
IBM Linux Technology Center
Confessions of a
Recovering Proprietary Programmer
Paul E. McKenney
IBM Distinguished Engineer & CTO Linux
Linux Technology Center / Linaro
August 1, 2011
Copyright © 2011 IBM

Overview

- What Does Paul Know About Open Source?
- Parable of Six Penguins and the Elephant
- Coding Style
- Source Code Management
- Summary

What Does Paul Know About Open Source?

Who is Paul and How Did He Get This Way?

- Grew up in rural Oregon
- First use of computer in high school (72-76)
  - IBM mainframe: punched cards and FORTRAN
  - Later ASR33 TTY and BASIC
- BSME & BSCS, Oregon State University (76-81)
  - Tuition provided by FORTRAN and COBOL
- Contract Programming and Consulting (81-85)
  - Building control system (Pascal/z80)
  - Security card-access system (Pascal/RT11/PDP-11)
  - Dining hall system (Pascal/RT11/PDP-11)
  - Acoustic navigation system (C/BSD2.8/PDP-11)

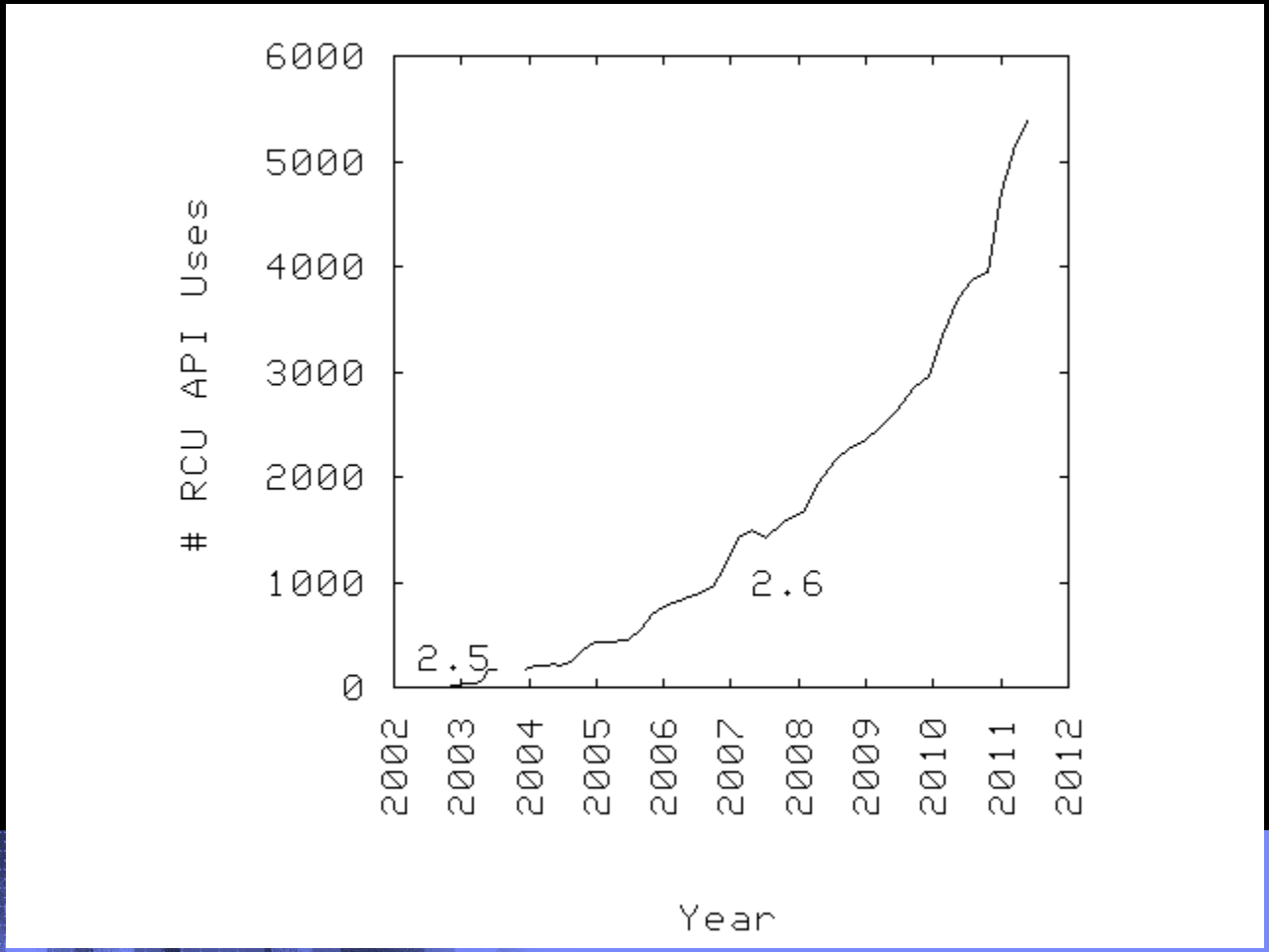
Who is Paul and How Did He Get This Way?

- SRI International (85-90)
  - UNIX systems administration (BSD/Pyramid90x)
  - Packet-radio research (C/SunOS/68K)
  - Internet protocol research (C/SunOS/SPARC)
- Sequent Computer Systems (90-00)
  - Communications performance (C/DYNIX-ptx/x86)
  - Memory allocators, TLB, RCU, timers, ...
- IBM (00-present)
  - NUMA-aware and brlock-like locking primitive in AIX
  - RCU maintainer for Linux kernel

What Does Paul Know About Open Source?

- Early member of IBM's Linux Technology Center
  - Helped define IBM's open-source strategy
- Active contributor to the Linux kernel:
  - 379 patches accepted into mainline since 2005
    - 1878 from gregkh, 2185 from tglx, 2592 from viro, 3395 from mingo, 3841 from davem, 10,411 from torvalds
  - Maintainer of read-copy update (RCU)
- Recognized expert in Linux community for concurrency, memory ordering, and RCU
  - One of a very few people to invent a synchronization primitive (RCU) with order-of-magnitude performance benefits that has been accepted into the Linux kernel
  - Numerous concurrency experts won't be forgiving him for this any time soon... :-)

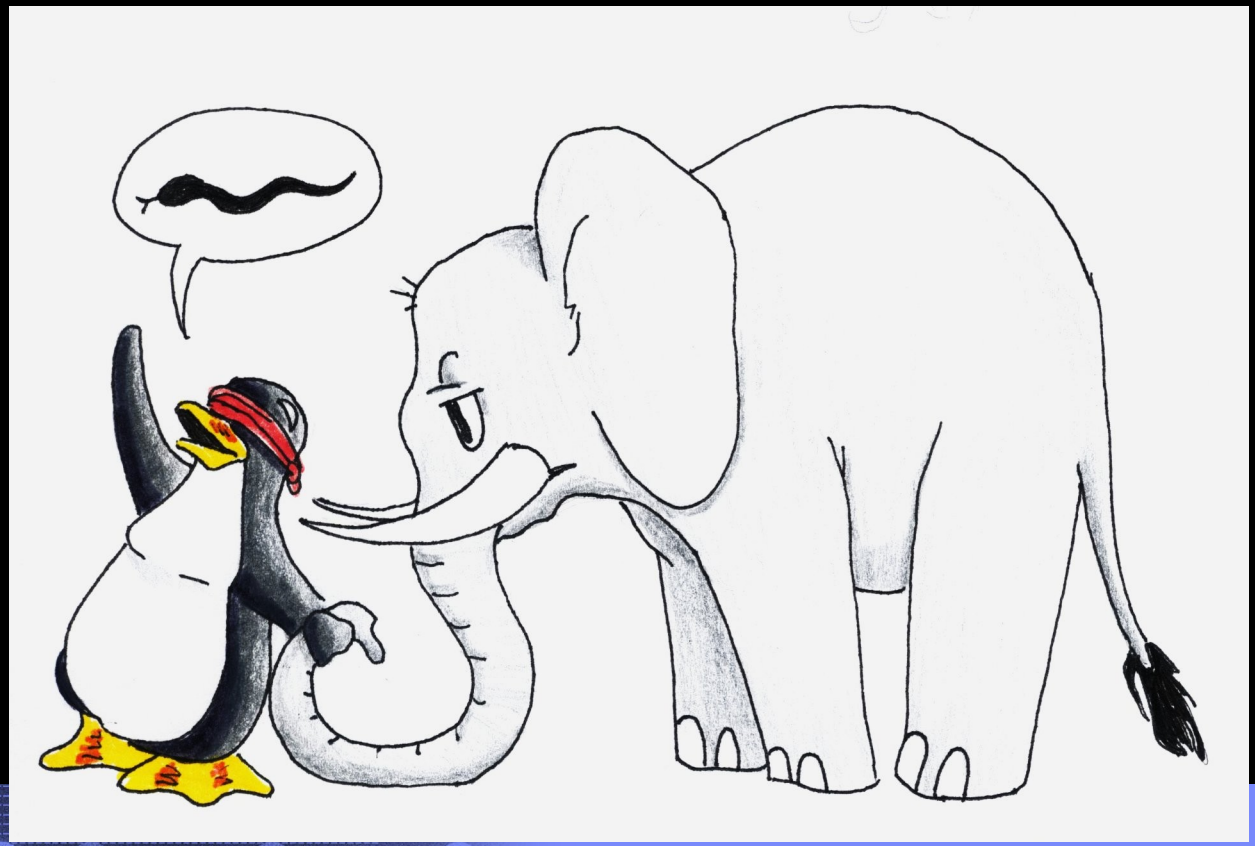
How Much is RCU Used in the Linux Kernel?



