MITOCW | ocw-6.033-lec15

0:00:03 -- is the next group of topics in 6.033 called fault tolerance. 0:00:07.582 And the goal here is to learn how to build reliable systems. 0:00:12.088 An extreme case, or at least our ideal goal, 0:00:15.372 is to try to build systems that will never fail. 0:00:18.961 And what will find is that we really can't do that, 0:00:22.78 but what we'll try to do is to build systems which maybe fail 0:00:27.362 less often than if you built them without the principles that 0:00:31.945 we're going to talk about. So the idea is how to build 0:00:37.064 reliable systems. So in order to understand how 0:00:40.581 to build reliable systems, we need to understand what 0:00:44.556 makes systems unreliable. And that has to do with 0:00:48.226 understanding what faults are. What problems occur in systems 0:00:52.813 that cause systems to fail? And you've actually seen many 0:00:57.094 examples of faults already. Informally, a fault is just 0:01:01.683 some kind of a flaw or a mistake that causes a component or a 0:01:05.426 module not to perform the way it's supposed to perform. 0:01:08.794 And we'll formalize this notion a little bit today as we go 0:01:12.411 along. So there are many examples of 0:01:14.594 faults, several of which you've already seen. 0:01:17.338 A system could fail because it has a software fault, 0:01:20.519 a bug in a piece of software, so when you run it, 0:01:23.513 it doesn't work according to the way you expect. 0:01:26.444 And that causes something bad to happen. 0:01:30 You might have hardware faults. You store some data on a disk. 0:01:34.377 You go back and read it. And it isn't there. 0:01:37.462 But it's been corrupted. And that's an example of a 0:01:41.05 fault that might cause bad things to happen if you build a 0:01:45.141 system that relies on a disk storing data persistently. 0:01:49.016 You might have design faults where a design fault might be 0:01:53.106 something where you try to, let's say, figure out how much 0:01:57.196 buffering to put into a network switch. 0:02:01 And you put into little buffering. 0:02:02.965 So what ends up happening is too many packets get dropped. 0:02:06.36 So you might actually just have some bad logic in there, 0:02:09.636 and that causes you to design something that isn't quite going 0:02:13.269 to work out. And you might, 0:02:14.818 of course, have implementation faults where you have a design, 0:02:18.451 and then you implement it, and you made a mistake in how 0:02:21.727 it's implemented. And that could cause faults as 0:02:24.526 well. And another example of the kind 0:02:26.67 of faults is an operational fault sometimes called a human 0:02:30.065 error where a user actually does something that you didn't 0:02:33.46 anticipate or was told not to do, and that caused bad things 0:02:36.974 to happen. For all of these faults, 0:02:40.993 there are really two categories of faults regardless of what 0:02:45.893 kind of fault it is. The first category of faults 0:02:49.88 are latent faults. So an example of a latent fault 0:02:53.95 is, let's say you have a bug in a program where instead of 0:02:58.684 testing if A less than B, you test if A greater than B. 0:03:04 So that's a bug in a program. But until it actually runs, 0:03:07.536 until that line of code runs, this fault in the program isn't 0:03:11.326 actually going to do anything bad. 0:03:13.41 It isn't going to have any adverse effect. 0:03:16 And therefore, this fault is an example of a 0:03:18.715 latent fault. And nothing is happening until 0:03:21.431 it gets triggered. And when it gets triggered, 0:03:24.273 that latent fault might become an active fault. 0:03:27.178 Now, the problem when a latent fault becomes an active fault is 0:03:31.094 that when you run that line of code, you might have a mistake 0:03:34.884 coming out at the output, which we're going to call an 0:03:38.231 error. So when an active fault is 0:03:41.483 exercised, it leads to an error. And the problem with errors is 0:03:45.02 that if you're not careful about how you deal with errors, 0:03:48.271 and most of what we're going to talk about is how to deal with 0:03:51.751 errors, if you're not careful about how you deal with errors 0:03:55.117 that leads to a failure. 0:03:57 0:04:02 So somewhat more formally, a fault is just any flaw in an 0:04:05.426 underlying component or underlying subsystem that your 0:04:08.669 system is using. Now, if the fault turns out not 0:04:11.545 to be exercised then. There's no error. 0:04:13.871 There's no error that results, and there's no failure that 0:04:17.359 results. It's only when you have an 0:04:19.439 active fault that you might have an error. 0:04:21.948 And when you have an error, you might have a failure. 0:04:25.13 And what we're going to try to do is to understand how to deal 0:04:28.863 with these errors so that when errors occur we're going to try 0:04:32.595 to hide them or mask them, or do something such that these 0:04:36.083 errors don't propagate and cause failures. 0:04:40 So the general goal, as I mentioned before, 0:04:42.248 is to build systems that don't fail. 0:04:44.122 So, in order to build systems that don't fail, 0:04:46.531 there are two approaches at a 50,000 foot level. 0:04:49.047 One approach to build a system that doesn't fail is to 0:04:52.367 build it out of, make sure, every system is going to be 0:04:54.776 built out of components or modules. 0:04:56.596 And those modules are going to be built out of modules 0:04:59.434 themselves. One approach might be to make 0:05:01.575 sure that no module ever fails, that no component that you use 0:05:04.841 to build your bigger system ever fails. 0:05:08 And it'll turn out that for reasons that will become clear 0:05:11.15 based on an understanding of the techniques we are going to 0:05:14.356 employ to build systems that don't fail, it'll turn out that 0:05:17.618 this is extremely expensive. It's just not going to work out 0:05:20.879 for us to make sure that our disks never fail, 0:05:23.366 and memory never fails, and our networks never fail, 0:05:26.185 and so on. It's just too expensive and 0:05:28.231 nearly impossible. So what we're going to do 0:05:31.351 actually is to start with unreliable components. 0:05:34 0:05:40 And we're going to build reliable systems out of 0:05:43.531 unreliable components or modules more generally. 0:05:47.062 And what this means is that the system that you build had better 0:05:51.795 be tolerant of faults that these underlying components have, 0:05:56.227 which is why the design of systems that don't fail or 0:06:00.134 rarely fail is essentially the same as designing systems that 0:06:04.642 are tolerant of faults, hence fault tolerance. 0:06:09 So that's the reason why we care about fault tolerance. 0:06:12.718 So let's take the example of the kinds of, 0:06:15.54 just to crystallize these notions of faults and failures a 0:06:19.465 little bit more. So let's say you have a big 0:06:22.426 system that has a module. Let's call it M1. 0:06:25.318 And this module uses a couple of other modules, 0:06:28.485 M2 and M3. And let's say M2 uses another 0:06:32.219 module, M4, where users might be an indication. 0:06:35.969 Or imagine this is an RPC call, for example. 0:06:39.26 And, let's say that M4 in here has some component inside M4 0:06:43.698 like a disk or something, some piece of software in M4. 0:06:47.831 And,

