DeltaV: Adding Versioning to the Web

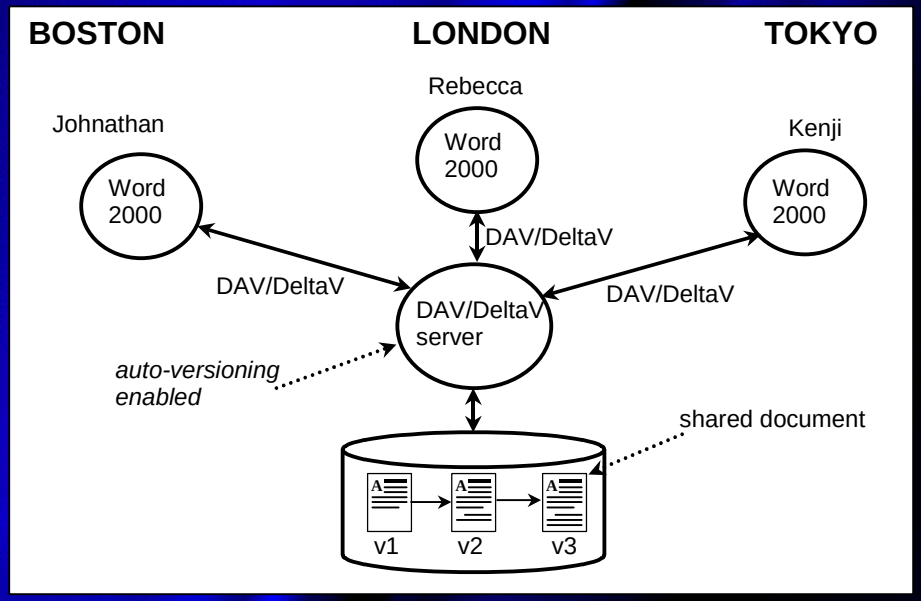
Jim Whitehead
University of California, Santa Cruz
ejw@soe.ucsc.edu

Documents Change

- Documents **change over time**
  - Web pages
  - Word processing documents
  - Spreadsheets
  - Images
  - …all are typically created via an **iterative authoring process**
- Document change tracking is desirable
  - Undo a mistake
  - Know **who** made a change, and **when**
  - Record **why** a change was made
- Change tracking provides
  - Greater *control* over change
  - An *archive* of past document states

Remote Collaborative Document Authoring Scenarios

- A task force of people from geographically dispersed business units need to develop a report together.
  - Team needs to solicit feedback, so they send out copies of the report.
  - As a result, the team needs a permanent copy of the exact report version they are having other people review.



Other Remote Collaboration Scenarios

- **Cross-company collaboration project**
  - Multiple companies, several countries.
  - Remote authoring of project Web site
    - Using a WebDAV-aware tool, such as Go Live, or Dreamweaver.
  - Since any project member can edit the Web site contents, the team wants to keep track of all changes
    - Records **who** changed **which** page
    - If mistakes are made, changes can be undone
- **Open source software project**
  - Team members from around the globe
  - Need to **record code changes**
  - Maintain **stable baselines** of public releases
  - Developers need to edit and compile source code on their local machine

WebDAV/DeltaV: In Support of Collaboration

- To support the collaboration scenarios, there is a need for a standard that supports:
  - Remote authoring of all types of documents
    - Web pages, word processing, spreadsheets, presentations…
  - Change tracking for these documents
  - Prevents collaborators from clobbering each other's work
  - Simultaneous work, in isolation from each other's changes
  - Recording stable baselines comprised of multiple documents
  - Recording information about documents (metadata)
- Which also…
  - Is supported by major tools
  - Is integrated into the Web
  - Is easy to use
- WebDAV and DeltaV do all this and more!

What is DeltaV?

- DeltaV is:
  - An application layer network protocol
    - Extends the WebDAV protocol
    - WebDAV itself extends HTTP 1.1
      - Thus DeltaV is also an extension to HTTP 1.1
      - HTTP is the core network protocol of the Web
  - An interoperability standard
    - Allows document authoring and software development applications to **interoperate** with versioning and configuration management systems using a **common interface**
  - A data model
    - DeltaV embodies a data model that is capable of modeling multiple versioning and configuration management repositories
    - The data model is independent of network protocol
  - A data integration infrastructure
    - A common place for cross-application data sharing

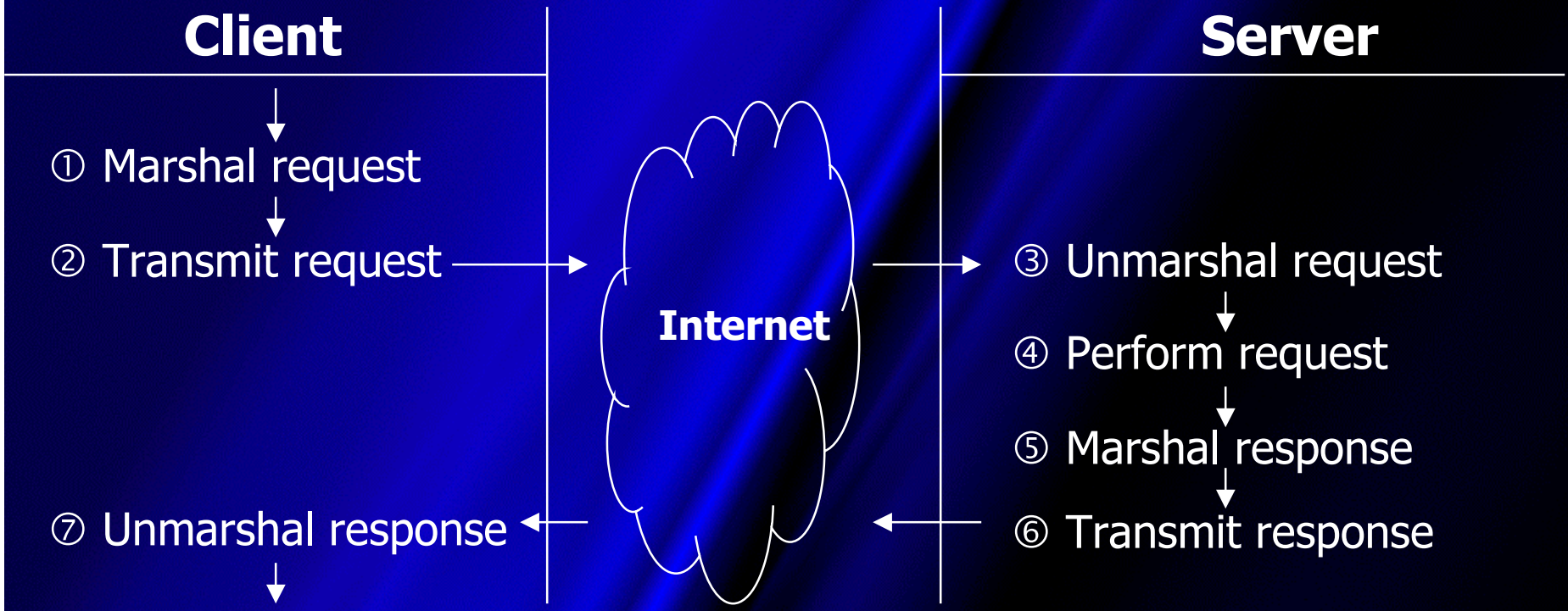
Who is DeltaV?

