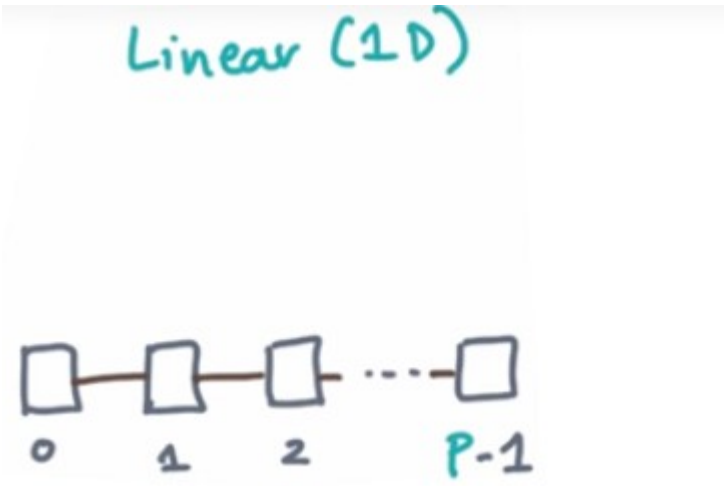
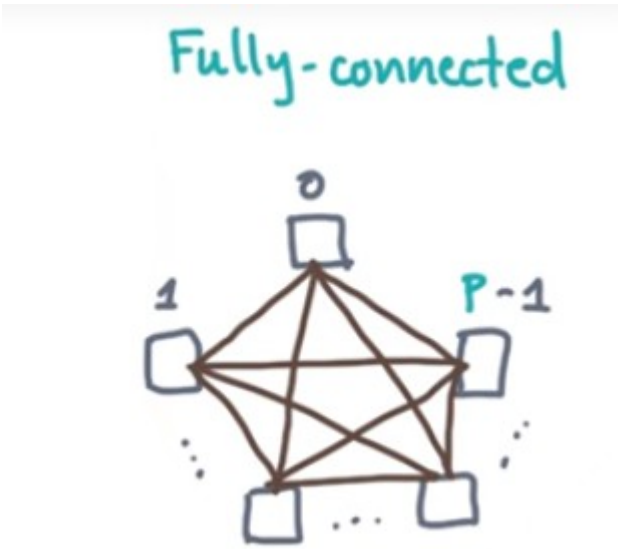
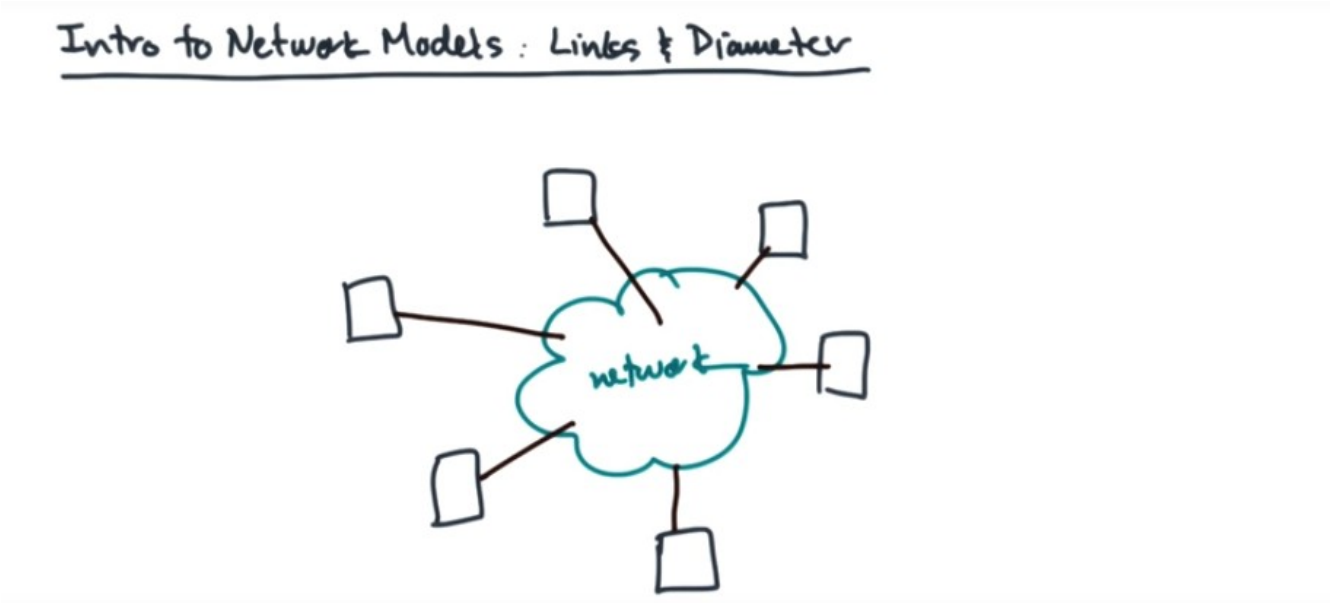
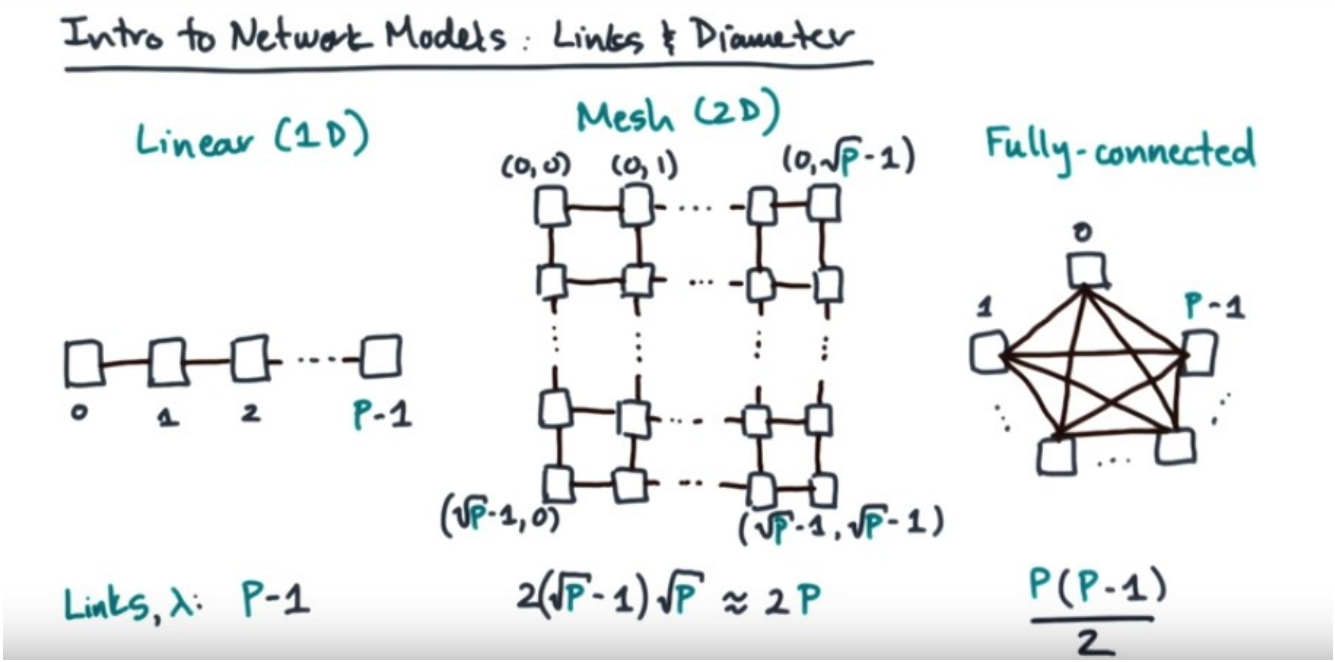




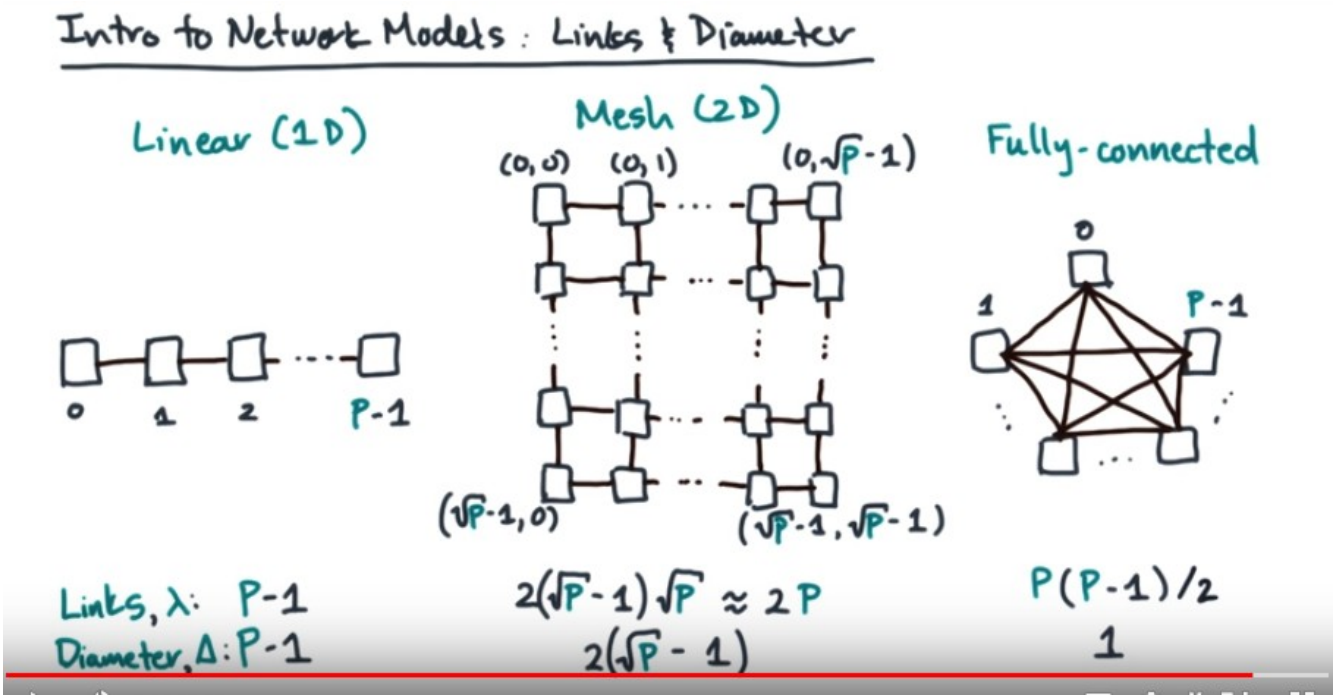
1. Algorithms in the message passing model, always assume some topology for the network. But suppose you design an algorithm for one kind of network, and then run it on another one. How well will it do? This lesson considers network topology, which will give you some techniques for thinking about this question. Now in the early days of parallel and distributed computing, network topology was a big deal. Then, through better engineering of networks and through virtualization, it seemed like it didn't matter so much anymore. That led to cost models, like the alpha beta or the latency bandwidth model, where you could just sweep away a lot of the details of network communication under the hood. But communication layers are getting faster all the time. And the scale at which we operate systems today, is getting bigger and bigger. Thus my personal opinion is that once you have a billion or more processor systems, the network topology is going to start to matter again a lot. And so we shouldn't forget its basic principles.



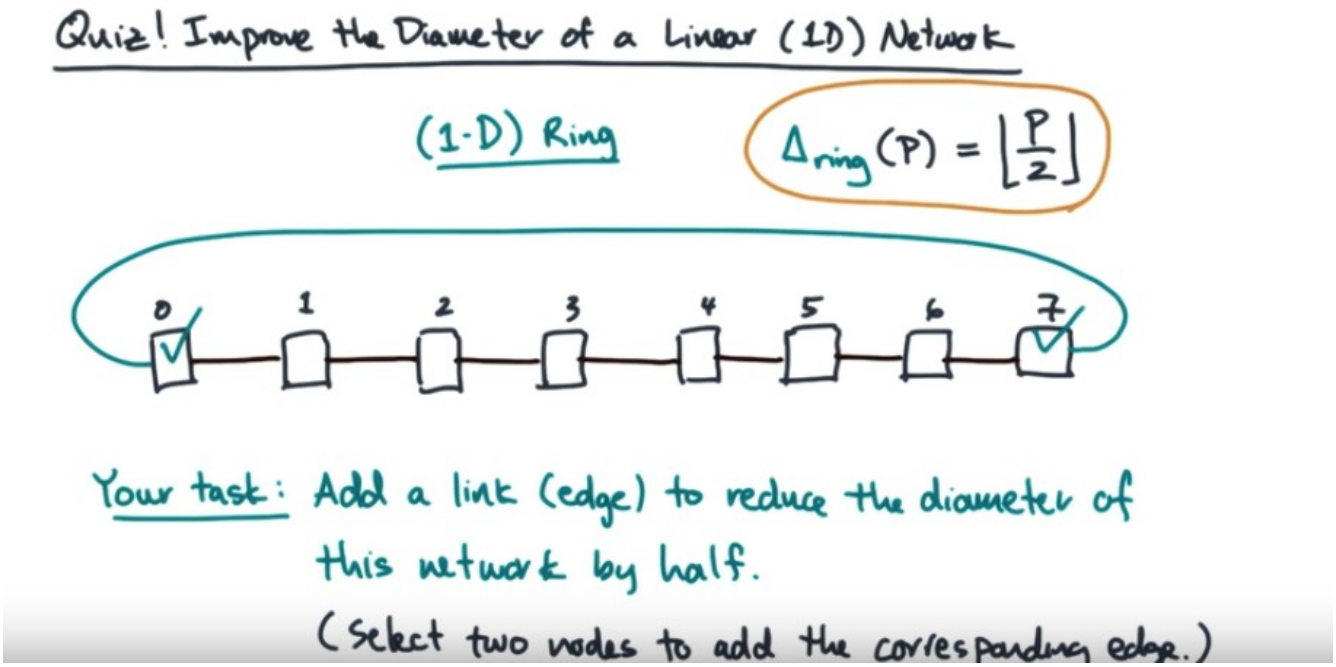
2. I want to start by introducing you to a couple of really important properties about a network whenever you're thinking about designing algorithms. Remember that abstractly our model of a distributive memory machine is a bunch of computing nodes connected by a network.



Let's take three examples. The first is a linear or 1D network. It's a set of p nodes connected in a line. A second example is a 2D mesh network. Again it's a set of p nodes, but this time arranged in a grid. Notice how every node is connected to its north, south, east, west neighbor. Now this third network is a fully connected network. Notice how every node has a direct connection to every other node. That brings us to our first important property which is the number of connections, or **links**. So given P nodes, a linear network evidently has P -1 links. A 2-D mesh network has this many links which is about 2 times P. A fully connected network has about P squared over 2 links. You care about the number of links, because it's a proxy for cost. That is, a network with many wires will probably be much more expensive than one with fewer wires.

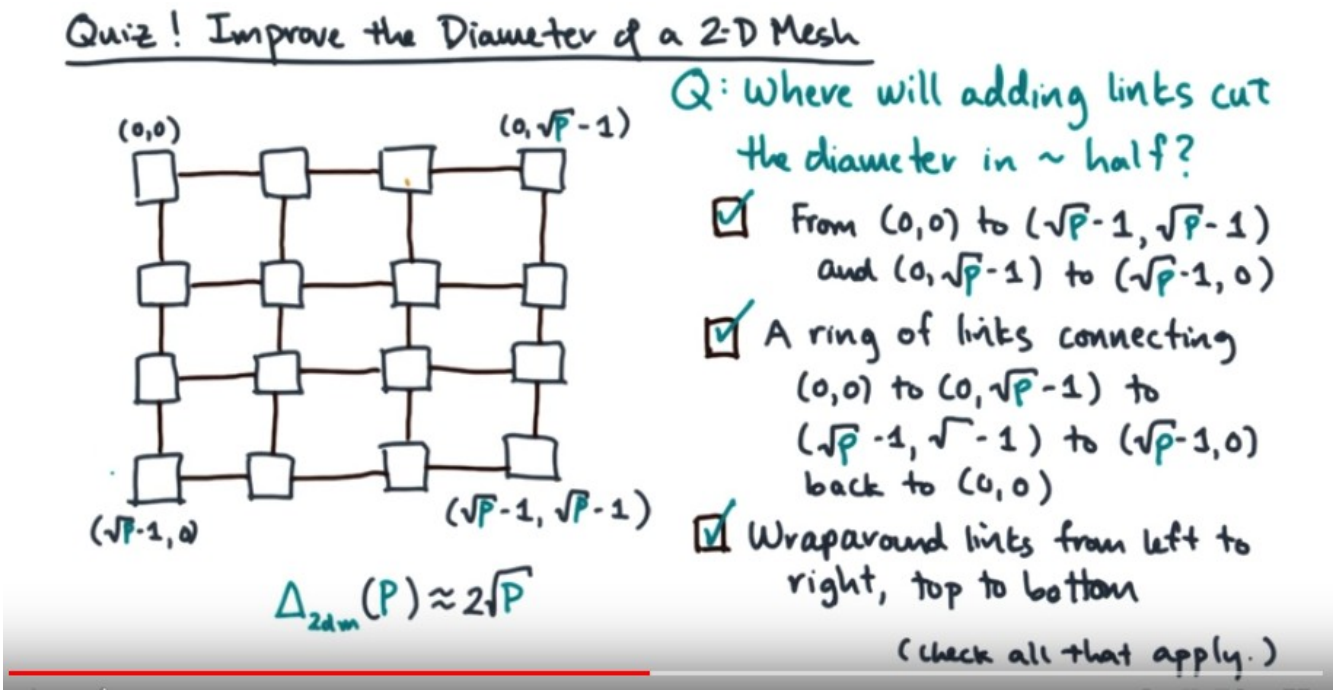


I don't know what they paid the person who wired this machine. But whatever it was, it's not enough. A second important property of a network is its **diameter**. To calculate it, you take all pairs of nodes, and compute the shortest path. Then you take the longest such path. 簡單地講, 就是 take all pairs of nodes, compute the paths, take the longest path. 為何原文要加 shortes path, 是為了防止有人在給定的一個 pair 下像出租車司機一樣故意找繞道的 path. 如果人人都不像這樣的出租車司機, 則 diameter 的定義就可簡化為: longest path among all pairs. That longest shortest path, is the diameter which I'll denote by capital delta. For the linear network, the longest shortest path is the one that connects the end points. The end points are separated by P- 1 links, therefore the diameter is P- 1. What about the mesh? The longest shortest path is the one that connects either of these end points. The distance is just the **Manhattan** distance. That's basically, root P- 1, followed by root P- 1 or 2 time root P- 1 links. A fully connected network has a link between every pair of nodes so it's diameter is just one. Diameter's an interesting property. Essentially it's a proxy for the maximum distance that any message must travel in the absence of network contention. I really wish I was running on a network with smaller diameter.



