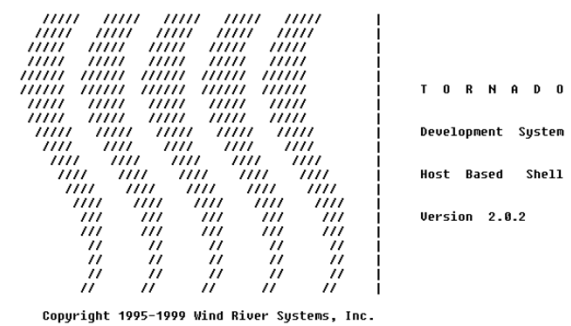
UNIVERSITY OF OSLO

FYS 4220 – 2011 / #11

Real Time and Embedded Data Systems and Computing

Software Quality Assurance – Availability – Safety - Security



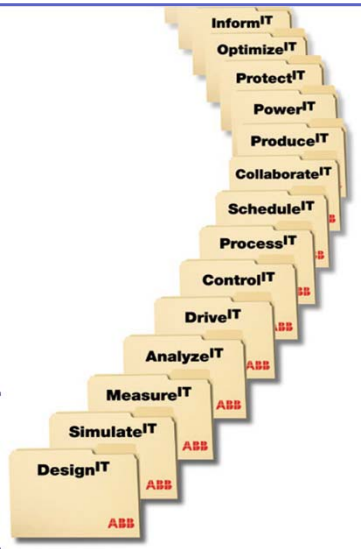
T.B. Skaali, Department of Physics, University of Oslo)

Reference:

MOST OF THIS NOTE IS BASED ON A LECTURE GIVEN BY SVEIN JOHANNESEN, ABB, IN 2009

Svein Johannessen

SQA and Security



© ABB Ltd. - 3

What I am going to talk about

- Why Software Quality Assurance helps avoiding security incidents
- Code examples of potential security holes
- SQA requirements for avoiding security incidents
  - I will assume that you already have a working SQA regime
- Reflections on how to avoid the next security scare

© Copyright year ABB.
All rights reserved.

So, what has SQA to do with Security?

- Embedded software is almost exclusively written in the “C” language which is a high risk language
  - “C” was developed as an alternative to assembly language for writing operating systems. It assumes that the programmer knows what he/she is doing.
  - “C” is almost always wrong in that assumption
- SQA is about writing professional code
  - The devil is in the details…
- Virus writers and other cybercriminals exploit the results of unprofessional code (sloppy coding practices)

© Copyright year ABB.
All rights reserved.

Naïve Reasons for Unprofessional Code

- "It works perfectly in the lab"
  - *No wonder – in the lab everybody* tries *to make it work*
- "I documented the restrictions to the parameters"
  - *Look for places in the documentation where it says "don't do this". Try as many variants of this as possible.*
- "I will think about security issues when the code works"
  - *When the code works, you will be thinking of the next project.*

© Copyright year ABB.
All rights reserved.

Example 1: The Buffer Overflow Exploit

- The strcpy() bomb
  - A frequent root cause in Microsoft security bulletins, the “Buffer Overflow” vulnerability is usually caused by uncritical use of the standard “C” string copy function strcpy().

    What this function *does*, is copying a string into a buffer. What it *does not* do, is checking whether the string fits inside the buffer. Therefore, strcpy() will happily keep copying the string data on top of whatever data that are adjacent to the buffer. This behavior causes all kinds of problems – from the obscure to the catastrophic.

© Copyright year ABB.
All rights reserved.

What can we learn from this example?

- This vulnerability has been known for years, but programmers still stumble into the same trap (can't be bothered to check the length of the input string)
- The code checker utility *lint* will not catch it, since it is a legal call to a legal function (as usual, "C" gives you full permission to shoot yourself in the foot)
- There has been a large paradigm shift in programming
  - We used to live in a world where "stupid users" were blamed when programs crashed.
  - We now live in a world where criminals try to crash your programs. You cannot shift the blame any longer

© Copyright year ABB.
All rights reserved.

Example 2: The Null Pointer Exploit

- The malloc() bomb
  - Several “C” library functions (e.g. malloc()) return a NULL pointer to indicate an error. Sloppy coding skips testing the returned pointer for NULL and uses it as if it were a valid pointer. Writing something into location 0x0000 (=NULL) - or close by – usually introduces a catastrophic fault at an unrelated part of the software.
    - Usually, the interrupt vector table lives close to address 0x0000.
  - Security experts expect NULL pointer exploits to be the next big wave of cyber attacks.

© Copyright year ABB.
All rights reserved.

What can we learn from this example?