- DeltaV is an **IETF** Working Group
  - IETF: Internet Engineering Task Force
    - Developed standards such as TCP, IP, SMTP, POP, FTP
  - Typical IETF & W3C spheres of standardization:
    - IETF: network protocol and identifier standards (HTTP, URL)
    - W3C: content standards (HTML, CSS, XML, XSS, RDF)
- IETF considers its working groups to consist of individual contributors, not corporate representatives
  - Anyone may join an IETF working group
  - Agreement is by rough consensus: no voting
- Still, corporate affiliations can be informative:
  - Chair: Jim Amsden, IBM
  - Document authors include Geoff Clemm, Rational, Chris Kaler, Microsoft
  - Other participants: Merant, Intersolv, OTI

Hypertext Transfer Protocol (HTTP)

Briefly, HTTP

- HyperText Transfer Protocol (RFC 2616)
- A **remote procedure call** protocol
  - Client sends a request to the server
  - Server processes request, and returns result



HTTP Request Format

- A request line:
  - {method} {Request-URI} {protocol version}
  - Example:
    - GET /users/ejw/flyer.doc HTTP/1.1
- A series of headers:
  - {header}: {value}
  - RFC 822 (email) style header encoding
  - Example:
    - Content-Type: text/html
- An optional request body
  - Separated from headers by a <CR><LF>
  - A sequence of 8-bit bytes (octets)

HTTP Response Format

- Very similar to the request format:
- A status line:
  - {protocol version} {status code} {status phrase}
  - Status codes:
    - 1xx: Informational
    - 2xx: Successful
    - 3xx: Redirection
    - 4xx: Client error
    - 5xx: Server error
  - Example:
    - HTTP/1.1 200 OK
- A series of headers
  - Same formatting as in request
- A response body
  - Same formatting as in request

Sample HTTP interaction

Request

GET / HTTP/1.1
Host: www.webdav.org
Content-Length: 0

Response

HTTP/1.1 **200 OK**
Date: Wed, 25 Apr 2001 22:58:22 GMT
Server: Apache
Last-Modified: Thu, 19 Apr 2001 05:30:49 GMT
ETag: "30e7da-2f9d-3ade7809"
Accept-Ranges: bytes
Content-Length: 12189
Content-Type: text/html

