Call for Discussion: Project Skara

Investigating source code management options for the JDK sources

Joseph D. Darcy (darcy, @jddarcy) and Erik Duveblad (ehelin)
Java Platform Group, Oracle
OpenJDK Committers' Workshop
August 2018

Call for Discussion: New Project: Skara

investigating source code management options for the JDK sources

- Lively email thread on discuss@openjdk.java.net
- Reactions on discuss@ojn and elsewhere range from:
  - “The only possible choice is github ftw!”
  - “You can take hg, mq, and webrev.ksh out of my cold, dead hands.”
- Have in-person discussions at this workshop

“The evaluation criteria to consider include but are not limited to:”

- Performance: time for clone operations from master repos, time of local operations, etc.
- Space efficiency
- Usability in different geographies
- Support for common development environments such as Linux, Mac, and Windows
- Able to easily host the entire history of the JDK and the projected growth of its history over the next decade
- Support for general JDK code review practices
- Programmatic APIs to enable process assistance and automation of review and processes

Five stages of changing an SCM

Not an outline for the rest of the discussion

1. Denial
2. Anger
3. Bargaining
4. Depression
5. Acceptance

Preface: a few recent pushes of note

Nest mates – JEP 181

JDK 11 feature

- Pushed as one changeset under JDK-8010319: Implementation of JEP 181: Nest-Based Access Control; updates to:
  - `javac` to generate new class file attributes
  - HotSpot to implement semantics of new class file attributes
  - Core reflection to expose new class file attributes

Start of JDK 12

- In JDK 12, build, HotSpot, and core libs engineers worked together to produce a coordinated set of fixes to address all of
  - 8205615: Start of release updates for JDK 12
  - 8205621: Increment JDK version for JDK 12
  - 8193292: Add SourceVersion.RELEASE_12
  - 8193290: Add source 12 and target 12 to javac
  - 8205619: Bump maximum recognized class file version to 56 for JDK 12
- Fixes pushed in a single changeset into the jdk/jdk master as part of b01 that updates the build, core libs, HotSpot, and javac.

Context: where were we

Integration topology of lines of development, JDK 8

A *graph* of integration hg forests

Integration topology of lines of development, JDK 8

A *graph* of integration hg forests

JDK 8 Master

Integration topology of lines of development, JDK 8

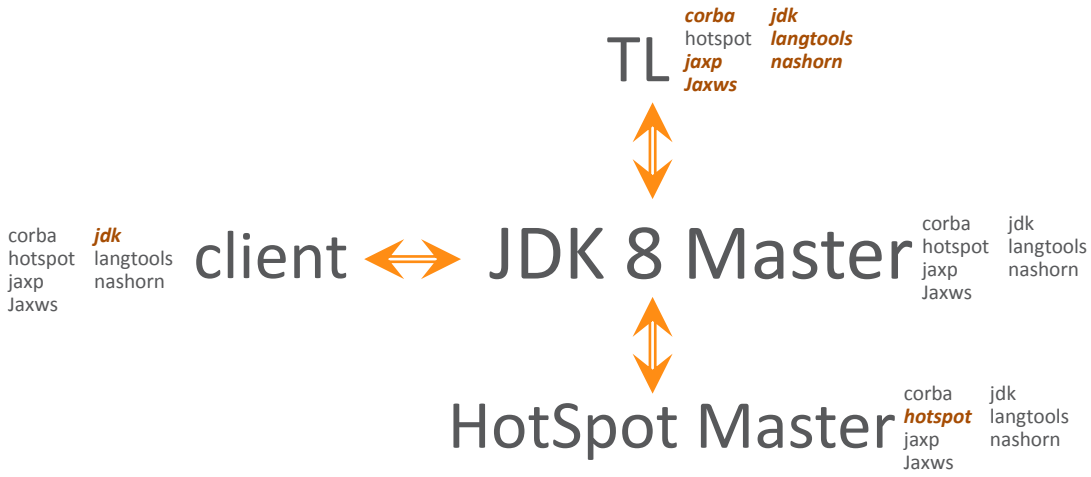
A *graph* of integration hg forests

Integration topology of lines of development, JDK 8

A *graph* of integration hg forests

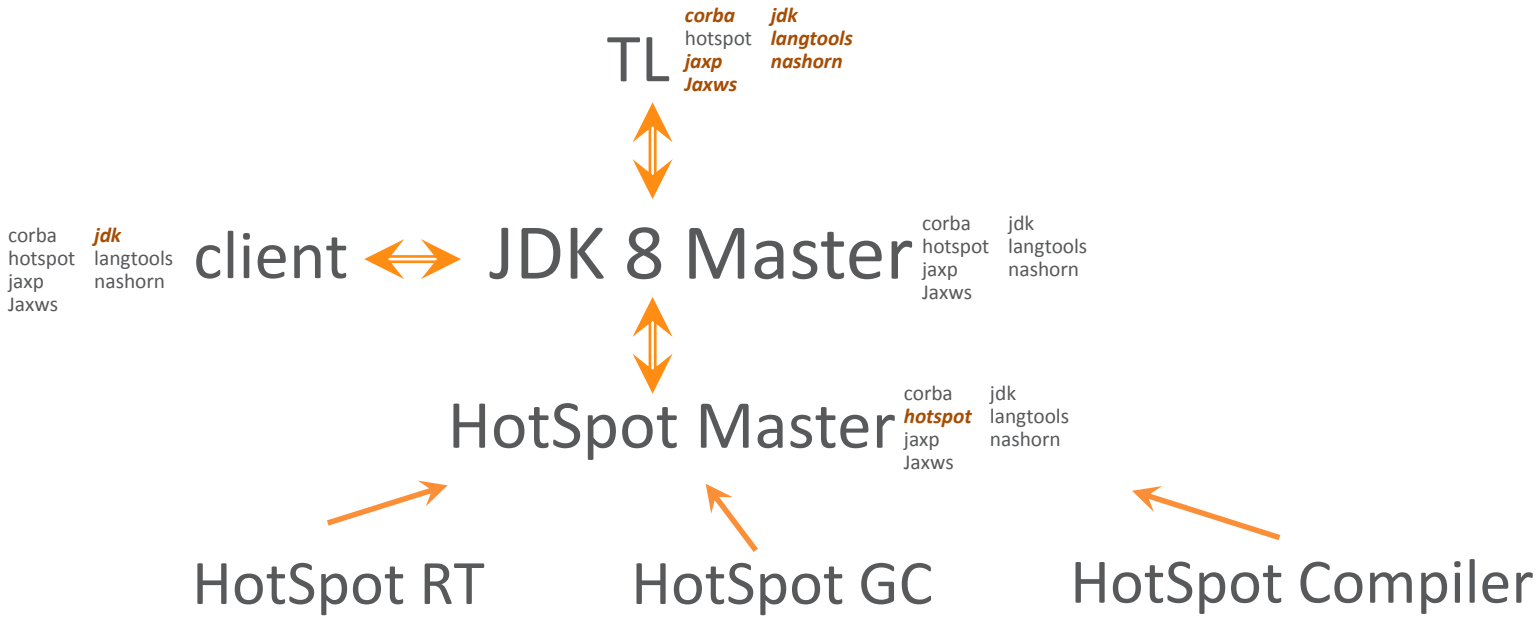
Integration topology of lines of development, JDK 8

