

PHILIPS

sense and simplicity

Tree conflicts

the problem and steps towards a solution

Nico Schellingerhout

Philips Healthcare

Outline



Background

- Subversion and Medical Software
- Tree conflicts
- Steps towards a solution
- Conclusions

Philips Medical product range

Businesses and products

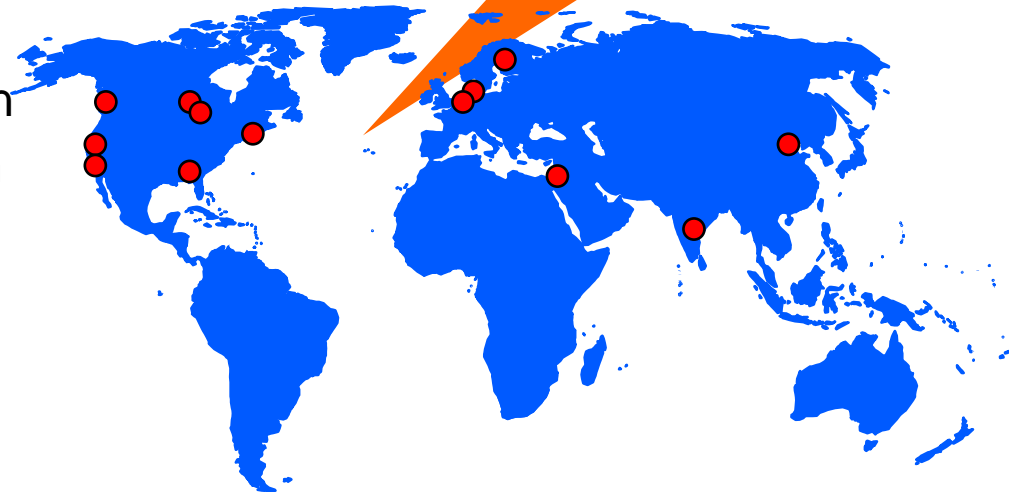
- General X-ray
- Cardio/Vascular X-ray
- Ultrasound
- Computed Tomography
- Magnetic Resonance Imaging
- Nuclear Medicine
- Positron Emission Tomography
- Radiation Therapy Planning
- Cardiac and Monitoring Systems
- Healthcare Informatics
- Customer Services