Paul is a Recovering Proprietary Programmer

The Parable of
The Six Blind Penguins and the Elephant

Proprietary Programming: Requirements

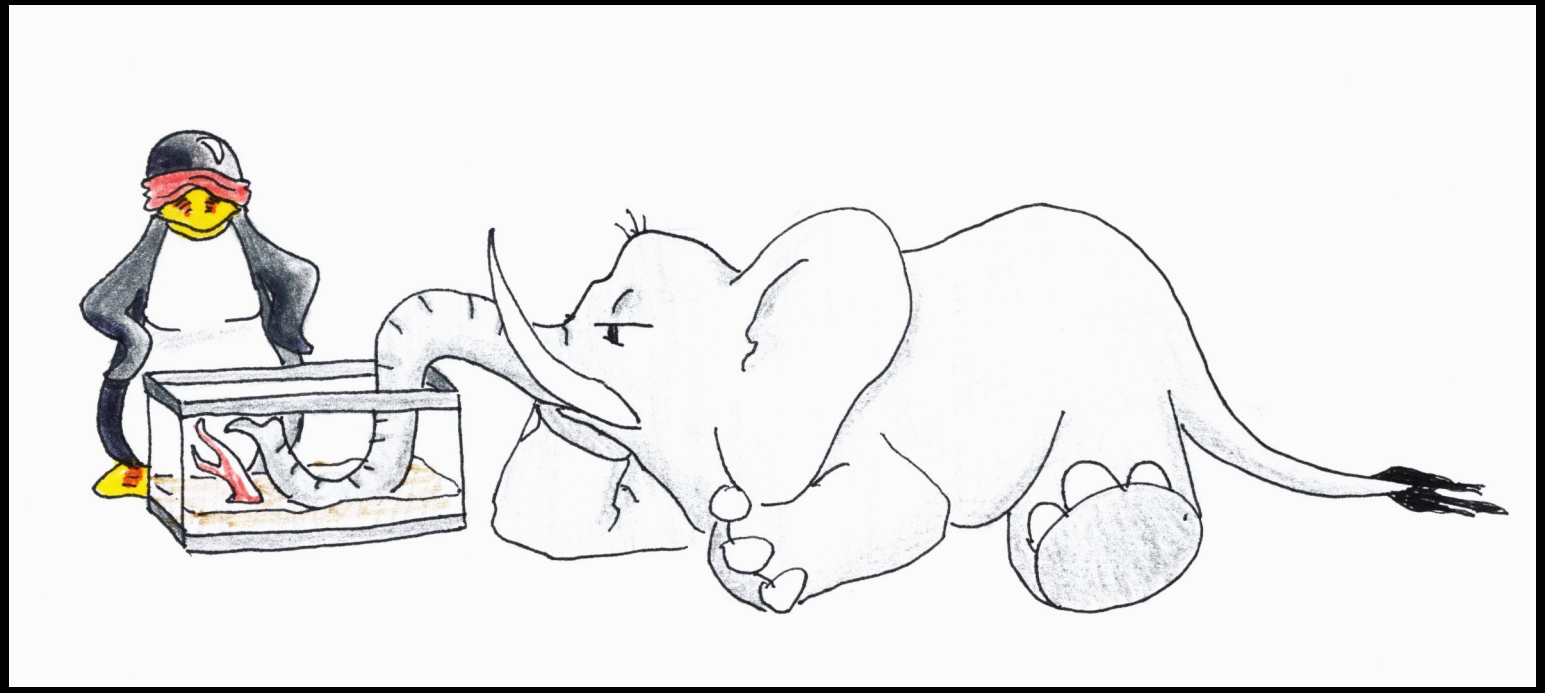


Proprietary Programming: “Solution”

Example: DYNIX/ptx RCU Implementation

- In late 1990s, I knew everything there was to know about RCU:
  - RCU read-side critical sections
    - "Free is a *very* good price!!!"
  - RCU grace periods
  - RCU quiescent states: context switches, CPU idle, syscall entry, trap entry, CPU offline
  - rcu_read_lock(), rcu_read_unlock(), read_barrier_depends, call_rcu(), kfree_rcu()
  - RCU application to lists, hash tables, trees, mode change, and waiting for ongoing interrupts
  - Impressive performance and scalability benefits for UNIX-based database servers
    - 64 CPUs SMP, 256 CPUs clustered

Proprietary Programming: “Solution”



But sooner or later...

The Entire Elephant Will Make Itself Known...

What I Didn't Know About RCU in the 1990s:

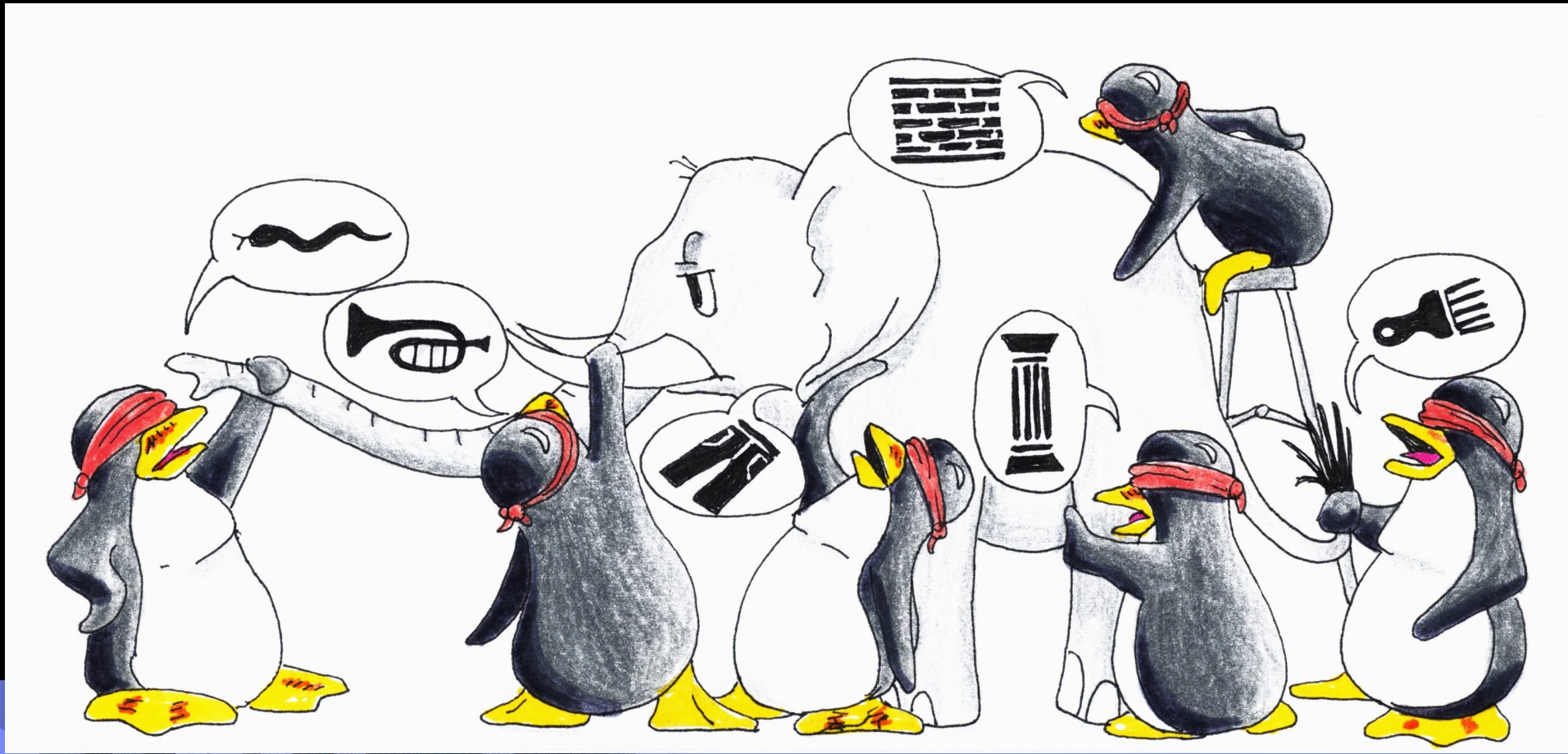
- DoS attacks
- Energy conservation
- Real-time response
- Sleeping RCU readers
- Wait for callbacks: rcu_barrier()
- RCU list primitives
- Burying memory barriers into RCU primitives
- Handling DEC Alpha
- Handling value speculation
- RCU semantics
- RCU proofs of correctness
- Runtime RCU validation
- Static RCU validation
- Handling more than 64 CPUs
- RCU priority boosting
- Early-boot RCU uses
- RCU tracing
- RCU and type-safe memory
- User-level RCU
- Multi-tail callback lists
- rcutorture
- Single-CPU implementations
- RCU-protected atomic list move
- Resizable RCU hash tables with wait-free readers
- Workqueue-based RCU
- Expedited grace periods

What does the red font signify?
What does the yellow font signify?

