JavaOne

Project Caroline: Platform As A Service, For Your Service, At Your Service

Bob Scheifler

Distinguished Engineer
Sun Microsystems, Inc.
http://research.sun.com/projects/caroline

TS-1991

Goal of This Talk

What you will gain

Learn how Project Caroline helps SaaS providers develop services rapidly, update them frequently, and automatically flex resource use to match changing runtime demands

Agenda

Project Caroline At-a-Glance

System Architecture

Programmatic Resource Allocation

Example Application

Current Implementation

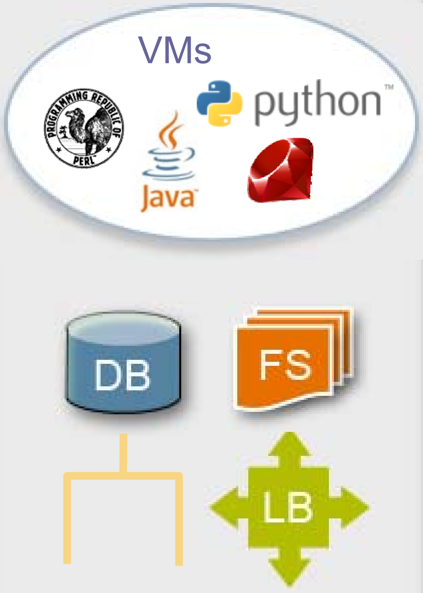
Summary

What Is Project Caroline?

- Advanced research project at Sun Microsystems
- Hosting platform for development and delivery of dynamically scalable Internet-based services
- Programmatically configurable pool of virtualized compute, storage, and networking resources

Developer View

Project Caroline Platform Resources

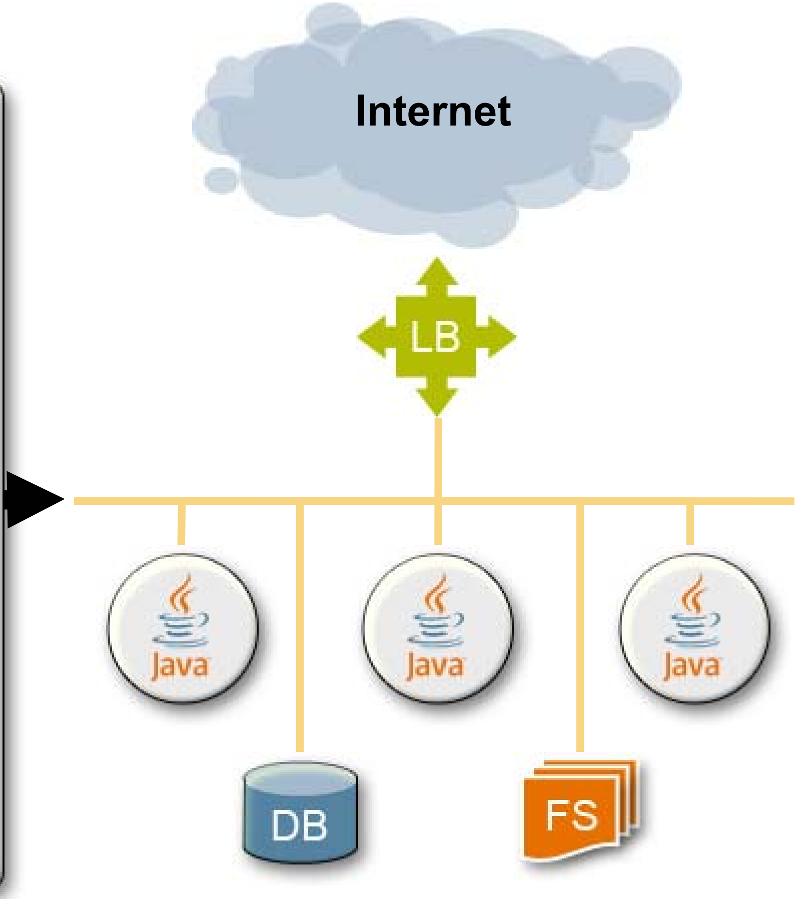


+

Your Service Code

Launch my Internet app across 2 load-balanced VMs, connected to a backend database.

Add or remove VMs to match demand (recycle removed VMs)



For Small-to-Medium SaaS Providers

- Who wants to embrace new business models and processes
  - Offer long-running and rapidly evolving services
  - Flex use-of-platform resources to match changing customer demands
  - Leverage hosted infrastructure
  - Use higher-level programming languages
    - Java™ programming language, Perl, Ruby, Python, ...

Key Platform Features (1)

Programmatically re/configure systems

- Programmatically allocate, monitor, and control virtualized compute, storage, and networking resources
- Services can themselves update and flex platform usage, dynamically and without human intervention

Key Platform Features (2)

High level virtualization abstractions

- Resources are exposed through high level abstractions
  - Language level VMs
  - Networks
  - Network accessible file systems and databases
- Improves developer productivity
- Insulates code from infrastructure changes

Key Platform Features (3)

Single system view

- Presents a horizontally scaled pool of resources as a single system
- Provides developers with a unified platform for allocating and controlling these resources
- Draws on resource pool to meet allocation requests of multiple applications

Agenda

Project Caroline At-a-Glance

System Architecture

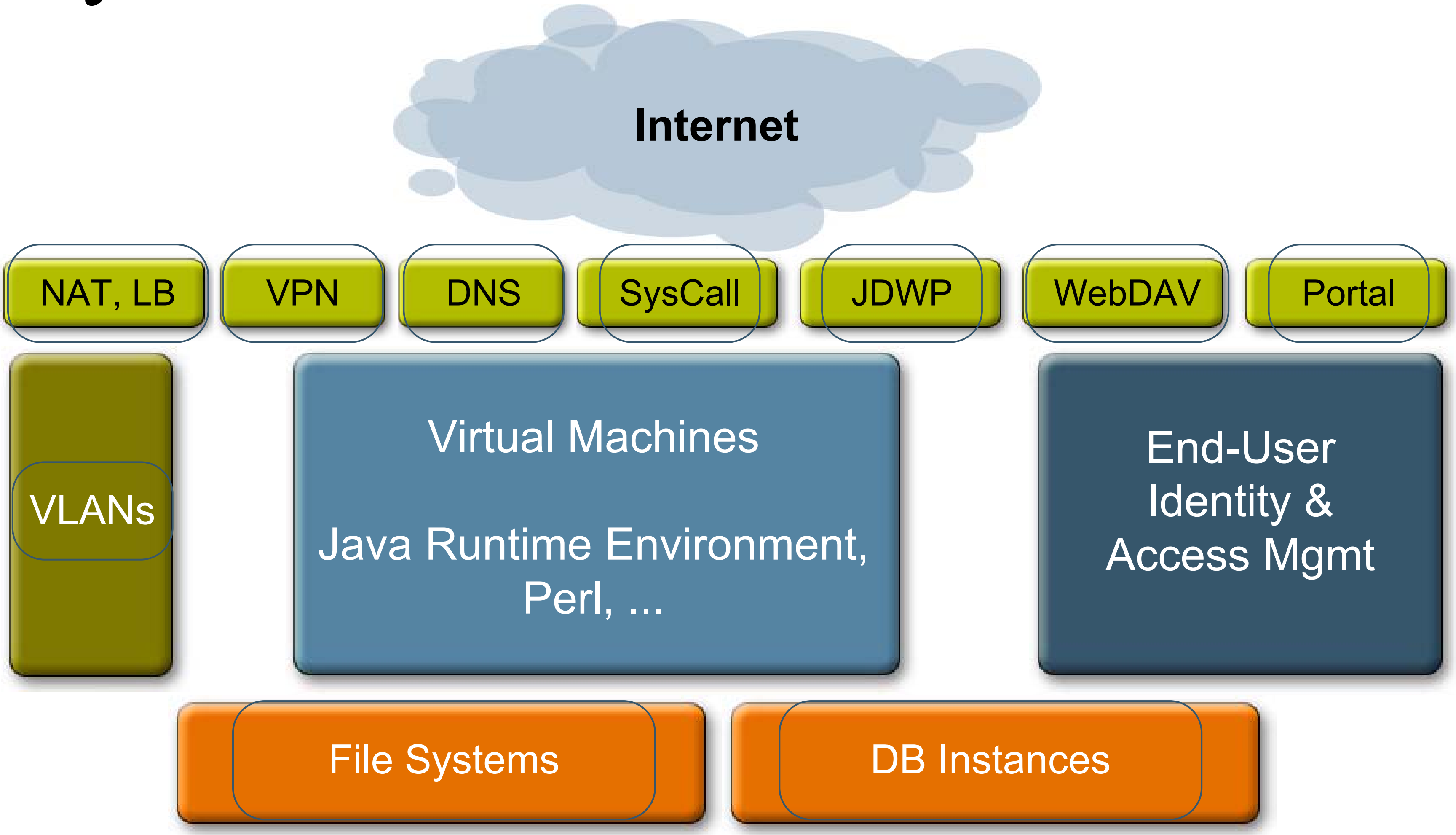
Programmatic Resource Allocation

Example Application

Current Implementation

Summary

System View


Internet
NAT, LB
VPN
DNS
SysCall
JDWP
WebDAV
Portal
VLANs
Virtual Machines
Java Runtime Environment, Perl, ...
End-User Identity & Access Mgmt
File Systems
DB Instances

Software Layers—APIs

- “System call” API
  - Java Platform, Standard Edition (Java SE)
  - Programmatic system resource allocation and control
  - Usable both on-grid and off-grid
  - Off-grid API and VPN allow mixed deployments
- Higher-level libraries and frameworks
  - Focus on selected existing packages
  - Deployment and management automation
    - Example: for Servlet containers
  - Aiding SaaS delivery
    - Examples: end-user management, master+workers w/flexing

Software Layers—Services

- Portal tools also deployable by customers
  - Forums, blogs, wikis, help desks, knowledge bases
- System-wide Liberty identity provider
  - End-user identity management and access control
  - End-users are the principals
  - Customer applications are the service providers

