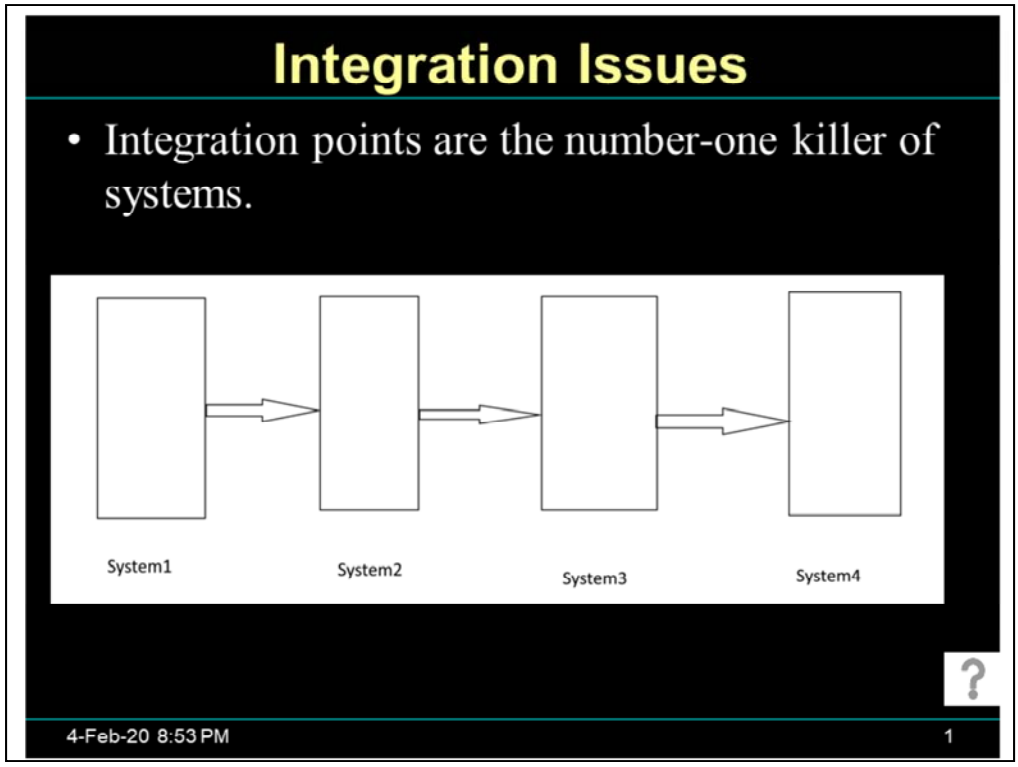


Circuit breaker code is SCircuitBreaker. Additional assignment is present in CircuitBreaker directory.

For Java different ways of implementing a Circuit Breaker is present in proxy.circuitBreaker of DesignPatternsParticipants Project.

For C# Circuit Breaker is present in proxy of DesignPatternsParticipants Project.

Connection timeouts for TCP/IP is in minutes, while creating a new connection. So the thread remains block. All threads in a pool can get blocked.

That is why I advocate supplementing internal monitors (such as log file scraping, process monitoring, and port monitoring) with external monitoring. A mock client somewhere (not in the same data center) can run synthetic transactions on a regular basis. That client experiences the same view of the system that real users experience. When that client cannot process the synthetic transactions, then there is a problem, whether or not the server process is running.

Blocked Threads are a major source of gradual slowdown and hung server. Use timeouts for sockets and library calls that block e.g. in java.util.concurrent.*

Use Circuit breaker e.g. After load crosses a certain limit, refuse more connections. E.g. If your website cannot handle high volume, then detect this high volume surge

and give the users a polite message to come back later.