let's say that that fails. So, it has a fault. 0:06:51.658 It gets triggered. It becomes active, 0:06:54.413 leads to an error. It actually fails, 0:06:57.168 that little component. So when this fault becomes a 0:07:02.133 failure, a couple of things could happen. 0:07:05.18 M4, which is the module to which this little failure 0:07:09.066 belongs, can do one of two things. 0:07:11.58 One possibility is that this fault that caused the failure 0:07:15.923 gets exposed to the caller. So, M4 hasn't managed to figure 0:07:20.342 out a way to hide this failure from M2, which means that the 0:07:24.838 fault propagates up. The failure gets visible, 0:07:28.266 and the fault propagates up to M2. 0:07:32 And now, M2 actually sees the underlying component's failure. 0:07:36.516 So the point here is that this little component fault caused a 0:07:41.107 failure here which caused M4 itself to fail because M4 now 0:07:45.397 couldn't hide this underlying failure, and reported something 0:07:49.913 that was a failure, that was an output that didn't 0:07:53.602 conform to the specification of M4 out to M2. 0:07:56.913 Now, as far as M2 is concerned, all that has happened so far is 0:08:01.58 that the failure of this module, M4, has shown up as a fault to 0:08:06.247 M2, right, because an underlying module has failed. 0:08:10.01 It doesn't mean that M2 has failed. 0:08:14 It just means that M2 has now seen a fault. 0:08:16.786 And M2 now might manage to hide this fault, which would mean 0:08:20.701 that M1 doesn't actually see anything. 0:08:23.156 It doesn't see the underlying fault that caused the failure at 0:08:27.203 all. But of course, 0:08:28.933 if M2 now couldn't hide this or couldn't mask this failure, 0:08:32.799 then it would propagate an erroneous output out to M1, 0:08:36.333 an output that didn't conform to the specification of M2, 0:08:40.066 leading M1 to observe this as a fault, and so on. 0:08:43.266 So, the general idea is that failures of sub-modules tend to 0:08:47.2 show up as faults in the higher level module. 0:08:50.133 And our goal is to try to somehow design these systems 0:08:53.666 that use lots of modules and components where at some level 0:08:57.533 in the end we would like to avoid failing overall. 0:09:02 But inside here, we won't be able to go about 0:09:05.051 making everything failure-free. I mean, there might be failures 0:09:09.351 inside sub-modules. But the idea is to ensure, 0:09:12.472 or try to ensure, that M1 itself, 0:09:14.691 the highest level system, doesn't fail. 0:09:17.326 So let's start with a few examples. 0:09:19.684 In fact, these all examples of things that we've already seen. 0:09:23.914 And even though we haven't discussed it as such, 0:09:27.174 we've seen a lot of examples of fault tolerance in the class so 0:09:31.474 far. So, for example, 0:09:34.105 if you have bad synchronization code like you didn't use the 0:09:38.181 locking discipline properly or didn't use any of the other 0:09:42.119 synchronization primitives properly, you might have a 0:09:45.711 software fault that leads to the failure of a module. 0:09:49.303 Another example that we saw when we talked about networking is 0:09:53.379 when we talked about routing where the idea in here was that we 0:09:57.525 talked about routing protocols that could handle failures of 0:10:01.531 links. So, certain links could fail, 0:10:03.949 leading to certain paths to not be usable. 0:10:06.782 But, the routing system managed to find other paths around the 0:10:10.996 network. And that was because there were 0:10:15.04 other parts available because the network itself was built 0:10:18.792 with some degree of redundancy underneath. 0:10:21.491 And the routing protocol was able to exploit that. 0:10:24.716 Another example that we saw again from networks is packet 0:10:28.402 loss. We had best effort networks 0:10:30.508 that would lose packets. And it didn't mean that your 0:10:33.931 actual transfer of a file at the ends and where would miss data. 0:10:39 We came up with retransmissions as a mechanism to use, 0:10:42.63 again, another form of redundancy where you try the 0:10:46.054 same thing again to get your data through. 0:10:48.863 Another example of the failure that we saw was congestion 0:10:52.698 collapse -- 0:10:54 0:11:00 -- where there was too much data being sent out into the 0:11:02.867 network too fast, and the network would collapse. 0:11:05.37 And our solution to this problem was really to shed load 0:11:08.238 was to run the system slower than it otherwise would, 0:11:10.949 by having the people sending data send data slower in order 0:11:13.973 to alleviate this problem. Another example which we saw 0:11:16.789 last time was, or briefly saw last time was 0:11:18.979 the domain name system where the domain name servers are 0:11:22.003 replicated. So, if you couldn't reach one 0:11:24.089 to resolve your domain name, you could go to another one. 0:11:28 And all of these, or most of these actually use 0:11:30.622 the same techniques that we're going to talk about. 0:11:33.472 And all of these techniques are built around some form of 0:11:36.665 redundancy on another except probably the locking thing. 0:11:39.8 But all of the others are built around some form of redundancy. 0:11:43.334 And we'll understand this more systematically today and in the 0:11:46.812 next couple of classes. So our goal here is to develop 0:11:49.833 a systematic approach -- 0:11:52 0:11:58 -- to building systems that are fault tolerant. 0:12:01.439 And the general approach for all fault tolerant systems is to 0:12:05.925 use three techniques. So the first one we've already 0:12:09.738 seen, which is don't build a monolithic system. 