下午好。

我真的很高兴能与这么多自动计算机程序员交谈。

如您所知，计算在我们的社会中变得越来越重要。 现在全世界有数千台计算机。

它们被用于从商业、会计到科学实验的各个领域，谁知道它们将被用来做什么。 所以越来越多的电脑，价格越来越低，尺寸越来越小，电脑曾经是整个大厅的大小，现在它们已经缩小到非常小的比例。

所以现在是计算领域真正快速变化的时期。 所以我认为对编程的未来进行一些展望并思考会很有趣，

根据我们现在所知道的，四十年后编程可能会是什么样子。 所以我今天要讲的，是在过去十年左右的时间里进行了一些非常有趣的研究，我认为这将对未来的编程产生很大的影响。

我想谈谈最近研究中产生的四个重要想法、四个重要主题。 但在讨论这四个重要想法之前，我想首先谈谈采用想法的本质。 所以基本上我们注意到的是技术变化很快。 人的思想是慢慢改变的。 因此采用新技术很容易。 采用新的思维方式可能很困难。

所以，技术方面。

戈登·摩尔（Gordon Moore）他拥有一家名为英特尔的公司。

大约十年前，他观察到计算能力随着时间的推移呈指数级增长，他将这一点推断到了现在，他的目标是正确的，谁知道他能坚持多久。 但这似乎很合理。 因此，从中吸取的教训是，我们可以认为这是理所当然的。 我们只是等待。 我们的计算机变得更快，能力更强。

我们只需等待，这就会发生。 如果我们等待的话，不会发生的事情是。 人们在改变，人们采用新的想法。 举个例子，我相信你们都记得这个人。

老IBM 六五十，你知道，IBM 第一种通用批量生产的计算机。 我们中的很多人都在这个家伙的身上开始学习编程。

一开始，我们都是用绝对二进制编程的。 当我们编码时，实际上是为每条指令编写数字代码。 这就是我们所做的。 那就是编程。 几年后，斯坦·波利出现了，他发明了这个他称之为汇编器的东西。 这就是符号优化汇编程序。 这种语言可以让你用文字书写。 如果你想让电脑添加一些东西，

你会写出广告这个词。 您可以使用符号变量名称而不是硬编码。 内存地址是一种更强大的编程思维方式。

你的工作效率更高。 我的意思是错误少得多。

向这些人展示了汇编，这些人编写二进制文件，但他们根本不感兴趣。 他们只是不明白。 他们认为做这些事情没有任何价值。 因此，新的工作方式可能会遇到很多阻力，需要你忘掉已经学到的东西并以新的方式思考。 甚至可能存在公然的敌意。 所以约翰尼·冯·诺依曼，伟大的科学家，他发明了我们使用的冯·诺依曼计算机架构，

他说了很多其他的话，我不明白为什么有人需要机器代码以外的任何东西。 还有一次他。 所以他有一群学生，学生们都在用二进制编码。 有一次，他的一个学生花了一点时间编写自己的小汇编程序，这样他就可以编写一个汇编程序。 冯·诺依曼对他感到愤怒，对他浪费宝贵的机器时间进行组装感到愤怒。

那是应该为人们做的文书工作，对吧？，所以我们看到同样的故事发生了一会儿，当 John bacus 和朋友们提出了这个想法，他们称之为 fortran，这种所谓的高级语言 您可以像编写数学符号一样写出公式。 你可以写出循环。 这被展示给程序员大会。

他们再一次不感兴趣。 他们没有看到其中的任何价值，他们只是没有得到它。

当我谈论我今天要谈论的四个重要想法时，我希望你们牢记这一点。 人们很容易认为，由于摩尔定律，技术总是变得更好，因为计算机总是变得更强大。 但要求人们忘掉已经学到的东西并以新的方式思考的想法往往会遇到巨大的阻力。

这里的人们认为他们知道自己在做什么。 他们认为他们知道什么是编程。 这种编程不是编程。 因此，采用新想法将会遇到很多阻力。

今天我想讲的四个想法。 这一切都来自最近的研究。 第一个是今天我们用代码编写程序。 我们编写了计算机要做的指令列表。 这是关于直接操作数据的一些非常有趣的研究，您可以直接操作数据结构，并隐式地构建一个供计算机遵循的程序。 今天我要讲的第二件事

