# 6

## *INVENTING THE FUTURE 1972–76*

*Can you tell me, what is Caltech? – Sunshine and palm trees in February – Wine-tasting with the dean – Driving across America – A question of priority – Two baffling problems – That sounds easy – Concurrent Pascal and Solo – Al Hartmann's compiler – The art of compromise – Getting a sore throat in Bombay – Returning to Marktoberdorf – The distraction of grants – Leaving the magic kingdom – A passion for clear thinking.*

After the completion of my operating system book, it was time to decide whether we should return to Europe or stay in the United States. One day, I received a long-distance phone call from Pasadena in Southern California. The caller said: "This is Gilbert McCann from Caltech. Would you be interested in being interviewed for a faculty position?" There was a long pause at the other end of the phone when I answered: "Certainly, but can you tell me, what is Caltech?"

California Institute of Technology is probably America's most demanding college of science and engineering. With about 900 undergraduates, 900 graduates and 300 faculty, it has one professor for every three undergraduates. The students are among the top 1% in the nation. Some of them are handpicked by professors who visit high-schools around the country.

Caltech has had a major effect on the aerospace industry in Southern California. The Jet Propulsion Laboratory, America's first center for space research, is staffed and managed by Caltech. After a flight of 380 million kilometers in 167 days, the Mariner 9 spacecraft was placed in orbit around Mars on November 13, 1971. By the end of June 1972, it had taken over 7,000 pictures and mapped the entire surface of the planet.

In his yearly report for 1971/72, Caltech president Harold Brown wrote: "Our traditions and capabilities at Caltech are strongly focussed on the most fundamental matters, at the leading edge of knowledge in each discipline.

Few of us would be at Caltech if we did not believe that such efforts are of surpassing value." I remember one professor telling me, "If MIT is working on something, we are not interested. Caltech cannot do everything. So we concentrate on areas in which we are unique." This philosophy has certainly paid off. In the first century of the Nobel Prize, 27 Nobel laureates have been associated with Caltech.

Caltech students are famous for their imaginative pranks. California Boulevard divides the campus into two parts. To walk across it, you have to push a button on a traffic signal and wait for a green light. On one occasion, students unscrewed the light cover, put the green glass at the top and the red one at the bottom of the signal. Drivers now had to get out of their cars and push the button to get a (brief) green signal. During rush hour, a mile-long line of cars moved at snail's pace. Since the city administration knew that Caltech students are very intelligent, they assumed that the wiring of the signal had been changed in some way. So they sent an electrician over to check it out. He was very puzzled when he couldn't find anything wrong—until a Caltech student walked up to him and said: "Excuse me, I thought the red light was supposed to be on top of the green one." Great ideas are often simple!

On "Ditch Day," Caltech seniors ditch their classes and vanish from campus. Any senior found on campus risks being caught and tied to a tree with duct-tape. Senior students have secured their doors in elaborate ways. Underclass students must then try to get past these "stacks" and into the seniors' rooms. In one memorable instance, a senior had filled his room completely with an enormous water balloon. When some students cut a hole in the balloon, a flood of water swept them down the corridor.

⋆ ⋆ ⋆

It was my luck to arrive at Caltech just as they were starting up a department of Information Science. In September 1971, Caltech dedicated the Jorgensen Laboratory for information and computer science. The building was a gift of Earle M. Jorgensen, a Trustee of the Institute since 1957, and his wife. An elite institution like Caltech, financed by wealthy donors, would (unfortunately) be unthinkable in most European countries, due to the prevailing egalitarian attitudes and high levels of taxation.

On Wednesday, February 16, 1972, at 4 p.m., I gave a seminar at Caltech on "Structured multiprogramming."[1] I described conditional critical regions

[1] P. Brinch Hansen, Structured multiprogramming. *Communications of the ACM 15*,

and explained how they waste unpredictable amounts of processing time by reevaluating boolean conditions until they are true. I then showed how to eliminate this inefficiency by extending a programming language with queuing variables, which give the programmer complete control of process scheduling within conditional critical regions. This idea became an essential ingredient of the future monitor notation.

My trip to Southern California was a welcome break from the winter in Pittsburgh. I enjoyed walking around Caltech in February, without a coat, looking at the palm trees and the beautiful San Gabriel mountains, a few miles north of Pasadena.

Two days later, I was invited to a wine-tasting dinner at the Athenaeum, Caltech's magnificent faculty club, built in Mediterranean style with beautiful landscaping and tennis courts. This elegant building was also a gift from private donors. The first formal dinner, in February 1931, was attended by three Nobel Prize winners: Albert Einstein, Robert Millikan and Albert Michelson.

The wine-tasting was held in a large, oak-panelled dining room. My host was the sixty-year old Francis Clauser, the charming chairman of the Division of Engineering and Applied Science. He had done brilliant research in aeronautics and was a member of the National Academy of Engineering. During the dinner, we tasted German white wines and California red wines from Caltech's private wine cellar. The wines were discussed by the wine chairman, professor Harold Wayland.

Within a month, Caltech obtained letters of recommendation for me from Tony Hoare (Belfast), Don Knuth (Stanford), Butler Lampson (Xerox Parc), Roger Needham (Cambridge), Alan Perlis (Carnegie-Mellon), and Niklaus Wirth (ETH Zurich).

In March 1972, I was invited to visit Caltech again, this time with Milena and our two small children. At the Los Angeles Airport, Caltech's private limousine was waiting for us. When the driver saw Milena's winter coat, he said "You won't be needin' that here, Ma'm!" After driving thirty miles north on the Harbor and Pasadena Freeways, we reached Colorado Boulevard in Pasadena, where Caltech had reserved a motel room for us.

In the evening, Gilbert McCann and his wife Betty took us out to dinner at a family restaurant. I remember Betty asking me: "Isn't it a handicap not to have a PhD?" I looked surprised and said: "No, I don't think so." The only degree offered by the Technical University of Denmark was a five-year

July 1972.

Master's. I still have reservations about the PhD. It seems to me, that it is not a good idea to ask young people to spend some of their most creative years taking more courses, passing final exams, and doing research that fits into the ideas (and grants) of their professors.

