

Common Multi-Protocol SCSI Target for SOLARIS

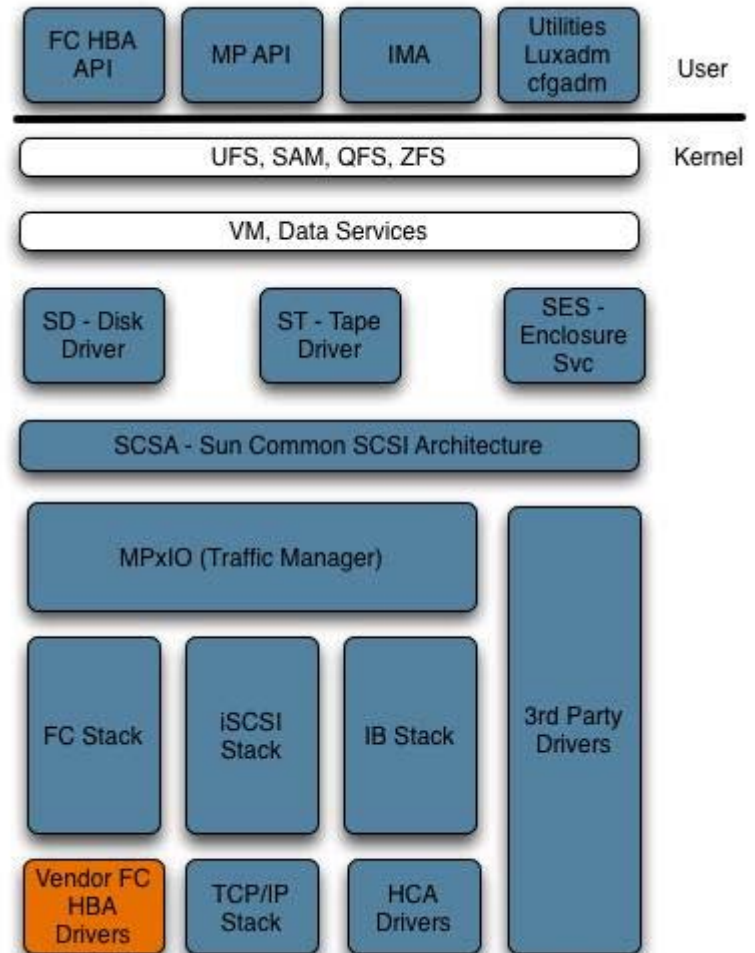
Presented by:

SUMIT GUPTA

Staff Engineer, SUN Microsystems Inc.

