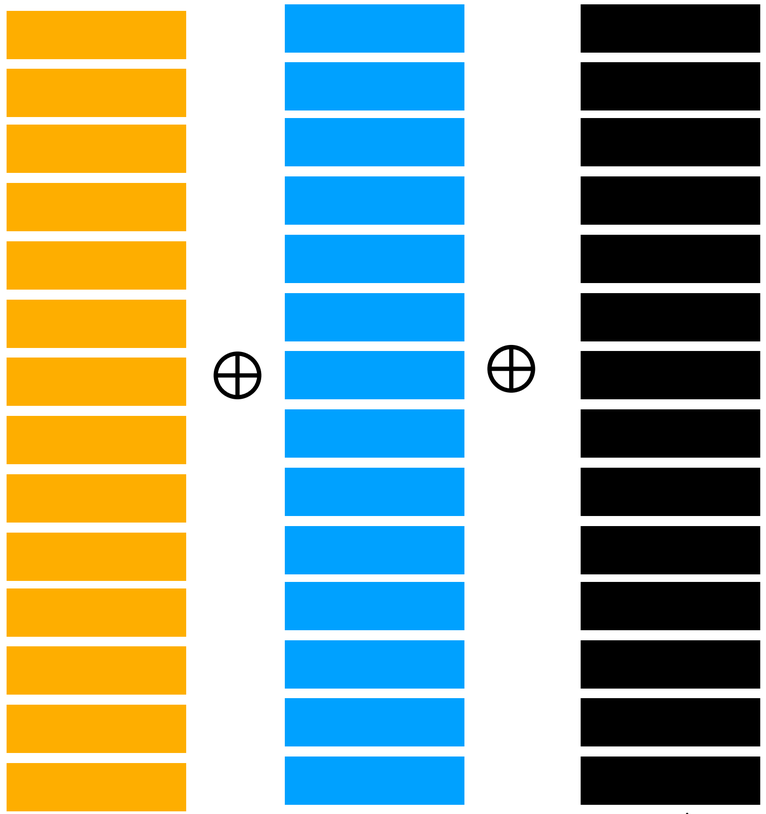
$\overrightarrow{D}$ $\overrightarrow{r_0}$ $\overrightarrow{r_1}$

How can we do a DORAM read in this setting?

Of course they get shares of $\overrightarrow{e_{i*}}$

(via MPC)

And then they just do a dot product! (not MPC but local)



$\overrightarrow{r_0}$ $\overrightarrow{r_1}$ $\overrightarrow{D}$

Oblivious Write Only Memory

What would be the best way to Oblivious Write Only Memory?

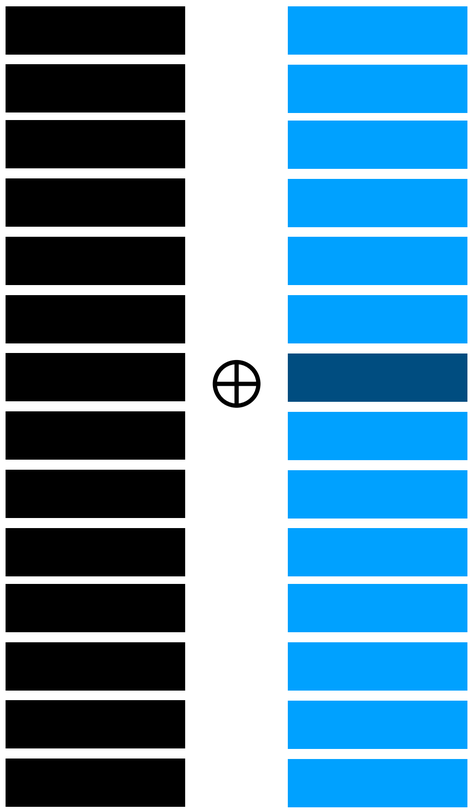
The first question to answer is how we would store the memory?

Is secret sharing a good option?

Why?

