ISOLATING UNTRUSTED EXTENSIONS

SRG seminar, 15 Nov. 2007

University of Cambridge

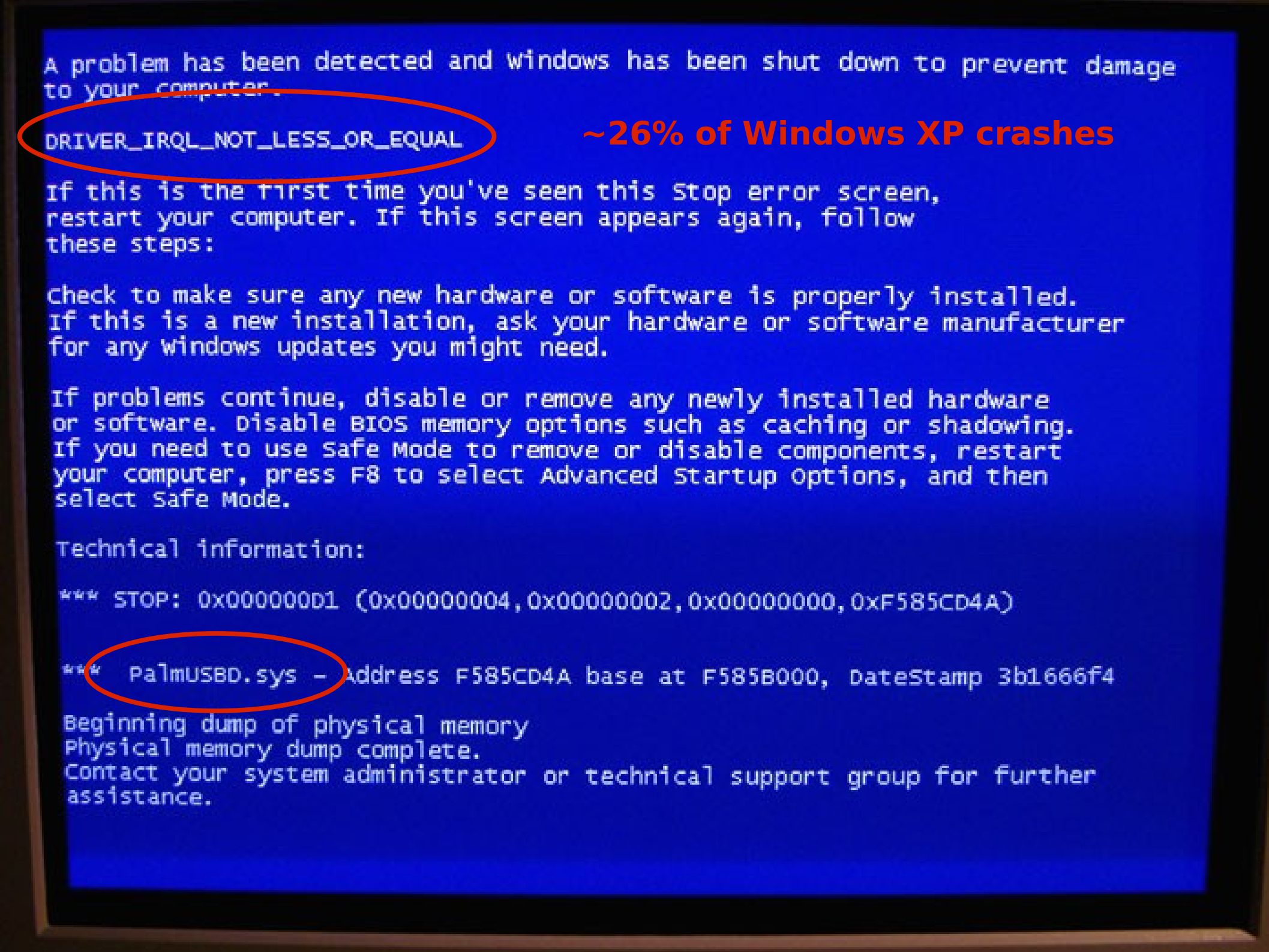
Jorrit N. Herder

Vrije Universiteit Amsterdam

Why is isolation needed?

"There are no significant bugs in our released software that any significant number of users want fixed."

-- Bill Gates, 1995


A problem has been detected and Windows has been shut down to prevent damage
to your computer.
DRIVER_IRQL_NOT_LESS_OR_EQUAL
~26% of Windows XP crashes
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x000000D1 (0x00000004,0x00000002,0x00000000,0xF585CD4A)
*** PalmUSBD.sys - Address F585CD4A base at F585B000, DateStamp 3b1666f4
Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.

It's not *just* a reboot

- Unacceptable for most (ordinary) users
  - grandma cannot handle computer crashes
- Big problem for large server farms
  - monthly reboot means many daily failures
    - even with 99% uptime a big problem
- Critical applications require reliability
  - ATMs, cars, power plants, etc.