我们编写程序。 基本上，这是一个让计算机对编程进行有趣研究的过程，使用目标，告诉计算机你想要什么，而不是如何做。 计算机本身就知道如何做到这一点。 第三，今天我们使用文本文件中的文本行进行编程。

人们正在做一些非常了不起的事情，就像过去五年或十年一样。 他们将视频显示器连接到计算机上。 当你这样做时，一切都会改变，我们可以开始思考信息的空间表示。 第四件事是我们以顺序编程模型进行编程。 基本上，这里有一堆说明。 计算机一个接一个地完成它们，

但硬件正在改变。 因此，我们将看到大规模并行硬件，并且我们将需要一个健全的并行编程模型来在该硬件上进行编程。

所以。

我要谈论的第一件事是数据的直接操作。 我将展示 Ivan seland 大约十年前完成的这个博士论文。 这个系统称为画板，画板是一个允许您在视频显示器上绘制图片的系统。 于是他拿起光笔，把它放在了屏幕上。 所以他画了那条线，画了那条线，画了更多的线，

做这个小上衣。 他，他想在这里拉一颗铆钉。

而且他画画真的很草率。 它有点向侧面倾斜。 它有点畸形。 因此，他所做的就是按住开关并向计算机系统指示其中几条线，表明他希望这些线相互垂直。 因此，系统运行一个迭代求解器，有点像意愿周围的线，并找出如何将它们变成相互特定的东西，以及如何将它们变成矩形。

所以。 基本上，

系统对矩形一无所知。 他能够通过直接应用一组约束来让它绘制一个矩形。 使它成为一个程序而不仅仅是一张图片的原因在于这些约束是动态维护的。 所以他得到了他画的铆钉，他调整了它这个角的大小，并调整了其他一些东西的大小。 解算器重新启动，将其变回完美的矩形。

所以本质上他创建了一个绘制矩形的程序，但他并不是通过编写代码来完成的。 他通过直接操作数据并直接对其应用一组约束来做到这一点。 这是一个简单的例子。 然后他开始做一些更奇特的事情，比如这是一个桥梁模拟。 所以它实际上模拟了桥梁的物理原理。 这是他手绘的。 这里的数字代表桥梁跨度的张力。

他可以改变悬挂在桥梁中心的负载的重量，并且它会变形，而系统画板系统对桥梁一无所知。 他通过直接绘制并直接应用一组非常通用的约束来创建这个桥梁模拟程序。

所以我绝对认为这在 34 年后将变得非常重要。 我可以想象通过直接操作 j 结构并让它构建代码来进行编程，但尤其是对于像这样的视觉或物理事物。 因此，如果你说几十年后，我想我们会在某种计算机网络上获得某种文档格式。

我确信我们将通过直接操作来创建所有这些文档。 不会有任何类似的标记语言或样式表语言，对吧？ 那样的话，那就没有意义了。 1962 年，伊万·萨瑟兰 (Ivan Sutherland) 向我们展示了如何做到这一点。因此，未来一切都将是直接操纵，这将是非常棒的。

所以我想谈的第二件事是使用目标进行编程。

我们通过 sketchad 的约束系统看到了一些这样的情况。 于是伊万突然想画一个矩形。 他没有编写一个程序来绘制矩形的每条边。 他应用了一组约束，系统本身就知道如何绘制该矩形。 所以他说出了他想要的东西。 我希望事物相互垂直。 他没有说怎么做。 解决者想出了如何做到这一点。

几年前出现的另一个很好的例子是 Carl hewitt 正在开发一个名为 planner 的系统，这真的很棒。 它实际上是双向的，所以它可以向前推理。 在程序上，它可以从目标向后推理。 因此，如果您告诉规划器苹果是红色的，那么如果您给它一个苹果，它就会知道，啊哈，它是红色的。 但你也可以说我想要红色，它会说，

