PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

Introduction

PERFORCE
Version everything.

- Introductions
- Class Schedule
- GUI vs. CLI
- P4Admin Demonstrations
- About the Exercises

Course Contents

PERFORCE
Version everything.

- Advanced Maintenance
- Offline Checkpoints
- Broker
- Replication – Introduction
- Replication – HA/DR, Build Farms, Forwarding Replicas
- Fully Distributed
- Security
- Advanced Tools
- Scripting

PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

Advanced Maintenance

Topics

PERFORCE
Version everything.

- Recover a Stored Spec Revision
- Lazy Copies
- Archive/Restore

Spec Depot

PERFORCE
Version everything.

- **Goal**
  - Recover specs such as clients and protection table
  - Keep history of changes to specs
  - Identify user who changed a spec

- **Implementation**
  - Separate spec depot automatically maintained by Perforce Server
  - Specs are stored as form files, which can be printed or synced
    - Grouped into directories by type, such as *client* or *label*

Spec Depot Usage

PERFORCE
Version everything.

- Spec depot stores specs like clients and protection table (not change)
- Tracing of changes by a user

```
p4 print -q //spec/label/lastbuild.p4s#1
# The form data below was edited by bruno
```

- Optional: controlling which specs are versioned

```
p4 depot specs
SpecMap:
    //specs/...
    -//specs/client/build_ws_*
```

Recovering a Stored Spec Revision

PERFORCE
Version everything.

- List revisions in the spec depot

```
p4 filelog //specs/client/bruno_ws.p4s
... #4 default change edit on 2008/11/01
... #3 default change edit on 2008/11/01
... #2 default change edit on 2008/11/01
... #1 default change add on 2008/11/01
```

- Display content of revisions

```
p4 print -a //specs/client/bruno_ws.p4s
```

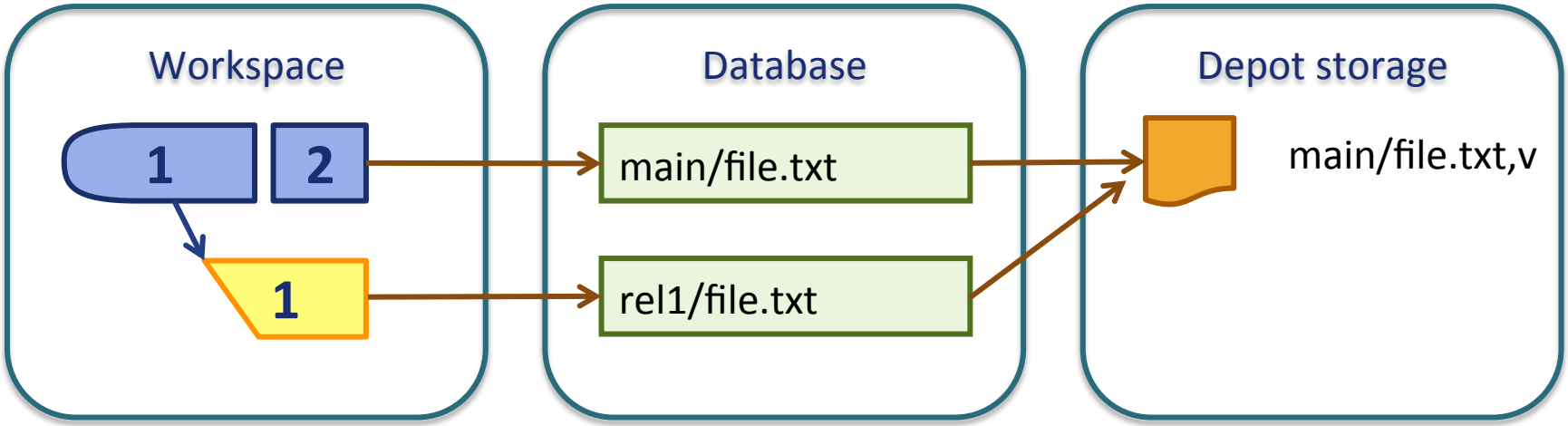
- Replace spec with earlier version

```
p4 print -q //specs/client/bruno_ws.p4s#3 | p4 client -i
```

Branching and Lazy Copies

PERFORCE
Version everything.

- Files branched or copied only create metadata entry in the database
  - Retain reference to original file location ➔ lazy copy



Lazy Copies and Snap

PERFORCE
Version everything.

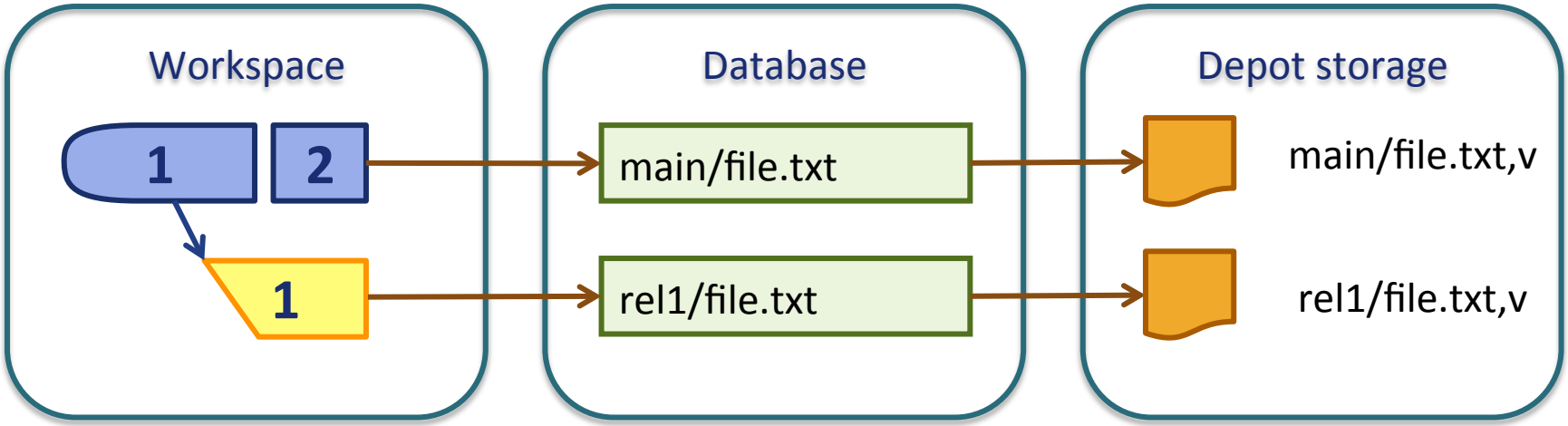
```
p4 fstat -Oc //depot/Jam/REL2.0/src/jam.c
...
... lbrFile //depot/Jam/MAIN/src/jam.c
... lbrRev 1.30
... lbrType text
... lbrIsLazy 1
```

(undocumented)

```
p4 snap //depot/Jam/REL2.0/src/jam.c
//depot/Jam/REL2.0/src/jam.c#1 -
        copy from //depot/Jam/MAIN/src/jam.c 1.30
```

After Snap

- Files in the depot storage are duplicated
- Useful when cleaning up depots with *obliterate*



Archiving and Restoring

PERFORCE
Version everything.

- Goal:
  - Free up space in active depots
  - Speed up backup and verify
  - Preserve history
  - Simple restore

- Implementation:
  - Separate archive depots (typically located on cheap storage)
  - Files can be archived and restored at individual revisions

Archiving and Restoring

PERFORCE
Version everything.

- Binary files *not branched* can be archived
  - Requires at least one depot of type *archive*
  - Preserves history

```
p4 archive -D archives //assets/...
```

- Restore files as needed

```
p4 restore -D archives  //assets/images/myimage.jpg#3
```

Verify Advanced Options

PERFORCE
Version everything.

- -z
- -v (Use with extreme caution).
- -u (Not generally needed on new servers).
- -b

- Explain batch mode
- Strategies for big data:
  - Parallel verify depot by depot
  - Use replica (see advanced class)
  - verify #head,#head

Archiving – Listing and Purging

PERFORCE
Version everything.

- Files in original depot are marked as *archive*

```
p4 files //assets/...
//assets/images/myimage.gif#1 - archive change 865 (ubinary)
...
```

- List files in archive depot

```
p4 files -A //archives/...
//archives/assets/images/myimage.gif#1
...
```

- Purge unneeded archived files (cannot be undone)

```
p4 archive -D archives -p //assets/...@2012/01/01
```

Exercises

Lab Set E1: Advanced Maintenance

New commands in this chapter (samples):

- `p4 archive`
- `p4 restore`
- `p4 snap (undoc)`

PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

Offline Checkpoints

Topics

PERFORCE
Version everything.

- Offline Checkpoints
  - Usage
  - Upgrades
  - Switch offline_db/root

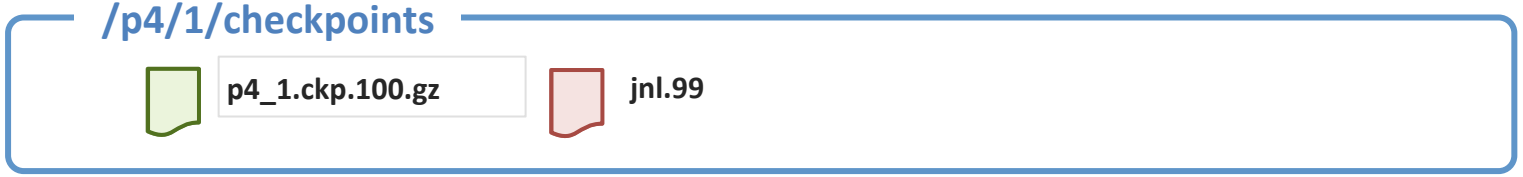
Offline Checkpoint

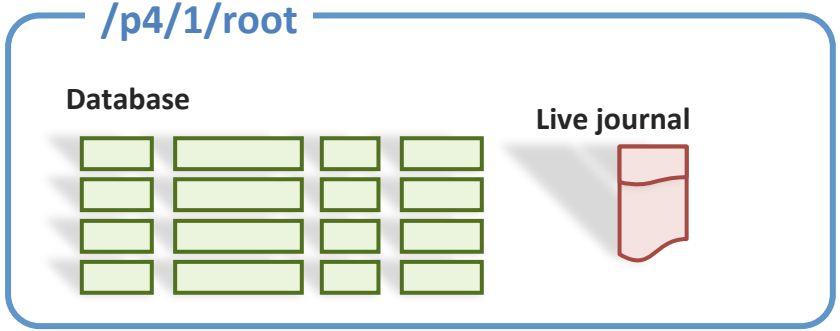
PERFORCE
Version everything.

