# Hyperlaunch Design Document

This post is a Request for Comment on the included v4 of a design document that describes Hyperlaunch: a new method of launching the Xen hypervisor, relating to dom0less and work from the Hyperlaunch project. We invite discussion of this on this list, at the monthly Xen Community Calls, and at dedicated meetings on this topic in the Xen Working Group which will be announced in advance on the Xen Development mailing list.

# Contents

# 1 Introduction

This document describes the design and motivation for the funded development of a new, flexible system for launching the Xen hypervisor and virtual machines named: "Hyperlaunch".

The design enables seamless transition for existing systems that require a dom0, and provides a new general capability to build and launch alternative configurations of virtual machines, including support for static partitioning and accelerated start of VMs during host boot, while adhering to the principles of least privilege. It incorporates the existing dom0less functionality, extended to fold in the new developments from the Hyperlaunch project, with support for both x86 and Arm platform architectures, building upon and replacing the earlier 'late hardware domain' feature for disaggregation of dom0.

Hyperlaunch is designed to be flexible and reusable across multiple use cases, and our aim is to ensure that it is capable, widely exercised, comprehensively tested, and well understood by the Xen community.

# 2 Document Structure

This is the primary design document for Hyperlaunch, to provide an overview of the feature. Separate additional documents will cover specific aspects of Hyperlaunch in further detail, including:

- The Device Tree specification for Hyperlaunch metadata
- New Domain Roles for Xen and the Xen Security Modules (XSM) policy
- Passthrough of PCI devices with Hyperlaunch

# 3 Approach

Born out of improving support for Dynamic Root of Trust for Measurement (DRTM), the Hyperlaunch project is focused on restructuring the system launch of Xen. The Hyperlaunch design provides a security architecture that builds on the principles of Least Privilege and Strong Isolation, achieving this through the disaggregation of system functions. It enables this with the introduction of a boot domain that works in conjunction with the hypervisor to provide the ability to launch multiple domains as part of host boot while maintaining a least privilege implementation.

While the Hyperlaunch project inception was and continues to be driven by a focus on security through disaggregation, there are multiple use cases with a non-security focus that require or benefit from the ability to launch multiple domains at host boot. This was proven by the need that drove the implementation of the dom0less capability in the Arm branch of Xen.

Hyperlaunch is designed to be flexible and reusable across multiple use cases, and our aim is to ensure that it is capable, widely exercised, comprehensively tested, and provides a robust foundation for current and emerging system launch requirements of the Xen community.

## 3.1 Objectives

- In general strive to maintain compatibility with existing Xen behavior
- A default build of the hypervisor should be capable of booting both legacy-compatible and new styles of launch:
    - classic Xen boot: starting a single, privileged Dom0
    - classic Xen boot with late hardware domain: starting a Dom0 that transitions hardware access/control to another domain
    - a dom0less boot: starting multiple domains without privilege assignment controls
    - Hyperlaunch: starting one or more VMs, with flexible configuration
- Preferred that it be managed via KCONFIG options to govern inclusion of support for each style

- The selection between classic boot and Hyperlaunch boot should be automatic
    - Preferred that it not require a kernel command line parameter for selection
- It should not require modification to boot loaders
- It should provide a user friendly interface for its configuration and management
- It must provide a method for building systems that fallback to console access in the event of misconfiguration
- It should be able to boot an x86 Xen environment without the need for a Dom0 domain

# 4 Requirements and Design

Hyperlaunch is defined as the ability of a hypervisor to construct and start one or more virtual machines at system launch in a specific way. A hypervisor can support one or both modes of configuration, Hyperlaunch Static and Hyperlaunch Dynamic. The Hyperlaunch Static mode functions as a static partitioning hypervisor ensuring only the virtual machines started at system launch are running on the system. The Hyperlaunch Dynamic mode functions as a dynamic hypervisor allowing for additional virtual machines to be started after the initial virtual machines have started. The Xen hypervisor is capable of both modes of configuration from the same binary and when paired with its XSM flask, provides strong controls that enable fine grained system partitioning.

