ownCloud

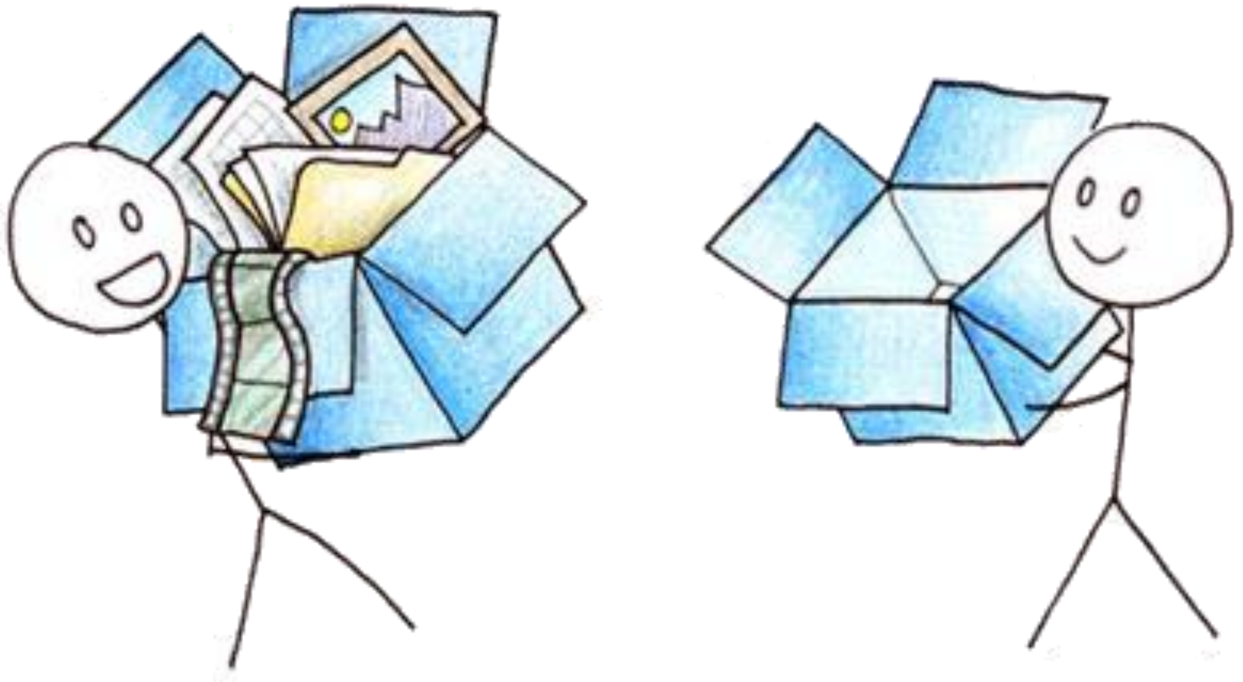
By Sirus Shahini

Iran University of Science and Technology
Department of Computer Engineering

Introduction

- Data and data storage
- Impotence of traditional storage mechanisms due to excessive need for larger and more flexible media
- Larger storage capacity cannot be attained locally
- Introduction of new approaches to data storage
- Web-based cloud storage as an incarnation of modern treatment of the issue
- Introducing OwnCloud as an open source cloud service

So why not keep it the Dropbox way?!



OwnCloud’s raison d'être

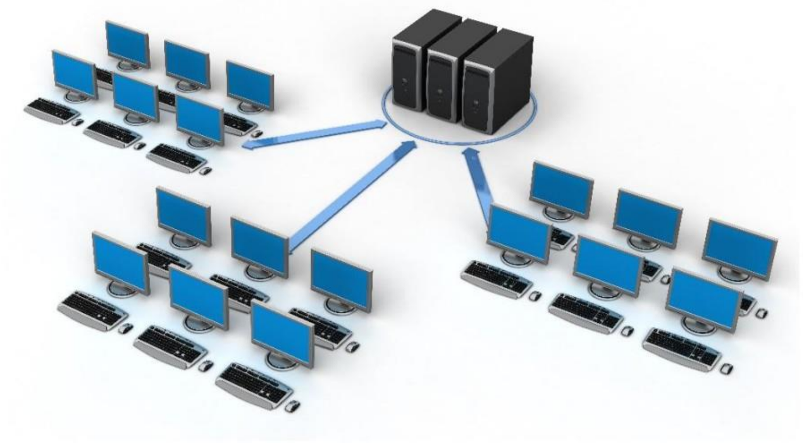
- Trust issues with outsourcing storage demands
  - Sensitive data is not in the sole possession of the company
  - Direct company’s moderation of data is not possible
  - Forensics and investigation is not possible
  - Speed, capacity and other features are limited to the outsider host’s policies
- Not in control:
  - Sensitive and confidential data
  - Storage and servers
  - User provisioning
  - Security
  - Governance