A *graph* of integration hg forests



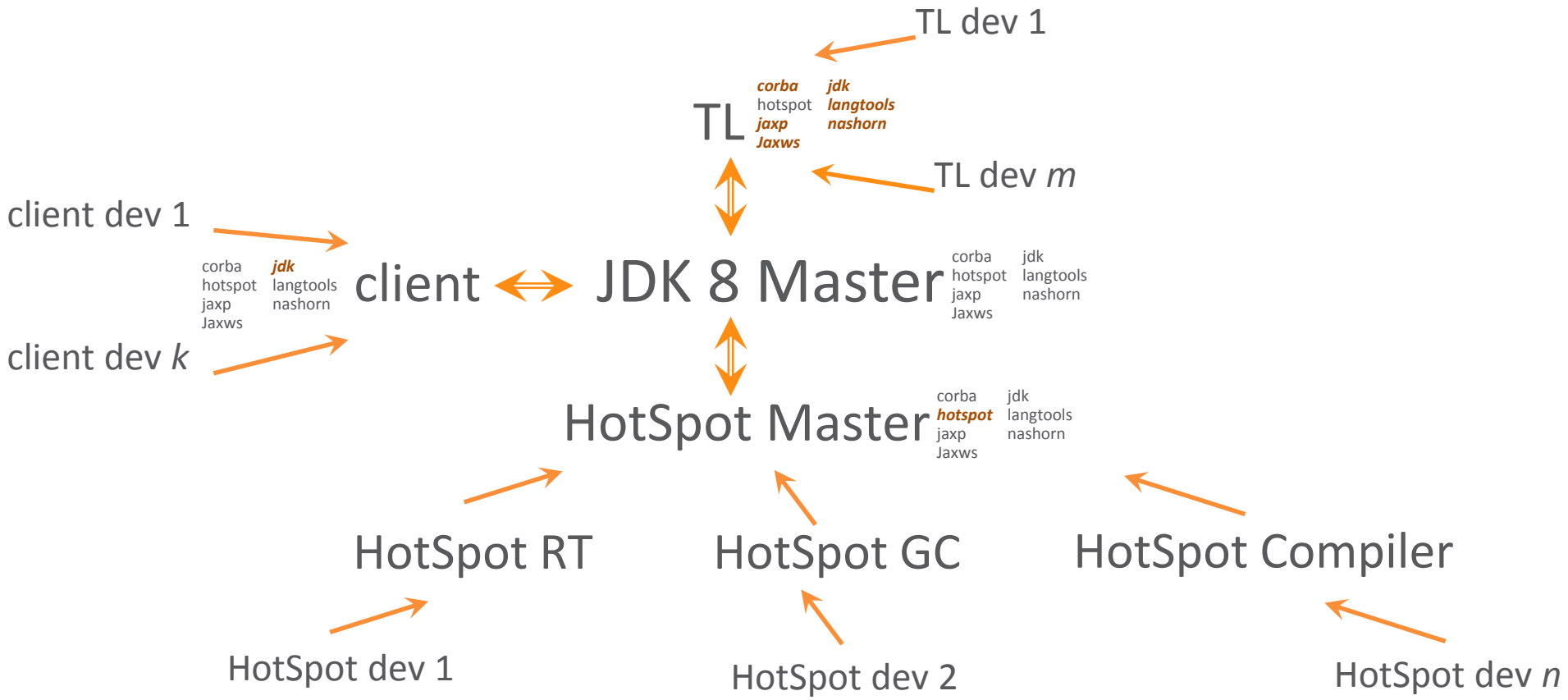
Integration topology of lines of development, JDK 8