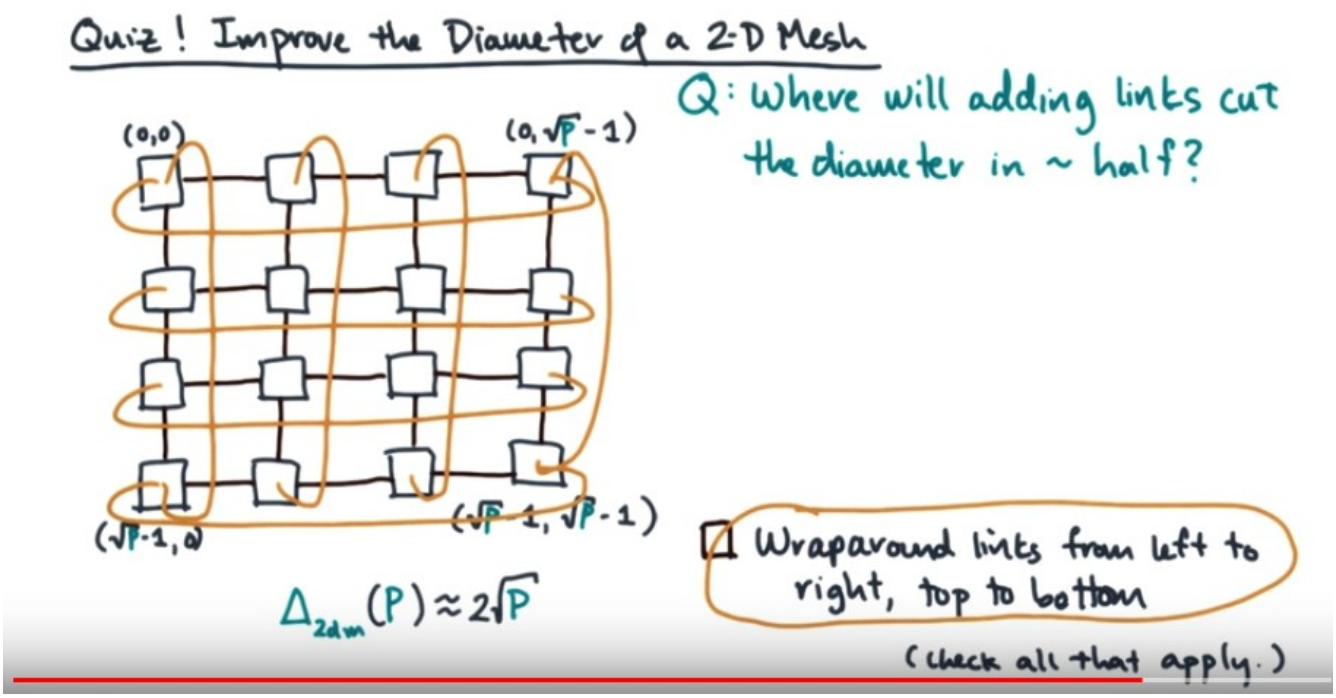
3. Quiz time, suppose you were given a linear network of nodes. I claim you can add one link to this network, and reduce its diameter by roughly a factor of 2. I want you to add this link. To add a link, just pick the two nodes that represent the endpoint. For example, if you think the link between 2 and 3 will cut the diameter in half, then just click on 2 and click on 3.

4. Remember that the diameter is defined as the longest shortest path. For a linear network, that path is the one that connects the endpoints. So, a good candidate is to connect those. In fact, if you do that, then you get what's called a ring network. Topologically, a ring is just a circle. So the longest path is half the perimeter and that's just about half the diameter of the original network.

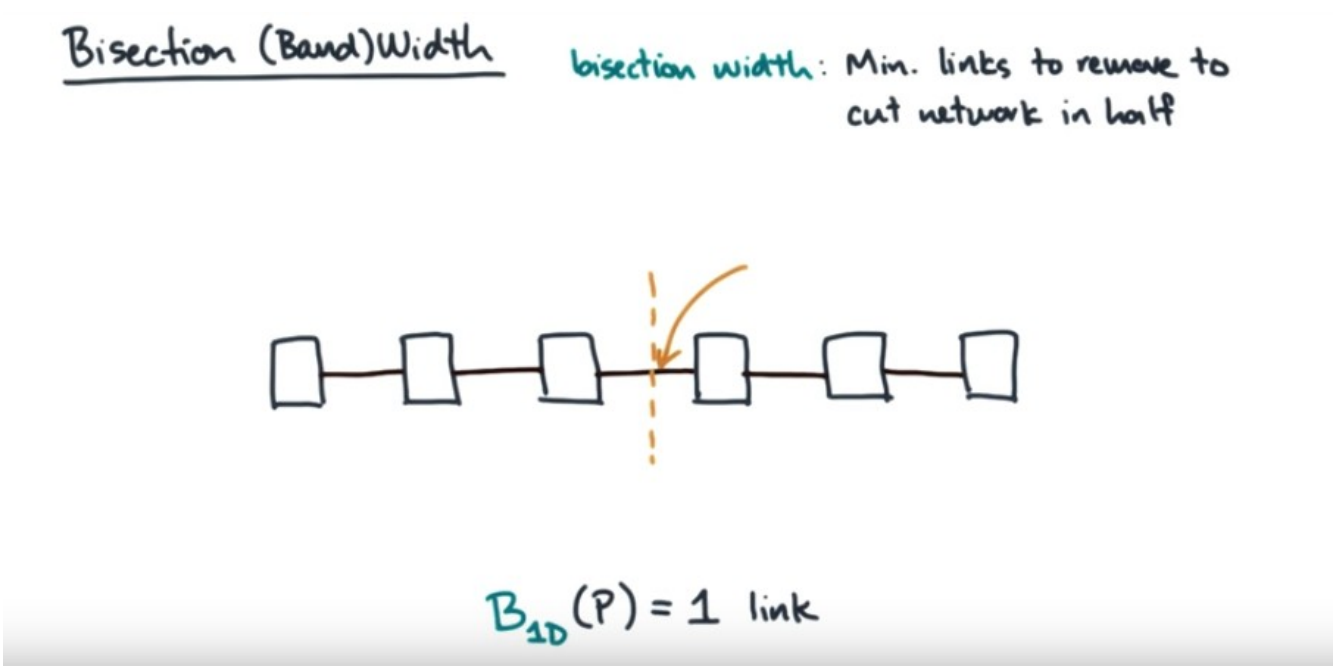


5. Suppose you're given this 2-D Mesh network. It has a diameter that's approximately 2 times the square root of P. Here's my question. Where do you need to add links to reduce the diameter by half? Let's label every node by its coordinates, starting at (0, 0) in the upper left hand corner and going to (root P- 1, root P- 1) in the lower right. Here are your choices. Is the answer to connect the opposite corners (即在該圖中加一個大 X)? Or is the answer to add a ring of links that connect the corners (即加一個外接圓)? Or is the answer to add wraparound links from left to right and from top to bottom (意思見下圖)? Check all the ones you think are correct. If you think none of them will cut the diameter in half, then submit without checking anything.

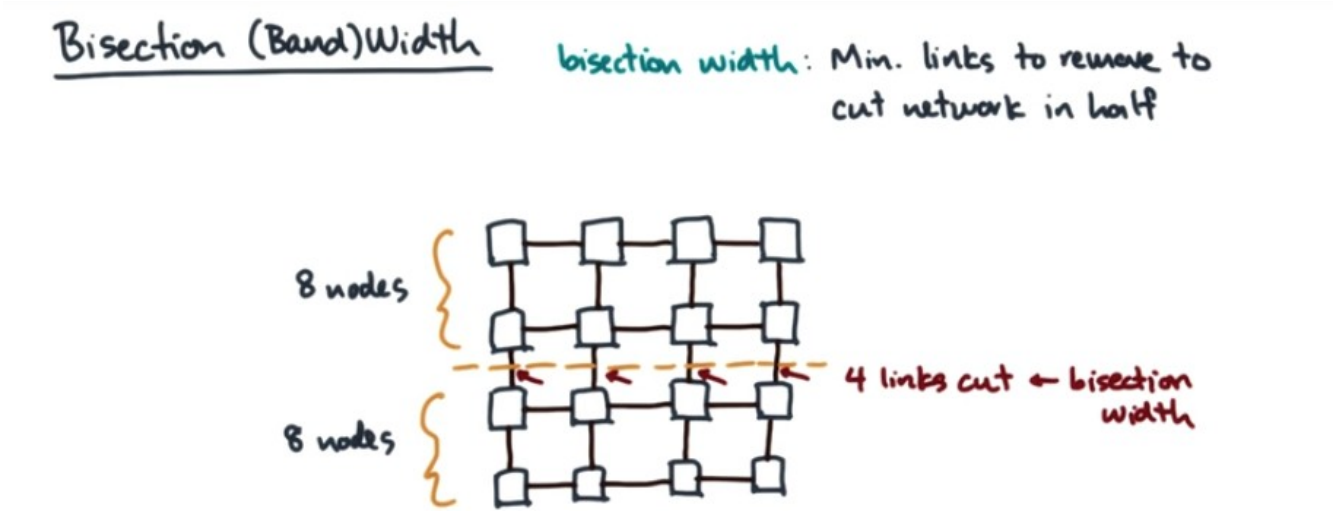
6. Here is the answer. In fact, all three options will reduce the diameter by half. The longest paths connect the corners. So, anything that shortcuts the corners will reduce the diameter.



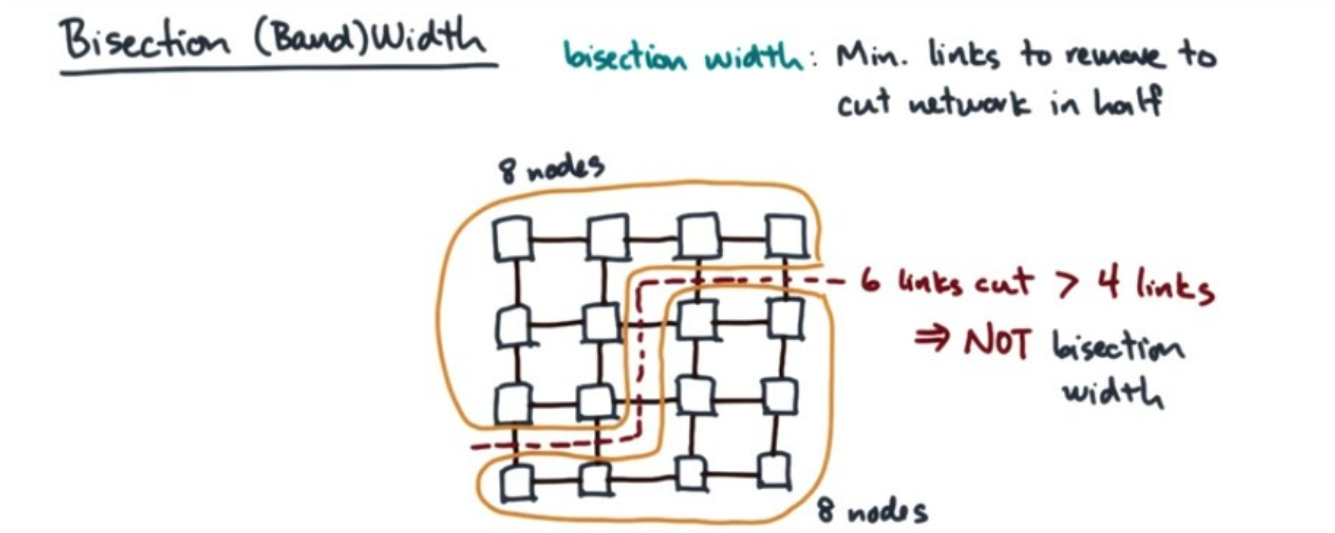
Now, the last option is special. It turns our 2-D mesh into a 2-D torus. Oops. Wrong torus.



7. Another property of a network that you should really care about is its bisection width and there's a related concept which is the bisection bandwidth. The bisection width is the minimum number of communication links that you have to take out in order to cut the network into two equal parts. Equal is measured by the number of nodes. For simplicity, lets assume that the number of nodes is even so that bisection is well defined. Eep! For linear network, cutting this link will break the network into two approximately equal size pieces. Again equal is measured in terms of the number of nodes. In this case, you only need to break one link, so the bisection width is one. I'll denote bisection widths by capital B and in general they'll be a function of the number of nodes.



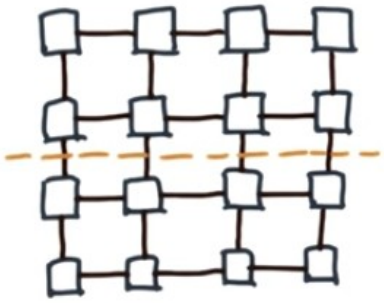
What's the bisection of a 2D mesh? Here's a cut that breaks the network into two equal size pieces. In this case each half has eight nodes a piece. This cut goes through four links. In fact, there is no cut with fewer than four links, so the bisection width is four.



For instance, here's another cut that breaks the network into two equal sized pieces. However, this cut is not a bisection. It goes through one, two, three, four, five, six links.

Bisection (Band)Width

bisection width: Min. links to remove to cut network in half



$$B_{2D}(P) = \sqrt{P}$$

Now in general, a mesh with P nodes, will have a bisection width of square root of P.

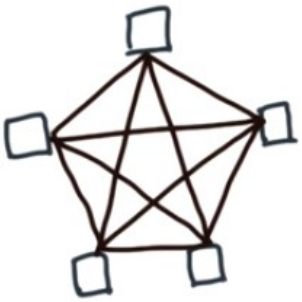
Bisection (Band)Width

All-to-all personalized exchange

So, why do you care about bisection? Well, one really important communication pattern is something called an all to all personalized exchange. An all to all is a collective, where every node wants to send a piece of data to every other node. Sort of like a bunch of nodes shouting at each other. And what this means is that every node in one half of the network will want to send messages to every node in the other half of the network. So all message traffic will have to go through the bisection. Of course networks with larger bisection widths will have a better capacity for carrying all that traffic.

Bisection (Band)Width

fully-connected



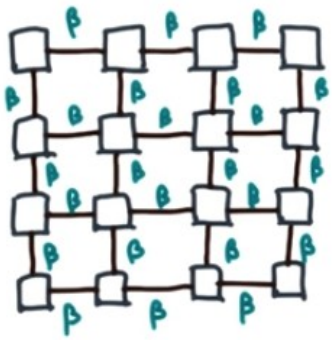
$\approx \frac{P^2}{2}$ links

$\Rightarrow B_{fc}(P) \approx \frac{P^2}{4}$

Now, for the fully connected network, we said there were about P^2 / 2 links in total. Remember that every node has a direct link to every other. So to cut in half, you'd have to take about half the links. That yields a bisection width of about P^2 / 4.

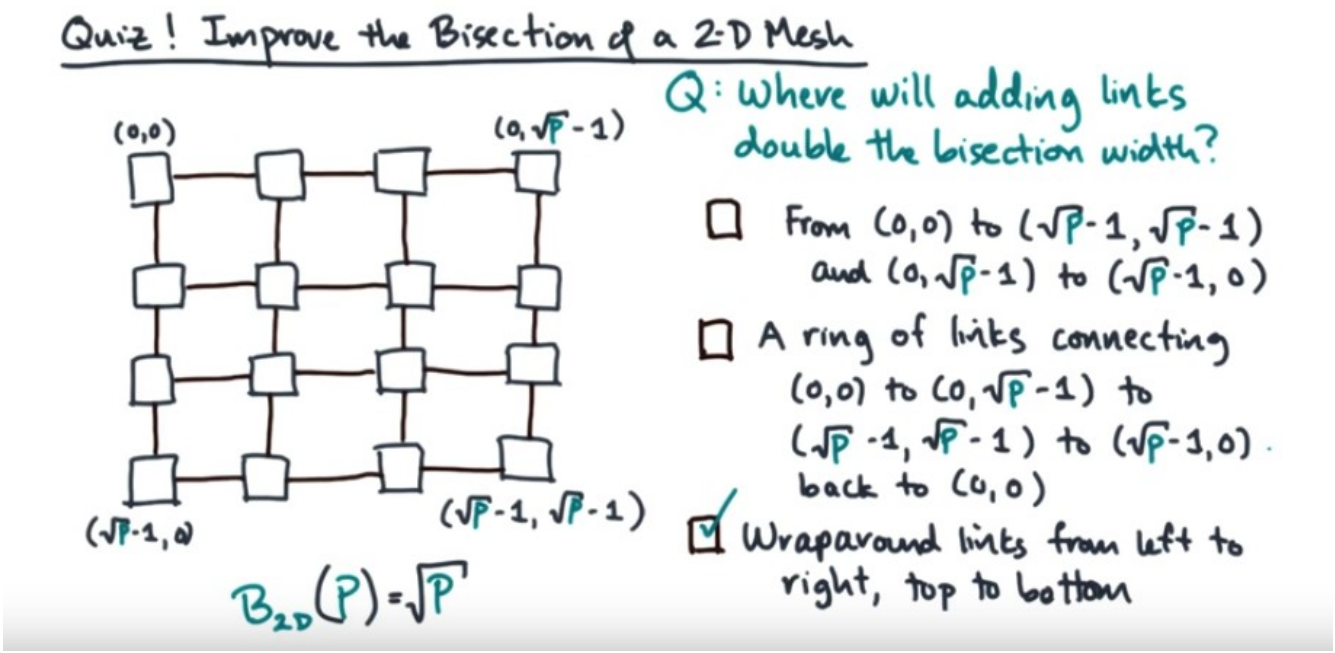
Bisection (Band)Width

$\beta \equiv$ link speed $\left[\frac{\text{words}}{\text{time}}\right]$



$B_{2D}(P) \equiv B_{2D}(P) \cdot \beta$

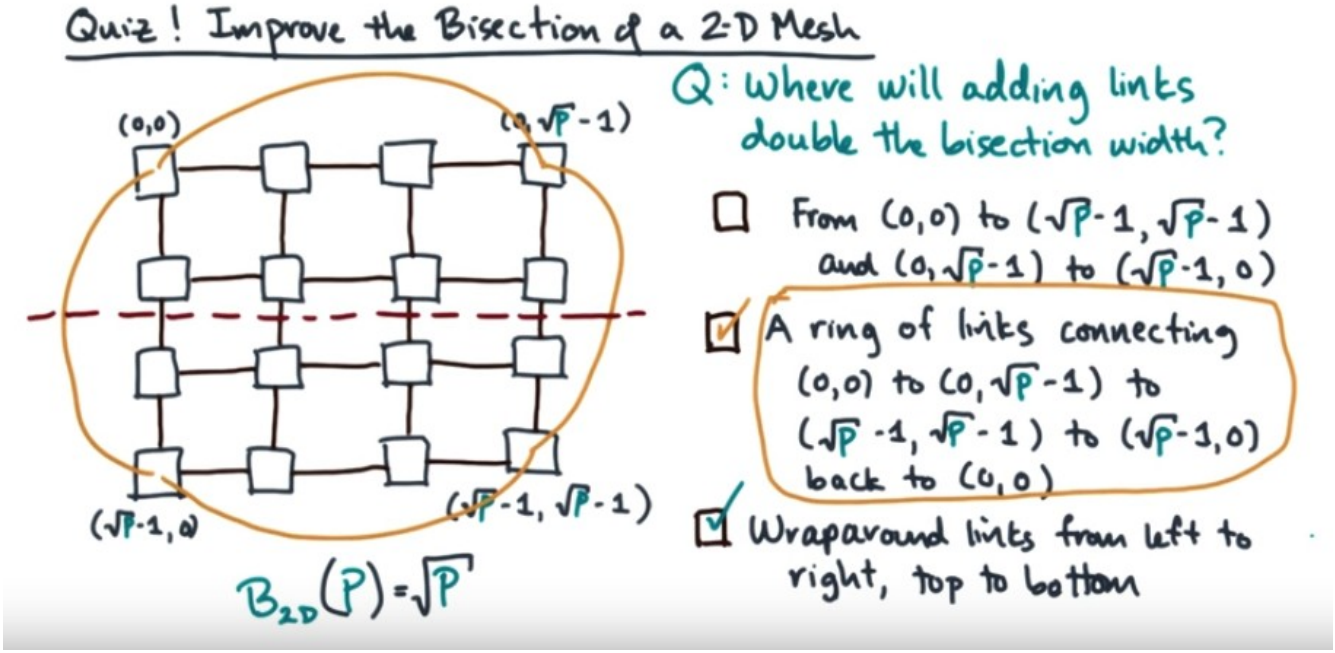
Now, related to the concept of bisection width is the concept of bisection bandwidth. Let's explain this by example. Suppose you have a 2D mesh. Let's say the speed of every link is beta measured in words per unit time. If all the links have the same speed, as in this example, then the bisection bandwidth is just the product of the bisection width with the link bandwidth. In other words, the speed across the bisection is the bisection width times the speed per link. What if you have links with unequal speed? In that case, you would look for a set of links that cut the network in two and have a minimum total bandwidth.