- **Goal**
  - Checkpoint without any downtime
  - Easy and fast recovery
  - Optional: regular database restoration
    - Restored databases are smaller than original, but contain equivalent data (Removes empty data pages and rebalances the b-tree indexes)
- **Implementation**
  - Separate offline database created from checkpoint
  - Regular updates through rotated journal
  - Offline database dumped into checkpoint

Prep Offline Checkpoint – Create Seed

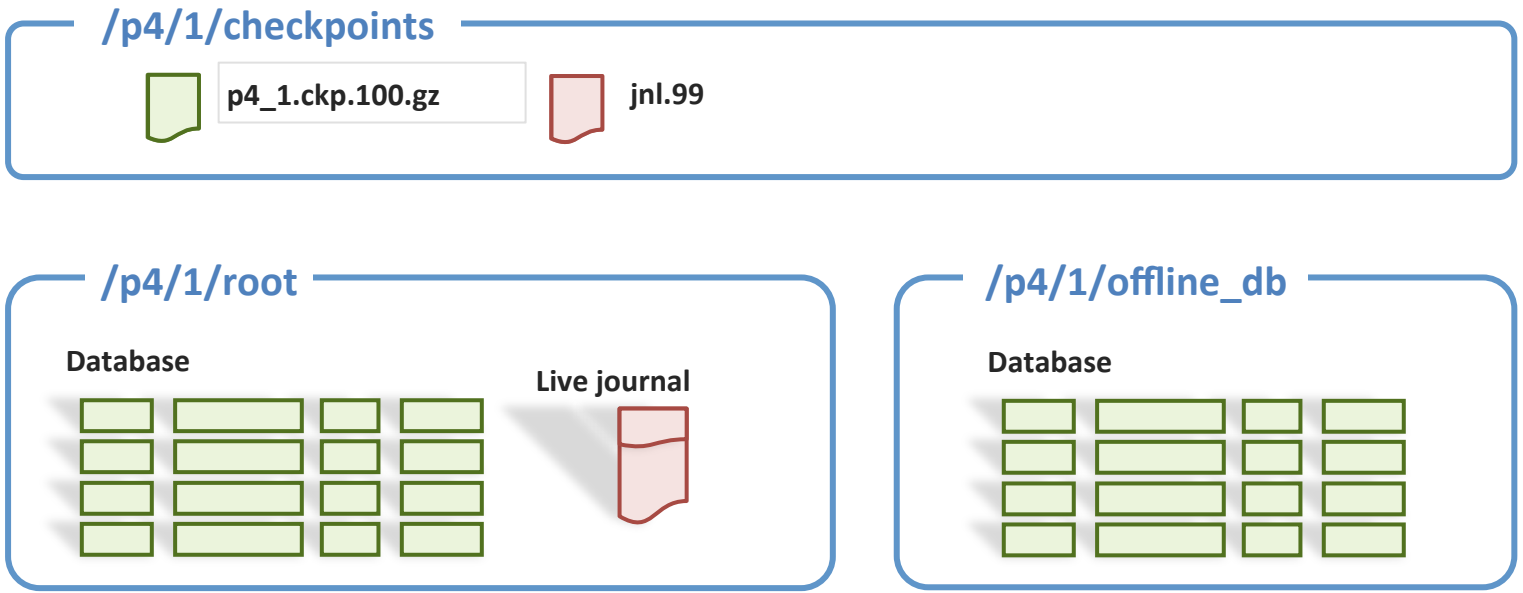
```
p4d -r /p4/1/root -jc -Z /p4/1/checkpoints/p4_1
```





Prep Offline Checkpoint – Apply Seed

```
p4d -r /p4/1/offline_db -jr -z /p4/1/checkpoints/p4_1.ckp.100.gz
```



Offline Checkpoint

PERFORCE
Version everything.

- Nightly:
  - Truncate journal on live database

```
p4d -r /p4/1/root -J /p4/1/logs/journal -jj /p4/1/checkpoints/p4_1
```

  - Restore journal to offline directory

```
p4d -r /p4/1/offline_db -jr /p4/1/checkpoints/p4_1.jnl.100
```

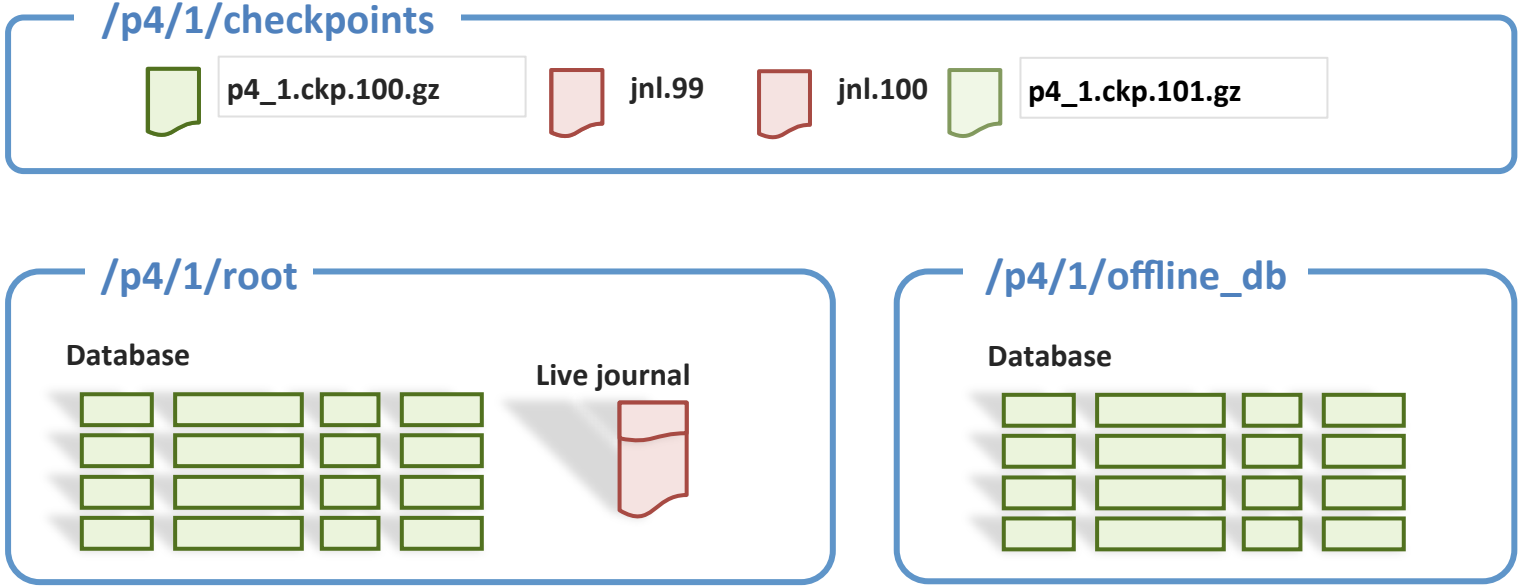
  - Dump the offline database to make a new checkpoint

```
p4d -r /p4/1/offline_db -jd -z /p4/1/checkpoints/p4_1.ckp.101.gz
```

Offline Checkpoint

PERFORCE
Version everything.

Truncate journal `-jj` › Restore journal `-jr` › Dump checkpoint `-jd`



Recreate Offline Database

PERFORCE
Version everything.

- Recreate the offline database from the new checkpoint

```
rm -f /p4/1/offline_db/db.*
p4d -r /p4/1/offline_db -jr -z /p4/1/checkpoints/p4_1.ckp.101.gz
```

Switch Offline Database/Root

PERFORCE
Version everything.

- Stop the production server
- Rotate the journal
- Replay the journal to the offline_db
- Move /p4/1/root/db.* /p4/1/root/save/
- Move /p4/1/offline_db/db.* /p4/1/root/
- Restart the master server
- Dump a checkpoint from /p4/1/root/save
- Recreate the offline database from the new checkpoint

Exercises

PERFORCE
Version everything.

- Lab Set E2: Offline Checkpoints

PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

Broker

P4Broker

PERFORCE
Version everything.

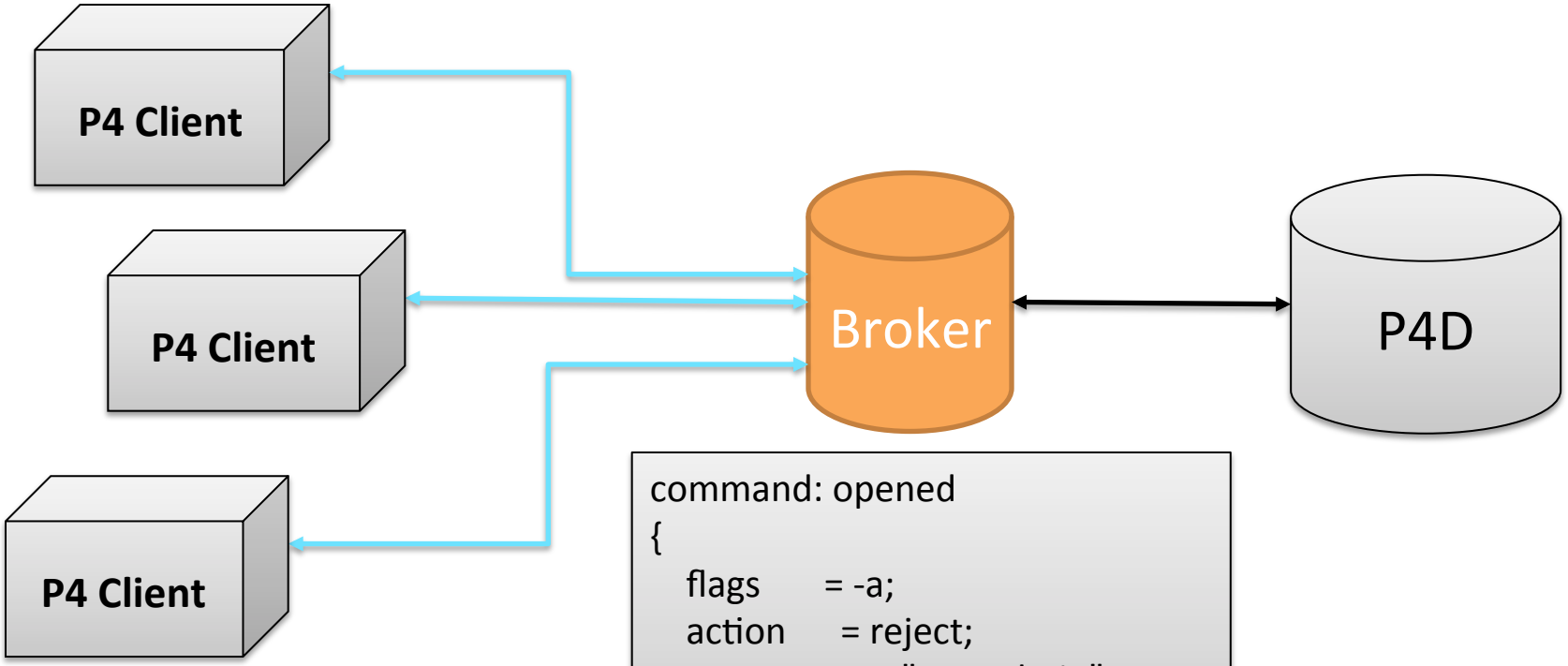
- Intercepts all incoming Perforce commands
- Command handling support:
  - Redirection
  - Blocking
  - Rewriting (undocumented)
- Great for notifying users when the server is down for maintenance.
- Sometimes used as part of HA/DR strategies to avoid DNS change delay.

P4Broker Use Cases

PERFORCE
Version everything.

- Policy Customizations
  - different capabilities than triggers
- Traffic Redirection for Load Distribution
  - not “load balancing”
- Traffic Redirection for execution of automated failover operations
  - advanced/custom usage

Perforce Broker




```
command: opened
{
    flags       = -a;
    action      = reject;
    message     = "Not admin";
}
```

Redirection

PERFORCE
Version everything.

- Selective – The default setting
  - Redirection allowed, but after the first command in a session hits the default server, all others in the same session use the default server and are not redirected.

- Pedantic – All redirected commands are redirected
  - Can cause the GUI to not update the icons correctly.

Read Only Commands

PERFORCE
Version everything.

- Redirect read only commands to replica servers to offload the master server
- P4Broker can do load balancing
  - Use the name “random” for the target server in a redirected command handler.
- http://kb.perforce.com/article/1354 -- Using P4Broker With Replica Servers
- (This particular use case can now be handled with a forwarding replica.)

Filter Scripts

PERFORCE
Version everything.

- When the action for a command handler is “filter”:
  - The broker executes the program or script and performs the action returned by the program.
- The broker invokes the filter program and passes in all the information about the command via stdin.
  - Your filter program must read this data from stdin before performing any additional processing.
- The filter program responds on stdout with one of the following:
  - action: PASS/REJECT/REDIRECT/RESPOND
  - message: Some message for the user

Mechanics: P4Broker Setup

PERFORCE
Version everything.

- Define an operating server.
- Generate a preliminary broker configuration file.
- Adjust the broker configuration to your needs.
- Set broker config file location.
- Initiate as a Windows service or Unix/Linux daemon.
- Documentation:
  - Latest Release Broker Notes
  - Distributing Perforce Manual

Exercises

PERFORCE
Version everything.

