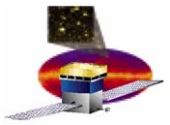
GLAST Large Area Telescope

Instrument Flight Software

Inter-Task Communication Functional Demonstration April 15, 2005

Stanford Linear Accelerator Center



Structure of the Inter-task Communications System

- The integrated inter-task communications system is built from three packages:
  - The ITC package defines the Inter-Task Communications standard:
    - ITC defines a common communications standard to unify communications between tasks on either the same or different CPUs.
    - It also provides the services to build up tasks that are capable of communicating according to the standard.
    - ITC is not itself a task, it's a toolbox.
  - The CTS package implements the ITC communications standard on 1553 transport hardware
    - It provides the service layer between applications (high level tasks) and the Command and Telemetry Database (CTDB) hardware driver.
    - It is constructed using the ITC toolbox, allowing CTS to present an ITC compliant (uniform) interface to the application layer tasks.
    - It implements tasks to communicate:
      - Spacecraft to CPU
      - CPU to spacecraft

Structure of the Inter-task Communications System (2)

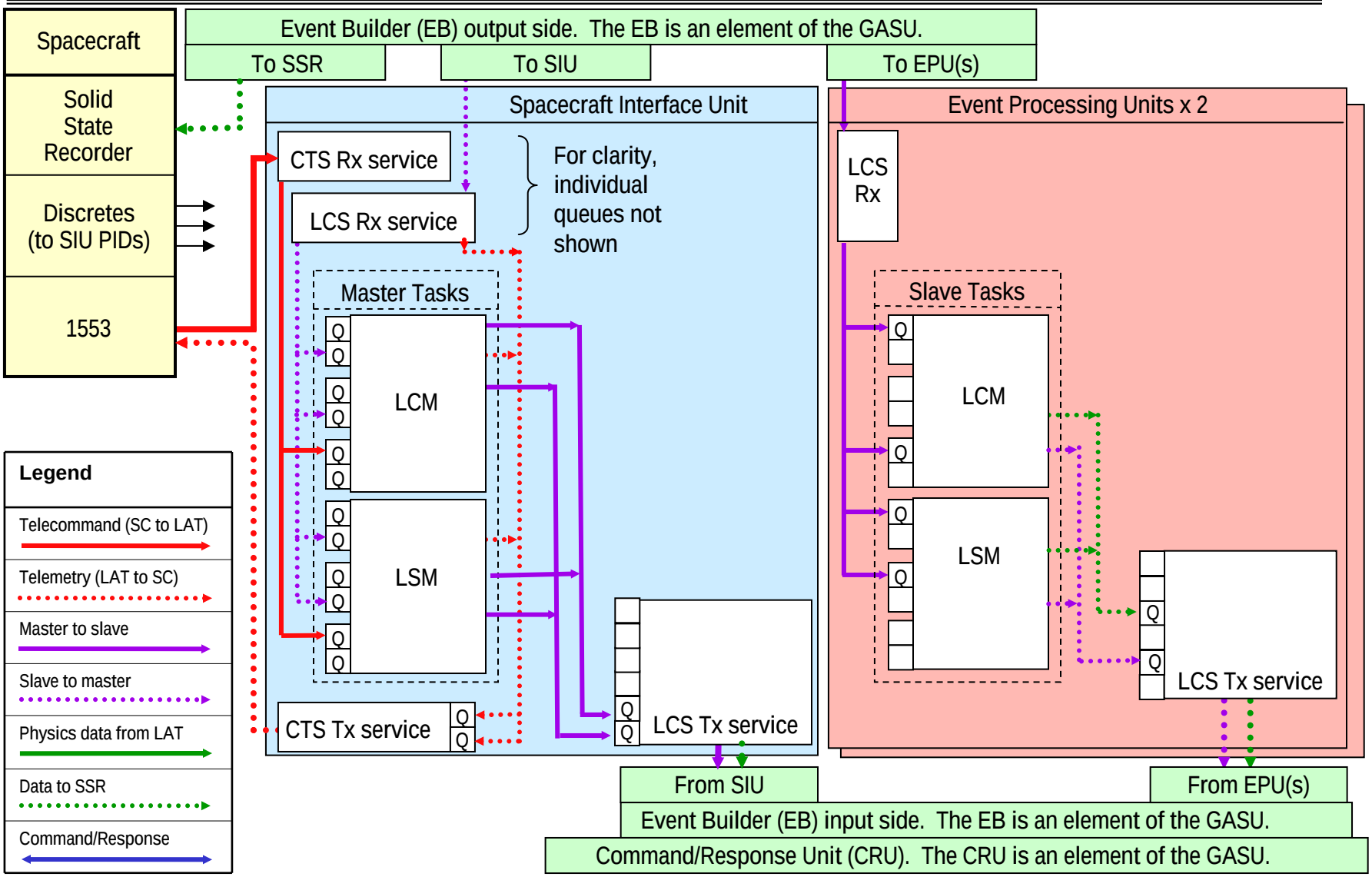
- The LCS package implements the ITC communications standard on LCB/LATp transport hardware:
  - It provides the service layer between applications (high level tasks) and the LCB hardware driver.
  - It is constructed using the ITC toolbox, allowing LCS to present an ITC compliant (uniform) interface to the application layer tasks.
  - It implements tasks to communicate:
    - CPU to CPU
    - CPU to SSR

