Semi-global and local alignment and gap penalties

UNIVERSITY OF MARYLAND

Maximization vs. Minimization

Edit distance:

$$\mathrm{OPT}(i,j) = \min \begin{cases} \mathrm{cost}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) & \textbf{match } x_i, y_j \\ c_{\mathrm{gap}} + \mathrm{OPT}(i-1, j) & x_i \textbf{ is unmatched} \\ c_{\mathrm{gap}} + \mathrm{OPT}(i, j-1) & y_j \textbf{ is unmatched} \end{cases}$$

Sequence Similarity: replace the *min* with a *max* — find the highest-scoring alignment. Gap costs and bad matches usually get a negative "score".

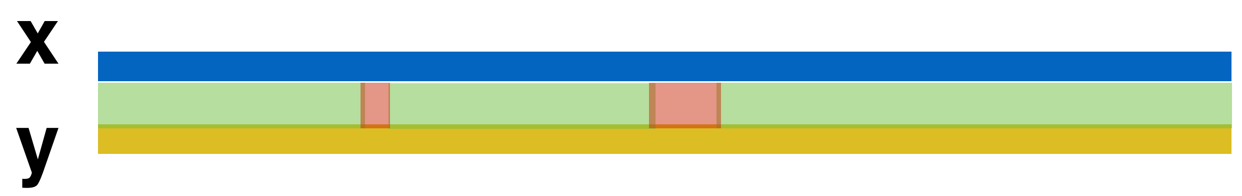
$$\mathrm{OPT}(i,j) = \max \begin{cases} \mathrm{score}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ s_{\mathrm{gap}} + \mathrm{OPT}(i-1, j) \\ s_{\mathrm{gap}} + \mathrm{OPT}(i, j-1) \end{cases}$$

gap penalty → gap score (probably negative)
match cost → match score

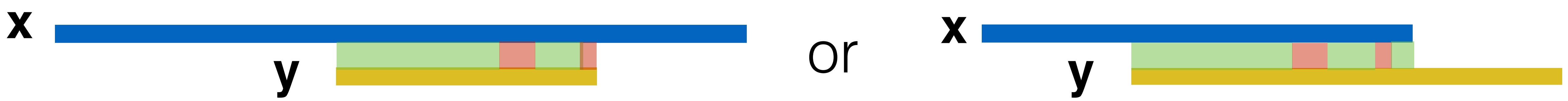
*

Alignment Categories

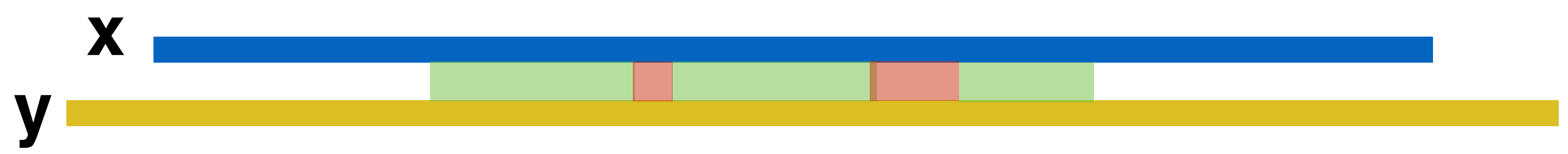
Global: Require an end-to-end alignment of **x**,**y**



Semi-global (glocal): Gaps at the beginning or end of **x** or **y** are free — useful when one string is significantly shorter than the other or for finding overlaps between strings

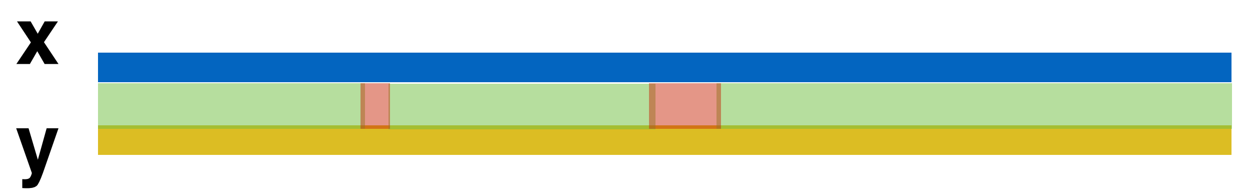


Local: Find the highest scoring alignment between **x'** a substring of **x** and **y'** a substring of **y** — useful for finding similar regions in strings that may not be globally similar

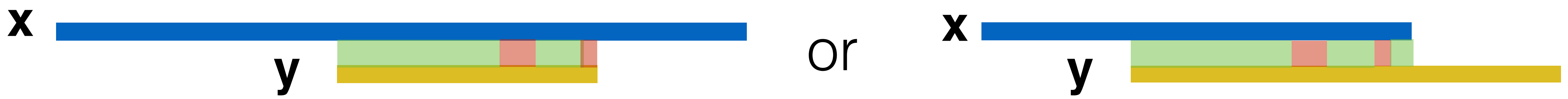


Alignment Categories Motivation

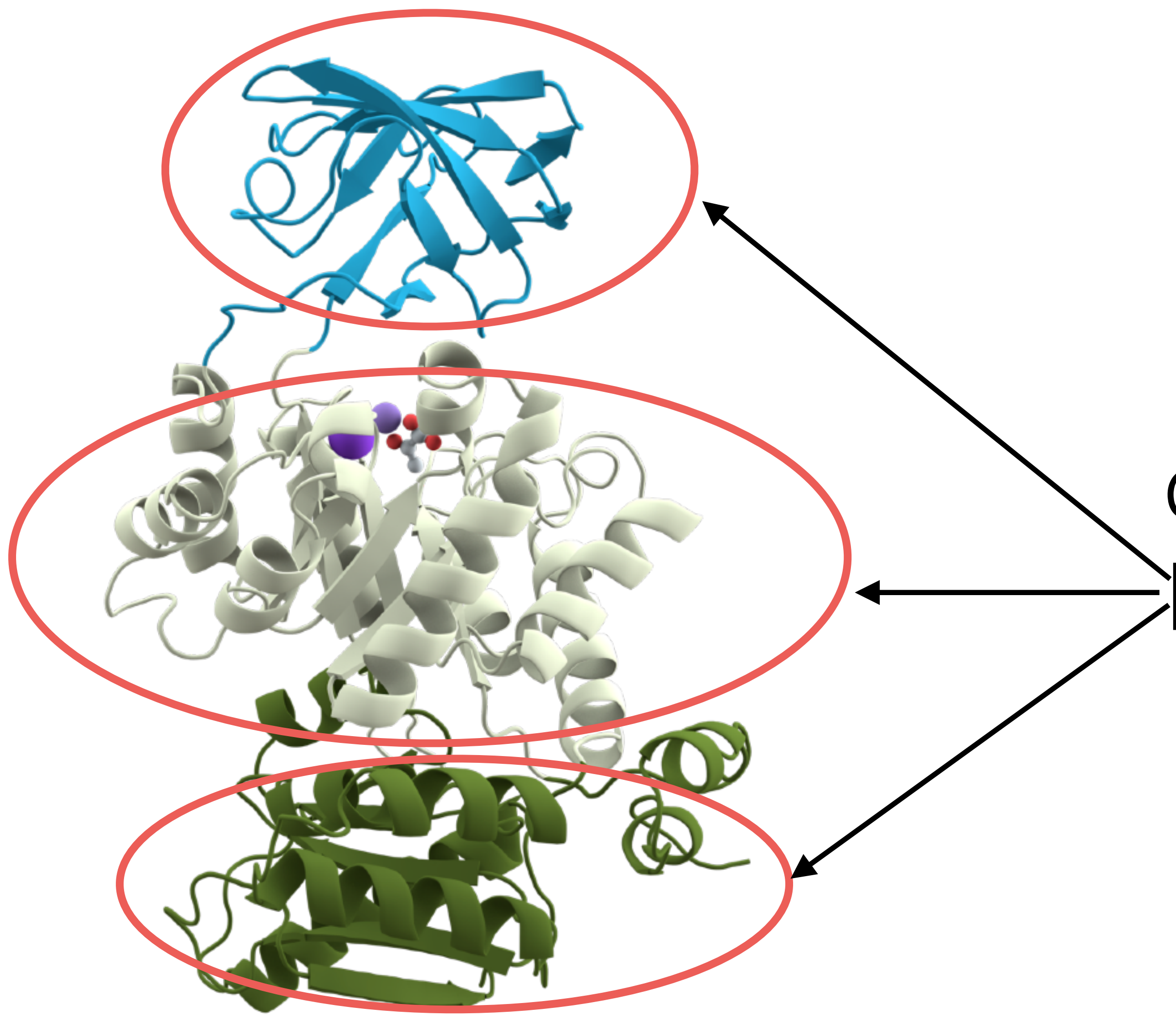
Global: **x** and **y** are similar proteins from closely-related species



Semi-global (glocal): **x** and **y** are sequencing reads we are trying to assemble. We want to find reads where the right end (suffix) of one matches the left end (prefix) of another.

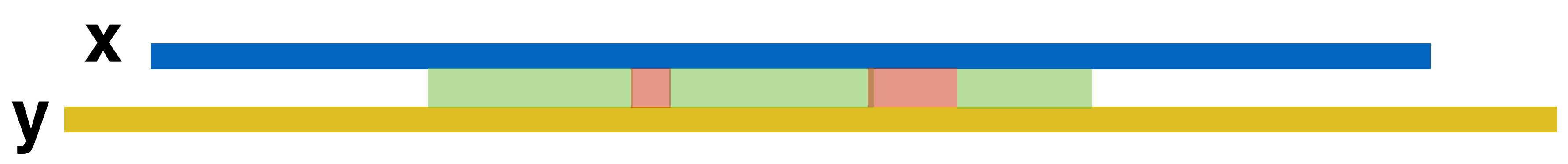


Alignment Categories Motivation



It's possible and somewhat common for specific domains to be conserved, but not the entire protein sequence / structure.

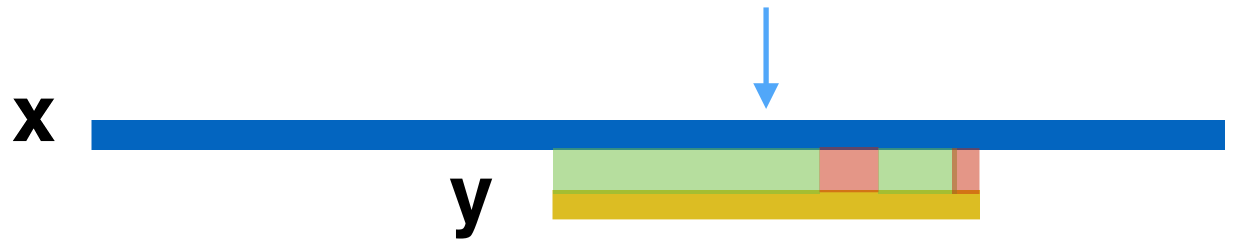
Local: **x** and **y** are similar proteins from potentially distantly related species. We don't expect the entire protein to be conserved, but certain "domains" should be.