## 4.1 Hypervisor Launch Landscape

This comparison table presents the distinctive capabilities of Hyperlaunch with reference to existing launch configurations currently available in Xen and other hypervisors.

| **Xen Dom0 (Classic)** | **Linux KVM** | **Late HW Dom** | **Jail house** | **Xen dom0less** | **Xen Hyperlaunch** | |
|---|---|---|---|---|---|---|
| | | | | | Static | Dynamic |
| Hypervisor able to launch multiple VMs during host boot | | | | | | |
| | | | Y | Y | Y | Y |
| Hypervisor supports Static Partitioning | | | | | | |
| | | | Y | Y | Y | |
| Able to launch VMs dynamically after host boot | | | | | | |
| Y | Y | Y* | Y | Y* | | Y |
| Supports strong isolation between all VMs started at host boot | | | | | | |
| | | | Y | Y | Y | Y |
| Enables flexible sequencing of VM start during host boot | | | | | | |
| | | | | | Y | Y |
| Prevent all-powerful static root domain being launched at boot | | | | | | |
| | | | | Y* | Y | Y |
| Operates without a Highly-privileged management VM (eg. Dom0) | | | | | | |
| | | Y* | | Y* | Y | Y |
| Operates without a privileged toolstack VM (Control Domain) | | | | | | |
| | | | | Y* | Y | |
| Extensible VM configuration applied before launch of VMs at host boot | | | | | | |
| | | | | | Y | Y |

| | | | | | | |
|---|---|---|---|---|---|---|
| Flexible granular assignment of permissions and functions to VMs | | | | | | |
| | | | | | Y | Y |
| Supports extensible VM measurement architecture for DRTM and attestation | | | | | | |
| | | | | | Y | Y |
| PCI passthrough configured at host boot | | | | | | |
| | | | | | Y | Y |

## 4.2 Domain Construction

An important aspect of the Hyperlaunch architecture is that the hypervisor performs domain construction for all the Initial Domains, ie. it builds each domain that is described in the Launch Control Module. More specifically, the hypervisor will perform the function of *domain creation* for each Initial Domain: it allocates the unique domain identifier assigned to the virtual machine and records essential metadata about it in the internal data structure that enables scheduling the domain to run. It will also perform *basic domain construction*: build the initial page tables with data from the kernel and initial ramdisk supplied, and as appropriate for the domain type, populate the p2m table and ACPI tables.

Subsequent to this, the boot domain can apply additional configuration to the initial domains from the data in the LCM, in *extended domain construction*.

The benefits of this structure include:

- Security: Contrains the permissions required by the boot domain: it does not require the capability to create domains in this structure. This aligns with the principles of least privilege.
- Flexibility: Enables policy-based dynamic assignment of hardware by the boot domain, customizable according to use-case and able to adapt to hardware discovery
- Compatibility: Supports reuse of familiar tools with use-case customized boot domains.
- Commonality: Reuses the same logic for initial basic domain building across diverse Xen deployments.
    - It aligns the x86 initial domain construction with the existing Arm dom0less feature for construction of multiple domains at boot.
    - The boot domain implementation may vary significantly with different deployment use cases, whereas the hypervisor implementation is common.
- Correctness: Increases confidence in the implementation of domain construction, since it is performed by the hypervisor in well maintained and centrally tested logic.
- Performance: Enables launch for configurations where a fast start of multiple domains at boot is a requirement.
- Capability: Supports launch of advanced configurations where a sequenced start of multiple domains is required, or multiple domains are involved in startup of the running system configuration
    - eg. for PCI passthrough on systems where the toolstack runs in a separate domain to the hardware management.

Please, see the ‘Hyperlaunch Device Tree’ design document, which describes the configuration module that is provided to the hypervisor by the bootloader.

The hypervisor determines how these domains are started as host boot completes: in some systems the Boot Domain acts upon the extended boot configuration supplied as part of launch, performing configuration tasks for preparing the other domains for the hypervisor to commence running them.

## 4.3 Common Boot Configurations

When looking across those that have expressed interest or discussed a need for launching multiple domains at host boot, the Hyperlaunch approach is to provide the means to start nearly any combination of domains. Below is an enumerated selection of common boot configurations for reference in the following section.

### *4.3.1 Dynamic Launch with a Highly-Privileged Domain 0*

**Hyperlaunch Classic: Dom0**

This configuration mimics the classic Xen start and domain construction where a single domain is constructed with all privileges and functions for managing hardware and running virtualization toolstack software.

**Hyperlaunch Classic: Extended Launch Dom0**

This configuration is where a Dom0 is started via a Boot Domain that runs first. This is for cases where some preprocessing in a less privileged domain is required before starting the all-privileged Domain 0.

**Hyperlaunch Classic: Basic Cloud**

This configuration constructs a Dom0 that is started in parallel with some number of workload domains.