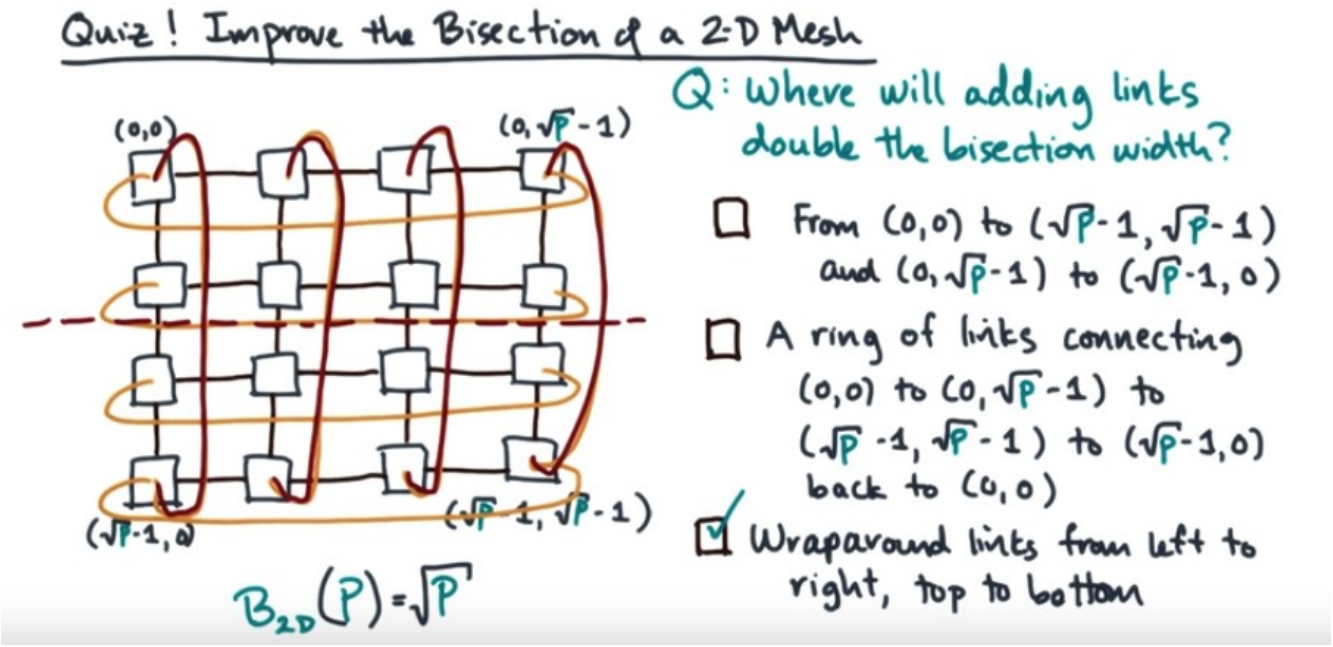
上圖中的幾個選項的意思跟前面那個 quiz 一樣.

8. Suppose you're given this 2D mesh. It's bisection width is square root of P. Suppose you want to double the bisection width. Where do you need to add links? Will connecting the opposite corners double the bisection width? Or will adding a ring of links connecting the corners double the bisection width? Or do you need to add wraparound links from left to right and top to bottom? I want you to check all that apply or check none if you think none of them will work.

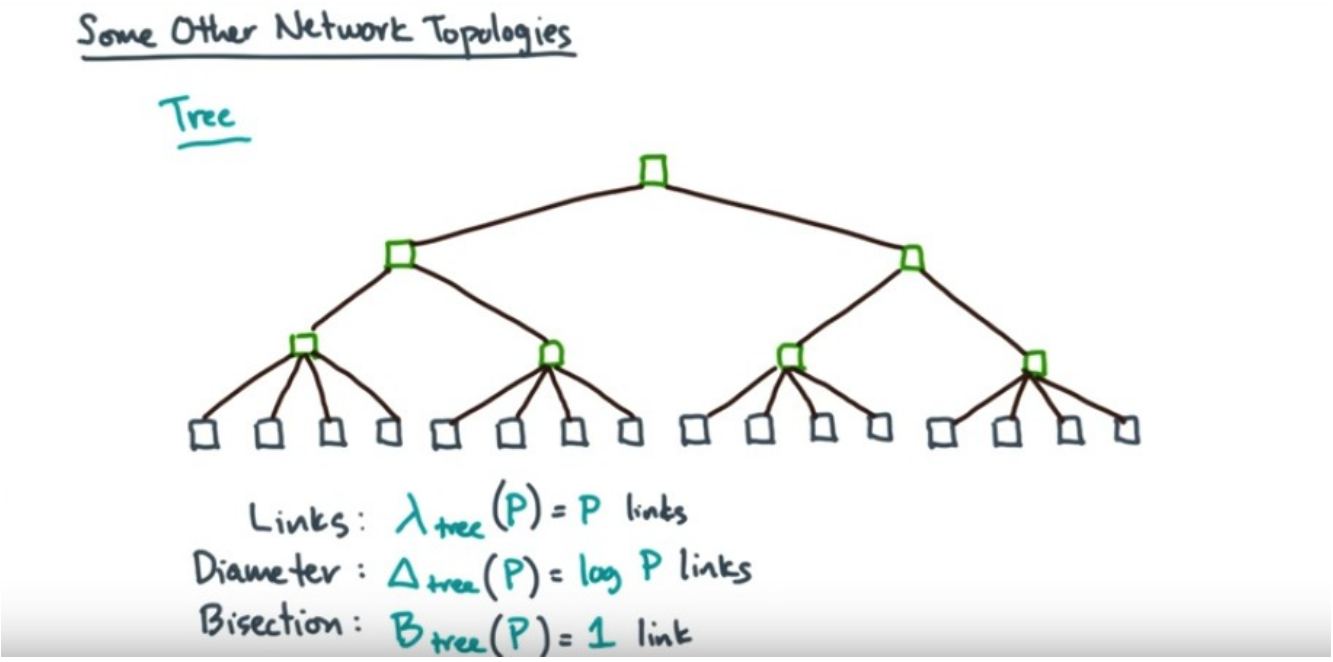
9. Of these options, only the third one doubles the bisection width. The other options reduce the diameter, but they can't asymptotically improve the bisection width.



As an example, let's take option two. Now, the original bisection cut is still a cut of this new network, but notice that the number of links only increases by one, two links. That is, it doesn't double.

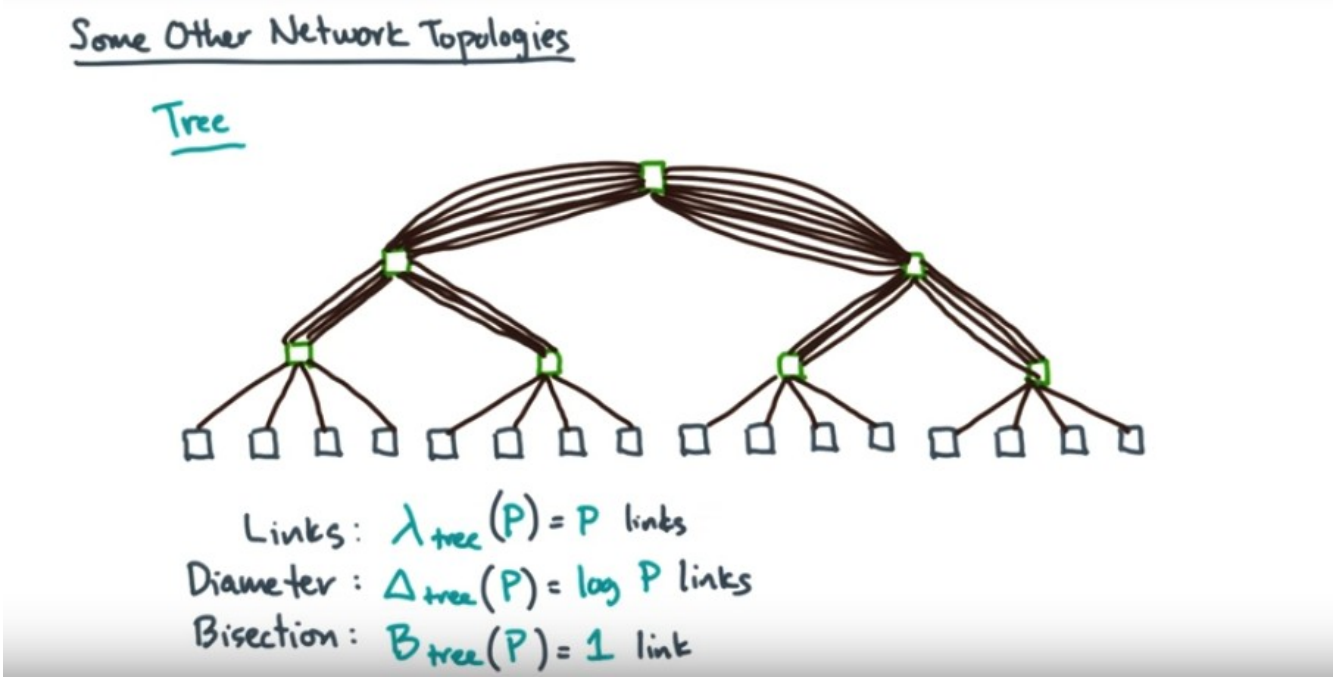


By contrast, consider option three. Notice what happens at the cut. There are in fact, square of P new links that go through this cut. That doubles the bisecting width. Yeah, I'm awesome. Do bulls moo? Moo, moo.

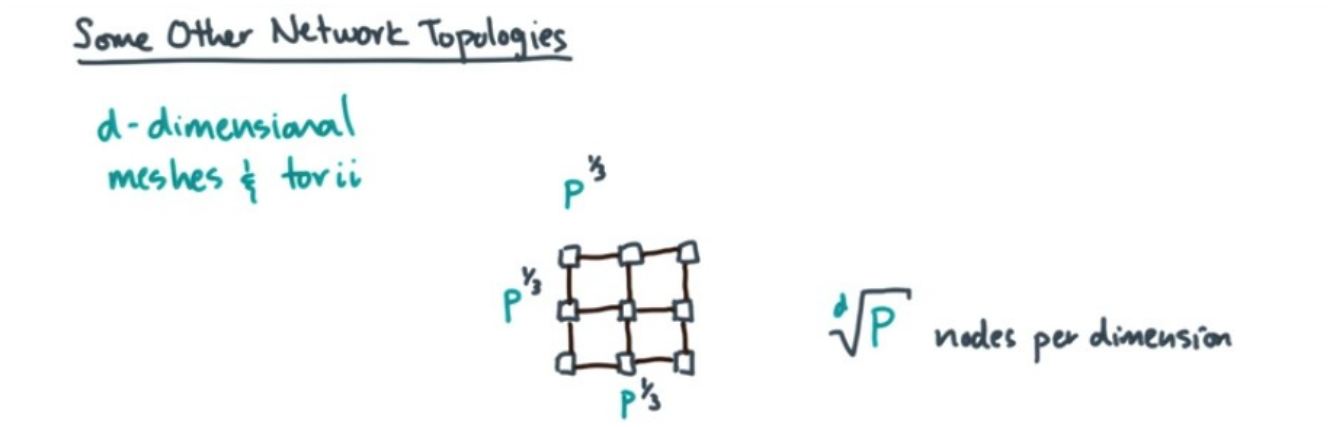


10. In the event you're ever spelunking in the parallel computing literature, you will come across a variety of other kinds of networks besides rings and meshes and torii. I want to mention a few here in passing. The first is a tree network. The compute nodes are the leaves of the tree. The interior nodes are just routers. So you can assume these higher level nodes in the tree are just moving traffic. They

don't do any actual compute. Now in this example, these three nodes at the top are binary, but they needn't be in general. Now, what about the network properties of a tree? A tree with P compute nodes at the leaves will have P links, it'll have a diameter of log P, and it will have a bisection width of just 1 link. The scaling of links and diameter is very good. But the scaling of bisection width being just a small constant is terrible. The bisection width is poor because to cut the network in half, all I have to do is cut one of these links near the top.

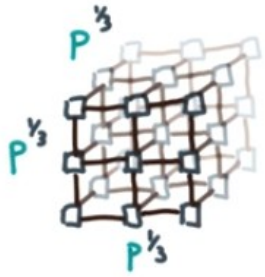


Now in practice, network designers typically fatten up the bandwidth as you move towards the top of the tree. Here's an example. At each higher level in the tree, I put a lot more wires to help carry more traffic. This variation on a tree network is sometimes called a fat tree. Fat trees are quite common in medium scale cluster environments. By medium size, I probably mean on the order of thousands of nodes today.



Some Other Network Topologies

d-dimensional meshes & torii



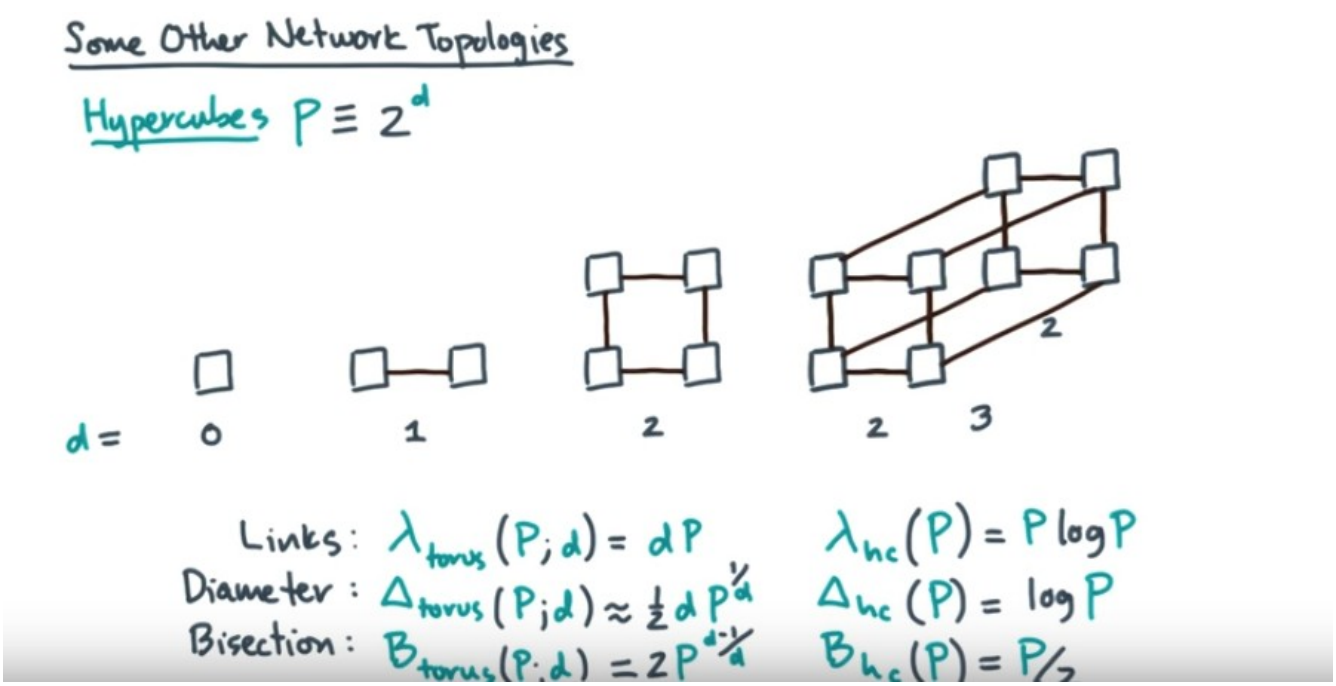
$\sqrt[d]{P}$ nodes per dimension

Links: $\lambda_{torus}(P;d) = dP$

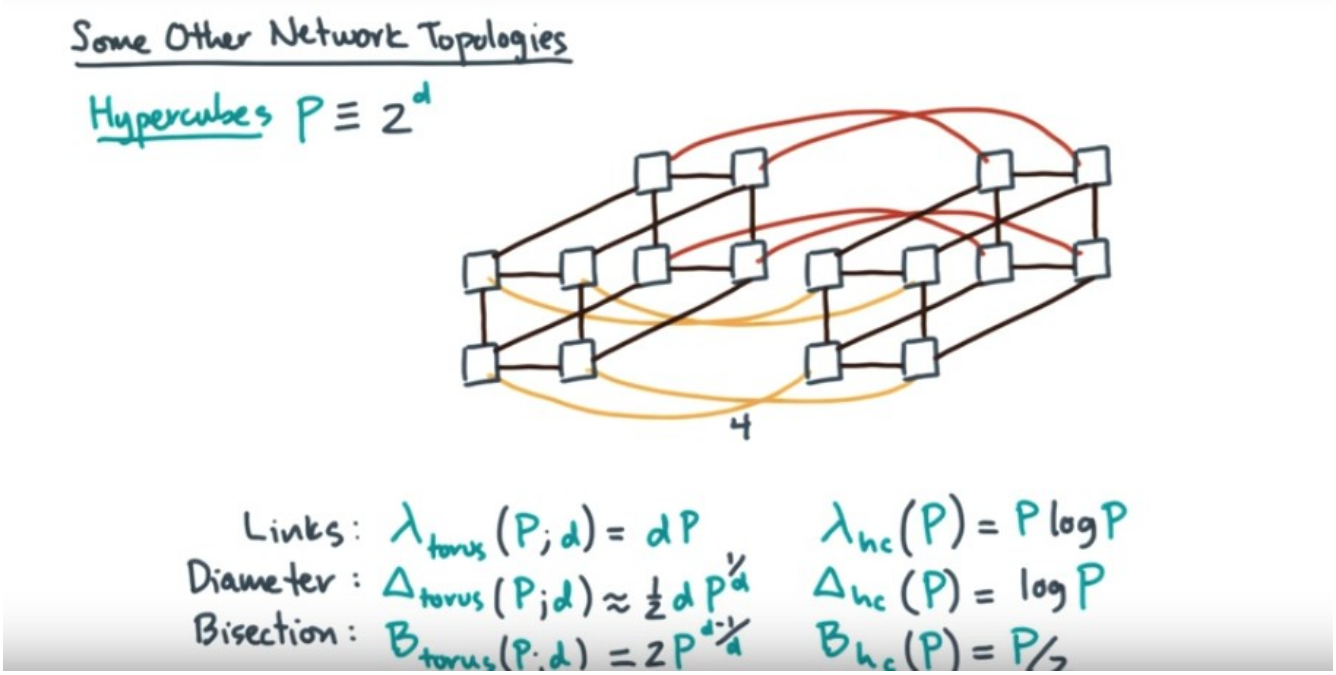
Diameter: $\Delta_{torus}(P;d) \approx \frac{1}{2} d P^{1/d}$

Bisection: $B_{torus}(P;d) = 2P^{d-1/d}$

Another kind of important network is a natural extension of 2D meshes and torii. The extension is to higher dimensions. A d-dimensional measure of torus is basically a high dimensional cube that is, say, the dth root of P nodes per edge. If the object is a torus, then it will have these properties. Naturally, these values depend on the number of dimensions in an interesting way. For example, the diameter decreases by the d through the P. So that's good, because as you increase the dimension, you'll decrease the diameter on the one hand. On the other hand, the diameter also depends linearly on d. So as you increase the number of dimensions, the diameter goes up. Now d-dimensional meshes and torii are very important in practical high end computing systems. In fact, many of the world's top supercomputers use low dimensional toroidal networks. As of the time of this recording, there's even one that has a dimension of six.

