Container-Based SCM and Inter-File Branching

A Promising Model Meets A Capable Technology

Laura Wingerd
Perforce Software

Overview

- Container-based software configuration management
  - What is it?
  - Why is it useful?
- Inter-File Branching
  - What is it?
  - How is it different from conventional file branching?
- Applying Inter-File Branching to container-based SCM

The complexity problem

- In the real world, nobody makes only <u>one product</u>
  - A product is usually composed of many components
  - Components are usually products themselves
- A product can be reused and customized for many targets (customers, platforms, applications)
- Components x targets = many configurations to keep track of

An analogy

- Family car trip

  - Parents have to know the needs of each child and pack the car accordingly
  - Parents have to keep track of where everything is in the car

An analogy

- Bus trip for adults
  - Each participant gets a trip description, packs accordingly, and shows up for the bus
  - Participants make sure their luggage is on board
  - Tour guide makes sure participants are on board

What is container-based SCM?

- Individual configuration items are "packed" into containers

- Containers, not individual items, are tracked and manipulated

Properties of SCM containers

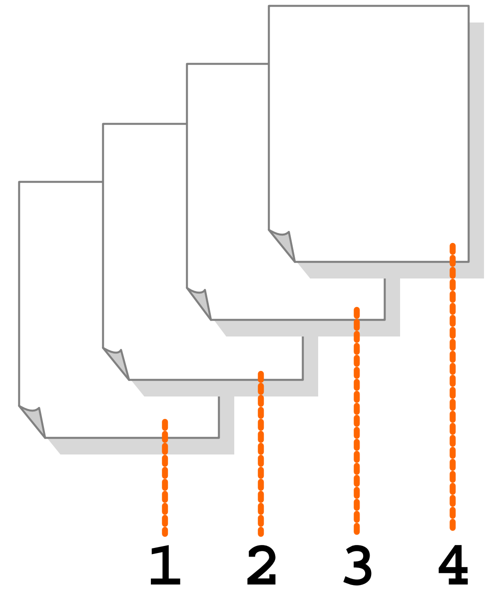
- Items are individually accessible and modifiable
- Items can be moved between containers
- Containers can contain other containers
- A container's state identifies the state of each of its items
- A change to an item changes the state of its container
- *Containers have the same SCM behaviors as individual items*

“Individual items” = “files”

- Files are the individual items of SCM
- Files have SCM behaviors that support:
  - parallel development
  - reproducibility
  - defect resolution

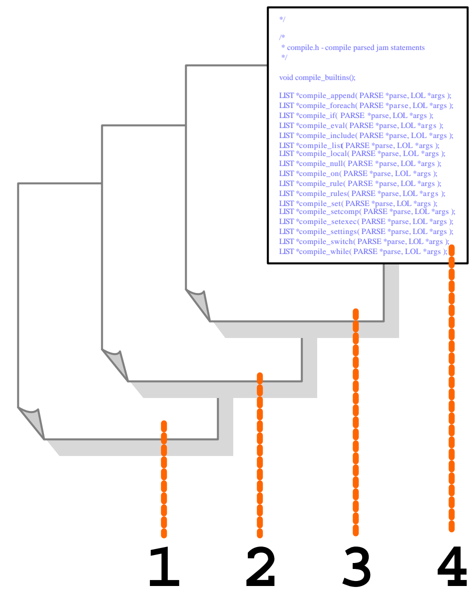
SCM behaviors of files

- A file evolves as a sequence of versions



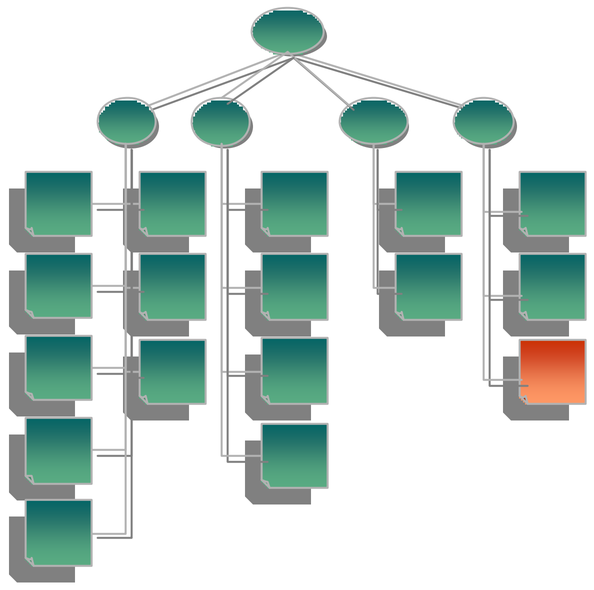
SCM behaviors of files

- A file evolves as a sequence of versions
- A file version identifies a known state of file content and attributes



