Next Generation Sequencing Technologies: Data Analysis and Applications

Mapping Algorithms

Dr. Riddhiman Dhar, Department of Biotechnology
Indian Institute of Technology, Kharagpur

Good day everyone. Welcome to the course on next generation sequencing technologies data analysis and applications. In the last couple of classes we discussed some of the methods that could allow us to do read mapping ok. So, we talked about different ways we can do this and we looked at the drawbacks of those methods. So, in this class we will continue our discussions on two different mapping algorithms. So, this class will give you an overview of the process by which we can do the mapping right.

So, these are the algorithms that drive some of the software tools that we will use in the future classes when you do the rate mapping hands on ok. So, it is good to know how these methods work and also we will understand the drawbacks of this method ok. So, this is what we what will be the topic that we will be discussing in this class it is called hash table based mapping algorithm. We will discuss what is hash table and how this mapping algorithm works and then we will take some examples.

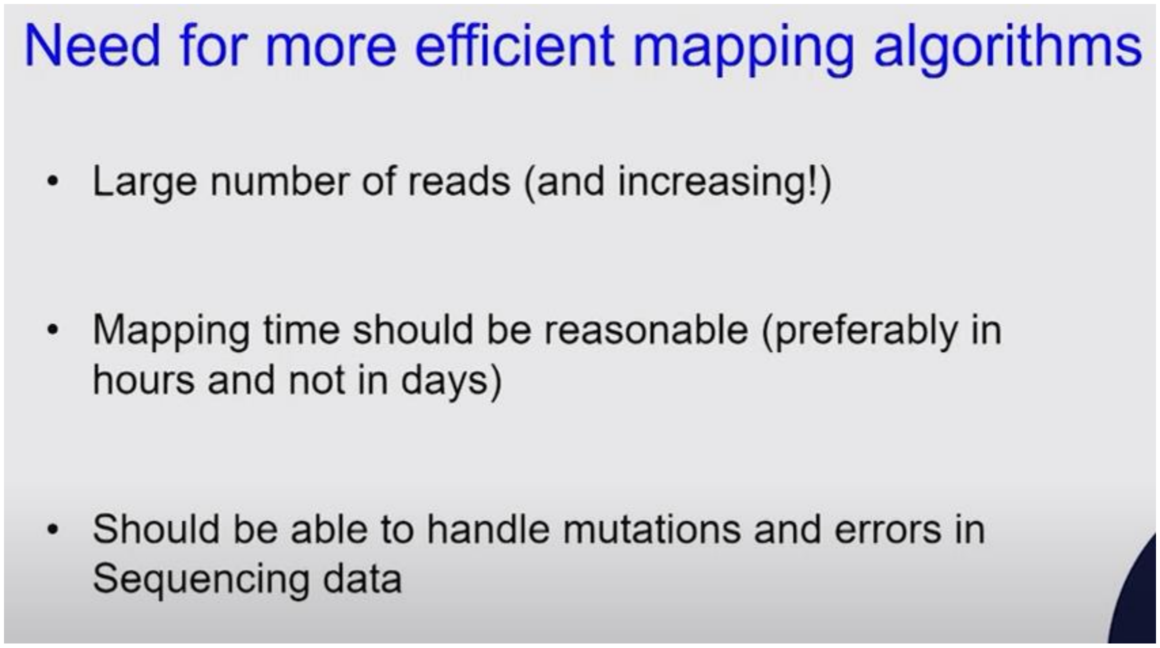
So, that you understand how this process actually operates ok. So, these are the keywords that we will come across. So, seeds you have seen this before in the last class we discussed what are seeds that we generate from the reference sequence or from the reach sequence that is the query. We will come across the term hash and we will talk about the term keys ok. So, just to give you a brief overview of what we have discussed.

So, we talked about the trivial mapping algorithm which is the brute force approach. So, we use a sliding window to map the read against the reference sequence, but as you can imagine this is a very slow process and we calculated some time right based on certain assumptions and it appeared to be very very slow right. So, this is not fit for this purpose. We then talked about the blast algorithm and we talked about the method by which it works it uses something called seed and extend. So, we generate seed sequences from the read or the query data and search against the reference.

It is still slow because this process requires this linear search in the reference sequence. We talked about an alternative to blast it is called BLAT. So, it is blast like alignment tool it utilizes something called indexing of the reference sequence. So, the seeds are generated from the reference sequence instead of the query or the read and we found out this is faster compared to blast there are also we looked at the data from the paper it was significantly faster than blast, but it is still not fast enough for our purpose and the purpose is to map millions or billions of reads right in a short period of time ok. So, the search for better algorithms continue right.

So, this is something what we will be looking at right. So, we need more efficient algorithms. So, by efficient we mean it should be able to process a very large number of reads and this number is also going up as the technology progresses right this is crossing from millions to billions now right. So, if you are processing multiple samples you will probably be handling billions of reads. So, the mapping time for these billions of reads should be reasonable right.

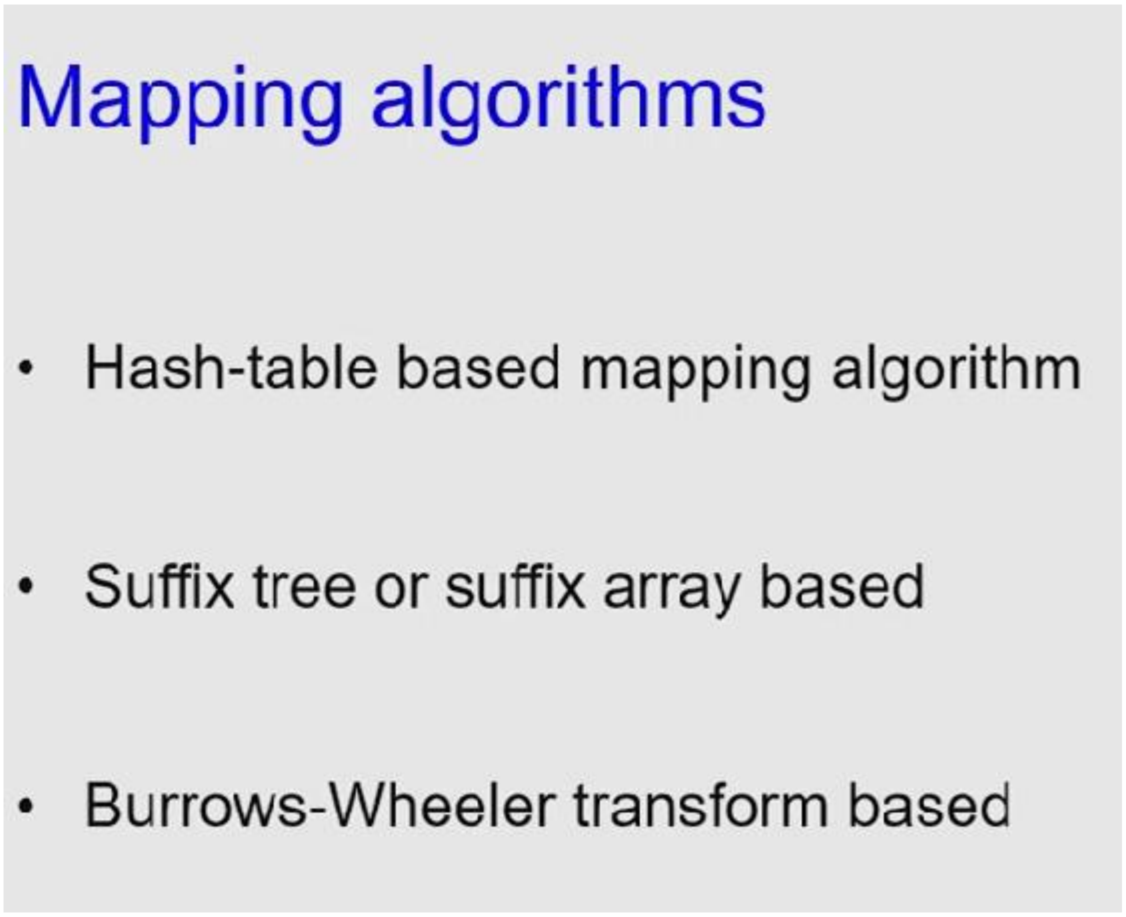
So, preferably in hours and not in days especially not in tens of days right maybe couple of days 3 days for billions of reads that is probably we what we can leave with and it should be able to handle mutations and errors in the sequencing data right. So, this is something that is very important it is and it compared to other kind of applications where we have perfect matches right. So, if you are looking for a pattern in a text or something we are mostly looking for perfect matches, but here when we are generating sequencing data we have genuine mutations right these are biological variations that you observe in our samples and also errors in the sequencing data right. So, we talked about why the errors come and how these are generated for different technology platforms right. So, these errors should also be taken into consideration right.



Need for more efficient mapping algorithms

- Large number of reads (and increasing!)
- Mapping time should be reasonable (preferably in hours and not in days)
- Should be able to handle mutations and errors in Sequencing data

So, the algorithm should not should not be able to perform just matching perfect data right it should also be able to get map these imperfections right it should be also be able to handle these imperfections ok. So, there are 3 types of algorithms broadly that have been developed to do this kind of task and we will actually discuss all of these in brief of course, we will get an overview not into the detailed mathematical discussion or the implementation at the coding level right how these are implemented we will talk about the algorithm, but if you if you are familiar with programming you can actually implement these methods yourself ok. So, the first one is called hash table based mapping algorithm which will be the topic of discussion today. The next one is called suffix free based method or suffix array based method. So, there are 2 alternative ways of doing this and we will discuss them in the next class.

