fault injection in operating systems

why, where, since when, how, and now?

why

test specification (if any)

- ensure dependability attribute / OS goals
  - availability, integrity, confidentiality, isolation, …
- fault model
  - fail-stop? fail-crash?
- repair
  - roll-back, service degradation, reset, …
- assess severity
  - assess risk and adequacy of failures

example

seL4 kernel

sources & reading

high-assurance microkernel

formal specification & proofs

guaranteed run times

bug-free (against spec.)

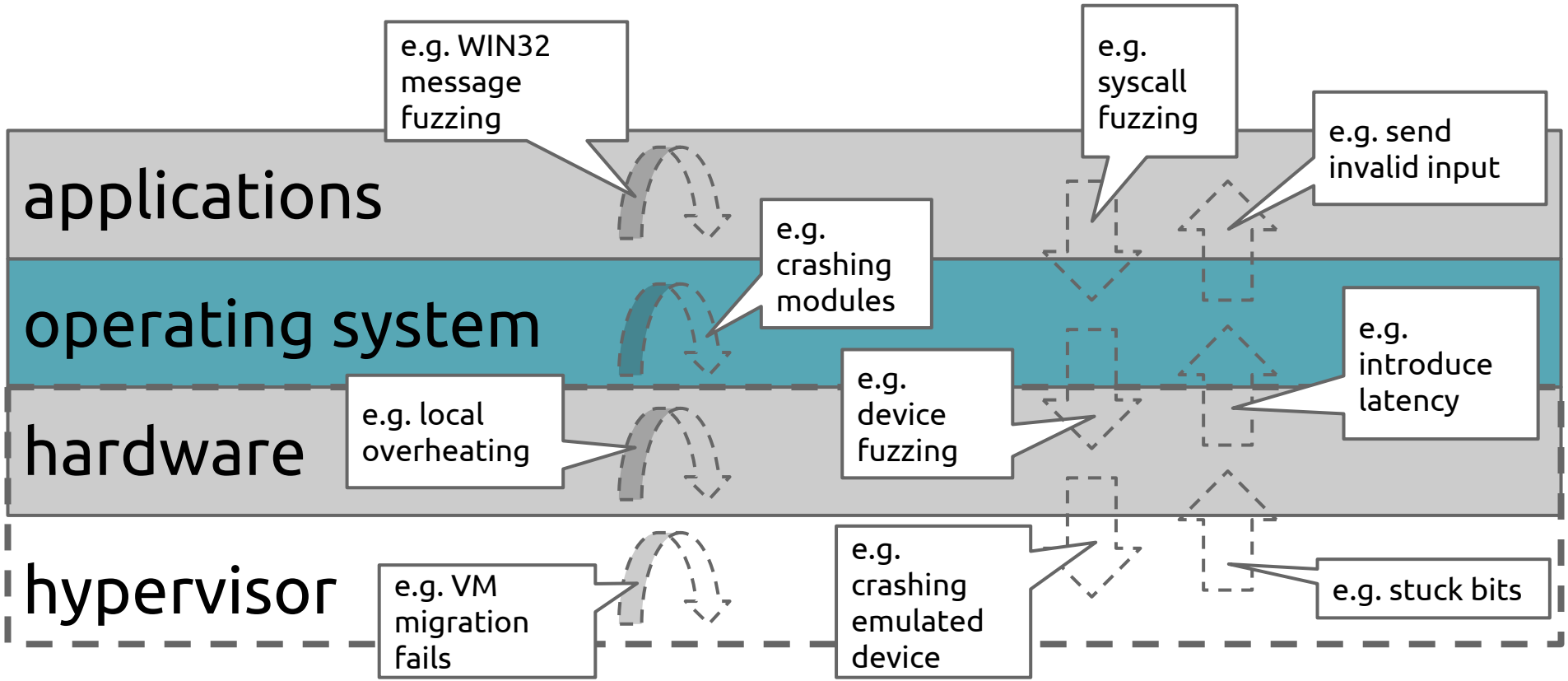
but how many OSs have a specification?

at least, minimal consensus seems to be

“it should not crash”

