Efficient Online Timed Pattern Matching by Automata-Based Skipping

Masaki Waga[1], Ichiro Hasuo[2], and Kohei Suenaga[3]

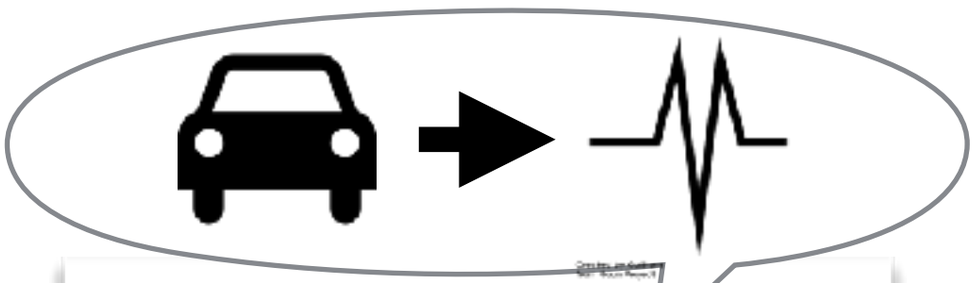
University of Tokyo[1], National Institute of Informatics[2], and Kyoto University[3]

6 Sep. 2017, FORMATS 2017

The authors are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST, Grants-in-Aid No. 15KT0012, JSPS, and JST PRESTO Grant Number JPMJPR15E5, Japan.

Many slides contributed by Masaki Waga

Monitoring



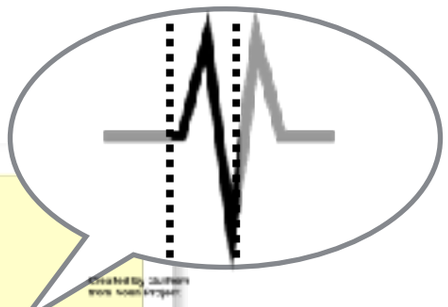
Log w

,

Spec. of bad behaviors $\mathcal{A}$

e.g. After a gear changed from *high* to *low*, it goes back to *high* again in 2 sec.

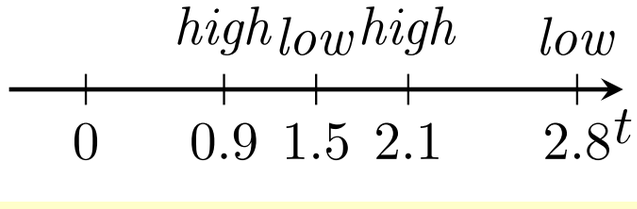
The set of intervals (t, t') such that $w|_{(t,t')} \in L(\mathcal{A})$



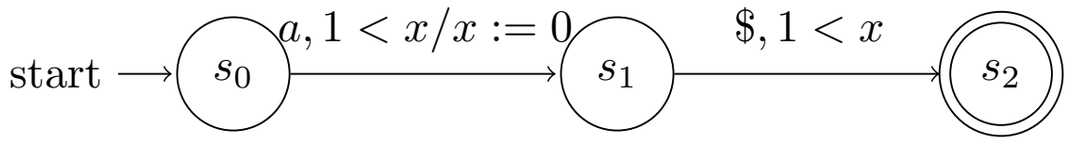
Timed Pattern Matching

Input

timed word w (target, log)



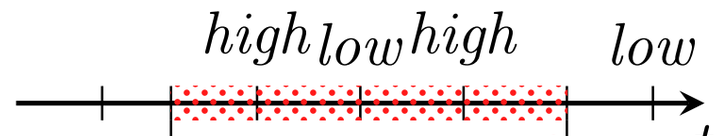
timed automaton $\mathcal{A}$ (pattern, spec)



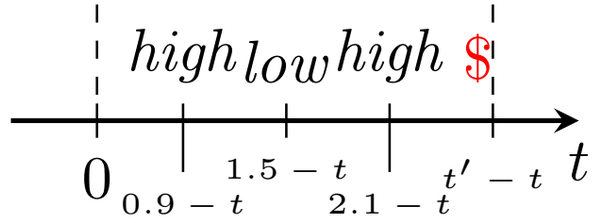
Output

$$\{(t, t') \mid w|_{(t,t')} \in L(\mathcal{A})\}$$

$w =$



$w|_{(t,t')} =$



Online Monitoring

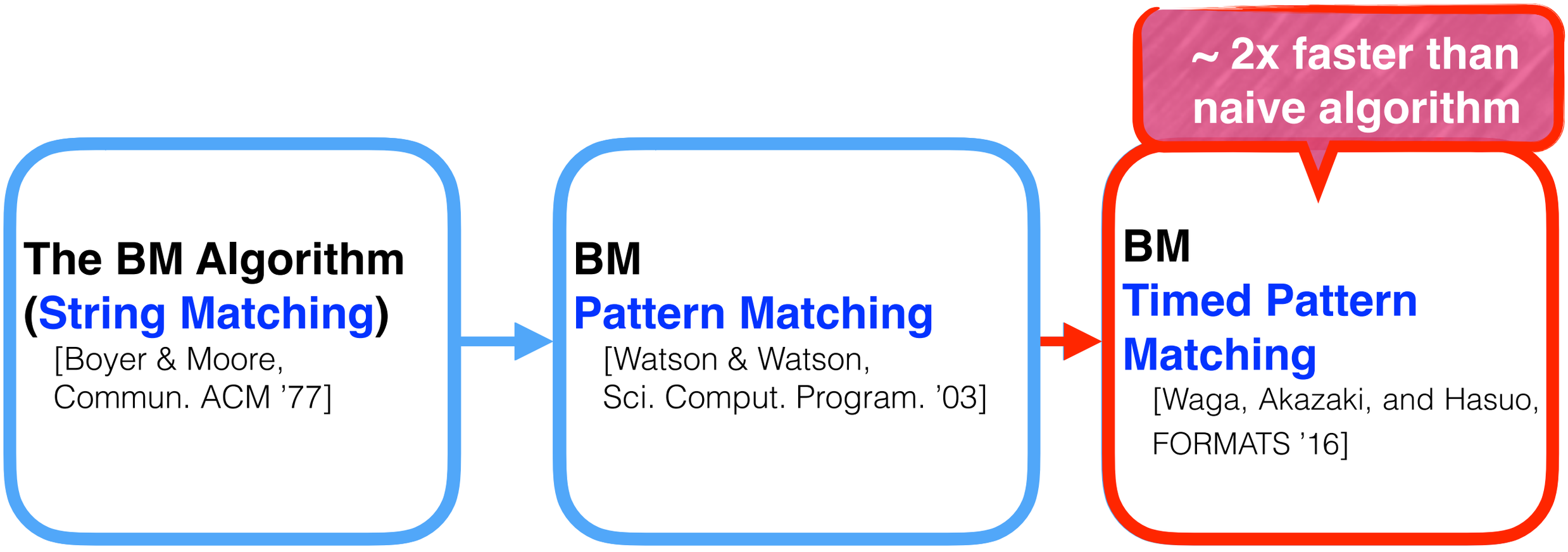
- Monitoring before the entire log is given
- Applications in **embedded systems**
  - System monitoring **during operation**
    - Log prefixes can be **thrown away**
- Faster monitoring
  ➜ denser logs, cheaper hardware
- Serious real-world problem
  - GBs of log per second
  - Cost sensitive (processors & memory)

Hasuo (NII, JP)

Previous Work: an **Offline** Algorithm for Timed Patten Matching

[Waga, Akazaki, and Hasuo, FORMATS ’16]

- Enhanced by **skipping** from string matching



Previous Work: an **Offline** Algorithm for Timed Patten Matching

```
 THIS_IS_A_STRONG_STRING
═══════════════════════
✗STRING
✗ STRING
✗  STRING
✗   STRING
        ⋮
```

FORMATS '16]

tring matching

~ 2x faster than naive algorithm

The BM Algorithm (String Matching)
[Boyer & Moore, Commun. ACM '77]

→

BM Pattern Matching
[Watson & Watson, Sci. Comput. Program. '03]

→

BM Timed Pattern Matching
[Waga, Akazaki, and Hasuo, FORMATS '16]

Previous Work: an **Offline** Algorithm for Timed Patten Matching

- **KMP Algorithm**
  [Knuth, Morris, and Pratt, SIAM J. Comput. ’77]
- **Quick Search** [Sunday, Commun. ACM ’90]
- **FJS Algorithm** (~ Quick Search + KMP) :
  [Franek, Jennings, and Smyth, J. Discrete Algorithms ’07]
  ⋮

The BM Algorithm (String Matching)
[Boyer & Moore, Commun. ACM ’77]

→

BM Pattern Matching
[Watson & Watson, Sci. Comput. Program. ’03]

→

BM Timed Pattern Matching
[Waga, Akazaki, and Hasuo, FORMATS ’16]

~ 2x faster than naive algorithm

Contribution: An Efficient Algorithm for **Online** Timed Pattern Matching

The BM Algorithm (String Matching)
[Boyer & Moore, Commun. ACM '77]

BM Pattern Matching
[Watson & Watson, Sci. Comput. Program. '03]

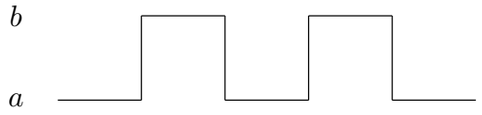
BM Timed Pattern Matching
[Waga, Akazaki, and Hasuo, FORMATS '16]

The FJS Algorithm (String Matching)
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

FJS Pattern Matching
[Current work]

FJS Timed Pattern Matching
[Current work]

target: **signals**
(“state-based”)

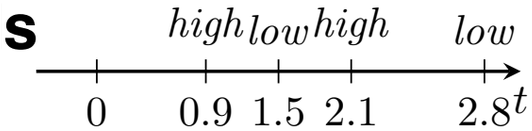


Related Work: Timed Pattern Matching

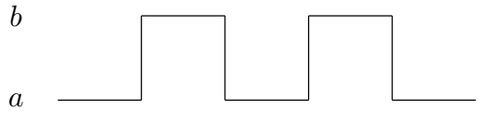
- Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler, “Timed Pattern Matching”. FORMATS ’14
- Thomas Ferrère, Oded Maler, Dejan Nickovic, and Dogan Ulus, “Measuring with Timed Patterns”. CAV ’15
- Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler, “Online Timed Pattern Matching Using Derivatives”. TACAS ’16
- Dogan Ulus, “Montre: A Tool for Monitoring Timed Regular Expressions”. CAV ’17
- Eugene Asarin, Oded Maler, Dejan Nickovic and Dogan Ulus, “Combining the Temporal and Epistemic Dimensions for MTL Monitoring”. FORMATS ’17

- M. Waga, Takumi Akazaki, and Ichiro Hasuo, “A Boyer-Moore Type Algorithm for Timed Pattern Matching”. FORMATS ‘16

target: **timed words**
(“event-based”)



target: **signals**
(“state-based”)



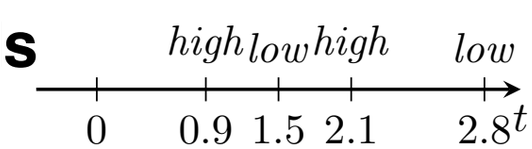
Related Work: Timed Pattern Matching

- Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler, “Timed Pattern Matching”. FORMATS ’14
- Thomas Ferrère, Oded Maler, Dejan Nickovic, and Dogan Ulus, “Measuring with Timed Patterns”. CAV ’15
- Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler, “Online Timed Pattern Matching Using Derivatives”. TACAS ’16
- Dogan Ulus, “Montre: A Tool for Monitoring Timed Regular Expressions”. CAV ’17
- Eugene Asarin, Oded Maler, Dejan Nickovic and Dogan Ulus, “Combining the Temporal and Epistemic Dimensions for MTL Monitoring”. FORMATS ’17

- M. Waga, Takumi Akazaki, and Ichiro Hasuo, “A Boyer-Moore Type Algorithm for Timed Pattern Matching”. FORMATS ‘16

automata-based

target: **timed words**
(“event-based”)



Outline

1. Original FJS Algorithm (for String Matching)
2. Our FJS Algorithm for Timed Pattern Matching
3. Experiments

String Matching

Input

target string w : `THIS_IS_A_STRONG_STRING`
pattern string pat : `STRING`

Output

$$\{(i,j) \mid w(i,j) = pat\} = \{(18,23)\}$$

```
          1234567890123456789012 3
THIS_IS_A_STRONG_STRING
                       STRING
```

String Matching

Input

target string w : THI

pattern string pat : STR

nm matching trials, naively

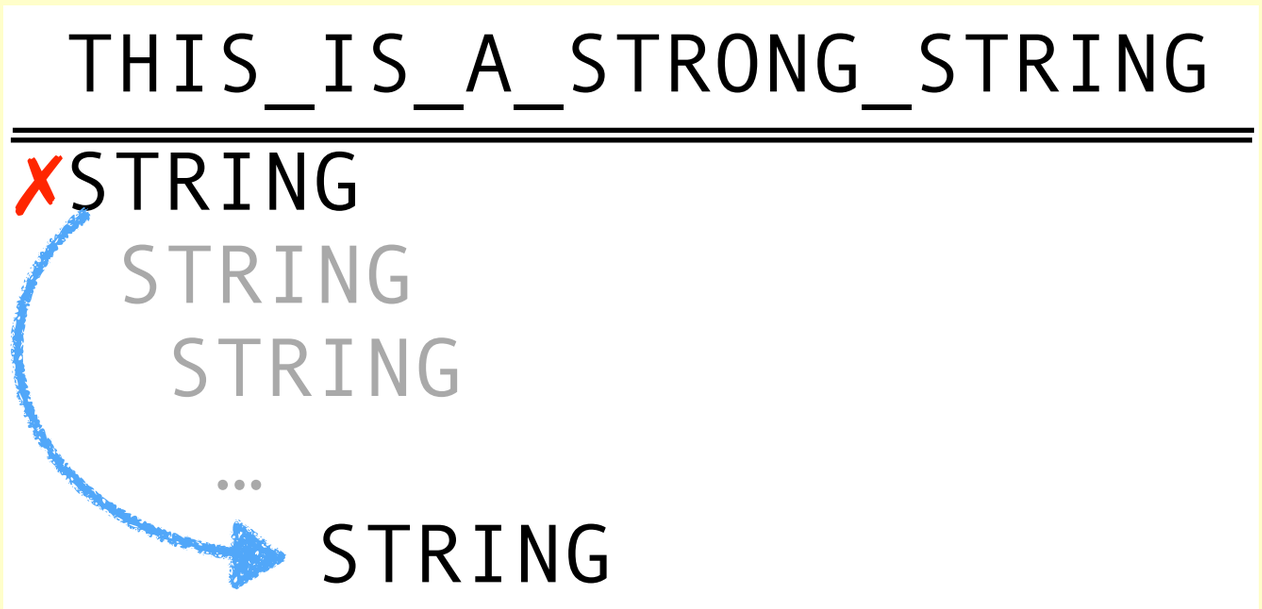
```
 THIS_IS_A_STRONG_STRING
═══════════════════════
✗STRING
✗ STRING
✗  STRING
✗   STRING
✗    STRING
                ⋮
```

