4

YOUNG MAN IN A HURRY 1966–70

Naur's vision of datalogy – Architect of the RC 4000 computer – Programming a real-time system – Working with Henning Isaksson, Peter Kraft, and Charles Simonyi – Edsger Dijkstra's influence – Head of software development – Risking my future at Hotel Marina – The RC 4000 multiprogramming system – I meet Edsger Dijkstra, Niklaus Wirth, and Tony Hoare – The genius of Niels Ivar Bech.

I was fortunate to start my programming career at Regnecentralen. For almost three years, I had participated in the design, programming, testing, and documentation of a large compiler. I knew it was time to leave the compiler group and try something else. Niels Ivar Bech had something in mind—but I had other ideas.

In a brilliant paper, Peter Naur (1966a) viewed compilation as a general data processing problem that involves more fundamental programming methods, which he felt should be taught as part of a core of computer science. At a time, when compiler contruction was still regarded as a fundamental subject in its own right, Naur's insight was ahead of its time.

In 1966, Bech invited me to join a working group consisting of Peter Naur, Christian Gram, Henning Bernhard Hansen, Jens Hald, and Alan Wessel. Their goal was to develop a systematic text on datalogy (as Naur called it). This was an exciting idea—but it was not mine. My answer to Bech was honest: "Thank you, but I prefer to wait until I am writing my own book."

Naur (1968) proceeded to outline a complete core course on computer science based on fundamental principles. His vision of computer science was published in the same year as Donald Knuth's famous Volume 1 of *The Art of Computer Programming* (1968).

For various reasons, the working group never finished its ambitious project (although parts of it was published in Danish). A short English version of the complete text was published in 1974, with Peter Naur as the only author.

⋆ ⋆ ⋆

After my three-year apprenticeship at Regnecentralen, Milena and I were talking about living abroad for a while. After my graduation, IBM had encouraged me to keep in touch, in case I would be interested, after some years, in working at one of their labs in Sweden, England, or the United States. That sounded promising after my enjoyable experience at the IBM Hursley Laboratory in the summer of 1961.

Henning Isaksson had asked Niels Ivar Bech for a systems programmer for quite some time. Since I was thinking of leaving anyhow, Bech suggested that I might join Isaksson's hardware group in Valby.

In the Polish city of Pulawy, the Danish engineering company, Haldor Topsøe, was designing the largest fertilizer plant in Europe. The company signed a contract with Regnecentralen to deliver a small computer with data logging software. The system was supposed to demonstrate that the plant satisfied the specifications guaranteed by Topsøe, and would also help management with maintenance of the plant.

From the beginning, Henning and I viewed the Pulawy-project as an opportunity to develop Regnecentralen's third computer. However, Bech did not see it that way. He strongly encouraged Henning to use the recent CDC 1700 computer. Bech was not known for cautious decisions. On this occasion, he may have been influenced by Regnecentralen's reorganization in 1964 as a limited company with shareholders.

Henning finally said: "Look, if we use CDC software in our process control programs, our customers will expect us to help them with the problems of software, that we have not developed." This argument persuaded Bech that we would be better off developing our own process control computer. He remembered only too well how Regnecentralen had been forced to use Fortran on its large CDC 1604 computer, because the Algol compiler from Control Data turned out to be unreliable (Isaksson 1976). In 1978, I gave the same answer, when the chairman of Mostek Corporation asked me, if I thought it would be a good idea to put an IBM mainframe computer on a chip.

Now, if something has a name, it obviously must exist. My favorite example is the medical term "essential hypertension." With a name like that, who could doubt that medical doctors know, what they are talking about. On the contrary, if doctors don't have a clue about the cause of hypertension, they call it "essential." So, although Gier's successor was still on the drawing board, we needed a name for it. The natural choice would have been calling it the RC 3. However, Regnecentralen was already marketing the RC 3000, a special-purpose device for data conversion. So I suggested calling the new computer the RC 4000, since "who would buy an RC 3 for a million kroner, when you can buy an RC 3000 for a lot less?" So RC 4000 it was.

⋆ ⋆ ⋆

Henning was an efficient manager and very pleasant to work for. At my request, he persuaded Bech to let Peter Kraft join us from the compiler group. Peter was an experienced programmer who had learned his craft during the Gier Algol project. Of average height, with a receding hairline and large, dark-rimmed glasses, his face was usually lit up by a brilliant smile. We worked well together, perhaps because he provided a calming influence on my own forceful personality. The hardware engineer, Villy Toft, wrote this portrait of Peter:

> Cheerful, humourous and positive, he had a constructive approach to problem solving, unhampered by potential problems. A gifted problem solver with an analytical talent, who showed no signs of stress, he worked quietly and calmly with other members of the group as a catalyst and inspirator. His modesty gave him a tendency to downplay his own achievements, even though others valued them highly.

Most of us should be so lucky to be remembered like that!

