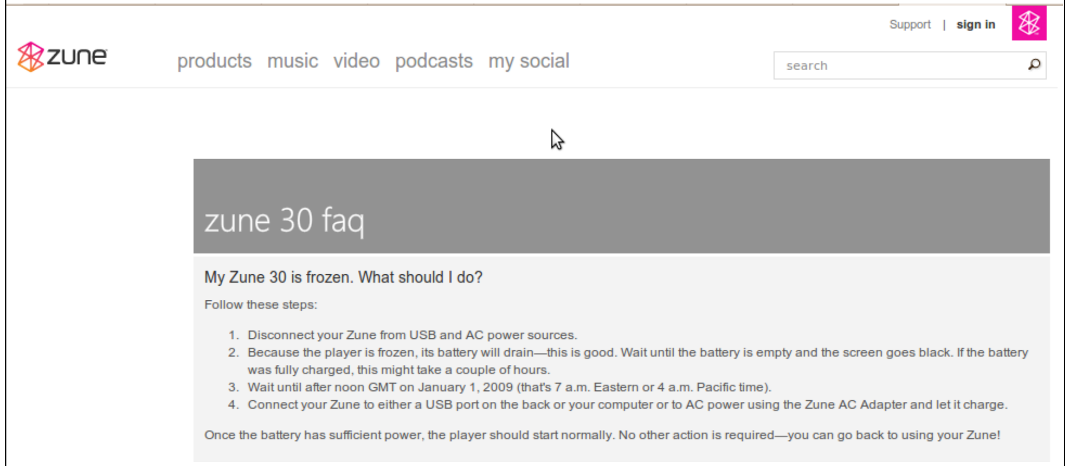
## 30GB Zunes Failing Everywhere, All At Once

2008-12-31, at 12:01 am

Gismodo, Bill Bradski

## http://www.zune.net/en-us/support/zune30.htm



zune 30 faq

My Zune 30 is frozen. What should I do?

Follow these steps:

1. Disconnect your Zune from USB and AC power sources.
2. Because the player is frozen, its battery will drain—this is good. Wait until the battery is empty and the screen goes black. If the battery was fully charged, this might take a couple of hours.
3. Wait until after noon GMT on January 1, 2009 (that's 7 a.m. Eastern or 4 a.m. Pacific time).
4. Connect your Zune to either a USB port on the back or your computer or to AC power using the Zune AC Adapter and let it charge.

Once the battery has sufficient power, the player should start normally. No other action is required—you can go back to using your Zune!

## Microsoft “Official” Report

http://zuneinsider.com/archive/2008/12/31/30gb-zune-issues-official-update.aspx

“Early this morning we were alerted by our customers that there was a widespread issue affecting our 2006 model Zune 30GB devices … . The technical team jumped on the problem immediately and isolated the issue: a bug in the internal clock driver related to the way the device handles a leap year.

The issue should be resolved over the next 24 hours as the time change moves to January 1, 2009. We expect the internal clock on the Zune 30GB devices will automatically reset tomorrow (noon, GMT). By tomorrow you should allow the battery to fully run out of power before the unit can restart successfully then simply ensure that your device is recharged, then turn it back on.

...

We know this has been a big inconvenience to our customers and we are sorry for that, and want to thank them for their patience.”

## Microsoft “Official” Report

http://zuneinsider.com/archive/2008/12/31/30gb-zune-issues-official-update.aspx

“Q: What fixes or patches are you putting in place to resolve this situation?

This situation should remedy itself over the next 24 hours as the time flips to January 1st.

Q: What’s the timeline on a fix?

The issue Zune 30GB customers are experiencing today will self resolve as time changes to January 1.

Q: Why did this occur at precisely 12:01 a.m. on December 31, 2008?

There is a bug in the internal clock driver causing the 30GB device to improperly handle the last day of a leap year.

Q: What is Zune doing to fix this issue?
The issue should resolve itself.”

