Implementing Internet Storage Service Using OpenAFS

Sungjin Chun(chunsj@embian.com)
Dongguen Choi(eastroot@embian.com)
Arum Yoon(toy7777@embian.com)

Overview

Introduction

Implementation

Current Status

Issues

Summary

Introduction

Internet Storage Service(ISS)

ISS is a service that provides **computer storage service** and related management to users

Works As ...

Service for Simple Means of **Sharing** Files

Service for Off-Site **Back Up**

Using ...

Web Browser

Proprietary Client

SMB/CIFS, WebDAV

All or Any Combinations of Above

ISS is Basically Storage Service and

Storage is **Most Important Component**

Storage Infra. Requirements

Scalability

Easy of Use/Development

Stability

Cost

Scalability

Needs Massive Storage: **Peta byte Scalability**

ex) **1 GB** Per User: **1 PB** for **1 Million** Users

Easy of Use/Development

Easy of Use: Administration/Maintenance

Transparent Operation: Work as Normal File System

Stability

No Single Point of Failure

Disposable Computing

Cost

Adaptive Scalability

Deployable on Cheap Hardware

Free/Open Source Software is Better :-)

Our Decision

OpenAFS as Our Storage Infrastructure!

Why OpenAFS?

Open Source

Scalable Architecture

Easy to Administer

Global, Uniform Namespace

Implementation

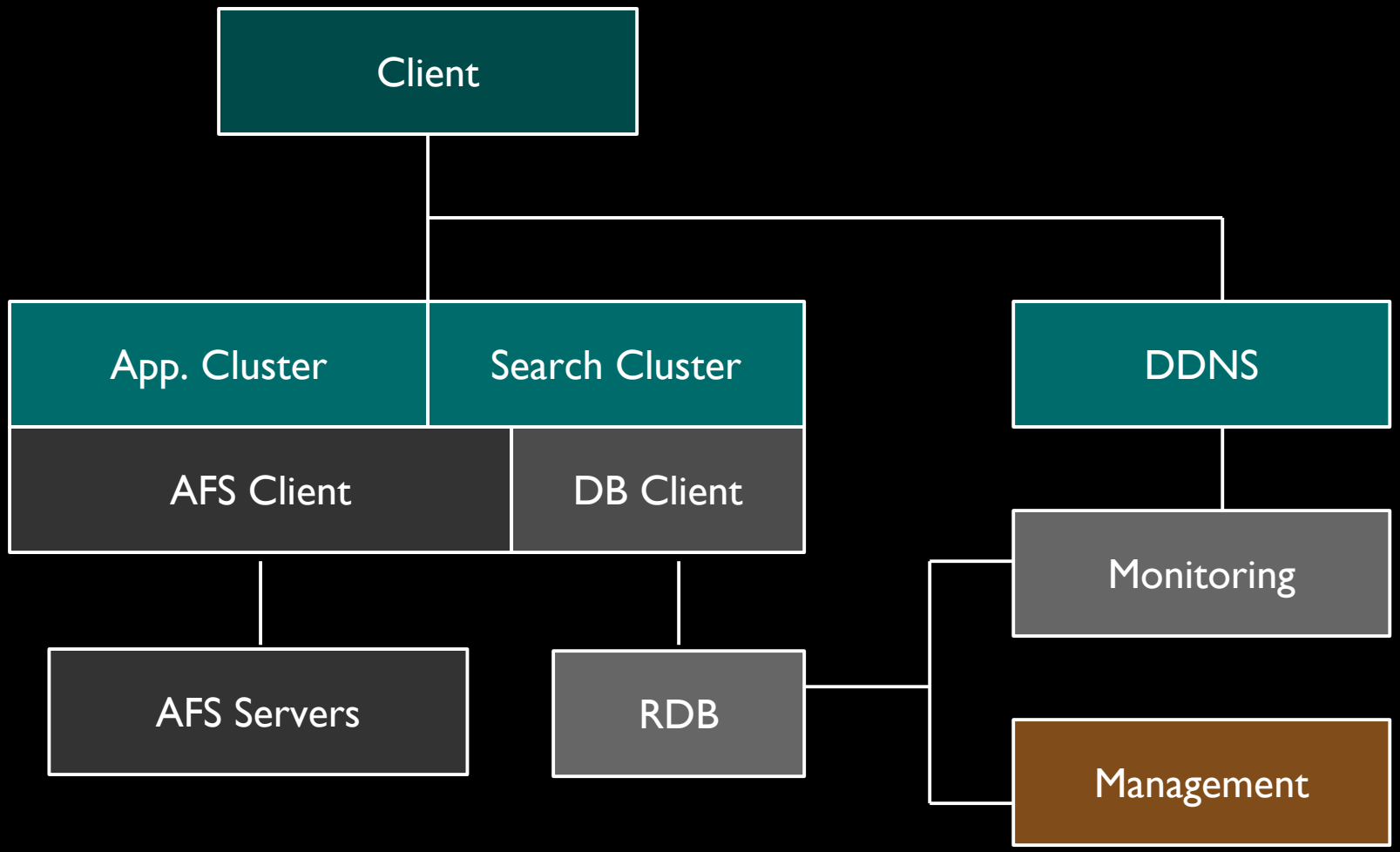
Embian eFolder

Embian's Implementation of ISS

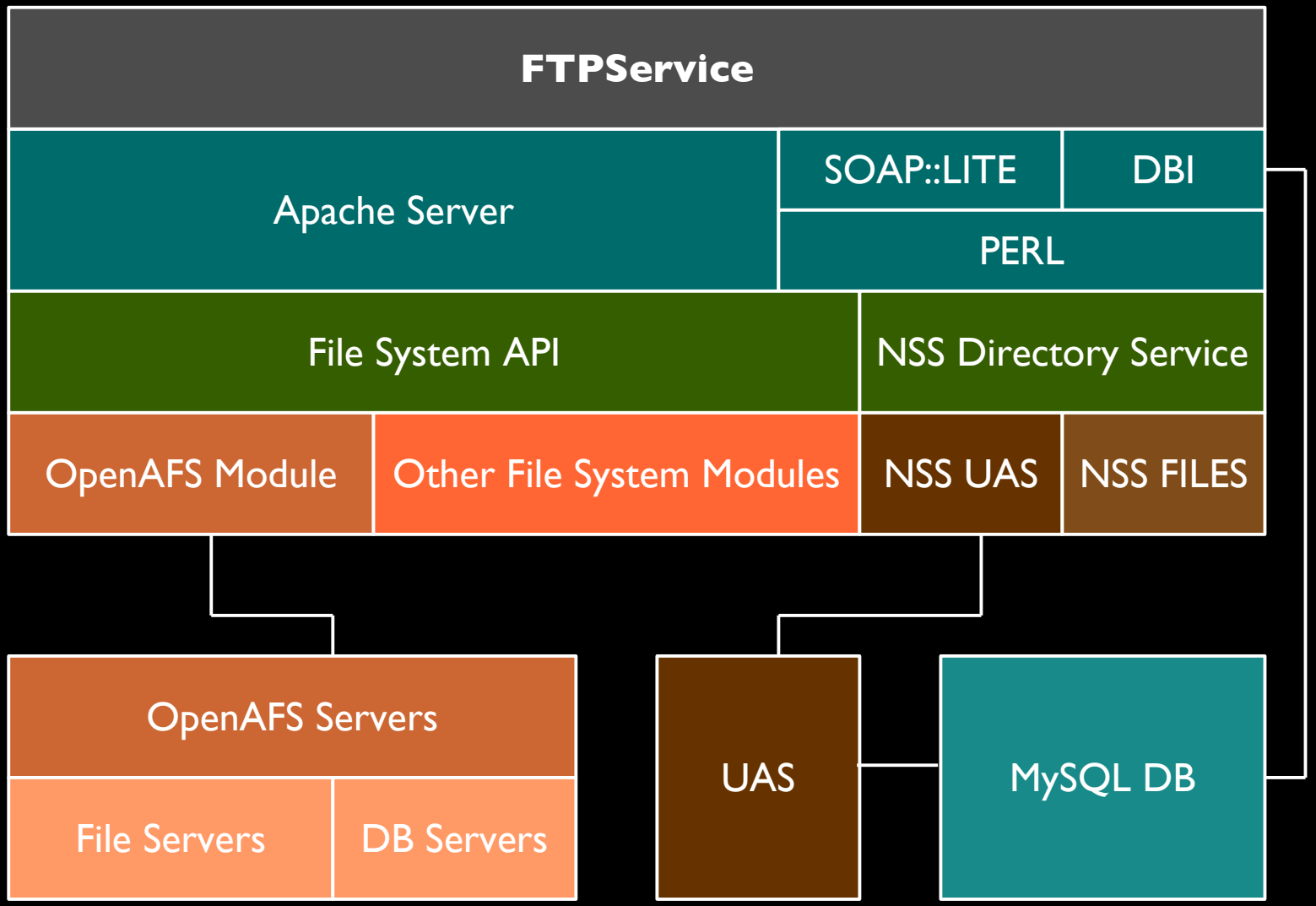
Storage/File System Independent Implementation

Uses SOAP/HTTP

Embian eFolder Architecture



Embian eFolder Components



Deployment Differences

OpenAFS as Workspace

vs.