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>WebDAV Resources</title>
```

Important HTTP 1.1 Concepts

- Requests are transmitted over a **reliable transport**
  - Invariably, this is TCP/IP
- Network connection is left **open** at the end of a request
  - Uses the network more efficiently
- **Stateless protocol**: all information needed to process a request is contained in the request

- **Entity tag** is a unique identifier for the state of a resource
  - If the GET response body changes, so does the Etag
  - Useful for determining if the resource has changed
    - Should a cache update its value?
    - Did someone modified this resource while I was editing it?

HTTP Methods

- Operations supported by HTTP include:
  - GET – read a resource
  - HEAD – just return headers about a resource
    - Useful for caching (quickly retrieve entity tag)
  - OPTIONS – lists supported methods
  - POST – protocol-tunneling method
  - PUT – write a resource
  - DELETE – remove a resource
- Although HTTP 1.1 supports PUT and DELETE, it is primarily a read-only protocol
  - PUT and DELETE are not widely supported by clients

Web Distributed Authoring and Versioning (WebDAV)

WebDAV

- The goal of WebDAV is to add capabilities for remote authoring and versioning to HTTP
  - But, the versioning part was dropped in order to finish the remote authoring protocol
  - RFC 2518 is the defining document for WebDAV
  - DeltaV picked up where WebDAV left off
    - The changes (Δ) needed to add the "V" back into WebDAV
- WebDAV capabilities include:
  - **Overwrite prevention:** lock, unlock
  - **Properties:** list, add, remove
  - **Namespace Operations:** move, copy
  - **Collections:** mkcol, hierarchy operations

Collaboration Infrastructure

- Whole resource locking supports:
  - Remote collaborative authoring of HTML pages and associated images
  - Remote collaborative authoring of any media type (word processing, presentations, etc.)
  - Locks time out, easing administration of locks
- *Infrastructure* for development of asynchronous, widely distributed, hypertext-aware, collaborative editing tools.

Metadata Recording Infrastructure

- Metadata support
  - ***Properties.*** (name, value) pairs can be created, modified, deleted, and read on Web resources.
  - Consistency of properties can be maintained by the server or the client
  - Property values are well-formed Extensible Markup Language (XML)
- *Infrastructure* for how to record information *about* Web data

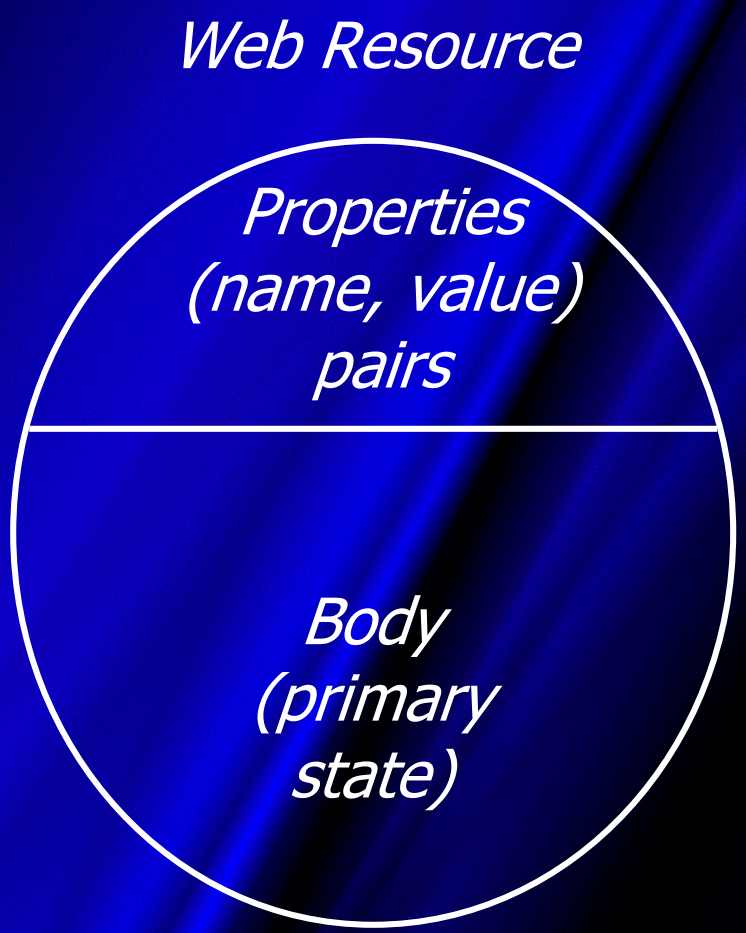
Namespace Management Infrastructure

- Remote name space management:
  - Copy and Move individual resources, and hierarchies of resource
  - Create and modify (ordered) collections of resources
  - Add/remove members by-reference
- *Infrastructure* for remotely organizing and viewing collections of Web resources

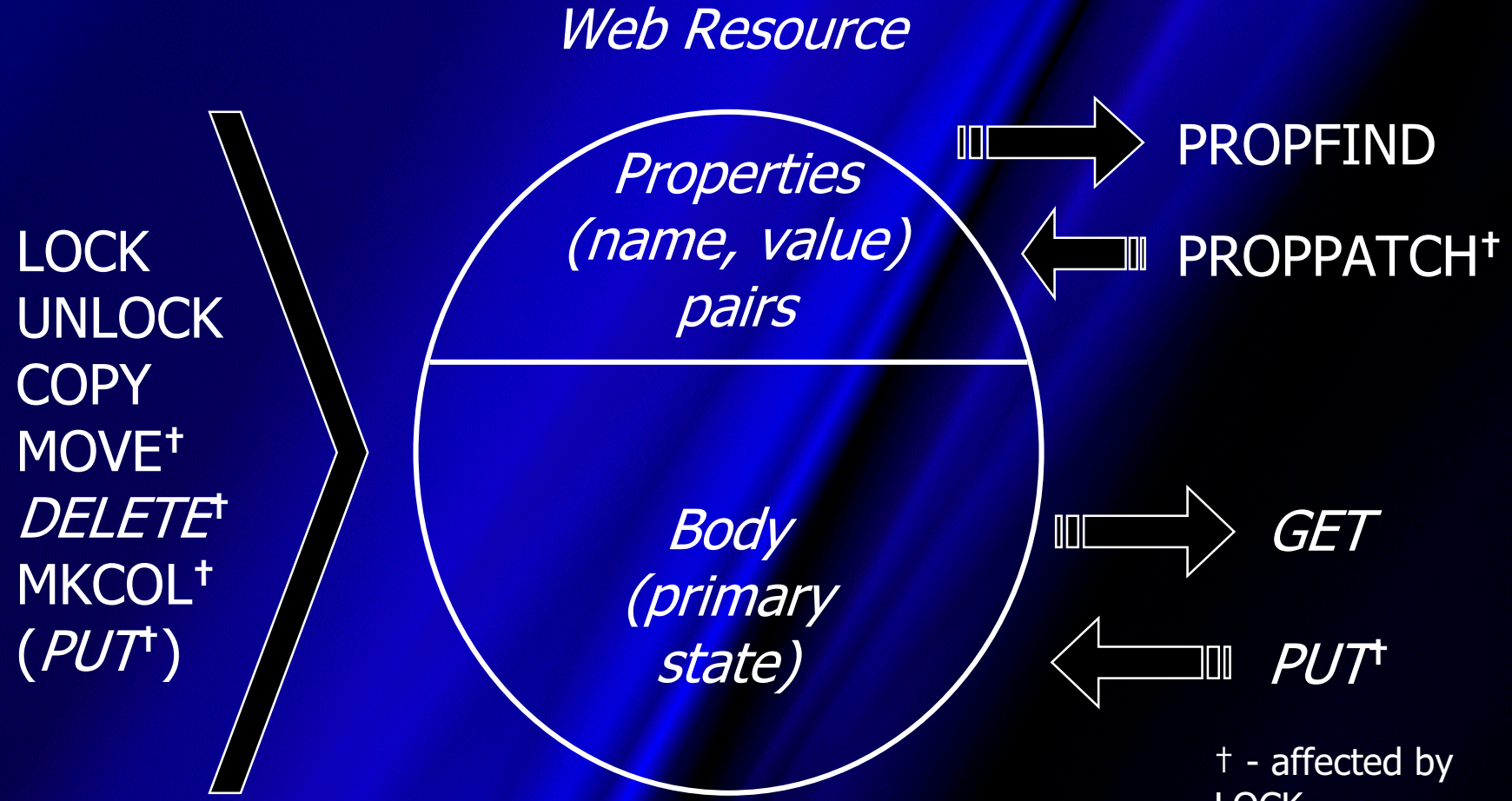
WebDAV Methods

- Overwrite Prevention:
  - LOCK – prevents non-lock holders from writing to the resource
  - UNLOCK – removes a lock
- Metadata Management:
  - PROPFIND – read properties from a resource
    - Allprop – all property names and values
    - Propname – only return property names
    - Prop – just return specified properties
  - PROPPATCH – write properties on a resource
- Namespace Management
  - COPY – duplicate a resource
  - MOVE – move a resource (preserving identity)
  - MKCOL – create a new collection

WebDAV Object Model



Scope of WebDAV Methods



Major WebDAV Servers

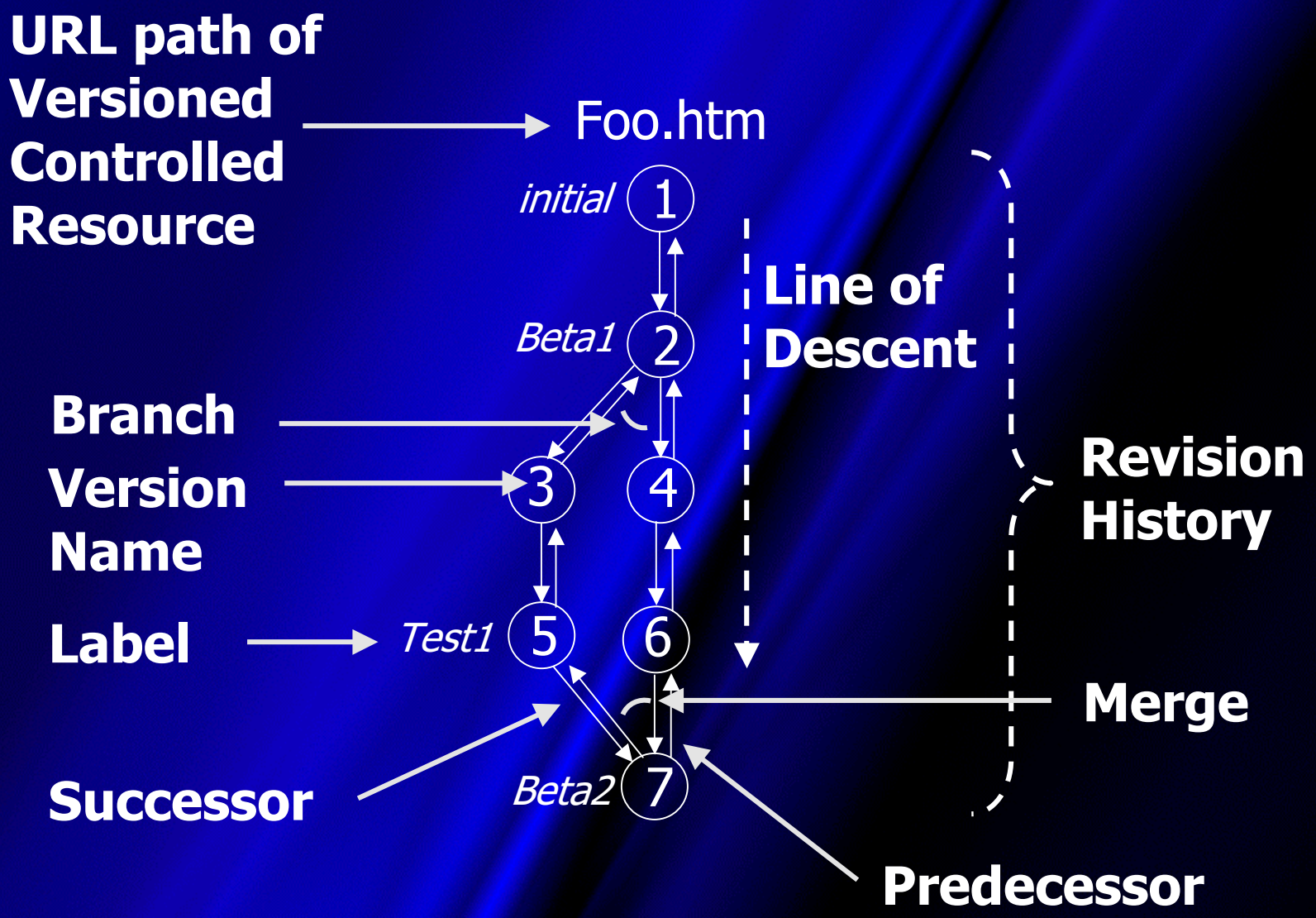
Microsoft: IIS 5, Exchange 2000, Sharepoint
Apache: mod_dav (over 10,000 sites)
Oracle: Internet File System
Adobe: InScope
Xythos: Storage Server
Novell: Netware 5.1, Net Publisher
W3C: Jigsaw
Endeavors: Magi-DAV
IBM: DAV4J (DeveloperWorks)
DataChannel: DataChannel Server (DCS 4.1)
Intraspect: Knowledge Server
OpenLink: Virtuoso
Hyperwave: Information Server 5.5

Major WebDAV Clients

- Application Software:
  - **Microsoft:** Office 2000 (Word, Excel, PowerPoint, Publisher)
  - **Adobe:** Photoshop 6, Acrobat 5
- Web Site Authoring
  - **Adobe:** Go Live 5
  - **Macromedia:** Dreamweaver 4
- Remote File Access:
  - **Apple:** Mac OS X webdavfs
    - OS X also ships with Apache and mod_dav (have to configure mod_dav to make it work)
