PRF with Weak Programming Privacy
and Applications to Private Information Retrieval

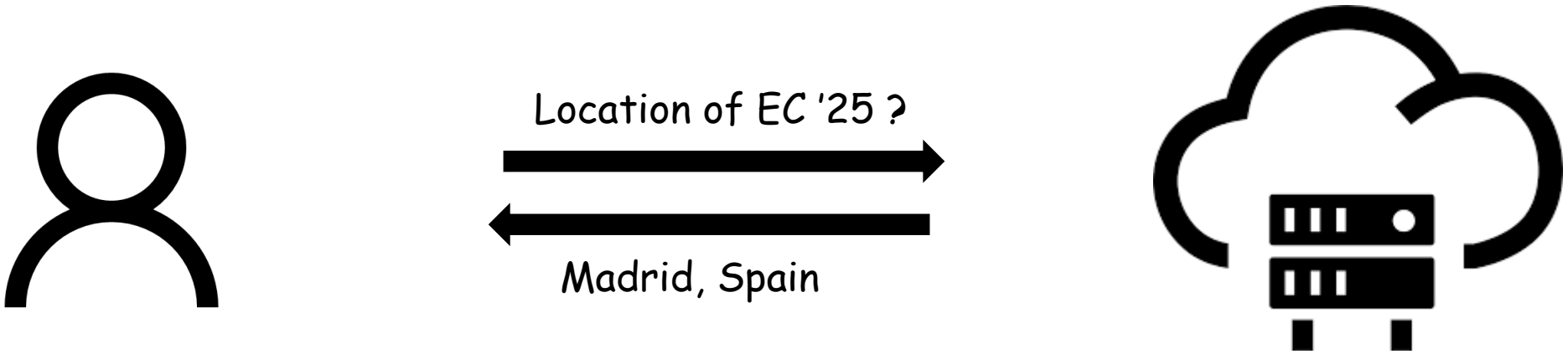
Bo Peng (Peking University)

joint work with

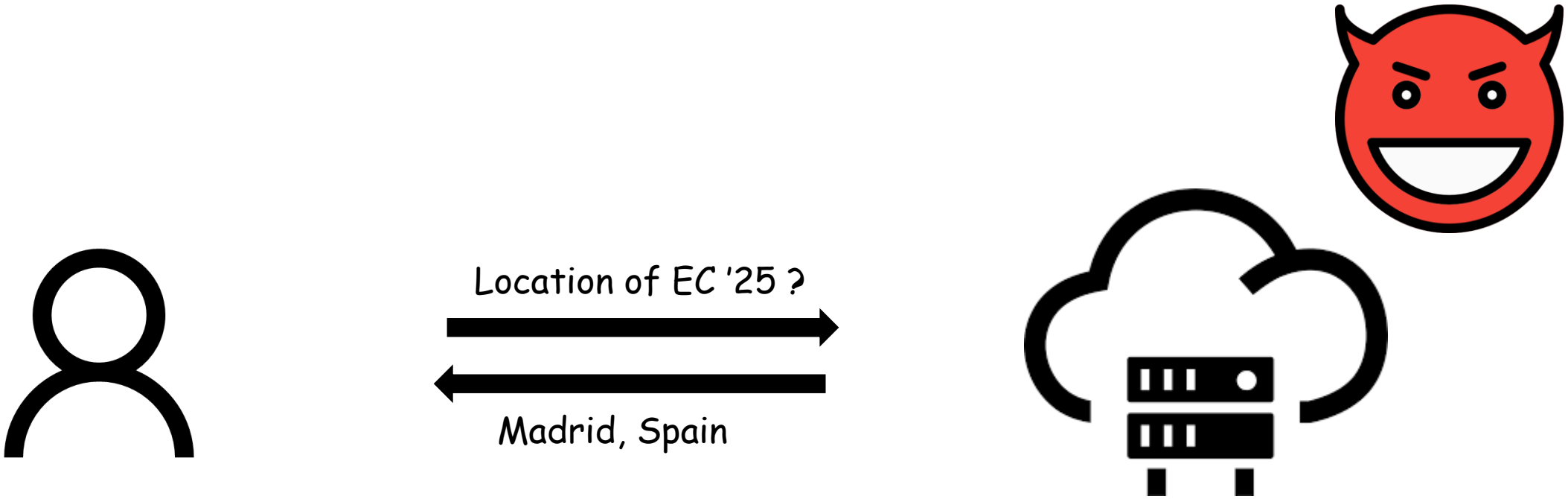
Ashrujit Ghoshal Mingxun Zhou Elaine Shi

Carnegie Mellon University

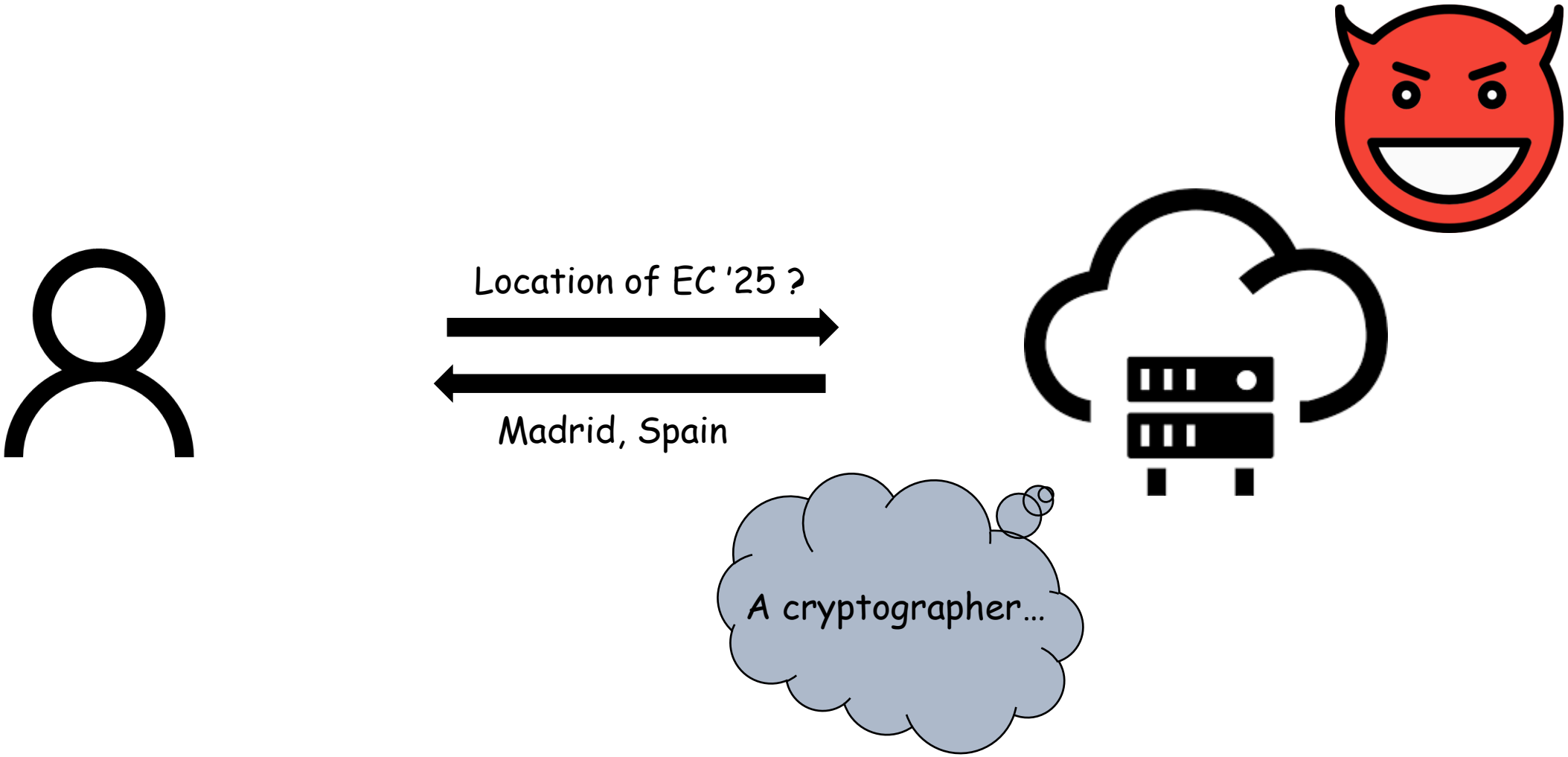
Information Retrieval Leaks User Privacy



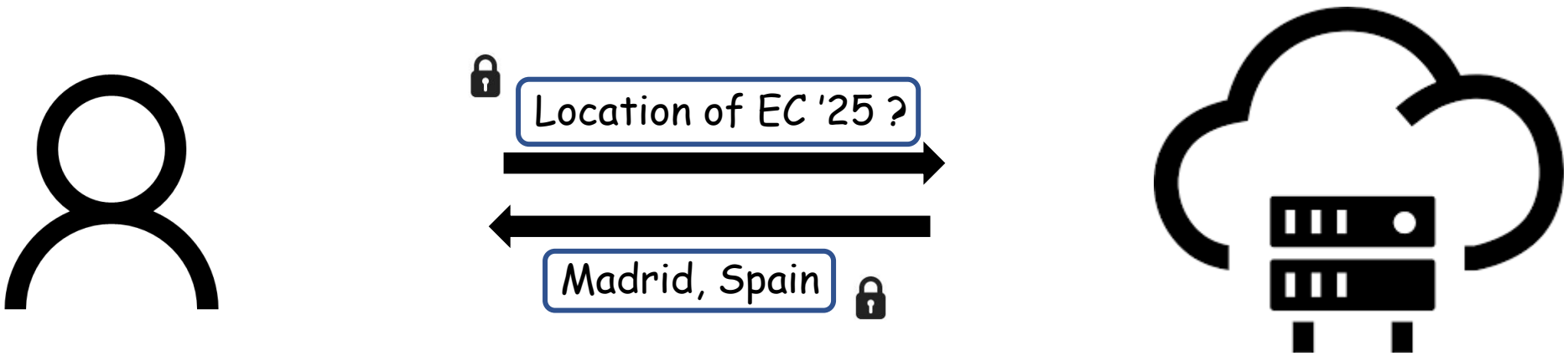
Information Retrieval Leaks User Privacy



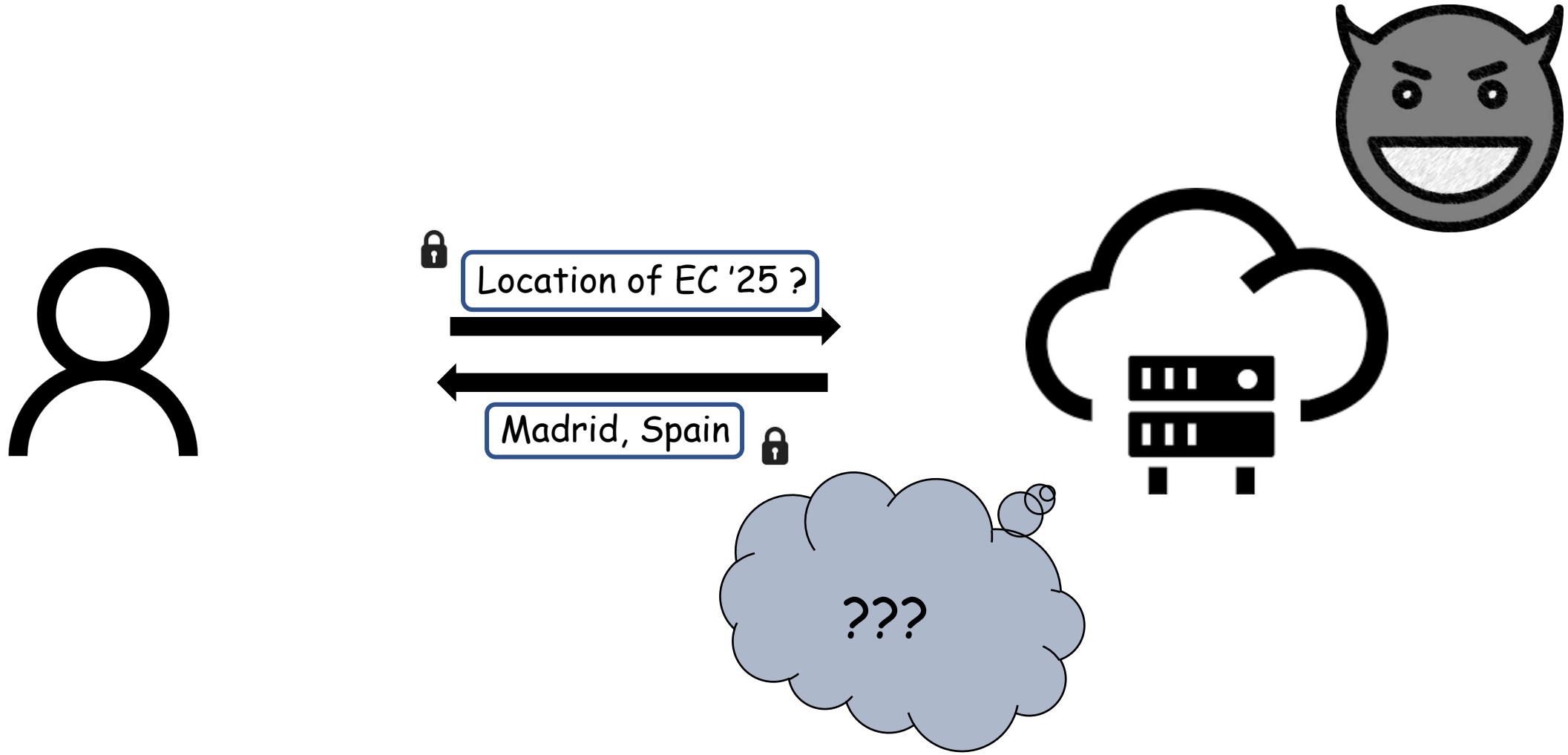
Information Retrieval Leaks User Privacy



Private Information Retrieval [CGKS95]



Private Information Retrieval [CGKS95]