The Caltech physicist, Richard Feynman, was even more blunt about his role as a PhD advisor (Gleick 1992):

> I do not like to suggest a problem and suggest a method for its solution and feel responsible after the student is unable to work out the problem by the suggested method...What happens is that I find that I do not suggest any method that I do not know will work and the only way I know it works is by having tried it out at home previously, so I find the old saying that "A Ph.D. thesis is research done by a professor under particularly trying circumstances" is for me the dead truth.

Back at Carnegie-Mellon, I wrote a letter informing family and friends that "California Institute of Technology has offered me a faculty position as an Associate Professor of Computer Science...I have now accepted this offer and will start working [at Caltech] on July 1."

⋆ ⋆ ⋆

Our trip across the United States and our life in California are described in Christmas letters to family and friends. We left Pittsburgh on June 1 and drove 3,100 miles (about 5,000 km) across America to Los Angeles. First, we went to Montreal, where I chaired a session on the design of operating systems at the Canadian Computer Conference. On the way, we got a last glimpse of Niagara Falls. From then on, we drove west through the states of Quebec, Ontario, Michigan, Illinois, Iowa, South Dakota, Wyoming, Utah, Nevada and California. Fifteen days in a row we traveled west, every evening towards a radiant sunset.

South Dakota made the greatest impression. Badlands: desert-like rock formations eroded by rivers below the surrounding flat country for millions of years. Mount Rushmore, where Gutzon Borglum, son of a Danish immigrant, carved 60-feet tall sculptures of George Washington, Thomas Jefferson, Abraham Lincoln, and Theodore Roosevelt on a granite mountain. The day we left Mount Rushmore, a dam collapsed, flooding the entire area and killing many people.

South Dakota is right in the middle of the old Wild West. This is where Buffalo Bill came from, and the Sioux Indians still live here. In Custer State Park we went by Jeep to see a flock of buffalos on the prairie. In Deadwood City we were shown around the Broken Boot gold mine by an old man who remembered both Calamity Jane and Wild Bill Hickock from his childhood. On Boot Hill, high above the city, we found their tombstones.

Yellowstone National Park in Wyoming is larger than Delaware and Rhode Island combined. It has an incredible collection of natural wonders in one place: hot springs, geysers, waterfalls, mountains and lakes. At one point, Milena got out of the car to photograph a black bear sitting beside the road. The bear didn't like that, so she had to run back to the car. Inside the park, we stayed overnight in a five-story alpine lodge.

The next day, we headed south, through the Grand Teton National Park. In Salt Lake City, Utah, we saw the Mormon Temple with the statue of Christ by the Danish sculptor, Bertel Thorvaldsen. In Salt Lake City, we bought a brand new set of radial tires, guaranteed for 40,000 miles. Sixty miles later, we had a flat tire in the middle of the Great Salt Desert. After mounting a worn-down spare, we crawled though Nevada until we reached Reno where we bought another tire (which later was punctured in Los Angeles).

In San Francisco, I wanted to show my family the Golden Gate bridge. But as we drove across it in the late afternoon the fog drifted in from the ocean and hid everything. We now drove south along the Pacific Ocean on the Cabrillo Highway though Monterey, portrayed in Steinbeck's novels. Finally we reached Los Angeles.

At Caltech we were met by a lady who handed us the key to a house we could live in over the summer. It was a white, two-story house on Boulder Road in the suburb of Altadena. On the second floor, a covered balcony extended from one end of the house to the other.

Two days later, I flew back to the east coast in four hours to lecture on the RC 4000 multiprogramming system at a Summer Institute of Computer Science at University of Maryland.

Greater Los Angeles: 8 million people in one place. Two murders a day and several thousand robberies. Brilliant weather (if you dare to breathe the air). The wind from the ocean blows the smog from downtown towards Pasadena, where the mountains stop it. Sometimes you could not see the Sierra Madre mountains, a few miles north of Caltech.

On the positive side, Southern California had so many attractions for the children: Los Angeles Zoo, Universal Studios, Magic Mountain, Farmers

Market, Knott's Berry Farm, Lion Country Safari, Marineland, Japanese Village, Busch Gardens, La Brea Tar Pits, Mount Wilson Observatory (and its Skyline Park with llamas, deer, goats and turkeys), the Queen Mary (the world's largest ocean liner), Disneyland, and, of course, the ocean beaches.

⋆ ⋆ ⋆

At age 27, Gilbert McCann earned a PhD in electrical engineering from Caltech. As a graduate student, a two-million-volt stroke from a surge generator paralyzed all his outer nerves and muscles for 24 hours and damaged one of his eyes permanently.

During World War II, he designed an analog computer that made it possible to shoot down most of the German V-1 rockets when they reached the coast of England. After the war, he became a faculty member at Caltech and started building a huge analog computer. By the 1950s, McCann's lab served every aircraft company in America and Europe. When the workload became too much for Caltech to handle, they spun off a commercial company, Computer Engineering Associates, with McCann as the largest shareholder.

When I met him in 1972, he was sixty years old. His manners were somewhat brusque. Since Americans habitually abbreviate first names (which we don't in Denmark), I wasn't sure what to call him. So I asked him: "Do you want me to call you Gilbert or McCann?" In an annoyed tone, he answered "I respond to either," which was not very helpful to me.

In his efforts to dominate his department, McCann could be ruthless towards faculty members whose research and financial support were independent of his own. This was obvious in his relationship with professor Fred Thompson, who had been the most promising student of the famous logician Alfred Tarski. At Caltech, Fred made a truly courageous gamble on the future of computing by working on the problem of using English for human interaction with computers. After several years of hard work, he had finished a software system in assembly language for a particular computer. At that crucial stage, McCann used his influence to support an offer from IBM to replace that computer with another one that was unable to execute Fred's program. Years of programming were wasted and Fred had to start all over again.

⋆ ⋆ ⋆

In 1961, Fernando Corbató pioneered timesharing at MIT. Ten years later, computing at Caltech was still a cumbersome affair based on old-fashioned

batch processing. First, you used a noisy machine, the size of a small desk, to punch your program on IBM cards. Then you carried your deck of cards to the neighboring Booth Center for Computing, and gave it to an operator behind a counter. Several hours later, you walked back to the center and picked up your punched cards and printed output from one of the small "pigeonholes" arranged alphabetically by user names.