Dimensions Exercised

- The ITC framework defines the following dimensions of inter-task communications:
  - Nodes: up to 8 supported -- includes CPUs, Spacecraft, SSR, broadcast
  - Tasks: 32 tasks maximum -- applications, services, and “lightweight tasks”
  - Queues:
    - Up to 256 queues per CPU
    - Maximum of 8 queues per task
    - Highest (META_HI) and lowest (META_LO) queues reserved to ITC. Remaining six queues available to application designer.
  - Communications protocols: dispatch and hardware
    - Dispatch protocols: RAW, CMD, APID
    - Hardware protocols for LATp fabric: NOACK, ACK

- For the demonstration:
  - Three nodes: SIU (node 0), EPU0 (node 1), EPU1 (node 2)
  - Two tasks: LCM (task 1), LSM (task 8)
  - Two queues: CTL_X (queue 4), CTL (queue 5)
  - Two hardware protocols: NOACK (protocol 2), ACK (protocol 3)
  - There will also be a SC (node 5), but the demonstration is focused on LAT internal communications (package LCS)

System Context for the Demo

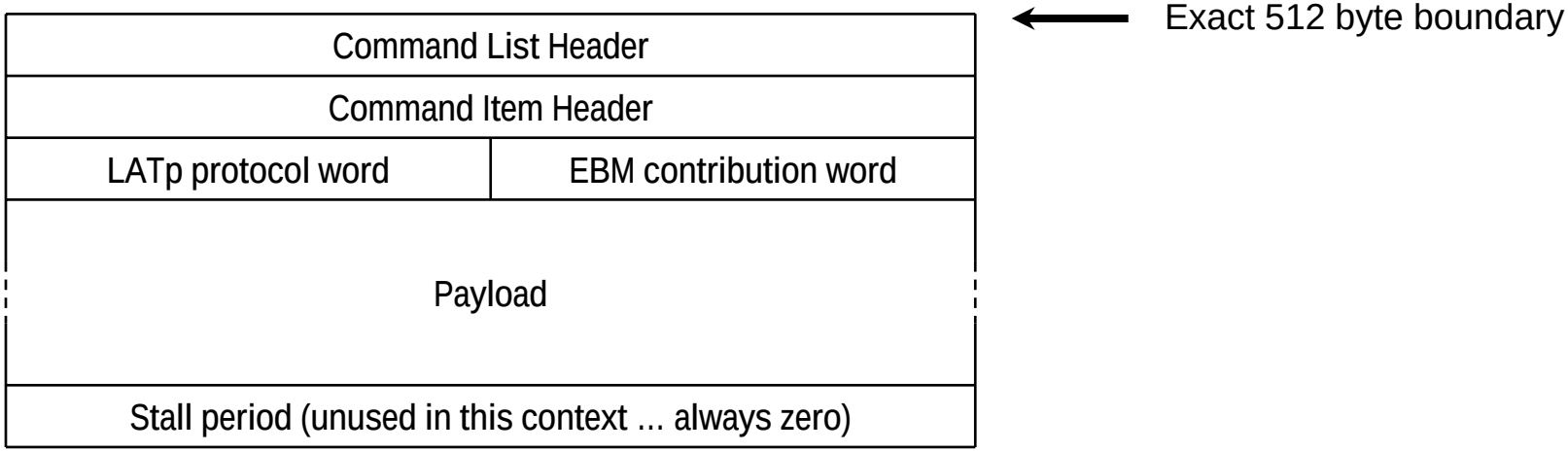


Communications Protocols Used by LCS

- LCS supports 2 communications protocols
  - Protocol 2 (NOACK): Unacknowledged (“fire and forget”)
  - Protocol 3 (ACK): Acknowledged
- Protocol 2
  - Uses
    - Talk to EPU in primary boot (primary boot code cannot acknowledge)
    - Talk to SSR (SSR cannot acknowledge)
    - CPU broadcasts
  - Characteristics