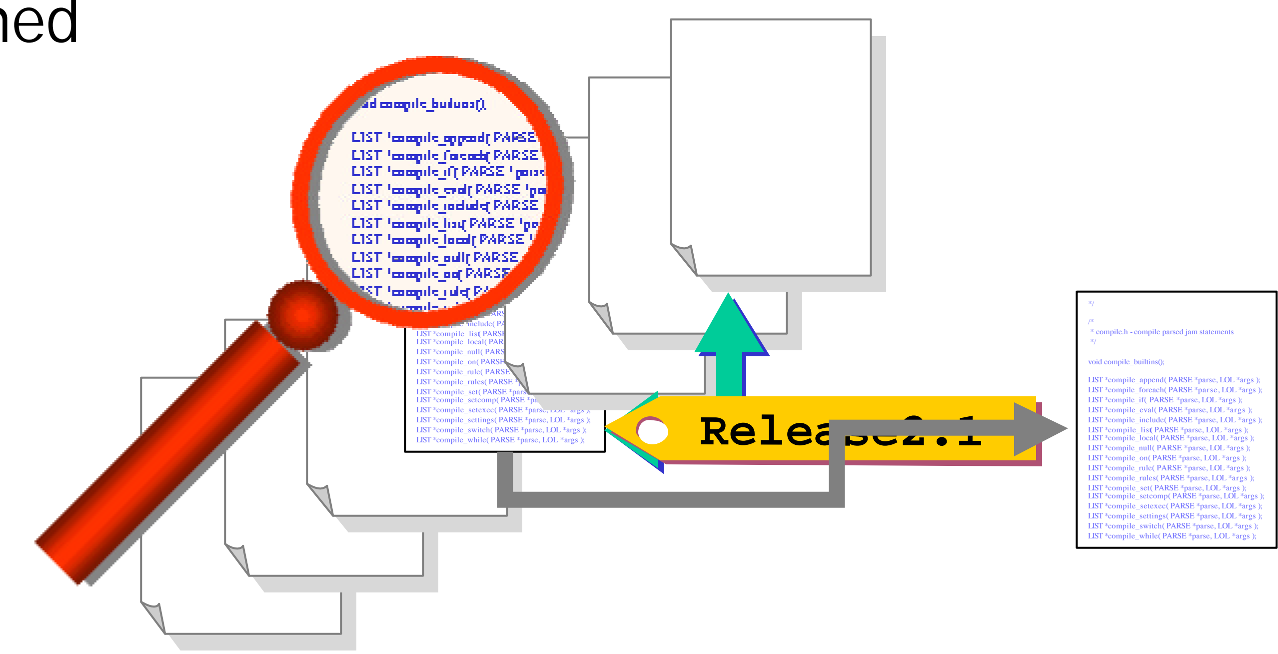
SCM behaviors of files

- A file evolves as a sequence of versions
- A file version identifies a known state of file content and attributes
- Files have relative locations in a path hierarchy