  - **Microsoft:** Windows Web Folders
  - **Wind River Software:** WebDrive
  - Goliath (Mac, open source)
  - Nautilus (GNOME project, Eazel)
  - WebDAV Explorer (UC Irvine, Feise/Kanomata, open source)
- XML editors
  - **Excosoft:** Documentor
  - **Altova:** XML Spy 3.5

Versioning Terminology and Data Model

Version History

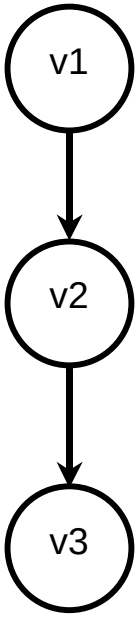


Placing a resource under version control

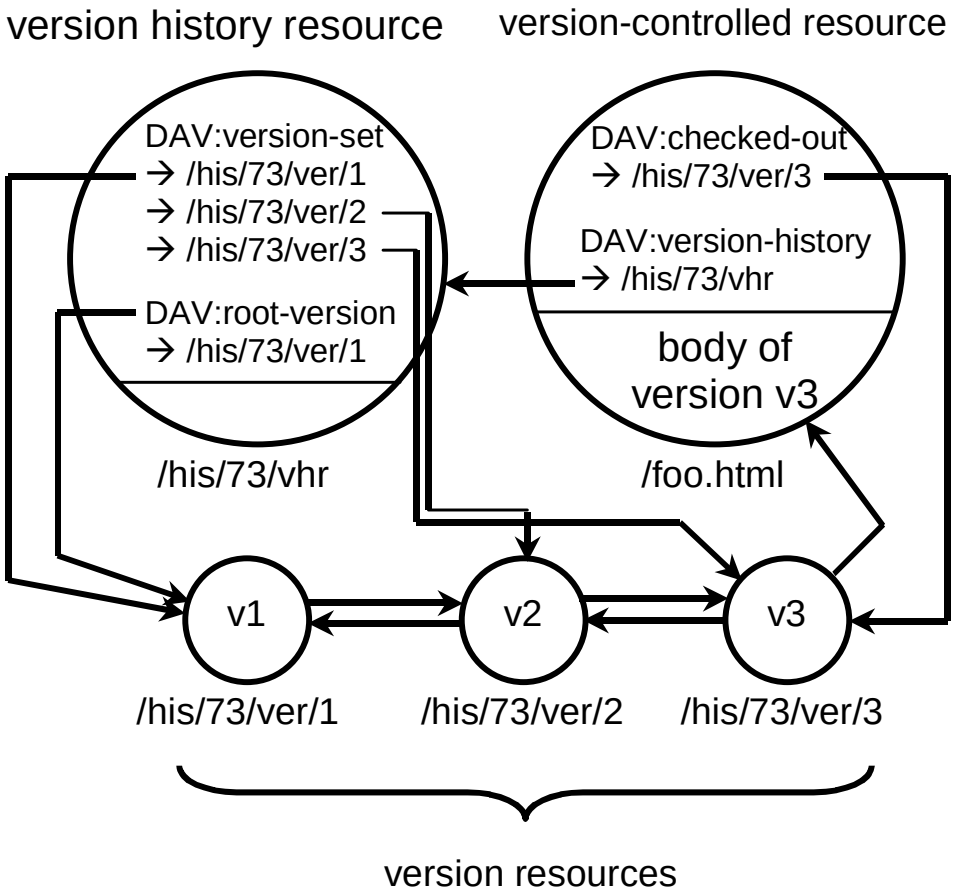
- VERSION-CONTROL puts an unversioned resource under version control
- VERSION-CONTROL does three operations:
  1. Creates a new **version history resource**
     - A list of URLs of the versions in this revision history
     - The URL of the root version
  2. Creates a new **version resource**
     - Predecessor and successor identifiers stored in properties (initially empty)
     - URL is **not** the same as the original unversioned resource
  3. Converts the unversioned resource into a **version controlled resource**
     - URL is the **same** as the original unversioned resource
     - Has a pointer to the version history resource
     - Records whether it is checked-out or checked-in

Representation of a Version History

Abstract version history of foo.html



Representation of foo.html in DeltaV



Unique Aspects of Versioning Data Model

- DeltaV data model is different from RCS/SCCS model
  - RCS: placing foo.html under version control causes:
    - foo.html to be made read-only
    - Creation of RCS/foo.html,v archive file
      - Archive contains version history and version data
  - Differences
    - Both use the original name as the place where commands are directed
    - But, version-controlled resource has more metadata
    - DeltaV separates version history from version storage
    - DeltaV gives each version a unique identifier (URL)
      - Note that delta storage can still be used in the underlying repository

Rationale for Versioning Data Model

- Each version is given its own URL, thus allowing hypertext linking to a specific version.
  - Hence, each version must be a separate resource
- Workspaces are the motivation for separating the version-controlled resource from the history resource.
  - A workspace allows a collaborator to work in isolation, in their own part of the namespace.
  - For example, Greg and Lisa might have workspaces:
    - /ws/users/greg/ and /ws/users/lisa/
  - If both want foo.html to appear in their local workspace, **each needs a version controlled resource**.
  - Thus, one version history can be associated with multiple version controlled resources.

Versioning Operations

CHECKOUT

- DeltaV uses the library metaphor for versioning.
  - To work on a resource, you:
    - Check out
    - Make edits
    - Check in
- CHECKOUT is applied to a version controlled resource
  - Allows modifications to the body and dead properties of the version controlled resource
  - If a version is already checked-out, or has a successor (is non-tip), a fork (branch) is possible.
    - DAV:checkout-fork property on a version affects branch formation
      - ok: branching is permitted
      - discouraged: the client must explicitly indicate branching is OK
      - forbidden: branching is not allowed (linear version history)

CHECKIN / UNCHECKIN

- CHECKIN freezes the state of a version controlled resource:
  - Creates a new version resource
    - Body and dead properties are the same as those of the version controlled resource
  - Freezes the body and dead properties of the version controlled resource (making them read-only)
  - A comment describing the purpose of a change can be written into the *DAV:comment* property (before CHECKIN)
  - Fork control options are available on CHECKIN as well
- UNCHECKOUT aborts the editing sequence
  - Cancels the CHECKOUT
  - Restores the pre-CHECKOUT state of the version controlled resource

Autoversioning

- There are many WebDAV-aware applications
  - They know nothing about versioning!
  - Would like to provide **auto**matic **versioning** support
- Two styles of autoversioning:
  1. Every modification creates a new version
     - PUT/PROPPATCH ➲ CHECKOUT → PUT/PROPPATCH → CHECKIN