Oblivious Read Only Memory

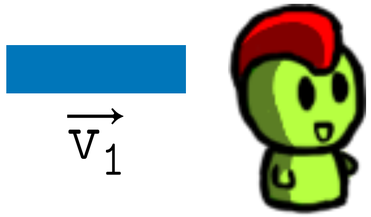
Alice $\vec{r_0}$



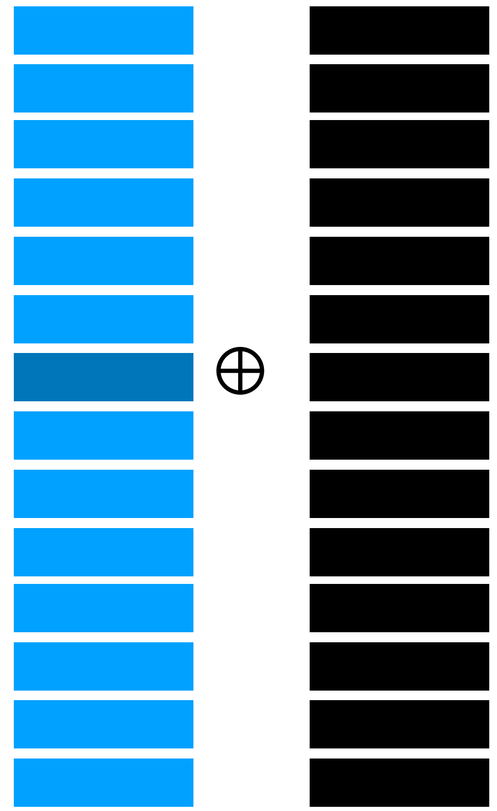
$$\vec{v_0} \oplus \vec{v_1} = \vec{e}_{i*} \cdot M$$

How will Alice and Bob get $\vec{v_0}$ *and* $\vec{v_1}$?

Well, we just saw the protocol!



Bob $\vec{r_1}$



DORAM from Single Function Memory

What is the difference between DORAM and a single-function memory?

we should be able to do both read and write operations!

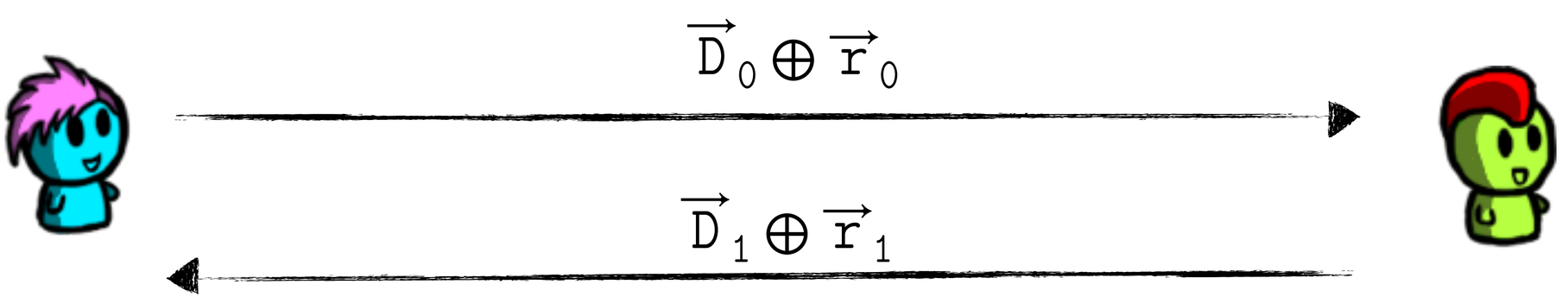
DORAM from Single Function Memory

Any suggestions on how to get a DORAM from Single Function Memory?

Perhaps one option is to find a way to switch from Write-Only-Memory to Read-Only-Memory?

Can you suggest a way?