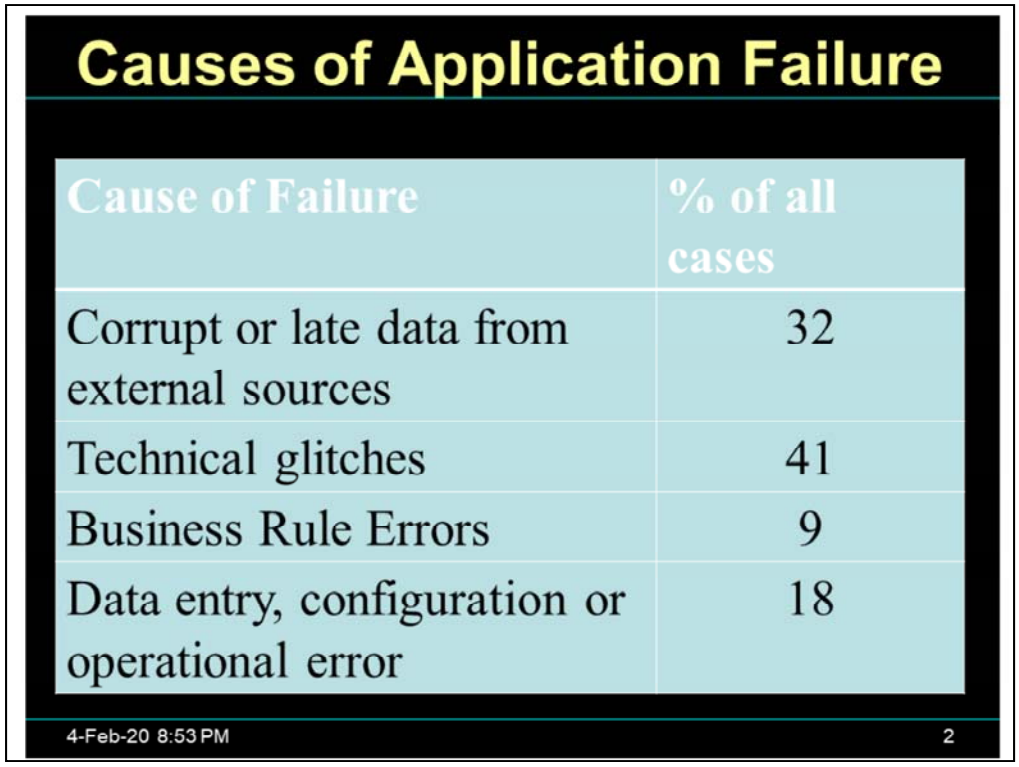
Integration Points:

Every single one of those feeds presents a stability risk.

Every socket, process, pipe, or remote procedure call can and will hang.

Even database calls can hang, in ways obvious and subtle.

Every feed into the system can hang it, crash it, or generate other impulses at the worst possible time.

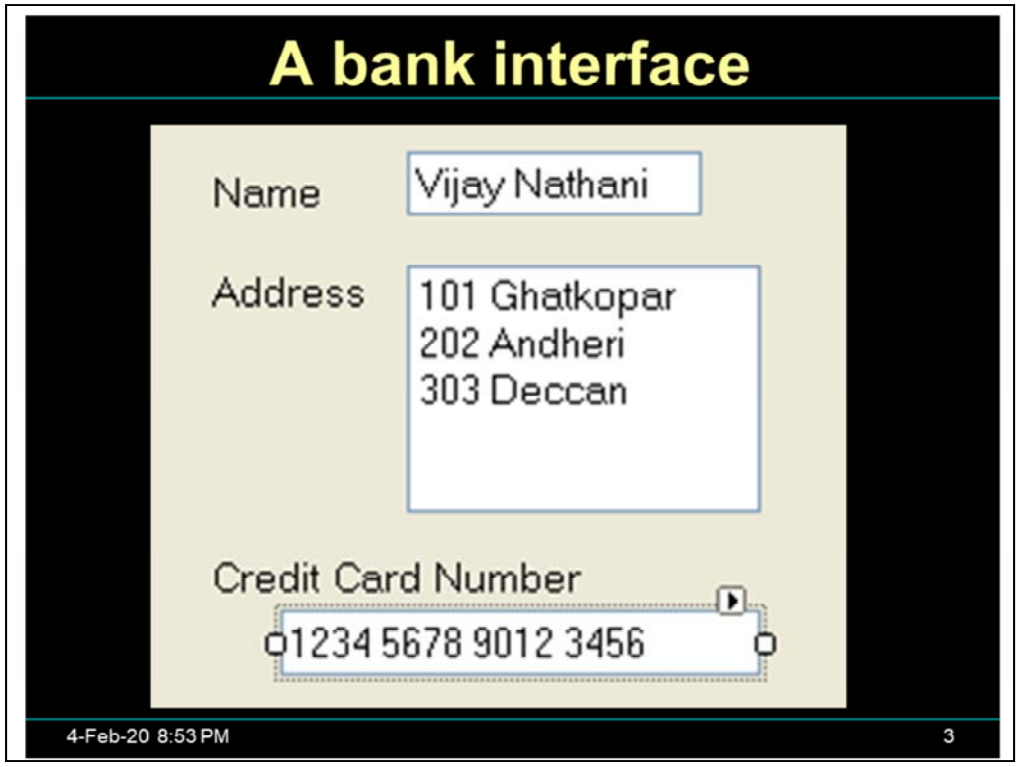


Source: **High-Assurance Design: Architecting Secure and Reliable Enterprise Applications** By Clifford J. Berg