- Lab Set E2: P4Broker

PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

Replication - Introduction

Why Replication?

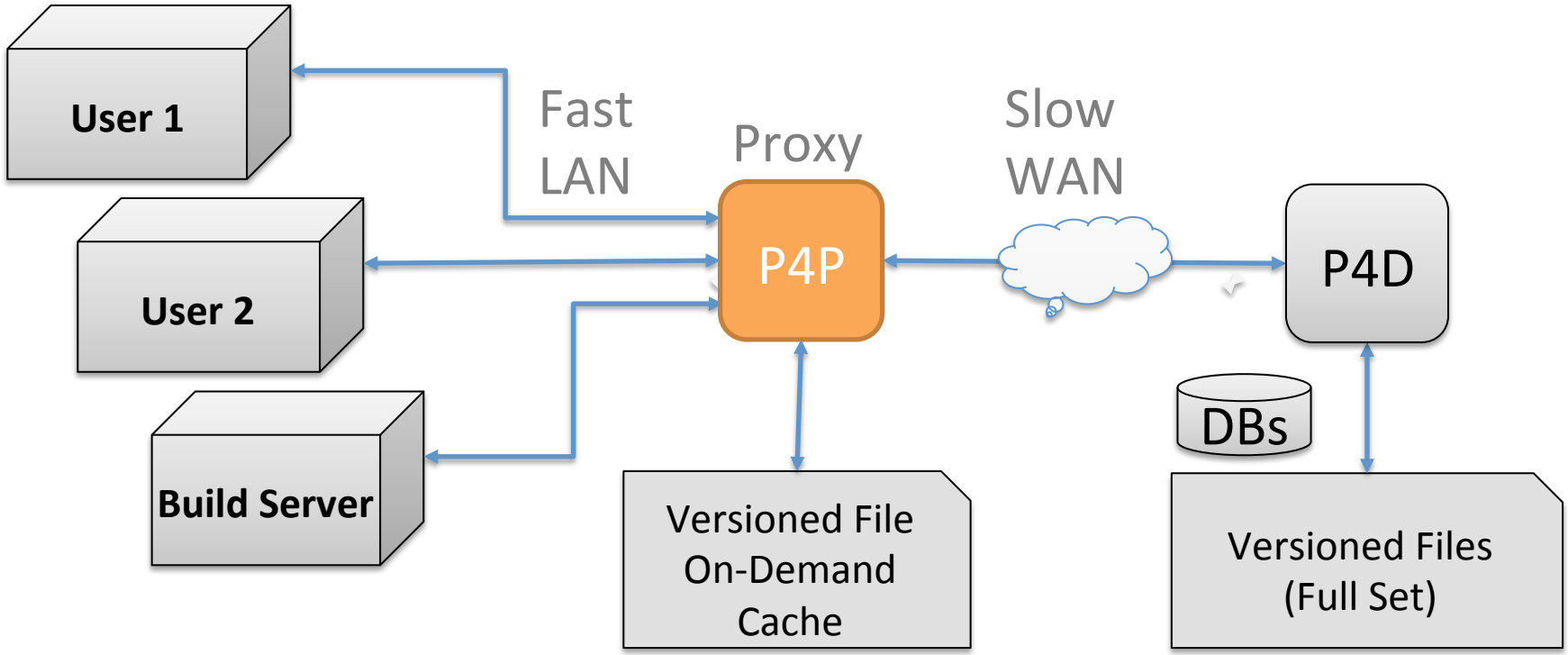
PERFORCE
Version everything.

- Disaster Recovery

- Offloading intense server traffic from:
  - Reports
  - Builds

- Forwarding Replica (aka Smart Proxy)
- Edge / Commit server architecture (distributed working)

Proxy (Review)



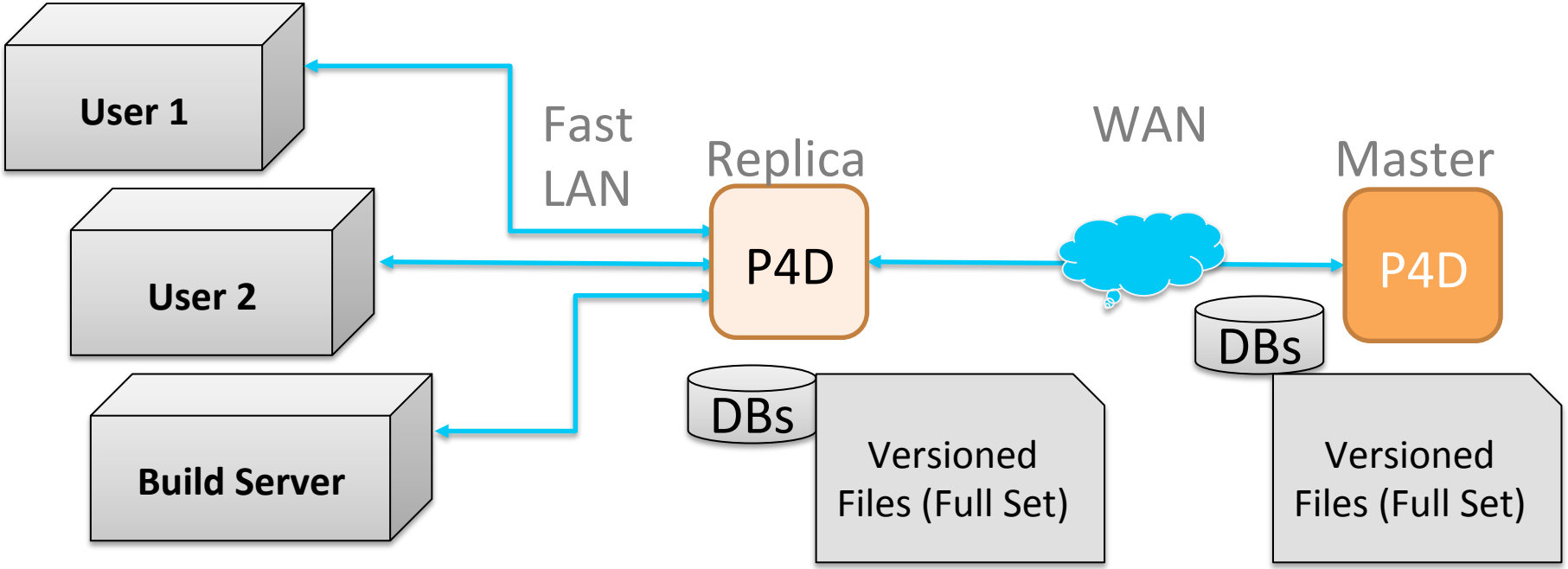
Perforce Replication - Implementation

PERFORCE
Version everything.

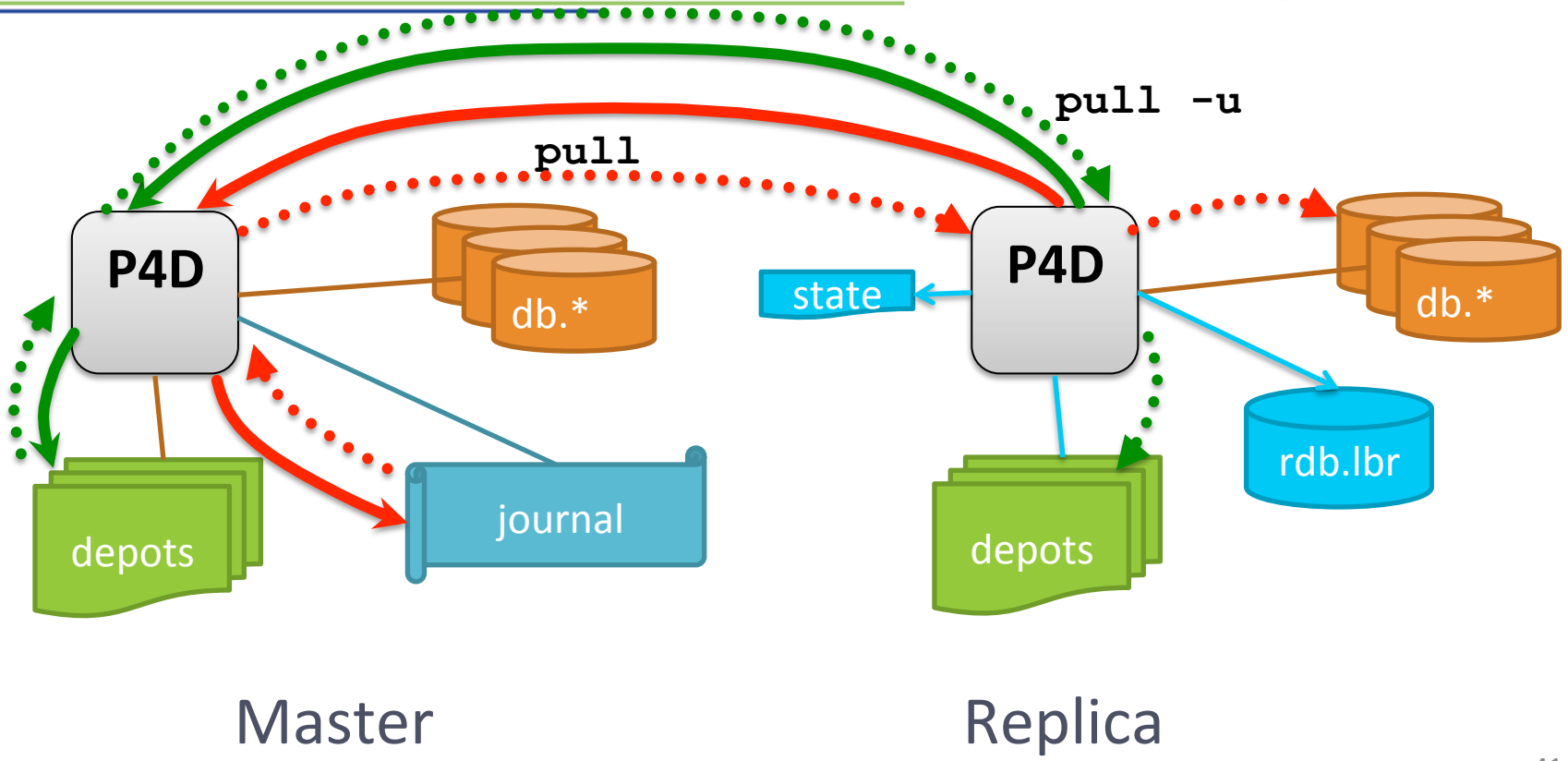
- Server-to-Server replication
  - Asynchronous based on journal file
  - Supports both Metadata-only and Full Replication
  - No need for external scripts, complete solution

- Replica server runs in read-only mode
- Replicas must initially be seeded with a checkpoint (metadata).
  - Versioned files are required for full replication.

Read-Only (Standard) Replica for DR



Replication Architecture



p4 pull

PERFORCE
Version everything.

- Runs as a background task inside the replica

Command	Effect
`p4 pull`	Retrieve missing journal entries, then terminate
`p4 pull -i <N>`	Continuously pull every <N> seconds
`p4 pull -u`	Retrieve missing file revisions, then terminate
`p4 pull -u -i <N>`	Continuously pull file revisions
`p4 pull -l`	List missing file revisions or errors
`p4 pull -l [-j | -s]`	Replica reporting

- Run `pull -lj` from the command line against the replica to check status

How does ‘pull’ keep track?

PERFORCE
Version everything.

- **state** file
  - Text file normally located in the replica P4ROOT directory
  - `journal#/offset`
  - Allows replication to be interrupted
  - Master server can rotate journal file
    - Configure journalPrefix if master uses journal prefix for checkpoints

- **rdb.lbr** database
  - Binary file located in the replica P4ROOT directory
  - Contains information on missing archive revisions

Journal rotation and Prefix

PERFORCE
Version everything.