Software Layers—IDE

- Off-grid IDE used to develop, debug and deploy on-grid application
- Automated deployment
  - Using WebDAV and off-grid system call API
- Debug on-grid Java platform processes
- Monitor on-grid resources
- NetBeans™ software and Eclipse

Agenda

Project Caroline At-a-Glance

System Architecture

Programmatic Resource Allocation

Example Application

Current Implementation

Summary

Programmatic Resource Allocation

- Process
- File System
- DB instance
- Network (VLAN)
- IP address (internal, Internet)
- Internet connectivity (NAT, L4/L7 Load Bal, VPN)
- Host name
- Host name mapping (DNS)

Common Resource Features

- Named when created
  - Avoids lost resource on network failure or caller crash
  - Separate name space for each resource class
- Identified by UUID (assigned by system)
  - Names can be reused over time
- Basic operations
  - Create (name, configuration)
  - Find (name, UUID, configuration content, meta-data)
  - Change configuration
  - Destroy

Process Configuration (1)

- Command line
  - argv[0] is an enum: Java runtime environment, Perl, …
- File system mounts
- IP addresses
- IP traffic blocking (IP Filter)
- Exit action (park, restart, destroy)

Process Configuration (2)

- Stdin/stdout/stderr files
- Working directory
- Home directory
- Environment variables
- HW constraints (shared/exclusive cores, memory)
- Non-collocation constraints

Process-Specific Operations

- Start
- Stop
  - Gentle (let shutdown hooks run)
  - Hard
- Generate thread dump
- Get current state
  - Starting, running, or not running
  - Incarnation # (# of start calls)
  - Last process outcome (exit value, signal, exception)
  - OK or stuck due to start failure

Process Model Impacts

- Native libraries not supported
  - OS and HW independence
- No local Runtime.exec
  - “Remote” exec via system call API
  - No stdio pipes between processes
    - Alternate communication mechanisms used instead

File Systems

- Solaris™ Zettabyte File System (ZFS) with NFS access
  - Thousands of file systems from one storage pool, efficiently and reliably
- Base: normal read/write file system
- Snapshot: read-only copy of a base or clone
  - Shares space with original until original is modified
- Clone: copy-on-write clone of a snapshot
  - Shares space with snapshot until clone is modified

File System Configuration

- Storage reservation
- Storage quota
- Access control
  - Mounts by customer’s processes
  - Mounts by other customer’s processes
  - WebDAV

File System-Specific Operations

- Rollback base or clone to most recent snapshot
- Backup to file
  - Entire snapshot
  - Delta between two snapshots
- Restore from backup
- Get current disk usage

Databases

- PostgreSQL instances with local storage
- System-managed installation

- Can always bring your own Java DB
  - Embedded or running as a separate process

Database Configuration

- IP addresses
- Storage reservation
- Storage quota
- Access control
  - Connections from processes on same networks

Internal Networks

- Network configuration
    - # of allocatable IP addresses
    - Access control
        - Address allocation by other customers
- Network-specific operations
    - Allocate IP address
    - Free IP address

Internet Connectivity

- Inbound and outbound traffic
  - Direct binding to Internet-routable addresses
  - Static NAT (Internet address ↔ internal address)
  - VPN (bind off-grid process to internal address)
- Inbound traffic
  - L4 load balancing: TCP, SSL, UDP
    - Internet address+port → {internal address+port}
  - L7 load balancing: HTTP, HTTPS
    - Internet address+port+{URI filter → {internal address+port}}
- Outbound traffic
  - Dynamic NAT (internal address → Internet address)

DNS Configuration

- Host name → list of IP addresses

Agenda

Project Caroline At-a-Glance

System Architecture

Programmatic Resource Allocation

Example Application

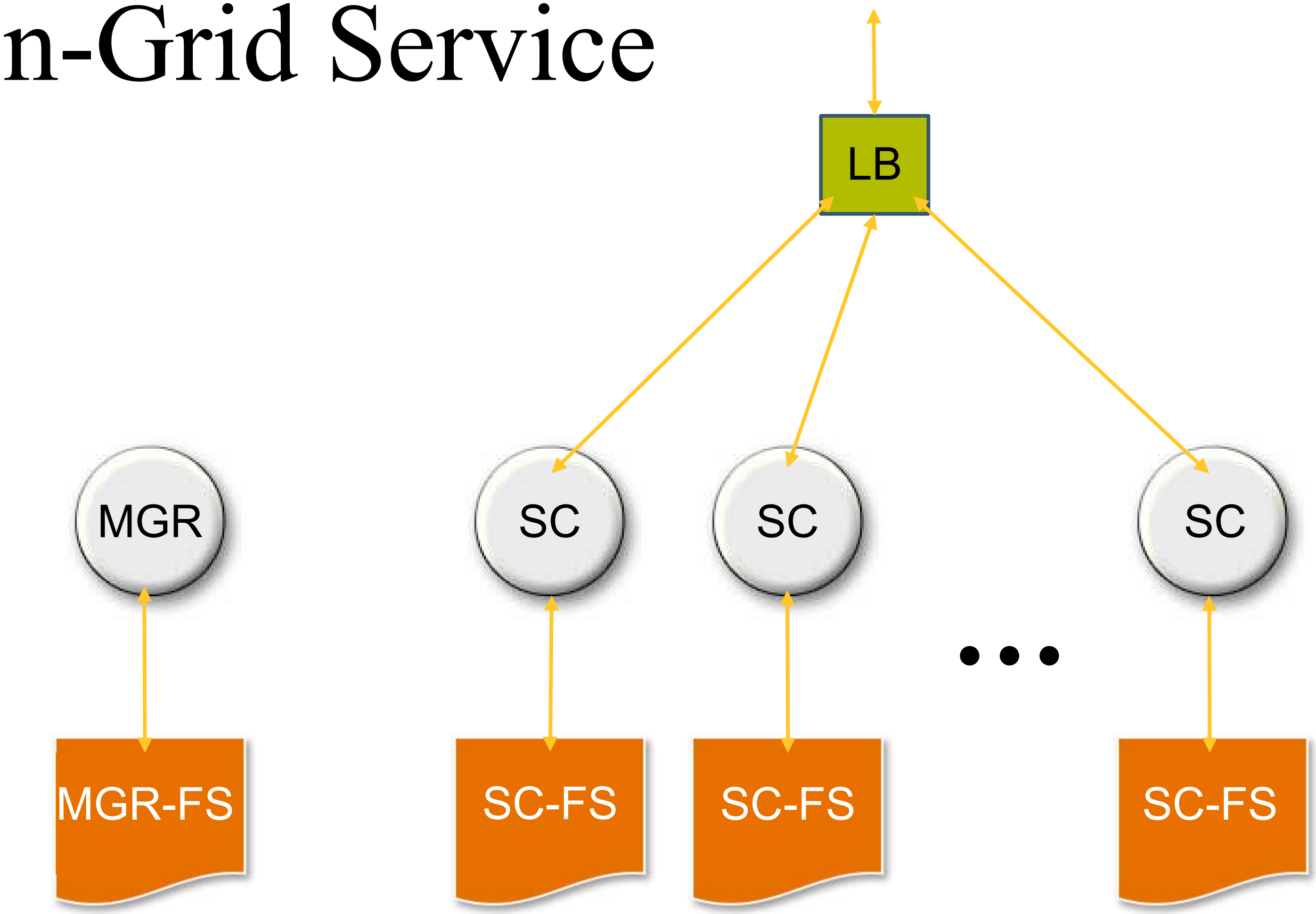
Current Implementation

Summary

Example Application

- On-grid manager process
  - Sets up N replicated Servlet containers
  - Sets up Internet-facing load-balancer
  - Flexes the number of replicas up and down to meet changing demand
- Off-grid setup app
  - Automates setup of manager process

On-Grid Service



Off-Grid App—Initial Setup

```
Grid grid = GridFactory.getGrid(gridURL, userid, passwd);

FileSystem mgrFS = grid.createFileSystem(“manager”);

WebdavResource wdr =
  new WebdavResource(new HttpURL(webdavURL + “/manager/”));
wdr.setUserInfo(userid, passwd);
wdr.putMethod(“manager.jar”, new File(“manager.jar”));
wdr.putMethod(“container.zip”, new File(“container.zip”));
wdr.putMethod(“servlets.zip”, new File(“servlets.zip”));

Network myNet = grid.createNetwork(“myNet”, 16);
```

Off-Grid App—Start Manager

```
ProcessConfiguration cfg = new ProcessConfiguration();
cfg.setRuntimeEnvironment(RuntimeEnvironment.JAVA);
cfg.setCommandLine(new String[]{“-jar”, “manager.jar”});
cfg.setFileSystems(Collections.singleton(
   new MountParameters(mgrFS, “manager”)));
cfg.setWorkingDirectory(“/files/manager”);
NetworkAddress mgrIP = myNet.allocateAddress(“manager”);
cfg.setNetworkAddresses(Collections.singleton(mgrIP));
cfg.setProcessExitAction(ProcessExitAction.RESTART);
cfg.setSystemSinks(“stdout.txt”, false,
                   “stderr.txt”, false);

ProcessRegistration mgrPR =
   grid.createProcessRegistration(“manager”, cfg);

mgrPR.start();
```

Manager Process—
Initial Setup and Replica Setup

```
// instance variables
Grid grid = GridFactory.getProcessContext().getGrid();
Network myNet = grid.getNetwork(“myNet”);
FileSystemSnapshot goldSnap;

FileSystem goldFS = grid.createFileSystem(“golden”);
grid.mountFileSystem(new MountParameters(goldFS, “golden”));
Util.unzip(“container.zip”, “/files/golden”);
Util.unzip(“servlets.zip”, “/files/golden”);
grid.unmountFileSystem(“golden”);
goldSnap = goldFS.createSnapshot(“goldsnap”);

for (int i = 1; i <= N; i++) {
    addReplica();
}
```