A *graph* of integration hg forests



Integration topology of lines of development, JDK 8

A *graph* of integration hg forests



Integration topology of lines of development, JDK 9

A *tree* of integration hg forests

Integration topology of lines of development, JDK 9

A *tree* of integration hg forests

JDK 9 Master

Integration topology of lines of development, JDK 9

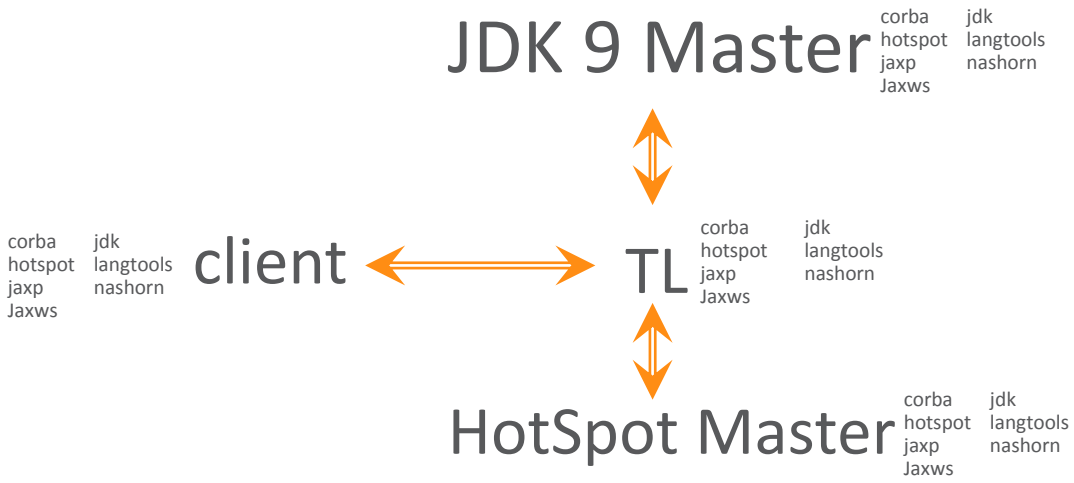
A *tree* of integration hg forests

Integration topology of lines of development, JDK 9

A *tree* of integration hg forests

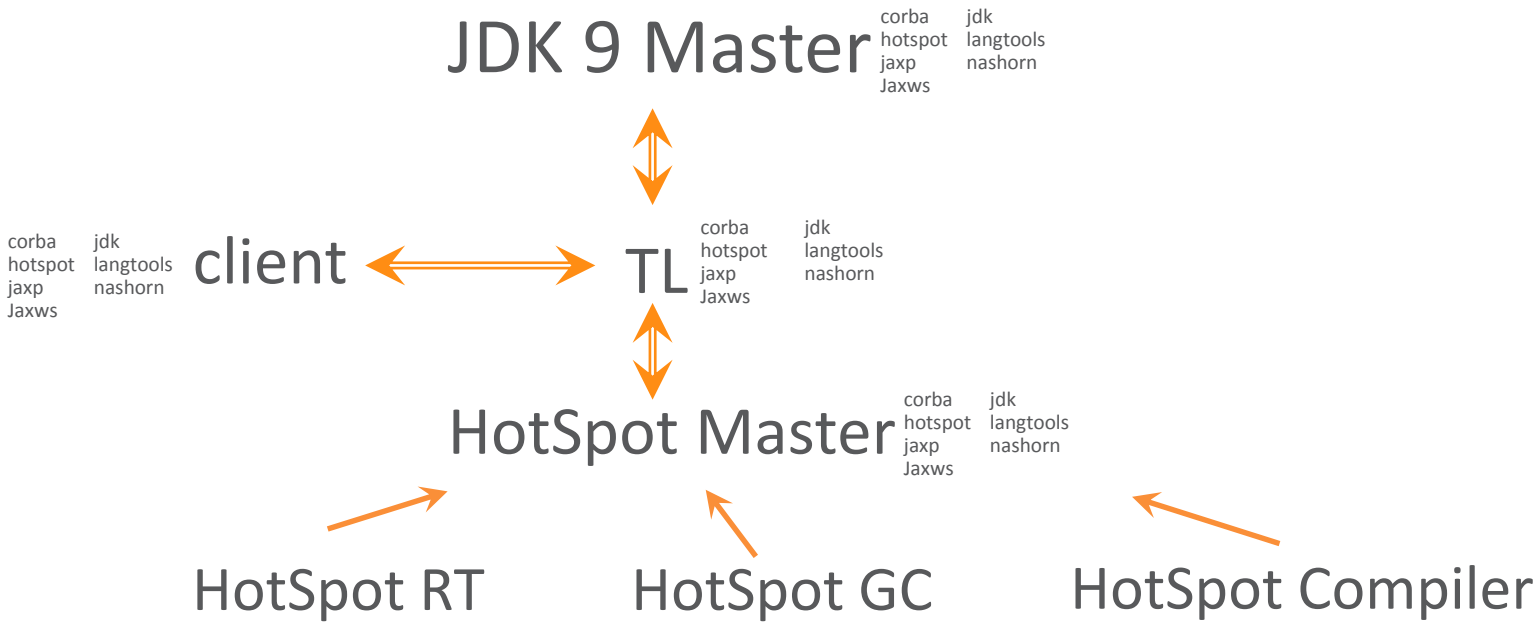
Integration topology of lines of development, JDK 9

A *tree* of integration hg forests



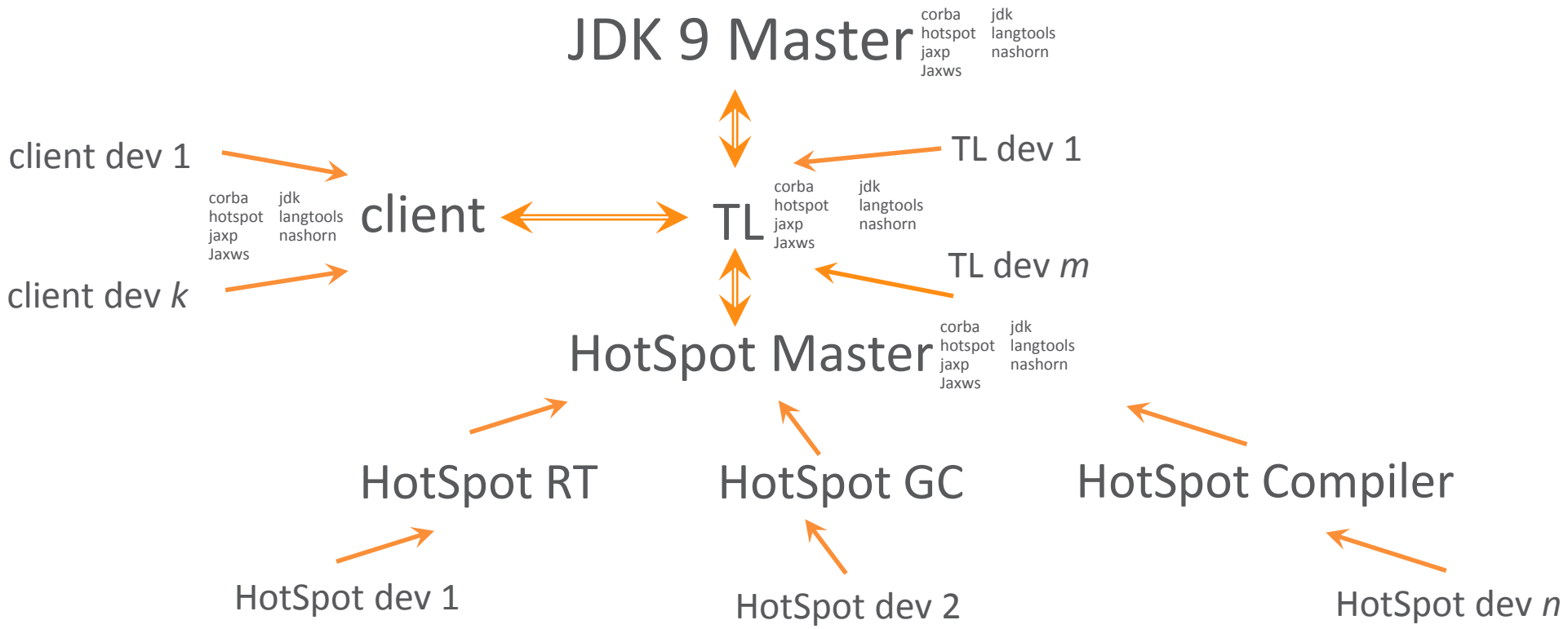
Integration topology of lines of development, JDK 9

