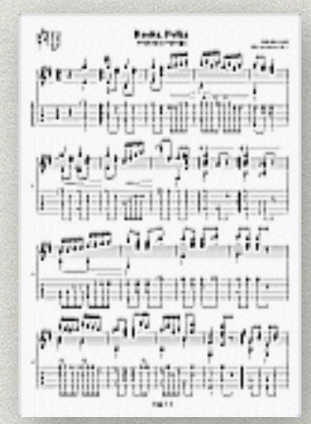
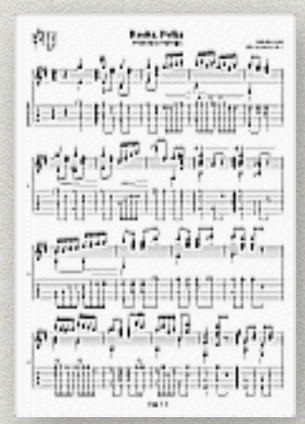
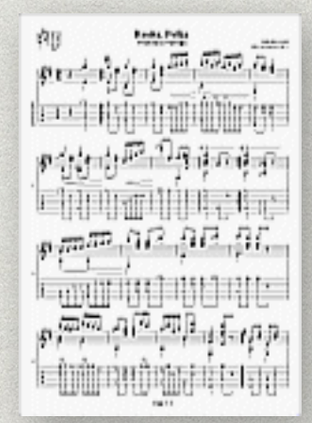
FORWARD PRIVATE SEARCHABLE ENCRYPTION