0:12:13.177 Always build it around modules. And the reason is that it will 0:12:17.738 that be easier for us to isolate these modules firstly one from 0:12:22.373 another. But then, when modules fail, 0:12:25.065 it will be easier for us to treat those failures as faults, 0:12:29.401 and then try to hide those faults, and apply the same 0:12:33.289 technique, which brings us to the second step, 0:12:36.654 which is when failures occur causing errors, 0:12:39.869 we need a plan for the higher level module to detect errors. 0:12:46 So failure results in an error. We have to know that it's 0:12:49 happened, which means we need techniques to detect it. 0:12:51.839 And, of course, once we detect an error we have 0:12:54.303 a bunch of things we could do with it. 0:12:56.285 But ideally, if you want to prevent the 0:12:58.321 failure of that system, of a system that's observed 0:13:01 errors, you need a way to hide these errors. 0:13:04 The jargon for this is mask errors. 0:13:06.824 And if we do this, if we build systems that do 0:13:10.563 this, then it's possible for us to build systems that can form 0:13:15.63 to spec. So the goal here is to try to 0:13:18.704 make sure that systems conform to some specification. 0:13:23.024 And if things don't conform to a specification, 0:13:26.846 then that's when we call it a failure. 0:13:31 And sometimes we play some tricks where in order to build 0:13:34.674 systems that "never fail", we'll scale back the 0:13:37.692 specification to actually allow for things that would in fact be 0:13:41.825 considered failures, but are things that still would 0:13:45.171 conform to the spec. So we relax the specification 0:13:48.386 to make sure that we could still meet the notion of a failure 0:13:52.323 free system or a fault tolerant system. 0:13:54.816 And we'll see some

examples of that actually in the next 0:13:58.425 lecture. And I've already mentioned, 0:14:01.983 the general trick for all of these systems that we're going 0:14:06.244 to study, examples that we're going to study, 0:14:09.476 is to use some form of redundancy. 0:14:11.9 And that's the way in which we're going to mask errors. 0:14:15.867 And almost all systems, or every system that I know of 0:14:19.76 that's fault tolerant uses redundancy in some form or 0:14:23.58 another. And often it's not obvious how 0:14:26.372 it uses it. But it does actually use 0:14:28.943 redundancy. So I'm going to now give an 0:14:32.724 example that will turn out to be the same example we'll use for 0:14:36.684 the next three or four lectures. And so, you may as well, 0:14:40.261 you should probably get familiar with this example 0:14:43.391 because we're going to see this over and over again. 0:14:46.648 It's a really simple example, but it's complicated enough 0:14:50.225 that everything we want to learn about fault tolerance will be 0:14:54.121 visible in this example. So it starts with the person 0:14:57.443 who wants to do a bank transaction at an ATM, 0:15:00.253 or a PC, or on a computer. You want to do a bank 0:15:04.783 transaction. And the way this works, 0:15:07.756 as you probably know, is it goes over some kind of a 0:15:12.087 network. And then, if you want to do 0:15:15.06 this bank transaction,, it goes to a server, 0:15:18.712 which is run by your bank. And the way this normally works 0:15:23.553 is that the server has a module that it uses a database system, 0:15:28.564 which deals with managing your account information. 0:15:34 And because you don't want to forget, and the bank shouldn't 0:15:39.488 forget how much money you have, there is data that's stored on 0:15:45.162 disk. And we're going to be doing 0:15:48.139 things that are actions of the following form. 0:15:52.325 We're going to be asking to transfer from some account to 0:15:57.627 another account some amount of money. 0:16:02 And now, of course, anything could fail in between. 0:16:05.089 So, for example, there could be a problem in the 0:16:07.993 network. And the network could fail. 0:16:10.155 Or the software running on the server could fail. 0:16:13.121 Or the software running this database system could crash or 0:16:16.704 report bad values or something. The disc could fail. 0:16:19.855 And do we want systematic techniques by which this 0:16:22.883 transfer here or all of these calls that look a lot like 0:16:26.281 transfer do the right thing? And so, this doing the right 0:16:30.512 thing is an informal way of saying meet a specification. 0:16:33.483 So, we first have to decide what we want for a 0:16:35.913 specification. That has to hold true no matter 0:16:38.344 what happens, no matter what failures occur. 0:16:40.666 So one example of a specification might be to say, 0:16:43.313 no matter what happens, if I invoke this and it 0:16:45.797 returns, then this amount of money has to be transferred from 0:16:49.038 here to here. So that could be a 0:16:50.712 specification that you might expect. 0:16:52.603 It also turns out this specification is extremely hard 0:16:55.466 to meet. And we're not even going to try 0:16:57.572 to do it. And this is the [weasel?] 0:16:59.408 wording I said before about, we'll modify the specification. 0:17:04 So, we'll change the specification going forward for 0:17:07.393 this example to mean, if this call returns, 0:17:10.187 then no matter what failures occur, either a transfer has 0:17:13.913 happened exactly once, or the state of the system is 0:17:17.307 as if the transfer didn't even get started, OK, 0:17:20.367 which is reasonable. I mean, and then if you really 0:17:23.694 care about moving the money, and you are determined that it 0:17:27.553 hasn't been moved, you or some program might try 0:17:30.681 it again, which actually is another form of using redundancy 0:17:34.606 where you just try it over again. 0:17:38 And we won't understand completely why a specification 0:17:41.11 that says you have to do this exactly once if it returns, 0:17:44.396 why that's hard to implement, why that's hard to achieve, 0:17:47.682 we'll see that in the next couple of classes. 0:17:50.264 So, for now, just realized that the 0:17:52.259 specification here is it should happen exactly once or it should 0:17:55.955 be as if no partial action corresponding to the 0:17:58.655 [UNINTELLIGIBLE] of this transfer happened. 0:18:02 So the state of the system must be as if the system never saw 0:18:06.148 this transfer request. So any module could failure. 0:18:09.604 So, let's take some examples of failures in order to get some 0:18:13.753 terminology that'll help us understand faults. 0:18:16.864 So one thing that could happen is that you could have a disk 0:18:20.943 failure. So the disc could just fail. 0:18:23.432 And one example of a disk failure is the disk fails and 0:18:27.165 then it just stops working. And it tells the database 0:18:31.687 system that's trying to read and write data from it that it isn't 0:18:36.008 working. So if that kind of failure 0:18:38.303 happens where this module here with this component just 0:18:41.949 completely stops and tells the higher-level module that it 0:18:45.797 stopped, that's an example of a failure. 