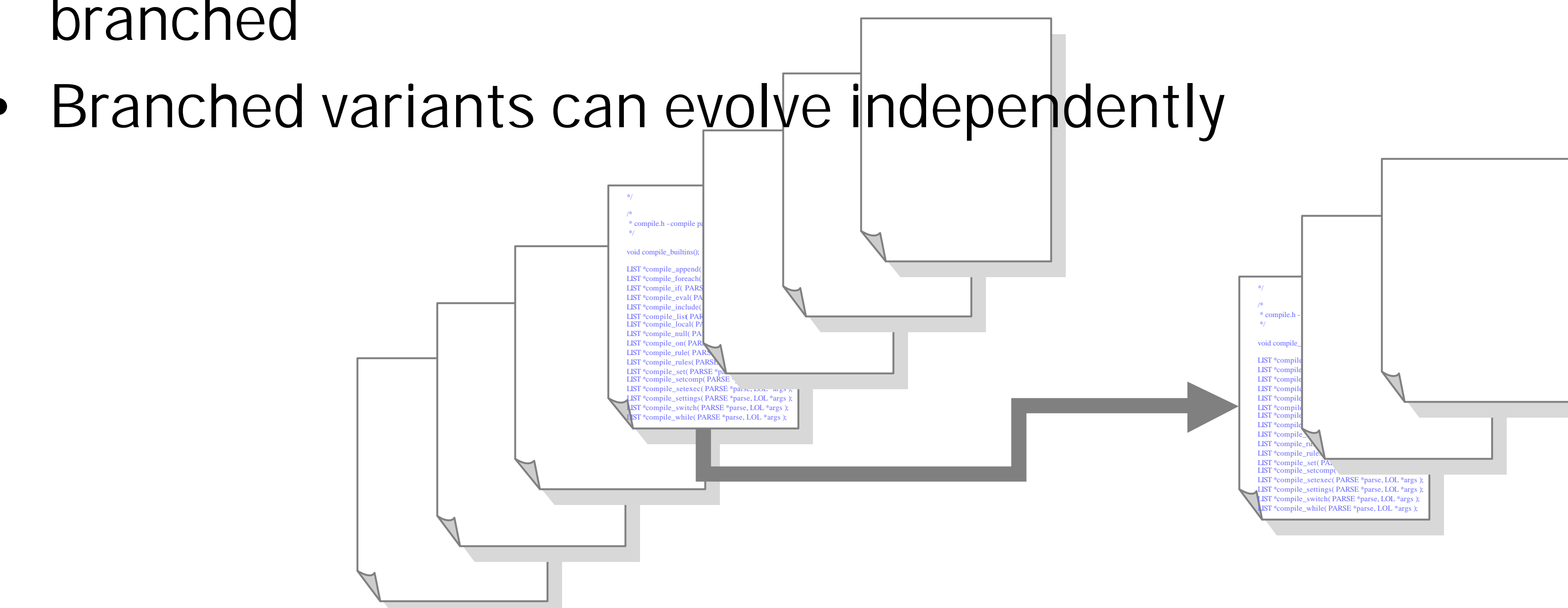
SCM behaviors of files

- File versions can be inspected, compared, labeled, and branched



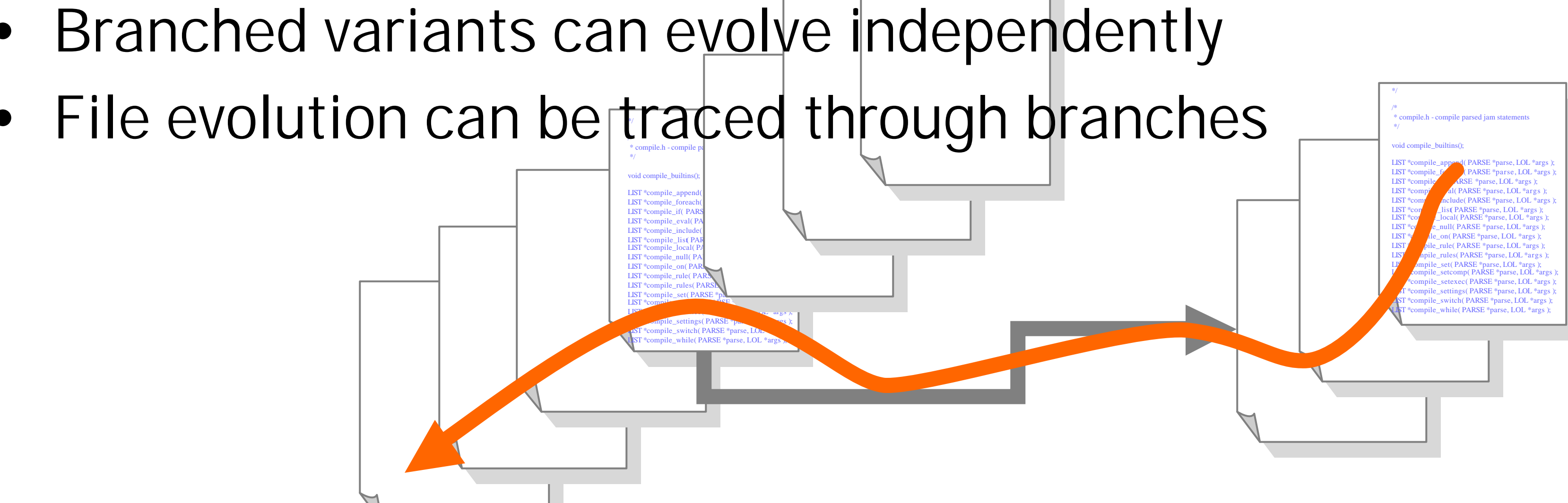
SCM behaviors of files

- File versions can be inspected, compared, labeled, and branched
- Branched variants can evolve independently



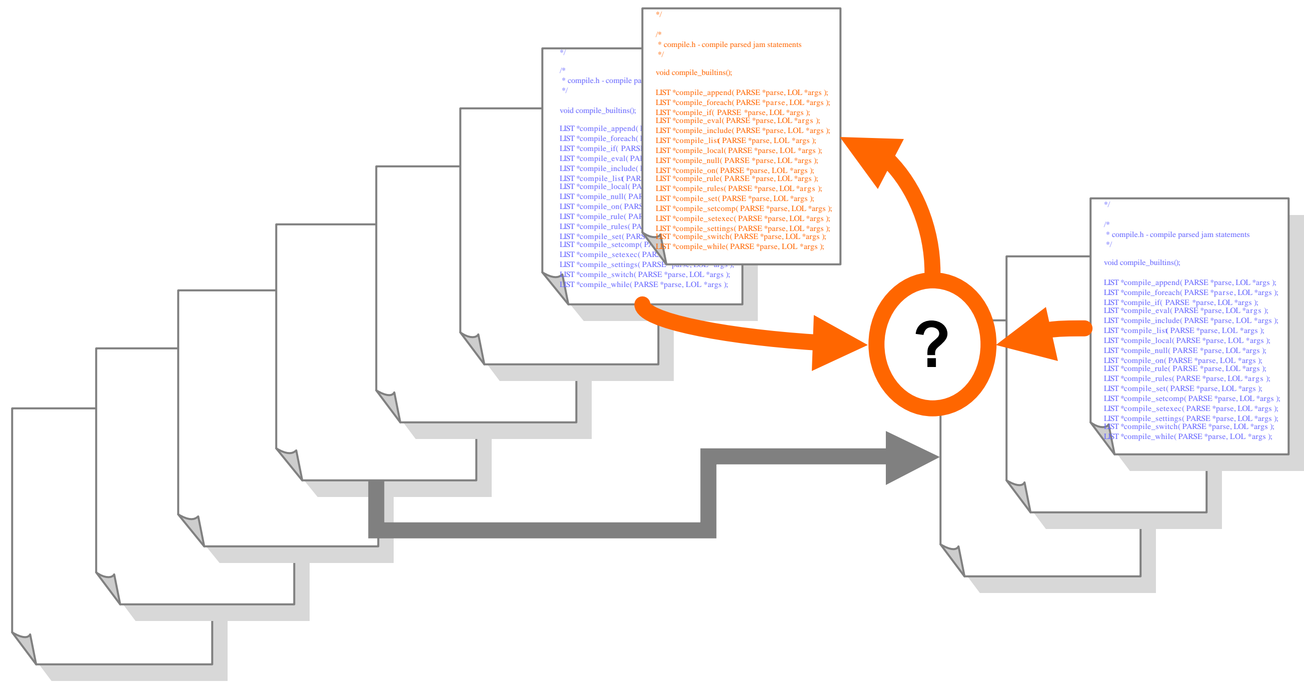
SCM behaviors of files

- File versions can be inspected, compared, labeled, and branched
- Branched variants can evolve independently
- File evolution can be traced through branches



SCM behaviors of files

- Variant content can be compared and merged

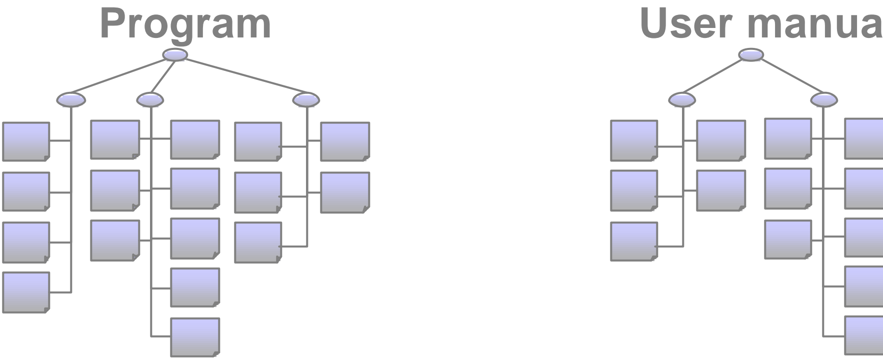


Components and streams

- Components and streams are types of SCM containers. They support:
  - parallel development
  - reproducibility
  - defect resolution
  - reuse
  - customization
  - configurability