SCSI Initiator mode on Solaris

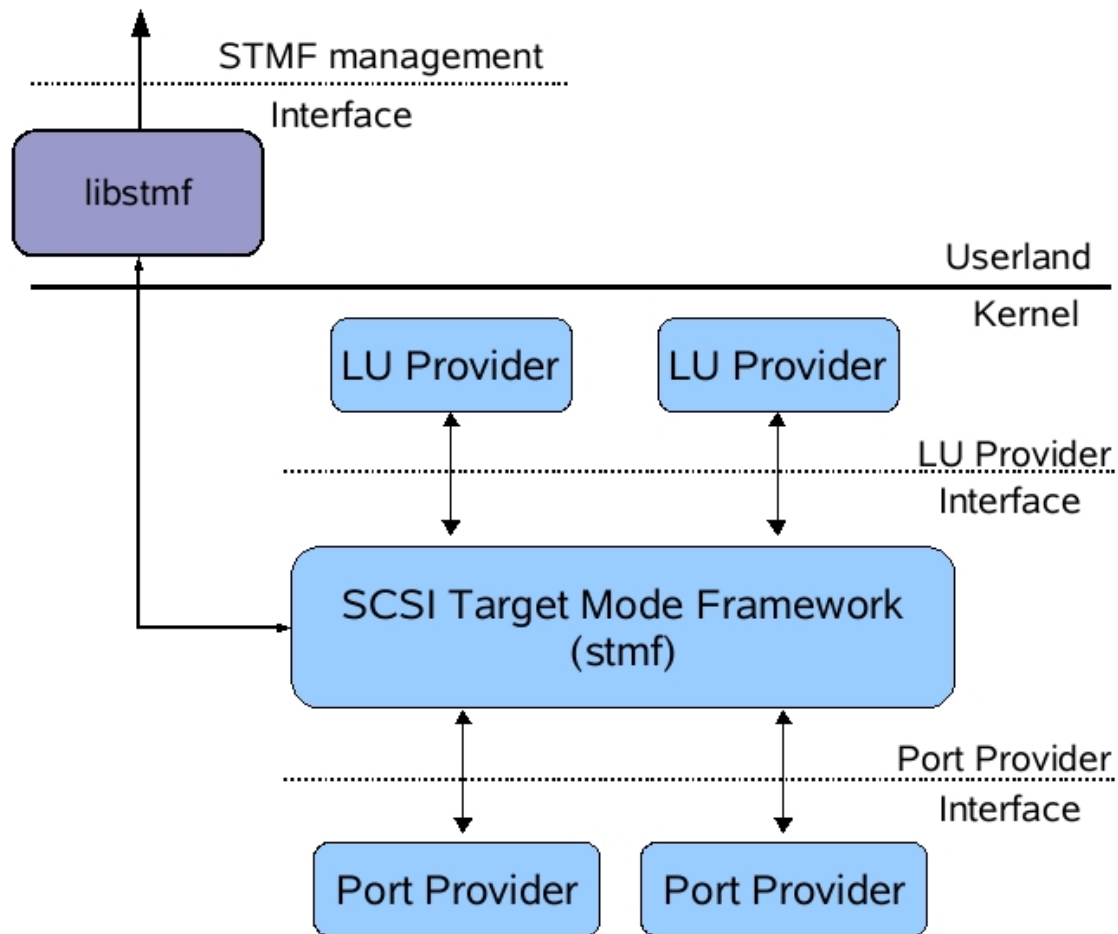
- SUN Common SCSI Architecture (SCSA).
- Transport protocols i.e. Fibre Channel, iSCSI, Parallel SCSI etc.
- SCSI Target drivers i.e. sd/ssd, st, ses etc.
- Available at opensolaris.org



Benefits of Solaris SCSI Target

- Open target mode platform. Complements open initiator platform.
 - Helps to create end to end solutions.
- Utilizes SOLARIS as a development platform.
 - Scalability.
 - DDI Compliance.
 - Rich APIs.
- Large variety of platform support.
 - X86, x64, sparc, small desktop, enterprise servers etc.

Project COMSTAR



Overview

- Solaris SCSI Target mode platform.
- Logical Unit Emulation code with no transport specific knowledge and vice versa.
- Extensive LUN Mapping and Masking operations.
- Handles resources related to SCSI task i.e. Task structure, DMA buffers etc.
- Provides execution context for SCSI Task.
- Manages execution of a SCSI task.
 - Calls appropriate entry points inside LU and Port during various SCSI phases.

Overview cont.

- Allows multiple data transfers in parallel for a given SCSI task.
 - One buffer could be with transport while another one is getting data from back end.
- Handles different cases of abnormal termination
 - Defines separate interfaces for completion vs. abort of a SCSI task.
 - A task can either be aborted or completed. Both of which have to finish before the task is de-allocated i.e. No race conditions.