R&D and Manufacturing Worldwide

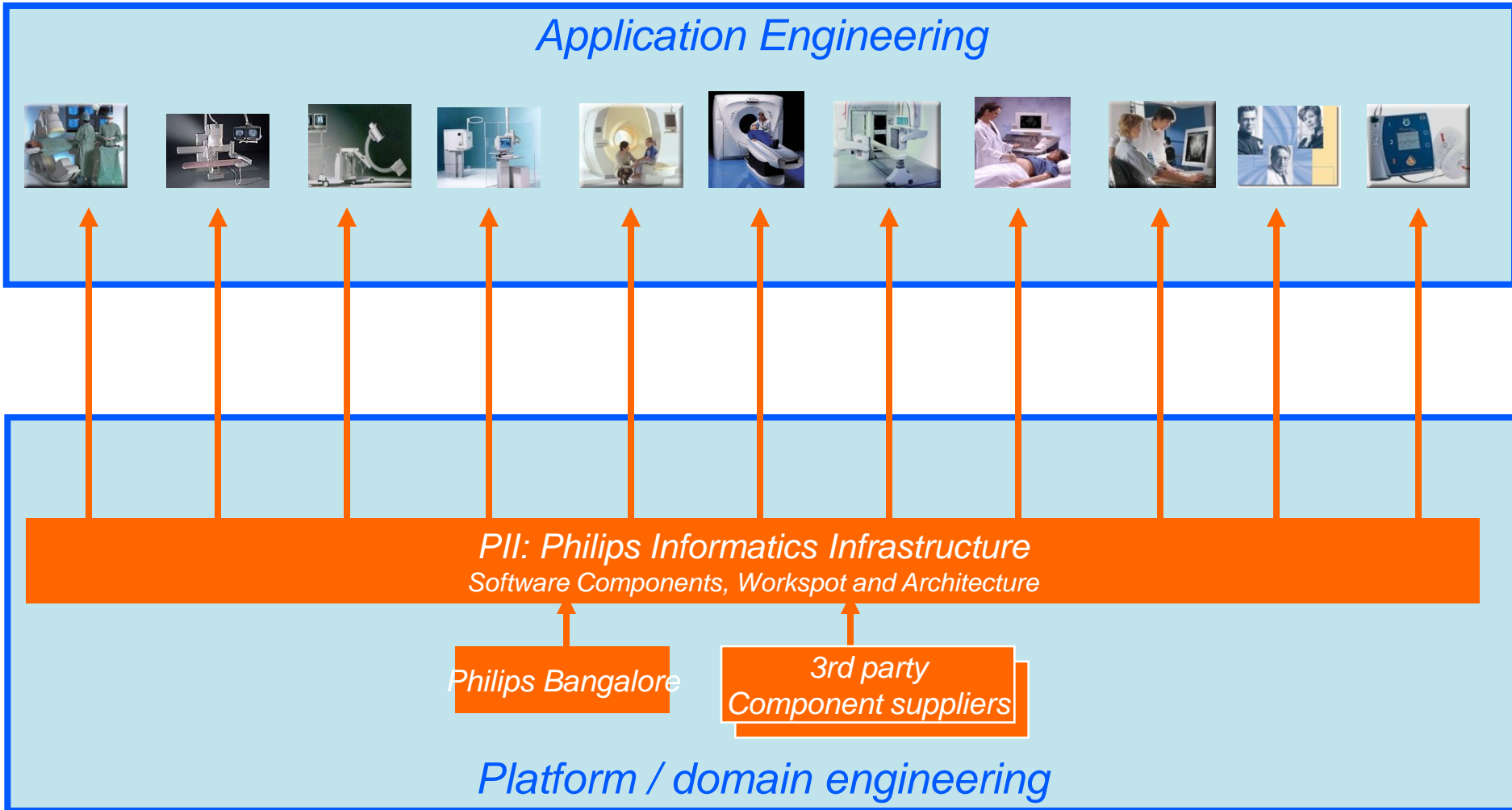
Sites

- USA:
Bothell and Seattle, Washington;
Reedsville and Philadelphia, Pennsylvania;
Andover, Massachusetts;
Milpitas, Oxnard, Foster City and Palo Alto, California;
Cleveland, Ohio;
Chicago, Illinois;
Madison, Wisconsin
- The Netherlands: Best and Heerlen
- Germany: Hamburg and Böblingen
- Israel: Haifa
- Finland: Helsinki
- India: Bangalore
- China: Shenyang



13 of these 20 locations are actively involved in our Product Line software development activities, and on many locations there are multiple sw-development teams

Product line engineering approach

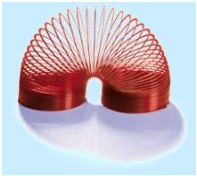


InnerSource development principles



Easy, but controlled, access

- All information must be easily accessible
- Access to information must be controllable



Release early and often

- Flexibility in timing
- Major (and minor) releases
- Snapshots
- Bleeding edge



Distributed Ownership and Control


- The platform team owns and develops components
- The customer is allowed to change components, but is responsible for the change
- The customer can offer the change (patch) back to the platform team
- The platform team can accept the patch into the platform, taking over ownership



Patch rather than workaround

- Avoid wasting effort on glue code that breaks at every new release
- Patches are improvements that may be offered back to the platform

Outline

- Background
-  • The importance of branching & merging
- Tree conflicts
- Steps towards a solution
- Conclusions

The importance of branching & merging

- Many **branches**
 - Medical systems have a **long lifetime**
 - Products *must* be supported for many years (FDA regulations)
 - Products may be upgraded during the lifespan
 - Platform approach
 - Wide variety of products share a single code base
 - **Many versions** of the platform must be **maintained**
- Quality requirements
 - **Solutions** for problems **propagated to newer branches** (raises overall quality of the platform)
 - Solutions for **critical problems propagated to all relevant branches** (**regulatory requirement**)
 - **Merging** essential (efficiency, quality)

Subversion's strong points


- Designed for global/distributed development
 - Optimized for low network traffic
 - Only infrequent contact with repository required
 - Parallel development model
- Traceability support
 - Change-set based:
 - Easy to trace a coherent set of changes (e.g. enables cherry picking)
 - Merge tracking (as of 1.5)
- Easy to branch
- Multi-archive support
 - Linking (externals)
 - Merging