- Always check the value of returned pointers. NULL pointers indicate an error!
- Additional checking may be needed on some CPU architectures
  - Several architectures crash (remember the picture of a bomb on early Macintoshes?) if the pointer has incorrect alignment
  - A popular architecture (ARM) does not crash, but returns a wrong value when the pointer is out of alignment
  - An Ethernet packet has a header of 14 or 18 bytes. This means that if the start of the Ethernet packet is aligned on a longword boundary, the data part will not be aligned.

© Copyright year ABB.
All rights reserved.

SQA consequences

- Make code review a mandatory part of the development process
  - The code reviewers must have instructions as to what to look for
- Extended code checkers (for example Splint – Safe Programming lint) should be used by the code reviewers
  - Such tools cannot do the whole job, since they are easily fooled by clever and lazy programmers

© Copyright year ABB.
All rights reserved.

Example 3: The Protocol Overflow Exploit

- Overloading the Protocol Handler
  - The classic example is the Denial-of Service attack where an enormous amount of packets robs the protocol handler of CPU and memory resources.
  - Another example: If you have implemented a “return status” function in your code with the implicit assumption that it will be called at most every second, what happens when it is called 500 or 10 000 times a second?

© Copyright year ABB.
All rights reserved.

What can we learn from this example?

- Always document your assumptions. Then think about what to do when those assumptions are violated.
  - Implicitly this means that a software design document must be a part of every software development project. This document is where assumptions and exceptions must be discussed and documented.

© Copyright year ABB.
All rights reserved.

SQA consequences

- Require a software design document for each code module
  - This document should contain all the assumptions used in designing the module
  - It should also contain a section on how the assumptions shall be enforced
- When the requirement specifications for a piece of software implies a security hole, document this clearly
  - Try to show how the requirement specification can be altered to avoid the security hole

© Copyright year ABB.
All rights reserved.

Example 4: The Buffer Underflow Exploit

- The recv() bomb
  - Several high-level protocols use the *recv()* or *recvfrom()* calls to handle data reception. Failing to properly check the return value leaves you wide open to exploitation:

```
char rxbuf[sizeof(MyProtocol)];
  union _sockaddr  s_info;
  int  rlen, addrlen;

  while (!Terminated) {
    memset(&s_info, 0, sizeof(s_info));
    addrlen = sizeof(s_info.sa);
    rlen = recvfrom(sock, rxbuf, sizeof(MyProtocol), 0,
      &s_info.sa, &addrlen);
    if (rlen>0)
      myProtocolHandler(rxbuf);
```

  - What if *rlen* is 1?

© Copyright year ABB.
All rights reserved.

SQA consequences

- Return values should be checked
  - It takes one code line.
  - All versions of lint will catch ignored return values
- In some cases the return values contain no useful information and can safely be ignored. This must be documented in the code!

© Copyright year ABB.
All rights reserved.

More SQA musings

- The default scope of a function should be *local* (instead of the default global scope “C” insists on)
  - That way, they are invisible to criminals when you accidentally release code containing debug symbols.
  - Yes, there are people out there who inspect your files with all kinds of tools to see what they can learn
    - Examples: The DVD protection algorithm, the Sony rootkit, etc., etc.

© Copyright year ABB.
All rights reserved.

Protecting against the next exploit

- Protect the automation network against the internet jungle
  - Firewalls, Network Address Translation
- Protect the automation network against unauthorized access
  - Access Control Lists, encryption, authorization
- Protect automation modules against malicious reprogramming

© Copyright year ABB.
All rights reserved.

Final Words

- Writing the actual code is only 10% of the job
  - A large part of the total time should be spent ensuring that “garbage in” does not result in “garbage out” or worse
- Take a lesson from the hardware developers!
  - Designing a device is fun and fast
  - Checking all the small details takes 90% of the time

© Copyright year ABB.
All rights reserved.

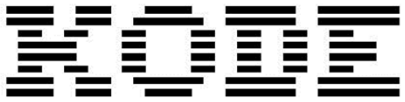
UNIVERSITY OF OSLO

INDUSTRIAL ELECTRONICS AND EMBEDDED SYSTEMS

Communication, Availability, Safety and Security

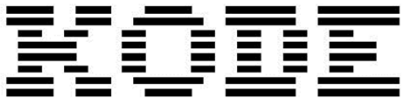
Software Quality Assurance (SQA) in Industrial Electronics

Svein Johannessen



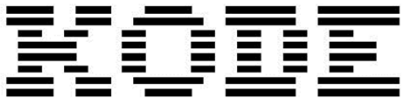
Core Themes in Industrial Electronics

- Communication
  - The area in consideration may be very large
- Availability
  - Shutdown due to equipment failure is not accepted
- Safety
  - Protecting human beings from injury
- Security
  - Protecting the infrastructure from sabotage



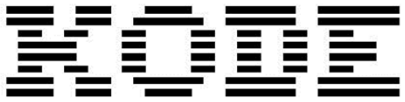
Communication between devices

- There are three basic device classes – *Controllers*, *Sensors* and *Actuators.*
- We need to transfer a information from the sensors to the controller and from the controller to the actuators.
- For that, we use some transfer medium (wires, fiber, air..)
- And in order to interpret the information, we need a set of *protocols*