Manager Process—addReplica() Part 1

```
// more instance variables
LinkedList<RealService> services = new LinkedList<RealService>();
LinkedList<FileSystem> clones = new LinkedList<FileSystem>();
LinkedList<ProcessRegistration> procRegs =
   new LinkedList<ProcessRegistration>();

void addReplica() {
   String replicaName = “replica-” + (services.size() + 1);
   NetworkAddress replicaIP = myNet.allocateAddress(replicaName);
   services.add(new RealService(replicaIP, 8080));
   FileSystem cloneFS = goldSnap.clone(replicaName);
   clones.add(cloneFS);
   grid.mountFileSystem(new MountParameters(cloneFS, “server”));
   editConfigurationFiles(“/files/server”, …);
   grid.unmountFileSystem(“server”);
```

Manager Process—addReplica() Part 2

```
    ProcessConfiguration cfg = new ProcessConfiguration();
    cfg.setRuntimeEnvironment(RuntimeEnvironment.JAVA);
    cfg.setCommandLine(new String[]{<<container command line>>});
    cfg.setFileSystems(Collections.singleton(
        new MountParameters(cloneFS, “server”)));
    cfg.setWorkingDirectory(“/files/server”);
    cfg.setNetworkAddresses(Collections.singleton(replicaIP));
    cfg.setProcessExitAction(ProcessExitAction.RESTART);
    cfg.setSystemSinks(“stdout.txt”, false, “stderr.txt”, false);

    ProcessRegistration replicaPR =
        grid.createProcessRegistration(replicaName, cfg);
    procRegs.add(replicaPR);

    replicaPR.start();
}
```

Manager Process—LB Setup

```
// more instance variables
L4VirtualServiceConfiguration lbCfg;
NetworkSetting myLB;

lbCfg = new L4VirtualServiceConfiguration();
NetworkAddress extIP =
   grid.allocateInternetAddress(“external”);
lbCfg.setExternalAddress(extIP);
lbCfg.setPort(80);
lbCfg.setProtocol(Protocol.TCP);
lbCfg.setRealServices(services);

myLB = grid.createNetworkSetting(“myService”, lbCfg);

grid.bindHostName(myHostName,
                  Collections.singletonList(extIP));
```

Manager Process—Flex Up

```
addReplica();

lbCfg.setRealServices(services);

myLB.changeConfiguration(lbCfg);
```

Manager Process—Flex Down

```
RealService svc = services.removeLast();
lbCfg.setRealServices(services);

myLB.changeConfiguration(lbCfg);

ProcessRegistration replicaPR = procRegs.removeLast();
replicaPR.destroy();

svc.getNetworkAddress().delete();

FileSystem cloneFS = clones.removeLast();
cloneFS.destroy();
```

Agenda

Project Caroline At-a-Glance

System Architecture

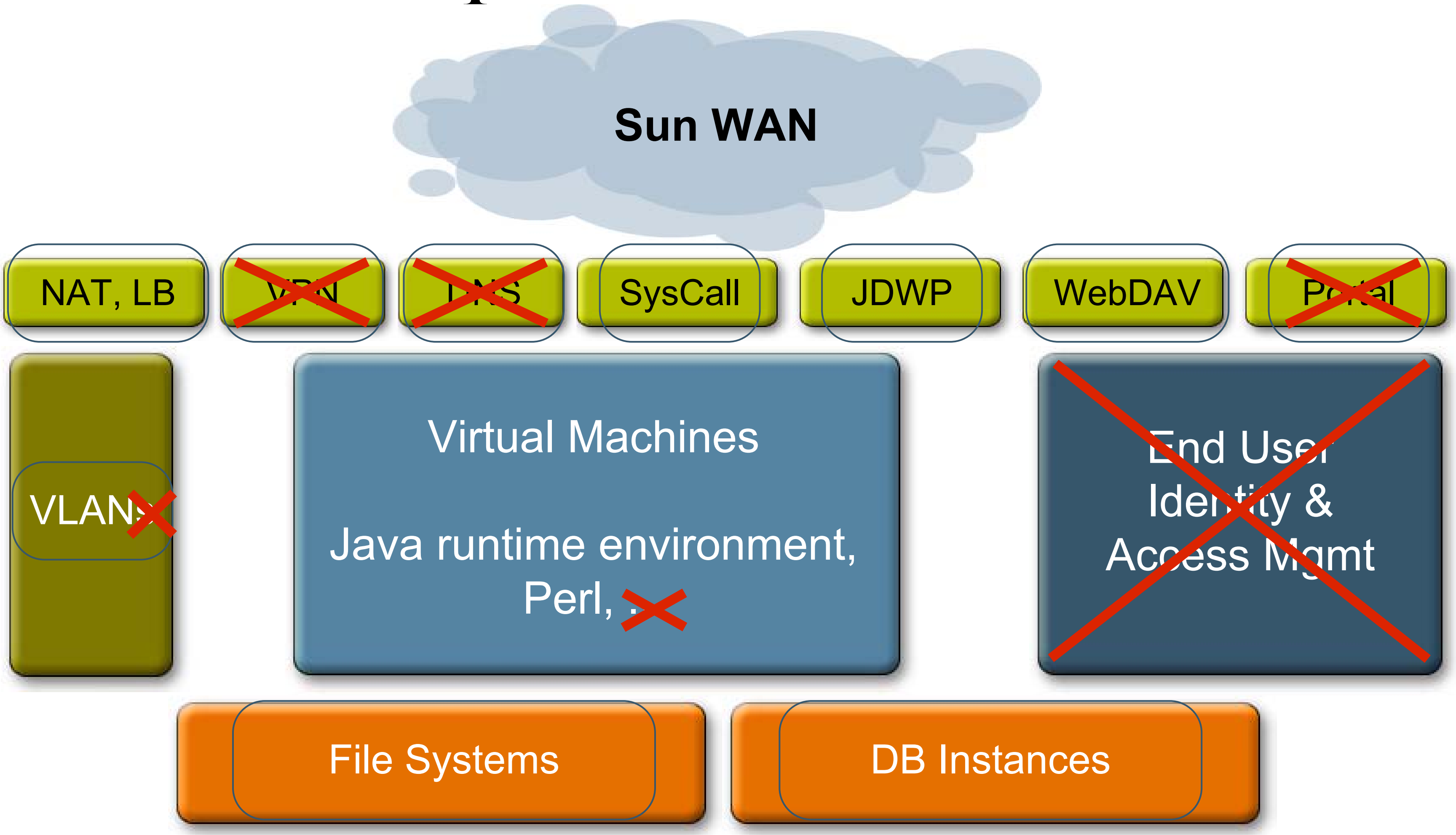
Programmatic Resource Allocation

Example Application

Current Implementation

Summary

Current Implementation



Sun WAN

NAT, LB

~~VPN~~

~~DNS~~

SysCall

JDWP

WebDAV

~~Portal~~

VLAN~~s~~

Virtual Machines

Java runtime environment, Perl, ~~...~~

~~End User Identity & Access Mgmt~~

File Systems

DB Instances

Processes

- Basic functionality in place
- WIP
  - More than one IP address
  - IP traffic blocking
  - HW constraints
  - Non-collocation constraints
- TBD
  - Beyond Java runtime environment and Perl

Storage

- Basic functionality in place
- File Systems
  - WIP
    - Dynamic mounts
  - TBD
    - Backup and restore
    - Access control
- Databases
  - TBD
    - Storage reservation and quota
    - Administrative control

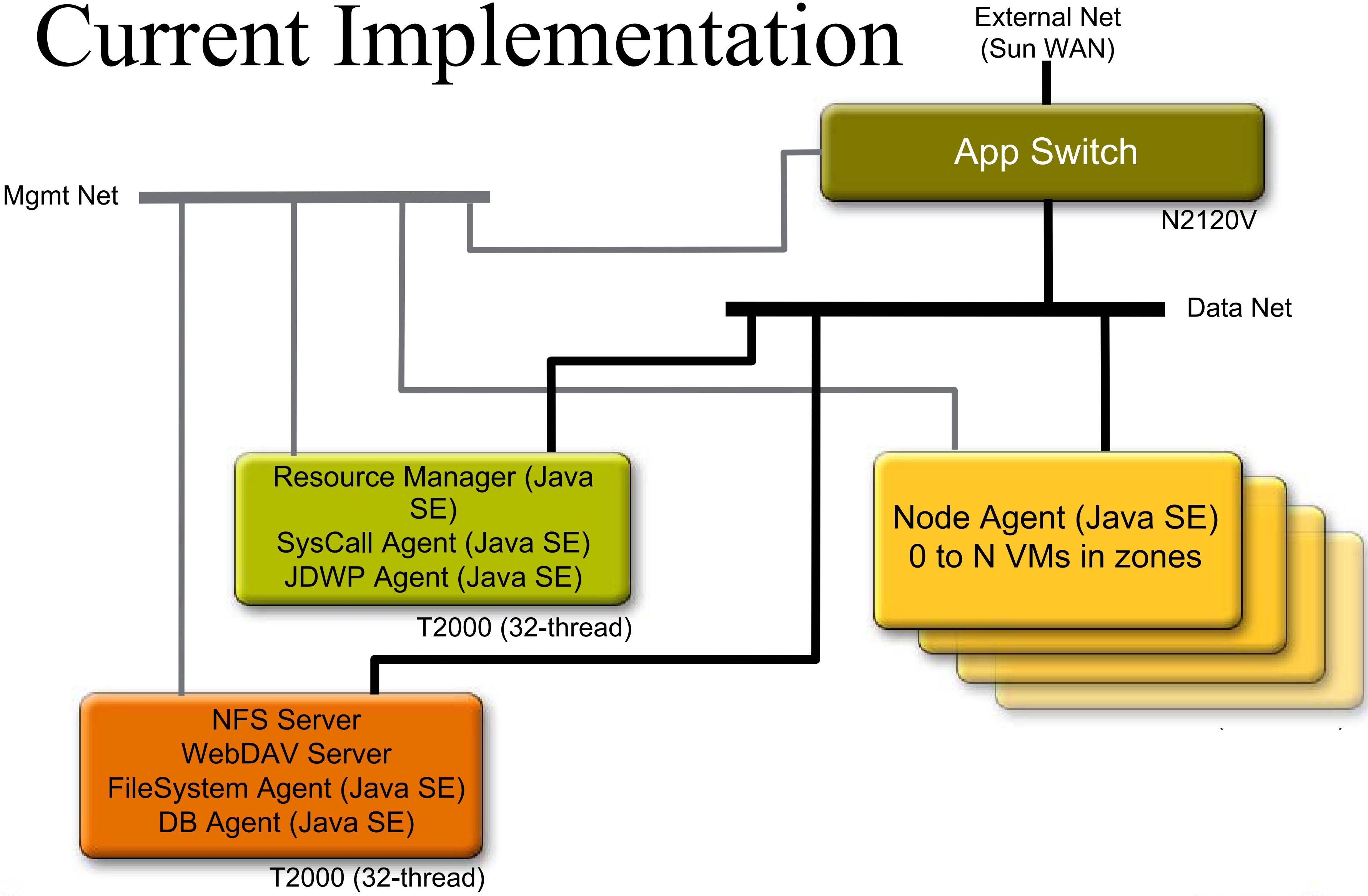
Networking

- Single shared internal network
- Basic NAT and LB connectivity in place
- WIP
  - Allocatable internal Networks
  - SSL and HTTPS load balancing
  - DNS
  - VPN
- TBD
  - Allocatable Internet addresses

