Solaris Hotplug Framework

Govinda Tatti

Solaris Platform Software

Sun Microsystems Inc.

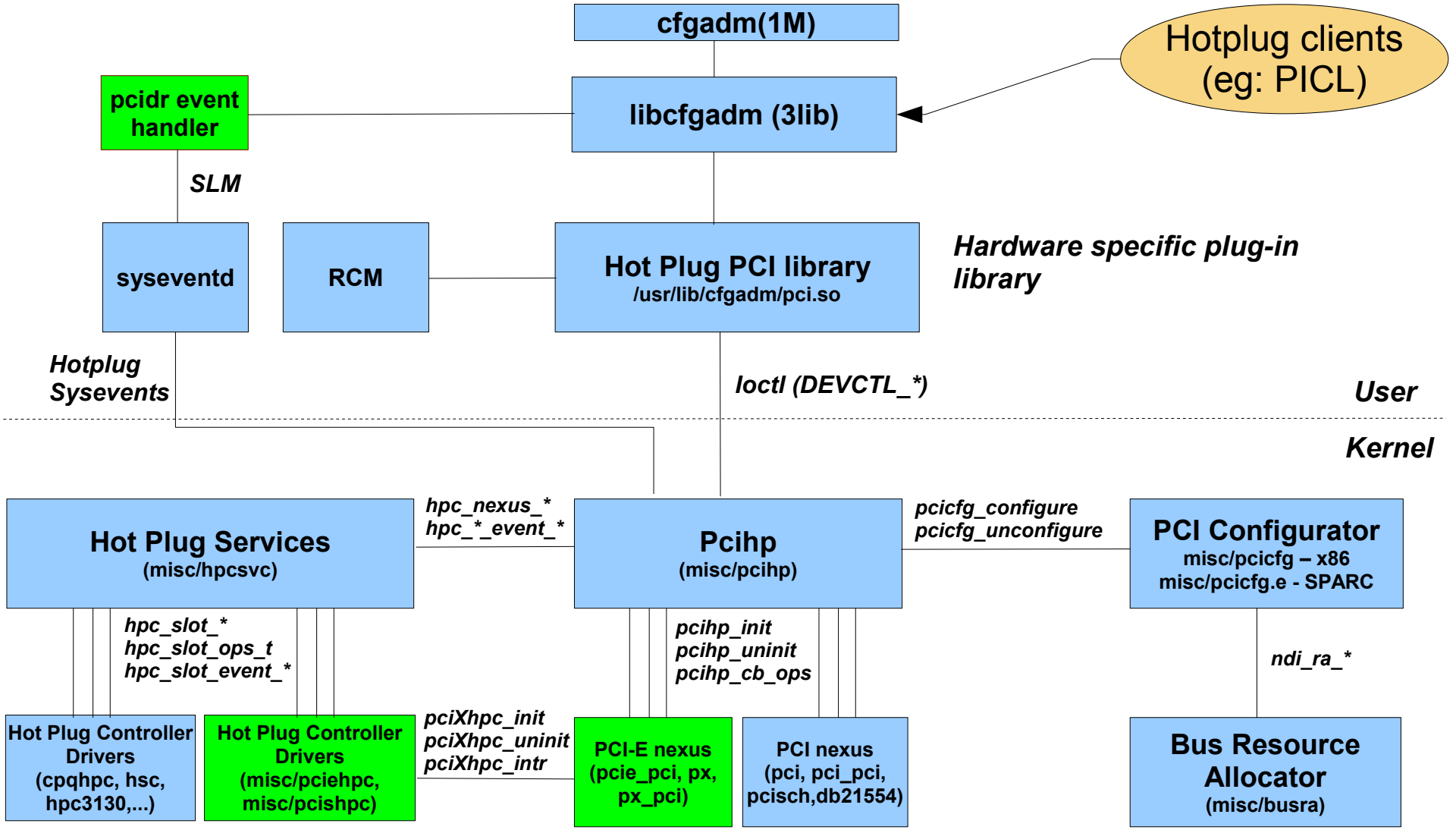
Agenda

- Overview of new Solaris hotplug framework
- State Machine
- HPC Driver Interfaces
- PCI Express Hotplug Enhancements
- Device Driver Interfaces
- Userland Hotplug Overview
- Implementation strategy
- References