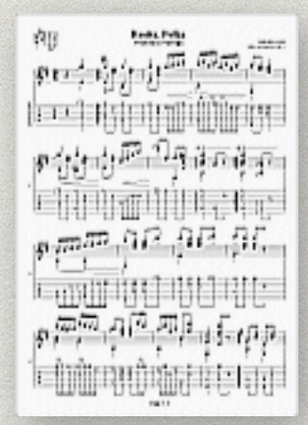
DATE 27/10/2016

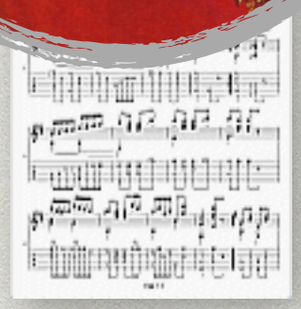
ACM CCS - RAPHAEL BOST

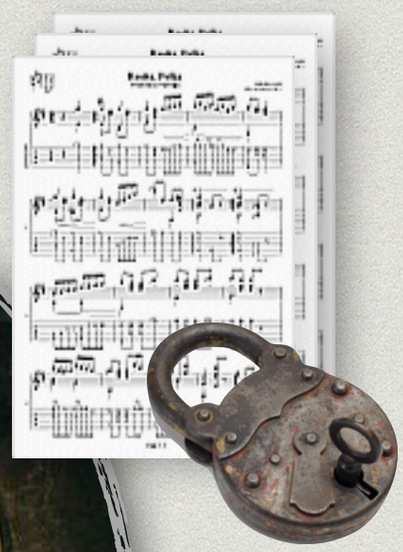


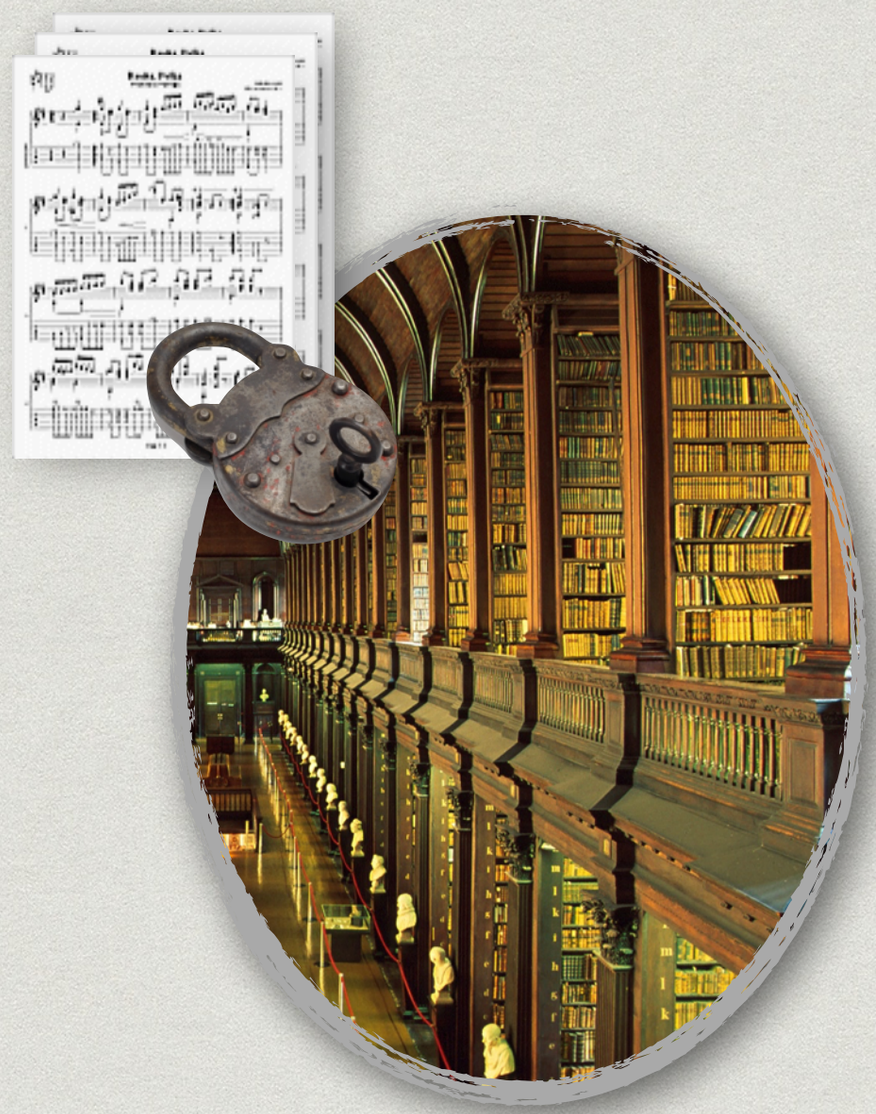


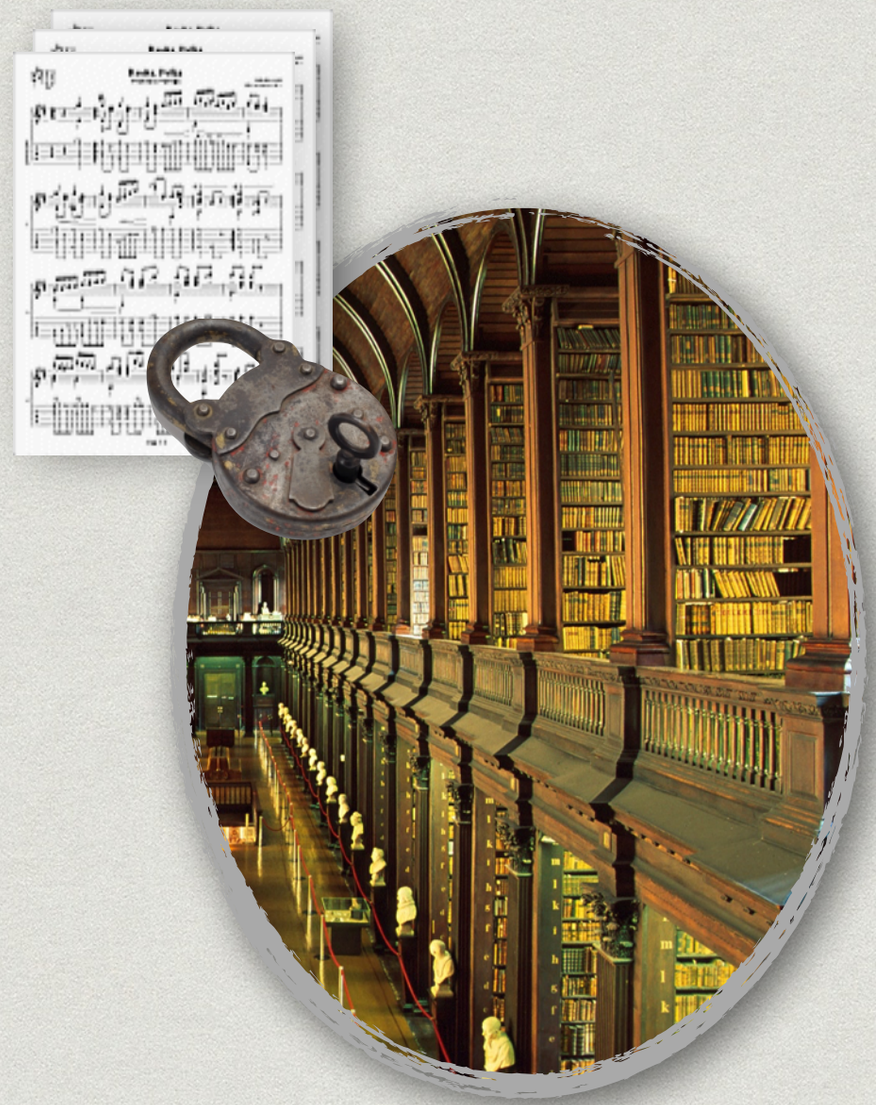


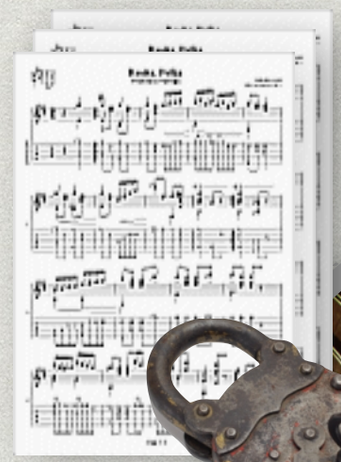


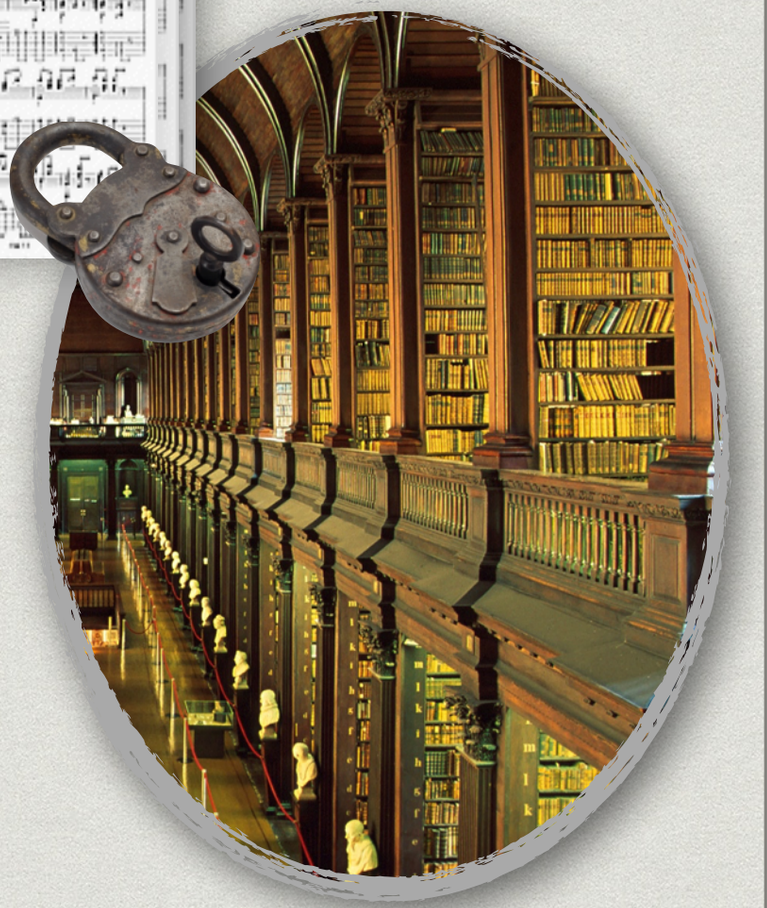


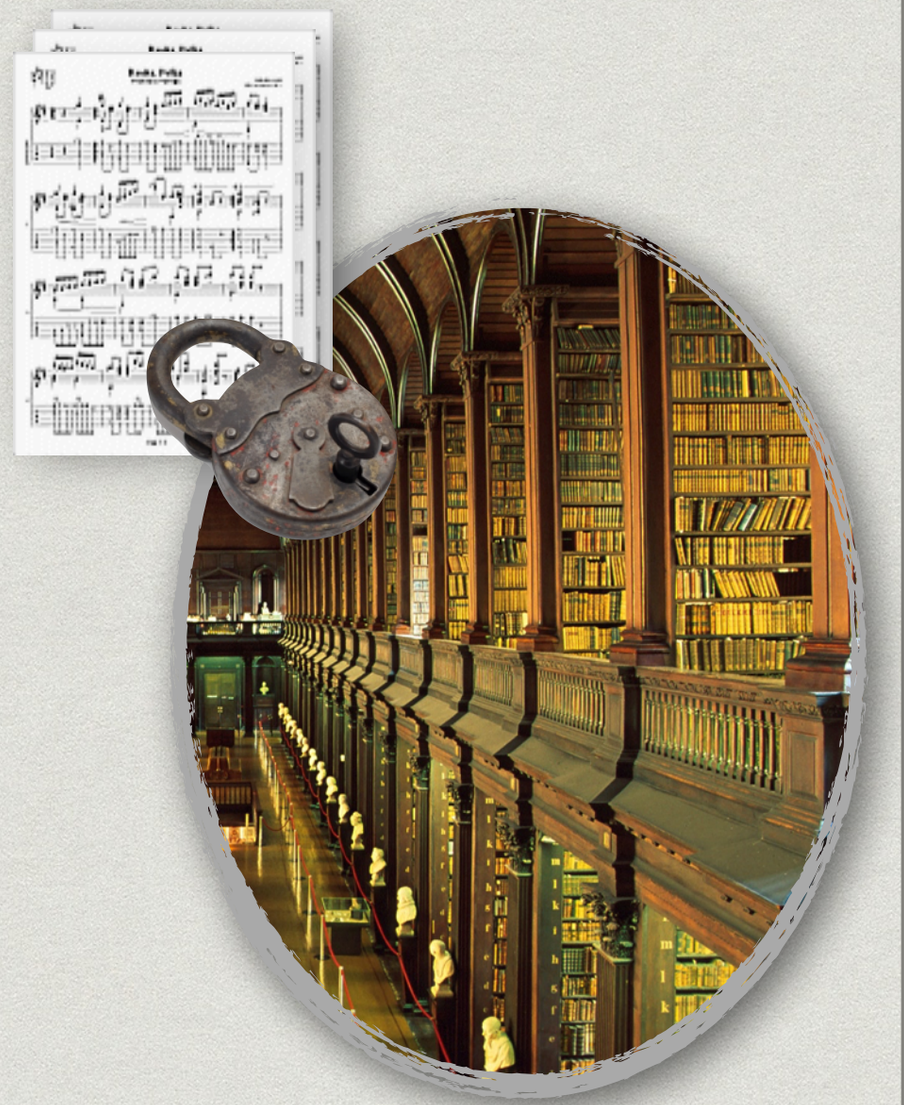


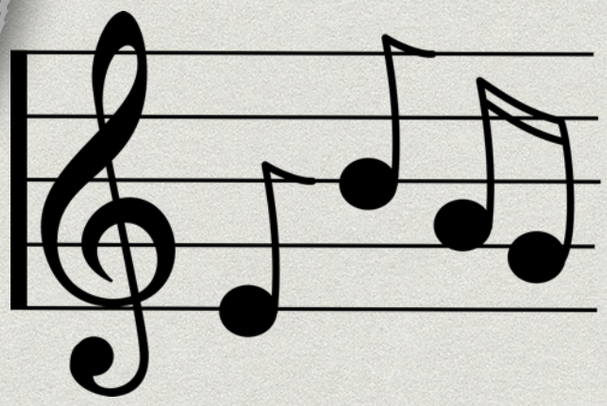


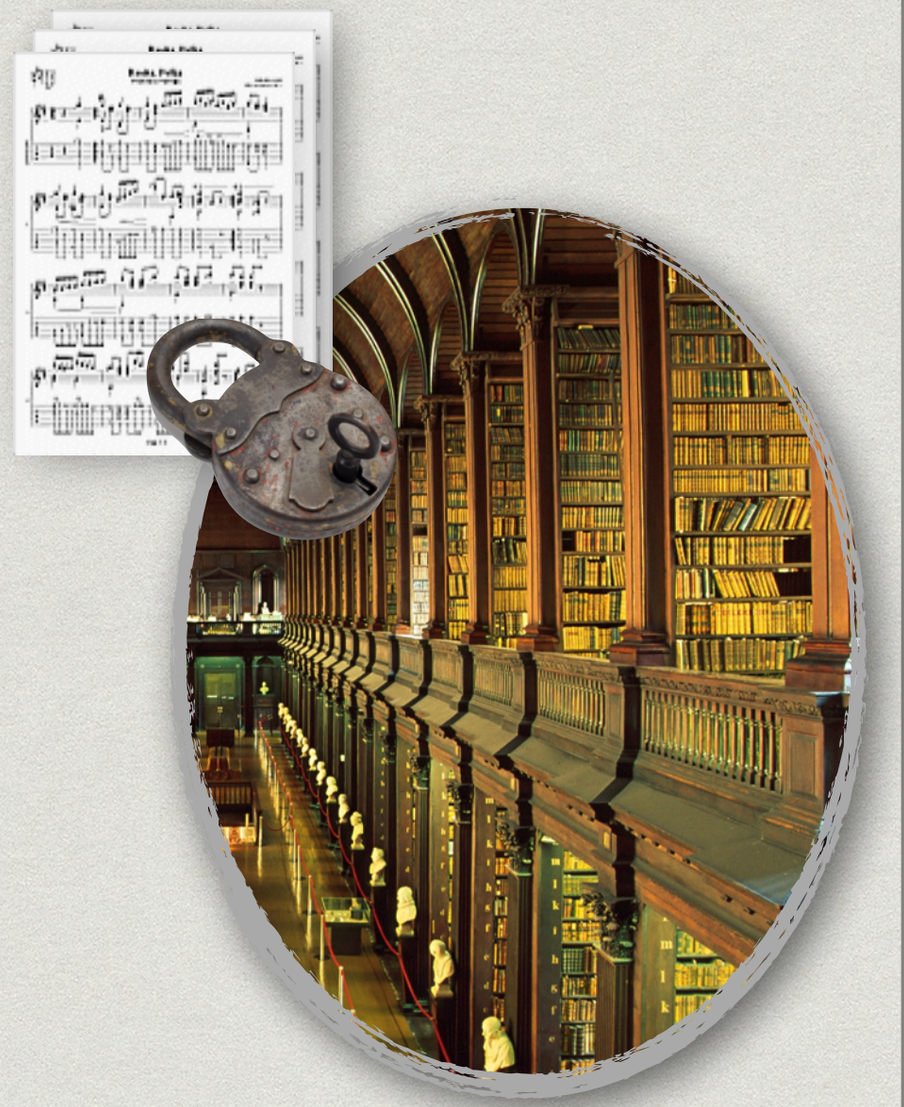












Searchable Encryption

Outsource data

Searchable Encryption

Outsource data

- securely

Searchable Encryption

Outsource data

- securely
- keep search functionalities

Searchable Encryption

Outsource data

- securely
- keep search functionalities
- aimed at efficiency

Generic Solutions

Generic Solutions

Fully Homomorphic Encryption, MPC, ORAM

- ✓ Perfect security
- ✗ Large overhead (computation, communication)

Ad-hoc Constructions

Ad-hoc Constructions

Can we get more efficient solutions?

Ad-hoc Constructions

Can we get more efficient solutions?

- Yes, but …

Ad-hoc Constructions

Can we get more efficient solutions?

- Yes, but …
- … we have to leak some information

Ad-hoc Constructions

Can we get more efficient solutions?

- Yes, but …
- … we have to leak some information

Security/performance tradeoff

Property Preserving Encryption

Property Preserving Encryption

Deterministic Encryption, OPE, ORE

Property Preserving Encryption

Deterministic Encryption, OPE, ORE

- ✓ Legacy compatible

Property Preserving Encryption

Deterministic Encryption, OPE, ORE

- ✓ Legacy compatible
- ✓ Very Efficient

Property Preserving Encryption

Deterministic Encryption, OPE, ORE

- ✓ Legacy compatible
- ✓ Very Efficient
- ✗ Not secure in practice (*cf*. next talk)

Index-Based SE [CGKO'06]

Index-Based SE [CGKO'06]

Encryption of the reversed index

Index-Based SE [CGKO'06]

Encryption of the reversed index

- Search leakage :
  - repetition of queries

Index-Based SE [CGKO'06]

Encryption of the reversed index

- Search leakage :
  - repetition of queries
- Update leakage:
  - updated documents
  - repetition of updated keywords

File Injection Attacks [ZKP'16]

Non-adaptive file injection attacks

- Insert purposely crafted documents in the DB. Use binary search to recover the query

D_1	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
D_2	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
D_3	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8

File Injection Attacks [ZKP'16]

