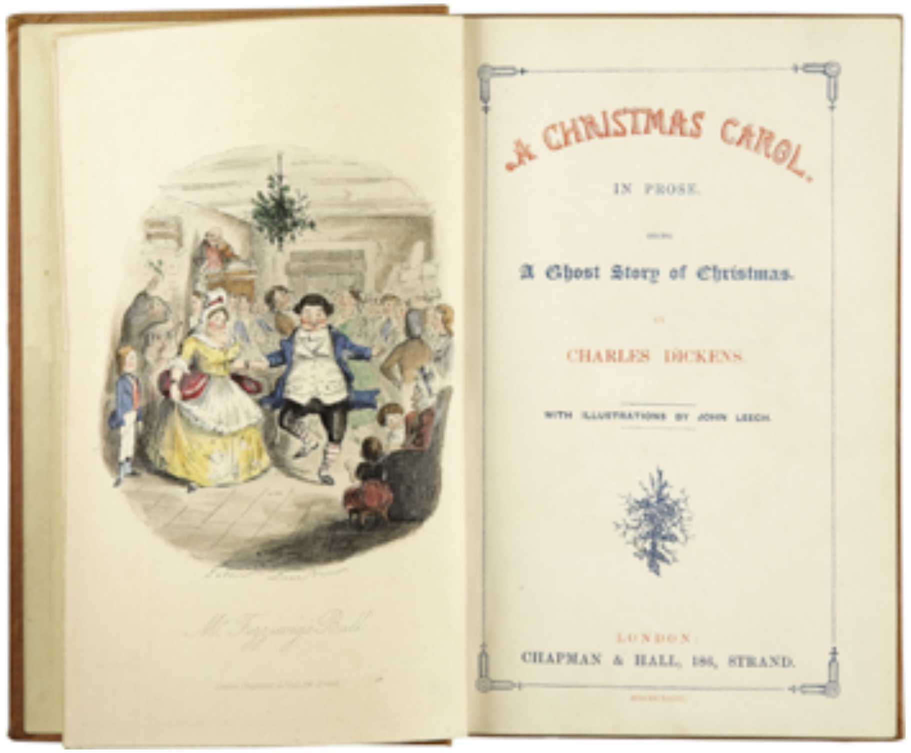
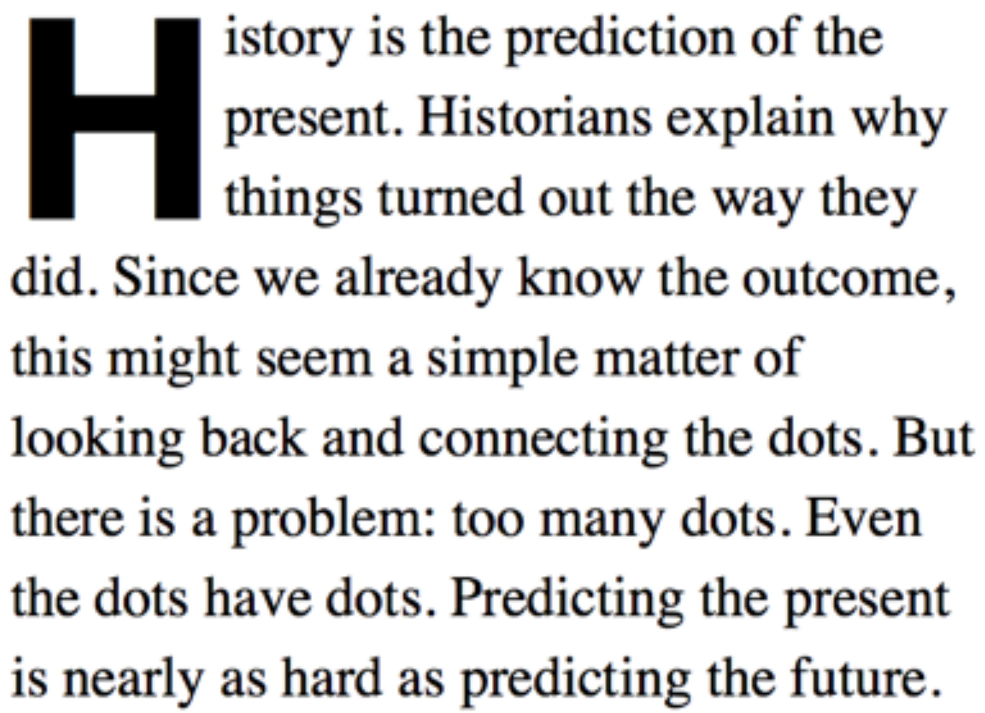
The Ghosts of Java Past, Present and Yet To Come

Steve Elliott
Oracle UK
8th December 2015

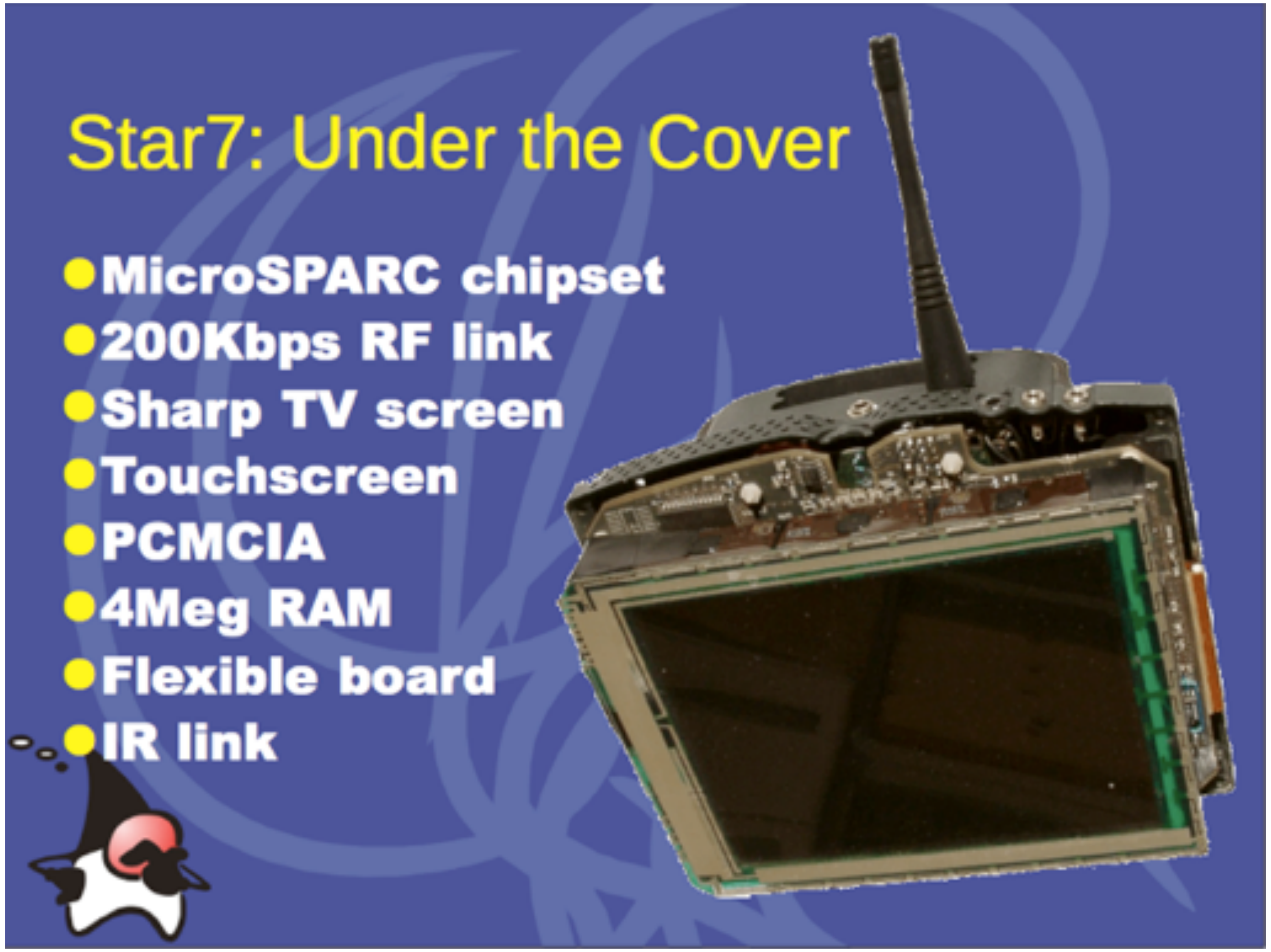

A CHRISTMAS CAROL.
IN PROSE.
A Ghost Story of Christmas.
CHARLES DICKENS.
WITH ILLUSTRATIONS BY JOHN LEECH.
LONDON:
CHAPMAN & HALL, 186, STRAND.
Mr. Fezziwig's Ball.

"It held a branch of fresh green holly in its hand…
But the strangest thing about it was, that from the crown of its head there sprung a bright clear jet of light, by which all this was visible; and which was doubtless the occasion of its using, in its duller moments, a great extinguisher for a cap, which it now held under its arm."



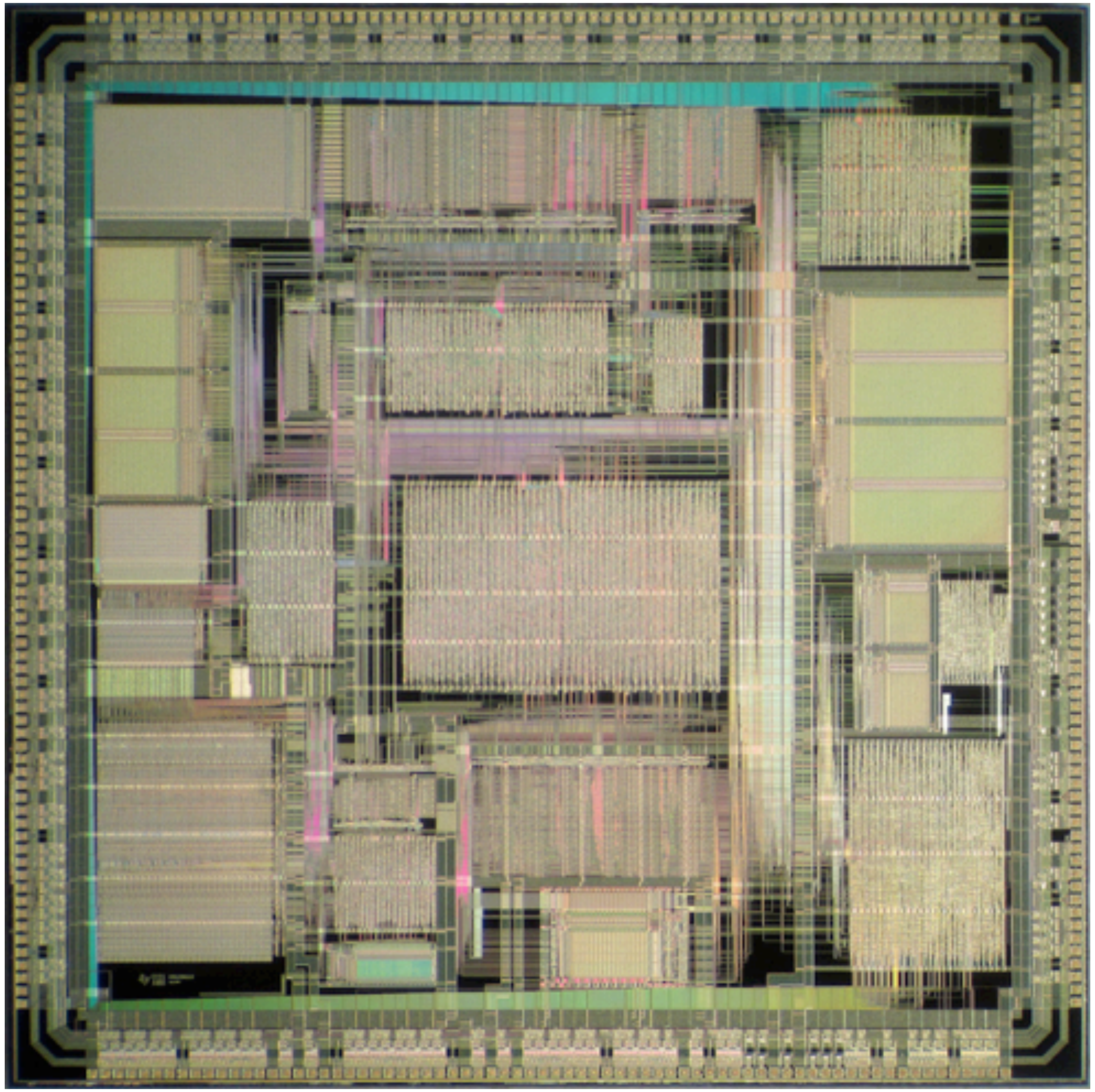
History is the prediction of the present. Historians explain why things turned out the way they did. Since we already know the outcome, this might seem a simple matter of looking back and connecting the dots. But there is a problem: too many dots. Even the dots have dots. Predicting the present is nearly as hard as predicting the future.

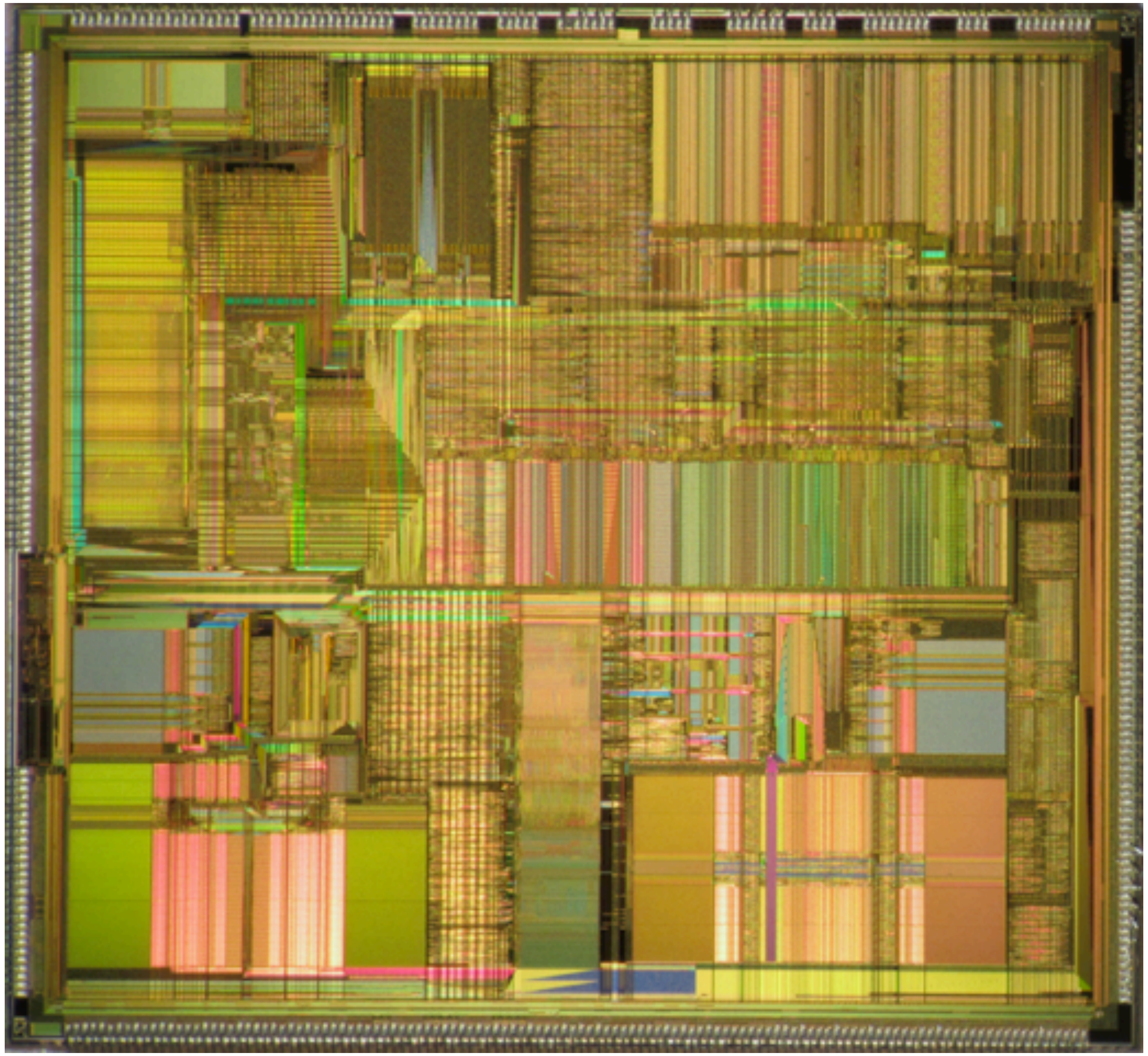
http://www.newyorker.com/magazine/2015/03/30/thinking-sideways

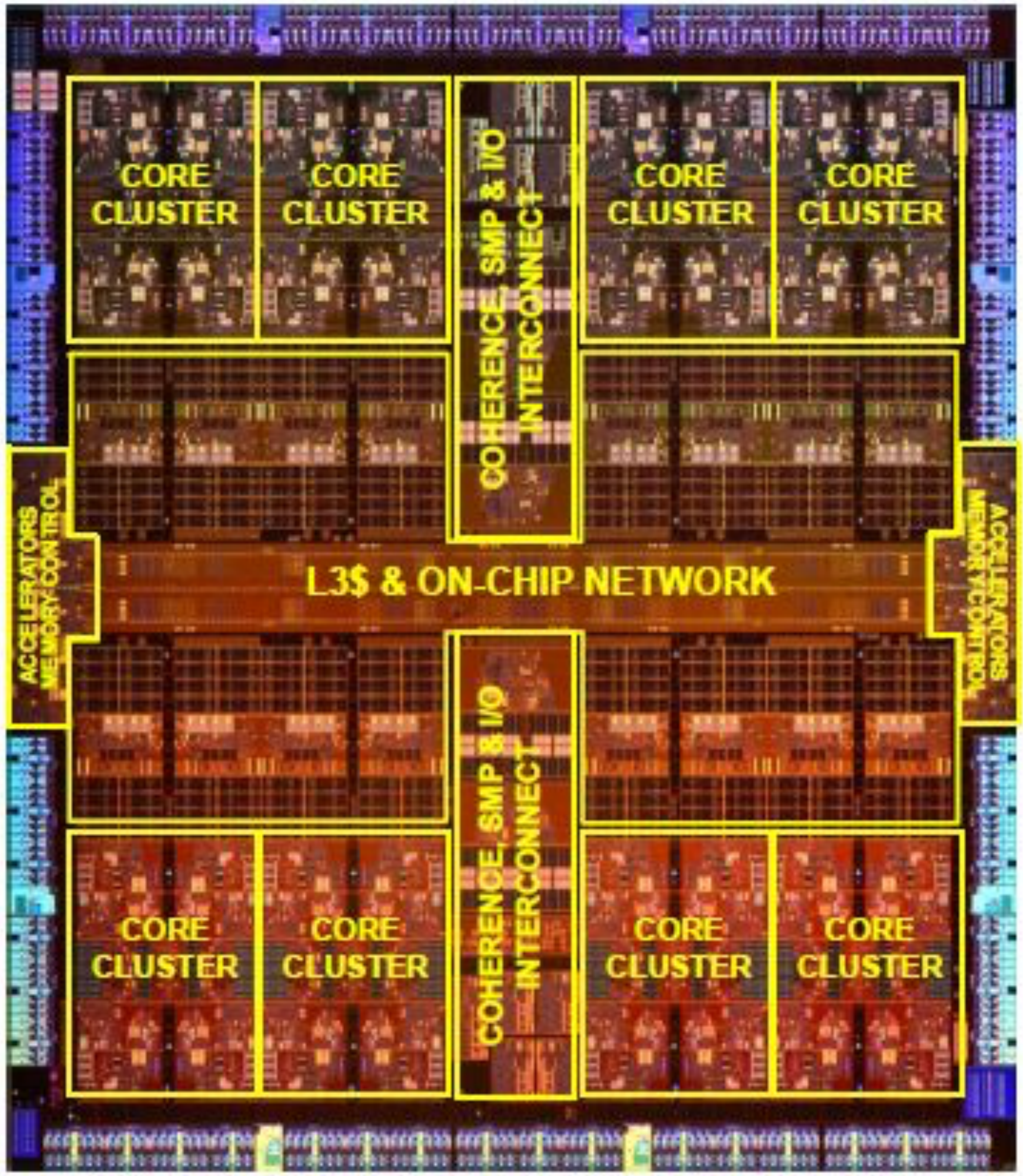


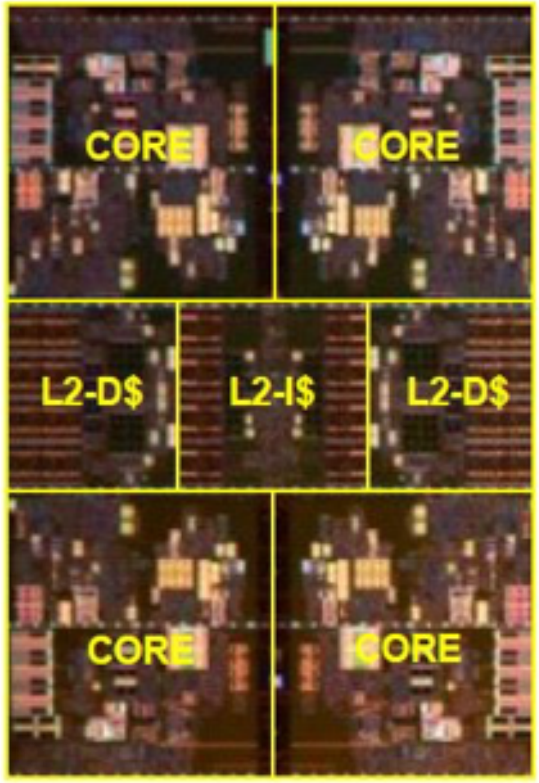
Star7: Under the Cover

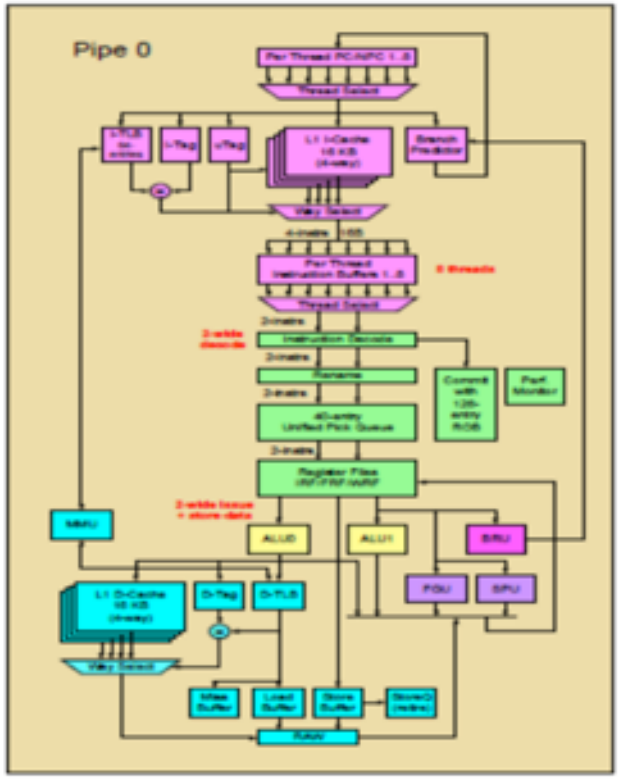
- MicroSPARC chipset
- 200Kbps RF link
- Sharp TV screen
- Touchscreen
- PCMCIA
- 4Meg RAM
- Flexible board
- IR link