When it became obvious that Peter and I would need another programmer, Bech sent us a Hungarian teenager! At age 14, Charles Simonyi wrote his first program for a huge Russian Ural II computer in Budapest, where his father was professor of electrical engineering. He desperately wanted to leave his communist country and emigrate to the United States. During a demonstration of the Gier in Budapest, Charles met Bech, who offered him a one-year job in Denmark. After completing high school, he left Hungary in the summer of 1966 and came to Denmark without an education. He was 17 years old, had a Beatles haircut and spoke limited English with a Hungarian accent. When he heard a Caravelle jet outside, he would open the window, stick his head out, and shout: "Vonderful Caravelle." At the end of the Pulawy project, Charles had saved enough money to go to California. At Berkeley, he paid for tuition by working as a programmer in the university computing center. After graduating in 1972, he joined Xerox Palo Alto Research Center, where his former professor, Butler Lampson, and others were pioneering personal computing on Alto personal computers with graphic interfaces, mice, laser printers, and ethernets. For the Alto, Butler and Charles designed Bravo, the first graphic text editor. When Simonyi joined Microsoft in 1981, he brought his knowledge of Bravo and turned it into Microsoft Word. Today, Charles is a member of the U.S. National Academy of Engineering. He is also very rich.

⋆ ⋆ ⋆

Before Simonyi joined us, Peter Kraft and I had already defined the architecture of the RC 4000, and the prototype for Pulawy was being assembled in a small lab right below our office.

With 1K words of memory only (about 5K bytes), the Gier computer had been designed for clever handcoding of compact machine code. Experienced programmers had been known to stare for days at small fragments of the Gier Algol compiler, trying to figure out what Jørn Jensen's code was doing.

However, by 1965, it seemed safe to predict that, by the end of the decade, most programs would be written in high-level languages for computers with large memories. Most machine code would then be generated by compilers (and not by programmers).

When programmers write compact code, they take advantage of all kinds of programming tricks. It is not so easy to write a compiler that does the same kind of optimization. From a compiler writer's point of view, the ideal computer should have a systematic instruction set that makes code generation straightforward (but not necessarily optimal). Faster processors and larger memories would soon make this approach practical.

In defining the instruction set of the RC 4000, our goal was to simplify program compilation (instead of hand-coding). Whereas most computers had several instruction formats, the RC 4000 would have only one. This meant that any instruction could use the full set of addressing modes.

Every instruction defined an operation between a memory location and a register. However, by addressing the registers as the first four words of memory, you could operate on any pair of registers. It was even possible to jump to a register and execute its value as an instruction. This feature was used to autoload an initial program into an empty memory.

The basic addressing modes were extended by an instruction, called Modify Next Address, which used its own address to increment the address part of the next instruction. (The operation changed only the effective address of the next instruction but left its displacement field unchanged.) This instruction made it possible to use any memory location as an index register. A sequence of these instructions could modify an instruction with the sum of several registers or perform multiple-levels of indirect adressing.

Since the hardware and software engineers lived in different worlds, I faced the problem of describing the architecture in a formal language that made sense to both groups. Although pictures with informal explanations were helpful, they could not always convey the finer details accurately.

I settled the issue by using Algol 60 as a hardware definition language. Before the computer was built, I wrote a reference manual that defined the instruction set completely by an Algol program. This program simulated the execution of RC 4000 machine code using simple and indexed variables to represent hardware registers and memory locations. It defined how operands were addressed in memory, and how arithmetic results were computed, bit by bit, with overflow detection. It also defined the instruction fetch cycle, the memory protection system, the interrupt system, and the function of the power-on and reset keys.

Peter Kraft still remembers that if we discussed some aspect of the architecture, which at first looked like a detail only, I would often go home and work throughout the night, revising and rewriting the description of the architecture one more time.

Inspired by my use of Algol 60 as a hardware definition language, one of Isaksson's engineers, Allan Giese, extended Algol and used it to describe the internal structure of the RC 4000 (the microprogram).

At some point, Niels Ivar Bech called a meeting with Christian Gram, Jørn Jensen, Peter Naur, Bjarner Svejgaard, Henning Isaksson and me to discuss the proposed RC 4000 architecture. This was a valuable opportunity to benefit from the comments of Regnecentralen's senior people. It was also a "final exam" I had to pass, before Isaksson would get the green light to build the machine.

In preparation for the meeting, I distributed a detailed draft of the RC 4000 reference manual. At the meeting, the architecture was accepted without much discussion. I may have learned something from Naur's performance in the Algol 60 committee. It may also have helped that I had seventeen years of experience in writing essays.

This was a group of people who (quite correctly) insisted on concise writing. I still chuckle when I remember what Christian Gram said at that meeting, almost forty years ago. In my draft, I wrote, "A special autoload instruction is used for initial program loading." Christian's response was: "All instructions are special." Bingo!

In April 1967, Regnecentralen published the first official edition of my *RC 4000 Computer Reference Manual*. Two years later, Christian Gram extended the manual with complete definitions of floating-point arithmetic (Brinch Hansen 1969b). At that point, it was no doubt the only reference manual in the world that made it possible for programmers to predict the result, bit by bit, of dividing two non-normalized floating-point numbers!

⋆ ⋆ ⋆

I don't mean to drag you through the details of fertilizer production and real-time programming. But I would like you to understand the gist of what we did, since this was my first encounter with a major revolution in computer programming that became the focus of my professional work for thirty years: *concurrent* or *parallel programming*—the art of making a computer execute several programs at the same time.