- **Master**
  - `p4 admin checkpoint/journal [-Z] prefix`
  - Do not use –z, use –Z (uppercase)
    - Compresses checkpoint but not rotated journal file
  - If you use a prefix, use the same prefix for ‘p4 pull’
    - Use ‘journalPrefix’ configurable (or pull -J)
- **Replica**
  - `p4 pull [-J prefix] [-i n]`
  - Journal will be rotated in sync with the master

journalPrefix

PERFORCE
Version everything.

- Specify journalPrefix on the master to
  - Simplify checkpoint and journal rotation
  - Avoid having to specify ‘p4 pull –J prefix’ in the replica(s)

- Specify journalPrefix in the replica to
  - Automatically rotate journal to correct location when master rotates
  - Help to prevent replica running out of disk space
  - Without journalPrefix, replica will rotate journal in P4ROOT

```
p4 configure set replica#journalPrefix=/replica/checkpoints/repl_1
```

Configuration

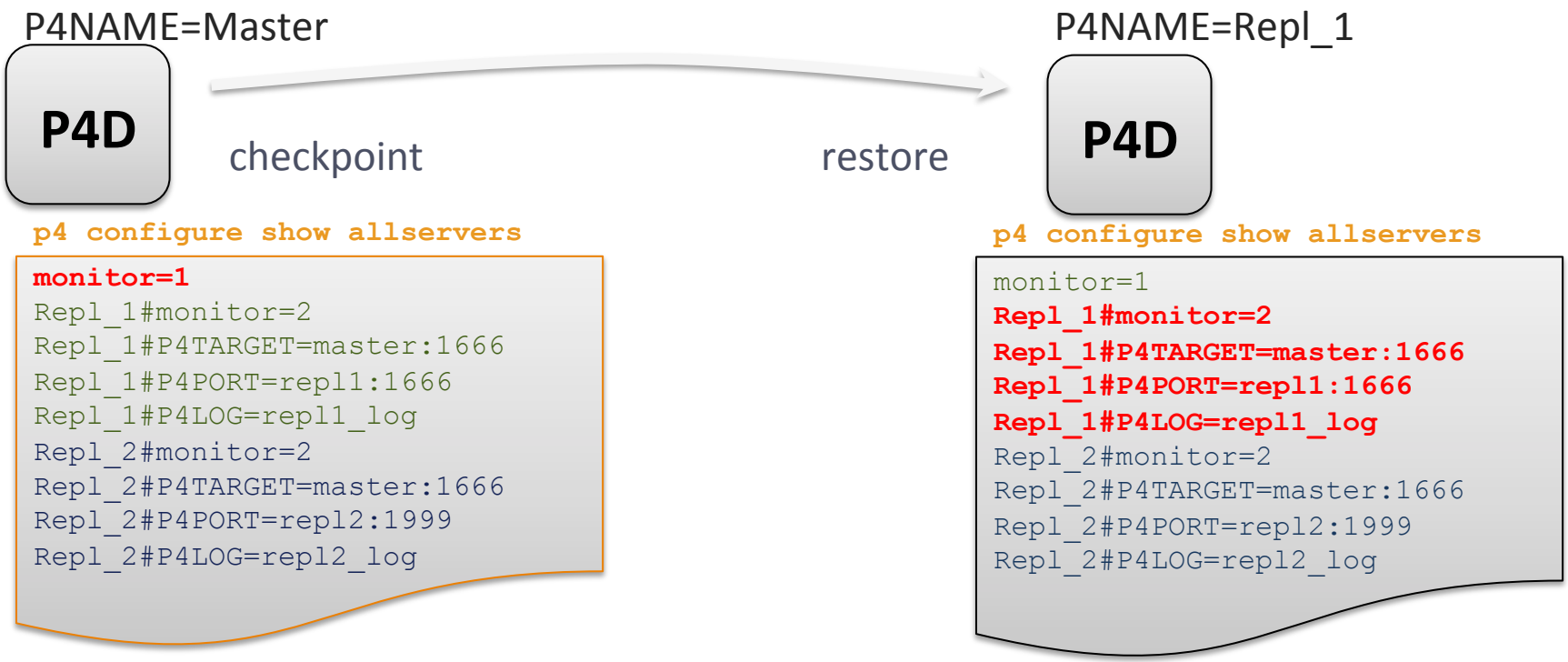
PERFORCE
Version everything.

- ‘p4 pull’ is designed to be a background process
  - Started from the replica server
  - One process for retrieving metadata
  - Several additional processes to retrieve archive data

- Use ‘p4 configure set’ for named servers

```
p4 configure set P4NAME#variable=value
p4 configure set repl_1#statefile=repl1_state
```

Prepare in the Master



P4NAME determines which configuration is active

How to set configurations

PERFORCE
Version everything.

- Command line flags
- Configure set/cset
- Environment variables
- (On Windows) registry variables

Configuration parameters

PERFORCE
Version everything.

Parameter	Sample Values
P4PORT	1666
P4TARGET	master:1666
db.replication	readonly
lbr.replication	readonly/ondemand
serviceUser	replica_1
monitor	1
startup.1	pull –i 1[-J prefix]
startup.2	pull –u –i 1
startup.3	pull –u –i 1

Service user

- Replication requires user of type `service`.
- The service user requires ‘super’ access.
- Add the user to a group (e.g. Automation.G) group with unlimited timeout.

- On replica machine login as the service user before starting replication
  - Define P4TICKETS location for the replica

```
export P4TICKETS=/replica/.p4tickets
p4 -u p4admin login service_user
```

Server and serverid

- `p4 server servername`
  - Creates or updates information about a server
  - Currently optional
    - except for build servers
  - Specifies information about a server such as its type
- `p4 serverid`
  - Displays or sets the serverid
  - Information is stored in a small text file `server.id`

Type	Definition
standard	Standard Server
replica	Replica Server
broker	P4Broker
proxy	Perforce Proxy
forwarding-replica	Smart Proxy
build-server	Build Server
P4AUTH	Authentication Server
P4CHANGE	Change Server

Monitoring

PERFORCE
Version everything.

- `p4 monitor show -a` (on replica)
  - `695 R service    72:22:23 pull -i 1`
  - `696 R service    72:22:23 pull -u -i 1`
  - `697 R service    72:22:23 pull -u -i 1`

- `p4 logtail` (on master, with server=1..3)
  - `rmt-Journal`
  - `rmt-FileFetch`

Active Replication Monitoring

PERFORCE
Version everything.

- `p4 journaldbchecksums`
  - Run on master, check log on replica
- `p4 pull -l [-j|-s]`
  - Reports pending transfers
- `p4 verify [-t]`
  - Option –t schedules content transfer of missing/damaged revision

Checkpointing a Replica

PERFORCE
Version everything.

- Checkpoint against replica is deferred

```
p4 admin checkpoint
The 'pull' command will perform the checkpoint at the next
rotation of the journal on the master.
```

- This keeps replica and master in sync
- Alternative to offline checkpoint
  - Create metadata-only replica
  - Rotate journal on master on regular basis

Commands on the replica

PERFORCE
Version everything.

- Only read-only commands are allowed
  - 'p4 sync -p', 'p4 print', but not 'p4 sync'
- Clients used against the replica must be created on the master server
  - Will be replicated across
- Timestamps do not get updated

Exercises

Lab Set E3: Replication

New commands in this chapter:

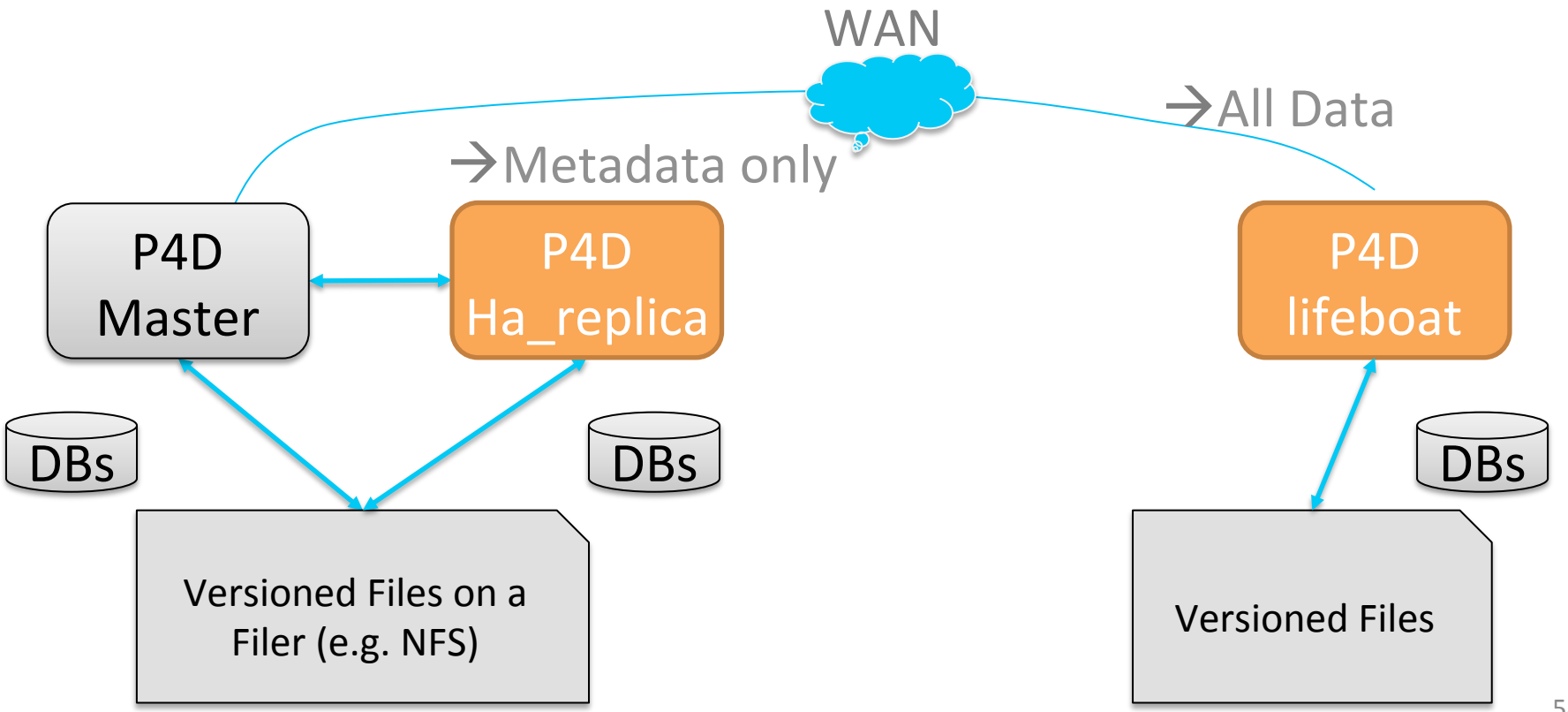
- `p4 configure set P4NAME#var=val`
- `p4 configure show allservers`
- `p4 pull`
- `p4 pull -l [-j | -s]`
- `p4 journaldbchecksums`
- `p4 verify -t`

PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

More Replication Options

Replicas for HA and DR



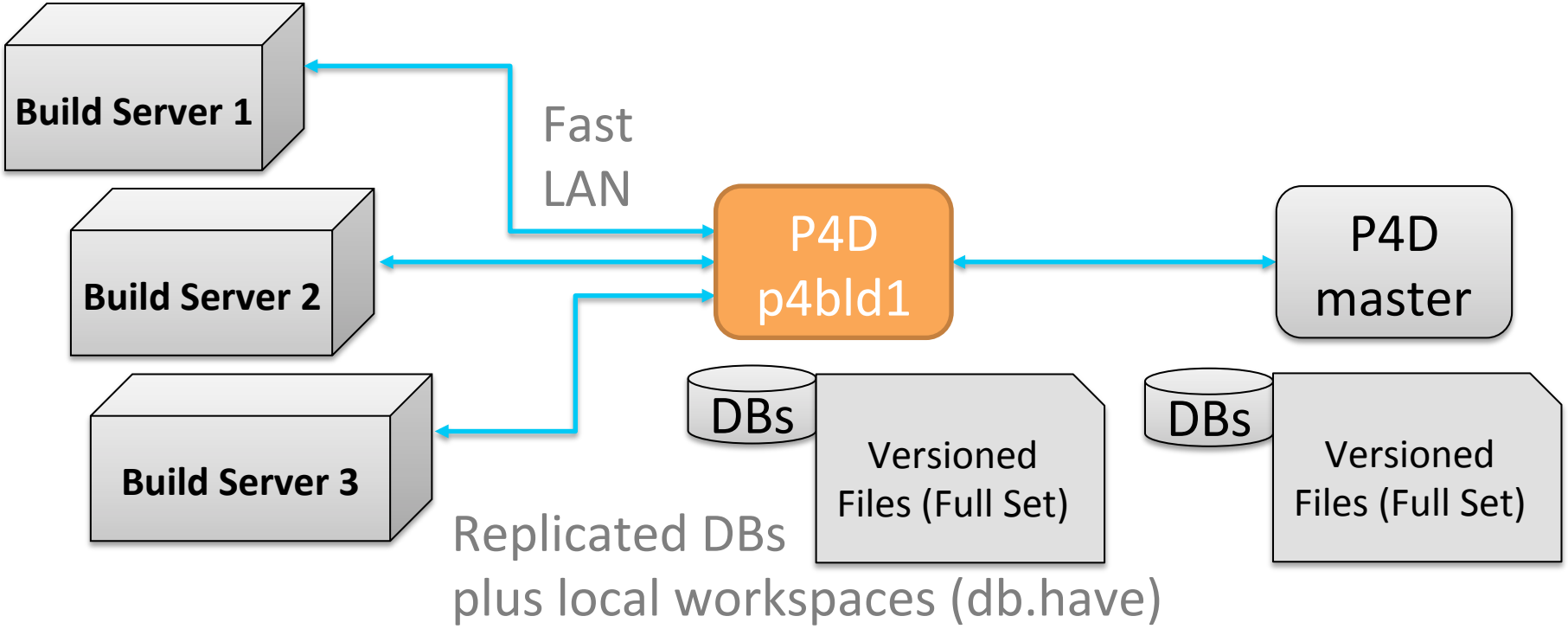
Build Farm Server

PERFORCE
Version everything.

- **Goal**
  - Provide efficient environment for (automated) build/test processes
  - Build servers have their own db.have table
  - Reduces load and storage requirements on the master server

- **Implementation**
  - Type: build-server
  - Needs server entry and serverid defined
  - Clients local to build server need to specify **serverID:** field

Build Farm – Read-Only Replica



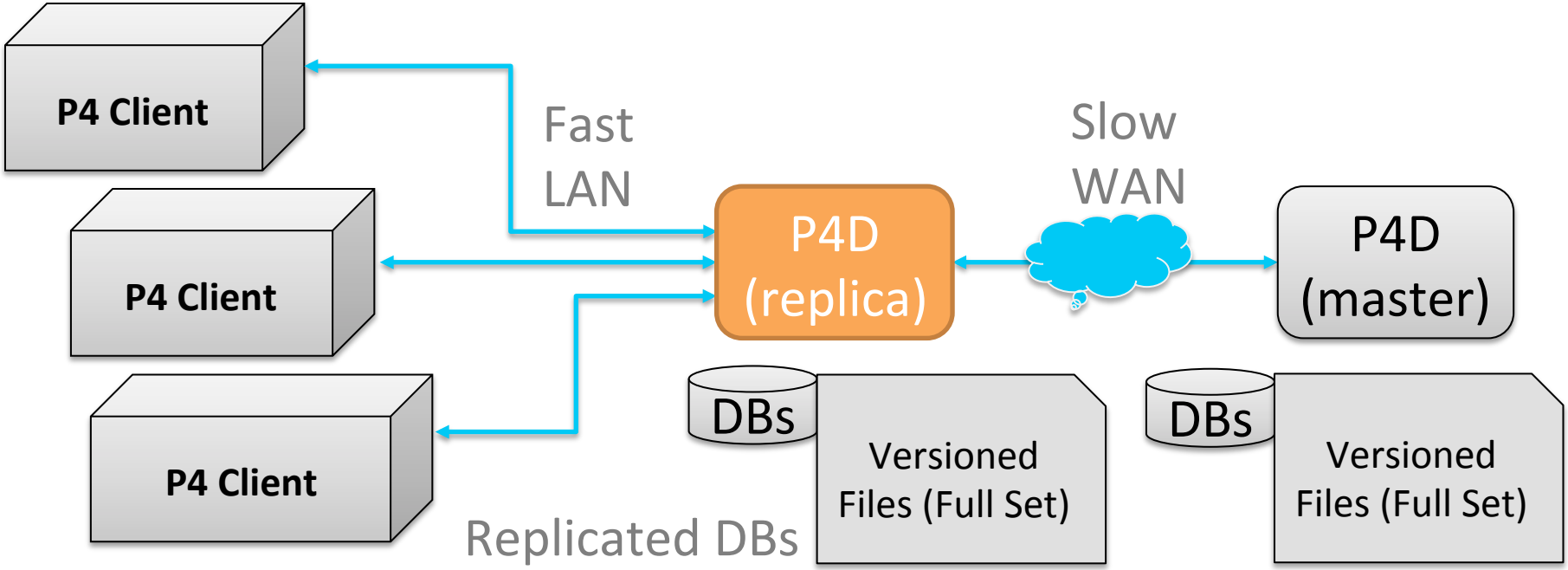
Configuring build workspaces

PERFORCE
Version everything.

```
p4 client build-ws-9201
```

```
Client:build-ws-9201
:
ServerID: Replica1
View:
:
```

Forwarding Replica



Forwarding replica = smart proxy

PERFORCE
Version everything.

- Advantages
  - Metadata is cached locally
  - Read operations much faster locally
  - Write commands allowed – but forwarded to main server

- Disadvantages
  - Backup/recovery of replica

Prepare in the Master

PERFORCE
Version everything.

```
p4 server Replica1

ServerID: Replica1
Name: Replica1
Type: server
Services: forwarding-replica
```

```
p4 configure set Replica1#rpl.forward.all=1
```

Replica filtering

- To exclude entire tables from a replica:

```
p4 pull -T db.have,db.client
```

- Detailed Filtering:

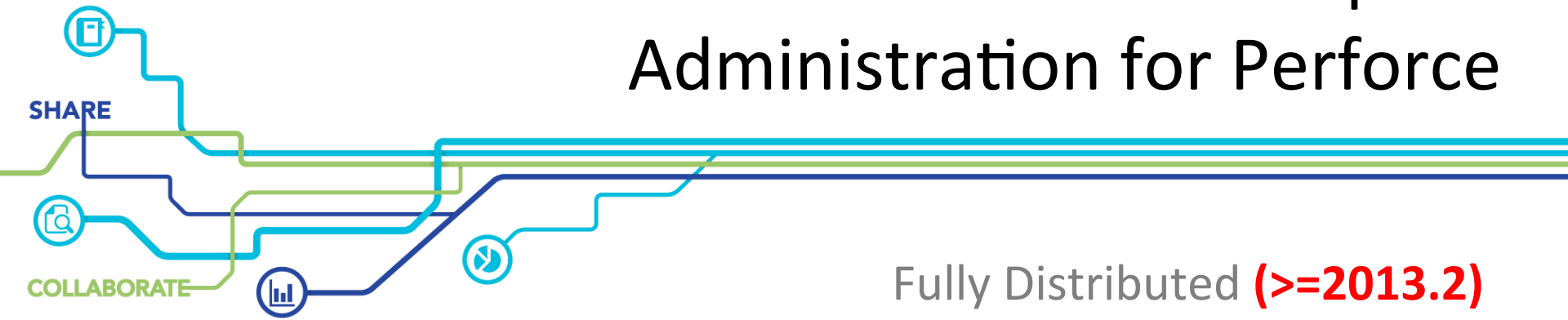
```
p4 server Replica1
ServerID: Replica1
:
ClientDataFilter:
  -//site2-ws-*
ArchiveDataFilter:
  //....c
  -//....mp4
```

```
p4 configure set
"Replica1#startup.1=pull -i
30 -P Replica1"
```

Exercises

Lab Set E5: Forwarding Replication

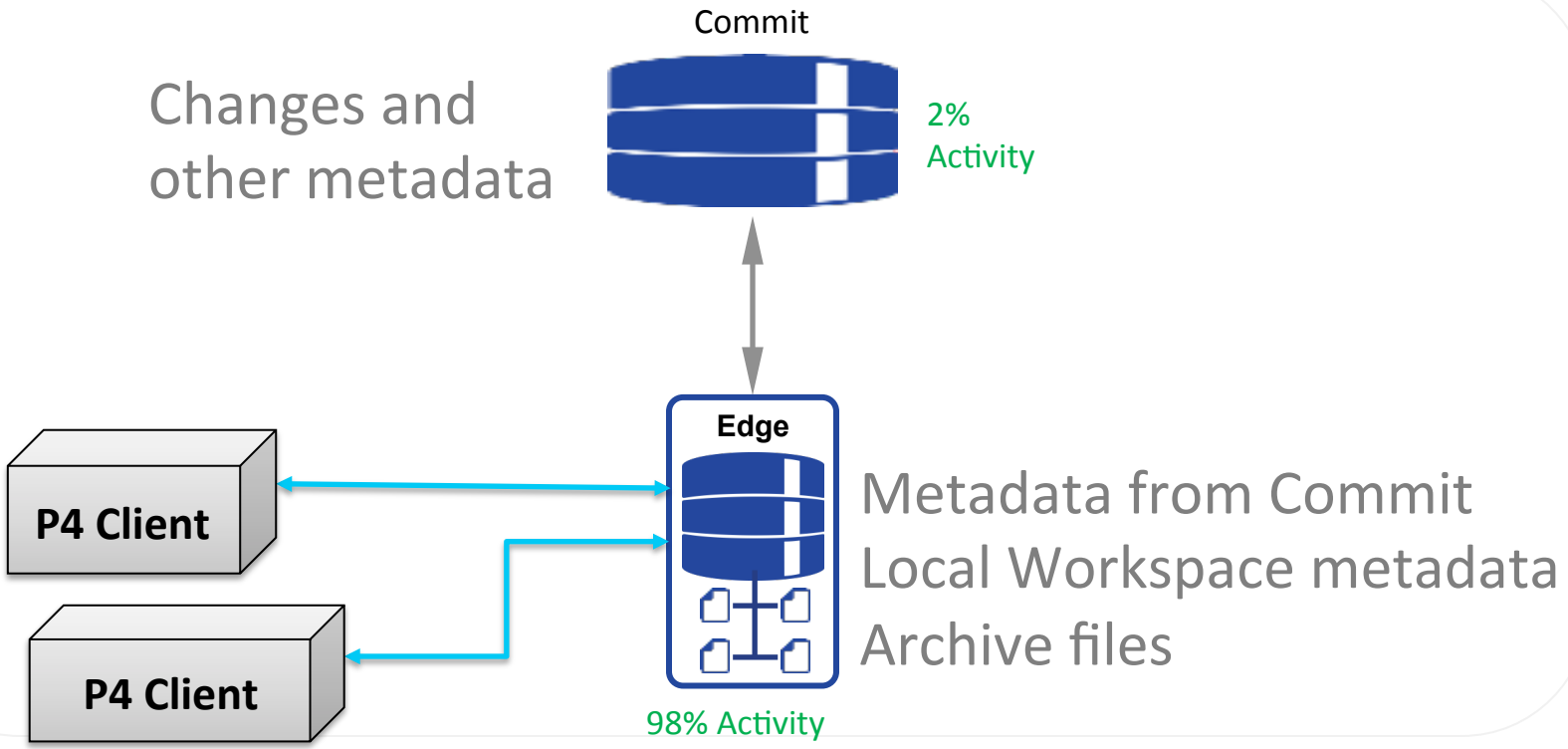
PERFORCE
Version everything.