CORE CLUSTER
CORE CLUSTER
COHERENCE, SMP & I/O INTERCONNECT
CORE CLUSTER
CORE CLUSTER
ACCELERATORS MEMORY CONTROL
L3$ & ON-CHIP NETWORK
ACCELERATORS MEMORY CONTROL
CORE CLUSTER
CORE CLUSTER
COHERENCE, SMP & I/O INTERCONNECT
CORE CLUSTER
CORE CLUSTER

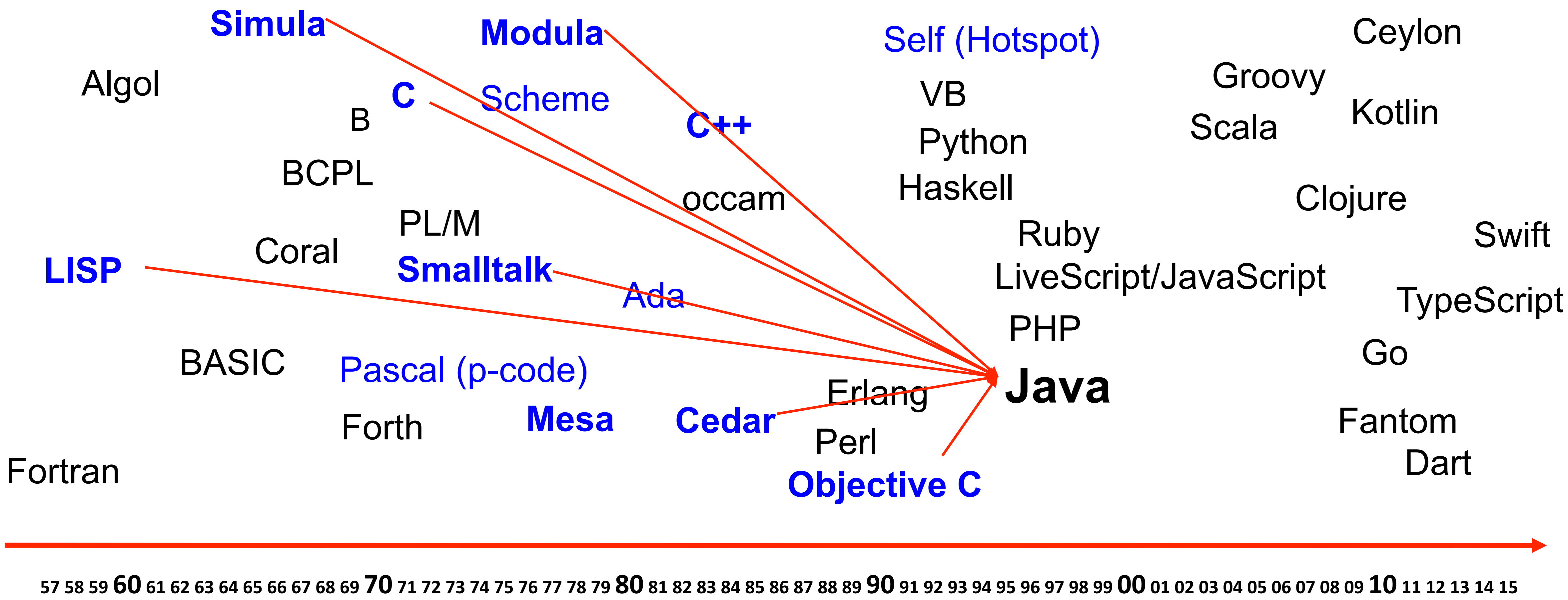




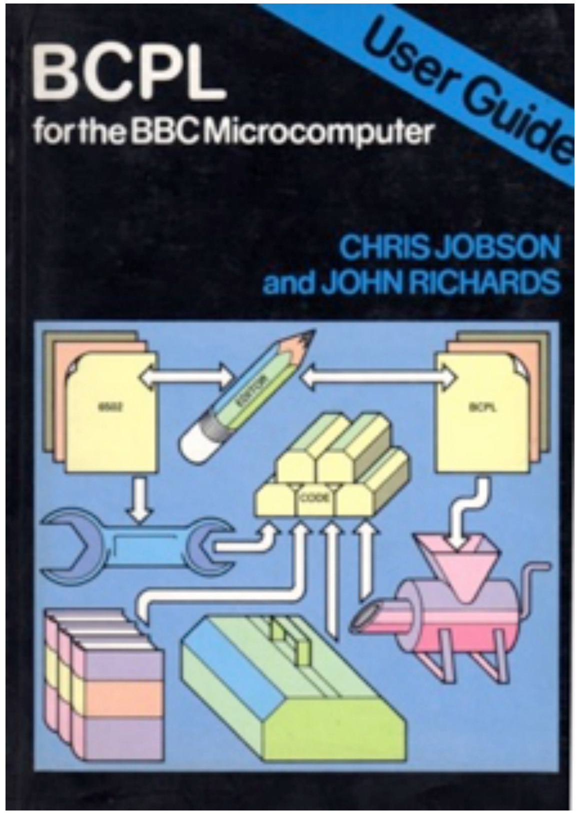
- Dynamically Threaded, 1-8 threads / core
- Concurrent Memory Migration, VA Masking
- Dual Issue OOO execution unit
- 64 Mb L3
- 256K L2 per core cluster
- Crypto acceleration
- …

(some) Influences on Java

(née GreenTalk/Oak)



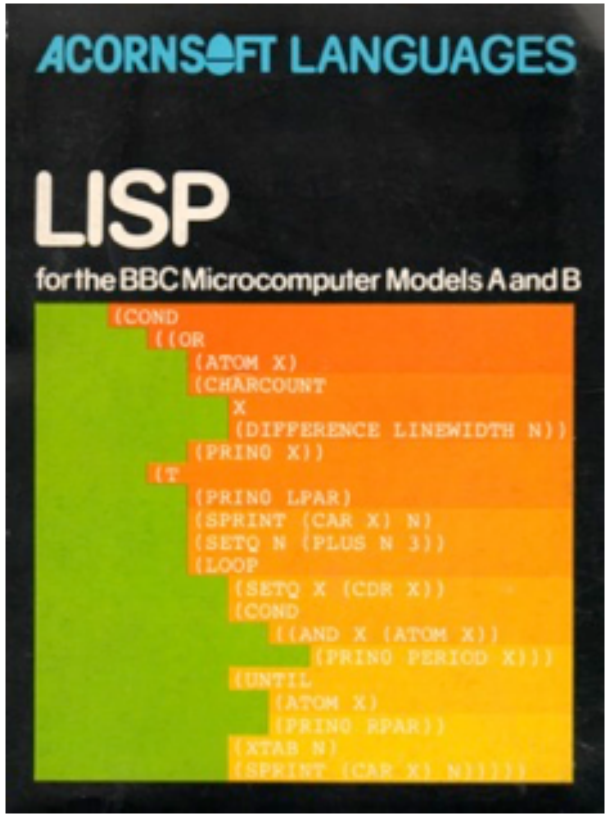
http://en.wikipedia.org/wiki/Timeline_of_programming_languages
http://hopl.murdoch.edu.au/


BCPL

for the BBC Microcomputer

User Guide

CHRIS JOBSON and JOHN RICHARDS

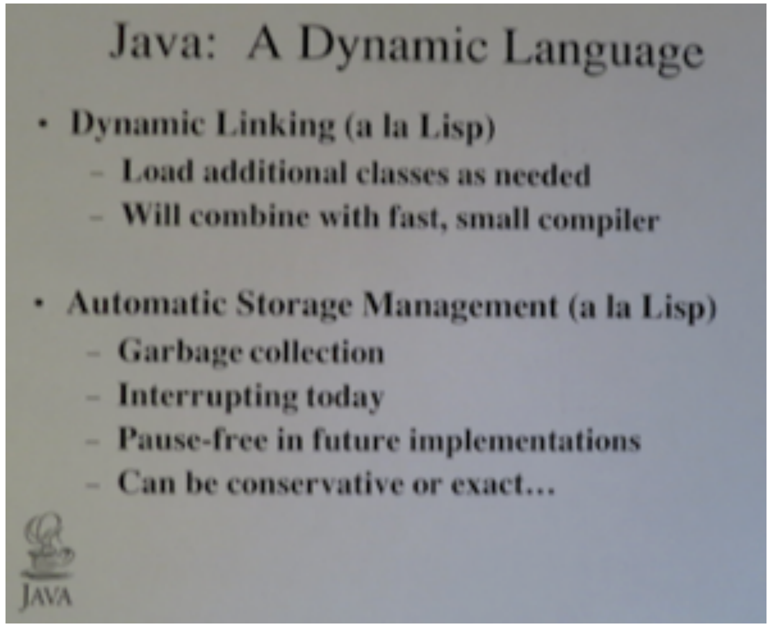

ACORNSOFT LANGUAGES

LISP

for the BBC Microcomputer Models A and B

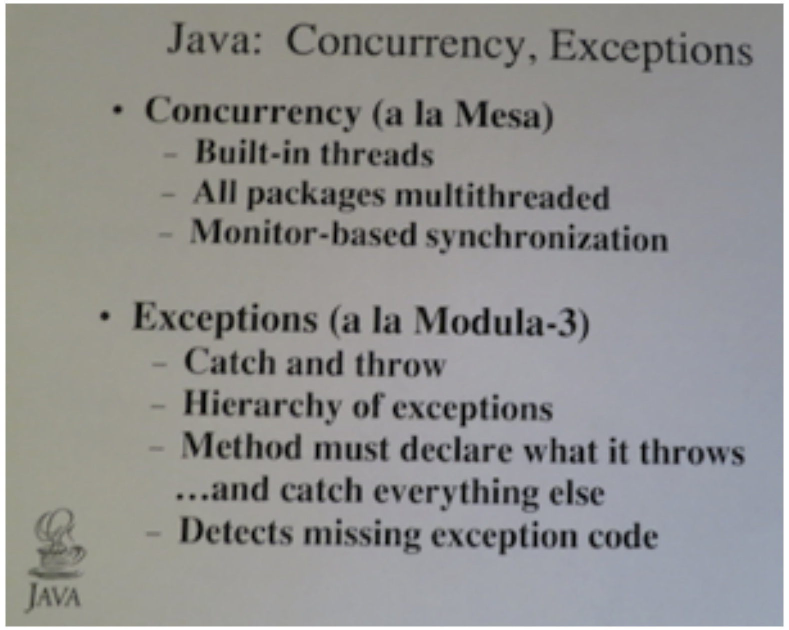
```
(COND
 ((OR
   (ATOM X)
   (CHARCOUNT
     X
     (DIFFERENCE LINEWIDTH N)))
   (PRINO X))
 (T
   (PRINO LPAR)
   (SPRINT (CAR X) N)
   (SETQ N (PLUS N 3))
   (LOOP
     (SETQ X (CDR X))
     (COND
       ((AND X (ATOM X))
         (PRINO PERIOD X)))
     (UNTIL
       (ATOM X)
       (PRINO RPAR))
     (XTAB N)
     (SPRINT (CAR X) N)))))
```

(some) Influences on Java
(née GreenTalk/Oak)


Java: A Dynamic Language

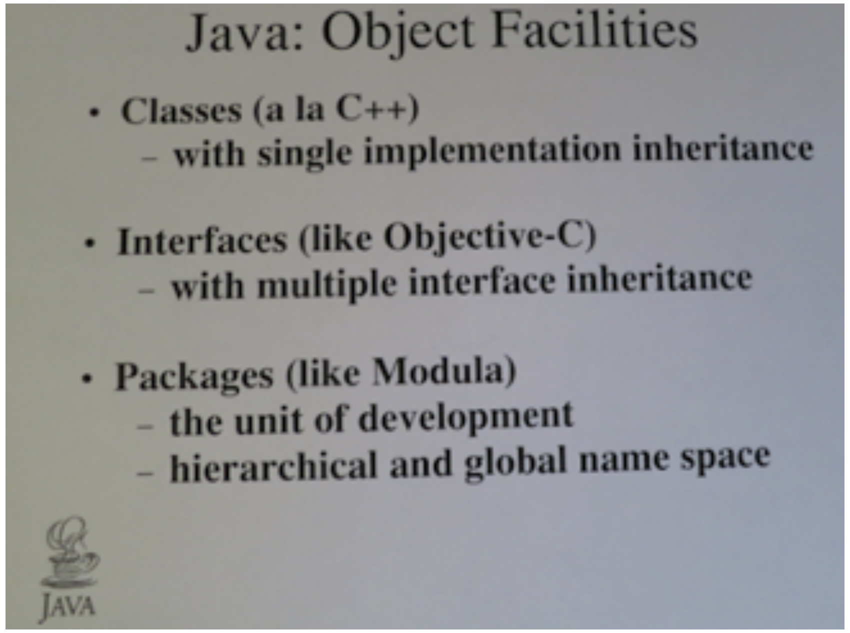
- Dynamic Linking (a la Lisp)
  - Load additional classes as needed
  - Will combine with fast, small compiler
- Automatic Storage Management (a la Lisp)
  - Garbage collection
  - Interrupting today
  - Pause-free in future implementations
  - Can be conservative or exact…

JAVA


Java: Concurrency, Exceptions

- Concurrency (a la Mesa)
  - Built-in threads
  - All packages multithreaded
  - Monitor-based synchronization
- Exceptions (a la Modula-3)
  - Catch and throw
  - Hierarchy of exceptions
  - Method must declare what it throws …and catch everything else
  - Detects missing exception code

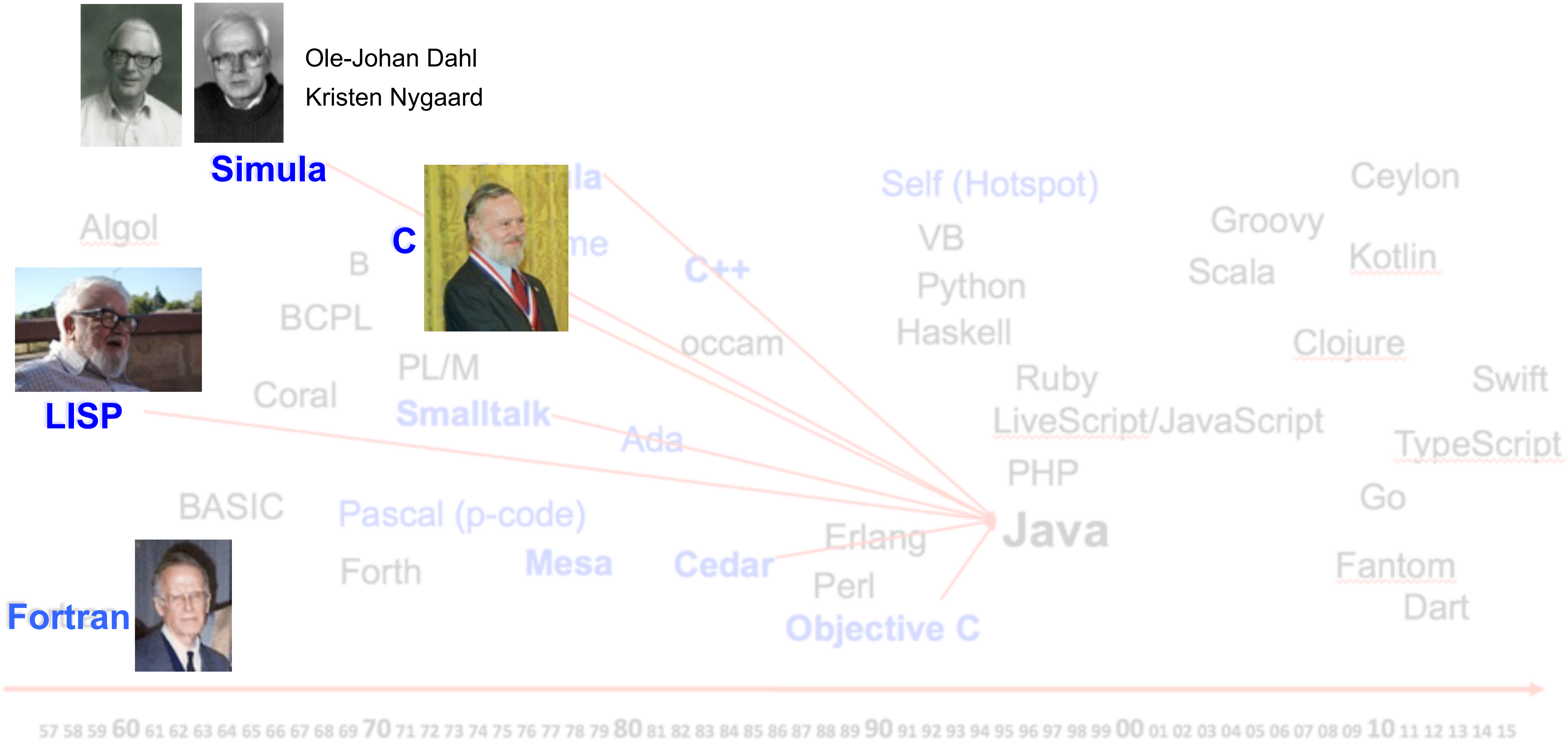
JAVA


Java: Object Facilities

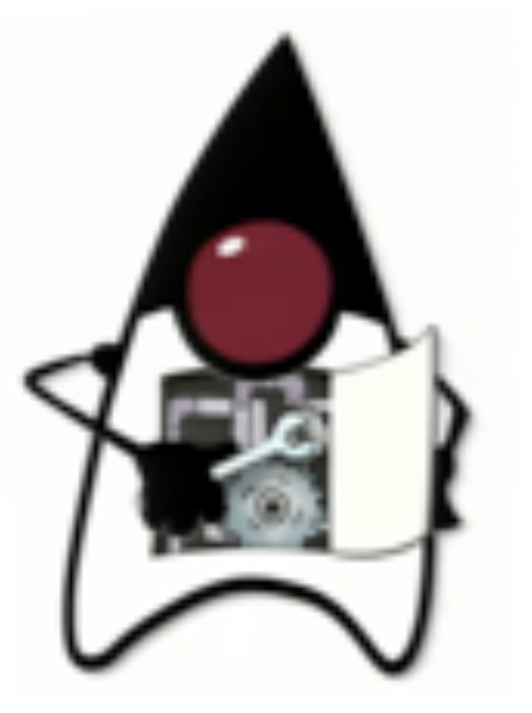
- Classes (a la C++)
  - with single implementation inheritance
- Interfaces (like Objective-C)
  - with multiple interface inheritance
- Packages (like Modula)
  - the unit of development
  - hierarchical and global name space