**Hyperlaunch Classic: Cloud**

This configuration builds a Dom0 and some number of workload domains, launched via a Boot Domain that runs first.

### *4.3.2 Static Launch Configurations: without a Domain 0 or a Control Domain*

**Hyperlaunch Static: Basic**

Simple static partitioning where all domains that can be run on this system are built and started during host boot and where no domain is started with the Control Domain permissions, thus making it not possible to create/start any further new domains.

**Hyperlaunch Static: Standard**

This is a variation of the “Hyperlaunch Static: Basic” static partitioning configuration with the introduction of a Boot Domain. This configuration allows for use of a Boot Domain to be able to apply extended configuration to the Initial Domains before they are started and sequence the order in which they start.

**Hyperlaunch Static: Disaggregated**

This is a variation of the “Hyperlaunch Static: Standard” configuration with the introduction of a Boot Domain and an illustration that some functions can be disaggregated to dedicated domains.

### *4.3.3 Dynamic Launch of Disaggregated System Configurations*

**Hyperlaunch Dynamic: Hardware Domain**

This configuration mimics the existing Xen feature late hardware domain with the one difference being that the hardware domain is constructed by the hypervisor at startup instead of later by Dom0.

**Hyperlaunch Dynamic: Flexible Disaggregation**

This configuration is similar to the “Hyperlaunch Classic: Dom0” configuration except that it includes starting a separate hardware domain during Xen startup. It is also similar to “Hyperlaunch Dynamic: Hardware Domain” configuration, but it launches via a Boot Domain that runs first.

**Hyperlaunch Dynamic: Full Disaggregation**

In this configuration it is demonstrated how it is possible to start a fully disaggregated system: the virtualization toolstack runs in a Control Domain, separate from the domains responsible for managing hardware, XenStore, the Xen Console and Crash functions, each launched via a Boot Domain.

### 4.3.4 Example Use Cases and Configurations

The following example use cases can be matched to configurations listed in the previous section.

#### 4.3.4.1 Use case: Modern cloud hypervisor

**Option:** Hyperlaunch Classic: Cloud

This configuration will support strong isolation for virtual TPM domains and measured launch in support of attestation to infrastructure management, while allowing the use of existing Dom0 virtualization toolstack software.

#### 4.3.4.2 Use case: Edge device with security or safety requirements

**Option:** Hyperlaunch Static: Boot

This configuration runs without requiring a highly-privileged Dom0, and enables extended VM configuration to be applied to the Initial VMs prior to launching them, optionally in a sequenced start.

#### 4.3.4.3 Use case: Client hypervisor

**Option:** Hyperlaunch Dynamic: Flexible Disaggregation

**Option:** Hyperlaunch Dynamic: Full Disaggregation

These configurations enable dynamic client workloads, strong isolation for the domain running the virtualization toolstack software and each domain managing hardware, with PCI passthrough performed during host boot and support for measured launch.

## 4.4 Hyperlaunch Disaggregated Launch

Existing in Xen today are two primary permissions, *control domain* and *hardware domain*, and two functions, *console domain* and *xenstore domain*, that can be assigned to a domain. Traditionally all of these permissions and functions are all assigned to Dom0 at start and can then be delegated to other domains created by the toolstack in Dom0. With Hyperlaunch it becomes possible to assign these permissions and functions to any domain for which there is a definition provided at startup.

Additionally, two further functions are introduced: the *recovery domain*, intended to assist with recovery from failures encountered starting VMs during host boot, and the *boot domain*, for performing aspects of domain construction during startup.

Supporting the booting of each of the above common boot configurations is accomplished by considering the set of initial domains and the assignment of Xen's permissions and functions, including the ones introduced by Hyperlaunch, to these domains. A discussion of these will be covered later but for now they are laid out in a table with a mapping to the common boot configurations. This table is not intended to be an exhaustive list of configurations and does not account for flask policy specified functions that are use case specific.

