Deployment Solution 7.1 SP1 Architecture and Dataflow

Thomas R Baird

Principle Technical Support Engineer

Agenda p1: Overview and Architecture

1. Overview: Background Information & Assumptions
2. Architecture: DS Plugins
3. Site Services and Components
4. PXE Services and Functions
5. PXE Communication with NS
6. About SBS Files
7. Ports, Protocols, Tables

Agenda p2 – Dataflows

1. Dataflows: General Task Execution
2. Console Updates sent to PXE
3. PXE Server Startup & Server Updates
4. BootWiz & Updating Preboot Configs
5. Image & Driver Replication
6. Initial Deployment process
7. Disk Imaging (all 4)

Overview: Assumptions & Background

- Unless otherwise specified, all portions of this presentation are applicable to Deployment Solution 7.1 SP1 built on the SMP 7.1 platform.
- We assume CMS is installed.
  - We will attempt to indicate other solution interactions where possible so that if you are doing component installs you can understand the difference.
  - Most of this is NOT component interactive, so the issue is moot most of the time.
- Commonly use acronyms/ abbreviations in this document:
  - DS (Deployment Solution)
  - SS (Site Server)
  - TS (Task Server)
  - PS (Package Server)
  - SMA (Symantec Management Agent)
  - NS or SMP (Notification Server or Symantec Management Platform)
  - CTA (Client Task Agent)

Overview: DS Dataflow and Architecture Discussion – What it is, and what it is not.

- DS is not a stand-alone product, as it was in the past.
- The NS SMP running on SQL is now the "engine" of DS
  - Task Server is now the main delivery mechanism for DS
  - Another delivery mechanism is the Symantec Management Agent via policies.
  - IIS is the main communication portal for client to server requests.
- This is NOT a discussion about NS, SMP, IIS, SQL.
  - All aspects of the supporting processes for DS will be discussed *only in context of how they are used for Deployment Solution*
  - The focus of this presentation is on what DS *adds to the SMP and CMS product*
- There is separate training available for SMP, Task, IIS, and SQL.

Overview: What does DS add to the SMP and CMS?

- Imaging
  - Site Server components including Ghost, RDeploy, and supporting utilities.
  - Several custom tasks to make this process easier for users
- PXE
  - SBS (Symantec Boot Services) aims to meet PXE needs, among other things (hence “boot services” vs simply PXE)
  - Includes automatic generation of WinPE and Linux preboot environments
- Advanced workstation configuration tools
  - DeployAnywhere and custom configuration tasks
- Custom tokens and token support
  - Though Tokens are a part of Task, Task was, essentially, created for DS, and the tokens in Task are almost exclusively used for DS tasks

NOTE: This document covers the architecture and process flows involved in these features, *not the features themselves.* Features and how to use them are covered in separate training.

Overview: What is a Plugin or Task Handler?

- A Plugin is simply an addition to the Symantec Management Agent (SMA) to expand its functionality.
  - Generally, this includes DLL's for expanded function interpretation, and supporting files for those functions.
- The SMA and "core" functionality of Notification Server is free
- Most solutions have a plugin of some sort.
  - These are what you pay for based on what you need.
  - Examples include software delivery, inventory, patch, and deployment.
- Task Handlers are a component of a plugin, but are often considered the same thing.
  - Task Handlers are the portion of a plugin meant to interpret Task data.
    - Contrasted with plugin information meant to interpret policies or perform functions unrelated to either one.
  - If a plugin ONLY has task handlers, then there really is no difference.
  - Sometimes, it's easier to refer to the plugin as a task handler to be sure the communication is clear.

Architecture: DS Plugins

There are, essentially, 3 policies that are used to deploy Deployment "plugins":

- Deployment Automation Folders
  - This isn't really a plugin. It's a package included in the same area and is often confused as one, but is not. These are not discussed in this presentation.
- Deployment Plugin for Windows
  - This contains Task Handlers for Client-Side functionality.
  - The tasks these DLL's support are all listed under Deployment when adding a new task.
  - The plugin also includes some files necessary for running those tasks.
- Deployment Site Server Components
  - This plugin contains Task Handlers for server-side functionality
  - Includes DLL's for policies as well.
  - Includes several other tools including PXE, creating preboot configurations, and a datastore for images.

Architecture: More on DS Site Server Components

When you install the DS Site Server components, you get far more than simply a DS Site Server or PXE server that DS 6.9 installed. The components include:

- Bootwiz (including custom drivers)
- Ghost (and utilities) and Rdeploy
- DeployAnywhere Driver Database
- Misc Tools, including the DS Import Utility
- Image Store (where all images are replicated to or captured to)
- PXE, including 4 services:
  - _netboot_Server
  - _netboot_NSInterface
  - _netboot_NSiSignal
  - _netboot_MTFTP
- WAIK (Windows Automation Installer Kit) used to build Preboot WIM's
- Deployment Share (contains all of this)
- Task Handler DLL's (both for the site server and for the agent DLL's)

Architecture: Non-DS Site Server Components

Prior to making DS function, you must have other things in place. DS 6.9 was it's own product, but since 7.0, it is an add-in for something larger: the Symantec Management Platform (SMP). These are some parts of what DS interacts with:

- Task Server (TS) – this is the site server component developed specifically for Deployments. It gives NS the only near-real-time functionality it has. All Jobs and Tasks (not policies) use Task Server, and DS uses Tasks or Jobs for nearly everything it does.
- Notification Server (NS) – this is the new "engine" for DS as compared to DS 6.9. the NS core, now called the SMP includes all rules, security, hooks to the database, etc., literally being the "engine" for everything DS.
- Package Servers (PS) – this site-server functionality is what replicates everytyhing from the NS or other locations to other site servers. DS uses these for replicating many things, including images.

Architecture: PXE Services and their role/purpose

- There are 4 PXE services:
  - **_netboot_NSInterface** – 'Interface'
    - reads SBS files from the SBS folder / file system and gives information in those files to 'Server'.
      - Offloads this process from 'Server' to allow Server to be more responsive to PXE requests.
    - duplicates individual computer SBS file instructions to all other SBS servers.
      - NOTE: This replication only includes active jobs, not all computers, to allow a computer being sent to PXE to be able to boot to *any pxe server* and still have the information it needs.
      - SBS files are, by default, only sent to the PXE server to which a client is expected to reboot. Replication is precautionary only.
  - **_netboot_Server** – 'Server'
    - is the main working process handling all client PXE communication / requests
    - listens for information from 'Interface' and stores information in tables in RAM
      - Basically, this includes the basic parameters of execution or 'Rules', what clients it manages, and what images it has..
    - if restarted, tells 'Signal' to run its process again.
  - **_netboot_NSiSignal** – 'Signal'
    - runs an EXE called "PXEStartupInfo.exe" on request or automatically when it starts.
      - PXEStartupInfo calls a webservice (DeploymentWebService.asmx), and creates the first SBS files needed for 'Server', and then tells 'Interface' to read them via a Windows event.
    - listens for requests from 'Server' to re-run the EXE whenever 'Server' restarts.
    - determines if the server has been registered yet with the NS, and if not, registers it (see TECH127551)
  - **_netboot_MTFTP** – 'TFTP'
    - file transfer service for PXE, sending all files to the client pre-automation
      - including boot.0, and the WIM used for WinPE (or the Linux preboot environment/LinuxPE).