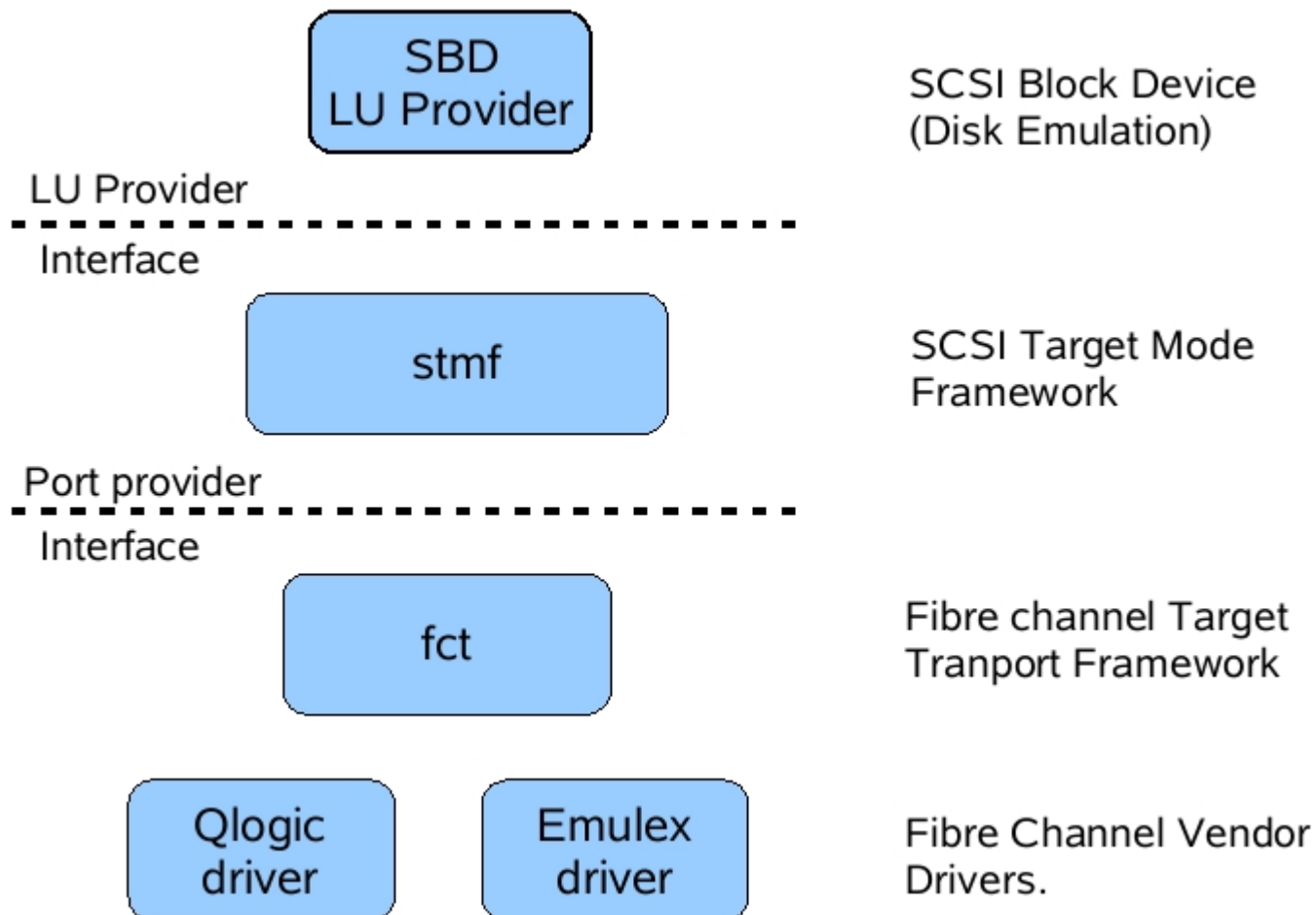
Overview cont.

- All phases of a task, including abort, use the same kernel thread for execution i.e. Again no race conditions.
- Clean 2-way handshake for startup and shutdown of both LUs and PORTs.
 - Framework terminates all I/Os for a LU and signals it for OFFLINE, LU ACKs and goes OFFLINE.
 - Framework signals PORT to go OFFLINE, PORT cleans up I/Os and ACKs.

Overview cont.

- Scalable design
 - Uses reader/writer locks instead of mutexes.
 - Uses atomic operations.
- Performance optimizations
 - Extensive dynamic caching of tasks, LU private data and data buffers.
 - Allows combining different operations into one operation e.g. Sending data and status together.