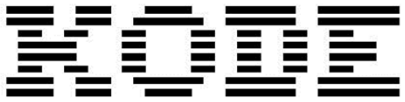
Why use protocols at all

- Even a “perfect” hardware solution may need some help since:
  - Almost all communication solutions have frame size limitations
  - Flow control may be necessary
  - Communication errors may occur
  - Source and destination may be on different hardware standards



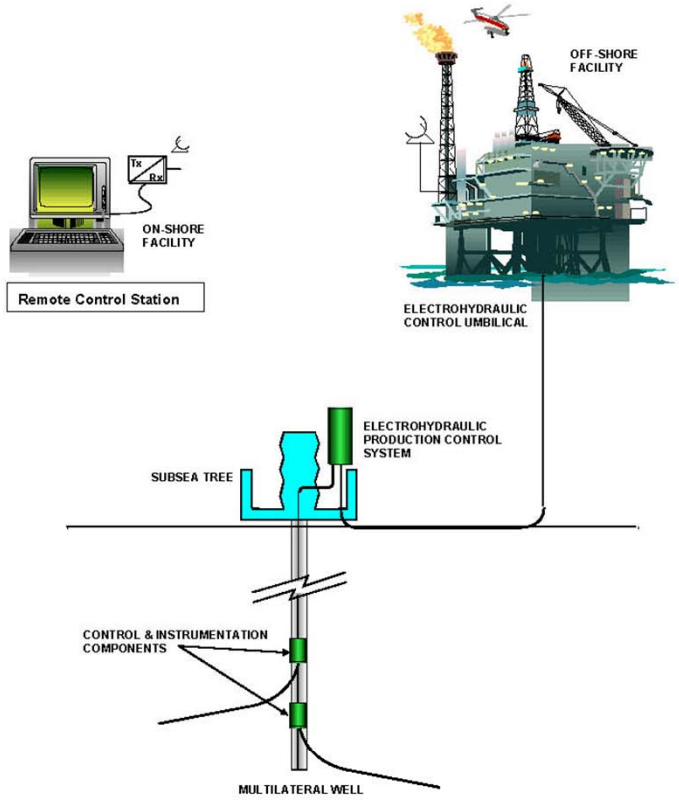
Availability means “keep working”

- Always means that one fault is not allowed to bring the system down
- This includes software faults!
- Especially required when:
  - Repair is extremely expensive (satellites, sea-bottom installations)
  - Repair is impossible (Jupiter fly-by, downhole installations)



Downhole installations

- Long distances (up to 10km below sea bottom)
- Horrible working conditions (up to 225°C, up to 1000 bar etc.)
- This is really trying to communicate with hell!

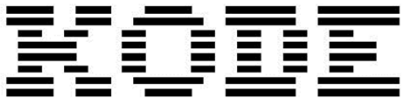


'INTELLIGENT WELL COMPLETION'

KODE

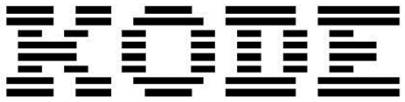
Introducing redundancy

- Redundancy means that you duplicate critical components
  - This can mean anything from duplicated sensors to duplicated subsystems
  - It can also mean duplicated software
    - In this case it means separate development teams, different compilers, possibly different coding languages…
    - And the added complexity of deciding which part to trust when the results differ



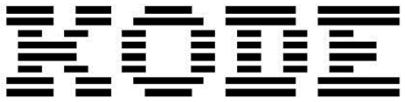
Giving up is not an option

- When faced with an error, amateurs print an error message and exit
- In an industrial context, an error must be handled by the software
  - Therefore the error handling is an important part of the software design
- Professional software relies on professional software design
  - Writing code without a design is the sure mark of an amateur



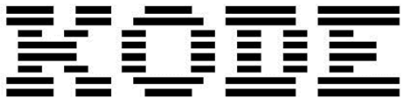
Safety – “freedom from injury”

- In our case, the relevant standard is IEC 61508
- It is titled "Functional safety of electrical/ electronic/programmable electronic safety-related systems“
- The IEC 61508 standard defines a set of Safety Integrity Levels (SIL) based on the probability of a dangerous failure over time. The IEC 61508 SIL3 rating is considered the highest level of risk reduction achievable using a single programmable electronic system.



