Order-Preserving Key Compression for In-Memory Search Trees

Huanchen Zhang

(CMU ⟶ Snowflake ⟶ Tsinghua)

Lily Liu (CMU), David G. Andersen (CMU), Michael Kaminsky (BrdgAI), Kimberly Keeton (Hewlett-Packard Labs), Andrew Pavlo (CMU)

Source code: https://github.com/efficient/HOPE

Search tree compression matters

⇨ Tree indexes consume a lot of memory

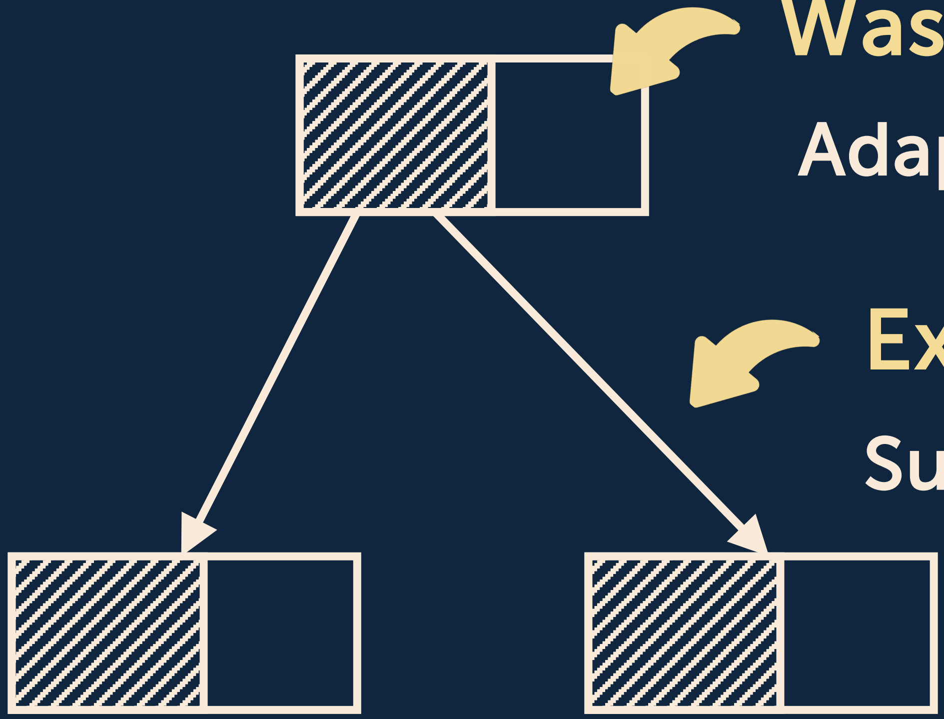
Benchmark	Tree Index Memory
TPC-C	58%

[SIGMOD'16]

⇨ Databases face tight memory budgets

vCPU	Mem(GB)		SSD(GB)
4	30.5	1 : 30	950

Prior work focuses on structural compression



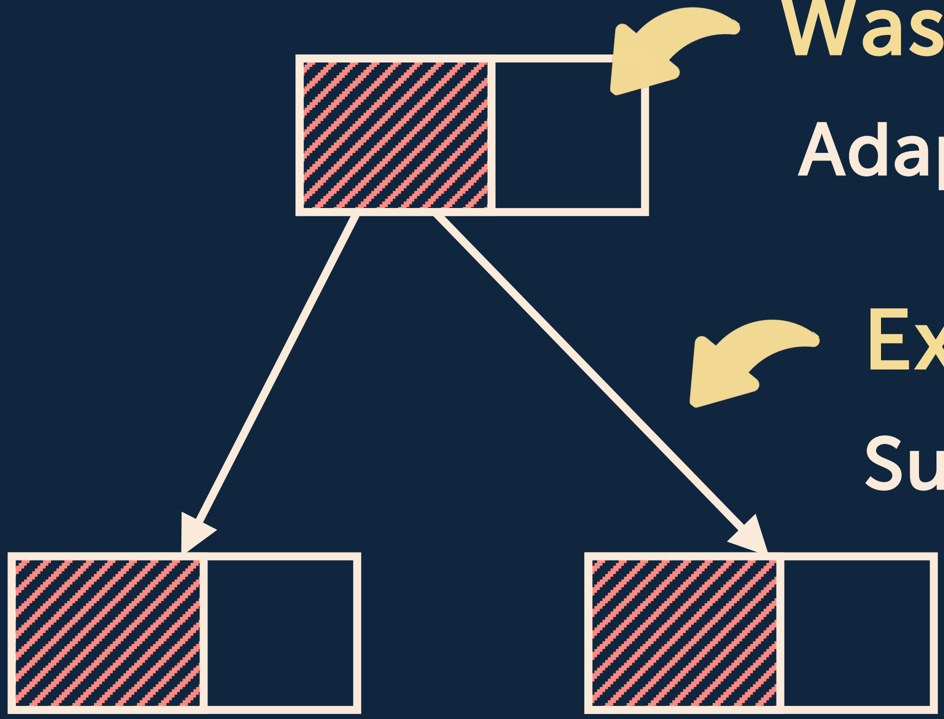
Wasted Empty Space

Adaptive node sizes (e.g., ART)

Excessive pointers

Succinct data structures (e.g., SuRF)