Non-adaptive file injection attacks

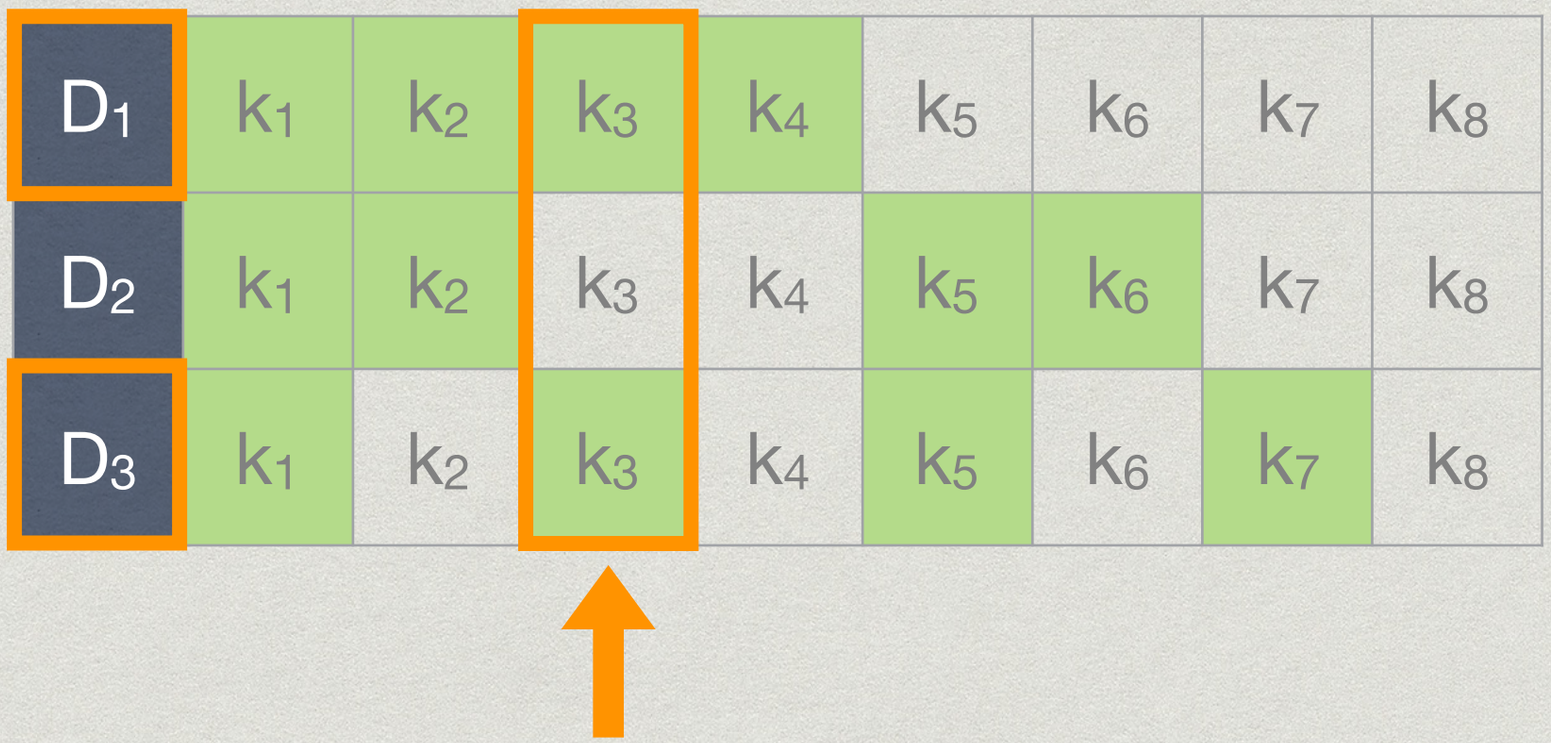
- Insert purposely crafted documents in the DB. Use binary search to recover the query

D_1	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
D_2	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
D_3	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8

File Injection Attacks [ZKP'16]

Non-adaptive file injection attacks

- Insert purposely crafted documents in the DB. Use binary search to recover the query



File Injection Attacks [ZKP'16]

Non-adaptive file injection attacks

- Insert purposely crafted documents in the DB. Use binary search to recover the query

D_1	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
D_2	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
D_3	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8

log K injected documents

File Injection Attacks [ZKP'16]

- Insert purposely crafted documents.
  Use binary search to recover the query

  ➡ Counter measure:
  no more than T kw./doc.

  (K/T) · log T injected documents

- Adaptive version of the attack

  (K/T) + log T injected documents

File Injection Attacks [ZKP'16]

- Insert purposely crafted documents.
  Use binary search to recover the query

  ➡ Counter measure:
  no more than T kw./doc.

  (K/T) · log T injected documents

- Adaptive version of the attack

  (K/T) + log T injected documents

  log T with prior knowledge

Adaptive File Injection

- The adaptive attack uses the update leakage:

Adaptive File Injection

- The adaptive attack uses the update leakage:
  - Most SE schemes leak if a newly inserted document matches a previous query

Adaptive File Injection

- The adaptive attack uses the update leakage:
  - Most SE schemes leak if a newly inserted document matches a previous query
  - We need SE schemes with oblivious updates

Adaptive File Injection

- The adaptive attack uses the update leakage:
  - Most SE schemes leak if a newly inserted document matches a previous query
  - We need SE schemes with oblivious updates

Forward Privacy

Forward Privacy

- Forward private: an update does not leak any information on the updated keywords

Forward Privacy

- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB

Forward Privacy

- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB
- Only one existing scheme so far [SPS'14]

Forward Privacy

- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB
- Only one existing scheme so far [SPS'14]

  ➡ ORAM-like construction

Forward Privacy

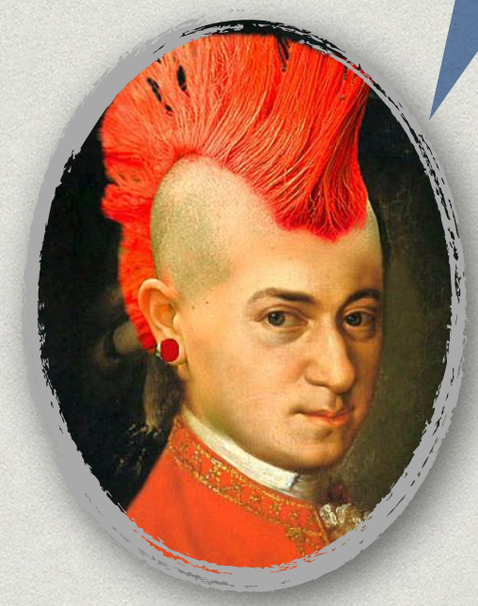
- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB
- Only one existing scheme so far [SPS'14]
  - ➡ ORAM-like construction
  - ✗ Inefficient updates

Forward Privacy

- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB
- Only one existing scheme so far [SPS'14]
  - ➡ ORAM-like construction
  - ✗ Inefficient updates
  - ✗ Large client storage

Σοφος

- Forward private index-based scheme
- Low search and update overhead
- Simpler than [SPS’14]



Add (ind_1,…,ind_c) to w

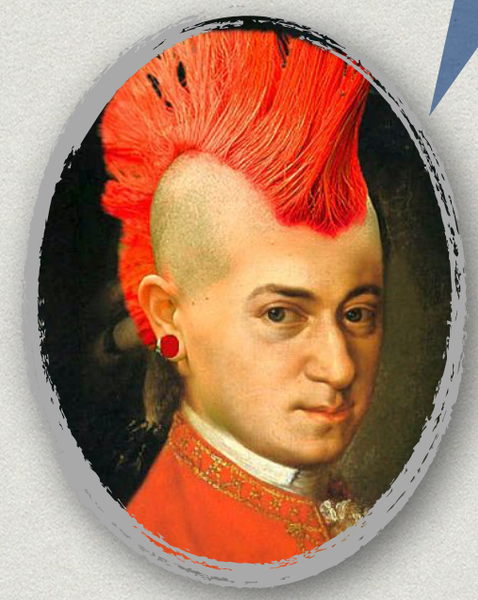
Add (ind_1,…,ind_c) to w

$UT_1(w)$

$UT_2(w)$

…

$UT_c(w)$



Add (ind_1,…,ind_c) to w

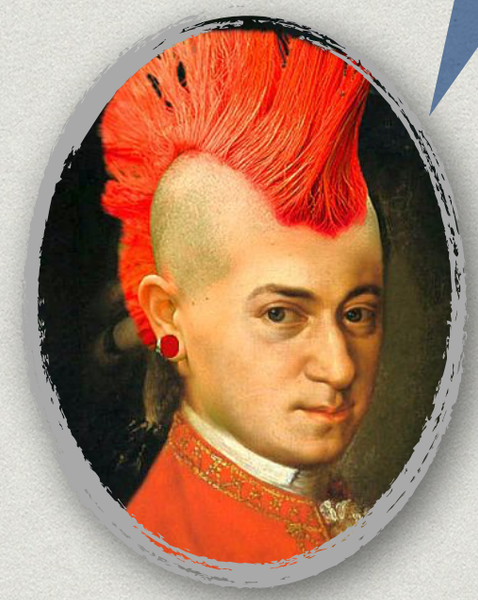
Search w

$UT_1(w)$

$UT_2(w)$

…

$UT_c(w)$



Add (ind_1,…,ind_c) to w

Search w

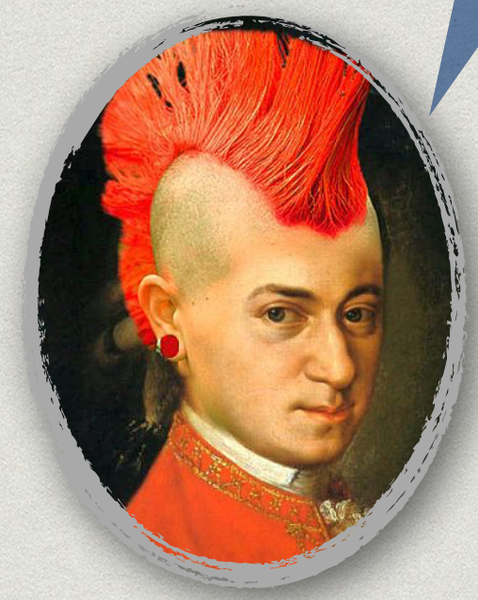
$UT_1(w)$

$UT_2(w)$

…

$UT_c(w)$

$ST(w)$



Add (ind_1,…,ind_c) to w

Search w

$UT_1(w)$

$UT_2(w)$

…

$UT_c(w)$

ST(w)

Add $(ind_1,\ldots,ind_c)$ to w

Search w

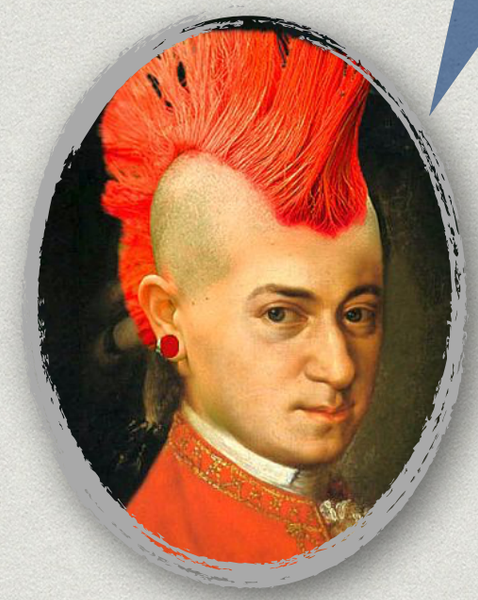
$UT_1(w)$

$UT_2(w)$

...