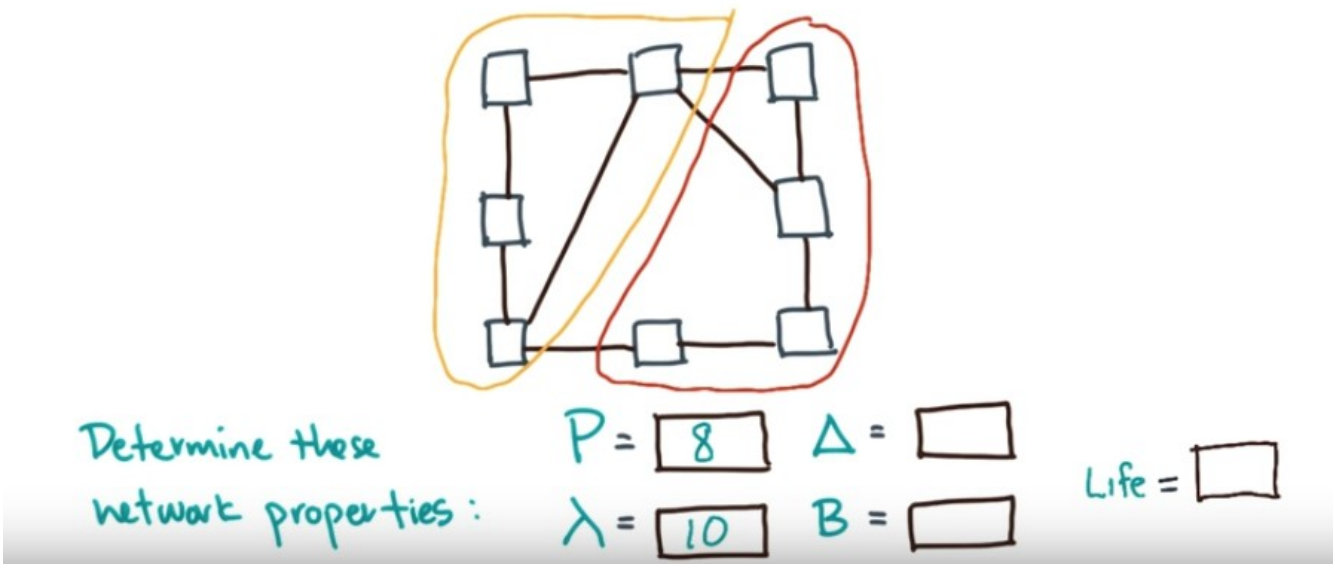


Related to meshes and torii is a type of network called a hypercube. Woah, easy there, fella. Very roughly speaking, a hypercube is a log p dimensional torus. More specifically, a P node hypercube has these properties. It has P log P links. It has a diameter of log P. And it has a bisection width that scales like P. Compare the hypercube and torus formulas. Hypercube is much more expensive in terms of the number of wires, but it has a lower diameter and a much larger bisection width. So, what does a hypercube look like? The easiest way to explain it is by visual construction. A hypercube is parameterized by a dimension little d. The number of nodes is a power of 2 in d. The smallest hypercube occurs when d is equal to 0. It's just a single node. To make a one-dimensional hypercube, start by making two copies of the lower zero dimensional hypercube. So here are two copies. You then connect the corresponding nodes of the two copies, so that's a one-dimensional hypercube. What about d = 2? Again, start by making two copies of the lower, one-dimensional system. Then connect the corresponding nodes of the two copies. For a higher dimensional hypercube, just keep repeating this process. Make two copies, then connect the corresponding nodes.

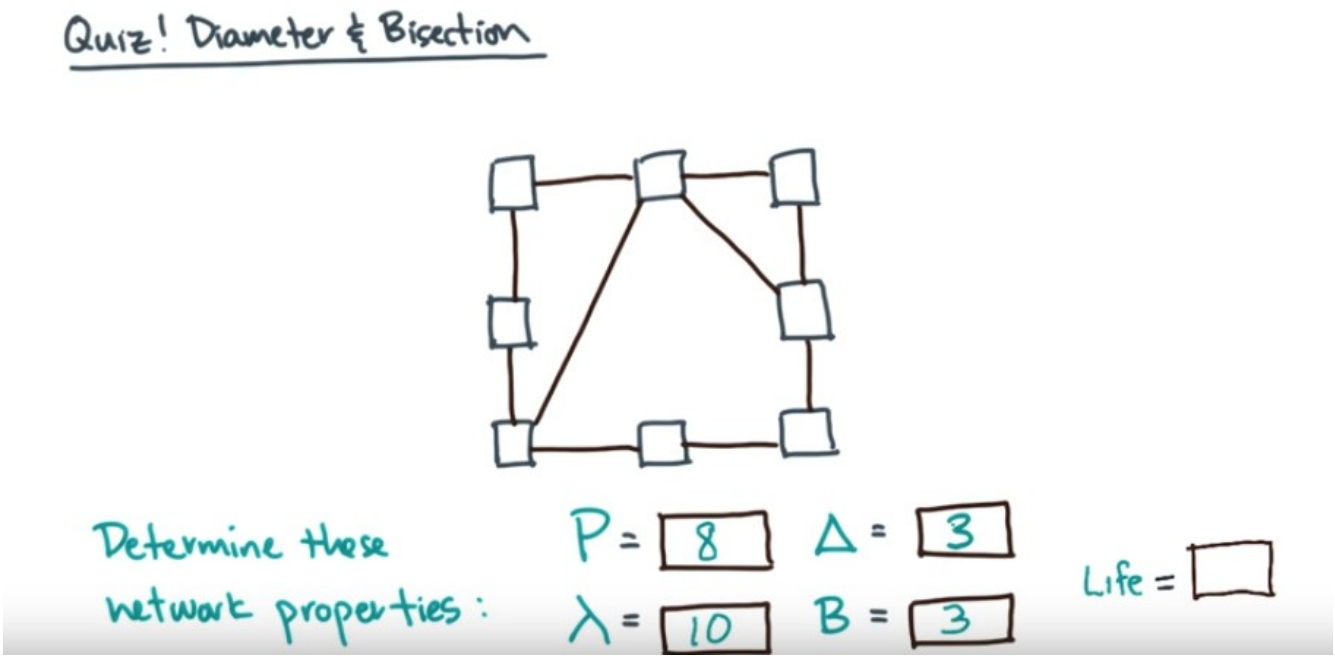


For d = 4, you do the same thing. Two copies, connect the links, voila. >From this construction process, I hope you can easily see that the bisection width is just P/2. That's because you just build the larger network by connecting the corresponding nodes of the smaller network, each of which is half the size. Now these days a hypercube network is more of a historical intellectual curiosity than something people actually build. There's a lot of elegant theory around them, and you'll see lots of algorithms designed for them in the literature. Now these have just been a few examples of network topologies, which is a very active area of research. I'll put some more references to some recent ideas in the instructor's notes.

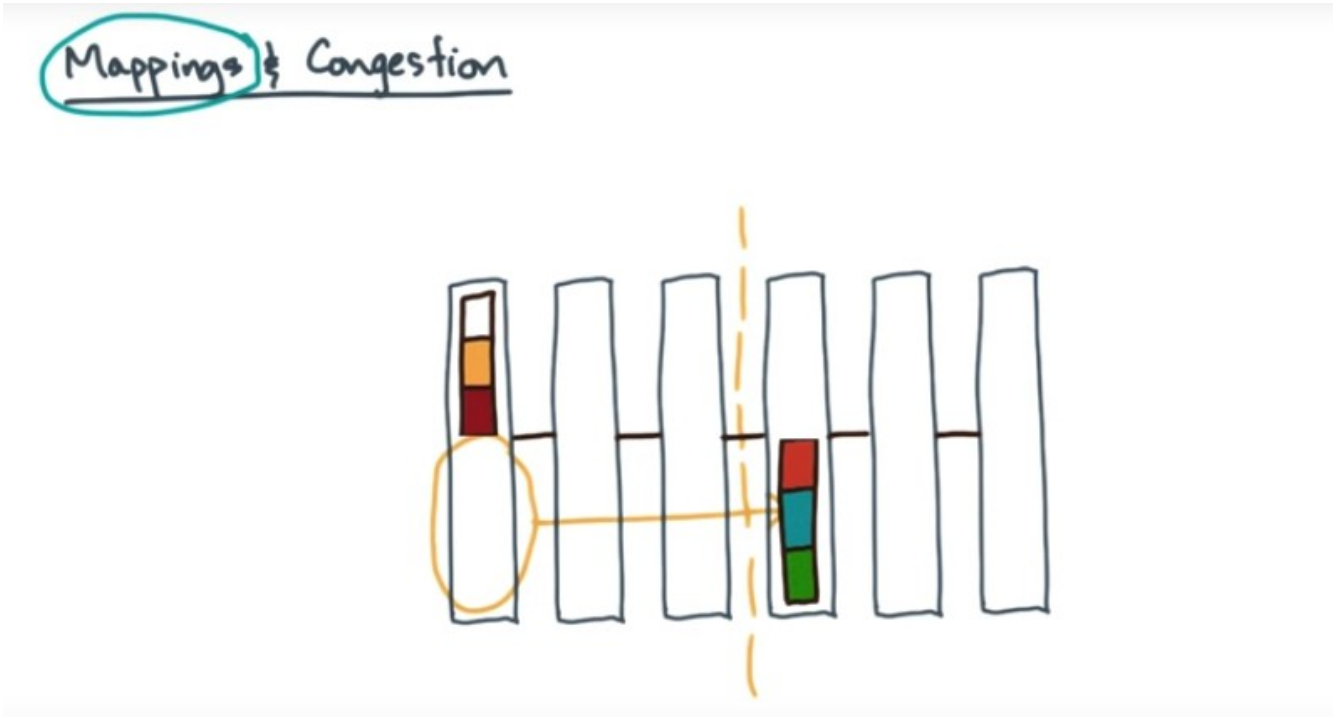
11. Consider this network, I have a bunch of questions for you. How many nodes does it have? How many links does it have? What's its diameter? What is its bisection width, and what, pray tell, is the meaning of life? Type your answers as integers in the boxes.



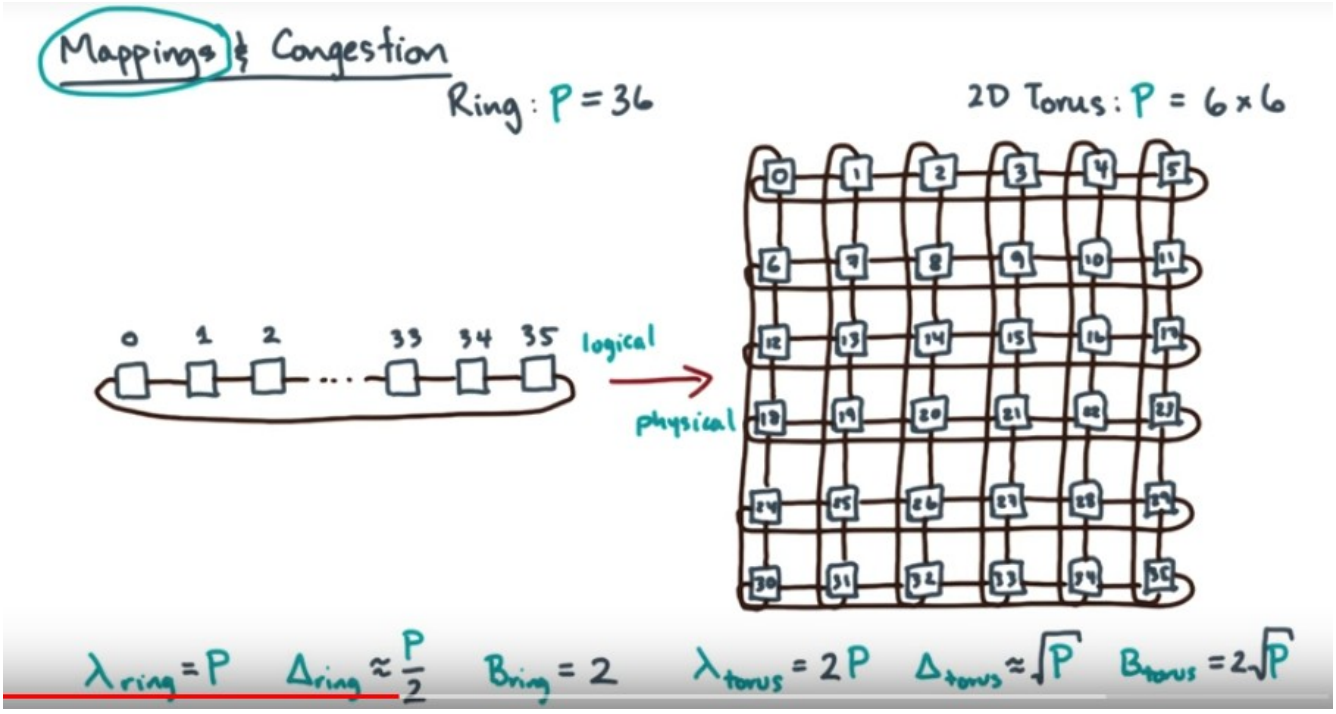
12. Let's start with a number of nodes. There are eight nodes. What about links? There are ten links. What about bisection? You need to divide the eight nodes into two subsets of size four each. There are many possible cuts, let me show you one. This partitioning cuts one, two, three edges. As it happens, all other partitionings will cut at least three edges as well, therefore the bisection width is three.



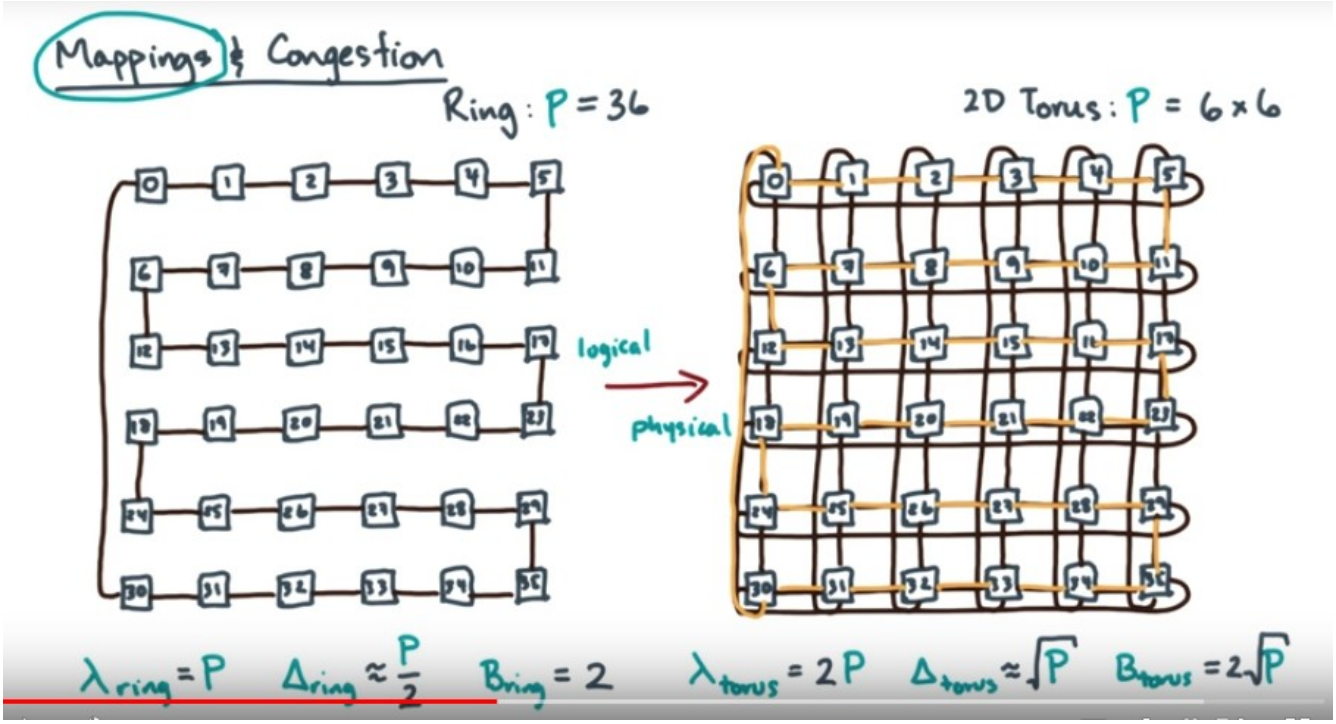
What about diameter? For any pair of nodes you can check that the shortest path between them is never more than three. For example, consider the corners. I'll let you check all other pairs. What you should find is that the diameter is three.



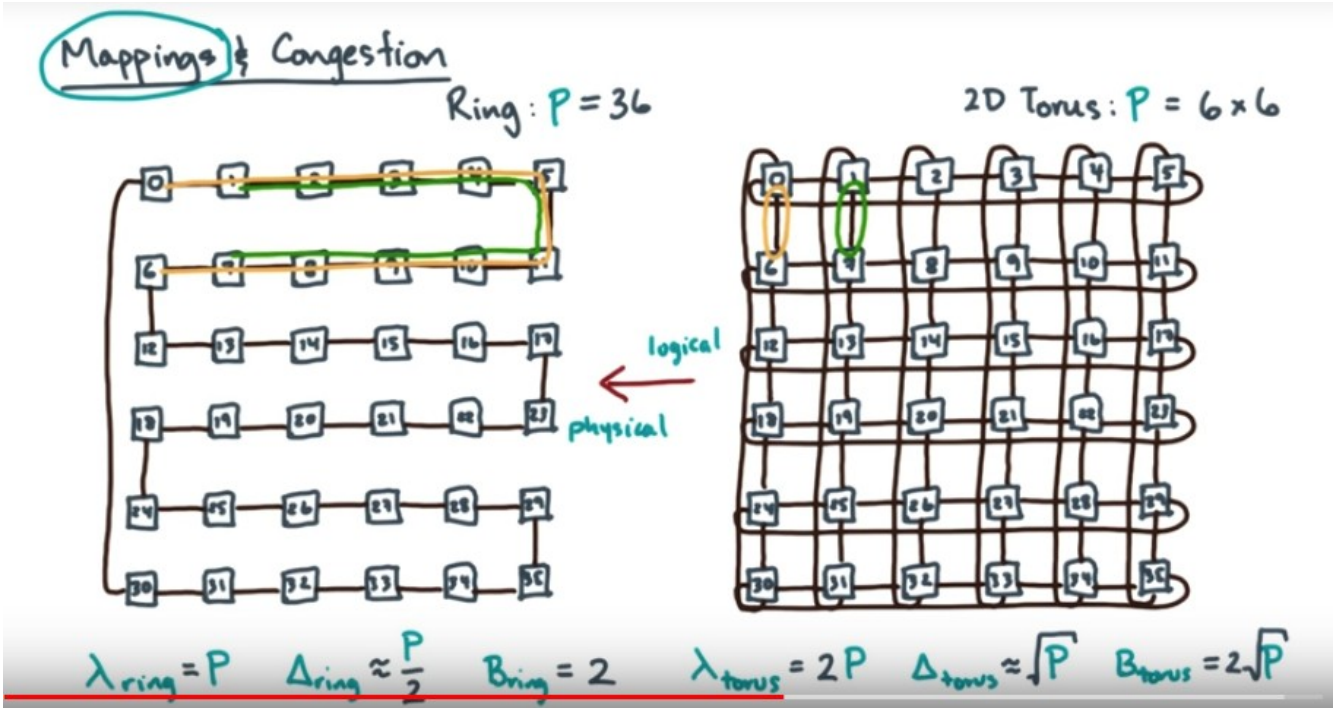
13. When you design an algorithm for a distributed memory system, the communication pattern of your algorithm implies a network. For example, you designed algorithms for scatters and gathers that worked well on a linear network. By working well, I mean concurrent messages never overlapped, and so would not contend for links. Now, suppose you design an algorithm for a linear network, but then decide to run it on something else, like a Mesh or Fat Tree. How well would it work? To answer this question, you need to map a logical (意思馬上見下圖) network, to the physical one.