Talk outline

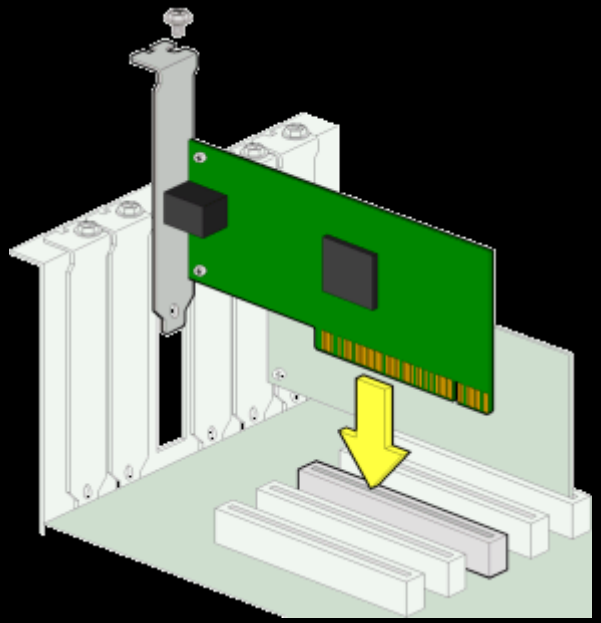
- Background and motivation
- How to improve dependability?
- MINIX 3 isolation architecture
- Experimental evaluation
- Discussion and conclusions

Talk outline

- Background and motivation
- How to improve dependability?
- MINIX 3 isolation architecture
- Experimental evaluation
- Discussion and conclusions

Even if OS were correct ...

- Plug-ins extend OS base functionality
  - provided by untrusted third parties
  - comprise up to 70% of entire OS
  - 3-7x more bugs than other OS code
- Still, extensions run in kernel
  - all powers of the system
  - no proper fault isolation



Software fault densities

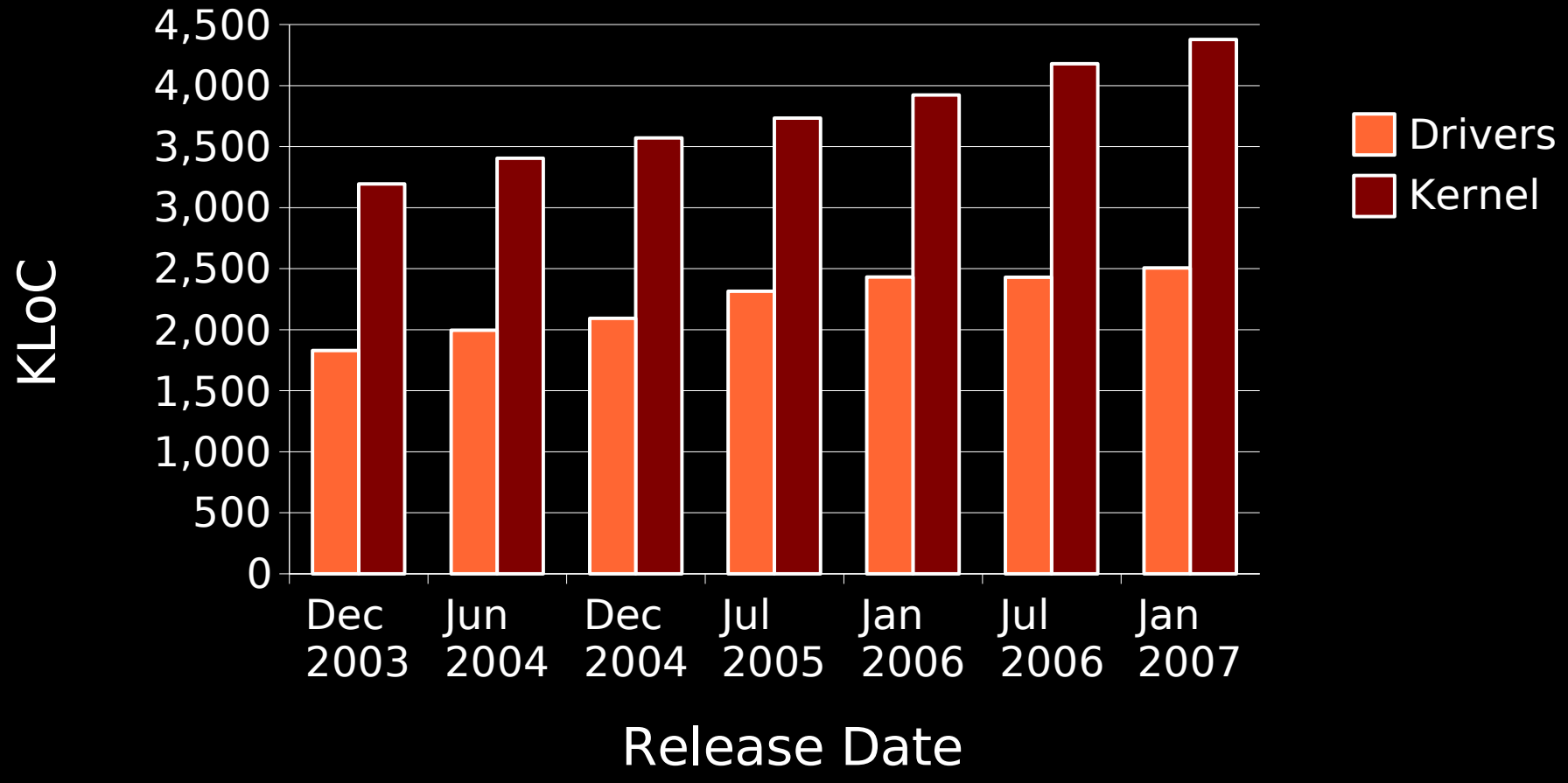
- Survey across many languages shows
  - ~6 bugs/KLoC for well-written software
  - ~1 bug/KLoC doable with best techniques
- Results of three independent studies
  - satellite planning system: 6-16 bugs/KLoC
  - inventory system: 2-75 bugs/KloC
  - FreeBSD: 3.35 post-release bugs/KLoC
    - surprising for high-quality open-source system