A *tree* of integration hg forests



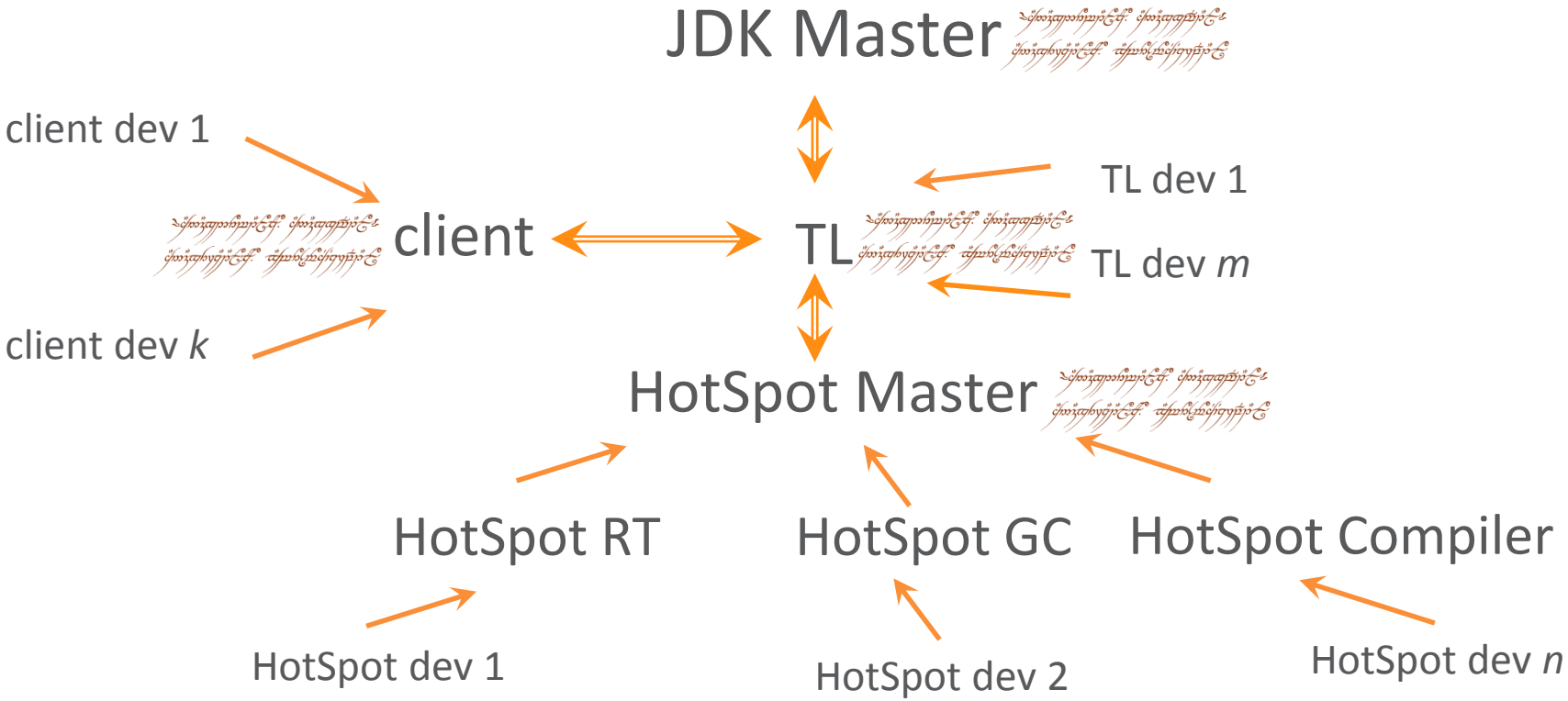
Integration topology of lines of development, JDK 9

A *tree* of integration hg forests



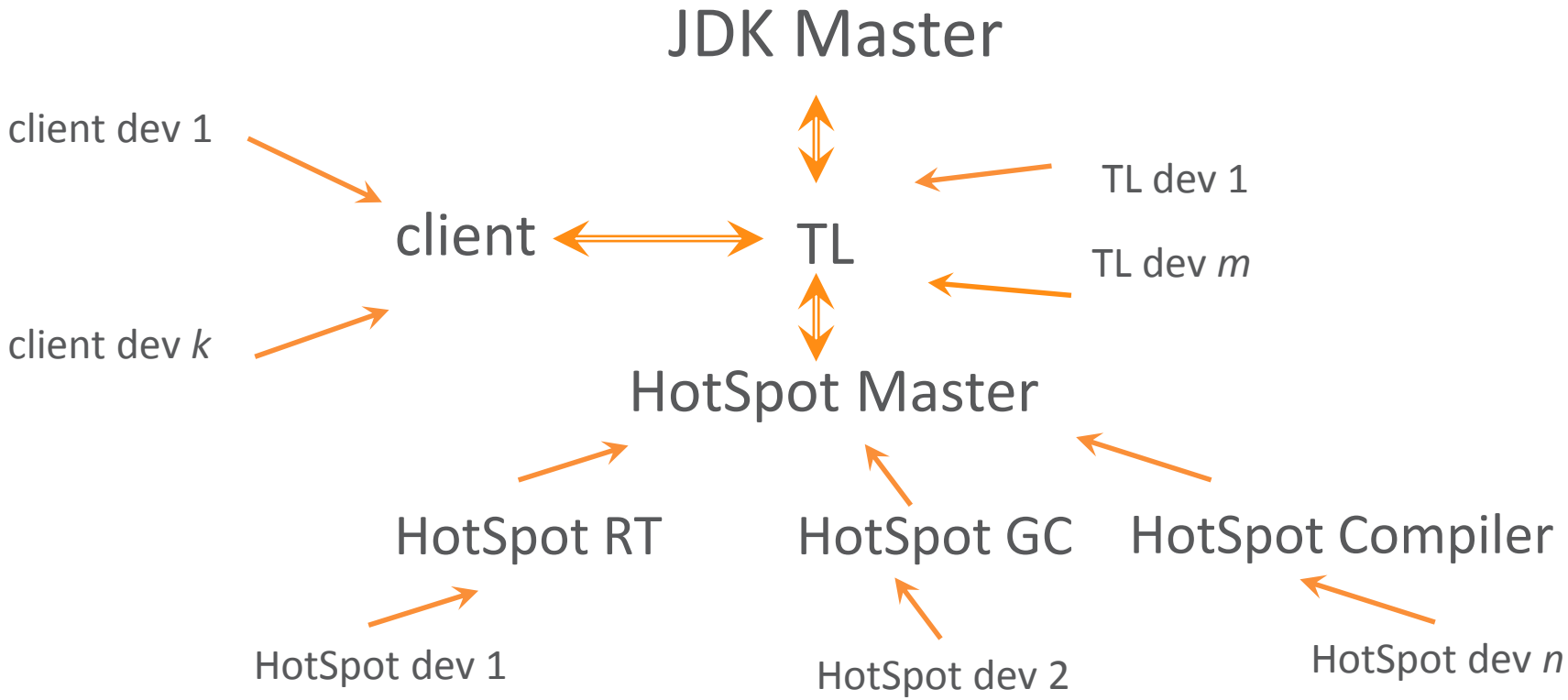
Integration topology of lines of development, JDK 10, 11, 12