JAVA

Ole-Johan Dahl
Kristen Nygaard



The Ghost in the (Java Virtual) Machine



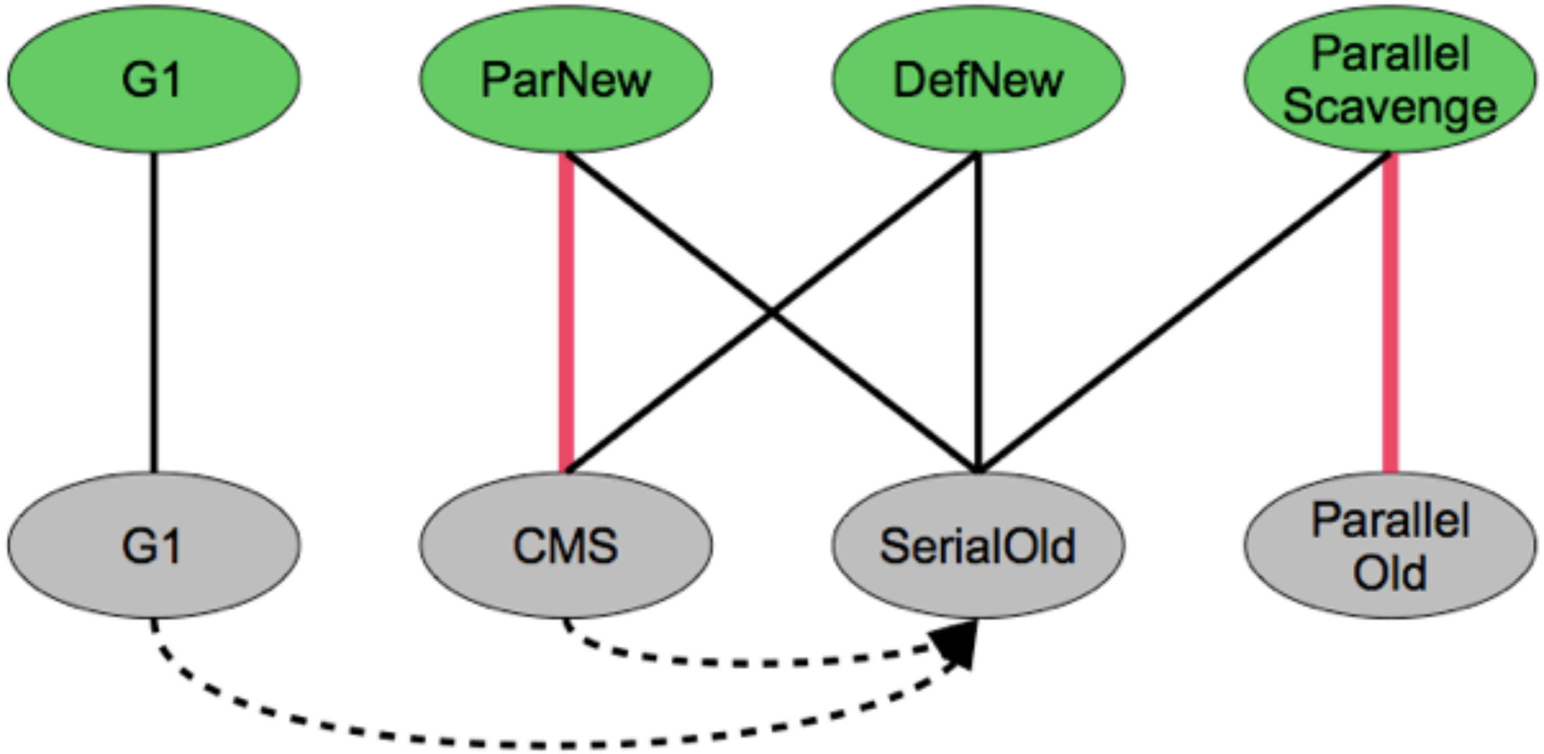
(some) JVMs & GC Collectors

• Oracle		
– HotSpot	Serial[Old], Parallel[Old],CMS	~~RTGC~~, G1 Shenendoah (OpenJDK JEP 189)
– JRockit	MSC gen/par/com	JRockit Real Time
• IBM		
– J9	thruput, avgpause, gencon	Balanced, Metronome
• Azul		
– Zing		C4 (née GPGC)

- Jikes (IBM), Maxine (Sun), Squawk (Sun, JavaME-SunSpots), Monty, KVM…

https://en.wikipedia.org/wiki/List_of_Java_virtual_machines

Hotspot Collectors



-XX:+UseSerialGC
-XX:+UseParallelGC
-XX:+UseParallelOldGC
-XX:+UseConcMarkSweepGC
-XX:+UseParNewGC
-XX:+UseG1GC

Self

STRONGTALK

Smalltalk... with a need for speed

http://www.strongtalk.org/history.html

mark-sweep garbage collection
[McCarthy, 1960]

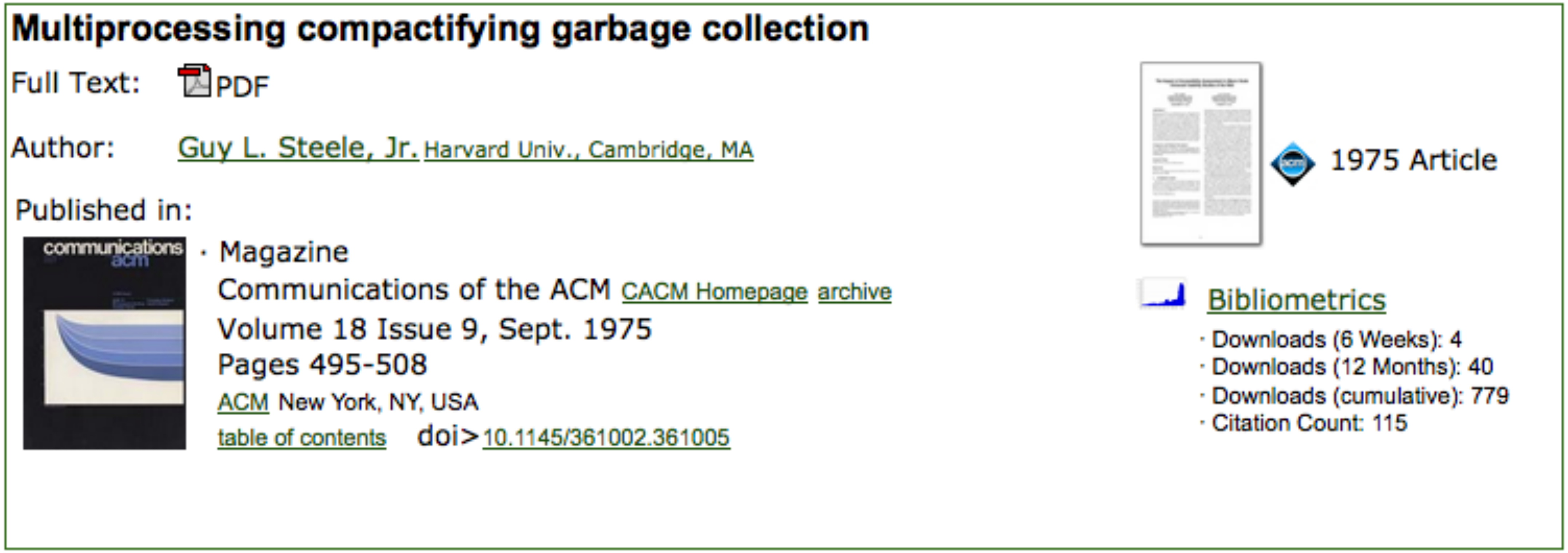
O-code
(BCPL,
Richards)

UCSD PASCAL

UCSD Jacobs | School of Engineering

P-code

"Most objects die young"
David Ungar - 1984



Multiprocessing compactifying garbage collection

Full Text: PDF

Author: Guy L. Steele, Jr. Harvard Univ., Cambridge, MA

Published in:

· Magazine
Communications of the ACM CACM Homepage archive
Volume 18 Issue 9, Sept. 1975
Pages 495-508
ACM New York, NY, USA
table of contents doi>10.1145/361002.361005

1975 Article

Bibliometrics
· Downloads (6 Weeks): 4
· Downloads (12 Months): 40
· Downloads (cumulative): 779
· Citation Count: 115

A brief history of (some) Java Virtual Machines…

- VM for Oak (1992 – Gosling)
- Sun JDK : Alpha/Beta (1995), 1.0 (1996)
- JIT added in 1.1 (1997)
  Cliff Click (later worked on C2 server Compiler)
- Sun acquires Animorphic (1997)

  David Ungar, Lars Bak, Urs Hölzle… (Self)
  + David Griswold, Gilad Bracha… (Strongtalk)
- HotSpot released (1999)
  add on for 1.2, default in 1.3
- CMS
  Printezis/Detlefs (2000)
- G1
  Printezis/Flood/Heller (2004)
- RTGC / Java RTS (2009) [not active]

- Appeal Virtual Machines – Stockholm (1998)
  Marcus Hirt, Marcus Lagergren…
- Acquired by BEA (2002)
- BEA Acquired by Oracle (2008)
- Dynamic GC
- Deterministic GC (~2005) – JRRT
  Eva Andreasson
- JRockit Mission Control (~2007)

A brief history of Java Virtual Machines (contd)

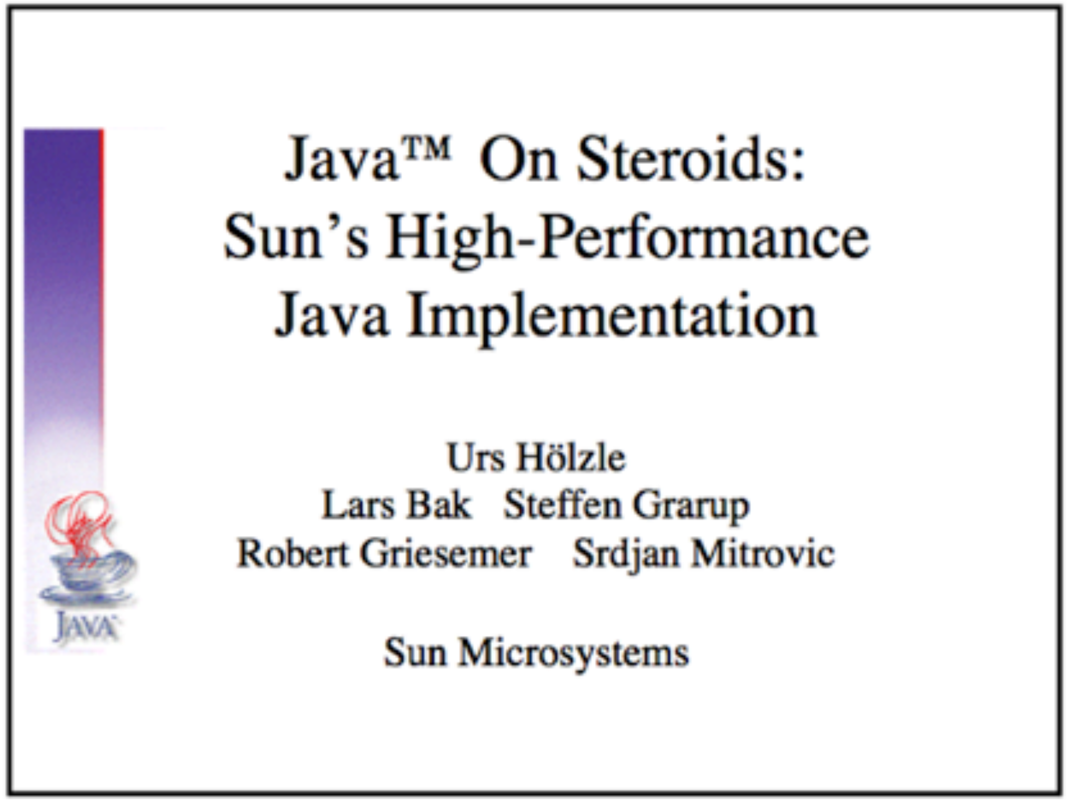
- Azul (2002)

  Sellers/Tene/Pillalamarri
- Stephen DeWitt + ex Sun (2004)
- Vega (2004)
- Zing 5 (2010) software only

  C4 Collector

- IBM Hursley - Java Technology Center (1996)
- J9 - IBM Ottowa Labs (OTI) 1.4 (2003)

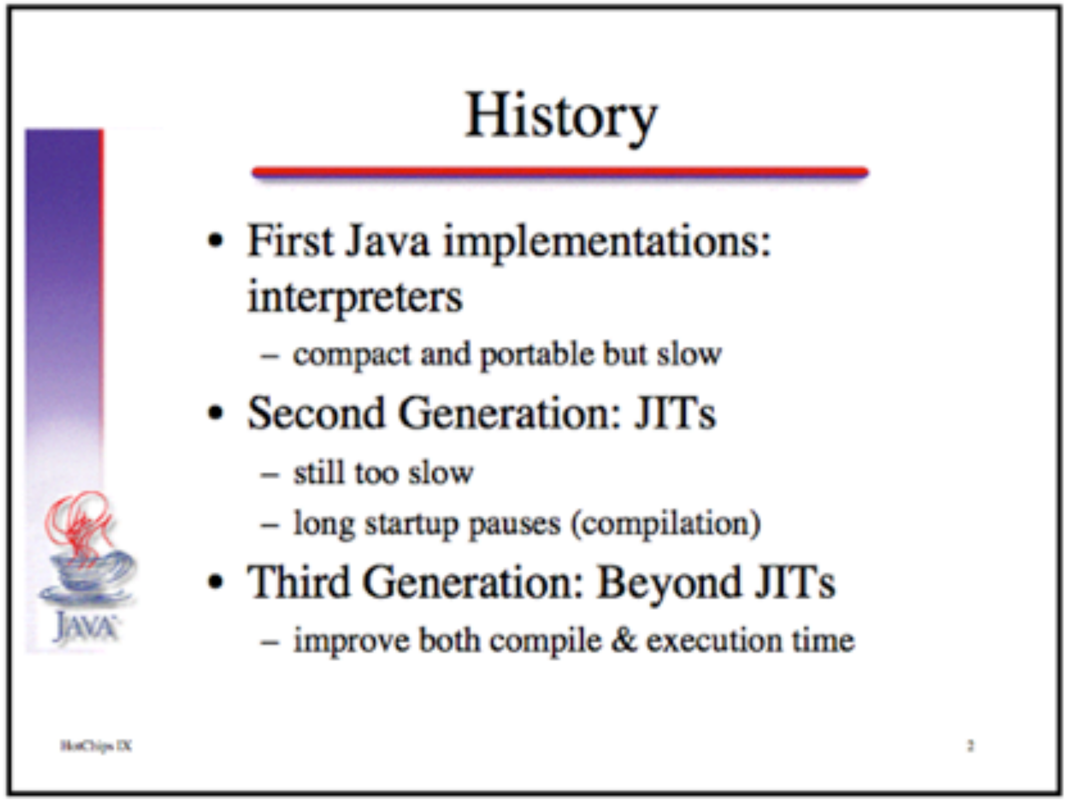
  IBM Toronto Labs
- Metronome RT collector (2003)
- Balanced Collector (2011)



Java™ On Steroids:
Sun's High-Performance
Java Implementation

Urs Hölzle
Lars Bak Steffen Grarup
Robert Griesemer Srdjan Mitrovic

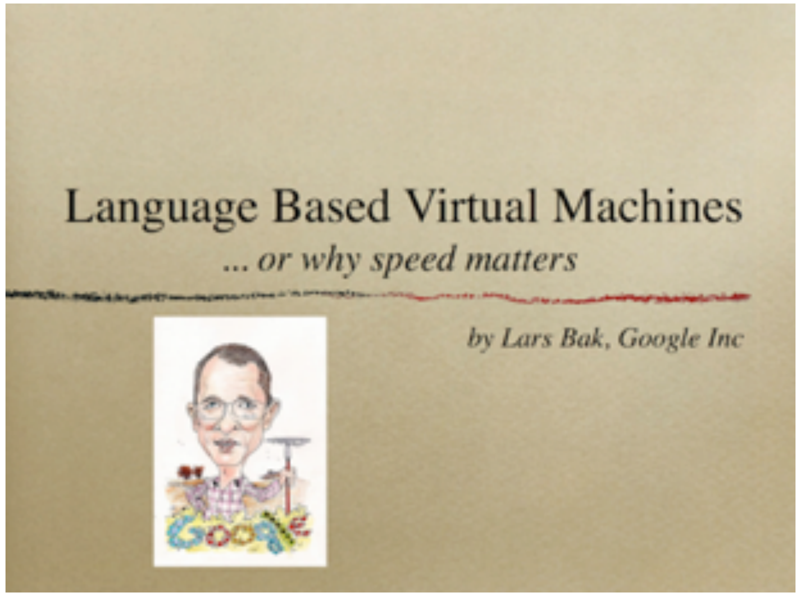
Sun Microsystems



History

- First Java implementations: interpreters
  - compact and portable but slow
- Second Generation: JITs
  - still too slow
  - long startup pauses (compilation)
- Third Generation: Beyond JITs
  - improve both compile & execution time

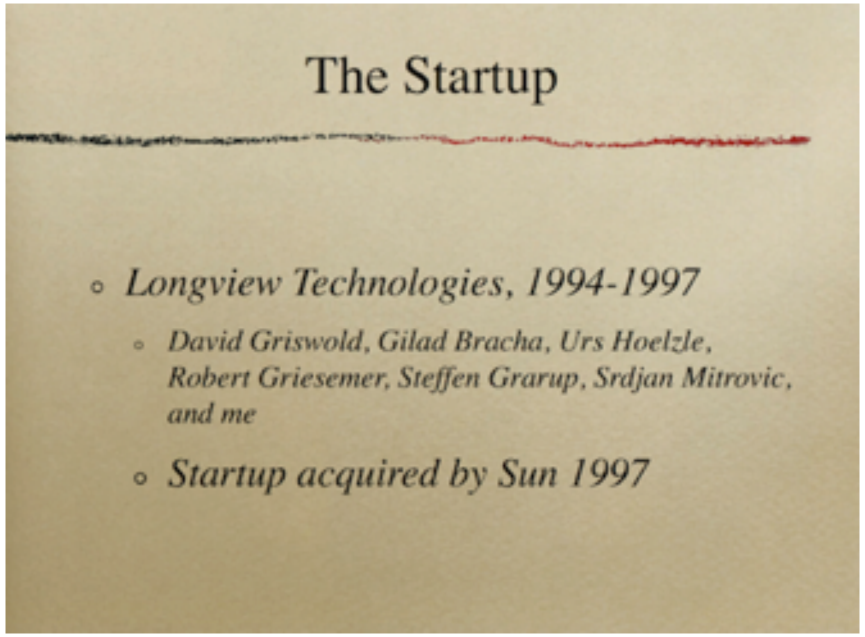
HotChips IX 2



Language Based Virtual Machines

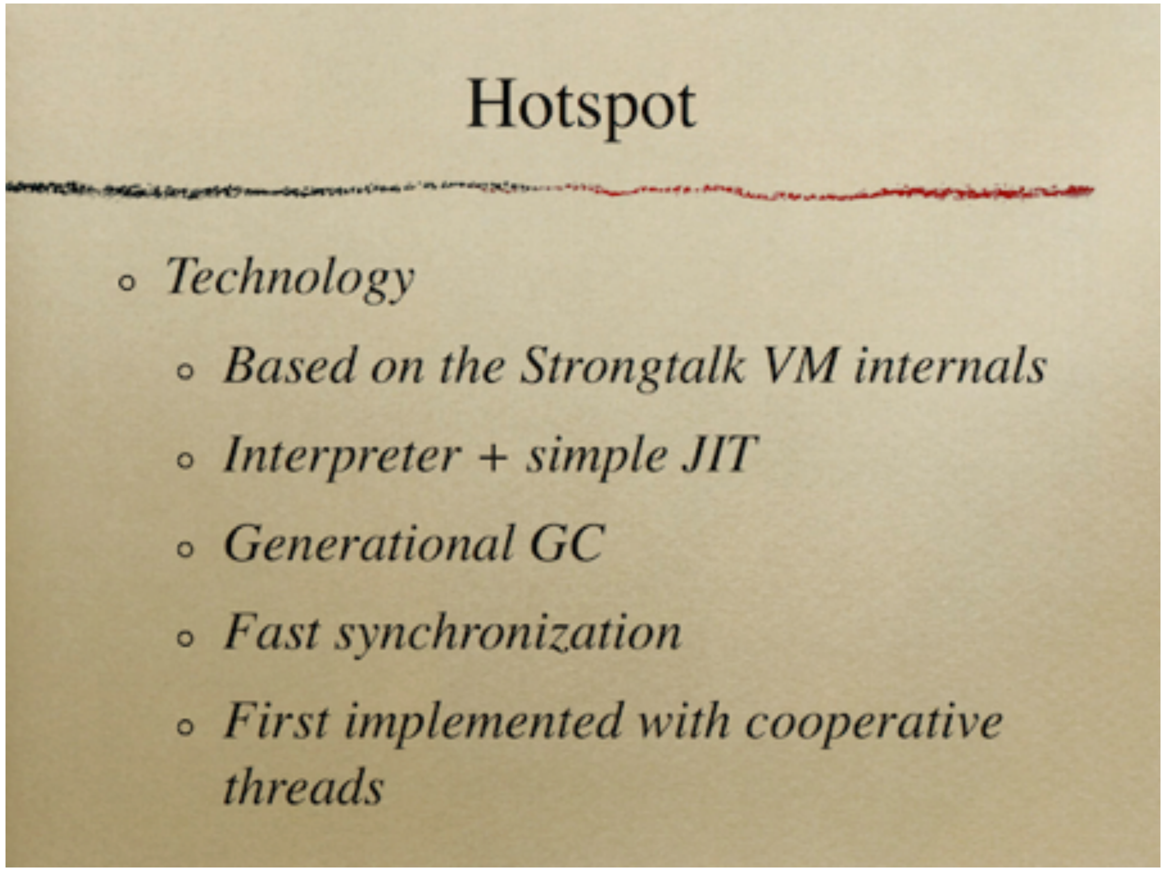
... or why speed matters

by Lars Bak, Google Inc



The Startup

- *Longview Technologies, 1994-1997*
  - *David Griswold, Gilad Bracha, Urs Hoelzle, Robert Griesemer, Steffen Grarup, Srdjan Mitrovic, and me*
  - *Startup acquired by Sun 1997*