Output

$$\{(i,j) \mid w(i,j) = pat\} = \{(18,23)\}$$

```
12345678901234567890123
THIS_IS_A_STRONG_STRING
                  STRING
```

String Matching Enhanced by Skipping



1. Pre-compute a look-up table (LUT) of **skipping values**
2. Try matching. Say it fails
3. Shift the pattern by **more than one**, based on the LUT
4. Try matching again

- (Potential) **constant speedup**, by skipping matching trials
- Skipping values **pre-computed**
  ➜ minimal computational overhead

The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms ’07]

- Interleaves two skipping mechanisms
  - KMP Algorithm [Knuth, Morris, and Pratt, SIAM J. Comput. ’77]
  - Quick Search [Sunday, Commun. ACM ’90]

The FJS Algorithm

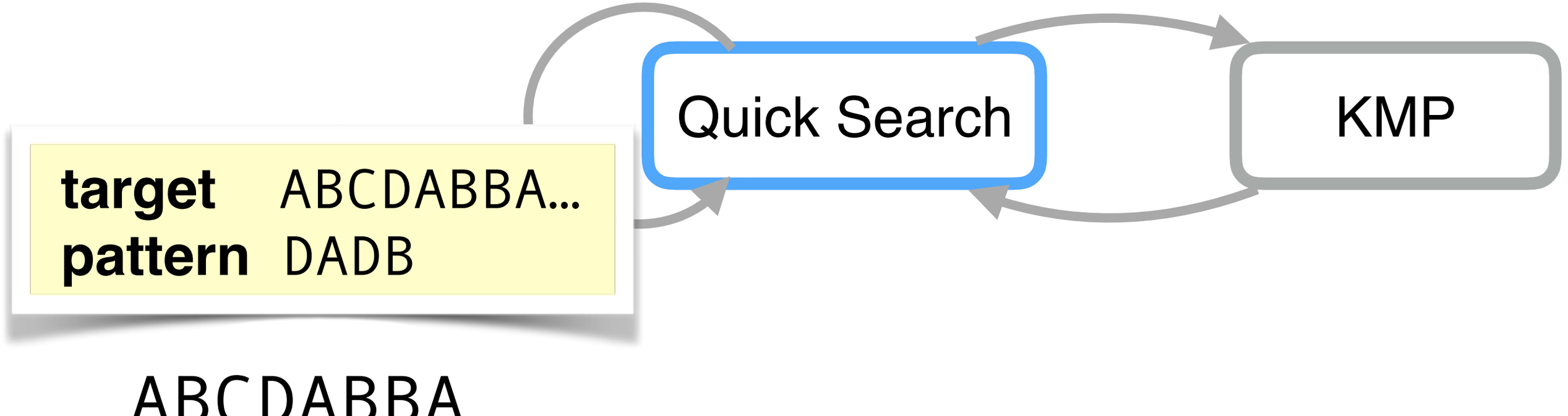
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

ABCDABBA…

DADB

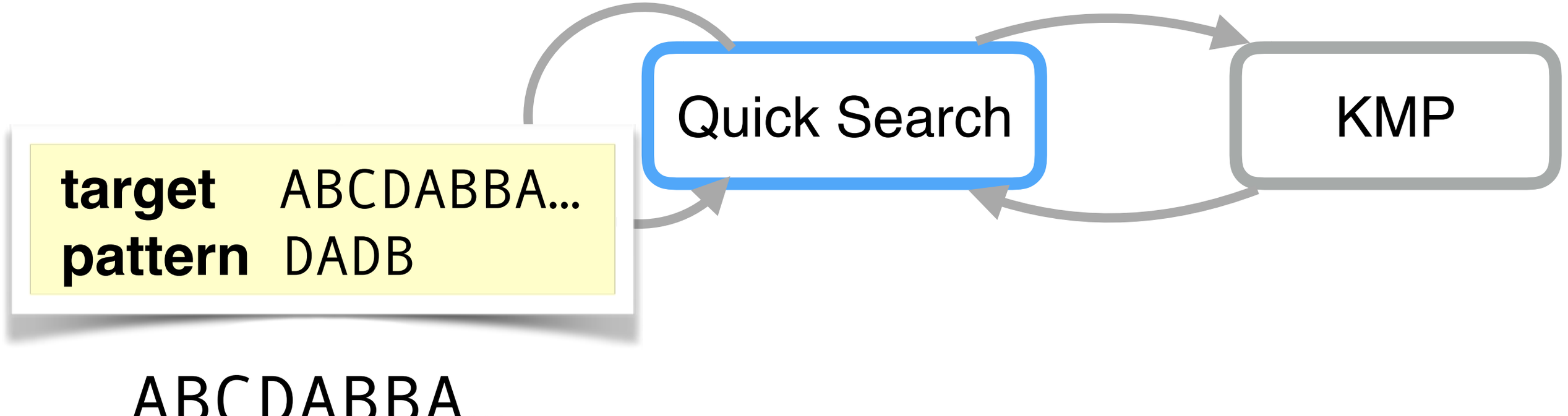
The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



The FJS Algorithm

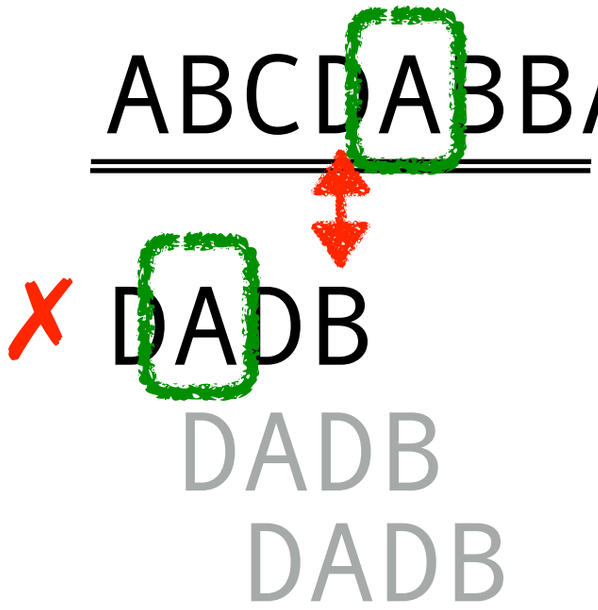
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



The FJS Algorithm

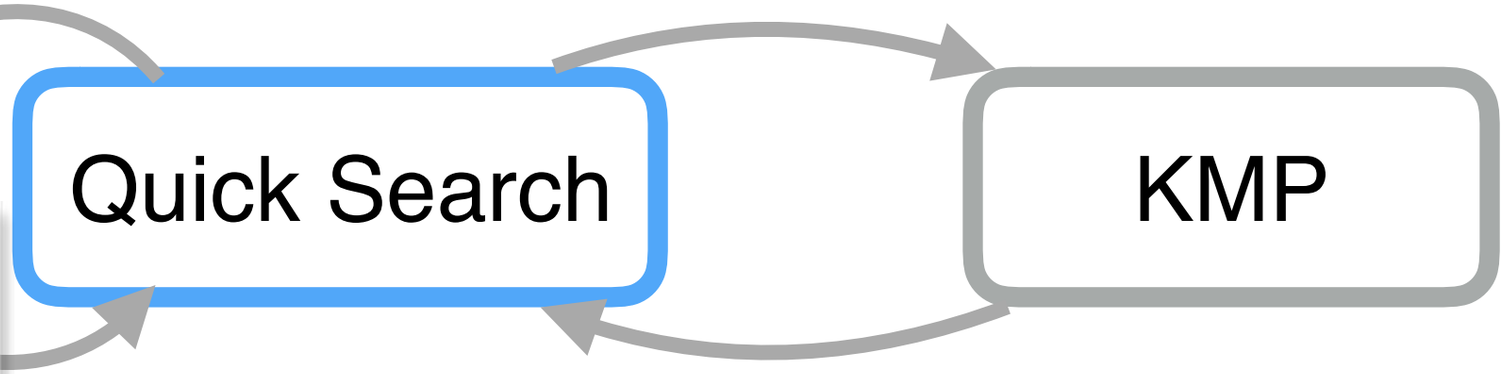
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

target ABCDABBA…
pattern DADB

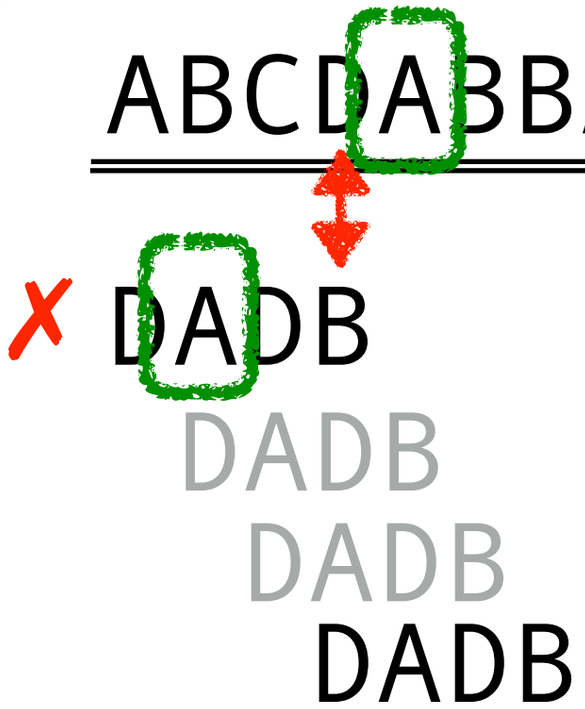


The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



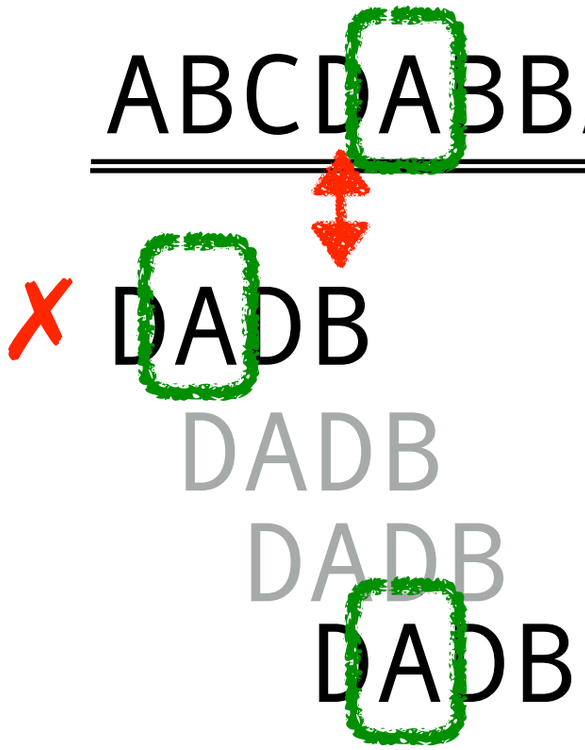
target ABCDABBA…
pattern DADB



The FJS Algorithm

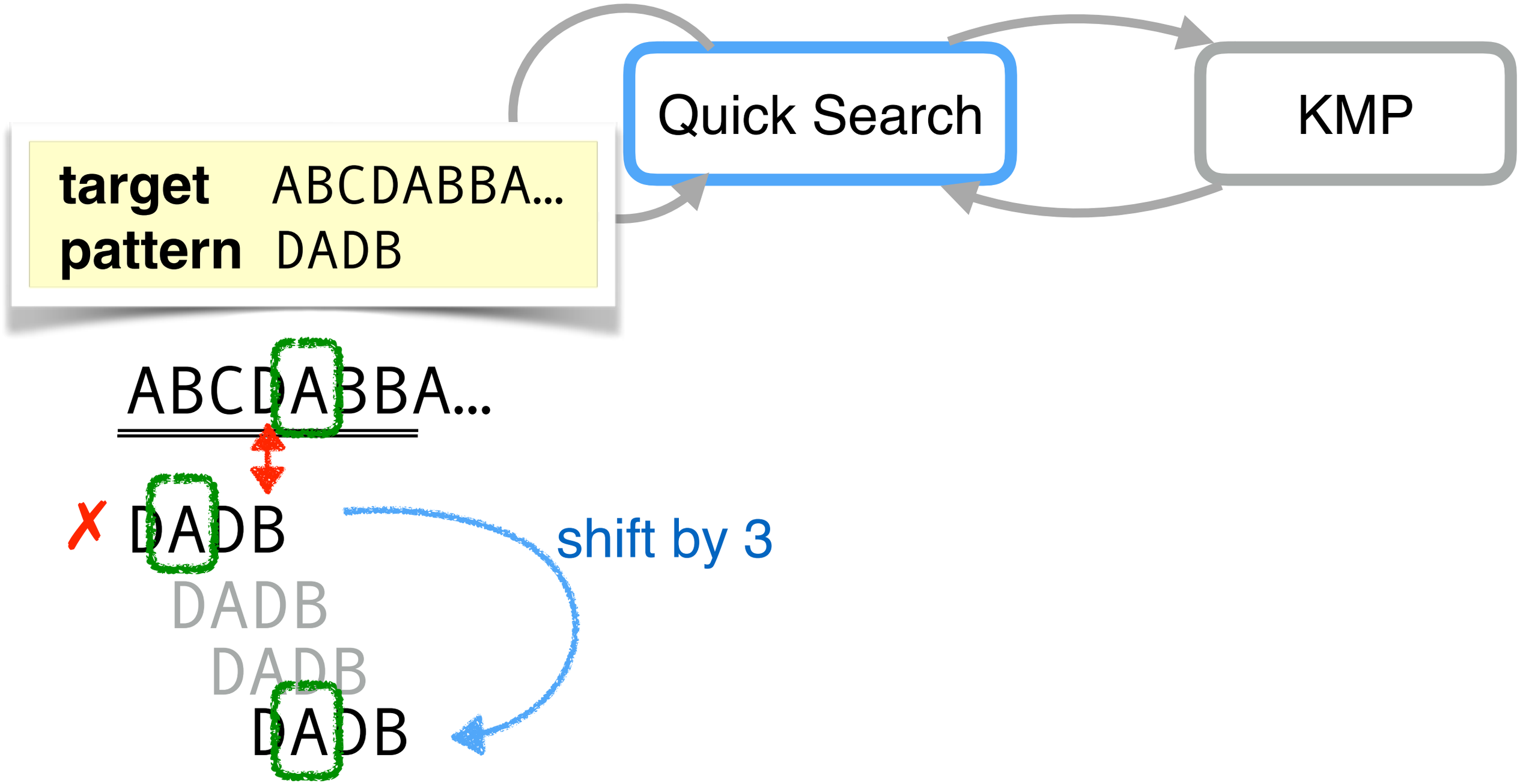
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