Repo consolidation; a tree of integration repos



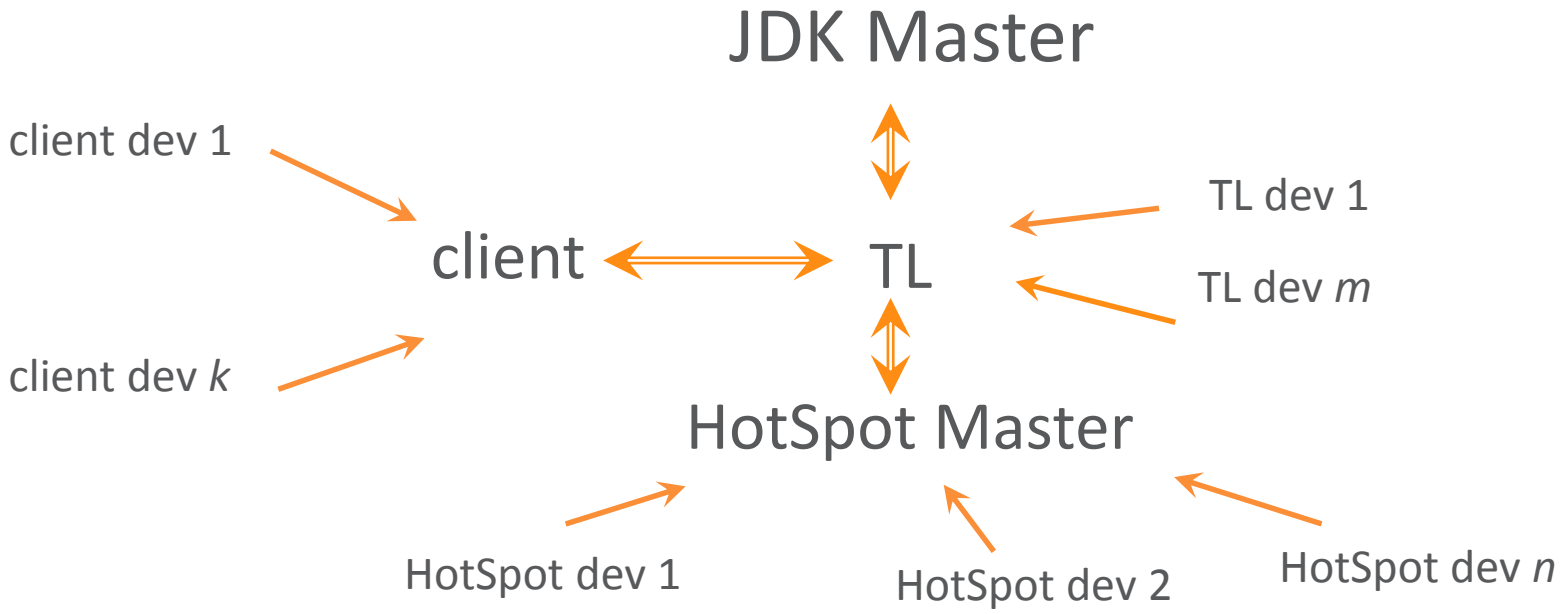
Integration topology of lines of development, JDK 10, 11, 12

Repo consolidation; a tree of integration repos



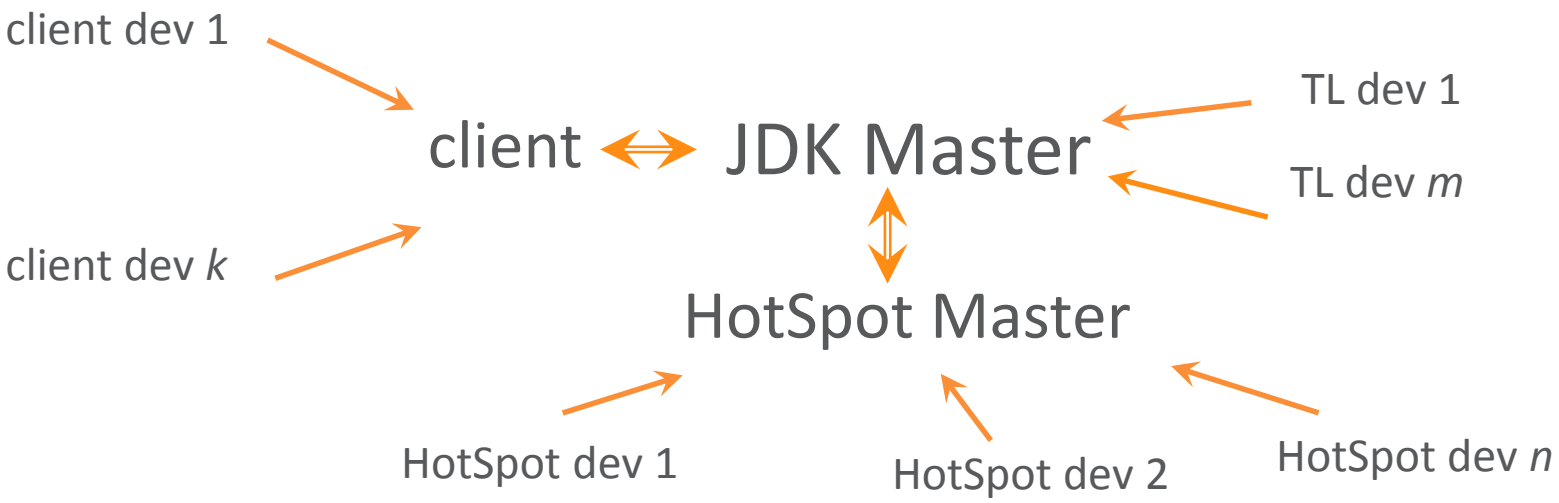
Integration topology of lines of development, JDK 10, 11, 12

Combining HotSpot lines of development



Integration topology of lines of development, JDK 10, 11, 12

Combining master and TL



Integration topology of lines of development, JDK 12

Combining HotSpot Master and master

Summary of changes in JDK 10, 11, and 12

- Repo consolidation in JDK 10 (JEP 296)
- Bulk syncs from JDK (*N* – 1) into JDK *N* starting with *N* = 10
- Disabled duplicate bug id check in jcheck to ease bulk syncing into JDK 10
- Combined dev and master into a single entity “master” in JDK 10, now jdk/jdk; integrations are just tags
- Multiple HotSpot lines of development combined into one
- HotSpot engineers push directly into master, i.e. no dedicated HotSpot line of development (end of JDK 10, JDK 11 so far)

Observations

- Many workflows possible with same tooling and SCM
  - New workflows enable more efficient operation; cross-component changes easier and much faster (no propagation delay across forests)
  - Important to get the high-order bits right
- What workflows and changes do we want to encourage or make easier?