    - Messages limited to a single packet (2 kByte limit)
      - Except for SSR (special case)
    - Completion notification comes when message has “left the building”
    - Hence, no timeout behavior
- Protocol 3
  - Uses
    - General CPU to CPU communications
  - Characteristics
    - Messages limited to 128 MByte (i.e. no effective limit)
    - Completion notification comes from delivery of message to receiving task/queue on *destination* CPU
    - Full timeout behavior

Notes on LCB Hardware Constraints

- LCB hardware places quite onerous constraints on requests it will service
  - The memory layout must be very precise



  - LCB hardware limits a single request to (4096 – 128) bytes

- Among other services, LCS makes these constraints invisible to the applications writer
  - Messages are realigned correctly before transmission
  - Long messages are broken down into a series of packets on the sender and reassembled into the original message on the receiver

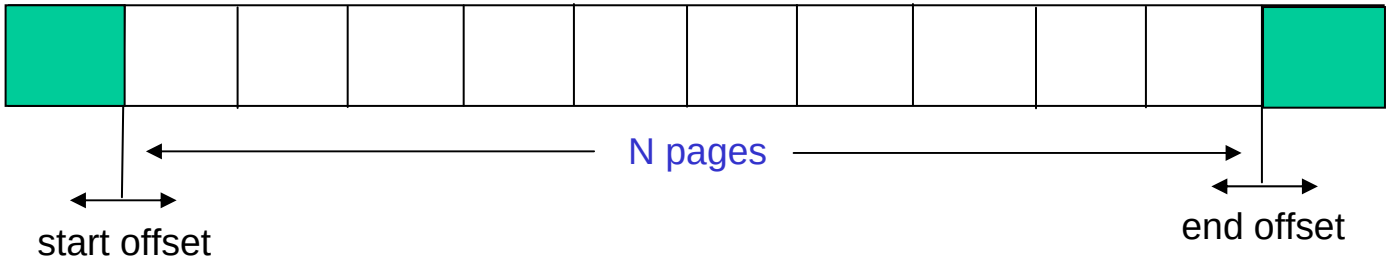
Sending A Test Message (1)

- The test suite must synthesize messages covering a wide parameter space
  - Both hardware protocols
  - A range of lengths
  - A range of alignments
  - Verification of packet sent against packet received
  - Verification of NON-overwrite of “un-owned” memory
- The messages can then be sent from any source to any destination

