CORRELATED PSEUDORANDOMNESS FROM EXPAND-ACCUMULATE CODES

Elette Boyle
IDC Herzliya, NTT Research

Geoffroy Couteau
IRIF

Niv Gilboa
Ben-Gurion University

Yuval Ishai
Technion

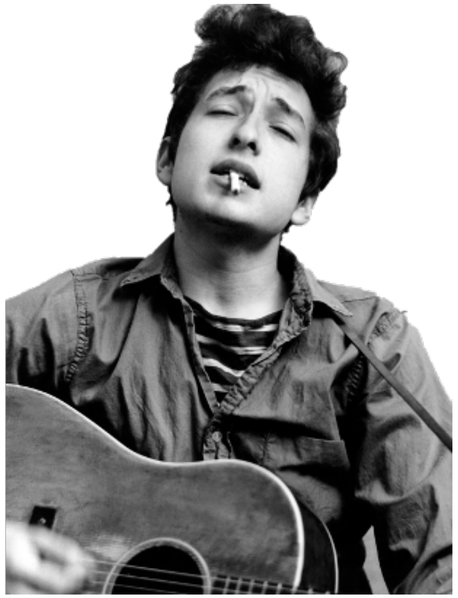
Lisa Kohl
CWI

Nicolas Resch
CWI → Uva

Peter Scholl
Aarhus University

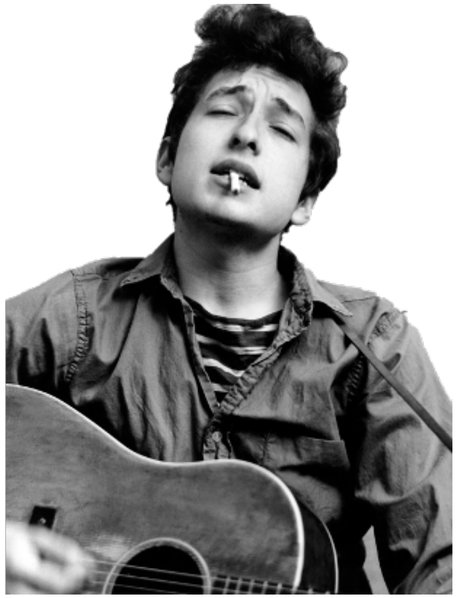
CORRELATED (PSEUDO)RANDOMNESS

SECURE COMMUNICATION: ONE-TIME PAD



SECURE COMMUNICATION: ONE-TIME PAD

m



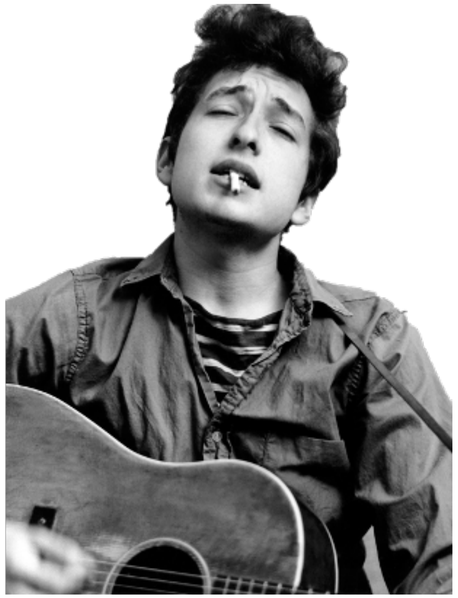
SECURE COMMUNICATION: ONE-TIME PAD

m

SECURE COMMUNICATION: ONE-TIME PAD

m OTP

OTP

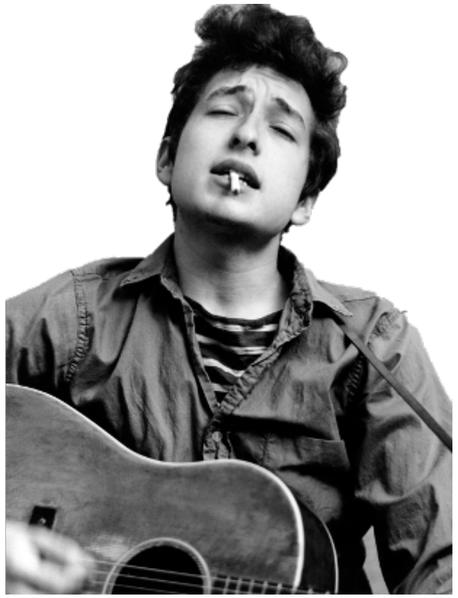


SECURE COMMUNICATION: ONE-TIME PAD

m OTP

$c = m \oplus \text{OTP}$

OTP

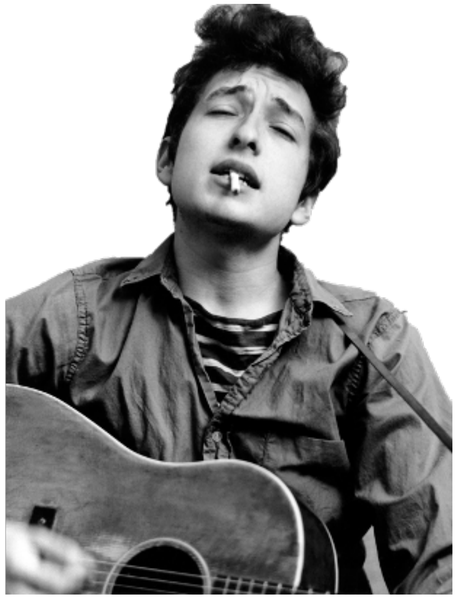


SECURE COMMUNICATION: ONE-TIME PAD

m OTP

$c = m \oplus \text{OTP}$

OTP

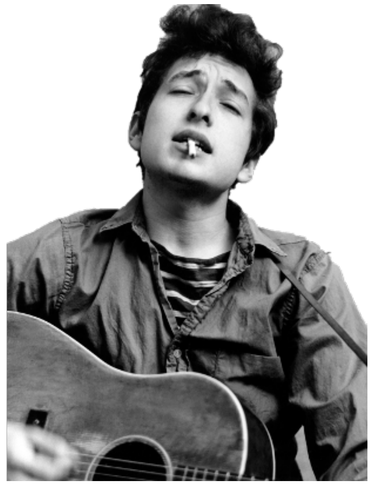


- Correlated Randomness: (OTP, OTP)

"equality" correlation

MULTIPARTY COMPUTATION (MPC)

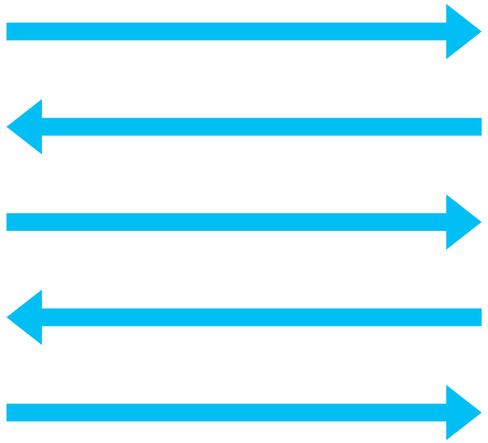
x

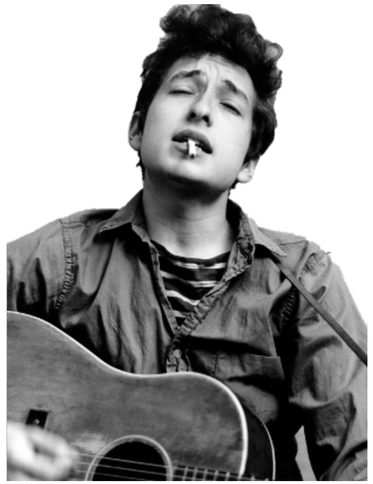


y

MULTIPARTY COMPUTATION (MPC)

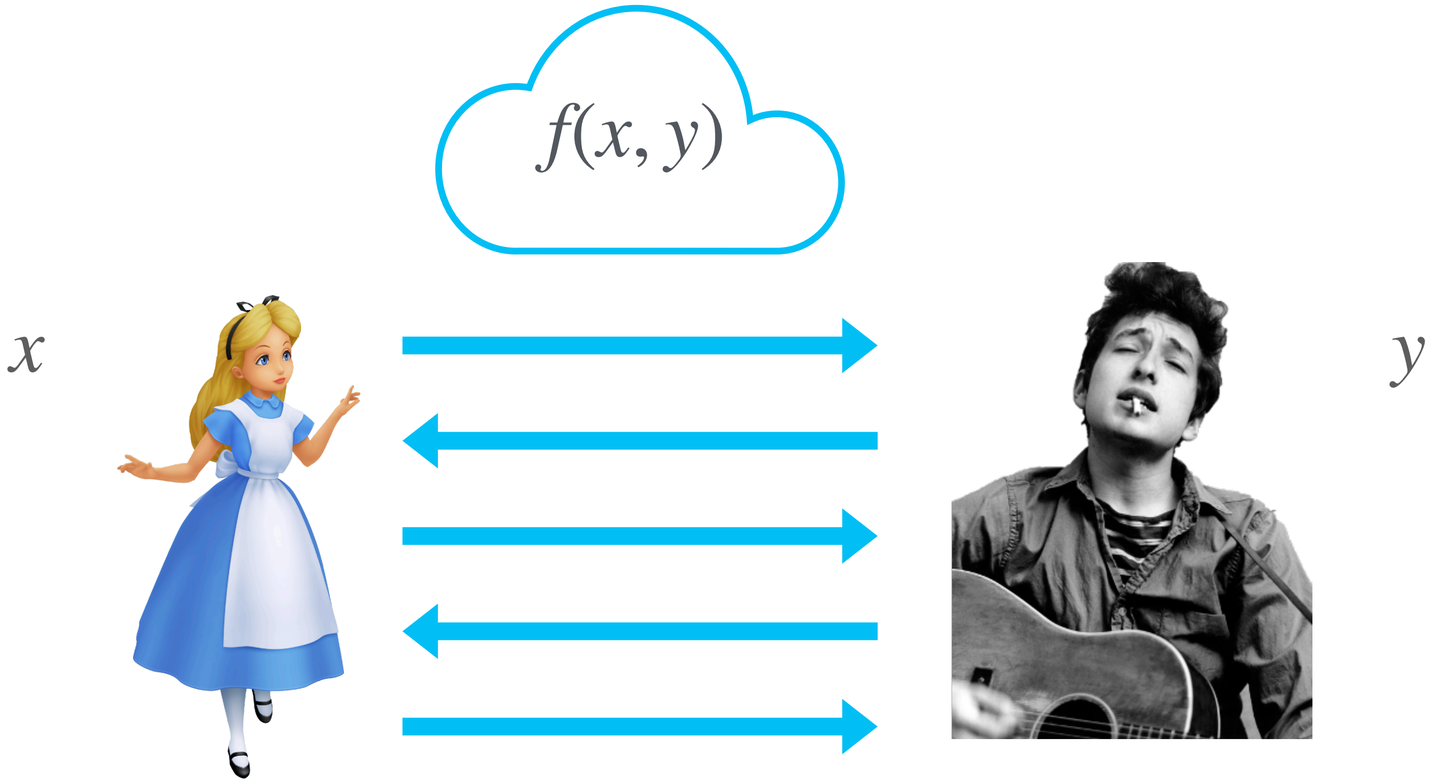
x





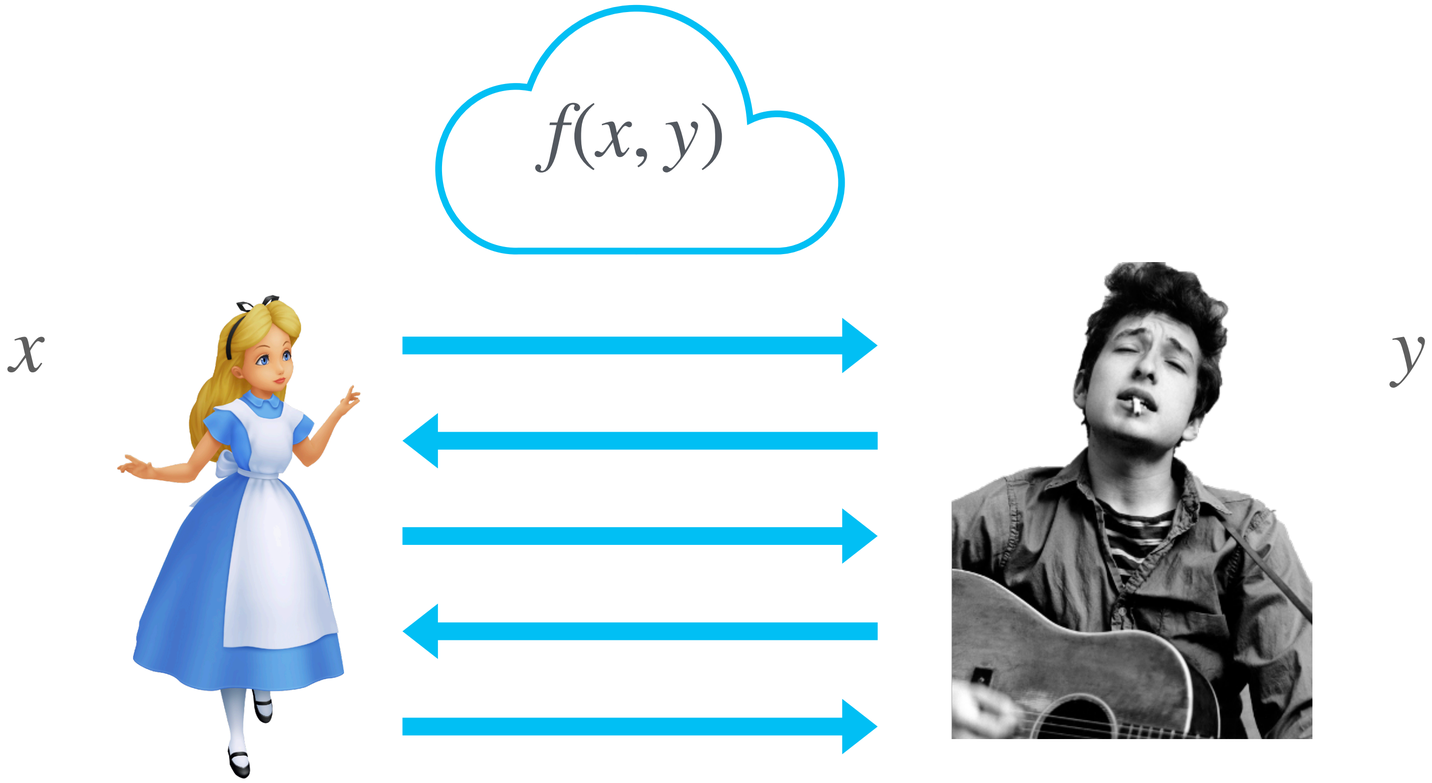
y

MULTIPARTY COMPUTATION (MPC)



Goal:
jointly compute $f(x, y)$,
without revealing
anything more about
private inputs x and y

MULTIPARTY COMPUTATION (MPC)



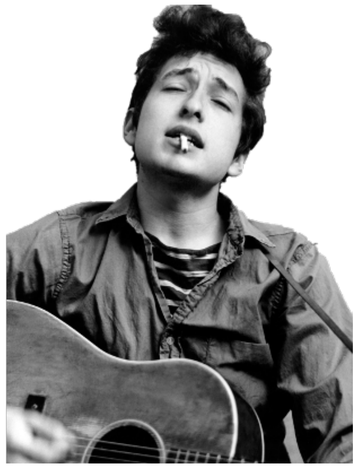
Goal:
jointly compute $f(x, y)$,
without revealing
anything more about
private inputs x and y

- **Today:** focus on case of 2 parties (2PC)

CORRELATION FOR 2PC (AND)

$x, (b, m_b)$

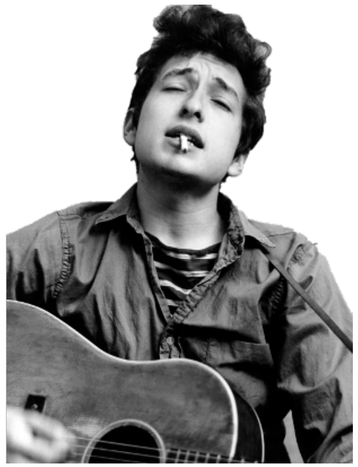
$y, (m_0, m_1)$



CORRELATION FOR 2PC (AND)

$x, (b, m_b)$

$y, (m_0, m_1)$



- Correlated randomness: ((b, m_b), (m_0, m_1))

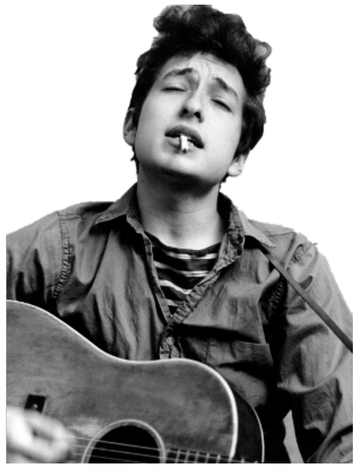
Oblivious Transfer (OT)

CORRELATION FOR 2PC (AND)

$x, (b, m_b)$

$c = x \oplus b$

$y, (m_0, m_1)$



- Correlated randomness: ((b, m_b), (m_0, m_1))

Oblivious Transfer (OT)

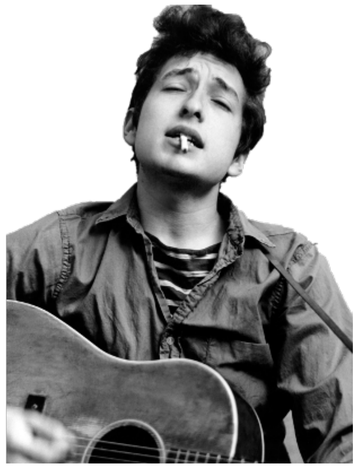
CORRELATION FOR 2PC (AND)

$x, (b, m_b)$

$c = x \oplus b$

$d = y \oplus m_{c \oplus 1}$

$y, (m_0, m_1)$



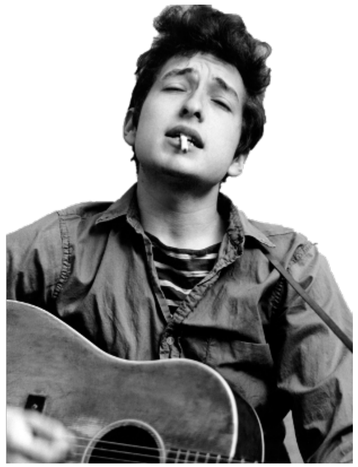
- Correlated randomness: ((b, m_b), (m_0, m_1))

Oblivious Transfer (OT)

CORRELATION FOR 2PC (AND)

$x, (b, m_b)$

$y, (m_0, m_1)$



$$c = x \oplus b$$

$$d = y \oplus m_{c \oplus 1}$$

$$x \wedge y = \begin{cases} 0 & \text{if } x = 0 \\ y = d \oplus m_b & \text{else} \end{cases}$$

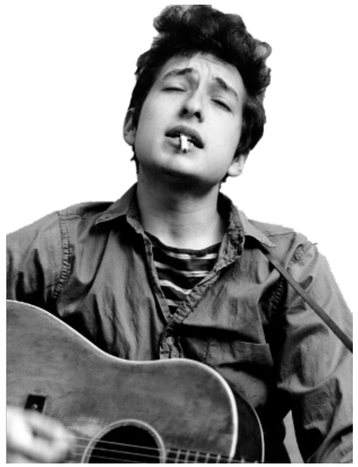
- Correlated randomness: ($(b, m_b), (m_0, m_1)$)

Oblivious Transfer (OT)

CORRELATION FOR 2PC (AND)

$x, (b, m_b)$

$y, (m_0, m_1)$



$$c = x \oplus b$$

$$d = y \oplus m_{c \oplus 1}$$

$$x \wedge y = \begin{cases} 0 & \text{if } x = 0 \\ y = d \oplus m_b & \text{else} \end{cases}$$

- Correlated randomness: ((b, m_b), (m_0, m_1))

Oblivious Transfer (OT)

MPC with preprocessing: exchange correlated randomness first, then run (fast!) protocol

COMPRESSING RANDOMNESS

COMPRESSING RANDOMNESS

- Problem: typically need *long* correlated random strings

COMPRESSING RANDOMNESS

- Problem: typically need *long* correlated random strings
  - Secure Communication: $|\text{OTP}| = |\text{message}|$

COMPRESSING RANDOMNESS

- **Problem:** typically need *long* correlated random strings
  - Secure Communication: $|\text{OTP}| = |\text{message}|$

Solution ['80's]: exchange short seeds, stretch with pseudorandom generator / function

COMPRESSING RANDOMNESS

- **Problem:** typically need *long* correlated random strings
  - Secure Communication: $|\text{OTP}| = |\text{message}|$

Solution ['80's]: exchange short seeds, stretch with pseudorandom generator / function

  - Secure Computation: 2 OT's per AND gate

COMPRESSING RANDOMNESS

- **Problem:** typically need *long* correlated random strings
  - Secure Communication: $|\text{OTP}| = |\text{message}|$

Solution ['80's]: exchange short seeds, stretch with pseudorandom generator / function

  - Secure Computation: 2 OT's per AND gate
    - Traditionally, need preprocessing phase with high communication

COMPRESSING RANDOMNESS

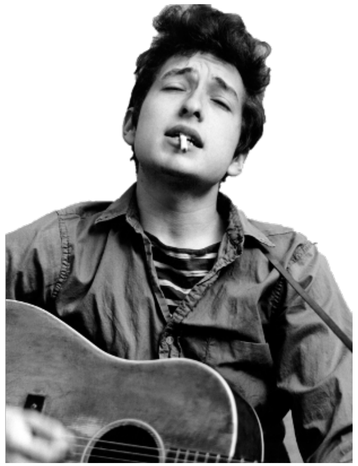
- **Problem:** typically need *long* correlated random strings
  - Secure Communication: $|\text{OTP}| = |\text{message}|$

Solution ['80's]: exchange short seeds, stretch with pseudorandom generator / function

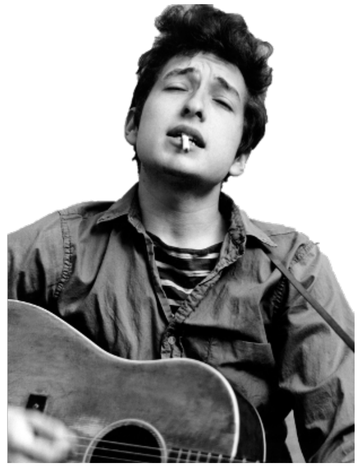
  - Secure Computation: 2 OT's per AND gate
    - Traditionally, need preprocessing phase with high communication

Solution [2015—]: exchange short seeds, stretch with pseudorandom *correlation* generator / function

PSEUDORANDOM CORRELATION GENERATOR (PCG)

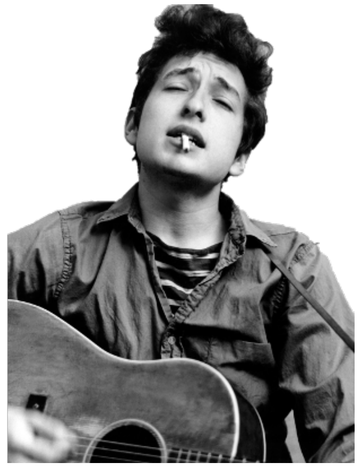


PSEUDORANDOM CORRELATION GENERATOR (PCG)



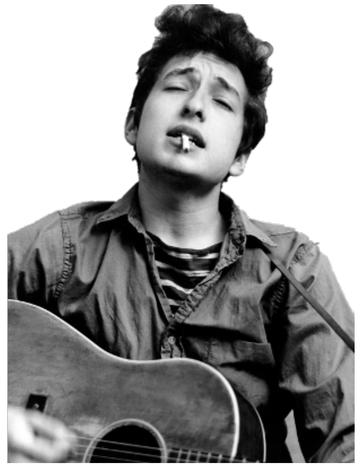
- PCG $=$ (Gen, Expand) for correlation C

PSEUDORANDOM CORRELATION GENERATOR (PCG)