Architecture: PXE Services Dependencies & Start Order

- By default, all services are set to “Manual” to avoid conflict with other PXE services that may be on the network.
  - ‘Interface’ is started by default for the replication process.
- There are no significant dependencies.
  - There is only one built-in dependency: ‘TFTP’ stops whenever ‘Server’ stops.
  - If all 4 services are set to “Automatic”, this is generally sufficient for everything to work correctly.
  - IF there is a significant delay in one service starting before or after some others, then dependencies start to show. (eg. Interface starting too late causes problems.)
- The following is a recommended start order and dependency list as a “best practice” but not required. Reasons are included.

1. **_netboot_NSInterface** – This service should always be running. Since it reads files created by Signal (along with all other SBS files), it should be on before Signal.
2. **_netboot_Server** – Server needs to start before Signal creates its files in order to accept the information from Interface. To ensure this, if Server is NOT on before Signal, it will call Signal again.
3. **_netboot_NSiSignal** – Signal sends information to Server, in a round-about way, and thus Server really should start before it does. This avoids running the EXE two times.
4. **_netboot_MTFTP** – This service literally doesn’t matter what order it’s in. So we generally put it last, since it will do nothing unless Server is fully up and running.

Architecture: PXE Communication with NS

PXE Server needs three things to be able to function:

- Configuration Settings, or the 'Rules' of engagement.
  - Initially, these are read from the initialPXEConfigPath.txt file in the SBS folder
  - PXEStartupInfo.exe requests an update from the NS for this file immediately after Server starts. It creates a file called *initialPXEConfigPath.sbs* to update Server. This completely overwrites the one read in the previous step.
  - Further updates to this information (i.e. changes to the console settings) are delivered via SBS file updates.
- List of clients it should manage and what to do with each.
  - PXEStartupInfo.exe requests an update from the NS for this file immediately after Server starts. . It creates a file called nnnnnnnnnn-client.sbs to update Server.
    - A new addition to this list in 7.1 SP1 is PredefinedComputers.SBS. Client.SBS only lists managed computers, and predefined computers are not yet managed. This file adds the predefined computes so PXE can process them properly as well.
  - Further updates to this information (i.e. changes to the console settings) are delivered via SBS file updates.
- List of images it can distribute to clients.
  - PXEStartupInfo.exe requests an update from the NS for this file immediately after Server starts. . It creates a file called nnnnnnnnnn-image.sbs to update Server.
  - Further updates to this information (i.e. changes to the console settings) are delivered via SBS file updates.
    - **Note:** 'nnnnnnnnnn' in the examples above is a random 10-digit numeric preceding the rest of the file name
    - **Note:** "SBS file updates" file-names are a GUID.sbs file, which is discussed more on the next slide

Architecture: SBS files – What are they?

- SBS files are information files for the PXE Server Service to know how to manage PXE. The information is stored in RAM in the Server Service
- Three (or four) are created every time PXE Services start.
  - PXEStartupInfo.exe, called by the Signal service, creates the 3-4 initial files for PXE Server mentioned on the previous slide (examples on the following slide) to populate PXE Server with the current known state of the database/system.
- Every time a policy change is made, or a new computer is added, an update file is sent to the PXE Server with the changes.
  - The update file is named <GUID>.SBS (eg: 77aa417a-c9d7-4154-8f79-97ef95ead2bc.sbs)
  - The file may contain any information relevant and always have the same 'name'.
  - The update only contains one piece of information at a time.
    - If there are 50 updates, there will be 50 files sent. (eg. 50 predefined computers added)
  - *These come to the SBS server in a task called "UpdateSBSInfo." This task is often seen in agent logs and task histories, and so should be remembered. It will be seen a few times in the dataflow diagrams as well.*
- SBS files are initially created in the SBS folder, read by Interface, handed to Server, and then moved to SBSStore. This happens in a fraction of a second.
- SBSStore folder is a historical folder, showing past/used activity, and enabling troubleshooting. There is no schedule to delete these.
- If SBS files are found in the SBS folder, it indicates services are stopped.

Architecture: SBS Samples

2139425051-client.sbs - Notepad

File Edit Format View Help

```
Status=UpdateComputerStatus
UUID={0A293DC4-4C6E-40C3-8A82-111A1FAF12A6}
MAC=00155D016F2B
UUID={3C0F36FA-75EB-4360-9E3B-FB62C0D3E151}
MAC=00155D016F25
UUID={B6E3CDC0-B6E9-4266-807B-5F784B15F221}
MAC=00155D61F009
UUID={97137FC9-6B37-4B45-9BF0-B488A7FAD6FD}
MAC=00155D016F20
UUID={3C6CD2E7-408B-421C-84CA-AD53330D2932}
MAC=00155D016F2A
UUID={14EB9C00-B4B3-4311-9A7C-1C7A0B0FFADA}
MAC=00155D016F21
SBSIPList=192.168.0.23,192.168.0.78
```

Client.sbs contains mac and UUID for all known computers assigned to that site/pxe server. It includes the SBS IP List.

initialPXEConfigPath.sbs - Notepad

File Edit Format View Help

```
Status=Configuration
Server_Log_Enabled=True
Mtftp_Log_Enabled=True
Helper_Log_Enabled=True
iFace_NS_Log_Enabled=True
ParsePxe_Log_Enabled=True
PxeOpts_Log_Enabled=True
FilterIP_Log_Enabled=True
FilterMAC_Log_Enabled=True
FilterUUID_Log_Enabled=True
Server_Ukn_Enabled=False
Server_Listeners="192.168.0.78"
Server_EnabledParsers="SbsParsePxe"
Mtftp_MTUSize=1456
Mtftp_Root_Folder=C:\Program Files\Altiris\Alti
Mtftp_Listeners="192.168.0.78"
Mtftp_MaxTransfers=200
EventTimeOut=10
Server_Ukn_Image=WinPEx86TRB
Server_Ukn_Arch=x86
```

InitialPXEConfigPath replaces the original completely, and includes logging options, how to respond to unknown computers, and IP addresses to bind to.

2141048499-image.sbs - Notepad

File Edit Format View Help

```
Status=UpdateImage
Image=BStrap
Arch=x86
OptID=99
Image=BStrap
Arch=x64
OptID=99
Image=WinPEx86TRB
Arch=x86
OptID=
```

Image.sbs includes the 2 default configurations, AND your custom configurations. This is used to build the PXE menu.

PredefinedComputers is new in 7.1 SP1. Created on startup, it lists non-managed computers imported for future management purposes. Image="" indicates "boot to production". Otherwise, it will say what PXE image to use.

*** *The **SBSIPLIST** (in Client.sbs, predefined.sbs, and the update files if the update is about clients (eg. below)) is used to tell Interface what servers to replicate these files to. This replication is applicable to all computer information, as seen in these examples.*

3131561-predefinedComputer.sbs - Notepad

File Edit Format View Help

```
Status=UpdateComputerStatus
UUID={a2c949d9-b1e3-42cd-9586-75a087d379cf}
MAC=00155D61F007
SerialNum=
Image=""
SBSIPList=192.168.0.23,192.168.0.78
```

The update files – that is, files sent after startup, always have a long GUID for he file-name. They will contain information on the image or changes to the SBS settings. If they contain information about computers, it will include the SBS IP List.