OpenAFS as Storage Space

OpenAFS as Workspace

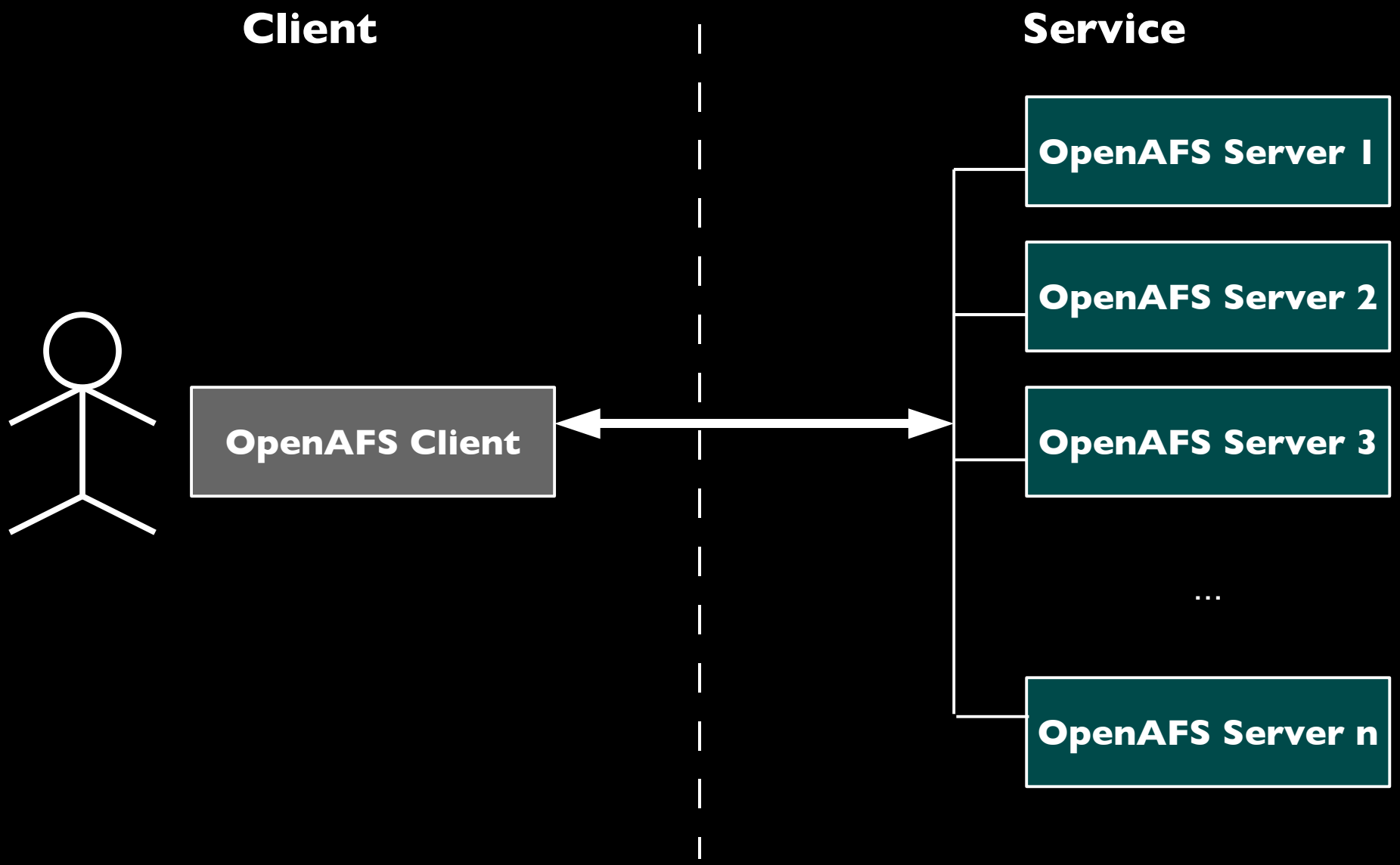
"Normal" OpenAFS Deployment Style

2 Tier Style

AFS is Exposed to User

AFS as a (Direct) Workspace

OpenAFS as Workspace



OpenAFS as Storage Space

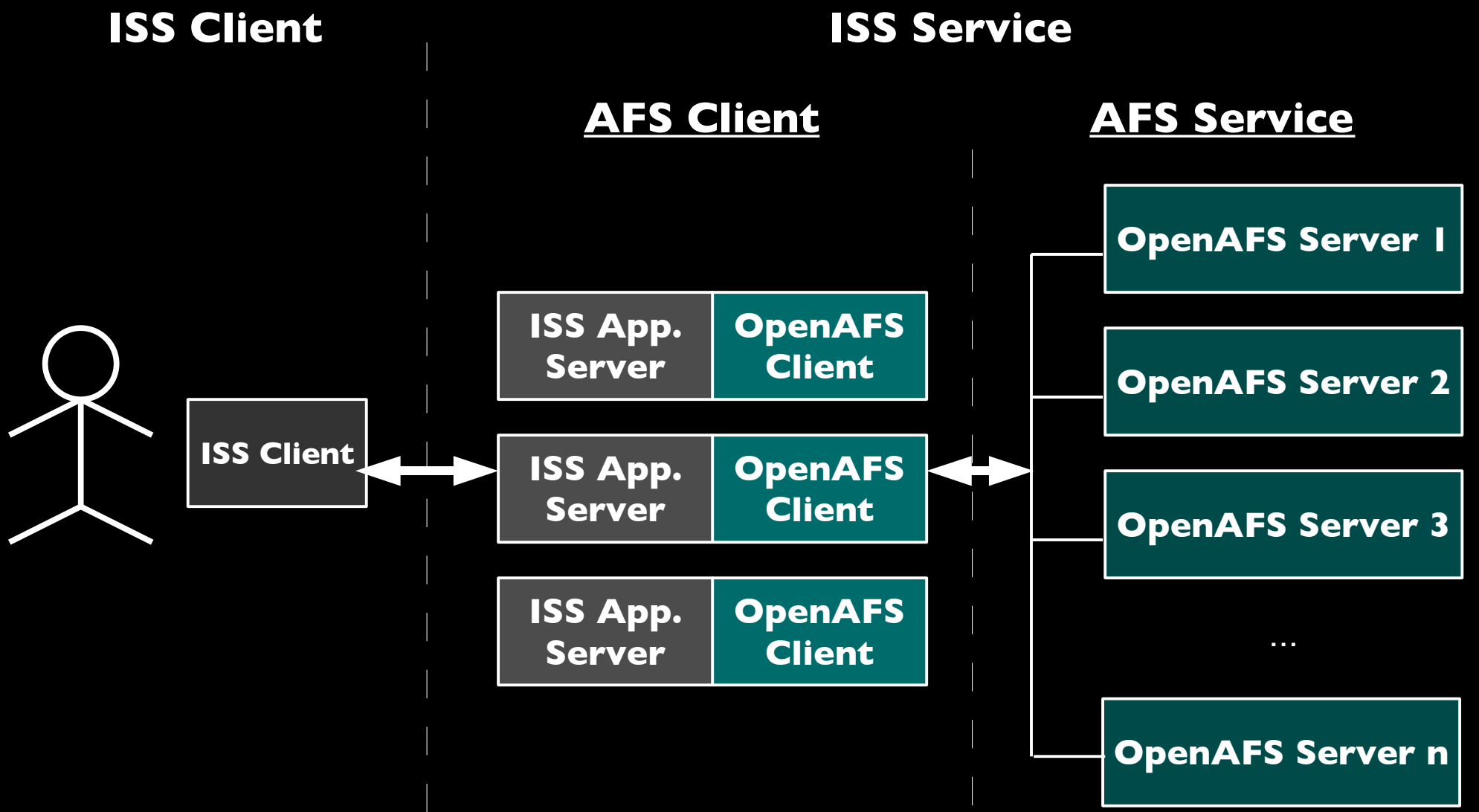
Deployment Style for ISS

3 Tier Style

Works as Virtual Large File System

Application Server Hides OpenAFS

OpenAFS as Storage Space

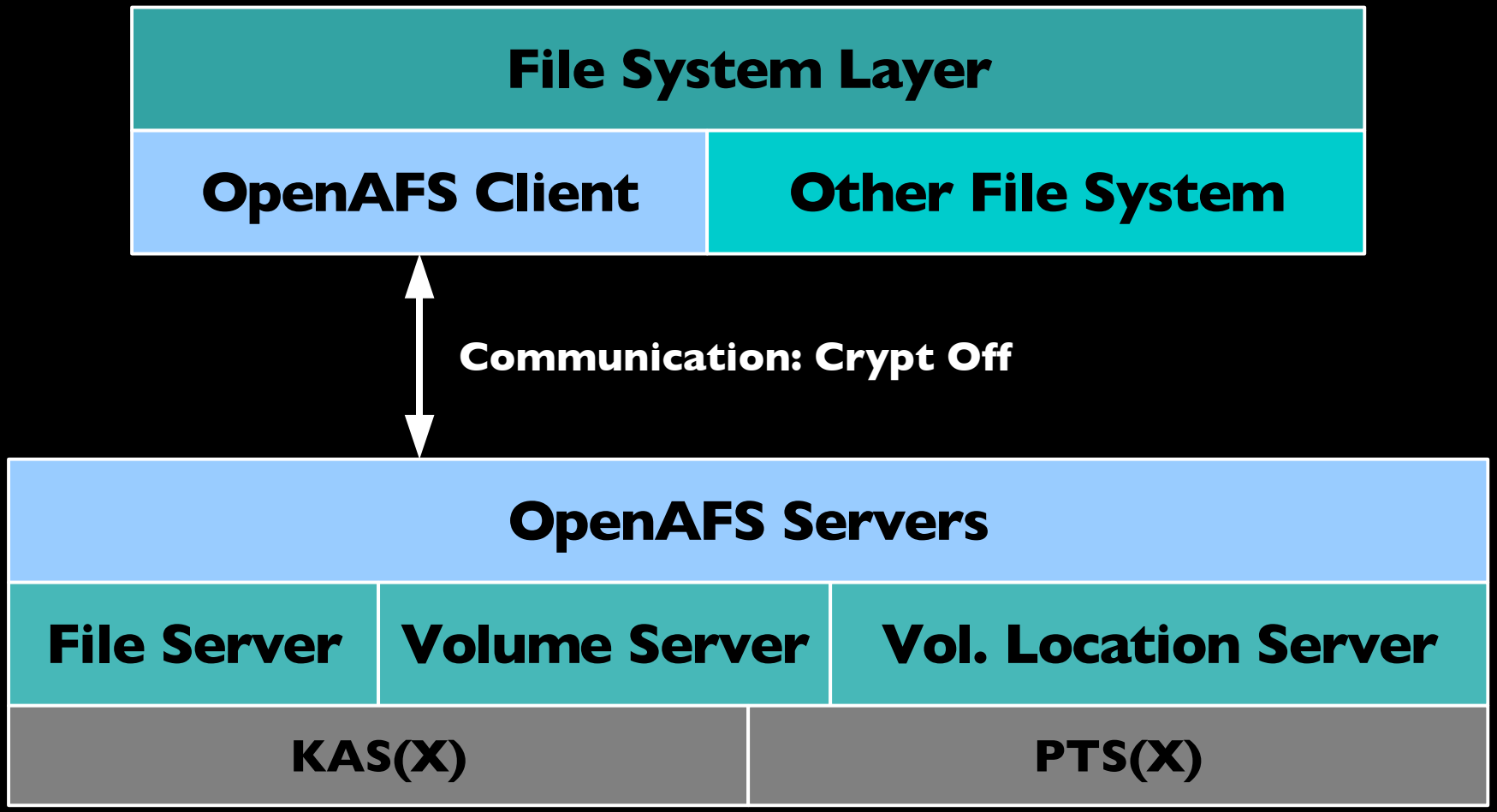


OpenAFS as Storage Space

We **Use - File System/Storage Related Parts**