target ABCDABBA…
pattern DADB



The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

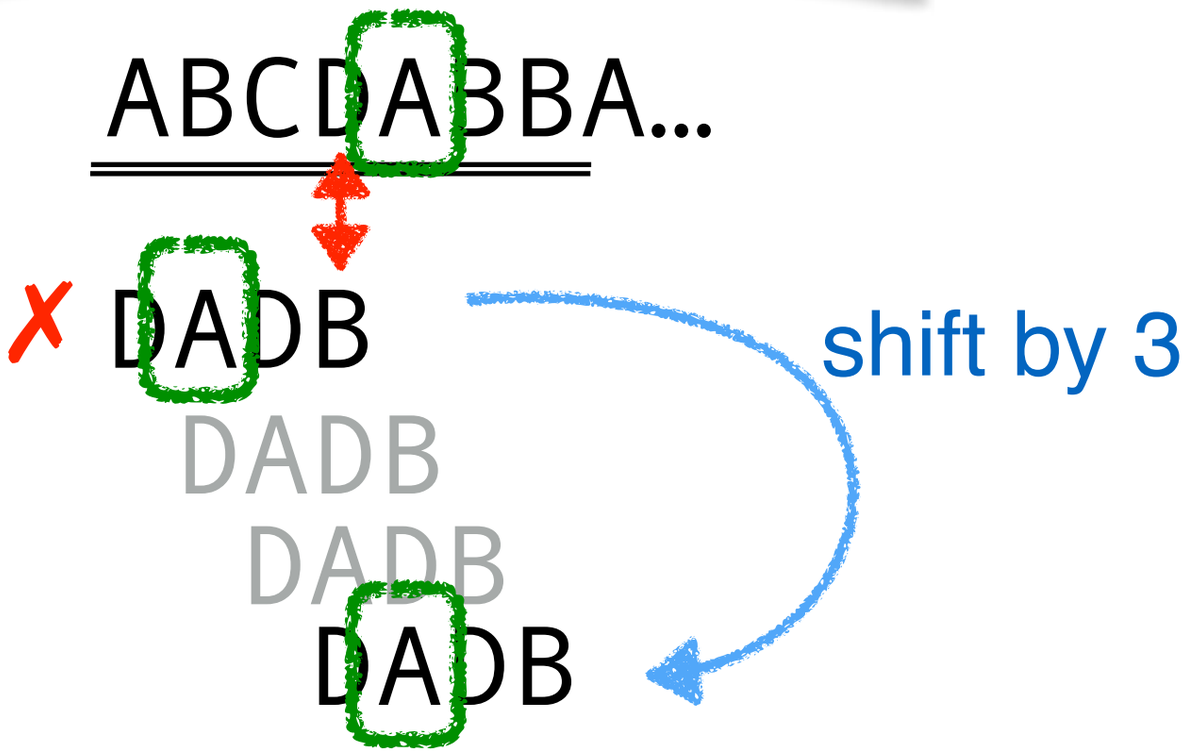


The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

Quick Search → KMP

target ABCDABBA…
pattern DADB

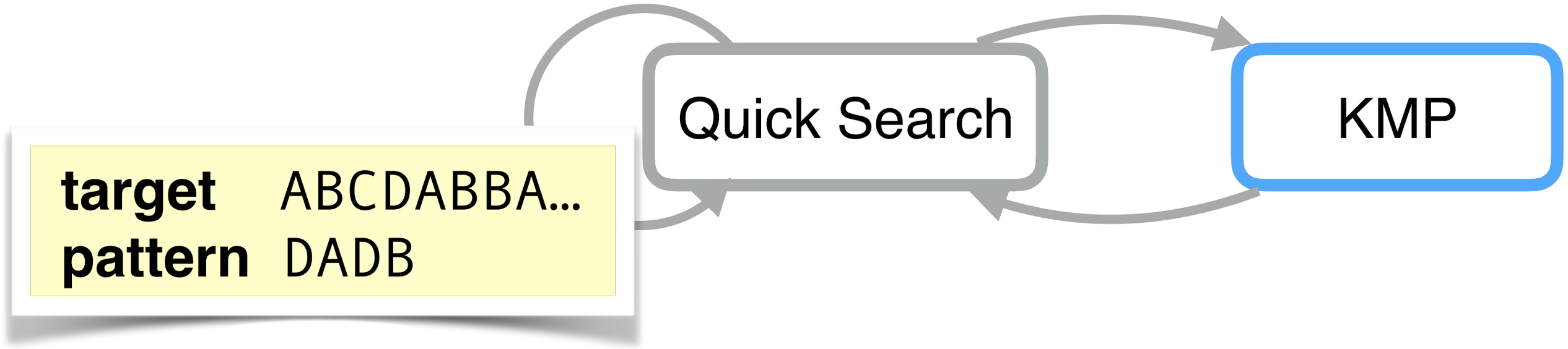


QS skip value table:

- **skip: Σ → N**
- depends on pattern but not on target
- for the pattern DADB:
  A→3, B→1, C→5,
  D→2

The FJS Algorithm

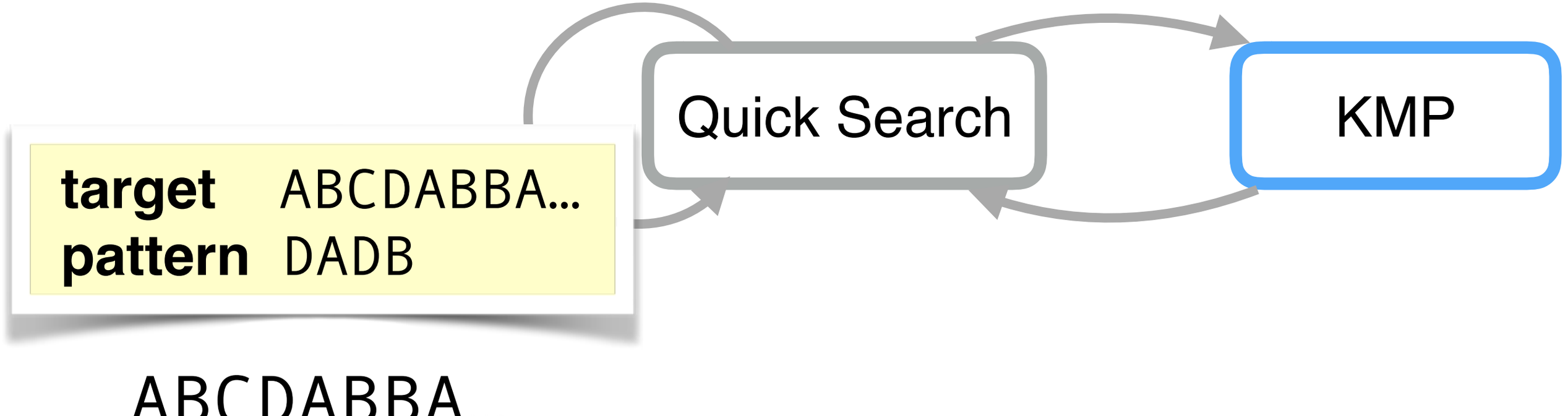
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]




```
ABCDABBA…
✗ DADB
   DADB
    DADB
     DADB
```

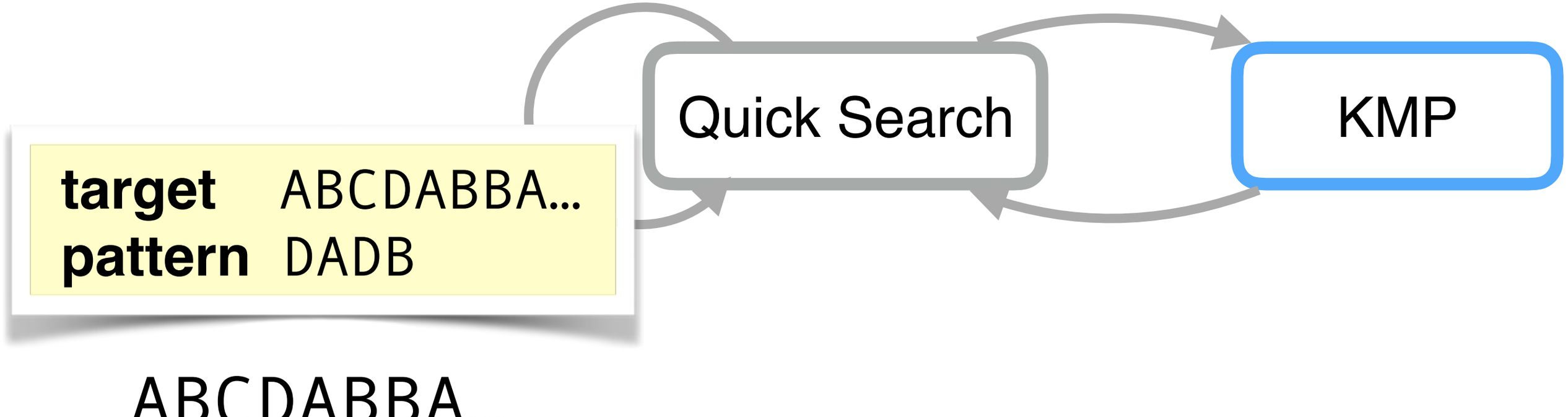
The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



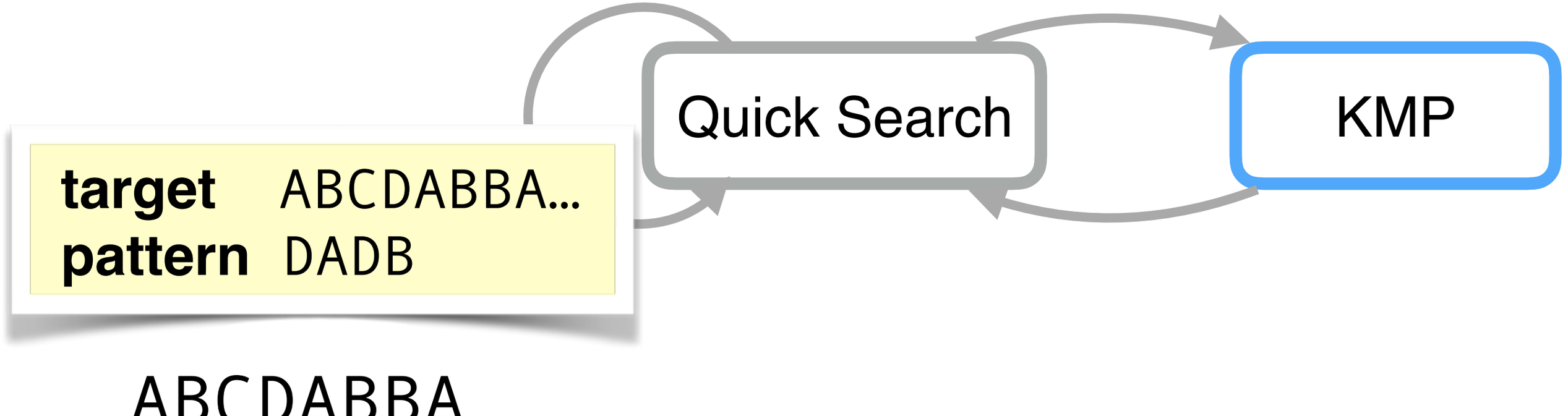
The FJS Algorithm

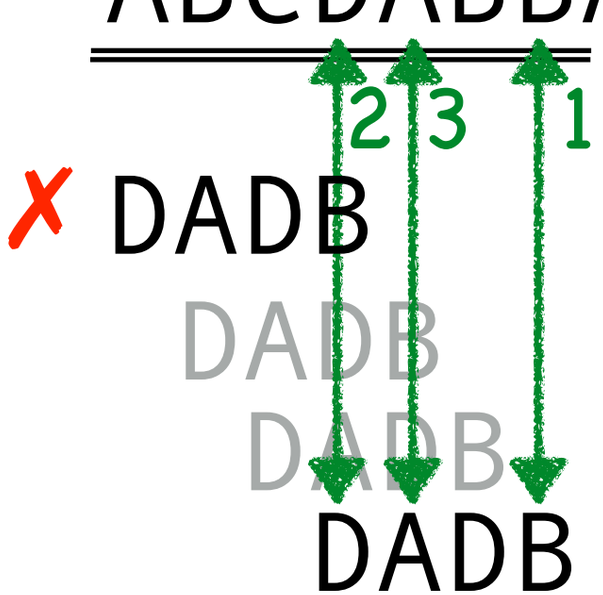
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



The FJS Algorithm

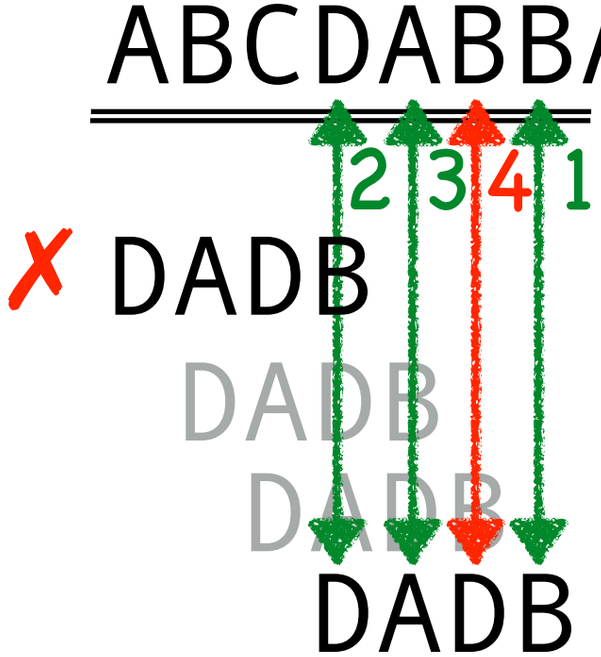
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]





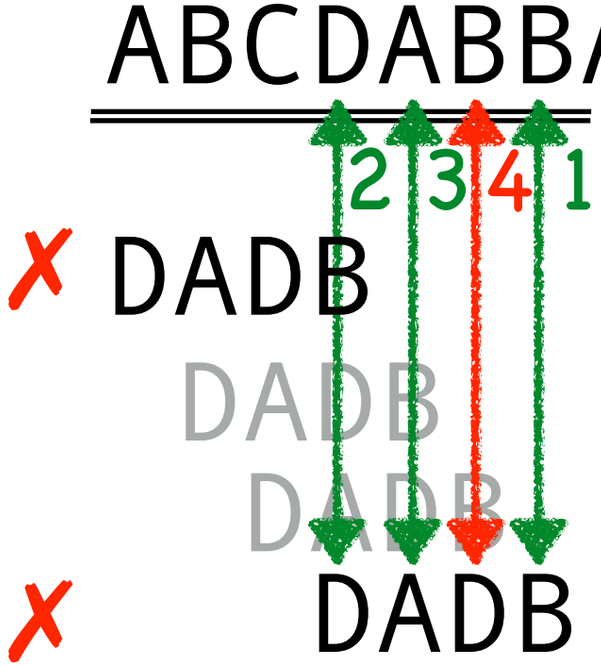
The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