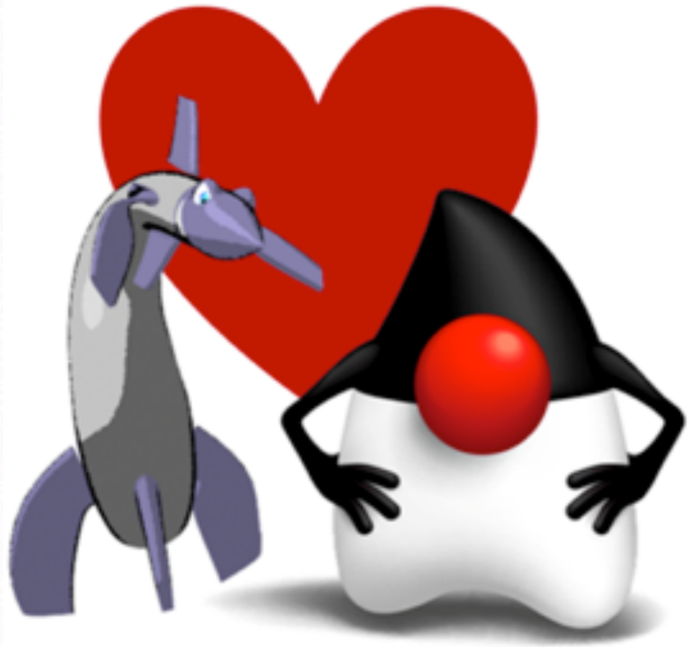
Hotspot

- *Technology*
  - *Based on the Strongtalk VM internals*
  - *Interpreter + simple JIT*
  - *Generational GC*
  - *Fast synchronization*
  - *First implemented with cooperative threads*

Design Rationales in the JRockit JVM

Marcus Lagergren

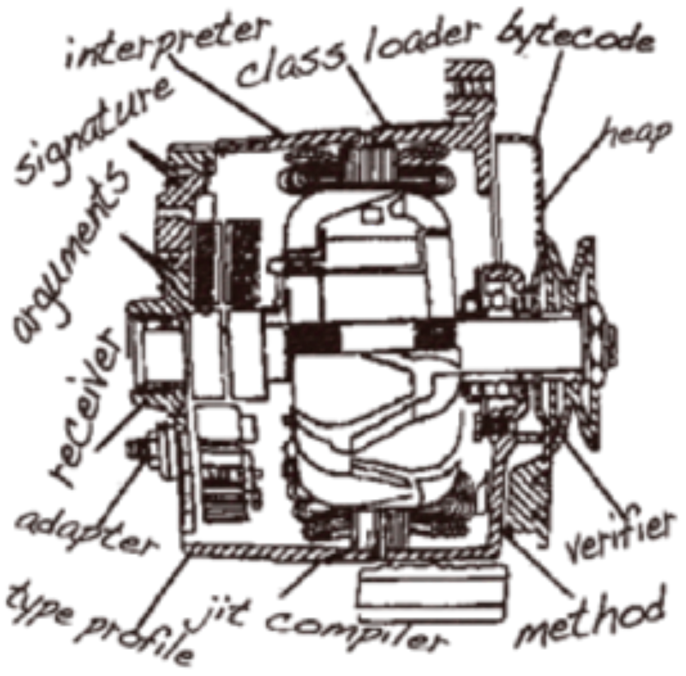
Java Language team, Oracle



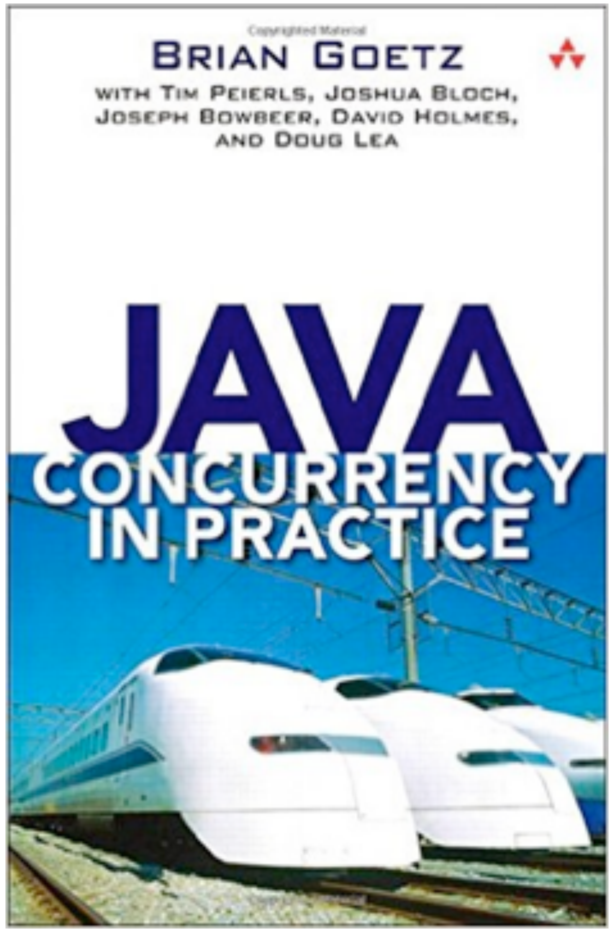
HotSpot / G1

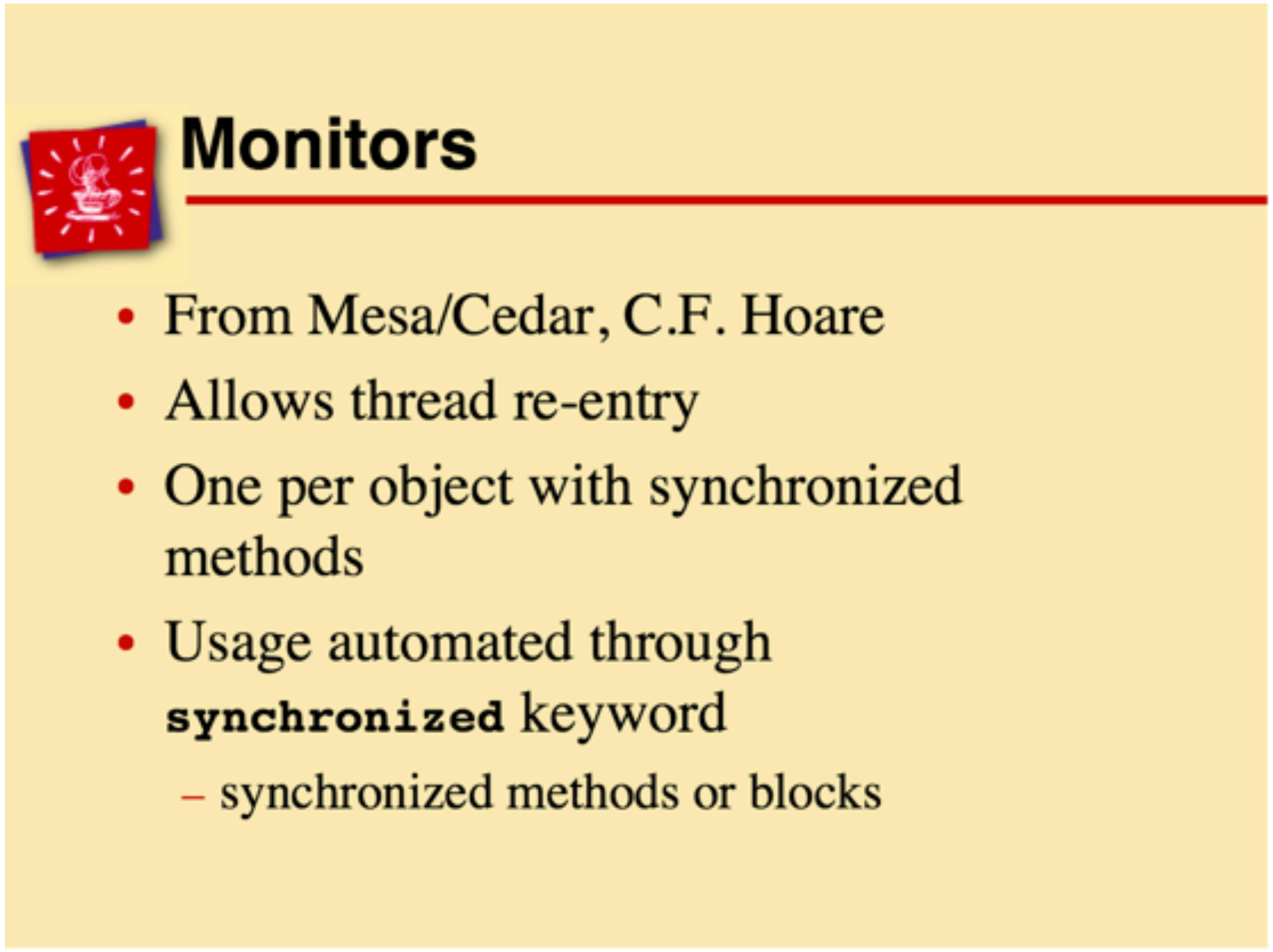
- Permgen removal
- Density
- JIT Compilers (C1/C2 Tiered Compilation)
- GC improvements / G1 / Rationalisation
- Ergonomics
- Instrumentation / Tuning / Performance

- Cloud – Multi-Tenancy. Isolation / Resource Management.
- AOT Compilation (https://www.youtube.com/watch?v=Xybzyv8qbOc)
- Deterministic GC / Low Latency



Concurrency / Threads / Java Memory Model


Copyrighted Material
BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA
CONCURRENCY IN PRACTICE

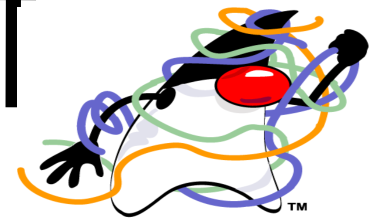


Monitors

- From Mesa/Cedar, C.F. Hoare
- Allows thread re-entry
- One per object with synchronized methods
- Usage automated through **`synchronized`** keyword
  – synchronized methods or blocks

Concurrency in Java

java.lang.Thread



1.4

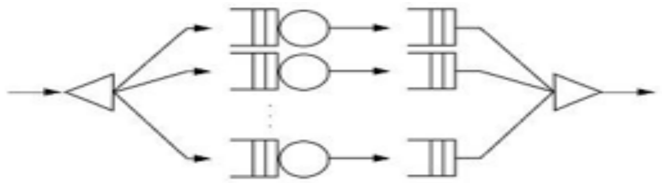
java.util.concurrent
(jsr166)

5.0

Phasers, etc
(jsr166)

6

Fork/Join Framework
(jsr166y)

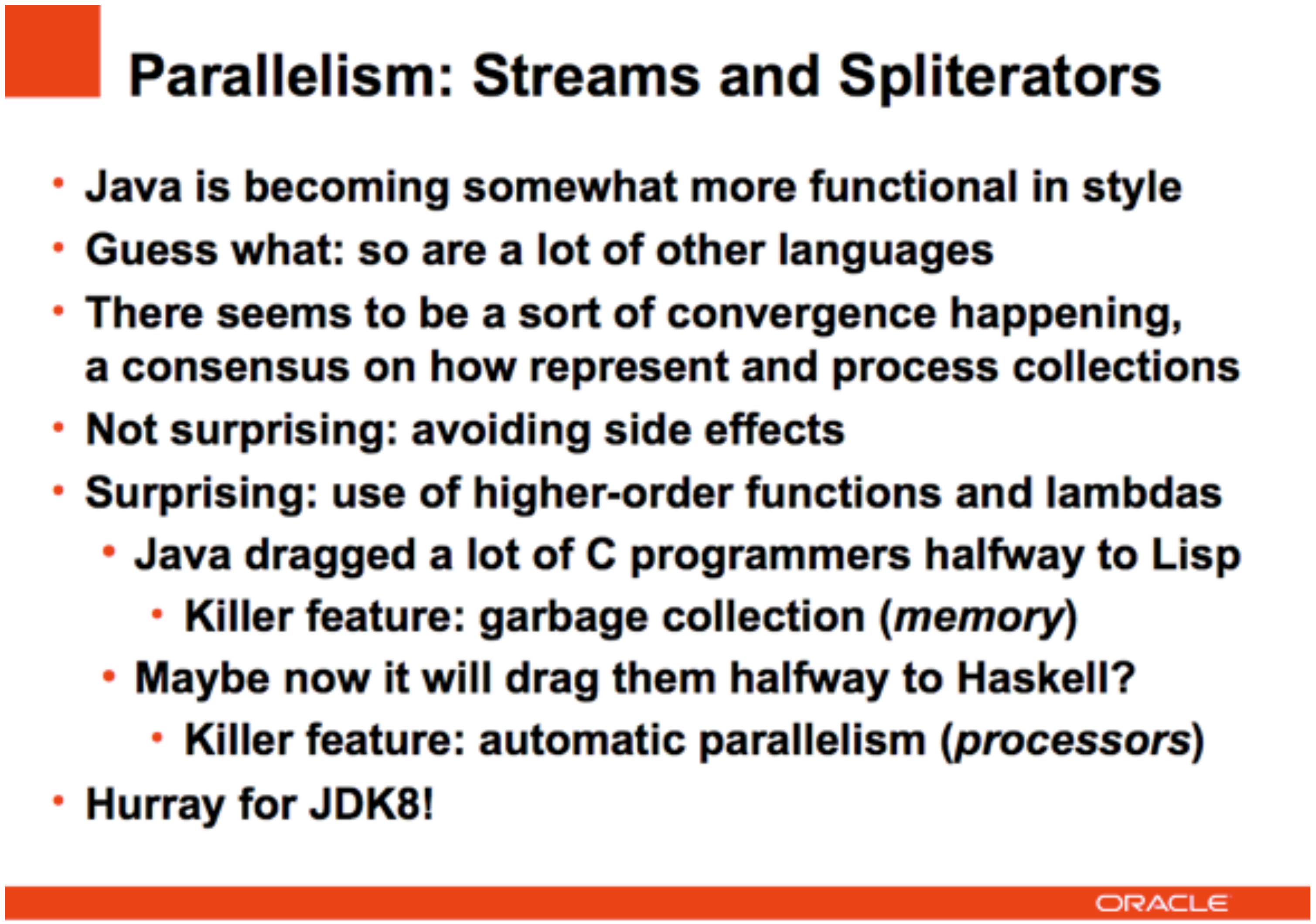


7

Project Lambda

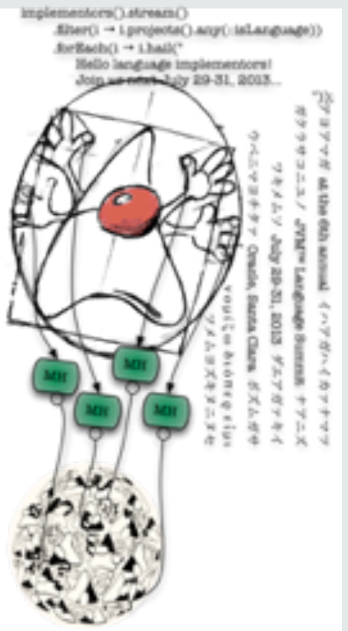
8

2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014



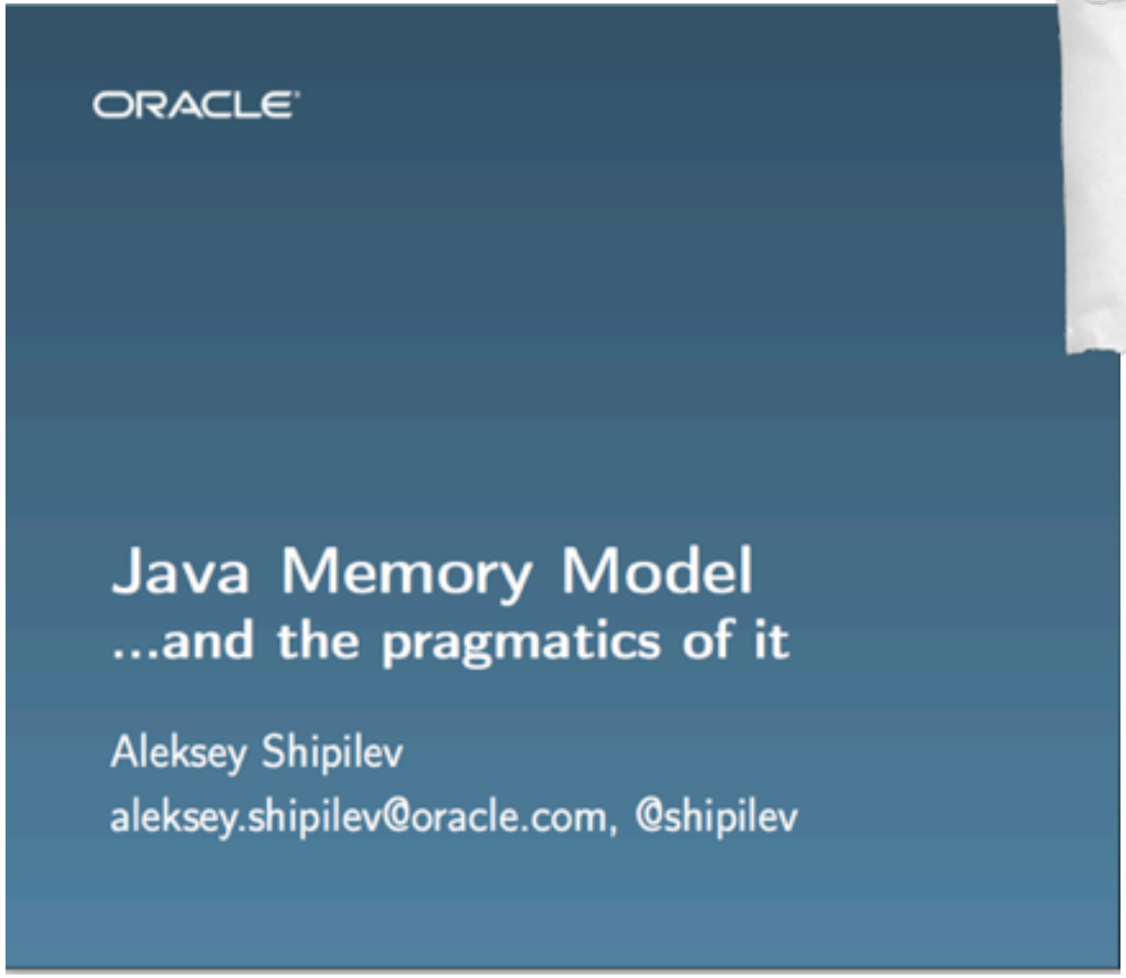
Parallelism: Streams and Spliterators

- Java is becoming somewhat more functional in style
- Guess what: so are a lot of other languages
- There seems to be a sort of convergence happening, a consensus on how represent and process collections
- Not surprising: avoiding side effects
- Surprising: use of higher-order functions and lambdas
  - Java dragged a lot of C programmers halfway to Lisp
    - Killer feature: garbage collection (*memory*)
  - Maybe now it will drag them halfway to Haskell?
    - Killer feature: automatic parallelism (*processors*)
- Hurray for JDK8!



```
implementors().stream()
    .filter(i → i.projects().any(::isLanguage))
    .forEach(i → i.hail("
        Hello language implementors!
        Join us next July 29-31, 2013...
```

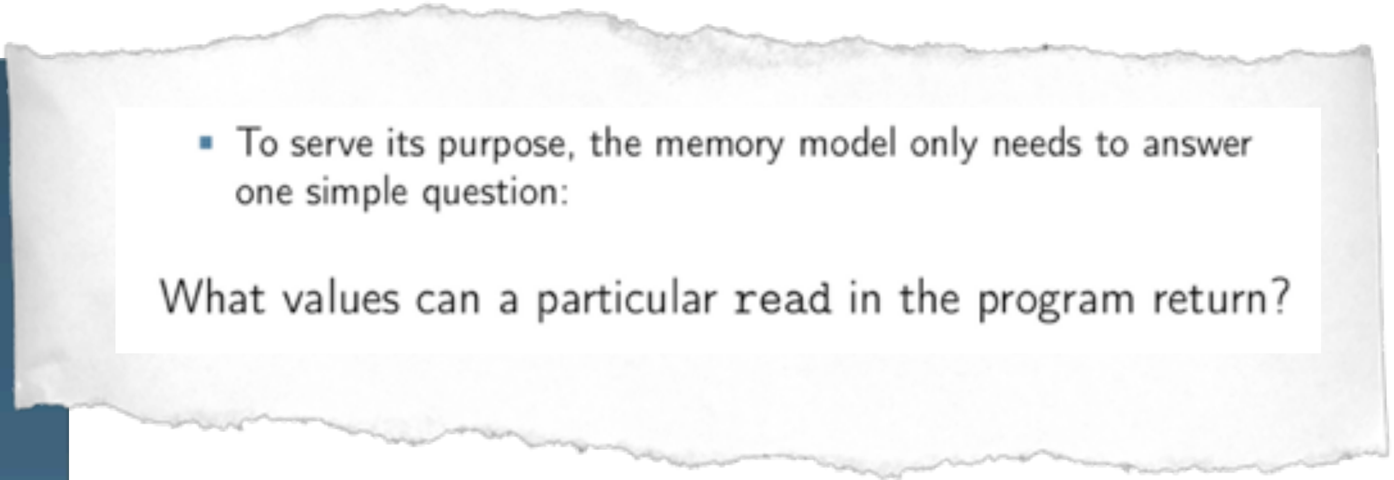
Guy Steele
JVM Language Summit 2013



ORACLE

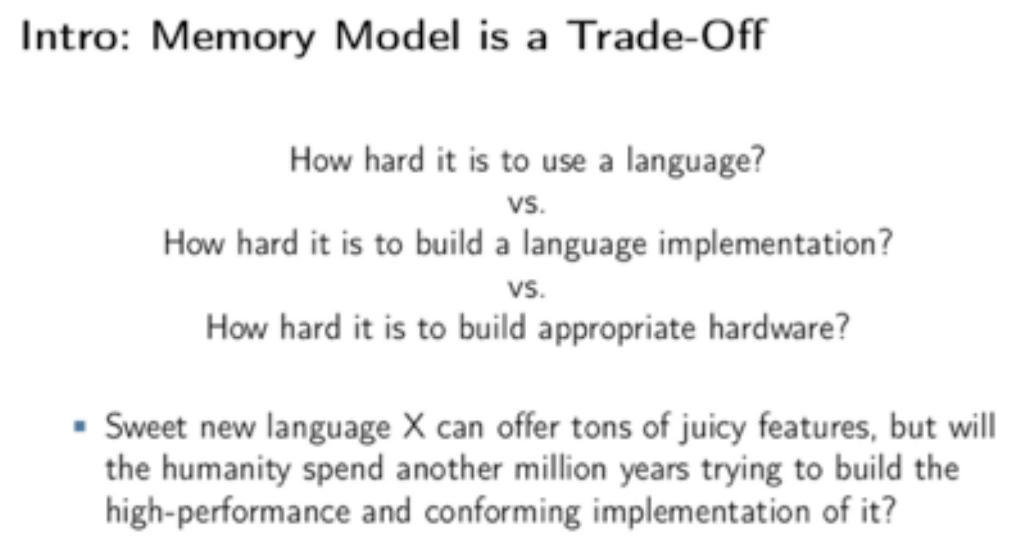
Java Memory Model
...and the pragmatics of it

Aleksey Shipilev
aleksey.shipilev@oracle.com, @shipilev



- To serve its purpose, the memory model only needs to answer one simple question:

What values can a particular `read` in the program return?