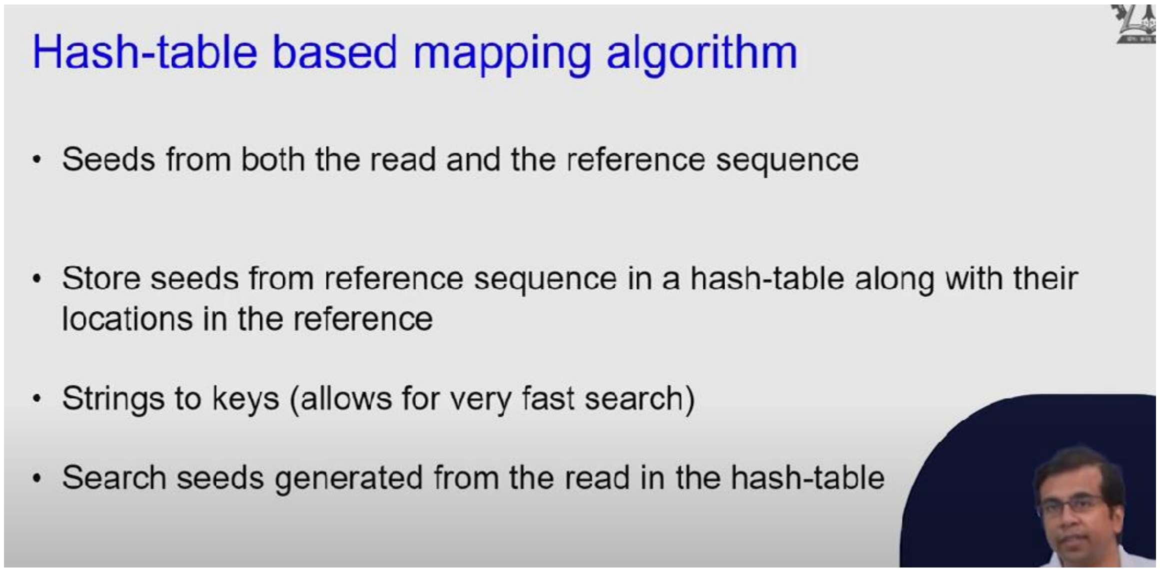

Mapping algorithms

- Hash-table based mapping algorithm
- Suffix tree or suffix array based
- Burrows-Wheeler transform based

We will talk about the advantages of each of these methods also and then we will finally, we will talk about the Barrows-Wheeler transform based algorithm ok. So, this will be our final algorithm for the mapping ok. So, let us begin we will start with the hash table based mapping algorithm ok. So, so how does it differ from whatever we have discussed so far the blast and BLAT ok. So, the major idea in hash table based mapping algorithm is that we build seeds from both the read and the reference sequence ok.

So, in compared to blast where we build seeds only from the read right or compared to BLAT where we read where we build the seeds from the reference sequence right here we build seeds from both the read and the reference sequence right. So, what are the seeds you remember right these are the substrings of certain length right. So, we can choose the k value right the length of that length of the substrings of the seeds ok. So, once you have built the seeds right for the reference sequence we store them in a hash table and along with their locations in the reference ok. So, we will take an example to understand how this is done and I will also describe what is a hash table ok.

Now, what this hash table does it actually converts these strings or the substrings to something called keys ok and for each key there will be a value so that value would be the location. Now, the moment you do this so this is a data structure the moment you do this it allows for very fast search ok. You can very quickly look up the hash table and find out the location of that seed ok. And once we have stored this reference seeds into the hash table we can generate seeds from the read and we can search against the hash table right. So, what we are searching we are searching the seeds which are stored as keys right and if there is a match in the key right we will get a value and that value will give the location of that seed ok.

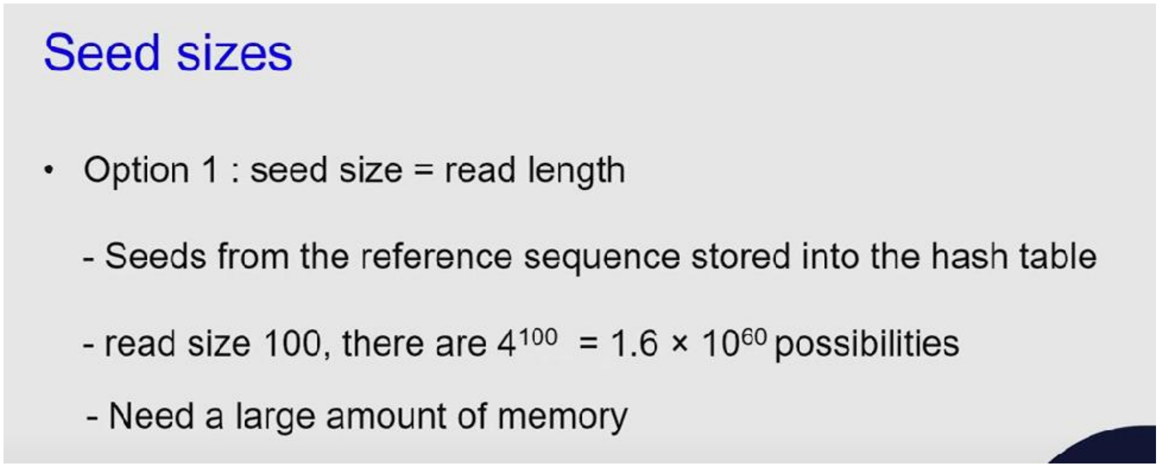


Hash-table based mapping algorithm

- Seeds from both the read and the reference sequence
- Store seeds from reference sequence in a hash-table along with their locations in the reference
- Strings to keys (allows for very fast search)
- Search seeds generated from the read in the hash-table

So, this is kind of the brief idea, but we will elaborate this further with more examples ok. So, there are various options right when you are trying to design the algorithm you can think about different options and how you want to proceed ok. So, the first option is a seed size equals to read length. So, we say ok so we choose the seed size as the read length. So, if you have read length of 50 base pair or 100 base pair that would be our seed size ok.

So, if that is the case what we would have to do is we then break down the reference sequence right to generate seeds of that size if it is 100 bases which generate seeds of length 100 right and then store them into the hash table. So, if you have read size 100 so there are about 4 to the power 100 possibilities right. Why is that so? Because for each position you can have either A, B, G or C right. So, if you consider all possible combinations for this 100 position this will be 4 to the power 100. So, it is it comes out around 1.

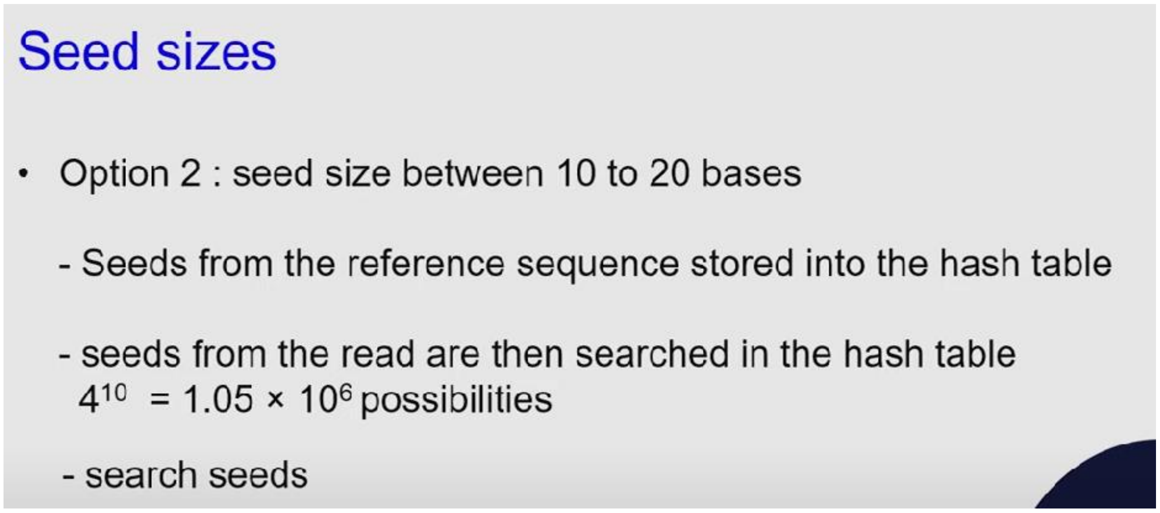


Seed sizes

- Option 1 : seed size = read length
  - Seeds from the reference sequence stored into the hash table
  - read size 100, there are $4^{100} = 1.6 \times 10^{60}$ possibilities
  - Need a large amount of memory

6 into 10 to the power 60 possibilities right. So, this is a really huge number. So, of course, in a genome sequence perhaps you will not have so many of them right you probably will not get all of these combinations, but at least you will get a large fraction of this right. So, even if it is 10 to the power 20 or so that that is still a huge number to store right. So, if you want to store this in the memory in the hash table right you will need a huge amount of memory ok.