Prior work focuses on structural compression



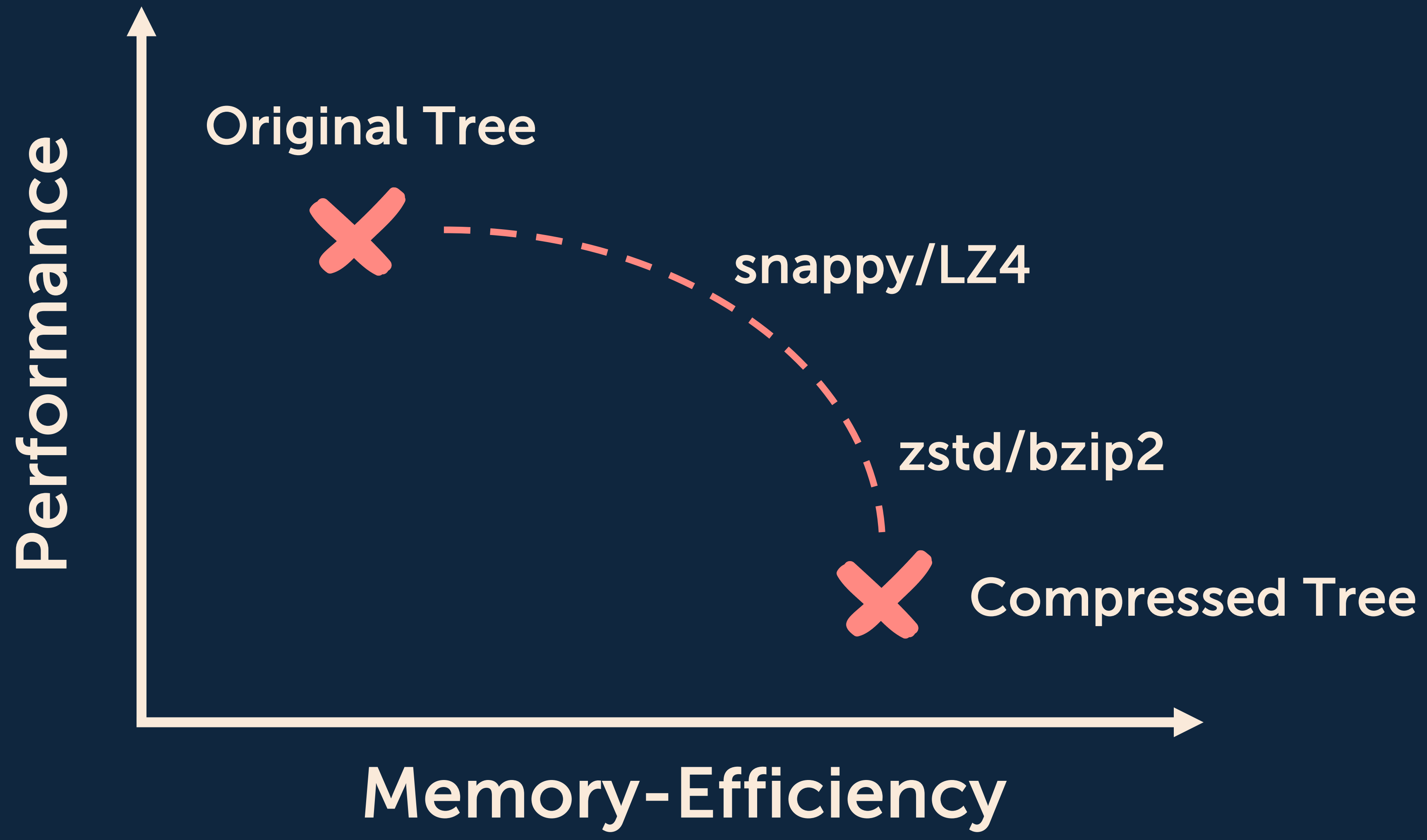
Wasted Empty Space

Adaptive node sizes (e.g., ART)

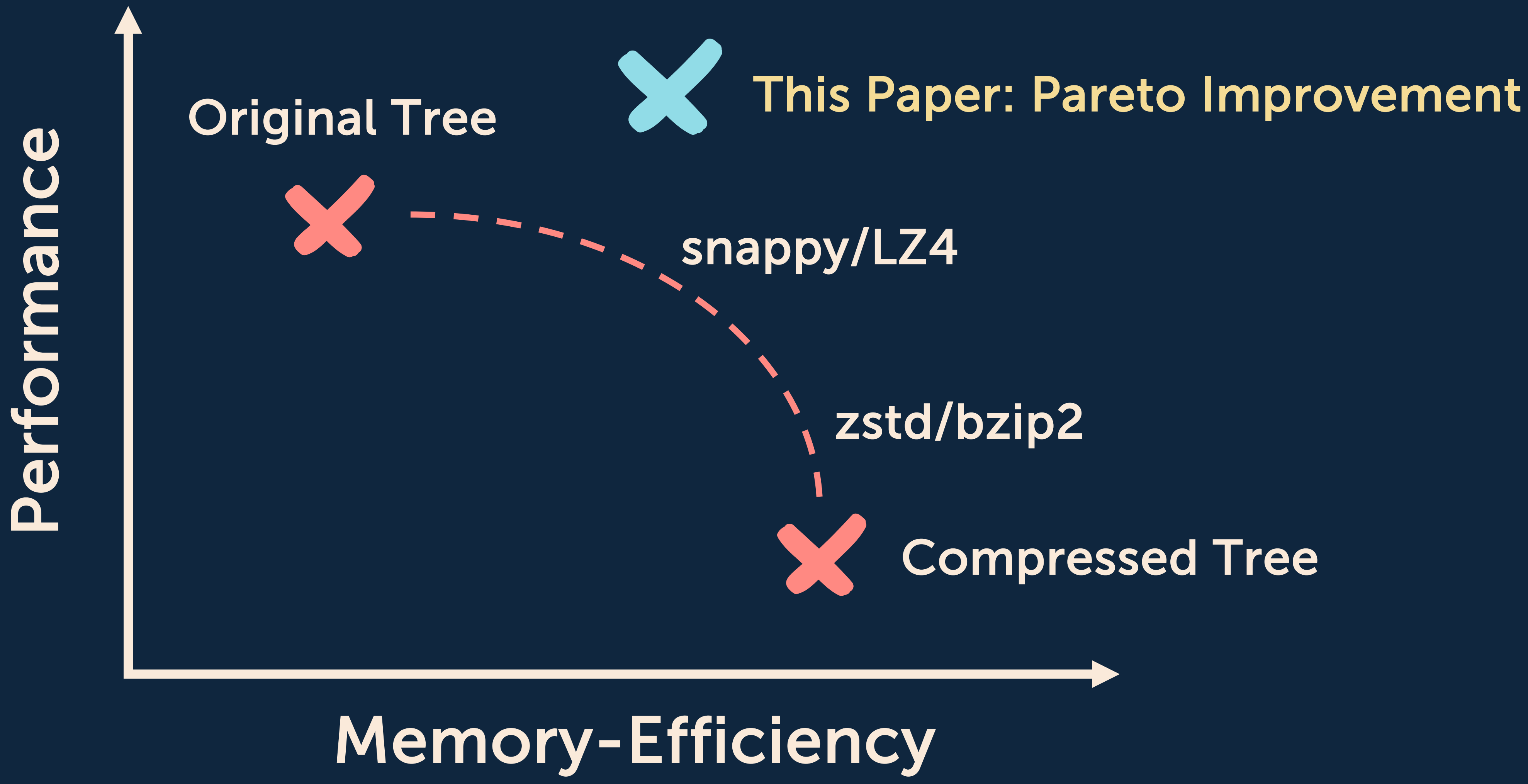
Excessive pointers

Succinct data structures (e.g., SuRF)

Block compression is slow



Block compression is slow



High-speed Order-Preserving Encoder

⇨ Stand-alone C++ library that can compress arbitrary keys while preserving order

CPR: 1.5 – 2.7

Mem: < 512KB

Speed: 4ns/char, 250MB/s

30% smaller

+

40% faster

High-speed Order-Preserving Encoder

- Stand-alone C++ library that can
  compress arbitrary keys while preserving order