Bug fixing is infeasible

- Continuously changing configurations
  - 88 new drivers/day
    - old data from 2004, worse now!
- Code maintainability very hard
  - changing kernel interfaces
  - unwieldy growth of kernel code base
    - supported by our Linux 2.6 kernel analysis

Linux 2.6 kernel analysis

- In 3 years, 32% growth, 57% by drivers



Consequences

- Downtime mainly due to faulty software
  - over 50,000 kernel bugs in Linux/Windows
    - if kernel size estimated at ~5 MLoC
    - and fault density put at 10 bugs/KLoC
  - any kernel bug is potentially fatal

- Windows crash dump analysis confirms:
  - extensions cause 65-83% of all crashes

Talk outline

- Background and motivation
- How to improve dependability?
- MINIX 3 isolation architecture
- Experimental evaluation
- Discussion and conclusions

Isolating extensions

- Techniques can classified as follows:
  - in-kernel wrapping and interposition
    - Nooks, SafeDrive, XFI
  - language-based protection
    - Singularity, VFiasco, Coyotos
  - virtualization
    - VmWare, Xen, L4
  - multiserver designs
    - QNX, L4/NIZZA, MINIX 3

Isolation requirements

- Consensus on isolation. Good!

- Still, extensions need powerful constructs
- Two important questions remain:
  1) who can do what?
  2) how can this be done safely?

Powers of extensions

1) CPU instructions
2) Memory access
3) Device I/O
4) DMA access
5) IPC infrastructure
6) System services
7) Resource locks

- Applicable to all isolation techniques!

General approach

- Apply principle of least authority
  - least privileges necessary to perform task
    - limits damage that can result from error
    - helps preventing unwanted use of privileges
- Implies extensions have isolation policy
  - different extensions need different powers
- Requires restriction mechanisms
  - means to enforce least authority at run-time

Hardware considerations

- Problems with legacy PC hardware
  - lack of protection mechanisms
    - bus-mastering DMA
    - PCI bus conflicts and lockups
    - and more ...
- Hardware support enables safe designs

  a) dependability via I/O MMU, PCI-E, etc.