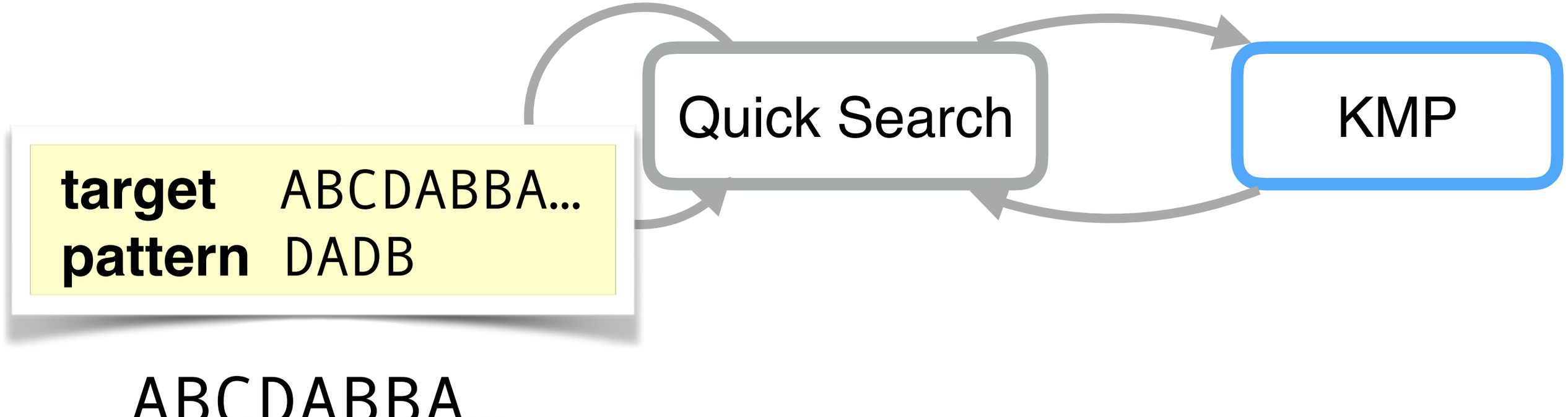
The FJS Algorithm

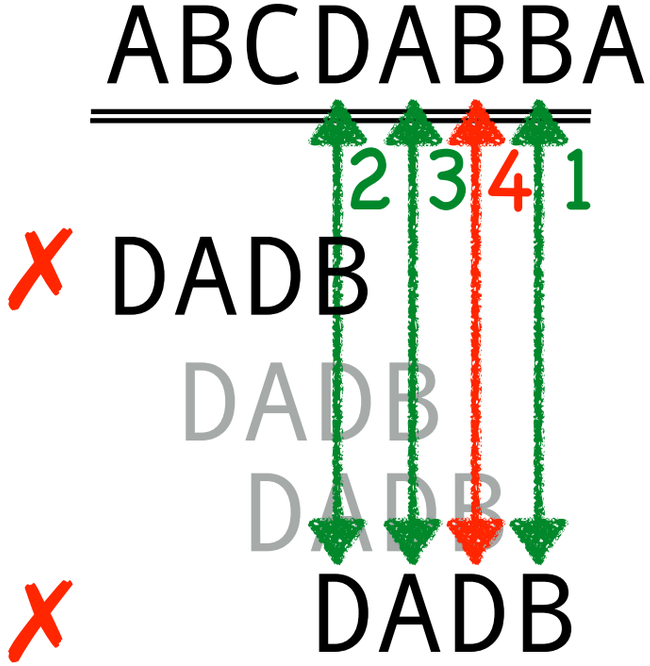
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



The FJS Algorithm

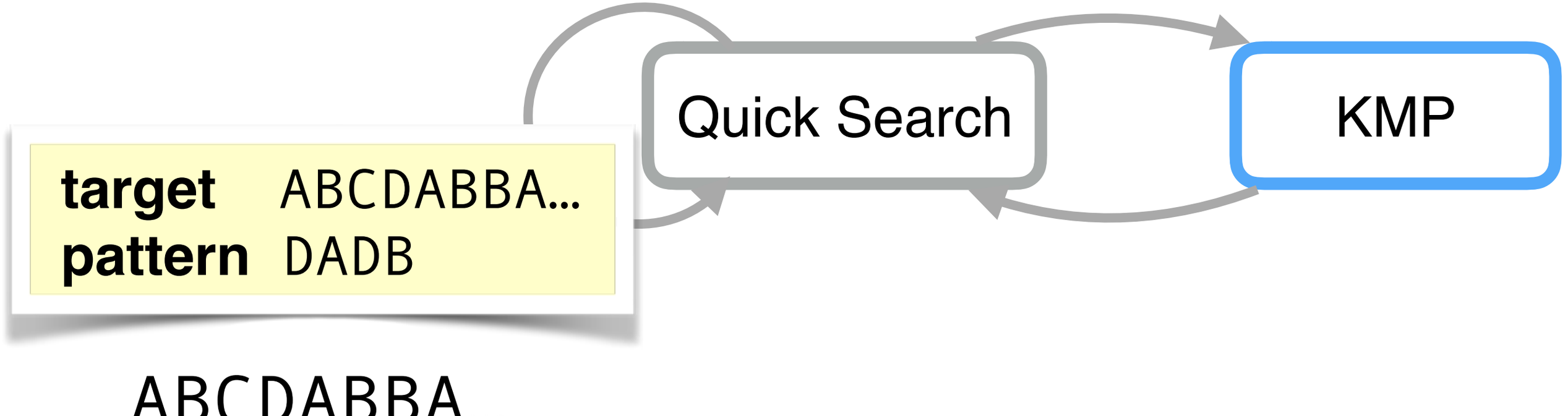
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]





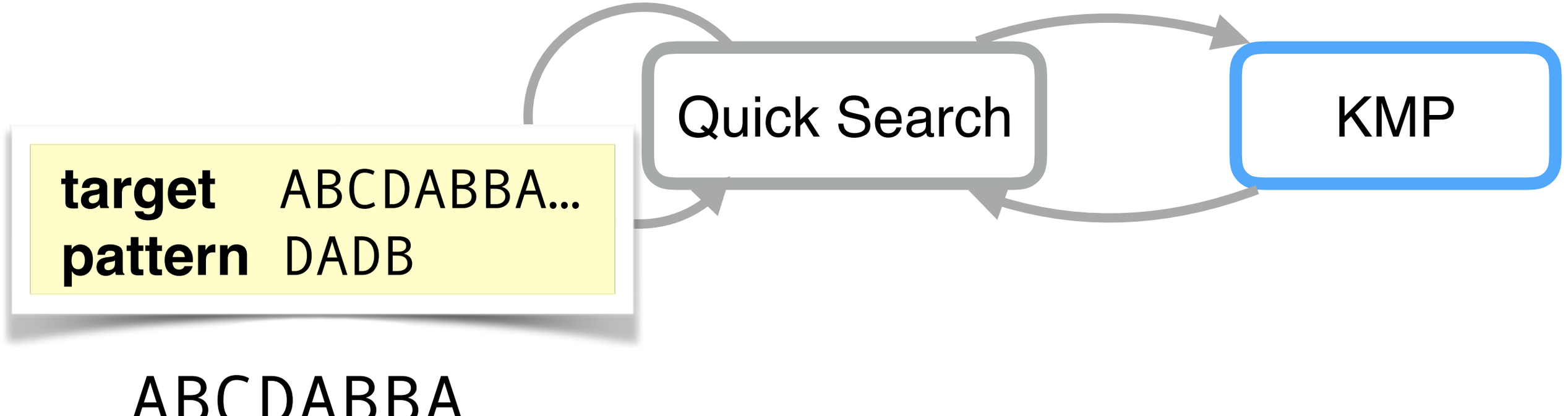
The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



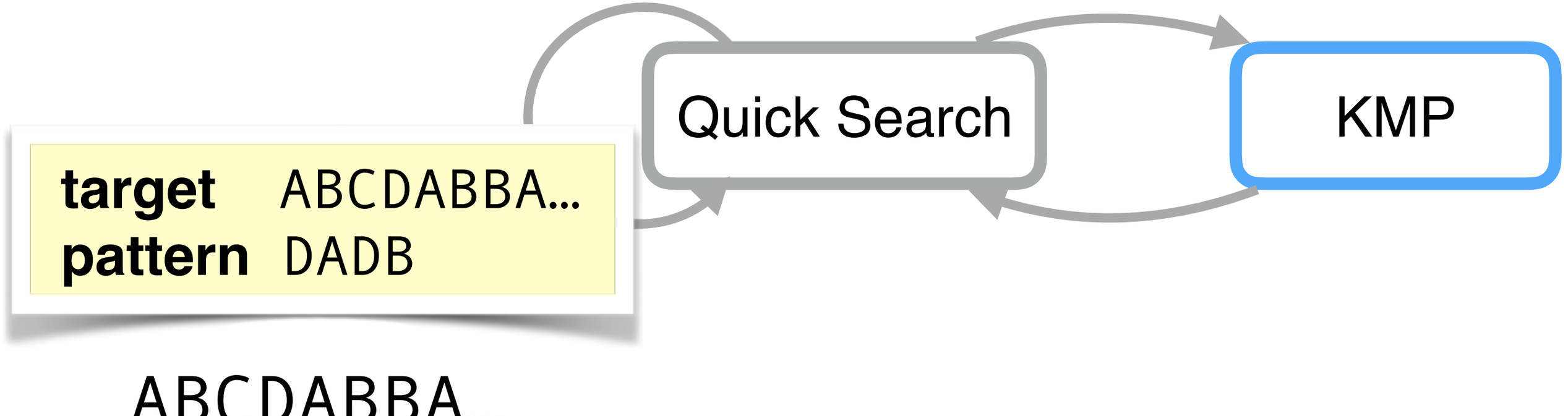
The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



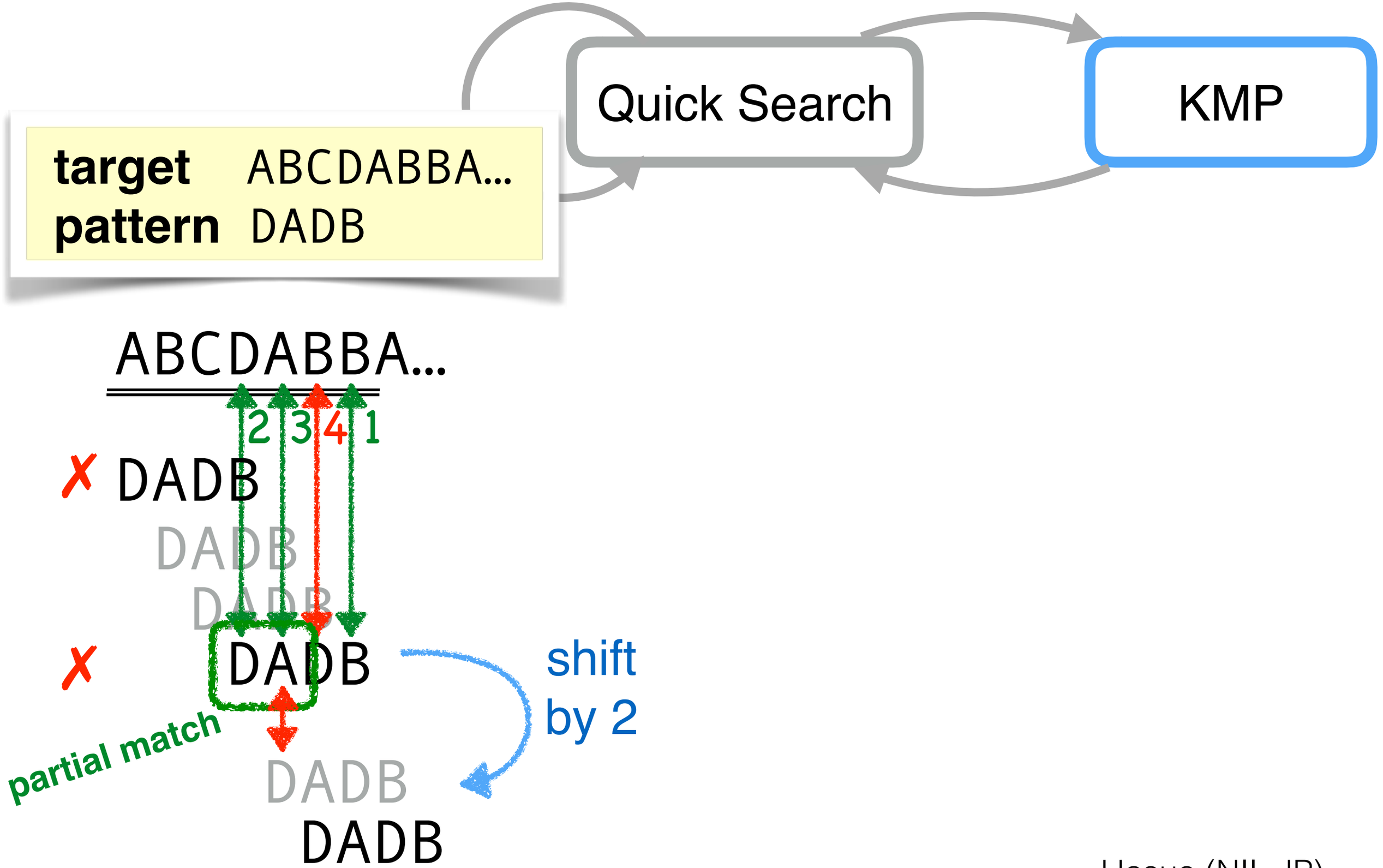
The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]