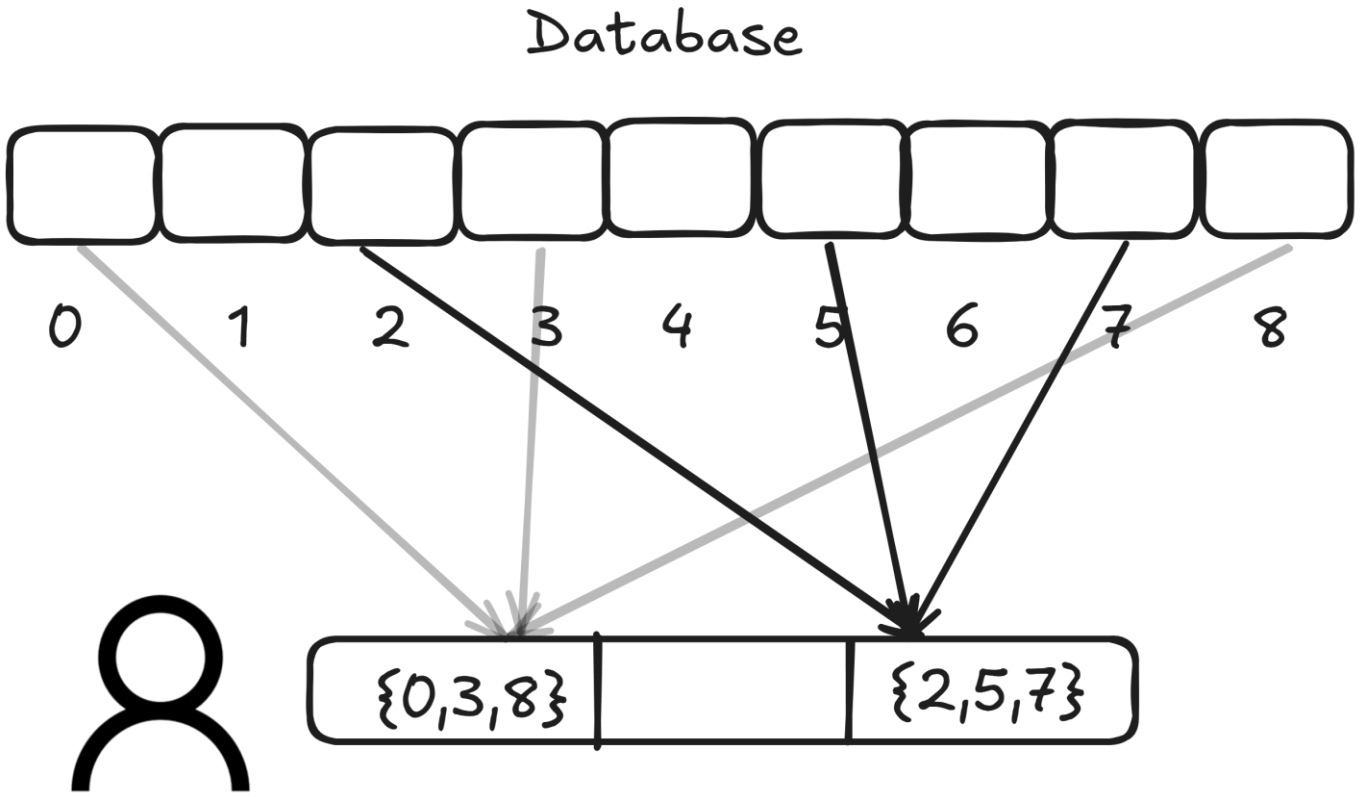


Recently…

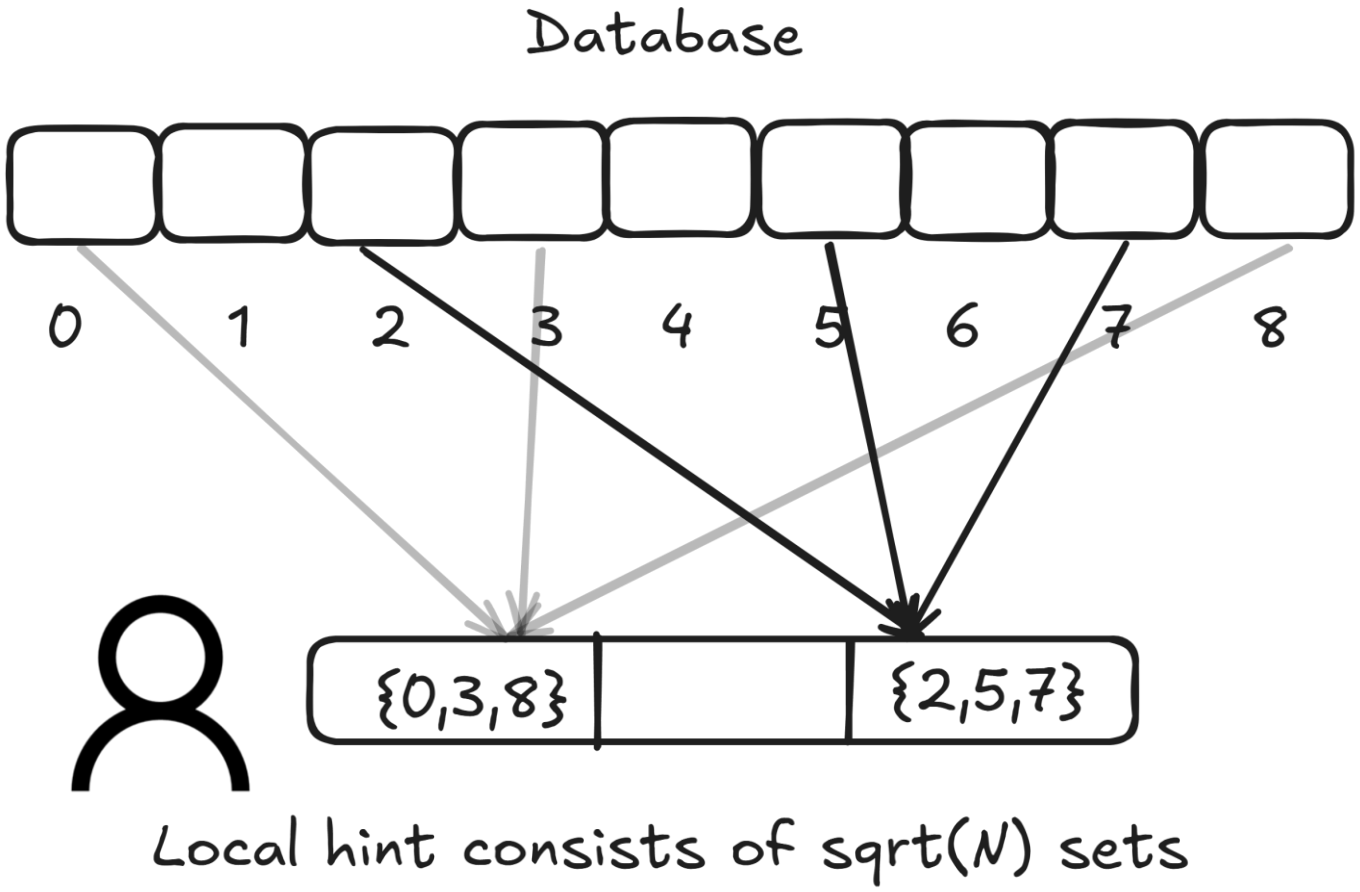
- Advance on Preprocessing PIR
  - [BIM00],[CK20],[KC21],[SACM21],[CHK22],[LP22],[LP22b],[HHCM+23],[ZLTS23],[LMW23],[LP23],[MIR23],[ZPSZ24],[HPPY24],[GZS24],[ISW24]