The overriding concern was to keep Caltech's *mainframe* computer running efficiently with as little human intervention as possible. You were not allowed anywhere near the computer equipment. Operators collected decks of punched cards from users and used a small computer to input a batch of jobs from punched cards to a magnetic tape. This tape was then mounted on a tape station connected to the mainframe computer. The jobs were now input and run one at a time in their order of appearance on the tape. The running jobs output data on another tape. The output tape was moved to a small computer and printed on a line printer. While the mainframe computer executed a batch of jobs, the small computers simultaneously printed a previous output tape and produced the next input tape. The final task of the operators was to separate the printed output manually and place it in the correct pidgeonholes.

Batch processing was severely limited by the sequential nature of magnetic tapes and early computers. Although tapes could be rewound, they were only efficient when they were accessed sequentially. And most computers could only execute one program at a time. It was therefore necessary to run a complete batch of jobs at a time and print the output in "first-come, first-served" order.

To reduce the computer time that was lost while operators changed magnetic tapes on the mainframe computer, it was essential to batch many jobs on the same tape. Unfortunately, large batches greatly increased service times from the users' point of view. It would typically take hours (or even a day or two) before you received the output of a single job. If the job involved a program compilation, the only output for that day might be an error message caused by a misplaced semicolon!

On campus there was an economic conflict between students, who needed to compile and run small programs with reasonable turn-around times to meet their deadlines for homework, and researchers, who ran large computations supported by research grants. The compromise adopted was to run large jobs at night or on weekends.

McCann directed the computing center for seven years. In his alloca-

tion of computer time for research, he apparently favored some faculty over others. Finally, a group of well-funded faculty went to the chairman of engineering and threatened to buy computer time outside campus, unless he replaced McCann, which he did.

⋆ ⋆ ⋆

I posed no threat to McCann's power and found him quite supportive of my work at Caltech. His main interest was now using computers to study the nervous system of the fly. He left it to younger faculty, including Giorgio Ingargiola and me, to develop academic courses.

Giorgio spoke English with a pronounced Italian accent interrupted frequently by an infectious laugh. His office was next door to mine. At noon, we would walk across the sunny campus and enjoy lunch at the Athenaeum. He taught a course on formal models of computation and directed a programming laboratory with student projects.

Caltech had a trimester system. In the first trimester, I taught structured programming, followed by compiler design in the second trimester, and ending with operating systems in the third trimester. These courses could be taken by students from any department. The enrollment in each course was 60–70 students, which was a large class at Caltech. I enjoyed teaching these smart kids, who raised many questions in class and often came up to me after class to continue our discussions.

The Caltech students published a booklet with candid comments about the teaching abilities of the faculty. About one professor, they wrote: "He obviously knows his stuff, and so would you—if only you could stay awake in his class!" They described my courses as "An easy way to get an A" (a viewpoint not shared by many students at Syracuse University).

My experience at Regnecentralen had taught me that professional programming is not a form of unsystematic trial-and-error. You need to think deeply until you understand exactly what you want your program to do. Before you compile a program for the first time, you should proofread it for logical consistency. And before you run it, you need to prepare a systematic testcase with output that demonstrates that every line of the program has been executed.

In this view of programming, thinking time is much more important than computer time. Programming takes place at a desk away from any computer. Today, when I see faculty and students spending hours at computer terminals, I wonder: Are they really thinking deeply, or are they just typing? Modern computing has turned us into amateur typists.

Thirty years ago, when we had to use batch processing with slow turnaround, the idea of using the computer as little as possible made all the more sense to me. In those days, students needed written permission from their instructors to set up personal accounts with limited amounts of computer time. To encourage my students to think more and compute less, I gave them less computer time than they needed to complete the compiler project. When their accounts ran out, I gave them 50 percent more, then 25 percent and so on.

At one point, Francis Clauser informed me that my well-intentioned policy had the unexpected side-effect of making some students "borrow" computer time from the accounts of other students. This was a clear violation of Caltech's Honor Code which states that no member of the community shall take unfair advantage of any other member of the community. As soon as I heard that, I sacrificed my miserly approach and gave everybody as much computer time as they needed.

The Honor Code gave us all remarkable freedom. You could, for example, tell students to go home and solve exercise 2.2.6.8 in Knuth's book on "Fundamental Algorithms"—without looking in the answers section! However, human nature being what it is, it was not easy for students to live up to the Honor Code. A survey showed that while most of them strongly supported the Honor Code, few were prepared to turn in their friends for violating it.

⋆ ⋆ ⋆

In the spring of 1972 I read about the *class* concept invented by the Norwegians Ole-Johan Dahl and Kristen Nygaard for their programming language Simula 67. Although Simula was *not* a concurrent programming language, it inspired me in the following way: So far I had thought of a monitor as a program module that defines all operations on a *single* data structure. From Simula I learned to regard a program module as the definition of a *class* of data structures accessed by the same procedures.

This was a moment of truth for me. Within a few days I wrote a chapter on resource protection for my operating system book. I proposed to represent monitors by *shared classes*. My book included a single monitor for a message buffer. Figure 1 shows it in a slightly simplified form. The shared class is a program module that combines three things: (1) the data representation of a message buffer, (2) the send and receive procedures, which define the only possible operations on a buffer, and (3) a statement that defines the initial buffer state as empty.

```
shared class buffer =
  slot: integer; empty: boolean;

procedure send(message: integer)
begin
  await empty;
  slot := message;
  empty := false;
end;

procedure receive(var message: integer);
begin
  await not empty;
  message := slot;
  empty := true;
end;

begin empty := true end;
```

Figure 6.1 The first monitor notation.

The key idea is that processes only have indirect access to the variables of the shared class. They can call the send and receive procedures, which operate on the buffer variables, but they do not have direct access to these variables. This scope rule has an important implication for program reliability: Once you have programmed and tested a shared class, it remains correct, and cannot easily be corrupted by other parts of the program.

A shared class is a notation that explicitly restricts the operations on a shared data structure and enables a compiler to check that these restrictions are obeyed. It also indicates that all operations on a particular instance must be executed as *critical regions*. In short, a shared class is a monitor type. My decision to use *await statements* in the first monitor proposal was a matter of taste. I might just as well have used the *queuing variables*, which I had proposed in 1972.