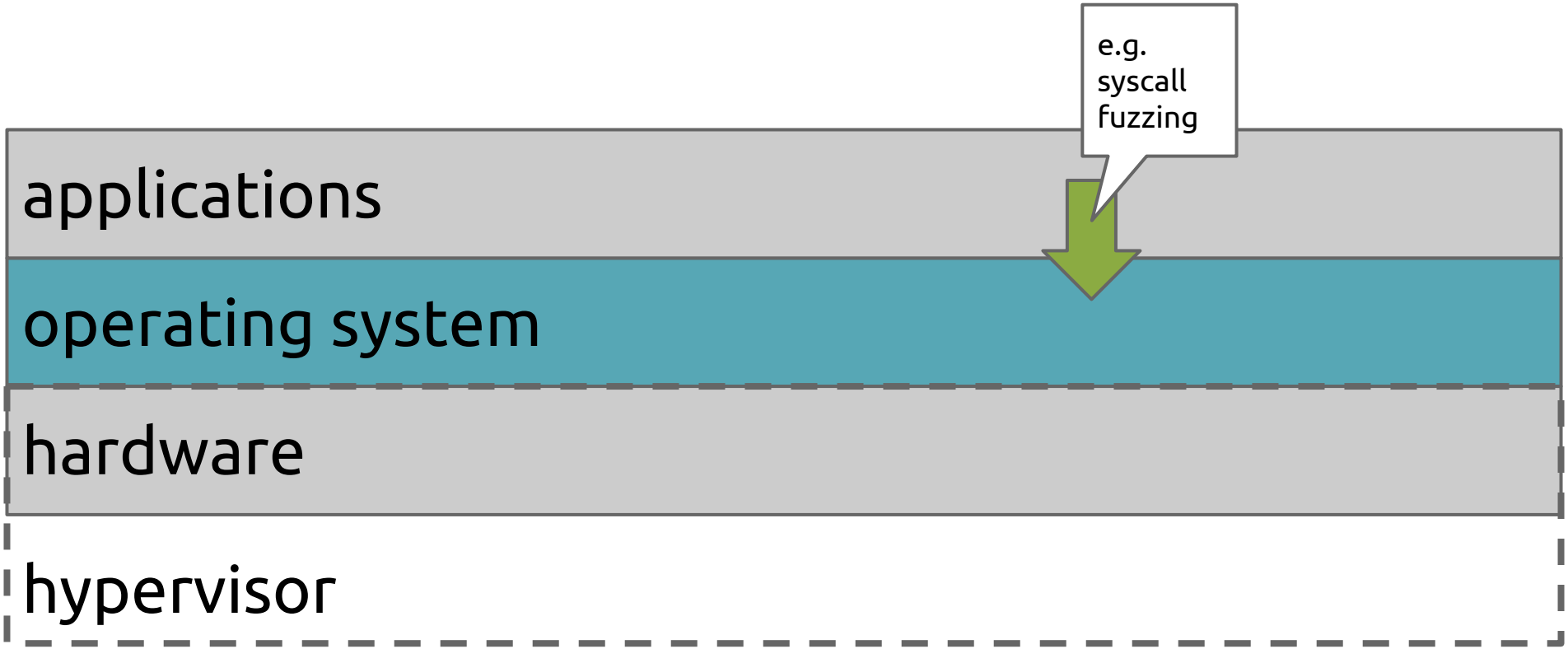
where

where to inject



where to inject

focus of this talk



since

when

Ball, M., and F. Hardie. "Effects and detection of intermittent failures in digital systems." Proceedings of the November 18-20, 1969, fall joint computer conference. ACM, 1969.
http://www.avizienis.info/

history

theoretical foundation

1969 hardware fault injection at IBM

simulated to evaluate integrity of logic units during design

faults: stuck transistors, open/shorted diodes

1970+ A. Avižienis: early theory on faults

coined “fault tolerance”, classification, modelling, …

OS support for fault-tolerant HW

⇒ foundation for injection & detection

history

fault injection in operating systems

1970 - 1993: **nothing!?**

fault **tolerance** in hard- and software

fault injection in **hardware**

hardware for fault injection in hardware…

} a lot

not applied to OSs or not explicitly mentioned?

what about **operating systems**?

history

fault injection in operating systems

Kao, Wei-Lun, Ravishankar K. Iyer, and Dong Tang. "**FINE**: A fault injection and monitoring environment for tracing the UNIX system behavior under faults." Software Engineering, IEEE Transactions on 19.11 (1993): 1105-1118.