0:18:48.43 That's called a fail stop failure. 0:18:50.658 And more generally, any module that tells the 0:18:53.628 higher-level module that it just stops working without reporting 0:18:57.881 anything else, no outputs, that's fail stop. 0:19:02 Of course, you could have disks. 0:19:03.926 And you might have failures that aren't fail stop. 0:19:06.97 You might have something where there is some kind of error 0:19:10.512 checking associated with every sector on your disk. 0:19:13.619 And, disk might start reporting errors that say that this is a 0:19:17.409 bad sector. So, it doesn't fail stop, 0:19:19.646 but it tells the higher level, the database system in this 0:19:23.188 case that some data that's read, or some data that's been 0:19:26.667 written, there's a bad sector, which means that the checksum 0:19:30.333 doesn't match the data that's being read. 0:19:34 When you have an error like that where it doesn't stop 0:19:36.969 working but it tells you that something bad is going on, 0:19:40.051 that's an example of a failure. That's called a fail fast 0:19:43.189 failure. I actually don't think these 0:19:45.206 terms, that most of these terms are particularly important. 0:19:48.456 Fail stop is usually important and worth knowing, 0:19:51.146 but the reason to go through these terms is more to 0:19:53.948 understand that there are various kinds of failures 0:19:56.75 possible. So in one case it stops 0:19:58.543 working. In another case, 0:20:01.221 it just tells you that it's not working but continues working. 0:20:05.879 It tells you that certain operations haven't been 0:20:09.544 correctly done. Now, another thing that could 0:20:12.903 happen when, for example, the disc has fail stop, 0:20:16.568 has fail fast is that the database system might decide 0:20:20.615 that right operations, you're not allowed to write 0:20:24.357 things to disk because the disk is either fail completely or is 0:20:29.091 fail fast. But it might allow actions or 0:20:33.03 requests that are read only. So, for example, 0:20:36.112 it might allow users to come up to an ATM machine, 0:20:39.543 and just read how much money they have from their account 0:20:43.465 because it might be that there is a cache of the data that's in 0:20:47.807 memory in the database. So it might allow read-only

0:20:51.309 actions, in which case the system's perform is functioning 0:20:55.301 with only a subset of the actions that it's supposed to be 0:20:59.293 taking. And if that happens, 0:21:01.184 that kind of failure is called a fail soft failure, 0:21:04.685 where not all of the interfaces are available, 0:21:07.837 but a subset of the interfaces are available and correctly 0:21:11.828 working. And the last kind of failure 0:21:16.051 that could happen is that in this example, 0:21:19.056 let's say that failures are occurring when there's a large 0:21:23.232 number of people trying to make these requests at ATMs. 0:21:27.189 And, there is some problems that have arisen. 0:21:31 And somebody determines that the problem arises when there is 0:21:34.326 too many people gaining access to the system at the same time. 0:21:37.709 And the system might now move to a mold where it allows only a 0:21:41.091 small number of actions at a time, a small number of 0:21:43.919 concurrent actions at a time, or maybe one action at a time. 0:21:47.191 So, one user can come at a time to the system, 0:21:49.686 which means the systems, there has been a failure, 0:21:52.403 but the way the system's dealing with it is that it 0:21:55.175 determines that the failure doesn't get triggered when the 0:21:58.336 load is low. So it might function at low 0:22:01.928 performance. It still provides all of the 0:22:04.785 interfaces, but just at very low performance or at lower 0:22:08.714 performance. And that kind of behavior is 0:22:11.571 called failsafe. So it's moved to a mode where 0:22:14.785 it's just scaled back how much work it's willing to do, 0:22:18.642 and does it at degraded performance. 0:22:22 0:22:42 OK, so the plan now is for the rest of today, 0:22:45.179 so tomorrow from the next lecture on, what we're going to 0:22:49.225 do is understand algorithms for how we go about and how you 0:22:53.416 build systems that actually do one or all of these in order to 0:22:57.823 meet the specification that we want. 0:23:00.352 But before we do that you have to understand a little bit about 0:23:04.832 models for faults. In order to build fault 0:23:08.423 tolerant systems, it's usually a good idea to 0:23:11.147 understand a little bit more quantitatively models or faults 0:23:14.799 that occur in systems. And primarily, 0:23:17.028 this discussion is going to be focused on hardware faults 0:23:20.495 because most people don't understand how software faults 0:23:23.9 are to be modeled. But since all our systems are 0:23:26.809 going to be built on hardware, for example discs are going to 0:23:30.523 be really, really common. Our network links are going to 0:23:34.771 be common. And all of those conform nicely 0:23:37.192 to models. It's worth understanding how 0:23:39.436 that works. So, for example, 0:23:41.03 a disk manufacturer might report that the error rate of 0:23:44.219 undetected errors, so disks usually have a fair 0:23:46.935 amount of error detection in them. 0:23:48.883 But, they might report that the error rate of undetected errors 0:23:52.544 is, say, ten to the minus 12 or ten to the minus 13. 0:23:55.556 And that number looks really small. 0:23:57.563 That says that out of that many bits, maybe one bit is 0:24:00.693 corrupted, and you can't detect it. 0:24:04 But, you have to realize that given modern workloads, 0:24:07.077 for example take Google as an example that you saw from last 0:24:10.569 recitation, the amount of data that's being stored in the 0:24:13.883 system like that or in the world in general is so huge that a ten 0:24:17.671 to the minus 13th error rate means that you're probably 0:24:20.867 seeing some bad data and file that you can never fix or never 0:24:24.418 detect every couple of days. Network people would tell you 0:24:27.791 that fiber optic links have an error rate of one error in ten 0:24:31.342 to the 12th. But you have to realize that 0:24:34.519 these links are sending so many gigabits per second that one 0:24:37.72 error in ten to the 12th means something like there's an error 0:24:41.031 that you can't detect maybe every couple of hours. 