77aa417a-c9d7-4154-8f79-97ef95ead2bc.sbs - Notepad

File Edit Format View Help

```
Status=UpdateComputerStatus
MAC=00155D61F007
UUID={}
SerialNum=
Image=""
SBSIPList=192.168.0.23,192.168.0.78
```

Architecture: Ports and Protocols

NOTE: since moving to the NS platform, most of the ports are default NS ports. Any exceptions are included here.

- PXE, including 4 services:
  - _netboot_Server
    - Binds on UDP 67 (BootPS) to listen for DHCP discover and request packets .
      - Unless on the same server as DHCP – not a recommended configuration.
    - Binds on UDP 4011 for unicast communication.
    - Talks to Signal on port 806 – will try over and over to establish this connection. (internal only)
    - Additional ports (internal only): 802, 804, and 4 additional ports that start at 2000 and increase sequentially every restart.
  - _netboot_NSInterface
    - Listens on TCP Port 490 for internal processes *and* externally to other interface services when they need to forward client job/task information.
  - _netboot_NSiSignal
    - Listens on TCP Port 806 for Server to bind to it and tell it to run again if Server restarts (Internal only).
  - _netboot_MTFTP
    - All TFTP unicast requests occur on UDP 69
    - Multicast requests are handled on UDP 1759
    - Multicast support was removed as of DS 7.1 SP1
- Remote Trace: uses port 415 to listen for all DS logging (all) of all clients from the server.
  - C:\Program Files\Altiris\Notification Server\NSCap\bin\Win32\X86\Deployment\Tools

NOTE: All ports above that are listed as "internal only" indicates that no router configurations should need to be made aware of them.

Architecture: Tables of Note

There are several other tables we use and share with other products, but here are a few worth noting.

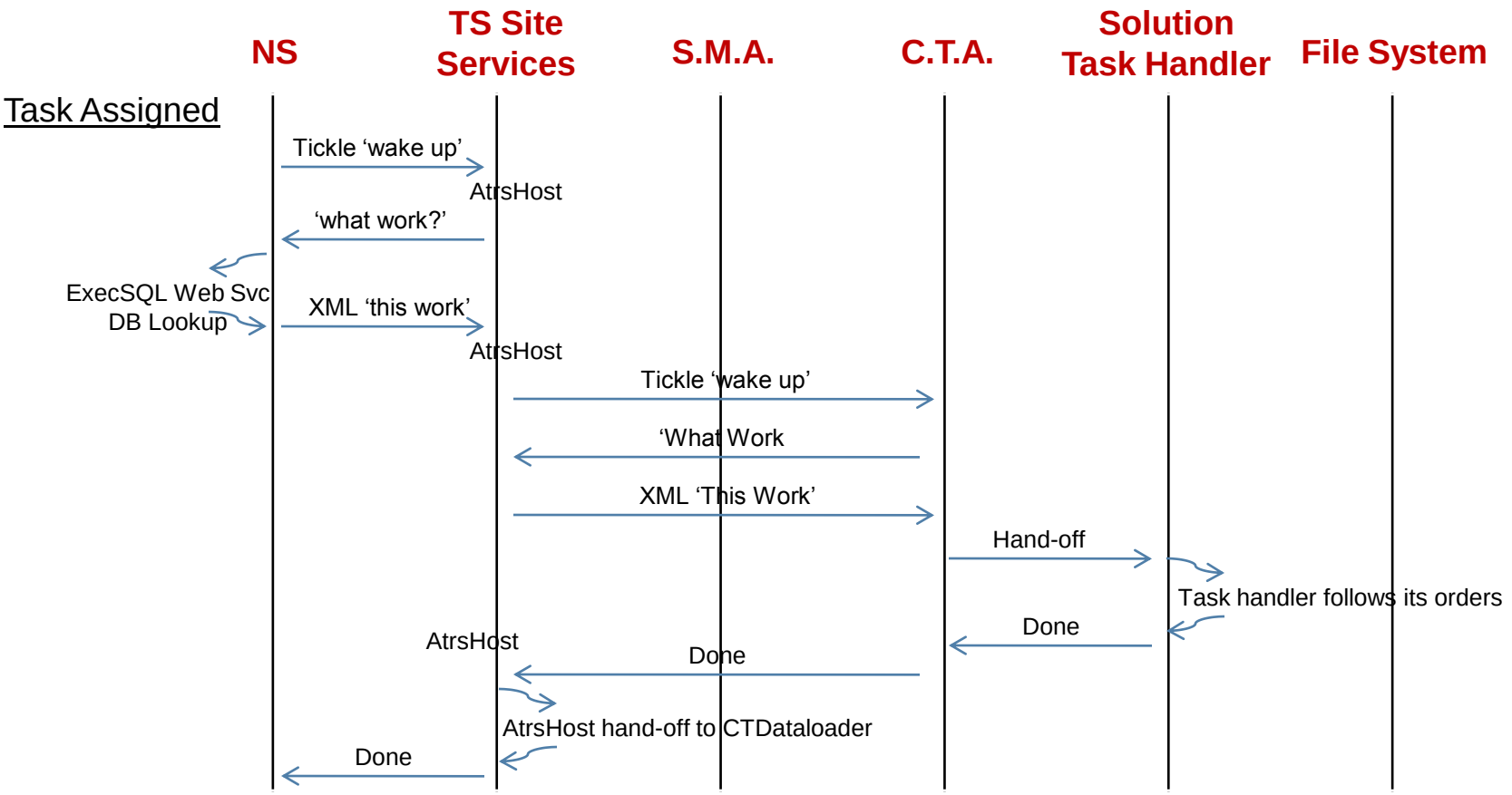
- SBS_ServerList
  - Guid, IP, enabled – static list of known SBS servers. We used to never update this, but we've been actively working on that.
  - This is one of the more important tables we troubleshoot, since if the information is incorrect in here, several other problems can manifest themselves.
- PXE_Image_List
  - The list sent to our PXE servers of what images are valid and should be shown to a PXE booting system.
- PXE_Image_Status
  - Pull from this if the image has been built or not
- Automation_image_configuration
  - All info for automation images, including if it has been created and if we want to do anything to it now (eg. Recreate)

Dataflow Diagrams

The following slides will demonstrate via Dataflow Diagrams how some of our processes work.

- Though some pieces behind-the-scenes will be skipped, enough should be presented to assist in troubleshooting.
- Things like SQL dataflow will not be presented.
- Dataflow of the NS services will not be presented, even where they overlap with DS. It will simply be “NS” and possibly “write to DB” but nothing more specific than that.

