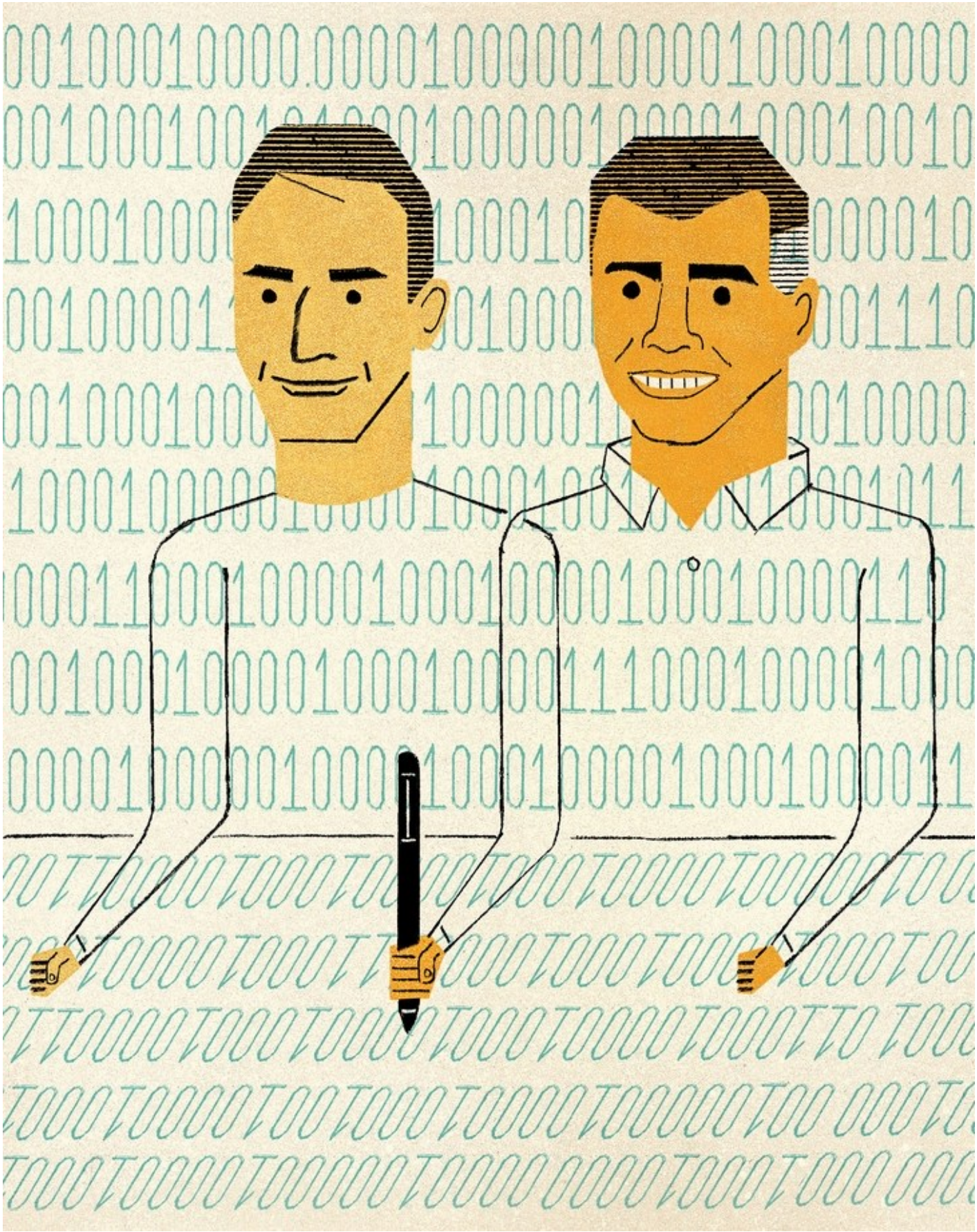
THE NEW YORKER

THE FRIENDSHIP THAT MADE GOOGLE HUGE

Coding together at the same computer, Jeff Dean and Sanjay Ghemawat changed the course of the company—and the Internet.

By James Somers



The company's top coders seem like two halves of a single mind. Illustration by David Plunkert

0:00 / 41:27

Audio: Listen to this article. To hear more, download the Audm iPhone app.

One day in March of 2000, six of Google's best engineers gathered in a makeshift war room. The company was in the midst of an unprecedented emergency. In October, its core systems, which crawled the Web to build an "index" of it, had stopped working. Although users could still type in queries at google.com, the results they received were five months out of date. More was at stake than the engineers realized. Google's co-founders, Larry Page and Sergey Brin, were negotiating a deal to power a search engine for Yahoo, and they'd promised to deliver an index ten times bigger than the one they had at the time—one capable of keeping up with the World Wide Web, which had doubled in size the previous year. If they failed, google.com would remain a time capsule, the Yahoo deal would likely collapse, and the company would risk burning through its funding into oblivion.