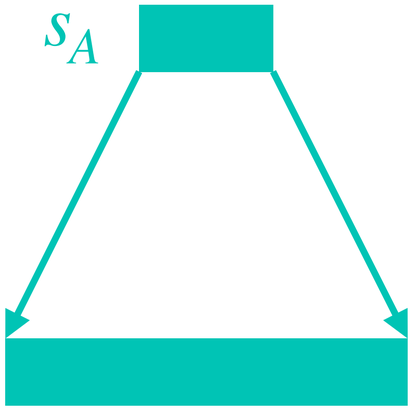


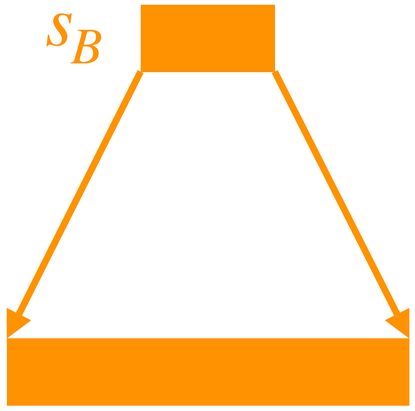
- PCG $=$ (Gen, Expand) for correlation C
- For $(s_A, s_B) \leftarrow \text{Gen}(1^\lambda)$:

PSEUDORANDOM CORRELATION GENERATOR (PCG)



- PCG = (Gen, Expand) for correlation C
- For $(s_A, s_B) \leftarrow \text{Gen}(1^\lambda)$:

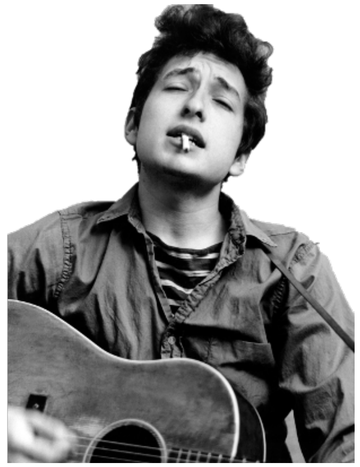


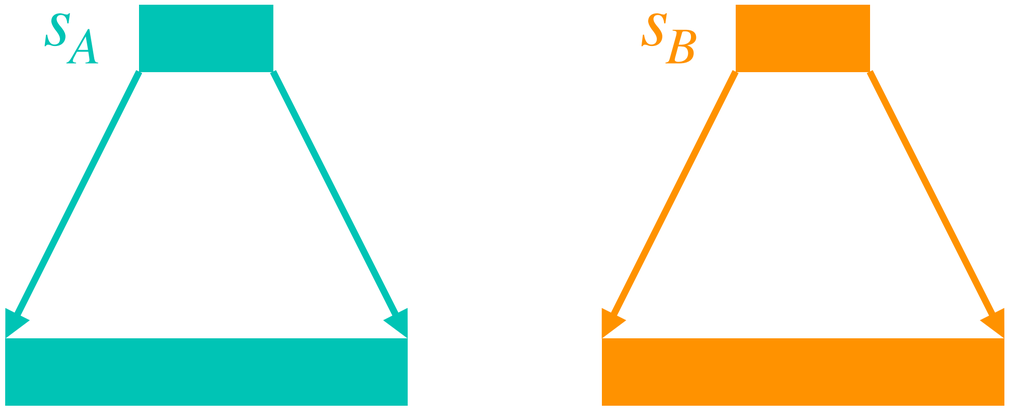


Expand(s_A)

Expand(s_B)

PSEUDORANDOM CORRELATION GENERATOR (PCG)

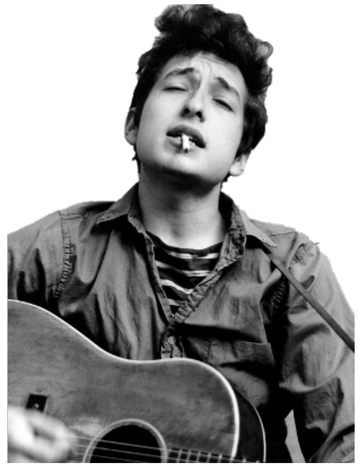


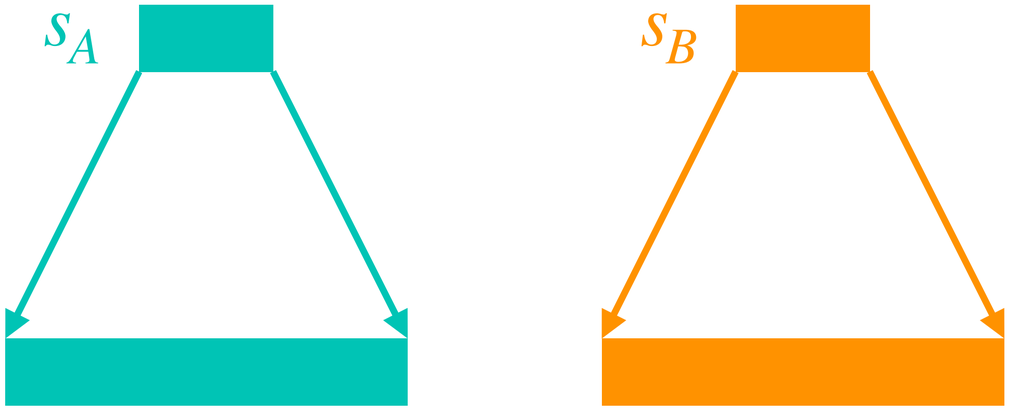


Expand(s_A) Expand(s_B)

- PCG = (Gen, Expand) for correlation C
- For $(s_A, s_B) \leftarrow \text{Gen}(1^\lambda)$:
- Correctness:
  - $(\text{Expand}(s_A), \text{Expand}(s_B)) \in C^N$

PSEUDORANDOM CORRELATION GENERATOR (PCG)



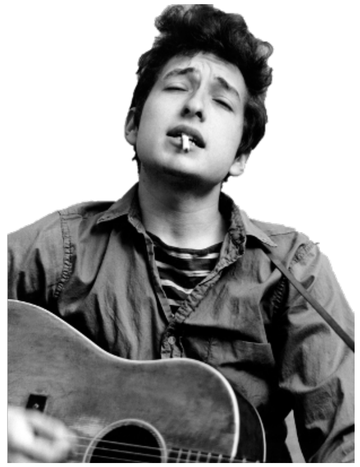


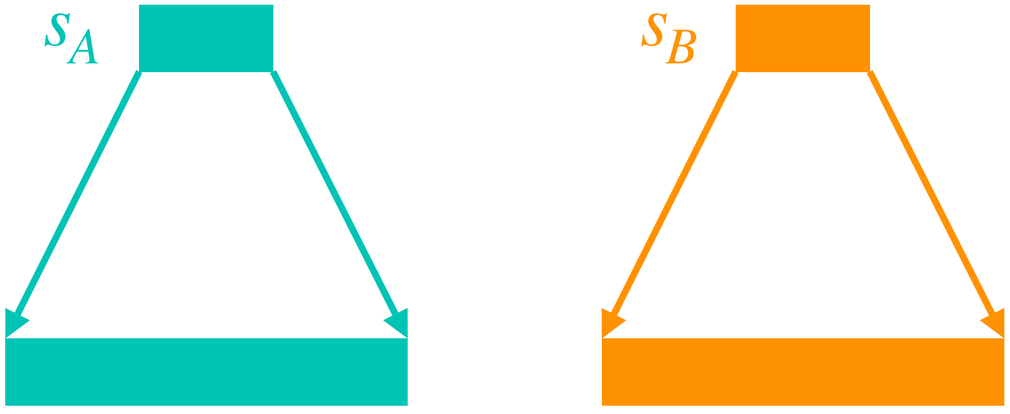
Expand(s_A) Expand(s_B)

- PCG = (Gen, Expand) for correlation C
- For $(s_A, s_B) \leftarrow \text{Gen}(1^\lambda)$:
- Correctness:
  - $(\text{Expand}(s_A), \text{Expand}(s_B)) \in C^N$

N indep. OT's

PSEUDORANDOM CORRELATION GENERATOR (PCG)



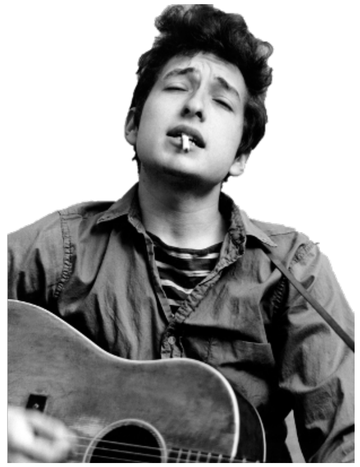


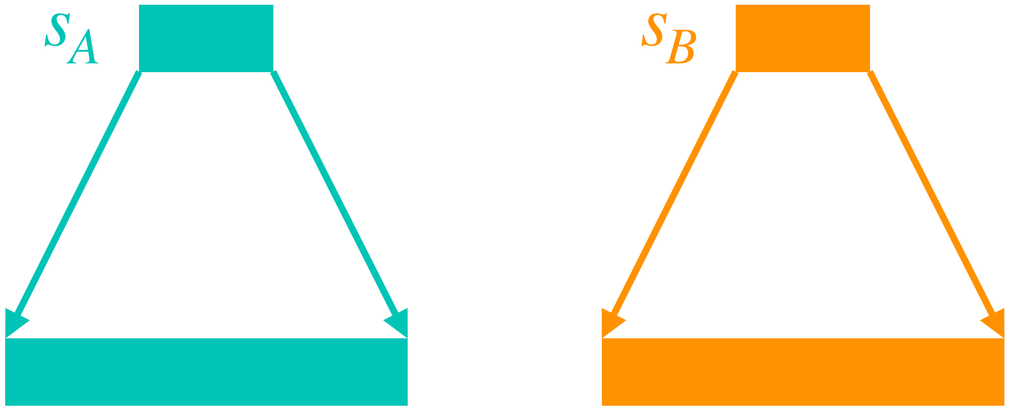
Expand(s_A) Expand(s_B)

- PCG = (Gen, Expand) for correlation C
- For $(s_A, s_B) \leftarrow \text{Gen}(1^\lambda)$:
- Correctness:
  - $(\text{Expand}(s_A), \text{Expand}(s_B)) \in C^N$
- Pseudorandomness:
  - Expand(s_A), Expand(s_B) pseudorand.

N indep. OT's

PSEUDORANDOM CORRELATION GENERATOR (PCG)





Expand(s_A) Expand(s_B)

- PCG = (Gen, Expand) for correlation C
- For $(s_A, s_B) \leftarrow \text{Gen}(1^\lambda)$:
- Correctness:
  - $(\text{Expand}(s_A), \text{Expand}(s_B)) \in C^N$
- Pseudorandomness:
  - Expand(s_A), Expand(s_B) pseudorand.
- Security:
  - Other party's output looks pseudorandom up to correlation

N indep. OT's

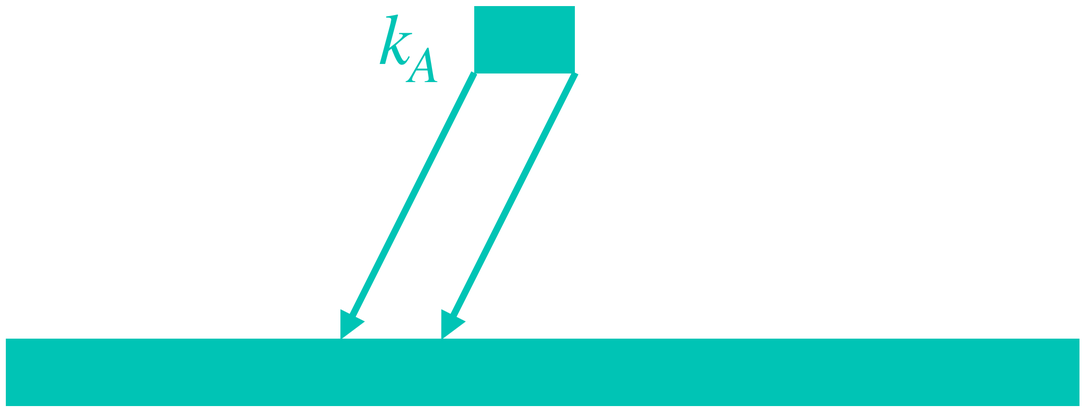
PSEUDORANDOM CORRELATION FUNCTION (PCF)

PSEUDORANDOM CORRELATION FUNCTION (PCF)

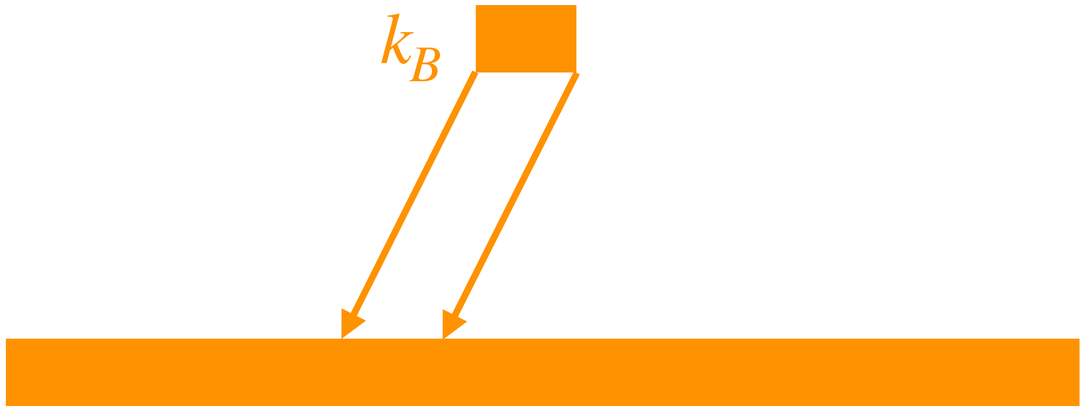
- For $(k_A, k_B) \leftarrow \mathsf{Gen}(1^\lambda)$

PSEUDORANDOM CORRELATION FUNCTION (PCF)

- For $(k_A, k_B) \leftarrow \text{Gen}(1^\lambda)$



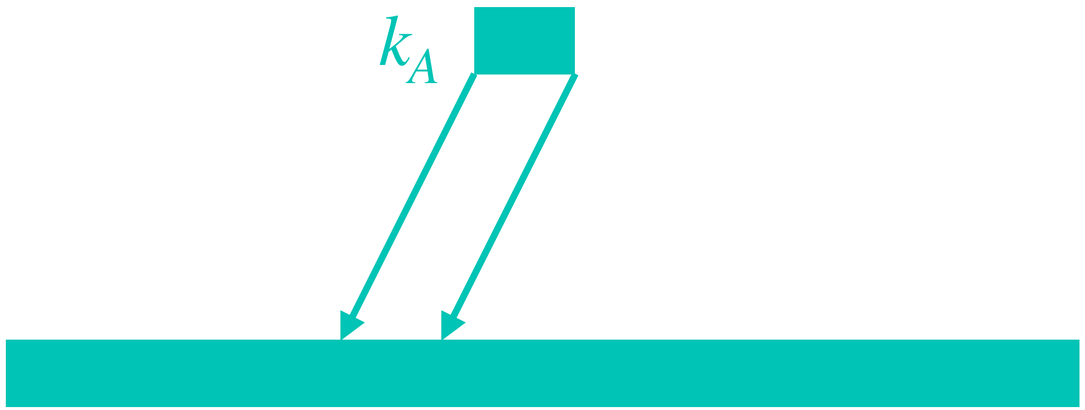
$\text{Eval}(k_A, x)$



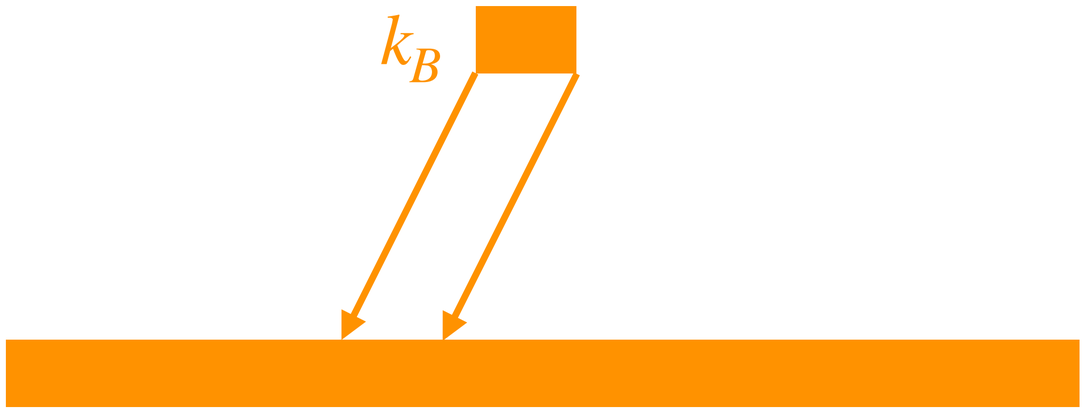
$\text{Eval}(k_B, x)$

PSEUDORANDOM CORRELATION FUNCTION (PCF)

- For $(k_A, k_B) \leftarrow \text{Gen}(1^\lambda)$



$\text{Eval}(k_A, x)$



$\text{Eval}(k_B, x)$

- Analogous correctness, pseudorandomness and security guarantees

OFFLINE-ONLINE PCG'S

OFFLINE-ONLINE PCG'S

PCG's: all work in offline phase

OFFLINE-ONLINE PCG'S

PCG's: all work in offline phase

PCF's: all work in online phase

OFFLINE-ONLINE PCG'S

PCG's: all work in offline phase

PCF's: all work in online phase

More flexibility: offline-online PCG's

OFFLINE-ONLINE PCG'S

PCG's: all work in offline phase

PCF's: all work in online phase

More flexibility: offline-online PCG's

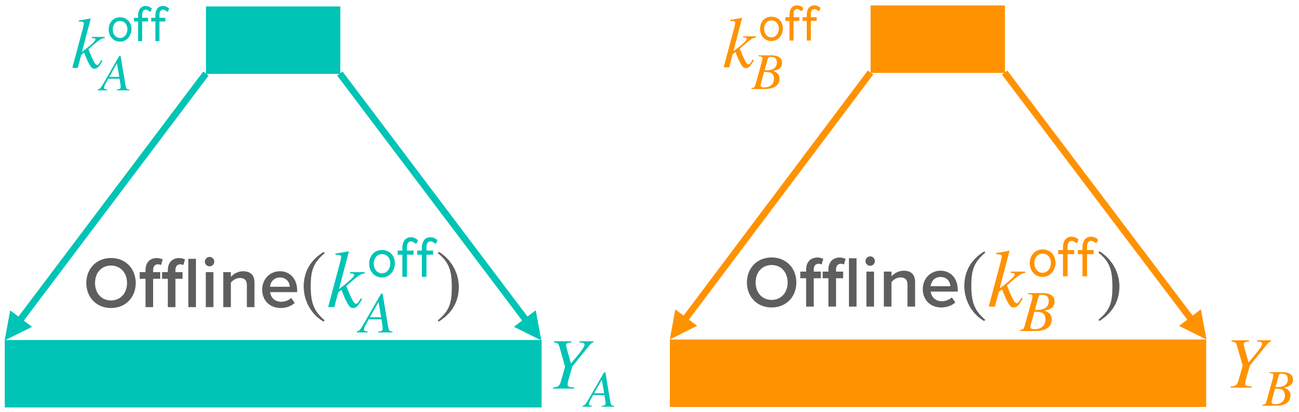
- PCG = (Gen, Offline, Online)
- $(k_A^{\text{off}}, k_B^{\text{off}}, k_A^{\text{on}}, k_B^{\text{on}}) \leftarrow \text{Gen}(1^\lambda)$

OFFLINE-ONLINE PCG'S

PCG's: all work in offline phase

PCF's: all work in online phase

More flexibility: offline-online PCG's



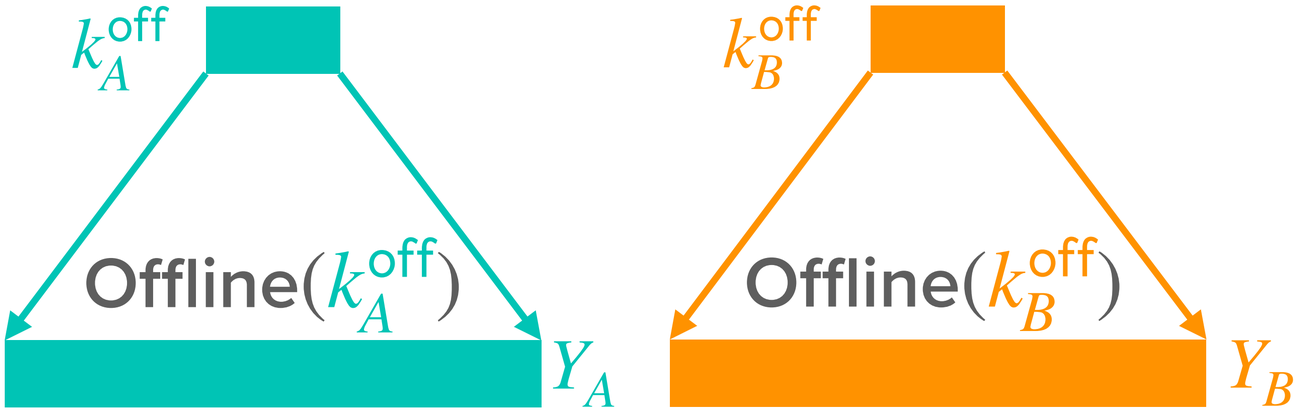
- PCG = (Gen, Offline, Online)
- $(k_A^{\text{off}}, k_B^{\text{off}}, k_A^{\text{on}}, k_B^{\text{on}}) \leftarrow \text{Gen}(1^\lambda)$

OFFLINE-ONLINE PCG'S

PCG's: all work in offline phase

PCF's: all work in online phase

More flexibility: offline-online PCG's



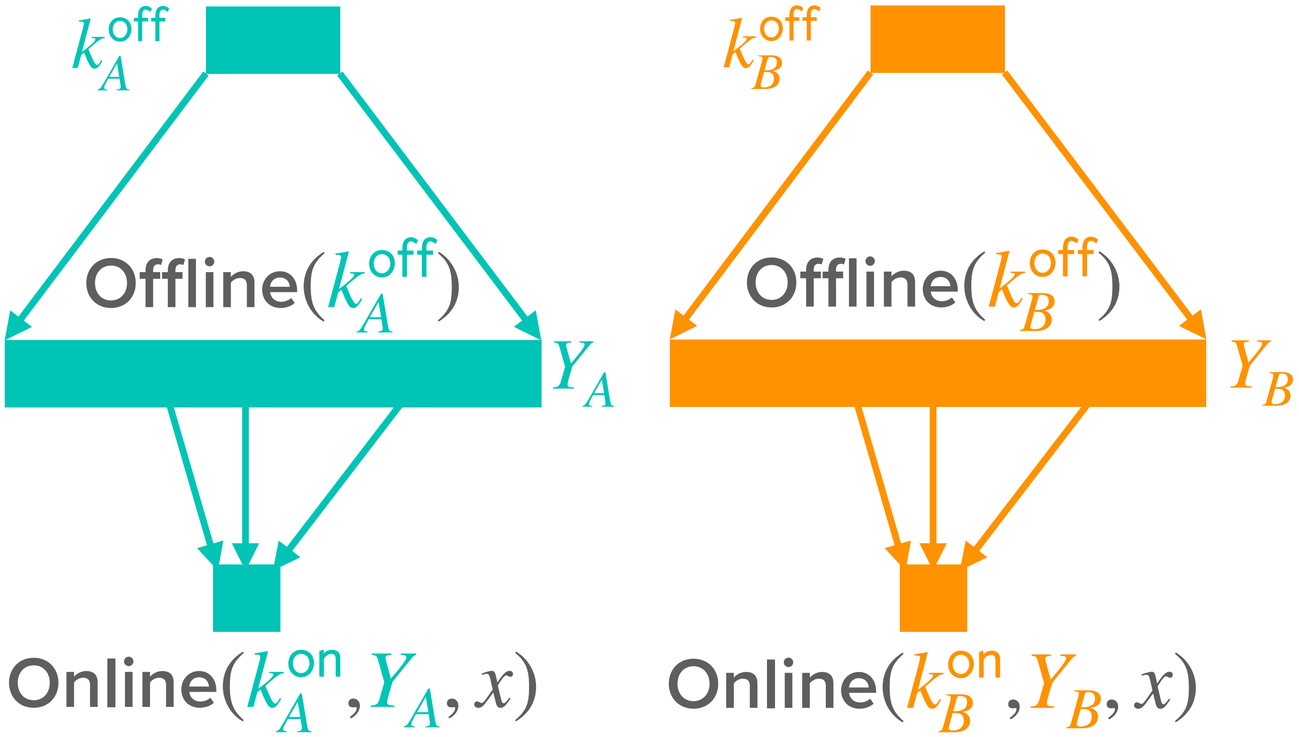
- PCG = (Gen, Offline, Online)
- $(k_A^{\text{off}}, k_B^{\text{off}}, k_A^{\text{on}}, k_B^{\text{on}}) \leftarrow \text{Gen}(1^\lambda)$
- Low storage: $|Y_\sigma| \leq N$