$UT_c(w)$

$ST(w)$



Add (ind_1,…,ind_c) to w

Search w

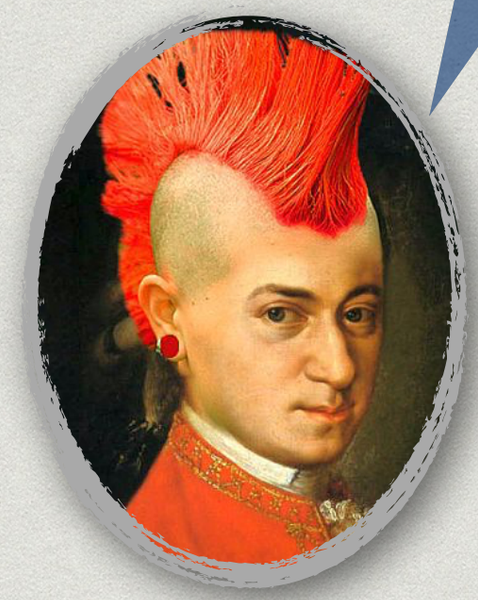
$UT_1(w)$

$UT_2(w)$

…

$UT_c(w)$

$ST(w)$



Add $(ind_1,\ldots,ind_c)$ to w

Search w

$UT_1(w)$ $UT_2(w)$ … $UT_c(w)$

$ST_1(w)$ $ST_2(w)$ … $ST_c(w)$

Add $(ind_1,\ldots,ind_c)$ to w

Search w

Add ind_{c+1} to w

$UT_1(w)$ $UT_2(w)$ … $UT_c(w)$

$ST_1(w)$ $ST_2(w)$ … $ST_c(w)$

Add $(ind_1,\ldots,ind_c)$ to w

Search w

Add ind_{c+1} to w

$UT_1(w)$ $UT_2(w)$ … $UT_c(w)$ $UT_{c+1}(w)$

$ST_1(w)$ $ST_2(w)$ … $ST_c(w)$

Add $(ind_1,\ldots,ind_c)$ to w

Search w

Add ind_{c+1} to w

$UT_1(w)$ $UT_2(w)$ … $UT_c(w)$ $UT_{c+1}(w)$

$ST_1(w)$ $ST_2(w)$ … $ST_c(w)$ $ST_{c+1}(w)$

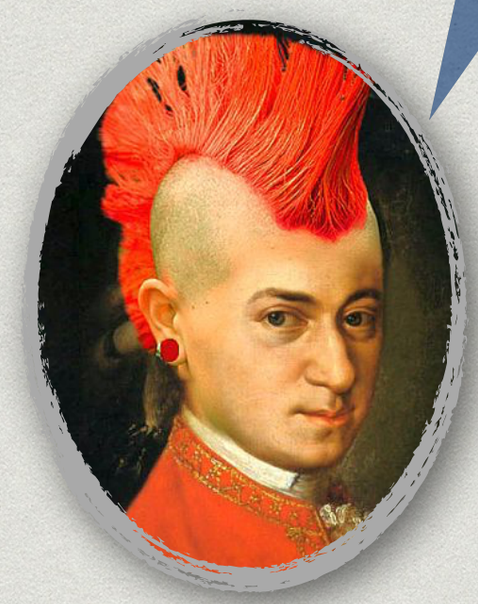
Add $(ind_1,\ldots,ind_c)$ to w

Search w

Add ind_{c+1} to w

$UT_1(w)$ $UT_2(w)$ … $UT_c(w)$ $UT_{c+1}(w)$

$ST_1(w)$ $ST_2(w)$ … $ST_c(w)$ $ST_{c+1}(w)$