In a conference room by a set of stairs, the engineers laid doors across sawhorses and set up their computers. Craig Silverstein, a twenty-seven-year-old with a small frame and a high voice, sat by the far wall. Silverstein was Google's first employee: he'd joined the company when its offices were in Brin's living room and had rewritten much of its code himself. After four days and nights, he and a Romanian systems engineer named Bogdan Cocosel had got nowhere. "None of the analysis we were doing made any sense," Silverstein recalled. "Everything was broken, and we didn't know why."

Silverstein had barely registered the presence, over his left shoulder, of Sanjay Ghemawat, a quiet thirty-three-year-old M.I.T. graduate with thick eyebrows and black hair graying at the temples. Sanjay had joined the company only a few months earlier, in December. He'd followed a colleague of his—a rangy, energetic thirty-one-year-old named Jeff Dean—from Digital Equipment Corporation. Jeff had left D.E.C. ten months before Sanjay. They were unusually close, and preferred to write code jointly. In the war room, Jeff rolled his chair over to Sanjay's desk, leaving his own empty. Sanjay worked the keyboard while Jeff reclined beside him, correcting and cajoling like a producer in a news anchor's ear.

Jeff and Sanjay began poring over the stalled index. They discovered that some words were missing—they'd search for "mailbox" and get no results—and that others were listed out of order. For days, they looked for flaws in the code, immersing themselves in its logic. Section by section, everything checked out. They couldn't find the bug.

Programmers sometimes conceptualize their software as a structure of layers ranging from the user interface, at the top, down through increasingly fundamental strata. To venture into the bottom of this structure, where the software meets the hardware, is to turn away from the Platonic order of code and toward the elemental universe of electricity and silicon on which it depends. On their fifth day in the war room, Jeff and Sanjay began to suspect that the problem they were looking for was not logical but physical. They converted the jumbled index file to its rawest form of representation: binary code. They wanted to see what their machines were seeing.

On Sanjay's monitor, a thick column of 1s and 0s appeared, each row representing an indexed word. Sanjay pointed: a digit that should have been a 0 was a 1. When Jeff and Sanjay put all the missorted words together, they saw a pattern—the same sort of glitch in every word. Their machines' memory chips had somehow been corrupted.

Sanjay looked at Jeff. For months, Google had been experiencing an increasing number of hardware failures. The problem was that, as Google grew, its computing infrastructure also expanded. Computer hardware rarely failed, until you had enough of it—then it failed all the time. Wires wore down, hard drives fell apart, motherboards overheated. Many machines never worked in the first place; some would unaccountably grow slower. Strange environmental factors came into play. When a supernova explodes, the blast wave creates high-energy particles that scatter in every direction; scientists believe there is a minute chance that one of the errant particles, known as a cosmic ray, can hit a computer chip on Earth, flipping a 0 to a 1. The world's most robust computer systems, at NASA, financial firms, and the like, used special hardware that could tolerate single bit-flips. But Google, which was still operating like a startup, bought cheaper computers that lacked that feature. The company had reached an inflection point. Its computing cluster had grown so big that even unlikely hardware failures were inevitable.

Together, Jeff and Sanjay wrote code to compensate for the offending machines. Shortly afterward, the new index was completed, and the war room disbanded.

Silverstein was flummoxed. He was a good debugger; the key to finding bugs was getting to the bottom of things. Jeff and Sanjay had gone deeper.

Until the March index debacle, Google's systems had been rooted in code that its founders had written in grad school, at Stanford. Page and Brin weren't professional software engineers. They were academics conducting an experiment in search technology. When their Web crawler crashed, there was no informative diagnostic message—just the phrase "Whoa, horsey!" Early employees referred to BigFiles, a piece of software that Page and Brin had written, as BugFiles. Their all-important indexing code took days to finish, and if it encountered a problem it had to re-start from the beginning. In the parlance of Silicon Valley, Google wasn't "scalable."

We say that we "search the Web," but we don't, really; our search engines traverse an index of the Web—a map. When Google was still called BackRub, in 1996, its map was small enough to fit on computers installed in Page's dorm room. In March of 2000, there was no supercomputer big enough to process it. The only way that Google could keep up was by buying consumer machines and wiring them together into a fleet.

Because half the cost of these computers was in parts that Google considered junk—floppy drives, metal chassis—the company would order raw motherboards and hard drives and sandwich them together. Google had fifteen hundred of these devices stacked in towers six feet high, in a building in Santa Clara, California; because of hardware glitches, only twelve hundred worked. Failures, which occurred seemingly at random, kept breaking the system. To survive, Google would have to unite its computers into a seamless, resilient whole.