⋆ ⋆ ⋆

In the spring of 1972, I had sent Tony Hoare a copy of my book manuscript which included my monitor concept. Six month later, he submitted a paper

on "A structured paging system," which was published in the fall of 1973, one month after the publication of my book. In this paper, Hoare used my shared classes and queuing variables, with minor changes, to outline an unimplemented demand paging system.

As an engineer, I had serious reservations about this paper. Nobody can have confidence in a theoretical specification of something as complicated as a demand-paging system—unless the validity of the model has been tested in an actual implementation. At Regnecentralen I had defined the instruction set of the RC 4000 computer completely by an Algol 60 program. Had we not built this computer, my hardware specification would have remained an unpublished, academic exercise.

On a personal level, I was surprised and hurt to find that, instead of citing my book as the original published source of the monitor concept, Hoare thanked me (and others) vaguely "for ideas, discussion, inspiration, and criticism on points too numerous to recall." When I pointed out that this was unacceptable, he acknowledged my invention of monitors in a tutorial, published in the following year (Hoare 1974a). However, the damage had been done, and, for years, people would continue to call them "Hoare's monitors."

Looking back, it was, of course, naive of me to publish the monitor concept in a textbook, instead of a professional journal. But I was young and idealistic and felt that my first book should include at least one original idea. It did not occur to me that researchers rarely look for original ideas in undergraduate textbooks.

At that point, I considered it premature to write a tutorial on the monitor concept. My professional standards were deeply influenced by Naur and Jensen's Gier Algol compiler, Dijkstra's THE multiprogramming system, Regnecentralen's RC 4000 multiprogramming system, and Wirth's Pascal compiler. Every one of these systems had been implemented *before* it was described in a professional journal. Since this was my standard of software research, I decided to implement monitors before writing more about them.

⋆ ⋆ ⋆

At Caltech, I started thinking about defining a programming language with concurrent processes and monitors. To reduce the effort, I decided to include these concepts in an existing sequential language. Pascal was an obvious choice for me, since I had used the language in my operating system book. I named the new language *Concurrent Pascal*. Apart from that, nothing else was obvious.

With a notation for monitors now in hand, you would think it would be easy to include it in Pascal. I had no idea of how to do this. I remember sitting in my garden in Altadena, day after day, staring at a blank piece of paper and feeling like a complete failure.

I faced two baffling problems for the first time: (1) how can you make a concurrent programming language *secure* from time-dependent behavior by using extensive compilation checks and minimal run-time checks? (2) When concurrent processes terminate, is it possible to reclaim and reuse their memory spaces without resorting to slow "garbage collection?" It took me almost two years to find reasonable solutions to the first problem and make compromises that enabled me to ignore the second one.

In September 1973, I sent Mike McKeag "a copy of a preliminary working document that describes my suggestions for an extension of Pascal with concurrent processes and monitors." This is the earliest evidence of Concurrent Pascal. In April 1974, I distributed a report on "Concurrent Pascal: a programming language for operating system design."

Concurrent Pascal extends Pascal with program modules defining monitor, process, and class types. (Since class types are related to sequential rather than concurrent programming, I will ignore them here.)

The monitor shown in Fig. 6.2 defines a single-slot buffer as a new data type. If a process tries to receive a message from an empty buffer, the monitor delays that process in a queuing variable. When another process sends a message through the same buffer, the monitor immediately continues the execution of the delayed process within the receive procedure. The process that performs the continue operation automatically returns from the send procedure. This *context switch* ensures mutual exclusion of monitor calls. Sending is similar to receiving.

Figure 6.3 defines a (trivial) process type that copies an endless stream of integers from one buffer to another.

The syntax clearly shows that each module defines a data structure and all the possible operations on it. The compiler must check that (1) every process and monitor only refers to its own variables; (2) processes interact through monitor procedures only; and (3) processes do not deadlock by calling monitors recursively (either directly or indirectly).

I now understood what I was doing. One day the president of Caltech, Harold Brown, came to my office and asked me to explain my research. After listening for half an hour, he said, "That sounds easy." I agreed because that was how I felt at the time. Caltech sure was different! This was the only time

```
type buffer =
monitor
slot: integer; empty: boolean;
sender, receiver: queue;

procedure entry send(message: integer);
begin
  if not empty then delay(sender);
  slot := message;
  continue(receiver);
end;

procedure entry receive(var message: integer);
begin
  if empty then delay(receiver);
  message := slot;
  continue(sender);
end;

begin empty := true end;
```

Figure 6.2 A monitor type.

in my life, I had the opportunity to discuss my research with a university president.

⋆ ⋆ ⋆

More than anyone else, Gordon Bell was the driving force behind the minicomputer revolution. At Digital Equipment Corporation, he was the main architect of the PDP 11, the first minicomputer that was powerful enough to support modern programming languages. When I first met him, he was spending a sabbatical year at Carnegie-Mellon. By 1975, as vice president of engineering at DEC, Gordon and his team had designed the 32-bit VAX computer, which became the standard computer for science and engineering. In 1983 he started Encore Computer, which built the Encore Multimax, a multiprocessor that I would later use for parallel programming at Syracuse University. In 1991, president George Bush awarded Gordon Bell the National Medal of Technology.

```
type copyprocess =
process(inp, out: buffer);
value: integer;
begin
  cycle
    inp.receive(inp, value);
    out.send(out, value);
  end
end;
```

Figure 6.3 A process type.

In 1970, when the first PDP 11s were delivered, over 170,000 were sold. At Caltech, McCann acquired a PDP 11/45 for his lab. Since it cost only a fraction of a mainframe computer, it was operated in open shop mode (just like Regnecentralen's Gier computer had been).

I had already made Pascal available for students on Caltech's mainframe computer. In this effort, I was assisted by Robert Deverill, a professional programmer working for McCann. At the time, no minicomputer supported Pascal. So we had to program the Concurrent Pascal compiler in Pascal and test it on Caltech's mainframe computer before moving it laboriously to the PDP 11.

An early six-pass compiler was never released. Although it worked perfectly, I found it too complicated. Each pass was written by a different student who had difficulty understanding the rest of the compiler.