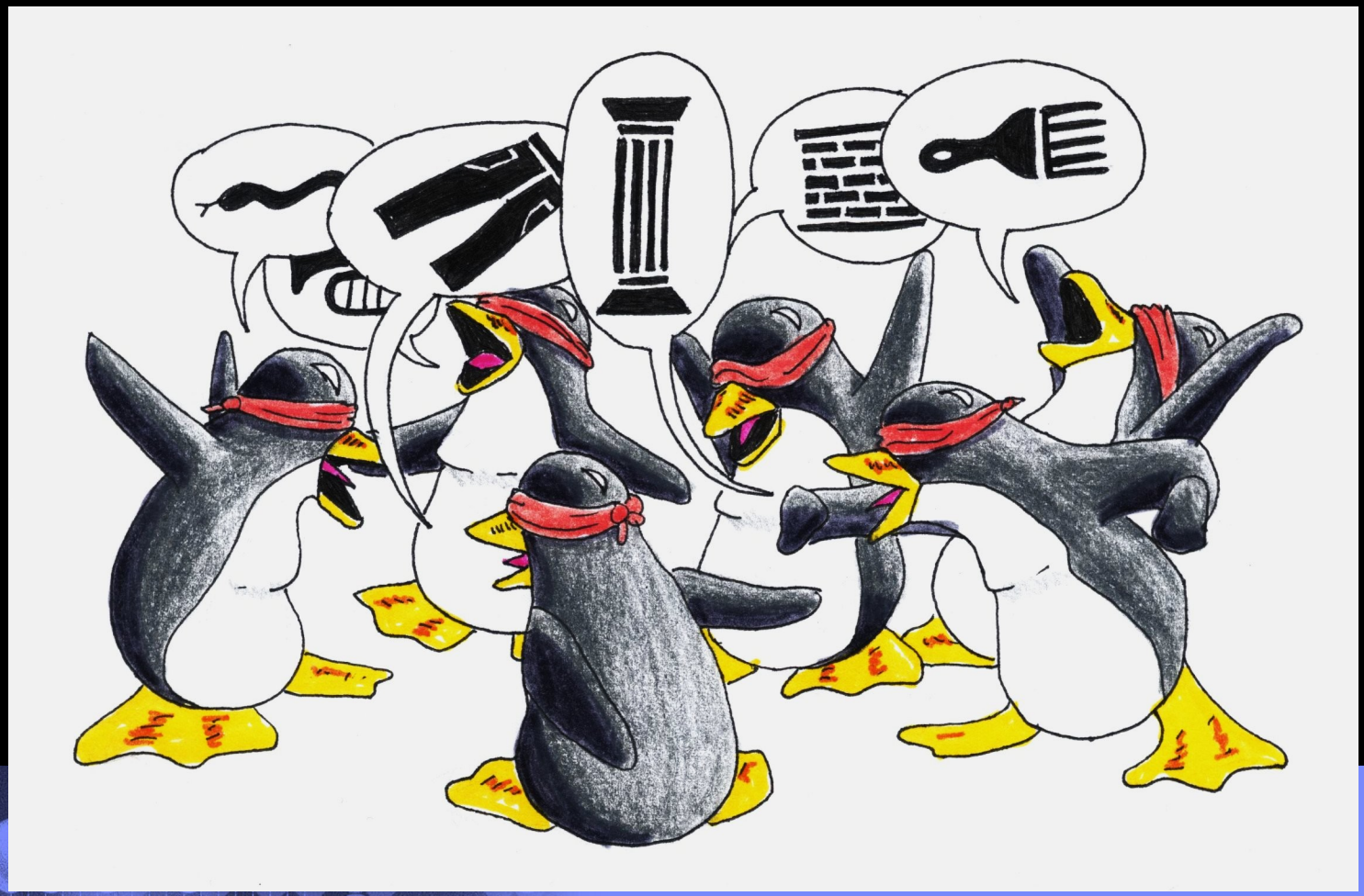
So What Happened?

- Yes, I was and am the world's expert on RCU
- But I learned a lot about RCU from newbies in the Linux community
- It was well worth wading through their naïve and silly suggestions to get the benefit of some extremely valuable ideas
  - Which are listed in red on the previous slide
    - Many of which I would never have thought of
  - The yellow items are things that I implemented, but in response to situations brought to my attention by RCU newbies
    - Situations that I never would have imagined myself: "why would you need *that*?"