- Many of them follow a common framework

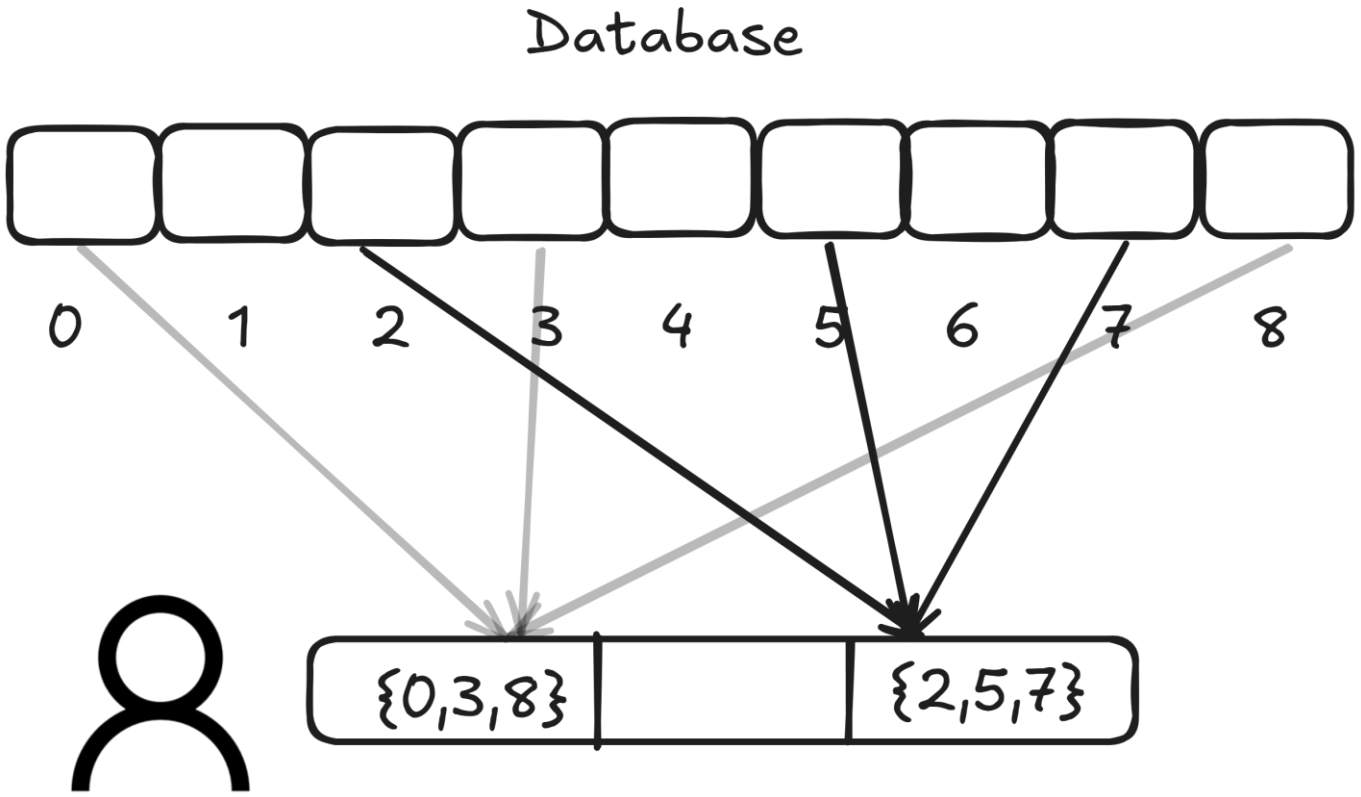
Preprocessing



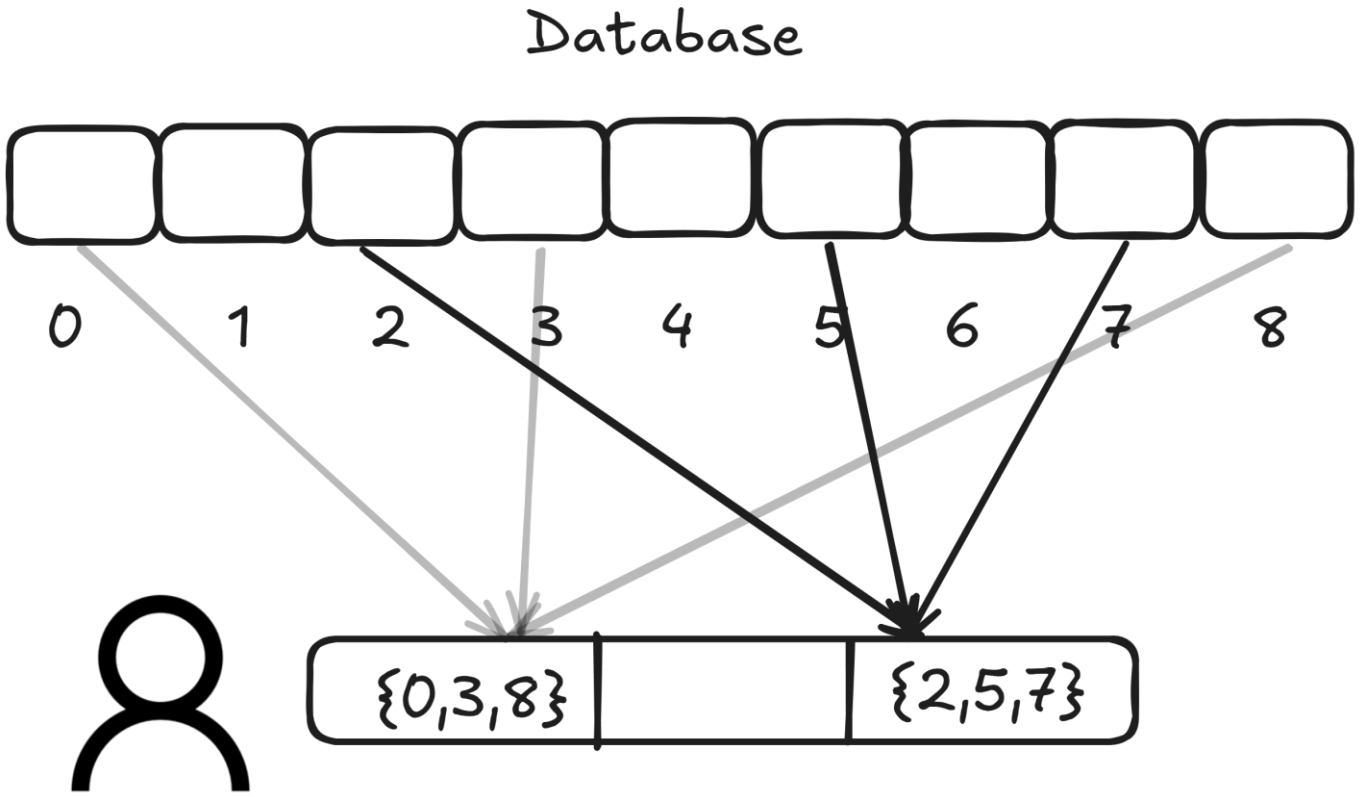
Preprocessing



Preprocessing

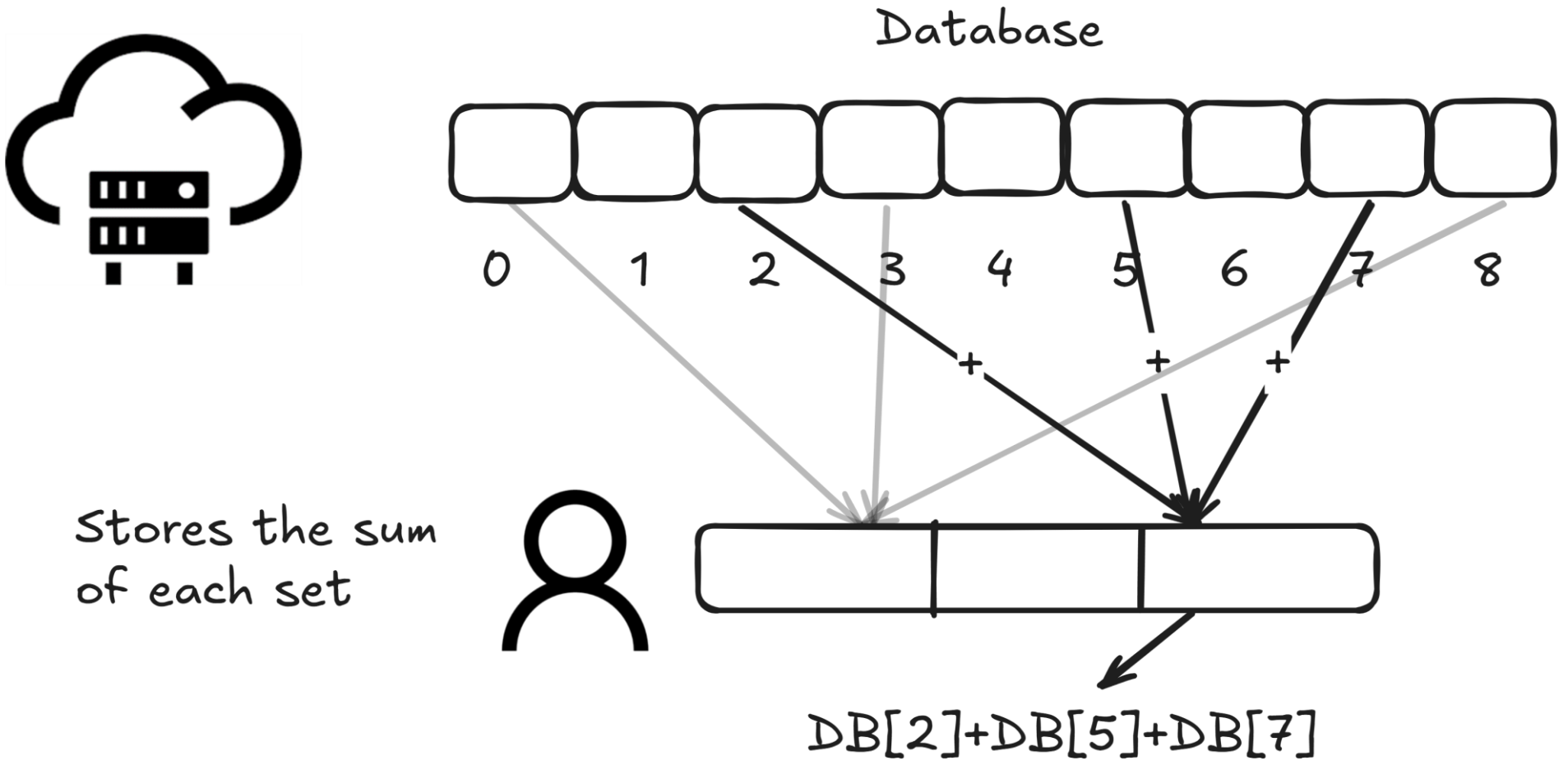


Preprocessing



Local hint consists of sqrt(N) sets
Each set is represented by a PRF key
Each set has sqrt(N) random elements

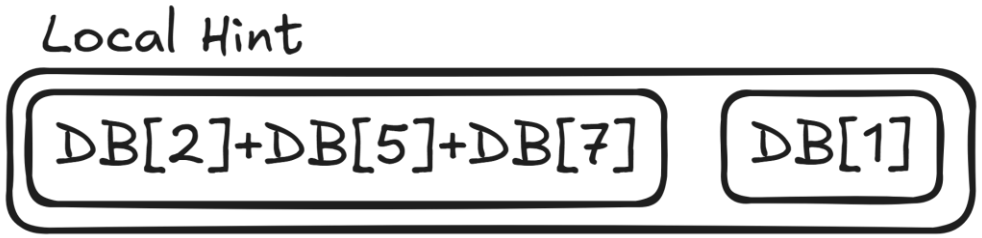
Preprocessing



Query

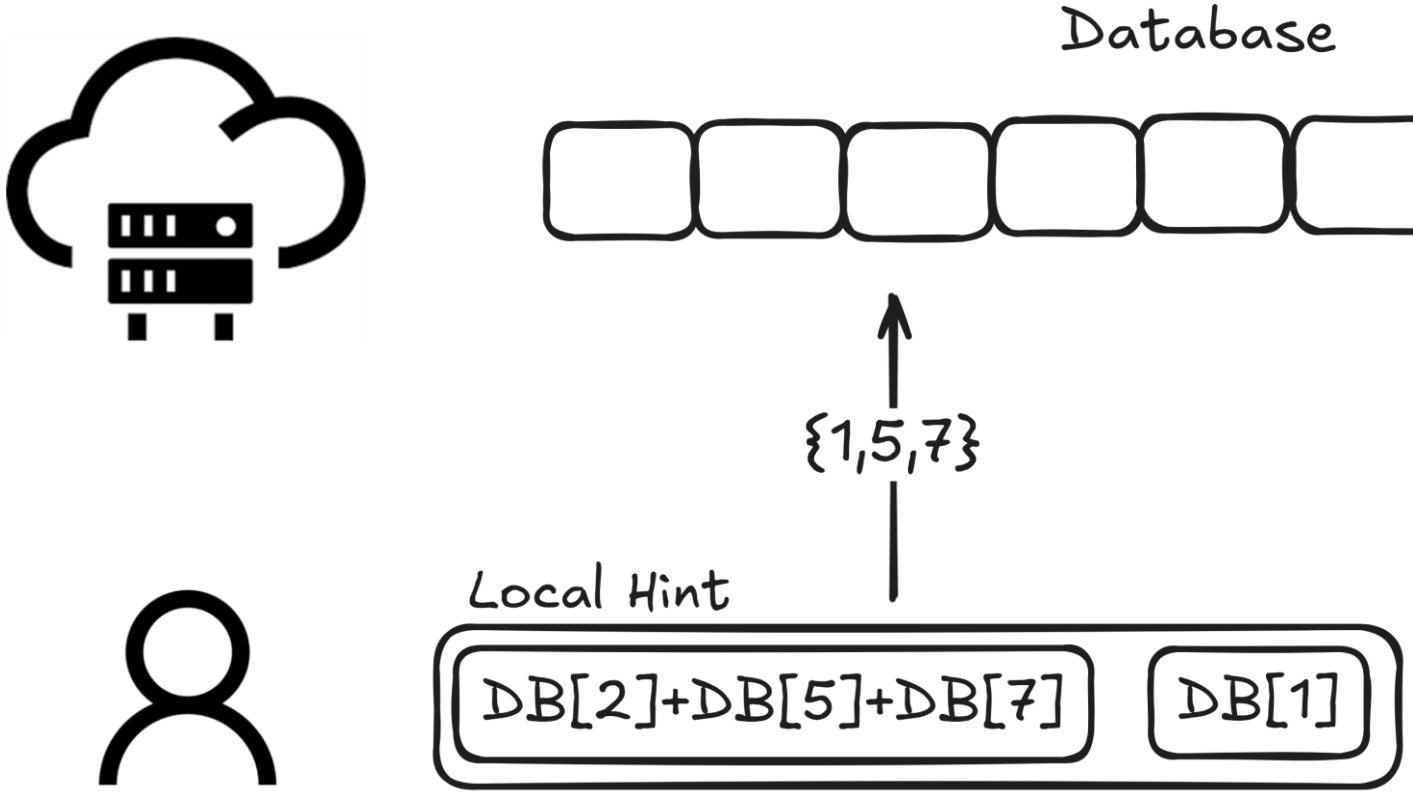
what is DB[2]?

Query



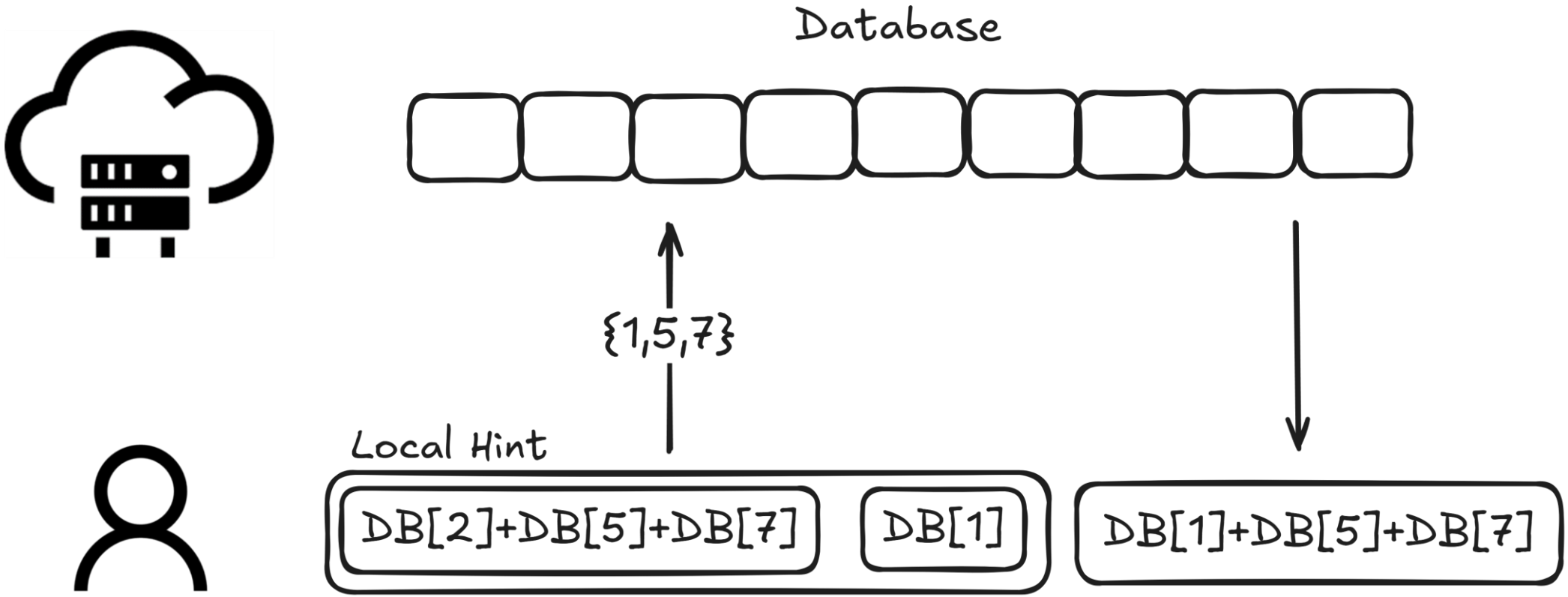
what is DB[2]?

Query



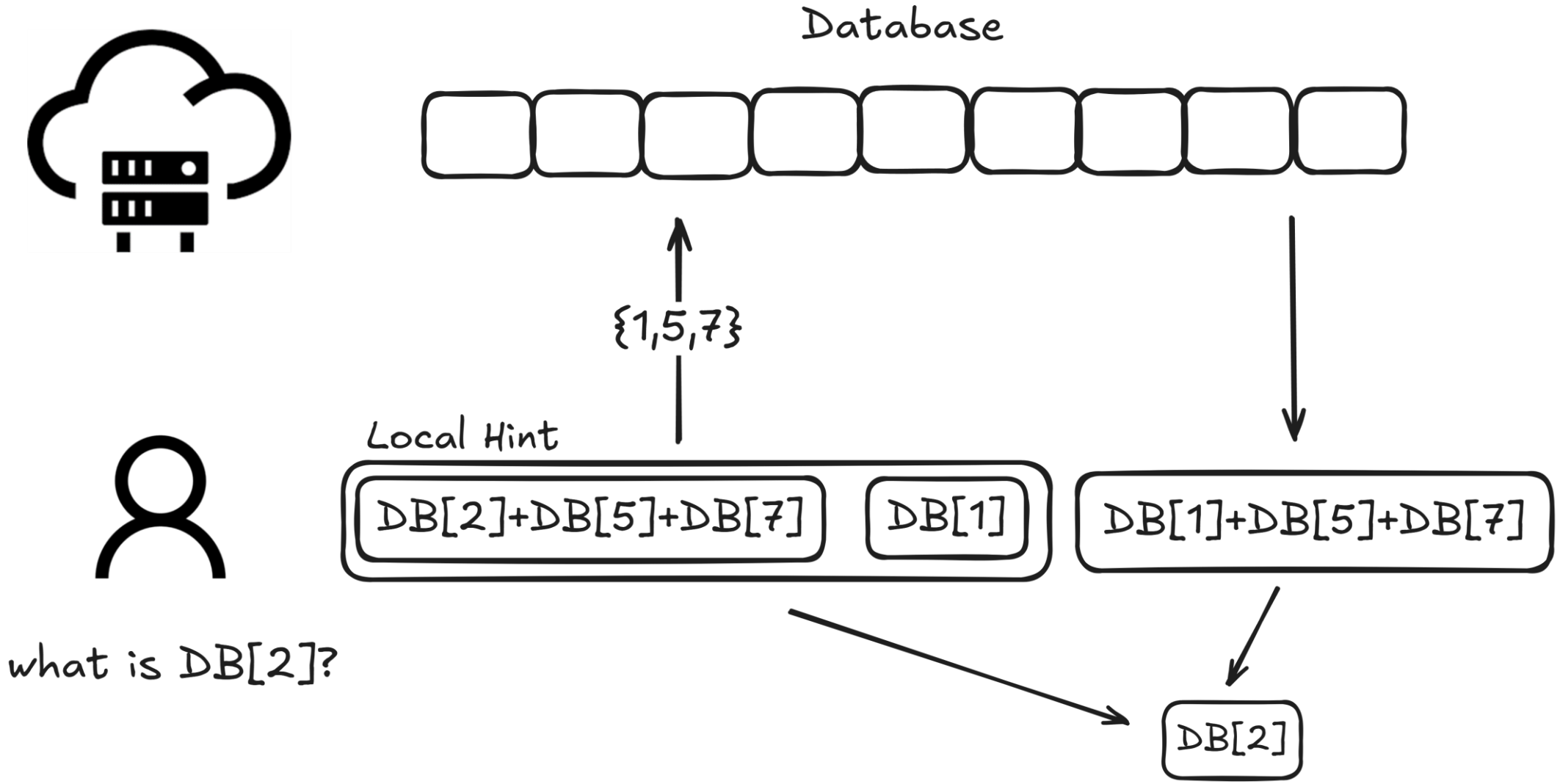
what is DB[2]?