CPR: 1.5 – 2.7

Mem: < 512KB

Speed: 4ns/char, 250MB/s

30% smaller

+

40% faster

Why whole-key dictionary compression fails

Table Column

Chopin	2
Mozart	3
Bach	1
Bach	1
Chopin	2
Bach	1

Dictionary

Bach	1
Chopin	2
Mozart	3

Why whole-key dictionary compression fails

Table Column

Chopin
Mozart
Bach
Bach
Chopin
Bach

2
3
1
1
2
1

Dictionary

Bach	1
Chopin	2
Mozart	3

Why whole-key dictionary compression fails

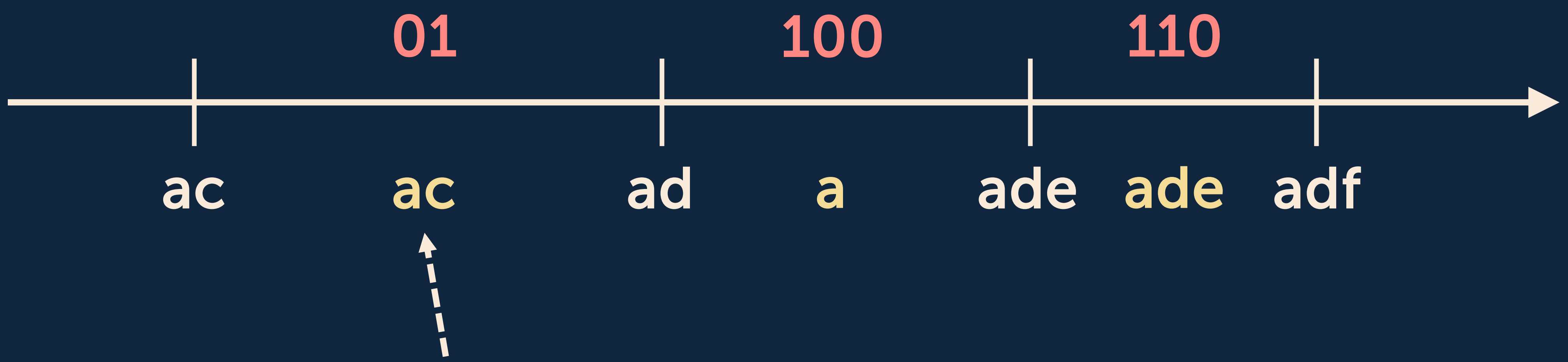
Tree Index

Dictionary

Bach	1
Chopin	2
Mozart	3

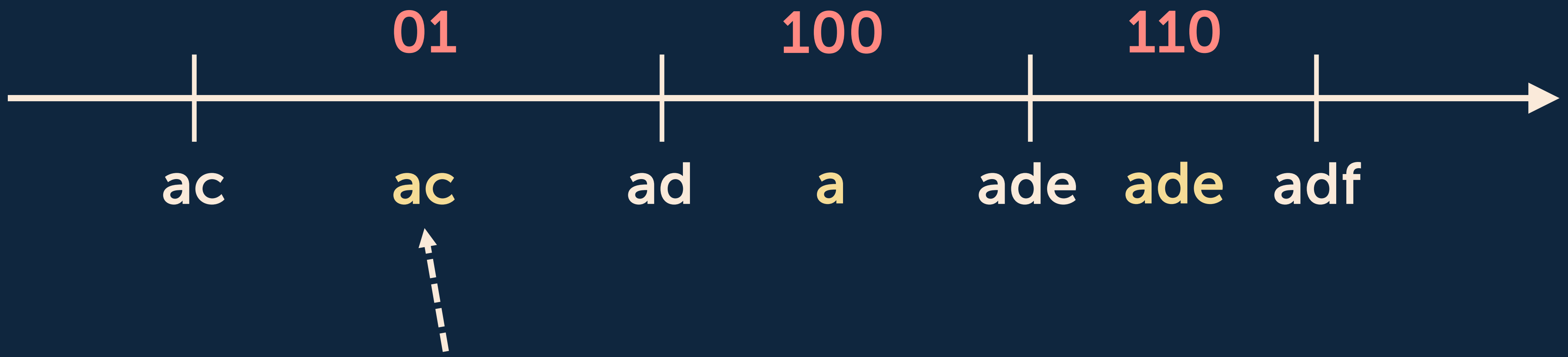
- ✗ No compression for unique keys
- ✗ Inefficiency in handling arbitrary input keys
- ✗ Overhead in maintaining dictionary order