OwnCloud customers

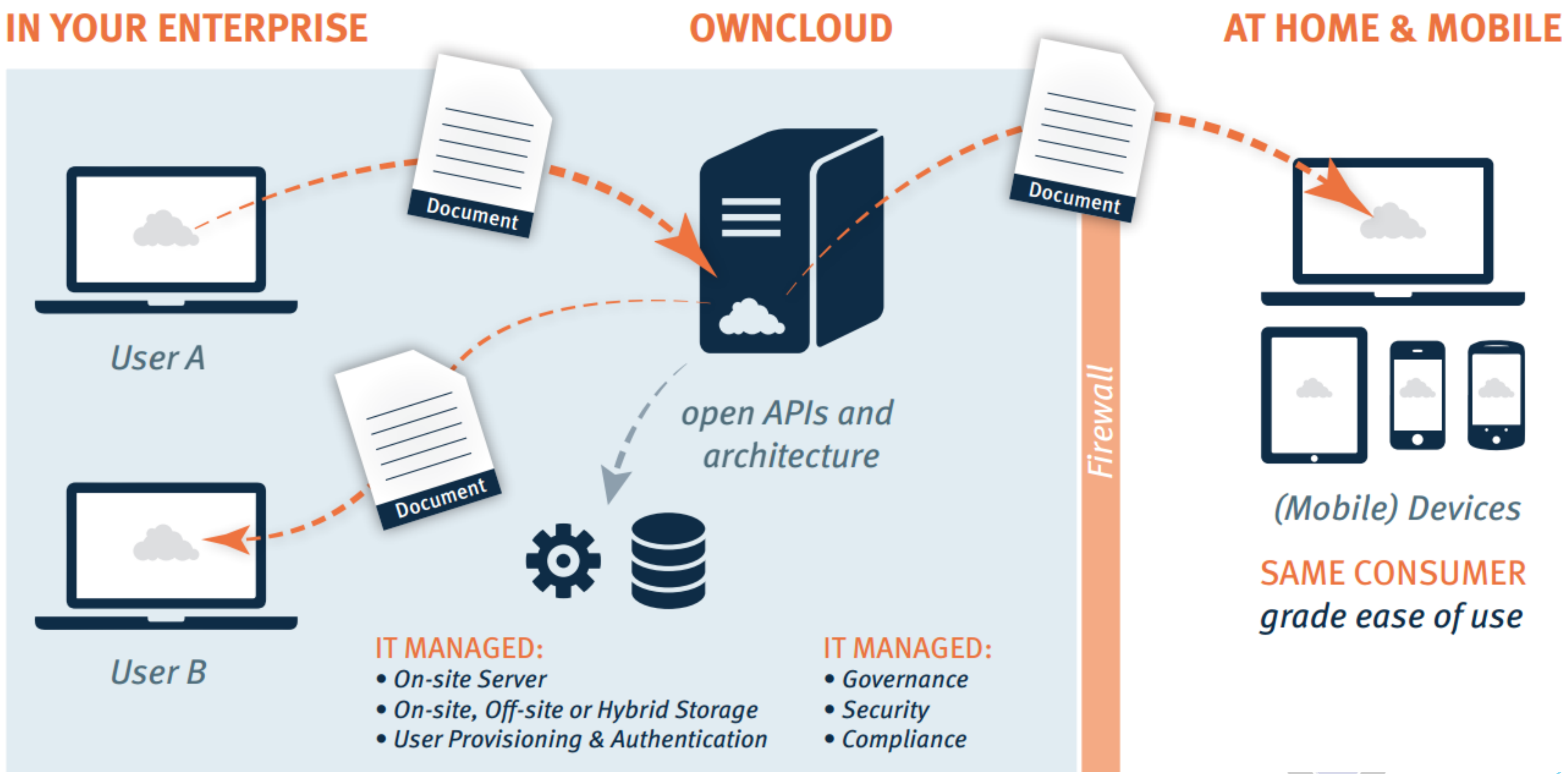
- University of Felorida
- University of California Riverside
- Southern Oregon Unversity
- Kansas State University
- University of California San Francisco
- iThreat Cyber group
- Ecomnets
- Paranet
- Etc.

Architecture overview: Features

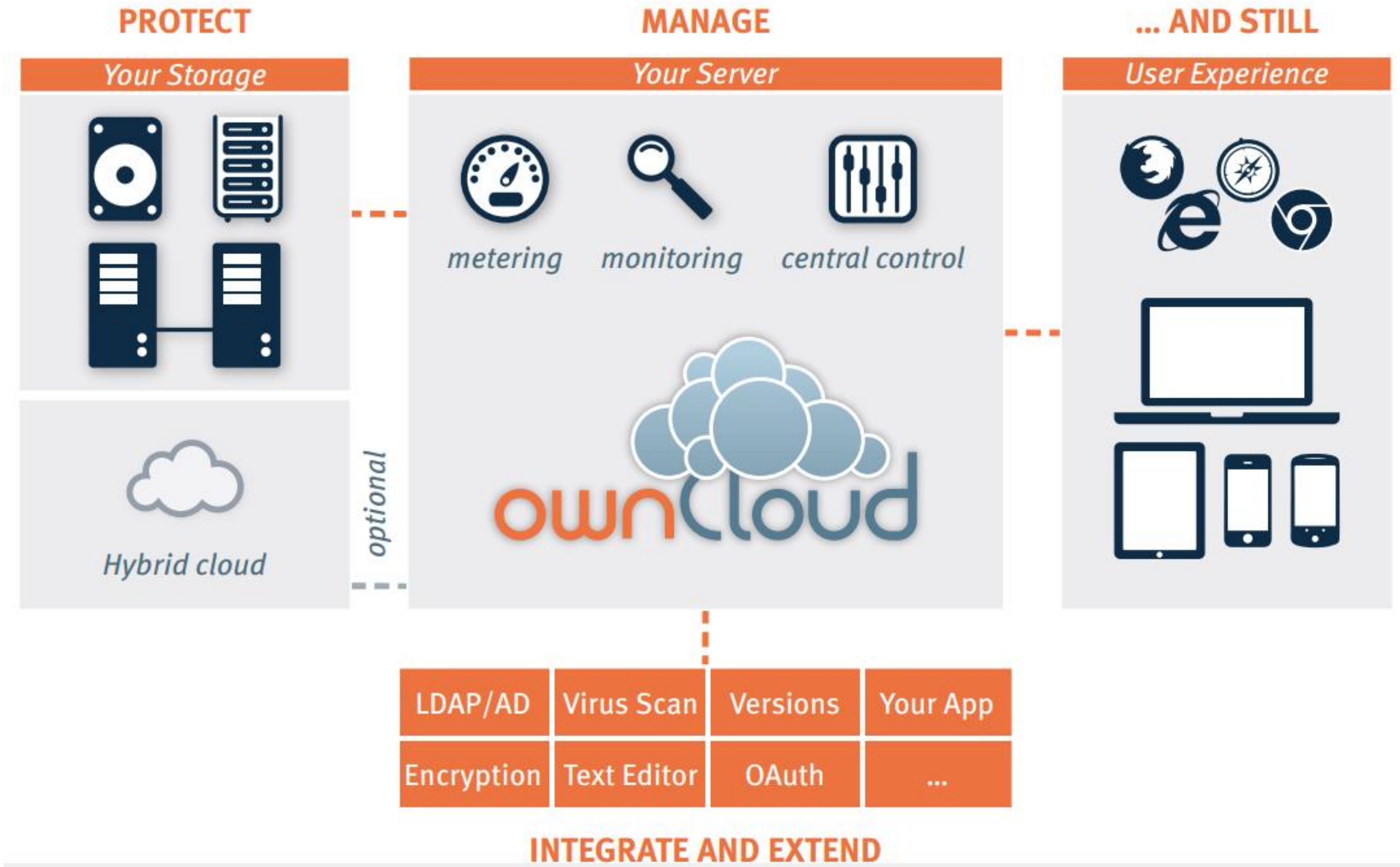
- Manageable and Protectable data
- Integrate-able with existing IT systems and policies
- Extensible functionality
- Providng users with clean, intuitive access from many types of devices