The three units of the Pulawy plant produced ammonia, nitric acid, and ammonium nitrate. The plant was operated manually under supervision of the RC 4000 prototype, which had a core memory of 4K words (about 12K bytes), but no drum or disk.

John Saietz, a chemical engineer from Haldor Topsøe, worked with Peter Kraft to specify the process control tasks:

The RC 4000 would count the production of fertilizer and the consumption of electricity by sampling digital signals from bag-filling devices and kilowatt-hour meters every second. Every five minutes, the computer would input about 350 analog measurements of pressures, temperatures, and material flow rates, checking that they remained within certain alarm limits. Every hour, two snapshots of the plant operation would be printed, listing some 550 analog measurements. When one shift of workers was replaced by another, the machine would print a report of material balances showing the production and energy consumption over the past eight hours.

Topsøe also wanted the operators to be able to make the computer perform some tasks more frequently, when units of the plant were being restarted after repair.

It was now up to us to translate the chemical engineer's specification into real-time control software.

Ignoring for the moment the engineering details of fertilizer production, a programmer might summarize the project as follows: a small computer has to perform a fixed number of cyclical tasks with frequencies determined by plant operators. These tasks must be able to share data tables and input/output devices (including an analog/digital converter and various printing devices).

It seemed natural to write a separate program for each control task. However, we could not expect to fulfill the real-time requirements by executing one task program at a time: two task programs might need to be started at the same time, and the time required for a single execution of a task program might also be longer than the time interval between successive executions of other task programs.

Ideally, we would have liked to be able to run task programs in parallel. However, since the computer could only execute one instruction at a time, we had to settle for a pseudo-parallel mode of execution, known as *multiprogramming.*

Four hundred times a second, an electronic timer interrupted the running task program and transfered control to a scheduling program, known as the *monitor.* The monitor then resumed the execution of another task program for 2.5 msec, and so on. In this way, the computer was shared cyclically among the active tasks.

The use of clock interrupts to simulate concurrent execution of programs was pionered on the Atlas computer by Tom Kilburn and David Howarth (1961). Multiprogramming is still the principle behind time-sharing operating systems (such as Unix or Windows).

Switching a single computer among multiple tasks is similar to a waiter (the computer) serving several tables (the tasks). As long as the waiter only spends a fraction of his time at each table, most of the customers will be able to eat at the same time.

In the Pulawy system, each task used only a few percent of the computer time. The rest of the time, the tasks would wait for slow peripheral devices. As soon as a task started waiting for the completion of input/output, the monitor switched to another task. So although the tasks never executed instructions simultaneously, the typewriters would nevertheless print at the same time.

In our real-time system, we needed some form of *synchronization* to prevent several tasks from using the same printer (or data table) at the same time. But what kind of synchronization?

The early multiprogramming systems were programmed in assembly language without any conceptual foundation. The slightest programming mistake could make these systems behave in a completely erratic manner that made program testing nearly impossible.

A common synchronization technique at the time was to suspend a task in a queue until it was resumed by another task. The trouble was that resumption had no effect if the queue was empty. This happened if resumption was attempted *before* a task was suspended. (This pitfall reminds me of a mailman who throws away your letters if you are not at home when he attempts to deliver them!)

This mechanism was unreliable because it made a seemingly innocent assumption about the relative timing of parallel events: A task must *never* attempt to resume another task that is not suspended.

Since the Pulawy operators could change the frequencies of individual tasks (and even stop some of them indefinitely), we could not make any assumptions about the relative (or absolute) speeds of the tasks. Time-dependent event queues would have been a disastrous choice for our real-time system. Around 1965 IBM's PL/I language included event queues of this kind. Surprisingly, the suspend and resume primitives are also included in the recent Java language.

Regnecentralen had no experience with multiprogramming. Fortunately, Edsger Dijkstra was kind enough to send me a copy of his 1965 monograph *Cooperating Sequential Processes*, with a personal dedication: "Especially made for graceful reading!" (I still have it.) One of the great works in computer programming, this masterpiece laid the conceptual foundation for concurrent programming.

It began by making the crucial assumption about *speed independence*:

> We have stipulated that processes should be connected loosely; by this we mean that apart from the (rare) moments of explicit intercommunication, the individual processes themselves are to be regarded as completely independent of each other. In particular, we disallow any assumption about the relative speeds of the different processes. [In Dijkstra's terminology, individual tasks were known as *processes.*]

Dijkstra proceeded to illustrate how concurrent processes can synchronize themselves correctly by sending timing signals through *semaphore variables* (as he called them).

Using semaphores, I was able to program the *RC 4000 real-time monitor* in only 400 words of memory, leaving 3,700 words for the data logging tasks that would be implemented by Peter Kraft and Charles Simonyi.

To reduce the size of the task programs, Peter and Charles defined a special-purpose computer that made it easy to write compact code for the complicated engineering computations. The code for this computer was executed by a small interpreter written in RC 4000 machine code. Such a machine, implemented in software (rather than hardware), is known as an "abstract machine." (In the 1980s, interpreted code would also be used in the first versions of the Microsoft Word and Excel programs for the Macintosh computers.)