String Axis Model



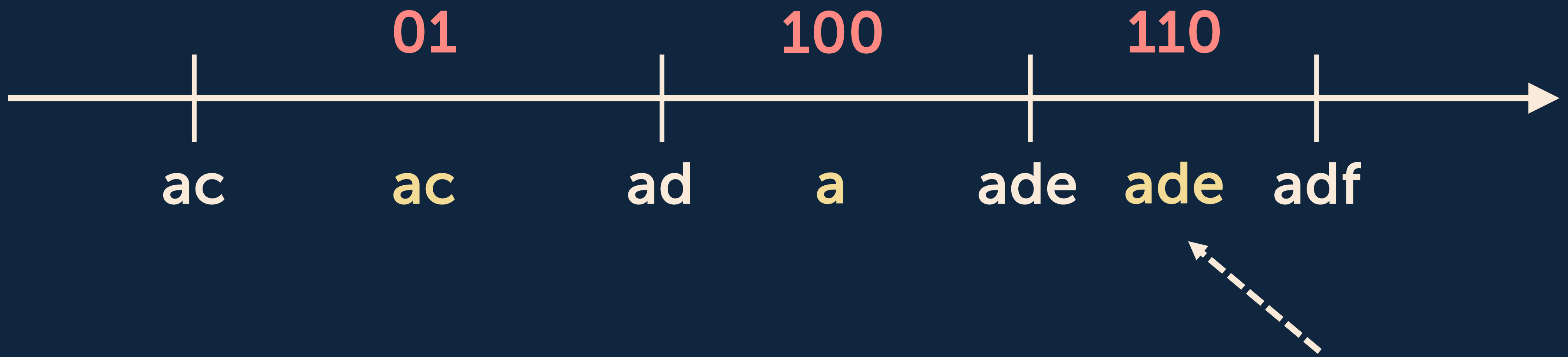
Example: academic

String Axis Model



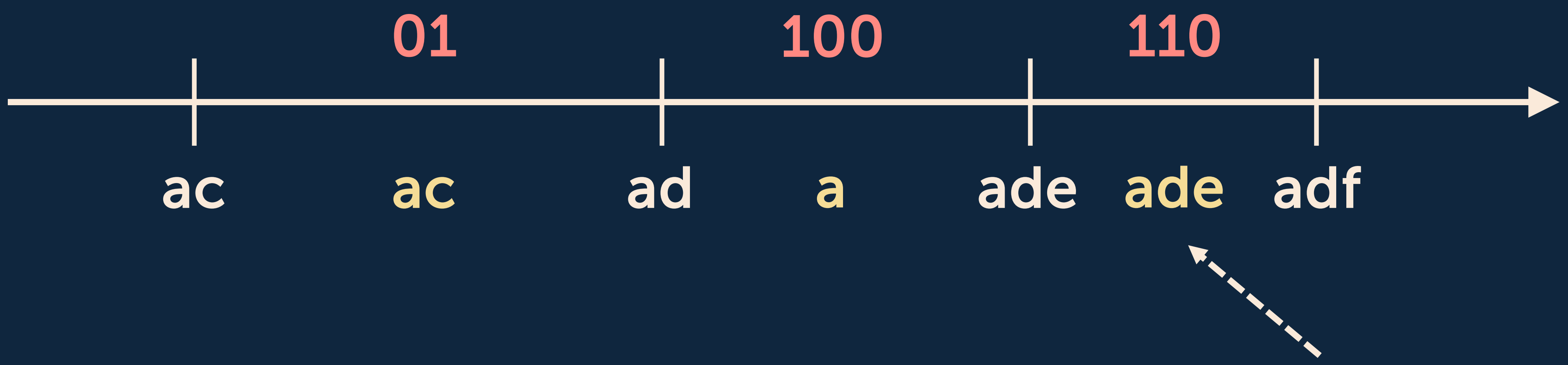
Example: academic ⟶ academic ⟶ 01ademic

String Axis Model



Example: academic ⟶ academic ⟶ 01ademic

String Axis Model



Example: academic → academic → 01ademic

→ 01ademic → 01110mic

String Axis Model

Example: academic ⟶ academic ⟶ 01ademic

⟶ 01ademic ⟶ 01110mic

• • •

String Axis Model

→ Dictionary Completeness

String Axis Model

- Dictionary Completeness
- Order-Preserving

String Axis Model

- Dictionary Completeness
- Order-Preserving

s1

∧

s2

String Axis Model

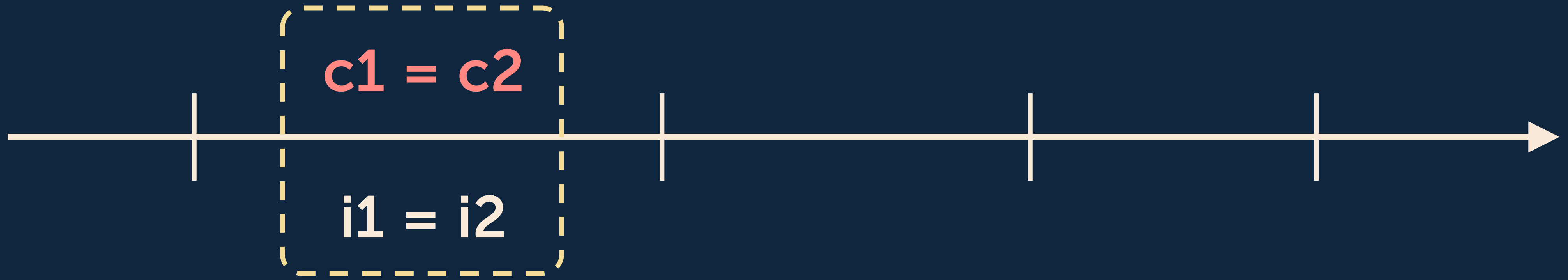
- Dictionary Completeness
- Order-Preserving

$$s1 \longrightarrow c1 \bullet s1_{suffix}$$

$$\wedge$$

$$s2 \longrightarrow c2 \bullet s2_{suffix}$$

String Axis Model



- Dictionary Completeness
- Order-Preserving

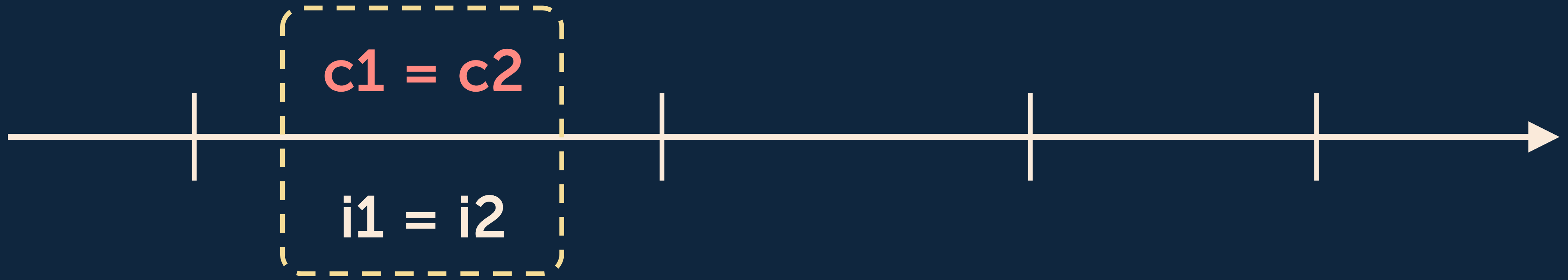
$$\begin{array}{lcl} s1 & \longrightarrow & c1 \bullet s1_{suffix} \\ \wedge & & \\ s2 & \longrightarrow & c2 \bullet s2_{suffix} \end{array}$$

String Axis Model

- Dictionary Completeness
- Order-Preserving

$$\begin{array}{ccc} s1 & \longrightarrow & c1 \bullet s1_{suffix} \\ \wedge & & \parallel \\ s2 & \longrightarrow & c2 \bullet s2_{suffix} \end{array}$$

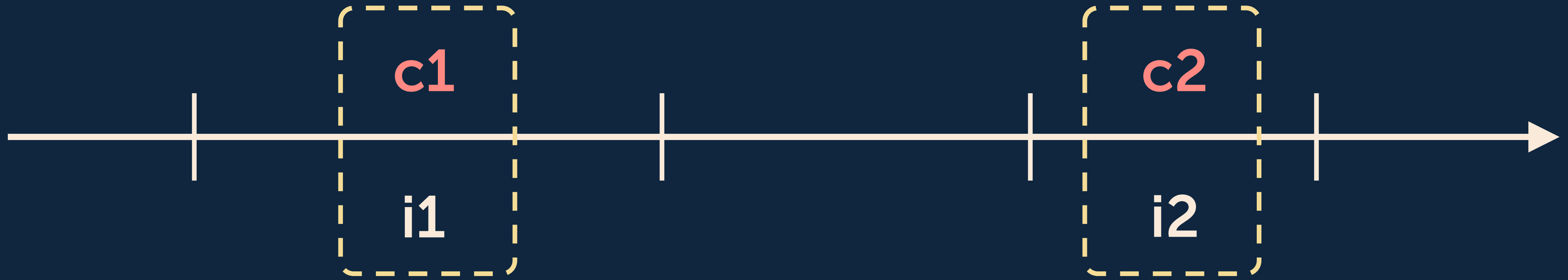
String Axis Model



- Dictionary Completeness
- Order-Preserving

$$\begin{array}{ccccc} s1 & \longrightarrow & c1 & \bullet & s1_{suffix} \\ \wedge & & \parallel & & \wedge \\ s2 & \longrightarrow & c2 & \bullet & s2_{suffix} \end{array}$$

