

# It's a Bug's Life

Some musings on software bugs

Ian Dowse  
Ian.Dowse@corvil.com  
iedowse@FreeBSD.org



Eötvös Loránd Tudományegyetem,  
Budapest, Hungary  
November 20, 2010

# Overview

- Background
- Panic
- Bug probability space
- Bug finding pitfalls
- Some special bug types
- Q&A



# Background

- School of Mathematics, Trinity College Dublin



- Around 1995-2000, many servers moved from commercial Unix to FreeBSD on PCs
- Early days of FreeBSD, busy systems, lots of users
  - And we started seeing occasional kernel panics.
  - Many repeated crash patterns, and we tracked down and fixed quite a few bugs

- Work now in Corvil Ltd.



Corvil

- High performance, high tech latency management systems deployed in modern trading environments
  - During development, see very similar bug patterns

Panic: vrese negative ref cnt

# Panic: vrele negative ref cnt

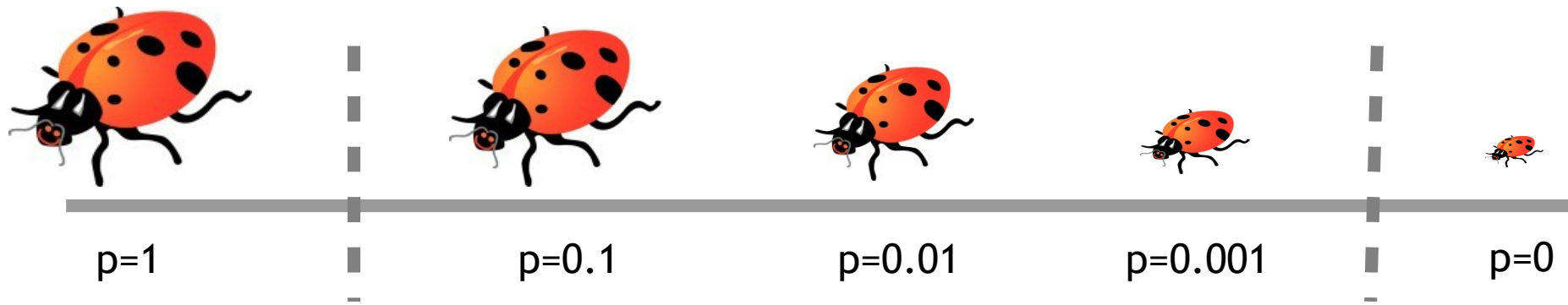
- Sept 1998 in Maths TCD we start seeing these panics
  - Means that vnode reference count has gone negative
    - Many many places in the kernel update vnode refs, so hard to debug
  - Some patterns emerged though
    - Mostly panic occurred on process exit on CWD vnode, always NFS
    - 1000+ users but dwmalone was convinced the problem was sometimes triggered by him logging out or sending mail
  - Eventually tracked it down
    - An incorrect vrele() call in the NFS filesystem when handling errors in the link() operation
    - Turned out dwmalone's mail client was attempting a cross-device link() on his NFS home directory, when sending mail, so the kernel did one vrele() too many
    - Then when he logged out, \*BAM\*

# Panic: vrele negative ref cnt

- The bug itself isn't particularly interesting, but some of the patterns are:
  - Right from the start there was more information than it seemed
    - There were many other FreeBSD bugs at the time, but this one caused maybe 95% of panics we saw
    - The problem began with an upgrade but we didn't notice
    - The correlation with logging out seemed unlikely, but was real
    - A few of the panic dumps actually had the NFS link() operation in the stack trace but we didn't make the connection
  - Everything about the panics made sense afterwards
    - When it started happening, when it triggered, and why it was almost always during logout
    - Even the probable root cause of the bug became obvious (the VOP\_LINK man page contained incorrect information)