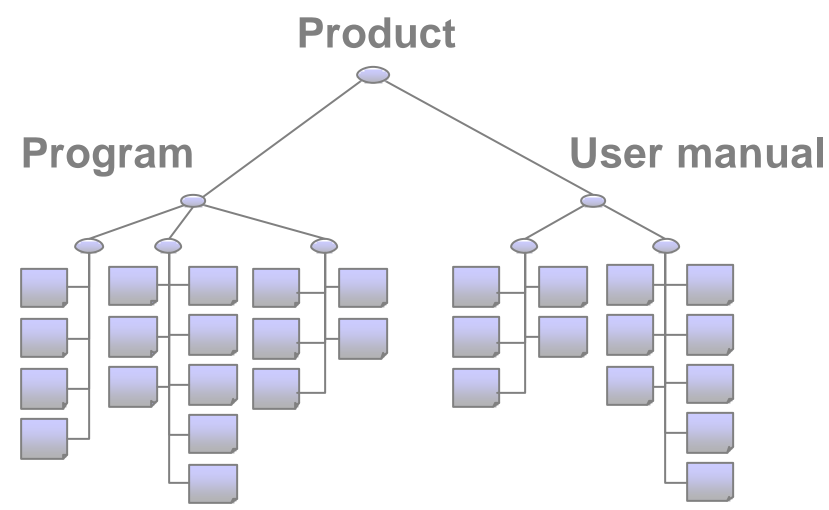
Components

- Components contain files grouped together for a common function or use. For example:
  - C++ source files that are compiled & linked together into an executable program
  - Files that comprise a user manual (e.g. HTML files)



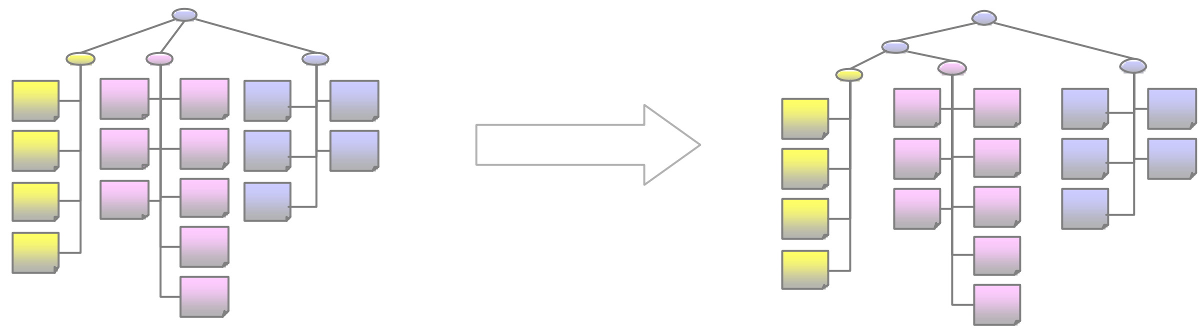
Components

- Components can contain other components. For example:
  - An executable program component and a user manual component can form a product component



Components

- Files can be modified within components
- Files and components can be rearranged



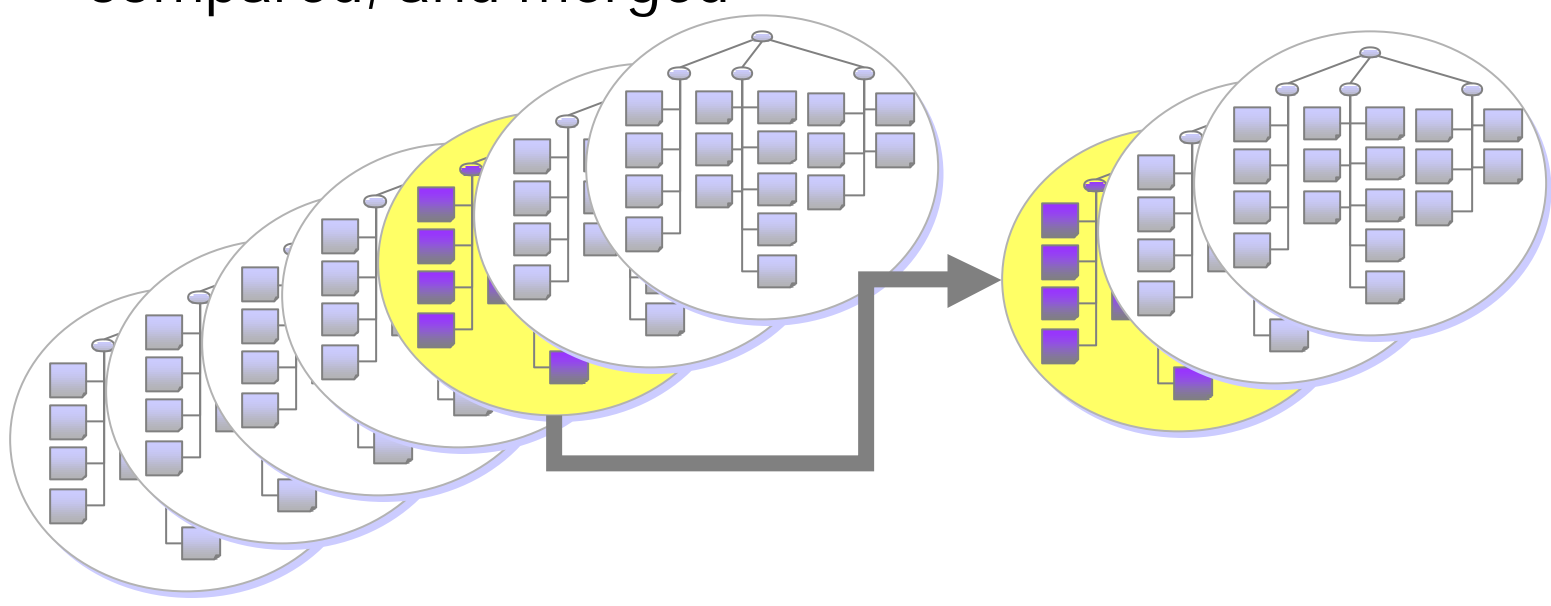
Components

- Components evolve as a sequence of versions
- A new version of any file in a component constitutes a new version of the component
- A component's revision identifier identifies the version of any file it contains

