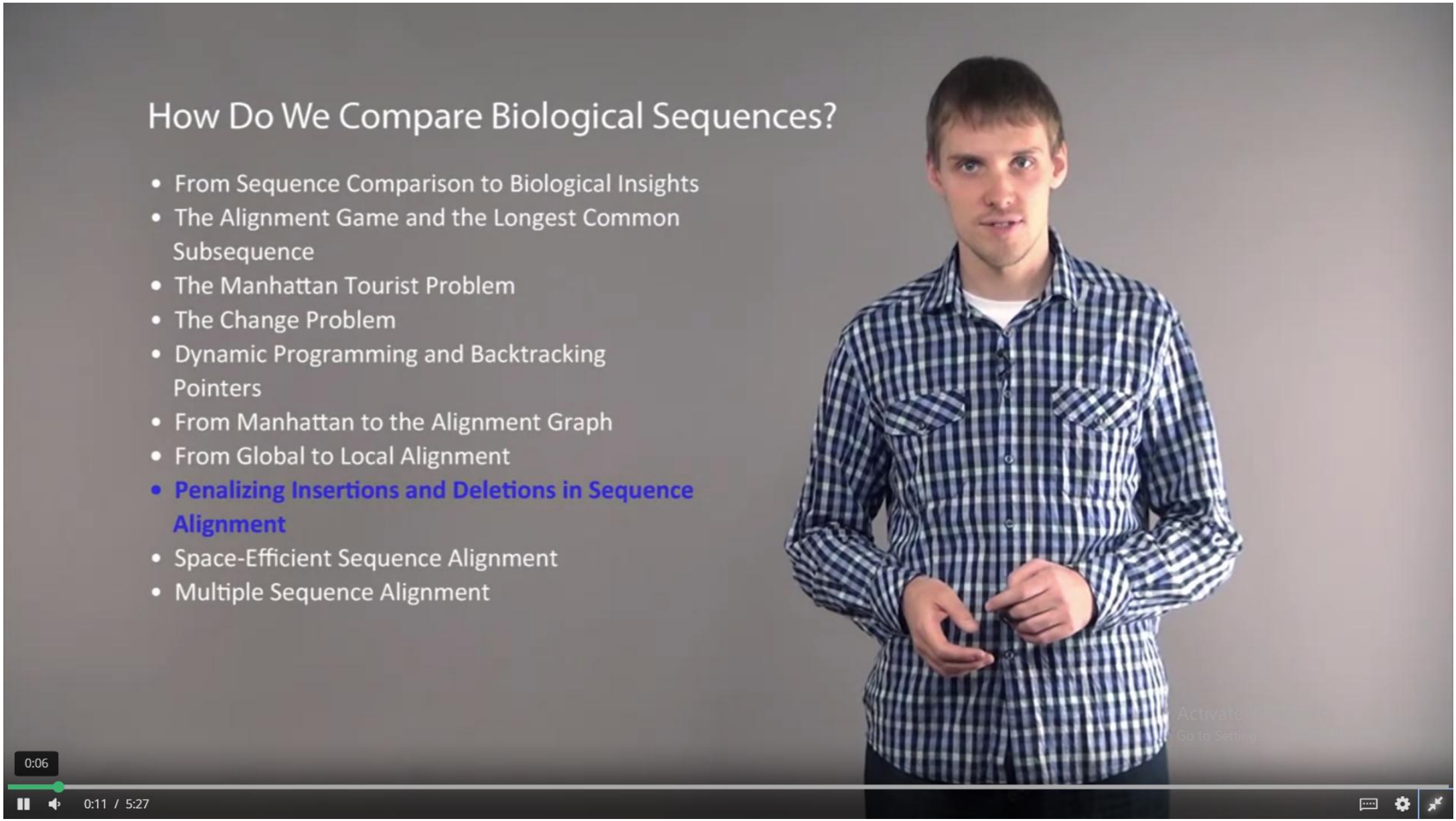

How Do We Compare Biological Sequences?
From Sequence Comparison to Biological Insights
The Alignment Game and the Longest Common Subsequence
The Manhattan Tourist Problem
The Change Problem
Dynamic Programming and Backtracking Pointers
From Manhattan to the Alignment Graph
From Global to Local Alignment
Penalizing Insertions and Deletions in Sequence Alignment
Space-Efficient Sequence Alignment
Multiple Sequence Alignment
0:06
0:11 / 5:27

Naive Scoring of Indels

We previously assigned a fixed penalty σ to each indel

However, this fixed penalty may be too severe for a series of 100 consecutive indels

A series of k indels often represents a single evolutionary event (**gap**) rather than k events:

two gaps (assign lower score)	GATCCAG GA-C-AG	GATCCAG GA--CAG	a single gap (assign higher score)

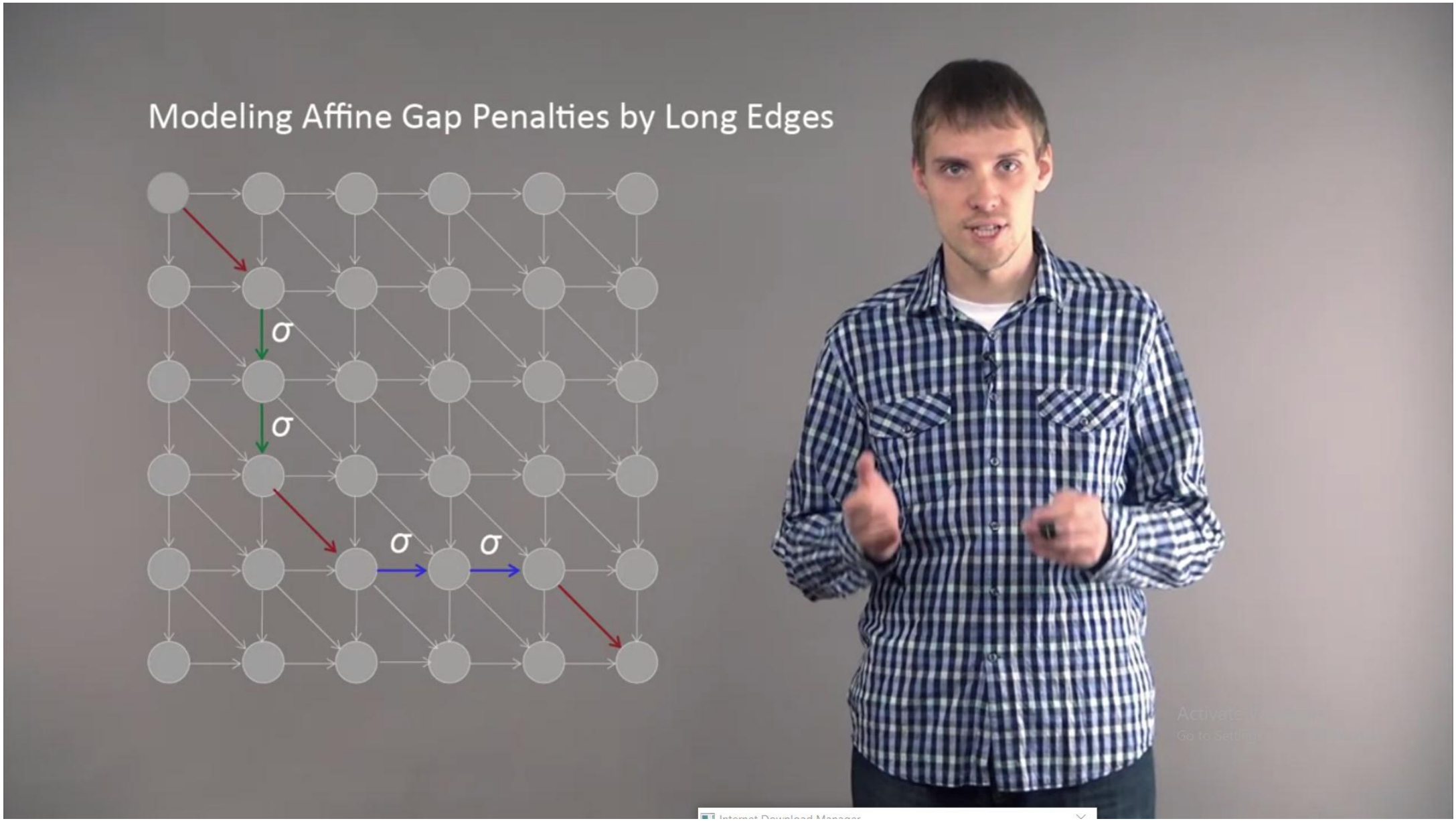
More Adequate Gap Penalties

Affine gap penalty for a gap of length k: $\sigma+\varepsilon\cdot(k-1)$

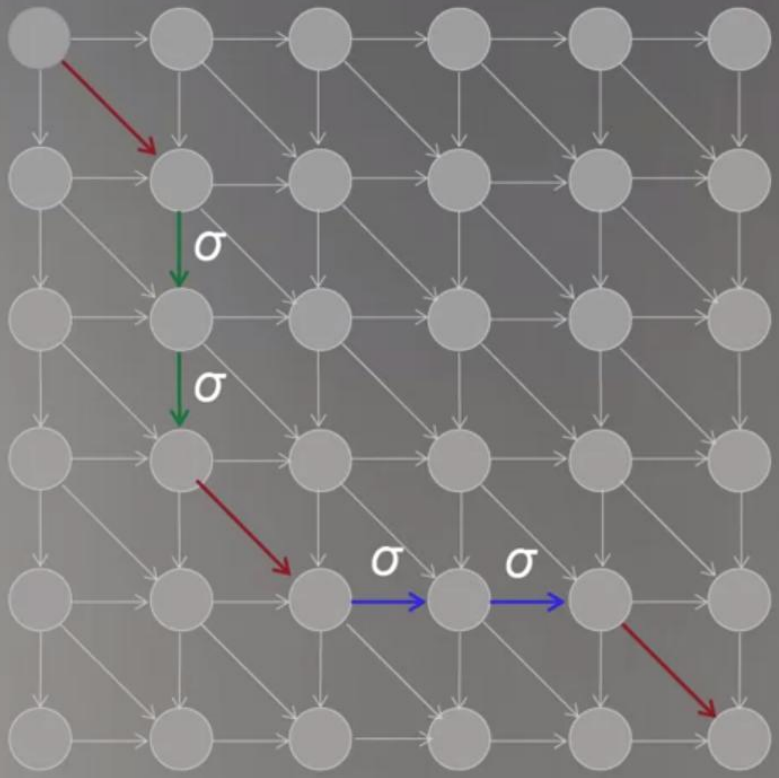
σ - the **gap opening penalty**

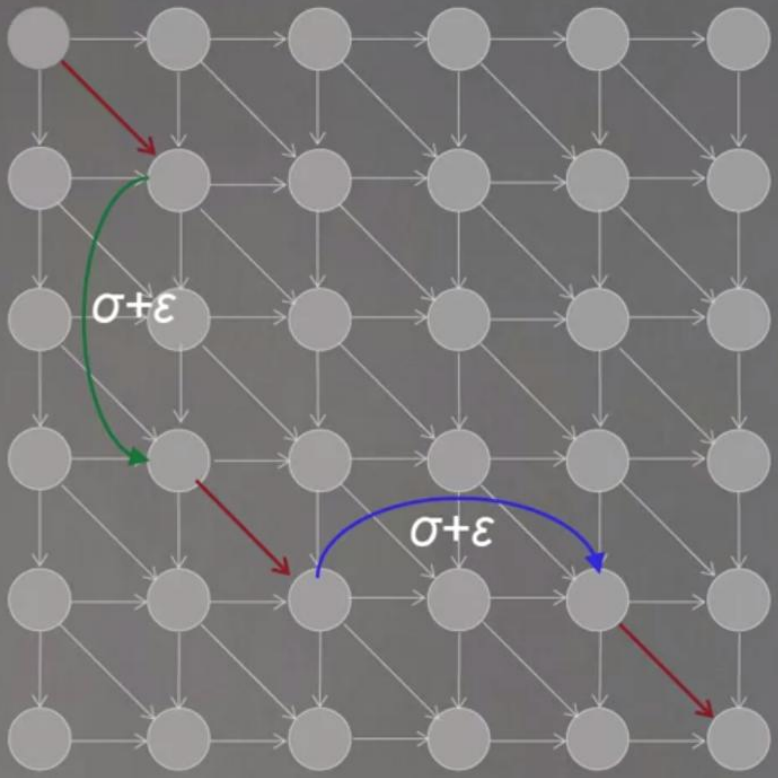
ε - the **gap extension penalty**

$\sigma > \varepsilon$, since starting a gap should be penalized more than extending it.

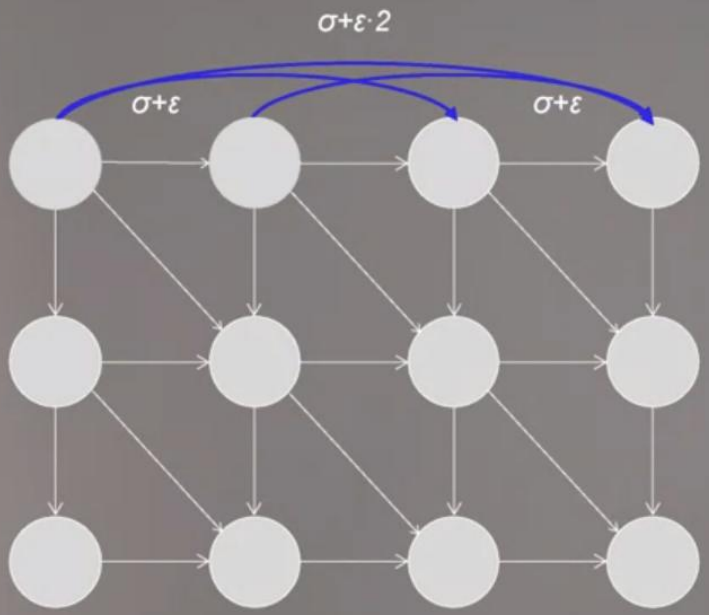

Modeling Affine Gap Penalties by Long Edges
σ
σ
σ
σ

Modeling Affine Gap Penalties by Long Edges

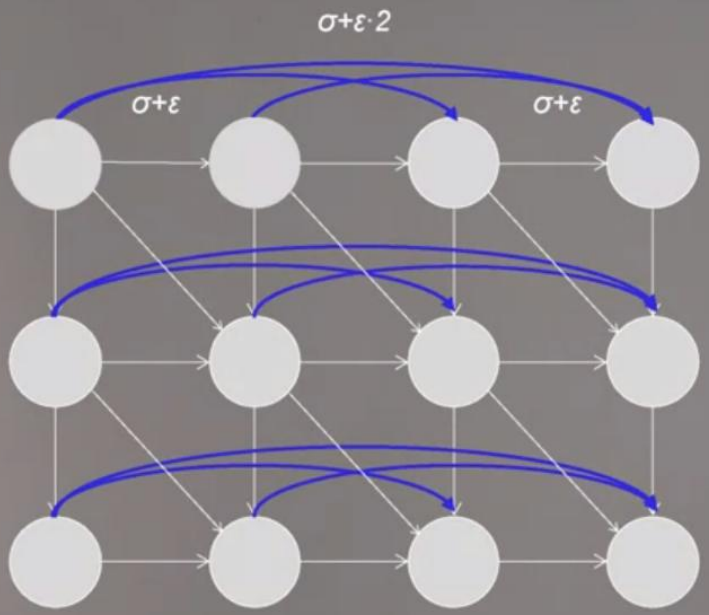




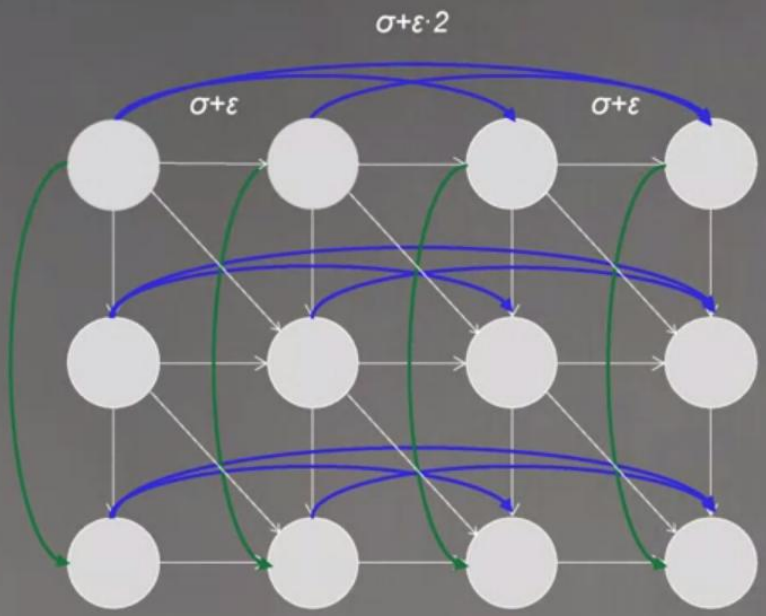
Building Manhattan with Affine Gap Penalties