In the table each number represents a separate domain being constructed by the Hyperlaunch construction path as Xen starts, and the designator, {n} signifies that there may be "n" additional domains that may be constructed that do not have any special role for a general Xen system.

| Configuration | Permission | | | Function | | | |
|---|---|---|---|---|---|---|---|
| | None | Ctrl | HW | Boot | Recovery | Console | Xenstore |
| Classic: Dom0 | | 0 | 0 | | 0 | 0 | 0 |
| Classic: Extended Launch Dom0 | | 1 | 1 | 0 | 1 | 1 | 1 |
| Classic: Basic Cloud | {n} | 0 | 0 | | 0 | 0 | 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Classic: Cloud | {n} | 1 | 1 | 0 | 1 | 1 | 1 |
| Static: Basic | {n} | | 0 | | 0 | 0 | 0 |
| Static: Standard | {n} | | 1 | 0 | 1 | 1 | 1 |
| Static: Disaggregated | {n} | | 2 | 0 | 3 | 4 | 1 |
| Dynamic: Hardware Domain | | 0 | 1 | | 0 | 0 | 0 |
| Dynamic: Flexible Disaggregation | {n} | 1 | 2 | 0 | 1 | 1 | 1 |
| Dynamic: Full Disaggregation | {n} | 2 | 3 | 0 | 4 | 5 | 1 |

## 4.5 Overview of Hyperlaunch Flow

Before delving into Hyperlaunch, a good basis to start with is an understanding of the current process to create a domain. A way to view this process starts with the core configuration which is the information the hypervisor requires to make the call to *domain_create*, followed by basic construction to provide the memory image to run, including the kernel and ramdisk. A subsequent step applies the extended configuration used by the toolstack to provide a domain with any additional configuration information. Until the extended configuration is completed, a domain has access to no resources except its allocated vcpus and memory. The exception to this is Dom0, which the hypervisor explicitly grants control and access to all system resources, except for those that only the hypervisor should have control over. This exception for Dom0 is driven by the system structure with a monolithic Dom0 domain predating introduction of support for disaggregation into Xen, and the corresponding default assignment of multiple roles within the Xen system to Dom0.

While not a different domain creation path, there does exist the Hardware Domain (hwdom), sometimes also referred to as late-Dom0. It is an early effort to disaggregate Dom0's roles into a separate control domain and hardware domain. This capability is activated by the passing of a domain id to the *hardware_dom* kernel command line parameter, and the Xen hypervisor will then flag that domain id as the hardware domain. Later when the toolstack constructs a domain with that domain id as the requested domid, the hypervisor will transfer all device I/O from Dom0 to this domain. In addition it will also transfer the "host shutdown on domain shutdown" flag from Dom0 to the hardware domain. It is worth mentioning that this approach for disaggregation was created in this manner due to the inability of Xen to launch more than one domain at startup.

### *4.5.1 Hyperlaunch Xen startup*

The Hyperlaunch approach's primary focus is on how to assign the roles traditionally granted to Dom0 to one or more domains at host boot. While the statement is simple to make, the implications are not trivial by any means. This also explains why the Hyperlaunch approach is orthogonal to the existing dom0less capability. The dom0less capability focuses on enabling the launch of multiple domains in parallel with Dom0 at host boot. A corollary for dom0less is that for systems that don't require Dom0 after all guest domains have started, they are able to do the host boot without a Dom0. Though it should be noted that it may be possible to start Dom0 at a later point. Whereas with Hyperlaunch, its approach of separating Dom0's roles requires the ability to launch multiple domains at host boot. The direct consequences from this approach are profound and provide a myriad of possible configurations for which a sample of common boot configurations were already presented.

To enable the Hyperlaunch approach a new alternative path for host boot within the hypervisor must be introduced. This alternative path effectively branches just before the current point of Dom0 construction and begins an alternate means of system construction. The determination if this alternate path should be taken is through the inspection of the boot chain. If the bootloader has loaded a specific configuration, as described later, it will enable Xen to detect that a Hyperlaunch configuration has been provided. Once a Hyperlaunch configuration is detected, this alternate path can be thought of as occurring in phases: domain creation, domain preparation, and launch finalization.

#### *4.5.1.1 Domain Creation*

The domain creation phase begins with Xen parsing the bootloader provided material, to understand the content of the modules provided. It will then load any microcode or XSM policy it discovers. For each domain configuration Xen finds, it parses the configuration to construct the necessary domain definition to instantiate an instance of the domain and leave it in a paused state. When all domain configurations have been instantiated as domains, if one of them is flagged as the Boot Domain, that domain will be unpaused starting the domain preparation phase. If there is no Boot Domain defined, then the domain preparation phase will be skipped and Xen will trigger the launch finalization phase.