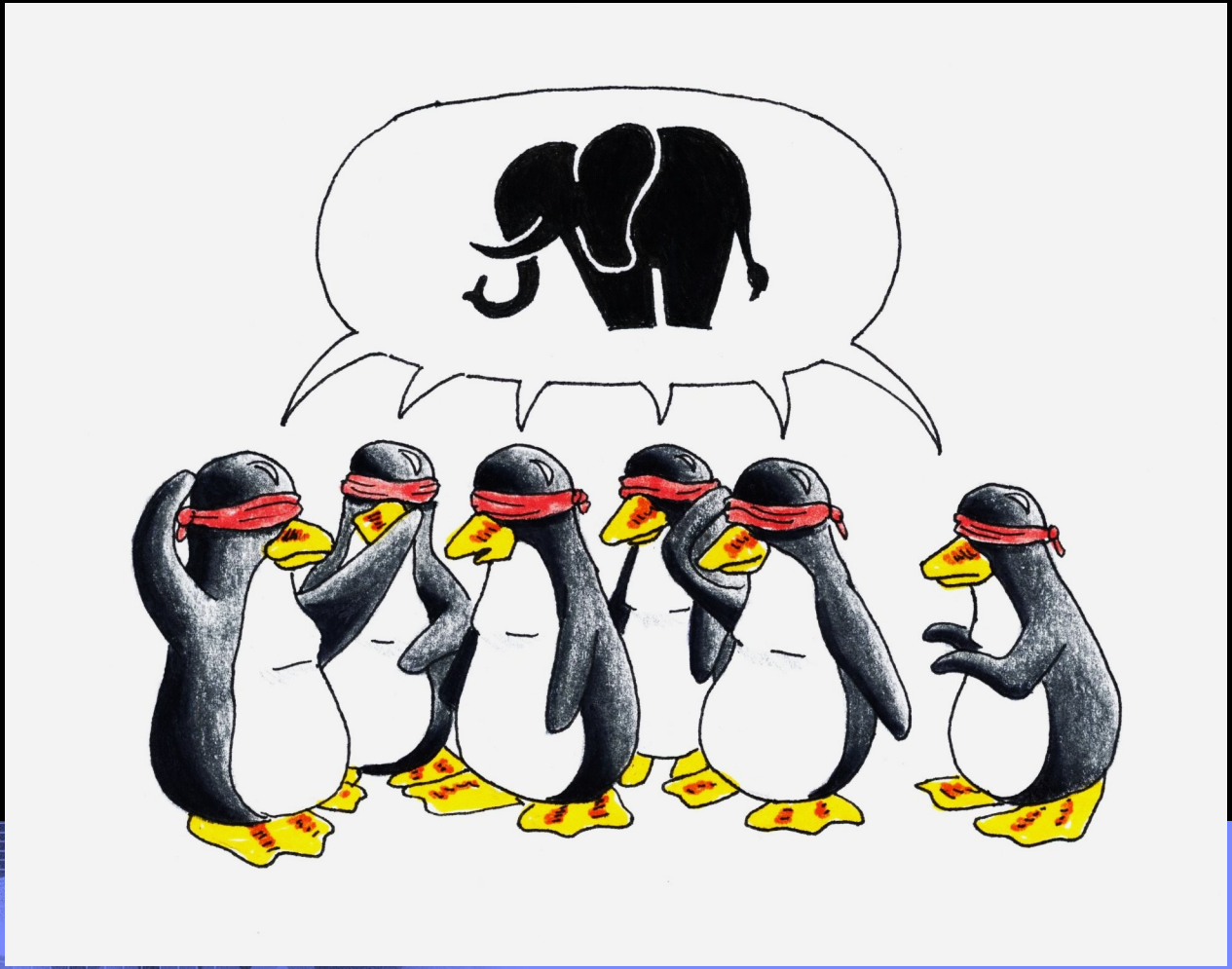
FOSS Programming: Requirements



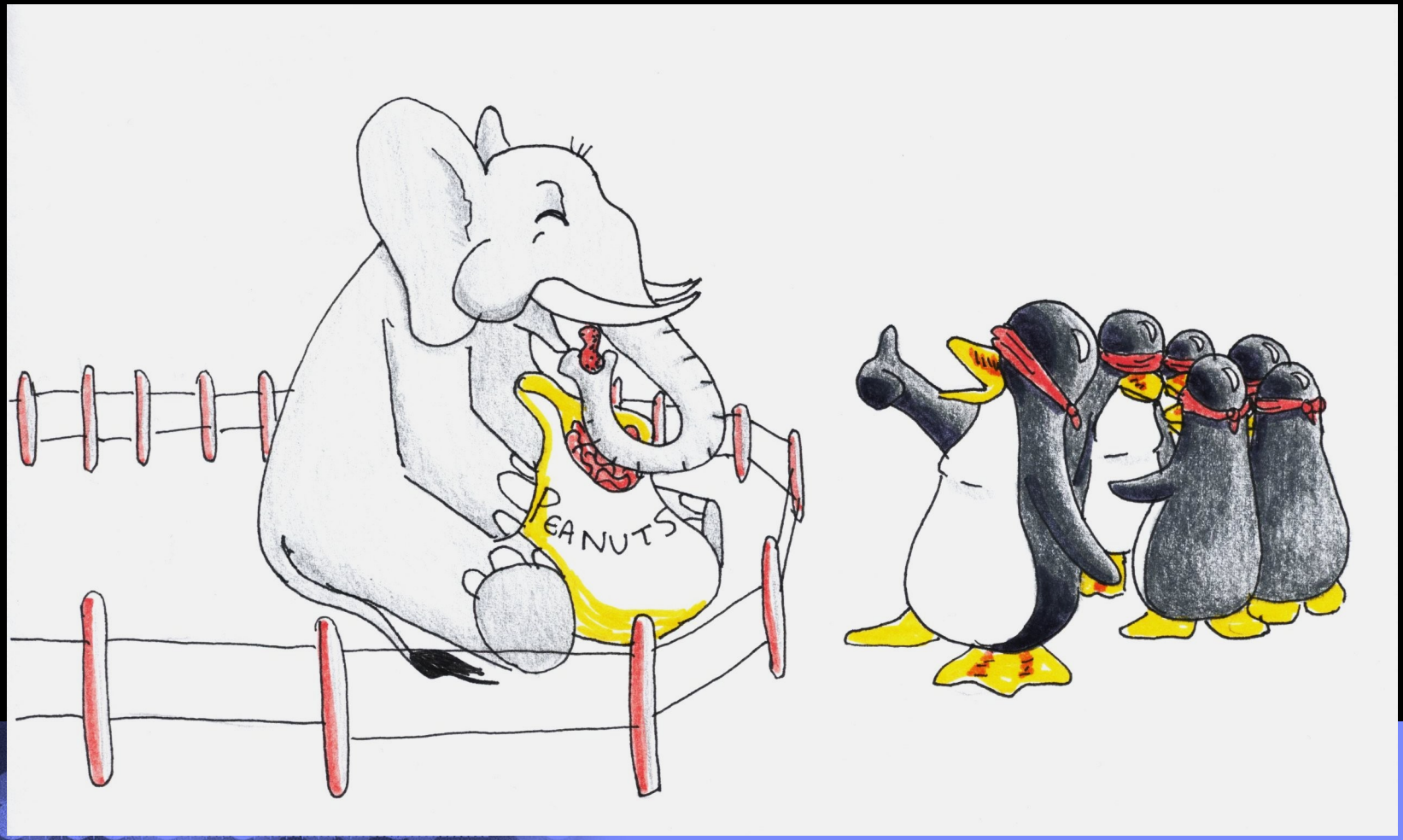
Just Another Day on LKML...



But Sometimes Consensus is Achieved



And an Appropriate Solution Produced Thereby

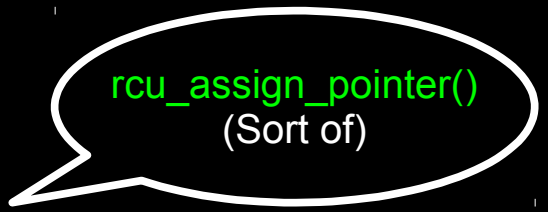


This is RCU in DYNIX/ptx

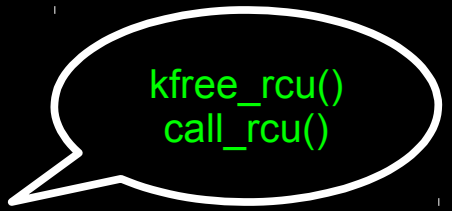
- rcu_read_lock()
- rcu_read_unlock()
- rcu_assign_pointer() [Sort of]
- kfree_rcu()
- call_rcu()
- list_first_entry_rcu()
- list_for_each_entry_rcu()
- list_for_each_continue_rcu()
- list_for_each_entry_continue_rcu()
- hlist_first_rcu()
- hlist_next_rcu()
- synchronize_rcu_bh()
- synchronize_rcu_bh_expedited()
- synchronize_sched()
- synchronize_sched_expedited()
- synchronize_srcu()
- synchronize_srcu_expedited()

This is RCU in DYNIX/ptx

rcu_read_lock()
rcu_read_unlock()



rcu_assign_pointer() [Sort of]



kfree_rcu()
call_rcu()

list_first_entry_rcu()
list_for_each_entry_rcu()
list_for_each_continue_rcu()
list_for_each_entry_continue_rcu()
hlist_first_rcu()
hlist_next_rcu()

synchronize_rcu_bh()
synchronize_rcu_bh_expedited()
synchronize_sched()
synchronize_sched_expedited()
synchronize_srcu()
synchronize_srcu_expedited()

This is RCU in Linux

- __rcu
- init_srcu_struct()
- cleanup_srcu_struct()
- RCU_INIT_POINTER()
- init_rcu_head_on_stack()
- destroy_rcu_head_on_stack()
- SLAB_DESTROY_BY_RCU
- rcu_read_lock()
- rcu_read_unlock()
- rcu_read_lock_bh()
- rcu_read_unlock_bh()
- rcu_read_lock_sched()
- rcu_read_lock_sched_notrace()
- rcu_read_unlock_sched()
- rcu_read_unlock_sched_notrace()
- srcu_read_lock()
- srcu_read_unlock()
- rcu_lockdep_assert()
- rcu_read_lock_held()
- rcu_read_lock_bh_held()
- rcu_read_lock_sched_held()
- srcu_read_lock_held()
- rcu_access_pointer()
- rcu_dereference()
- rcu_dereference_bh()
- rcu_dereference_bh_check()
- rcu_dereference_bh_protected()
- rcu_dereference_check()
- rcu_dereference_index_check()
- rcu_dereference_protected()
- rcu_dereference_raw()
- rcu_dereference_sched()
- rcu_dereference_sched_check()
- rcu_dereference_sched_protected()
- srcu_dereference()
- srcu_dereference_check()
- list_entry_rcu()
- list_next_rcu()
- list_first_entry_rcu()
- list_for_each_entry_rcu()
- list_for_each_continue_rcu()
- list_for_each_entry_continue_rcu()
- hlist_first_rcu()
- hlist_next_rcu()
- hlist_pprev_rcu()
- hlist_for_each_entry_rcu()
- hlist_for_each_entry_rcu_bh()
- hlist_for_each_entry_continue_rcu()
- hlist_for_each_entry_continue_rcu_bh()
- hlist_nulls_first_rcu()
- hlist_nulls_for_each_entry_rcu()
- hlist_bl_first_rcu()
- hlist_bl_for_each_entry_rcu()
- rcu_assign_pointer()
- list_add_rcu()
- list_add_tail_rcu()
- list_del_rcu()
- list_replace_rcu()
- hlist_del_rcu()
- hlist_del_init_rcu()
- hlist_replace_rcu()
- hlist_add_head_rcu()
- hlist_add_before_rcu()
- hlist_add_after_rcu()
- list_splice_init_rcu()
- hlist_nulls_del_init_rcu()
- hlist_nulls_del_rcu()
- hlist_nulls_add_head_rcu()
- hlist_bl_set_first_rcu()
- hlist_bl_del_init_rcu()
- hlist_bl_del_rcu()
- hlist_bl_add_head_rcu()
- kfree_rcu()
- call_rcu()
- call_rcu_bh()
- call_rcu_sched()
- rcu_barrier()
- rcu_barrier_bh()
- rcu_barrier_sched()
- synchronize_net()
- synchronize_rcu()
- synchronize_rcu_expedited()
- synchronize_rcu_bh()
- synchronize_rcu_bh_expedited()
- synchronize_sched()
- synchronize_sched_expedited()
- synchronize_srcu()
- synchronize_srcu_expedited()