Architecture overview



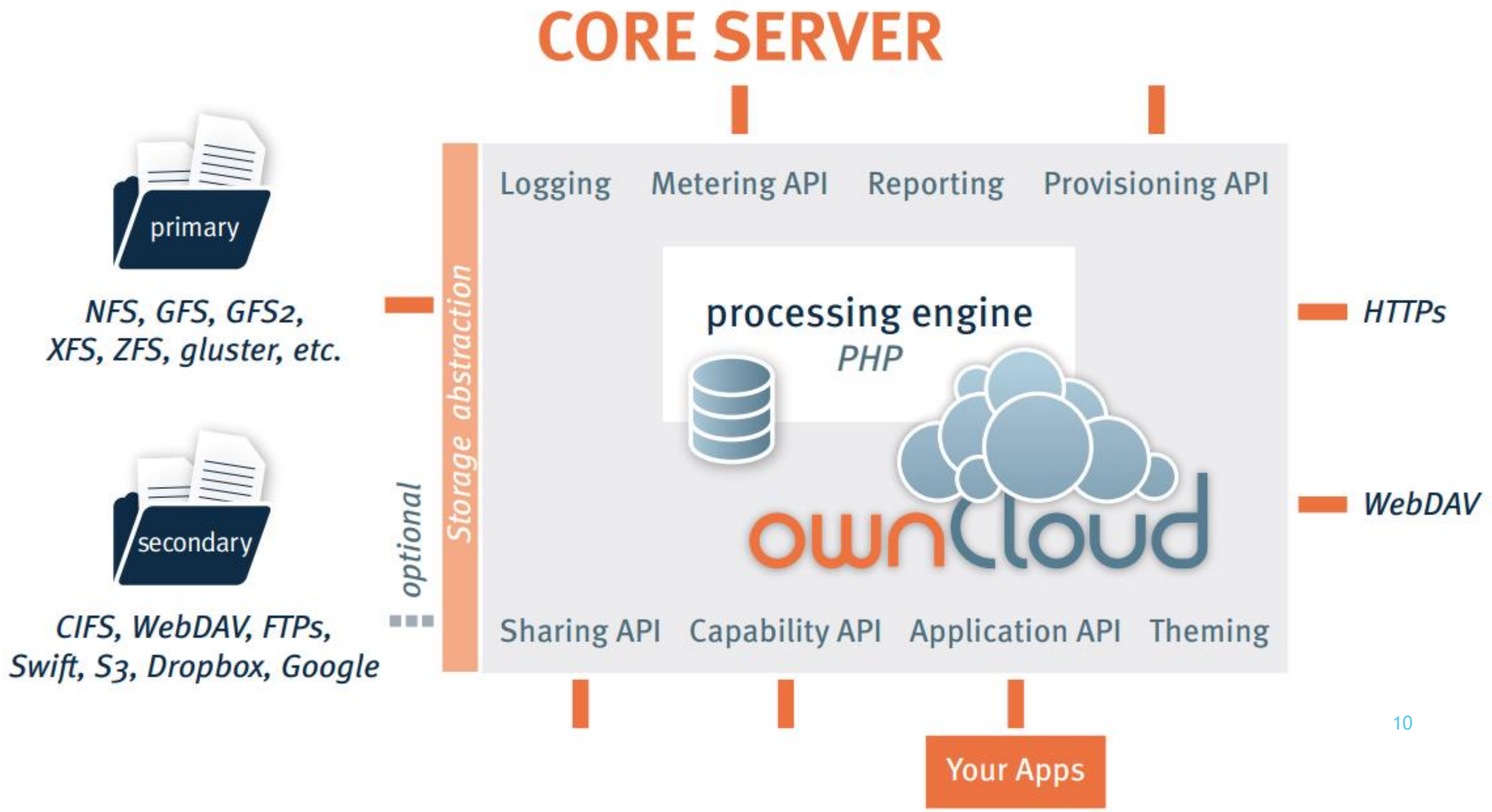
Architecture overview



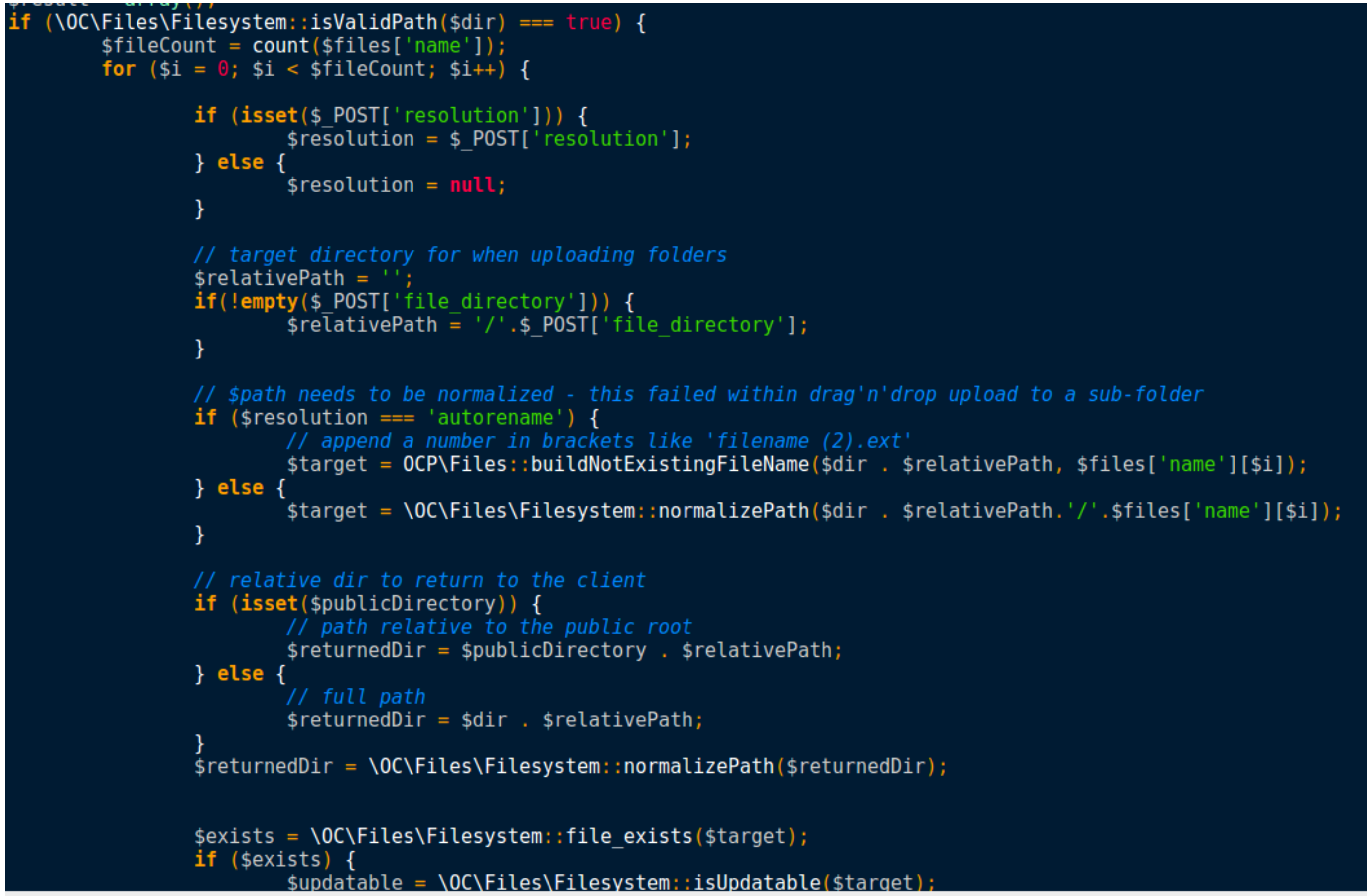
Architecture overview: Server

- A PHP web server running on Apache or IIS
- Attached to the web server is a database to store:
  - Users
  - User-shared file details
  - Plug-in application state
  - OwnCloud file cache
- Access to database is provided by an abstraction layer
  - This enables support for many kinds of databases
- Complete web server logging
- Abstracted storage tier to enable a broad range of storage
- alternatives

Architecture overview: Server (cont.)