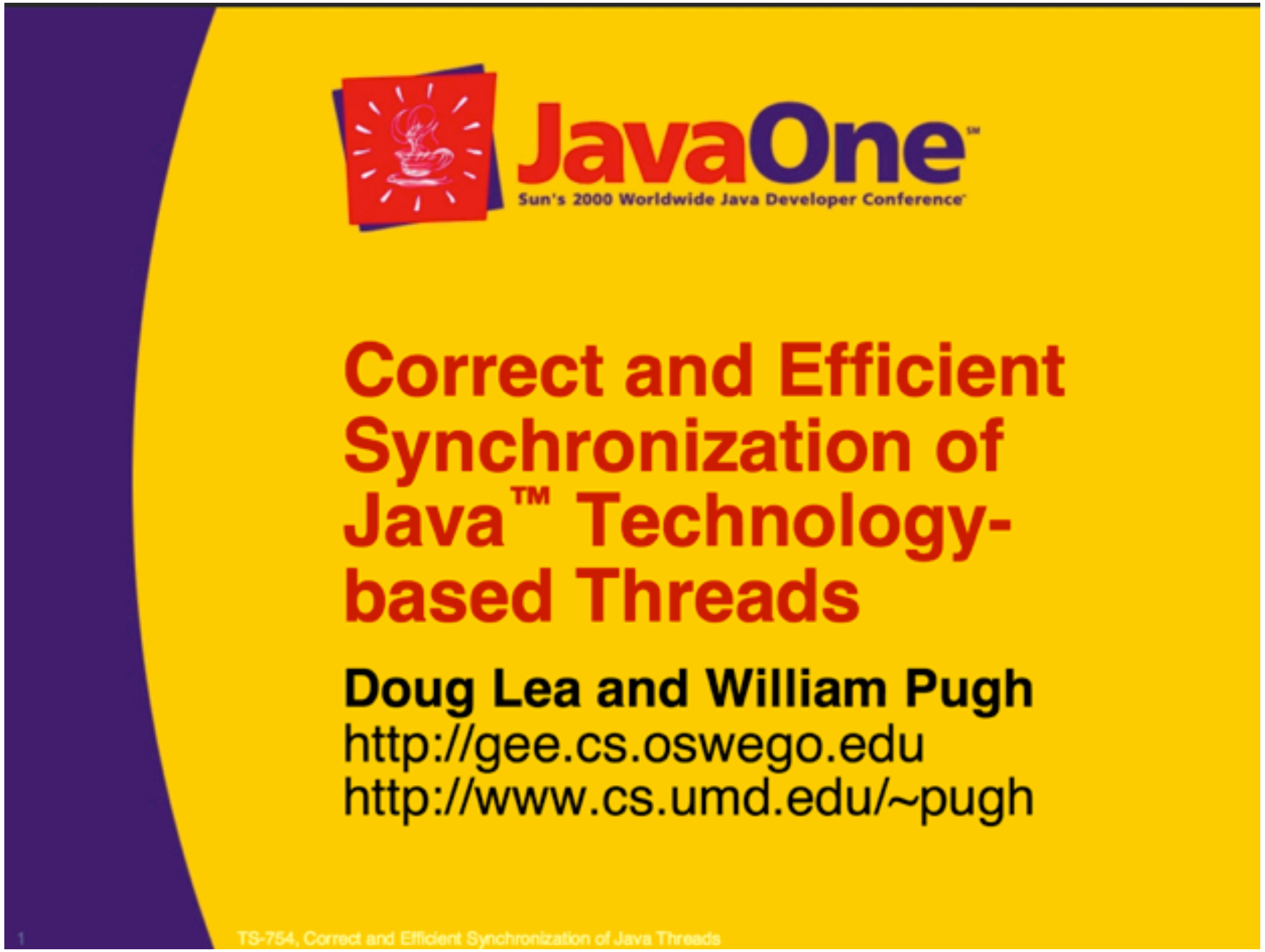
Intro: Memory Model is a Trade-Off

How hard it is to use a language?
vs.
How hard it is to build a language implementation?
vs.
How hard it is to build appropriate hardware?

- Sweet new language X can offer tons of juicy features, but will the humanity spend another million years trying to build the high-performance and conforming implementation of it?

http://shipilev.net/talks/narnia-2555-jmm-pragmatics-en.pdf

https://www.youtube.com/watch?v=TxqsKzxyySo (VJUG)



JavaOne

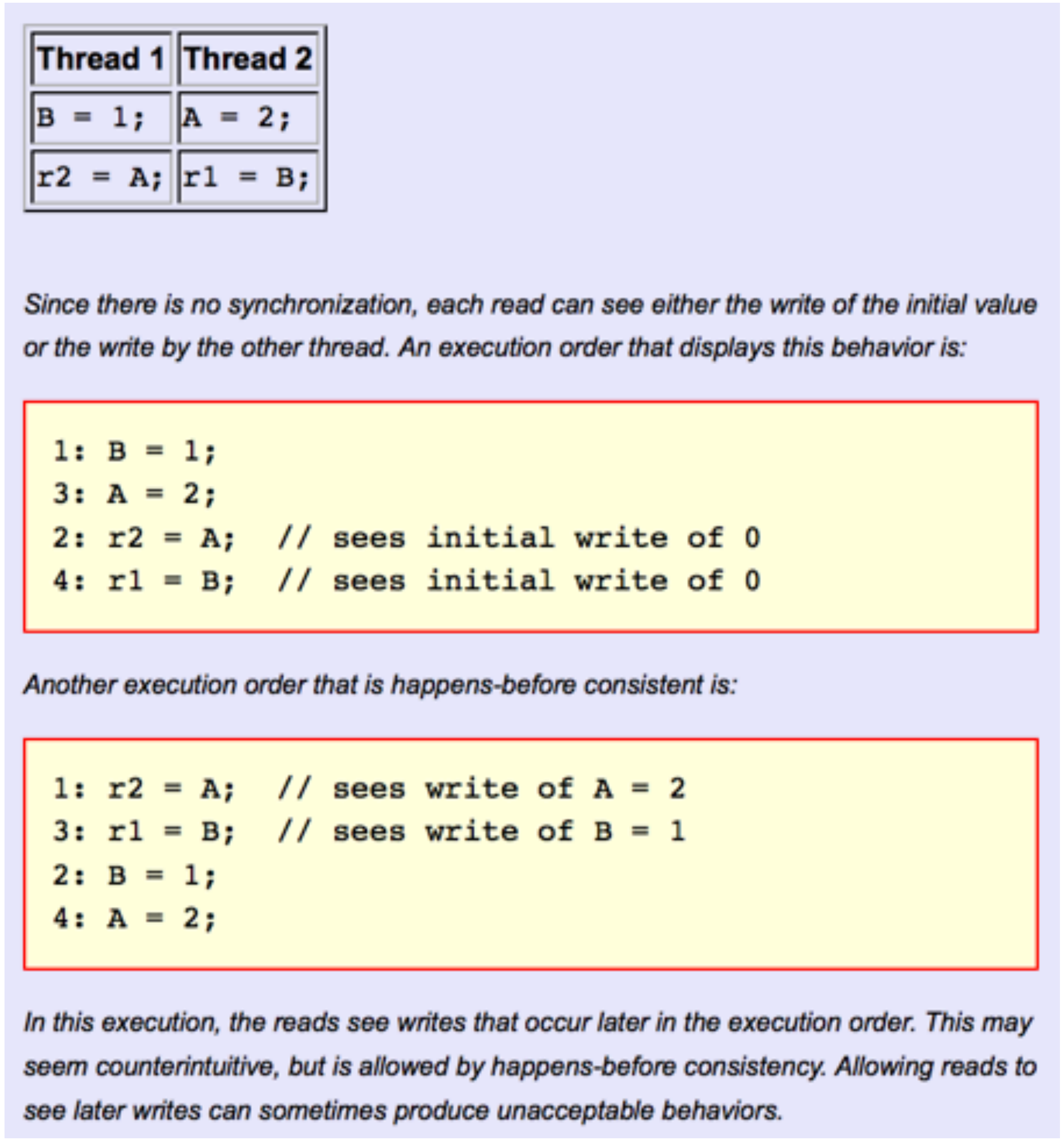
Sun's 2000 Worldwide Java Developer Conference

Correct and Efficient Synchronization of Java™ Technology-based Threads

Doug Lea and William Pugh
http://gee.cs.oswego.edu
http://www.cs.umd.edu/~pugh

1 TS-754, Correct and Efficient Synchronization of Java Threads

http://www.cs.umd.edu/~pugh/java/memoryModel



Thread 1	Thread 2
`B = 1;`	`A = 2;`
`r2 = A;`	`r1 = B;`

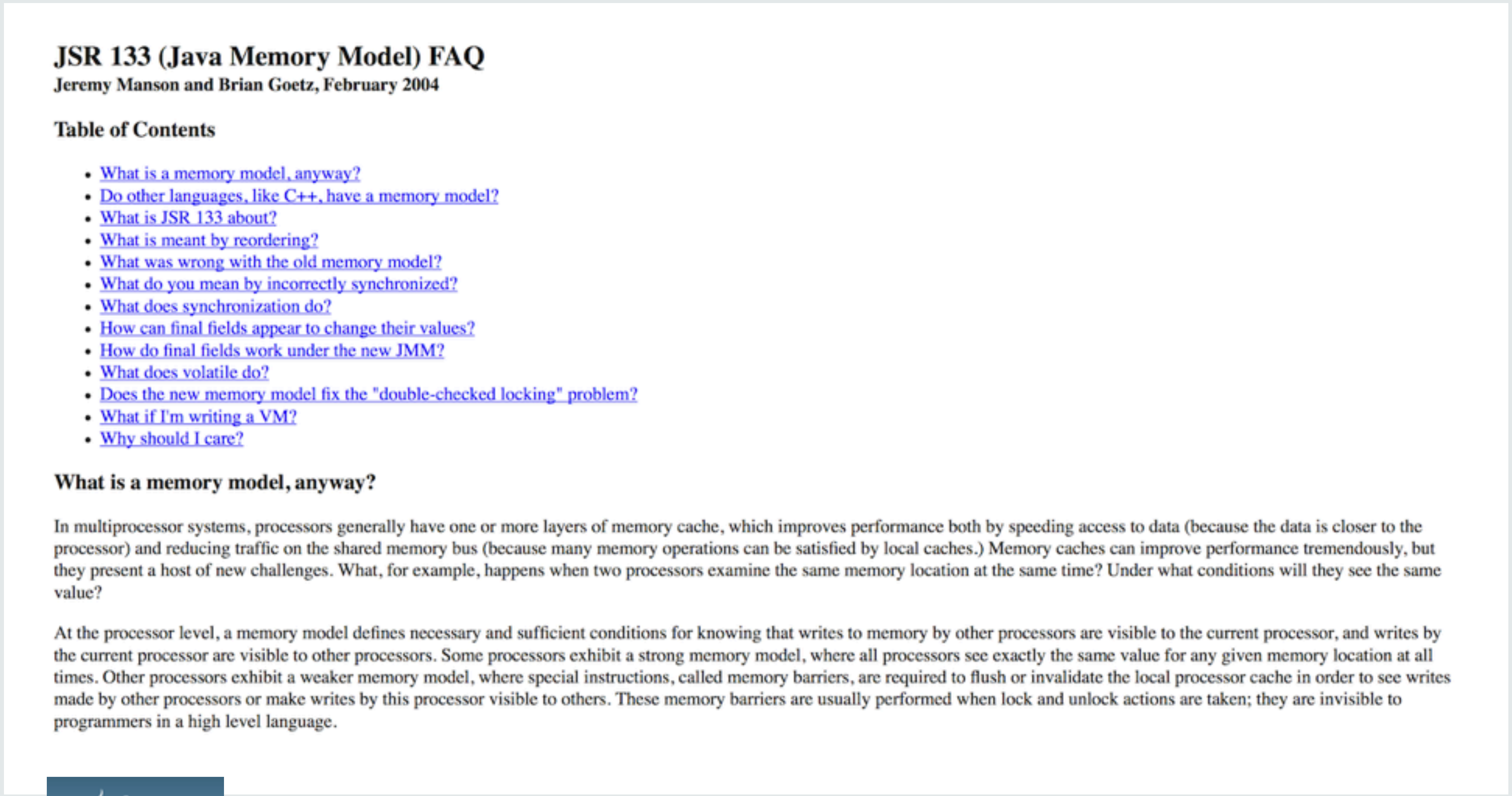
Since there is no synchronization, each read can see either the write of the initial value or the write by the other thread. An execution order that displays this behavior is:

```
1: B = 1;
3: A = 2;
2: r2 = A;  // sees initial write of 0
4: r1 = B;  // sees initial write of 0
```

Another execution order that is happens-before consistent is:

```
1: r2 = A;  // sees write of A = 2
3: r1 = B;  // sees write of B = 1
2: B = 1;
4: A = 2;
```

In this execution, the reads see writes that occur later in the execution order. This may seem counterintuitive, but is allowed by happens-before consistency. Allowing reads to see later writes can sometimes produce unacceptable behaviors.



JSR 133 (Java Memory Model) FAQ

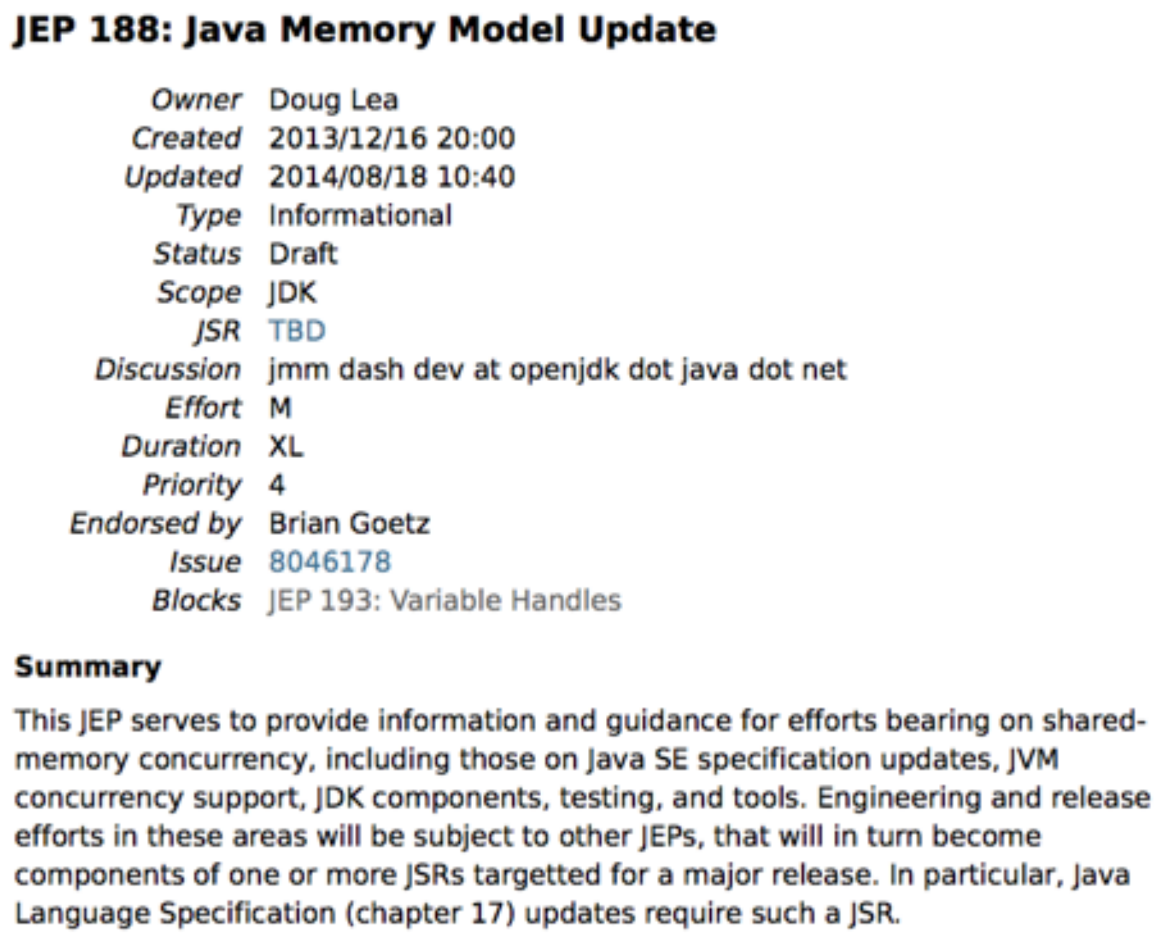
Jeremy Manson and Brian Goetz, February 2004

Table of Contents

What is a memory model, anyway?

In multiprocessor systems, processors generally have one or more layers of memory cache, which improves performance both by speeding access to data (because the data is closer to the processor) and reducing traffic on the shared memory bus (because many memory operations can be satisfied by local caches.) Memory caches can improve performance tremendously, but they present a host of new challenges. What, for example, happens when two processors examine the same memory location at the same time? Under what conditions will they see the same value?

At the processor level, a memory model defines necessary and sufficient conditions for knowing that writes to memory by other processors are visible to the current processor, and writes by the current processor are visible to other processors. Some processors exhibit a strong memory model, where all processors see exactly the same value for any given memory location at all times. Other processors exhibit a weaker memory model, where special instructions, called memory barriers, are required to flush or invalidate the local processor cache in order to see writes made by other processors or make writes by this processor visible to others. These memory barriers are usually performed when lock and unlock actions are taken; they are invisible to programmers in a high level language.

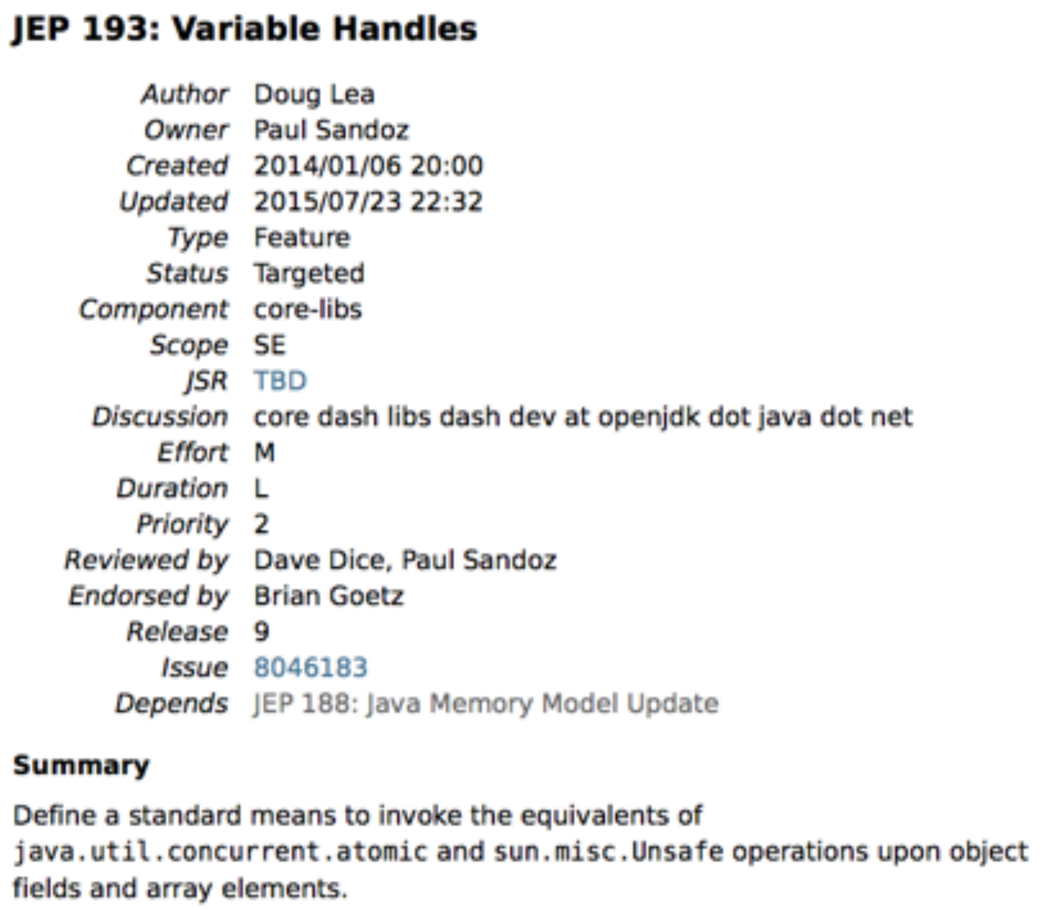


JEP 188: Java Memory Model Update

Owner	Doug Lea
Created	2013/12/16 20:00
Updated	2014/08/18 10:40
Type	Informational
Status	Draft
Scope	JDK
JSR	TBD
Discussion	jmm dash dev at openjdk dot java dot net
Effort	M
Duration	XL
Priority	4
Endorsed by	Brian Goetz
Issue	8046178
Blocks	JEP 193: Variable Handles

Summary

This JEP serves to provide information and guidance for efforts bearing on shared-memory concurrency, including those on Java SE specification updates, JVM concurrency support, JDK components, testing, and tools. Engineering and release efforts in these areas will be subject to other JEPs, that will in turn become components of one or more JSRs targetted for a major release. In particular, Java Language Specification (chapter 17) updates require such a JSR.