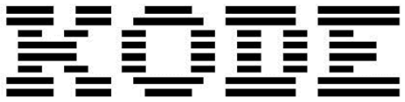
Facts about safety systems

- You cannot just implement a system and call it "safe", it has to be *certified*.
- This means that every part of the design must be documented
  - The certification, design and documentation part is also applicable to the *software*
  - The complete software solution (including the RTOS) must be certified, not just one module
  - For that reason, it is close to impossible to certify existing software systems



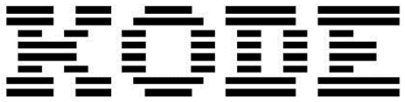
Common safety design elements

- A “safe state” is defined (and certified)
  - A train standing still at a station is in a safe state
- In the case of an error, the system must automatically go to the safe state
- The system is only let out of the safe state as long as safety-proven set of inputs allow it
  - If one or more inputs become unavailable, the system must time out to the safe state



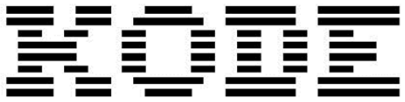
About availability and safety

- A system in the safe state is unavailable
- Therefore, safe inputs and outputs are usually implemented using redundancy and validation
- The easiest way of implementing safe I/O is usually through duplicated subsystems
  - The inputs are combined in a safe way
  - The outputs are combined using AND



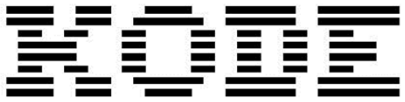
Security – protecting the installation

- The Internet gave us freedom of information – but it also gave criminals new opportunities
  - Cyber-blackmail is a constant threat
- In industrial systems, usually no confidential data can be compromised
  - But the systems can be brought to a standstill (loss of revenue)
  - Safety subsystems can be disabled (litigation)



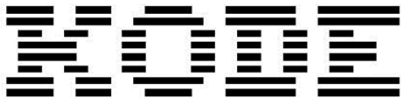
Protecting against the next exploit

- Protect the automation network against the internet jungle
  - Firewalls, Network Address Translation
- Protect the automation network against unauthorized access
  - Access Control Lists, encryption, authorization
- Protect automation modules against malicious reprogramming



Summing it all up

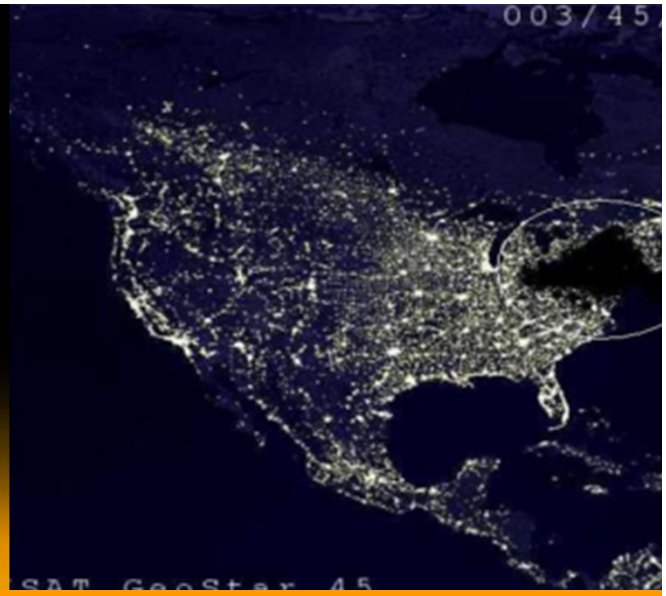
- Industrial systems require professional software
- Professional software is *designed*
  - Error handling is an important part of the design
- Professional software *checks* the input parameters and return values
  - Avoid amateur errors like buffer overflow and NULL pointer exploits



Networked instrumentation – a security issue

- Modern lab instrumentation and control systems are networked. In the beginning nobody thought about the security risks that this introduced
- What follows are some pages from a CERN Student lecture in 2009 by Stefan Lüders





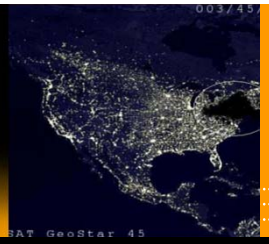
Control Systems Under Attack !?