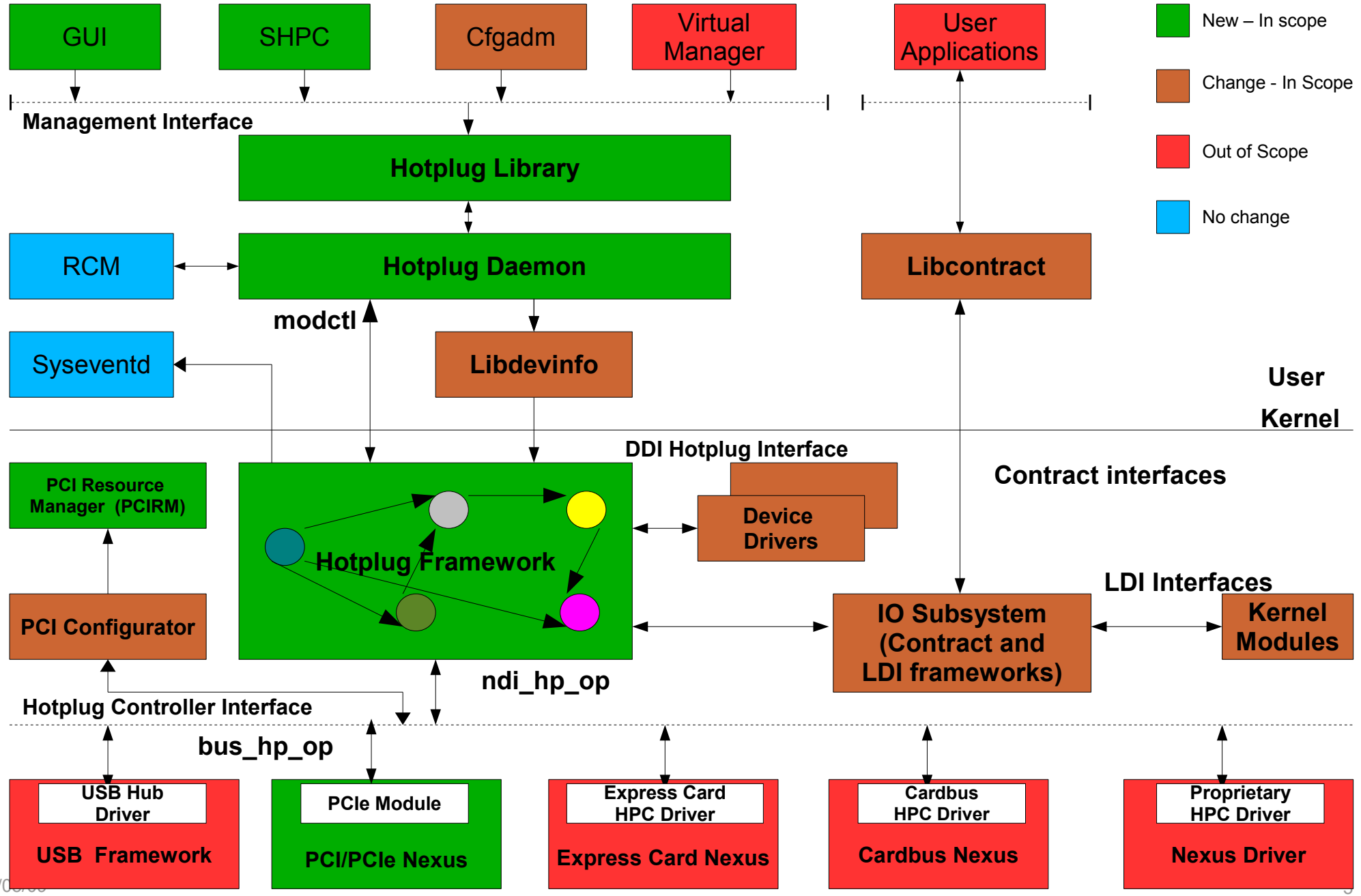
Solaris Hotplug Framework - Overview

- A generic common hotplug framework to support the hotplug functionality for any hotpluggable bus
- Provides generic and common hotplug interfaces
  - Management Interfaces
    - Generic or Bus specific hotplug admin applications including GUIs can be written using these interfaces
  - Hotplug Event Interfaces
    - Hotplug-aware users such as drivers, kernel modules and hotplug aware applications can receive both synchronous and asynchronous hotplug events
  - Device Driver Interfaces
    - Device drivers can work with the hotplug framework to participate in hotplug activities such as surprise removal, hot replacement, dynamic resource balancing, error handling
  - Hotplug Controller Driver (HPC) or Nexus Interfaces
    - HPC drivers will communicate with the hotplug framework through these interfaces
    - New HPC driver can be written easily for any hotpluggable bus

