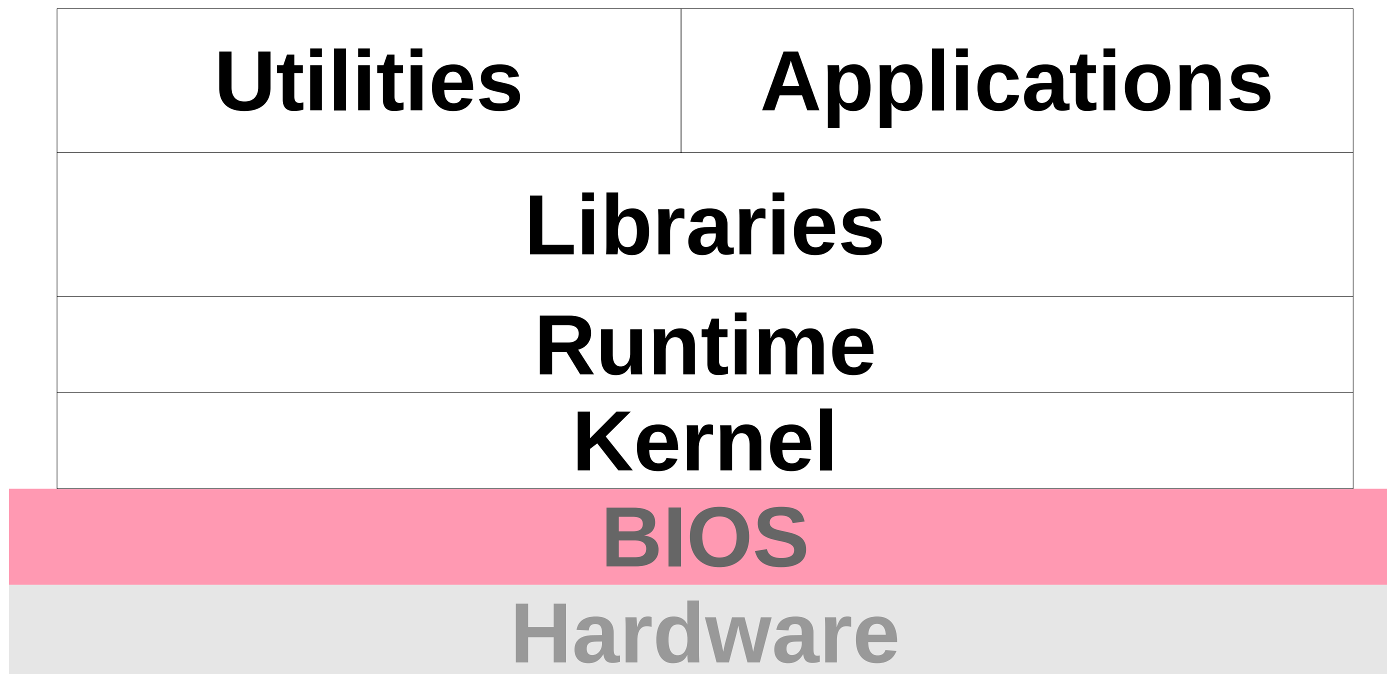
Introducing TM into the "Real World"

Ulrich Drepper

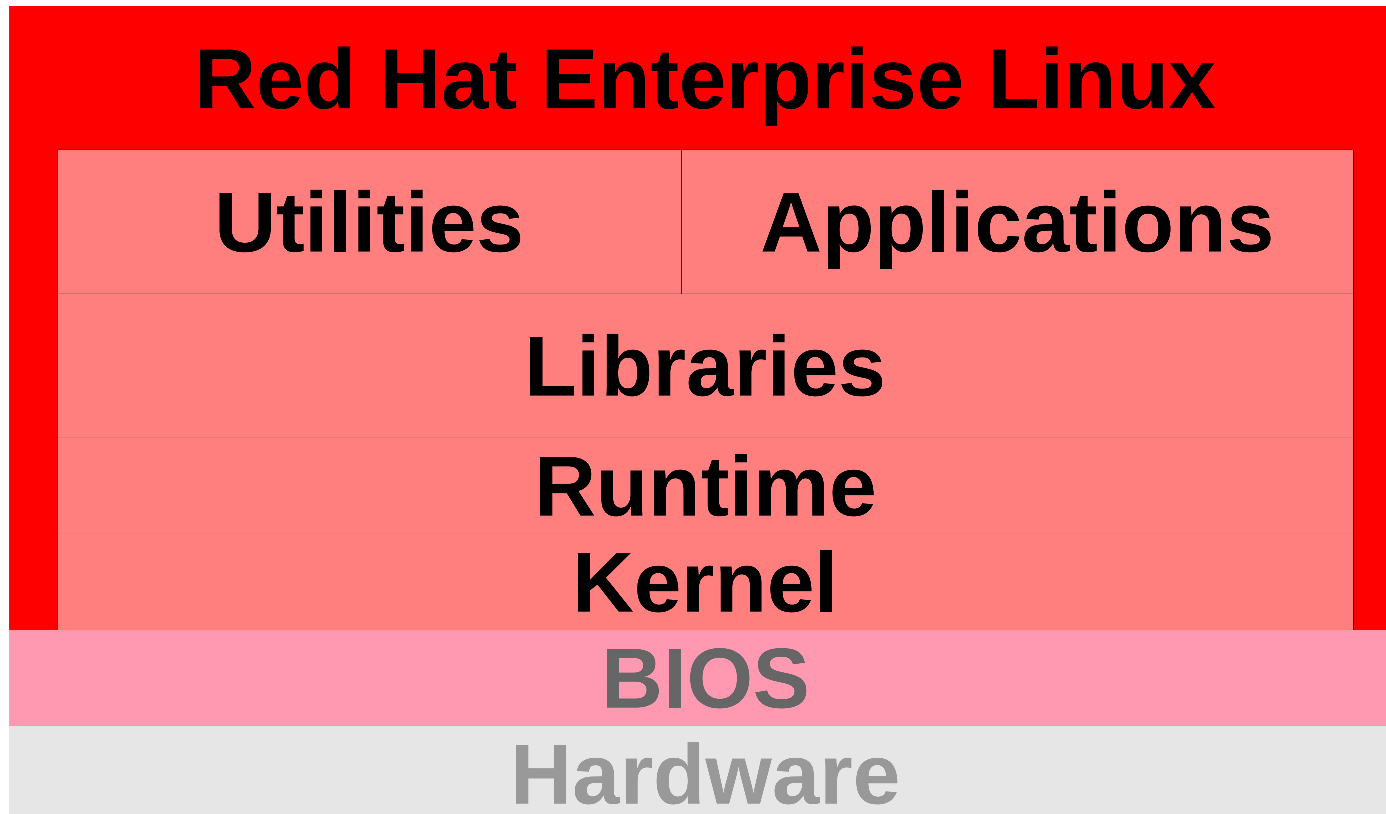
redhat

OS Vendor Situation



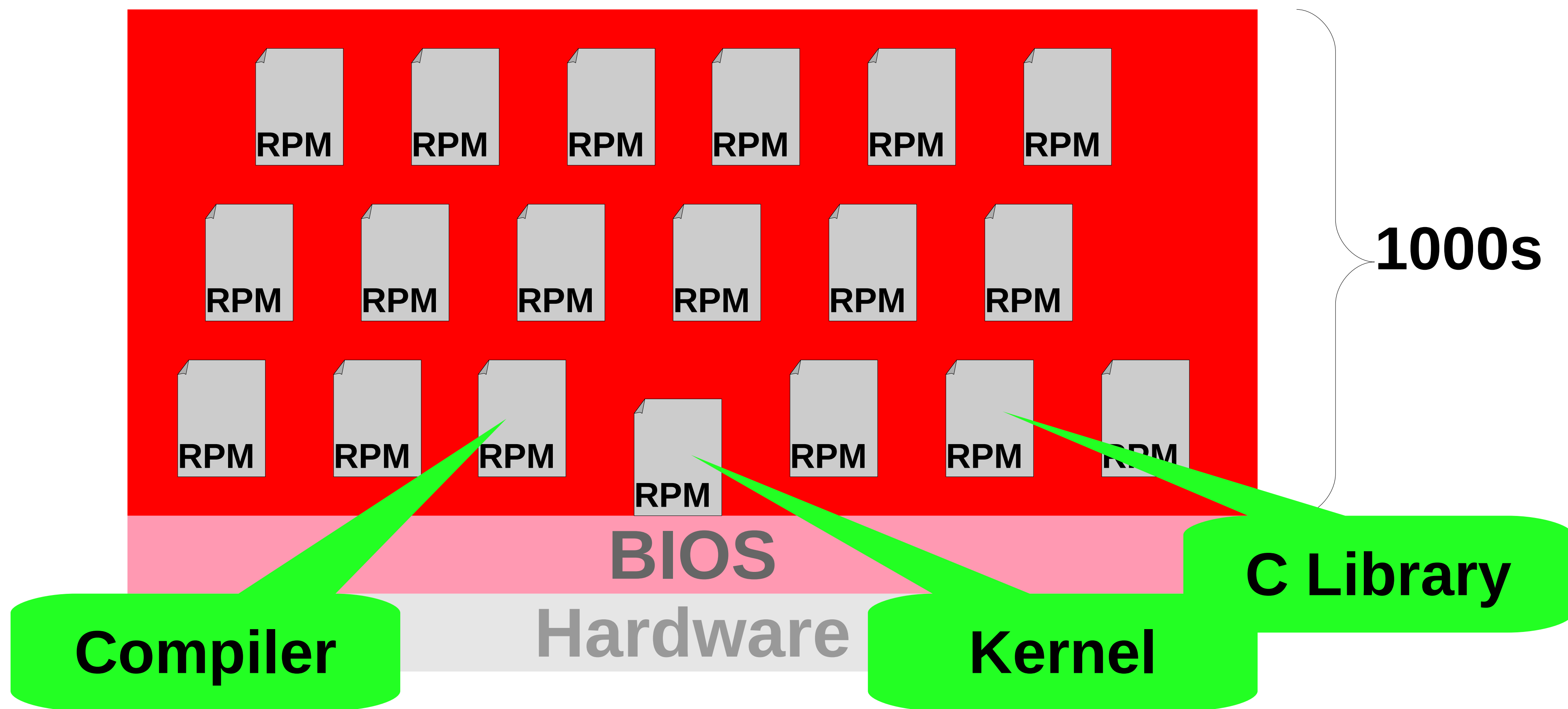
redhat

OS Vendor Situation



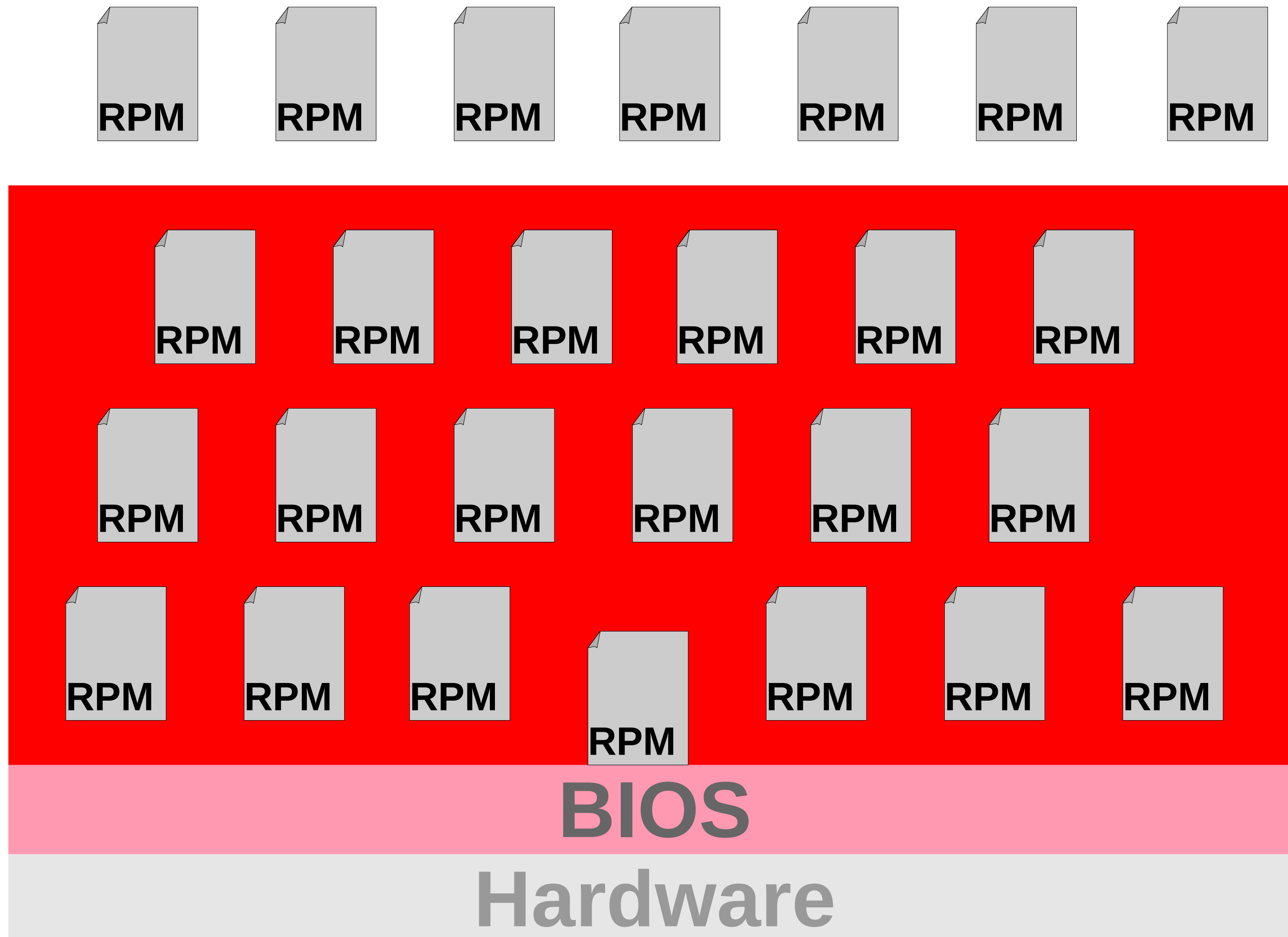
redhat

OS Vendor Situation



redhat

Plus 3rd Party Applications

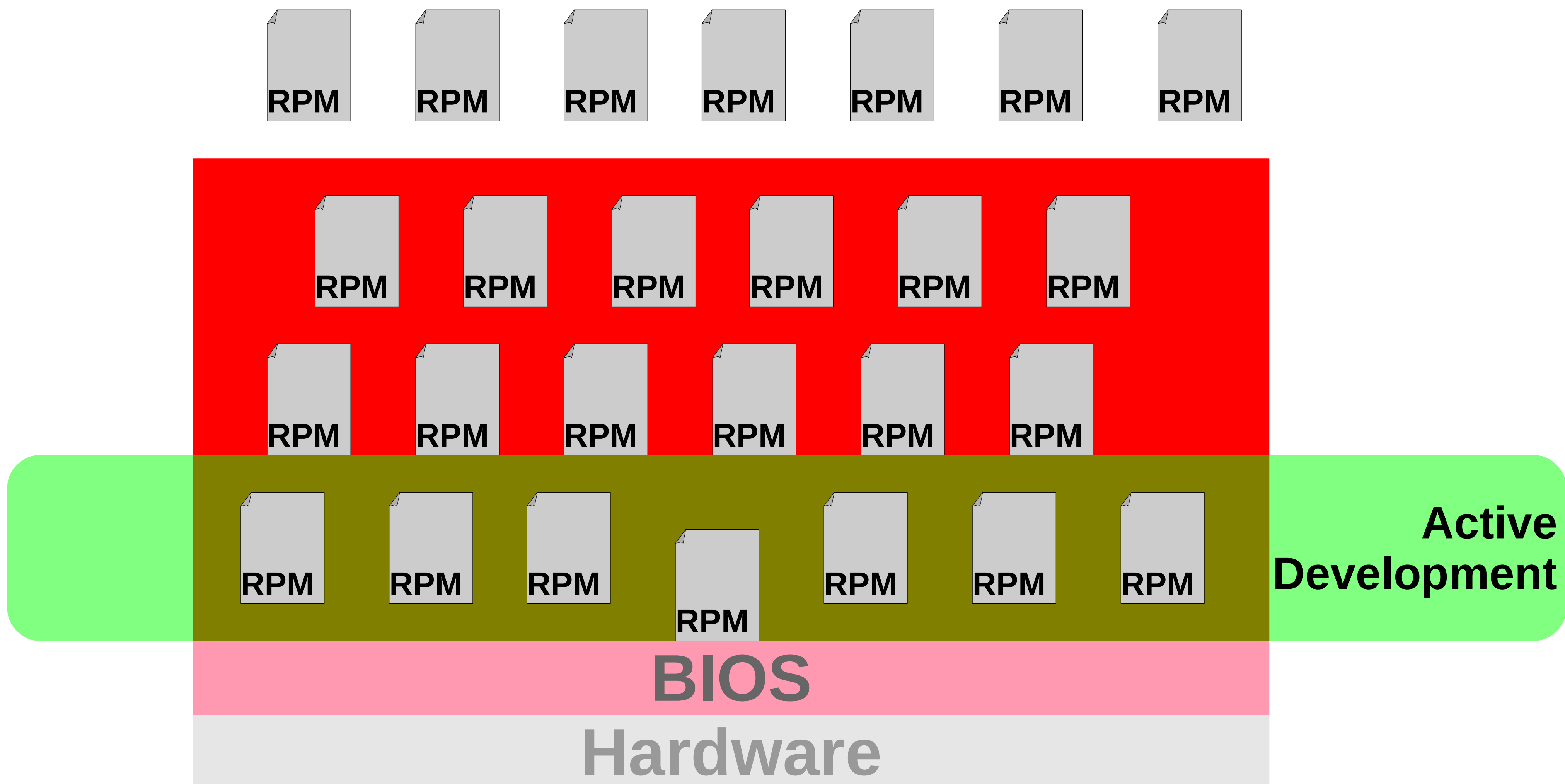


Challenges

- No company can effort maintaining 1000s of packages
- Constantly updated *upstream* packages
  - Any local change means additional work
  - Pushing changes *upstream* requires generalization
- Different guarantees
  - Security fixes
  - & + bug fixes
  - & + API/ABI guarantees

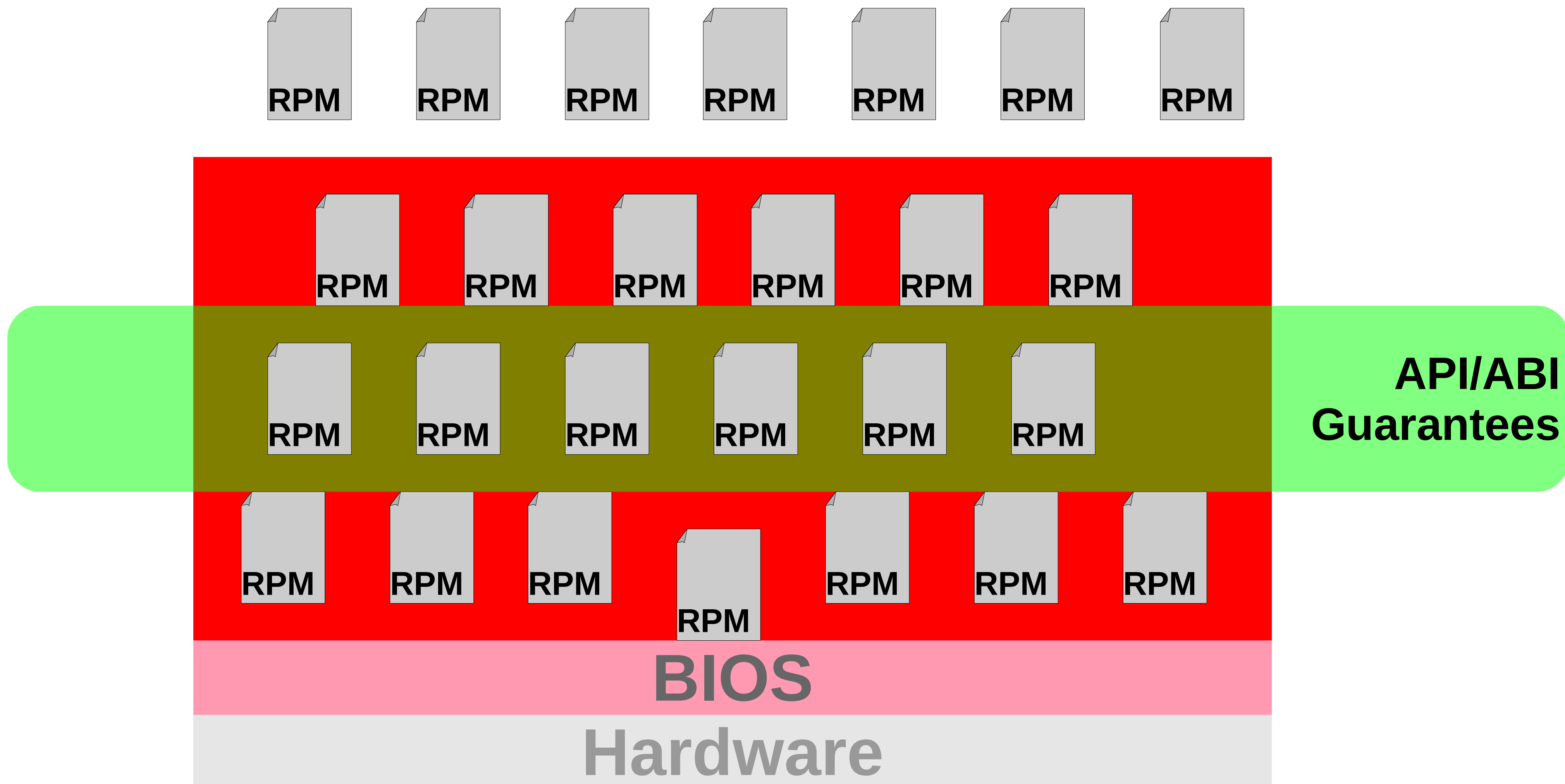
redhat

Levels of Support



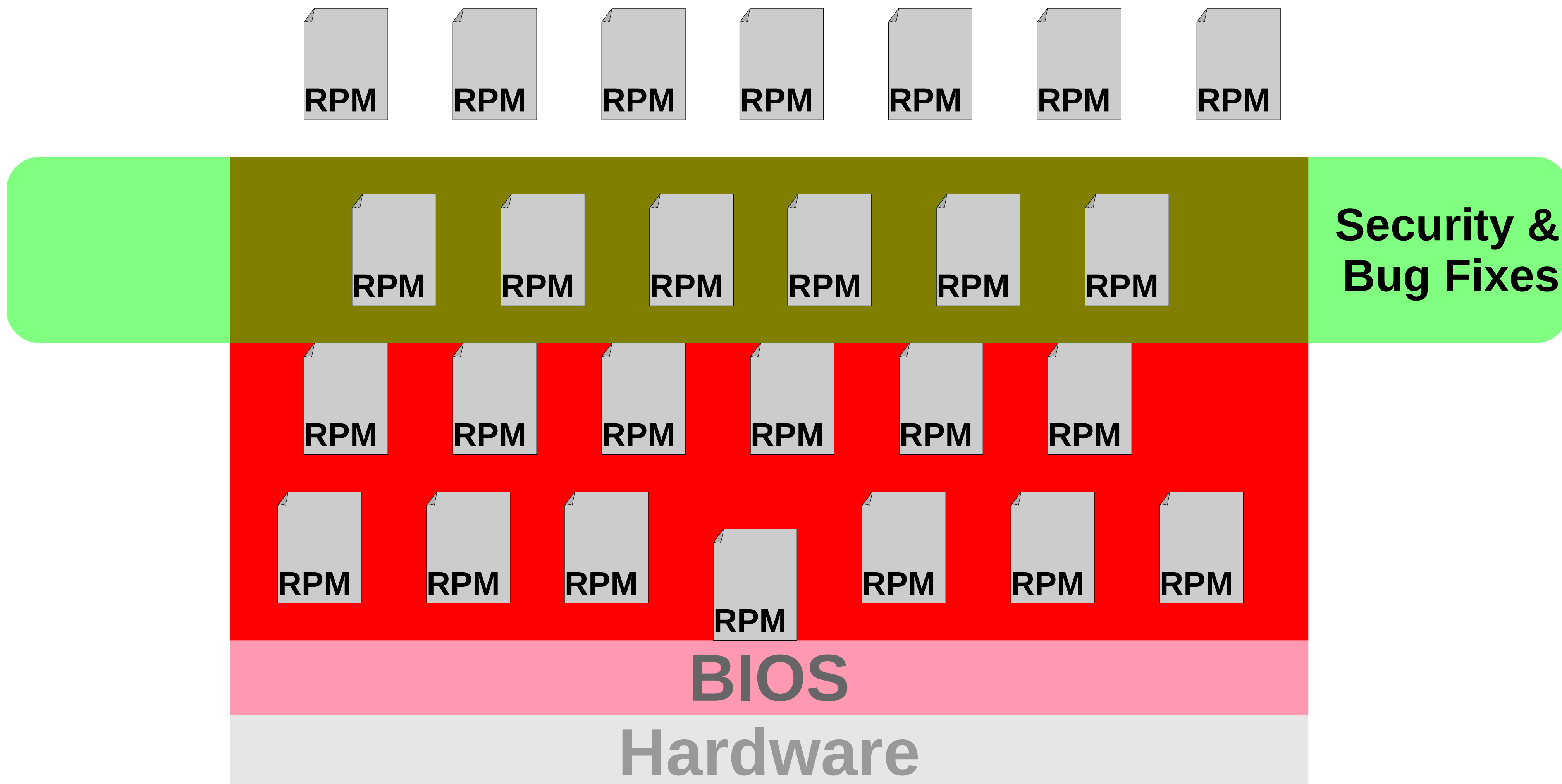
redhat

Levels of Support



redhat

Levels of Support



redhat.

Guarantees

- No ABI breakage
  - At all within major release
  - For core packages among all releases
- No regressions
  - Same performance characteristics
- Updated
  - Technology update during long life-time of release

redhat.

Programming Practices

- Dynamic linking prevalent (good)
- Programmers misuse dynamic linking for abstraction (bad)
- Result:
  - Every DSO might find their way (directly or indirectly) into some applications
  - No incompatible runtimes possible system-wide
  - Duplicated system functionality likely cause problems

Performing TM implementation

- Dependencies
  - Hardware
    - Lock-free data structure implementation
    - Possibly virtualization
  - Thread implementation
  - OS Scheduler
- Deep integration into runtime needed for performance
  - Code inlined by compiler
  - Fast access to thread-local data

redhat

What Does This Mean?

- ***There can only be one TM implementation***
- ***No negative performance impact on code not using TM***
  - Strong isolation likely not a possibility
- Implementation must be flexible
  - Handle different STM implementations
    - No clear overall winner
  - Handle (different) HTM implementations
  - Co-exist with HTM-based lock-free data structures
  - Future-safe in general
    - At least backward compatible

TM In Existing Code

- Unlikely only new code used in TM binaries
- TM-ify existing code
- Happens over time
  - (Possible) performance problem found by profiling
  - Create TM version of library interface
  - Rinse and repeat
  - For actively supported code part of OS vendor's job
- Must ***not*** require recompilation code using libraries
- Examples:
  - String functions: `strcpy memcpy memmove`
    - Needs support for STM
  - `malloc`: special version needed

Separate Code Paths

- Remember: no slowdown for non-TM code
- Not possible:

```
int foo(int *arr, int b) {
  if (in_TM) begin_tm();
  int c = 42;
  for (i = 0; i < b; ++i)
    int v = in_TM ? read_val4(&arr[i]) : arr[i];
    c = MAX(v,c);
  }
  if (in_TM) end_tm();
  return c == 42 ? c : -1;
}
```

- Not realistic for more variants (HTM, …)
- Increased I-cache footprint

redhat.

Starting Transaction

- One or more implementations: select one
- On restart: maybe select another

```
variant = begin_transaction(available_set);
switch (variant) {
  case var_single_thread: goto code_single_thread;
  case var_stm: goto code_stm;
  case var_htm1: goto code_htm1;
  case var_htm2: goto code_htm2;
}
```

- `begin_transaction` is `setjmp`-like for restart

redhat

Mixing TM-safe and TM-unsafe Code

- Initially most code not TM-safe

```
int foo(int *arr, int b) {
  __tm_atomic {
    int s = 0;
    for (int i = 0; i < b; ++i)
      S = bar(arr[i], s);
  }
  return s;
}
```

- What if `bar` is legacy code?
  - With side effects?
- Must annotate existing functions

redhat

Declaring Existing Code

- Pure functions need no TM-safe variant
  - `__attribute__((tm_pure))`
- Functions with TM-variants must be recognizable
  - `__attribute__((tm_callable))`
- Functions which might get TM-variants should test for them
  - `__attribute__((tm_unknown))`
- Functions which cannot be TM-safe (side effects, …)
  - `__attribute__((tm_irrevocable))`
- Header files indicate which case

redhat

Minimize Changes

- Minimize header file changes:
  - Marking all functions creates conflicts with upstream sources
  - Better: block-level marking
    - `#pragma TM push(tm_callable)`
  - Or: compiler command line switches
- Minimize source changes:
  - No marking of individual memory accesses
    - Required compiler support
    - Still: optimizations for thread-local memory access
      - Aliasing analysis important
  - No need to duplicate source code to get multiple variants

redhat

Minimize Changes

- Generate variants
  - Compiler knows all variants (command line parameter)
  - Compiler decides automatically for static functions
  - Attributes in header files specific `tm_callable`, etc
  - Function attributes to overwrite

```
int __attribute__((tm(“stm,asf”)))
foo(int *arr, int b) {
  __tm_atomic {
    ...
  }
  return s;
}
```

Function Pointers

- Two possibilities
  - Function prologue contains demultiplexer
  - Select variant when determining pointer
- Problem:
  - TM variant not represented in type system
  - How to ensure calling through function pointer is safe?
  - Variant must be determinable from pointer
  - Demultiplexer adds overhead
    - Violates “no performance penalty” condition
    - Caller might make wrong initial decision about mode
      - Costly restart
  - Not rare: C++ virtual function tables

redhat

Representing Variants

- Separate functions for variants really needed
- How to address them?
- Possibility #1: name mangling
  - Conflict with other name mangling (e.g., C++)
  - Not scalable with many variants
- Possibility #2: alternate symbol tables
  - ELF demands currently one symbol table
  - Not really a problem to have multiple
  - One table for each variant

redhat.

Summary

- Need to describe current and future form of existing interfaces
- Minimal changes to do that
- Deep integration into system
  - Compiler, executable format, runtime, …
- Code made TM-aware must be picked up automatically
- No overhead in non-TM-aware variants
- Flexible ABI for future extension with backward compatibility
- Integration with HTM use for lock-free data structures

Velox: `http://www.velox-project.eu/`

Questions?

drepper@redhat.com | people.redhat.com/drepper