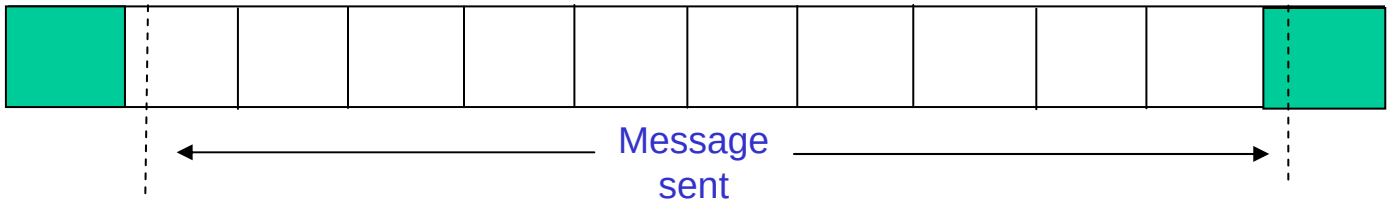
- All the tests demonstrated today use the same message description
  - A base of N pages (a page is 512 bytes)
  - A byte offset (positive or negative) on the start of message
  - A byte offset (positive or negative) on the end of message

Sending A Test Message (2)

- What happens
  - (N+2) pages of memory are allocated
  - The buffer is painted with predictable pattern



- What gets sent
  - The section of the buffer between the requested offsets



- When message transmission completes
  - Check that buffer still contains the predicted pattern
  - Free the buffer

Receiving A Test Message

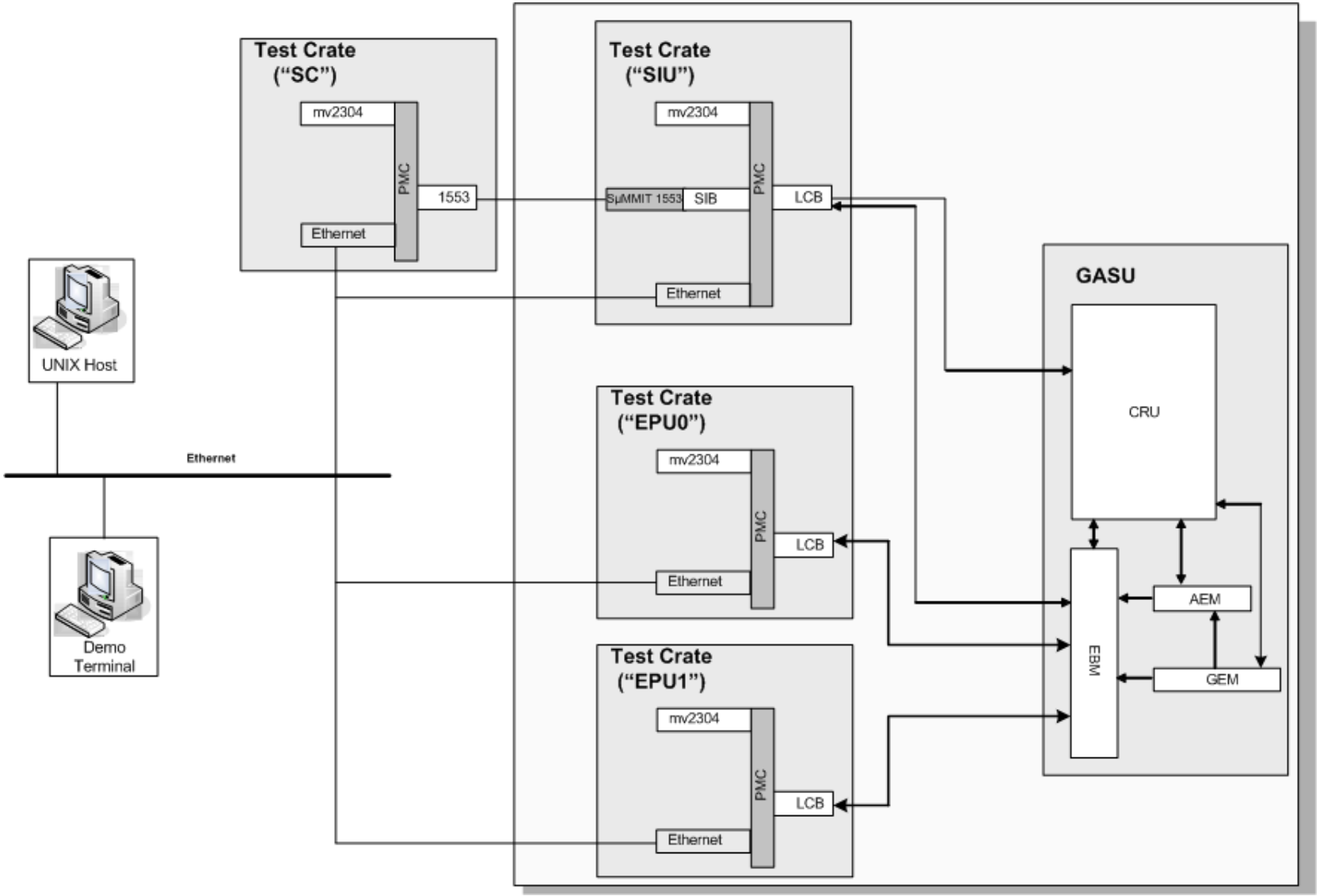
- When message first received
  - Allocate a receive buffer (in fact it's slightly over-allocated)
  - Prepaint buffer with fixed pattern (**`0xdeafbede`**)
  - Overwrite buffer with incoming message
- When message completes (may be many packets)
  - Forward to destination task/queue
- At receiving task/queue
  - Check that buffer contains predicted pattern (NOT **`0xdeafbede`**)
  - Check that area *past* the message *does* contain **`0xdeafbede`**
  - Free the buffer