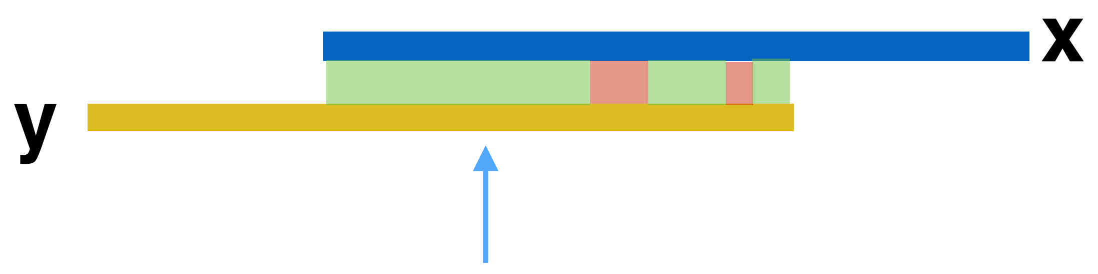


Semi-global Alignment Example

Semi-global (glocal): Gaps at the beginning or end of **x** or **y** are free. Useful when one one string is significantly shorter than the other or we want to find an overlap between the suffix of one string and a prefix of the other

sometimes called "cost-free-ends" or "fitting" alignment





sometimes called "overlap" alignment

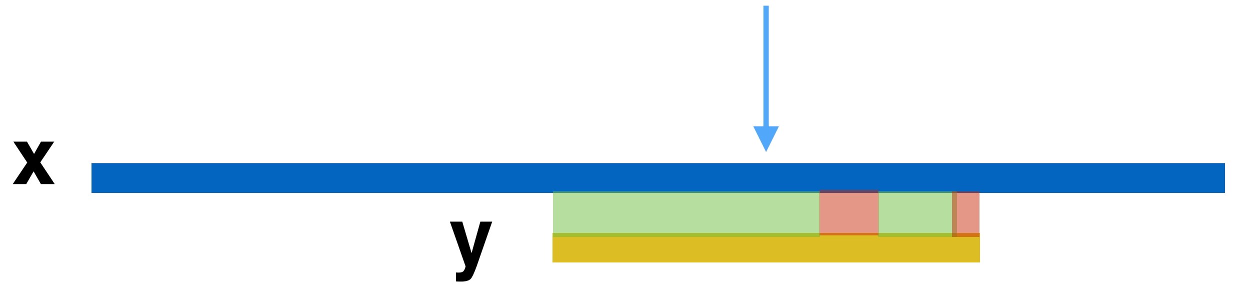
Motivation:

Useful for finding similarities that global alignments wouldn't. Also, can view "read mapping" as a variant of the semi-global alignment problem. Want to align entire read but it's a tiny fraction of the genome. *Note*: won't use semi-global alignment with the full genome for read mapping in practice.

Semi-global Alignment Example

Semi-global (glocal): Gaps at the beginning or end of **x** or **y** are free — one useful case is when one string is significantly shorter than the other

sometimes called "cost-free-ends" or "fitting" alignment



We'll discuss the "fitting" variant for in the next few slides for simplicity, but the same basic idea applies for the "overlap" variant as well.

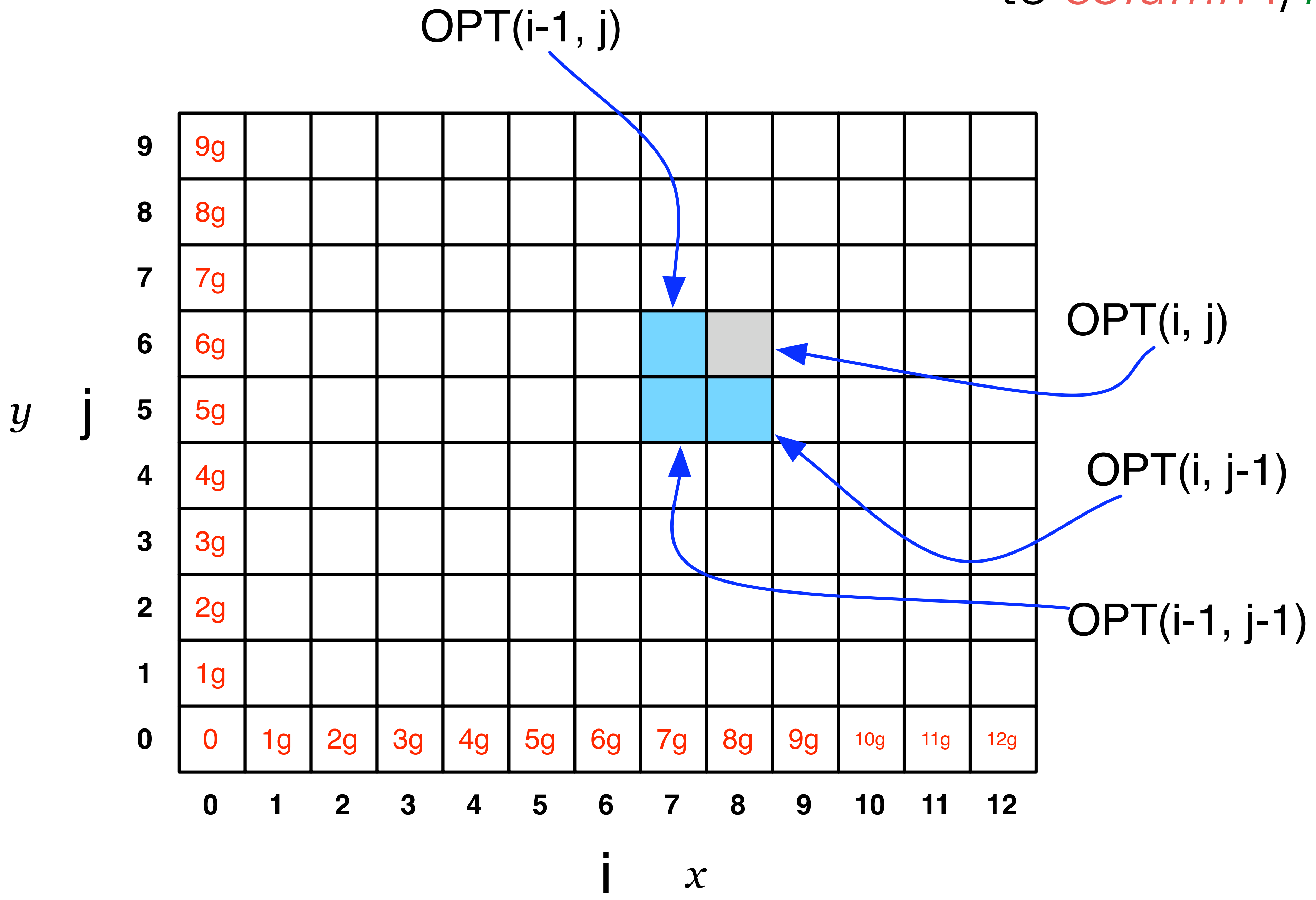
Recall: Global Alignment Matrix

OPT(*i*,*j*) contains the score for the best alignment between:

the first *i* characters of string *x* [**prefix** *i* of *x*]

the first *j* character of string *y* [**prefix** *j* of *y*]

NOTE: observe the non-standard notation here; OPT(**i**,**j**) is referring to *column* i, *row* j of the matrix.



*

How to do semi-global alignment?

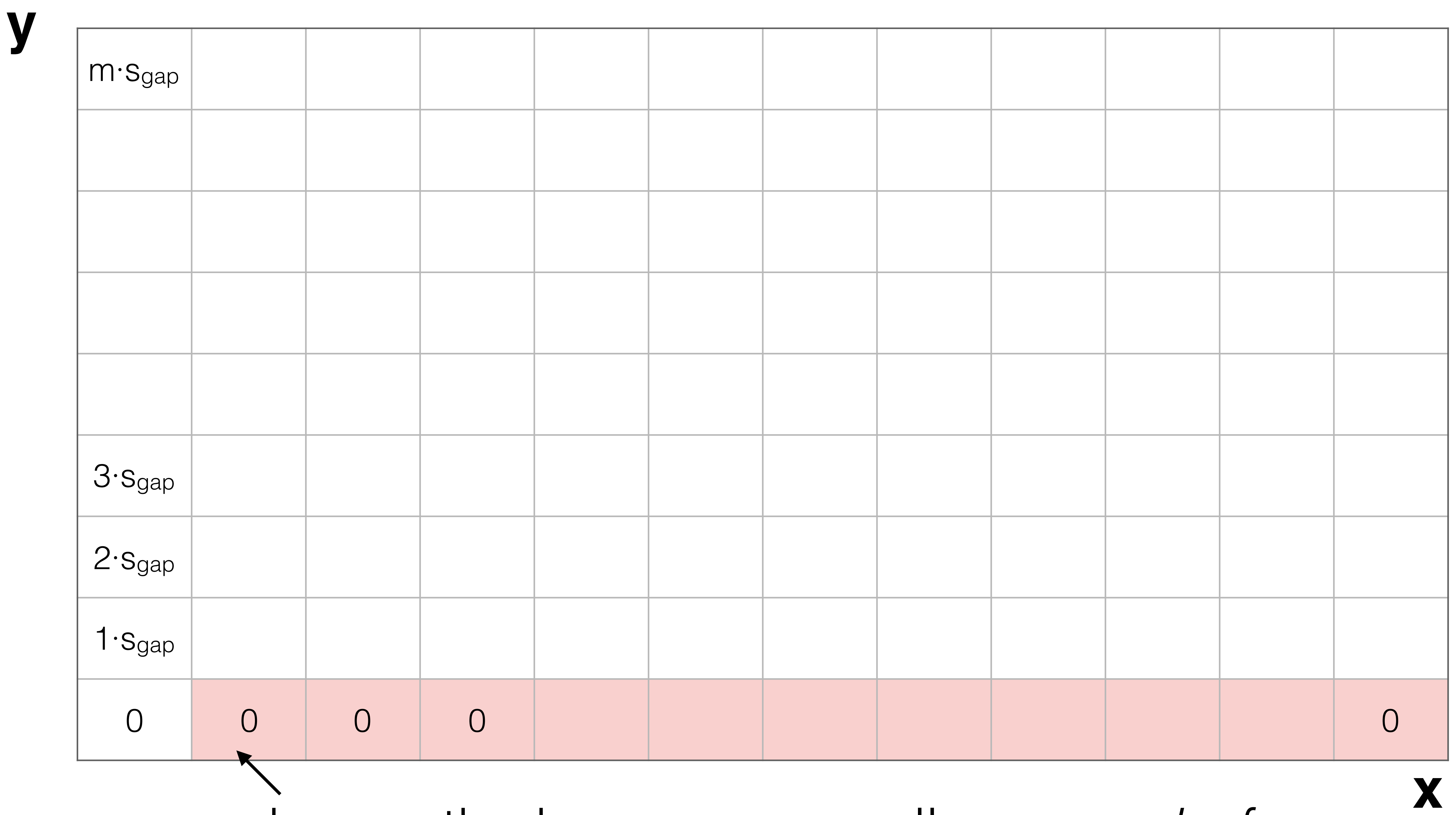
y

$m \cdot s_{gap}$											
$3 \cdot s_{gap}$											
$2 \cdot s_{gap}$											
$1 \cdot s_{gap}$											
0	$1 \cdot s_{gap}$	$2 \cdot s_{gap}$	$3 \cdot s_{gap}$								$n \cdot s_{gap}$

x

Start with the original global alignment matrix

How to do semi-global alignment?



change the base case — allow gaps *before* y

How to do semi-global alignment?