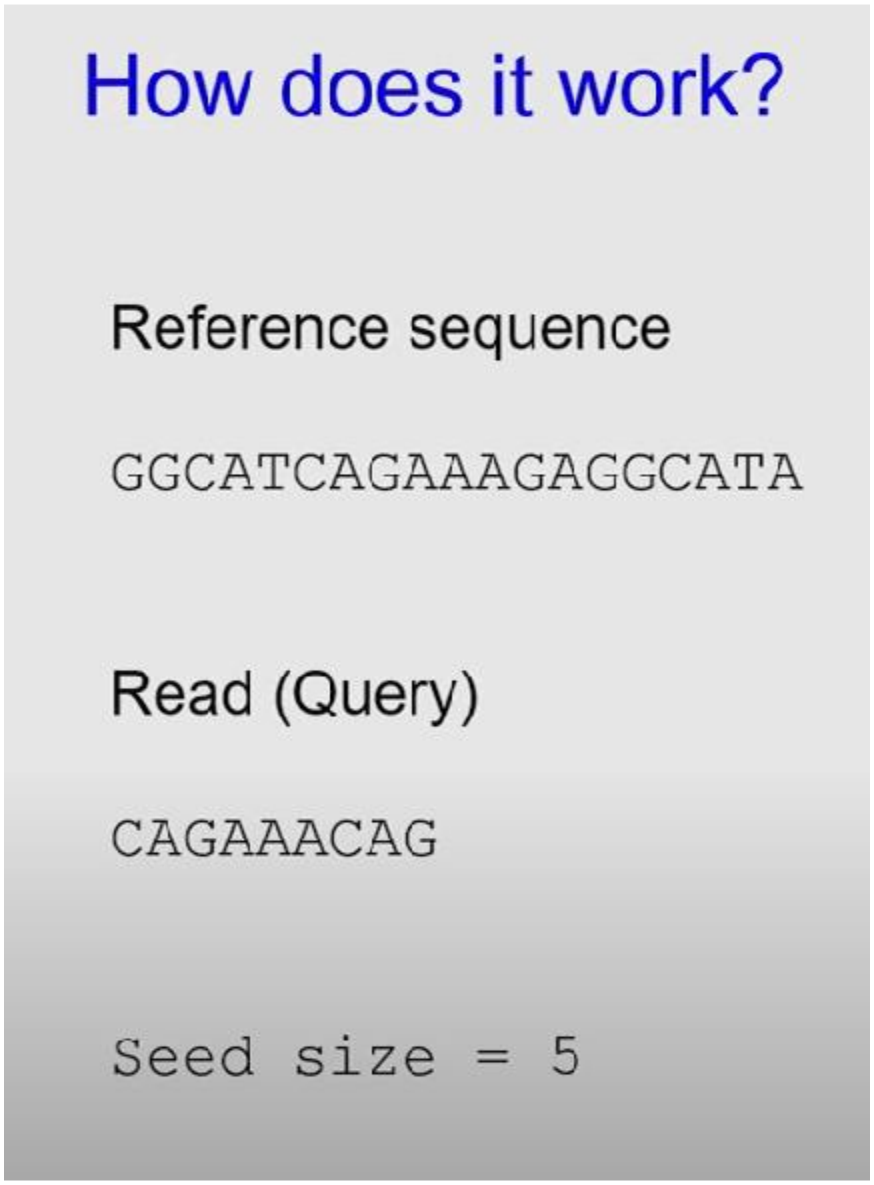
So, this is one of the biggest challenges right. So, you probably will not be implement this in a desktop computer or even a normal server probably you need connection like a compendium of servers or maybe a supercomputer right. So, that option is not a feasible option right. So, what what is the feasible option? The option 2 is probably look at seed size that between that is between 10 to 20 bases ok. So, again this can vary depending on the read length that you work with and researchers have experimented with different read lengths and have seen what was based for whatever read size you are working with ok.



Seed sizes

- Option 2 : seed size between 10 to 20 bases
  - Seeds from the reference sequence stored into the hash table
  - seeds from the read are then searched in the hash table
    $4^{10} = 1.05 \times 10^{6}$ possibilities
  - search seeds

So, again we extract the seeds or generate seeds from the reference sequence and stored in the hash table and if we have seed size of 10 right. So, this is about 4 to the power 10 possibilities. So, which is close to a million possibilities ok. So, this is quite manageable compared to the earlier scenario. And once we have done that we generate these seeds from the read sequence and then search against this hash table right.



How does it work?

Reference sequence

GGCATCAGAAAGAGGCATA

Read (Query)

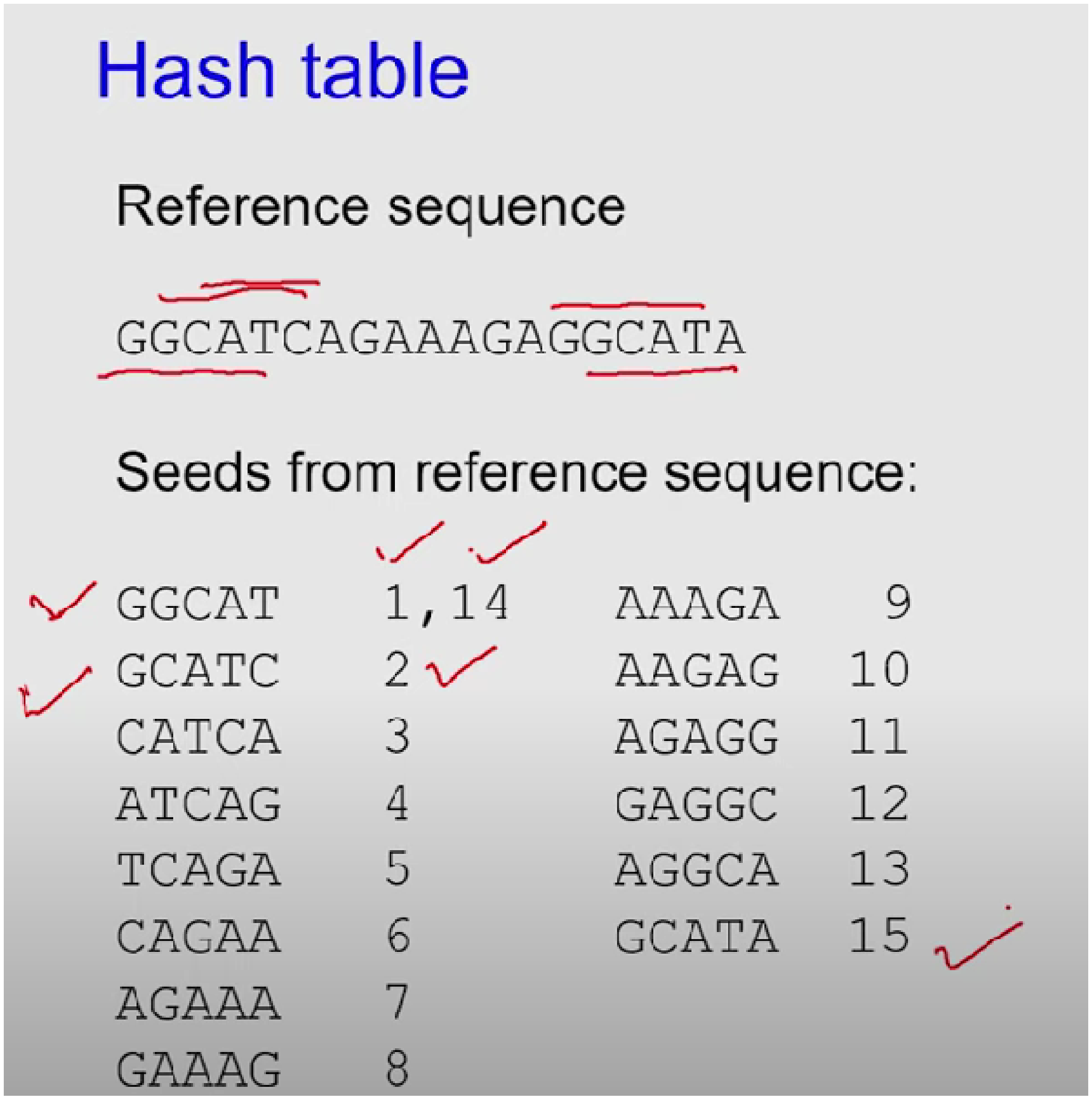
CAGAAACAG

Seed size = 5

We have millions of million possibilities and among them we have to match right from the seed ok. So, let us look at how it works right and we will take some examples right. So, let us take this example where we have this reference sequence this is an arbitrary example and we are working with small reference sequence are read. So, that it is manageable right we can complete this example here within this slide and within a reasonable amount of time ok. The process actually is the same it is just will take more time more memory and of course, more elaborate right.

So, here is the reference and then you have the read sequence which is the query ok and seed size we choose as 5 ok. You can choose any other seed size does not really matter ok. So, here are the seeds that we have extracted from the reference sequence along with their positions ok. So, if you look carefully right what we are doing the size is 5 right. So, we look at first 5 right we take first 5 letters from the reference ok and we build this seed ok and this is the first key ok.

And here we have the location. So, the location is 1 this is the first position where the seed is present. So, we add this in the value ok. Then we move on the next one right g c a t c right and we add the we add that substring to the hash table we have the position location which is 2 ok. And then we go on right we slowly move on along this reference sequence and finally, we see this one here g g c a t and this one was already present right. So, what we do is we just add the value add an additional value for that key right.



Hash table

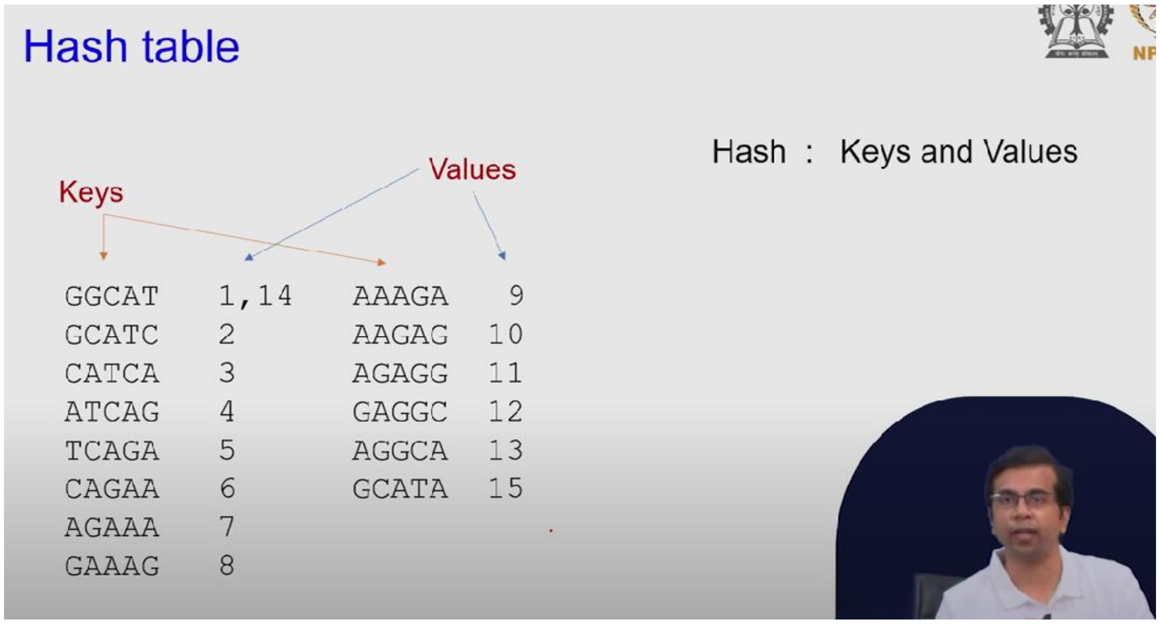
Reference sequence

GGCATCAGAAAGAGGCATA

Seeds from reference sequence:

GGCAT	1,14	AAAGA	9
GCATC	2	AAGAG	10
CATCA	3	AGAGG	11
ATCAG	4	GAGGC	12
TCAGA	5	AGGCA	13
CAGAA	6	GCATA	15
AGAAA	7		
GAAAG	8		

So, what it means this key this seed is present in two positions right it is repeated in the reference sequence ok. So, we need to store this alternative locations and then finally, we have the final one here you see ATA which gives us this here right. So, this is the hash table that we build as we go along the reference sequence ok. Now, we will utilize this hash table now to actually map our read ok the read that we have taken. For the read ok so, here is the hash table right as I am mentioned these are the keys I mentioned and these are the values ok.