r1 r2 r3 r4

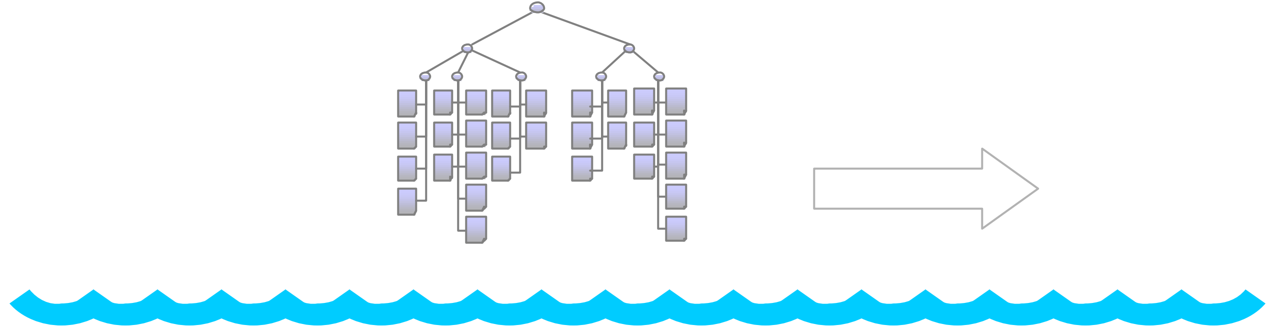
Components

- Components have a history
- Components can be inspected, labeled, branched, compared, and merged



Streams

- <u>Streams</u> contain components managed together because they're passing through the same evolutionary stage.



Streams

- Streams evolve as a sequence of versions
- A new version of any file or component in a stream constitutes a new version of the stream
- A stream's revision identifier identifies the version of any file or component it contains
- Streams can be inspected, labeled, compared, branched, and merged

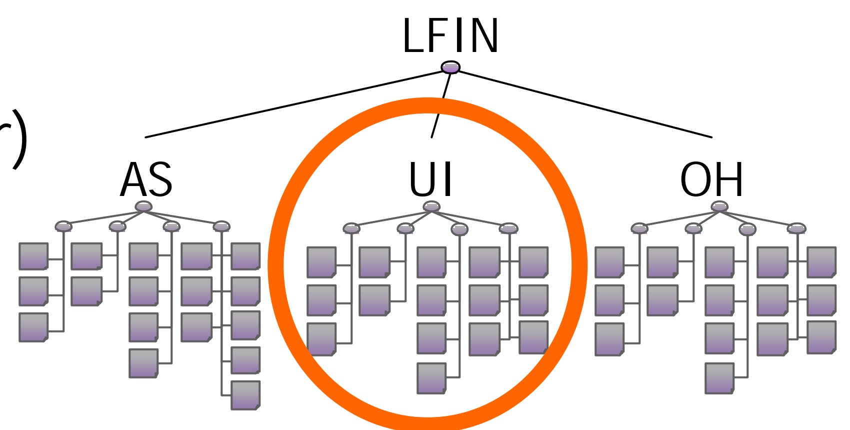
r1 r2 r3 r4

A container-based SCM scenario

- Company: LHC (Large Hypothetical Corporation)
- Product: LFIN (Large Financial Application)
  - Customer-specific variants (corporate logos, etc.)
  - Locale-specific variants
- Customers:
  - MonoBanque (Euro, French)
  - BigBank (USD, English)

A container-based SCM scenario

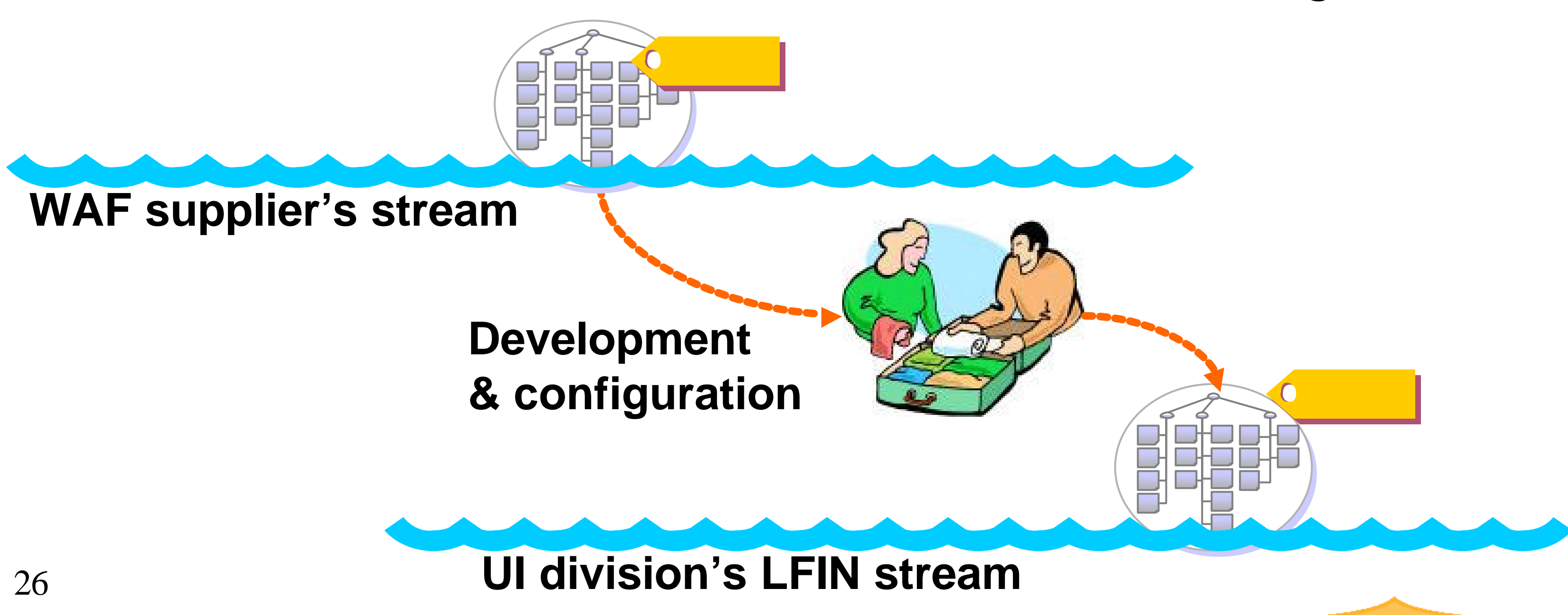
- LFIN components:
  - AS (web application server)
  - UI (user interface)
  - OH (online help system)