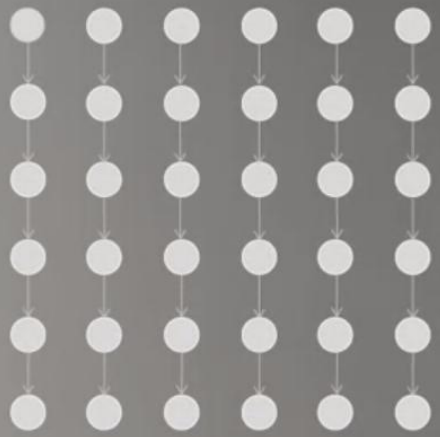
Building Manhattan with Affine Gap Penalties



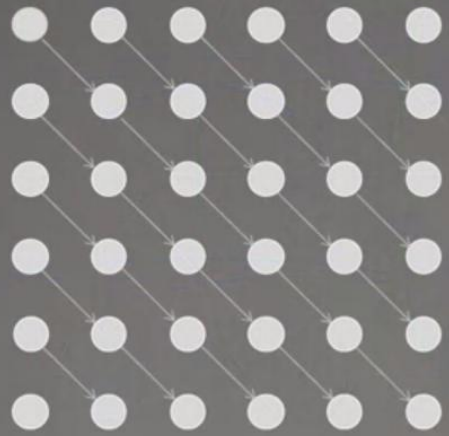
Building Manhattan with Affine Gap Penalties



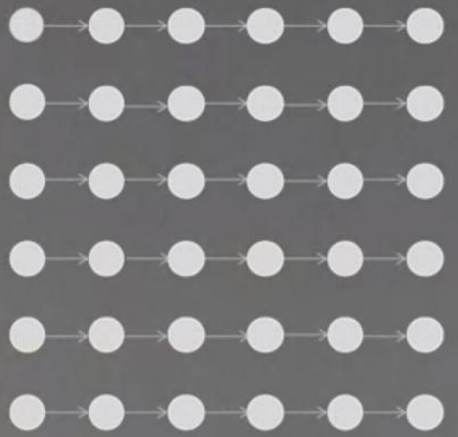
Instead:
Building Manhattan on 3 levels



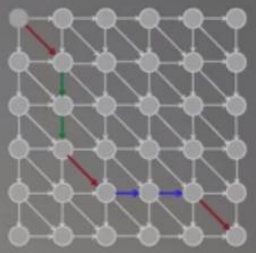
bottom level
(insertions)



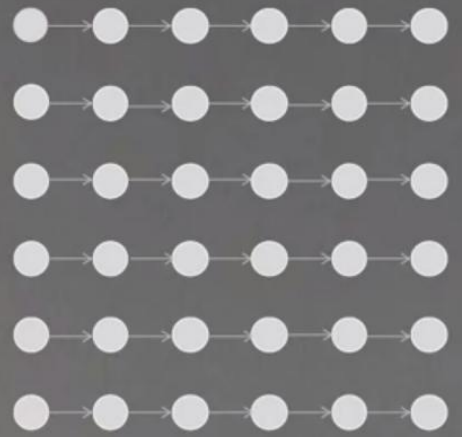
middle level
(matches/mismatches)

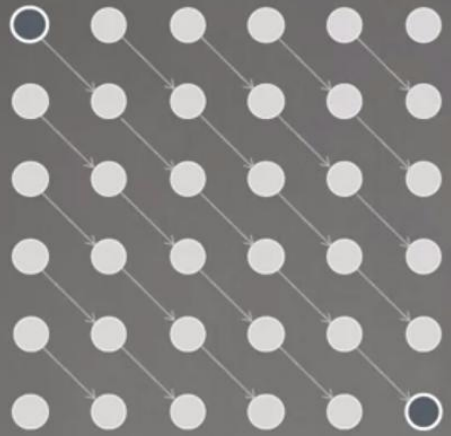


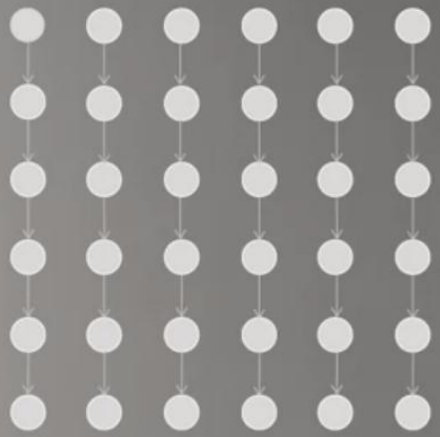
upper level
(deletions)

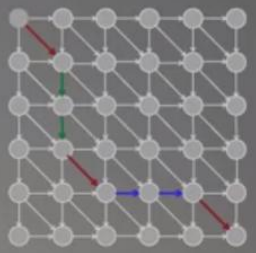


How can we emulate this path in the 3-level Manhattan?

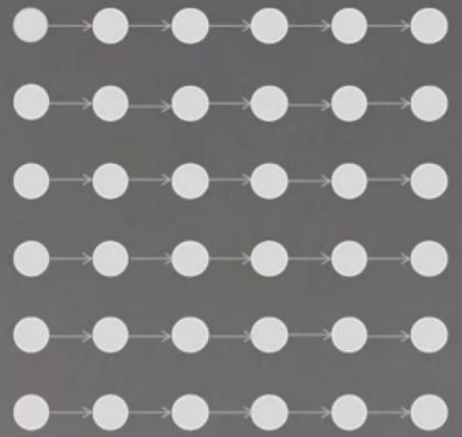


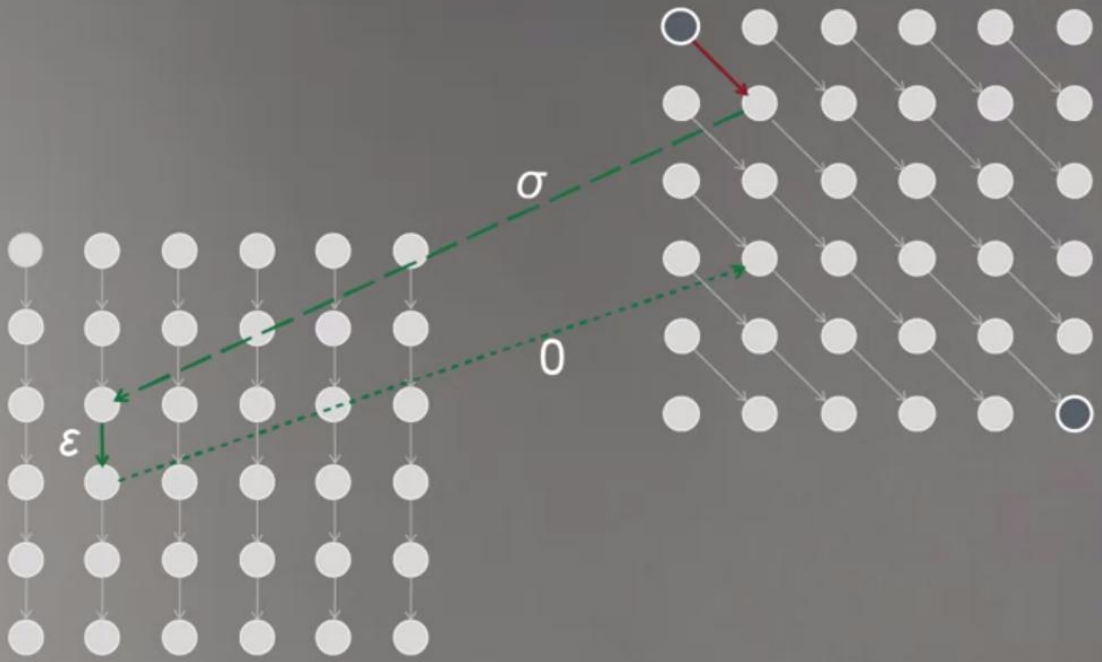


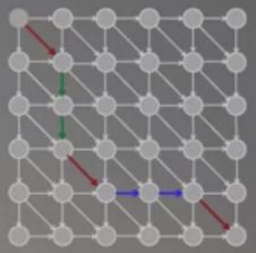




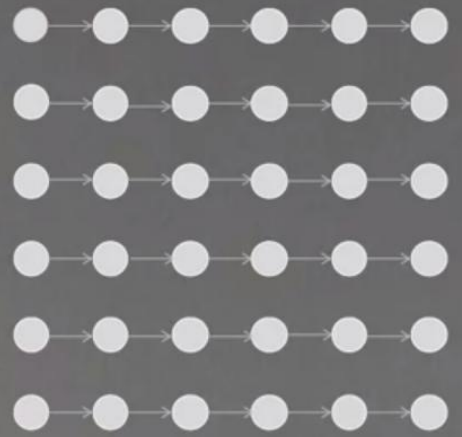
How can we emulate this path in the 3-level Manhattan?

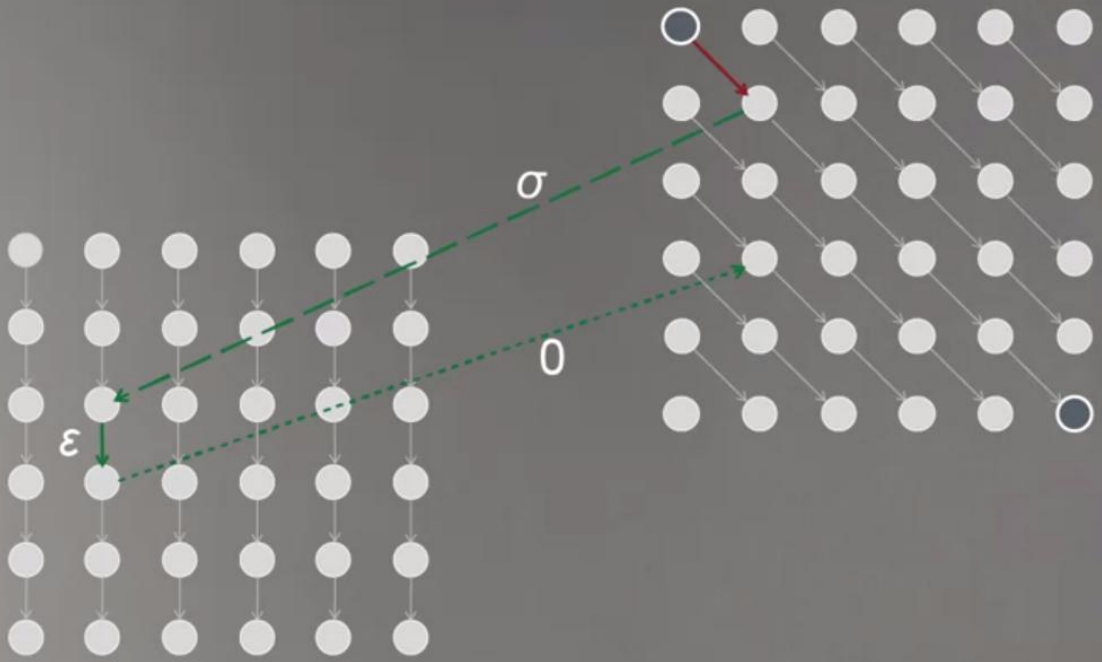


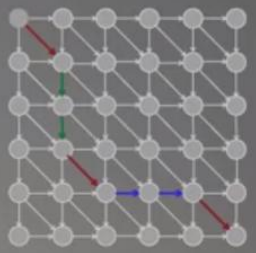




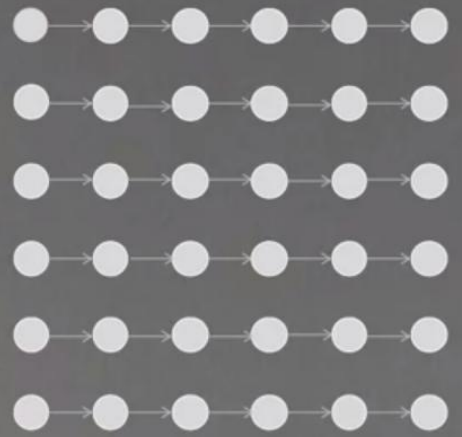
How can we emulate this path in the 3-level Manhattan?

