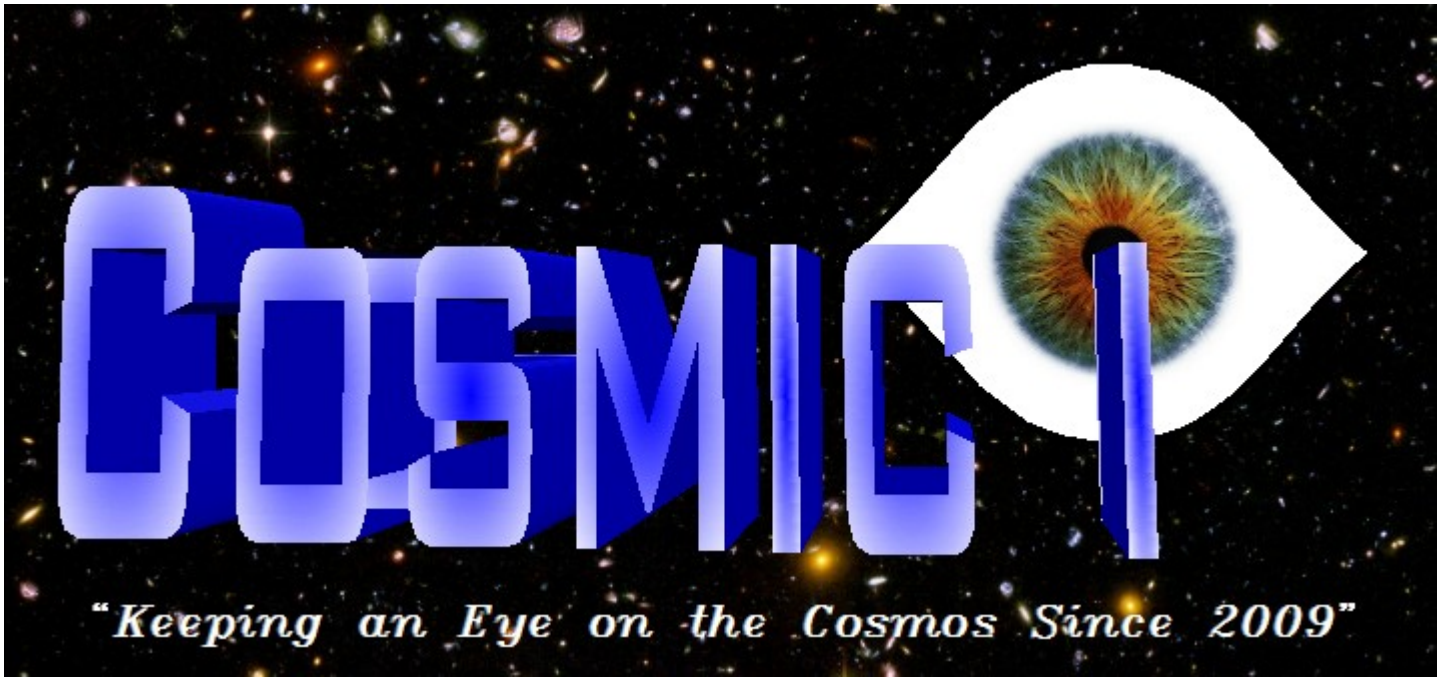
COSMICi Server
Python Code Workshop



Michael P. Frank
FAMU Dept. of Physics &
FAMU-FSU College of Engineering

Purpose of This Presentation

- Familiarize students with the architecture of the COSMICi project's present (incomplete) central server application.
  - A framework of numerous useful low-level modules already exists.
    - Further work should fit in with / build on top of these.
- Point the way towards some design directions for completing the full server, with features like:
  - Analysis & visualization of incoming pulses
  - Database storage of pulse data for offline analysis
  - (Eventually) communication between multiple server nodes in a peer-to-peer network.

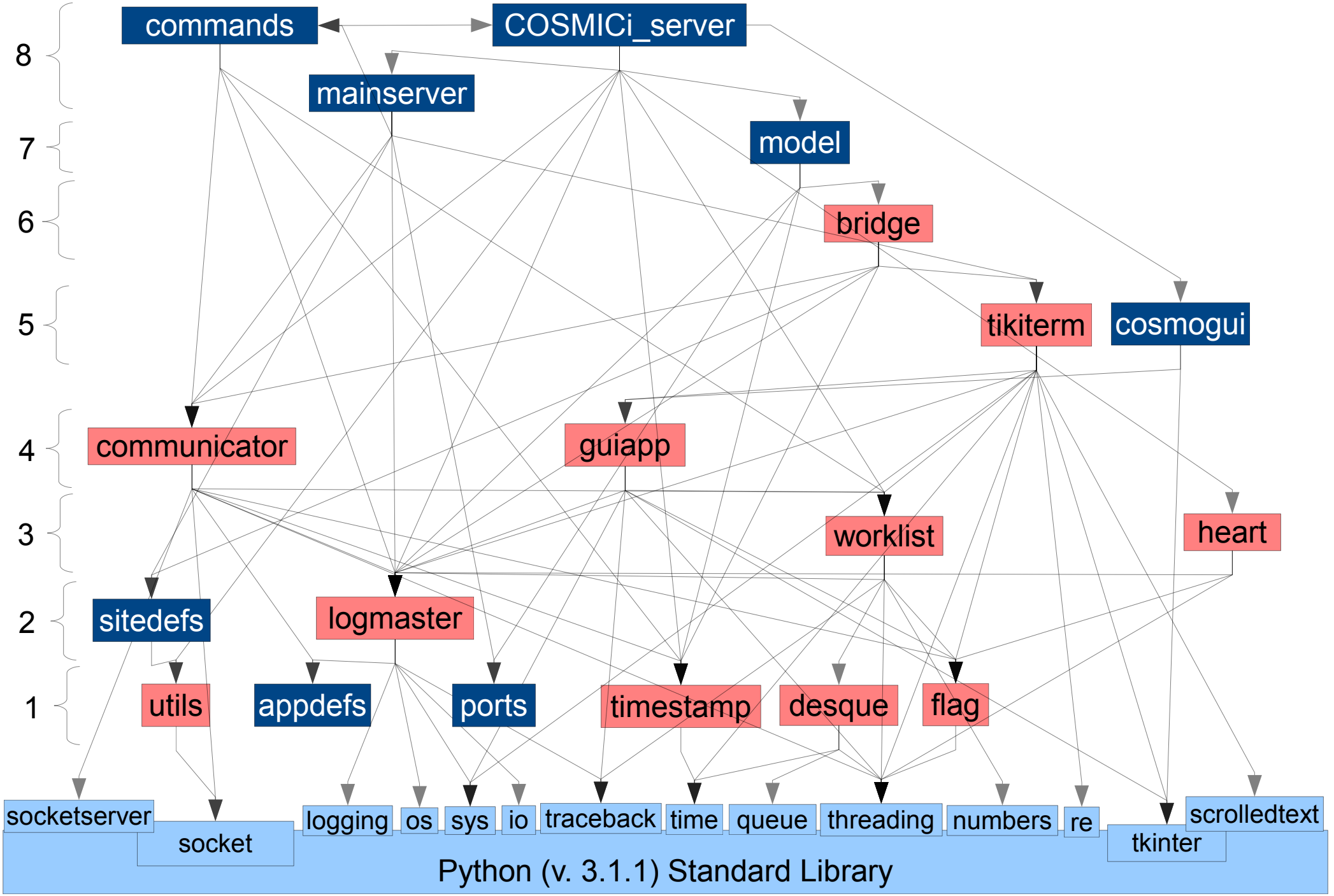
List of Modules in Present Code

- Level 8:
  - COSMICi_server
  - commands
  - mainserver
- Level 7:
  - model
- Level 6:
  - bridge
- Level 5:
  - cosmogui
  - tikiterm