- UI subcomponents:
  - WAF (general-purpose windowing app framework)
  - LFIN-specific UI logic
  - Locale-specific modules
  - Customer-specific graphics & skins

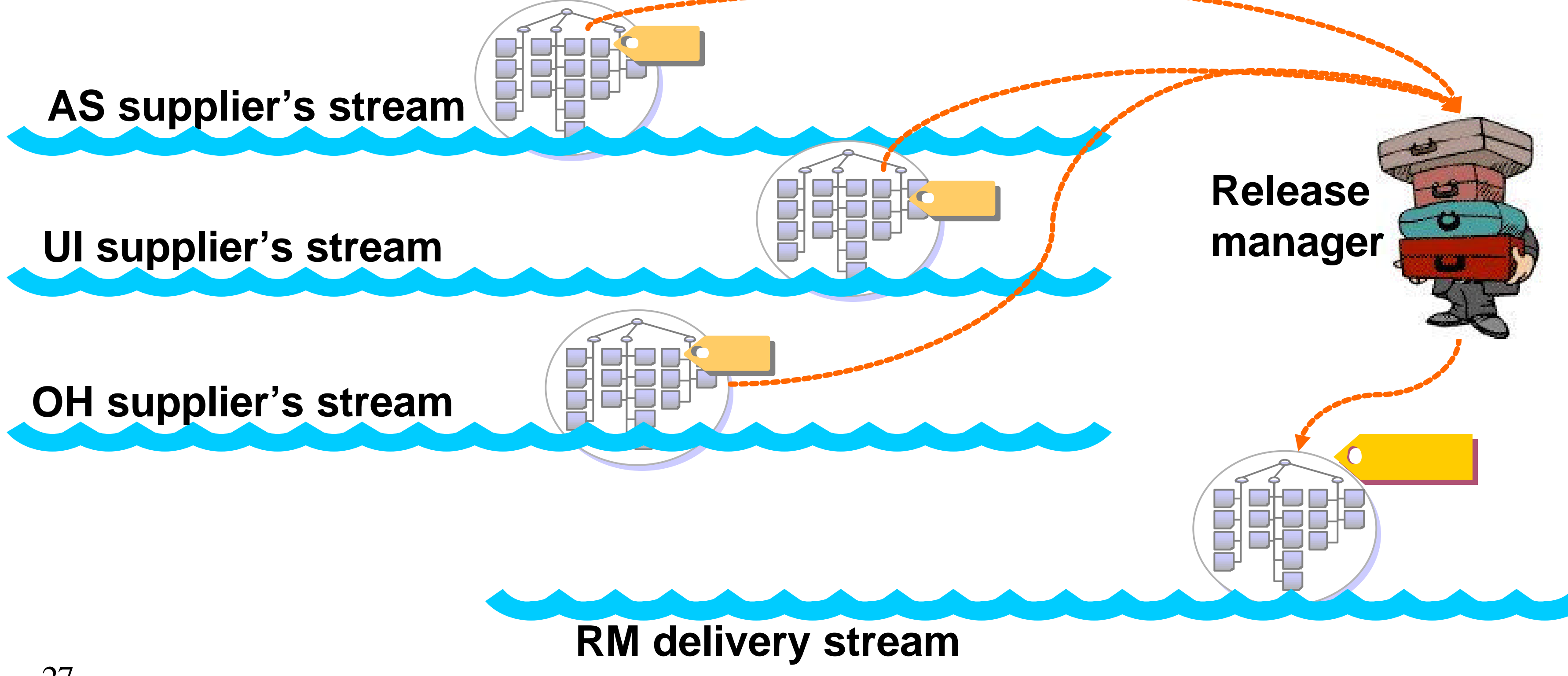
A container-based SCM scenario

- What the UI division do:
  - Get a WAF version from supplier's stream
  - Develop & configure LFIN UI per specs
  - Deliver a version of LFIN UI to release manager



A container-based SCM scenario

- Release manager assembles, packages, tests, & delivers LFIN for each customer



What is Inter-File Branching?

- Compared to int*ra*-file branching
- Side effect: aggregated file history

Traditional (in*tra*-file) branching

- A branch is a version of a file
- File revision number identifies branch
- Labels can be used to name branches

```
/www/index.html 1.1
/www/index.html 1.2
/www/index.html 1.3
/www/index.html 1.3.1.1
/www/index.html 1.3.1.2
/www/index.html 1.3.1.2.1.1
/www/index.html 1.1.1.1
```

Inter-File Branching

- Branched files are peers, not versions, of their originals
- Branched files have different names from originals
- Paths can be used to name branches

```
/www/test.html 1
/new/index.html 1
/www/index.html 1
/www/index.html 2
/www/index.html 3
/dev/index.html 1
/dev/index.html 2
```

Inter-File Branching

- Branch and merge history stored in metadata
- Merging occurs between files, not within a file

