Managing the Solution Stack: Open Source and Closed Source Together

Russell Pavlicek

Senior Linux Architect

Professional Services

Cassatt Corporation

pavlicek@linuxprofessionalsolutions.com

Who is this Fat Geek?

- Linux user since 1995
- 20+ years in industry (Cassatt, DEC, Compaq, Gannett)
- Former Linux columnist for Infoworld, Processor magazines
- Authored one of the first books to explain the business of Open Source

What does he know about mixed solution stacks?

- Working for Cassatt Corporation
  - Agile automated infrastructure solution (booth 1510 on the Expo floor)
  - Closed source application (Collage) running atop RHEL
  - Our product fits in this category
  - Most of our customers are using mixed Open Source/closed source solution stacks

What's the problem?

- The age of the closed source solution stack is essentially over
- Most current solutions use a blend of Open Source and closed source elements
- How do you know when Open Source elements make more sense than closed source – or vice versa?
- What are the risks?

Some definition of terms

- Open Source
  - Software where the source code can be examined, modified, and redistributed without cost
  - Examples include: the Linux kernel, Apache webserver, Sendmail, Perl, Python, PHP, anything covered by an OSI-approved license, anything labeled “Free Software” by the FSF

More definition of terms

- Closed Source
  - Any piece of software which does not rise to the watermark set by Open Source
  - Examples include: the Java runtime, Oracle database, 99+% of the code produced by Microsoft (including their Shared Source initiative)

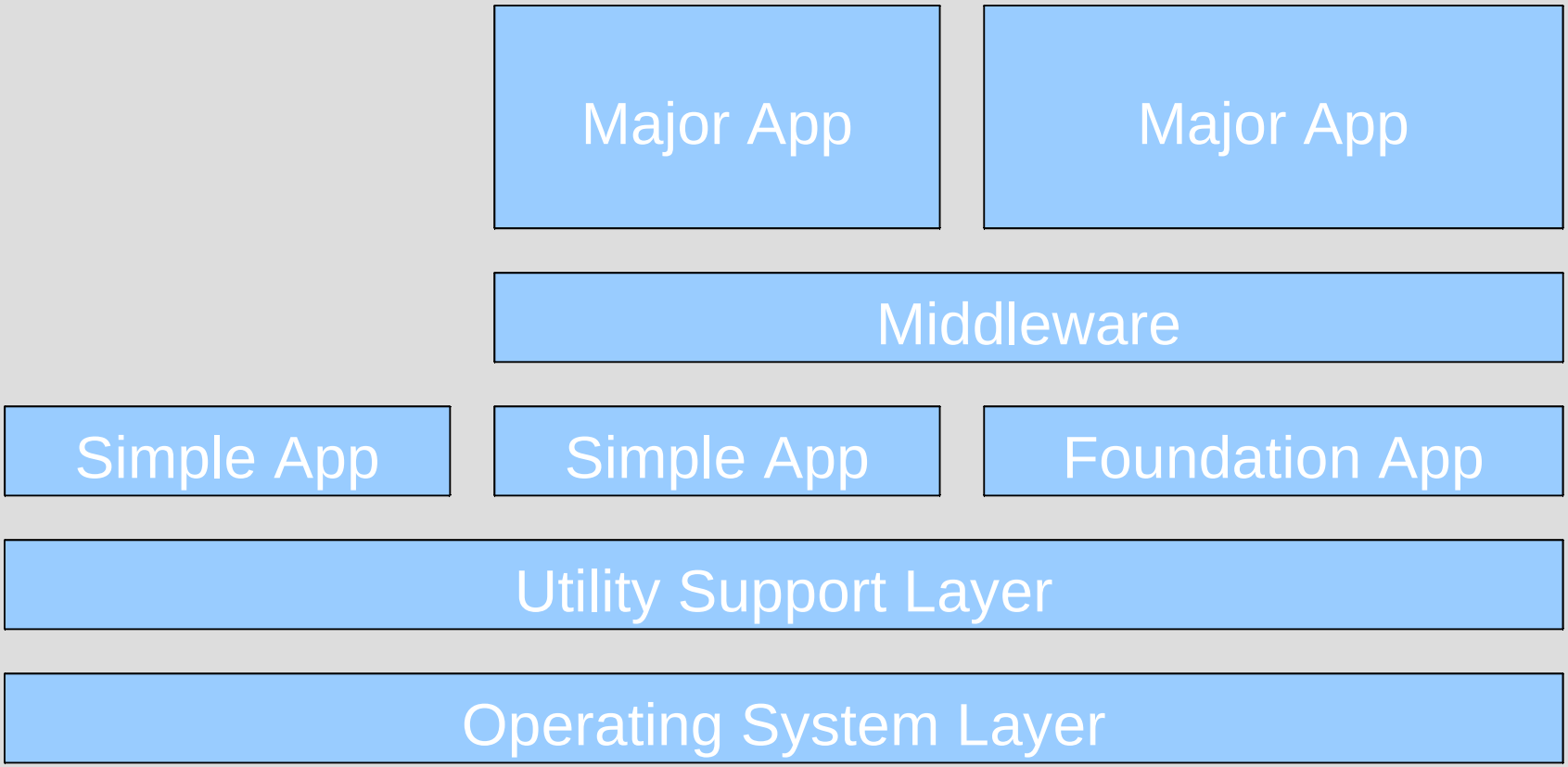
The old solution stack

- From about 1980 until the late 1990s, most solution stacks were closed source
  - Commercial applications with a variety of licensing terms and costs.
  - Little control over software direction
  - Cost control a major focus of IT management
  - Invasive Business Software Alliance audits feared

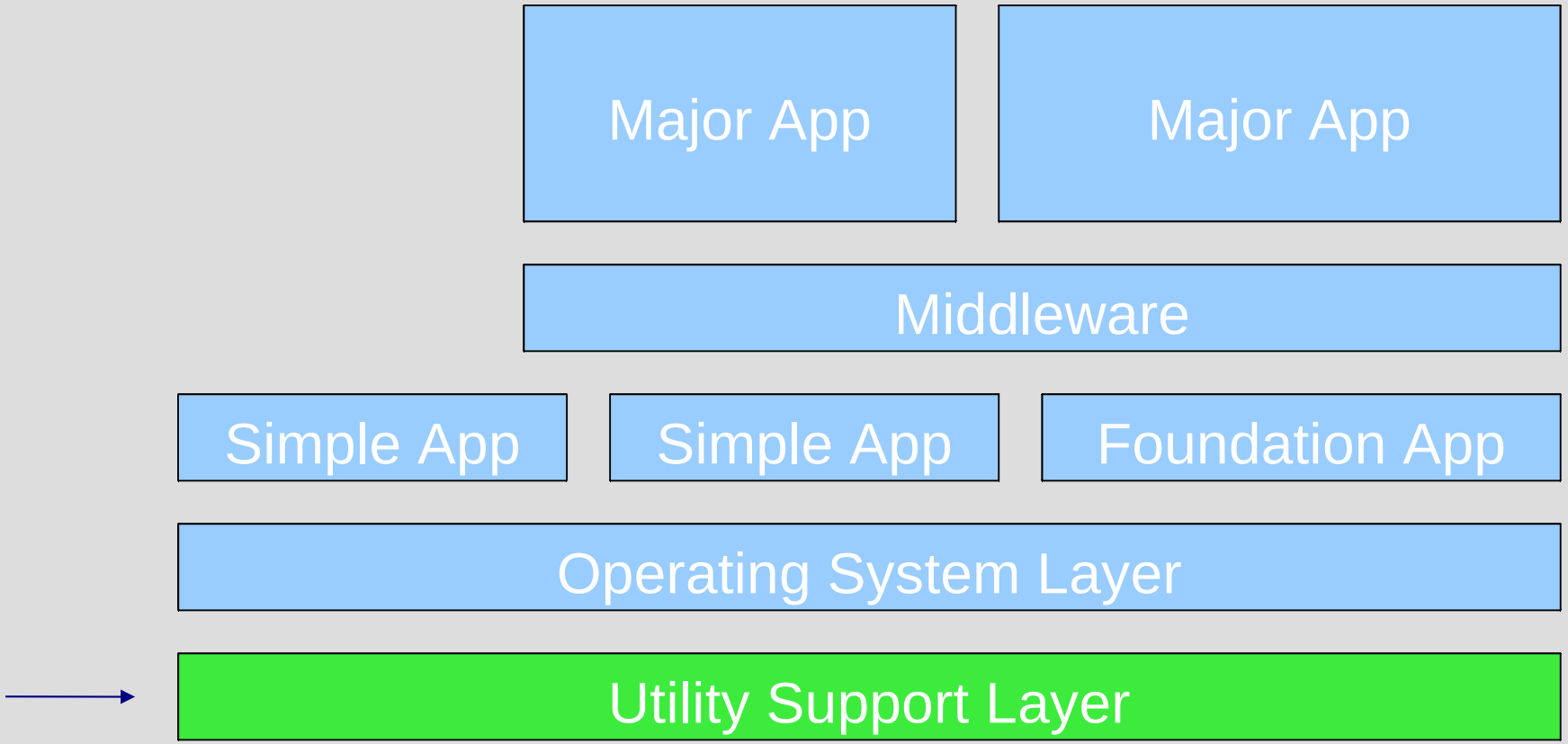
The new solution stack

- Mixture of Open Source and closed source elements
  - Potentially much less cost in licensing
  - Greater control over software development direction
  - Audits become much more tolerable; license bookkeeping greatly simplified

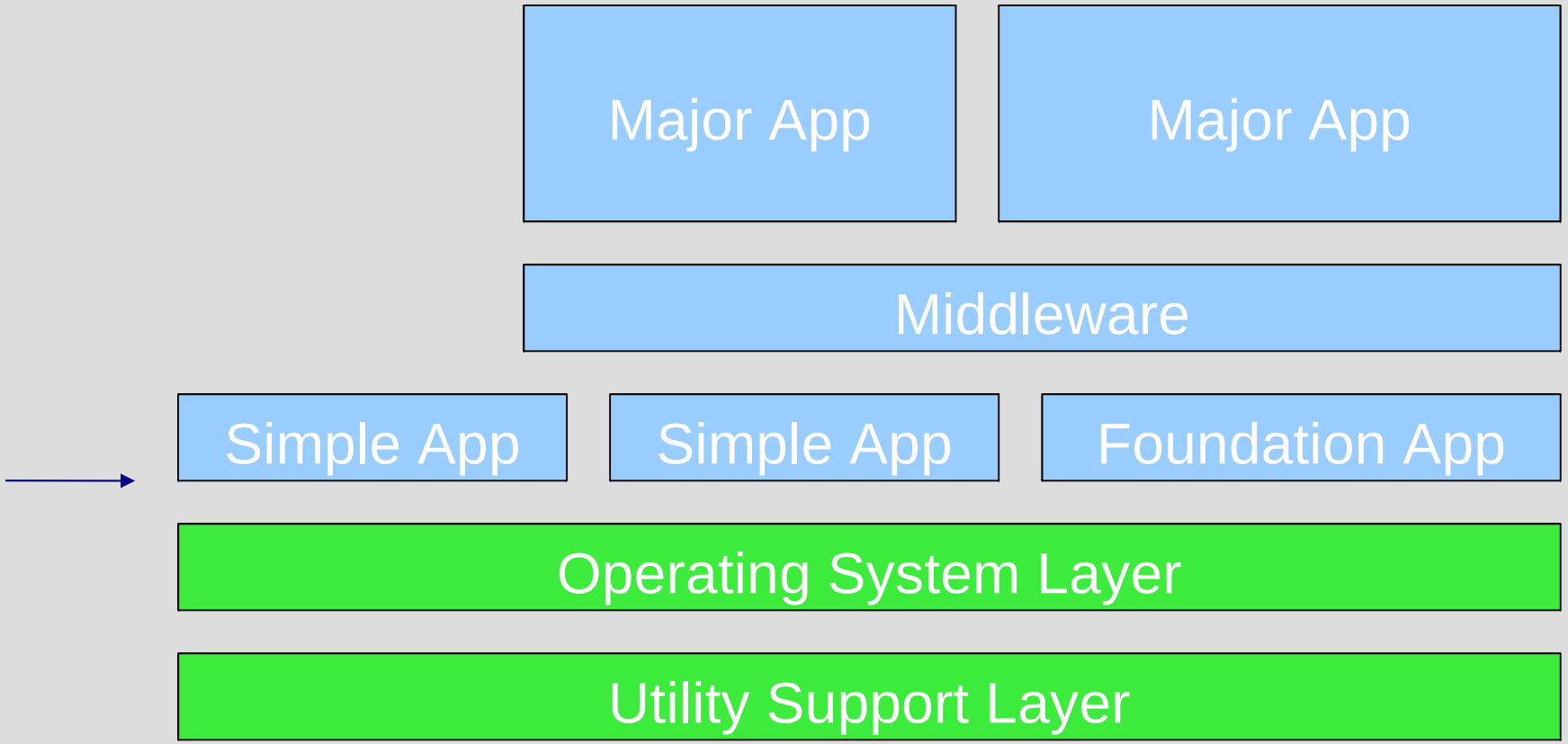
A Traditional Stack Diagram



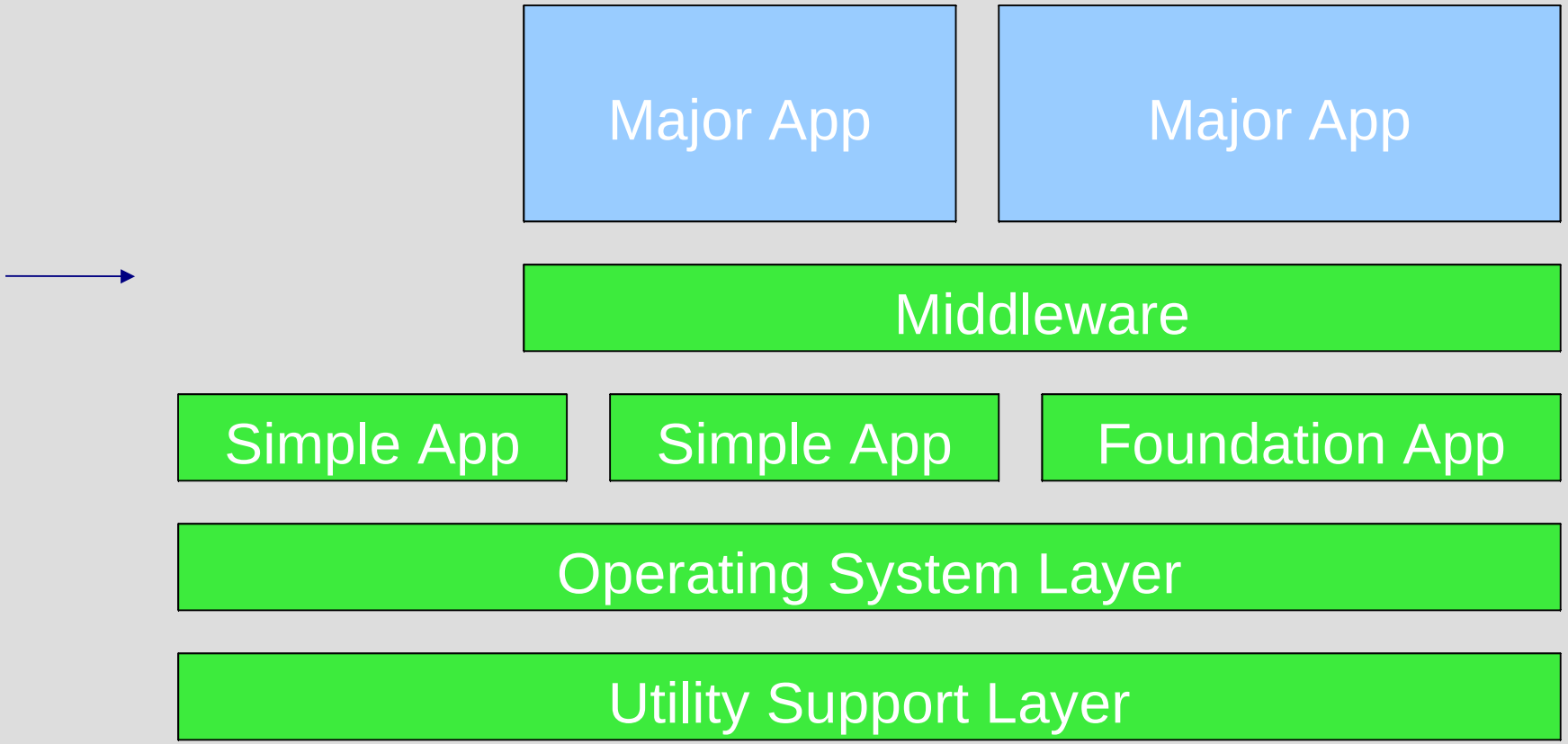
Water level: 1990s



Water level: Last Few Years



Water level: Future



Designing a mixed stack

- Rule: The waterlevel of common Open Source infrastructure will rise over time
- Application: Plan on increasing Open Source components over time
  - The reality of history; the trend is established
  - As the fundamental levels become solid, development increases on the application stack

Designing a mixed stack

- Rule: Optimize Modularity
- Application: Open Source is by nature a modularity enabler
  - Code reuse brings on modular practices
  - Modular code makes replacement of components easy
  - Avoid vendor lock-in
  - Maximize control: customize your stack

Designing a mixed stack

- Rule: Modularity enhances compatibility
- Application: Open Source is capable of partial stack upgrade
  - It is not always necessary to upgrade the entire stack when upgrading one component
  - Better control while avoiding the “upgrade whirlwind”

Designing a mixed stack

- Rule: Don't flush your toilets with Perrier
- Application: Don't pay top dollar for common infrastructure; use Open Source
  - Maximize ROI by treating Open Source like a public utility (Doc Searls)
  - Low cost infrastructure, available to all

Designing a mixed stack

- Rule: Pay for performance
- Application: If you need exceptional performance in a particular area, this could be an area to evaluate closed source options
  - Classic example: Oracle
  - But note that even Oracle has contenders: Enterprise DB, Greenplum on Expo floor

Designing a mixed stack

- Rule: Secret sauce costs money
- Application: If something is really a “secret”, you need to buy it or write it
  - Before you buy (closed source), make sure it really is valuable
  - If you write it, consider bolting it to an Open Source base; you don't need to publish your secret unless you redistribute the code (GPL)
  - Can save huge dollars allowing community to develop the non-secret part of the software

Designing a mixed stack

- Rule: The Future is not protected if the Past is neglected
- Application: Archival data should be in open formats
  - Open Source gives you open formats and the code needed to read it
  - Critical for governments; important for business
  - Information locked in a proprietary format accessed by an obsolete program on an unavailable operating system is lost

Designing a mixed stack

- Rule: Upgrades are easiest when the stack in intact
- Application: Consider simplifying stack per machine when upgradability is at risk
  - Linux easiest to upgrade when you stay fairly close to the as-distributed state
  - Some closed source products certify to certain distributions and/or certain kernels; conflicts can occur when products require different versions
  - Consider virtualization to segment multiple closed source products into VM “devices”

Designing a mixed stack

- Rule: Using an unpopular Open Source tool is risky, but using an unpopular closed source tool is deadly
- Application: If you use unpopular tools, they should not be closed source
  - Closed source tools rarely survive the death of the product or the company which provided it
  - Software escrow does not solve the ancient tool chain problem
  - Open Source tools can survive long after the provider is gone: e.g., Eazel and Nautilus

Questions and Answers

- Discussion
- Latest version of these slides will be available on linuxprofessionalsolutions.com after the conference
  - Just Google “russell pavlicek bibliography” and you'll find the right page