Architecture overview: Server (cont.)



```
if (\OC\Files\Filesystem::isValidPath($dir) === true) {
        $fileCount = count($files['name']);
        for ($i = 0; $i < $fileCount; $i++) {

                if (isset($_POST['resolution'])) {
                        $resolution = $_POST['resolution'];
                } else {
                        $resolution = null;
                }

                // target directory for when uploading folders
                $relativePath = '';
                if(!empty($_POST['file_directory'])) {
                        $relativePath = '/'.$_POST['file_directory'];
                }

                // $path needs to be normalized - this failed within drag'n'drop upload to a sub-folder
                if ($resolution === 'autorename') {
                        // append a number in brackets like 'filename (2).ext'
                        $target = OCP\Files::buildNotExistingFileName($dir . $relativePath, $files['name'][$i]);
                } else {
                        $target = \OC\Files\Filesystem::normalizePath($dir . $relativePath.'/'.$files['name'][$i]);
                }

                // relative dir to return to the client
                if (isset($publicDirectory)) {
                        // path relative to the public root
                        $returnedDir = $publicDirectory . $relativePath;
                } else {
                        // full path
                        $returnedDir = $dir . $relativePath;
                }
                $returnedDir = \OC\Files\Filesystem::normalizePath($returnedDir);


                $exists = \OC\Files\Filesystem::file_exists($target);
                if ($exists) {
                        $updatable = \OC\Files\Filesystem::isUpdatable($target);
```

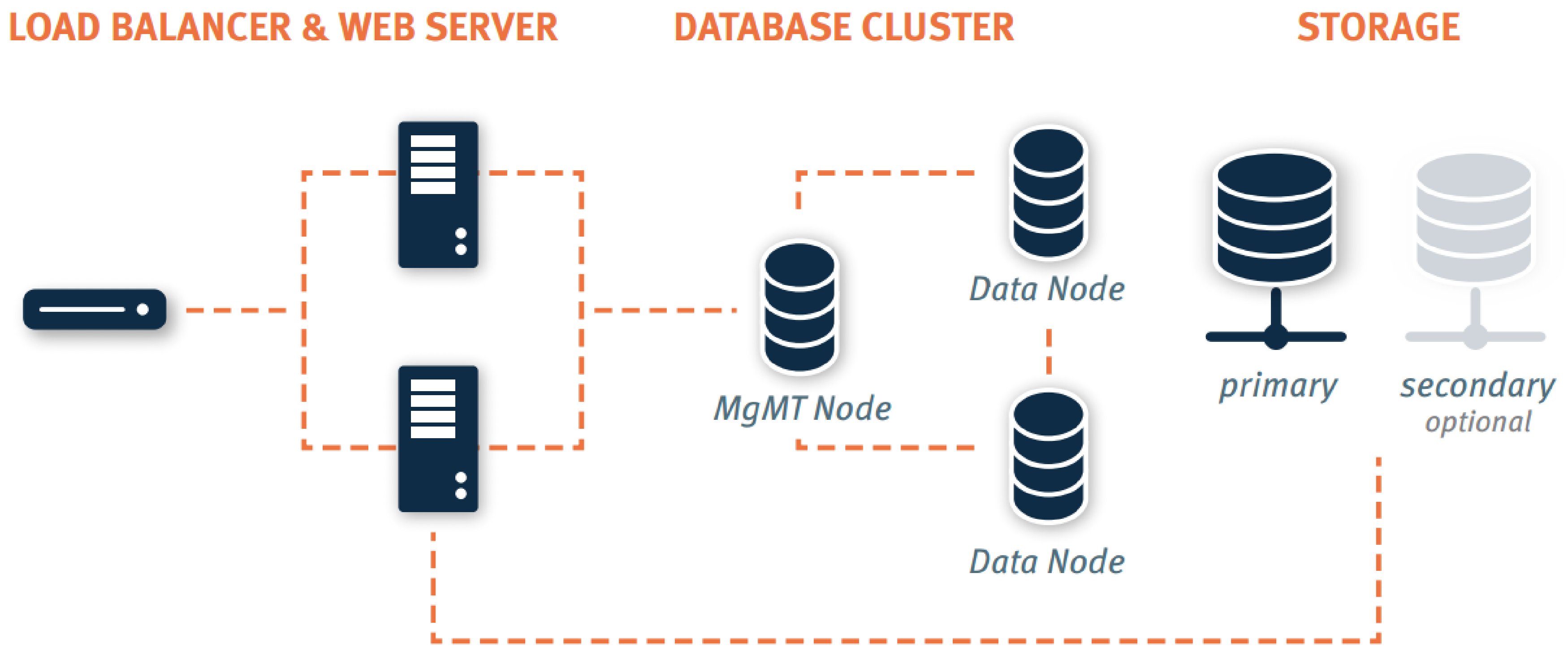
Architecture overview: Server (cont.)

- Abstracted storage tier:
  - Ability to leverage just about any storage protocol
    - CIFS
    - NFS
    - GFS
    - Clustered systems like Red Hat Storage
  - Ability to use external file system applications
    - Live, Windows home directories, FTP, WebDAV,
    - External cloud storage services: S3, Swift, Google Drive, Dropbox
- Enabling data segregation and multi-tenant deployments
  - User configuration can include dynamically allocated storage driven by user directory enteties

Architecture overview: API

- Activity: Provides an RSS feed or API call to deliver all activites
- Applications: The most powerful API to expand OwnCloud
- Capability: Offers information about features and plugins of the Owncloud on the server
- External provisioning: Provides the ability to remotely manage users and query metering information of the server
- Sharing: Provides the ability for external apps to share files from remote devices
- Theming: A simplified mechanism to accommodate stylistic needs to company’s needs

Deployment


LOAD BALANCER & WEB SERVER
DATABASE CLUSTER
STORAGE
Data Node
MgMT Node
Data Node
primary
secondary
optional