  b) fast CPUs make isolation practical

Talk outline

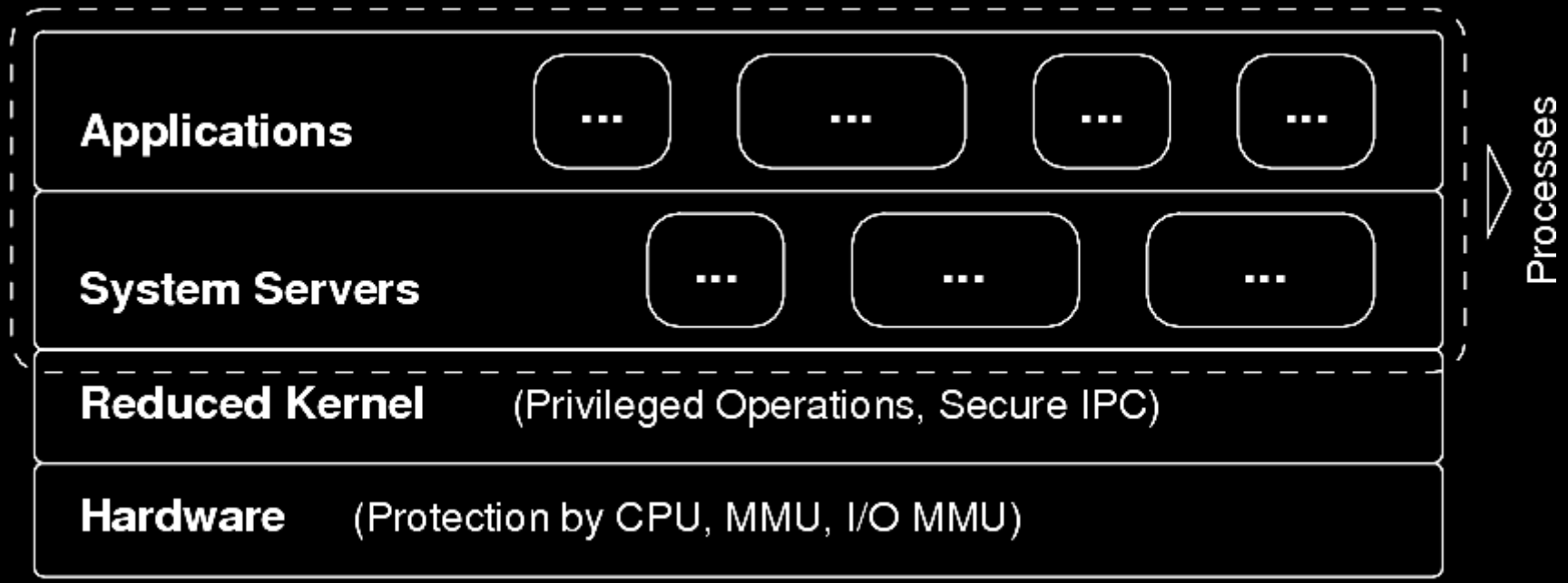
- Background and motivation
- How to improve dependability?
- MINIX 3 isolation architecture
- Experimental evaluation
- Discussion and conclusions

MINIX 3 rationale

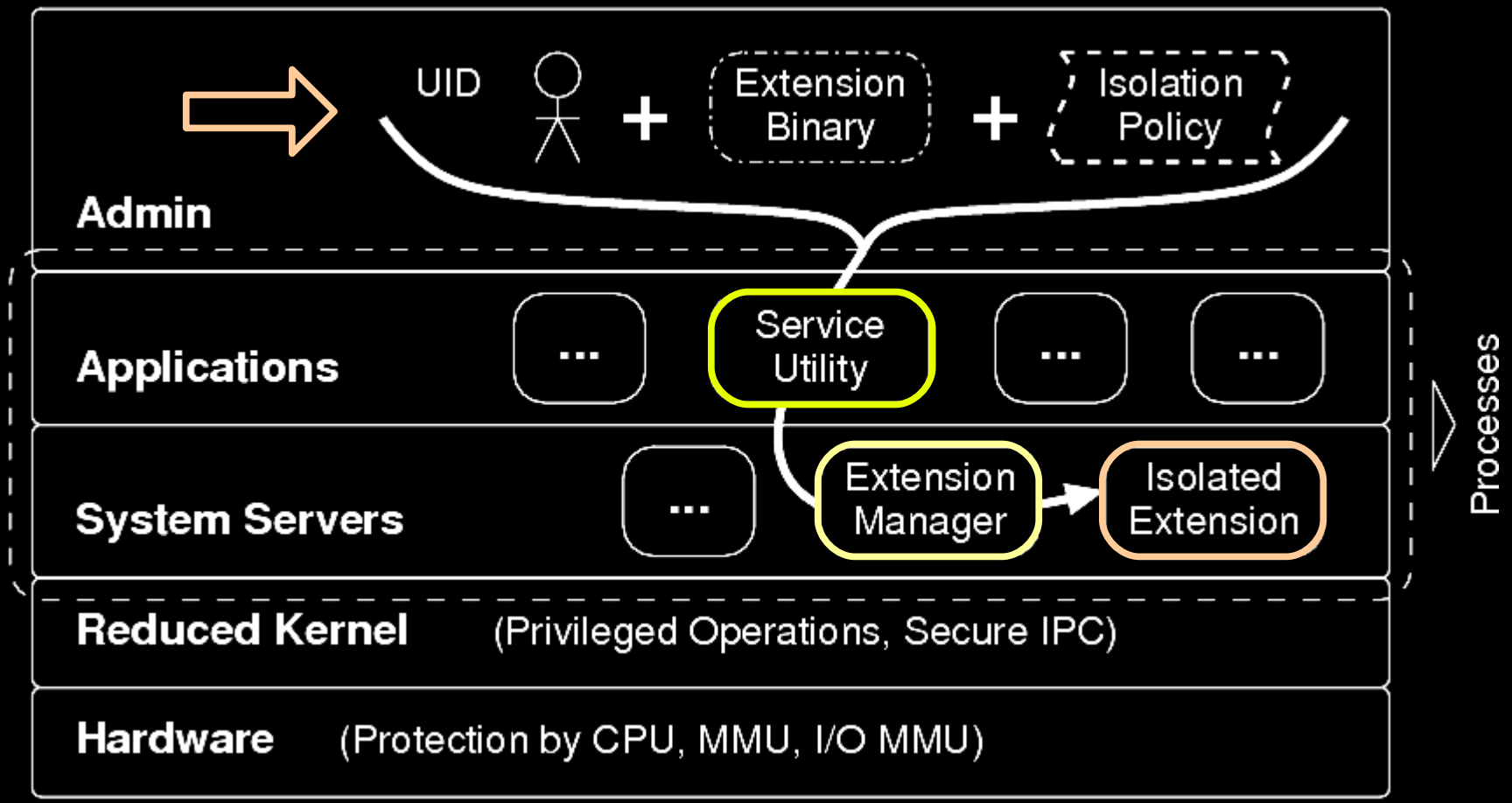
- Rethink OS design to improve reliability
- But keep well-known UNIX “look and feel”
  - MINIX 3 is POSIX compliant UNIX clone
  - runs about 500 standard UNIX applications
    - standard shell, file, and text utilities
    - language processors, ACK and GCC compiler
    - virtual file system infrastructure
    - TCP/IP stack with BSD sockets
    - X Window System