Side by side, Jeff and Sanjay took charge of this effort. Wayne Rosing, who had worked at Apple on the precursor to the Macintosh, joined Google in November, 2000, to run its hundred-person engineering team. "They were the leaders," he said. Working ninety-hour weeks, they wrote code so that a single hard drive could fail without bringing down the entire system. They added checkpoints to the crawling process so that it could be re-started midstream. By developing new encoding and compression schemes, they effectively doubled the system's capacity. They were relentless optimizers. When a car goes around a turn, more ground must be covered by the outside wheels; likewise, the outer edge of a spinning hard disk moves faster than the inner one. Google had moved the most frequently accessed data to the outside, so that bits could flow faster under the read-head, but had left the inner half empty; Jeff and Sanjay used the space to store preprocessed data for common search queries. Over four days in 2001, they proved that Google's index could be stored using fast random-access memory instead of relatively slow hard drives; the discovery reshaped the company's economics. Page and Brin knew that users would flock to a service that delivered answers instantly. The problem was that speed required computing power, and computing power cost money. Jeff and Sanjay threaded the needle with software.

Alan Eustace became the head of the engineering team after Rosing left, in 2005. "To solve problems at scale, paradoxically, you have to know the smallest details," Eustace said. Jeff and Sanjay understood computers at the level of bits. Jeff once circulated a list of "Latency Numbers Every Programmer Should Know." In fact, it's a list of numbers that almost no programmer knows: that an L1 cache reference usually takes half a nanosecond, or that reading one megabyte sequentially from memory takes two hundred and fifty microseconds. These numbers are hardwired into Jeff's and Sanjay's brains. As they helped spearhead several rewritings of Google's core software, the system's capacity scaled by orders of magnitude. Meanwhile, in the company's vast data

centers technicians now walked in serpentine routes, following software-generated instructions to replace hard drives, power supplies, and memory sticks. Even as its parts wore out and died, the system thrived.

Today, Google's engineers exist in a Great Chain of Being that begins at Level 1. At the bottom are the I.T. support staff. Level 2s are fresh out of college; Level 3s often have master's degrees. Getting to Level 4 takes several years, or a Ph.D. Most progression stops at Level 5. Level 6 engineers—the top ten per cent—are so capable that they could be said to be the reason a project succeeds; Level 7s are Level 6s with a long track record. Principal Engineers, the Level 8s, are associated with a major product or piece of infrastructure. Distinguished Engineers, the Level 9s, are spoken of with reverence. To become a Google Fellow, a Level 10, is to win an honor that will follow you for life. Google Fellows are usually the world's leading experts in their fields. Jeff and Sanjay are Google Senior Fellows—the company's first and only Level 11s.

The Google campus, set beside a highway a few minutes from downtown Mountain View, is a series of squat, unattractive buildings with tinted windows. One Monday last summer, after a morning of programming together, Jeff and Sanjay went to lunch at a campus cafeteria called Big Table, which was named for a system they'd helped develop, in 2005, for treating numberless computers as though they were a single database. Sanjay, who is tall and thin, wore an ancient maroon Henley, gray pants, and small wire-frame glasses. He spied a table outside and walked briskly to claim it, cranking open the umbrella and taking a seat in the shade. He moved another chair into the sun for Jeff, who arrived a minute later, broad-shouldered in a short-sleeved shirt and wearing stylish sneakers.

Like a couple, Jeff and Sanjay tell stories together by contributing pieces of the total picture. They began reminiscing about their early projects.

"We were writing things by hand," Sanjay said. His glasses darkened in the sun. "We'd rewrite it, and it was, like, 'Oh, that seems near to what we wrote last month.' "

"Or a slightly different pass in our indexing data," Jeff added.

"Or slightly different," Sanjay said. "And that's how we figure out—"

"This is the essence," Jeff said.

"—this is the common pattern," Sanjay said, finishing their thought.

Jeff took a bite of the pizza he'd got. He has the fingers of a deckhand, knobby and leathery; Sanjay, who looks almost delicate in comparison, wondered how they ended up as a pair. "I don't quite know how we decided that it would be better," he said.

"We've been doing it since before Google," Jeff said.

"But I don't know why we decided it was better to do it in front of one computer instead of two," Sanjay said.

"I would walk from my D.E.C. research lab two blocks away to his D.E.C. research lab," Jeff said. "There was a gelato store in the middle."

"So it's the gelato store!" Sanjay said, delighted.

Sanjay, who is unmarried, joins Jeff, his two daughters, and his wife, Heidi, on vacations. Jeff's daughters call him Uncle Sanjay, and the five of them often have dinner on Fridays. Sanjay and Victoria, Jeff's eldest, have taken to baking. "I've seen his daughters grow up," Sanjay said, proudly. After the Google I.P.O., in 2004, they moved into houses that are four miles apart. Sanjay lives in a modest three-bedroom in Old Mountain View; Jeff designed his house, near downtown Palo Alto, himself, installing a trampoline in the basement. While working on the house, he discovered that although he liked designing spaces, he didn't have patience for what he calls the "Sanjay-oriented aspects" of architecture: the details of beams, bolts, and loads that keep the grand design from falling apart.