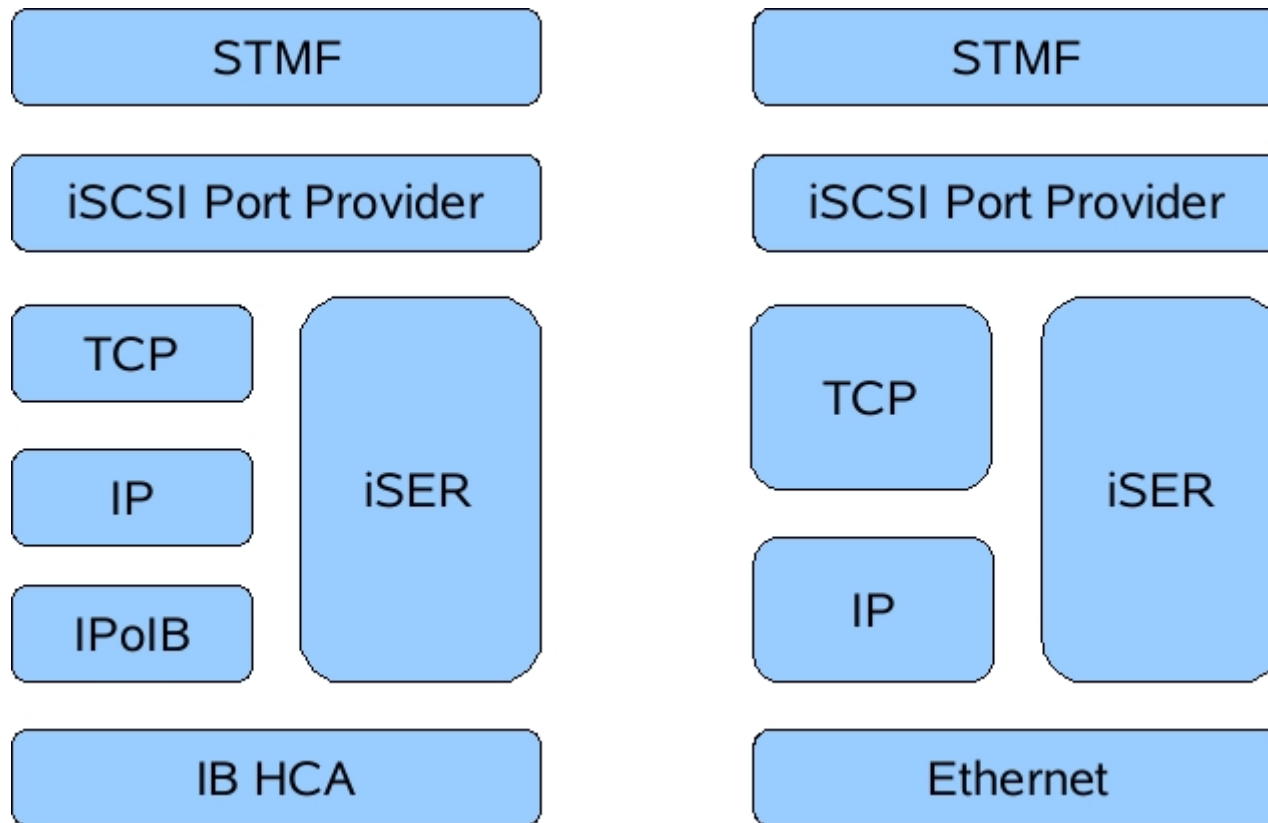
Fibre Channel Target Transport



FC Target Transport

- Handles complete FC discovery.
 - FCA drivers package ELSes and ship them to 'fct' for processing.
- Handles link events and session affecting ELSes.
 - Takes care of doing cleanup.
- Maintains state of all the logged in ports.
- Tracks commands and XCHG resources.
- All the non performance work is done in a single threaded manner (per port) to avoid race conditions.

iSCSI Target Transport



LU Provider Interface

- Public interface for developing LU Emulation.
- No knowledge of Transport protocol needed.
- Not dependent upon LUN mappings.
- Allows synchronous operation.
 - Threads can block while the LU is fetching data from the back end.
- Also allows asynchronous operation.
 - LU can maintain its own threads.
 - I/O is not done until explicitly indicated by LU.