MINIX 3 multiserver design

- All OS services are user-mode processes
  - Process manager, virtual file system
  - Extension manager, file servers, drivers, etc.



Isolation in MINIX 3



Policy-based isolation

- MINIX 3 policies are simple text files
- Each key grants one or more powers
  - E.g., 'ipc' denotes allowed IPC destinations
- Suitable for dynamic resource discovery
  - E.g., restrict driver to a specific PCI device
    - PCI bus driver looks up device's resources

Example driver policy

```
driver rtl8029 {                    # isolation policy
    pci device      10ec/8029;      # PCI RTL8029
    ipc             pci             # PCI bus driver
                    kernel;         # Kernel task
    ipc kernel      SAFE_COPY       # Memory copying
                    SAFE_IO         # Device I/O
                    ...;
};
```

- Note: MINIX 3 provides only mechanisms

Loading an extension

- User sends request to extension manager
  - only privileged users may start extensions
- Actual loading and policy setting

  1) extension manager forks new process
     - IPC endpoint used to identify extension
  2) all OS servers are informed about policy
  3) finally process can execute binary

Restriction mechanisms

- Enforce least authority at run-time
- One restriction mechanism per power
    1) CPU instructions
    2) Memory access
    3) Device I/O
    4) DMA access
    5) IPC infrastructure
    6) System services
    7) Resource locks

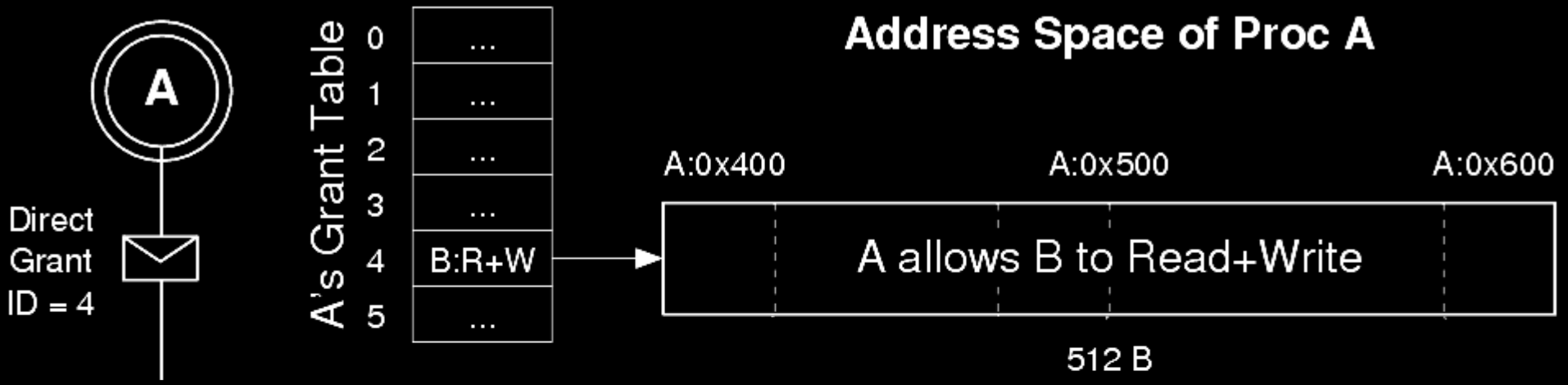
1) Deprivileging extensions

- Only microkernel has full CPU privileges
  - manageable due to small size < 5,000 LoC
- Extensions run as user-mode processes
  - immediate dependability benefits
    - cannot change page tables, halt CPU, etc.

