Risks of Computers

“Anything that can go wrong, will.”

As in so many other things, computers intensify the effect. . .

Why

- Speed
- Complexity
- Access
- Arrogance
- Excessive trust

Speed

- Today, things can go wrong at multigigahertz speeds
- Multicore makes them go wrong even faster
- Often too fast for human review or intervention

Complexity

- We generally do not have full understanding of our systems
- There are often unforeseen interactions
- It is rarely possible to test thoroughly

Access

- Many people have remote access to online machines (i.e., most of them), far more than would have access to corresponding non-computerized systems
- Often, the access is not intended by the designers. . .
- There are often inadquate logs

Arrogance

- Designers and programmers think they can build arbitrarily complex — but correct — systems
- They also think they can do it on time and under budget
- Purchasers believe them

Trust

- People *trust* computer output
- “Garbage in, gospel out”

Interactions

- Programs interact internally
- Systems of programs interact with each other
- Programs interact with users
- Programs interact with the environment
- All of these interactions interact with each other!

Users

- Users do *strange* things, things unanticipated by the designer
- There may be surrounding features unknown to the designer, different styles of operation, etc.
- There are often subtle timing issues

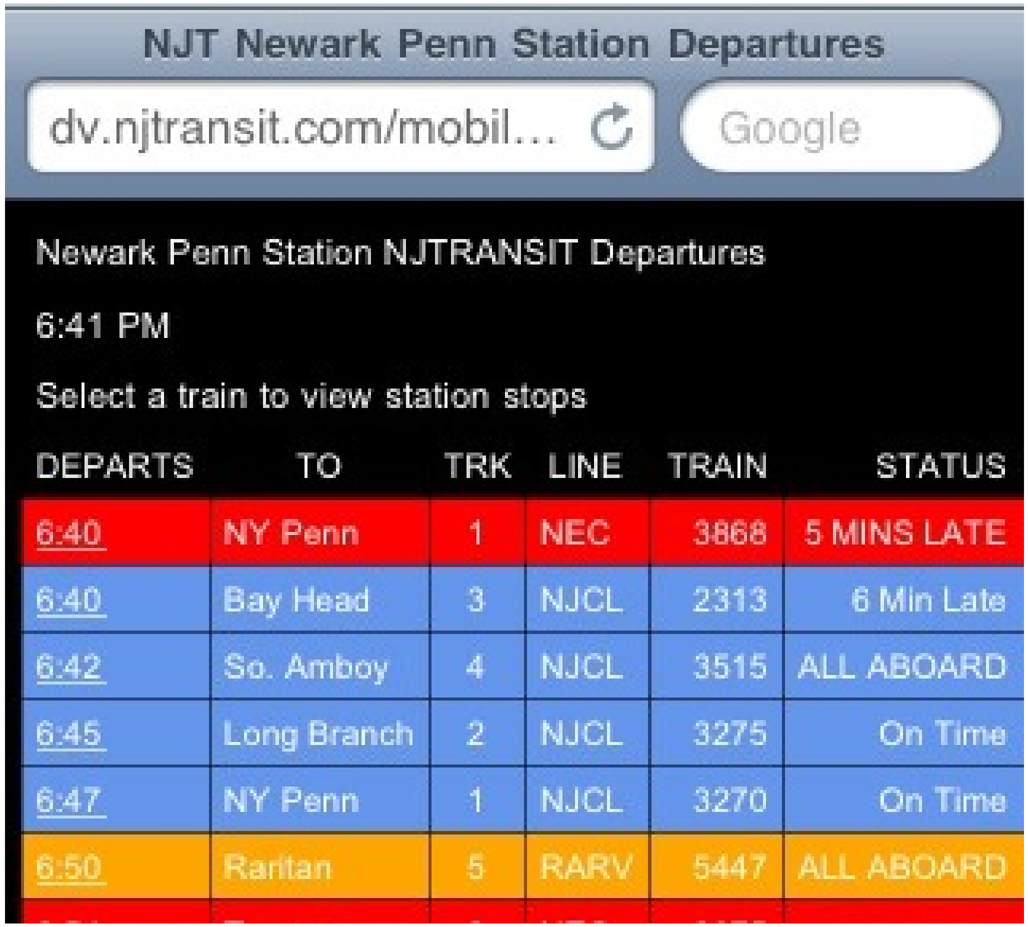
The Environment

- Differences in OS version, configuration, etc., are problematic
- Physical interactions — RF energy, cosmic rays, alpha particles, voltage fluctuations — can lead to failures
- Unanticipated constraints — RAM size, CPU speed, free disk space, etc. — can cause trouble