The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

Quick Search → KMP

target ABCDABBA…
pattern DADB

ABCDABBA…

2 3 4 1

✗ DADB
DADB
DADB
✗ DADB
DADB
DADB

partial match

shift by 2

partial match

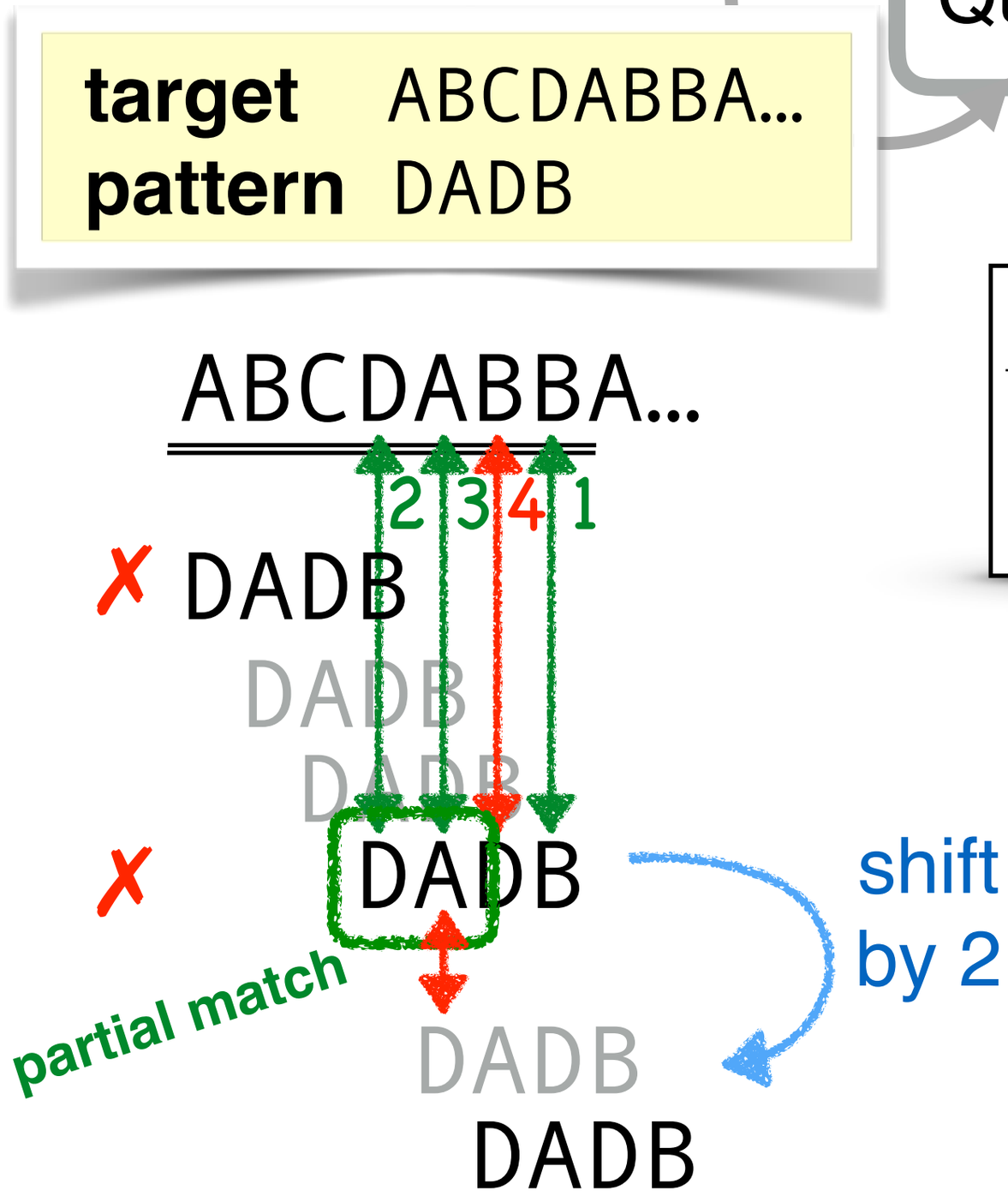
	D A D B
✗	* D A D
✔	* * D A

The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

Quick Search → KMP

target ABCDABBA…
pattern DADB



partial match

	D A D B
✗	* D A D
✔	* * D A

KMP skip value table:

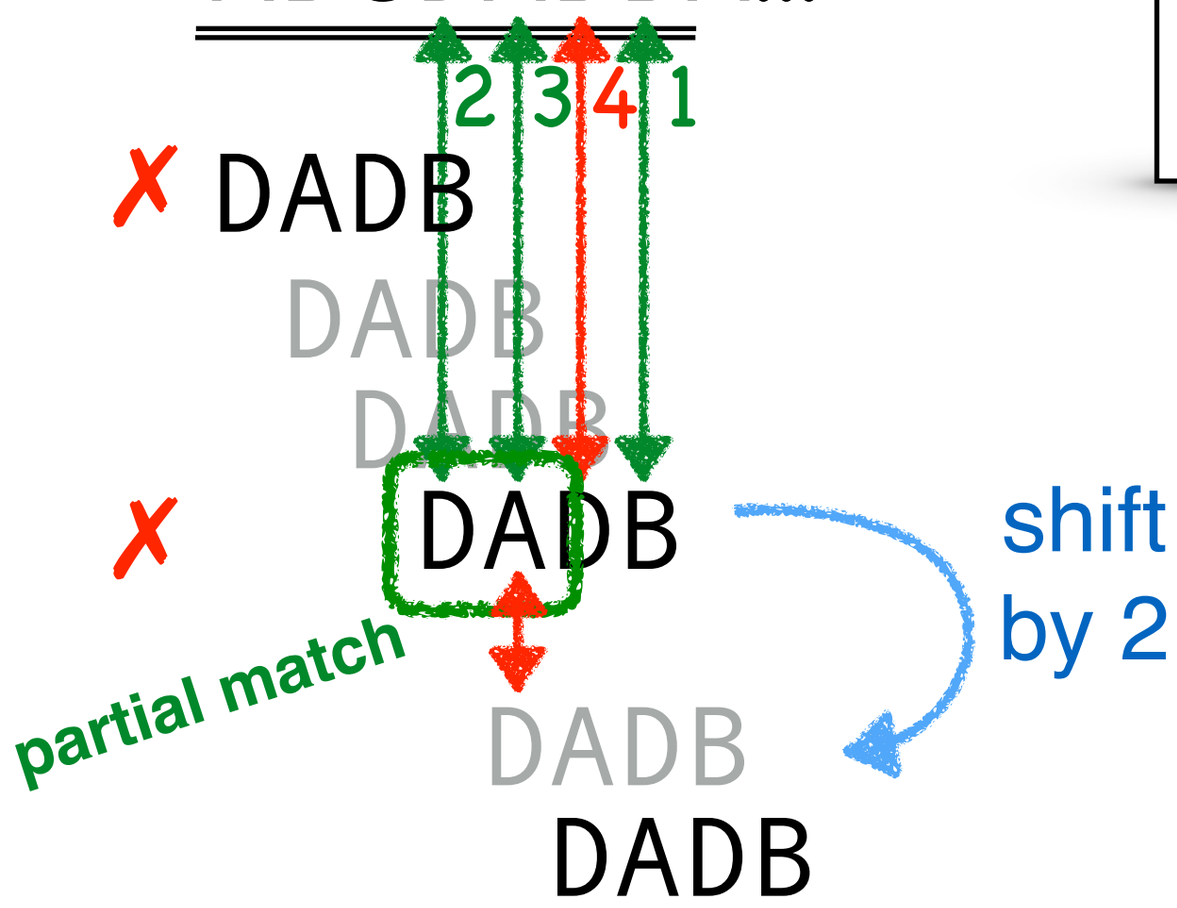
- **skip: [0, |pat|+1] → N.**
  The arg. is the length of partial match
- match **pattern** w/ **shifted pattern**, and detect **inconsistency**
- depends on pattern but not on target
- for the pattern DADB:
  0→1, 1→1, 2→2, 3→2, 4→2

The FJS Algorithm

[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

Quick Search → KMP

target ABCDABBA…
pattern DADB



partial match

	D A D B
✗	* D A D
✔	* * D A

partial match

	D A D B
✗	* D A D
✔	* * D A

KMP skip value table:

- **skip: [0, |pat|+1] → $\mathbb{N}$.**
  The arg. is the length of partial match
- match **pattern** w/ **shifted pattern**, and detect **inconsistency**
- depends on pattern but not on target
- for the pattern DADB:
  0→1, 1→1, 2→2, 3→2, 4→2

Contribution: An Efficient Algorithm for **Online** Timed Pattern Matching

The BM Algorithm (String Matching)
[Boyer & Moore, Commun. ACM '77]

BM Pattern Matching
[Watson & Watson, Sci. Comput. Program. '03]

BM Timed Pattern Matching
[Waga, Akazaki, and Hasuo, FORMATS '16]

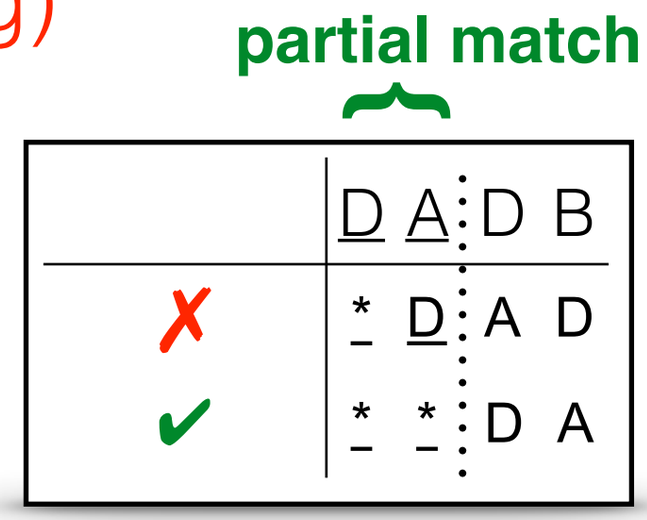
The FJS Algorithm (String Matching)
[Franek, Jennings, and Smyth, J. Discrete Algorithms '07]

FJS Pattern Matching
[Current work]

FJS Timed Pattern Matching
[Current work]

Outline

1. Original FJS Algorithm (for String Matching)
2. Our FJS Algorithm for Timed Pattern Matching
   (We focus on the KMP-type skipping)
3. Experiments



KMP-Type Skipping for (Timed) Pattern Matching

pattern/spec is an automaton

String Matching

- Compare pattern w/ shifted pattern, and detect inconsistency
- **skip: [0, |pat|+1] → $\mathbb{N}$**

partial match

	D A D B
✗	* D A D
✔	* * D A

(Timed) Pattern Matching

??

- "Length of partial match" doesn't say much (many different "partial matches" for an automaton!)
- What is "inconsistency"?

KMP-Type Skipping for (Timed) Pattern Matching

String Matching

- Compare pattern w/ shifted pattern, and detect inconsistency
- **skip: [0, |pat|+1] → $\mathbb{N}$**

partial match

	D A D B
✗	* D A D
✓	* * D A

(Timed) Pattern Matching

- "Length of partial match" doesn't say much
  ➜ **Use "to which automaton state the partial match has led"**

	The words leading to s	$= L_s$
✗	*	$L(\mathcal{A})$
✗	* *	$L(\mathcal{A})$
✓	* * *	$L(\mathcal{A})$

KMP-Type Skipping for (Timed) Pattern Matching

String Matching

- Compare pattern w/ shifted pattern, and detect inconsistency
- **skip: [0, |pat|+1] → N**

partial match

	D A D B
✗	* D A D
✓	* * D A

(Timed) Pattern Matching

	The words leading to s	$= L_s$
✗	*	$L(\mathcal{A})$
✗	* *	$L(\mathcal{A})$
✓	* * *	$L(\mathcal{A})$

- What is “inconsistency”?

→

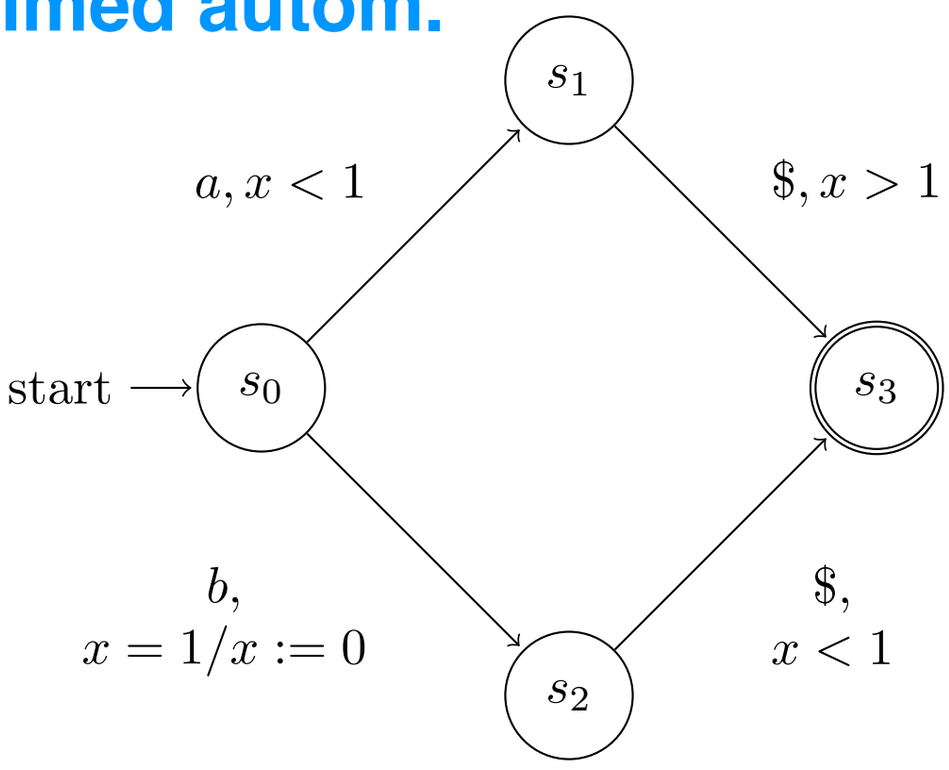
Is there any word

- **that leads to state s, and**
- **whose suffix survives in A?**

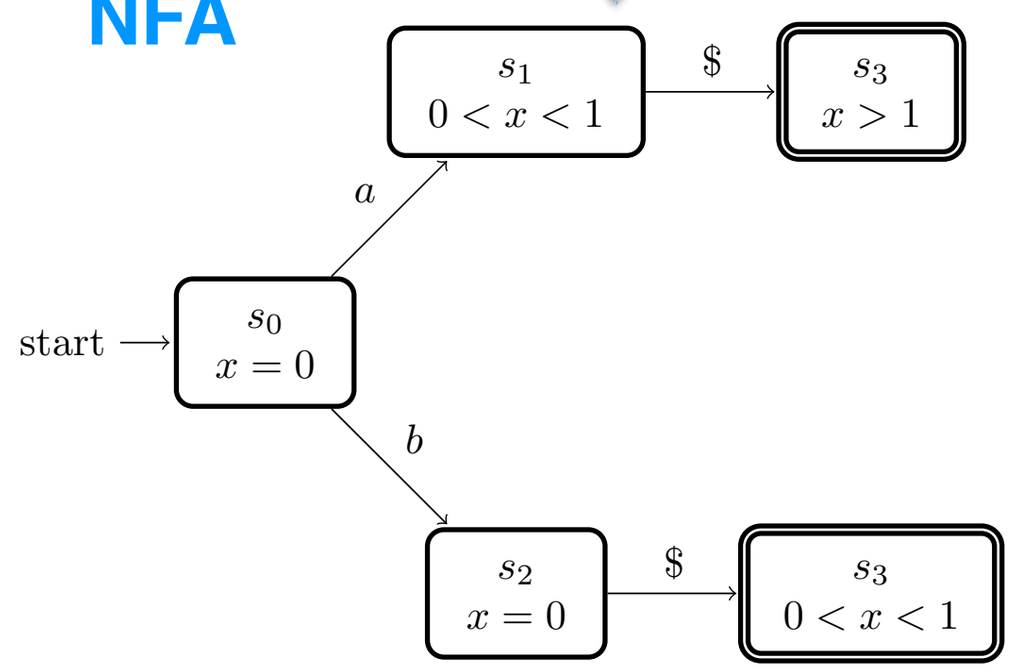
Reachability Checking by Zone Construction

Labelled by “zones,” i.e.
sets of “similar” clock valuations

timed autom.