From June through September 1974 my first PhD student, Al Hartmann, wrote another Concurrent Pascal compiler. His goal was to be able to compile small operating systems on a PDP 11/45 with at least 32 K bytes of memory and a slow, removable disk (about two feet in diameter). The compiler was divided into seven passes to fit into the small memory. It consisted of 8,300 lines written in Pascal and could be completely understood by one person. Systematic testing of the compiler took three months, from October through December 1974.

The Concurrent Pascal compiler was used from January 1975 without problems. It was described in Hartmann's PhD thesis (1975), later published as a monograph.

In another month Al Hartmann derived a compiler for a Pascal subset,

which we called *Sequential Pascal* (Brinch Hansen 1975b). On the PDP 11, it compiled the largest pass of the Concurrent Pascal compiler in 3 min. The compilation speed was limited mostly by the slow disk.

The Concurrent Pascal compiler generated code for a simple machine tailored to the language. I borrowed this idea from a portable Pascal compiler distributed by Wirth's group (Nori 1974). My main concern was to simplify code generation. The portability of Concurrent Pascal was just a useful by-product of this decision. Twenty years later, the Java language would resurrect the idea of "platform-independent" concurrent programs. Unfortunately, Java replaced the secure monitor concept of Concurrent Pascal with *insecure* shortcuts (Brinch Hansen 1999b).

The Concurrent Pascal machine was simulated by a kernel of 8 K bytes written in assembly language. The kernel multiprogrammed the PDP 11/45 processor among concurrent processes and executed them using an efficient technique known as *threaded code* (Bell 1973). It also performed basic input/output from a typewriter, a disk, a magnetic tape, a line printer, and a card reader.

I defined the kernel in Pascal (extended with classes). Tom Zepko, a Caltech undergraduate, helped Bob Deverill hand-translate the kernel into assembly language for the PDP 11. It was completed in January 1975 and described in a report (Brinch Hansen 1975d).

*The programming tricks of assembly language were impossible in Concurrent Pascal*: there were no typeless memory words, registers, and addresses in the language. The programmer was not even aware of the existence of physical processors and interrupts. *The language was so secure that concurrent processes ran without any form of memory protection.*

In defining Concurrent Pascal, I made major compromises to make program execution as efficient as possible on a minicomputer that could only address two small memory segments simultaneously: (1) All procedures were non-recursive, (2) All processes, monitors, and classes existed forever, and (3) All processes and monitors were activated by an initial process.

These compromises made memory allocation trivial. The first rule enabled the compiler to determine the memory requirements of each module. The first two rules made static memory allocation possible. The third rule made it possible to combine the kernel, the program code, and all monitor variables into a single memory segment that was included in the address space of every process. This prevented fragmentation of the limited address space and made monitor calls almost as fast as simple procedure calls.

By putting simplicity and efficiency first we undoubtedly lost generality. But the psychological effect of these compromises was phenomenal. Suddenly an overwhelming task seemed manageable.

⋆ ⋆ ⋆

In January 1975, Milena and I traveled to India. The United Nations had donated funds for the country to open a center of software research and acquire a large computer at the Tata Institute of Fundamental Research in Bombay. To celebrate this event, a conference was organized for all computer science teachers in the country. Bill Wulf from Carnegie-Mellon, Rod Burstall from the University of Edinburgh, and I were invited to lecture.

Since Bombay is on the opposite side of the globe, we broke the long flight from Los Angeles in half by stopping overnight in Frankfurt, Germany. In Bombay we stayed at the famous Taj Mahal hotel. The place was swarming with international guests: oil sheiks in white garments with golden stripes, Indian women in colorful saris, Japanese tourists—and us. The hotel had French, Indian, and Chinese restaurants, as well as some sort of cafeteria. In addition there was room service twenty-four hours a day.

The first couple of days, I ate Indian breakfasts. Since I couldn't read the menu, I started from the top and ordered a new dish every morning. A typical dish consisted of some very spicy curry balls. After a few days, I had to stop this diet and see the hotel's doctor about my sore throat.

Right outside our luxury hotel, the poor were sleeping on the pavement. It was difficult to accept that small children of the same age as our children were running ahead of us begging.

After a week of lecturing, all of us flew to Aurangabad to see the famous Ajanta caves—a row of temples cut into massive rock centuries ago. Later, Milena and I traveled inland to Hyderabad and visited a company that produced a minicomputer similar to the PDP 11. I noticed that most of their peripheral devices came from communist countries in Eastern Europe. When we returned to our hotel room after dinner and turned on the lights, an army of well-fed cockroaches scampered under the bed. We called room service and they sent an employee who sprayed the room with kerosene. That night we slept with the lights on, breathing the smelly fumes.

At Caltech, I had an Indian graduate student, named Sriram Udupa. One evening, Milena and I visited his family in Bombay. They were orthodox Brahmins. They served rice and thin fine bread on tin plates covered with palm leaves. We sat in a circle on the floor and ate with our fingers, since

Brahmins regard knives and forks as unclean. The dinner was intended for the men and their sons only. The women served and watched us eat. However, they made an exception for Milena and allowed her to eat with us. It was all very dignified and made a deep impression.

After the conference in Bombay, Milena and I flew to New Delhi and on to Agra to see the famed Taj Mahal mausoleum, built by the Mogul emperor Shah Jahan (1592–1666) for his favorite wife, Mumtaz Mahal. The only thing we did that day was sit in the park in front of this master piece built of white marble with inlaid semiprecious stones. To me, Taj Mahal was as unique as Michelangelo's sculptures in Florence, Italy—one of those rare miracles, which human beings create once every five hundred years. In the basement under the building there was a marble casket with a big hole in the lid. The small detail that was missing was the 160-carat Kohinoor diamond, which an Indian prince gave England's Queen Victoria. It is now part of the British crown jewels.

The Taj Mahal reminded me of one of my favorite quotes (Bronowski 1973):

> The most powerful drive in the ascent of man is his pleasure in his own skill. He loves to do what he does well and, having done it well, he loves to do it better. You see it in his science. You see it in the magnificence with which he carves and builds, the loving care, the gaiety, the effrontery. The monuments are supposed to commemorate kings and religions, heroes, dogmas, but in the end the man they commemorate is the builder.