We Discard - Authentication/Authorization/Security Parts of OpenAFS Components

OpenAFS as Storage Space



We Use **OpenAFS** as **File System Backend, Large Storage Space**

Not as “Normal” Workspace Style Deployment for Distributed Environment

Our Unique Deployment Strategies

OpenAFS Client Grouping by Their Roles

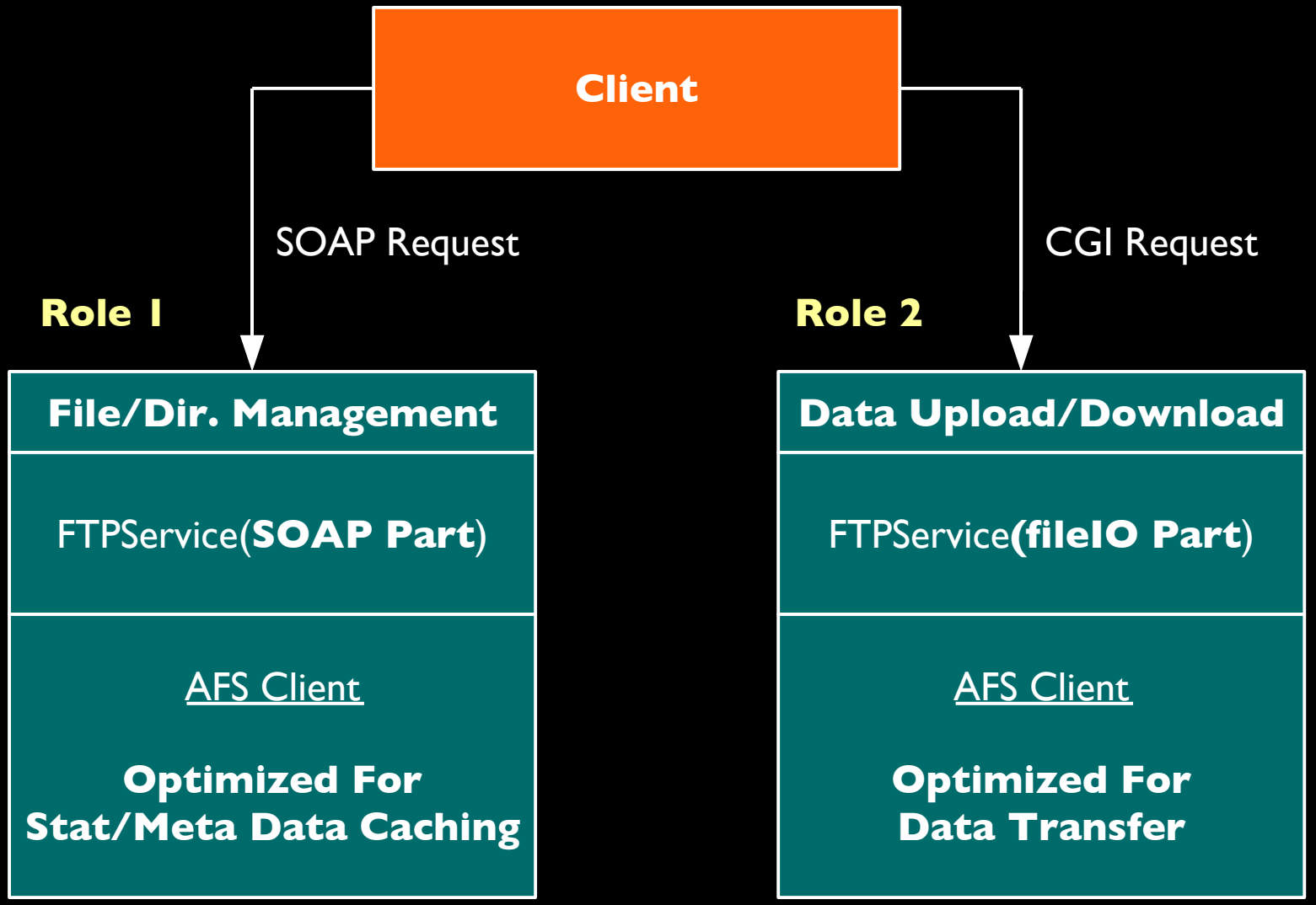
OpenAFS Network Grouping for Efficiency

OpenAFS Client in eFolder

Role 1. File Directory Service

Role 2. File Transfer Service

OpenAFS Client Deployment in eFolder



Why Separated Roles?