…about the Cyber-Security of modern Control Systems

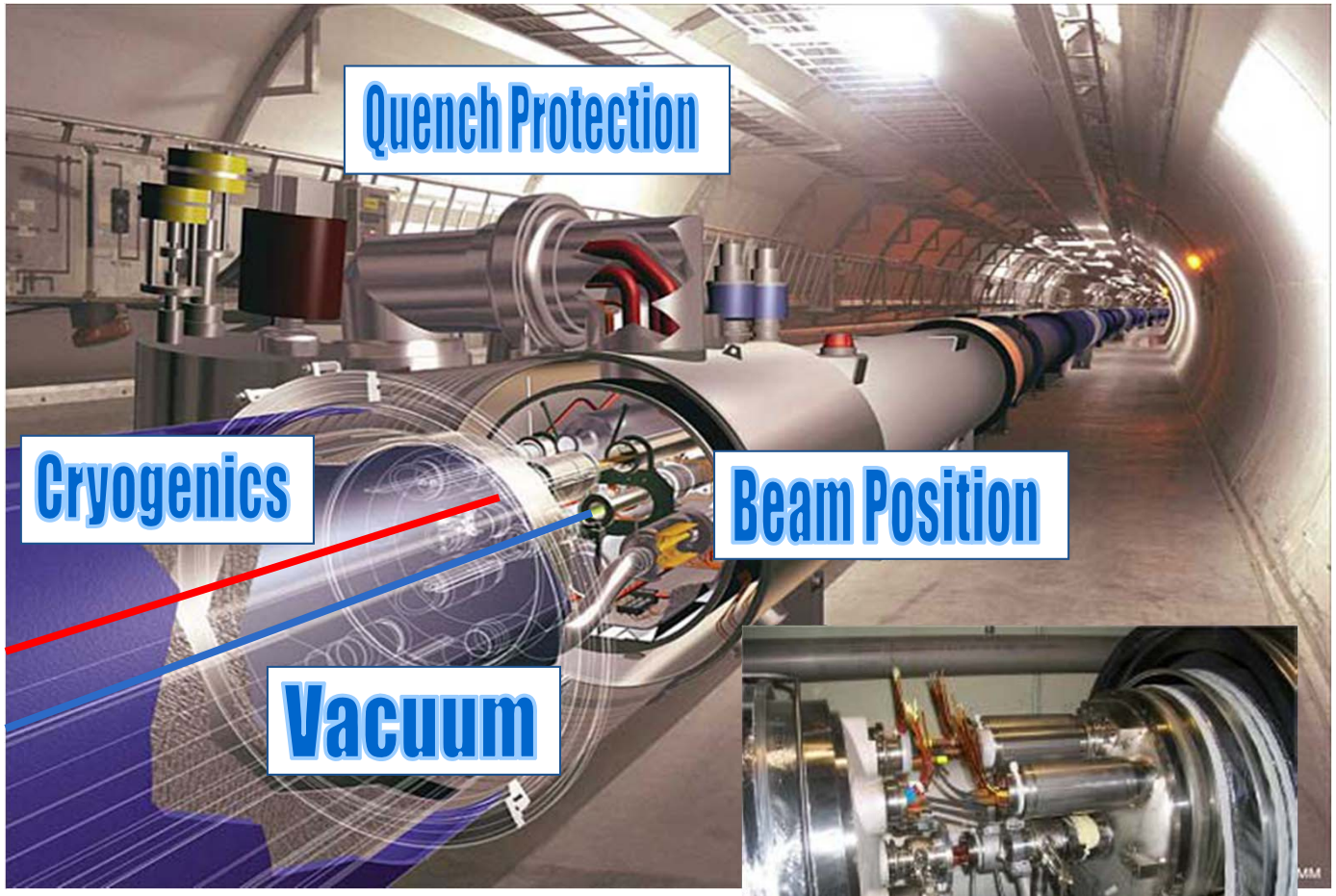
Dr. Stefan Lüders (CERN Computer Security Team)

CERN Student Lectures

January 13th 2009

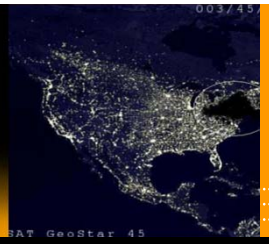



LHC Beam Optics



Steer a beam of 85 kg TNT through a 3mm hole 10000 times per second !