/www/test.html 1

/new/index.html 1
/new/index.html 2
/new/index.html 3

tested

/www/index.html 1
/www/index.html 2
/www/index.html 3
/www/index.html 4

Rel2.1

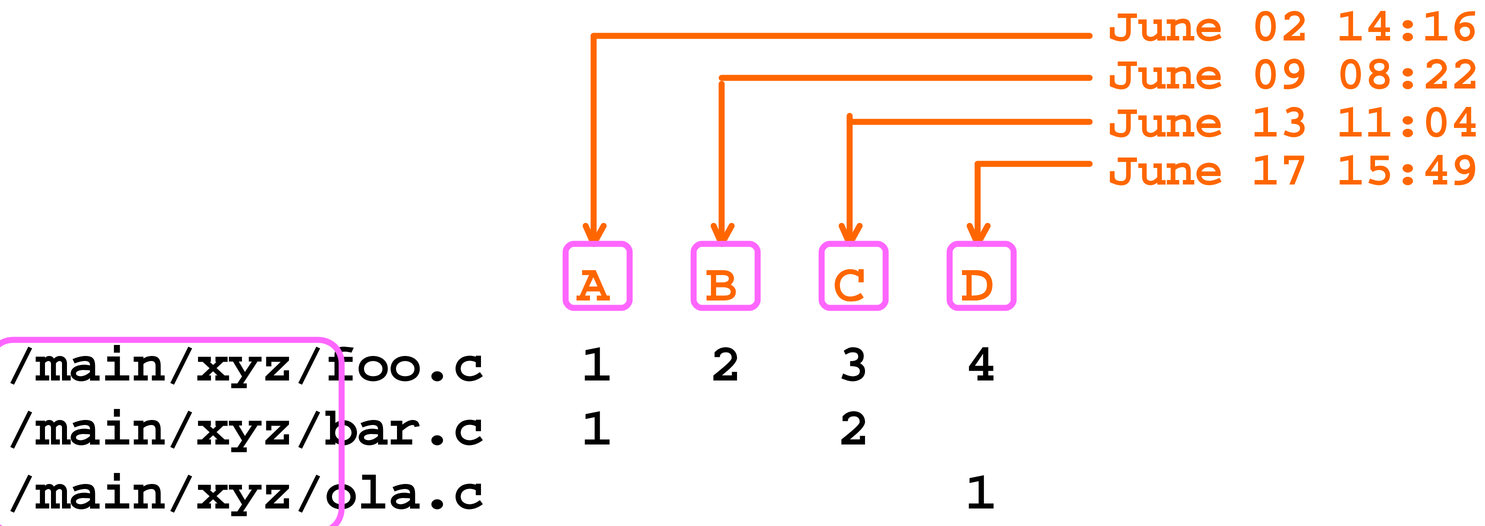
/dev/index.html 1
/dev/index.html 2
/dev/index.html 3

beta - 1

- Labels can be applied to any file revision

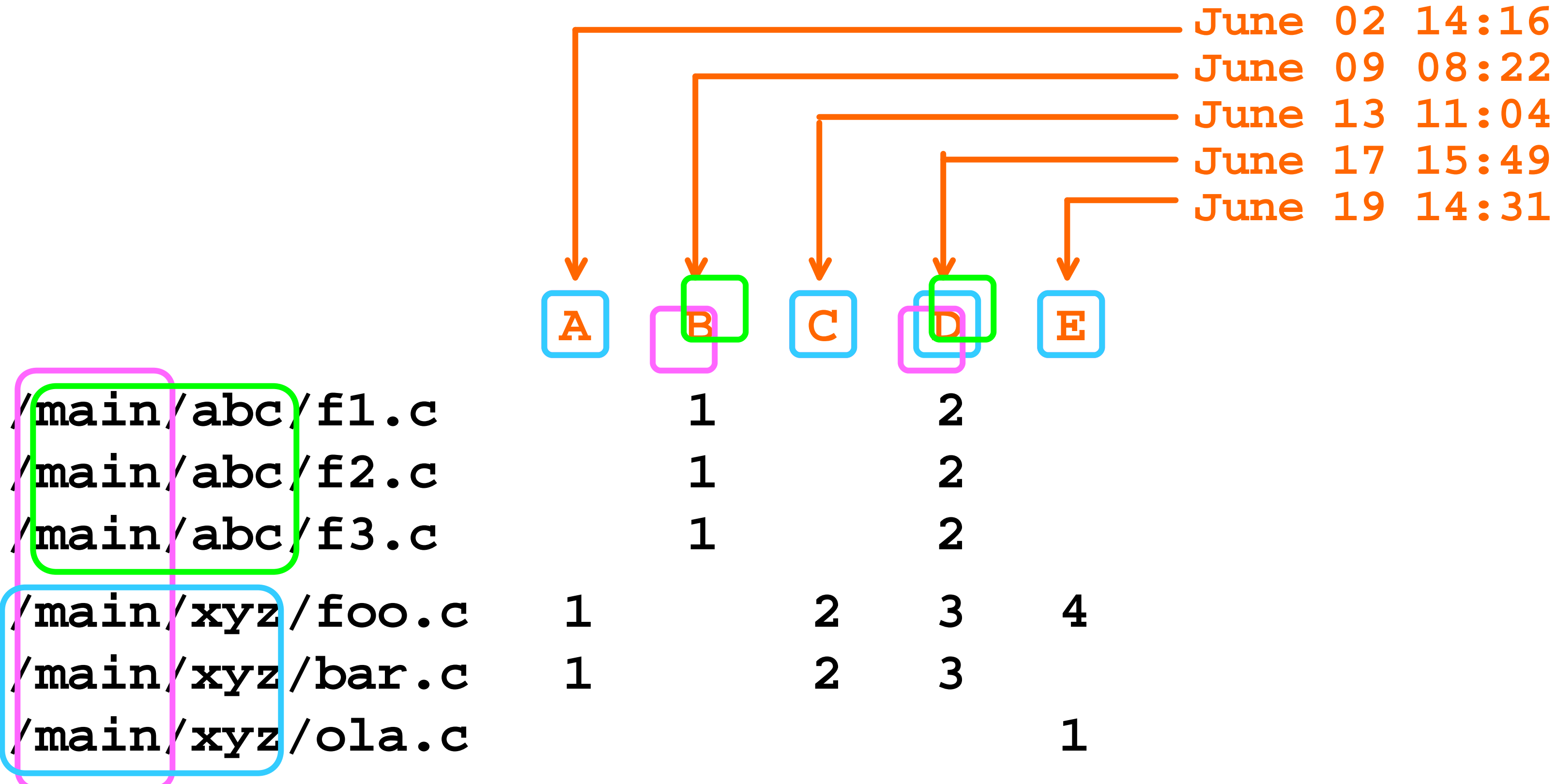
Aggregating file history

- Aggregating the history of files in a repository path yields the history of the path itself