String Axis Model



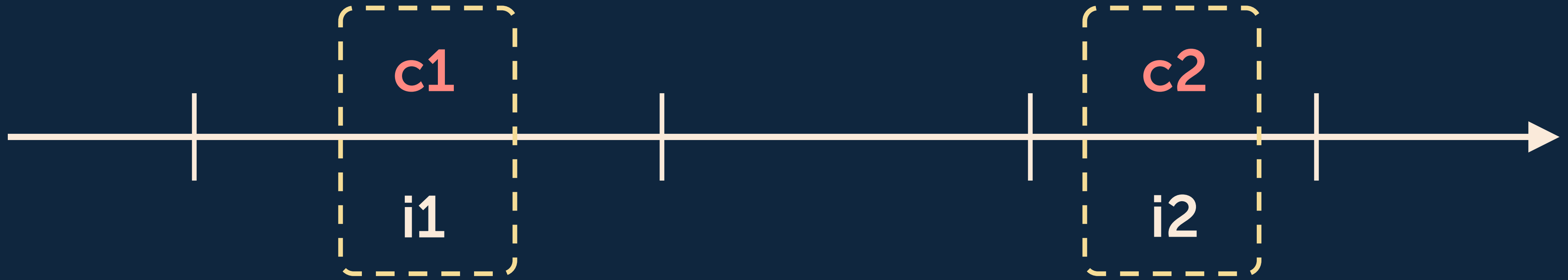
- Dictionary Completeness
- Order-Preserving

$$s1 \longrightarrow c1 \cdot s1_{suffix}$$

$$\wedge$$

$$s2 \longrightarrow c2 \cdot s2_{suffix}$$

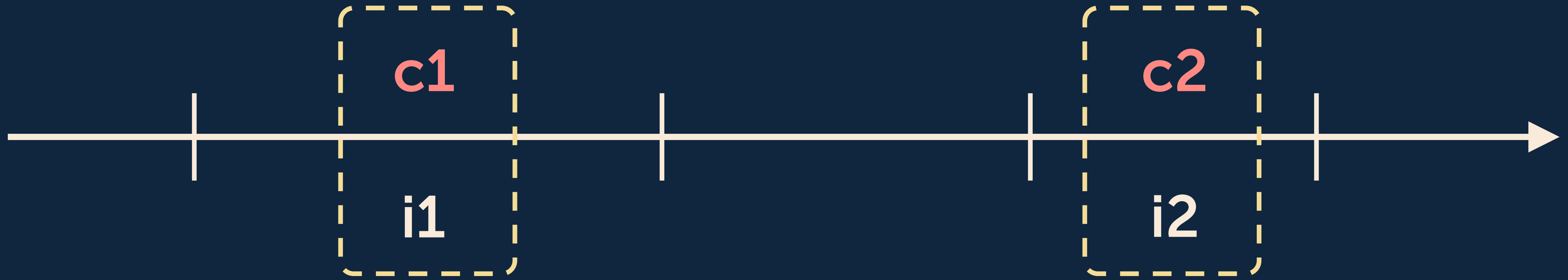
String Axis Model



- Dictionary Completeness
- Order-Preserving

$$\begin{array}{ccc} s1 & \longrightarrow & c1 \bullet s1_{suffix} \\ \wedge & & \wedge \\ s2 & \longrightarrow & c2 \bullet s2_{suffix} \end{array}$$

String Axis Model



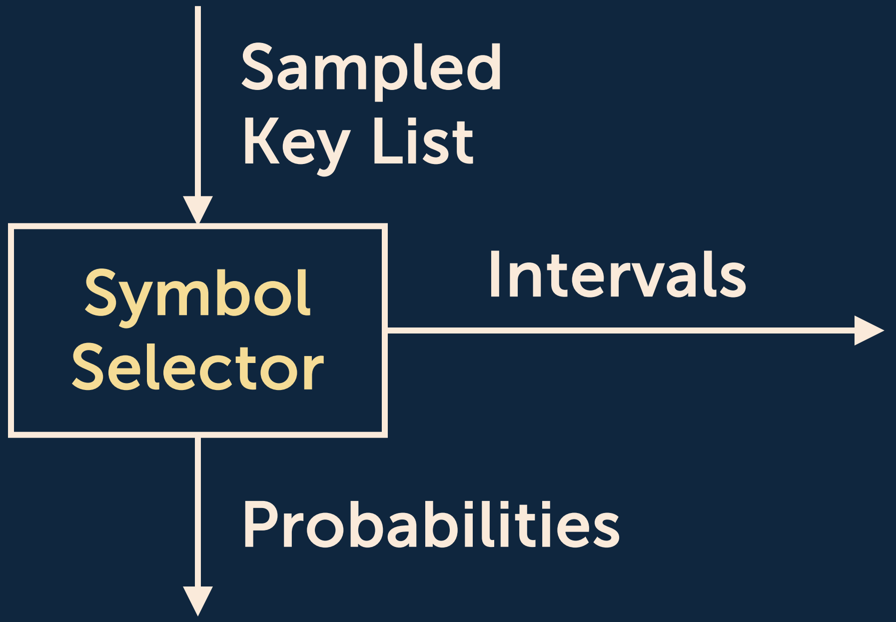
- Dictionary Completeness
- Order-Preserving

$$
\begin{array}{ccccc}
s1 & \longrightarrow & c1 \cdot s1_{suffix} & \dashrightarrow & Enc(s1) \\
\wedge & & \wedge & & \wedge \\
s2 & \longrightarrow & c2 \cdot s2_{suffix} & \dashrightarrow & Enc(s2)
\end{array}
$$