Bug probability space

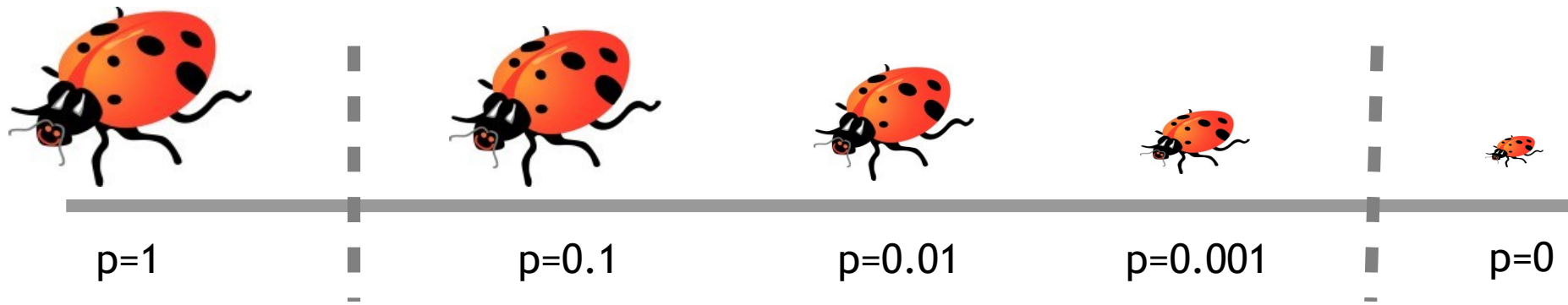
# Bug probability space



- Some bugs are certain (e.g. a crash is guaranteed every time)
- Some bugs are actually impossible to trigger
  - For example surrounding code prevents execution of the problematic case or operation
- Most remaining bugs are relatively rare
  - This has one quite interesting implication: the most commonly occurring bug will typically trigger far more often than the next most common one, e.g. 5 or even 100 times more often
- Of course the actual occurrence rate depends on usage patterns, so can be site-specific, user-specific etc.



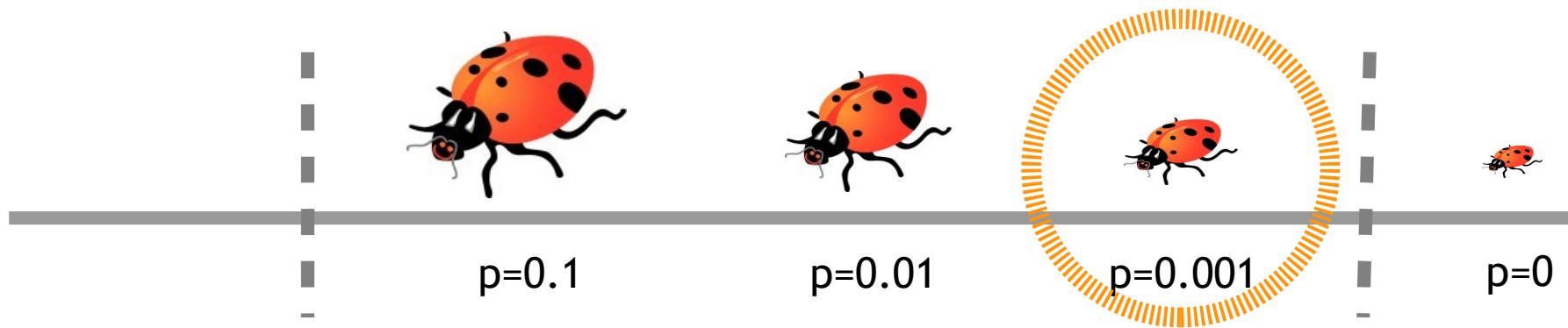
# Bug probability space



- Certain ( $p=1$ ) bugs are very common during development
  - Running any tests at all will catch these (e.g. does the program or system still work at all after the code change?)
- You'll often find impossible ( $p=0$ ) bugs when tracking down a problem.
  - It's easy to then think you've fixed the real problem but in fact you won't have changed the overall situation at all in practice
- Remember there are (generally) always more bugs out there
  - Need to be sure that the bug you fix is actually the one causing the main problem

# Bug finding pitfalls

# Ah, this must be it!




- After hours of searching for a bug, it's really easy to convince yourself that you've finally found it
  - Especially if you find what looks like a problem in related code, or find a problem that could confuse state or corrupt memory
- In many (most?) cases you actually get it wrong
  - Maybe you've just found an impossible bug or a much less likely one
- Need to be really sure the apparent bug matches the problem
  - Even obscure memory corruption bugs often cause surprisingly consistent crash patterns
- Humans maybe just aren't very good at finding bugs!