NFA



Thm. (soundness and completeness)
Zone automaton construction maintains state reachability.

KMP-Type Skipping for (Timed) Pattern Matching

String Matching

- Compare pattern w/ shifted pattern, and detect inconsistency
- **skip: [0, |pat|+1] → N**

partial match

	D A D B
✗	* D A D
✓	* * D A

(Timed) Pattern Matching

	The words leading to s	$= L_s$
✗	*	$L(\mathcal{A})$
✗	* *	$L(\mathcal{A})$
✓	* * *	$L(\mathcal{A})$

- What is “inconsistency”?

→

Is there any word

- **that leads to state s, and**
- **whose suffix survives in A?**

KMP-Type Skipping for (Timed) Pattern Matching

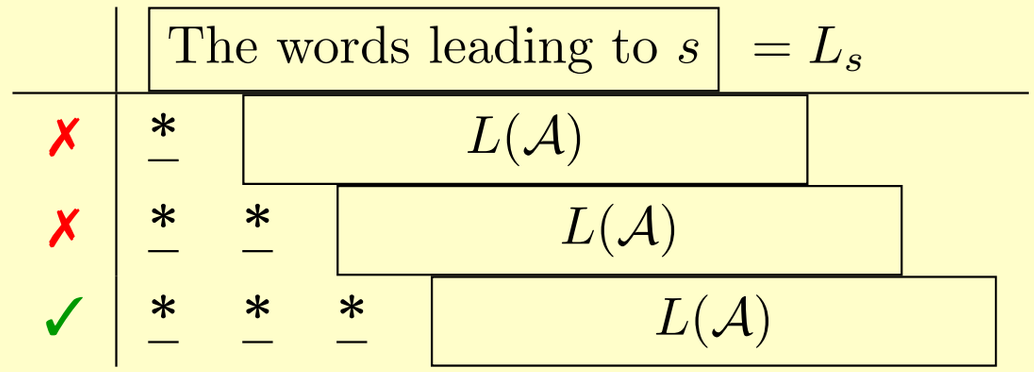
String Matching

- Compare pattern w/ shifted pattern, and detect inconsistency
- skip: $[0, |pat|+1] \to N$

Solved by:

- making copies A_1, A_2 of A (changing initial/final states),
- taking product $A_1 \times A_2$ (for intersection), and
- reachability check by zone construction

(Timed) Pattern Matching



- What is "inconsistency"?

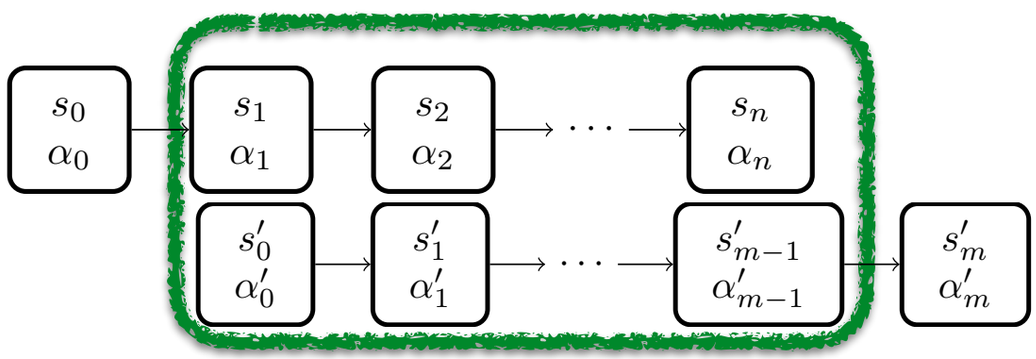
→

Is there any word

- **that leads to state s, and**
- **whose suffix survives in A?**

KMP-Type Skipping for rn Matching

In other words…



Reachability checking of this segment
→ intersection and zone construction

Str

- C
- s
- C

- **skip: [0, |pat|+1] → N**

partial match

	D	A	D	B
✗	*	D	A	D
✓	*	*	D	A

(Timed) Pattern Matching

	The words leading to s			$= L_s$
✗	*			$L(\mathcal{A})$
✗	*	*		$L(\mathcal{A})$
✓	*	*	*	$L(\mathcal{A})$

- What is “inconsistency”?

→

Is there any word

- **that leads to state s, and**
- **whose suffix survives in A?**

KMP-Type Skipping for (Timed) Pattern Matching: **Summary**

String Matching

- Compare pattern w/ shifted pattern, and detect inconsistency
- **skip: [0, |pat|+1] → N**

partial match

	D A	D B
✗	* D	A D
✓	* *	D A

(Timed) Pattern Matching

	The words leading to s	$= L_s$
✗	*	$L(\mathcal{A})$
✗	* *	$L(\mathcal{A})$
✓	* * *	$L(\mathcal{A})$

- "Length of partial match"
  ➜ "to which state the match has led"
- "Inconsistency"?
  ➜ reachability check in $A_1 \times A_2$
- **skip: S → N**

Outline

1. Original FJS Algorithm (for String Matching)
2. Our FJS Algorithm for Timed Pattern Matching
3. Experiments

Settings

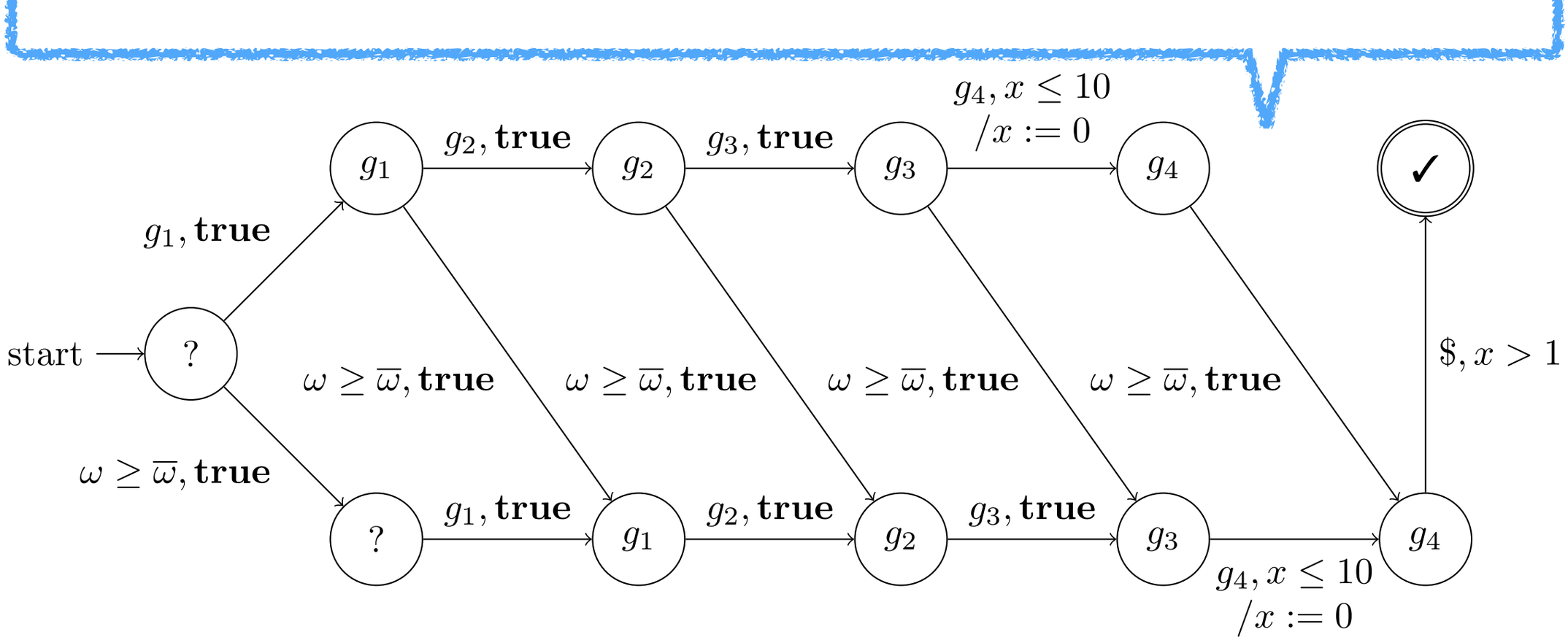
- Implementation in C++,
  run on MacBook Pro, 2.6 GHz Intel Core i5, 8 GB RAM
  (except the comparison w/ Montre)

- Problem:
  Monitoring of a Simulink model of an automatic transmission model
  - The model and specs are from [Hoxha et al., ARCH ’15]
  - Logs are generated by simulation of the Simulink model

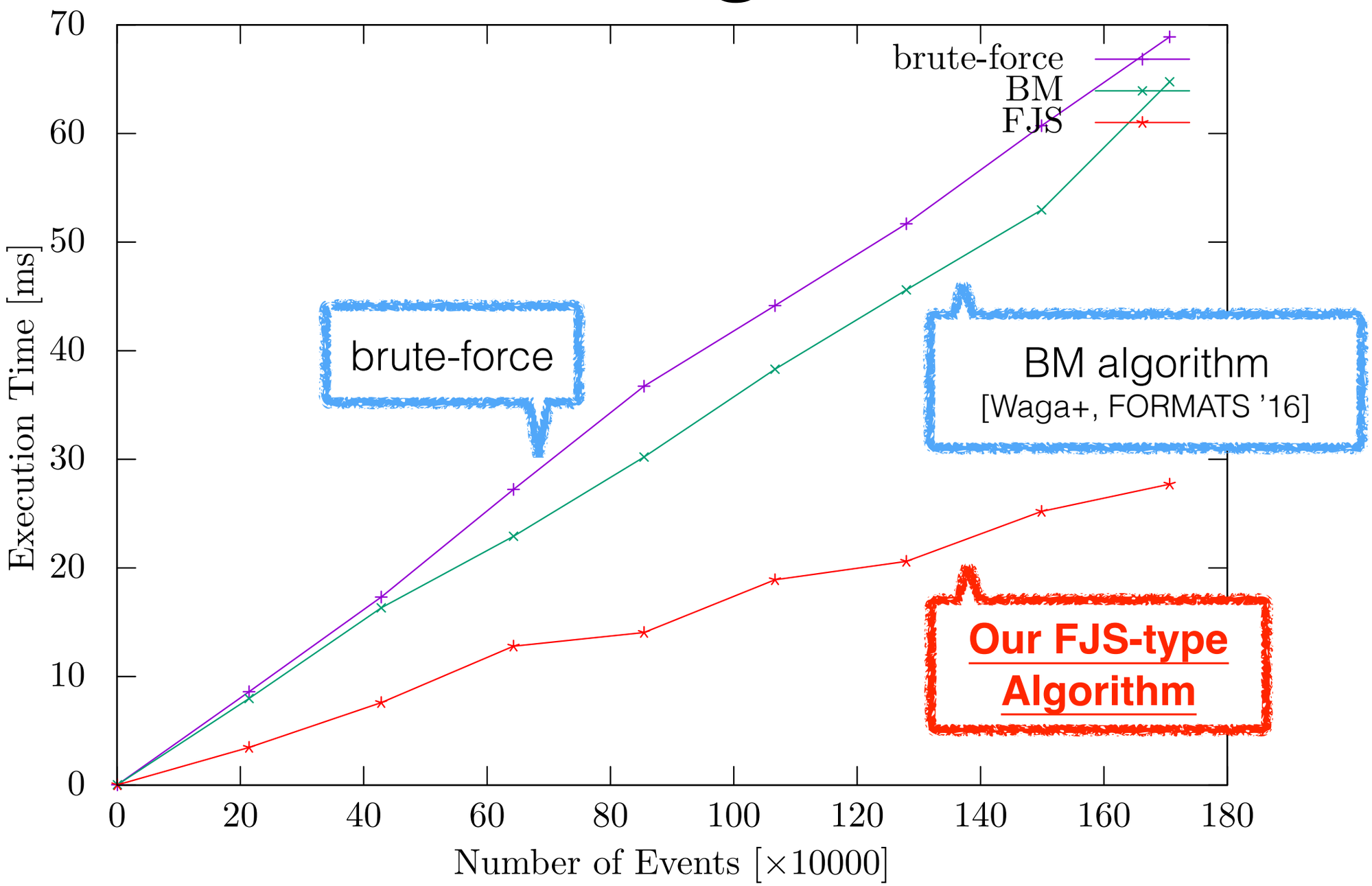
- Events: gears: g_1, g_2, g_3, g_4
  velocity: $v > \overline{v},\ v \leq \overline{v}$
  RPM: $\omega > \overline{\omega},\ \omega \leq \overline{\omega}$

Specification: Example

Gear changes from g_1 to g_4 in 10 sec. and RPM changes to high enough, but velocity is still low.