Query

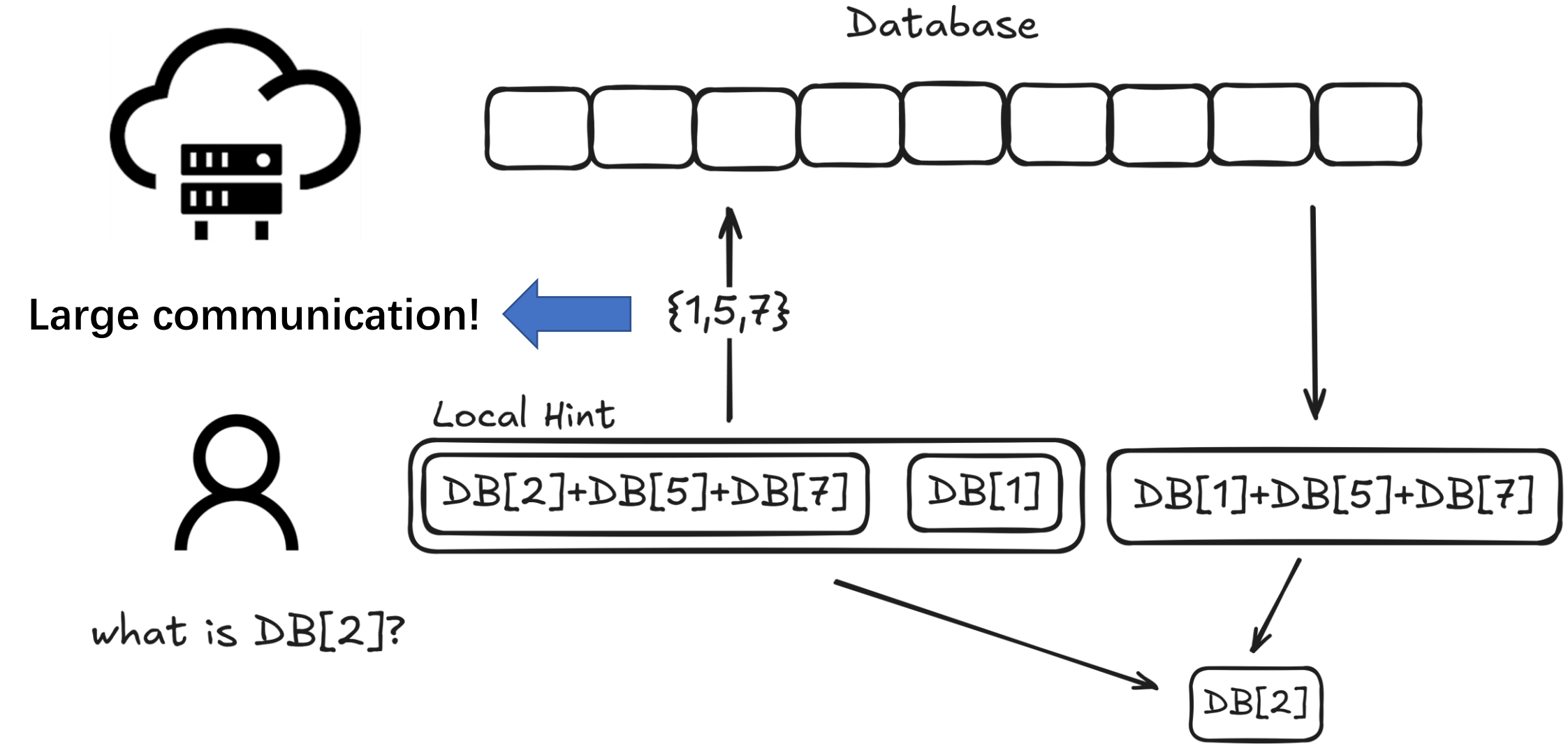


what is DB[2]?

Query


Database
{1,5,7}
Local Hint
DB[2]+DB[5]+DB[7]
DB[1]
DB[1]+DB[5]+DB[7]
what is DB[2]?
DB[2]

Query


Database
Large communication!
{1,5,7}
Local Hint
DB[2]+DB[5]+DB[7]
DB[1]
DB[1]+DB[5]+DB[7]
what is DB[2]?
DB[2]

Bottleneck

Bottleneck

- How to send the set in a succinct manner?

Bottleneck

- How to send the set in a succinct manner?

- Recall: $\{1,5,7\} = \{2,5,7\} - \{2\} + \{1\}$

Bottleneck

- How to send the set in a succinct manner?

- Recall: $\{1,5,7\} = \{2,5,7\} - \{2\} + \{1\}$
- Transform a PRF key msk into another key psk:

Bottleneck

- How to send the set in a succinct manner?

- Recall: $\{1,5,7\} = \{2,5,7\} - \{2\} + \{1\}$
- Transform a PRF key msk into another key psk:
  - $\mathrm{Eval}(msk,\cdot)$ and $\mathrm{Eval}(psk,\cdot)$ differ at one position

Bottleneck

- How to send the set in a succinct manner?

- Recall: $\{1,5,7\} = \{2,5,7\} - \{2\} + \{1\}$
- Transform a PRF key msk into another key psk:
  - $\mathrm{Eval}(msk,\cdot)$ and $\mathrm{Eval}(psk,\cdot)$ differ at one position
  - psk leaks no information about the differing position

Bottleneck

- How to send the set in a succinct manner?

- Recall: $\{1,5,7\} = \{2,5,7\} - \{2\} + \{1\}$
- Transform a PRF key msk into another key psk:
  - $\mathrm{Eval}(msk,\cdot)$ and $\mathrm{Eval}(psk,\cdot)$ differ at one position
  - psk leaks no information about the differing position
  - Minimize $|psk|$

Bottleneck

- How to send the set in a succinct manner?

- Recall: $\{1,5,7\} = \{2,5,7\} - \{2\} + \{1\}$
- Transform a PRF key msk into another key psk:
  - $\mathrm{Eval}(msk,\cdot)$ and $\mathrm{Eval}(psk,\cdot)$ differ at one position
  - psk leaks no information about the differing position
  - Minimize $|psk|$

- Privately Programmable PRF (PPPRF)

Puncturable PRF (pPRF)

Puncturable PRF (pPRF)

- Evaluation algorithm $\mathbf{Eval}(msk, x)$

Puncturable PRF (pPRF)

- Evaluation algorithm $\mathbf{Eval}(msk, x)$
- For any y, can generate a punctured key psk_y from msk

Puncturable PRF (pPRF)

- Evaluation algorithm $\mathbf{Eval}(msk, x)$
- For any y, can generate a punctured key psk_y from msk
- Punctured evaluation algorithm $\mathbf{PEval}(psk_y, x)$

Puncturable PRF (pPRF)

- Evaluation algorithm $\mathbf{Eval}(msk, x)$
- For any y, can generate a punctured key psk_y from msk
- Punctured evaluation algorithm $\mathbf{PEval}(psk_y, x)$

- Correctness: $\mathbf{Eval}(msk, x) = \mathbf{PEval}(psk_y, x), \forall x \neq y$

Puncturable PRF (pPRF)

- Evaluation algorithm $\mathbf{Eval}(msk, x)$
- For any y, can generate a punctured key psk_y from msk
- Punctured evaluation algorithm $\mathbf{PEval}(psk_y, x)$

- Correctness: $\mathbf{Eval}(msk, x) = \mathbf{PEval}(psk_y, x), \forall x \neq y$
- Security: given psk_y, $\mathbf{Eval}(msk, y) \approx_c \mathbf{random}$

Puncturable PRF (pPRF)

- Evaluation algorithm $\mathbf{Eval}(msk, x)$
- For any y, can generate a punctured key psk_y from msk
- Punctured evaluation algorithm $\mathbf{PEval}(psk_y, x)$

- Correctness: $\mathbf{Eval}(msk, x) = \mathbf{PEval}(psk_y, x), \forall x \neq y$
- Security: given psk_y, $\mathbf{Eval}(msk, y) \approx_c \mathbf{random}$

- Can be constructed using classical GGM PRF

Privacy and Programmability

Privacy and Programmability

- Ordinarily, a punctured key psk_y reveals y
  - As is the case of GGM PRF

Privacy and Programmability

- Ordinarily, a punctured key psk_y reveals y
  - As is the case of GGM PRF

- Privacy: psk_y does not reveal anything about y

Privacy and Programmability

- Ordinarily, a punctured key psk_y reveals y
  - As is the case of GGM PRF

- Privacy: psk_y does not reveal anything about y
- Programmability: can program psk_y to a desired value at y

Privacy and Programmability

- Ordinarily, a punctured key psk_y reveals y
  - As is the case of GGM PRF

- Privacy: psk_y does not reveal anything about y
- Programmability: can program psk_y to a desired value at y

- Applications: watermarking PRFs, MPC, PIR, ⋯

How to construct a PPPRF?

How to construct a PPPRF?

- Previous constructions: based on LWE or iO

How to construct a PPPRF?

- Previous constructions: based on LWE or iO
  - All LWE constructions are similar to layered FHE [PS18, …]

How to construct a PPPRF?

- Previous constructions: based on LWE or iO
  - All LWE constructions are similar to layered FHE [PS18, …]
  - Strong assumption & Too slow!

How to construct a PPPRF?

- Previous constructions: based on LWE or iO
  - All LWE constructions are similar to layered FHE [PS18, …]
  - Strong assumption & Too slow!
- Previous construction: based on OWF [BGIK22]

How to construct a PPPRF?

- Previous constructions: based on LWE or iO
  - All LWE constructions are similar to layered FHE [PS18, …]
  - Strong assumption & Too slow!
- Previous construction: based on OWF [BGIK22]
  - Evaluation time $\Omega(n)$, where n is the domain size

How to construct a PPPRF?

- Previous constructions: based on LWE or iO
  - All LWE constructions are similar to layered FHE [PS18, …]
  - Strong assumption & Too slow!
- Previous construction: based on OWF [BGIK22]
  - Evaluation time $\Omega(n)$, where n is the domain size
  - Too slow!

How to construct a PPPRF?

- Previous constructions: based on LWE or iO
  - All LWE constructions are similar to layered FHE [PS18, …]
  - Strong assumption & Too slow!
- Previous construction: based on OWF [BGIK22]
  - Evaluation time $\Omega(n)$, where n is the domain size
  - Too slow!

- Can we construct PPPRF based on OWF, with efficient evaluation?

Strawman

Strawman

- Baseline: [BGIK22], evaluation time linear in domain size

Strawman

- Baseline: [BGIK22], evaluation time linear in domain size
- Make the domain size smaller?

Strawman

- Baseline: [BGIK22], evaluation time linear in domain size
- Make the domain size smaller?

- Divide the domain size n into $n^{1/2}$ blocks

Strawman

- Baseline: [BGIK22], evaluation time linear in domain size
- Make the domain size smaller?