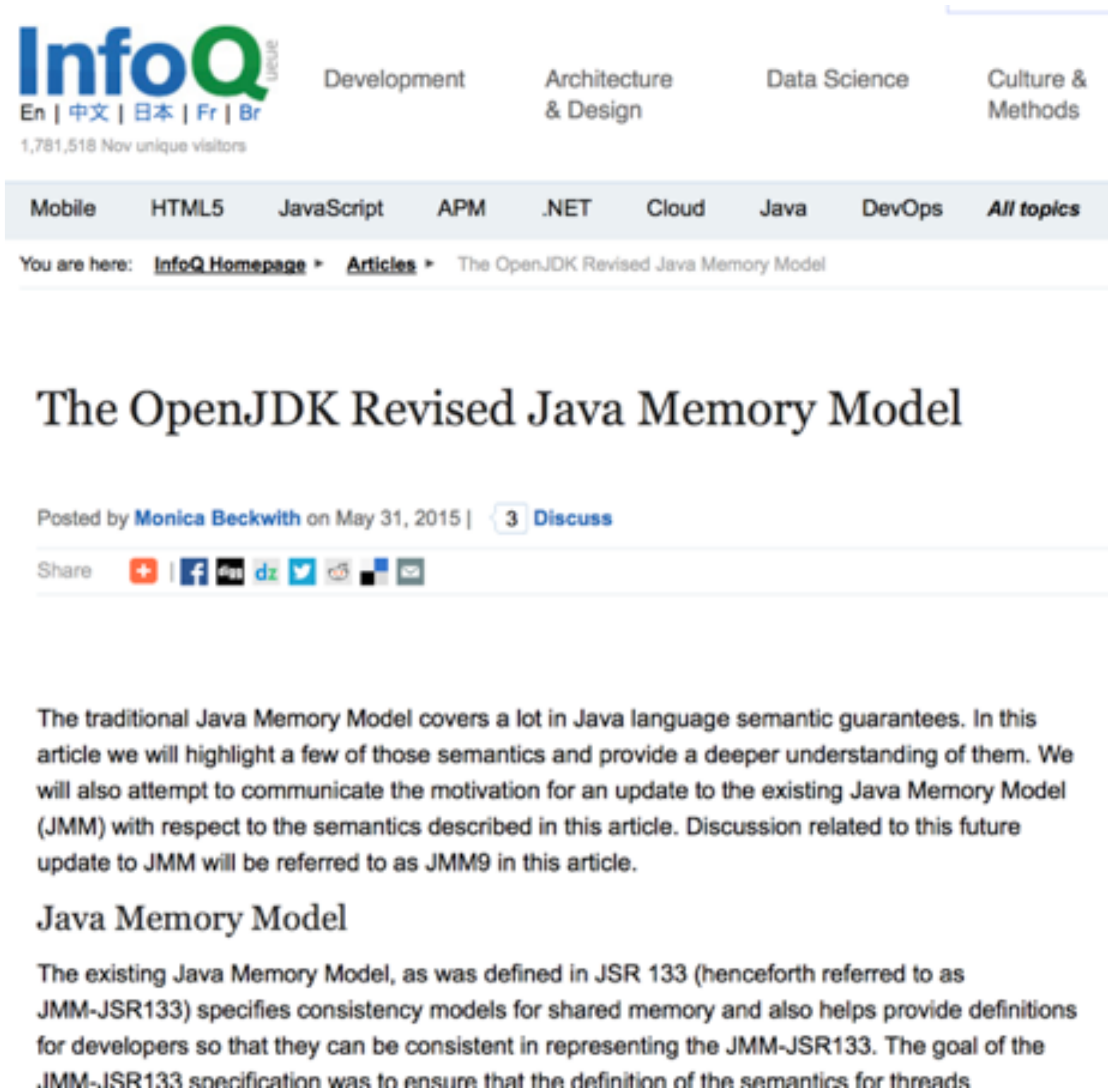


JEP 193: Variable Handles

Author	Doug Lea
Owner	Paul Sandoz
Created	2014/01/06 20:00
Updated	2015/07/23 22:32
Type	Feature
Status	Targeted
Component	core-libs
Scope	SE
JSR	TBD
Discussion	core dash libs dash dev at openjdk dot java dot net
Effort	M
Duration	L
Priority	2
Reviewed by	Dave Dice, Paul Sandoz
Endorsed by	Brian Goetz
Release	9
Issue	8046183
Depends	JEP 188: Java Memory Model Update

Summary

Define a standard means to invoke the equivalents of `java.util.concurrent.atomic` and `sun.misc.Unsafe` operations upon object fields and array elements.


InfoQ
En | 中文 | 日本 | Fr | Br
1,781,518 Nov unique visitors
Development
Architecture & Design
Data Science
Culture & Methods
Mobile
HTML5
JavaScript
APM
.NET
Cloud
Java
DevOps
All topics
You are here: InfoQ Homepage ▸ Articles ▸ The OpenJDK Revised Java Memory Model
Share

The OpenJDK Revised Java Memory Model

Posted by Monica Beckwith on May 31, 2015 | 3 Discuss

The traditional Java Memory Model covers a lot in Java language semantic guarantees. In this article we will highlight a few of those semantics and provide a deeper understanding of them. We will also attempt to communicate the motivation for an update to the existing Java Memory Model (JMM) with respect to the semantics described in this article. Discussion related to this future update to JMM will be referred to as JMM9 in this article.

Java Memory Model

The existing Java Memory Model, as was defined in JSR 133 (henceforth referred to as JMM-JSR133) specifies consistency models for shared memory and also helps provide definitions for developers so that they can be consistent in representing the JMM-JSR133. The goal of the JMM-JSR133 specification was to ensure that the definition of the semantics for threads

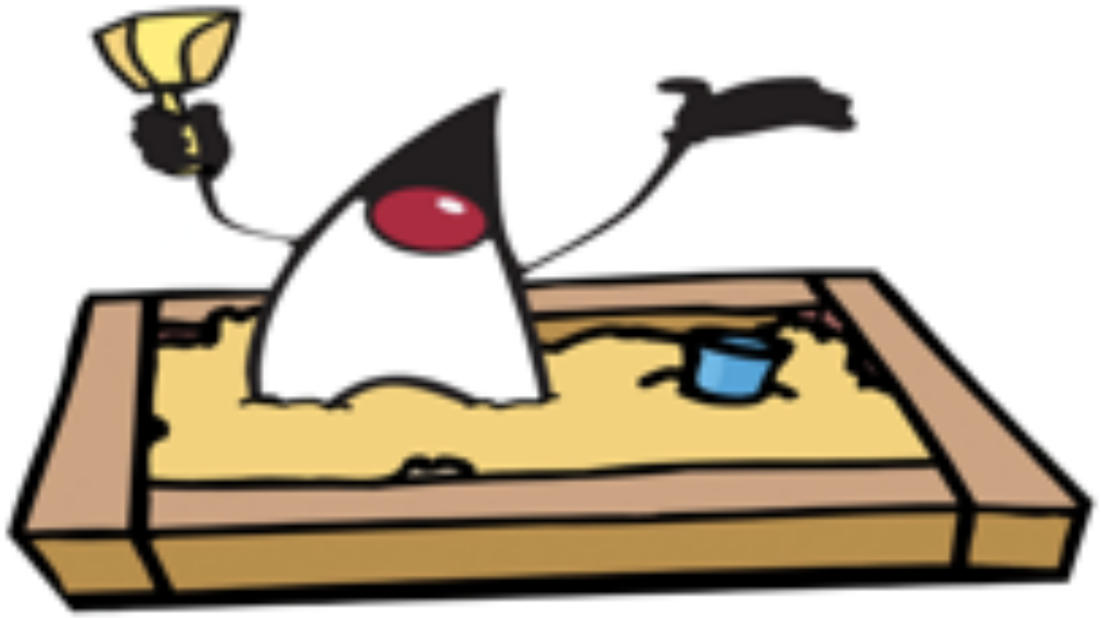
http://www.infoq.com/articles/The-OpenJDK9-Revised-Java-Memory-Model

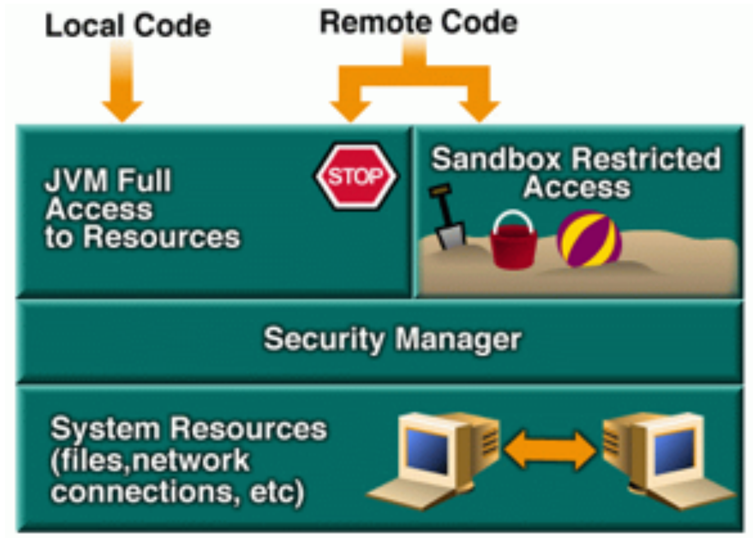
JVM Language Summit 2015
VarHandles (nee enhanced volatiles)

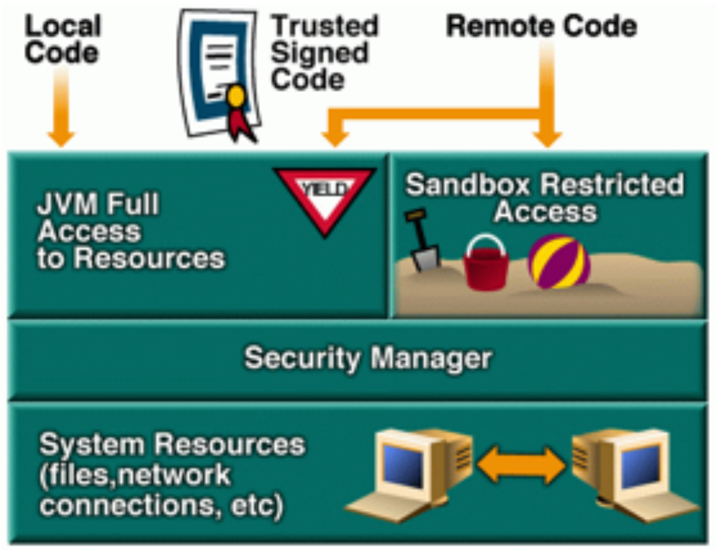
(Paul Sandoz video from August 11th 2015)
https://www.youtube.com/watch?v=ycKn18LtNtk

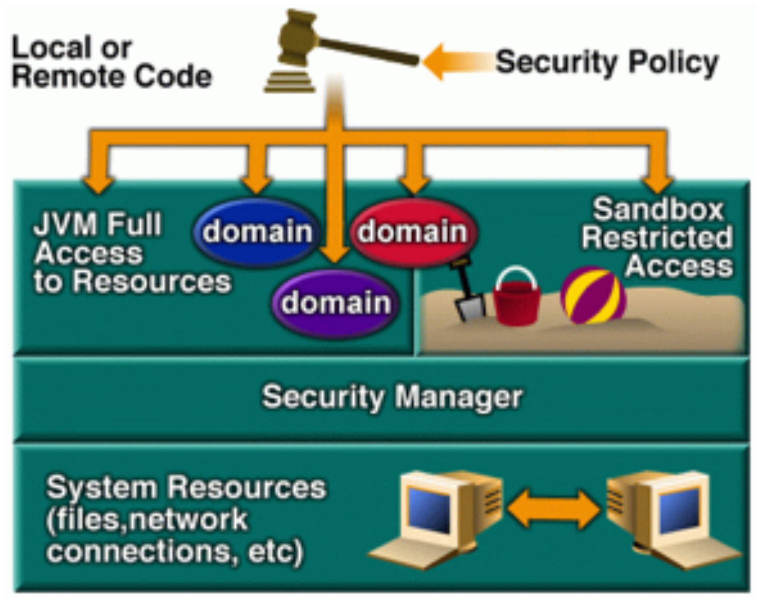
Variable handles
http://openjdk.java.net/jeps/193
https://bugs.openjdk.java.net/browse/JDK-8080588

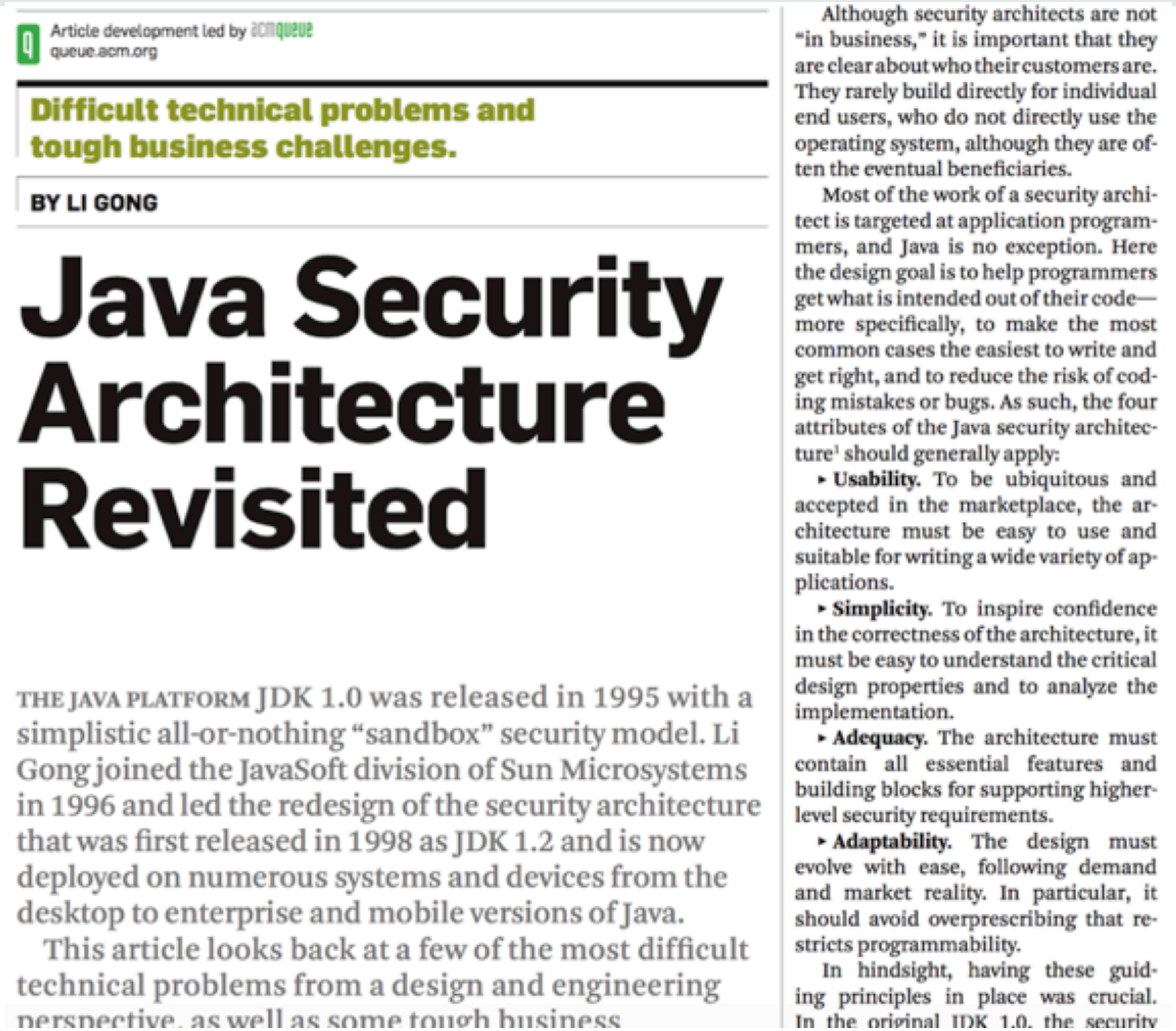
Security




Local Code
Remote Code
JVM Full Access to Resources
STOP
Sandbox Restricted Access
Security Manager
System Resources (files,network connections, etc)


Local Code
Trusted Signed Code
Remote Code
JVM Full Access to Resources
YIELD
Sandbox Restricted Access
Security Manager
System Resources (files,network connections, etc)


Local or Remote Code
Security Policy
JVM Full Access to Resources
domain
domain
domain
Sandbox Restricted Access
Security Manager
System Resources (files,network connections, etc)



Article development led by acmqueue
queue.acm.org

Difficult technical problems and tough business challenges.

BY LI GONG

Java Security Architecture Revisited

THE JAVA PLATFORM JDK 1.0 was released in 1995 with a simplistic all-or-nothing "sandbox" security model. Li Gong joined the JavaSoft division of Sun Microsystems in 1996 and led the redesign of the security architecture that was first released in 1998 as JDK 1.2 and is now deployed on numerous systems and devices from the desktop to enterprise and mobile versions of Java.

This article looks back at a few of the most difficult technical problems from a design and engineering perspective, as well as some tough business

Although security architects are not "in business," it is important that they are clear about who their customers are. They rarely build directly for individual end users, who do not directly use the operating system, although they are often the eventual beneficiaries.

Most of the work of a security architect is targeted at application programmers, and Java is no exception. Here the design goal is to help programmers get what is intended out of their code—more specifically, to make the most common cases the easiest to write and get right, and to reduce the risk of coding mistakes or bugs. As such, the four attributes of the Java security architecture[1] should generally apply:

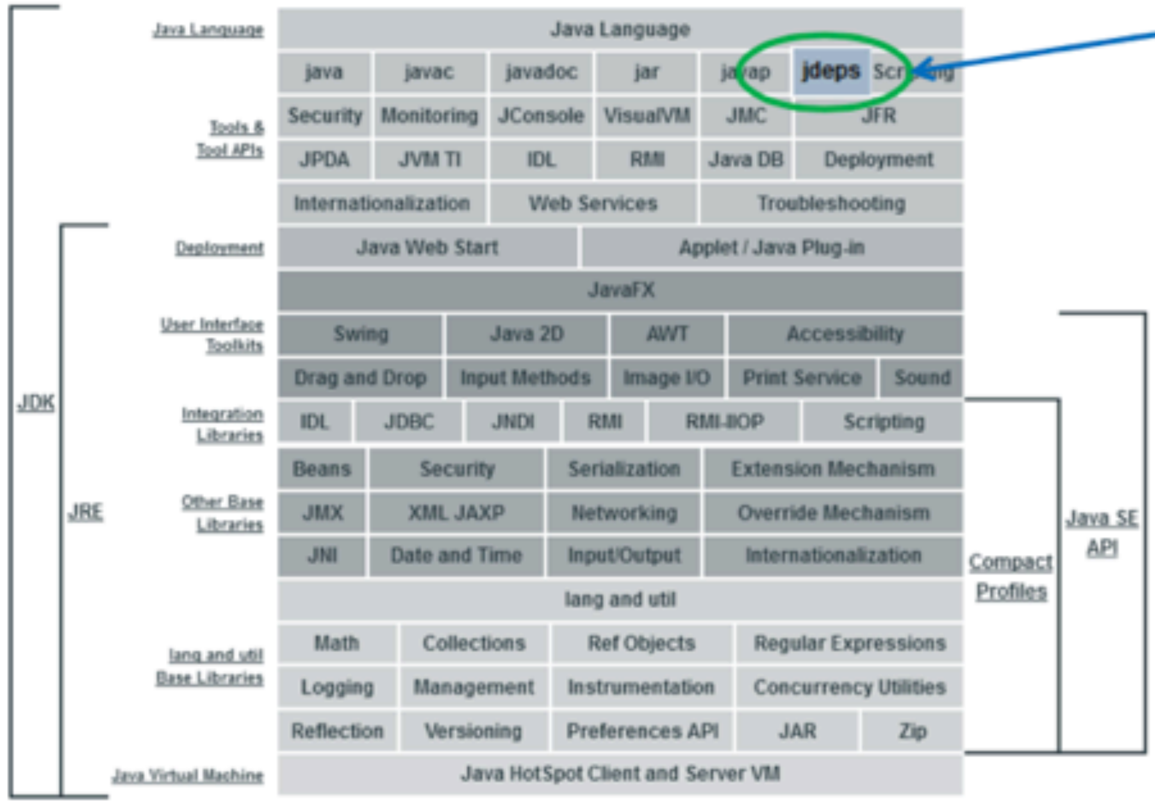
- **Usability.** To be ubiquitous and accepted in the marketplace, the architecture must be easy to use and suitable for writing a wide variety of applications.
- **Simplicity.** To inspire confidence in the correctness of the architecture, it must be easy to understand the critical design properties and to analyze the implementation.
- **Adequacy.** The architecture must contain all essential features and building blocks for supporting higher-level security requirements.
- **Adaptability.** The design must evolve with ease, following demand and market reality. In particular, it should avoid overprescribing that restricts programmability.