OFFLINE-ONLINE PCG'S

PCG's: all work in offline phase

PCF's: all work in online phase

More flexibility: offline-online PCG's



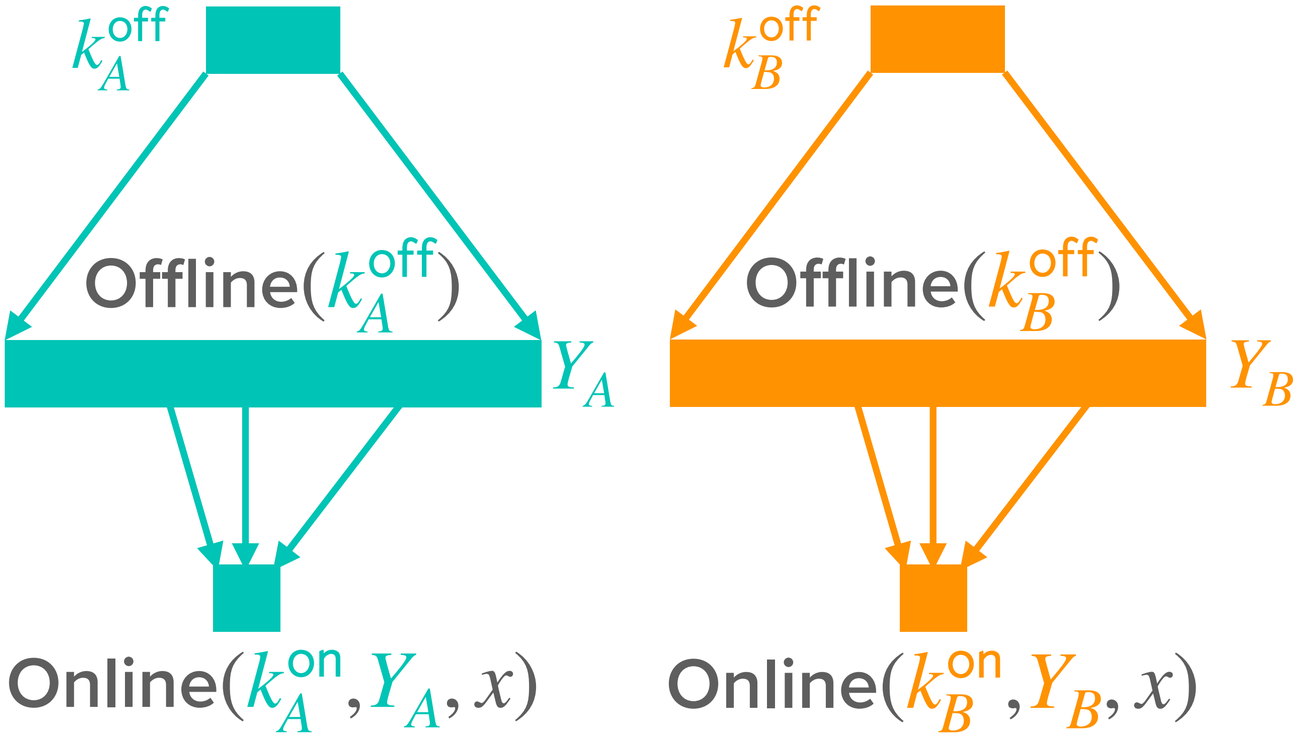
- PCG = (Gen, Offline, Online)
- $(k_A^{\text{off}}, k_B^{\text{off}}, k_A^{\text{on}}, k_B^{\text{on}}) \leftarrow \text{Gen}(1^\lambda)$
- Low storage: $|Y_\sigma| \leq N$

OFFLINE-ONLINE PCG'S

PCG's: all work in offline phase

PCF's: all work in online phase

More flexibility: offline-online PCG's



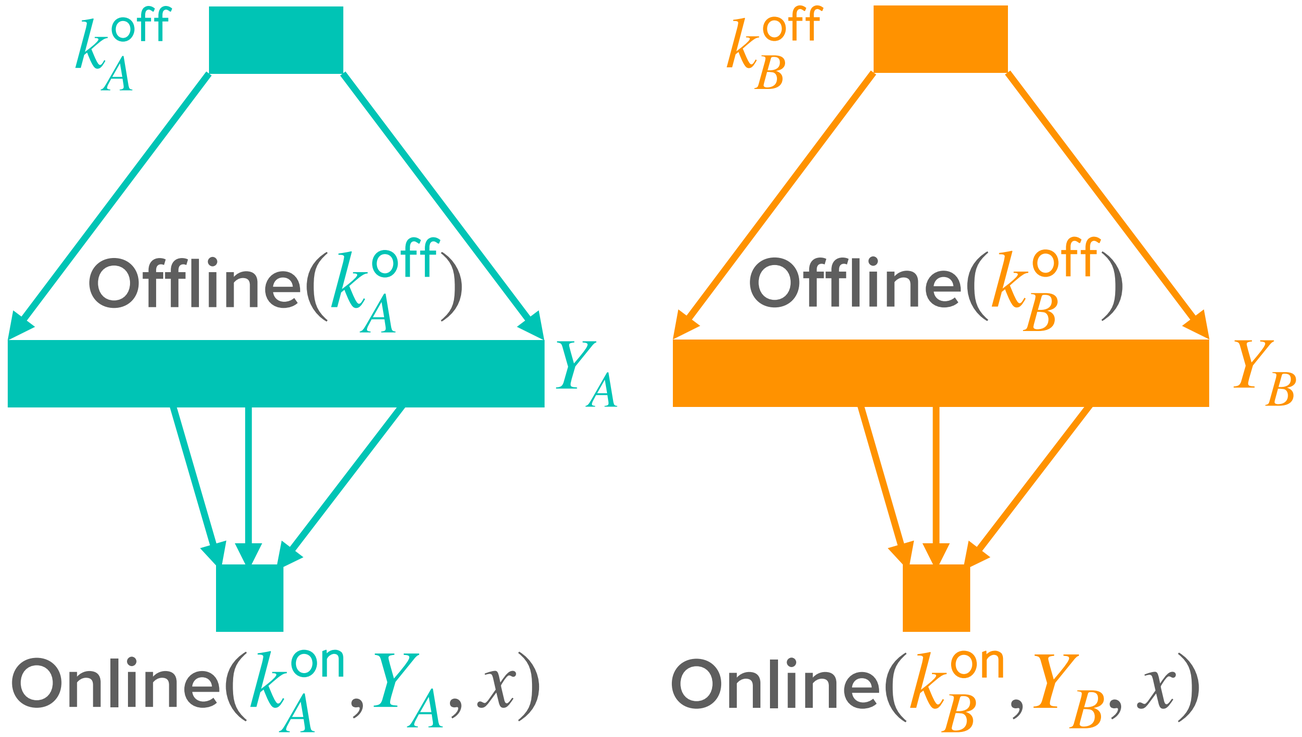
- PCG = (Gen, Offline, Online)
- $(k_A^{\text{off}}, k_B^{\text{off}}, k_A^{\text{on}}, k_B^{\text{on}}) \leftarrow \text{Gen}(1^\lambda)$
- **Low storage:** $|Y_\sigma| \leq N$
- **Output locality:** Online reads $\leq \ell$ entries of Y_σ

OFFLINE-ONLINE PCG'S

PCG's: all work in offline phase

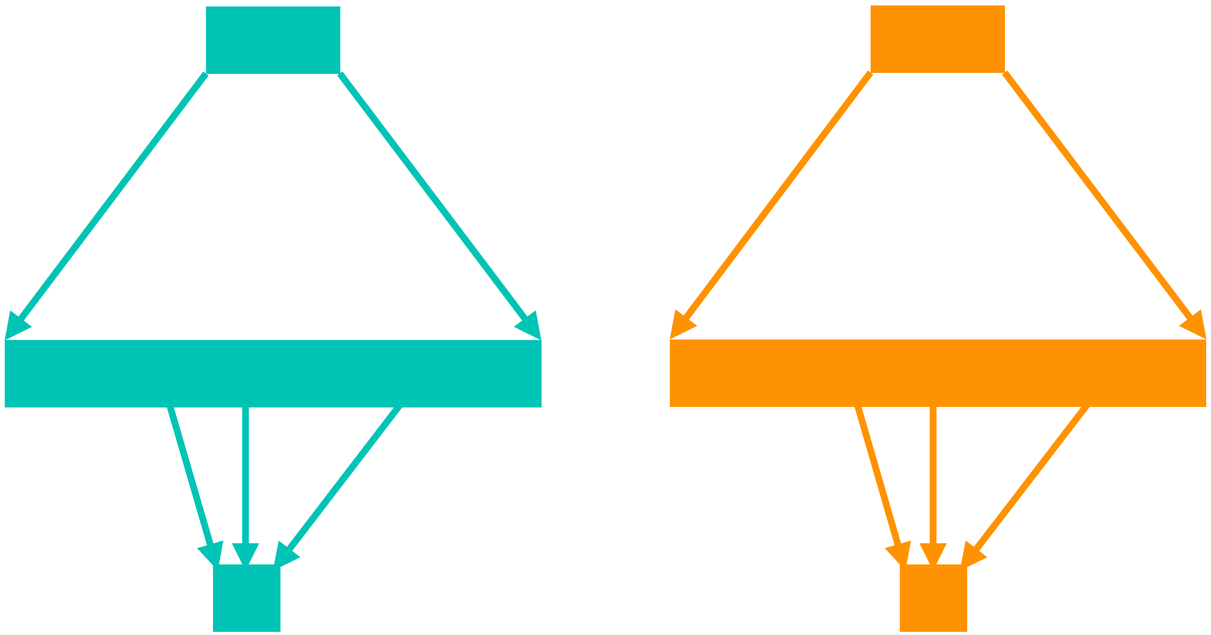
PCF's: all work in online phase

More flexibility: offline-online PCG's



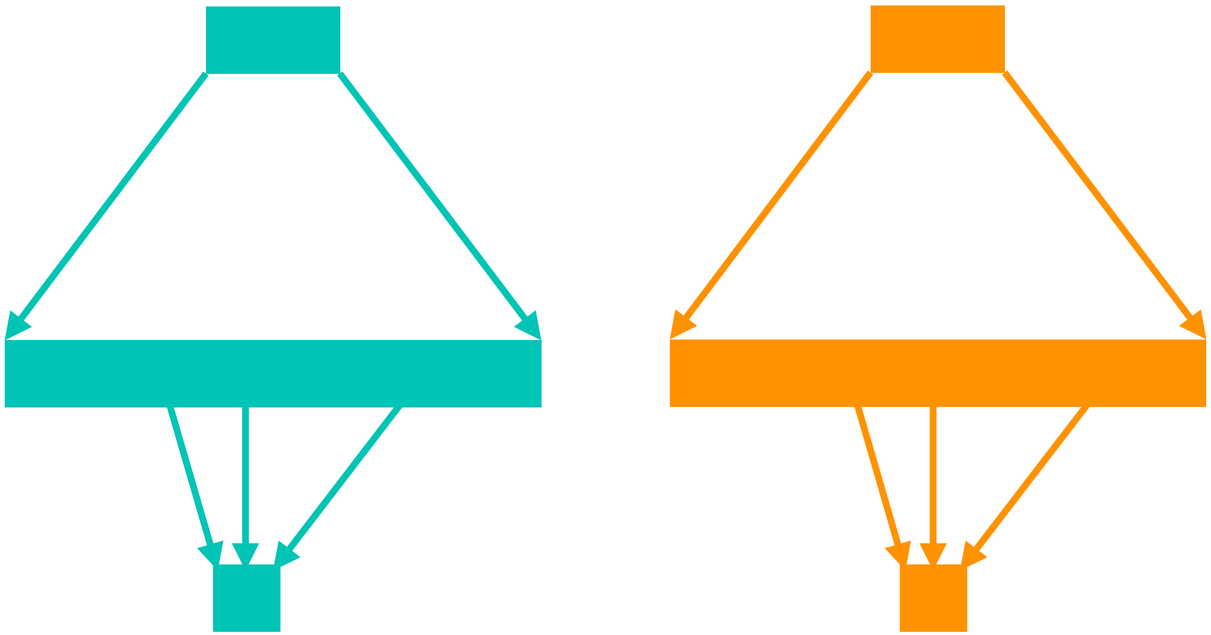
- PCG $=$ (Gen, Offline, Online)
- $(k_A^{\text{off}}, k_B^{\text{off}}, k_A^{\text{on}}, k_B^{\text{on}}) \leftarrow \text{Gen}(1^\lambda)$
- **Low storage:** $|Y_\sigma| \leq N$
- **Output locality:** Online reads $\leq \ell$ entries of Y_σ
- Analogous correctness, pseudorandomness & security

OUR (MAIN) CONTRIBUTION



OUR (MAIN) CONTRIBUTION

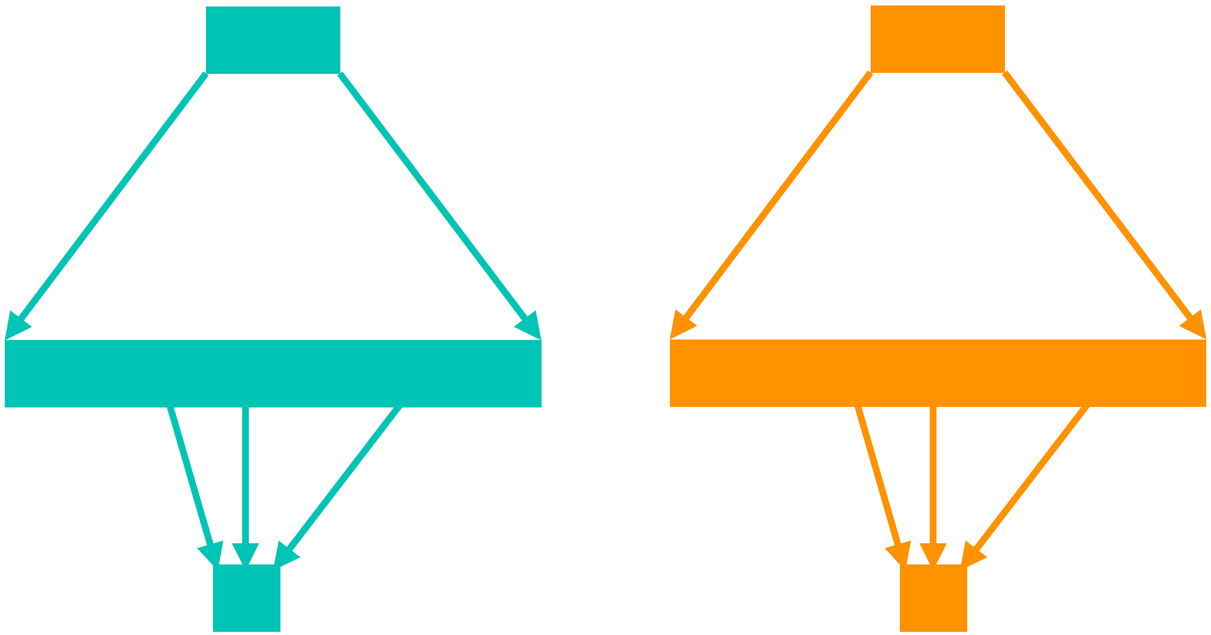
Offline-online PCG's from Expand-Accumulate Codes



OUR (MAIN) CONTRIBUTION

New class of codes!

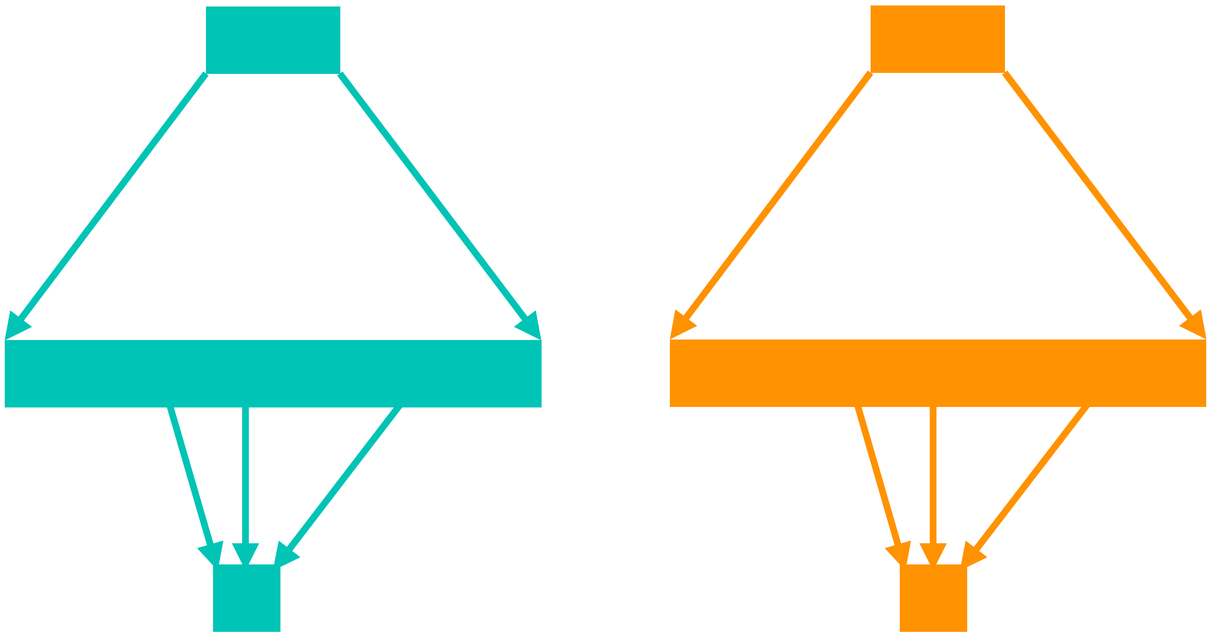
Offline-online PCG's from Expand-Accumulate Codes



OUR (MAIN) CONTRIBUTION

New class of codes!

Offline-online PCG's from Expand-Accumulate Codes



Offline Phase

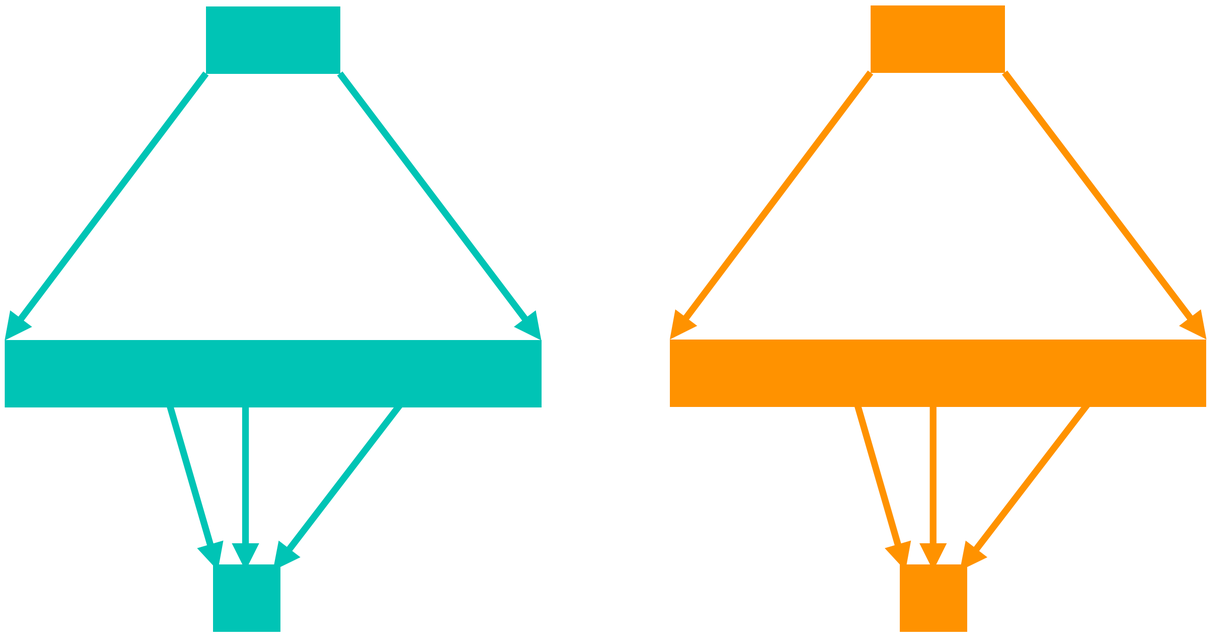
Highly parallelizable

Cache friendly

OUR (MAIN) CONTRIBUTION

New class of codes!

Offline-online PCG's from Expand-Accumulate Codes



Offline Phase

Highly parallelizable

Cache friendly

Online Phase

Low output locality

HOW TO CONSTRUCT EFFICIENT PCGS?