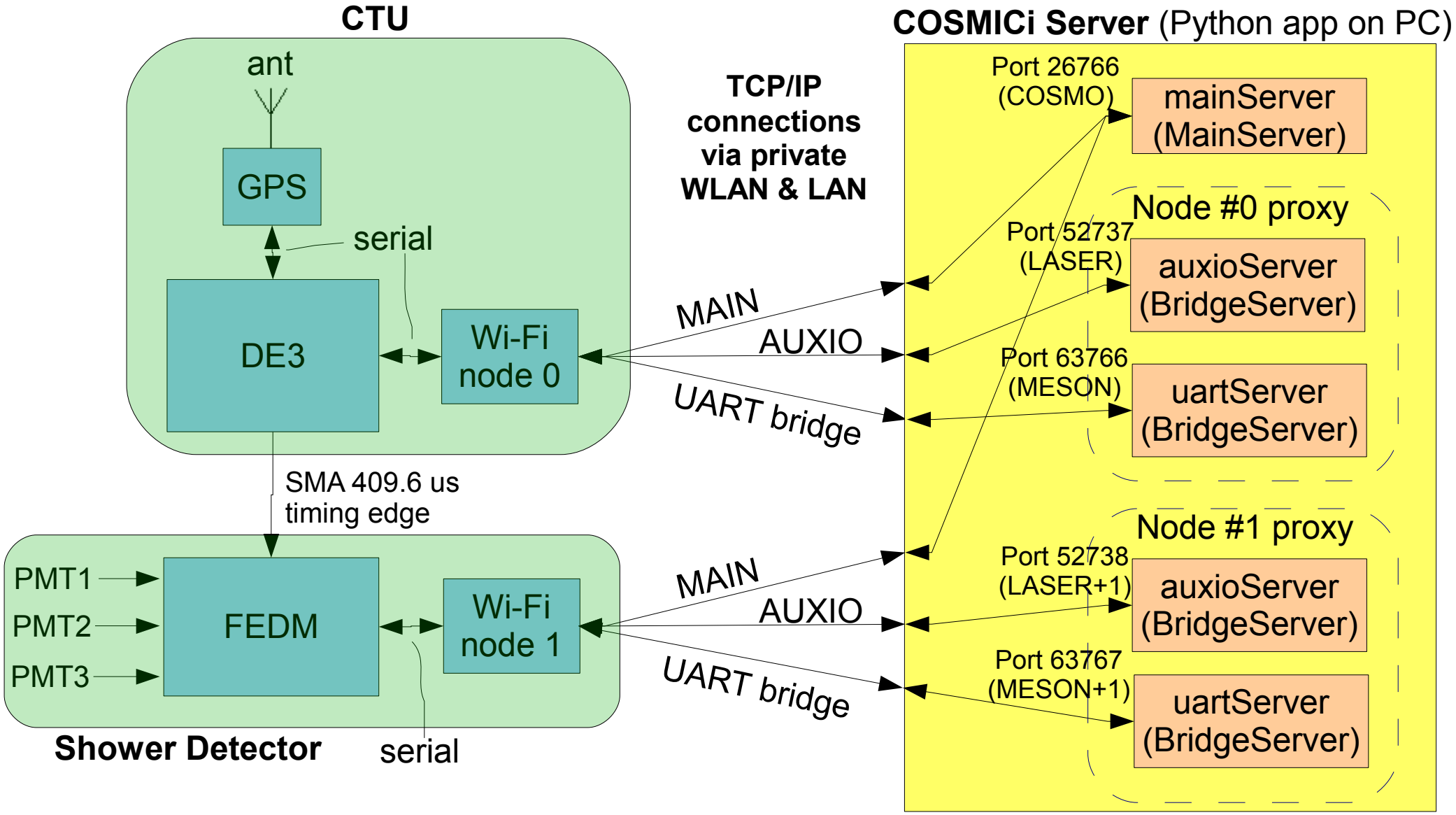
~~pinger~~
currently unused

- Level 4:
  - guiapp
  - communicator
- Level 3:
  - heart
  - worklist
- Level 2:
  - sitedefs
  - logmaster
- Level 1:
  - appdefs
  - ports
  - timestamp
  - utils
  - desque
  - flag

COSMICi Server Module Hierarchy (approx.)