Add $(ind_1,\ldots,ind_c)$ to w

Search w

Add ind_{c+1} to w

$UT_1(w)$ $UT_2(w)$ … $UT_c(w)$ $UT_{c+1}(w)$

$ST_1(w)$ $ST_2(w)$ … $ST_c(w)$ $ST_{c+1}(w)$

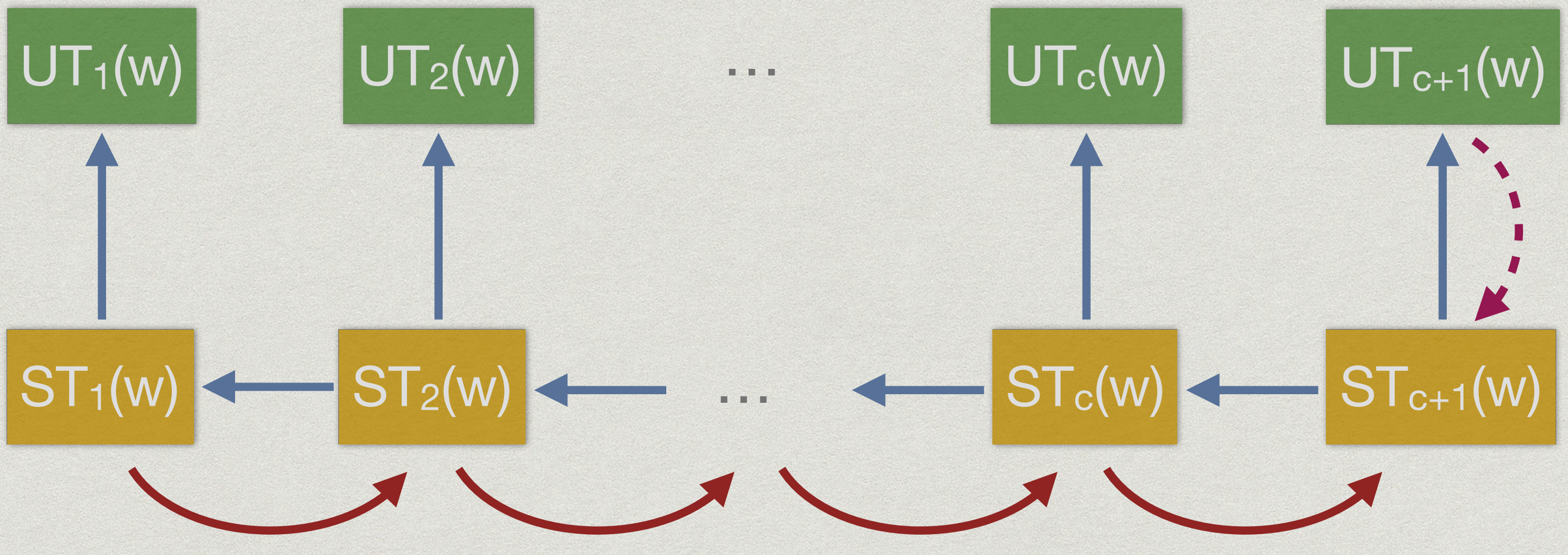
Add $(ind_1,\ldots,ind_c)$ to w

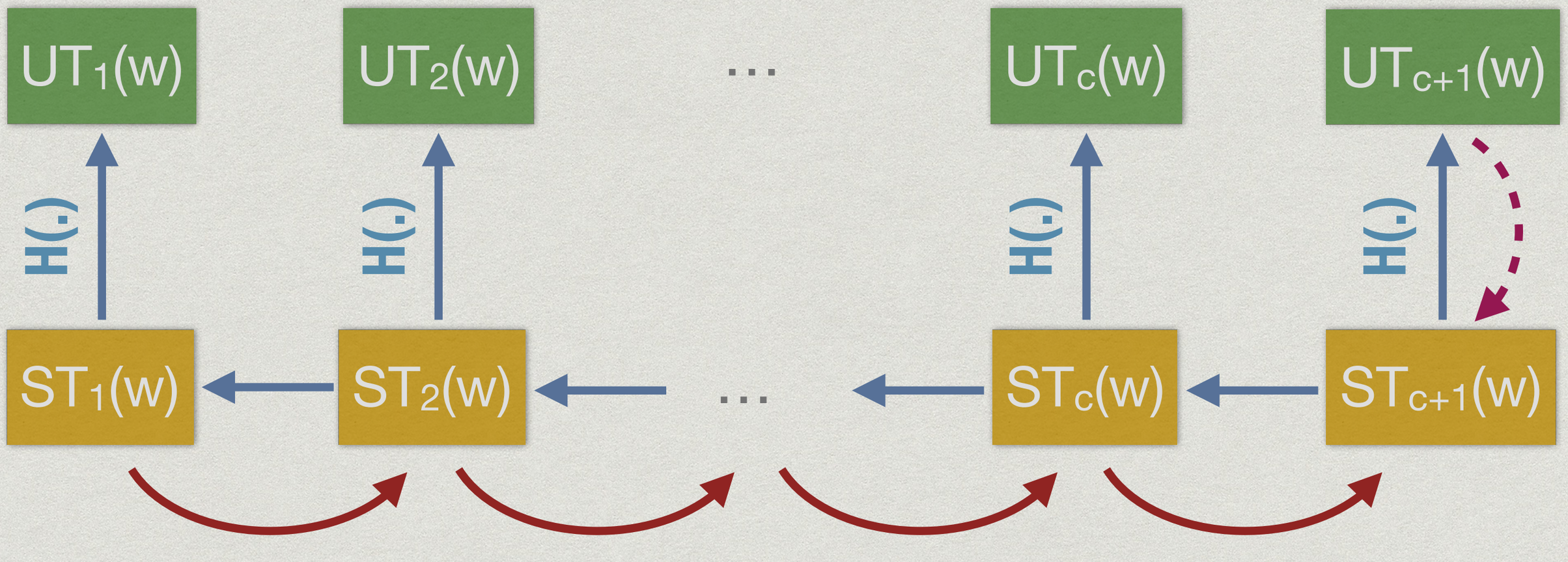
Search w

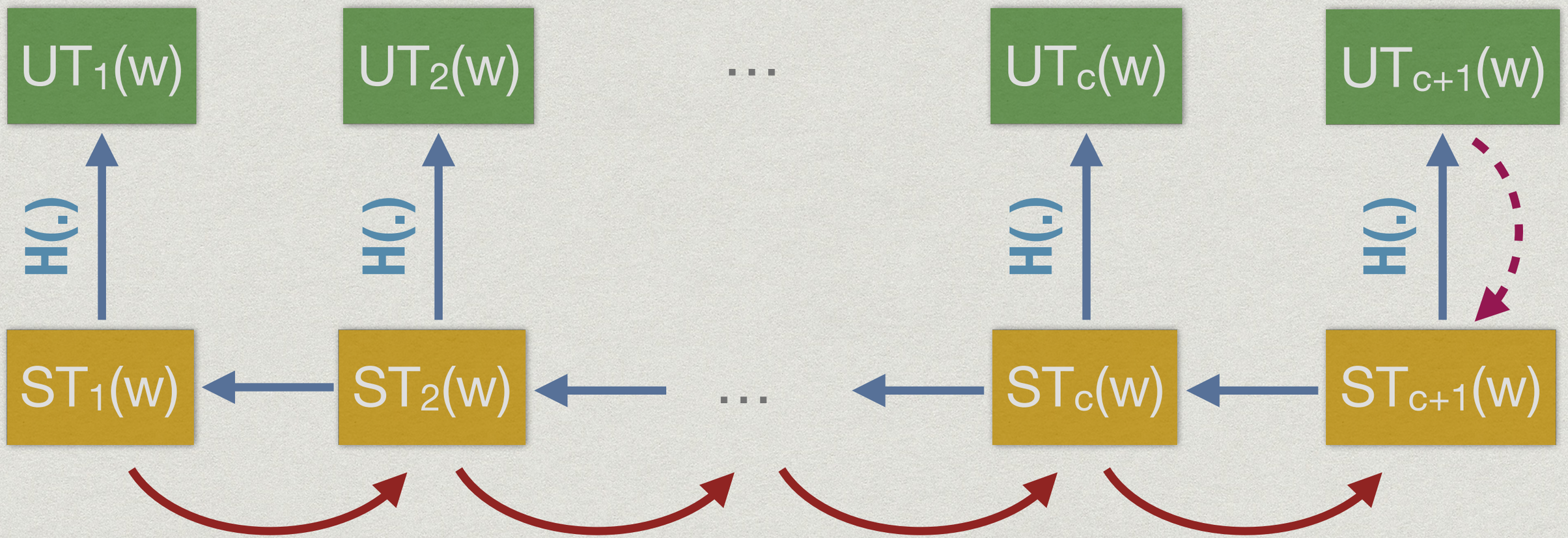
Add ind_{c+1} to w

$UT_1(w)$ $UT_2(w)$ … $UT_c(w)$ $UT_{c+1}(w)$

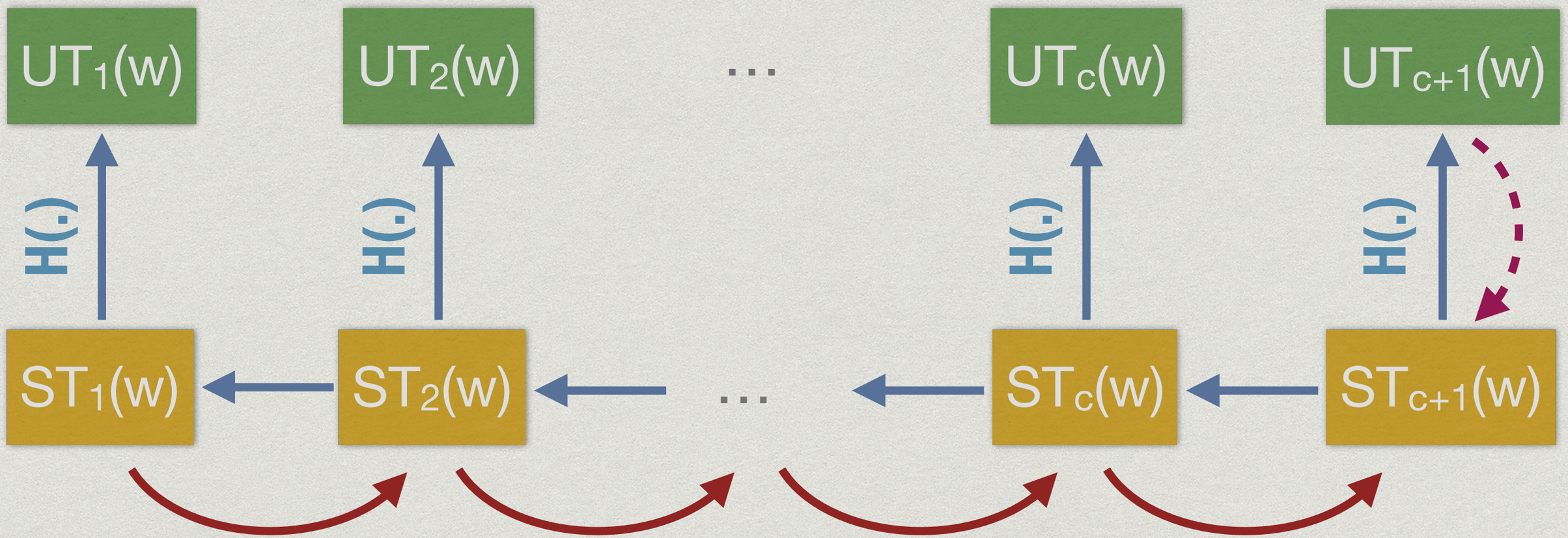
$ST_1(w)$ $ST_2(w)$ … $ST_c(w)$ $ST_{c+1}(w)$


UT1(w)
UT2(w)
...
UTc(w)
UTc+1(w)
ST1(w)
ST2(w)
...
STc(w)
STc+1(w)

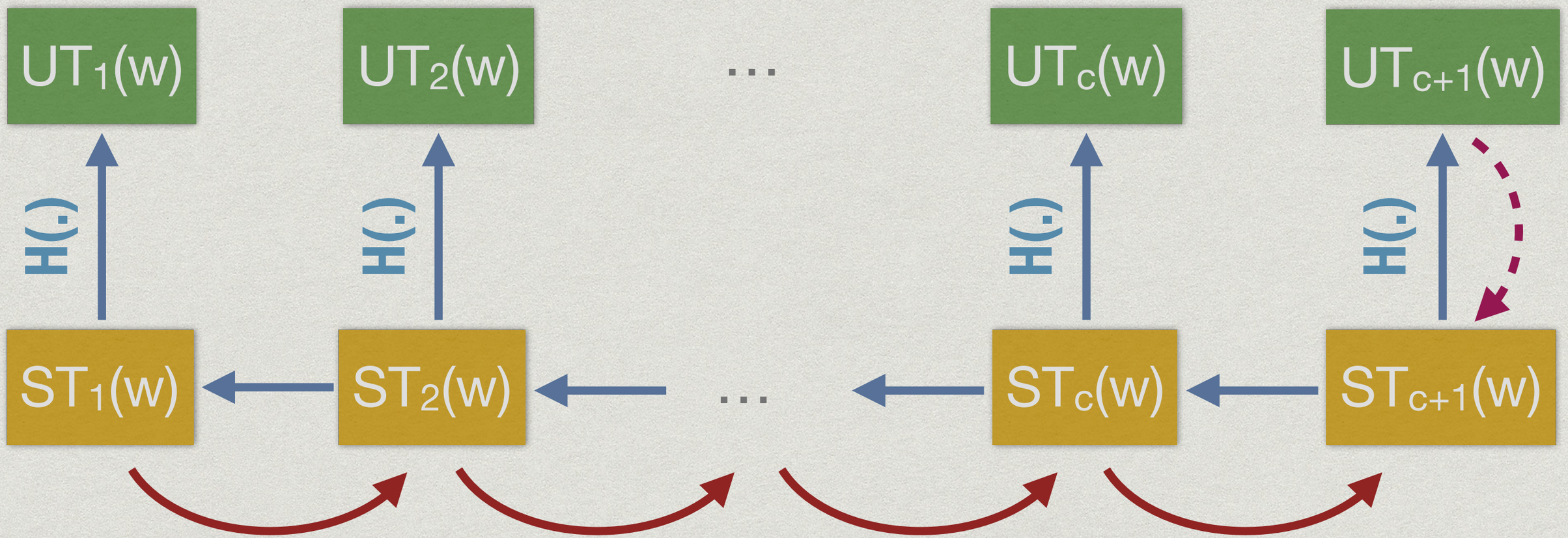

UT1(w)
UT2(w)
...
UTc(w)
UTc+1(w)
H(·)
H(·)
H(·)
H(·)
ST1(w)
ST2(w)
...
STc(w)
STc+1(w)



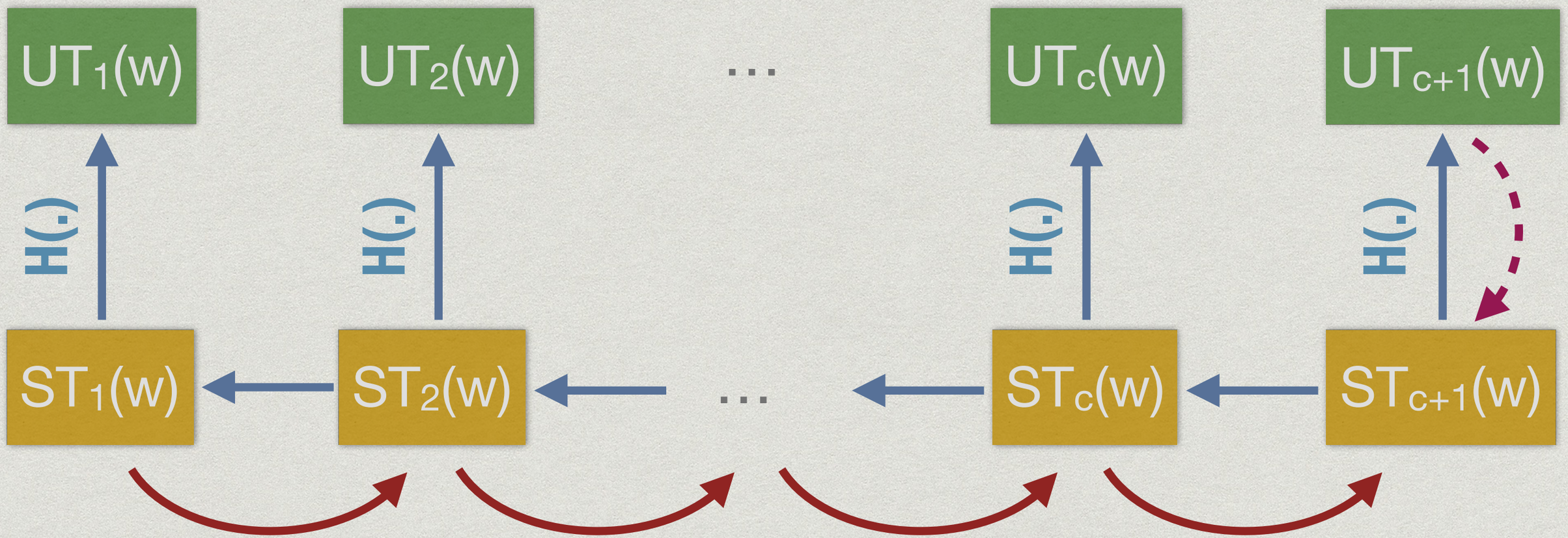
- Naïve solution: $ST_i(w) = F(K_w, i)$



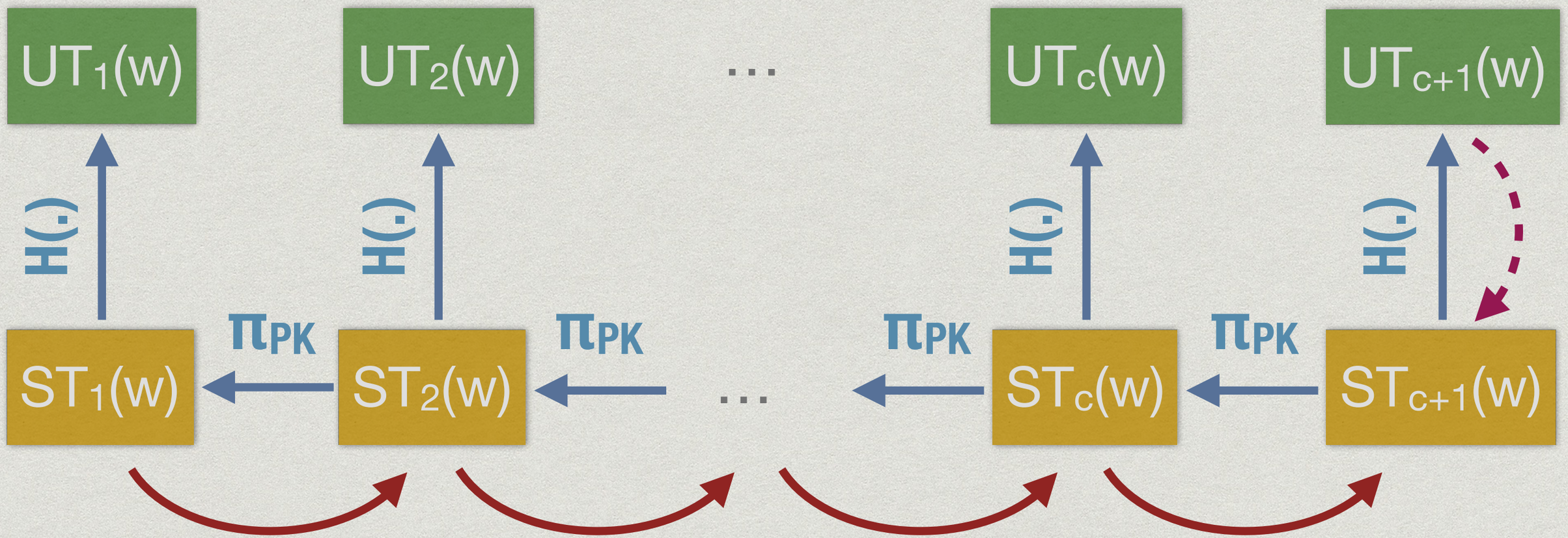
- Naïve solution: $ST_i(w) = F(K_w, i)$
  - ✗ Sending only K_w is not forward private