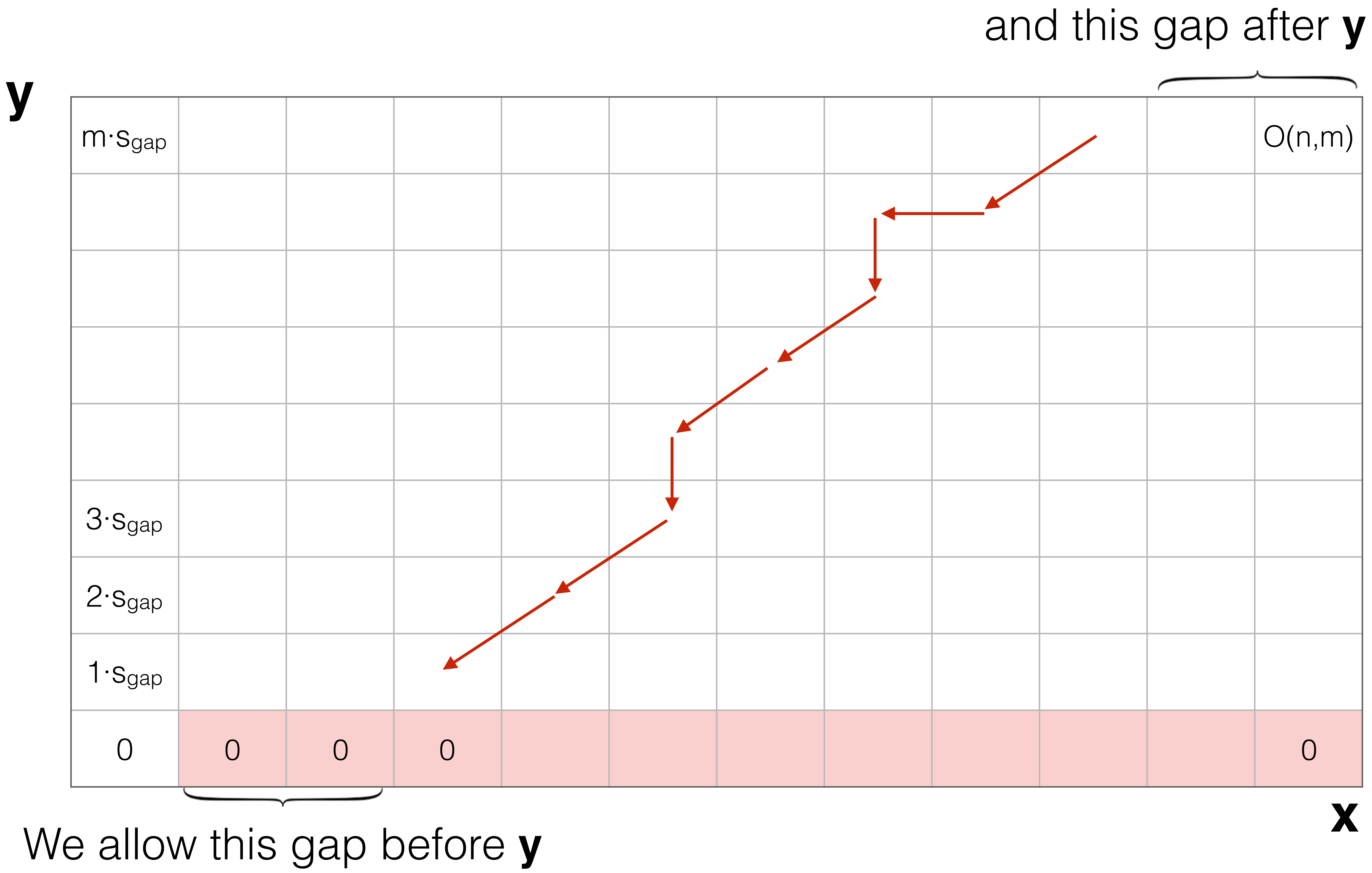
y

$m \cdot s_{gap}$											$O(n,m)$
$3 \cdot s_{gap}$											
$2 \cdot s_{gap}$											
$1 \cdot s_{gap}$											
0	0	0	0								0

x

start traceback at $\max_{0<i\leq n} OPT(i,m)$ — this allows gaps after **y**; why?

Semi-global alignment example



Semi-global Alignment

What is the same and different between the "global" and semi-global ("fitting") alignment problems?

*assuming |y| < |x| and we are "fitting" y into x

Global

$$\mathrm{OPT}(i,j) = \max \begin{cases} \mathrm{score}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ \mathrm{s_{gap}} + \mathrm{OPT}(i-1, j) \\ \mathrm{s_{gap}} + \mathrm{OPT}(i, j-1) \end{cases}$$

Base case: OPT(i,0) = i x s_{gap}

Traceback starts at OPT(n,m)

Semi-global ("fitting")

$$\mathrm{OPT}(i,j) = \max \begin{cases} \mathrm{score}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ \mathrm{s_{gap}} + \mathrm{OPT}(i-1, j) \\ \mathrm{s_{gap}} + \mathrm{OPT}(i, j-1) \end{cases}$$

Base case: OPT(i,0) = 0

Traceback starts at $\max_{0<j\leq n}$ OPT(j,m)

Semi-global Alignment

The recurrence remains the *same*, we only change the base case of the recurrence and the origin of the backtrack

1) Ignore gaps before x $\longrightarrow$ change base case; $OPT(0,j) = 0$

2) Ignore gaps after x $\longrightarrow$ change traceback; start from $\max_{0<j\leq m} OPT(n,j)$

3) Ignore gaps before y $\longrightarrow$ change base case; $OPT(i,0) = 0$

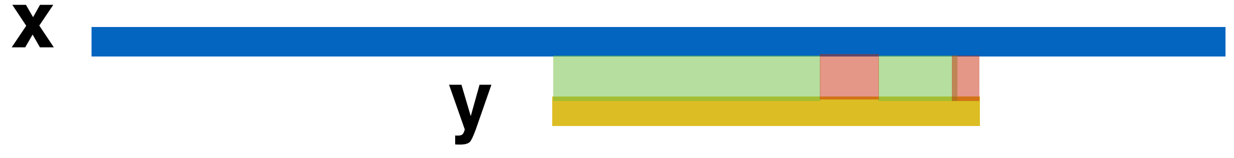
4) Ignore gaps after y $\longrightarrow$ change traceback; start from $\max_{0<i\leq n} OPT(i,m)$

Semi-global Alignment

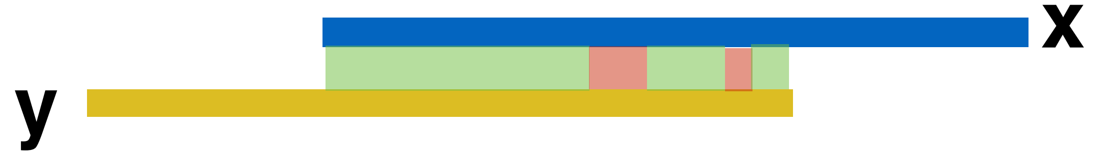
1) Ignore gaps before x
2) Ignore gaps after x
3) Ignore gaps before y
4) Ignore gaps after y