#### *4.5.1.2 Domain Preparation Phase*

The domain preparation phase is an optional check point for the execution of a workload specific domain, the Boot Domain. While the Boot Domain is the first domain to run and has some degree of control over the system, it is extremely restricted in both system resource access and hypervisor operations. Its purpose is to:

- Access the configuration provided by the bootloader
- Finalize the configuration of the domains
- Conduct any setup and launch related operations
- Do an ordered unpause of domains that require an ordered start

When the Boot Domain has completed, it will notify the hypervisor that it is done triggering the launch finalization phase.

#### *4.5.1.3 Launch Finalization*

The hypervisor handles the launch finalization phase which is equivalent to the clean up phase. As such the steps taken by the hypervisor, not necessarily in implementation order, are as follows,

- Free the boot module chain
- If a Boot Domain was used, reclaim Boot Domain resources
- Unpause any domains still in a paused state
- Boot Domain uses a reserved function thus can never be respawned

While the focus thus far has been on how the Hyperlaunch capability will work, it is worth mentioning what it does not do or limit from occurring. It does not stop or inhibit the assigning of the control domain role which gives the domain the ability to create, start, stop, restart, and destroy domains or the hardware domain role which gives access to all I/O devices except those that the hypervisor has reserved for itself. In particular it is still possible to construct a domain with all the privileged roles, i.e. a Dom0, with or without the domain id being zero. In fact what limitations are imposed now become fully configurable without the risk of circumvention by an all privileged domain.

## 4.6 Structuring of Hyperlaunch

The structure of Hyperlaunch is built around the existing capabilities of the host boot protocol. This approach was driven by the objective not to require modifications to the boot loader. The only requirement is that the boot loader supports the Multiboot2 (MB2) protocol. For UEFI boot, our recommendation is to use GRUB.efi to load Xen and the initial domain materials via the multiboot2 method. On Arm platforms, Hyperlaunch is compatible with the existing interface for boot into the hypervisor.

### 4.6.1 x86 Multiboot2

The MB2 protocol has no concept of a manifest to tell the initial kernel what is contained in the chain, leaving it to the kernel to impose a loading convention, use magic number identification, or both. When considering the passing of multiple kernels, ramdisks, and domain configuration along with any existing modules already passed, there is no sane convention that could be imposed and magic number identification is nearly impossible when considering the objective not to impose unnecessary complication to the hypervisor.

As it was alluded to previously, a manifest describing the contents in the MB2 chain and how they relate within a Xen context is needed. To address this need the Launch Control Module (LCM) was designed to provide such a manifest. The LCM was designed to have a specific set of properties,

- minimize the complexity of the parsing logic required by the hypervisor
- allow for expanding and optional configuration fragments without breaking backwards compatibility

To enable automatic detection of a Hyperlaunch configuration, the LCM must be the first MB2 module in the MB2 module chain. The LCM is implemented using the Device Tree as defined in the Hyperlaunch Device Tree design document. With the LCM implemented in Device Tree, it has a magic number that enables the hypervisor to detect its presence when used in a Multiboot2 module chain. The hypervisor can confirm that it is a proper LCM Device Tree by checking for a compliant Hyperlaunch Device Tree. The Hyperlaunch Device Tree nodes are designed to allow,

- for the hypervisor to parse only those entries it understands,
- for packing custom information for a custom boot domain,
- the ability to use a new LCM with an older hypervisor,
- and the ability to use an older LCM with a new hypervisor.

### 4.6.2 Arm Device Tree

As discussed the LCM is in Device Tree format and was designed to co-exist in the Device Tree ecosystem, and in particular in parallel with dom0less Device Tree entries. On Arm, Xen is already designed to boot from a host Device Tree description (dtb) file and the LCM entries can be embedded into this host dtb file. This makes detecting the LCM entries and supporting Hyperlaunch on Arm relatively straight forward. Relative to the described x86 approach, at the point where Xen inspects the first MB2 module, on Arm Xen will check if the top level LCM node exists in the host dtb file. If the LCM node does exist, then at that point it will enter into the same code path as the x86 entry would go.

### 4.6.3 Xen hypervisor

It was previously discussed at a higher level of the new host boot flow that will be introduced. Within this new flow is the configuration parsing and domain creation phase which will be expanded upon here. The hypervisor will inspect the LCM for a config node and if found will iterate through all modules nodes. The module nodes are used to identify if any modules contain microcode or an XSM policy. As it processes domain nodes, it will construct the domain using the node properties and the modules nodes. Once it has completed iterating through all the entries in the LCM, if a constructed domain has the Boot Domain attribute, it will then be unpaused. Otherwise the hypervisor will start the launch finalization phase.