哦，我们试试苹果吧。 因此，您可以根据目标、您希望从计划中获得的结果来表达您的计划，但您也可以提供实现某些类型的子目标的程序策略。 非常非常有趣的编程思维方式，这导致了另一个系统。 几年后称为序言，它只保留了向后的部分规划器。 因此，我将空间计划作为目标，系统本身使用搜索或其他方式来尝试找出如何实现这些目标。

因此，这导致了一种称为逻辑编程的编程流派，但这并不是这里真正重要的部分。 重要的是把你的程序表达为你想要做的事情，而不是一套关于如何做的说明，让他们自行估算，弄清楚如何做。

同类概念的另一个例子是模式匹配。 所以我相信你们都记得你们都记得雪球。 这是文本操作语言。 如果你有一堆文本想要浏览，你可以向其中扔一个滚雪球脚本或程序。 Snowball 具有内置的模式匹配功能，因此您可以表达想要与文本匹配的模式。 稍晚一些，

Ken Thompson，所以贝尔实验室致力于这个系统，他们称之为 unix。

我知道，对，unix，但他采用了 k 的正则表达式概念来对文本进行模式匹配。

因此，当你进行模式匹配时，如果你想消化一大块文本，你就不会去编写一个按程序遍历它的解析器。 你提供一个模式，这种模板。 这就是我正在寻找的东西，系统本身会找出如何匹配文本。 反对这种模式。

所以。

所有这些例子，

草图本、约束规划器、序言、模式匹配，同样，它们都是给计算机高层次目标的例子，说这就是我正在寻找的东西，让计算机自己弄清楚如何去做， 我们看到了一些这样的事情并优化了泰勒斯，但我认为从现在开始的几十年内它会非常普遍。 这对于学校指导的东西如此重要的原因与此有关，

舔打火机的想法正在流行。 众所周知，利克利特正在领导政府资助机构 ARPA。 他一直在推动全球计算机网络的想法，将世界上所有的计算机连接起来。 他将其称为星际计算机网络，因为他知道工程师总是提供最低限度的服务。 所以如果他要求一个跨越银河系的网络

他希望至少能得到一个跨越世界的东西。 人们称之为 AR 教育网。 现在它正在变成某种兴趣。 这是一个可爱的想法。 可能有用。 当你拥有这种全球计算机网络时，你就会遇到所谓的与外星人通信的问题。

所以他把它放在这里。 这个问题本质上是科幻小说作家所讨论的问题。 如何在完全不相关的智能生物之间开始沟通？ 我会解释他的意思。

假设您有一个计算机网络，并且有一些程序，该程序是由某人在某个时间用某种语言编写的，该语言使用某种协议。 你这里有另一个程序，是其他人在其他时间编写的，说的是完全不同的语言，写的是完全不同的语言。 这两个程序彼此一无所知。

但在某个时刻。

该程序将找出该程序需要的服务。

他们必须互相交谈。 所以这两个程序对彼此一无所知，完全不同的时间。 现在他们需要能够沟通。

那么他们要怎么做呢？

好吧，只有一个真正有效的答案，那就是他们必须弄清楚如何彼此交谈。

他们需要互相谈判，因为他们必须互相试探。 他们必须动态地找出一种共同语言，以便交换信息。 并实现人类程序员赋予他们的目标。 这就是为什么当我们拥有互联网时，这个目标导向的东西将变得如此重要，因为你无法编写程序，因为我们不知道与这些远程程序对话的程序。

这些程序本身必须找出相互交谈的程序并实现更高层次的目标。 因此，如果我们没有这个全球网络，我认为这是唯一能够扩展不起作用的模型，这将是一场彻底的灾难。

我要在这里造一个术语，API，这个概念是指你有一个人类程序员针对某个远程程序公开的固定接口进行编写。 首先，这要求程序已经相互了解。

当你用这种语言编写这个程序时，现在它们就联系在一起了。 因此，第一个程序无法出去寻找并找到实现它们绑定在一起的相同服务的其他程序。 如果这个语言的改变破坏了这个语言，它真的很脆弱，无法扩展。 最糟糕的是，这基本上是机器代码问题。 你需要一个人来处理一些低级的细节，而这些细节应该由机器来处理。