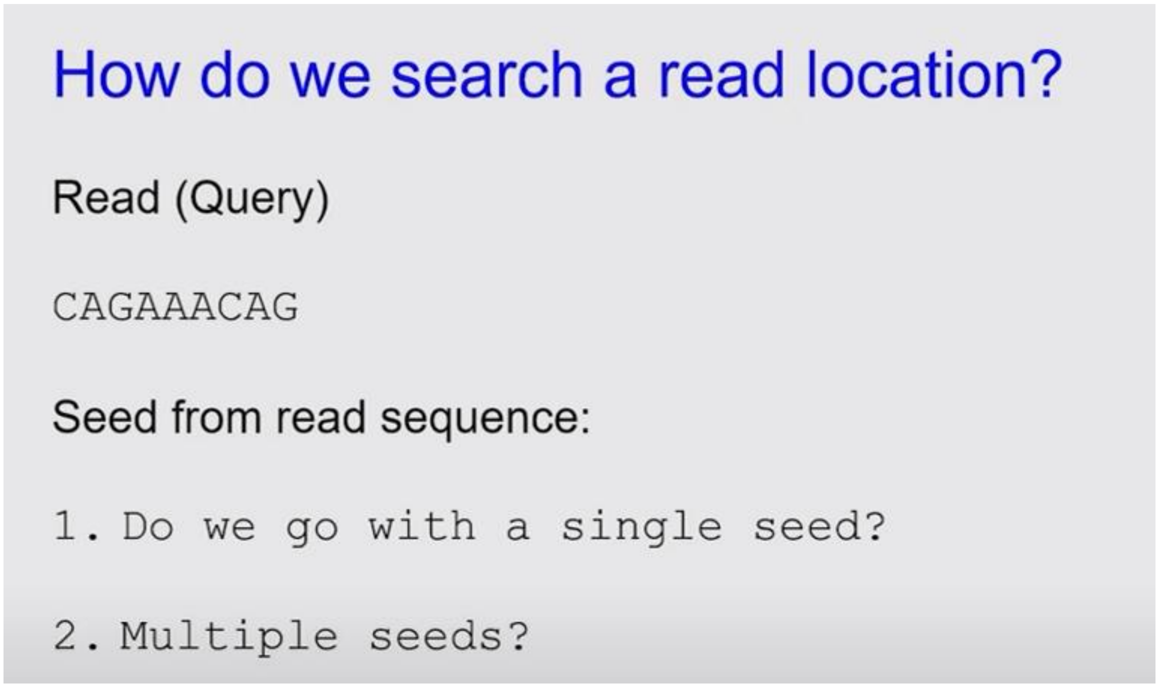


Hash table

Hash : Keys and Values

GGCAT	1,14	AAAGA	9
GCATC	2	AAGAG	10
CATCA	3	AGAGG	11
ATCAG	4	GAGGC	12
TCAGA	5	AGGCA	13
CAGAA	6	GCATA	15
AGAAA	7		
GAAAG	8		

So, these keys if we can find these keys right we can get the values we can retrieve the values by just calling the keys ok. So, this is a specific data structure which we call as hash and because we have many of them we call them as hash table ok. So, now, how do we search a read location right using this hash table we have built. Now, once we have built this for a specific reference genome you can store this in a file right you do not have to ah rebuild this hash table again and again for the same genome right. So, let us say we have we have built the hash table for human genome human reference genome and we can store this in a file right what are the locations what are the keys that we have for a specific seed size ok, but if the seed size changes then of course, you will have to rebuild the ah the hash table ok.



How do we search a read location?

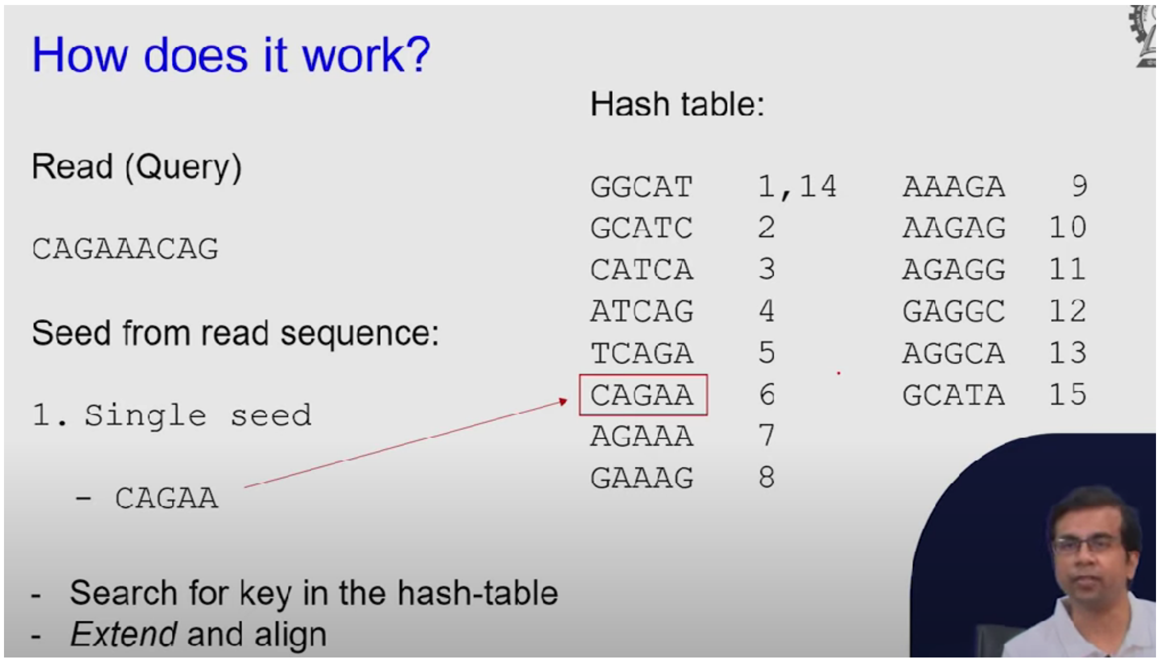
Read (Query)

CAGAAACAG

Seed from read sequence:

1. Do we go with a single seed?
2. Multiple seeds?

Now, this read ah that we have now we actually have to generate these seeds again from the read sequence. Now, again we have two options we can say we generate a single seed and try to find that out ah in the hash table right and find its location or we can go with multiple seeds ok. So, let us look at what actually would happen what are the advantages or drawbacks of each of these right. So, if you are looking up a single seed we have generated the single seed again the k is 5 right seed length. So, C A G A we have ah generated the seeds its here right single seed and we then look for this ah seed in the hash table and here it is right we have this C A G A and we get the value right so, which is 6 ok.



How does it work?

Read (Query)

CAGAAACAG

Seed from read sequence:

1. Single seed

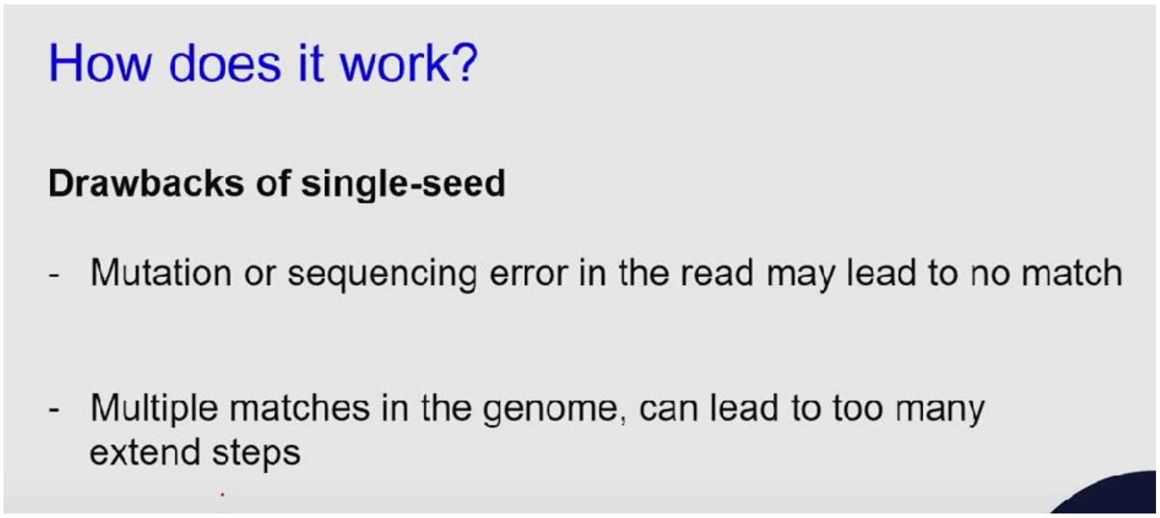
   - CAGAA

- Search for key in the hash-table
- *Extend* and align

Hash table:

GGCAT	1,14	AAAGA	9
GCATC	2	AAGAG	10
CATCA	3	AGAGG	11
ATCAG	4	GAGGC	12
TCAGA	5	AGGCA	13
CAGAA	6	GCATA	15
AGAAA	7		
GAAAG	8		

So, this is in the 6 position of the reference sequence we have the seed present. So, what it means is that these read starts at the 6th position right. So, that information is now available to us after we have managed to search the seed in the hash table. Now, once we have done that we can simply extend and align, but we can use an alignment algorithm to see what are the matches, mismatches etcetera and get the final results ok. So, that that sounds quite easy and this is really straightforward right and its also ah does not it does not take too much time right.