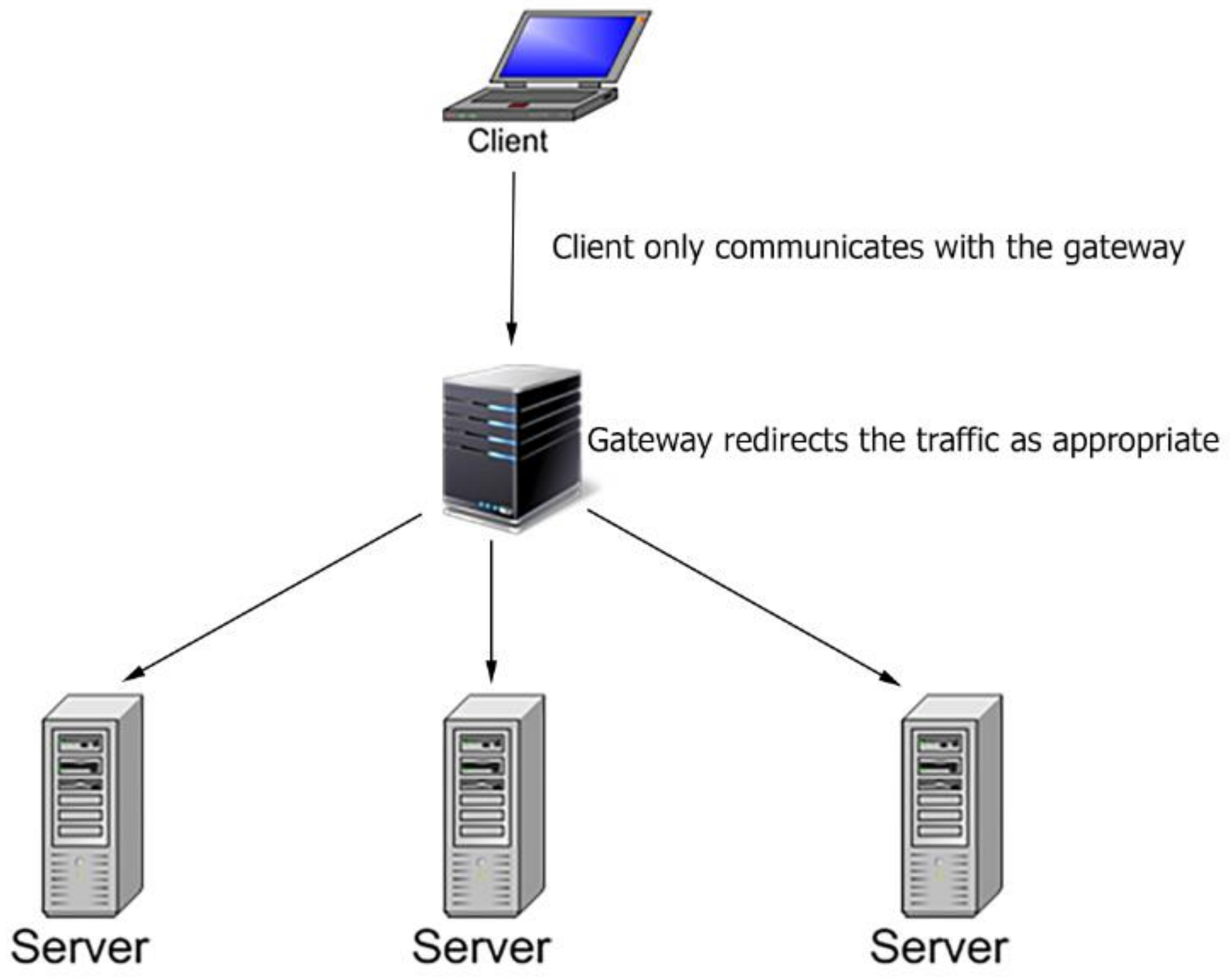
Deployment

- Most often deployed as an n-tier load balancer web application
- A load balancer in the front-end of the deployment and connected to at least to web servers
- PHP core as the webserver is hosted on Apache/IIS powered servers
- Webservers are connected to a database which is preferably a clustered MySQL database instance
- Webservers are connected to one or more backend storage
- Support for REST-API based storage through application plugins

A real world deployment project

- As the developers teams chief you are asked to implement the webserver infrastructure of the company in a way that stringently satisfies these requirement:
  - The company IT management refuses to schedule periodic or sporadic storage media synchronization due to critical timing policies. For example if at any time one of the storage media for any reason gets damaged there must be a backup of all data of the storage in question on another backup server.
  - There must an automatic redirection mechanism to transparently route clients traffic to an auxiliary server during down time of the master one.