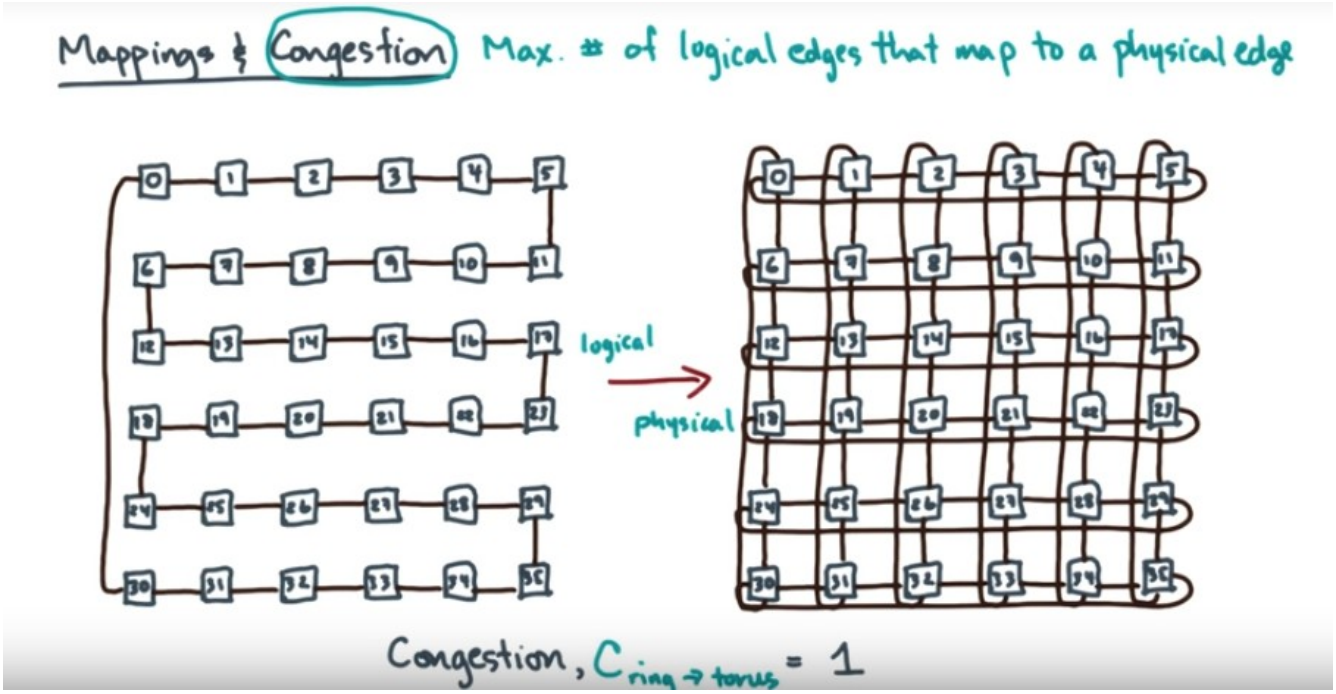
Consider an example. Suppose you design an algorithm assuming this network. It's a ring with, in general, p nodes. Diameter of p over 2 and a bisection width of 2. Now suppose the underlying machine is physically a 2D Torus. You first need to decide how to assign nodes of this network, to nodes of this network. Let's use a very natural row by row numbering. So the first 6 nodes of the ring go here. The second 6 nodes go here. And, so on, row by row.



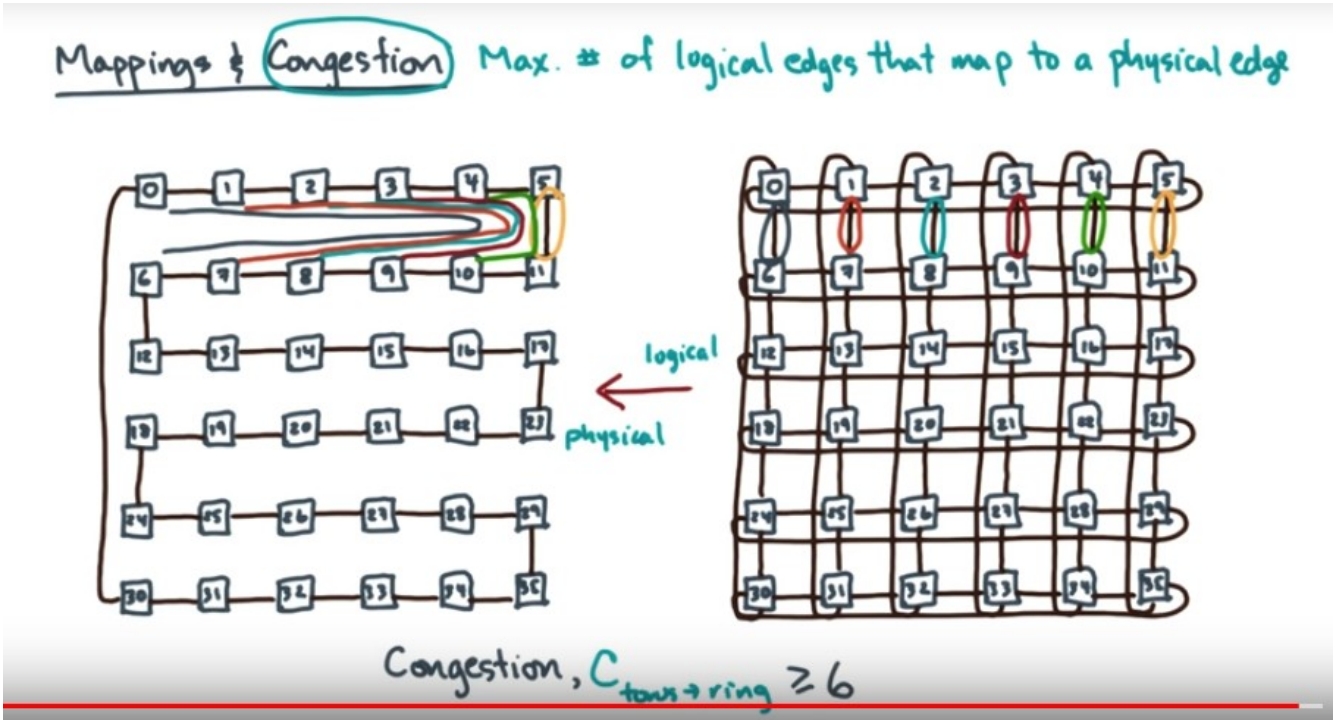
In fact, let's redraw the linear network, according to this numbering scheme. Notice, that the edges of the 2D Torus, are a superset of the edges of the ring. So, what does that mean? If there's no contention for length in the linear network, then there will be no contention for links in the torus either, at least not under this mapping.



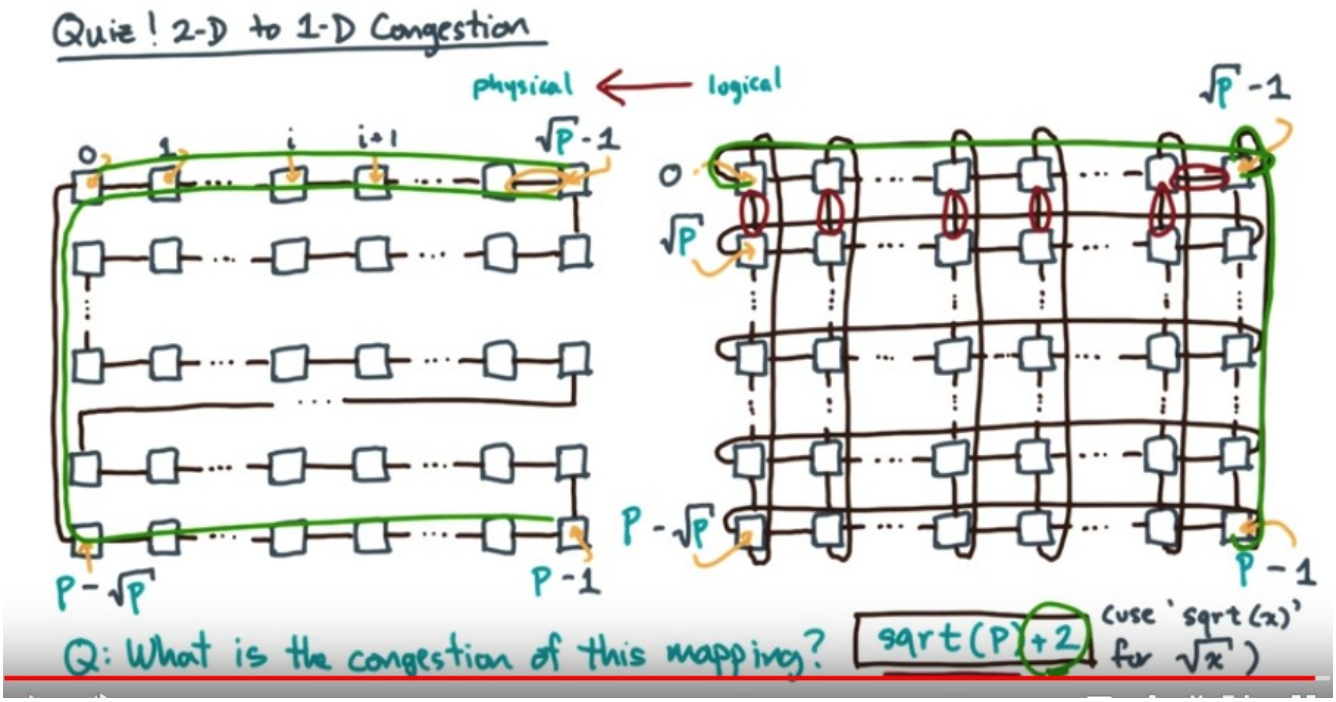
So, what if you did the reverse? That is, what if this is the logical network, and this is the physical one? Intuitively, there are a lot more links over here, than there are over here. So even if there's no contention in the torrus, there might be a lot of contention in the ring. To see what would happen, let's start with the same node numbering scheme. Let's look at a couple of edges. For example, let's consider this edge from 0 to 6. In the ring, where's the shortest path between 0 and 6? It's here. Now, let's take a different edge. How about 1 to 7 in the 2D Torus? In the ring, it must go along this path. So what if messages, from 0 to 6 and 1 to 7, are trying to go at the same time? Then the messages might have to serialize, because the paths overlap.



To help model this problem, let me introduce a new property of the mapping. This property is called **congestion**. Achoo. I'll define congestion as follows. Given a logical to physical mapping, the congestion of the mapping, is the maximum number of logical edges that map, to a given physical edge. That's a mouthful. For example, suppose the logical network is a ring. Suppose the physical network is a 2D Torus. Suppose the mapping is this row by row numbering. But suppose that a path, in the logical network, maps two a shortest path in the physical network. You already argue that there's a one to one mapping, from each logical edge to each physical edge. Because of this one to one correspondence, the congestion of a ring, going to a torus, is just one.

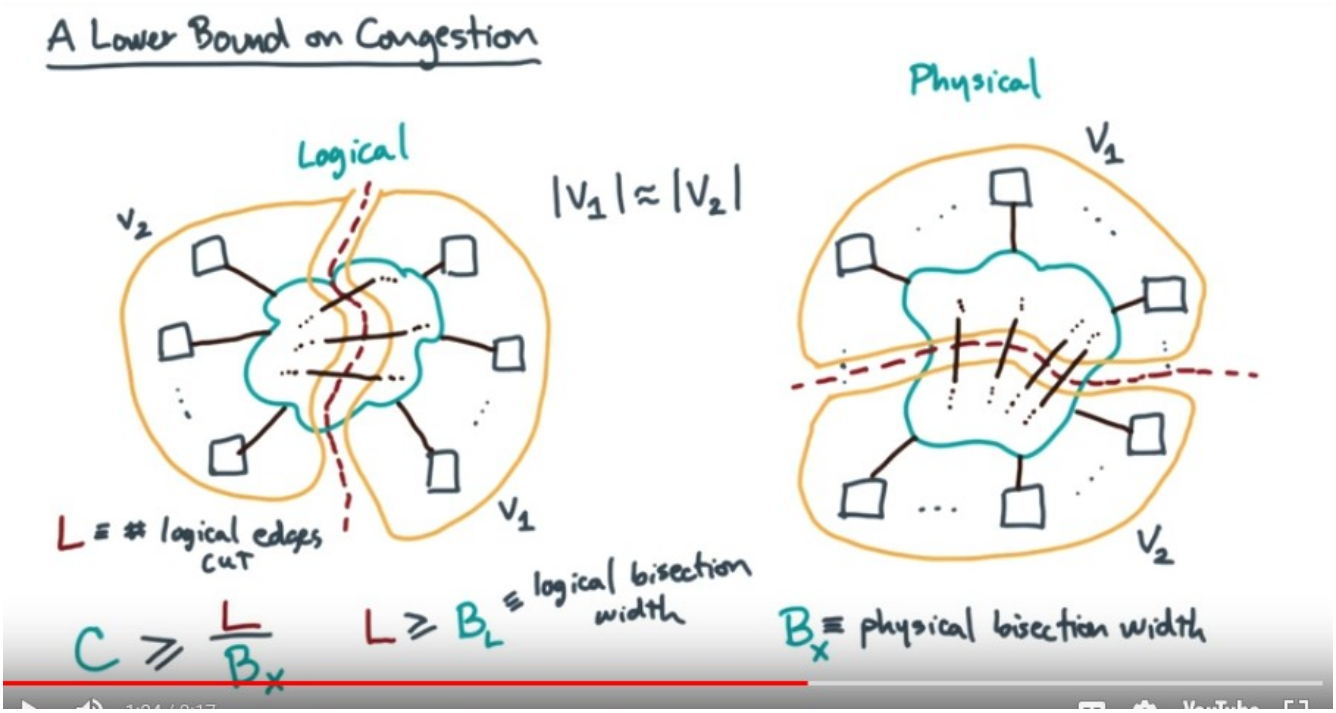


Let's consider the other direction, with the same node to node mapping. That is suppose the 2D Torus is logical, and the ring is physical. Now let's pick an edge. How about this one? Now this edge also appears on the logical network. But remember your earlier analysis? What if you take this edge from 4 to 10? Or this edge from 3 to 9? These map to shortest paths that intersect the 5 to 11 edge in the physical network. In fact, you can keep going. All of these edges in this row have to go through the 5 to 11 edge. So, we have so far, that at least 6 logical edges map to this 1 physical edge. Therefore, we have that the congestion has to be at least 6.



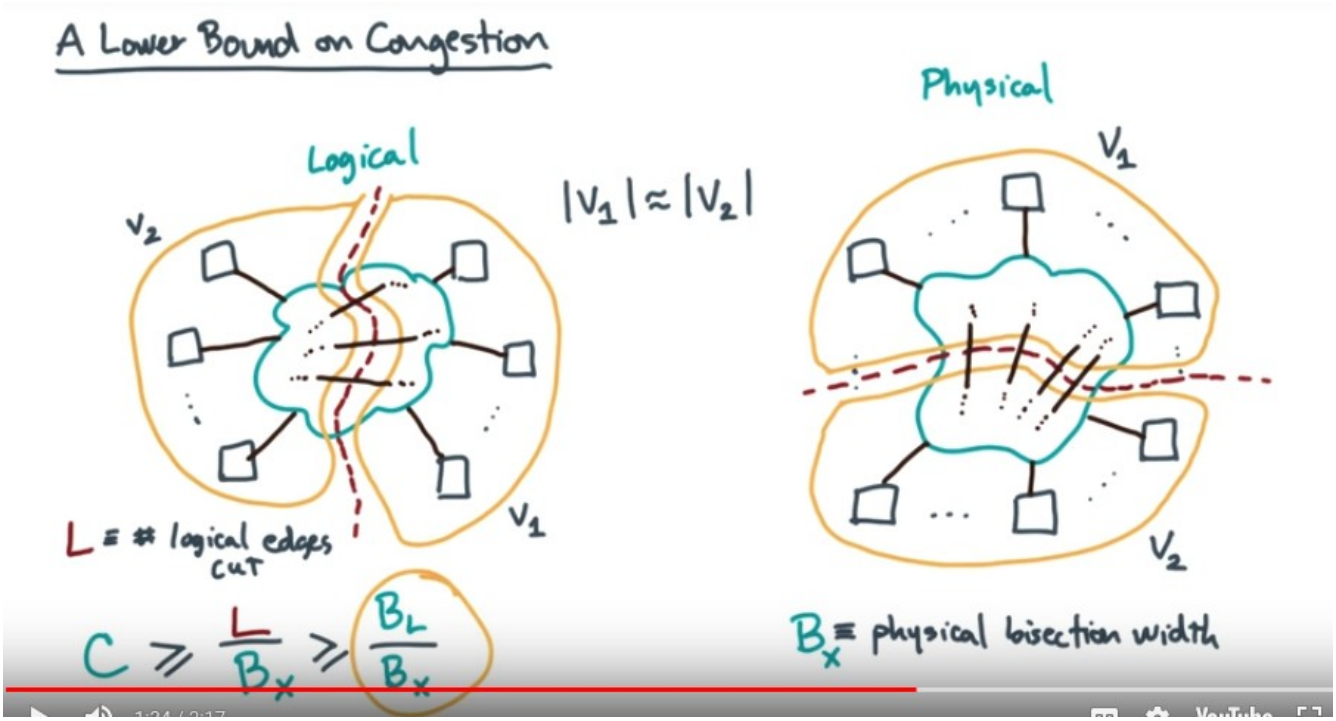
14. Consider the following general case of a mapping. The logical network is a 2-D Taurus. The physical network is a 1-D ring. The numbering of mapping is as shown. Here's my question. What is the congestion of this mapping? I want you to type your symbolic answer here. If you need to put in the symbol square root of x, just type sqrt (x).

15. The answer is sqrt(P)+2. Let's take an edge, I don't know, how about this one (左圖右上角中的黃圈)? Let's see which logical edges map to this physical edge. There are square root of P edges that come from the matching edge as well as all of these vertical edges, but that's not all. There are also these pesky wraparound edges (右圖中的綠線). This wraparound edge corresponds to this path. And what about the other wraparound edge? The shortest path for this wraparound edge would start here, go along the first row, down this wraparound edge in the ring, and back across the bottom row. So notice that it also has to touch this edge. So the wraparound edges give me the + 2. If you were off by one or two, the auto-grader should have forgiven you. Asymptotically, the important point is that congestion grows by square root of P.