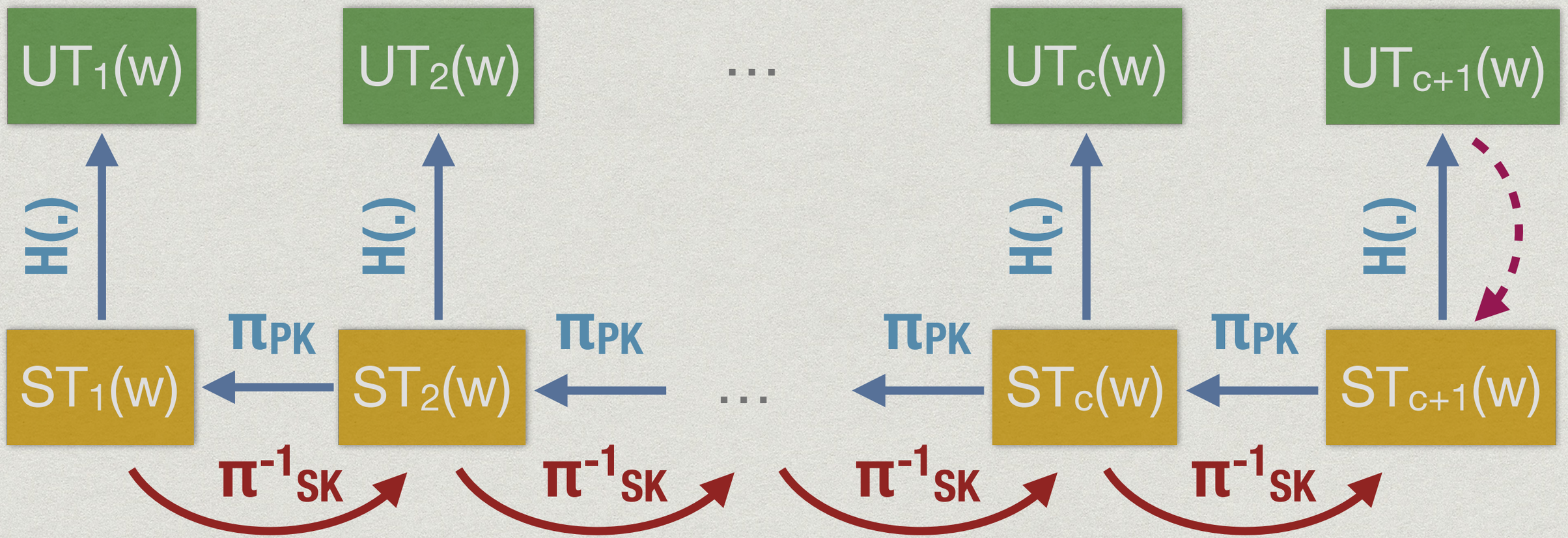
- Naïve solution: $ST_i(w) = F(K_w, i)$
  - ✗ Sending only K_w is not forward private
  - ✗ O/W client needs to send c tokens



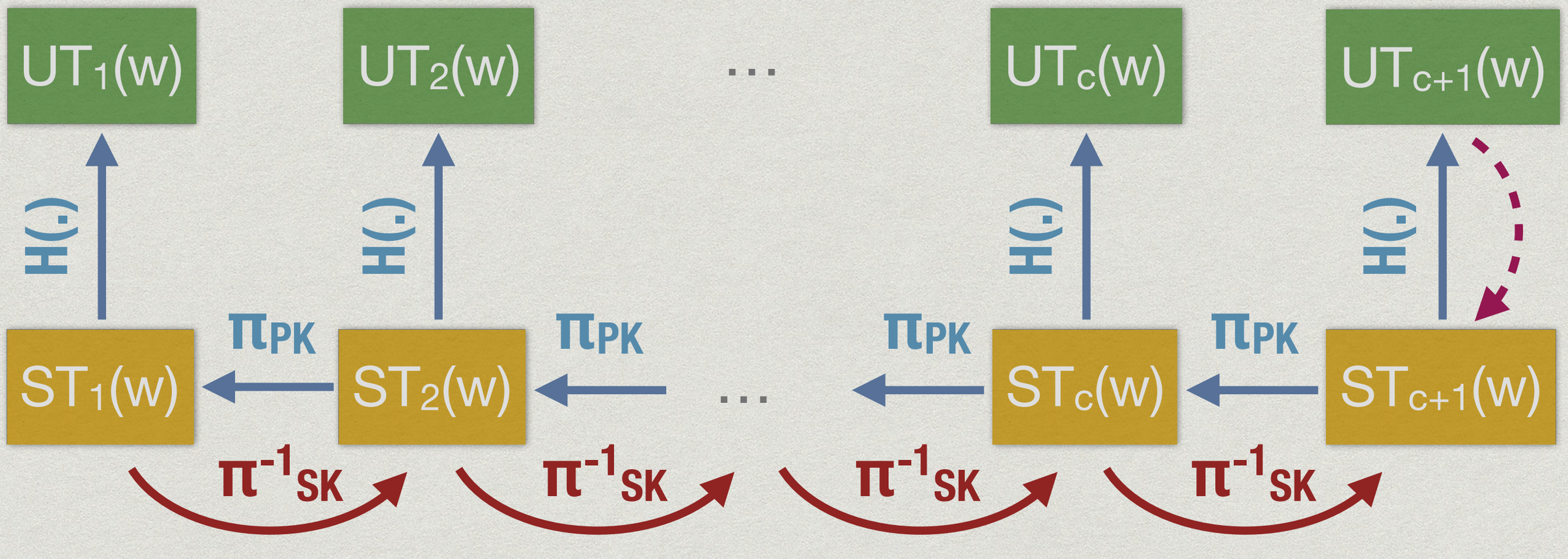
- Naïve solution: $ST_i(w) = F(K_w, i)$
  - ✗ Sending only K_w is not forward private
  - ✗ O/W client needs to send c tokens
- Use a trapdoor permutation

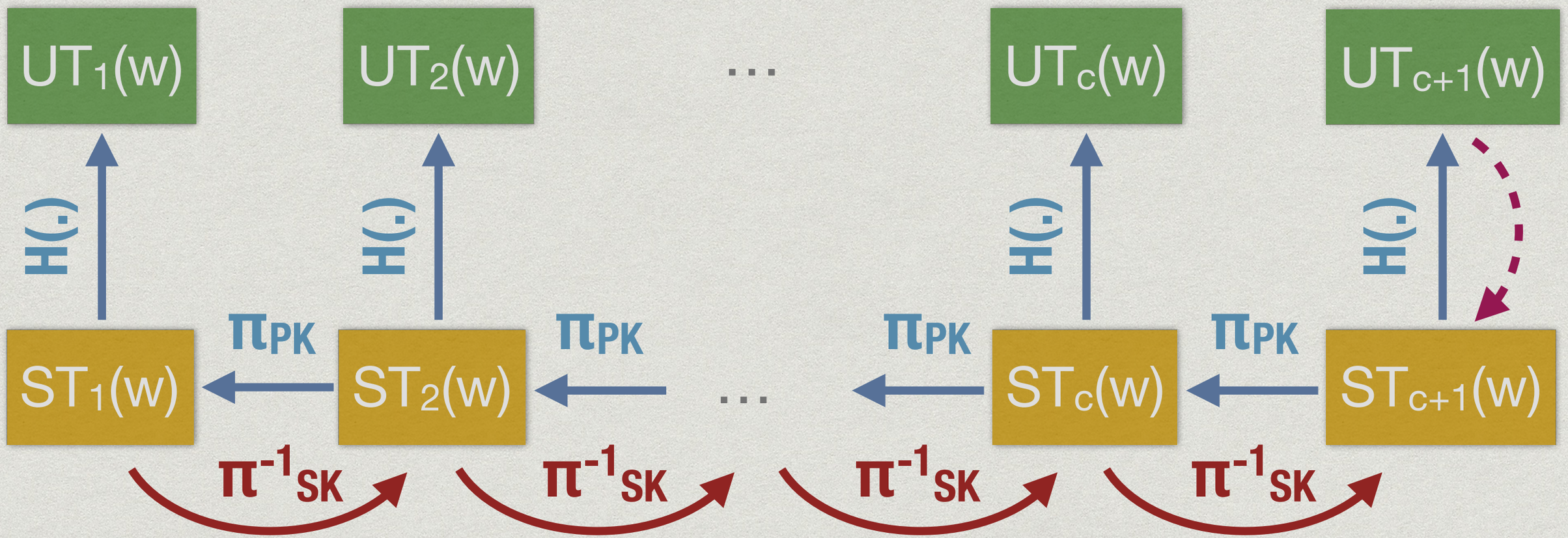


- Naïve solution: $ST_i(w) = F(K_w, i)$
  - ✗ Sending only K_w is not forward private
  - ✗ O/W client needs to send c tokens
- Use a trapdoor permutation