For legible version, see: http://lwn.net/Articles/418853/

Without Contributions From Linux Community:

- Use of RCU would be error-prone:
  - Burying memory barriers into RCU primitives
  - Runtime RCU validation
  - Static RCU validation
  - RCU semantics
  - RCU proofs of correctness
- RCU would not be robust:
  - DoS attacks
- RCU would fail to handle important use cases:
  - Sleeping RCU readers
  - Early-boot RCU uses
  - Wait for callbacks: rcu_barrier()
  - RCU and type-safe memory
  - User-level RCU
  - Resizable RCU hash tables with wait-free readers
  - Workqueue-based RCU
- RCU would be slow and energy-inefficient
  - Expedited grace periods
  - Multi-tail callback lists
  - Energy conservation (dyntick-idle mode)

Lessons Learned From the RCU Experience

- Linux runs an incredible variety of workloads
  - Embedded, realtime, desktop, network, server, supercomputer...
  - Your solution might be perfect for embedded, but bad elsewhere
- Linux powers significant networking infrastructure
  - You can't hide behind a firewall: Linux *is* the firewall
- Linux runs realtime workloads: Realtime effects are pervasive
- Very large number of kernel developers (thousands)
  - If one person year of work saves 1% of everyone's time:
  - Linux: ~10,000 developers gives ~100 person-years per year payback
    - Investment pays off in less then four days
    - Even if only 500 full-time-developer equivalents, payoff in about 10 weeks
  - Proprietary: ~40 developers gives ~0.4 person-years per year payback
    - Investment takes more than two years to pays off
- Code developed in specialized environments will need serious modifications!!!

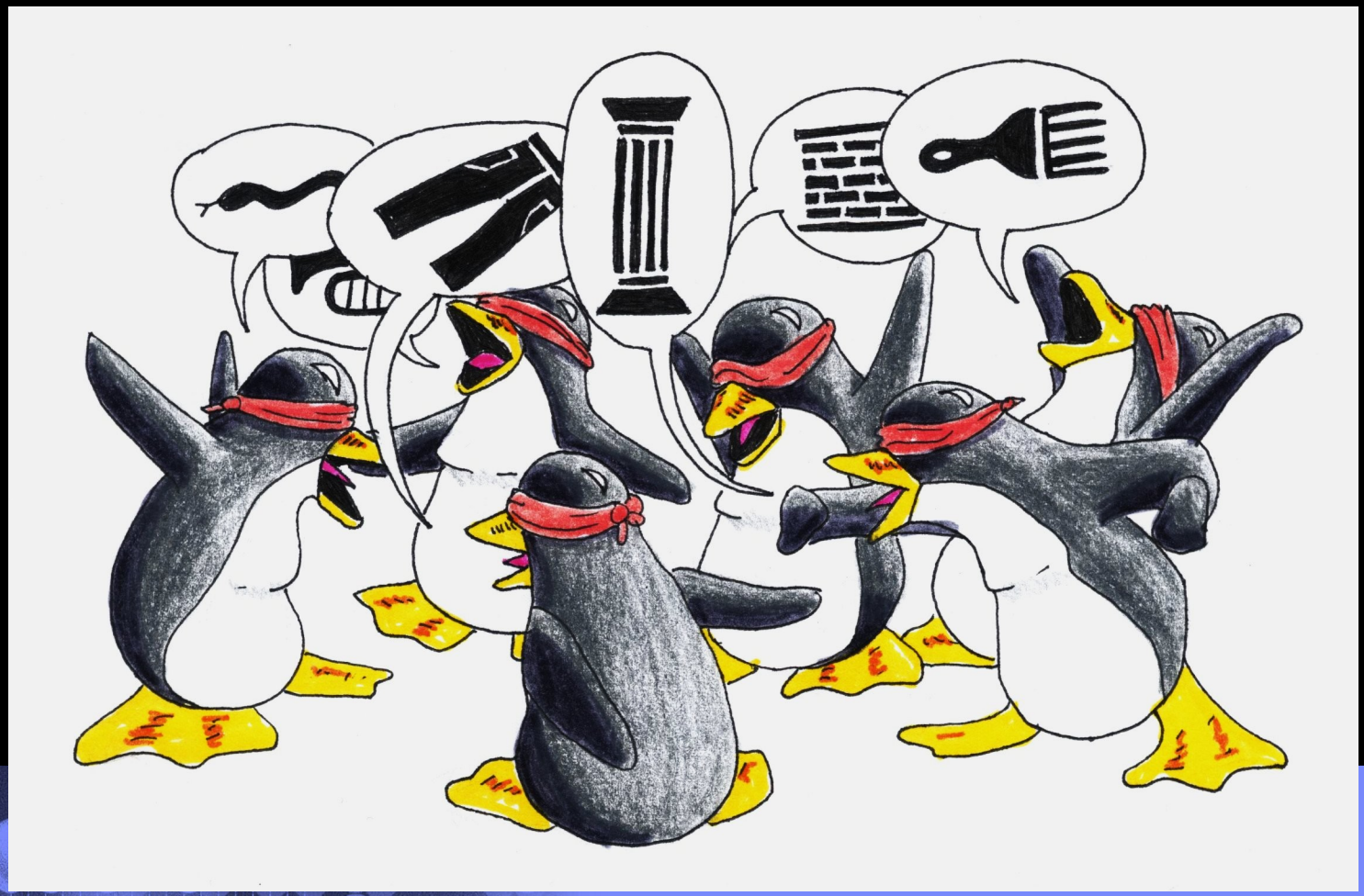
Lessons Learned From the RCU Experience

- The Linux kernel community probably does not know who you are or what you are capable of
  - You will need to prove yourself to them
  - Just as you would to any new community you were to join
  - Time spent learning about the community is time well spent
    - LWN articles, mailing-list archives, ...
- Respond quickly: hours or days, not weeks or months
- Maintain a professional bearing and attitude
  - If flamed, respond to the technical points, not to the emotion
    - The irritation is momentary, but an ill-considered reply is archived forever
  - It sometimes takes some effort to tease out technical points