In spite of the exotic sights, it was depressing to visit India after growing up in a Scandinavian welfare state. Right outside the Taj Mahal, there was a small, stinking village. They said that American pilots slept in their airplanes in Calcutta to avoid seeing the hell, which was called life there. And Bombay smelled like a garbage dump everywhere (even inside the Taj Mahal hotel). Life seemed depressing even for the well-educated middle class of engineers and researchers we met.

After another stop in Frankfurt we returned home to our children, who had enjoyed staying with friends and never missed us.

⋆ ⋆ ⋆

After returning to Caltech, I wrote three model operating systems in Concurrent Pascal to evaluate the language. The modular concurrency had a dramatic (and unexpected) impact on my style of programming.

It was the first time I had programmed in a language that enabled me to divide programs into modules that could be programmed and tested separately. The creative part was clearly the initial selection of modules and the combination of modules into hierarchical structures. The programming of each module was often trivial. I soon adopted the rule that each module should consist of no more than one page of text. This discipline made programs far more readable and reliable than traditional programs that operate on global data structures.

In May 1975 I finished the *Solo* system, a single-user operating system for the development of Concurrent and Sequential Pascal programs on a PDP 11/45. The operating system was written in Concurrent Pascal. All other programs, including the Concurrent and Sequential Pascal compilers, were written in Sequential Pascal. The heart of Solo was a job process that compiled and ran programs stored on a removable user disk. Two additional processes performed input and output simultaneously. System commands enabled the user to replace Solo with any other Concurrent Pascal program stored on disk, or to restart Solo again. Al Hartmann had already written the compilers. I wrote the operating system and its utility programs in three months. Wolfgang Franzen measured and improved the performance of the disk allocation algorithm.

The Solo system was the first major example of a concurrent program consisting of processes, monitors, and classes (Brinch Hansen 1975c). It enabled us to use Sequential and Concurrent Pascal on the PDP 11/45 without going through the cumbersome batch processing at Caltech's computing center.

At Regnecentralen we had used the RC 4000 computer to implement process control programs for a chemical plant, two power plants, and a weather bureau. These real-time applications had one thing in common: each was unique in its software requirements. Consequently the programs were expensive to develop.

When the cost of a large program cannot be shared by many users, the only practical way of reducing cost is to give process control engineers a high-level language for concurrent programming. I illustrated this point by means of a real-time scheduler, which had been programmed in assembly language at Regnecentralen. I now reprogrammed the same scheduler in Concurrent Pascal.

The *real-time scheduler* executed a fixed number of task processes with frequencies chosen by an operator. I wrote it in three days. It took 3 hours

of machine time to test it systematically. Writing a description took another couple of days. So the whole program was developed in less than a week (Brinch Hansen 1975e).

At the end of 1975 I wrote a *job-stream system* that compiled and executed short Pascal programs input from a card reader and output on a line printer. Input, execution, and output took place simultaneously using buffers stored on a disk. A user job was preempted if its compilation and execution time exceeded 1 minute. I designed, programmed, and tested the system in 10 days. When the system was finished, it ran short jobs continuously at the speed of the line printer (Brinch Hansen 1976a).

Each model operating system was a Concurrent Pascal program of about 1,000 lines of text divided into 15–25 modules. A module was roughly one page of text (50–60 lines) with about 5 procedures of 10–15 lines each (Table 1).

Table 1 Model operating systems.

| | Solo | Job stream | Real time |
|---|---|---|---|
| Lines | 1,300 | 1,400 | 600 |
| Modules | 23 | 24 | 13 |
| Lines/module | 57 | 58 | 46 |
| Procedures/module | 5 | 4 | 4 |
| Lines/procedure | 11 | 15 | 12 |

These examples showed that it was possible to build nontrivial concurrent programs from very simple modules that could be studied page by page (Brinch Hansen 1977).

Compared to assembly language, Concurrent Pascal reduced my programming effort by an order of magnitude and made concurrent programs so simple that a software journal could publish the entire 1,300 lines of the Solo program text (Brinch Hansen 1975c).

I tested the modules of a concurrent program one at a time starting with those that did not depend on other modules. In each test run, the initial process was replaced by a short test process that called the top module and made it execute all its statements at least once. When a module worked, another one was tested on top of it.

Dijkstra had used a similar procedure to test the THE multiprogramming system, which was written in assembly language. However, Concurrent Pas-

cal made bottom-up testing secure. The compilation checks of access rights ensured that new (untested) modules could not make old (tested) modules fail. My experience was that a well-designed concurrent program of one thousand lines required a couple of compilations followed by one test run per module. And then it worked (Brinch Hansen 1977).

⋆ ⋆ ⋆

In his book, "Advice to a Young Scientist," the Nobel Laureate Peter Medawar (1979) wrote:

> Ever since Bacon's day experimentation has been thought to be so deeply and so very necessarily a part of science that exploratory activities that are not experimental are often denied the right to be classified as sciences at all.

Unfortunately, this obvious requirement has often been ignored in academic research on software design. In a guest editorial introducing the Solo papers, I commented on the sad state of my profession (Brinch Hansen 1976c):

> It is not uncommon for a computer scientist to make a proposal without testing whether it is any good in practice. After spending 3 days writing up the monitor proposal and 3 years implementing it, I can very well understand this temptation. It is perhaps also sometimes a human response to the tremendous pressure on university professors to get funding and recognition fast.
>
> Nevertheless, we must remember that only one thing counts in engineering: Does it work?...What would we think of mathematicians if most of their papers contained conjectures only? Sometimes an educated guess can be a great source of inspiration. But we must surely hope that the editors of computer journals will reject most proposals until they have been tried at least experimentally.

There was no doubt in my mind, that it was essential to put monitors to a realistic test before I could recommend them as a proven tool for software engineering. That was the whole purpose of developing Concurrent Pascal and Solo.

In July 1975, I described Concurrent Pascal and Solo at the International Summer School in Marktoberdorf, Germany. After presenting our system, that had been working for three months, I found it odd to hear Tony Hoare present an outline of an *unimplemented* operating system, which would be published in the proceedings of the Summer School (Hoare 1976b).