Kanawati, Ghani, Nasser Kanawati, and Jacob Abraham. "**FERRARI**: A flexible software-based fault and error injection system." Computers, IEEE Transactions on 44.2 (1995): 248-260.

1993 FINE

fault-injection environment for UNIX

analyzed error propagation latency

software & memory: high, hardware: low

1995 FERRARI

flexible SW-based fault and error injection system

injects into operating system

Rathaus, Noam, and Gadi Evron. Open source fuzzing tools. Syngress, 2011.

1989: Prof. Barton P. Miller, University of Wisconsin

dial-up connection to campus computer

thunderstorm → noise on the phone line

random characters crashed UNIX utilities

→ let students write a random character generator

to test as much UNIX utilities as possible

tool called “**fuzz**”

early fuzzing

a lot of utilities crash

Miller, Barton P., Louis Fredriksen, and Bryan So. "An empirical study of the reliability of UNIX utilities." Communications of the ACM 33.12 (1990): 32-44.

Utility	VAX (v)	Sun (s)	HP (h)	i386 (x)	AIX 1.1 (a)	Sequent (d)
/lib/ccom				–	–	●
checkeq				–		
checknr				–	–	
col	●○	●	●	●○	●	●
colcrt				–	–	
colrm				–	–	
comm						
compress					–	
/lib/cpp						
csh	●○	○	○	–	○	○
dbx		*	–	–		
dc				○		
deqn		●	–	–	–	–
deroff	●	●	●		●	●
diction	●	–	●		–	●
diff						
ditroff	●○	●	–	–	–	
dtbl			–	–	–	–
emacs	●	●	○	–	–	
eqn		●	●	●		
expand					–	
f77	●		–	–	–	–
fmt						
fold					–	
ftp	●	●	●	–	●	●
graph					–	
grep						
grn			–	–	–	–
head					–	
ideal			–	–	–	–
indent	●○	●○	●	–	–	●
join		⊕				
latex			–	–	–	–
lex	●	●	●	●	●	●
lint						
lisp		–		–	–	–
look	●	○	●	●	–	●

Table 2: List of Utilities Tested and the Systems on which They Were Tested (part 1)

*● = utility crashed, ○ = utility hung, * = crashed on SunOS 3.2 but not on SunOS 4.0, ⊕ = crashed only on SunOS 4.0, not 3.2. – = utility unavailable on that system. ! = utility caused the operating system to crash.*

Utility	VAX (v)	Sun (s)	HP (h)	i386 (x)	AIX 1.1 (a)	Sequent (d)
prolog	●○	●○	●○	–	–	–
psdit				–	–	
ptx	–	●	●	○		○
refer	●	*	●	–	–	!●
rev				–	–	
sed						
sh				–		
soelim					–	
sort						
spell	●○	●	●	○	●	●
spline					–	
split						
sql		–			–	–
strings					–	
strip						
style	●	–	●		–	●
sum						
tail						
tbl						
tee						
telnet	●	●	●	–	●	○
tex			–	–	–	–
tr						
troff	–	–	–			
tsort	●	*	●	●	●	●
ul	●	●	●	–	–	●
uniq	●	●	●	●	●	●
units	●○	●	●	●	●	●
vgrind	●		–	–	–	
vi	●		●	–		
wc						
yacc						
# tested	85	83	75	55	49	73
# crashed/hung	25	21	25	16	12	19
%	29.4%	25.3%	33.3%	29.1%	24.5%	26.0%

Table 2: List of Utilities Tested and the Systems on which They Were Tested (part 2)

*● = utility crashed, ○ = utility hung, * = crashed on SunOS 3.2 but not on SunOS 4.0, ⊕ = crashed only on SunOS 4.0, not 3.2. – = utility unavailable on that system. ! = utility caused the operating system to crash.*

history

random testing / fuzzing

again, what about **operating systems**?