0:24:43.689 What that really means is that at the higher layers, 0:24:46.457 you need to do more work in order to make sure that your 0:24:49.441 data is protected because you can't actually rely on the fact 0:24:52.697 that your underlying components have these amazingly low numbers 0:24:56.116 because there's so much data going on, being sent or being 0:24:59.209 stored on these systems that you need to have other techniques at 0:25:02.682 a higher layer to protect, if you really care about the 0:25:05.612 integrity of your data. In addition to these raw 0:25:09.877 numbers, there's two or three other metrics that people use to 0:25:14.648 understand faults and failures. The first one is the number of 0:25:19.418 tolerated failures. So, for example, 0:25:22.156 if you build a system to store data and you're worried about 0:25:26.77 discs failing or discs reporting at earnest values, 0:25:30.916 you might replicate that data across many, many discs. 0:25:36 And then when you design your system, one of the things you 0:25:39.441 would want to analyze and report is the number of tolerated 0:25:42.883 failures of discs. So, for example, 0:25:44.901 if you build a system out of seven discs, you might say that 0:25:48.402 you can handle up to two failed disks, or simply like that, 0:25:51.843 depending on how you've designed your system. 0:25:54.454 And that's usually a good thing to report because then people 0:25:58.015 who use your system can know how to provision or engineer your 0:26:01.635 system. The second metric which we're 0:26:04.812 going to spend a few more minutes on is something called 0:26:08.25 the mean time to failure. 0:26:10 0:26:21 And what this says is it takes a model where you have a system 0:26:25.041 that starts at time zero, and it's running fine. 0:26:28.154 And then at some point in time, it fails. 0:26:30.804 And then, when it fails, that error is made known to an 0:26:34.381 operator. Or it's made known to some 0:26:36.7 higher level that has a plan to work around it to repair this 0:26:40.675 failure. And then, once the failure gets 0:26:43.769 repaired, it takes some time for the failure to get repaired. 0:26:47.192 And once it's repaired it starts running again. 0:26:49.817 And then it fails at some other point in the future. 0:26:52.728 And when it goes through the cycle of failures and repairs, 0:26:56.038 you end up with a timeline that looks like this. 0:26:58.72 So you might start at time zero. 0:27:00.489 And the system is working fine. And then there is a failure 0:27:05.374 that happens here. And then the system is down for 0:27:09.683 a certain period of time. And then somebody repairs the 0:27:14.432 system, and then it continues to work. 0:27:17.685 And then it fails again, and so on. 0:27:20.675 And so, for each of the durations of time that the 0:27:24.984 system is working, let's assume it's starting at 0:27:29.118 zero, each of these defines a period of time that I'm going to 0:27:34.482 call TTF or time to fail. OK, so this is the first time 0:27:40.121 to fail interval. This is the second time to fail 0:27:43.634 -- 0:27:44 0:27:52 This is the second time to fail interval, time to repair, 0:27:55.964 and this is the third time to fail interval, 0:27:59.008 and so on. And analogously in between 0:28:01.668 here, I could define these time to repair intervals, 0:28:04.941 TTR1, TTR2, and so on. So, the mean time to failure is 0:28:08.342 just the mean of these values. There's some duration of

time. 0:28:12.192 You're like, three hours the system worked, 0:28:14.887 and then it crashed. That's TTF1. 0:28:16.941 And then, somebody to repair it worked now for six hours. 0:28:20.534 That's TTF2, and so on. 0:28:21.946 If you run your system for a long enough period of time like 0:28:25.732 a disk or anything else, and then you observe all these 0:28:29.197 time to fail samples, and take the mean of that, 0:28:32.213 that tells you a mean time to failure. 0:28:36 The reason this is interesting is that you could run your 0:28:40.011 system for a really long period of time, and build up a metric 0:28:44.382 called availability. So, for example, 0:28:46.961 if you're running a website, and the way this website works 0:28:51.116 is it runs for a while and then every once in awhile it crashes. 0:28:55.629 So its network crashes and people can't get to you. 0:29:00 So you could run this for months or years on end, 0:29:02.952 and then observe these values. You could run this every month. 0:29:06.704 You could decide what availability is, 0:29:08.98 and decide if it's good enough or if you want to make it higher 0:29:12.794 or lower. So you could now define your 0:29:15.07 availability to be the fraction of time that your system is up 0:29:18.822 and running. And the fraction of time that 0:29:21.344 the system is up and running is the fraction of time on this 0:29:24.973 timeline that you have this kind of shaded thing. 0:29:27.926 OK, so that's just equal to the sum of all the time to failure 0:29:31.678 numbers divided by the total time. 0:29:35 And the total time is just the sum of all the TTF's and the 0:29:39.393 TTR's. 0:29:40 0:29:45 OK, and that's what availability means is the 0:29:49.221 fraction of time that your system is available is up. 0:29:54.21 Now, if you divide both the top and the bottom by N, 0:29:59.103 this number works out to be the mean time to failure divided by 0:30:05.051 the mean time to failure plus the mean time to repair. 0:30:11 0:30:18 So this is a useful notion because now it tells you that 0:30:21.372 you can watch your system for a very long period of time, 0:30:24.99 and build up a mean estimate, mean values of the time to 0:30:28.363 failure and the time to repair, and just come up with the 0:30:31.797 notion of what the availability of the system is. 0:30:34.74 And then, decide based on whether it's high enough or not 0:30:38.174 whether you want to improve some aspect of the system and whether 0:30:42.099 it's worth doing. So it turns out this mean time 0:30:46.694 to failure, and therefore availability is related for 0:30:51.365 components to a notion called the failure rate. 0:30:55.497 So let me define the failure rate. 0:30:59 0:31:09 So the failure rate is defined, it's also called a hazard 0:31:14.142 function. That's what people use the term 0:31:17.816 H of T, the hazard rate. That's defined as the 0:31:21.948 probability that you have a failure of a system or a 0:31:26.632 component in time, T to T plus delta T, 0:31:30.122 given that it's working, we'll say, OK, 0:31:33.612 at time T, OK? So it's a conditional 0:31:37.212 probability. It's the probability that 0:31:39.348 you'll fail in the next time instant given that it's 0:31:42.293 correctly working and has been correctly working. 