Course "Debugging"

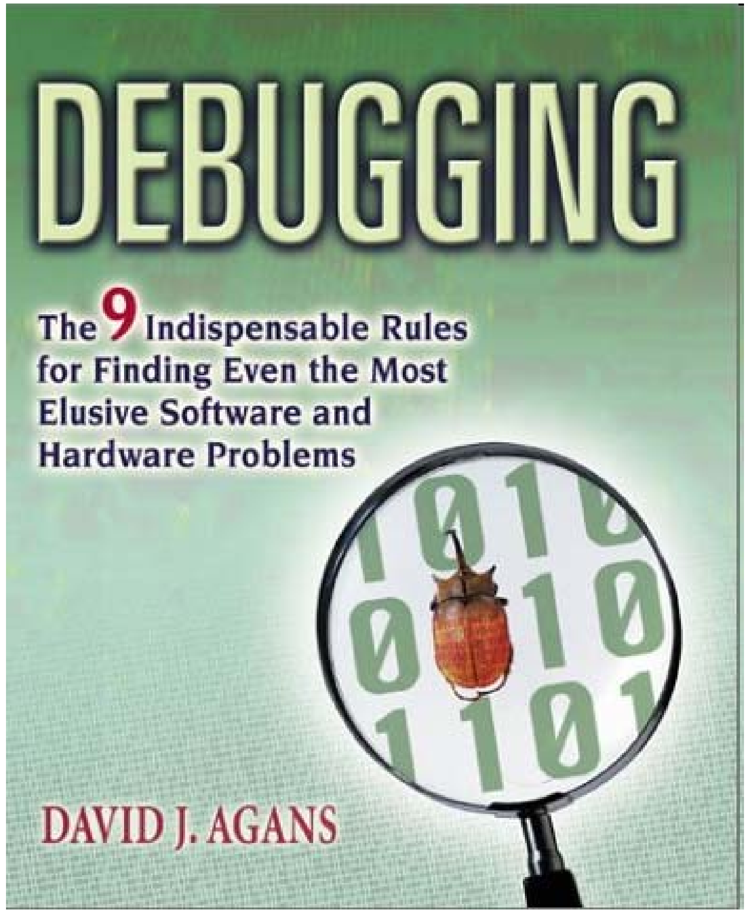
Debugging – An Introduction

Prof. Dr. Lutz Prechelt
Freie Universität Berlin, Institut für Informatik
http://www.inf.fu-berlin.de/inst/ag-se/

- Universal method of debugging
- 9 rules
  - subrules
  - examples ("war stories")

Source

- David J. Agans:
  "Debugging – The 9 indispensable rules for finding even the most elusive software and hardware problems", Amacom, NY, 2002
  - Only 180 pages
  - A fun read!


DEBUGGING
The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems
DAVID J. AGANS

Obvious?

- Most of these rules may look obvious

BUT:

- Obvious does not mean easy
- It is not obvious how to apply them to any one problem
- Often neglected in the "heat of the battle"
- Few people follow <u>all</u> of these rules naturally
  - "Debugging is an art"

Universal

The rules work for

- software
- computer hardware
- other electronics
- cars
- houses
- human bodies
- etc.

The rules work if the system

- has been designed wrong
- has been built wrong
- has been used wrong
- is broken

What the rules are about

The purpose of the rules is to

- determine the causes of misbehavior (defects)
- correct the causes of misbehavior

The purpose is NOT to

- prevent defects ("process management")
- detect the presence of defects ("testing", "use")
- decide whether a defect should be corrected (an aspect of "quality management")

Tatatataaaaa: The nine rules

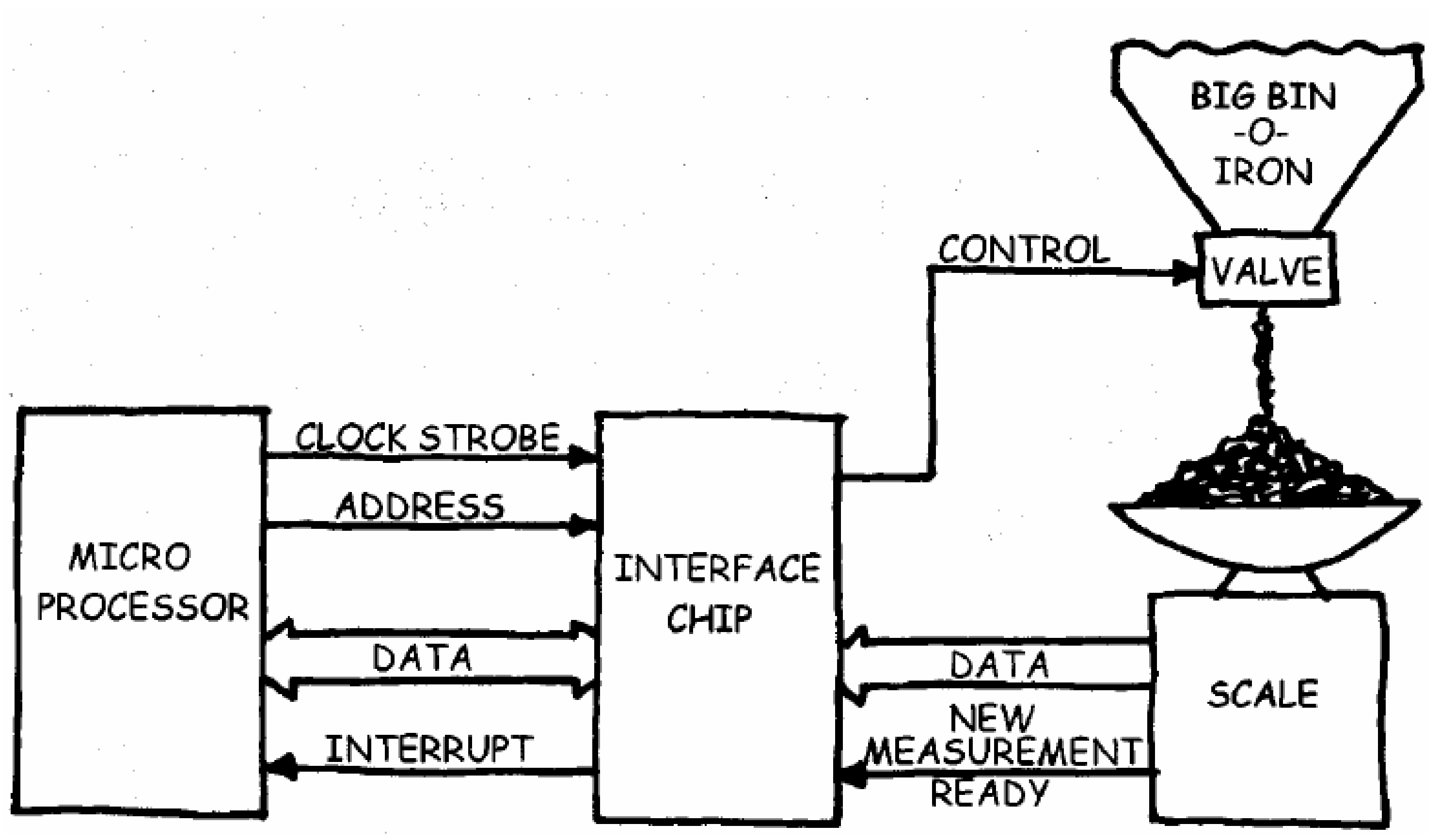
1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Understand the system: The war story (1)