Comparison with Our Previous Algorithms



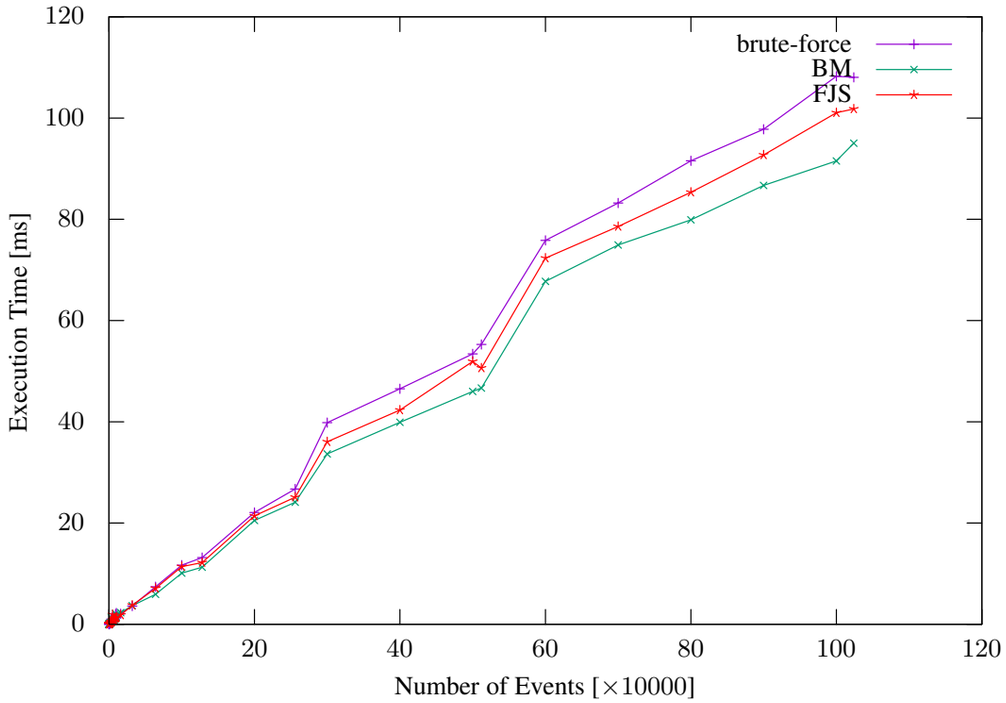


Fig. 15. SIMPLE: exec. time

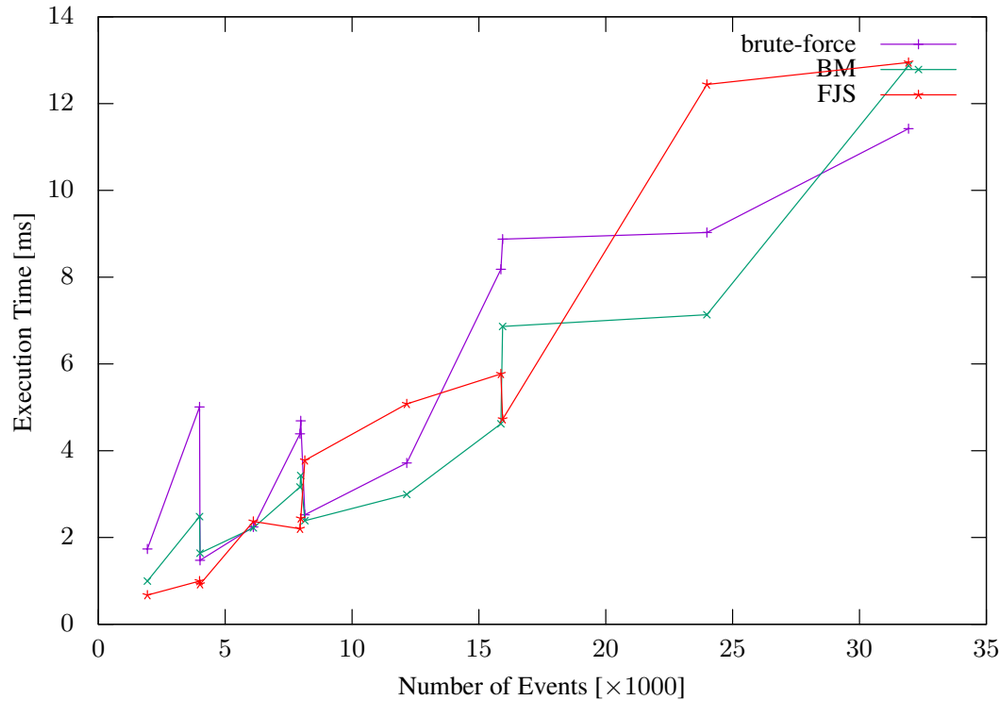


Fig. 16. LARGE CONSTRAINTS: exec. time

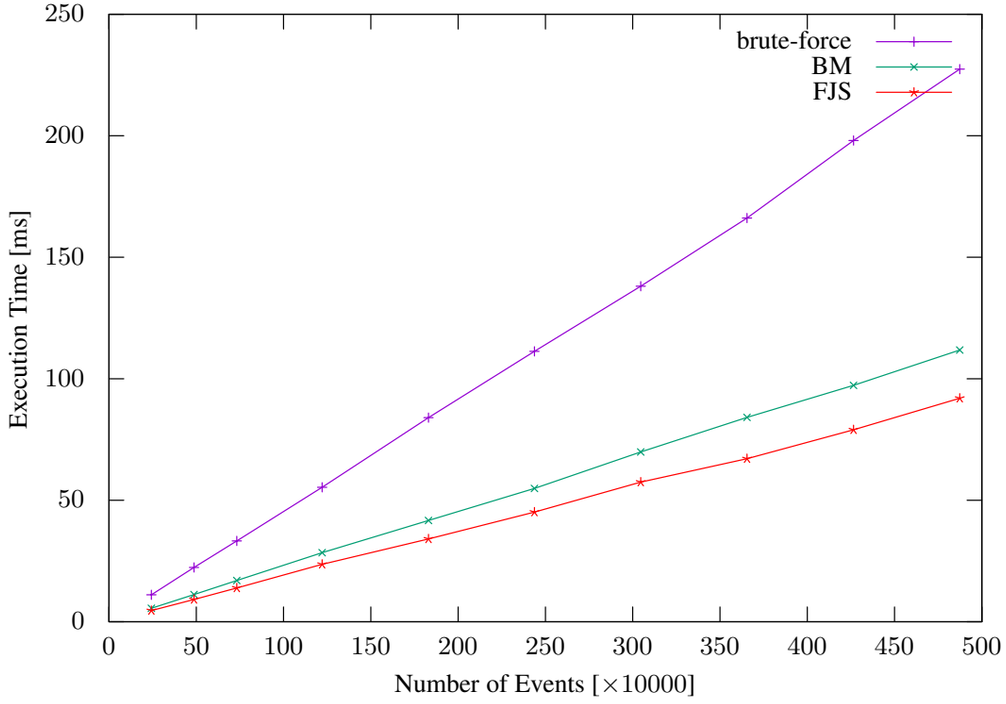


Fig. 17. TORQUE: exec. time

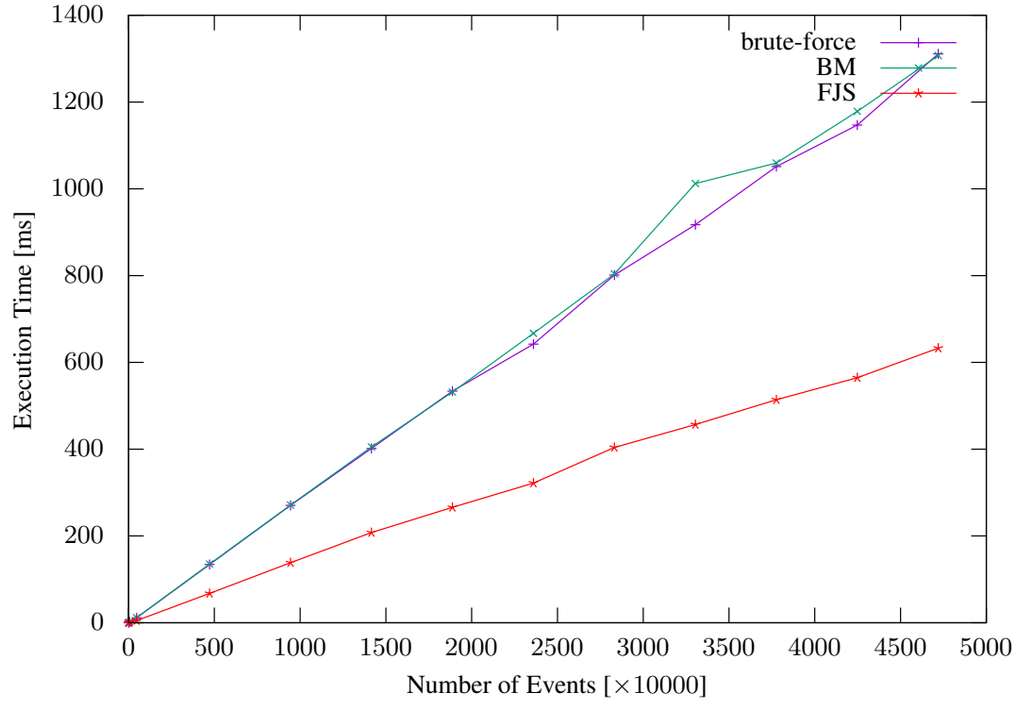


Fig. 18. SETTLING: exec. time

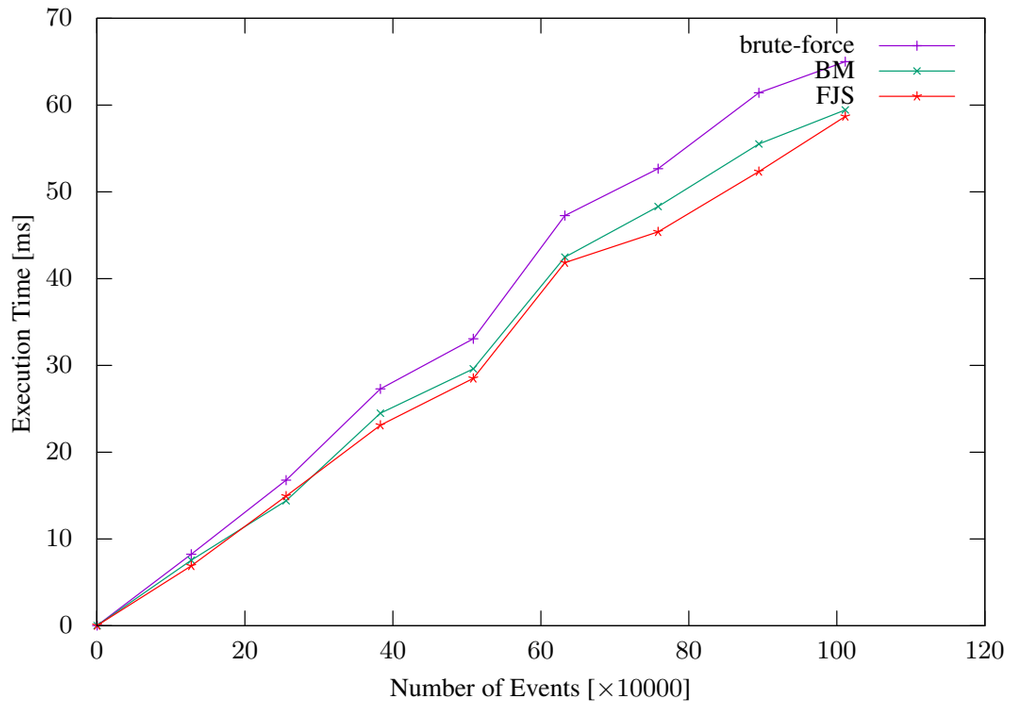


Fig. 19. GEAR: exec. time

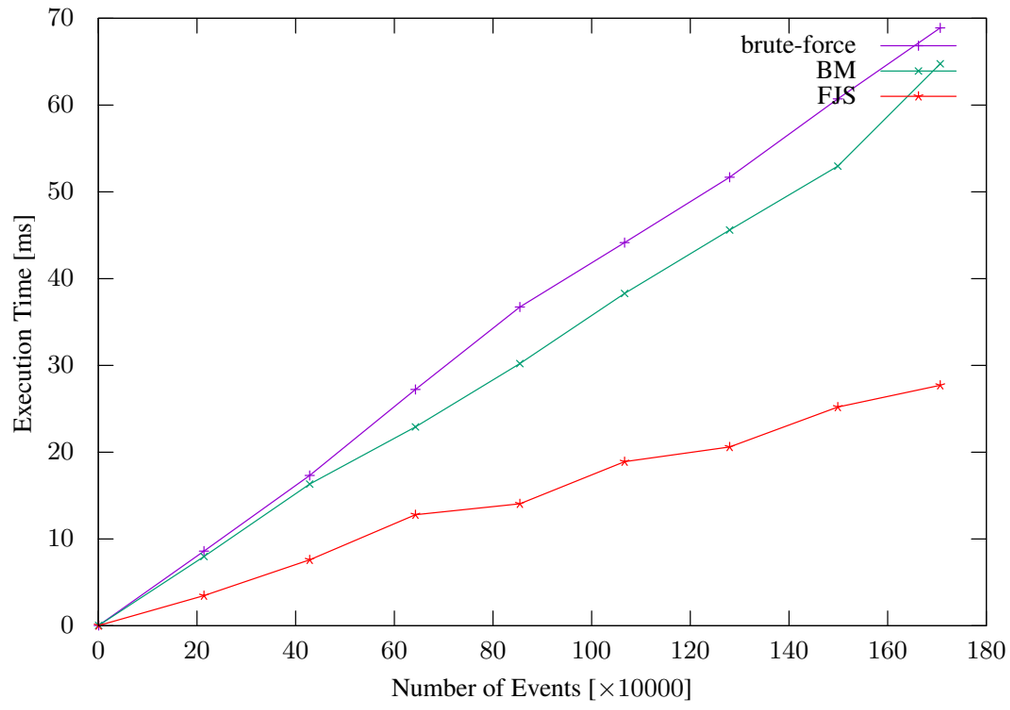


Fig. 20. ACCEL: exec. time

Online vs. Offline

Table 6. Memory consumption of FJS (online) and BM

$\|w\|$	BM (MB)	FJS (MB)
300	1.16	1.16
30,000	2.61	1.16
300,000	15.55	1.16
3,000,000	145.21	1.16
6,000,000	289.25	1.16
9,000,000	433.31	1.16
12,000,000	577.32	1.19
15,000,000	721.37	1.18
18,000,000	865.42	1.19
21,000,000	1,009.46	1.16
24,000,000	1,153.50	1.16
27,000,000	1,297.57	1.16
30,000,000	1,441.61	1.16

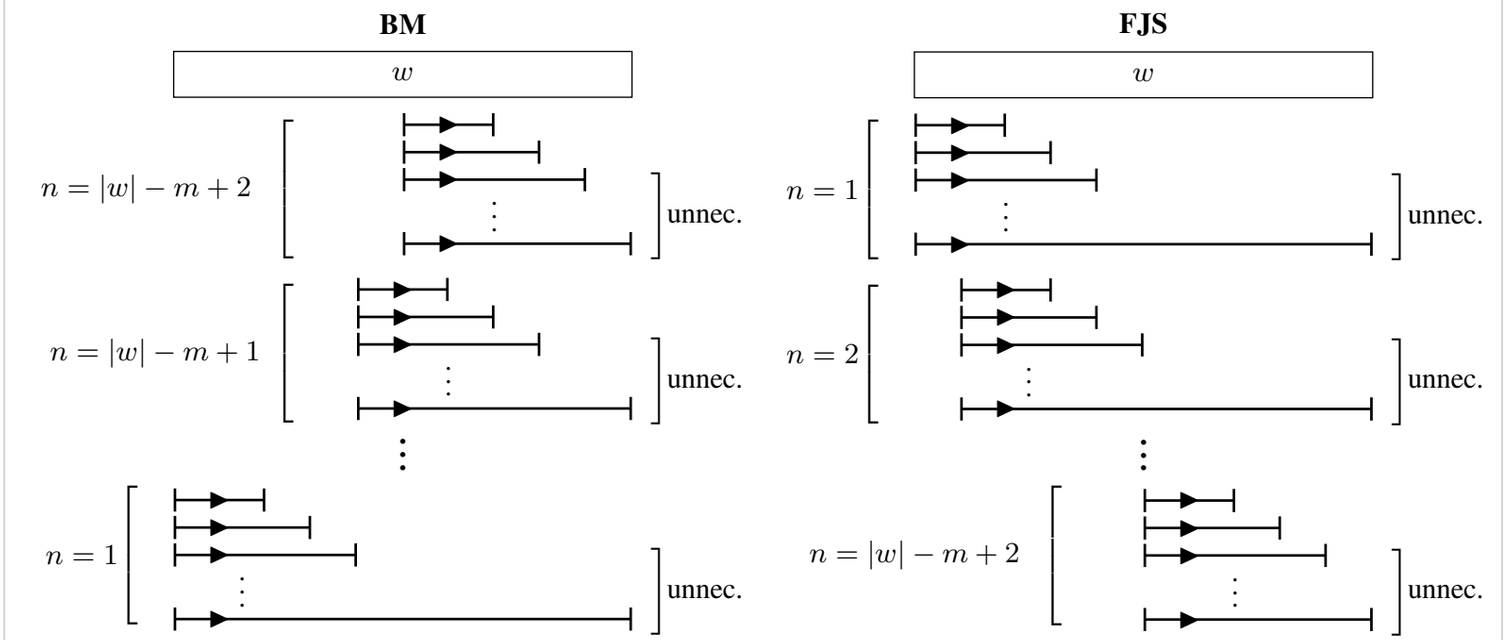


Fig. 8. How matching trials proceed: our previous BM-type algorithm (on the left) and our current FJS-type algorithm (on the right).

