1. Rcpp: if vec1 is a known IntegerVector, and if you do IntegerVector vec2=vec1, vec2 is like a pointer at vec1. Whatever changed vec1 will affect vec2.  
  
4. std::accumulate(iterator1, iterator2, 0.0)if you want to aggregate to a double, write 0.0 there! Not just 0!

5. Just moving the pointer in an array and just incrementing integer index take the same amount of time.

6. a[i] actually is \*(a+i), so incrementing i and using i to get a[i]'s value is much slower than iterator

7. Dear God!! k=i++ such kind of thing should never appear!! kicked my ass so hard and cost me almost 3 days!! Made me doubt so many things!! such as vector is unstable during recursion!!!

8. int\*i and std::vector<int>iterator i are not the same type, but could be similar enough to be both treated like pointers.

14. It seems there's no way you can write some container on your own to beat the STL container to speed.

16. // [[Rcpp::export]]std::vector<std::vector<double> > f(NumericVector v,unsigned len){   std::vector<std::vector<double> >M(len);   theMatrix(v,len,M);   return M;}// [[Rcpp::export]]std::vector<std::vector<double> > g(NumericVector v,unsigned len){std::vector<std::vector<double> >M;M=f(v,len);return M;}Good news! STL containers with wrapable types are directly translated into R objects! Don't need to manually translate them!BUT THIS WILL RESULT IN ERROR IF YOU USE RcppParallel!!!!

23. Today I met a very wired problem. My comonotonic function will do nothing and go bugged even when I let it std::cout<<"1.1\n" at the very beginning of the function. In the end I found it's because one of my vector goes out of bound. Now I am thinking the way the compiler does his job is gather all the non-output-on-screen things and do it after executing other stuff. No! Now I am thinking, it's because the program is so fast that it went bugged before the console had the chance to project the results onto the screenAnd also, it's possible that you didn't script your new functions! Be careful! Many years later: this is what's called undefined behavior.

24. Access the column or row in Rcpp::NumericMatrix X, use X(\_,1) or X(1,\_). Access the column in Rcpp::DataFrame is like X[1]

30. a=1:3; b=a;(or b<-a) and then if you input a and b as parameter in Rcpp function, the iterator or reference will share the same container!!

38. Reference the column or row in a matrix:  
NumericMatrix::Column x=X(\_,0);  
and this x is actually able to be seen as a numeric vector, so just do std::vector<double>y(x.begin(), x.end())!!

39. When writing the comparing function for std::sort(), do CompareValx(CellRank const &a,CellRank const &b)  
  
40. void f(DataFrame x) will also accept x as a matrix!!  
  
41. I guess the advantage of using NumericMatrix is only easily accessing the rows?.. Yes!!  
  
42. A data frame is essentially a list. Calling dataframe[,"a"] is slower than calling dataframe[["a"]] since [[]] is primitive function!  
  
54. Now for COPY the vector in data frame, you can directly do std::vector<double>v=X[0] where X is a data frame.  
  
55. std::lower\_bound returns the first element in the sequence that is equal or greater than val  
  
56. for .resize(), if the target size is smaller than the original size, at least for a vector of primitive types, .resize() will only modify the vector's header's "end" pointer. If the vector is expanded and the new size is less than its capacity, elements after the old .end() will still be deconstructed !!  
  
57. std::vector has 2 parts, one part is a header containing 3 members, .begin(), .end() and .capacity()  
  
58. when you make a struct of vectors, only the vectors' headers are stored continuously in memory  
  
59. MSVS configuration: Make sure that Run Time Library is set to "Multi-threaded DLL (/MD)", otherwise the .exe will run >15 times slower.  
  
60. Concurrency::auto\_partitioner(), some error in MSVS if you want to do parallel computing  
  
61. template <int N>// this is how you define a stack array..  
struct TestStruct {  
    int length;  
    int values[N];  
};  
TestStruct<3> t3 = {3, {0, 1, 2}};  
TestStruct<2> t2 = {2, {0, 1}};  
int main() {}  
  
63. for multiple .cpp source files in 1 visual C++ project where each source file contains a main(), go to solution explorer->source files->select the file that you want to exclude from the project building->right click->properties->exclude from build(Yes)  
  
72. construct a vector costs the same amount of time as use new, delete and initialize!

74. to use c++17 in R, do // [[Rcpp::plugins(cpp17)]]

75. To control the precision of the output file, do this:  
myfile.precision(16);  
myfile.setf(std::ios::fixed);  
myfile.setf(std::ios::showpoint);

76. An important mistake, for the stl container, I did this: std::vector<int>&v=V1 where V1 is another vector, and then I did v=V2...I thought it would be like rcpp object, v is just alias. However, by strict C++ semantic, when i did v=V2, V1 will be changed to V2!!! Be careful!!!

93. It is interesting that Rcpp will return a vector of vectors as a list, and will convert a list of vectors as a vector of a vector

94. return value optimization: if do std::vector<int>v=f(...) or std::vector<int>v(f(...)) where f returns a std::vector<int>, RVO will happen, but if do std::vector<int>v; v=f(...), RVO won't happen, since v has already been declared and initialized which means the head pointer of the dynamically array has been settled!

95. it's confirmed that Rcpp Matrix and RcppArmadillo matrix is implemented as a 1d vector. If the matrix is MxN, then the first M elements in the vector is the first column

97. !!!It is confirmed now: the return value and swap optimization will only happen to pure stl containers, like vector and vector of vectors. If you write a class which contains 2 vectors, the swap of classes and return value optimization will not happen!!! In this case, write your own swap function!!, and call this swap function at the end of your function to mimic return value optimization(the return value is brought out by parameter reference). As long as the inputs to std::swap() are two STL vectors of the same type, optimization is guaranteed.  
  