Improvement points

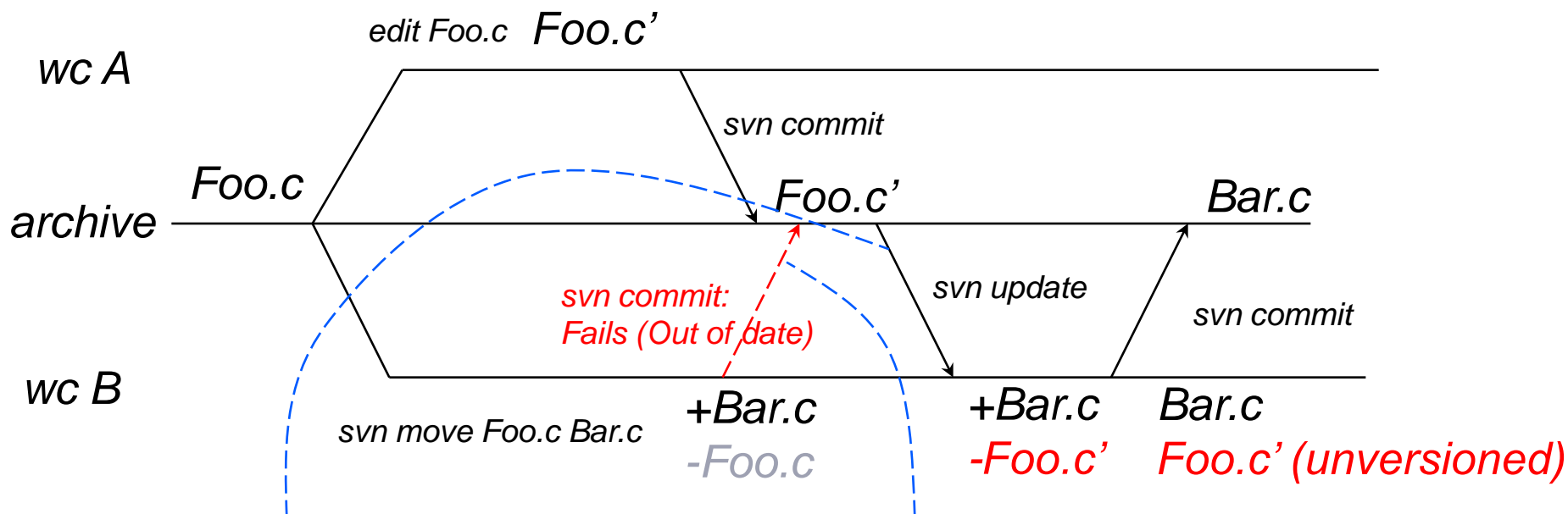
Rename & move related problems

- Tree conflicts
 - Merging does not behave as expected
 - Changes may be “lost”
 - Resolution is error prone and time consuming
- Other problems
 - Case change problems
 - Move back and forth
 - Partial moves

Outline

- Background
- Subversion and Medical Software
-  Tree conflicts
- Steps towards a solution
- Conclusions