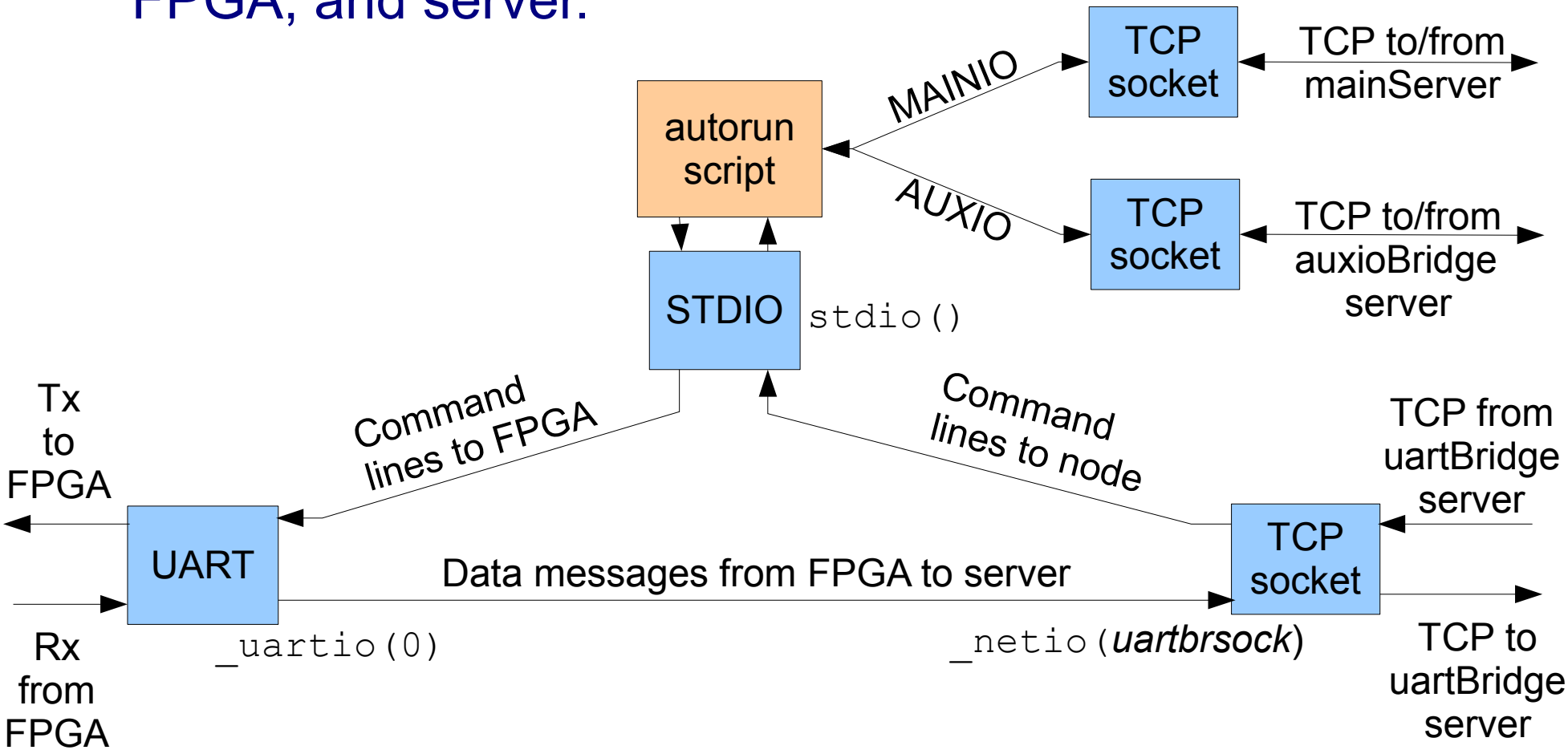


Communication in COSMICi (Stage 1 Prototype)

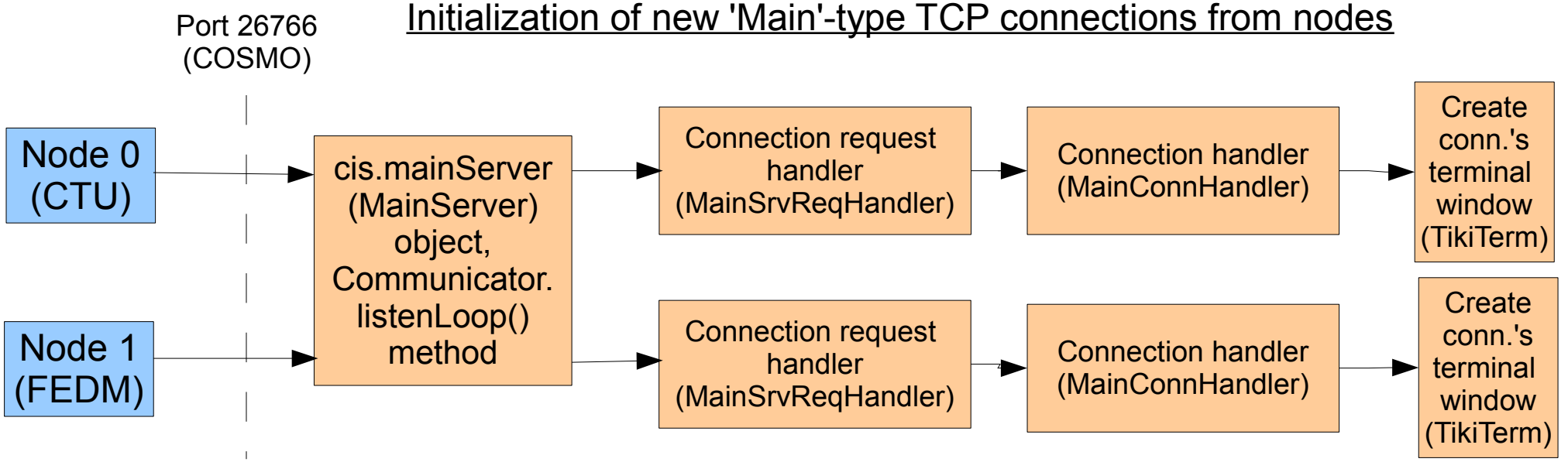


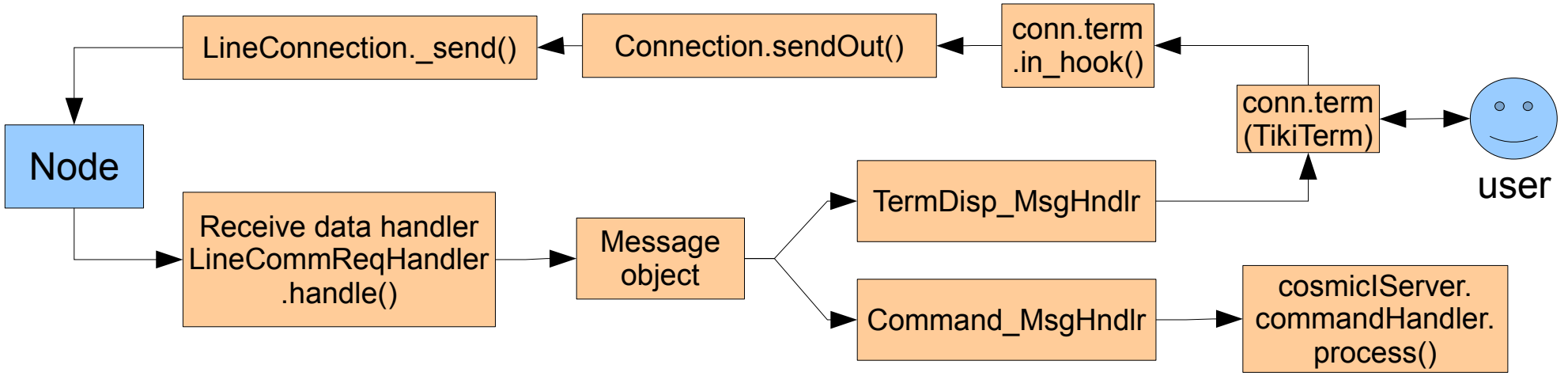
Wi-Fi Module Configuration

- This is a custom bridging configuration which I call the “Trefoil” configuration.
  - It facilitates 3-way communication between Wi-Fi script, FPGA, and server.