### 4.6.4 Boot Domain

Traditionally domain creation was controlled by the user within the Dom0 environment whereby custom toolstacks could be implemented to impose requirements on the process. The Boot Domain is a means to enable the user to continue to maintain a degree of that control over domain creation but within a limited privilege environment. The Boot Domain will have access to the LCM and the boot chain along with access to a subset of the hypercall operations. When the Boot Domain is finished it will notify the hypervisor through a hypercall op.

### 4.6.5 Recovery Domain

With the existing Dom0 host boot path, when a failure occurs there are several assumptions that can safely be made to get the user to a console for troubleshooting. With the Hyperlaunch host boot path those assumptions can no longer be made, thus a means is needed to get the user to a console in the case of a recoverable failure. The recovery domain is configured by a domain configuration entry in the LCM, in the same manner as the other initial domains, and it will not be unpaused at launch finalization unless a failure is encountered starting the initial domains.

Xen has existing support for a Crash Environment where memory can be reserved at host boot and a kernel loaded into it, to be jumped into at any point while the system is running when a crash is detected. The Recovery Domain functionality is a separate, complementary capability. The Crash Environment replaces the previously active hypervisor and running guests, and enables a process for mounting disks to write out log information prior to rebooting the system. In contrast, the Recovery Domain is able to use the functionality of the Xen hypervisor, that is still present and running, to perform recovery handling for errors encountered with starting the initial domains.

#### 4.6.5.1 Deferred Design

To be determined:

- Define what is detected as a crash
- Explain how crash detection is performed and which components are involved
- Explain how the recovery domain is unpaused
- Explain how and when the resources assigned to the recovery domain are reclaimed
- Define what the recovery domain is able to do
- Determine what permissions the recovery domain requires to perform its job

### 4.6.6 Control Domain

The concept of the Control Domain already exists within Xen as a boolean, *is_privileged*, that governs access to many of the privileged interfaces of the hypervisor that support a domain running a virtualization system toolstack. Hyperlaunch will allow the *is_privileged* flag to be set on any domain that is created at launch, rather than only a Dom0. It may potentially be set on multiple domains.

### 4.6.7 Hardware Domain

The Hardware Domain is also an existing concept for Xen that is enabled through the *is_hardware_domain* check. With Hyperlaunch the previous process of I/O accesses being assigned to Dom0 for later transfer to the hardware domain would no longer be required. Instead during the configuration phase the Xen hypervisor would directly assign the I/O accesses to the domain with the hardware domain permission bit enabled.

### 4.6.8 Console Domain

Traditionally the Xen console is assigned to the control domain and then reassignable by the toolstack to another domain. With Hyperlaunch it becomes possible to construct a boot configuration where there is no control domain or have a use case where the Xen console needs to be isolated. As such it becomes necessary to be able to designate which of the initial domains should be assigned the Xen console. Therefore Hyperlaunch introduces the ability to specify an initial domain which the console is assigned along with a convention of ordered assignment for when there is no explicit assignment.

# 5 Communication of Domain Configurations

There are several standard methods for an Operating System to access machine configuration and environment information: ACPI is common on x86 systems, whereas Device Tree is more typical on Arm platforms. There are currently implementations of both in Xen.

- For dom0less, guest Device Trees are dynamically constructed by the hypervisor to convey domain configuration data
- For PVH dom0 on x86, ACPI tables are built by the hypervisor before the domain is started

Note that both of these mechanisms convey static data that is fixed prior to the point of domain construction. Hyperlaunch will retain both the existing ACPI and Device Tree methods.

Communication of data between a Boot Domain and a Control Domain is of note since they may not be running concurrently: the method used will depend on their specific implementations, but one option available is to use Xen's hypfs for transfer of basic data to support system bootstrap.

