Editor: **Gerard J. Holzmann**
NASA/JPL
gholzmann@acm.org

The Weakest Link

Gerard J. Holzmann

MOST NONTRIVIAL SOFTWARE applications we use today are multithreaded. This is true not just for mail readers and spreadsheet applications but also for the really important code that keeps everything from Wall Street to the engine control unit in your car running smoothly. Multithreading creates the illusion that a single computer or computer core can do many things at once. All of that's fine until two or more threads of execution need access to a single shared resource, such as a disk, a network port, or your bank account. Then, things get interesting because we need a reliable set of rules, a protocol, for negotiating an orderly sequence of access.

None of this is unique to software engineering. Mutual-exclusion problems appear everywhere in daily life. Just think of cars that share the road and must negotiate access to intersections, or the swarms of customers who all try to be the first to enter a store on Black Friday. Humans are generally good at resolving those problems in real time and usually manage to avoid collisions. But computers must be taught these skills.

Computers are very good at following rules, but we humans aren't necessarily very good at defining them safely. For example, consider the mid-1800s, when railway engineers were trying to figure out safe rules for trains sharing single-track railway lines. Coming up with a basic schedule for nominal train movements across a single-track line isn't all that difficult, of course. The problem is to come up with the right rules for resolving the inevitable conflicts that occur when delays happen and you must deviate from the normal schedule.

The 15-Minute Rule

In 1856, the North Pennsylvania Railroad adopted the following rule for single-track railway lines: any train awaiting the passing of a train traveling in the opposite direction had to wait at least an additional 15 minutes if that train failed to appear. Unfortunately, adherence to this rule led to a fatal accident. On 17 July 1856, a special excursion train was scheduled to carry schoolchildren from Philadelphia to Fort Washington for an annual picnic. The line between those two stations was single-track. The excursion train was scheduled to arrive in Fort Washington at 6 a.m., and the next train in the opposite direction was scheduled to leave that station at 6:15 a.m., so there was a reasonable margin. (More background on this accident is at https://en.wikipedia.org/wiki/Great_Train_Wreck_of_1856.)

As it happened, the special train was delayed and couldn't make it to Fort Washington by 6 a.m. The conductor of that train consulted the delay rule and decided he had enough time to proceed anyway. He believed that the train in the opposite direction would wait an additional 15 minutes before departing, which meant that he only had to get to Fort Washington by 6:30 a.m. Unfortunately, the other conductor reckoned that if the special train didn't arrive within 15 minutes after its scheduled arrival time, it was waiting for him to go first. So he did.

The flaw of the rule was that it didn't specify whether the 15 minutes started at the scheduled arrival time of the special train or the scheduled departure time of the local train. So, the trains collided just outside Fort Washington, killing 60 passengers, most of whom were schoolchildren. The rules were too vague, allowing a fatal race condition to occur.

Figure 1 illustrates the accident scenario with the help of a Marey chart. Marey charts were first clearly

0740-7459/16/$33.00 © 2016 IEEE

described by the French scientist Etienne-Jules Marey (1830–1904), who attributed their invention to a French engineer named Ibry.[1] The Marey chart is a clever way to represent large amounts of information in a compact visual format. Edward Tufte also chose a Marey chart for the cover of his seminal book *The Visual Display of Quantitative Information*.[1]

Marey charts such as the one in Figure 1 visually capture the schedule of trains traveling in opposite directions across long distances. Figure 1 shows just two stations on the horizontal axis; time increases from top to bottom. The slanted lines indicate train movements, with the slope of each line giving an immediate visual clue for the speed of the train. A train standing still at a station, waiting for its scheduled departure time, would appear as a vertical line. The intersection of two lines corresponds to two trains crossing, which on single-track lines would normally be aligned with a station or some place along the tracks with a siding, where trains can pass each other safely.

This last point—that trains should be scheduled to pass each other only where it's safe to do so—might seem obvious. It would also seem unlikely that someone would design a train schedule that violates that rule. But life can be surprising.