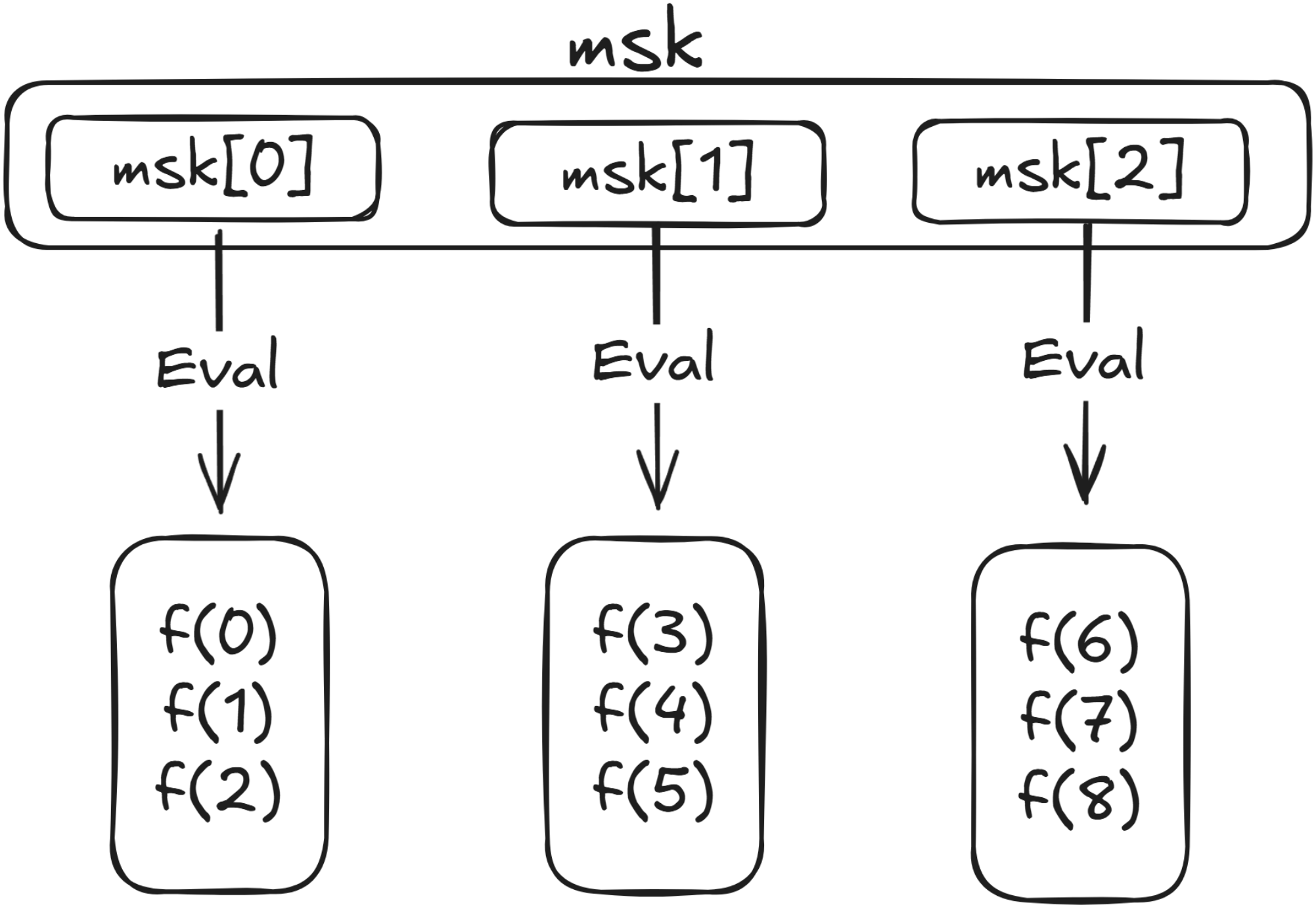
- Divide the domain size n into $n^{1/2}$ blocks
- Assign each block a [BGIK22] key

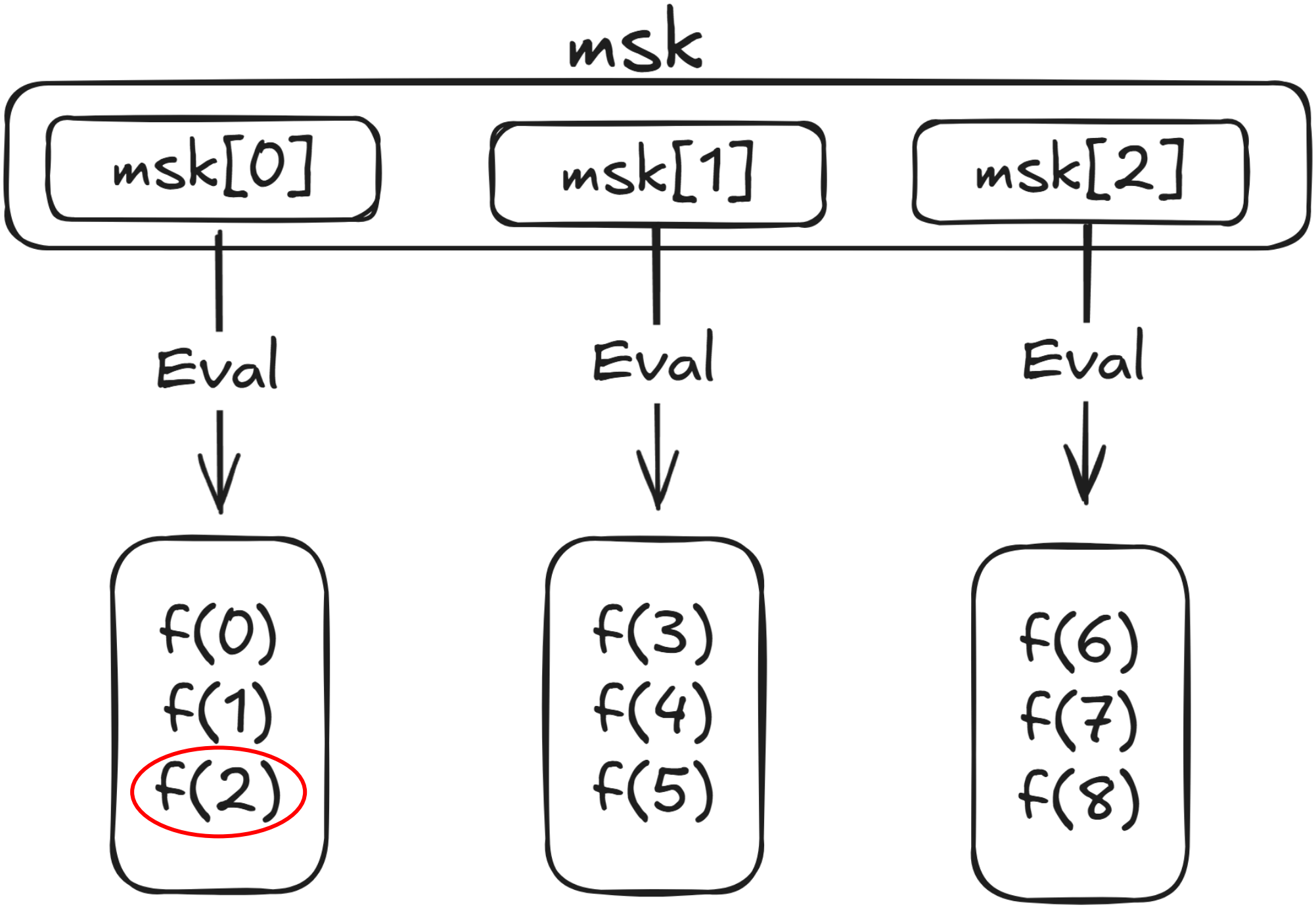
Strawman

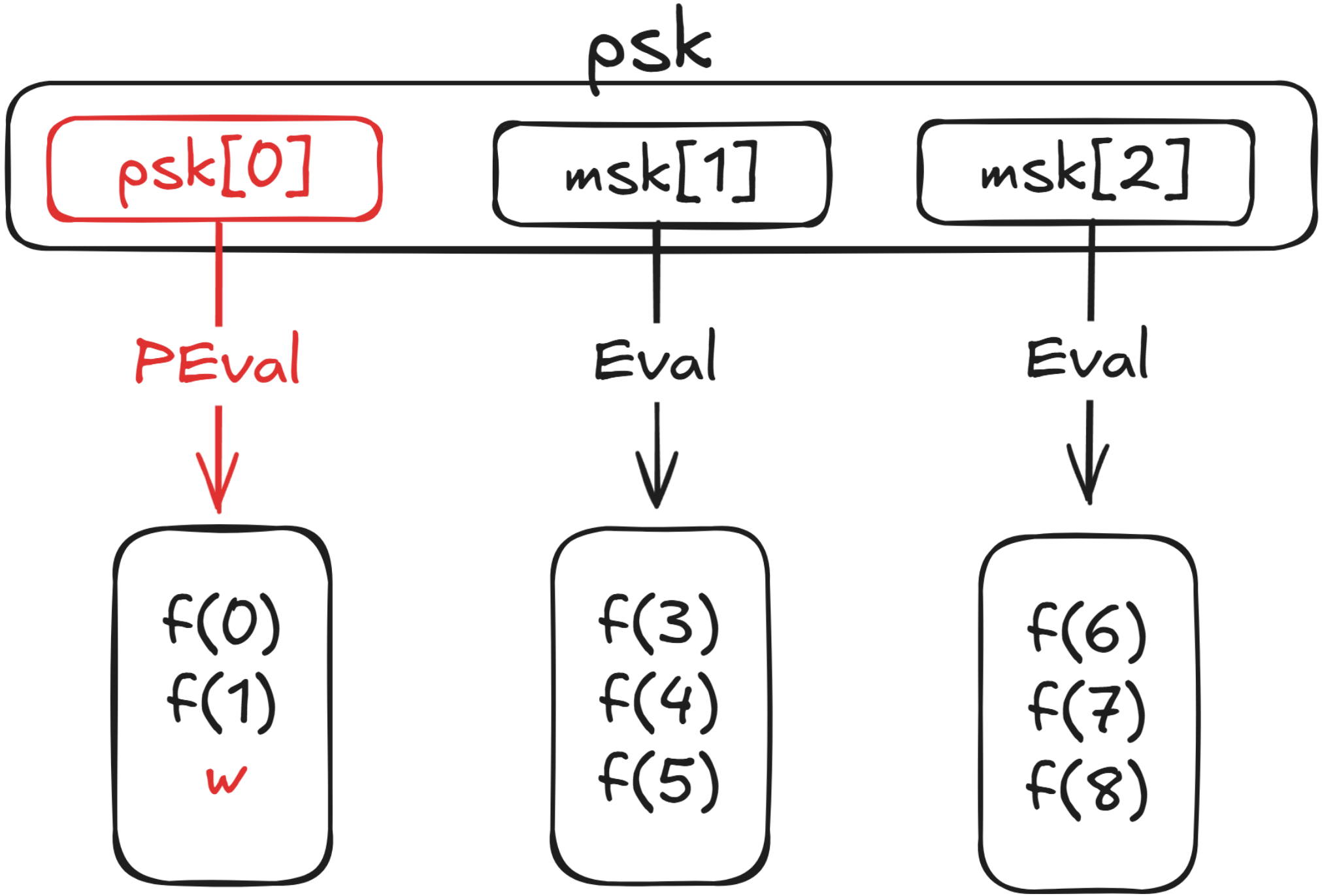
- Baseline: [BGIK22], evaluation time linear in domain size
- Make the domain size smaller?

- Divide the domain size n into $n^{1/2}$ blocks
- Assign each block a [BGIK22] key
- Expand a single key during Eval/Prog


msk
msk[0]
msk[1]
msk[2]


msk
msk[0]
msk[1]
msk[2]
Eval
Eval
Eval
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)


msk
msk[0]
msk[1]
msk[2]
Eval
Eval
Eval
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)


psk
psk[0]
msk[1]
msk[2]
PEval
Eval
Eval
f(0)
f(1)
w
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)

Strawman

- Key size: $O_\lambda(n^{1/2})$
- Eval/Prog: $O_\lambda(n^{1/2})$

Strawman

- Key size: $O_\lambda(n^{1/2})$
- Eval/Prog: $O_\lambda(n^{1/2})$

- Privacy?

Strawman: privacy

Strawman: privacy

- Easily identify the programmed block!

Strawman: privacy

- Easily identify the programmed block!

- A generic solution: flooding

Strawman: privacy

- Easily identify the programmed block!

- A generic solution: flooding
  - Assign each block $\mathbf{poly}(\lambda)$ keys, each randomly normal or programmed

Strawman: privacy

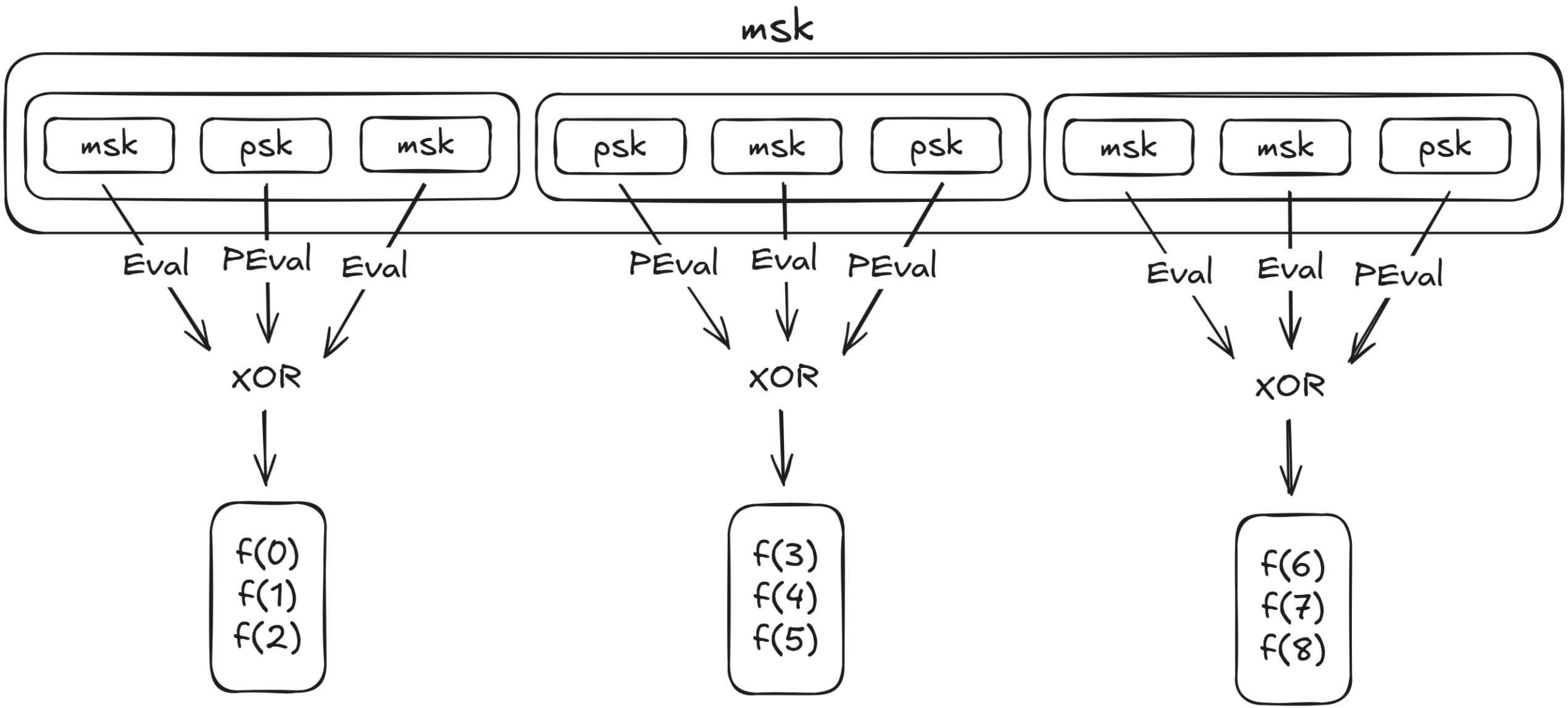
- Easily identify the programmed block!