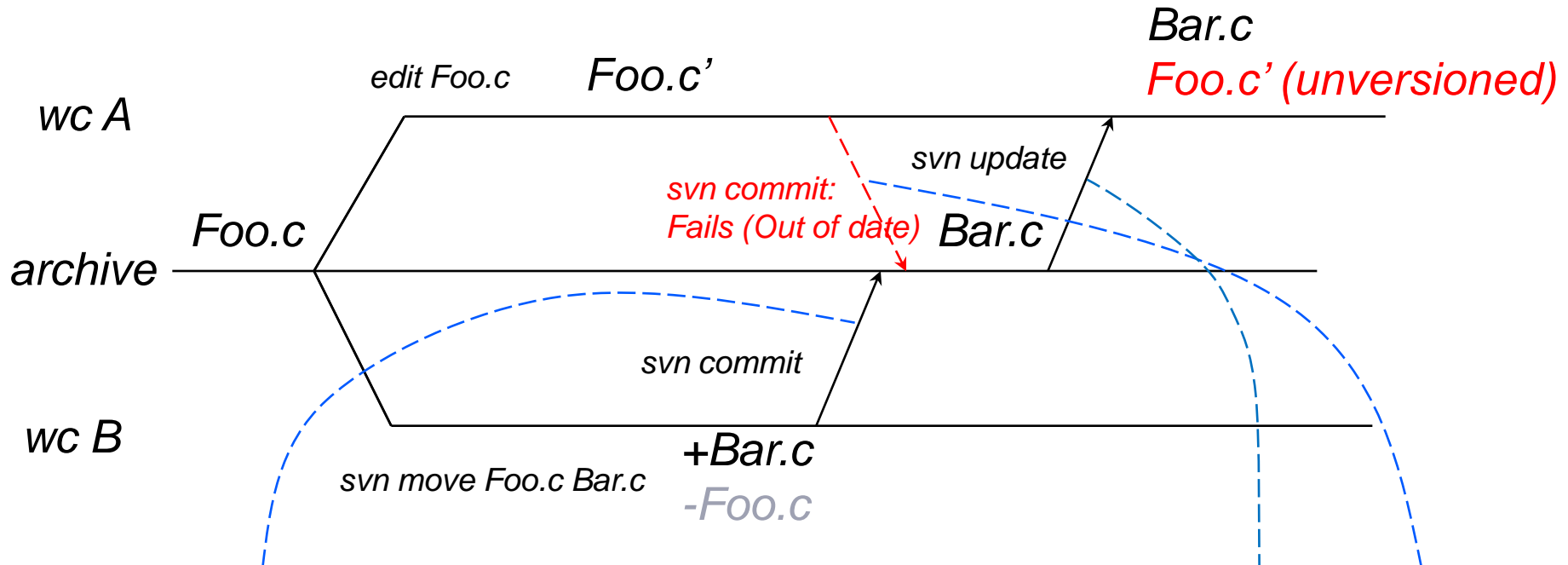
Simultaneous change: Modify before move



Assessment:

- ☹ B's changes are scheduled for deletion without warning
 - Changes left **unversioned** in WC of B
 - Changes removed from archive (**no notification to A**)
- ☺ Statistically **unlikely** (simultaneous edit & move happen infrequently)

Simultaneous change: Move before modify

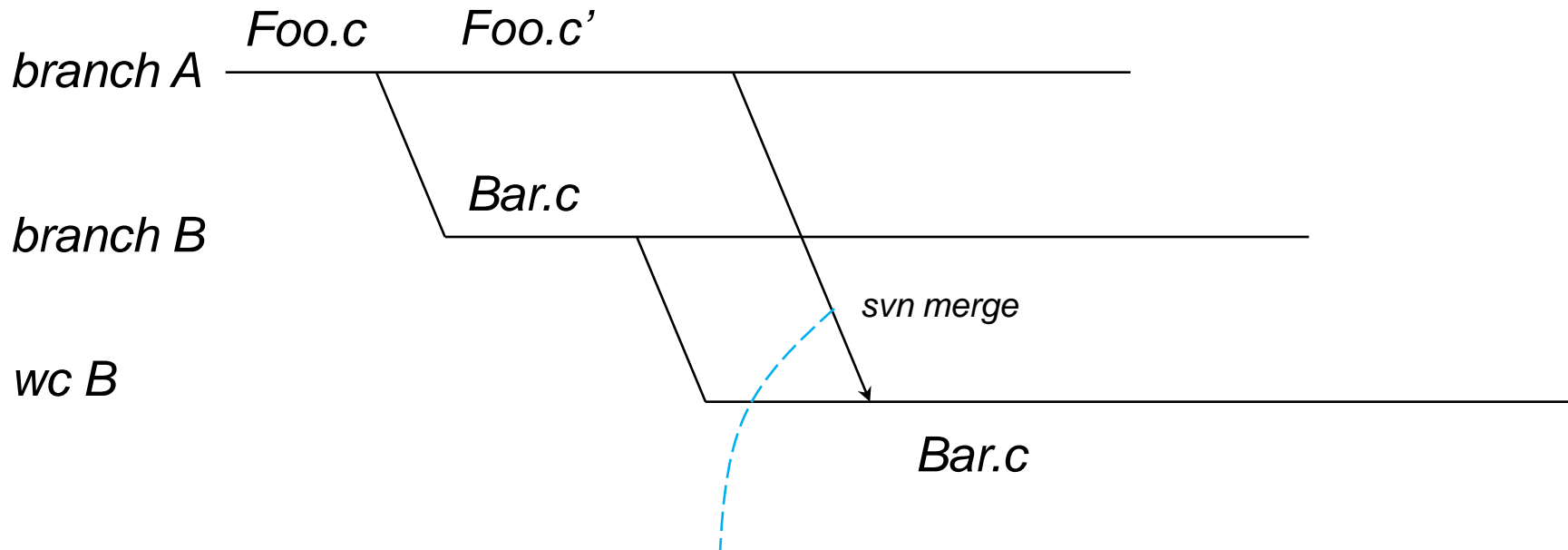


Assessment:

- ☹️ If A misses the fact that his changes were deleted by the update
 - Changes left **unversioned** in WC of A
 - Changes do not make it in the archive
- 😊 Statistically **unlikely** (simultaneous edit & move happen infrequently)

Merge: Modify onto move

a.k.a. "Maintenance use case"

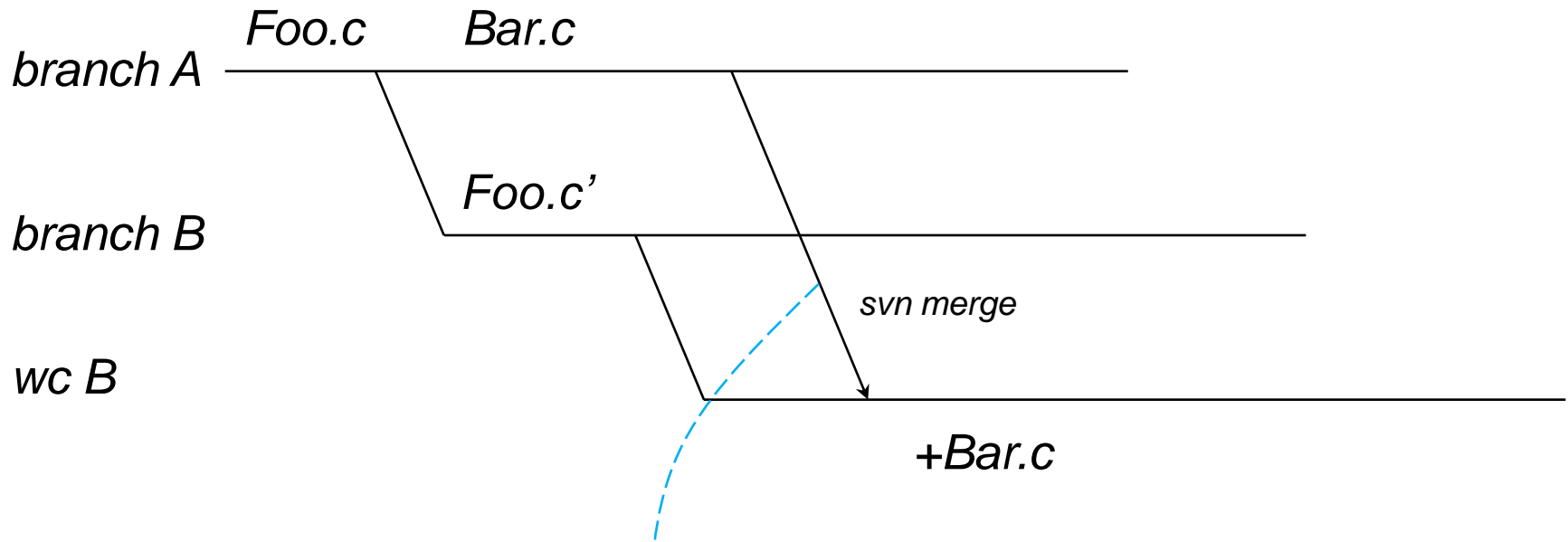


Assessment:

- ☹ If B misses the “skipped missing target” message
 - Changes are **not merged** to branch B
- ☹ If B notices the problem
 - B has to find the correct element (from svn log) and merge “by hand”
- ☹ Statistically **very likely** (happens for every merge onto a moved file)

Merge: Move onto modify

a.k.a. "Parallel development use case"




Assessment:

- ☹ B is **very likely** to miss this problem
 - Changes in branch B **removed** by merge
- ☹ If B notices the problem
 - B has to find the correct element (from svn log) and merge "by hand"
- ☹ Statistically **likely** (conflicting changes do not need to be simultaneous)