- Situation: A microprocessor-based valve controller, built from a re-used design
- Problem: When the scale had a new measurement, the interface chip never passed on an interrupt to the processor
  - Even that was difficult to find out
  - No progress in finding out why

Understand the system: The war story (2)

- The system:



Understand the system: The essence

- You cannot find problems if you do not understand how the system is *supposed* to work
- Essentially, you have to "read the instructions"
  - Preferably *before* things go wrong
- Experience shows that the least understood parts of a complex system invariably have the most problems
  - You need the understanding at *design time*!
- Unfortunately, understanding a complex (software) infrastructure can be extremely time-consuming
  - This is usually the most time-consuming rule to follow
  - But also usually the most worthwhile

The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Make it fail: What and why

- For efficient debugging, you must be able to reproduce the failure at will
  - You need to find out the exact steps to make it fail

Why you need this:

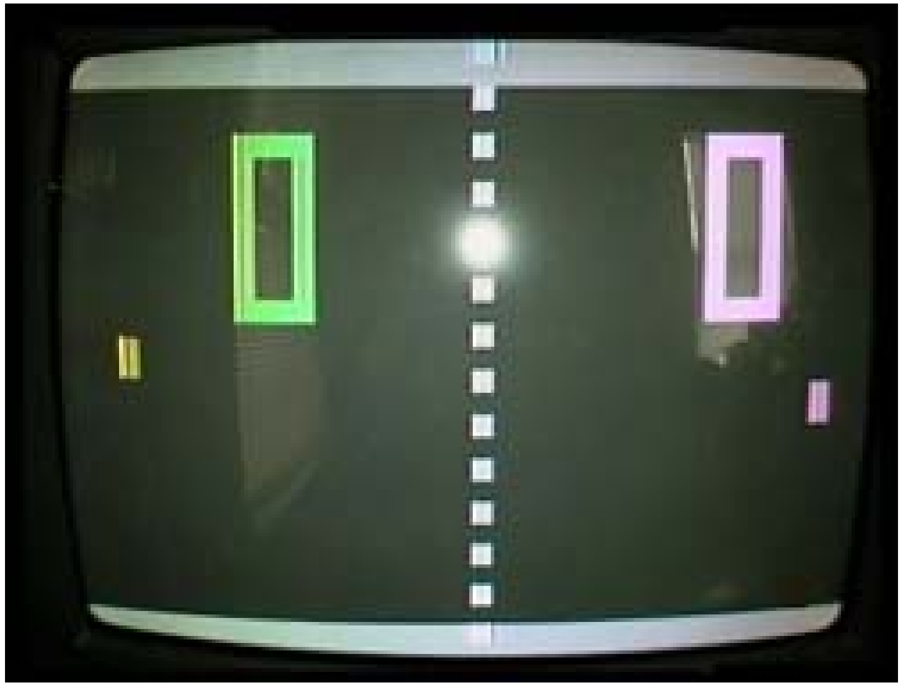
- To look at the failure itself
  - and understand its characteristics
- To be able to focus on finding the cause
  - rather than worry how to make it fail at all
- So you can tell when you have fixed it

Make it fail: Practical hints

- Do it again
  - After you found out how to make it fail, do it once more to make sure you have the right procedure
- Start at the beginning
  - Make sure you can reproduce the failure from a repeatable (clean) initial state: restart everything from scratch
- Automate
  - If making it fail involves a lot of steps, automate their execution
    - see the war story below
  - If the conditions of the failure are rare, create them artificially
    - e.g. an allergy test by explicitly applying allergens
    - e.g. create heavy-load conditions to provoke a known heavy-load failure

Make it fail: War story (Automation)

- Context: Debugging an analog TV 'Pong' game
- Problem: The ball would sometimes wrongly bounce off the 'practice wall'
  - Manual playing kept attention away from observing the failure
- Automation:
  - Both ball position (x, y) and paddle position (y) were represented by voltages
  - Connect paddle y to the ball y voltage and the game will play itself.



Make it fail: War story (Stimulation)

- Context: A house
- Problem: A particular window leaks in heavy rain -- but only sometimes
- Stimulation: Create artificial heavy rain by using a hose
  - The leaking happens only when the rain comes from southeast
  - Closely investigating the window finds a break in the caulking at that side of the window
  - After fixing the caulking, another hose test confirms that the defect has been fixed

Make it fail: Practical hints (2)

- Do <u>not</u> simulate the failure
  - It is good to stimulate (even amplify) the real failure
    - see the window/hose example
  - It may be helpful to try and make a <u>similar</u> system fail
    - so you can narrow down the possible causes
  - But it can be misleading to simulate the failure only
    - Because that involves guessing the failure mechanism
    - If your guess is wrong, you'll end up on the wrong track

Make it fail: What if it's intermittent?

- Some failures appear to be irreproducible (intermittent)
- But they aren't:
  - The factors evoking the failure are fixed (remember?: laws of nature!)
- However
  - a. you may not know what the particular factors are
    - and there may be many to choose from   or
  - b. you may not be able to control those factors
    - or the ones you would like to check for

Make it fail: What if it's intermittent? (2)

If you do not know the relevant factors:

- Try to find some relevant factor by trying to make the failure more frequent
- Method: trial-and-error experiments
  - Guess all kinds of conceivable factors
  - Change them and observe
  - If multiple factors are involved, only randomization helps
    - but randomize systematically

- Sometimes making it fail somewhat more often may be the best you can achieve
  - but that may be very helpful

Make it fail: What if it's still intermittent? (3)

If you cannot control the relevant factors (or just still don't know them):

- Instrument the system to capture enough information about the few failures you get and about normal executions
  - and compare those two kinds
- Systematic differences usually provide the clue for finding and fixing the problem

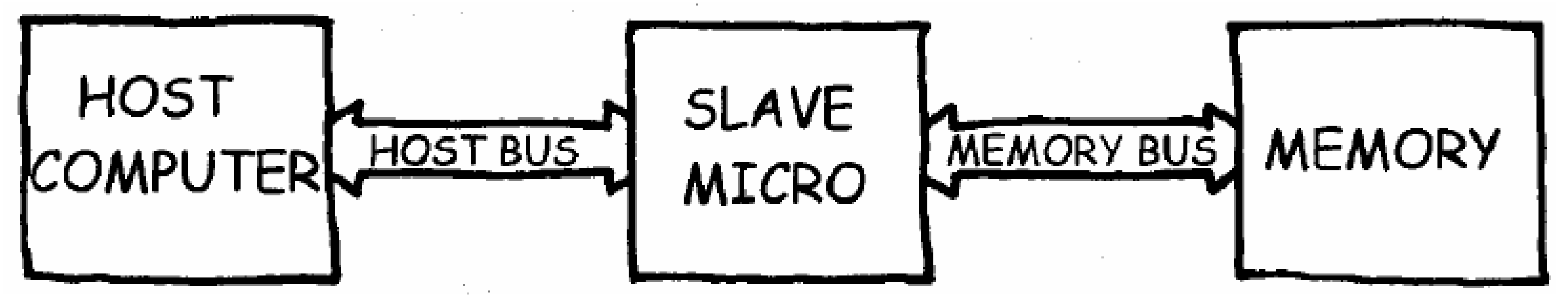
- Problem: How can you make sure you fixed it?
  - You need to find a failure signature: Any run showing these conditions will fail
  - Then after your corrections, when you see such a run, but no failure, you know you have removed your problem.