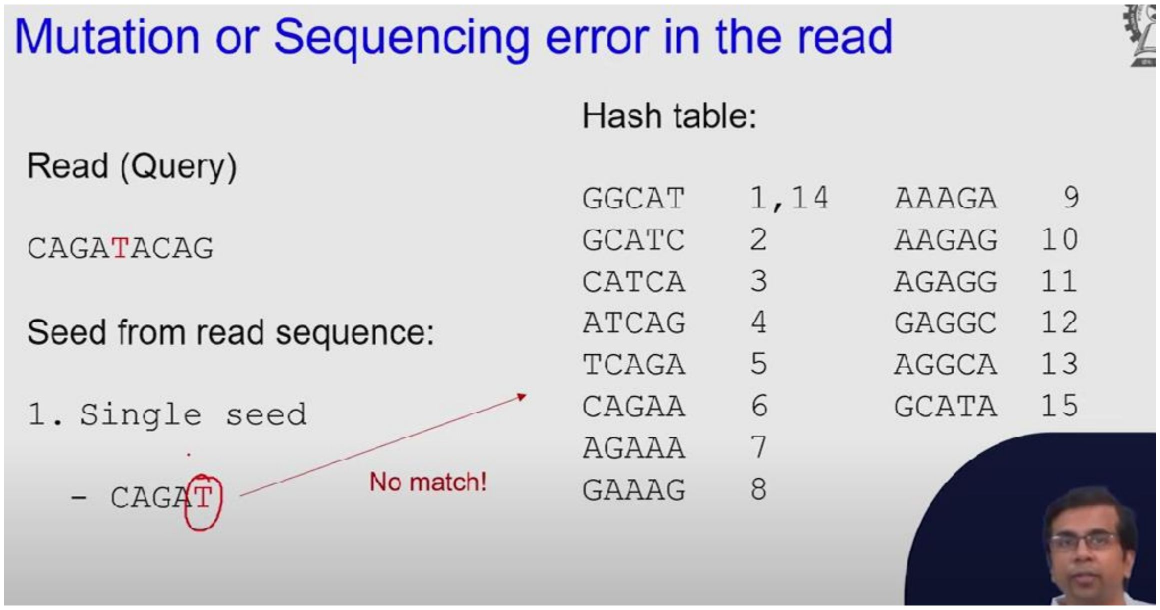


How does it work?

Drawbacks of single-seed

- Mutation or sequencing error in the read may lead to no match
- Multiple matches in the genome, can lead to too many extend steps

So, you just search for the single one it is a it is an incredibly fast ah process if you searching through an hash table. Now, what what what would be the major drawback why do not you use this ah most of the time is because if you have a mutation or sequencing error in the read this may lead to no match ok. So, you may not be able to find that seed in the hash table. So, imagine this example right instead of this ah A here you have a T right in red this is the mutation that we have in the read or sequencing error in the read and when you compare against the hash table of the reference sequence we do not see any match ok. So, if you do not see any match right we cannot look at the read we can say ok that we do not find the read in the reference genome and we discard it right that is what, but that would be wrong right just because of a single mutation if you discard that read you cannot map that read that that means, you are losing data ok.



Mutation or Sequencing error in the read

Read (Query)

CAGATACAG

Seed from read sequence:

1. Single seed

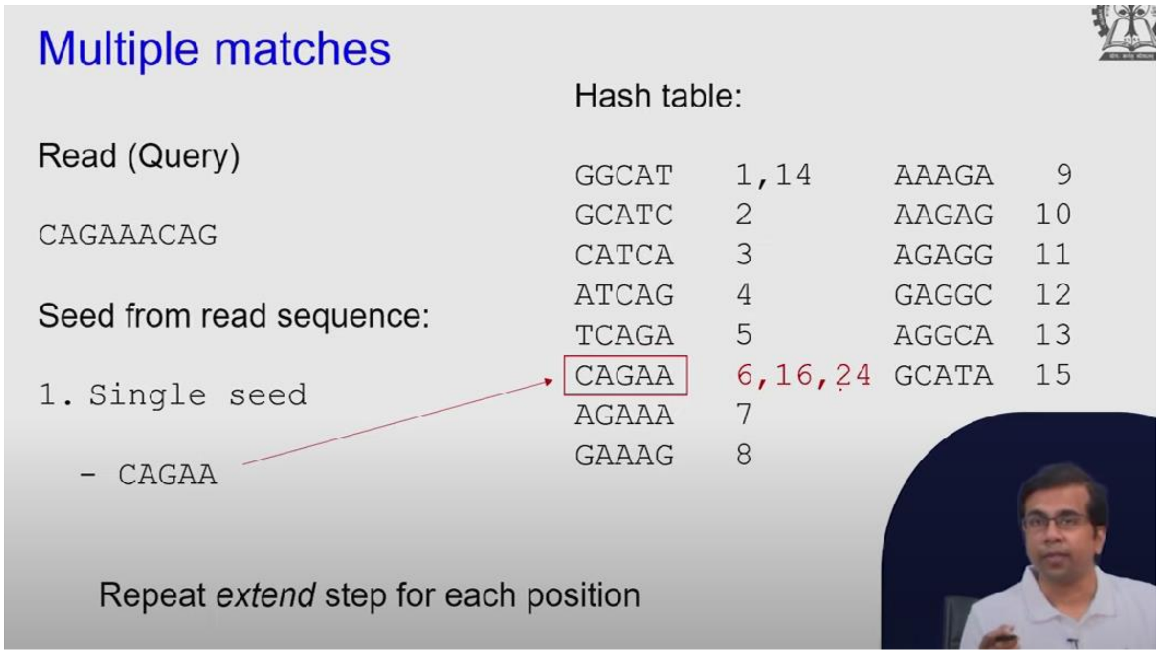
 - CAGAT

No match!

Hash table:

GGCAT	1,14	AAAGA	9
GCATC	2	AAGAG	10
CATCA	3	AGAGG	11
ATCAG	4	GAGGC	12
TCAGA	5	AGGCA	13
CAGAA	6	GCATA	15
AGAAA	7		
GAAAG	8		

So, this is something that we want to avoid. So, probably better way of doing this would be actually going from multiple seeds there is another disadvantage right. So, if a single seed it matches to multiple positions in the genome right ah we have seen it can right if you have for example, this seed here it can match to 1 and 14 right 2 positions you have to now see align this for at both of these positions right you have to do this extend and align for both this position and see which one actually aligns better right. Again the alignment part is a time consuming process and if you have to do this for more than 2 or 5 or 7 right that will actually take up time.



Multiple matches

Read (Query)

CAGAAACAG

Seed from read sequence:

1. Single seed

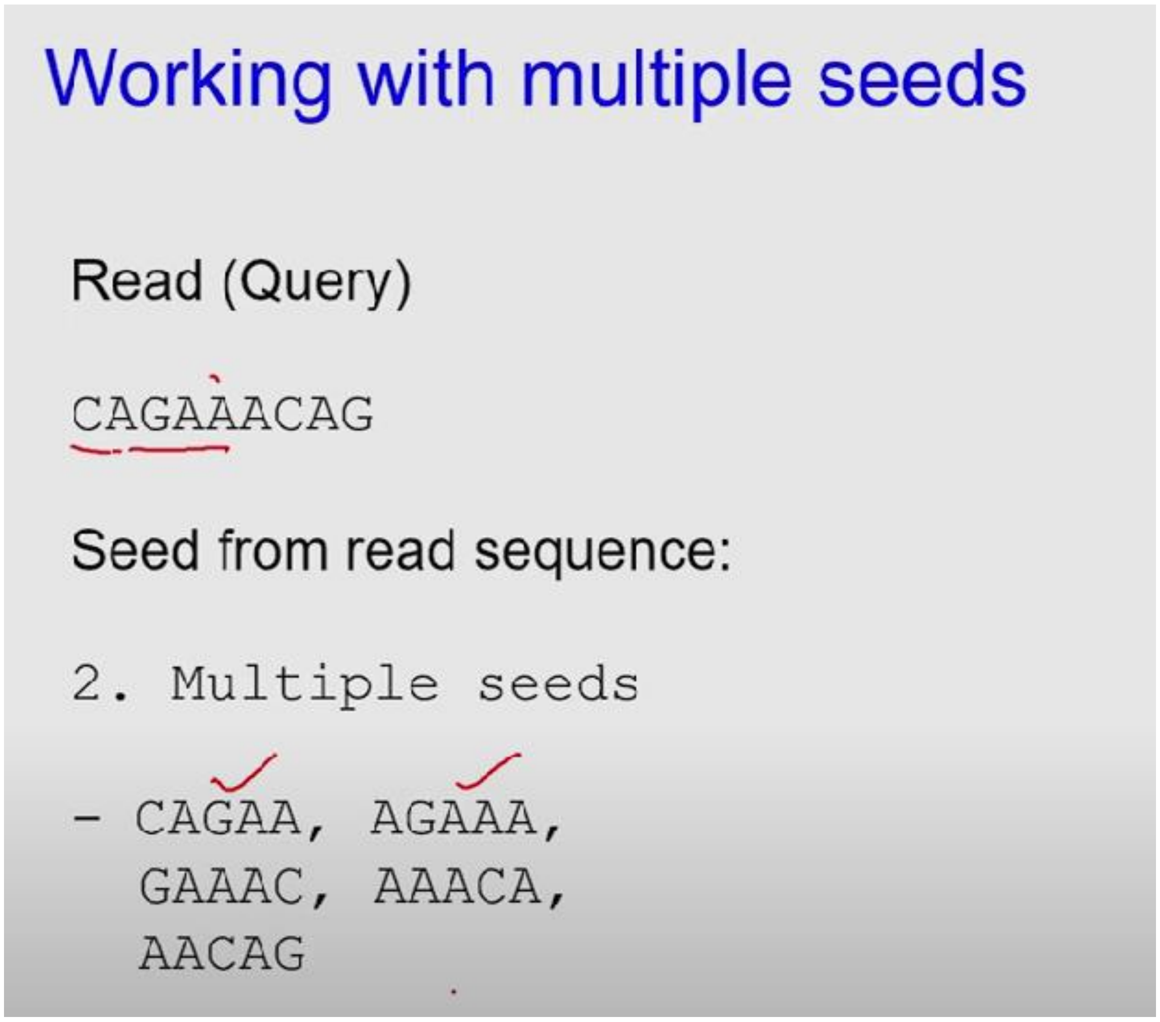
   - CAGAA

Hash table:

GGCAT	1,14	AAAGA	9
GCATC	2	AAGAG	10
CATCA	3	AGAGG	11
ATCAG	4	GAGGC	12
TCAGA	5	AGGCA	13
CAGAA	6,16,24	GCATA	15
AGAAA	7		
GAAAG	8		

Repeat *extend* step for each position

Now, what you can do is ok. So, here is this right. So, again if you imagine instead of 6 you have 16 24 right for 3 positions here you have this match for the seed ah then you have to do this extend process ah extend and align process for all 3 positions ok. So, this takes time right. So, the probably this is better to it is better to work with multiple seeds ok. So, what what we mean by multiple seeds is that again for the same seed length we have this CAGGAA right we are generating all possible seeds here right we just slide in this window again and we get these 5 seeds ok.