The impact of tree conflicts

- The platform team in Philips Healthcare
 - deals with **huge** and **long-living** archives
 - refactors continuously to adapt to changing requirements while maintaining high code quality
 - Virtually **every file** in the archive is **moved** at some time
 - merges many changes between many branches (every day)
 - If the tree conflict problem remains unsolved, it
 - leads to avoidable errors being introduced in the archive
 - forces a very centralized merging strategy
 - is very costly (both financially as well as motivation-wise)
- Tree conflict handling **essential**

Outline

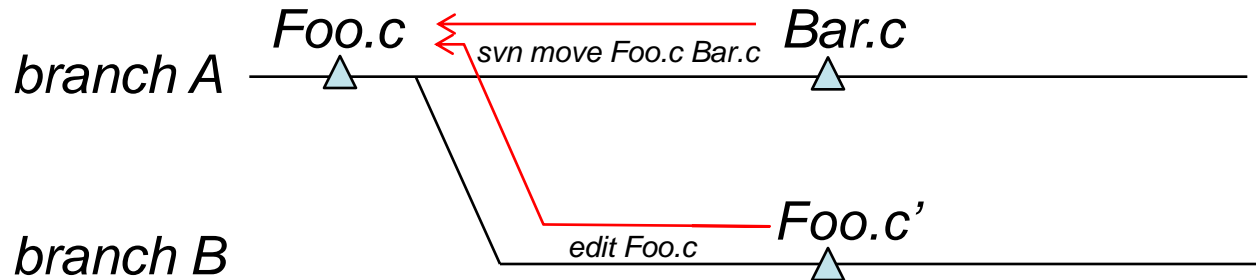
- Background
- Subversion and Medical Software
- Tree conflicts
-  Steps towards a solution
 - Tree conflict detection and raising
 - Tree conflict resolution (truMerge)
- Conclusions

Steps towards a solution (I): *Tree conflicts detection and raising*

- Tree conflict concept introduced in Subversion
 - Joint effort by Philips, CollabNet, Elego and Subversion community
 - Planned for release in Subversion 1.6
 - Benefits
 - Detection of tree conflicts during update, switch, and merge
 - Conflicts are raised persistently (much like textual merge conflicts)
 - Warns users of potential problems; prevents most common errors
 - Limitations
 - No automatic resolution of “trivial” tree conflicts
 - Manual resolution is inefficient, error prone
- Be sure to catch [Stephen Butler & Stefan Sperling's talk on tree conflict detection!](#)

Steps towards a solution (II): *Identity tracking*

- Tree conflicts would go away if Subversion would correctly identify the “same” element on different branches:



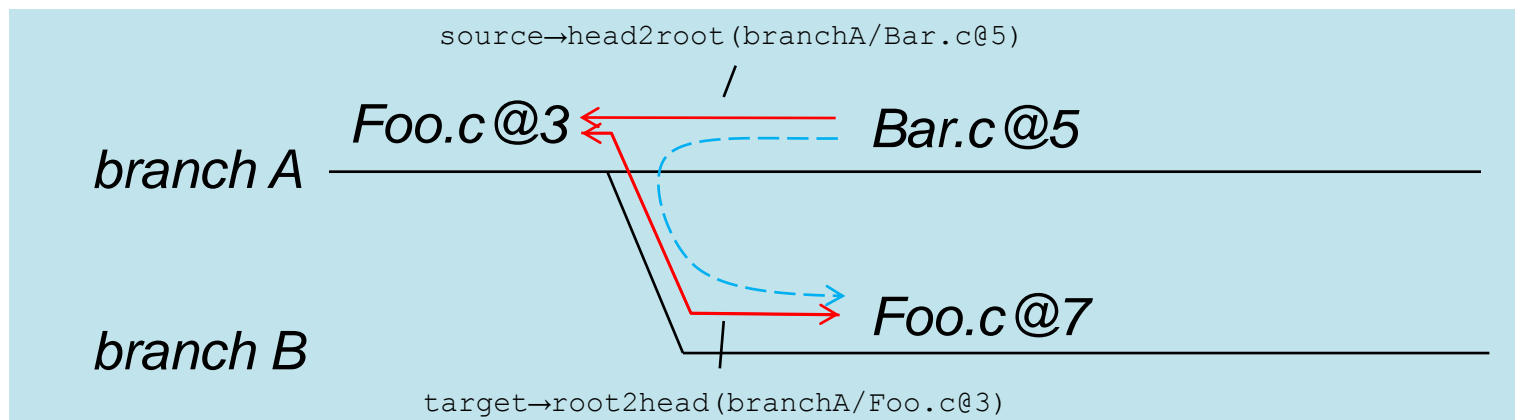
- In this example: [branchA/Bar.c](#) matches [branchB/Foo.c'](#)
- Can be achieved by assigning a [globally unique ID](#) to every element
- In Subversion there is [no explicit GUID](#)
- Still, Subversion [keeps sufficient metadata](#) to do the work