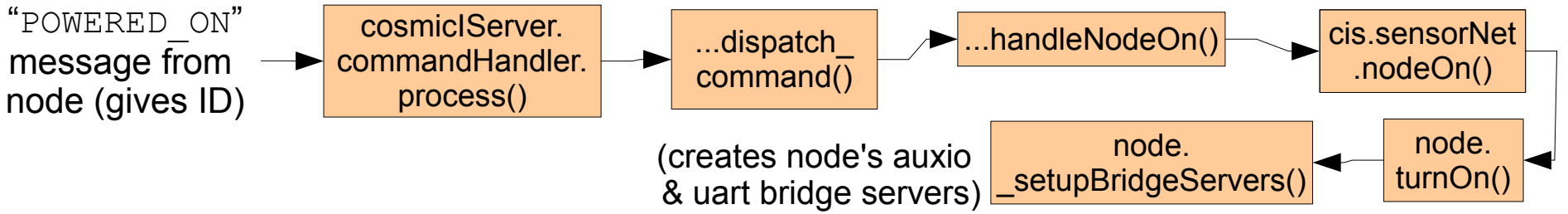
MainServer communications



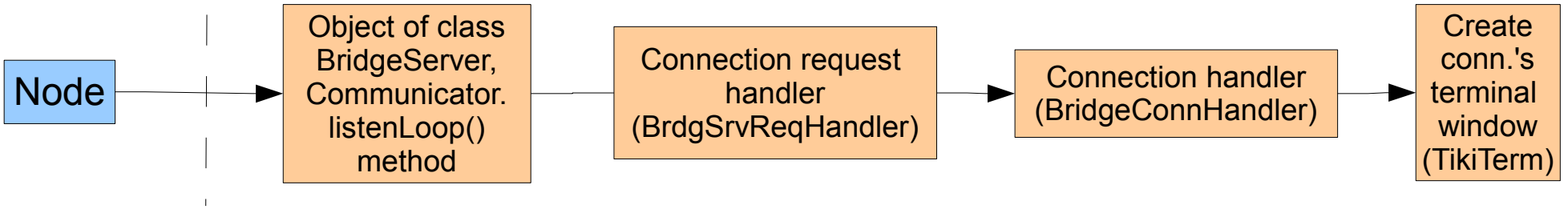


BridgeServer communications

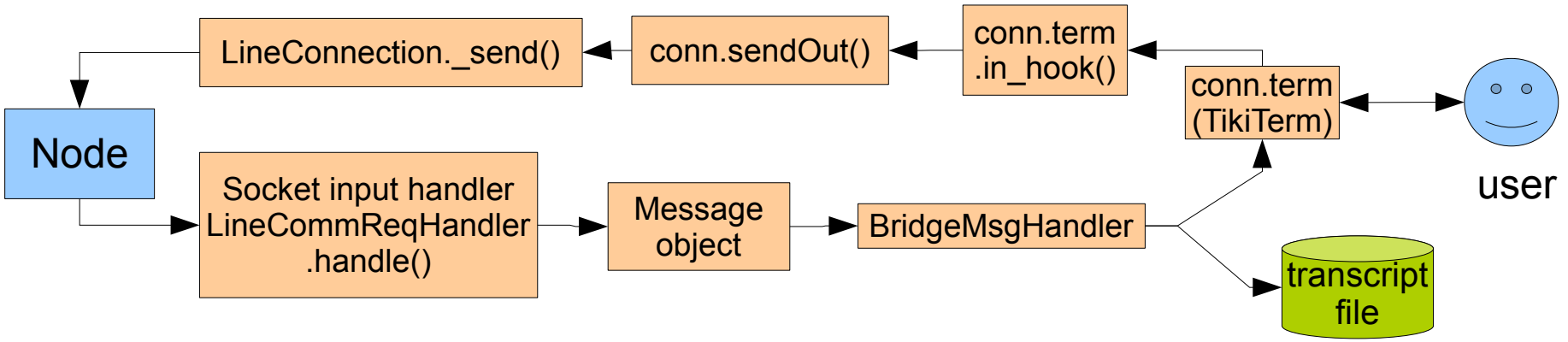
Creation of new 'bridge'-type TCP servers



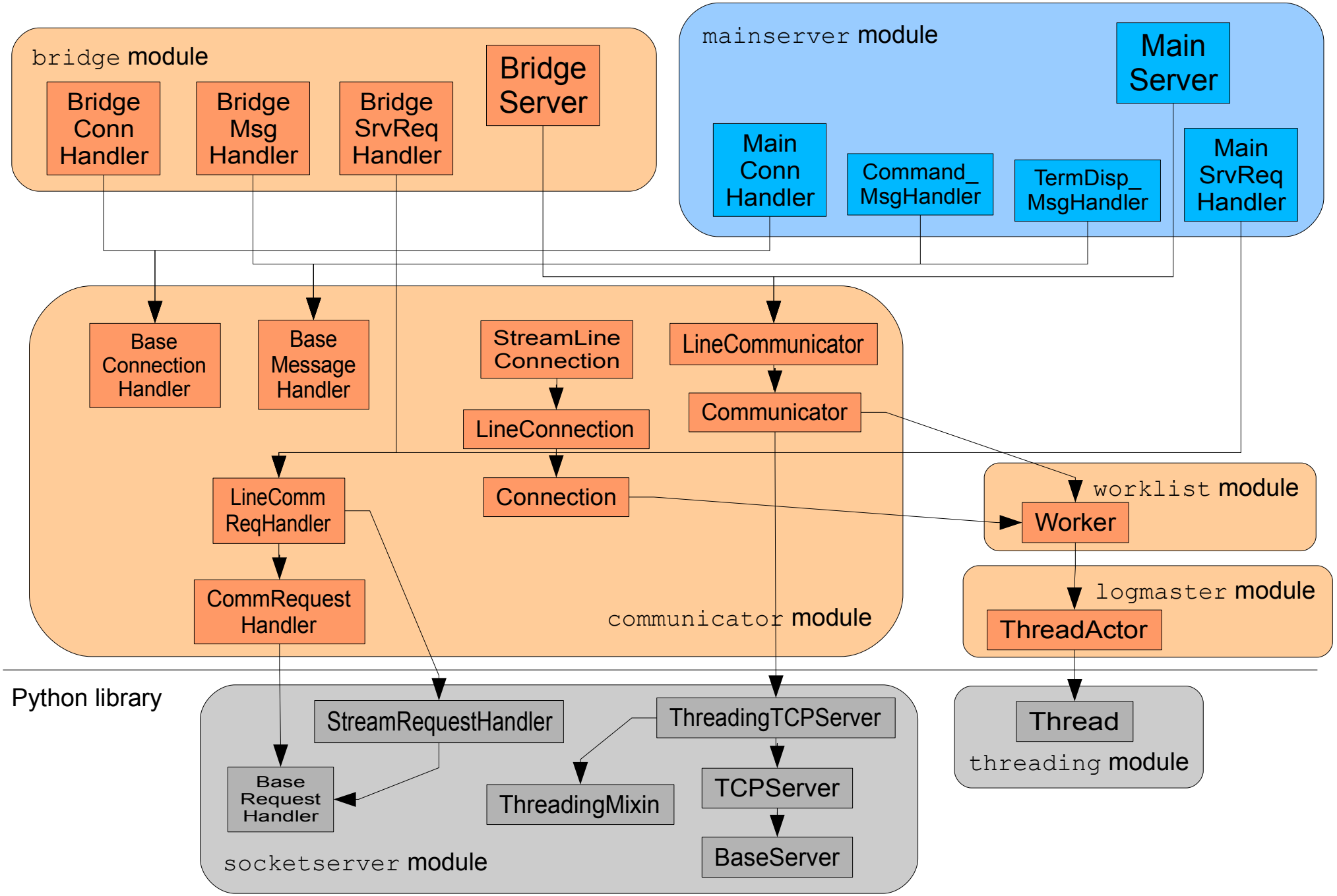
Initialization of new 'bridge'-type TCP connections from nodes