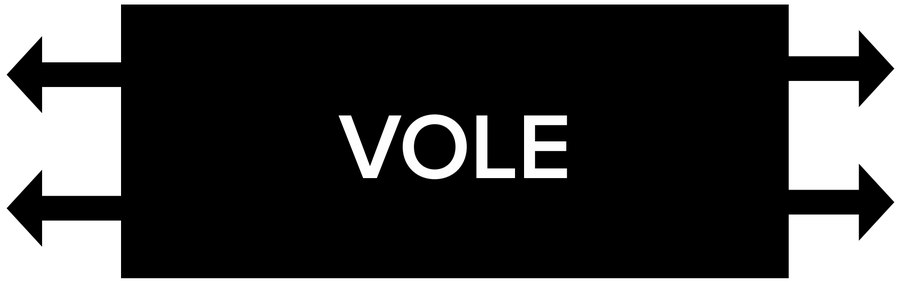
RECIPE FOR PCGS

RECIPE FOR PCGS

- Goal: construct PCG for *vector* OLE (VOLE) correlation
  - $(\vec{a}, \vec{c}_0) \in \mathbb{F}^N \times \mathbb{F}^N$ and $(b, \vec{c}_1) \in \mathbb{F} \times \mathbb{F}^N$ s.t. $b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$

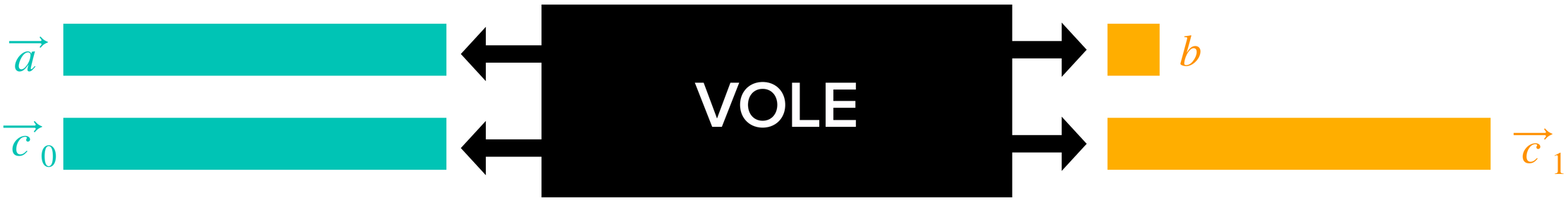
RECIPE FOR PCGS

- Goal: construct PCG for *vector* OLE (VOLE) correlation
  - $(\vec{a}, \vec{c}_0) \in \mathbb{F}^N \times \mathbb{F}^N$ and $(b, \vec{c}_1) \in \mathbb{F} \times \mathbb{F}^N$ s.t. $b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$



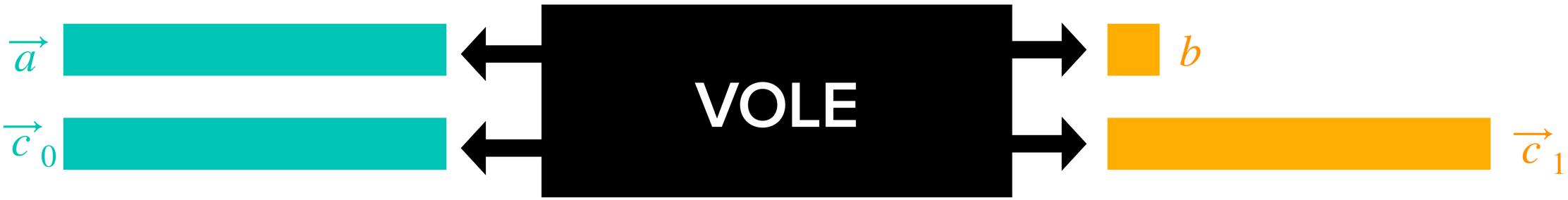
RECIPE FOR PCGS

- Goal: construct PCG for *vector* OLE (VOLE) correlation
  - $(\vec{a}, \vec{c}_0) \in \mathbb{F}^N \times \mathbb{F}^N$ and $(b, \vec{c}_1) \in \mathbb{F} \times \mathbb{F}^N$ s.t. $b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$



RECIPE FOR PCGS

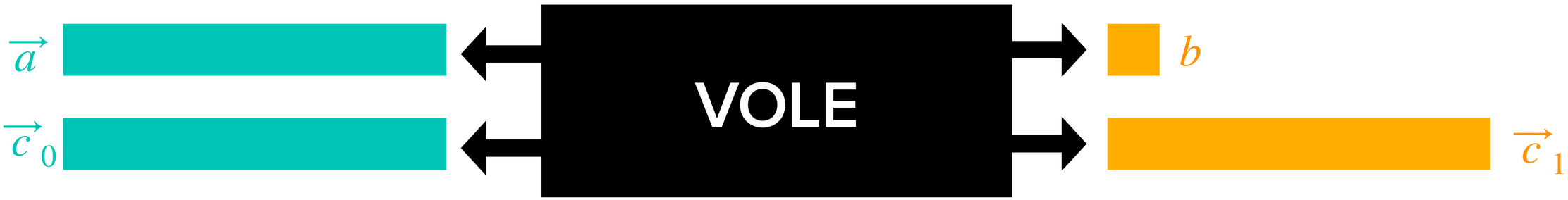
- Goal: construct PCG for *vector* OLE (VOLE) correlation
  - $(\vec{a}, \vec{c}_0) \in \mathbb{F}^N \times \mathbb{F}^N$ and $(b, \vec{c}_1) \in \mathbb{F} \times \mathbb{F}^N$ s.t. $b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$



- Consider function $F : [N] \to \mathbb{F}$ defined by $F(x) = b \cdot a_x$

RECIPE FOR PCGS

- Goal: construct PCG for *vector* OLE (VOLE) correlation
  - $(\vec{a}, \vec{c}_0) \in \mathbb{F}^N \times \mathbb{F}^N$ and $(b, \vec{c}_1) \in \mathbb{F} \times \mathbb{F}^N$ s.t. $b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$



- Consider function $F : [N] \to \mathbb{F}$ defined by $F(x) = b \cdot a_x$

- Idea: additively share F between Alice and Bob

(ADDITIVE) FUNCTION SECRET-SHARING (FSS)

(ADDITIVE) FUNCTION SECRET-SHARING (FSS)

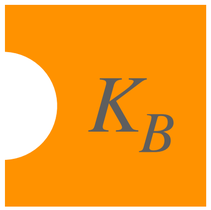
- Share function F s.t. $\forall$ inputs x

(ADDITIVE) FUNCTION SECRET-SHARING (FSS)

- Share function F s.t. $\forall$ inputs x

(ADDITIVE) FUNCTION SECRET-SHARING (FSS)

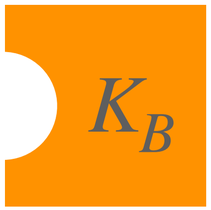
- Share function F s.t. $\forall$ inputs x



$$\text{Eval}(K_A, x) \quad + \quad \text{Eval}(K_B, x) \quad = \quad F(x)$$

(ADDITIVE) FUNCTION SECRET-SHARING (FSS)

- Share function F s.t. $\forall$ inputs x

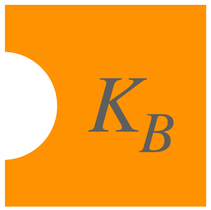


$$\text{Eval}(K_A, x) \quad + \quad \text{Eval}(K_B, x) \quad = \quad F(x)$$

- Goal: $|K_A|$, $|K_B|$ small (secret-sharing truth-table too expensive!)

(ADDITIVE) FUNCTION SECRET-SHARING (FSS)

- Share function F s.t. $\forall$ inputs x



$$\text{Eval}(K_A, x) \quad + \quad \text{Eval}(K_B, x) \quad = \quad F(x)$$

- Goal: $|K_A|$, $|K_B|$ small (secret-sharing truth-table too expensive!)
- Efficient FSS for point functions [Gilboalshai'14], or small sums

$$F_y^{\alpha}(x) = \begin{cases} \alpha & \text{if } x = y \\ 0 & \text{else} \end{cases} \qquad F_{y_1,\ldots,y_t}^{\alpha_1,\ldots,\alpha_t}(x) = \begin{cases} \alpha_i & \text{if } x = y_i \\ 0 & \text{else} \end{cases} \qquad t \text{ small}$$

(ADDITIVE) FUNCTION SECRET-SHARING (FSS)

- Share function F s.t. $\forall$ inputs x

$$\text{Eval}(K_A, x) \quad + \quad \text{Eval}(K_B, x) \quad = \quad F(x)$$

- Goal: $|K_A|$, $|K_B|$ small (secret-sharing truth-table too expensive!)
- Efficient FSS for point functions [Gilboalshai'14], or small sums

$$F_y^{\alpha}(x) = \begin{cases} \alpha & \text{if } x = y \\ 0 & \text{else} \end{cases} \qquad F_{y_1,\ldots,y_t}^{\alpha_1,\ldots,\alpha_t}(x) = \begin{cases} \alpha_i & \text{if } x = y_i \\ 0 & \text{else} \end{cases} \qquad t \text{ small}$$

- Can efficiently share $F(x) = b \cdot a_x$ if $\vec{a}$ is sparse

SPARSE VOLE

[BoyleCouteauGilboalshai'18]

SPARSE VOLE

[BoyleCouteauGilboalshai'18]

SPARSE VOLE

$b \cdot \vec{e} = \vec{c}'_0 + \vec{c}'_1$ [BoyleCouteauGilboalshai'18]

SPARSE VOLE

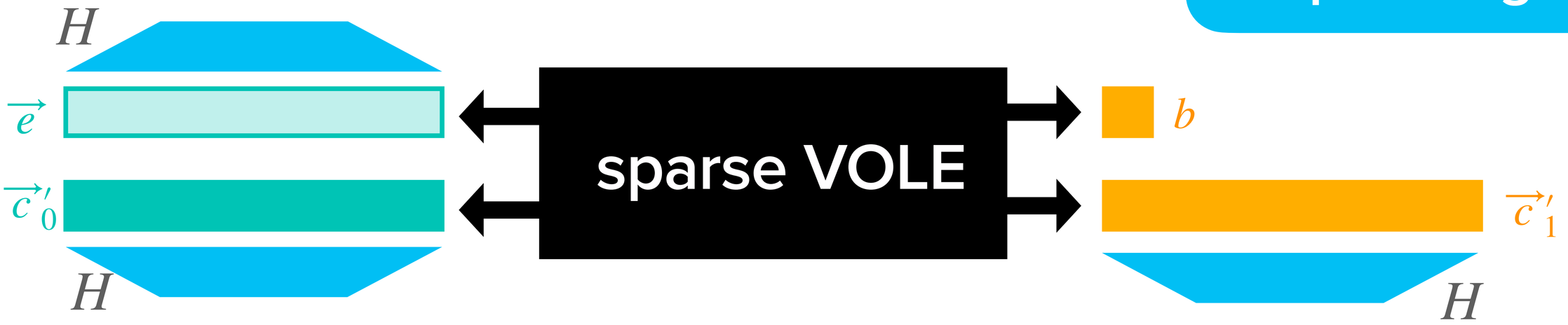
$b \cdot \vec{e} = \vec{c}'_0 + \vec{c}'_1$ [BoyleCouteauGilboalshai'18]

H linear compressing map

SPARSE VOLE

$b \cdot \vec{e} = \vec{c}'_0 + \vec{c}'_1$ [BoyleCouteauGilboalshai'18]

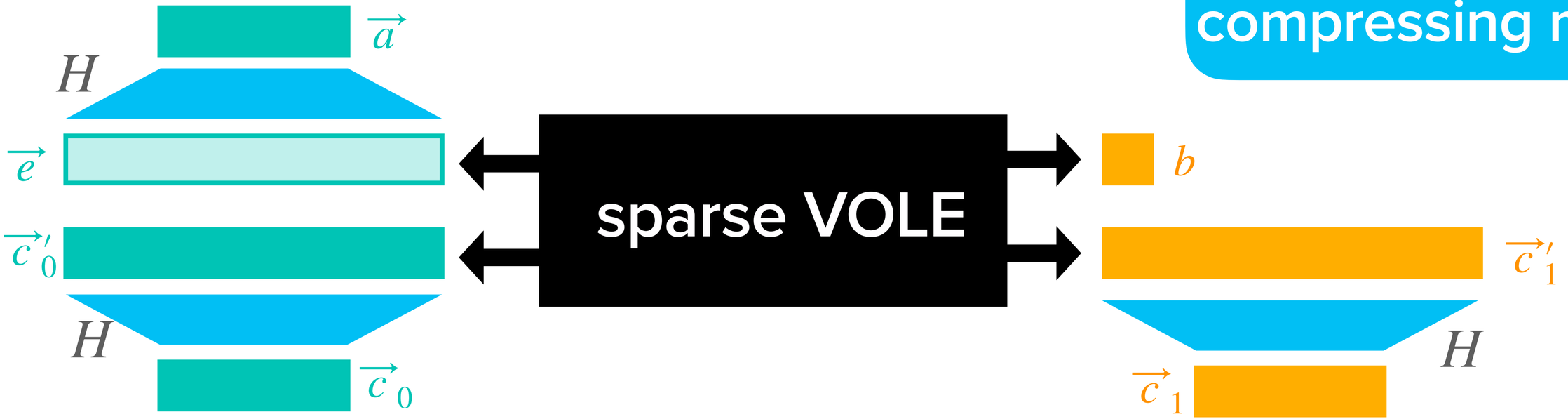
H linear compressing map



SPARSE VOLE

$b \cdot \vec{e} = \vec{c}'_0 + \vec{c}'_1$ [BoyleCouteauGilboalshai'18]

H linear compressing map



SPARSE VOLE

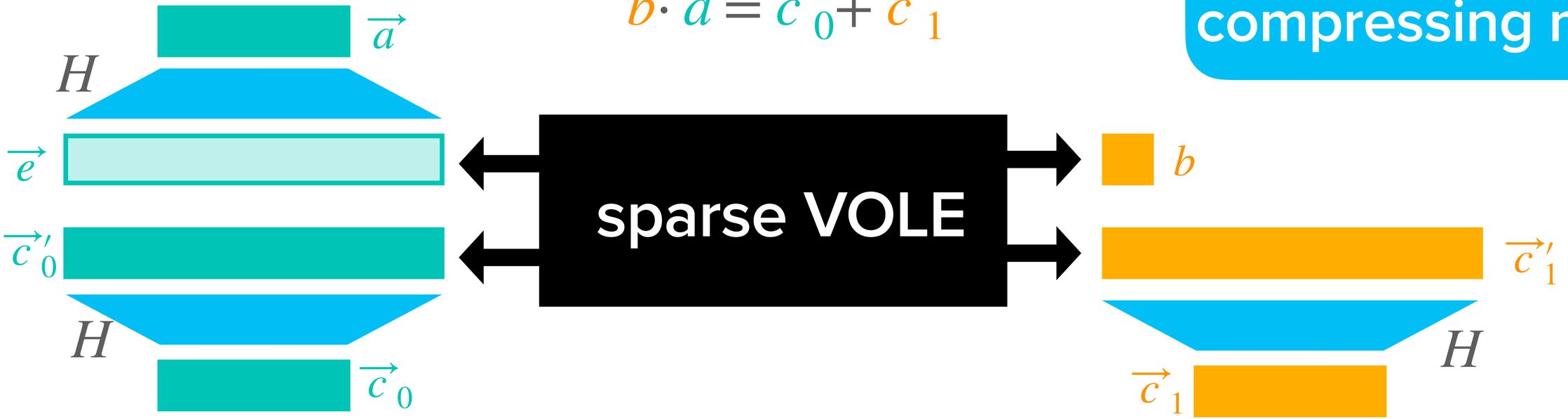
$$b \cdot \vec{e} = \vec{c}'_0 + \vec{c}'_1$$

$$\Downarrow$$

$$b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$$

[BoyleCouteauGilboalshai'18]

H linear compressing map



SPARSE VOLE

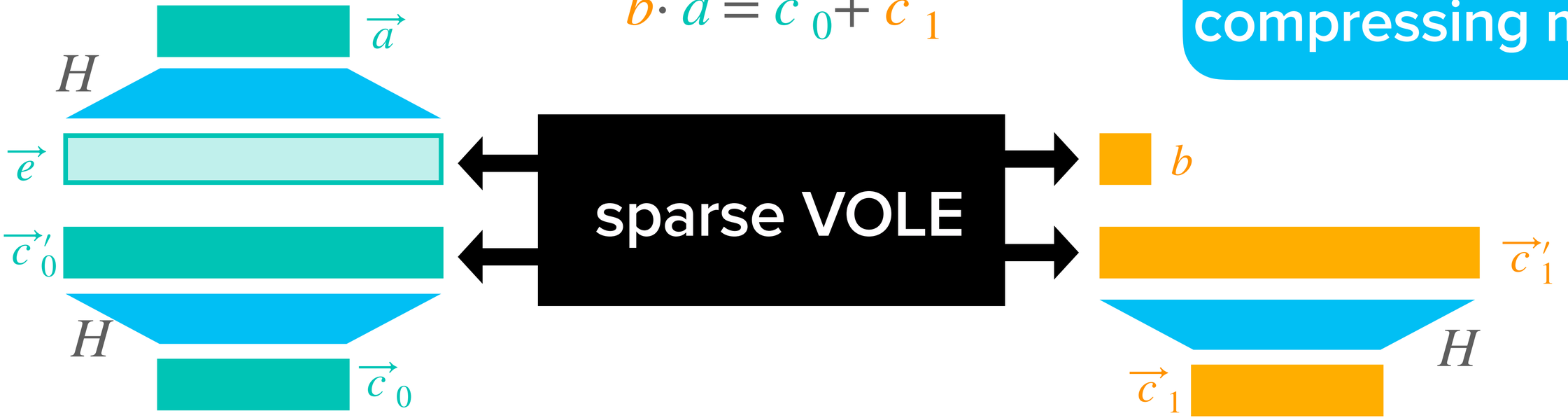
$$b \cdot \vec{e} = \vec{c}'_0 + \vec{c}'_1$$

$$\Downarrow$$

$$b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$$

[BoyleCouteauGilboalshai'18]

H linear compressing map



How to choose H?
Need $\vec{a} \approx_c$ Unif

SPARSE VOLE

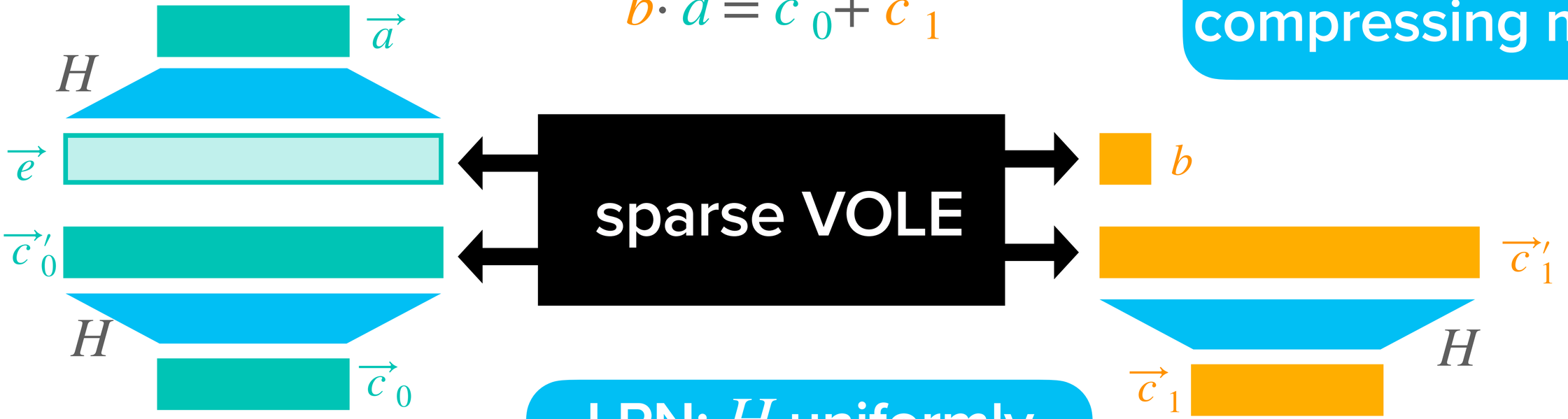
$$b \cdot \vec{e} = \vec{c}'_0 + \vec{c}'_1$$

$$\Downarrow$$

$$b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$$

[BoyleCouteauGilboalshai'18]

H linear compressing map



LPN: H uniformly random works!

How to choose H?
Need $\vec{a} \approx_c$ Unif

SPARSE VOLE

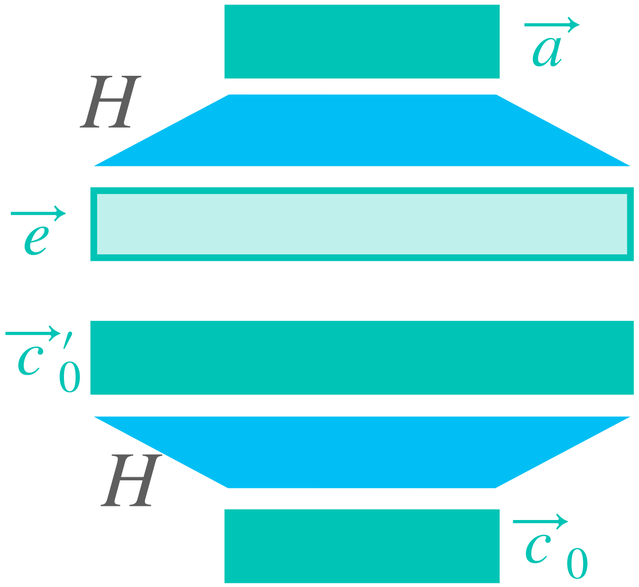
$$b \cdot \vec{e} = \vec{c}'_0 + \vec{c}'_1$$

$$\Downarrow$$

$$b \cdot \vec{a} = \vec{c}_0 + \vec{c}'_1$$

[BoyleCouteauGilboalshai'18]

H linear compressing map



sparse VOLE

How to choose H?
Need $\vec{a} \approx_c$ Unif

LPN: H uniformly random works!

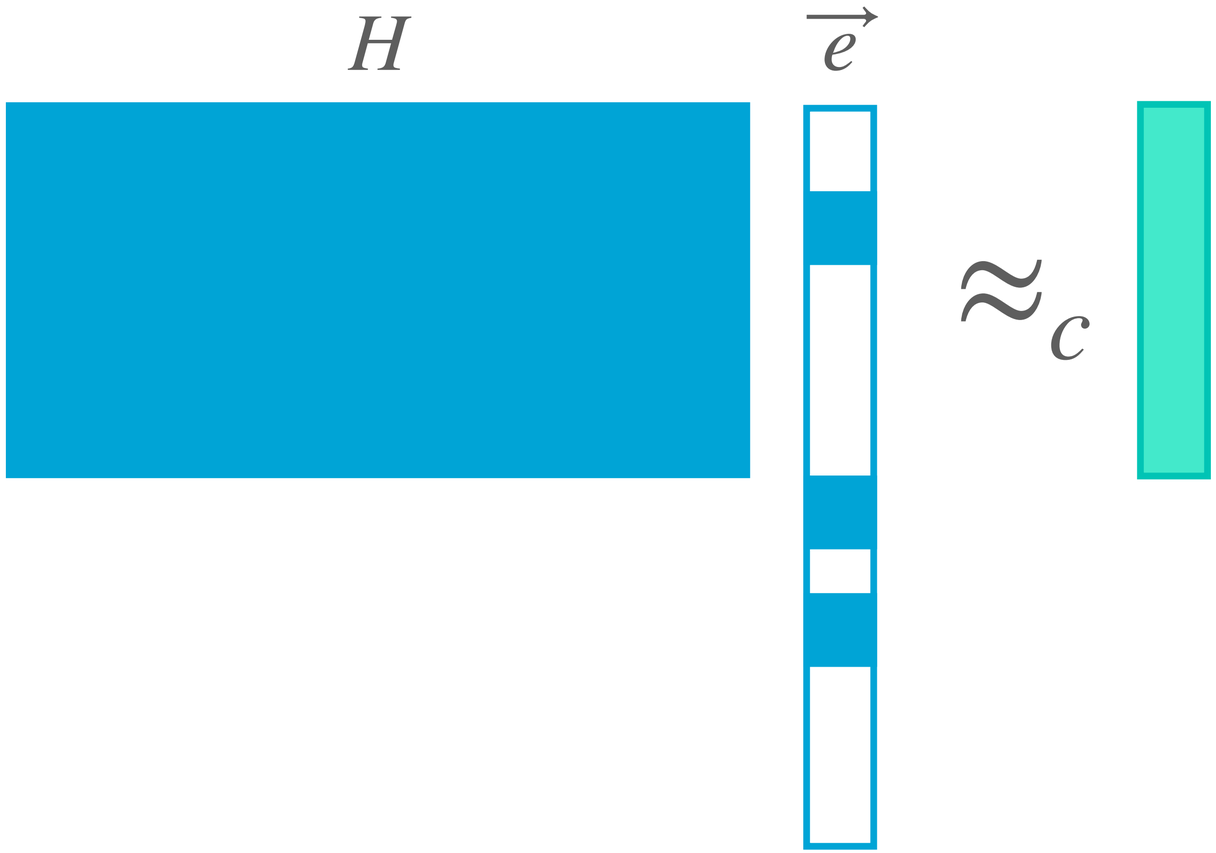
Better efficiency?