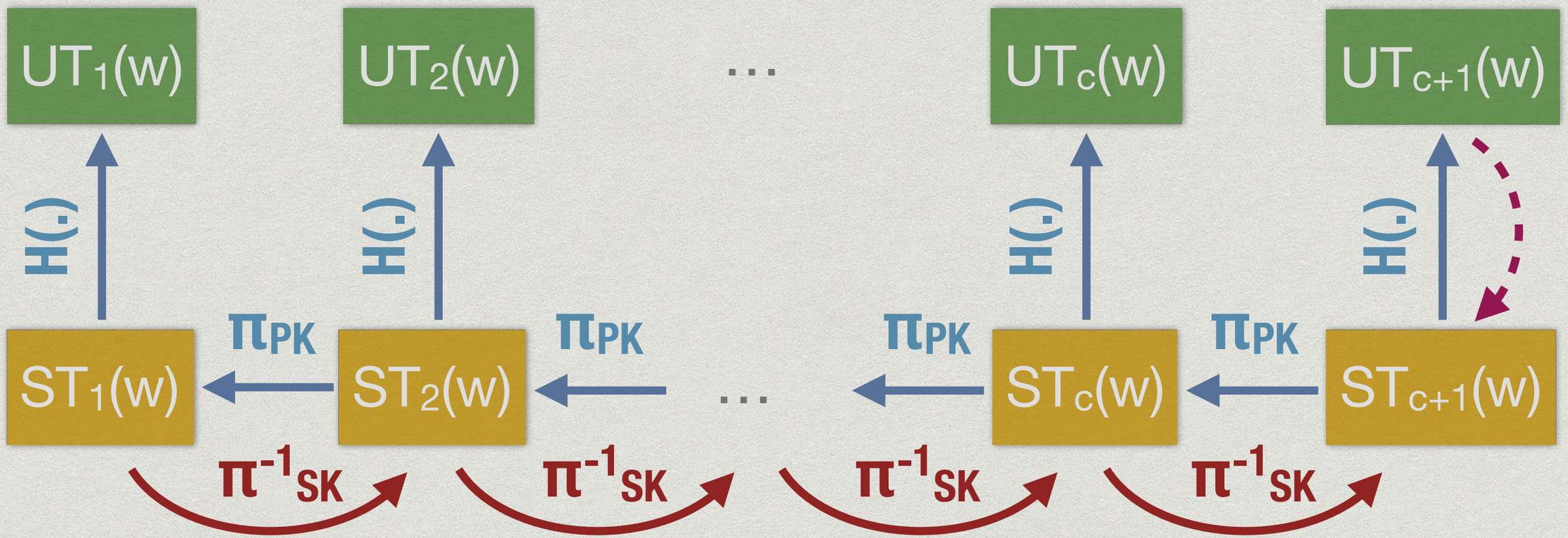


- Naïve solution: $ST_i(w) = F(K_w, i)$
  - ✗ Sending only K_w is not forward private
  - ✗ O/W client needs to send c tokens
- Use a trapdoor permutation

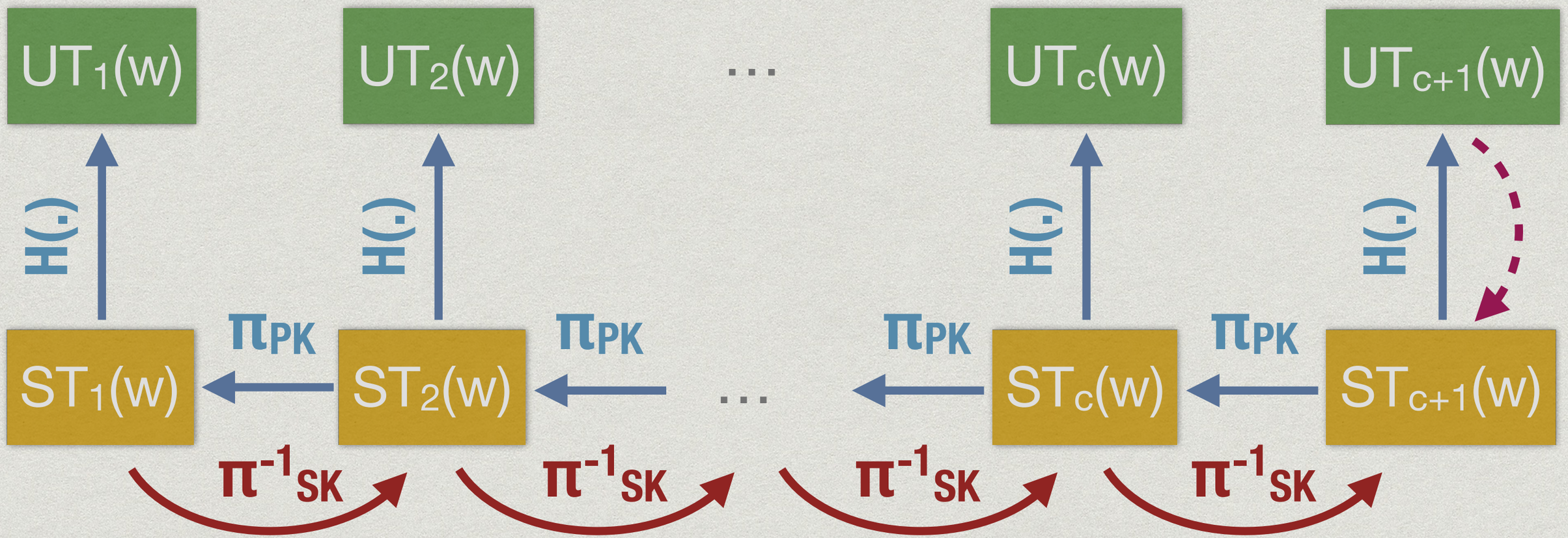

UT1(w)
UT2(w)
...
UTc(w)
UTc+1(w)
H(·)
H(·)
H(·)
H(·)
πPK
πPK
πPK
πPK
ST1(w)
ST2(w)
...
STc(w)
STc+1(w)
π-1SK
π-1SK
π-1SK
π-1SK



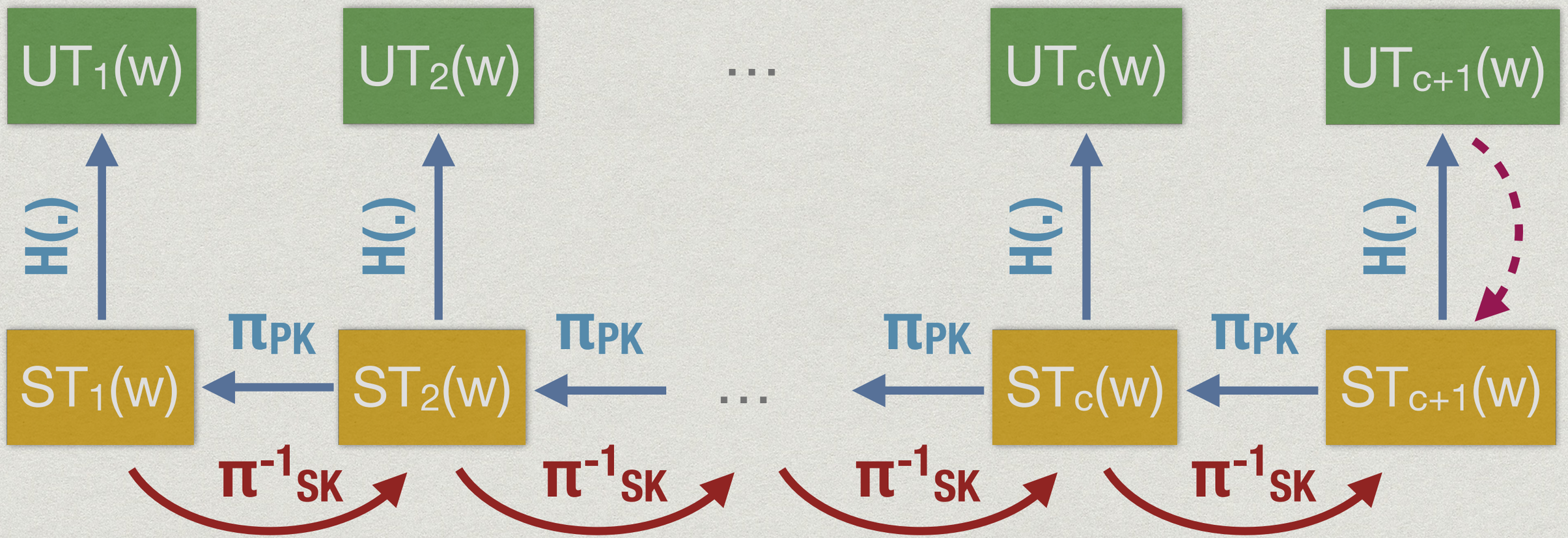
- Client stores W[w] := $ST_c(w)$



- Client stores W[w] := $ST_c(w)$
- Search w: send $ST_c(w)$

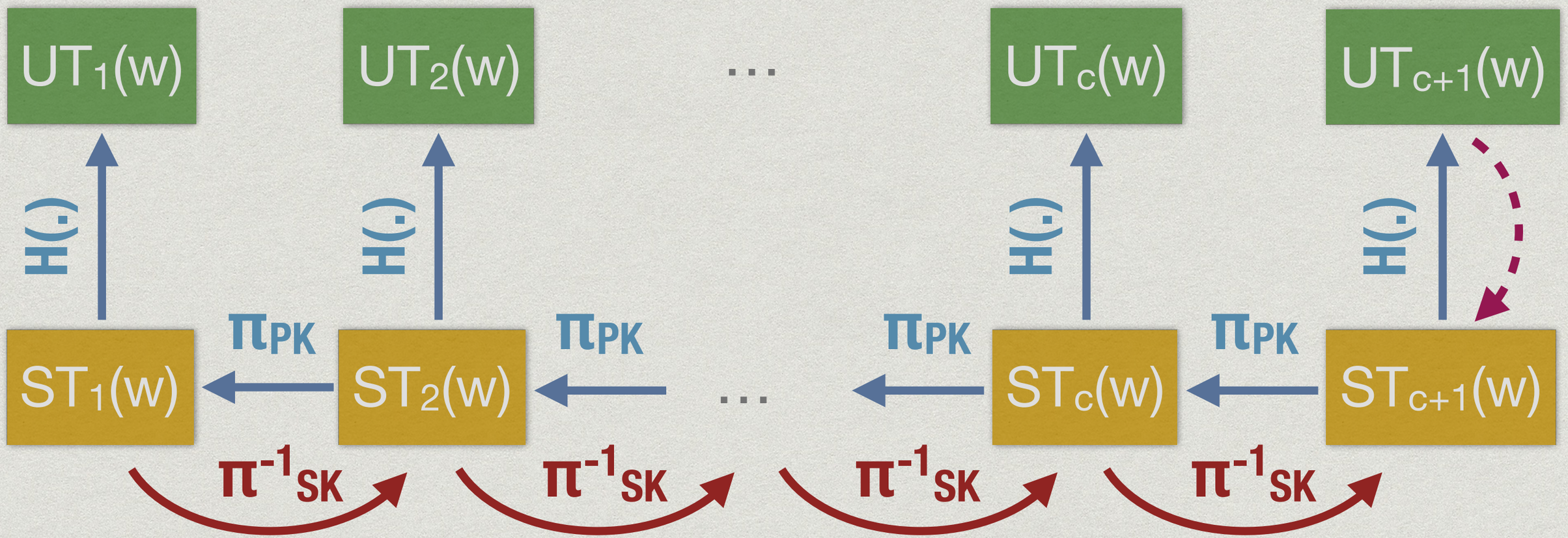


- Client stores $W[w] := ST_c(w)$
- Search w: send $ST_c(w)$
- Update: $W[w] := \pi^{-1}_{SK}(ST_c(w))$



Search:

- Client: constant
- Server: O(|DB(w)|)