Reducing Latency in File Listing/Managing Operation

Caching Efficiency: File Stat/Meta Data

Faster File Transfer

Tuning - File Directory Service

Caching Stat/Meta Data as much as possible

Large enough to support increasing number of users

Cache Settings(Directory Service)

Cache Type: Disk (RAM Disk)

Cache Size: 320 MB

-stat 4000 -daemons 6 -volumes 4096 -files 102400

Cache Usage(Directory Service)

Avg. Usage: 40 %

Usage Range: 10% ~ 90%

(We should have used Session Load Balancer for Better Distribution)

Tuning Result(Directory Service)

Avg. SOAP Processing Time: Less Than 0.1 Sec

Slowest SOAP Call: ListDir

Most Frequent Call: UserProfile

SOAP Calls in Directory Service

Authenticate

DriveInfo

ListDir

VolumeList

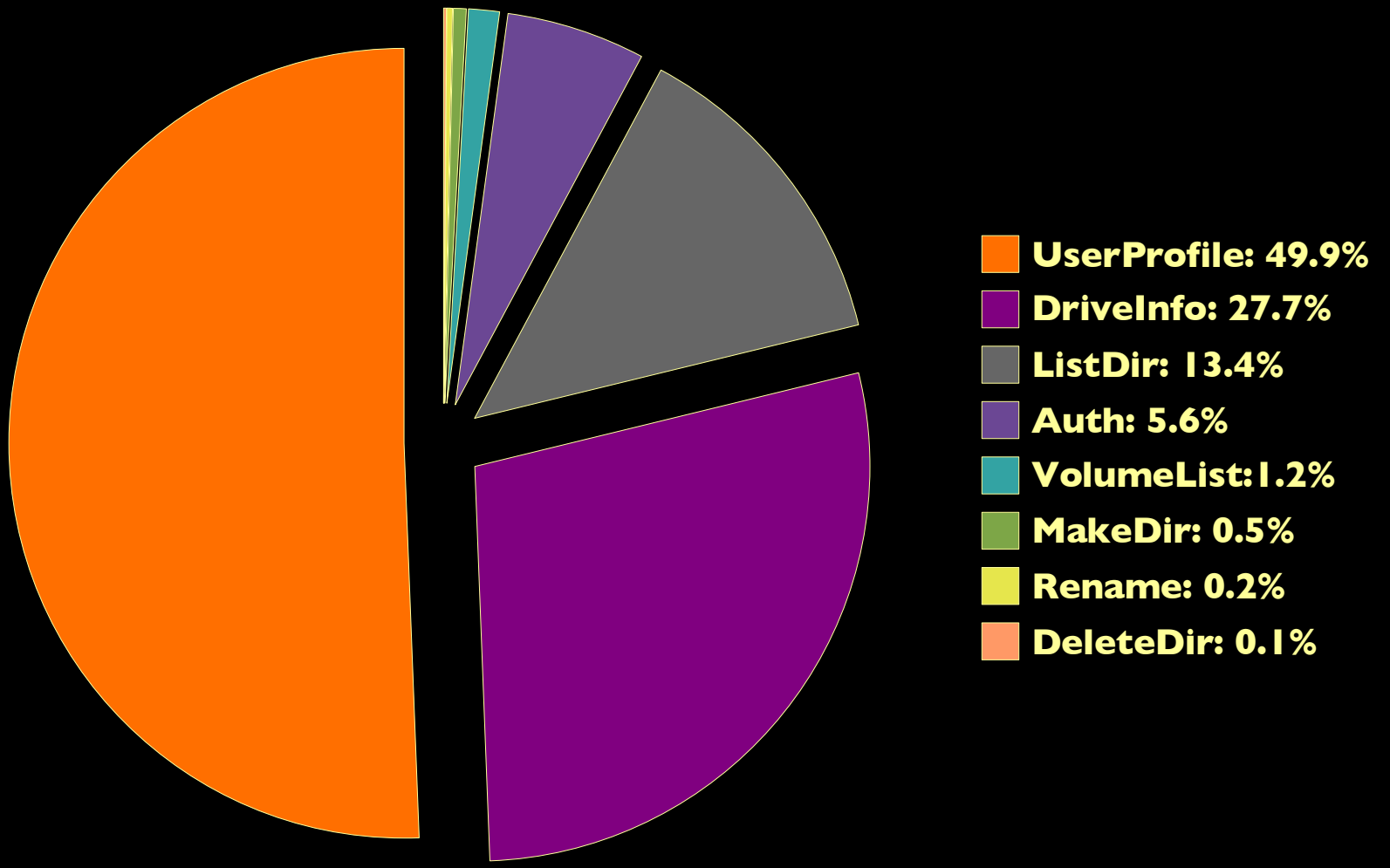
UserProfile

MakeDir/DeleteDir

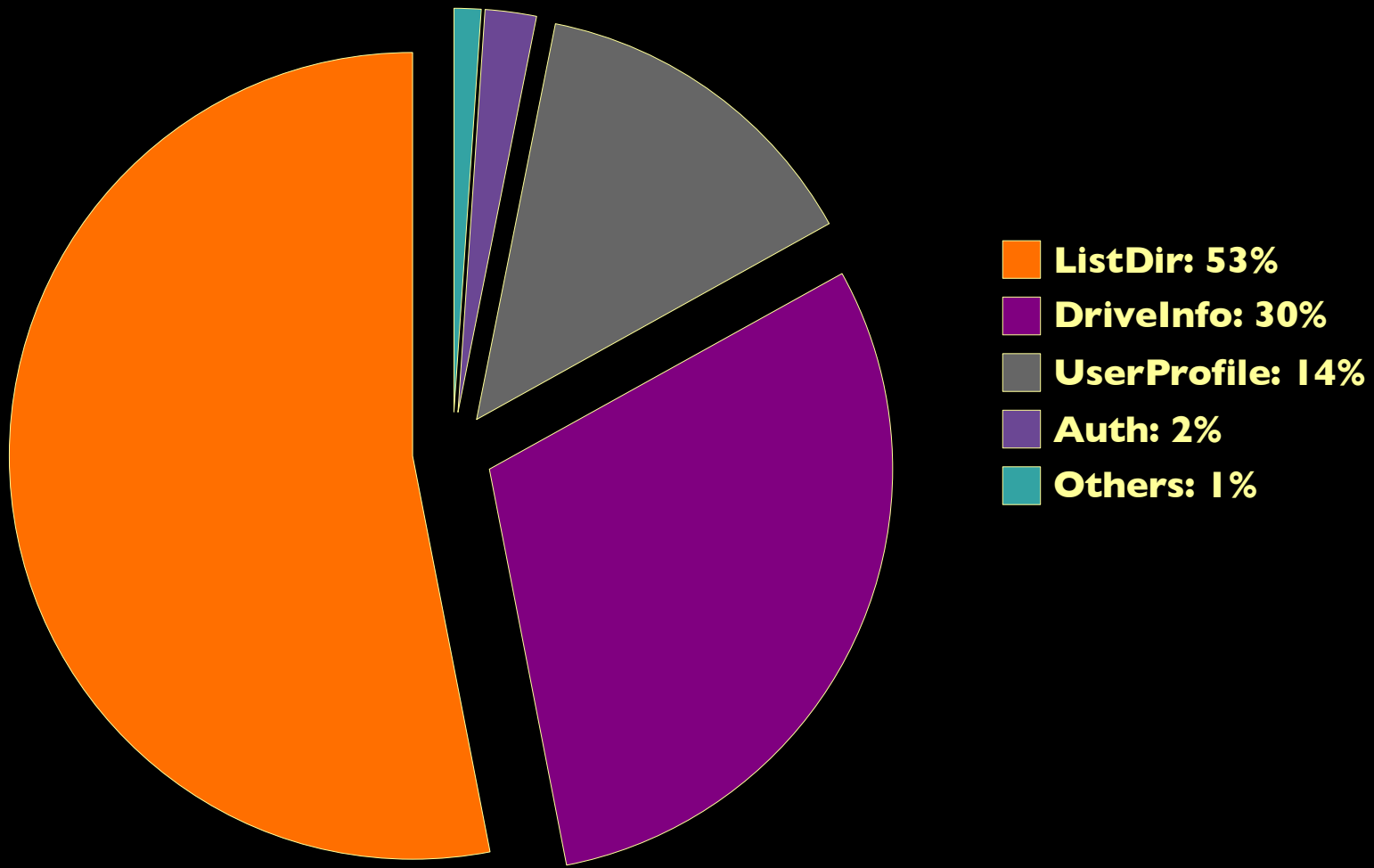
Rename

Much like Normal FTP Service Protocol

SOAP Call Frequency Distribution



SOAP Call Execution Time Distribution



Tuning - File Transfer Service

Caching is almost **Of No Use**: Files are **Too Big**

Memory Cache Works Better

IA64(64 Bit Architecture) shows Better Performance

Cache Settings(File Transfer Service)