[BoyleCouteauGilboalshaiKohlScholl'19, '20, CouteauRindalRaghuraman'21]

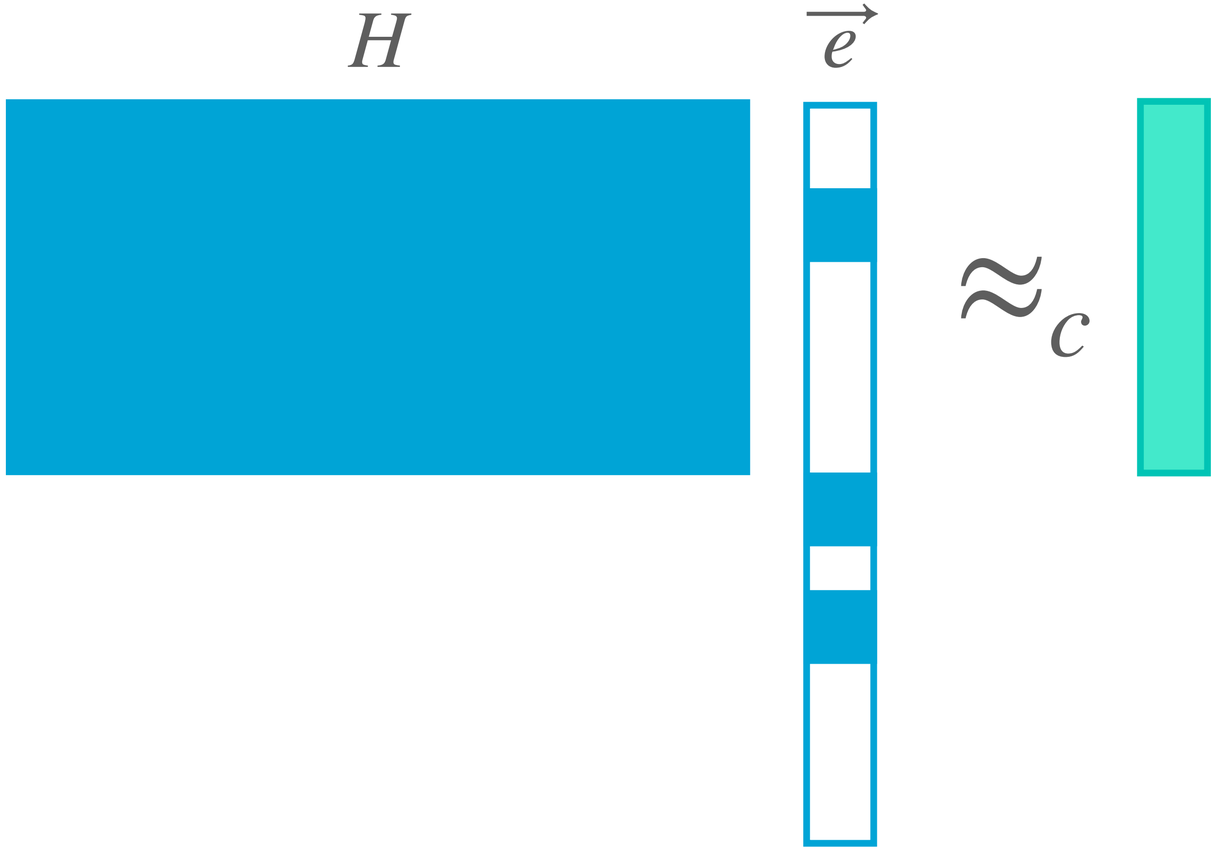
EXPAND-ACCUMULATE CODES

HOW TO CHOOSE H?

HOW TO CHOOSE H?

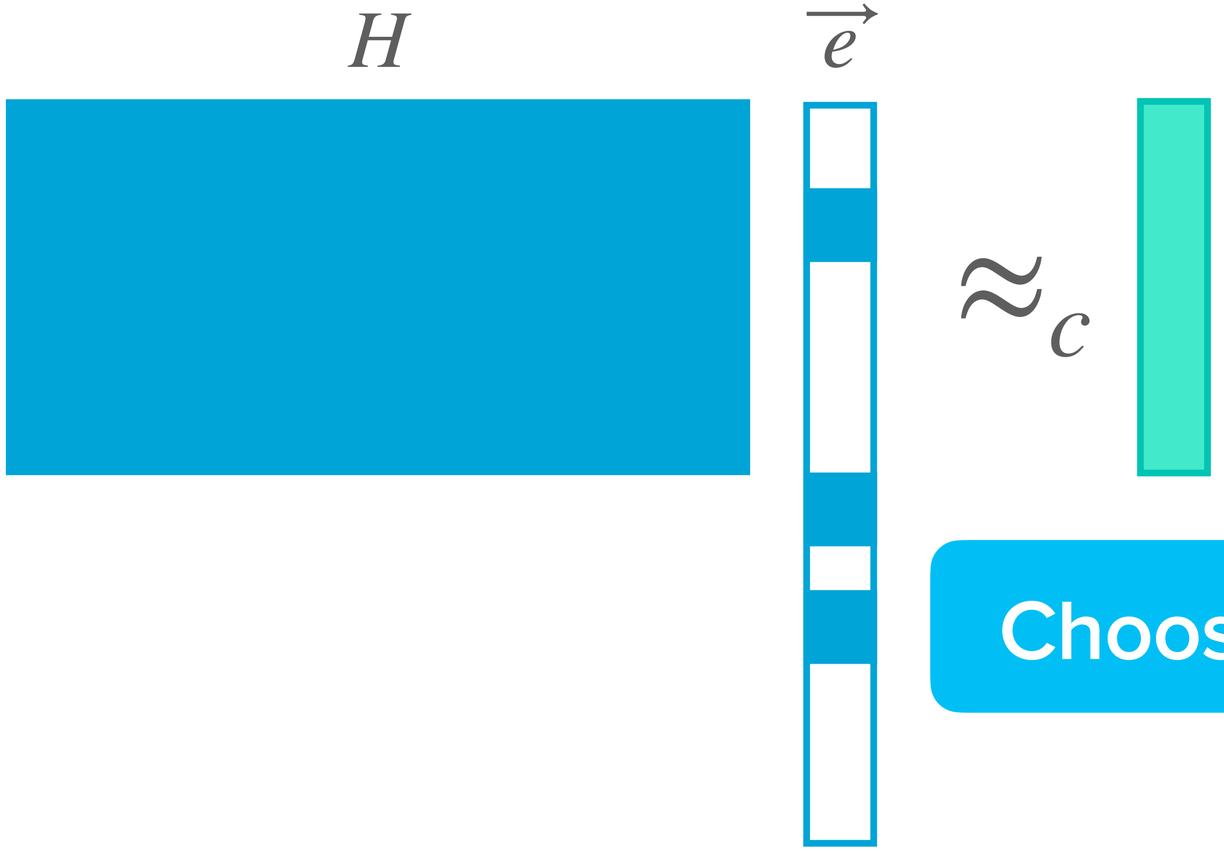


HOW TO CHOOSE H?



Want: $H \cdot \vec{e}$ efficiently computable

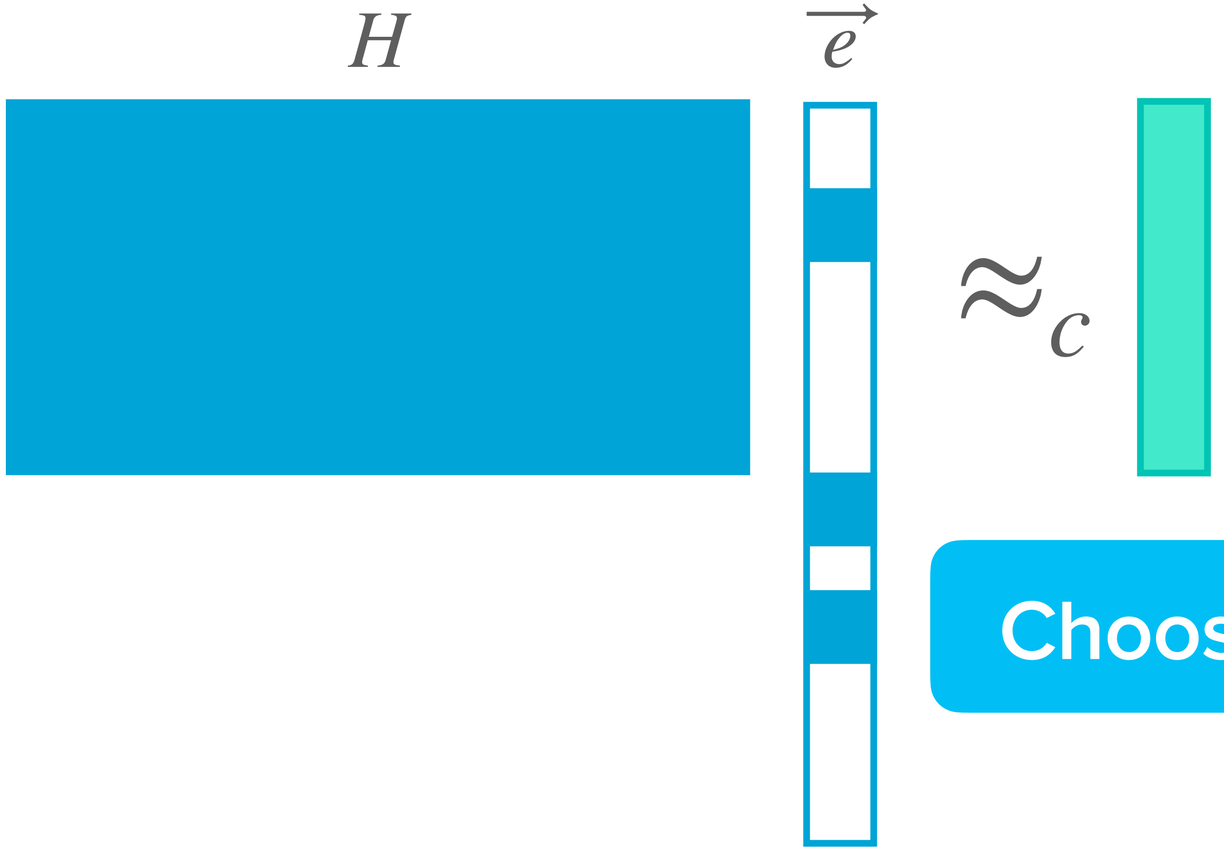
HOW TO CHOOSE H?



Want: $H \cdot \vec{e}$ efficiently computable

Choose H sparse?

HOW TO CHOOSE H?



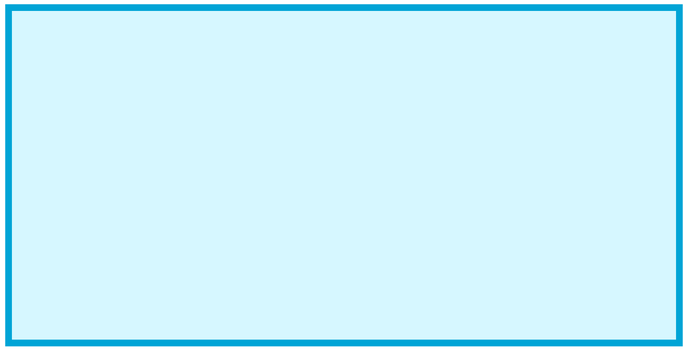
Want: $H \cdot \vec{e}$ efficiently computable

Choose H sparse?

Problem: $H \cdot \vec{e}$ also sparse!

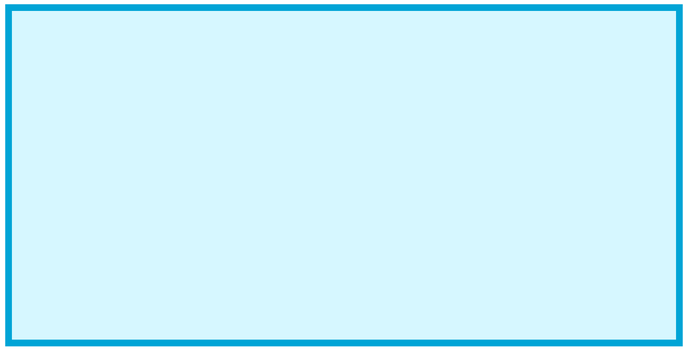
EXPAND-ACCUMULATE (EA) CODES

EXPAND-ACCUMULATE (EA) CODES

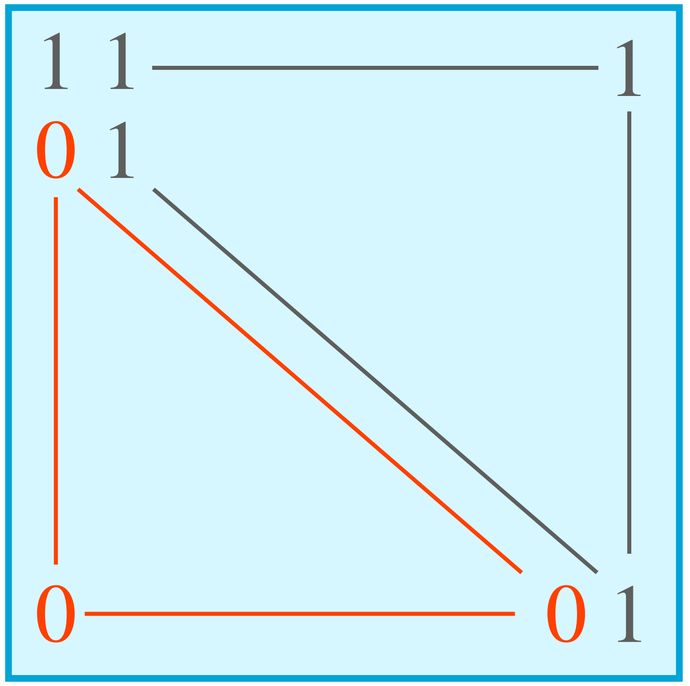


sparse matrix
(sample randomly)

EXPAND-ACCUMULATE (EA) CODES

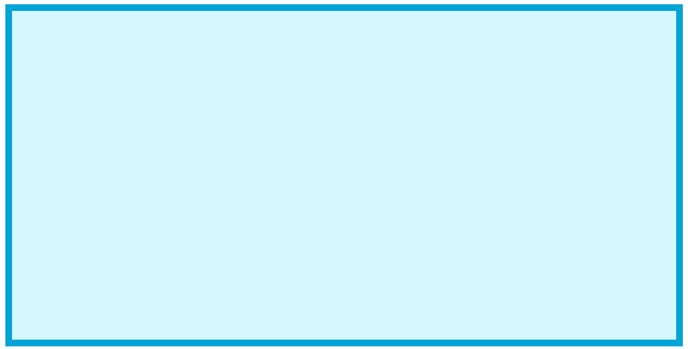


sparse matrix
(sample randomly)



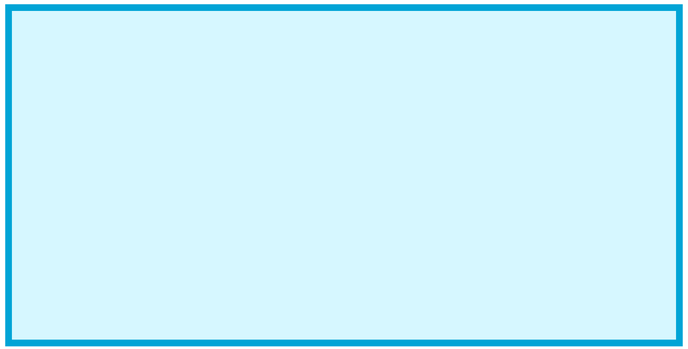
accumulator matrix A

EXPAND-ACCUMULATE (EA) CODES

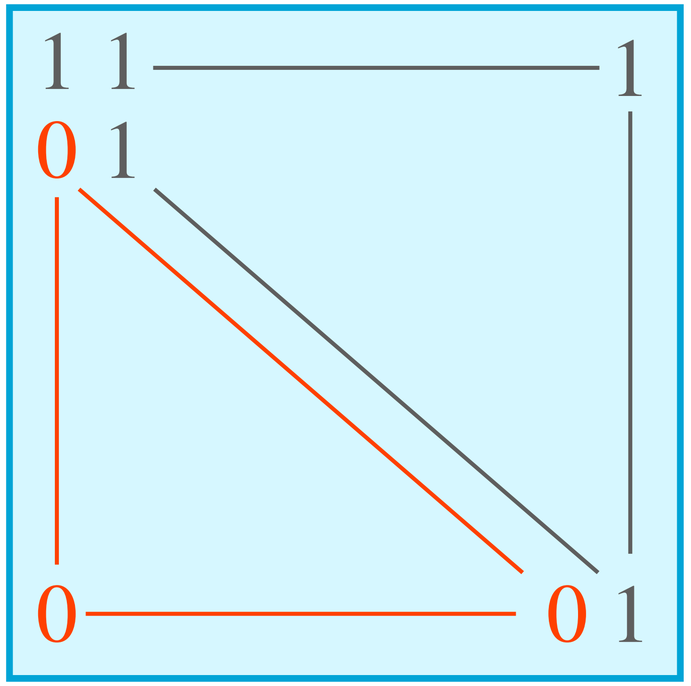


EA generator matrix
H

=

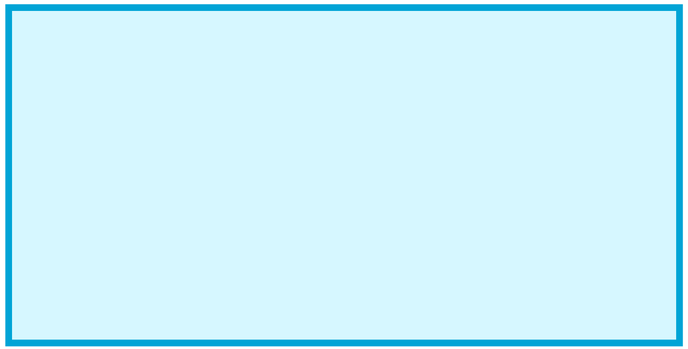


sparse matrix
(sample randomly)



accumulator matrix A

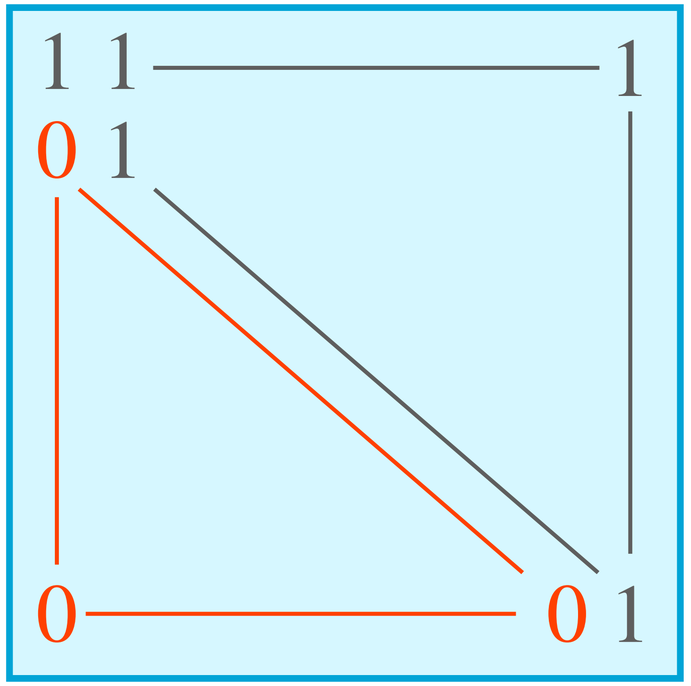
EXPAND-ACCUMULATE (EA) CODES



EA generator matrix
H

=

sparse matrix
(sample randomly)



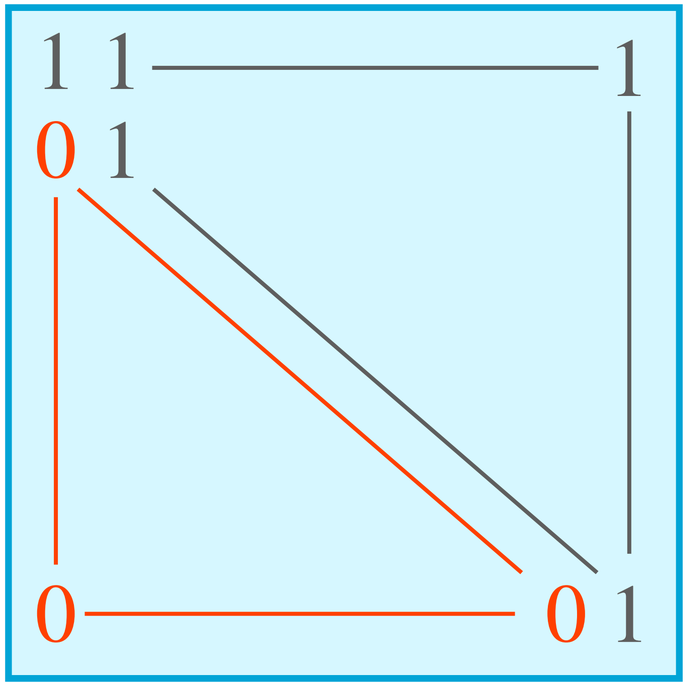
accumulator matrix A

EXPAND-ACCUMULATE (EA) CODES

EA generator matrix
H

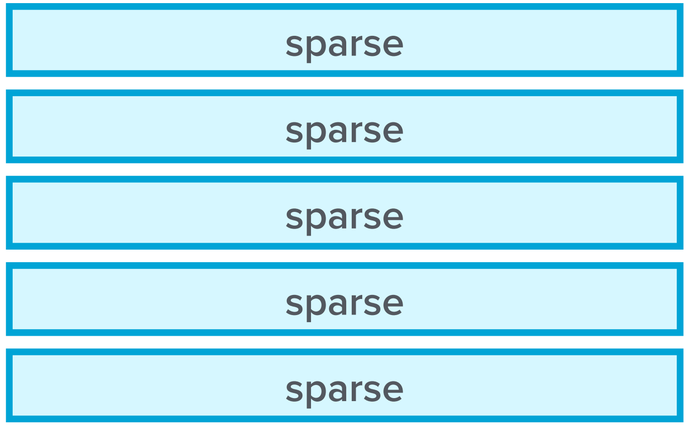
=

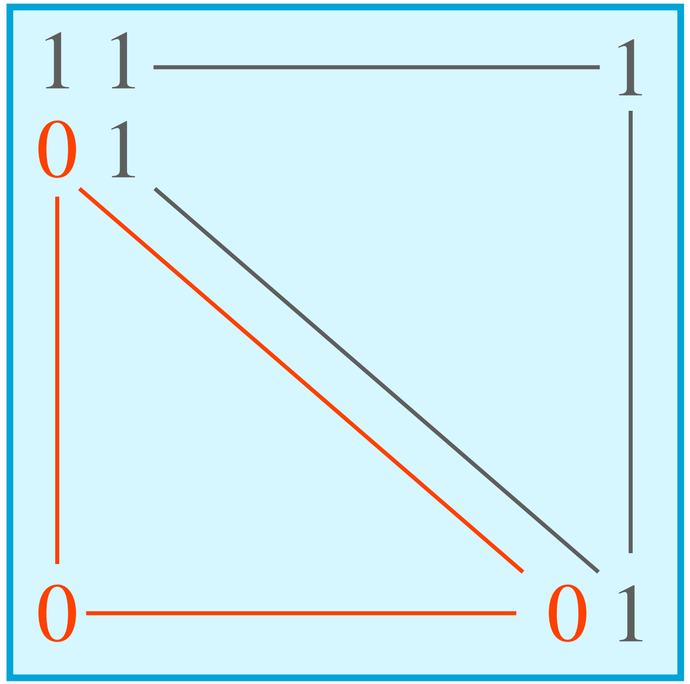
sparse matrix
(sample randomly)