In hindsight, having these guiding principles in place was crucial. In the original JDK 1.0, the security

Java Security

- Java Language Security and Bytecode Verification
- Security APIs and Libraries
  - Cryptography: JCA/JCE
  - Public-Key Infrastructure (PKI): Certificates, CertPaths
  - Authentication: JAAS, Kerberos
  - Secure Communication: JSSE (SSL/TLS), GSS-API, SASL
  - Access Control: Security Manager, Policy, JAAS
  - XML Signature
- Tools: jarsigner, keytool

Asset: Java Runtime. Concern: Work on upgrade.



JDeps

Find where your code and dependencies use undocumented internal APIs.

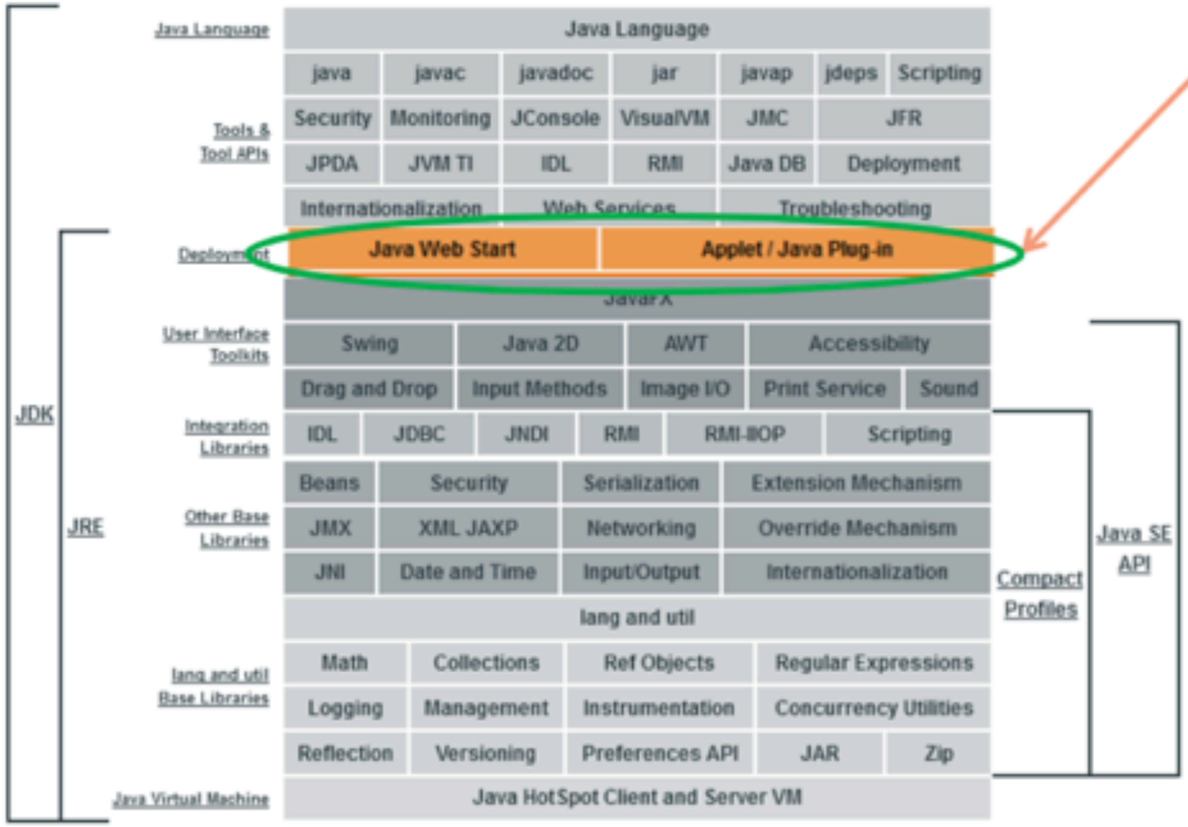
Change to supported APIs.

https://wiki.openjdk.java.net/display/JDK8/Java+Dependency+Analysis+Tool

Maybe upgrade some libraries.

https://wiki.openjdk.java.net/display/quality/Quality+Outreach

Asset: Java Runtime. Concern: Client technology.

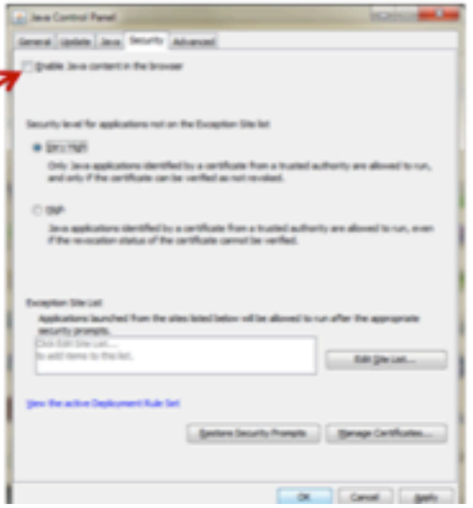


Web Start and/or Applet

Most JavaOne attendees are probably writing server applications and not using this anyways.

If you use the Server JRE, it's not even there.

Or just turn it off in the control panel.



Java Control Panel

General | Update | Java | Security | Advanced

☐ Enable Java content in the browser

Security level for applications not on the Exception Site list

● Very High

Only Java applications identified by a certificate from a trusted authority are allowed to run, and only if the certificate can be verified as not revoked.

○ High

Java applications identified by a certificate from a trusted authority are allowed to run, even if the revocation status of the certificate cannot be verified.

Exception Site List

Applications launched from the sites listed below will be allowed to run after the appropriate security prompts.

Click Edit Site List...
to add items to this list.

Edit Site List...

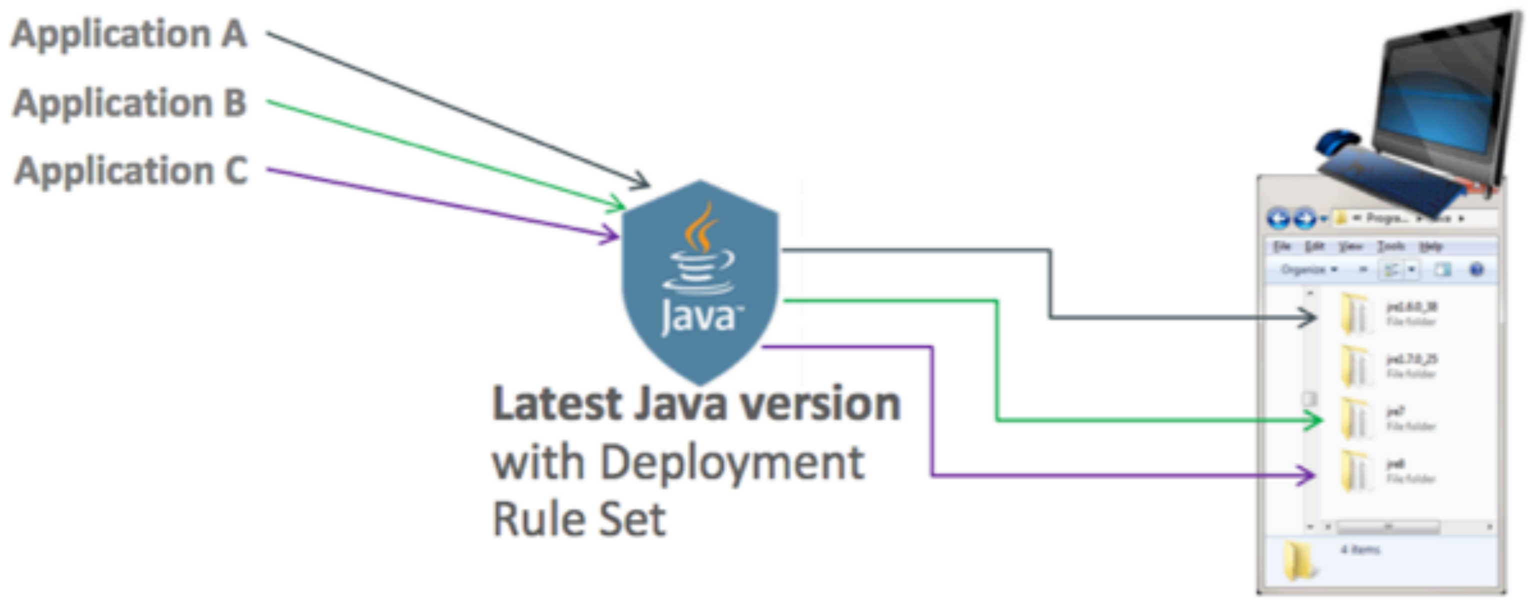
View the active Deployment Rule Set

Restore Security Prompts | Manage Certificates...

OK | Cancel | Apply

If you do use client: **Deployment Rule Set**

- Security of current JRE, compatibility with older versions.
- Can control dialogs.
- Can default-block.



Reduce Attack Surface: Compact Profiles (JDK 8) and Jigsaw (JDK 9, future)

- **Asset:** My application / runtime.
- **Threat:** Unknown future risk.
- **Mitigation:** Remove unused pieces.
- Difficulty: Removing things you do need breaks your own program.

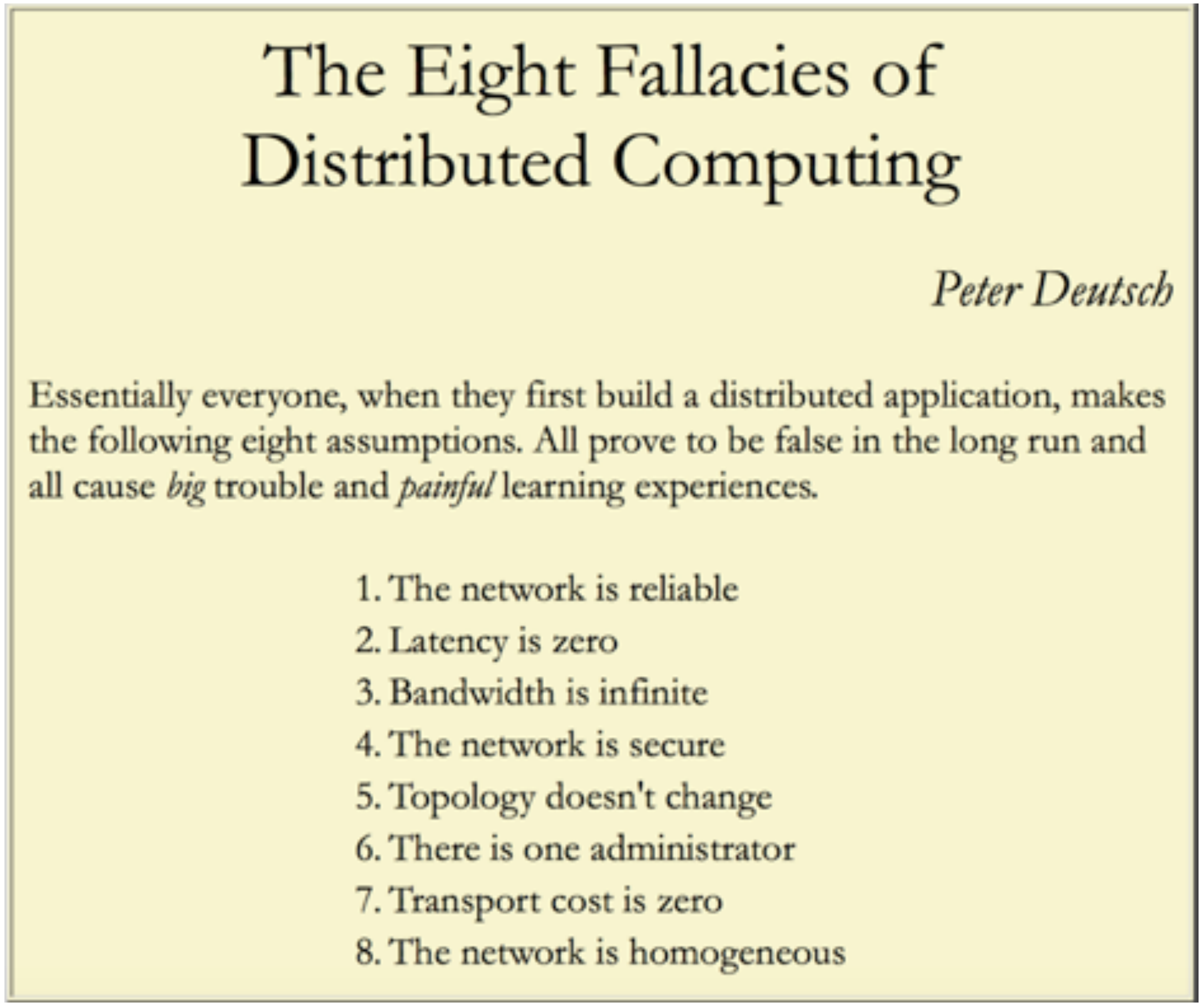
JDK 8 (embedded):
Regular JRE: See conceptual diagram. About 163MB.
Compact 3: Remove graphics, CORBA, and sound. About 21MB.
Compact 2: No Kerberos and JMX monitoring. About 15MB.
Compact 1: No JDBC and XML. About 11MB.

JDK 9 Security Features

http://openjdk.java.net/jeps/0

- JEP 219: Datagram Transport Layer Security (DTLS)
- JEP 229: Create PKCS12 Keystores by Default
- JEP 232: Improve Secure Application Performance
- JEP 244: TLS Application-Layer Protocol Negotiation Extension
- JEP 246: Leverage CPU Instructions for GHASH and RSA
- JEP 249: OCSP Stapling for TLS
- JEP 273: DRBG-Based SecureRandom Implementations

Distributed Java


The Eight Fallacies of Distributed Computing

Peter Deutsch

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause *big* trouble and *painful* learning experiences.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

c 1994 (via James Gosling's old blog)

CORBA
(1991)

http://
(1991)

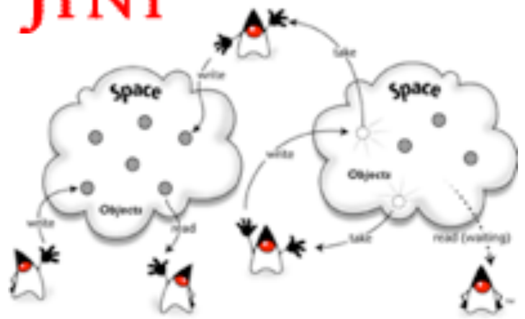
RMI-IIOP

IIOP
(1996)

RMI
(1996)

DCOM
(1996)

JINI



Project JXTA
(2001)

MQQT/COAP
XMPP etc

http2
(2014)

WebSockets
(2011)

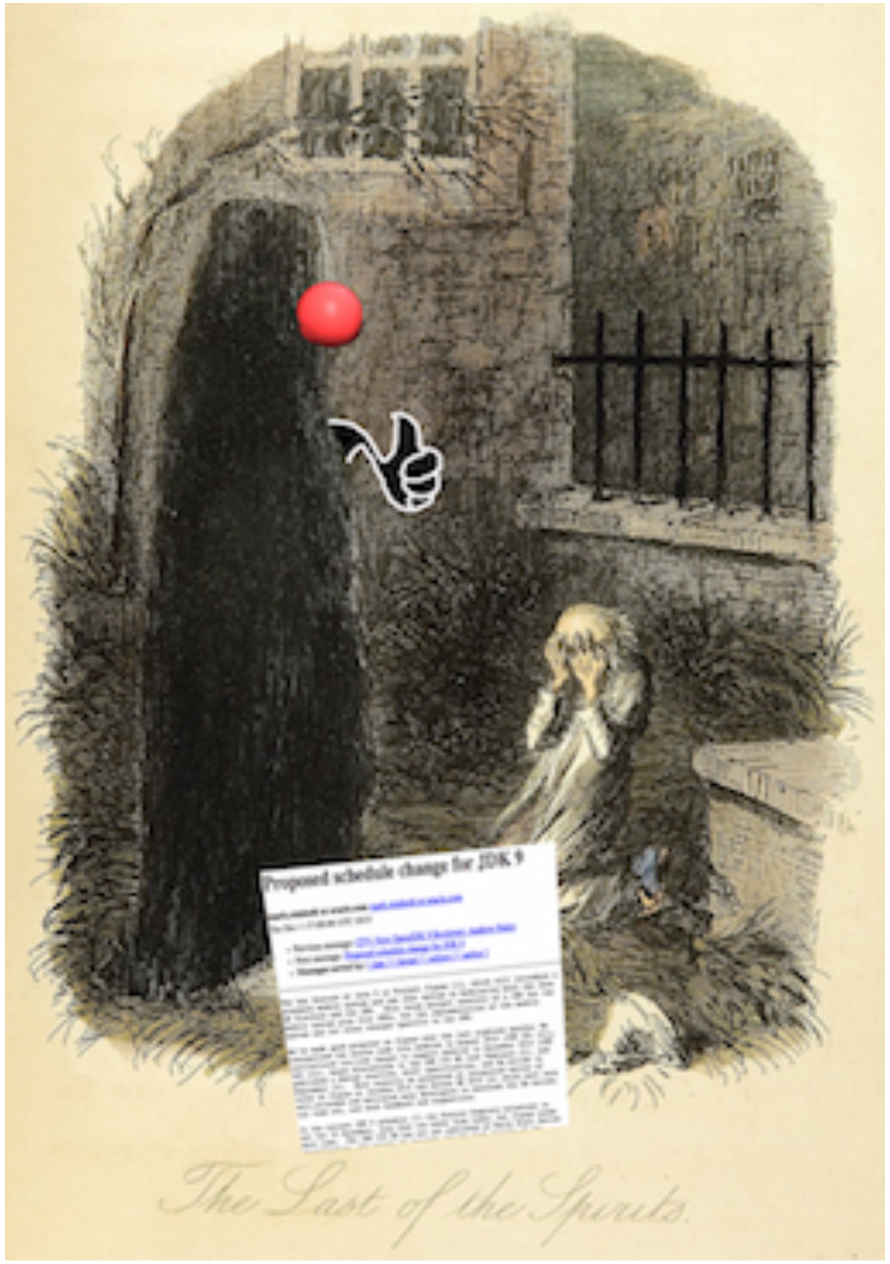
Web Services
SOAP
(1998)

SSL / TLS
(1995) (1999)

REST
(2000 - http1.1 1996-1999)

ORACLE
Java
1995–2015
20
YEARS
#1 Development Platform,
Now in the
Cloud
#Java20

EBENEZER
SCROOGE
The Last of the Spirits.