Current PCI Hotplug Framework



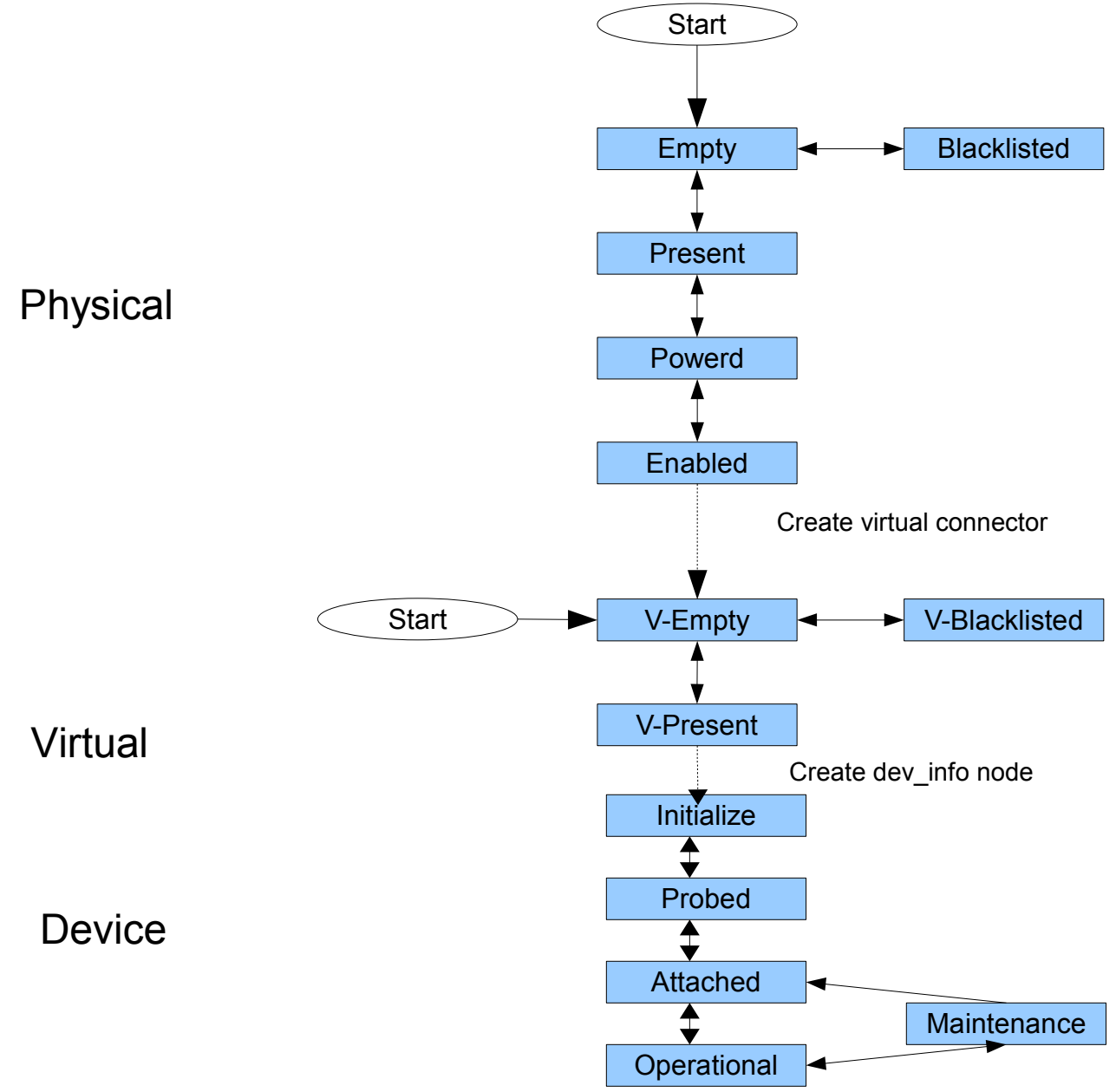
Solaris Hotplug Framework - Block Diagram