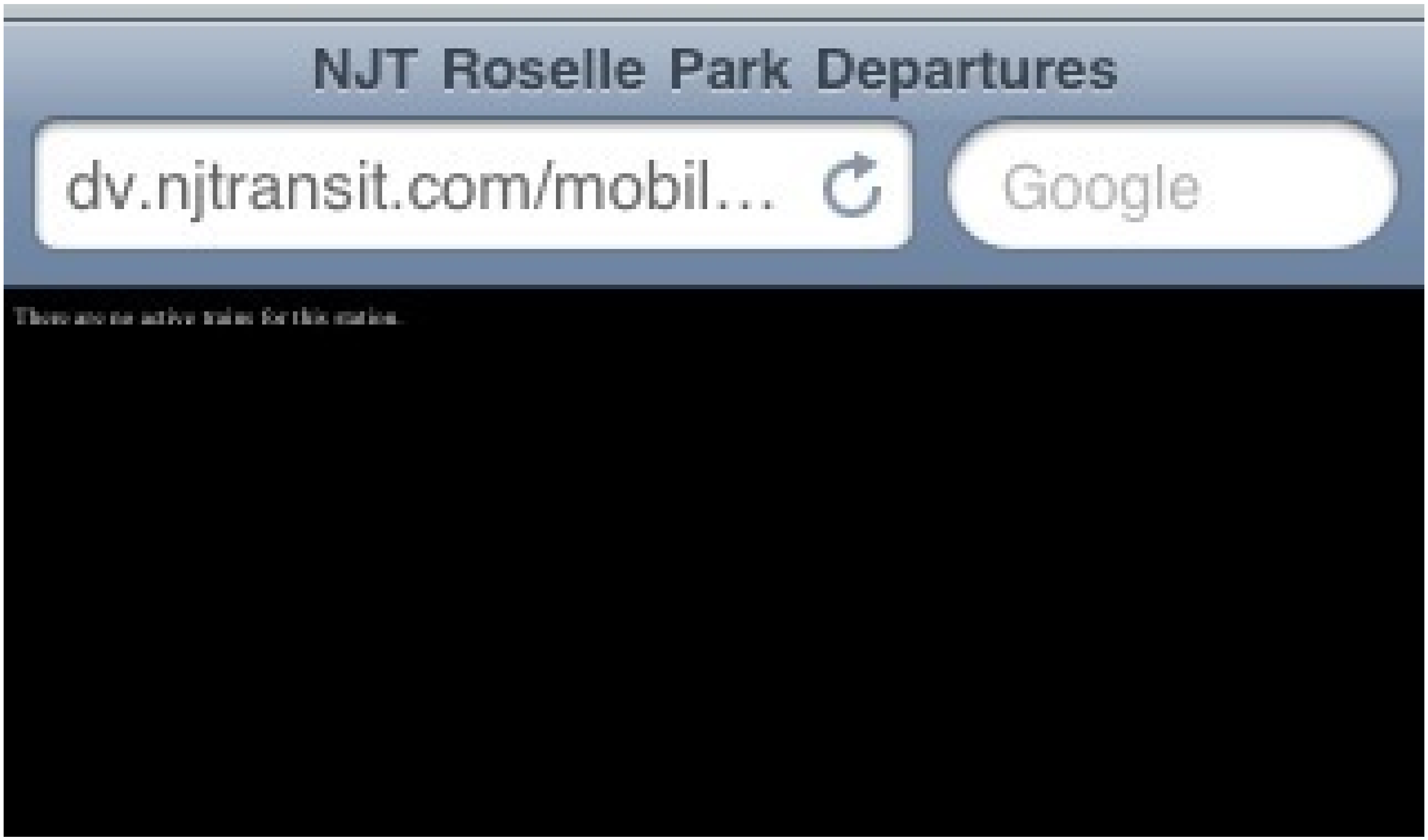
Error Handling

- Programmers often do a poor job handling errors
- “That can’t happen”. . .
- Is it detected? What is the response? To panic? To die a horrible death? To recover?
- How do you test, especially if it’s a hardware failure indication
- Sometimes, misbehaving hardware *really* misbehaves

NJ Transit: It Should Look Like This



Recently, I Saw This. . .



Bugs Happen

- It's hard to test for all possible failure conditions
- Many problems are caused by combinations of bugs
- Complex systems fail for complex reasons

Example: 2003 Northeast Blackout

- Multiple causes!
- The operators didn't fully understand their system
- The monitoring computers failed
- Power lines failed — and as some failed, other had to carry more of the load, so they heated up and sagged.
- It was a warm day and the wind died, so there was less cooling; this made them sag more — until one touched a tree
- Poor real-time data caused a cascade. . .

The Computer Failure

- The primary alarm server failed, because the alarm application failed and/or because of too much data queued for remote terminals
- The system properly failed over to the backup server
- But — the alarm application moved its data to the backup server, so it crashed, too…

Reliability is Hard

- Sometimes, critical systems are engineered for robustness
- Adding such features adds complexity
- This in turn can cause other kinds of failures

Example: the Space Shuttle

- The shuttle had four identical computers for hardware reliability, plus another running different code
- The four were designed to have no single point of failure — which meant that they couldn't share *any* hardware
- A voting circuit matched the outputs of the primary computers; the crew could manually switch to the backup computers
- But — a common clock would violate the "no single point of failure rule"...

What Time is It?

- In a hard real-time system like space shuttle avionics, *something* will always happen very soon
- Take the value of the first element in the timer queue as “now”
- However, there must be a special case for system initialization, when the queue is empty
- A change to a bus initialization routine meant that $1/67$ of the time, the queue wouldn’t be empty during certain crucial boot-time processing