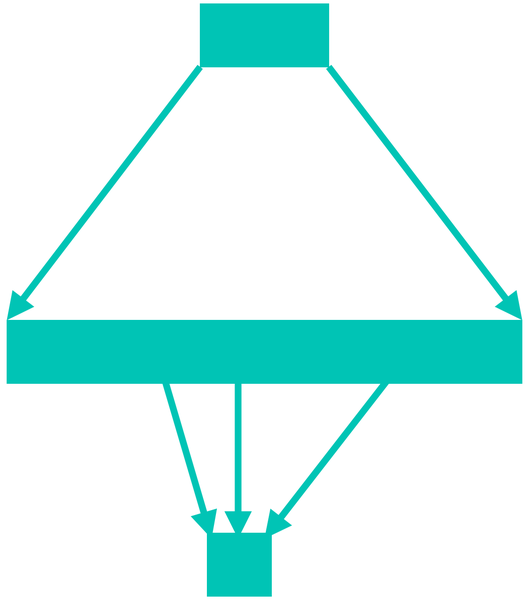


accumulator matrix A

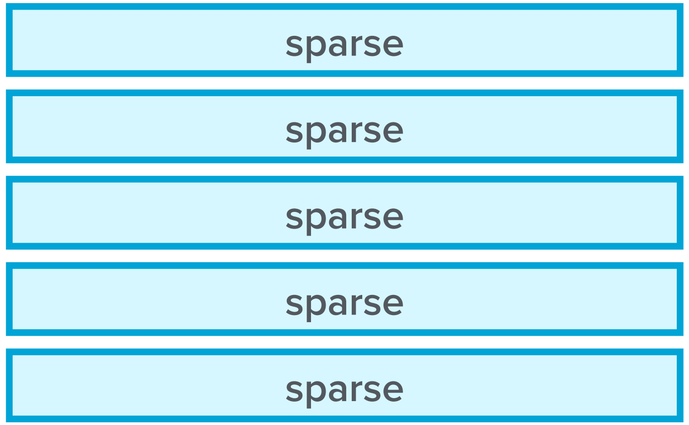
OFFLINE-ONLINE PCG FROM EA CODES

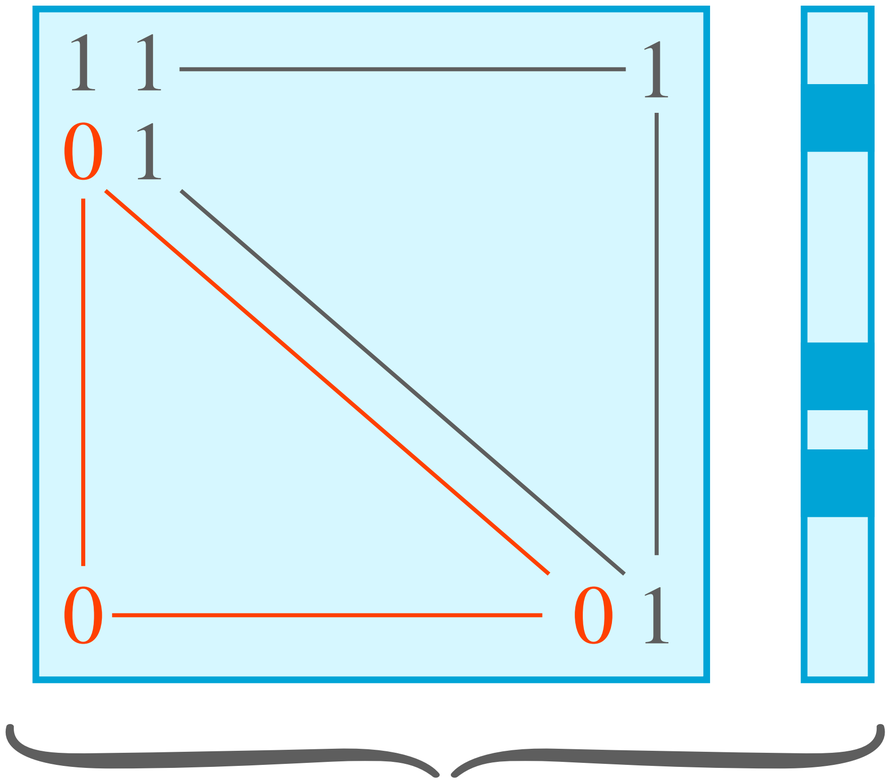


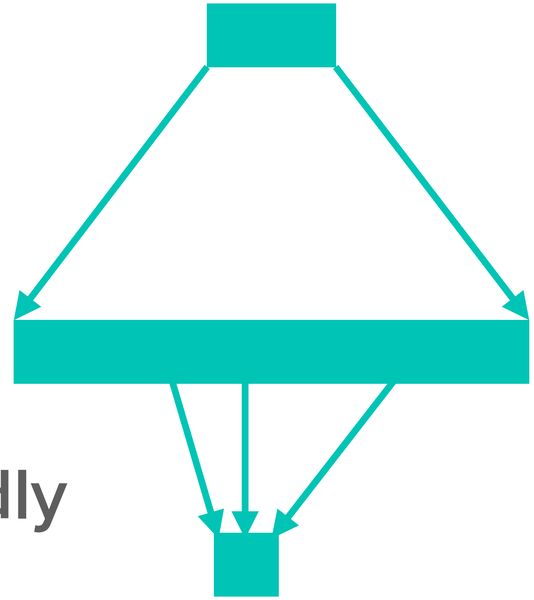




OFFLINE-ONLINE PCG FROM EA CODES

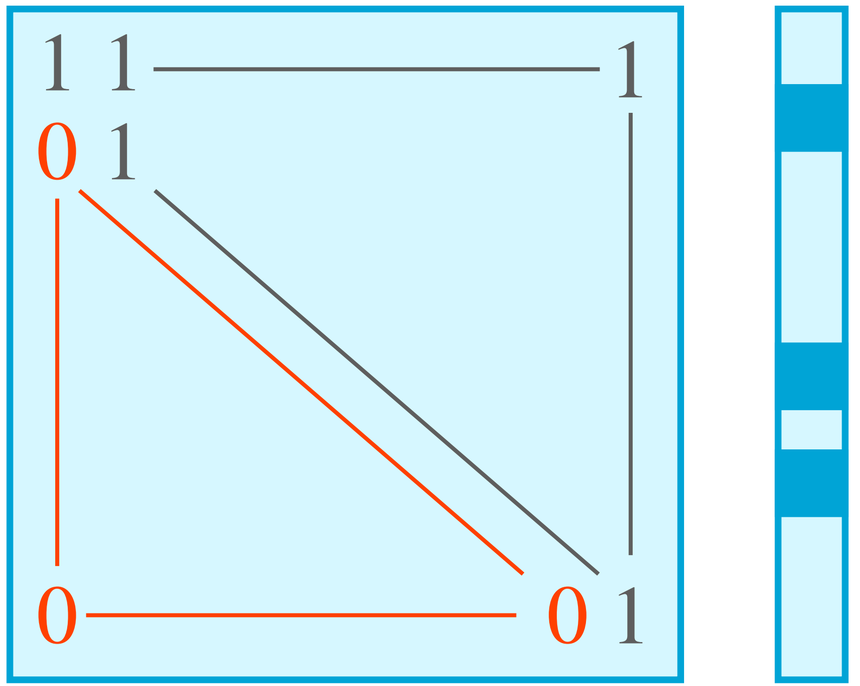


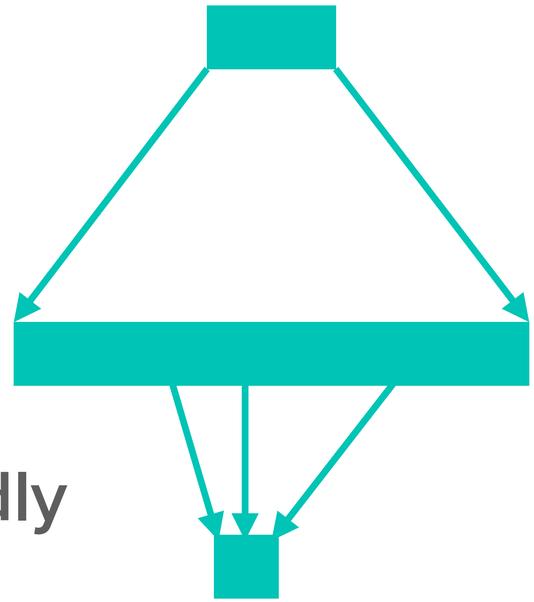




Offline: Parallelizable & cache-friendly

OFFLINE-ONLINE PCG FROM EA CODES

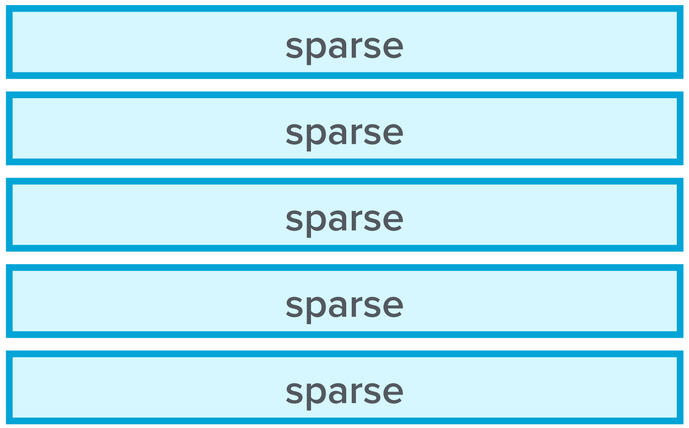


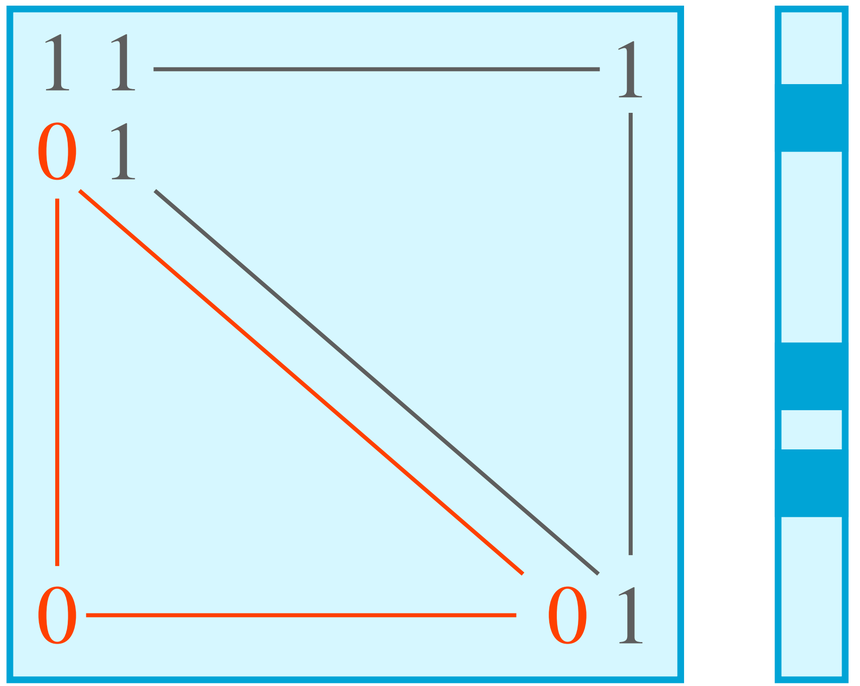


Offline: Parallelizable & cache-friendly

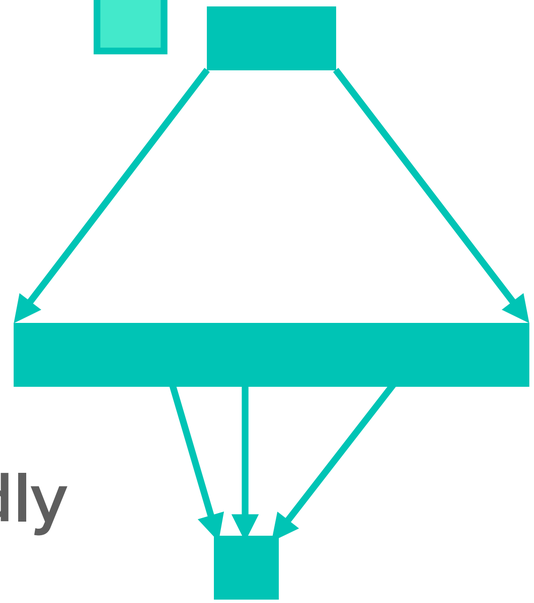
Online: Low locality

OFFLINE-ONLINE PCG FROM EA CODES





$\approx_c$



Offline: Parallelizable & cache-friendly

Online: Low locality

SECURITY?

SECURITY?

- Why should we believe $H \cdot \overrightarrow{e}$ pseudorandom?

SECURITY?

- Why should we believe $H \cdot \overrightarrow{e}$ pseudorandom?

Ideally:

Reduce to known hard problem (LPN)

SECURITY?

- Why should we believe $H \cdot \overrightarrow{e}$ pseudorandom?

Ideally:

Reduce to known hard problem (LPN)

Alternatively:

Rule out known attacks

SECURITY?

- Why should we believe $H \cdot \overrightarrow{e}$ pseudorandom?

Ideally:	**Alternatively:**
Reduce to known hard problem (LPN)	Rule out known attacks

- Almost all* attacks boil down to following strategy:

*relevant to our parameters which don't exploit algebraic structure

SECURITY?

- Why should we believe $H \cdot \overrightarrow{e}$ pseudorandom?

Ideally:	**Alternatively:**
Reduce to known hard problem (LPN)	Rule out known attacks

- Almost all* attacks boil down to following strategy:
  - Look at H and find vector $\overrightarrow{x}^\top$ s.t. $\overrightarrow{x}^\top H$ is sparse

*relevant to our parameters which don't exploit algebraic structure

SECURITY?

- Why should we believe $H \cdot \vec{e}$ pseudorandom?

Ideally:	**Alternatively:**
Reduce to known hard problem (LPN)	Rule out known attacks

- Almost all* attacks boil down to following strategy:
  - Look at H and find vector $\vec{x}^\top$ s.t. $\vec{x}^\top H$ is sparse
  - Compute $\vec{x}^\top \vec{y}$, where $\vec{y}$ is (i) $H \cdot \vec{e}$ or (ii) unif. rand.

*relevant to our parameters which don't exploit algebraic structure

SECURITY?

- Why should we believe $H \cdot \vec{e}$ pseudorandom?

Ideally:
Reduce to known hard problem (LPN)

Alternatively:
Rule out known attacks

- Almost all* attacks boil down to following strategy:
  - Look at H and find vector $\vec{x}^\top$ s.t. $\vec{x}^\top H$ is sparse
  - Compute $\vec{x}^\top \vec{y}$, where $\vec{y}$ is (i) $H \cdot \vec{e}$ or (ii) unif. rand.

Linear tests framework

*relevant to our parameters which don't exploit algebraic structure

SECURITY

Resistance to linear tests

SECURITY

Resistance to linear tests

$\Longleftarrow$

No low-weight (non-zero) vector in code $\{\overrightarrow{x}^{\top} H\}$ $\Longleftrightarrow$ good min. dist.

SECURITY

Resistance to linear tests

$\Longleftarrow$

No low-weight (non-zero) vector in code $\{\overrightarrow{x}^{\top} H\}$ $\Longleftrightarrow$ good min. dist.

This work: rule out linear attacks

SECURITY

Resistance to linear tests $\Longleftarrow$ No low-weight (non-zero) vector in code $\{\overrightarrow{x}^{\top} H\}$ $\Longleftrightarrow$ good min. dist.

This work: rule out linear attacks

- For i.i.d. Bernoulli matrix with $p = O(\log N/N)$, get minimum distance $\Omega(N)$ with probability $1 - 1/N^{\Omega(1)}$

SECURITY

Resistance to linear tests $\Leftarrow$ No low-weight (non-zero) vector in code $\{\vec{x}^\top H\}$ $\Leftrightarrow$ good min. dist.

This work: rule out linear attacks

- For i.i.d. Bernoulli matrix with $p = O(\log^2 N/N)$, get minimum distance $\Omega(N)$ with probability $1 - 1/N^{\Omega(1)}$ $1 - 1/N^{\omega(1)}$

CONCRETE EFFICIENCY

CONCRETE EFFICIENCY

Offline Phase

Estimated ~100ms to generate Y_σ for 10 million OTs

Factor $\sim k$ speedup if k processors available

CONCRETE EFFICIENCY

Offline Phase

Estimated ~100ms to generate Y_σ for 10 million OTs

Factor $\sim k$ speedup if k processors available

Online Phase

CONCRETE EFFICIENCY

Offline Phase

Estimated ~100ms to generate Y_σ for 10 million OTs

Factor $\sim k$ speedup if k processors available

Online Phase

Theoretically

40 lookups + 1 hash per OT

CONCRETE EFFICIENCY

Offline Phase

Estimated ~100ms to generate Y_σ for 10 million OTs

Factor $\sim k$ speedup if k processors available

Online Phase

Theoretically

40 lookups + 1 hash per OT

Experimentally

7 lookups + 1 hash per OT

CONCRETE EFFICIENCY

Offline Phase

Estimated ~100ms to generate Y_σ for 10 million OTs

Factor $\sim k$ speedup if k processors available

Online Phase

Theoretically	Experimentally
40 lookups + 1 hash per OT	7 lookups + 1 hash per OT

Good pseudo-distance suffices

CONCRETE EFFICIENCY

Offline Phase

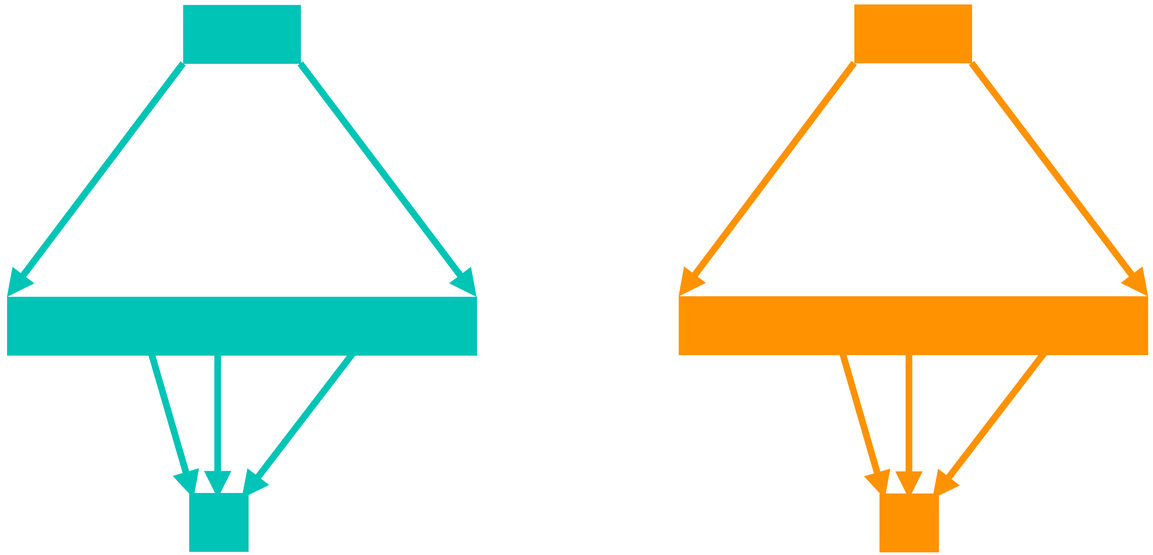
Estimated ~100ms to generate Y_σ for 10 million OTs

Factor $\sim k$ speedup if k processors available

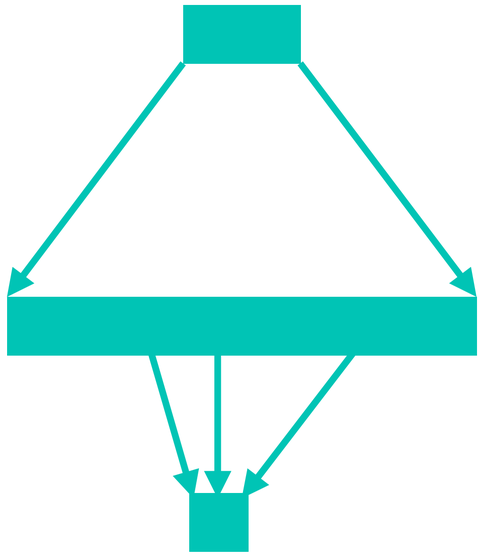
Online Phase

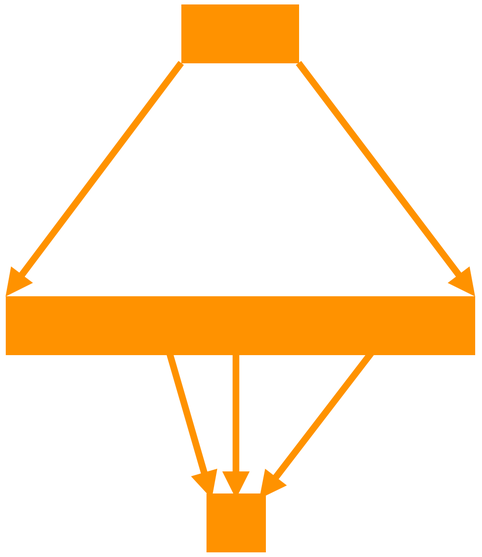
Theoretically	Experimentally
40 lookups + 1 hash per OT	7 lookups + 1 hash per OT
Good pseudo-distance suffices	Needs further cryptanalysis!

RECAP



RECAP



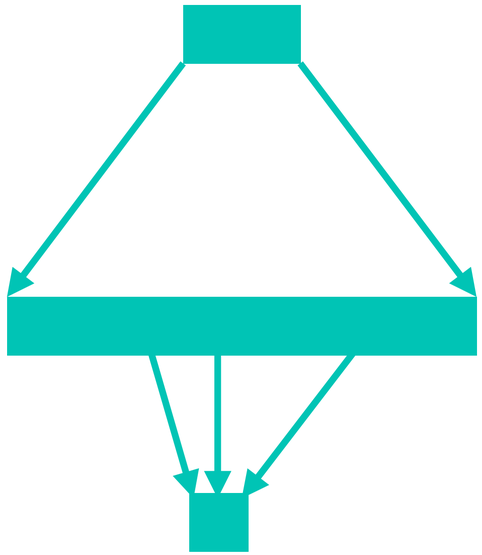


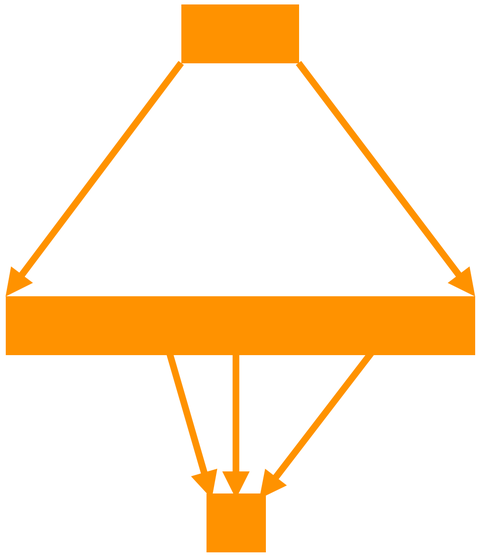
Offline Phase

Highly parallelizable

Cache friendly

RECAP





Offline Phase

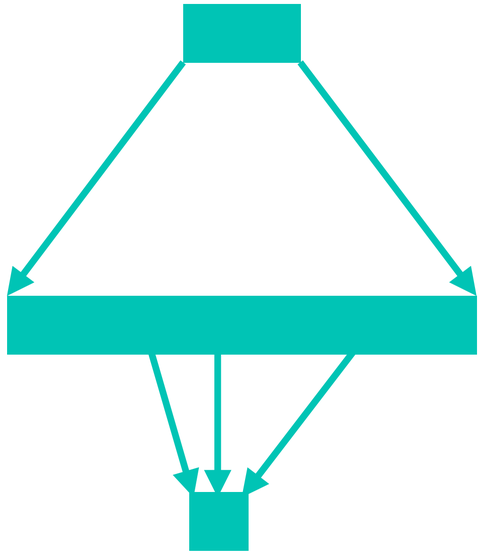
Highly parallelizable

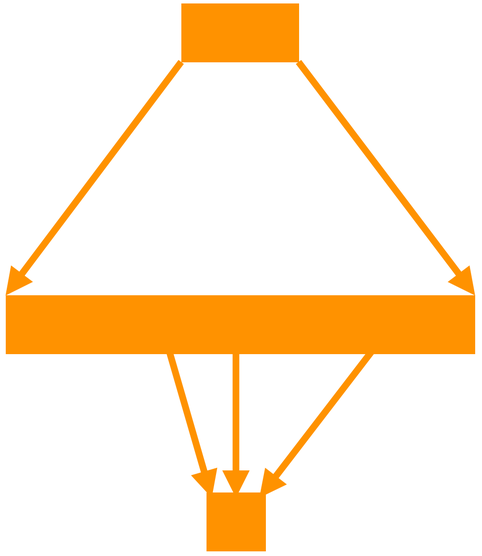
Cache friendly

Online Phase

Low output locality

RECAP





Offline Phase

Highly parallelizable

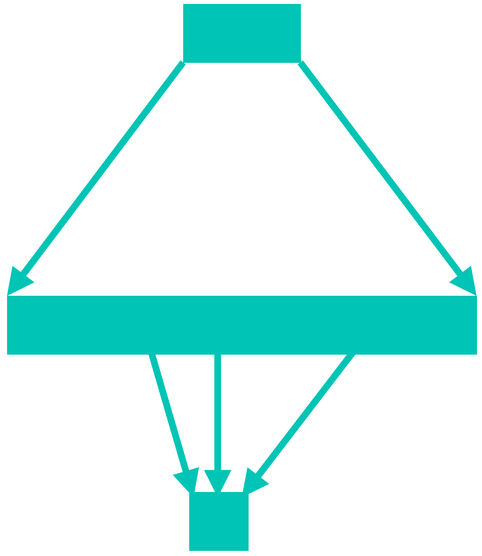
Cache friendly

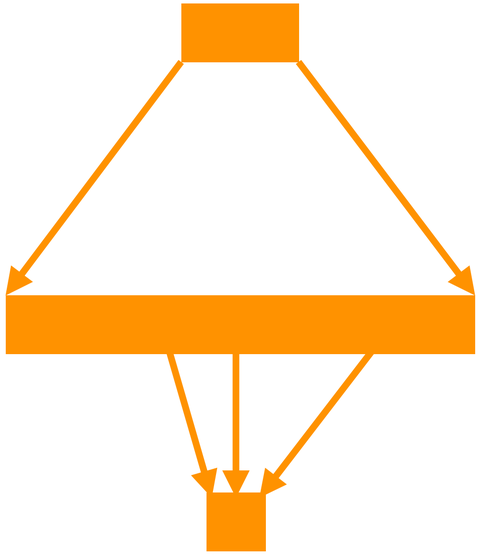
Online Phase

Low output locality

	Cache-friendly?	Parallelizable?
Silver [CouteauRindalRaghuaman'21]		
RAA Codes		
This work		