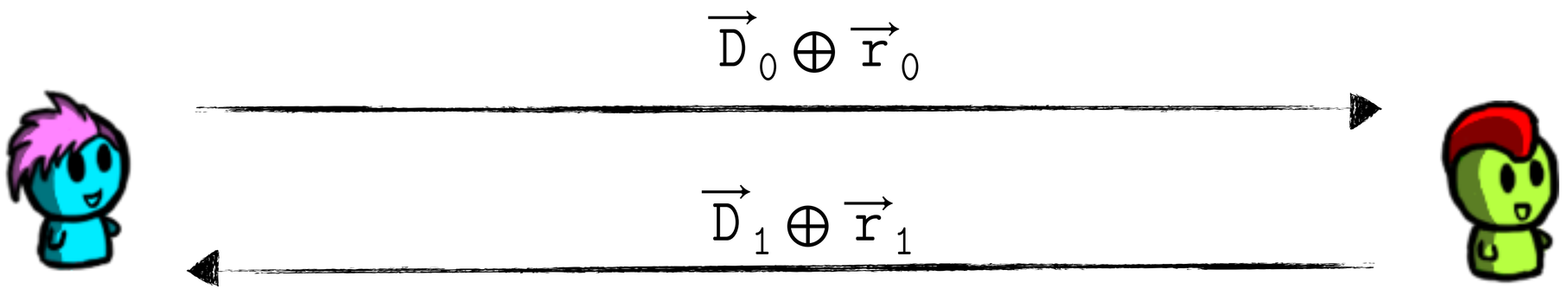
Write Only to Read Only



DORAM from Single Function Memory

Can you suggest a way?

Write Only to Read Only



Do we need a way to convert from *Read Only* to Write Only memory?

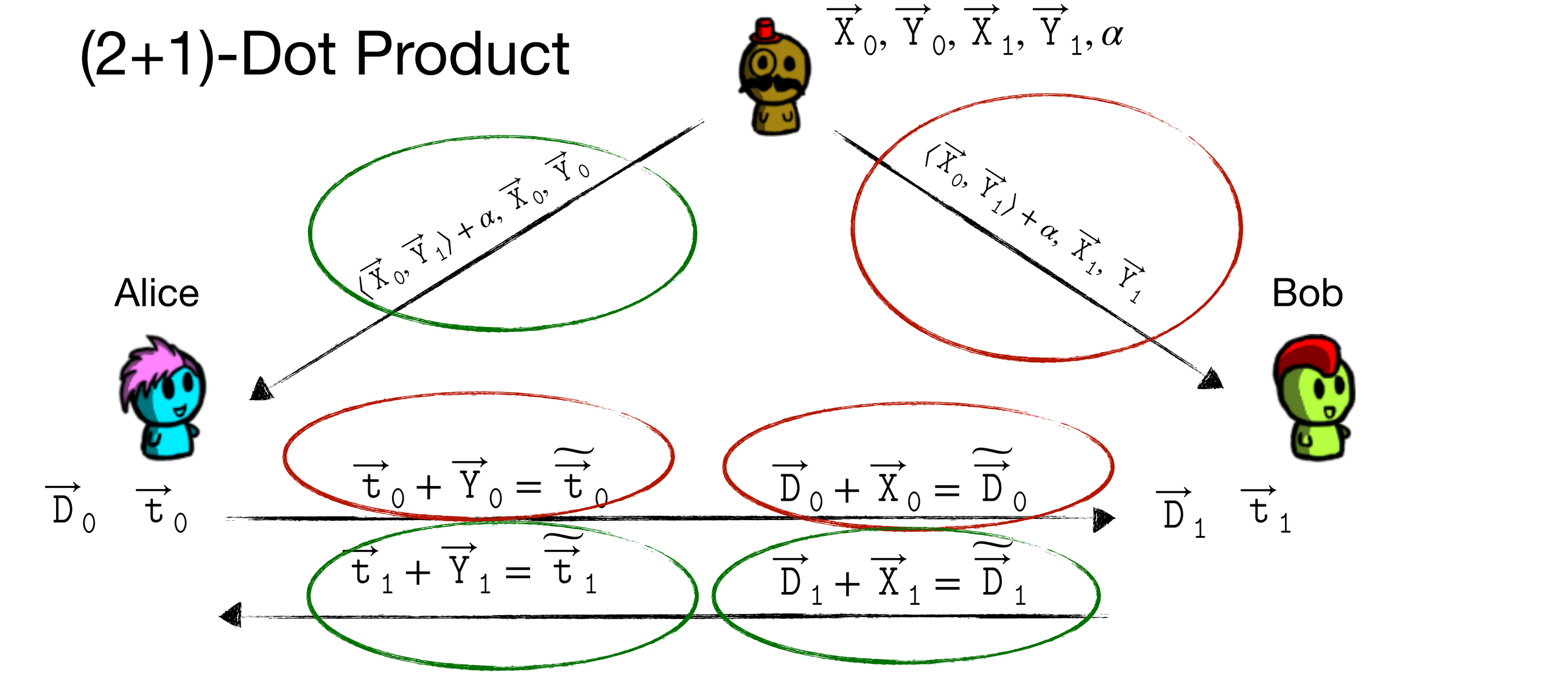
DORAM from Single Function Memory

The conversions from Write Only Memory to Read Only Memory is expensive!