Interpreted code has also been a marvelous tool for language implementation. The idea of letting a compiler generate code for an abstract machine tailored to a programming language goes back (at least) to LISP in the early 1960s. A major advantage of interpreted code is that it is "portable"— it can run on any computer, if you reprogram the interpreter in the machine code of the target machine. In the 1970s, abstract machines would be used to implement portable compilers for Pascal and Concurrent Pascal. Twenty years later, the same technique would be used to make the Java language "platform-independent."

Now back to the RC 4000 real-time system. The final system occupied 98 percent of the 4K word memory. How was it possible to predict the size of the software that precisely? Well, John Saietz simply gave us a list of desirable features that could be omitted, if necessary. We just kept adding more of these until the memory was full.

At the NordSAM conference in Oslo, Norway, on June 13, 1967, Peter Kraft presented two papers, written by me, on the RC 4000 computer and its real-time control system (Brinch Hansen 1967a, 1967b). According to *Electronic News*:

> Some 400 representatives from the Scandinavian countries participated in NordSAM 67...The RC 4000 system developed by A/S Regnecentralen of Copenhagen, a general-purpose computer especially suited for real-time control, was held up as perhaps the most promising development for international markets. (July 10, 1967)

In the same month, Regnecentralen began installing the RC 4000 prototype and its real-time system in Poland. The communist government was responsible for construction of the plant. Unfortunately, under Wladislaw Gomulka's regime, efficiency was not a top priority. When Villy Toft and Peter Kraft arrived in Pulawy, the computer room still left something to be desired: it had neither doors nor electricity. It required patience and diplomacy to get the room finished. However, a week later, these problems had been fixed, and the installation could begin. The final operational tests of the plant and its real-time datalogging system took place in the spring of 1968.

Later, I heard that the Polish government had not built enough railroad capacity in Pulaway, and was unable to ship the fertilizer as fast as it was being produced to prevent it from piling up. While I cannot confirm this story, I do know Villy had to deal with mice and rats, who ate the backplane wiring and relieved themselves on the circuit boards.

Strangely enough, I never visited the chemical plant in Poland, probably because I didn't like flying. I did travel once by train from Copenhagen to a meeting in Warsaw, enjoying warm tea from a samovar in the rear of the train. Returning through East Germany, I saw armed border guards using huge mirrors to look under the train for refugees trying to escape the communist regime.

⋆ ⋆ ⋆

In June 1967, I returned from Regnecentralen's hardware group in Valby to the Rialto Center as head of RC 4000 software development. From Aage Melbye's group, Peter Kraft and I were joined by Christian Gram, who would define floating-point arithmetic and numerical procedures, and by Søren Lauesen, who would be responsible for developing an Algol compiler.

I was now officially in charge of more projects than I could hope to participate in. However, it still seemed important to me to continue doing, what I enjoyed most—designing and documenting systems programs. I therefore decided to head an operating system group consisting of Jørn Jensen, Peter Kraft, Søren, and me.

In Pulawy, we had tailored a small real-time system to the specific needs of a single customer. We were now hoping to develop a more general monitor program for the RC 4000. It remained to be seen what kind of generality we were looking for.

First we proposed a system with fixed memory partitions for simultaneous execution of three programs, with runtimes of the order of seconds, minutes, and hours, respectively. But, we soon realized that this ad hoc system was not a "general" solution to anything.

Finally, I went to Bech and said: "Look, we aren't getting anywhere. Is it all right with you if Jørn, Søren, Peter, and I stay at a country inn for a weekend?" Bech immediately agreed (he had done the same thing when Regnecentralen's Cobol project had come to a standstill).

I wanted us to discuss the software issues in depth in cozy surroundings to give ourselves one last chance. We had already agreed that we would either return with new ideas or give up and settle for copying the best ideas we could find elsewhere.

How on earth did I have the nerve, at age 29, to gamble my career on a single weekend? Never underestimate the power of the dreams of youth (and its blissful ignorance of "the real world")!

Anyhow, on October 28, 1967, we checked in at Hotel Marina by the seashore, north of Copenhagen. We talked for two days, drank coffee with French cognac, and enjoyed fine dinners. One evening, Jørn and I saw a black-and-white western at a local movie theater—just what we needed to relax after intense discussions.

And it worked! The thought of returning to Regnecentralen without new ideas was simply unacceptable to us. Out of that weekend came the first seminal ideas for the *RC 4000 multiprogramming system.*

Over the next four months, our discussions moved beyond known concepts to the cutting edge of operating system design. I will not attempt to describe how our ideas slowly emerged in daily discussions (I remember some, but not all, of it). Instead I will explain how one thing led to another until everything fit together nicely. Just keep in mind that our conclusions did not emerge nearly as orderly as I present them here.

Regnecentralen was already using computers for software development and business data processing. The Pulawy project added real-time applications to this mix. Unfortunately, real-time software is often unique for each application. And, Regnecentralen simply did not have enough system programmers to develop a different operating system for each RC 4000 installation.

To avoid that trap, we had to look at operating systems in a new way. Computer manufacturers were still developing different operating systems for batch processing, time-sharing, and real-time scheduling. Our hope was to develop a monitor program that would provide the necessary mechanisms to implement *all forms* of multiprogramming.

Regnecentralen's main service center updated large files on magnetic tapes. Early on, we decided to simplify the operator's task in such an installation. Instead of using device numbers, programs would refer to tapes by device-independent *names*. This convention would allow an operator to mount a tape on any available unit and give it a temporary name.