I/O on 'bridge'-type Server Connections



Communication Class Hierarchy



Communication Class Hierarchy is Pretty Complex & Baroque

(Out of date) map of attribute/method inheritance for the server class hierarchy

	BaseServer	TCPServer	ThreadingMixin	ThreadingTCPServer	Communicator	LineCommunicator	MainServer	BridgeServer
.timeout	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
.address_family	-	TCPServer	-	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.socket_type	-	TCPServer	-	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.request_queue_size	-	TCPServer	-	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.allow_reuse_address	-	TCPServer	-	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.defaultRole	-	-	-	-	Communicator	LineCommunicator	MainServer	LineCommunicator
.defaultAddr	-	-	-	-	Communicator(None)	Communicator(None)	MainServer	Communicator(None)
==> .__init__()	BaseServer	TCPServer	BaseServer	TCPServer	Communicator	LineCommunicator	LineCommunicator	BridgeServer
.server_activate()	pass	TCPServer	pass	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
==> .serve_forever()	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
==> .shutdown()	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
==> .handle_request()	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
._handle_request_noblock()	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
.handle_timeout()	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)
.verify_request()	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)	BaseServer(pass)
.process_request()	BaseServer	BaseServer	ThreadingMixin	ThreadingMixin	ThreadingMixin	ThreadingMixin	ThreadingMixin	ThreadingMixin
.server_close()	BaseServer(pass)	TCPServer	BaseServer(pass)	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.finish_request()	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
.close_request()	BaseServer(pass)	TCPServer	BaseServer(pass)	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.handle_error()	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
.get_request()	(undef)	TCPServer	(undef)	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.server_bind()	-	TCPServer	-	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.fileno()	-	TCPServer	-	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.setup()	-	-	-	-	(undef.)	(undef.)	MainServer	(undef.)
._announce()	-	-	-	-	Communicator	Communicator	Communicator	Communicator
._addConn()	-	-	-	-	Communicator	Communicator	Communicator	Communicator
.conNum()	-	-	-	-	Communicator	Communicator	Communicator	Communicator
==> .addConnHandler()	-	-	-	-	Communicator	Communicator	Communicator	Communicator
==> .startListening()	-	-	-	-	Communicator	Communicator	Communicator	Communicator
==> .sendAll()	-	-	-	-	Communicator	Communicator	Communicator	Communicator
==> .start()	-	-	-	-	-	-	MainServer	BridgeServer
==> .send()	-	-	-	-	-	-	-	BridgeServer
.server_address	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
.RequestHandlerClass	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
.__is_shut_down	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
.__serving	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer	BaseServer
.socket	-	TCPServer	-	TCPServer	TCPServer	TCPServer	TCPServer	TCPServer
.role	-	-	-	-	Communicator	Communicator	Communicator	Communicator
.connHandlers	-	-	-	-	Communicator	Communicator	Communicator	Communicator
.conns	-	-	-	-	Communicator	Communicator	Communicator	Communicator
.ncons	-	-	-	-	Communicator	Communicator	Communicator	Communicator
._wlock	-	-	-	-	Communicator	Communicator	Communicator	Communicator
.name	-	-	-	-	-	-	-	BridgeServer
.node	-	-	-	-	-	-	-	BridgeServer

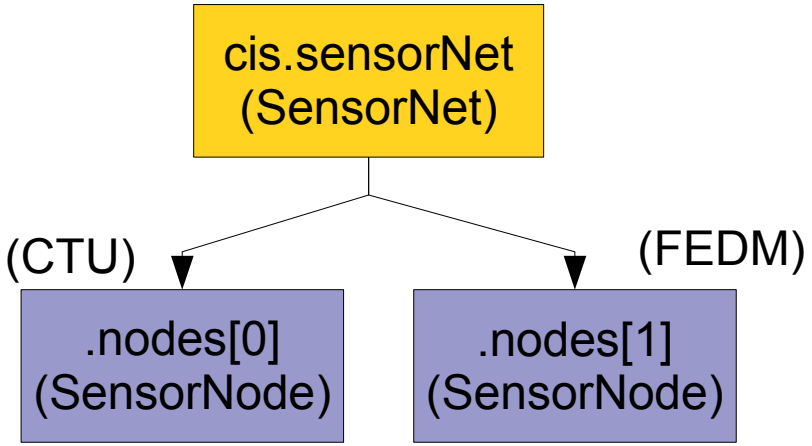
.class_variable	DefiningClass	InheritedFromClass	OverridingClass
.instance_method()	DefiningClass	ExtendingClass	InheritedFromClass	OverridingClass
.instance_variable	InitializingClass		InheritedFromClass

New Software Components Needed

- New classes needed in object model of sensor net (`model.py`):
  - `WiFi_module`, `Sensor`, `CTU_GPS_Sensor`, `FEDM_Sensor`
- New data structures for use by Sensor models:
  - `ctu` module (file `ctu.py`)
    - Classes `GPS_Tick`, `CTU_Run`
  - `fedm` module (file `fedm.py`)
    - Classes `Pulse`, `Shower`, `FEDM_Run`
- New modules to handle various specialized processing tasks:
  - `fitter` module (file `fitter.py`) – Pulse fitting.
  - `filter` module (file `filter.py`) – Shower filtering.
  - `geometry` module (file `geometry.py`) – Shower geometry.
  - `astronomy` module (file `astronomy.py`) – Galactic coordinates.
- New visualization classes (file `visualize.py`):
  - `PulseDisplay`, `EventAnimator`, `MapDisplay`

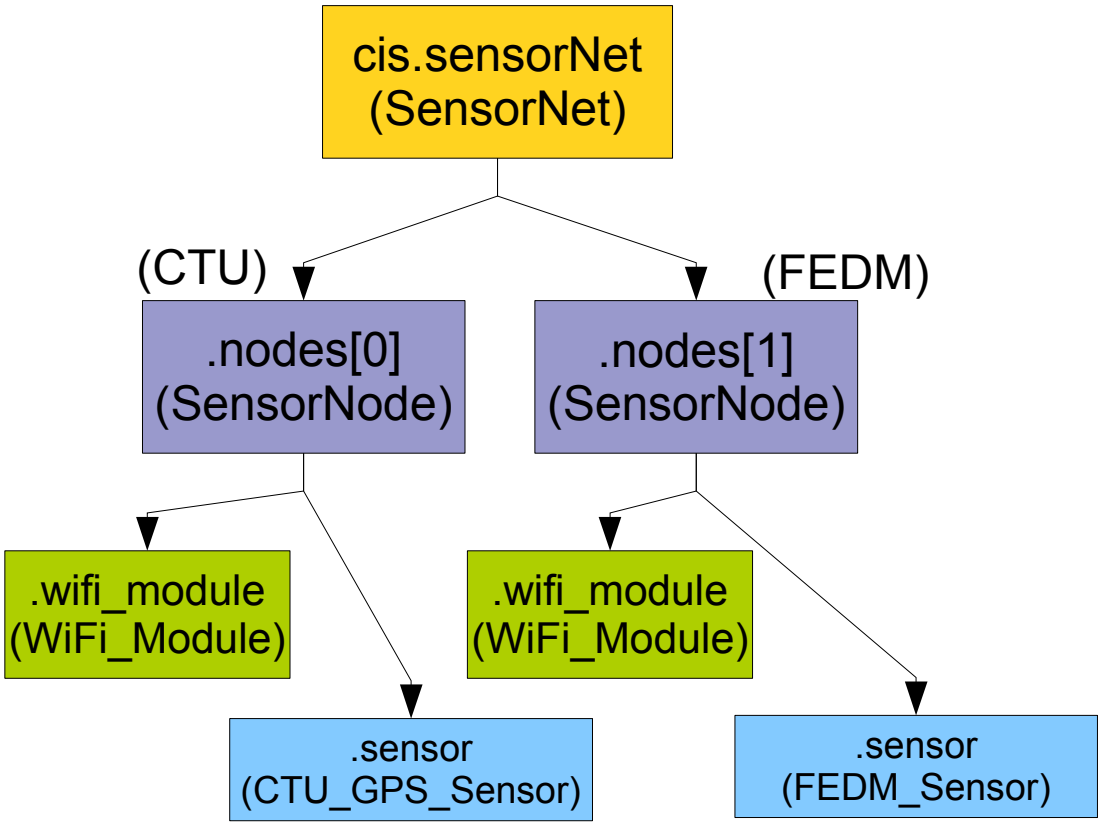
Improved Object Model of Sensor Network

Current Model:



The present model doesn't distinguish between the CTU & the FEDM nodes at all.