98. A lesson learned again: Never use a vector of NumericVector or other Rcpp objects, or a class that contains Rcpp objects. Always do copy, and it is fast! RcppParallel is not working correctly with Rcpp::List. Don't know RcppParallel works well with other Rcpp objects, but probably they won't. So, always copy those babies! If memory is limited, save those objects as binary files on disk and read from disk!!

100. A hard lesson on numeric precision: ordering multiple columns, if you are not sure, you'd better using integers as the keys or at least using rounded numerics!!!!

121. gccAssembly: <http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html>

124. std::vector<std::vector<> > will optimize the erase function without copying the inner vectors

125. In parallelRcpp, void operator()(std::size\_t st, std::size\_t end), if st=0, then in the for loop end will be 1, if st=1, then in the for loop end will be 2.... don't expect end to be the Number of cores!!!

129. Now, you finally know how to lock a share data in parallelCpp. Why don't you put tthread::mutex m as the global variable, and cap the code block with m.lock() and m.unlock()...

132. multithread numeric issue: there's no numeric issue you stupid!! You deleted the Makevars.win file in the src folder! Cost you an entire afternoon!

135. bug: Jesus Christ! Remember, x%0 will blow everything up!!!

139. c++ output file append: std::ofstream of("C:/Users/i56087/Desktop/hurricaneClustering/error.csv", std::ofstream::out|std::ofstream::app);

145. backslash \ in string: it is to stop the machine to interpret the next character. For example, add one at the end of line if you want to new a line in the string then it will ignore the new line command though in your text the new line is issued, making your code more readable.  An example of backslash: "create (m:Movie{'The Devil\\\'s Advocate'})". After processing this string, it will give: create (m:Movie{'The Devil\'s Advocate'}), which you can put in neo4j broswer.

146. Always do v.assign(size, 0) other than v.resize(size, 0) if you really need the initial vector to be filled with zeros.

160. now you know what the path variable is in Windows or Linux: for Windows, just go to computer -> advanced system -> Environment Variables -> Path -> and add the .exe you want to directly execute from windows command line.

165. c/c++ include file: whenever unsure, just include the full absolute file path and here is a good explanation: <https://caligari.dartmouth.edu/doc/ibmcxx/en_US/doc/complink/tasks/tuinclud.htm>

172. namespace kmClassical{...} good god it is useful! No more worries about same function/class names in different source files when building packages!

173. C++ writing style:  
1. Use spaces after commas and semicolons. Don't use spaces before commas and semicolons.  
2. Use spaces around binary operators, except ->. Don't use spaces with unary operators.  
3. Use a consistent placement-style for curly-braces..  
4. Put spaces after keywords like for, if, while, etc.

174. How to remove an R object to garbage collector from within C++:  
Language("rm","tmp").eval(); Language("gc").eval()  
BE CAREFUL: "tmp" is the name of that object in the R environment!

180. debug: Be careful of the constructor trap. Inside the constructor, you copy the arguments to the class's members, and at this time, if the arguments and class members have the same names, then modifications on the class members will not be effective because the compiler is confused and only assigns the modifications to the constructor's arguments!

184. Declare all functions in header files "inline" !! <https://stackoverflow.com/questions/10103161/inline-keyword-vs-header-definition>

185. volatile modifier prevents compiler reordering the code! good!  
   
186. When thinking in multithreading, always assume 2 threads A and B. If there are N lines of code including the gaps between the statements, evaluate the scenario when A is at line i and B is anywhere between line 1 and line N. This is how the code can be fully debugged.

189.  members in public Worker for parallelFor will be COPY to each thread!

190. Seems longjump in parallelFor's void operator()() definition will bug the program! It will succeed in the first run, but will bug the program in more runs.

199. copying 256 doubles is about 10 times slower than than arithmetics on two single doubles. Cache warming-up considered?

200. tested, memcpy has about the same speed as std::copy on vectors

201. modern compiler optimizes multiplication of 1 and -1  
  
207. Change the compiler optimization to -O3: Makeconf in etx/x64 directory

208. Rcpp output different types given different conditions: define the return type SEXP. More on using SEXP in Rcpp programming: <https://stackoverflow.com/questions/25172419/how-can-i-get-the-sexptype-of-an-sexp-value>

211. Initializing the vector with variable sizes inside a loop requires reallocating buffers (buffer head address changes).  
  
212. Initializing the vector with constant size inside a loop requires reallocating buffers (buffer head address changes).

213. Initializing the vector before a loop and resizing it in the loop requires reallocating the buffer until the vector's maximal size during this loop is achieved.

215. bug: compile with -O3 needs aligned data. You don't put a raw vector and enforce different types on this vector freely.

219. print to console in multithreaded environment: std::cout << "x = " + std::to\_string(x) + ", "

221. <http://www.catb.org/esr/structure-packing/> : Storage for the basic C datatypes on an x86 or ARM processor doesn’t normally start at arbitrary byte addresses in memory. Rather, each type except char has an alignment requirement; chars can start on any byte address, but 2-byte shorts must start on an even address, 4-byte ints or floats must start on an address divisible by 4, and 8-byte longs or doubles must start on an address divisible by 8. Signed or unsigned makes no difference.