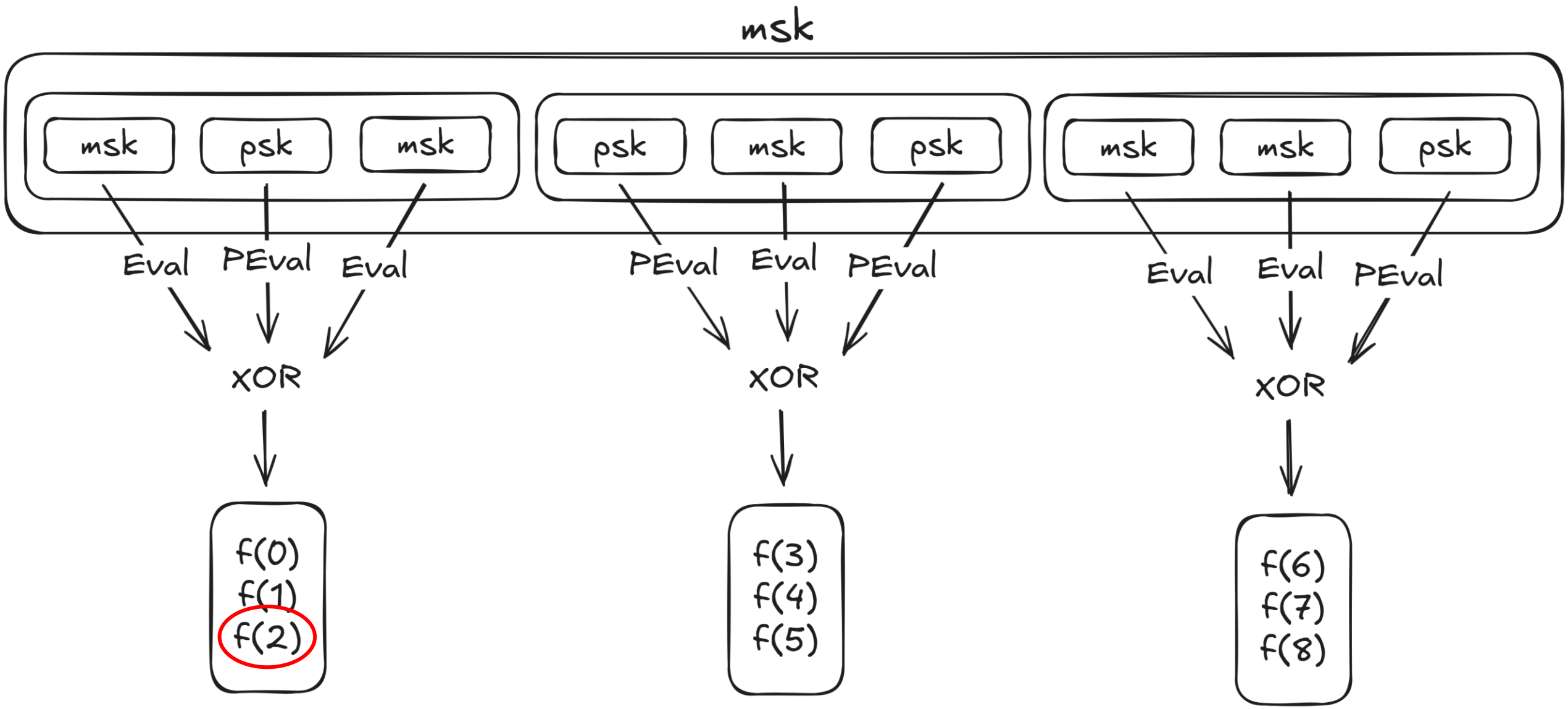
- A generic solution: flooding
  - Assign each block $\mathbf{poly}(\lambda)$ keys, each randomly normal or programmed
  - Program one of the normal keys

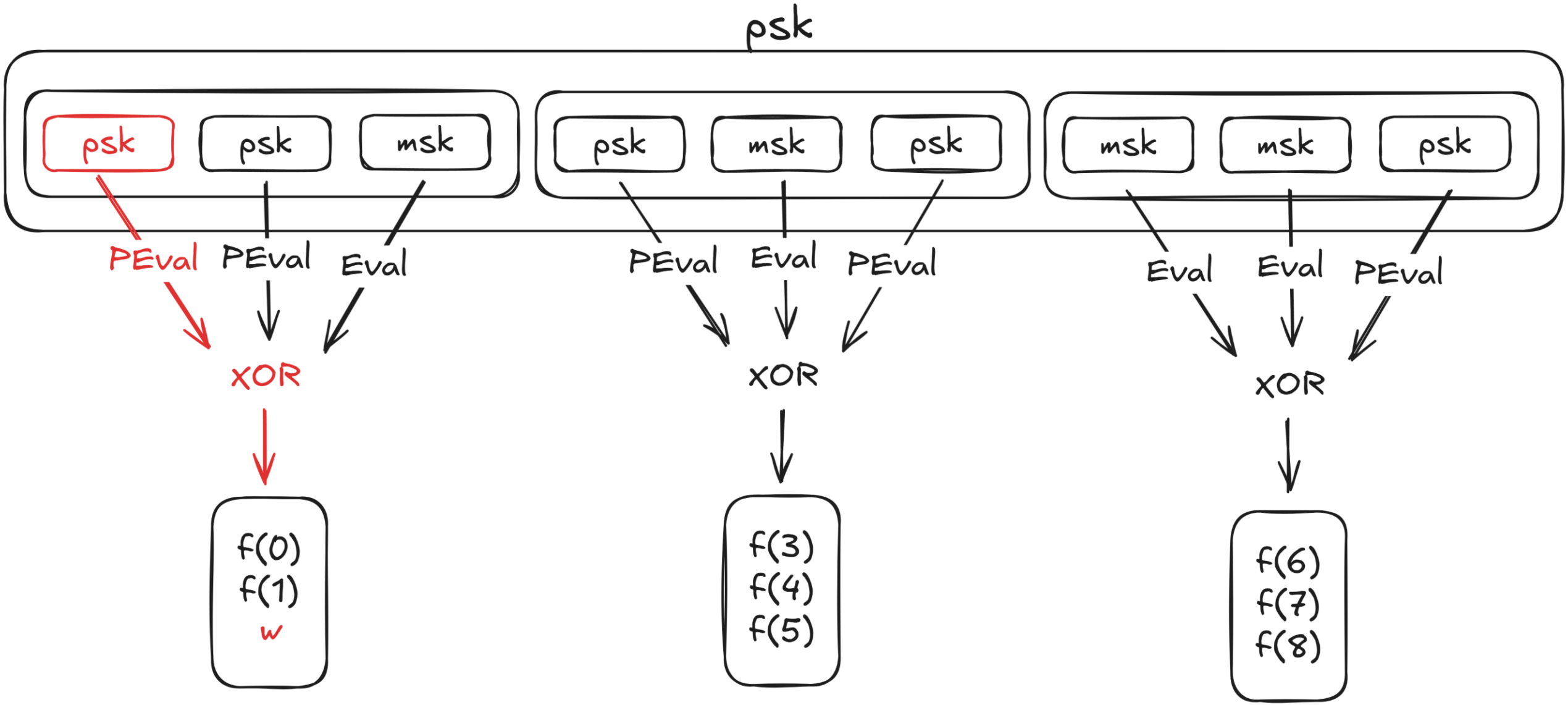
Strawman: privacy

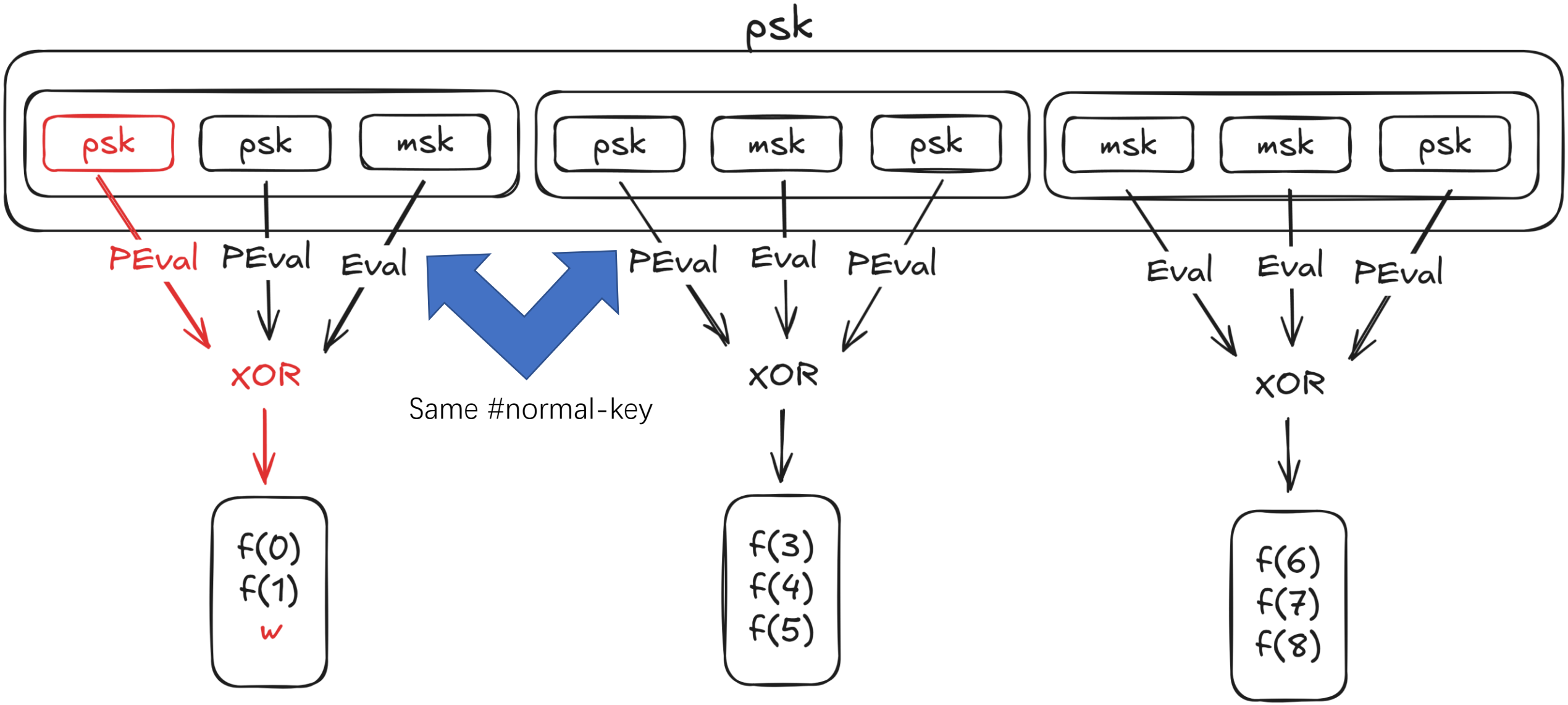
- Easily identify the programmed block!

- A generic solution: flooding
  - Assign each block $\mathbf{poly}(\lambda)$ keys, each randomly normal or programmed
  - Program one of the normal keys
  - Can’t tell which block has one more programmed key


msk
msk
psk
msk
psk
msk
psk
msk
msk
psk
Eval
PEval
Eval
PEval
Eval
PEval
Eval
Eval
PEval
XOR
XOR
XOR
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)


msk
msk
psk
msk
psk
msk
psk
msk
msk
psk
Eval
PEval
Eval
PEval
Eval
PEval
Eval
Eval
PEval
XOR
XOR
XOR
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)


psk
psk
psk
msk
psk
msk
psk
msk
msk
psk
PEval
PEval
Eval
PEval
Eval
PEval
Eval
Eval
PEval
XOR
XOR
XOR
f(0)
f(1)
w
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)


psk
psk
psk
msk
psk
msk
psk
msk
msk
psk
PEval
PEval
Eval
PEval
Eval
PEval
Eval
Eval
PEval
XOR
XOR
XOR
Same #normal-key
f(0)
f(1)
w
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)

Strawman: privacy

Strawman: privacy

- Nevertheless, some information is leaked by #normal-key.

Strawman: privacy

- Nevertheless, some information is leaked by #normal-key.

- It achieves a notion of “weak” PPPRF:

Strawman: privacy

- Nevertheless, some information is leaked by #normal-key.

- It achieves a notion of “weak” PPPRF:
- Cannot distinguish psk_y from psk_0 w.p. $> 1/\mathrm{poly}(\lambda)$

Strawman: privacy

- Nevertheless, some information is leaked by #normal-key.

- It achieves a notion of “weak” PPPRF:
- Cannot distinguish psk_y from psk_0 w.p. $> 1/\mathrm{poly}(\lambda)$
  - Relaxation of statistical distance

Strawman: privacy

- Nevertheless, some information is leaked by #normal-key.