From Dijkstra, we had learned to regard the execution of a program as a sequential process. Now, when you think about it, a sequence of input/output operations on a magnetic tape is also a sequential process. This insight led us to regard program execution and input/output as different kinds of processes—we called them *internal* and *external processes*.

Since we had already decided to assign names to peripheral devices, it was more or less inevitable that we would also end up giving names to internal processes. The beauty of this idea was that programs could refer to processes by their names without knowing where in memory they were located.

We still had to deal with the problem of process synchronization. Semaphores were not robust enough for our purposes. If a program used semaphores incorrectly, it could crash any operating system. Instead, the monitor would implement a reliable form of *message passing*. When one process sent a message to another process, the message was copied inside the monitor and linked to a message queue associated with the receiver. The memory protection guaranteed that the message would remain intact until it had been safely delivered to the receiver.

So we now had named processes communicating by messages. You didn't have to be a genius to suggest that it would be a neat idea to let internal processes request input/output by sending messages to external processes. The monitor would, of course, have to maintain a message queue for each device.

However, since an input/output operation could fail, it would be necessary to return an acknowledgment to the process that sent a command to a device. The way we handled this problem was elegant (but not so obvious). In the end, every communication with a peripheral device consisted of an exchange of a message and an answer between an internal process and an external process. Devices received input/output commands as messages and returned acknowledgments as answers. Needless to say, we soon decided to let internal processes communicate the same way.

Every communication could now be viewed as a procedure call from one process to another: a message identified the procedure and supplied its input parameters; the corresponding answer returned the results of the procedure call. In distributed systems, this form of communication is now known as *remote procedure calls.*

Our final idea was to let internal processes form a tree structure in memory. In this tree, every process would function as the operating system for its children, who, in turn, would control their own children, and so forth. The leaves of the tree would be user processes. The idea of running several operating systems at the same time is, of course, beautiful, but who needs it? Over the years, I have learned not to worry about such questions. If an idea is elegant, you will, sooner or later, find an (unexpected) use for it. The generality of a process tree does, for example, provide an orderly way of switching between different operating systems. It also enables you to test a new operating system on top of an old one, which is a lot easier than developing it on an empty machine.

In February 1968, before programming the system, I described our design philosophy, which drastically generalized the concept of an operating system:

> The system has no built-in assumptions about program scheduling and resource allocation; it allows any program to initiate other programs in a hierarchal manner. Thus, the system provides a general frame[work] for different scheduling strategies, such as batch processing, multiple console conversation, real-time scheduling, etc. [Here I obviously meant "processes" rather than "programs."]

The RC 4000 multiprogramming system was not a complete operating system, but a small *kernel* upon which operating systems for different purposes could be built in an orderly manner The kernel provided the basic mechanisms for creating a tree of parallel processes that communicated by messages.

This radical idea was probably the most important contribution of the RC 4000 system to operating system technology. If the kernel concept seems obvious today, it is only because it has passed into the general stock of knowledge about system design. According to the IEEE Computer Society (2002):

> The RC 4000 multiprogramming system introduced the now-standard concept of an operating system kernel and the separation of policy and mechanism in operating system design. The

microkernels and remote procedure calls used in modern operating systems can trace their roots back to the RC 4000 system.

A well-documented reliable version of the RC 4000 multiprogramming system was running in the spring of 1969. At that point, I described it in a 5-page journal paper.[1] I then used this paper as an outline of the 160-page system manual by expanding each section of the paper.[2]

⋆ ⋆ ⋆

Regnecentralen built several operating systems on top of the RC 4000 kernel. Some of them used dynamic swapping of processes between main memory and backing storage. As usual, the kernel only provided a mechanism for doing this, but left the policy of how and when it was used to an operating system. The latter would ask the kernel to stop a running process and its descendants temporarily. The operating system would then output the memory image of the process to a backing store, and use the same memory segment to reload the image of another process, that had been stopped earlier. The operating system would then ask the kernel to restart that process and its descendants.

In the early 1970s, Regnecentralen developed RC 4000 software for two Danish power plants, Vestkraft and Nordkraft. Villy Toft was again the system installation manager. Working with Niels Nedergaard from Vestkraft, Peter Kraft and Otto Vinter designed a process control system that combined real-time tasks with swapping of Algol programs running as background jobs. Later Regnecentralen's Einar Mossin joined forces with Peter and Niels in designing and programming a real-time system for Nordkraft. One of the challenges here was to record the avalance of alarms, that occurs when the plant is close to a breakdown.

I hired a student, Leif Svalgaard, who became so absorbed in programming the RC 4000 for a Danish Weather Bureau, that he forgot to take his final exam. Leif wrote an operating system that received data and plotted weather maps in real-time. This operating system coexisted with another one that used swapping to perform scientific computations in parallel. According to Leif: "The RC 4000 kernel made all this safe, efficient, and easy,

[1]P. Brinch Hansen, The nucleus of a multiprogramming system, *Communications of the ACM 13*, April 1970.

[2]P. Brinch Hansen, *RC 4000 Computer Software: Multiprogramming System*, Regnecentralen, Copenhagen, Denmark, April 1969.

allowing us to concentrate on meteorological problems instead of fighting the operating system."