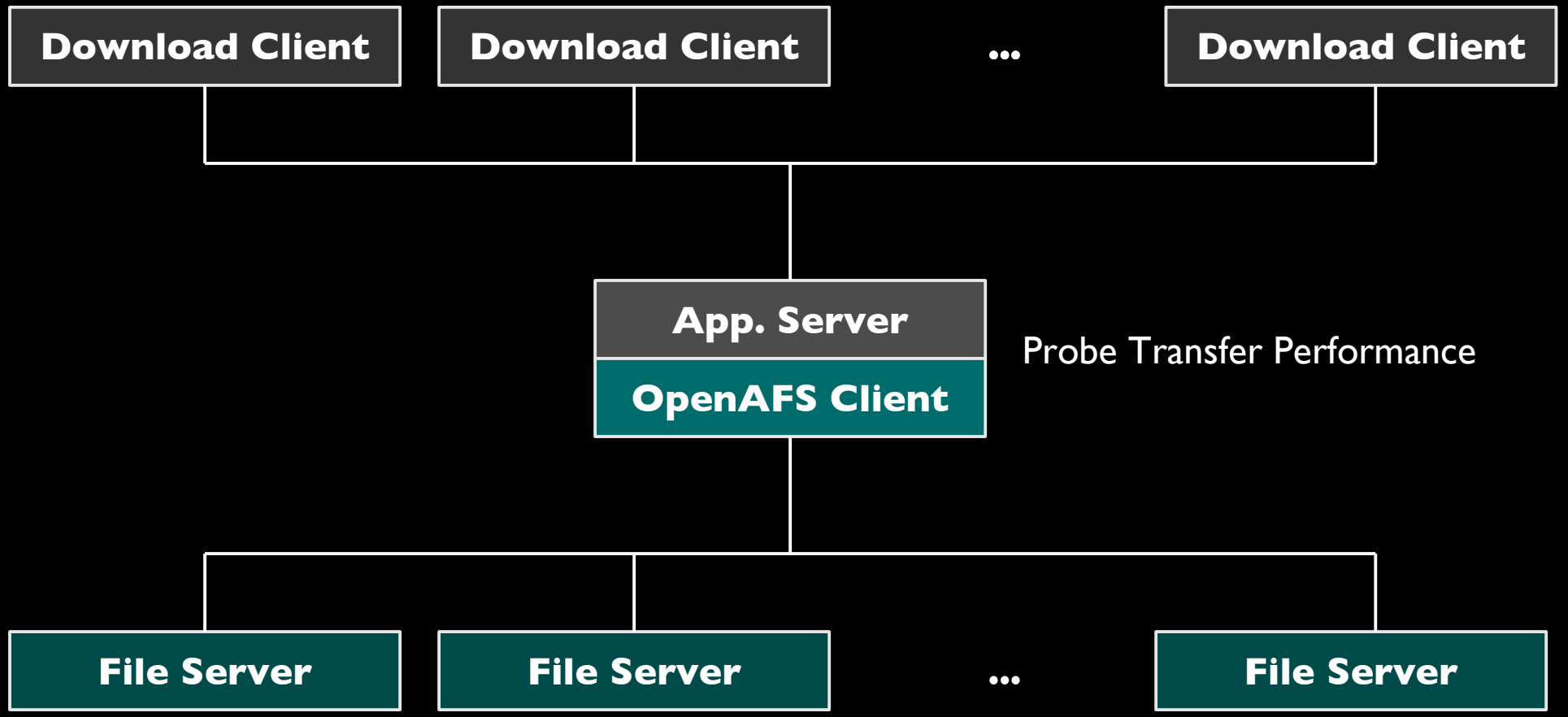
Cache Type: Memory

Cache Size: 80 MB

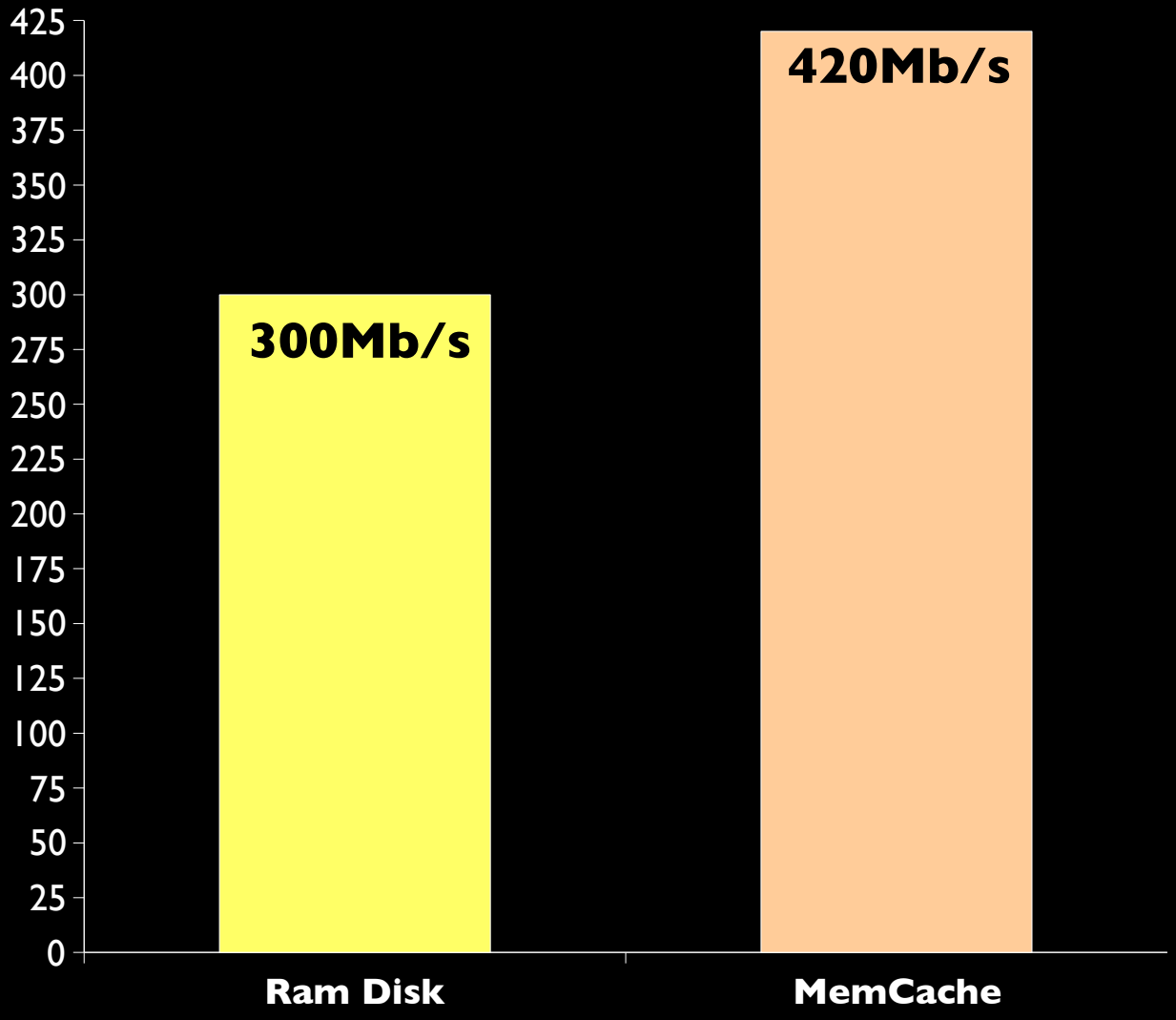
Chunk Size: 20

-daemons 6 -chunksize 20 -memcache -blocks 81920

File Transfer Performance Testing

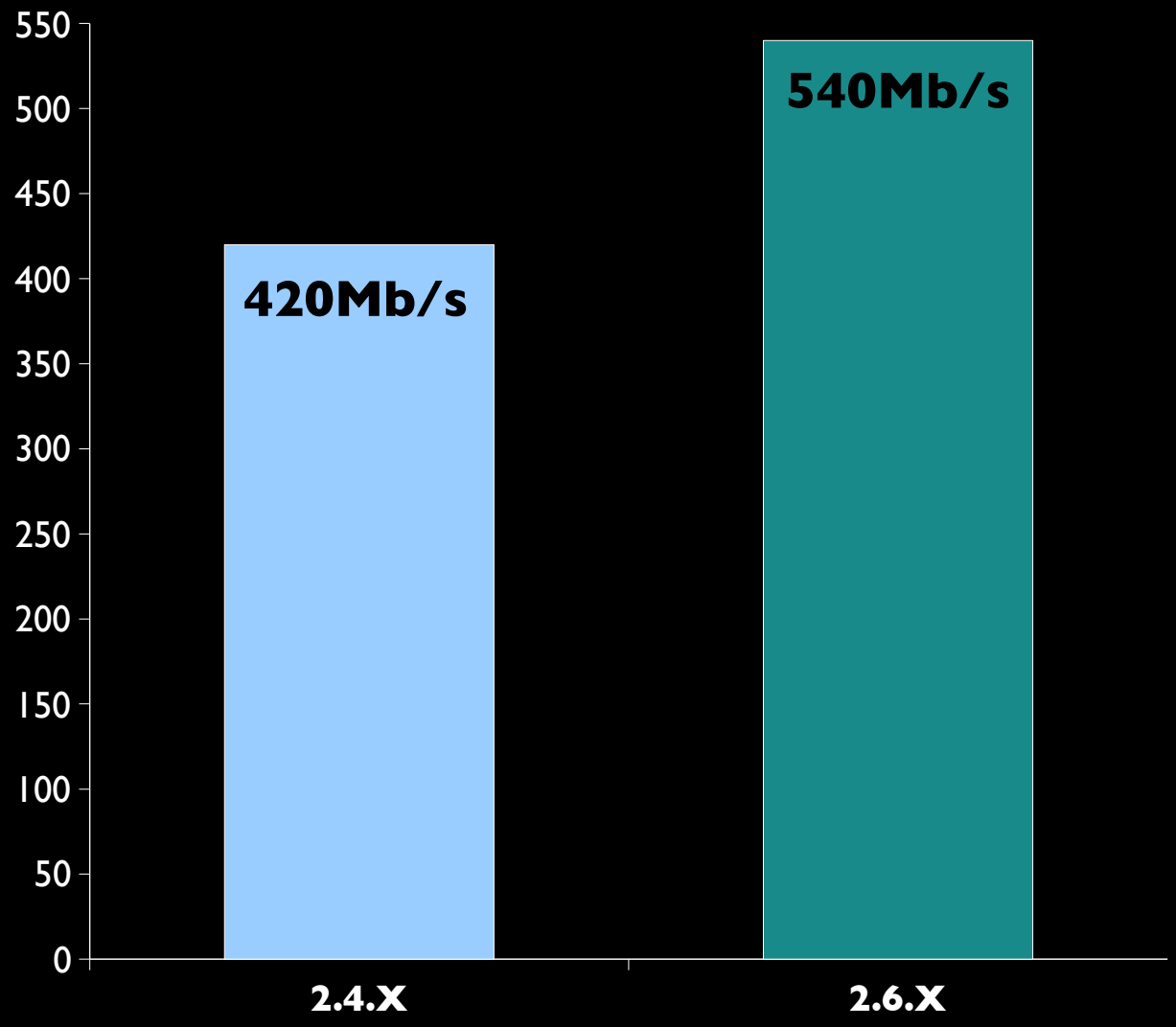


Max. Throughput – By Cache Type(IA32)



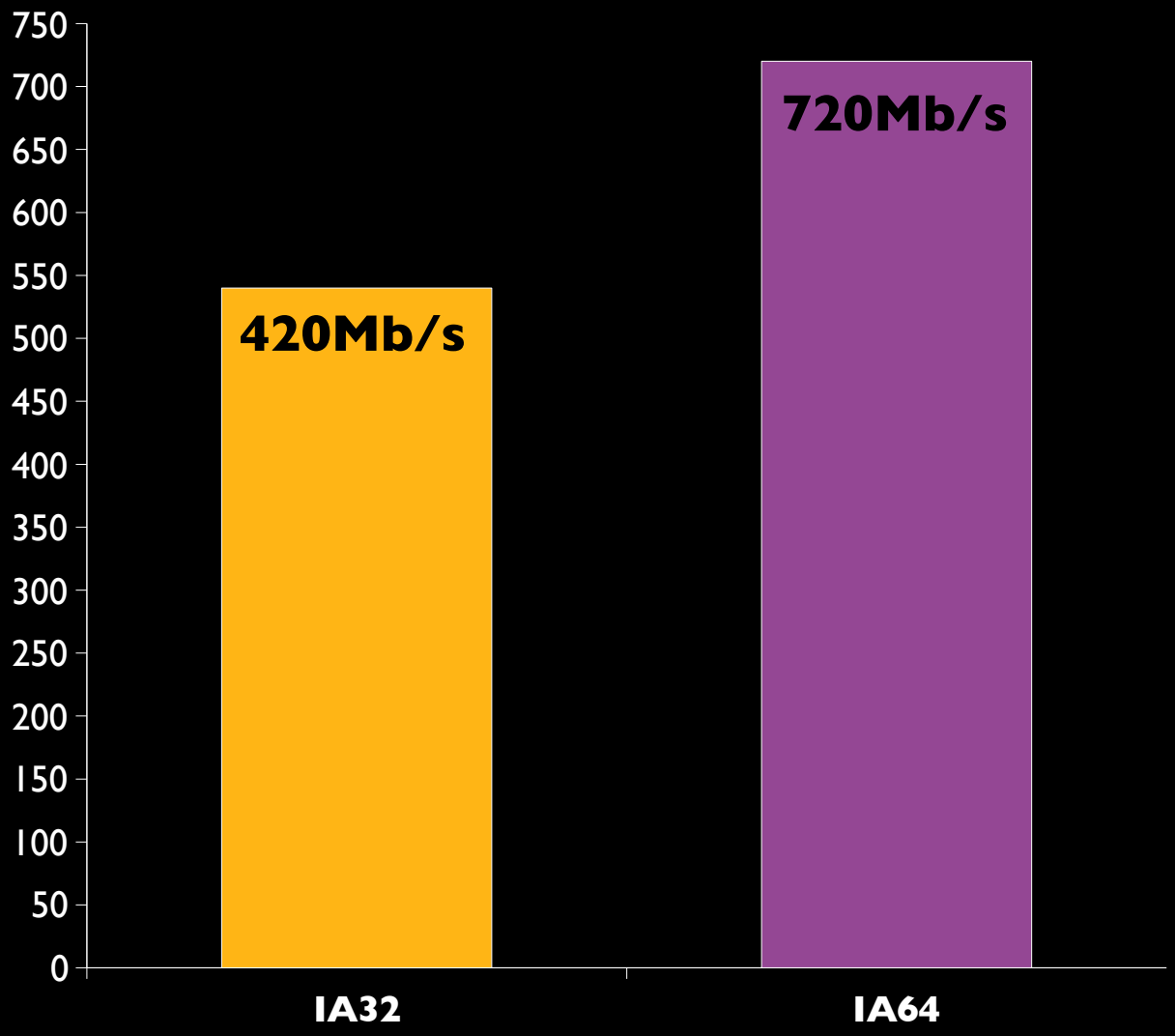
40% Better
with
MemCache

Max. Throughput – By Kernel Version(IA32)



24% Better
with
2.6 Kernel

Max. Throughput – By Architecture



33% Better with IA64

Best Config. for File Transfer Service

Architecture: IA64

Cache Type: MemCache

Kernel: 2.6.X

By **separating** client clusters with its role, we can

1. **optimize service performance better**
2. **detect performance problems better**