Matching elements between branches

- Basic idea:
 - Every element in an archive can be identified by the following **root**:
 - Creation path
 - Creation revision
 - This identity is *invariant* under copy (as well as under branching)
- Potential problems
 - The root is not a true GUID:
 - Copying an element results in two elements sharing the same root
 - In practice, this happens **very rarely**
 - Traceability can be broken by poor usage:
 - Using: **cp a b; svn add b; svn delete a**
instead of: **svn move a b**
 - **Happens from time to time**; by mistake or ill-informed developers

Implementation of identity tracking

- “svn log -v” produces all information needed to calculate an element’s root
- However, we need the **reverse mapping** as well

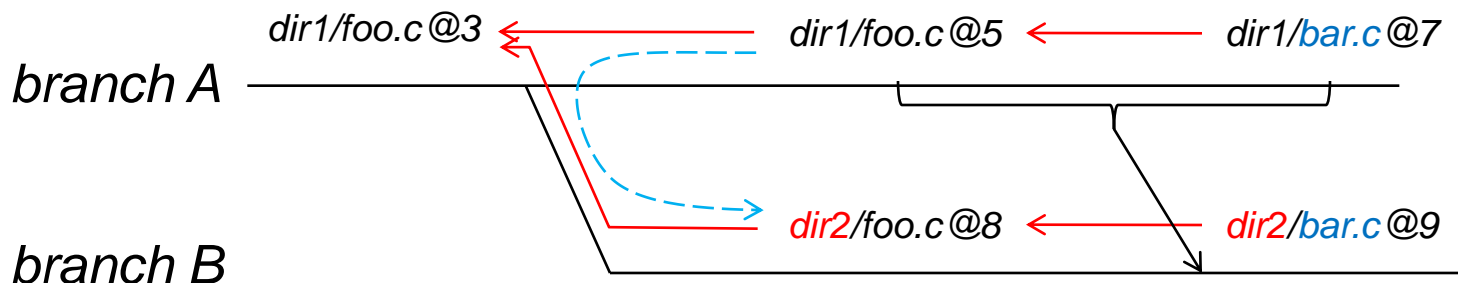


- Calculating the reverse mapping is prohibitively expensive
- Performance enhancement: **store hash maps in cache files**
 - Finding the heads related to a root is **very fast**
 - Initial construction (one-time preparation per branch) is **slow**
- Pathname-similarity heuristics as fallback
 - Multiple target candidates
 - No target candidates