Types of semi-global alignments

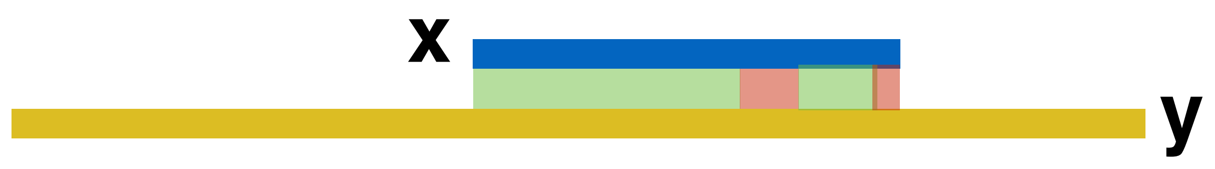
use mods 3&4



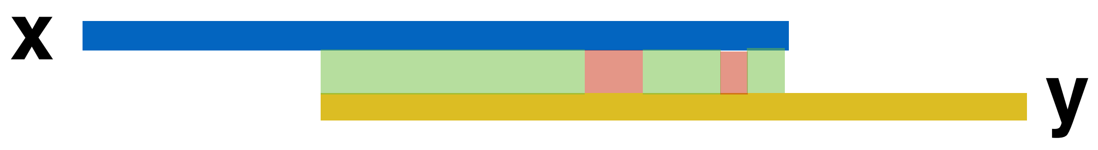
use mods 1&4



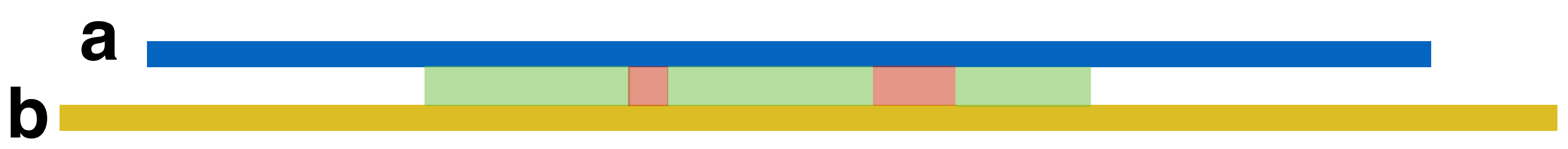
use mods 1&2



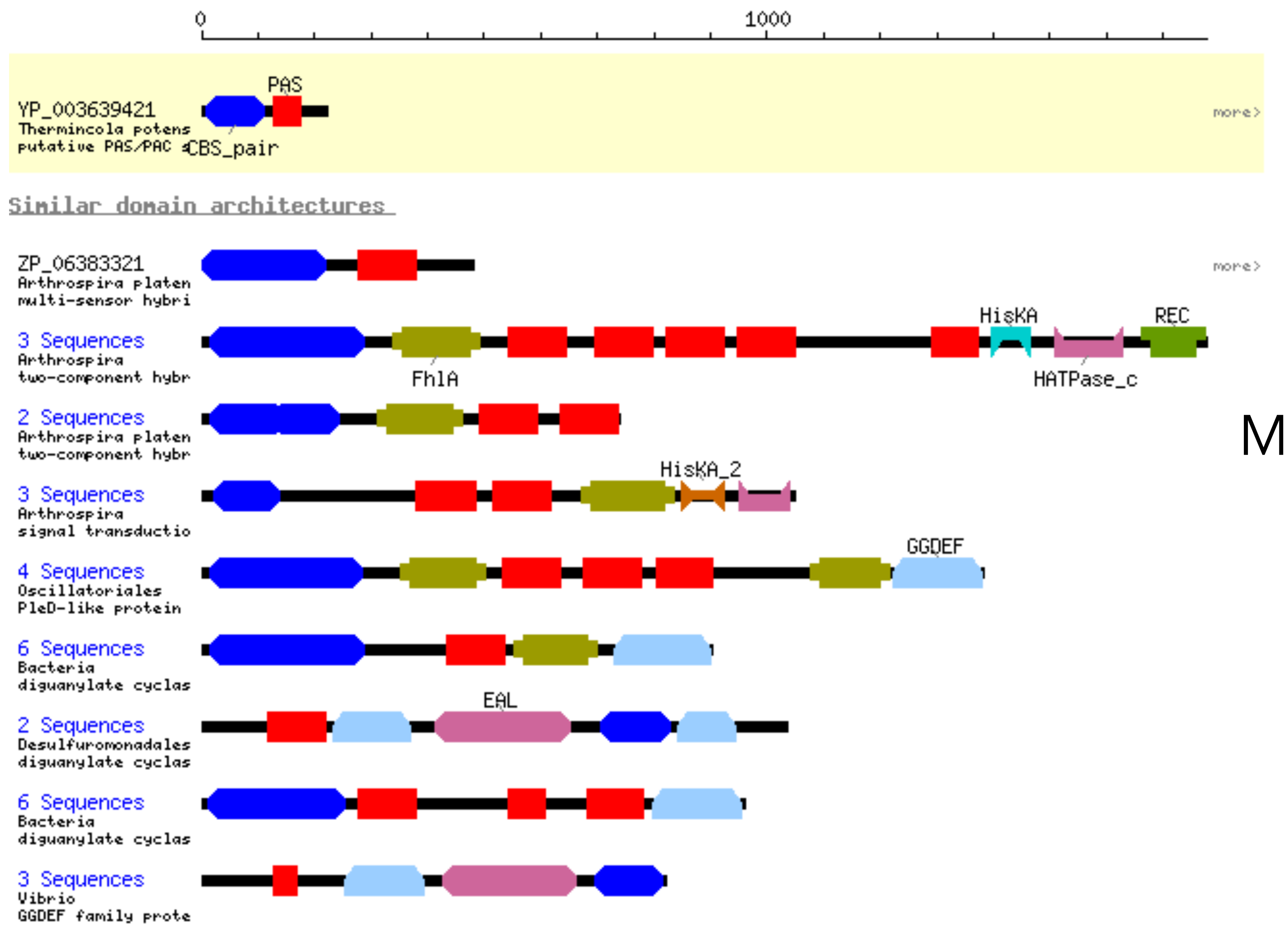
use mods 2&3



Local Alignment



Local alignment between a and b: Best alignment between a subsequence of a and **a** subsequence of **b**.



Motivation:

Many genes are composed of *domains*, which are subsequences that perform a particular function.

*

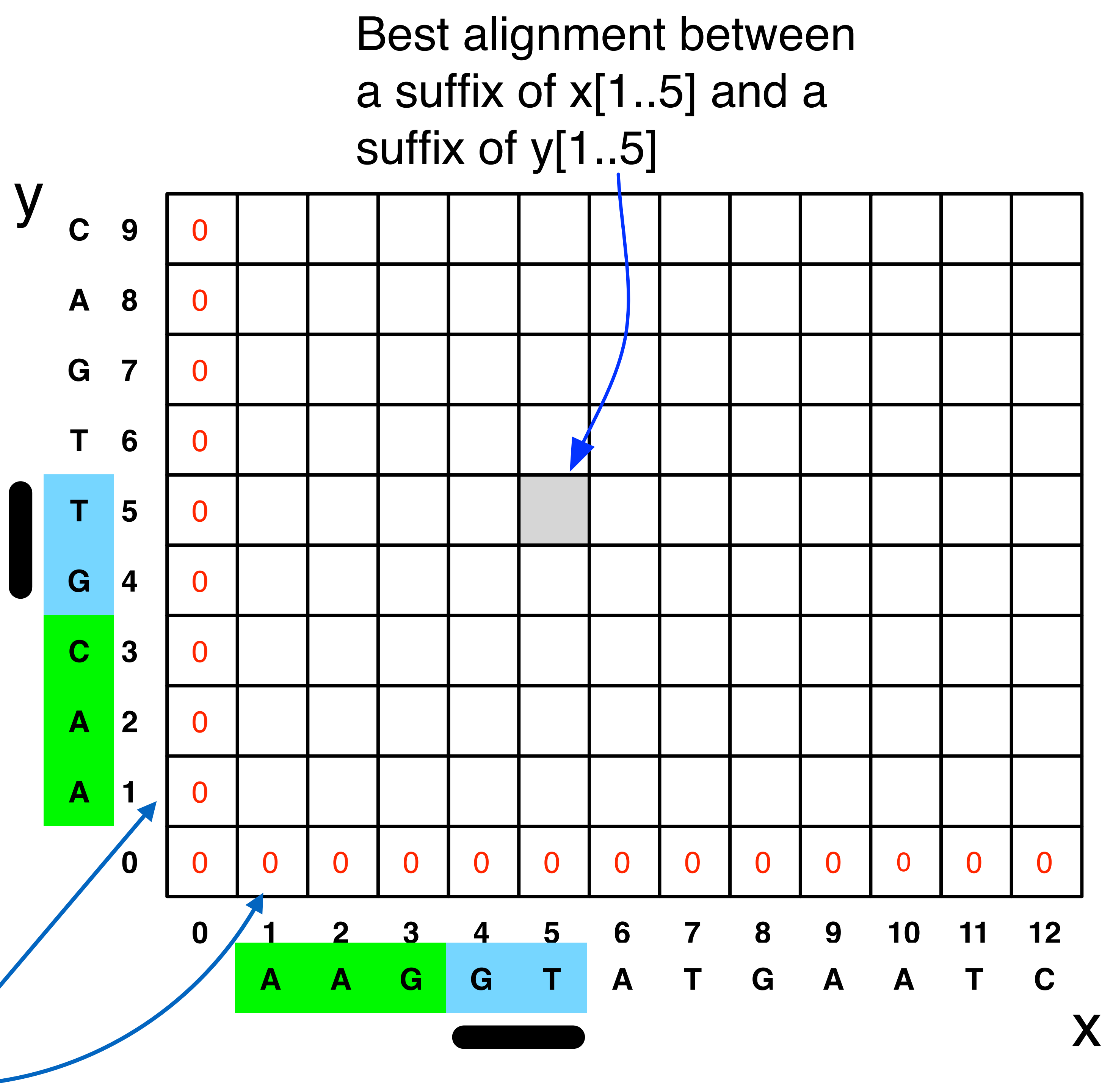
Local Alignment

New meaning of entry of matrix entry:

OPT(i, j) = best score between:

some suffix of *x*[1…*i*]

some suffix of *y*[1…*j*]



Same base-case trick we used in semi-global alignment

*

Local Alignment

New meaning of entry of matrix entry:

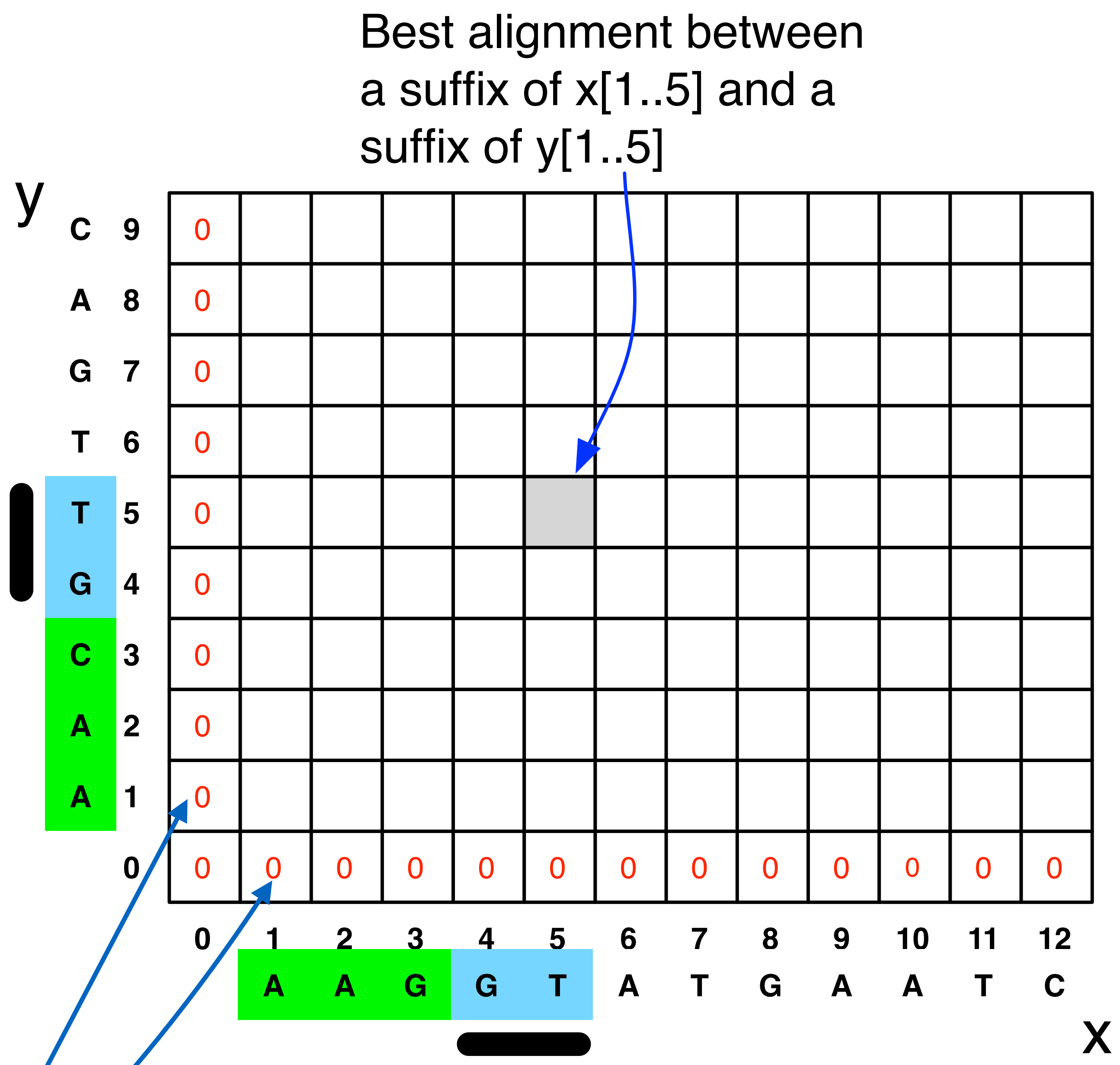
OPT(i, j) = best score between:

some suffix of *x*[1…*i*]

some suffix of *y*[1…*j*]

What else do we need to change to allow local alignments?