Other Software Layers

- Basic NetBeans software support in place
- WIP
  - LB’d Servlet container deployment automation
- Investigations
  - Libraries, Frameworks, and Portal tools
- TBD
  - End-user identity and access management
  - Eclipse support

Current Implementation



External Net
(Sun WAN)

App Switch

N2120V

Mgmt Net

Data Net

Resource Manager (Java SE)
SysCall Agent (Java SE)
JDWP Agent (Java SE)

T2000 (32-thread)

Node Agent (Java SE)
0 to N VMs in zones

NFS Server
WebDAV Server
FileSystem Agent (Java SE)
DB Agent (Java SE)

T2000 (32-thread)

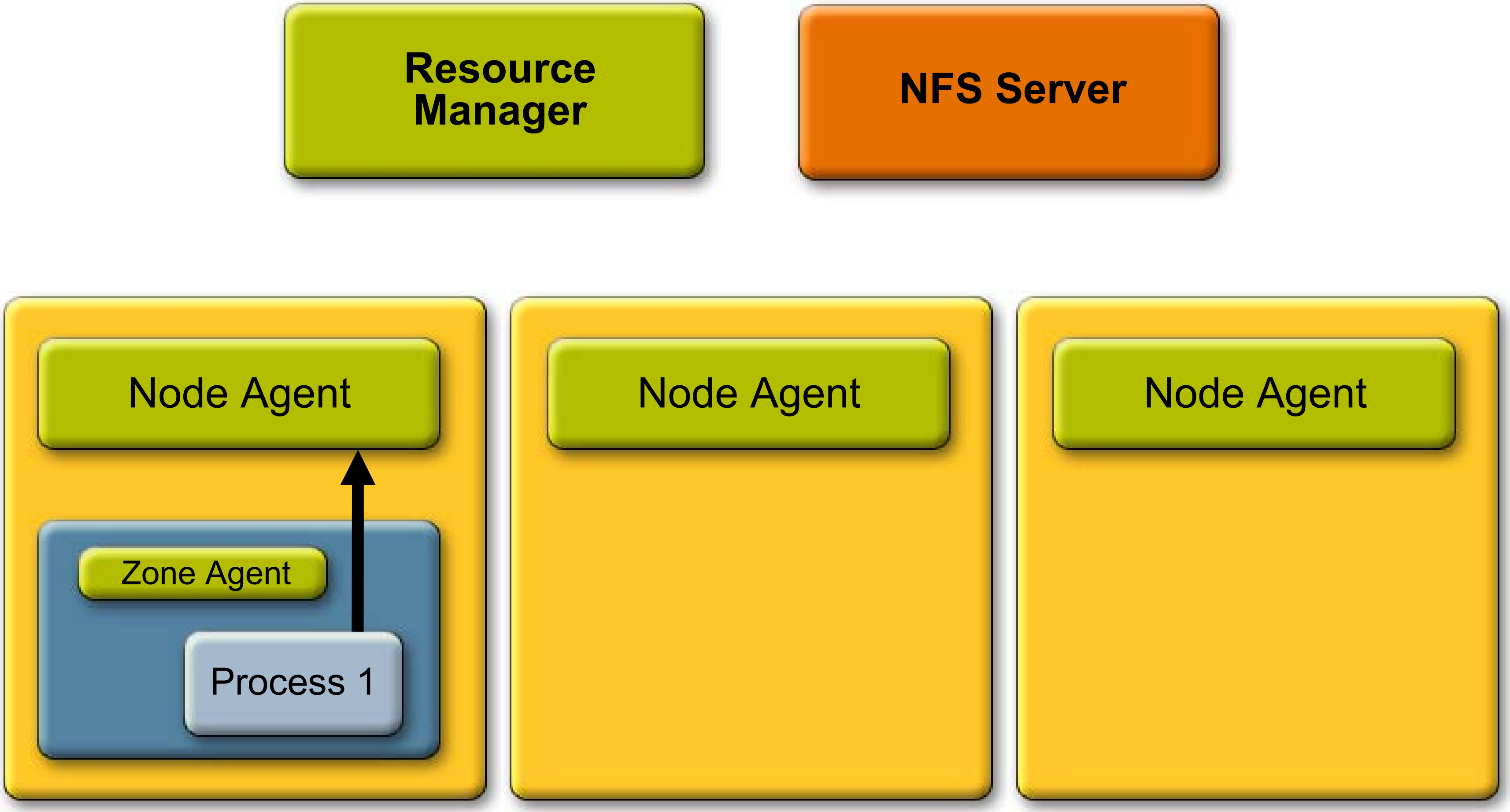
Solaris 10 OS Feature Use

- Zones
- Solaris ZFS, snapshots, clones
- NFS
- PostgreSQL
- IP Instances, IP Filter, Crossbow
- Resource Pools
- Fair Share Scheduler
- Extended Accounting