RECAP





Offline Phase

Highly parallelizable

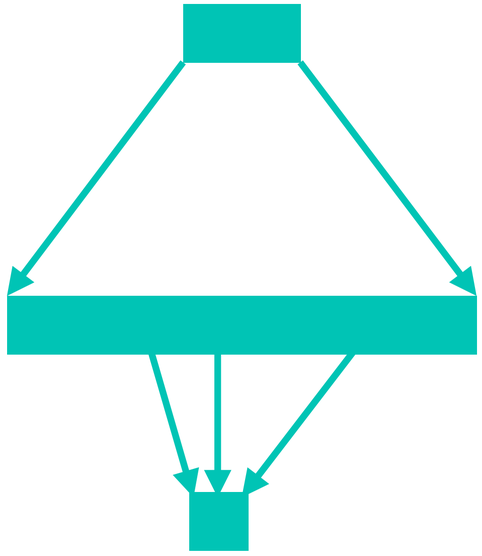
Cache friendly

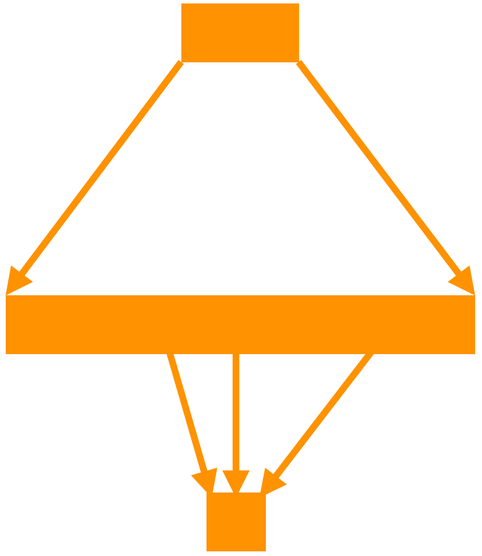
Online Phase

Low output locality

	Cache-friendly?	Parallelizable?
Silver [CouteauRindalRaghuaman'21]	✓	✗
RAA Codes		
This work		

RECAP





Offline Phase

Highly parallelizable

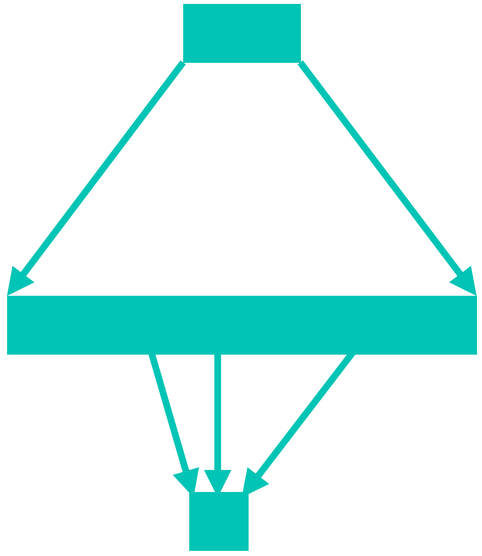
Cache friendly

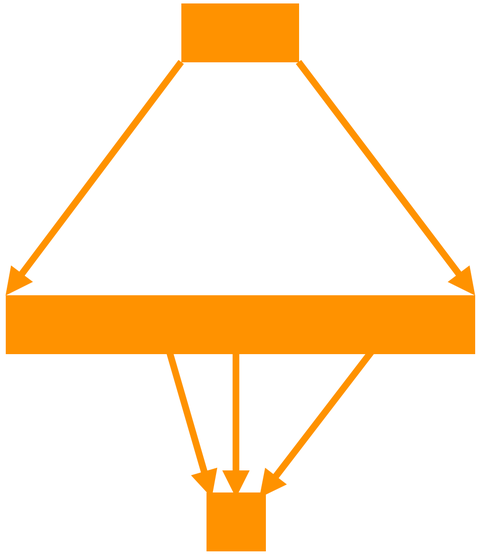
Online Phase

Low output locality

	Cache-friendly?	Parallelizable?
Silver [CouteauRindalRaghuaman'21]	✓	✗
RAA Codes	✗	✓
This work		

RECAP





Offline Phase

Highly parallelizable

Cache friendly

Online Phase

Low output locality

	Cache-friendly?	Parallelizable?
Silver [CouteauRindalRaghuaman'21]	✓	✗
RAA Codes	✗	✓
This work	✓	✓

FURTHER RESULTS & RECAP

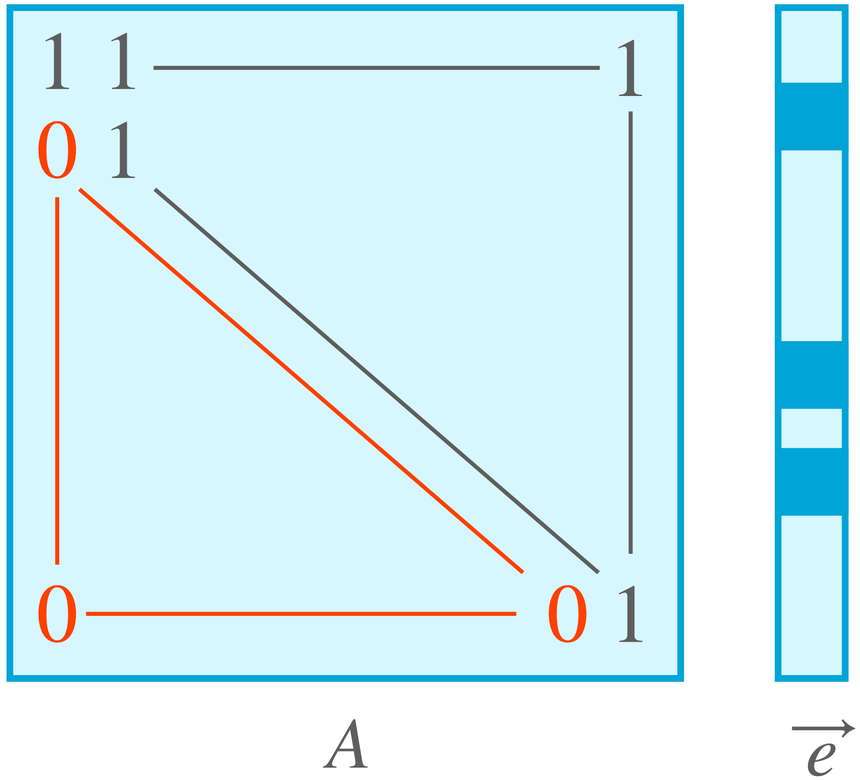
CONSTRUCTING PCFS

CONSTRUCTING PCFS

- Problem: if $|\overrightarrow{e}| = 2^{\lambda}, A \cdot \overrightarrow{e}$ too expensive to compute!

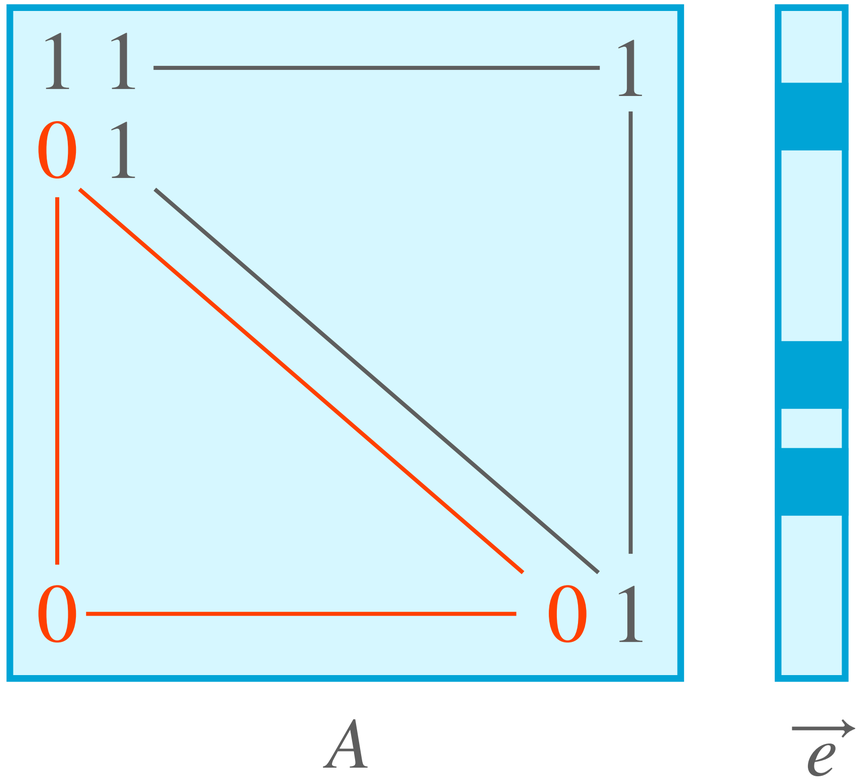
CONSTRUCTING PCFS

- Problem: if $|\overrightarrow{e}| = 2^{\lambda}, A \cdot \overrightarrow{e}$ too expensive to compute!



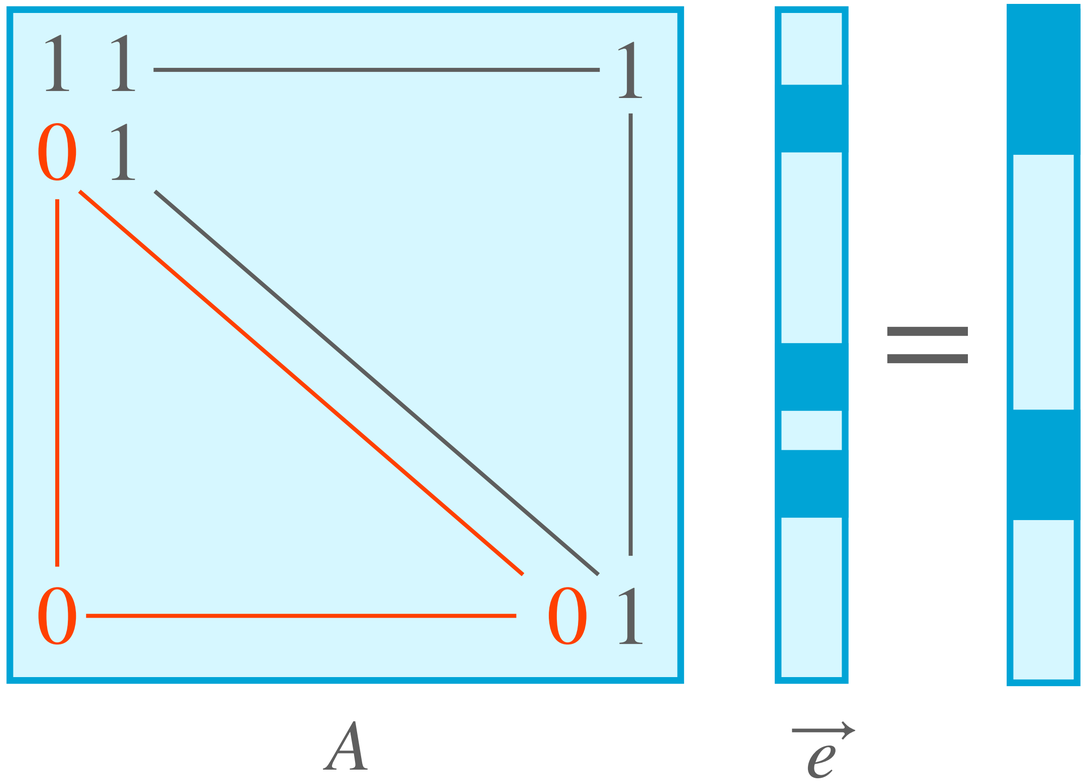
CONSTRUCTING PCFS

- Problem: if $|\vec{e}| = 2^{\lambda}, A \cdot \vec{e}$ too expensive to compute!
- Solution: use FSS for $F(x) := b \cdot (A \cdot \vec{e})_x$



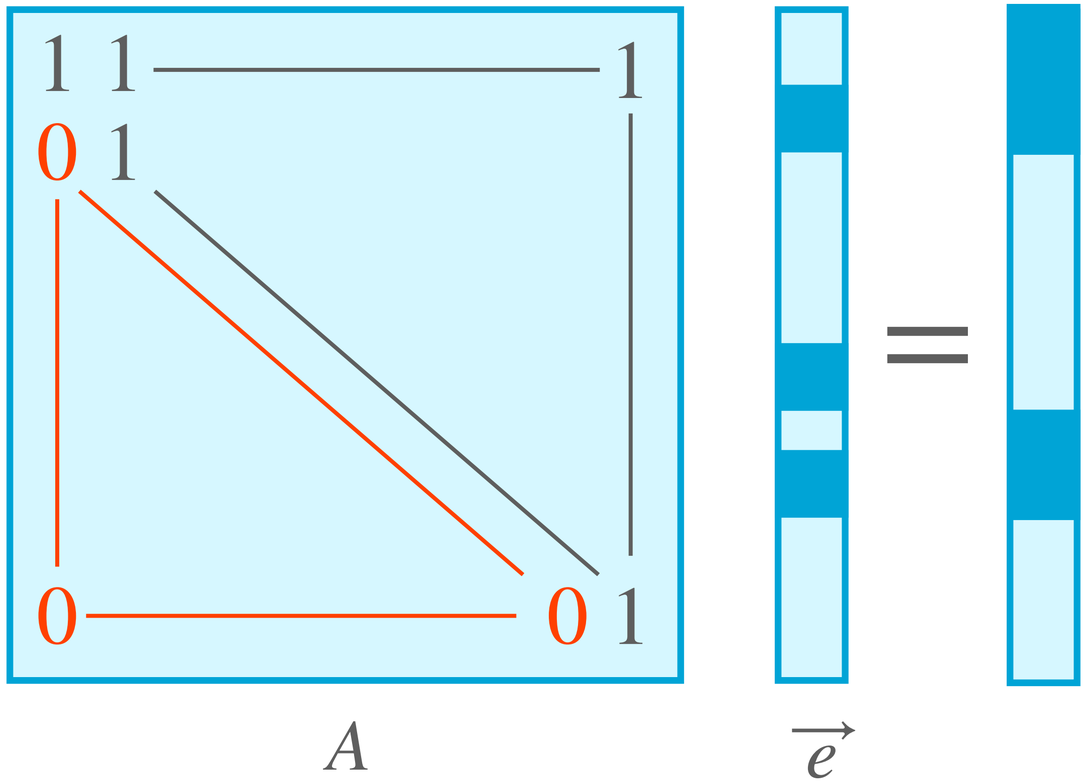
CONSTRUCTING PCFS

- Problem: if $|\vec{e}| = 2^{\lambda}, A \cdot \vec{e}$ too expensive to compute!
- Solution: use FSS for $F(x) := b \cdot (A \cdot \vec{e})_x$



CONSTRUCTING PCFS

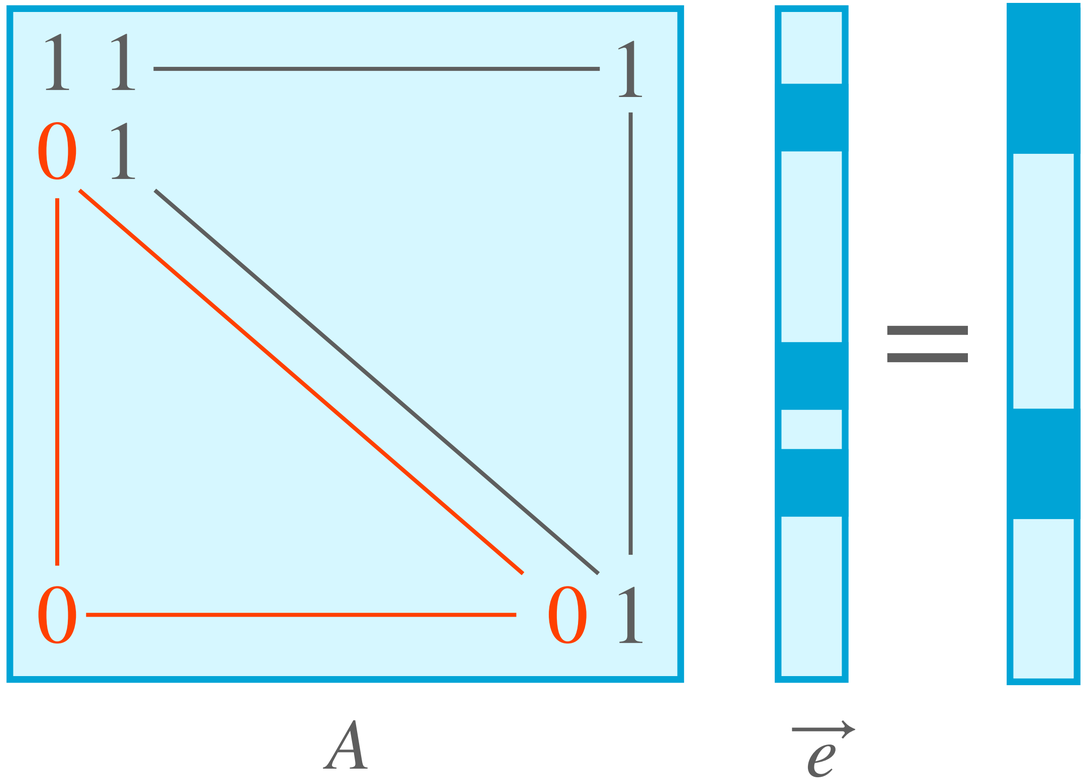
- Problem: if $|\vec{e}| = 2^{\lambda}, A \cdot \vec{e}$ too expensive to compute!
- Solution: use FSS for $F(x) := b \cdot (A \cdot \vec{e})_x$



Corresponds to (small sum of) comparison functions

CONSTRUCTING PCFS

- Problem: if $|\vec{e}| = 2^{\lambda}, A \cdot \vec{e}$ too expensive to compute!
- Solution: use FSS for $F(x) := b \cdot (A \cdot \vec{e})_x$

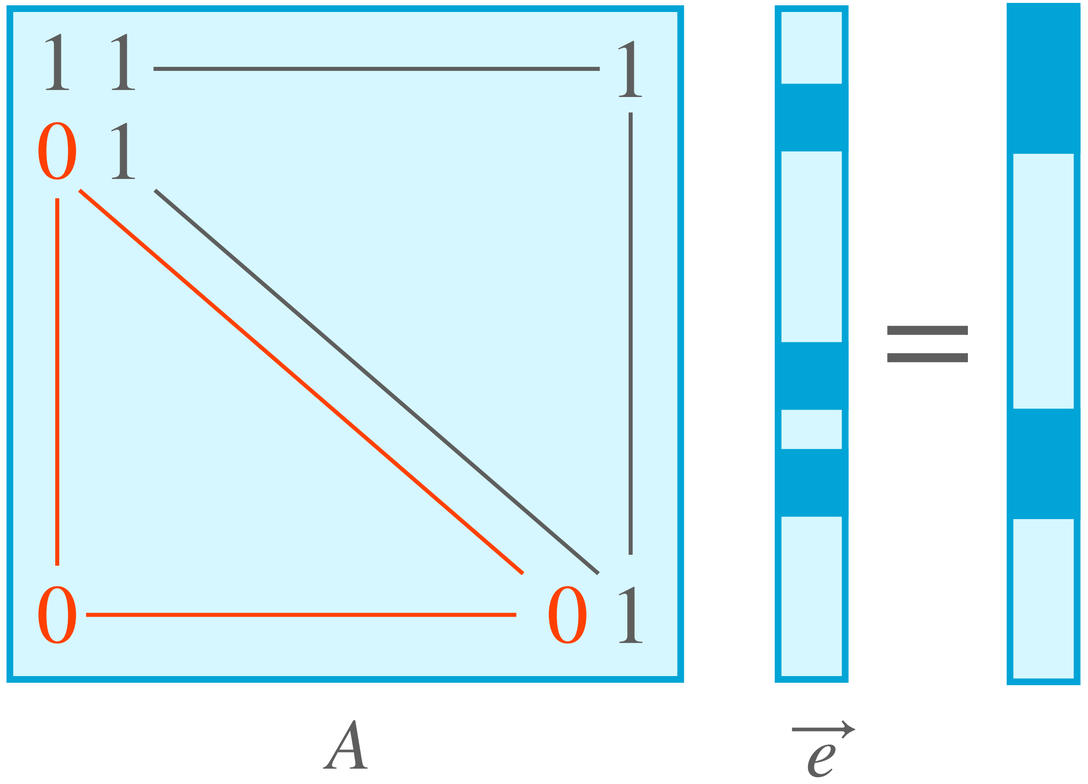


Corresponds to (small sum of) comparison functions

$$F_y^{\alpha}(x) = \alpha \cdot 1\{x \le y\} = \begin{cases} \alpha & x \le y \\ 0 & x > y \end{cases}$$

CONSTRUCTING PCFS

- Problem: if $|\vec{e}| = 2^{\lambda}, A \cdot \vec{e}$ too expensive to compute!
- Solution: use FSS for $F(x) := b \cdot (A \cdot \vec{e})_x$