Hint: The empty alignment is always a valid local alignment!



Same base-case
trick we used in semi-global alignment

*

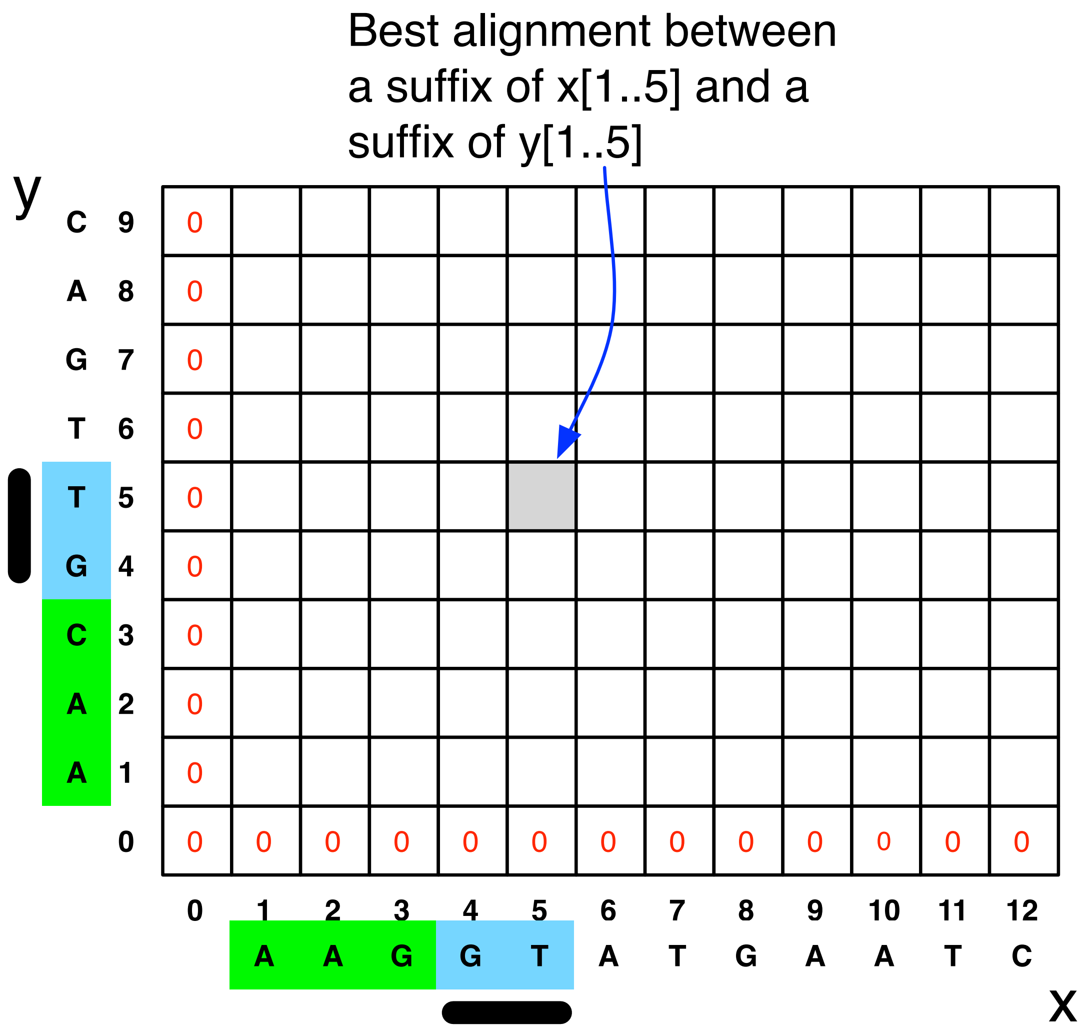
How do we fill in the local alignment matrix?

$$\mathrm{OPT}(i,j) = \max \begin{cases} \mathrm{score}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) & \text{(1)} \\ s_{\mathrm{gap}} + \mathrm{OPT}(i-1, j) & \text{(2)} \\ s_{\mathrm{gap}} + \mathrm{OPT}(i, j-1) & \text{(3)} \\ 0 \end{cases}$$

(1), (2), and (3): same cases as before: match x and y, gap in y, gap in x

New case: 0 allows you to say the best alignment between a suffix of *x* and a suffix of *y* is the empty alignment.

Lets us "start over"



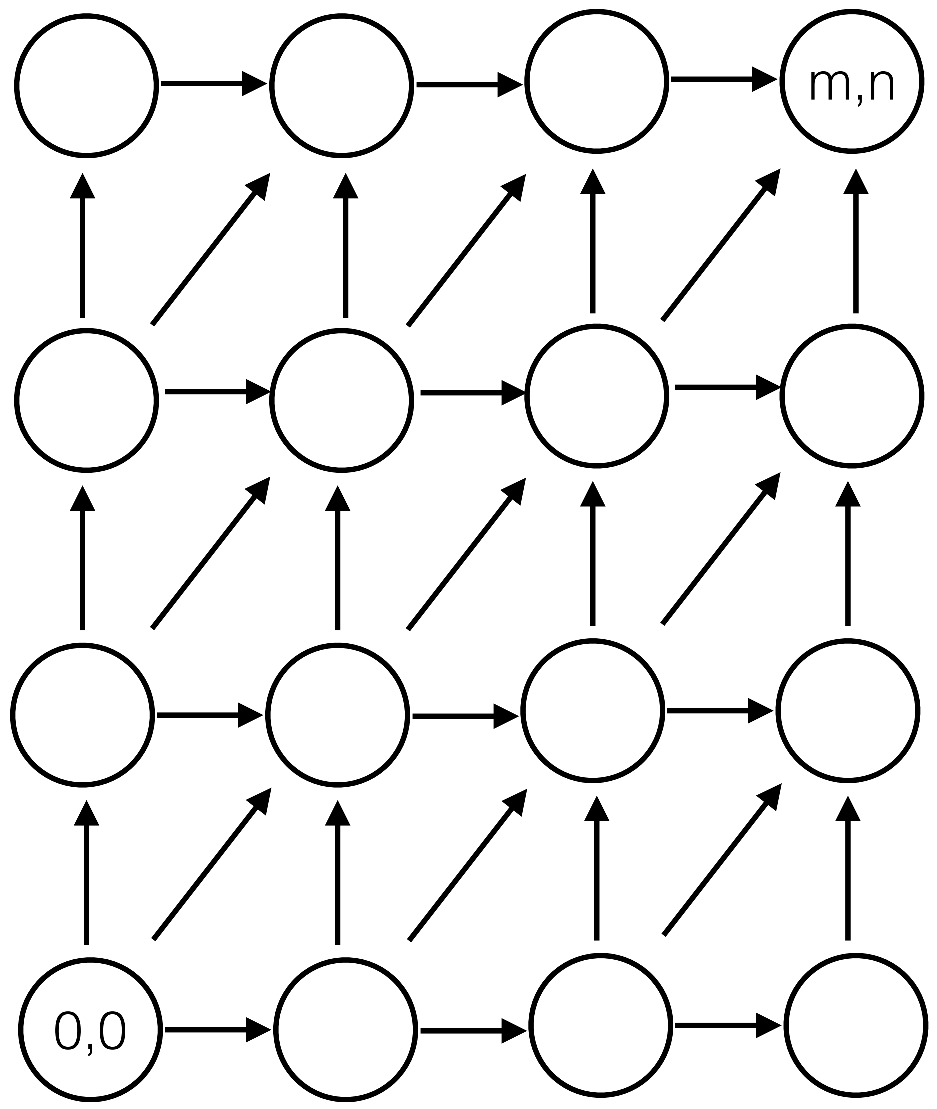
*

Local Alignment

- Initialize first row and first column to be 0.
- The score of the best local alignment is the largest value in the entire array.
- To find the actual local alignment:
    - start at an entry with the maximum score
    - traceback as usual
    - stop when we reach an entry with a score of 0