## Source code for the Zune's clock driver (http://pastie.org/349916)

```
//
// Copyright (c) Microsoft Corporation.  All rights reserved.
//
//
// Use of this source code is subject to the terms of the Microsoft end-user
// license agreement (EULA) under which you licensed this SOFTWARE PRODUCT.
// If you did not accept the terms of the EULA, you are not authorized to use
// this source code. For a copy of the EULA, please see the LICENSE.RTF on your
// install media.
//
//------------------------------------------------------------------------------
//
//  Copyright (C) 2004-2007, Freescale Semiconductor, Inc. All Rights Reserved.
//  THIS SOURCE CODE, AND ITS USE AND DISTRIBUTION, IS SUBJECT TO THE TERMS
//  AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT
//
//------------------------------------------------------------------------------
//
//  Module: rtc.c
//
//  PQOAL Real-time clock (RTC) routines for the MC13783 PMIC RTC.
//
//------------------------------------------------------------------------------
```

## Zune's Clock Driver

http://www.zuneboards.com/forums/zune-news/38143-cause-zune-30-leapyear-problem-isolated.html

The Zune's real-time clock stores the time in terms of days and seconds since January 1st, 1980.

When the Zune's clock is accessed, the driver turns the number of days into years/months/days and the number of seconds into hours/minutes/seconds. Likewise, when the clock is set, the driver does the opposite.

The Zune frontend first accesses the clock toward the end of the boot sequence. Doing this triggers the code that reads the clock and converts it to a date and time.

## Source code for the Zune's clock driver (http://pastie.org/349916)

```
...
// Global Variables
//These macro define some default information of RTC
#define ORIGINYEAR       1980                  // the begin year
#define MAXYEAR          (ORIGINYEAR + 100)    // the maxium year
#define JAN1WEEK         2                     // Jan 1 1980 is a Tuesday
#define GetDayOfWeek(X)  (((X-1)+JAN1WEEK)%7)
...
//------------------------------------------------------------------------------
static int IsLeapYear(int Year)
{
    int Leap;

    Leap = 0;
    if ((Year % 4) == 0) {
        Leap = 1;
        if ((Year % 100) == 0) {
            Leap = (Year%400) ? 0 : 1;
        }
    }

    return (Leap);
}
```

## Source code for the Zune's clock driver (http://pastie.org/349916)

```
...
BOOL ConvertDays(UINT32 days, SYSTEMTIME* lpTime)
{
...
...
    year = ORIGINYEAR;

    while (days > 365)
    {
        if (IsLeapYear(year))
        {
            if (days > 366)
            {
                days -= 366;
                year += 1;
            }
        }
        else
        {
            days -= 365;
            year += 1;
        }
    }
...
```

Where is the error?

## The AT&T Network Crash of 1990

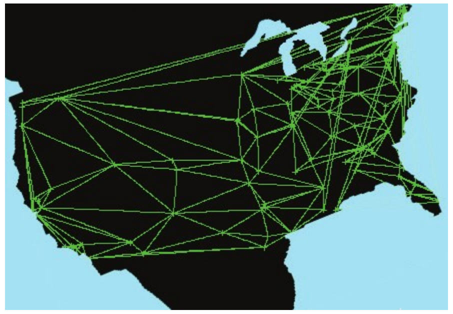
On January, 15 1990 more than half of the AT&T network crashed.

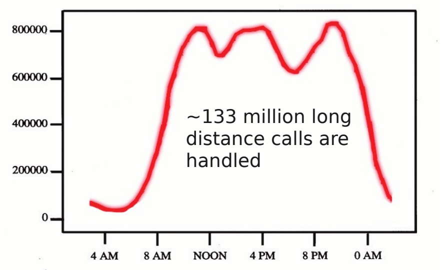
More than 75 million were unanswered during the 9 hours as AT&T attempted to fix and find the problem.

Despite the beliefs of law enforcement and telco security that hackers were responsible for the crash, the real problem was an embarrassment for AT&T during a time of growing competition.

The cause was a bug in a routine software update of it 4ESS systems in December. Something happened to the particular switch (Switch A) that caused the disaster and it sent out a broadcast declaring it was no longer accepting any new messages. Once Switch A was back up, it sent out another broadcast telling the other switches that it's back online. That's how it's supposed to work. However, the bug was, while one of the switches (Switch B) was still resetting itself to acknowledge Switch A being back in service, Switch A sent this one another message, confusing it. Switch B went down with the same problem, and it resulted in a chain reaction. As the network continued to repeat this problem.