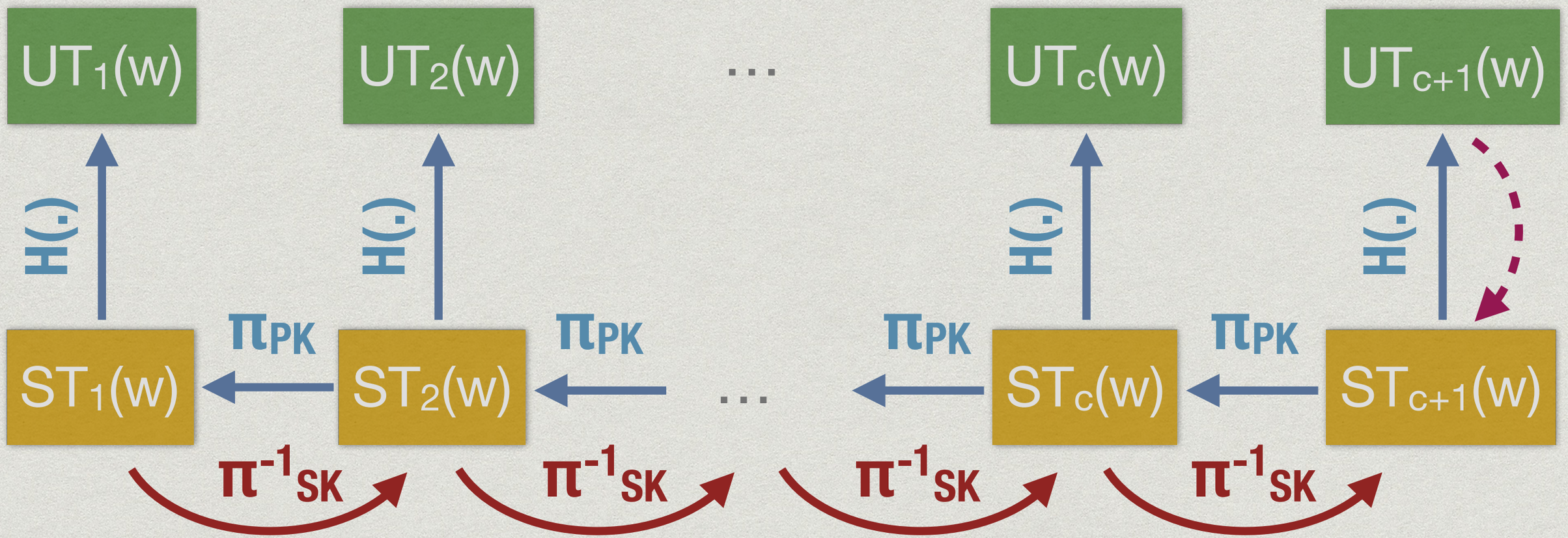


Search:

- Client: constant
- Server: O(|DB(w)|)

Update:

- Client: constant
- Server: constant



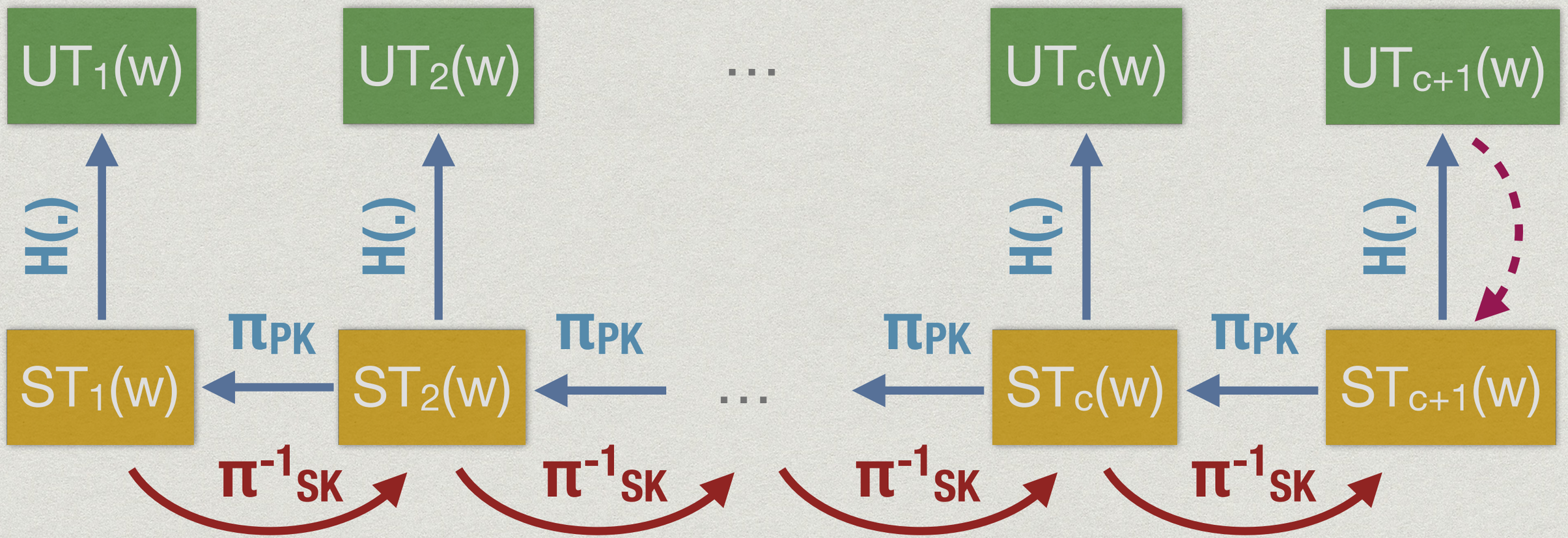
Search:

- Client: constant
- Server: O(|DB(w)|)

Update:

- Client: constant
- Server: constant

Optimal



Storage:

- Client: O(K)
- Server: O(|DB|)

Σοφος

- TDP π? RSA or Rabin

Σοφος

- TDP π? RSA or Rabin
  - ✗ Elements (STs) are large (2048 bits).

Σοφος

- TDP π? RSA or Rabin
  - ✗ Elements (STs) are large (2048 bits).
  - ✗ Huge client storage.

Σοφος

- TDP π? RSA or Rabin
  - ✗ Elements (STs) are large (2048 bits).
  - ✗ Huge client storage.
- Client only stores c, pseudo-randomly generates $ST_1(w)$, computes $ST_c(w)$ on the fly

Σοφος

- TDP π? RSA or Rabin
  - ✗ Elements (STs) are large (2048 bits).
  - ✗ Huge client storage.
- Client only stores c, pseudo-randomly generates $ST_1(w)$, computes $ST_c(w)$ on the fly
  - ✓ Efficient (non-iterative) using RSA
- Search is embarrassingly parallelizable

$$x^{d^{\cdot^{\cdot^{\cdot^{d}}}}} = x^{(d^c \bmod \phi(N))} \bmod N$$

Σοφος - Security

- Update leakage: nothing **Forward private!**
- Search leakage:
  - search pattern
  - ‘history’ of w: the timestamped list of updates of keyword w

Adaptive security (ROM)

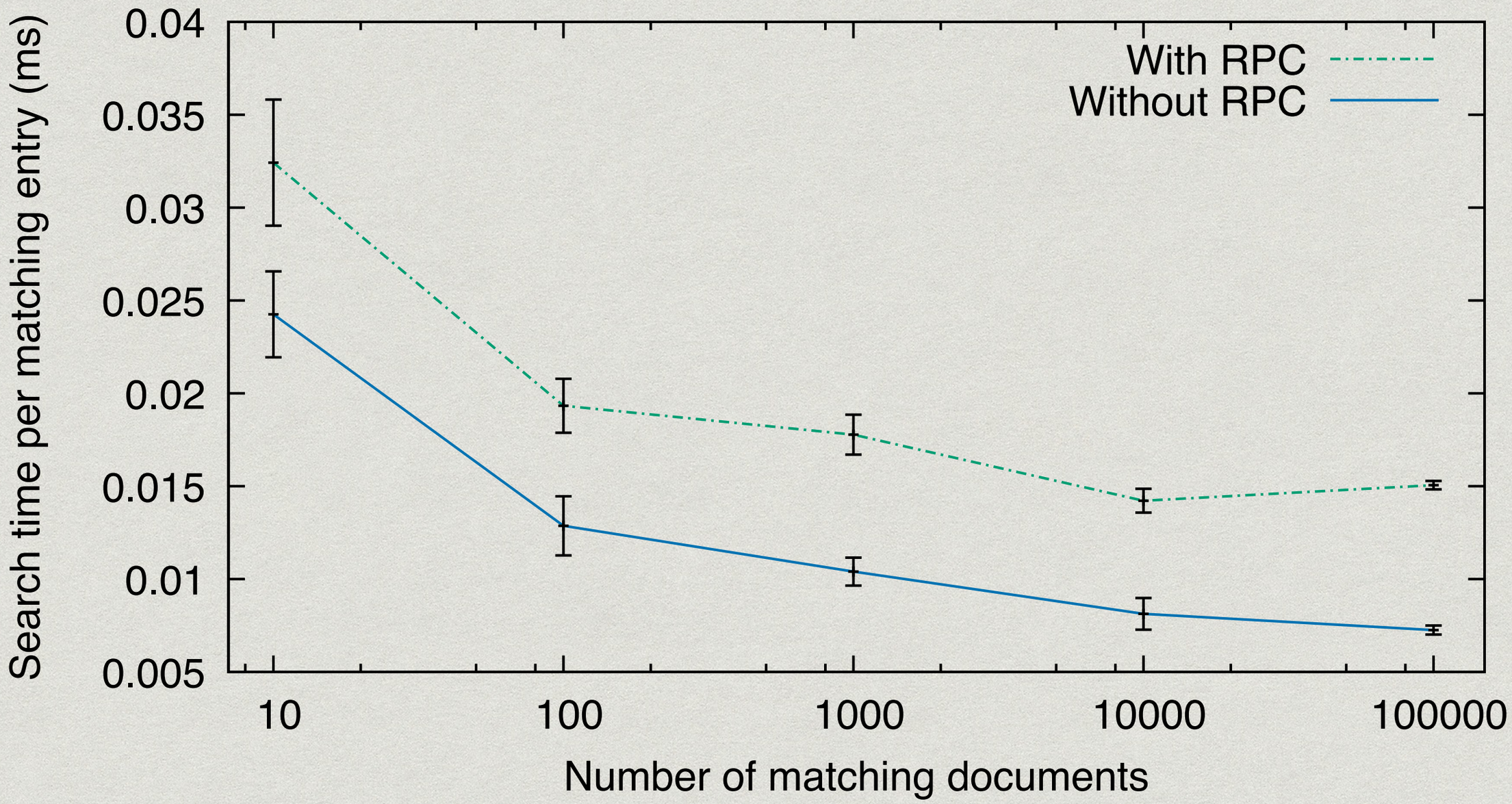
Σοφος - Evaluation

- C/C++ full fledged implementation
- Server KVS: RocksDB
- Evaluated on a desktop computer
  4 GHz Core i7 CPU, 8GB RAM, SSD

https://gitlab.com/sse/sophos

Σοφος - Evaluation

2M keywords, 140M entries
5.25GB server storage, 64.2 MB Client storage



Σοφος

- Provable forward privacy

Σοφος

- Provable forward privacy
- Very simple

Σοφος

- Provable forward privacy
- Very simple
- Efficient search (IO bounded)

Σοφος

- Provable forward privacy
- Very simple
- Efficient search (IO bounded)
- Asymptotically efficient update (optimal)

Σοφος

- Provable forward privacy
- Very simple
- Efficient search (IO bounded)
- Asymptotically efficient update (optimal)
- In practice, very low update throughput (4300 p/s - 20x slower than other work)

THANKS!