---

# 6 Appendix

## 6.1 Appendix 1: Flow Sequence of Steps of a Hyperlaunch Boot

Provided here is an ordered flow of a Hyperlaunch with a highlight logic decision points. Not all branch points are recorded, specifically for the variety of error conditions that may occur.

```
1. Hypervisor Startup:
2a. (x86) Inspect first module provided by the bootloader
    a. Is the module an LCM
        i. YES: proceed with the Hyperlaunch host boot path
        ii. NO: proceed with a Dom0 host boot path
2b. (Arm) Inspect host dtb for `/chosen/hypervisor` node
    a. Is the LCM present
        i. YES: proceed with the Hyperlaunch host boot path
        ii. NO: proceed with a Dom0/dom0less host boot path
3. Iterate through the LCM entries looking for the module description
   entry
    a. Check if any of the modules are microcode or policy and if so,
       load
4. Iterate through the LCM entries processing all domain description
   entries
    a. Use the details from the Basic Configuration to call
       `domain_create`
    b. Record if a domain is flagged as the Boot Domain
    c. Record if a domain is flagged as the Recovery Domain
5. Was a Boot Domain created
    a. YES:
        i. Attach console to Boot Domain
        ii. Unpause Boot Domain
        iii. Goto Boot Domain (step 6)
    b. NO: Goto Launch Finalization (step 10)
6. Boot Domain:
7. Boot Domain comes online and may do any of the following actions
    a. Process the LCM
    b. Validate the MB2 chain
    c. Make additional configuration settings for staged domains
    d. Unpause any precursor domains
    e. Set any runtime configurations
8. Boot Domain does any necessary cleanup
9. Boot Domain make hypercall op call to signal it is finished
    i. Hypervisor reclaims all Boot Domain resources
```

```
    ii. Hypervisor records that the Boot Domain ran
    ii. Goto Launch Finalization (step 9)
10. Launch Finalization
11. If a configured domain was flagged to have the console, the
    hypervisor assigns it
12. The hypervisor clears the LCM and bootloader loaded module,
    reclaiming the memory
13. The hypervisor iterates through domains unpausing any domain not
    flagged as the recovery domain
```

## 6.2 Appendix 2: Considerations in Naming the Hyperlaunch Feature

- The term “Launch” is preferred over “Boot”
    - Multiple individual component boots can occur in the new system start process; Launch is preferable for describing the whole process
    - Fortunately there is consensus in the current group of stakeholders that the term “Launch” is good and appropriate
- The names we define must support becoming meaningful and simple to use outside the Xen community
    - They must be able to be resolved quickly via search engine to a clear explanation (eg. Xen marketing material, documentation or wiki)
    - We prefer that the terms be helpful for marketing communications
    - Consequence: avoid the term “domain” which is Xen-specific and requires a definition to be provided each time when used elsewhere
- There is a need to communicate that Xen is capable of being used as a Static Partitioning hypervisor
    - The community members using and maintaining dom0less are the current primary stakeholders for this
- There is a need to communicate that the new launch functionality provides new capabilities not available elsewhere, and is more than just supporting Static Partitioning
    - No other hypervisor known to the authors of this document is capable of providing what Hyperlaunch will be able to do. The launch sequence is designed to:
        - Remove dependency on a single, highly-privileged initial domain
        - Allow the initial domains started to be independent and fully isolated from each other
        - Support configurations where no further VMs can be launched once the initial domains have started
        - Use a standard, extensible format for conveying VM configuration data
        - Ensure that domain building of all initial domains is performed by the hypervisor from materials supplied by the bootloader
        - Enable flexible configuration to be applied to all initial domains by an optional Boot Domain, that runs with limited privilege, before any other domain starts and obtains the VM configuration data from the bootloader materials via the hypervisor

        - Enable measurements of all of the boot materials prior to their use, in a sequence with minimized privilege
        - Support use-case-specific customized Boot Domains
        - Complement the hypervisor's existing ability to enforce policy-based Mandatory Access Control
- "Static" and "Dynamic" have different and important meanings in different communities
    - Static and Dynamic Partitioning describe the ability to create new virtual machines, or not, after the initial host boot process completes
    - Static and Dynamic Root of Trust describe the nature of the trust chain for a measured launch. In this case Static is referring to the fact that the trust chain is fixed and non-repeatable until the next host reboot or shutdown. Whereas Dynamic in this case refers to the ability to conduct the measured launch at any time and potentially multiple times before the next host reboot or shutdown.
        - We will be using Hyperlaunch with both Static and Dynamic Roots of Trust, to launch both Static and Dynamically Partitioned Systems, and being clear about exactly which combination is being started will be very important (eg. for certification processes)
    - Consequence: uses of "Static" and "Dynamic" need to be qualified if they are incorporated into the naming of this functionality
        - This can be done by adding the preceding, stronger branded term: "Hyperlaunch", before "Static" or "Dynamic"
        - ie. "Hyperlaunch Static" describes launch of a Statically Partitioned system and "Hyperlaunch Dynamic" describes launch of a Dynamically Partitioned system.
        - In practice, this means that "Hyperlaunch Static" describes starting a Static Partitioned system where no new domains can be started later (ie. no VM has the Control Domain permission), whereas "Hyperlaunch Dynamic" will launch some VM with the Control Domain permission, able to create VMs dynamically at a later point.