- It achieves a notion of “weak” PPPRF:
- Cannot distinguish psk_y from psk_0 w.p. $> 1/\mathrm{poly}(\lambda)$
  - Relaxation of statistical distance
  - Intuitively, psk_y leaks y w.p. at most $1/\mathrm{poly}(\lambda)$

Strawman

- Key size: $O_\lambda(n^{1/2})$
- Eval/Prog: $O_\lambda(n^{1/2})$
- Privacy: $1/\mathbf{poly}(\lambda)$

Strawman

- Key size: $O_\lambda(n^{1/2})$
- Eval/Prog: $O_\lambda(n^{1/2})$
- Privacy: $1/\mathbf{poly}(\lambda)$

- Drawback: key size is not $\mathbf{poly}(\lambda)$

Strawman

- Key size: $O_\lambda(n^{1/2})$
- Eval/Prog: $O_\lambda(n^{1/2})$
- Privacy: $1/\mathrm{poly}(\lambda)$

- Drawback: key size is not $\mathrm{poly}(\lambda)$

- **Observation: only used $O_\lambda(1)$ keys during Eval/Prog**

Strawman

- Key size: $O_\lambda(n^{1/2})$
- Eval/Prog: $O_\lambda(n^{1/2})$
- Privacy: $1/\mathrm{poly}(\lambda)$

- Drawback: key size is not $\mathrm{poly}(\lambda)$

- **Observation: only used $O_\lambda(1)$ keys during Eval/Prog**
- Use a PPPRF to generate the keys!

Reducing the key size: Recursion!

Reducing the key size: Recursion!

- Let msk be a [BGIK22] key

Reducing the key size: Recursion!

- Let msk be a [BGIK22] key

- Top-down evaluation:

Reducing the key size: Recursion!

- Let msk be a [BGIK22] key

- Top-down evaluation:
- Level 1: msk expands into $n^{1/2}$ [BGIK22] keys

Reducing the key size: Recursion!

- Let msk be a [BGIK22] key

- Top-down evaluation:
- Level 1: msk expands into $n^{1/2}$ [BGIK22] keys
- Level 2: each key expands into $n^{1/2}$ pseudorandom values

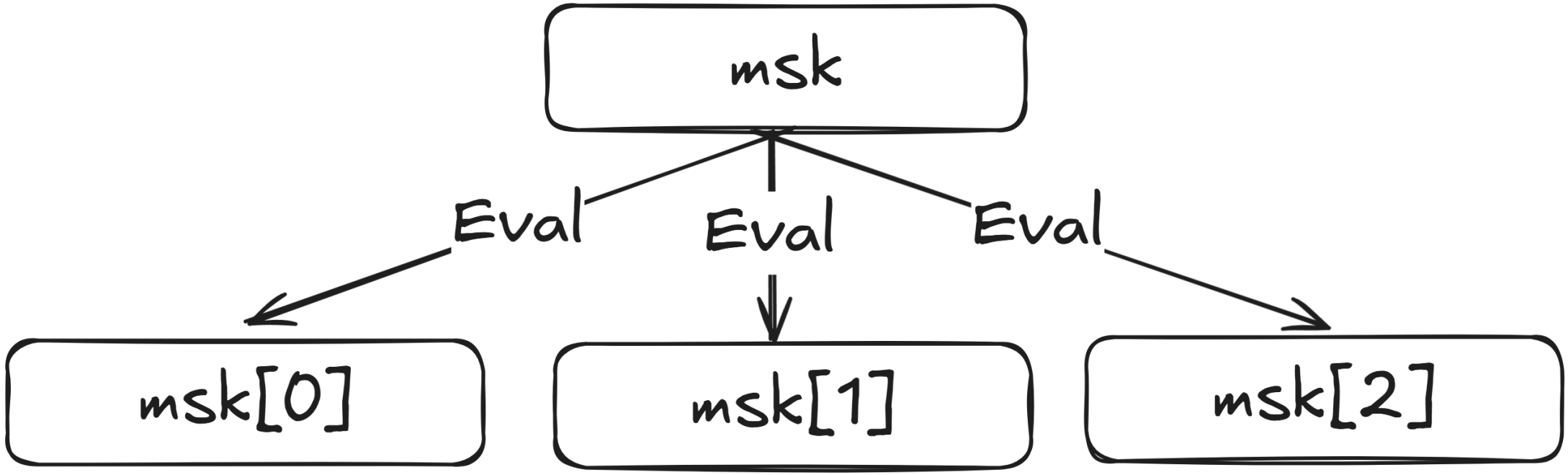
Reducing the key size: Recursion!

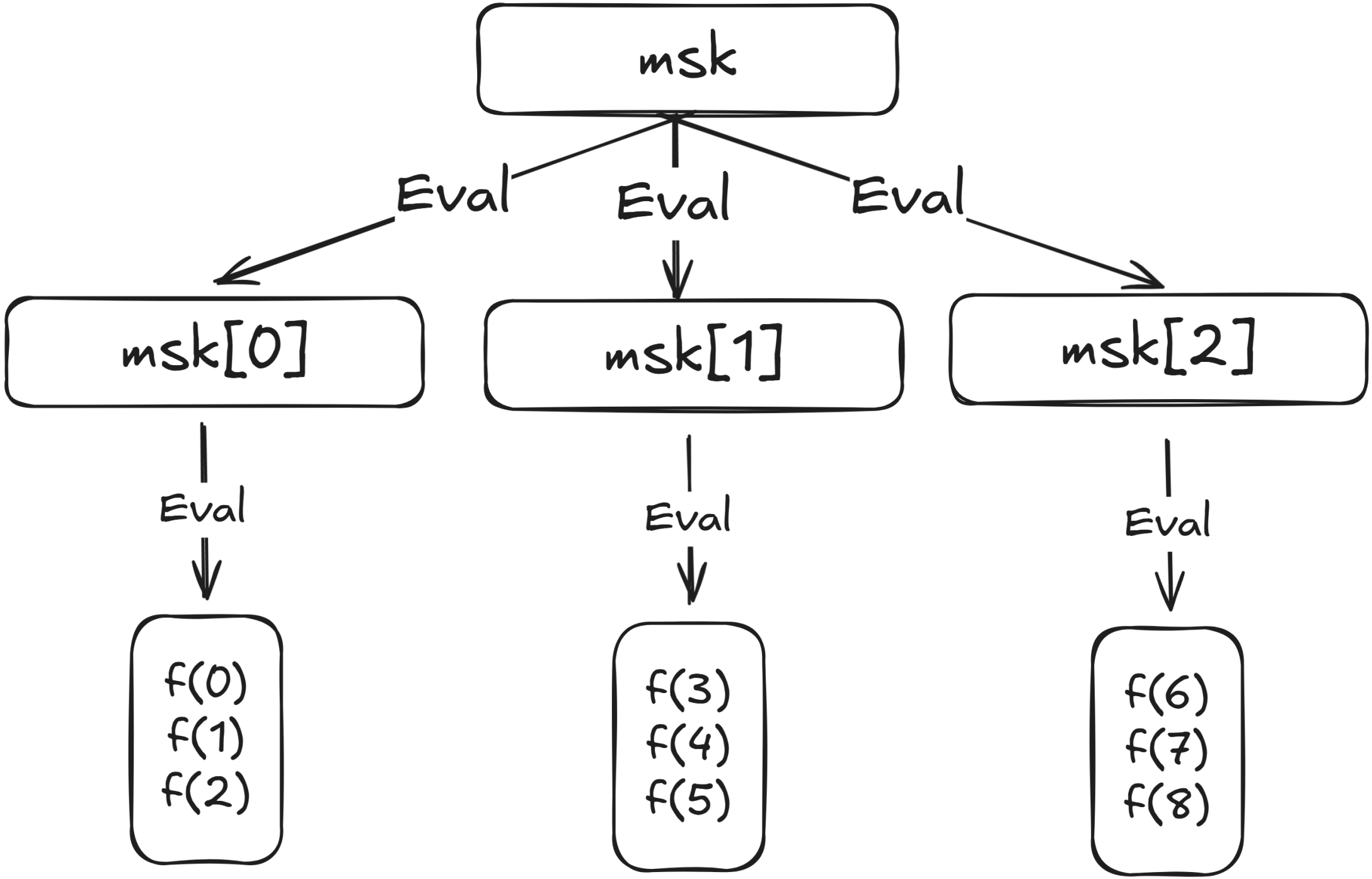
- Let msk be a [BGIK22] key

- Top-down evaluation:
- Level 1: msk expands into $n^{1/2}$ [BGIK22] keys
- Level 2: each key expands into $n^{1/2}$ pseudorandom values

- Like a GGM PRF, but with larger fan-out


msk

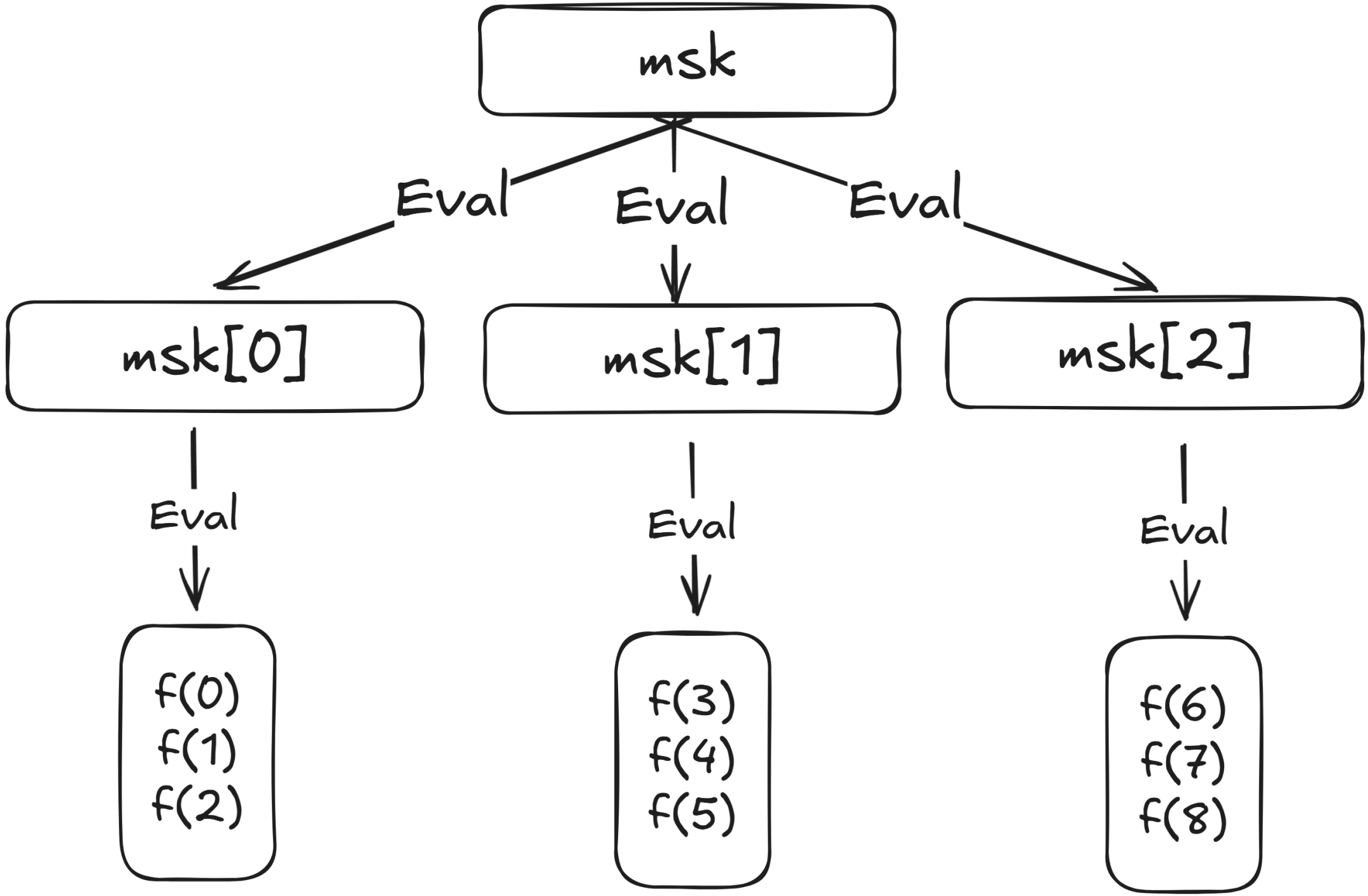

msk
Eval
Eval
Eval
msk[0]
msk[1]
msk[2]

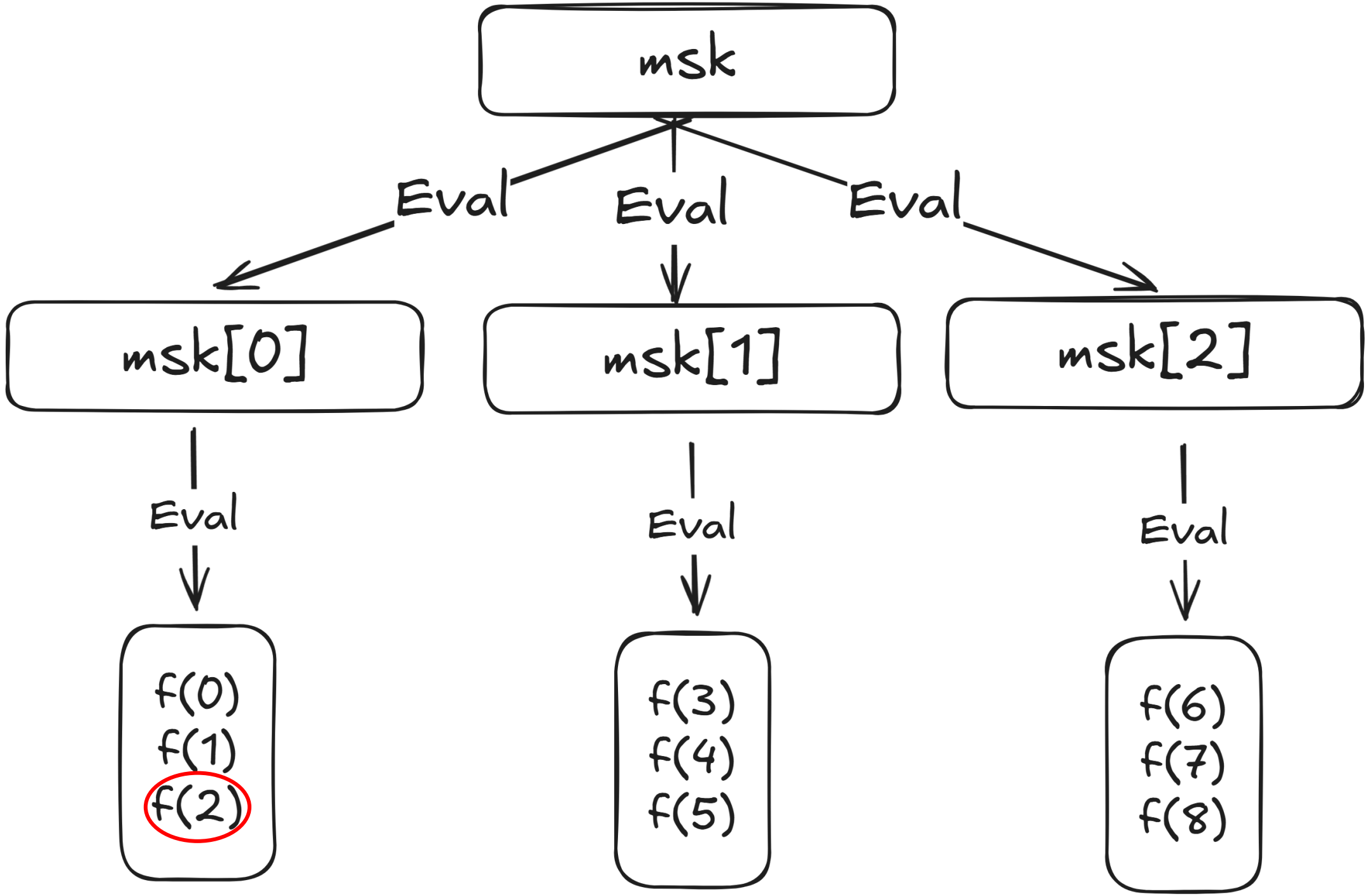

msk
Eval
Eval
Eval
msk[0]
msk[1]
msk[2]
Eval
Eval
Eval
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)