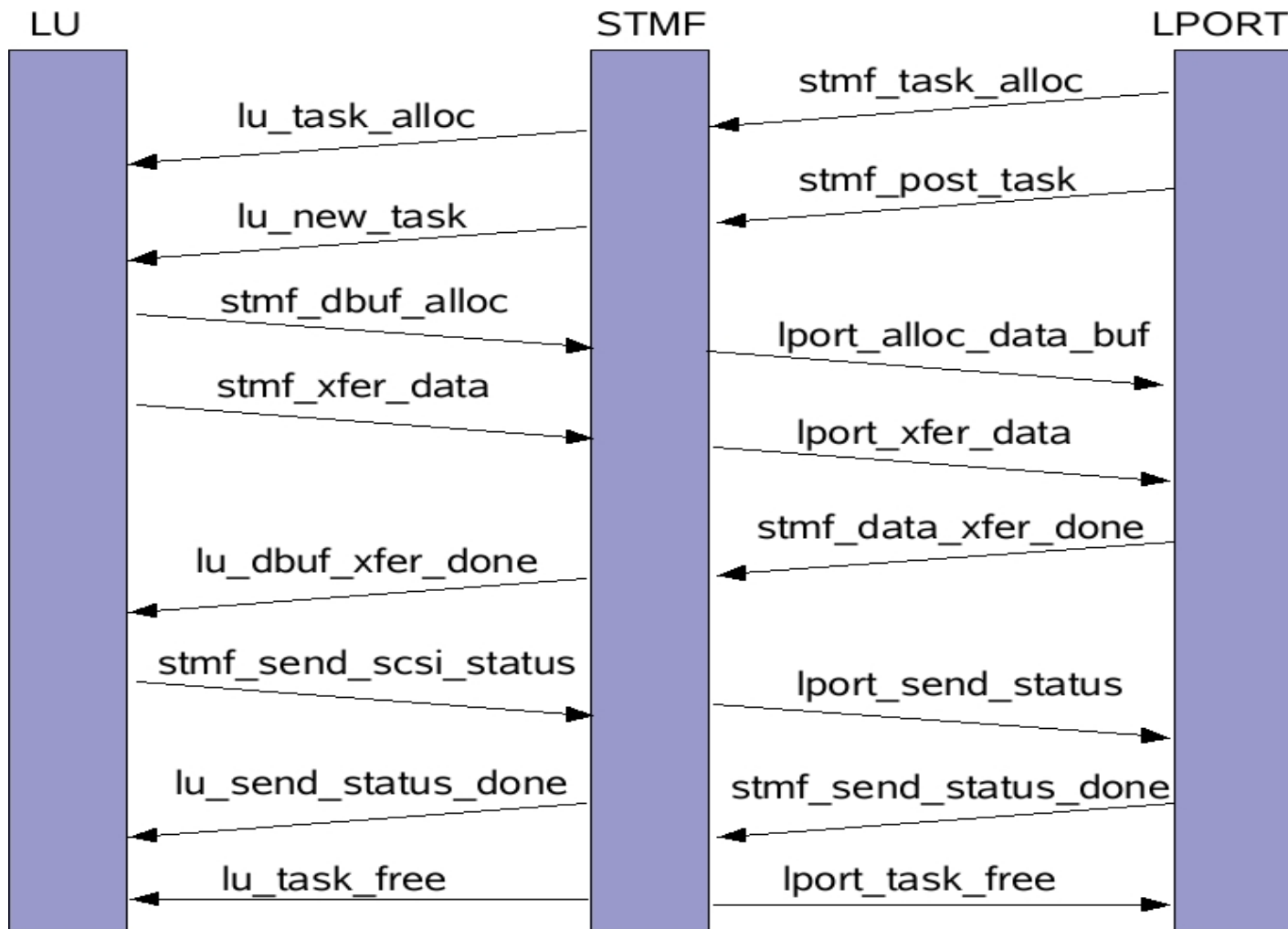
LU Provider Interface - cont.

- Allows LUs to have per task private data.
 - Caches the memory allocation for per task private data to increase performance.
- Allows for multiple data transfers in parallel.
 - LU can allocate a buf, fill it and ship it, allocate another buf, fill it (while the 1st one is getting shipped) and ship it before returning.

Port Provider Interface

- Public interface for implementing SCSI target ports.
- Protocol specific. No knowledge of the LUs behind the port.
- Responsible for providing buffers for data transfer using the `stmf_data_buf_t` abstraction.
- Callback threads from Port are not used for I/O execution.

I/O flow



Sample I/O flow

- The Target port receives a new SCSI task from the wire.
- The Port calls `stmf_task_alloc()` interface to get a `scsi_task_t` structure.
 - STMF keeps a cache of `scsi_task_t` on a per LU basis to increase performance.
- The Port initializes the `scsi_task_t` and posts the task to STMF by calling `stmf_post_task()`
- STMF receives a new `scsi_task_t` from the port and assigns an internal thread to it.