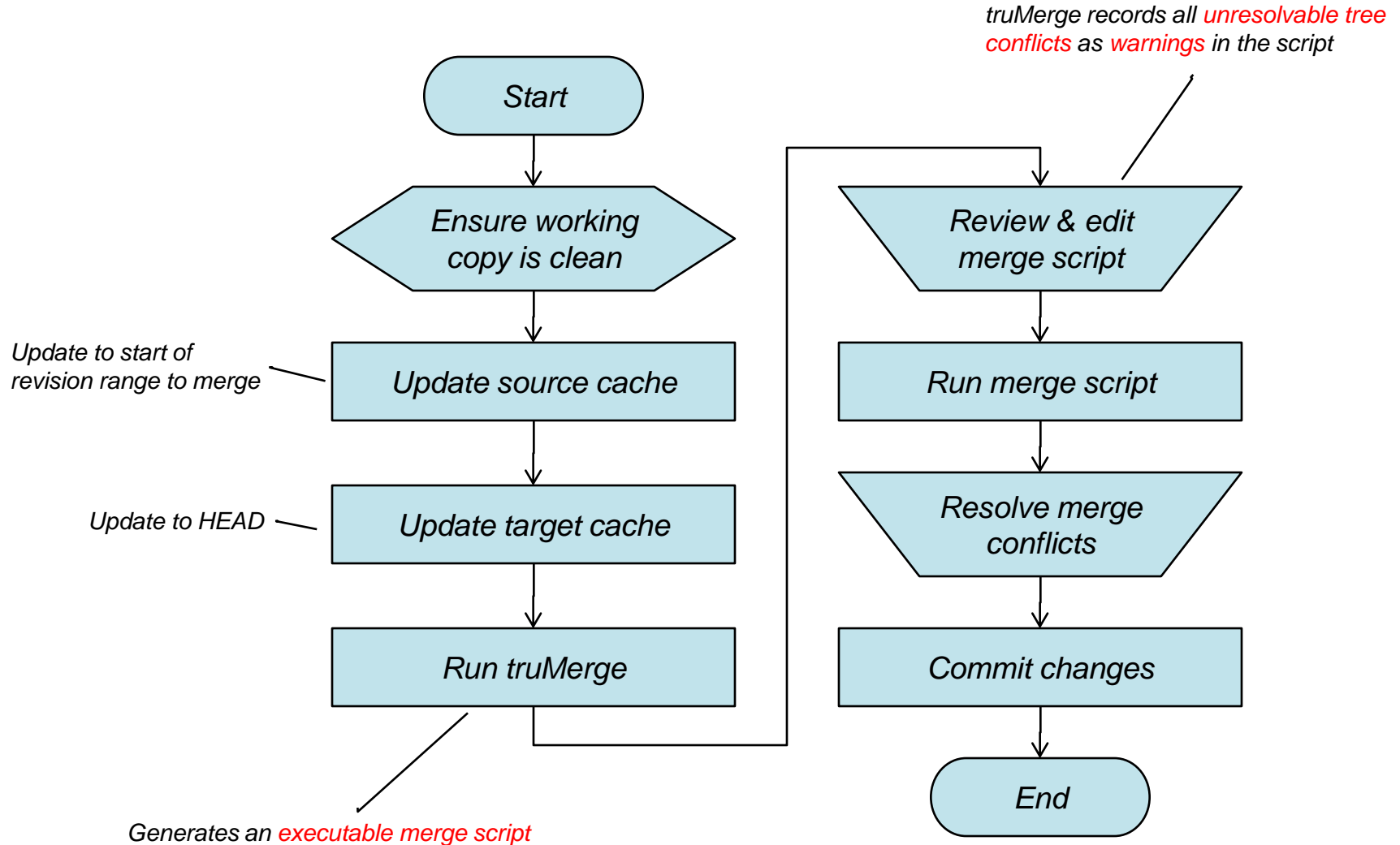
Steps towards a solution (III):

truMerge: true rename capable merge helper tool

- Uses identity tracking to find corresponding elements
- Analyzes changes on source branch in the given revision range (adds, deletes, moves, modifications)
- “Replays” changes on target branch
 - add → copy to **corresponding parent**
 - delete → delete **corresponding** element
 - move → “**similar**” move
 - Moves can be traced entirely on the target branch
 - modifications → textual merge between **corresponding** elements



How to use truMerge



truMerge's past, present and future

- truMerge was developed by Philips Healthcare
 - Is in daily active use
 - Provides significant productivity and quality improvements
- truMerge is Open Source!
 - URL: <http://trumerge.open.collab.net/>
 - Runs on Windows XP, written in Perl
- You are welcome to:
 - Use it
 - Discuss it
 - Report problems
 - Improve it

Outline

- Background
- Subversion and Medical Software
- Tree conflicts
- Steps towards a solution

 **Conclusions**

Conclusions

- Subversion is in use at Philips Healthcare, supporting
 - globally distributed development
 - the overall product-line approach
 - the InnerSource way-of-working
- Tree conflicts are a **serious problem**
 - When not detected: can lead to **errors**
 - When detected: **costly** and tedious to solve
- Steps towards a solution
 - Tree conflict detection and raising (slated for 1.6)
 - Be sure to catch Stefan Sperling's/Steve Butler's talk!
 - **truMerge** for automatic tree conflict resolution
 - get it at: <http://trumerge.open.collab.net/>

PHILIPS

*Thank you for
your attention.
Questions?*



**Now being in a hospital doesn't
feel like being in a hospital.**
Philips Ambient Experience.

PHILIPS
sense and simplicity