Dataflow: Task Distribution to a client

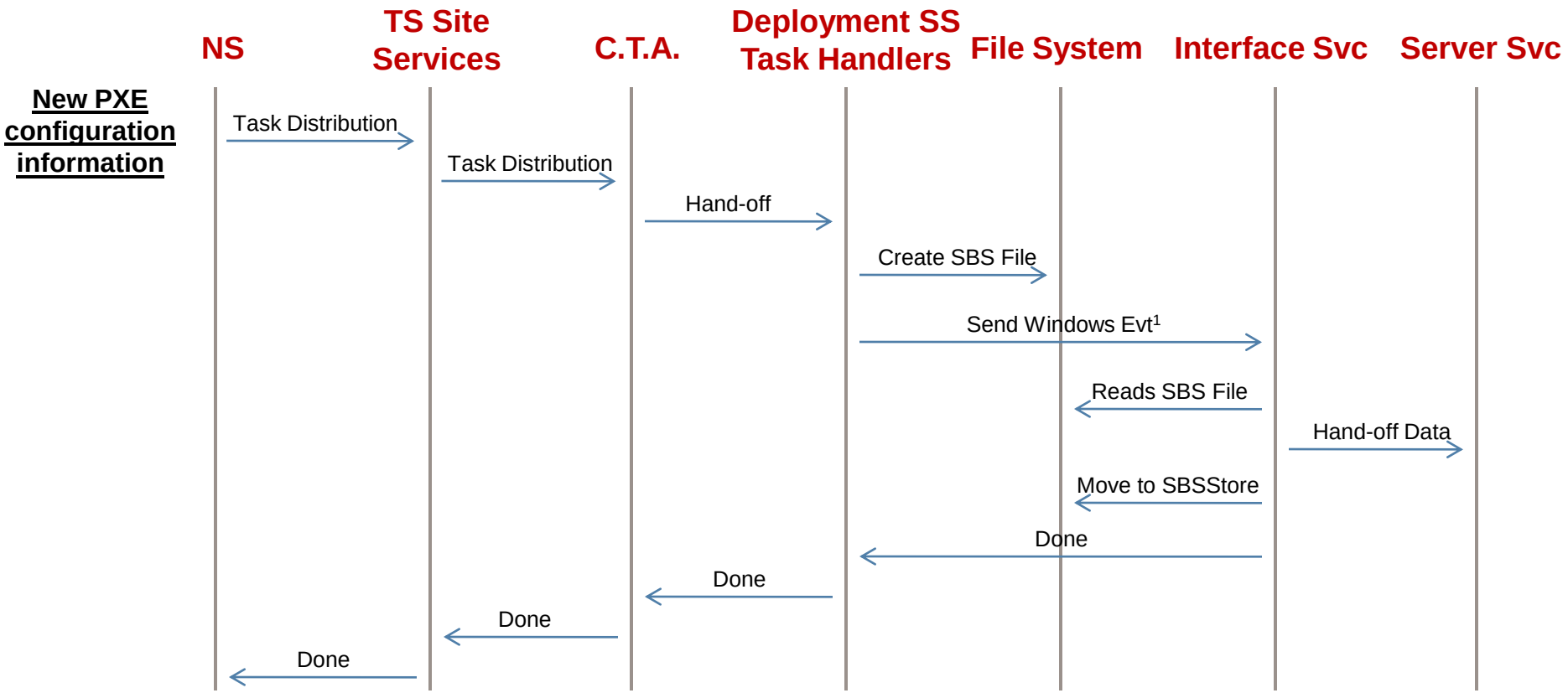


Though not specifically Deployment Solution related, this process is utilized by all Deployment Solution tasks. Therefore, it is documented here for troubleshooting purposes.

In succeeding slides, we will use "Task Distribution" to refer to either all, or a portion of this process.

1. We tell the Object Host Service (ATRSHost) what to do (tickle on port 50122?).
2. He tells the Client Task Agent what to do (tickle on port 50124).
3. Client Task Agent (CTA) looks at the XML, determines the correct Task Handler and hands-off the XML instructions.
4. We return the statuses up the chain.
   - The only break in the flow back is that ATRSHost doesn't report up the status – the Client Task Dataloader (CTDataloader) does this.

Dataflow: How a Console Update gets to the PXE Server Service



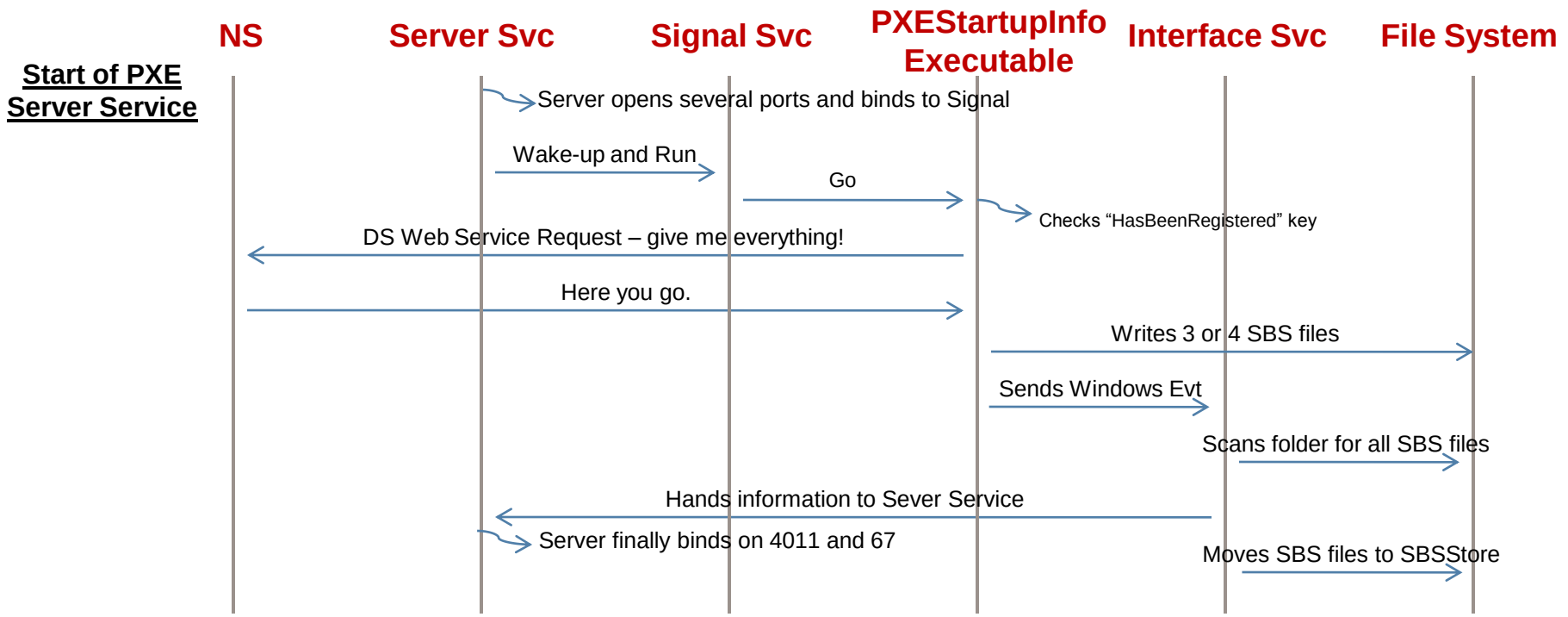
[1]*More on this in the next slide*

After a change is made in the NS console (New PXE Info), we use TASK to update the PXEServer service (netboot_Server)

There are a few items to pay attention to here:

- This is nearly 100% a task-based process.
- Though there is reporting that the SBS file was delivered, *there is no reporting that the file was read successfully to the server service.*
- To get information on if the Server Service read the file, the Server logs would have to be analyzed. However, the fact that the file actually moved to the SBSStore folder is a fairly strong indication that the information was received.