- This in turn made it impossible for the back-up computer to synchronize with the four primaries
- They scrubbed the very first launch, before a world-wide live TV audience, due to a software glitch

Example: the Phone System

- In January 1990, a processor on an AT&T phone switch failed
- During recovery, the switch took itself out of service
- When it came back up, it announced its status, which triggered a bug in *neighboring switches'* processors
- If those processors received two call attempts within 1/100th of a second, they'd crash, causing a switch to the backup processor
- If the backup received two quick call attempts, *it* would crash
- When those processors rebooted, they'd announce that to their neighbors. . .
- The root cause: a misplaced **`break`** statement
- The failure was a *systems* failure

N-Version Programming

- Common assumption: have different programmers write independent versions; overall reliability should increase
- But — bugs are correlated
- Sometimes, the specification is faulty
- Other times, common misperceptions or misunderstandings will cause different programmers to make the same mistake

Achieving Reliability

- All that said, there are some very reliable systems
- Indeed, the space shuttle's software has been widely praised
- The phone system almost always works (though of course not always)
- How?

The Phone System

- A 1996 study showed four roughly-equal causes of phone switch outage: hardware error, software error, operator error, and miscellaneous
- The hardware was already ultrareliable — how did they get the software failure rate that low?
- A lot of hard work and good design — which is reflected in the wording of the question

Switch Design

- Primary goal: keep the *switch* working at all times
- No single call is important
- If anything appears wrong with a call, blow it away
- The caller will mutter, but retry — and the state in the phone switch will be so different that the retry will succeed
- Plus — lots of error-checking, roll-back, restart, etc.
- All of this is *hideously* expensive

Conclusion

- We are building — and relying on — increasingly complex systems
- We do not always understand the interactions
- The very best systems are very, very expensive — and even they fail on occasion