2) Memory protection

- Process address spaces not enough
  - drivers typically need to exchange data
- Capability-like scheme: 'memory grants'
  - defines byte-granularity memory area
  - lists access rights for a specific process
    - immediate revocation by unsetting a flag
    - delegation supported via indirect grants
- Grant used with SAFE_COPY kernel call
  - kernel checks validity and copies data

Using memory grants



- Tell kernel about grant table location
- Add memory grant to allow access
- Pass index into grant table to grantee

3) Restricting device I/O

- Fine-grained device I/O access control
  - SAFE_IO kernel call checks isolation policy
  - memory-mapped I/O checked upon mapping
- User-mode IRQ handling
  - minimal, generic interrupt handler in kernel
    - structurally prevents IRQL_NOT_LESS_OR_EQUAL
    - responsible for 26% of all Windows XP crashes

4) Securing DMA access

- Prevent access to DMA controller
  - impractical for bus-mastering DMA
- Solution based on I/O MMU hardware
    - we used AMD's Device Exclusion Vector (DEV)
  - I/O MMU driver programs I/O MMU
  - allows only DMA to caller's address space
  - reset I/O MMU upon process exit

5) Safeguarding IPC

- Isolation policy restricts IPC capabilities
  - extensions can only contact few parties
- IPC design prevents resource exhaustion
  - no dynamic allocation of kernel resources
- Deadlocks prevented by IPC protocol
  - system servers should be nonblocking
    - always reply using nonblocking IPC
    - always use asynchronous IPC to call drivers

6) Shielding system services

- Individual request types further restricted
  - IPC not concerned with message contents
    - requests must be inspected by server
- Different implementations are used
  - kernel calls restricted via policy bitmap
    - most powerful calls restricted to trusted servers
  - several servers check client's user ID
    - request denied if not superuser

7) Denying resource locks

- Mutually exclusive resources do not exist
  - clients cannot hold locks and block others
- Each device can only have a single driver
  - enforced by extension manager upon loading
- Problems with legacy hardware
  - e.g., PCI has shared, level-triggered IRQ lines
    - virtually impossible to solve for any system

Talk outline

- Background and motivation
- How to improve dependability?
- MINIX 3 isolation architecture
- Experimental evaluation
- Discussion and conclusions

Fault-injection testing

- Goal: “Show that common OS errors in a properly isolated extension cannot propagate and damage the system.”
- Method: “Inject faults into an extension in order to induce a failure, and observe how the system is affected.”

Faults mutate binary code

1) change source register (source fault)
2) change destination register (dest. fault)
3) change address calculation (pointer fault)
4) ignore parameter given (interface fault)
5) invert termination condition of loops (loop fault)
6) flip a bit in an instruction (text fault)
7) elide an instruction (nop fault)
8) random fault selected from above types

- Shown to be representative for OS errors

Experimental setup (1/2)

- Fault injection targeted networking
  - largest driver population and fastest growing
    - in 3 years, networking code grew by 24%
    - now ~450 KLoC or 10% of Linux 2.6 kernel
- Injection into text segment at run-time
  - driver kept busy with TCP/IP requests
  - injected up to 100 faults per trial
    - ensures injected faults get triggered

Experimental setup (2/2)