Can we find a way to keep the memory the same for both read and write Memory?

MPC dotproducts!

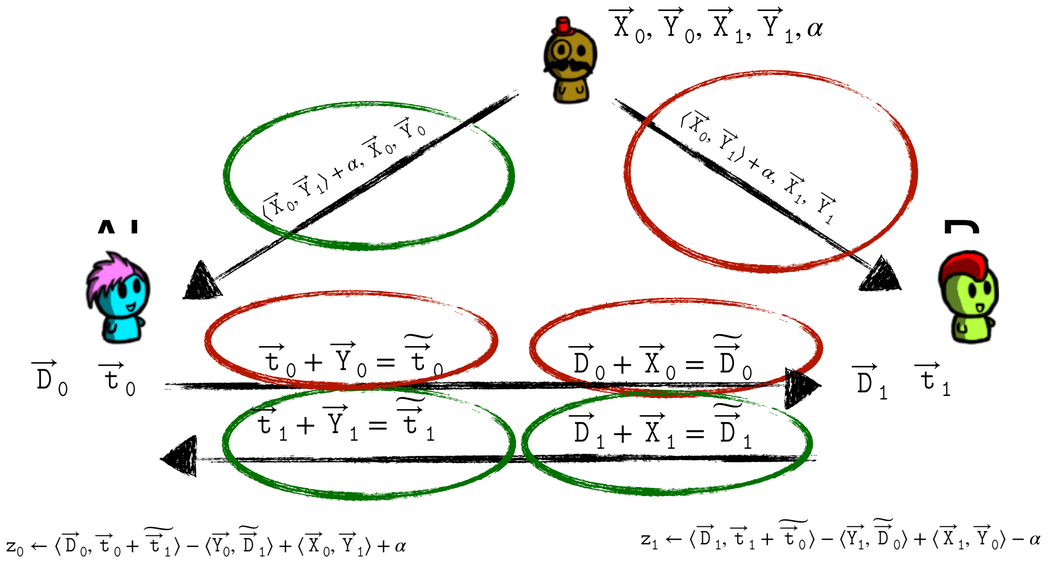
(2+1)-Dot Product



$z_0 \leftarrow \langle \vec{D}_0, \vec{t}_0 + \widetilde{\vec{t}_1} \rangle - \langle \vec{Y}_0, \widetilde{\vec{D}_1} \rangle + \langle \vec{X}_0, \vec{Y}_1 \rangle + \alpha$

$z_1 \leftarrow \langle \vec{D}_1, \vec{t}_1 + \widetilde{\vec{t}_0} \rangle - \langle \vec{Y}_1, \widetilde{\vec{D}_0} \rangle + \langle \vec{X}_1, \vec{Y}_0 \rangle - \alpha$

(2+1)-Dot Product



Is this is a good way to a DORAM read?

Do you see any problem with this?

Huge Communication cost is a problem

Everytime a new element needs to be read O(n) communication cost

Oblivious Read

(3P-MPC-dotproduct)

$\vec{t}_0^{(2)}$ $\vec{t}_1^{(3)}$

$\gamma_0 \leftarrow -\langle \vec{t}_1^{(3)}, \zeta_0 \rangle + \rho$

$\gamma_1 \leftarrow -\langle \vec{t}_0^{(2)}, \zeta_1 \rangle - \rho$

Alice

Bob

invariants need for 3P-MPC Dot Product

$$\vec{D}_0 + \zeta_0 = \widetilde{\vec{D}}_0$$

$$\vec{D}_1 + \zeta_1 = \widetilde{\vec{D}}_1$$

$\vec{D}_0$ $\vec{t}_0^{(1)}$ $\vec{t}_0^{(2)}$ $\vec{t}_0^{(3)}$

$\langle \vec{D}_0 + \widetilde{\vec{D}}_1, t_0^{(1)} \rangle - \langle \zeta_0, t_0^{(3)} - t_0^{(1)} \rangle + \gamma_0$