Corresponds to (small sum of) comparison functions

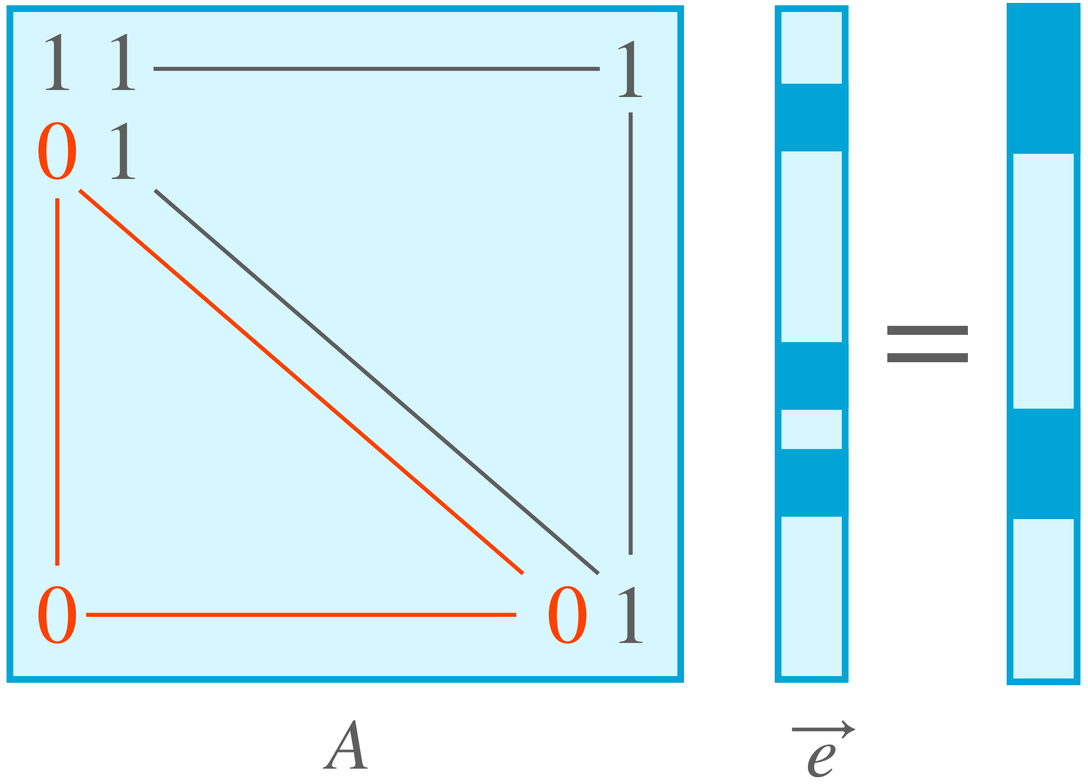
$$F_y^{\alpha}(x) = \alpha \cdot 1\{x \le y\} = \begin{cases} \alpha & x \le y \\ 0 & x > y \end{cases}$$

Efficient FSS due to [BGI'19,BCGGIKR'21]

CONSTRUCTING PCFS

Also get efficient PCFs for general degree-2/circuit-dependent correlations

- Problem: if $|\vec{e}| = 2^\lambda, A \cdot \vec{e}$ too expensive to compute!
- Solution: use FSS for $F(x) := b \cdot (A \cdot \vec{e})_x$



Corresponds to (small sum of) comparison functions

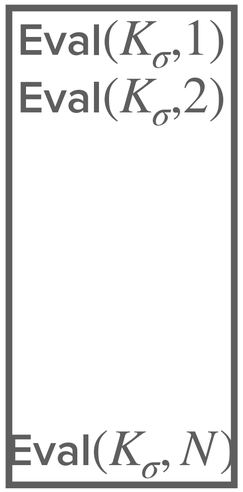
$$F_y^\alpha(x) = \alpha \cdot 1\{x \le y\} = \begin{cases} \alpha & x \le y \\ 0 & x > y \end{cases}$$

Efficient FSS due to [BGI'19,BCGGIKR'21]

SPEEDING UP OFFLINE

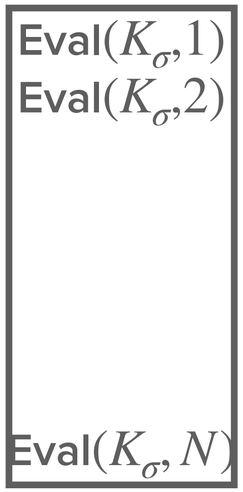
SPEEDING UP OFFLINE

- Bulk of offline work: $\mathsf{EvalAll}(K_\sigma) :=$



SPEEDING UP OFFLINE

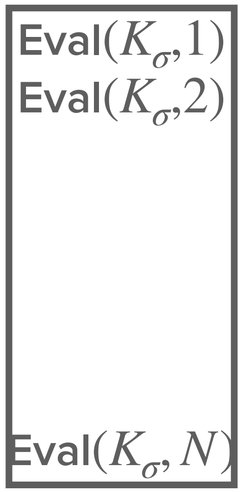
- Bulk of offline work: EvalAll$(K_\sigma) :=$



- FSS built from punctured PRF

SPEEDING UP OFFLINE

- Bulk of offline work: $\mathsf{EvalAll}(K_\sigma) :=$



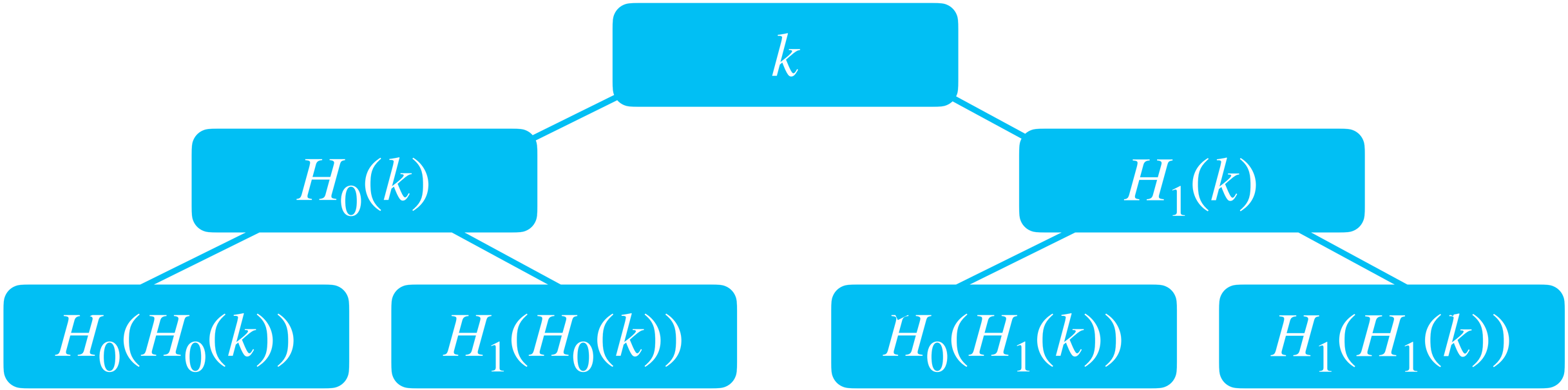
- FSS built from punctured PRF
- Basically: corresponds to computing entire GGM tree

SPEEDING UP OFFLINE

- Bulk of offline work: EvalAll(K_σ) :=

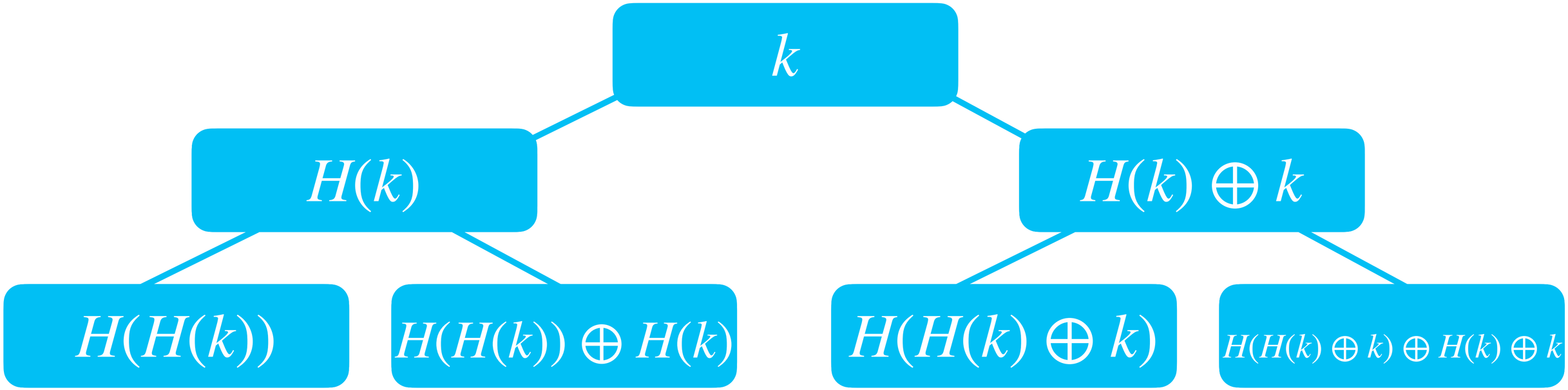
$$\begin{bmatrix} \text{Eval}(K_\sigma, 1) \\ \text{Eval}(K_\sigma, 2) \\ \\ \\ \\ \text{Eval}(K_\sigma, N) \end{bmatrix}$$

- FSS built from punctured PRF
- Basically: corresponds to computing entire GGM tree



SPEEDING UP OFFLINE

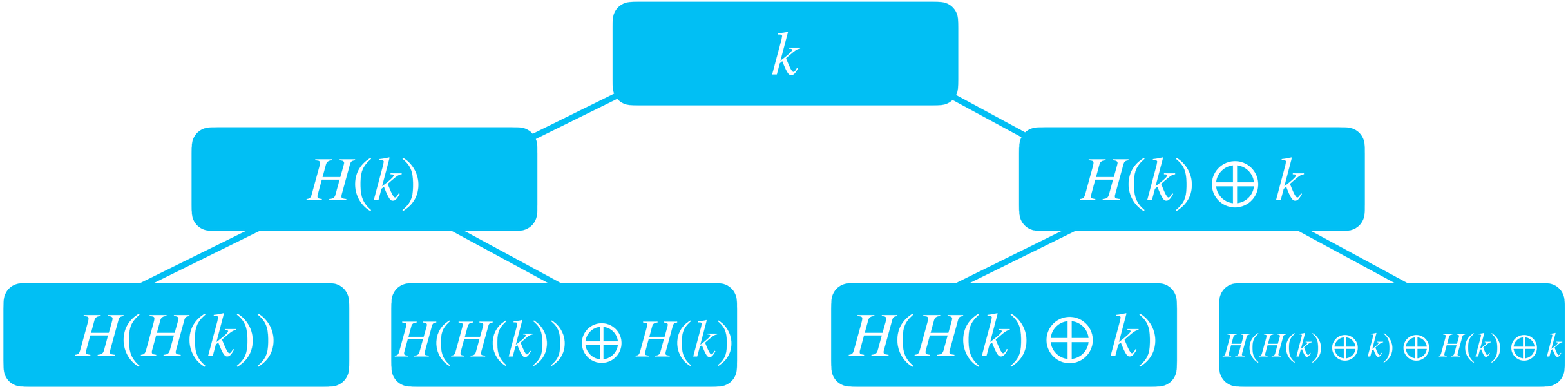
- Bulk of offline work: EvalAll$(K_\sigma) :=$ $\boxed{\begin{array}{l}\text{Eval}(K_\sigma,1)\\ \text{Eval}(K_\sigma,2)\\ \\ \\ \\ \text{Eval}(K_\sigma,N)\end{array}}$
- FSS built from punctured PRF
- Basically: corresponds to computing entire GGM tree



SPEEDING UP OFFLINE

Yields: unpredictable punctured function

- Bulk of offline work: $\text{EvalAll}(K_\sigma) :=$ $\begin{matrix}\text{Eval}(K_\sigma,1)\\ \text{Eval}(K_\sigma,2)\\ \\ \\ \\ \text{Eval}(K_\sigma,N)\end{matrix}$
- FSS built from punctured PRF
- Basically: corresponds to computing entire GGM tree

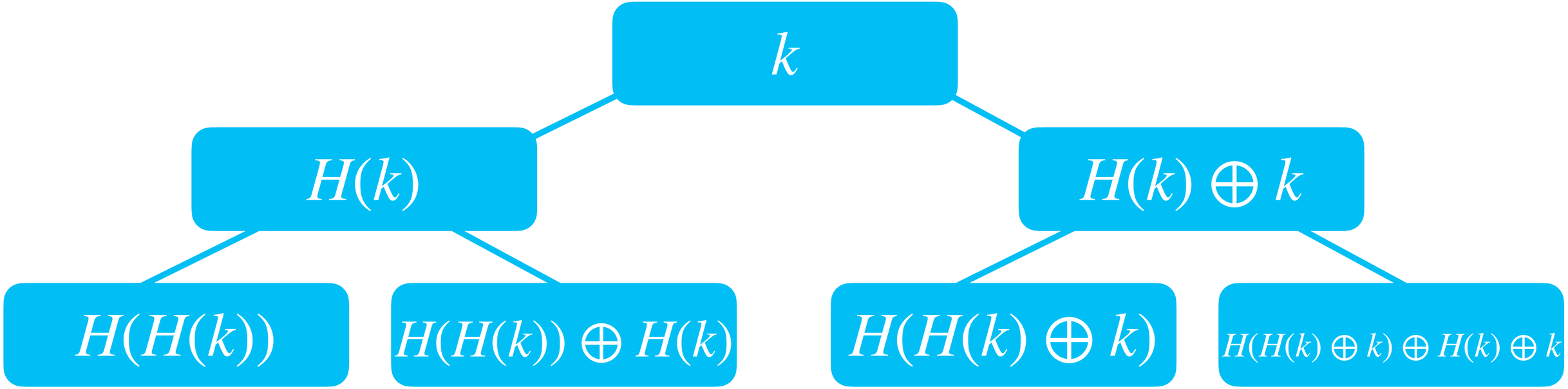


SPEEDING UP OFFLINE

- Bulk of offline work: $\mathsf{EvalAll}(K_\sigma) :=$ $\mathsf{Eval}(K_\sigma, 1)$ $\mathsf{Eval}(K_\sigma, 2)$ $\mathsf{Eval}(K_\sigma, N)$
- FSS built from punctured PRF
- Basically: corresponds to computing entire GGM tree

Yields: unpredictable punctured function

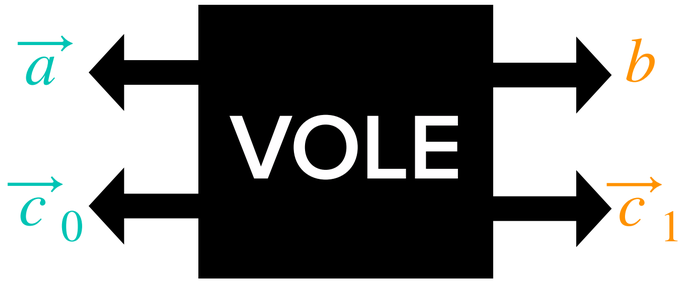
One hashing layer: yields genuine punctured PRF



RECAP

RECAP

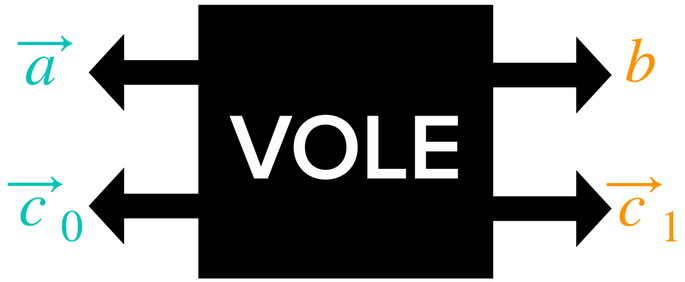
(Pseudo)random correlations



$$b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$$

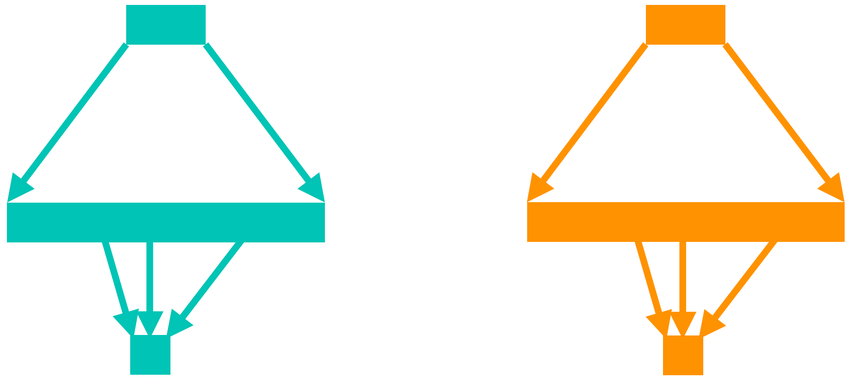
RECAP

(Pseudo)random correlations



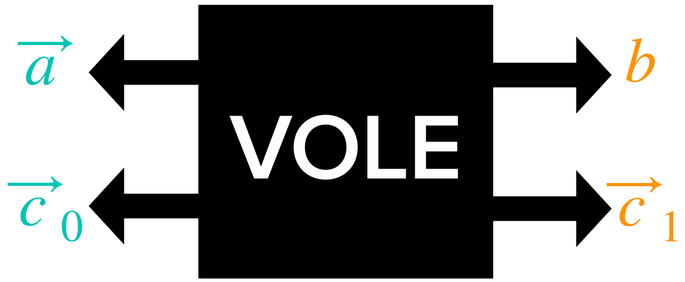
$$b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$$

Offline-Online PCGs



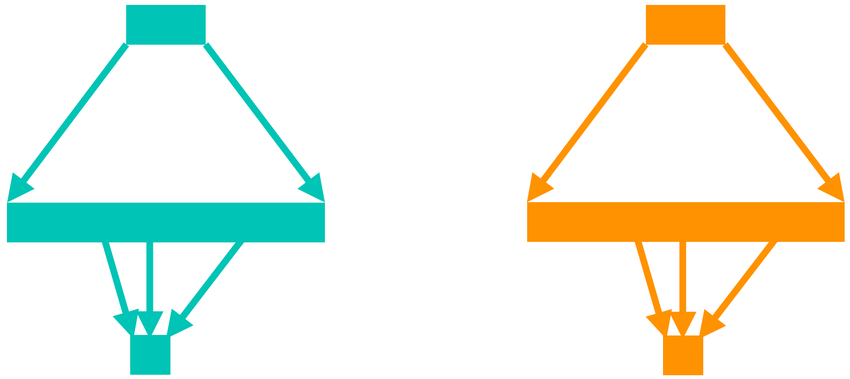
RECAP

(Pseudo)random correlations



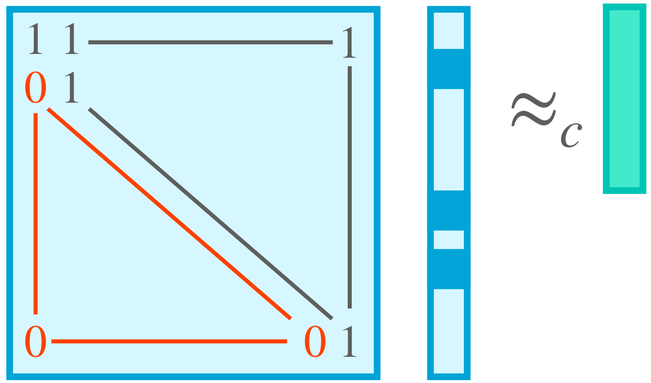
$$b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$$

Offline-Online PCGs



Expand-Accumulate

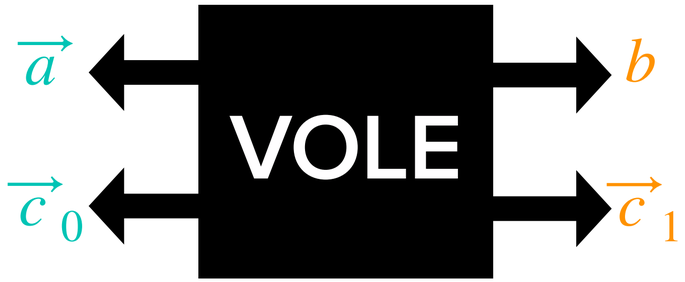
sparse



$\approx_c$

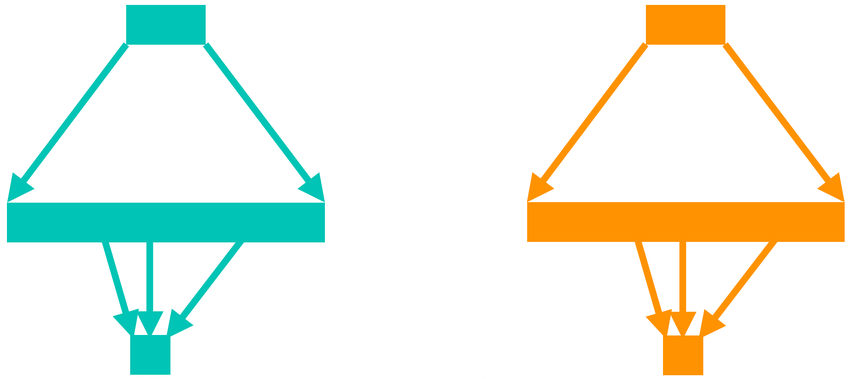
RECAP

(Pseudo)random correlations



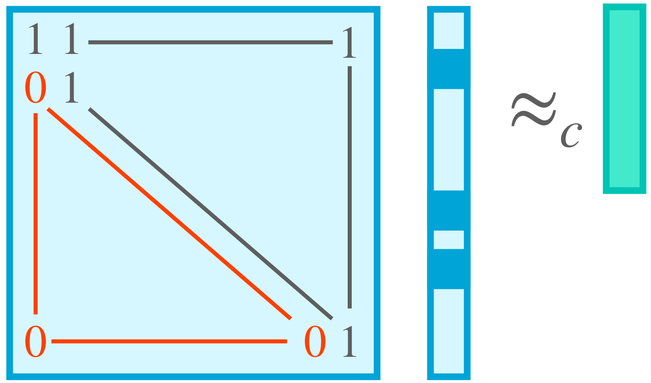
$$b \cdot \vec{a} = \vec{c}_0 + \vec{c}_1$$

Offline-Online PCGs



Expand-Accumulate

sparse



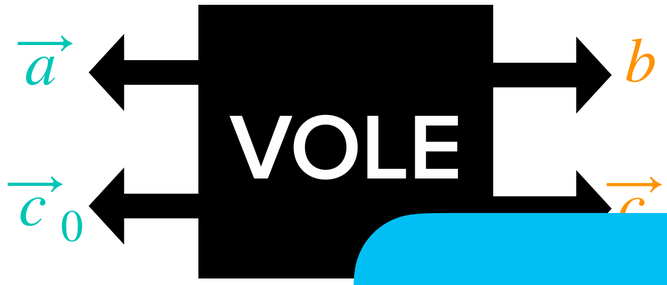
$\approx_c$

Other results

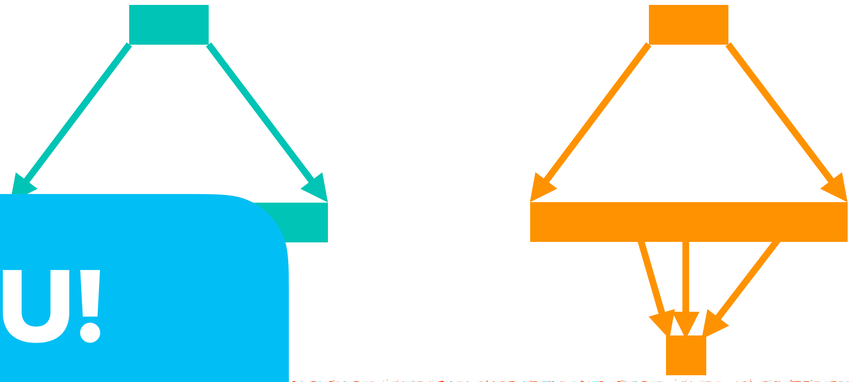
- Concretely efficient PCF's
- New correlations
- Sped-up offline phase

RECAP

(Pseudo)random correlations

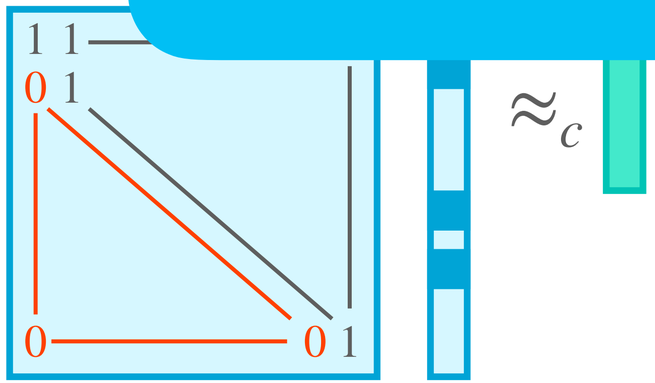


Offline-Online PCGs



Expand-Ac

sparse



er results

- Concretely efficient PCF's
- New correlations
- Sped-up offline phase

THANK YOU!

QUESTIONS?