222. read and write binary files <http://www.cplusplus.com/doc/tutorial/files/>

231. To avoid copying R data with RcppArmadillo, set the arguments of the exported function as reference, something like: // [[Rcpp::export]]  
void test(arma::mat &x) {\*x.begin() = 9999999;} // This will change x in R's global environment.  
An armadillo object will be return-value-optimized, but if it is returned to the R environment, the object will be copied!

232. Initialize empty matrix in Rcpp: Rcpp::NumericMatrix(0, 0, 0)  
  
233. Note that '-ofast' makes "std::isfinite()".. ineffective!  
  
234. Rcpp::Rcout to file, use sink(<filePath>); ...; sink()

236. Initialize list, initialize empty list, initialize rcpp list:  
// [[Rcpp::export]]  
int testnull(List x = R\_NilValue)  
{  
  if(x.size() == 0) return 0;  
  return 1;  
}

237. Parallelization could make the program slower if the calculation is extremely simple..

242. For recursion that has constants at all levels, declare these constants as members of a struct and move the recursive function inside this struct.

243. opencl: for maximal performance and avoiding unexpected errors, always choose local work group size of multiples of 64, and N(global items) a power of 2 and no less than the local group size.  
  
244. For C (C++) unknown, in the R package environment, if you want to fopen() a file, the relative path must be relative path to src, for example, in your source file in src, you should do fp = fopen("src/kernels/vector\_add\_kernel.txt", "r");

247. On the Linux cluster, the command "module load gcc-7.3" is useful. It will load the corresponding version of gcc and R will automatically connect to it and compile the code with the latest compiler. Notice that command needs to be included in .qsub files. Use module avail to see all the available environment modules.

248. Advance random number generator: mt19937.discard(). Look it up, useful!

250. std::pow(x, 2.0) is more than 30 times slower than x \* x...

252. The right way of making an R package with using an external C++ library is documented in tests in your Ubuntu machine. And, see stack overflow upvotes baby.

257. std::vector::insert and push\_back both grow the container size by two times if capacity is reached.

272. // std::chrono::time\_point<std::chrono::steady\_clock> nowtime = std::chrono::steady\_clock::now();  
  // std::chrono::time\_point<std::chrono::steady\_clock> endtime = nowtime + std::chrono::seconds(std::size\_t(duration));  
  // auto milliseconds = std::chrono::duration\_cast<std::chrono::milliseconds> (endtime - nowtime);  
  // auto timepassed = milliseconds.count();  
  // std::chrono::duration<float> difference = endtime - std::chrono::steady\_clock::now();

273. warning: vector constructor like vec<Astruct> x(3, Astruct(..)) is dangerous because  Astruct(..) is copied. So if there are pointers inside pointing to member vectors, .. you know what would happen.

274. A simple program that adds doubles up could be 10x faster under -Ofast than -O3. All this is due to the extra flag -ffast-math. But as of 20220305, -Ofast's speed is only about 1.06x -O2 for GCC-8.3 when -mfpmath=sse -msse2 -mstackrealign are enabled, which is R's default. This might be related to the new hardware.

275. Define macros before including header files that will use these macros! That will save redefinitions in the header files.

277. There is nothing wrong to std::vector<std::atomic<int> > x(len) when len is a variable computed on the fly.

292. Move semantics:   
int main()  
{  
    std::string str = "Salut";  
    std::vector<std::string> v;  
   
    // uses the push\_back(const T&) overload, which means  
    // we'll incur the cost of copying str  
    v.push\_back(str);  
    std::cout << "After copy, str is " << std::quoted(str) << '\n';  
   
    // uses the rvalue reference push\_back(T&&) overload,  
    // which means no strings will be copied; instead, the contents  
    // of str will be moved into the vector.  This is less  
    // expensive, but also means str might now be empty.  
    v.push\_back(std::move(str));  
    std::cout << "After move, str is " << quoted(str) << '\n';  
   
    std::cout << "The contents of the vector are { " << quoted(v[0])  
                                             << ", " << quoted(v[1]) << " }\n";  
}

293: std::stack and std::queue are all based on std::deque by defaut. See <https://stackoverflow.com/questions/6292332/what-really-is-a-deque-in-stl> and <https://www.codeproject.com/Articles/5425/An-In-Depth-Study-of-the-STL-Deque-Container> for more details.

295. Look at mat(ptr\_aux\_mem, n\_rows, n\_cols, copy\_aux\_mem = true, strict = false) in the armadillo library! You CAN wrap existing memory into an arma::matrix!

296. In armadillo, given z = x \* y where z is a matrix container larger than what x \* y would be, then z's address would stay the same before and after the assignment, and values in the leftover space will not change, but z's nomial dimensionality, e.g. z.size() and z.n\_rows will change.

297. To avoid warning when Rcpp::sourceCpp("pathToApackage"), hide DESCRIPTION and NAMESPACE into another folder, and remove RcppExport.R and RcppExport.cpp.

299. Don't be concerned about OpenMP thread pool. The threads will not be created every time! See upvoted questions on SO!

301. std::string behaves like a std::vector and cannot be in-place constructed upon a memory buffer.

302. You can supply MPI with a file of network addresses which it will use to determine which machines to run processes on. You can use the -npernode parameter to specify how many processes per node you want, so if you have 8 nodes and specify -npernode 1 , you should get 8 processes on 8 nodes.