     - Works best for authoring clients that replicate resources to a local disk, and only write to the server at the end of an editing session.
  2. Every LOCK/UNLOCK pair creates a new version
     - LOCK ➲ LOCK → CHECKOUT
     - UNLOCK ➲ CHECKIN → UNLOCK
     - Works well for authoring clients that work directly on the WebDAV server and take out locks, like Office 2000.
  - Style is controlled by *DAV:autocheckout* and *DAV:autocheckin* properties on version controlled resource

Labels

- **Human-readable** strings that can be attached to a particular version
- **Guaranteed uniqueness** within a version history
- Can be **reused** across version histories
  - For example, "release_beta" could be used to identify different versions in multiple version histories
  - Can be used as a simple way to record configurations
    - Drawback: the mapping of labels to versions can change over time, so configurations are not guaranteed to be stable
- LABEL method is used to set/move/remove a label on a version
- Label: header can be used with GET/PROPFIND on a version controlled resource to retrieve information from the named version.

UPDATE

- Body and dead properties of a version controlled resource are typically the same as the last checked-in version
  - In a linear version history, the version controlled resource typically tracks the tip version
- UPDATE
  - Modifies the content and dead properties…
  - … of a checked-in version-controlled resource
  - … to be those of a specified version from the version history
- Benefit: An **arbitrary version** can be reflected by the version controlled resource.

Merging

- Branches in a version history are used for two purposes:
  - Representing simultaneous development by multiple collaborators (parallel development)
  - Representing variants of a resource
    - Natural language, computing platform, etc.
- It is useful to merge together branches that represent parallel development
  - Combine the efforts of multiple people into a consistent whole
- MERGE combines two versions
  - Minimally combines the two branches in the version history
  - Performs a best-effort content merge if the server understands the content type of the two versions
    - text/* merging will likely be the only types uniformly supported

Version Tree Report

- A **graphical visualization** of a version history aids understanding
  - Information needed to create this view:
    - List of versions
    - Version names
    - Predecessor/successor relationships
  - In DeltaV data model, information is distributed:
    - Version history resource: list of versions
    - Version resources: version names, pred/succ relationships
    - Using PROPFIND, would require N+1 network requests in the worst case (N=# of versions)
  - REPORT method supports many kinds of reports
    - **Version tree report** allows arbitrary properties to be requested from all versions of a version history
      - *DAV:version-name* & *DAV:successor-set* properties
    - Now, only one network request is needed

Activities

Purpose and Definition

- Frequently, a logical change spans multiple versions in multiple version histories
  - For example, fixing a specific software bug can involve the creation of multiple versions of several files
- Activities associate a logical change with the set of versions created while making that change.
  - Versions from a given version history are **limited to a single line of descent**
  - Activities are not versioned
- Sometimes want to ensure that all work done by team members is on a single line of descent
  - Avoids merging between team members
  - Activities can be used to enforce working on just a single line of descent

Working with Activities

- MKACTIVITY creates a new activity resource
- Upon checkout, the author specifies an activity
  - This activity is associated with the new version created at checkin
  - Specifically:
    - URL of the checked-out resource is stored in activity's *DAV:activity-checkout-set* property
    - On checkin, the new version's URL is recorded in activity's *DAV:activity-version-set* property…
    - … and the checked-out resource is removed from *DAV:activity-checkout-set*
- Using activities, it is possible to select a particular logical change for merging into another workspace