## A normal day in the life of a telephone system (Wednesday April 5, 1995)

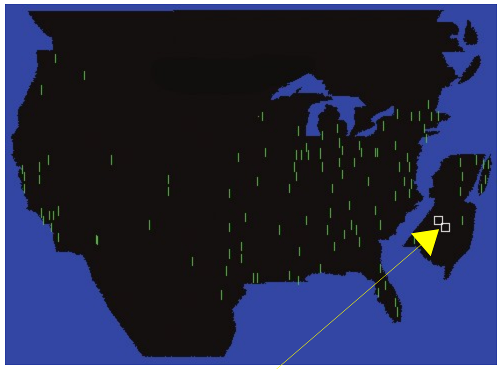




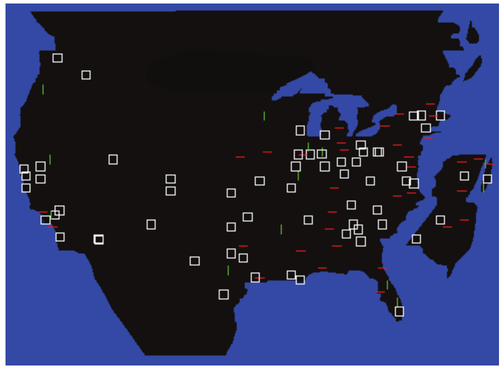
CS118, Lecture 1, slide 10

## A not so ordinary day: Monday January 15, 1990 – "the software glitch day"

1:23 pm



1:43 pm



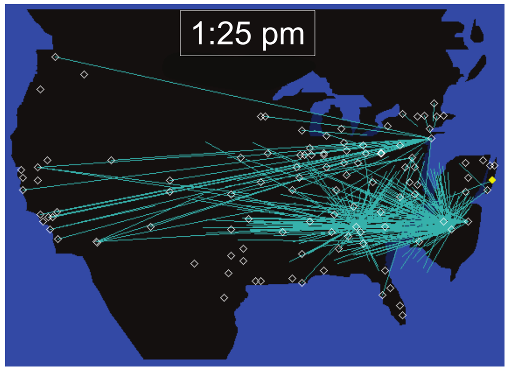
2 switches crash and reboot

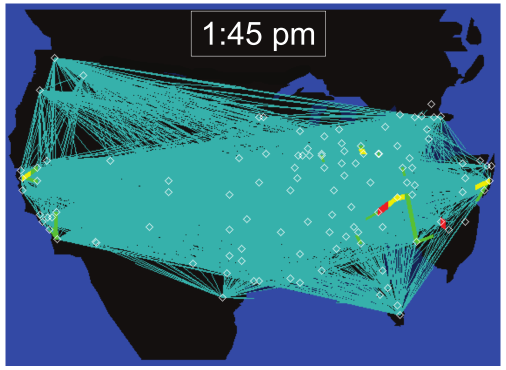
requirement for telephone switches:
≤ 15 seconds down-time per month

green - switch is up
white - switch is down
red - switch unreachable

CS118, Lecture 1, slide 11

## Monday January 15, 1990 (MLK Day) (green: blocked long distance links)





```
for (;;) {
  switch (x) {
  case A: …; break;
  case B: …; break;
  }
  …
}
```

```
for (;;) {
  if (x == A) {
    …;
  } else {
    …; break;
  } …
}
```

CS118, Lecture 1, slide 12

## The AT&T Network Crash of 1990

AT&T solved the problem by reducing the messaging load of the network. That allowed the switches to rest themselves and the network to stabilize.

Despite AT&T's press release for the crash, it gave a much stronger need to execute Operation Sun Devil, the nationwide crackdown on hackers in the early 1990s.

(http://everything2.com/title/AT%2526T+Crash+of+1990)

## A replica of the code for 4ESS Central Office Switching System (http://www.gowrikumar.com/blog/2009/01/06/att-break-bug/)

```
network code()
{
  switch (line) {
    case THING1:
      doit1();
      break;
    case THING2:
      if (x == STUFF) {
        do_first_stuff();
        if (y == OTHER_STUFF)
          break;
        do_later_stuff();
      } /* coder meant to break to here... */
      initialize_modes_pointer();
      break;
    default:
      processing();
  } /* ...but actually broke to here! */
  use_modes_pointer();/* leaving the modes_pointer uninitialized */
```

## From “THE HACKER CRACKDOWN: Law and Disorder on the Electronic Frontier” by Bruce Sterling (bruces@well.sf.ca.us) (1/9)

http://www.farcaster.com/sterling/part1.htm

PART ONE: Crashing the System

On January 15, 1990, AT&T's long-distance telephone switching system crashed.

This was a strange, dire, huge event. Sixty thousand people lost their telephone service completely. During the nine long hours of frantic effort that it took to restore service, some seventy million telephone calls went uncompleted.