Søren Lauesen (1975) described an ambitious operating system designed to handle batch processing, remote job entry, time sharing, jobs generated internally by other jobs, as well as process control simultaneously. It used over a hundred parallel activities, one for every peripheral device and job process. Since the RC 4000 multiprogramming system was limited to 23 concurrent processes, the "Boss 2" system (as it was called) simulated another level of multiprogramming inside a single RC 4000 process. The additional processes were known as "coroutines" (a programming concept that goes back to the early 1960s). Using induction, Søren proved that his system was deadlock-free and guaranteed to complete any request for service. It was implemented and tested by four to six people over a period of two years:

> When we started the Boss 2 design, we knew that the RC 4000 software was extremely reliable. In a university environment, the system typically ran under the simple [manual] operating system for three months without crashes...The crashes present were possibly due to transient hardware errors ...During the first year of operation, the [Boss 2] system typically ran for weeks without crashes. Today it seems to be error free.

⋆ ⋆ ⋆

My descriptions of the RC 4000 multiprogramming system caught the attention of leading computer scientists in Europe and America.

The accolade was a letter from Edsger Dijkstra, professor at the Technological University of Eindhoven in the Netherlands:

> I would like to express my gratitude and admiration for the manual of the multiprogramming system for the RC 4000. It is admirable! You wrote "We present our system as a systematic and practical solution..." and I have the feeling that you are fully right in doing so: it strikes me as a convincing demonstration that it is worthwhile to do a clean job and that it pays to be elegant. My appreciation is equally divided between what the manual describes and the way in which you have described it: it was a pleasure to read it! (Letter from Edsger Dijkstra, August 1, 1969).

After receiving a copy of the RC 4000 multiprogramming system manual, the Swiss computer scientist, Niklaus Wirth, wrote:

> I am much impressed by the clarity of the multiple process concept, and even more so by the fact that a computer manufacturer adopts it as the basis of one of its products. I have come to the same conclusion with regard to semaphores, namely that they are not suitable for higher level languages. Instead, the natural synchronization events are exchanges of message. (Letter from Niklaus Wirth, July 14, 1969)

For almost forty years, Wirth developed innovative programming languages, such as Euler, Algol W, PL 360, Pascal, Modula, Modula-2, and Oberon. When I first met him, he had returned from Stanford, after ten years in Canada and the United States, and was now an assistant professor at the ETH (The Federal Institute of Technology) in Zurich, Switzerland.

At a meeting at the ETH, two years before "Algol" became synonymous with Algol 60, European and American computer scientists had outlined an earlier version, called Algol 58. In May 1968, when the ETH celebrated the Tenth Anniversary of Algol 58, I was invited to participate in a panel discussion on operating systems. The panel, chaired by Niklaus Wirth, consisted of Alfred Schai (Switzerland), Michael Griffith (France), Brian Randell (England), Edsger Dijkstra (The Netherlands), Hans Rudolf Wiehle (Germany), and me (Denmark). About 50 computer scientists attended the discussion in an auditorium with terrible acoustics that made it difficult to hear anything. I don't remember much about the meeting, except that I met Niklaus Wirth for the first time.

In August 1968, Peter Naur, Paul Lindgreen, Søren Lauesen and I attended the IFIP Congress in Edinburgh. Tony Hoare, the inventor of the famous Quicksort algorithm, presented a paper on data structures in a two-level store. In the lobby of his hotel, he listened patiently, while I explained the concepts behind our multiprogramming system.

The small group of Danes at IFIP 68 soon became regulars at the conference bar. If I showed up early, the bartender would say: "Your friends are not here yet." Here I met David Howarth, the designer of the Atlas supervisor that pioneered multiprogramming and demand paging. After explaining that the Scots drink their best whisky and export the rest, David introduced me to Crawford's Five Star whisky. Sure enough, when I asked for this de luxe whisky in a liquor store in London, the owner explained that he, unfortunately, only carried Crawford's Three Star.

1968 was also the year in which the first Nato Conference on Software Engineering was held in Garmisch, Germany. Dijkstra (1999) viewed this as a turning point in the history of computer programming:

> It was there and then that the so-called "Software Crisis" was admitted and the condition was created under which programming as such could become a topic of academic interest. The latter, not surprisingly, turned programming from an intuitive activity into a formal one.

In October 1969 I attended the 2nd Nato Conference on Software Engineering in Rome, Italy (Naur 1969). About sixty people from eleven countries attended. Looking like a Who's Who in Programming, the list of participants included: Fritz Bauer, Bob Barton, Edsger Dijkstra, Tony Hoare, Butler Lampson, Roger Needham, Alan Perlis, Brian Randell, John Reynolds, Doug Ross, Jules Schwartz, Christopher Strachey, Niklaus Wirth, and Mike Woodger.

Niklaus Wirth's working paper on "The programming language Pascal and its design criteria" was my introduction to the first secure programming language that was powerful enough to implement its own compiler.

Edsger Dijkstra talked about "Structured Programming." This method of stepwise programming boils down to breaking a program into small abstract programs that can be divided further, until you reach a level of detail supported by the programming language. At that point, you turn around and combine the small, final pieces into a complete program. The method is similar to the mathematician's way of dividing a theorem into lemmas that can be verified separately and then used to prove the theorem.