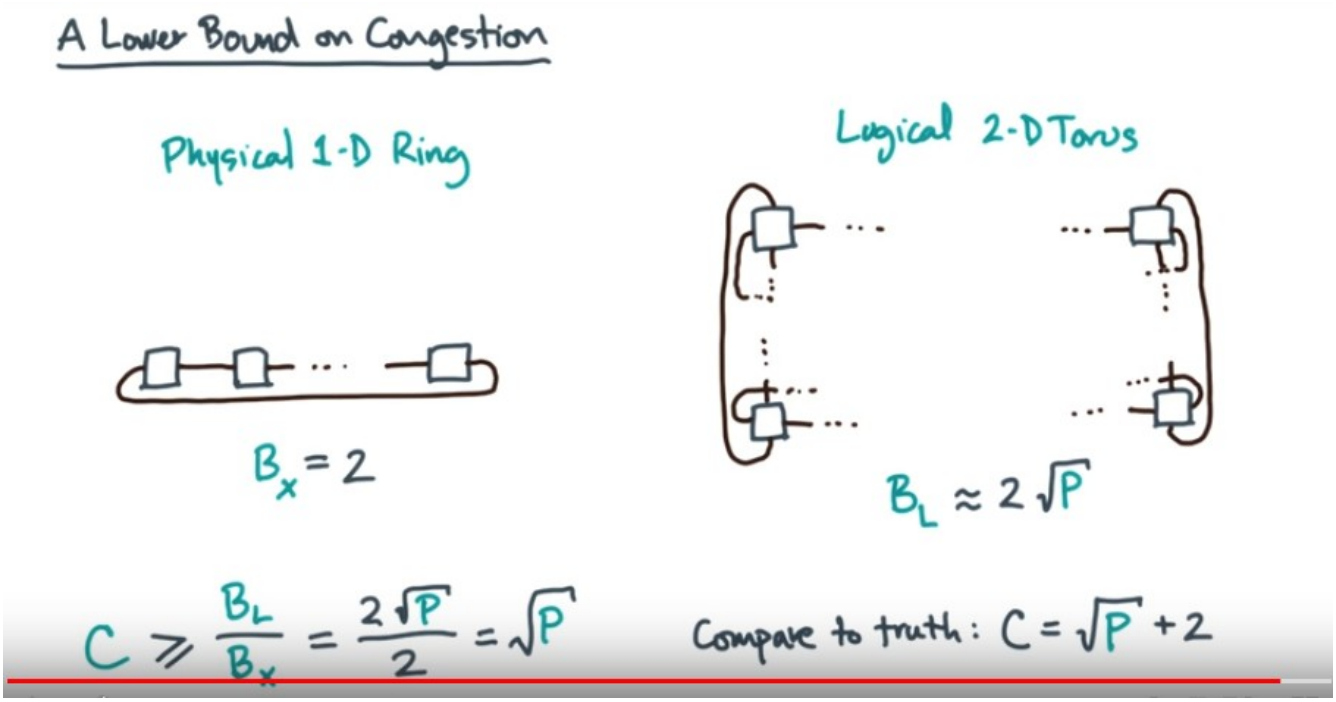


對 L>=B_L 的理解可以見上上圖.

16. I have a confession, picking a mapping and counting edges to compute congestion can make you more than a little crazy. Eep. Let me help you out. Here's a shortcut to estimate a lower bound on congestion. Consider two hypothetical networks, a logical and a physical one. Now suppose you find a bisection in the physical network. By the definition of bisection width this bisection will divide the network into two pieces of roughly the same size. Here I am using V1 to represent the nodes of one half. And V2 to represent the nodes of the other half. Now the bisection cut will also go through a bunch of links. The number of such links is the bisection width. Let's call that **B_X**. Now bisecting the physical network implies a cut of the logical network as well. So schematically, it might look like this. Now the cut in the logical network will go through some number of edges. Let's call the number of such edges on the logical network capital **L**. So in other words, what we're saying is that there are at least L, logical edges that map to B_X, physical edges. So the congestion of the mapping has to be at least L divided by B. We can say more. The minimum possible value that L could have is the bisection width of the logical network (**B_L**). That's because the size of the cut that splits the network in two has to be at least the bisection width.



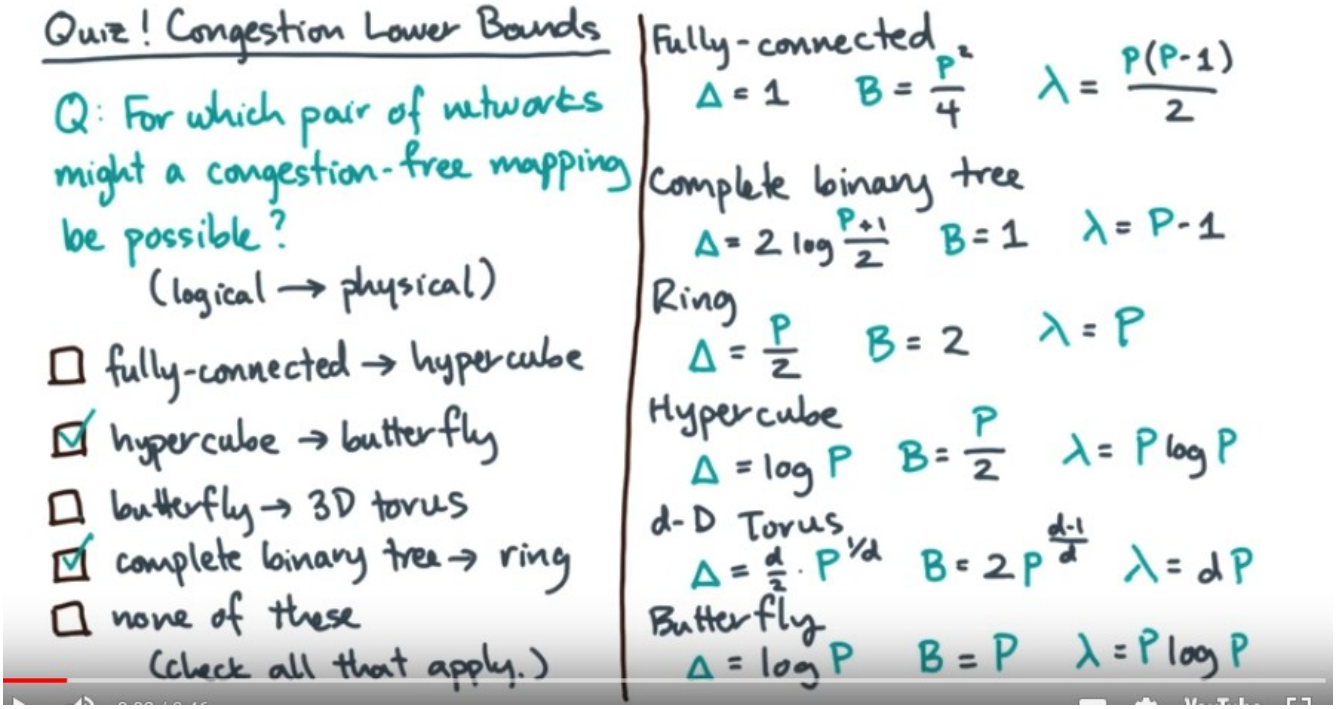
Again, that follows from the definition of bisection width, therefore a lower bound on the congestion is the ratio of the logical bisection width to the physical bisection width.



Neat! Let's see if this works. Lets go back to the 1-D ring and 2-D taurus example. So let's say we start with a logical 2-D torus. And suppose we map that to a physical 1-D ring. Do you remember the bisection widths of these two networks? They are 2 edges for the ring and approximately 2 root P for the Torus. Let's plug these in. What you'll get is square root of P. Now the true value of the congestion of this mapping is square root of P + 2. So you can see the lower bound in this case is pretty good.

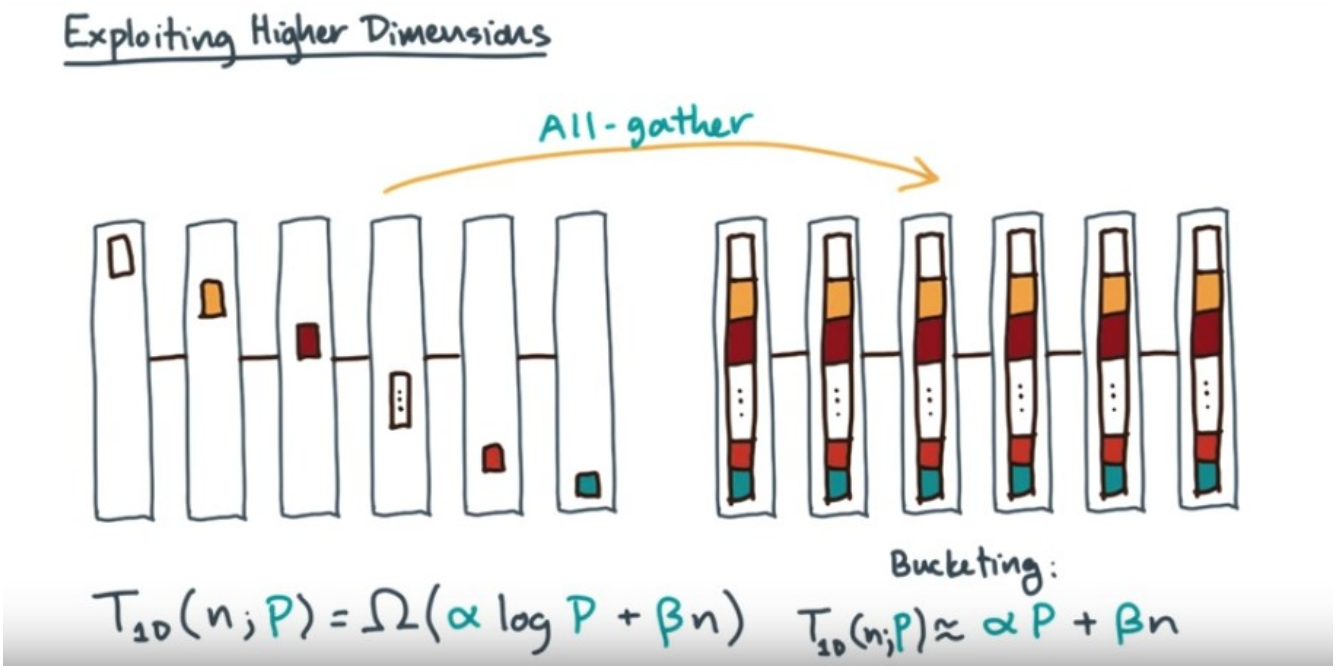
Here's what knowing the congestion allows you to do. You can estimate how much worse the cost of your algorithm might be, if you run a physical network, that has less bisection capacity, than your logical network.

17. Now I want you to apply the lower bound on congestion, consider this list of networks. For each network I give you the network's diameter, the network's bisection width, and the number of links in the network assuming p nodes. Now suppose you wish to map a logical network onto a physical one. Here's a list of pairs of mappings. The pairs go from logical on the left hand side, to physical on the right hand side. My question is, which of the following pairs would you expect that it might be possible to have no congestion, assuming the lower bound analysis?

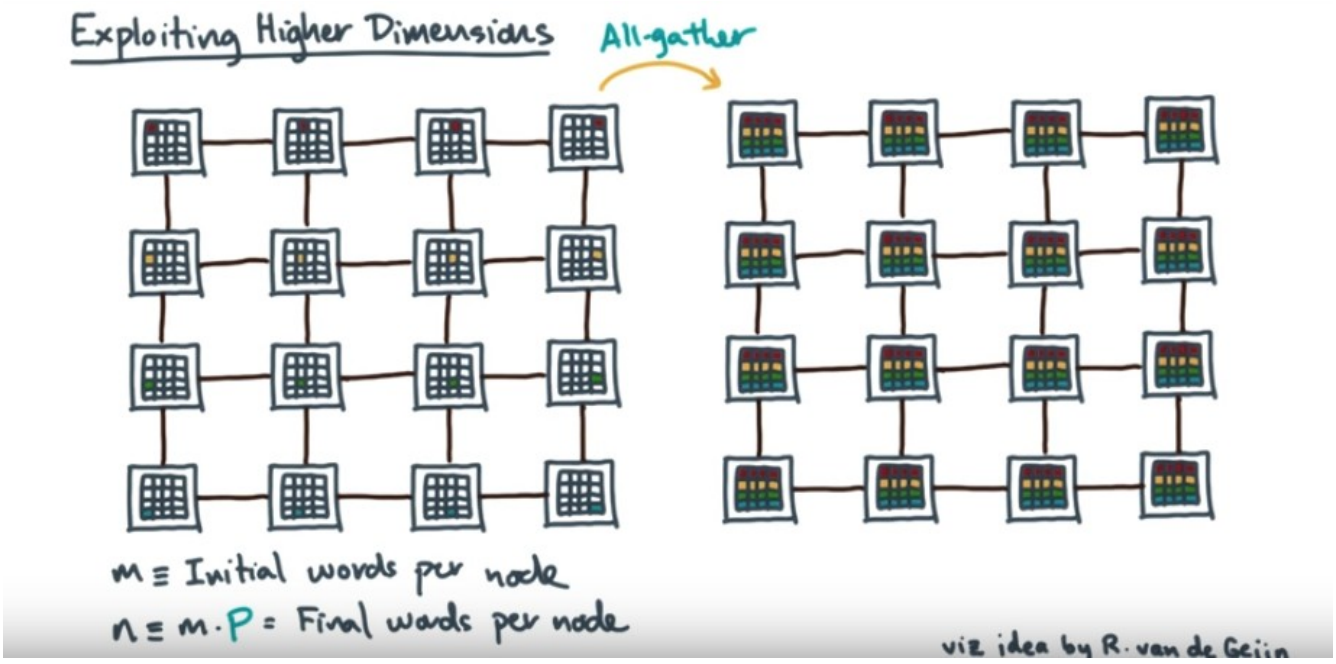


(本題只會用到 B 的值)

18. Here's the answer. The lower bound analysis says you take the bisection width of logical network and divide it by the bisection width of the physical network. So for example, let's take fully-connected to hypercube. Fully-connected is o of p squared. Hypercube is o of p, so the ratio of fully-connected to hypercube is p. So the lower bound suggests you should expect some congestion. What about complete binary tree to ring? Complete binary tree has a bisection of 1, and ring has a bisection of 2. So the ratio, one half is less than or equal to 1. That tells us there might be a way to do a congestion free mapping from complete binary tree to ring. So you can do the same exercise on all the options.

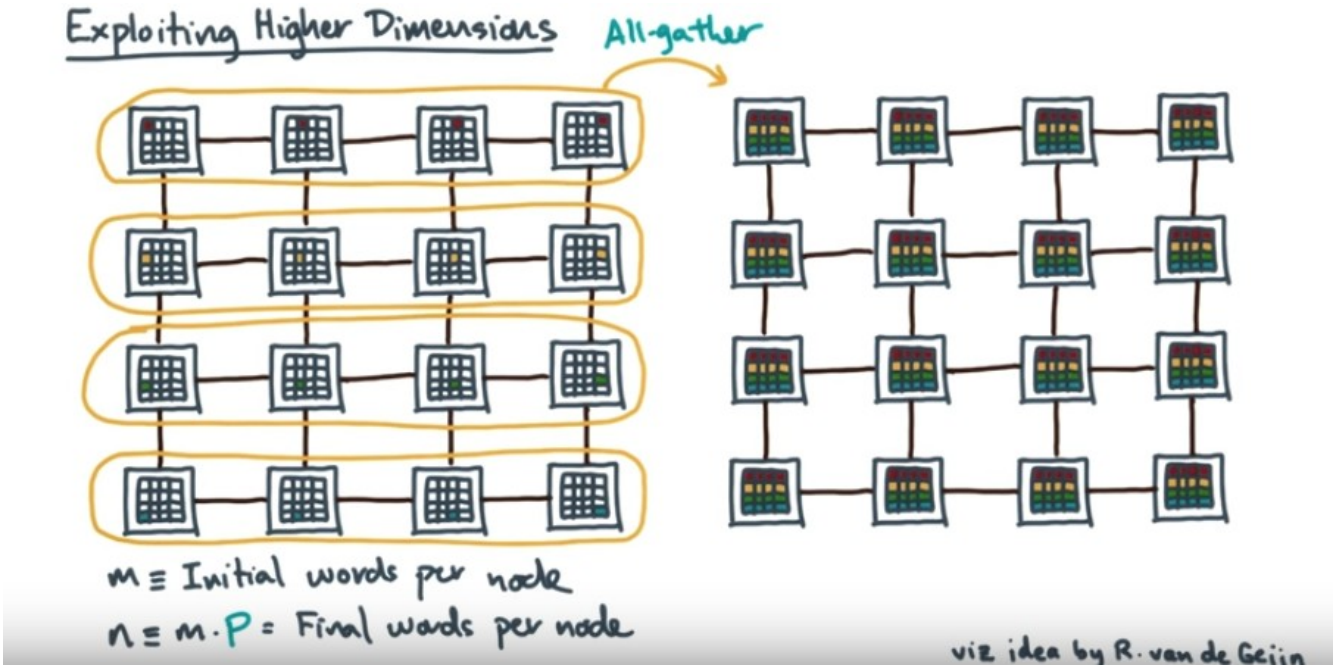


19. One cool thing about a higher dimensional network is figuring out how to exploit it both for fun and profit. Let's start with a low dimensional example. Suppose we're doing an all-gather on a linear network. Here's the before and the after picture. This is a lower bound on the execution time. n, in this case, is the length of each output vector. Now, if you did the all-gather using a bucketing scheme, then you would find its execution time to be approximately alpha times P + beta n. So this matches the lower bound beta, but not in alpha.