1991: *crashme*

by George J. Carrette

developed for nowadays outdated platforms

SUN-4, DECstation, IBM RT, Nixdorf, HP-UX, …

actually test operating system robustness

how

crashme

invokes random data as procedure

programmatic approach

```
calculate_random(seed, &rand);
(rand)();
```

no detection, monitoring, tracing, …

crashme

manually triggered during runtime (of kernel)

fault model

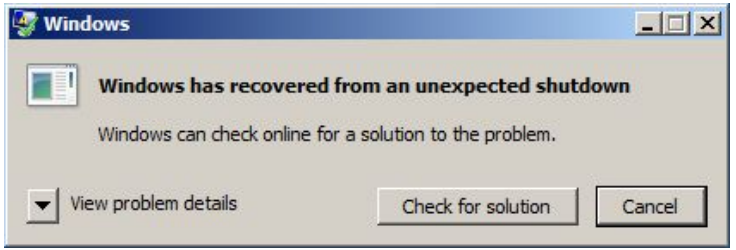
- crash fault
  - /!\ must be detected by user/tester

crashme

used for Linux kernel testing

- ~ since late 1996 (2.0.20)
- rare crash reports via mailing list
  - ~400 messages

crashme



theoretically runs on wide variety of OSs

ran on Linux, Windows 7, 8, Server 2008, 2012

crashed & rebooted: 7, 8.1, 2008; hung: 7, 8.1

no fault injection required for Win10: installer hung

no success on Windows XP and 9x

binary header produced by Visual Studio > 2005 forbid execution

installed Windows XP & Visual Studio 2005

compiles, runs, no errors, no output -.-

let’s take a look in the lab (demo)

implementations using system calls

- use API instead of instructions to inject
  - see Angelo Haller's slides
    - Trinity
      - found bugs and still actively used
      - grey-box testing
        - knows about sanity checks in kernel
    - iknowthis, sysfuzz
      - grey-box testing

implementations

using system calls: **ballista**

1998 - 2001

- originally intended as internet service (???)
- test generated & executed based on API descriptions
  - black-box: only knows interface
    - functions, parameters, data types
  - pretty similar to Trinity?
- aimed to be highly repeatable
  - detailed logging, reporting
- later implemented for POSIX C-API on 15 OSs
  - does not run on recent Linux (as noted on the Web site)

Koopman, Philip, and John DeVale. "Comparing the robustness of POSIX operating systems." Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on. IEEE, 1999.

implementations

using system calls: **ballista**

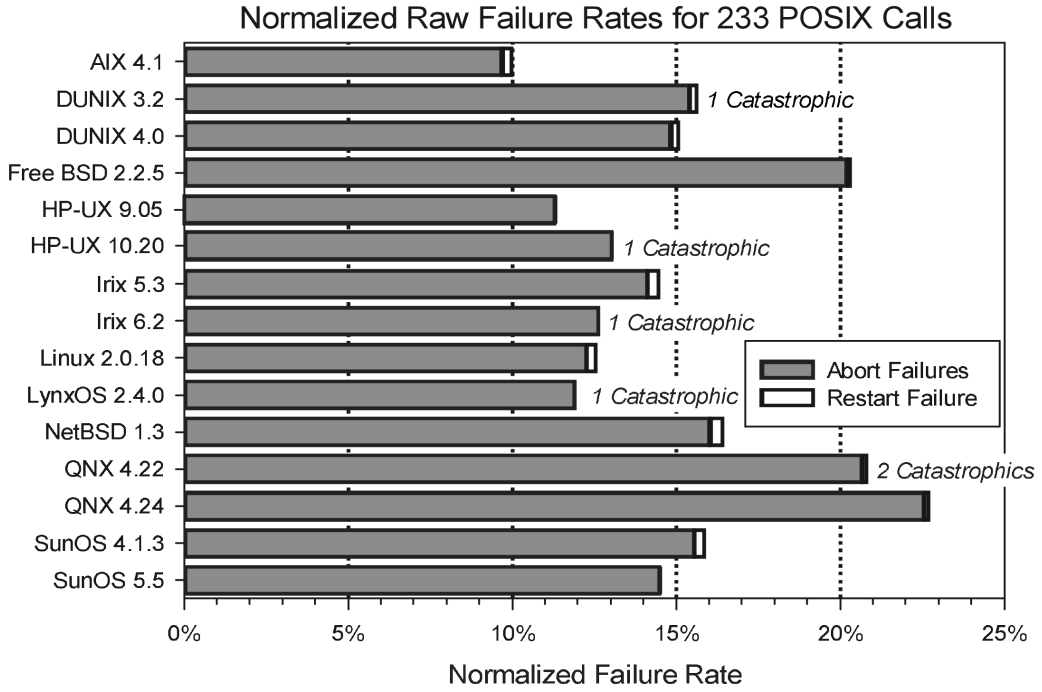


Figure 1. Raw robustness tests on 15 POSIX operating systems reveal a significant Abort failure rate and several Catastrophic failures.