The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

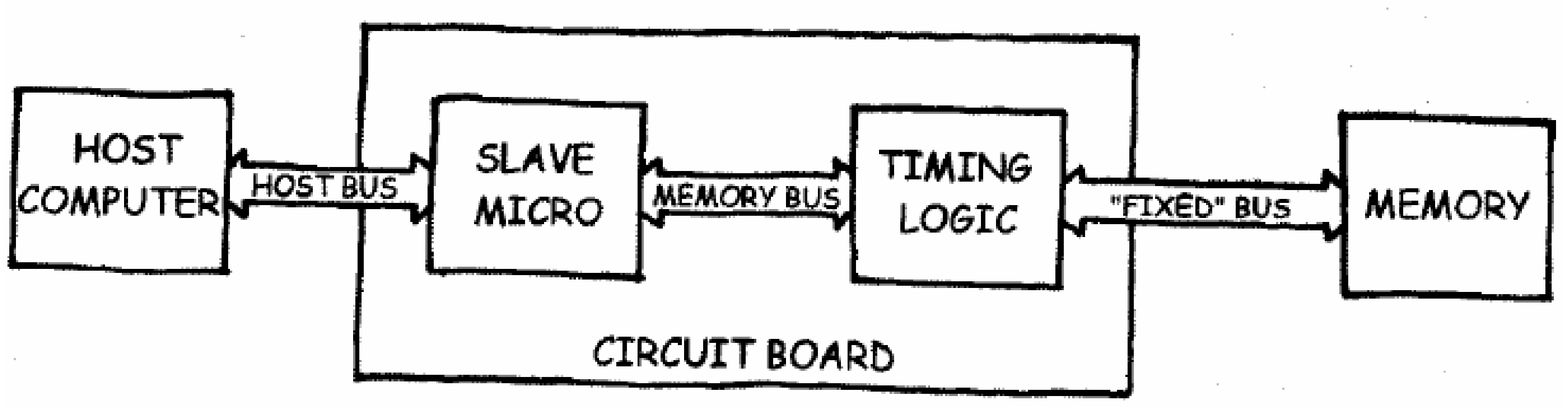
Quit thinking and look: War story

- PC card with slave microprocessor failed sometimes after the program upload
  - memory checksum was incorrect
- Several junior engineers were assigned to fix this
- Their first test: Repeated writes into a register on the card microprocessor
  - result was always correct



Quit thinking and look: War story (2)

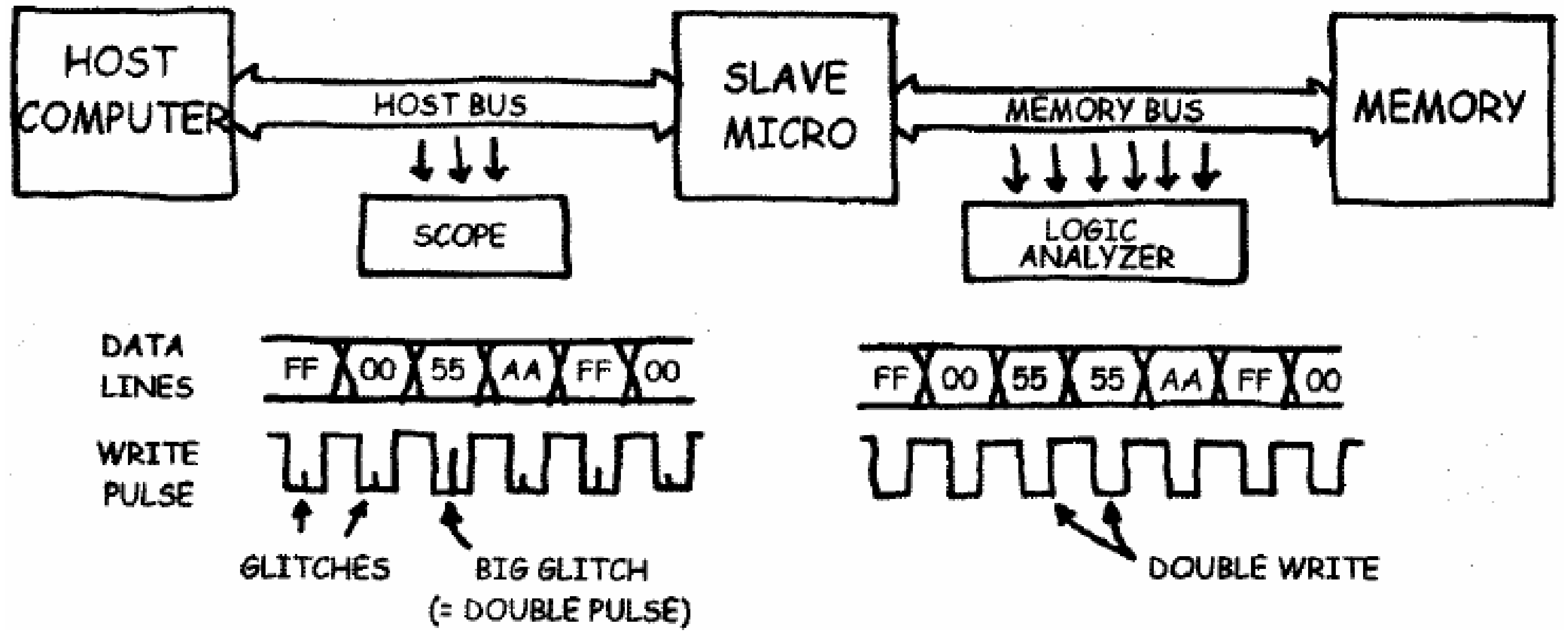
- Conclusion: Data transfer into the card works alright
- Next step: Understand the system
  - They analysed the memory interface circuits
  - They found that its timing design was borderline
- They assumed this was the problem
- They worked out an additional "fix-the-timing" circuit
  - That took several months!