The first operating system written in Concurrent Pascal (called *Deamy*) was used only to evaluate the expressive power of the language and was never built (Brinch Hansen 1974a). The second one (called *Pilot*) was used for several months but was too slow. They were described in internal working documents only.

In a collection of his best papers, Hoare (1989) wrote:

> The ultimate test of an idea, and the one that deserves the most trust, is when it has been applied successfully in some important project. . .These more substantial tests have always been left to my readers.

He was apparently looking for a "royal road" to software research that would save him from being personally involved in the completion of his "model operating system." It would be another four years before Hoare's coworkers completed their own monitor language, Pascal Plus (Welsh 1979). They never developed an operating system in Pascal Plus capable of compiling and executing real programs. None of this detracts from Hoare's accomplishments as a theoretician. But as a software developer, he was obviously not in the same class as Peter Naur, Edsger Dijkstra, and Niklaus Wirth.

In a paper on programming languages for real-time control, Tony Hoare (1976a) had this to say about Concurrent Pascal:

> This is one of the few successful extensions of Pascal, and includes well structured capabilities for parallel processing, for exclusion and for synchronization. It was tested before publication in the construction of a small operating system, which promises well for its suitability for real-time programming. Although it does not claim to offer a final solution of the problem it tackles, it is an outstanding example of the best of academic research in this area.

At Marktoberdorf, Bill Wulf talked about his Hydra operating system, which used run-time checking of access rights (called "capabilities"). In a summary of the Summer School, Dijkstra (1975) wrote:

> Bill Wulf (Carnegie Mellon) and Per Brinch Hansen (Cal.Tech.) reported both on their development projects (the Hydra system and a pilot model to try out the applicability of Concurrent Pascal, respectively). Both gave eight lectures, and it was a pity that their subject were so similar: sometimes all the details became rather boring and the relative importance of operating system design became overstressed.

I remember thinking, has Dijkstra forgotten that it was his development of a *working* system that gave us confidence in the ideas behind the THE multiprogramming system?

Alas, by 1975 Dijkstra had already formed the dogmatic opinion that programming is a mathematical discipline in which there is no place for concise informal reasoning supported by other means of documentation, such as pictures, "operational" explanations (as he called them), and systematic test cases. He was no longer interested in programs that were too large to be proven mathematically correct. (This pretty much ruled out any program of more than a couple of pages).

However, Niklaus Wirth was not in doubt about what we had done:

> I thank you very much for sending me the two reports on Concurrent Pascal and on the Solo operating system. They are truly encouraging and describe solid engineering progress. This is extremely refreshing after the large heaps of papers that flood the literature and which only present new, abstract ideas, and usually make things more complicated than they were before. May I ask you to kindly send me a second copy of these valuable reports for our library. (Letter from Wirth, October 14, 1975.)

⋆ ⋆ ⋆

I have always regarded research proposals as a distraction from my work. By the time I get a small grant, I have already done so much work, that I hardly need the money! The problem is that funding agencies cannot afford to admit that awarding research grants is like active management of investment funds: program directors like to believe that they are making rational choices, but in reality they just take turns being lucky!

Most professors will never make a major discovery. So why do we grant them tenure? Because we have no way of knowing which ones will make the

fundamental contributions! So we gamble on all of them. And, if only one in ten researchers change their fields, it is still an excellent investment from society's point of view. That's the reality of research. However, if funding agencies respected this fact, they would have to award grants on a random basis. Since it is difficult to acknowledge this inconvenient truth, the charade begins: faculty members make promises, they know they can't keep. And funding agencies shy away from the lone inventor and express a preference for grandiose "multi-disciplinary" research involving several departments (or even universities).

As a new faculty member at Caltech, I had applied for a grant, but was unsuccessful. One anonymous reviewer wrote: "What the world needs is parallel computers, not parallel languages!" When Tony Hoare visited me in January 1974, I asked him "what's wrong with my research proposal?" He looked at it and gave me some worldly advice: "Instead of saying that your ideas are a great improvement over those of professor X, why don't you say: This work builds on the foundation established by professor X." I followed his advice and, in September 1974, was awarded a grant of $71,200 by the National Science Foundation. Of course, by then I had already worked on Concurrent Pascal for two years without external support and would finish the research in another nine months.

I used some of the money to pay McCann for computer time on the PDP 11/45. Towards the end of my project, I decided to cut my computer time in half. The next day, McCann's secretary, Evelyn Johnson, informed me that McCann had just doubled the hourly rate for computing. However, that is the only time I personally felt that McCann misused his power. I am grateful to him for letting me use a minicomputer that made my work accessible thoughout the world.

At Caltech we prepared a distribution tape with the source text and portable code of the Solo system, including the Concurrent and Sequential Pascal compilers. The system reports were supplemented by implementation notes (Brinch Hansen 1976b).

I used part of my grant to hire a secretary, named Barbara. For the job interview, she wore a dress, high heels, and war paint. As soon as she reported to work, she wore jeans, like everybody else. By the spring of 1976, she had distributed the system to 75 companies and 100 universities in 21 countries: Australia, Austria, Belgium, Canada, Denmark, Finland, France, Germany, Great Britain, Holland, India, Ireland, Italy, Japan, Norway, South Africa, the Soviet Union, Spain, Sweden, Switzerland, and the

United States.

We charged around $100 to pay for the expenses of shipping the system tape and manuals. After a while, we had accumulated a small surplus. When I left Caltech, a check for this amount was issued to NSF. In response, the program director wrote: *"You are supposed to spend our money—not return it!"*

⋆ ⋆ ⋆

Shortly after my arrival at Caltech, Robert Cannon replaced Francis Clauser as chairman of engineering and applied science. He would move computer science at Caltech in a new direction and end McCann's dominance of the field.

Bob Cannon wanted to know how Caltech could become unique in computing. So he put together a committee that included Carver Mead (Applied Physics), Herb Keller (Applied Mathematics), John Pierce (former Head of Bell Labs and inventor of the communications satellite), Gilbert McCann and me (Information Science). He met with us regularly and kept asking the same question: "Where is the gold buried in computing?"