commercial or OSS does not seem to correlate with quality

Koopman, Philip, and John DeVale. "Comparing the robustness of POSIX operating systems." Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on. IEEE, 1999.

implementations

using system calls: **ballista**

Table 1: Data types most commonly associated with abort robustness failures for 15 operating systems.

Data Value	Percent associated with robustness failures
Invalid file pointers (excluding NULL)	94.0%
NULL file pointers	82.5%
Invalid buffer pointers (excluding NULL)	49.8%
NULL buffer pointers	46.0%
MININT integers	44.3%
MAXINT integers	36.3%

so much crashes on expectable invalid input…

implementations

Ormandy, Tavis. "An empirical study into the security exposure to hosts of hostile virtualized environments." (2007): 1-18.
https://github.com/qemu/qemu/blob/master/docs/image-fuzzer.txt

iofuzz

- random input for IO devices
  - to test hypervisors
- actually found *serious* bugs in major products
  - ⇒ testing in VMs: not clear if host or guest crashed
- source not available
  - also via T. Ormandy directly (asked via email)
  - but probably easy to build (go! :))
    - QUEMU features a similar test
    - looks inspired by iofuzz

implementations

Linux Kernel

via debugfs

`/sys/kernel/debug/fail*`

currently mem & IO

configurable triggers

interval, probabilities, count

filterable by processes

extensible

some modules (e.g. NFS) come with extension

and

now?

random testing

non-determinism

- OS scheduler
  - workload
- interrupts
  - hardware
- memory positions
  - kernel address space randomization
- ⇒ hard/impossible to repeat