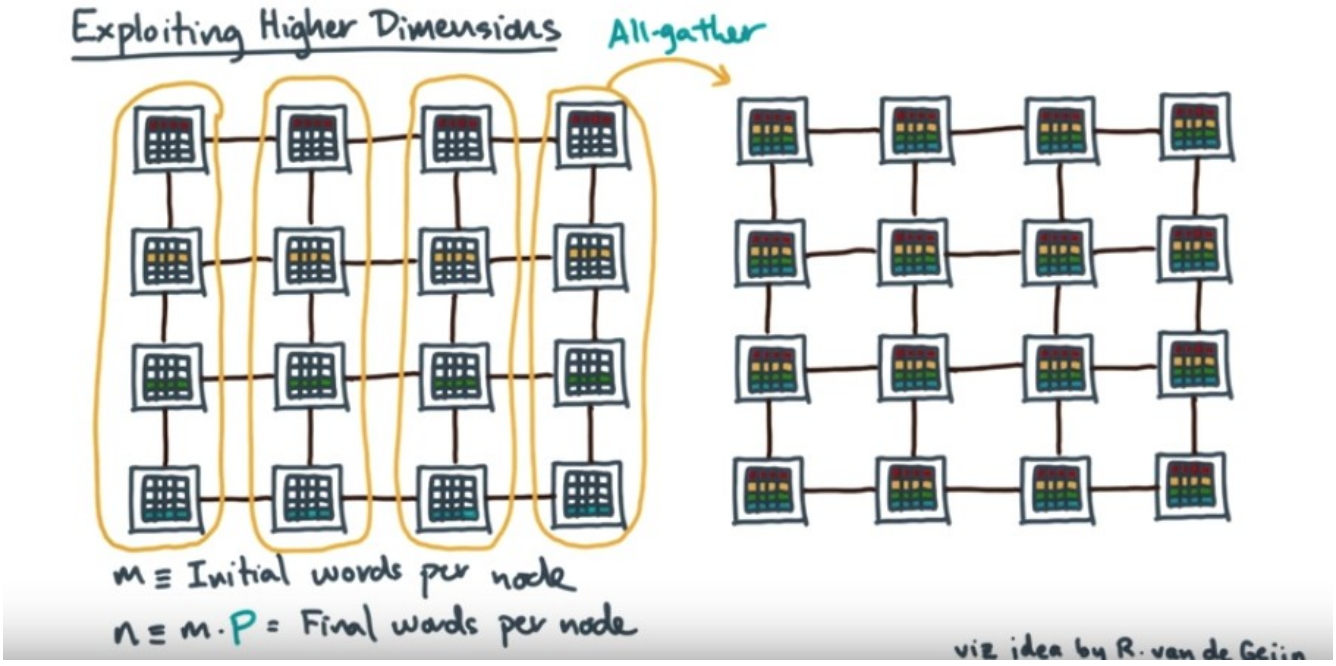


Here's a question. If I gave you a higher dimensional mesh, could you do better? Let's take a 2D mesh as an example. Initially, suppose there are m words per node. So, since this is an all-gather on every node, we have the capacity to store all of the words from all of the processes, but, initially, we only have one little piece. Now, if you do an all-gather, here's the output. That is, when it completes, all

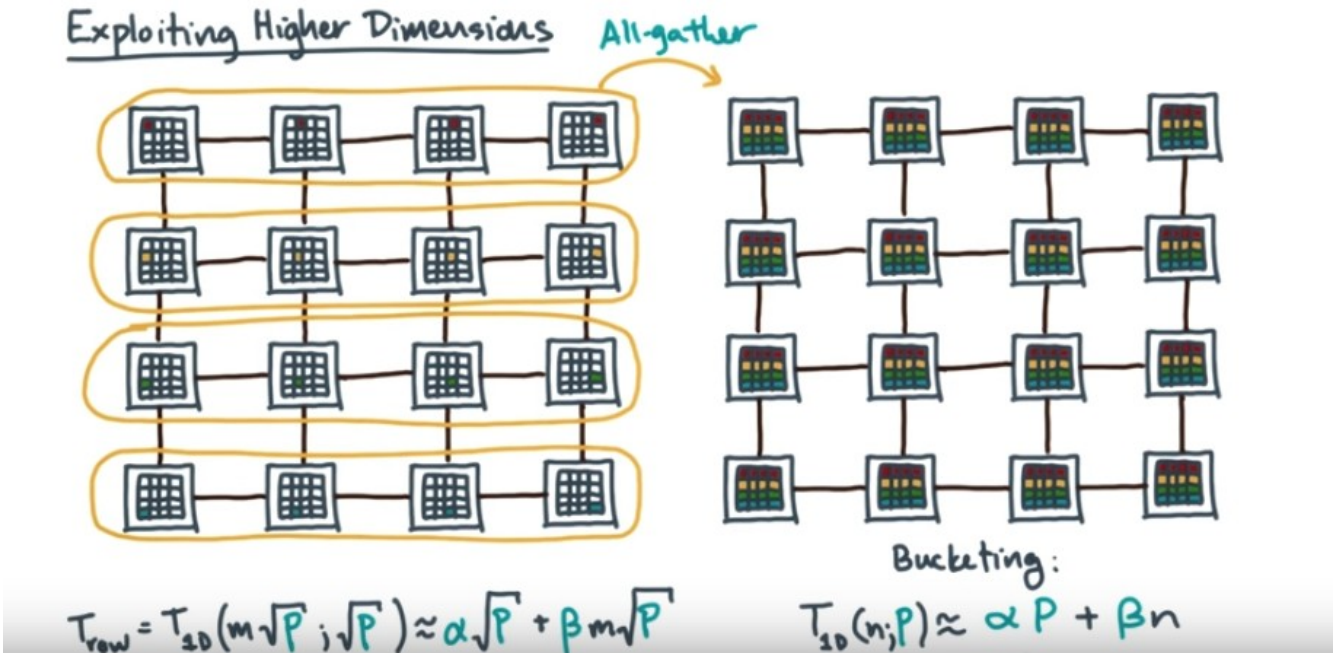
nodes have a copy of all the data. So how do you implement this operation? You could certainly run the 1D linear algorithm. The question is, can you exploit the extra capacity that comes from extra links?



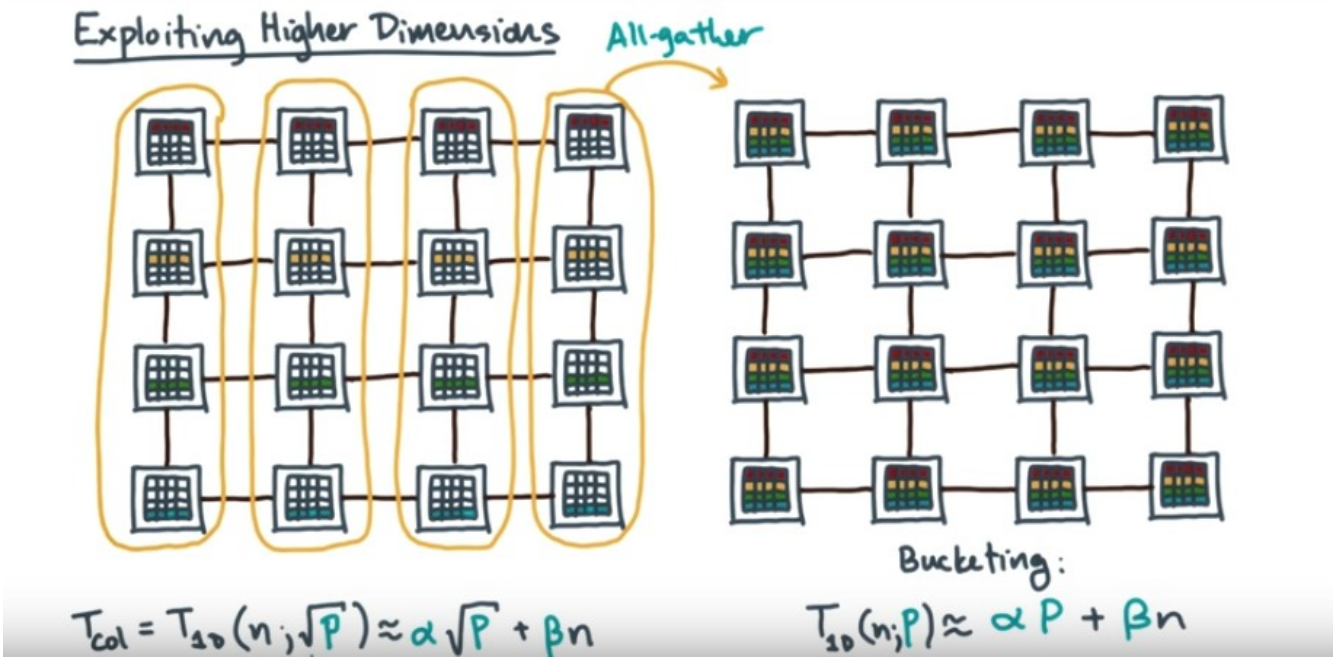
Here's an idea. Why don't you start by doing a 1D all-gather within each process row? When that completes, each node will have a complete row of data.



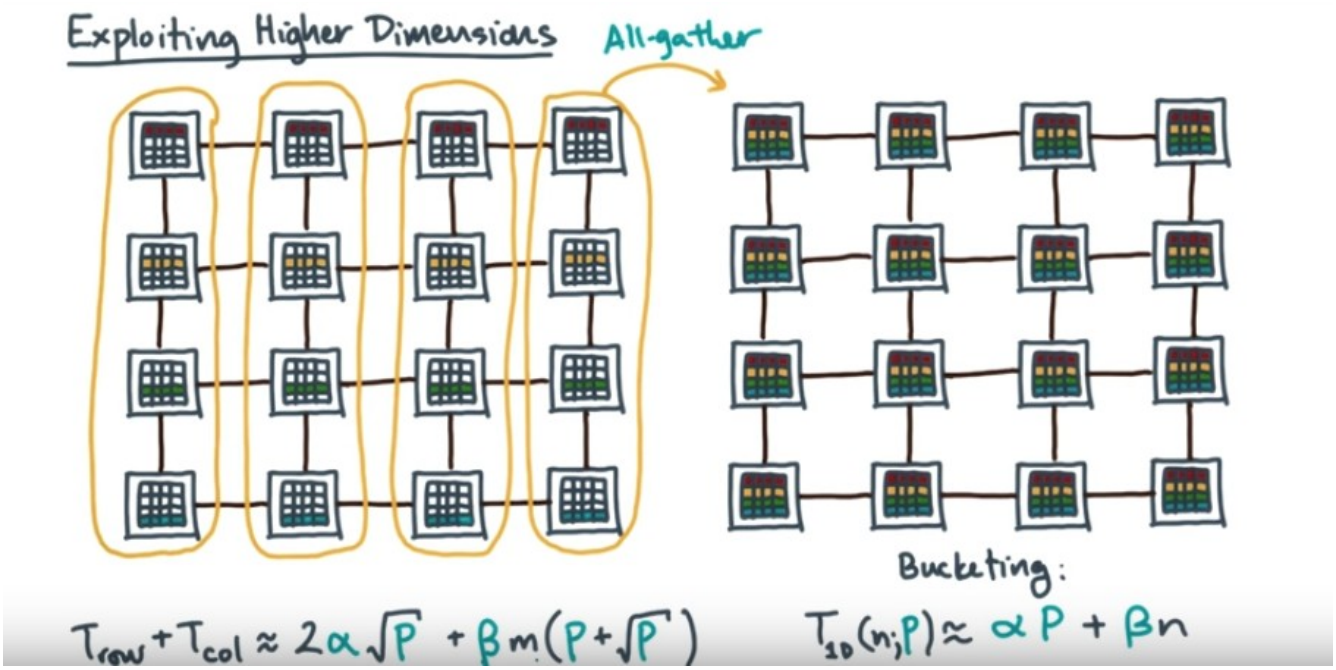
Now do an all-gather, but this time within each column. When it completes, you will have the final output.



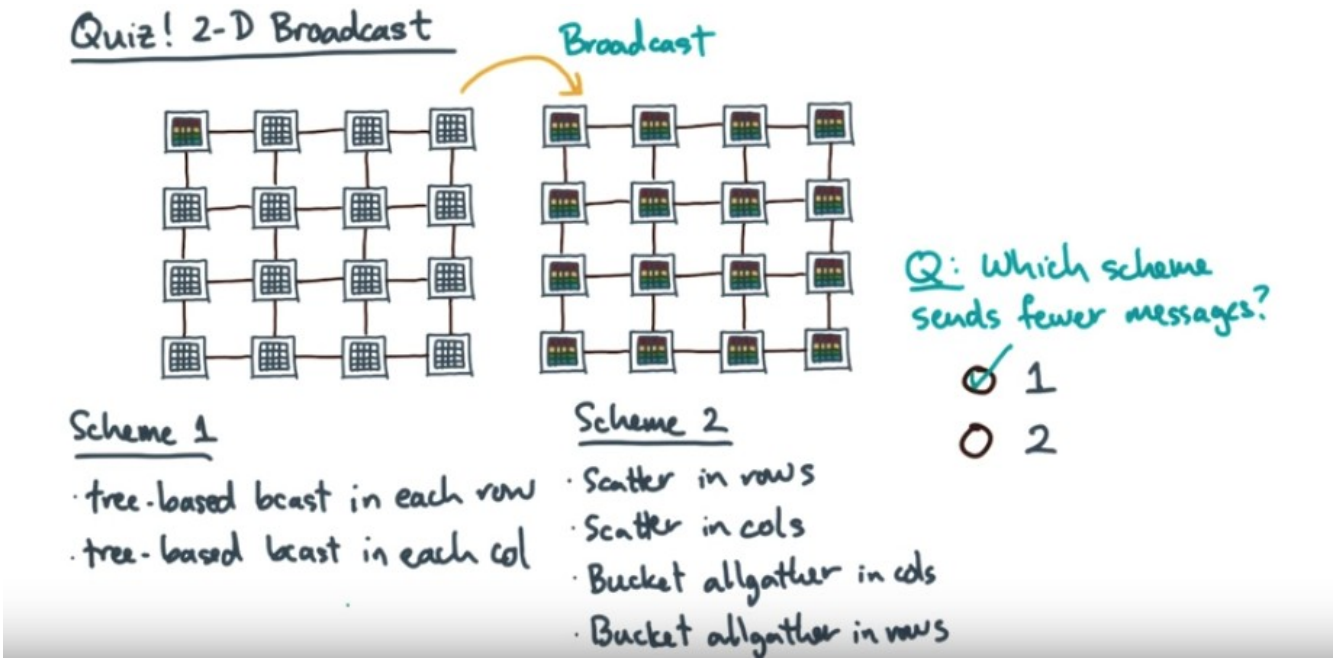
So how much time does this take? The row and the column gathers are one dimensional. Let's suppose we use the one dimensional bucketing algorithm. Remember the cost of that algorithm. Let's consider the row all-gathers first. Within each row, there are square root of P processes participating. The total output size is m times the square root of P. Therefore, this is the cost of the row part. Notice that relative to the one D algorithm alone, we've already improved the latency. It scales like square root of P, rather than P.



What about the column communication? The data volume is now n, but you still only have square root of P processes participating. Therefore, the execution time to run the column part of the algorithm is the 1D algorithm on an input of size, n, over square root of P processes.



Let's sum up the row and the column times. Asymptotically, the total time is now proportional to just square root of P in the alpha term. With respect to the beta term, it's basically optimal. In other words, taking advantage of a higher dimensional network gets us a little bit closer to the ultimate lower bound, at least on the number of messages.

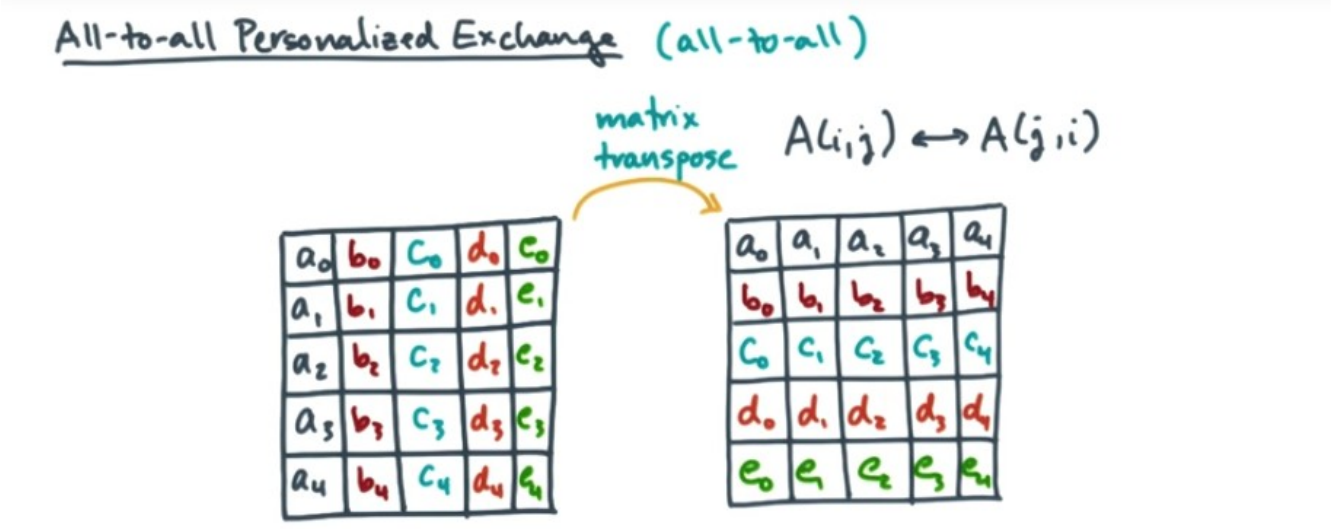


20. Suppose you want to do a broadcast on a 2-D network. Here are two schemes. In the first scheme, you run a tree-based algorithm broadcast in each row followed by a tree-based broadcast algorithm in each column. The second scheme has four steps. Scatter in the rows, followed by a scatter in the columns, followed by a bucket all gather in the columns, followed by a bucket all gather in the rows. Stop for a minute and convince yourself that these two schemes are plausible. Then once you've done that, I want you to answer this question. Which scheme has the lowest alpha term if you estimate the

cost? Is it Scheme 1 or Scheme 2? Choose your answer here.

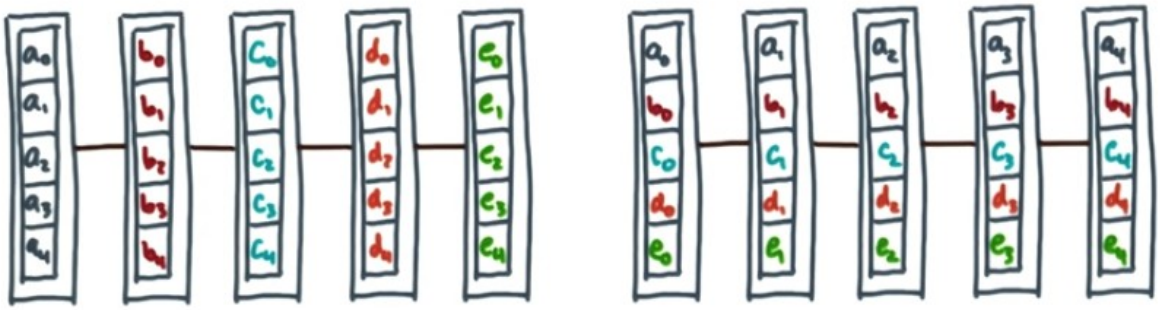
21. The answer is scheme 1. Remember that the alpha term cost of a tree-based algorithm is proportional to log of the number of nodes. By contrast, any sort of bucketing will be linear. Now in this case, since we're operating on a 2-D mesh, it'll be linear and square root of P rather than linear and P. But that's still worse than log. So, at least with respect to the alpha term, scheme 1 is better. Now, had I asked you which scheme has the largest beta term, what would you have said?

22. Let's discuss another collective. It's called the All-to-all Personalized Exchange. In the business, you can just refer to this as all-to-all, for short. In an all-to-all, every node wants to send a unique message to every other node. Now, this picture is not quite accurate, because each node is just sort of shouting a message to all other nodes. But in fact, in an all-to-all personalized exchange, every node has a personal message to send to every other node.



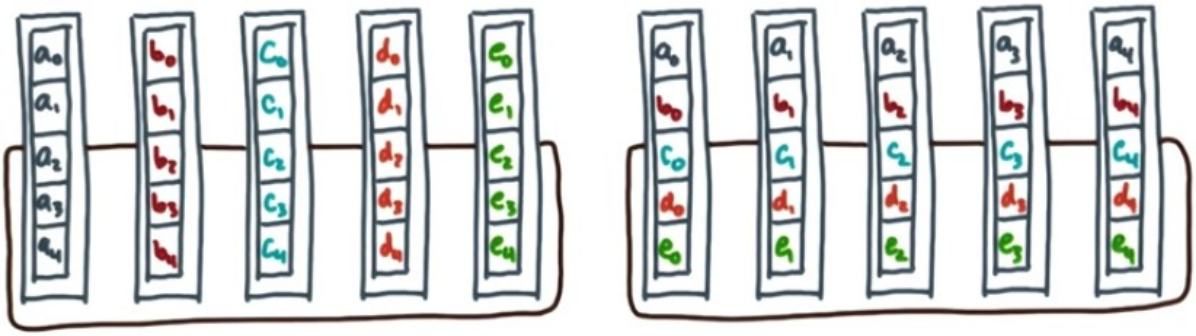
As an example, consider a matrix transpose. Start with a matrix, initialize to some set of values. A transpose rotates the elements. More formally, every element A(i,j) swaps positions with A(j,i).