$\vec{D}_1$ $\vec{t}_1^{(1)}$ $\vec{t}_1^{(2)}$ $\vec{t}_1^{(3)}$

$\langle \vec{D}_1 + \widetilde{\vec{D}}_0, t_1^{(1)} \rangle - \langle \zeta_1, t_1^{(2)} - t_1^{(1)} \rangle + \gamma_1$

$$\vec{t}_0^{(1)} + t_1^{(1)} = \vec{t}_0^{(2)} + t_1^{(2)} = \vec{t}_0^{(3)} + t_1^{(3)} = \vec{e}_{i*}$$

How is this protocol different from the (2+1)-Party Du-Atallah protocol that was asked in Quiz 1

Oblivious Read

(3P-MPC-dotproduct)

$\overrightarrow{t}_0^{(2)}$ $\overrightarrow{t}_1^{(3)}$

$\gamma_0 \leftarrow -\langle \overrightarrow{t}_1^{(3)}, \zeta_0 \rangle + \rho$

$\gamma_1 \leftarrow -\langle \overrightarrow{t}_0^{(2)}, \zeta_1 \rangle - \rho$

Alice

Bob

$\overrightarrow{D}_0$ $\overrightarrow{t}_0^{(1)}$ $\overrightarrow{t}_0^{(2)}$ $\overrightarrow{t}_0^{(3)}$

$\langle \overrightarrow{D}_0 + \widetilde{\overrightarrow{D}}_1, t_0^{(1)} \rangle - \langle \zeta_0, t_0^{(3)} - t_0^{(1)} \rangle + \gamma_0$

invariants need for 3P-MPC Dot Product

$$\overrightarrow{D}_0 + \zeta_0 = \widetilde{\overrightarrow{D}}_0$$

$$\overrightarrow{D}_1 + \zeta_1 = \widetilde{\overrightarrow{D}}_1$$

$\overrightarrow{D}_1$ $\overrightarrow{t}_1^{(1)}$ $\overrightarrow{t}_1^{(2)}$ $\overrightarrow{t}_1^{(3)}$

$\langle \overrightarrow{D}_1 + \widetilde{\overrightarrow{D}}_0, t_1^{(1)} \rangle - \langle \zeta_1, t_1^{(2)} - t_1^{(1)} \rangle + \gamma_1$

Oblivious Read

$\overrightarrow{t}_0^{(2)}$ $\overrightarrow{t}_1^{(3)}$

$\gamma_0 \leftarrow -\langle \overrightarrow{t}_1^{(3)}, \zeta_0 \rangle + \rho$

$\gamma_1 \leftarrow -\langle \overrightarrow{t}_0^{(2)}, \zeta_1 \rangle - \rho$

Alice

Bob

invariants need for 3P-MPC Dot Product

$$\overrightarrow{D}_0 + \zeta_0 = \widetilde{\overrightarrow{D}}_0$$

$$\overrightarrow{D}_1 + \zeta_1 = \widetilde{\overrightarrow{D}}_1$$

$\overrightarrow{D}_0$ $\overrightarrow{t}_0^{(1)}$ $\overrightarrow{t}_0^{(2)}$ $\overrightarrow{t}_0^{(3)}$

$\langle \overrightarrow{D}_0 + \widetilde{\overrightarrow{D}}_1, t_0^{(1)} \rangle - \langle \zeta_0, t_0^{(3)} - t_0^{(1)} \rangle + \gamma_0$

$\overrightarrow{D}_1$ $\overrightarrow{t}_1^{(1)}$ $\overrightarrow{t}_1^{(2)}$ $\overrightarrow{t}_1^{(3)}$

$\langle \overrightarrow{D}_1 + \widetilde{\overrightarrow{D}}_0, t_1^{(1)} \rangle - \langle \zeta_1, t_1^{(2)} - t_1^{(1)} \rangle + \gamma_1$

What happens to the invariant when we do an oblivious update?

DORAM

When will the invariant (in the previous slide) be violated?

Whenever a write operation!