- Every message sent today will be subjected to these verifications
  - Any flaw will result in a printed error message
  - Exceptions for raw performance tests and deliberately broken messages

Demonstration Technique

- Terminal windows will be opened to monitor the Spacecraft crate and CPU crates used in the demo
  - Routines from the LCS test suite will be invoked from the command line to exercise various pieces of the inter-task communications system
  - Results will be displayed at the terminal windows

Hardware Context



On to the Demonstrations

- Send single message from arbitrary source node/task to arbitrary destination node/task/queue with arbitrary protocol
  - Audience participation invited
- Pratfall tests
  - Bad destination node
  - Bad destination task
  - Bad destination queue
- Speed test
  - Send 4 large messages back-to-back and time them
- Priority test
  - Send 1 short, high priority message through long, multi-part low priority message
- Send a long series of messages using protocol 2 (NOACK)
  - Takes about 1 minute to complete

And More Demonstrations

- Corruption tests
  - Use low-level LCB code to write errors into the outbound data, mimicking a hardware corruption:
    - Broken sequence number
    - Broken transaction ID
    - Broken length
    - Stalled transmission
- Torture test (the hardware, not me)
  - Audience participation invited to select:
    - 4 sources on SIU to 4 destinations on EPU0 or EPU1
    - 4 sources on EPU0 to 4 destinations on SIU or EPU2
    - 4 sources on EPU1 to 4 destinations on SIU or EPU1
  - Transmit a long series of messages for each combination of source and destination (about 15 minutes)
  - Review statistics for this large block of exchanged messages
- Maybe a surprise demo thrown in!

Forward Work

- Complete the test suite
  - Ensure that reconstruction of *hardware* fragmentation works
  - Ensure all timeout race conditions are accounted for
- Implement the CPU -> SSR service
  - Easier
    - SSR can't talk back! (no protocol 3)
    - Only one queue (no point and *dangerous* to interleave messages)
  - Harder
    - Packetization must follow CCSDS rules (14 byte headers … aarrghh)
  - Left to last because
    - No immediate customers
    - Only clumsy test methods available (bring on the VSC)
- Implement broadcast
  - May be omitted (no identified customers)
- Write the manual