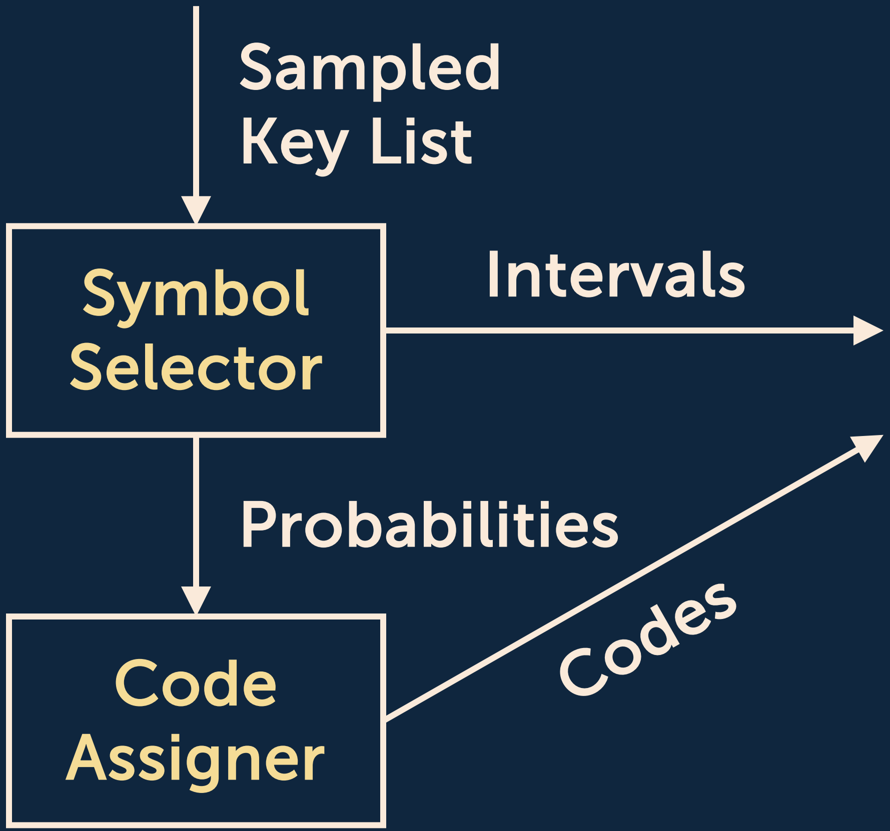
String Axis Model

- Dictionary Completeness
- Order-Preserving
- Small Dictionary

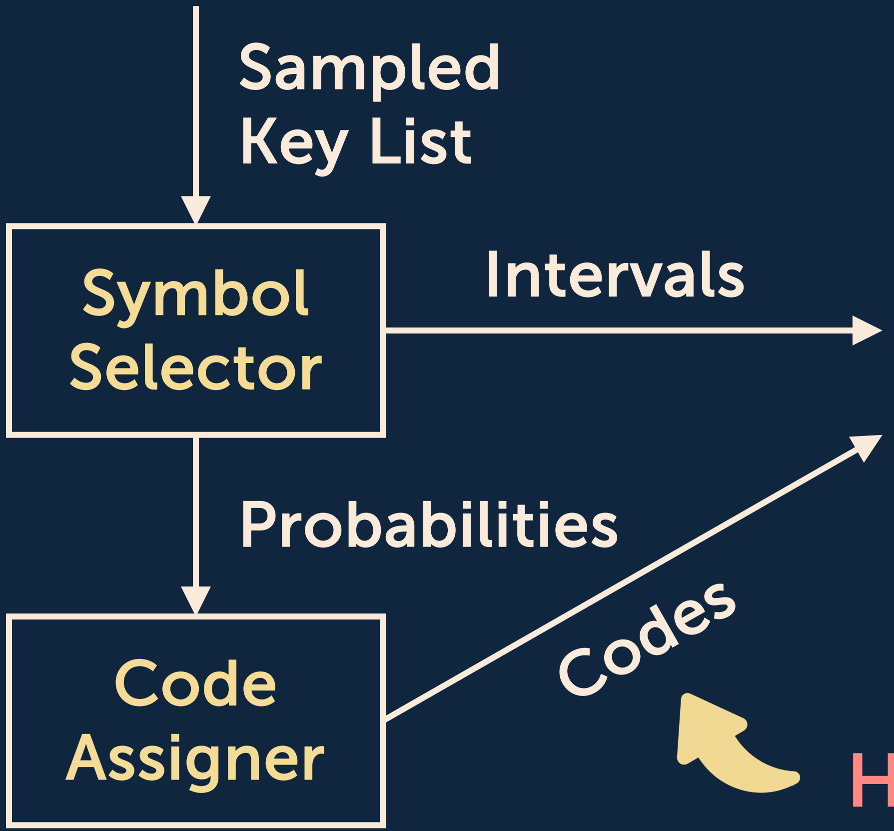
HOPE is fast and extensible



HOPE is fast and extensible

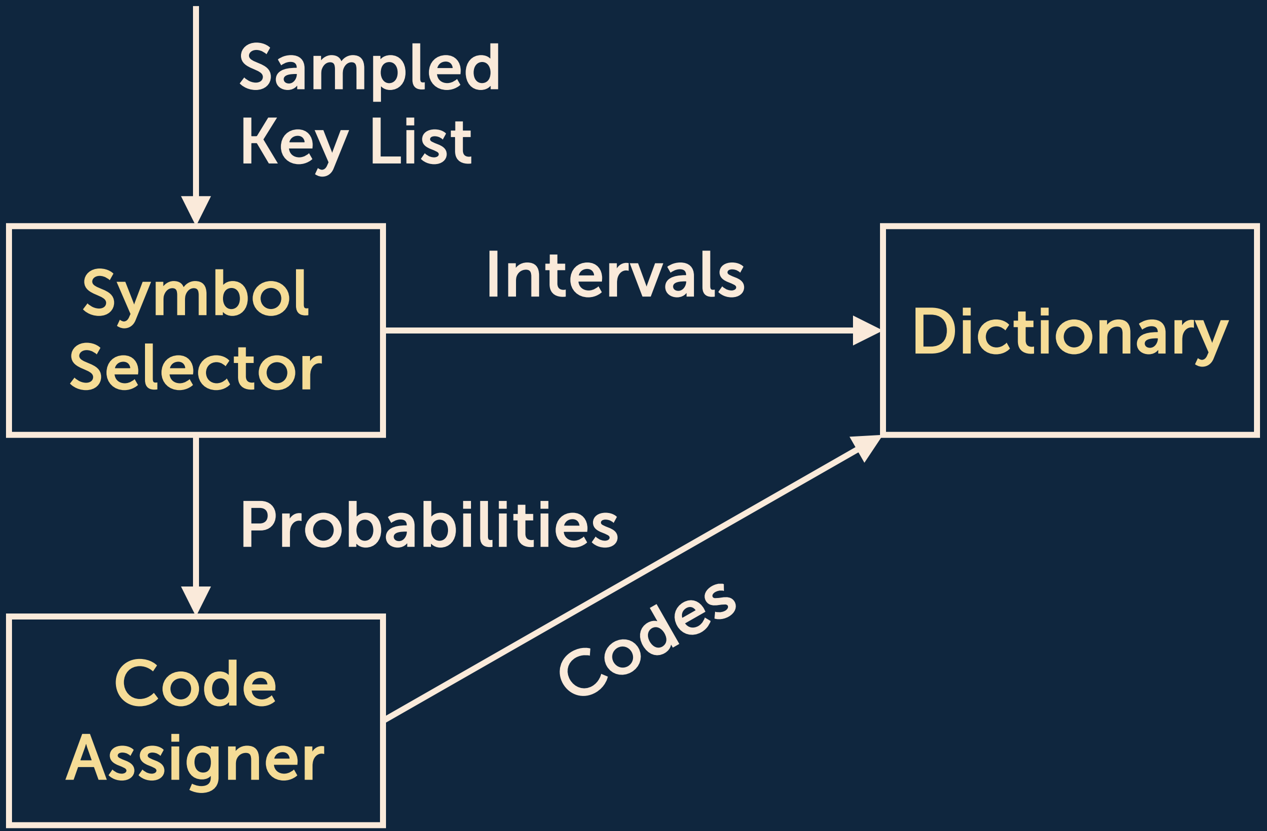


HOPE is fast and extensible

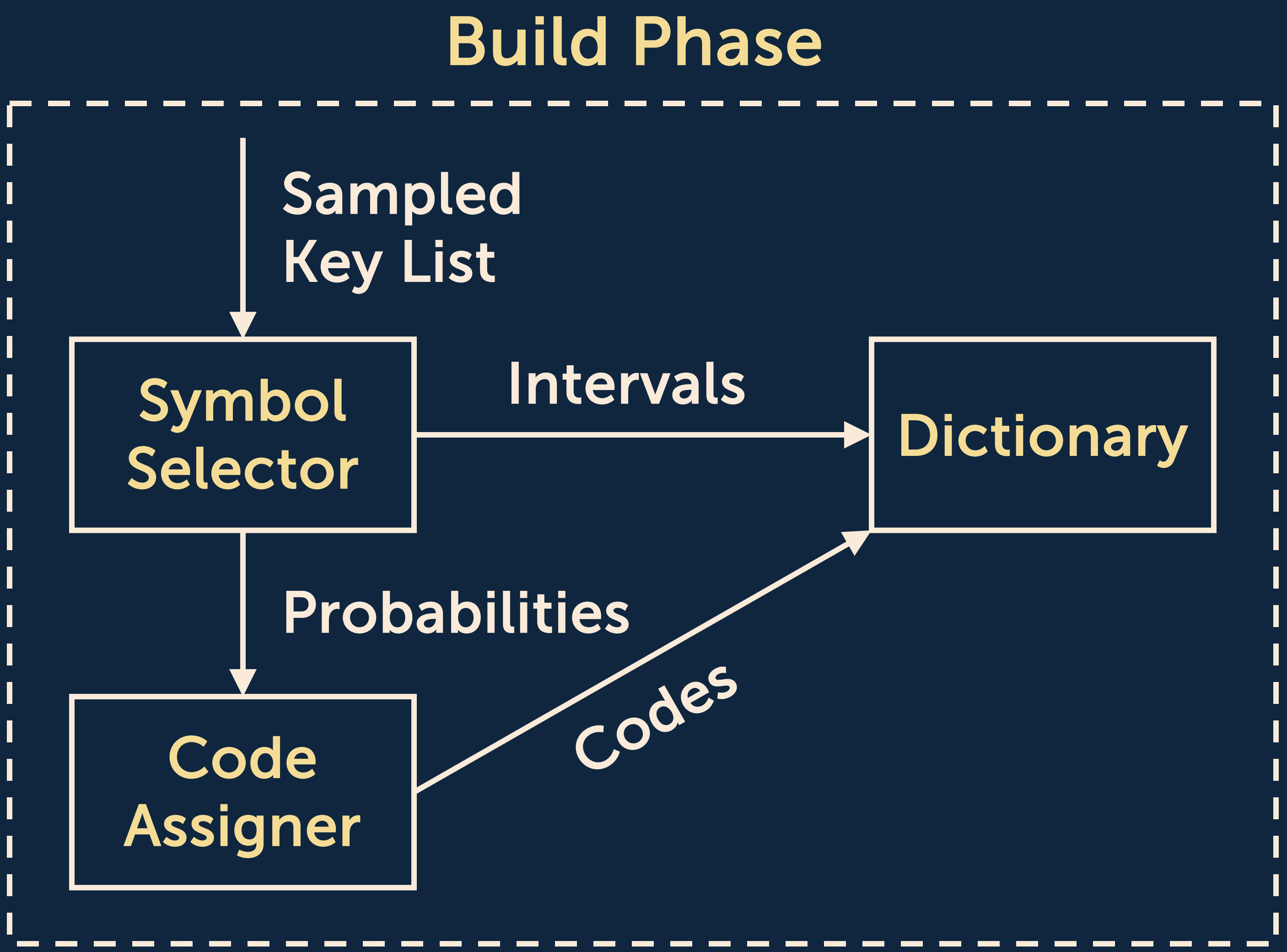


Hu-Tucker Codes