Dataflow: PXE Server Startup and Initial Information Population



Remember, PXE requires the data from all 3 or 4 files (3 files if no predefined computers have recently been imported). It needs the client information, image information, and it's own "current" startup information.

Only after getting ALL the information it needs will it finally bind on 4011 and 67, thus listening for PXE requests. Until then, it's not listening.

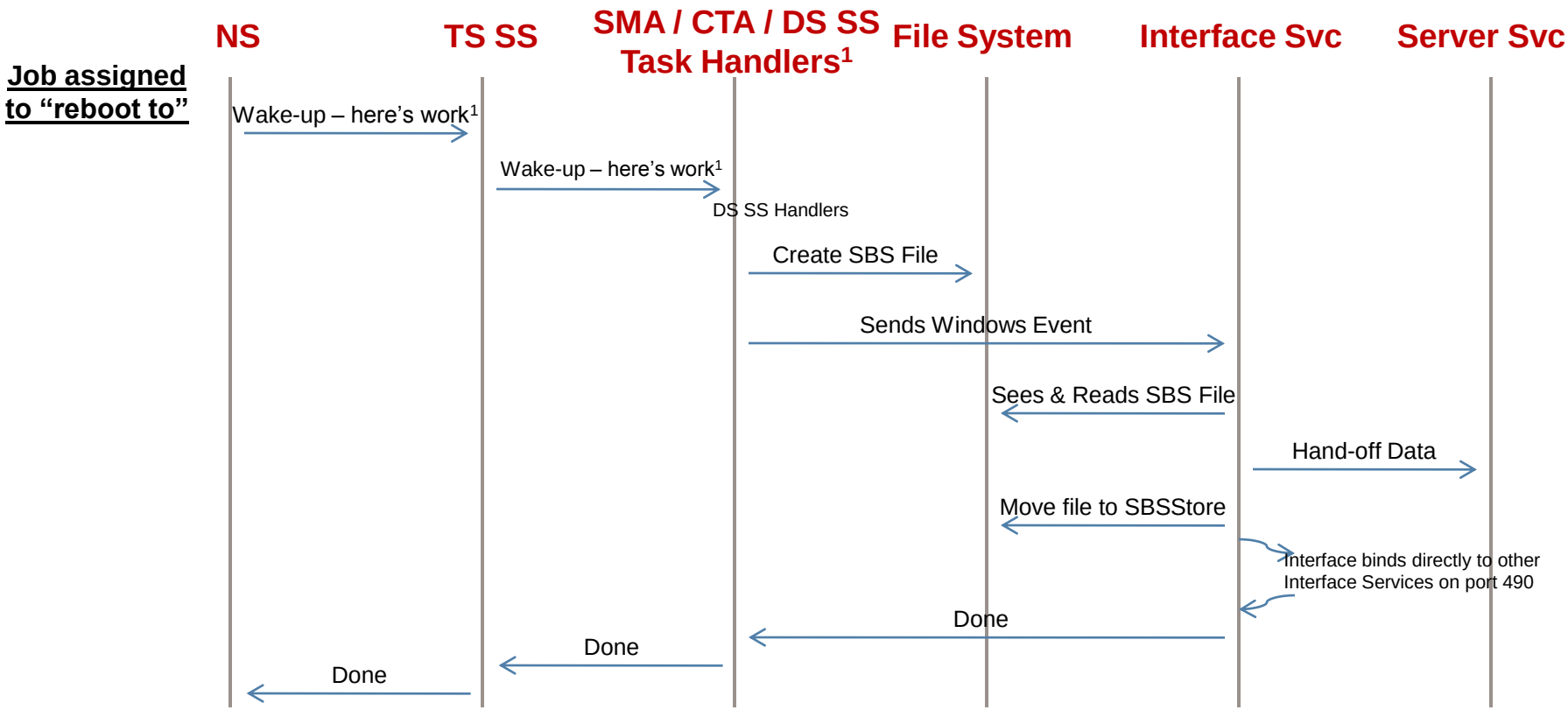
NOTE: If Interface is not running, the data will not be handed to Server, and Server will never function as a PXE server.

NOTE: Interface does 2 things relative to SBS files. First, it looks in its folder for any SBS files when it starts (by default, the SBS folder). Then, it waits for a Windows event to tell it to scan the folder again. Signal and any task (eg UpdateSBSInfo) sends a Windows event to trigger this search. Until it receives an event, it does nothing at all, saving resources. *When it receives an event, it will pull any and all SBS files and their information from that folder*, not just the one recently created. Same thing as when it starts – it scans the folder for ALL SBS files, reads them, & hands the data to Server.

NOTE: Other than log files, because this is not a server-initiated process, there is no record of this happening on the server. Thus, there is no server-based status on if the PXE Services are functional on any PXE server at this time.

NOTE: The check for "HasBeenRegistered" does nothing MOST of the time. Only when the value is 0, when it will change the value to 1 and popluate a table in NS to indicate that we now have a valid PXE server.