Losses of service, known as "outages" in the telco trade, are a known and accepted hazard of the telephone business. Hurricanes hit, and phone cables get snapped by the thousands. Earthquakes wrench through buried fiber-optic lines. Switching stations catch fire and burn to the ground. These things do happen. There are contingency plans for them, and decades of experience in dealing with them. But the Crash of January 15 was unprecedented. It was unbelievably huge, and it occurred for no apparent physical reason.

## From “THE HACKER CRACKDOWN: Law and Disorder on the Electronic Frontier” by Bruce Sterling (bruces@well.sf.ca.us) (2/9)

http://www.farcaster.com/sterling/part1.htm

The crash started on a Monday afternoon in a single switching-station in Manhattan. But, unlike any merely physical damage, it spread and spread. Station after station across America collapsed in a chain reaction, until fully half of AT&T's network had gone haywire and the remaining half was hard-put to handle the overflow. Within nine hours, AT&T software engineers more or less understood what had caused the crash. Replicating the problem exactly, poring over software line by line, took them a couple of weeks. But because it was hard to understand technically, the full truth of the matter and its implications were not widely and thoroughly aired and explained. The root cause of the crash remained obscure, surrounded by rumor and fear.

The crash was a grave corporate embarrassment. The "culprit" was a bug in AT&T's own software -- not the sort of admission the telecommunications giant wanted to make, especially in the face of increasing competition. Still, the truth *was* told, in the baffling technical terms necessary to explain it.

## From “THE HACKER CRACKDOWN: Law and Disorder on the Electronic Frontier” by Bruce Sterling (bruces@well.sf.ca.us) (3/9)

http://www.farcaster.com/sterling/part1.htm

This is how the problem manifested itself from the realm of programming into the realm of real life. The System 7 software for AT&T's 4ESS switching station, the "Generic 44E14 Central Office Switch Software," had been extensively tested, and was considered very stable. By the end of 1989, eighty of AT&T's switching systems nationwide had been programmed with the new software. Cautiously, thirty-four stations were left to run the slower, less-capable System 6, because AT&T suspected there might be shakedown problems with the new and unprecedently sophisticated System 7 network.

The stations with System 7 were programmed to switch over to a backup net in case of any problems. In mid-December 1989, however, a new high-velocity, high-security software patch was distributed to each of the 4ESS switches that would enable them to switch over even more quickly, making the System 7 network that much more secure. Unfortunately, every one of these 4ESS switches was now in possession of a small but deadly flaw.

## From “THE HACKER CRACKDOWN: Law and Disorder on the Electronic Frontier” by Bruce Sterling (bruces@well.sf.ca.us) (4/9)

http://www.farcaster.com/sterling/part1.htm

In order to maintain the network, switches must monitor the condition of other switches -- whether they are up and running, whether they have temporarily shut down, whether they are overloaded and in need of assistance, and so forth. The new software helped control this bookkeeping function by monitoring the status calls from other switches. It only takes four to six seconds for a troubled 4ESS switch to rid itself of all its calls, drop everything temporarily, and re-boot its software from scratch. Starting over from scratch will generally rid the switch of any software problems that may have developed in the course of running the system. Bugs that arise will be simply wiped out by this process. It is a clever idea. This process of automatically re-booting from scratch is known as the "normal fault recovery routine." Since AT&T's software is in fact exceptionally stable, systems rarely have to go into "fault recovery" in the first place; but AT&T has always boasted of its "real world" reliability, and this tactic is a belt-and-suspenders routine.

## From “THE HACKER CRACKDOWN: Law and Disorder on the Electronic Frontier” by Bruce Sterling (bruces@well.sf.ca.us) (5/9)

http://www.farcaster.com/sterling/part1.htm

The 4ESS switch used its new software to monitor its fellow switches as they recovered from faults. As other switches came back on line after recovery, they would send their "OK" signals to the switch. The switch would make a little note to that effect in its "status map," recognizing that the fellow switch was back and ready to go, and should be sent some calls and put back to regular work. Unfortunately, while it was busy bookkeeping with the status map, the tiny flaw in the brand-new software came into play. The flaw caused the 4ESS switch to interacted, subtly but drastically, with incoming telephone calls from human users. If -- and only if -- two incoming phone-calls happened to hit the switch within a hundredth of a second, then a small patch of data would be garbled by the flaw. But the switch had been programmed to monitor itself constantly for any possible damage to its data. When the switch perceived that its data had been somehow garbled, then it too would go down, for swift repairs to its software. It would signal its fellow switches not to send any more work. It would go into the fault-recovery mode for four to six seconds. And then the switch would be fine again, and would send out its "OK, ready for work" signal.