"I don't know why more people don't do it," Sanjay said, of programming with a partner.

"You need to find someone that you're gonna pair-program with who's compatible with your way of thinking, so that the two of you together are a complementary force," Jeff said.

They pushed back from the table and set out in search of soft-serve, strolling through Big Table and its drifting Googlers. Of the two, Jeff is more eager to expound, and while they walked he shared his soft-serve strategy. "I do the squish. I think the

pushing-up approach adds stability," he said. Sanjay, pleased and intent, swirled a chocolate-and-vanilla mix into his cone.

In his book "Collaborative Circles: Friendship Dynamics and Creative Work," from 2001, the sociologist Michael P. Farrell made a study of close creative groups—the French Impressionists, Sigmund Freud and his contemporaries. "Most of the fragile insights that laid the foundation of a new vision emerged not when the whole group was together, and not when members worked alone, but when they collaborated and responded to one another in pairs," he wrote. It took Monet and Renoir, working side by side in the summer of 1869, to develop the style that became Impressionism; during the six-year collaboration that gave rise to Cubism, Pablo Picasso and Georges Braque would often sign only the backs of their canvases, to obscure which of them had completed each painting. ("A canvas was not finished until both of us felt it was," Picasso later recalled.) In "Powers of Two: Finding the Essence of Innovation in Creative Pairs," the writer Joshua Wolf Shenk quotes from a 1971 interview in which John Lennon explained that either he or Paul McCartney would "write the good bit, the part that was easy, like 'I read the news today' or whatever it was." One of them would get stuck until the other arrived—then, Lennon said, "I would sing half, and he would be inspired to write the next bit and vice versa." Everyone falls into creative ruts, but two people rarely do so at the same time.

In the "theory building" phase of a new science or art, it's important to explore widely without getting caught in dead ends. François Jacob, who, with Jacques Monod, pioneered the study of gene regulation, noted that by the mid-twentieth century most research in the growing field of molecular biology was the result of twosomes. "Two are better than one for dreaming up theories and constructing models," Jacob wrote. "For with two minds working on a problem, ideas fly thicker and faster. They are bounced from partner to partner. They are grafted onto each other, like branches on a tree. And in the process, illusions are sooner nipped in the bud." In the past thirty-five years, about half of the Nobel Prizes in Physiology or Medicine have gone to scientific partnerships.

After years of sharing their working lives, duos sometimes develop a private language, the way twins do. They imitate each other's clothing and habits. A sense of humor osmoses from one to the other. Apportioning credit between them becomes impossible.

But partnerships of this intensity are unusual in software development. Although developers sometimes talk about "pair programming"—two programmers sharing a single computer, one "driving" and the other "navigating"—they usually conceive of such partnerships in terms of redundancy, as though the pair were co-pilots on the same flight. Jeff and Sanjay, by contrast, sometimes seem to be two halves of a single mind. Some of their best-known papers have as many as a dozen co-authors. Still, Bill Coughran, one of their managers, recalled, "They were so prolific and so effective working as a pair that we often built teams around them."

In 1966, researchers at the System Development Corporation discovered that the best programmers were more than ten times as effective as the worst. The existence of the so-called "10x programmer" has been controversial ever since. The idea venerates the individual, when software projects are often vast and collective. In programming, few achievements exist in isolation. Even so—and perhaps ironically—many coders see the work done by Jeff and Sanjay, together, as proof that the 10x programmer exists.

Jeff was born in Hawaii, in July of 1968. His father, Andy, was a tropical-disease researcher; his mother, Virginia Lee, was a medical anthropologist who spoke half a dozen languages. For fun, father and son programmed an IMSAI 8080 kit computer. They soldered upgrades onto the machine, learning every part of it.

Jeff and his parents moved often. At thirteen, he skipped the last three months of eighth grade to help them at a refugee camp in western Somalia. Later, in high school, he started writing a data-collection program for epidemiologists called Epi Info; it became a standard tool for field work and, eventually, hundreds of thousands of copies were distributed, in more than a dozen languages. (A Web site maintained by the Centers for Disease Control and Prevention, "The Epi Info Story," includes a picture of Jeff at his high-school graduation.) Heidi, whom Jeff met in college, at the University of Minnesota, learned of the program's significance only years later. "He didn't brag about any of that stuff," she said. "You had to pull it out of him." Their first date was at a women's basketball game; Jeff was in a gopher costume, cheerleading.