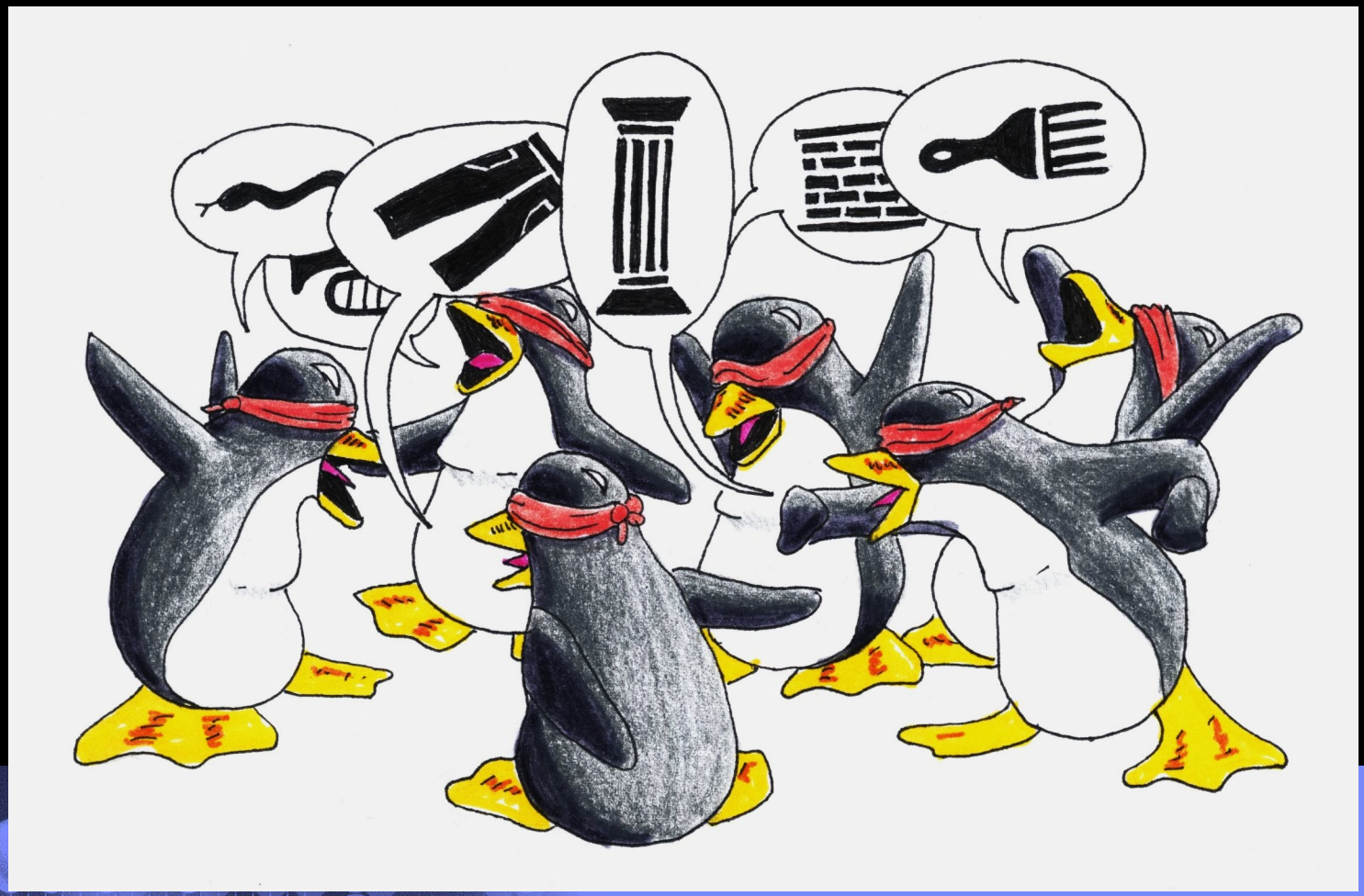
Other Examples of Good Solutions

- Dynticks
  - Better consolidation on mainframes
  - Better battery life on embedded devices
- Real-time Linux
  - Changes that improve real-time response...
  - ... often improve scalability on multicore systems
- Group scheduling
  - Helps servers manage their workloads
  - And also helps kernel hackers get good response times during large kernel builds

But Sometimes Things Get Stuck Here



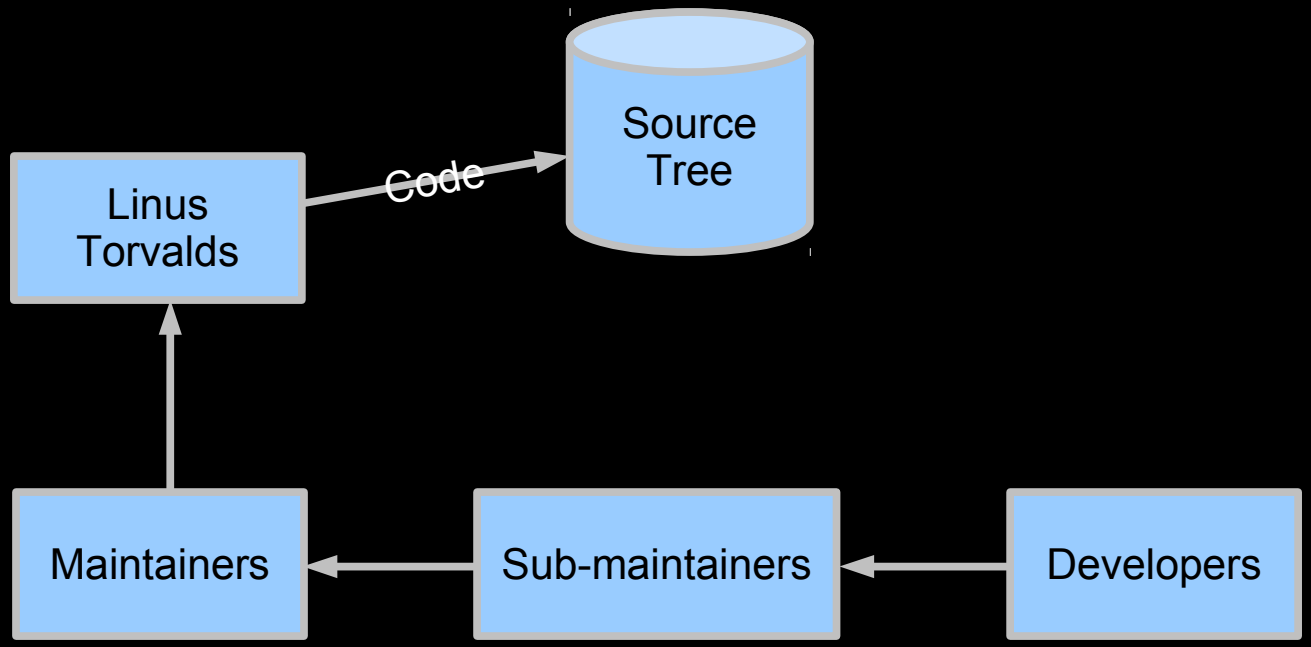
But Sometimes Things Get Stuck Here: Android!



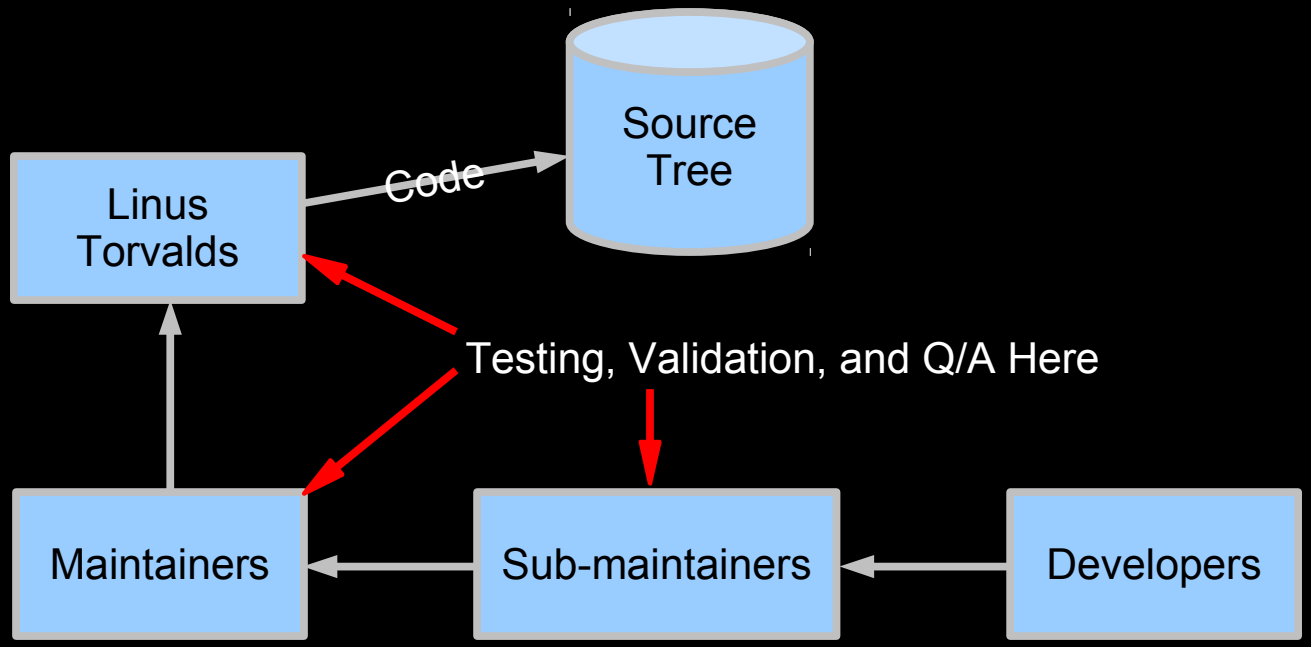
Confessions of a Recovering Proprietary Programmer

Maintainership Structure, or "Why Do Those Idiots Keep Rejecting My Stuff?"