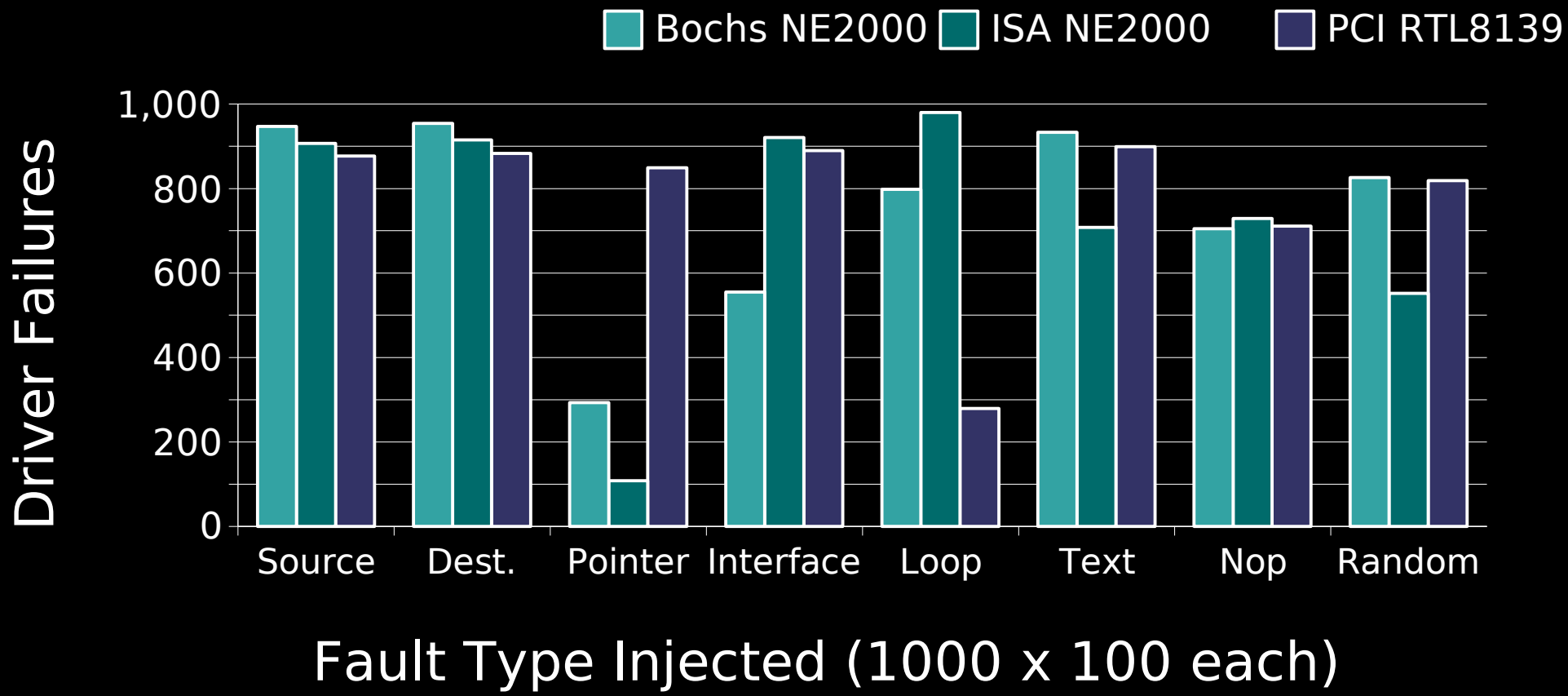
- Experiments done on 3 configurations:
  - emulated NE2000 card, Bochs v2.2.6
  - NE2000 ISA card, Intel Pentium III, 700 MHz
  - RTL8139 PCI card, AMD Athlon 64 X2 3800+
    - Device Exclusion Vector used to restrict DMA
- In total, we injected millions of faults
  - related work injected at most thousands
    - nasty bugs show up only after millions

Dependability results (1/2)

- One experiment injected 2,400,000 faults
  - 3 configs * 8 types * 1000 trials * 100 faults
- This induced 18,038 detectable failures
    - CPU or MMU exceptions: 10,019
    - exits due to internal panic: 7,431
    - missing driver heartbeats: 588
- Transparent recovery in all 18,038 cases
    - “Failure Resilience for Device Drivers,” Proc. 37th DSN, pp. 41-50, June 2007

Dependability results (2/2)