World's largest superconducting installation (27km @ 1.9°K) worth 2B€

Data Acquisition Control



(Sub-)Detectors

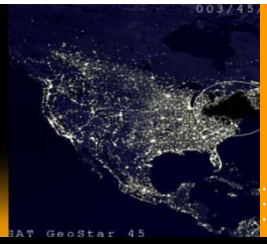
Data Acquisition

Triggering

Experiment

Run Control

About 100 million data channels

Control Systems for Experiments

Electricity

Gas Distribution

Cooling & Ventilation

High Voltage

Magnet

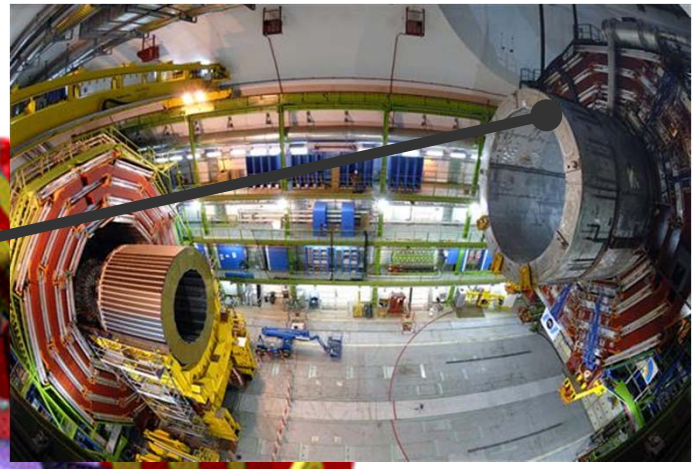
Safety

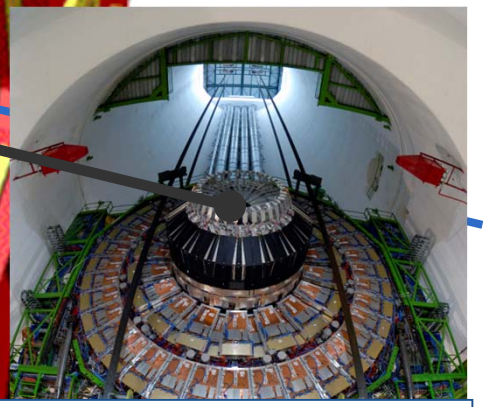
Cryogenics

Radiation

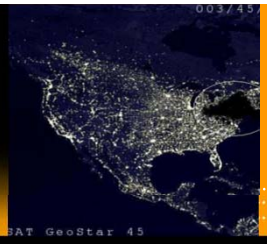
Smoke

Sniffer





About one million control channels

(R)Evolution of Control Systems

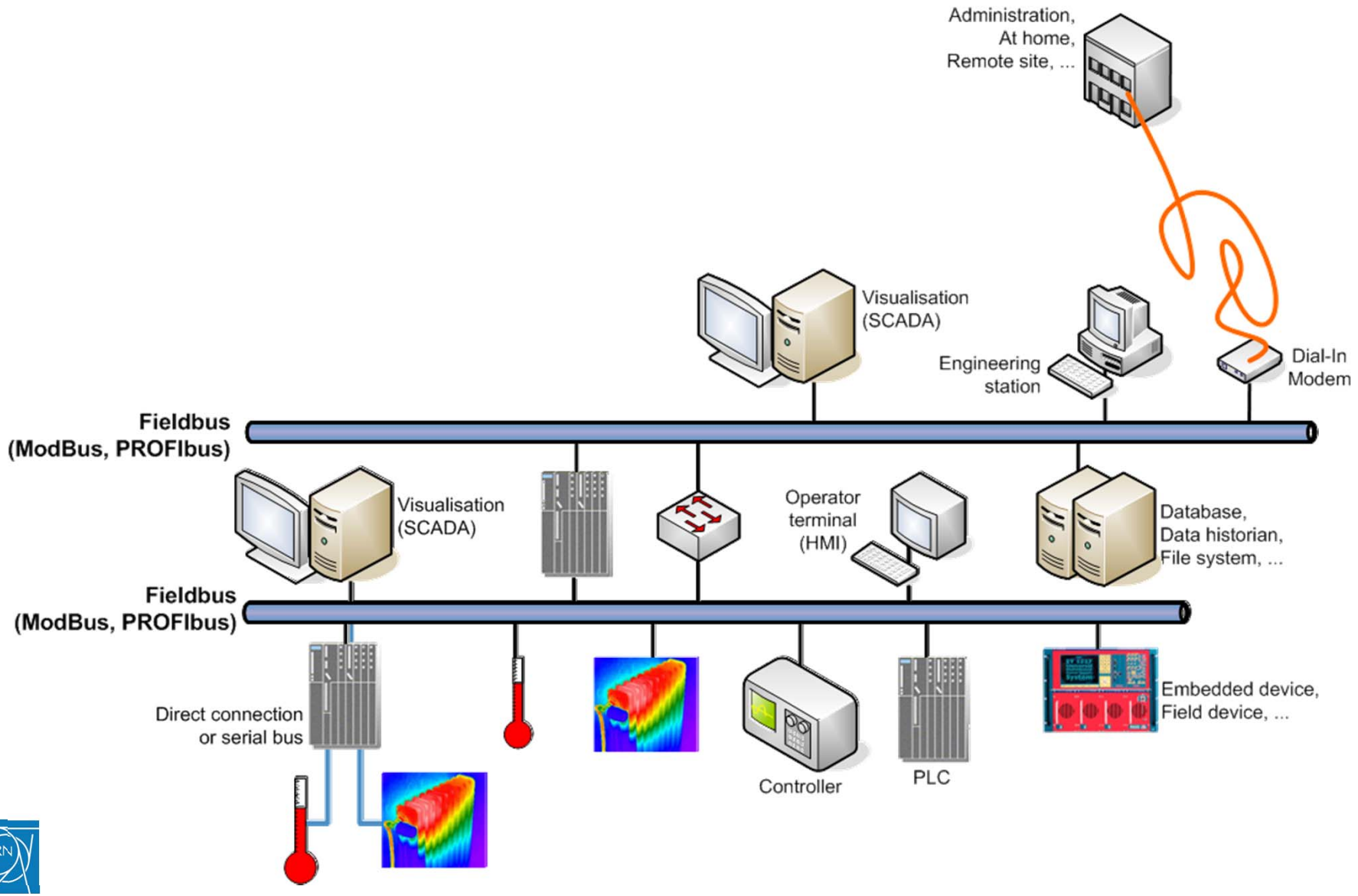
Windows & Unix
Wireless & Laptops

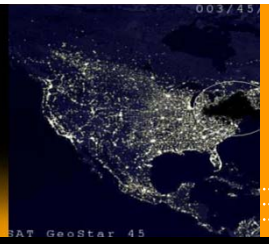
PROFInet, Modbus/TCP
OPC, FTP & Telnet

Web & Emails
C#, Java, PHP, Python, ...