所以我非常有信心这永远不会发生。 未来我们不会再有 API。 程序知道如何弄清楚如何相互交谈，这需要编程和目标。

我想谈的第三个重要想法是信息的空间表示。 所以今天，我们的程序基本上是很多行文本，一个充满文本行的大文件，当您的程序位于一堆打孔卡或纸带或磁带上时，这是有意义的。 非常线性的媒体让您的节目采用这种线性形式是有意义的。 如果您使用的是电传打字机，那么telltype就是用来吐出文本行的。

这就是它的作用。 所以你的程序当然会以文本行的形式出现。 但正如我所提到的，人们现在正在做一些非常疯狂的事情，那就是连接计算机的视频显示器。 当您将视频显示器连接到计算机时，您可以开始将计算机视为一张非常动态的纸，您可以在其中空间地表示事物。 所以。

道格·恩格尔巴特在斯里。

有一个他称之为在线系统的系统。 五年前他做了一个大型演示。 您可能已经看到过，该系统有很多非常了不起的东西。 最引人注目的之一是在屏幕上、视频屏幕上显示信息的概念。 所以他有这个设备，这个设备叫做鼠标，你可以在桌子上滚动，这很难解释，但是你可以用它来指向屏幕的不同部分，并表明你想要有关某事物的更多信息 你所指的。

它还具有不同信息视图的概念。 所以你可以在这里看到他在列表中有一些数据，他们可以翻转它并查看与这种二维图相同的数据。 所以真正开始思考我们如何在空间上表示信息动态信息。

另一个伟大的系统也同时出现。 来自兰德公司的产品被称为“grail”，这是一个使用视频显示器上的流程图进行编程的系统，这里的输入设备是平板电脑上的手写笔，您可以放下这些流程图，让我向您展示它是如何工作的。

程序员正在绘制这个盒子，并且完全自由地处理它，他绘制了一个盒子，系统将其识别为一个盒子并将其转换为流程图框，因此为他正在做的这些绘图分配语义。

他想给它贴上标签，所以他就开始写信。 系统识别出了他的笔迹。

1968年系统识别手写体。 将其转换为文本。

在这里，他用一根线等将这个盒子连接到那个盒子。 所以这都是非常直接的操纵。 如果他想摆脱这条线的话。 他只是潦草地写下然后就走了。 因此，真正思考当你有视频显示、当你可以用二维表达事物时，编程意味着什么。

但当我谈论信息的空间表示时，我不仅仅谈论流程图之类的东西。 施乐在帕洛阿尔托有一个小型研究中心。 那里有一些孩子正在研究他们称之为“闲聊”的东西。 在闲聊中，源代码以文本形式表达，但没有包含一大堆代码的大而长的文本文件。 它以空间方式组织，

这就是他们所说的浏览器。 因此，在此列表中，这是所有类的集合，这是该集合中的所有类，这是该类中的所有协议，这是该协议中的所有方法，这是该特定方法的源代码。 所以方法定义是文本，但它们不是一大行文本。 它们是按空间组织的，因此您可以非常快速地浏览系统并查看正在发生的情况。

所以在角巴特和圣杯之间，

闲聊，这些非常不同的空间表示信息的方式。 所以我完全有信心四十年后我们不会在文本文件中编写代码。

我们已经被告知了方法并作为旁注。

我刚刚向你们展示了所有这些系统，恩格尔，巴特系统，圣杯，smalltalk，伊利诺伊大学正在进行的称为柏拉图的系统也是非常有趣的系统。 这些是交互式计算新浪潮的一部分。 当您坐在计算机前时，您实际上是在与计算机进行实时交互。 这些人知道他们正在努力证明这个新概念。

因此，他们从最底层设计了系统，以便能够立即做出响应。 用户界面始终立即响应。 当你与任何事物互动时，你会立即得到回应。 所以它有点模拟物理对象。 因此，如果交互式计算起飞，而且我认为它会起飞，那么我认为很明显，四十年后我们的用户界面，如果你与它们交互，

您永远不会遇到任何延迟或滞后。 是的，因为这些人证明了拥有即时响应的用户界面是多么重要，他们在六十年代就这样做了。 因此，随着我们的计算机速度提高一百万倍，显然操作系统的用户界面没有任何延迟或滞后的理由。 所以这将非常令人兴奋。