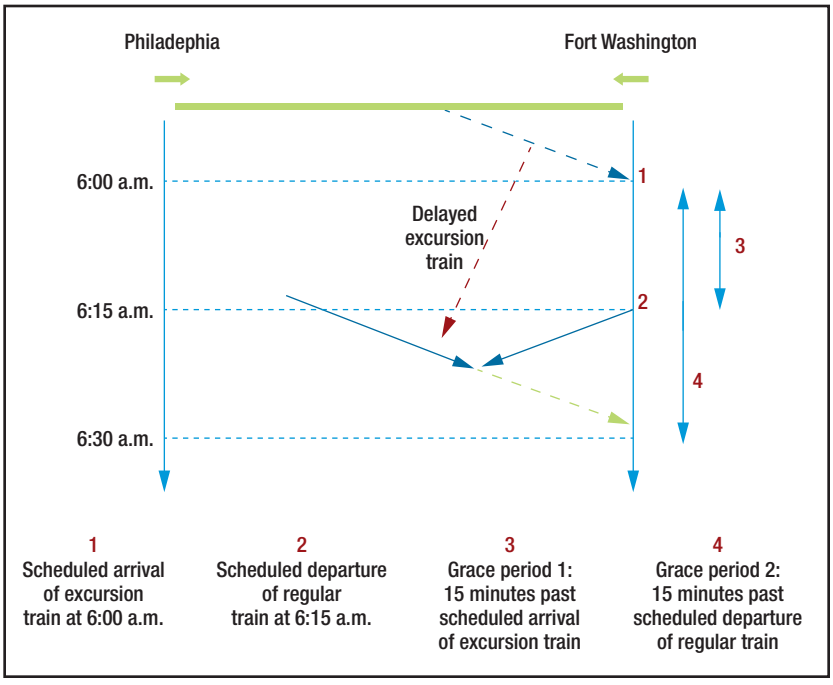

FIGURE 1. A Marey chart illustrating a scheduling problem that led to two trains colliding in 1856. The rules were too vague, allowing a fatal race condition to occur.

Safety First

Two trains collided on 8 June 1975, also on a single-track stretch of railway line, between the stations in Schaftlach and Warngau in Southern Germany. Figure 2 shows the relevant part of the Marey chart for the train schedule for that day. There were no delays, yet it's not hard to see what the cause of the accident was. That day was the second day of a new summer timetable. (On the first day of the schedule, no train went from Schaftlach to Warngau at that specified time, so the problem didn't occur.) Astonishingly, according to this timetable, the two trains were scheduled to cross in the middle of the single-track line segment, which is indeed where they met with a bang. Of course, the train dispatchers at the stations should have noticed the problem in the schedule and prevented the accident, but they didn't. Both the dispatchers and the man who created the schedule were later convicted and had to serve prison sentences.

In software engineering, we know all too well that even very smart people can occasionally make really dumb mistakes. This means that in safety-critical software development, every effort is made to check and double-check that all critical design decisions are sound. This applies especially to resource-sharing algorithms in multithreaded software systems. Critical systems often rely on multiple independent layers of protection, to ensure that the errors that occasionally slip through one filter are caught in the next. Still, as I said before, life can be surprising.

The Human Element

The accident near Warngau led to the adoption of a host of protection mechanisms for train scheduling and signaling, with multiple layers of protection to rule out even the smallest chance of a repeat. And yet, trains still collide on single-track lines. At the time of writing, the most recent such accident happened on 9 February 2016 near Bad Aibling in Germany, less than 20 miles from Warngau. Two trains collided

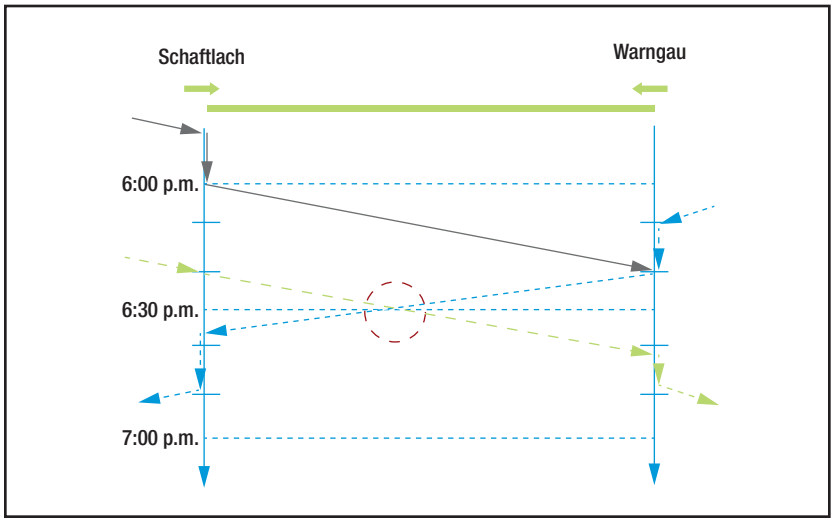

FIGURE 2. The relevant part of the Marey chart for the train schedule for 8 June 1975, when two trains collided between the stations in Schaftlach and Warngau in Southern Germany. The train dispatchers at the stations should have noticed the problem in the schedule and prevented the accident, but they didn't.

at full speed on a blind curve. The protections that had been added after the Warngau accident included a fully automatic system that detects when a single track is in use and forces a train to a stop by engaging its brakes if the train driver fails to heed the warnings. So, how could this new accident have happened?