What I remember most clearly is Butler Lampson from the Berkeley Computer Corporation. Butler talked like a machine gun. For those who don't know Butler: The rate of human speech is measured in "millilampsons." Butler is the only one who has reached the absolute limit of "1 lampson."

⋆ ⋆ ⋆

Years ago, I wrote an autobiographical sketch, entitled "The programmer as a young dog" (Brinch Hansen 1976d), about my time at Regnecentralen. The title was inspired by James Joyce's "A portrait of the artist as a young man" and Dylan Thomas's "Portrait of the artist as a young dog." When

I showed it to my colleague, Skip Mattson, at Syracuse University, he said: "I would like to know more about your Danish boss." All right then, I will tell you what I know about him.

Niels Bech was born in 1920 in Lemvig, a small Danish town in Northern Jutland. As a child and youth, he would stutter helplessly when he tried to pronounce the combination of an "s" and a "b" in his name. He invented the middle name, Ivar, to be able to say his own name. Growing up in a small, provincial city, he must have been at the receiving end of many cruel jokes.

Bech was a tall guy. At a movie theater in his hometown, a man behind him repeatedly asked him to sit down. Finally, Bech had enough—he stood up and turned around to show how tall he really was. At that point, somebody shouted: "My God, now he is standing on the seat!"

After the completion of Dask in 1957, Niels Ivar Bech became managing director of Regnecentralen. When I first met him, he was 43 years old. A man of many contradictions, he did not hesitate to make bold decisions that put his tiny company at financial risk. Yet, because he was afraid of taking exams at the university, he never completed a higher education.

He trusted his coworkers implicitly and let them pursue their own ideas with minimal intervention. He even tolerated that his hardware development groups, headed by Bent Scharøe Petersen and Henning Isaksson, used different standards of documentation. However, to achieve his goals, he would sometimes bypass his group managers. Bech's underground style of managing through unofficial channels was known as "moling." On his fiftieth birthday, his staff gave him a stuffed mole in a glass cage.

Sometimes, Bech's moling worked brilliantly. One of Scharøe Petersen's electronic engineers, Kurt Henning Andersen, wanted to develop the world's fastest tape reader, using an electronic buffer to stop the paper tape gradually without breaking it. Scharøe did not support the idea, correctly pointing out that Regnecentralen had no expertise in the development of electromechanical devices. However, when Bech heard about the idea, he gave Kurt enough money to develop the paper tape reader in his own kitchen. When the RC 2000 paper tape reader was presented in the fall of 1963, it read paper tape at the unbelievable rate of 2,000 char/sec. With a speed of 15 feet/sec, the tape emerged from the reader like exhaust from a jet plane during take-off, and landed in a waste basket eight feet away. Over a ten-year period, Regnecentralen sold about 1,200 RC 2000s.

But Bech's interference could also be frustrating. At a meeting with a potential customer, he once asked me how long it would take to finish the Algol compiler for the RC 4000. My realistic estimate was twelve months. That was not the answer Bech wanted to hear. So he turned to Jørn Jensen and asked him: "Don't you think we can do it in six months?" Taking the hint, Jørn said "Sure, we can." It made me angry that Bech undermined my credibility as software manager in front of a customer. As it turned out, it took eighteen months to finish the compiler.

In the end, I believe that Bech's unorthodox management style limited Regnecentralen's potential for future growth. His moling worked when Regnecentralen was small. But, eventually, he would have needed a growing staff of professional managers, who would not have tolerated his habit of bypassing them, whenever he found it convenient to do so.

However, in all other aspects, Bech was an inspiring leader. His directive for the RC 4000 software development was rather amazing. His only request to me was: "I need something new in multiprogramming!"

In my opinion Niels Ivar Bech was somewhat of a gambler and showman. He could rarely resist the temptation to do the unexpected. I once participated in a negotiation between Bech and a customer about the sale of an RC 4000 in the middle of a noisy discotheque. Perhaps it is true that unconventional acts rarely succeed in business (we did not sell a machine that evening), but they almost always work in research.

Research is gambling at the highest level. A cautious effort only leads to uninteresting results. A research director must have a sense of which problem to attack next and the courage to give his collaborators the freedom to solve it without imposing narrow constraints. The talent for inspiring his associates to create new things of world-wide renown was one that Bech possessed in the highest degree. Once you have known a leader with this intellectual courage, it is quite depressing to realize how extremely rare this quality is.

Niels Ivar Bech was a dreamer in the most creative sense of the word. His time scale was longer than the one I adopted as a young, impatient engineer. I found it unreasonable that he gave some of his associates time to write textbooks on computer science, without considering how this would influence the immediate needs of the company. That was short-sighted of me. While Bech gave younger colleagues the chance to create new things, he gave his senior people the opportunity to lay the foundation of computer science education in Denmark.

Denmark has made four world-class contributions to computer technology: the Algol 60 report, the Gier Algol compiler, the RC 2000 paper tape reader, and the RC 4000 multiprogramming system. Each of these products combined radically new ideas, which were years ahead of their time (and therefore could not be motivated by an immediate "need"). Without Niels Ivar Bech's brilliant sense of innovation, a small Danish company could probably not have attracted so many outstanding young engineers and be at the cutting edge of programming technology for more than a decade.