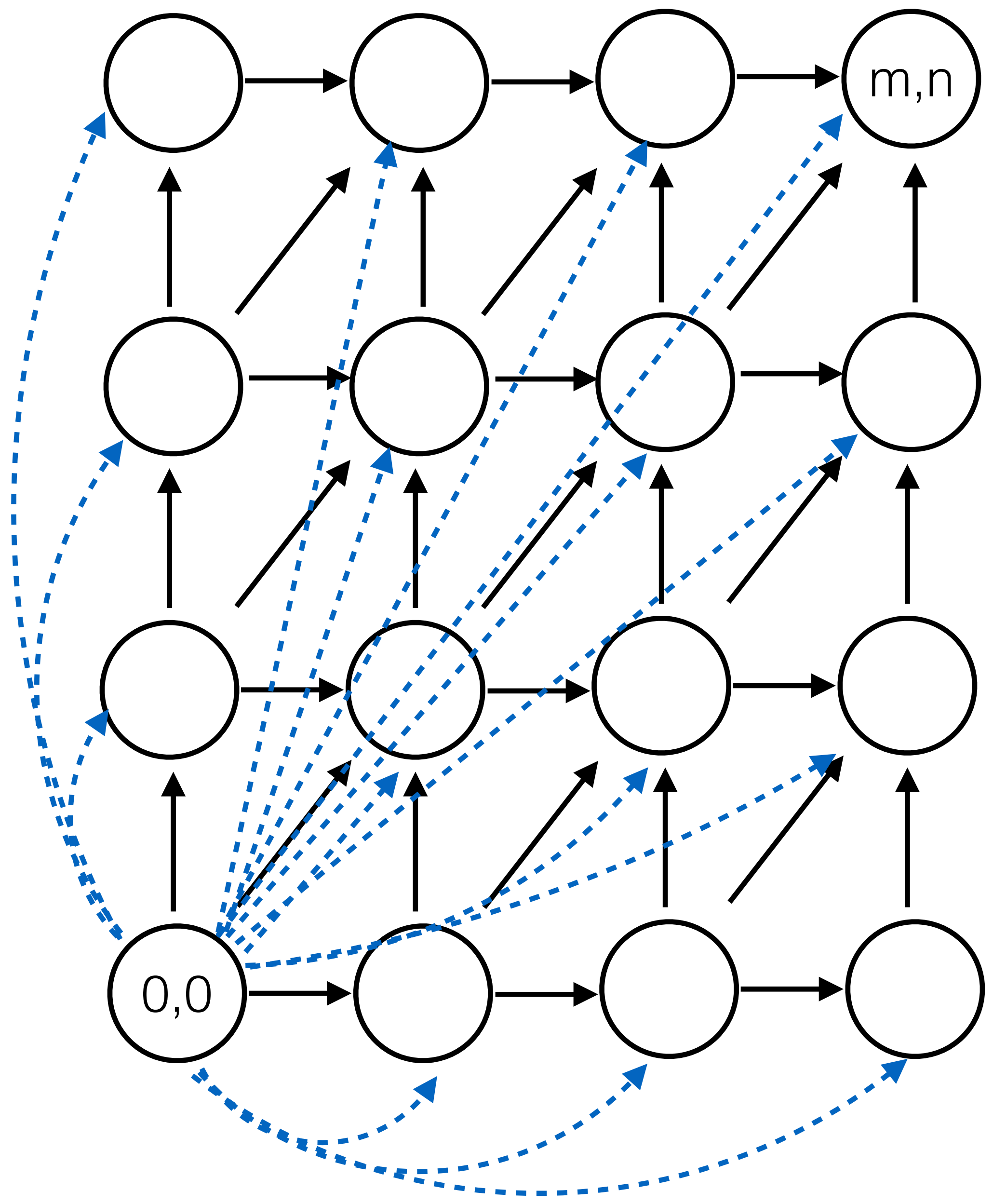
*

Local Alignment in the DAG framework



Local Alignment in the DAG framework

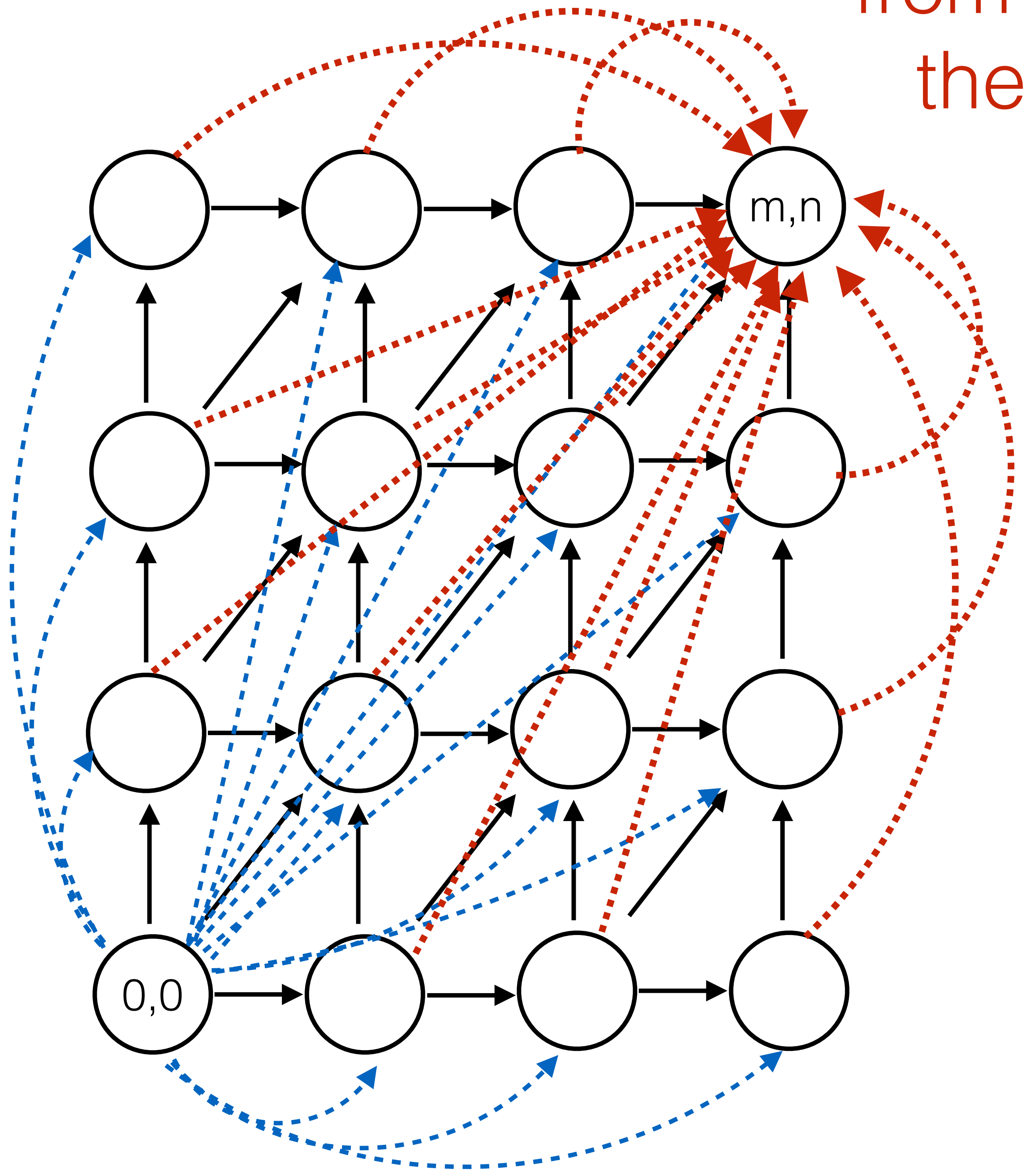
Add 0 score edge from the source to every node



Local Alignment in the DAG framework

Add 0 score edge from the source to every vertex

Add 0 score edge from every vertex to the target vertex



Local Alignment Example #1

`local align("AGCGTAG", "CTCGTC")`

	*	A	G	C	G	T	A	G
*	0	0	0	0	0	0	0	0
C	0	0	0	10	3	0	0	0
T	0	0	0	3	5	13	6	0
C	0	0	0	↖ 10	3	6	8	1
G	0	0	10	3	↖ 20	13	6	18
T	0	0	3	5	13	↖ **30**	23	16
C	0	0	0	13	6	23	25	18

```
Score(match) = 10
Score(mismatch) = -5
Score(gap) = -7
```

Note: this table written top-to-bottom instead of bottom-to-top

*

Local Alignment Example #2

`local align(“bestoftimes”, “soften”)`

	*	b	e	s	t	o	f	t	i	m	e	s
*	0	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	↖10	←3	0	0	0	0	0	0	10
o	0	0	0	3	5	↖13	6	0	0	0	0	3
f	0	0	0	0	0	6	↖23	16	9	2	0	0
t	0	0	0	0	10	3	16	↖33	26	19	12	5
e	0	0	10	3	3	5	9	26	28	21	29	22
n	0	0	3	5	0	0	2	19	21	23	22	24

```
Score(match) = 10
Score(mismatch) = -5
Score(gap) = -7
```

Note: this table written top-to-bottom instead of bottom-to-top

More Local Alignment Examples

```
Score(match) = 10
Score(mismatch) = -5
Score(gap) = -7
```

local align("catdogfish", "dog")

	*	c	a	t	d	o	g	f	i	s	h
*	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	↖10	3	0	0	0	0	0
o	0	0	0	0	3	↖20	13	6	0	0	0
g	0	0	0	0	0	13	↖**30**	23	16	9	2

local align("mississippi", "issp")

	*	m	i	s	s	i	s	s	i	p	p	i
*	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	10	3	0	↖10	3	0	10	3	0	10
s	0	0	3	20	13	6	↖20	13	6	5	0	3
s	0	0	0	13	30	23	16	↖30	←23	16	9	2
p	0	0	0	6	23	25	18	23	25	↖**33**	26	19

local align("aaaa", "aa")

	*	a	a	a	a
*	0	0	0	0	0
a	0	10	10	↖10	10
a	0	10	**20**	**20**	↖**20**

*

Local / Global Recap

- Alignment score sometimes called the “edit distance” between two strings.
- Edit distance is sometimes called Levenshtein distance.

- Algorithm for local alignment is sometimes called “Smith-Waterman”
- Algorithm for global alignment is sometimes called “Needleman-Wunsch”

- Same basic algorithm, however.
- Underlies BLAST

*

General Gap Penalties

```
AAAGAATTCA              AAAGAATTCA
               vs.
A-A-A-T-CA              AAA----TCA
```

These have the same score, but the second one is often more plausible.

A single insertion of "GAAT" into the first string could change it into the second — Biologically, this is much more likely as **x** could be transformed into **y** in "one fell swoop".

- Currently, the score of a run of k gaps is $s_{gap} \times k$
- It might be more realistic to support general gap penalty, so that the score of a run of k gaps is $|\mathbf{gscore}(k)| < |(s_{gap} \times k)|$.
- Then, the optimization will prefer to group gaps together.

*

General Gap Penalties — The Problem

```
AAAGAATTCA          AAAGAATTCA
A-A-A-T-CA    vs.   AAA----TCA
```

Previous DP no longer works with general gap penalties.

Why?

*

General Gap Penalties — The Problem

```
AAAGAATTCA         AAAGAATTCA
A-A-A-T-CA   vs.   AAA----TCA
```

The score of the *last character* depends on *details* of the previous alignment:

```
AAAGAAC          AAAGAATC
AAA----   vs.    AAA-----
```

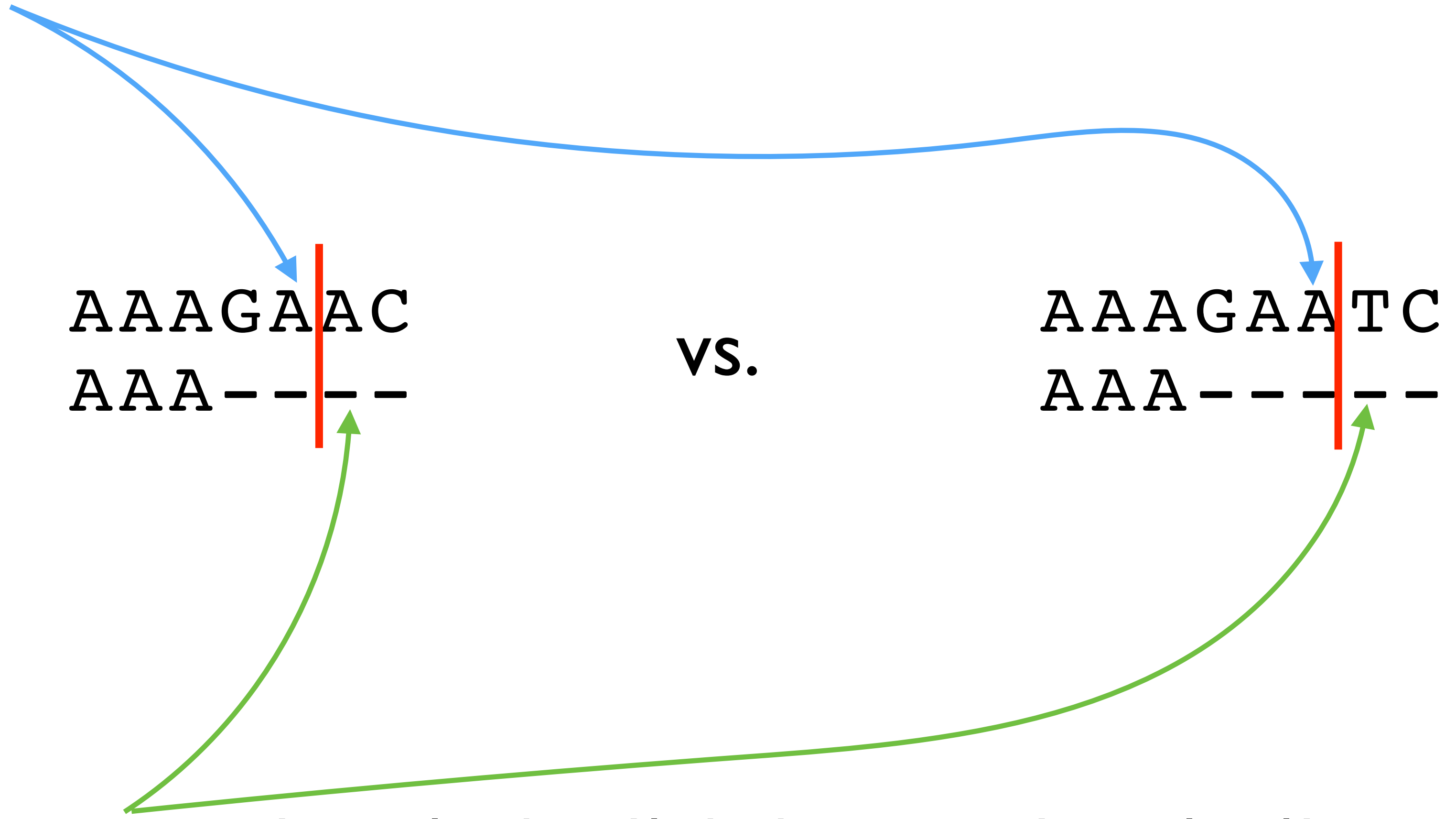
We need to “know” how long a final run of gaps is in order to give a score to the last subproblem.