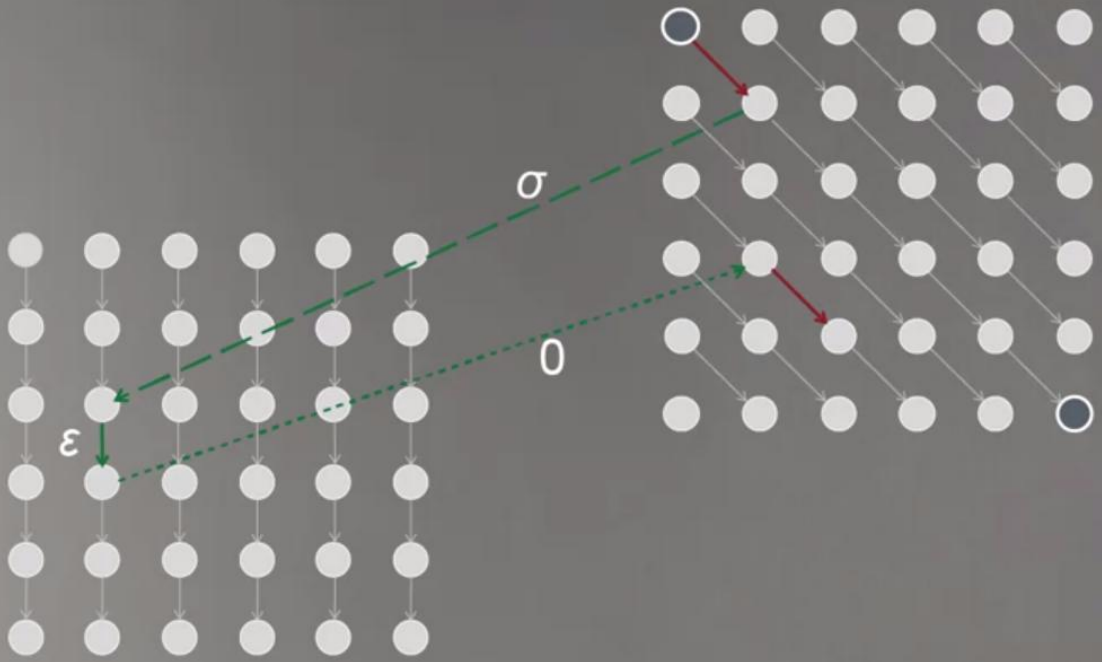


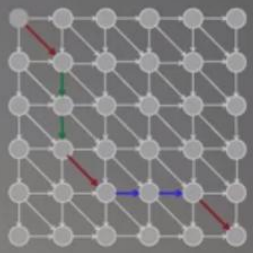




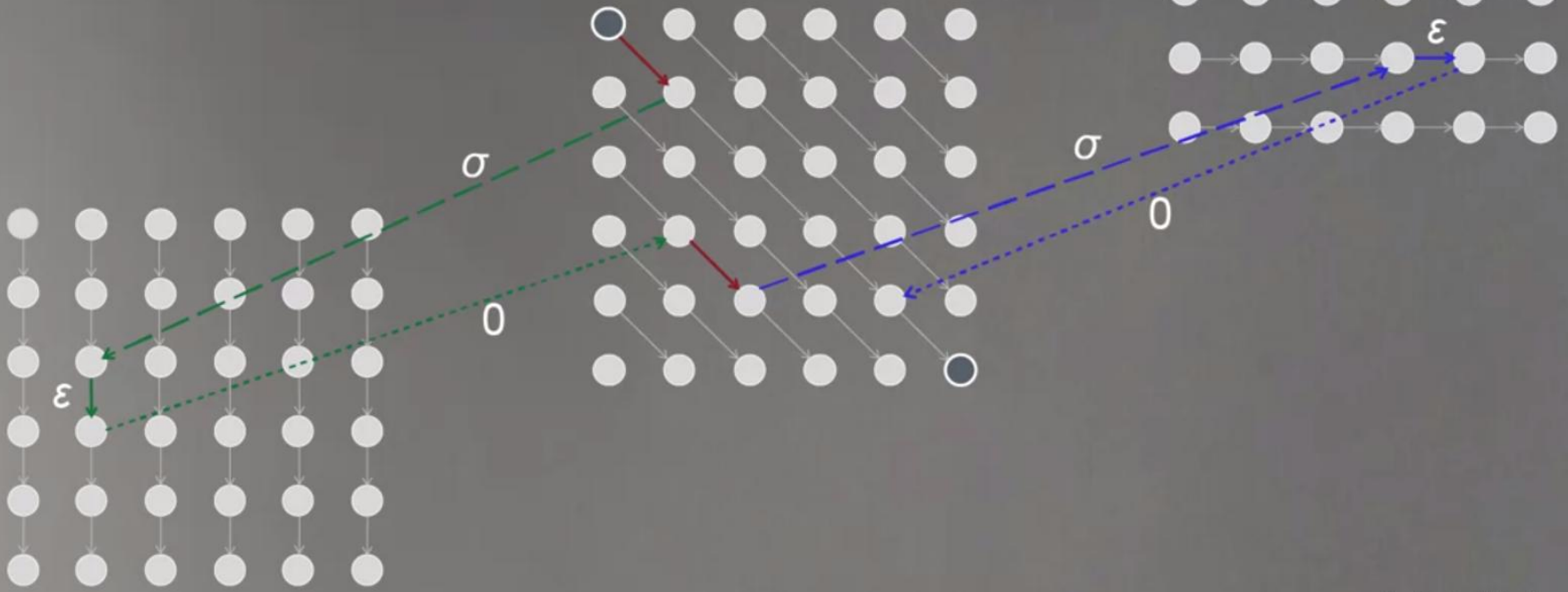
How can we emulate this path in the 3-level Manhattan?

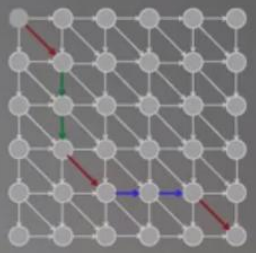






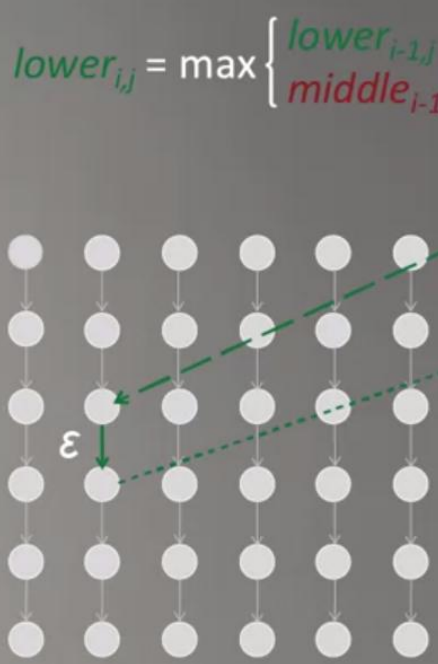
How can we emulate this path in the 3-level Manhattan?

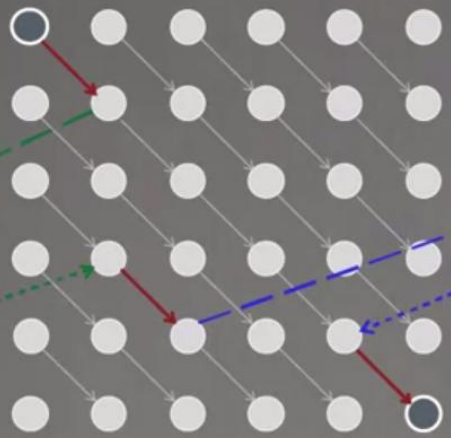


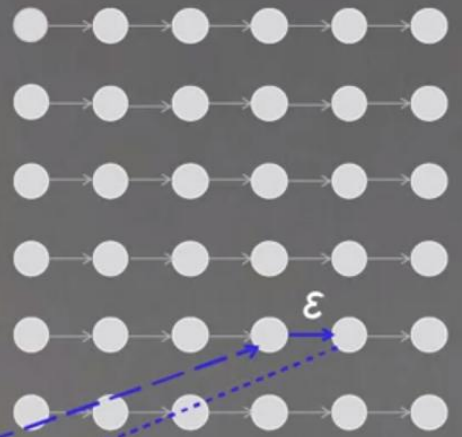


How can we emulate this path in the 3-level Manhattan?

$$lower_{i,j} = \max \begin{cases} lower_{i-1,j} - \varepsilon \\ middle_{i-1,j} - \sigma \end{cases}$$



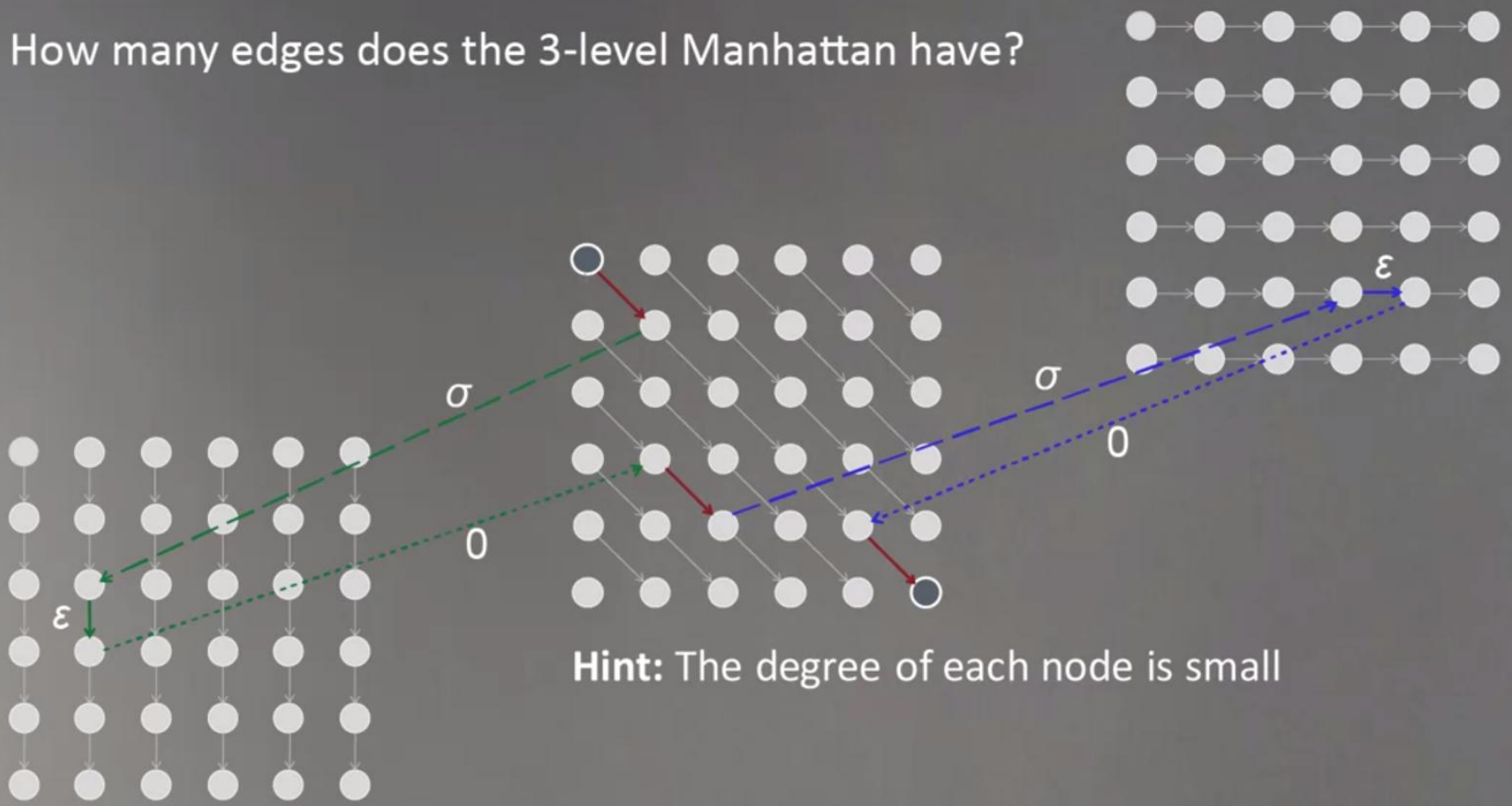




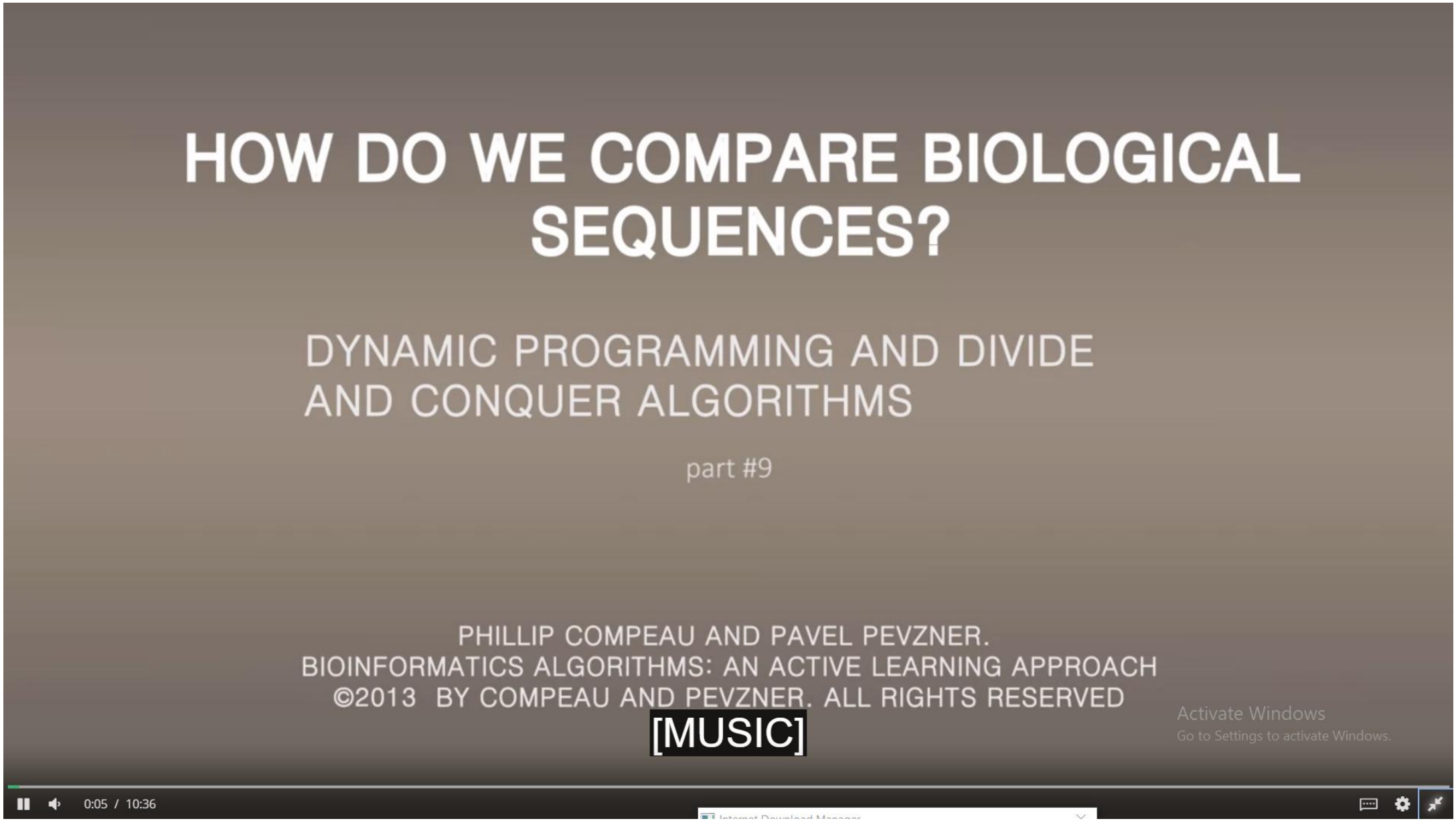
$$upper_{i,j} = \max \begin{cases} upper_{i,j-1} - \varepsilon \\ middle_{i,j-1} - \sigma \end{cases}$$