Jeff's Ph.D. focussed on compilers, the software that turns code written by people into machine-language instructions optimized for computers. "In terms of sexiness, compilers are pretty much as boring as it gets," Alan Eustace said; on the other hand, they get you "very close to the machine." Describing Jeff, Sanjay twirled his index

finger around his temple. "He has a model going on as you're writing code," he said. " 'What is the performance of this code going to be?' He'll think about all the corner cases almost semi-automatically."

Sanjay didn't touch a computer until he went to Cornell, at the age of seventeen. He was born in West Lafayette, Indiana, in 1966, but grew up in Kota, an industrial city in northern India. His father, Mahipal, was a botany professor; his mother, Shanta, took care of Sanjay and his two older siblings. They were a bookish family: his uncle, Ashok Mehta, remembers buying a copy of "The Day of the Jackal," by Frederick Forsyth, its binding badly worn, and watching the Ghemawat children read the broken book together, passing pages along as they finished. Sanjay's brother, Pankaj, became the youngest faculty member ever awarded tenure at Harvard Business School. (He is now a professor at N.Y.U. Stern.) Pankaj went to the same school as Sanjay and had a reputation as a Renaissance man. "I kind of lived in the shadow of my brother," Sanjay said. As an adult, he retains a talent for self-effacement. In 2016, when he was inducted into the American Academy of Arts and Sciences, he didn't tell his parents; their neighbor had to give them the news.

In graduate school, at M.I.T., Sanjay found a tight-knit group of friends. Still, he never dated, and does so only "very, very infrequently" now. He says that he didn't decide not to have a family—it just unfolded that way. His close friends have learned not to bother him about it, and his parents long ago accepted that their son would be a bachelor. Perhaps because he's so private, an air of mystery surrounds him at Google. He is known for being quiet but profound—someone who thinks deeply and with unusual clarity. On his desk, he keeps a stack of Mead composition notebooks going back nearly twenty years, filled with tidy lists and diagrams. He writes in pen and in cursive. He rarely references an old notebook, but writes in order to think. At M.I.T., his graduate adviser was Barbara Liskov, an influential computer scientist who studied, among other things, the management of complex code bases. In her view, the best code is like a good piece of writing. It needs a carefully realized structure; every word should do work. Programming this way requires empathy with readers. It also means seeing code not just as a means to an end but as an artifact in itself. "The thing I think he is best at is designing systems," Craig Silverstein said. "If you're just looking at a file of code Sanjay wrote, it's beautiful in the way that a well-proportioned sculpture is beautiful."

At Google, Jeff is far better known. There are Jeff Dean memes, modelled on the ones about Chuck Norris. ("Chuck Norris counted to infinity . . . twice"; "Jeff Dean's résumé lists the things he hasn't done—it's shorter that way.") But, for those who know them both, Sanjay is an equal talent. "Jeff is great at coming up with wild new ideas and prototyping things," Wilson Hsieh, their longtime colleague, said. "Sanjay was the one who built things to last." In life, Jeff is more outgoing, Sanjay more introverted. In code, it's the reverse. Jeff's programming is dazzling—he can quickly outline startling ideas—but, because it's done quickly, in a spirit of discovery, it can leave readers behind. Sanjay's code is social.

"Some people," Silverstein said, "their code's too loose. One screen of code has very little information on it. You're always scrolling back and forth to figure out what's going on." Others write code that's too dense: "You look at it, you're, like, 'Ugh. I'm not looking forward to reading this.' Sanjay has somehow split the middle. You look at his code and you're, like, 'O.K., I can figure this out,' and, still, you get a lot on a single page." Silverstein continued, "Whenever I want to add new functionality to Sanjay's code, it seems like the hooks are already there. I feel like Salieri. I understand the greatness. I don't understand how it's done."

On a Monday morning this spring, Jeff and Sanjay stood in the kitchenette of Building 40, home to much of Google's artificial-intelligence division. Behind them, a whiteboard was filled with matrix algebra; a paper about unsupervised adversarial networks lay on a table. Jeff, wearing a faded T-shirt and jeans, looked like a reformed beach bum; Sanjay wore a sweater and gray pants. The bright windows revealed a stand of tall pines and, beyond it, a field. Wherever Jeff works at Google, espresso machines follow. On the kitchenette's counter, a three-foot-wide La Marzocco hummed. "We're running late," Sanjay said, over a coffee grinder. It was eight-thirty-two.

After cappuccinos, they walked to their computers. Jeff rolled a chair from his own desk, which was messy, to Sanjay's, which was spotless. He rested a foot on a filing cabinet, leaning back, while Sanjay surveyed the screen in front of them. There were four windows open: on the left, a Web browser and a terminal, for running analysis tools; on the right, two documents in the text editor Emacs, one a combination to-do

list and notebook, the other filled with colorful code. One of Sanjay's composition notebooks lay beside the computer.