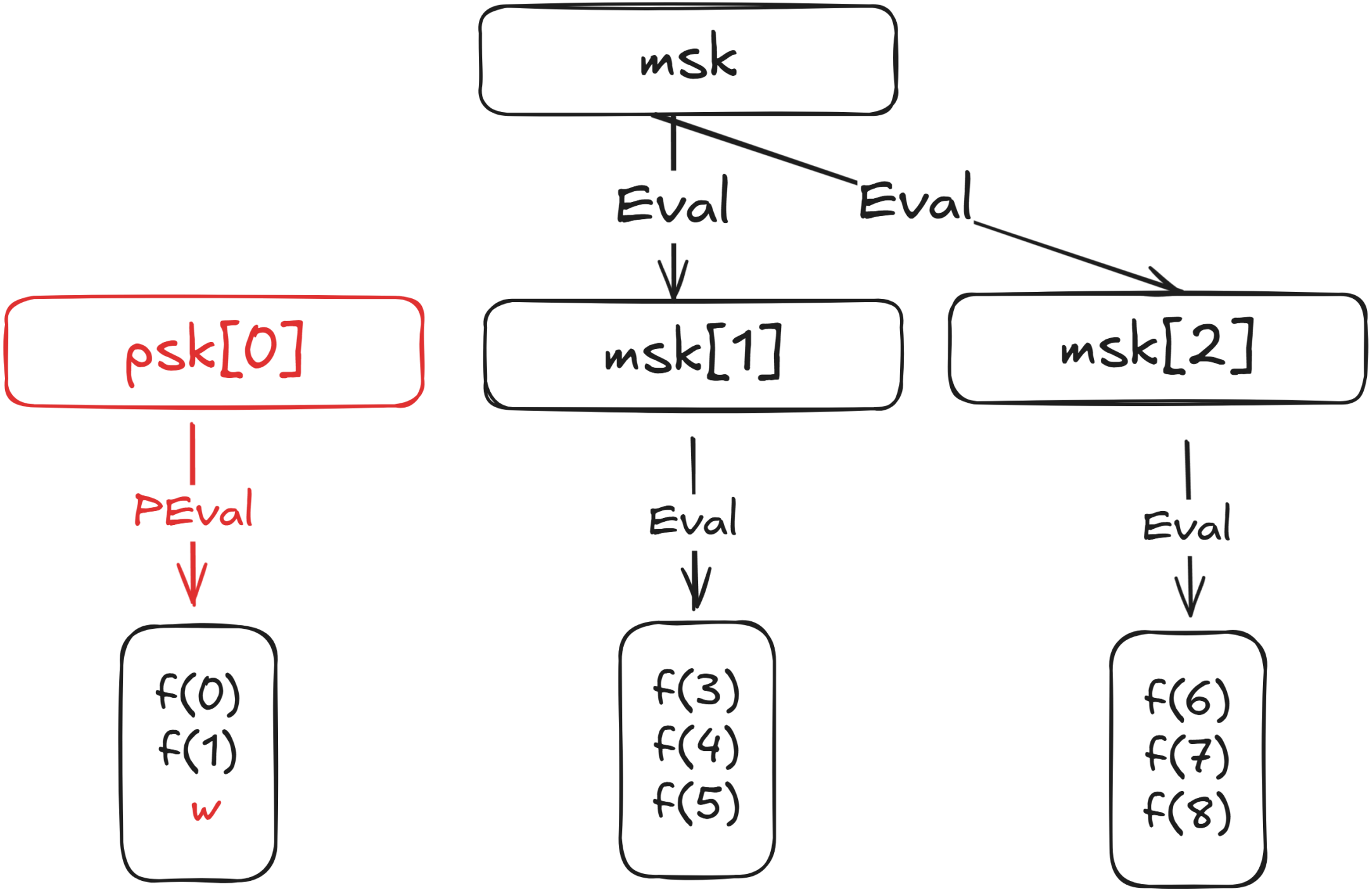
- Bottom-up programming:

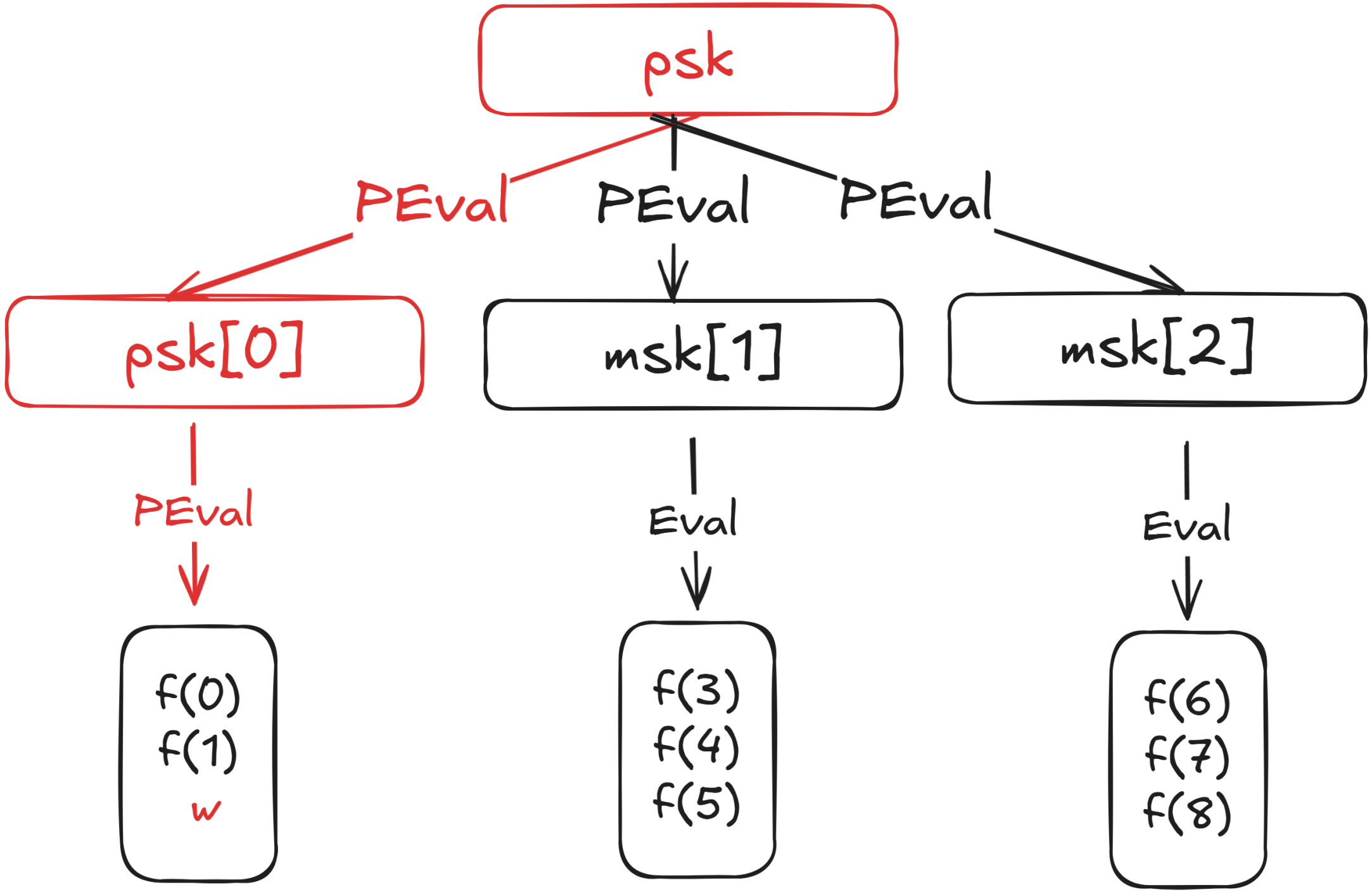
- Bottom-up programming:
- Level 2: program the level-2 key first

- Bottom-up programming:
- Level 2: program the level-2 key first
- Level 1: program msk to **output the programmed level-2 key**


msk
Eval
Eval
Eval
msk[0]
msk[1]
msk[2]
Eval
Eval
Eval
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)


msk
Eval
Eval
Eval
msk[0]
msk[1]
msk[2]
Eval
Eval
Eval
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)


msk
Eval
Eval
psk[0]
msk[1]
msk[2]
PEval
Eval
Eval
f(0)
f(1)
w
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)


psk
PEval
PEval
PEval
psk[0]
msk[1]
msk[2]
PEval
Eval
Eval
f(0)
f(1)
w
f(3)
f(4)
f(5)
f(6)
f(7)
f(8)

Reducing the key size: Recursion!

- Key size: $O_\lambda(1)$
- Eval/Prog: $O_\lambda(n^{1/2})$

Reducing the key size: Recursion!

- Key size: $O_\lambda(1)$
- Eval/Prog: $O_\lambda(n^{1/2})$

- Privacy?

Reducing the key size: Recursion!

- Key size: $O_\lambda(1)$
- Eval/Prog: $O_\lambda(n^{1/2})$

- Privacy?
- Easily identify the programmed path!

Reducing the key size: Recursion!

- Key size: $O_\lambda(1)$
- Eval/Prog: $O_\lambda(n^{1/2})$

- Privacy?
- Easily identify the programmed path!
  - Again, flooding with $O_\lambda(1)$ msk, programming one of them

Reducing the key size: Recursion!

- Key size: $O_\lambda(1)$
- Eval/Prog: $O_\lambda(n^{1/2})$

- Privacy?
- Easily identify the programmed path!
  - Again, flooding with $O_\lambda(1)$ msk, programming one of them
  - See our paper for details

Reducing the key size: Recursion!

- Key size: $O_\lambda(1)$
- Eval/Prog: $O_\lambda(n^{1/2})$

- Privacy?
- Easily identify the programmed path!
  - Again, flooding with $O_\lambda(1)$ msk, programming one of them
  - See our paper for details

- Can we do better?

We need to go deeper

- Why stop at two levels?
- No fundamental difference between 2 levels and 10 levels!

We need to go deeper

- Why stop at two levels?
- No fundamental difference between 2 levels and 10 levels!

- Key size: $O_\lambda(1)$
- Eval/Prog: $O_\lambda(n^{1/10})$

We need to go deeper

- Why stop at two levels?
- No fundamental difference between 2 levels and 10 levels!

- Key size: $O_\lambda(1)$
- Eval/Prog: $O_\lambda(n^{1/10})$

- Can we take it all the way to $\log n$ levels?

We need to go deeper

- Why stop at two levels?
- No fundamental difference between 2 levels and 10 levels!

- Key size: $O_\lambda(1)$
- Eval/Prog: $O_\lambda(n^{1/10})$

- Can we take it all the way to $\log n$ levels?
  - Sadly, no, because key size blows up exponentially

Summarize

- Assumption: OWF
- For arbitrary constant $0 < \epsilon < 1$:
  - Key size: $\lambda^{O(1/\epsilon)}$
  - Eval/Prog: $O_\lambda(n^\epsilon)$
- Security: $1/\mathrm{poly}(\lambda)$

$1/\mathrm{poly}(\lambda)$ Security

$1/\mathbf{poly}(\lambda)$ Security

- Why would anyone care about $1/\mathbf{poly}(\lambda)$ security?

$1/\mathbf{poly}(\lambda)$ Security

- Why would anyone care about $1/\mathbf{poly}(\lambda)$ security?

- Security amplification by [BGIK22]: boost to negligible

$1/\mathbf{poly}(\lambda)$ Security

- Why would anyone care about $1/\mathbf{poly}(\lambda)$ security?
- Security amplification by [BGIK22]: boost to negligible
- Apply security amplification to PIR
  - See our paper for more details

Preprocessing PIR in Minicrypt

- Assuming $O_\lambda(n^{1/2})$ storage,

	Online Comm.	Offline comm.	Computation
Prev. best [ZPSZ24]	$O_\lambda(n^{1/4})$	$O_\lambda(n^{1/2})$	$O_\lambda(n^{1/2})$
Ours	$O_\lambda(1)$	$O_\lambda(n^{1/2+\epsilon})$	$O_\lambda(n^{1/2+\epsilon})$

- where $0 < \epsilon < 1$ is an arbitrary constant.

Future directions

- Concretely efficient PPPRF, based on LWE?
- More applications of security amplification?
- More primitives based on OWF?

- Read our paper:
  - ia.cr/2025/300

p-b-p-b.github.io

bo.peng@stu.pku.edu.cn

Applying for PhD this year ☺