0:31:45.064 It's correctly working at time T. 0:31:46.912 So, if you look at this for a disk, most discs look like the 0:31:50.318 picture shown up here. This is also called the bathtub 0:31:53.378 curve because it looks like a bathtub. 0:31:55.515 What you see at the left end here are new discs. 0:31:58.228 So, the X axis here shows time. I guess it's a little shifted 0:32:02.465 below. You can't read some stuff 0:32:04.031 that's written at the bottom. But the X axis shows time, 0:32:06.81 and the Y axis shows the failure rate. 0:32:08.679 So, when you take a new component like a new light bulb 0:32:11.407 or a new disc or anything new, there is a pretty high chance 0:32:14.388 that it'll actually fail because manufacturers, 0:32:16.712 when they sell you stuff, don't actually sell you things 0:32:19.491 without actually burning them in first. 0:32:21.411 So for semiconductors, that's also called yield. 0:32:23.785 They make a whole number of chips, and then they're burning 0:32:26.716 a few, and then they only give you the rest. 0:32:30 And the fraction that survives the burning is also called the 0:32:34.137 yield. So what you see on our left, 0:32:36.482 the colorful term for that is infant mortality. 0:32:39.655 So it's things that die when they are really, 0:32:42.689 really young. And then, once you get past the 0:32:45.724 early mortality, you end up with a flat failure, 0:32:48.965 a conditional probability for failure. 0:32:51.517 And what this says is that, and I'll get to this and a 0:32:55.172 little bit. But what this says is that once 0:32:58.068 you are in the flat region, it says that the probability of 0:33:02.068 failure is essentially independent of what's happened 0:33:05.655 in the past. And then you stay here for a 0:33:09.872 while. And then if the system has been 0:33:12.346 operating like a disk has been operating for awhile, 0:33:15.756 let's say three years or five years typically for discs, 0:33:19.434 then the probability of failure starts going up again because 0:33:23.446 that's usually due to wear and tear, which for hardware 0:33:27.057 components is certainly the case. 0:33:30 There are a couple of interesting things about this 0:33:32.83 curve that you should realize, particularly when you read 0:33:36 specifications for things like discs. 0:33:38.037 Disc manufacturers will report a number, like the mean time to 0:33:41.49 failure number. And the mean time to failure 0:33:43.924 number that the report might usually, I mean for discs might 0:33:47.264 be 200,000 hours or 300,000 hours. 0:33:49.132 I mean, that's a really long period of time. 0:33:51.566 That's 30 years. So when you look at a number 0:33:54.056 like that, you have to ask whether what it means is that 0:33:57.169 discs really survive 30 years. And anybody who is on the 0:34:01.605 computer knows, you know, most discs don't 0:34:04.237 survive 30 years. So they are actually reporting 0:34:07.254 one over the reciprocal of this thing at the flat region of the 0:34:11.235 curve because this conditional failure probability rate, 0:34:14.766 at this operation time when the only reason things fail is 0:34:18.426 completely random failures not related to wear and tear. 0:34:21.957 So when disc manufacturers report a mean time to failure 0:34:25.488 number, they are actually reporting something that that's 0:34:29.083 the time that you're disc is likely to work. 0:34:33 What that number really says is that during the period of time 0:34:36.753 that the disc is normally working, the probability of a 0:34:40.076 random failure is one over the mean time to failure. 0:34:43.215 That's what it really says. So the other number that they 0:34:46.661 also report, often in smaller print, is it the expected 0:34:49.984 operational lifetime. And that's usually something 0:34:53 like three years or four years or five years, 0:34:55.707 whatever it is they report. And that's where this thing 0:34:59.03 starts going up, and beyond a point where the 0:35:01.738 probability of failures above some threshold, 0:35:04.446 they report that as the expected operational lifetime. 0:35:09 Now, for software, this curve doesn't actually 0:35:11.206 apply, or at least nobody really knows what the curve is for 0:35:14.1 software. What is true for software, 0:35:15.816 though, is infant mortality, things were the conditional 0:35:18.513 probability of failure is

high for new software, 0:35:20.818 which is why you are sort of well advised, 0:35:22.829 the moment the new upgrade of something comes around, 0:35:25.379 most people who are prudent wait a little bit to just make 0:35:28.174 sure all the bugs are out, and things get a little bit 0:35:30.773 stable. So they wait a few months. 0:35:33.674 You are always a couple of revisions behind. 0:35:36.442 So I do believe that for software, the left side of the 0:35:39.919 curve holds. It's totally unclear that there 0:35:42.688 is a flat region, and it's totally unclear that 0:35:45.649 things start rising again with age. 0:35:47.839 So the reason for this curve, being the way it is, 0:35:50.993 is a lot of this is based on the fact that things are 0:35:54.342 mechanical and have wear and tear. 0:35:56.466 But the motivation for this kind of curve actually comes 0:36:00.008 from demographics and from human lifespans. 0:36:04 So this is a picture that I got from, it's a website called 0:36:07.99 mortality.org, which is a research project run 0:36:11.085 by demographers. And they have amazing data. 0:36:14.044 There's way more data available but human life expectancy and 0:36:18.171 demographics than anything about software. 0:36:20.992 What this shows here is actually the same bathtub curve 0:36:24.707 as in the previous chart. It just doesn't look like that 0:36:28.491 because the Y axis is on a log scale. 0:36:32 So given that it's rising linearly between 0.001 and 0.01, 0:36:35.487 on a linear scale that looks essentially flat. 0:36:38.24 So human beings for the probability of death, 0:36:40.932 at a certain time, given that you are alive at a 0:36:43.808 certain time, that follows this curve here, 0:36:46.377 essentially a bathtub curve. At the left hand, 0:36:49.13 of course, there is infant mortality. 0:36:51.333 I think I pulled the data down. I think I pulled the data down. 0:36:55.126 This is from an article that appeared where the data for the 0:36:58.736 US population 1999. It starts off again with infant 0:37:02.903 mortality. And then it's flat for a while. 