Quit thinking and look: War story (3)

- Result: The card failed just as often as before
- Now a senior engineer stepped in and insisted they first see the actual failure
- He hooked up a logic analyser to the memory bus and observed the results of repeated writes of the following pattern (to subsequent byte addresses): 00 55 AA FF
  - He sometimes found 00 55 55 AA FF
- So the writes to the card could be duplicate sometimes
  - He checked the write pulse to the card and found it to have noise which sometimes made it look like two pulses
- In the junior engineers "write register" test, this could not be observed
  - Writing the same value twice *to the same register* is not a problem

Quit thinking and look: War story (4)



Quit thinking and look: Subrules

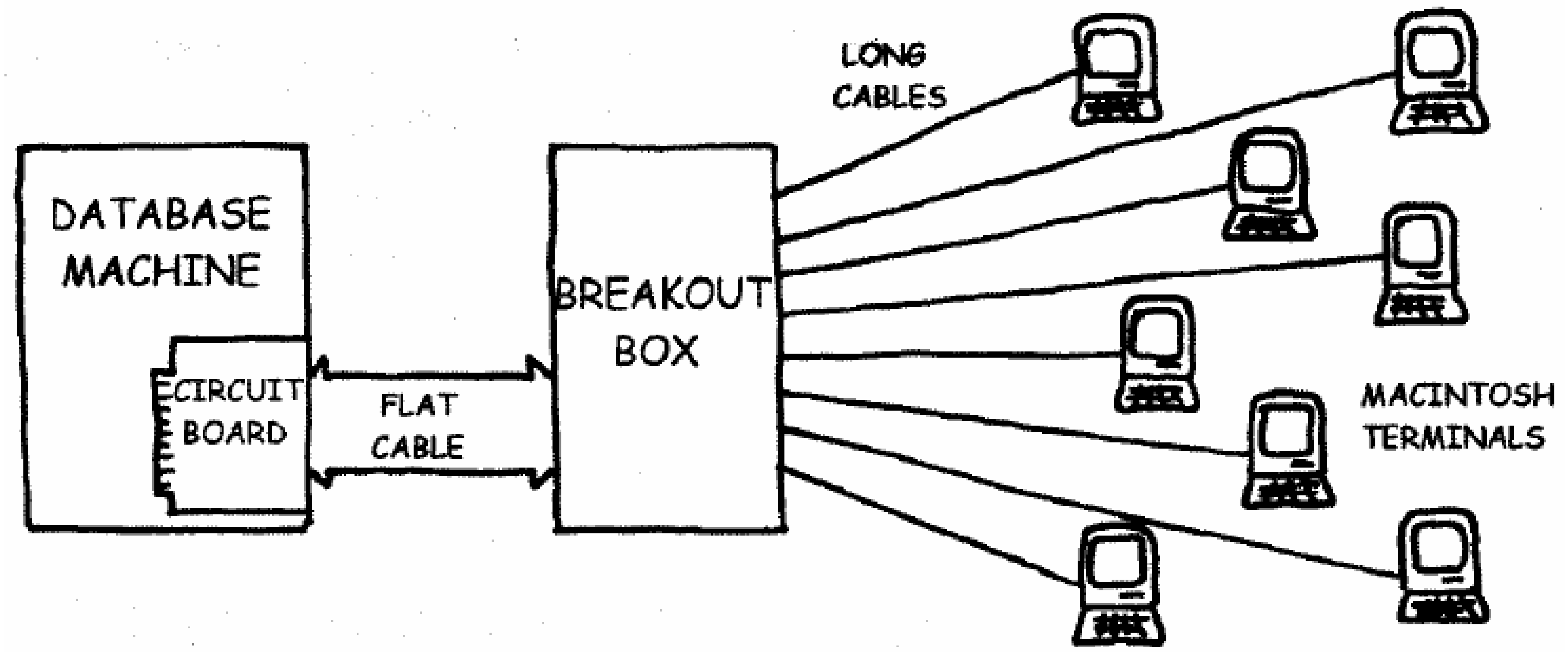
- Subrule: See the failure
  - We tend to jump to conclusions when really what we are seeing is the consequence of a failure, not the failure itself
    - The junior engineers never saw the timing fail
- Subrule: See the details
  - Looking once is seldom enough
  - More typically, each looking provides a little more information; you will understand the failure bit by bit
- Subrule: Now you see it, now you don't
  - Seeing the actual low-level failure mechanism will be helpful later on when verifying a fix
- Subrule: Instrument the system
  - Looking from the outside may not be easy or good enough
  - Build observation aids (*instrumentation*) into the system

The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Divide and conquer: War story

- Setting: A server (database machine) in a hotel, with 8 Macintosh computers attached as clients
  - Communication over serial cables
- Problem: Database retrieval had become very slow

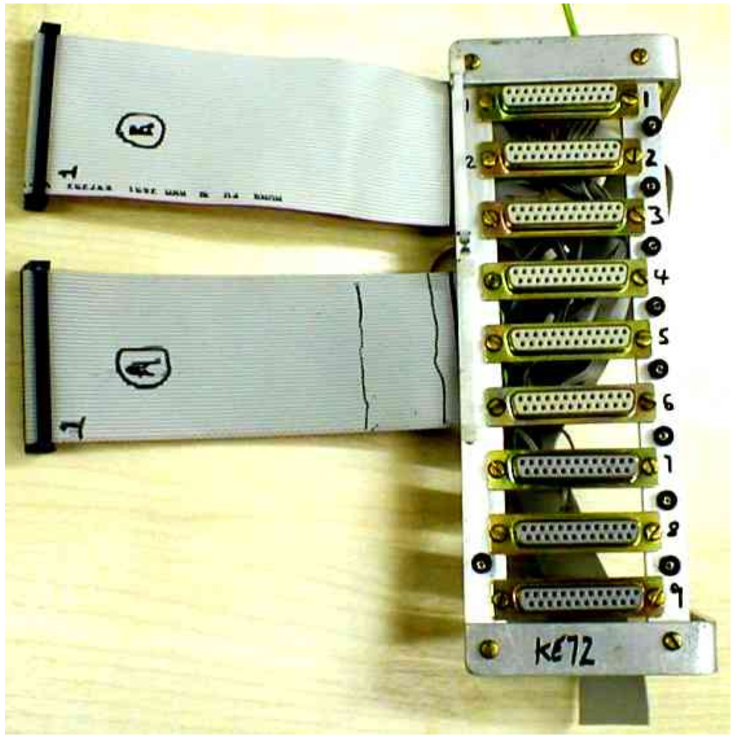


Divide and conquer: War story (2)

Checks of the technician (in order):

- 1. Guess: There are data transmission problems.
  - Check found error messages in the communication log
- 2. Guess: No SW changes, so problem is probably HW
  Guess: All terminals work, so it's
  probably not them
  - Oscilloscope check at server port
    found good signals going out to
    the terminals, and
    weak signals coming back
- 3. Checked halfway between there and the Macs:
  At the terminal sockets of the breakout box
  - Inverse situation: weak signals coming from server, and
    strong signals coming from the clients