OpenAFS Network Grouping

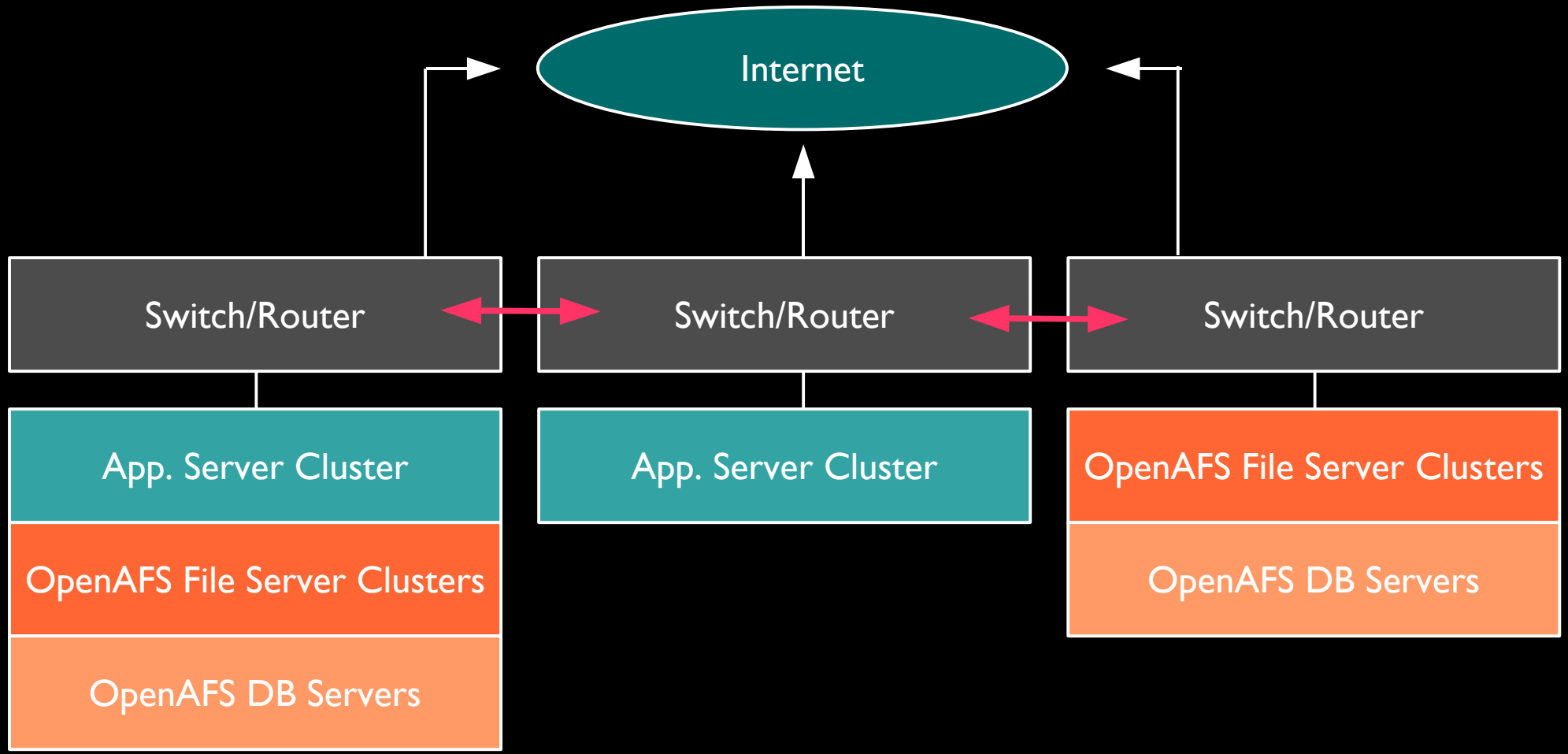
To Work Around Real World Limitations Like:

IP Address Range

IDC Space Allocation

Number of Switch/Router Port

Inappropriate Grouping of OpenAFS Network



Cross Group Traffic is Harmful To ...