SCM matters

SCM check-in and history

- Have been using hg as the SCM for the JDK for nearly 11 years
- Previously TeamWare used for at least a dozen years, pre-1995 to 2007
  - TeamWare first generation distributed SCM
  - Built on top of SCCS; no notion of cross-file changeset in common mode of usage
- SCM landscape and hosting options have changed since OpenJDK inception
- SCM impacts
  - Operational efficiency (disk usage, clone times, etc.)
  - Review and tools ecosystem
  - Community engagement

“Did you know git is now more popular than Mercurial?”

- Yes. E.g.
  - 2015 Stackoverflow survey on version control:
    - **Git:** **69.3%**
    - SVN: 36.9%
    - **hg:** **7.9%**
    - CVS: 4.2%
    - No VC: 9.3%
- Significant cost to changing SCMs

- 2018 Stackoverflow survey on version control: (all respondents)
  - **Git:** **87.2%**
  - SVN: 16.1%
  - …
  - No VC: 4.8%
  - **hg:** **3.6%**

Boundary Systems

Including, but are not limited to

- Bug database (hgupdater)
- Programmatic structural checks (jcheck)
- Identity management
- Build system (hg tips)
- Code review
- CI systems
- Personal boundary systems: scripts, muscle memory, etc.
- …

SCM and hosting choices

- git != github.com
  - Distinguish between hg → git migration decision and self-hosting an SCM vs. using a hosting provider
  - Many git hosting providers: github, gitlab, bitbucket, various others
  - Hosting providers have free vs paid offerings
- Hosting providers can offer a more attractive cost model for development
  - Lots of repos/branches/lines of development at low transaction cost; enables new development practices
  - Would want to validate speed in different geos, etc.

Summary of state of SCM in the JDK

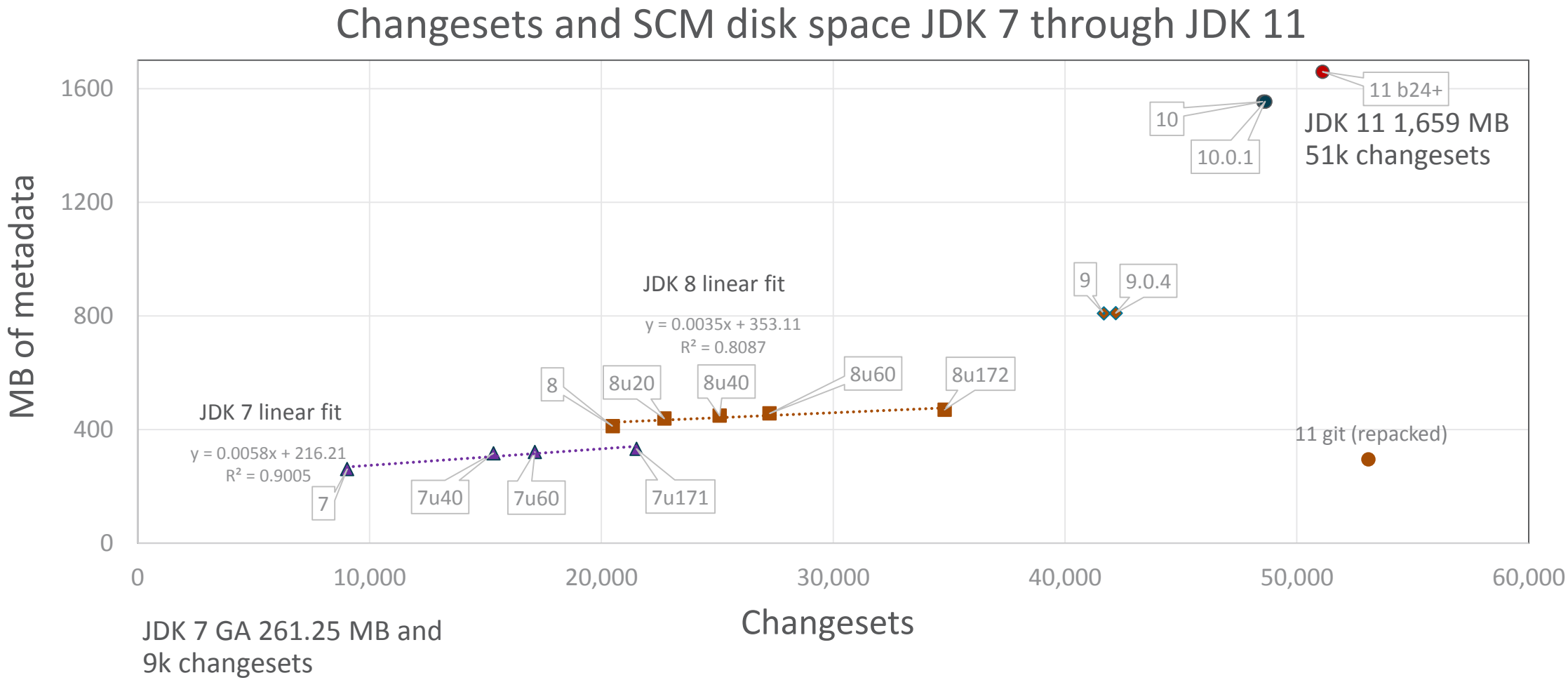
- Hg servers hg.openjdk.java.net and closedjdk.us.oracle.com in production since circa December 2007; JDK 7 b24 first hg-based promotion
- JDK hg functionality stable over the last decade

Growth in size of repos in hg

- When a file is moved, hg restarts the history
  - Source files moved in JDK 9 for modularization (JEP 201)
  - Files moved again in JDK 10 for repo consolidation (JEP 296)
- Unified repo is large and slow download times remain a source of complaints
  - "hg clone is unbelievably slow", jdk-dev, February 2018, Andrew Haley, Aleksey Shipilev, et al.
  - Oracle-internal complaints as well