## From “THE HACKER CRACKDOWN: Law and Disorder on the Electronic Frontier” by Bruce Sterling (bruces@well.sf.ca.us) (6/9)

http://www.farcaster.com/sterling/part1.htm

However, the "OK, ready for work" signal was the *very thing that had caused the switch to go down in the first place.* And *all* the System 7 switches had the same flaw in their status-map software. As soon as they stopped to make the bookkeeping note that their fellow switch was "OK," then they too would become vulnerable to the slight chance that two phone-calls would hit them within a hundredth of a second.

At approximately 2:25 p.m. EST on Monday, January 15, one of AT&T's 4ESS toll switching systems in New York City had an actual, legitimate, minor problem. It went into fault recovery routines, announced "I'm going down," then announced, "I'm back, I'm OK." And this cheery message then blasted throughout the network to many of its fellow 4ESS switches.

## From “THE HACKER CRACKDOWN: Law and Disorder on the Electronic Frontier” by Bruce Sterling (bruces@well.sf.ca.us) (7/9)

http://www.farcaster.com/sterling/part1.htm

Many of the switches, at first, completely escaped trouble. These lucky switches were not hit by the coincidence of two phone calls within a hundredth of a second. Their software did not fail -- at first.

But three switches -- in Atlanta, St. Louis, and Detroit -- were unlucky, and were caught with their hands full. And they went down. And they came back up, almost immediately. And they too began to broadcast the lethal message that they, too, were "OK" again, activating the lurking software bug in yet other switches.

As more and more switches did have that bit of bad luck and collapsed, the call-traffic became more and more densely packed in the remaining switches, which were groaning to keep up with the load. And of course, as the calls became more densely packed, the switches were *much more likely* to be hit twice within a hundredth of a second.

## From “THE HACKER CRACKDOWN: Law and Disorder on the Electronic Frontier” by Bruce Sterling (bruces@well.sf.ca.us) (8/9)

http://www.farcaster.com/sterling/part1.htm

It only took four seconds for a switch to get well. There was no *physical* damage of any kind to the switches, after all. Physically, they were working perfectly. This situation was "only" a software problem.

But the 4ESS switches were leaping up and down every four to six seconds, in a virulent spreading wave all over America, in utter, manic, mechanical stupidity. They kept *knocking* one another down with their contagious "OK" messages.

It took about ten minutes for the chain reaction to cripple the network. Even then, switches would periodically luck-out and manage to resume their normal work. Many calls -- millions of them -- were managing to get through. But millions weren't.

The switching stations that used System 6 were not directly affected. Thanks to these old-fashioned switches, AT&T's national system avoided complete collapse. This fact also made it clear to engineers that System 7 was at fault.

## From “THE HACKER CRACKDOWN: Law and Disorder on the Electronic Frontier” by Bruce Sterling (bruces@well.sf.ca.us) (9/9)

http://www.farcaster.com/sterling/part1.htm

Bell Labs engineers, working feverishly in New Jersey, Illinois, and Ohio, first tried their entire repertoire of standard network remedies on the malfunctioning System 7. None of the remedies worked, of course, because nothing like this had ever happened to any phone system before.

By cutting out the backup safety network entirely, they were able to reduce the frenzy of "OK" messages by about half. The system then began to recover, as the chain reaction slowed. By 11:30 pm on Monday January 15, sweating engineers on the midnight shift breathed a sigh of relief as the last switch cleared-up.

By Tuesday they were pulling all the brand-new 4ESS software and replacing it with an earlier version of System 7.