0:37:05.421 Then it rises up. Now, there's a lot of 0:37:07.754 controversy, it turns out, for whether the bathtub curve 0:37:11.131 at the right end holds for human beings or not. 0:37:13.956 And, some people believe it does, and some people believe it 0:37:17.578 doesn't. But the point here is that for 0:37:19.912 human beings anyway, the rule of thumb that 0:37:22.491 insurance companies use for determining insurance premiums 0:37:25.991 is that the log of the death rate, the log of the probability 0:37:29.675 of dying in a certain age grows linearly with the time that 0:37:33.236 somebody has been alive. And that's what this graph 0:37:37.76 shows, that on large scale on the Y axis, you have a line. 0:37:41.822 And that's what they use for determining insurance premiums. 0:37:45.817 OK, so the reason this bathtub curve is actually useful is, 0:37:49.744 so if you go back, let's go back here. 0:37:52.25 The reason both these numbers are useful, the flat portion of 0:37:56.312 the bathtub curve and the expected operational lifetime is 0:38:00.171 the following. It's not like this flat portion 0:38:03.97 of the curve where the disc manufacturer reports the mean 0:38:07.528 time to failure. That's 30 years. 0:38:09.562 It's not like that's useless even though you're disc only 0:38:13.121 might run for three to four years. 0:38:15.218 The reason is that if you have a project, if you have a system 0:38:19.095 where you are willing to upgrade your disk every three years 0:38:22.844 where you've budgeted for upgrading your discs every said 0:38:26.403 three years, then you might be better off buying a disk whose 0:38:30.216 expected lifetime is only five years but whose flat portion is 0:38:34.093 really low. So in particular, 0:38:37.28 if you're given to discs, one of which has a curve that 0:38:41.345 looks like that, and another that has a curve 0:38:44.658 that looks like that, and let's say this is five 0:38:48.197 years, and this is three years. If you're building a system and 0:38:52.865 you've budgeted for upgrading your discs every four years, 0:38:57.157 then you're probably better off using the thing with the lower 0:39:01.75 value of mean time to failure because its expected lifetime is 0:39:06.343 longer. But if you're willing to 0:39:09.16 upgrade your discs every two years or one year, 0:39:11.384 then you might be better off with this thing here with the 0:39:14.14 lower meantime to failure, even though its expected 0:39:16.558 operational lifetime is smaller. So both of these numbers are 0:39:19.459 actually meaningful, and it depends a lot on how 0:39:21.731 you're planning to use it. I mean, it's a lot like spare 0:39:24.391 tires on your car. I mean, the spare tire was run 0:39:26.712 perfectly fine as long as you don't exceed 100 miles. 0:39:30 And the moment you exceed 100 miles, then you don't want to 0:39:33.651 use it at all. And it might be a lot cheaper 0:39:36.359 to build the spare tire that runs just 100 miles because the 0:39:40.074 users, you are guaranteed that you will get to a repair shop is 0:39:43.977 in 100 miles. It's the same concept. 0:39:46.181 OK. 0:39:47 0:40:02 So one of the things that we can define, once we have this 0:40:06.178 condition of failure rate is the reliability of the system. 0:40:10.429 We'll define that as the probability, R of T, 0:40:13.654 is the probability that the system's working at time T, 0:40:17.612 given that it was working at time zero, or more generally 0:40:21.717 assuming that everything is always working at time zero, 0:40:25.748 it's the probability that you're OK at time T. 0:40:30 And it turns out that for components in the flat region of 0:40:34.199 this curve, H of T, the conditional failure rate is 0:40:37.884 a constant, on systems that satisfy that, 0:40:40.831 and would satisfy the property that the actual unconditional 0:40:45.178 failure rate is a memory-less process where the probability of 0:40:49.894 failure doesn't depend on how long the system's been running. 0:40:54.315 It turns out that for the systems that satisfy those 0:40:58.073 conditions, which apparently discs do in the operation when 0:41:02.347 they're actually not at the right edge of the curve, 0:41:06.105 which discs do, the reliability, 0:41:08.389 this function goes as the very nice, simple function, 0:41:12.221 which is an exponential decaying function, 0:41:15.242 E to the minus T over MTTF. And this is under two 0:41:20.085 conditions. H of T has to be flat, 0:41:21.79 and the unconditional failure rate has to be something that 0:41:24.788 doesn't depend on how long the system's been running. 0:41:27.475 And for those systems, it's not hard to show that your 0:41:30.214 reliability is just an exponential decaying function, 0:41:32.901 which means you can do a lot of things like predict how long the 0:41:36.157 system is likely to be running, and so on. 0:41:39 And that will tell you when to upgrade things. 0:41:42.593 OK, so given all of this stuff, we now want techniques to cope 0:41:47.463 with failures, cope with faults. 0:41:49.939 And that's what we're going to be looking at for the next few 0:41:54.73 lectures, let's take one simple example of a system first. 0:42:00 And like I said before, all of these systems use 0:42:03.113 redundancy in some form. So the disk fails at a certain 0:42:06.691 rate. Just put in multiple disks, 0:42:08.811 replicate the data across them, and then hope that things 0:42:12.521 survive. So the first kind of redundancy 0:42:15.105 that you might have in the example that I just talked 0:42:18.55 about, spatial redundancy, where the

0:42:17.66 that you might have in the example that I just talked about, spatial redundancy, where the idea is that you have 0:42:22.327 multiple copies of the same thing, and the games we're going 0:42:26.236 to play all have to do with how we're going to manage all these 0:42:30.343 copies. And actually, 0:42:32.88 this will turn out to be quite complicated. 0:42:35.727 We'll use these special copies in a number of different ways. 0:42:39.793 In some examples, we'll apply error correcting 0:42:42.842 codes to make copies of the data or use other codes to replicate 0:42:47.111 the data. We might replicate data and 0:42:49.551 make copies of data in the form of logs which keep track of, 0:42:53.549 you know, you run an operation. You store some results. 0:42:57.208 But at the same time, before you store those results, 0:43:00.732 you also store something in a log server, the original data 0:43:04.73 went away; your log can tell you what to do. 0:43:09 Or you might just do plain and simple copies followed by 0:43:12.762 voting. So the idea is that you have 0:43:15.156 multiple copies of something, and then you write to all of 0:43:19.055 them. In the simplest of schemes, 0:43:21.244 you might write to all of them, and then when you want to read 0:43:25.416 something, you read from all of them. 0:43:27.879 And then just what? And go with the majority. 