Working with multiple seeds

Read (Query)

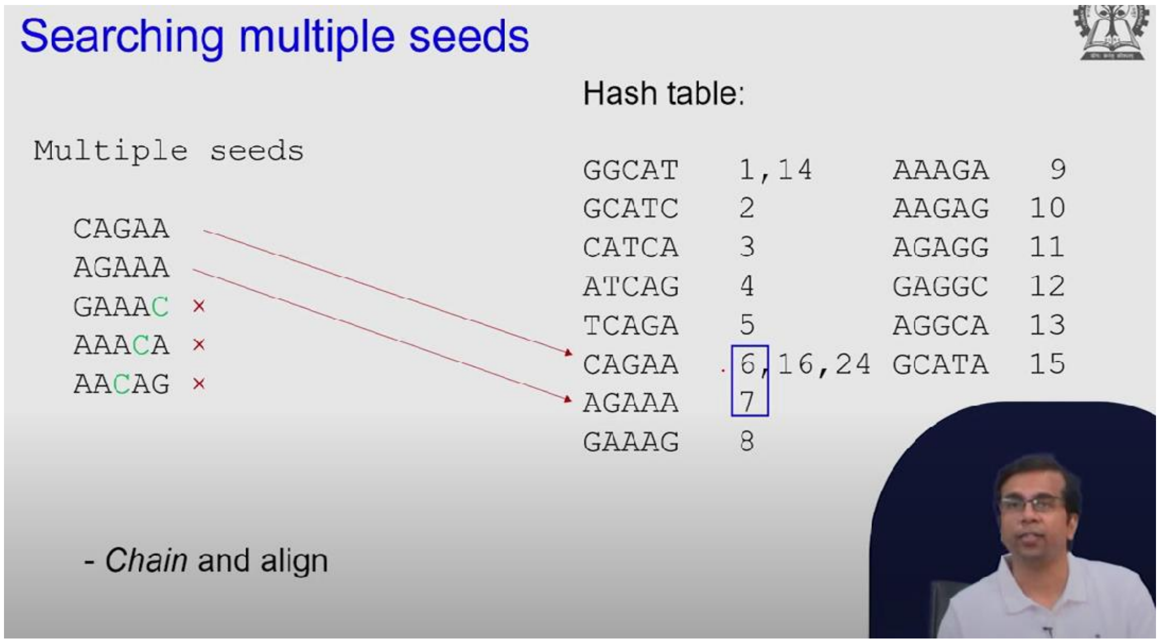
CAGAAACAG

Seed from read sequence:

2. Multiple seeds

- CAGAA, AGAAA,
GAAAC, AAACA,
AACAG

Now, what we will do now is we will search each of these seeds for ah for their positions in the hash table ok and let us do that right and what we see is that this seed actually matches with this one here right this seed here the second one matches with this one here right, but the rest of them they do not match there is no match and I have highlighted this C base here because this is a mutation ok this is a change ah that is in the region could be a mutation or a sequencing error right. So, these 3 reads ah are not matching, but the 2 these 2 the first 2 have matched ok and the first one is matching let us say to this 3 positions ok. So, there are 3 values for this first match. Now, in the earlier method you would have to extend for all 3 right, but in this method we can see the first one matches at position 6 the next one matches to a single position position 7 ok. So, this is closest right this is close by and there is no other option for the second match right.



Searching multiple seeds

Multiple seeds

CAGAA
AGAAA
GAAAC ×
AAACA ×
AACAG ×

Hash table:

GGCAT	1,14	AAAGA	9
GCATC	2	AAGAG	10
CATCA	3	AGAGG	11
ATCAG	4	GAGGC	12
TCAGA	5	AGGCA	13
CAGAA	6,16,24	GCATA	15
AGAAA	7		
GAAAG	8		

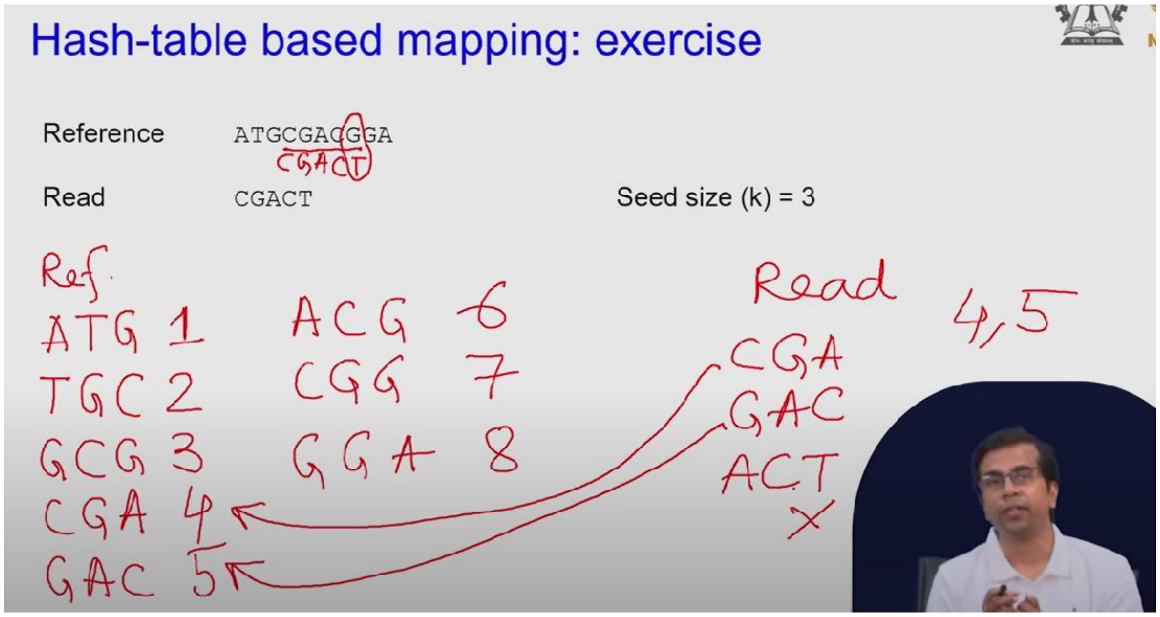
- *Chain* and align

So, what it means is that the full read right it comes from probably position 6 7 it covers that position right and not 16 24 right. So, if you have this multiple matches and in nearby locations you can probably be sure more or less is that this is where the read will map ok and you can go ahead with probably that only one alignment ok. So, having that multiple seeds we can actually do this much faster because we do not have to extend for those multiple positions ok all right. So, it is something it is a process that we call ah chain and align right. So, we are kind of chaining these locations right these ah they are mapping to the seeds are mapping to neighborhood locations and you can simply do chaining and alignment ok.

So, so this is the these are the 2 steps that we have for multi seed hash table based mapping ah. It is a 2 step process it is seed and search and then you have chain and align ok. So, let us look at another ah exercise right let us look at this take this example and ah let us illustrate right what we ah will how we will find this ah ah read position in the reference you know right also the seed size is given k equals to 3 right. So, let us build those seeds from the reference stored in the hash table and then also generate the seeds from the read and we will try to find them in the reference hash table ok.

So, let us do that for reference ok. So, for reference 1 we have k equals to 3. So, what we will do is we will generate this ATG right its location is 1 we have TGC location is 2 we have GCG location 3, CGA location 4, GAC location 5, we have ACG location 6, CGG location 7, GGA location 8 ok. So, we we need to just check whether there there are kind of repeated ah seeds right. So, we do not see any repeats right if we see repeats then we have to merge the locations right the values will be will be one of them right.

So, we will we can ah separated by a comma ok. So, once we have ah generated the seeds now right from the reference we have to generate the seeds from the read itself ok. So, for the read 1 right we have same seed size right CGA, GAC and ACT right we have generated these 3 and we can now search the hash table right. So, this is the hash table here we have the keys and the values ah and let us search ok. So, the first one will search CGA ok and we see is here right and the value is 4 right. So, its matching to fourth position what about then the next one which is the GAC right.