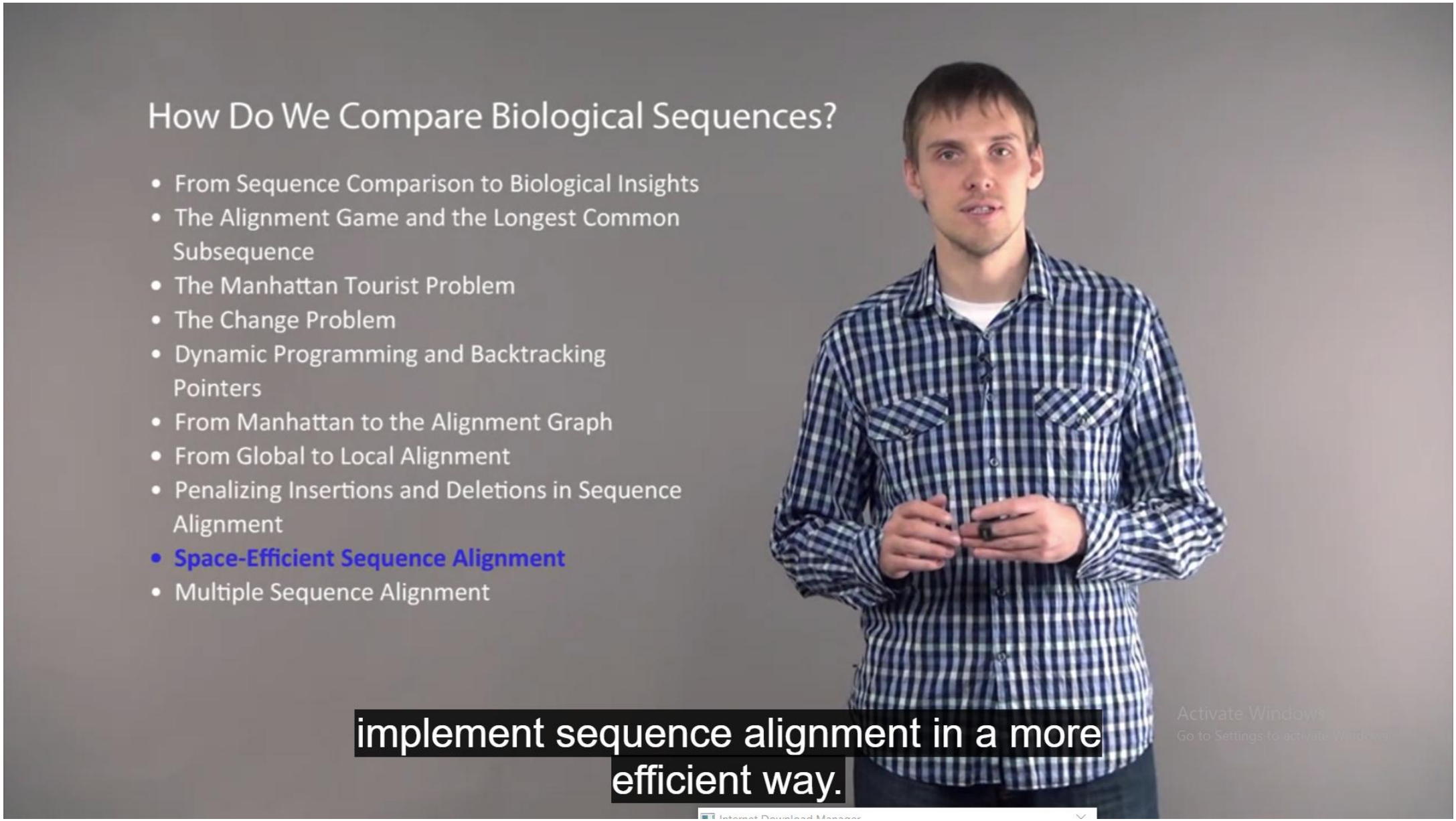
$$middle_{i,j} = \max \begin{cases} lower_{i,j} \\ middle_{i-1,j-1} + score(v_i, w_j) \\ upper_{i,j} \end{cases}$$

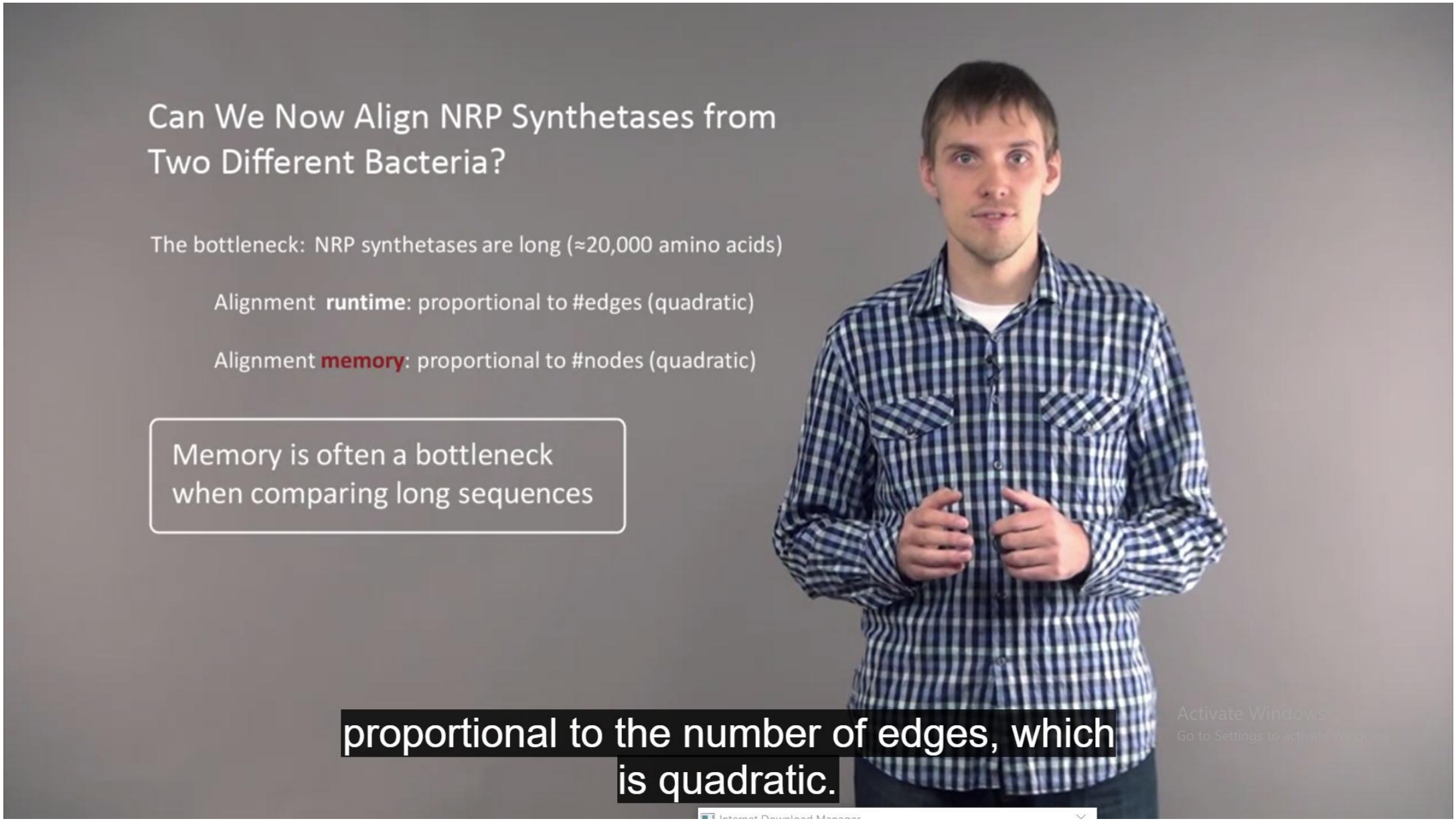
How many edges does the 3-level Manhattan have?

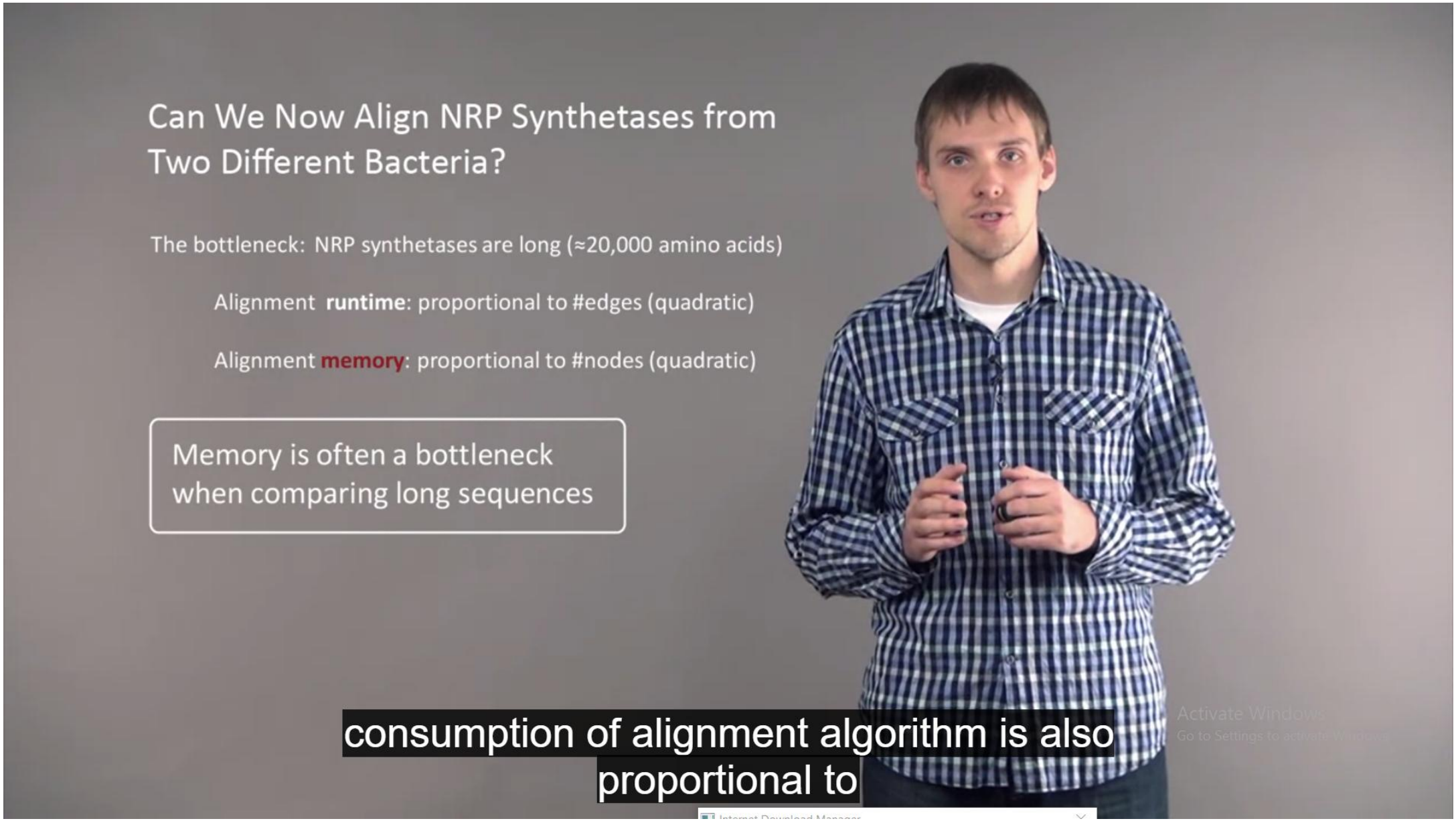


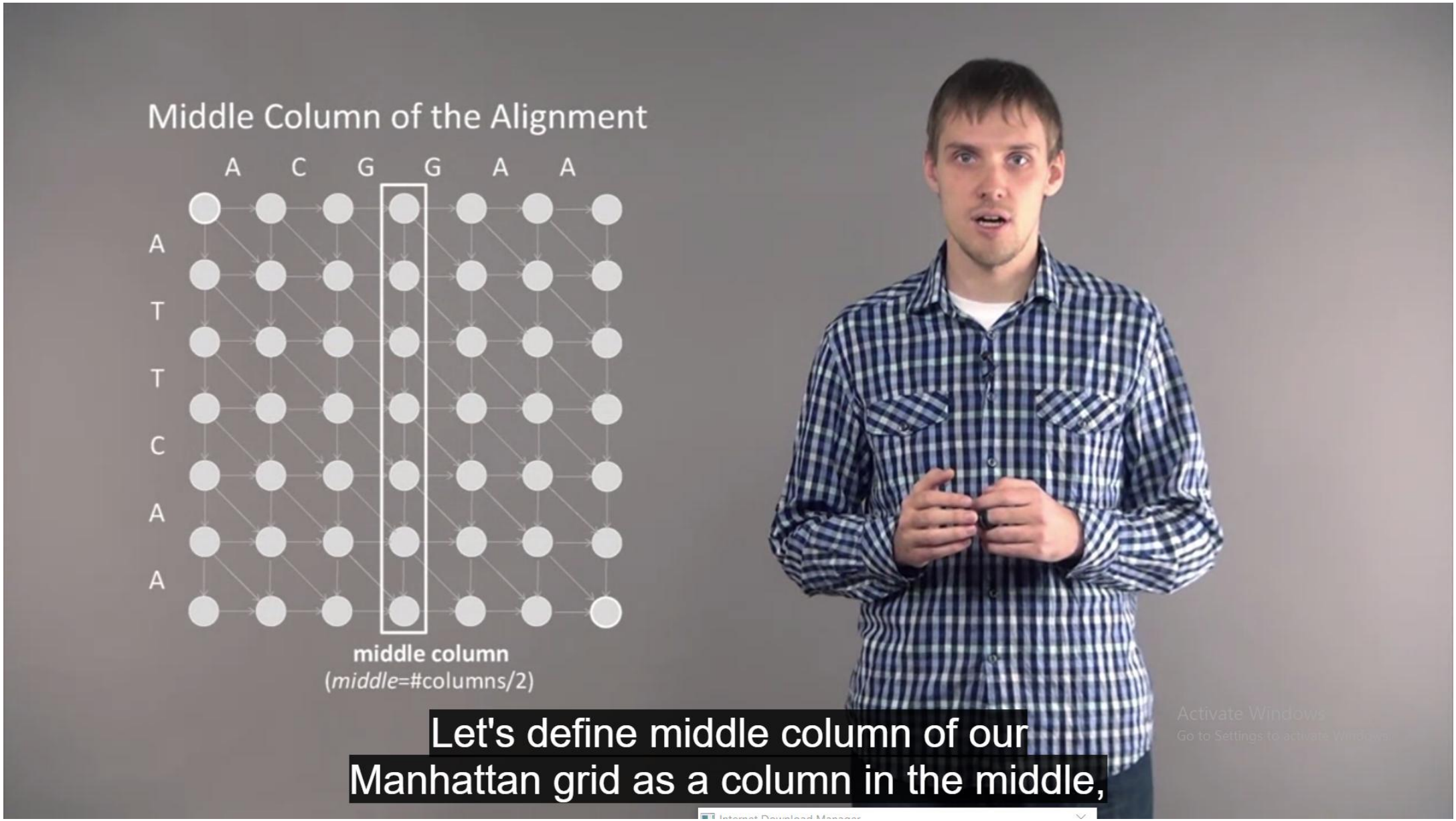
Hint: The degree of each node is small


HOW DO WE COMPARE BIOLOGICAL SEQUENCES?
DYNAMIC PROGRAMMING AND DIVIDE AND CONQUER ALGORITHMS
part #9
PHILLIP COMPEAU AND PAVEL PEVZNER.
BIOINFORMATICS ALGORITHMS: AN ACTIVE LEARNING APPROACH
©2013 BY COMPEAU AND PEVZNER. ALL RIGHTS RESERVED
[MUSIC]
Activate Windows
Go to Settings to activate Windows.
0:05 / 10:36


How Do We Compare Biological Sequences?
From Sequence Comparison to Biological Insights
The Alignment Game and the Longest Common Subsequence
The Manhattan Tourist Problem
The Change Problem
Dynamic Programming and Backtracking Pointers
From Manhattan to the Alignment Graph
From Global to Local Alignment
Penalizing Insertions and Deletions in Sequence Alignment
Space-Efficient Sequence Alignment
Multiple Sequence Alignment
implement sequence alignment in a more efficient way.
Activate Windows
Go to Settings to activate Windows.


Can We Now Align NRP Synthetases from Two Different Bacteria?
The bottleneck: NRP synthetases are long (≈20,000 amino acids)
Alignment runtime: proportional to #edges (quadratic)
Alignment memory: proportional to #nodes (quadratic)
Memory is often a bottleneck when comparing long sequences
proportional to the number of edges, which is quadratic.
Activate Windows
Go to Settings to activate Windows.