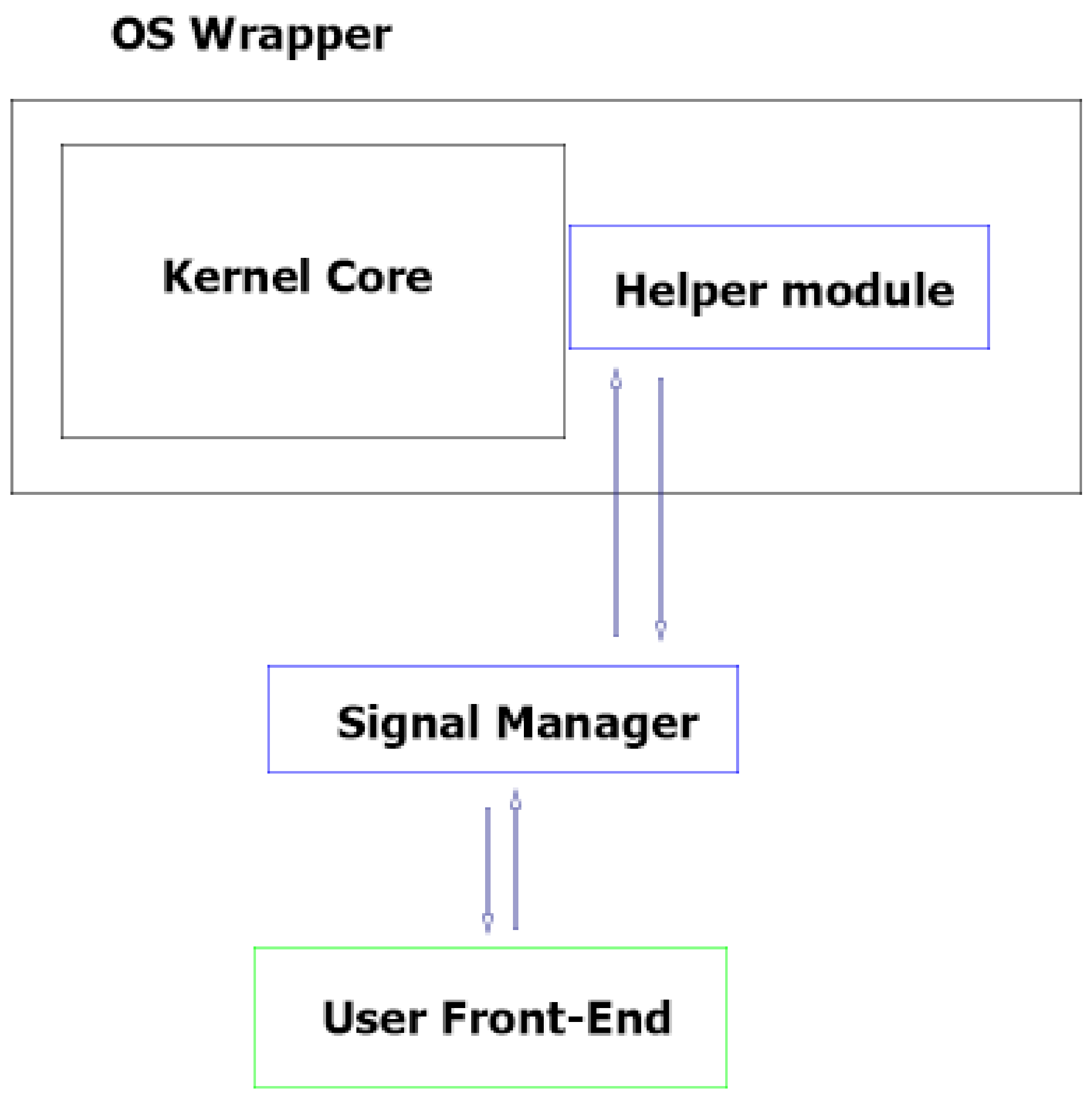
Service topology


Client
Client only communicates with the gateway
Gateway redirects the traffic as appropriate
Server
Server
Server

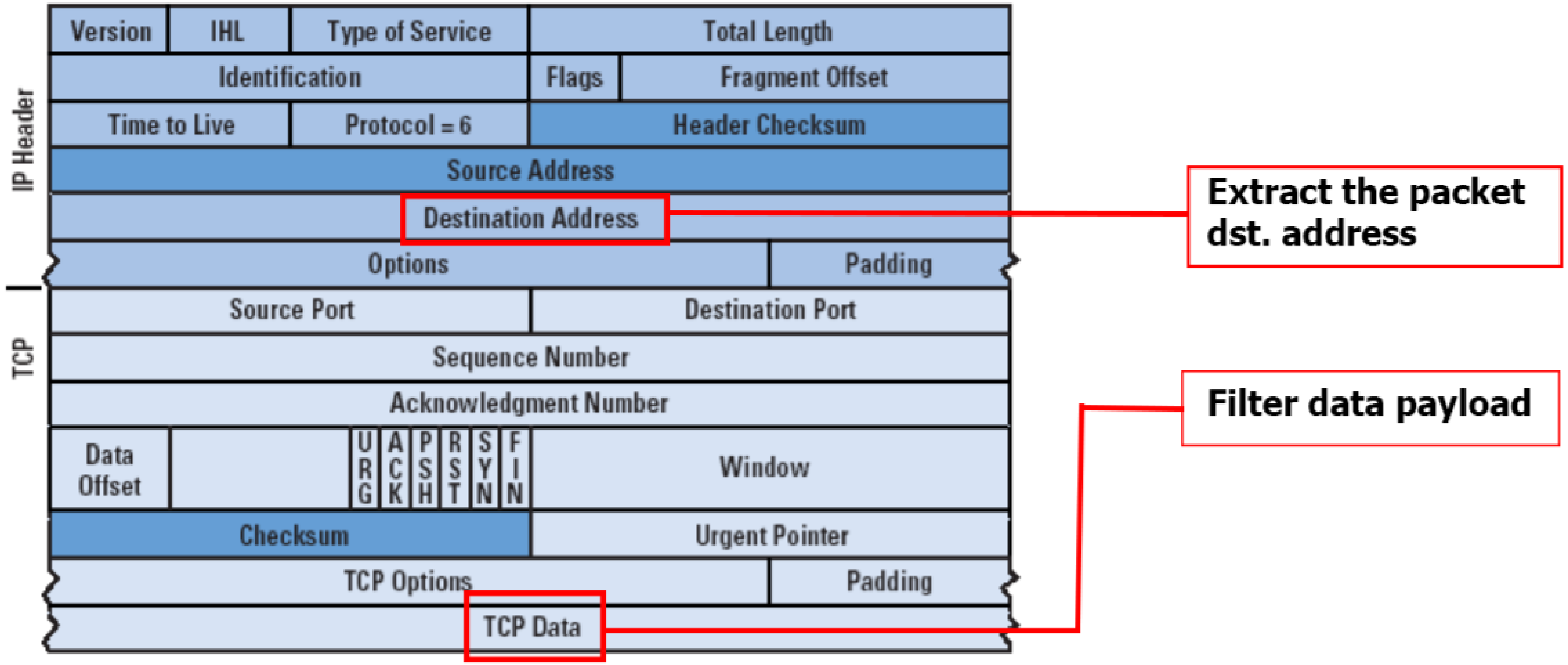
Solutions

- Using NFS-like sync and database constraints
  - Very inflexible
  - Hard to customize
  - Hard to accommodate to system changes
  - Completely dependent on web application internal functioning
- Building a custom middleware running on the gateway to monitor and manage the traffic
  - Independent from version changes
  - Completely under control