第四件事是我想说的。

是并行编程。

所以今天我们的程序基本上是一系列指令。 计算机这样做，然后那样做，然后那样做，然后那样做。

我们对顺序模型进行编程的原因之一与硬件有关。

因此，我们一直在使用这种称为冯·诺依曼计算机体系结构的计算机体系结构，其中有一个处理器，然后它连接到一个大内存，并从内存中获取单词。

因此，当您只有一个进程，一次只能做一件事时，顺序编程模型就有意义了，您给出了进程要做的一系列事情，它只是执行其中的每一项。

不过，冯·诺依曼架构的一个特点就是这一点。

大部分内存大部分时间都是空闲的，所以你这里有一个小处理器，它的处理速度尽可能快。 但只有一个内存字被访问过。 剩下的记忆只是坐在那里等待。 当你的处理器是由真空管或继电器制成的，而且它又大又贵，而你的内存是由核心或转鼓制成的，而且又大又贵，而且又不同时，这就有效了，

那么你就可以逃脱惩罚，但是。 我们开始看到了。 一个令人难以置信的发明进入了计算领域，现在的计算，我认为这将改变一切，那就是集成电路半导体集成电路。

这是一家叫英特尔的公司制造的东西。 它称为微处理器，是一块硅片上的完整处理器。

因此，整个处理器都是由晶体管制成的，而晶体管只是蚀刻在硅上的一个小图片。 整个电路只是蚀刻在硅片上的一张复杂的大图。 所以我们的处理器只是由晶体管和硅制成的。 我们的存储器也将由硅上的晶体管制成。 都是一样的东西。

因此，当您从这个角度观察冯·诺依曼架构时，您会发现这些晶体管正在努力处理事物。 这里有大量的晶体管，其中大部分只是坐着。 等待。 他们不处理。 他们什么也没做。 所以。

如果你想。 如果您确实想最大限度地提高硅片的处理量，请让这些晶体管发挥作用。

你需要开始关注类似这样的事情。

因此，计算机希望在硅上成为许多小型计算机，就像一大群微型计算机，它们有自己的进程、自己的小状态、做自己的事情、相互通信。 这就是最大化每个硅面积的计算量的方法。 还有秤。 因此，当过渡变小时，当硅日记变大时，你就有了所有这些额外的空间，

你只需用更多完成的处理器来填充它。 真的很容易。

这就是我们将来要编程的架构。

除非，你知道，除非英特尔以某种方式控制了市场的竞争进程，以及它如何推动这种架构向前发展三十年。 但这不会发生。 我们将对这些事情进行编程。 当你拥有这个硬件时，你必须开始思考如何，我们如何在其上编程，我们针对此类硬件的编程模型是什么？

以及我们今天编程的方式。

是有线程和锁的。 你有一些连续的控制线程，你可以通过将它们多路复用到处理器上来假装它们是并行的，并且它们试图将彼此锁定在共享资源之外。

就像这样永远行不通，对吧？ 这无法扩展。 您无法推断数百个线程同时使用线程和锁来访问同一共享内存。 他们有点死胡同，对吧？ 所以我认为四十年来我们仍在使用线程和锁。 我们应该收拾行李回家，因为我们作为工程领域显然失败了。 那么，如果不是威胁和锁定，那么什么会起作用呢？

卡尔休伊特，他们就是卡尔休伊特。 Planner 提出了这个他称之为演员模型的想法。

因此，演员模型是受物理学启发的计算模型。 所以在物理学中，你有所有这些粒子，并且所有粒子都独立地做自己的事情。 他们有自己的小状态，并且与周围的人互动。 卡尔也以同样的方式思考计算机流程。 你有一大堆流程。 他们都在异步地做自己的小事。

他们很少发表声明，只是互相发送消息。 非常有趣、非常新颖且令人兴奋的编程思维方式。 这么多。 现在有点升温。 法国的J kh over 有一些想法。 我认为托尼·霍（Tony hor）已经涉足其中。 他将调用通信顺序进程，甚至拉面米勒也可能会加入其中。 这里发生了非常令人兴奋的事情。