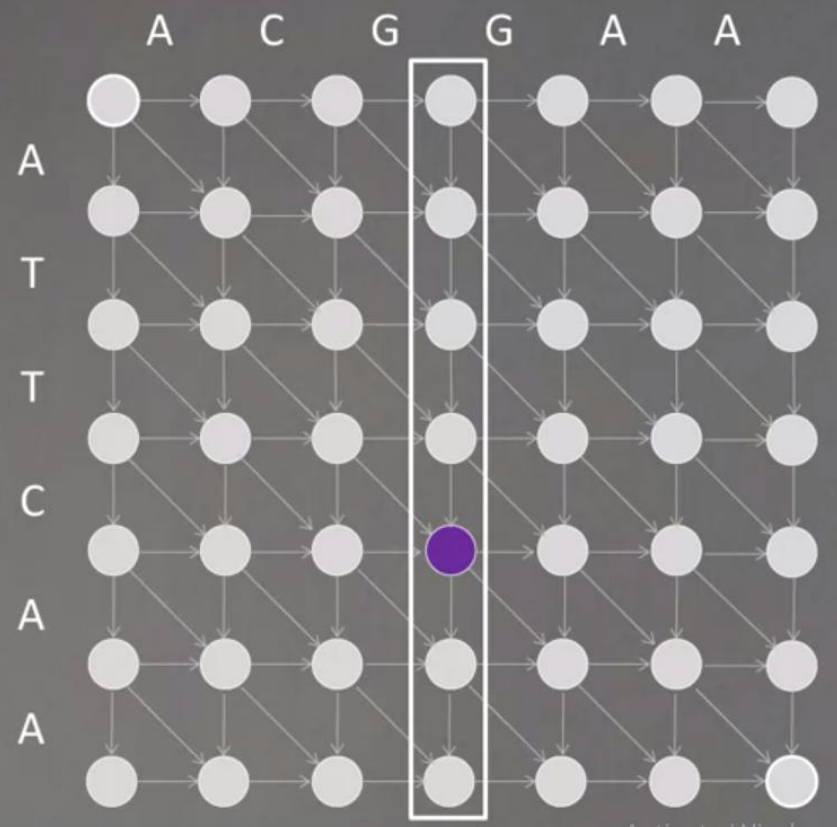

Can We Now Align NRP Synthetases from Two Different Bacteria?
The bottleneck: NRP synthetases are long (≈20,000 amino acids)
Alignment runtime: proportional to #edges (quadratic)
Alignment memory: proportional to #nodes (quadratic)
Memory is often a bottleneck when comparing long sequences
consumption of alignment algorithm is also proportional to
Activate Windows
Go to Settings to activate Windows.


Middle Column of the Alignment
A C G G A A
A
T
T
C
A
A
middle column
(middle=#columns/2)
Let's define middle column of our
Manhattan grid as a column in the middle,
Activate Windows
Go to Settings to activate Windows.

Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source, sink*)

find *MiddleNode*



Let's design algorithm to find sequence alignment.

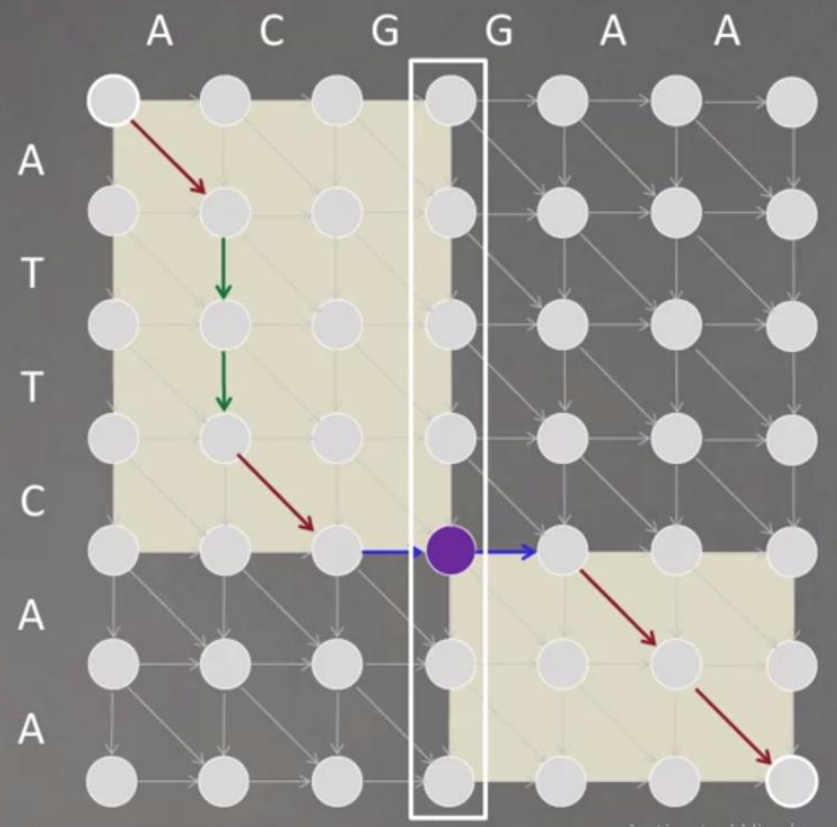
Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source, sink*)

find *MiddleNode*

AlignmentPath(*source, MiddleNode*)

AlignmentPath(*MiddleNode, sink*)



since we know all parts of a path, we can do it recursively.

Divide and Conquer Approach to Sequence Alignment

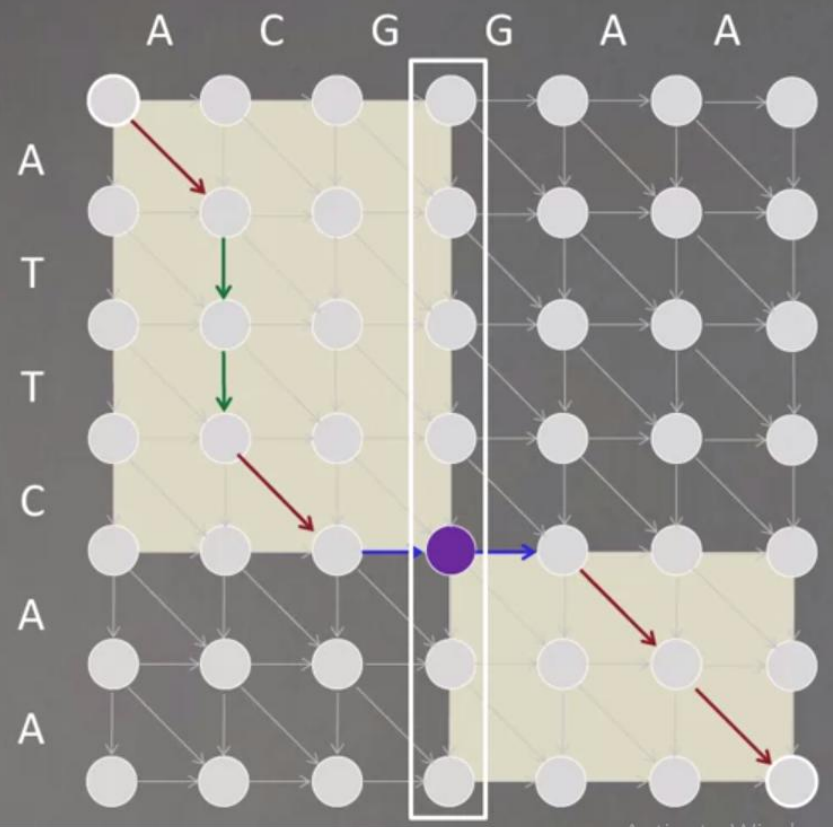
AlignmentPath(*source, sink*)

find *MiddleNode*

AlignmentPath(*source, MiddleNode*)

AlignmentPath(*MiddleNode, sink*)

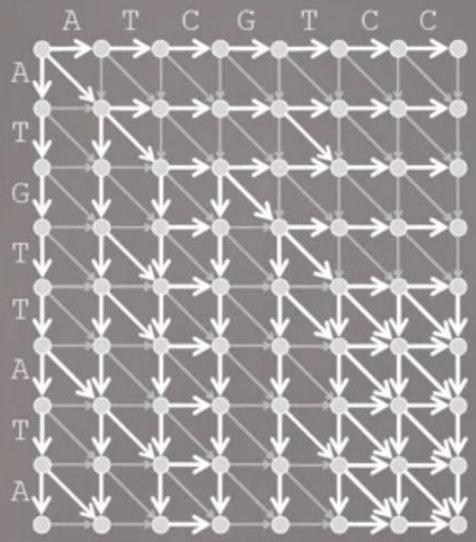
The only problem left is how to find this middle node in **linear space**!



problem is how to find this middle node, because we don't know yet.

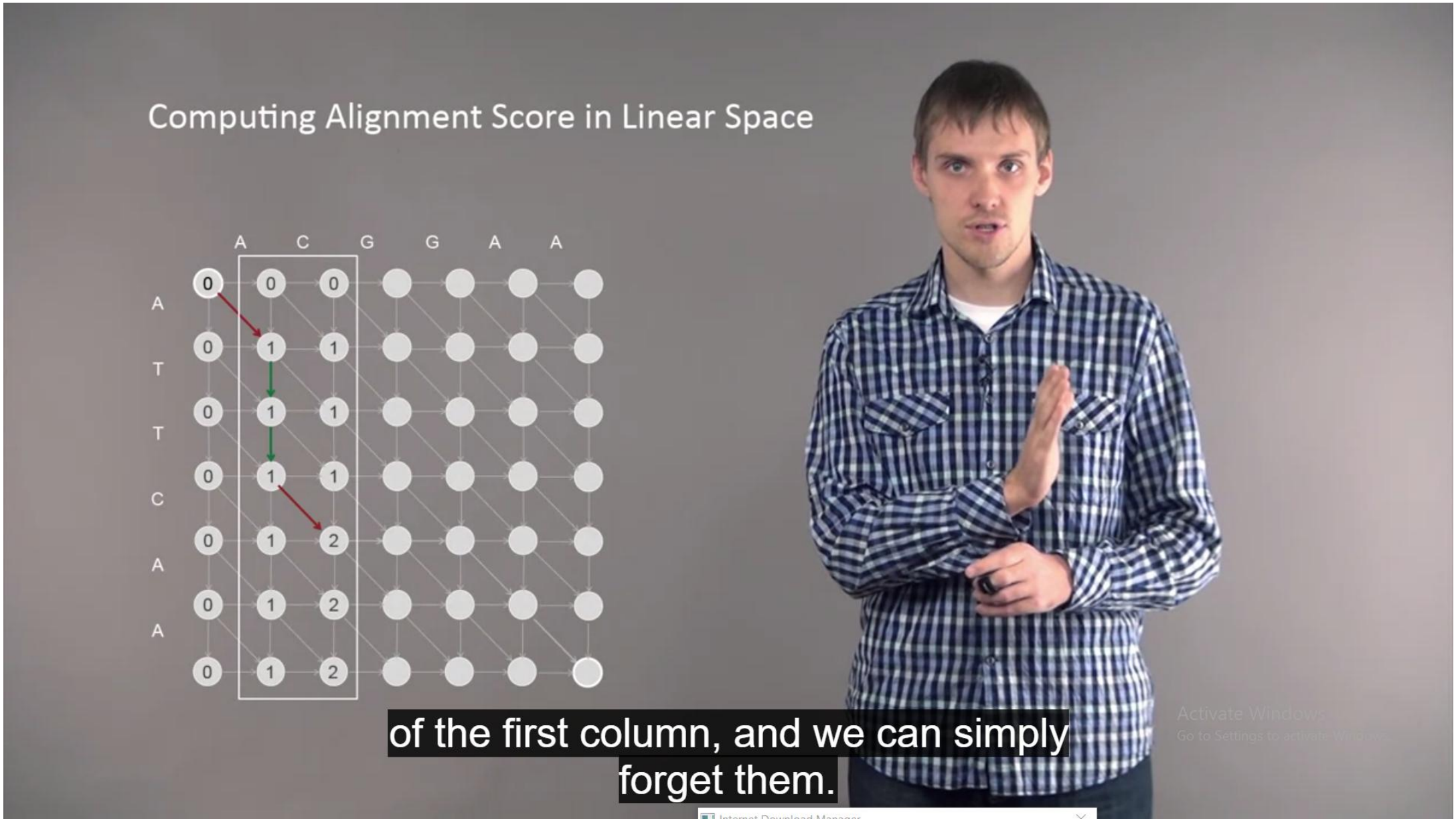
Computing Alignment Score in Linear Space

Finding the **longest path** in the alignment graph **requires** storing all backtracking pointers – O(*nm*) memory

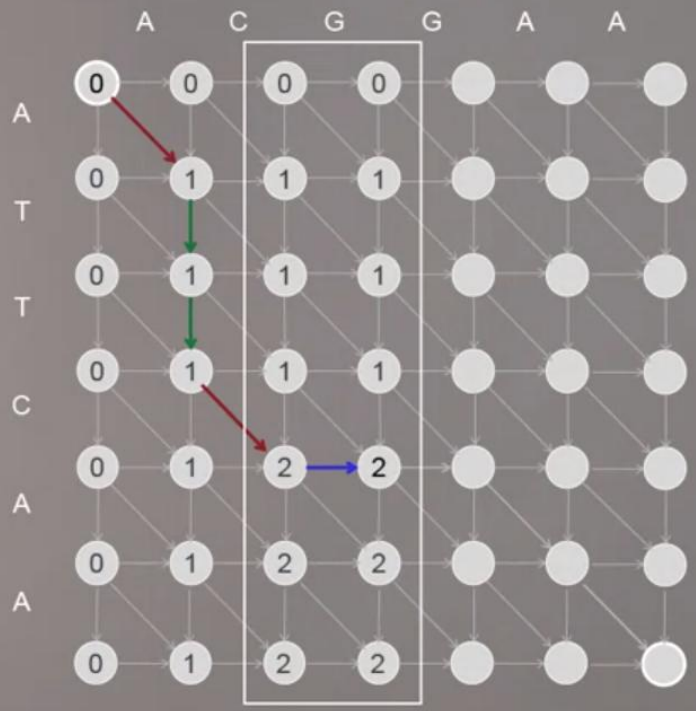


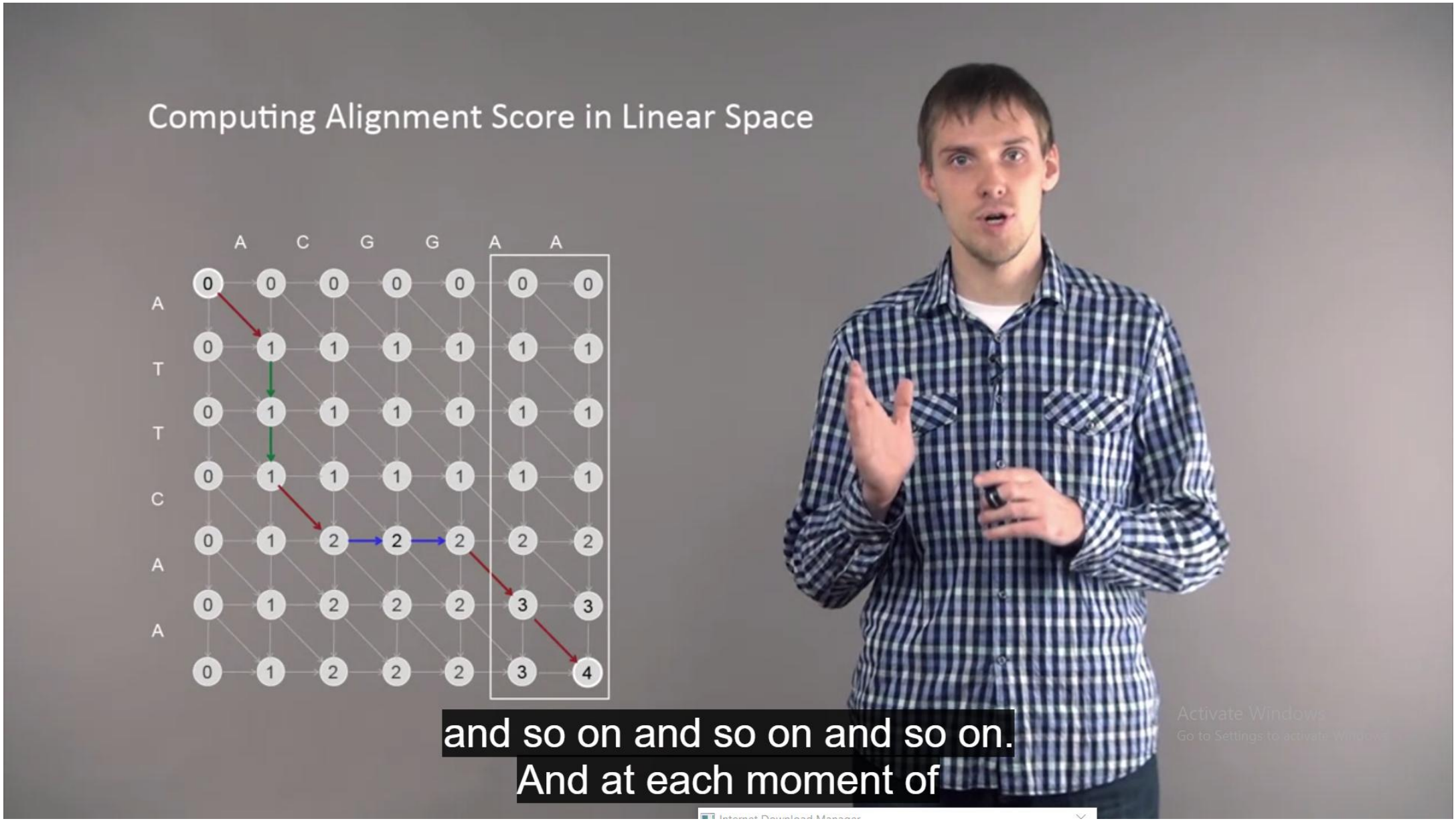
Computing the **length of the longest path does not require** storing any backtracking pointers – O(*n*) memory