304. Sadly, there is no way of avoiding copies for operator+(). Read <https://www.boost.org/doc/libs/1_54_0/libs/utility/operators.htm#symmetry>

305. OK, stop overthinking how to most efficiently implement temporary containers and if you should write all functions as functors. NO, don't do the latter unless necessary. Instead, create a ReusedContainers class object RC in the main function, reference this object in your functions that need extra space, and adopt a program-as-you-go style: just add new containers to  ReusedContainers' definition or make references to existing containers in it that can be recycled.

306. It has been tested, that std::vector<char> also honor the 8-byte alignment, at least compiled with gcc 8.3, cpp17, -Ofast,

307. vec<unsigned char> is 45x faster than vec<bool> for read and write given -Ofast!

308. Passing 64-bit integers from C++ to R: Use NumericVector x but do x.attr("class") = "integer64"

309. Be careful: assign a negative double to another double, the negative sign could be erased!

310. Don't think it too much anymore: there is no performance gain of for(int i = 0; i < 1000000; ++i) { do something; } against for(int i = 0, iend = x.size(); i < iend; ++i) { do something to x; }

311. It has been tested: even if you add all the GCC optimization flags that are included in -Ofast but not in -O2 to -O2's left, the speed result is still -O2 or even slower. You can only add those extra flags on -O2's right and then expect the speed result to become -Ofast.

312. Radix sort good document: <http://www.codercorner.com/RadixSortRevisited.htm>

313. C++ lambda can be like this since C++2014:

auto tmpprint = [](auto \*t, int size) {

      for(int i = 0; i < size; ++i) std::cout << t[i] << ", "; };

That is, arguments of the lambda function can be "auto", and the compiler will deduce its type automatically !!

314. If only the atomicity is needed, remember adding "std::memory\_order\_relaxed" option in the atomic member function!

315. Disable clang diagnostic in rstudio: .rs.setClangDiagnostics(0)

316. type of Rcpp object SEXP: <https://stackoverflow.com/questions/25172419/how-can-i-get-the-sexptype-of-an-sexp-value>

317. It has been confirmed. Using Rcpp::sourceCpp(), find the temporary directory it creates for building the binaries, and source the ".cpp.R" script from different processes spawned in snow, will work, and this means you don't need to compile in every process.

318. Avoid Strict aliasing rule pitfall: never reuse variables stored in a buffer that are casted to store something else.

319. Do not be superstitious about ordering index vector by the value vector! The cache locality violation would drastically slow down the process!

320. Use typedef! typedef unsigned char Uchar; typedef has local scope. It can be declared in a class like a member. Basically it works just like declaring a variable.

321. For Rocker, execute /bin/bash to enable copy and pasting. Use Rdevel for debugging!

322. NUMA means nonuniform memory access. L1, L2 caches are for each individual CPU, which is NUMA, but the L3 cache is shared by all processors. So when the data is very tiny in multithreaded computing, NUMA and UMA will make a big difference. When the data is large enough, hopefully, when 1 core is computing the data, the other core is reading the data from L3 or the memory.

323. Debug experience: often, you enjoy using the pointers and size information to represent vector, but when the vector capacity is 0, &v[0] and &\*v.begin() will give you runtime error: reference binding to null pointer.

324. Now you finally understand why STL uses f(Iter1 a, Iter1 aend, Iter2 b) so much. This is for template auto reduction! From now on, use as many as template parameters as possible, for dealing with different containers, data types, etc!

325. Call functions like pbeta() in C++:  
#include <Rcpp.h>   
#include <R.h>  
#include <Rmath.h>   
R::pbeta(…)  
log1pexp(…) // function defined in Rmath.h

326. Rcpp::String x is a reference to string objects in R environment. To change some character in Rcpp::String, do

auto y = (char\*)x.get\_cstring();

y[1] = ‘Y’;

will change the 1st character to ‘Y’.

327. Eigen C++ library’s superiority is that it writes its own matrix routines that do not rely on low level libraries such as BLAS. This is in contrast with Armadillo. Eigen is also header-only and implements numerous compiler-level optimizations.

328. Eigen::VectorXd xv = Eigen::Map<Eigen::VectorXd> (x , dim) is how to **copy** an existing buffer to an Eigen object. Using Eigen::Map<AnEigenObject> is not equivalent to using Eigen::AnEigenObject! The Map object is still just a **reference** to the underlying object, although it acts like AnEigenObject.

329. To disable Eigen stupid warnings, uncomment the macros for GCC in src/Core/util/DisableStupidWarnings.h

329. Think more about exceptions in C++ now. Things like throw std::invalid\_argument(“The size should not be negative”).

330: Eigen: Remember to add .noalias() before matrix assignment!

331: try{} catch(…) {} . Here, the 3 dots … means “any kind of exceptions”.

332: If you have a template class with template member function, be careful that the template class parameters should always be resolved first.

333. Check if object is f certain type: std::is\_same<X, Y>::value.

334. if constexpr ( true ) tells the compiler to ignore any error in the other branch during compile time.

335. For C++ functor overloading template operator(). If the compiler cannot deduce the template parameter automatically, you would need to call it like A.operator()<T,S>(x, y).

336. Avoid using Rcpp::DataFrame as much as you can! Constructing Rcpp::DataFrame is **SUBTANTIALLY SLOWER** than Rcpp::List.

337. To make custom hashable, consider overloading the hash function or just use string\_view, which in C++ can be hashed.