Kernel Hotplug Support

- Solaris Hotplug framework is responsible for
  - Manage State information for each hotplug connectors (physical and virtual)
  - Communicate with Hotplug Controller (HPC) drivers to perform all physical and virtual hotplug operations on each connector
    - Register, unregister and post events
    - Communicate with HPC drivers to perform hotplug operations
  - Generate Hotplug Events to all hotplug-aware Kernel and Userland entities
    - Device Contract Events for Hotplug-aware User Applications
    - LDI Events for Hotplug-aware Kernel consumers
    - Newly defined Hotplug Events for Device Drivers

Hotplug Framework State Machine



Hotplug Controller Drivers

- A hotplug controller driver (HPC) is a Solaris device driver written for a standard/proprietary bus specific hotplug controller hardware
- New hotplug interfaces for HPC device drivers
  - Register, unregister and post events through ndi_hp_register(), ndi_hp_unregister() and ndi_hp_state_change_req()
  - Communicate with HPC drivers to perform hotplug operations through bus_hp_op()

NDI Hotplug Interfaces

```
int ndi_hp_register(dev_info_t *dip,        /* parent node of the connection */
        char *cn_name);                     /* name of the connection */

int ndi_hp_unregister(dev_info_t *dip,      /* parent node of the connection */
        char *cn_name);                     /* name of the connection */

int ndi_hp_state_change_req(
        dev_info_t *dip,                    /* parent node of the connection */
        char *cn_name,                      /* name of the connection */
        ddi_hp_cn_state_t state,            /* target state */
        int flag)                           /* flag: SYNC or ASYNC */
```

bus_hp_op

```
int bus_hp_op(dev_info_t dip,     /* parent node of the connection */
        char cn_name,             /* name of the connection */
        ddi_hp_op_t op,           /* operations to be done for the connection */
        void arg,                 /* additional input arg for the operation */
        void result);             /* operation result */
```

```
typedef enum {
    DDI_HPOP_CHECK_HP_SUPPORT = 1,   /* 01 Check hotplug support */
    DDI_HPOP_CN_GET_STATE,           /* 02 Get connection state */
    DDI_HPOP_CN_CHANGE_STATE,        /* 03 Change connection state */
    DDI_HPOP_CN_PROBE,               /* 04 Probe connection */
    DDI_HPOP_CN_UNPROBE,             /* 05 Unprobe connection */
    DDI_HPOP_CN_ONLINE_CHILDREN,     /* 06 Online all the children of connection */
    DDI_HPOP_CN_OFFLINE_CHILDREN,    /* 07 Offline all the children of connection */
} ddi_hp_op_t;
```

Virtual Hotplug Support

- Hotplug arbitrary device nodes in the device tree
  - This feature will be used to migrate devices between the domains in virtualized environment
  - Dynamically add or remove any device nodes into device tree based on user requests
  - Create a virtual hotplug attachment point for every device node
  - Current virtual hotplug implementation is limited to PCI/PCIe devices

PCIe Hotplug enhancements

- Software stack for PCIe hotplug functionality will be revised
  - PCI (SHPC based) and PCI Express (Native and ACPI) hotplug functionality will be migrated to work with the new Solaris hotplug framework.
  - Auto-Configuration operation
  - Hot Replacement
  - Dynamic Resource Re-balancing
  - Hotplug FMA support
    - Generic PCIe error handling will be enhanced to handle errors from the hotplug capable devices in a graceful manner
    - Card Removed, MRL opened, and Power-fault errors are no longer treated as catastrophic errors, rather treated as surprise removal events

Device Drivers

- Device drivers should work closely with hotplug framework to provide better hotplug experience
- New DDI hotplug interfaces for device drivers
  - Inquiry about hotplug capability
  - Register, receive and take action on various hotplug events such as surprise removal, dynamic resource re-balancing, hot replacement, power-fault
  - Inform hotplug framework of any device specific errors so that the framework or HPC driver can take appropriate action (e.g. To turn on Attention or Fault LEDs)

Inquiry hotplug capability

To check a specific device instance is hotplug capable or not. OS will return the answer based on the platform and device capabilities

```
int     ddi_hp_is_hotplug_capable(dev_info_t *dip)
```