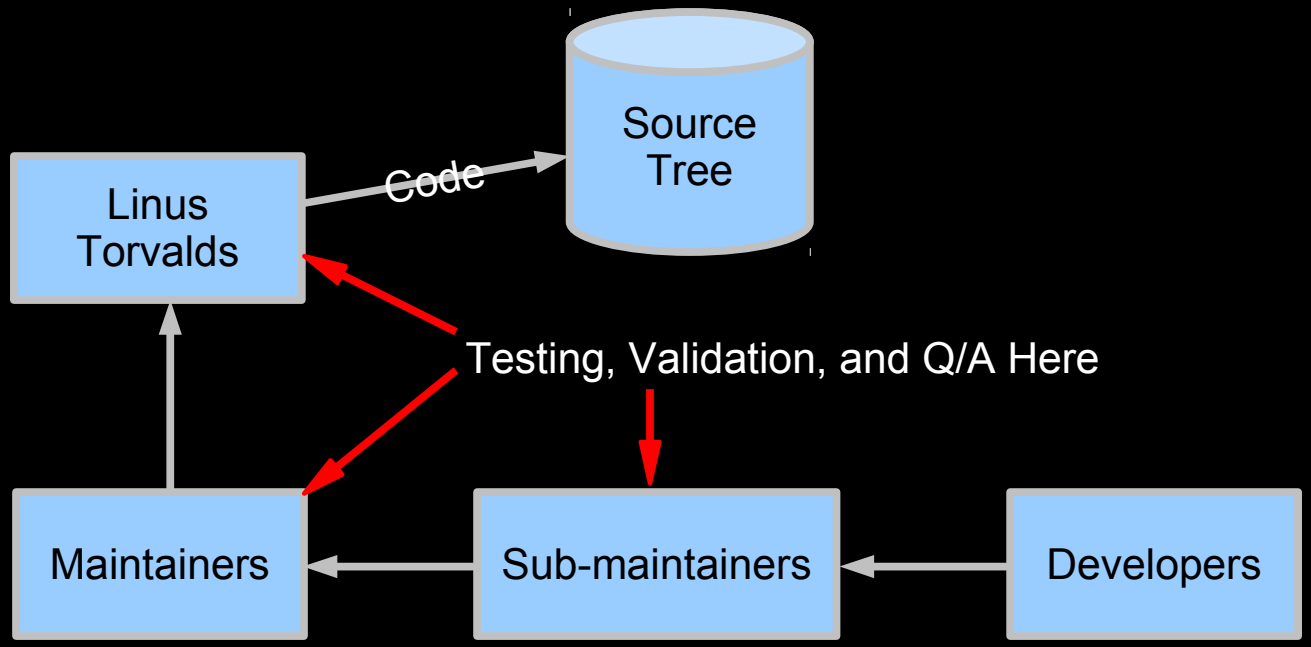
Linux Kernel Structure: People



Linux Kernel Structure: People

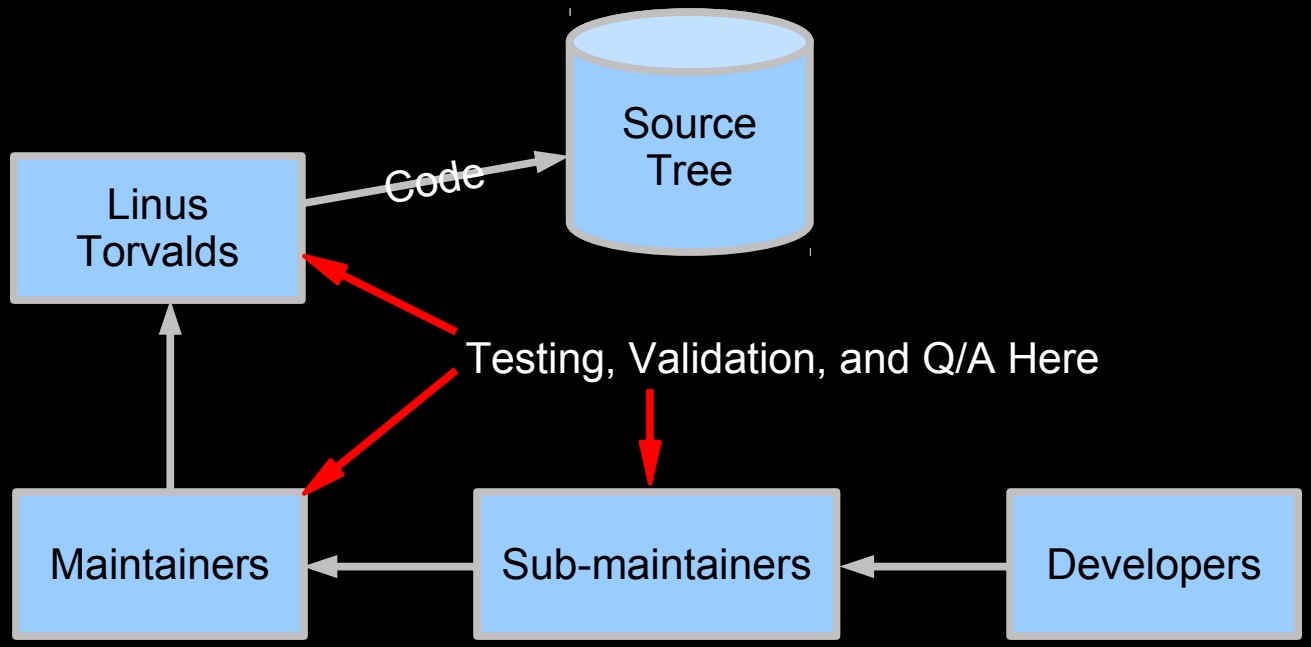


Linux Kernel Structure: People



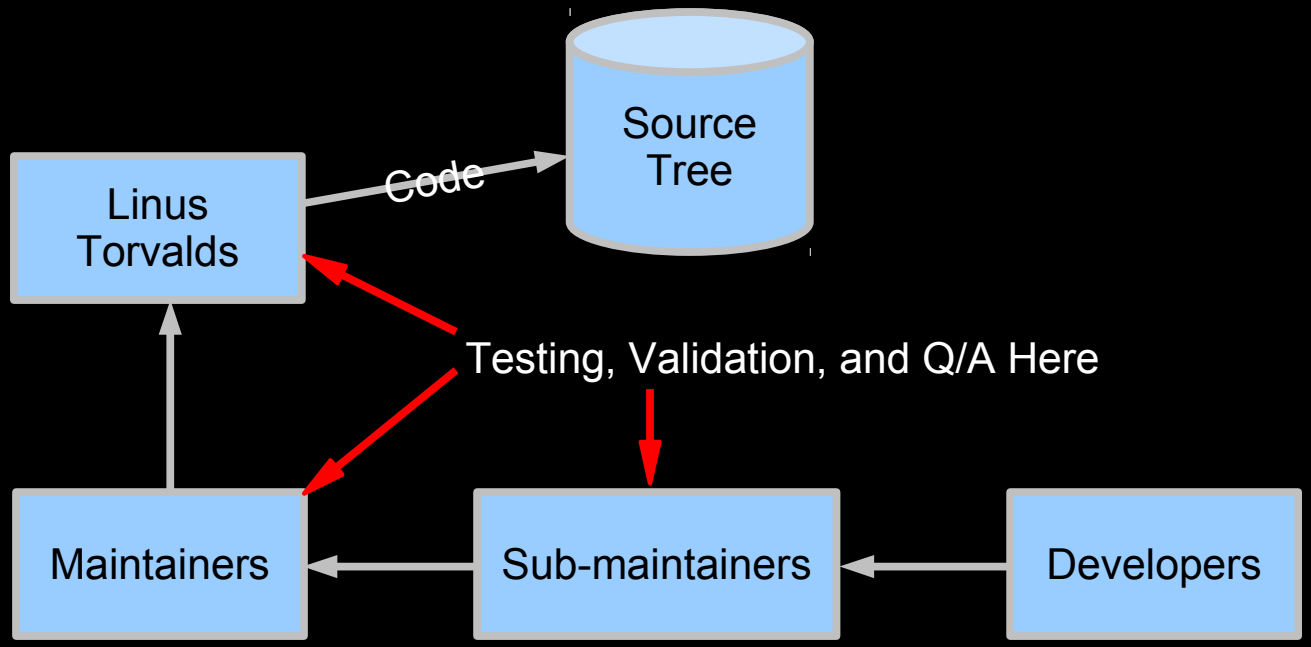
If I accept an RCU patch, then I am taking responsibility for it.