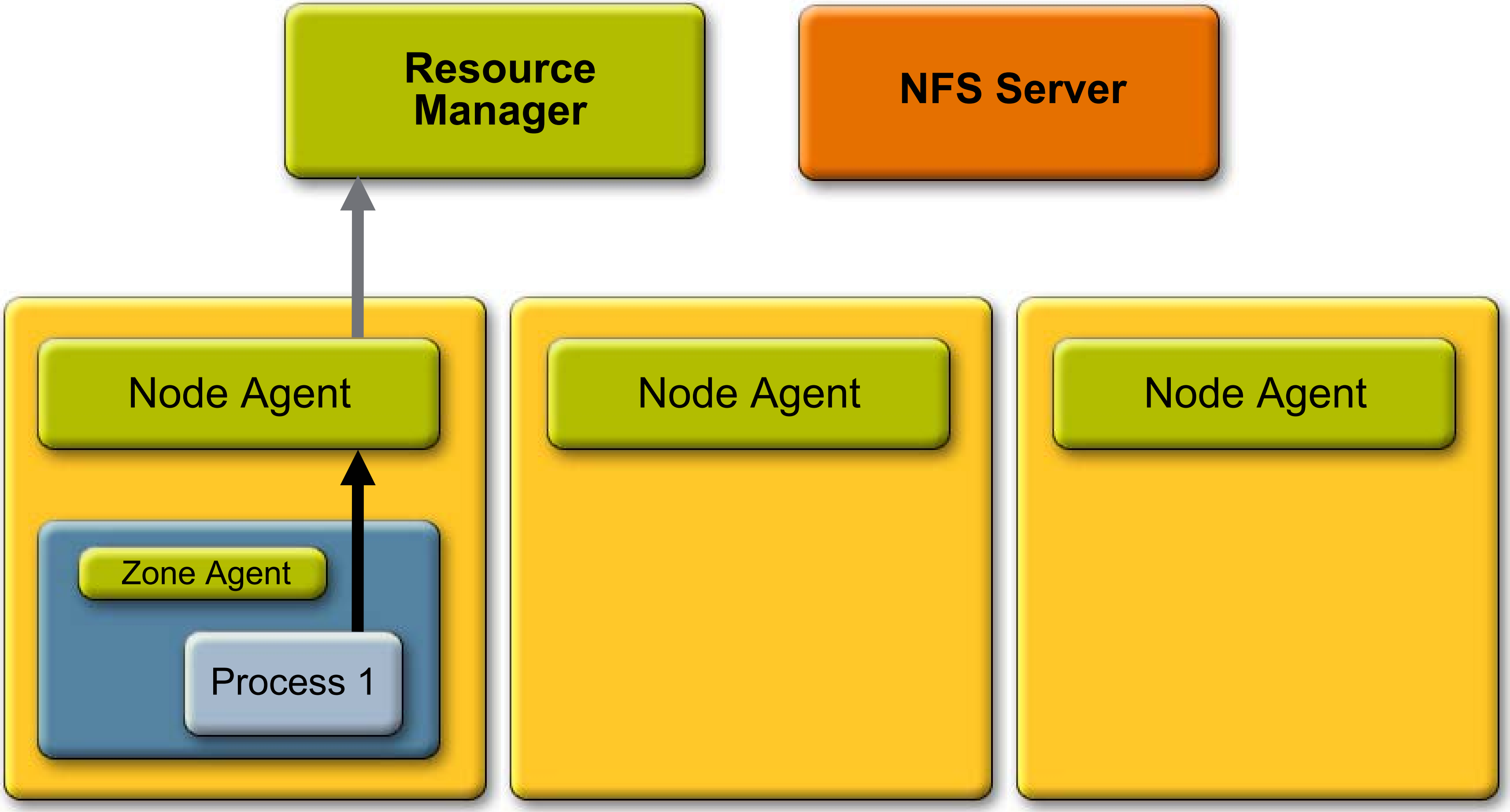
Behind the Scenes—Two System Calls

- Starting a process
- Creating a load balancer

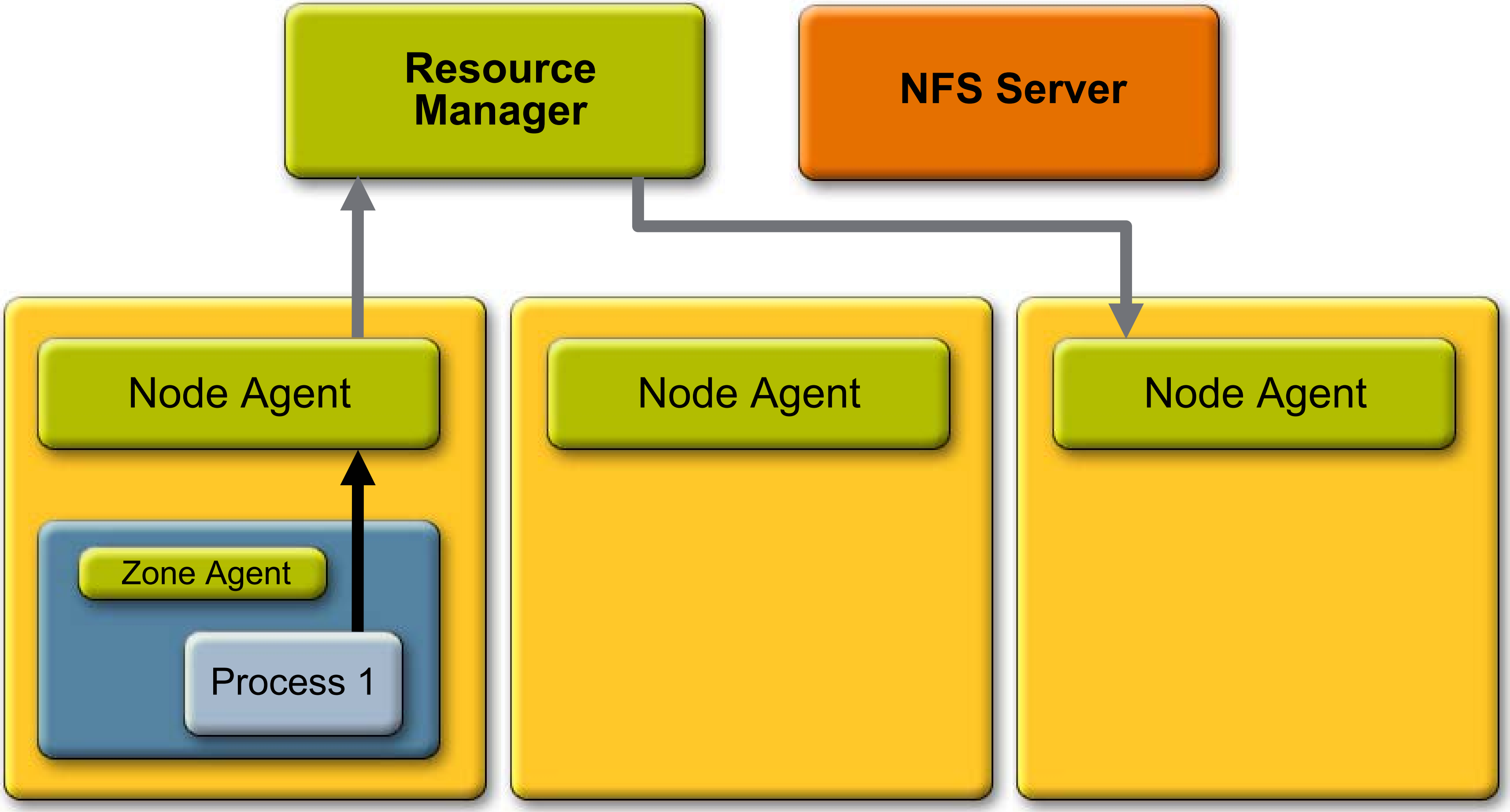
Start Process—System Call



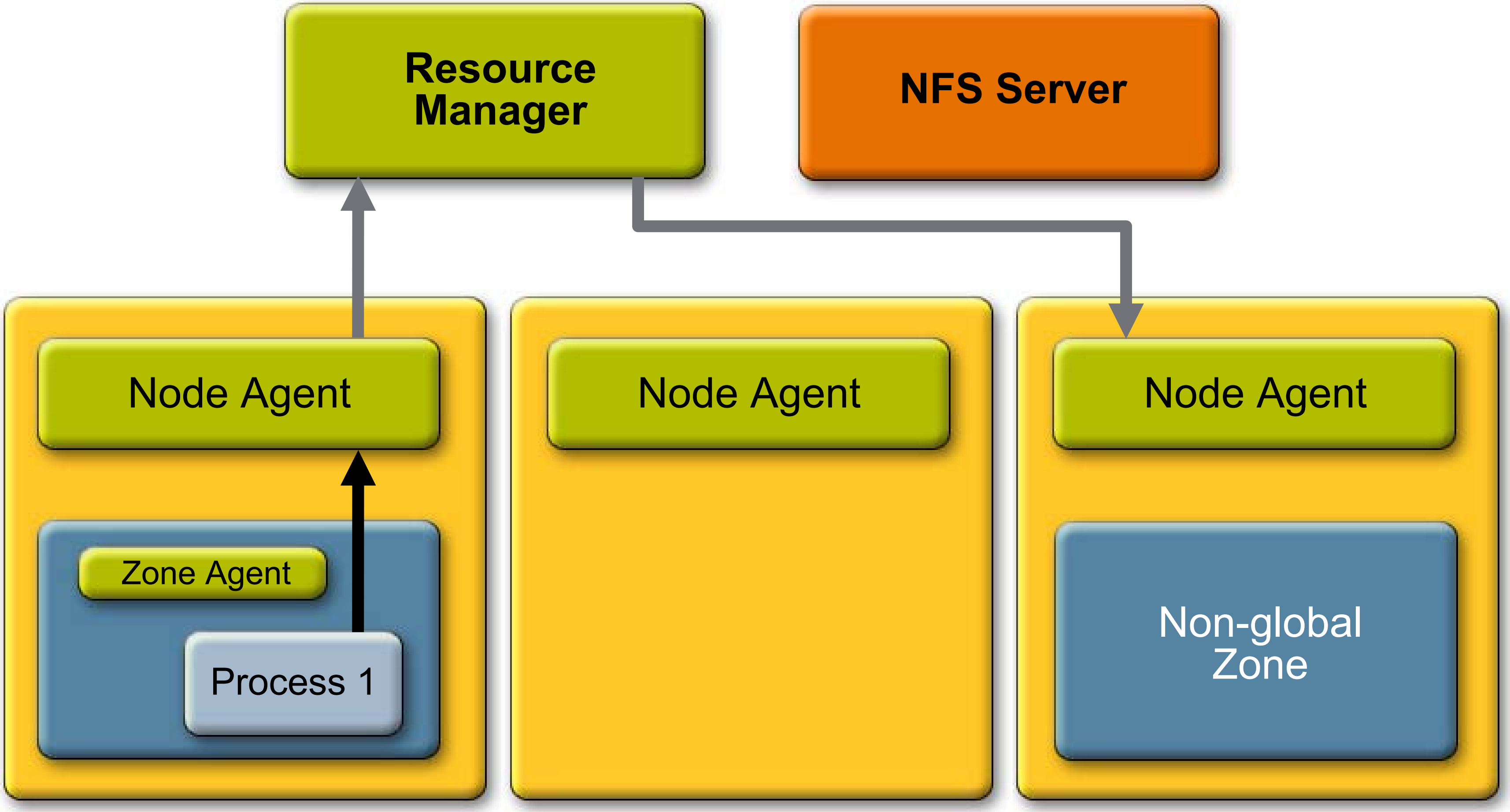
Start Internals—RM Request



Start Internals—Node Request



Start Internals—Boot Zone



Resource Manager

NFS Server

Node Agent

Zone Agent

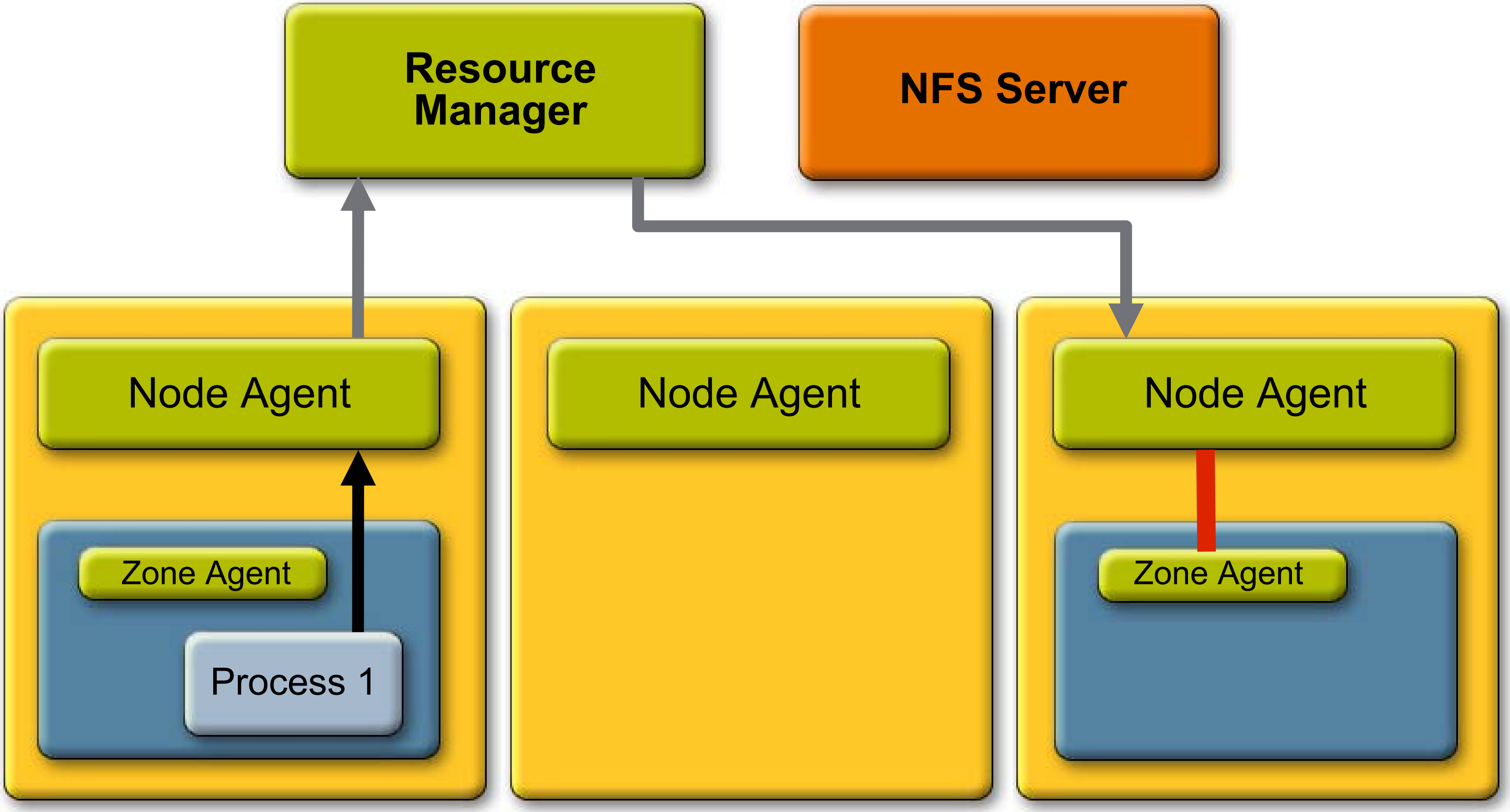
Process 1

Node Agent

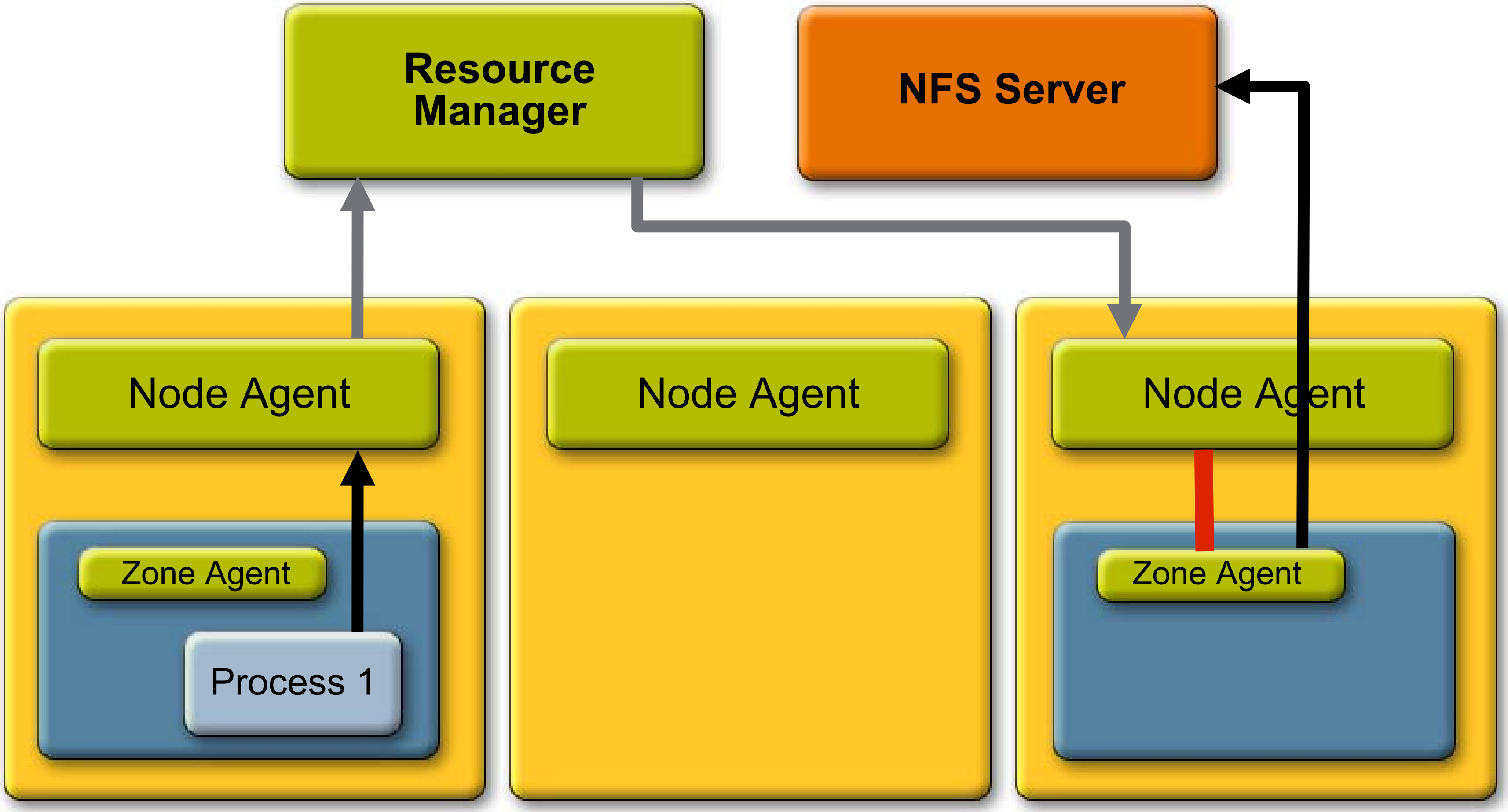
Node Agent

Non-global Zone

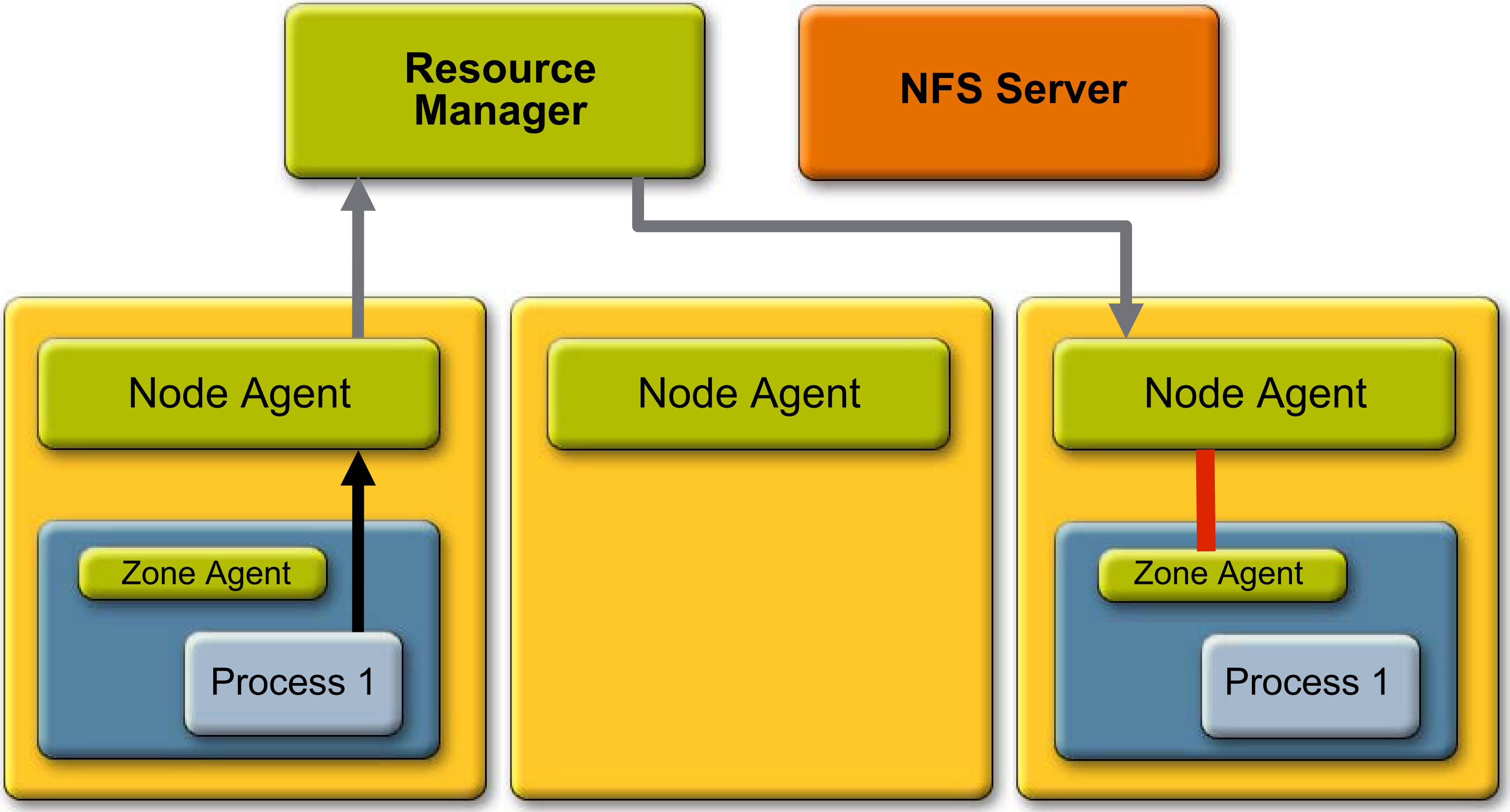
Start Internals—Run Zone Agent