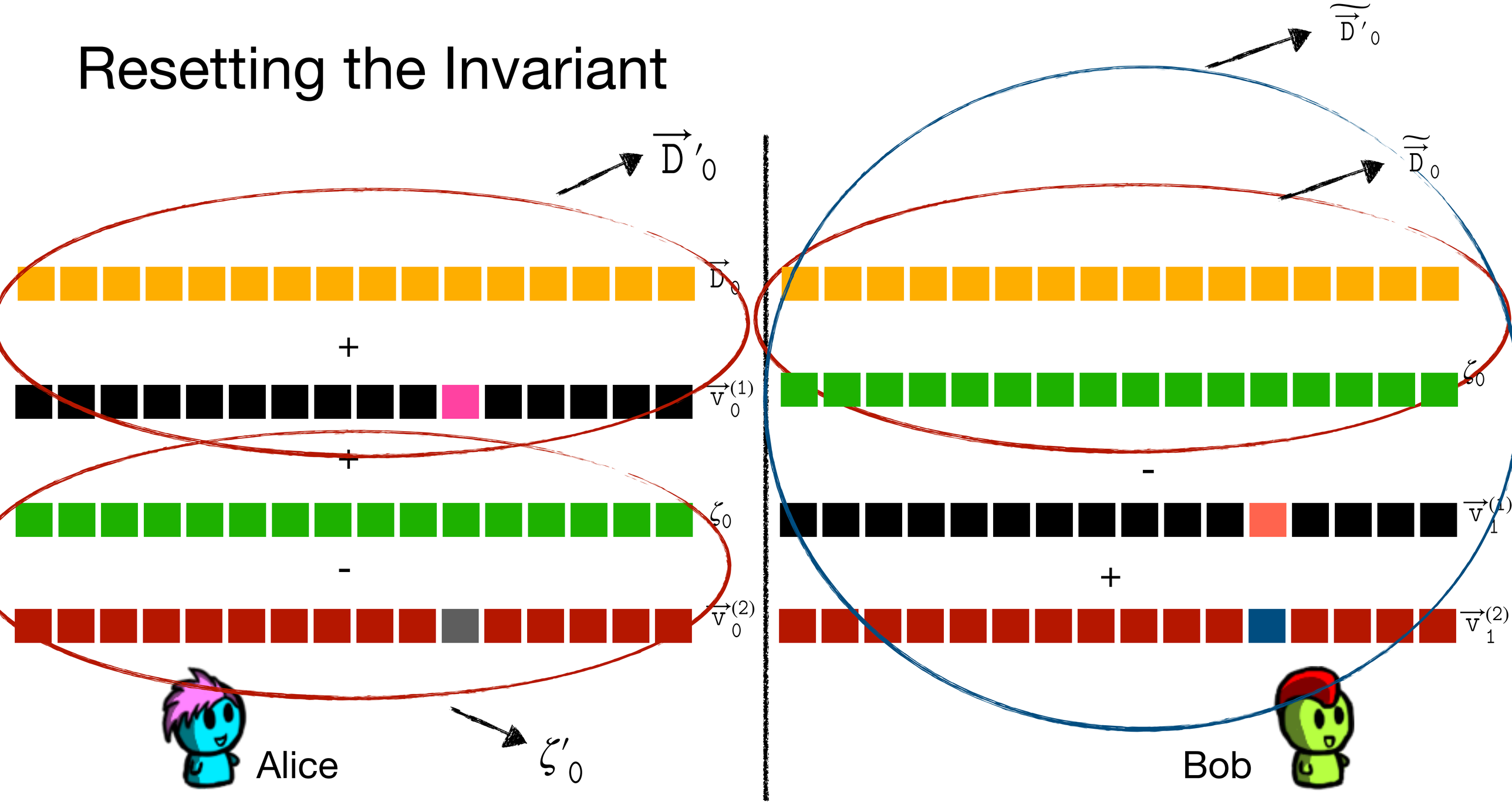
$$\overrightarrow{D}_0 + \zeta_0 = \widetilde{\overrightarrow{D}}_0$$

$$\overrightarrow{D}_1 + \zeta_1 = \widetilde{\overrightarrow{D}}_1$$

$$\overrightarrow{D'}_0 + \zeta_0 \neq \widetilde{\overrightarrow{D}}_0$$

$$\overrightarrow{D'}_1 + \zeta_1 \neq \widetilde{\overrightarrow{D}}_1$$

Resetting the Invariant



What did we achieve?

We are able reset the invariant with no communication in the online phase!

All the DPF evaluations (i.e., shares of standard-basis vectors are produced in the online phase) — that is not part of the course, you have seen some tricks to do that, but not all … can you tell me what are the missing pieces in the puzzle?