MORE FROM THIS ISSUE

DECEMBER 10, 2018

LIFE AND LETTERS

Edward Gorey's Enigmatic World

By Joan Acocella

POP MUSIC

Earl Sweatshirt and Rap's Murky In-Between Generation

By Carrie Battan

HOMECOMING

A Homecoming Powwow for Native New Yorkers

By Elizabeth Barber

SHOUTS

Algorit

By Henr

"All right, what were we doing?" Sanjay asked.

"I think we were looking at code sizes of TensorFlow Lite," Jeff said.

This was a major new software project related to machine learning, and Jeff and Sanjay were worried that it was bloated; like book editors, they were looking for cuts. For this task, they'd built a new tool that itself needed to be optimized.

"So I was trying to figure out how slow it is," Sanjay said.

"It's pretty slow," Jeff said. He leaned forward, still relaxed.

"So that one was a hundred twenty kilobytes," Sanjay said, "and it was, like, eight seconds."

"A hundred twenty thousand stack calls," Jeff said, "not kilobytes."

"Well, kilobytes of text, yeah—about," Sanjay said.

"Oh, yeah, sorry," Jeff said.

"I don't quite know what threshold we should pick for a unit size," Sanjay said. "Half a meg?"

"Seems good," Jeff said. Sanjay began to type, and Jeff was drawn into the screen. "So you're just saying, if it's bigger than that we'll just sample . . ." He left the rest unsaid; Sanjay answered him in code.

When Sanjay drives, he puts his hands at ten and two and stares attentively ahead. He is the same way at the keyboard. With his feet spread shoulder-width apart, he looked as if he were working on his posture. His spindly fingers moved gently across the keys. A few younger programmers began to trickle in.

Soon they reached a minor milestone, and Sanjay typed a command to test their progress. Seeming worn out, he checked his e-mail while it ran. The test finished. He didn't notice.

"Hey," Jeff said. He snapped his fingers and pointed at the screen. Although in conversation he is given to dad jokes and puns, he can become opinionated, brusque, and disapproving when he sits at a computer with Sanjay. Sanjay takes this in stride. When he thinks Jeff is moving too fast, he'll lift his hands off the keyboard and spread his fingers, as if to say, "Stop." (In general, Jeff is the accelerator, Sanjay the brake.) This is as close as they get to an argument: in twenty years together, they can't remember raising their voices.

Sanjay scrolled, bringing a new section of code into view. "Like, all that can be made into a routine, couldn't it?" Jeff said.

"Mmm," Sanjay agreed.

Jeff cracked his knuckles. "Seems doable. Should we do that?"

Sanjay was wary. "No, I—"

"So we're going to ignore a problem?" Jeff said indignantly.

"No, I mean, we're just trying to get an idea of the types of things that are going on. So we could make notes about it, right?"

"O.K.," Jeff said happily, his mood having turned on a dime. They dictated a note together.

Lunchtime approached. They had worked for two hours with one ten-minute break, talking most of the time. (A lesser programmer watching them would have been impressed, more than anything else, by the fact that they never stopped or got stuck.) It's standard engineering practice to have your code reviewed by another coder, but Jeff and Sanjay skip this step, entering, in their log, a perfunctory "lgtm," for "looks good to me." In a sense, they had been occupied by minutiae. Their code, however, is executed at Google's scale. The kilobits and microseconds they worry over are multiplied as much as a billionfold in data centers around the world—loud, hot, warehouse-size buildings whose unending racks of processors are cooled by vats of water. On days like these, Jeff has been known to come home and tell his daughters, "Sanjay and I sped up Google Search by ten per cent today."

Jeff and Sanjay gave Google what was arguably its biggest single upgrade in the course of four months in 2003. They did it with a piece of software called MapReduce. The idea came to them the third time they rewrote Google's crawler and indexer. It occurred to them that, each time, they had solved an important problem: coördinating work in a vast number of geographically distributed, individually unreliable computers. Generalizing their solution would mean that they could avoid revisiting the problem again and again. But it would also create a tool that any programmer at Google could use to wield the machines in its data centers as if they were a single, planet-size computer.

MapReduce, which Jeff and Sanjay wrote in a corner office overlooking a duck pond, imposed order on a process that could be mind-bendingly complicated. Before MapReduce, each programmer had to figure out how to divide and distribute data, assign work, and account for hardware failures on her own. MapReduce gave coders a structured way of thinking about these problems. Just as a chef maintains *mise en place*—prepping ingredients before combining them—so MapReduce asks programmers to divide their tasks into two stages. First, a coder tells each machine how to do the "map" stage of a task (say, counting how many times a word appears on a Web page); next, she writes instructions for "reducing" all the machines' results (for instance, by adding

them up). MapReduce handles the details of distribution—and, by doing so, hides them.