338. Be careful about bit manipulation truncation issue.

339. Primes that are near power of twos: uint64\_t primes[64] = {

2ull, 3ull, 5ull, 11ull, 23ull, 47ull, 97ull, 199ull, 409ull, 823ull, 1741ull,

3469ull, 6949ull, 14033ull, 28411ull, 57557ull, 116731ull, 236897ull,

480881ull, 976369ull, 1982627ull, 4026031ull, 8175383ull, 16601593ull,

33712729ull, 68460391ull, 139022417ull, 282312799ull, 573292817ull,

1164186217ull, 2364114217ull, 4294967291ull, 8589934583ull, 17179869143ull,

34359738337ull, 68719476731ull, 137438953447ull, 274877906899ull,

549755813881ull, 1099511627689ull, 2199023255531ull, 4398046511093ull,

8796093022151ull, 17592186044399ull, 35184372088777ull, 70368744177643ull,

140737488355213ull, 281474976710597ull, 562949953421231ull,

1125899906842597ull, 2251799813685119ull, 4503599627370449ull,

9007199254740881ull, 18014398509481951ull, 36028797018963913ull,

72057594037927931ull, 144115188075855859ull, 288230376151711717ull,

576460752303423433ull, 1152921504606846883ull, 2305843009213693951ull,

4611686018427387847ull, 9223372036854775783ull, 18446744073709551557ull};

340. Again, std::string is just a std::vector of chars! Elements of the string can be assigned. And check out all its member functions that align with std::vector!

341. When linking all the .o files together, another way of preventing naming conflict is to create class and make variables/functions static. This will allow you to call function in the fashion of class::func()!

342. **lvalue, rvalue mystery no more!** **Keep using rvalue reference as much as possible**, e.g., void f(int &&x)! This has 2 advantages: if you want to supply x as a reference, just do f(std::move(x)), and x’s value will be changed inside f() if you do change it. If x is just a temporary variable, e.g. f(1 + 2 + 3) will be fine! If you want to supply x as a copy, just do f(int(x))! Nice Nice Nice!! Can’t believe it took you this long to enjoy rvalue reference.

343. **Segmentation fault vs. stack smashing**: segmentation fault is when you try to access memory location out of bound or location that does not exist. Stack smashing is alert that warns about accessing out-of-bound locations on the stack. **Stack overflow** is when the stack is not large enough to hold you program. It happens if your recursion is too deep. **Memory leak** means the allocated memory is not deleted. **Buffer overflow** is segmentation fault.

344. Rule of 3: a class has destructor, copy constructor, and copy assignment. Rule of 5: a class has destructor, copy constructor, and copy assignment (=) overloaded, move constructor, move assignment (=) overloaded.

345. struct A { double x; }; const A\* a; decltype(a->x) y; // type of y is double (declared type)

346. When a template class has another template class with a template function, you need to write something like: **a.template run<1> (x);** See your SO post: <https://stackoverflow.com/questions/77189046/when-a-template-class-has-an-instantiation-of-another-template-class-with-a-temp> And what’s worse, if it is the operator that is overloaded, you would need to do something like this: **lms.template operator() <monoType>(y, y + (xend - x), indx.data());**

347. **Template meta programming Lesson 1**: **Check if an object is certain type, partial template specialization** **which is not allowed for function.**  
Use partial template initialization to achieve deciding an object’s type:  
*template<typename T> struct isVector { constexpr bool operator()() { return false;} };*

*template<typename T, typename A> // Partial template specialization.  
struct isVector<vec<T, A>> { constexpr bool operator()() { return true; }; };*

What the above does: we first define a generic template class isVector that always returns false given any type, then we define a specialized template class that will override the generic class if the instantiation takes a vec<T,A> as the template parameter. The syntax is *struct Name<theType> { …… }*.  
Example 1: *if constexpr ( isVector<D>()() ) doSomething else doSomethingElse*.  
Example 2: *if constexpr ( isVector<decltype(objectLvalueReference)>()() ) doSomething else doSomethingElse*.

348. **Template meta programming Lesson 2**: **Variadic template, variadic parameter, parameter packing.***// Treat typename... as a single keyword.  
// ellipsis on the left of a type definer means the object is a parameter pack, on the right means unpacking.*

*template <typename T, typename... Args>  
auto give(std::size\_t size, Args... restSizes) // restSizes is a pack. Return value can be auto.  
{*

*// sizeof...() is a single operator. It gives the number of parameters in restSizes.*

*if constexpr (sizeof...(restSizes) == 0) return giveCore<T> (size);*

*// The other branch must be wrapped inside else {} to let the compiler know they are mutually exclusive.*

*else*

*{*

*// give<T>(restSizes...) will take the first parameter in the unpacked as the size argument, and the rest as a subpack.*

*vec<decltype( give<T>(restSizes...) )> rst(size); // Get the type of a function’s return value by just run it.*

*for (auto &x: rst) give<T>(restSizes...).swap(x);*

*return rst;*

*}}*

349. template<typename T> using vec = std::vector<T>; // This will replace #define vec std::vector. A better choice.

350. In any scope, order of destructor calls is guaranteed to be the reverse order of construction declaration, no matter how compiler optimizes the code.