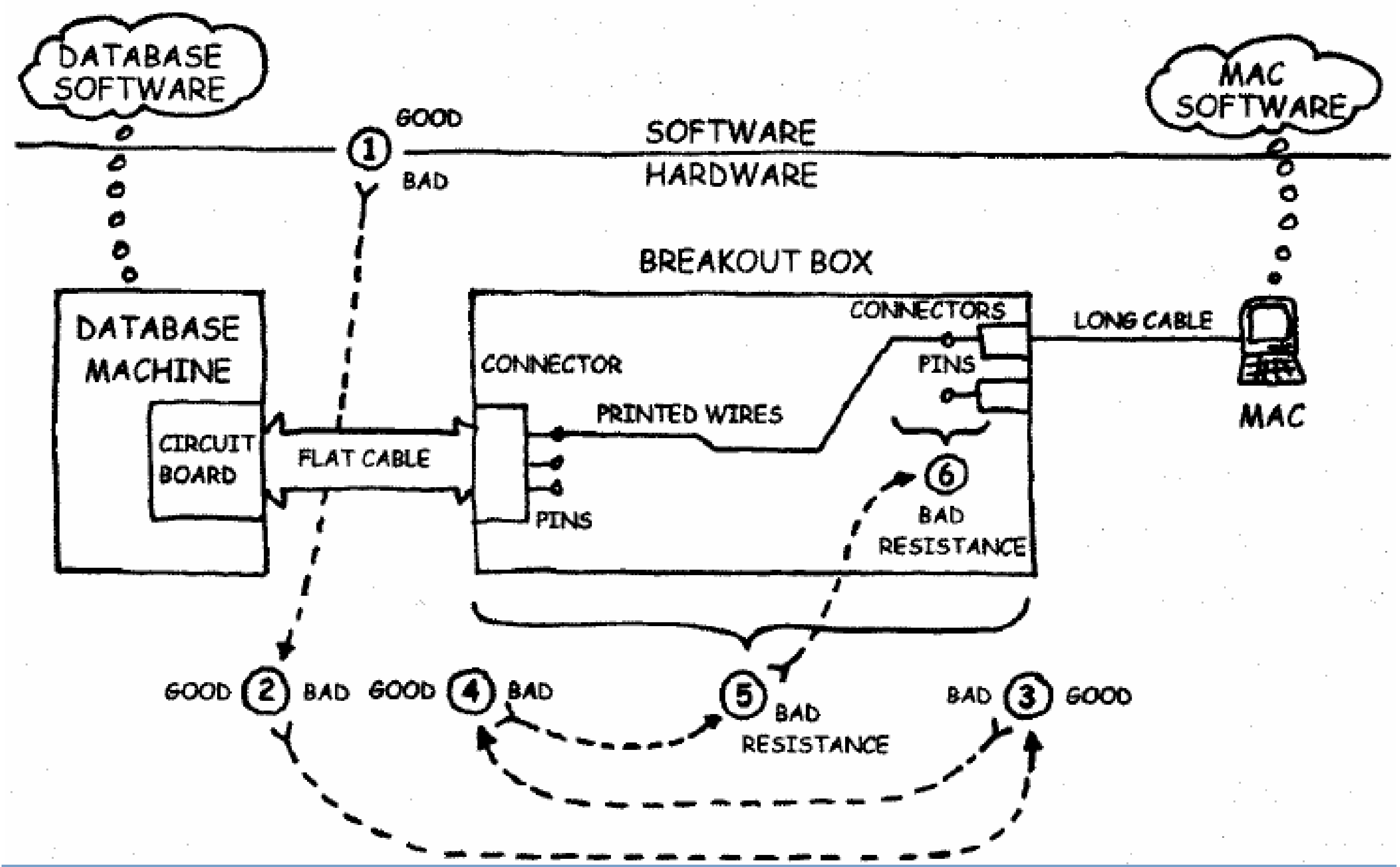
Breakout boxes



Divide and conquer: War story (3)

- 4. Checked end of flatband cable (input to breakout box)
  - Again inverse situation: strong signals coming from server, again and weak signals coming from the clients
- 5. Conclusion: The problem must be in the breakout box
  Check: Measure resistance of the breakout box
  Result: Resistance is much too high
  Consequence: Open the breakout box
- 6. Measure resistance between various points.
  - Found hairline cracks where serial connector pins were soldered to circuit board

Divide and conquer: War story (4)



Divide and conquer: War story (5)

- Repair:
  - Disconnected all clients and the server
  - Reheated all spots with a soldering iron
  - Reconnected all clients and the server
  - Checked that the clients worked fast again
  - Only now re-assembled the breakout box:
    - Disconnect all clients and server
    - Reassemble and reinstall breakout box
    - Reconnect all clients and server
- (Subrule: Never reassemble more than absolutely necessary before checking that your fix works as intended)

Divide and conquer: The essence

- "Divide and conquer" is the central rule of debugging
  - All others are just auxiliary
- Think of it somewhat like binary search:
  Of the range of all possibilities, pick one half,
  check it, and then
  - if the defect is in it: cut it in half again
  - if not: <u>check</u> the other half!
    - if the defect is in it: cut it in half again
    - if not: think again what your range is and why
- Let's try: *I chose a number between 1 and 100.* Guess it.
- Debugging is successive approximation of the cause of a phenomenon
  - Unfortunately, the "range" is not often as obvious as in the breakout box example, which even continued:

Divide and conquer: War story (6)

- After the breakout box repair, all Macs were working fast again – except one (Number 8)
  - That also had been the slowest of all before
- 1. After another soldering attempt, the technician started over by looking at the communication log again
  - it now showed errors for outgoing data only
- 2. Opened the serial connector plug at the breakout box
  - the outgoing signal looked OK
- 3. Guess: The problem must be "downstream". Opened the connector at the client end of that cable. Found that one of the wires was not even connected.
  - Cable had 6 wires; only 4 had to be used.
  - Blue had been connected instead of purple.
    - 30 m of cable length induced enough electrical coupling

Divide and conquer: The essence (2)

Debugging as binary search for a cause:

- The rules' purpose is
  - to help understand what the range is
  - to help understand what useful halfs may be
  - to pick the more likely half
  - to simplify the checking
  - to make sure your check works
  - to help interpreting the check result
  - to help you find another range entirely in case you went wrong altogether (or have no idea at all)
  - etc.
- We will come back to this view later

Divide and conquer: War story revisited

As neat and clean as the divide and conquer looks our technician used other rules as well:

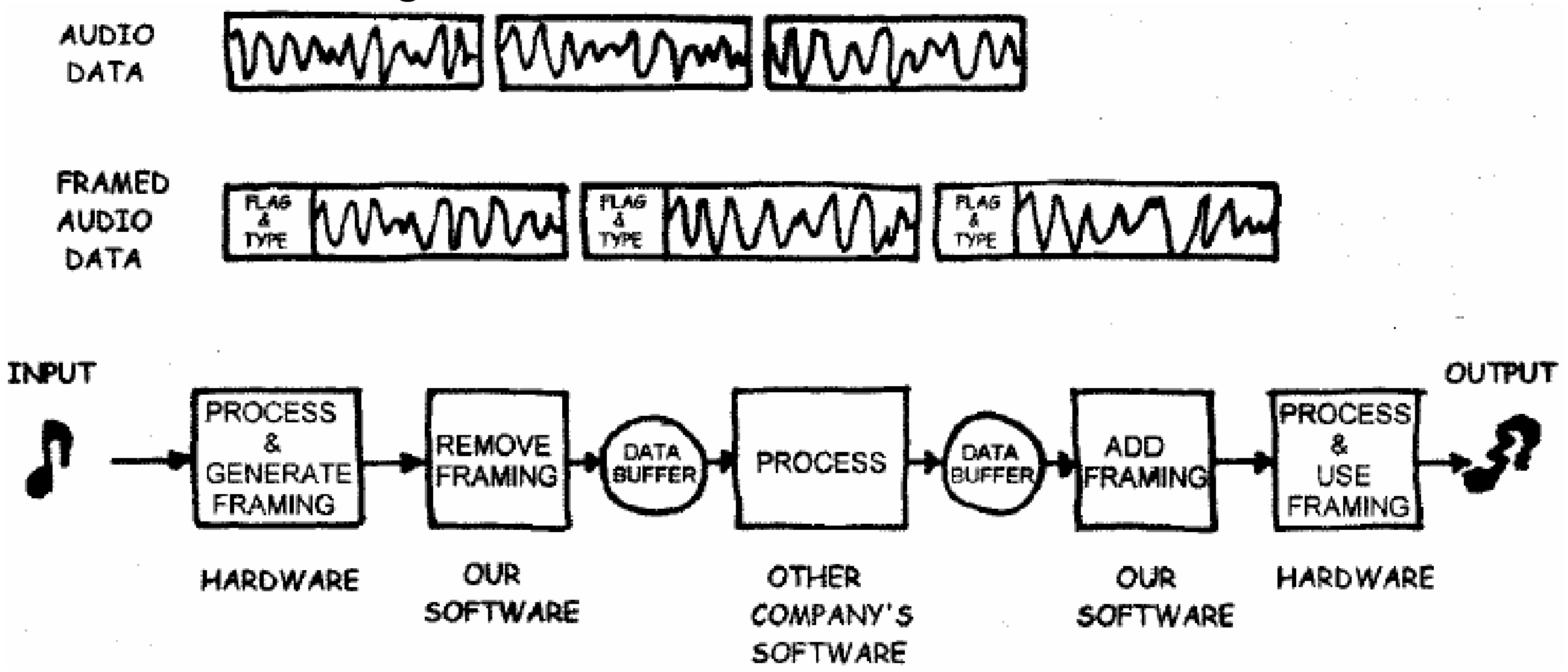
- He understood the system
  - used debug logs, then zoomed in on the hardware part
- He quit thinking and looked
  - rather than starting to replace lots of hardware
  - (which probably would have failed miserably)
- He made the system fail frequently
  - In fact, the regular "I am still here"-traffic between clients and server did it for him
  - But he made sure he really saw the failure (by measuring resistance)

The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Change one...: Audio war story

- A system handling audio data, involving special hardware, our own SW, third party SW, and a speaker.
  - Audio is processed in chunks
  - Sometimes these chunks are 'framed', sometimes raw
- The resulting audio sounded bad



Change one...: Audio war story (2)

- 1. Guess: The engineer suspected that at one point in the process framing was missing
- 2. He added framing in the SW at that point
  - The audio still sounded bad.
- 3. He could not think of other reasons and called in a debugging wizard (DW)
  - DW insisted they tried with simple, known data and instrument the SW
- 4. After some work, they could see the data getting clobbered and traced the cause to a pointer defect
  - They fixed the defect and found that the test data got through the defect spot unharmed
- 5. They tried real audio and it still sounded bad.

Change one...: Audio war story (3)

- 6. They took an hour to reconfirm that their fix would really fix the problem
- 7. They went to reconfirm that their test system was really running the corrected version of the SW
  - At this point, the engineer recognized that he had not taken out his previous, useless 'fix' that inserted framing
  - After taking it back out, the system ran perfectly.

Change one thing at a time: The essence

- "In debugging, always use a rifle, never a shotgun"
  - If you change multiple things at once, you learn little about the effect of each one

It may be still better to change nothing at all, until you really understand what is going on as good as possible:

- Subrule: Grab the brass bar with both hands!
  - According to legend, nuclear submarines have a horizontal brass bar in front of the power station control panel
  - The engineers are trained to grab that bar with both hands when any status alarms go off
  - They have to hold on until they have analyzed and understood all information presented on the panel
    - Overcome the urge to do something

The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Keep an audit trail: Plaid shirt war story

- Debugging a video compression chip used for video conferencing
    - Setup used live signals from a camera
    - Target data rate: 30 frames per second; usually possible
- Sometimes, the chip would slow down to 2 fps
    - and stay there until it was restarted
- The failure cause had nothing to do with uptime
    - sometimes it failed quickly, sometimes not for hours
- Once the chip did not fail a whole day
    - The tester considered different room temperature as the cause
    - Tried heating and cooling the chip – no effect

Keep an audit trail: Plaid shirt war story (2)

- Then suddenly the tester noted the chip failed just when the tester got up from his chair.
- In fact this failure was repeatable
  - He sat back down, restarted the chip, got up: it failed
  - It also worked the other way round: restart while standing, then sitting down: it failed
- But then why had it not failed all day yesterday?
  - He had gotten up multiple times then, too
- What might be the actual technical cause?

Keep an audit trail: Plaid shirt war story (3)

Solution:

- The tester usually wore plaid flannel shirts
- Yesterday was an exception: plain blue formal shirt
- The chip gave up when it tried compressing a very complex signal (the moving plaid pattern)

Lesson:

- The seemingly insignificant **does** matter!
  - At least sometimes
  - But you never know when or what

Keep an audit trail: Everyday application

- When you have a food allergy, the doctor will make you protocol
  - all that you eat and drink (when, what, how much) and
  - the symptoms you get (when, what, how much)
- The food list alone is not very useful
- The symptoms list alone is almost useless
- Even both lists together, but without the times, will be not very useful

- The audit trail must
  - be complete and detailed about all relevant events
  - and must correlate events

Keep an audit trail: Write it down

- Subrule: Write down what you did, in what order, and what happened
  - Or else your short-term memory will be overloaded
  - In your head, you cannot analyze for more than your current hypothesis or focus – much work will be lost
- Subrule: The shortest pencil is longer than the longest memory
  - Written audit trails can be copied,
  - attached to logs,
  - forwarded and shown to other people, and
  - reproduced weeks later when investigating something else

Keep an audit trail: Be specific

Subrule: Be specific!

- e.g. when a program crashes, do not just write down "crashed"
  - Did it produce a proper UI-level error message? Which?
  - If yes, did it terminate afterwards?
  - Did it stop with an exception message, stack trace, memory dump? Contents?
  - Did it freeze? In which observable state?
- if more than one machine is involved, always indicate which one you are talking about
- if a symptom has describable nature, describe it
  - e.g. size, intensity, color, duration, shape etc.