The following year, Jeff and Sanjay rewrote Google's crawling and indexing system in terms of MapReduce tasks. Soon, when other engineers realized how powerful it was, they started using MapReduce to process videos and render the tiles on Google Maps. MapReduce was so simple that new tasks kept suggesting themselves. Google has what's known as a "diurnal usage curve"—there's more traffic during the day than there is at night—and MapReduce tasks began soaking up the idle capacity. A dreaming brain processes its daytime experiences. Now Google processed its data.

There were inklings, early on, that Google was an A.I. company pretending to be a search company. In 2001, Noam Shazeer, who shared an office with Jeff and Sanjay, had grown frustrated with the spell-checker that Google was licensing from another company: it kept making embarrassing mistakes, such as telling users who'd typed "TurboTax" that they probably meant "turbot ax." (A turbot is a flatfish that lives in the North Atlantic.) A spell-checker is only as good as its dictionary, and Shazeer realized that, in the Web, Google had access to the biggest dictionary there had ever been. He wrote a program that used the statistical properties of text on the Web to determine which words were likely misspellings. The software learned that "pritany spears" and "brinsley spears" both meant "Britney Spears." When Shazeer demonstrated the program at Google's weekly T.G.I.F. gathering, employees tried, but mostly failed, to fool it. In collaboration with Jeff and an engineer named Georges Harik, Shazeer applied similar techniques to associate ads with Web pages. Ad targeting became a river of money that the company directed back into its computing infrastructure. It was the beginning of a feedback loop—bigness would be the source of Google's intelligence; intelligence the source of its wealth; and wealth the source of its growth—that would make the company extraordinarily, and unsettlingly, dominant.

As enterprising coders used MapReduce to derive insights from Google's data, it became possible to transcribe users' voice mails, answer their questions, autocomplete their queries, and translate among more than a hundred languages. Such systems were developed using relatively uncomplicated machine-learning algorithms. Still, "very simple techniques, when you have a lot of data, work incredibly well," Jeff said. As "data, data, data"—stored and processed with BigTable, MapReduce, and their

successors—became the company's prime directive, Google's globe-spanning infrastructure became more seamless and supple. The idea of distributed computation was an old one; concepts like "cloud computing" and "big data" predated Google's rise. But, by making it intellectually manageable for ordinary coders to write distributed programs, Jeff and Sanjay had given Google a new level of mastery over such technologies. Users may have sensed that something had changed: Google's cloud was getting smarter.

In 2004, because Jeff and Sanjay thought it would be useful to astronomers, geneticists, and other scientists with lots of data to process, they wrote a paper, "MapReduce: Simplified Data Processing on Large Clusters," and made it public. The MapReduce paper arrived as a deus ex machina. Cheap hardware and the growth of Web services and connected devices had led to a deluge of data, but few companies had the software to process the information. Two engineers who'd been struggling to scale a small search engine called Nutch—Mike Cafarella and Doug Cutting—were so convinced of MapReduce's importance that they decided to build a free clone of the system from scratch. They eventually called their project Hadoop, after a stuffed elephant beloved by Cutting's son. As Hadoop matured, it was adopted by half of the Fortune 50. It became synonymous with "Big Data." Facebook used "Hadoop MapReduce," as it's often known, to store and process user metadata—information about what was clicked, what was Liked, and which ads were viewed. At one point, it had the largest Hadoop cluster in the world. Hadoop MapReduce helped power LinkedIn and Netflix. Randy Garrett, a former director of technology at the National Security Agency, remembers demonstrating the technology to the agency's director, General Keith Alexander. Hadoop performed an analysis task eighteen thousand times faster than the previous system had. It became the foundation for a new approach to intelligence gathering which some observers call "collect it all."

Jeff has a restless nature: a problem becomes less interesting to him once he can see the shape of its solution. In 2011, as the world embraced the cloud, he began collaborating with Andrew Ng, a computer-science professor from Stanford who was leading a secretive project at Google to conduct research on neural networks—software programs composed of virtual "neurons." Jeff had encountered neural nets during his undergraduate days; back then, they hadn't been able to solve real-world problems. Ng had told Jeff that this was changing. At Stanford, researchers had achieved some

exciting results when the nets were given access to large quantities of data. With Google's scale, Ng thought, neural networks could become not just useful but powerful.

Neural networks are profoundly different from traditional computer programs. Their behavior isn't specified by coders in the usual way; instead, it's "learned" using inputs and feedback. Jeff's knowledge of neural networks hadn't advanced much since his undergrad years, and Heidi watched as their bathroom filled with textbooks. Jeff began spending about a day a week on the project, which was called "Google Brain." Many at Google were doubtful of the technology. "What a waste of talent," Alan Eustace, his manager at the time, remembers thinking. Sanjay couldn't understand Jeff's move, either. "You work on infrastructure," he thought. "What are you doing over there?"