*

General Gap Penalties — The Problem

The score of the *last character* depends on *details* of the previous alignment:

Knowing the optimal alignment at the substring ending here.



Doesn't let us simply build the optimal alignment ending here.

*

Three Matrices

We now keep 3 different matrices:

M(i,j) = score of best alignment of x[1..i] and y[1..j] ending with a character-character **match or mismatch**.

X(i,j) = score of best alignment of x[1..i] and y[1..j] ending with a **gap in X.**

Y(i,j) = score of best alignment of x[1..i] and y[1..j] ending with a **gap in Y.**

$$\mathrm{M}(i,j) = \mathrm{score}(x_i, y_j) + \max \begin{cases} \mathrm{M}(i-1, j-1) \\ \mathrm{X}(i-1, j-1) \\ \mathrm{Y}(i-1, j-1) \end{cases}$$

$$\mathrm{X}(i,j) = \max \begin{cases} \mathrm{M}(i, j-k) + \mathrm{gscore}(k) & \text{for } 1 \le k \le j \\ \mathrm{Y}(i, j-k) + \mathrm{gscore}(k) & \text{for } 1 \le k \le j \end{cases}$$

$$\mathrm{Y}(i,j) = \max \begin{cases} \mathrm{M}(i-k, j) + \mathrm{gscore}(k) & \text{for } 1 \le k \le i \\ \mathrm{X}(i-k, j) + \mathrm{gscore}(k) & \text{for } 1 \le k \le i \end{cases}$$

*

The M Matrix

We now keep 3 different matrices:

M(i,j) = score of best alignment of x[1..i] and y[1..j] ending with a character-character **match or mismatch**.

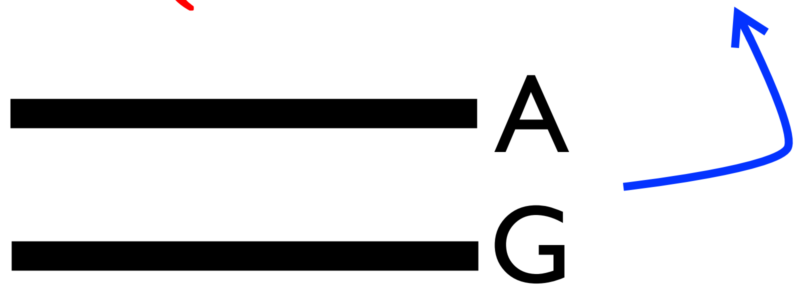
X(i,j) = score of best alignment of x[1..i] and y[1..j] ending with a **gap in X**.

Y(i,j) = score of best alignment of x[1..i] and y[1..j] ending with a **gap in Y**.

By definition, alignment ends in a match/mismatch.

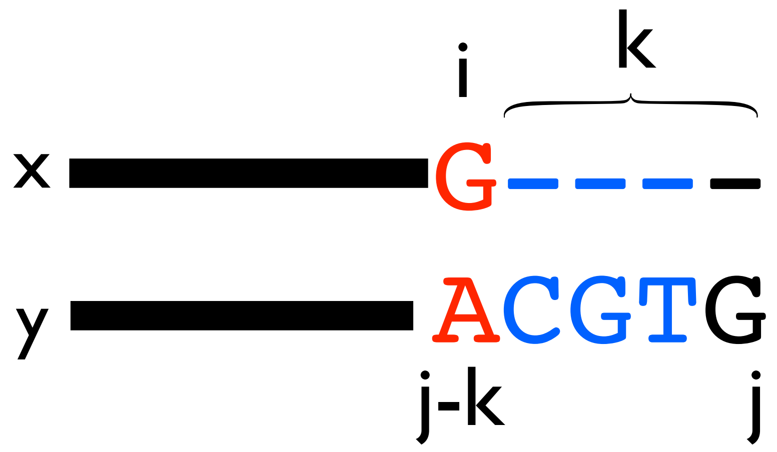
$$\mathrm{M}(i,j) = \mathrm{score}(x_i, y_j) + \max \begin{cases} \mathrm{M}(i-1, j-1) \\ \mathrm{X}(i-1, j-1) \\ \mathrm{Y}(i-1, j-1) \end{cases}$$

Any kind of alignment is allowed before the match/mismatch.



*

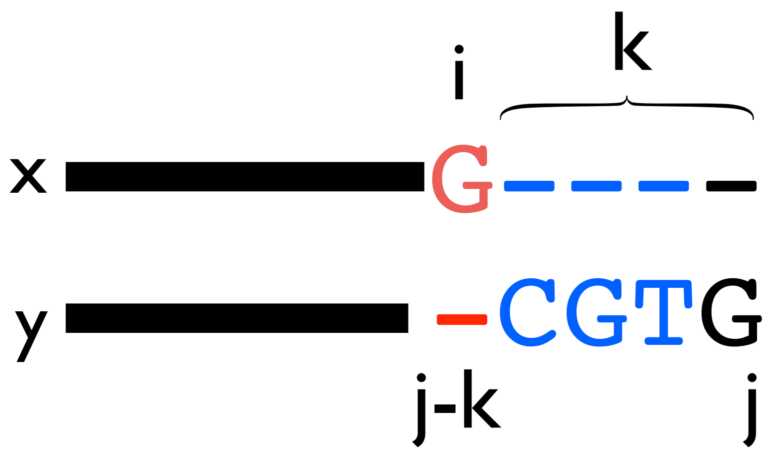
The X (and Y) matrices



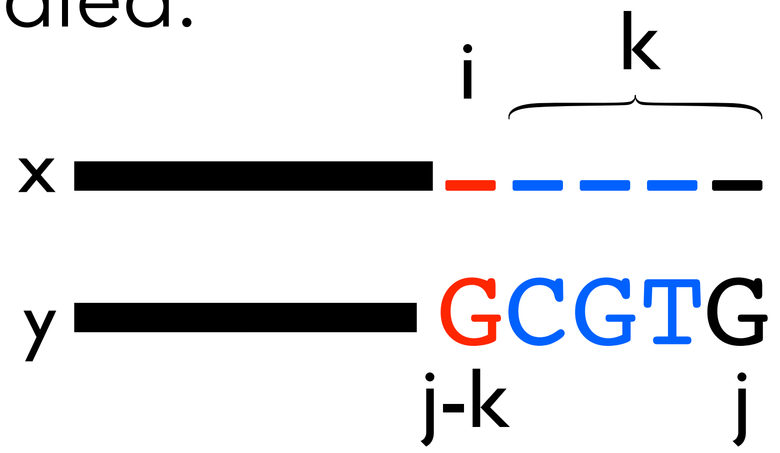
k decides how long to make the gap.

We have to make the whole gap at once in order to know how to score it.

$$X(i,j) = \max \begin{cases} M(i, j-k) + \text{gscore}(k) & \text{for } 1 \le k \le j \\ Y(i, j-k) + \text{gscore}(k) & \text{for } 1 \le k \le j \end{cases}$$



This case is automatically handled.



*

Running Time for Gap Penalties

$$\mathrm{M}(i,j) = \mathrm{score}(x_i, y_j) + \max\begin{cases}\mathrm{M}(i-1, j-1)\\ \mathrm{X}(i-1, j-1)\\ \mathrm{Y}(i-1, j-1)\end{cases}$$

$$\mathrm{X}(i,j) = \max\begin{cases}\mathrm{M}(i, j-k) + \mathrm{gscore}(k) & \text{for } 1 \le k \le j\\ \mathrm{Y}(i, j-k) + \mathrm{gscore}(k) & \text{for } 1 \le k \le j\end{cases}$$

$$\mathrm{Y}(i,j) = \max\begin{cases}\mathrm{M}(i-k, j) + \mathrm{gscore}(k) & \text{for } 1 \le k \le i\\ \mathrm{X}(i-k, j) + \mathrm{gscore}(k) & \text{for } 1 \le k \le i\end{cases}$$

Final score is max {M(n,m), X(n,m), Y(n,m)}.

How do you do the traceback?