Keep an audit trail: Correlate

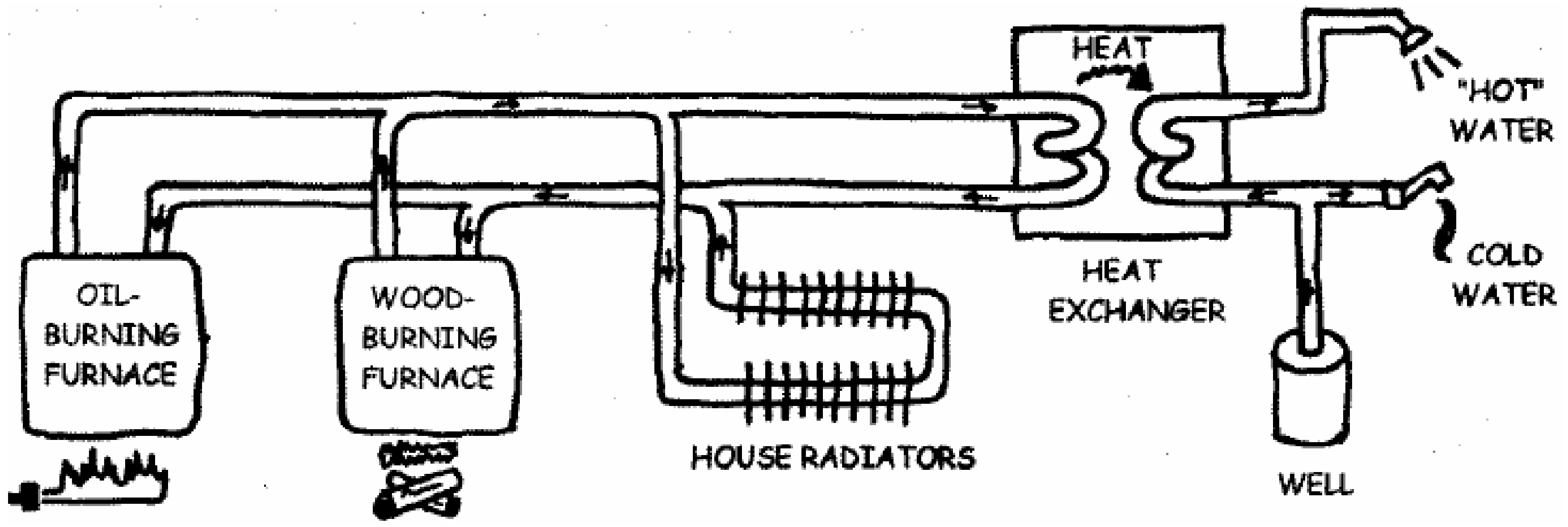
Subrule: Correlate events

- Often you have some information in a log and other information is directly observed
- Make sure you know where the observations fit into the log
    - Time stamps are a good way of doing this
    - Synchronize clocks as precisely as you can
    - In particular if multiple machines keep logs
- If you can't, instrument accordingly

The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Check the plug: Old house war story



- 90 year-old house; most things were present twice
- Heating: Previous owner had added an oil furnace as a backup to the primary furnace heated with wood
- The new owner took the wood furnace out of service

Check the plug: Old house war story (2)

- Problem: When showering, the water would quickly turn cold
- Idea 1: Hot-water pressure drops
  - Possible solution: A pressure-balanced valve
  - But such a thing was already in place!
- Idea 2: Not enough hot water available
  - But the system was instantaneous: It cannot run out of hot water, as it heats it just-in-time
- Idea 3: Hot water production temperature not set hot enough (at heat exchanger)
  - But it was set to 60˚C, enough even for the dishwasher
- Solution: Oil furnace was set to only 74˚C rather than the required 88˚C
  - Reason: It had been meant as a backup only!

Check the plug: Old house war story (3)

Analysis:

- A wrong assumption was at work:
  That the furnace would produce enough heat
- Consequently, the divide-and-conquer approach started with a range that was too narrow
- The assumption was discovered only when the radiators also did not work well in autumn
  - thus pointing to a different kind of problem than previously considered

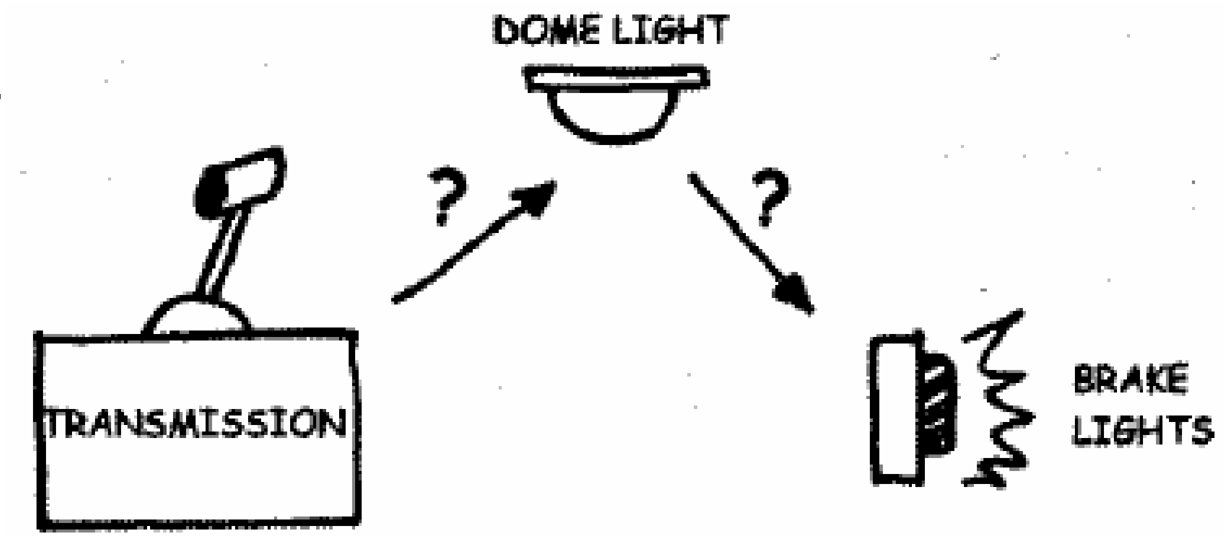
- This kind of "foundation factor" is particularly likely to be overlooked
  - We are often too deep into details to consider the basics

The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Get a fresh view: Brake light war story

- A car blew the fuse of the brake lights whenever you put the transmission into reverse gear
- Several attempts made it clear that this was repeatable
- Reason???
- Owner mentioned the problem to somebody familiar with repairing this brand of car
- Immediate answer: "The dome light is pinching a wire against the frame of the car Insulate that wire and you will be fine."



Get a fresh view: What would have happened?