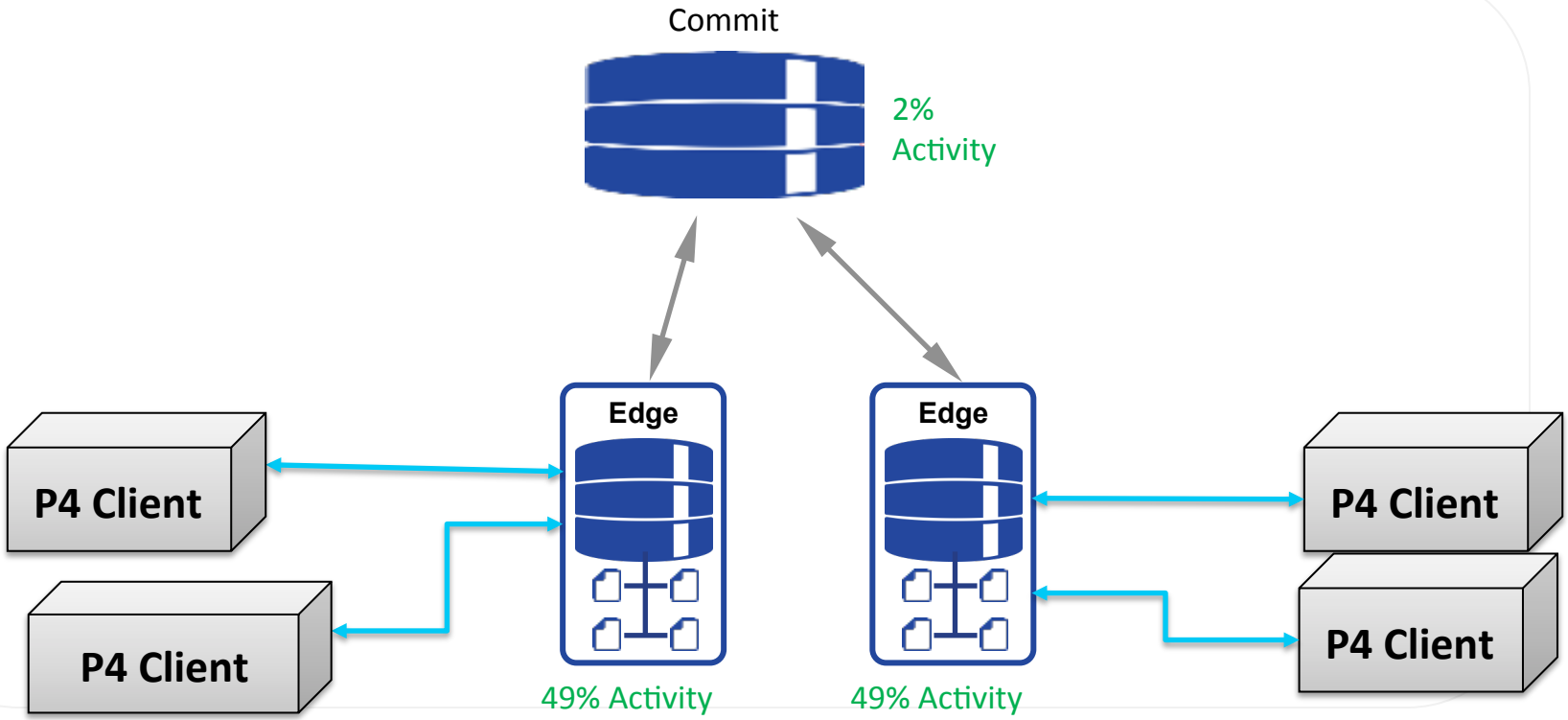
Advanced Enterprise Administration for Perforce

Fully Distributed **(>=2013.2)**

Edge/Commit Server Architecture

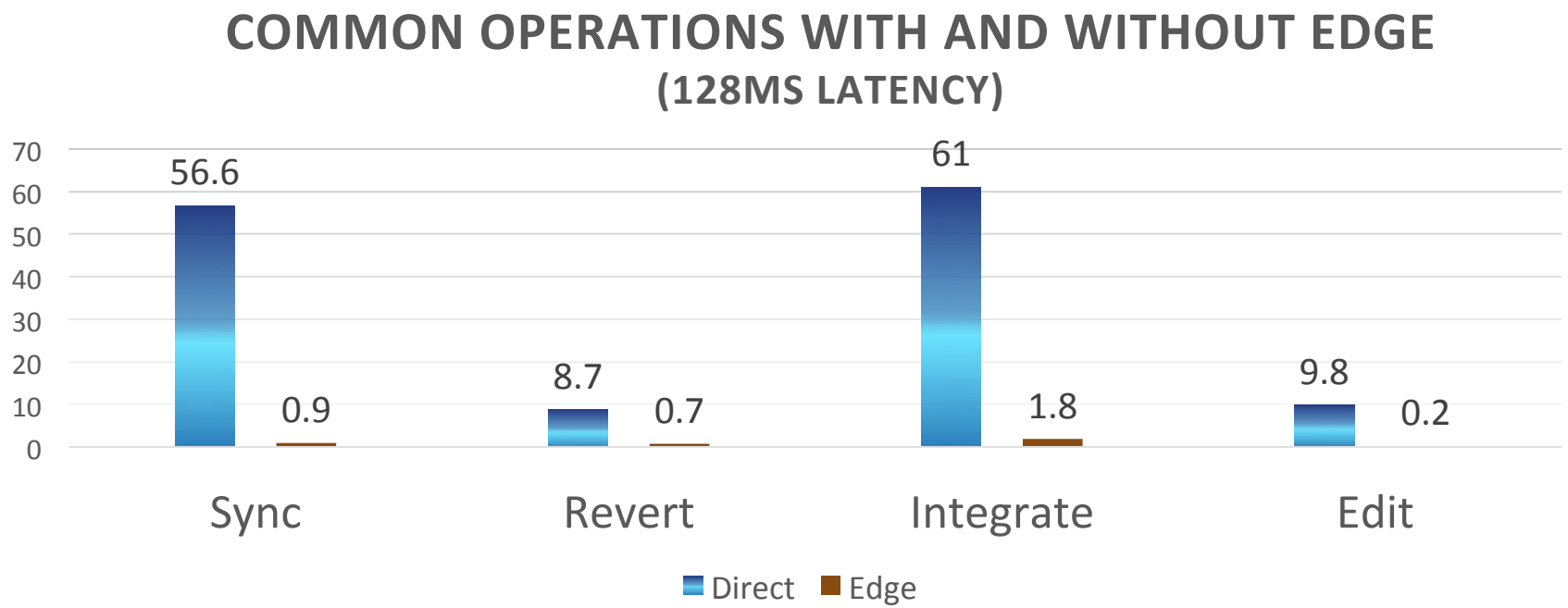


Edge/Commit Server Architecture



Results

PERFORCE
Version everything.



Perforce Lab Benchmarks

Edge/Commit Considerations

PERFORCE
Version everything.

- Edge servers require backup/recovery
- Information is distributed – you may need to interrogate all edge servers
- Forwarding replicas are simpler and address many needs…

PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

Security

Server Security

PERFORCE
Version everything.

- Server security levels (0-3)
  - p4 configure set security=3
- Turn off auto user creation
  - p4 configure set dm.user.noautocreate=2
- Set changelists to restricted by default
  - p4 configure set defaultChangeType=restricted

Connection Protocols

PERFORCE
Version everything.

- TCP
  - Default protocol

```
P4PORT=tcp:p4server:1666
```

- RSH
  - Starts up the server for each request
  - Useful for testing and inetd support

```
P4PORT=rsh:/usr/local/bin/p4d -r $P4ROOT -L $P4LOG -i
```

- SSL
  - SSL encrypted connection when using “ssl:” prefix

```
P4PORT=ssl:p4server:1667
```

RSH connection

PERFORCE
Version everything.

- Starts up a server on client request
- No TCP/IP connection to server
  - Uses stdout/stdin bound to client (with -i option)

- Usage examples:
  - Sidetrack server (specify different log file)
  - Test environments (P4Python, P4Ruby, P4Perl)

SSL Encryption

PERFORCE
Version everything.

- 2012.2+ release support SSL encryption
  - Perforce Server, Perforce Proxy, P4Broker
  - Requires 2012.2+ client (P4, P4V, …)
    - Consider implications with 3rd party integrations
  - If enabled, all clients require SSL connection.
    - Run two P4Ds to offer SSL and non-SSL (one with “ssl:”, one without)
- Client needs fingerprint in its P4TRUST file

```
p4 trust
```

p4 trust

- Client-side command for handling fingerprints
- Uses P4TRUST environment variable (default $Home/.p4trust)

```
p4 trust -h
```

`p4 trust -y`	Accept the fingerprint
`p4 trust -n`	Reject the fingerprint
`p4 trust -f`	Force overwriting of the fingerprint
`p4 trust -l`	List accepted fingerprints
`p4 trust -d`	Delete a fingerprint

SSL Setup

PERFORCE
Version everything.

- P4SSLDIR -> directory with key and certificate

```
cd $P4ROOT
mkdir ssl        # optionally create config.txt
chmod 700 ssl    # drwx------
export P4SSLDIR=ssl
p4d -r . -Gc     # key and certificate
p4d -r . -p ssl:1667
```

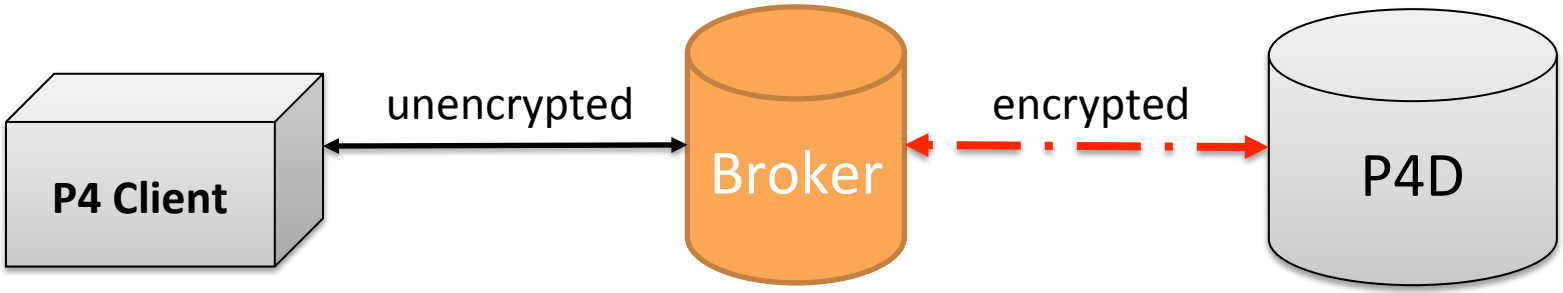
- Client needs to accept fingerprint

```
p4 -p ssl:p4server:1667 trust -y
```

Phasing-in SSL encryption with P4Broker

PERFORCE
Version everything.

- Use P4Broker
  - P4D runs with SSL encryption enabled
  - P4Broker itself runs unencrypted
  - Allows phasing-in of encrypted connections



Exercises

Lab Set E6: Security

New commands in this chapter:

- `p4d -Gc`
- `p4 trust`

PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

Advanced Tools

Advanced Tools

PERFORCE
Version everything.

- perfmerge
- perfsplit
- p4-migrate
- Checkpoint surgery
- Conversions
- ftp://ftp.perforce.com/perforce/tools

perfmerge

PERFORCE
Version everything.

- **Goal**
  - Merge two Perforce Servers into a single Perforce Server

- **Implementation**
  - Perfmerge tool reads both databases
  - Choice on change merging
    - Append
    - Intersperse and order in time
    - Append with offset

perfsplit

PERFORCE
Version everything.