Dataflow: PXE Server Updates

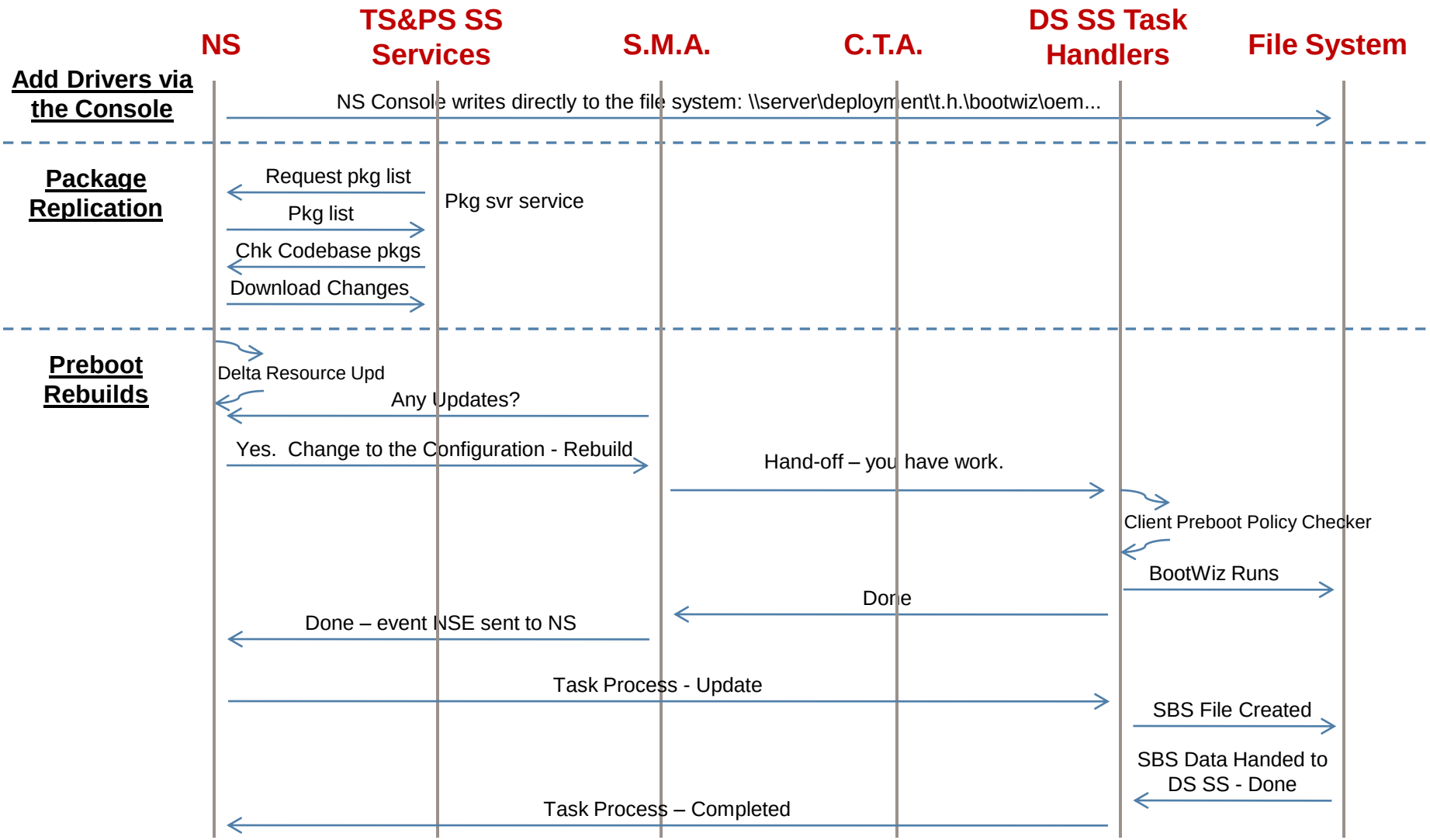


[1] The communication between the Symantec Management Agent, Client Task Agent and DS Site Server handlers is covered in another slide. So is the "wake-up, here's work" process of notifying a task server there's work. See slide "How a console update gets to a PXE server"

It is important to note that an update to a computer-related set of information always includes a list of PXE servers to which Interface should replicate to. Just like the Client SBS file. These files are all replicated to all PXE servers, just in case the client system somehow boots to another PXE server.

However, the process of replication can be slow, so it should not be counted on. It takes about 36 seconds *each* for those files to arrive on the remote PXE server (due to some scheduling within the exe's) and for something like a mass roll-out of 50 systems, by the time #50 arrives, it's almost 30 minutes later, and nearly worthless. The first few work though.

Dataflow: BootWiz – adding drivers to Automation Configurations



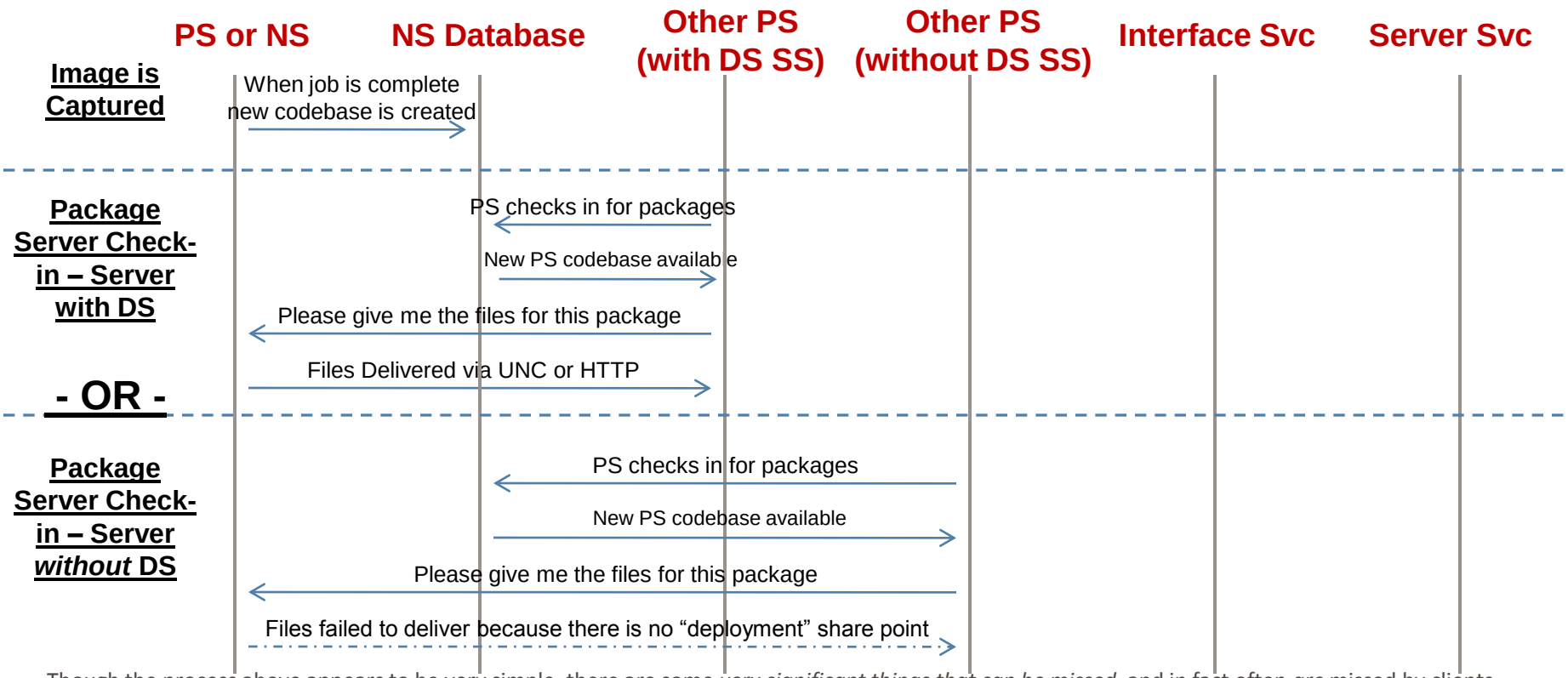
Dataflow: BootWiz – adding drivers to Automation Configurations -- Notes --

- It is critical to note that this is a 2-Step process – *NOTE that this is the same process for new configs.*
  - each step has a two parts: a manual part where the administrator does something, and an automatic part that fires on schedule(s).
  - Unfortunately, there are often no automatic triggers between the steps.
  - Some

1. Drivers are added to BootWiz.
   - In the console, the administrator adds the drivers
   - This copies directly to the file system
   - **On schedule**, package replication takes the drivers to the site servers (can be triggered manually) (not necessary on the NS Site – it's ready immediately)
2. Preboot Configurations are set to be rebuilt.
   - In the console, the administrator selects a preboot configuration and selects "rebuild" .
   - **On schedule**, Delta Resource Membership is updated to assign this policy change to the appropriate computers.
   - **On schedule**, the client systems (Site Servers) check in, get the policy, client preboot policy checker notices the need to rebuild, and then calls BootWiz to rebuild the configuration(s).
     - As soon as the event is received to note this completion, a task is created and sent to the SBS server to let the server knows it has a "new" (updated) configuration for use.