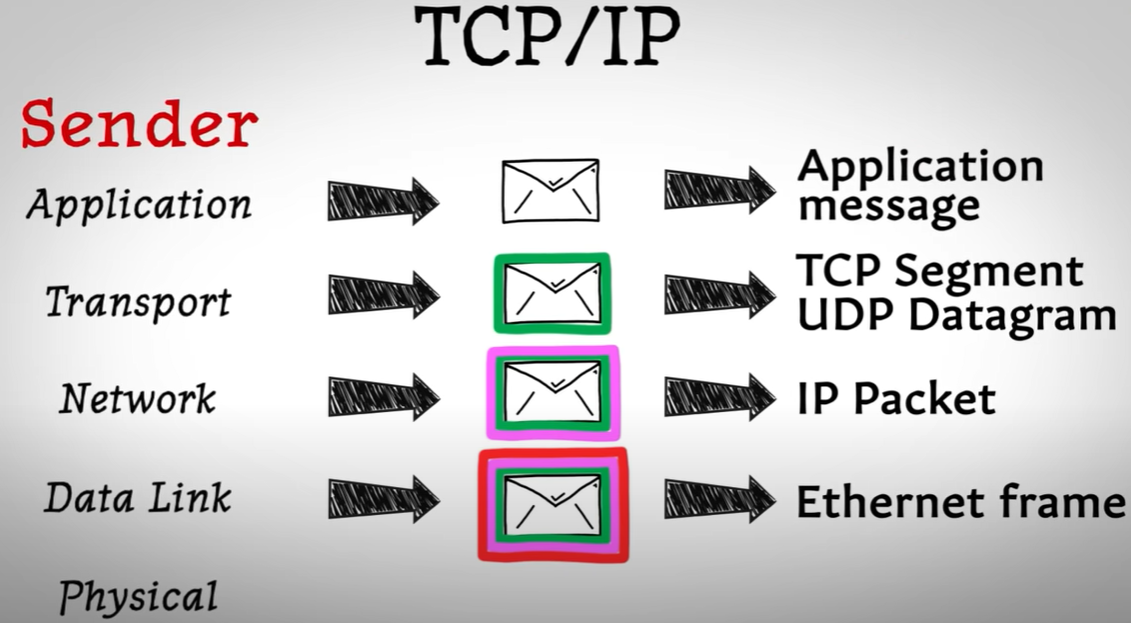
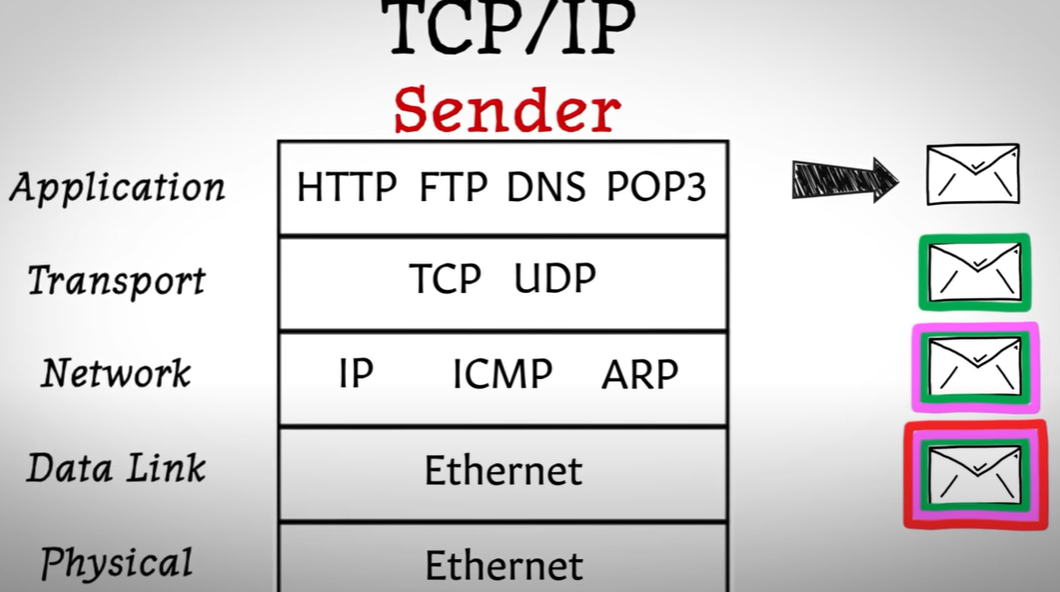
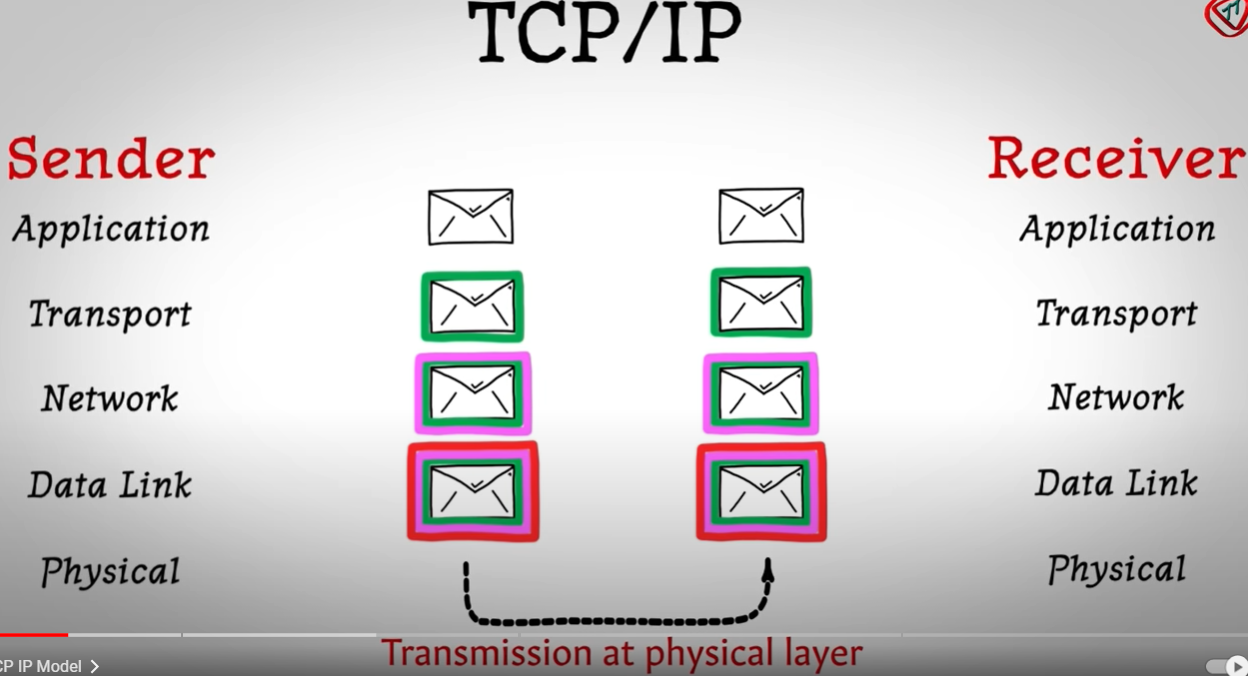
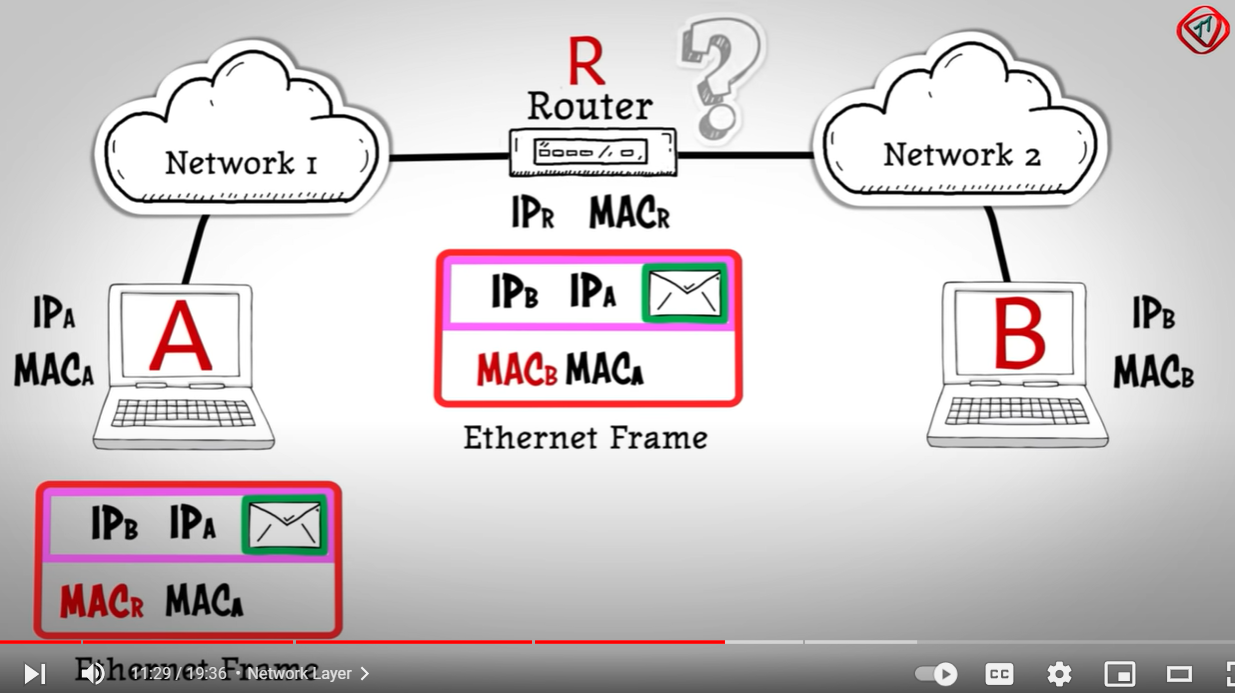
Frequent questions:

1. Virtual function is a way to achieve runtime polymorphism. Let’s consider the situation where you declare a virtual member function in the parent class, and then you override the same member function with different definitions in the children classes. Now if you have a parent class pointer that points to a child object, and then if you use this pointer to invoke that member function, you will invoke the definition in the children class, not the parent class. Without the virtual keyword, this action will invoke the parent definition.

I think different compilers will have different implementations to achieve this. But I guess in general, the compiler will add a static member pointer that points to the vtable, which basically is an array of function pointers.

I hardly use virtual function in my codebase, but I sometimes do use the std::function class, which in my understanding is implemented basically under the hood as base class + derived classes and virtual functions. It’s more complicated than lambda expression, which basically is implemented as a functor, like a struct with overloaded parentheses operator.

I am a big fan of using functors and templates.  
std::function is an object, so it will be deep copied.

1. TCP/IP: https://www.youtube.com/watch?v=2QGgEk20RXM
2. 
3. 
4. 

TCP segment is wrapped with the sender’s and receiver’s IP addresses and the sender’s MAC address and the router’s MAC address. This wrapper is an Ethernet frame. It is then sent to the router which the receiver’s network connects to. The router resolves the IP address of the receiver to find the MAC address of the receiver’s physical device, and the TCP segment is extracted from the ethernet frame and sent to the receiver.