- **Goal**
  - Extract data from a main server with its exact revision history
  - Split a Perforce Server into two separate Perforce Server

- Implementation
  - Perfsplit reads directly from an existing Perforce Server
  - It uses a splitmap to determine which files are split
    - Same syntax as the label view map
  - Only creates metadata, depot files need to be copied separately

p4migrate

PERFORCE
Version everything.

- **Goal**
  - Migrate a Perforce Server from a case-insensitive to a case-sensitive platform

- **Implementation**
  - Reads a checkpoint to find case inconsistencies
  - Generates a case-fix map
  - Use the map to correct the checkpoint
  - Once the checkpoint is case-consistent it can be used for migration
  - Tool can also be used to rename depot paths
  - Migration from case-sensitive to case-insensitive is not supported

Checkpoint/Journal Format

PERFORCE
Version everything.

- Text file containing journal records
- Each record has a type
  - Checkpoint only has @pv@ entries
- Strings are surrounded by @ symbol
- Each value record refers to
  - A database table
  - The table version

Record	Type
@pv@	Put value = insert
@dv@	Delete value = delete
@rv@	Replace value = update
@vv@	Verify value = select
@ex@	commit
@mx@	flush
@nx@	Journal note

- http://www.perforce.com/perforce/doc.current/schema/

Log Analysis and Reporting

PERFORCE
Version everything.

- Standard Log
  - Log Analyzer
    - Upload your logs
    - Download our tools
  - Track2SQL
- Structured Logs
- Performance monitoring using the log
- Metrics with P4toDB (replication technology)
- Discovering overall trends

Structured Log

PERFORCE
Version everything.

#	Structured Logs	Description
1	`all`	All loggable events (commands, errors, audit, etc...)
2	`commands`	Command events (command start, compute, and end)
3	`errors`	Error events (errors-failed, errors-fatal)
4	`audit`	Audit events (audit, purge)
5	`track`	Command tracking (track-usage, track-rpc, track-db)
6	`user`	User events; one record every time a user runs p4 logappend.
7	`events`	Server events (startup, shutdown, checkpoint, journal rotation, etc.)

Structured Logs

- Enable specific structured logs with:

```
p4 configure set serverlog.file.n=logtag.csv
p4 configure set serverlog.maxmb.n=1024
p4 configure set serverlog.retain.n=45
```

- Enabling all structured logging files can consume considerable space and impact performance.
- Structured logs are automatically rotated on checkpoint, journal rotation, exceeding size limit, or when ‘p4 logrotate’ is run.

Conclusion

PERFORCE
Version everything.

- Database schema is public
- There are tools that can use the checkpoint or the database directly
- Handle with care

- Ask Perforce support or consulting if you are not sure

Exercises

Lab Set E7: Structured Logs

New commands in this chapter (samples):

- `p4 configure set serverlog.file.n=errors.csv`
- `p4 configure set serverlog.maxmb.n=30Mb`
- `p4 configure set serverlog.retain.n=45`
- `p4 logappend`
- `p4 logrotate`

PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

Scripting

Preliminary Decisions

PERFORCE
Version everything.

- Uses of scripts
- Choosing the interface
- Setting Environment Variables
- User Authentication

Uses of scripts

PERFORCE
Version everything.

- Reporting tools
- Daemons and recurring processes
- Wrappers for Perforce commands
- Triggers
- Workflow and policy enforcement
- P4V customization (P4JsAPI)
- P4Broker
- Legacy SCM data import

Typical workings of a script

PERFORCE
Version everything.

- Data processing in batches
  - Retrieve information such as files or changes
  - Process the data in the script
  - Potentially update Perforce

- Form handling
  - Retrieve a form such as a client workspace
  - Modify the form in the script
  - Update the form in Perforce

Workflow and Policy enforcement

PERFORCE
Version everything.

- Triggers
  - Submit/Shelving triggers
  - Authentication triggers
  - Form triggers
  - Archive triggers
  - Fix triggers

- P4Broker
  - Block, redirect or modify commands

Choosing the interface

PERFORCE
Version everything.

- Wrap P4 command
  - \+ Simple solution that will run everywhere
  - \+ Batch scripting built into the OS and requires no installation
  - – Requires parsing of output
- APIs
  - \+ Language-specific integration
  - \+ Extendable
  - \+ Performance (reduced connection overhead)
  - – Requires installation (and/or build/compilation)

API's Available for Scripting

PERFORCE
Version everything.

- Programming Languages
  - C++
  - P4Java
  - Perforce Objective-C
  - Perforce .NET
- Derived APIs (C++ API wrappers)
  - P4Python
  - P4Perl
  - P4Ruby
  - P4PHP

http://www.perforce.com/product/components/apis

Wrapping the command line client P4

PERFORCE
Version everything.

- Command line returns lines of text

```
p4 describe -s 13
Change 13 by sknop@alita on 2010/03/02 12:58:51

 Branching foo from bar.
 Test branch only.

Affected files ...

... //depot/tests/foo#1 branch
```

Capture errors, warnings and messages

PERFORCE
Version everything.

- Use -s to precede each output line with “info” or “error”

```
p4 -s sync ...
info: //depot/foo#3 - updating /client/foo
error: Can't clobber writable file /client/foo
exit: 1
```

Tagging output: Command line and API

PERFORCE
Version everything.

- Format output by using -ztag

```
p4 -ztag clients
... client bruno_ws
... Update 1104271684
... Access 1104340062
... etc.
```

- Perforce API based on tagged data output

Form handling: bypassing an editor

PERFORCE
Version everything.

- Redirect to standard output

  ```
  p4 change -o
  ```

- Read from standard input

  ```
  p4 submit -i
  ```

- Submit without invoking an editor

  ```
  p4 submit -d "Fixed off-by-one error."
  ```

- Example: Create a client workspace without invoking an editor

  ```
  p4 client -o | p4 client -i
  ```

Setting the environment for scripts

PERFORCE
Version everything.

- Command line flags

```
p4 -p server:1666 -u script_user -c script_ws info
```

- P4CONFIG (next slide)
- Environment and registry variables

- Recommendation:
  - Use P4CONFIG
  - Set P4CONFIG in the scripts to make sure it is set in the environment
  - This will keep scripts independent of Perforce Server and location

P4CONFIG

PERFORCE
Version everything.

- P4CONFIG points to a file name

```
p4 set P4CONFIG=P4Config.txt
export P4CONFIG=/p4/scripts/.p4config
```

- File usually located in the workspace root or scripts folder

- File contains the Perforce variables

```
P4PORT=server:1666
P4CLIENT=script_ws
P4USER=script_user
```

User authentication for scripts

PERFORCE
Version everything.

- `p4 login`
  - Works for all Perforce Server security levels
  - Works if Perforce is integrated with AD
  - Works if Perforce is integrated with SSO

- Either: Store password in local (hidden/restricted access) file

```
p4 login < /p4/scripts/.password
```

- Or: Use ever-lasting ticket (ideally with separate P4TICKETS file)

Use a group to extend session

PERFORCE
Version everything.

```
p4 group scripts
```

```
Group:    scripts
MaxResults:     1000000
Maxscanrows:    5000000
MaxLockTime:      30000
Timeout:      unlimited
Subgroups:
Owners:
                bruno
Users:
                script_user
```

P4TICKETS

- P4TICKETS points to a ticket file

```
export P4TICKETS=/p4/scripts/.script_p4tickets
```

- Important when scripts may be run as a different user (default value is home directory which is different per user)
- This will provide safety from someone accidently logging out a script user
  - Beware of p4 –u script_user logout –a
    - Invalidates all tickets for this user

Scripts and protection table

PERFORCE
Version everything.

- Guard Server-local script users through protection table
- Prevents access from any other client machine

`p4 protect`

```
Protections:
   super group scripts 127.0.0.1  //...
```

Spoof a user

PERFORCE
Version everything.

- As super user only

```
p4 login username
```

- Creates a local ticket (beware: overwrites P4PASSWD registry entry)
- Does not prompt for password
- Useful for import scripts, triggers, ...

```
p4 -u username command
p4 -u username logout
```

Exercises

Lab Set E8: Basic command automation

PERFORCE
Version everything.

Advanced Enterprise Administration for Perforce

Scripting – Language wrappers
Derived APIs

Derived APIs introduction

PERFORCE
Version everything.

- P4Perl
- P4Python
- P4Ruby
- P4PHP

- Thin layer on top of the C++ API – language specific
- Faster and easier to use than command line wrappers
  - Same commands as if run from the command line ...
  - ... but with built-in parsing
  - Compatible with newer versions of the server

Derived APIs

PERFORCE
Version everything.

- Each API defines a P4 class
- Interface is “identical” for all four products
- Build from source code

- Released together with Perforce Server
- Get version string via P4.identify():

  Rev. P4PYTHON/LINUX26X86_64/2011.1/284414 (2011.1/284414 API) (2011/01/28).

Example (P4Python)

PERFORCE
Version everything.

```
from __future__ import print_function    # Python 2/3 compatibility
import P4
p4 = P4.P4()
p4.port = "1666"
p4.connect()
for user in p4.run("users"):
   print("Hello %s" % user["User"])
p4.disconnect()        # Automatic if script exits
```

Hello Anna_Schmidt

Hello Aruna_Gupta

Hello bruno

Hello dai

Etc…

Using Language-specific output

- Create Python dictionary object

```
p4 -G job -o ## or use -Mg
```

- Create Ruby dictionary object

```
p4 -R info    ## or use -Mr
```

- Create PHP dictionary object

```
p4 -Mp info
```

Sample Python Script and p4 -G Output

PERFORCE
Version everything.

```
#!/usr/bin/env python
#Example script named readmarshal.py

import marshal, sys
try:
      while True:
            vars = marshal.load(sys.stdin)
            print vars
except EOFError:
         '' #note that these are two single-quotes
```

```
p4 -G user -o raj | readmarshal.py
{'Email': 'raj@p4demo.com', 'Update': '2005/12/06 11:16:30', 'Reviews1':
'//depot/dev/main/...', 'Reviews0': '//depot/www/...', 'FullName': 'Raj
Bai', 'User': 'raj', 'code': 'stat', 'Access': '2006/07/06 15:16:30'}
```

Sample Ruby script and p4 -R output

PERFORCE
Version everything.

```
# Example script named readruby.rb
f = IO.popen("p4 -R user -o " + ARGV[0])
user_info = Marshal.load(f)
p user_info
```

```
readruby.rb bruno
{"code"=>"stat", "User"=>"bruno",
"Email"=>"bruno@p4demo.com",
"FullName"=>"Bruno Batswan"}
```

Installing and Invoking the API

PERFORCE
Version everything.

- Distributed in source code
  - Needs P4API (C++ interface) to compile
  - Follow the release notes to build and install
- Windows binaries on Perforce website
  - 32/64 bit and language version dependent
    - Python 2.6, 2.7, 3.1 – 3.4
    - Ruby 1.8, 1.9, 2.0
    - Perl 5.6 – 5.18
- Once installed, import the P4 module into your script (language specific)

P4 Object

PERFORCE
Version everything.

- Needs to be created first
- Represents a connection to the server
  - connect() method establishes connection
  - Connection stays open until disconnect()
- Central method is run()
  - Returns arrays of strings or dictionaries
  - Errors and Warnings become exceptions (if supported)
- Environment can be defined via attributes
  - port, user, client ...

Environment settings

PERFORCE
Version everything.

- Derived APIs pick up connection parameters from the environment

- Usual order of precedence applies:
  - Directly defined attributes
  - P4CONFIG
  - Environment variables, registry, defaults
- Attribute p4config_file (read only)

- Most attributes can be overwritten

Attributes (Type String)

PERFORCE
Version everything.

Name	Description
port	P4PORT
user	P4USER
client	P4CLIENT
charset	P4CHARSET
host	P4HOST
cwd	Current working directory
password	P4PASSWD
ticket_file	P4TICKETS
prog	The name of the application (monitor and log)
version	The version of the application (monitor and log)

P4Perl: use SetXxx() and GetXxx(), respectively

Attributes (Type Integer)

PERFORCE
Version everything.