Callback Register/Unregister interface

Register and unregister a device driver callback handler

```
int   ddi_cb_register(dev_info_t *dip,
      ddi_cb_flags_t  flags,
      ddi_cb_func_t cbfunc,
      void *arg1, void *arg2,
      ddi_cb_handle_t *ret_hdlp);

 int ddi_cb_unregister(ddi_cb_handle_t hdl);
```

flags:

DDI_CB_FLAG_INTR	Driver is IRM aware
DDI_CB_FLAG_HOTPLUG	Drive is Hotplug aware

Callback function

```
typedef int (*ddi_cb_func_t)(dev_info_t *dip,
        ddi_cb_action_t action,
        void *cbarg,
        void *arg1, void *arg2);
```

action:

DDI_CB_INTR_ADD	More available interrupts
DDI_CB_INTR_REMOVE	Fewer available interrupts
DDI_CB_HP_SURPRISE_REMOVAL	Surprise removal (async event)
DDI_CB_HP_QUERY_REMOVE_DEVICE	Query device can be removed
DDI_CB_HP_REMOVE_DEVICE	Remove device (sync event)
DDI_CB_HP_CANCEL_REMOVE_DEVICE	Cancel remove device
DDI_CB_HP_QUERY_STOP_DEVICE	Query device can be stopped
DDI_CB_HP_STOP_DEVICE	Stop - resource rebalance
DDI_CB_HP_START_DEVICE	Start – resource rebalance

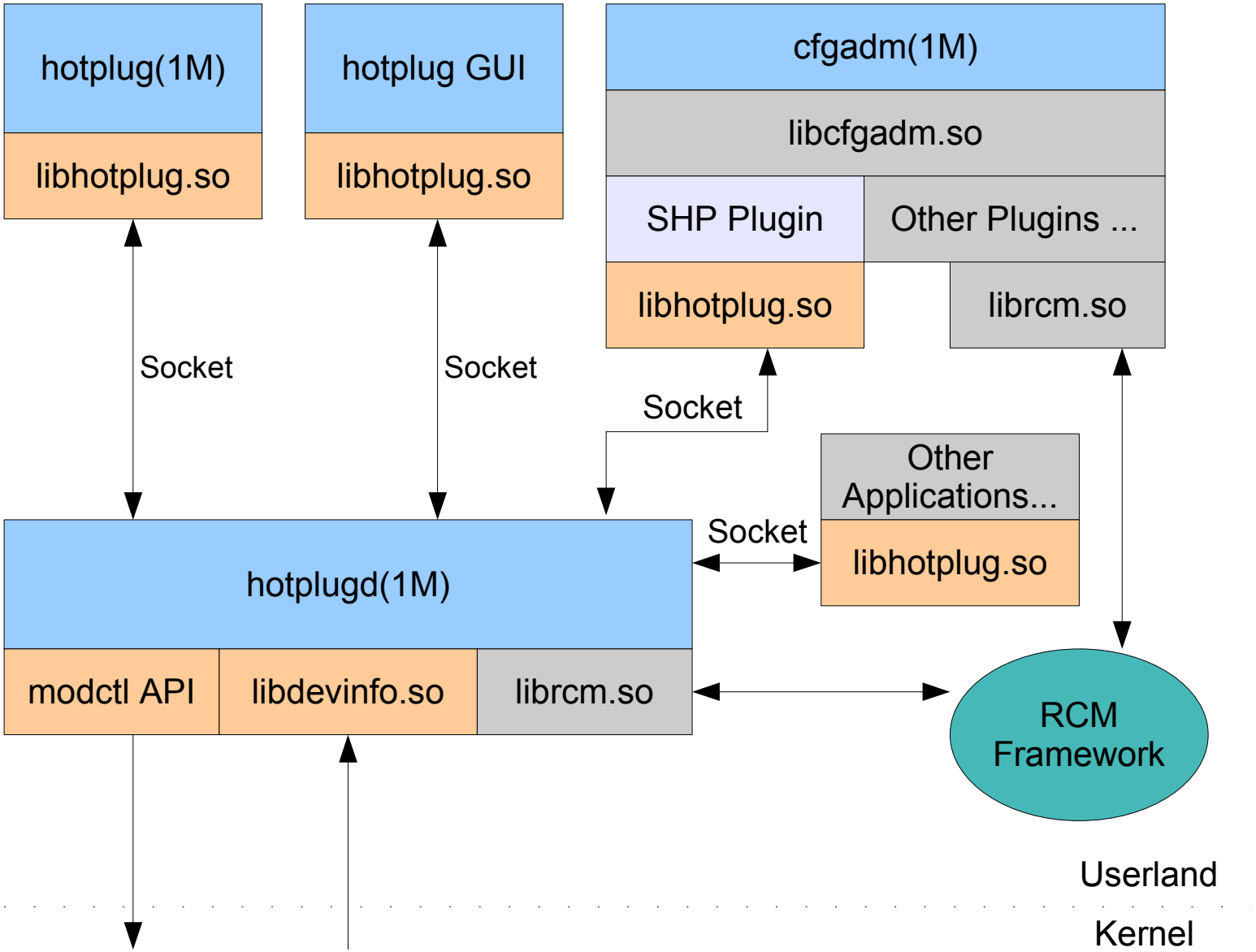
cbarg:

The cbarg parameter points to an action specific argument

Userland hotplug support

- hotplug admin tools
  - This project will deliver a new CLI and GUI based hotplug admin tools
  - The existing Cfgadm tool will be extended to work with the new hotplug framework
- hotplug library
  - It will provide generic management interfaces, so that generic or bus-specific user applications including GUIs can be written using these interfaces
- hotplug daemon
  - It is a general purpose user-level daemon to process and control the hotplug activities in userland. It will provide any required locking or serialize these requests before passing them to the kernel through modctl interface
- hotplug-aware user applications
  - They can use existing libcontract interfaces to register for the device contract events. This project will make some minor changes to libcontract library to support more device contract events

Userland Hotplug Overview



hotplug (1m)

- The hotplug command is used to configure hotplug connectors and ports
  - Commands
    - list; changestate; passthru;
  - Options
    - -a all; -v verbose; -f force; -q query;
  - Objects
    - <path> <physical connector> <virtual connector> <alias>

Implementation Strategy

- Phase-I (Nevada: Sept 2009; S10U9)
  - Basic hotplug framework features
  - Virtual hotplug support for PCIe devices
  - Physical hotplug support for PCI (SHPC based) and PCIe (ACPI, Native) devices
  - New hotplug CLI and plugin for cfgadm
- Phase-II (CY10 Q1)
  - Hotplug events through Device Contract and LDI frameworks
  - New hotplug DDI interfaces for device drivers
  - New PCI resource allocator and support for Dynamic resource re-balancing
  - Support hotplug FMA for PCIe devices
  - Support replacement for PCIe devices
- Phase-III (TBD)
  - Support for Express Card
  - Support for GUI admin tool

Implementation Strategy (2)

- Platforms Supported
  - SPARC:
    - Sun SPARC Enterprise M5000 Server (OPL)
    - Sun Blade T6300 (St.Paul)
    - Sun Blade T6320 (Glendale)
    - Sun Blade T6340 (Scotts dale)
  - x86/x64:
    - Sun Blade X6220 (Gemini)
    - Sun Blade X6350 (Wolf)
    - Sun Blade X8400 (Andromeda)
    - Sun Blade X8420 (Capella)
    - Sun Blade X8440 (Mira)
    - Sun Blade X6250 (Pegasus)
    - Sun Blade X6450 (Hercules)

Reference

- Hotplug Webpage
  - http://pcie.sfbay/hotplug
- Specs
  - PSARC/2008/181 – One Pager
  - PSARC/2008/181 - Overall Proposal
  - PSARC/2008/181 – Userland hotplug Software
- ARC cases
  - PSARC/1993/687 Hot Plugging
  - PSARC/1996/285 DR for CPU/Memory Boards
  - PSARC/1998/327 PCI Hotplug Support
  - PSARC/1998/460 RCM Framework
  - PSARC/1999/122 PCI Bus Resource Allocator
  - PSARC/1999/287 Solaris Embedded Fcode Interpreter
  - PSARC/2003/193 Solaris Contracts and restart agreements
  - PSARC/2005/375 PCI Hotplug Extensions for PCIe
  - PSARC/2007/290 Retire Agent for IO devices
- Aliases
  - mailto:ddi-hp-iteam@sun.com - Solaris Hotplug Framework Dev Team