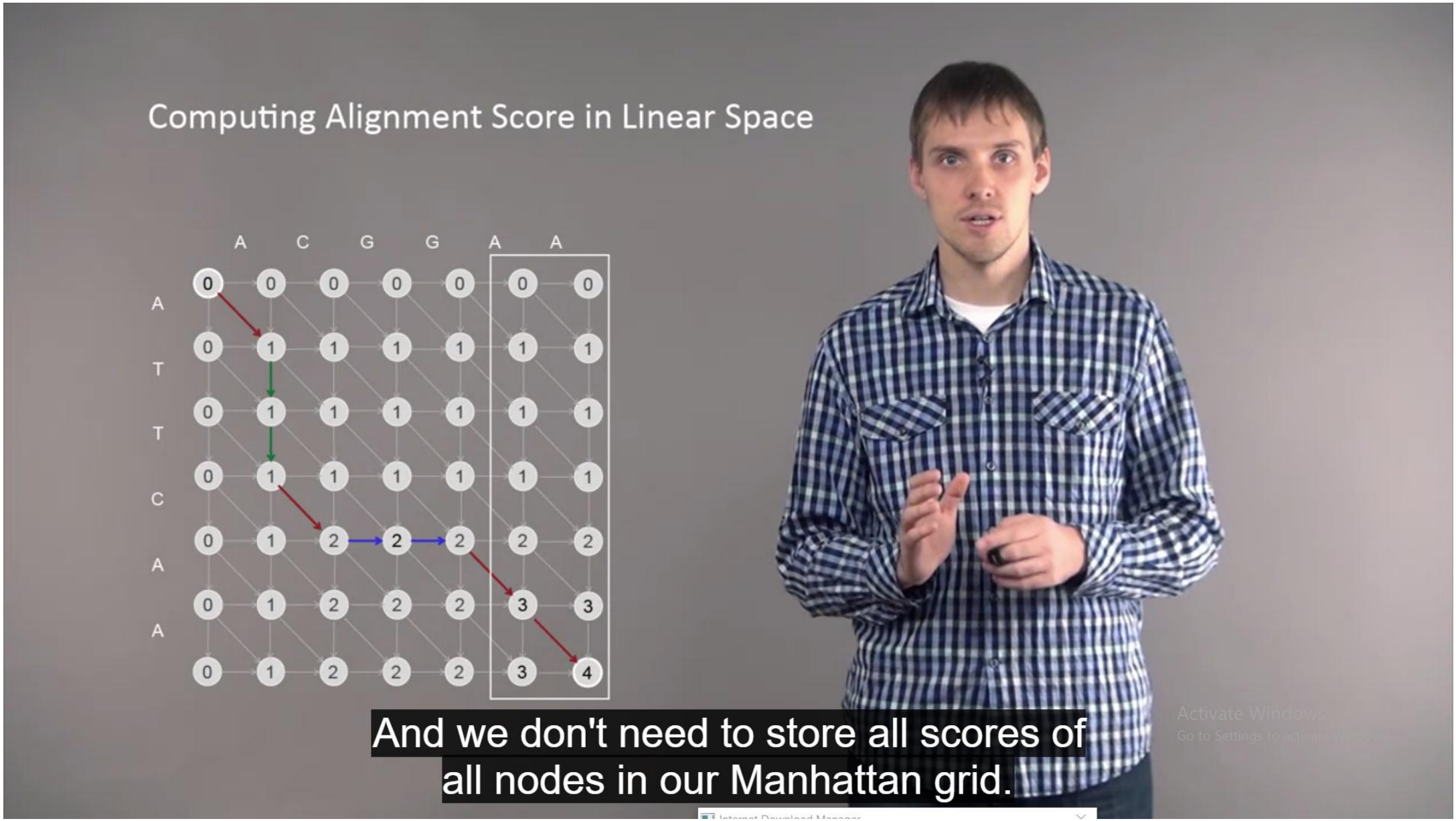
Can we skip

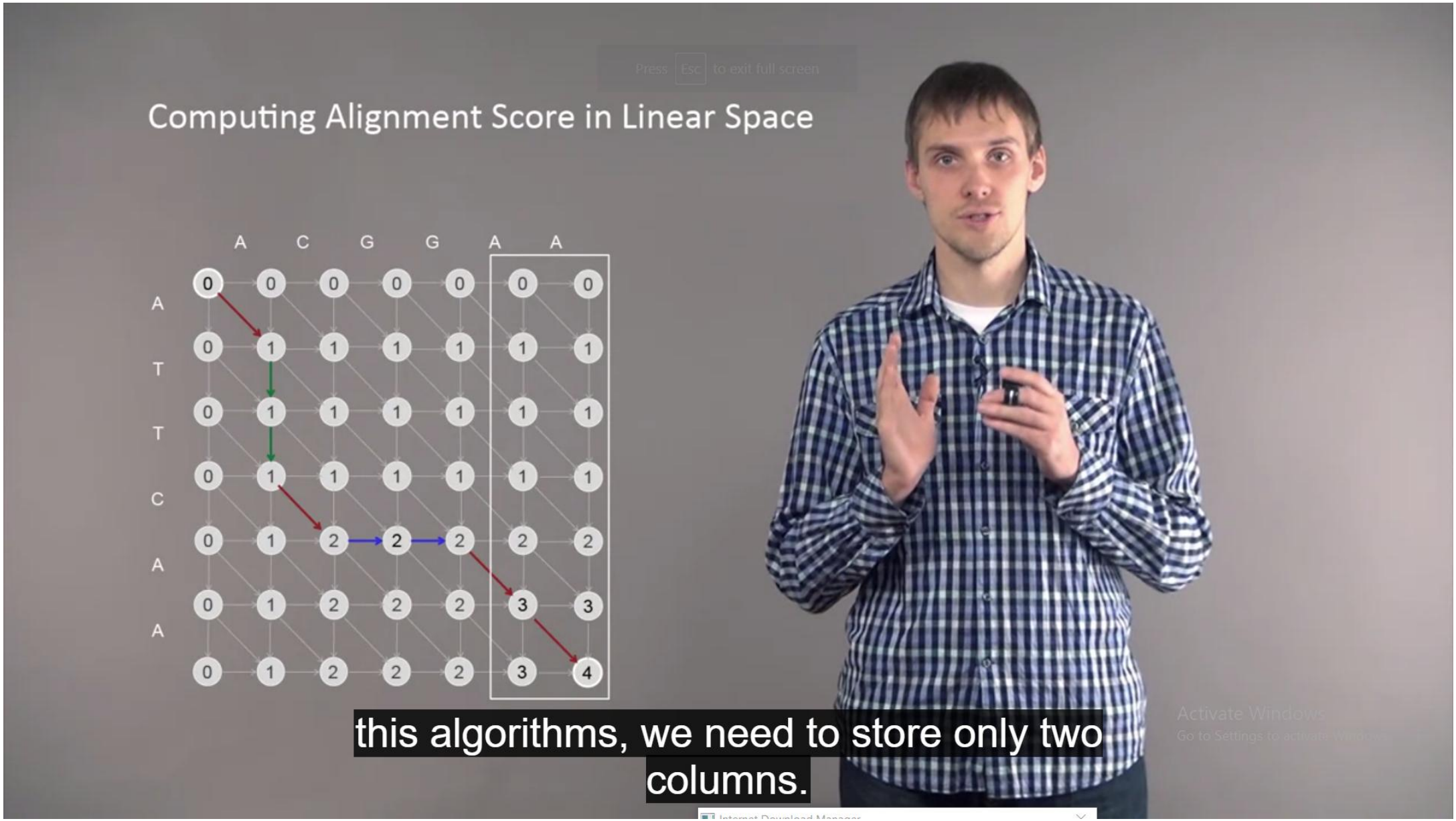

Computing Alignment Score in Linear Space
A C G G A A
A T T C A A
0 0 0
0 1 1
0 1 1
0 1 1
0 1 2
0 1 2
0 1 2
of the first column, and we can simply
forget them.
Activate Windows
Go to Settings to activate Windows.

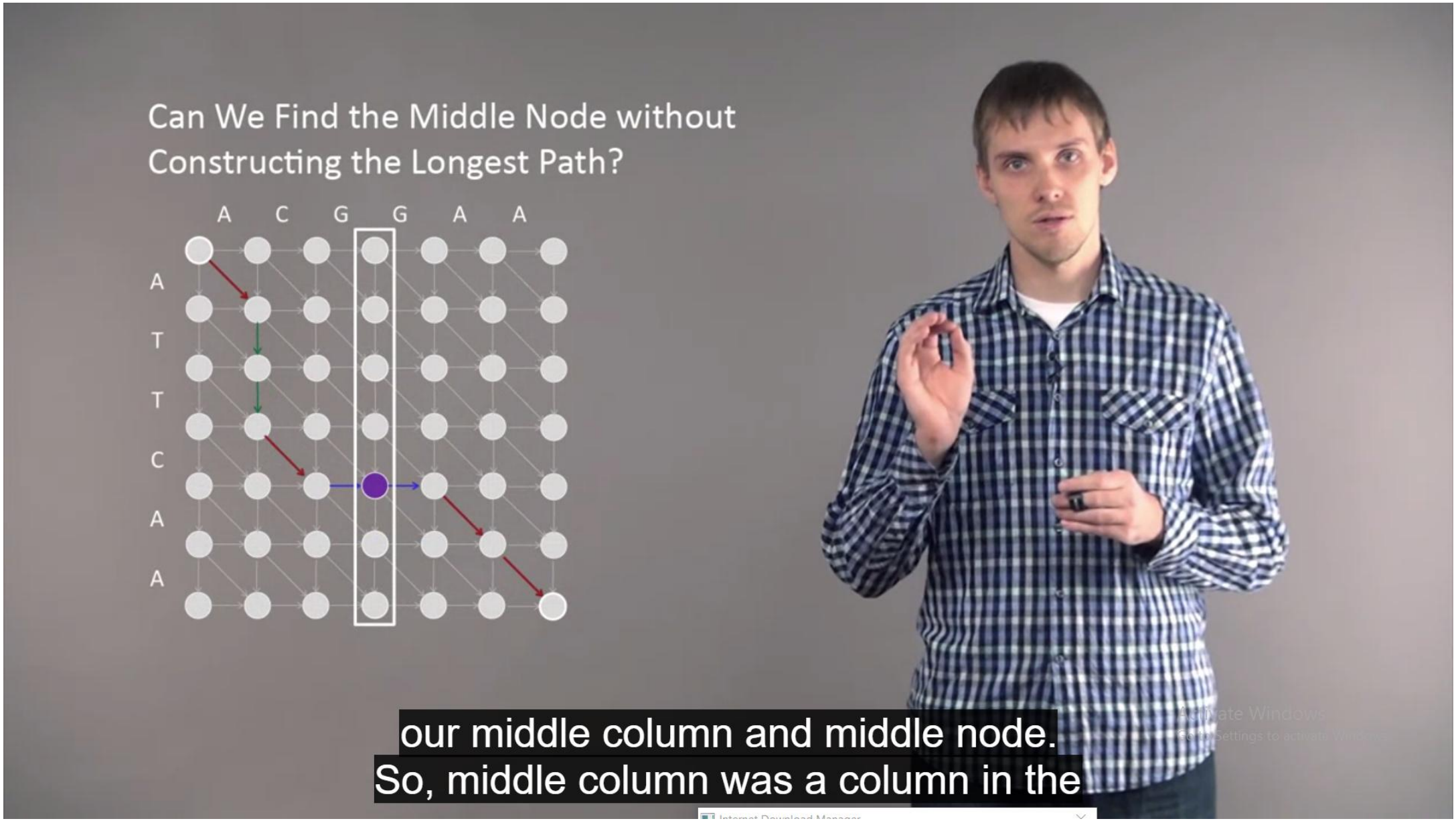
Computing Alignment Score in Linear Space

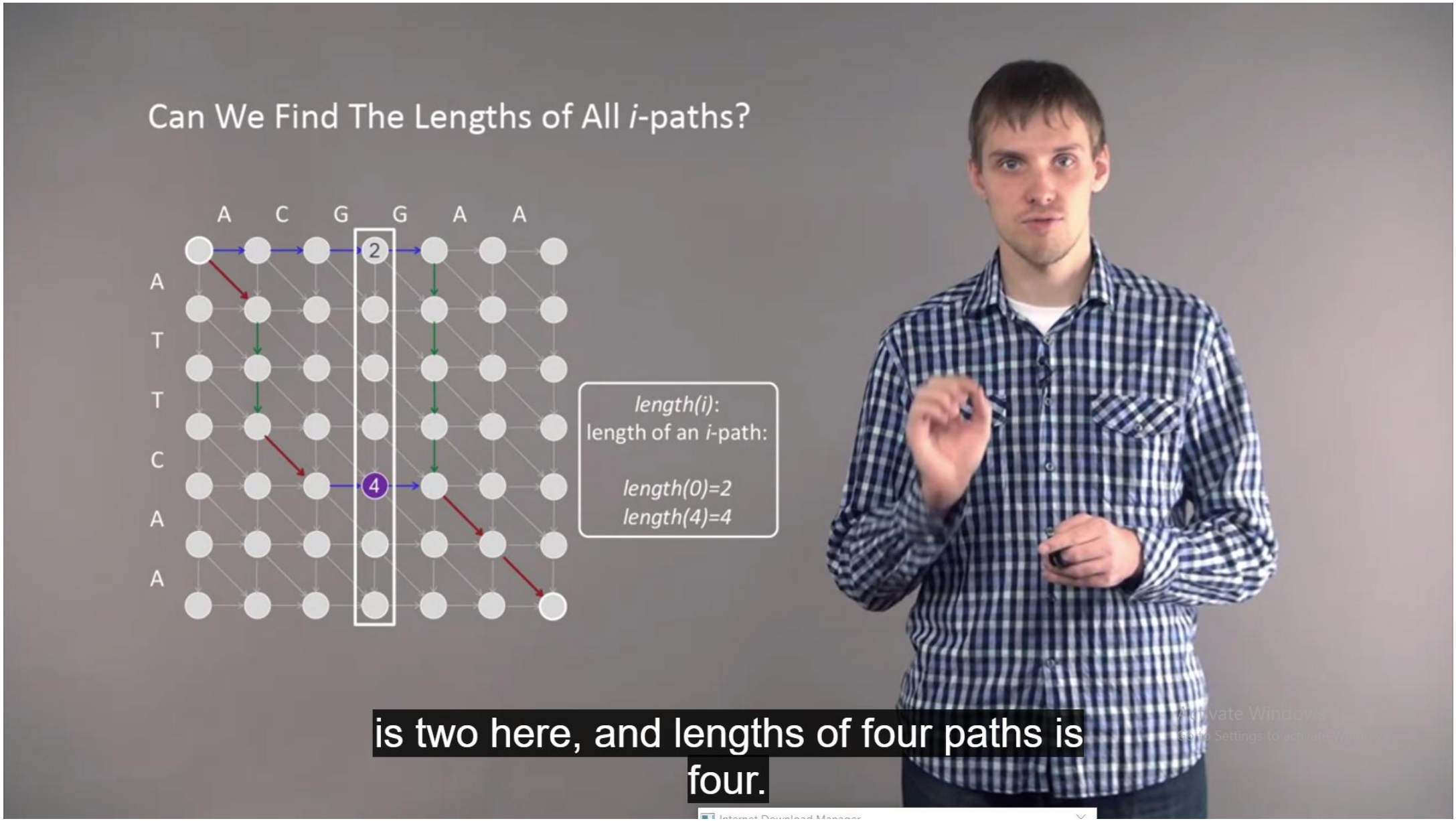



Computing Alignment Score in Linear Space
A C G G A A
A T T C A A
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 1 2 2 2 3 3
0 1 2 2 2 3 4
and so on and so on and so on.
And at each moment of
Activate Windows
Go to Settings to activate Windows.