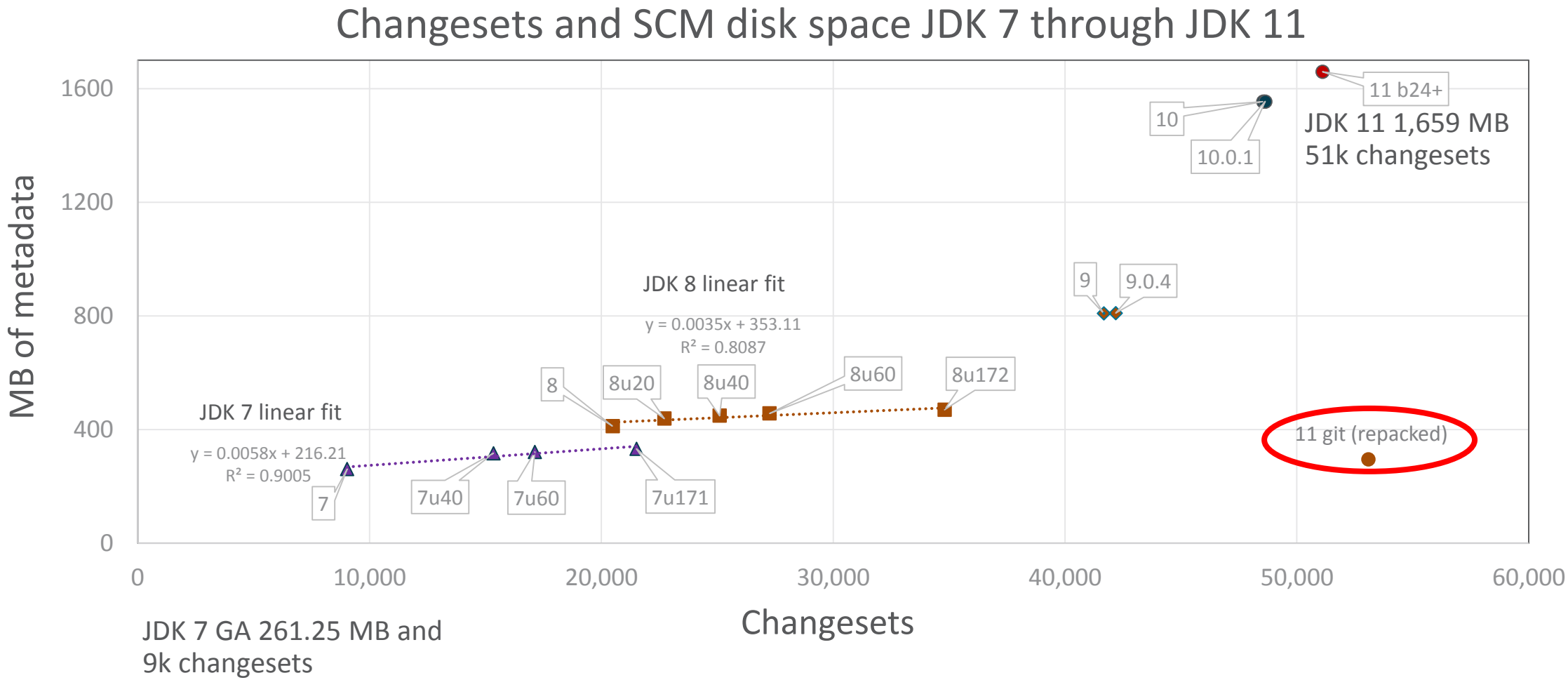
Git repos with full history ≈5X *smaller* than hg.

Changesets and repo metadata size;
E.g.: sum of `du -k --total `find . -name ".hg" -type d`



Changesets and repo metadata size;

E.g.: sum of `du -k --total `find . -name ".hg" -type d`



Repacked JDK 11 git repo

- Naïve hg → git import of OpenJDK is larger than current hg representation; *but* git representation can be compacted
  - `git repack -a -d --depth=250 --window=250 -f` (HT Erik Joelsson) https://metalinguist.wordpress.com/2007/12/06/the-woes-of-git-gc-aggressive-and-how-git-deltas-work/
  - tl;dr – using a larger window size and repacking can remove the file move overhead since git can use both forward and backward diffing
- Radically smaller than current hg representation
  - Includes the *entire* history of OpenJDK ☺
  - Verified to match consolidated hg repos at all tag boundaries

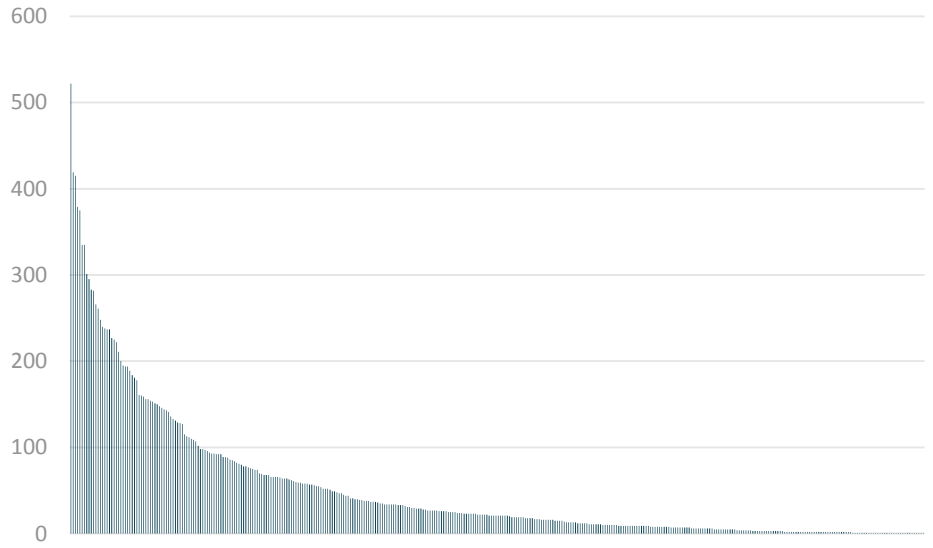
Representative clone times from github are a minute or two for the full OpenJDK; 3X to 30X *faster* than from hg.openjdk.java.net.