A Better Model:



The idea behind the new model is:

- Commands to control the EZURiO Wi-Fi script go through the WiFi_Module class. These commands are identical for both nodes.
- Commands to our Nios firmware running on the Stratix FPGA go through the .sensor object, which may be of either of two classes, depending on which subsystem it is. Both are subclasses of Sensor, which handles common functions (STOP, RESET, etc.)

New Data Structures Needed

- For use by `CTU_GPS_Sensor` model/proxy:
  - `ctu` module (file `ctu.py`)
    - `GPS_Tick` class – For storing information associated with a single GPS "tick" (PPS edge timing capture & associated NMEA data).
    - `CTU_Run` class – Encapsulates a database of all tick information received from the CTU since the start of the run.
- For use by `FEDM_Sensor` model/proxy:
  - `fedm` module (file `fedm.py`)
    - `Pulse` class – For storing data associated with a single PMT pulse event.
    - `Shower` class – For storing information about a candidate cosmic-ray shower event.
    - `FEDM_Run` class – Encapsulates a database of all shower information received from FEDM since the start of the run.

New Data Analysis Modules Needed

- New modules to handle various specialized processing tasks:
  - `fitter` module (file `fitter.py`) – Pulse fitting.
    - `PulseFitter` class – Pulse shape reconstruction
  - `filter` module (file `filter.py`) – Shower filtering.
    - `ShowerFilter` class – Filters for candidate shower events.
  - `geometry` module (file geometry.py)
    - `AngleFinder` class – Infer azimuthal heading from timing data.
  - `astronomy` module (file `astronomy.py`)
    - `EventMapper` class – Works from azimuthal heading & event time to obtain Galactic angular coordinates.

New Visualization Modules Needed

- New `visualize` module (file `visualize.py`)
  - Class `PulseDisplay` – Graphically charts raw pulse data received from PMT sensors.
  - Class `EventAnimator` – Displays a rotatable 3D visualization of shower front sweeping through site.
  - Class `MapDisplay` – Displays an elliptical sky map in galactic coordinates with event markers.

An interactive map display based on Google Sky would also be desirable, but more research is needed to figure out how to integrate such a feature into the application architecture.

Summaries of all existing modules

Lowest-Level Modules (level 1)

- These do not use/reference any other custom modules, only modules in the Python standard library.
- General-purpose modules:
  - `flag` – Waitable, checkable Boolean variables.
  - `desque` – Double-ended synchronized queues.
  - `utils` – Miscellaneous utility functions.
    - So far just a couple of networking-related functions.
- Application-specific modules:
  - `timestamp` – Coarse- and fine-grained time data types
    - Fine-grained type still needs to be written
  - `appdefs`
    - Application-specific constants required to customize otherwise general-purpose modules.
  - `ports` – Defines constants for certain port numbers used in this app.

`flag` module (file "`flag.py`")

- **Purpose:** Provides a capability for checkable, waitable boolean condition variables that are rather more full-featured and flexible than the Python library's built-in `threading.Condition` objects.
  - For logical event signaling between threads.
- **Uses modules:**
  - `threading` (in Python standard library)
- **Used by modules:**
  - `communicator`, `guiapp`, `worklist`, `tikiterm`, `heart`
- **Exports names:**
  - `Flag` (class)
- **Present status:**
  - Complete; pretty extensively tested; no known bugs.

`Flag` class (`flag.Flag`)

- Public methods:
  - Constructor:
    `flag = Flag([<`*`initiallyUp`*`>,[<`*`lock`*`>]])`
  - Call handler: `flag()`
  - Boolean handler: `flag`
  - Properties: `flag.up`
  - Waiter methods (all take [<*`timeout`*>]):
    - `.wait()`, `.waitUp()`, `.waitDown()`, `.waitRise()`, `.waitFall()`, `.waitChange()`, `.waitWave()`, `.waitTouch()`.
  - Modifier methods:
    - `.setTo(<`*`beUp`*`>)`, `.rise()`, `.fall()`, `.change()`, `.touch()`, `.wave()`.

desque module (file "desque.py")

- **Purpose:** Provides a new thread-safe queue container type ("double-ended synchronized queue") that supports an extra "insert at front" operation (not available in normal queues).
  - Combines features of `deque` and `Queue` classes from standard library.
- **Uses modules:**
  - `threading`, `time`, `queue` (all from standard library)
- **Used by modules:**
  - `worklist`
- **Exports names:**
  - `Desque` (container class), `Empty`/`Full` (exception classes)
- **Present status:**
  - Complete; well tested; no known bugs.

`Desque` class (`desque.Desque`)

- Public methods:
  - Constructor:
    - `desque = Desque([<maxsize>])`
  - Other methods:
    - `.flush()`
    - `.put(<item>, [<block>, [<timeout>, [<front>]]])`
    - `.put_nowait(<item>, [<front>])`
    - `.putfront(<item>, [<block>, [<timeout>]])`
    - `.putfront_nowait(<item>, [<timeout>])`
  - Selected methods inherited from `queue.Queue`:
    - `.get([<block>[, <timeout>]])`
    - `.get_nowait()`

`utils` module (file "`utils.py`")

- **Purpose:** To provide miscellaneous general-purpose low-level utility functions.
  - So far only includes a couple of networking-related functions, but others can be added as needed.
- **Uses modules:**
  - `socket` (from standard library)
- **Used by modules:**
  - `COSMICi_server`
- **Exports names:**
  - `get_hostname()`, `get_my_ip()` (functions)
- **Present status:**
  - Complete; well tested; no known bugs.

`timestamp` module (file "`timestamp.py`")

- **Purpose:**
  - Define abstract data types for various kinds of time-stamp objects (only one defined so far).
- **Uses modules:**
  - `time` (in Python standard library)
- **Used by modules:**
  - `bridge`, `commands`, `communicator`, `COSMICi_server`, `model`
- **Exports names:**
  - `CoarseTimeStamp` (class)
- **Present status:**
  - `CoarseTimeStamp` works fine for general purposes, but has only 1 ms precision, and is only assumed to be accurate within ± ~10 ms.
    - Acc. of COSMICI PC sys. clk. given NTP sync w. NIST server in GA.
  - One or more additional classes for representing more precise & accurate types of timestamps still need to be defined
    - For time-tagging particle detection events (~100 ns acc., ~1 ns prec.).