- NOTE: DeployAnywhere drivers are replicated the same way the Bootwiz drivers are.
- NOTE: New configurations are essentially the same as these updates, minus the package replication.

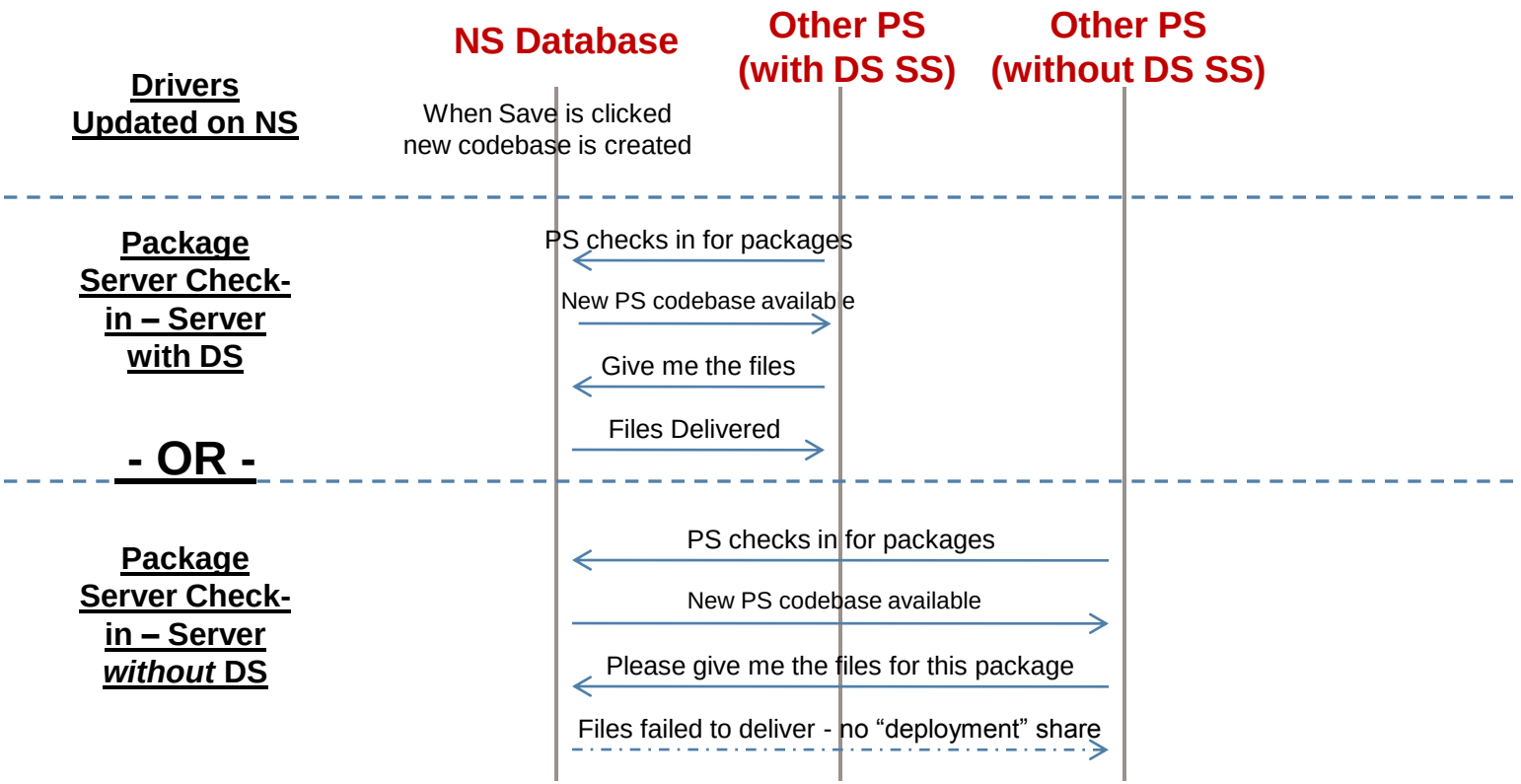
Dataflow: Image Replication



Though the process above appears to be very simple, there are some *very significant things that can be missed*, and in fact often *are* missed by clients, that can have very large repercussions.

- Images are replicated via the default Package Replication process in NS.
- All Packages in NS have a source location, or "Source Of Truth". If captured on a site server, the site server is the source of truth.
- If anything happens to the "Source of Truth", those changes are replicated out to all other package servers.
  - –This includes deletion of files. Thus, if you delete an image on a site server that is NOT the source of truth, the full image will be re-replicated to it. If you delete it from the Source of Truth, it will be gone from ALL site servers in short-order!!
- The NS, by default is NOT a package server.
  - –Thus, if you capture an image on a site server, it is NOT replicated back to the NS unless it is made a package server (not recommended).
- Images are replicated, by default, to ALL package-servers. However, if a PS doesn't have the DS SS on them, errors in PS replication will occur.

Dataflow: Driver Replication (DA Drivers and BootWiz Drivers)



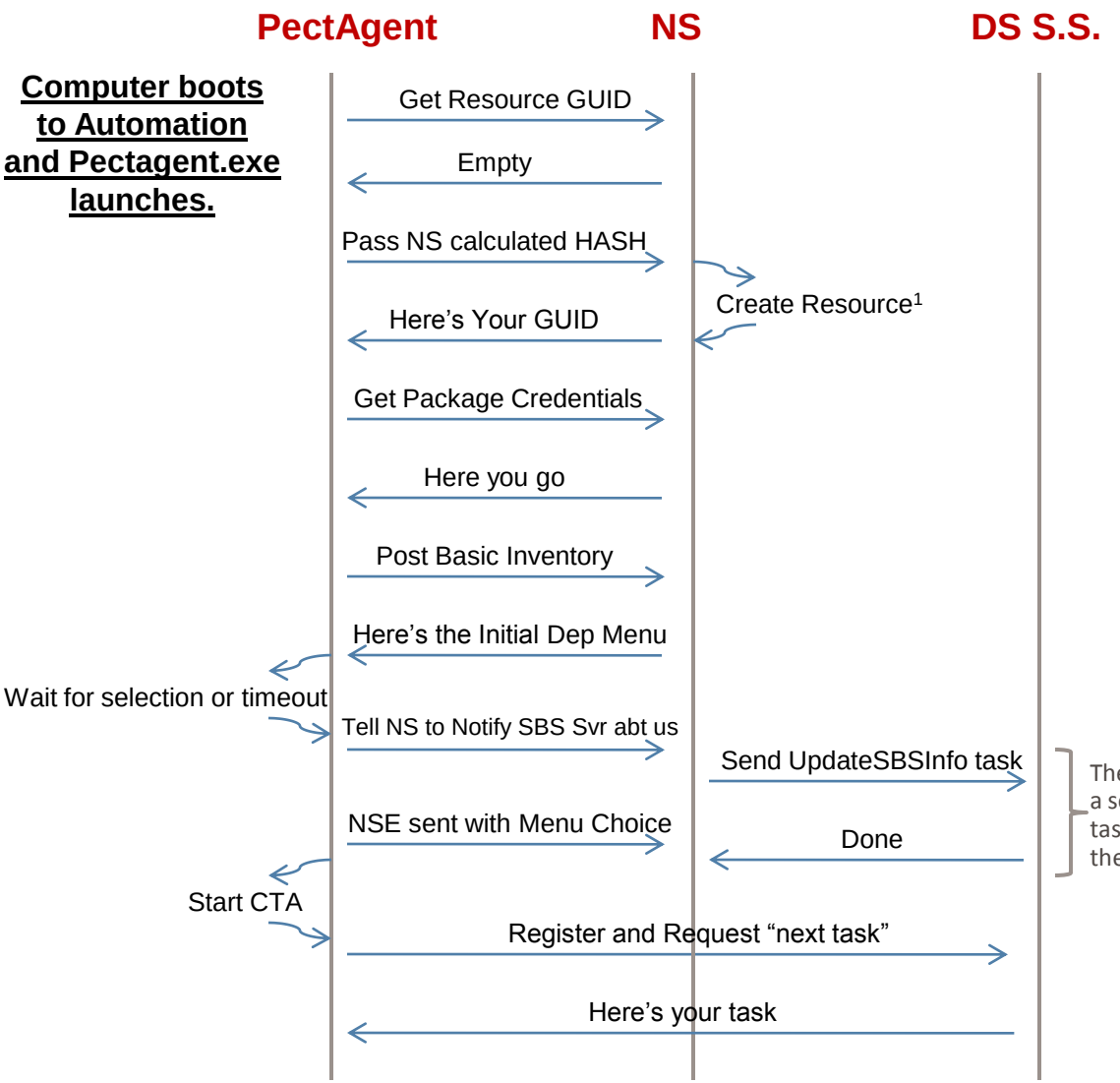
Notice that this process is very similar to the image replication process. There are a few key differences:

- The source of truth is always the NS
- The NS will have the drivers all the time.

Another very important point in this particular process is that for the BootWiz process to be successful on a Site Server after adding drivers, *this process must complete first* Many forget that once you add drivers, you can't simply kick-off a boot-wiz process and expect it to work. You have to first be sure the packages have replicated to the destination, and *THEN* start the Bootwiz process.

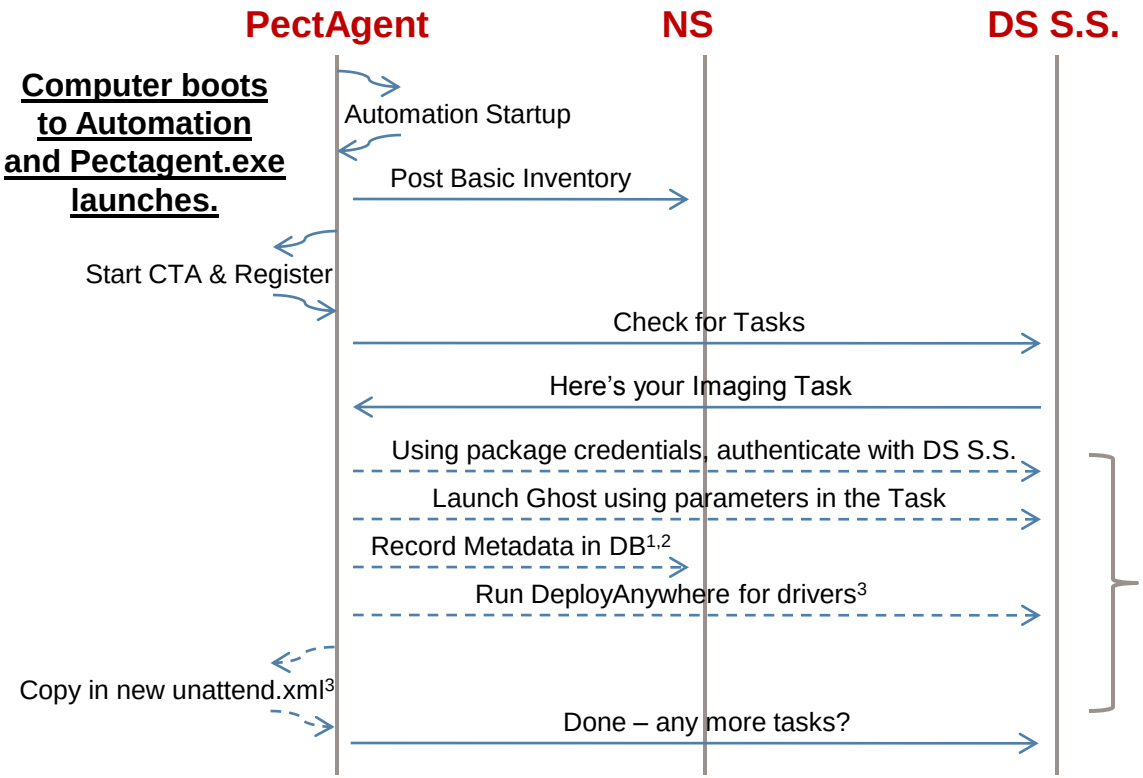
The same thing applies to DA drivers. If you want them to be picked-up for use during image deployment, ensure they've replicated first!

Dataflow: Initial Deployment



- [1]*This is new in 7.1 SP1, added because prior to this, Delta Resource Membership had to be run and it significantly delayed the process! We now insert the resource directly to the DB rather than waiting for Delta.*
- UpdateSBSinfo task runs at the same time as the NSE is being sent to the NS
- The first 7 steps (prior to inventory being posted) will be considered the "Automation Startup". This is because it is very similar to all other processes in automation, except for the 2nd and 3rd steps which are missing on already managed systems along with needing to create a resource
  - In future slides we'll just call it "Automation Startup"
- Note that the sending of the UpdateSBSInfo task is occurring at the same time as we send the NSE up.
  - After the selection or auto-selection, the next steps all occur in 1 second, and the UpdateSBSInfo task is occurring at the same time, but in the background.
  - This is one example of how some DS tasks have more than one task running simultaneously.
  - You can't build tasks like this in the console.

Dataflow: Disk Imaging in Automation (UNC)



Notice, this is all one task (dotted lines indicates no hand-off of communication or processes) with *no communication with the NS*, starting from accessing the DS to copying in the unattend.xml. The first communication with the NS is after the entire task is completed and we tell Object Host we're finished.

There are 4 imaging tasks that all have the same "root" process, which is why we combined it here. *The super-scripts indicate steps that only occur in one or more process* and apply to 1) Create Image, 2) Create Backup Image, 3) Deploy Image or 4) Restore Backup Image. If a step applies to more than one but not all 4, then more than one number will be indicated.

Finally, it is important that we record the image in the DB *after* the image is created. This is why you might see the image created but not show up in the console if something right at the end of the Ghost imaging process errors. This metadata includes if we found Windows Pro vs Win Enterprise, and things like that.

Please note that the metadata includes how the image was created. Backup images can *only* be used with restore tasks, and capture images with deploy tasks.

Thank you!

Thomas R Baird
Thomas_Baird@Symantec.com