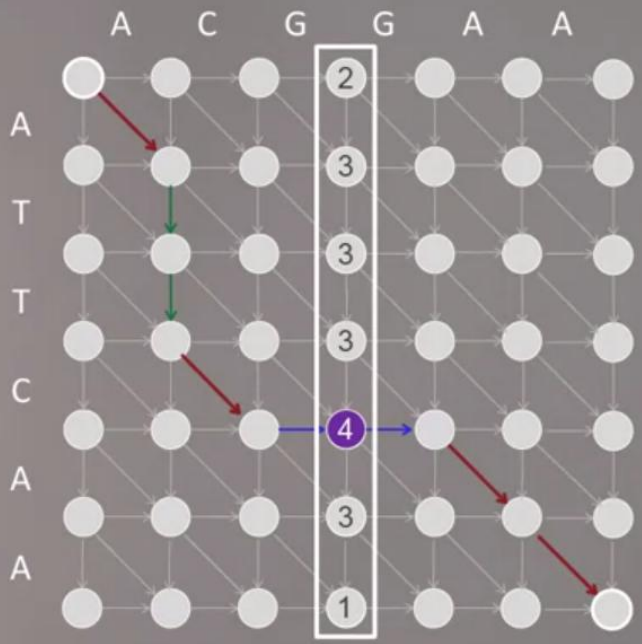

Computing Alignment Score in Linear Space
A C G G A A
A T T C A A
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 1 2 2 2 3 3
0 1 2 2 2 3 4
And we don't need to store all scores of all nodes in our Manhattan grid.
Activate Windows
Go to Settings to activate Windows.


Press Esc to exit full screen
Computing Alignment Score in Linear Space
A C G G A A
A T T C A A
this algorithms, we need to store only two columns.
Activate Windows
Go to Settings to activate Windows.

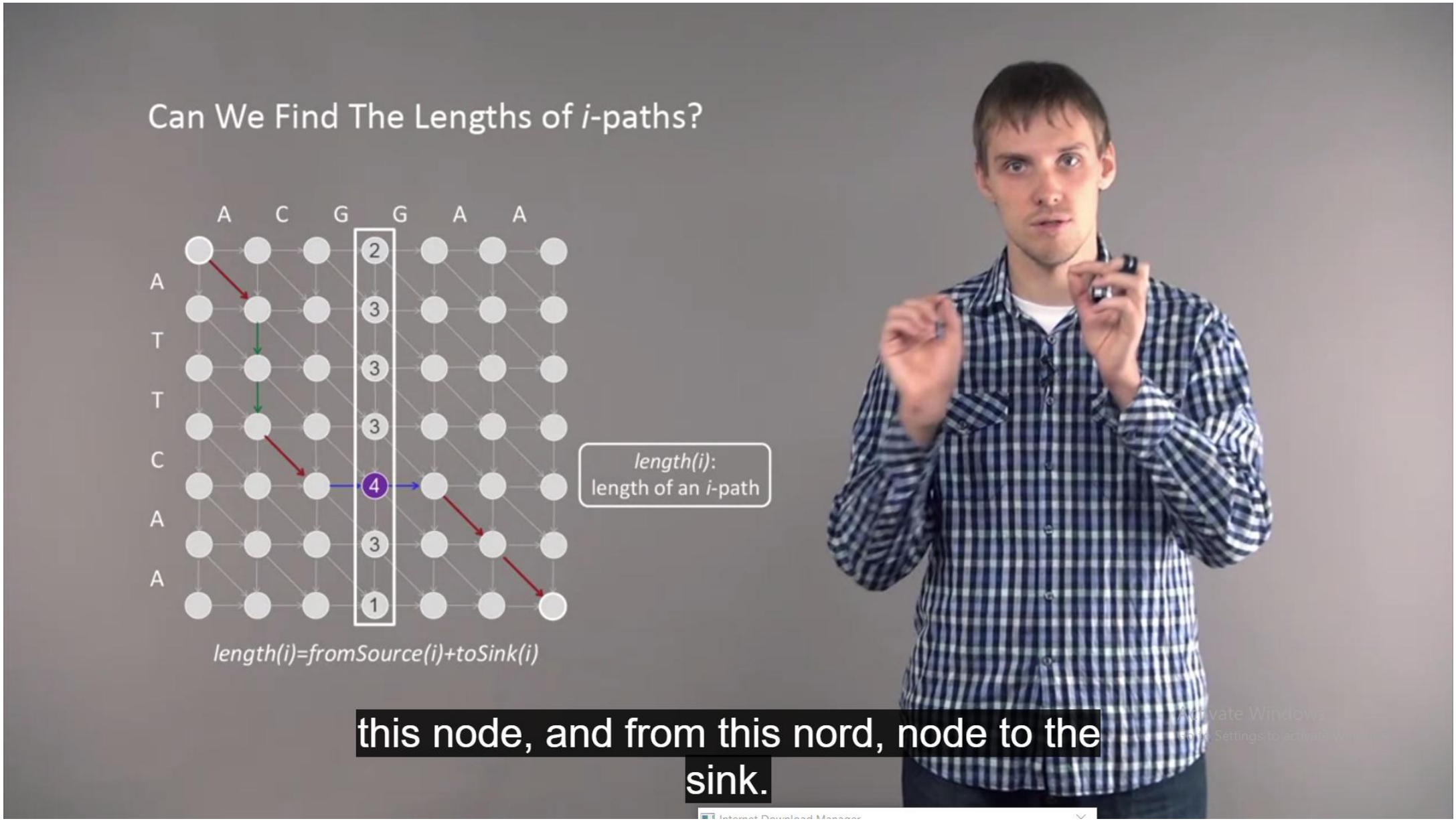

Can We Find the Middle Node without Constructing the Longest Path?
A C G G A A
A
T
T
C
A
A
our middle column and middle node.
So, middle column was a column in the


Can We Find The Lengths of All *i*-paths?
A C G G A A
A T T C A A
2
4
length(i):
length of an *i*-path:
length(0)=2
length(4)=4
is two here, and lengths of four paths is four.

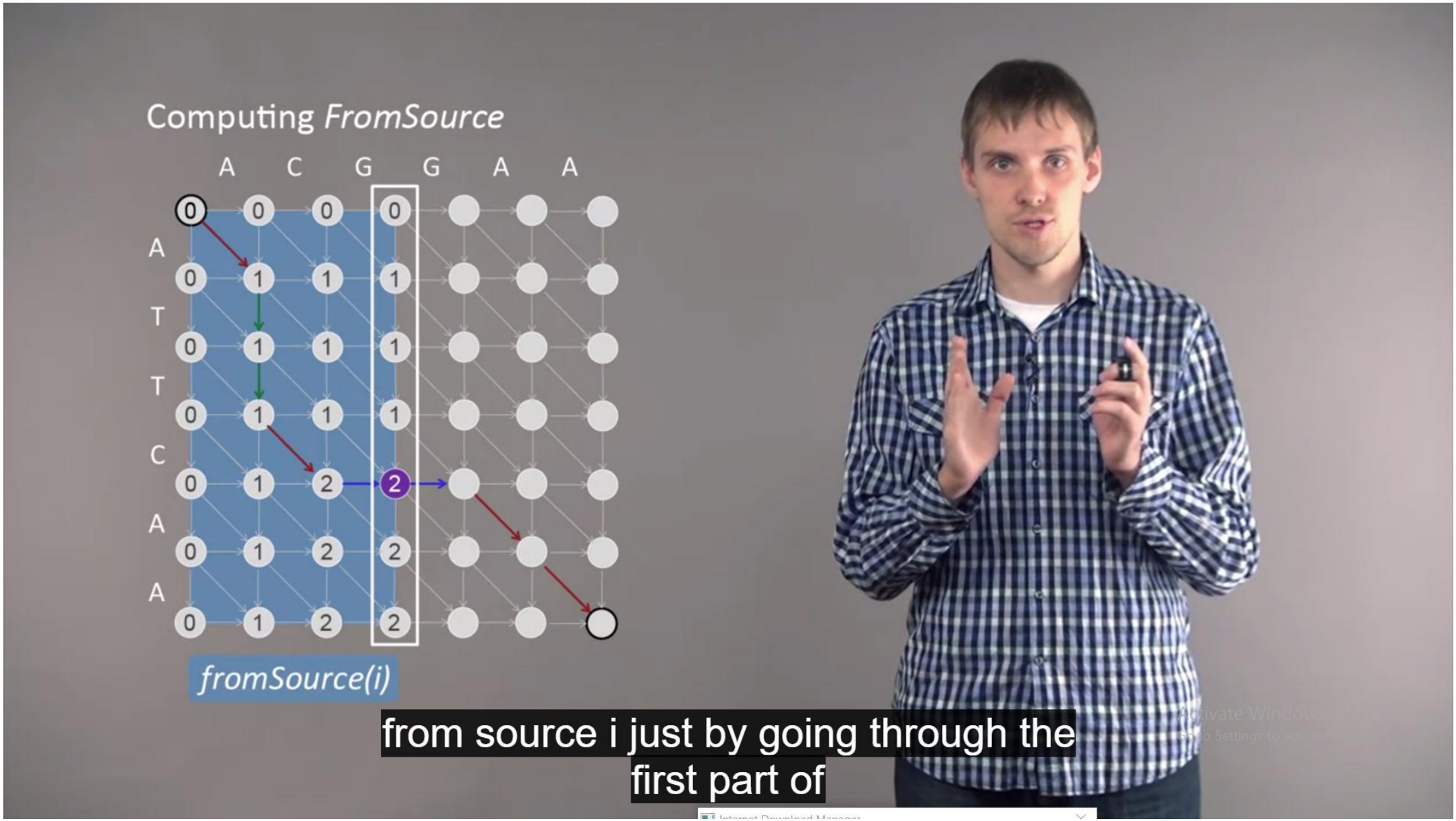
Can We Find the Lengths of All *i*-paths?

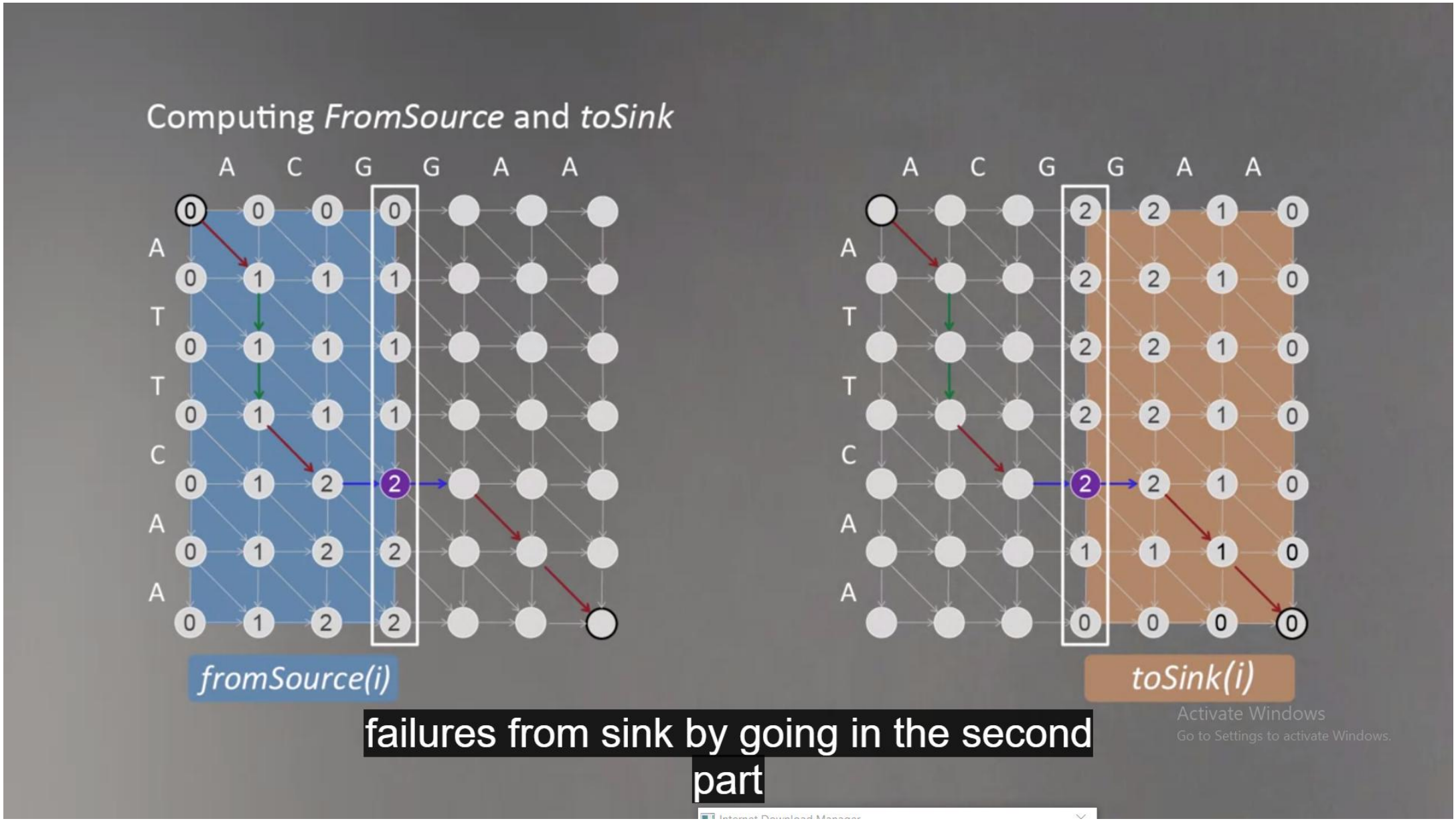


For example here, in this graph, it's two, three, three, four, three, and one.