`CoarseTimeStamp` class (`timestamp.CoarseTimeStamp`)

- Public methods:
  - Constructor:
    - cts = CoarseTimeStamp(*`<floating_time>`*)
      - The *`<floating_time>`* argument is the number of floating-point seconds since the epoch, as returned by `time.time()`.
  - String converter:
    - str(cts)
      - Displays the time (incl. ms) as a string in a human-readable format.
- Public attributes:
  - `.fsecs` – Seconds since epoch, as float
  - `.secs` – Integer seconds since epoch (floor of fsecs)
  - `.msecs` – Milliseconds since the start of the second
    - rounded to nearest integer

`appdefs` module (file "`appdefs.py`")

- **Purpose:**
  - Define the application-specific values of certain constants required to customize otherwise general-purpose modules for use in the present application.
- **Uses modules:**
  - none
- **Used by modules:**
  - `communicator`, `logmaster`
- **Exports names:**
  - `systemName`, `appName` (constant strings)
- **Present status:**
  - This is an extremely simple module. It is definitely bug-free.
  - Other definitions can be added to this module as needed.

`ports` module

- **Purpose:**
  - Define IP port numbers used by this application.
- **Uses modules:**
  - none
- **Used by modules:**
  - `mainserver`, `model`
    - These are the modules responsible for actually setting up listeners on specific ports.
- **Exports names:**
  - `COSMO_PORT`, `LASER_PORT`, `MESON_PORT`
- **Present status:**
  - Complete; extremely simple; no possible bugs.

Level 2 Modules

- These depend only on level 1 modules (and on the standard library).
  - So far we only have two of these (but 1's a doozy).
- General-Purpose Modules:
  - `logmaster`
    - A sophisticated logging facility for multithreaded applications.
- Application-Specific Modules:
  - `sitedefs`
    - Determines the values of site-specific constants, such as the server's IP address.

`logmaster` module

- **Purpose:** Provide a logging facility for multithreaded applications serving multi-component systems.
  - Mostly a general-purpose facility, but customized a bit for the COSMICi application due to the UWSERR log level specifically for tracking low-level errors in EZURiO wireless scripts.
- **Uses modules:**
  - System library modules:
    - `os`, `sys`, `io`, `logging`, `threading`, `traceback`
  - Custom modules:
    - `appdefs` – Defines application-specific constants.
- **Used by modules:**
  - `bridge`, `commands`, `communicator`, `COSMICi_server`, `guiapp`, `heart`, `mainserver`, `model`, `tikiterm`
    - i.e., most of the higher-level modules
- **Present status:** Complete; well tested; no known bugs.

Names exported from `logmaster`

- Global constants & variables:
  - `NORMAL_LEVEL`, `LOG_FILENAME`, `LOG_FORMATSTR`, `CONS_WARN`, `CONS_INFO`, `CONS_DEBUG`, `LOG_INFO`, `LOG_DEBUG`, `systemName`, `sysName`, `appName`
- Global objects:
  - `theLoggingContext`, `mainLogger`, `sysLogger`, `appLogger`, `logFormatter`
- Exception classes:
  - `LoggedException`, `InfoException`, `ExitException`, `WarningException`, `ErrorException`, `CriticalException`, `FatalException`
- Ordinary classes:
  - `LoggingContext`, `ThreadActor`, `AbnormalFilter`, `NormalLogger`, `NormalLoggerAdapter`
- Functions:
  - `initLogMaster()`, `configLogMaster()`, `normal()`, `debug()`, `info()`, `error()`, `warning()`, `warn()`, `error()`, `exception()`, `critical()`, `lvlname_to_loglevel()`, `byname()`, `getLogger()`, `testLogging()`, `updateStderr()`, `setThreadRole()`, `setComponent()`

`sitedefs` module

- **Purpose:**
  - Define values of site-specific constants.
    - In particular, so far just the server's IP address.
- **Uses modules:**
  - Custom modules:
    - `utils`
- **Used by modules:**
  - `mainserver`, `bridge`
    - These are the modules that set up our IP listeners.
- **Exports names:**
  - MY_IP (string constant) – Server's IP address.
- **Present status:**
  - Complete; tested; no known bugs.

Level 3 Modules

- These only use modules at level 2 and below.
- General-purpose modules:
  - `heart`
    - Entity generating periodic heartbeat events.
  - `worklist`
    - Facility for passing work items between threads.

`heart` module

- **Purpose:**
  - Provides a facility that generates logged heartbeat events; this can be useful to verify that the server process is still running and has not crashed.
- **Uses modules:**
  - Standard library modules: `time`, `threading`
  - Custom modules: `flag`, `logmaster`
- **Used by modules:**
  - `COSMICi_server`
- **Exports names:**
  - Heart (thread class)
  - MINS_PER_BEAT (numeric constant)
- **Present status:**
  - Complete; tested; no known bugs.

`worklist` module

- **Purpose:**
  - Provides a facility that allows multiple threads to send work items to other, “worker” threads for execution in a serialized order.
- Uses modules:
  - Standard library modules:
    - `sys`, `traceback`, `threading`, `numbers`
  - Custom modules:
    - `flag`, `desque`, `logmaster`
- Used by modules:
  - `commands`, `communicator`, `COSMICi_server`, `guiapp`, `tikiterm`
- Present status:
  - Complete for present purposes; well tested; no known bugs.

Names exported from `worklist`

- Exception classes:
  - Inherited from `Desque`: `Empty`, `Full`
  - Newly defined by `worklist`:
    - `WorkItemException`, `NotOwner`, `AlreadyStarted`, `WorkAborted`, `EarlyCompletion`, `ExitingByRequest`, `WorkerExiting`, `NullCallable`, `WorklistClosed`, `WorklistClosedForever`, `WorkerException`
- Ordinary classes:
  - `WorkItem`, `Worklist`, `Worker`, `RPCWorker`
- Functions:
  - `HireThread()`, `bind()`

`Worker` class

- Base classes:
  - `ThreadActor` (defined in logmaster module)
- Key public instance methods:
  (These are all external methods: Call them from *other* threads)
  - Constructor:
    - `w = Worker(`[<*worklist*>, [<*target*>, [<*start*>, [<*onexit*>, [<*role*>, [<*component*>]]]]]]`)`
  - Call handler:
    - `w(`<*task*>, [<*block*>, [<*timeout*>, [<*front*>, [<*override*>]]]]`)`
  - Other methods:
    - `r = w.getResult(`<*task*>, [<*block*>, [<*timeout*>, [<*front*>]]]`)`
    - `w.stop()`

Level 4 Modules

- Use only modules at level 3 and below
- General-purpose modules:
  - `guiapp`
    - Framework for building multithreaded TkInter apps.
  - `communicator`
    - Flexible, powerful facility for multithreaded TCP servers, including line-based servers.

`guiapp` module

- Purpose:
  - Provide a facility for writing multithreaded GUI-based applications using the TkInter library.
- Uses modules:
  - Standard library modules:
    - `sys`, `traceback`, `_thread`, `threading`, `tkinter`
  - Custom modules:
    - `flag`, `worklist`, `logmaster`
- Used by modules:
  - `COSMICi_server`, `cosmogui`, `tikiterm`
- Present status:
  - Complete, tested, and fairly stable, but the app experiences occasional GUI-related errors, which suggests that this module might conceivably still have some bugs.