Bech's drive and vision went far beyond his job at Regnecentralen. From 1959, he was instrumental in organizing the Nordic Symposiums on Computing, known as NordSAM. In 1960, he became one of the founding members of the International Federation for Information Proccessing (IFIP). For his contributions to IFIP, Bech received the Silver Core Award in 1974.

The decision to publish a Scandinavian journal of computing was made over a glass of beer in Bech's office in 1960. Bech provided economic support and offered to let Regnecentralen handle the administration and distribution. The first issue of the journal BIT gave readers the following choice:

1. *Yes*, I want to subscribe.
2. *No*, I do not hesitate. Put me on your subscription list.
3. *I don't know* of any good reason why I should not subscribe.

From its start, Peter Naur served as co-editor of BIT. His seminal papers on the Gier Algol compiler, elimination of go to statements, type checking, program assertions ("general snapshots"), and modular programming ("action clusters") all appeared in BIT. Naur's paper on *Go to statements and good Algol style* (1963b) appeared five years before Dijkstra's more widely publicized *Go to statements considered harmful* (1968a).

My early papers on the Siemens Cobol compiler, the RC 4000 architecture and the real-time system at Pulawy were also published in BIT.

During the cold war, American companies were not allowed to sell computers in Eastern Europe. This gave Bech a unique opportunity to sell Regnecentralen's equipment in Poland, Czechoslovakia, Hungary, Bulgaria, Rumania, East Germany, and Yugoslavia.

If a communist country was short on western currency, Bech was not above a little horse trading (seriously). On one occasion he apparently delivered computer equipment to Poland in return for a shipment of horseflesh, which he somehow managed to sell in Denmark.

His boundless energy and visionary thinking made it inevitable that some of his efforts would meet resistance. Around 1960, the Technical University of Denmark and the National Engineering and Science Foundation (Statens teknisk-videnskabelige fond) recommended that the government support the use of Gier computers for research and education at Danish universities. This idea was successfully opposed by Willy Olsen, manager of the government's own computing center, Datacentralen, opened in 1959 at the initiative of Viggo Kampmann, minister of finance. I once asked Kampmann, who was married to my father's cousin, why he supported the creation of Datacentralen. He said he thought competition would be good for Regnecentralen. Willy Olsen, apparently, did not share Kampmann's belief in competition.

Throughout the 1960s, Bech tried unsuccessfully to persuade Scandinavian computer manufacturers to merge. Once, Bent Scharøe Petersen and I accompanied him to a meeting with DataSaab in Linkőping, Sweden. That evening, we returned to Copenhagen on a tiny airplane, flying through a blizzard with zero visibility. Sitting in front with the pilot, Scharøe mentioned that we had lost all radio contact. That did not seem to worry Bech in the least.

On another flight in Bulgaria, passengers started screaming and praying when an engine caught fire. The story goes that Bech calmly ordered beer for everybody.

Towards the end of the 1960s, it became increasingly clear that the pioneering era of the Danish computer industry was coming to an end. In 1970, I left Regnecentralen and moved to the United States. At that time, Niels Ivar Bech was already showing signs of illness. Since then I only saw him briefly at the IFIP 71 Congress in Milena's hometown, Ljubljana, in Slovenia.

In 1971, Bech was fired by Regnecentralen's board of directors. I don't know why he was dismissed. In his dealings with business leaders and government employees, he probably had the misfortune of thinking big among people who were not used to thinking big. Since he was decades ahead of his time, Bech undoubtedly would have found some of his board members shortsighted. They, in turn, would almost certainly have found him unrealistic (as I did, on the occasion when Bech told me that his goal was "to push IBM back into the Atlantic Ocean"). Perhaps, in the words of Vartan Gregorian, "He did not and could not serve people he did not respect, especially those who were political hacks, men without integrity, mission or vision, empty suits." Who knows?

On July 25, 1975, he died of a heart attack at age 55. With Niels Ivar Bech's death, Denmark lost its leading role in the development of programming technology. Four years later, Regnecentralen ceased to exist. However, by then it hardly mattered. Bech had already made a lasting contribution to his country by training the first generation of computer pioneeers and laying the groundwork for computer science education in Denmark. Over the years, a number of Regnecentralen's senior people became faculty members at Danish universities: Peter Naur, Aage Melbye, Ole Møller, Poul Sveistrup, Christian Andersen, Henning Isaksson, Henning Bernhard Hansen, Christian Gram, Peter Kraft, Søren Lauesen, Paul Lindgreen, and I (for a short time).

Those of us, who were privileged to start our careers under Bech's visionary leadership, will always remember Regnecentralen as the lost paradise.

In 1983, I dedicated my book, *Programming a Personal Computer*, to the memory of Niels Ivar Bech.

⋆ ⋆ ⋆

My years at Regnecentralen were some of the happiest years of my professional life. I had worked in compilers, computer architecture, and operating systems. And I had met four computer scientists, who would influence my future work: Peter Naur, Edsger Dijkstra, Niklaus Wirth, and Tony Hoare.

It was time to move on. I was now planning to go abroad and write the first systematic textbook on operating system principles.