A simple architecture

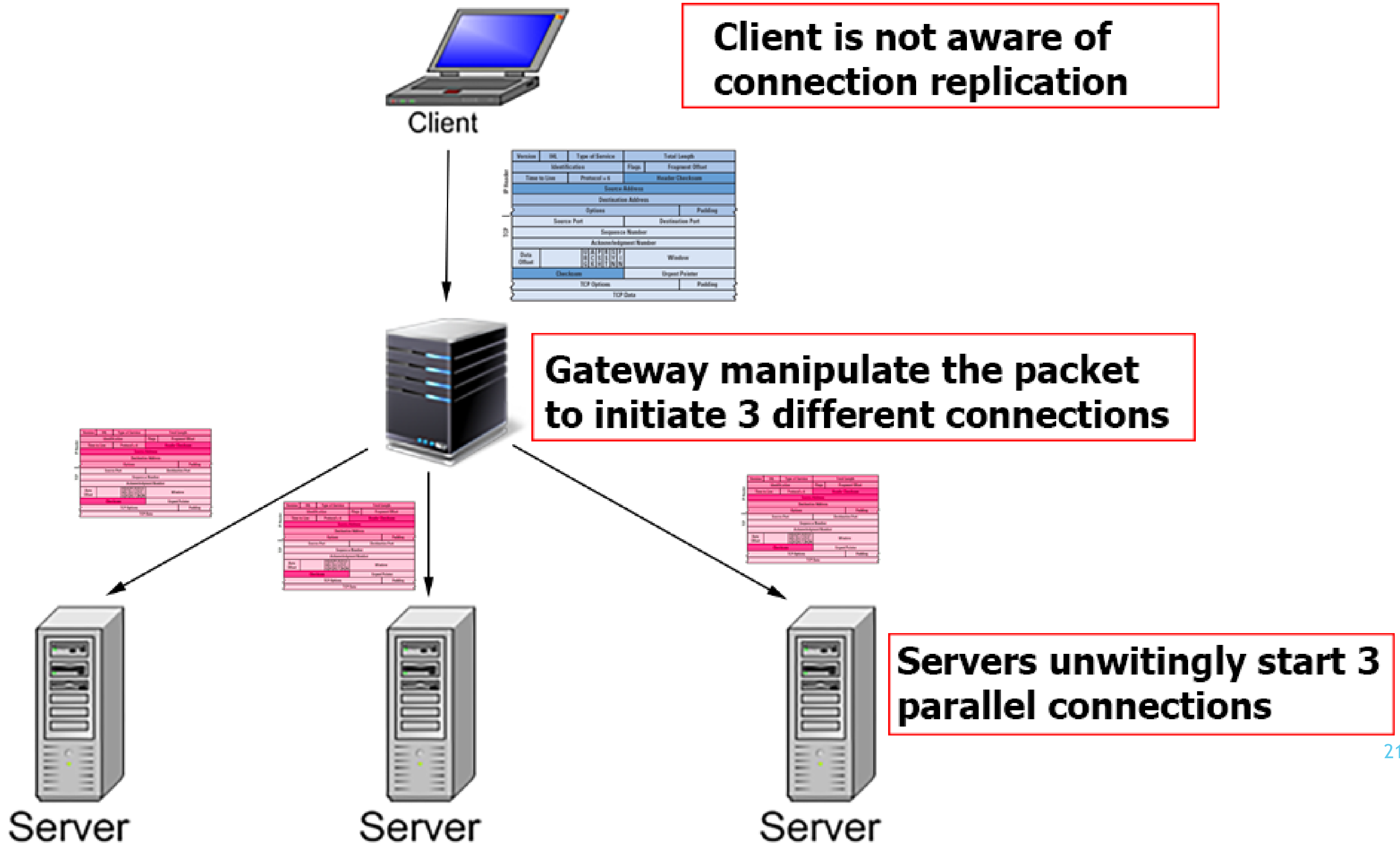


Dissecting the problem



Check data and if the packet is to be stored on backup servers pipe the packet to each separate line

On-the-fly session replication

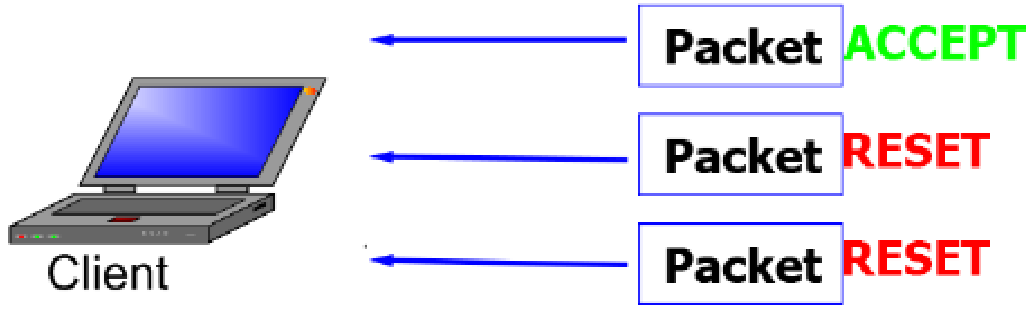


Client is not aware of connection replication

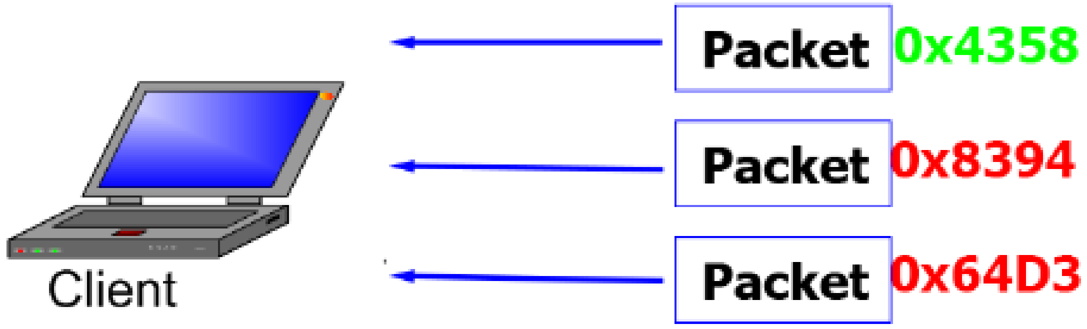
Gateway manipulate the packet to initiate 3 different connections

Servers unwitingly start 3 parallel connections

What happens the way back to the client?