  - The activity is used in a MERGE request
    - Latest version in an activity (for each version history) is the one used in the merge

Latest Activity Version Report

- Since an activity only records a set of version URLs, determining the latest version in an activity would require:
  - Retrieving the list of versions in an activity
  - Determining the associated set of version histories
  - Retrieving the version tree report for each version history
  - Evaluating the version graphs to determine which of the activity versions is the latest
- The latest activity version report returns the latest version in an activity, for one version history
  - REPORT is invoked on a version history resource
  - Passes URL of an activity resource

Workspaces

Purpose and Definition

- A **workspace** is a location…
  - … where a person can work **in isolation**
  - … from **ongoing changes** made by other collaborators
  - to the **same set** of resources.

- Workspace use scenarios:
  - Two people each modify a common source code file, and want to test their work before integrating changes
  - The directory structure of a large project is being changed
    - Don't want to require everyone to stop working while this takes place
    - Keep namespace changes invisible to team until complete
  - A developer is trying a change that might not pan out
    - New files are in their workspace, but not under version control until the final viability of the change is known

Workspaces

- Two kinds of workspaces: server and client
  - Main difference is whether the server maintains a portion of its namespace for each workspace
- Workspaces can hold versioned and unversioned data

Server workspaces

- Maintains a separate namespace for each workspace
- For example: Lisa and Chuck both have workspaces
  - Lisa's: /users/people/**lisa**/projectX/…
    - Lisa does all her work in this tree
  - Chuck's: /users/people/**chuck**/projectX/…
    - Chuck does all his work in this tree
- Local replication of versions may still occur
- Advantages
  - Permit access from multiple locations and machines
  - Permit namespace operations (like MOVE) to be isolated
- Disadvantages
  - Requires significant manipulation of the server namespace
    - Might be tricky to implement for filesystem-based repositories

Client workspaces

- No server namespace is maintained
- Working files are replicated to local machines
- Server reserves a **working resource** for each checkout
  - When the edit session is done, data is written to the working resource
- Advantages
  - Less use of server namespace
  - Close match to CVS replicate-work-synchronize work pattern
- Disadvantages
  - Cannot access workspace from multiple machines and locations
  - Namespace operations are immediately visible

Server Workspace Mechanics

- MKWORKSPACE creates a new workspace resource
  - This is the top-level directory of the project
  - E.g., /users/people/lisa/projectX/
- VERSION-CONTROL adds version controlled resources to a workspace
  - Creates a new version controlled resource for a version history
  - The version history will have **multiple** version controlled resources
  - Think of the version controlled resource as the access point, at some location in the namespace, for operations on a version history.

Synchronizing Workspaces

- At some point in time, a person working in a workspace will need to bring in changes made by other collaborators.
- MERGE combines two workspaces
  - Typically, the source of the merge is the common project workspace, and the destination is a personal workspace
  - MERGE acts to select the most recent checked-in version for each version-controlled resource in the workspace
- A client may want to present the outcome of a workspace merge before it is actually performed
  - Allows the user to decide if they like the outcome
  - The **merge-preview** report describes the changes that would occur due to a MERGE operation
    - Invoke it using REPORT

Server Workspace Example

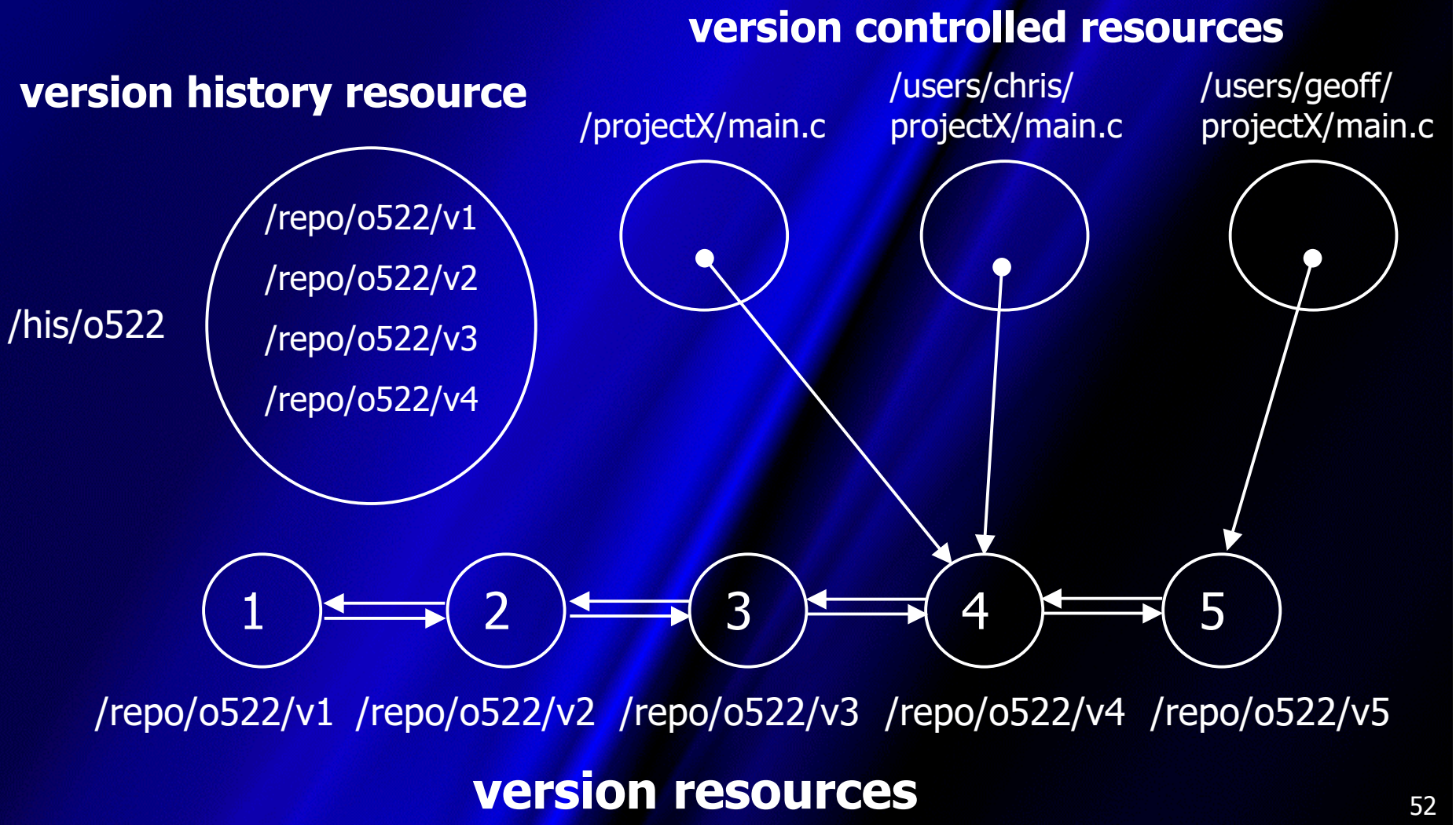
- Consider a program with the following source files:
  - `/projectX/makefile`
  - `/projectX/main.c`
  - `/projectX/defs.h`
- Geoff and Chris want to collaborate together on this project