Aggregating file history

- Every path level has its own history



So what?

- Inter-File Branching and aggregated file history bestow desirable SCM behaviors to repository paths
- Repository paths make good streams and components for container-based SCM

SCM behaviors of repository paths

- Entire hierarchies of files can be branched, compared, and merged
- Every repository path has a history of reproducible states
- Repository paths can have branch and merge histories as well as change histories
- A version of a repository path identifies the versions of the lower-level paths and files it contains
- Any change to a file creates a new version of the repository paths it resides in

Repository paths as components

- Components may contain other components
- Items in a component must be accessible
- Items in a component have relative locations
- Components have locations relative to each other

- Repository paths are hierarchical
- Files and lower-level paths are accessible in a repository path
- Files in a repository path have relative locations
- Repository paths have locations relative to each other

Repository paths as streams

- Streams contain components
- Components may be created, modified, moved, renamed within streams
- Components may be branched from one stream to another

- Paths designated for streams can have sub-paths designated for components
- Path hierarchies can be created, modified, moved, renamed within a repository
- Files in a path hierarchy can be branched from one path to another

Back to LHC...

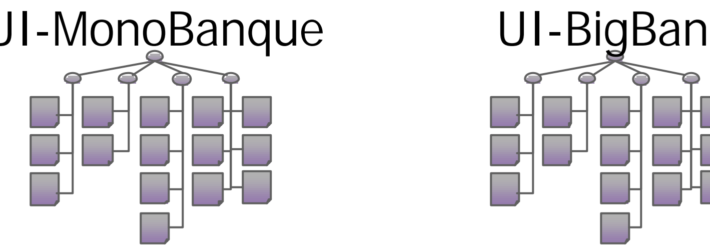
- LHC's UI division (UID) develop customized LFIN UI components
- UID's LFIN development stream

  //UID/LFIN/...

- contains UI components customized for each customer:

  //UID/LFIN/UI-MonoBanque/...

  //UID/LFIN/UI-BigBank/...



Container history

```
                                   A B C D E
//UID/LFIN/UI-MonoBanque/...       * *   * *
//UID/LFIN/UI-BigBank/...            * *   *
```

- Component history
- Stream history

Taking delivery of components

- LHC's release management division (RM) assembles and tests LFIN packages for customers
- RM's product packaging stream for LFIN:

  //RM/LFIN/...

- ...contains a component for each customer:

  //RM/LFIN/MonoBanque/...

  //RM/LFIN/BigBank/...

Taking delivery of components

- RM branches released UI components into its product packaging stream

//UID/LFIN/UI-MonoBanque/...

//UID/LFIN/UI-BigBank/...

//RM/LFIN/MonoBanque/UI/...

//RM/LFIN/BigBank/UI/...

Taking delivery of components

- RM takes delivery of AS component -- same component goes into each customer package:

```
//ASD/AS/...
            //RM/LFIN/MonoBanque/UI/...
            //RM/LFIN/MonoBanque/AS/...
            //RM/LFIN/BigBank/UI/...
            //RM/LFIN/BigBank/AS/...
```

Taking delivery of components

- RM takes delivery of LFIN online help components from the Doc division, matching language to customer:

```
//DOCD/LFIN-OH/FR/...
//DOCD/LFIN-OH/EN/...
                //RM/LFIN/MonoBanque/UI/...
                //RM/LFIN/MonoBanque/AS/...
                //RM/LFIN/MonoBanque/OH/...
                //RM/LFIN/BigBank/UI/...
                //RM/LFIN/BigBank/AS/...
                //RM/LFIN/BigBank/OH/...
```

Inspecting containers

- Reviewing changes to BigBank package

```
p4 changes //RM/LFIN/BigBank/...
```

- Comparing AS components between customer packages

```
p4 diff2 //RM/LFIN/MonoBanque/AS/...
         //RM/LFIN/BigBank/AS/...
```

- Checking labels that have been applied

```
p4 labels //RM/LFIN/MonoBanque/...
p4 labels //RM/LFIN/BigBank/...
```

R2.1.complete

R2.1.pending

Replacing Defective Components

- What the release manager does when tests fail:
  - Reports component failures to suppliers
  - Gets new component versions from suppliers
  - Rebuilds & retests

Tracing component origins

//RM/LFIN/BigBank/UI/...

//UID/LFIN/UI-BigBank/...

//UID/WAF/...

//WD/WAF/...

Replacing components

//RM/LFIN/BigBank/UI/...

//UID/LFIN/UI-BigBank/...

//UID/WAF/...

//WD/WAF/...

Real-world complexities

- Changes to component composition
- Intermediate assembly & ancillary files
- Target platform variants
- Release numbering

Suggested practices

- Do SCM operations on containers, not individual files
- Replace bad components; don't fix them in downstream containers
- Use a uniform naming convention
  - streams, components, labels
- Use labels to identify container versions or states, not as containers themselves

In summary…

- Container-based SCM promises to simplify large-scale software development by packing files into containers
- Repository paths work well as containers of files
- Inter-File Branching gives repository paths desirable SCM behaviors
- Inter-File Branching is well suited for container-based SCM