Sample I/O flow - cont.

- STMF calls `(*lu_task_alloc)(scsi_task_t *task)` entry point of the LU (using the assigned internal thread).
- LU allocates per task private data and initializes `task->task_lu_private` with this pointer.
 - This allocation will be cached by STMF. Next task will not call `lu_task_alloc()`.
- STMF then calls the following LU entry point to submit the task for processing.
 - `(*lu_new_task)(scsi_task_t *task, stmf_data_buf_t *initial_dbuf)`

Sample I/O flow - cont.

- The LU decodes the task. It finds out that the task needs data transfer.
- The LU calls `stmf_dbuf_alloc(task)` to get a data buffer.
- STMF calls `lport_alloc_data_buf()` entry point of the port to get a data buffer.
 - STMF links the data buffer with the task to do garbage collection once the task completes.
 - STMF also caches the data buffers to improve performance.

Sample I/O flow - cont.

- The LU initializes the data buffer with I/O direction, relative offset, data size etc. For host side READs, the LU also fills in the data.
- The LU then calls `stmf_xfer_data()` to start the data transfer for this data buffer.
- STMF tracks task state and then calls `(*lport_xfer_data)()` entry point of the port to initiate data transfer
- The LU can also ask the Port to send SCSI good status along with this transfer by setting a bit in the `stmf_data_buf_t` structure.

Sample I/O flow - cont.

- The LU can also inform STMF that it is done processing the CMD by setting a flag bit in `stmf_xfer_data()` call.
- The LU can return from the `(*lu_new_task)()` entry point at this time or get more data buffers and start transfer on them.
 - The max # of parallel data buffers supported is indicated by `task->task_max_nbuf`.

Sample I/O flow - cont.

- The Port completes the data transfer and initializes the transfer status and any residual counts in the `stmf_data_buf_t`
- The Port then calls `stmf_data_xfer_done()` to inform STMF about the completion of the data transfer.
- STMF posts the task to its worker thread for further processing.
- If the LU is not already done with the task, STMF calls (in the worker thread context) the `(*lu_dbuf_xfer_done)()` entry point of the LU.

Sample I/O flow - cont.

- The LU at this point can either continue with more data transfers or it can move into the status phase.
- To send status the LU initializes the transfer counters (for framework to calculate resid) and status information (scsi status, sense data etc.) in the `scsi_task_t` and calls `stmf_send_scsi_status()` interface.
 - At this point the LU can indicate that it is done with the I/O (unless it wants status confirmation) to the STMF.

Sample I/O flow - cont.

- STMF calls `(*lport_send_status)()` entry point of the port.
 - The port makes a copy of the sense data (if any).
- The port completes sending the status and calls back using the `stmf_send_status_done()` interface.
 - The port at this point has to indicate that it is done with the I/O. Otherwise the I/O enters abort processing mode.
 - If FCP_CONF is enabled, port will not call this interface until it receives FCP_CONF.

Sample I/O flow - cont.

- By this time if both LU and Port has indicated that the I/O is done, stmf starts the post processing.
- STMF frees any `stmf_data_buf_t` still attached to the I/O.
- STMF calls the `(*lport_task_free)()` entry point to tell the port to remove all references to this task.
- STMF then caches the task on an internal per LU cache list, unless an upper threshold of caching has been reached, in which case it calls `(*lu_task_free)()` to free LU private data and eventually free the `stmf_task_t` itself.

Abort Processing

- Starts by a call to `stmf_queue_task_for_termination()`
- Can be called anytime before indicating task complete.
- Results a call to `(*lu_abort)()` and `(*lport_abort)()` entry point.
- Needs explicit ack by a call to `stmf_abort_lport_done()` and `stmf_abort_lu_done()`
- Task is not freed until aborts are acked.

Links

- COMSTAR Project page.
 - <http://opensolaris.org/os/project/comstar/>
- Storage at SUN
 - <http://www.sun.com/bigadmin/hubs/storage/>
- IETF iSER Draft
 - <http://www.ietf.org/internet-drafts/draft-ietf-ps-iser-06.txt>



STORAGE DEVELOPER CONFERENCE

Where The Storage Development Community Connects

2007



COMSTAR

Presented by:

SUMIT GUPTA

SUMIT.GUPTA@SUN.COM