Most failures resulted from various and sundry technical "glitches" situations where the software designer might have said, "That is unlikely to happen," and the support staff later said, "We'll never be able to reproduce that." This category also includes those errors that cannot be explained by the technical staff. E.g. Virus, SQL Injection, Denial of Service, Sniffing, Man in the middle, etc.

Developers and Testers tend to concentrate on "Business Rule Errors"

Operational error: An intern in hospital moves about many departments in his/her few months of work at hospital. By the end of these few months, he/she has more rights on the system than many permanent doctors. People call when they don't have access rights. Nobody calls when access rights have to be removed.



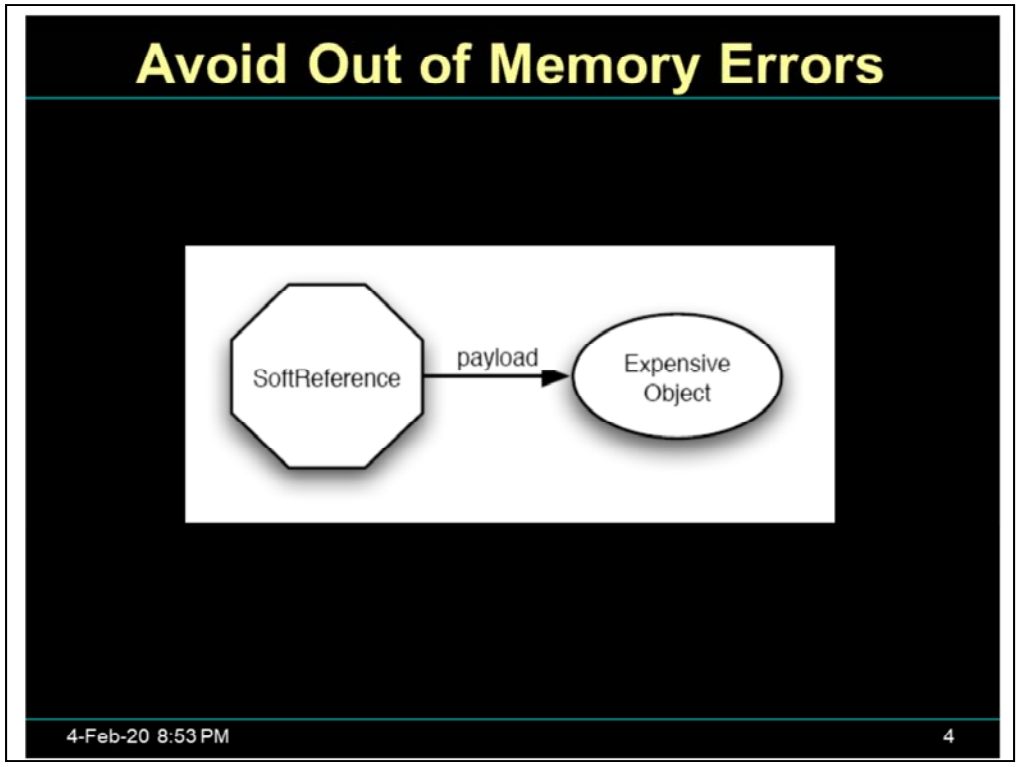
A banking representative asks:

1) Name
2) Give me your address
3) Give me last four digits of your credit card number.

The person is getting all the data. He is transient, located in a remote area and probably lowly paid. This is high risk.

In a better design, he should

1) Enter the last four digits and the system should say match or not.
2) He should enter at least a part of address e.g. house number.



Check for all required resources at the start of application.

•empty disk space

•amount of RAM

•Each new user session supports cookies.

Automate everything. Anything Manual is error prone e.g. Administrator will delete old logs.

Consider periodic health check.