but: no domain knowledge required

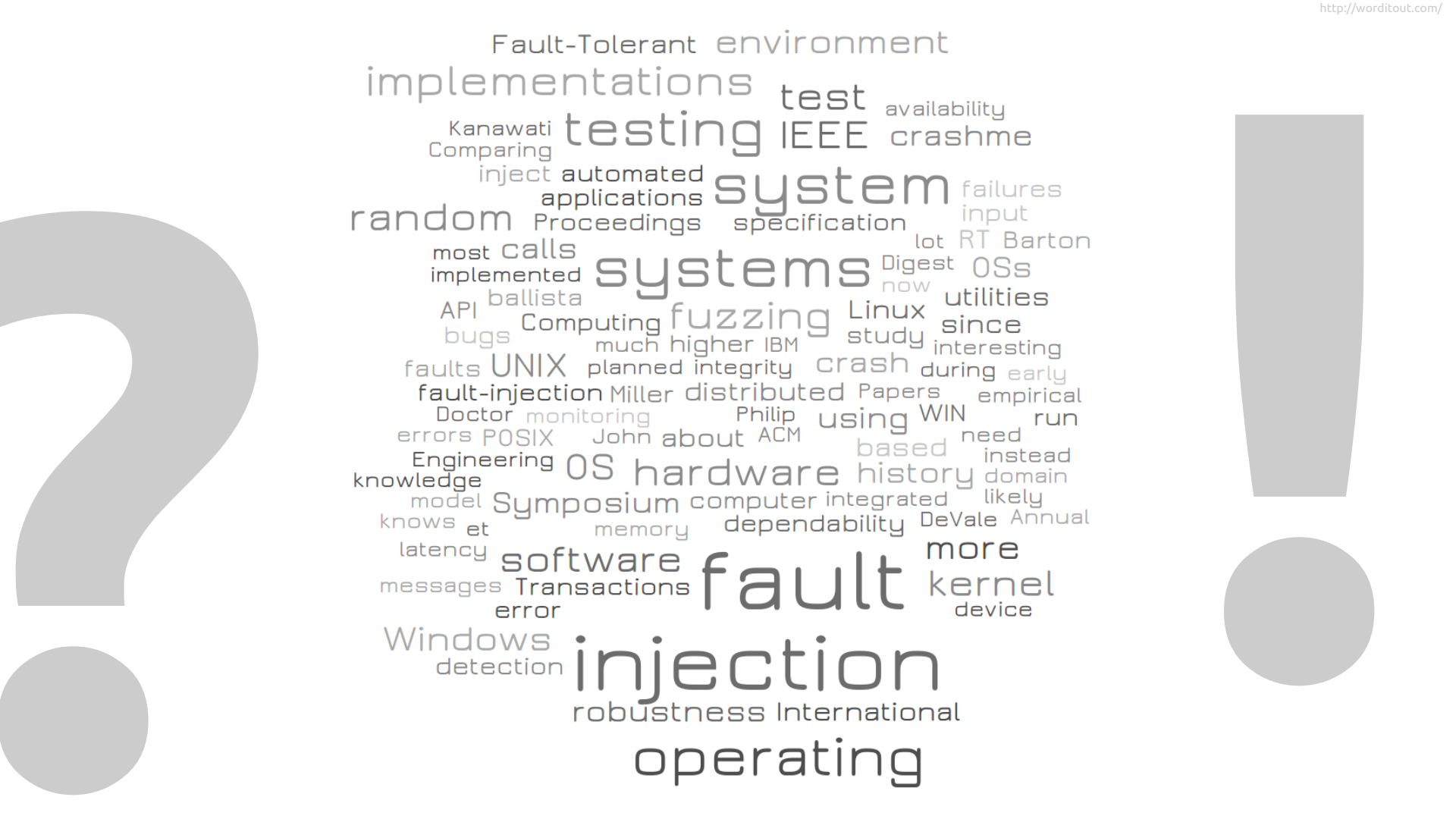
fault-injection tradeoffs

generic		vs	specific
higher re-use			higher impl. efforts
effective	(in exec.)		efficient
fuzzing			incl. domain knowledge
edge-case			usual-case
black box			white box

work ahead?

- do we need a taxonomy for the tradeoffs?
  - there are most likely more dimensions
- do we need a(nother) generic framework
  - that can be made more specific by extending it?
  - that can work on specifications?
  - that can automatically test virtualized?
  - that is implemented more high-level?
    - for higher development speed?
- can we learn from testing old OSs?

puh...



you now enter the slide graveyard…

history

fault injection in operating systems

1995 FIAT

fault injection based automated testing environment

distributed RT systems

planned injection in OS

1995 DOCTOR

SW fault injection env. for distributed RT systems

yet focussed on HW (CPU, mem, net) instead of OS

Segall, Zary, et al. "**Fiat**-fault injection based automated testing environment." Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years., Twenty-Fifth International Symposium on. IEEE, 1995.
Han, Seungjae, Kang G. Shin, and Harold Rosenberg. "**Doctor**: An integrated software fault injection environment for distributed real-time systems." Computer Performance and Dependability Symposium, 1995. Proceedings., International. IEEE, 1995

Sullivan, Mark, and Ram Chillarege. "Software defects and their impact on system availability: A study of field failures in operating systems." FTCS. 1991.

results more or less confirm intuition

common: overruns, wrong pointers, wrong allocs etc.

skim paper for tables for more information

Forrester, Justin E., and Barton P. Miller. "An empirical study of the robustness of Windows NT applications using random testing." Proceedings of the 4th USENIX Windows System Symposium. 2000.

interesting points

random WIN32 messages cause high failure rates

they can be sent from app to app!

even OS can crash

e.g. when generating very much HID input

UNIX more likely than Windows

Gu, Weining, et al. "Characterization of linux kernel behavior under errors." null. IEEE, 2003.

looks **really** elaborate & interesting

seems to

stick to establishes wording

explain kernel fault-injection API

do detailed fault analysis

Mukherjee, Arup, and Daniel P. Siewiorek. "Measuring software dependability by robustness benchmarking." Software Engineering, IEEE Transactions on 23.6 (1997): 366-378.

have good discussion about

- random pro/con
- properties of test tools
  - in an abstract manner

features a comparison of failure classifications