Runtime:

- Assume |X| = |Y| = n for simplicity: $3n^2$ subproblems
- $2n^2$ subproblems take O(n) time to solve (**because we have to try all k**)

⇒ O(n^3) total time

*

Affine Gap Penalties

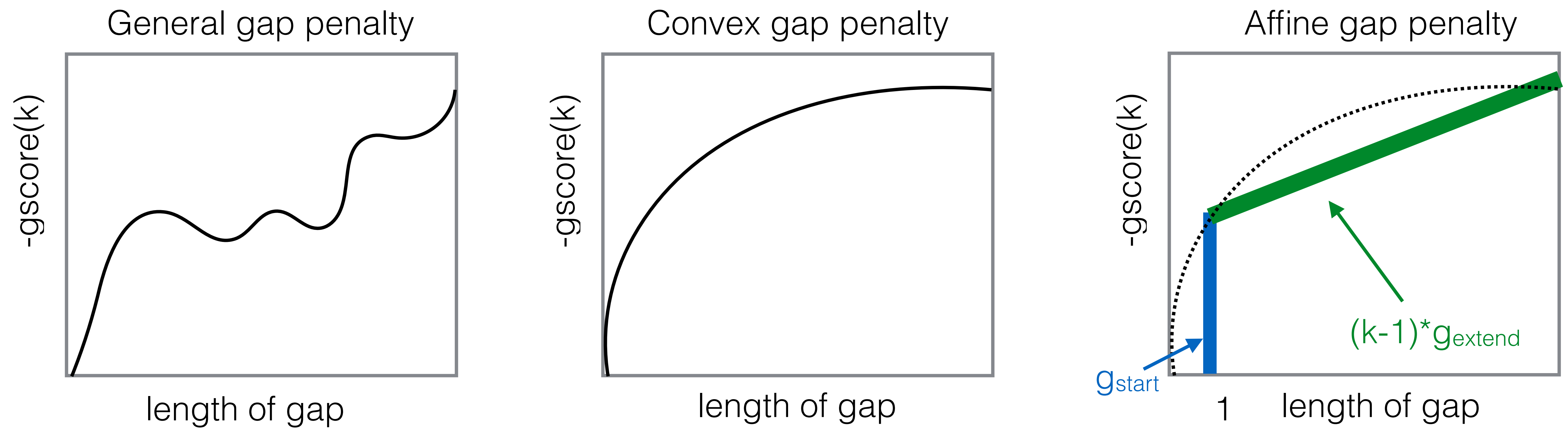
- $O(n^3)$ for general gap penalties is usually too slow…
- We can still encourage spaces to group together using a special case of general penalties called *affine gap penalties*:

  g_{start} = the cost of starting a gap

  g_{extend} = the cost of extending a gap by one more space

  $gscore(k) = g_{start} + (k-1) \times g_{extend}$

less restrictive ⇒ more restrictive

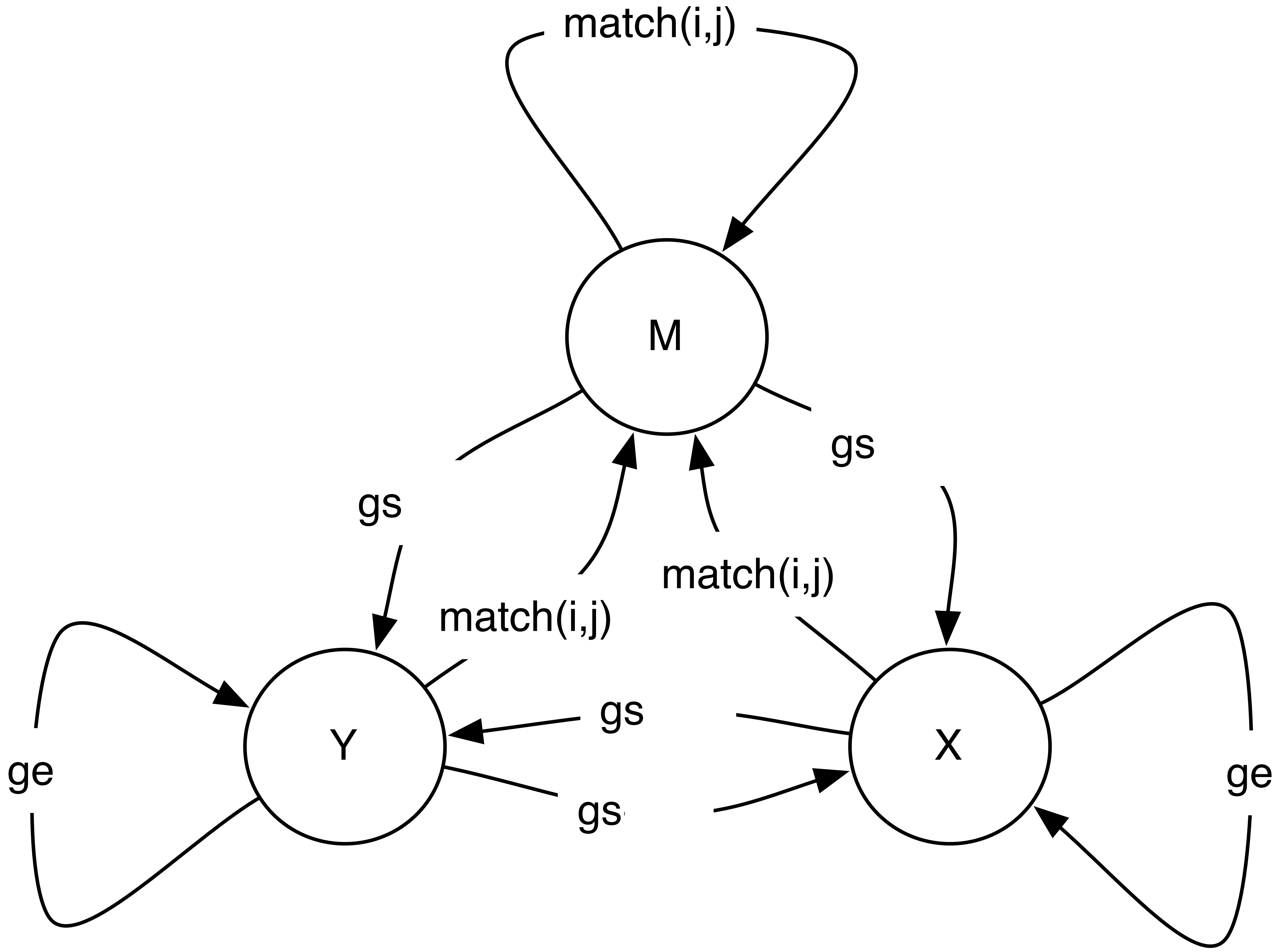


Benefit of Affine Gap Penalties

- Same idea of using 3 matrices, but now we *don't need to search over all gap lengths*, we just have to know whether we are **starting a new gap** or **not**.

*

Affine Gap as Finite State Machine



*

Affine Gap Penalties

$$M(i,j) = \text{score}(x_i, y_i) + \max \begin{cases} M(i-1, j-1) \\ X(i-1, j-1) \\ Y(i-1, j-1) \end{cases}$$

(mis)match between x and y

If previous alignment ends in (mis)match, this is a new gap

$$X(i,j) = \max \begin{cases} g_{\text{start}} + M(i, j-1) \\ g_{\text{extend}} + X(i, j-1) \\ g_{\text{start}} + Y(i, j-1) \end{cases}$$

gap in x

If we're using the X matrix, then we're extending a gap.

If we're using the Y matrix, then we're starting a new gap in this string.

$$Y(i,j) = \max \begin{cases} g_{\text{start}} + M(i-1, j) \\ g_{\text{start}} + X(i-1, j) \\ g_{\text{extend}} + Y(i-1, j) \end{cases}$$

gap in y

*

Affine Base Cases (Global)

- M(0, *i*) = "score of best alignment between 0 characters of *x* and *i* characters of *y* that ends in a match" = -∞ because no such alignment can exist.

- X(0, *i*) = "score of best alignment between 0 characters of *x* and *i* characters of *y* that ends in a gap in *x*" = gap_start + (i-1) × gap_extend because this alignment looks like:

```
----------
yyyyyyyyyy
```

- X(i, 0) = "score of best alignment between i characters of x and 0 characters of y that ends in a gap in X" = -∞

```
xxxxxxxxxx-
-----------
```

← not allowed

- M(i, 0) = M(0, i) and Y(0, i) and Y(i, 0) are computed using the same logic as X(i, 0) and X(0, i)

*

Affine Gap Runtime

- $3mn$ subproblems
- Each one takes **constant** time
- Total runtime $O(mn)$:
  - back to the run time of the basic running time.

Traceback

- Arrows now can point **between** matrices.
- The possible arrows are given, as usual, by the recurrence.
  - E.g. What arrows are possible leaving a cell in the M matrix?

*

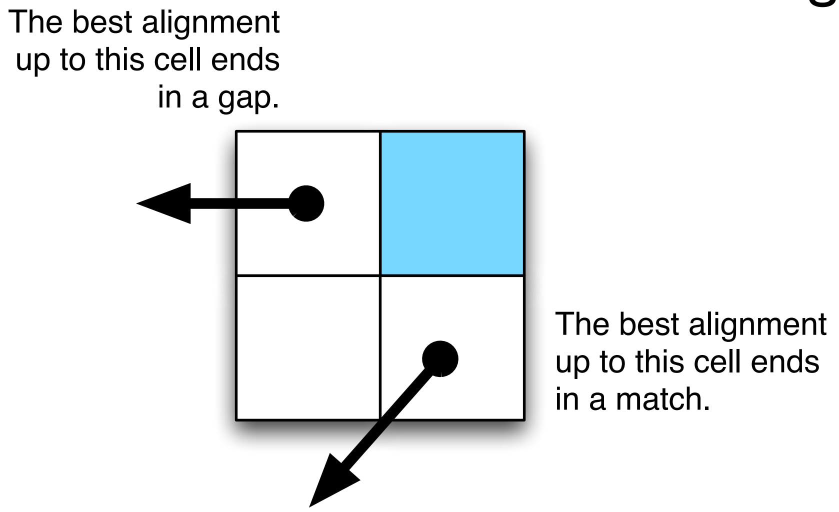
Why do you "need" 3 functions?

- Alternative WRONG algorithm:

```
M(i,j) = max(
    M(i-1, j-1) + cost($x_i$, $y_j$),
    M(i-1, j) +($g_{start}$ if Arrow(i-1, j) != ← , else $g_{extend}$),
    M(j, i-1) + ($g_{start}$ if Arrow(i, j-1) != ↓ , else $g_{extend}$)
)
```

WRONG Intuition: we only need to know whether we are starting a gap or extending a gap.

The arrows coming out of each subproblem tell us how the best alignment ends, so we can use them to decide if we are starting a new gap.


The best alignment up to this cell ends in a gap.
The best alignment up to this cell ends in a match.

PROBLEM: The best alignment for strings x[1..i] and y[1..j] doesn't have to be used in the best alignment between x[1..i+1] and y[1..j+1]

*

Why 3 Matrices: Example

match = 5, mismatch = -2, gap = -1, gap_start = -10

`x=CARTS, y=CAT`

```
CART
CA-T
```

$$\mathrm{OPT}(4, 3) = \text{optimal score} = \underbrace{15 - 10} = 5$$

```
CARTS
CA-T-
```

$$\mathrm{WRONG}(5, 3) = \overbrace{15 - 10} - 10 = -5$$

```
CARTS
CAT--
```

$$\mathrm{OPT}(5, 3) = 10 - 2 - 10 - 1 = -3$$

this is why we need to keep the X and Y matrices around.
they tell us the score of ending with a gap in one of the sequences.

*

Side Note: Lower Bounds

- Suppose the lengths of *x* and *y* are *n*.
- Clearly, need at least $\Omega(n)$ time to find their global alignment (have to read the strings!)
- The DP algorithms show global alignment can be done in $O(n^2)$ time.
- A trick called the "Four Russians Speedup" can make a similar dynamic programming algorithm run in $O(n^2 / \log n)$ time.
  - We probably won't talk about the Four Russians Speedup.
  - The important thing to remember is that only one of the four authors is Russian...

(Alrazarov, Dinic, Kronrod, Faradzev, 1970)

- Open questions: Can we do better? Can we prove that we can't do better? No#

#: Backurs, Arturs, and Piotr Indyk. "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)." *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. ACM, 2015.

*

Recap

- Local alignment: extra “0” case.
- General gap penalties require 3 matrices and O(n^3) time.
- Affine gap penalties require 3 matrices, but only O(n^2) time.

*