Reviewing and committing

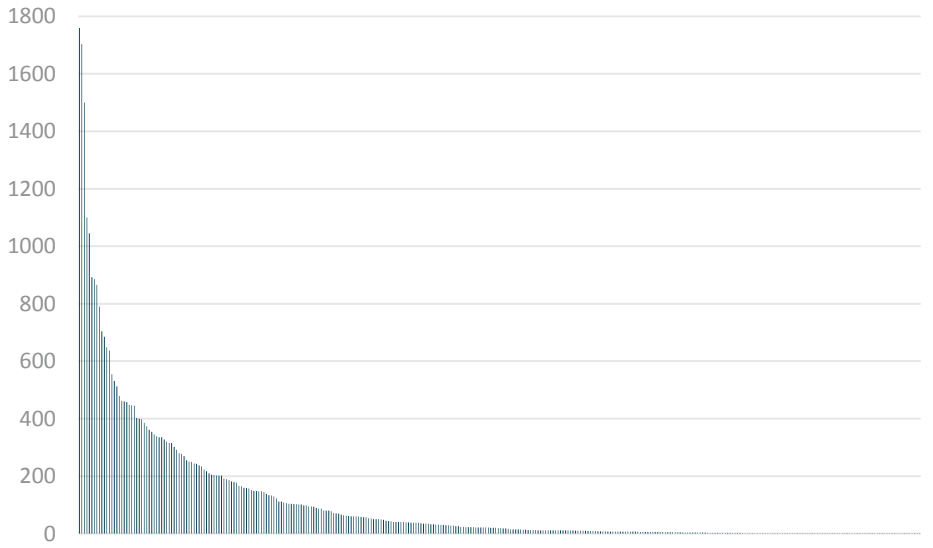
Reviewing and committing distributions

- Since JDK 9 opened:
  - Of developers who have committed, commit counts range over more than two orders or magnitude
  - Of developers who have reviewed, review counts range over more than three orders of magnitude
- Different tooling may be adequate at different activity levels

Committers and reviewers: JDK 9 – present



376 committers with at least one changeset.
Top 10% of committers (37) committed nearly 47% of changesets (9167/19539).
Average number of commits per developer in this time is ≈52.



338 reviewers with at least one review.
Top 10% of reviewers (33) committed over 58% of reviews (21702/37231).
Average number of reviews per developer in this time is ≈110.

Efficient review workflows are important!

What services could bots provide?

- OCA check
- jcheck
- Bug database update
- Build and test status from external build + test system

A civilized transition

"The essence of civilization is being about to benefit from someone else's experience without having to relive it."
—W. Kahan

Experience from the Python community: PEP 512

- PEP 512 -- Migrating from hg.python.org to GitHub

“In 2014, it became obvious that Python's custom development process was becoming a hindrance. As an example, for an external contributor to submit a fix for a bug that eventually was committed, the basic steps were:

1. Open an issue for the bug at bugs.python.org [5].
2. Checkout out the CPython source code from hg.python.org [1].
3. Make the fix.
4. Upload a patch.
5. Have a core developer review the patch using our fork of the Rietveld code review tool [6].
6. Download the patch to make sure it still applies cleanly.
7. Run the test suite manually.
8. Update the *NEWS*, *ACKS*, and "What's New" document as necessary
9. Pull changes to avoid a merge race.
10. Commit the change manually.
11. If the change was for a bugfix release, merge into the in-development branch.
12. Run the test suite manually again.
13. Commit the merge.
14. Push the changes.

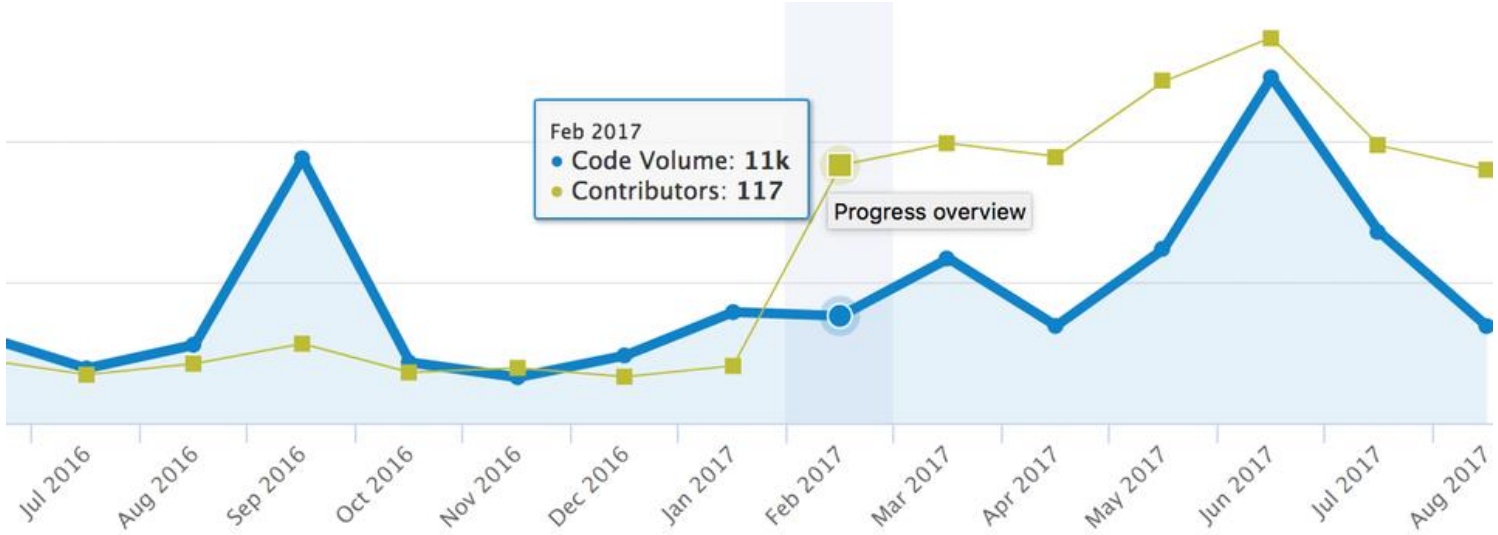
This is a very heavy, manual process for core developers.”

More from PEP 512

- “On January 1, 2016, the decision was made by Brett Cannon to move the development process to GitHub. The key reasons for choosing GitHub were [10]:
- Maintaining custom infrastructure has been a burden on volunteers (e.g., an unmaintained, custom fork of Rietveld [6] is currently being used).
- The custom workflow is very time-consuming for core developers (not enough automated tooling built to help support it).
- The custom workflow is a hindrance to external contributors (acts as a barrier of entry due to time required to ramp up on development process unique to CPython itself).
- There is no feature differentiating GitLab from GitHub beyond GitLab being open source.
- Familiarity with GitHub is far higher among core developers and external contributors than with GitLab.”

Still more from PEP 512

- **Adding GitHub username support to bugs.python.org**
- **A bot to enforce CLA signing**
- Additional background: https://snarky.ca/the-history-behind-the-decision-to-move-python-to-github/
- Moved Jan 2017; how did it turn out? (reddit)



Increasing the likelihood of a smooth transition

Possible future work

- Read-only mirrors on hosting providers;
  gain experience, different commands, etc.
- Command line tools
- Playground repos with bots

Discussion

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.