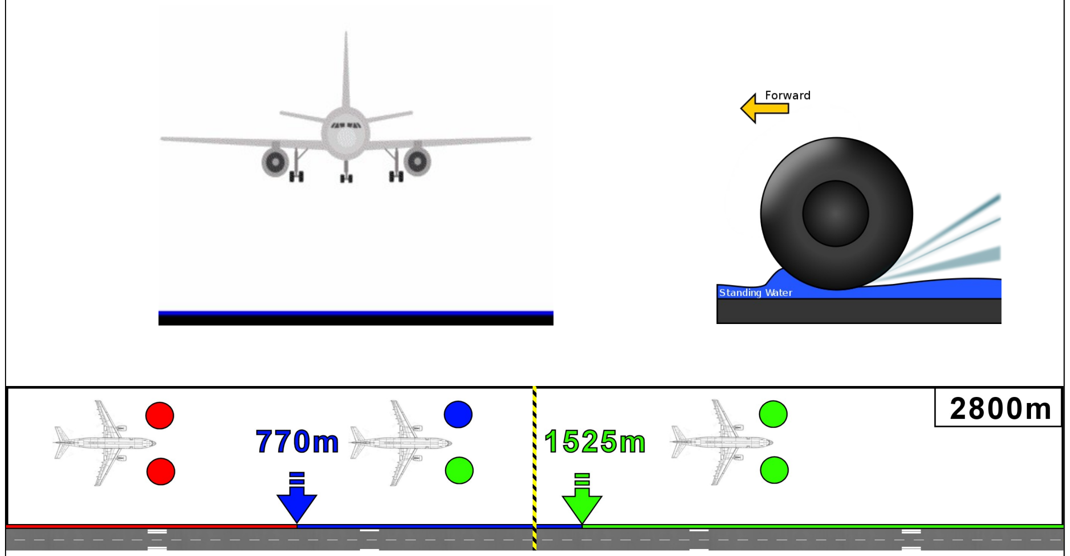
## Lufthansa Flight 2904, Airbus 320-211, 14 September 1993 (1/3)

**Details about the design features of the aircraft**

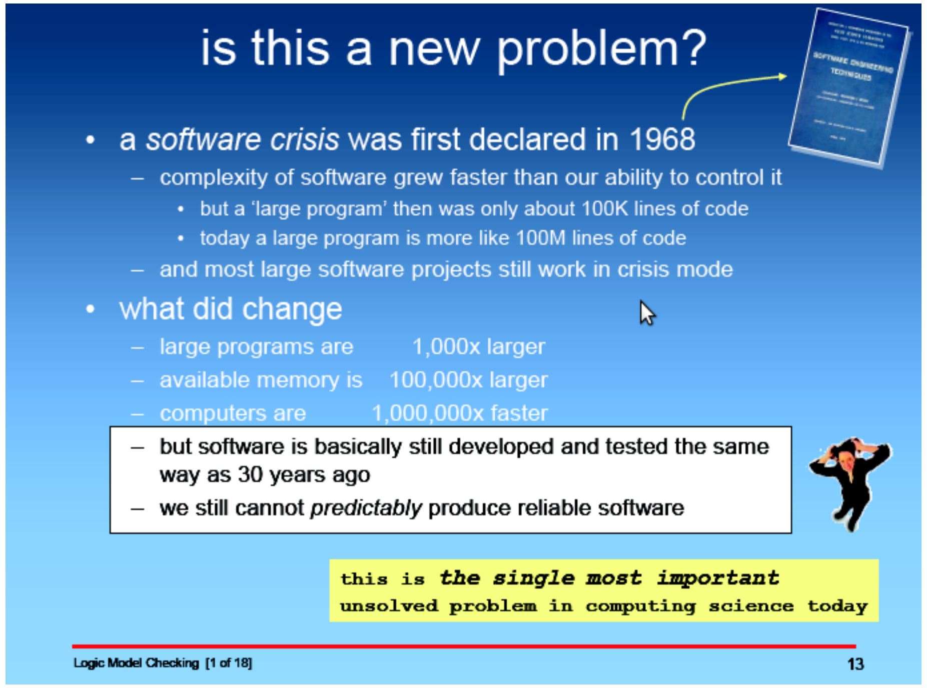
To ensure that the thrust-reverse system and the spoilers are only activated in a landing situation, all of the following conditions have to be true for the software to deploy these systems:

- there must be weight of over 12 tons on each main landing gear strut
- the wheels of the plane must be turning faster than 72 knots (133 km/h)
- the thrust levers must be in the idle (or reverse thrust) position

## Lufthansa Flight 2904, Airbus 320-211, 14 September 1993 (2/3)



## Lufthansa Flight 2904, Airbus 320-211, 14 September 1993 (3/3)



## is this a new problem?

- a *software crisis* was first declared in 1968
  - complexity of software grew faster than our ability to control it
    - but a 'large program' then was only about 100K lines of code
    - today a large program is more like 100M lines of code
  - and most large software projects still work in crisis mode
- what did change
  - large programs are 1,000x larger
  - available memory is 100,000x larger
  - computers are 1,000,000x faster
  - but software is basically still developed and tested the same way as 30 years ago
  - we still cannot *predictably* produce reliable software

**this is *the single most important* unsolved problem in computing science today**

Logic Model Checking [1 of 18] 13

CS118, Lecture 1, slide 13