2008 Linux Developer Symposium - China

Introducing Technology Into Linux

*Or: Introducing your technology Into Linux will require introducing a **lot** of Linux into your technology!!!*

Paul E. McKenney, Distinguished Engineer
IBM Linux Technology Center

January 28, 2008

Overview

- Where to find out more about RCU
- RCU usage within the Linux kernel
- How Linux changed RCU
- Lessons learned

Where to Find out More About RCU

Where to Find out More About RCU

- **RCU can be thought of as a replacement for reader-writer locking with extreme read-side performance, scalability, and determinism**
- **For more information on RCU, please see:**
  - Linux Weekly News “What is RCU Really?” Series
    - What is RCU, Fundamentally?
      - http://lwn.net/Articles/262464/
    - What is RCU? Part 2: Usage
      - http://lwn.net/Articles/263130/
    - RCU part 3: the RCU API (includes annotated bibliography)
      - http://lwn.net/Articles/264090/
  - Paul McKenney's RCU page
    - http://www.rdrop.com/users/paulmck/RCU/

Why RCU's Performance Matters

4-CPU 1.8GHz AMD Opteron 844 system

Operation	Cost (ns)	Ratio
Clock period	0.6	
Best-case CAS	37.9	63.2
Best-case lock	65.6	109.3
Single cache miss	139.5	232.5
CAS cache miss	306.0	510.0

RCU Readers

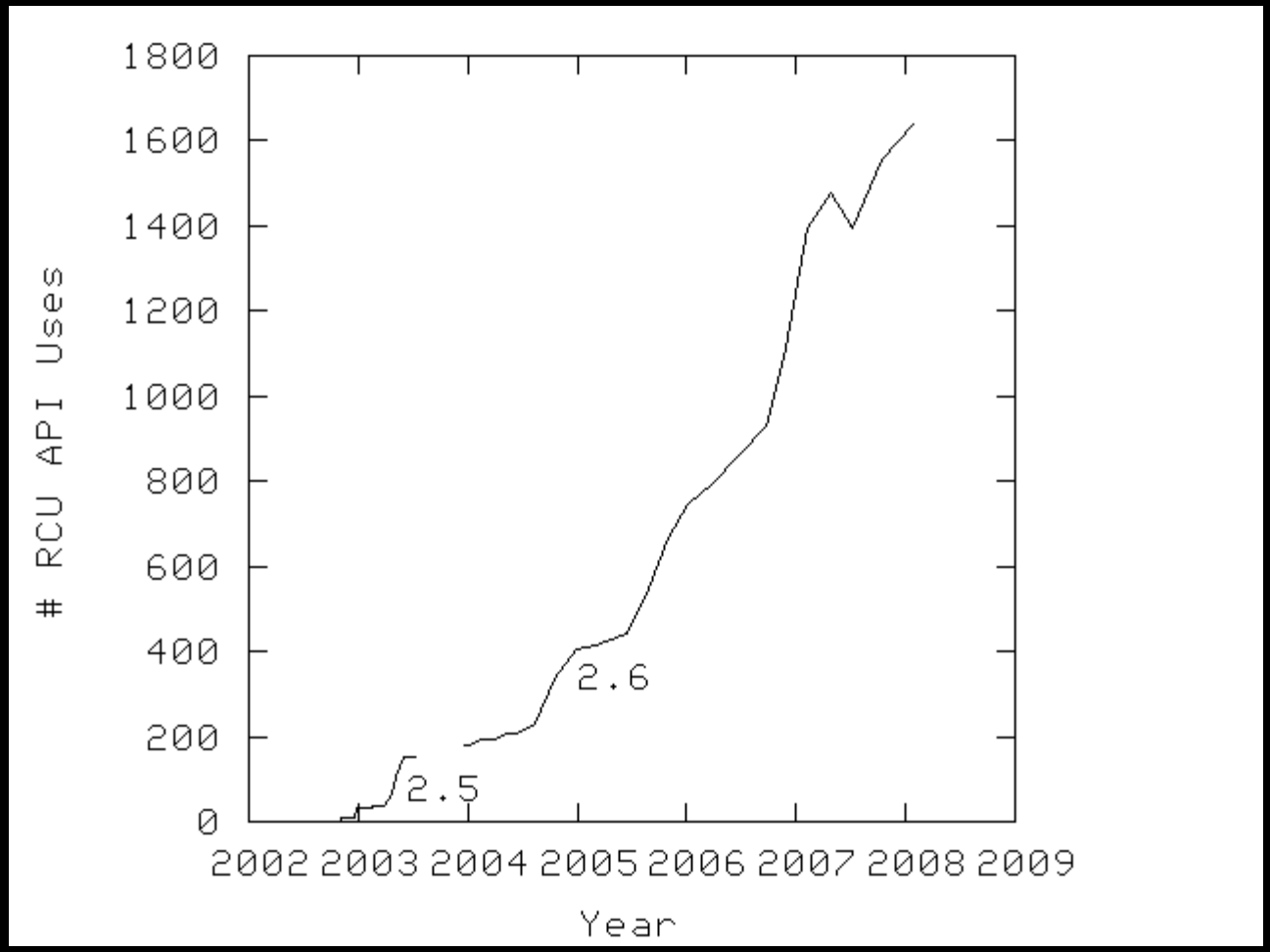
Heavily optimized reader-writer lock might get here for readers (but too bad about those poor writers...)

Typical synchronization mechanisms do this a lot

RCU readers use low-cost instructions, other approaches use expensive instructions.

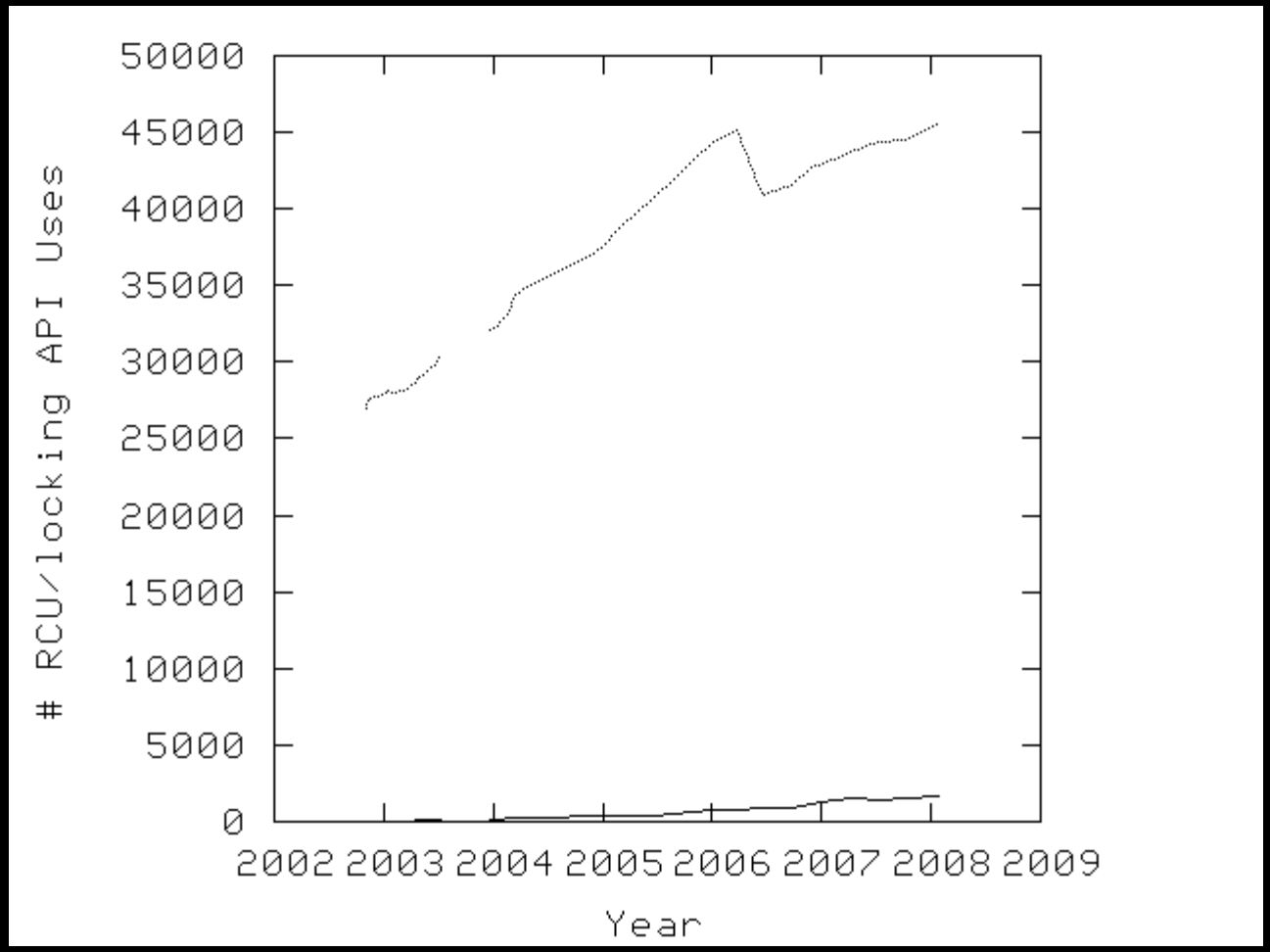
RCU Usage Within the Linux Kernel

RCU Usage Within the Linux Kernel (2.6.24)

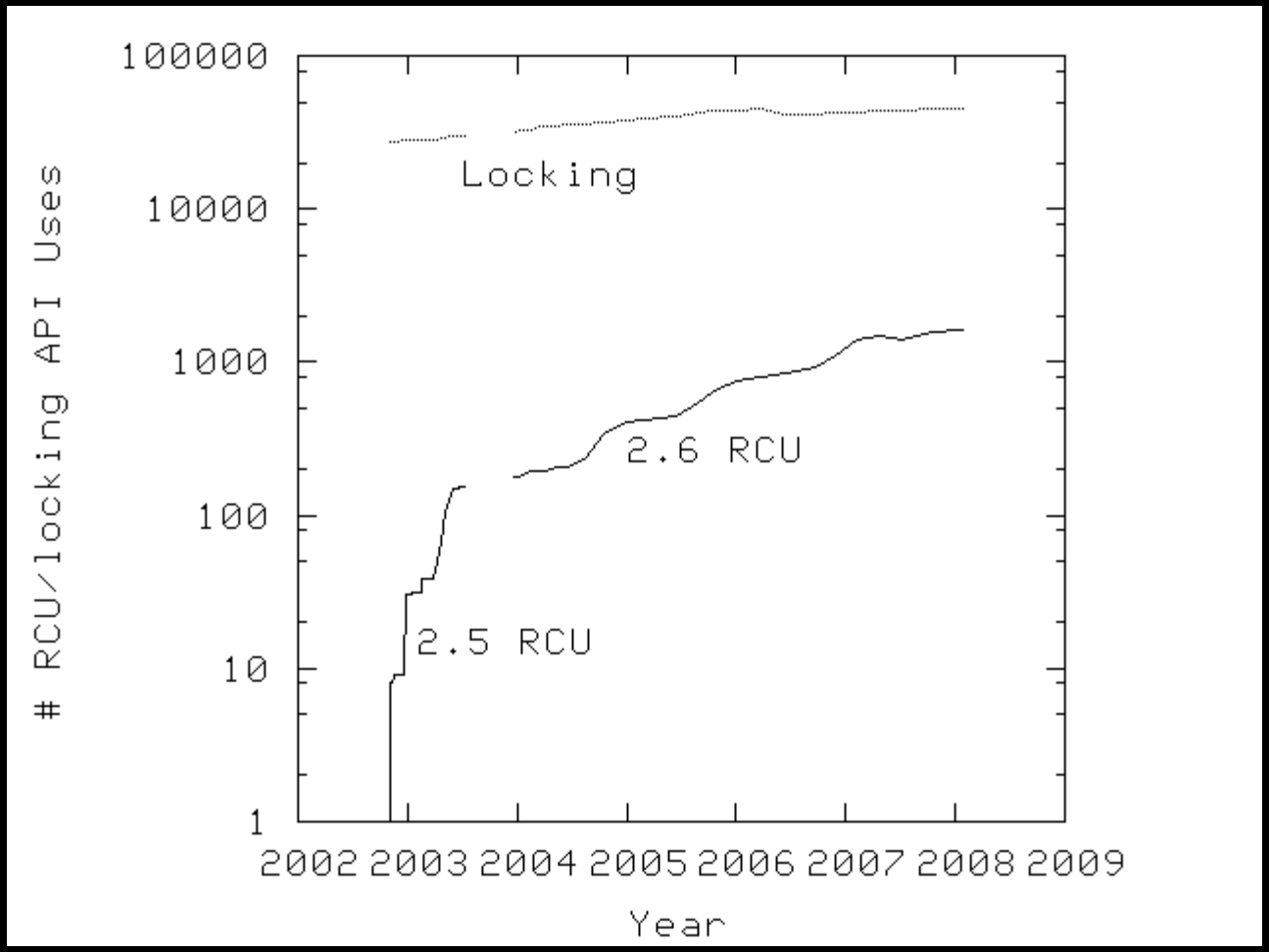


In case there was any doubt, the Linux community ***can*** handle RCU.

RCU Usage Within the Linux Kernel vs. Locking



RCU Usage Within the Linux Kernel vs. Locking



RCU Usage Within the Linux Kernel vs. Locking

RCU is a reasonably successful niche technology within the Linux kernel

How did RCU get there?

It got there by being changed dramatically by Linux!!!

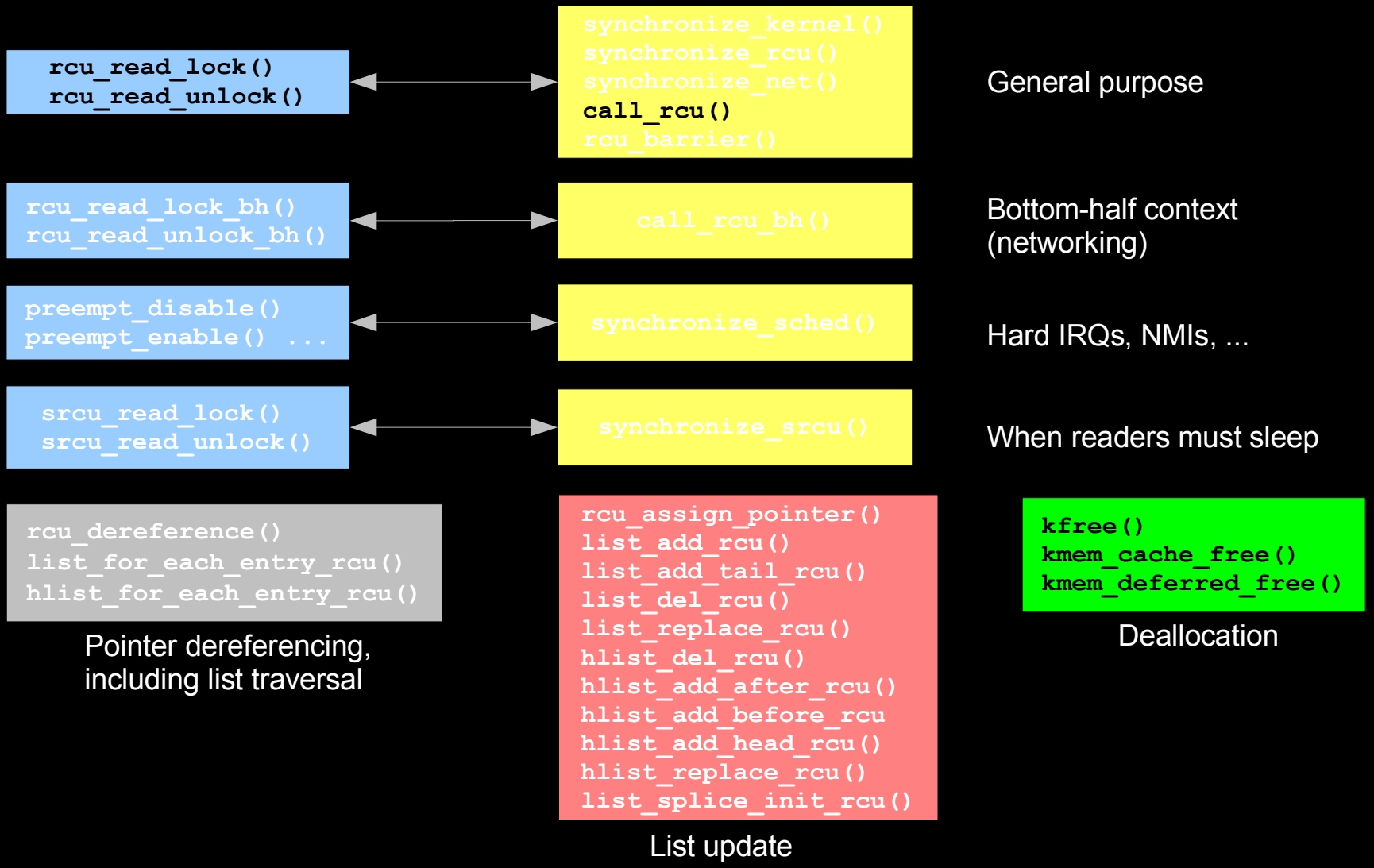
How Linux Changed RCU

Pre-Linux Experience With RCU

- **Enterprise systems: big parallel database servers**
  - Tens of CPUs, tens of GB of memory (big by mid-90s standards)
- **Protected networking environment**
  - firewalls, clients, restricted usage modes
- **No realtime response required**
  - Unless you count TPC/A requirement that 90% of transactions complete in two sections
- **Very small number of kernel developers (a few tens)**

- **Significant changes required to adapt to Linux**
  - As we will see as we fly through the following slides
  - Additional details and references provided in case you wish to review the slides later
  - Intent is to give you an overall flavor of the magnitude of change

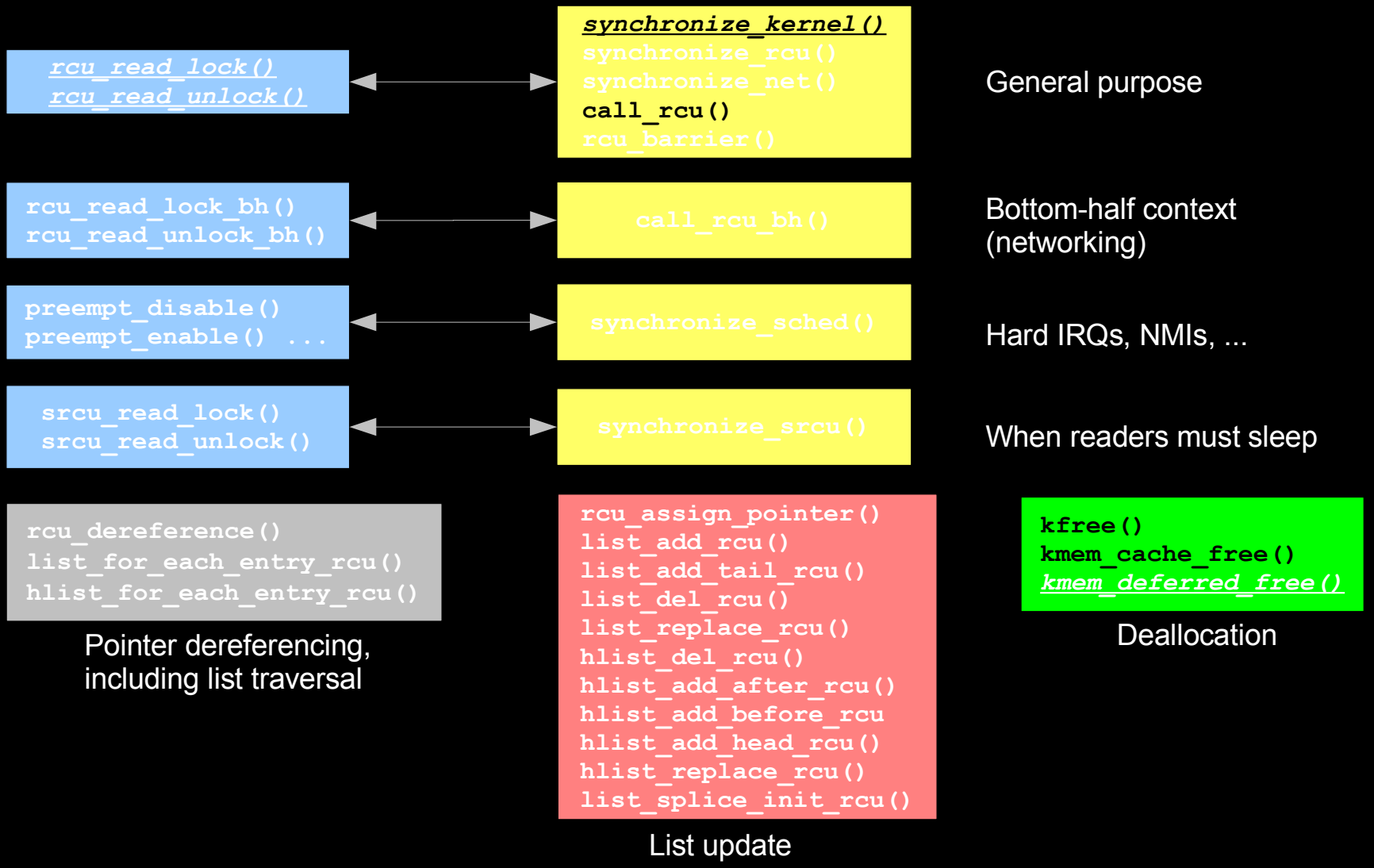
Linux RCU API Based on Pre-Linux Experience



Linux Kernel Developers Value Simplicity

- **Painful though it may be at times, this is a good thing**
- **In many cases, complexity is a symptom of lack of understanding**
  - If you know only one way to do something, the odds are against it being the best way!
  - Kudos to Andi Kleen, Rusty Russell, Andrea Arcangeli, and many others for generating alternative implementations
  - And to Dipankar Sarma for doing the implementation and incorporating the plethora of excellent ideas from the Linux community

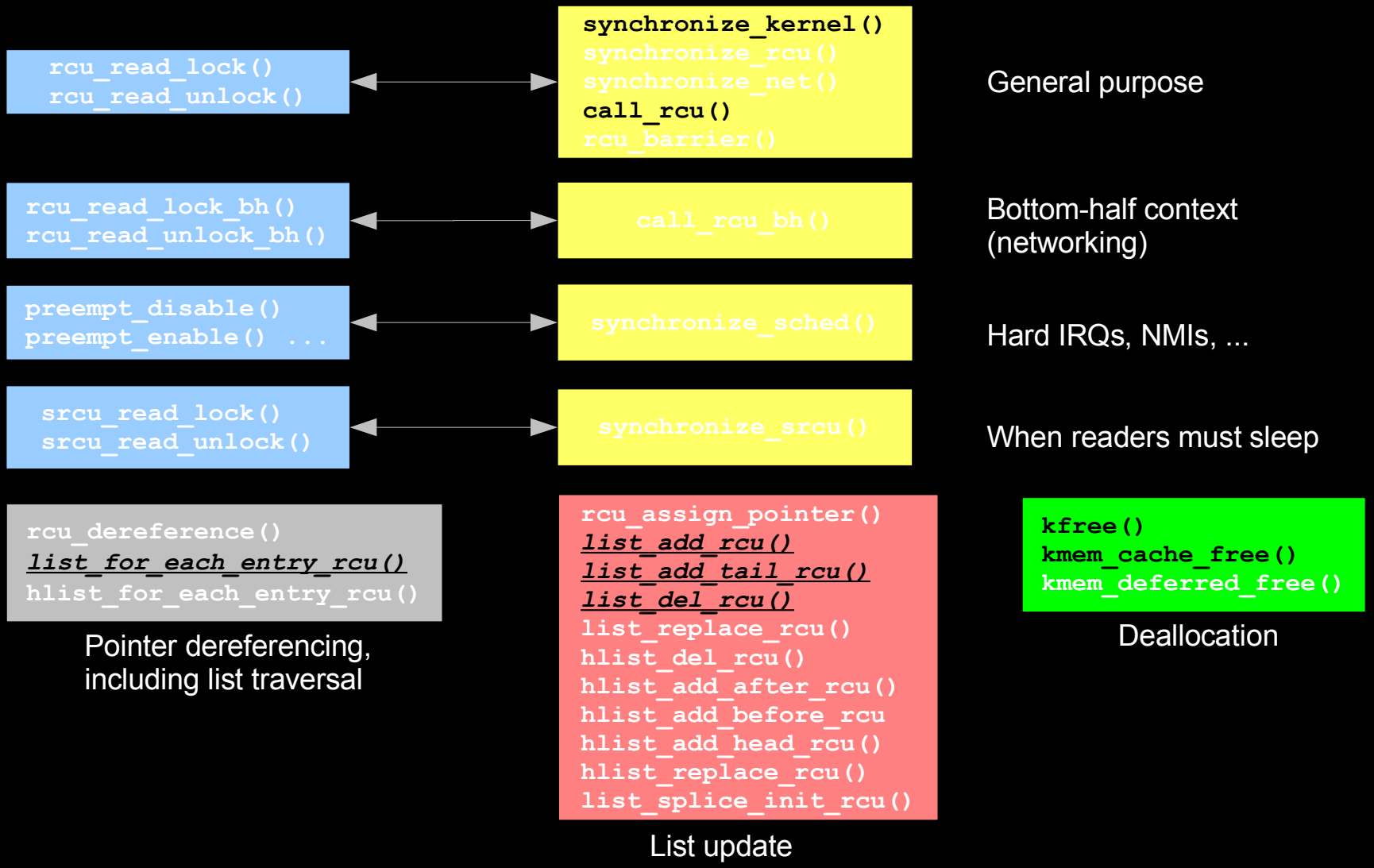
Simplified Linux RCU API



Memory Barriers are Unloved

- **With good reason**
- **They are hard to understand and easy to get wrong**
- **Many maintainers had a blanket policy:**
  - "Reject any patch containing a memory barrier"
  - This has since been softened to require meaningful comments on memory barriers
- **It is far better to bury any needed memory barriers into a well-designed API**
  - Kudos to Manfred Spraul for suggesting the RCU list API!

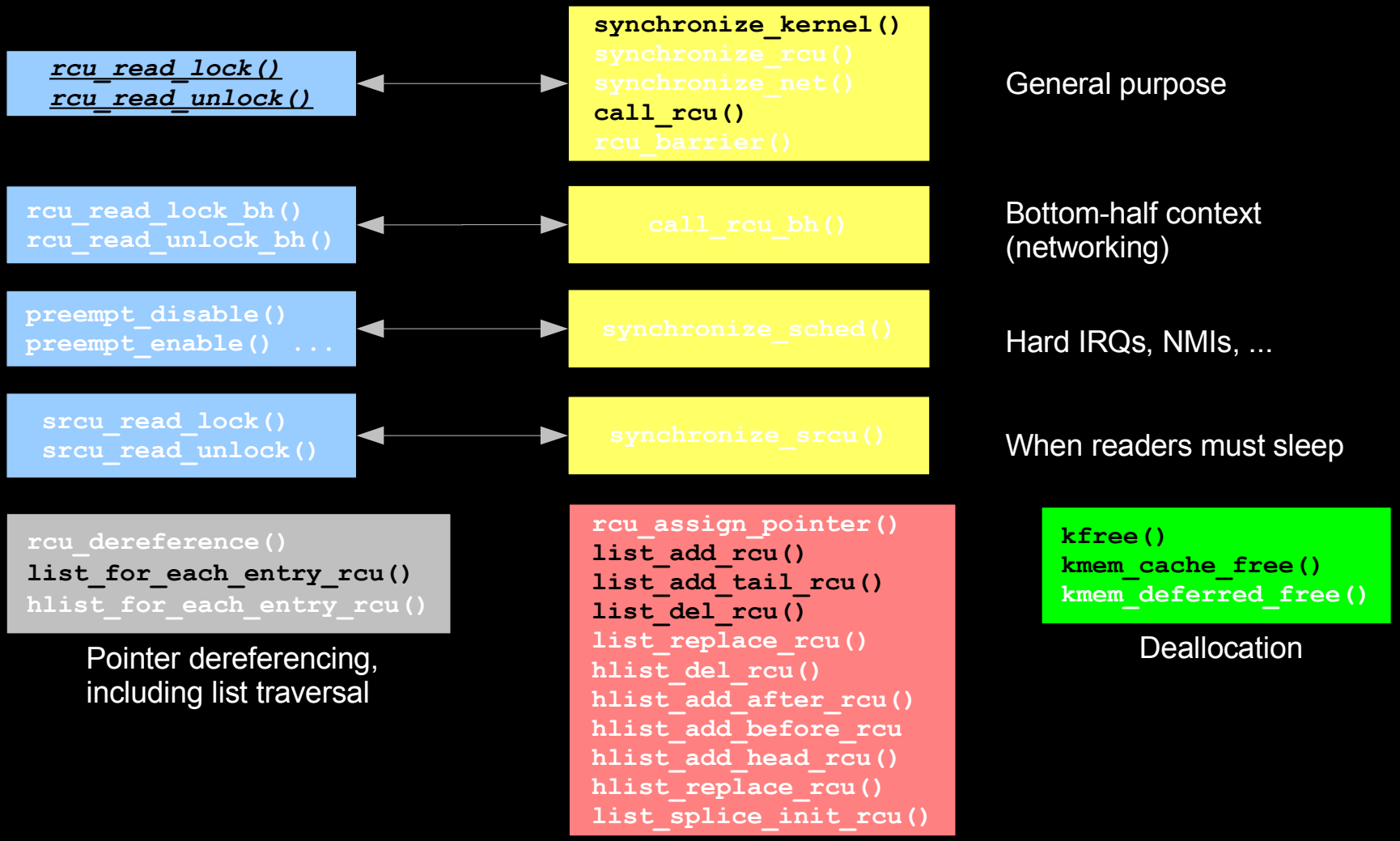
Linux Kernel Memory Barriers Unloved, Take 1



Linux Kernel Does Real-Time Systems, Take 1

- **The CONFIG_PREEMPT function enters the kernel**
- **The kernel becomes preemptable, invalidating key RCU assumption**
- **Easy fix requires bringing rcu_read_lock() and rcu_read_unlock() back into the Linux kernel**
  - Where they have been invaluable documentation aids

Linux Kernel Does Real-Time Systems, Take 1

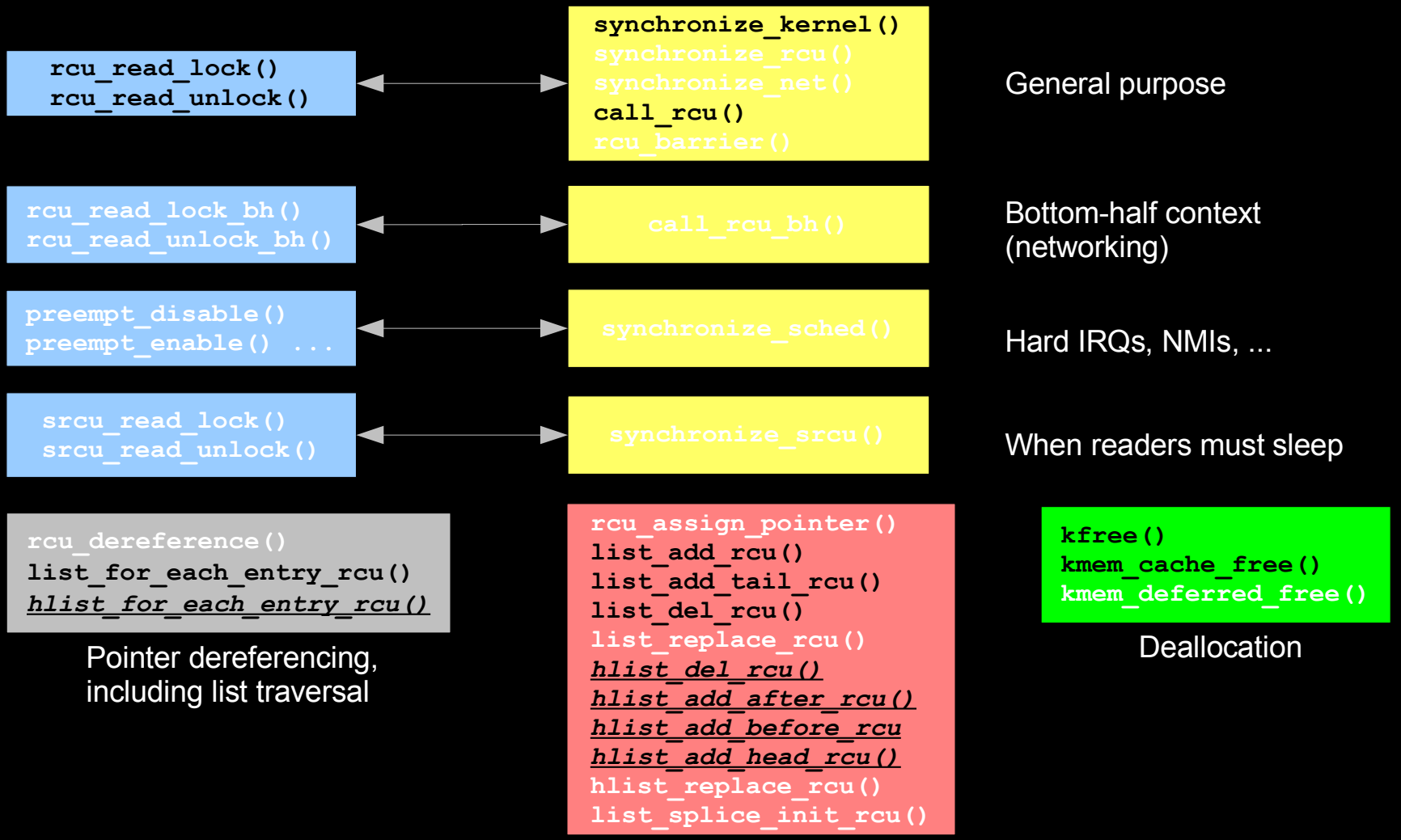


2.5.43

Linux Kernel Runs on Small-Memory Systems

- **Linux does circular doubly linked lists**
  - Consumes two pointers per hash bucket
- **Problematic given large hash tables on small-memory machines**
- **Solution: hlist, a linear doubly-linked list**
  - Consumes only one pointer per hash bucket
  - But adds another group of RCU list APIs
  - Implemented by Andi Kleen

Linux Kernel Runs on Small-Memory Systems

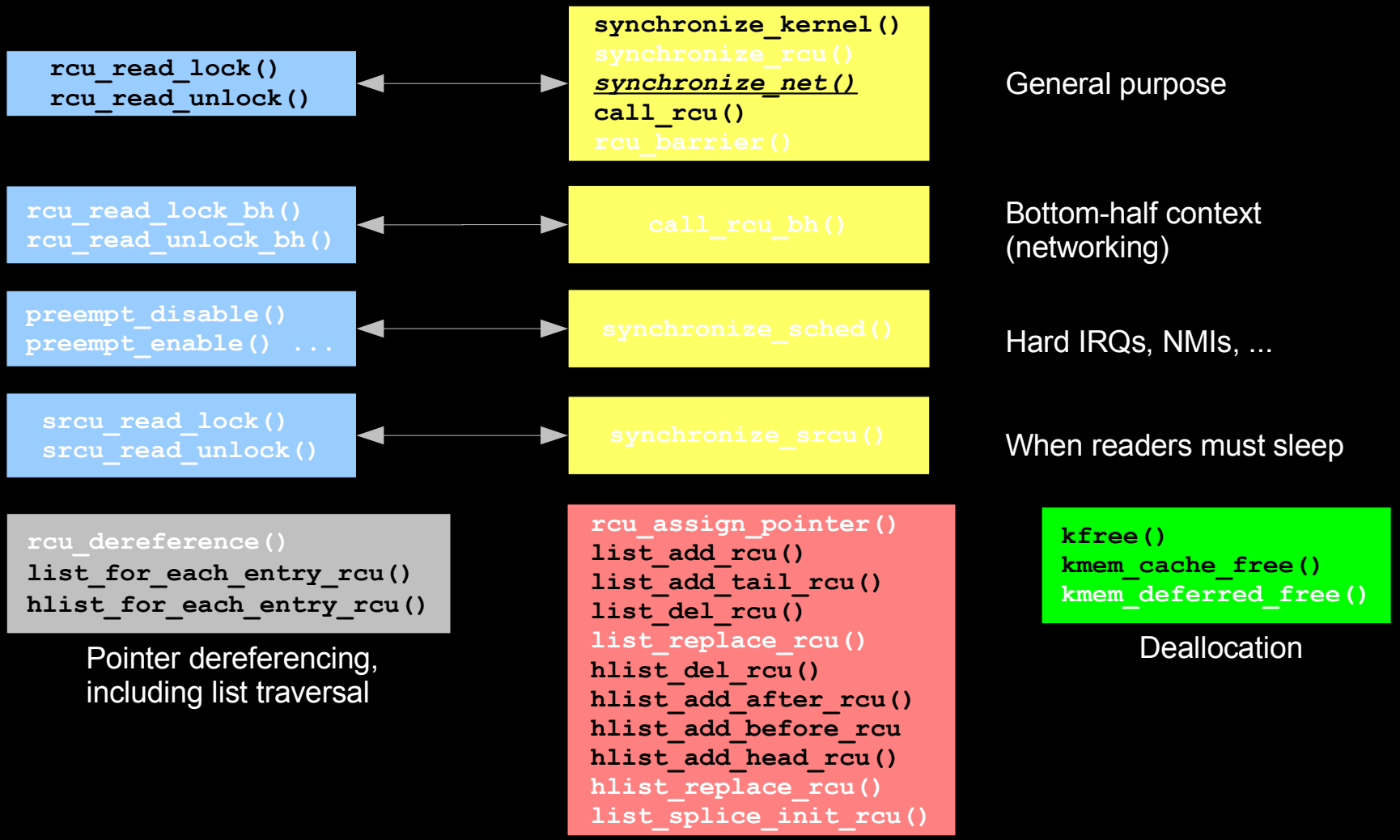


2.5.64

Linux Kernel Runs Heavy Networking Workloads

- **Steve Hemminger converts networking code from brlock to RCU, introducing synchronize_net() to ease the transition**
  - And synchronize_net() continues to be a reasonably useful documentation aid

Linux Kernel Runs Heavy Networking Workloads

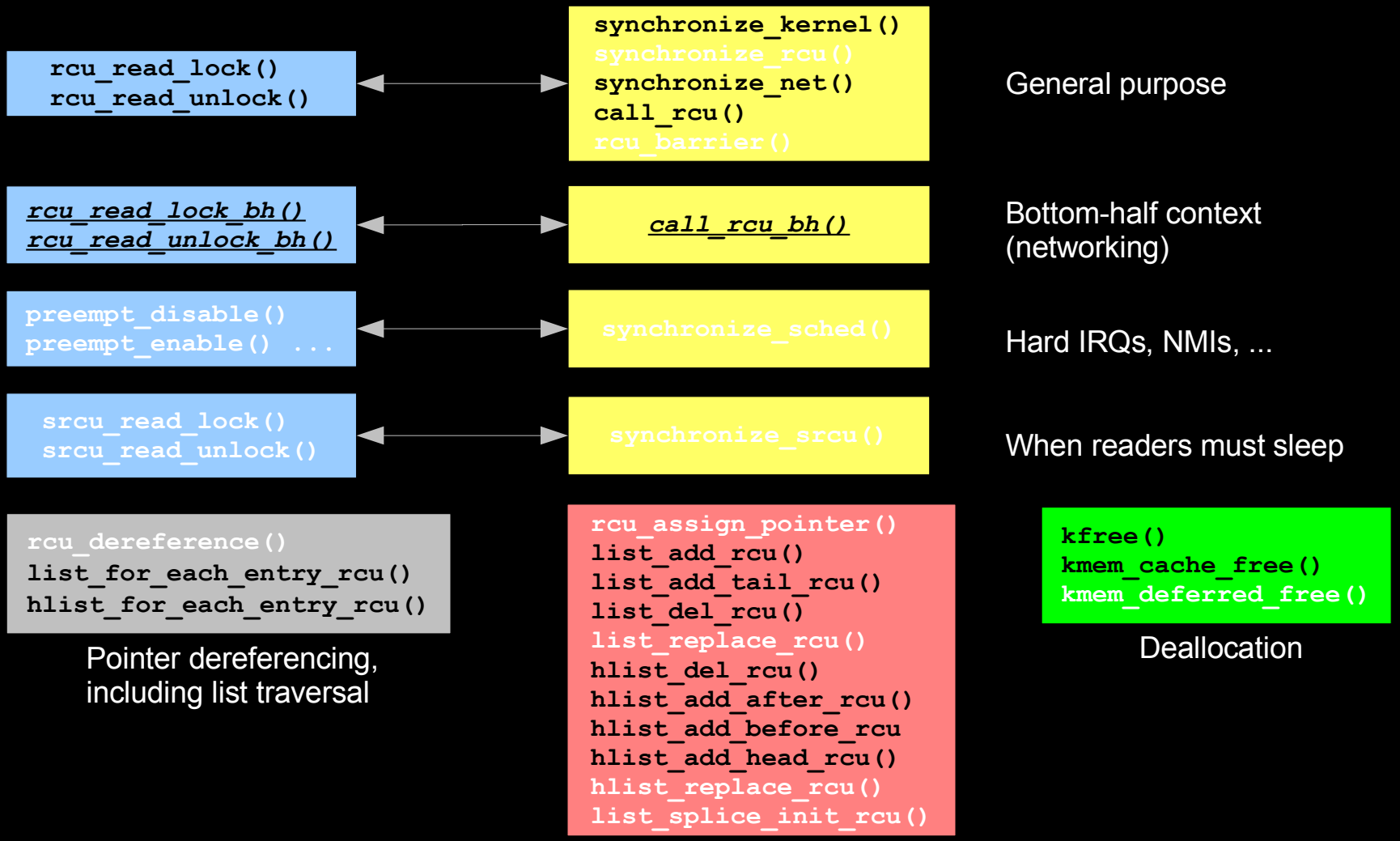


2.5.69

Linux Kernel Must Survive Networking DoS Attacks

- **Extremely heavy networking loads from denial-of-service attacks prevent RCU from doing its work**
  - Indefinitely postpones grace periods
- **New _bh variant of RCU avoids this problem**
  - Implemented by Dipankar Sarma with Robert Olsson

Linux Kernel Must Survive Networking DoS Attacks

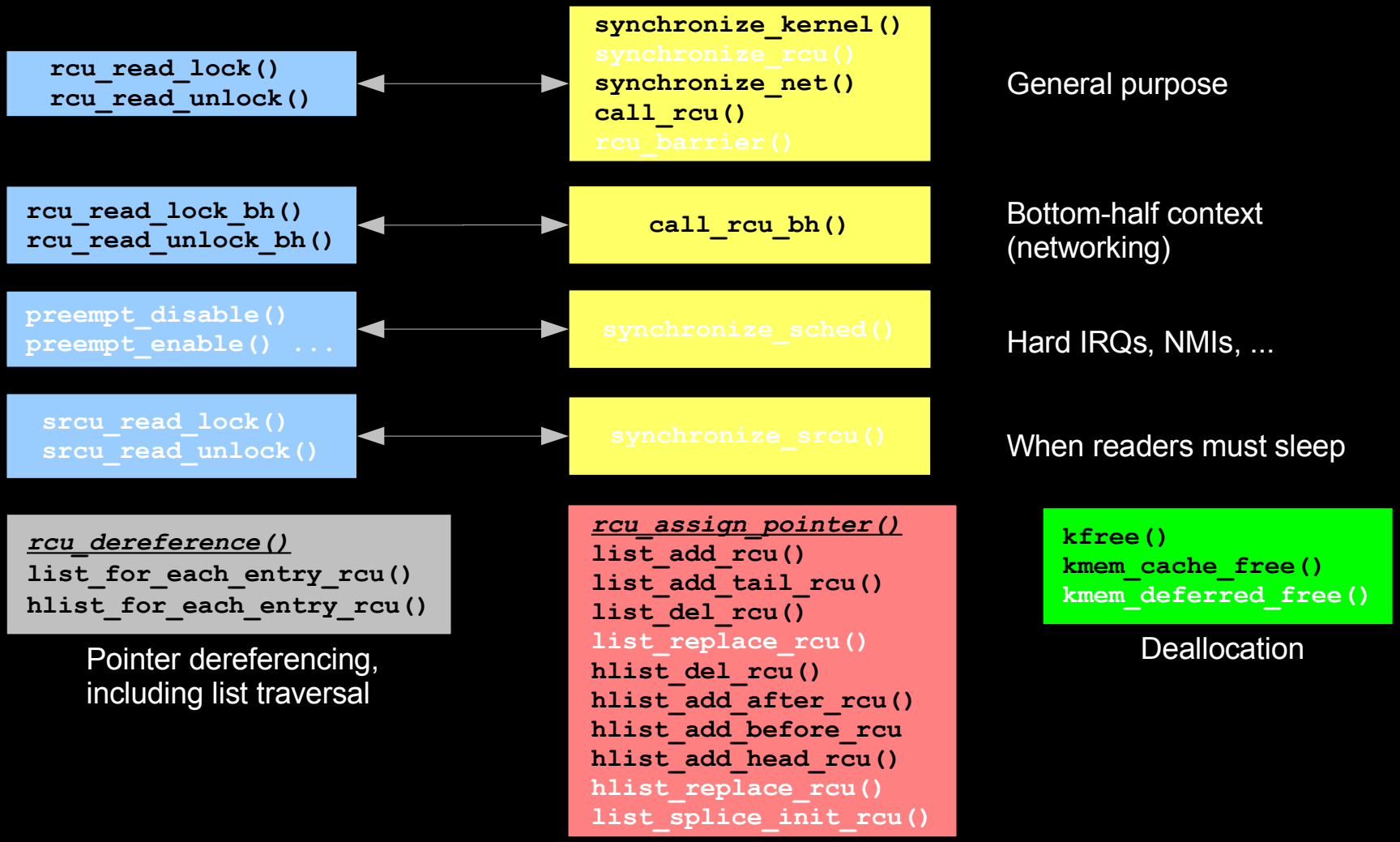


2.6.9

Linux Kernel Memory Barriers Unloved, Take 2

- **Burying memory barriers in list primitives does not help when applying RCU to non-list data structures**
- **People start applying RCU to trees and the like**
- **Therefore, created primitives to handle this case**

Linux Kernel Memory Barriers Unloved, Take 2

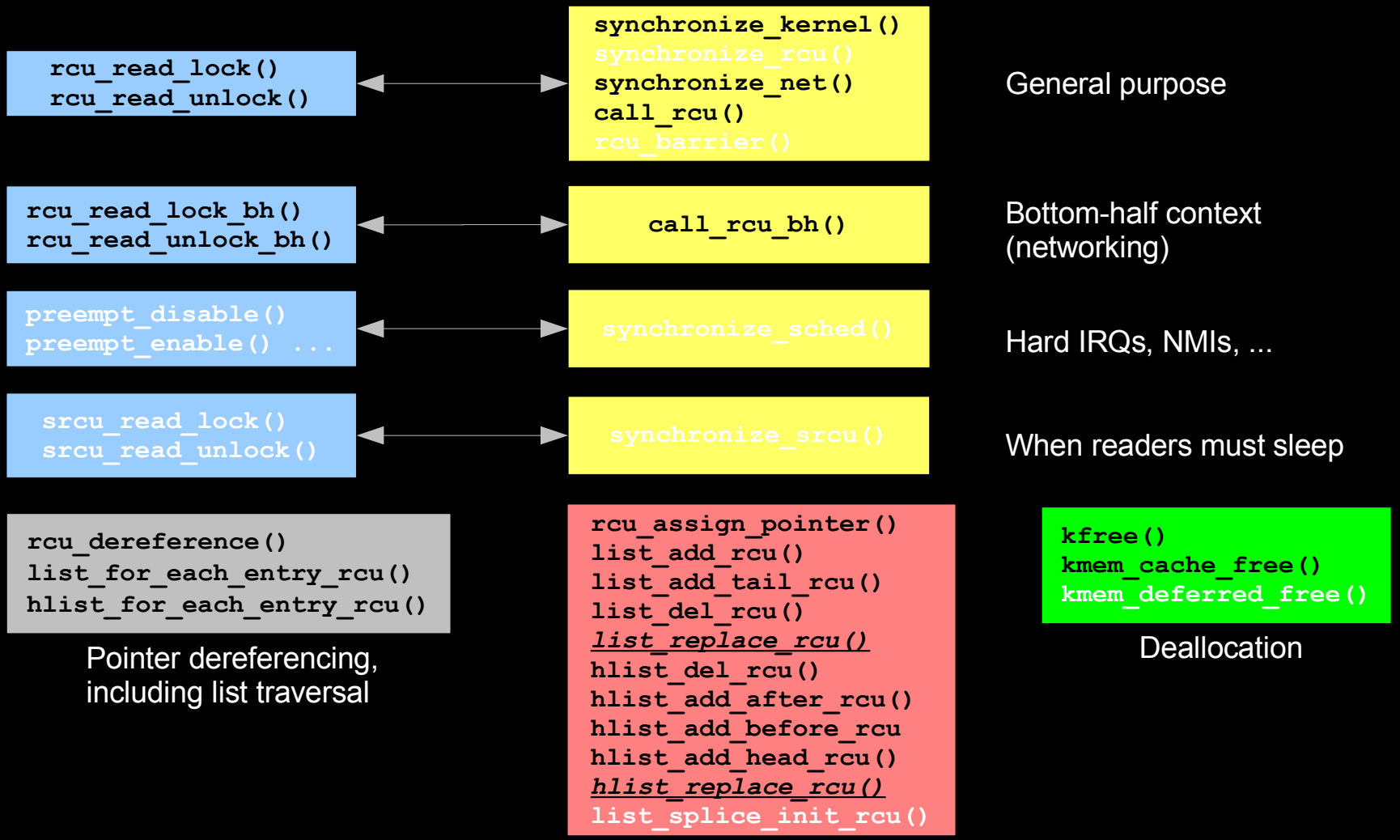


2.6.10

Linux's RCU Finally Implements Its Name

- **"RCU" stands for "read-copy update"**
  - Readers access the data structure with copy-based updates
- **As Murphy would have it, this turned out to be an unusual use case**
- **But the Linux kernel eventually needed it**
  - Kaigai Kohei implements it for SELinux AVC work

Linux's RCU Finally Implements Its Name

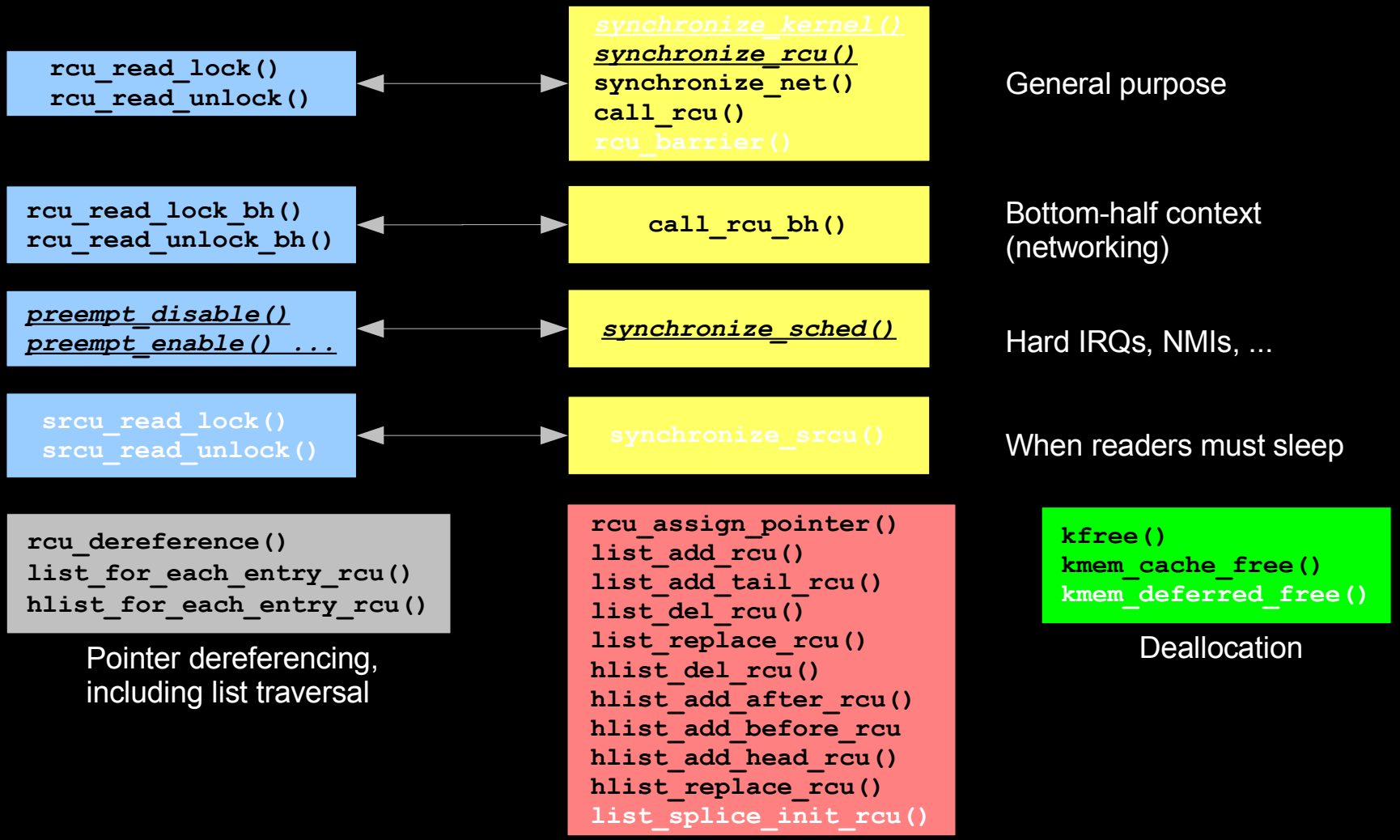


2.6.11

Linux Kernel Does Real-Time Systems, Take 2

- **People use RCU for its side effects**
  - For example, waiting for interrupts and NMIs handlers to complete
- **This makes it hard to implement an aggressive realtime implementation of RCU**
  - So we create an alternative API specifically for waiting for interrupt handlers and NMIs
  - See http://lwn.net/Articles/134484/ for details

Linux Kernel Does Real-Time Systems, Take 2

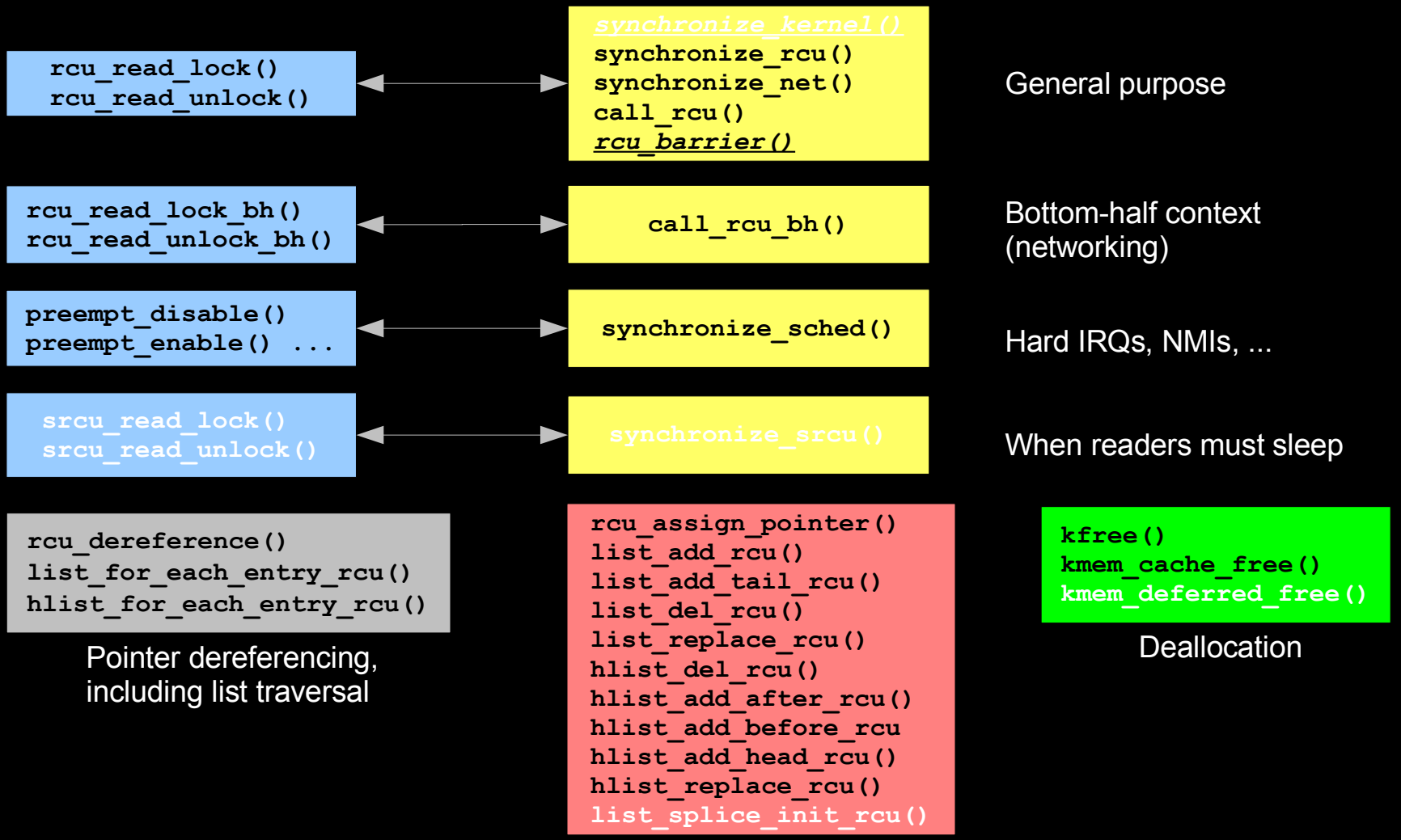


2.6.12

Linux Kernel Uses RCU in Unloadable Modules

- **A given module's RCU callbacks can execute after a module is unloaded**
  - So that the affected callbacks cannot find their object code
  - See http://lwn.net/Articles/217484/ for details
- **Added Dipankar Sarma's rcu_barrier() primitive to allow a module to wait for all of its RCU callbacks to complete**

Linux Kernel Uses RCU in Unloadable Modules

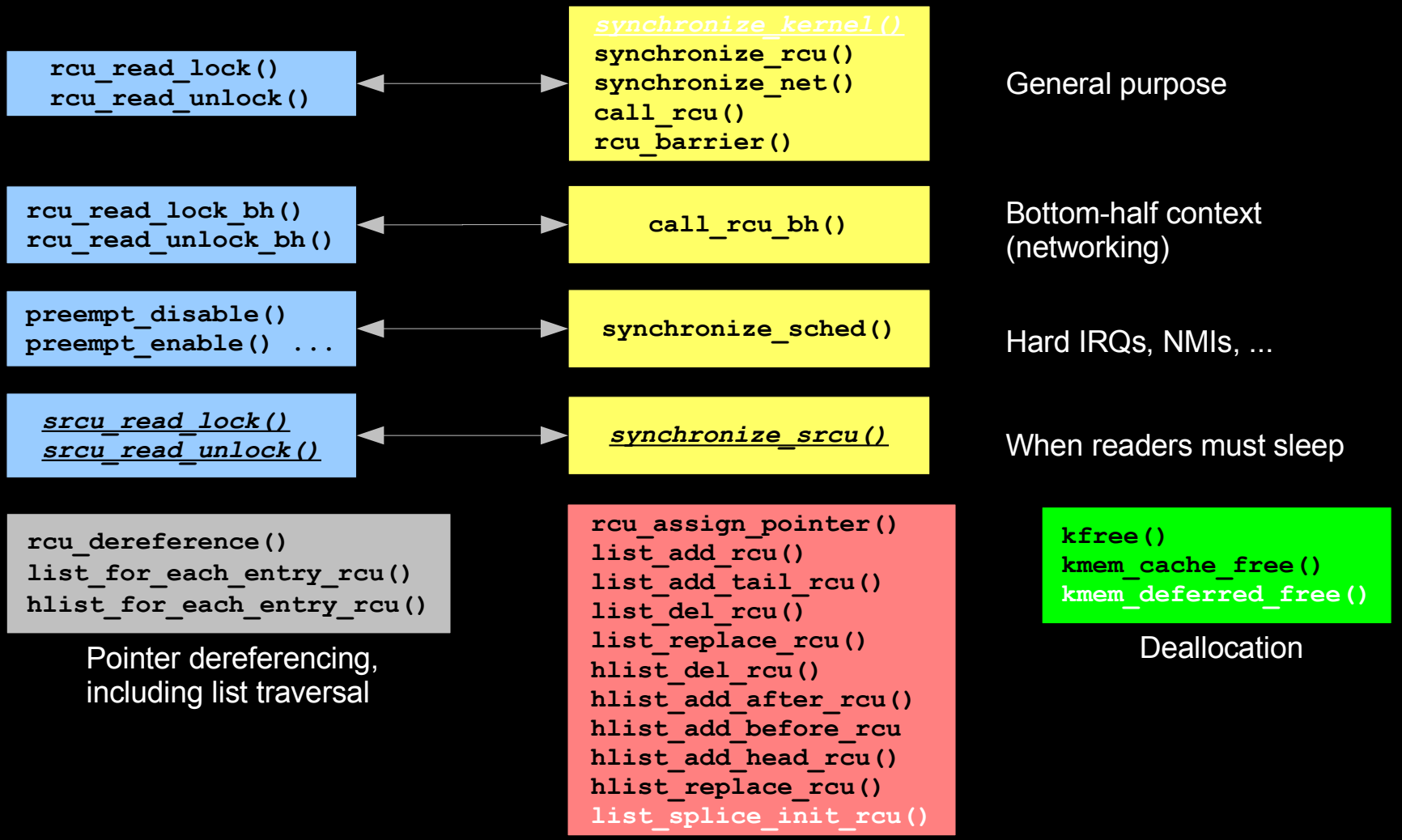


2.6.15

Linux Kernel's RCU Readers Need to Sleep

- **For more than a decade, “I need my RCU readers to be able to sleep” meant that the speaker didn't really understand RCU**
- **Until 2006, when I found someone who really did need RCU readers to sleep**
- **Hence SRCU...**
  - See http://lwn.net/Articles/202847/ for more details

Linux Kernel's RCU Readers Need to Sleep

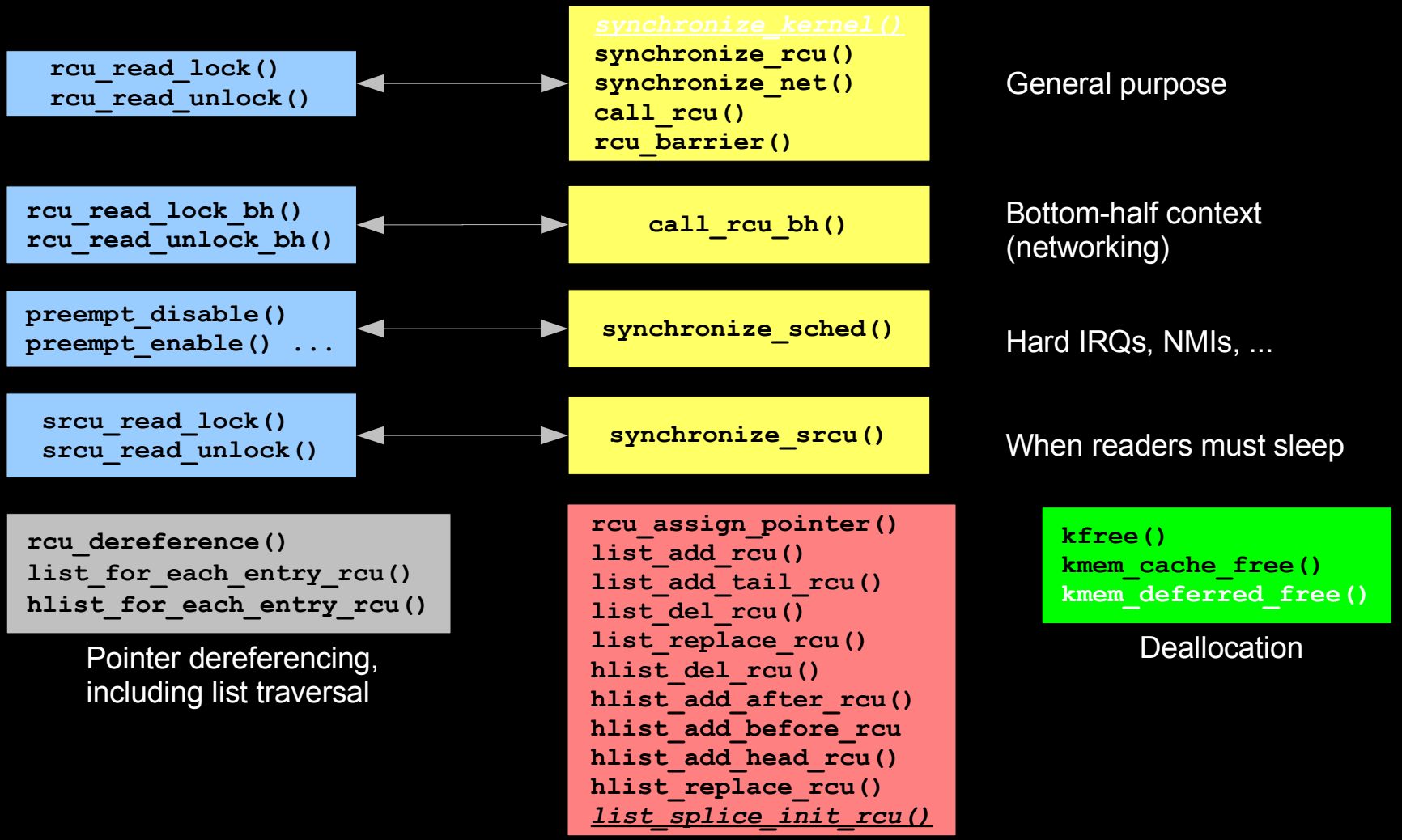


2.6.19

Linux Kernel Does Sophisticated List Manipulation

- **Reap an entire list while being traversed by RCU readers**
- **We were going to open-code it, but Christoph Hellwig made us create a primitive for this situation**
  - Corey Minyard does the heavy lifting

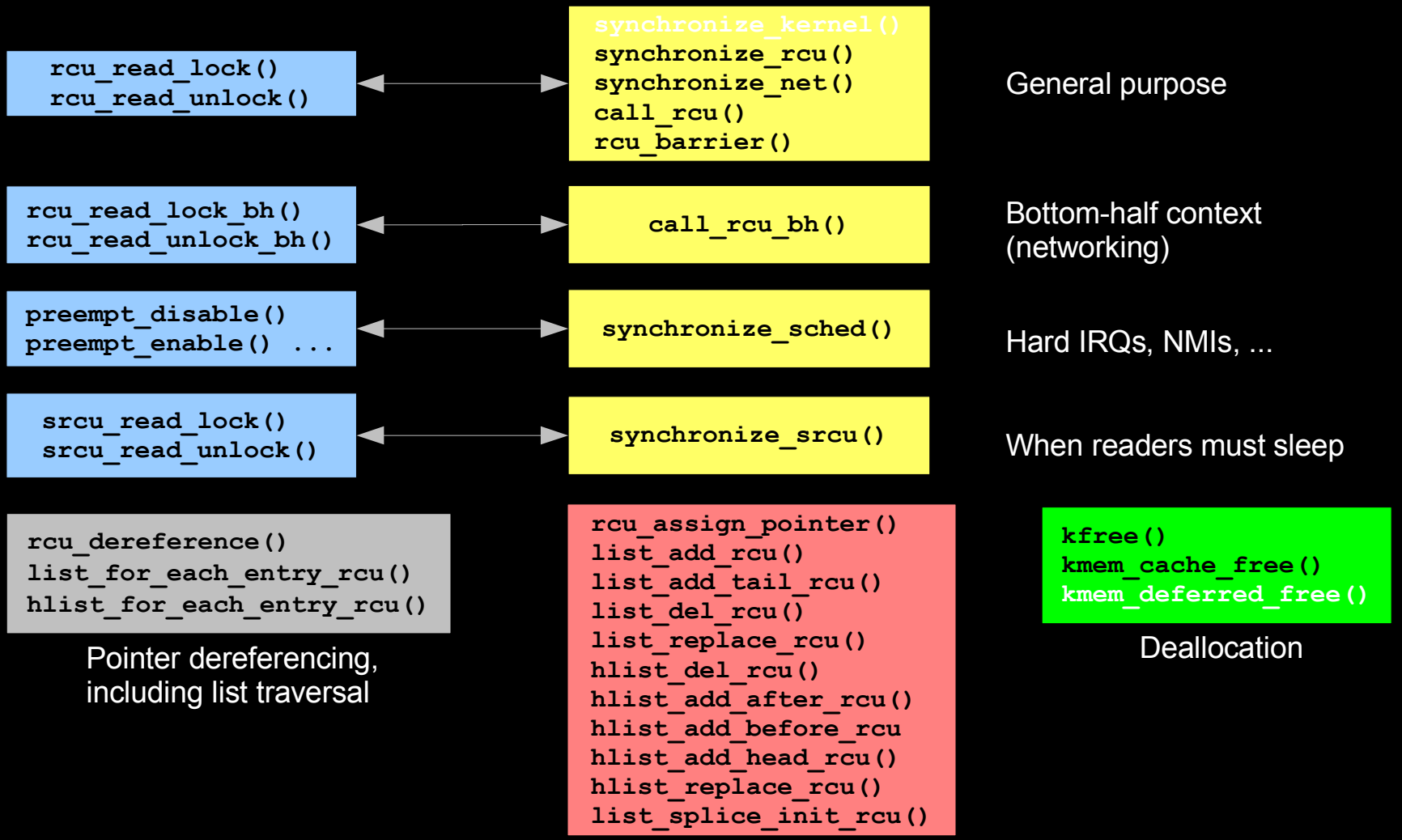
Linux Kernel Does Sophisticated List Manipulation



2.6.21

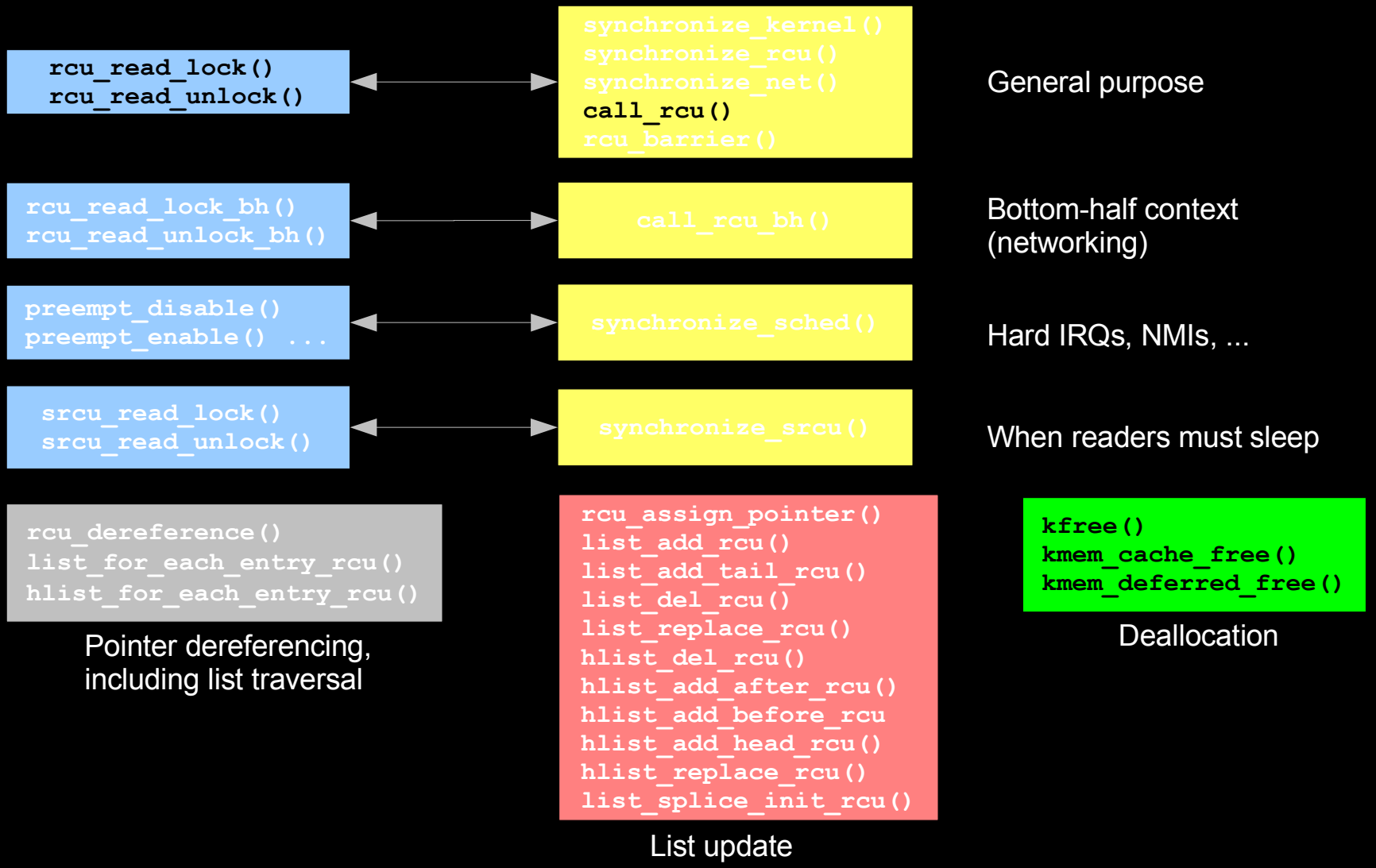
So Where Does That Leave RCU Today?

Linux RCU API as of 2.6.24

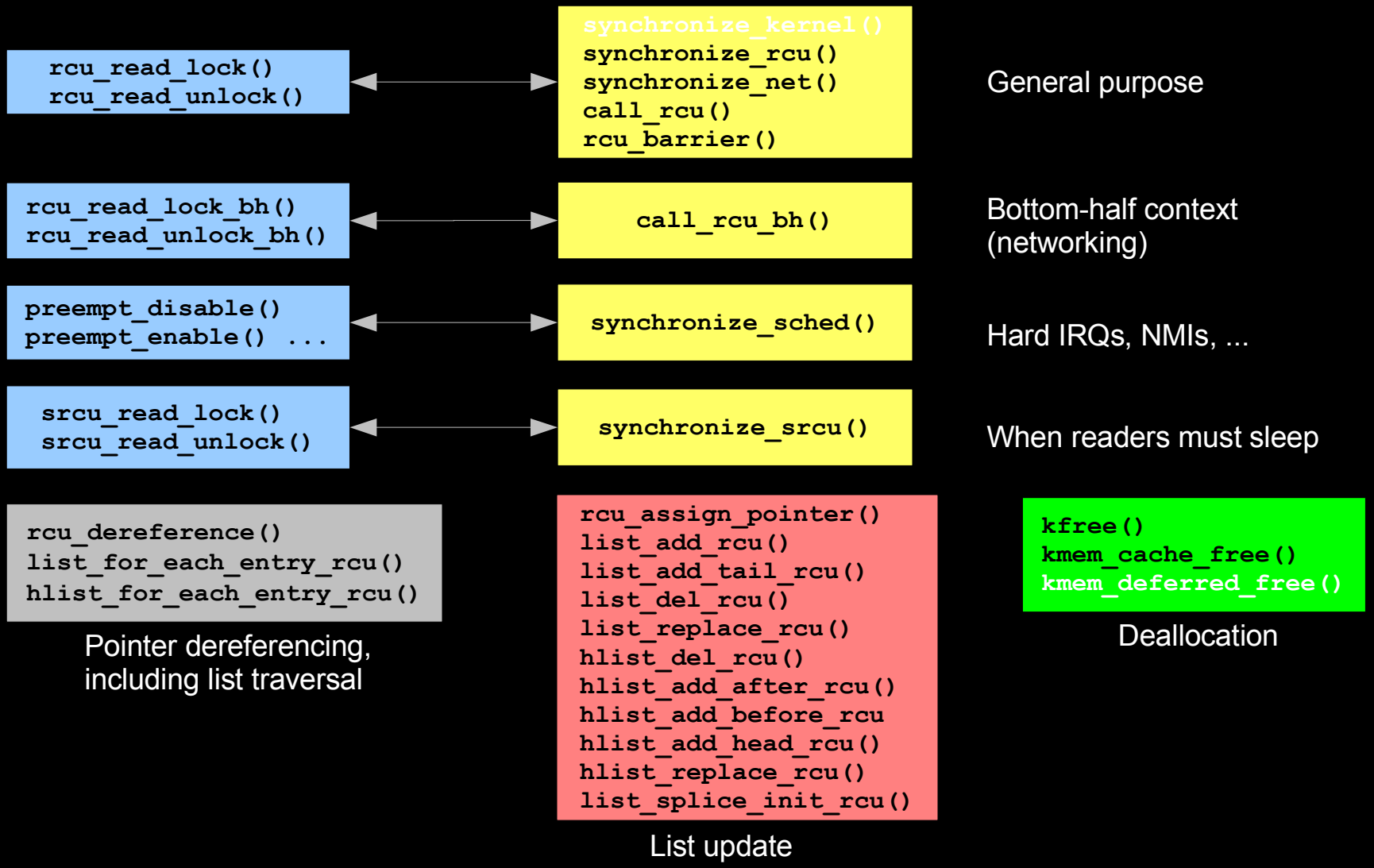


Summary of How Linux Changed RCU

This is Your Technology



This is Your Technology on Linux: 6x API Increase



Lessons Learned From the RCU Experience

Lessons Learned From the RCU Experience

- **Linux runs an incredible variety of workloads**
  - Embedded, realtime, desktop, network, server, supercomputer...
- **Linux powers significant networking infrastructure**
  - Linux ***is*** the firewall; it is not protected but rather protects
- **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 pays off in more than two years
- **Technology developed in more-protected environments will need serious modifications!!!**
  - Putting technology into Linux is a rewarding learning experience

Conclusion

Contributing Technology to Linux is Extremely Rewarding

But not to be taken too lightly!!!

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM, IBM (logo), e-business (logo), pSeries, e (logo) server, and xSeries 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.

Questions?