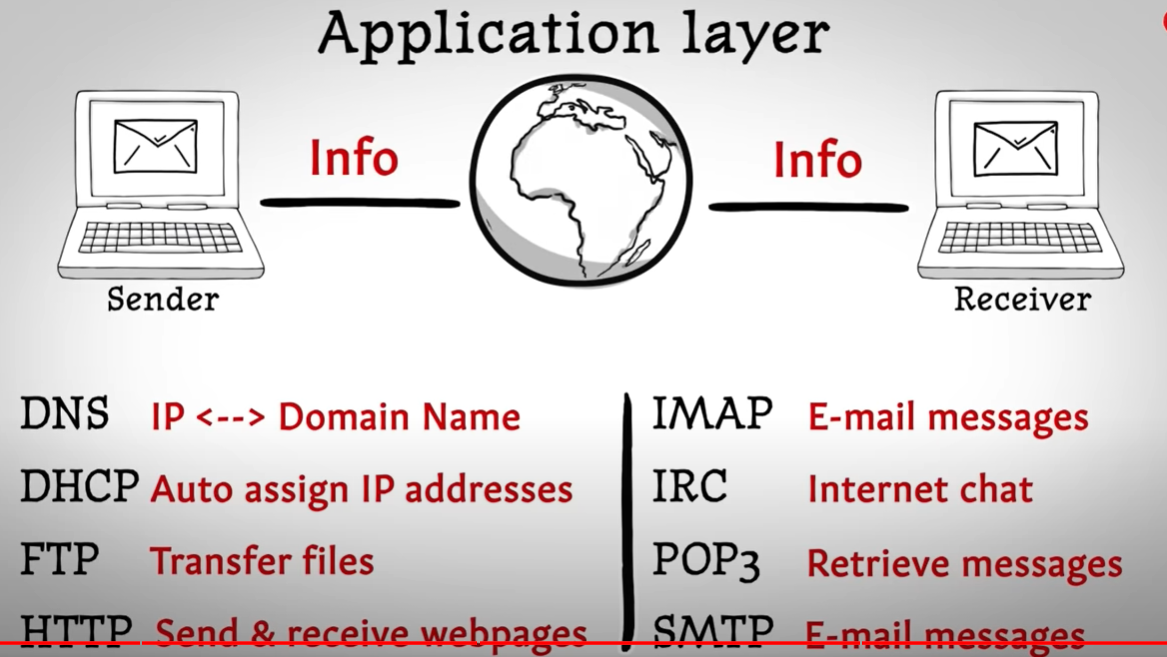
The transport layer receives a message from the application layer. If TCP is used, we divide the message into small pieces. Each piece is added with some header information to form a TCP segment. The header information includes the segment index for reassembling the pieces on the receiver’s side, and the checksum for error correction. The checksum can be just some hash value, or just some XOR operation on all the bits. (Mathematically it’s not guaranteed that the error will always be discovered, but the false negative rate will be less than the 1 over the number of the atoms in the whole universe).

TCP establishment: 3-way TCP connection establishment. Sender to receiver Request 🡪 receiver to sender acknowledgement 🡪 sender to receiver acknowledgement. The communication is robust. Every data packet the receiver receives, it will send back acknowledgement. If receiver finds something is wrong in the data packet, it will not acknowledge. In this way, TCP ensures there will be no missing packets and correctness of the data.

TCP can also control the network flow to avoid congestion. TCP will send packets at an increasing speed until the sender fails to get acknowledgements, which implies the network capacity has been reached. It then will keep sending the packets in a less rate until near the end of the transmission.

TCP ends the connection in a 4-way handshake. Sender to receiver request 🡪 receiver to sender acknowledgement 🡪 receiver to sender request 🡪 sender to receiver acknowledgement.

On the other hand, if UDP is used, the messages from the application layer should be short enough to be fitted in UDP datagrams. UDP datagrams are considered unreliable, because the packets could reach the receiver in a different order. UDP is called a connectionless protocol since it does not create a virtual path from the sender to the receiver. Packets can travel in different paths from the sender to the receiver. Packets could be lost during transmission and UDP does not bother correcting it.



1. Volatile, memory order, cache coherence, try-catch block.
2. **Memory order** - **sequential consistency**: No code before any atomic operation will be executed after the atomic operation.
3. **Memory order – relaxed order**: It is only guaranteed that, for a sequence of operations on the same atomic variable, their order of execution will always be respected. This sequence can be shifted by the compiler. The gap between neighboring elements in the sequence can be changed by the compiler. Use this only if you want atomicity but no synchronization**. No synchronization between threads is guaranteed. Typically it is used only when atomicity is needed but not synchronicity**.
4. **Memory order – acquire/release order**: something between sequential consistency and relaxed order. For any 2 sequences of operations on 2 independent atomic variables X and Y, the compiler can freely swap the order of operations if the order of operations on X and the order of operations on Y are preserved.
5. **Cache coherence**: Protocols to ensure that caches dedicated to different cores have the same copies. The protocols include MSI, MESI, MOSI, MOESI, etc. These protocols basically give different states (Modified, Shared, Invalid states etc.) to cache lines, and then the machine will employ bus snooping or cache directory to achieve coherence.

Atomicity is achieved by cache coherence mechanism with some extra engineering like exclusive read and write to the cache lines of interest.

When reading from memory, value will leave copies in multiple levels of caches. After they come into the registers and are computed, values can be written back to caches or write through to main memory. Most architectures do write-back cache. Write-back cache will write the cache line back to main memory only when it is pushed out of the cache. There is a dirty bit to inform if the cache line has been modified or not.