# How did this ever work?!

```
if (error) ;  
return (-1);
```

- Some bugs involve code that is blatantly wrong
  - Typos, cut & paste errors
  - Misplaced braces or semicolons
  - Uninitialised variables
  - Access past the end of an array or allocation
- Makes you wonder how the code ever worked at all
- Easy to forget to be logical when you find these
  - Just want to get rid of that awful mistake
- Fixes for this class of bugs actually cause a surprising number of fallout problems

# But how did it really ever work?

```
if (error)  i  
    return -1;
```

- One of the most interesting sides of bug fixing
  - You think “all bets are off”, “all hell will break loose” if code like this executes
  - But computers of course just continue logically
- If it did seem to work before, you can find out how and why
  - Maybe that code is never executed or nothing uses the results?
  - Maybe the compiler happened to use the same CPU register for the value you wanted and the uninitialised variable you actually used?
  - Maybe the corrupted memory is in practice “safe” to corrupt?
- Leads to a real sense that the bug is fully understood
  - And frequently to remarkably simple “how to repeat” steps

# Use the core dump

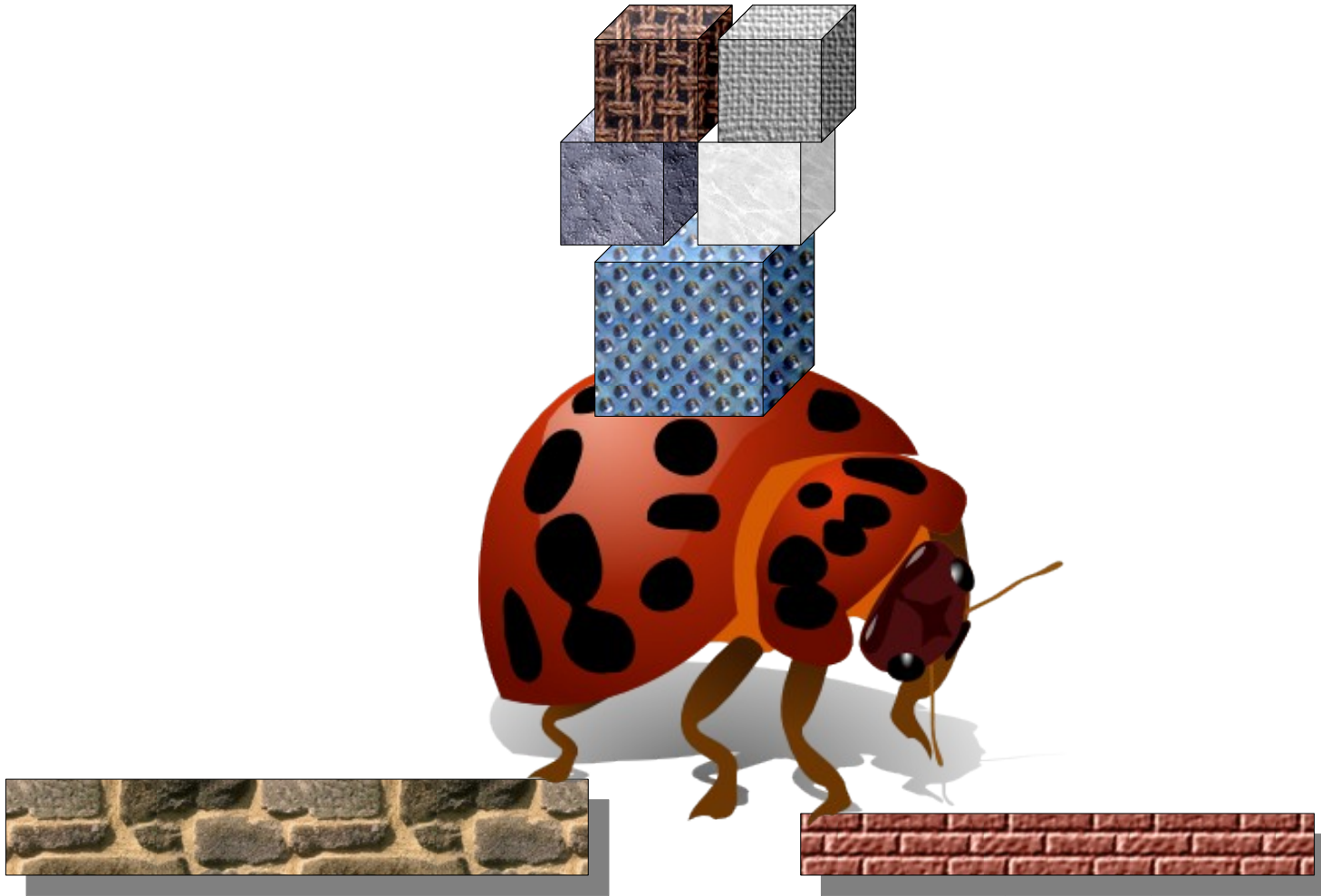


- Core dumps often appear to contain no useful information
  - The crash may occur long after the bug triggered
  - Maybe you don't have an executable with debugging symbols
- A bit of knowledge about assembly language, calling conventions and registers helps a lot
  - Even without debugging symbols you can find function arguments, local variables, structure contents etc.
- Often later after you've found the bug, you realise there was actually direct evidence of the problem in the core dump(s)
  - It's useful to remember this every time you start looking at a new bug

# Some special bug types

(suggested by work colleagues)

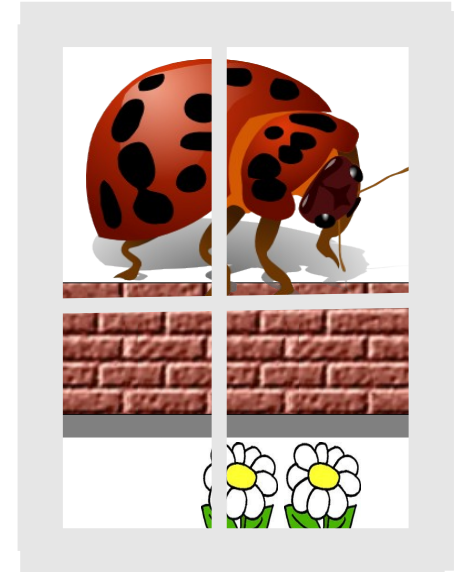
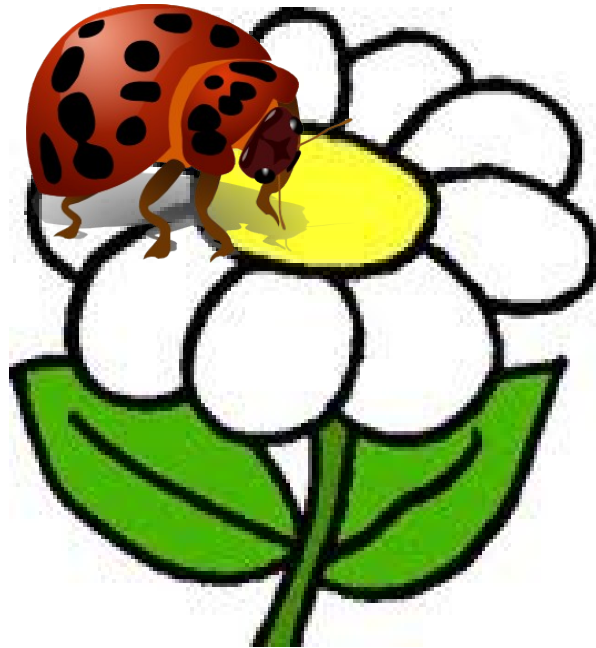
# The load-bearing bug



- You fix the bug and everything falls apart
  - E.g. A bug preventing a buggy optimisation from ever being applied



# Small or far away?



- Some bugs are small
  - Some are far away
- (Reference to Father Ted series)



Q&A