All-to-all Personalized Exchange (all-to-all)



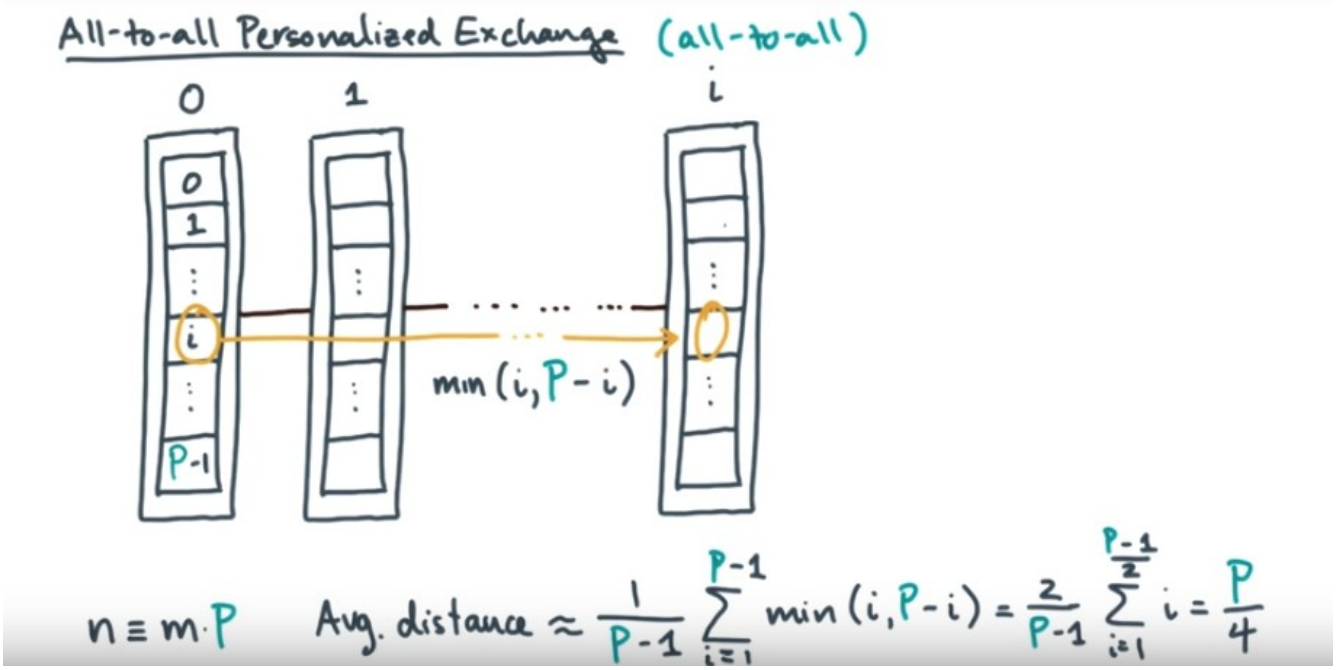
Now suppose our initial matrix A is distributed column wise among several nodes in a linear network. So that might look like this. After the transpose we want the system to look like this. So how do you implement it?

All-to-all Personalized Exchange (all-to-all)

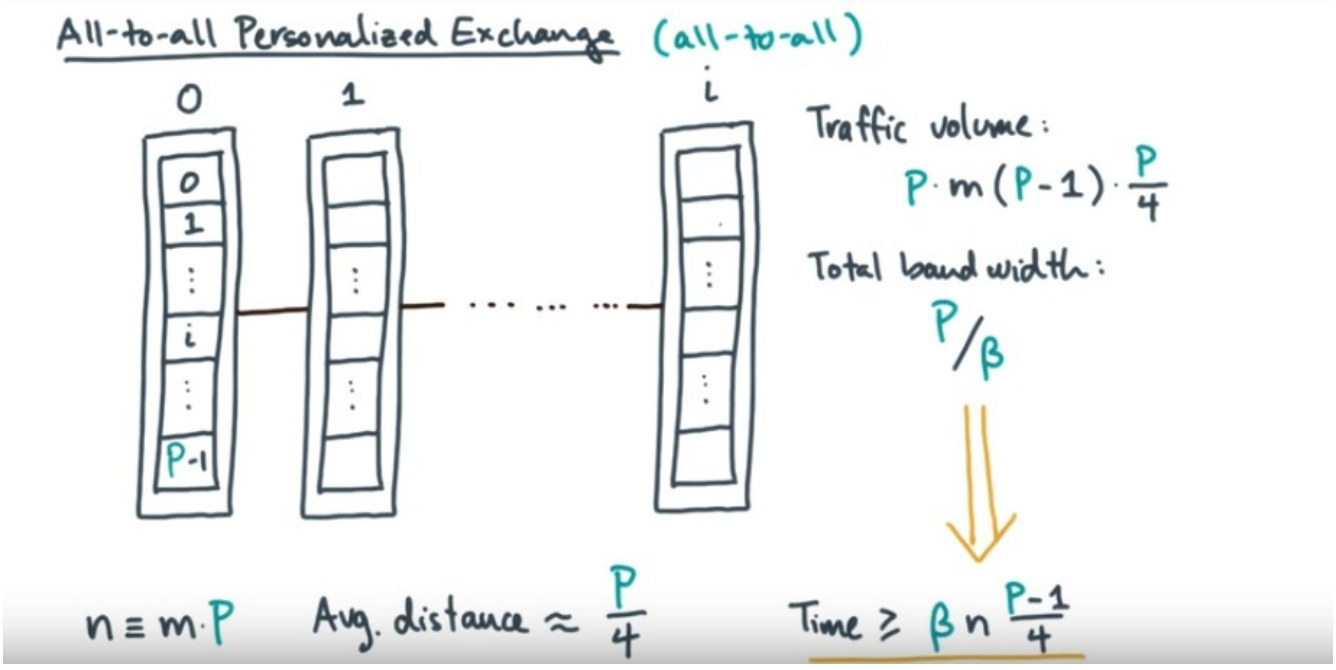


$n \equiv m \cdot P$

To make this a little easier to see let's switch from a linear network to a ring network. And let's suppose that the size of the data that every node wants to send to every other node is m. So each of these little boxes is of size m and the total length of a vector on any node is n which is m times P. Before I give you an algorithm I want you to come up with a performance target. Start by considering one of these nodes, and the P - 1 messages that it wants to send.

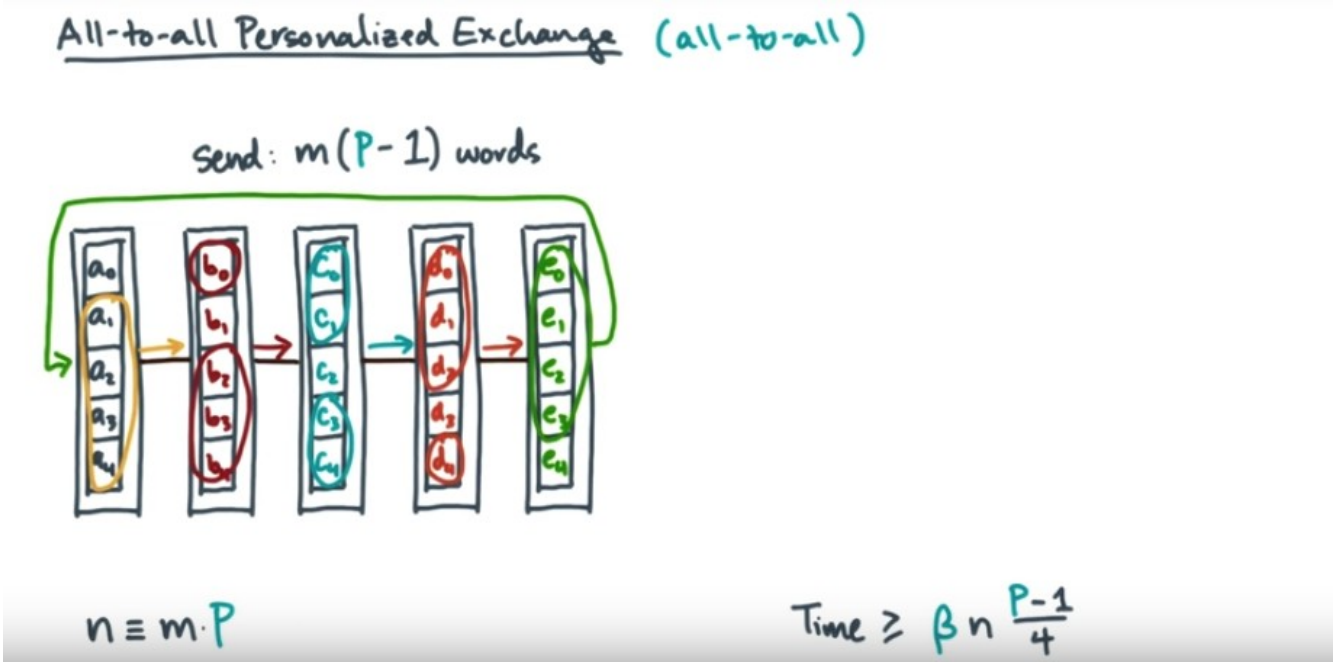


So, let's start with node 0 and let's look at the ith message that it needs to send. The ith message needs to go to node i. So, how far away is that node? I claim that that distance is min(i, P- i). Why? On a ring, the message could go either this way or this way, depending on which one is shorter. Now, from this fact, you can estimate the average distance that a message must travel. So, each of the messages from 1 to P-1 need to travel a distance of this. We take the average over all messages. For simplicity, this calculation assumes that P-1 is even. If you work it all out, the average turns out to be P over 4. So let's remember this handy fact.

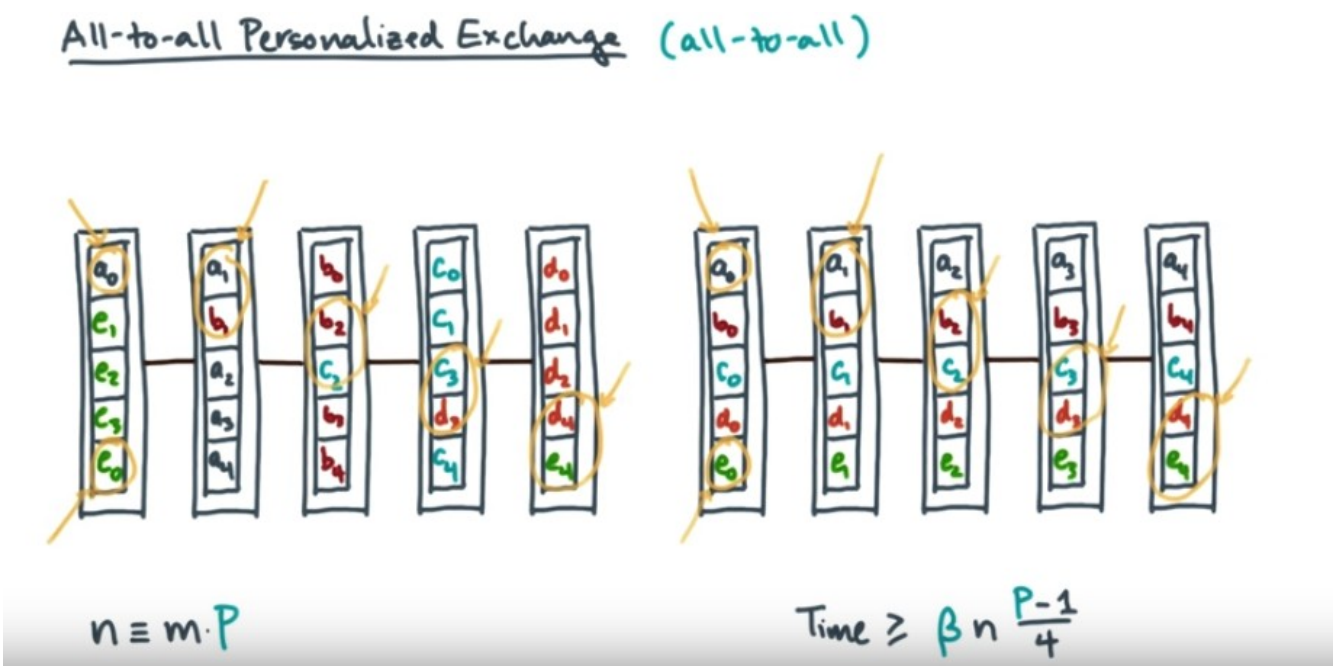


Now all P nodes in the network need to do these sends. What's the total volume of traffic that the network has to carry? Well, it's the number of nodes (P) times the total volume per node for all

messages (m (P - 1)) times the average distance per message (P/4). Here's another question, how much bandwidth is available to carry all of this traffic? Well the total bandwidth is the number of links in the network times the speed per link, which is just one over the inverse bandwidth. Again we're setting a performance target, so we're making optimistic assumptions. Now from these two facts, we can estimate a lower bound on the communication time. It's just the total volume divided by the total speed. So now what we want to do is write down an algorithm and then compare that algorithm to this lower bound.



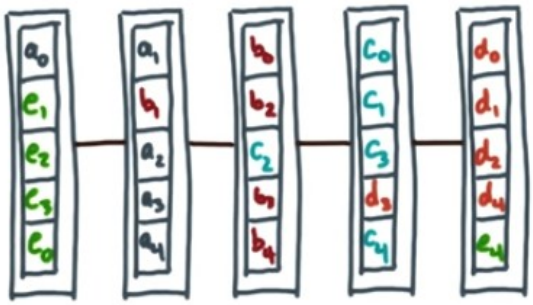
Okay, let's talk about a specific algorithm. Here's an idea, which is to perform a sequence of circular shifts. Here's the initial state. In the first step, each node will send m times P-1 of it's data to its neighbor. For example, node 0 will send these words to the right. What it is sending is all of the data that needs to go somewhere. So while node 0 is doing that, node 1 is doing basically the same thing. It's packing up the data that needs to go to other nodes and sending it. Node 2 does the same thing. It keeps the one piece of data it's supposed to hang onto and ships off everything else. And so on.



Here's what the system will look like at the end of this first step. Compare this to our goal output. So after one step, you can see that every node has two of the elements that it needs to retain. So node 0 has a0 and e0, node 1 has a1 and b1, node 2 has b2 and c2. This is exactly what they need in the final output. So that means in the next step, we could repeat the same process, again retaining the data we're supposed to keep and shipping off everything else.

So in step two, every node would send m(P-2) of its data. And this process will repeat until we're done.

All-to-all Personalized Exchange (all-to-all)



Round i: send $m(P-i)$ words

$$T_{1D}(n \equiv mP; P) = \sum_{i=1}^{P-1} (\alpha + \beta m(P-i))$$

$$= (\alpha + \beta m \underbrace{\frac{P}{2}}_{= \frac{n}{2}})(P-1)$$

$n \equiv m \cdot P$

Time $\geq \beta n \frac{P-1}{4}$

To analyze the cost of this algorithm, let's consider some round. Call it i. In round i, a node sends m times P-i of it's data to a neighbor. So how much time does this whole thing take. First, the overall scheme needs P-1 steps. In each step i, nodes send m times P-i words, so plug in our usual per message cost model, then simplify and finally enjoy your handiwork. Now remember, our usual convention is to let total size n be m times P. So then this expression becomes n over 2. So notice that the bandwidth term is now linear mP. Compare that to this lower bound. Hey, your algorithm seems pretty good. It's within a factor of 2 of our lower bound estimate. Tofu the parting hypercube says oh yeah. You know I 'm SCS no one knows you're a tofu.

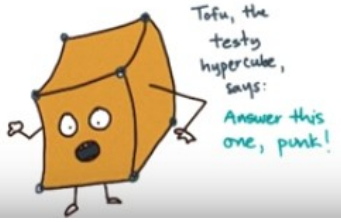
Quiz! All-to-all in Higher Dimensions

Q: Which network has the best chance to reduce the asymptotic running time to

$O(\alpha \log P + \beta n \log P)$?

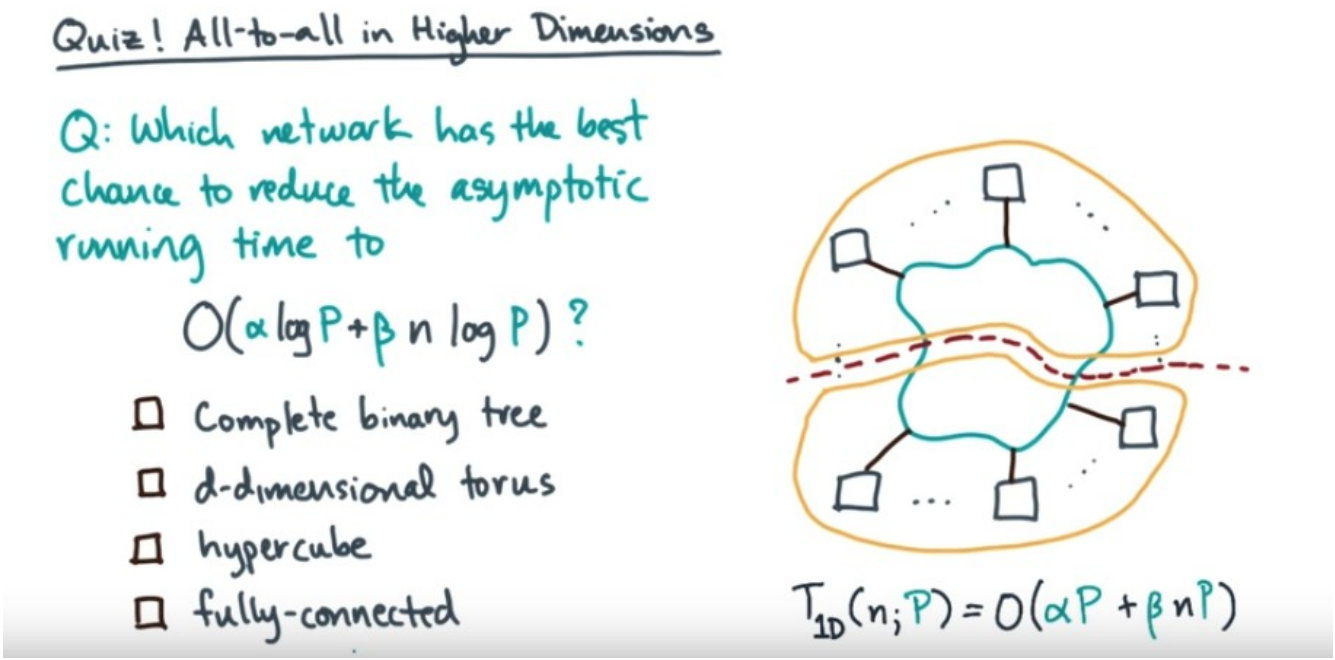
- ☐ Complete binary tree
- ☐ d-dimensional torus
- ☑ hypercube
- ☑ fully-connected

$T_{1D}(n; P) = O(\alpha P + \beta n P)$



23. Recall the running time for an all to all personalized exchange on a ring network. Here is Mr. Tofu's question. What network has the best chance of reducing the asymptotic running time from being linear nP to being, say logarithmic in P? I'll give you four choices. Complete binary tree, d

dimensional torus, a hypercube, and a fully connected network. And I want you to check all that you think apply.



24. This is actually a really hard question, and probably way too unfair for a quiz. Yeah, I'm a jerk like that. So I would say that your best bets are the hypercube and fully connected networks. Here's my thought process. An all to all is intrinsically bi-section, limited. So, of these choices, only hypercubes and fully connected networks have linear bi-section widths or better. That's what you would need to knock down this bandwidth term from being linear in P, to something that is logarithmic in P, or maybe even constant. Okay, so that's not a rigorous proof that these are the right answers, it's just an intuitive idea. So, what I'll do is I'll put some pointers in the instructor's notes where you can do a little bit more reading if you're interested. I know you'll be interested, because if you do the reading, Mr Tofu will give you a cupcake. This cupcake is for you, Mr Tofu wants to give you a cupcake.

25. [SOUND] Oh, good, you made it to the end. Or did you take some longest, shortest path through the lesson to get here? Well, here's what I think are the two big ideas of this lesson. The first one is the [COUGH] excuse me, [COUGH] the first one is the idea of congestion. [SOUND] >> Boo. >> This concept lets you design for one topology and then estimate whether it will map well or poorly to another. The second big idea is to exploit higher dimensional networks, both for fun and for profit. For instance, you should have noticed that there are big algorithmic scalability wins from taking advantage of a 3D mesh network instead of a 2D one. Now, in closing, let me pose one last question. As systems get bigger, will topology matter again? If it does, you will be ready.