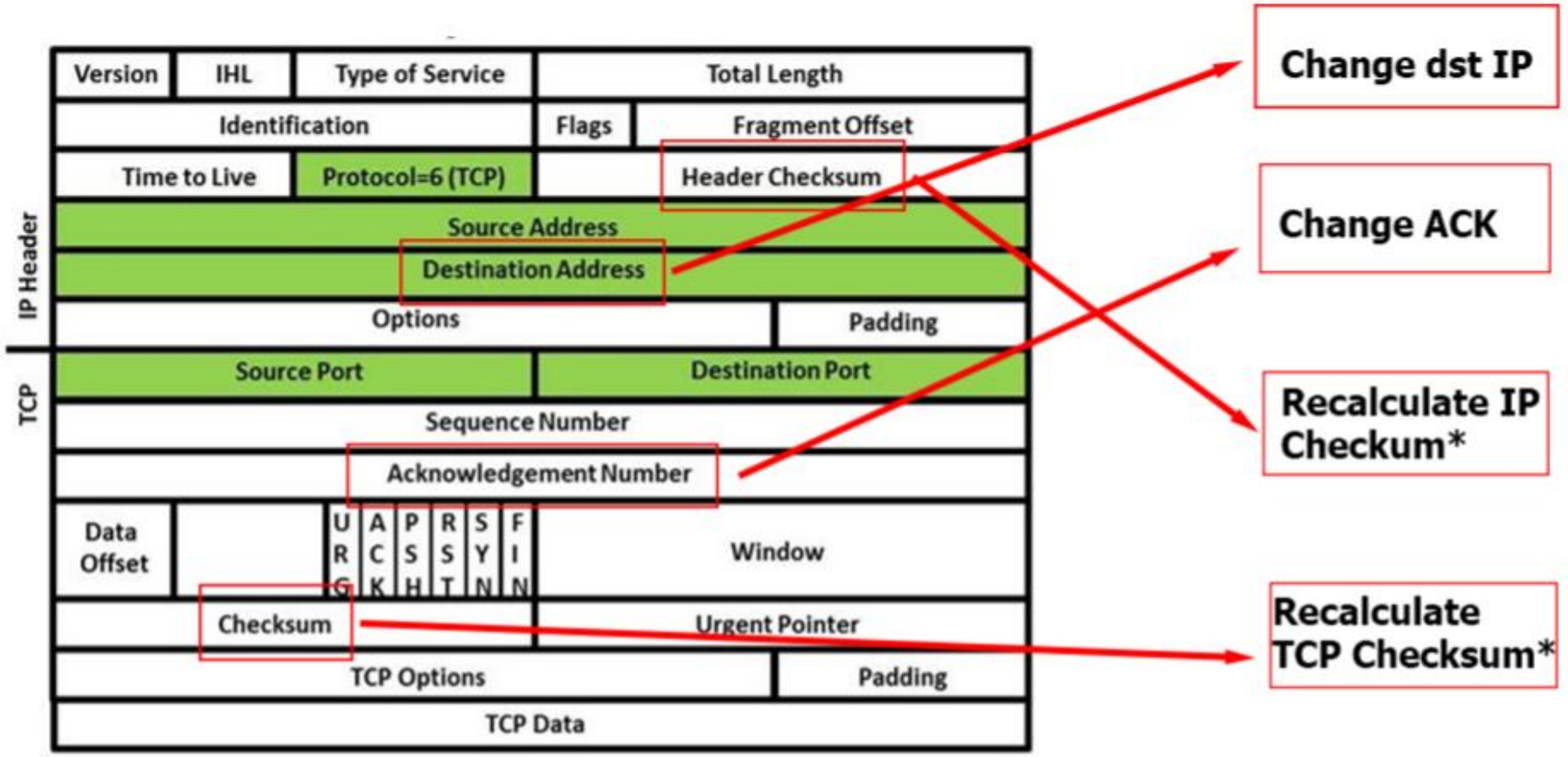


Client expects only one response not 3



Only one sequence number is accepted for each connection

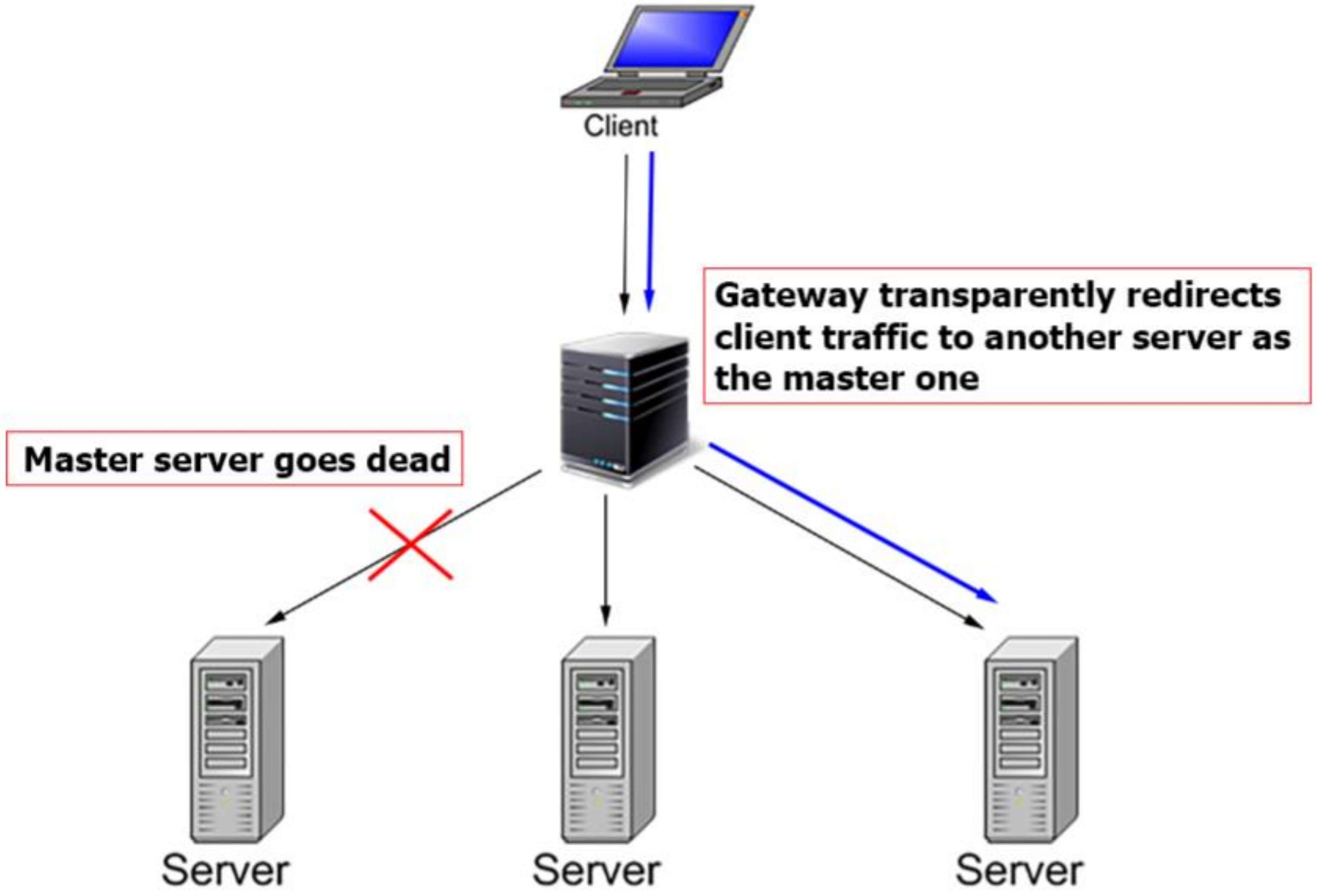
Revising the packets



Abstracting the 3-way handshake

- Client send “TCP SYN” to gateway.
- Gateway routes the packet to all 3 servers while considering one as the master
- All servers reply with “TCP SYN-ACK”
- Gateway routes the master reply to the client and saves sequence number of the other 2
- Client sends “TCP ACK” to finish the handshake.
- Gateway builds two additional packets and revise the addresses and acknowledgment numbers
- Gateway routes all 3 packets to their corresponding servers
- Handshake is complete and all 3 connections enter “ESTABLISHED” state
- From this point on all data packets are redirected to all servers

What about automatic redirection?


Client
Gateway transparently redirects client traffic to another server as the master one
Master server goes dead
Server
Server
Server

☺

And that’s how it is done ;)