  - Each person works in a separate server-side workspace
- Create the workspaces:
  - MKWORKSPACE `/users/`**`geoff`**`/projectX/`
  - MKWORKSPACE `/users/`**`chris`**`/projectX/`

Server Workspace Example (2)

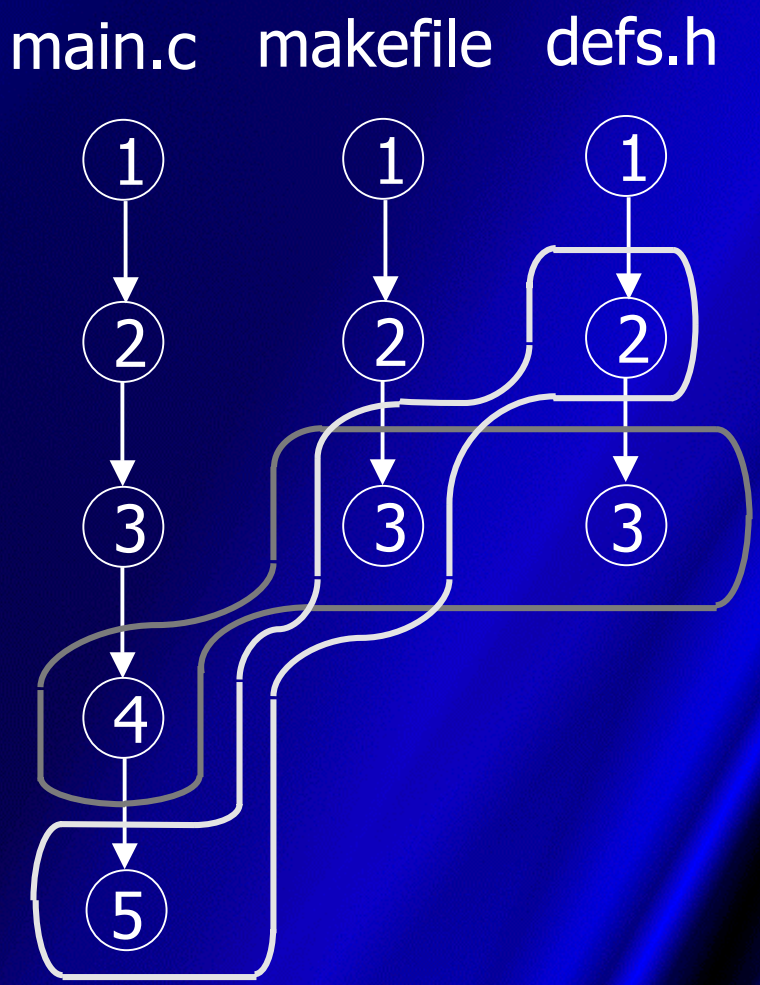
- Use version control to add project files:
  - Populate Geoff's workspace:
    - VERSION-CONTROL `/users/geoff/projectX/makefile` from `/projectX/makefile`
    - VERSION-CONTROL `/users/geoff/projectX/main.c` from `/projectX/main.c`
    - VERSION-CONTROL `/users/geoff/projectX/defs.h` from `/projectX/defs.h`
  - Populate Chris' workspace:
    - VERSION-CONTROL `/users/chris/projectX/makefile` from `/projectX/makefile`
    - VERSION-CONTROL `/users/chris/projectX/main.c` from `/projectX/main.c`
    - VERSION-CONTROL `/users/chris/projectX/defs.h` from `/projectX/defs.h`

Server Workspace Example (3)

- Version history of main.c:



Server Workspace Example (4)



/users/geoff/projectX/
main.c, 5
makefile, 3
defs.h, 2

/users/chris/projectX/
main.c, 4
makefile, 3
defs.h, 3

Client Workspaces

- Client maintains control over namespace of the locally replicated data
  - Server is required to maintain very little state
- Working resource
  - Created upon checkout
  - A location on the server where the client can write the contents of the checked-out resource
  - Goes away upon checkin
    - CHECKIN converts working resource into a version resource
  - A version history can have multiple simultaneous working resources

Client Workspace Example

- Assume work on file main.c:

1. Client first replicates a version of main.c using GET
   - Result is stored under a local name:
     - /{local workspace name}/main.c
2. Then, client uses CHECKOUT to checkout the same version
   - Creates a working resource
   - URL of working resource found in Location: header of CHECKOUT response
3. Client works on local copy of main.c
4. When done, client saves changes back to the server
   - PUT to the working resource
5. Then, CHECKIN of working resource
   - Turns working resource into a version

Baselines

Purpose of Baselines

- When a large collection of documents is released outside the developing organization, there is a need to record exactly which document versions were released.
  - Especially true for software project releases
- A baseline can record the exact version of a large set of resources under version control
- Baselines can also be used for recording intermediate project states
  - The state of the project as of last Friday
    - Quality assurance can then have a stable snapshot to work from

Baseline Definition

- A *baseline* is a special kind of version resource that captures the state of the version-controlled members of a configuration.
- A *baseline history* is a version history whose versions are baselines
  - Baselines are **versioned** and change over time
- New baselines are created by checking out and then checking in a special kind of version-controlled resource called a *version-controlled configuration*
  - Baselines thus have the same three resource data model as other resources under version control
    - Version history for baselines → baseline history
    - Version resources → baseline
    - Version controlled resource → version controlled configuration

Baseline Mechanics

- A collection is placed under **baseline control** using the BASELINE-CONTROL method
  - Example:
    - Source code for a project is in the /project/src/ collection
    - Apply BASELINE-CONTROL to /project/src/
- Creating a new baseline history:
  - BASELINE-CONTROL method:
  - Creates a new version controlled configuration
    - Puts its URL in *DAV:version-controlled-configuration* property of collection
      - Records association between baseline history and collection

Baseline Mechanics (2)

- Creating a new baseline:
  - CHECKOUT version controlled configuration
    - Makes the version controlled configuration changeable
- Taking a snapshot of a project:
  - CHECKIN the version controlled configuration
    - Creates a new baseline
      - I.e., a new version in the version history of baselines
    - Also creates a new collection…
    - … in a **server-defined** portion of the namespace …
    - … which contains a set of version-controlled resources …