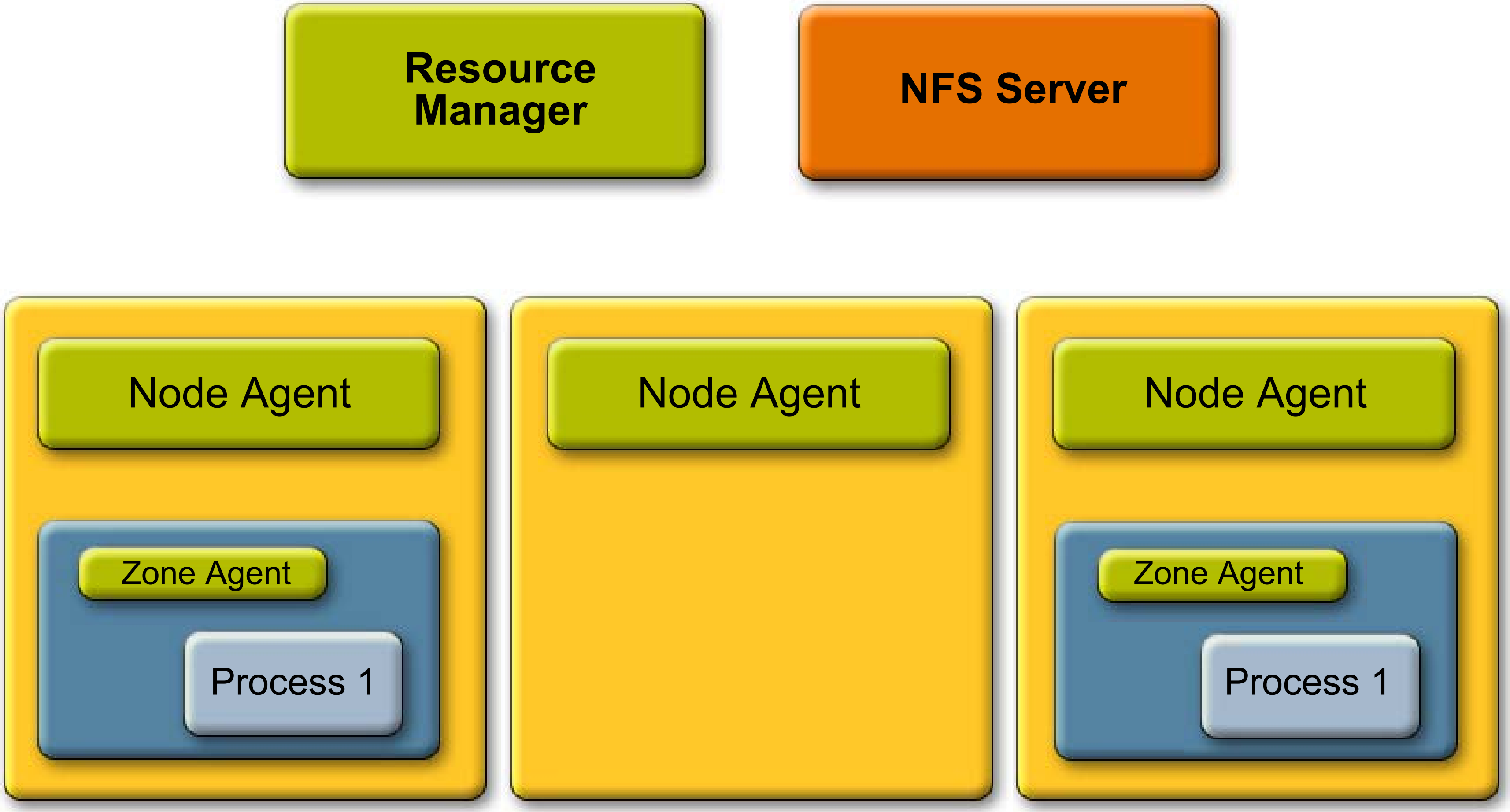
Start Internals—Run Zone Agent



Start Internals—Exec Process



Start Process—Done


Resource Manager

NFS Server

Node Agent

Zone Agent

Process 1

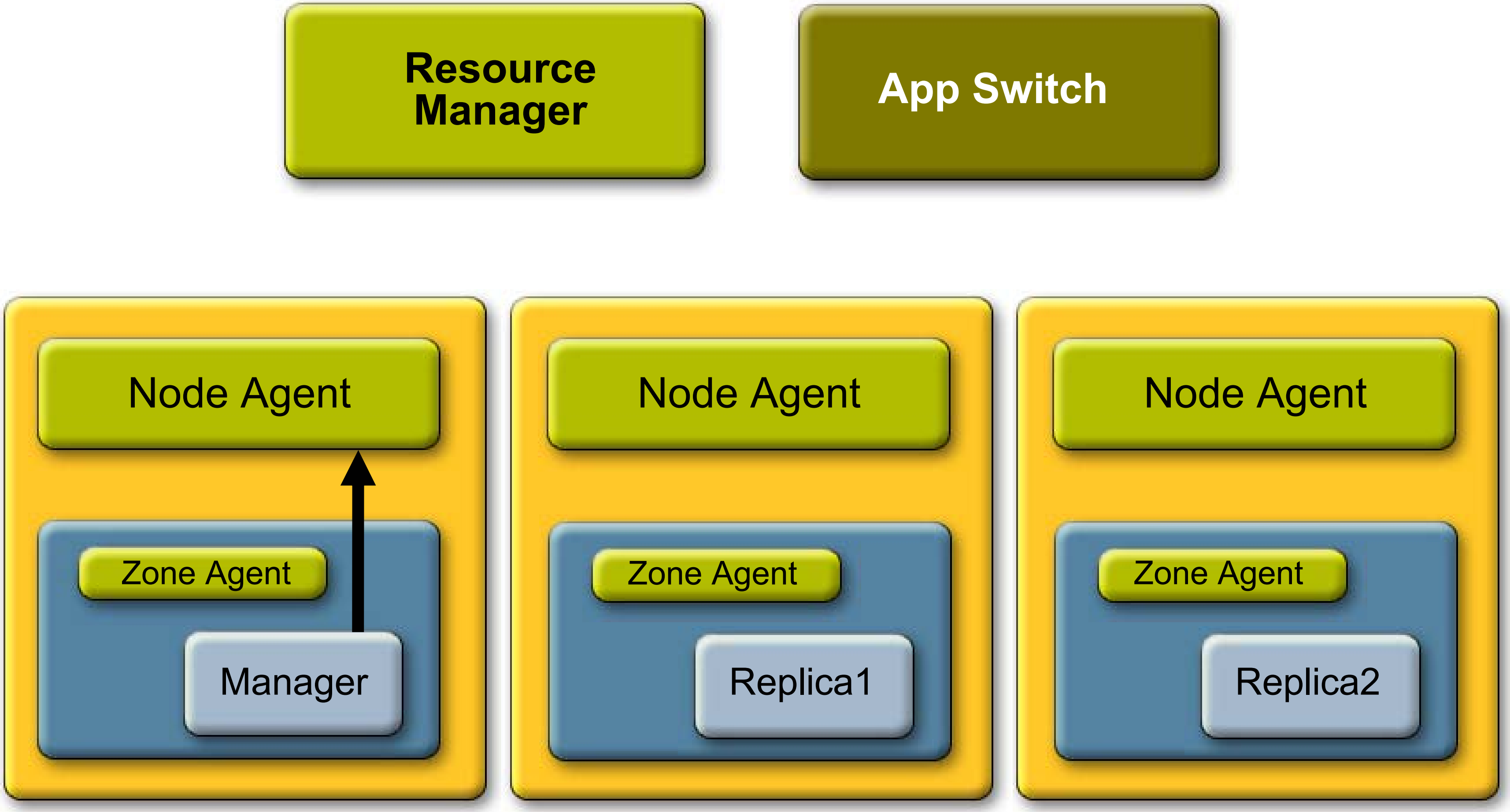
Node Agent

Node Agent

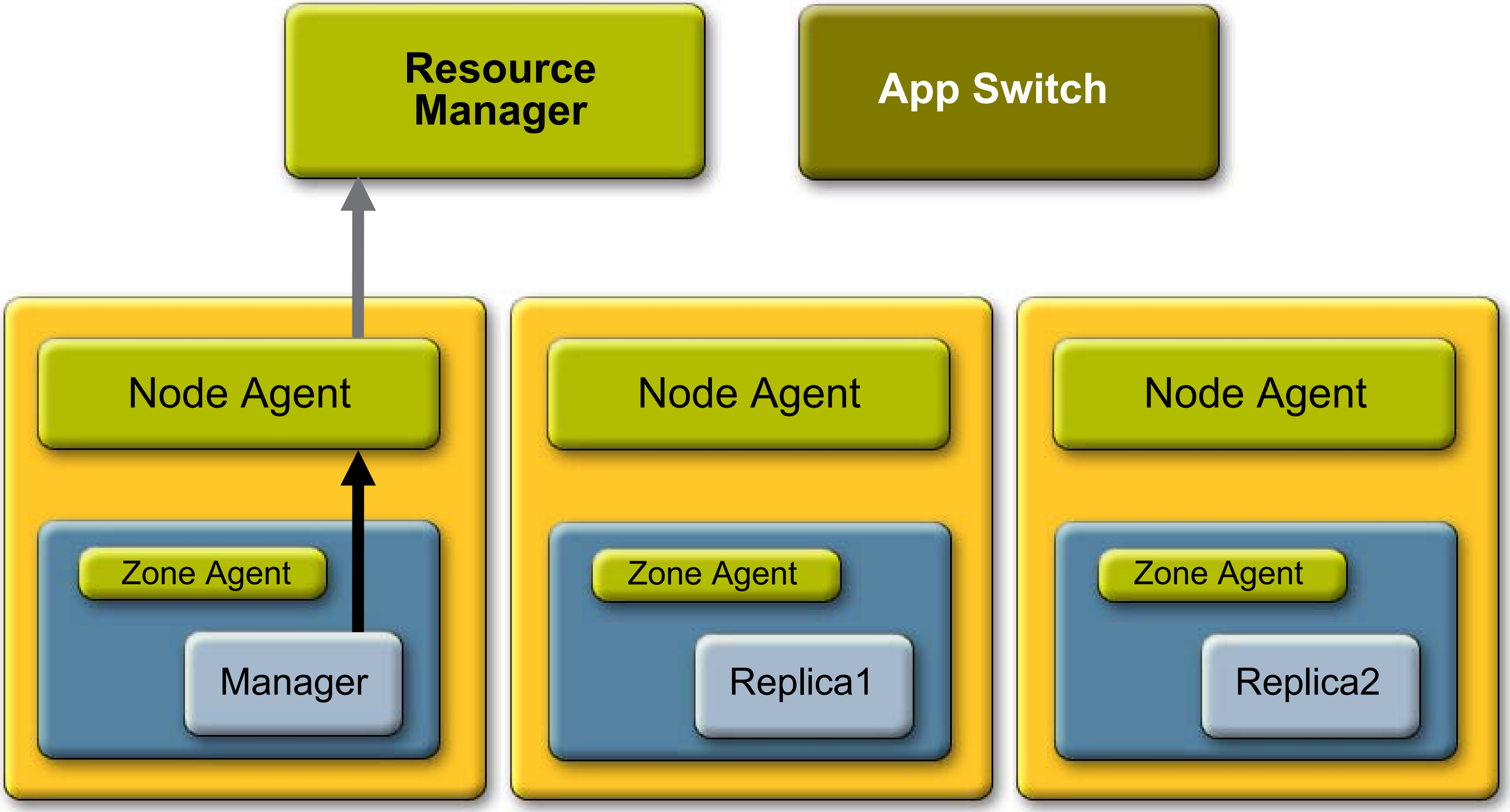
Zone Agent

Process 1

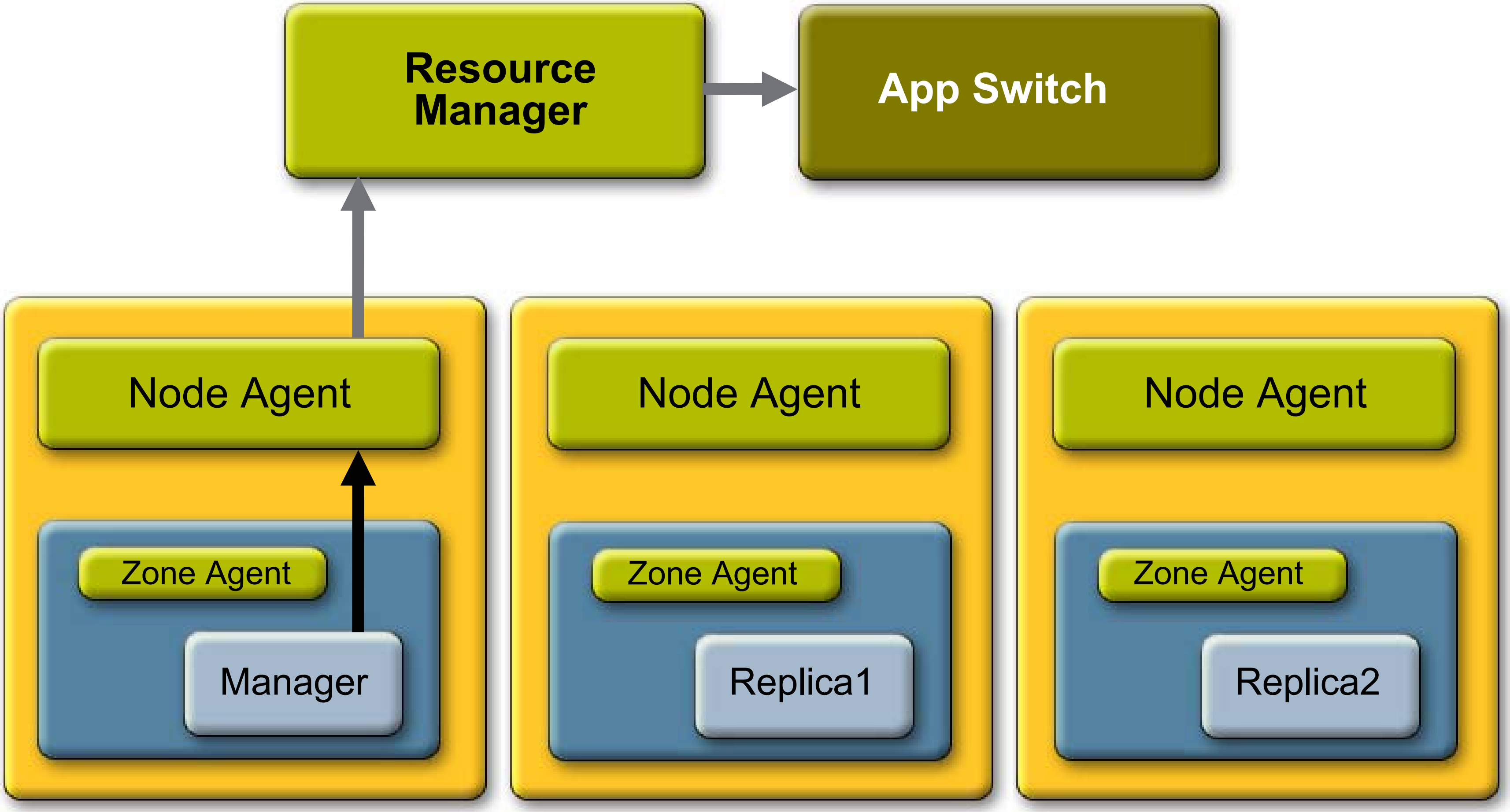
Create Load Balancer—System Call



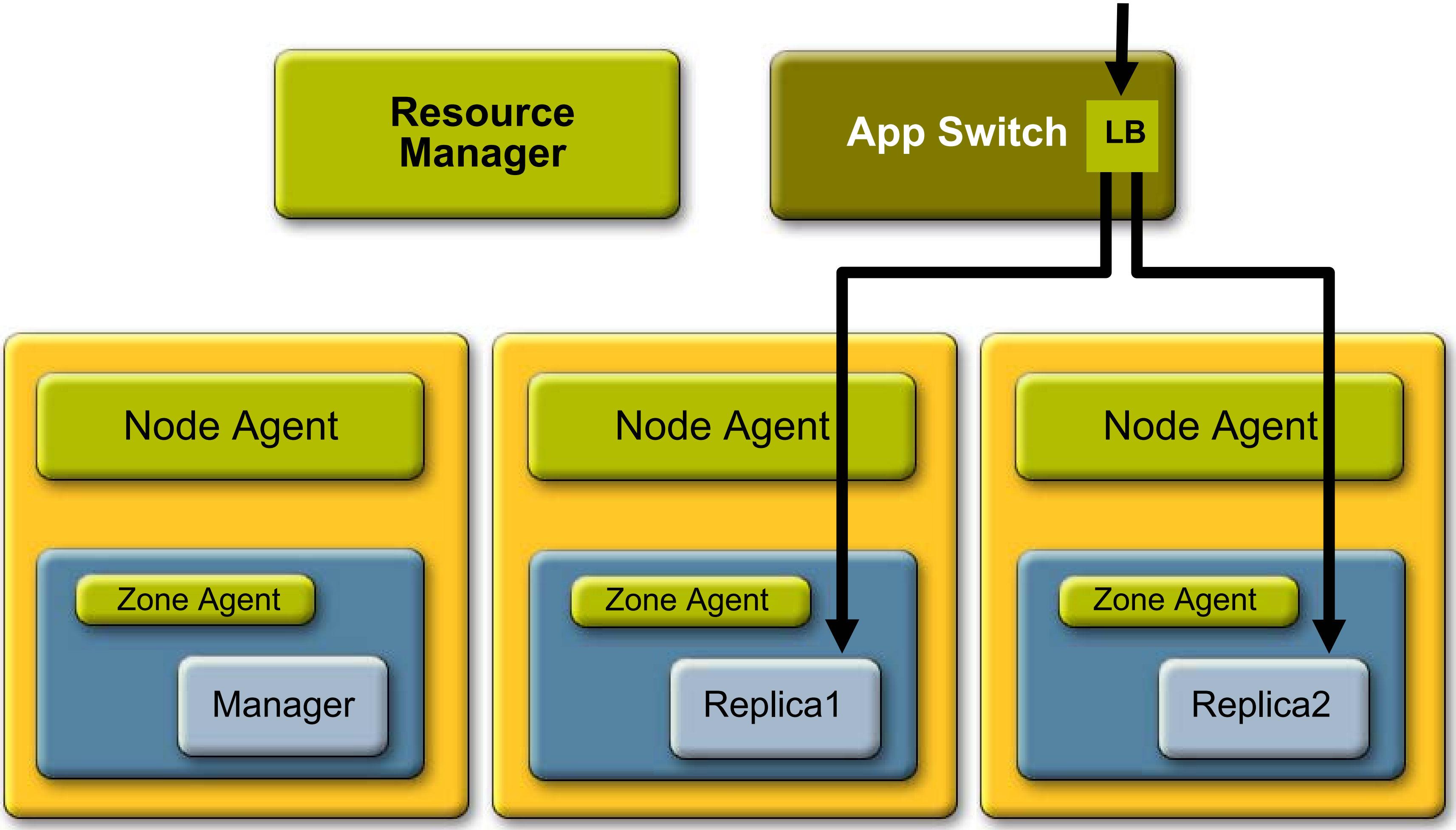
Create Internals—RM Request



Create Internals—App Switch CLI



Create Load Balancer—Done


Resource Manager

App Switch

LB

Node Agent

Zone Agent

Manager

Node Agent

Zone Agent