What would have happened if one had applied only the other rules:

- Understand the system
  - Obtain(!) and study car wiring diagrams
- Make it fail ; Quit thinking and look
  - He did this alright
- Divide and conquer
  - Rip out the car's wiring in parts?
  - Or obtain measurements throughout the wiring?
- Check the plug
  - There was no assumption that could be questioned

Get a fresh view: Subrules

- Subrule: Don't be proud
  - Asking someone is not a sign of weakness (if you've done your part before), but rather of good judgement
- Subrule: Do not assume you are an idiot and the expert is a god
  - Mistrust expert judgement just like your own
- Subrule: Report symptoms, not theories
  - or you would reduce your chances of getting new insight
  - And if you are the helper: Don't get poisoned. Cover your ears and loudly sing LA-LA-LA-LA-LA.
- Subrule: Include observations you have not understood
  - If it is confusing for you, it may be just where somebody else can help

The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

If you don't fix it, ...: Used car war story

After buying a used car:

- Event 1: While going up a hill, suddenly the engine stopped
  - After stopping in the breakdown lane, the motor started again at the first attempt
  - During the slow drive up the rest of the hill, the car did not fail again
- Event 2: After filling up at a dubious little gas station on a bitter-cold day, the engine stopped again while going up a hill
  - Again, it started again at the first attempt
  - Driver thought: Maybe water in the fuel line. Applied drygas spray.

If you don't fix it, ...: Used car war story (2)

- Event 3: Engine stopped while driving fast on a perfectly flat road
  - Did not start on first attempt; but did start on the second
- Experimentation found that the engine would stop after going at more than 80 km/h for some short while
- Driver took the car to a repair shop
  - They replaced some wires and told him it was an electrical problem. Cost: 75 Dollars.
  - The car failed again the next day; just as before
- Idea: Maybe the carburettor does not receive enough fuel in high-load situations?
  - Get a fresh view: Asked a colleague at work.
    Answer: "Dirty fuel filter.".
  - The repair cost 50 cents.

If you don't fix it, ...: Subrules

- Subrule: Check that it's really fixed
  - You made it fail before, did you? Try again.
- Subrule: Check that it's really your fix that fixed it
  - After checking that it was fixed, take out your fix and make it fail
  - Note: Sometimes this is unecessary, too risky, or too cumbersome
- Subrule: It never just goes away by itself
  - If it stops failing without a proper fix, put in instrumentation to understand the failure next time
  - or analyze the differences to the failing version if you can
- Subrule: Fix the cause
  - Look behind the first-level cause of the failure and try fixing its root cause.

The nine rules: Summing up again

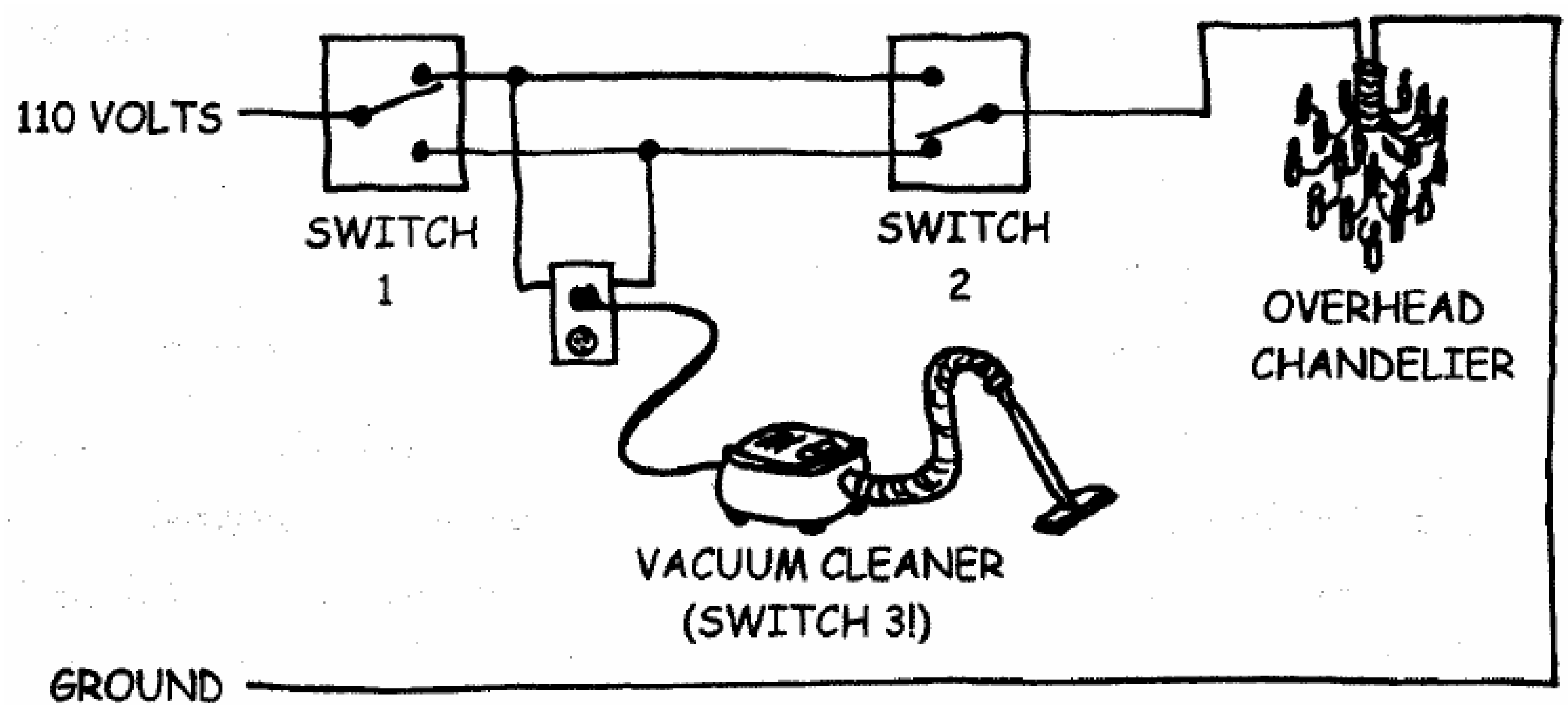
1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Small virtual debugging exercise

- Assume you move into a rather old house
- On the first day, your partner comes and tells you s/he tried to use the vacuum cleaner, but
  - when s/he switched it on, it did not start.
  - Instead, the room lights turned on.
- What do you do?

- **Now I am the house.**
- **Debug me!**

Solution to the exercise



Summary: The nine rules

1. Understand the system
   - even if that is hard
2. Make it fail
   - even if that is hard
3. Quit thinking and look
   - even if you think you know what is the matter
4. Divide and conquer
   - and do not jump to conclusions
5. Change one thing at a time
   - Even if it seems too simple
6. Keep an audit trail
   - in writing!
7. Check the plug
   - at least after a while
8. Get a fresh view
   - at least after a while
9. If you don't fix it, it ain't fixed
   - so fix it and make sure

Thank you!