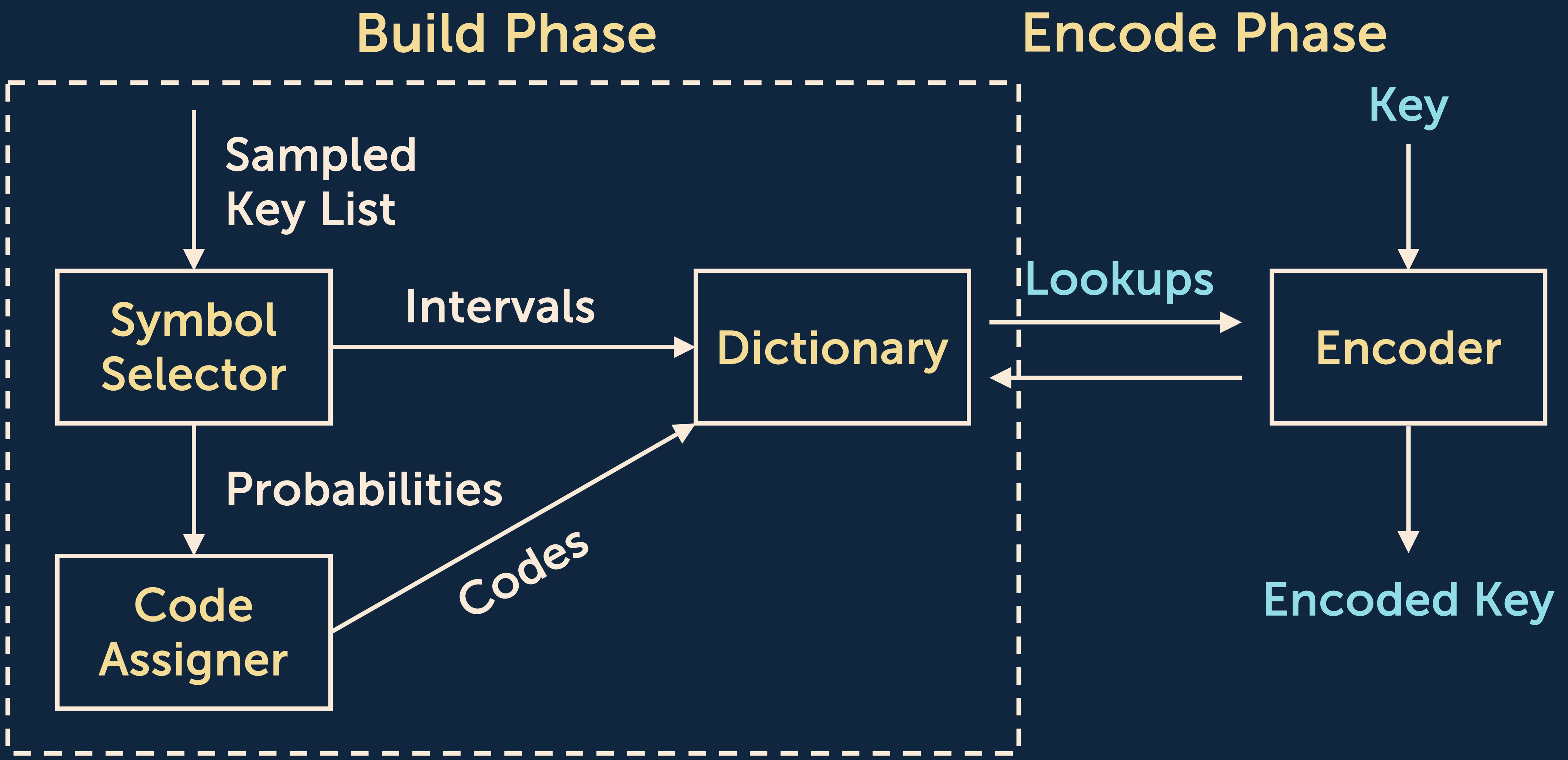
HOPE is fast and extensible



HOPE is fast and extensible



HOPE is fast and extensible



Applying HOPE to in-memory search trees

Structures: B+tree, Prefix B+tree, ART, HOT, SuRF

Keys: Emails, Wikipedia Titles, URLs

Operations: Lookup, Scan, Insert, Update

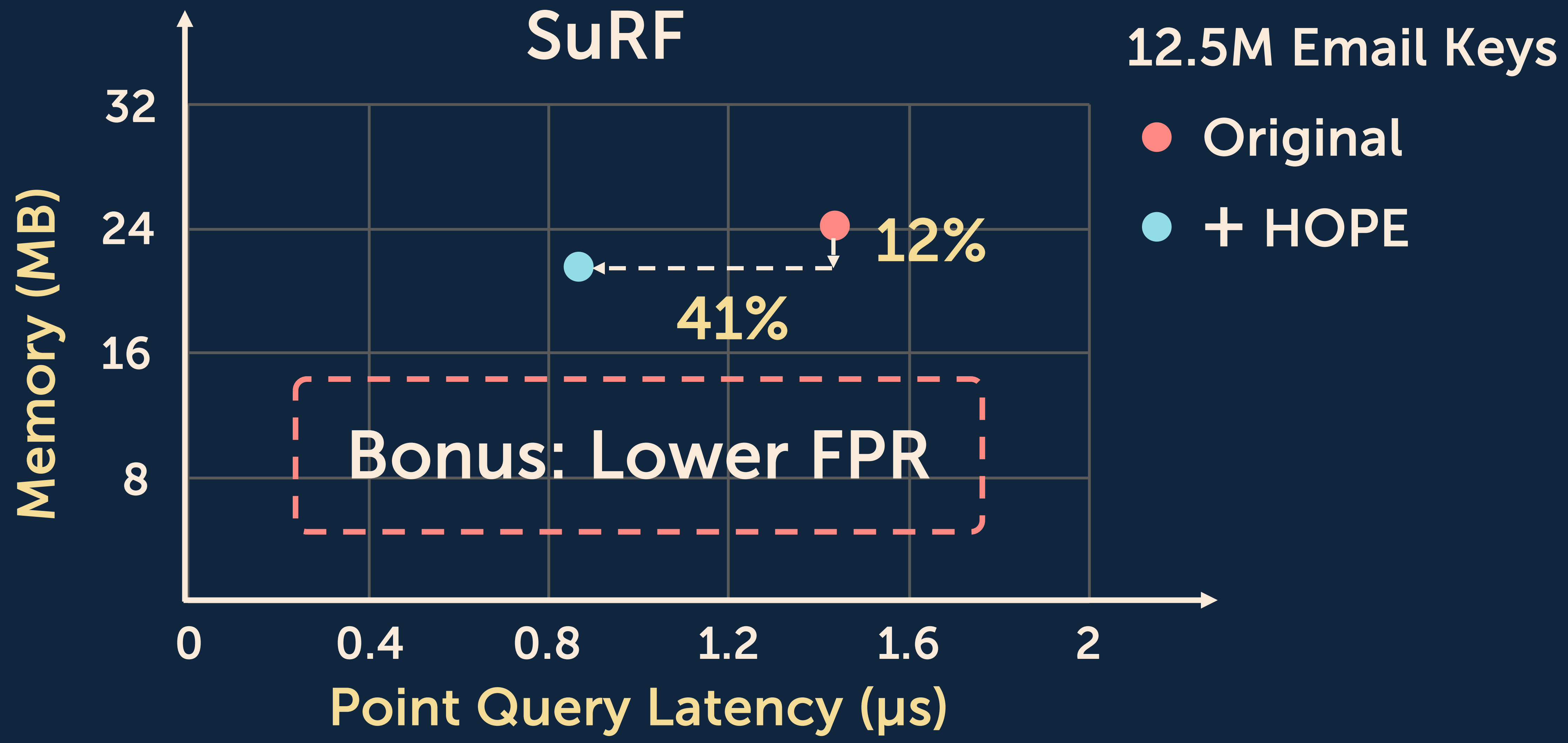
30% Smaller + **40%** Faster

? ?

CPR: 1.5 – 2.7

Overhead: 4ns/char

HOPE is orthogonal to structural compression



Applying HOPE to string columns?

Pros

+ Disk/Memory space savings
+ Speedup queries by processing less data

Cons

- Variable-length codes

HOPE Takeaways

- Improves Space AND Performance.
- Benefit beyond search trees?
- Source code: https://github.com/efficient/HOPE