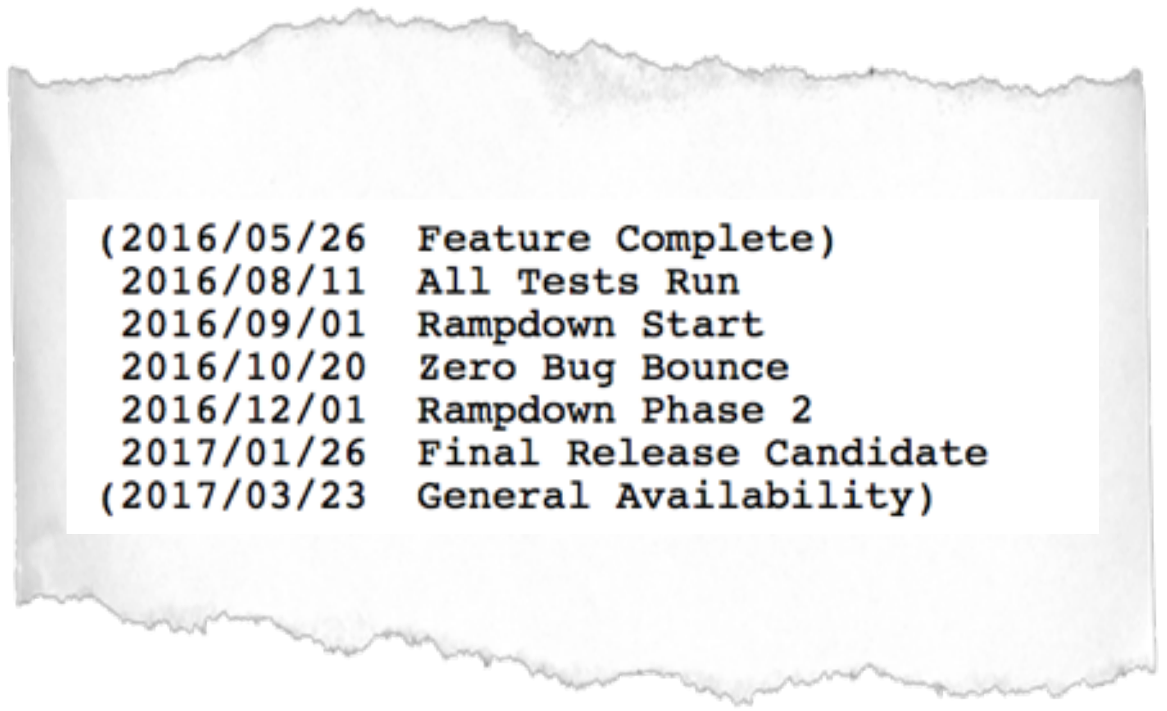
Proposed schedule change for JDK 9

[illegible]

Roadmap Update

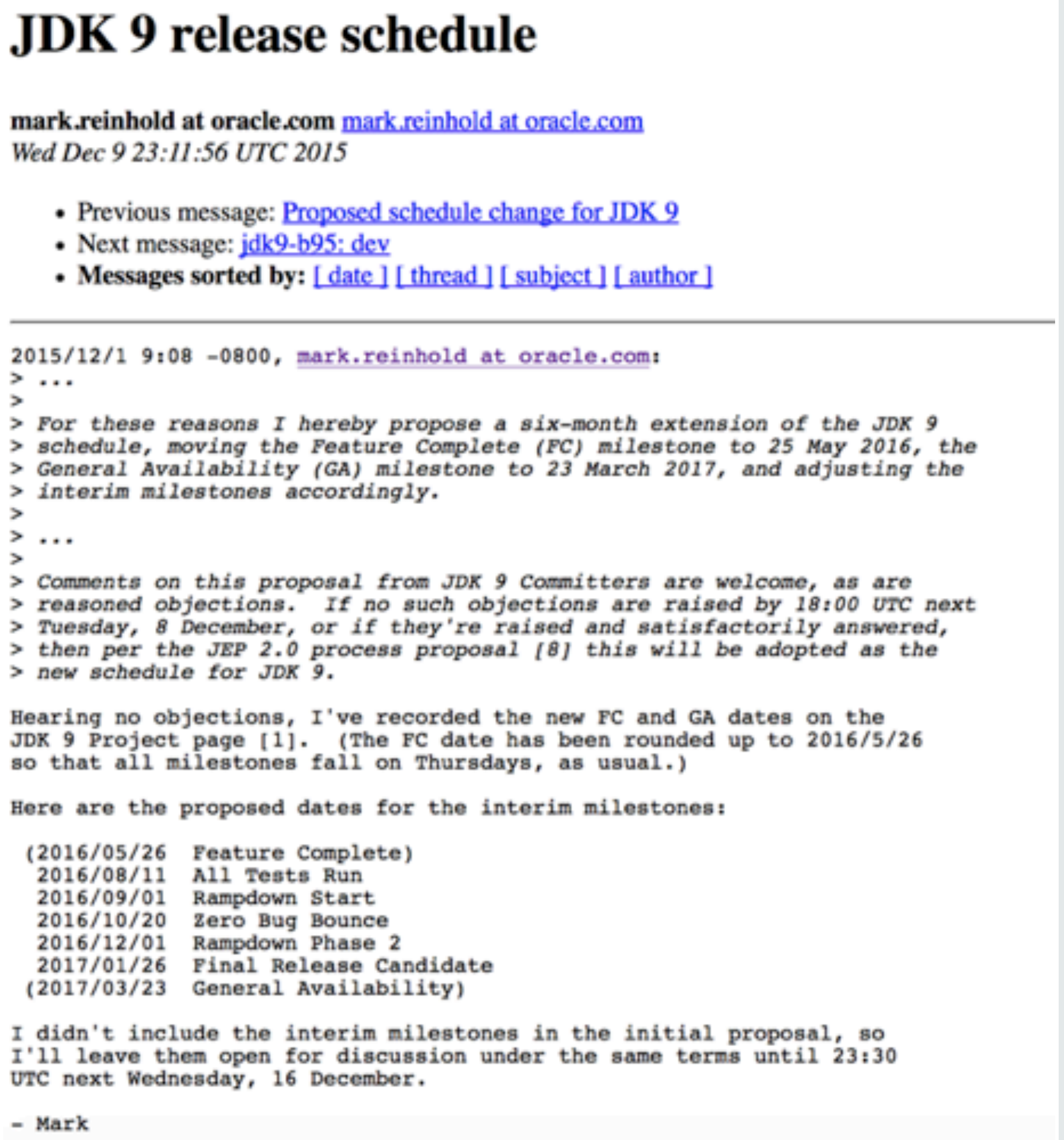
Updated version post talk on 8th Dec.

http://mail.openjdk.java.net/pipermail/jdk9-dev/2015-December/003237.html



```
(2016/05/26   Feature Complete)
 2016/08/11   All Tests Run
 2016/09/01   Rampdown Start
 2016/10/20   Zero Bug Bounce
 2016/12/01   Rampdown Phase 2
 2017/01/26   Final Release Candidate
(2017/03/23   General Availability)
```

http://openjdk.java.net/projects/jdk9
https://wiki.openjdk.java.net/display/Adoption/JDK+9+Outreach



JDK 9 release schedule

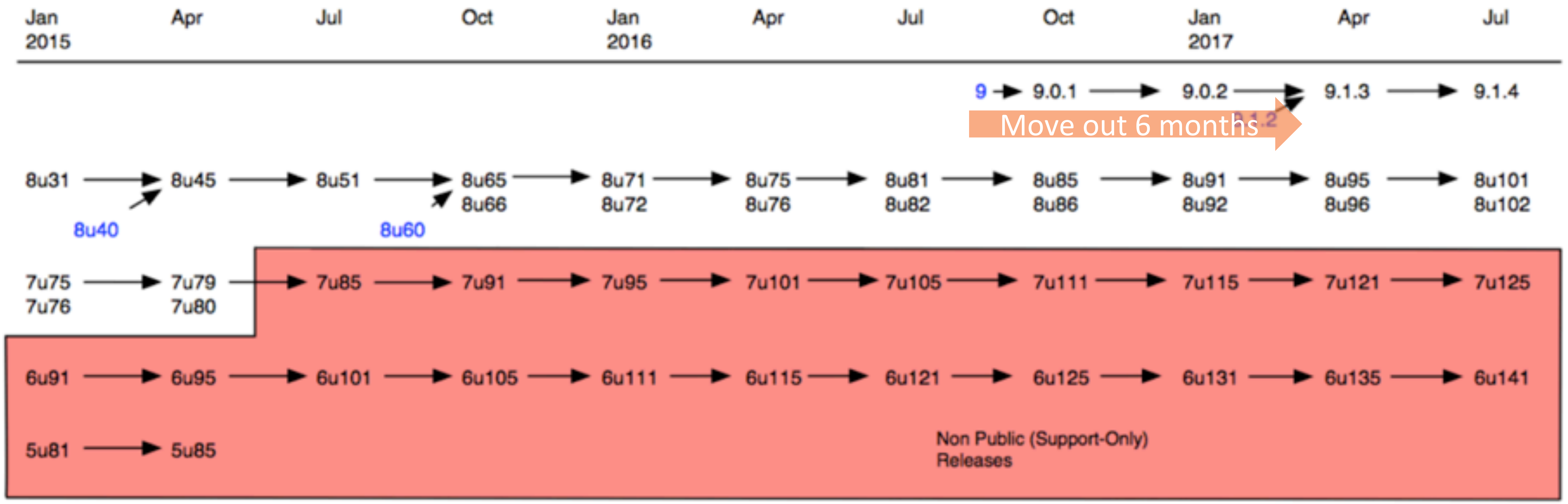
mark.reinhold at oracle.com mark.reinhold at oracle.com
Wed Dec 9 23:11:56 UTC 2015

- Previous message: Proposed schedule change for JDK 9
- Next message: jdk9-b95: dev
- **Messages sorted by:** [date] [thread] [subject] [author]

```
2015/12/1 9:08 -0800, mark.reinhold at oracle.com:
> ...
>
> For these reasons I hereby propose a six-month extension of the JDK 9
> schedule, moving the Feature Complete (FC) milestone to 25 May 2016, the
> General Availability (GA) milestone to 23 March 2017, and adjusting the
> interim milestones accordingly.
>
> ...
>
> Comments on this proposal from JDK 9 Committers are welcome, as are
> reasoned objections.  If no such objections are raised by 18:00 UTC next
> Tuesday, 8 December, or if they're raised and satisfactorily answered,
> then per the JEP 2.0 process proposal [8] this will be adopted as the
> new schedule for JDK 9.

Hearing no objections, I've recorded the new FC and GA dates on the
JDK 9 Project page [1].  (The FC date has been rounded up to 2016/5/26
so that all milestones fall on Thursdays, as usual.)

Here are the proposed dates for the interim milestones:

 (2016/05/26  Feature Complete)
  2016/08/11  All Tests Run
  2016/09/01  Rampdown Start
  2016/10/20  Zero Bug Bounce
  2016/12/01  Rampdown Phase 2
  2017/01/26  Final Release Candidate
 (2017/03/23  General Availability)

I didn't include the interim milestones in the initial proposal, so
I'll leave them open for discussion under the same terms until 23:30
UTC next Wednesday, 16 December.

- Mark
```

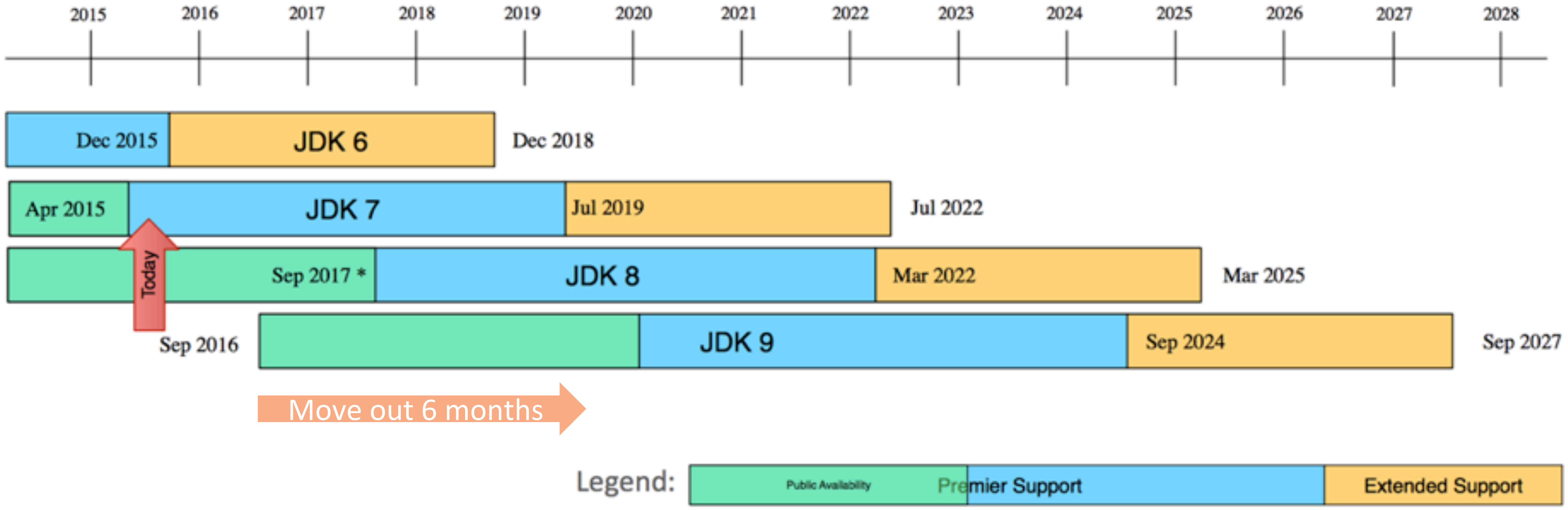
Next Feature Release: JDK 9 – ~~2016-09-22~~ 23rd March 2016

Periodic security fixes and bug fixes for all releases until then



Last reviewed on 2015/10 All future release dates subject to change

Current Releases – Support Period



Java
End of Public Updates (EoPU)

- Public Java updates are available until all three of these conditions occur
  - Three years after general availability
  - One year after being superseded by a new major release
  - Six months after the new major release is made the default on java.com

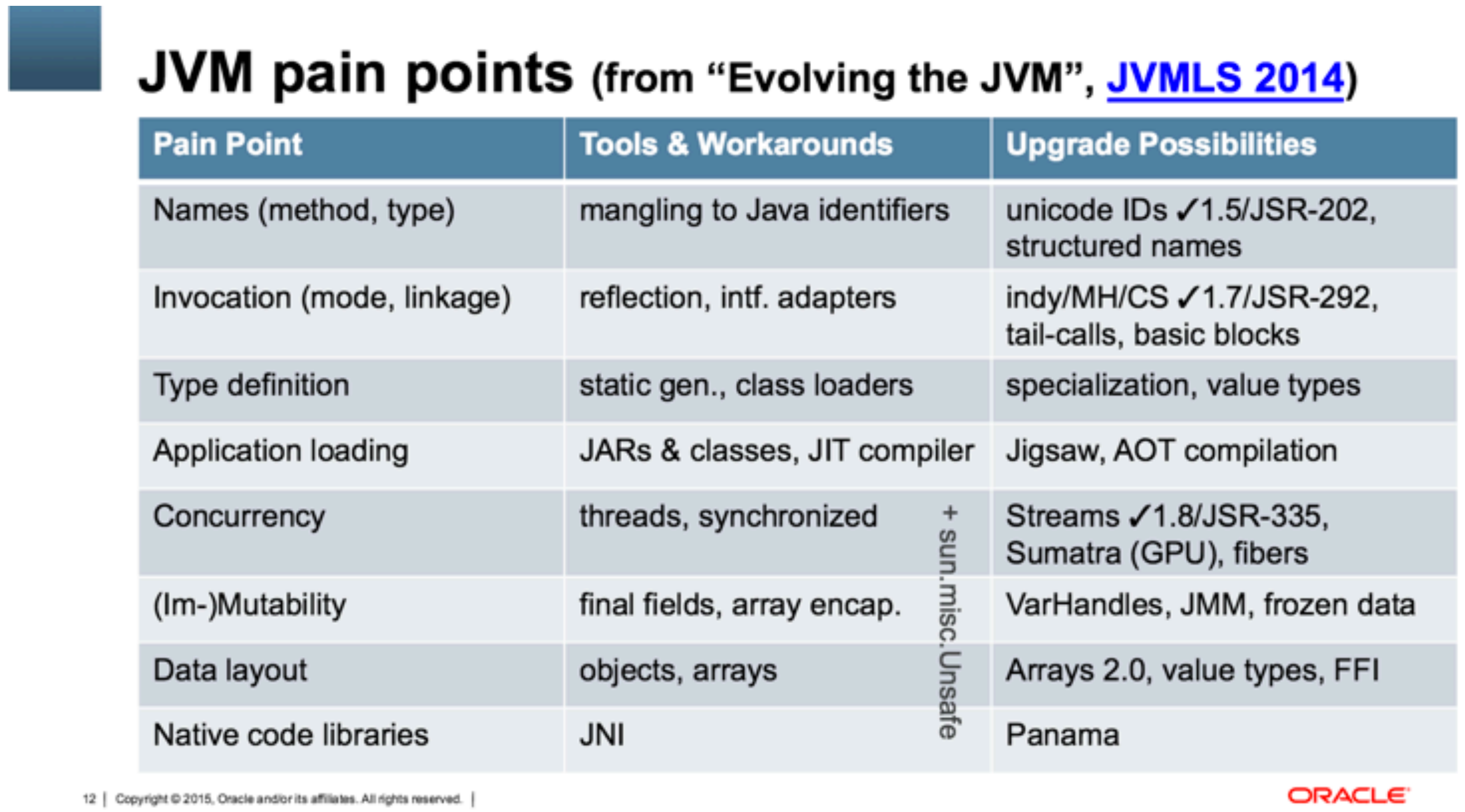
Post Java 9

- **Project Valhalla** http://openjdk.java.net/projects/valhalla
  - Value Types – aggregates without identity
    http://cr.openjdk.java.net/~jrose/values/values-0.html
  - Specialization – templated types on demand
    http://cr.openjdk.java.net/~briangoetz/valhalla/specialization.html
  - JMM Update – VarHandles
- **Project Panama** http://openjdk.java.net/projects/panama
  - Arrays 2.0 – flexible array implementation and organization
  - Layouts – flexible object layout
  - FFI (JEP 191) – better native code interop

http://mail.openjdk.java.net/pipermail/valhalla-dev
http://mail.openjdk.java.net/pipermail/panama-dev

John Rose @ JVM language summit July 2014

http://www.oracle.com/technetwork/java/javase/community/jlssessions-2255337.html



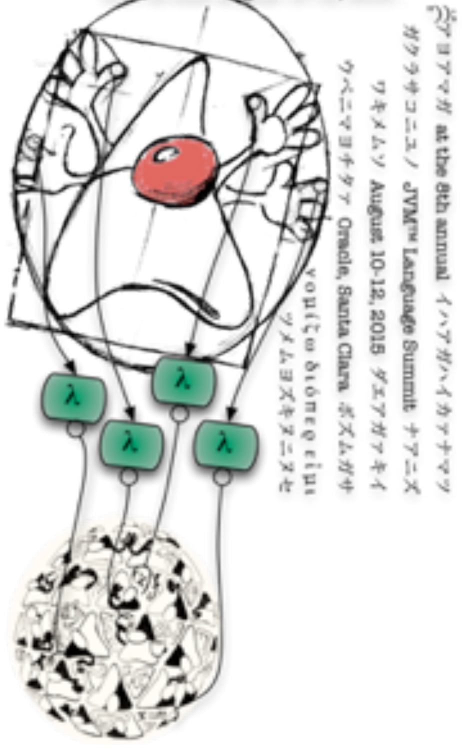
JVM pain points (from “Evolving the JVM”, JVMLS 2014)

Pain Point	Tools & Workarounds	Upgrade Possibilities
Names (method, type)	mangling to Java identifiers	unicode IDs ✓1.5/JSR-202, structured names
Invocation (mode, linkage)	reflection, intf. adapters	indy/MH/CS ✓1.7/JSR-292, tail-calls, basic blocks
Type definition	static gen., class loaders	specialization, value types
Application loading	JARs & classes, JIT compiler	Jigsaw, AOT compilation
Concurrency	threads, synchronized	Streams ✓1.8/JSR-335, Sumatra (GPU), fibers
(Im-)Mutability	final fields, array encap.	VarHandles, JMM, frozen data
Data layout	objects, arrays	Arrays 2.0, value types, FFI
Native code libraries	JNI	Panama

+ sun.misc.Unsafe

12 | Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

```
implementors().stream()
    .filter(i → i.projects().any(::isLanguage))
    .forEach(i → i.hail("
        Hello language implementors!
        Join us next August 10-12, 2015...
```


"));アヨアマガ at the 8th annual イハアガハイカアナマツ
ガクラキコニユノ JVM™ Language Summit ナアニズ
ワキメムソ August 10-12, 2015 ダエアガアキイ
ウヘニマヨチタア Oracle, Santa Clara ボズムガサ
νομίζω διόπερ ειμι
ツメムヨズキヌニヌヒ
λ
λ
λ
λ

What should the JVM look like in 20 years?

(eight not-so-modest goals)

- Uniform model: Objects, arrays, values, types, methods "feel similar"
- Memory efficient: tunable data layouts, naturally local, pointer-thrifty
- Optimizing: Shared code mechanically customized to each hot path
- Post-threaded: Routine confinement/immutability, granular concurrency
- Interoperable: Robust integration with non-managed languages
- Broadly useful: Safely and reliably runs most modern languages.
- Compatible: Runs 30-year-old dusty JARs.
- Performant: Gets the most out of major CPUs and systems.

ORACLE

Java ORACLE

Happy Holidays from Oracle Technology Network
ORACLE
TECHNOLOGY NETWORK
oracle.com/technetwork/java

MERRY CHRISTMAS

Acknowledgments & Links (1/2)

- Wikipedia
  - https://en.wikipedia.org/wiki/File:Charles_Dickens-A_Christmas_Carol-Title_page-First_edition_1843.jpg
  - https://en.wikipedia.org/wiki/File:The_Last_of_the_Spirits-John_Leech,_1843.jpg
  - https://en.wikipedia.org/wiki/File:TI_microSPARC_I_die.jpg
  - https://en.wikipedia.org/wiki/File:Intel_Pentium_P54C_die.jpg
- JVM Language Summit
  - http://openjdk.java.net/projects/mlvm/jvmlangsummit

Acknowledgments & Links (2/2)

- Others
  - Oracle M7 Presentation from HotChips 2014 / Stephen Philips
  - Language Based Virtual Machines / Lars Bak http://aosd.net/2012/images/stories/bak.pdf
  - Aleksey Shipilev http://shipilev.net/blog/2014/jmm-pragmatics
  - Oracle Java Documentation https://docs.oracle.com/javase/specs/jls/se8/html/jls-17.html
  - ACM Queue https://queue.acm.org/detail.cfm?id=2034639
  - The Eight Fallacies of Distributed Computing / James Gosling (old) home page
  - Roadmaps (OpenWorld 2015) / Aurelio Garcia-Ribeyro

  https://published-rs.lanyonevents.com/published/oracleus2015/sessionsFiles/1467/CON9682_Garcia-Ribeyro-CON9682-OOW2015-JavaSEAdvanced.pdf
  - Threat-Modeling the JVM [CON2031] (JavaOne 2015) / Erik Costlow
  - Duke images https://duke.kenai.com
    Merry Christmas https://duke.kenai.com/animations/DukeTuxChristmas.gif