Linux Kernel Structure: People



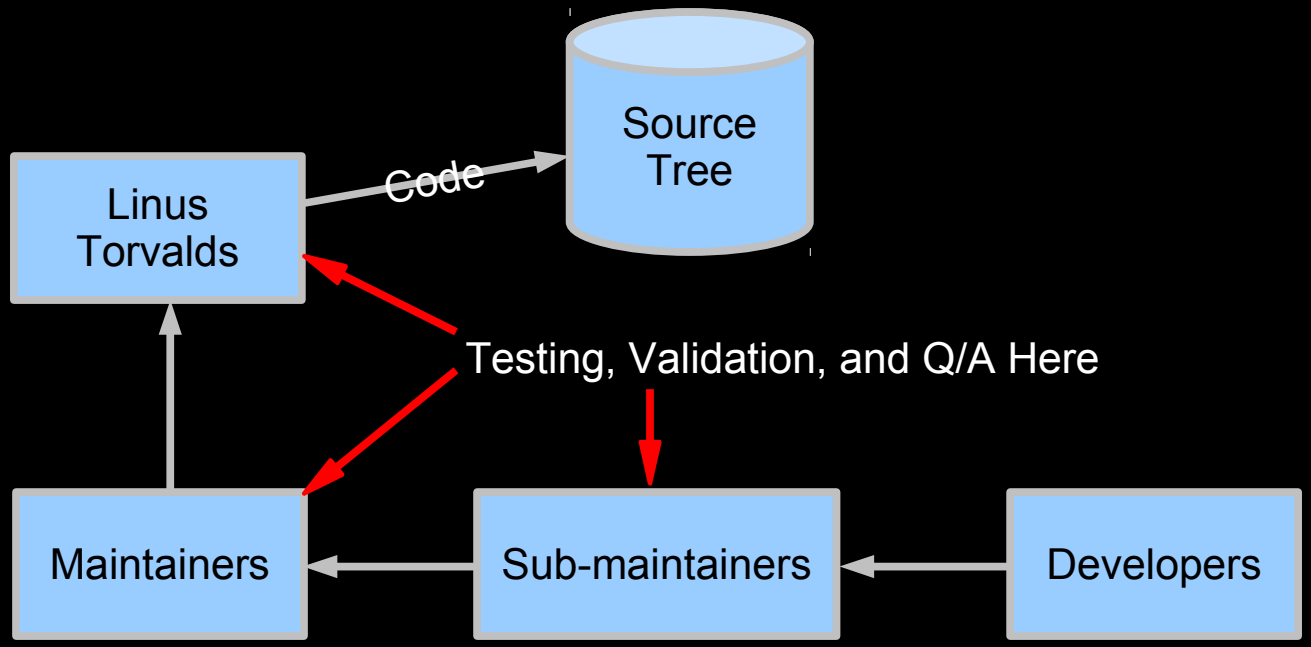
If I accept an RCU patch, then I am taking responsibility for it.
And the same thing applies to my upstream maintainer.

Linux Kernel Structure: People



If I accept an RCU patch, then I am taking responsibility for it.
And the same thing applies to my upstream maintainer.
So we reject patches with quality/maintainability problems.

Linux Kernel Structure: People



If we did otherwise, the Linux kernel would be a complete mess.

Confessions of a Recovering Proprietary Programmer

Coding Style, or "When in Rome..."

Confessions: Coding Style

- "do { } while (0)" for statement-like cpp macros
  - But static inline functions instead whenever possible
  - Exceptions: polymorphic, iterators, declarators
- 80-column line limitation
- 8-space tabs
- No trailing space on lines
- Memory barriers must always be commented
- "return foo;", not "return (foo);"
- Omit unnecessary parentheses
- No braces for one-line "then" or "else" clauses

- No "#if" or "#ifdef" in .c files

Confessions: Coding Style: #ifdef

- I had been using #ifdef wherever I felt like it for two decades
  - Why change?
- Started a new project
  - Smallish, so simply wrote it both ways
  - Quickly abandoned the #ifdef-in-.c version
  - Why?
- Sometimes the Romans are right!

Confessions: Coding Style

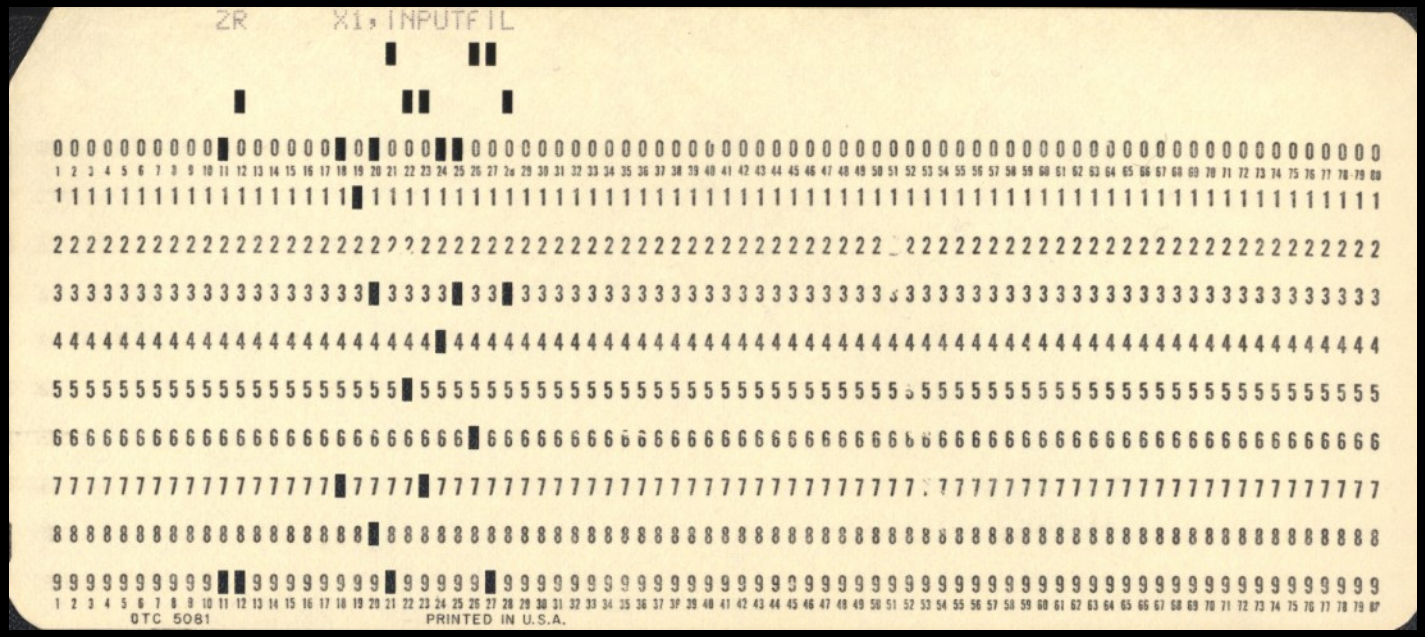
- "return foo;", not "return (foo);"
  - Fewer characters, more likely to fit in 80-characters
- Omit unnecessary parentheses
  - Fewer characters, more likely to fit in 80-characters
  - Good exercise for one's memory, I guess...
- No braces for one-line "then" or "else" clauses
  - Fewer lines, more likely to fit on single screen
  - But it does get me in trouble reasonably often

- Key point: Linux kernel is read far more often than it is written and/or debugged
  - The needs of the many readers outweigh those of the few(er) writers and debuggers

Confessions of a Recovering Proprietary Programmer

Source-Code Management

Confessions: Source-Code Management



Confessions: Source-Code Management

Confessions: Why so Primitive???

Confessions: Source-Code Management

- Revision Control System (RCS)
  - Created by Walter Tichy in 1980s
  - I used it for about 20 years
- Strong Points:
  - Less need to retype old versions from printouts
  - Trivial to track changes on file-by-file basis
- Shortcomings:
  - Hard to get consistent past view of set of files
  - Hard to use collaboratively
    - Lots of different script wrappers for RCS to make this work
  - Merges are extremely painful and easy to get wrong

Confessions: Source-Code Management

- bitkeeper
  - Created by Larry McVoy in late 1990s
  - I used it very lightly for a few years
- Strong points:
  - Consistent global view of source tree
  - Better support for merges
  - Easier to use collaboratively
- Shortcomings:
  - Proprietary software
  - Massive political hassles

Confessions: Source-Code Management

- git
  - Created by Linux community in mid-00s
- Strong points:
  - Consistent global view of source tree
  - Way better support for merges: difference in kind
  - Easier to use collaboratively
  - Integrates patch handling and maintainer roles
  - Automated branch rebasing
- Shortcomings:
  - Learning curve!!!
  - Don't try this on 1990s storage hardware...

- From “I hate git” to reasonably happy git user

Summary

- Paul knows something about open source
  - But don't take my word for it, ask Google!!!
  - And a lot of other Linaro folks know even more
- Open discussion often produces better results than isolated development
- Open-source development has surprising implications on coding style
- Open-source software has resulted in great advances in source-code management

Summary: Additional Material

- Greg Kroah-Hartman's “Write and Submit your first Linux kernel Patch” (2010)
  - http://archive.fosdem.org/2010/schedule/events/linuxkernelpatch
- Jonathan Corbet's “How to Participate in the Linux Community” (2008)
  - http://ldn.linuxfoundation.org/how-participate-linux-community
- Randy Dunlap's “Linux Kernel Development: Getting Started” (2005)
  - http://www.xenotime.net/linux/mentor/linux-mentoring.pdf
- Greg Kroah-Hartman's “HOWTO do Linux kernel development – take 2” (2005)
  - http://lwn.net/Articles/160191/
- Linux kernel documentation
  - Documentation/SubmittingPatches
  - Documentation/SubmitChecklist
  - Documentation/SubmittingPatches

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.
- This material is based upon work supported by the National Science Foundation under Grant No. CNS-0719851.

Questions?

Backup