Midair Mayhem

Before I answer that question, let's look at another collision, this time of two airplanes in midair. By coincidence, this accident also happened in Southern Germany, on 1 July 2001, near Überlingen, about 150 miles from Warngau. Each plane had an automatic TCAS (Traffic Alert and Collision Avoidance System), which detects when planes are on a collision course and automatically alerts the pilot of each plane to make evasive moves.

A Boeing 757 cargo plane heading for Brussels was alerted by its TCAS of an imminent collision with a Tupolev passenger jet heading for Barcelona and was commanded to descend sharply. The pilot did so.

The Tupolev was also alerted by its TCAS, which commanded its pilot to climb to a higher altitude. But, before the pilot could do so, an air traffic controller contacted him. The air traffic controller had failed to sufficiently separate the planes and discovered his mistake at the last moment. In a panic, he instructed the Tupolev to descend. This contradicted the advice the TCAS had given the pilot. The pilot followed the instructions from the air traffic controller, who presumably had more detailed information. The pilot ignored the alarms in the cockpit and descended, leading the plane straight back on a collision course with the Boeing 757, which was also descending. All 71 passengers and crew of the two planes died. The weakest link was the air traffic controller, who had tried to override the automatic protection system.

Let's now go back to the train collision near Bad Aibling. As you might suspect, the accident occurred because one of the dispatchers decided it was safe to overrule the automatic system by allowing a delayed train to enter the single-track line. To activate the override, the dispatcher had to instruct the train driver to ignore alarm bells and hold down a button in his cab to suppress the automatic braking system from halting his train while entering the single-track segment. The dispatcher's mistake cost 11 people their lives, including the drivers of both trains. (You can read more about these types of railway accidents in *Red for Danger: The Classic History of British Railway Disasters*.[2])

Whom Do You Trust?

Over the years, we've gone from accidents that were caused by flawed designs insufficiently backed up by human intelligence to provide for safe overrides, to systems that are safe but allow well-meaning humans to override them in unsafe ways. Are we ready to admit that machines can now be smarter than the average human in preventing accidents?

If you want to win at chess, go, or even *Jeopardy!* you'd be well advised to pick the advice of a computer over the advice of a human, even if that human is the world's best player. The number of possibilities to evaluate to find the best move or answer in any of these games is astoundingly large. In days long past, we might have thought that no machine could possibly match an expertly trained human in these skills. Today, it has become unimaginable that a mere human could match the raw power of a well-designed machine. A great example is IBM's Watson computer, which after its *Jeopardy!* victory is

honing new skills to give medical advice to doctors.[3] This change of mind-set will likely have profound implications for how we design safety-critical systems.

In the 1968 movie *2001: Space Odyssey*, astronaut Dave Bowman tries to get the HAL 9000 computer to follow a voice command. HAL 9000 guesses correctly that Dave's ultimate goal is to shut it down. At some point it responds ominously, "I'm sorry Dave; I'm afraid I can't do that." The computer then explains, "This mission is too important for me to allow you to jeopardize it." When this movie was made, the notion that a computer could overrule a human in judgment was pure science fiction. Today we might be very near the point at which this has become a simple fact of life. Should we be more concerned with a computer overruling a human in key decisions, or vice versa?

References

1. E.R. Tufte, *The Visual Display of Quantitative Information*, Graphics Press, 1983.
2. L.T.C. Rolt, *Red for Danger: The Classic History of British Railway Disasters*, Sutton, 1999.
3. B. Upbin, "IBM's Watson Gets Its First Piece of Business in Healthcare," *Forbes*, 8 Feb. 2013; www.forbes.com/sites/bruceupbin/2013/02/08/ibms-watson-gets-its-first-piece-of-business-in-healthcare/#3da31e6244b1.