Hash-table based mapping: exercise

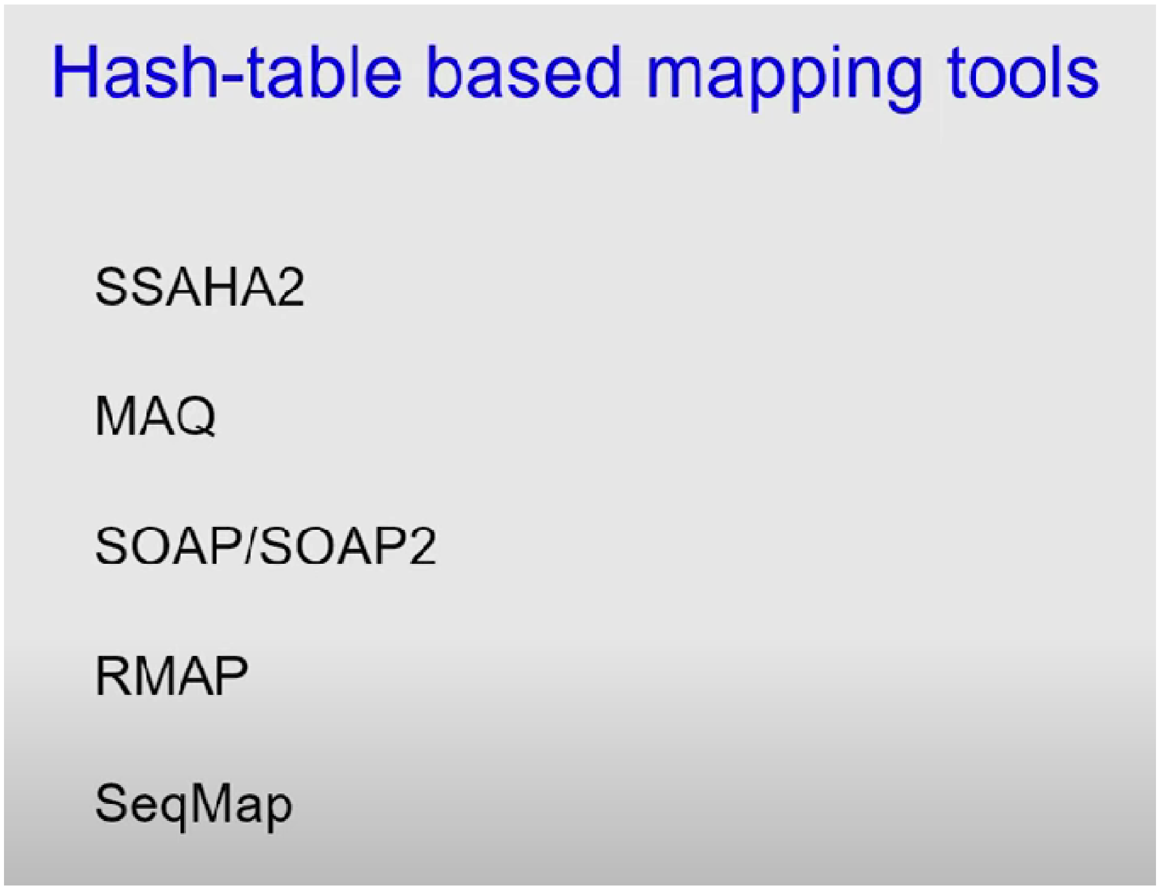
Reference ATGCGACGGA

Read CGACT

Seed size (k) = 3

So, it seems GAC is here right 5 what about ACT can you find ACT anywhere right I do not find ACT in the hash table right. So, we cannot find this one ok, but out of these 3 seeds from the read we have located 2 of them right and they are located at position 4 and 5 in the reference sequence ok. So, it kind of gives us an idea that probably the read will start from position 4 or so, its its in the region of this position right around this region 4 pi ok. So, and as you can see its actually in the reference if we go back its actually matching here right.

So, this is where it should be mapping right. So, if you write CGA C T ok and this is the reason why we did not we did not match or we we could not get the seed in the hash table this seed ACT right because there is a mutation right G to T mutation, but the other seeds actually work well right they could give us the location of the seed right. So, in in this simple example I hope its clear right how would you generate this hash table right for the reference sequence and then how do you generate the seeds from for the read and then how do you map these seeds to the reference hash table and in the in that process you will identify the location of the read ok. So, this such process is very very fast right. So, you remember it in the last class when we were talking about blast and blood right one of the problems one of the drawbacks we had is that we are doing a lot of comparisons right most of them were unsuccessful comparisons right and one of the discussions we had is that we want to minimize this number of unsuccessful comparisons right. So, here you see like we are kind of trying to maximize these successful comparisons right we are we are not going to do a lot of these unsuccessful comparisons So, I hope this this this is clear how it actually works right.



Hash-table based mapping tools

SSAHA2

MAQ

SOAP/SOAP2

RMAP

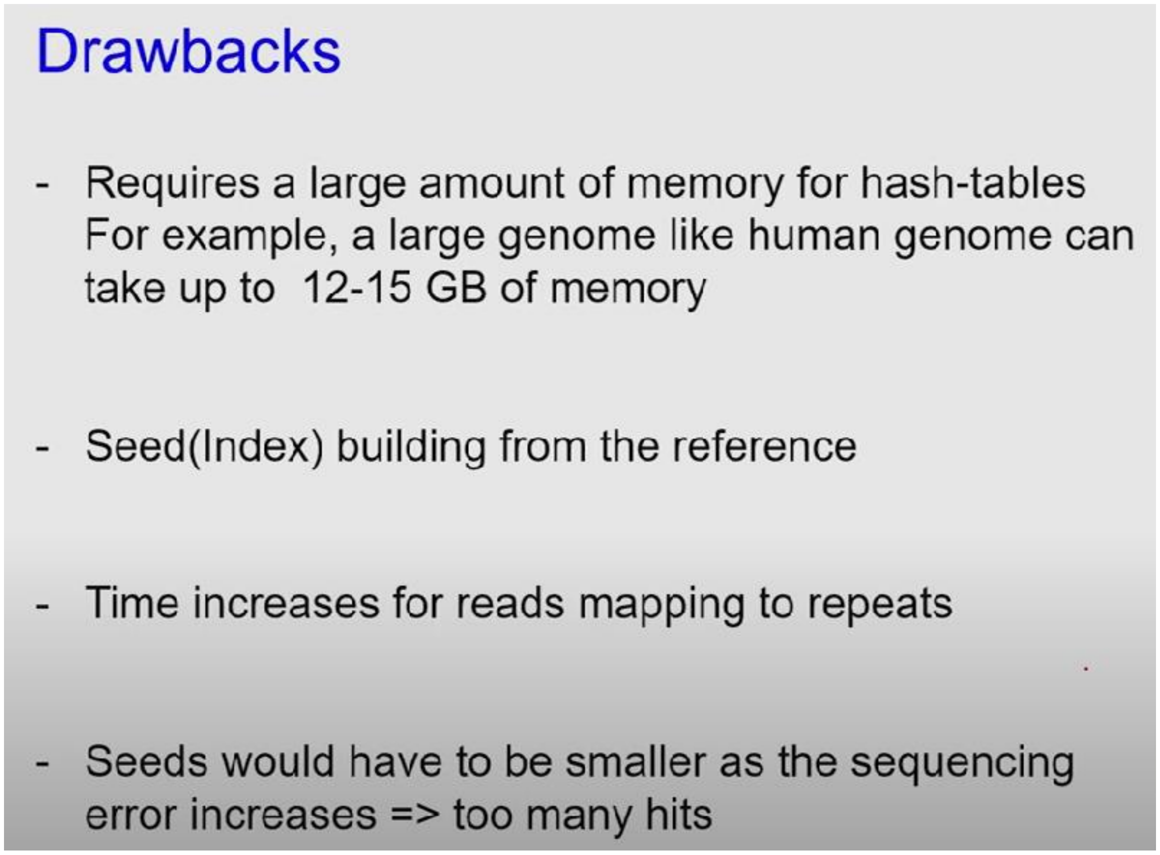
SeqMap

So, there are lot of tools for that utilizes this hash table right hash table based mapping tools here is a list of them right and you can of course, look up search and see how this work. So, we might ask why do we have so many different tools if they are based on the same algorithm of course, they are using slightly different implementations and how they actually go about implementing the algorithm right. So, you can of course, have variations in different in the implementation process and probably you can also add a few bit of optimizations here and there right. So, this is why you have so many of these algorithms ok they combine different ways of doing the same thing, but they are based on the same algorithm that we have just described ok.

So, what are the advantages of hash table based mapping ok. So, one of the first thing is that it is very fast right and if you just go and read literature for some of these tools you will see you can map millions of reads per hour ok. So, that is that is a very good number ok if you if you can let us say even map 1 million read per hour right. So, that is that is within few hours you can map a few millions right. So, this is a very fast algorithm compared to whatever we have we are we have discussed right blast and blast ok. So, this this this is very important for us and it can handle or map reads with mutations and sequencing errors as we have just described right.

We have seen that even if you have mutations or some sequencing errors there are there are generate mismatches with the reference sequence this process can easily handle those right because we are working with a lot a large number of seeds from the read right at maybe some of them will contain mutations right, but rest of them will match perfectly to the reference sequence or the hash table right increasing the hash table ok. So, that kind of allows for this mapping process right. What are the drawbacks then? If this is I mean we have seen this is very fast and we can handle reads with mismatches what are the drawbacks? So, as you can probably imagine right storing that data in the storing the hash table in the memory it will take a lot of space ok. So, if you if you want to store this in a hard drive also you need to give space right. So, during running the algorithms right we are running the tools you will have to load that data in the memory right that is why that is why the whole process is very fast because this whole hash table is in the memory of the system and you can simply search the seeds from the read against this hash table ok.

So, if you have to load this whole thing in the memory it will take a lot of space ok. So, for example, here if you if you are working with a large genome like human genome it can take up to 12 to 15 GB of memory right. So, the it means you are not going to run this in your desktop computer right. So, that is that is a limitation means it it prevents accessibility right. So, if you if you do not have an access to a let us say a workstation with a large amount of memory or a server where you have a lot of memory you will not be able to run this ok.