During the next seven years, the Google Brain team developed neural nets that beat the state of the art in machine translation and speech and image recognition. Eventually, they replaced Google's most important algorithms for ranking search results and targeting ads, and Google Brain became one of the fastest-growing teams in the company. Claire Cui, an engineer who started in 2001, said that Jeff's involvement marked a turning point for A.I. at Google: "There were people who believed in it, and there were people who didn't believe in it. Jeff proved that it can work."

A.I. had turned out to depend on scale, which Jeff, the systems engineer, delivered. As part of the effort, he led the development of a program called TensorFlow—an attempt to create something like the MapReduce of A.I. TensorFlow simplified the task of distributing a neural network across a fleet of computers, turning them into one big brain. In 2015, when TensorFlow was released to the public, it became the lingua franca of A.I. Recently, Sundar Pichai, Google's C.E.O., declared it an "A.I. first" company and made Jeff the head of its A.I. initiatives.

Jeff now spends four days a week running Google Brain. He directs the work of three thousand people. He travels to give talks, holds a weekly meeting to work on a new computer chip (the Tensor Processing Unit, designed specifically for neural networks), and is helping with the development of AutoML, a system that uses neural nets to design other neural nets. He has time to code with Sanjay only once a week.

eats of engineering tend to erase themselves. We remember the great explorers of the eighteenth century—James Cook, George Vancouver—but not John Harrison, the carpenter from Yorkshire who, after decades of work, made a clock reliable enough to reckon longitude at sea. Recently, Jeff and Sanjay were enjoying margaritas and enchiladas at Palo Alto Sol, a Mexican restaurant they frequent, when Jeff pulled out his phone. "When did Gmail first come out?" he asked. The phone replied, "April 1st, 2004." Sanjay, who is sensitive in social situations, seemed not to appreciate the dinner-table distraction, but Jeff was elated. Google could now talk, listen, and answer questions, using a stack of programs, seamlessly integrated and largely invisible, stretching from his phone to data centers around the world.

Today, their roles have diverged. At Google, Sanjay is what is known as an "individual contributor"—a coder who works alone and manages no one. For this, he feels grateful. "I would not want Jeff's job," he says. He's currently working on software that makes it easier for engineers to combine and control the dozens of programs—for fetching news, photographs, prices—that start running as soon as a user enters text into Google's search box. Once a week, he meets with a group of "Area Tech Leads"—Google's engineering Jedi council—to make technical decisions that affect the entire company. If Google were a house, Jeff would be building an addition. Sanjay is shoring up the structure, reinforcing the beams, tightening the bolts.

Meanwhile, during their Monday coding sessions, they have started something new. It's an A.I. project: an attempt, Jeff says, to train a "giant" machine-learning model to do thousands, or millions, of different tasks. Jeff has been thinking about the idea for years; recently, he decided that it was possible. He and Sanjay plan to build a prototype that a team can grow around. In the world of software, the best way to lead is with code.

"I think they miss each other," Heidi, Jeff's wife, says. It was when their collaboration slowed that they began having their Friday dinners.

On a Sunday in March, Jeff and Sanjay met for a hike outside Cupertino. The weather was clear and brisk, but hot in the sun. Jeff arrived at the trailhead in a blue Tesla Roadster with a Bernie 2016 bumper sticker. Sanjay, close behind, had his own Tesla, a red Model S. Sanjay had spent the morning reading. Jeff had played soccer. (A device attached to his calf told him that he'd run 7.1 miles.) Two decades after building the

March index, Jeff resembled a retired endurance athlete, his skin worn by the sun. Sanjay seemed hardly to have aged.

The path was a six-mile loop that climbed through dense forests. Jeff led the way. In the woods, they reminisced about how quickly Google had grown. Sanjay recalled how, during the company's first growth spurt, a plumber had installed two toilets in a single stall in the men's bathroom. "I remember Jeff's comment," he said. " 'Two heads are better than one!' " He laughed.

They descended out of the woods and into dry, exposed country. A turkey vulture flew overhead.

"The hills here are steeper than I thought," Jeff said.

"I thought somebody said this was a pretty flat hike," Sanjay said.

"I guess this explains why there's no biking roads up that side," Jeff said.

They climbed back into the woods. On a switchback, Jeff caught a glimpse beyond the trees. "We're gonna have a good lookout at some point," he said.

The trail opened out onto a hilltop, high and wide, treeless, with panoramic views. There was a haze on the horizon. Still, they could see the Santa Cruz Mountains to the south and Mission Peak to the east. "Sanjay, there's your office!" Jeff said. They stood together, looking across the valley. ♦

This article appears in the print edition of the December 10, 2018, issue, with the headline "Binary Stars."

James Somers is a writer and a programmer based in New York. Read more »