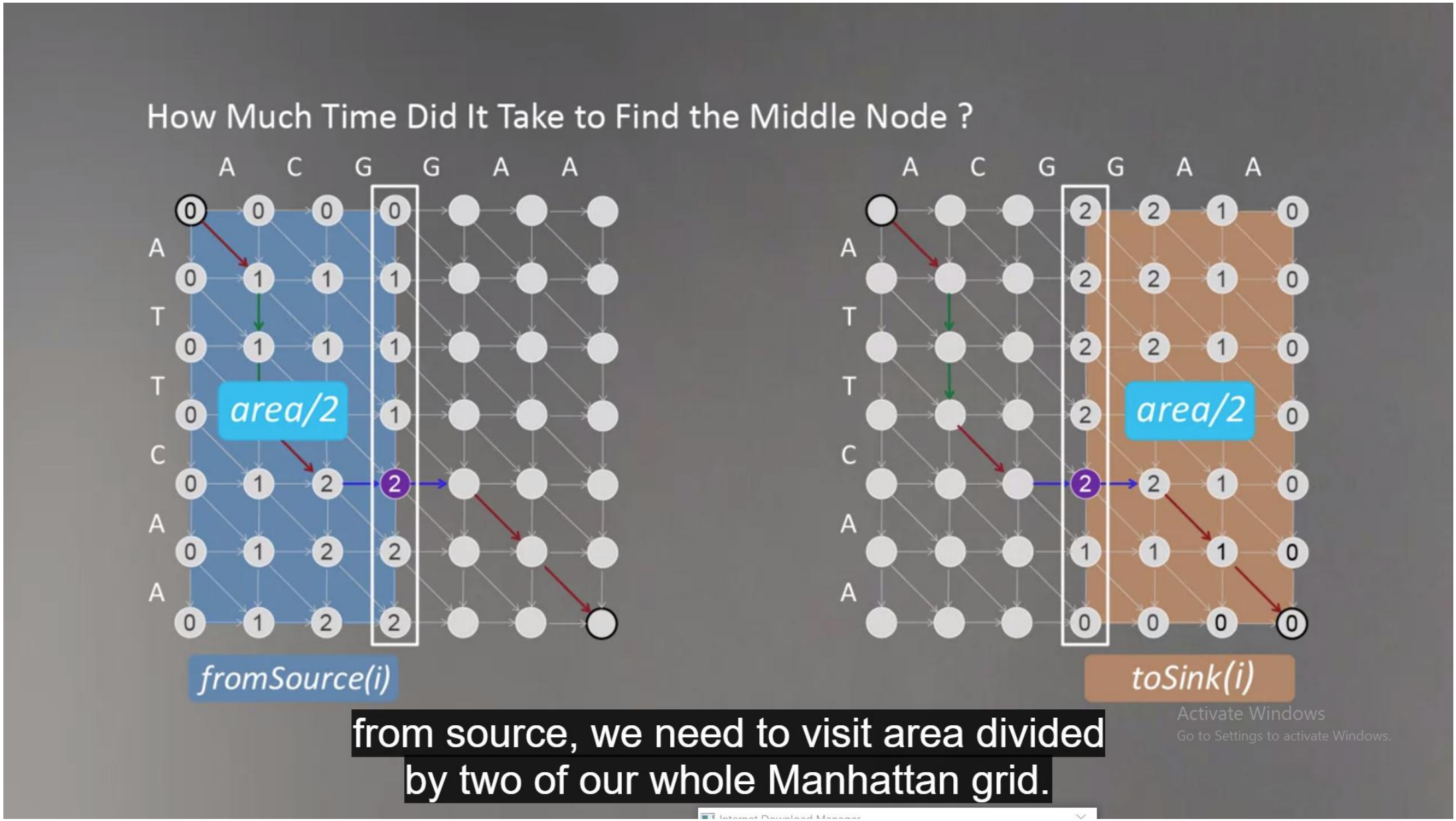

Can We Find The Lengths of *i*-paths?
A C G G A A
A T T C A A
2 3 3 3 4 3 1
length(i): length of an *i*-path
length(i)=fromSource(i)+toSink(i)
this node, and from this nord, node to the sink.


Computing FromSource
A C G G A A
A T T C A A
fromSource(i)
from source i just by going through the
first part of


Computing FromSource
A C G G A A
A T T C A A
fromSource(i)
from source i just by going through the
first part of

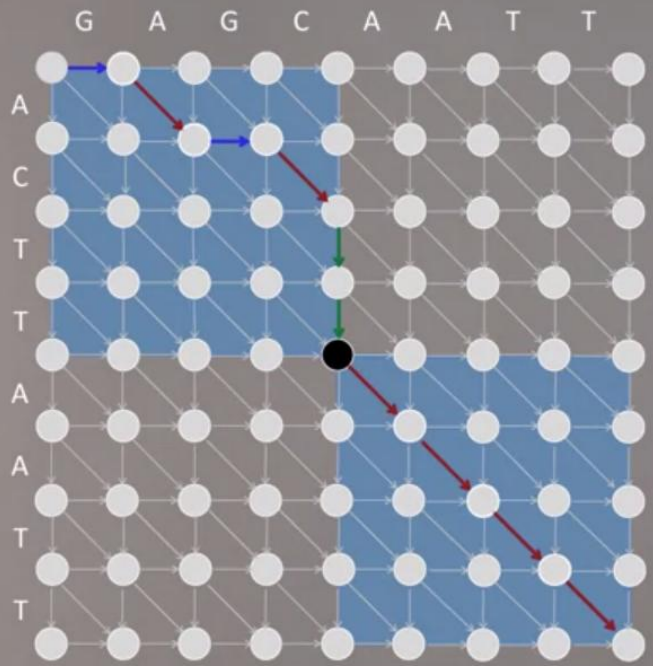


failures from sink by going in the second part


How Much Time Did It Take to Find the Middle Node ?
A C G G A A
A T T C A A
area/2
fromSource(i)
A C G G A A
A T T C A A
area/2
toSink(i)
from source, we need to visit area divided
by two of our whole Manhattan grid.
Activate Windows
Go to Settings to activate Windows.

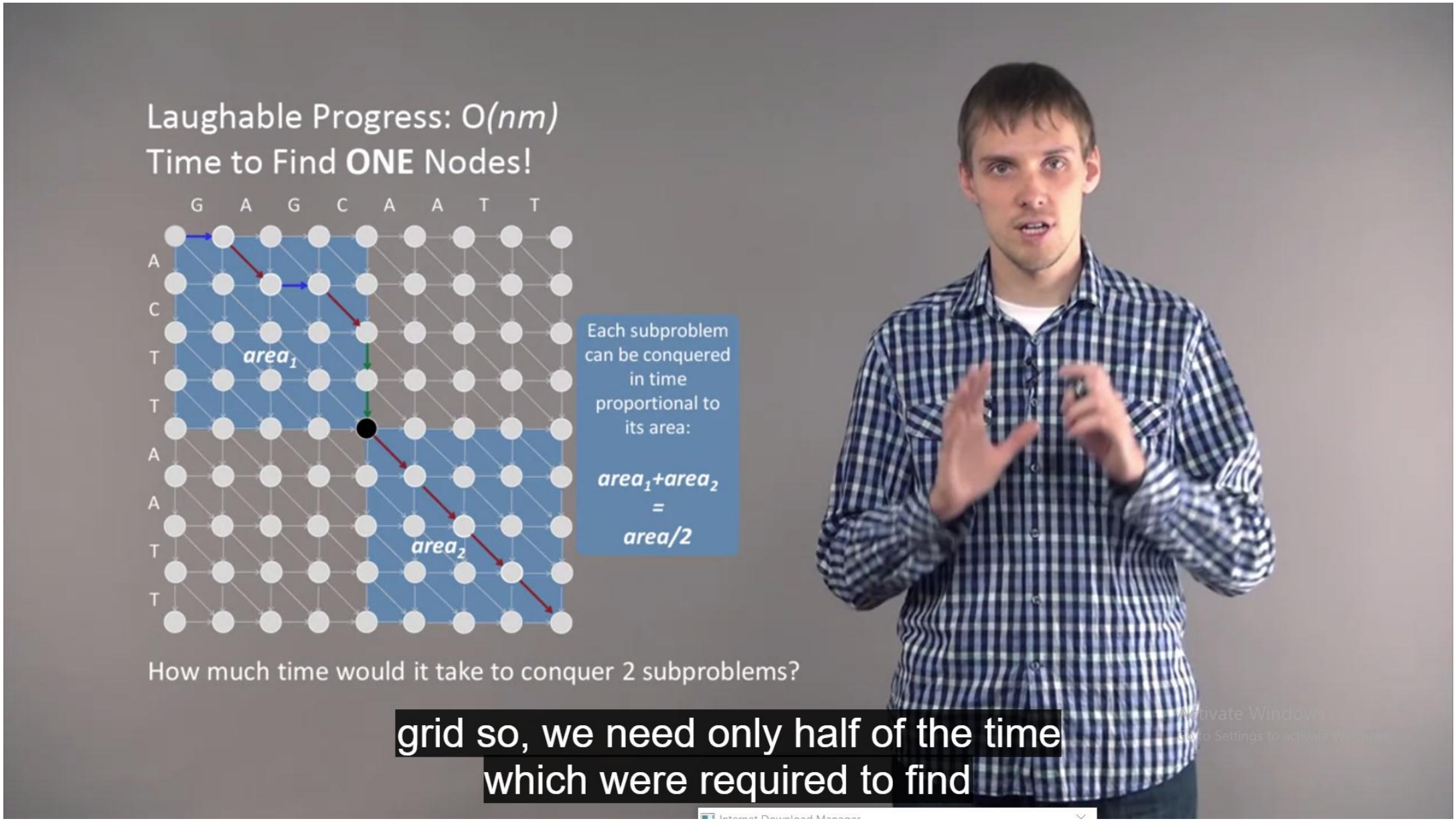

Laughable Progress: O(nm)
Time to Find ONE Nodes!
area/2
area/2+area/2=area
fromSource(i)
Using the same run time as before, we found only middle node.

Laughable Progress: O(*nm*)
Time to Find **ONE** Nodes!

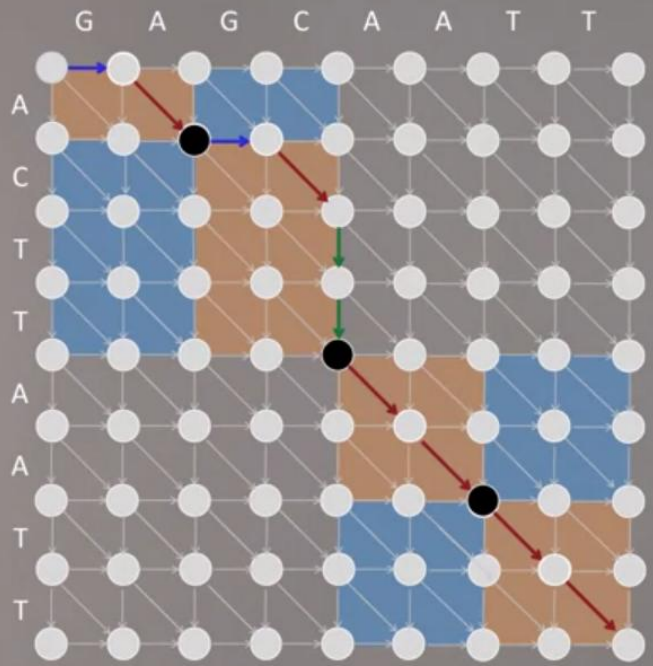


How much time would it take to conquer 2 subproblems?

here should be visited after you've found
middle node, because our


Laughable Progress: O(nm)
Time to Find ONE Nodes!
G A G C A A T T
A C T T A A T T
area1
area2
Each subproblem can be conquered in time proportional to its area:
area1+area2 = area/2
How much time would it take to conquer 2 subproblems?
grid so, we need only half of the time
which were required to find

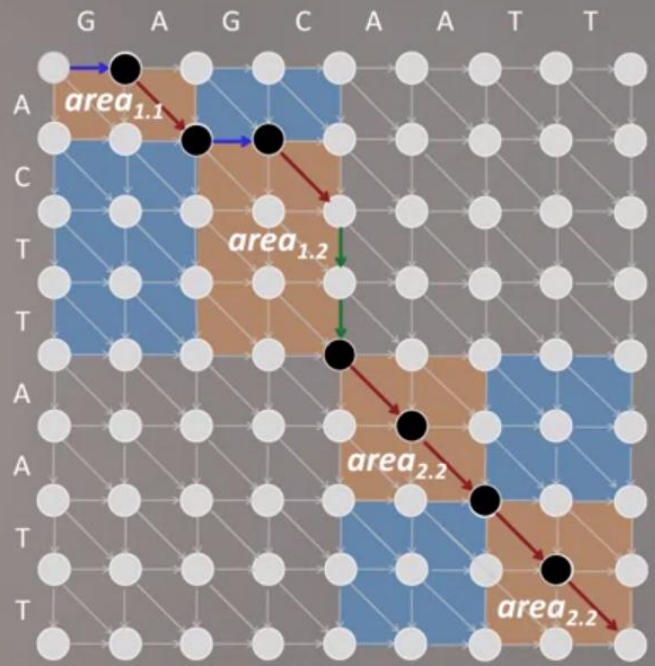
Laughable Progress: O(*nm+nm/2*)
Time to Find **THREE** Nodes!



How much time would it take to conquer 4 subproblems?

step we will split our area into smaller rectangles

Laughable Progress: O(*nm+nm/2*)
Time to Find **THREE** Nodes!



Each subproblem can be conquered in time proportional to its area:

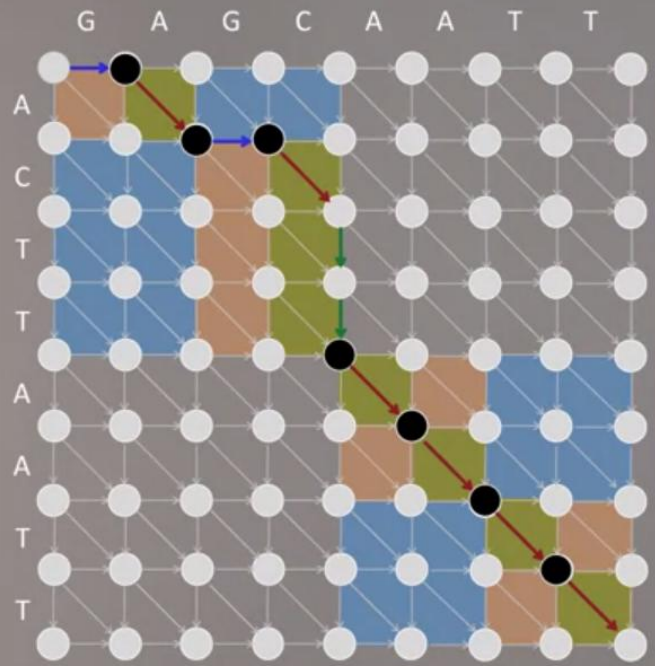
$area_{1.1}+area_{1,2}$
+
$area_{2,1}+area_{2,2}$
=
$area_1/2$
+
$area_2/2$
=
$area/4$

How much time would it take to conquer 4 subproblems?

we need, to take a look on, is
proportional to the

$O(nm+nm/2+nm/4)$

Time to Find **NEARLY ALL** Nodes!



area+
area/2
+area/4
+area/8
+area/16
+....+
<
2·area

How much time would it take to conquer ALL subproblems?

And on the next step we see that area divided by four, and so on and so on.


Total Time:
area+area/2+area/4+area/8+area/16+...
1st pass: 1 area
2nd pass: 1/2
On the second pass we need to take a look on that half of area.


Total Time:
area+area/2+area/4+area/8+area/16+...
1st pass: 1 area
3rd pass: 1/4
4th pass: 1/8
2nd pass: 1/2
It's fourth, and so on and so on.