Comparison with **Montre**

D. Ulus. Montre: A tool for monitoring timed regular expressions. CAV 2017

$\|w\|$	FJS (online) (sec.)	Montre (offline) (sec.)	Montre (online) (sec.)
653	0.00	0.01	69.05
214,142	0.06	0.63	Timeout
428,428	0.13	1.25	Timeout
642,922	0.20	1.88	Timeout
854,456	0.26	2.50	Timeout
1,066,815	0.33	3.12	Timeout
1,279,713	0.40	3.75	Timeout
1,499,021	0.46	4.38	Timeout
1,706,614	0.53	4.99	Timeout

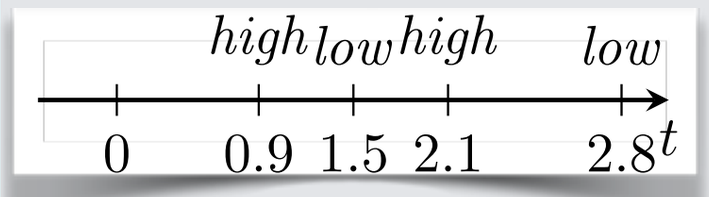
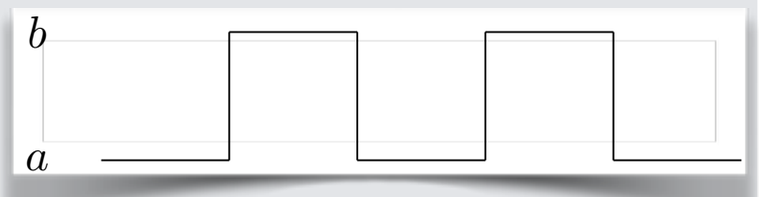
30 min.

Amazon EC2 c4.xlarge instance (April 2017, 4 vCPUs and 7.5 GB RAM), Ubuntu 14.04 LTS (64 bit)

- Performance gap may be due to impl. details
  (e.g. Montre is in **Pure**, a functional language based on rewriting)

Hasuo (NII, JP)

Comparison with Montre

	This Work	Montre [Ulus, CAV '17]
Pattern	Timed Automaton	Timed Regular Expression
Target String	Timed Word 	Signal 

Conclusions

- Efficient timed pattern matching with **skipping**
  - Skipping in string matching, adapted to (timed) automata
  - From BM (offline) to FJS (**online**)
    - Satisfactory online/offline performance (twice faster than brute-force, constant memory usage)
- Future work
  - Sophisiticated tool, frontend
  - Evaluation in real embedded applications
  - Event-based vs. signal-based

Conclusions

- Efficient timed pattern matching with **skipping**
  - Skipping in string matching, adapted to (timed) automata
  - From BM (offline) to FJS (**online**)
    - Satisfactory online/offline performance (twice faster than brute-force, constant memory usage)
- Future work
  - Sophisiticated tool, frontend
  - Evaluation in real embedded applications
  - Event-based vs. signal-based
- "Anything you can do, I can do better with automata" [Vardi?]

Appendix

Pattern Matching

Input

Target String

- String w : dbadcdc
- Regular Language L : dc*{ba|dc}

Pattern

Output

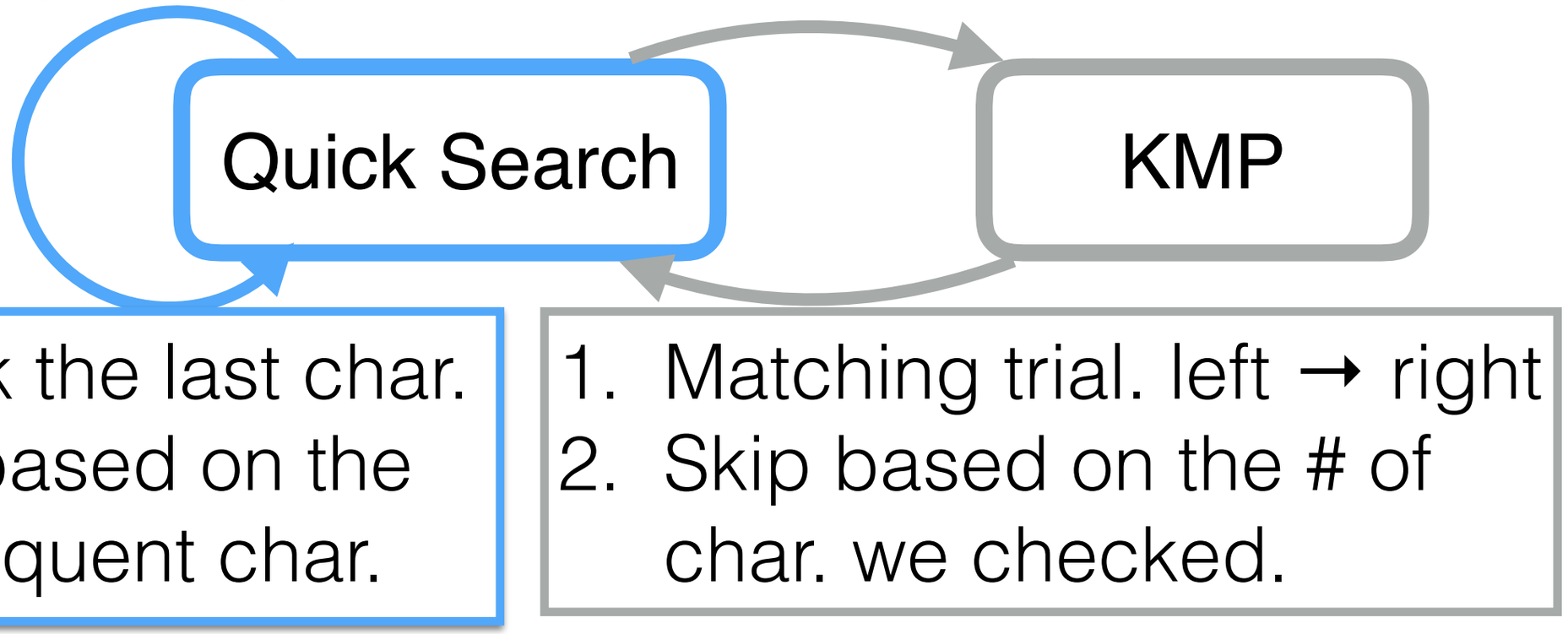
$$\{(i,j) \mid w(i,j) \in L\} = \{(1,3),(4,7)\}$$

$$w(1,3) = \texttt{dba} \in L, w(4,7) = \texttt{dcdc} \in L$$

Quick Search Type Pruning in Pattern Matching

The last char. does not match

The last char. matches



Quick Search Type Pruning in Pattern Matching

The last char. does not match

Quick Search

1. Check the last char.
2. Skip based on the subsequent char.

The last char. matches

KMP

1. Matching trial. left → right
2. Skip based on the # of char. we checked.

```
ABCDABB

   DADB
```

The discussion is independent of the position of the char. 🙂

Quick Search Type Pruning in Pattern Matching

The last char. does not match

Quick Search

1. Check the last char.
2. Skip based on the subsequent char.

ABCDABB

✗

✗DADB

The last char. matches

KMP

1. Matching trial. left → right
2. Skip based on the # of char. we checked.

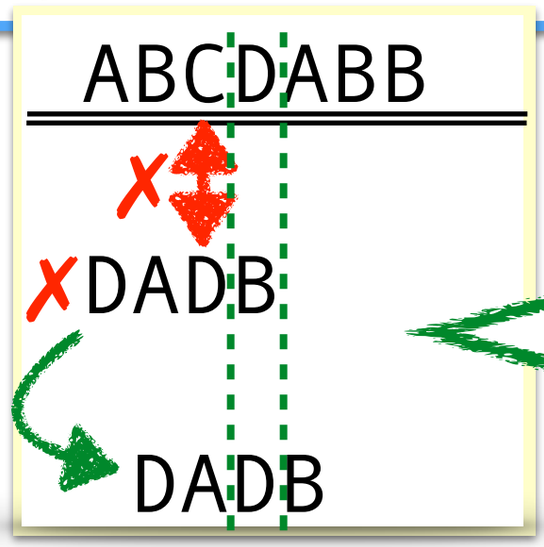
The discussion is independent of the position of the char. 🙂

Quick Search Type Pruning in Pattern Matching

The last char. does not match

Quick Search

1. Check the last char.
2. Skip based on the subsequent char.



The last char. matches

KMP

1. Matching trial. left → right
2. Skip based on the # of char. we checked.

The discussion is independent of the position of the char. 🙂

Quick Search Type Pruning in Pattern Matching

- $L = L(\{\mathtt{ab} \mid \mathtt{cd}\}\mathtt{c}^{+}\mathtt{d})$
- L': the prefixes of L as long as the shortest elem. of L

$$L(\mathcal{A}) = \left\{ \begin{array}{ll} \mathtt{abcd}, & \mathtt{cdcd} \\ \mathtt{abcc}d, & \mathtt{cdcc}d \\ \mathtt{abcc}cd, & \mathtt{cdcc}cd \\ \vdots & \vdots \end{array} \right\} \rightsquigarrow \left\{ \begin{array}{l} \mathtt{abcd}, \mathtt{cdcd} \\ \mathtt{abcc}, \mathtt{cdcc} \end{array} \right\} \Sigma^*$$

Over approximation!

The position of the “last” char. :
the length of the shortest elem. of L

Quick Search Type Pruning in Pattern Matching

The “last” char. does not match

The “last” char. matches

Quick Search

KMP

1. Check $(min|L|)$-th char.
2. Skip based on the subsequent char.

1. Matching trial. left → right
2. Skip based on the # of char. we checked.

All elements of L' have the same length!! 🙂

Quick Search Type Pruning in Pattern Matching

The “last” char. does not match

The “last” char. matches

Quick Search

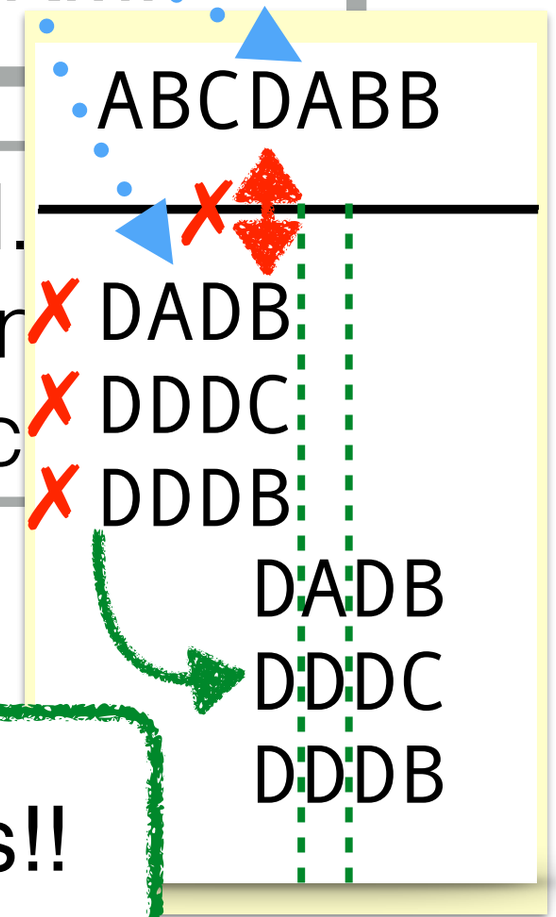
KMP

1. Check $(min|L|)$-th char.
2. Skip based on the subsequent char.

The rightmost occurrence of A in the pattern!

All elements of L' have the same length!! 🙂

Skip 3 trials!!



KMP Type Skipping in Pattern Matching

The “last” char. does not match

The “last” char. matches

Quick Search

KMP

1. Check ($\underline{min|L|}$)-th char.
2. Skip based on the subsequent char.

1. Matching trial. left → right
2. Skip based on the # of char. we checked.

1. No problem! 🙂
2. # of char. in L in unbounded… 😟

KMP Type Skipping in Pattern Matching

String Matching

For a position n ($\leq |pat|$),

$w(i, j) = pat(1, n)$

$pat(1, 2) = w(i, j)$

	D	A
✗	D	A
✔		D A

A shifted pattern pat

Pattern Matching

For a state $s \in S$, $w(i, j) \in L_s$

L_s : the words leading to s

Infinite words !

Including $w(i, j)$

The words leading to s	$= L_s$
✗ *	L
✗ * *	L
✓ * * *	L

A shifted pattern L

KMP Type Skipping in Pattern Matching

- $L = L(low\ (high \cdot low)^{+})$
- L': the prefixes of L as long as the shortest elem. of L

$$L(\mathcal{A}) = \left\{ \begin{array}{ll} \texttt{abcd}, & \texttt{cdcd} \\ \texttt{abcc}d, & \texttt{cdcc}d \\ \texttt{abcc}cd, & \texttt{cdcc}cd \\ & \vdots \end{array} \right\} \rightsquigarrow \left\{ \begin{array}{l} \texttt{abcd}, \texttt{cdcd} \\ \texttt{abcc}, \texttt{cdcc} \end{array} \right\} \Sigma^*$$

Over approximation!

Such suffix is unnecessary when calculating skip value.

KMP Type Skipping in Pattern Matching

String Matching

For a position n ($\leq |pat|$),
$w(i, j) = pat(1, n)$

$pat(1, 2) = w(i, j)$

	D A⋮
✗	D⋮ A
✔	⋮D A

A shifted pattern pat

Pattern Matching

- $L = L(low\ (high \cdot low)^{+})$

Including a prefix of $w(i, j)$

	{low high}⋮ = L's
✗	low⋮ high low
✔	⋮low high low

Some prefixes of L

KMP Type Skipping in Pattern Matching

The “last” char. does not match

The “last” char. matches

Quick Search

KMP

1. Check $(min|L|)$-th char.
2. Skip based on the subsequent char.

1. Matching trial. left → right
2. Skip based on the state of the NFA we checked.

1. No problem! 🙂
2. State space of NFA is finite ! 🙂

Quick Search Type Skipping in Timed Pattern Matching

The “last” char. does not match

The “last” char. matches

Quick Search

KMP

1. Check ($min|L|$)-th char.
2. Skip based on the subsequent char.

1. Matching trial: left → right
2. Shift

We only focus on events!!