- Driver failed, but the OS never crashed

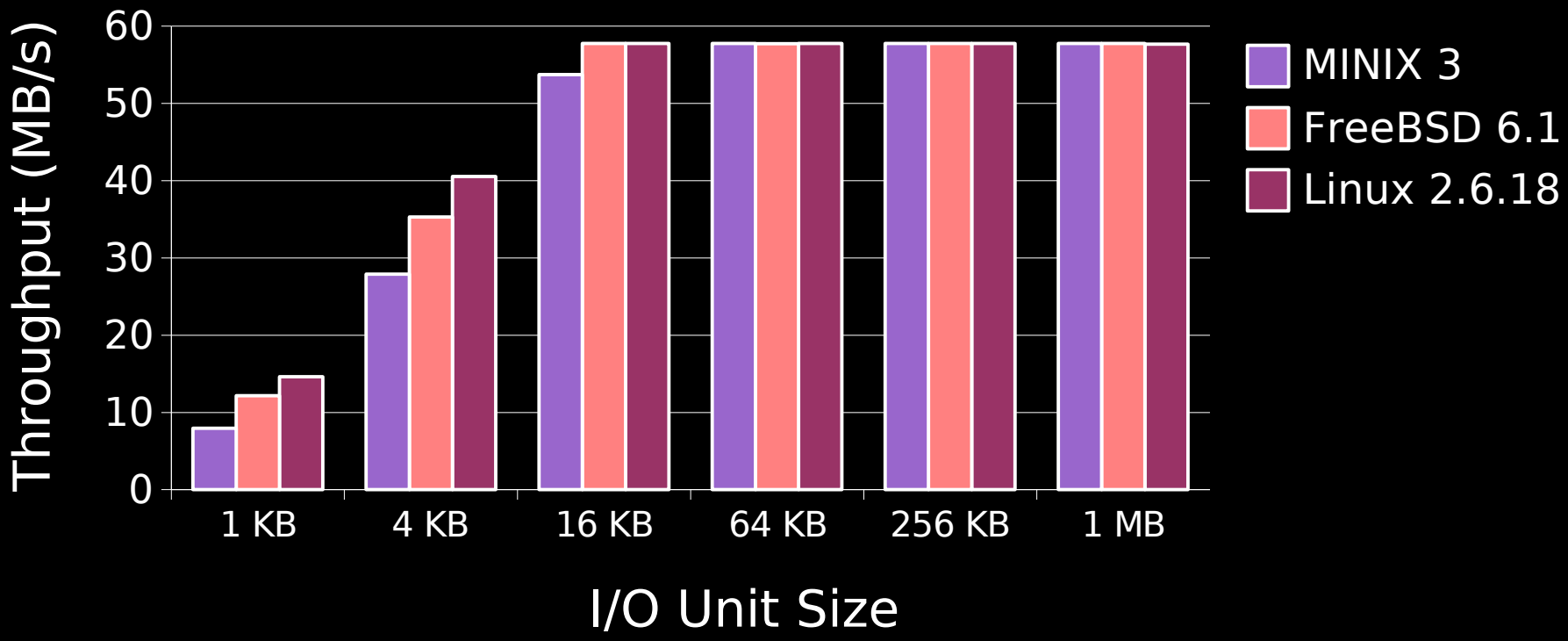


Performance considerations

- Modular design incurs some overhead
  - overhead can be limited to 5-10% range
    - L4 raw IPC performance (U Karlsruhe)
    - L4Linux measurements (TU Dresden)
    - SawMill Linux protocol design (IBM Watson)
    - Linux' user-level drivers (Nicta/UNSW)
- Our primary focus is dependability
  - nevertheless, MINIX 3 performs well
    - e.g., full system build in under 6 sec

Raw disk throughput

- Sequential read for various I/O unit sizes
  - highlights worst-case overhead



Engineering effort

- Statistics of executable source code

Component	# files	LoC	Comments
Fault injector	14	3,066	1,097
Extension manager	4	2,021	546
I/O MMU driver	1	329	10
PCI bus driver	3	2,798	339
VFS server	24	6,050	2,434
SATA driver	3	2,443	851
TCP/IP server	53	20,033	1,691
RTL8139 driver	1	2,398	345
NE2000 driver	3	2,769	424
Microkernel	54	4,753	2,600

Talk outline

- Background and motivation
- How to improve dependability?
- MINIX 3 isolation architecture
- Experimental evaluation
- Discussion and conclusions

Related work

- Various multiserver systems exist
  - e.g., Mach, SawMill, L4/Nizza, GNU Hurd, QNX
- Nevertheless, MINIX 3 first of its kind
  - strict interpretation of least authority
  - proven dependability through fault injection
- How about other isolation techniques?
  - “Can we make Operating Systems Reliable and Secure?”, IEEE Computer, pp. 44-51, May 2006

Wrapping and interposition

- In-kernel isolation represents a trade-off
  - backwards compatible with legacy OS
  - kernel-mode code may circumvent isolation
    - e.g., XFI is notable exception
- Works now, but not suitable for future
  - introduces additional kernel complexity
  - changing kernel APIs break wrapper code
  - new extensions require new wrappers

Language-based protection

- Novel approach for future systems
  - not backwards compatible with any code
  - new and untested programming model
  - complete formally verified OS nonexistent
- Potentially gives hard safety guarantees
  - trust in RTS, compilers and code verifiers
  - however, other protection is still needed
    - e.g., how to deal with memory bit flips?
    - I/O MMU required for bus-mastering DMA

Virtualization

- Run untrusted code in virtual machine
  - relies on hardware protection
  - excellent backwards compatibility
- In principle, can provide perfect safety
  - running entire OS in single VM not enough
  - run each extension in paravirtualized OS
  - inter-VM communication breaks isolation
    - starts to look like multiserver design

Multiserver designs

- Compartmentalize operating system
  - combines software and hardware protection
  - POSIX compatibility for applications
  - short development cycle for extensions
- Provides hard safety guarantees
  - in addition, works with other techniques
    - wrapping possible with IPC interpostion
    - incremental path for language protection
    - unmodified reuse through paravirtualization

Conclusions (1/2)

- Extensions threaten OS dependability
  - isolation of untrusted code is needed
- MINIX 3 employs a multiserver design
  - rethink OS internals to isolate extensions
  - but keep UNIX “look and feel” for users
- This work studied isolation techniques
  - classification of dangerous powers
  - software and hardware requirements

Conclusions (2/2)

- MINIX 3 design improves dependability
  - structural restrictions
  - per-extension isolation policy
  - dynamic protection mechanisms
- Fault-injection testing proves viability
  - targeted 3 Ethernet driver configurations
  - injected over 2,400,000 common OS faults
  - driver failed, but MINIX 3 never crashed

Future work

- Support stateful dynamic updates
  - replace buggy service with patched one
- Support stateful recovery
  - extend failure-resilience model

Acknowledgements

- Organisation
  - Steven Hand

- The MINIX 3 team
  - Ben Gras
  - Philip Homburg
  - Herbert Bos
  - Andy Tanenbaum

Thank you! Questions?

- Try it yourself!
  - Download MINIX 3

    - www.minix3.org
- More information:
  - Web: www.minix3.org
  - News: comp.os.minix
  - E-mail: jnherder@cs.vu.nl