    - … each of which has the value of the most recently checked-in version in its version history.
    - Net effect:
      - The new collection is a **snapshot of the checked-in state** of the baselined collection at the moment of checkin

Baseline Details

- A workspace is defined to be a collection
  - Can place a workspace under baseline control
- It is possible to compare baselines
  - Use **compare-baseline** report
    - Invoke with REPORT method
  - Describes which versions have been added, removed, or changed

Features & Packages

Basic Features

- Protocol functionality is divided into:
  - Features
  - Assigned to be either Basic or Advanced
- Basic Features:
  - VERSION-CONTROL
    - Basic versioning
    - Automatic versioning only (no checkin or checkout)
    - REPORT method and version-tree report
  - CHECKOUT
    - Checkout, checkin, and uncheckout
  - VERSION-HISTORY
    - Version history resource has a URL in the server namespace
  - WORKSPACE
    - Server workspace functionality
  - WORKING RESOURCE
    - Client workspace functionality
  - UPDATE
  - LABEL

Advanced Features

- Advanced Features:
  - MERGE
    - Considered advanced due to workspace capabilities
  - BASELINE
    - Recording checked-in state of a collection
  - ACTIVITY
    - Versions associated with a logical change
  - VERSION-CONTROLLED-COLLECTION
    - Collections can be versioned
- Feature discovery is via OPTIONS
  - Each feature has an associated tag that appears in the DAV: response header
    - E.g.: VERSION-CONTROL feature → "version-control" in DAV response header
      - DAV: 1, 2, version-control

Packages

- Features are divided into packages.
  - Attempt to reduce the number of possible server feature permutations a client might encounter
- A basic versioning server should support one of the following feature sets:
  - **Core-Versioning**
    - VERSION-CONTROL only
  - **Basic-Server-Workspace**
    - VERSION-CONTROL
    - WORKSPACE
    - VERSION-HISTORY
    - CHECKOUT
  - **Basic-Client-Workspace**
    - VERSION-CONTROL
    - WORKING-RESOURCE
    - UPDATE
    - LABEL

Advanced Packages

- There are two packages for advanced features:
  - **Advanced-Server-Workspace**
    - Basic-Server-Workspace
    - Plus all advanced features
      - MERGE
      - BASELINE
      - ACTIVITY
      - VERSION-CONTROLLED-COLLECTION
  - **Advanced-Client-Workspace**
    - Basic-Client-Workspace
    - Plus all advanced features
- Thus, a client will likely encounter only servers that have implemented one of six packages

DeltaV Resources

WebDAV

- **WebDAV Resources**
  - http://www.webdav.org/
    - A central collection of pages and links to all things WebDAV.
- **WebDAV Working Group**
  - http://www.ics.uci.edu/pub/ietf/webdav/
    - Contains links to active documents, and a complete list of WebDAV-supporting applications.
- **RFC 2518 – WebDAV Distributed Authoring Protocol**
  - http://www.ics.uci.edu/pub/ietf/webdav/protocol/rfc2518.pdf
    - This is the WebDAV Distributed Authoring Protocol specification
- **WebDAV: A network protocol for remote collaborative authoring on the Web**
  - *Proc. of the Sixth European Conference on Computer-Supported Cooperative Work*, Sept. 12-16, 1999, Copenhagen, Denmark, pp. 291-310.
  - http://www.ics.uci.edu/~ejw/papers/dav-ecscw.pdf
    - An academic paper giving an overview of the WebDAV Distributed Authoring Protocol.

DeltaV

- **Delta-V Working Group web page**
  - http://www.webdav.org/deltav/
    - The home page for the IETF Delta-V Working Group, with links off to the most recent specifications.
- G. Clemm, J. Amsden, C. Kaler, J. Whitehead, "**Versioning Extensions to WebDAV**", Internet-Draft, work-in-progress, draft-ietf-deltav-versioning-15, April 17, 2001.
  - http://www.webdav.org/deltav/protocol/draft-ietf-deltav-versioning-15.htm
    - The most recent revision of the versioning and configuration management protocol specification.
- **The Future of Distributed Software Development on the Internet**
  - Web Techniques, Vol. 4, No. 10, October, 1999, pages 57-63
  - http://www.webtechniques.com/archives/1999/10/whitehead/
    - An introduction to WebDAV and DeltaV that describes the advantages of DeltaV over CVS for remote collaborative software development