Cost

Performance

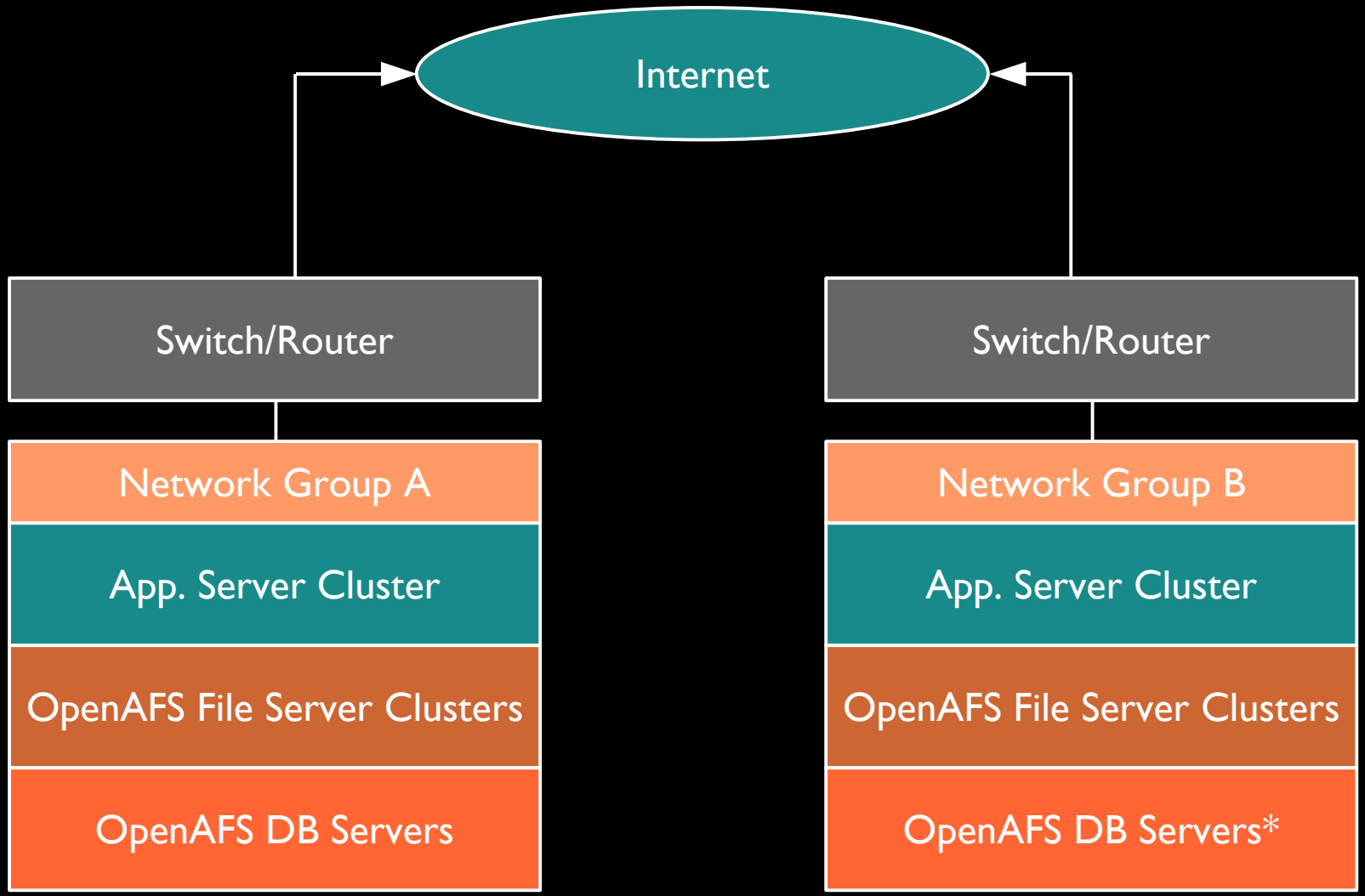
Efficiency

OpenAFS Network Group As Independent Working Component

which has **no dependency** with other network group

which has **no cross group traffic**

Proper Grouping of OpenAFS Network



Current Status

Deployed Sites

Company A

Company B

Cluster Size

About **200 Storage Servers**

About **300 Application Servers**

About **20 Database Servers**

About **10 Search Application Servers**

and about 20 auxiliary servers

Storage Space

Total Data Size: 0.5 PB

Avg. # of Volumes per Server: 1300

Avg. Data Size per Server: 2.6 TB

Avg. Storage Space per Server: 3.5 TB

Traffics

Max. **Outbound** Traffic: **XY** Gbit/s

Max. **Inbound** Traffic: **X** Gbit/s

Working Load

Avg. # of SOAP Requests per Day: **0.5 Million**

Number of Concurrent Logged In Users per Server: 700

Avg. # of Concurrent File Transfer Requests per Server: 30

S/W & H/W

Linux (Debian Sarge) Based System

LAMP Stack

OpenAFS

IA32/IA64 + IDE(ATA/SATA) RAID Box

Issues

Current Issues

Stability Issues

Performance Issues

Scalability Issues

Stability Issues

Service Affected by File Server Problems

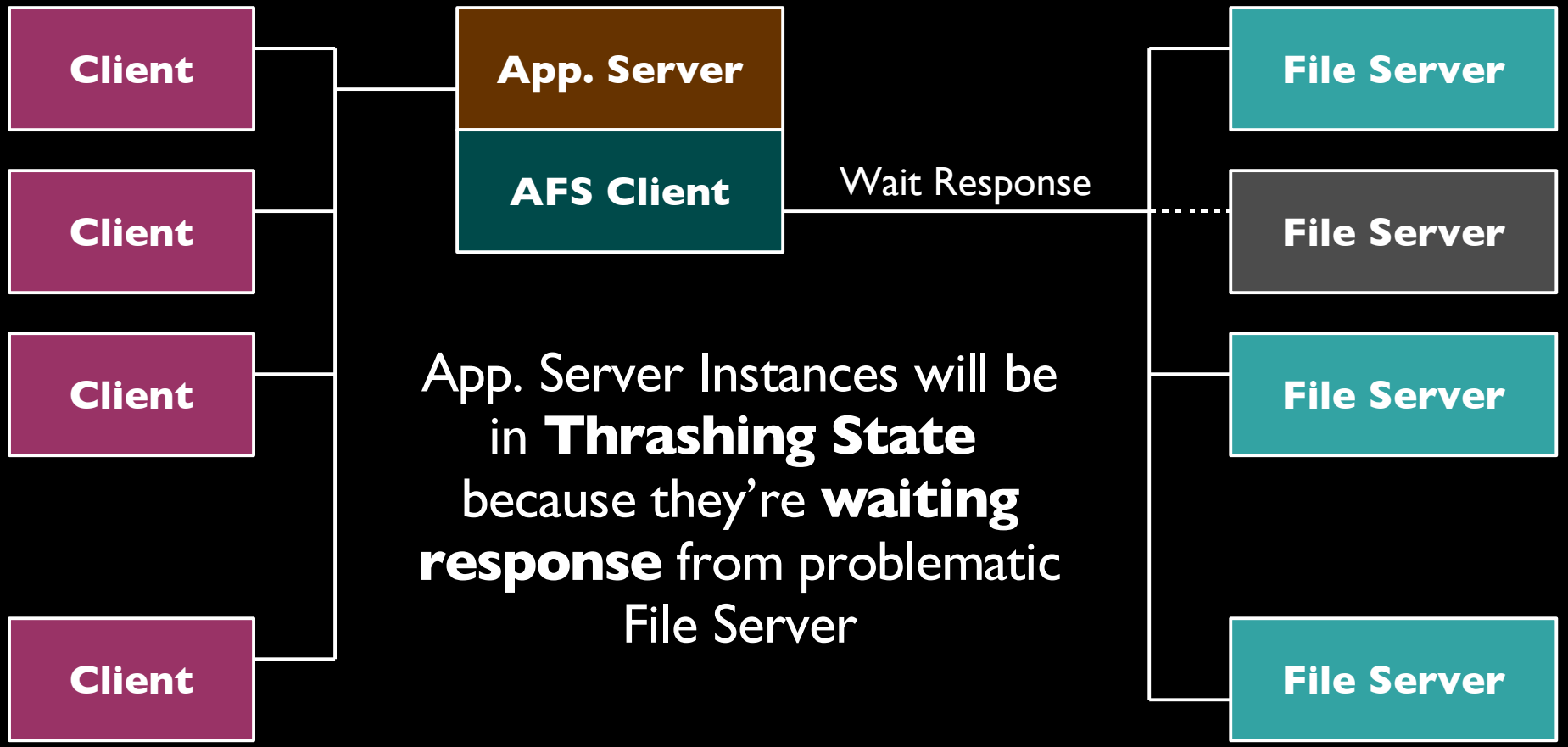
Administration Issue by Kernel Module Problem

AFS Database Server Fail-over Issue

File Server Problem

If a **File Server stops working** due to disk problem or so, **Application Server** which has requests to that File Server will be in **Thrashing State** and **Crash**.

Application Server Crashing Scenario



Solution

Server Problem should be **detected**

before

application server **crashing starts**

Our Approach

We cannot detect this problem systematically,

instead, we are trying to **prevent** this

and **pragmatic approach** to **detect** when this occurs

Preventing File Server Error

99% of File Server Error is caused by Disk Error,

we monitor **SMART** Data and

try to replace Disks before problem occurs.

If File Server Errors **detected**, requests to that server are **blocked/routed**

Client Side Monitoring

We use 'ls -l /afs' style **scripts** to monitor OpenAFS error and if error detected, we should perform "**cold restart**"

AFS Database Fail-Over Issue

We have master-replication configuration, but we need more **faster fail-over system**.

We have no solid solution on this yet.

Performance Issues

File Server Performance: RAID, Network Traffic

OpenAFS Client Performance: CPU

File Server Performance Issues

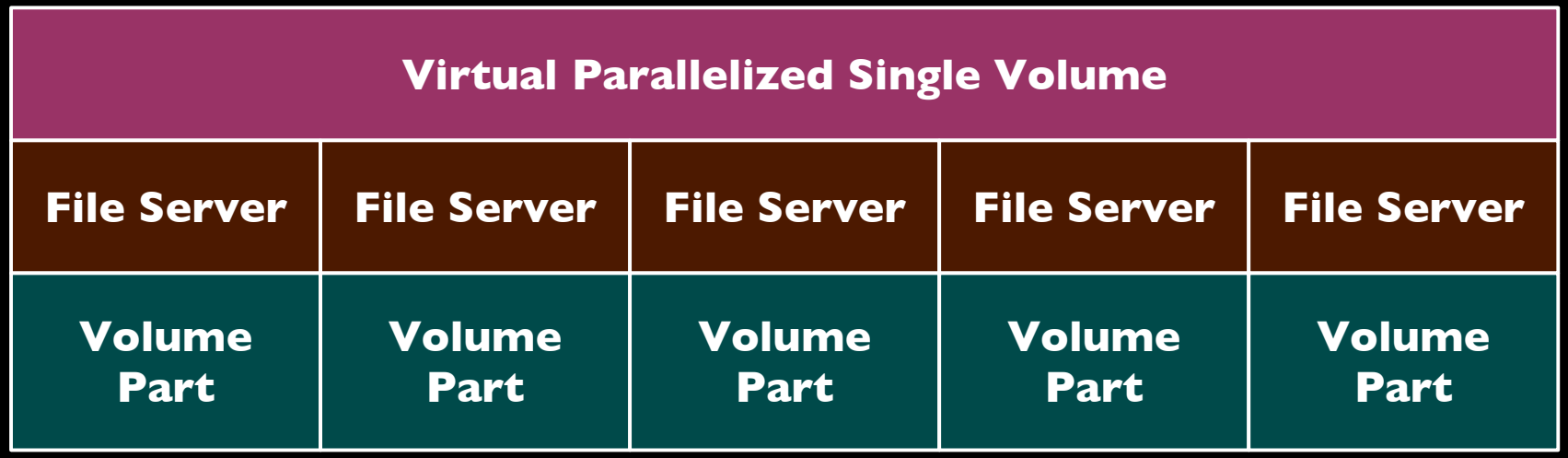
Can we make **RAID of File Servers**

1. to **balance** too **many requests** to single server

2. to provide **faster writing** performance ?

Clustered Volumes – RAID of File Servers

5 Times Faster Read/Write
Fail-Over with Parity Data



OpenAFS Client Performance Issue

It seems that client performance highly depends on

CPU performance/architecture

How much CPU performance is required for 1 Gbit traffic?

Or is it just tuning problem?

Scalability Issues

What is the **limit** of ...

Total Number of Clients

Total Number of Servers

Total Number of Volumes

Total Number of Files

and **other unknown limitations**?

Summary

OpenAFS has been
good storage space infrastructure
for **eFolder:**

Open Source

Scalable Architecture

Easy to Administer

Global, Uniform Namespace

We Need More

Solid Monitoring System

and **Administrative Strategy/Experience**

to Overcome following **Stability Issues**:

1. **File Server Problem**
2. **Client Kernel Module Problem**
3. **AFS Database Server Fail-over**

Current Performance is not bad but

we want more performance

Thank you