现在来说说这个。 这个特定的、这些特定模型的细节并不重要。

我确实认为，如果演员模型被瑞典电话公司或其他公司选中的话，那会很酷。 就这样吧。 很奇怪，但是。

但这里重要的是我们将拥有大规模并行硬件。 我们需要一个适合硬件的健全的并行编程模型，而我们将使用类似的模型。

这就是我想谈的四件事。

直接操作数据，类似于画板，您可以在其中动态绘制图片，直接向这些图片添加约束，操作数据结构而不是为程序编写指令。

使用目标和约束进行编程。 像草图本、约束、计划表和序言之类的东西。

正则表达式，其他类型的模式匹配，你告诉计算机你想做什么，而计算机本身有求解器来计算如何做到这一点。

信息的空间表示。

我们将不再有文本文件。 我们将在空间上表示信息，因为我们有视频显示器。

根本上是并行的思维方式。 并行硬件、并行程序模型。 不再有线程和锁。 不再有顺序思维。

这就是我想谈的四件事。

你知道，我试图对未来做出一些预测，但你无法真正预测未来，对吗？ 所以这些都是一些好主意。 我不知道他们会发生什么。 想法有时会分裂又融合，有时会流行，有时会过时。 所以，你知道，任何事情都有可能发生。

但我确实认为，如果四十年后我们仍然在程序、文本文件、顺序编程模型中进行编码，那将是一种耻辱。

我认为会的。 这表明我们没有从计算机科学的这个真正丰饶的时期学到任何东西。

那么这将是一场悲剧。

但。

如果这些想法被遗忘，那么比这些想法没有被使用更悲剧的是，如果有人向他们展示这些东西并真正感到惊讶的话。

但这还不是最大的悲剧。 这并不是真正的悲剧。 真正的悲剧是，如果人们忘记了你首先可以对编程模型有新的想法。

那么让我解释一下我的意思。

我认为最糟糕的情况是，如果下一代程序员在成长过程中从未接触过这些想法。 下一代，下一代程序员，在成长过程中只会被灌输一种编程思维方式。 所以他们就采用这种编程方式。 他们充实了所有细节。 你知道，它们有点解决特定的编程模型。

他们弄清楚了这一切，然后将其传授给下一代。 所以第二代长大后就会想，哦，一切都已经弄清楚了。 我们知道什么是编程，我们知道我们在做什么。

他们是在教条中长大的。

一旦你在教条中长大，就很难摆脱它。

您知道为什么所有这些想法和许多其他好想法出现在六十年代这个特定的时期吗？ 七十年代初期？ 为什么这一切会发生呢？

这是因为。

技术。 技术发展到可以真正用计算机做事的程度已经够晚了，但还太早，没有人知道什么是编程。

没有人知道编程应该是什么，他们也知道自己不知道。 所以他们喜欢尝试一切。 他们的思想完全自由，他们只是说也许我们可以像这样专业。 也许我们可以这样编程。 他们只是尝试了他们能想到的一切。

所以。 最多。

作为一个有创造力的人，你可能会产生危险的想法。 就是认为你知道自己在做什么。

因为一旦你认为你知道自己在做什么。 你不再四处寻找其他的做事方式，也不再能够看到其他的做事方式。 你变得盲目，你变得像这里的这些人一样，编码和二进制。 有人向他们展示汇编语言，所以有人向他们展示 foran，但他们甚至看不到。 这完全超出了他们的想象，因为他们知道自己在做什么。

他们知道什么是编程。 这是编程而不是编程。 所以他们完全错过了这种更强大的思维方式。

所以这次演讲的信息，你知道，这不是真的，这不是真正的东西，对吧？ 这次演讲的信息是。 如果你不想成为这样的人。 如果你想开放并接受新的思维方式，就必须发明新的思维方式。 我认为第一步是你必须对自己说。

我不知道我在做什么。 作为一个领域，我们不知道自己在做什么。

我想你不得不说我们不知道什么是编程，我们不知道什么是计算，我们甚至不知道什么是计算机。

而一旦你。

真正理解这一点并且一旦你真正相信这一点。

然后你就自由了。 你可以想任何事情。

谢谢。