Names exported by `guiapp`

- Global objects:
  - `guibot` – Worker thread to execute all GUI ops.
  - `theMainWin` – Top-level window of application.
- Exception classes:
  - `OnlyOneMainWin`, `NoMainWindow`, `WrongThread`
- Module functions:
  - `MainWin()`, `TopWin()`, `guigo()`, `ismain()`, `ambot()`, `shutdown()`, `initguiapp()`

`communicator` module

- **Purpose:**
  - Provide a flexible multithreaded server framework for two-way TCP communication.
- **Uses modules:**
  - Standard library modules:
    - `io`, `time`, `socket`, `threading`, `socketserver`
  - Custom modules:
    - `flag`, `timestamp`, `appdefs`, `logmaster`, `worklist`
- **Used by modules:**
  - `bridge`, `commands`, `COSMICi_server`, `mainserver`
- **Present status:**
  - Complete, tested, and fairly stable, but the app experiences occasional network-related errors, which suggests that this module might conceivably still have some bugs.

Names exported by `communicator`

- Classes:
  - `Message`, `BaseMessageHandler`, `Connection`, `BaseConnectionHandler`, `CommRequestHandler`, `Communicator`, `LineConnection`, `StreamLineConnection`, `LineCommReqHandler`, `LineCommunicator`
- Constants:
  - `DIR_IN`, `DIR_OUT` – Message transfer directions.

Level 5 Modules

- Use only modules from level 4 and below.
- General-purpose modules:
  - `tikiterm`
    - Provides a convenient TkInter-based text terminal widget.
- Application-specific modules:
  - `cosmogui`
    - Provides misc. COSMICi-specific GUI elements.

`tikiterm` module

- **Purpose:**
  - Provides a simple text terminal widget based on the TkInter library, suitable for diagnostic purposes & for text-based user interaction with remote entities.
- **Uses modules:**
  - Standard library modules:
    - `sys`, `re`, `time`, `threading`, `tkinter`, `tkinter.scrolledtext`
  - Custom modules:
    - `flag`, `terminal`, `worklist`, `guiapp`, `logmaster`
- **Used by modules:**
  - `bridge`, `COSMICi_server`, `mainserver`
- **Present status:**
  - Complete, tested, and fairly stable, but the app experiences occasional GUI-related errors, which suggests that this module might conceivably still have some bugs.

Names exported from `tikiterm`

- String constants (color identifiers):
  - `Black`, `Red`, `Green`, `Blue`, `Cyan`, `Magenta`, `Yellow`, `White`, `DarkRed`, `DarkGreen`, `DarkBlue`, `DarkCyan`, `DarkMagenta`, `DarkYellow`, `LightGray`, `DarkGray`, `Pink`, `Purple`
- Classes:
  - `TikiTermTextStyle`, `TikiTerm`
- Module functions:
  - `style()`

`TikiTerm` class

- Nested classes: `In`, `Out` (File-like interfaces)
- Constructor:
  - `tt = TikiTerm([<master>, [<title>, [<fg>, [<bg>, [<insertbackground>, [<width>, [<height>, [<role> [<in_hook>]]]]]]]])`
- Other key methods:
  - `.put(<text>, [<style>])`
  - `.set_title(<title>)`
  - `.closewin()`

`cosmogui` module

- **Purpose:**
  - Provide miscellaneous GUI elements specific to the COSMICi application.
    - So far, this is just the project logo image
- **Uses modules:**
  - Custom modules:
    - `tkinter`, `guiapp`
- **Used by modules:**
  - `COSMICi_server`
- **Exports names:**
  - `setLogoImage()`, `logoimage`
- **Present status:**
  - Very simple module, no bugs.

Level 6 Modules

- These only use modules at level 5 and below.
  - There is only one level-6 module at present.
- General-purpose modules:
  - `bridge` – Facility for bridging incoming TCP connections to interactive terminal widgets.

`bridge` module

- **Purpose of module:**
  - Provides a generally useful facility that bridges between incoming TCP connections and interactive terminal windows that pop up.
    - We use it to monitor AUXIO and UART streams from nodes; we can also send back user commands.
- **Uses modules:**
  - Standard library modules: time
  - Custom modules:
    - logmaster, communicator, tikiterm, timestamp, sitedefs
- **Used by modules:** model
- **Exports names:**
  - BridgeMsgHandler, BridgeConnHandler, BrdgSrvReqHandler, BridgeServer
- **Present status:**
  - Complete; tested; no known bugs; but app experiences occasional errors, so this module might conceivably have some bugs.

Level 7 Modules

- These only use modules at level 6 and below.
  - There is only one level-7 module at present.
- Application-specific modules:
  - `model` – Object-oriented proxy for the COSMICi sensor network and its various sub-components.

`model` module

- **Purpose of module:**
  - Provide an object-oriented proxy for the various systems and subsystems comprising the local sensor net.
    - For tracking the state of remote components & sending them messages (querying/informing/commanding them).
- **Uses modules:**
  - Standard library modules: `threading`
  - Custom modules: `timestamp`, `ports`, `logmaster`, `bridge`
- **Used by modules:**
  - `COSMICi_server`
- **Exports names:**
  - `SensorNet`, `SensorNode` (more to be added)
- **Status:**
  - OK so far, but much application functionality needs to be added.

Level 8 Modules

- The relationships among these modules are presently somewhat complex.
  - They use each other in a cyclic fashion.
  - `COSMICi_server` is the main module, but several of the others also reference it recursively!
  - Design really needs to be refactored to clean up the module hierarchy.
- Application-specific modules:
  - `COSMICi_server` (top-level module)
  - `commands` (command-line interface to server)
  - `mainserver` (handles initial TCP connections)

`mainserver` module

- **Purpose:**
  - Provides main TCP server functionality for accepting connections from new nodes in local sensor net, and processing high-level messages from them.
    - Mainly, status updates & log messages to the server from the Wi-Fi script.
- **Uses modules:**
  - Standard library modules: `time`
  - Custom modules:
    - Imports `logmaster`, `communicator`, `ports`, `sitedefs`, `tikiterm`,
    - `COSMICi_server`, `command` are used but not imported
- **Used by modules:** `COSMICi_server`
- **Exports names:**
  - `Acknowledge_MsgHndlr`, `TermDisp_MsgHndlr`, `Command_MsgHndlr`, `MainConnHandler`, `MainSrvReqHandler`, `MainServer`
- **Present status:**
  - Complete; no known bugs; but app has occasional IP/GUI errors, so it's possible there's a bug somewhere in this module.

`commands` module

- **Purpose of module:**
  - Provide the COSMICi server's primary central command processing facility.
    - All command messages to the server from other system components are dispatched through this facility.
- **Uses modules:**
  - Custom modules:
    - Imports `logmaster`, `worklist`, `communicator`, `threading`, `timestamp`
    - Also uses `COSMICi_server`, `model` (but without importing them).
- **Used by modules:**
  - `COSMICi_server`, `mainserver`
- **Exports names:**
  - `EmptyCommand`, `Command`, `CommandHandler`
- **Status:** No known bugs. Many commands need to be added.

`COSMICi_server` module

- **Purpose of module:**
  - This is the top-level module of the entire COSMICi central server application.
- **Uses modules:**
  - Standard library modules:
    - `time`, `sys`, `threading`
  - Custom modules:
    - `utils`, `logmaster`, `guiapp`, `tikiterm`, `cosmogui`, `timestamp`, `commands`, `heart`, `worklist`, `communicator`, `mainserver`, `model`
- **Used by modules:**
  - `mainserver`, `commands`
- **Exports names:**
  - `console`, `CosmicIServer`, `cosmicIServer`, `main()`
- **Present status of application:**
  - Terminal I/O done. However, application is not completely bug-free, & much general functionality still to be added.