At our suggestion, he invited leading computer scientists from other universities to meet with the committee at Caltech. The visitors included John McCarthy (creator of the programming language LISP), Ivan Sutherland (a pioneer in computer graphics), and (at my suggestion) Tony Hoare. Mead and Sutherland hit it off immediately and started traveling around the country asking researchers the same question as Bob Cannon. They came back and told Cannon: "VLSI technology is the future of computing!" Carver Mead predicted that by 2001 transistor sizes would shrink by a factor of 100, and he was absolutely right.

In 1976, Caltech hired Sutherland to lead computer science at Caltech. Only then did Cannon deal with the political problem of McCann: He informed McCann that the Jorgenson Laboratory would now house two separate departments, named Bioinformation Systems (headed by McCann) and Computer Science (headed by Sutherland).

As an untenured faculty member I was now caught in the middle of a power struggle that was beyond my control. Carver Mead would indeed put Caltech at the cutting edge in hardware technology and would eventually receive the National Medal of Technology in a White House ceremony. Unfortunately, as far as I could tell, he had absolutely no appreciation of

modern programming. He believed that programming, as we knew it then, would become superfluous once you could put a million transistors on a chip.

As soon as Mead and Sutherland decided to concentrate on VLSI technology, Caltech was no longer interested in the fundamental ideas of programming explored by Dijkstra, Hoare, Wirth, and me. Under those circumstances, I considered it professional suicide to apply for tenure at Caltech. After five exciting years, I decided to leave the magic kingdom in Pasadena. On April 30, 1976, I submitted my letter of resignation to Bob Cannon (Fig. 6.4).

⋆ ⋆ ⋆

When I left Caltech, I was 38 years old and had just completed some of my best work. In a historical paper (Brinch Hansen 1993), my colleague, Giorgio Ingargiola, described his impression of me at Caltech:

> You had this tremendous clarity about what you were doing in concurrency and languages; you made restrictive choices usually on the basis of efficiency (you list a number of such choices in your paper). You stated something like "start with as few and simple mechanisms as possible; add later only if it becomes necessary."
>
> At least in your discussions and lectures, you built programs from English statements, making explicit the invariants and refining these statements, usually not modifying them, until the program was done.
>
> I was amazed at how slowly you developed code when lecturing, and, by contrast, how fast you got debugged running code for the Concurrent Pascal compiler, and for various concurrent programs and the Solo OS.
>
> You had very little interest in computer science topics outside of the area in which you were doing research. You made polite noises, you indicated interest, but your span of attention was minimal.

My PhD student, Al Hartmann, contributed this amusing portrait:

> There are really two histories interwoven in this paper—the history of the development of concurrent modular programming, and the history of one man's ruthless quest for simplicity in design and programming. The former topic is indifferent to whether

*California Institute of Technology*
*Information Science 286-80*
*Pasadena, California 91125*

*30 April 1976*

*Dr. Robert H. Cannon, Jr.*
*Division Chairman*
*Engineering and Applied Science*
*Caltech 104-44*

*Dear Bob,*

*I have decided to leave Caltech as an Associate Professor of Computer Science on August 31, 1976.*

*Computer Science at Caltech is now changing completely as indeed it should. I am sure that Ivan Sutherland will give Caltech strength in computer applications.*

*To make an outstanding contribution to computer engineering you will, of course also need some of the most creative minds in computer design, programming, and theory.*

*Although the Computer Science Committee initially declined to make offers to three of the most outstanding computer scientists: Tony Hoare, Edsger Dijkstra, and Niklaus Wirth, I hope that you eventually will reconsider this decision. The combination of any one of them and myself would have given Caltech a strength in programming that would have been unequaled anywhere else.*

*I have enjoyed working with Caltech students for the past four years. Together we have developed the first abstract programming language for Concurrent programming. It has now been distributed to about 60 companies and 85 universities throughout the worId.*

*I will be very pleased to serve on the Computer Science Program Committee until I leave campus.*

*Yours sincerely,*

*Per Brinch Hansen*
*Associate Professor of Computer Science*

Figure 6.4 My resignation letter.

one chooses to develop concurrency mechanisms for greater expressive power and more complex functionality, or, as you have chosen, to radically shorten and simplify the design of common concurrent systems. The Solo operating system is downright primitive in the sparseness of its features, representing a countercultural current against ever-increasing operating system complexity. Your style and taste in programming run almost counter to the second law of thermodynamics, that all closed systems tend towards increasing entropy and disorder.

In a world of Brinch Hansens (which may exist in some parallel dimension to ours), all systems tend towards reduced entropy over time and toward a blissful state of ultimate simplicity. Each new release of the operating system for one's personal workstation is smaller than the previous release, consumes fewer system resources, runs faster on simpler hardware, provides a reduced set of easier to use features than the last release, and carries a lower price tag. Hardware designers espousing the same philosophy produce successive single-chip microprocessors with exponentially declining transistor counts from generation to generation, dramatically shrinking die sizes, and reducing process steps by resorting to fewer, simpler device types. No one would need to "invent" RISC computing in this world, since reduced feature sets would be an inexorable law of nature.

The Concurrent Pascal project had a profound influence on Tom Zepko:

Part of the history you describe is an important part of my own history. At the time I was involved with Concurrent Pascal, I was an undergraduate and not so much concerned with the conceptual significance of the language as with learning how to build a language system from the ground up. I got the practical experience I wanted by working on the Concurrent Pascal compiler, the threaded code interpreter, and the operating system kernel. I have continued to do this same kind of work for the last fifteen years.

The concepts behind the Concurrent Pascal, the evolution of the ideas as you describe them, are clearer to me now than they were as a student. The needs you were addressing do require some years of experience to appreciate. But even as a student,

> some things left a lasting impression. What I learned from you, beyond specific programming techniques, is what I can only describe as a passion for clear thinking. This was obvious in the way you approached program design, and it was obviously the driving force behind the design of the Concurrent Pascal language.
>
> Some of the ideas embodied in Concurrent Pascal were radical at the time. That they seem less so now is a tribute to the trailblazing nature of your work. Your approach to programming and to language design now has many advocates. Structured programming, modular design, strong typing, data encapsulation, and so on, are all considered essential elements of modern programming and have found their way into a wide variety of languages. I'm thankful to have played a part in this work.

Although I did not seek tenure at Caltech, I still treasure a Christmas card from my student, Bart Locanthi, that simply said: "*Being my teacher is a tenured position.*"