Name	Description
api_level	Lock output format to specific client level
tagged	Whether to use tagged output (explained later)
maxresults	Overrides maxresults from group spec
maxscanrows	Overrides maxscanrows from group spec
maxlocktime	Overrides maxlocktime from group spec
exception_level	When to throw exceptions (explained later)
server_level	Server level (Read only)
debug	Debug level for additional output from the script
track	Enable tracking of commands
streams	Whether to enable streams-specific output (2011.1)

P4Perl: SetApiLevel(), GetApiLevel()

Connecting to the Perforce Server

PERFORCE
Version everything.

- Need to set P4PORT before connecting
  - Cannot be changed until disconnected
- P4.connect() establishes connection
- Connected until script terminates or ...
- P4.disconnect() closes connection

Example (P4Perl)

PERFORCE
Version everything.

```
use P4;
my $p4 = new P4;
$p4->SetPort( "1666" );
$p4->Connect() or die ("connect");
for my $user ($p4->Run("users")) {
  print "Hello $user->{ 'User' }\n";
}
$p4->Disconnect();
```

Example (P4Ruby)

PERFORCE
Version everything.

```
require "P4"
p4 = P4.new
p4.port = "1666"
p4.connect
p4.run("users").each { |user|
  puts "Hello #{user["User"]}"
}
p4.disconnect
```

Example (P4PHP)

PERFORCE
Version everything.

```
<?php
$p4 = new P4();
$p4->port = "1666";
$p4->connect();
foreach($p4->run("users") as $user)
  print "Hello {$user['User']}";
$p4->disconnect();
?>
```

The Run(command, args) method

PERFORCE
Version everything.

- Args is a list, not a single string
  - `["-m1", "-c", "myws"]`, not `"-m1 -c myws"`
- Returns
  - Array of hash dictionaries (tagged mode)
  - Array of strings (untagged mode)

- Throws a P4Exception (Python/Ruby/PHP)
  - Perl has to check errors and warnings

Example: P4.run() command

PERFORCE
Version everything.

- P4.run("users", "-a", "-l")

User	antony
Fullname	Anthony Alpha
Email	anthony@acme.com

User	bruno
Fullname	Bruno Batswan
Email	bruno@acme.com

...

User	zig
Fullname	Zig Zichterman
Email	zig@acme.com

Error Handling

PERFORCE
Version everything.

- P4.exception_level determines severity:

Level	Name	Description
0	RAISE_NONE	No exceptions thrown
1	RAISE_ERRORS	Only errors are thrown
2	RAISE_ALL	Errors and warnings are thrown

- Default level is 2 (RAISE_ALL)
- P4.errors, P4.warnings and P4.messages
  - Attributes of type list (array)

Catching exceptions

PERFORCE
Version everything.

```
try: # Python
    p4.run("obliterate", "foo")
except P4.P4Exception as e:
    print( e )

begin # Ruby
    p4.run("obliterate", "foo")
rescue P4Exception => e
    p e
end
```

Error Handling and exception_level

PERFORCE
Version everything.

```
p4.exception_level = P4.RAISE_ALL
p4.run("sync")
```

- Exception [Warning]: 'File(s) up-to-date.'

```
p4.exception_level = P4.RAISE_ERRORS
p4.run("sync")
print p4.warnings, p4.messages
```

- ['File(s) up-to-date.']
- [[Gen:17/Sev:2]: File(s) up-to-date.]

Error handling in P4Perl

PERFORCE
Version everything.

```
if ( $p4->ErrorCount() ) {
    foreach my $str ( $p4->Errors() ) {
        print("ERR: Error message: $str \n");
    }
}
elsif ( $p4->WarningCount() ) {
    foreach my $str ($p4->Warnings()) {
        print("WARN: Warning messages: $str \n");
    }
}
```

Tagged and Untagged mode

PERFORCE
Version everything.

```
p4.run("counter", "change")
        [{'counter': 'change', 'value': '702'}]
p4.tagged = False
p4.run("counter", "change")
        ['702']
```

- Untagged mode is useful for a few commands
  - `p4 diff2` or `p4 describe` to see differences

Fixing the API level

PERFORCE
Version everything.

- Client protocol changes for each release
- Server and clients can be updated independently
- Fix the API level if scripts need to rely on server output

```
p4.api_level = 72
```

- Needs to be set before first connection!

Generated and overloaded Run methods

PERFORCE
Version everything.

- Dynamically generated Run methods
  - Python/Ruby: **run_xxx()** => **run("xxx")**
  - Perl: **RunXxx()** => **run("xxx")**
- Some of these methods are overloaded:

run_filelog()	Returns DepotFile[]
run_login()	Takes p4.password as input
run_password(old, new)	Sets the password w/o prompting
run_print()	Combines print chunks together
run_submit(), run_shelve()	Can take change form

  - Original behaviour with p4.run("command")

Details on run_login and run_password

PERFORCE
Version everything.

```
p4.password = "MySecret"
p4.run_login() # uses p4.password

p4.input = "MySecret"
p4.run("login")

p4.run_password("MySecret", "Hidden")
p4.run_password("", "FirstPassword")
p4.run_password("OldPassword", "")
```

Special methods for form handling

PERFORCE
Version everything.

Method	Description
fetch_<form>	Equivalent to run("<form>", "-o")[0]
save_<form>	Equivalent to run("<form>", "-i") with set input
parse_<form>	Parse a text document and convert it into a hash dictionary
format_<form>	Format a hash dictionary into a text document
delete_<form>	Equivalent to run("<form>", "-d")

- Forms are of type P4.Spec
  - Subclass of hash dictionary
- Special access methods for values

Form examples (P4Python)

PERFORCE
Version everything.

```
cl = p4.fetch_client("myws")
cl._options = \
   cl["Options"].replace("normdir", "rmdir")
p4.save_client(cl)

ch = p4.fetch_change()
ch._description = "My latest changes."
p4.run_submit(ch)
```

Parse Form examples (P4Python)

PERFORCE
Version everything.

```
client = p4.parse_client(clientAsString)
client._options = myDefaultOptions # etc ...
clientAsString = p4.format_client(client)
```

- This is useful for form triggers

Scripting examples

PERFORCE
Version everything.

- General guidelines

- Submitting changes
- Maintenance (clients, labels)
- Review daemon
- Form manipulation
- Triggers

Write efficient scripts

PERFORCE
Version everything.

- Minimize number of p4 commands
- Limit number of lines of data returned
- Narrow scope of commands
  - Refer to known workspace
  - Access small groups of depot files
- Use care when referencing labels

Minimize number of commands

PERFORCE
Version everything.

- Access files in batches by
  - Change number or label
  - Dedicated workspace views
- Process files locally
- Update files in batches

```
p4.run_edit( ['jam.c', 'jam.h', 'Jambase'] )
```

Limit number of lines returned

PERFORCE
Version everything.

- Reporting and other commands support -m max option
  - clients, labels, branches, streams, changes, users, groups, files, filelog, ...

```
p4.run_changes("-m1")
```

- Form reporting commands (clients, labels, branches) have a filter option
  - -e nameFilter
  - -E case-insensitive

```
p4.run_clients("-E", "svr-dev-rel*")
```

Example: Submitting a change

PERFORCE
Version everything.

- Retrieve change form from the the server
- Set the description
- Optional: add jobs to complete
- Submit the change

- Optionally: process result
  - "SubmittedChange" in the submit result

Code: Submitting a change

PERFORCE
Version everything.

```
if p4.run_opened(): # anything to submit
   change = p4.fetch_change()
   change._description = "Script-submit"
   change._jobs = ["job012345", "job000001"]

   try:
    result = p4.run_submit(change)
   except P4.P4Exception as e:
    # deal with exception here
# process result
```

Example: Delete or unload old workspaces

PERFORCE
Version everything.

- Each form has an Access field
  - Last time this form was used (sync, edit, ...)
- Timestamp in seconds since epoch

- Idea:
  - List all client workspaces
  - Find workspaces with access older than a year
  - Delete or unload those workspaces

Code: Delete or unload old workspaces

PERFORCE
Version everything.

```
from P4 import P4
from time import time # for current time
p4 = P4()
p4.connect()
for c in p4.run_clients():
  age = time() - int(c["Access"])
  if age > 86400 * 7 * 52: # 52 weeks
   p4.delete_client("-f", c["client"])
   # p4.run_unload("-f", "-c", c["client"])
p4.disconnect()
```

Triggers

PERFORCE
Version everything.

- Reminder: trigger types
  - Authentication triggers
  - Change triggers
  - Shelve triggers
  - Form triggers
  - Fix triggers
  - Archive triggers
- Triggers are executed on the server machine
  - Same user and permissions that started the Perforce Server
  - Need to consider connection parameters and user authentication

Trigger examples

PERFORCE
Version everything.

- Authentication trigger
- Submit triggers
  - Analyze description field
  - Check case trigger
- Form triggers
  - Default client workspace settings

- Archive trigger

Authentication trigger

PERFORCE
Version everything.

- auth_check trigger
- Verify password outside of Perforce (for example, LDAP)
- Example triggers are provided on website

- Provide %username% to user
- Password passed to trigger via stdin

- Trigger exits with 0 = success or 1 = failure

Trivial authentication trigger example

PERFORCE
Version everything.

```
#!/bin/bash
USERNAME=$1
PASSWORD=secret
read USERPASS  # read from stdin
if [ "$USERPASS" = $PASSWORD ]
then
   exit 0
fi
echo checkpass.sh: password $USERPASS for $USERNAME is incorrect
exit 1
```

Change triggers

PERFORCE
Version everything.

- Most change triggers follow the same pattern:
  - Receive changelist number
  - Run p4 describe
  - Analyze description, file list and jobs
  - Content-trigger: analyze file content
  - Accept or reject change

Change trigger template

PERFORCE
Version everything.

- Example templates for Python and Ruby on public depot: P4Triggers.(py|rb)
- Change stored in a P4Change instance
- Subclass for your own trigger
  - Override setUp() and validate() methods

- Use cases:
  - Check Case Trigger
    - Prevent case conflicts on case sensitive Perforce Servers
  - Check description or jobs
  - Protect code lines without modifying the protection table

Default workspace spec – form-out trigger

PERFORCE
Version everything.

- Provide default settings for workspaces without using a template workspace
- Idea: use a form-out trigger for new workspaces

- Problems:
  - How do you identify it is new workspace?
  - The form-out trigger provides a filename

Identify new workspace?

PERFORCE
Version everything.

```
clientName = sys.argv[1]
filename = sys.argv[2]

p4.client = clientName
p4.connect()
clientInfo = p4.run_info()[0]['clientName']
if clientInfo != '*unknown*':
   sys.exit(0) # trigger succeeds w/o modification
p4.disconnect()
```

Convert file into spec and back

PERFORCE
Version everything.

```
clientAsString = READFILE(filename)

client = p4.parse_client(clientAsString)
client._options = myDefaultOptions # etc ...
clientAsString = p4.format_client(client)

WRITEFILE(filename, clientAsString)

sys.exit(0)
```

Archive triggers

PERFORCE
Version everything.

- Required for new file type +X (2009.1 +)
- Trigger of type “archive” with 3 arguments
  - %op% - read/write/delete
  - %file% - name of archived file
  - %rev% - revision of archived file
- Provides complete control over the way a file is stored or accessed, for example
  - Database storage
  - Offline storage on separate device
- `p4 verify -X` avoids verifying revisions accessed via an archive trigger

Conclusion

PERFORCE
Version everything.

- Choose the right scripting environment
  - Derived APIs more powerful
- Set up the connection and user parameters
  - Either environment or directly provided
- Think about authentication
  - Daemons and triggers

- Script efficiently

The End

PERFORCE
Version everything.

All Perforce manuals and technical notes are available at www.perforce.com.

Follow and participate with the Perforce Community and Forums at www.perforce.com/community

Report problems and get technical help from support@perforce.com.