Drawbacks

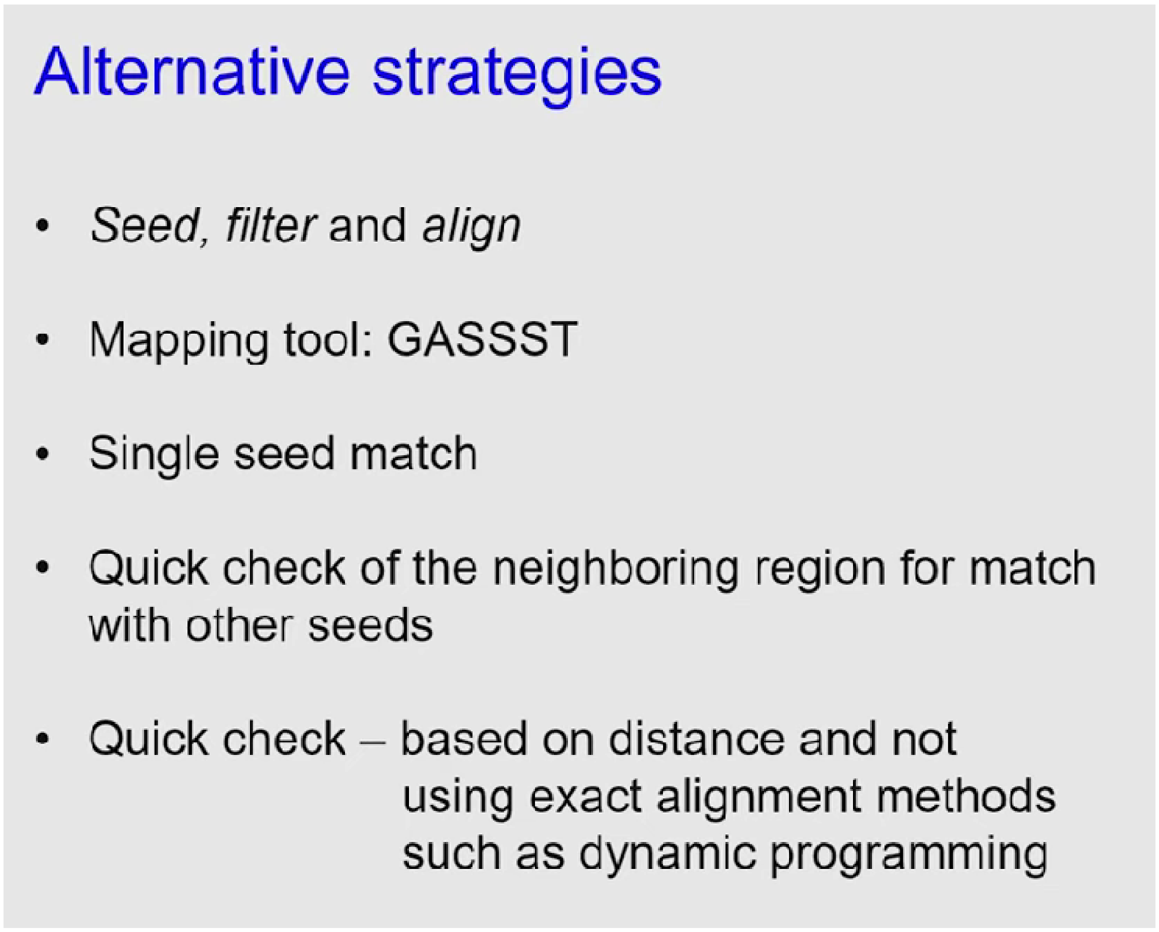
- Requires a large amount of memory for hash-tables For example, a large genome like human genome can take up to 12-15 GB of memory
- Seed(Index) building from the reference
- Time increases for reads mapping to repeats
- Seeds would have to be smaller as the sequencing error increases => too many hits

So, this requires specialized systems ok. The other process which is the seed building from the reference or the hash table that we generate ok that actually takes quite a bit of time right because as you can imagine you have to go through the go through this window right sliding window approach you have to go through the whole genome right to generate this hash table. And if you if your seed length varies like the k that we have discussed here if that varies then you have to build this every time you are working with some data right. So, again the question is can we not keep the seed length same right can we not let us say set k equals to 10 or k equals to 5. Now, the this actually is a problem right we cannot keep this fixed because if you are working with let us say different data from different platforms right they come with different yield length right and also they come with different sequencing errors ok. So, if you think about this carefully right if your sequencing error actually increases which is the case for long lead sequencing you have to design seeds to be shorter ok.

So, that a single seed most of the seeds will have some match against the hash table and they will not contain the mutation or the sequencing error right. So, this is something that will kind of impact your choice of seed length ok. Now, the other problem we have is that if you have reads that are mapping to repeats right you have you probably will have some increase in time right because if you see there is like a lot of matches for two regions you have to again then extend align and make that decision right whether these read is mapping to multiple regions or not ok whether it is read is coming from repeat sequences or not. Now, this is a problem that we have just mentioned right you you have to design smaller seeds right you have to reduce k as the sequencing error increases right. So, so, as to avoid that some seeds will contain a perfect match with the keys in the hash table.

Now, the moment you do that what will happen is that you will get too many hits right you can imagine right if you have a if you take a 3 mar right versus say 7 mar you will see more hits for 3 mar in the genome compared to the 7 mar right. So, so, as you increase the length of k you tend to pick up more unique sequences or unique combinations right. So, if you reduce the length you will get too many hits and again we have the same problem right we are getting too many hits means you have to again check probably through this alignment against multiple regions ok. So, this will again increase time for the whole algorithm. So, what are the alternative strategies that we have? So, this is an alternative strategy where you have something called seed filter and align right.

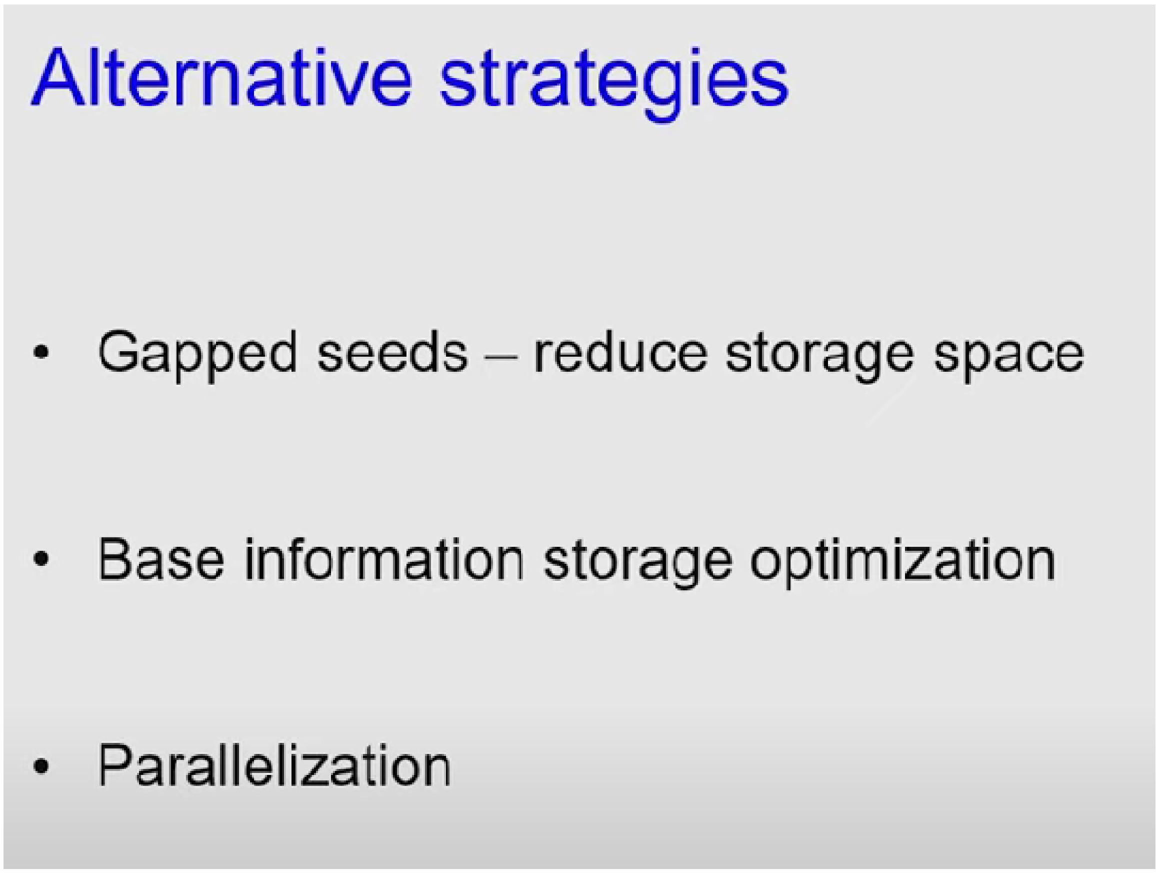
So, we have this seed right and this is implemented by a tool like a GSSSST right. So, what you have is a single seed match right. So, you so, this tool actually takes a single seed it matches against the reference hash table right. Now, let us say it finds multiple matches right multiple locations multiple values ok. Now, instead of trying to extend or aligning against all these matches all these locations what it would do it will quickly search right to this neighboring region for match ok.



Alternative strategies

- *Seed, filter* and *align*
- Mapping tool: GASSST
- Single seed match
- Quick check of the neighboring region for match with other seeds
- Quick check – based on distance and not using exact alignment methods such as dynamic programming

So, for match with other seeds ok. How does it actually quickly check do you think it is just align or extend method it is not actually it is a very fast algorithm based on distance ok. So, it is just counting number of bases in the seed versus the neighborhood and see which is likely to be a match which is not likely to be a match right. So, if it is a match you are likely to have similar number of bases right in the neighborhood of that seed ok. So, it is not using an exact alignment method such as dynamic programming which is time consuming, but it is a it is a very fast distance based method right.



Alternative strategies

- Gapped seeds – reduce storage space
- Base information storage optimization
- Parallelization

So, it works pretty well ok. So, there are other ways you can reduce now this storage requirement or memory requirement it is called gap seeds you can instead of generating all seeds from the reference you can say I will store only seeds at certain distances right maybe non overlapping seeds right. So, that will that will reduce the number of seeds and that will require less memory right. You can also encode the basis in some ways that you actually require less storage or less memory right this optimization can be done without going into too much details of that and you can of course, parallelize the whole process in case you have this multiple matches of a seed in the reference sequence. So, after this discussion what we need actually. So, we need really fast mapping which these tool actually does mapping to repeat sequence you should be able to handle this multiple matches very well which there is some problem with this method able to handle errors.

So, this method actually does that very well low memory requirement right again this is where this method would fail because it requires a lot of memory or storage ok. So, these are the references that we have covered and to conclude. So, this hash table based mapping algorithms are really fast and they can map reads containing sequencing errors. So, there is no issue with mismatches, but they require large memory right. So, this is this means you have to have special machines special servers for doing the mapping and it also becomes inefficient for its mapping to multiple positions or repeat sequences right and also increase in sequencing error which long lead sequencing platforms have right you see in this sequencing error it will require lowering of seed size which will mean multiple hits and it will increase mapping time that is it. Thank you.