**Naming Proposal:**

- New Term: "Hyperlaunch" : the ability of a hypervisor to construct and start one or more virtual machines at system launch, in the following manner:
    - The hypervisor must build all of the domains that it starts at host boot
        - Similar to the way the dom0 domain is built by the hypervisor today, and how dom0less works: it will run a loop to build them all, driven from the configuration provided
        - This is a requirement for ensuring that there is Strong Isolation between each of the initial VMs
    - A single file contains the VM configs ("Launch Control Module": LCM, in Device Tree binary format) is provided to the hypervisor
        - The hypervisor parses it and builds domains
        - If the LCM config says that a Boot Domain should run first, then the LCM file itself is made available to the Boot Domain for it to parse and act on, to invoke operations via the hypervisor to apply additional configuration to the other VMs (ie. executing a privilege-constrained toolstack)

- New Term: “Hyperlaunch Static”: starts a Static Partitioned system, where only the virtual machines started at system launch are running on the system
- New Term: “Hyperlaunch Dynamic”: starts a system where virtual machines may be dynamically added after the initial virtual machines have started.

In the default configuration, Xen will be capable of both styles of Hyperlaunch from the same hypervisor binary, when paired with its XSM flask, provides strong controls that enable fine grained system partitioning.

- Retiring Term: “DomB”: will no longer be used to describe the optional first domain that is started. It is replaced with the more general term: “Boot Domain”.
- Retiring Term: “Dom0less”: it is to be replaced with “Hyperlaunch Static”

## 6.3 Appendix 3: Terminology

To help ensure clarity in reading this document, the following is the definition of terminology used within this document.

**Basic Configuration**

the minimal information the hypervisor requires to instantiate a domain instance

**Boot Domain**

a domain with limited privileges launched by the hypervisor during a Multiple Domain Boot that runs as the first domain started. In the Hyperlaunch architecture, it is responsible for assisting with higher level operations of the domain setup process.

**Classic Launch**

a backwards-compatible host boot that ends with the launch of a single domain (Dom0)

**Console Domain**

a domain that has the Xen console assigned to it

**Control Domain**

a privileged domain that has been granted Control Domain permissions which are those that are required by the Xen toolstack for managing other domains. These permissions are a subset of those that are granted to Dom0.

**Device Tree**

a standardized data structure, with defined file formats, for describing initial system configuration

**Disaggregation**

the separation of system roles and responsibilities across multiple connected components that work together to provide functionality

**Dom0**

the highly-privileged, first and only domain started at host boot on a conventional Xen system

**Dom0less**

an existing feature of Xen on Arm that provides Multiple Domain Boot

**Domain**

a running instance of a virtual machine; (as the term is commonly used in the Xen Community)

**DomB**

the former name for Hyperlaunch

**Extended Configuration**

any configuration options for a domain beyond its Basic Configuration

**Hardware Domain**

a privileged domain that has been granted permissions to access and manage host hardware. These permissions are a subset of those that are granted to Dom0.

**Host Boot**

the system startup of Xen using the configuration provided by the bootloader

**Hyperlaunch**

a flexible host boot that ends with the launch of one or more domains

**Initial Domain**

a domain that is described in the LCM that is run as part of a multiple domain boot. This includes the Boot Domain, Recovery Domain and all Launched Domains.

**Late Hardware Domain**

a Hardware Domain that is launched after host boot has already completed with a running Dom0. When the Late Hardware Domain is started, Dom0 relinquishes and transfers the permissions to access and manage host hardware to it..

**Launch Control Module (LCM)**

A file supplied to the hypervisor by the bootloader that contains configuration data for the hypervisor and the initial set of virtual machines to be run at boot

**Launched Domain**

a domain, aside from the boot domain and recovery domain, that is started as part of a multiple domain boot and remains running once the boot process is complete

**Multiple Domain Boot**

a system configuration where the hypervisor and multiple virtual machines are all launched when the host system hardware boots

**Recovery Domain**

an optional fallback domain that the hypervisor may start in the event of a detectable error encountered during the multiple domain boot process

**System Device Tree**

this is the product of an Arm community project to extend Device Tree to cover more aspects of initial system configuration

## 6.4 Appendix 4: Copyright License