Replica1

Node Agent

Zone Agent

Replica2

Some Examples We've Run

- Automated deployment of LB'd Tomcats
- Distributed Conway's Game of Life (X, sockets)
- Master+workers with automated flexing (Jini™ network technology)
- Distributed ballistics simulation (Java RMI)
- Ruby on Rails demo (JRuby)
- JPetStore (Tomcat, Spring, Struts, HSQLDB)
- LB'd web service (XFire, Derby, Hibernate)

Summary

- Hosting platform for development and delivery of dynamically scalable Internet-based services
- Programmatically allocate, monitor, and control virtualized compute, storage, and networking resources
- Resources are exposed through high-level abstractions
- Presents a horizontally scaled pool of resources as a single system

For a Live Demo

Come to the Project Caroline pod (#981) in the Sun area of the Pavilion

Source: Earth Sciences and Image Analysis Laboratory, NASA Johnson Space Center.
Mission: ISS002, Roll: E, Frame: 6368, Feature: Caroline Island, http://eol.jsc.nasa.gov/

Q&A

http://research.sun.com/projects/caroline

Project Caroline: Platform As A Service, For Your Service, At Your Service

Bob Scheifler

Distinguished Engineer
Sun Microsystems, Inc.
http://research.sun.com/projects/caroline

TS-1991