0:43:31.749 So intuitively that can tolerate a certain number of 0:43:35.318 failures. And all of these approaches 0:43:37.837 have been used. And people will continue to 0:43:40.776 build systems along all of these ideas. 0:43:43.435 But in addition, we're also going to look at 0:43:46.444 temporal redundancy. And the idea here is try it 0:43:49.733 again. So this is different from 0:43:51.902 copies. What it says is you try 0:43:54.002 something. If it doesn't work and you 0:43:56.521 determine that it doesn't work, try it again. 0:44:01 So retry is an example of temporal tricks. 0:44:03.733 But it will turn out will also use not just moving forward and 0:44:07.8 retrying something that we know should be retried, 0:44:11.066 we'll also use the trick of undoing things that we have 0:44:14.666 done. So we'll move both directions 0:44:16.933 on the time axis. We'll retry stuff, 0:44:19.266 but at the same time we'll also undo things because sometimes 0:44:23.266 things have happened that shouldn't have happened. 0:44:26.533 Things went half way. And we really want to back 0:44:29.666 things out. And we were going to use both 0:44:32.333 of these techniques. So one example of spatial 0:44:37.126 redundancy is a voting scheme. And you can apply this to many 0:44:42.228 different kinds of systems. But let's just apply it to a 0:44:46.906 simple example of, there is data stored in 0:44:50.392 multiple occasions. And then whenever data is 0:44:54.134 written, it's written to all of them. 0:44:57.196 And then when you read it, you read from all them, 0:45:01.363 and then you vote. And in a simple model where 0:45:05.668 these components are fail stop, which means that they fail; 0:45:09.253 they just fail. Excuse me, on a simple model 0:45:11.911 where things are not fail stop or fail fast, 0:45:14.569 but just report back this data to you, so you are voting on it, 0:45:18.401 and these results come back. You've written something in 0:45:21.801 them and when you read things back, arbitrary values might get 0:45:25.571 returned if there's a failure. And if there's no failure here, 0:45:29.342 correct values get returned. Then as long as two of these 0:45:33.448 copies are correctly working, or two of these versions are 0:45:36.396 correctly working, then the vote will actually 0:45:38.724 return to you at the correct output. 0:45:40.534 And that's the idea behind voting. 0:45:42.241 So if the reliability of each of these components is some R, 0:45:45.293 that's the probability that the system's working at time T 0:45:48.241 according to that definition of reliability. 0:45:50.465 Then, under the assumption that these are completely independent 0:45:53.724 of each other, which is a big assumption, 0:45:55.793 particularly for software. But it might be a reasonable 0:45:58.586 assumption for something like a disk, under the assumption that 0:46:01.793 these are completely independent, then you could 0:46:04.224 write out the reliability of this three-voting scheme of this 0:46:07.327 thing where you are voting on three outputs. 0:46:11 But you know that the system is correctly working if any two of 0:46:14.852 these are correctly working. So that happens under two 0:46:18.144 conditions. Firstly, all three are 0:46:20.195 correctly working, right? 0:46:21.686 Or, some two of the three are correctly working. 0:46:24.606 And, there's three ways in which you could choose some two 0:46:28.147 of the three. And, one of them is wrongly 0:46:30.633 working. And it turns out that this 0:46:34.402 number actually is very, very large, much larger than R, 0:46:39.485 when R is close to one. And, in general, 0:46:43.089 this is bigger than R when each of the components has high 0:46:48.356 enough reliability, namely, bigger than half. 0:46:52.422 And so, let's say that each of these components has a 0:46:57.227 reliability of 95%. If you work this number out, 0:47:01.54 it turns out to be a pretty big number, much higher than 95%, 0:47:04.841 much closer to one. And, of course, 0:47:06.711 this kind of voting is a bad idea if the reliability of these 0:47:10.012 components is really low. I mean, if it's below one half, 0:47:13.093 then chances are that you're more likely the two of them are 0:47:16.339 just wrong, and you agree on that result. 0:47:18.705 And it turns out to reduce the reliability of the system. 0:47:21.786 Now, in general, you might think that you can 0:47:24.207 build systems out of this basic voting idea, and for various 0:47:27.453 reasons it turns out that this idea has limited applicability 0:47:30.754 for the kinds of things we want to do. 0:47:34 And a lot of that stems from the fact that these are not, 0:47:37.111 in general, in computer systems. 0:47:38.833 It's very hard to design components that are completely 0:47:41.833 independent of each other. It might work out OK for 0:47:44.611 certain hardware components where you might do this voting 0:47:47.777 or other forms of spatial redundancy that gives you these 0:47:50.888 impressive reliability numbers. But for software, 0:47:53.555 this independent assumption turns out to be really hard to 0:47:56.722 meet. And there is an approach to 0:47:58.5 building software like this. It's called N version 0:48:01.222 programming. And it's still a topic of 0:48:04.38 research where people are trying to build software systems out of 0:48:07.912 voting. But you have to pay a lot of 0:48:09.845 attention and care to make sure that these software components 0:48:13.212 that are doing the same function are actually independent, 0:48:16.358 maybe written by different people running on different 0:48:19.284 operating systems, and so on. 0:48:20.83 And that turns out to be a pretty expensive undertaking. 0:48:23.866 It's still sometimes necessary if you want to build something 0:48:27.178 highly reliable. But because of its cost it's 0:48:30.535 not something that is the sort of cookie-cutter technique for 0:48:33.827 achieving highly reliable software systems. 0:48:36.13 And so what we're going to see starting for next time is a 0:48:39.257 somewhat different approach for achieving software reliability 0:48:42.603 that doesn't rely on voting, which won't actually achieve 0:48:45.675 the same degree of reliability as these kinds of systems. 0:48:48.747 but will achieve

a different kind of reliability that we'll 0:48:51.928 talk about starting from next time.