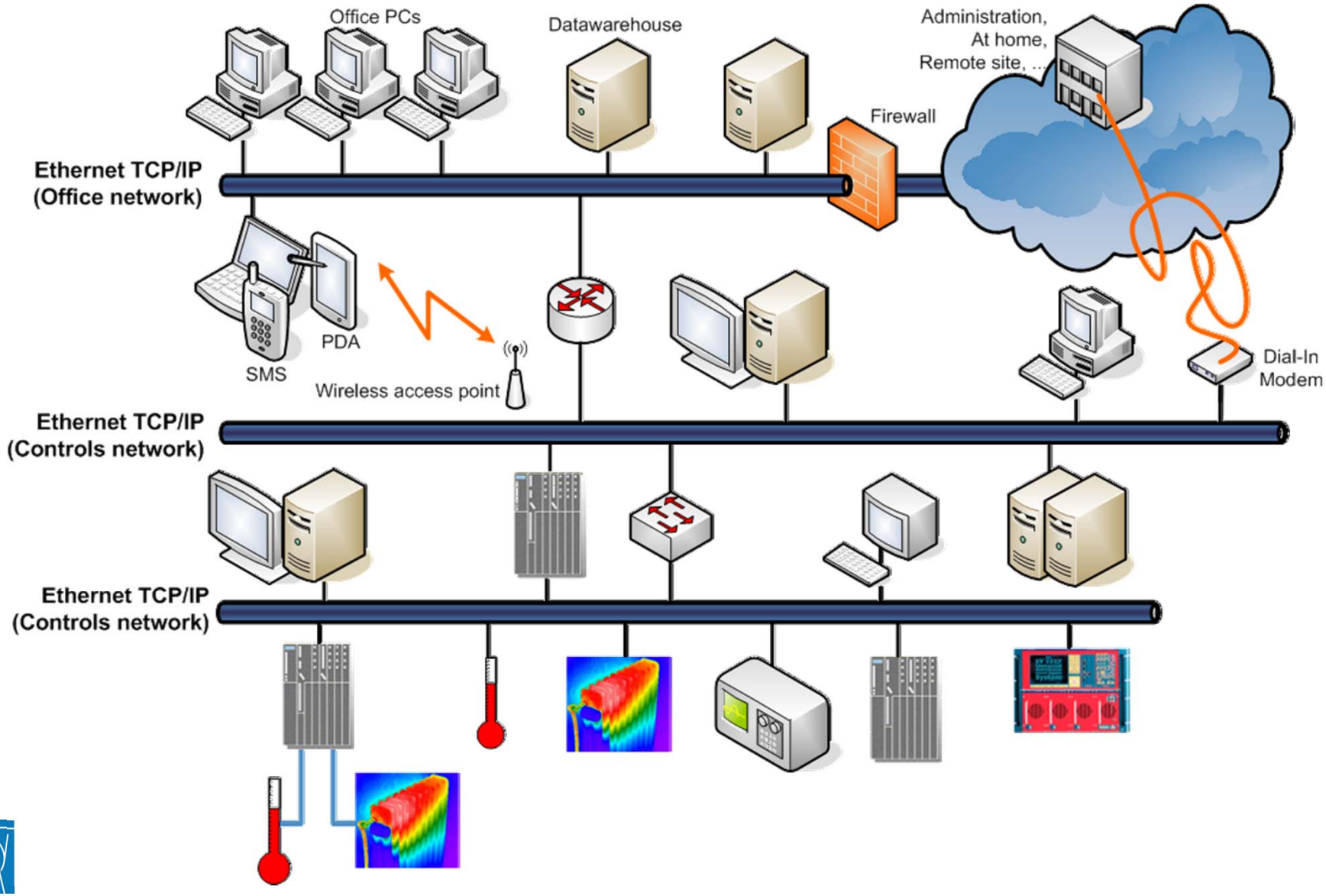


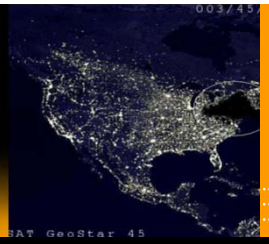

(R)Evolution: The Past





(R)Evolution: Today

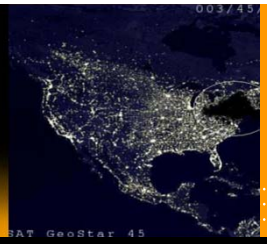


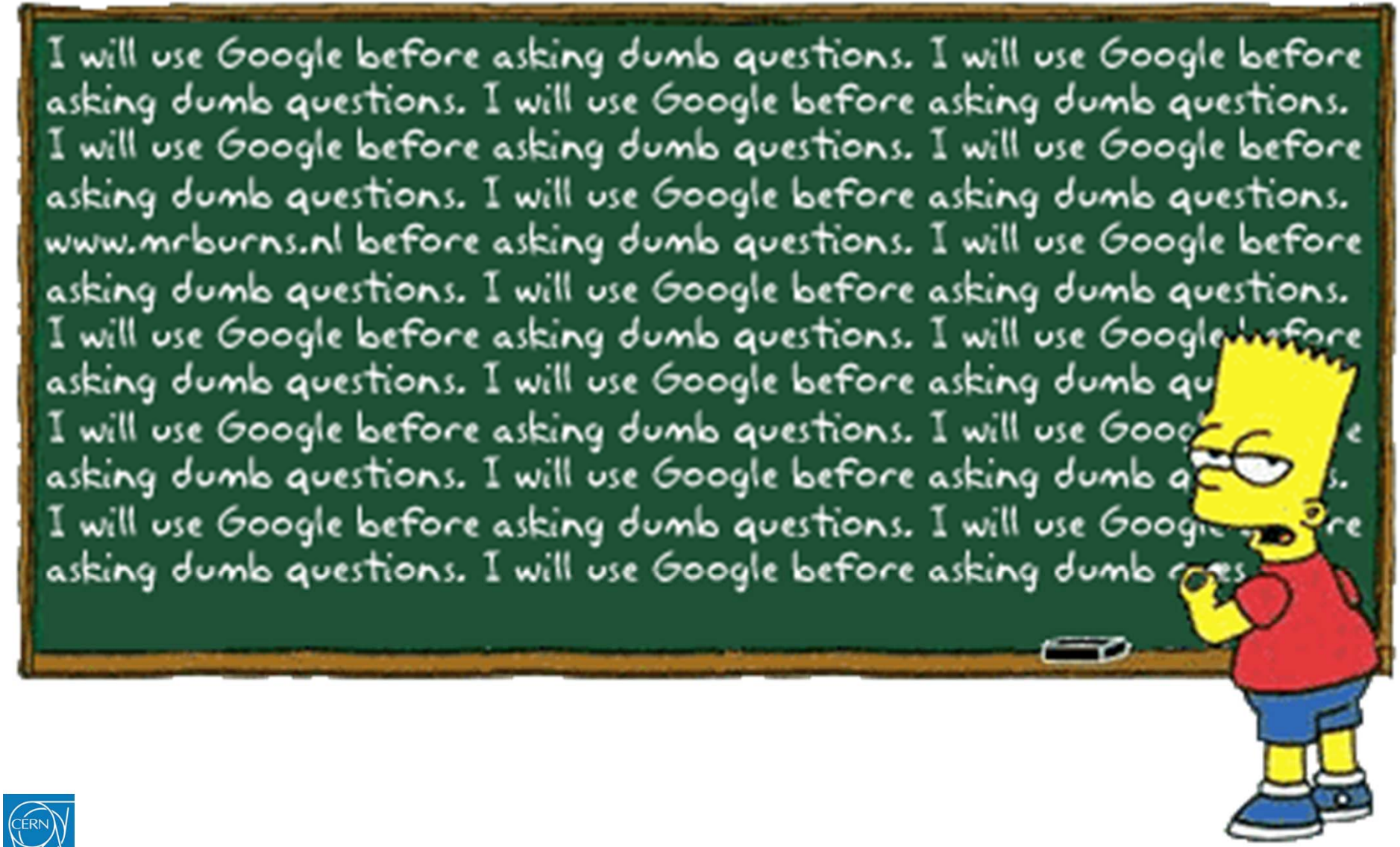
“Controls” is *not* IT ! (2)

	“Office IT”	“Controls”
Changes	frequent, formal & coordinated	rare, informal & not always coordinated
Patches & Upgrades	frequent	infrequent or impossible (needs extensive tests)
Antivirus Software	standard	rare or impossible (might block CPU)
Reboots	standard	rare or impossible (processes will stop)
Password Changes	standard	rare or impossible (password “hardwired”)

"Do not touch a running system !!!"



Thank you !


I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. www.mrburns.nl before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions.