Introducing Log4j 2.0

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Presented For The Apache Foundation By
LINUX FOUNDATION

History of Apache Log4j

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Presented For The Apache Foundation By
LINUX FOUNDATION

Early Java Logging

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

- System.out and System.err
- Originally, Java didn't have any form of logging other than abuse of the standard output and error PrintStream objects.
- The shell running a Java application could simply redirect standard output and standard error to files.
- It was common to include a sort of debug system property to enable or disable logging messages.
- Many programmers still haven't upgraded from Logging 0.1.

Presented For The Apache Foundation By
LINUX FOUNDATION

Example Code

```
boolean debug =
    Boolean.getBoolean(“DEBUG”);

if (debug) {
    System.out.println(“Low priority.”);
    System.err.println(“High priority.”);
}

catch (final Throwable t) {
    t.printStackTrace();
}
```

The Original Log4j

APACHECON DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

- Written by Ceki Gülcü
- Provided a system of named Loggers that aided in categorising and filtering log messages.
- Allowed for more than two levels of logging similar to Apache HTTPD Server and other custom logging systems.
- Easily configurable using a Java properties file, XML file, or programmatically.
- Provided various ways to output and save log messages.

Example Code

```
private static final Logger LOGGER =
    Logger.getLogger(“org.apache.Foo”);

LOGGER.debug(“Low priority.”);
LOGGER.info(“Next level up.”);
LOGGER.warn(“High priority.”);
LOGGER.error(“Higher priority.”);
LOGGER.fatal(“Catastrophic priority.”);

catch (final Throwable t) {
    LOGGER.error(“Caught exception.”, t);
}
```

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014
Presented For The Apache Foundation By
LINUX FOUNDATION

Logback

- Gülcü went on to create SLF4J and Logback
- Provided parameterised log messages with placeholders.
- Added markers for additional filterable message information.
- Separated the logging API from the implementation.
  - Simple provider – console logging.
  - Logback – the main implementation.
  - Other bridges for code already using Log4j 1.2, java.util.logging, and Apache Commons Logging.

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014
Presented For The Apache Foundation By
LINUX FOUNDATION

Example Code

```
private static final Logger LOGGER =
    LoggerFactory.getLogger(“org.apache.Foo”);

final String msg = “{} priority.”;
LOGGER.debug(msg, “Low”);
LOGGER.info(msg, “Medium”);
LOGGER.warn(msg, “High”);
LOGGER.error(msg, “Higher”);

catch (final Throwable t) {
    LOGGER.error(“Caught exception.”, t);
}
```

Log4j 2

- Written by Ralph Goers to address SLF4J problems.
- Added new standardised SYSLOG format from RFC 5424.
- Decoupled loggers and configurations using a bridge pattern to allow for runtime configuration changes.
- Provided appender failover configuration to avoid ignoring appender exceptions.
- Numerous synchronisation and other performance bottlenecks fixed.
- Separate plugin interface for easy extensibility.

Example Code

```
private static final Logger LOGGER =
    LogManager.getLogger("org.apache.Foo");

final String msg = "{} priority.";
LOGGER.debug(msg, "Low");
LOGGER.info(msg, "Medium");
LOGGER.warn(msg, "High");
LOGGER.error(msg, "Higher");

catch (final Throwable t) {
    LOGGER.catching(t);
}
```

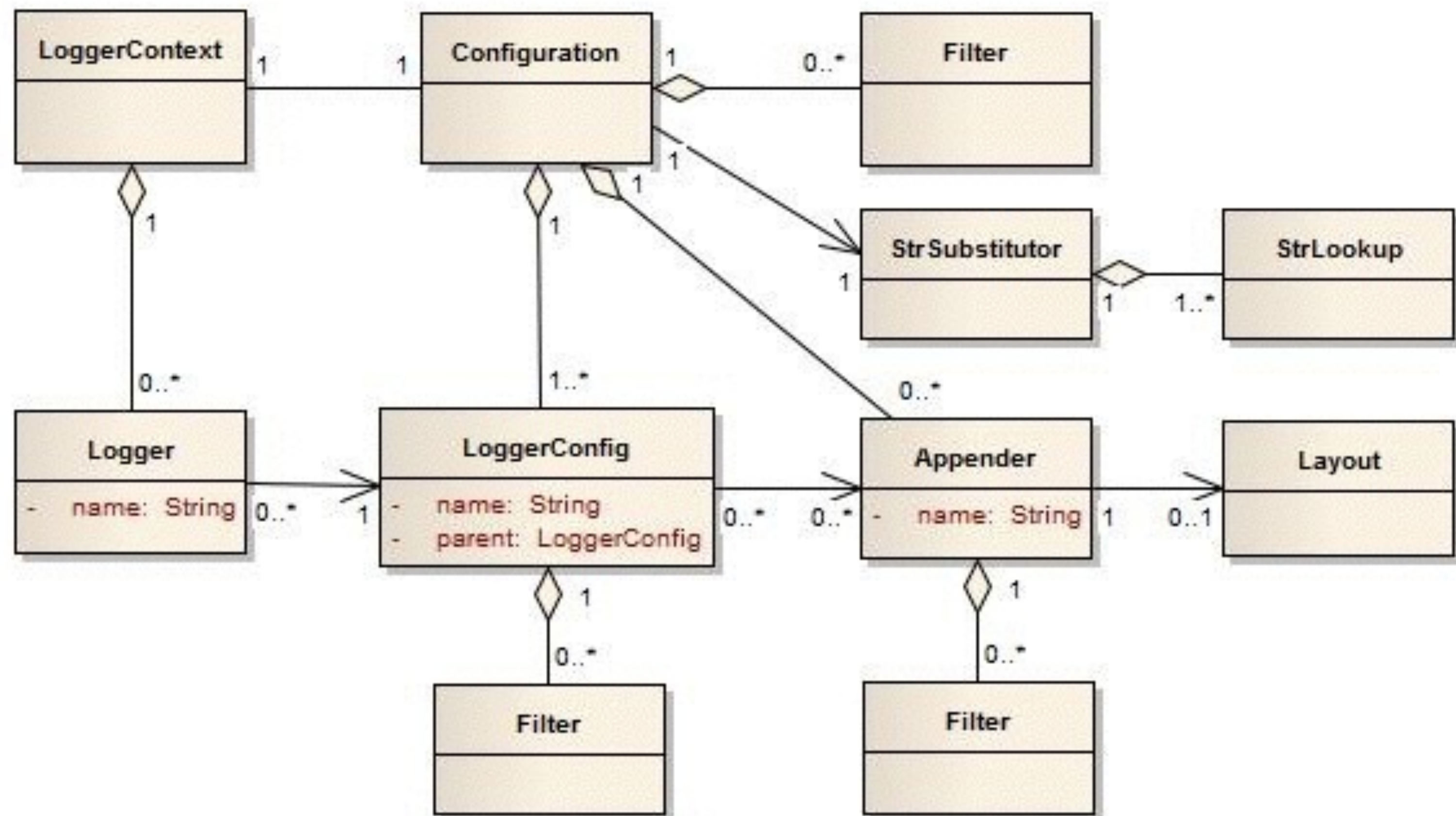
APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Presented For The Apache Foundation By
LINUX FOUNDATION

Log4j 2 API & Core Architecture

APACHECON DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Presented For The Apache Foundation By
LINUX FOUNDATION



class Log4j Classes
LoggerContext
Configuration
Filter
StrSubstitutor
StrLookup
Logger
- name: String
LoggerConfig
- name: String
- parent: LoggerConfig
Appender
- name: String
Layout
Filter
Filter
1
1
1
0..*
1
1
1
1
1
1..*
0..*
1..*
0..*
0..*
1
0..*
0..*
1
0..1
1
0..*
1
0..*

LogManager

- Root of the logging system.
- Finds an available LoggerContextFactory on initialisation.
- Used for getting and creating Loggers and LoggerContexts
- Provides a convenient way to get a Logger that uses printf-style parameterised messages instead of the default {}-style.

Presented For The Apache Foundation By
LINUX FOUNDATION

Logger

APACHECON DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

- Same concept as all the other logging frameworks.
- Each class usually has a private static final Logger.
- Use LogManager.getLogger() to get a logger named after the calling class. This is the standard named logger pattern.
- Use the various logging levels to log messages: TRACE, DEBUG, INFO, WARN, ERROR, FATAL.
- Also available: Logger.catching(Throwable), Logger.throwing(Throwable), Logger.entry(Object…), Logger.exit(ret)

Presented For The Apache Foundation By
LINUX FOUNDATION

Logger Hierarchy

- Loggers are named in a hierarchy similar to Java package names.
- “org” is the parent of “org.apache” which is the parent of “org.apache.logging”, etc.
- The root logger is the parent of all and is named “”.
    - This can be obtained through LogManager.getRootLogger()
- This hierarchy is used for configuring and filtering loggers.

Marker

- Simple mechanism to name and filter loggers.
- Independent of loggers, but can have their own hierarchy.
- Useful for aspect-oriented logging and other cross-cutting concerns (e.g., system initialisation).
- Similar to loggers, markers are obtained through the MarkerManager.getMarker family of methods.

LoggerContext

- Anchor point for the logging system.
- There can be multiple LoggerContexts active (e.g., on an application server or a servlet container).
- Used for tracking existing loggers and creating new ones on request.
- When multiple LoggerContexts are available, a ContextSelector is used for selecting the appropriate one.
  - Use a given ClassLoader to associate with a context.
  - Use JNDI lookups for named contexts.

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Presented For The Apache Foundation By
LINUX FOUNDATION

Configuration

- Part of the Core API.
- Represents a parsed configuration to be used with one or more LoggerContexts.
- Currently supports XML, JSON, and YAML file formats along with programmatic creation of this interface.
- Independent of the loggers and contexts it applies to in order to support live configuration updates.

Lookup

- Provides property variables for configuration files.
- Can be obtained through several sources:
  - Environment variables and system properties
  - JNDI
  - ServletContext
  - ThreadContextMap (MDC/NDC)
  - MapMessages
  - StructuredDataMessages

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Presented For The Apache Foundation By
LINUX FOUNDATION

LoggerConfig

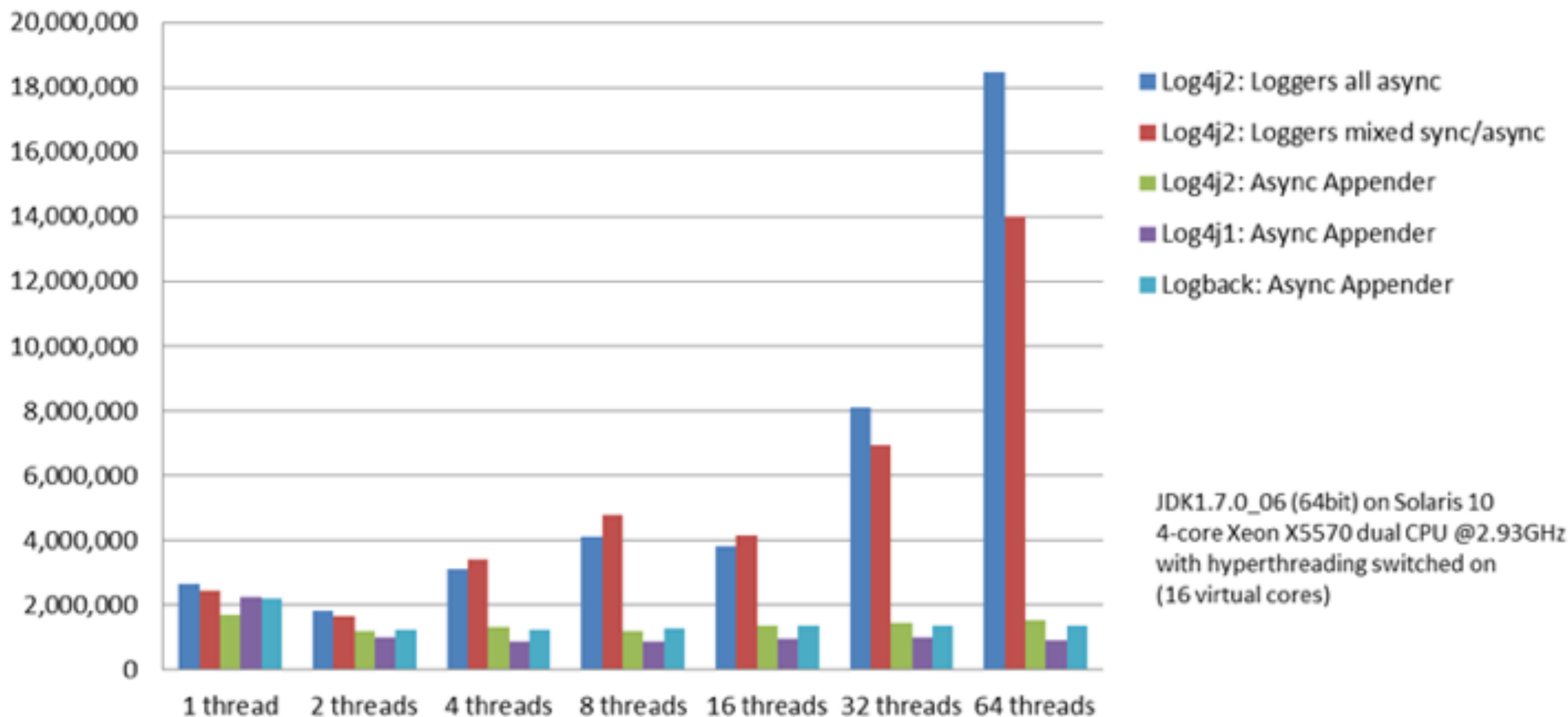
- Another part of the Core API.
- These represent the logger elements in the configuration file.
- Links one Configuration to an arbitrary number of Loggers.
- Filters are available at this level (e.g., through levels, markers, regular expressions, and diagnostic contextual information).
- Loggers pass along their logged messages to their respective LoggerConfig.

Appender

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

- Used for routing log messages to a physical destination.
- Contains a Layout object for unmarshalling LogEvent objects.
- Additively used based on the Logger hierarchy and config.
- Example appender types:
    - Console, File, OutputStream.
    - TCP, UDP, Syslog, Flume, and other network clients/servers.
    - SMTP, JMS, JPA, NoSQL, and other frameworks.
    - Failover appenders for handling appender errors.
    - Asynchronous logging for massive performance gains.

Presented For The Apache Foundation By
LINUX FOUNDATION


Async Logging Throughput (msg/sec) - higher is better
20,000,000
18,000,000
16,000,000
14,000,000
12,000,000
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
1 thread
2 threads
4 threads
8 threads
16 threads
32 threads
64 threads
Log4j2: Loggers all async
Log4j2: Loggers mixed sync/async
Log4j2: Async Appender
Log4j1: Async Appender
Logback: Async Appender
JDK1.7.0_06 (64bit) on Solaris 10
4-core Xeon X5570 dual CPU @2.93GHz
with hyperthreading switched on
(16 virtual cores)

Filter

- Selects which log events should be logged or not.
- Can use many types of contextual information to determine whether or not a log event should move along the logging system.
- In the flow from Logger to LoggerConfig to Appender, can be:
    - Applied to a Logger before LoggerConfig.
    - Applied to a LoggerConfig before any Appender.
    - Applied to a LoggerConfig for specific Appenders.
    - Applied to specific Appenders.

Pattern

- Configure what relevant log event data to output.
- Some data is what is provided by the programmer in the log call itself (e.g., the logger name, message, marker, throwable).
- Other information is dynamically calculated as needed (e.g., caller class/method/location, thread name, date/time, mapped and nested diagnostic context information).
- Usually used with a PatternLayout (which is also the default Layout if unspecified).

Layout

- Configures the output format of log events.
- Specifies common header and footer data to include.
- Besides plain text, there are other useful layouts:
  - Syslog (simple) and RFC 5424 (far more information).
  - HTML, XML, and JSON text.
  - Serialised Java objects.

Questions?